SZENT ISTVÁN EGYETEM

BENKŐ JÁNOS

Logisztikai folyamatok szimulációja

Dr. Benkő János
egyetemi tanár

Logisztikai folyamatok szimulációja

Gödöllő
2015.

A SZIE Gazdaság- és Társadalomtudományi Karán javasolt tankönyv

Lektorálta:
Dr. Lipovszki György
egyetemi docensr
Szarvas Lajos
gépészmérnök

Kiadja a Logisztika Oktatásért és Kutatásért Alapítvány
2103 Gödöllő, Páter K. u. 1.

Nyomás: Szent István Egyetemi Kiadó
2100 Gödöllő, Páter Károly u. 1.

Tartalom

1. Bevezetés a logisztikai rendszerek szimulációjába

A szimuláció általában számítógépre adaptált, a valós rendszerek viselkedését utánzó módszerek és alkalmazások széles gyűjteménye. Az alkalmazások megtalálhatók a tudomány és a termelés majdnem minden ágában. Például a társadalmi csoportok viselkedésének szimulációja konfliktus helyzetekben, a biológiai rendszerek szimulációja, a termelési rendszerek szimulációja, a pilóták kiképzésére használt repülőgép szimulátor, a személygépkocsik szimulációja szélcsatornában mind közismert eszközök és eljárások. Napjainkban a szimuláció a számítógépek és a szoftverek fejlődésének köszönhetően sokkal közismertebb és erőteljesebb eszköz, mint bármikor korábban.

A szimuláció más analitikai módszerekhez hasonlóan magában foglalja a vizsgált rendszert és annak modelljét, vagyis a szimuláció rendszermodellezéssel dolgozik, ahol a rendszer lehet létező vagy megvalósítandó eszköz vagy folyamat, például egy ipari üzem gépekkel, emberekkel, szállítóeszközökkel és tároló helyekkel. A rendszerek tanulmányozásának célja a teljesítmények mérése, a működés javítása, illetve új rendszerek esetén azok tervezése. A rendszer irányítói a naponta ismétlődő műveletekről olyan ismeretekkel szeretnének rendelkezni, mint például: mi a teendő akkor, ha egy üzemben egy fontos gép meghibásodik.

A szimulációs technikák fejlődésének és alkalmazásának igazán nagy lendületet a nagysebességű számítógépek megjelenése adott. Ennek köszönhetően a szimuláció a logisztikai operációk tervezésének is fontos kísérleti eszköze lett. A közelmúltban számos új szoftver jelent meg a piacon. A szoftverek között a csúcstechnikát azok a grafikus programozású rendszerek jelentik, amelyek egyetlen programsor leírása nélkül is képesek összetett rendszerek állapotváltozóinak időbeni alakulását és statisztikai jellemzőit meghatározni.

Eddig a hangsúlyt a logisztikai folyamatok és operációk matematikai modellezésére és megoldására helyeztük. Ennek a megközelítésnek az a nagy előnye, hogy felfedi a rendszerek struktúrájának a lényegét, ezáltal bepillantást nyerünk a rendszeren belüli ok-okozati viszonyokba. Ezért, ha olyan matematikai modellt tudunk felállítani, amely a problémát adekvátan leírja és ráadásul analitikusan vagy numerikusan kezelhető, akkor a matematikai modellt a szimulációval szemben előnyben kell részesíteni. Sokszor találkozunk azonban olyan problémákkal, melyek matematikai szabályokkal (formulákkal) nem írhatók le, vagy azok megoldására nem találunk sem analitikus, sem heurisztikus módszereket. Ez leggyakrabban olyan sztochasztikus folyamatoknál jelentkezik, amelyeket nagyszámú valószínűségi változó jellemez, esetleg különböző típusú eloszlásokkal. Ilyenkor a modellezés egyetlen járható útja a szimuláció.

1.1. Mit jelent a szimuláció?

A szimuláció a vizsgált rendszer részeinek viselkedésére vonatkozó tényeken és feltételezéseken alapuló imitációja, azzal a céllal, hogy ismereteket szerezzünk a rendszer egészének viselkedéséről. A szimulációs modell tehát a rendszer működését annak közvetlen leírása helyett, a rendszer egyes elemeinek egyedi eseményein keresztül írja le. Pontosabban, a rendszert olyan összetevőkre vagy elemekre bontjuk, amelyek viselkedése a rendszer minden állapota és lehetséges bemenete esetén megjósolható, legalábbis valószínűségi értelemben.

Az elkészült modellt úgy működtetjük, hogy véletlenszám-generátorral eseményeket szimulálunk az elemek viselkedésének megfelelő valószínűségi eloszlás szerint. Az eredmény a rendszer tulajdonságait és időbeli működését tükröző szimuláció. Megismételve ezt az eljárást a rendszer különböző konfigurációira és működési politikáira, az eredmények összevetése után kiválaszthatjuk a legígéretesebb változatot vagy változatokat. A szimuláció tehát abban segít,

hogy adott technikai és működési feltételek mellett megtaláljuk a legjobb megoldásokat, illetve elkerüljük a gazdaságtalan működést.

Azt is mondhatjuk, a szimuláció lényegében nem más, mint a rendszer modelljén végzett mintavevő kísérletek sorozata. Az adatok és a számítások nagy tömege miatt a szimulációs modellt általában számítógépre adaptáljuk. Segítségével az adott termelési folyamat gyenge pontjait, rosszul működő elemeit már a tervezés fázisában felfedezhetjük, és még a megvalósítás előtt kiszűrhetjük ezeket a problémákat.

1.2. A modell felállítása és működése

Tekintsük az általánosan ismert *M*/*M*/1 sorbanállási modellt (*Poisson*-féle bemenet, exponenciális kiszolgálási idők, egy kiszolgáló). Bár ez a modell analitikusan is megoldható, tanulságos lesz szimulációval is tanulmányozni. A rendszer működése a következő: az érkező ügyfél beáll a sorba, kiszolgálják, majd távozik. A szimulációs modell feladata ebben az esetben, összhangba hozni az ügyfél érkezését és kiszolgálását. A megoldásra a számítógépen két módszer áll rendelkezésünkre: az ún. **rögzített időnövekmény** és **az esemény növekmény** módszere.

A **rögzített időnövekmény módszerében** a következő kétlépéses eljárást alkalmazzuk. Feltételezve, hogy a folyamat kezdetén a rendszer adott időpontban valamely állapotban van, először növeljük az időt egy adott rögzített mennyiséggel, mondjuk 1-el. (Az időt számláló "rendszeróra" értékéhez adjunk 1-et.) Ezután meghatározzuk, milyen esemény történt az elmúlt rövid időtartam alatt, és mi a rendszer pillanatnyi állapota. Ezt a két lépést annyiszor ismételjük, ahányszor csak akarjuk, illetve ahányszor csak szükséges.

Az említett *M*/*M*/1 sorbanállási problémában minden eltelt egységnyi idő alatt csak kétféle esemény történhet: egy vagy több érkezés és egy vagy több kiszolgálás befejezése. Sőt, ha az időegységet elég rövidnek választjuk, akkor az egynél több érkezésnek és az egynél több kiszolgálás befejezésének valószínűsége is elenyésző. Ezért a két lehetséges esemény egy ilyen időintervallumban: egy ügyfél érkezése és egy ügyfél kiszolgálásának a befejezése.

Mindkét eseménynek ismerjük a valószínűségét. (Az érkezési időközök és a kiszolgálási idők egyaránt független exponenciális eloszlású valószínűségi változók.) Ezért a számítógépnek csak véletlen számokat kell generálnia, amely azt szimulálja, hogy az esemény bekövetkezett-e vagy sem.

Tegyük fel, hogy egy ügyfél érkezésének a valószínűsége az adott időegység alatt 0,007, akkor a számítógépnek a lehetséges 1000 érték (000, 001,…,999) közül kell véletlenszerűen egyet generálnia. Az ügyfél érkezés eseményéhez hét számot rendelünk az 1000 érték közül (legyen ez a hét érték mondjuk 000, 001,...,006), a többi érték annak fog megfelelni, hogy az esemény nem következik be. A generált véletlen szám egy tényleges kimenetet fog reprezentálni.

A számítógép, ismerve a kiszolgálási idők eloszlását, ugyanilyen módszerrel szimulálja a kiszolgálás befejezésének az eseményét (feltéve, hogy az adott időpontban egy ügyfél tartózkodik a kiszolgáló csatornában). Ha nincs ügyfél a rendszerben, akkor természetesen a számítógép automatikusan azt mondja, hogy az adott idő alatt nem fejeztek be kiszolgálást. Hasonlóképpen egy számláló jegyezheti a sorban éppen várakozók számát.

Tehát minden „időlépés” után a számítógép feljegyzi az imént említett adatokat, és minden egyéb információt, ami a rendszer működése szempontjából fontos. Például feljegyezheti a rendszerben lévő ügyfelek számát, az éppen kiszolgált ügyfél várakozási idejét. Ha elég ezeknek a valószínűségi változóknak a középértékét megbecsülni (a teljes eloszlás helyett), akkor a számítógép egyszerűen csak hozzáadja az esedékes értéket a korábbiak összegéhez. A szi-

mulációs mintaátlagot úgy kapjuk meg, hogy a teljes összeget elosztjuk a minta méretével, nevezetesen most az eltelt időegységek számával, illetve az összes ügyfelek számával.

1.1. táblázat

Sorszám	Véletlen számok		Szimulált folyamat				
	az érkezési folyamat	a kiszolgálási folyamat	Érkezési időköz	Kiszolgálási időtartam	Érkezési időpont	Kiszolgálás befej. időpontja	Várakozási idő
	szimulációjához		$t_{érk}$	t_{kisk}	$T_{érk}$	T_{kisz}	t_w
1	0,90	0,63	16,0	6,9	16	22,9	0,0
2	0,39	0,63	3,5	6,9	19,5	29,8	3,4
3	0,45	0,64	4,1	7,0	23,6	36,8	6,2

Az **eseménynövekmény módszerében** a rendszeróra által mutatott idő az előző eljárástól eltérően rögzített helyett változó időközökkel növekszik. A szimuláció alatt az idő addig halad előre, amíg a következő esemény be nem következik. A számítógép ezt úgy valósítja meg, hogy kiszámítja, mikorra várhatók a következő események, és az első ilyen eseményre ugrik, majd feljegyzi a változást. A lépéseket annyiszor ismételjük, ahányszor csak szükséges.

Az *M*/*M*/1 sorbanállási példánkban a számítógépnek két jövőbeli eseményt kell szimulálnia, nevezetesen a következő érkezést ($T_{érk}$) és a következő kiszolgálás befejezését (T_{kisz}), ha éppen van ügyfél a kiszolgálási csatornában. Ezeket az időpontokat a számítógép program az érkezési időközök ($t_{érk}$) és a kiszolgálási idők (t_{kisz}) valószínűségi eloszlásából vett véletlen megfigyelésekből származtatja. Mint korábban említettük, a megfigyelést a számítógép véletlen szám generálásával végzi. Tehát minden pillanatban, amikor egy érkezés vagy egy kiszolgálás befejeződött a számítógép meghatározza, mennyi idő múlva várhatók a következő események, és hozzáadja ezt az időt a rendszeróra akkori idejéhez. Az elmondottak a *1.1. táblázatban* is követhetők.

A szimulációs vizsgálatok első lépése az, hogy elkészítsük a vizsgálandó rendszer modelljét. Ez a lépés feltételezi a rendszer valóságos működésének és a vizsgálat céljának alaposan ismeretét. Ezek birtokában elkészíthetjük a valódi rendszer logikai diagramját. A diagramban a rendszert összetevőkre (elemekre) bontjuk, amelyeket a diagram vonalai kötnek össze egymással. Az összetevők maguk is további elemekre bonthatók. (A felbontás mélységéről csak a feladat alapos átgondolása után és a célok ismeretében dönthetünk.) Az elemekre működési szabályokat állapítunk meg. Ezek a működési szabályok megjósolják a megfelelő elem által generált eseményeket, esetleg valószínűségi eloszlás alakjában. Ezután nem kell mást tennünk, mint az összetevőket a köztük lévő viszonyok alapján összekapcsolni. Lényegében a szimulációs modellépítés két fázisból áll: először egyszerűbb, többé-kevésbé önálló részeket (ezek akár alrendszerek is lehetnek, mint például egy egyszerű sorbanállási rendszer) fogalmazzunk meg, majd ezeket a részeket összerakjuk a természetes rendjük szerint.

Miután meghatároztuk az elemeket, a működési szabályokat és a logikai kapcsolatokat, a modellt részletesen ellenőrizzük. Az ellenőrzést úgy valósíthatjuk meg, hogy a szimulációt nagy vonalakban számítógéppel végigkövetjük, annak megállapítására, hogy a bemenetek a megfelelő forrásból származnak-e, és a kimenetek elfogadhatók-e a következő összetevő számára. Ezen kívül, a modell egyedi elemeinek belső működését is ellenőrizni kell.

Hangsúlyoznunk kell, hogy a szimulációs modellnek, mint akármilyen más modellnek, nem kell a valóságos rendszert tökéletesen tükröznie. A tapasztalat szerint a szimulációs modellek többségének az a hibája, hogy túlságosan részletekbe menő. Ilyenkor a modell könnyen jelentéktelen részletek tömegévé válhat, és kevés információ megszerzése is igen sok programo-

zást és gépidőt igényelhet. Az sem szorul magyarázatra, hogy ha nem tudjuk megragadni a rendszer lényegét, akkor az eredmények is nehezebben értelmezhetőek.

A valószínűségi eloszlásokkal kapcsolatban felmerülhet az a kérdés, hogy a múltbeli adatok gyakoriság-eloszlását használjuk-e, vagy keressünk elméleti eloszlást, amely a legjobban illeszkedik az adatainkhoz. Ez utóbbi eljárás jobbnak tűnik, mert elkerüli, hogy reprodukáljuk a múlt egyes időszakaszainak a sajátosságait.

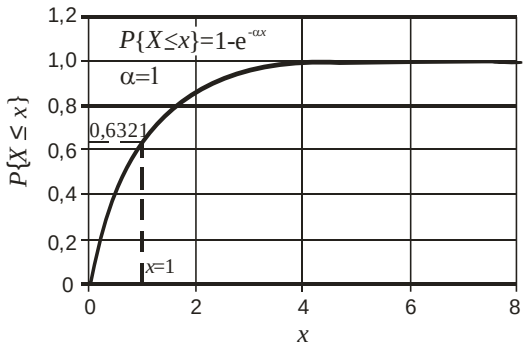
1.3. Véletlen számok és a valószínűségi eloszlásokból vett véletlen megfigyelések generálása

A véletlen számok adott tartományon belül képzett olyan számok, amelyek meghatározott gyakorisággal, de teljesen véletlen sorrendben következhetnek egymás után. A számok előfordulási gyakorisága bármilyen lehetséges eloszlást követhet. Számunkra a legfontosabbak az egyenletes eloszlású véletlen számok, mert ezekből az összes többi eloszlás szerinti véletlen szám előállítható.

Egyenletes eloszlású véletlen számokat különböző módszerekkel lehet képezni (pl. dobókockával 1-6, rulettel 0-36 közötti egyenletes eloszlású véletlen számok sorozata állítható elő). A számítógépekhez fejlesztett programozási nyelvek többsége rendelkezik véletlen számokat generáló függvénnyel, amelyekkel folyamatosan tetszőleges számú véletlen szám állítható elő.

Hogyan lehet valamely valószínűségi eloszlásból vett véletlen megfigyeléseket generálni, ha adott az egyenletes eloszlású véletlen számoknak egy sorozata? Egyszerű diszkrét eloszlások esetében semmi mást nem kell tenni, mint egymáshoz rendelni a véletlen számok lehetséges és a valószínűségi eloszlás értékeit.

Bonyolultabb eloszlásokra is lényegében hasonló a válasz, jóllehet a konkrét eljárás valamivel körülményesebb. Az elsőként megválasztjuk az $F(x)=P\{X\leq x\}$ eloszlásfüggvényt, ahol X a szóban forgó valószínűségi változó. Az eloszlásfüggvényt reprezentálhatja a függvényre felírt egyenlet, a függvény grafikus ábrája, vagy olyan táblázat, amely megadja az x értékeit az $F(x)$ 0 és 1 közötti egyenlő közű értékeire. A második lépésben kellően sokjegyű véletlen tizedes számot generálunk 0 és l között (esetleg megfelelően sok nullával az elején), és eléje helyezzük a tizedesvesszőt. Az utolsó lépésben $P\{X\leq x\}$-et egyenlővé tesszük a kapott véletlen számmal, és az így kapott egyenletet megoldjuk x-re. Az x-nek ez az értéke a valószínűségi eloszlásból vett véletlen megfigyelés [38]. Ezt az eljárást a *1.1. ábra* szemlélteti, ahol egy exponenciális eloszlás eloszlásfüggvényt grafikusan ábrázoltunk, és a véletlen szám történetesen 0,6321.



1.1. ábra. Véletlen megfigyelés generálása az eloszlásfüggvényből.

Ha az adott valószínűségi eloszlás folytonos, akkor az előbb vázolt eljárás a folytonos eloszlást olyan diszkréttel közelíti, amelynek szabálytalanul elhelyezkedő pontjai azonos valószí-

nűségűek. Ebből semmi baj sem származik, mert a közelítés tetszőleges pontosságú lehet, ha elég sokjegyű véletlen számot használunk.

Számítógépen tárolt táblázatba foglalni az eloszlásfüggvény értékeit lényegében egyenértékű a "kézi" grafikus eljárással. A táblázatból kereső módszernek az a hátránya, hogy a táblázat összeállítása munkaigényes, továbbá a korlátozott tárolókapacitás miatt pontatlanságok is előfordulhatnak. A másik említett módszer az egyenlet felírása az eloszlásfüggvényre, majd annak a pontnak a megkeresése, amelyre a függvény egyenlő a kapott véletlen számmal. Ez néhány fontos valószínűségi eloszlásnál egyszerű egyenlet megoldáshoz vezet. Más eloszlások esetében általában időigényes numerikus módszerek szükségesek ahhoz, hogy a megoldást megtaláljuk. Szerencsére fontosabb valószínűségi eloszlásokhoz a véletlen megfigyelések generálására hatékony technikákat fejlesztettek ki.

Tekintsük például a *1.1. ábrán* is látható exponenciális eloszlást, amelynek eloszlásfüggvénye:

$$P\{X \le x\} = 1 - e^{-\alpha x},$$

ahol $x \ge 0$ és $1/\alpha$ az eloszlás várhatóértéke. Az egyenlet megoldási módszerben ezt a függvényt egyenlővé tesszük egy 0 és 1 közé eső véletlen tizedes számmal (amelyet r jelöl). Így az

$$1 - e^{-\alpha x} = r,$$

azaz

$$e^{-\alpha x} = 1 - r$$

Vegyük mindkét oldal természetes logaritmusát:

$$\ln(e^{-\alpha x}) = \ln(1 - r),$$

amelyből

$$x = \frac{\ln(1-r)}{-\alpha}$$

a kívánt véletlen megfigyelés az exponenciális eloszlásból.

Megemlítjük, hogy más, bonyolultabb, de gyorsabb módszereket is kifejlesztettek az exponenciális eloszlás kezelésére. Minthogy a 0 és 1 az közé eső véletlen szám nem más, mint a 0 és az 1 közötti megfelelő beosztású diszkrét egyenletes eloszlásból vett véletlen megfigyelés az $(1-r)$ is véletlen tizedes szám. Ezért, hogy a kivonást megtakarítsuk, a gyakorlatban rögtön $(1-r)$-et generáljuk, ami ugyanaz, mint r generálása.

Az exponenciális eloszlásra vonatkozó eredmény természetesen kiterjeszthető az *Erlang-fé1e* (gamma-) eloszlásra. A k darab független, exponenciális eloszlású valószínűségi változó összege, ha mindegyiknek a várhatóértéke $1/k\alpha$, *Erlang*-eloszlású k alakparaméterrel és α várhatóértékkel. Ezért, ha adott k véletlen tizedes szám sorozata 0 és 1 között, mondjuk r_1, r_2, ...,r_k, akkor az *Erlang*-eloszlásból vett véletlen megfigyelés

$$x = \sum_{i=1}^{k} \frac{\ln(1-r_i)}{-k\alpha},$$

ami úgy is felírható, hogy

$$x = -\frac{1}{k\alpha} \ln\left\{\prod_{i-1}^{k} (1-r)\right\}.$$

Itt is megtakaríthatjuk a kivonásokat az előzőhöz hasonlóan, ha közvetlenül $(1-r_i)$-t generáljuk r_i helyett.

A normális eloszlásból vett véletlen megfigyelések generálására különösen egyszerű módszert ad a centrális határeloszlás tétele. Mivel egy véletlen tizedes szám lényegében egyenletes eloszlású 0 és 1 között, a középértéke 1/2, szórása $1/\sqrt{12}$. Ezért az említett tétel szerint n véletlen tizedes szám összege közelítőleg normális eloszlású az $n/2$ középértékkel és a $\sqrt{12}$ szórással. Ha tehát $r_1, r_2, ...,r_n$ véletlen tizedes számok, akkor

$$x = \frac{\sigma}{\sqrt{n/12}} \sum_{i=1}^{n} r_i + \left(\mu - \frac{n}{2} \frac{\sigma}{\sqrt{n/12}} \right)$$

véletlen megfigyelés egy közelítőleg normális eloszlásból, amelynek várhatóértéke μ, szórása pedig σ. Ez a közelítés nagyon pontos, még kis n-ek esetén is, kivéve az eloszlás legszélét. Gyakran ezért 5 és 10 közötti n értéket használnak. Az n=12 nagyon kényelmes érték, mert kiküszöböli a négyzetgyököt az előbbi kifejezésből.

A χ^2 (*chi-négyzet*)-eloszlás *standard* normális eloszlások összege. Ha tehát az $y_1, y_2, \ldots, y_n$ 0 várhatóértékű és 1 szórású normális eloszlásból vett véletlen megfigyelések, amelyeket az előbb leírt módon kaptunk meg, akkor az

$$x = \sum_{i=1}^{n} y_i^2$$

egy n szabadságfokú *chi-négyzet*-eloszlásból vett véletlen megfigyelés.

1.4. A szimulációs programok működése

A szimulációs modellek a vizsgált rendszer logikai vagy matematikai modelljei, amelyek számos strukturális és kvantitatív közelítést és feltételezést tartalmaznak a valóságos rendszerhez viszonyítva. A logikai modell rendszerint, mint számítógép program jelenik meg, amely a modell viselkedésére vonatkozó kérdésekre ad válaszokat, feltéve, hogy a modell hű reprezentációja a rendszernek. Az utóbbi esetben remélhetjük, hogy új ismereteket szerezhetünk a rendszer viselkedéséről. Mivel a valós rendszer helyett a számítógép programban változtathatjuk a bemenő paramétereket és a modell viselkedését, egy sor kérdésre lényegesen olcsóbb és gyorsabb válaszokat kapunk. Ugyanakkor a kísérletezés során előforduló hibák nem járnak olyan drámai következményekkel, mint amikor azokat a valóságban követjük el. A számítástechnika fejlődése és a számítási költségek csökkenése sok más területhez hasonlóan kedvezően hat a logikai modellek komputer analízisében rejlő lehetőségek kihasználására.

A komputer szimuláció olyan módszer, amely a valóságos rendszerek működését, tulajdonságait imitáló szoftvert használ a valóságos rendszerek modelljeinek tanulmányozására, numerikus kiértékelésére. Praktikusan a szimuláció a valós létező, vagy megalkotandó rendszer komputerizált modelljének tervezési és alkotási folyamata, azzal a céllal, hogy adott feltételek és inputok esetén lehetőséget teremtsen a rendszer viselkedésének jobb megértésére. A szimuláció nem az egyedüli eszköz egy modell tanulmányozására, mégis a leggyakrabban választott módszer, aminek oka, hogy amikor a rendszer valósághű modelljére van szükség, akkor ennek a szimulációs modellek jobban megfelelnek. A szimulációs modellek ugyanis kevésbé követelnek komplexitást csökkentő megszorításokat. Más módszerek alkalmazása, sokkal több és drasztikusabb, a modell érvényességét csökkentő, egyszerűsítő feltételt igényel.

A számítógépes szimulációt rendszerint nagy szaktudással és sok munkával megvalósított modellépítés és programozás előzi meg. A szimulációs modell programozója általában nem azonos a modell alkotójával, mivel a kódoláshoz magas szintű programozási ismeretekre van szükség. Az egyedi fejlesztés eredménye általában egy olyan program, amely csak néhány feladat típus szimulációs vizsgálatához alkalmazható.

A digitális számítógépek megjelenésével szinte egy időben, az 1950-es és az 1960-as években általános célú programnyelveket, például a **FORTRAN**-t használták a szimulációs programok megírására. Ezek a programnyelvek általában a szimulációt támogató csomagokkal is rendelkeztek. Később jelentek meg a speciális célú szimulációs nyelvek, mint a **GPSS, Simscript, SLAM, SIMAN**, és biztosítottak sokkal jobb, kényelmesebb programozási környezetet a felhasználóknak. E nyelvek rendkívül népszerűek voltak és széles körben nyertek alkalmazást.

A szimulációs programcsomagok tervezésénél a cél (az egyedi fejlesztéstől eltérően), hogy a szimulációs eszköz ne csak egy-egy feladat, hanem a valós problémák széles skálájának megoldására legyen alkalmas. E fejlesztési koncepció eredményei azok a minden elemükben menü-vezérelt programcsomagok, amelyekkel a modellezés könnyen és gyorsan megvalósítható, és amelyeknél fokozott hangsúlyt helyeztek a képi megjelenítésre. Az alkalmazó minden részletre kiterjedő, rajzfilmszerű mozgóképpel kísérheti végig a teljes szimulációs folyamatot. A szimulációs programcsomagok olyan egyéni elemekkel kiegészíthető fejlesztő eszközök, amelyekkel a felhasználó különböző típusú gyártási (termelési) és logisztikai feladatokat elemezhet saját személyi számítógépén.

Az utóbbi években jelentek meg az ún. magasszintű **szimulátorok**, amelyeket elsősorban a könnyű használhatóság, az interaktív grafikus felhasználói felület, a menük és a dialógusok jellemeznek. A szimulátorban a rendelkezésre álló szimulációs modell elemekből választhatunk, azokat összekapcsolhatjuk, paraméterezhetjük és a programot futtatva a rendszerkomponensek mozgását, változását grafikusan megjeleníthetjük (animálhatjuk). Legtöbb szimulátor alkalmazási területe erősen korlátozott (pl. gyártás, kommunikáció), és általában nem rugalmasak. Sokak szerint ezek a szoftvercsomagok túl messzire mentek azzal, hogy feláldozták a rugalmasságot a könnyű használat oltárán.

A felhasználók a hazai szoftver piacon számos, árban és szolgáltatások tekintetében eltérő szimulációs programcsomag vagy szimulátor közül válogathatnak (*Taylor II, Taylor ED, ARENA, Plant Simulation, AnyLogic, Witness*).

2. Valószínűség és statisztika

2.1. Bevezetés

A szimulált rendszereket általában egy vagy több olyan véletlenszerűen változó elem alkotja, amelyek között a kapcsolatok bizonytalanok. Az ilyen, időben véletlenszerűen változó rendszereket **sztochasztikus rendszereknek** nevezzük. A sztochasztikus rendszerekben az elemek változása nem jósolható meg pontosan, ezért a változások leírására (modellezésére) a valószínűség számítás módszereit alkalmazzuk. Természetesen a szimulációs modell kimenetei is valószínűségi változók, ezért ezek is statisztikai eszközökkel interpretálhatók. Némi jártasságot feltételezve a matematikai statisztikában, ebben a fejezetben áttekintjük a modellezéssel és analízissel kapcsolatos legfontosabb statisztikai alapfogalmakat.

2.2. Kísérlet, eseménytér, kimenetek (eredmények)

A **kísérlet** az a jól definiált eljárás vagy folyamat, amelynek kimenetei megfigyelhetők, de azok előre nem ismertek vagy nem jósolhatók meg teljes bizonyossággal. A lehetséges elemi események (**kimenetek**) halmazát **eseménytérnek** nevezzük. Ha az eseménytér véges, vagy megszámlálhatóan végtelen, akkor diszkrétnek, különben folytonosnak nevezzük.

A kimenetek (eredmények) kombinálásával új kimeneteket[1] kaphatunk a következő halmazműveletekkel: unió($\cup$) és metszet($\cap$). Ha a **C** kimenet úgy definiált, mint **A** és **B** kimeneti halmazok uniója, akkor azt **C**=**A**$\cup$**B**–vel jelöljük, és a **C** az **A** és **B** halmazok minden elemét tartalmazza. Ha a **D** kimenet az **A** és **B** metszete, amit **D**=**A**$\cap$**B** -vel jelölünk, akkor a **D** tartalmazza az **A**-ban és **B**-ben is megtalálható közös elemeket.

Az előző fogalmak bemutatásához tekintsük egy olyan bankot, amelynek egy pénztára van. A bankba érkező ügyfelek beállnak pénztár előtti sorba és várakoznak a kiszolgálásukra. Az ügyfelek érkezései között eltelt időtartamok és az ügyfelek kiszolgálásának időtartamai egyaránt változnak. Az első kísérletünk legyen az ügyfelek érkezési időközeinek a megfigyelése. A kísérletünk eseménytere tartalmazza az összes érkezések között megfigyelhető időtartamot. Mivel az ügyfelek érkezése között eltelt idő bármely nem-negatív valós szám lehet az eseménytér **folytonos**. Egy elemi esemény (kimenet) úgy definiálható, mint az eseménytér részhalmaza, ezért például a 8 és 9 perc közötti érkezési időközök előfordulása egy lehetséges kimenetként definiálható.

Második példaként tekintsük az első órában kiszolgált ügyfelek számát. A megfigyelt ügyfelek száma lehet 0, 1, 2, 3, …, vagyis a nem-negatív egészszámok halmaza. Ebben az esetben az eseménytér **diszkrét**. Egy óra alatt 5 ügyfél kiszolgálása egy lehetséges kimenetet jelent.

2.3. A valószínűség

Egy kimenet valószínűsége a kimenet előfordulásainak a mértéke vagy gyakorisága. Pontosabban a valószínűség mértéke egy függvény $P()$, amely a kimeneteket valós számokkal írja le és eleget tesz a következő valószínűségi axiómákat:

1. $0 \le P(E) \le 1$ bármely E kimenetre.

2. $P(S) = 1$, ahol S az eseménytér, vagy biztos kimenet.

[1] Az "esemény" szó általában az eredmények kombinációját jelenti. A szimulációs terminológiában azonban másként definiáljuk, ezért használatától itt tartózkodunk.

3. Ha az $E_1, E_2, E_3, \ldots$ egymást kölcsönösen kizáró kimenetek, akkor a

$$P(E_1 \cup E_2 \cup E_3 \cup \ldots) = P(E_1) + P(E_2) + P(E_3) + \ldots$$

Ezekből az axiómákból és a halmazelmélet szabályaiból vezethetők le a valószínűség- számítás alaptörvényei, bár ezek az axiómák nem elegendőek ahhoz, hogy a kimenetek valószínűséget kiszámítsuk. A valószínűségek számszerű értékei általában ritkán és nehezen érhetők el, azonban létezésüket hasznos posztulátumként feltételezni.

Néhány egyszerűbb esetben kombinatorikai számítással is meghatározhatjuk a pontos valószínűséget. Például így kiszámíthatjuk a "*h*" fej bekövetkezésének valószínűségét "*n*" érme feldobásakor, vagy 3 ász valószínűségét 5 kártyalapban, de a legtöbb esetben a kimenet pontos valószínűségét nem tudjuk kiszámítani. Ilyenkor csak az esemény valószínűségének a közelítő értékét kaphatjuk meg a relatív gyakoriság interpretációjával.

Ha egy kísérletet *n*-szer megismétlünk és az *E* kimenet *k*-szor fordul elő, akkor a *k*/*n* arányos az *E* előfordulásainak a számával. Az *E* valószínűségét interpretálhatjuk a

$$P(E) = \lim_{n \to \infty} \frac{k}{n}$$

határértékkel, feltételezve, hogy a határérték létezik. Kellően nagyszámú *n*-t választva az előfordulások aránya (*k*/*n*) közelíti az *E* valószínűségét. Az így kapott közelítő értékekről belátható, hogy kielégítik a korábban tárgyalt axiómákat. Ennek a megközelítésnek a hátránya, hogy gyakran nem lehetséges vagy nem gazdaságos az összes kísérletet elvégezni.

2.4. Valószínűségi változók és eloszlásaik

Azokat a mennyiségeket, amelyek értéke nem állandó, hanem esetről esetre más és más lehet, de meghatározható, hogy milyen valószínűséggel esik megadott határok közé, **valószínűségi változóknak** nevezzük. A valószínűségi változó az eseménytéren egy függvényt értelmez, amely az eseménytér kimeneteihez egy valós számot rendel, amely alapján a valószínűségi változó pontos matematikai definíciója a következő:

Valószínűségi változónak nevezzük az elemi események halmazán értelmezett minden olyan valós értékű függvényt, amelyre tetszőleges valós x szám esetén az $X(E) \le x$ *egyenlőtlenséghez tartozó E elemi események halmaza matematikai értelemben vett esemény.*

Attól függően, hogy egy valószínűségi változó véges vagy megszámlálhatóan végtelen sok értéket vesz fel, vagy pedig nem megszámlálhatóan végtelen sokat, különbséget teszünk diszkrét valószínűségi változók és folytonos értékeket felvevő valószínűségi változók között. A példánkban az érkezési időköz folytonos, az egy óra alatt kiszolgált ügyfelek száma pedig diszkrét valószínűségi változó.

Ha egy X valószínűségi változó csak véges, vagy megszámlálhatóan végtelen értéket vehet fel, akkor ***diszkrét valószínűségi változóknak****, az eloszlás függvényét pedig diszkrét valószínűségeloszlásnak (röviden diszkrét eloszlásnak) nevezzük.*

A valószínűségi eloszlás egyik lehetséges ábrázolási módja a kumulált eloszlásfüggvény vagy röviden eloszlásfüggvény *F*(*x*). Ha az *X* lehetséges értékei rendre x_1, x_2, … számok, akkor az *X* eloszlásfüggvénye

$$F(x) = P\{X \le x\} = \sum_{x_i \le x} P(x = x_i),$$

ahol az összegzés kiterjed minden olyan *i* indexre, amelyhez tartozó $x_i \le x$.

Az $F(x)$ függvény annak a valószínűsége, hogy X valószínűségi változó x-nél kisebb vagy azzal egyenlő értéket vesz fel. A valószínűség számítás axiómáiból $F(x)$ jellemző tulajdonságai:

$$0 \le F(x) \le 1 \quad \text{minden } x\text{-re},$$

$$F(-\infty) = 0,$$

$$F(\infty) = 1.$$

Diszkrét valószínűségi változóestén az eloszlásfüggvény helyett kényelmesebb a

$$p(x_i) = P(X = x_i)$$

valószínűséggel dolgozni, amely a valószínűségi változó minden értékéhez hozzárendel egy valószínűséget. A $p(x)$ függvényt valószínűségi sűrűségfüggvénynek nevezzük.

A függvény az x_i minden lehetséges értékéhez meghatároz egy valószínűséget, amikor az X valószínűségi változó felveszi az x_i értékét. A valószínűség számítás axiómái előírják a következőket x_i -re:

$$0 \le p(x_i) \le 1 \text{ minden } i\text{-re},$$

$$\sum_i p(x_i) = 1 \text{ minden } i\text{-re}.$$

Elvileg közömbös, hogy egy diszkrét valószínűségi változóra az $F(x)$ vagy a $p(x)$ valószínűségeket ismerjük, mert az egyik a másikból könnyen meghatározható. Az eloszlásfüggvény kapcsolata a sűrűségfüggvénnyel:

$$F(x) = \sum_{x_i \le x} p(x_i).$$

Ha adottak a $p(x_i)$ valószínűségek, akkor a fenti képlettel meghatározható az eloszlásfüggvényük, míg ha az $F(x)$ ismert, akkor

$$p(x) = F(x_i) - F(x_{i-1}),$$

mert a $p(x_i)$ nem más, mint az $F(x)$ ugrása az x_i helyeken.

A diszkrét valószínűségi eloszlásra példaként, tekintsük a következő kísérletet. Dobjunk fel 3 érmét és az X valószínűségi változó jelölje a fejek számát, amely 0, 1, 2, 3 diszkrét értékeket vehet fel. Kérdés mekkora a valószínűsége annak, hogy a feldobás 0, 1, 2, 3 fejet eredményez. Összesen 8 elemi esemény (kimenet) létezik, melyből az egy-egy 0 vagy 3 fejet, a három-három pedig 1 vagy 2 fejet eredményez (*2.1. táblázat*). Az X sűrűségfüggvényét és eloszlás függvényét a *2.1.* és a *2.2. ábrák* szemléltetik.

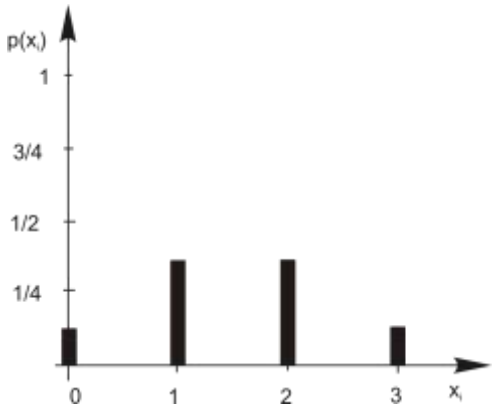
2.1. táblázat

Elemi esemény	Az események előfordulásának a száma	Valószínűség
0 fej	1	1/8
1 fej	3	3/8
2 fej	3	3/8
3 fej	1	1/8

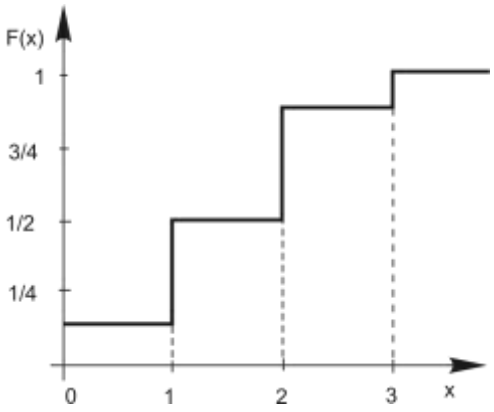
Ha egy X valószínűségi változó F(x) eloszlásfüggvénye az egész számegyenes minden pontjában folytonos (mint egy közönséges egyváltozós függvény) továbbá véges számú x hely kivételével differenciálható, és a deriváltja legalább szakaszonként folytonos, akkor az X-et ***folytonos valószínűségi változónak****, eloszlásfüggvényét, amelyre fennáll, hogy*

$$F(x) = \int_{-\infty}^{x} f(y)dy = P(X \le x)$$

abszolút folytonos eloszlásnak *mondjuk.*



2.1. ábra: Példa a diszkrét valószínűség-sűrűségfüggvényre



2.2. ábra: Példa a diszkrét eloszlásfüggvényre

Az $F(x)$ függvény megadja annak a valószínűségét, hogy a folytonos valószínűségi változó (X) kisebb vagy egyenlő, mint x.

Az abszolút folytonos eloszlásfüggvény deriváltját ***valószínűség sűrűségfüggvénynek****, vagy egyszerűen sűrűségfüggvénynek nevezzük. Az analízisből tudjuk, hogy ekkor*

$$[F(x)]' = f(x).$$

Mivel a valószínűségi változó bármilyen, végtelen sok értéket felvehet, egy adott pontban az előfordulásának a valószínűsége 0. Ez nem azt jelenti, hogy ilyen érték nem fordulhat elő, csak azt, hogy nagyon valószínűtlen a végtelen sok lehetséges érték miatt. Egy intervallumban két különböző pont, a és b között a változó értékeinek valószínűsége viszont nem lesz 0. A diszkrét esetre definiált sűrűségfüggvény helyett ezért a **folytonos valószínűség sűrűségfüggvényt**, a $f(x)$-et használjuk.

Ha a valószínűségi-sűrűségfüggvényt integráljuk a és b között, akkor megkapjuk annak a valószínűségét, amit a valószínűségi változó a és b közötti felvesz.

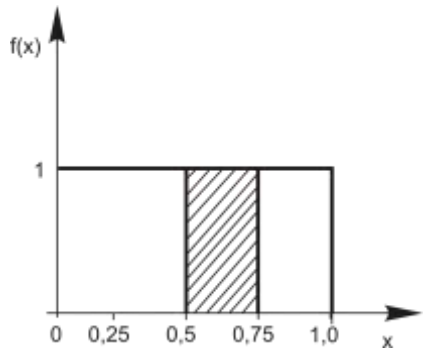
$$P(a \le X \le b) = \int_{a}^{b} f(x)dx$$

Következésképpen a valószínűség-számítás axiómáival összhangban, a valószínűség sűrűségfüggvénynek ki kell elégítenie a következőket:

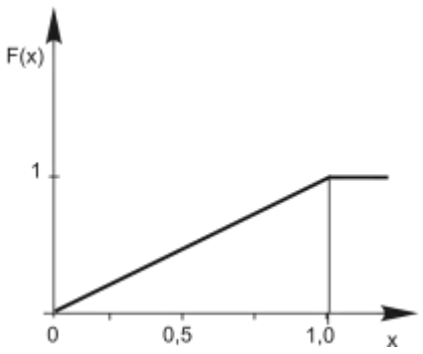
$$f(x) \ge 0.$$

$$\int_{-\infty}^{\infty} f(x)dx = 1.$$

A folytonos valószínűségi eloszlás bemutatására vegyünk egy valószínűségi változót (x-et), amely 0 és 1 között bármilyen értéket felvehet (*2.3.* és *2.4. ábrák*). Feltételezve, hogy minden nem megszámlálható, végtelen sok lehetséges érték valószínűsége egyenlő. A megfelelő valószínűség-sűrűségfüggvényt és az eloszlásfüggvényt a *2.3.* és a *2.4. ábra* mutatja. Annak valószínűsége, hogy az x 0,5 és 0,75 közötti értéket vesz fel, a görbe alatti területtel (a vonalkázott terület) egyezik meg, esetünkben 0,25-tel egyenlő.

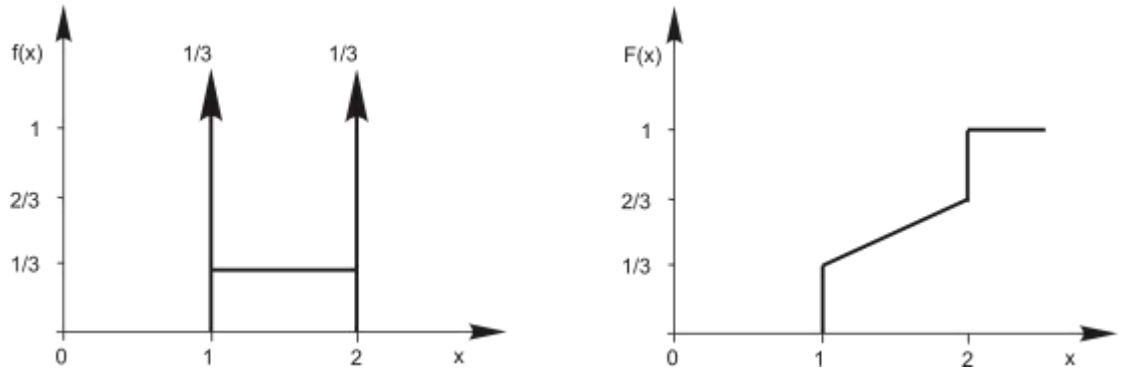


2.3. ábra: Példa a folytonos valószínűség-sűrűségfüggvényre



2.4. ábra: Példa a folytonos eloszlásfüggvényre

A valószínűségi változó egyszerre lehet folytonos és diszkrét. Az ilyen változókat "kevert" eloszlásúaknak nevezzük. A valószínűségi változó felvehet diszkrét értékeket véges valószínűséggel vagy a valószínűségi sűrűségfüggvénnyel leírt folytonos értékeket. A *2.5. ábra* ilyen eloszlást ábrázol, ahol az 1 és 2 diszkrét értékek egyaránt 1/3 valószínűséggel fordulnak elő. Az 1 és 2 közötti értékeket az $f(x)=1/3$ sűrűségfüggvény írja le.



2.5. ábra: Példa a kevert eloszlásra

Ilyen eloszlást kapnánk, ha a mintákat egy folytonos eloszlásból vennénk, ahol a 0 és 3-közötti értékek valószínűsége egyenlő, és a 2-nél nagyobb értékeket 2-nek, az 1-nél kisebb értékeket pedig 1-nek tekintjük. Erre az eloszlásfüggvény alakja:

$$F(x)=\begin{cases}0 & x<1\\ \frac{1}{3}+\frac{x-1}{3} & 1\le x\le 2\\ 1 & x\ge 2\end{cases}$$

Az eloszlásfüggvényből és a *2.5. ábrából* látható, hogy az $F(x)$ x=1–nél és x=2-nél nem folytonos. A szakadási helyeken a $P(X=x)$ egyenlő azzal az ugrással, amellyel az $F(x)$ változik az x pontnál, például a $P(X=1)=1/3$. Azonban az $1<x<2$ tartományban bármely x-nél az $F(x)$ folytonos és $P(X=x)=0$.

Az eloszlásfüggvények megmutatják egy valószínűségi változó karakterisztikáját. Néhány közismert eloszlásfüggvény: a normál vagy *Gauss*, az egyenletes, a háromszög, az exponenciális, a lognormális, az *Erlang*, a béta, a gamma és a *Poisson*. Később ezek grafikáit és alkalmazási lehetőségeit is bemutatjuk.

Ezen eloszlások generálásához és számítógépes szimulációjához szükségünk lesz véletlen számokra, amit ha digitális számítógépen generálunk, pszeudovéletlen számnak nevezzük.

Ezek egyenletes eloszlású véletlenszerű minták 0 és 1 közötti értékekkel (*2.3. ábra*), és az egymást követő minták függetlenek egymástól.

Egy eloszlásból pszeudovéletlen szám használatával generált véletlen megfigyelést véletlen változásnak, eltérésnek, vagy mintának nevezzük. Ez azt jelenti, hogy a véletlen megfigyelések sokasága közelíti azt az eloszlást, amelyből a véletlenszerű mintákat generáltuk, az összes megfigyelés pedig teljes egészében jellemzi ezt az eloszlást. A pszeudovéletlen számokról és véletlenszerű mintákról később bővebben szólunk.

2.5. Várhatóértékek és momentumok

A valószínűségi változó véletlen ingadozásait az eloszlásfüggvénye egyértelműen leírja. Gyakran kívánatos azonban, hogy a véletlen ingadozást egyetlen számértékkel jellemezzük. Mechanikai analógiával élve, a valószínűség fogalma analóg a tömeg fogalmával, a valószínűség-eloszlás pedig a tömegeloszlás fogalmával, amiből következik, hogy a valószínűség-eloszlás „centruma” a tömegközéppont fogalmához hasonlóan definiálható.

Ha a valószínűségi változó lehetséges értékei x_1, x_2,…x_n, amelyek valószínűségei $p(x_1)$, $p(x_2)$,…$p(x_n)$, akkor a tömegközéppont analógiájára definiálhatjuk a következő mennyiséget:

$$\frac{x_1 p(x_1) + x_2 p(x_2) + ... + x_n p(x_n)}{p(x_1) + p(x_2) + ... + p(x_n)},$$

mivel a $p(x_1) + p(x_2) + ... + p(x_n) = 1$, ezért a fenti mennyiség, amit az X valószínűségi változó várhatóértékének nevezünk:

$$x_1 p(x_1) + x_2 p(x_2) + ... + x_n p(x_n).$$

Ha az X valószínűségi változó lehetséges értékei x_1, x_2,…x_n, amelyeket rendre $p(x_1)$, $p(x_2)$,…$p(x_n)$ valószínűséggel vesz fel, akkor az X valószínűségi változó lehetséges értékeiből ezek valószínűségivel, mint súlyokkal képzett középértékét az X ***várhatóértékének*** *nevezzük, feltéve, hogy létezik.*

Az X valószínűségi változó várhatóértéke, amit $E(X)$-el jelölünk, az elmondottak értelmében:

$$E(X) = \sum_i x_i p(x_i), \qquad \text{ha az } X \text{ diszkrét,}$$

$$E(X) = \int_x x\, f(x)\, dx, \qquad \text{ha az } X \text{ folytonos.}$$

A várhatóérték az X valószínűségi változó lehetséges értékeinek súlyozott átlaga, vagyis az eloszlás központosságának mérőszáma, ezért középértéknek is nevezik.

Az X^n függvény várhatóértékét (n=1, 2,…) az X valószínűségi változó ***n-nedrendű momentumának*** *nevezzük, és a következő kifejezésekkel definiáljuk:*

$$E\{X^n\} = \sum_i x_i^n p(x_i), \qquad \text{ha az } X \text{ diszkrét,}$$

$$E\{X^n\} = \int_x x^n f(x)\, dx, \qquad \text{ha az } X \text{ folytonos.}$$

Könnyen belátható, hogy az első momentum (n=1) éppen az X várhatóértéke.

Az n-dik momentum középértéktől való eltérést mutatja az ún. ***n*-dik centrális momentum**:

$$E\{[X - E(X)]^n\},$$

ami az *X* valószínűségi változó és a várhatóértéke közötti eltérés *n*-dik hatványának a várhatóértéke.

Egy valószínűségi változó várhatóértéke azt a számértéket jelenti, amely körül a változó véletlen ingadozásokat végez. A várhatóérték azonban nem ad felvilágosítást az ingadozások nagyságáról. Az ingadozás mértékének mérésére többféle mérőszámot konstruáltak, amelyek közül a legfontosabb a **szórás**, ami nem más, mint a második centrális momentum. A második centrális momentumot, közismerten a szórásnégyzetet az *X* **varianciájának** is nevezzük, és *Var*(*X*)-el vagy σ^2 -tel jelöljük:

$$\sigma^2 = Var(X) = E\{[X - E(X)]^2\}.$$

A szórás definíciójából és a várhatóértékre vonatkozó tételekből levezethető, hogy a

$$\sigma^2 = Var(X) = E(X^2) - E^2(X).$$

A valószínűségi változó varianciája tehát a valószínűségi eloszlás ingadozásának (szóródásának) a mérőszáma. A kicsi szórás azt jelzi, hogy a minták a várhatóérték közelében helyezkednek el. A szórásnégyzet négyzetgyöke a valószínűségi változó alapeltérése.

Két vagy több valószínűségi változó együttes vizsgálatakor, ha nem függetlenek egymástól, általában két fontos kérdés merül fel. Az egyik a két valószínűségi változó közötti kapcsolat, amit sztohasztikus kapcsolatnak nevezünk, szorosságának a mérése. A másik fontos kérdés, hogyan lehet következtetni az egyik változó adott értékéből a másik változó értékeire. Az utóbbi kérdéssel a regressziós analízis foglalkozik.

Ha *X* és *Y* valószínűségi változó, akkor *X* és *Y* kovariánsát Cov(*X*,*Y*)-el jelöljük és következők szerint definiáljuk:

$$\mathrm{Cov}(X,Y) = E\{[X - E(X)][Y - E(Y)]\}.$$

Ha az *X* kimenet nincs hatással az *Y* kimenetre, akkor az *X* és az *Y* független, és a Cov[*X*,*Y*] nulla lesz. Ennek magyarázata a következő. Ha az *X* és *Y* valószínűségi változók függetlenek, akkor az *X*-*E*(*X*) és az *Y*-*E*(*Y*) változók is függetlenek, és ezért a két változó szorzatának a várhatóértéke a várhatóértékeik szorzatával egyenlő. Mivel azonban egy valószínűségi változónak a saját várhatóértékétől vett eltérésének a várhatóértéke mindig nulla, e szorzat várhatóértéke is nulla.

Az *X* és az *Y* akkor és csak is akkor független egymástól, ha

$p(y|x) = p(y)$ diszkrét esetben, ahol $p(y|x)$ annak valószínűsége, hogy az *Y*=*y* adott az *X*=*x*-el,

$f(y|x) = f(y)$ folytonos esetben, ahol $f(y|x)$ a feltételes sűrűségfüggvénye *Y*-nak *X*=*x*-re.

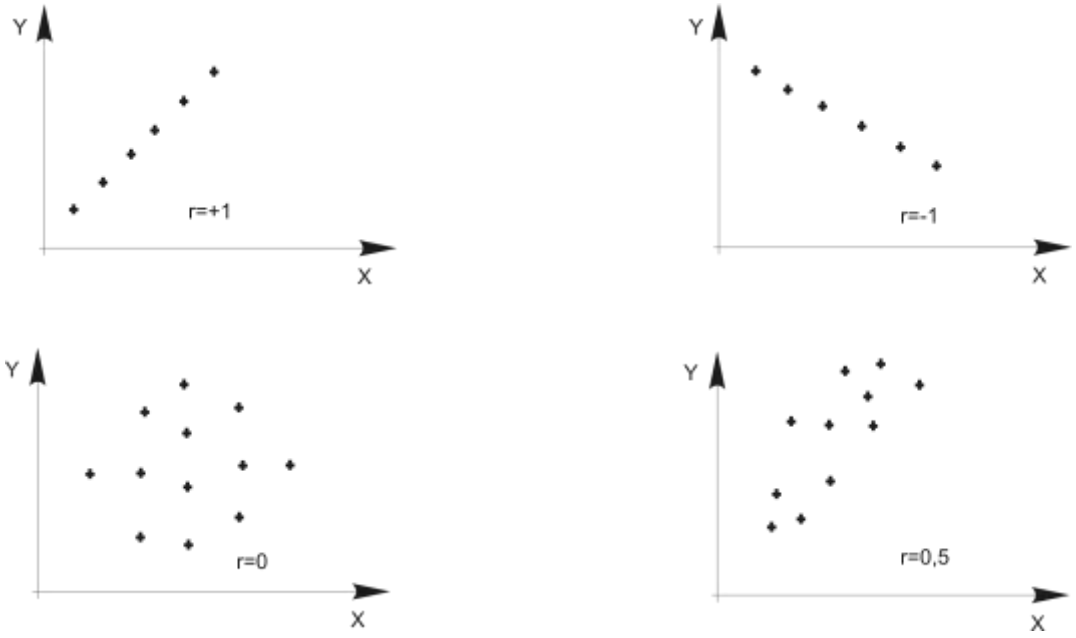
Ezen állítások szerint az *X* ismeretéből származó *Y* valószínűségi sűrűségfüggvény ugyanaz a függvény, mint amit *X* ismerete nélkül kapunk.

A kovariánstól való függés mértéke a korrelációs tényező (*r*):

$$r = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)}}$$

A korrelációs tényező –1 től +1-ig között vehet fel értékeket, és az *r*=0 azt jelenti, hogy nincs összefüggés *X* és *Y* között. A *2.6 ábrán* jellegzetes diagrammok láthatóak az *r* különböző értékeire. Pozitív előjel esetén növekvő *X*-hez növekvő *Y* tartozik, negatív előjel esetén az *Y*

értéke csökken, ha az X növekszik. Az r nagysága jelzi az X függvényében ábrázolt Y linearitásának mértékét. Ha az ábrázolt Y az X függvényében egyenes vonal, akkor a $\rho=\pm 1$. Ha X és Y független, akkor az ábra rendszertelen ponthalmazt mutat, és az $r=0$.



2.6. ábra: Az X-Y értékek szóródása különböző r értékek esetén

2.5.1 A teljes valószínűségelmélet

B kimenet valószínűsége egyenlő az egymást kölcsönösen kizáró B és az A_i kimenetek feltételes valószínűségének összegével, súlyozva az A_i valószínűségével, azaz

$$P(B)=\sum_i P(B|A_i)P(A_i)$$

Ezzel analóg módon az Y valószínűségi változó várhatóértéke:

$$E(Y)=\sum_i E(Y|X=x_i)\,P(X=x_i)$$

2.5.2 Együttes valószínűségek

A több valószínűségi változó együttes valószínűsége a változók feltételes valószínűségeinek a szorzata:

$$P(Y_1,Y_2,...,Y_n)=P(Y_1)P(Y_2|Y_1)P(Y_3|Y_1,Y_2)...P(Y_n|Y_1...Y_{n-1}).$$

Ha feltételezzük, hogy a valószínűségi változó markovi tulajdonságokkal rendelkezik, vagyis $P(Y_j|Y_1,Y_2,...,Y_{j-1})=P(Y_j|Y_{j-1})$, akkor a

$$P(Y_1,Y_2,...,Y_n)=P(Y_1)P(Y_2|Y_1)P(Y_3|Y_2)...P(Y_n|Y_{n-1}).$$

Ha feltételezzük a függetlenséget, akkor a

$$P(Y_1,Y_2,...,Y_n)=P(Y_1)P(Y_2)P(Y_3)...P(Y_n).$$

2.6. Valószínűségi változók függvényei

Egy valószínűségi változó függvénye maga is egy valószínűségi változó, Ebben a részben valószínűségi függvények néhány fontos tulajdonságát foglaljuk össze. Ha X és Y valószínű-

ségi változó és k egy tetszőleges konstans, akkor a várhatóérték következő tulajdonságai vezethetők le:

$$E(X+Y) = E(X) + E(Y)$$
$$E(kX) = kE(X)$$
$$E(X+k) = E(X) + k$$

A szórásra vonatkozó hasonló tulajdonságok már kevésbé magától értetődőek:

$$\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(y) + 2\mathrm{Cov}(X,Y)$$
$$\mathrm{Var}(kX) = k^2\,\mathrm{Var}(X)$$
$$\mathrm{Var}(X+k) = \mathrm{Var}(X)$$
$$\mathrm{Var}(kX+nY) = k^2\,\mathrm{Var}(X) + n^2\,\mathrm{Var}(Y) + 2kn\mathrm{Cov}(X,Y)$$

Megjegyezzük, ha X és Y független valószínűségi változók, akkor Cov(X,Y) nulla, és a

$$\mathrm{Var}(X+Y) = \mathrm{Var}(x) + \mathrm{Var}(Y) \qquad \text{minden független } X\text{-re és } Y\text{-ra.}$$

A statisztikában az egyik legfontosabb valószínűségi változó a mintaközép $\left(\overline{X}_N\right)$, egy valószínűségi eloszlás N számú mintájának a jellemzője. Definíció szerint a minták összege osztva a minták számával, azaz

$$\overline{X}_N = \frac{1}{N}\sum_{i=1}^{N} X_i$$

Feltételezve, hogy az X_i–k függetlenek és azonos eloszlású valószínűségi változók, a várhatóérték és a szórásnégyzet tulajdonságait használva a következőket kapjuk:

$$E(\overline{X}_N) = E(X)$$
$$\mathrm{Var}(\overline{X}_N) = \frac{\mathrm{Var}(X)}{N}$$

Az N számú független minta mintaközépének szórása az $1/N$-szer kisebb, mint annak a valószínűségi változónak szórása, amelyből a mintákat vettük. Ebből következik, hogy megfelelően nagy N választásával a mintaközép szórása tetszőlegesen kicsi értékre csökkenthető.

Fontos, hogy ($\overline{X}_N$) szórására adott fenti összefüggés csak akkor alkalmazható, ha a minták függetlenek. Ha ez nem igaz, Var($\overline{X}_N$) számításához szükség van a minták közti kovariancia vizsgálatára. A banki pénztár példában az egymást követő vásárlók várakozási idői összefüggenek egymással, mivel nagyobb a valószínűsége az (i+1)-dik vásárló várakozásának, ha az i-edik vásárló is várakozik, ahhoz képest, hogy az i-ediket éppen kiszolgálják. Ilyenkor az átlagos várakozási idő szórása nem számítható egyszerűen a várakozási idő szórásának és a minták számának hányadosaként. Az ilyen, egymással összefüggő minták sorát autokorrelációs sornak nevezzük. Ezen sorok becslésének problémájával a 2.13. alfejezetben foglalkozunk.

2.6.1 Független valószínűségi változók véletlenszerű összege

Ha X_1,X_2,…,X_k független és egyenletes eloszlású valószínűségi változók és K X_i-től független diszkrét valószínűségi változó, akkor összegük

$$Y = \sum_{i=1}^{K} X_i\ ,$$

és

$$E(Y) = E(X)E(K),$$

$$\mathrm{Var}(Y) = E(K)\,\mathrm{Var}(X) + \mathrm{Var}(K)E^2(X).$$

2.6.2 A módosítható képlet változásai

Ha adott:

$$Y_j = g_j(W_1, W_2, ..., W_n),\ j = 1,2,...,n,$$

akkor az Y_j, $f_y(.)$ együttes sűrűségfüggvénye

$$f_y(y_1, y_2, ..., y_n) = f_w(w_1, w_2, ..., w_n)\frac{1}{|J|},$$

ahol $f_w(.)$ az együttes sűrűségfüggvénye $W_1, W_2, \ldots, W_n$ és J a *Jacobi-féle* determináns mátrix:

$$\begin{pmatrix} \frac{\partial g_1}{\partial w_1} & \frac{\partial g_1}{\partial w_2} & \cdots \frac{\partial g_1}{\partial w_n} \\ \cdots & \cdots & \cdots\ \cdots \\ \frac{\partial g_n}{\partial w_1} & \frac{\partial g_n}{\partial w_2} & \cdots \frac{\partial g_n}{\partial w_n} \end{pmatrix}$$

és a $|J|$ a J abszolút értéke. Ez a módszer a valószínűségi változó transzformációit írja le.

2.6.3 Rao-Blackwell elmélet

Legyen X és Y valószínűségi változó, Y középértéke μ és szórása $\sigma_y^2 > 0$. Legyen $E(Y|x) = \phi(X)$, ekkor $E\{\phi(X)\} = \mu$ és $\sigma_{\phi(X)}^2 \le \sigma_Y^2$.

Ez az elmélet azt mondja, hogy ha az Y valószínűségi változó statisztikai tulajdonságaira vagyunk kíváncsiak, és ismerjük a vele összefüggő $\phi(X)$ valószínűségi változót, amely egyenlő az X-től függő Y várhatóértékével, akkor a $\phi(X)$ várhatóértékéből meg tudjuk becsülni μ-t, és ennek a becslésnek a szórásnégyzete legalább olyan kicsi lesz, mint a közvetlen becslés szórásnégyzete. Az elmélet megalapozza, hogy mintaközép becslésekor érdemes használni az előzetes információkat[2].

2.7. Generátorfüggvények

A valószínűség számításban gyakran alkalmazott függvény az ún. generátorfüggvény. A számos ismert típus közül e helyen csak a valószínűségek és a momentumok meghatározására használható generátorfüggvénnyel foglalkozunk. Egy diszkrét valószínűségi változó valószínűség generátorfüggvénye:

$$A(s) = \sum_i p(x_i)s^i$$

Ha az $A(s)$ zárt formája ismert, akkor a $p(x_i)$ valószínűségek az i-dik deriváltjából kaphatók s figyelembevételével, s értékének nullára állításával. (Ha az $A(s)$ egy polinom, a $p(x_i)$-t megfigyeléssel kaphatjuk.) Az X várhatóértékét kapjuk az $A(s)$ első deriváltjából, ha s=1. Magasabb rendű momentumokat hasonlóan, de a deriváltak kombinációival állítjuk elő. Ilyen függvény a Z- transzformáció.

[2] Ez a Rao-Blackwell elméletre vonatkozó megfigyelés képzi az előzetes információk segítségével történő varianciacsökkentő technika alapját. Erre először James R. Wilson mutatott rá.

Az X valószínűségi változó momentum generátorfüggvénye (**MGF**–moment generating function):

$$M(s) = E\left[e^{sX}\right].$$

Az indulástól számított n-dik momentum az s szerinti n-dik deriváltjaként kaphatjuk, azaz

$$\frac{d^n M(s)}{ds^n} = E(X^n e^{sX}).$$

Ha s=0, megkapjuk $E[X^n]$ -t. Ilyen függvény a karakterisztikus függvény. A generátorfüggvényekből a valószínűségi változók momentumai mellett megkapjuk azok összegeinek momentumait is. Például: ha W=X+Y, valamint X és Y függetlenek, akkor

$$E(e^{sW}) = E(e^{s(X+Y)}) = E(e^{sX})E(e^{sY})$$

Ezért W **MGF**-je X és Y **MGF**-jének terméke. W tényezőit megkaphatjuk annak **MGF** -jéből.

2.8. Nagy számok törvénye és a központi határeloszlás tétele

A „nagy számok törvényei" gyűjtőnév, amely a relatív gyakoriság és a valószínűség viszonyát megvilágító konvergencia tételeket tartalmazza. Az első ilyen *Bernoulli*-tól származó tétel, a **nagy számok erős törvénye**, ami kimondja, hogy ha minta mérete (N) tart a végtelenhez, akkor annak valószínűsége 1, hogy az $\overline{X}_N$ mintaközép tart az $E(X)$ várhatóértékhez, azaz

$$\lim_{N\to\infty} P\left\{\left|\overline{X}_N - E(X)\right| < \varepsilon\right\} = 1,$$

ahol az ε>0.

Ezzel összefüggésben a nagy számok gyenge törvénye:

$$\lim_{N\to\infty} P\left\{\left|\overline{X}_N - E(X)\right| \geq \varepsilon\right\} = 0 \text{ minden } \varepsilon\text{>0-ra}$$

Ez a határérték egyszerűen azt jelenti, hogy az ε bármilyen kicsi pozitív értékére annak a valószínűsége, hogy $\overline{X}_N$ és $E(X)$ közötti különbség eléri az ε-t, tart nullához, ha az N tart a végtelenhez. A *Bernoulli–tétel* sztohasztikus konvergenciát mond ki. Nem azt állítja, hogy a kísérletek számát növelve a relatív gyakoriság konvergál a valószínűséghez, hanem csupán azt, hogy egy előírt számnál nagyobb eltérés egyre valószínűtlenebb.

A második fontos elmélet, amely $\overline{X}_N$ viselkedését írja le a **központi határeloszlás tétel.** A tétel azt mondja ki, hogy bizonyos gyenge feltételek megléte estén az X-ből származó N független minták összegének eloszlása közelítőleg normális eloszlású, ha az N tart a végtelenhez, függetlenül az X eloszlásától.

Ebből következik, hogy megfelelően nagy N-nél a mintaközép ($\overline{X}_N$) megközelítően normális eloszlású. Nehéz megmondani, hogy mekkora mintaméret elegendő a normális eloszlás biztosításához. Gyakran kicsi, 10-15-ös minta elegendő. *Feller** két elméletet állított fel a központi határeloszlás tétellel kapcsolatosan használt normális közelítés minőségére vonatkozóan. Ezekkel az elméletekkel *Bratley*, *Fox* és *Schrage* foglalkozott. A központi határeloszlás tételnek sok változata létezik, de csak egy tartalmazza azokat a feltételeket, amely alkalmassá teszi a függő valószínűségi változók sorainak vizsgálatára. Ezeket a feltételeket a következő fejezetben vizsgáljuk.

2.9. Rekurrens események aszimptotikus normalitása

Ha egy esemény tartósan ismétlődő, továbbá az események közötti idő átlaga (μ) és szórásnégyzete (σ^2) véges, akkor T_r és N_t aszimptotikusan normális eloszlásúak, ahol T_r az r-dik esemény bekövetkezéséig eltelt idő, az $E(T_r)=r\mu$, és a $\mathrm{Var}(T_r)=r\sigma^2$. Az N_t a t idő alatt bekövetkezett események száma, $E(N_t)=t/\mu$ és $\mathrm{Var}(N_t)=t\ \sigma^2/\mu^3$. Például egy szimulációban az átlagos érkezési időköz μ=10 és a szórásnégyzet σ^2=4, akkor az érkezések száma t=1000 időegység alatt közelítőleg normális eloszlású 100 átlaggal és 4 szórásnégyzettel.

Az előző állítás a függőváltozók sorozatára vonatkozó központi határeloszlás tétel, és a szimulációs modell bemeneti jelgenerátorainak ellenőrzésére szolgálhat.

A szabályos eseményekre vonatkozó elmélet mellett létezik olyan központi határeloszlás tétel is, amely a mintaközép vagy a stacionárius sztochasztikus folyamatok szabályos körülményeit vizsgálja. *Moran* a mozgóátlag típusú folyamatokról számolt be, *Dianada* pedig olyan folyamatokkal foglalkozott, amelyekben független események r késéssel vagy időperiódusonként történnek.

2.10. Adatgyűjtés és analízis

Az adatok gyűjtése és analizálása a szimulációs modellezés lényeges elemei. Ezek egyrészt a bemenetek definiálásához, másrészt a modellel végzett kísérletek eredményeinek értelmezéséhez szükségesek. Ebben a fejezetben az adatok gyűjtéshez és analizálásához szükséges statisztikai alapokat tekintjük át.

2.10.1 Adatgyűjtés

Adatgyűjtés az a folyamat, amikor a vizsgált jelenségről információkat szerzünk. Ennek különböző módjai ismertek. Néhány esetben az adatok létező dokumentumokban találhatók és már elérhető formában állnak rendelkezésre. Ilyenkor a feladat ezeknek megtalálása. Más esetben az adatgyűjtés magában foglalhat kérdőív feldolgozásokat, felméréseket, megfigyeléseket vagy fizikai kísérleteket.

Összetett modellekhez (ilyenek a társadalmi vagy gazdasági rendszerek) szükséges adatok gyakran létező dokumentumokból nyerhetők. Ilyen modelleknél gyakori források a népszámlálási jelentések, a statisztikai kiadványok, a nemzetközi (pl. ENSZ) publikációk és más kormányzati valamint nemzetközi szervezetek jelentései. Ezek gyakran írott formában és számítógépen (pl. internetes adatbázisokban) egyaránt megtalálhatóak.

Vállalati és üzleti rendszerek modelljeinél értékes adatforrások a cégek könyvelései vagy a mérnöki dokumentációk. Ezek ritkán elegendőek a várható kereslet, gyártási költség és más hasonló adatok megállapításához, azonban megfelelő kiindulópontot jelentenek. A kérdőívek és a felmérések szintén fontos adatforrások az ipari modellek felállításához.

Fizikai kísérlet általában az adatszerzés költséges és időigényes módja. Ez magában foglalja a méréseket, az adatok gyűjtését és feldolgozását. A reprezentatív körülményeket, és a korrekt adatrögzítést biztosító kísérletek megtervezésére nagy gondot kell fordítani.

Néhány esetben az adatok nem léteznek és a kísérleteket anyagi keret hiányában nem tudjuk elvégezni, vagy annak természete miatt a kísérlet eleve lehetetlen. Ilyennek tekinthető például a különböző tervszinten létező gyártóvonalak működésének összehasonlítása szimulációs modellezéssel. Ebben az esetben az adatgyűjtést mesterséges vagy becsült adatok előállításával helyettesítjük. Például a tevékenységek időtartamát táblázatok, szabványok, korábbi megfigyelések felhasználásával becsüljük.

2.10.2 Leíró statisztikák

Minden esetben, amikor adatokat gyűjtünk egy modell bemeneti jellemzőinek meghatározásához, akkor szembesülünk azzal a problémával, hogyan rendezzük a nyersadathalmazt áttekinthető és felhasználható formába. Az emberi elme képtelen nagy és rendezetlen adathalmazok áttekintésére, ezért szükségünk van valamilyen tömörítési eljárásra. Ezek az eljárások ugyan információkat áldoznak fel, helyette azonban áttekinthetőséget és feldolgozásra való alkalmasságot kapunk.

Az adatok csoportosítása: Az adatok használható formába való rendezésének egyik lehetséges módja az osztályokba vagy cellákba sorolás. Ez azt jelenti, hogy egy-egy csoportba (osztályba) vonjuk össze azokat az adatokat, amelyek értéke egy bizonyos intervallumon belül helyezkedik el, és azt is meg tudjuk adni, hogy egy-egy részterjedelemhez (osztályhoz) hány adat tartozik. Az egyes osztályokba eső adatok számát nevezzük **gyakoriságnak.** Az osztályokba sorolt adatok így összevonhatók és áttekinthető táblázatba rendezhetők. Ezt a táblázatot **gyakorisági eloszlás táblázatnak** nevezzük, ami átfogó képet nyújt az adatokról.

2.2. táblázat

Várakozási idő (sec)	Ügyfelek száma
0-20	21
20-40	35
40-60	42
60-80	35
80-100	19
100-120	10
>120	10

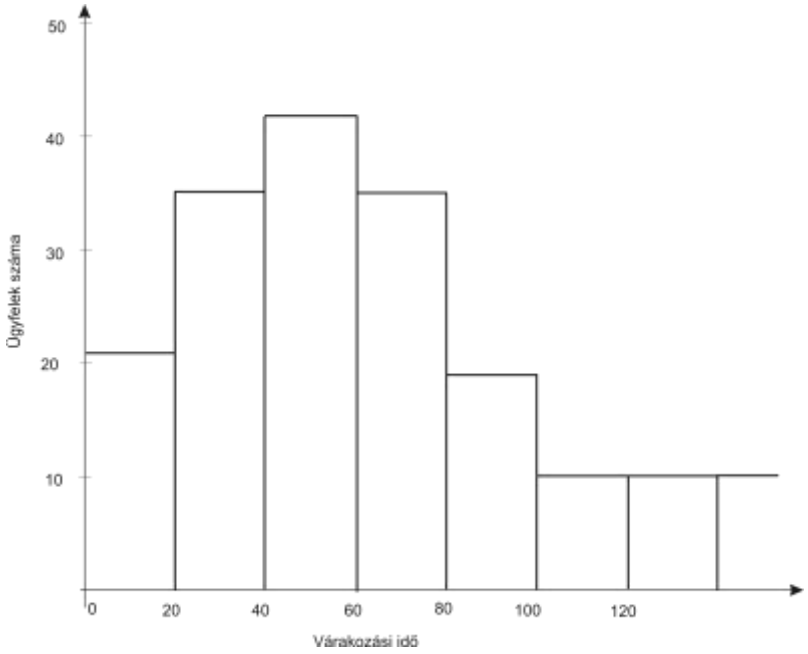
A gyakorisági eloszlás táblázat alkalmazásának jellegzetes példája az ügyfelek várakozási idejének csoportosítása (*2.2. táblázat*). A táblázatban 172 ügyfél várokozási idejét figyeltük meg, és ezeket 20 szekundum szélességű osztályokba soroltuk. A jobboldali oszlopba írt, számokat (ügyfelek száma) osztályozott adatoknak vagy **osztálygyakoriságnak** hívjuk. A baloldali oszlop számai jelzik az osztályok határait és határozzák meg az egyes osztályok intervallum szélességét. Az alsó és felső osztályhatár közötti különbséget **osztályszélességnek** hívjuk.

2.3. táblázat

Várakozási idő kisebb mint (sec)	Ügyfelek kumulált száma
20	21
40	56
60	98
80	133
100	152
120	162
∞	172

A végtelen alsó és felső határú osztályokat **nyitottnak**, véges határú osztályokat **zártnak** nevezzük. Gyakran az első és/vagy az utolsó osztály nyitott. A gyakorisági táblázatnak több olyan változata létezik, amelyek hasznosak lehetnek a csoportosított adatok bemutatására. Az **egyik változat** az ügyfelek várakozási idejének jellemzésére a **halmozott gyakorisági táblázat**, ami a *2.2. táblázatból* a gyakoriságok kumulálásával (folytonos hozzáadásával) nyerhető (*2.3. táblázat*).

A jobboldali oszlopban a halmozott gyakoriságok mutatják azoknak az ügyfeleknek a teljes számát, akik kevesebb időt várakoztak, mint a baloldali oszlopban megadott felső osztályhatár. A **másik változatot** úgy kapjuk, hogy a gyakorisági táblázat (vagy a halmozott gyakorisági táblázat) jobboldali oszlopaiban található gyakoriságokat (vagy halmozott gyakoriságokat) elosztjuk az összes adatpontok számával (a példában 172-vel). Az így kapott hányadosokat relatív gyakoriságoknak is nevezik. A relatív gyakorisági eloszlás kifejezetten előnyös, 2 vagy több eloszlás összehasonlításakor.



2.7. ábra: Ügyfelek várakozási idejének a hisztogramja.

Az adatok értelmezhetőségének növelésének érdekében a gyakorisági és a kumulált gyakorisági eloszlást időnként grafikusan ábrázolják. A leggyakrabban használt és a legismertebb grafikus ábrázolás a hisztogram, amelyen az osztálygyakoriságokat olyan téglalapok jelölik, amelyek magassága arányos az osztálygyakorisággal. A *2.7. ábra* például ügyfelek várakozási időiből (*2.2. táblázat*) szerkesztett hisztogramot mutatja.

A gyakorisági eloszlás táblázat megtervezésekor az elsődlegesen eldöntendő kérdés az osztályok számának, és azok alsó és felső határainak a meghatározása. Ezek a döntések az adatok természetétől és végleges felhasználásától függenek, és ajánlatos a következő irányelvek figyelembevétele:

1. Az osztályok szélessége lehetőleg egyenlő nagyságú legyen. Kivétel ez alól az első és utolsó osztály, ami általában nyitott.

2. Az egyes adatok csak és kizárólag egyetlen osztályban szerepelhetnek. Azaz, az osztályok intervallumai nem fedhetik át egymást, ami kizárja, hogy egy adat két osztályba essen

3. Normál esetben legalább 5, de nem több mint 20 osztályt használjunk.

Paraméterbecslés: Ha az adathalmaz valószínűségi változóra vonatkozó összes lehetséges megfigyelést tartalmazza, akkor **populációnak**, ha viszont csak a lehetséges megfigyelések egy részét, akkor **mintának** nevezzük. Az adathalmaz összegzésére egy másik lehetséges eljárás, hogy az adatokat olyan mintának tekintjük, amellyel megbecsüljük a forrás- vagy alappopulációt. A populáció legfontosabb paraméterei a középérték, amely a központosság mértékét, és a variancia (szórásnégyzet) ami, az eltérés mértékét jellemzi.

A szemléltetéshez térjünk vissza az ügyfelek várakozási idejének adataihoz. Ez az adathalmaz úgy is tekinthető, mint egy minta a populációból, ami a megfigyelt ügyfelek várakozási idejét tartalmazza. Ezt a mintát felhasználhatjuk arra, hogy megbecsüljük a populációjához tartozó lehetséges ügyfelek átlagos várakozási idejét és várakozási idő szórásnégyzetét.

A populáció paramétereit, illetve az ezek becslésére használt, a mintából számított értékeket különböző szimbólumok jelölik. A populáció középértékét és szórását (varianciáját) általában görög betűk jelölik: μ a populáció középértéke, σ^2 a populáció szórásnégyzete. Az x_1, x_2, x_3,… x_i, N elemű minta alapján becsült átlagot pedig $\bar{x}_N$ -nel, és a varianciát s_X^2 -nel jelöljük. A további megkülönböztetés a populáció és a minta leírása használt számok között, hogy az előbbit paraméterként, utóbbit pedig statisztikai változóként használjuk.

Mielőtt a leíró statisztikák tárgyalásába belemennénk, tisztáznunk kell a véletlen minták és a valószínűségi változók kísérleti értékeinek (sztohasztikus sorainak) jelöléseit. A megfigyelés előtt egy valószínűségi változó kísérleti értékeit X_i-vel és a megfigyelés után x_i -vel jelöljük. A megfigyelés előtti a mintaközép $\overline{X}_N$, ami az N-elemű minta eleminek összege osztva N-nel, a megfigyelés utáni átlag pedig $\bar{x}_N$, ami az N számú megfigyelt x_i összege osztva N-nel. Hasonló módon, az S_X^2 egy valószínűségi változó becsült varianciája mielőtt a tényleges értékét meghatároznánk, és az s_X^2 a megfigyelt értékek becsült varianciája. Ez a jelölés rendszer igazodik ahhoz a gondolatmenethez, hogy a véletlen változókat nagybetűkkel jelöljük, ahol az lehetséges, és a numerikus mennyiségeket kis betűkkel.

Egy populáció minta-adatsoron (adatokon) alapuló paraméterbecslésének két, alapvetően különböző esete létezik. Az első esetben olyan minta-adatsort használunk, amelynek csak megfigyelt értékeit vesszük figyelembe, és az időt nem, amikor a megfigyeléseket rögzítettük. Az ügyfél várakozási időket jellemző adatsor (*2.2. táblázat*) jó példája ennek a megfigyelési módnak, ahol csak a várakozási időket rögzítettük, és nem foglalkoztunk azzal, hogy a várakozás mikor történt. Az időtől független minta adatsoron alapuló statisztikai adatokat **megfigyelésen alapuló statisztikai adatoknak** hívjuk.

A második esetben olyan változókat használunk, amelyek értékeit az időhöz rendeljük. Például egy bankban egy adott pillanatban foglalt ügyintézők száma olyan valószínűségi változó, ami az idő függvényében változik. Ebben az esetben nemcsak a foglalt ügyintézők számát kell megfigyelni, hanem a megfigyelés időpontját is rögzíteni kell, és hozzá kell rendelni a valószínűségi változóhoz. Azokat a statisztikákat, amelyek időfüggő értékekből származtatott adatsorok, **időben-folytonos statisztikai adatoknak** hívjuk.

Az $\bar{x}_i$ és az s_x^2 kiszámítására alkalmas képleteket mind a megfigyeléseken alapuló, mind az időben-folytonos statisztikákhoz a *2.4. táblázatban* foglaltuk össze. Az időben-folytonos adatsor mintaátlagát $\bar{x}_T$ -vel jelöltük, ahol T a megfigyelés teljes időintervalluma. Az s_x^2 -re megadott képletektől eltérő formák is ismertek. Számítási célokra a táblázatban feltüntetett forma a legalkalmasabb. Megjegyezzük, hogy a megfigyelésen alapuló statisztikáknál a

$$\sum_{i=1}^{N} x_i \,,\ \sum_{i=1}^{N} x_i^2$$

értékek ismerete és a minták száma N elégséges az $\bar{x}_i$ és s_x^2 kiszámításához. Hasonlóképpen az időben-folytonos változón alapuló statisztikák számításához a

$$\int_0^T x\,dt \,,\ \int_0^T x^2\,dt$$

és *T* értékek szükségesek.

2.4. táblázat

Paraméter	Képlet	
	megfigyelésen alapuló statisztika	időben-folytonos statisztika
Mintaközép	$\bar{x}_N = \frac{\sum_{i=1}^{N} x_i}{N}$	$\bar{x}_T = \frac{\int_0^T x(t)\,dt}{T}$
Mintaszórás	$s_X^2 = \frac{\sum_{i=1}^{N} x_i^2 - N\,\bar{x}_N^2}{N-1}$	$s_X^2 = \frac{\int_0^T x^2(t)\,dt}{T}$

Adathalmazok jellemzésére általánosan használt statisztika a szórás koefficiens $s_X / \bar{x}_N$, amit **relatívszórásnak** is neveznek. A relatívszórás a minta normális eltérését mutatja a mintaközéphez viszonyítva. A relatívszórás használata előnyös, ha két vagy több adathalmaz szórását hasonlítunk össze.

Szórásbecslés: Egy ehhez kapcsolódó, de sokkal bonyolultabb probléma a populáció eloszlásának meghatározására a minta-adatsor segítségével. Modellezéskor ez a probléma gyakran felmerül, mert a rendszer véletlenszerű tényezőit partikuláris eloszlásokkal kell jellemezni.

Bár az elméleti eloszlások tulajdonságainak a leírása segíti a modellezőt a megfelelő eloszlás megválasztásában, gyakran szükséges az adatsorra vonatkozó hipotézis tesztelése egy vagy több illeszkedési teszt segítségével. Ezek közül a *khi-négyzet* és a *Kolmogorov-Szmirnov* tesztek valószínűleg a legismertebbek. Ezek leírásai és alkalmazásai minden statisztikai szakkönyvben megtalálhatók. A adatsornak megfelelő eloszlásfüggvény meghatározásával, az eloszlásfüggvények illesztésével később részletesebben is foglalkozunk

2.11. Statisztikai következtetés

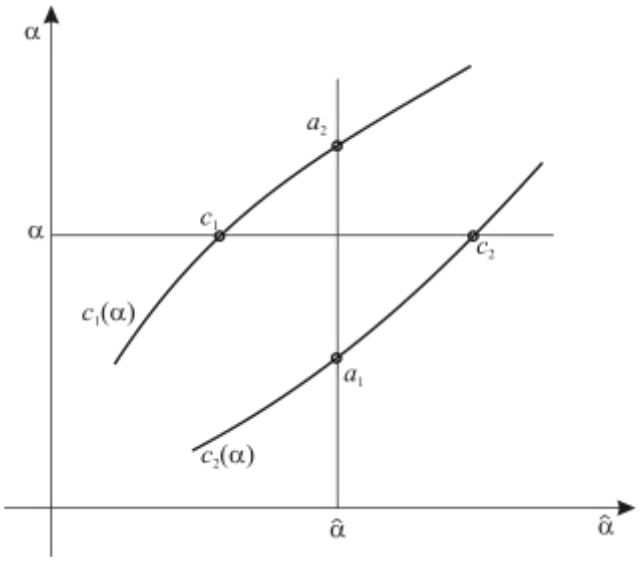
Szimulációs tanulmányokban a vizsgált rendszer viselkedésére vonatkozó becsléseknek és előrejelzéseknek a rendszerből szerzett kísérleti eredményeken alapulnak. Mivel a szimulációs modell véletlenszerű elemeket tartalmaz, a szimuláció kimenetein is véletlen változók mintái figyelhetők meg. Következésképpen, bármely, a rendszer működésére vonatkozó állítás, amely a szimuláció eredményein alapul, a szimuláció kimenetének változatosságától elválaszthatatlan állításnak tekinthető. Ez a változatosság összegződik, ezért az eredményeket mindig a konfidencia-intervallumokkal vagy a hipotézis vizsgálatokkal együtt kell kezelni.

2.11.1 A konfidencia-intervallumok és a megbízhatósági szint fogalma

Ha *X* valószínűségi változó eloszlásnak valamilyen α paraméterére *N*-elemű minta alapján $\hat{\alpha}$ becslést készítünk, akkor még a legjobb statisztikai tulajdonságok esetén is, ez az $\hat{\alpha}$ becslés nem adja meg pontosan α értékét. Tekintve, hogy $\hat{\alpha}$ valószínűségi változó, ezért az elméleti érték és a becslés között mindig jelentkezik legalább egy véletlen jellegű eltérés, amelynek nagyságára valamit mondanunk kell ahhoz, hogy a mintából megbízható következtetéseket tudjunk levonni. Szükség van, tehát olyan alsó és felső értékkel határolt intervallum megadására, amely az ismeretlen paramétert elég nagy valószínűséggel tartalmazza. Más szóval olyan (a_1, a_2) intervallumot keresünk, ami 1-hez közeli valószínűséggel tartalmazza α valódi értékét:

$$P(a_1 < \alpha < a_2) = 1 - \varepsilon,$$

ahol ε egy általunk választott, tetszőlegesen kicsi pozitív szám.



2.8. ábra: A konfidencia-intervallumok konstrukciójának alapelve

Az (a_1, a_2) véletlen helyzetű intervallum $1-\varepsilon$ valószínűséggel lefedi az α paramétert. Abban az esetben, ha az ismeretlen paraméter egy állandó, a megbízhatósági intervallumot pontosabban a következőképpen definiáljuk. Az X valószínűségi változó eloszlásfüggvénye tartalmazza az α paramétert. Vegyünk egy mintát X értékeire és ennek alapján készítsünk el egy $\hat{\alpha}$ becslést α-ra. Ennek a becslésnek az eloszlása természetesen tartalmazza az α paramétert, így minden $\hat{\alpha}$ értékhez található olyan α-tól és ε -tól függő c_1 és c_2 szám, hogy a $c_1 < \hat{\alpha} < c_2$ egyenlőtlenség $1-\varepsilon$ valószínűséggel teljesüljön:

$$P(\hat{\alpha} < c_1) = \varepsilon/2 \text{ és } P(\hat{\alpha} < c_2) = 1 - \varepsilon/2, \text{ amiből:}$$

$$P(c_1 < \hat{\alpha} < c_2) = 1 - \varepsilon, \quad \varepsilon > 0.$$

Rögzített mintaelem-szám és ε mellett c_1 és c_2 csak α-tól függ, α változtatásával az (α, c_1) és (α, c_2) pontok egy görbét írnak le, amint azt a *2.8. ábra* mutatja.

Jelölje a_1 és a_2 azokat a pontokat ahol az $\hat{\alpha}$ pontban húzott függőleges egyenes metszi a görbéket. Ekkor a

$$c_1 < \hat{\alpha} < c_2 \text{ és } a_1 < \alpha < a_2$$

egyenlőtlenségek ekvivalensek egymással, ezért a

$$P(a_1 < \alpha < a_2) = 1 - \varepsilon$$

összefüggés is fennáll.

Az (a_1, a_2) véletlen helyzetű intervallumot **konfidencia-** (megbízhatósági) **intervallumnak**, az $(1-\varepsilon)100$ %-ot a **megbízhatóság szintjének**, az intervallum kezdő és végpontját pedig **konfidencia határoknak** nevezzük. A fenti összefüggés tehát azt fejezi ki, hogy az (a_1, a_2) intervallum $1-\varepsilon$ valószínűséggel lefedi az ismeretlen paramétert. Az (a_1, a_2) megbízhatósági határok valószínűségi változók, minden egyes mintához más és más intervallum tartozik, α pedig állandó, ezért egy konkrét minta esetén az intervallum vagy tartalmazza az ismeretlen α értéket, vagy nem.

Az $1-\varepsilon$ megbízhatóság tehát azt jelenti, hogy ha sok mintát veszünk, akkor az intervallum átlagosan az esetek $100(1-\varepsilon)$ %-ában tartalmazza a paraméter valódi értékét, míg 100ε %-ában nem. Eszerint kisebb ε nagyobb megbízhatósági szintet ad, ez azonban azonos mintanagyság mellett általában nagyobb intervallumhoz vezet. Adott megbízhatósági szinten az intervallum a mintaelem-szám növelésével szűkíthető, N növelésével $\sqrt{N}$ arányban csökken az intervallum hossza.

Konfidencia-intervallum a normális eloszlás várhatóértékére ismert szórás esetén

A szimulációs analízisben elsődleges fontosságú paraméter a populáció középérték (μ). A konfidencia-intervallum klasszikus alkalmazása a középérték becslésére azt feltételezi, hogy a mintaelemek függetlenek, és ismerjük a szórásukat (σ).

A következő eljárás abban az esetben is alkalmazható, ha az X valószínűségi változó nem normális eloszlású, de N elég nagy. A centrális határeloszlás tételből következik, hogy $\overline{X}_N$ eloszlása közelítően normális –elég nagy N esetén– függetlenül attól, hogy maga az X valószínűségi változó milyen eloszlást követ.

Legyen X normális eloszlású μ várható értékkel és σ szórással. Tegyük fel, hogy σ értékét ismerjük, azonban μ értéke ismeretlen. Vegyünk egy N-elemű mintát és becsüljük az ismeretlen μ paramétert a minta átlagával. Ez az eset áll fenn például, ha egy fizikai mennyiségre N mérést végzünk, és ismert a műszer pontossága. Mivel $\overline{X}_N$ torzítatlan becslése az μ várható értéknek, azaz $E(\overline{X}) = \mu$ és $\overline{X}_N$ szórása X szórásának $\sqrt{N}$ -ed része, ezért $\overline{X}_N$ normális eloszlású μ várható értékkel és $\sigma_{\overline{X}} = \sigma/\sqrt{N}$ szórással. Ebből következik, hogy az

$$u = \frac{\overline{X}_N - \mu}{\sigma_{\overline{X}}}$$

valószínűségi változó $N(0, 1)$ eloszlású. Az $N(0, 1)$ eloszlás táblázatából adott $\varepsilon>0$ esetén meghatározható az az u_ε szám, amelyre teljesül, hogy

$$P(|u| \le u_\varepsilon) = 1-\varepsilon = 2\Phi(u_\varepsilon) - 1$$

Ebből

$$P\left(\left|\overline{X}_N - \mu\right| \le u_\varepsilon \sigma_{\overline{X}}\right) = P\left(\overline{X}_N - u_\varepsilon\sigma_{\overline{X}} \le \mu \le \overline{X}_N + u_\varepsilon\sigma_{\overline{X}}\right) = 2\Phi(u_\varepsilon) - 1$$

$$2\Phi(u_\varepsilon) - 1$$

ami azt fejezi ki, hogy az

$$\overline{X}_N - u_\varepsilon\sigma_{\overline{X}};\ \overline{X}_N + u_\varepsilon\sigma_{\overline{X}}$$

véletlen helyzetű intervallum $2\Phi(u_\varepsilon) - 1$ valószínűséggel lefedi az eloszlás μ várható értékét, ez tehát μ-re $(1-\varepsilon)100\%$ szintű konfidencia-intervallum. Ebben az esetben a mintából kapott $\overline{X}_N$ eltérése az μ várható értéktől legfeljebb az intervallum $u_\varepsilon\sigma_{\overline{X}}$ félhossza $(1-\varepsilon)$, 100 %-os szinten.

Felvethetjük a kérdést, hogy milyen nagynak kell lenni a minta elemszámának ahhoz, hogy adott megbízhatósági szint és megengedett $|d|$ hiba esetén a minta átlagának a várható értéktől való eltérése legfeljebb d legyen.

Ekkor $|\overline{X}_N - \mu| \le d$ és az előbbiek alapján

$$u_\varepsilon \frac{\sigma}{\sqrt{N}} \le d$$

egyenlőtlenségből, a szükséges mintaelem-számra az

$$N \ge \frac{u_\varepsilon^2 \sigma^2}{d^2}$$

összefüggést kapjuk.

Konfidencia-intervallum a normális eloszlás várhatóértékére ismeretlen szórás esetén

Az előző részben tárgyalt esettel szemben gyakoribb, amikor a vizsgált valószínűségi változó szórását nem ismerjük, hanem azt is a minta alapján az S_X korrigált empirikus szórással becsüljük. Ekkor a

$$t = \frac{\overline{X}_N - \mu}{S_X / \sqrt{N}} = \frac{\overline{X}_N - \mu}{S_{\overline{X}}}$$

változó már nem normális eloszlású, és a *t* kifejezése felírható a következő formában is:

$$t = \frac{\sqrt{N-1}\,\dfrac{\overline{X}_N - \mu}{\sigma_{\overline{X}}}}{\sqrt{N-1}\,\dfrac{S_{\overline{X}}}{\sigma_{\overline{X}}}},$$

ahol a számlálóban, egy $N(0, 1)$ eloszlású valószínűségi változó, a nevezőben pedig $(N-1)$ szabadságfokú χ- eloszlású változó szerepel. A *t*-eloszlása tehát $(N-1)$ szabadságfokú *Student*-eloszlást követ. Ebben az esetben a *Student*-eloszlás táblázatból, adott megbízhatósági szinthez meghatározható az a kritikus t_p szám, amelyre teljesül a

$$P(|t| \le t_p) = S_{N-1}(t_p) = 1 - p$$

egyenlőség. Az így adódó $(\overline{X}_N - t_p S_{\overline{X}};\ \overline{X}_N - t_p S_{\overline{X}})$ konfidencia-intervallum $(1-p)100\%$ megbízhatósági szinttel rendelkezik a μ paraméterre.

2.11.2 Tolerancia intervallumok

A **tolerancia intervallum** egy olyan statisztikai intervallum, amelybe egy statisztikai sokaság meghatározott hányada bizonyos valószínűséggel beleesik. (Emlékezzünk vissza, hogy a megfigyelt sokaság és annak átlaga véletlen változók.) Ez az intervallum különbözik a konfidencia-intervallumtól, amelynek határai között a statisztikai sokaság paraméterei (átlag, szórás, stb.) találhatók meg nagy valószínűséggel. A tolerancia intervallum határai által megadott tartományba viszont a statisztikai sokaság meghatározott arányát képviselő adatok lehetséges értékei tartoznak. Egyszerűbben fogalmazva, míg konfidencia-intervallum azt mutatja, hogy mit tudunk egy adott mennyiségről (értékről), addig a tolerancia intervallum egy adathalmaz egészének értékeit jellemzi. Ha a valószínűség 100%, mert a statisztikai sokaság eloszlásának paraméterei pontosan ismertek, akkor a tolerancia intervallum a valószínűségi intervallummal azonos.

Először egy intervallumot határozunk meg a véletlen változó részére, majd meghatározzuk a véletlen minta bizonyos hányadát, amelytől megkívánjuk, hogy a tolerancia határokon belül helyezkedjen el, továbbá kikötjük, hogy az intervallum milyen megbízhatósággal tartalmazza ezt a véletlen minta hányadot. Pontosabban, előírjuk $(1-\delta)$ valószínűséggel, hogy az *N* méretű

véletlen mintán alapuló $\overline{X}_N$ megfigyelés $(1-\in)$ hányada a tolerancia határok közé essen. *Wilson* kifejlesztett egy formulát a tartomány kiszámítására [10]:

$$\overline{X}_N \pm u_{\in} Q \frac{S_x}{\sqrt{N}}$$

ahol $u_{\in}$ a normál eloszlás egy kritikus értéke megfelel az $(1-\in)$ megbízhatóságnak. A Q a következő formulával adott:

$$Q = \frac{(2N+1)}{2N} \sqrt{\frac{(N-1)}{\chi^2_{\delta,N-1}}}$$

ahol $\chi^2_{\delta,N-1}$ az $(1-\delta)$ megbízhatósági szintű és $(N-1)$ szabadsági fokú *khi*-négyzet eloszlás kritikus értéke.

Wilson egy N futtatásból álló kísérletre, a következő egyszerűsített tolerancia intervallum formulát vezette be:

$$\overline{X}_N \pm u_{\in} Q \sigma_{\overline{X}},$$

amelynek a valószínűsége legalább $(1-\delta)(1-\in)$, és amelybe X_{N+1} megfigyelés beleesik.

2.12. Hipotézisvizsgálat

Egy statisztikai sokasággal kapcsolatban különböző hipotéziseket, feltevéseket fogalmazhatunk meg. Ezeknek a feltevéseknek a sokaságból származó az $X_1,\ldots,X_N$ minta alapján történő vizsgálatát **hipotézisvizsgálatnak,** vagy **statisztikai próbának** nevezzük. A hipotézisvizsgálat alapján döntünk arról, hogy a hipotézist elfogadjuk, vagy pedig elutasítjuk.

A hipotézis vonatkozhat a statisztikai sokaság eloszlására, az eloszlás valamilyen paraméterére, vagy a sokaság valamilyen jellemzőjére. A szimulációs modellezéskor is gyakran feladat az, hogy eldöntsük egy paraméterrel kapcsolatos állításról, hogy igaz vagy hamis. Például, el kívánjuk dönteni, hogy egy rakodási módszer megváltoztatása csökkenti-e az átlagos várakozási időt a vizsgált folyamatban.

A statisztikai próba végrehajtásának logikai menete a következő:

- a nullhipotézis felállítása,
- a próbafüggvény megszerkesztése és becslése,
- a kiválasztott p szignifikanciaszint mellett a kritikus érték meghatározása,
- a nullhipotézis elfogadása, illetve elvetése a próbafüggvény konkrét értéke és a kritikus érték alapján.

A statisztikai hipotézisek ellenőrzése –véletlen minta alapján– a valószínűség számítás törvényszerűségeinek alkalmazásával csak akkor végezhető el, ha a hipotézis matematikai formában fogalmazható meg. Ennek szokásos módja matematikai formában megfogalmazott a **nullhipotézis** (jele : H_0) **felállítása,** amely feltevés úgy fogalmazható meg, hogy az összehasonlítandó jellemzők között nincs statisztikailag igazolható különbség, vagyis a különbség várható értéke 0. Mivel a nulla egyformaságot jelent, el kell dönteni, hogy az összehasonlítandó adatok között ténylegesen az egyformaság vagy a különbözőség a jellemző. A nullhipotézissel szembeni lehetőséget **ellenhipotézisnek,** vagy **alternatív hipotézisnek** nevezzük (jele: H_1).

A nullhipotézis elfogadásából természetesen nem következik, az hogy a nullhipotézis igaz, csak annyi, hogy a véletlen minta alapján nincs elegendő bizonyítékunk a hipotézis elutasítá-

sára. Ennek oka, hogy a nullhipotézis is valószínűségi állítás, mivel a biztos következtetéshez csak az alapsokaság egészének vizsgálatával lehetne eljutni.

A nullhipotézist és az ellenhipotézist úgy vesszük fel, hogy kizárják egymást, és általában az is igaz, hogy a kettő közül az egyik mindig bekövetkezik. A nullhipotézis és az ellenhipozézis is lehet (egymástól függetlenül) egyszerű vagy összetett. Az **egyszerű hipotézisnek** egyetlen eleme van, míg az **összetett hipotézis** több önálló hipotézis összessége.

Például, ha be akarunk vezetni egy rakodási módszert (A), azért hogy csökkentsük az átlagos várakozási időt B rakodási módszerhez viszonyítva, akkor a nullhipotézist és az ellenhipotézist a következők szerint fogalmazhatjuk meg:

H_0 : az átlagos várakozási idő az A rakodási módszer esetén egyenlő a B rakodási módszer átlagos várakozási idejével,

H_1 : az átlagos várakozási idő az A rakodási módszer esetén kisebb, mint a B rakodási módszer átlagos várakozási ideje.

Ekkor az A és a B rakodási módszerre kapott szimulációs eredményeket felhasználva próbáljuk elutasítani a H_0 –t a H_1 javára.

A nullhipotézis felállítása után egy olyan **próbafüggvényt kell szerkeszteni**, amely egyrészt a véletlen minta elemeinek függvénye, tehát a valószínűségi változó, másrészt a valószínűségi eloszlása –a nullhipotézis helyességének feltételezése mellett– matematikai úton meghatározható.

A próbafüggvény ismeretében kijelölhető egy olyan intervallum, amely tetszőlegesen nagy valószínűséggel magában foglalja a próbafüggvény értékeit, másképpen: a jellemzők közötti különbségek eloszlása alapján megállapítható annak valószínűsége is, hogy a minták jellemzői közötti különbség meghalad-e bizonyos határértéket. Az intervallum két végpontja a **kritikus érték**, maga az intervallum pedig az **elfogadási tartomány**. Az elfogadási tartomány komplementere az **elutasítási** vagy **kritikus tartomány**.

A kritikus érték kijelölésekor a statisztikai biztonságból kell kiindulni. A normális eloszlás esetében a már ismert szórás-többszörös által megadott határok közé esés valószínűsége a **statisztikai biztonság**. Például, $\pm 1{,}96\sigma$ intervallumon belül található az ismérvértékek 95%-a. Ez azt jelenti, hogy 95% a statisztikai biztonsága annak, hogy az egyes értékek a véletlen érvényesülése következtében nem lépik túl az előbbi intervallumot, és csak 5% a valószínűsége annak, hogy az egyes értékek a $\pm 1{,}96\sigma$ által adott határokon kívül esnek. Az előbbi valószínűség a statisztikai biztonság, az utóbbi pedig **a határvalószínűség** vagy **szignifikanciaszint,** amit p-vel (probabilitás) jelölnek. (A statisztikai biztonság tehát $1-p$**,** amit a próba szintjének is neveznek.)

Az elmondottakat figyelembe véve a nullhipotézist a következőképpen ellenőrizhetjük:

Ha a próbafüggvény számított értéke az elfogadási tartományba esik, akkor a tapasztalati adatok $100(1-p)\%$-os szinten nem mondanak ellent a nullhipotézis fennállásának. Ez másképpen azt jelenti, hogy a **nullhipotézist elfogadjuk**, mivel a próbafüggvény nagy valószínűséggel előforduló értékei az elfogadási tartományba esnek.

Ha a próbafüggvény számított értéke kívül esik az elfogadási tartományon, tehát a kritikus tartományba esik, a **nullhipotézist elvetjük**, mivel a próbafüggvény kicsi valószínűséggel előforduló értékei a kritikus tartományba esnek.

Bárhogy is választjuk meg a kritikus tartományt, elképzelhető, hogy nem döntünk helyesen, azaz elképzelhető, hogy egy valójában helyes hipotézisről úgy döntünk, hogy az nem helyes (**első fajú hiba**), illetve hogy egy valójában helytelen hipotézist elfogadunk (**másodfajú hi-**

ba). Általában előre kikötjük, hogy az elsőfajú hiba elkövetésének mennyi lehet a valószínűsége, és ennek megfelelően választjuk meg a kritikus tartományt. Így, ha a feladatunk az, hogy valamely hipotézis helyes vagy hamis volta felől p szinten döntsünk, akkor ez azt jelenti, hogy a kritikus tartományt úgy kell megválasztani, hogy ha a hipotézis igaz, akkor annak valószínűsége, hogy a minta a kritikus tartományba esik p legyen. Az első fajú hiba elkövetésének valószínűsége tehát a szignifikancia szinttel egyenlő. Ez egyben azt is jelenti, hogy az első fajú hiba elkövetésének valószínűsége tetszőlegesen csökkenthető, hiszen a p érték megválasztása tőlünk függ. A p értékét általában 0,1, 0,05, 0,03, 0,02, 0,01 számok valamelyikének választjuk.

2.5. táblázat

Döntés	A nullhipotézis	
	igaz	hamis
a hipotézis elfogadása	helyes döntés	másod fajú hiba
a hipotézis elvetése	első fajú hiba	helyes döntés

A szignifikancia szintet csökkentve (kisebb lesz az első fajú hiba elkövetésének valószínűsége) az elfogadási tartomány a kritikus tartomány rovására növekszik, tehát megnő annak valószínűsége, hogy elfogadjuk a helytelen nullhipotézist. E döntések és az elkövethető hibák kapcsolatát a *2.5. táblázat* foglalja össze.

Megjegyezzük, a p érték megadása nem definiálja egyértelműen a kritikus tartományt, ezért különböző módszereket, úgynevezett próbákat (u-próba, t-próba, (χ^2-próba, ν^2-próba) használunk az egyes hipotézisek helyes vagy hamis voltának eldöntésére, azaz a kritikus tartomány megválasztására. A különböző típusú nullhipotézisek vizsgálatához kidolgozott próbafüggvények és ezek különböző szignifikancia szintjeihez tartozó kritikus értékek táblázatos formában adottak.

Bár nem minden konfidencia-intervallumot definiálnak hipotézisvizsgálat segítségével, a konfidencia-intervallum és a hipotézisvizsgálat bizonyos értelemben kiegészíti egymást. Egy általános megközelítés szerint a $100(1-p)$ %-os szintű konfidencia-intervallum az a tartomány, ahol nem vetik el a nullhipotézist 100p %-os szinten. Ezért a hipotézisvizsgálattal kapcsolatos feltevések a megbízhatósági intervallumra is átvihetők. Ez a megközelítés azonban feltételezi, hogy rendelkezésre áll egy teszt.

Kényelmes lehet úgy tekinteni, hogy a konfidencia-intervallum a hipotézisvizsgálat elfogadási tartománya, de ezzel vigyázni kell, mert másként is lehet megbízhatósági intervallumot definiálni. Ekkor ez csak közelítőleg érvényes, és az ebből levont következtetések megbízhatósága kérdéses.

2.13. A szimuláció statisztikai problémái

A szimulációs modell eredményein alapuló döntésanalízishez minimálisan szükség van szimulációval becsült mennyiség átlagértékére és annak szórására. Mindkét becslésre hatással vannak a kísérlet feltételei. A modellező által megadott kísérleti feltételek magukban foglalják a szimuláció kezdeti vagy induló állapotát, a statisztikai adatgyűjtés kezdetének időpontját, a futás idejét és az ismétlések számát. Ebben a részben bevezető jelleggel néhány, a feltételek meghatározásával kapcsolatos, megfontolandó szempontot és problémát mutatunk be. A 7.2. alfejezetben ezeket a problémákat részletesebben tárgyaljuk.

2.13.1 Kezdeti feltételek

A szimulációhoz implicite minden szimulációs modell rendelkezik kezdeti feltételekkel vagy kiindulási állapottal. A legegyszerűbb, és valószínűleg a leggyakrabban használt kezdeti állapot az ″üres és a tétlen”, amikor a szimuláció kezdetekor nincs entitás a rendszerben és min-

den szerver tételen állapotban van. Ennek a kezdeti feltételnek a megfelelősége függ a modellezett rendszer természetétől, attól, hogy mi fontos számunkra a rendszer tranziens (átmeneti) vagy a stacionárius[3] (állandó) viselkedése.

Ha az elemzésünk célja a rendszer stacionárius viselkedésének tanulmányozása, akkor sokszor a becsült átlagértéket javíthatjuk azzal, hogy nem üres és tétlen állapotban indítjuk a szimulációt. A kiindulási feltételt megalapozhatjuk egy olyan kezdeti állapot becslésével, amely a rendszer hosszú távú viselkedését reprezentálja. A becsléshez esetleg próbafuttatásból származó kimeneti adatokat lehet megfigyelni. A tranzienselemzésnél a kiindulási feltételeknek a rendszer kezdeti állapotát kell tükrözniük.

2.13.2. Adatcsonkítás

Az adatcsonkítás egy olyan eljárás, amit sűrűn használunk arra, hogy csökkentsük a becsült stacionárius közép kezdeti feltételek okozta eltérését, az által, hogy késleltetjük a statisztikai adatgyűjtését addig, ameddig el nem érjük a „felmelegedési periódus" (warm up period) végét. Ezt megtehetjük egy csonkítási időpont kijelölésével. E időpont előtt keletkező szimulációs adatok értéke nem kerül bele a statisztikai becslésekbe. A törekvés célja a kiindulási feltételek torzító hatásának a csökkentése a szimuláció tranziens periódusában rögzített értékek kizárásával. Azonban az adatok egy részének elhagyásával, mivel a megfigyelések egy részét nem használjuk fel, a becsült középérték szórásnégyzete növekedhet. Ilyen módon, a csonkítással ugyan javítjuk a becsült középérték minőségét, azonban ennek az ára lehet a nagyobb szórás a szimuláció kimenetén.

A legáltalánosabb eljárás a csonkítási pont meghatározására, hogy a kísérleti szimulációs futtatás válaszértékeinek görbéjét megrajzoljuk és megvizsgáljuk. A csonkítási pontot ott kell kijelölni az időben, ahol a válaszértékek közt láthatóvá válik a stacionárius állapot elérése. Léteznek más eljárások is, amelyek törvényszerű formát öltve automatizálják ezt a procedúrát, és amelyek a szimulációs programba beépülve automatikusan határozzák meg a csonkítási pontot a szimuláció végrehajtása során.

2.13.3 A futás hossza és az ismétlések száma

Fontos kísérlettervezési döntés a szimuláció futtatási hossza és az ismétlések száma közötti helyes arány megtalálása. A kevesebb ismétlés és hosszabb futási idő, a több ismétlés és rövidebb futási idővel szemben általában jobb becslést biztosít a stacionárius középértékre, mert a kezdeti eltérés kevesebbszer mutatkozik, és így a csonkítás során kevesebb adatot kell elhagyni. (Ennek magyarázata, hogy a felmelegedési periódus hossza független a futás hosszától.) Azonban a csökkentett számú mintának megfelelő kevesebb válasz növelheti a középérték szórásnégyzetének becsült értékét. Másfelől viszont a több és rövidebb futtatás a gyakrabban ható kiindulási feltételek következtében nagyobb torzulást idézhet elő.

A szimuláció időtartamának meghatározására több lehetséges megoldás ismert. Talán a legegyszerűbb módszer az, amely közvetve annak az időpontnak a meghatározására irányul, amikor a szimulációnak be kell fejeződnie. A hátránya ennek a megoldásnak, az hogy az összegyűjtött minták száma is valószínűségi változó, és a minták száma ismétlésenként különböző lehet. Az első, a minta nagyságát szabályozó eljárás a modellbe belépő entitások számát korlátozza. Ebben az esetben a szimuláció addig tart, amíg a rendszerbe belépő, előírt számú entitás feldolgozása be nem fejeződik. Ilyenkor a szimuláció a rendszer üres és tétlen állapotában áll le. Hasonló, de az előzőtől kicsit eltérő eljárás, hogy azoknak az entitásoknak a számát határozzuk meg, amelyeket a rendszerben teljesen feldolgozunk. Megjegyezzük, hogy

[3] A stacionárius viselkedés nem jelenti a szimulációs válaszok változatosságának hiányát, de e változatosságot leíró, változatlan valószínűségi mechanizmus pontosan meghatározott, amelyre többé nem hatnak a kezdeti feltételek.

ebben az esetben nem feltétlenül üres és tétlen a rendszer abban a pillanatban, amikor a szimuláció leáll. Ennél a megoldásnál ezért meg kell bizonyosodnunk arról, hogy a rendszerben maradó, feldolgozásokra váró entitások reprezentálják-e az eredeti entitás arányokat. Például ez a módszer helytelen akkor, ha feldolgozáskor a legrövidebb feldolgozási idő szerint soroljuk az entitásokat (legrövidebb feldolgozási idő szerinti kiválasztási szabályt alkalmazzuk), ilyenkor, leálláskor a sor végén várakozó, hosszabb feldolgozási időt igénylő entitások maradnak a rendszerben.

Egy másik megközelítés a szimuláció időtartamának szabályozására, az ún. automatikus leállító szabályok használata. Ezek az eljárások automatikusan megfigyelik a szimuláció eredményeit a kijelölt intervallumban a szimuláció futása alatt. Az eljárás a szimulációt akkor állítja le, amikor a becsült középérték szórásnégyzete az előírt határon belül van. Az automatikus leállító szabályok használatával később foglalkozunk bővebben.

Ha az ismétlések száma alapján becsüljük az X kimeneti változó szórásnégyzetét, és ha feltételezzük, hogy X normáleloszlású (ahol X egy jól felvett középérték), akkor ahhoz, hogy $\overline{X}$ az előírt konfidencia-intervallumba essen, a szükséges független szimulációs ismétlések száma a következő formulával számítható:

$$N = \left(\frac{t_{\alpha/2,I-1} S_X}{g} \right)^2$$

ahol:

- $t_{\alpha/2,N-1}$ kritikus érték a t-eloszlás táblázatból $N-1$ szabadságfoknál,
- g az előírt konfidencia-intervallum fél szélessége,
- S_X X valószínűségi változó becsült szórása.

Az N kiszámítására alkalmas formulának a használata feltételezi a t-statisztika ismeretét és kezelését adott $N-1$ szabadságfok és S_x esetén. Ennek lépései: felveszünk N-re egy értéket, végrehajtjuk N ismétléssel a szimulációt, ezekre a futásokra alapozva nyerjük a t és S_x értékét, majd a kapott értékeket a fenti formulába behelyettesítve teszteljük a kezdeti feltételek alkalmasságát, illetve meghatározzuk a további ismétlések szükséges számát.

2.14. Összefoglalás

Ez a fejezet a szimulációs analízishez szükséges statisztikai és valószínűségelméleti alapismeretekkel foglalkozott. A bemutatott fogalmak köre és mélysége nem elegendő a szimuláció széles tárgyköréhez tartozó ismeretek teljes lefedésére. A bevezetőnek szánt anyag ahhoz elegendő, hogy segítséget nyújtson a szimulációs modellezés fogalmainak megértéséhez, és betekintést engedjen a szimulációs analízis természetébe.

3. Bevezetés az Arena használatába

3.1. Bevezetés

Az **Arena** üzleti alkalmazásokhoz fejlesztett modellező és szimulációs eszköz. Az **Arena**-t arra tervezték, hogy az ellátási láncokkal, a gyártási folyamatokkal, a logisztikával (elosztással, raktározással, stb.) és a kiszolgálási rendszerekkel összefüggésben analizálhassuk a változások hatását, beleértve akár a lényeget is érintő komplex újratervezést. Az **Arena** maximális rugalmasságról gondoskodik, és az alkalmazások széles körét fedi le, illetve teszi azokat modellezhetővé a kívánatos szinten, kellő részletességgel és bonyolultsággal.

Tipikus alkalmazások:

A gyártási rendszerek, beleértve az anyagmozgatási rendszereket is, részletes elemzése.
Vevőszolgálati rendszerek és az eladás menedzsment komplex analízise.
Az ellátási láncok (raktározás, szállítás és logisztikai rendszerek) vizsgálata.
A rendszerek működésének előrejelzése a költségek, a teljesítmény, a ciklusidő és a kihasználtság alapján.
A folyamatok szűkkeresztmetszetének azonosítása a kialakuló sorokban és az erőforrások túlterheltségének meghatározása.
A humán erőforrás, az eszköz- és anyagszükséglet tervezése.

Ha az elmondottak alapján még nem ismertük volna fel teljesen, az **Arena** birtokában egy olyan hatékony eszközzel rendelkezünk, ami előnyösen használható akkor, amikor kíváncsian, a jövőt fürkészve, a „*Mi van akkor, ha...?*" kérdésekre keresünk választ.

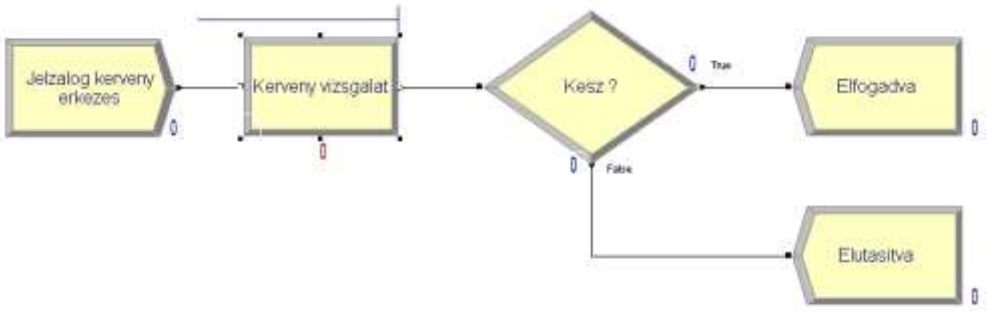
Az **Arena**-val

Modellezhetjük a folyamatainkat annak érdekében, hogy definiáljuk, dokumentáljuk és kommunikáljuk azokat.

Szimulálhatjuk a rendszerünk működését, azért, hogy megértsük a komplex kapcsolatokat és azonosítsuk a fejlesztési lehetőségeinket.

A dinamikus animációval **láthatóvá tehetjük** a rendszerünk működést.

Számtalan lehetséges alternatív konfiguráció esetén **elemezhetjük** a rendszerünk működését, és ezek közül biztonságosan kiválaszthatjuk a legjobbat.



3.1. ábra: A jelzálogkérelem elbírálás folyamatábrája

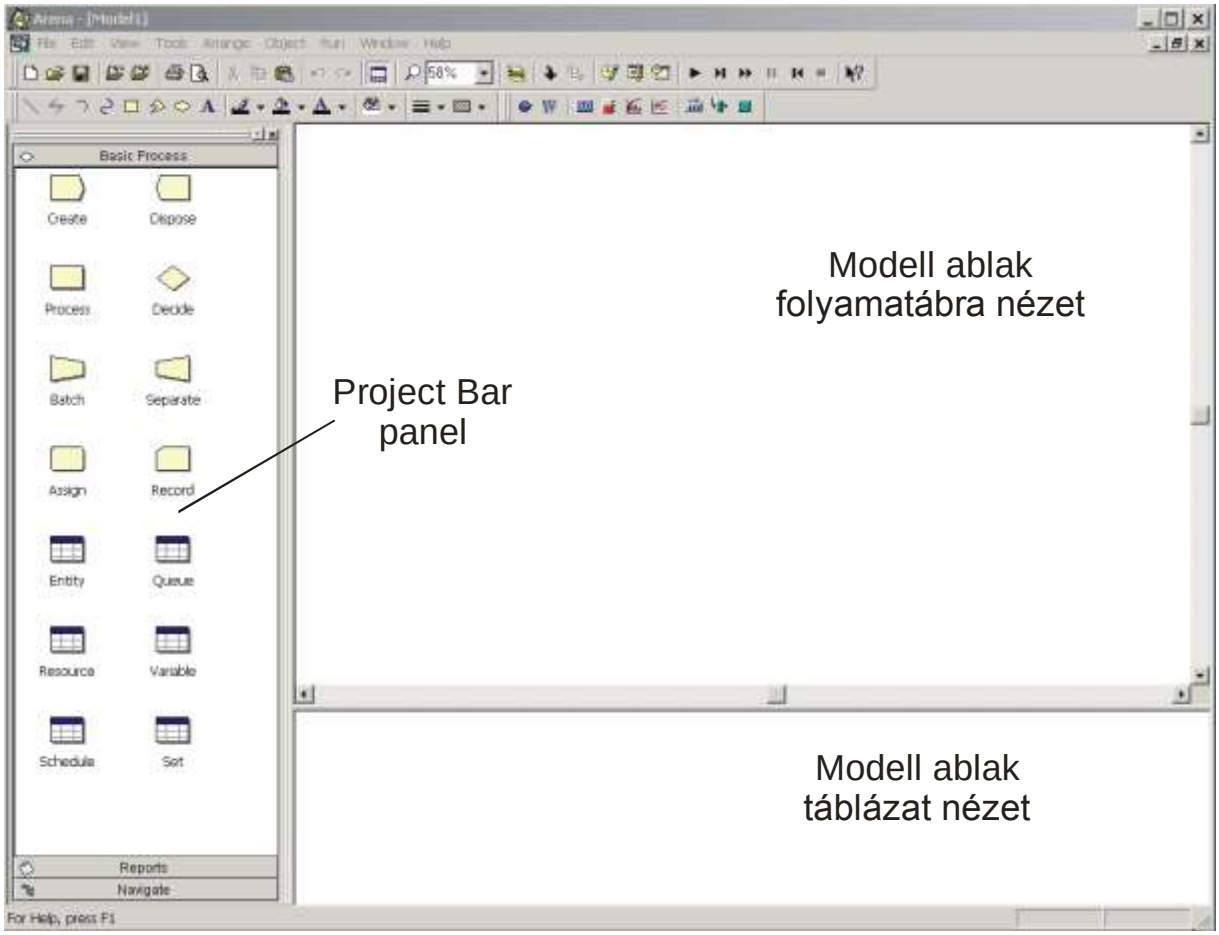
3.2. Jelzálogkérelem elbírálási folyamat elemzése

Ebben a fejezetben, az **Arena** modellező, szimulációs, szemléltető és elemző képességeinek a bemutatására, egy egyszerű jelzálogkérelem elbírálási folyamatot elemzünk. Kezdetnek te-

kintsük a jelzálogkérelem érkezési és elbírálási folyamatát. Bevezetésként a folyamat modellezéséhez és szimulálásához készítünk egy **Arena** folyamatábrát.

3.3. Az Arena modellezési környezet

Indítsuk az **Arena**-t a Start menüből a *Programok>Rockwell Software>Arena* választással. Az **Arena** modellezési környezet új modell ablakkal nyílik meg, amint az a *3.2. ábrán* látható.



3.2. ábra: Az Arena modellezési környezete

Az **Arena**-ban a modellezés során az alkalmazási ablakban három területen dolgozunk. A **Project Bar** panelt az elsődleges objektum típusokkal fogjuk használni:

a **Basic Process,** az **Advanced Process** és az **Advanced Transfer** panelek a folyamatok definiálásához *moduloknak* nevezett modellformákat (alakzatokat) tartalmaznak;

a **Reports** panel lehetővé teszi a szimulációs az eredmények kijelzését;

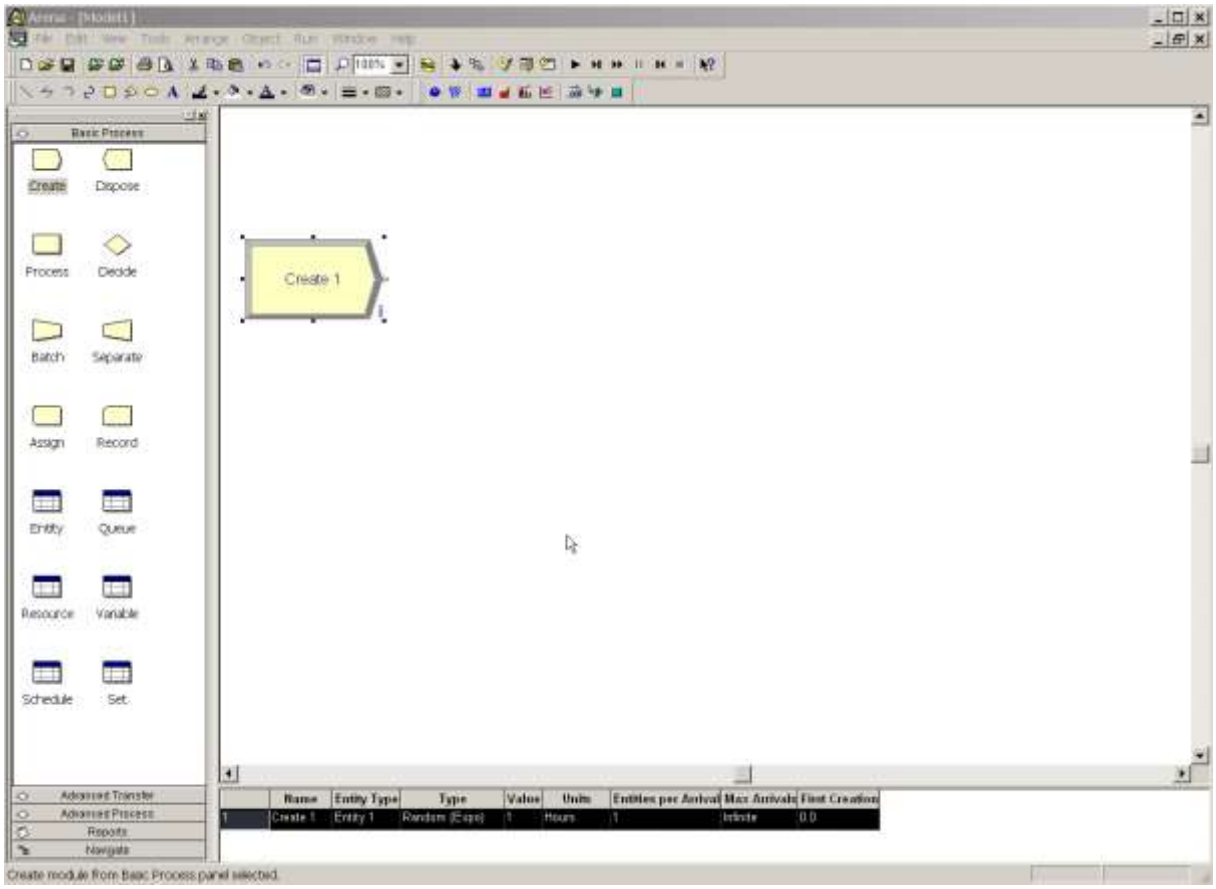
a **Navigate** panel a modell különböző szempontok szerinti megjelenítését segíti, beleértve a hierarchikusan felépülő almodellek közötti navigációt is.

A modellablaknak két fő területe van. A folyamatábra nézet (*flowchart view*) a grafikákat, beleértve a folyamatábrát, az animációt és egyéb rajzi elemeket tartalmazza. Az alsó ablakban a táblázat nézet (*spreadsheet view*) a modelladatokat jeleníti meg táblázatosan, mint pl. az idő, a költség és más paraméterek.

A jelzálog probléma modellezésekor úgy mutatjuk be az **Arena** három fő munkaterületét, hogy közben dolgozunk is ezeken.

3.4. A folyamatok ábrázolása folyamatábrával

A jelzálogkérelem elbírálás folyamatának vizsgálatát, modellezését kezdjük a folyamatábra felépítésével. A folyamatábra-építés kifejezés azt jelenti, hogy felépítjük a folyamat térképét tartalmazó ábrát, amely a folyamat bizonyos elemeinek áramlását ábrázolja vagy írja le. Ez a folyamatmodellezésben felvet egy kulcskérdést, mégpedig azt, hogy pontosan mi is az a bizonyos elem, ami az ábrán keresztül áramlik.



3.3. ábra: A **Create** modul létrehozása a modell ablakban

Esetünkben ezek az elemek a jelzálogkérelmek dokumentumok, amelyeket **entitásoknak** fogunk nevezni. Az entitások azok az elemek, amelyek modellben az egyik helyről a másikra mozognak. A példában az entitások a jelzálogkérelmet benyújtó személyekhez tartozó adatok, amelyek papíron vagy elektronikus formában jelennek meg. Amikor felépítjük a folyamatábrát, hasznos átgondolni a folyamatot és választ keresni a következő kérdésekre:

A jelzálogkérelem hol lép be a folyamatba?
Mi történik vele az egyes lépéseknél?
Milyen erőforrások szükségesek a munka elvégzéséhez, a kérelmek elbírálásához?

Az első, amit lerajzolunk, az a jelzálogkérelmi folyamatot reprezentáló folyamatábra, és amire mint jelzálogkérelem elbírálási folyamatra hivatkozunk.

3.4.1 A jelzálogkérelem entitásainak létrehozása

A folyamatábra szerkesztést a **Basic Process** panelen **Create** modul kiválasztásával kezdjük:

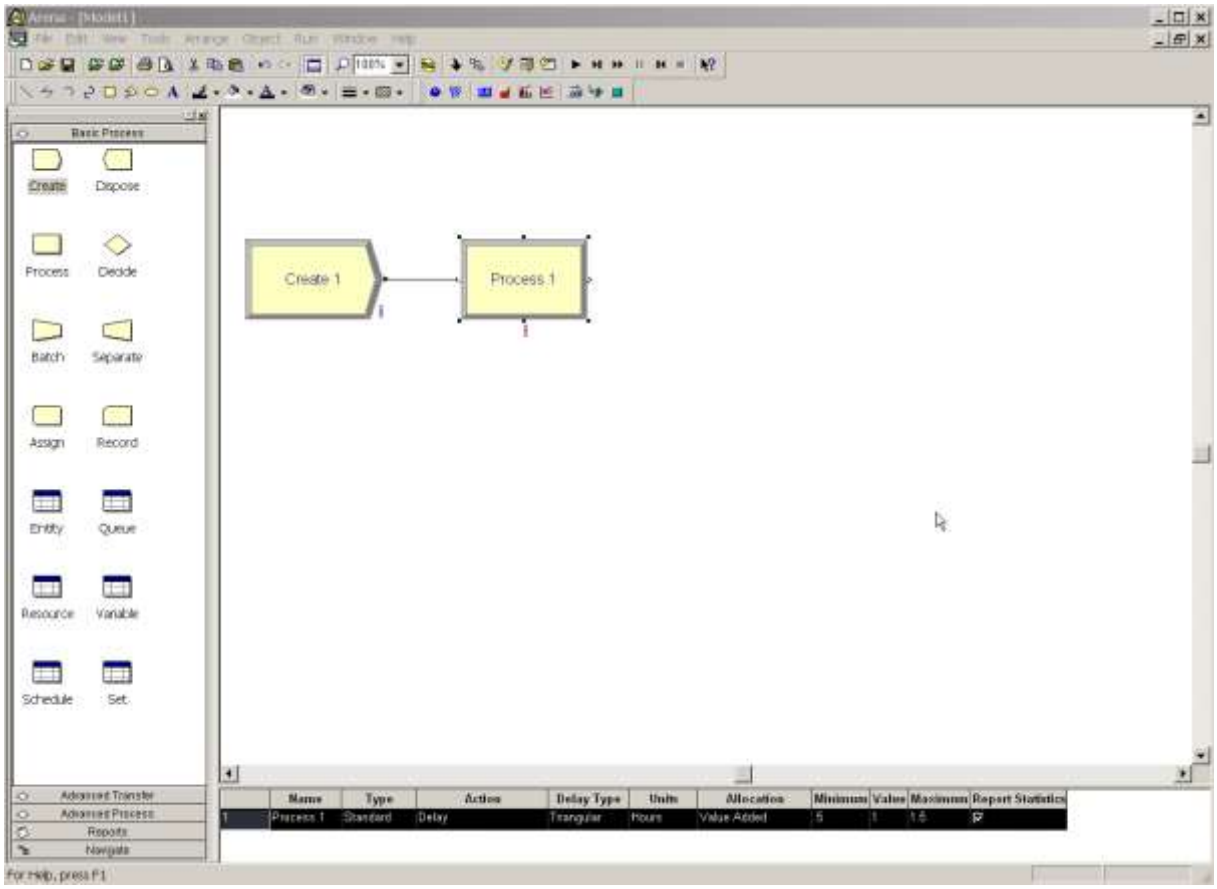
1. Húzzuk át a **Basic Process** panelről a **Create** modult a modell ablakba (*3.3. ábra*).

Az áthelyezéskor a modul alapértelmezés szerinti neve *Create 1* lesz. Később még visszatérünk e kérdéshez, annak érdekében, hogy részletesebben leírjuk a modult és a szimulációt támogató adatokkal is kiegészítsük.

3.4.2 A jelzálogkérelem vizsgálata

A következő modul a folyamatábrában a **Process** modul, amit ugyancsak a **Basic Process panelről** vihetünk a modell ablakba (*3.4. ábra*).

1. Az **Arena** a **Process** és **Create** modulokat automatikusan összekapcsolja, győződjünk meg arról, hogy a **Create** modult ki legyen választva.



3.4. ábra: A **Process** modul és a kapcsolat létrehozása a modell ablakban

2. Húzzuk át a **Basic Process panelről** a **Process** modult a modell ablakba, és helyezzük a **Create** modul jobb oldalára. Az **Arena** a két modult automatikusan összekapcsolja. A **Create** modulhoz hasonlóan a **Process** modul is kap egy alapértelmezett nevet, amit később megváltoztathatunk.

Megjegyzés: ha a kapcsolat a **Create** és a **Process** modulok között nem jön létre, akkor a kapcsolatot megrajzolásához klikkeljünk az *Object>Connect* menü pontokra vagy a **Connect** eszköztár gombra. Ekkor a kurzor szálkeresztre változik. A kapcsolatrajzoláshoz először kattintunk a **Create** modul kilépési pontjára, majd a **Process** modul belépési pontjára.

3.4.3 Hogyan használjuk a Snap és a Grid tulajdonságokat

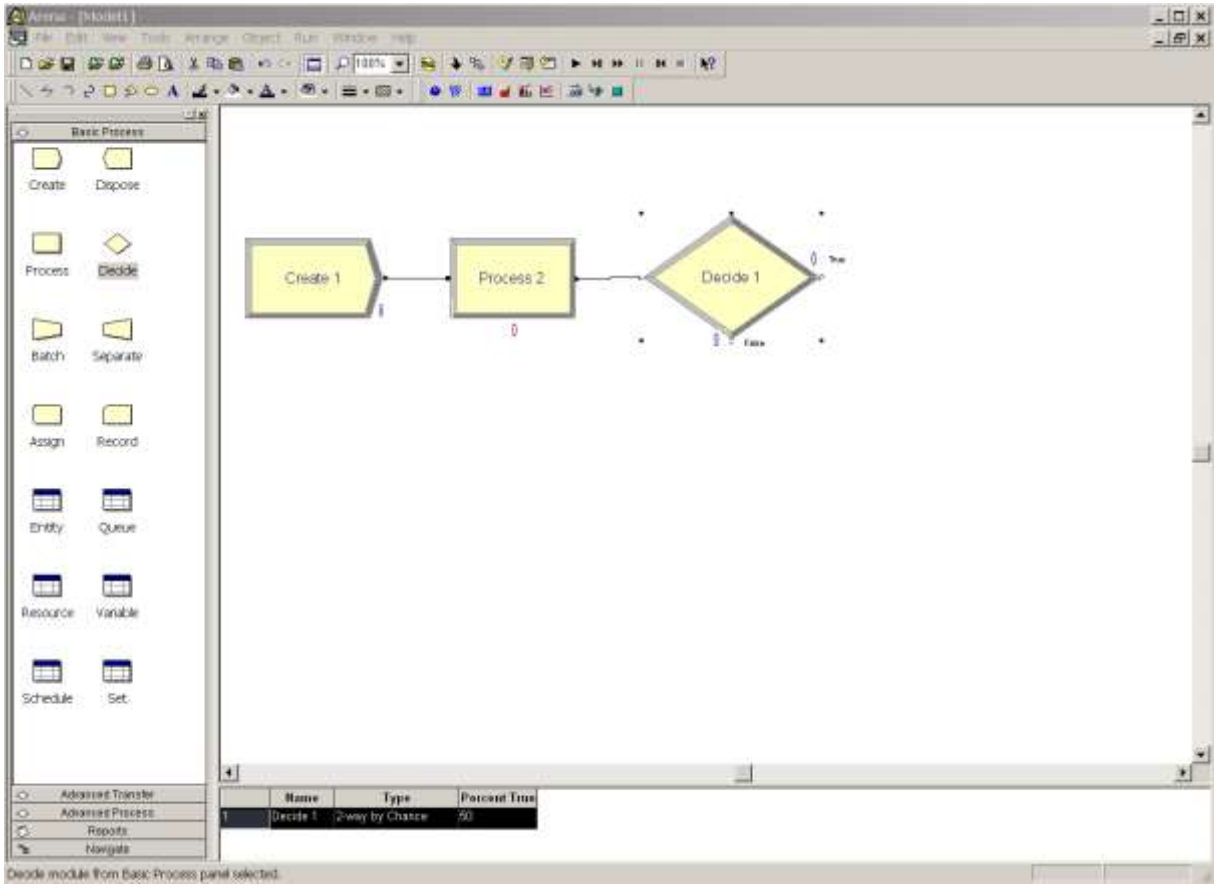
Ha a folyamatábra alakzatai nem esnek egy egyenesbe, használhatjuk a **Snap** és **Grid** (rács) tulajdonságokat a megfelelő illesztéshez. Először kapcsoljuk be a *Snap* opciót a *View* menüben így az újonnan elhelyezett alakzat a meghatározott snap ponthoz igazodik. A megrajzolt alakzatok újrarendezéshez válasszuk ki az alakzatokat a **Ctrl** billentyű lenyomásával és az

alakzatra kattintással, majd állítsuk be pozíciójukat az *Arrange>Snap to Grid* menüpontokkal, amely a rácspontokhoz igazítja azokat.

A rács megjelenik, ha a *View* menüben bekapcsoljuk a *Grid* opciót. A *Snap* és *Grid* opciók ki és bekapcsolhatók.

3.4.4 Döntés arról, hogy a kérelem megfelel vagy sem

A **Process** modul után a **Basic Process** panelről szerkesszük be a **Decide** modult a modellbe, amely majd meghatározza, hogy a kérelem megfelelő-e.



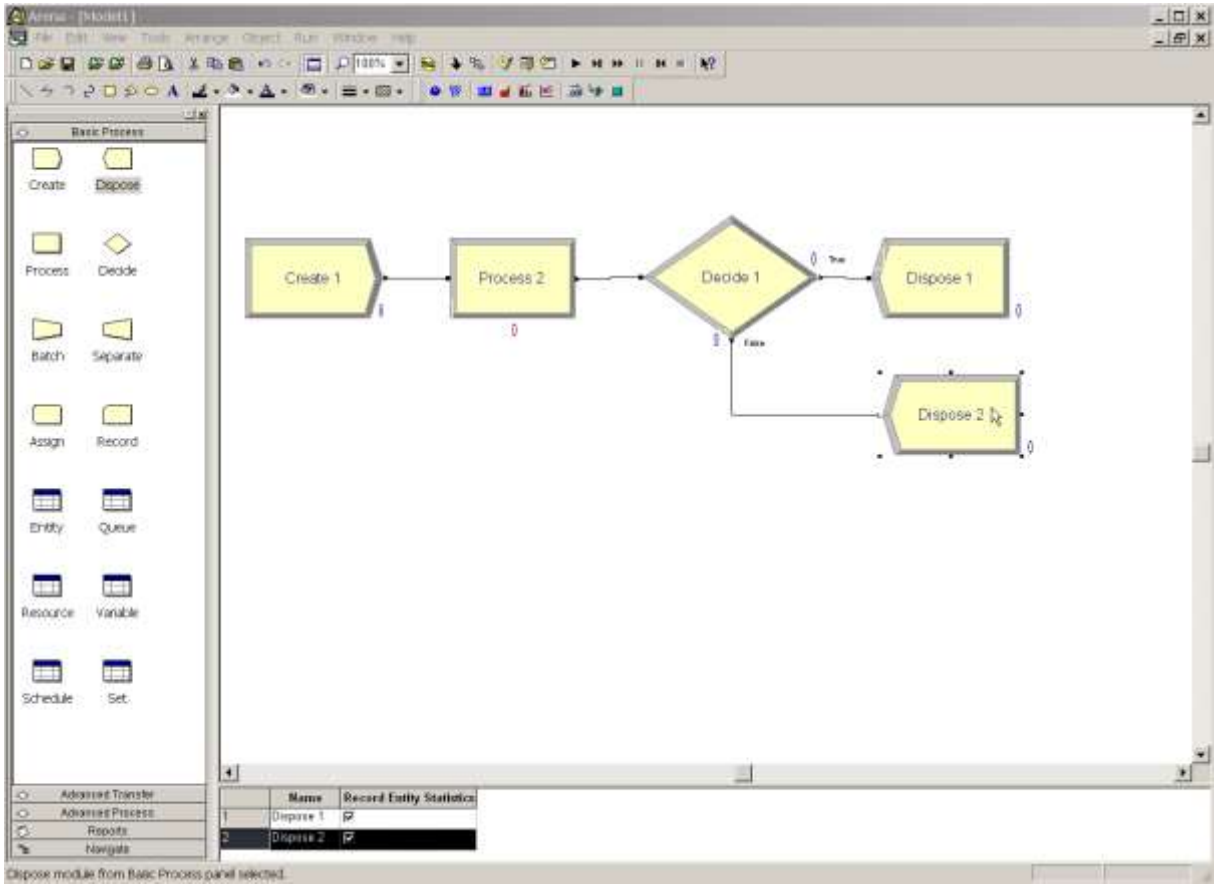
3.5. ábra: A **Decide** modul és a kapcsolat létrehozása a modell ablakban

1. Ha használjuk az **Auto-Connect** tulajdonságot (ez szabályozható az *Object>Auto-Connect* menüben), akkor a kiválasztott **Process** modulhoz a **Decide** modul automatikusan kapcsolódik hozzá.

2. Húzzuk a **Decide** modult a **Process** modul jobb oldalára (*3.5. ábra*). Ha a jelzálogkérelem tartalmazza a szükséges információkat (megfelelően van kitöltve), akkor az a **Decide** modult a deltoid jobb oldalán hagyja el, ez megfelel a true (igaz) feltételnek. A hiányos kérelmek (false eredmény) az alakzat alsó csúcsán távoznak.

3.4.5 A folyamat lezárása a kérelmek diszponálásával

A következő lépés a **Dispose** modul beszerkesztése a **Basic Process** panelről, amelybe az elfogadott kérelmek lépnek be a **Decide** modul jobboldaláról. Aztán befejezzük a folyamatábrát egy másik **Dispose** modullal, ahol az elutasított kérelmek távoznak.



3.6. ábra: A Dispose modulok létrehozása a modell ablakban

1. Jelöljük ki a **Decide** modult, így az első **Dispose** modul automatikusan kapcsolódik.

2. Húzzuk a **Dispose** modult **Decide** modul jobb oldalára. Az **Arena** ezt a **Decide** modult elsődleges kilépési (*True*) pontjához kapcsolja.

3. A második **Dispose** modult áthúzása előtt jelöljük ki újra a **Decide** modult, így azt az **Arena** automatikusan a *False* kilépési ponthoz kapcsolja.

3.4.6 Mi a modul?

Az **Arena**-ban a **modulok** egyrészt a folyamatábra elemei (folyamatábra modulok), másrészt adatobjektumok (adatmodulok), amelyek meghatározzák a szimulált folyamatot. Minden információ, amely az adott folyamat szimulációjához szükséges a modulokban tárolódik.

A folyamatábra szerkesztéséhez, azaz a folyamat leírásához a modulokat. A **Basic Process** panelről a modell ablakra húztuk és azokat összekapcsoltuk. A **Basic Process** panel nyolc alakzatot tartalmaz:

Create: A folyamat kezdete. Az entitás itt lép be a szimulációba.

Dispose: A folyamat vége. Az entitás itt távozik a szimulációból.

Process: Egy tevékenység, rendszerint egy vagy több erőforrással működik, és időt igényel a végrehajtása.

Decide: Elágazás a folyamatban. A modellben csak egy elágazás történik.

Batch: Több entitás összegyűjtése (halmazok képzése), mielőtt a folyamat folytatódna.

Separate: Az entitások duplikálása konkurens vagy párhuzamos feldolgozásra, vagy a korábban képzett entitás halmazok (*batch*-ek) szétválogatása.

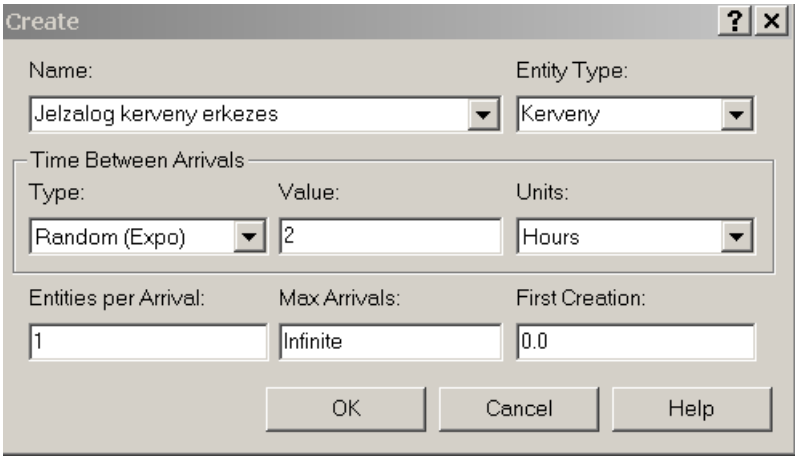
Assign: A paraméterek változtatása (a szimuláció alatt), például az entitás típusa, vagy egy modell változó értéke.

Record: Statisztika gyűjtése, pl.: az entitások száma vagy a ciklus idő.

A szimuláció beállítása a *Run>Setup>Replication Parameters* dialógus ablakban történik. Ezenkívül a különböző folyamatelemek (erőforrások, sorok, stb.) tulajdonságainak definiálására a **Data modules** (adatmodulok*)* szolgálnak.

3.5. A modell adatok definiálása

Miután megszerkesztettük a jelzálogkérelem folyamatábráját, definiáljuk a modulokhoz tartozó adatokat, beleértve a modul nevét, az információkat, amelyek a folyamat szimulációjához szükségesek.



3.1. képernyő: A **Create** modul dialógusablaka

3.5.1 A jelzálogkérelem benyújtása (Create modul)

Először tekintsük a **Create** modult, amelynek a *„Jelzalog kerveny erkezes”* nevet adjuk. (Itt jegyezzük meg, hogy az **Arena** változók, attribútumok, modulnevek stb. nem tartalmazhatnak ékezetes karaktereket. Ez a megszorítás az oka annak, hogy a modulparaméterek magyar neveit a bemutatott modellekben ékezet nélkül írjuk, és a zavar csökkentése érdekében a leírásban e szavakat idézőjelbe foglaljuk.) A **Create** modul adatai tartalmazzák az entitás típusát, esetünkben *„Kerveny”*. Definiálni kell, hogy milyen gyakran nyújtanak be kérelmet. A feladatban átlagosan 2 óránként érkezik egy kérelem, és az időben véletlenszerű érkezést feltételezünk exponenciális eloszlással:

1. A tulajdonság ablak megnyitásához kattintsunk kétszer a **Create** modulra (*3.1. képernyő*).
2. A Name mezőbe gépeljük be: *„Jelzalogkerelem erkezes”*
3. Az entitás típusa (Entity Type) mezőbe: *„Kerveny”*
4. Gépeljünk 2-t az érték mezőbe (Value), amely az érkezések közötti időtartam.
5. Kattintsunk az *OK* gombra, hogy bezárjuk a dialógus ablakot.

3.5.2 Mik az entitások?

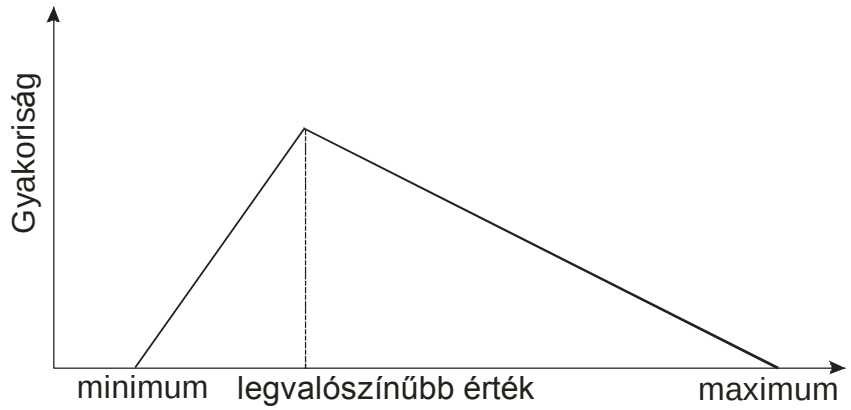
Az entitások a szimulációban áramló objektumok, olyan elemek (vásárló, dokumentum, alkatrész, stb.), amelyeket kiszolgálnak, termelnek, más szóval aktiválnak a folyamatban. Az üzleti folyamatokban az entitások gyakran dokumentumok, elektronikus rekordok (csekkek, szerződések, kérelmek, megrendelések stb.). A kiszolgáló rendszerekben az entitások rendszerint emberek (az ügyfelek az éttermekben, a kórházakban, a repülőtereken stb.). A gyártási model-

lekben meghatározott munkadarabok, részegységek, végtermékek stb. haladnak a folyamatokban. Más modellekben különböző típusú entitások fordulhatnak elő, pl. adatcsomagok, levelek, dobozok, csomagolóeszközök, stb. Ugyanabban a modellben egyidejűleg különböző típusú entitások áramolhatnak. Például a repülőtéren az ellenőrző pontnál szeparálják az utasokat első osztályú, üzleti osztályú és elsődlegességet élvező ügyfelekre. Bizonyos esetekben az entitástípusok nemcsak néhány osztályt alkotnak, hanem teljesen különböző formájúak is lehetnek. Például egy gyógyszertárban az entitásként modellezett receptek kiszolgálási folyamaton mennek keresztül. Ezzel egyidejűleg a vásárlók a gyógyszerészt különböző egészségügyi kérdésekkel is zaklatják, akik szintén entitásként modellezhetőek.

3.5.3 A kérelmek elbírálása (Process modul)

Miután a folyamatábrát már megszerkesztettük, most a folyamatot az entitás szemszögéből vizsgáljuk. A **Create** modul az entitás áramlásának a kezdőpontja a modellezett rendszerben. A következő ponton (**Process** modul) a *jelzálog elbíráló hivatalnok* megvizsgálja a kérelmeket. A modul neve legyen *„Kerveny vizsgalat”*. Mivel a vizsgálat időt igényel, ezért az entitást ezen a ponton feltartjuk, késleltetjük (*delay*), ugyanakkor a vizsgálat végrehajtása erőforrást (*resource*) igényel, amit a **Process** modulban rendelünk a modellhez.

Az időbeni késleltetésre célszerű olyan természetes változékonyságot használni, amely a legtöbb folyamatban létezik. Nagyon gyakran az emberek vagy a gépek által végzett munka időigényének a változékonyságát háromszög eloszlással írják le, amely jól közelíti ezeket a folyamatokat. Meghatározhatjuk azt a minimális időt (*minimum),* amely alatt a munka befejeződhet, a késleltetés legvalószínűbb értékét (*most likely value*), és a folyamat maximális (*maximum*) tartamát. (Az **Arena**-ban elérhető eloszlásokat a *Arena Standard Edition User’s Guide* melléklete ismerteti.)



3.7. ábra: A háromszög eloszlás sűrűségfüggvénye

A szimuláció futása alatt, amikor egy entitás belép a folyamatba az **Arena** vesz (kalkulál) egy mintát a megadott paraméterű eloszlásból, esetünkben a háromszög eloszlásból. Egy hosszú ideig futó szimuláció alatt, amikor több ezer egyedi mintavétel történik, a folyamatok szimulált időtartamai követik a *3.7. ábrán* illusztrált eloszlást.

A kérelem elbírálási folyamatban az általunk használt minimális idő 1 *óra*, a legvalószínűbb érték 1,75 *óra* és a maximum 3 *óra*. Megadjuk az elbírálási folyamat erőforrását (a hivatalnokot), aki végrehajtja a folyamatot. A lépések a következők (*3.2. képernyő*):

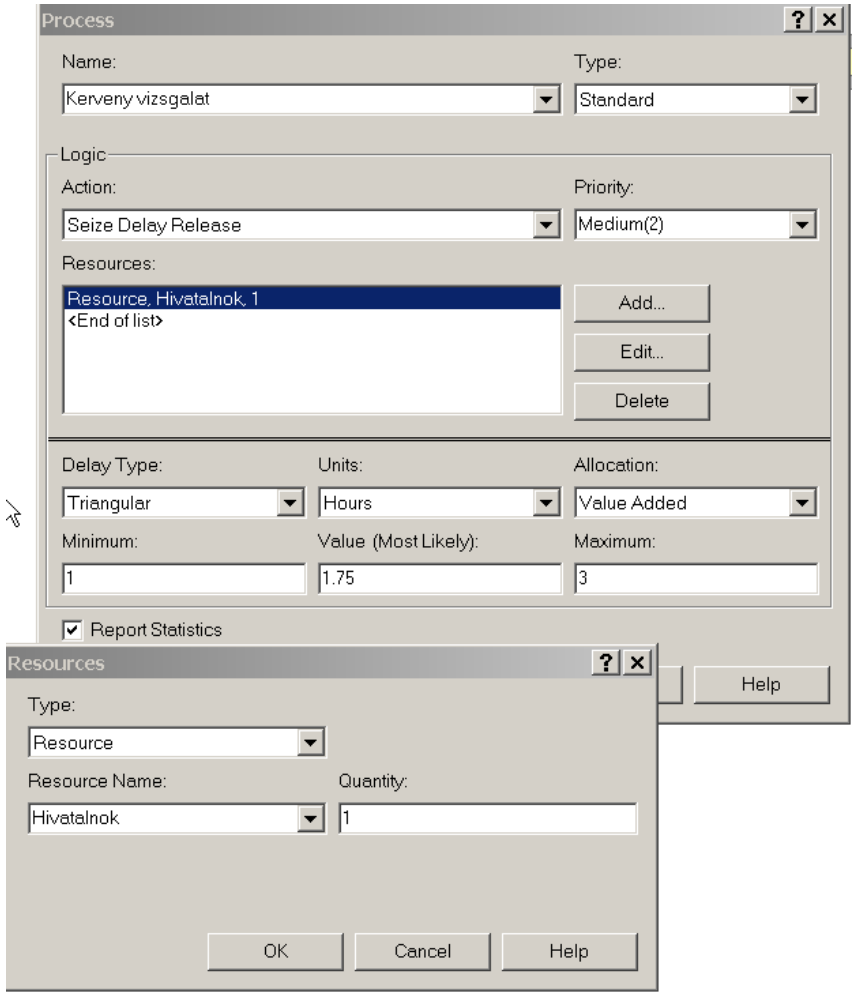
1. A modul tulajdonságait tartalmazó dialógusablak megnyitásához kattintsunk kétszer a **Process** modulra.

2. A Name mezőbe gépeljük be: *„Kerveny vizsgalat”*.

3. A folyamat elvégzéséhez szükséges erőforrás megadásához először nyissuk meg a *Action* (tevékenység) listát és válasszuk a *Seize Delay Release* tevékenység sort**.**

Az érkező entitások addig várakoznak, amíg az erőforrás elérhető lesz számukra. Pontosabban egy kérelem addig várakozik, amíg sorra nem kerül, azaz a hivatalnok feldolgozás céljából kézbe nem veszi. A folyamat három eleme: a kézbevétel (*Seize*), a feldolgozás (*Delay*) és a továbbítás további feldolgozásra (*Release*).

4. A **Resources** (erőforrás) lista a dialógusablak közepén jelenik meg. Az Add gombra kattintva megadhatjuk a folyamat erőforrását, illetve annak tulajdonságait (*3.2. ábra*).



3.2. képernyő: A **Process** modul dialógusablaka

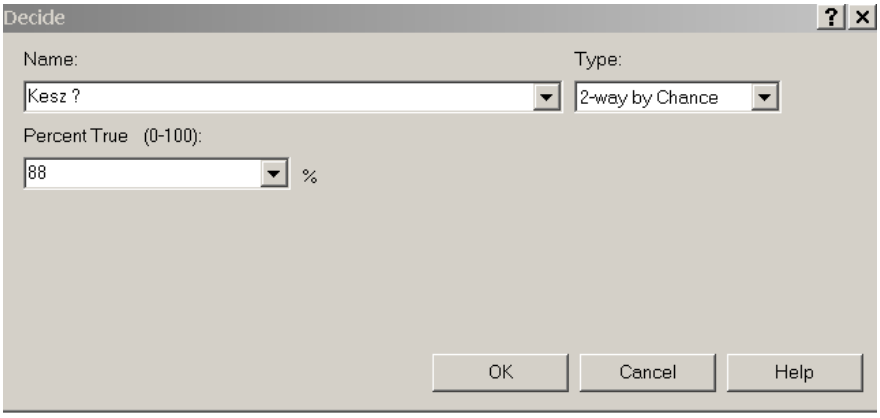
Ha a folyamat elvégzése egyidejűleg egynél több forrást igényel, akkor a Quantity mezőben adjuk meg annyit, amennyire szükség van (*3.2. képernyő*). Az entitás feldolgozása (kiszolgálása) csak akkor kezdődik el, amikor az erőforrás elérhetővé válik.

5. A megnyíló **Resource** dialógus ablakban a Resource Name mezőbe gépeljük be: *„Hivatalnok”*.

6. Az *OK* gombra kattintva zárjuk be az ablakot.

7. Definiáljuk a folyamat paramétereit a Minimum, a Value (Most Likely) és a Maximum mezőkben, ezek rendre: 1, 1,75 és 3. (Megjegyezzük: az alapértelmezett Delay Type a *Trianguláris* és az alapértelmezett Units (időegység) az óra.)

8. Az *OK* gombra kattintva zárjuk be a dialógust.



3.3. képernyő: A **Decide** modul dialógusablaka

3.5.4 Kész ? (Decide modul)

A jelzálogkérelem áttekintése után eldönthetjük, hogy a kérelem elfogadható-e vagy sem. A **Decide** modult akkor használjuk, ha a folyamatban elágazás vagy elágazások találhatók. A döntési modul segítségével kettő vagy több alternatívából választhatunk.

A jelzálogkérelem folyamatban egy egyszerű becsült valószínűséget használunk a döntéshez, amely szerint a kérelmek 88%-a elfogadható, azaz formailag helyes.

1. A dialógusablak megnyitásához kattintsuk kétszer a **Decide** modulra (*3.3. képernyő*).

2. A Name mezőbe írjuk be: *„Kesz ?”*

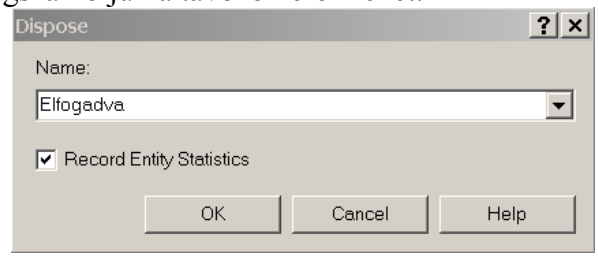
   Amikor ún. két utas (*2-way by Chance*) döntést használunk, az entitás belép **Decide** modulba és valamelyik kimeneti ponton át távozik. Ha az entitásról másolatot akarunk készíteni, azzal a céllal, hogy egy párhuzamos folyamatot modellezzünk, akkor elágaztatásra használjuk a **Separate** modult is.

3. A Percente True mezőbe írjunk *88* %-ot, jelezve, hogy az entitások 88%-a formailag megfelel, azaz „*Kész*”, és ezek a **Decide** modul jobb oldalán távoznak.

4. A dialógust zárjuk be az **OK** gombbal.

3.5.4 Elfogadva, Elutasítva (Dispose modulok)

Az egyszerű jelzálogkérelem példában leírt folyamattal kapcsolatos összes teendőt elvégeztük. A kérelmek most már elhagyhatják a modellt. A folyamat a **Dispose** modullal fejeződik be. Mivel a kimenet két utas, elfogadva vagy elutasítva, két **Dispose** modult használunk, és mind a két kimeneten megszámoljuk a távozó kérelmeket.



3.4. képernyő: A **Dispose** modul dialógusablaka

1. Kattintsunk kétszer a *True* ághoz kapcsolódó **Dispose** modulra, és a dialógusablak (*3.4. képernyő*) megnyílása után írjuk a Name mezőbe: *„Elfogadva”*.

2. Kattintsunk kétszer a másik **Dispose** modulra és a megnyíló dialógusablakban a Name mezőbe írjuk be: *„Elutasítva”*.

3. A dialógust zárjuk be az *OK* gombbal.

3.5.5 Hivatalnok (Resource Data module)

A folyamatábrához kapcsolódó, a modell elemeit (entitások, sorok, erőforrások, stb.) jellemző további paramétereket is definiálhatunk. A jelzálogkérelem folyamatban egy egyszerű, a hivatalnok tevékenységével összefüggő költséget definiálunk, így a szimuláció eredményei között megjelennek a művelet költségei. A hivatalnok költsége legyen állandó, 12$ óránként.

Az **Arena** modellben ezeket a paramétereket adatmodulhoz tartozó táblázatokba (spreadsheet) kell bevinni, esetünkben a **Resource** adatmodulba (*3.5. képernyő*).

1. A **Resource** adatmodul megnyitásához a **Basic Process panelen** kattintsunk a *Resource* ikonra.

2. Mivel a hivatalnokot, mint erőforrást már korábban definiáltuk a folyamatban az **Arena** automatikusan megjeleníti a táblázat első sorában, a Name mezőben a *„Hivatalnok”*-ot. Kattintsunk a Busy/Hour (munkával töltött idő) cellára és adjuk meg a költséget, 12 $/óra. Kattintsunk a Idle/Hour (várakozással eltöltött idő) cellára, és írjuk be a várakozás költségét, ami ugyancsak 12 $/óra.

Resource - Basic Process

	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1	Hivatalnok	Fixed Capacity	1	12	12	0.0		0 rows	☑

Double-click here to add a new row.

3.5. képernyő: A **Resource** adatmodul

Az **Arena** bármely adatmodulját (**Entity, Queue, Resource,** stb.) megjeleníthetjük és szerkeszthetünk. Egyszerűen kattintsunk a megfelelő ikonra a **Basic Process** panelen**,** és a táblázat megjelenik.

3.5.6 A szimuláció előkészítése

A modell szimulációra való teljes előkészítéséhez meg kell adni néhány általános projekt információt és a szimuláció futásának az időtartamát. A modellünk teszteléséhez viszonylag rövid időt 20 napot adunk meg.

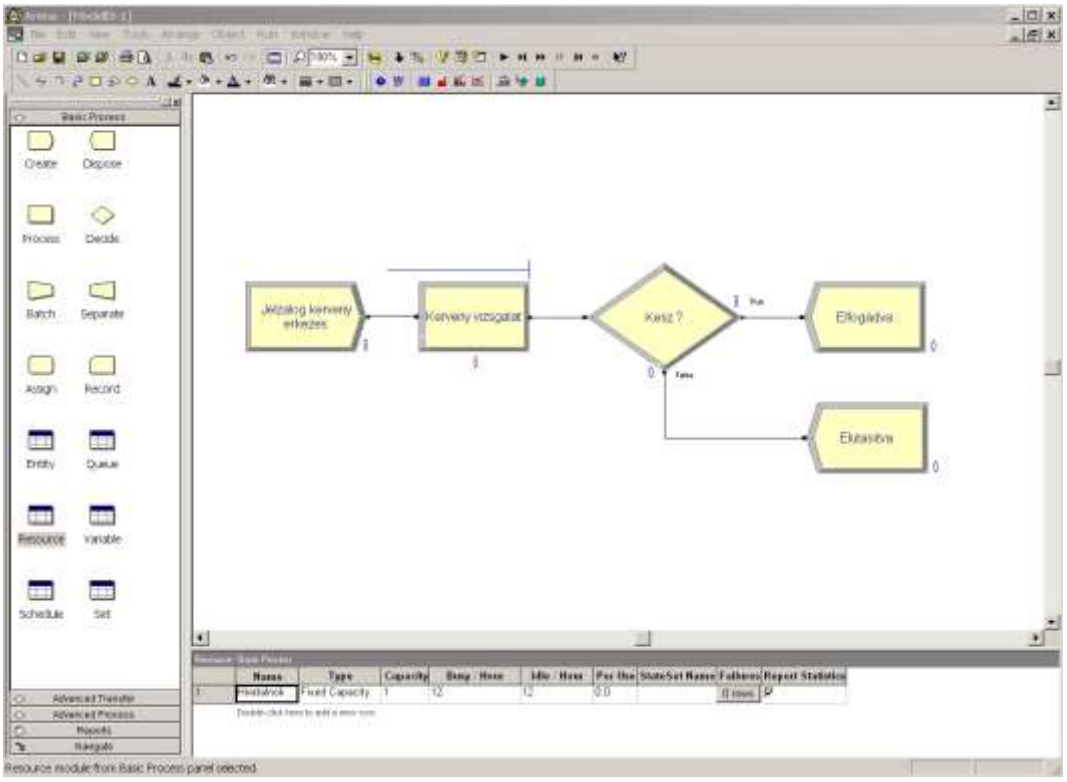
1. A *Run >Setup* menüt használva, a *Project Parameters* fülre kattintva, nyissuk meg a **Project Parameters** dialógust. A Project Title mezőbe írjuk be *„Jelzalog vizsgalat”*, a Statistics Collection ellenőrző dobozait hagyjuk változatlanul (*Costing, Entities, Queues, Resources,* és *Processes* ellenőrizve).

2. A dialógus ablakban a *Replication Parameters* fülre kattintva állítsuk be a paramétereket. A Replication Length (a szimuláció időtartama) mezőbe írjunk 20-at, és közvetlen a Replication Length mezőtől jobbra, a Time Units (időegység) mezőbe a lenyíló menüből válasszuk a *days* (napok) értéket. Az *OK* gombra kattintva zárjuk be a dialógust.

3.5.7 A szimulációs modell mentése

A modell előkészítése befejeződött és a munkánkat elmenthetjük. A modell végleges formáját a *3.8. ábra* mutatja. Kattintsunk a *Save* ikonra a **Standard** eszközsoron, vagy válasszuk a *File>Save* menüpontot. Az **Arena** megnyitja a **Mentés** dialógusablakot, felajánl egy könyvtárat és egy fájl nevet a mentéshez, amit ízlésünk szerint változtathatunk.

Az **Arena** modellfájl a modell valamennyi definícióját tárolja, beleértve a folyamatábrát, a modell adatait és a modellhez kapcsolt grafikákat. A szimuláció futtatása után az eredmények is egy adatbázisba kerülnek ugyanazon a néven, mint a modellfájl.



3.8. ábra: A jelzálogkérelem modell az adatok definiálása után

Ha az **Arena** hibaüzenetet küld, akkor a probléma forrásának lokalizálására használhatjuk a *Find* gombot a hiba ablakban. A *Window* menüben választhatunk a hiba- és a modell ablak között.

3.6. A folyamat szimulációja

Ezzel a néhány lépéssel késszen állunk arra, hogy megjósoljuk a jövőt. A jelzálogkérelem modell tartalmaz minden olyan információt, ami a modell futásához szükséges.

Indítsuk a szimulációt, kattintsunk a *Go* gombra, vagy válaszuk a *Run>Go* menüpontot. Az **Arena** először ellenőrzi az általunk definiált modell helyességét, majd elindítja a szimulációt.

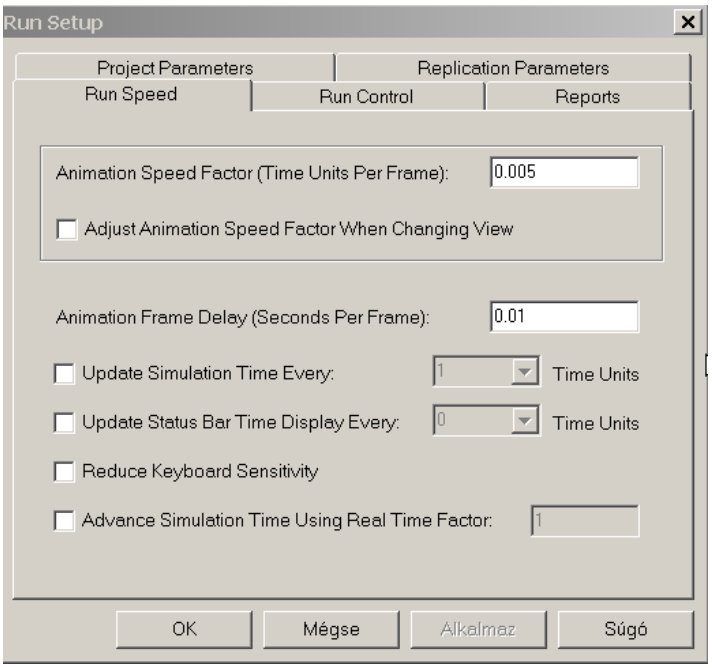
Amíg a szimuláció halad láthatjuk a folyamatábra alakzatai között mozgó entitások képét és ezzel párhuzamosan a modell változói változtatják értéküket, amint az a *3.8. ábrán* is érzékelhető. Ha a szimuláció túl gyors, akkor lassíthatjuk, állítva az animáció sebességét. A *Run> Setup* menüponttal nyissuk meg **a Run Setup** dialógusablakot, majd a **Run Speed** fülhöz tartozó ablakban az *Animation Speed Factor*-nak adjunk kisebb értéket (pl. 0,005) (*3.6. képernyő*). Használhatjuk a *kisebb mint* billentyűt (<) a sebesség csökkentésére. A billentyű egyszeri lenyomása 20%-al csökkenti a futás sebességét. A használat előtt győződjünk meg arról, hogy a modellablak aktív, és nem a **Navigate** panelen állunk. A < billentyűt ismételten lenyomva finoman hangolhatjuk az animáció sebességét. A *nagyobb mint* billentyű (>) lenyomása, fordítva, 20%-al növeli az animáció sebességét. A szimuláció szüneteltetéséhez a *Pause* gombot vagy az *Esc* billentyűt nyomjuk meg.

Az animációs skála (Animation Frame Delay) két egymást követő képernyőfrissítés közötti idő érték. A kisebb értékek finomabb, lassúbb animációt eredményeznek (*3.6. képernyő*).

Az alapértelmezett automatikus folyamatábra animáció megjeleníti a létrehozott entitások számát, a vizsgálat alatt álló entitások számát, a **Decide** modul kimenetein kilépő entitások számát, amelyek a **Dispose** modulok valamelyikén lépnek ki a szimulációból. Ezek a változók a

modell verifikálásában, ellenőrzésében fontos szerepet játszanak.

Lehetőség van a lépésenkénti szimulációra is. A *Pause* gombbal állítsuk meg a szimulációt és kattintsunk a *Step* gombra vagy nyomjuk az *F10* billentyűt. Minden alkalommal, amikor egy lépést végrehajtunk, egy entitás áthalad a folyamatábrán. Rendszerint látható az entitás mozgása, bár néha vizuális változás nem történik (pl. amikor a következő esemény létrehoz egy új entitást). Ha ez a jelenség fordul elő, akkor ismételjük meg a léptetést, hogy elmozduljunk a következő esemény irányába.



3.6. képernyő: A *Run>Setup>Run Speed* beállítása

Ha a futás befejeződik mielőtt kontrolálnánk a folyamatot, akkor az eredmények megtekintésére vonatkozó kérdésre válaszoljunk nemmel, és az újra futtatáshoz kattintsunk a *Start Over* gombra a **Run** eszköztáron.

3.7. A szimulációs riport megtekintése

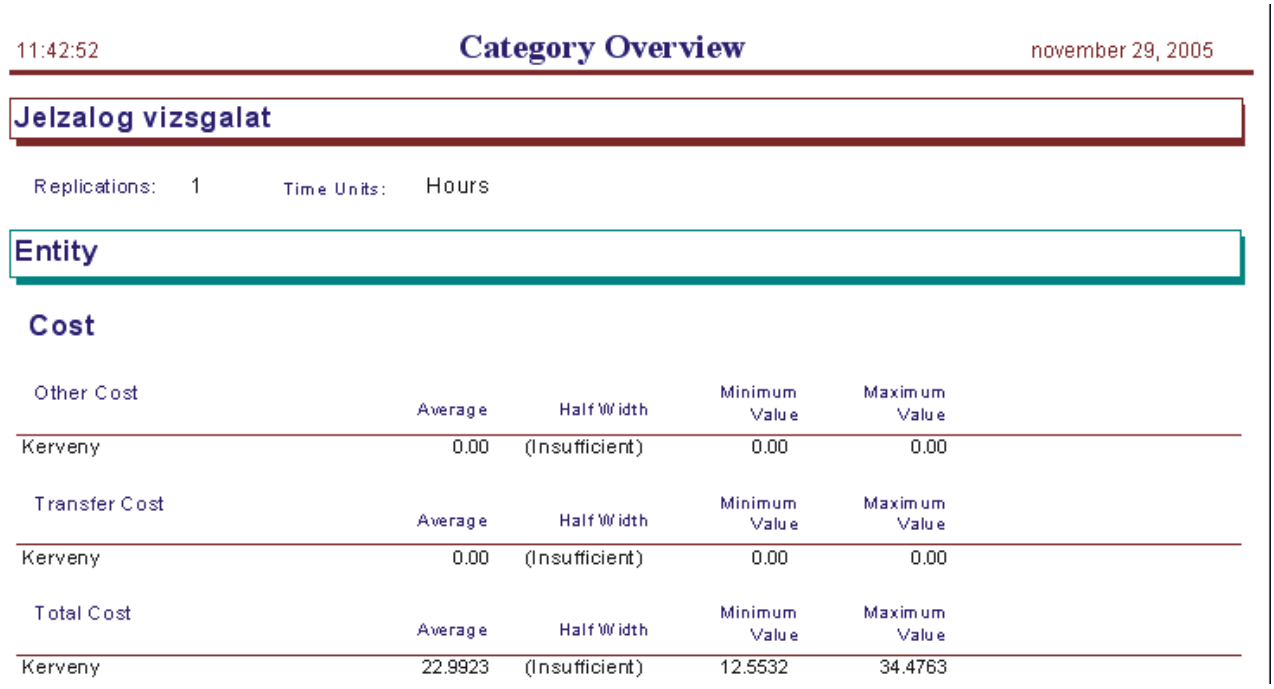
Miután az animáció segítségével megfigyeltük a folyamatot, felgyorsíthatjuk az animációt. A *Pause* gombbal állítsuk meg a szimulációt és kattintsuk a *Fast Forward* gombra. Ekkor a szimuláció a képernyő frissítése nélkül fut.

A futás végén az **Arena** megkérdezi, hogy szeretnénk-e megtekinteni a riportokat. A *Yes* gombra kattintva a képernyőn a riport ablakban az alapértelmezett *Category Overview* című riport látható (*3.9. ábra*).

Az **Arena** alkalmazásban miden riport önálló ablakban jelenik meg. Az ablakokban a kontrol gombra kattintva vagy az ablak menü kinyitásával használhatjuk a standard ablak opciókat (maximize, minimize, stb.).

Minden riportablak baloldalán külön ablakban megjelenik egy fa felépítésű lista a riportokban elérhető információk típusáról. A projekt név (esetünkben *„Jelzalog vizsgalat”*) a fa tetején látható, amit a kategorizált (*Entity, Process, Queue*, stb.) címek követnek. A riport az összes ismétlésre (*replications*) összegzi az eredményeket (bár a modellünkben csak egy ismétlés van). Más riportok bemutatják az ismétlések részleteit is.

A kategória szekciók belépési pontjaira kattintva megnyílnak az ablakok és megtekinthetjük a szimuláció futásának különböző eredményeit. A *3.1. táblázat* a jelzálog vizsgálat szimulációs eredményeinek elemzéséhez néhány kérdést fogalmaz meg, és bemutatja a Category Overview riportban található válaszokat.



11:42:52 **Category Overview** november 29, 2005

Jelzalog vizsgalat

Replications: 1 Time Units: Hours

Entity

Cost

Other Cost	Average	Half Width	Minimum Value	Maximum Value
Kerveny	0.00	(Insufficient)	0.00	0.00

Transfer Cost	Average	Half Width	Minimum Value	Maximum Value
Kerveny	0.00	(Insufficient)	0.00	0.00

Total Cost	Average	Half Width	Minimum Value	Maximum Value
Kerveny	22.9923	(Insufficient)	12.5532	34.4763

3.9. ábra: A Category Overview riport

3.1. táblázat

Kérdés	Riport szekció	Válasz
A jelzálogkérelem átlagosan mennyi időt töltött a modellezett folyamatban	Entity>Total Time >Average oszlop	16.51 óra
A jelzálogkérelmek vizsgálatának mennyi az átlagos költsége	Entity>Total Cost > Average oszlop	$22.99
Mennyi volt a leghosszabb vizsgálati idő	Process>Total Time> Maximum oszlop	33,45 óra
Mennyi volt a vizsgálatra várakozó kérelmek maximális értéke	Queue>Number Waiting> Maximum oszlop	21 kérelem
Mennyi volt a hivatalnok munkával töltött idejének aránya	Resource>Utilization> Average oszlop	97%

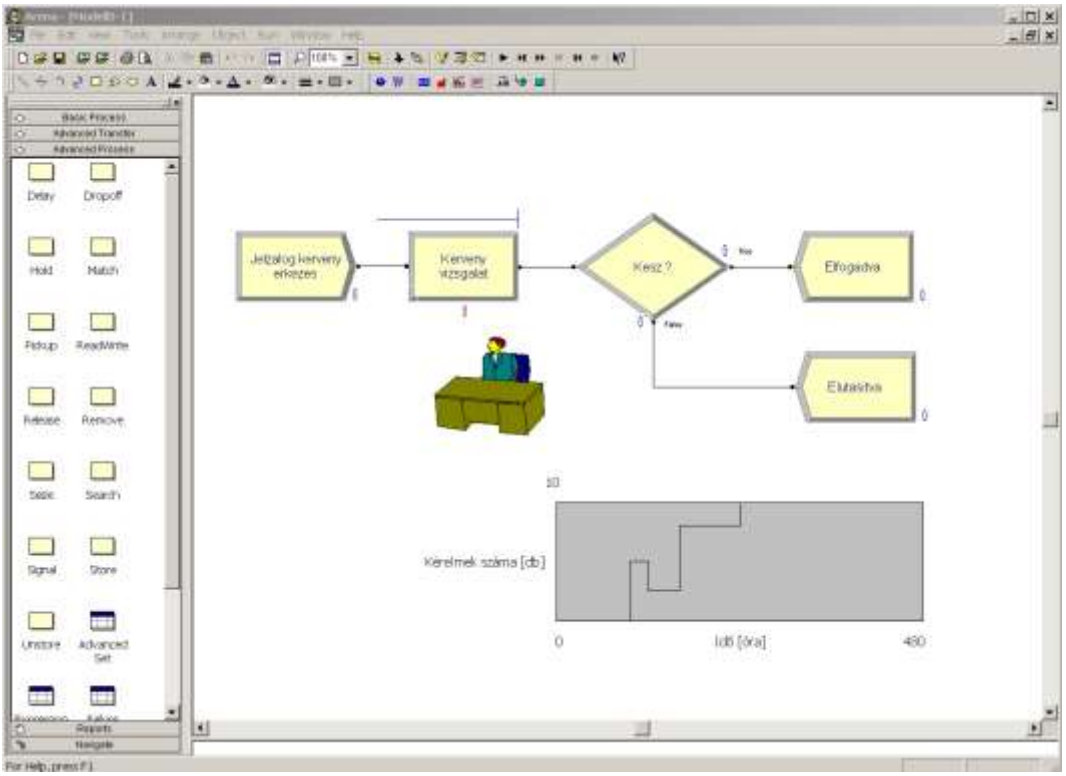
A *Category Overview* riport megtekintése után zárjuk be a riport ablakot. Más riportokat is megnézhetünk, ha a *Projekt Bar*-on a megfelelő ikonra kattintunk. Minden riport a saját ablakában jelenik meg. A modell ablakhoz a riport ablak bezárásával, vagy az *Window* menüben a modell fájl (*„Jelzalog”*) választásával térhetünk vissza.

Miután megtekintettük a riportokat és visszatértünk a modell ablakhoz a *Run>End* menüpont választásával állítsuk le a szimulációt.

3.8. A szimuláció vizuális hatásának növelése

Miután a jelzálogkérelem vizsgálatához a kezdő lépéseket megtettük, visszatérünk a modellhez annak érdekében, hogy feldíszítsük grafikus elemekkel. Az animációnak nagy haszna egyebek mellett, hogy látványosabbá tehetjük a folyamatfejlesztési elképzeléseinket.

A jelzálogvizsgálat modellhez két animációs elemet adunk. Az első a jelzálogkérelmeket vizsgáló hivatalnok, aki egy asztalnál dolgozik, és váltakozva elfoglalt vagy tétlen. Annak érzékeltetéséhez, hogy mennyi kérvény várakozik a folyamatban ún. **WIP** (*work-in-process*) szimulációs változót alkalmazunk, hogy bemutassuk a folyamat dinamikáját. Az **Arena** modellben ezek az animációs elemek *3.10. ábrán* látható módon jelennek meg miután ezeket az objektumokat a modellhez adtuk.



3.10. ábra: A vizuális hatás növelése

A eszköztáron a *Split Screen* gombra kattintva, vagy a *View>Split Screen* menüpontot használva választhatunk a teljes képernyős megjelenítés és a kétablakos (folyamatábra és táblázat nézet egyidejűleg) megjelenítés között. A teljes képernyős módban a **Basic Process** panelon valamely modulra kattintva a modulnak megfelelő nézet, folyamatábra vagy táblázat jelenik meg.

3.8.1 A jelzálogkérelmeket vizsgáló hivatalnok animálása

A szimuláció ideje alatt a kérelmeket vizsgáló hivatalnok kétféle állapotban lehet. Ha nincs kérelem (entitás) a folyamatban, akkor a hivatalnok (resource) tétlen (idle). Ennek az állapotnak a szemléltetésére egy olyan képet használunk, amelyen az asztalnál ülő személy tétlennek látszik (*3.10. ábra*). Amikor egy kérelem érkezik, a hivatalnok állapota megváltozik, lekötötté válik (busy), és ekkor a kép egy olyan személyt mutat, aki a kérelmeket vizsgálja.

1. Kattintsunk a *Resource* gombra az **Animate** eszköztáron.

2. A **Resource** elhelyezés dialógus megjelenik. Válasszuk a jelzálogkérelmet vizsgáló hivatalnokot az Identifier mező lenyíló listájából (*3.11. ábra*). Ez az objektum fogja animálni a hivatalnokot.

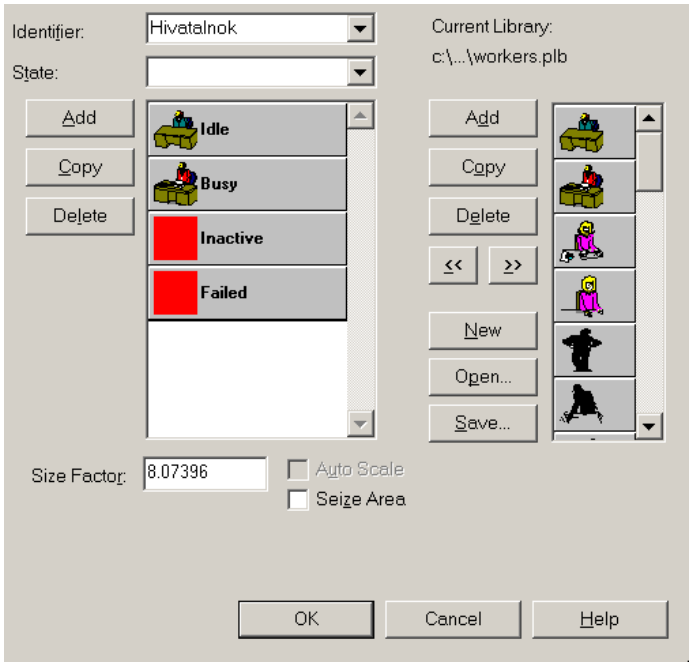
3. Az *Open* gombra kattintva nyissuk meg a Workers picture könyvtárat, azaz keressük meg a Workers.plb fájlt az **Arena** alkalmazás könyvtárban.

4. Változtassuk a tétlen (*Idle*) képet.

Kattintsunk az *Idle* gombra a táblázat baloldalán.

Válasszuk a könyvtár ablakban, a jobboldalon az asztalnál tétlenül ülő embert.

Kattintsunk a *Transfer* gombra (<<) a két tábla között.



3.11. ábra: Resource elhelyezés dialógusablak

5. Változtassuk a foglalt (*Busy*) képet.

   Kattintsunk az *Busy* gombra a táblázat baloldalán.

   A könyvtár ablak jobboldalán válasszuk az asztalnál dokumentumot olvasó embert.

   Kattintsunk a *Transfer* gombra (<<) a két tábla között.

6. Zárjuk be a dialógust, kattintsunk az *OK* gombra.

7. Mozgassuk a hajszálkeresztre változó kurzort a modell ablakban arra a helyre, ahova animációs képet el akarjuk helyezni, majd kattintsunk az egérrel.

8. Ha szeretnénk a képet nagyítani vagy kicsinyíteni, akkor jelöljük ki a képet és használjuk az átméretező eszközt.

3.8.2 A folyamatban mozgó kérvények számának kijelzése

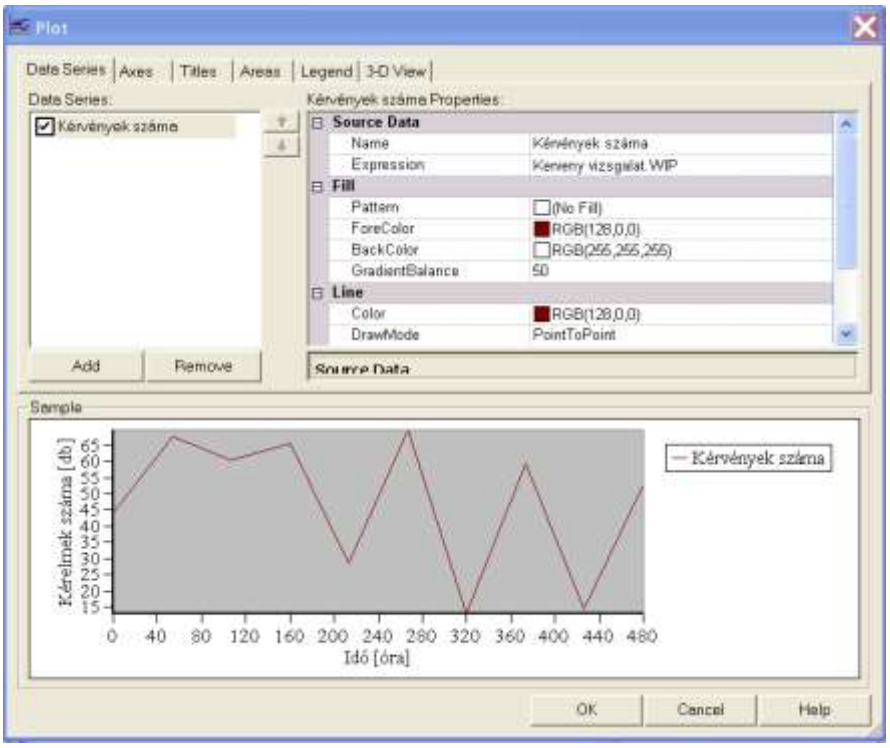
A második animációt fokozó lépésünk annak a diagramnak a szerkesztése, amely azt mutatja, hogy adott pillanatban a szimulációs folyamatban hány kérvény található. A diagram érzékelteti a terhelés dinamikáját.

1. Kattintsunk a *Plot* gombra az **Animate** eszköztáron.

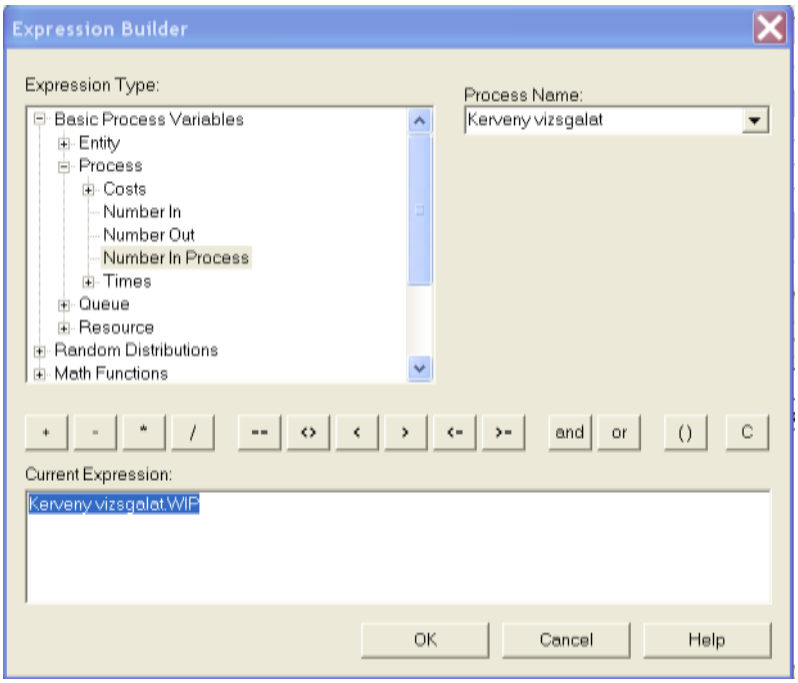
2. A **Plot** dialógus megjelenik *(3.12. ábra***)**. A képernyőn a kérvény vizsgálathoz kapcsolódó kifejezést WIP (*work-in process*) szeretnénk megjeleníteni. A kifejezés hozzáadásához, a *Data Series* fül alatt, kattintsunk az *Add* gombra.

3. A dialógusablak jobboldalán az Expression mezőn lenyíló listájából válasszuk **Build Expression…** opciót, amely megnyitja az **Expression Builder** (kifejezés szerkesztő) párbeszédablakot (*3.13. ábra*).

4. Mivel a vizsgálat előtt és alatt álló entitások számát akarjuk megjeleníteni az idő függvényében, az **Expression Builder** ablakban az Expression Type mezőben válasszuk a *Basic Process Variables> Process> Number in Process* változót, és a Process Name mezőből lenyíló listából a „*Kerveny vizsgalat”*-tot. Kattintsunk az *OK* gombra, hogy bezárjuk az **Expression Builder**-t.



3.12. ábra: A **Plot** dialógusablak



3.13. ábra: Az **Expression Builder** dialógusablak

5. Az *Axes* fül alatt állítsuk be az *x* és *y* tengelyek paramétereit. Beállíthatjuk a tengelyek skálázását (a minimum és a maximum értékeket, a fő és a kisléptéket), nevet adhatunk a tengelyeknek, betűtípust és –nagyságot választhatunk, megadhatjuk a betűk színét. A szi-

muláció futásának hossza 20 nap, ezért az *x* tengely maximális értékét 480 órára választjuk. A korábbi szimulációs futások jelentésiből megtudhatjuk, hogy a kérelmek maximális szám 22 volt, ezért az *y* tengely maximális értékét 25-re állítjuk.

6. A *Titles* fül alatt címet adhatunk a diagramnak, és azt a diagram felett (*Header*), vagy alatt (*Footer*) jeleníthetjük meg.

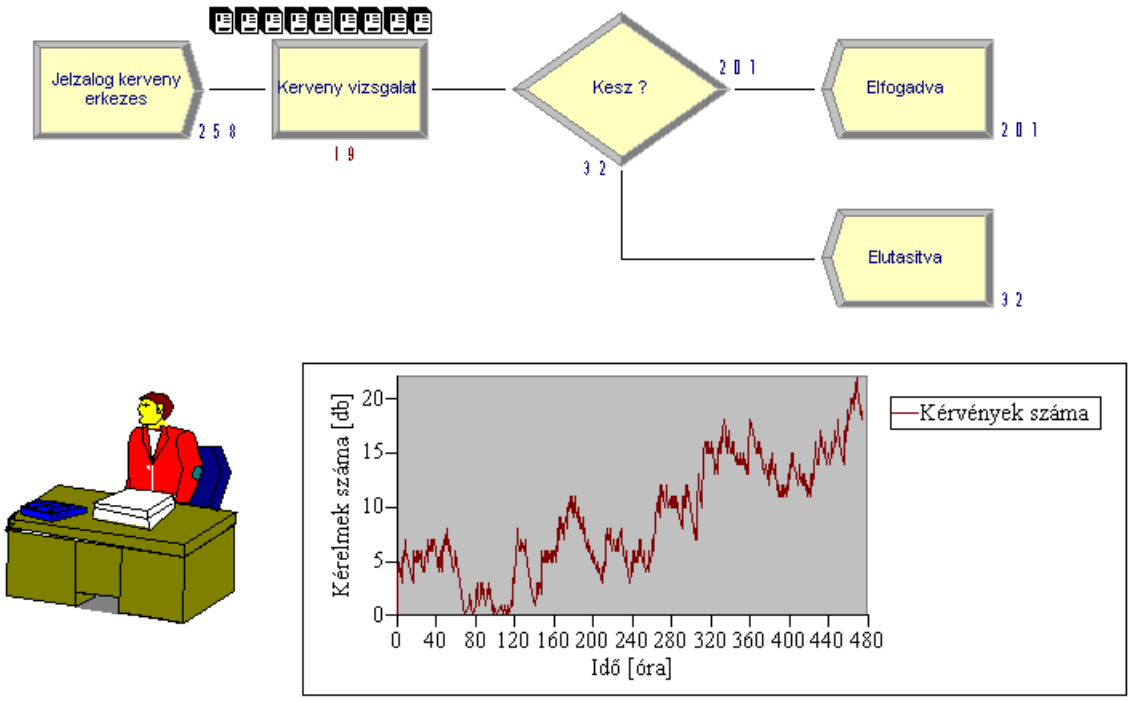
7. Az *Areas* fül alatt a diagram különböző területeinek keret és kitöltési színét változtathatjuk meg.

7. A diagramszerkesztés befejezéséhez, az *OK* gombra kattintva zárjuk be a **Plot** dialógust.

8. A modell ablakban mozgassuk a hajszálkeresztre változó kurzort a folyamatábra alá, a hivatalnoktól jobbra és az egérre kattintva rögzítsük a diagram helyét.

A szerkesztés befejezése után mentsük a modellünket.

A szimuláció futása alatt számos más diagramot is rajzolhatunk, amelyeket a **Plot** dialógus ablakban kell definiálni. A különböző adatokat (terhelés, várakozási idő, stb.) megjelenítő diagramok színét az összehasonlíthatóság érdekében megváltoztathatjuk.



3.14. ábra: A grafikus elemel és diagrammal kiegészített modell

3.8.3 A szimuláció újra futtatása

Az animációs elemekkel kiegészített szimulációs modellt, ami érdekesebb és értékesebb lett, futassuk újra. Mivel a folyamat paramétereit nem változtattuk a szimuláció eredményei sem változnak.

Kattintsunk a *Run* menüpontra vagy nyomjuk meg az *F5* billentyűt. A szimuláció futása alatt érzékelhetjük, hogy a hivatalnok képe folyamatosan változik a tétlen állapotról a dokumentumot olvasó elfoglalt állapotra és fordítva (*3.14. ábra*). A diagram néhány kiemelkedő csúcsot mutat, a **Create** modulban definiált az érkezési folyamat változó érkezési időközeinek, *Exp*(2) és a **Process** modulban definiált kérelem vizsgálat változó időigényének, *Tria*(1, 1,75, 3), pontosabban ezen idők kombinációinak köszönhetően.

4. A modellezés alapműveletei és bemenetei

A 3. fejezetben bemutattunk egy egyszerű jelzálogkérelem vizsgálati rendszert (3.1. modell), és elkészítettük a rendszer **Arena** modelljét. Ebben a fejezetben egy kicsit bonyolultabb és a valósághoz közelebb álló összeszerelő rendszerrel foglalkozunk, amelynek több változatát elemezzük. Létrehozzuk a változatok részletes modelljeit, és ezek komplexitását új modellezési elvek, koncepciók hozzáadásával növeljük. Szót ejtünk arról is, hogyan lehet a tanulmányozás alatt álló rendszer bemeneteit reprezentáló valószínűségi eloszlásokat valósághűen meghatározni.

A 4.1 alfejezetben ismertetjük a modellezés tárgyát képező rendszert, amelynek a feladata légmentesen zárt elektronikus-egységek összeszerelése és minőségellenőrzése. Ezt követően megbeszéljük, hogyan építhető fel a valóságot közelítő modell. A rendszer az eddigieknél egy kicsit bonyolultabb, ezért a modellezése is több problémát vet fel. Az építés során bevezetünk néhány új **Arena** fogalmat, majd megmutatjuk, miként futtassuk a modellt és értékeljük az eredményeket. Remélhetően ezen idő alatt kellően elmélyítjük a modellezés területén szerzett tapasztalatainkat. A 4.2. részben kiegészítjük a modellt az ütemezés, a hiba és az erőforrások állapotának vizsgálatával, és megismerkedünk az alternatív eredmények értékelésének lehetőségeivel. A 4.3 részben megmutatjuk, hogy animáció segítségével hogyan tehető a modell szemléletesebbé. A 4.4. részben általánosítjuk az entitások mozogását, bevezetve a **Stations** (állomások) és **Routes** (utak) fogalmát nem zéró utazási idővel és a szállítások animációját. Végül a 4.5 részben megválaszoljuk azt a kérdést, hogyan határozzuk meg a szimulációt működtető mennyiségi bemeneteket, beleértve valószínűségi eloszlásokat, amelyekből a megfigyelt véletlen változókat generáljuk.

Amikor a fejezetet végére érünk, elvileg képesek leszünk akár bonyolultabb saját modellek felépítésére, csakúgy, mint meghatározni a modellbemenetek alkalmas és valóságos eloszlásait.

4.1.Egy elektronikus-egység szerelő és ellenőrző rendszer (4.1. modell)

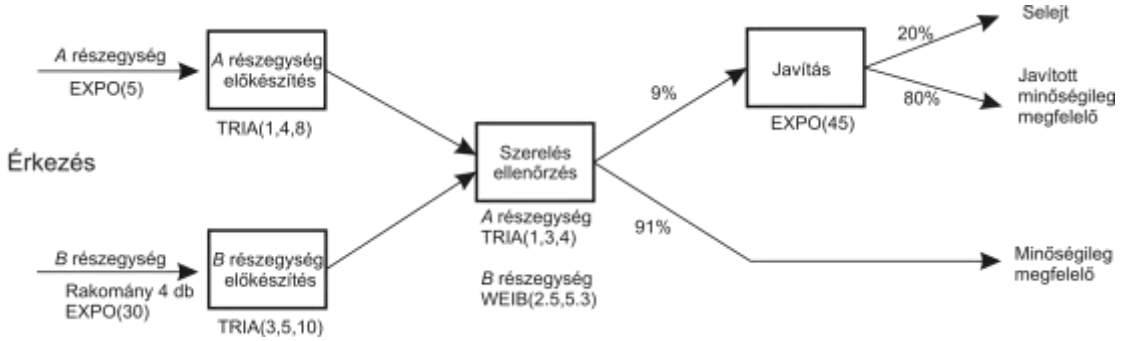
A *4.1. ábrán* látható rendszer két különböző, légmentesen zárt elektronikus-egység összeszerelésének végső műveleteit szemlélteti. Az érkező részegységek (alkatrészcsomagok) öntött fémházak és elektronikus alkatrészek. A gyártási folyamat során az elektronikus alkatrészek beépítik a fémházakba, majd a késztermékeket ellenőrzik.

Az ***A***–nak nevezett részegységek e modell határain kívül, a szomszédos üzemben készülnek, az érkezési időközök exponenciális eloszlásúak, és az átlagos érkezési időköz 5 perc. (Minden idő percben lesz megadva). A részegységek érkezéskor az egységeket azonnal az ***A*** részek előkészítési területére irányítják, ahol a dobozok illeszkedő felületeit megmunkálják, és a jó tömítés, valamint illeszkedés érdekében a felületeket csiszolják és tisztítják. Az ***A*** részek előkészítési területén végzett összetett művelet ideje a TRIA(1,4,8) eloszlást követi. Az ***A*** részeket a megmunkálás után az összeszerelő területre szállítják (azonnal és folyamatosan).

A ***B*** –nek nevezett részegységeket ugyancsak e modell határain kívül, más épületben gyártják. Innen 4 darabos tételekben (batch) szállítják a szerelési területre (a modellezés területére) és tárolják. Az adagokban érkező ***B*** részegységek érkezési időközei is exponenciális eloszlást mutatnak, és az átlagos érkezési időköz 30 perc. A ***B*** részegységek előkészítési területére érkező tételeket 4 egyedi darabra választják szét, és a továbbiakban ezeket egyenként kezelik. Az egyedileg kezelt részek (azonnal) tovább haladnak az előkészítő területre. A ***B*** részegységek előkészítő területén az előkészítés ugyanazokból műveletből áll, mint az ***A*** részegységek előkészítése, és az összetett eljárás műveleti ideje is trianguláris eloszlású, de az eloszlás pa-

raméterei különböznek: TRIA(3,5,10). A ***B*** részegység az előkészítő műveletek elvégzését követően az összeszerelő területre kerül (azonnal).

Az összeszerelés során az elektronikus részeket beépítik a házakba, majd légmentesen lezárják, és végül a lezárt egységeket ellenőrzik. Az eljárás teljes műveleti ideje a részegység típusától függ: az ***A*** rész műveleti ideje TRIA(1,3,4) eloszlású, míg a ***B*** rész műveleti ideje WIEB(2,5, 5,3) eloszlású, ahol a léptékparaméter: β=2,5; az alakparaméter: α=5,3, (lásd a függelékben). A tapasztalat szerint az összeszerelt termékek 91%-a megfelel a minőségi előírásoknak, és ezek azonnal a szállítási osztályra kerülnek. A minőségileg nem megfelelő egységeket pedig a javítási területre irányítják, ahol az egységeket szétszerelés után javítják, tisztítják, összeszerelik és újra tesztelik. A javított egységek 80%-a már megfelelő minőségű, és a szállítási osztályra továbbítható, a maradék 20%-ot selejtezik. Egy egység javításának átlagos időigénye 45 perc, a javítási idő exponenciális eloszlást mutat, és független az egység típusától, valamint a javítás eredményétől (megfelelő vagy selejtes).



4.1. ábra: Elektronikus-egység szerelő és ellenőrző rendszer

Minden egyes területen szeretnénk különböző statisztikákat gyűjteni az erőforrások kihasználásáról, a sorok hosszáról, a sorbanállási időről, a minőségileg megfelelő, a javított minőségileg megfelelő és a selejtes egységek ciklusidejéről külön-külön. Kezdetben négy egymást követő nyolc órás műszakot, azaz 1920 percet szimulálunk.

4.1.1 A modell fejlesztése

A szimulációs modell építése csak egy része a teljes modellezési feladatnak. A teljes modellezési feladatot később részletezzük. Kezdetben a tanulmány céljainak meghatározására, a tanulmányozott rendszer definiálására koncentrálunk, továbbá célunk megtanítani a szimulációs modellek fejlesztésének fogásait **Arena** környezetben. A rendszer definíciókat már korábban megadtuk. Valós problémák esetén saját magunknak kell megfogalmazni ezeket a definíciókat, valamint összegyűjteni és elemezni az adatokat, és így meghatározni a bemenő paramétereket, valamint eloszlásokat (lásd még 4.5 alfejezetben). A fejlesztés következő lépése a modellezési eljárás kialakítása. Egy valós probléma az adatstruktúra definiálását, a rendszer alrendszerekre osztását, illetve az irányítási logika kifejlesztését igényelheti. A probléma kezeléséhez azt szükséges eldönteni, mely **Arena** modulok fogják kielégíteni az általunk elvárt, a modell megfelelő szintű és részletességű működését garantáló igényeket. Ki kell találni azt is, hogyan fogjuk modellezni az ***A*** és ***B*** részek különböző műveleti idejét az összeszerelés során. Annak érdekében, hogy leegyszerűsítsük, és áttekinthetővé tegyük a feladatot, osszuk fel a modellt a következő részekre: részegységek érkezése, előkészítési területek, összeszerelési műveletek, javítás (újramunkálás), egységek távozása és egységek animálása. A rendszerben az entitások reprezentálják a megmunkálandó önálló ***A*** és ***B*** részegységeket.

Mivel az entitások két helyről és különböző intenzitással érkeznek a modellbe, két **Create** modult fogunk használni az érkező részegységek generálásához. Az egységek típusától (***A*** és ***B*** részegységek) függően az összeszerelési idő különböző, ezért két **Assign** modult is beillesz-

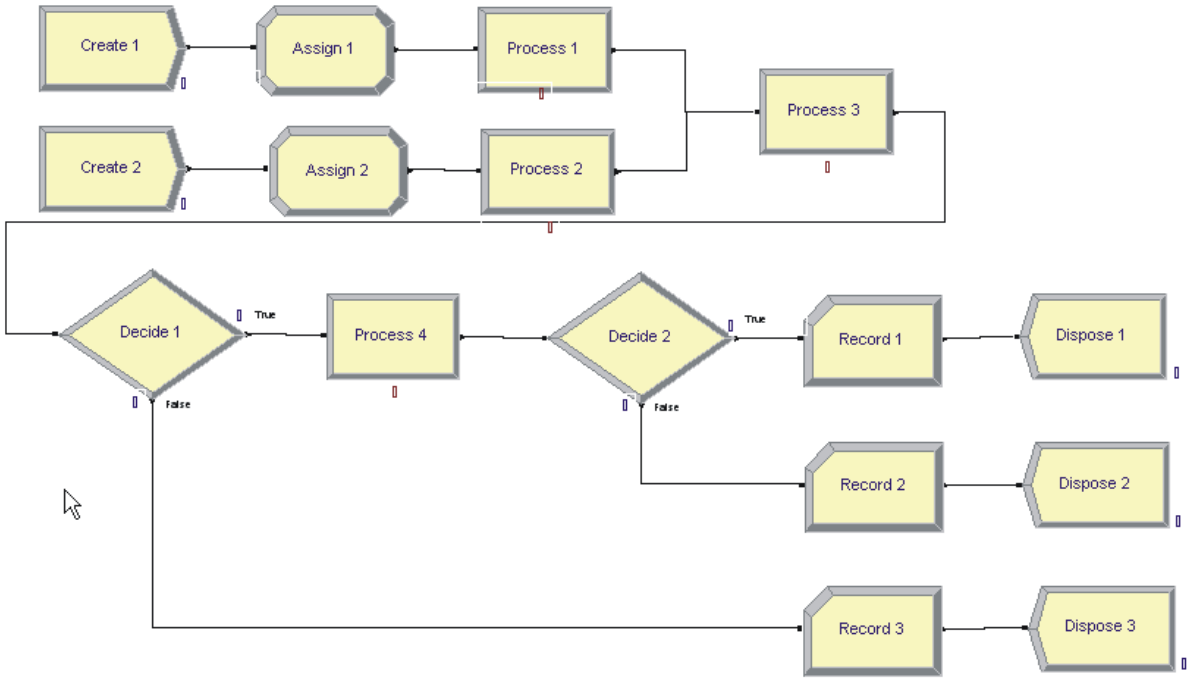
tünk, amelyekben a részegységekhez rendeljük a *„Szerelesi idonek”* nevezett attribútumot (tulajdonságot), amit majd később a szerelési és ellenőrzési műveletek során fogunk felhasználni. Ez azt jelenti, hogy a szerelés és ellenőrzés műveleti idejét nem a *„Szereles minoseg-ellenorzes”*-nek nevezett **Process** modulban definiáljuk, hanem korábban, közvetlen az érkezés után az **Assign** modulokban *(„Hozzarendeles az A egyseghez”, „Hozzarendeles az B egyseghez”*).

A két előkészítési területen végzett tevékenységet, és az összeszerelést (szerelés + minőség-ellenőrzés) különálló **Process** modulokkal modellezzük. A minőségellenőrzést követően a kész elektronikus-egységek a minőség-ellenőrzés eredményétől függően (megfelelő vagy hibás) különböző helyekre áramlanak. Az elágazást a **Decide** modul biztosítja. A modulban a döntéshez egy egyszerű valószínűséget használunk, amelynek a hibás (*True*) ágán az elektronikus-egységek 9%-a, a minőségileg megfelelő (*False*) ágán pedig 91%-a távozik. A javítási folyamatot, amely a minőségellenőrzést is magában foglalja, egy **Process** és **Decide** modullal modellezzük. Az elektronikus-egységek három irányban távozhatnak, minőségileg megfelelő, javított minőségileg megfelelő és selejt. Ezeket három **Record** és három **Dispose** modullal modellezzük.

Az említett modulok a **Basic Process** panelen érhetők el, és biztosítják azokat a statisztikákat, amelyeket a korábban igényként megfogalmaztunk.

4.1.2 Modellépítés

A modellépítéshez indítsuk el az **Arena** alkalmazást, vagy ha már fut, akkor a *File>New* menüponttal nyissunk új modellablakot, majd a **Basic Process** panelről a szükséges modulokat: két **Create**, két **Assign**, négy **Process**, két **Decide**, három **Record** és három **Dispose** modult helyezzük a modellablakba.

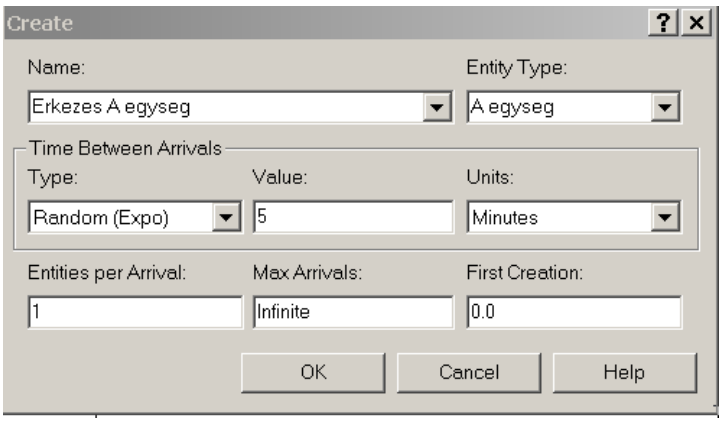


4.2. ábra:. A modellablak a modulokkal

A műveletek elvégzése után a modellablak a *4.2. ábrához* hasonlóan néz ki, feltéve, hogy mialatt a modulokat a megfelelő sorrendben elhelyeztük, az *Auto-Connect* tulajdonsággal

(*Object* menü) létrehoztuk a kapcsolatokat, vagy az *Object>Connect* menüpontot használva utólag összekapcsoltuk a modulokat. A *File>Save* menüfunkcióval elmenthetjük a modellünket.

A következő lépés a modulok működéséhez szükséges információk megadása, ehhez nyissuk meg a modulokat egyenként. Kezdjünk a *Create 1*-es modullal, ami létrehozza az ***A*** *egyseg* entitást. Kattintsuk kétszer a *Create 1* modulra. A *4.1. képernyőn* látható, megnyíló dialógusablakban adhatjuk meg azokat az információk, amelyek modul működéséhez szükségesek. Átnevezhetjük a modult, megadhatjuk az entitás érkezési időközeit jellemző eloszlás típusát, ami esetünkben exponenciális eloszlás 5 perces átlagos érkezési időközzel. Az időegység legyen perc, a többi paramétert változatlanul hagyjuk. A modul paramétereit a *4.1. táblázat* foglalja össze.



4.1. képernyő. Dialógusablak az ***A*** részegység létrehozásához

4.1. táblázat

A *Create 1* modul paraméterei

Name	Erkezes A egyseg
Entity Type	A egyseg
Type Value Units	Random(Expo) 5 Minutes

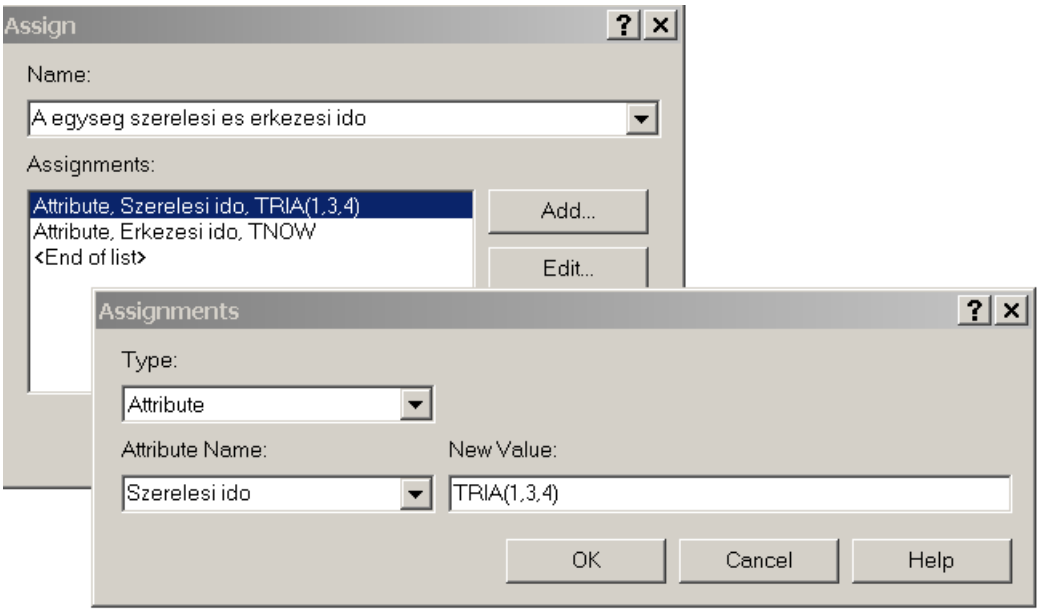
Nyissuk meg a *Create 2* modult, és az előzőekhez hasonlóan írjuk be a *4.2. táblázatban* megadott paramétereket a dialógus ablakba. A ***B*** részegységek 4 db-os bath-ben érkeznek, ezért az Entities per Arrival mezőbe 4-et írunk.

4.2. táblázat

A *Create 2* modul paraméterei

Name	Erkezes B egyseg
Entity Type	B egyseg
Type Value Units	Random(Expo) 30 Minutes
Entities per Arrival	4

Az érkező részegységek létrehozása után az **Assign** modulokban definiáljuk a szerelési és az érkezési időt, amelyek az ***A*** és ***B*** részegységeknél különbözőek. Az ***A*** részegységhez tartozó hozzárendelés dialógusablaka a *4.2. képernyőn* látható, illetve az *Assign 1* dialógusablakba beírandó paramétereket a *4.3. táblázat* tartalmazza.



4.2. képernyő. A szerelési és az érkezési idő hozzárendelése az ***A*** részegységhez

4.3. táblázat

Az *Assign 1* modul paraméterei

Name	A egyseg szerelesi es erkezesi ido
Type Attribute Name New Value	Attribute Szerelesi ido TRIA(1,3,4)
Type Attribute Name New Value	Attribute Erkezesi ido TNOW

A szerelési és ellenőrzési időnek megfelelő *Szerelesi ido* attribútum trianguláris eloszlású. A másik attribútum az *Erkezesi ido*, ami az entitások érkezési idejét jellemzi. Ennek értékét a TNOW **Arena** változó határozza meg. A TNOW értéke a pillanatnyi szimulációs idő, ami esetünkben a részegység érkezési ideje.

4.4. táblázat

Az *Assign 2* modul paraméterei

Name	B egyseg szerelesi es erkezesi ido
Type Attribute Name New Value	Attribute Szerelesi ido WEIB(2.5,5.3)
Type Attribute Name New Value	Attribute Erkezesi ido TNOW

A ***B*** részegységhez rendelt attribútumokat a *4.4. táblázat* foglalja össze. Bár a *Create 2* modulban minden érkezés alkalmával négy entitást hoztunk létre, az *Assign 2* modulban mindegyikhez külön rendeljük hozzá a *„Szerelesi ido”* eloszlásából a különböző értékeket.

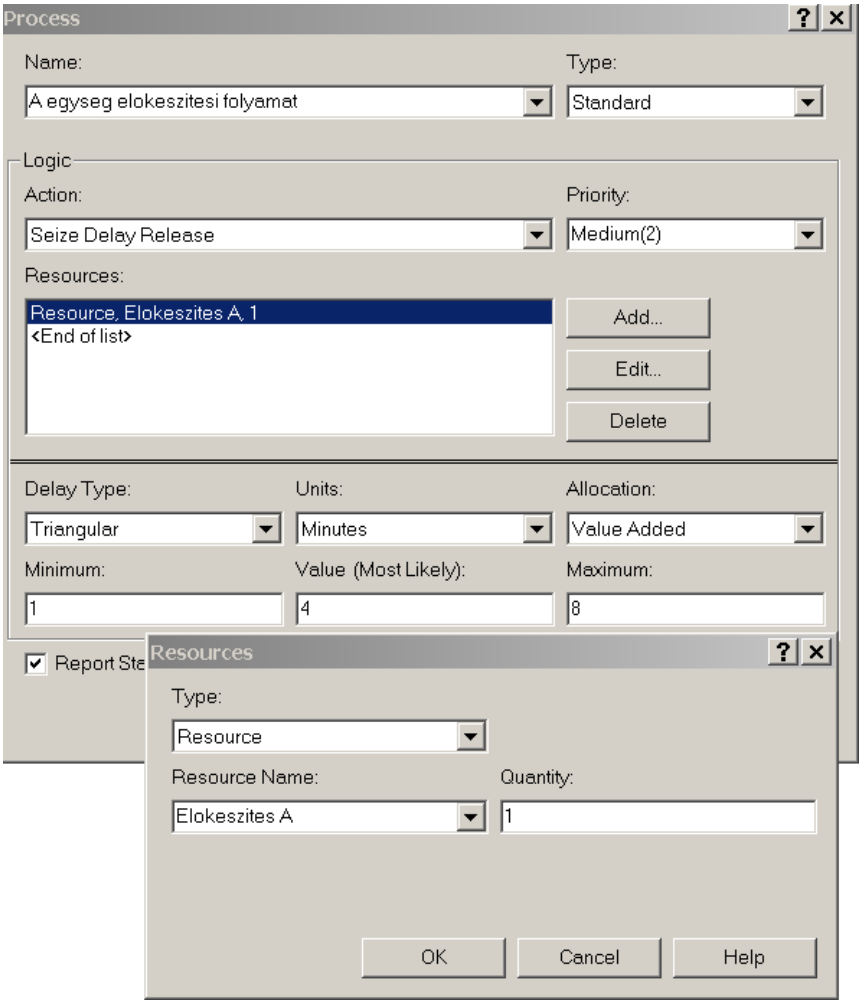
A **variables** (változók) a felhasználó által definiált globális adattároló objektumok, amelyekben módosítható állapotinformációkat tárolunk. A változókat inicializálhatjuk, és a program futása alatt változtathatjuk az értéküket. Olyan globális változók, amelyek a modellben bárhol láthatóak, vagyis elérhetőek, vizsgálhatók és módosíthatók a modell bármely komponensében. Az Arena programban a változók tipikus alkalmazásának helyei a **Decide** modulok, és az értéküket az **Assign** modulokban változtatjuk. A felhasználó által definiált változóktól

eltérően az Arena **belső rendszerváltozói** csak olvashatóak (read-only), vagyis felhasználhatjuk azokat különböző vizsgálatokhoz, statisztikák gyűjtéséhez, de az értéküket nem tudjuk megváltoztatni, kizárólag csak a rendszerváltozások hatására változik az értékük. Például az *NQ*(*A egyseg elokeszitesi folyamat.Queue*) változó az „*A egyseg elokeszitesi folyamat.Queue"* sorban pillanatnyilag várakozó entitások számát tárolja. Hasonlóan az *NR*(*Elokeszites A*) változó a pillanatnyilag lekötött (*Busy*) „*Elokeszites A*" nevű erőforrások számát tárolja. Fontos Arena változó a *TNOW*, amely a szimulációs idő (szimulációs óra) pillanatnyi értéket adja vissza és a *TFIN* amely a szimuláció befejezésének időpontját tárolja. Az Arena váltók listája az *Arena Variables Guide* című online könyvben és az *Arena Help*-ben található.

Az **Expressions** (kifejezések) speciális változók, amelyek valamilyen formulával (expression) társítva tárolnak értékeket. Ezek kényelmesen használható matematikai kifejezések, amelyek a modellben bárhol és bármikor meghívhatók. Amikor egy kifejezés megjelenik a modellben, akkor az azonnal kiértékelődik az adott időpontban, és a kiszámított érték a kifejezés nevéhez társul. A kifejezésekben a rendszer és a felhasználó által definiált változók, valamint az attribútumok egyaránt használhatók.

Az **Attributes** (attribútumok) az entitásokhoz társított adattároló objektumok. A változóktól eltérően, amelyek globálisak, az attribútumok lokálisan az entitásokhoz kötődnek. Egy entitás különböző elemeihez különböző attribútum értékek tartózhatnak. Például az ügyfelek (mint entitások) érkezési időpontjait tárolhatjuk egy az ügyfél entitáshoz tartozó attribútumban, ami később lehetővé teszi az individuális várakozási idők kiszámítását. Amikor az érkezéskor egyidejűleg több entitás érkezik, akkor az érkezés típusa is attribútumként az entitáshoz köthető, ami lehetővé teszi a szeparált statisztika gyűjtését minden entitás típusra.

Miután kitöltöttük a két érkező részegységhez tartozó **Assign** modulokat és a részegységekhez rendeltük a szerelési és az érkezési időt, a modellépítést az ***A*** és ***B*** részegységek előkészítését modellező **Process** modulok paraméterezésével folytatjuk.



4.3. képernyő. Az ***A*** *egyseg elokeszitesi folyamat* dialógusablaka

4.5. táblázat

A *Process1* modul paraméterei

Name	A egyseg elokeszitesi folyamat
Action	Seize Delay Release
Resources	
Type	Resource
Resource Name	Elokeszites A
Quantity	1
Delay Type	Triangular
Units	Minutes
Minimum	1
Value (Most Likely)	4
Maximum	8

A **Process** modulban az Action lenyíló listából (*4.3 képernyő*) négy különböző tevékenység választható: *Delay* (Késleltetés), *Seize Delay* (Megfogás Késleltetés), *Seize Delay Release* (Megfogás Késleltetés, Elengedés), *Delay Release* (Késleltetés, Elengedés).

A ***Delay*** (késleltetés) tevékenység azt jelenti, hogy az entitás specifikus késleltetési állapotba kerül. Ez a tevékenység nem igényel erőforrást, és egyidejűleg több entitás is várakozási állapotba kerülhet.

A ***Seize Delay*** azt jelenti, hogy az entitás lefoglalja erőforrást, az entitás késleltetési, (várakozási) állapotba kerül, de a folyamat végén nem szabadítja fel az erőforrást a következő entitás számára. Az erőforrás felszabadítása csak később történik meg.

A ***Seize Delay Release*** azt jelenti, hogy entitás lefoglalja az erőforrást, amit a késleltetés követ, majd a folyamat végén az erőforrás felszabadul.

A ***Delay Release*** azt jelenti, hogy az erőforrás már korábban foglalttá vált, az entitás késleltetési állapotba kerül, majd az erőforrás felszabadul.

Mivel az előkészítési területeken gépeket (erőforrásokat) használunk, olyan tevékenységre van szükségünk, amely megengedi a sorban való várakozást, amíg az erőforrás szabaddá nem válik, a műveleti időnek megfelelően késlelteti az entitást, majd a folyamat végén felszabadítja az erőforrást. Ezeknek az igényeknek a *Seize Delay Release* tevékenység felel meg.

Nyissuk meg a *Process 1* modul dialógusablakát (*4.3 képernyő*), gépeljük be a *4.5 táblázatban* megadott paramétereket. Az erőforrás információk megadásához kattintsunk az *Add* gombra.

Az adatok megadásánál, ahol lehet, célszerű a lenyíló listákból választani. Ennek oka, ha egyszer beírtunk egy nevet, akkor más helyen is ugyanezt kell használni. Az **Arena** nevek nem érzékenyek a kicsi és nagy betűkre, de betűknek és az üres helyeknek azonosnak kell lenniük, továbbá nem használhatunk ékezetes betűket. A listából való választás csökkenti a tévedés lehetőségét. Ha a megadott név csak kismértékben tér el az először beírttól, akkor az **Arena** hibaüzenetet küld. Rosszabb esetben a modell hibásan fut.

Jegyezzük meg, hogy amikor a modellablakba helyezünk egy modult, az **Arena** automatikusan alapértelmezés szerinti nevet és beállításokat fog hozzárendelni. Ezek az alapértelmezett nevek objektum nevek (*module, resource* stb.) hozzájuk fűzött számokkal. A névhez fűzött számok minden egyes modul hozzáadás után növekednek, például, *Process 1, Process 2*, és így tovább, azaz a modellben a modulok önálló nevet kapnak. Ennek két oka is van. Az első ok a kényelem, elfogadhatjuk az alapértelmezett erőforrás nevet, vagy megváltoztathatjuk. A második ok az, hogy az **Arena**-ban minden objektumnak egyedi névvel kell rendelkeznie,

még akkor is, ha az objektumok típusa különböző. Máskülönben az **Arena** nem tudja meghatározni melyik objektumot kell társítania azzal a névvel, amit egynél többször használtunk.

A felhasználó segítése érdekében az **Arena** számos automatikus elnevezést használ, amelyek többségét észre sem lehet venni. Például, ha rákattintunk a **Queue** adatmodulra, azt fogjuk látni, hogy az **Arena** az *„A egyseg elokeszitesi folyamat”* területen az *„A egyseg elokeszitesi folyamat.Queue”* nevet hozzárendelte a sorhoz. Természetesen az alapértelmezett angol nevek helyett saját elnevezéseket is használhatunk.

Vegyük észre, amikor két választott tevékenység közül az egyik *Seize*, akkor a modul bezárása után az **Arena** elhelyez egy animáló sort (egy félegyenest, egy vízszintes és egy rövid függőleges vonal a jobb oldalon) a szóban forgó **Process** modul felett. Ez lehetővé teszi a várakozó entitások megjelenítését a szimuláció futása alatt. Ha rákattintunk a vonalra, megjelenik a sor neve is.

A *Process 2* modul kitöltéséhez használjuk a *4.6. táblázatot*.

A következő lépés a szerelés és minőség-ellenőrzés művelet adatainak a megadása, amelyeket a *Process 3* modul dialógusablakába kell beírni. A dialógusablak mezőinek a kitöltését a *4.7. táblázat* segíti. Emlékezzünk arra, hogy korábban az **Assign** modulokban megadtuk a szerelés és minőség-ellenőrzés attribútumait. Amikor egy alkatrész az erőforrás irányítása alá kerül (*Seize*), akkor elkezdődik az összeszerelése (*Delay*), ami az **Assign** modulban definiált *„Szerelesi ido”* kifejezés értékének megfelelő ideig tart.

4.6. táblázat

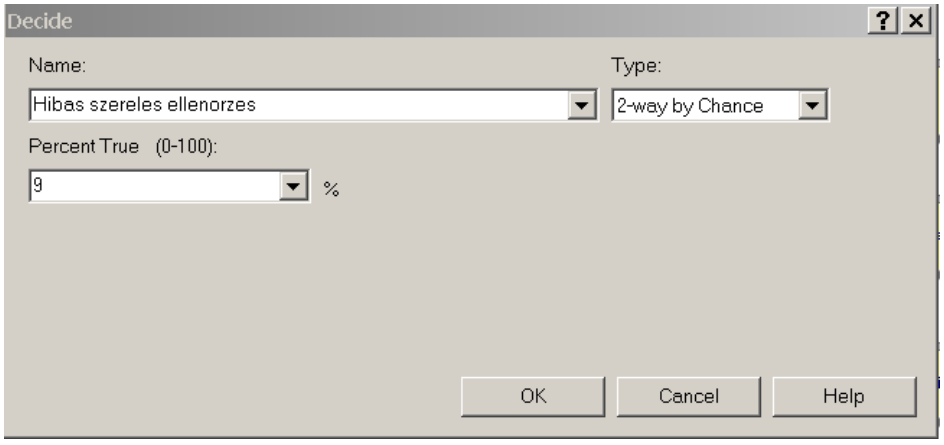
A *Process 2* modul paraméterei

Name	B egyseg elokeszitesi folyamat
Action	Seize Delay Release
Resources	
Type	Resource
Resource Name	Elokeszites B
Quantity	1
Delay Type	Triangular
Units	Minutes
Minimum	3
Value (Most Likely)	5
Maximum	10

4.7. táblázat

A *Process 3* modul paraméterei

Name	Szerelesi folyamat
Action	Seize Delay Release
Resources	
Type	Resource
Resource Name	Szereles
Quantity	1
Delay Type	Expression
Units	Minutes
Expression	Szerelesi ido



4.4. képernyő. A minőségellenőrzés dialógusablaka

Az összeszerelést és minőségellenőrzést követő, és a minőség-ellenőrzés eredményétől függő döntést a *Decide 1* modullal modellezzük. A modul dialógusablakát *4.4. képernyő* szemlélteti. A **Name** mezőbe *„Hibas szereles ellenorzes"* nevet írjuk, és a **Type** mezőnél az alapértelmezett *2-way by Chance* opciót fogadjuk el, vagyis az entitás a modult csak *True* vagy *False* értékkel hagyhatja el. A párbeszédablakban ki kell még tölteni a **Percent True** mezőt. Esetünkben a hibás egységek a *True* oldalon lépnek ki, ezek aránya 9%, ezért a mezőbe ezt az értéket írjuk be. Ez azt eredményezi, hogy a minőségileg megfelelő egységek (91%) a *False*, a hibás egységek (9%) pedig a *True* kimenetnél hagyják el a modult. A minőségileg kifogástalan egységeket a késztermék raktárba irányítjuk, a hibás egységeket pedig a javítási területre.

4.8. táblázat

A *Decide 1* modul paraméterei

Name	Hibas szereles ellenorzes
Percent True	9

A minőségileg nem megfelelő egységek javítását (újramunkálását) a *Process 4* modullal modellezzük. A modul adatai a *4.9. táblázatban* találhatók. A javítás időtartama nem függ az egység típusától, ezért elegendő egy **Process** modult használni, amelynek a működése leginkább a *Process1* és *Process2* modulokéra hasonlít.

4.9. táblázat

A *Process 4* modul paraméterei

Name	Javitasi folyamat
Action	Seize Delay Release
Resources	
Type	Resource
Resource Name	Javitas
Quantity	1
Delay Type	Expression
Units	Minutes
Expression	EXPO(45)

A második *Decide 2* modult a javított egységek minőségellenőrzését követő döntés modellezésére használjuk. Ebben a modulban a javítás után hibás (20%) és a javítás után megfelelő (80%) egységeket választjuk szét ugyanazzal a módszerrel, amit a *Decide 1* modulnál alkal-

maztunk. A *Decide 2* dialógusablakába beírandó adatokat a *4.10. táblázat* foglalja össze. A hibás egységek itt is a *True* ágon távoznak.

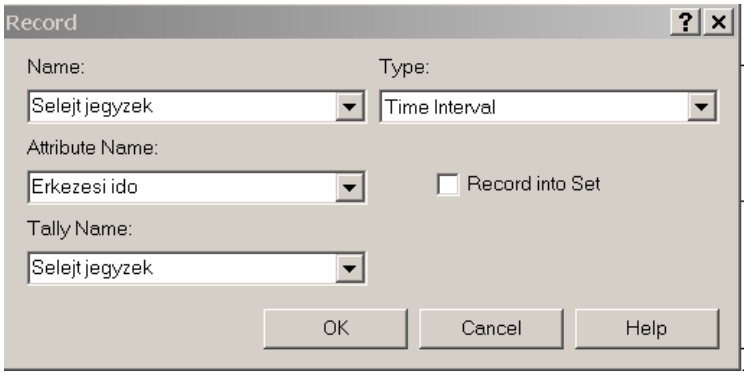
4.10. táblázat

A *Decide 2* modul paraméterei

Name	Hibas javitas ellenorzes
Percent True	20

Miután a modellünkben definiáltuk a műveleteket, már csak a **Record** és **Dispose** modulokat kell kitöltenünk. Emlékezzünk arra, hogy a szimuláció kimeneteként, összesített statisztikákat akarunk gyűjteni, az erőforrások kihasználásáról, a sorok hosszáról, a sorban eltöltött időről. Ez a három statisztika automatikusan elkészül, amikor a **Process** modult *Action* opcióval használjuk, feltéve, hogy a **Process** dialógusablakában (*4.3. képernyő*) a *Report Statistics* ellenőrződoboz, és a *Run>Setup> Project Parameters* ablakban pedig a *Processes* ellenőrződoboz be van kapcsolva. Kíváncsiak vagyunk az átfutási időkre is, még pedig a minőségileg megfelelő, a javított minőségileg megfelelő és selejtes egységekre vonatkozóan külön-külön. A átfutási idők összegyűjtéséről a **Record** modul gondoskodik a *Tally* (ellenőrző) formátumnak megfelelően.

A selejtes egységek statisztikáit rögzítő modul kitöltött párbeszédablaka a *4.5. képernyőn* látható. A Type mező értékét a *Time Interval*-t a legördülő listából választjuk. Az alapértelmezett Tally Name a modul nevével *(„Selejt jegyzek”)* azonos, amit megváltoztathatunk. A **Record** modul egy olyan *Tally* statisztikát produkál, ami az *„Erkezesi ido”* nevű attribútum entitáshoz rendelése és az entitás **Record** modulba érkezése között eltelik, ez gyakorlatilag az entitás rendszerben eltöltött idejét jelenti. A két másik **Record** modul (*„Javitasi jegyzek”, „Szallitasi jegyzek”*) tartalmának részletezésétől eltekintünk, mivel azok a *„Selejt jegyzek”* nevű modullal teljesen analóg módon paraméterezhetők.



4.5. képernyő. A *Selejt jegyzek* Tally dialógusablaka

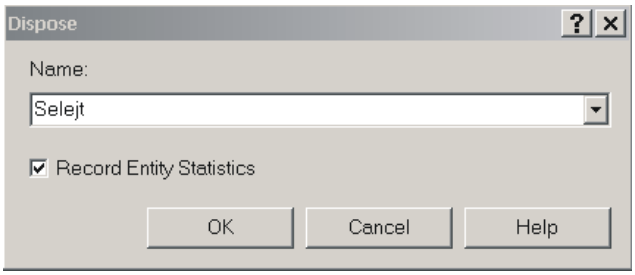
4.11. táblázat

A *Record 1* modul paraméterei

Name	Selejt szamlalas
Type	Time Interval
Attribute Name	Arrive Time
Tally Name	Selej jegyzek

Az utolsó három **Dispose** modul az elektronikus-egységek elhelyezést, a modulból való kiléptetését modellezi. Egy-egy **Dispose** modulon az összes azonos tulajdonságú entitás átfut, ezért a modul tartalmaz egy animációs változót (számlálót), amely a modul jobb sarkánál jelenik meg, és a modellen áthaladó entitások aktuális számát mutatja. A változó értéke mind a három

modulnál megjelenik, így a futás ideje alatt összehasonlítható a selejtes, a javított minőségileg megfelelő és a minőségileg megfelelő entitások száma.



4.6. képernyő. A selejtes egységek kilépésének dialógusablaka

A Selejt nevű **Dispose** modul adatai a *4.6. képernyőn* és a *4.12. táblázatban* láthatóak. A dialógusablakban a *Recorded Entity Statistics* box alapértelmezés szerint bekapcsolt állapotban van. Ha azonban csak azt szeretnénk tudni, hogy mennyi a minőségileg megfelelő egységek száma, beleértve a javított egységeket is, és csak ezekről kívánunk statisztikát készíteni, akkor az ellenőrzést ki kell kapcsolni. Természetesen az ellenőrző boksz-ot a másik két modulban is bekapcsolva hagyjuk.

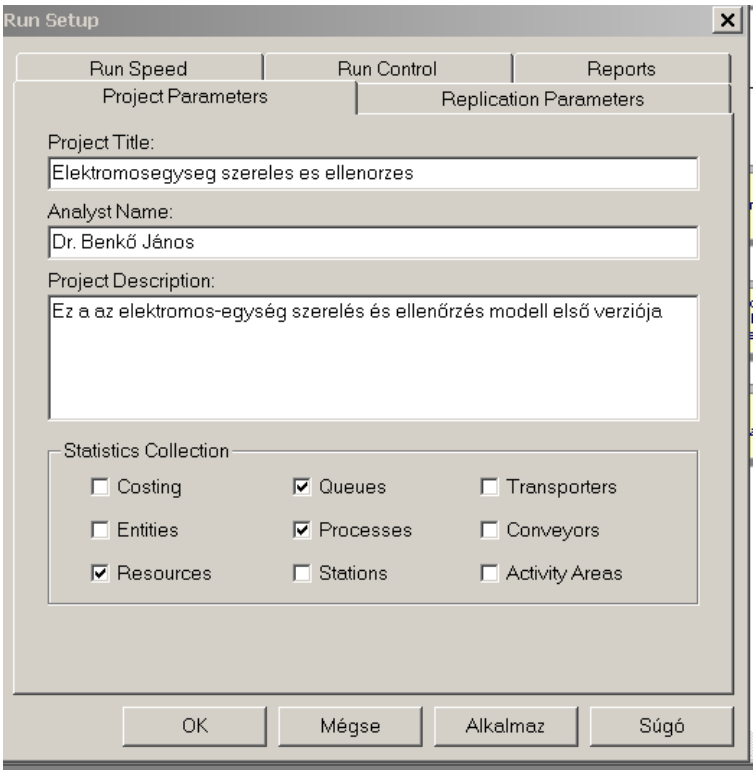
4.12. táblázat

A *Dispose 1* modul paraméterei

Name	Selejt

A másik két **Dispose** modul, a *Javitott* és a *Szallitott* hasonlóan töltendő ki.

A modell majdnem készen áll a futtatására, már csak néhány apró lépést kell megtenni, a modell teljes felépítéséhez.



4.7. képernyő: A Run Setup Project Parameters dialógusablak

A modell elvileg már ebben az állapotában is futtatható, de ha elindítanánk, akkor soha nem állna meg, mivel az **Arena** még nem tudja, hogy mikor kell leállítani a szimulációt. A futás

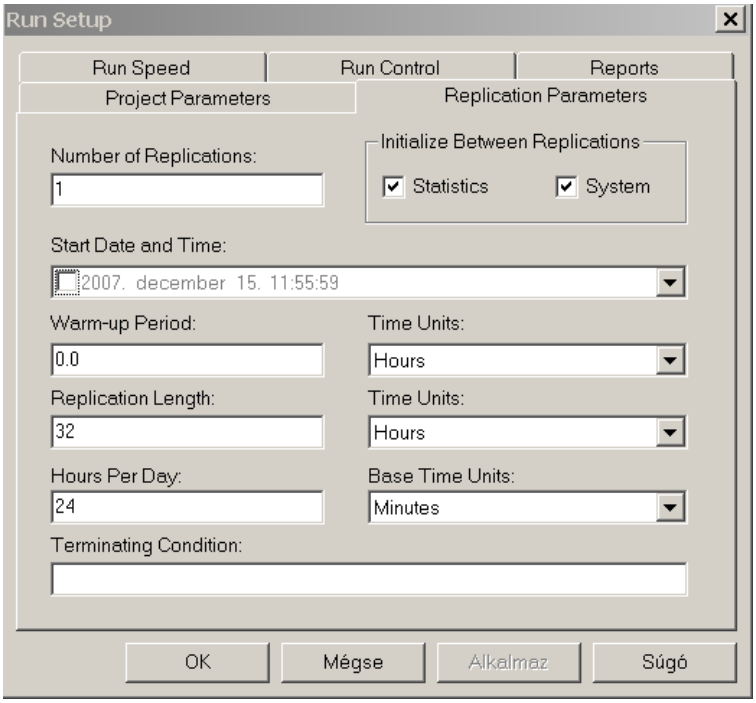
paramétereit *Run>Setup* menüponton keresztül, a szimuláció futását szabályozó dialógusablak megnyitásával állíthatjuk be. Az ablakhoz tartozó fülek: **Run Speed, Run Control, Reports, Project Parameters, Replication Parameters**.

4.13. táblázat

A **Run Setup Project Parameters** dialógusablak paraméterei

Project Title Analyst Name	Elektronikus-egyseg szereles es ellenorzes Dr. Benkő János
Project Description	Ez az elektronikus-egység szerelés és ellenőrzés modell első verziója
Statistics Collection Entities	Törölve (üres)

Az adatok az első fülhöz, (**Project Parameters**), a *4.13 táblázatban* láthatóak. Megadtuk a projekt címét (Project Title) és az elemző nevét (Analyst Name), amelyek megjelennek a kimeneti jelentéseken, csak úgy, mint a Projekt leírást dokumentáló a rövid összegzés. Az ellenőrző boxok között az Entities boxot kapcsoljuk ki, mivel az elemzésünkhöz nincs szükségünk az entitásokat jellemző statisztikákra. Ha mindkét beállítással futtatjuk a szimulációt, akkor érzékelhetjük a különbséget a kimeneti jelentéseknél.



4.8. képernyő: A **Run Setup Replication Parameters** dialógusablak

4.14. táblázat

A **Run Setup Replication Parameters** dialógusablak paraméterei

Replication Length	32
Base Time Units	Minutes

Szükségünk lesz a futtatási idejének a meghatározására is, amit a **Replication Parameters** fülhöz tartozó ablakban definiálhatunk (*4.8. képernyő*). Az ismétlési idő hosszát (Replication Length) 32 órára állítottuk (4 egymást követő 8 órás műszak) az alap időegységet (Base Time Units) percekben adjuk meg, és a többi mezőben ne változtassuk az alapértelmezett értékeket. A **Run Speed, Run Control, és Reports** fülekhez tartozó ablakokban szintén hagyjuk meg az alapbeállításokat. Az elérhető beállítások megtekintéséhez nyissuk meg a füleket.

Mielőtt futtatnánk a modellünket, alkalmazzunk még egy utolsó finomítást. Amikor egy modellt futtatunk, az **Arena** alapértelmezésben az összes entitást ugyanazzal az ikonnal jeleníti meg. Mivel két különböző terméktípusunk van (***A*** és ***B*** termék), célszerű lenne, ha ezeket az animáció alatt vizuálisan is meg tudnánk különböztetni. A megoldáshoz **Basic Process Panelen** kattintsunk az **Entity** adatmodulra, amelynek hatására a modellablak alsó részén megjelenik a *4.9. képernyőn* látható táblázat.

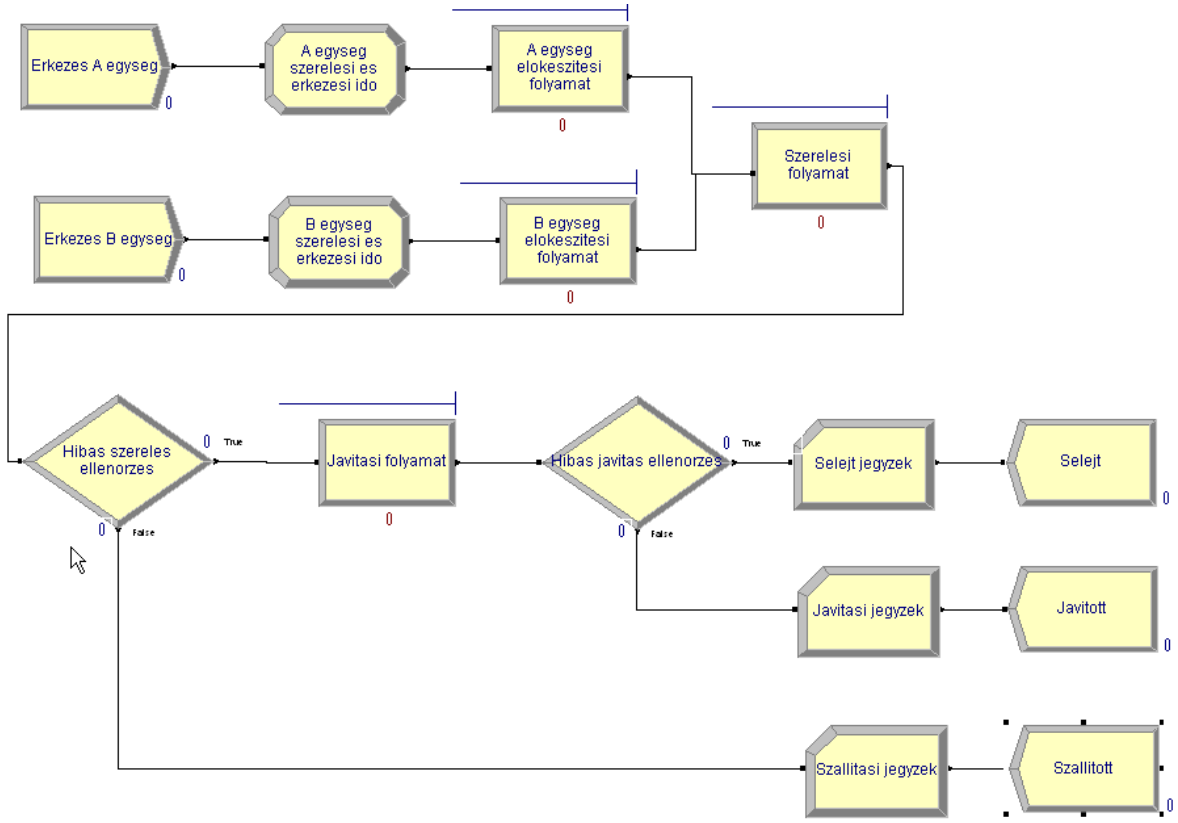
Entity - Basic Process

	Entity Type	Initial Picture
1	A egyseg	Picture.Blue Ball
2	B egyseg	Picture.Red Ball
	Double-click here to add a new row.	

4.9. képernyő: Az **Entity** adatmodul részlete

Nyissuk ki az *„A egyseg”* és a *„B egyseg”* sorokban az Initial Picture (kezdeti kép) mező lenyíló ablakot, és a listából válasszunk egy képet. A *4.9. képernyőn* a *„B egyseg”*-hez a *Red Ball*-t (kék labda) és az *„A egyseg”*-hez a *Blue Ball*-t (piros labda) választottuk. Ha szeretnénk alaposabban megnézni az ikonokat az *Edit>Entity Pictures* menüpont választása után megnyíló **Entity Picture Placement** ablakban megjelennek az elérhető ikonok. Később megbeszéljük részletesen, hogy miként lehet ezt a lehetőséget használni.

A végső modellünknek úgy kellene kinéznie, mint ahogy azt a *4.3. ábra* mutatja.



4.3. ábra: Az elektronikus-egység szerelés és ellenőrzés modell végleges alakja

4.1.3. A modell futtatása

Mielőtt futtatnánk a modellt, ellenőrizzük a hibákat. Ezt megtehetjük úgy, hogy a **Run Interaction** eszközsoron a *Check* gombra (√) kattintunk, vagy a *Run>Check Model* menüpontot választjuk, amit a klaviatúrán az F4 funkció billentyű lenyomásával is elérhetünk.

A válasz egy kis ablakban jelenik meg, és ha szerencsénk van, akkor az üzenet a következő: *„No errors or warnings in model”*. Ha nincs szerencsénk, akkor a megjelenő hibaüzenet leírja a hibát. Hibaüzenet esetén használjuk a *Find* opciót a hiba megkereséséhez, ha a gomb be van kapcsolva. Ez a tulajdonság megpróbálja megmutatni, hogy az **Arena** hol feltételezi a hibát.

Szándékosan tegyünk egy hibát a modellbe és próbáljuk ki ezt a lehetőséget. Később, összetettebb modelleknél tapasztalni fogjuk, hogy egyre gyakrabban kell élni ezzel a lehetőséggel.

Ha a modellellenőrzés eredménye azt jelzi, hogy nincs hiba, akkor készen állunk a szimuláció futtatására. Négy lehetőség van a modell futtatására, de itt csak háromról beszélünk. Az első mód: a szimuláció futtatása animációval. Használjuk a *Go* gombot a Standard eszközsoron, *Run > Go* parancs, vagy az *F5* gomb. Ha még nem ellenőriztük a modellünket, vagy az utolsó ellenőrzés óta változtattunk rajta, akkor az **Arena** futás előtt először ellenőrizi a modellt, majd alaphelyzetbe állítja a modellt és adatait, végül futtatja azt. A futtatás során az **Arena** elrejt néhány ábrát, hogy az animációra tudjon figyelni. Emiatt nem kell aggódni, a futtatás végén az ábrák vissza fognak térni (de a *View > Layers* paranccsal ellenőrizhetjük is az ábrák meglétét).

Ha a státuszsort aktívan hagyjuk (a képernyő alján) láthatjuk, hogy az **Arena** mit csinál. E sor jobb oldalán három másik információ is megjelenik: az ismétlések száma, a pillanatnyi szimuláció idő és a szimuláció státusza.

A szimuláció indítása után felgyorsíthatjuk, vagy lelassíthatjuk az animáció sebességét. Ezt megtehetjük a modell futtatása közben: a „<” billentyű lenyomásával lassítjuk, a „>” billentyű lenyomásával gyorsítjuk az animációt. Ha lenyomjuk e billentyűk valamelyikét, akkor megjelenik a pillanatnyi animáció sebesség (*Animation Speed Factor*) a státuszsor bal sarkában. Az animáció sebesség szintén növelhetjük, vagy csökkenthetjük a **Run Speed** fülnél a **Run Setup** párbeszédablakban. Ez az opció arra is használható, hogy megadjunk a pontos sebességértéket.

A szimuláció futását megállíthatjuk a **Run** eszközsoron a *Pause* gombra kattintva (), a *Run > Pause* paranccsal, vagy az *Esc* billentyűvel. Ez ideiglenesen felfüggeszti a szimulációt, és a "*User interrupted*" üzenet jelenik meg a képernyőn a státuszsorban.

A Pause módban az animációban látható entitásokra duplán klikkelve megjelenik az **Entity Summary** párbeszédablak, amely listázza az entitás tulajdonságainak értékeit. Ez nagyon hasznos lehet akkor, amikor hibát keresünk egy modellben. A **Run** eszközsoron használhatjuk a *Step* gombot, hogy az egyes tulajdonságok változását lépésenként tekintsük meg. A *Go* gomb lenyomásával bármikor folytathatjuk a szimulációt „normális” futtatását.

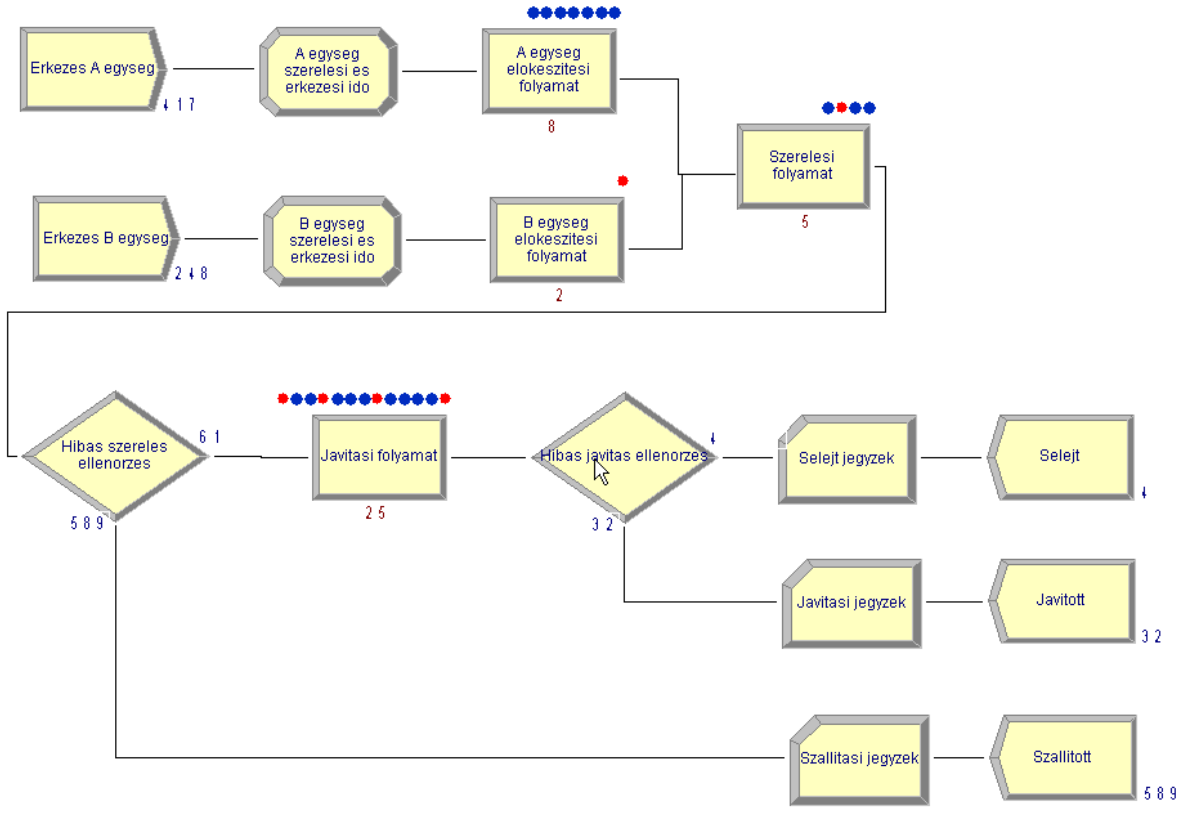
A szimuláció futtatásnak ez a módja biztosítja a legtöbb információt, de így hosszú ideig tarthat a futtatás. Ebben az esetben a szükséges idő az animáció sebesség faktortól függ. Előreugorhatunk az időben a *Pause* gomb lenyomásával majd a **Run** eszközsorban a *Fast-Forward* gombra () kattintva, vagy *Run > Fast-Forward* paranccsal. Ilyenkor a szimuláció sokkal gyorsabban fut, de az animációs grafikát nem frissíti. A futtatást bármikor megállíthatjuk (*Pause*) és visszatérhetünk animációs módba, ráközelíthetünk (*Zoom In* +), távolíthatunk (*Zoom Out* −), mozoghatunk a szimulációs ablakban a gördítősáv, vagy a kurzorbillentyűk segítségével.

A *Fast-Forward* használatával a szimuláció jóval kevesebb időt igényel. Ha csak a numerikus szimuláció eredményei érdekelnek bennünket, akkor kikapcsolhatjuk az animációt (a számítás grafikai megjelenítését). Ezt megtehetjük a *Run* > *Run Control* > *Batch Run* (nincs animáció) opció segítségével.

A *Run* > *Run Control* opció lehetővé teszi, hogy konfiguráljunk egyéb futtatási idő opciókat. Most válasszuk a *Batch Run* (nincs animáció) opciót. Jegyezzük meg, ha visszatérünk ehhez az opcióhoz van egy ellenőrző a baloldalon. Hagyjuk jóvá ezt az opciót és klikkeljünk a *Run* gombra. Észrevehető mennyivel gyorsabb a szimuláció futása. Az egyetlen hátrány, hogy le kell állítanunk a futtatást és vissza kell állítanunk az animációs beállításokat ahhoz, hogy újra animációval futtathassunk. Ha nagy modellünk van, vagy előre láthatóan hosszú futtatásra számíthatunk, és csak a numerikus eredmények után érdeklődünk, akkor ez az opció sokkal előnyösebb és gyorsabb, mint a *Fast-Forward*.

Egy modell építése alatt az eszközsorok többségének megjelenítésére és elérhetőségére valószínűleg szükség van. Ezzel szemben a modell futtatásakor a legtöbb eszközsor csak a helyet foglalja mivel nem aktívak. Az **Arena** felismeri ezt, és menti az eszközsor beállításokat minden módban. Ennek előnyét úgy használhatjuk ki, hogy megállítjuk a futtatást (*Pause*), és eltávolítjuk azokat az eszközsorokat, amelyekre nincs szükségünk a futtatás során. Mikor a futtatás befejeződik, ezek az eszközsorok újra megjelennek.

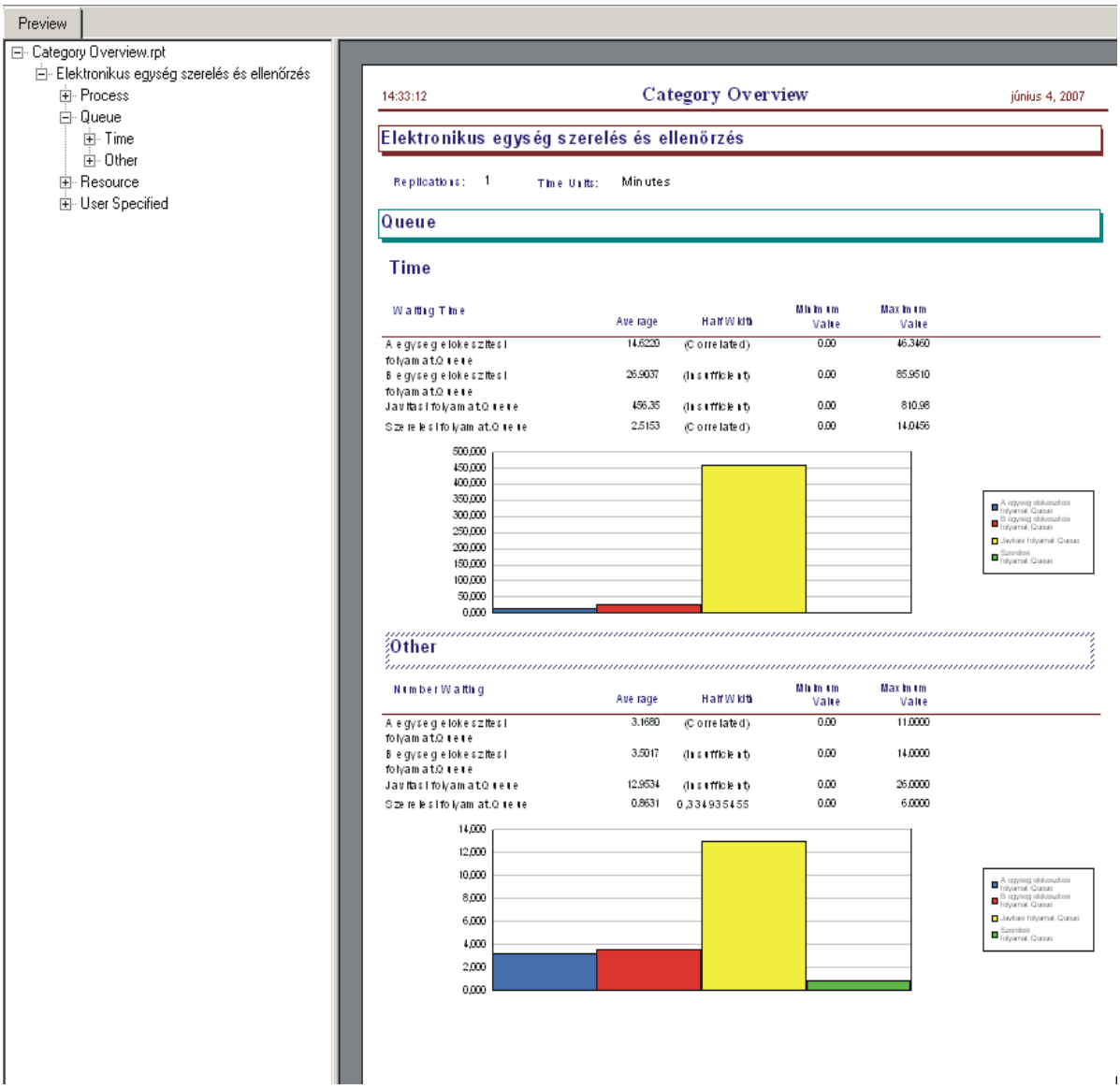
4.1.4 Az eredmények megtekintése



4.4. ábra: Az animáció eredménye

Ha kiválasztottuk a *Run>Go* menüopciót (vagy a ▶ gombot), valószínűleg észrevesszük, hogy a rendszeren áthaladó kék és piros labdákon (***A*** vagy ***B*** entitások) kívül, még néhány animált számláló is megjelenik a szimuláció futtatása során. Minden egyes **Create**, **Process** **Dispose** modulhoz egy számláló és a **Decide** modulhoz két számláló tartozik. A **Create,**

Decide és Dispose modulok számlálói eggyel növekednek, valahányszor egy entitás belép a modulba. A **Process** modul esetében a számláló a pillanatnyilag a modulban tartózkodó entitások számát mutatja, beleértve az erőforrásra várakozó és már feldolgozás (megmunkálás) alatt álló entitásokat. Ha a *Run>Fast-Forward* menü opciót (vagy a gombot) választjuk, akkor ezek a számlálók (és az entitások száma a sorokban) a futás végén automatikusan frissítődnek, akkor is, ha megszakítjuk a modell futását vagy nézetet váltunk. A szimuláció végén megjelenő számok (eredmények) a *4.4. ábrán* láthatók.



4.5. ábra: Összefoglaló jelentés a sorokról a *4.1. modellben*

A modell futásának befejeződésekor az **Arena** megkérdezi, hogy akarjuk-e látni az eredményeket. Ha a *Yes*-re kattintunk, akkor megnyílik a **Category Overview Report** (kategorizált áttekintő jelentés) ablaka (ez az alapbeállítás). Természetesen csak a **Run Setup Project Parameters** dialógusablakban (*4.8. képernyő*) kiválasztott statisztikák jelennek meg. Ha a dialógusablakban a *Statistics Collection* jelölő négyzetek valamelyikét üresen hagyjuk, akkor a hiányzó statisztika helyett *„No Summary Statistics Are Available”* (az összesítő statisztika nem elérhető) üzenetet kapunk. Mindazonáltal megváltoztathatjuk ezeket a beállításokat, hogy megnézzük a különbséget a két jelentés között.

A jelentésablak bal felső sarkában a **Preview** fül alatt megjelenő, faszerkezetbe rendezett jelentések közül választhatunk, a jelentésablak jobb oldalán és alján található gördítő gombokkal is navigálhatunk, valamint az ablak bal felső sarkában lévő léptető gombokkal az egyik jelentésről a másikra ugorhatunk (*4.5 ábra*). A jelentés olyan statisztikákat állít elő, amelyeket korábban a **Run Setup** párbeszédablakban a **Project Parameters** fül alatt a *Statistics Collection* területen bejelöltünk. A jelentésünkben megjelenő statisztikák a **Process, Queue, Resource** és **User Specified.** A **User Specified** (felhasználó által definiált) statisztikák annak köszönhetőek, hogy a modellünkbe **Record** modulokat építettünk azért, hogy azok a *selejt, javított, szállított* kategóriák szerint csoportosított statisztikákat gyűjtsenek a ciklusidőkről.

A jelentésünkben **háromféle statisztika található**: ellenőrző (*tally*), folytonos (*persistent*) és számláló (*counter*). A negyedik statisztika típus a kimeneti adatok (*output*) csak a többszörös ismétlés (*multiple replication*) esetén érhető el. Az ellenőrző statisztika a **Record** modulok által gyűjtött folyamatidőt, várakozási időt és intervallum időket tartalmazhat. Az időben folytonos statisztikákban a sorokban várakozó entitások száma, az eszközhasználat és az eszközkihasználtság szerepelnek. A számlálótípusú statisztikák a kumulált időt, a belépő, a kilépő és az összes feldolgozott entitások számát tartalmazzák. A felsorolt kategorizált statisztikák a *4.5. ábrán* bemutatott formában jelennek meg.

Az ellenőrző és a folytonos statisztikákban egyaránt megjelenik a becsült átlag 95%-os konfidenciaszintjéhez tartozó konfidencia-határ. A statisztikákban a *Half Width* a konfidencia-intervallum felét jelenti, feltételezve, hogy a konfidencia határok a becsült átlaghoz viszonyítva ± irányban szimmetrikusak (részletesen lásd a 2. fejezetben). A statisztikák ezenkívül tájékoztatnak a megfigyelt értékek minimális és maximális értékéről is.

Minden egyes ismétlés végén a program minden megfigyelt statisztika *steady-state* (állandósult) állapothoz tartozó várhatóértékére próbálja kiszámítani a 95%-os konfidenciaszinthez tartozó konfidencia-határokat, és ehhez az ún. „*batch mean*" eljárást használja. Az **Arena** először ellenőrzi, hogy elegendő adat gyűlt-e össze a kritikus statisztikai hipotézis (korrelálatlan tételek) vizsgálatához, ami a „*batch-means*" módszerhez szükséges. Ha nincs elég adat, akkor „*Insufficient*" (nem elégséges) felirat jelenik meg a jelentésben, és a jelentés nem tartalmazza a konfidencia intervallumot. Ha már van elegendő adat a korrelálatlan tételek teszteléséhez, de a vizsgálat hibát jelez, akkor „*Correlated*" (korrelált) felirat jelenik meg, és nincs konfidencia határ. Végül, ha elegendő adat gyűlt össze, és a hipotézisvizsgálat is sikeres, akkor a konfidencia-határok és a várhatóérték is megjelenik a jelentésben. Az **Arena** így kizárja a megbízhatatlan konfidencia-határok használatát.

Hibás eredményekre vezet, ha a rövid futási idő alapján próbálunk konklúziókat levonni, ezért nagyon fontos a futás hosszának és az ismétlések számának helyes meghatározása. Mégis, ha megnézzük a várakozási időre vonatkozó eredményeket és a várakozások számát a sorokban (*4.5. ábra*), láthatjuk, hogy a *„Javitasi folyamat"* modulban a várakozási idő kiemelkedően nagy, és a sor sokkal hosszabb, mint más munkaállomásokon (a kapacitás egyensúly hiányát megerősíti a **Category Overview** jelentés **Process** és **Resources** szektora is). Ez egyrészt azt jelzi, hogy a *„Javitas"* nevű erőforrás kapacitása kevés az elvégzendő feladat ellátására, másrészt remek lehetőséget kínál az állomás variálására. Hamarosan hivatkozni fogunk erre a megállapításra a 4.2 alfejezetben. (Vegyük észre a *4.5. ábrán*, hogy amikor többféle folyamat tartozik egy kategóriához, esetünkben a **Queue** (sor) kategóriához négy folyamat sorai, akkor a **Category Overview** jelentés az átlagokat ábrázoló grafikákat is megjeleníti, amelyek színkódolásúak, azonban ezeket érthető módon a fekete-fehér nyomtatásban nem tudjunk érzékeltetni.)

4.2. Az elektronikus-egység összeszerelő- és minőségellenőrző rendszer fejlesztése (4.2. modell)

Miután megépítettük és lefuttattuk a modellünk első változatát, a következő feladat ellenőrizni (verifikálni) a „kódolás" (**Arena** fájl) hibamentességét és megerősíteni (validálni), hogy a koncepcionális (fogalmi) modell megfelel a valóságos rendszernek. A **verifikálás** a modell helyes működésének, a **validálás** pedig a modell érvényességének vizsgálatára irányul. A vizsgálat az egyszerű mintapéldánkban nem túl bonyolult. A modulokból összeállított logikai konstrukciót elegendő összehasonlítani az eredeti probléma definíciójával. Sokkal nagyobb és összetettebb rendszerek esetén azonban a vizsgálat embert próbáló feladat lehet. Az animáció a verifikáció és a validáció alatt egyaránt hasznos lehet, mert általa teljesebb képet kapunk a modellezett rendszerről és annak működéséről. Lefuttatva a kifejlesztett és animációval gazdagított modellt, vizuálisan is érzékelhetjük, hogy hasonlóan működik-e leírt rendszerhez. Gyakran a verifikáció csak részben vezet eredményre, ilyenkor a következő lépés, a validáció sem lehet teljes, mert a validáció során azt kell bebizonyítani, hogy a szimuláció úgy viselkedik, mint a valóság. Ha esetleg a rendszer nem is létezik, akkor is kimeneti eredményekre (értékekre) támaszkodva meggyőzzük magunkat és másokat arról, hogy a modellünk megragadja a lényeget, és alkalmas arra, hogy a valósrendszer eseményeit megjósolja.

Tételezzük fel, hogy az eddigi erőfeszítéseink termékét, a modellt és annak eredményeit megmutatjuk a termelésvezetőnek, akinek az első észrevétele, hogy a rendszer működésének leírása nem teljesen pontos. A probléma definíciójában a modell csak egy műszakban működik. A rendszer a valóságban azonban kétműszakos, és a második műszakban a javítási folyamathoz két munkás van hozzárendelve. Ez megmagyarázza azt a korábbi megfigyelésünket, miszerint a javítási folyamat kapacitása nem elegendő.

A termelésvezető megjegyzi azt is, hogy a szerelési folyamatban meghibásodások is előfordulnak. Az összeszerelő sor időnként elromlik. A mérnökök nemrég vizsgálták ezt a problémát, és adatokat gyűjtöttek az összeszerelő sor hibáinak a meghatározásához. Arra az eredményre jutottak, hogy a meghibásodásoknak nincs lényeges hatása a folyamatra, és a létezésük nem jelent szűkkeresztmetszetet. A megfigyeléseikről azonban jegyzőkönyvet vettek fel, amelyek hozzáférhetőek. A jegyzőkönyv szerint a hasznos idő átlagértéke (meghibásodás végétől a következő meghibásodás kezdetéig) 120 perc, és a hasznos idő eloszlása exponenciális (a hasznos idő leírására ez gyakran használt eloszlás a valós modellekben, ha a hibák véletlenszerűen és egyenletesen jelentkeznek az idő előrehaladásával). A javítási idő eloszlása szintén exponenciális és átlagosan 4 percet vesz igénybe.

A termelésvezető a javítási területen a várakozó munkadarabok tárolására alkalmas speciális állványokat akar beépíteni. Az állványok egyenként tíz munkadarabot képesek tárolni, és a termelésvezető szeretné tudni, mennyit kell belőlük vásárolni.

A további feladatunk a modell megváltoztatása annak érdekében, hogy érvényesüljenek a termelésvezető észrevételei. A három új szempontot beépítése egyúttal alkalmat teremt az **Arena** további lehetőségeinek megismerésére.

Ahhoz, hogy beépítsük ezeket a változtatásokat meg kell ismerkednünk néhány új fogalommal. A *4.1 modellben* a futási időt négy 8 órás műszakra állítottuk be, és nem tettünk különbséget az első és második műszak között. Csak azt feltételeztük, hogy a rendszer állapota a műszak kezdetén ugyanaz, mint az előző műszak végén és eltekintettünk a köztes időtől. A termelés vezető ezt kifogásolta. Szükség van ezenkívül arra, hogy modellben változtatni tudjuk a műszakok jellemzőit is, mivel a javítási területen az első műszakban csak egy munkás dolgozik, a másodikban pedig kettő. A változtatás az egy- és kétműszakos folyamat között nagyon egyszerű megvalósítható. A jellemzőket a *„Javitasi folyamat"*-hoz tartozó **Resource**

Schedule (erőforrás ütemezés) megismerése révén illeszthetjük be a modellünkbe. Az erőforrás ütemező a futtatás alatt az automatikusan változtatja a javítási folyamat erőforrásainak számát, azaz változtatja az erőforrás kapacitását. Ezzel egyidejűleg növeljük a futás hosszát is, ezáltal több mint két napot szimulálunk a kétműszakos folyamatból. A **Resource Failure** (erőforrás meghibásodás) modul segítségével modellezni fogjuk a szerelőgép hibáját, amelylyel az idő vagy a gyártott mennyiség függvényében változtathatjuk az elérhető erőforráskapacitást (hasonlóan, mint az erőforrás ütemezésnél). Ez a speciális eszköz további lehetőségekkel bír az erőforrás állapotok reprezentálására. Végül a *Frequencies* (gyakorisági) statisztikát alkalmazzuk, hogy információt nyerjük a szükséges állványok számának megállapításához.

4.2.1 Az erőforrás ábrázolás kiterjesztése: ütemezés és fázisok

Eddig erőforrásainkat (előkészítés, szerelés, javítás), mint önálló erőforrásokat fixkapacitással (*Fixed Capacity*) és 1 kapacitásértékkel modelleztük. Emlékezzünk arra, hogy ezek a paraméterek a **Resources** modul alapértelmezései. Ha a két javító munkás modellezéséhez a *„Javitas”* erőforrás kapacitását egyszerűen 2-re változtatnánk, akkor az azt jelentené, hogy a két javító munkás mindkét műszakban elérhető lenne. Ezért változtassuk az erőforrás típusát *Based On Schedule*-ra (ütemezés alapján), és ütemezzünk egy javító munkást az első műszakra, (feltételezve, hogy minden műszak 8 órás) és két javító munkást a második műszakra. Ezt az **Arena**-ba épített ütemezővel tehetjük meg. Az ütemező lehetővé teszi, hogy az erőforrás kapacitását egy előre definiált minta szerint változtassuk az idő függvényében. Az erőforrás ütemezés úgy definiálható, mint időfüggő erőforrás-kapacitások sorozatának a változása.

Ezenkívül számolnunk kell a szerelő erőforrás időszakos meghibásodásával is. Ezt modellezhetnénk az ütemező segítségével is úgy, hogy a hasznos időben az elérhető erőforrás kapacitást 1-nek és a javítási idő alatt a kapacitást és 0-nak definiáljuk. Azonban erőforráshibák kezeléséhez az **Arena** program rendelkezik egy speciális eszközzel. Az eszköz használata előtt azonban ismerkedjünk meg az *Resource States* (erőforrás állapot) fogalmával.

Arena automatikusan négy erőforrás állapotot kezel: *Idle* (tétlen), *Busy* (foglalt), *Inactive* (nem elérhető), *Failed* (hibás). A statisztikai jelentés elkészítéséhez az **Arena** időben követi az erőforrások állapotát, azaz tudható, hogy a négy lehetséges állapot közül az adott pillanatban melyik a jellemző. Az erőforrás *Idle,* ha entitásra várakozik, amint egy entitás eléri az erőforrást, akkor az azonnal *Busy*-vé válik. Az erőforrás állapota *Inactive,* amikor az **Arena** nem engedélyezi az erőforrás terhelését, azaz a kapacitása 0. Az erőforrás állapota *Failed,* ha **Arena** hibás állapotba helyezi azt, ami szintén azt jelenti, hogy az erőforrás elérhetetlen az entitások számára.

Amikor egy hiba bekövetkezik, akkor **Arena** az egész erőforrást elérhetetlenné teszi. Ha a kapacitás 2, akkor például mindkét erőforrásegység *Failed* állapotba kerül a javítás idejére.

4.2.2 Erőforrás ütemezések

Mielőtt hozzáadjuk erőforrás ütemezésünket a javítási folyamathoz, először definiáljuk az új 16 órás napunkat. Ezt a *Run* > *Setup* menüopcióban a **Replication Parameters** fülnél megnyíló dialógusablakban tehetjük meg. Változtassuk az Hours per Day értékét 24-ről 16-ra, (hagyjuk figyelmen kívül a naptárra utaló figyelmeztetést). A párbeszédablakban a Replication Length időegységét (Time Unit) változtassuk napokra, és a Replication Length értékét 10 napra.

A definiálást elkezdhetjük a **Resource** (erőforrás) vagy **Schedule** (ütemezés) adatmodulban. Kezdjük az erőforrás modullal. A **Basic Process** panelen kattintunk a **Resource** modulra, a képernyő alján megnyíló ablakban megtekinthetjük a modell pillanatnyi erőforrásait. A *„Javitas”* erőforrás során klikkeljünk a Type (típus) oszlopra, és válasszuk az *Based on Schedule*

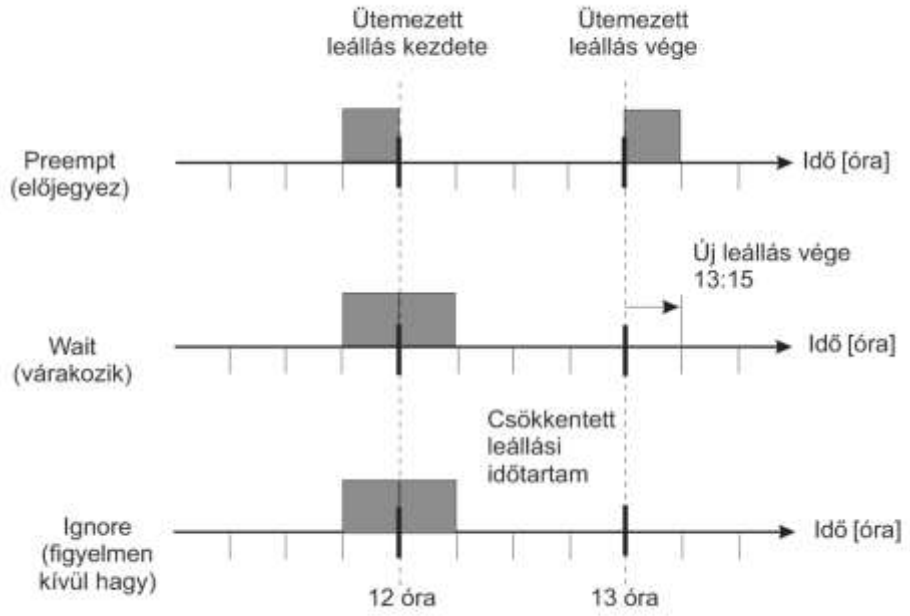
(ütemezés alapján) opciót a listából. Ha ezt az opciót választjuk, akkor az **Arena** két új oszlopot helyez az ablakba: Schedule Name (ütemezés neve) és Schedule Rule (ütemezés szabálya). Vegyük észre, hogy a *„Javitas”* erőforrás sorában a kapacitás mező halvány (nem szerkeszthető), mert a kapacitás az ütemezésen fog alapulni. A listában, értelemszerűen, a két új oszlop mezői a másik három erőforrás számára lesznek elérhetetlenek, azaz a fixkapacitású erőforrások sorában ezek a mezők szintén halvány színben jelennek meg. A következő lépésben a *„Javitas”* erőforrás sorában a Schedule Name mezőben adjuk meg az ütemezés nevét: *„Javitas utemezes”* (*4.10. képernyő*).

Resource - Basic Process

	Name	Type	Capacity	Schedule Name	Schedule Rule
1	Elokeszites A	Fixed Capacity	1	1	Wait
2	Elokeszites B	Fixed Capacity	1	1	Wait
3	Szereles	Fixed Capacity	1	1	Wait
4	Javitas	Based on Schedule	Javitas utemezes	Javitas utemezes	Ignore
	Double-click here to add a new row				

4.10. képernyő: A **Resource Data Module** a Resource Schedule választás után

Végül válasszuk a *Schedule Rule*-t (ütemezési szabály), amely az ütemezésben megadott kapacitások időzítésére van hatással. Az ütemezési szabály három lehetősége: *Wait* (várakozik), *Ignore* (figyelmen kívül hagy) és *Preempt* (előjegyez). Ha egy ütemezett kapacitás *x* egységgel csökken, és az adott pillanatban legalább *x* egységnyi erőforrás tétlen (*Idle*) állapotban van, akkor mind a három opció azonnal azt eredményezi, hogy *x* egységnyi erőforrás inaktívvá válik. Ha azonban kevesebb, mint *x* egységnyi erőforrás tétlen, akkor az ütemezési szabályok különböző módon reagálnak (a három opciót a *4.6. ábrán* ábrázoltuk).



4.6. ábra: A **Preempt**, **Wait** és **Ignore** opciók

Az *Ignore* opció esetén az erőforrás kapacitás csökkenés azonnal megtörténik, tekintet nélkül arra, hogy egy entitás az erőforrást éppen leköti vagy sem. Azonban a folyamatban lévő entitás feldolgozása befejeződik, ezért előfordulhat, hogy az erőforrás nem az ütemezett időpontban szabadul fel, hanem csak később. Amikor az erőforrásegysé(gek) felszabadul(nak), azaz befejeződik az entitás feldolgozás, akkor az erőforrásegység(ek) inaktív

állapotba kerül(nek). Előfordulhat az is, hogy a csökkentett kapacitás ütemezési ideje már lejárt, és az erőforrás pedig még nem szabadult fel. Ilyenkor a kapacitáscsökkentés be sem következik. A leírtak alapján az *Ignore* szabállyal ütemezett, csökkentett erőforrás kapacitás működési időtartama az ütemezetthez viszonyítva rövidülhet (például a *4.6. ábrán* 15 perccel), vagy el is tűnhet. Ezt a jelenséget hálózati effektusnak nevezik.

A *Wait* opció jellemzője – ahogy a neve is jelzi – az aktuális kapacitás csökkentés kezdeti időpontja addig várakozik (a kapacitáscsökkentés nem kezdődik el azonnal), amíg a feldolgozás alatt álló entitások fel nem szabadítják a lekötött erőforrásegységeiket. Az erőforrás felszabadítása után az erőforrás csökkenés időtartama egyenlő lesz az ütemezett időtartammal. Ily módon a csökkentett kapacitás ideje mindig határozott időtartamú lesz, de a csökkentett kapacitás kezdeti és befejezési időpontjai eltolódhatnak, és ezért a leállások közötti intervallumok növekedhetnek, illetve csökkenhetnek.

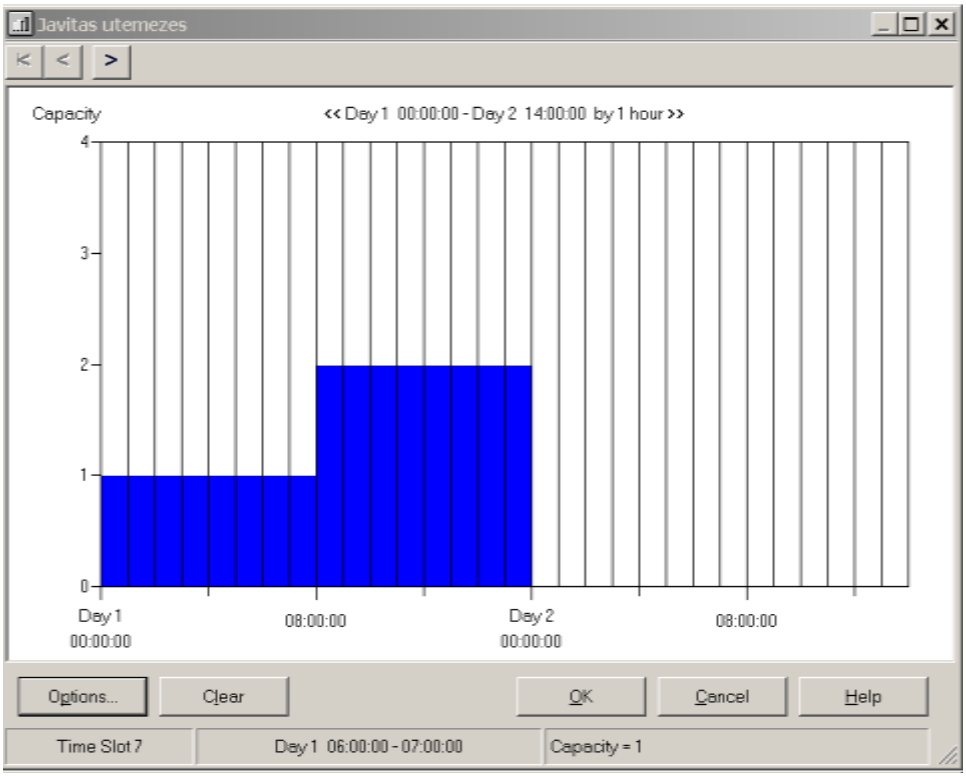
Az *Preempt* opció megkísérli előjegyzésbe venni a lekötött erőforrás utolsó egységet és kivonni azt az entitás kontrolja alól. Ha az előjegyzés sikeres, amihez egyetlen kapacitásegység is elegendő, akkor a kapacitáscsökkentés azonnal megkezdődik. Az előjegyzett entitást az **Arena** addig tartja nyilván, amíg az erőforrás újra elérhetővé nem válik. Ekkor az entitás újra allokálja az erőforrást, és folytatódik a megszakadt folyamat a folyamatidő végéig. Ez egy akkurátus és praktikus módja az ütemezések és hibák modellezésének. A gyakorlatban, sok esetben a munkadarabok megmunkálása félbeszakad a műszak végén, vagy amikor egy erőforrás meghibásodik. Ha az előjegyzés nem sikerül, vagy több mint egy kapacitásegység szükséges, akkor az *Ignore* szabály kerül alkalmazásra minden fennmaradó kapacitáson.

Azt, hogy mikor melyik szabályt használjuk, nem könnyű megválaszolni. Általánosságban azt tanácsoljuk, hogy vizsgáljuk meg alaposan az aktuális folyamatunkat, és azt válasszuk, amely legjobban leírja az aktuálisan megjelenő ütemváltozásainkat és a fellépő meghibásodásainkat. Ha az erőforrások jelentik a rendszer szűkkeresztmetszetét, akkor a választásunk jelentősen hat az eredményekre. Néha azonban nem igazán egyértelmű, mi a teendő, és mivel nincs egyértelmű szabály, csak irányelveket lehet a javasolni. (1) Ha az ütemezett csökkentett kapacitás időtartama nagyon nagy (hosszú ideig tart) a műveleti időhöz képest, akkor az *Ignore* opció lehet a megfelelő választás. (2) Ha a kapacitáscsökkenés ritkán fordul elő, és a csökkentett kapacitás időtartama rövid, akkor a *Wait* opció lehet a jó megoldás. A modellünkhöz mi az *Ignore* opciót választottuk, mert az esetek többségében a kisebb kapacitás (egy munkás) be fogja fejezni a feladatát, mielőtt elhagyná a rendszert, és plusz munkaidő (túlóra) ritkán szükséges.

A kitöltött **Resource** táblázat első öt oszlopát a *4.10. képernyő* mutatja (a táblázat további oszlopai a jobb oldalon most nem láthatók). Az egér jobb gombjával a Schedule Name oszlopban a *„Javitas utemezes”* mezőre klikkelve lenyílik egy lista, amelyből az *Edit via Dialog* opciót választva megnyitható az adatok bevitelére szolgáló **Resource** párbeszédablak.

Miután nevet adtunk az ütemezésünknek és beállítottuk az ütemezési szabályokat, az erőforrás által követendő aktuális ütemet kell definiálni. Ennek egyik módja, hogy a **Schedule** adatmodulra klikkelünk, és a megnyíló ablakban beírjuk az ütemezési adatokat a táblázatba. A táblázat első sora tartalmazza az újonnan definiált *„Javitas utemezes”*-t. A Duration (időtartam) oszlopra kattintva kinyílik a grafikus ütemezés szerkesztő (**Graphical Schedule Editor**), amely egy grafikus interface az ütemezési adatok bevitelére. A vízszintes tengelyen a naptár vagy szimulációs idő látható. (Ne felejtsük el, hogy a munkanapjaink 16 órásak a nap definícióknak megfelelően) A függőleges tengelyen az erőforrás kapacitását ábrázoljuk. Egy adatot úgy vihetünk be hogy, az *x-y* helyre kattintunk, pl. az első nap első órájában az 1 kapacitásértékre. Ekkor egy kék hasáb jelenik meg, amely a kívánt kapacitásértéket (esetünkben 1-

et) jelképezi az első órában.. A kitöltést folytathatjuk ismételt klikkelgetéssel (jobbra haladva), vagy a hasábot megfogva (a jobboldali shift gomb lenyomásával és klikkeléssel) az első nyolc órára húzhatjuk. Fejezzük be az ütemezést azzal, hogy a 9-16 óra közötti intervallumban a kapacitásértéket 2-re növeljük. Nem szükséges a második nap adatait bevinni, mivel a szimulációban az első nap adatai automatikusan ismétlődnek. A teljes ütemezés képe a *4.7. képernyőn* látható. Megjegyezzük, hogy a függőleges tengelyen a kapacitás 10-ről 4-re csökkentéséhez az *Options* gombot használtuk. Az **Options** párbeszédablakban más paramétereket is megváltoztathatunk, mint például, melyen hosszú legyen az időtengely osztása (*Time slot duration*), hány osztás (*Range time slots*) legyen, vagy az ütemezés az elejétől ismétlődik vagy fixkapacitással folytatódik a végtelenségig.

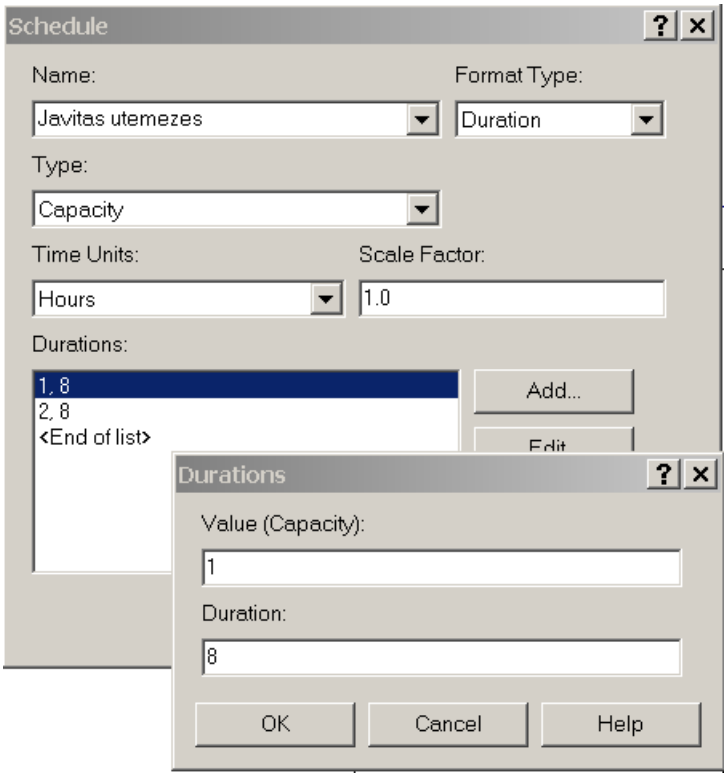


4.7. képernyő: A grafikus ütemezés szerkesztő a *„Javitas utemezes”*-sel

Az adatokat manuálisan is beírhatjuk úgy, hogy a **Schedule** adatmodul ablakában a Durations (időtartamok) oszlopra klikkelünk az egér jobb gombjával és kiválasztjuk az **Edit via Dialog** (szerkesztés párbeszéddel) opciót. Ha ezt az opciót választjuk, akkor a megnyíló ablakban először az ütem nevét kell kiválasztani, majd az *Add* (hozzáadás) gombra klikkelve megnyithatjuk a Durations (időtartam) ablakot. Itt definiálhatjuk az ütemezés elemeit, a kapacitás-időtartam párokat. Esetünkben két érték pár: 1, 8 és 2, 8 (*4.12. képernyő*). Ez azt jelenti, hogy az első 480 percben a kapacitás 1 lesz, majd a következő 480 percben 2. A program futásakor, a szimuláció teljes időtartam alatt ez az ütemezés ismétlődik. Természetesen annyi kapacitás-időtartam párunk lehet, amennyit a rendszerünk pontos modellezése igényel. Például, beiktathatunk munkaszüneteket vagy ebédidőt is az ütemtervbe. Felhívjuk a figyelmet egy veszélyre vagy tulajdonságra, amit nem árt szem előtt tartani. Ha bármely párnak nem adjuk meg az időtartamát, akkor az alaphelyzetben végtelen nagy lesz. Ez azzal jár, hogy a szimuláció hátralévő időszakában a kapacitással rendelkező erőforrás végig le lesz kötve. Mivel a modellünkben az időtartamok pozitív számok, az ütemezés a szimuláció teljes futása alatt ismétlődik.

Ha az ütemezés elkészítéséhez a grafikus ütemezés szerkesztőt (**Graphical Shedule Editor**) használjuk, és utána megnyitjuk a párbeszédablakot, akkor láthatjuk, hogy az értékek (kapaci-

tás-időtartam párok) automatikusan beíródtak a Durations mezőbe. Jegyezzük meg hogy, nem használhatjuk a grafikus ütemezés szerkesztőt, ha az időtartamaink nem egész számok, vagy a bejegyzéseink matematikai összefüggések (például az időtartam egy valószínűségi változó).



4.12. képernyő: A Schedule adatmodul dialógus ablaka

4.2.3 Erőforrás meghibásodások

Az ütemezés arra alkalmas, hogy az erőforrásaink elérhetőségét az előre tervezhető változásoknak megfelelően modellezzük. Az ilyen változások okai a műszakváltások, a szünetek, a szabadságok, a megbeszélések stb. lehetnek. Ezzel szemben a hibaeseményekkel elsősorban a véletlenszerűen bekövetkező erőforrás kieséseket modellezzünk. A hibák definiálását elkezdhetjük az **Resource** (erőforrás) vagy a **Failure** (meghibásodás) adatmodulban. A **Failure** adatmodul az **Advanced Process** panelen található (amit, lehet, hogy be kell kapcsolnunk a menüben a *File>Template Panel>Attach* útvonalon, ha az nem elérhető a **Project Bar**-on). Mivel az ütemezés programozásakor a **Resource** modullal kezdtünk, ezért kezdjük most szerelőgép meghibásodás modellezését a **Failure** adatmodullal.

Ha láthatóvá kívánjuk tenni az **Advanced Process** panelt a **Project Bar**-on, akkor klikkeljünk a nevére, aztán klikkeljünk a **Failure** adatmodulra. A megnyíló modul ablaka üres, nincsenek bejegyzések a táblázatban. Klikkeljünk duplán a jelzett területre, hogy új sort adjunk a táblázathoz. Válasszuk ki az alapértelmezett nevet (default) és írjuk át egy jellemző hibamegnevezésre, esetünkben *„Szerelogep hiba”*-ra, majd válasszuk ki a hiba típusát a (Type mező) legördülő menüjéből, ami lehet *Count* (számlált) vagy *Time* (idő) alapú (*4.13. képernyő*). Az erőforrás számlált alapú hibája meghatározott számú entitás áthaladása után következik be. Ez a szám lehet állandó, vagy generálható egy kifejezésből. A számlált alapú meghibásodások elég gyakoriak az iparban. Például a szerszámcsere, tisztítás, gépbeállítás inkább a megmunkált munkadarabok számától, mint az eltelt időtől függenek. Habár ezeket normál esetben nem tekintjük hibának, azonban időszakos megjelenésükkel az erőforrást megállásra késztetik. Gyakran használjuk azonban az időalapú hibákat a modellezésben, mert a meghibásodási

adatok gyűjtésének is az idő az alapja. A modellezett problémára is időalapú meghibásodás a jellemző.

Klikkeljünk ezért a típus mezőre, és válasszuk a *Time* (idő) opciót. Ha ezt megtettük, akkor a jobb oldalon a sorban megjelennek a kívánt adatok, és láthatjuk a különbséget a két opció között. A szerelőgép Up Time (meghibásodás mentes idő) és Down Time (meghibásodási idő) időértékei exponenciális eloszlást követnek, 120 és 4 perces középértékkel. Az Up Time és a Down Time mértékegységét állítsuk át óráról percre (*4.13. képernyő*).

Az utolsó mezőben (Uptime in this State Only) lehetőségünk van arra, hogy csak az erőforrás hibaállapotát definiáljuk „számláltként". Ha ez a mező alapértelmezett (defaulted), akkor az összes lehetséges állapotot megfigyeljük. Ennek a tulajdonságnak a használata nagymértékben attól függ, hogy az adatainkat hogyan gyűjtjük, illetve modellünk naptári ütemezésétől. A legtöbb hibaadat önállóan megfigyelt adat, például ha csak a hiba időpontja megfigyelt. Ebben az esetben a szabadságok, ebédszünetek, holtidők a meghibásodások közötti időbe tartoznak, és ekkor a mezőt alapbeállítással kell használni. Csak akkor kell ezt az opciót használni, ha a meghibásodások közötti idők közvetlenül egy specifikus állapothoz kapcsolhatók. Sokszor az eszközforgalmazók adatokat szolgáltatnak a várható meghibásodás mentes működési időtartamról, ez esetben ajánlatos ezt az opciót használni, és meghatározni a *Busy* (foglalt) állapotot. Megjegyezzük, ha ezt az opciót használni kívánjuk, akkor definiálnunk kell a *Busy* (foglalt) állapotot a **StateSet** (állapot beállítás) adatmodulban, amely az **Advanced Process** panelen található. Az erőfeszítéseink eredményeként kialakuló ablak képe a szerelőgép hibával a *4.13. képernyőn* látható.

Failure - Advanced Process

	Name	Type	Up Time	Up Time Units	Down Time	Down Time Units	Uptime in this State only
1	Szerelogep hiba	Time	EXPO(120)	Minutes	EXPO(4)	Minutes	

Double-click here to add a new row.

4.13. képernyő: A **Failure** adatmodul a *„Szerelogep hiba"*-val

Miután befejeztük a *„Szerelogep hiba"* definiálását, hozzá kell csatolnunk azt a *„Szereles"* erőforráshoz. Nyissuk meg a **Resource** adatmodult és klikkeljünk *„Szereles"* erőforrás során a Failures (hibák) oszlopra. Ez kinyit egy új ablakot a **Failures** (hibák) táblázattal (*4.14. képernyő*). Egy új sor hozzáadásához klikkeljünk kétszer az ablak területére, és a Failure Name (hibanév) mezőben a legördülő menüből válasszuk a *„Szerelogep hiba",* a Failure Rule (hibaszabály) mezőben a *Wait* opciót. Az utóbbi szabályt a *Wait, Ignore, Preempt* opciók közül választottuk. Ezek az opciók az ütemezés szabályaival megegyeznek és azokhoz hasonlóan működnek. Visszatérve a hibaszabály választás előírásaihoz, mivel a várható hibamentes idő 120 perc, jóval nagyobb a hiba időtartamánál (4 perc), ezért várakozás (*Wait*) opciót alkalmazzuk. A végső táblázatnézet *4.13 képernyőn* látható. Ha több erőforrásunk van ugyanazzal a hiba profillal, akkor azok hivatkozhatnak ugyanarra a hibanévre, mivel a szimuláció futtatása alatt a hiba generálásához mindegyik erőforrás saját, egymástól független véletlen mintát használ.

Failures

	Failure Name	Failure Rule
1	Szerelogep hiba	Wait

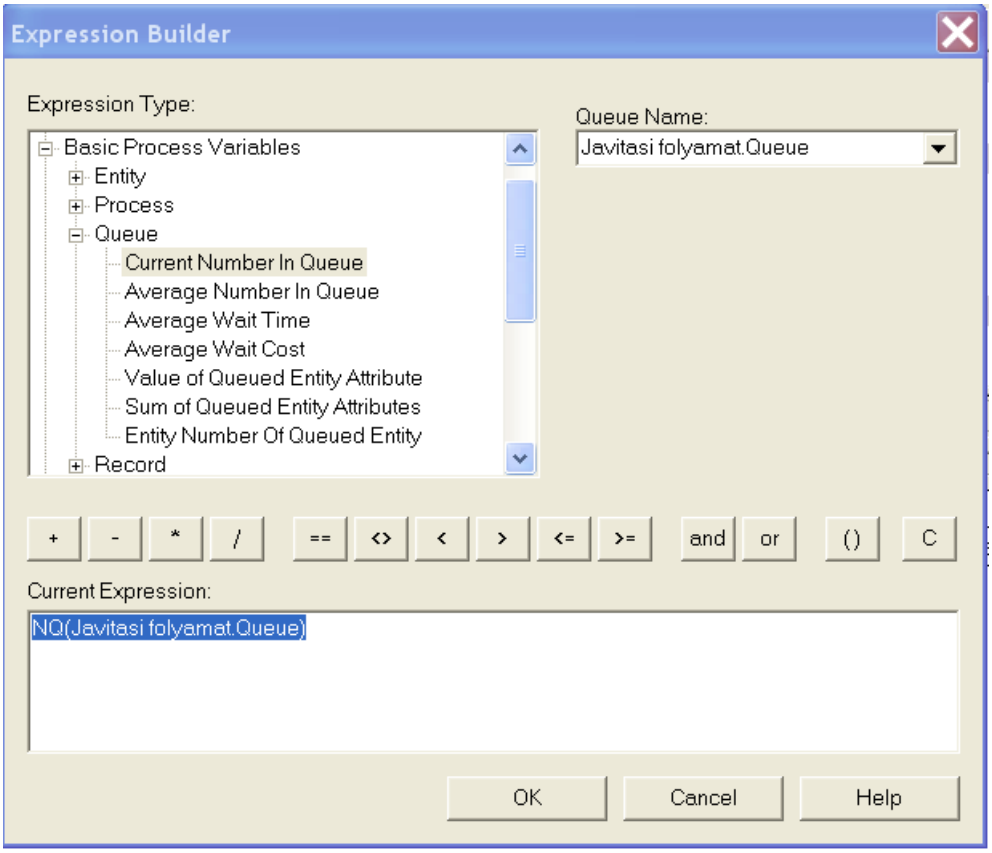
Double-click here to add a new row.

4.14. képernyő: A **Resource** adatmodulban az *„Szerelogep"* **Failures** táblázata

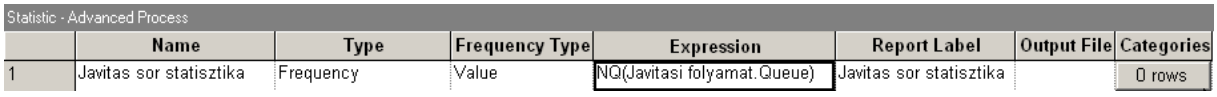
4.2.4 Gyakoriságok

A gyakoriság fogalmát arra használjuk, hogy meghatározzuk egy **Arena** változóval, kifejezéssel, vagy egy erőforrás állapotával kapcsolatos, időben folytonos történések előfordulásá-

nak gyakoriságát. A gyakorisági statisztika fogalmát használhatjuk információszerezésre is, pl. arra, hogy a javító részlegbe telepítendő állványok szükséges számának megállapításához adatokat nyerjünk. Egészen pontosan arra vagyunk kíváncsiak, hány állványt kell vásárolni ahhoz (egy állványon 10 munkadarab tárolható), hogy a várakozási folyamat ideje alatt biztosítva legyen a szükséges tárolókapacitás. Erre a kérdésre a javítórészlegben képződő sor állapotai adhatnak választ, azok az állapotok, pontosabban ezek előfordulásának gyakoriságai érdekelnek bennünket, amikor a sorban álló munkadarabok száma: 0 (nincs szükség állványra), nullánál nagyobb, de 10-nél kevesebb (1 állvány szükséges), vagy 10-nél nagyobb, de 20-nál kevesebb (2 állvány szükséges) stb.



4.15. képernyő: Az **Expression Builder** (kifejezés szerkesztő) dialógus ablaka



4.16. képernyő: A **Statistic** adatmodul

A gyakorisági statisztika az **Advanced Process** panelen található **Statistic** adatmodulban érhető el. Erre a modulra kattintva egy új ablak jelenik meg, amely kezdetben üres. Kattintsunk duplán az ablakra, hogy új sort adjunk a táblázathoz. Elsőként adjuk meg a statisztika nevét: *„Javitas sor statisztika”*. A Type (típus) oszlopban a lenyíló menüből válasszuk *Frequency* (gyakoriság) opciót, a Frequency Type oszlopban pedig az alapértelmezett *Value* értéket. A típus választásakor a Report Label (jelentés címke) mező neve automatikusan *„Javitas sor statisztika”*-ra változik, amelyet ennek a kimenetnek a jelölésére használunk majd a jelentésben (*4.16. képernyő*).

Meg kell még határoznunk azt az összefüggést, amely a *„Javitasi folyamat”* modul sorában várakozó entitások számát reprezentálja. Ahhoz, hogy elérjük ezt az információt, ismernünk

kell a sorban állók számát (a sor hosszát) tartalmazó **Arena** változó nevét (*NQ*). A sor nevét (*Javitasi folyamat.Queue*) a **Queue** (sor) adatmodulból kaphatjuk meg, amely ugyancsak a **Basic Process** panelen található.

A kifejezés így, amelyet be akarunk írni az *NQ*(*Javitasi folyamat.Queue*). Ekkor megkérdezhetnénk, hogy miért írjuk le mindezt? Egy tapasztalt SIMAN felhasználó számára ez nyilvánvaló, de egy új felhasználó számára közel sem az. Szerencsére az **Arena** egyszerű eszközt kínál az ilyen típusú összefüggések meghatározására, és nincs szükség a titkos szavak ismeretére (mint pl.: *NQ*). Vigyük a kurzort az üres Expression (kifejezés) mező fölé, klikkeljünk a jobb egérgombbal és a megnyíló listából válasszuk a **Build Expression** (kifejezés szerkesztés) opciót. Ez megnyitja az **Arena Expression Builder** (kifejezés szerkesztő) ablakát, amely a *4.15. képernyőn* látható. Az **Expression Type** (kifejezés típus) kategória alatt a **Basic Process Variables** kategória alkategóriájaként megtalálhatjuk, a **Queue**–t (sor). Klikkeljünk a + jelre, hogy kiterjesszük a lehetőségeket, és válasszuk a *Current Number in Queue* (pillantnyi entitás a sorban) opciót. A műveletek elvégzése után két dolog történik: a Queue. Name (sor név) megjelenik az ablak jobb oldalán: *„A egyseg elokeszitesi folyamat.Queue”*, és Current Expression (pillanatnyi kifejezés) mezőbe beíródik a sor neve. Esetünkben a megjelenő kifejezés az *NQ*(*A egyseg elokeszitesi folyamat.Queue*), amely nem az amit akarunk. Használjuk a legördülő menüt a Queue.Name mezőnél, és válasszuk ki a „*Javitasi folyamat.Queue*” sort. Ha az *OK* gombra klikkelünk, az ablak bezáródik és a kifejezés automatikusan bekerül abba a mezőbe, ahonnan az kifejezés szerkesztőt megnyitottuk (*4.16. képernyő*).

Az egér jobb gombjával klikkelhetünk bármely mezőre, amelybe egy kifejezést akarunk beírni az **Expression Builder** segítségével. Például használhatjuk az **Expression Builder**-t, arra is, hogy megtaláljuk a pillanatnyi szimulációs idő (TNOW) kifejezést. Az **Expression Builder**–ben szerkeszthetünk összetett kifejezéseket a funkció gombok használatával vagy közvetlen beírással a **Current Expression** mezőbe (ha tudjuk mit kell beírni).

A *„Javitas sor statisztika”* beállítás utolsó lépése, a megjelenítendő értékek osztályhatárainak a definiálása. Ezt megtehetjük a **Categories** (osztályok) oszlopban (*4.16. képernyő*). Klikkeljünk a „0 rows” mezőre, amely megnyitja a **Categories** ablakot, ahol új sorok hozzáadásával új osztályokat definiálhatunk. Az osztályok lehetnek tartományok vagy konstans értékek.

Categories

	Constant or Range	Value	High Value	Category Name	Category Option
1	Constant	0		0 Allvany	Include
2	Range	0	10	1 Allvany	Include
3	Range	10	20	2 Allvany	Include
4	Range	20	30	3 Allvany	Include
5	Range	30	40	4 Allvany	Include
	Double-click here to add a new row.				

4.17. képernyő: Kategóriák a *„Javitas sor statisztika”* gyakoriság statisztikájához

A *4.17. képernyő* az első öt osztály bejegyzéseit mutatja. Az első bejegyzésnél a sor mérete **Constant** (állandó) és 0 értékű (a javítási területen nincs várakozó munkadarab és 0 állvány szükséges). A következő bejegyzések Range (tartomány) jellegűek, és 1 állványra, 2 állványra stb. vonatkozó kategóriák. Ha a sor hossza meghaladná a 40 db-os munkadarabszámot, akkor az **Arena** egy tartományon kívül (out of range) kategóriát hoz létre a kimeneti jelentésben (**output report**). A Range (tartomány) kapcsán jegyezzük meg, hogy az Value (érték) nem tartozik bele a tartományba, de a High Value (felső érték) igen. Például a harmadik sorban a Value = 10 és High Value =20 definiálja a tartományt, ami szigorúan 10-nél nagyobb és kisebb-egyenlő 20 számok halmazát jelenti.

A **Statistic** adatmodul elhagyása előtt még további információt kérhetünk a *„Szereles”* erőforrásról. Ha lefuttatjuk a modellünket a *„Szereles”* kihasználási mutatói megtalálhatók a **Category Overview** jelentésben. Azonban nem kapunk specifikus jelentést arról, hogy az erőforrás mennyi időt töltött hibás állapotban. Ilyen statisztikát úgy kérhetünk, hogy egy új sort adunk a Statisztikai adatmodulhoz, ahogy azt a *4.18. képernyőn* látjuk. Ehhez a statisztikához beírjuk a nevet (*„Szerelogep allapot”*), a típust (*Frequency*) és a frekvencia típust (*State*). Végül a Resource Name mezőben a *„Szereles”* erőforrást választjuk. Ez olyan statisztikát eredményez számunkra, amelynek alapja a *„Szereles”* erőforrás három lehetséges állapota (*Busy, Idle* és *Failed*).

Statistic - Advanced Process

	Name	Type	Frequency Type	Expression	Resource Name	Report Label	Output File	Categories
1	Javitas sor statisztika	Frequency	Value	NQ(Javitasi folyamat.Queue)		Javitas sor statisztika		5 rows
2	Szerelogep allapot	Frequency	State		Szereles	Szerelogep allapot		0 rows

4.18. képernyő: A **Statistic** adatmodul a *„Szerelogep allapot”* statisztikával

Mielőtt véglegesen futtatnánk a modellt, azt ajánljuk, hogy ellenőrizzük a *Run >Run Control >Batch Run* (nincs animáció) opcióval, amely jelentősen csökkenti a modell futásához szükséges időt. Kicsit lassabb, de alternatívát jelent a *Run > Fast-Forward* futási mód választása. Ne felejtsük el, a program futását bármikor megállíthatjuk, hogy ellenőrizzük a folyamat előrehaladását.

4.2.5 A 4.2 modell eredményei

A *4.15. táblázat* utolsó oszlopa néhány válogatott adatot tartalmaz a 4.2. modell futása során nyert eredményekből, és az összehasonlíthatóság érdekében ugyanezeket az adatokat 4.1. modellhez készült jelentésekből is kiszedtük. Az összes adatot két tizedesre kerekítettük, kivéve a kihasználási mutatókat, ahol négy tizedes pontossággal adtuk meg a számokat.

Az eredmények a 4.2. modellben különböző okok miatt eltérnek a 4.1. modell eredményeitől. Ezen okok egyike a szimuláció időtartama. A 4.2. modellben 10 × 16 óra munkanap, azaz 160 óra a futási idő, míg a 4.1. modellben 4 × 8 = 32 óra. A másik ok a modelldefiníció változása a szerelési és a javítási területen. Az előbbinél a szerelőgép meghibásodásának modellezése, az utóbbinál a szerelőkapacitás növekedése jelenti az eltérést.

A 4.1. modellből a 4.2. modellbe való átmenet nem okoz semmilyen változást a modellben az *A* és a *B* egységek előkészítési folyamataiban. A különbségek az eltérő szimulációs időnek és a véletlenszerű ugrásoknak köszönhetőek. A *B* egységek előkészítési folyamatának soraira vonatkozó eredmények közötti különbségek azonban figyelemreméltóak. A magyarázat lehet az, hogy az idő előre haladásával egyre több entitás torlódik ezen a területen, vagy az eredmények bizonytalanok. Nem tudjuk melyik állítás igaz (és még sok oka lehet a kimeneti statisztika változásának, amelyet most itt nem tárgyalunk).

A 4.2 modell esetében a szerelőgéphez tartozó sor statisztikák (az átlagos várakozási idők és a sor hosszúságok) jelentősen több torlódást mutatnak. Ez azzal magyarázható, hogy a modellben definiáltuk a szerelőgép hibáit, ami miatt szerelőgép időnként nem működik, és a hiba időtartama alatt a munkadarabok várakozásra kényszerülnek, azaz sorban állnak. A szerelőgép kihasználási statisztikái nem nagyon különböznek egymástól a két modellben, habár, amikor a szerelőgép hibás állapotban van, akkor nem használható, így eszek az időszakok nem számítanak bele a szerelőgép kihasználási statisztikájába.

Eltérően a szerelőgéptől, a *„Javítasi folyamat”* sokkal egyenletesebb a 4.2 modellben, ami annak köszönhető, hogy a *„Javitas”* erőforrás kapacitását duplájára növeltük a második nyolc órás műszakban. Ez átlagosan 50% kapacitás növekedést jelent, de ennek megfelelően az *„Ja-*

vitas" erőforrás kihasználási mutatója alaposan csökkent (részletesebben a különböző kihasználtságokról később szólunk).

4.15. táblázat

A 4.1. és a 4.2. modellek eredményeinek az összehasonlítása

Eredmény megnevezése	Modell 4.1	Modell 4.2
Average Waiting Time in Queue (Átlagos várakozási idő a sorban)		
A egység előkészítési folyamat	14,62	19,20
B egység előkészítési folyamat	29,90	51,42
Szerelési folyamat	2,52	7,83
Javítási folyamat	456,35	116,25
Average Number Waiting in Queue (Átlagos entitás szám a sorban)		
A egység előkészítési folyamat	3,17	3,89
B egység előkészítési folyamat	3,50	6,89
Szerelési folyamat	0,86	2,63
Javítási folyamat	12,95	3,63
Average Time in System (Átlagosan eltöltött idő a rendszerben)		
Szállított egységek	28,76	47,36
Javított egységek	503,85	203,83
Selejtezett egységek	737,19	211,96
Instantaneous Utilization of Resource (Erőforrás pillanatnyi kihasználtsága)		
A egység előkészítési folyamat	0,9038	0,8869
B egység előkészítési folyamat	0,7575	0,8011
Szerelési folyamat	0,8595	0,8425
Javítási folyamat	0,9495	0,8641
Scheduled Utilization of Resource (Erőforrás ütemezett kihasználtsága)		
A egység előkészítési folyamat	0,9038	0,8869
B egység előkészítési folyamat	0,7575	0,8011
Szerelési folyamat	0,8595	0,8425
Javítási folyamat	0,9495	0,8567

A háromféle kilépő termék (szállított, javított és selejtezett egységek) rendszerben való átlagos tartózkodási idejét tekintve, egyértelműen kitűnik, hogy a változtatások hatása a szerelési és a javítási folyamatra jelentősebb. Minden darabot érint a lassú szerelési folyamat, ami a szállított egységeknek a rendszerben töltött átlagos átfutási idejét növeli. A javítási folyamat kapacitás növelésének köszönhetően a javított és a selejtezett egységeknek a rendszerben tartózkodási ideje pedig jelentősen csökken.

A továbbiakban az **Arena** jelentés kihasználási mutatóit az eddigieknél kicsit mélyebben fogjuk vizsgálni. Mindenegyes erőforrásról az **Arena** két kihasználási statisztikát készít, az egyik az **Instantaneous Utilization** (pillanatnyi kihasználtság) a másik a **Scheduled Utilization** (ütemezett kihasználtság).

Az **Instantaneous Utilization**-t (pillanatnyi kihasználtság):

$$\frac{1}{T}\int_0^T U(t)dt = \frac{1}{T}\int_0^T \frac{B(t)}{M(t)}dt,$$

ahol:

$U(t)$ az erőforrás kihasználtsága a t időpontban $B(t)/M(t)$, ha $M(t)>0$,

ha az $M(t)=0$, akkor az $U(t)= 0$,

$B(t)$ a foglalt erőforrás kapacitás a t időpontban,

$M(t)$ az ütemezett erőforrás kapacitás a t időpontban,

T a szimuláció időtartama.

A jelentésben megjelenő **Instantaneous Utilization** tehát az idő szerint súlyozott átlagos kihasználtságot jelenti. Fontos tudni, hogy az **Arena** számol azokkal az időközökkel is, amikor nincs ütemezett kapacitás, $M(t)=0$, azaz a T idő tartalmazza azokat az időközöket is, amikor a kihasználtság, $U(t) =0$. Ez statisztika hasznos lehet, mert a kihasználtságot az idő függvényében követi, azonban óvatosan kezelendő, mert bizonyos körülmények között megtévesztő eredményeket produkálhat (részletesebben később).

Scheduled Utilization (ütemezett kihasználtság) egyenlő:

$$\frac{\frac{1}{T}\int_0^T B(t)dt}{\frac{1}{T}\int_0^T M(t)dt} = \frac{\int_0^T B(t)dt}{\int_0^T M(t)dt},$$

azaz, az időben átlagosan foglalt erőforrás kapacitás és az időben átlagosan ütemezett erőforrás kapacitás hányadosa. Ebben az esetben a nullával való osztás nem jelentkezik problémaként, mivel az ütemezett erőforrásnak létezik elérhető kapacitása (egyetlen helyzetben állhat elő probléma, ha az erőforrás még nem jelent meg a modellben).

Különböző feltételek mellett a kétféle kihasználási mutató mindegyike hasznos információt nyújthat. Ha azonban egy erőforrás kapacitása állandó a szimuláció alatt (pl. az erőforrás nem hibásodik meg, és nem kerül inaktív állapotba), a jelentésben a kétféle kihasználtság értéke azonos. Ezt igazolják a *4.15. táblázat* adatai. Például a 4.2 modellben, ahol erőforrások állandó kapacitásúak (***A*** és ***B*** egység előkészítési folyamat), ott a mutatók megegyeznek, és ahol az erőforrások kapacitásai változnak (szerelési és javítási folyamat), ott a mutatók jelentősen eltérnek. Ha egy ütemezés kapcsolódik egy erőforráshoz, és az ütemezett kapacitások nullák vagy pozitív konstansok (például: 1), akkor a jelentésben megjelenő **Instantaneous Utilization** (pillanatnyi kihasználtság) értékek azt mutatják, hogy az erőforrás átlagosan mennyire volt foglalt az egész futtatás ideje alatt (beleértve a nulla kapacitású, azaz a nulla kihasználtságú periódusokat is). Az $U(t)$ függvény görbéjéről leolvasható, hogy az ütemezés mennyire követi az erőforrás elvárásait, ez hasznos lehet pl. személyzet tervezés vizsgálatakor. A jelentésben szereplő másik mutató, a **Scheduled Utilization** (ütemezett kihasználtság) azt jelzi, hogy átlagosan mennyire volt foglalt az erőforrás az idő elteltével, amikor elérhető volt, azaz a kapacitása nem volt nulla.

Például tekintsük azt a helyzetet, amikor egy erőforrást úgy ütemezünk, hogy a T idő 2/3 részében elérhető az kapacitása $M(t)=1$, és a T idő fennmaradó 1/3 részében nem elérhető, azaz a kapacitása $M(t)=0$. Feltételezzük továbbá, hogy az erőforrás a teljes elérhetőségi idő 50%-a alatt, azaz

$$1/2\times 2/3\ T = 1/3\times T$$

ideig foglalt $B(t)=1$, és ugyanennyi ideig nem működik $B(t)=0$. E példára készített jelentésben az **Instantaneous Utilization** (pillanatnyi kihasználtság) mutató értéke:

$$\frac{1}{T}\int_0^T U(t)\,dt = \frac{1}{T}\int_0^T \frac{B(t)}{M(t)}\,dt = \frac{1}{T}\frac{(1/3)\times 1 + (1/3)\times 0}{1}T = 0{,}3333.$$

Ez azt mutatja, hogy az erőforrás az egész futtatási időhöz (T) viszonyítva az idő 1/3 részében lesz kihasznált.

A **Scheduled Utilization** (ütemezett kihasználtság) mutató:

$$\frac{\int_0^T B(t)dt}{\int_0^T M(t)dt} = \frac{[(1/3)\times 1 + (1/3)\times 0]\times T}{2/3\times T} = 0{,}5000,$$

amely szerint az erőforrás elérhetőségi idejéhez viszonyítva az erőforrás 1/2 részben lesz kihasznált. A példában az **Instantaneous Utilization** (pillanatnyi kihasználtság) mutató kisebb vagy egyenlő mint a **Scheduled Utilization** (ütemezett kihasználtság) mutató.

Az előzőek módosulnak, azaz finomítani kell az eszmefuttatásunkat, ha az Schedule Rule–t (ütemezési szabály) nem *Wait*-nek (várakozik) választjuk, ekkor ugyanis lehetséges, hogy az **Scheduled Utilization**-ra (ütemezett kihasználtság) 1-nél nagyobb értéket kapunk. Ha például az *Ignore* (figyelmen kívül hagy) opciót választjuk, és az erőforrás éppen foglalt, amikor az ütemezés szerint elérhetetlenné válik (a kapacitása 0 lesz), akkor az erőforrás kapacitása 0-ra csökken, de az erőforrás foglalt marad addig, amíg fel nem szabadul, azaz az aktuális folyamatot be nem fejezi. Mondjuk futtatási idő 10 óra, de az erőforrás ütemezése szerint csak az első 5 órában érhető el. Lehetséges azonban, hogy az erőforrás foglalt lesz a 0 időpillanattól 6 órán keresztül. Az erőforrás kapacitása az 5-dik órában ugyan 0-ra csökken, de az a folyamat befejezése miatt foglalt maradt a 6-dik óráig. A jelentésbe kerülő **Instantaneous Utilization** (pillanatnyi kihasználtság) 0,6, mivel a 10 órából 6 órát volt kihasználva. A **Scheduled Utilization** (ütemezett kihasználtság) értéke viszont 1,2, tekintve, hogy az ütemezés szerint csak 5 órán keresztül volt elérhető, de 6 órát volt kihasznált.

19:05:45 **Frequencies** június 14, 2007

Elektronikus egység szerelés és ellenőrzés Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 9 600,00 Time Units: Minutes

Javitas sor statisztika	Number Obs	Average Time	Standard Percent	Restricted Percent
0 Allvany	41	69.4722	29.67	29,67
1 Allvany	52	119.96	64.98	64,98
2 Allvany	12	42.8210	5.35	5,35

Szerelogep allapot	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	697	11.6044	84.25	84,25
FAILED	68	4.1861	2.97	2,97
IDLE	640	1.9173	12.78	12,78

4.7. ábra: A 4.2. modellhez készített **Arena** gyakorisági jelentés

A következő és egyben utolsó felmerülő komplikáció, amikor egy ütemezés kapcsolódik az erőforráshoz és az ütemezett kapacitások különböző pozitív értékek között változnak az időben (pl.:1, 7, 4 stb.), eltérően a 0 és egy pozitív konstans közötti változástól. Ha ez a szituáció áll fenn, akkor nyomatékosan ajánljuk, hogy ne használja a **Instantaneous Utilization** (pillanatnyi kihasználtság) mutatót, még akkor se, ha az meg fog jelenni a jelentésben. A kapacitástól, az időtartamoktól és a használattól függően ez a kihasználási mutató lehet kisebb egyenlő vagy nagyobb a **Scheduled Utilization**-i (ütemezett kihasználtság) mutatónál. Ez esetben azt tanácsoljuk, hogy helyette használja a gyakorisági statisztikát, amely részletes és pontosabb

információt nyújt az erőforrás kihasználtságáról. a Bővebb ismertetés található a **Help** témák között, a „**Resource Statistics: Instantaneous Utilization Vs. Scheduled Utilization”** (Erőforrás statisztikák: a pillanatnyi kihasználtság szemben a ütemezett kihasználtsággal) cím alatt. Így például a 4.2 modellben az *„Javitas”* erőforrás esetén, az tanácsolható, hogy használjuk a **Scheduled Utilization** mutató 0,8567 értékét az **Instantaneous Utilization** mutató 0,8641 értéke helyett (*4.15. táblázat*).

Az új gyakorisági statisztikák nem részei a **Category Overview** (kategória áttekintés) jelentésnek. A **Frequencies report** (gyakorisági jelentésekre) kell klikkelni **Project bar**-on a **Reports** panelen. Az eredmények a *4.7. ábrán* láthatók.

Az első szekció azokat a statisztikákat mutatja, amelyeket azért definiáltunk, hogy a *„Javitas”* területén meghatározzuk a szükséges állvány számot. Ebben a különleges futtatásban az *„Javitas”* modulhoz tartozó sorban a statisztika szerint 20-nál több termék nem várakozott (nincs listázott adat 3 vagy 4 állványról), és tíznél több is csak az idő 5,35%-ban várakozott. Ebből következik, hogy egy vagy legfeljebb kettő állvány elegendő. A *„Szerelogep allapot”* statisztikák megadják, hogy a szerelőgép az idő hány százalékában volt *Busy* (foglalt), *Failed* (hibás) vagy *Idle* (tétlen) állapotban volt.

Egy utolsó fontos megjegyzés a gyakorisági statisztikákhoz. Eredményeinkben a két utolsó oszlop, a Standard és a Restricted Percent megegyezik. Ki lehet zárni a szelektív eredményeket és a különbségeket a két oszlop között. Például ha kizárjuk a *Failed* (hibás) állapotot a szerelőgép gyakoriságából, a Standard Percent maradna ugyanaz, de a Restricted Percent oszlop értékei csak *Busy* és *Idle* lennének, amelyek összege 100% lenne.

4.3. Az animáció kiterjesztése és bővítése (4.3 modell)

Ebben a fejezetben eddig azokat az alapértelmezett animációkat használtuk, amelyeket az alkalmazott modulok felkínáltak. Az alap animáció általában elegendő annak eldöntésére, hogy a modellünk megfelelően működik-e. A modellünket azonban sokszor szeretnénk még realisztikusabb külsővel felruházni, mielőtt megmutatnánk azt a döntéshozóknak. Az animáció realisztikusabbá tétele általában nagyon egyszerű és nem túl időigényes. Nagyvonalakban a ráfordítandó idő attól függ, hogy milyen részletesen akarjuk megjeleníteni a folyamatot, illetve milyen a hallgatóságunk összetétele. Általános megfigyelés, hogy az érintettek meggyőzése érdekében, annál több időt töltünk az animáció készítésével minél magasabban szinten helyezkednek el a szervezetben a hallgatóság tagjai. Látni fogjuk, hogy az animáció csinosítása nemcsak örömmel jár, hanem akár megszállottsággá is válhat. Tehát ezzel a tudattal fedezzük fel, mit is tehetünk az ügy érdekében.

A 4.2 modellt fogjuk módosítani 4.3 modell néven, és ezt kezdjük az alap animáció megtekintésével. Ennek az animációnak három komponense van: *entitások*, *sorok*, és *változók*. Az entitások, amelyek az **Entity** adatmodulban találhatók, akkor láthatók, mikor egyik modulból a másikba mozognak, illetve amikor egy sorban állnak. Minden egyes alkalmazott **Process** modulhoz az **Arena** automatikusan hozzáad egy animációs sort, amely a futtatás alatt megjeleníti a sorban várakozó entitásokat. A modulban tartózkodó, illetve a modult elhagyó entitások számát jellemző változókat (**Variables**) szintén az **Arena** helyezi a modellbe.

Az animációk a modellben a modul elhelyezésével egyidejűleg jelennek, és kétféle módon kötődnek a modulokhoz. Egyrészt az animációs objektumok nevei, illetve azonosítói a modulok dialógusablakában megadott értékekből származtatottak, és az animáció dialógusablakában közvetlenül nem is lehet megváltoztatni azokat. Másrészt a modulok mozgatásakor a hozzájuk tartozó animációs objektumok a modulokkal együtt mozognak. Ez a kapcsolat azonban megszüntethető. Ha például, azt akarjuk, hogy a modulok mozgatásakor az animációk helyben maradjanak, akkor a modul mozgatásával egyidejűleg a SHIFT gombot tartsuk lenyomva.

Bizonyos esetekben, az áttekinthetőség érdekében célszerű az animációt a logikai folyamattól távolabb, a modulablak egy másik területén elhelyezni és „széthúzni”. Esetleg érdemes önálló névvel jelölt nézetet készíteni az előre és visszalépések megkönnyítése érdekében. Ha egy animációt teljesen el akarunk választani a modultól, amelyhez eredetileg tartozott, akkor a *Cut and Paste* módszerrel helyezzük a vágólapra, majd vissza a modulba. Az animáció minden jellemzőjét meg fogja őrizni, de a továbbiakban nem fog a modulhoz kapcsolódni. Másik alternatíva: a modul animációját töröljük, majd tetszőleges helyen az *Animate* eszköztárból újra építjük.

Az automatikusan elhelyezett animációkat a helyükön is hagyhatjuk és egyszerűen a másolatukat helyezzük el különálló animációként. Ebben az esetben követni kell néhány szabályt. Bármely animációs összetevő, amely információt tartalmaz (pl. változók és grafikonok) másolható, megkettőzhető. Azok az animációs összetevők, amelyek egy entitás aktivitását mutatják (pl. sorok és eszközök) nem duplikálhatóak az animációban. Ennek oka egyszerű. Ha van két animált sorunk ugyanazzal a névvel, az **Arena** nem képes eldönteni, melyik sorban jelenítse meg a várakozó entitást. Bár az **Arena** engedélyezi az animált sorok duplikációját, ezek azonban ugyanazokat a várakozási jellemzőket fogják mutatni, amint az utoljára elhelyezett animált sor.

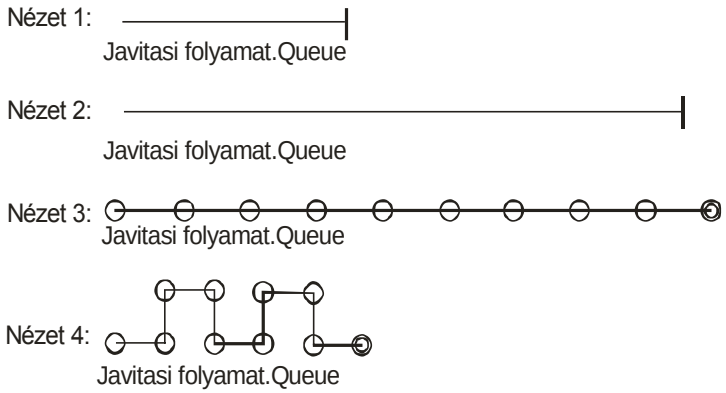
A kiterjesztett animációk létrehozásához a „széthúzás” módszerét fogjuk használni. Kezdjük a kicsinyítés (*Zoom-Out*) használatával. A *View>Zoom Out* (Nézet >Kicsinyítés), a modell méretét csökkenti. Egy sor új területre helyezéséhez, ahol fel akarjuk építeni az animációt, kliekkeljünk a sorra, és használjuk az *Edit>Cut* (Szerkesztés > Kivágás) menü pontot, vagy a *Ctrl+X* gombot a kivágáshoz, majd az *Edit>Paste* (Szerkesztés > Beillesztés) menü pontot, vagy a *Ctrl+V* gombot a beillesztéshez. Ezt az eljárást ismételjük meg a többi sor esetében is, és az elemeket az eredeti modell elrendezésével megegyezően helyezzük el. Ezek után az új területet felnagyítva megkezdhetjük a bővített animáció elkészítését. A sorok megváltoztatásával kezdjük. Új képeket készítünk az entitásokhoz és hozzáadjuk a erőforrás képeket. Végül beillesztjük a változókat és grafikonokat.

4.3.1 Az animációs sorok megváltoztatása

A sor animációkat alaposabban megvizsgálva észrevehetjük, hogy az alapértelmezett sorokban maximálisan 14 entitást láthatunk, többet akkor sem, ha a sorhoz tartozó változó alapján többnek kellene megjelennie. Ennek oka, hogy az **Arena** a sorokban megjelenített animált entitások számát a sorokat ábrázoló vonalak hosszának megfelelő szintre korlátozza, azaz annyi entitás látható, amennyi elfér a rajzolt vonalon. Például egy adott pillanatban, a szimuláció hiába tartalmaz 30 entitást egy sorban, ha az alapértelmezett animációban csak 14 fér el, akkor csak az első 14 jelenik meg. Amint egy entitást távozik a sorból, a következő, addig rejtett, entitás kerül a helyére. Ez azonban nem befolyásolja a számított statisztikák helyességét, de a kezdő felhasználó számára akár megtévesztő is lehet, arra a következtetésre vezethet, hogy bár a rendszer jól működik, de a számított sorok hossza túlságosan nagy. A probléma kiküszöbölésének három egyszerű módja van: figyeljük és jelenítsük meg a sorhoz tartozó animációs változók értékét, a változók értékéhez igazítva növeljük az animációs sor méretét, vagy csökkentsük az entitás kép méretét, annak érdekében, hogy vizuálisan is közelítsük a pontosságot.

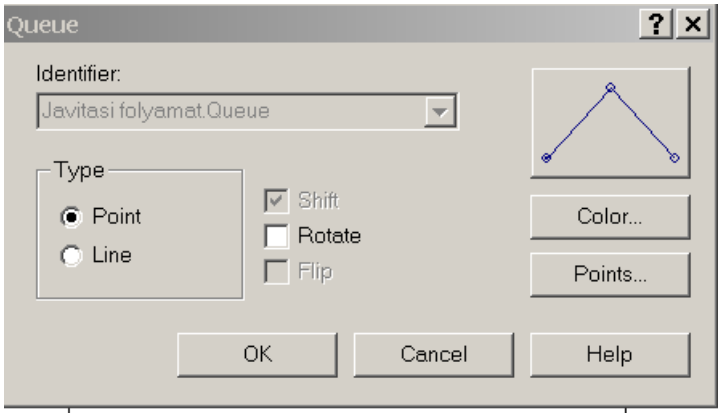
Kezdjünk a sort ábrázoló egyenes hosszának a növelésével. Kövessük a *4.8. ábra* lépéseit. Válasszuk ki a sort (*Nézet 1*) egy kattintással a *„Javitasi folyamat”* során, amelynek a neve: *„Javitasi folyamat.Queue”*. A sor mindkét végén egy-egy kör fog megjelenni. A bal oldali kör fölé helyezve az egérmutatót, a mutató kereszt alakot vesz fel. A kört megfogva, a sor hosszát és irányát változtatni tudjuk (*Nézet 2*). Elindítva a szimulációt látható, hogy időnként

a változtatás előttinél több várakozó munkadarab is láthatóvá válik a *„Javitasi folyamat”* sorában.

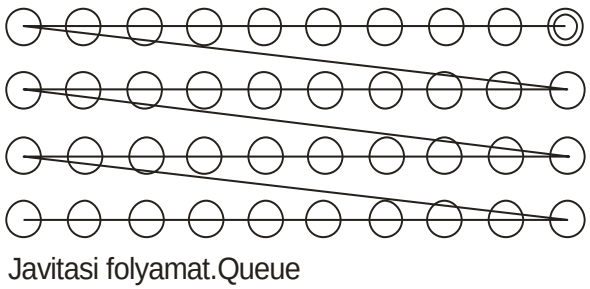


4.8. ábra: Alternatív sor megjelenítési módok

A sorok alakját is megváltoztathatjuk ábrázolva az entitások fizikai elhelyezkedési pontjait is. Duplán kattintva a kiválasztott sorra, megjelenik a **Queue** (sor) párbeszédablak (*4.18 képernyő*).



4.19. képernyő: A **Queue** (sor) párbeszédablak



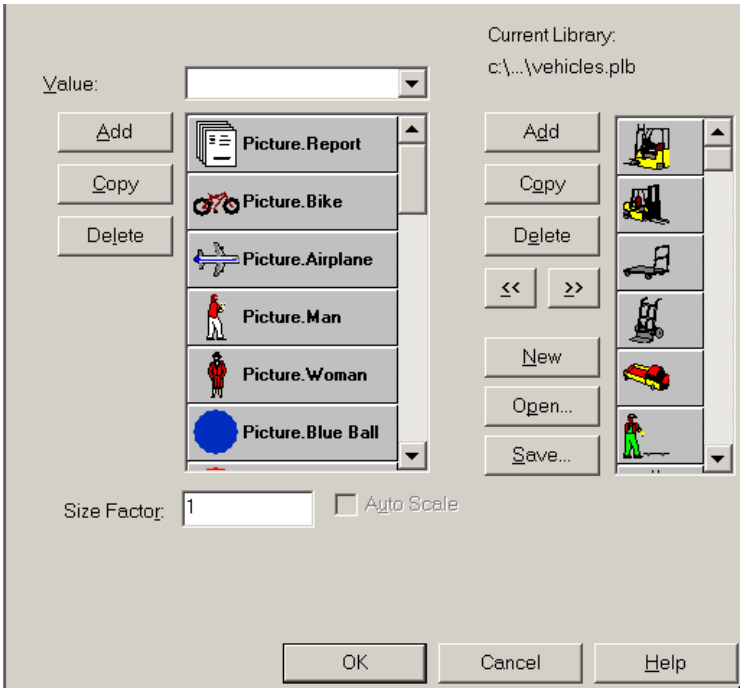
4.9. ábra: A *„Javitasi folyamat.Queue”* 40 ponttal

A sor legyen pont típusú, ezért a Type területen jelöljük meg a *Point* (pont) rádiógombot (*4.19. képernyő*). A lehetőségek: Point (pont) vagy Line (vonal). A *Points* (pontok) gomb megnyit egy új dialógusablakot, ahol egymásután az *Add* (hozzáadás) gombra kattintva további pontokat is beilleszthetünk. Az entitás forgásirányát minden pontban megváltoztathatjuk, de egyelőre fogadjuk el az alapértékeket. Ezeket a beállításokat végrehajtva az eredményként kapott sor a *4.8. ábra* harmadik nézetén látható sorhoz fog hasonlítani. Vegyük észre, hogy a sor elejét két körrel körülvett pont jelöli. A felvett pontok szabadon mozgathatók, a pontot megfogva azt tetszőleges irányban húzhatjuk, és tetszőleges alakzat alakítható ki (*4.8. ábra, Nézet 4*). Ha ezeket a pontokat sorba akarjuk rendezni, a 3. fejezetben tárgyalt

Snap opció segítségével tehetjük ezt. Az **Arena** ezek után az animáció alatt entitásokat helyez el a pontokon és mozgatja azokat előre, éppen úgy, ahogyan az a valóságban, egy sorbanállás esetén történik. Az animációnkban mi egyszerűen csak megnyújtjuk a sorokat (*4.8. ábra, Nézet 2*) az *„A egyseg elokeszitesi folyamat”*, a *„B egyseg elokeszitesi folyamat”* és a *„Szerelesi folyamat”* területeken. A *„Javitasi folyamat”* sorral egy kicsit trükköztünk. A sort pontokká alakítottuk át, és a 38 pontot adtunk hozzá. Ezzel lehetővé vált, hogy a 40 pontot egyenként 10 pontot tartalmazó sorokba rendezzük, amelyek a 4 különálló állványnak felelnek meg (*4.9 ábra*). (A *Snap* opció bekapcsolása a szerkesztési műveletet lényegesen megkönnyíti).

4.3.2 Az entitás képek megváltoztatása

A továbbiakban koncentráljuk a figyelmünket entitások animálására. Az eddigi animációnkban kék és piros labdákat rendeltünk képként az entitásokhoz. Mondjuk, az entitásaink továbbra is legyenek a labdákhoz hasonlónak, de szeretnénk ha a labdák „***A***” illetve „***B***” betű jelölést is kapnának. Az új képeket az **Entity Picture Placement** (entitáskép elhelyező) ablakban készítjük el, amit az *Edit >Entity Pictures* menüponttal nyithatunk meg (*4.10. ábra*). A megnyíló ablak bal oldala a már elérhető entitás képeket tartalmazza a hozzárendelt nevekkel együtt. Az ablak jobb oldalán a képeket tartalmazó könyvtárakat érhetjük el, amelyek fájlokban tárolt képek gyűjteményei. Az **Arena** több kép-könyvtárat tartalmaz, egy kezdeti ikoncsomaggal. Modelljeink animálása előtt érdemes végignézni ezeket az ikonokat (kiterjesztésük: *.plb).



4.10. ábra: Az **Entity Picture Placement** (entitáskép elhelyező) ablak

Az új képek animációba illesztésére többféle módszer is kínálkozik. A baloldali **Add** (hozzáadás) gomb használatával az új képet áthúzhatjuk, illetve beilleszthetjük a meglévők mellé, vagy a *Copy* (másolás) gomb használatával a jobboldali könyvtárban rendelkezésre álló képek közül másolhatunk. Az *Add* (hozzáadás) funkció használatakor az új bejegyzéshez nem tartozik kép és név egészen addig, amíg a képet át nem húzzuk és a Value (érték) mezőben meg nem adjuk a nevét. A *Copy* (másolás) után a másolt kép neve az eredetivel fog megegyezni. A pillanatnyi entitáskép listához új képet adhatunk a jobboldali könyvtár képei közül úgy is, hogy a baloldalon álló képek közül megjelöljük azt, amit helyettesíteni akarunk, a jobboldalon pedig jelöléssel kiválasztjuk az új képet, majd a kettős balra nyíllal (<<) átmásoljuk. Építhetünk és kialakíthatunk saját kép-könyvtárakat is az *New* (új) gomb választásával. Az új saját

tervezésű képeinket elmenthetjük későbbi használatra. A clip art képeit is egyszerűen beszerkeszthetjük a könyvtárunkba a *Copy* és *Paste* parancsokkal.

A példa kedvéért most is nagyon egyszerű képeket fogunk választani, természetesen bonyolultabb, cifrább képekkel is dolgozhatunk. A kék és piros labdák nagyjából megfelelő méretűek. A baloldali ablakban található képek közül válasszuk ki a *Picture.Blue Ball* (kék labda) ikont és kattintsunk a *Copy* (másolás) gombra. Így két egyforma képet kapunk, válasszuk ki az egyiket és a Value (érték) mezőben változtassuk meg a nevét *„A egyseg”*-re. Megjegyezzük, ha begépeljük az új nevet a kiválasztott ikonon név is azonnal megváltozik. A képek szerkesztéséhez, megváltoztatásához elegendő duplán rákattintani a kiválasztott képre, ami megnyitja a **Picture Editor** (képszerkesztő) ablakot. Mielőtt bármit változtatnánk a képen, vegyük észre a kis szürke kört a négyzet közepén. Ez az ún. referencia pont, amely meghatározza az entitás helyzetét a többi animációs objektumhoz viszonyítva. Alapvetően ez a pont fogja követni a kijelölt útvonalat az entitás mozgása közben, illeszkedik a megfogási pontra, amikor az entitás egy erőforrás kontrolja alá kerül, stb.

4.11. ábra: A végleges entitás képek

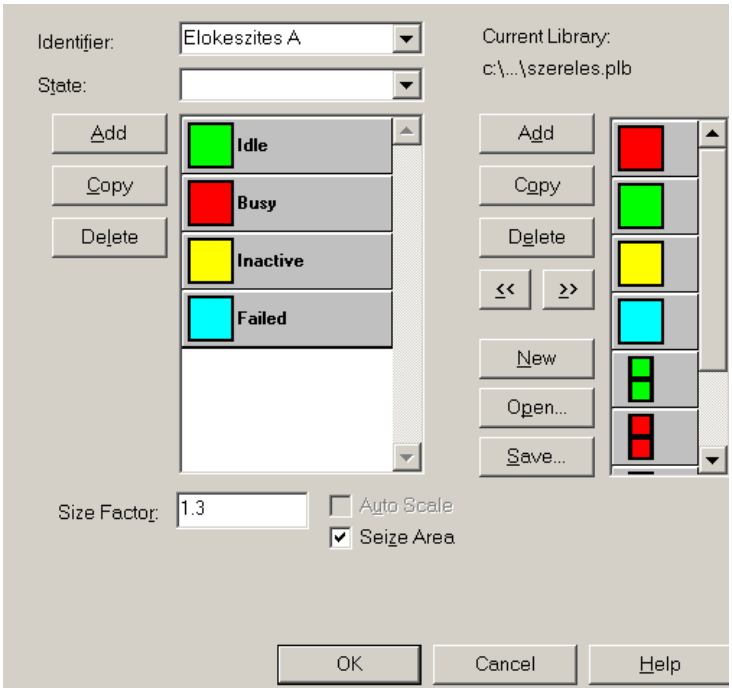
Illesszünk a képbe egy „A” betűt, a labda közepébe. Megváltoztathatjuk a labda kitöltő színét, vagy a betű színét és méretét, hogy a betű látható legyen. Miután az ablakot bezárjuk *„A egyseg kep”* név alatt az új képet találjuk. Ezután ismételjük meg az előző eljárást, és hozzuk létre *„B egyseg kep”* néven képet. Végül a kapott képek a *4.11 ábrához* fognak hasonlítani.

Mielőtt az ablakot bezárnánk, vegyük észre, hogy *„A egyseg kep és B egyseg kep”* nevek nem találhatók meg a képek nevét tartalmazó listában, mert nincs elég hely hozzá. A képre kattintva viszont a teljes név meg fog jelenni az *Value* (érték) mezőben. Jegyezzük meg, hogy van egy *Size Factor* (méretarány mező) az ablak bal alsó sarkában. A méretarány értékének megváltoztatásával növelhető/csökkenthető az entitás kép mérete. Példánkban 1-ről 1,3-re növeltük a méretarányt.

Utolsó lépésként az új képeinket összekapcsoljuk a megfelelő entitásokkal, így az animációnkban is kirajzolódnak. Ennek végrehajtásához kattintsunk az **Entitás** adatmodulra és a két entitáskép nevét írjuk be a *Initial Picture* (kezdő kép) mezőbe. Az új nevek automatikusan nem jelennek meg a gördülő menüben, oda nekünk kell azokat begépelni. A már beírt és mentett adatok megjelennek a listánkban.

4.3.3 A resource (erőforrás) képek hozzáadása

Miután befejeztük az animációs sorok és entitások csinosítását, adjunk új erőforrásképeket az animációhoz. Az **Animate toolbar**-on (animációs eszköztár) a **Resource** (erőforrás) gombra klikkelve megnyílik **Resource Picture Placement** (erőforráskép elhelyező) ablak (*4.12. ábra*), ami nagyon hasonlít az entitáskép elhelyező ablakhoz, és lehetővé teszi, hogy új erőforrásképeket adjunk a modellünkhöz. Az entitás és erőforrás képek, és az azokra való hivatkozás közti különbség nagyon kicsi. Az entitásokhoz úgy rendeljük a képeket, hogy a képek neveire hivatkozunk, amelyeket azonosítóként a képekhez adtunk valahol modellben. A szimuláció futása alatt az erőforrások a képeket állapotfüggően használják. A 4.2.1 bekezdésben részleteztük az erőforrások 4 lehetséges állapotát (*Idle, Busy, Failed és Inactive*). Amikor megnyitjuk a **Resource Picture Placement** ablakot, akkor láthatjuk, hogy mind a 4 alapállapothoz tartozik egy-egy alapértelmezett kép. Az erőforrások különböző állapotait ábrázoló képeket megváltoztathatjuk, ahogyan azt az entitás képekkel is tettük.



4.12. ábra: A erőforrás képek

Az első lépésben meg kell határozni, hogy melyik erőforrásképét akarjuk létrehozni. Az Identifier (azonosító) mezőhöz tartozó legördülő lista erőforrásai közül választanunk egyet *„Elokeszites A”*-t (*4.12. ábra*). Ahogyan az entitás képeknél is tettük, cseréljük ki a képeinket. Duplán kattintsunk az *Idle* képünkre, hogy megnyissuk **Picture Editor** (képszerkesztő) ablakot. Használjuk a háttérszínt a kitöltésre, a vonalvastagságot állítsuk 3 pontnyira (*Draw* eszköztár), és változtassuk meg a vonal színét. A doboznak kijelöltnek kell lennie, hogy ezeket a változtatásokat végrehajthassuk. A kis kör a keresztel az erőforrás referencia pontja, ami meghatározza, hogy a többi objektumok (pl. entitás képek) hogyan helyezkednek el a képhez képest. Az egérrel húzzuk ezt a dobozunk közepébe. A képszerkesztő ablak bezárásával fogadjuk el ezt az ikont és térjünk vissza az **Resource Picture Placement** ablakba. Most készítsük el a saját kép-könyvtárunkat. A *New* gombra kattintva egy új, üres könyvtár file-t nyílik meg. Válasszuk ki újonnan elkészített ikonunkat, kattintsunk **Current Library** terület alatti *Add* (hozzáadás) gombra, majd a jobbra (>>) nyílra. A *Save* (mentés) gombra kattintva mentsük az új képünket (pl. konyv.plb néven).

E kép felhasználásával fogjuk létrehozni a többi erőforrás képünket is. Jelöljük ki a *Busy* képet a baloldalon, és könyvtárunkban található új képet a jobb oldalon, és a bal nyíl (<<) gombbal a *Busy* képet felülírjuk az új *Idle* képünkkel. Amikor az animáció fut az entitás képünk a doboz közepén fog elhelyezkedni, innen tudjuk, hogy foglalt. Ellenőrizzük a **Seize Area** (megfogási terület) kapcsolót az ablak alján, legyen bekapcsolt állapotban. Másoljuk át ugyanezt a könyvtár képet a *Failed* és az *Inactive* állapotokhoz is. Nyissuk meg ezeket a képeket és töltsük ki azokat olyan színnel, amiből kiderül, hogy *Failed* (pl. piros) vagy *Inactive* (pl. szürke) állapotban vannak. Másoljuk vissza ezeket a képeket a könyvtárunkba és mentsük el azokat. A *4.12. ábra* szemlélteti, a művelet után az erőforrás képeink hogyan néznek ki. A méretarányt szintén 1,3 szeresre növeljük, így az entitás képekkel megegyező lesz.

Miután az erőforrás képeket elfogadtuk és visszatértünk a fő ablakba a pointer kereszt alakú lesz. Pozícionáljuk a pointert megközelítőleg arra a helyre, ahová az erőforrást el akarjuk helyezni, és klikkeljünk. Az erőforrást ilyen módon az animációba helyezzük. (A modell elmentéséről persze ne feledkezzünk meg!) Ha az új erőforrás ikonunk nem megfelelő méretű, akkor valamelyik sarkánál megfogva a kívánt mértre kicsinyíthetjük vagy nagyíthatjuk. Az erő-

forrás képek tartalmaznak egy objektumot is, ami egy szaggatott vonallal rajzolt koncentrikus körök formájában láthatóvá válik, ha kétszer klikkelünk a képre, és a megnyiló **Resource Picture Placement** ablakban bekapcsoljuk a *Seize Area* kapcsolót (*4.12. ábra*). Az objektum az erőforráskép alsó részéhez kapcsolódik, ez a **Seize Area** (megfogó terület). Ezen a területen helyezkedik el az entitás, amikor az erőforrás kontrolja alatt van, ha szükséges helyezzük az erőforrás kép közepére. Futtassuk az animációt, hogy lássuk megfelelő méretűek-e az entitás és erőforrás képeink. Ha nem megfelelőek, akkor szakítsuk meg a program futását és finomítsunk a képek helyzetén. A *View>Layers* menüopcióval a **Seize Area** láthatóvá válik. Miután ezt megtettük kikapcsolhatjuk a **Seize Area** réteget és tovább futtathatjuk az animációt.

Ha már megfelelőnek találjuk az *„Elokeszites A”* erőforrás animációját, akkor foglalkozzunk a többi erőforrással, és illesszünk animációt ezekhez is. Alkalmazhatjuk az előzőleg leírt eljárást vagy az *„Elokeszites A”* animációt másoljuk az *„Elokeszites B”* és a *„Szereles”* erőforrásokra. A másolás után, a **Resource Picture Placement** ablak megnyitásához, kattintsunk duplán a beillesztett erőforrásra, majd az Identifier mezőben rendeljük a megfelelő erőforrás nevet az animációhoz (a mezőhöz tartozó legördülő listából választhatunk).

A *„Javitas”* erőforrás képét (a többi erőforrás képéhez képest) meg kell változtatni, mivel a második műszakban a kapacitása 2-re nő. Kezdjünk most is a *Copy-Paste* paranccsal, de miután az **Resource Picture Placement** ablakot megnyitjuk, és a képszerkesztőben szerkesztjük az *Idle* képet, illesszünk még egy négyzetet (*Edit>Duplicate* vagy *Edit>Copy* majd *Edit>Paste* parancsokkal) az első kép mellé vagy alá. A két négyzet a kettő értékű kapacitásnak felel meg, és azt is jelenti, hogy az erőforrás egyidejűleg két entitás feldolgozását (javítását) végezheti a második műszakban. Ezt a képet másoljuk a könyvtárunkba és használjuk fel a hiányzó *„Javitas”* erőforrás képek elkészítéséhez is. Nevezzük át az erőforrást (*„Javitas”*) és zárjuk be az ablakot.

Az eredeti animációban egy **Seize Area** (megfogási terület) volt, kattintsunk kettőt erre, majd a *Points* gombra, és adjunk hozzá egy második **Seize Area**-t. A **Seize Area**-k leginkább pontokból álló sorokhoz hasonlítanak, bármennyi pontból állhatnak, a pontok számát az erőforrás kapacitása határozza meg. A **Seize Area**-k is egy vonalként lehetnek ábrázolva, ahogyan a sorok is, ez ne tévesszen meg minket. Zárjuk be ezt az ablakot és helyezzük el a két **Seize Area** pontot a két dobozban, ezek az erőforrást fogják jelképezni.

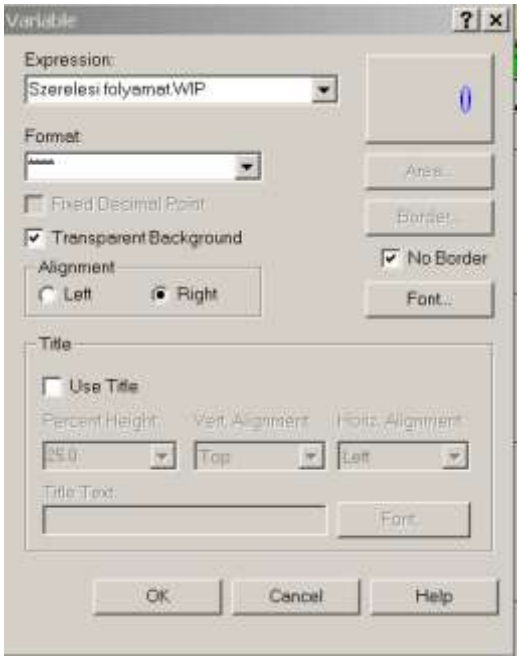
Az erőfeszítéseink eredményeként az animációnk már kezd hasonlítani a vizsgált rendszerhez. Természetesen tovább finomíthatjuk az erőforrásokat, sorokat, állomásokat stb., egészen addig, amíg elégedettek nem leszünk az animáció működésével. Ha a modellt saját magunk készítettük/építettük fel, a 4.2 modellből, akkor ne felejtsük el kikapcsolni a *Run>Run Control > Batch Run* (*No Animation*) opciót, azért, hogy élvezzük az eddigi munkánk gyümölcseit, a művészi munkát, amivel megszerkesszük a 4.3 modellt. Csatolhatunk szövegeket, feliratokat, elhelyezhetünk vonalakat, dobozokat a sorok vagy falak jelzésére, még néhány cserepes virággal is díszíthetjük a modellünket.

4.3.4 Változók és görbék hozzáadása

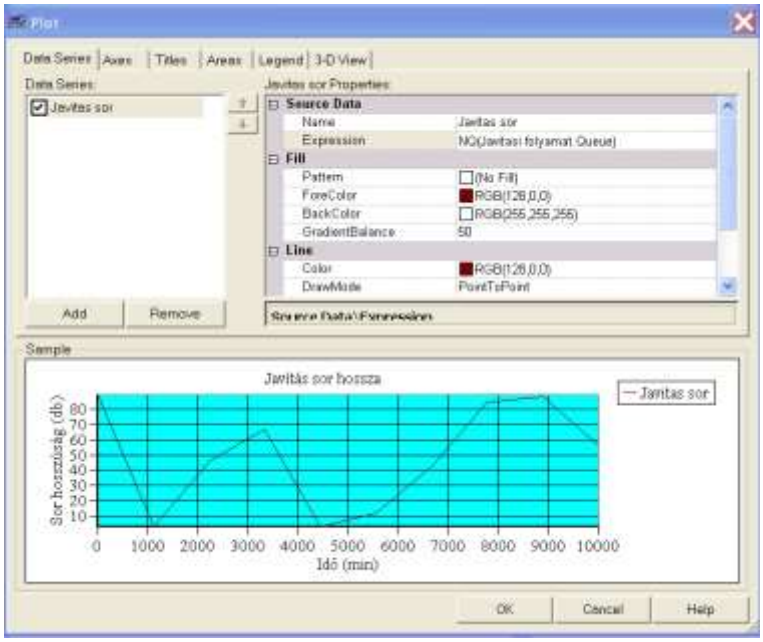
Az utolsó lépésben néhány változót és diagramot adunk az animációhoz. Gazdasági változókat, a folyamatban lévő (kiszolgálás alatt és a sorban álló) és a három kilépési ponton távozó elkészített darabok mennyiségét mutató változókat. A változókat a *flowchart* modulokban érhetjük el, és a *copy-paste* paranccsal másolhatjuk a megfelelő helyre.

Először másoljuk a **Process** modulokból kapott 4 változót, és illesszük be őket közvetlenül az erőforrás képek alá, és kicsinyítsük/nagyítsuk őket a megfelelő méretre. Kettőt kattintva a változóra megjelenik a **Variable** (változó) ablak (*4.13 ábra*). Itt újraformázhatjuk a változót, megváltoztathatjuk a betűtípusát, betűméretét, betűszínét. Ismételjük meg ugyanezt a másik 3,

a *Dispose* modulokhoz tartozó változó esetében is. Végül, használjuk a *Text* (szöveg) szerkesztőt az *Animate* eszköztárról, amellyel felcímkézhetjük a változókat.



4.13. ábra: A változók ablaka



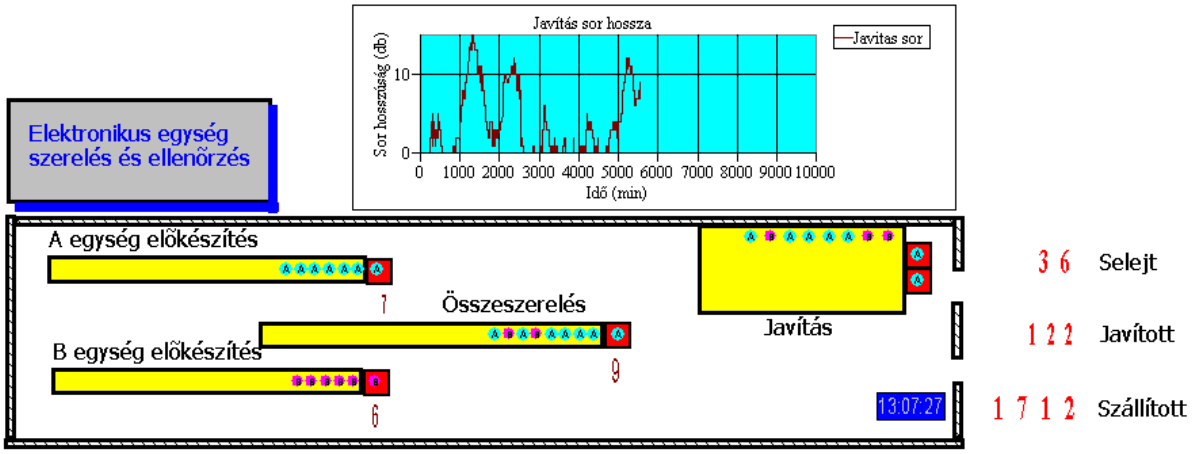
4.20. képernyő: A **Plot** (diagram) ablak

A sor hosszának a kijelzésére társítsunk egy grafikont *„Javitasi folyamat.Queue”* sorhoz. Az **Animate** eszköztáron kattintsunk a *Plot* gombra, ami megnyitja a **Plot** ablakot (*4.20. képernyő*). A **Data Series** fül alatt az *Add* gombbal adjuk az *NQ*(*Javitasi folyamat.Queue*) kifejezést az *Expression* mezőhöz. Ehhez használhatjuk a mező lenyíló listájából elérhető **Expression Builder**-t. Emlékezzünk vissza, ugyanezt a kifejezést használtuk, amikor a gyakorisági adatokat létrehoztuk. Az *Plot* szerkesztő változó paramétereit a *4.16. táblázat* foglalja össze.

4.16. táblázat

A **Plot** ablak változó paraméterei

Data Series		Source Data	Name	Javítás sor
			Expression	NQ(Javitasi folyama.Queue)
Axes	Time(X) Axis	Major	Visible	True
		Scale	Minimum	0
			Maximum	10000
			MajorIncrement	1000
		Title	Text	Idő (min)
			Visible	True
	Left Value (Y) Axis	Major	Visible	True
		Scale	Minimum	0
			Maximum	40
			MajorIncrement	10
		Title	Text	Sor hosszúság (db)
			Visible	True
Titles	Header	Text	Text	Javítás sor hossza



4.14. ábra: Az animáció az 13 óra:07 perc:27 másodperckor

Miután jóváhagytuk az adatokat, a modellablakban a grafikont tetszőlegesen elhelyezhetjük, megnövelhetjük méretét, és a teljes animációt egy másik területére mozgathatjuk.

Az animáció ezzel elkészült, amelynek a *4.14. ábrán* szemléltetett pillanatképhez hasonlóan kell kinéznie, amit a szimuláció 13 óra:07 perc:27. másodpercében rögzítettünk. A végső numerikus eredmények természetesen megegyeznek a 4.2 modell eredményeivel, mivel csak az animációt változtattuk finomítottuk a jelenlegi 4.3 modellben.

4.4: Elektronikus-egység összeszerelő és ellenőrző rendszer munkadarab szállítással (4.4. modell)

E fejezetig azzal a feltevéssel fejlesztettük az elektronikus-egység összeszerelő és ellenőrző rendszer modelleket, hogy a műveletek közötti munkadarab-mozgások időt nem igényelnek, és azonnal megtörténnek. A valós folyamatok pontosabb közelítése érdekében módosítsuk ezt a feltételezésünket, modellezzük úgy a rendszert, hogy valamennyi munkadarab mozgatása 2 perc időtartamú, tekintet nélkül arra, hogy honnan jönnek, és merre tartanak. Ez vonatkozik az érkező munkadarabok előkészítő területre történő mozgatására, az összeszerelő, illetve a javító területekről elszállított selejtes, javított minőségileg megfelelő és a minőségileg megfelelő egységekre. Az új modellünket a *4.3.* modell módosításával alkotjuk meg.

4.4.1 Új Arena fogalmak: Stations és Transfers

Az alkatrészek 2 perces mozgásainak modellezéséhez, meg kell ismerkednünk két új **Arena** fogalommal, ezek a *Stations* (állomások) és a *Transfers* (szállítások). Az **Arena** a fizikai rendszerek modellezését a helyek azonosításával kezdi, amelyeket állomásoknak nevezzük. Az állomásokat úgy kell elképzelni, mint olyan helyeket, ahol valamilyen folyamat megy végbe. A példánkban az állomások olyan helyeket reprezentálnak, ahova részegységek vagy egységek érkeznek, ezek a 4 db gyártócella (**Process** modulok) és a kiadó terület. Minden állomásnak önálló nevet adunk. A modellben az állomások, egyrészt a modell logikai folyamatának belépő pontjaiként szerepelnek, másrészt együtt dolgoznak az *Station Transfers*-nek (állomástranszfernek) nevezett új fogalommal, amit a következőkben részletezünk.

A *Station Transfers* használatával entitásokat tudunk küldeni egyik állomásról a másikra anélkül, hogy azok közvetlen kapcsolatban lennének egymással. A kötött pályás anyagmozgató gépekhez, és az entitás típusától függő rugalmas szállítási pályákhoz tartozó pozitív szállítási idők eléréséhez az **Arena** különböző típusú *Station Transfers*-eket használ. A modellünkben használt *Station Transfers* a *Route* (útvonal), és amely lehetővé teszi az entitások mozgatását az egyik állomásról a másikra. A *Route*-tok feltételezik, hogy az állomások közti mozgás időt vesz igénybe, de nem kezel további késleltetési időket és kötöttségeket, mint például lezárt átjárók, vagy nem elérhető anyagmozgató eszközök miatti várakozások. A *Route* idő megadható, egy konstans számmal, egy eloszláshoz tartozó mintával, illetve bármilyen érvényes kifejezés számított értékével.

Egy rendszerben az állomásokra általában, mint fizikai helyszínekre gondolunk, de ez nem feltétlenül igaz, valójában az állomások hatékonyan használhatóak sok más egyéb modellezési célra. Egy pillanatra lépjünk vissza, és felejtsük el, hogy az állomásokat egy rendszer ábrázolásakor általában mire is szoktuk használni, vizsgáljuk meg mi is történik az **Arena** programban, amikor egy entitást szállítunk *routolással* egy állomásra. Először vegyük szemügyre a modell logikai folyamatát, az entitások mozgását modulról modulra a program futása során. A felszín alatt, ahogy azt korábban megismertük, a szimuláció futásának hajtóerejét az entitások szolgáltatják, létrejönnek, áthaladnak a logikai hálón, amikor késleltetés (várakoztatás) történik, akkor bekerülnek az eseményjegyzékbe, és végül megsemmisülnek. Ebből a szempontból nézve a *Station Transfers* -ek (*route*-ok), nem mások, mint a késleltetés (várakoztatás) eszközei.

Amikor a szállító mechanizmusként működő *Route* által meghatározott módon egy entitás kilép a modulból, akkor az **Arena** az entitást az eseményjegyzékbe helyezi az út időtartama által meghatározott eseményidővel (a **Delay** modulhoz hasonló módon). Később, amikor az entitás az eseményjegyzékből távozik, az **Arena** visszairányítja az entitást a modul logikai folyamatába, megkeresi azt a modult, amelyet a célállomásaként határoztunk meg, ez általában egy **Station** modul. Például feltételezzük, hogy létezik egy **Route** modul, amelynek feladata, hogy az entitást a „*Szerelo”* nevű állomásra küldje. Amikor az entitás kikerül az eseményjegyzékből, az **Arena** megkeresi a „*Szerelo”* állomásként definiált állomást, és az entitást e modulhoz irányítja.

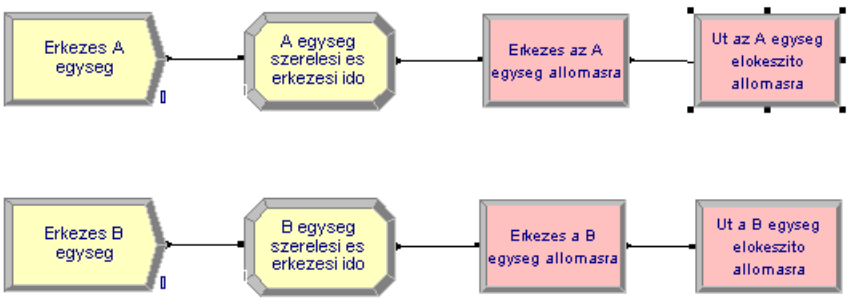
Ez egy kicsit más, mint az eddig megismert közvetlen modulkapcsolatok, amelyekben az entitások mozgása modulról modulra történt, anélkül, hogy az entitás az eseményjegyzékbe került volna, és a kapcsolat grafikusan jelent meg a modellben, mint egy kapcsoló vonal két modul közt. Amíg a közvetlen kapcsolatok folyamatábraszerű kinézetet kölcsönöznek a modellnek, ami egyértelművé teszi, hogyan mozognak az entitások a modulok közt, addig az állomástranszferek, amint később látni fogjuk, inkább jó kezelhetőséget és flexibilitást biztosítanak az entitások irányításában, nyomon követésében.

A modellanimációban jelentős szerepet betöltő állomások és állomástranszferek adják az entitások állomások közti mozgásának hajtóerejét a modell futása alatt, miközben szemléltetik a mozgást. Amikor a modell folyamatábra nézetben jelenik meg, akkor az animációban az állomásokat ún. *station marker* szimbólumok jelölik. Ezek az állomások a modell rajzán ott helyezkednek el, ahol az állomástranszferek kezdődnek vagy végződnek. Az entitások állomások közti mozgásának nyomvonalát *route path objectek* (útvonal objektumok) határozzák meg, amelyek vizuálisan kötik össze az állomásokat egymással, ezzel biztosítják az entitások útvonalának megjelenítését.

Látni fogjuk, hogy az *station marker*-ek (állomás jelzők) vagy a *route path* végpontjaként egy célállomást határoznak meg, vagy a *route path* kezdőpontjaként egy indulóállomást definiálnak, ami egy állomástranszferen keresztül teszi lehetővé egy modulból a kiszállítást. Az animációban az állomásokat szimbolizáló *station marker*-ek az *Animate Transfer* eszköztáron található *Station object* használatával adhatók a modellhez. Két *Station object* között futó útvonalakat ugyancsak az *Animate Transfer* eszköztár elérhető *Route object* segítségével rajzolhatjuk meg. A szimuláció futásakor az entitások ezen a grafikus útvonalon fognak haladni, és a program a haladásukat entitásképek mozgásával szemlélteti. Ez maga után vonja a kérdést: hogyan viszonyul ez, ahhoz a korábban ismertetett logikához, miszerint amikor az entitás az eseményjegyzékben van, akkor az út időtartamának megfelelő ideig ott várakozik. A válasz egyszerű, az **Arena** animációs motorja koordinálja az alaplogikai egységet, esetünkben az eseményjegyzéket. Amikor egy entitás találkozik egy *route*-tal a modellben, akkor az eseményjegyzékbe kerül, az animációban pedig a képe a két állomást összekötő *Route object* vonalán mozog. A két motor összehangoltan dolgozik, így az entitás az animációban épp akkor ér a kijelölt állomásra, amikor az entitás az eseményjegyzékből törlődik. Az animáció ennek köszönhetően minden pillanatban megfelel a modell logikai menetének.

4.4.2 A Route (útvonal) logika hozzáadása

Most már tudjuk, hogy az állomások és transzferek hozzáadása nemcsak a modellre, hanem az animációra is hatással van, fejlesszük tovább az utolsó (4.3 modell) modellünket. Nyissuk meg a modellt, majd mentsük el *Modell4-4.doe* néven (*File>Save As*). Kezdjük az egységek érkezésével. Töröljük a kapcsolatot a két **Assign** modul, valamint az *„A egyseg elokeszitesi folyamat”* és *„B egyseg elokeszitesi folyamat”* nevű modulok közt, majd toljuk el a két **Create/Assign** modul párt balra a képernyőn, hogy legyen helyünk a *route* logikát megvalósító modulok hozzáadásához. A *4.15 ábra* szemlélteti, hogy milyen változtatásokat hajtottunk végre a modellen.

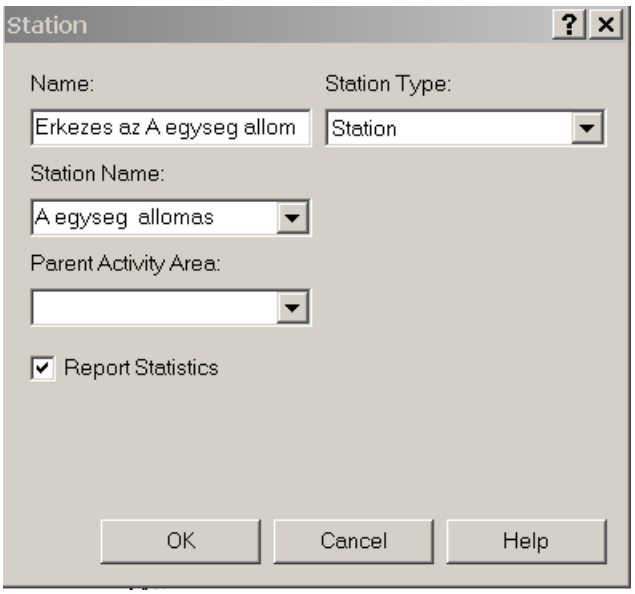


4.15. ábra: Az egységek érkezésének logikai moduljai

A már létező *„A egyseg elokeszitesi folyamat”* és *„B egyseg elokeszitesi folyamat”* nevű **Process** modulok, a **Create** és **Assign** modulok változatlanok maradnak, az eredeti modellel megegyezők. Az állomások és a transzferek hozzáadásához először azokat az állomásokat kell

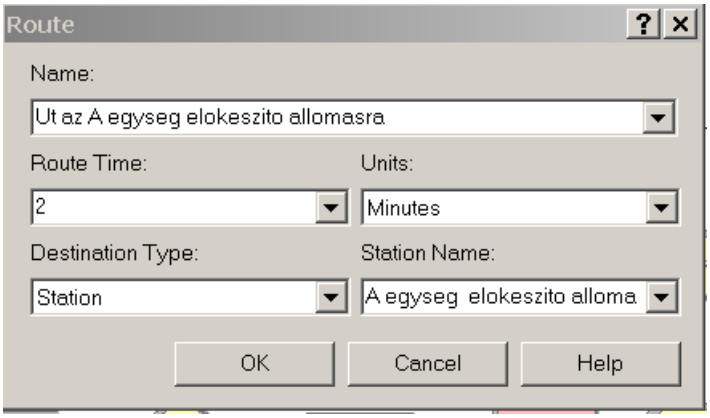
definiálni, ahol az entitás az adott pillanatban tartózkodik, majd a **Route** modulokat, amelyek az entitást a célállomásra irányítják.

Kezdjük az *„A egyseg”*-gel. Csatoljunk egy **Station** modult a modellhez az **Advanced Transfer** panelről, amivel meghatározzuk az *„A egyseg”* érkezési helyét (*4.21. képernyő*). Megadjuk a modul nevét (*„Erkezes az A egyseg allomásra”*), az állomás típusát (*Station*) és az állomás nevét (*„A egyseg allomas”*). Ez a modul határozza meg az új állomást (vagy helyet), és hozzárendeli azt az érkező *„A egyseg”*-ekhez. Jegyezzük meg, hogy **Station** modul ablakban, a *Name* (név) mezőben megjelenített szöveg (*„Erkezes az A egyseg allomásra”*) csak a modul neve, és nem az állomás neve, azaz nem határozza meg magát az állomást. Az állomást a *Station Name* (állomásnév) mező (*„A egyseg allomas”*) definiálja, mint azonosító, amellyel a modell logikai folyamatában azonosítjuk állomást.



4.21. képernyő: A Station (állomás) modul

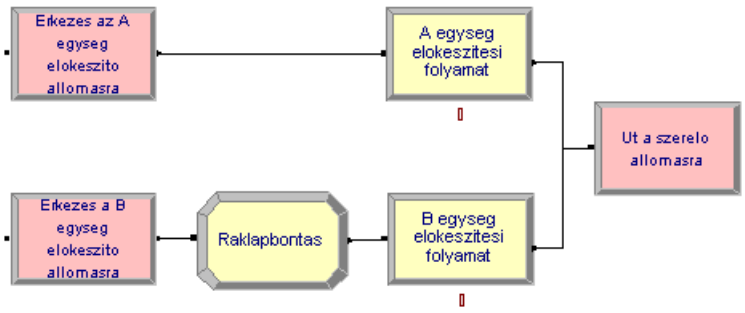
Következő lépésben az **Advanced Transfer** panelről a **Route** modult illesztjük a modellbe, ami az érkező alkatrészt *„A egyseg elokeszitesi folyamat”* területre fogja küldeni 2 perces átviteli idővel. Elnevezzük a modult (*„Ut az A egyseg elokeszito allomasra”*), megadjuk a *Route Time* (utazási idő) időtartamát (2 perc), a *Destination Type*-ot (célállomás típusát) *Station*-nek választjuk, majd a *Station Name* mezőben megadjunk a célállomás nevét (*„A egyseg elokeszito allomas”*) (*4.22. képernyő*). Az átalakítás azt eredményezi, hogy minden, az *„Erkezes az A egyseg erkezesi allomásra”* nevű állomásra érkező *„A egyseg”*-et az *„A egyseg elokeszito allomas”* nevű állomásra küldünk.



4.22. képernyő: A Route (út) modul

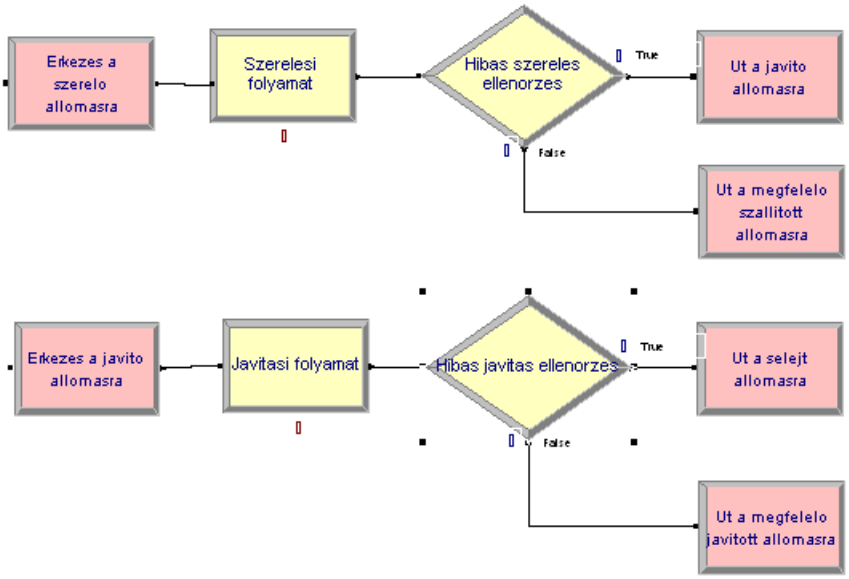
A két új modult hasonlóan illesztjük a ***B*** egység érkezését leíró modulsorba, azzal a különbséggel, hogy a megfelelő mezőkbe ***A*** helyett ***B***–t írunk. Észrevehető, hogy a két **Route** modulból nincsen közvetlen kimenő kapcsolat, mert amint korábban megbeszéltük, az **Arena** gondoskodik arról, hogy az entitások a megfelelő állomásokra kerüljenek.

Haladjunk tovább az ***A*** és ***B*** egységek előkészítési területei felé, és hajtsuk végre a szükséges változtatásokat, aminek eredményét a *4.16. ábra* mutatja. A két **Station** modul, amelyek megelőzik az *„A egyseg elokeszitesi folyamat”* és a *„B egyseg elokeszitesi folyamat”* modulokat, az *„Erkezes az A egyseg elokeszito allomasra”* és az *„Erkezes az B egyseg elokeszito allomasra”* nevű modulok és ezek definiálják az új állomásokat.



4.16. ábra: Az előkészítési terület logikai moduljai

Az állomások adatigénye értelemszerűen megegyezik az előző **Station** modulokéval (*4.21. képernyő*), csak a modulokhoz tartozó mezők tartalma változik. Azok az egységek (alkatrészek), amelyeket a **Route** modulok szállítanak az érkezési területről, ezen állomások egyikére fognak megérkezni. A két **Process** modul változatlan marad. Ezután egy egyedülálló **Route** modult adunk a modellhez az *„A egyseg elokeszitesi folyamat”* és a *„B egyseg elokeszitesi folyamat”* modulokból induló bemenő kapcsolatokkal (*4.16. ábra*). Ilyen módon elérhető, hogy a két különböző helyen található előkészítési területekről, az ***A*** és ***B*** egységeket ugyanarra helyre, a szerelő állomásra továbbítsák.

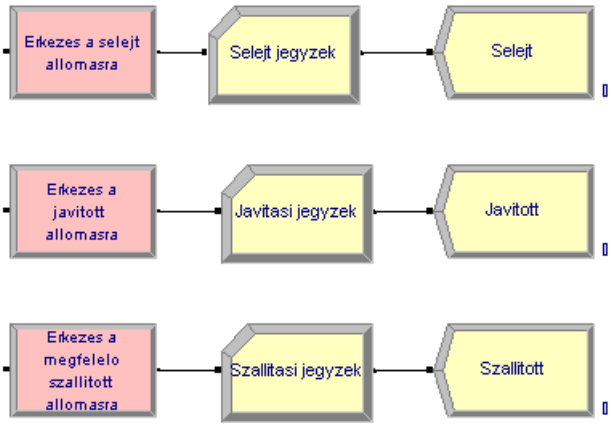


4.17. ábra: A szerelési és a javítási terület logikai moduljai

Az **Arena** nyilvántartja az egységek (alkatrészek) származási helyét, amelyek az *„A egyseg elokeszitesi folyamat”* és a *„B egyseg elokeszitesi folyamat”* nevű területek. A modul ablak-

ban ismét csak a modul nevet (*„Erkezes a szerelo allomasra”*) és az állomás nevet (*„Szerelo allomas”*) kell megadni.

Az elmondottak alapján remélhetőleg világossá vált, mit kell még tenni ahhoz, hogy a modellünk fejlesztésének a végére érjünk. A különböző helyekre állomásokat kell telepítenünk, ahol szükséges **Station** és **Route** modulokat kell beillesztenünk. A *4.17 ábra* mutatja a szerelési (*„Szerelesi folyamat”*) és a javítási (*„Javitasi folyamat”*) területekhez tartozó logikai modell vázat. Hasonló elnevezéseket használjuk, mint eddig: *„Szerelo allomas”*, *„Javito allomas”*, *„Selejt allomas”*, *„Megfelelo javitott allomas”*, *„Megfelelo szallitott allomas”* állomások.



4.18. ábra: A selejt, a javított és a szállított termékek logikai moduljai

A selejtes, a javított, és a minőségileg megfelelő termékek szállítási területeinek logikai vázát a *4.18. ábra* mutatja.

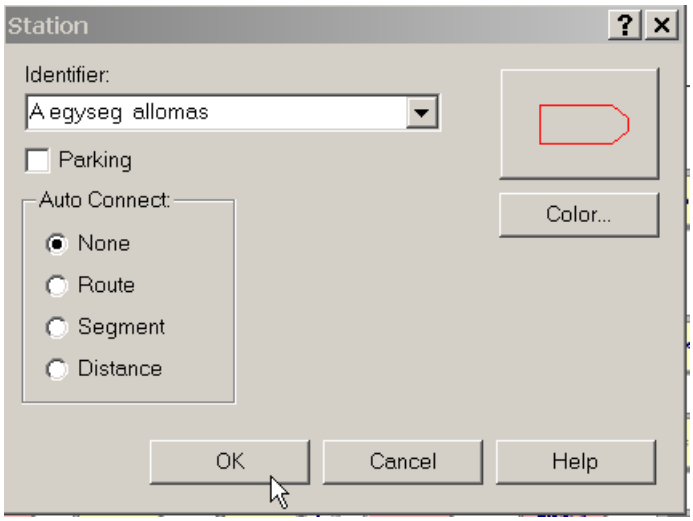
Ebben az állapotban a modell már készen áll a helyes futásra, bár az animációban nem fogunk lényeges eltérést tapasztalni, kivétel talán a *„B egyseg elokeszitesi folyamat”* és a *„Javitasi folyamat”* tartozó sorok sokkal hosszabbak lettek (ezért nagyobb helyet hagytunk ezekben a sor animációkban és átméreteztük az *y* tengelyt a *„Javitasi folyamat.Queue”* sor grafikonján. Ha az eredményeket vizsgáljuk, észrevehetjük, hogy az alkatrészek ciklus idői hosszabbak, mint a 4.2 és 4.3 modellekben, ami a hosszabb szállítási időknek köszönhető. Ha ezeket a modelleket többszörös ismétléssel futtatnánk, azt tapasztalnánk, hogy a különböző számú ismétlésekhez tartozó, a működést jellemző kimeneti értékek igen nagy változatosságot mutatnának, amiből azt a konklúziót tehát lehet levonni, hogy az egyszeres ismétlés ebben a modellben a 4.2 és 4.3 modellhez viszonyítva igen kockázatos dolog.

Talán csodálkozunk azon, hogy a valós idejű (nem 0) szállításokat miért nem egyszerűbb közelítéssel modellezzük. Minden **Station** és **Route** modul pár helyett használhatnánk egy **Process** modult, amely megszakítaná a folyamatot azokon a helyeken, ahol a szállítás előfordul, és amelyben az *Action* mezőt *Delay*-nek (késleltetés), a *Delay Type* mezőt *Constant*-nak és a *Value* mezőt a szállítási időnek megfelelően, 2 percnek választanánk. A modellünk ebben az esetben is működne és valós numerikus kimeneti eredményeket szolgáltatna, de az entitások mozgását nem tudnánk megjeleníteni (pedig ez teszi érdekessé és látványossá a modellt). Ebből a meggondolásból kiindulva, továbbra is útvonal transzferekkel fogunk dolgozni az animációban.

4.4.3 Az animáció megváltoztatása

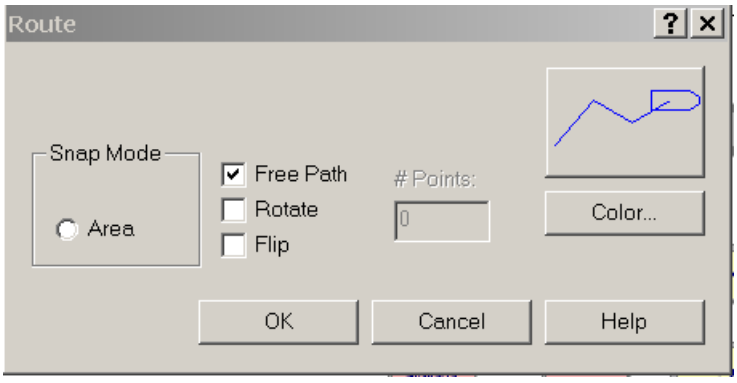
Az animációnk megváltoztatásakor sok, már létező objektumot mozgattunk. Ha szeretnénk elérni, hogy az animáció a könyvben bemutatotthoz hasonlóan nézzen ki, akkor először tanulmányozzuk a *4.19. ábrát*, illetve nyissuk meg a Modell4-4.doe nevű modellt.

Először nézzük, hogyan lehet állomásokat és útvonalakat az animációba illeszteni. Kezdjük az animációs *station markers* (állomásjelzők) hozzáadásával. Az *Animate Transfer* eszköztárban a *Station* gombra kattintva megnyitjuk a *Station* (állomás) párbeszédablakot, amelyet a *4.23. képernyő* szemléltet. Ha nem látható *Animate Transfer* eszköztár, akkor a *View> Toolbars* menüpontokkal megnyithatjuk a *Customize* ablakot, ahol bekapcsolhatjuk az *Animate* opciót, vagy a jobb egérgombbal bármely nyitott eszköztárra kattintva a megnyíló legördülő menüben is megtehetjük ezt. Bontsuk ki az *Identifier* mezőhöz tartozó listát, ahol a már létrehozott állomásainkat fogjuk látni. Ebből válasszuk ki az induló, *„A egyseg allomas”* nevű állomást. Az *OK* gombbal bezárjuk a dialógusablakot, ekkor az egérmutató keresztre fog változni. A keresztet pozícionáljuk az *„A egyseg elokeszitesi folyamat.Queue”* sor bal oldalára, majd kattintással illesszünk be az állomás jelzőt.



4.23. képernyő: Az állomás párbeszédablak

Ezt a lépéssorozatot ismételve helyezzünk el az *„A egyseg elokeszito allomas”* állomásjelzőt az *„A egyseg elokeszitesi folyamat.Queue”* sor alá. Miután az első két állomást elhelyeztük, adjuk meg az állomásokat összekötő útvonalat. Kattintsunk az *Animate Transfer* eszköztár, *Route* gombjára. Ez megnyitja a *Route* dialógusablakot (*4.24. képernyő*). Animációnkhoz egyszerűen fogadjuk el az alapértékeket, de ne felejtsük el, hogy az útvonal tulajdonságait a különböző gombokra kattintva külön-külön megváltoztathatjuk.



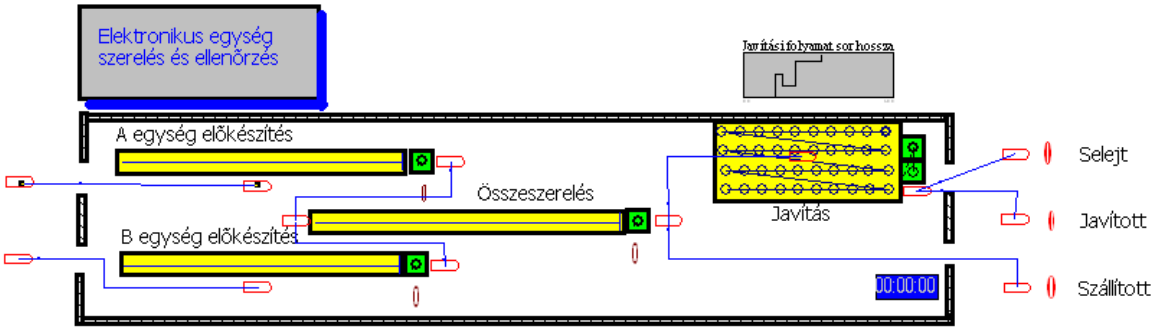
4.24. képernyő: A route (útvonal) párbeszédablak

A *Route* opciók beállításainak megismeréséhez kattintsunk a *Whats This?* (mi ez?) *help* gombra, és válasszuk ki azt a tételt, amelyről közelebbi információt szeretnénk kapni. Miután bezárjuk a dialógusablakot, az egérmutató ismét kereszt alakot vesz fel. A keresztet vigyük az út kezdőpontjára, az *„A egyseg allomas”* állomásjelzőjére, majd kattintsunk az egérrel, ezzel megadjuk az útvonal startpontját. Építsük fel az útvonal pontos nyomvonalát, jelöljük ki a

sarkokat és irányváltásokat, mintha egy összetett vonalat rajzolnánk. Az útvonal végpontjának a kijelöléséhez kattintsunk az *„A egyseg elokeszito allomas”* állomásjelzőjére.

Ha az első, általunk készített animált útvonal működését szeretnénk megtekinteni, akkor futtassuk le a modellt, és látni fogjuk, ahogyan az új alkatrészek megérkeznek az *„A egyseg allomas”*-ra, majd onnan az *„A egyseg elokeszito allomas”*-ra kerülnek.

Ha nem vagyunk elégedettek az útvonalak helyzetével a képernyőn, akkor könnyedén megváltoztathatjuk azt. Klikkeljünk az *„A egyseg elokeszitesi folyamat”* sora feletti állomás jelzőre (ha a *View* > *Data Tips* mező ki van pipálva, akkor elég csak az állomásjelző fölé helyeznünk a pointert), az állomás neve (*„A egyseg elokeszito allomas”*) meg fog jelenni a jelző alatt. Az állomásjelzőt bárhová mozgathatjuk (természetesen a modellablakon belül). Az állomásjelző mozgatásakor a kapcsolódó útvonal vele együtt mozog. Ha kielégítettük a kíváncsiságunkat, akkor helyezzük el ***B*** egységhez tartozó állomásjelzőket is, valamint a *„B egyseg allomas”*-t és a *„B egyseg elokeszito allomas”*-t összekötő útvonalat.



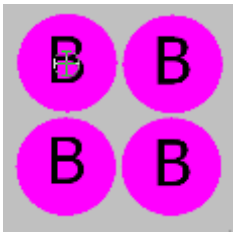
4.19. ábra: A 4.4. modell végső animációja

Miután a hiányzó állomásokat és útvonalakt is beillesztettük, az animációnk teljes lesz. Azonban vizuális szempontból az animációnk nem fogja teljes pontossággal ábrázolni az egységeink (alkatrészeink) áramlását. Emlékezzünk vissza: az *„A egyseg elokeszito allomas”*-t és a *„B egyseg elokeszito allomas”*-t a sorok alá, az *„A egyseg allomas”*-t és a *„B egyseg allomas”*-t a soroktól balra az épületen kívül helyeztük el. Tételezzük fel, hogy a valóságban, az alkatrészek fizikailag az előkészítési terület jobb oldalán lépnek ki. Ha csak a meglévő *„A egyseg elokeszito allomas”* és *„B egyseg elokeszito allomas”* nevű állomásokat használnánk, akkor az alkatrészek az előkészítési területtől balra, ugyanott lépnének ki, ahol beléptek. Az ezt kiküszöbölő megoldás nagyon egyszerű. Illesszünk be újabb állomásjelzőket a két előkészítési terület jobb oldalára, a meglévőkkel megegyező állomás azonosítókkal, és ezeket kössük össze a *„Szerelo allomas”* állomásjelzővel. A további útvonalak építésekor mindig kettő, megegyező nevű állomásjelzőt használnunk az összeszerelés és a javítás területeknél. Az animáció végső képének a *4.19. ábrához* hasonlónak kell lennie.

Végül az animációnak van még egy finomítható pontja, amit eddig nem említettünk. Futtassuk az animációt és figyeljük meg hogyan haladnak a ***B*** egységek entitásai a *„B egyseg allomas”* területről a *„B egyseg elokeszito allomas”* terület felé. Emlékezzünk arra, hogy egyszerre négy entitást hozunk létre a **Create** modulban, de mozogni csak egyet látunk. Azonban, a sorhoz érkezés után már mind a négy megjelenik a képernyőn. Ennek magyarázata, hogy az entitások „egymáson utaznak”, mivel egy időben indulnak és ugyanazzal a sebességgel haladnak. Az ellenőrzéshez nyissuk meg az *„Ut a B egyseg elokeszito allomasra”* **Route** modult és változtassuk meg az utazási idejét konstans 2 percről EXPO(2)-re, ami miatt az entitások utazási ideje egymástól eltérő lesz. Az animáció újrafuttatásakor azt fogjuk észlelni, hogy a 4 entitás ugyan egyszerre indul, de ahogy a sor fellé haladnak elválnak egymástól. A folyamat

jobban látható, ha az út hosszát is megnöveljük. A négy entitás mozgása így látható lesz, de ez nem a valóságot reprezentálja, mivel a modellben feltételeztük, hogy minden entitás konstans, 2 perces utazási idővel halad.

Van egy egyszerű fortély arra, hogy a modell feltételezéseit is megtartsuk (a valóságnak megfelelően) és az animációnk is helyes mozgásokat mutasson. Ha a ***B*** alkatrészek megérkezését egy raklap képpel ábrázoljuk, akkor a kívánt látványt érjük el. Továbbra is 4 entitás mozog a *„B egyseg elokeszito allomas”* felé, de ezek közül csak egy, a felső látszik az animációban. Később, amikor az entitások belépnek a *„B egyseg elokeszitesi folyamat”* sorba képüket egyetlen entitássá konvertáljuk. Ahhoz, hogy bemutassuk ezt a koncepciót, létre kell hoznunk egy raklap képet és 2 helyen meg kell változtatnunk a modellt.



4.20. ábra: A ***B*** raklap entitás képe

Az *Edit* > *Entity Pictures* menüből nyissuk meg az entitáskép elhelyező ablakot, majd készítsünk egy másolatot a ***B*** alkatrész képéről, amit *„B egyseg kep”*-nek nevezzünk el a *Value* (érték) mezőben. Az új kép megjelenik a listában, és dupla kattintás után szerkeszthetővé válik. A kép szerkesztővel copy-paste módszerrel készítsünk négy ***B*** alkatrész kört (*4.20. ábra*), és állítsuk az entitás referencia pontot a négyes csomag középpontjába. Ezután zárjuk be a képszerkesztő ablakot, kattintsunk az OK gombra a modellablakba való visszatéréshez.

Miután elkészítettük a raklap képünket, már csak azt kell meghatároznunk, hogy a programnak pontosan mikor kell az új entitásképet használnia. Nyissuk meg a *„B egyseg szerelesi es erkezesi ido”* nevű **Assign** modulját, és az *Assignments* listához adjunk egy új hozzárendelést (mindegy, hogy a listán a sorban hová kerül, a feladatok nem függnek egymástól). Az hozzárendelés *Entity Picture* típusú legyen, és az *Entity Picture* legyen a *„B raklap kep”* (a kép nevét itt kell először megadnunk). A változtatás hatására az **Arena *B*** alkatrészeknek az érkezési *„B egyseg allomas”* és a *„B egyseg elokeszito allomas”* közötti mozgását a *„B raklap kep”* nevű képpel fogja megjeleníteni. A *„B egyseg elokeszito allomas”* területre való érkezés után a raklap képét, vissza kell konvertálnunk, az egyedi entitás képre.

Ehhez egy új **Assign** modult kell a *„B egyseg elokeszito allomas”* és a *„B egyseg elokeszitesi folyamat”* **Process** modul közé iktatni. Ebben a modulban a hozzárendelés *Entity Picture* típusú, és az *Entity Picture „B egyseg kep”* (a legördülő listából az entitás képek közül választható ki) legyen, azaz a ***B*** alkatrész képét fogjuk használni. Ezzel elérjük, hogy amikor az entitások belépnek a sorba vagy *„B egyseg elokeszitesi folyamat”* **Process** modulba, akkor már egyedi entitásként jelenjenek meg.

Ismét futtassuk az animációt és látni fogjuk, hogy az előkészítő területen négy entitást tartalmazó raklapok mozognak, amelyek *„B egyseg elokeszitesi folyamat”* sorba belépve ismét entitásonként jelennek meg. Ajánlott különböző entitás-képek használata, hogy a rendszerben mozgó alkatrészeket könnyebben meg tudjuk különböztetni. Nézzük meg közelebbről, hogyan is néz ki a *„B egyseg kep”* a rendszeren. Bár továbbra is négy képünk van, egymás hegyén-hátán, az új *„B egyseg kep”* úgy hat, mintha egyetlen, 4 egységből álló raklap mozogna.

Remélhetőleg az útvonal koncepció lényegét sikerült megérteni, később az ismereteink elmélyítése érdekében, a 7. fejezetben komplexebb rendszerek modellezésére is használni fogjuk.

4.5. Input analízis: A modellparaméterek és az eloszlások meghatározása

Ha úgy éreznénk, hogy egy működő szimulációs modell alkotásához minden szükséges ismerettel rendelkezünk, akkor csalódást kell okozni, mivel még nagyon sok részletet kell tisztázni. Az eddigiek alapján valószínűleg mindenki azt gondolja, hogy a modell logikai aspektusai az elsődlegesek, mint például mi az entitás és az erőforrás szerepe, az entitások hogyan lépnek be és hagyják el a modellt, milyen forrásokra van szükségük, az entitások milyen pályákat követnek, és így tovább. A fejlesztési tevékenységünknek ezt a részét **strukturális modellezésnek** is nevezhetjük, hiszen ezek alapozzák és valósítják meg azt az alapvető logikát, amit a modellezéssel el akarunk érni, és amit láttatni szeretnénk.

Amint azt észrevehettük, egy modell specifikálásának vannak olyan elemei is, amelyek inkább numerikus vagy matematikai természetűek (ezért talán inkább evilágiak). A 4.2 modell esetében például az ***A*** egységek érkezési időközeit exponenciális eloszlásúnak tekintettük 5 perces átlagos idővel, a ***B*** egységek előkészítő területén a teljes feldolgozási idő háromszög-eloszlású (3, 5, 10) paraméterekkel. Növeljük az összeszerelés idejét és tekintsük exponenciális (120) eloszlásúnak, és ütemezzük a javítási területen dolgozókat különböző időkkel. Az ilyen változtatások hatásának a vizsgálatát, **kvantitatív modellezésnek** nevezzük, amelynek az eredményei legalább annyira fontosak, mint a strukturális modellezése feltételezései.

Honnan vettük a számokat és az eloszlásokat 4.2. modellhez (csakúgy, mint a könyvben található összes többi modellhez)? Nem érdemes tagadni, elismerjük, hogy némi játszadozás után tulajdonképpen úgy vettük fel a bemeneti adatokat, hogy olyan eredményeket kapjunk, amelyekkel megfelelőképpen illusztrálhatjuk a különböző pontokat. Tettük ezt azért, mert csak egy tankönyvet írunk, és nem valamilyen valóságos munkát végzünk. Azonban ezt a luxust nem mindig engedhetjük meg magunknak. Épp ellenkezőleg, meg kell figyelnie a valóságos rendszert (ha létezik), vagy specifikációkat használni (ha nem létezik), olyan adatokat kell gyűjtenie, amelyek megfelelnek a kvantitatív modellezésnek, és a szimulációban specifikált vagy generált adatokat elemezni kell, annak érdekében, hogy elfogadható modellt és értelmes eredményeket kapjunk. A 4.2. modell esetében az ***A*** egységek érkezési időközeiről, a ***B*** egységek összeszerelési idejéről a javítóállomáson dolgozók állományáról, és más a modellt meghatározó bemeneti mennyiségről kellene adatokat gyűjteni. Ezután a bementi adatokat abból a szempontból kellene vizsgálnia, hogy a pontosság, realitás és érvényesség tekintetében megfelelnek-e modellnek.

Ebben a fejezetben ezt az eljárást ismertetjük, és bemutatjuk, hogyan használható az *Arena Input Analyzer*, ami az **Arena**-val együttműködő önálló alkalmazás. Az *Arena Input Analyzer* segít abban, hogy a változó megfigyelt mennyiségekre valószínűségi eloszlásokat illesszünk.

4.5.1. Determinisztikus vagy véletlenszerű inputok

A kvantitatív modellezés egyik alapvető kérdése, hogy az input mennyiségeket determinisztikus (határozott), vagy valamely valószínűségi eloszlást követő random változóként kívánjuk modellezni. Néha egyértelműen eldönthető, hogy valami determinisztikus-e (például a javítási területen dolgozók száma), még akkor is, ha az értékeket futtatásról futtatásra szeretnénk megváltoztatni azért, hogy lássuk, milyen hatással van ez a teljesítményre.

Azonban nem mindig ilyen egyszerű és könnyű a döntés, és amit egyértelműen tanácsolni lehet, azt tegyük, ami a legreálisabbnak és legmegalapozottabbnak látszik. A 4.5.2 fejezetben beszélünk arról, hogyan is használjuk a modellben az úgynevezett *sensitivity analysis*-t (érzékenységi elemzés), hogy milyen fontos egy adott input az output szempontjából, ami jelezheti mennyire kell figyelni arra, hogy azt determinisztikus vagy random inputként modellezzük.

Csábítónak látszik az a megoldás, hogy kizárjuk az inputok véletlenszerűségének lehetőségét, hiszen így az egész probléma sokkal egyszerűbb, és a modell outputjai sem lesznek véletlen-

szerűek. A modell valósághűsége szempontjából azonban ez egy kicsit kockázatos, hiszen gyakran éppen a véletlenszerűség az, ami elvezet olyan fontos rendszertulajdonságokhoz, amit a szimulációs modellünkből szeretnénk kiolvasni.

Például egy egy-kiszolgálós sor esetében feltételezzük, hogy az érkezési időköz pontosan 1 perc, és a feldolgozási idő pontosan 59 másodperc. Mindkét adat megegyezik a megfigyelt érkezések és a kiszolgálások átlagértékével. Ha a modellt üresen, a kiszolgáló *Idle* állapotban indítjuk, és az első érkezés időpontja 0, akkor sohasem alakul ki sor, hiszen minden egyes ügyfél kiszolgálása befejeződik és távozik, még mielőtt a következő ügyfél megérkezik. Ha azonban a valóságban az érkezési időköz és a kiszolgálási idő olyan exponenciális eloszlású véletlen változó (tehát nem állandó értéke), amelynek átlagértéke 1 perc illetve 59 másodperc, máris különböző hosszúságú sorokat kapunk. Építsünk egy egyszerű **Arena**-modellt erre a példára, és futtassuk hosszú ideig. A hosszú távú futtatásból kiderül a valóság, a sorban álló ügyfelek átlagos száma 58,0167 értékre adódik, ami korántsem egyenlő nullával. A jelenség magyarázata világos, a korrekt véletlenszerű modellben előfordul, hogy néhány kellemetlen ügyfél kiszolgálására több időt kell fordítani, vagy több ügyfél közel azonos időpontban érkezik. Pontosan az ilyen véletlenszerű történések okozzák a sor kialakulását, ami egy inkorrekt determinisztikus modellben sohasem történik meg.

4.5.2. Adatgyűjtés

A szimulációs modell megtervezésének első lépése annak eldöntése, hogy milyen adatok szükségesek a modell működéséhez. Az adatok felkutatása és használatra való előkészítése időigényes, drága, és gyakran fárasztó lehet, továbbá az adatok elérhetősége és minősége befolyásolhatja az általunk alkalmazott megközelítést, és a modellezés részletezésségét, pontosságát.

Sokféle adat gyűjtésére lehet szükségünk. A legtöbb modell sok időlekötésre vonatkozó adatot használ: érkezési időközök, feldolgozási (kiszolgálási) idő, utazási idő, dolgozók munkabeosztása, stb. Sok esetben a valószínűségek (mint például egy művelet eredményessége százalékban, a különböző típusú vásárlók aránya, vagy annak valószínűsége, hogy egy hívó félnek hangkódos telefonja van-e) becslésére is szükség lehet. Ha olyan rendszert modellezünk, ahol az entitások fizikai mozgása az állomások közötti megjelenítésre kerül, az anyagmozgatási rendszer működési paraméterei és azok fizikai megjelenítése egyaránt szükségesek. Sokféle forrásból szerezhetünk adatokat, az elektronikus adatbázisoktól kezdve, a tanulmányozni kívánt területen dolgozó emberek megkérdezéséig. A szimulációs tanulmányhoz történő adatgyűjtés során mind a bőség, mind a hiány előfordulhat, ami mindig egyedi kihívást jelent.

Úgy gondolhatnánk, ha egy létező (vagy más valahol létezőhöz hasonló) rendszert modellezünk, akkor könnyebb dolgunk van, hiszen sok adat áll rendelkezésünkre. Azonban valószínűleg azzal a jelenséggel fogunk találkozni, hogy a rendelkezésre álló adatok nem azok, amire szükségünk lenne. Szokás például gépek műveleti idejének adatait összegyűjteni (a géphez érkezéstől a feldolgozási folyamat befejezéséig eltelt idő), ami első ránézésre alkalmasnak látszik a szimulációs modell műveleti idejének reprezentálására. De ha a gyakorlatban megfigyelt időtartam magában foglalja a sorban állás vagy a gép meghibásodása miatt kieső időt is, akkor ez az idő már nem illeszkedik a modell logikai rendszerébe, ami a sorban állási időt és a gép meghibásodását a működési időtől szigorúan elkülönítve kezeli.

Más oldalról, ha egy teljesen új rendszert, vagy egy már létező rendszer jelentős módosítását szeretnénk modellezni, rögtön a másik szélsőséggel szembesülünk, azaz kevés vagy semmilyen adatunk nem lesz. Ebben az esetben modellünk a tervezők, az értékesítők, stb. durva becsléseire van bízva. Erre vonatkozóan a 4.5.5. fejezetben lesz néhány javaslatunk.

Bárhogy is, ha már egyszer elhatároztuk, milyen és mennyi adatot gyűjtsünk, fontos szem előtt tartani a következőket:

Érzékenységi elemzés: Szimulációs tanulmányok fejlesztésekor gyakran elhanyagolt szempont, hogy mi fontos, mi nem. Az érzékenységi elemzést már a vizsgálat nagyon korai szakaszaiban használhatjuk annak felmérésére, milyen hatással bír egy-egy adatváltozás a modell eredményeire. Ha a rendszerünk valamely részéhez a megfelelő és megbízható adatok nem érhetők el, akkor futtassuk a modellt különböző bemeneti értékekkel, hogy lássuk, mekkora a változtatások hatása a szimuláció végeredményére. Ha a változás nem jelentős, akkor felesleges túl sok energiát beleölni az adatok gyűjtésébe, mert biztosak lehetünk a konklúziók helyességében. Ha azonban a kimenetek nagy eltéréseket mutatnak, akkor meg kell találni az adatgyűjtés helyes módját, ellenkező esetben eredményeik és javaslataink elnagyoltak lesznek.

A modell részletessége és az adatok minősége: A bemenő adatok minőségének az ismerete a fejlesztés korai fázisában azért előnyös, mert segít eldönteni milyen mélységig érdemes a modell logikai felépítését megbontani. Ha a rendszer bizonyos részéihez társítható adatok megbízhatatlanok, akkor semmi értelme sincs annak logikai felépítését részletezni, hacsak nem feltételezzük, hogy egy későbbi időpontban a megbízható adatok elérhetővé vállnak.

Költség: Tekintve, hogy a rendszeradatok összegyűjtése és használatra való előkészítése drága lehet, előfordulhat, hogy úgy döntünk, egyes adatok esetében inkább laza becslésekre hagyatkozunk. Ilyen esetekben az érzékenységi elemzés segítséget nyújthat annak meghatározásában, vajon a becsült adat mekkora szerepet játszik a modell végeredményeit illetően.

„Szemét be, szemét ki”: Tartsuk szem előtt, hogy egy szimulációs tanulmány eredményei és javaslatai csak annyira használhatók, amennyire a modell és az input adatok megbízhatóak. Ha a modell kritikus elemeihez nem találunk pontos adatokat, következtetéseink megbízhatóságába sem vethetünk túl nagy bizalmat. Ez nem azt jelenti, hogy „jó” adat hiányában nem végezhetünk értékelhető kísérletet. Még így is mély betekintést nyerhetünk egy komplex rendszer működésébe, az elemek közti kölcsönhatásokba, és bizonyos szinten előrejelzést adhatunk a működésre vonatkozóan is. Azonban, mindig fordítsunk gondot arra, és artikuláljuk, hogy a becslések megbízhatósága a bevitt adatok minőségén alapul.

Az utolsó észrevétel, amivel szolgálhatunk, hogy az adatgyűjtés (és az adatelemzés) gyakran a szimulációs tanulmányok elkészítésének legnehezebb, legköltségesebb, leginkább időigényes és legunalmasabb része. Ez részben az adatgyűjtés- és elemzés során felbukkanó számos problémának köszönhető, részben pedig annak a tagadhatatlan ténynek, hogy egyszerűen nem olyan izgalmas, mint a logikai rendszer felépítése és működtetése. Próbáljuk elviselni víg kedéllyel ezt a megpróbáltatást, és emlékeztessük magunkat folyamatosan arra, milyen fontos ez a tevékenység (még ha az nem is izgalmas vagy kellemes) a szimuláció szempontjából.

4.5.3 Az adatok használata

Ha ismert időfüggő adatokat használunk (pl. egy gép meghibásodási gyakoriságára és javításának az időtartamára vonatkozó rekordokat), vagy ismerjük, miként működik egy rendszer adott része (pl. a dolgozók ütemezett időbeosztása), még mindig ott a probléma, hogyan is építsük be az adatokat a modellünkbe. Az alapvető kérdés, közvetlenül listából használjuk az adatokat, vagy illesszünk valószínűségi eloszlásfüggvényt az adatokra. A kérdés eldöntése elméleti és gyakorlati megfontolásokon alapul.

Elméleti szempontból a gyűjtött adatok a múltbeli történésekre vonatkoznak, torzított vagy torzítatlan becslései a jövőbeni történéseknek. Amennyiben az időbeli adatok generálásának körülményei megszűnnek (vagy az idő folyamán megváltoztak), akkor az időbeli adat lehet torzított vagy a folyamat néhány fontos részletét illetően lehet egyszerűen hiányos. Ha például az időbeli adatsor az elmúlt 12 hónap megrendelői adatbázisából származik, és négy hónappal ezelőtt új terméket vezettek be, akkor az azt megelőző nyolc hónap adatai használhatatlanok, tekintve, hogy nem tartalmazzák az új körülményt. Ha az időbeli adatokat közvetlenül alkalmazzuk, akkor csak a feljegyzett értékeket használhatjuk. Ha viszont illesztett valószínűségi eloszlásból vesszük a mintát, akkor elképzelhető, hogy lehetetlen értékeket (pl. végtelenhez tartó eloszlás függvény esetén) kapunk, vagy elveszítünk fontos tulajdonságokat (pl. bimodális adatok, egymást követő minták).

Gyakorlati megfontolások is számításba jöhetnek. Elképzelhető, hogy nem rendelkezünk elég időbeli adattal ahhoz, hogy az elemzésünk alátámasztásához elég hosszú szimulációt futtathassunk le. Figyelembe kell venni az adatok elérésének a szimuláció sebességére gyakorolt hatását is. Az adatok egy fájlból történő beolvasása általában lassabb, mint a mintavétel a valószínűségi eloszlásból, ezért az idősorok beolvasása lassítja a szimulációs modell futását.

Választásunkra való tekintet nélkül az **Arena**, olyan beépített eszközökkel rendelkezik, amelyek az időbeli adatok használatának mechanizmusát felügyelik a modellünkben. Ha úgy döntünk, hogy az adatainkra valószínűségi eloszlást illesztünk, az *Input Analyzer* (inputelemző) eszközzel, ez az eljárás ad egy kifejezést, amit a modellben közvetlenül használhatunk. Erről bővebben a 4.5.4 fejezetben lesz szó. Ha a modellt közvetlenül az időbeli adatokra akarjuk építeni, az értékeket azonnal a modell adatstruktúrájának részéve tehetjük, vagy a szimuláció futtatás alatt dinamikusan olvashatjuk be.

4.5.4 Input eloszlás illesztése az Input Analyzer -rel

Ha úgy döntünk, hogy a megfigyelt adatainkra illesztett valószínűségi eloszlásfüggvényt használjuk, akkor vagy magunk választhatjuk ki az eloszlást, vagy többféle eloszlást rendelünk az adatokhoz és az Input Analyzer ezek közül választja ki a legmegfelelőbbet. Bárhogy tegyünk is, az *Input Analyzer* gondoskodik az eloszlás paramétereinek az értékeiről (az általunk biztosított adatok alapján), továbbá kész kifejezéseket (formulákat) biztosít, amelyeket csak át kell másolnunk a modellünkbe.

Amikor az *Input Analyzer* eloszlást illeszt az adatainkra, megbecsli az eloszlás paramétereit (beleértve minden olyan eltolást vagy csonkítást, amely ahhoz szükséges, hogy érvényes kifejezést alkothasson), és számításokat végez, annak megállapítására, hogy az eloszlás mennyire jól illeszkedik az adatainkra. Ez az információ alkalmas lehet a legjobban illeszkedő eloszlás kiválasztására, erről a későbbiekben szót ejtünk.

A valószínűségi eloszlásokat két csoportra bonthatjuk: elméleti és empirikus (tapasztalati) eloszlásokra. Az elméleti eloszlásokból, mint pl. az exponenciális- és a gamma-eloszlás, matematikai képletek alapján generálhatunk mintákat. A tapasztalati eloszlás esetében az aktuális adatokat csoportokba soroljuk, majd meghatározzuk a csoportokban található adatértékek gyakoriságát (súlyát) az összes értékhez viszonyítva, esetleg a további pontosítás kedvéért az egyes pontok közötti interpolálhatunk.

Mindkét csoportot tovább bonthatjuk folytonos és diszkrét típusokra. Az **Arena** által támogatott folytonos elméleti eloszlások a korábban már említett exponenciális, trianguláris, Weibull-eloszlás, béta-, Erlang-, gamma-, lognormális-, uniform-, és normális-eloszlás. Ezeket az eloszlásokat folytonos eloszlásoknak nevezzük, mert az értelmezési tartományukon belül bármilyen valós értéket felvehetnek. A szimulációs modellezésben rendszerint az időtartam megjelenítésére használatosak. A Poisson-eloszlás diszkrét eloszlás, csak egész értékekre ér-

telmezett. Általában egy időintervallumban megtörtént események vagy véletlenszerűen változó halmazok eloszlásának a leírására használjuk.

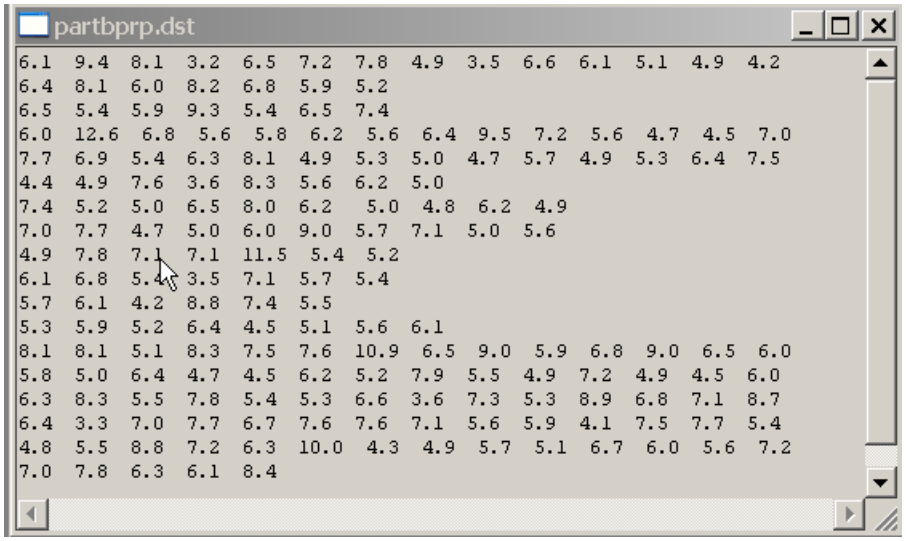
A diszkrét empirikus és a folytonos empirikus valószínűségi eloszlások egyaránt használhatók. Mindkettő úgy definiálható, mint a valószínűség/adatérték párok sorozatai, amelyek hisztogramot alkotnak. A diszkrét empirikus eloszlás csak magát az adatértéket képes visszaadni, a valószínűség alapján választ az individuális adatértékek közöl. Gyakran használatosak a különböző entitás-típusok véletlenszerű hozzárendelésére. A folytonos empirikus eloszlások egyaránt használják a valószínűséget és az adatértéket, hogy visszatérítsenek egy valósértékű mennyiséget. Ezeket az elméleti eloszlások helyett lehet használni azokban az esetekben, amikor az adat szokatlan tulajdonságokkal bír, vagy amikor nincs jól illeszthető elméleti eloszlás.

Az *Input Analyzer* bármely említett eloszlást képes ráilleszteni az adatainkra, azonban saját magunknak kell eldönteni, elméleti vagy empirikus eloszlást akarunk-e választani, és sajnos ehhez a választáshoz nem szolgálhatunk semmilyen segítséggel. Általában, ha az adataink hisztogramja, amit az *Input Analyzer* automatikusan kirajzol, teljesen uniformnak tűnik, vagy csak egy „pupja" van, és nem tartalmaz üres adatértékek nélküli részeket, akkor nagy valószínűséggel valamely elméleti módszer vezethet kellő eredményre. Ha azonban több olyan adatcsoport van, amelyek többféle megfigyelésből származnak (multimodálisak), vagy több olyan adatpontja is van, amely jelentősen különbözik az átlagos megfigyelésektől, akkor a tapasztalati módszer megbízhatóbb képet adhat az adott adatelemről. A tapasztalati eloszlás egyik alternatívája lehet, ha az adatot valamilyen módon több részre osztjuk, erre a fejezet végén térünk ki.

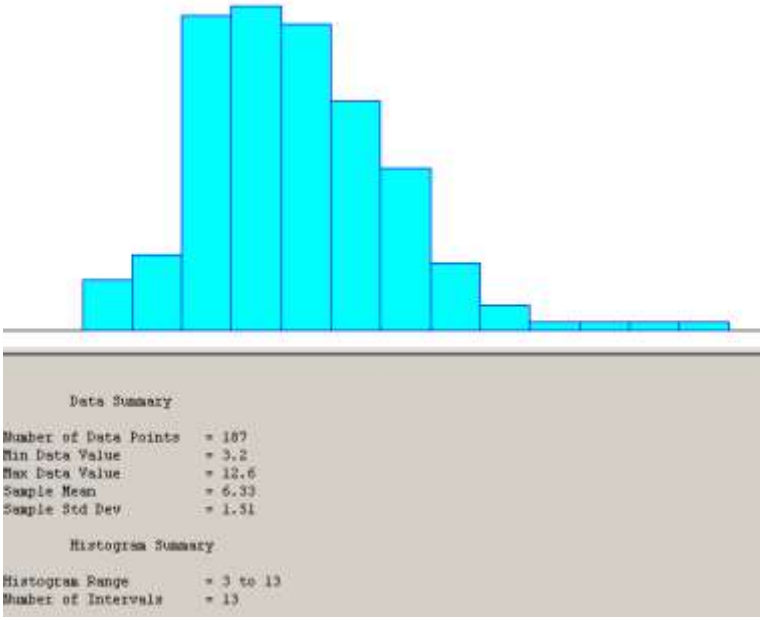
Az *Input Analyzer* olyan standard eszköz, ami az **Arena** program része, és arra tervezték, hogy eloszlásokat illesszen a megfigyelt adatokra, becslésekkel szolgáltassa az eloszlás paramétereit, valamint meghatározza az illeszkedés szorosságát. Négy lépés szükséges ahhoz, hogy az *Input Analyzer* -t használva valószínűségi eloszlásfüggvényt illesszünk az adatainkra, amelyet a továbbiakban a modellünkben input adatként használhatunk:

hozzuk létre az adatértékeinket tartalmazó szövegfájlt,
illesszünk egy vagy több eloszlást az adatokra,
kiválasztjuk melyik eloszlással szeretnénk dolgozni,
másoljuk az *Input Analyzer*-rel generált kifejezést az **Arena** modellben a megfelelő mezőbe.

Az adatfájl előkészítéséhez, egyszerűen hozzunk létre egy egy ASCII szövegfájlt, amely az adatokat szabad formátumban tartalmazza. Ehhez bármilyen szövegszerkesztőt, word processzort, vagy táblázatkezelő programot használhatunk. Az egyes adatelemeket egymástól egy vagy több *white space* (üres hely) karakterrel kell elválasztani (space, tab, soremelés). Más formai követelmény nincs. Egy sorba annyi adatértékünk írhatunk, amennyit csak akarunk, továbbá az egyes sorokban lévő adatok mennyisége soronként változhat. Ha word processzort vagy táblázatkezelőt használunk, figyeljünk arra, hogy a fájlt „text" formátumban mentsük el. Ez eltünteti azokat a karakter- vagy bekezdés formázási jeleket, amelyek máskülönben megmaradnának. A *4.21. ábra* például a partbprp.dst nevű ASCII fájl tartalmát mutatja (az *Input Analyzer* -ben az adatfájlok alapértelmezett kiterjesztés .dst), amely a 187 db ***B*** egység előkészítési idejéről tartalmaz megfigyeléseket. Vegyük észre, hogy az egyes értékeket üres helyek, vagy soremelés választja el egymástól, és soronként különböző számú megfigyelés található a fájlban, az adatstruktúrában nincs különösebb rendszeresség vagy megjelenítési forma.



4.21. ábra: Az ASCII fájl listája (partbprp.dst)



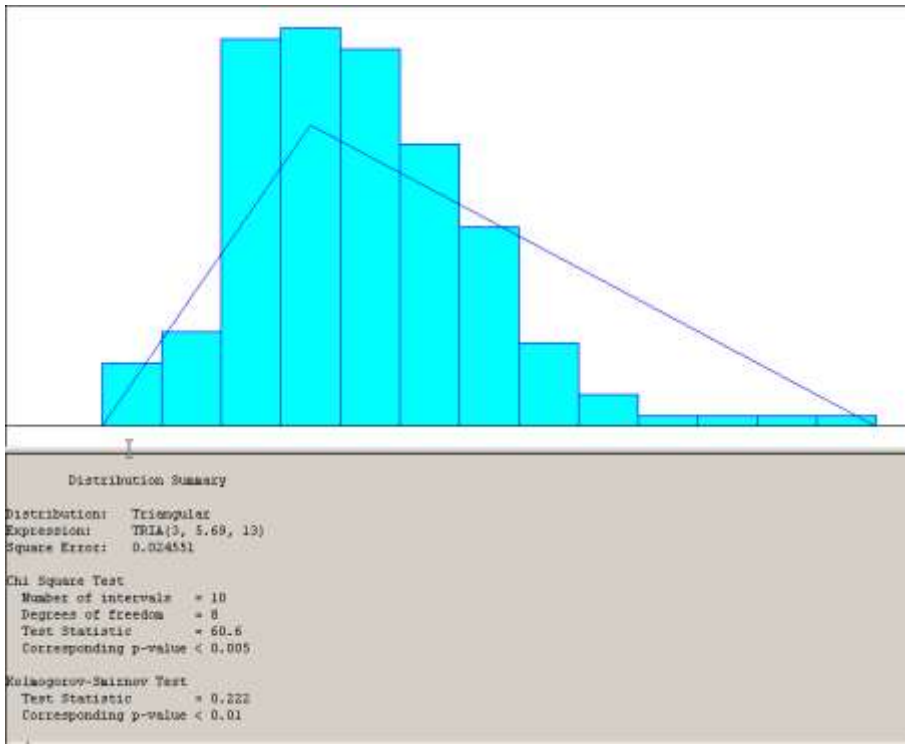
4.22. ábra: A partbprp.dst adatfájl hisztogramja és összesítése

Ahhoz, hogy eloszlást illeszthessünk az adatainkra, futtassuk az *Input Analyzer-t* (válasszuk az *Tools >Input Analyzer* menüpontot az **Arena**-ból). Nyissunk egy új ablakot a *File > New* menüponttal vagy a megfelelő ikonnal, majd a *File > Data File > Use Existing* menüpontok használatával vagy a megfelelő ikon segítségével csatoljuk az adatfájlt (partbprp.dst) az input ablakhoz. Az *Input Analyzer* ablakában azonnal megjelenik a *4.22. ábrán* látható hisztogram, és a hisztogramot, illetve az adatokat jellemző összegzés. Az ablak osztott, a felső részében jelenik meg az adatokból szerkesztett hisztogramot, az alsó részében pedig összegzés. Az adatok esetleges megtekintéséhez használjuk a *Window > Input Data* menüpontot.

Az ablakok relatív méretét az ablak közepén található elosztóval igazíthatjuk. Az adatösszegző ablakban az adatok görgetésére rendelkezésünkre áll a *scroll bar*, ezzel végig pásztázhatjuk a szöveget. Az egyéb opciók, pl. a hisztogram jellemzőinek megváltoztatása, az online Help leírásában található.

Az *Input Analyzer Fit* menüpontja lehetővé teszi különböző valószínűségi eloszlások választását és illesztését az adatokra, és gondoskodik a kiválasztott eloszlás paramétereinek a ki-

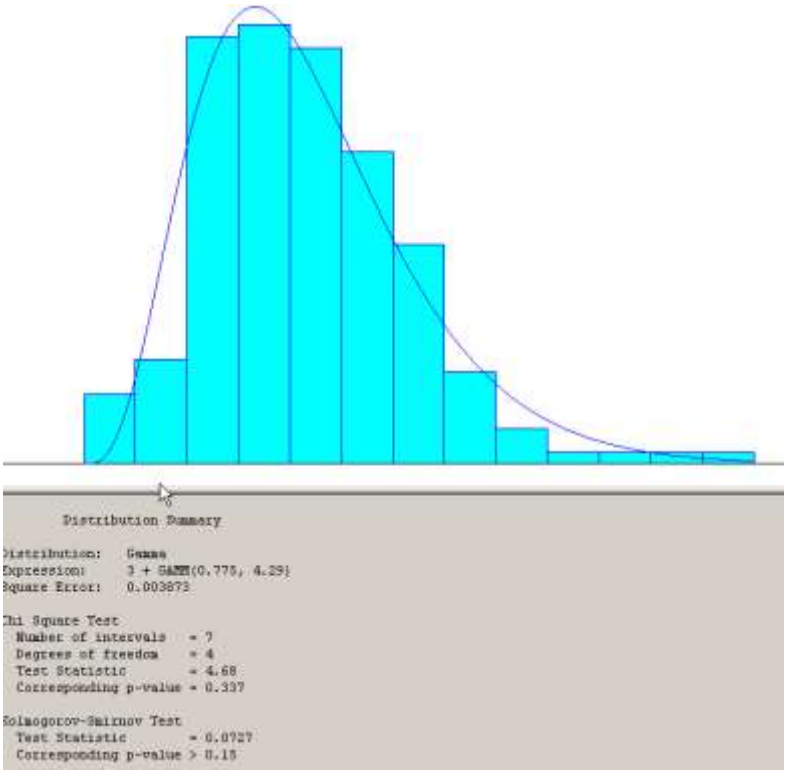
számításáról, becsléséről. Miután kiválasztottunk egy eloszlást, annak sűrűségfüggvénye a hisztogram ablakban a hisztogramra rajzolódik, az illesztés jellemzőinek összefoglalója pedig az alsó szövegablakban jelenik meg (*4.23. ábra*). Az információ a DISTRIBUTION.OUT nevű egyszerű ASCII fájlba is beíródik, ahol a DISTRIBUTION utal a kiválasztott eloszlásra, a trianguláris esetben pl. a fájl neve TRIANGLE.OUT lesz. Az **Arena**-ban az adatok megjelenítéséhez szükséges pontos kifejezés szövegablakban jelenik meg: TRIA(3, 5.69, 13). Ezt a *Edit > Copy Expression* menüponttal átvihetjük az **Arena**-ba. Az **Arena**-ban nyissuk meg a megfelelő párbeszédablakot, és illesztjük be a kifejezést (*Ctrl+V*) a megfelelő mezőbe. A *4.23. ábra* mutatja a trianguláris eloszlást illesztését a partbprp.dst fájlban található adatokra. Bár a különböző illesztések szorosságát később tárgyaljuk, *a 4.23. ábrán* látható, hogy a trianguláris eloszlás nem mindenütt illeszkedik a legjobban az adatokra.



4.23. ábra: Trianguláris eloszlás illesztése a partbprp.dst-ra

Ha elméleti eloszlást akarunk használni a modellünkben, kezdjük a *Fit > Fit All* menüponttal. Ez automatikusan az adatokra illeszti az összes alkalmazható eloszlást, és mindegyikhez kiszámítja a statisztikai értékeket (lásd később), valamint megjeleníti azt az eloszlást, amelynek a hibanégyzet értéke (ez méri az eloszlás illesztésének a szorosságát) a legkisebb. A a *Fit All* parancs eredményét, a ***B*** egység előkészítési idejére a *4.24. ábra* mutatja. A „legjobb” illesztést (a legkisebb hibanégyzet alapján) a β=0,775 és α=4,29 paraméterű gamma-eloszlás (3 értékkel jobbra igazítva) adja. A *4.23. ábra* képével összehasonlítva a jelen ábrát, láthatjuk, hogy a gamma-eloszlás határozottan jobb illeszkedést mutat, mint a trianguláris eloszlás illesztése. Az elméleti eloszlások kiválasztásakor megfontolandó tanácsok az alábbiakban következnek.

Ha diszkrét vagy folytonos empirikus eloszlást akarunk használni, akkor válasszuk az *Empirical* opciót a *Fit* menüből. Először állítsuk be a hisztogram cellák (oszlopok) számát, amely meghatározza, az empirikus eloszlásban mennyi valószínűség-adatérték párral fogunk számolni. Ehhez válasszuk az *Options > Parameters > Histogram* menüpontot és a megjelenő *Histogram Parameters* ablakban változtassuk meg a *Number of Intervals* mezőben az intervallumok számát.



4.24. ábra: *Fit All* illesztés a partbprp.dst-ra

Az *Input Analyzer* a döntés meghozatalát segítve az illesztés minőségének jelzésére három numerikus mérőszámot is biztosít. Az első (és ezt a legkönnyebb megérteni is) a hibanégyzet átlag. A hibanégyzet átlagértéke, a hisztogram cellánként meghatározott, az adott cellában található megfigyelések relatív gyakorisága, és a cella adattartományhoz illesztett valószínűségi eloszlásfüggvény relatív gyakorisága közötti különbségek négyzetének az átlagértéke. Minél nagyobb ez az érték, annál messzebb van az illesztett eloszlás a valós megfigyelt adatoktól, vagyis annál pontatlanabbul illeszkedik. Ha minden lehetséges eloszlást hozzárendelünk az adatokhoz, a *Window >Fit All Summary* menüpontot választva, akkor az eloszlások a *Fit All Summary* táblázatban a hibanégyzet szerint csökkenő sorrendben jelennek (*4.25. ábra*)

Function	Sq Error
Gamma	0.00387
Weibull	0.00443
Beta	0.00444
Erlang	0.00487
Normal	0.00633
Lognormal	0.00871
Triangular	0.0246
Uniform	0.0773
Exponential	0.0806

4.25. ábra: *Fit All Summary* ablak listája

A másik két mutató az illesztés szorosságának az elemzésére a χ^2 (khí-négyzet) próba és a Kolmogorov-Szmirnov (K-S) hipotézis vizsgálat. Ezek olyan szabványos statisztikai hipotézis tesztek, amelyek segítségével megállapító, az elméleti eloszlás megfelelően illeszkedik-e a megfigyelt adatokra. Az *Input Analyzer* a vizsgálat eredményéről a szövegablakban (lásd a *4.23.* és a *4.24. ábrák* alját) tájékoztat. Mindkét vizsgálatnál megkülönböztetett figyelmet kell szentelni a *Corresponding p-value* (megfelelő *p*-érték)-nek, amely mindig 0 és 1 közé esik. A nagyobb *p*- értékek jelentik a jobb illeszkedést. A 0,05-nél kisebb *p*-értékek azt jelzik, hogy az eloszlás nem illeszkedik túl jól. Természetesen, úgymint bármely statisztikai hipotézis

vizsgálatnál, a nagy *p*-érték itt sem garancia a helyességre, csupán azt jelzi, hogy az eredmény semmiképpen sem tekinthető rossznak.

Amikor döntési helyzetbe kerülünk, és eloszlást kell választanunk, egyetemes, örökérvényű segítségre, szabályra (a legjobb eloszlás kiválasztásához) sajnos nem támaszkodhatunk. A különböző statisztikai vizsgálatok (a K-S és a *khi-teszt*) az eloszlásokat különbözőképpen rangsorolják, vagy az eloszlás jobb illeszthetősége érdekében nem azonos módon készítik elő az adatokat (pl. a hisztogram cellák száma tekintetében).

Első kritikus döntésünk az lesz, hogy elméleti, vagy empirikus eloszlást használjunk. A K-S vagy a *khi-teszt* eredményeinek vizsgálata ehhez támpontot nyújthat. Ha egy vagy több eloszlás esetében a *p*-értékek elég magasak (pl. 0,10 vagy magasabbak), akkor nyugodt szívvel használhatunk elméleti eloszlást, valószínűleg helyénvaló eredményt kapunk az adatokkal kapcsolatban (hacsak nem túl kicsi mintavételi egységgel dolgozunk, mert akkor a vizsgálat nem tekinthető megalapozottnak). Ha a *p*-értékek alacsonyak, akkor jobban tesszük, ha empirikus eloszlást választunk a vizsgálathoz.

Ha az elméleti eloszlás mellett döntöttünk, akkor gyakran több olyan eloszlás is lesz közöttük, amely meglehetősen jó illeszkedési statisztikát mutat. Ebben az esetben további döntési kritériumokat is be kell vezetnünk a vizsgálat folytatásához:

Először is el kell döntenünk, korlátos vagy korlátlan eloszlással karunk-e dolgozni, ez attól függ, hogyan fogjuk az adatokat a modellünkben felhasználni. A feldolgozási (kiszolgálási) időhöz például legtöbbször a trianguláris (korlátos) vagy a normális (korlátlan) eloszlás használható legjobban. Hosszú szimulációs folyamatok során hasonló eredményeket produkálhatnak, de a futtatás alatt a normális eloszlás időnként meglehetősen nagy időértékeket generál, olyan értékeket, amelyek a valós rendszer gyakorlatilag nem fordulnának elő. Egy normális eloszlás nullához közeli átlagértékkel nagyszámú nullaértékű mintaelemet eredményezhet, ha az eloszlást olyan mennyiségre használjuk, amely nem vehet fel negatív értékeket, például az idő (Az **Arena** minden negatív inputot nullára konvertál, ha olyan kontextusban fordul elő, amelyben a negatív értékek nincsenek értelmezve, például az érkezési idő). Ez alapos és kényszerítő ok arra, hogy olyan esetekben, amikor a mennyiségek, mint a feldolgozási idő nem vehetnek fel negatív értéket, kerüljük a normális eloszlás használatát. Másfelől, a korlátos trianguláris eloszlás megbízhatóan biztosítja az adatok átfogó reprezentációját, azonban kizárhat az egy-két olyan kirívó értéket, amelyeknek meg kellene jelenniük a szimulációs modellben.

Egy másik, sokkal gyakorlatiasabb megfontolás az, hogy néhány eloszlás paraméterezése könnyebb mint másoké. Ha bármely olyan változtatást készülünk eszközölni a modellben, amely érinti az eloszlási paramétereinek a beállítását (pl. érzékenységi elemzés vagy különböző lehetőségek vizsgálata), jobb olyan módszerekkel dolgozni, amelyek paraméterei könnyebben érthetőek. Az érkezési időközök vizsgálata során például a *Weibull*- és az exponenciális eloszlások közel azonos minőségű illesztést produkálnak, de sokkal könnyebb változtatni az érkezési időközöket az exponenciális eloszlás középértékével, mint a *Weibull*-eloszlásnál, mivel ennél az átlagérték a paraméterek bonyolult függvénye.

Ha azon aggódunk, vajon jól választottunk-e, minden egyes lehetőséggel futtassuk le a programot, és nézzük meg, van-e szignifikáns eltérés a kapott eredmények között (ebben az esetben a jó következtetések levonásához meg kell várni, amíg a modell teljesen elkészül). Tovább vizsgálódhatunk az eloszlások illesztését befolyásoló tényezők területén (például illeszthetőségi próbák végzésével) az **Arena** online Help-je vagy más források segítségével, mint Pegden, Shannon, és Sadowski (1995), vagy Law és Kelton (2000). Egyébként tanácsos az előzőleg említett minőségi és gyakorlati szempontok alapján dönteni.

Mielőtt lezárnánk az *Input Analyzer* ismertetését, szeretnénk egy korábban már említett témára visszatérni, nevezetesen arra, hogy mi a teendő azokban az esetekben, ha az adatoknak több csúcsa van, vagy esetleg néhány extrém értéke. Ezek az esetek rendszerint lehetetlenné teszik, hogy a hagyományos elméleti módszerrel eredményt érjünk el. Mint azt már korábban említettük, megoldás lehet az empirikus eloszlás használata, ami talán egyben a legjobb megoldás is, kivéve, ha a mintaméret nagyon kicsi. Szélsőséges értékek esetén feltétlenül ellenőrizzük az adatok helyességét, nehogy valamilyen hibát (esetleg csak egy elírást) tartalmazzanak. Ha az adatérték hibásnak bizonyul, és nincs lehetőség annak módosítására, egyszerűbb, ha inkább töröljük.

Bármilyen furcsaság (többszörös csúcs vagy szélsőséges értékek) esetén hasznos lehet, ha az adathalmazt két részre osztjuk, mielőtt beolvassuk az *Input Analyzer*-be. Tételezzük fel például, hogy gépek állásidejét leíró adatokkal rendelkezünk, és az adatokból szerkesztett hisztogram két csúcsértéket mutat, tehát az adathalmaz bimodális. Miután újra megvizsgáljuk az adatok eredeti forrását, rájövünk, hogy ezek az állásidők két különböző szituációból erednek: műszaki hibából és rendszeres karbantartásból. Ha két részre bontjuk a halmazt, akkor kiderül, hogy a műszaki hiba miatt bekövetkezett állásidők sokkal hosszabbak, mint a karbantartásból eredők, és innen származik a két különböző csúcsérték. Ilyenkor (még mielőtt elkezdenénk használni az *Input Analyzer*-t) célszerű szétválasztani az adatokat, és a két adathalmazokhoz különböző eloszlást rendelni, majd úgy megváltoztatni a modell logikai felépítését, hogy az a kétfajta állásidőt egymástól függetlenül kezelje.

Egy adathalmaz szétválasztását közvetlen az *Input Analyzer*-ben is elvégezhetjük, ami egyszerűen a tartományhatárok kijelölésén alapul. Az adatfájl betöltése után válasszuk az *Options* > *Parameters* > *Histogram* parancsokat, hogy meghatározzuk a szétválasztás helyét vagy helyeit, az alsó és a felső értékeket. Ezek az értékek fogják definiálni a teljes adathalmaz részhalmazainak határait. Bimodális (két csúcsot tartalmazó) adatok esetén először a baloldali csúcsra fókuszálva, az alsó értéket meghagyjuk, a felső értéket pedig úgy jelöljük ki, hogy az a két csúcs között a legalacsonyabb pontra essen, majd a jobboldali csúcsra koncentrálva az utóbbi érték lesz az alsóhatár és a teljes halmazhoz tartozó eredeti felsőérték a felsőhatár. A határok helyes kijelölése kísérletezést igényelhet. Ezt követően a legígéretesebbnek tűnő eloszlásfüggvényt mindkét részhalmazra ráillesztjük, vagy használhatjuk a *Fit All* opciót, az adattartományok ábrázolására. Ha az alsó és felső határok kijelölése előtt már illesztettünk eloszlásfüggvényt az adatokra, akkor az *Input Analyzer* a szétválasztás helyének megfelelően azonnal újra elvégzi az illesztést és kijelzi a részhalmazokra vonatkozó új eredményeket. Ha a *Fit All* opciót választjuk, akkor a különböző eloszlásokból a legjobban illeszkedő lesz kiválasztva. Az utóbbi illesztést a két részhalmazra külön kell elvégezni. Praktikus okokból a részhalmazok számát korlátozzuk kettő vagy három halmazra, mivel az eljárás időigényes és fáradtságos lehet, továbbá az sem egyértelmű, hogy hol vannak a legjobb szétválasztási pontok. A szimulációs modellben az eredeti adathalmazból generált több eloszlásfüggvény alkalmazásának egy lehetséges megoldása, hogy egy-egy részhalmazhoz tartozó eloszlásokat véletlenszerűen választjuk ki. A választások valószínűségei megfelelnek a részhalmazok relatív méreteinek. Például, ha egy 200 elemű, két csúcsot tartalmazó (bimodális) adathalmazunk van, és a szétválasztási pontot a baloldali 120 kisebb értékű, és a jobboldali 80 nagyobb értékű adat közé helyezzük, akkor a baloldali adatokra illesztett eloszlás 0,6 valószínűséggel és a jobboldali adatokra illesztett eloszlás 0,4 valószínűséggel fordul elő. Ennek az ötletnek a megvalósítását az Arena nem teljesen támogatja automatikusan, ezért a művelethez nekünk is hozzá kell tenni valamit. Ha például a szóban forgó érték valamilyen aktív idő, például egy entitás késleltetése, akkor egy lehetséges megoldás, hogy a **Decide** modult használjuk *2-way by Chance* vagy *N-way by Chance* opcióval, attól függően, hogy kettő vagy több részhalmazból kell mintát venni. A **Decide** modullal szétválasztott entitásokat **Assign** modulokhoz kapcsoljuk, ame-

lyekben a megfelelő eloszlásokból attribútumként az entitásokhoz rendeljük a megfelelő időértékeket. Ezt követően a megfelelő helyeken ezek az attribútum értékek érvényesülnek. Egy másik lehetséges megoldás az Arena kifejezések (Expression) alkalmazása, amivel részletesen később foglalkozunk.

4.5.5 Nincs adat?

Akár tetszik akár nem, időnként előfordul, hogy egyszerűen nem érhetők el olyan megbízható adatok, amelyekre a modellezéshez szükségesünk lenne. Ez számos helyzet miatt jelentkezhet, mint például egyszerűen nem létezik a rendszer, az adatgyűjtés túl drága vagy károkat okozna, nincs meg a szükséges együttműködés, illetve engedély. Ebben az esetben önkényes feltételezésekre vagy becslésekre kell hagyatkozni, amit 'ad hoc adatoknak' hívunk. Nem ígérhetünk semmilyen üdvözítő megoldást, de létezik pár javaslat, ami hasznos lehet. Bármit is tehetünk annak érdekében, hogy javítsuk eredmények megbízhatóságát és pontosságát. Ajánlatos elvégezni egy összefüggés elemzést a kimenet és az 'ad hoc' bemenet között. Felvehetünk olyan determinisztikus értékeket, amit a tanulmányban használni fogunk (esetleg a modellt többször futtathatjuk különböző értékekkel), vagy valamilyen valószínűségi eloszlással reprezentálhatjuk a rendszer-karakterisztikát.

Ha ezek az érték nem időbeli késleltetések (várakozások), hanem valamilyen más valószínűségi változókkal leírható például működési paraméterek, vagy fizikai rendszerjellemzők, mindkét esetben választhatunk konstans determinisztikus értéket, vagy néhány esetben valószínűségi eloszlást. Ha determinisztikus értéket használunk, azaz egy konstanst viszünk be modellbe (pl.: 15% meghibásodási esély), akkor ajánlatos valamiféle összefüggés elemzést végezni annak érdekében, hogy megtudjuk miként hat a paraméter változása az eredményekre. Ha az érték kisebb változása is hatással van a rendszer működésére, akkor célszerűbb a rendszerelemzést meghatározott értékhatárok között (kicsi, közepes és nagy) végezni, mint a legjobb becslésére hagyatkozni.

Ha az adatok időbeli késleltetést reprezentálnak, akkor majdnem biztos, hogy valószínűségi változót fogunk használni, hogy mind a tevékenységben rejlő változatosságot, mind az adattal kapcsolatos bizonytalanságát kiküszöböli. Az, hogy melyik eloszlást használjuk, a tevékenység természetétől, illetve az adat típusától függ. Amikor kiválasztjuk az eloszlást, akkor meg kell adnia a helyes paramétereket, amelyek becsléseken és a felmért folyamat-változatosságon alapulnak.

4.17. táblázat

Adathiány esetén alkalmazható eloszlások

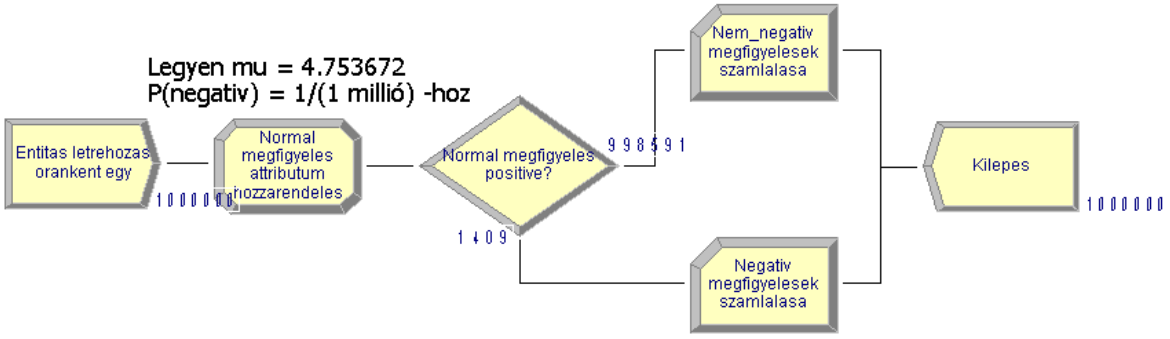
Eloszlás neve	Paraméterei	Karakterisztika	Példa
Exponenciális	átlag	nagy eltérés balról korlátos jobbról korlátos	érkezési időköz gépek meghibásodási időtartama (állandó meghibásodási szint)
Trianguláris (háromszög)	Min, Mód, Max	Szimmetrikus vagy nem szimmetrikus két oldalról korlátos	tevékenység idők
Uniform (egyenletes)	Min, Max	Minden adat egyenlő két oldalról korlátos	az alkalmazás kismértékű ismerete

Az empirikus adatok hiányában, először tekintsük az exponenciális a trianguláris, a normális és a uniform (egyenletes) eloszlást. Ezen eloszlások paramétereinek kezelés igen könnyű, és az sok alkalmazás modellezéséhez megfelelő tulajdonságokkal rendelkeznek, ami a *4.17. táblázatban* látható.

Ha az idő függetlenül váltakozik, azaz egyik adatnak nincs befolyása a másikra, a középérték becslése nem túl nagy, és nagymértékű az idő változatossága, akkor az exponenciális eloszlás a jó választás. Gyakran használt eloszlás az érkezési időközök leírására. Erre jó példa lehet a vendégek érkezése egy étterembe, vagy a rendelési igények érkezése egy raktárba.

Ha az idő egy olyan folyamatot képvisel, amelyet a legvalószínűbb idő körüli váltakozás jellemez, akkor a trianguláris eloszlást célszerű használni, mert ez az eloszlás kis és nagymértékű változatosságot mutató folyamatokat képes leírni, és a paraméterei könnyen kezelhetőek. A trianguláris eloszlást a minimum, legvalószínűbb (modus) és a maximum értékekkel határozhatjuk meg, ami a szükséges idő megbecslésének egy természetes módja a bizonyos folyamatoknál. Nagy előnye, hogy megengedi a legvalószínűbb érték körüli nem szimmetrikus eloszlást, ami általában jellemző a valós folyamatokra. Korlátos eloszlás, aminek köszönhetően, egy adatérték sem lesz kisebb a minimumnál, illetve nagyobb a maximumnál, ami lehet helyes vagy helytelen bemutatása a valós folyamatoknak.

Talán csodálkoznak azon, hogy miért kerültük eddig a leginkább ismert eloszlást, a normális eloszlást, ami az átlag és a szórás által meghatározott klasszikus haranggörbe. A visszatérített értékek az átlag körüli szimmetrikus oszlanak el. Nem korlátos eloszlás, ami azt jelenti, hogy egy nagyon kicsi és nagyon nagy értékeket is kaphatunk. Abban az esetben, ha a modellben nem használhatunk negatív értékeket (pl. egy folyamat időigénye), akkor a normális eloszlásból származó negatív mintákat az **Arena** 0 értékűre állítja. Ha az eloszlás átlaga közel 0, akkor a normális eloszlás alkalmatlan az folyamat leírására.



4.26. ábra: A 4.5 modell

4.18. táblázat

Negatív értékek előfordulása a normális eloszlásban $\sigma = 1$ szórással

Közép μ	Érkezések száma	Negatív megfigyelések száma	Negatív megfigyelések előfordulásának egzakt valószínűsége	„A sokból egy" negatív lesz (átlagosan)
3,0	1 millió	1409	0,001350	741
3,5	1 millió	246	0,000233	4298
4,0	1 millió	37	0,000032	31560
4,5	1 millió	3	0,000003	294048
4,753672	10 millió	7	0,000001	1000000

Másrészről, ha az átlag pozitív és egészen kicsit nagyobb a szórásnál, a negatív értéknek kicsi a valószínűsége, talán egy az egy millióhoz. Ez kicsinek hangzik, de ne felejtsük el, hogy hosszú idejű szimulációknál, vagy amit sokszor ismétlünk, az egy az egy millióhoz előfordulhat, főleg modern és gyors számítógépek alkalmazásakor. Ebben a ritka esetben az **Arena** a negatív értékeket 0-ra csonkítja, ha a modellben csak pozitív vagy 0 érték alkalmazható, ami lehet, hogy nem várt vagy extrém hatást okoz, vagy éppen érvényteleníti az eredményeket.

A *4.26 ábra* egy teljesen használhatatlan **Arena** modellt (4.5 modell) mutat, amelyben az entitások pontosan óránként érkeznek, a megfigyelésből a középérték: μ=3, és a szórás: σ=1 jellemzi a normál megfigyelést. Az entitások útja a két **Record** modul valamelyikéhez vezet, attól függően, hogy a „*Normal megfigyeles*" nem-negatív vagy negatív eredményű. A **Record** modulok megszámolják a nem-negatív és a negatív megfigyeléseket (tanulmányozza a folyamatábrát és futtassa a 4.5. modellt). A *4.18. táblázat* állandó σ=1 szóráshoz és több μ értékhez tartozó eredményeket mutat. Láthatjuk, hogy a negatív érték akkor is előfordul, ha az átlag μ=3 vagy μ=4 (vagy több), ha a szórás 0 fölött van. Elektronikus normál táblákból, az egzakt valószínűsége annak, hogy negatív eredményt kapjunk kiszámítható, és ez a szám megadja közelítőleg a megfigyelések számát (átlagban), amikor negatív értékeket kapunk. Nem túl sokat, figyelembe véve ezek kiszámítási idejét (hozzávetőleg egy 2 GHz-es notebook számítógépen 6 másodperc szükséges az egy millió entitás, és egy perc a 10 millió entitás létrehozásához.). A táblázat utolsó μ=4,753672 értéke azóta került a táblázatba, amióta kiderült, hogy ez az eset is közel áll az egy-az egymillióhoz esethez (az első egy millió ugyan nem produkál negatív eredményt, de a 7-a 10 millióhoz már igen, ami közelít az egy az egy millióhoz valószínűséghez). Nos, elismerésreméltó és csodálatos a nagy matematikus *Gauss*, de ettől még nem nagyon szerencsés a normális eloszlást a szimuláció bemenő eloszlásaként használni, logikailag nem negatív dolgok leírására, mint amilyen a folyamatidő, még ha az **Arena** meg is engedi ezt. Azok az adathalmazok, amelyek a normális eloszlással jól közelíthetők, más eloszlások, amelyek teljesen kizárják a negatív értékeket, mint a *Weibull, gamma, lognormal, Erlang, beta* vagy talán az *empirikus* is, majdnem hasonló szorossággal illeszthetők és emellett nem áll fenn a veszélye negatív értékek előfordulásának.

Végezetül, ha tényleg nincs sok elképzelésünk a folyamatról, de meg tudjuk becsülni a minimum és a maximum értékeket, akkor az uniform (egyenletes) eloszlást használjuk. Ez egyenlő valószínűséggel adja vissza az adatértékeket a minimum és maximum között.

4.5.6 Nem stacionárius érkezési folyamatok

Úgy tűnik, ez a speciális témakör külön bekezdést érdemel, mivel gyakran előfordul és a rendszer tulajdonságokat nagyban befolyásolja. Sok rendszer jellemzője a külső érkezés, pl. az embereket kiszolgáló rendszerek, a telefonközpontok, külső vevői igényeket alapján működő gyártási rendszerek, tapasztalati érkezések terhelései, amelyek drámaian változnak a szimuláció futási ideje alatt. Például: az ebédidőben érzékelhető roham egy gyorsétteremben, koradélután sűrűsödő telefonhívások egy műszaki szolgálat vonalaira, és egyes időszakokban tapasztalható kiugróan magas igény a gyártási rendszerekben. Az ilyen rendszerek leírására egy speciális valószínűségi modell, a *nem stacionárius Poisson folyamat* alkalmas, amely általában képes megfelelni az időben változó érkezések kihívásainak. Két kérdéssel kell foglalkozni: hogyan becsüljük vagy specifikáljuk az intenzitásfüggvényt, és azután hogyan generáljuk az érkezések mintáit a szimulációban.

A megfigyelt adatokból származó intenzitásfüggvény becslésének, illetve specifikálásának számos módja van, és a módszerek némelyike meglehetősen bonyolult. Most egy általánosan használt, sok alkalmazásban működő és igen egyszerű, a szakaszosan-állandó intenzitásfüggvény becslési eljárást mutatjuk be. Először megállapítjuk azt az időtartamot, amelyben az érkezési intenzitás közel egyenletes. Például egy telefonos ügyfélszolgálatnál a beérkező hívá-

sok egy-egy fél órás periódusban egészen egyenletesek, de a különböző periódusokban a hívások intenzitása lényegesen eltérhet. Számoljuk meg a beérkezések számát az egyes időszakokban, amelyekből kiszámíthatjuk az érkezések intenzitását az egyes időszakokra. Például: feltételezzük, hogy egy telefonos ügyfélszolgálatnál a beérkező hívások száma 8:00 és 10:00 óra a között, a 4 fél órás periódusban a következő: 20, 35, 45 és 50. Így a hívásintenzitás, a percenkénti hívások száma a periódusokban a következők szerint alakul: 20/30=0,67, 35/30=1,17, 45/30=1,50 és 50/30=1,67.

Miután megbecsültük az intenzitás függvényt, biztosítani kell, hogy az **Arena** ezt a sémát kövesse, amikor az érkezéseket generálja (az entitásokat létrehozza) a modellben. Ehhez a **Create** modul dialógusablakában, az érkezést definiáló a **Time Between Arrivals** szakaszában az érkezés típusát *Schedule*-nak, azaz ütemezettnek kell választani. Ezt követően **Schedule** adatmodul segítségével, hasonlóan ahhoz, amint azt a 4.2.2 erőforrások ütemezésénél tettünk, megszerkesztjük az érkezés intenzitás függvényt. Figyelem! Az **Arena** lehetőséget ad a tetszőlegesen alkalmazható időegységek keverésére és együttes alkalmazására, de nagyon körültekintően kell a számokat és időegységeket definiálni. Speciálisan a Schedule modul dialógusablakban a *Time Units*-tól függetlenül a *Value (Arrival Rate)* mezőben az időegység alatti érkezések számát mindig érkezés/óra egységben kell beírni, amit a program automatikusan átszámít érkezés/Time Units egységre.

4.5.7 Többváltozós és korrelált bemenő adatok

Általában azt feltételezzük, hogy az összes véletlenszerű változó, egymástól függetlenül létrehozott szimulációt eredményez, az alkalmazott eloszlástól függetlenül. Néha – habár nem ez a legjobb feltételezés – ennek fizikai okai vannak. Például a 4.2. modellben elképzelhető, hogy különböző okok miatt az egyes munkadarabok összeszerelésénél problémák adódnak bonyolultak. Ez modellben úgy érzékelhető, hogy az előkészítés több időt igényel, amelyből az következik, hogy a szerelési és az ellenőrzési idő hossza is nagyobb. Ebben az estben a két idő pozitívan korrelál egymással, és a korreláció figyelmen kívül hagyása rossz modellt és hibás adatokat eredményezhet.

Számos lehetőség van arra, hogy az ehhez hasonló szituációkat modellezzünk, megbecsüljük a szükséges paramétereket (beleértve a korreláció erősségét is) és generáljuk a szükséges megfigyeléseket a valószínűségi változókon a szimuláció alatt. Néhány eljárás ezek közül a valószínűségi változókat egy többváltozós eloszlással összekapcsolt valószínűségi vektor koordinátáiként kezeli. Ezen kívül meghatározható valamilyen képlettel leírható összefüggés a bemenő mennyiségek között. Az igazat megvallva, megbecsülni egy rendszer viselkedését és azt létrehozni a szimuláció alatt, bizony nehéz feladat. Erről lásd bővebben: Law and Kelton (2000) vagy Devroye (1986).

4.6. Összefoglalás

A fejezetnek anyagának elolvasása és megértése után, már elmondhatjuk, hogy tapasztalatokat szereztünk az **Arena** használatában. Az ismeretek szélesítése és elmélyítése érdekében javasoljuk, hogy az eddig megismert modulokban próbáljon használni más „gombokat”, illetve kísérletezzen az eddig ne használt modulokkal. Elakadás esetén éljünk az online segítség lehetőségével és tanulmányozzuk a programcsomag részét képező online könyveket. Eddig viszonylag szerényen éltünk az Arena animációs képességeivel. Az animációs lehetőségeket illetően ne legyünk türelmetlenek. A következő fejezetekben megismerhetjük az Arena sokoldalú animációs eszköztárait és statisztikai képességeit.

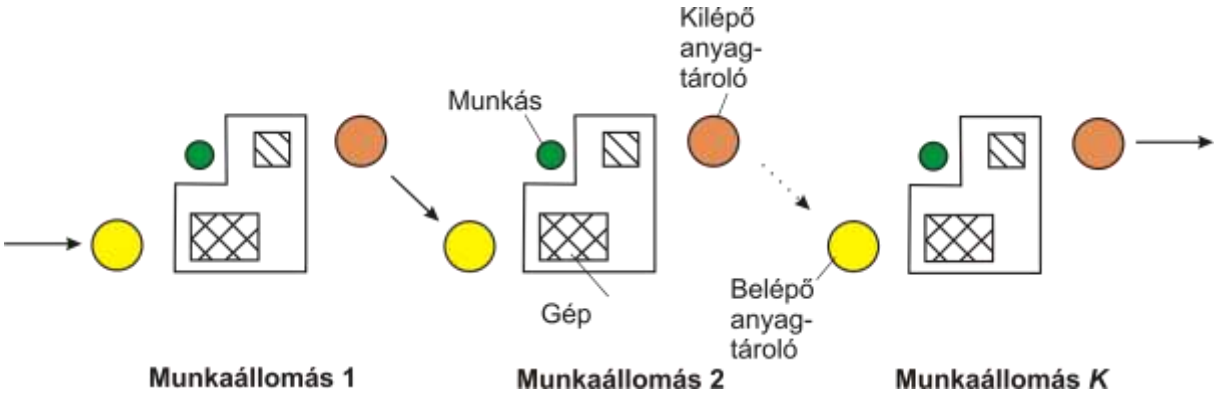
5. Csomagolósorok modellezése

Az ipari termelésben a gyártási rendszerek nagyon sok esetben gyártási lépcsők sorozatából épülnek fel. A meghatározott sorrendben elvégzendő különböző műveleteknek megfelelően a munkadarabok az egyik lépcsőből a másikba áramlanak, és végül kész vagy félkész termékként elhagyják a rendszert. Az ilyen gyártási rendszereket **gyártósoroknak** nevezzük.

A számítógéppel vezérelt gépekből álló gyártósorok a műveletek széles körének rugalmas kezelését teszik lehetővé. A gyártócsarnokokban vagy műhelyekben elhelyezett intelligens munkaállomások sorozata biztosítja a különböző termékek igényeknek megfelelő változatos megmunkálását vagy összeszerelését. Ebben a fejezetben a soros elrendezésű munkaállomások szimulációs modellezését egy csomagolósoron mutatjuk be.

5.1. A gyártósorokról

Tekintsük a *5.1. ábrán* sematikusan ábrázolt általános gyártási rendszert. A gyártási rendszer egy gyártósor, amely sorosan elrendezett munkaállomásokból és az állomások részét képező, a termékek átmeneti tárolására alkalmas közvetítő pufferekből épül fel. Minden munkaállomás egy vagy több gépből áll, egy vagy több munkással, esetleg robottal és egy korlátozott méretű munkahelyi tárolóval (ún. WIP puffer). Egy munkaállomáson a munkafolyamatok befejezése után a munkadarabok bekerülnek a következő munkaállomás WIP pufferébe (belépő anyagtároló), feltéve, hogy a következő munkaállomás puffere nem telített. Ha a puffer telített, akkor a munkadarab addig várakozik az aktuális munkaállomáson (kilépő anyagtároló), amíg a következő munkaállomás belépő pufferében fel nem szabadul egy hely.



5.1. ábra: Egy általános felépítésű gyártósor

Az *5.1. ábrán* látható általános elrendezésű gyártósor variációjaként kisebb módosításokkal számos gyakorlati modell építhető fel. Például egy modellben egy bizonyos folyamat vagy folyamatok egyszer vagy többször ismétlődhetnek, vagy az egyes munkaállomások a munkadarabokat csomagban (batch-ben) dolgozzák fel. Más esetekben a munkadarabok szállítása az egyik munkaállomásról a másikra központi fontosságú, ilyenkor a kötetlen pályás eszközökkel (transzporterekkel), vagy a kötött pályás eszközökkel (konvejorokkal) történő szállítás részletesebb modellezésre lehet szükség (lásd később a 9. fejezetben). Végül a munkadarabok elhagyják a rendszert, amely elméletileg bármely munkaállomásról megtörténhet. Általában a gyártósorok **toló-rendszerben** (push regime) működnek, amely a végtermékkészletek nagyságának kevés figyelmet szentel. A gyártósor egyszerűen annyit gyárt (tol), amennyit csak lehetséges, azt feltételezve, hogy az összes végterméket felhasználják. Különben, amikor a végtermékkészlet túl naggyá válik, a gyártósor leállhat, és ettől a ponttól a toló-rendszer átkapcsolhat **húzó-rendszerre** (pull regime). Húzó-rendszerben, amit *Kanban* gyártásnak is neveznek, a folyamat csak annyit termel, amennyi a specifikus igény. A toló és a húzó típusú

gyártási rendszerek modellezésével részletesen Altiok (1997) és Buzacott-Shanthikumar (1993) foglalkoztak.

Általában egy munkaállomáson a tárolóhely korlátozás előbb vagy utóbb felveti a szűkeresztmetszet problémát, ami egyaránt okozhat torlódást (blokkolást) és a hiányt. Ennek e jelenségnek a forrásai általában a helykorlátozások és a költségmegtakarítás. A szűkeresztmetszet explicit vagy implicit módon hat a gyártósor lépcsői közötti tárolók készletszintjére. Ha a helykorlátozás (véges puffer) valamely áramlásirányú munkaállomáson jelentkezik, akkor az a megelőző, az áramlással ellentétes irányban elhelyezkedő munkaállomásokat leállásra kényszerítheti. Ezt a jelenséget nevezzük *blokkolásnak*. A blokkolási politika nem azonos az előretervezett vagy ütemezett leállásokkal. E politika szerint, amikor egy áramlásirányú munkaállomás puffere telítődik, akkor az áramlással ellentétes irányú, megelőző munkaállomáson a munka azonnal leáll. Az áramlással ellentétes irányban elhelyezkedő munkaállomás tétlen (*Idle*) állapotba kerül addig, amíg az áramlásirányú munkaállomás pufferében elegendő üres hely nem keletkezik. Amikor ez bekövetkezik, attól a pillanattól az áramlással ellentétes irányú munkaállomáson folytatódik a feldolgozás. Az ilyen típusú blokkolást gyakran **kommunikációs blokkolásnak** nevezik, mivel az ilyen jellegű blokkolás gyakran fordul elő a kommunikációs rendszerekben. Egy másik lehetséges politika a következő munkadarab megmunkálását kezdeményezi, de a megelőző, az áramlással ellentétes irányú munkaállomáson addig késlelteti a befejezést, amíg azt az áramlásirányú munkaállomás nem képes befogadni. Az ilyen típusú blokkolási politika leggyakrabban gyártással összefüggésben fordul elő, ezért **termelésblokkolásnak** nevezik. A különböző típusú blokkolási mechanizmusokkal Perros (1994) foglalkozott.

Fontos felismerés, hogy a blokkolás visszafelé hat az áramlással ellentétes irányban, pontosabban a gyártósorban itt helyezkedő, egymást követő munkaállomásokra. Hasonlóan, néhány munkaállomás, az áramlással ellentétes oldalon elhelyezkedő munkaállomások hiányos munkadarab áramának köszönhetően kerülhet tétlen állapotba. Ezt a jelenséget a nyilvánvaló okok miatt **hiányállapotnak** nevezik. Ezért fontos felismerés az is, hogy a hiány a blokkolással szemben az áramlás irányában hat a gyártósor egymást követő, az áramlásirányban elhelyezkedő munkaállomásaira. Ténylegesen a blokkolás és a hiány együttesen hatnak, de ellentétes irányban, ami alapján a szűkeresztmetszetet jelentő munkaállomás úgy definiálható, mint amely a gyártósort két szegmensre osztja, úgymint az áramlással ellentétes irányú munkaállomásokra, amelyek blokkoltak, és áramlásirányú munkaállomásokra, amelyeket a munkadarab hiány jellemez.

Egy másik tétlenséget okozó jelenség a gyártósor meghibásodása okozta kényszerített tétlenség. Rendszerint a meghibásodott gyártósor javítása azonnal megkezdőik, mihelyt az lehetséges és a javítás befejezéséig a műveletek állnak. A váltakozó működési és meghibásodási periódusokat az Arena-ban **uptimes** és **downtimes** periódusoknak nevezzük. Nyilvánvaló, hogy a meghibásodott munkaállomás szűkeresztmetszetet jelent, blokkolja a megelőző munkaállomásokat és hiányt okoz a követő munkaállomásokon. A hiba jelentkezésekor a megszakadt művelet kezelésére ki kell találni valamilyen eljárást, amit a hiba megszűnése után végre kell hajtani. Például, a megszakított munkafolyamatot az elejétől újra kezdjük vagy elegendő folytatni. Néhány esetben a megszakított munkafolyamat nem folytatható, a munkadarabot selejtezzük.

5.2. A gyártósorok modelljei

Ha a gépek blokkolás vagy hiány miatt tételen állapotba kerülnek, az potenciálisan hatékonyságcsökkenést eredményez. Ez egyrészt a meghibásodásoknak, másrészt a munkaállomások között felhalmozott túlzott készletek okozta szűkeresztmetszetek miatt jelentkező fojtásnak

köszönhető. Továbbá a gyártósorok ritkán determinisztikusak, a véletlenszerűség a változó műveleti időből, a véletlenszerűen jelentkező meghibásodásokból, illetve az ezt követő javítások idejének a változékonyságából fakad. Ez a véletlenszerűség megnehezíti ezeknek a rendszereknek a szabályozását és a viselkedésük előrejelzését. A várható viselkedés előrejelzésében a matematikai és a szimulációs modellek segíthetnek.

A gyártósorok tervezési problémái elsődlegesen erőforrás allokálási (lekötési) problémák. Ezek a problémák magukban foglalják egy munkaállomás halmaz munkaterhelés és a puffer kapacitás lekötését a műveleti időkkel összefüggésben. Általában a gyártási rendszerekben a tervezési problémákat nagyon nehéz megoldani. Ez részben a problémák kombinatorikus természetének köszönhető.

A gyártósorok működésének analízise arra törekszik, hogy a rendszerparaméterek halmazának függvényében számszerűsített teljesítménymutatókat képezzen. A leggyakrabban használt teljesítménymutatók: az átbocsátóképesség, az átlagos készletszint a pufferekben, az állásidők valószínűsége, a blokkolás valószínűsége a szűkkeresztmetszetet jelentő munkaállomásokon, és az átlagos rendszeridő vagy gyártási idő.

A gyártási rendszerek vizsgálatakor e mutatók segítségével feltárhatjuk, hogy a hatékonyságcsökkenés mely területeken jelentkezik és azonosíthatjuk a legveszélyesebb helyeket.

5.3. Csomagolósor működésének jellemzői

Tekintsünk egy termékek csomagolására alkalmas általános csomagolósort, amely gyógyszeripari termékek, vagy élelmiszeripari termékek csomagolásának a modellezésére egyaránt használható. A sor munkaállomásokból áll, amelyek a töltés, a lezárás, a címkézés, a krimpelés és a kartonokba helyezés műveletek hajtják végre. Az általános leírás érdekében ezekre a termékekre, mint egységekre hivatkozunk.

A következő feltételezésekkel élünk:

(1) A töltő munkaállomáson mindig van elegendő anyag, azaz az állomás anyaghiány miatt soha nem áll le.

(2) A munkaállomások között a puffer nagysága 5 termékegység.

(3) A munkaállomás blokkolódik, ha a következő munkaállomás pufferében nincs elegendő hely (termelésblokkolás).

(4) A munkaállomásokon a műveleti idők a következők. töltés: 6,5 s, a lezárás: 5 s, a címkézés: 8 s, a krimpelés: 5 s, és a kartonokba helyezés: 6 s.

Megjegyezzük ezek a feltételezések a csomagolósort a toló-rendszerű gyártás kategóriájába sorolják. Annak érdekében, hogy egyszerűsítsük a dolgokat, azt feltételezzük, hogy a csomagolósor működése determinisztikus, azaz a műveleti idők nem véletlenváltozók.

A feladat a csomagolósor modelljének kidolgozása és viselkedésének analizálása. Az első munkaállomás, a töltő a rendszer motorja, amely az áramlásirányú munkaállomásokat anyaggal (termékegységekkel) látja el. Világos, hogy a sorban a leglassúbb (legkisebb kapacitású) munkaállomás (ha a munkaállomások kapacitásai különbözőek) megkülönböztetett figyelmet érdemel. A sor átbocsátóképessége megegyezik a legkisebb kapacitású munkaállomás átbocsátóképességével, továbbá a legkisebb kapacitású munkaállomáshoz viszonyítva az áramlással ellentétes irányú oldalon elhelyezkedő munkaállomások WIP pufferét a túltöltöttség jellemzi. Ezzel ellentétben a legkisebb kapacitású munkaállomáshoz viszonyítva az áramlás irányában elhelyezkedő munkaállomásokon kicsi terhelés vagy üres puffer is előfordulhat. Így a legkisebb kapacitású munkaállomás, mint szűkkeresztmetszet meghatá-

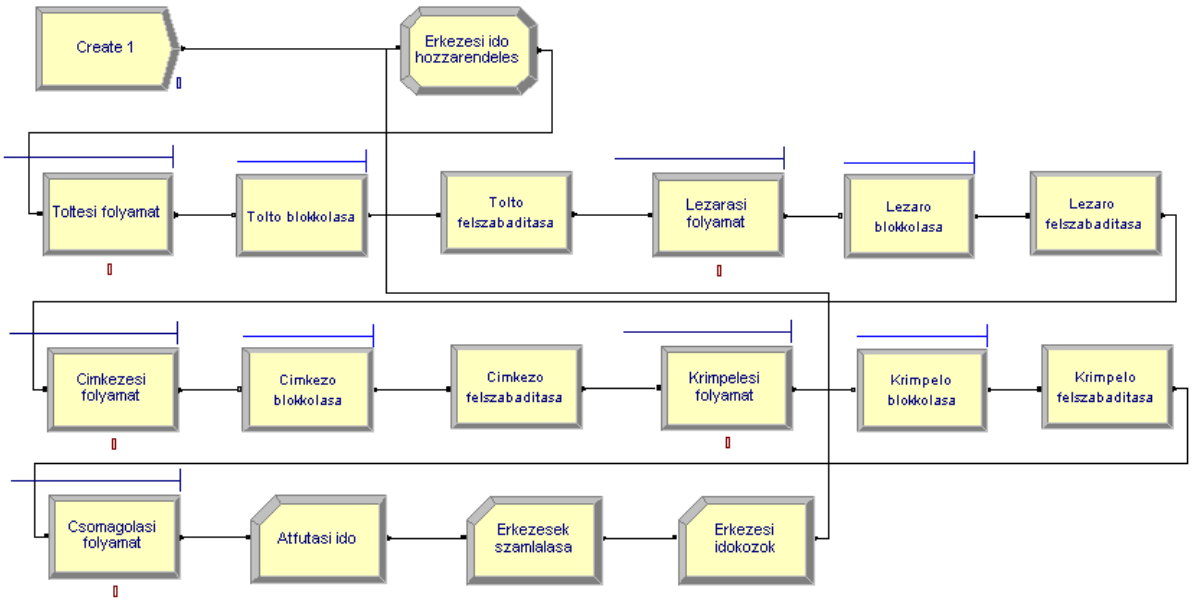
rozza a csomagolósor működését. A leírt viselkedés a determinisztikus, toló-rendszerű gyártás sajátos tulajdonsága.

5.3.1. A csomagolósor modellezése

A csomagolósor Arena modelljét az *5.2. ábra* szemlélteti, amelyben a modulok reprezentálják az egyes műveleteket, és a termékegységek az egyik modulból a másikba áramlanak. A modell logika elegendő számú termékegységgel (entitással) látja el a töltőállomást, ilyen módon biztosítja annak folyamatos működését. Ezt a célt úgy érjük el, hogy a 0 időpontban, 30 entitást hozunk létre, amelyekhez az „*ErkIdo*” nevű attribútum tartozik. Az „*ErkIdo*” értékét, a *TNOW* változót az „*Erkezesi ido hozzarendeles*” nevű **Assing** modulban rendeljük az attribútumhoz. Ezek az entitások azonnal, a 0 időpontban a töltőállomás pufferébe, pontosabban a „*Toltesi folyamat*” nevű **Process** modulhoz tartozó sorba kerülnek. Az utolsó, az „*Erkezesi idokozok*” nevű **Record** modulból kilépő entitások nem távoznak a rendszerből, hanem visszaküldjük azokat az *Erkezesi ido hozzarendeles*” nevű modulhoz. Jegyezzük meg, ez a technika csak egy modellezési eszköz arra, hogy a töltőállomáson soha ne jelentkezzen hiány, amit korábban a működési feltételek között is megfogalmaztunk. A „*Toltesi folyamat*” nevű **Process** modult elhagyó termékegységek áthaladnak a lezárás, a címkézés, a krimpelés és a csomagolás (kartonokba helyezés) tevékenységeket reprezentáló **Process** modulokon és végül a statisztikák gyűjtésére beépített **Record** modulokon. A követező részben a modell logikát részletesebben ismertetjük.

5.3.2 A részfolyamatok moduljai (5.1. modell)

A csomagolósor minden munkaállomását a **Basic Process** Panelen elérhető **Process** modulokkal modellezzük. Minden részfolyamathoz tartozik egy sor, amelyeket a modulok felett elhelyezkedő félegyenesek szimbolizálnak az *5.2. ábrán*. E sorokhoz tartozó információkat a **Queue** adatmodul megnyitása után tekinthetjük meg. Például, tekintsük a „*Toltesi folyamat*” nevű **Process** modul dialógusablakát (*5.3. ábra*). Ehhez a **Process** modulhoz tartozó sor neve „*Toltesi folyamat.Queue*” és az erőforrása a „*Tolto*”. Az *Action* mezőben a *Seize Delay* opciót választjuk, így az entitás a sorban addig várakozik, amíg a „*Tolto*” nevű erőforrás szabaddá nem válik. Mihelyt a az erőforrás elérhetővé válik, a sorban várakozó entitások közül az első leköti az erőforrást (feltéve, hogy a sor nem üres). Ezt követően a műveleti időnek megfelelően (6,5 s) a *Delay* késlelteti az entitás továbbhaladását.



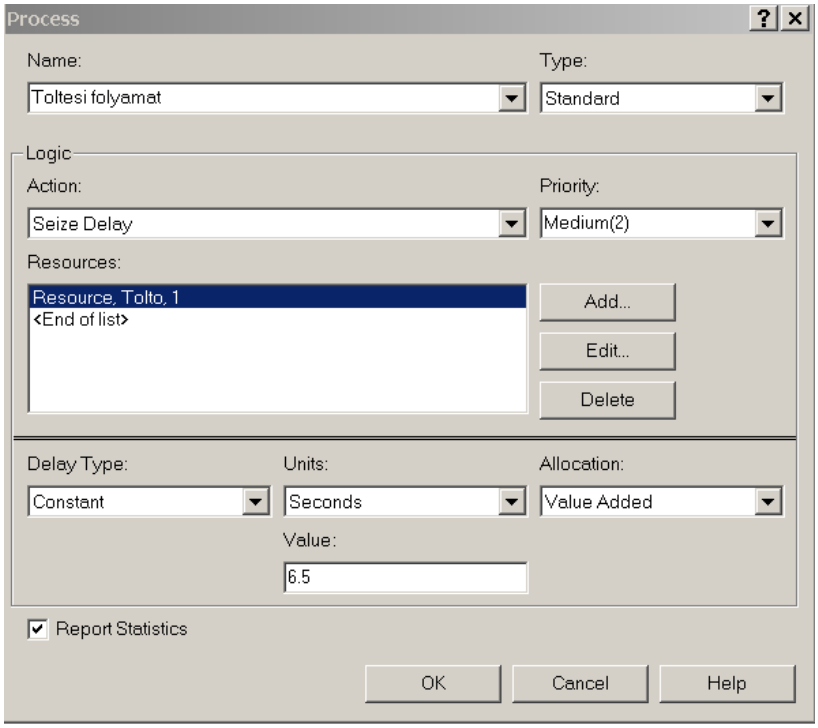
5.2. ábra: A csomagolósor Arena modellje

A „*Toltesi folyamat*” nevű **Process** modult elhagyva, az entitás belép a „*Tolto blokkolasa*” nevű **Hold** modulba, amely az előző szakaszokban részletezett blokkolást valósítja meg. Az entitás csak akkor haladhat tovább, ha a következő részfolyamat, a lezárás pufferében, pontosabban a sorában elegendő hely áll rendelkezésre. Ha van elegendő hely a sorban, akkor az entitás belép a „*Tolto felszabaditasa*” nevű **Release** modulba, ahol a „*Tolto*” nevű erőforrást felszabadítjuk és befejezzük a részfolyamatot. Valójában a **Process**, **Hold** és a **Release** modul sorozat ismétlődik az első 4 részfolyamatnál. Az utolsó munkaállomáson, a csomagolásnál („*Csomagolasi folyamat*”), már nincs szükség blokkolásra.

5.3.3 A modell blokkolása a Hold modullal

A **Hold** modul segítségével feltételhez köthetjük az entitás mozgását. Az entitás csak akkor hagyhatja el a modult, ha a *Condition* mezőben megadott feltétel teljesül. A **Hold** modulnak ezt a tulajdonságát használjuk fel a termelésblokkolás modellezésére.

Az Arena modellben (*5.2. ábra*) 4 db **Hold** modult alkalmazunk. Figyeljük meg a **Hold** modulok sorait animáló félegyeneseken megjelenő entitásokat, amelyek az adott munkaállomás pufferében felhalmozódó termékegységek számát mutatják. Például a „*Tolto blokkolasa*” nevű **Hold** modult az *5.4. ábra* mutatja, amely a töltési folyamathoz tartozik.



5.3. ábra: „*Toltesi folyamat*” nevű **Process** modul dialógusablaka

A **Hold** modulban a *Condition* mezőben (*5.4. ábra*) a feltétel:

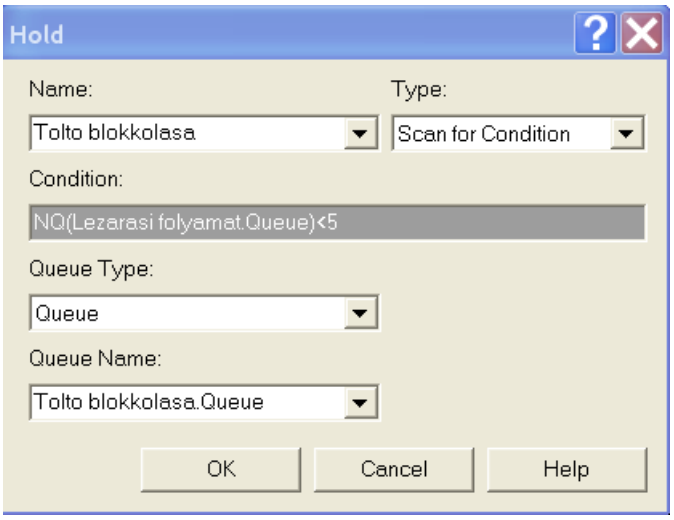
$$NQ(\textit{Lezarasi folyamat.Queue}) < 5.$$

Más szóval, azt vizsgáljuk, hogy a lezárási folyamat puffere képes-e fogadni a töltő munkaállomáson kilépésre várakozó entitást. Ha feltétel teljesül, az adott termékegység elhagyja a töltő munkaállomást, és az erőforrás felszabadítása után azonnal bekerül a lezárás munkaállomás pufferébe, azaz csatlakozik a „*Lezarasi folyamat.Queue*” nevű sorhoz. Különben az entitás blokkolódik, és a **Hold** modulban marad, pontosabban a „*Lezarasi folyamat.Queue*” nevű sorban addig, amíg a feltétel nem teljesül. Mihelyt a termékegység elhagyta a **Hold** modult, késlekedés nélkül belép a „*Tolto felszabaditasa*” nevű **Release** modulba, amely felszabadítja a „*Tolto*” erőforrást a következő entitás számára. Az entitás pedig csatlakozik a „*Leza-*

rasi folyamat.Queue” sorhoz. Továbbá, mivel a töltő munkaállomás erőforrás kapacitása 1, bármely időpontban csak egy entitás lehet a „*Tolto blokkolasa.Queue*” nevű sorban. Ez azt jelenti, hogy

$$\Pr\{NQ(\textit{Tolto blokkolasaQueue}) > 0\} = \frac{\textit{Tolto blokkolasa.Queue foglalt}}{\text{Az összes szimulációs idő}},$$

azaz, annak valószínűsége, hogy a „*Tolto blokkolasa.Queue*” nevű nagyobb, mint 0, a sor átlagos hosszával egyenlő. Ezt a megállapítást később felhasználjuk arra, hogy meghatározzuk a részfolyamatok blokkolásának valószínűségét. Egyszerűen a **Statistic** adatmodulban, mint *Time Persistent* statisztikát definiáljuk az *NQ*(*X-edik folyamat.Queue*) kifejezéssel.



5.4. ábra: A „*Tolto blokkolasa*” nevű **Hold** modul dialógusablaka

Vegyük észre, hogy a modellben minden **Hold** modult egy **Release** modul követ. Következésképpen az entitás **Hold** modulban töltött időtartama növeli a megelőző **Process** modul erőforrásának a foglaltsági időtartamát. Ezt a megállapítást később, az 5.3.6 szakaszban felülvizsgáljuk, hogy megállapíthassuk, az erőforrás effektív kihasználtságát.

5.3.4 Erőforrások és sorok

A Basic Process panelen elérhető **Resource** modul betöltése után megjelenő dialógusablak táblázatnézete a modell összes erőforrását mutatja (*5.5. ábra*).

Resource - Basic Process

	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures
1	Tolto	Fixed Capacity	1	0.0	0.0	0.0		0 rows
2	Lezaro	Fixed Capacity	1	0.0	0.0	0.0		0 rows
3	Krimpelo	Fixed Capacity	1	0.0	0.0	0.0		0 rows
4	Csomagolo	Fixed Capacity	1	0.0	0.0	0.0		0 rows
5	Cimkezo	Fixed Capacity	1	0.0	0.0	0.0		0 rows

5.5. ábra: A **Resource** modul dialógusablaka táblázatnézetben

Hasonlóan a **Queue** modul táblázatnézete a modellben definiált sorokat jeleníti meg, beleértve a blokkolt sorokat is (*5.6. ábra*).

A fejlécben a *Type* mező alatti oszlopban választott opció (*5.6. ábra*) meghatározza a sor működését. Az ábrán valamennyi sor a *FIFO* elv szerint működik, de a legördülő menüből *LIFO*, *Lowest Attribute Value* (legkisebb attribútum érték), *Highest Attribute Value* (legnagyobb attribútum érték) is választható. Az utóbbi kettő opció azt jelenti, hogy az entitás kiválasztása

az entitáshoz tartozó partikuláris attribútum értéke alapján történik. A legegyszerűbb esetben az entitáshoz egy prioritás attribútumot rendelünk. Más esetekben az ún. *STP* (shortest processing time) szabályt is alkalmazhatjuk, amely lehetővé teszi a legrövidebb feldolgozási idő szerinti kiválasztást. E szabály alkalmazásához deklarálni kell egy attribútumot, mondjuk „*Feldolgozasi ido*” néven, és a *Type* oszlopban a *Lowest Attribute Value* vagy a *Highest Attribute Value* opciót kell választani. Ez a választás automatikusan létrehoz egy új oszlopot *Attribute Name* fejléccel az *5.6. ábrán* látható táblázatban, amelybe a felhasználónak a megfelelő attribútum nevet, esetünkben a „*Feldolgozasi ido*”-t kell beírnia.

Queue - Basic Process

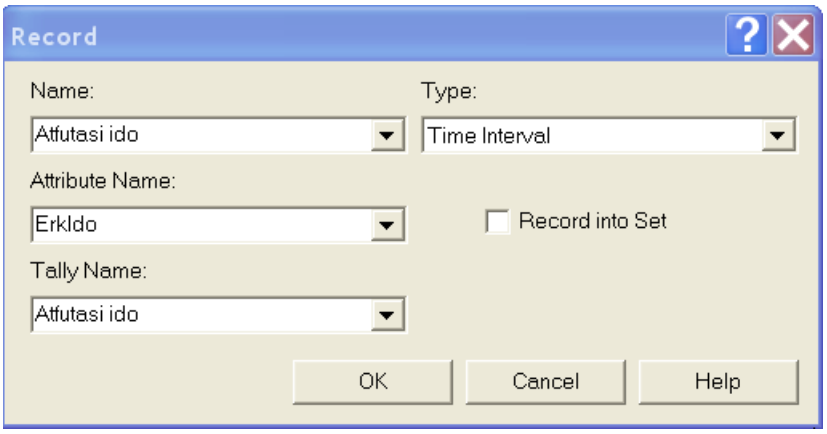
	Name	Type	Shared	Report Statistics
1	Toltesi folyamat.Queue	First In First Out	☐	☑
2	Lezarasi folyamat.Queue	First In First Out	☐	☑
3	Cimkezesi folyamat.Queue	First In First Out	☐	☑
4	Cimkezo blokkolasa.Queue	First In First Out	☐	☑
5	Lezaro blokkolasa.Queue	First In First Out	☐	☑
6	Krimpelesi folyamat.Queue	First In First Out	☐	☑
7	Krimpelo blokkolasa.Queue	First In First Out	☐	☑
8	Csomagolasi folyamat.Queue	First In First Out	☐	☑
9	Tolto blokkolasa.Queue	First In First Out	☐	☑

5.6. ábra: A **Queue** modul dialógusablaka táblázatnézetben

A véges kapacitású sor alkalmazása a modellezésben a blokkolással analóg eredményre vezet. Ha egy sort szeretnénk megosztani több **Seize** modul vagy **SEIZE** block között, akkor a *Shared* nevű oszlopban a jelölő négyzetet be kell kapcsolni.

5.3.5 Statisztikák gyűjtése

A „*Atfutasi ido*” nevű **Record** modul dialógusablaka az *5.7. ábrán* látható, amely a az entitások belépése és kilépés között eltelt időt méri és átlagolja az entitáshoz tartozó „*ErkIdo*” attribútum felhasználásával. A méréshez a *Type* mezőben a *Time Interval* opciót választjuk, és a *Tally Name* mezőben statisztikának a „*Ataramlasi ido*” nevet adjuk.



5.7. ábra: Az „*Atfutasi ido*” nevű **Record** modul dialógusablaka

A csomagolósor modell **Statistic** adatmoduljának dialógusablaka az *5.8. ábra* látható. Az 1-3. és a 9-10. sorok a részfolyamatok (töltés, lezárás, krimpelés, címkézés és csomagolás) *Frequency* **típusú** (az állandósult állapot valószínűségi becslései) statisztikái olvashatók. A kifejezésekkel leírt, specifikus folyamatot jellemző események valószínűségei *Time Persistent* típusú statisztikák (5-8. sorok). Például annak valószínűsége, hogy a töltési folyamat sorában várakozó entitások száma kettő vagy több, a következő kifejezéssel írható le:

NQ (*Toltesi folyamat.Queue*) >= 2.

Statistic - Advanced Process

	Name	Type	Expression	Report Label	Output File	Frequency Type	Resource Name	Report Label
1	Tolto allapotok	Frequency		Tolto allapotok		State	Tolto	Tolto allapotok
2	Lezaro allapotok	Frequency		Lezaro allapotok		State	Lezaro	Lezaro allapotok
3	Krimpelo allapotok	Frequency		Krimpelo allapotok		State	Krimpelo	Krimpelo allapotok
4	Teljesitmeny db per sec	Output	1/TAVG(Erkezesi idokozok)	Teljesitmeny db per sec		Value		Teljesitmeny db per sec
5	Tolto blokkolva	Time-Persistent	NQ(Tolto blokkolasa.Queue)	Tolto blokkolva		Value		Tolto blokkolva
6	Lezaro blokkolva	Time-Persistent	NQ(Lezaro blokkolasa.Queue)	Lezaro blokkolva		Value		Lezaro blokkolva
7	Krimpelo blokkolva	Time-Persistent	NQ(Krimpelo blokkolasa.Queue)	Krimpelo blokkolva		Value		Krimpelo blokkolva
8	Cimkezo blokkolva	Time-Persistent	NQ(Cimkezo blokkolasa.Queue)	Cimkezo blokkolva		Value		Cimkezo blokkolva
9	Cimkezo allapotok	Frequency		Cimkezo allapotok		State	Cimkez	Cimkezo allapotok
10	Csomagolo allapotok	Frequency		Csomagolo allapotok		State	Csomagolo	Csomagolo allapotok

5.8. ábra: A **Statistic** adatmodul dialógusablaka

A 4. sorban egy kifejezéssel, az átlagos érkezési időköz reciprok értékével definiált *Output* típusú statisztika az egy entitásra eső átlagos érkezési időközt számítja. A szimuláció végén ez a kifejezésből számított eredmény a rendszer teljesítményét (átbocsátóképességét) mutatja.

Végül az 5-8. sorok a töltés, lezárás, krimpelés és címkézés blokkolásának valószínűségét számítják (emlékezzünk arra, hogy a csomagolásnál nincs blokkolás). Megjegyezzük, a kifejezések a **Hold** modulokkal blokkolt sorok hosszát (a sorokban aktuálisan várakozó entitások számát) az időben átlagolják. Mivel minden blokkolt sorban legfeljebb egy entitás várakozik, a számított időátlag éppen a blokkolás valószínűségével egyenlő.

5.3.6 A szimuláció eredményei

A csomagolósor modelljének futási idejét 100.000 másodpercre állítjuk. A szimuláció végén a *Riport panelen* elérhető jelentések közül a sorokról (*Queues*) készített jelentést az *5.9. ábra* mutatja be.

Vegyük észre, hogy a jelentésben minden sorhoz *Waiting Time* (várakozási idő) és *Number Waiting* (várakozó entitások száma, azaz a sor hossza) statisztika tartozik. A „*Toltesi folyamat.Queue*” nevű sorban maximálisan 29 termékegység várakozott (emlékezzünk arra, hogy a szimuláció indításakor 30 entitást hoztunk létre), és a sor átlagos hossza 15,64 termékegység. A „*Lezarasi folyamat.Queue*” nevű sor az idő jelentős részében telített, a sor átlagos hossza 4,98, és az átlagos várakozási idő 39,9 s. A” *Cimkezesi folyamat.Queue*” nevű sor, hasonlóan az előzőhöz, majdnem mindig telített, az átlagos entitás szám a sorban 4,99 termékegység. Ezzel szemben a krimpelési és a csomagolási részfolyamatok sorai üresek voltak a szimuláció alatt. A sorokról szóló jelentésben a **Hold** modulokhoz tartozó blokkolt sorok statisztikái is megtalálhatók. Ezeket később elemezzük a blokkolás valószínűségével összefüggésben.

A **Riport** panel erőforrásokról (*Resources*) szóló jelentését a az *5.10. ábra* jeleníti meg. Az *Inst Util* oszlopban látható, hogy az erőforrások közül a „*Cimkezo*”, a „*Lezaro*” és a „*Tolto*” kihasználtsága 100%-os. A „*Tolto*” kivételnek tekinthető, mivel a rendszert eleve úgy terveztük, hogy a töltési folyamat működéséhez folyamatosan rendelkezésre áll a szükséges számú entitás. Ezeknél az erőforrásoknál nem jelentkezik a tétlen (*Idle*) állapot. Ugyanakkor a „*Csomagolo*” és a „*Krimpelo*” az idő 75, illetve 62%-ban foglalt. Az erőforrásokról szóló riport oszlopai további statisztikákat is tartalmaznak, amelynek az elemzését az olvasóra bízzuk.

A felhasználó által definiált statisztikákat összefoglaló jelentés az *5.11. ábrán* tanulmányozható. Ezeket a statisztikákat a **Statistic** és a **Record** modulokban definiáltuk. Az ábrán a *Tally* szakaszban található az „*Erkezesi idokozok*” és az „*Ataramlasi ido*” nevű statisztika. Az érkezési időközök becsült középértéke 8 s (a *Minimum* és a *Maximum* oszlopokból kiolvasható

értékekből következik, hogy minden megfigyelés értéke egyforma, 8 s). A triviálisnak látszó eredmény ellenére, ez a statisztika a modell verifikációjának köszönhető, amivel az 5.4. szakaszban részletesebben foglalkozunk. Az töltési és a csomagolási folyamat közötti átáramlási idő középértéke 239,78 s, ami 12497 megfigyelésen alapul (ennyi entitás feldolgozása fejeződött be, lásd a „*Erkezesek szamlalasa*” nevű statisztikát.). A minimális és a maximális átáramlási idő 30,5 és 262,5 s. Vegyük észre, hogy az átáramlási idő középértéke (239,78 s) a minimálistól (30,5 s) távolabb és a maximálishoz (262,5 s) közelebb helyezkedik el.

A *Time Persistent* szakasz a blokkolások valószínűségéről ad tájékoztatást. A számokból világosan látszik, hogy a rendszer működésében a lezárási folyamatnál jelentkezik akut probléma, amely az idő 37%-ban blokkolt, és ami a lezárást követő hosszú címkézési időnek köszönhető. A lezárási folyamat blokkolása hatással van a töltési folyamatra, amely az 18,7 %-ban blokkolt.

Az *Output* szakasz a rendszer átbocsátóképességéről (0,125 db/s) közöl becsült információt, ami egyenértékű a *Tally* szakaszban közölt 8 s/db érkezési időköz értékkel.

A *Counter* szakaszban olvasható érték („*Erkezesek szamlalasa*”=12497) azt jelenti, hogy a szimulációs idő alatt 12497 entitás haladt át a folyamaton.

6:31:23 **Queues** október 12, 2011

Csomagolósor Hold modulokkal blokkolva Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 100 000,00 Time Units: Seconds

Cimkezesi folyamat.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	39.9557	(Correlated)	0	40.0000
Other	**Average**	**Half Width**	**Minimum**	**Maximum**
Number Waiting	4.9951	(Correlated)	0	5.0000

Cimkezo blokkolasa.Queue

Other	Average	Half Width	Minimum	Maximum
Number Waiting	0	(Insufficient)	0	0

Csomagolasi folyamat.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	0	0,000000000	0	0
Other	**Average**	**Half Width**	**Minimum**	**Maximum**
Number Waiting	0	(Insufficient)	0	0

Krimpelesi folyamat.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	0	0,000000000	0	0
Other	**Average**	**Half Width**	**Minimum**	**Maximum**
Number Waiting	0	(Insufficient)	0	0

Krimpelo blokkolasa.Queue

Other	Average	Half Width	Minimum	Maximum
Number Waiting	0	(Insufficient)	0	0

Lezarasi folyamat.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	39.8641	(Correlated)	0	40.0000
Other	**Average**	**Half Width**	**Minimum**	**Maximum**
Number Waiting	4.9860	(Correlated)	0	5.0000

Lezaro blokkolasa.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	2.9997	(Correlated)	0.5000	3.0000
Other	**Average**	**Half Width**	**Minimum**	**Maximum**
Number Waiting	0.3743	(Correlated)	0	1.0000

Toltesi folyamat.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	124.97	(Correlated)	0	188.50
Other	**Average**	**Half Width**	**Minimum**	**Maximum**
Number Waiting	15.6448	(Correlated)	0	29.0000

Tolto blokkolasa.Queue

Time	Average	Half Width	Minimum	Maximum
Waiting Time	1.5000	(Correlated)	1.0000	1.5000
Other	**Average**	**Half Width**	**Minimum**	**Maximum**
Number Waiting	0.1868	(Correlated)	0	1.0000

5.9. ábra: Jelentés a sorokról

6:30:10 **Resources** október 12, 2011

Csomagolósor Hold modulokkal blokkolva Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 100 000,00 Time Units: Seconds

Resource Detail Summary

Usage

	Inst Util	Num Busy	Num Sched	Num Seized	Sched Util
Cimkezo	1,00	1,00	1,00	12 499,00	1,00
Csomagolo	0,75	0,75	1,00	12 497,00	0,75
Krimpelo	0,62	0,62	1,00	12 498,00	0,62
Lezaro	1,00	1,00	1,00	12 505,00	1,00
Tolto	1,00	1,00	1,00	12 511,00	1,00

5.10. ábra: Jelentés az erőforrásokról

6:28:14 **User Specified** október 12, 2011

Csomagolósor Hold modulokkal blokkolva Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 100 000,00 Time Units: Seconds

Tally

Between	Average	Half Width	Minimum	Maximum
Erkezesi idokozok	8.0000	0,000000000	8.0000	8.0000

Interval	Average	Half Width	Minimum	Maximum
Atfutasi ido	239.78	(Correlated)	30.5000	262.50

Counter

Count	Value
Erkezesek szamlalasa	12,497.00

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Cimkezo blokkolva	0	(Insufficient)	0	0
Krimpelo blokkolva	0	(Insufficient)	0	0
Lezaro blokkolva	0.3743	(Correlated)	0	1.0000
Tolto blokkolva	0.1868	(Correlated)	0	1.0000

Output

Output	Value
Teljesitmeny db per sec	0.1250

5.11. ábra: Jelentés a felhasználó által definiált statisztikákról

6:35:51 **Frequencies** október 12, 2011

Csomagolósor Hold modulokkal blokkolva Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 100 000,00 Time Units: Seconds

Cimkezo allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	1	99,988.50	99.99	99,99
IDLE	1	11.5000	0.01	0,01

Csomagolo allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	12,497	6.0000	74.98	74,98
IDLE	12,498	2.0018	25.02	25,02

Krimpelo allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	12,498	5.0000	62.49	62,49
IDLE	12,498	3.0013	37.51	37,51

Lezaro allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	29	3,446.62	99.95	99,95
IDLE	29	1.6552	0.05	0,05

Tolto allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	1	100,000.00	100.00	100,00

5.12. ábra: Jelentés a gyakorisági statisztikákról

A modell **Riport** panelen elérhető *Frequencies* statisztikáját az *5.12. ábra* mutatja. Ez a statisztika az erőforrások *busy* (foglalt), *idle* (tétlen) állapotainak valószínűségéről ad tájékoztatást. A töltő, a címkéző, a krimpelő és a csomagoló erőforrások közvetlen megfigyelt *busy* állapotának valószínűségei tartalmazzák a blokkolás valószínűségét is, mivel a blokkolás időtartama alatt az erőforrás nem működik, de foglalt állapotban van. Ezért az erőforrások effektív kihasználtságát (annak az időtartamnak a valószínűségét, amikor az erőforrás ténylegesen működik) úgy tudjuk becsülni, hogy a becsült blokkolási valószínűséget (*5.11. ábra Time Persistent* szakasz) kivonjuk a *busy* állapot becsült valószínűségéből. A művelet elvégzése után kapott eredményeket, az effektív kihasználtságokat az *5.1. táblázat* foglalja össze.

5.1 táblázat

A részfolyamatok effektív kihasználtsága

Folyamat	Töltés	Lezárás	Címkézés	Krimpelés	Csomagolás
Kihasználtság	0,813	0,625	0,999	0,625	0,750

5.4. A rendszer viselkedése és a modell verifikációja

Az 5.3. alfejezetben bemutatott modell általánosítva úgy is tekinthető, mint állandó műveleti idővel jellemzett, munkaállomások sorozata, azzal a kikötéssel, hogy az első munkaállomás soha nem tételen. Így a rendszer teljesítményét (átbocsátóképességét) a leglassúbb (legkisebb teljesítményű) munkaállomás határozza meg.

Például az 5.3. szakaszban megismert modellben a címkéző munkaállomás (erőforrása a „*Címkezo*”) a leglassúbb, a műveleti ideje 8 s/termékegység. A címkéző munkaállomás kihasználtsága értelemszerűen 100%, mivel minden megelőző munkaállomás kisebb műveleti idővel működik. Tekintettel arra, hogy a megelőző, a töltő és a lezáró munkaállomások gyorsabbak, mint a címkéző, a címkéző puffere alkalmanként biztosan telítődik, ami miatt először a lezáró és később a töltő munkaállomások blokkolódnak. Ennek következtében ezeken a munkaállomásokon a blokkolás valószínűsége kiemelkedően magas. Ugyanakkor a címkéző munkaállomás után elhelyezkedő munkaállomásokon (krimpelő és csomagoló) előfordul az

Idle (tételen) állapot, mivel ezeken a helyeken a műveleti idő rövidebb, mint a címkéző munkaállomáson. Következésképen a címkéző és a krimpelő soha nem blokkolt. A címkéző munkaállomást 8 másodpercenként hagyják el a termékegységek, ami megegyezik a követő munkaállomás érkezési időközével, és meghatározza a rendszer teljesítményét (1/8=0,125 termékegység/s). Értelemszerűen, ez egyenlő a többi munkaállomás teljesítményével. A teljesítmény a kihasználási tényező és az átlagos műveleti idő hányadosaként is számítható. Például a töltő munkaállomáson 0,813/6,5=0,125 vagy a lezáró munkaállomáson 0,625/5=0,125.

A szimulációs modell verifikációjához a következőkben ellenőrizzük az átlagos átáramlási időt és a rendszerben tartózkodó termékegységek átlagos számát. Az utóbbi a *Little* formulával számítható. A *Little*-formula szerint a rendszerben levő igények átlagos száma megegyezik az igények intenzitásának és a rendszerben töltött átlagos időnek a szorzatával. Definiáljuk az *S* alrendszert, mint a folyamatok sorozatát a „*Toltesi folyamat*” modultól a „*Csomagolasi folyamat*” modulig (*S* magában foglalja a töltési folyamat előtti a nyersanyagtárolót is). Az átlagos átáramlási idő *S*-en keresztül, a „*Toltesi folyamat.Queue*”-ba való érkezéstől az „*Erkezesi idokoz*” modulba való belépésig eltelt idő. A becsült átlagos átáramlási idő 239,78 s (lásd a *5.11. ábrát*). A termékegységek átlagos száma *S*-ben $\overline{N}_S$ éppen az átlagos puffer tartalmak összege (lásd az *5.9. ábrán* a *Number Waiting* sorokat) plusz a kiszolgálás alatt álló termékegységek átlagos száma (a *Busy* állapot átlagos időtartama alatt, lásd az *5.12. ábrát*):

(5.1) $\overline{N}_S = 15{,}6448 + 4{,}9860 + 4{,}9551 + 1{,}0 + 0{,}9995 + 0{,}9999 + 0{,}6249 + 0{,}7498 = 29{,}66,$

amely közel áll az elméleti várhatóértékhez, ami 30 (vegyük észre, hogy ez a szám a rendszerben cirkuláló entitások száma). Így a *Little* formula

(5.2) $$\overline{N}_S = \overline{o}\overline{F}_S$$

ahol: $\overline{N}_S$ a termékegységek átlagos száma az *S*-ben, $\overline{o}$ az *S* átbocsátóképessége, (az *S* átbocsátóképessége az érkezési ráta stacionárius állapotban), és $\overline{F}_S$ az átlagos átáramlási idő. Felhasználva az *5.11. ábra* eredményeit, az

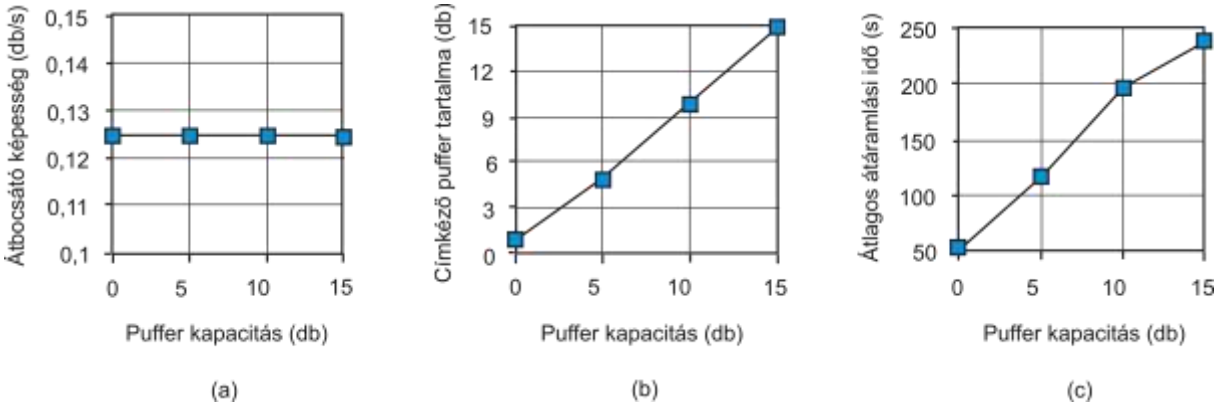
(5.3) $$\overline{N}_S = 239{,}78 \times 0{,}125 = 29{,}9725.$$

Mivel az 5.1 és az 5.2 egyenletek baloldalai azonosak, ezért felhasználhatók a modell becsült értékeinek verifikálására. A szimulációs idő növelésével az 5.1 és az 5.3 egyenletekből számított eredmények közötti eltérésnek progresszíven csökkennie kell.

A következő vizsgálat, amit elvégzünk az ún. **érzékenységi analízis**, amellyel azt tanulmányozzuk, hogy a puffer kapacitások hogyan hatnak a rendszer viselkedésére. Ez az analízis a blokkolás természetének megértését segíti (a fojtás jelensége a helytelenül megválasztott kiszolgálási teljesítményekre és a véges puffer kapacitásokra vezethető vissza). Az *5.13. ábra* három teljesítményjellemző változását szemlélteti a lezáró és a címkéző pufferek összesített kapacitása függvényében. Balról jobbra a jellemzők sorban: *S* teljesítménye (átbocsátóképessége), a címkéző puffer tartalma (WIP), és az átlagos átáramlási idő *S*-ben, a töltő puffer kivételével. (Ha töltő pufferén való átáramlást is figyelembe vennénk, akkor az átáramlási idő állandó lenne. Miért?).

Érdekes, hogy a puffer méret az *S* átbocsátóképességére nincs hatással, ugyanakkor a címkézési folyamatban a feldolgozás alatt álló és a feldolgozásra várakozó (WIP) termékegységek száma és az átáramlási idő a puffer méret függvényében növekszik. E megfigyelések megértéséhez emlékezzünk arra, hogy a rendszerben a részfolyamatok műveleti időtartamai állandó értékek. Továbbá ismert, hogy a puffereket általában azért helyezzük a gyártósorokba, hogy a véletlenszerűen változó műveleti idő, véletlenszerű leállások, stb. miatt, véletlenszerűen vál-

tozó anyagáramokat kiegyenlítsék. Az ilyen események okozzák a változékonyságot, és ezek lassítják a termékmozgást. Olyan rendszerekben azonban, ahol nincs változékonyság (amilyen a mi rendszerünk), a pufferek mérete nincs hatással az átbocsátóképességre, amint az, az *5.13/a ábrán* is látható. Ugyanakkor a szűkkeresztmetszetet jelentő munkaállomások előtt elhelyezkedő pufferek méretének a növelése a WIP szint emelkedéséhez vezet, és következésképpen hosszabb átáramlási időhöz. Ezzel szemben a szűkkeresztmetszet mögött elhelyezkedő munkaállomások pufferei a kapacitásuktól függetlenül mindig üresek.



5.13. ábra: A lezáró és a címkéző pufferek méretek egyidejű növelésének hatása
a) *S* átbocsátóképességére, b) a termékegységek átlagos számára a címkéző pufferben (WIP), és c) az átlagos átáramlási időre

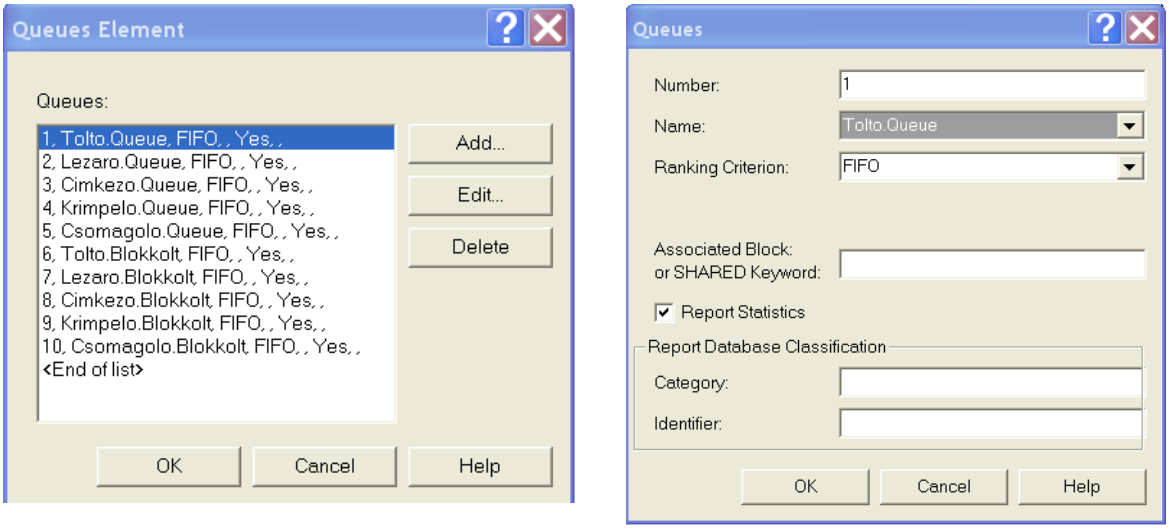
Összegezve az érzékenységvizsgálat eredményeit, a determinisztikus gyártósorok tekintetében néhány értékes és meglepő törvényszerűséget sikerült feltárni. Nevezetesen, pufferek méretének a növelése káros hatással lehet a rendszer működésére. Egyrészt a nagyobb puffer méret nem növeli az átbocsátóképességet, másrést nagyobb készleteket eredményez, ami költségnövekedést okozó tőkelekötéssel párosul. Még rosszabb, hogy növeli az átfutási időt, aminek a hatása költségnövekedéshez vezethet (gondoljunk a hosszabb gyártási idő miatti veszteségekre). Ne felejtsük azonban, hogy a véletlenszerűség jelenléte estén, a változékony termékáram kiegyenlítése érdekében, az ésszerűen megválasztott méretű pufferekre szükség van. Az 5.6 részben a pufferek elhelyezésének hasznos hatásait elemezzük, amikor a rendszerben meghibásodások és az ezzel járó javítások miatti időveszteségek fordulnak elő.

5.5. Gyártósorok modellezése indexelt sorokkal és erőforrásokkal (5.2. modell)

Amikor a modellek ismétlődő logikájú komponenseket tartalmaz, akkor a modell építés unalmassá válhat, következésképpen hajlamossá válunk a hibázásra. Az 5.3. pont alatt ismertetett csomagolósor esetében a részfolyamatok logikája analóg, csak a paraméterek különböznek. Pontosabban minden egyes részfolyamatban (töltés, lezárás, címkézés, krimpelés és csomagolás) lekötünk egy erőforrást, elvégezzük az aktuális műveletet, ellenőrizzük a blokkolást, és végül felszabadítjuk az erőforrást. Az ilyen ismétlődő logika szerint működő modellekhez az Arena egy hatékony eljárást kínál, amit indexelésnek nevezünk, és amely nagyban segít az ismétlődések eliminálásában. Ez a közelítés azt a tényt használja ki, hogy minden Arena objektumhoz tartozik egy egyedi, egészszámú azonosító (*ID* number) az objektum osztályon belül. Ilyen objektum osztályokat alkotnak a korábban megismert *queues* (sorok), *resources* (erőforrások) és *expressions* (kifejezések). Gyakorlatilag egy osztályon belül az objektumokat objektum vektorok írják le, azaz az egyedi objektumokhoz indexet rendelünk, amelyek segítik az elérésüket és kezelésüket. Az indexek egészszámok, 1-től a vektor méretéig (a létrehozott objektumok számáig) változnak, és meghatározzák az objektum sorszámát a vektorban. Megjegyezzük a vektor mérete dinamikus változó, és a modellező az indexelés

segítségével meghatározhatja az objektum helyét a vektorban. Az indexeléshez a modellező az **Elements** template panelről használhatja a **Queues**, **Resources** és **Expressions** elemeket. Ehhez természetesen az előzőleg az **Elements** template panelt elérhetővé kell tenni, amihez használhatjuk a *Template Attach* gombot a *Standard* eszköztárról.

A modellezésben az ismétlődő komponensek indexelésének bemutatására módosítjuk az 5.3. pontban megalkotott csomagolósor modellünket. Az Arena indexelési tulajdonságainak kihasználásához először sorszámot adunk az erőforrásoknak 1-5-ig, a soroknak 1-5-ig, és a blokkolt soroknak 6-10-ig, majd az **Elements** template panelről a **Resources** és **Queues** elemeket használva deklaráljuk ezeket az objektumokat.



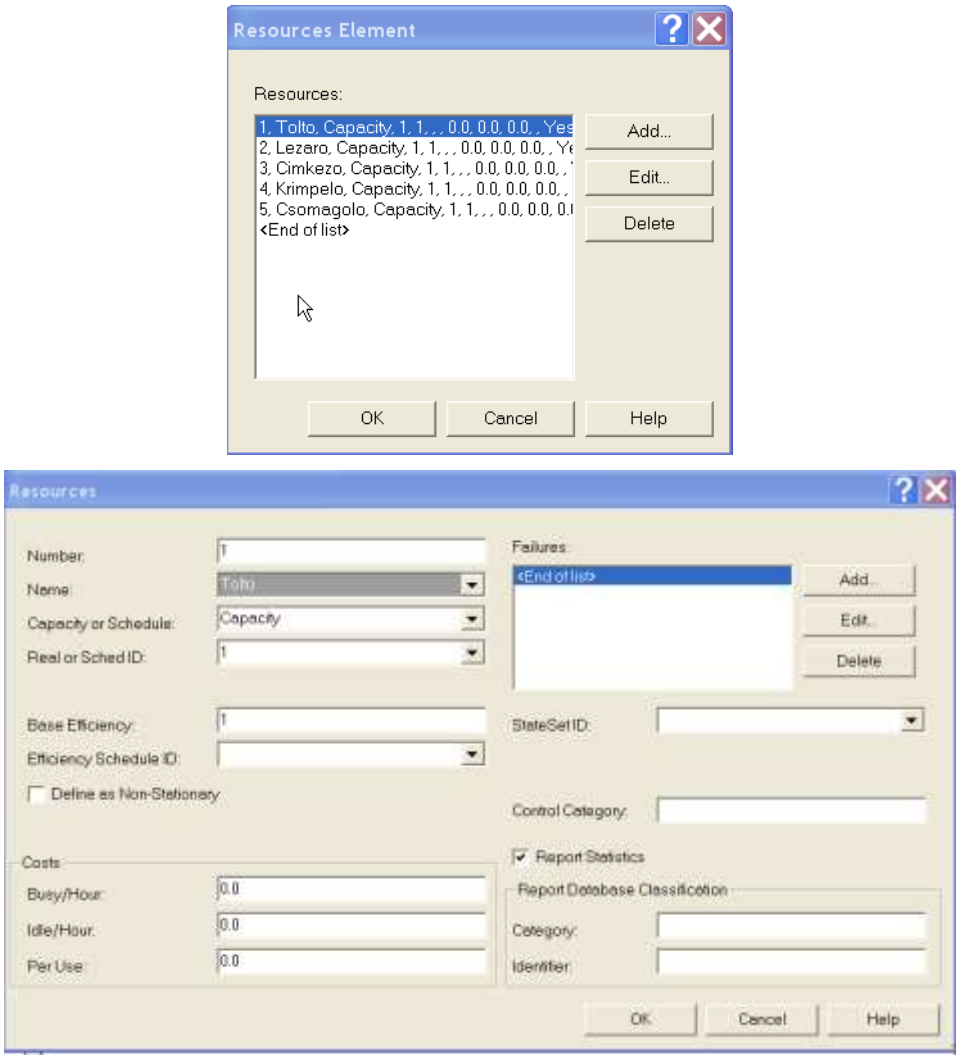
5.14. ábra: A **Queues Element** (balra) és a **Queues** (jobbra) dialógusablakai

Az *5.14. ábra* baloldalán a **Queues Element** dialógusablaka, jobboldalán pedig „*Tolto. Queue*” nevű sorhoz kapcsolódó dialógus. Kezdetben értelemszerűen a **Queues Element** ablak üres. Az *Add* gombra kattintva megnyílik a jobboldali dialógus, amelyben definiálhatjuk az aktuális sor attribútumait, például a Name mezőben a sor nevét: „*Tolto. Queue*”. Az ablakban a legfontosabb mezők a sor indexe, és sorbanállási elméletben fontos szerepet játszó prioritási szabály, az előbbit a Number mezőben lehet megadni, az utóbbit pedig a Ranking Criterion mező lenyíló listájából lehet kiválasztani (a példában FIFO). Ezenkívül, a modellező a Associated Block or SHARED Keyword mezőben két opció közül választhat. (1) megadhatunk egy SIMAN block-ot, amelyhez az aktuális sor társul, (2) enter the keyword shared, thereby signaling that the queue will be used in multiple **Hold** and **Seize** modules.

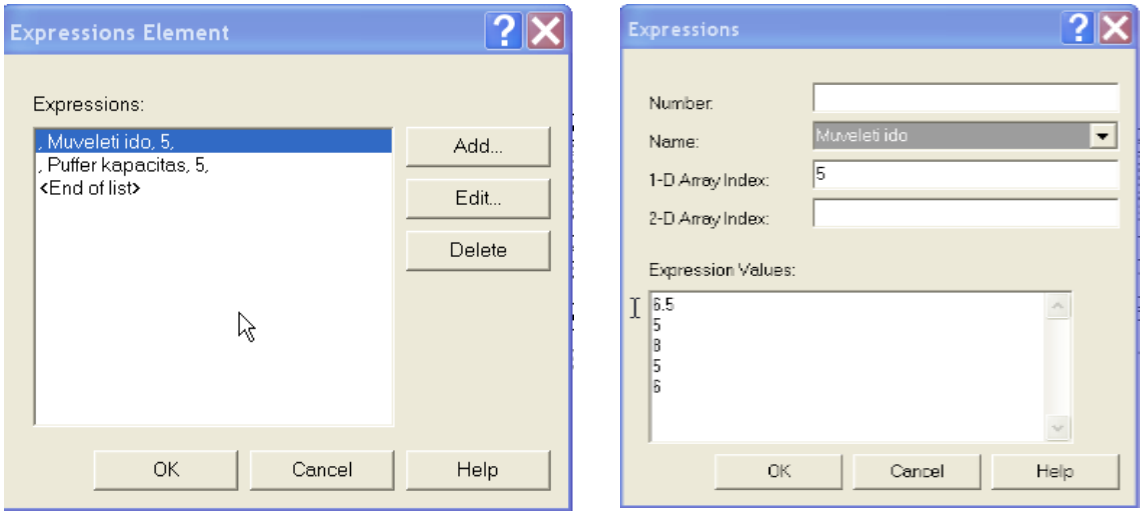
A felhasználó által definiált index hozzárendelése az erőforrásokhoz hasonlóan történik. Az *5.15. ábrán* a **Resources Element** dialógusablak (fent) és a társuló **Resource** dialógus (lent) látható. Vegyük észre, hogy a **Resource** dialógusban egy erőforráshoz (az ábrán a „*Tolto*”-höz) tartozó specifikus információk adhatók meg. Ezek a Number mezőben az index, és a Name mezőben az erőforrás neve, a Capacity or Schedule mezőben az időben állandó és az ütemezett kapacitás közül választhatunk. Az *5.15. ábrán* csupán az indexet rendeltük és a „*Tolto*” nevű erőforráshoz.

Végül csomagolósor Arena kifejezéseinek indexelését az *5.16. ábra* mutatja, amelyen a **Expressions Element** dialógusablaka (balra) és a társított **Expressions** dialógusablaka (jobbra) tanulmányozható. Egy **Expression** lehet vektor (egydimenziós), vagy mátrix (kétdimenziós). A vektor definiálásához az 1-D Array Index mezőben megadjuk az elemek számát (a vektor méretét). Hasonlóan definiáljuk a mátrixokat, csak ekkor az 1-D Array Index mezőben a mátrix sorainak és a 2-D Array Index mezőben a mátrix oszlopainak a számát adjuk meg. Az

aktuális kifejezésvektor elemeit, az *Add* gombot használva, egymásután soronként írjuk be a dialógusablakba. A jobboldali ablakban megadott Expression Values (kifejezés értékek), konstans számok, amelyek a műveleti időket és a puffer kapacitásokat (a munkaállomások között a puffer nagysága 5 termékegység) határozzák meg.



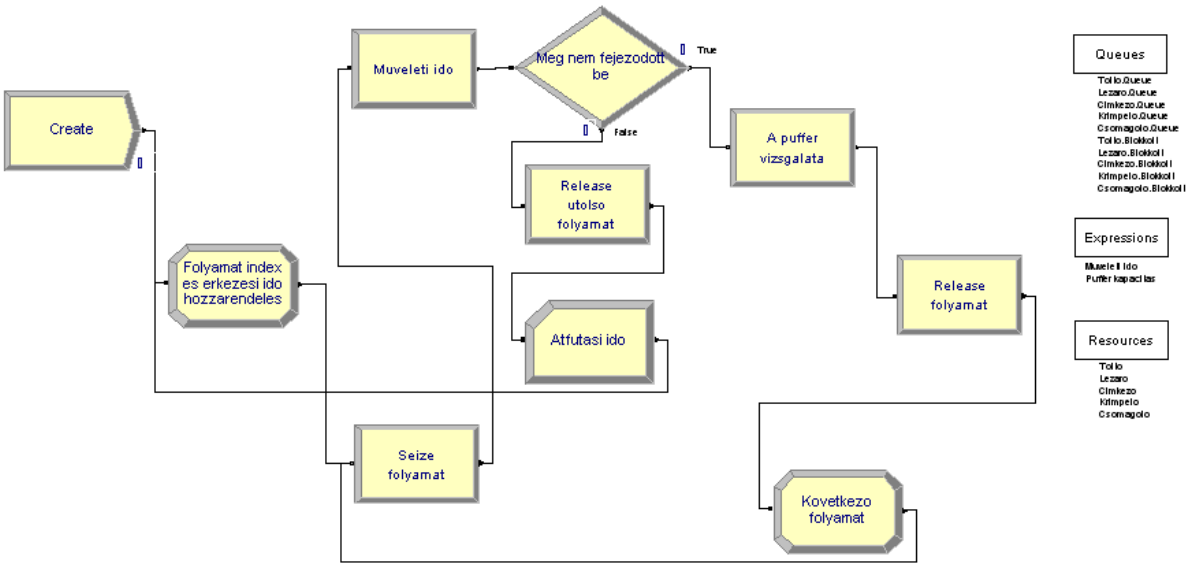
5.15. ábra: A **Resources Element** (fent) és a **Resources** (lent) dialógusablakai



5.16. ábra: Az **Expressions Element** (balra) és az **Expressions** (jobbra) dialógusablakai

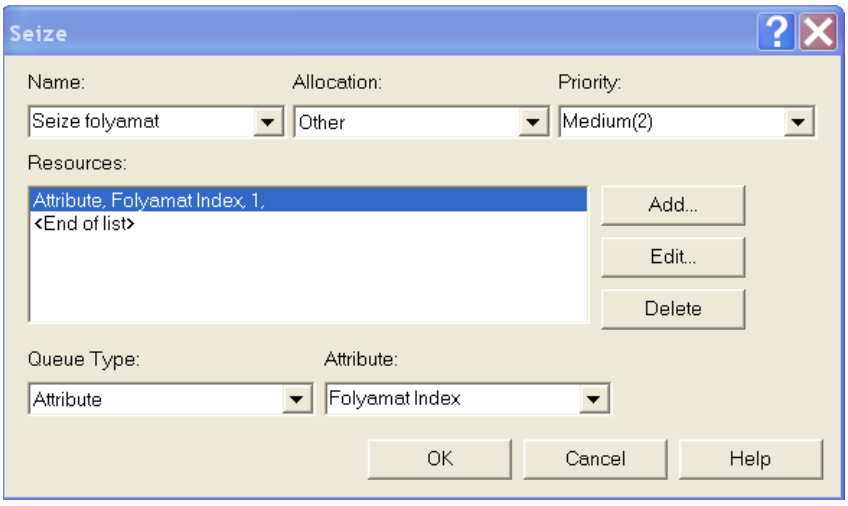
Az *5.17. ábrán* látható, indexeket használó, módosított modell ekvivalens az 5.3. alfejezetben megismert Arena modellel. Az új modell indexelt sorai, kifejezései és erőforrásai az ábra jobb oldalán jelennek meg. A modell logika a *Create* nevű **Create** modullal kezdődik, amely teljesen azonosan működik, mint az *5.2. ábra Create1* modulja, azaz a 0 időpontban, 30 entitást hozunk létre, amelyek a modellben cirkulálnak. Az entitások a termékegységek. Azonban a termékegységek mozgásának a logikája a csomagolósor munkaállomásain keresztül a programozási nyelvekből ismert Do…Loop típusú hurokkal analóg módon valósul meg. Esetünkben a „*Folyamat Index*” nevű attribútum értékét, mint futóindexet változtatjuk 1-től 5-ig, és figyeljük az értékét a hurokban.

Miután az termékegység entitás elhagyja a **Create** modult belép a „*Folyamat index es erkezesi ido hozzarendeles*” nevű **Assign** modulba, ahol hozzárendeljük a „*Folyamat Index*” nevű attribútum kezdeti értékét, 1-et. Ezzel egyidejűleg a pillanatnyi szimulációs időt (*TNOW*) tároló „*Erk_ Ido*” nevű attribútumot is az entitáshoz rendeljük, amit később az utolsó munkaállomás elhagyása után a teljes átfutási idő méréséhez fogunk felhasználni. Vegyük észre, hogy a módosított csomagolósor modellben az erőforrásokat (töltő, lezáró, címkéző, krimpelő és csomagoló) nem a nevükkel, hanem az indexeikkel 1,2,3,4 és 5 azonosítjuk, ami megfelel az *5.17. ábrán* a **Resources Element** alatti sorrendnek. Az erőforrásokhoz tartozó sorokat hasonlóan, a **Queues Element** alatti listának megfelelően, indexeljük. Minden termékegység entitás ugyanabban a sorrendben mozog a modulok között, miközben a „*Folyamat Index*” nevű attribútum értéke eggyel növekszik minden iterációban, így a felkeresett munkaállomás az indexnek megfelelő, következő munkaállomás lesz. Amint látni fogjuk, a munkaállomásokon elvégzett műveletek is az attribútum alapján lesznek kiválasztva.

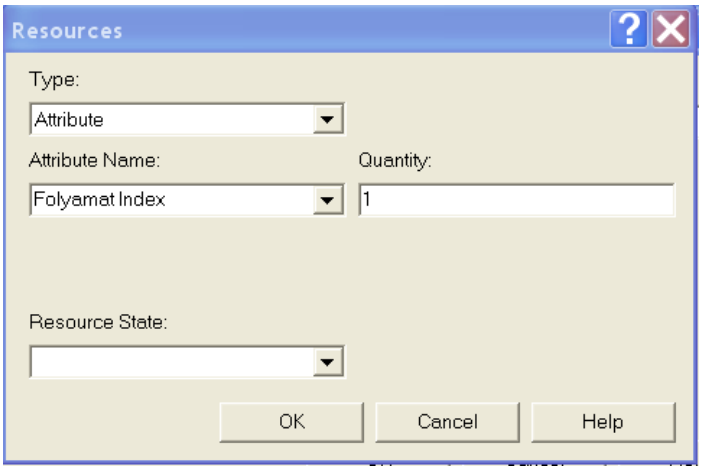


5.17. ábra: A csomagolósor indexelt Arena modellje

A hurok első modulja a „*Seize folyamat*” nevű **Seize** modul. A termékegység entitás a hurokba itt lép be. A modul dialógusablaka az *5.18. ábrán* látható. Ebben a Resources mező jelzi, hogy a pillanatnyilag lekötött erőforrás a pillanatnyi index értékével meghatározott, amit a „*Folyamat Index*” nevű attribútumban tárolunk. Ennek a kezdeti értéke 1, ami megfelel a töltő erőforrásnak. A pillanatnyi erőforrást definiáló dialógusablak képe az *5.19. ábrán* jelenik meg. Vegyük észre, hogy megfelelő erőforrás attribútum alapján (a Type mezőben választott opció) lesz azonosítva, és az Attribute Name mezőben adjuk meg „*Folyamat Index*” nevű attribútumot, mint azonosítót. A Quantity mezőbe 1-et írunk.

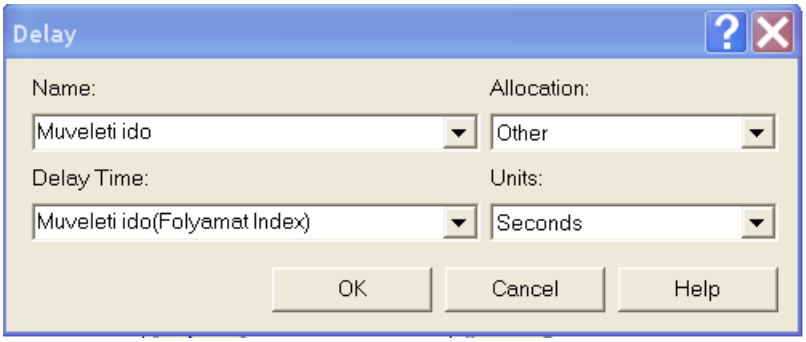


5.18. ábra: A hurok első moduljának (**Seize**) a dialógusablaka



5.19. ábra: A pillanatnyi erőforrás dialógusablaka a hurokban

Az erőforrás lekötése után a termékegység entitás belép a hurok második moduljába, a „*Muveleti ido*” nevű **Delay** modulba. A modul párbeszédablakát az *5.20. ábra* mutatja. Ebben a Delay Time mezőben adjuk meg a pillanatnyi művelet elvégzéséhez szükséges időtartamot, amit a „*Muveleti ido*(*Folyamat Index*)” nevű vektor, pontosabban a „*Folyamat Index*” attribútum által azonosított eleme határoz meg. A vektor elemeinek az értékei az *5.16. ábra* dialógusablakában láthatók.



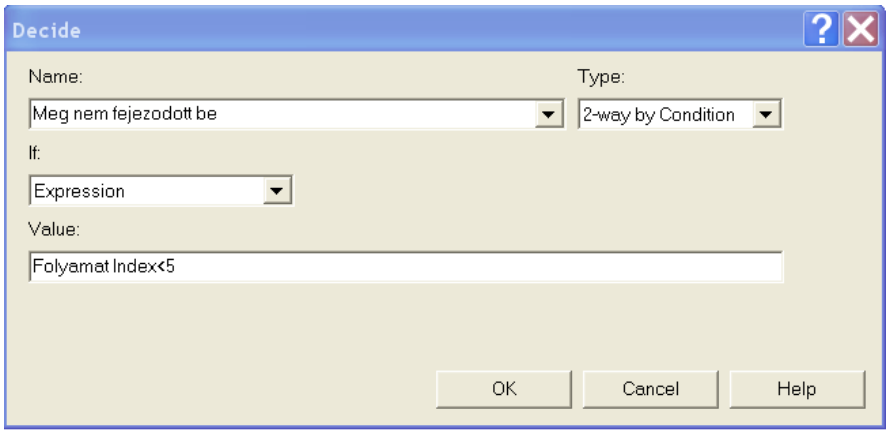
5.20. ábra: A hurok második moduljának (**Delay**) dialógusablaka

Amikor a termékegység entitás késleltetése befejeződik a „*Muveleti ido*” nevű **Delay** modulban, az azt jelenti, hogy a pillanatnyi munkaállomáson befejeződött a művelet. Ezt követően döntést kell meghozni:

Ha a munkaállomás nem az utolsó a műveleti sorrendben, akkor a hurok következő munkaállomásán folytatódik a folyamat. Mivel az utolsó részfolyamat, a csomagolás indexszáma 5, azt ellenőrizzük, hogy a „*Folyamat Index*” értéke kisebb-e 5-nél, azaz „*Folyamat Index*”<5.

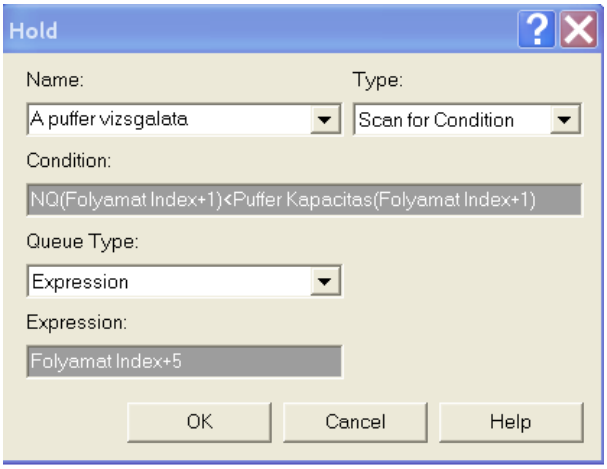
Különben, vagyis ha a „*Folyamat Index*”= 5, az átfutási idő mérésével (az öt munkaállomáson való átfutás ideje) és a ciklus újraindításával folytatódik a folyamat.

A döntést a hurok harmadik moduljában, a „*Meg nem fejezodott be*” nevű **Decide** modulban hozzuk meg (*5.21. ábra*). A modul dialógusablakában a Type mezőben a *2-way by Condition* opciót választjuk. A feltételt kifejezésként definiáljuk, ezért az If mezőben az Expression, és a Value mezőben a „*Folyamat Index*”< 5 látható. Ha a kifejezés értéke False („*Folyamat Index*”== 5), akkor a termékegység entitás feldolgozás befejeződött, és már csak néhány befejező adminisztrációt kell elvégezni.



5.21. ábra: A hurok harmadik moduljának (**Decide**) dialógusablaka

Történetesen, a termékegység entitás a „*Release utolso folyamat*” nevű **Release** modulba kerül, ahol felszabadítjuk az utolsó munkaállomás erőforrását, majd belép a „*Atfutasi ido*” nevű **Record** modulba, amelyben a termékegységek átfutási időit gyűjtjük. Végül a termékegység entitást a „*Folyamat index es erkezesi ido hozzarendeles*” nevű **Assign** modulba küldjük, és újraindítjuk az öt munkaállomást érintő ciklust.



5.22. ábra: A hurok negyedik moduljának (**Hold**) dialógusablaka

Ha azonban a „*Folyamat Index*”<5 feltétel teljesül, akkor a termékegység entitást a hurok negyedik moduljába küldjük, nevezetesen a „*A puffer vizsgalata*” nevű **Hold** modulba, amelynek a dialógusablakát az 5.22. ábra mutatja. Ebben a modulban ellenőrizzük, hogy a

következő munkaállomás (az indexe *Folyamat Index*+1) puffere telített-e. A Condition mezőben az ellenőrzésre szolgáló kifejezés:

NQ(*Folyamat Index*+1)<*Puffer Kapacitas*(*Folyamat Index*+1)

A "*Puffer Kapacitas*" egy vektor kifejezés, amelyben a pufferek kapacitásait tároljuk, és amelyet korábban definiáltunk (*5.16. ábra*). Figyeljünk arra, hogy a blokkolt sorok indexei kifejezések, amit a Queue Type mezőben az *Expression* választás jelez, és amelynek az értéke az Expression mezőben *Folyamat Index*+5, mivel az első blokkolt sor indexe 6 (*5.14. ábra*). Így a **Hold** modul az utolsó részfolyamat kivételével mindenhol használt, és feladata a termékegység entitások blokkolása addig, amíg a következő munkaállomás pufferében az entitások száma 5.

Mihelyt a következő munkaállomás pufferében van üres hely, vagyis a következő munkaállomás nem blokkolt, a termékegység entitás tovább mozog a hurok ötödik moduljába, nevezetesen a „*Release folyamat*" nevű **Release** modulba azért, hogy felszabadítsa a pillanatnyilag lekötött erőforrást. A termékegység entitás ezt követően belép a hurok utolsó, a hatodik moduljába, a „*Kovetkezo folyamat*" nevű **Assign** modulba, amelyben a „*Folyamat Index*" attribútum értékét növeljük 1-el, amivel a ciklusban a következő munkaállomásra ugrunk. Végül a ciklus végén visszatérünk az első modulhoz, a „Seize folyamat" folyamat nevű **Seize** modulhoz, és elkezdődik az iteráció következő ciklusa.

Összehasonlítva az ekvivalens csomagolósor modellek (*5.2.* és *5.17. ábrák*) futási eredményeit, ahogyan az várható azonos statisztikákat produkálnak.

5.6. Alternatív módszer a blokkolás modellezésére (5.3. modell)

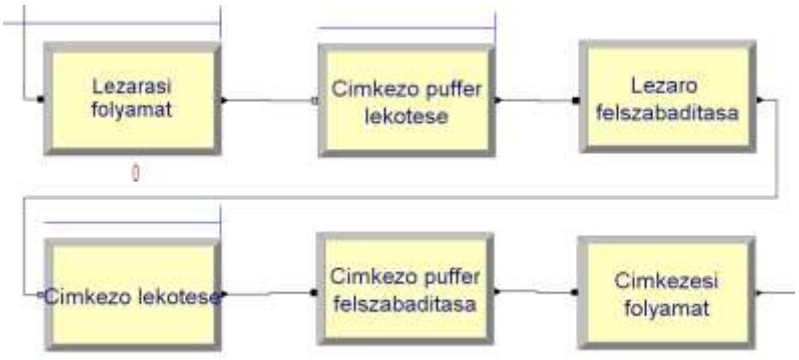
A blokkolás modellezésnek az Arena-ban több megoldása létezik. Az 5.3. pont alatt bemutatott csomagolósor példában a blokkolásra az **Advanced Process** panelről a **Hold** modult használtuk. Egy alternatív útja a blokkolásnak, hogy puffereket erőforrásként modellezzük. A megoldást az *5.23.* és az *5.24. ábrák* szemléltetik.

Resource - Basic Process

	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures
1	Cimkezo puffer	Fixed Capacity	5	0.0	0.0	0.0		0 rows

5.23. ábra: A **Resource** adatmodul dialógusablaka az alternatív blokkolásnál

A címkéző munkaállomáson az *5.23. ábrán* látható „*Cimkezo puffer*" nevű erőforrás szabályozza a „*Cimkezo lekotese*" nevű **Seize** modulhoz tartozó (*5.24. ábra*), „*Cimkezo lekotese.Queue*" nevű sor hosszát. E sor fizikailag megfelel a címkéző munkaállomás pufferének. A **Resource** adatmodulban a Capacity mezőt 5-re állítjuk, ami a címkézési folyamat puffer kapacitásával egyenlő.



5.24. ábra: A blokkolás alternatív modellezése **Resource** modulokkal

A cél, hogy amikor a címkézési folyamathoz tartozó sor (puffer) telített, akkor megelőző folyamat, a lezárás erőforrását ne lehessen felszabadítani. A modellben ezt úgy érjük el, hogy még a „*Lezaro*” nevű erőforrás felszabadítása előtt, a „*Lezarasi folyamat*”-ból kilépő termékegység entitások a „*Cimkezo puffer lekotese*” nevű **Seize** modulban megpróbálják elérni a „*Cimkezo puffer*” nevű erőforrás egységnyi kapacitását,. Ha „*Cimkezo puffer*” nevű erőforrásból nincs szabadkapacitás, akkor a „*Lezaro*” nevű erőforrás blokkolódik, ezáltal feltartja a termékegység entitást addig, amíg a „*Cimkezo puffer*” egységnyi kapacitása elérhetővé nem lesz. Mihelyt ez megtörténik, a termékegység entitás felszabadítja a „*Lezaro*” nevű erőforrást és az entitás tovább halad a címkéző munkaállomáshoz, ahol azonnal belép a „*Cimkezo lekotese*” nevű **Seize** modulba, pontosabban a „*Cimkezo lekotese.Queue*” nevű sorba, és itt a „*Cimkezo*” nevű erőforrás elérésére várakozik. A várakozás alatt a termékegység entitás a „*Cimkezo puffer*” erőforrás egységnyi kapacitását köti le, amit akkor szabadít fel, amikor számára a „*Cimkezo*” nevű erőforrás elérhetővé válik. Mivel a „*Cimkezo puffer*” kapacitása 5 egység, a „*Cimkezo lekotese.Queue*” nevű sorban, fizikailag a címkéző munkaállomás pufferében maximálisan 5 termékegység entitás várakozhat.

Megjegyezzük, a blokkolásnak ez a közelítése költségesebb, mint a korábban az 5.3. pont alatt megismert megoldás, amelyben egy folyamat blokkolására csak egy **Hold** modult használtunk. Az alternatív megoldásban egy folyamathoz három modulra (**Seize**, **Release** és **Resource**) volt szükségünk, ami nemcsak bonyolultabbá teszi, hanem növeli is a modell méretét. Az utóbbi problémát is okozhat, például az Arena student verziója korlátozza a modulok számát a modellben.

5.7. Gép meghibásodások modellezése (5.4. modell)

A gyártási és a szolgáltatási rendszerekben a folyamatok véletlenszerű leállása szinte elkerülhetetlen. Ezek a jelenségek általában a gépek meghibásodásával állnak összefüggésben. A rendszerek meghibásodása a felveti a megbízhatóság kérdését, amely annak valószínűsége, hogy valamely gép vagy rendszer a várható üzemeltetési feltételek mellett, a tervezett üzemidő alatt mennyire megbízhatóan fog működni. Lényegében a megbízhatóság a hibamentesség, a tartósság és a javíthatóság fogalmainak összessége. A megbízhatóság növelése nemcsak az üzemeltetők, hanem gyártók elemi érdeke is. Ezért a megbízhatóság javításában a tervezőktől, a gyártókon keresztül, az üzemeltetőkig mindenkinek vannak feladatai. Ezek a teljességigénye nélkül: a tervezésnél a kritikus helyek túlméretezése, tartalékelemek készenlétben tartása (redundancia), a fellépő zavarok okainak rendszeres vizsgálata, az elemek megbízhatóságának fokozása, szigorú minőség-ellenőrzés, statisztikai módszerek alkalmazása, stb.

Az üzemeltetők értelemszerűen abban érdekeltek, hogy a leállások lehetőleg hosszú hibamentes periódusok után jelentkezzenek. A leállási idők ugyanis, amit zavaridőnek is neveznek, veszteséget okoznak, mivel a csökkentik az effektív termelési időt. A veszteségidők hatékony hibaelhárítással minimalizálhatók. A szimuláció maga csak a hibák rendszerre gyakorolt hatásainak vizsgálatában és megértésében segít.

A rendszerhibák időbeli jelentkezései sztohasztikus folyamatok, és valamely időszakok zavarmentes működésének valószínűsége a meghibásodási ráta:

λ=Megfigyelt zavarok száma/Megfigyelés időtartama,

amelyből a két zavar közötti időtartam ($1/\lambda$). A megbízhatóság, különböző eloszlásfüggvényekkel leírható valószínűségi változó, ami a meghibásodási ráta és az üzemidő függvénye. Például exponenciális eloszlás esetén a megbízhatóság:

$$\rho(t) = \exp(-\lambda \cdot t).$$

Ebből adódóan a meghibásodásokat érkezési folyamatokhoz hasonlóan lehet modellezni.

Egy önálló munkaállomáson a meghibásodás forgatókönyve a következő. A normál működési periódus (*uptime*) után hiba jelentkezik, ami a munkaállomást leállásra kényszeríti, és megkezdődik a hiba elhárítása (kijavítása), amelynek az időtartama a *downtime*. (Néha a javításhoz kötődő járulékos időket, pl. előkészítés, önállóan kell modellezni, de általában a javítási idő ezeket is magában foglalja.). Ha akkor jelentkezik a hiba, amikor a gép működik, akkor *működésfüggőnek* nevezzük. Azonban az újfajta, számítógéppel vezérelt gépeknél a meghibásodás bármikor előfordulhat. Az ilyen hibákat *működésfüggetlen* hibáknak nevezzük, amelyek működési és indítási hibákat, a takarítást, a beállítást, valamint egyéb specifikus időkiesést okozó tevékenységeket is magukban foglalnak. Az eddig megismert, két alapvető gépállapotot (*Idle* és *Busy*) ki kell egészíteni, a hiba - és a leállási állapotokkal, mint például a *Down, Cleaning, Adjustment,* stb.

Az Arena az *auto-states*-nek nevezett beépített állapotokon kívül megengedi a felhasználó által definiált állapotok alkalmazását is. Pontosabban az Arena erőforrások a négy automatikus állapota: *Idle, Busy, Failed* és *Inactive*, és ezek közül az erőforrás az idő bármely pillanatában csak az egyikben lehet. Az átmenetet az egyik állapotból a másikba valamilyen esemény váltja ki. Például, amikor hibaesemény történik, akkor a gép automatikusan *Failed* állapotba kerül. Ugyanakkor a felhasználó által definiált átmeneteket a modellező tetszés szerint programozhatja. Általában ezen állapotok long-run valószínűségében vagyunk érdekeltek, és ilyen valószínűségek a **Statistic** adatmodul *Frequency* opciójával becsülhetők.

A modellező az erőforrás kapacitását az idő függvényében a **Schedule** adatmodulban (lásd az 5.8 szakaszban részletesen) tudja változtatni, időben változó kapacitás szint definiálásával. Például tervszerű megelőző karbantartás esetén modellezett kényszerített leállással (downtimes) az erőforrás kapacitását 0-ra csökkentjük, azaz inaktív állapotba helyezzük. Ekkor az erőforrás *Inactive* állapotának valószínűsége nem 0.

Tegyük fel, hogy a csomagolósor modellben a töltő munkaállomás időnként meghibásodik, és ezenkívül minden 250 palack távozása után beállítást igényel. Feltételezzük, hogy a hibamentes idő (uptimes), azaz a javítás vége és a következő meghibásodás között eltelt idő exponenciális eloszlású, 50 óra középértékkel, ugyanakkor a javítási idő egyenletes eloszlású, 1,5 és 3 óra között. Az említett beállítás időtartama is egyenletes eloszlású, 10 és 25 perc között. A csomagoló munkaállomáson is gyakoriak a mechanikus meghibásodások, a leállási idő (downtimes) trianguláris eloszlású, minimum 75 perc, maximum 2 óra és 90 perc legvalószínűbb értékkel. Az ehhez tartozó hibamentes idő exponenciális eloszlású, 25 óra középértékkel. Végül azt feltételezzük, hogy a hibák a gépek működése alatt következnek be (*működésfüggő hibák*). A módosított csomagolósor modellre (5.4. modell), mint meghibásodásokkal jellemzett csomagolósor modellre hivatkozunk.

Failure - Advanced Process

	Name	Type	Up Time	Up Time Units	Count	Down Time	Down Time Units	Uptime in this State
1	Tolto hiba	Time	EXPO(50)	Hours	HoursToBaseTime(EXPO(50))	UNIF(1.5 , 3)	Hours	Busy
2	Csomagolo hiba	Time	EXPO(25)	Hours	HoursToBaseTime(EXPO(25))	TRIA(75 , 90 , 120)	Minutes	Busy
3	Karbantartas	Count	1.0	Hours	250	UNIF(10 , 25)	Minutes	

5.25. ábra: A **Failure** adatmodul dialógustáblázata

A meghibásodásokkal jellemzett csomagolósor modellhez tartozó **Failure** adatmodult az *5.25. ábra* szemlélteti. A Name mezőben adjuk meg a hibák neveit, a Type mezőben pedig a hiba típusát választhatjuk ki, amely lehet *Time* (időalapú hibaérkezés) és *Count* (számláláson alapuló hibaérkezés) típusú. Például, véletlenszerű mechanikus hibák normálisan időalapúak,

mivel a hibák jelentkezései idő specifikusak. Ezzel szemben a gépek tisztítási igényei bizonyos számú befejezett művelet után jelentkeznek, ezért a tisztítás miatti leállások számláláson alapulnak. Például az *5.25. ábrán* a „*Karbantartas*” nevű hiba/leállás *Count* típusú, azaz számláláson alapuló.

Az időalapú hibák sorában az Up Time oszlop határozza meg egy javítás után következő hiba megjelenéséig eltelt időt, az Up Time Units oszlop pedig az ehhez tartozó időegységet. Hasonlóan a számlált alapú hibák sorában a Count oszlopban adjuk meg következő hiba jelentkezéséig távozó entitások számát (a „*Karbantartas*” nevű hiba sorában ez 250). Vegyük észre, hogy a nem használt oszlopok (a hiba típusától függően) árnyékoltak.

A Down Time oszlop a leállás időtartamát definiálja, és a Down Time Units oszlop a leállás időegységét. Végül, az Uptime in this State oszlopot csak az időalapú hibáknál használjuk, annak meghatározására, hogy a hiba az erőforrás milyen állapotában következik be. Például az időalapú hibák az *5.25. ábrán* akkor történnek, amikor az adott erőforrás működik, vagyis *Busy* állapotban van.

Az Arena gondoskodik egy olyan mechanizmusról is, amely az erőforrások állapotait definiálja, és azokat hozzákapcsolja a hibákhoz. E mechanizmus működtetésére szolgál az Advanced Process panelen elérhető **StateSet** adatmodul. Az *5.26. ábra* szemlélteti a **StateSet** használatát a csomagolósor modellben. A baloldali dialógustáblázat (*5.26. ábra*) Name oszlopában az állapothalmaz nevét adjuk meg (célszerűen úgy, hogy utaljon a hiba nevére), a States oszlopban pedig az állapotok számát. Például az első sorban a „*Toltes allapot*” nevű állapothalmazhoz 4 állapot tartozik (*4 rows*). A States mezőre klikkelve megjelenik a States nevű dialógustábla (*5.26. ábrán* a jobboldalon) a részletes állapotinformációkkal. Itt a State Name nevű oszlopban a felhasználó által definiált vagy az *auto-state* állapotnevek jelennek meg, ugyanakkor az AutoState or Failure oszlop az ezekhez rendelt, felhasználó által definiált vagy *auto-state* hibaneveket tartalmazza.

StateSet - Advanced Process

	Name	States
1	Toltes allapot	4 rows
2	Lezaras allapot	2 rows
3	Cimkezes allapot	2 rows
4	Krimpeles allapot	2 rows
5	Csomagolas allapot	3 rows

States

	State Name	AutoState or Failure
1	Idle	Idle
2	Busy	Busy
3	Karbantartas	Karbantartas
4	Down	Tolto hiba

5.26. ábra: A **StateSet** modul (balra) és a töltő erőforrás állapotait mutató **States** modul (jobbra) dialógustáblái

Resource - Basic Process

	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1	Tolto	Fixed Capacity	1	0.0	0.0	0.0	Toltes allapot	2 rows	☑
2	Lezaro	Fixed Capacity	1	0.0	0.0	0.0	Lezaras allapot	0 rows	☑
3	Krimpelo	Fixed Capacity	1	0.0	0.0	0.0	Krimpeles allapot	0 rows	☑
4	Csomagolo	Fixed Capacity	1	0.0	0.0	0.0	Csomagolas allapot	1 rows	☑
5	Cimkezo	Fixed Capacity	1	0.0	0.0	0.0	Cimkezes allapot	0 rows	☑

Failures

	Failure Name	Failure Rule
1	Tolto hiba	Preempt
2	Karbantartas	Wait

5.27. ábra: Resource adatmodul (fent) és a **Failures** adatmodul dialógustáblái

Végül az *5.27. ábra* mutatja a meghibásodásokkal módosított modellben a **Resource** adatmodulban a „*Tolto*” és a „*Csomagolo*” erőforrások soraiban és a StateSet Name oszlopban a változásokat, illetve a hiba előfordulásakor az erőforrások viselkedését. A felső táblázat (*5.27. ábra*) első sorában a Name oszlopban a „*Tolto*” nevű erőforráshoz két hibatípus tartozik (a Failures oszlopban *2 rows*). Az egérrel a mezőre kattintva megjelenik a Failures nevű dialógustáblázat (*5.27. ábra* alul), amely a „*Tolto*” nevű erőforrás két hibáját „*Tolto hiba*” és „*Karbantartas*” mutatja a Failure Name oszlopban. A Failure Rule (hibaszabály) oszlop azt a hiba jelentkezésekor követendő tevékenységsorozatot határozza meg, ami az időalapú hiba előfordulásakor (a számláláson alapuló hibáknál nem alkalmazzuk) az éppen feldolgozás alatt álló entitással történik. (A 4. fejezetben foglalkoztunk az ütemezési szabályokkal (*Schedule Rule*), amelyek hatása sok tekintetben hasonló a hibaszabályok hatásához.) A tevékenységsorozat a választott opciótól függ:

A *Preempt* választásakor a hiba jelentkezése azonnal kezdeményezi a leállást, az opció az erőforrás működést haladéktalanul felfüggeszti.

A *Wait* opció megengedi a feldolgozás alatt álló entitás folyóműveletének a befejezését, csak ezután függeszti fel az erőforrás működését és kezdődik a leállás.

Az *Ignore* opció is akkor indítja a leállást, amikor a feldolgozás alatt álló entitás folyóművelete befejeződött. Azonban ez az opció a leállási időt (downtime), a tervezett leállási idő és az aktuális művelet befejezése között eltelt idővel csökkenti (eltérően a *Wait* opciótól, ahol a leállási idő egyenlő a tervezett leállási idővel).

A példában a „*Tolto hiba*”-nál a *Preempt* opciót, a „*Karbantartas*”-nál, mivel számláláson alapuló hiba, a *Wait* opciót alkalmazzuk, még akkor is, ha más opciót kellene választani (emlékezzünk arra, hogy számláláson alapuló hibáknál általában hibaszabályt nem alkalmazzuk).

13:31:07 **Frequencies** október 12, 2011

Csomagolósor erőforrás meghibásodásokkal Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 1 000 000,00 Time Units: Seconds

Cimkezo allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
Busy	322	2,064.03	66.46	66,46
Idle	323	1,038.33	33.54	33,54

Csomagolo allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
Busy	79,545	6.0155	47.85	47,85
Down	5	5,661.17	2.83	2,83
Idle	79,542	6.2004	49.32	49,32

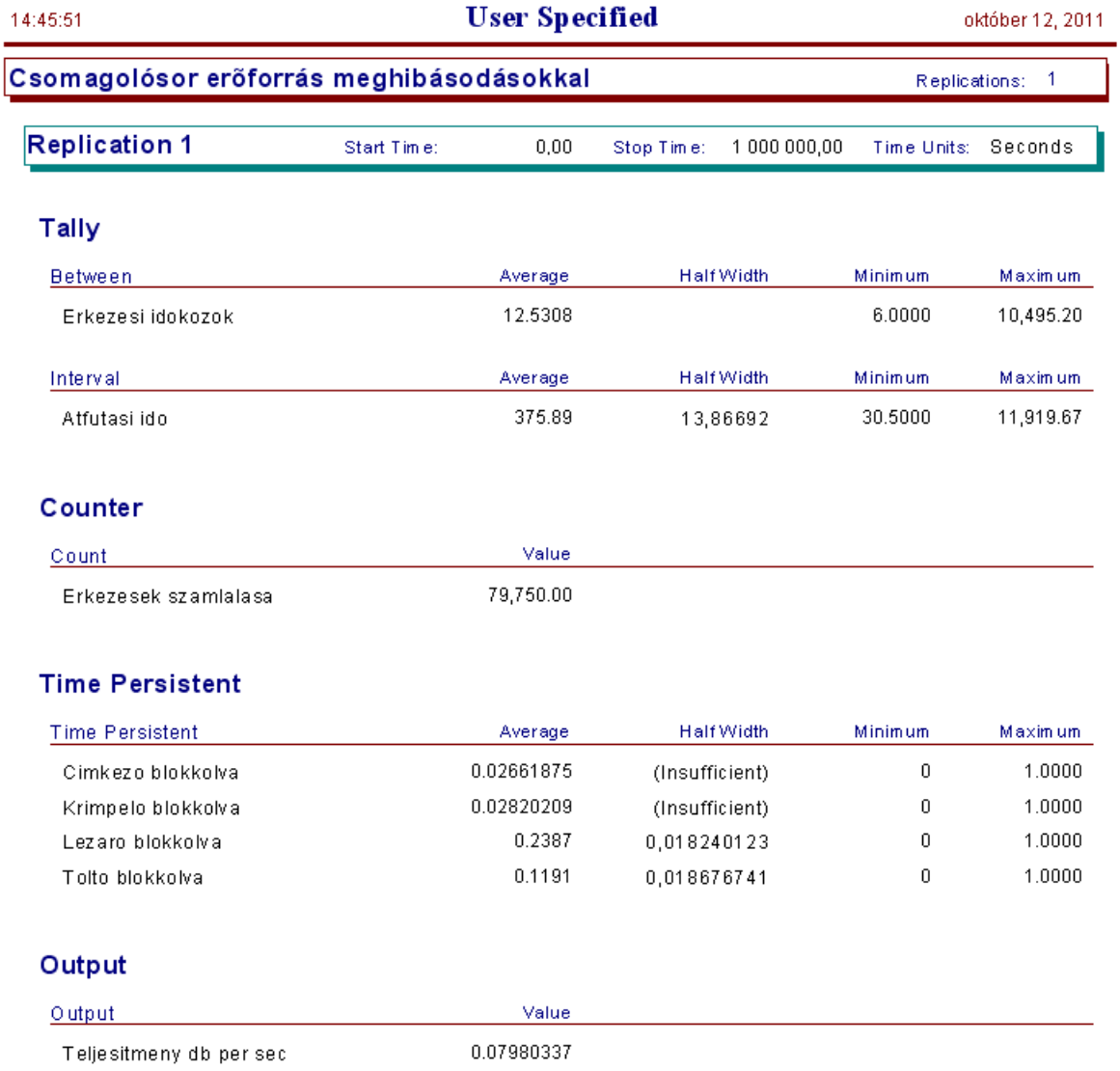
Krimpelo allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
Busy	79,645	5.3607	42.70	42,70
Idle	79,646	7.1949	57.30	57,30

Lezaro allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
Busy	9,309	68.4781	63.75	63,75
Idle	9,310	38.9407	36.25	36,25

Tolto allapotok	Number Obs	Average Time	Standard Percent	Restricted Percent
Busy	322	1,979.70	63.75	63,75
Down	3	7,294.03	2.19	2,19
Karbantartas	319	1,067.88	34.07	34,07

5.28. ábra: A meghibásodásokkal jellemzett csomagolósor modell *Frequencies* jelentése

Az erőforrások állapotvalószínűségeit a *Frequency* statisztika lekérdezésével érhetjük el. Ehhez a **Statistic** adatmodulban, a Type oszlopban a *Frequency* opciót, a Frequency Type oszlopban pedig a *State* opciót kell választani. Az *5.28. ábra* az eredményként kapott *Frequencies* riportot mutatja.



14:45:51 **User Specified** október 12, 2011

Csomagolósor erőforrás meghibásodásokkal Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 1 000 000,00 Time Units: Seconds

Tally

Between	Average	Half Width	Minimum	Maximum
Erkezesi idokozok	12.5308		6.0000	10,495.20

Interval	Average	Half Width	Minimum	Maximum
Atfutasi ido	375.89	13,86692	30.5000	11,919.67

Counter

Count	Value
Erkezesek szamlalasa	79,750.00

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Cimkezo blokkolva	0.02661875	(Insufficient)	0	1.0000
Krimpelo blokkolva	0.02820209	(Insufficient)	0	1.0000
Lezaro blokkolva	0.2387	0,018240123	0	1.0000
Tolto blokkolva	0.1191	0,018676741	0	1.0000

Output

Output	Value
Teljesitmeny db per sec	0.07980337

5.29. ábra: A meghibásodásokkal jellemzett csomagolósor modell felhasználó által definiált statisztikái

Hasznos lehet az eredeti és a meghibásodásokkal jellemzett csomagolósor modellek működésének az összehasonlítása. A meghibásodásokkal jellemzett csomagolósor modelltől azt várjuk el, hogy az eredetinél szerényebb teljesítményt mutasson. Valóban, az *5.28.* és az *5.12. ábrák* összehasonlítása feltárja, hogy a meghibásodásokkal jellemzett csomagolósor modellben a töltő munkaállomás után elhelyezkedő munkaállomások erőforrásainak működését a tétlenség feltűnő növekedése jellemzi. Ennek a kimenetnek a magyarázata egyszerű. Mivel a hibákkal módosított modellben a véletlen meghibásodások a „*Tolto*” erőforrást időnként leállásra kényszerítik, az kevesebbet termel, mint az eredeti modellben. Következésképpen, az áramlásirányú munkaállomáson gyakrabban jelentkezik hány, ami végül is növekvő tétlenséget okoz a rendszerben. Hasonló módon, az *5.29. ábrán,* a *User Specified* riportból kiolvasható eredmények elemzése is erre a megállapításra vezet.

Az *5.29.* és az *5.11. ábrák* összehasonlítása további evidensnek látszó megállapításokat tár

fel, például a módosított modell teljesítménye kisebb, mint az eredetié. Ennek is az, az oka, hogy a töltési folyamat áramlásirányban a tétlenség növekedését eredményezi a meghibásodásokkal jellemzett modellben, ami miatt az „*Erkezesi idokozok*" átlagosan 8 másodpercről 12,5 másodpercre növekszenek, és ezzel egyidejűleg a rendszer teljesítménye 0,125 db/s-ról 0,08 db/s-ra csökken. A példa kimerítően demonstrálja a géphibák és a leállások ártalmas hatását a rendszer teljesítményére. Az eredmények ökonómiai konzekvenciája, a rendszerteljesítmény szükségszerűen követi a változásokat.

A géphibák hatása a rendszerteljesítményre, és ezeknek a hatásoknak a redukálása a gépek közé helyezett pufferekkel széles körben tanulmányozott kutatási téma. A témáról további fejtegetés a következő forrásokban található: Buzacott és Shanthikumar (1993), Gershwin (1994), Altiok (1997), és Papadopoulos et al. (1993). A véges pufferekkel működő gyártóhálózatok elemzése Whitt (1983) munkájában érhető el.

5.8. Átfutási idők eloszlásának becslése (5.5. modell)

Az Arena-ba épített *Frequency statistics* (gyakorisági statisztika) leegyszerűsíti a sorokban vagy a pufferekben várakozó entitások számának, pontosabban azok eloszlásának a becslését. Azonban az átfutási idők (a rendszerben töltött összes idő) eloszlásának becslése egy kicsit több munkát igényel. Példaként feltételezzük, hogy az eredeti csomagolósor modellben szeretnénk megismerni a termékegységnek a csomagoló munkaállomás pufferében eltöltött várakozási idejét (T_V), illetve annak eloszlását. Pontosabban szeretnénk megbecsülni a következő várakozási idők valószínűségeit:

$$P\{T_V < 1\},\ P\{1 \le T_V < 3\},\ P\{3 \le T_V < 5\} \text{ és } P\{T_V \ge 5\}.$$

A várakozási idők valószínűségének becsléséhez szükséges modell átalakításokat az *5.30. ábra* mutatja. A termékegység entitás balról jobbra halad át a szegmens moduljain. Megérkezve a csomagoló munkaállomásra, az első **Assign** modulban a termékegység entitás „*Varakozas kezdete*" nevű attribútumában rögzítjük a pillanatnyi időt (*TNOW*). A termékegység entitás belép a „*Seize csomagolo*" nevű **Seize** modulba, és az állapotától függően, leköti a munkaállomás „*Csomagolo*" nevű erőforrását, vagy az erőforrásra várakozik a „*Seize csomagolo.Queue*" nevű sorban. Ezt követően az entitás belép a második **Assign** modulba („*Varakozas szamitasa*"), ahol a csomagoló pufferben (ez a „*Seize csomagolo.Queue*") töltött várakozási időt a következő kifejezéssel számítjuk (*5.30. ábra*):

$$Varakozasi\ ido = TNOW - Varakozas\ kezdete.$$

A számított időt a termékegység entitás "*Varakozasi ido"* nevű attribútumához rendeljük. A négy kategóriába sorolt várakozási idők számát az *N1, N3, N5* és *NM* változókban tároljuk. A várakozási idők számához hozzáadjuk a zárójelbe foglalt logikai kifejezések kimeneteit (*5.30. ábra*), amelyek 1 (ha *true*) vagy 0 (ha *false*) értékűek lehetnek. Például az

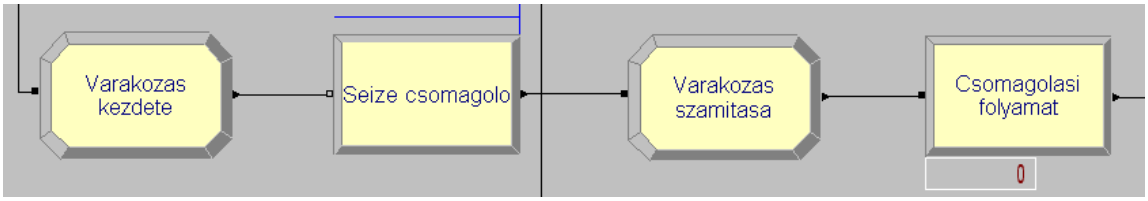
$$N1 = N1 + (Varakozasi\ ido < 1).$$

Így az *N1* változó azoknak a megfigyeléseknek a számát tartalmazza, amelyekben a várakozási idő szigorúan kisebb mint 1 s, az *N3* azokat, amelyekben a várakozási idő 1 és 3 s közötti intervallumba esik, *N5* azokat, amelyekben a várakozási idő 3 és 5 s közötti intervallumba esik, és *NM* azokat, amelyekben a várakozási idő nagyobb vagy egyenlő mint 5 s. Vegyük észre, hogy minden belépő termékegység entitás egy és csak is egy változó értékét növeli eggyel. Végül a „*Mefigyelesek szama*" nevű változó az összes megfigyelt várakozást tartalmazza.

Amikor a futó szimuláció leáll, az érdeklődésre számot tartó várakozási idők valószínűségét *Output* típusú statisztika számítja, amit a **Statistic** adatmodulban definiálunk. Az *5.31. ábra* szemlélteti az adatmodult táblázatnézetben, a négy kategóriába sorolt várakozási idők való-

színűségeinek a számítására alkalmas kifejezésekkel együtt. Például, azoknak a várakozási időknek a valószínűsége, amikor a várakozási idő 3 és 5 közötti intervallumba esik:

$$P\{3 \le T_V < 5\} = N5/\ Mefigyelesek\ szama.$$



Name	Varakozas kezdete
Assignments	
Type	Attribute
Attribute Name	Varakozas kezdete
New Value	TNOW

Name	Varakozas szamitasa
Assignments	
Type	Attribute
Attribute Name	Varakozasi ido
New Value	TNOW − Varakozas kezdete
Type	Variable
Variable Name	N1
New Value	N1 + (Varakozasi ido <1)
Type	Variable
Variable Name	N3
New Value	N3 +(Varakozasi ido >=1 .and. Varakozasi ido <3)
Type	Variable
Variable Name	N5
New Value	N5+(Varakozasi ido >=3 .and. Varakozasi ido <5)
Type	Variable
Variable Name	NM
New Value	NM+(Varakozasi ido >=5)
Type	Variable
Variable Name	Mefigyelesek szama
New Value	Mefigyelesek szama + 1

5.30. ábra: A csomagoló pufferben a várakozási idő valószínűségének becslésére szolgáló Arena szegmens

Statistic - Advanced Process

	Name	Type	Expression	Report Label	Output File
1	P Varakozasi ido LT 1	Output	N1/Mefigyelesek szama	P Varakozasi ido LT 1	
2	P Varakozas ido LT 3	Output	N3/Mefigyelesek szama	P Varakozas ido LT 3	
3	P Varakozas ido LT 5	Output	N5/Mefigyelesek szama	P Varakozas ido LT 5	
4	P Varakozas ido GE 5	Output	NM/Mefigyelesek szama	P Varakozas ido GE 5	

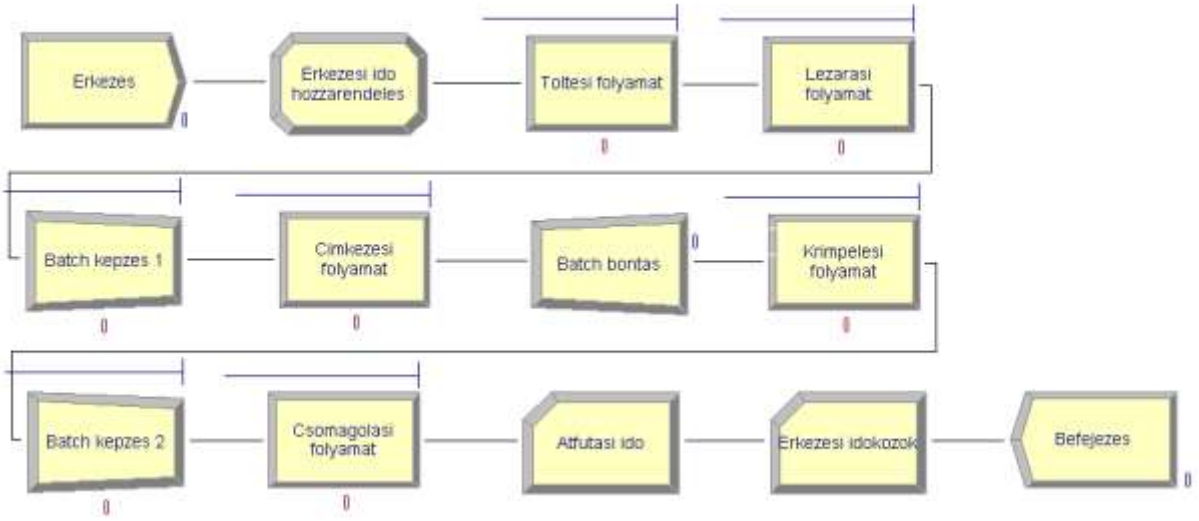
5.31. ábra: A **Statistic** adatmodul dialógustáblája *Output* statisztikával a csomagoló pufferben a várakozási idő valószínűségének a becslésére

Hasonló módszerrel becsülhetők más átfutási idő valószínűségek, ilyen például a gyártási idő (az első művelettől az utolsó művelet befejezéséig eltelt idő). Az átfutási idő becslésének az implementálását gyakorlásként az olvasóra bízzuk.

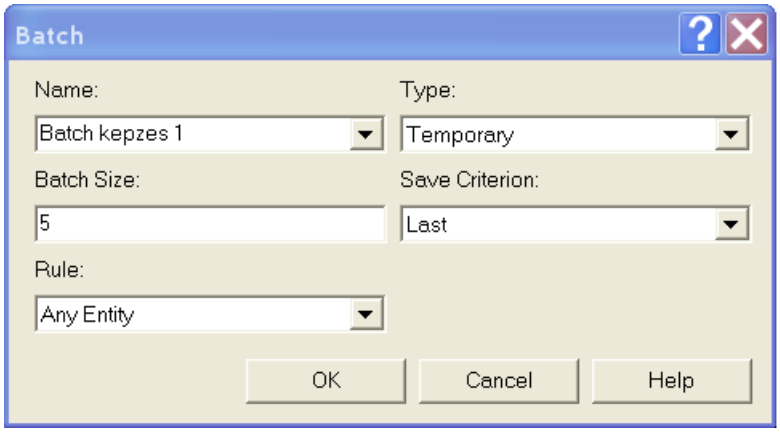
5.9 Batch feldolgozás (5.6. modell).

Tipikus gyártási környezetekben gyakori, hogy egy folyamat vagy gép munkadarabok csoportját dolgozza fel, illetve munkálja meg, azaz egyidejűleg több munkadarab megmunkálását végezi. A csoport mérete általában előre adott állandó szám, de lehet véletlenszerű is. Sok esetben a munkadarabok az összeszerelés eredményeként alkotnak állandó munkadarab csoportot, és az így összeszerelt egység gyártási folyamata folytatódik. Néhány esetben ennek ellentéte történik, és a munkadarabok csoportját egyedi munkadarabokra bontjuk. A gyártási szakzsargonban a munkadarab csoportokat **batch**-eknek nevezik. Például a festés, a hőkezelés, csomagolás, az autó és elektronikai iparban az összeszerelés, a vegyiparban és a gyógyszergyártásban a sterilizáció, a rázás, a centrifugálás sokszor batch-ben történik, de lehetne számos más műveletet is említeni. A csoportos megmunkálás modellezését az Arena-ban a **Batch** és **Separate** modulok teszik lehetővé. A **Batch** modul mind a permanens (állandó), mind a temporary (ideiglenes) batch-ek létrehozását támogatja. A **Separate** modul a temporary batchek felbontására alkalmas, az eredeti alkotóelemek visszaállítására. A permanens batcheket nem lehet felbontani.

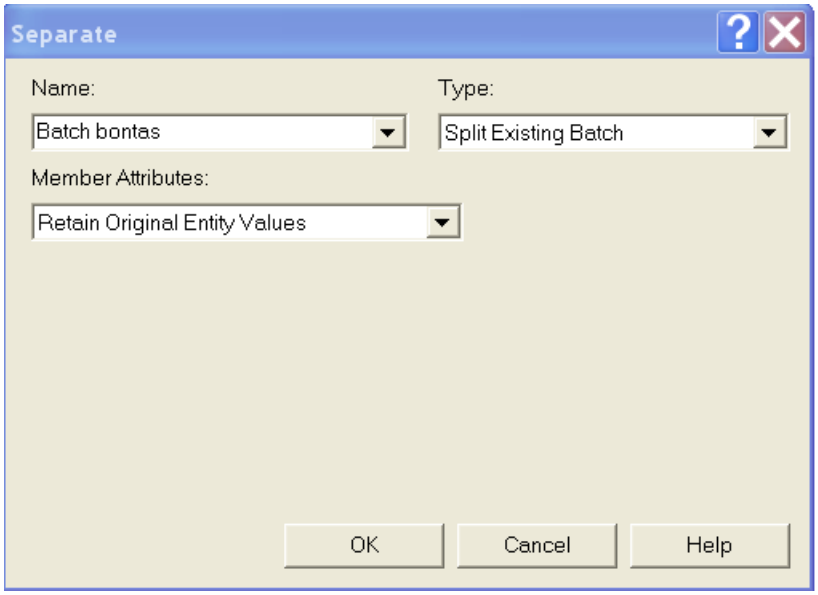
A batch feldolgozás bemutatásához az 5.7. szakaszban megismert meghibásodásokkal módosított csomagolósor modellt fejlesztjük tovább. A **Batch** és **Separate** modulokkal kiegészített modell logikája az *5.32. ábrán* látható. Az érkezési időköz a batch feldolgozással módosított modellben legyen egyenletes eloszlású 5 és 10 másodperc között. Tegyük fel, hogy a címkézési részfolyamat batch mérete 5, amit a folyamatban később egyedi darabokra kell felbontani. Feltételezzük továbbá, hogy a csomagolási részfolyamatban 10 egységet kezelünk együtt, azaz a csomagolás batch mérete 10. Minden csoportos címkézés, illetve csomagolás műveleti ideje 25 illetve 30 s. A csoportosítás és a felbontás megvalósításához, a címkézési és a csomagolási részfolyamatok puffer méretét a batch mérethez kell igazítani. Ezt egyszerűen úgy érjük el, hogy megszüntetjük a blokkolást és megadjuk a véges puffer kapacitásokat. A meghibásodásokat és a leállásokat a módosított modellben megtartjuk. Arra vagyunk kíváncsiak, hogy a „*Toltesi folyamat.Queue*” és a „*Atfutasi ido*” nevű **Record** modul között mekkora az átfutási idő.



5.32. ábra: A batch feldolgozással módosított csomagolósor modell



5.33. ábra: A „*Batch képzés 1*” nevű **Batch** modul dialógusablaka



5.34. ábra: A „*Batch bontas*” nevű **Separate** modul dialógusablaka

A lezárási részfolyamatot követő „*Batch képzés 1*” nevű **Batch** modul dialógusablaka az *5.33. ábrán* látható. Ezzel a modullal temporary (ideiglenes) batch-eket képzünk (Type mező), és batch-ek mérete 5 (Batch Size mező). A Rule mező azt határozza meg, hogy a batch minden belépő entitást tartalmaz (az opció *any Entity*), vagy csak bizonyos közös tulajdonsággal rendelkező entitásokat, amely tulajdonságokat az entitások attribútumai írják le (az opció *By Attribute*). A Type mezőben a *Temporary* opció azt jelenti, hogy később a batch-et fel kívánjuk bontani, ezzel szemben a *Permanent* opció állandó batch-et hoz létre. Végül a Save Criterion mezőt arra használjuk, hogy a batch-et alkotó entitások attribútumai közül kiválasszuk azt, amelyet a batch-hez akarunk rendelni. Az *5.33. ábrán* a választott opció *Last*, ami azt jelenti, hogy az adott batch attribútuma örökli az utoljára belépő alkotóentitás attribútumát. A modellben például a batch-be utoljára belépő entitás „*ErkIdo*” nevű attribútumának az értékét rendeljük a batch-hez.

A címkézési részfolyamatot követő „*Batch bontas*” nevű **Separate** dialógusablakát az *5.34. ábra* mutatja. A Type mezőben a *Split Existing Batch* opció választás a batch-et az eredeti alkotóelemeire bontja fel. A Type mező másik opcióját a *Duplicate Original* arra használjuk, hogy a modulba belépő entitásokat duplikáljuk. A *Retain Original Entity Values* opció választás azt eredményezi, hogy a felbontás után a batch minden alkotóeleme az eredeti attribútumát nyeri vissza.

A krimpelési részfolyamat után a termékegység entitásokat 10 entitást tartalmazó batch-ekben csomagoljuk. A csomag entitás ezt követően belép a további modulokba, amelyekben megmérjük az átfutási időt és az érkezési időközöket és végül a „*Befejezes*” nevű **Dispose** modulon keresztül távozik a rendszerből.

A szimuláció eredményei az *5.35. ábrán* megjelenő *Resources* jelentésben és az *5.36. ábrán* megjelenő *User Specified* jelentésben tanulmányozhatók.

7:45:19 **Resources** október 23, 2011

Batch feldolgozással módosított csomagolósor Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 100 000,00 Time Units: Seconds

Resource Detail Summary

Usage

	Inst Util	Num Busy	Num Sched	Num Seized	Sched Util
Cimkezo	0,67	0,67	1,00	2 666,00	0,67
Csomagolo	0,40	0,40	1,00	1 332,00	0,40
Krimpelo	0,67	0,67	1,00	13 328,00	0,67
Lezaro	0,67	0,67	1,00	13 335,00	0,67
Tolto	0,87	0,87	1,00	13 336,00	0,87

5.35. ábra: A batch feldolgozással módosított modell *Resources* jelentése

7:45:51 **User Specified** október 23, 2011

Batch feldolgozással módosított csomagolósor Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 100 000,00 Time Units: Seconds

Tally

Between	Average	Half Width	Minimum	Maximum
Erkezesi idokozok	74.9941		65.0000	88.6404

Interval	Average	Half Width	Minimum	Maximum
Atfutasi ido	91.9193	0,033092160	91.5000	97.9789

Output

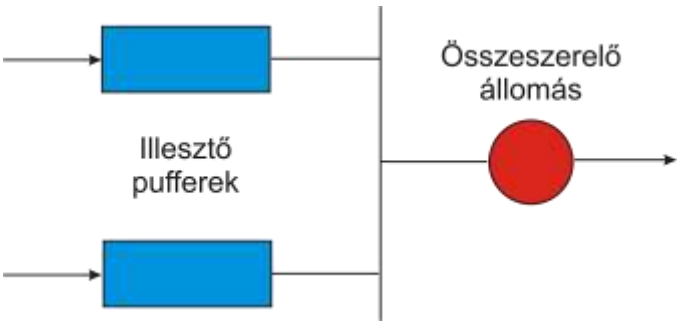
Output	Value
Teljesitmeny db per	0.01333438

5.36. ábra: A batch feldolgozással módosított modell *User Specified* jelentése

5.10. Szerelési műveletek

Gyártási környezetben nagyon gyakoriak az olyan szerelési műveletek, amikor különböző pufferekből érkező alkatrészeket egy végtermékké vagy fél-késztermékké kell összeszerelni. Az érkező összetartozó alkatrészeket az összeszerelő állomás előtt az ún. illesztő pufferekben helyezzük el (*5.37. ábra*).

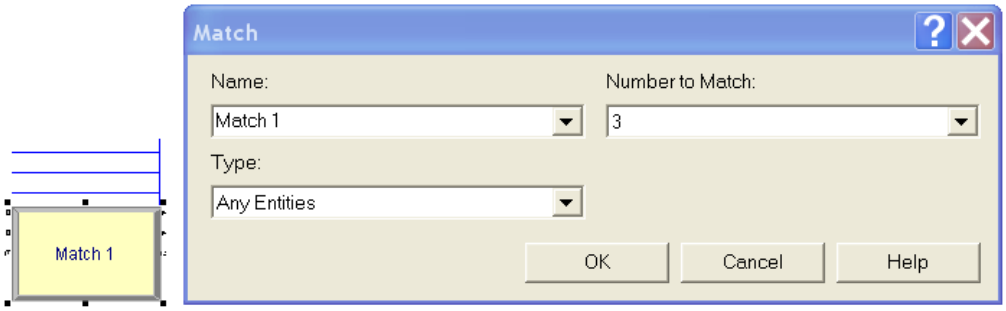
Az összeszereléshez az illesztő pufferből kivett alkatrészek száma változatos lehet. Azonban az összeszerelési művelet csak akkor csak is akkor kezdődhet el, ha az összetartozó alkatrészek mindegyike jelen van a pufferekben.



5.37. ábra: Az összeszerelési művelet sematikus vázlata

Az Arena-ban a **Match** modul gondoskodik az összetartozó alkatrészek szerelés előtti egymáshoz rendeléséről. Pontosabban a **Match** modult arra használjuk, hogy szinkronizálja a különböző termékegység entitások mozgását a modellben. Például az érkező összetartozó entitások addig várakoznak egymásra a puffekrekben, amíg a valamennyi alkatrész meg nem érkezik. Amikor az összetartozó alkatrészeket tartalmazó batch kompletté válik, az entitások együtt folytatják a mozgásukat a modell logika által kijelölt útvonalon.

A **Match** modul és a dialógusablaka az *5.38. ábrán* látható. Vegyük észre, hogy a modell ikon felett a sorokat szimbolizáló félegyenesek száma azonos a Number to Match mezőben megadott értékkel (az *5.38. ábrán* ez 3), továbbá a **Match** modul kilépési pontjainak a számát is ez a paraméter határozza meg.



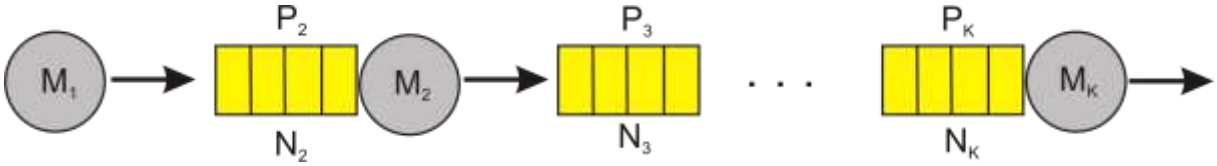
5.38. ábra: A **Match** modul ikon és a dialógusablaka (jobbra)

A **Match** modul működési elve a következő. Mihelyt **Match** modul minden pufferében (sorában) legalább egy entitás található, akkor a modul minden pufferből pontosan egy entitást emel ki, és ezeket egyidejűleg a kilépési pontokhoz kapcsolt modulba vagy modulokba küldi. Ez a működési elv lehetővé teszi az entitásmozgások szinkronizálását. Lényegében a **Match** modult úgy használhatjuk az összeszerelési műveletek modellezésére, hogy az egyik kilépési pontot a kijelölt munkaállomáshoz kapcsoljuk (ezen az ágon mozgó entitás lesz az összeszerelt termékegység), a többi kilépési pontot pedig egy **Dispose** modulhoz, amelyen keresztül az entitások elhagyják a rendszert. A szinkronizált entitásokat a Type mezőben kiválasztott opció határozza meg. Az *Any Entities* opció az illesztő pufferekben lévő valamennyi entitását szinkronizálja, a *Based on Attribute* opció csak azokat az entitásokat, amelyeknek az attribútuma az Atribute Name mezőben megadott értékkel azonos. Ha az entitásokhoz korábban szín attribútumot rendeltünk, akkor az utóbbi opcióval az azonos színű entitásokat illeszthetjük egymáshoz. Például, olyan batch-et definiálhatunk, amely csak vörös entitásokat tartalmaz. Ez a batch csak akkor jön létre, ha valamennyi pufferben (sorban) legalább egy vörös entitás talál-

ható. Azonban jegyezzük meg, hogy valamennyi pufferből csak egy és csak is egy vörös entitás lesz kiválasztva. Ha ugyanabból az entitásból több egységentitást kell egy termékbe beépíteni, akkor a **Match** modul előtt egy **Batch** modult kell használni, ami az adott számú entitást batch-be összevont egységentitássá alakítja. Ezután a **Match** modullal szinkronizáljuk a több elemi entitásból álló batch-eket.

5.11 A gyártósor modellek verifikációja

Ebben a szakaszban a gyártósorok néhány aspektusát vizsgáljuk, amelyek fontos szerepet játszanak a sorok viselkedésének a megértésében és a modell logika verifikálásában.



5.39 ábra: Egy általános gyártósor sematikus vázlata

Tekintsük az *5.39. ábrán* látható gyártósort. Itt az M_i jelöli az *i*-edik munkaállomást és P_i az *i*-edik puffert az M_{i-1} és az M_i munkaállomás között. Az *i*-edik munkaállomás kapacitása véges, N_i (amely nem tartalmazza az éppen megmunkálás alatt álló entitásokat egyetlen munkaállomáson sem). Az M_i műveleti ideje X_i legyen véletlen változó. Feltételezhetjük, hogy az M_1 munkaállomáson mindig van elegendő alapanyag, és anyaghiány miatt a munkaállomás soha nem várakozik. Egy másik lehetőség, hogy az M_1 –hez tartozik egy puffer, amelybe az entitások véletlenszerűen vagy ütemezetten érkeznek.

Definiáljuk a következő valószínűségeket:

$P_i(I)$ annak valószínűsége, hogy az M_i tétlen (*idle* állapot).

$P_i(B)$ annak valószínűsége, hogy az M_i blokkolt (*busy* állapot).

$P_i(D)$ annak valószínűsége, hogy az M_i meghibásodott (*failed* állapot).

$P_i(U)$ annak valószínűsége, hogy az M_i működik (*busy* állapot).

Ekkor az M_i kihasználtsága a következő kifejezéssel számítható:

$$P_i(U) = 1 - P_i(I) - P_i(B) - P_i(D), \tag{5.4}$$

ami azt jelenti, hogy az M_i akkor működik, ha nem tétlen, nem blokkolt és nem hibásodott meg. Amíg az M_i működik ($P_i(U)$ valószínűséggel *Busy* állapotban van), az alatt $1/E[X_i]$ sebességgel termel és a teljesítménye:

$$\overline{o}_i = \frac{P_i(U)}{E[X_i]}. \tag{5.5}$$

Abban az esetben, ha nincs veszteség (nincs selejt), a termékegységek hosszú ideig tartó áramlása (teljesítménye) *K* számú munkaállomás mindegyikére azonos:

$$\overline{o}_1 = \overline{o}_2 = ... = \overline{o}_K \tag{5.6}$$

Így a munkaállomások teljesítményei (átbocsátóképességük) azonos és egyenlő a gyártósor teljesítményével ($\overline{o}_l$). Ha van termékegység entitás veszteség, az adott valószínűséggel leírható selejtnek köszönhetően, akkor a teljesítményt korrigálni kell, méghozzá az áramlás irányában elhelyezkedő munkaállomásokon azonos mértékben. Az *5.4–5.6* egyenletek segítenek a modell verifikálásában.

A *Little* formula (5.2 egyenlet) ugyancsak alkalmazható a teljes gyártósorra (lásd az 5.4 szakaszban). Tekintsük ismét az *5.39. ábra* szerinti gyártósort K számú munkaállomással és $K-1$ számú pufferrel, és legyen $\overline{F}_S$ a termékegység entitás rendszerben töltött átlagos időtartama. Legyen továbbá a $\overline{P}_j$ puffer átlagos tartalma $\overline{N}_j$ és $\overline{N}_S$ jelölje a termékegység entitások átlagos számát a rendszerben. A Little formula

$$\overline{N}_S = \overline{o}_l \overline{F}_S , \tag{5.7}$$

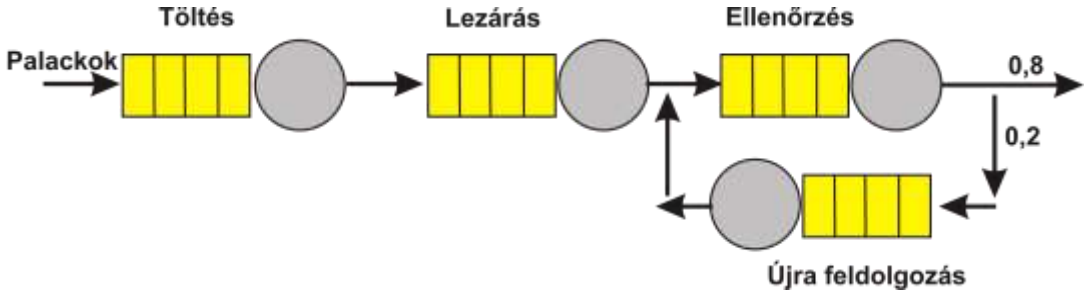
amelyet az 5.4 szakaszban már használtunk. Más oldalról az $\overline{N}_S$-t közvetlenül is lehet számítani a

$$\overline{N}_S = \sum_{j=2}^{K} \overline{N}_j + \sum_{j=1}^{K} P_j(U) + \sum_{j=1}^{K-1} P_j(B) \tag{5.8}$$

kifejezéssel, ahol a jobboldalon az első tag a gyártósor átlagos teljes puffer tartalma (kivéve az első munkaállomást, amelynek nincs puffere, és kivéve a kiszolgálás alatt álló entitásokat), a második tag az átlagos kiszolgálás alatt álló entitások száma, amikor munkaállomás erőforrása *busy*, és végül a harmadik tag az átlagos kiszolgálás alatt álló entitások száma, amikor munkaállomás erőforrása blokkolt. Verifikálási célokra a modellező az 5.7 és az 5.8 egyenleteket egyaránt használhatjuk.

Gyakorlatok

1. Háromlépcsős gyártósor: Tekintsük az alábbi ábrán vázolt töltő, lezáró és ellenőrző munkaállomásokból álló gyártósort.



A palackok exponenciális eloszlás szerint érkeznek átlagosan 2 perces érkezési időközzel. A belépő palackok a töltő munkaállomás végtelen pufferében nyernek elhelyezést. A töltő munkaállomáson a palackok töltése batchben történik és a batch mérete 5 palack. A töltési idő egyenletes eloszlású 3 és 5 perc között. Ezután a teljes batch belép a lezáró véges kapacitású sorába (a sor hossza 10 palack). Ha azonban a lezáró nem képes fogadni a teljes batchet a töltő addig blokkolódik, amíg a lezáró sorában elegendő hely nem keletkezik. A töltő meghibásodása átlagosan 50 percenként következik be, és a meghibásodás-mentes idő eloszlása exponenciális. A javítási idő trianguláris eloszlású, (2, 5, 10) perc paraméterekkel. A hiba bármely időpontban bekövetkezhet. A lezáró munkaállomáson egyidejűleg 3 palackot zárunk le, azaz műveletet ugyancsak batchben végezzük. A művelet (3 palack lezárása) pontosan 3 percet igényel. A minőségellenőrző munkaállomás pufferének kapacitása is véges, 10 palack. Az ellenőrzés időtartama 1,45 perc palackonként. A gyakorlati tapasztalatok szerint a palackok 80%-a átmegy az ellenőrzésen és elhagyja a rendszert, a maradékot az újra feldolgozó állomásra küldjük, ahol korrigáljuk a töltést és a lezárást. Az újra feldolgozó munkaállomás puffer-kapacitása végtelen nagy, és egy szerveres. A műveleti idő egyenletes eloszlású, 1 és 4 perc között. A töltés és a lezárás korrekciója után a palackokat az ellenőrző munkaállomás elé küldjük vissza. Az újra feldolgozott palackok prioritása nagyobb, mint a többi palacké. Megjegyezzük ugyanaz a palack többször is áthaladhat az újra feldolgozó munkaállomáson.

Fejlesszük ki a leírt gyártósor Arena modelljét, és szimulációs idő hosszát állítsuk 365 napra és 8 óra munkaidő/napra.

Becsüljük a következő statisztikákat:

Az átlagos WIP-et minden pufferben.
Az erőforrás kihasználást minden munkaállomáson.
Átlagos átfutási időt a rendszerben.
Az átfutási idő eloszlását 5 intervallummal.
Állapotvalószínűségeket (*busy, idle, down, blocked*),
Annak valószínűségét, hogy a töltő, a lezáró és az újra feldolgozó munkaállomásokat az ellenőrző munkaállomás egyidejűleg blokkolja.

2. Gyártórendszer minőségellenőrzéssel és újra feldolgozással. Tekintsük a következő gyártó rendszert, amely nyomtatott áramköröket gyárt többlépcsős rendszerben és korlátlan puffer kapacitásokkal. A termékegységek érkezési időköze exponenciális eloszlású 2,3 óra átlaggal. Az érkeztető terület két részre osztott: *Alkatrész1* és *Alkatrész2*. A megérkezés pillanatában az alkatrészek szeparálódnak és két ágon, két különböző folyamatnak megfelelően haladnak. Az egyik ágon haladó *Alkatrész1*-ről a kémiai dezintegrálását (maratását) követően, eltávolítják az áramkört (vezetőpályákat) fedő lakréteget, majd további műveletek eredményeként kialakul a nyomtatott áramkör. A dezintegrálás időtartama egyenletes eloszlású 0,5 és 2 óra között, az áramkör végleges kialakításának az időtartama is egyenletes eloszlású 1 és 2 óra között. Az *Alkatrész2* a másik ágon halad, ahol elektromechanikai tisztítás után alakítják ki az áramkört. A két művelethez tartozó időtartamok exponenciális eloszlásúak 1,4 és 1,5 óra középértékekkel. Az összeszerelő munkaállomás előtt minkét ágról kiemelnek egy-egy alkatrészt és azokat egy termékegységgé szerelik össze. A szerelés időtartama pontosan 1,5 óra. Az összeszerelt egységek a minőségellenőrző munkaállomásra mozognak, ahol az ellenőrzés batchben történik. A batch mérete 5 egység. Az ellenőrzési idő trianguláris eloszlású, amelynek a paraméterei 6, 8, és 10 óra. A minőség-ellenőrzés után a batchet elemeire bontják, amelyek elhagyják a rendszert. Nyolc óra működés után a folyamat leáll 1 órára, amely alatt elvégzik a karbantartást. A folyamatoknál a következő véletlen meghibásodásokat figyelték meg:

Munkaállomás neve	Hiba típusa	Hibamentes idő (óra\egység)	Javítási idő (óra)
Dezintegrálás	Random	Expo(1/20)	Unif(1, 2)
Tisztítás	Random	Expo(1/30)	Beta(5, 1)
Áramkör kialakítás 1	Random	Expo(1/25)	Unif(2, 5)
Áramkör kialakítás 2	Random	Expo(1/25)	Beta(5, 1)
Minőségellenőrző	Adjustment	200 egység	Tria(1, 3, 4)

Ne felejtsük el, hogy a béta eloszlás paramétereinek sorrendje fordított, és az exponenciális eloszlás paramétere a középérték.

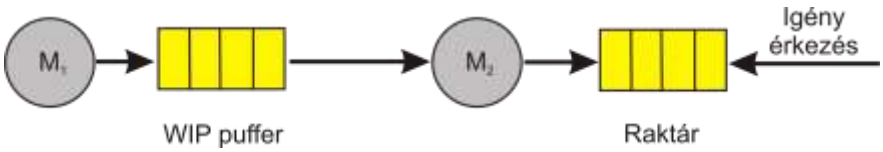
a) Fejlesszük ki a gyártási rendszer Arena modelljét, a szimuláció időtartamát állítsuk 1 évre.

b) Becsüljük meg a következő statisztikákat:

Minden folyamat kihasználtságát.
Az átlagos műveleti időt (delay) minden folyamatnál.
Az összeszerelt egység átfutási időtartamát.
A minőségellenőrző munkaállomás pufferében a munkadarabok számának az eloszlását.
A rendszerben a karbantartás előfordulásának valószínűségét.
A folyamatok állapotainak valószínűségeit (idle, busy, failed és karbantartás).

3. Kétlépcsős gyártási rendszer raktározással. Tekintsük a következő kétlépcsős gyártási rendszert, amelyben az első lépcsőben gyártott termékeket a második lépcsőben tároljuk.

Az ügyfelek a termékigényeikkel a raktárhoz érkeznek és megrendeléseiket a raktár készletéből kielégítjük (lehetőség van az utánrendelésre is). Amikor a raktárban a készlet felülről eléri az újrarendelési pontot az M_2 gép elkezd termelni, amihez az alapanyagot a WIP pufferben tároljuk. A puffert az M_1 gép táplálja.



Az első lépcső így az M_1 gépet és a WIP puffert tartalmazza, amelynek a kapacitása 40 termékegység. Az M_1 gép működéséhez mindig rendelkezésre áll az alapanyag, és egy termékegység műveleti ideje 1 óra. A második lépcső az M_2 gépet és a végterméktároló raktárt foglalja megában. Az M_2 gép műveleti időtartama 0,75 óra/termékegység. A végterméket igénylő ügyfelek a raktárhoz *Poisson* folyamat szerint érkeznek 1/9 óra paraméterrel. Az igény eloszlása is Disc({(0,3, 3), (0,3, 6), (0,4, 12)}), és a többlet ügyféligényt utánrendeléssel elégítik ki.

Az M_1 gép blokkolt, amikor a végtermék nem tud belépni a WIP pufferbe helyhiány miatt, és addig marad blokkolt, amíg a WIP pufferben a szükséges hely elérhetővé válik. Az M_2 gép működését az újrarendelési pont, ami 60 termékegység, szabályozza. A termelés azonnal indul, mihelyt a raktárban a készletszint alacsonyabb, mint 60 termékegység, és az M_2 gép blokkolttá válik, ha a készletszint eléri a 60-at. (Megjegyezzük, a raktározást részletesebben tárgyaljuk a 6. fejezetben az ellátási láncokkal összefüggésben.)

a) Fejlesszük ki az ismertetett gyártási rendszer Arena modelljét és a szimuláció időtartama legyen 10 év.

b) Becsüljük a következő statisztikákat:
- A gépkihasználást (kizárva a tétlenséget).
- A blokkolás valószínűségét mindkét gépre.
- Az átlagos készletszintet a WIP pufferben és a raktárban.
- Az átlagos újrarendelési szintet a végtermékraktárban.
- Az érkező ügyfelek újrarendelésének a valószínűségét.

6. Ellátási-lánc rendszerek modellezése

Az ellátási-láncok fontossága abból a tényből ered, hogy a gazdasági tevékenységek jelentős szeletét fedik le (az Egyesült Államokban például az ellátási-láncok körülbelül 10 százalékkal járulnak hozzá a bruttó hazai termékhez). A vállalatok annak érdekében, hogy javítsák a helyzetüket, versenyelőnyt szerezzenek, vagy éppen életben maradjanak, érdekeltek az ellátási-láncok hatékony működésében, ami a vevői elvárásokkal is találkozik.

Az ellátási-lánc rendszer (röviden ellátási-lánc) a termeléstől az értékesítésig terjedő hálózatnak tekinthető, amelyet az entitások áramlása köt össze, beleértve a termékek életciklusát. E hálózat is csomópontokból és élekből áll. A csomópontok képviselik a szállítókat, gyártókat, elosztókat, nagy- és kiskereskedőket, illetve az ezek termékeit tároló létesítményeket. A csomópontokat összekapcsoló élek az utak, amelyeken árucikkeket szállítanak különféle módon, (közúton, vasúton, légifolyosókon, és így tovább). Elméletileg közelítve, az ellátási-láncok lényegében lépcsősen rendezett, ellátó hálózati struktúrák, előre, visszafelé és mindkét irányban áramló összetevőkkel. A nyersanyagok, alkatrészek, termékek, stb. a főáram irányában mozognak, a pénzmozgások ellentétes irányúak, és végül az információk mindkét irányban áramlanak. Ennek ellenére, az ellátási-láncokban az entitásáramlások iránya nincs szigorúan meghatározva, mert például a hibás termékek visszafelé mozoghatnak javításra, vagy előfordulhat, hogy az átutalt pénzt valamilyen oknál fogva vissza kell fizetni. Az alapvető ellátási-lánc lépcsők a következők:

1. Az **ellátási lépcső** nyersanyagokkal vagy alkatrészekkel látja el a gyártást.
2. A **termelési lépcső** a nyersanyagokat és az alkatrészeket késztermékké alakítja át.
3. Az **elosztási lépcső** az elosztási hálózatból áll: raktárak, az elosztó központok, szállítási létesítmények, amelyek késztermékeket mozgatnak a kereskedőkhöz.
4. A **kereskedelmi lépcső** értékesíti a termékeket a végfelhasználóknak (vevőknek).

Gyakorlatban a komplex ellátási-láncok széles tartományt ölelnek át az egyszerű ellátási-láncoktól, ahol mindegyik lépcső egyetlen csomópont, a bonyolultakig, ahol mindegyik lépcső önmaga is egy bonyolult hálózat, amely több csomópontból és élből áll. Például a termelési lépcső lehet hierarchikus, amelyben a beszállító üzemek alkatrészekkel látják el az alkatrészeket bonyolultabb félkész vagy végtermékekké összeszerelő integrátor üzemet. Hasonlóan az elosztóhálózatok tartalmazhatnak nagyszámú hierarchikusan szervezett raktárt az elosztó központtól a nagykereskedelmi raktáron át a kiskereskedelmi raktárig, Tény az ellátási-lánc átlépheti a nemzeti határokat és kiterjedhet több kontinensre.

Az **ellátási-lánc menedzsment** (SCM: supply chain management) küldetése elméletileg, hogy a gyártott és elosztott termék a megfelelő mennyiségekben, a megfelelő helyen és a megfelelő időben álljanak rendelkezésre, alacsony költségű és magas szintű vevőszolgálat mellett. Lényegében, az SCM a küldetését úgy teljesíti, hogy egyensúlyt keres rendszerköltségek és a vevői elégedettség között. Az SCM ennek érdekében a következő ellátási-lánc elemek költséghatékony integrációjával és koordinációjával foglalkozik:

1. Az ellátási-lánc csomópontjait felölelő szállítók, gyárak, raktárak, és boltok.
2. A csomópontokat összekapcsoló szállítási hálózat.
3. Az informatika-infrastruktúra, amely lehetővé teszi az adatcserét az ellátási-lánc csomópontjai között, támogatja az ellátási-lánc tervezését és a napi operációk lebonyolítását.
4. A módszerek és algoritmusok anyagáramok és a készletmenedzsment irányításához.

SCM sok kérdéssel, kihívással és nehéz döntésekkel néz szembe. Például kérdés, hogy az ellátási-lánc döntési rendszere központosított legyen (egyetlen döntéshozó felel az egész hálózatért, aki minden elérhető információt felhasználhat), vagy osztott (több decentralizált döntéshozó, akik helyi információkat használnak, a helyi hálózatot érintő döntésekhez)? A dön-

téshozóknak megengedjük-e, hogy információikat megosszák egymással (átláthatóság)? Hogyan csökkenthető a rendelési tételnagyság változékonysága? Például figyelemre méltó tapasztalat, hogy egy ellátási-láncban az átláthatóság hiánya az úgynevezett ostorhatást idézi elő, amely szerint a rendelés változékonysága növekszik, ahogy az ember magasabbra jut az ellátási-láncban. Amíg ezek a kérdések vállalatorientált kérdések, addig az ellátási-láncokban ügyfélorientált kérdések is előfordulnak, például a vevői elégedettség, amely a hiány gyakoriságával összefüggő probléma. Például, ha a vevői igény a rendelkezésre álló készletből nem teljesen elégíthető ki. A hiányt utánrendeléssel lehet pótolni, vagy az eladás elveszik. Mindkét esetben a készlettartási költség növekedését kell szembeállítani a nem teljesen kielégített vevői igény okozta veszteséggel.

A felvetett kérdéseket tanulmányozva, a modellezők jellemzően a következő kulcsfontosságú kimeneti mutatókra koncentrálnak, amelyek összefüggenek a pénzügyi mutatókkal:

Készletezési költségek.
Átlagos igényszint.
Átlagos készletszint és az utánrendelés szintje.
A vevőszolgálat színvonala (kielégített vevőigények aránya, vagy kiszolgálási arány).
Az elveszett eladások aránya és mennyisége.

A jó működés érdekében, az ellátási-láncok irányításához ún. készletezési politikákat (mechanizmusokat) alkalmaznak, amelyekkel a készletek pótlására szolgáló újrarendeléseket szabályozzák. A **rendelési időköz** attól függ, hogy a rendeléseket rögzített t időközönként kell feladni, vagy a készletutánpótlásról akkor döntünk, amikor a készletszint valamilyen s **minimális készletszintre** (újrarendelési pontra) csökken. A **rendelési tétel nagysága** lehet rögzített Q, vagy a rendelési tétel akkora mennyiségre szól, hogy a beérkezés után a készlet egy előre meghatározott S **maximális készletszintet** ér el. A készletgazdálkodási mechanizmusok ezek kombinációi, így beszélhetünk (t,Q), (t,S), (s,Q) és (s,S) mechanizmusokról [7]. Az utóbbi két készletezési mechanizmust a készletszint átlépés elve vezérli. A maximális készletszintnek nevezzük azt, amelyet alulról érünk el, vagy lépünk át, és minimális készletszintnek nevezzük azt, amelyet felülről érünk el, vagy lépünk át. A készletet figyelés lehet folytonos vagy periodikus, és a rendeléseket akkor kezdeményezzük, amikor a készletszint az újrarendelési pontra vagy az alá esik. Az iparban a leggyakrabban a következő tipikus készletezési politikákat használják:

(s, S) **készletezési politika**: A készlet célszintje (a maximális készlet) S, és az újrarendelési pont s. A készletfeltöltést (az újrarendelést) azonnal felfüggesztjük, mihelyt a készletszint eléri vagy átlépi a célszintet. A felfüggesztés addig tart, amíg a készlet az újrarendelési pontig vagy az alá nem csökken. Ekkor a készletfeltöltés folytatódik. Például, amikor a beszállító egy termelőüzem, akkor a feltöltés felfüggesztése a termelés leállítását, a feltöltés folytatása pedig a termelés indítását jelenti.

Rendelés S-ig készletezési politika: Ez az (s, S) készletezési politika speciális esete, ahol $s = S \geq 1$. Más szóval a feltöltést folytatódik, mihelyt a készletszint felülről éri el a célszintet (S), vagy az alá csökken.

(s, Q) **készletezési politika**: Q a rendelési tételnagyság és s az újrarendelési pont. Amikor a készletszint az újrarendelési pontra (s), vagy az alá csökken, akkor újrarendeléssel a készletszintet Q mennyiséggel növeljük.

Jellemzően, amikor egy megrendelést feladnak a beszállító felé, akkor a rendelés beérkezéséig időbeni késés jelentkezik, ez az **utánpótlási idő**. Az utánpótlási idő alatti igény annak az igénynek a nagysága, ami gyakorlatilag az utánpótlási időtartama alatt keletkezik, ami jellemzően véletlenszerű, és ami miatt hiány is keletkezhet. Következésképpen, hogy csillapítsák a bizonytalan utánpótlási idő alatti igény hatását, a vállalatok **biztonsági készletet** képeznek,

ami egy extrakészlet, a megfelelő vevőszolgálat-színvonal fenntartása érdekében. A vállalatok úgy is határozhatnak, hogy az új megrendeléseket még az előző megrendelés érkezése előtt feladják. A **készletpozíció** a készletszint plusz a megrendelt készlet, mínusz az utánrendelés. Rendelési döntések sokszor a készletpozíción alapulnak és nem a készletszinteken.

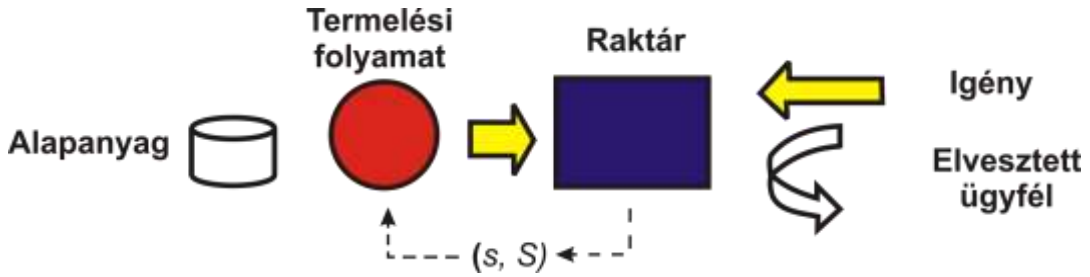
Ez a fejezet ellátási-láncok lépcsői közötti készletmenedzsmentre és anyagáramlásra helyezi a hangsúlyt.

6.1. Egytermékes termelési és készletezési rendszer (6.1. modell)

Ebben az alfejezetben egy termelési-készletezési rendszer általános modelljét mutatjuk be, amely egy olyan termelőlétesítményből áll, ami időnként meghibásodik, és amely egy raktárt lát el egy fajta termékkel. Ez az általános modell azt illusztrálja, hogy a készletezési-politika hogyan szabályozza a termék áramlását termelés és raktár között.

6.1.1 A probléma ismertetése

Tekintsünk egy termelési-készletezési rendszert, ahol egyetlen termék gyártása folyik. A rendszer vázlatos működése a *6.1. ábrán* látható. A termelési folyamat nyersanyag forrása az alapanyag tároló, és a termelés eredményeként keletkező késztermékeket (egységeket) a raktárban tárolják. Az ügyfelek (vevők) termékrendeléseikkel (igényeikkel) érkeznek a raktárhoz, és ha egy igény nem teljesen elégíthető ki a raktári készletből, akkor a kielégítetlen igény elveszik.



6.1. ábra: Egy általános termelési-készletezési rendszer

A rendszer működésével kapcsolatban a következő feltevéseket tesszük:

Az alapanyag tárolóban mindig van elegendő nyersanyag, ezért a termelési folyamatban nem jelentkezik hiány.

A termelés 5 termékegységet tartalmazó tételekben (lot-okban) történik, és a tételeket a raktárban helyezik el. Egy tétel (lot) gyártásának a műveleti ideje egyenletes eloszlású, 10 és 20 perc között.

A termelési eljárásban véletlenszerű meghibásodások tapasztalhatók, amelyek bármilyen időpontban jelentkezhetnek (részletesen lásd a 4. fejezetben). A meghibásodások közötti idők exponenciális eloszlásúak 200 perces átlaggal, a javítási időközök normális eloszlásúak 70 perces átlaggal és 30 perces szórással.

A raktárban (*s*, *S*) készletezési politikát alkalmaznak, az újrarendelési pont $s = 150$ egység, a célszint $S = 500$ egység.

Az ügyfelek (vevők) érkezési időközei egyenletes eloszlásúak 3 és 7 óra között, és az ügyfelek igényei ugyancsak egyenletes eloszlást mutatnak 50 és 100 egység között. A programozás egyszerűsítése érdekében, az igényelt mennyiség bármilyen valós szám lehet az adott tartományban, amit az ANINT(UNIF(50, 100)) **Arena** függvénnyel egész számmá konvertálunk, így az igényelt mennyiségek egész számúak lesznek. Az ügyfél

érkezéskor a készletet ellenőrizzük, és ha elegendő készlet van a raktárban, akkor az igényt azonnal kielégítjük. Különben csak részlegesen, és a kielégítetlen igény elveszik.

A nyitókészlet 250, ezért a termelési folyamat a szimuláció kezdetén tétlen.

A rendelési költség két komponense a konstans beállítási költség (független a rendelt mennyiségtől) K=10000 Ft/rendelés és a darabköltség c=100 Ft/db. A fajlagos készlettartási költség h=2 Ft/db/óra és a fajlagos hiányköltség p=8 Ft/db/óra

A modellezés eredményeként, következő kimenő paramétereket szeretnénk megismerni:

1. A termelési folyamat kihasználtsága.
2. A termelőeszköz javítási idejének valószínűsége.
3. Az átlagos készletszint a raktárban.
4. Az átlagos igényszint.
5. A nem teljes mértékben kielégített ügyfelek aránya.
6. A nem teljes mértékben kielégített ügyfelekre vonatkoztatott átlagos elvesztett igény.
7. Az egységnyi időre eső átlagos készletezési költség Ft/órában

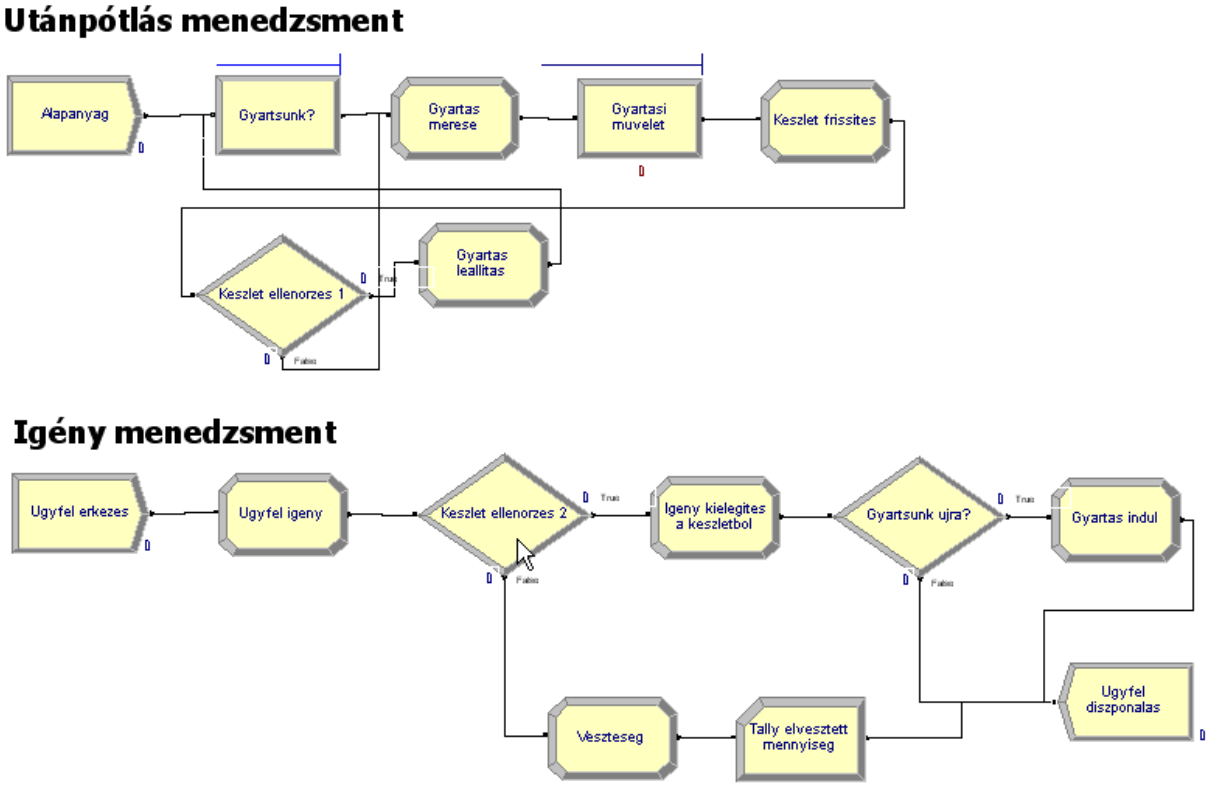
Az előzőleg részletesen leírt termelési-készletezési probléma lehet, hogy bonyolultnak tűnik, ennek ellenére csak a valóságos ellátási rendszerek durva egyszerűsítésének tekinthető. Azonban az alkalmazott elvek, módszerek kiterjeszthetők olyan valóságos rendszerekre, amelyekben gyakran egyidejűleg több terméket kell kezelni, vagy összetett, többlépcsős termelési/elosztási rendszerekre, amelyekben az egymáshoz kapcsolódó és egymástól függő lépcsőket együttesen kell modellezni.

6.1.2 A probléma Arena modellje

A probléma leírása és tanulmányozása után építettük fel *6.2. ábrán* látható egytermékes termelési-elosztási rendszer **Arena** modelljét.

A modell két szegmensből áll:

Utánpótlás menedzsment szegmens a termékegység entitásokat követi nyomon. Az entitás definíciókat táblázatnézetében a **Basic Process** panelen elérhető **Entity** adatmodulban vizsgálhatjuk és szerkeszthetjük. A modellben a termelési folyamatot a **Process** modul realizálja, amelyben az *Action* mező típusa *Seize Delay Release*. A *Seize* az erőforrást („*Gyartoeszkoz"*) a várakozó „*Anyag Entitas*"-hoz rendeli, amikor az erőforrás felszabadul. A *Delay* szimulálja a gyártást, amelynek a tételnagysága (batch mérete) öt. A gyártás után a *Release* felszabadítja az erőforrást. A „*Keszlet frissites*" nevű **Assign** modulban a pillanatnyi készletet a tételnagysággal (öttel) növeljük, majd a „*Keszlet ellenorzes 1*" nevű **Decide** modulban döntünk a gyártás folytatásáról vagy leállításáról a készletszint nagyságától függően. Ha a készlet nagyobb vagy egyenlő, mint a célszint (S = 500 egység), akkor a gyártást leállítjuk, különben a gyártás folytatódik. A gyártás leállítása addig tart, amíg a készletszint az újrarendelési pontra (s = 150) vagy az alá nem csökken. A vezérléshez a *Gyartas* nevű, 0 és 1 értékű döntési változót használjuk, amelynek az értékét a „*Gyartas leallitas*" nevű **Assign** modulban változtatjuk meg. A változó értékét a „*Gyartsunk?*" nevű **Hold** modulban vizsgáljuk, ha a változó értéke 0, akkor a gyártás áll, és akkor indul újra, amikor a változó értéke 1-re változik. A modellben azt feltételezzük, hogy a nyersanyag korlátlan mennyiségben mindig rendelkezésre áll. A „*Gyartas merese*" nevű **Assign** modulban számláljuk az adott pillanatig legyártott mennyiséget.



6.2. ábra: Az egytermékes termelési-készletezési rendszer **Arena** modellje

Igénymenedzsment szegmens generálja az ügyfeleket és az igényeiket, valamint változtatja a készletszintet az igények kielégítését követően. A „*Gyartsunk ujra?*” nevű **Decide** modulban ellenőrizzük a készletszintet, és ha a készletszint az újrarendelési pontig (s = 150), vagy az alá csökken, akkor a gyártást újraindítjuk. A ki nem elégített részigényeket, és a részben kielégített ügyfelek számát a „*Veszteseg*” nevű **Assign** modulban frissítjük.

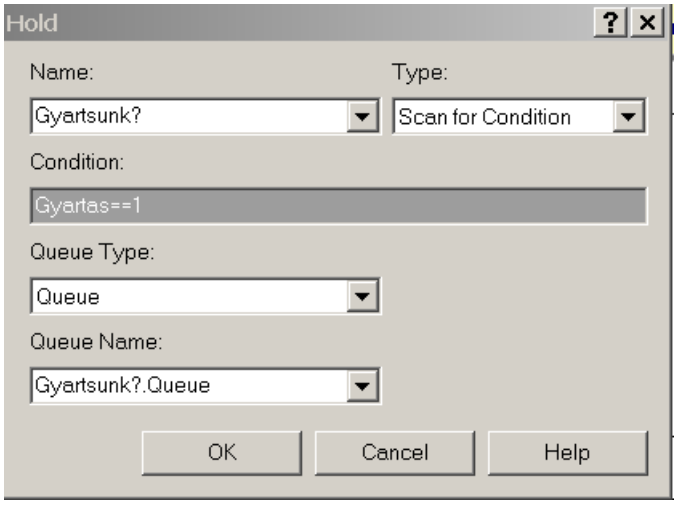
Az input és output adatok mindkét szegmensben vegyesen fordulnak elő. Logikailag a szegmensek input/output-modulokból állnak (változók, erőforrások, statisztika, stb.), beállítják az inputváltozók kezdeti értékeit, statisztikát számolnak, és összefoglaló jelentéseket hoznak létre. A következő részben bemutatjuk a *6.2. ábrán* látható modell kicsit részletesebb vizsgálatát.

6.1.3 Az utánpótlás menedzsment szegmens

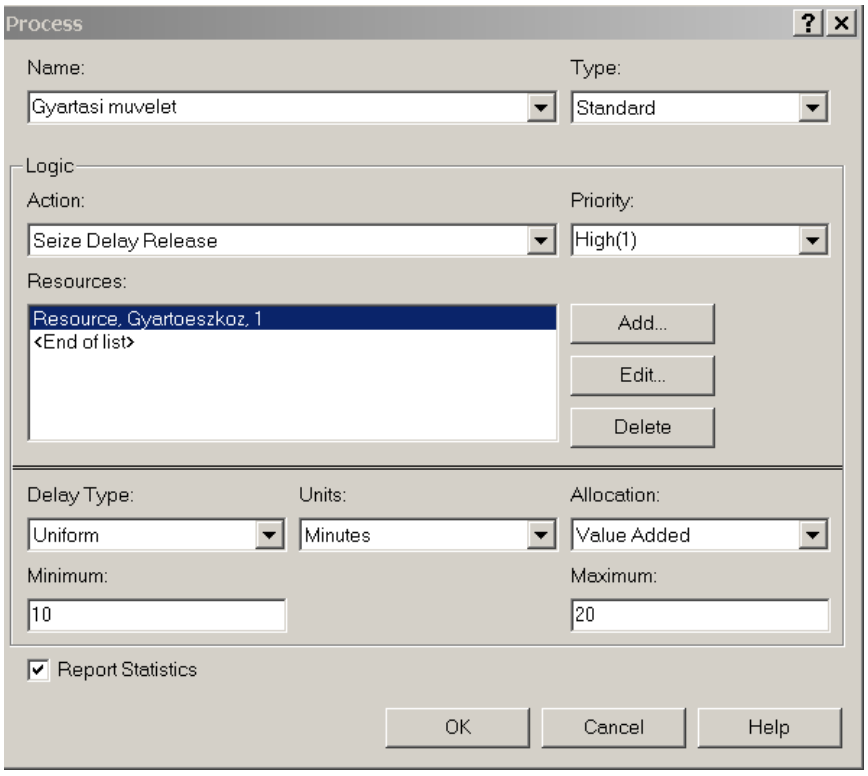
A modell logika felépítését az utánpótlás menedzsment szegmenssel kezdjük. Az ”*Alapanyag*” nevű **Create** modulban létrehozzunk egyetlen „*Anyag Entitas*” nevű kontrol entitást, amely a gyártás (csomagolás) tételenkénti (batching) működését vezérli. A **Hold** modul (amelynek a neve egy sejtelmes kérdés „*Gyartsunk?*”) arra szolgál, hogy indítsa vagy leállítsa a gyártást oly módon, hogy a **Process** modulba igyekvő kontrollentitást megállítja, vagy tovább engedi (*6.3. ábra*). Az **Advanced Process** panelen található **Process** modul (erőforrás lekötés, gyártás, erőforrás felszabadítás) valósítja meg a termelést, amelynek a műveleti idejét, Unif(10, 20) percben adjuk meg. A műveleti idő a korábbi leírás alapján 5 egység csomagolására vonatkozik.

Az „*Alapanyag*” nevű **Create** modulban egyszerűen a 0 időpontban egyetlen entitást („*Anyag Entitas*”) hozunk létre, ezért a *Max Arrivals* mezőbe 1-et írunk, és az érkezési időköz irreleváns. A **Create** modul ezek után inaktívvá válik. Az „*Anyag Entitas*” ezután a modellben kering, és minden kör megfelel egy termelési ciklusnak. Az entitás először a belép a **Hold** modulba („*Gyartsunk?*”), ahol ellenőrizzük a *Gyartas* nevű, 0 és 1 értékű (0=Off és 1=On), dön-

tési változót, amelynek a kezdeti értéke 0. A **Hold** modul egy olyan kapu, amely az entitást a logikai feltétel (*Condition*) teljesülésétől függően (*6.3. ábra*) tovább engedi, vagy blokkolja. Ha a *Gyartas*==1 igaz, és a gyártás folytatódhat, különben az entitás várakozik, és a gyártás szünetel. Miután az entitás felszabadul a „*Gyartas merese*” nevű **Assign** modulba lép, amelyben a „*Gyartott mennyiseg*” nevű változó értékét a „*Batch meret*”-tel növeljük.



6.3. ábra: A „*Gyartsunk?*”nevű **Hold** modul dialógusablaka



6.4. ábra: A „*Gyartasi muvelet*” nevű **Process** modul dialógusablaka

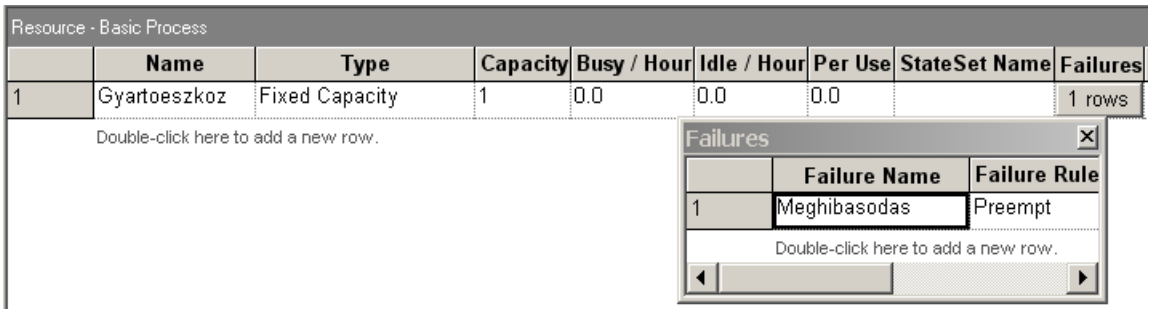
Ezt követően a tovább haladó kontrollentitás belép a **Process** modul sorába, és az erőforrás foglaltságától függően várakozik, vagy azonnal elkezdődik a gyártás. A „*Gyartasi muvelet*” nevű **Process** modul dialógusablakát *6.4. ábra* mutatja. A modul paramétereit a *6.1. táblázat* foglalja össze. A modulhoz tartozó „*Gyartasi muvelet.Queue*” nevű sorban a kontrollentitás arra várakozik, hogy a „*Gyartoeszkoz*” erőforrás egy egysége szabaddá váljon és elkezdődjék a gyártás. Az erőforrás nevét „*Gyartoeszkoz*” és mennyiségét (kapacitását) a *Resource* mező jobboldalán elhelyezett *Add* gombra kattintva adhatjuk meg. Amikor az erőforrás elérhető,

akkor azt a legmagasabb prioritású entitás fogja lekötni (*Priority High*(1)) a *Seize* hatására, és amíg a művelet folyik (*Delay*), azaz az erőforrás foglalt, a modul más entitást nem enged belépni. A gyártási művelet után a *Release* felszabadítja az erőforrás egy egységét. Megjegyezzük, egy entitás egyidejűleg akár több elérhető erőforrást is leköthet, és fordítva egy entitás egyidejűleg több lekötött erőforrást is felszabadíthat.

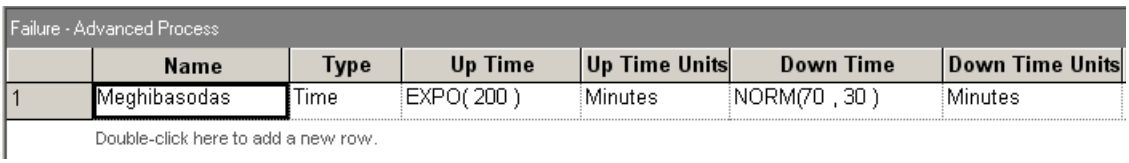
6.1. táblázat

A „*Gyartasi muvelet*” nevű **Process** modul paraméterei

Name Action	Gyartasi muvelet Seize Delay Release
Resource Type Resource Name Quantity	 Resource Gyartoeszkoz 1
Delay Type Units Minimum Maximum	Uniform Minutes 10 20



6.5. ábra: A **Resource** modul táblázatnézete felül a **Failures** párbeszédablakkal



6.6. ábra: A **Failure** adatmodul táblázat nézete

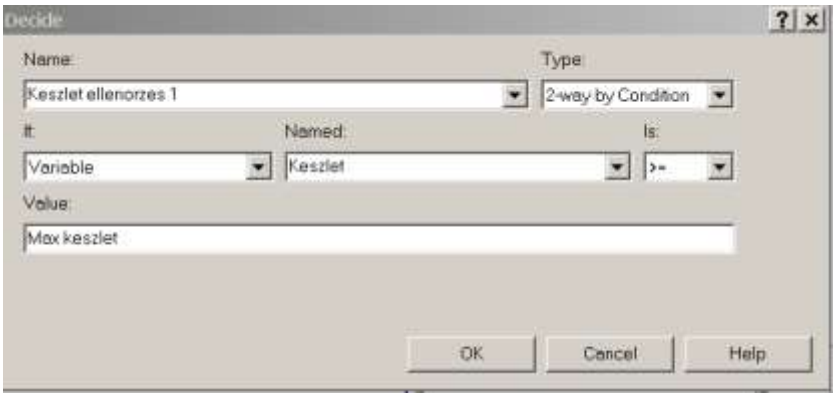
A **Resource** adatmodul táblázatnézetében a korábban már definiált „*Gyartoeszkoz*” nevű erőforrás a *6.5 ábra* felső részén látható. A *Failures* mezőhöz tartozó **Failures** ablakban, az ábra alján, adjuk meg a meghibásodások nevét: „*Meghibasodas*”. A „*Meghibasodas*” nevű hibát részletesen a **Failure** adatmodulban definiáljuk (*6.6. ábra*)**.** Itt adjuk meg a meghibásodások közötti időt (*Up Time*) és a hiba kijavításának az időtartamát (*Down Time*).

A raktári készletszintet a *Keszlet* nevű változó tartalmazza, amelynek a kezdeti értéke 250. Amikor a kontrollentitás belép a „*Keszlet frissites*” nevű **Assign** modulba, akkor a *Keszlet* változó értékét a *Batch meret*-tel, azaz öttel növeljük, vagyis a raktári készletszint öt egységgel növekszik.

Az **Assign** modult elhagyó kontrollentitás a „*Keszlet ellenorzes 1*” nevű **Decide** modul felé halad, amelynek a dialógusablaka a *6.7. ábrán* látható. Itt megvizsgáljuk, hogy a készletszint elérte-e a célszintet (*Max keszlet*). A vizsgálat két kimenete:

(1) Ha a készlet nagyobb vagy egyenlő, mint a célkészlet (*Max keszlet*), akkor a kontrollentitás a következő „*Gyartas leallitas*” nevű **Assign** modul felé mozog, amelyben a *Gyartas* nevű változó értékét 0-ra állítjuk, ami a gyártás felfüggesztését jelenti.

(2) Különben a kontrollentitás a *False* ágon lép ki és a **Hold** modul mögött belép a **Process** modulba, vagyis a gyártás folytatódik.

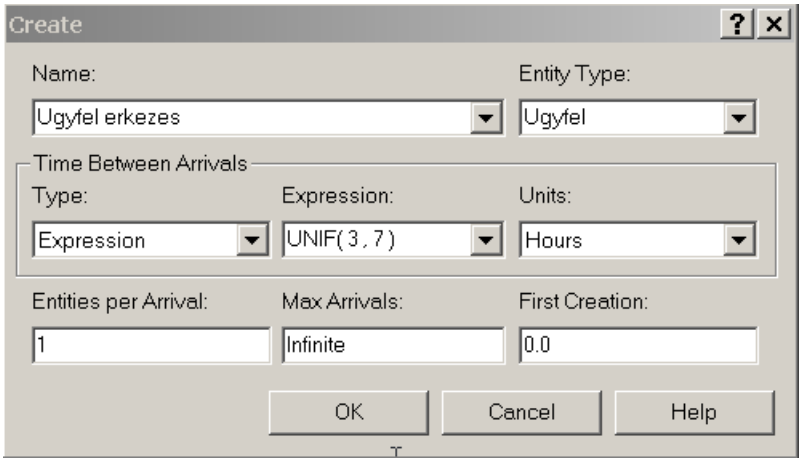


6.7. ábra: Az „*Keszlet ellenorzes 1*” nevű **Decide** modul dialógusablaka

Normál esetben egy entitás átmenetileg tartózkodik a rendszerben, létrejön és végül a **Dispose** modulon keresztül távozik. A kontrollentitás („*Anyag entitas*”) azonban folyamatosan keringhet a rendszerben, eljátszva az új nyersanyag érkezését, mivel a „*Gyartasi muvelet*” nevű modulban soha nem jelentkezik késés a nyersanyaghiány miatt. Ez a modellezési eszköz logikailag egyenlő az entitás diszponálásával és ismételt létrehozásával. Azonban számítástechnikai szempontból sokkal hatékonyabb, mivel extra számítási erőfeszítéseket takarít meg, és így a szimuláció futása gyorsabb lesz. Ajánlatos összehasonlítani kontrollentitással futó modellt a hagyományos megoldással, a sebességbeli különbség a logikailag korrekt optimalizálásnak köszönhetően érzékelhető lesz. Fontos megjegyezni, ha a gyártási műveletben hiány jelentkezhet, akkor az alkalmazott megoldás nem megengedett. Ilyenkor a szimuláció futása alatt „*Anyag entitas*”-t diszponálni kell, és új entitást kell létrehozni.

6.1.4 Az igénymenedzsment szegmens

Ebben az alfejezetben az igény menedzsment szegmens (*6.2. ábra*) logikai felépítésével foglalkozunk. Az ügyféligény forrása az „*Ugyfel erkezes*” nevű **Create** modul, amelynek dialógus ablak a *6.8. ábrán* látható.



6.8. ábra: „*Ugyfel erkezes*” nevű **Create** modul dialógusablaka

A vevők érkezése véletlenszerű, az érkezési időköz elosztása pedig egyenletes eloszlású, *Unif* (3, 7). Mindenegyes érkezés egy-egy ügyfélentitásnak („*Ugyfel*”) felel meg, amelyekhez saját attribútum halmazt (*Igeny*, *Elvesztett mennyiseg*) rendelünk. Érkezés után az ügyfélentitás először belép az „*Ugyfel igeny*” nevű **Assign** modulba, ahol a véletlenszerű, egyenletes eloszlású *Igeny* attribútumot, amelynek az értéke: *Unif*(50, 100), hozzárendeljük az entitáshoz. A modulban definiált további változókat a *6.2. táblázat* tartalmazza.

6.2. táblázat

Az „*Ugyfel igeny*” nevű **Assign** modul paraméterei

Name	Ugyfel igeny
Type Attribute Name New Value	Attribute Igeny UNIF(50, 100)
Type Variable Name New Value	Variable Pillanatnyi igeny Igeny
Type Variable Name New Value	Variable Osszes ugyfel Osszes ugyfel + 1
Type Variable Name New Value	Variable Osszes igeny Osszes igeny + Igeny

Az ügyfélentitás tovább halad a „*Keszlet ellenorzes 2*” nevű **Decide** modul felé (*6.3. táblázat*), ahol ellenőrizzük, hogy a raktárkészlet elegendő-e az ügyféligény kielégítéséhez. A vizsgálat két kimenetet eredményez:

(1) Ha a *Keszlet* változó értéke nagyobb vagy egyenlő az *Igeny* attribútum értékével, akkor az ügyféligény kielégíthető, és az ügyfélentitás a *True* oldalon hagyja el a **Decide** modult, és lép be „*Igeny kielegites a keszletbol*” nevű **Assign** modulba, ahol a raktárkészletet az ügyfél igényével csökkentjük. Ezt követően az entitás „*Gyartsunk ujra?*” nevű **Decide** modulba lép (*6.4. táblázat*). Itt megvizsgáljuk, hogy a *Keszlet* változó értéke elérte-e *Ujrarendelesi pont* változó értékét. Amikor az ügyfélentitás belép az „*Gyartas indul*” nevű **Assign** modulba, akkor a „*Gyartas*” nevű döntési változó értékét 1-re változtatjuk, ami a ”*Gyartsunk?*” nevű **Hold** modulban feltartott kontrollentitást felszabadítja, azaz valójában folytatódik a termelési eljárás a gyártás. Végül az ügyfélentitás az „*Ugyfel diszponalas*” nevű **Dispose** modulon keresztül elhagyja a rendszert.

6.3. táblázat

A „*Keszlet ellenorzes 2*” nevű **Decide** modul paraméterei

Name	Keszlet ellenorzes 2
Type	2-way by Condition
If	Variable
Value	Keszlet ≥ Igeny

6.4. táblázat

A „*Gyartsunk ujra?*” nevű **Decide** modul paraméterei

Name	Gyartsunk ujra?
Type	2-way by Condition
If	Variable
Value	Keszlet ≤ Ujrarendelesi pont

6.5. táblázat

A „*Veszteseg*” nevű **Assign** modul paraméterei

Name	Veszteseg
Type Variable Name New Value	Variable Elvesztett ugyfel Elvesztett ugyfel + 1
Type Attribute Name New Value	Attribute Elvesztett mennyiseg Igeny – Keszlet
Type Variable Name New Value	Variable Osszes elvesztett mennyiseg Osszes elvesztett mennyiseg + Elvesztett mennyiseg
Type Variable Name New Value	Variable Keszlet 0
Type Variable Name New Value	Variable Gyartas 1

(2) Ha a „*Keszlet ellenorzes 2*” nevű **Decide** modulban a *Keszlet* változó értéke szigorúan kisebb, mint az *Igeny* attribútum értéke, akkor az ügyféligény csak részlegesen vagy egyáltalán nem elégíthető ki. A *False* ágon távozó ügyfélentitás belép az „*Veszteseg*„ nevű **Assign** modulba, ahol a „*Keszlet*” változó értékét 0-ra csökkentjük, frissítjük a többi változó értékét is (*6.5. táblázat*).

Az ügyfélentitás ezután belép a “*Tally elvesztett mennyiseg*” nevű **Record** modulba, ahol az egy ügyfélre eső nem teljesen kielégített igényt („*Elvesztett mennyiseg*”) számláljuk. Végül az ügyfélentitás az „*Ugyfel diszponalas*” **Dispose** modulon keresztül kilép a rendszerből.

Variable - Basic Process

	Name	Rows	Columns	Clear Option	Initial Values	Report Statistics
1	Keszlet			System	1 rows	☐
2	Gyartas			System	1 rows	☐
3	Osszes ugyfel			System	0 rows	☐
4	Elvesztett ugyfel			System	0 rows	☐
5	Batch meret			System	1 rows	☐
6	Ujrarendelesi pont			System	1 rows	☐
7	Max keszlet			System	1 rows	☐
8	Osszes igeny			System	0 rows	☐
9	Fajlagos gyartasi koltseg			System	1 rows	☐
10	Fajlagos keszlettartasi koltseg			System	1 rows	☐
11	Gyartott mennyiseg			System	1 rows	☐
12	Inditasok szama			System	1 rows	☐
13	Fajlagos inditasi koltseg			System	1 rows	☐
14	Ciklusido			System	0 rows	☐
15	Ciklus kezdet			System	0 rows	☐
16	Pillanatnyi igeny			System	0 rows	☐
17	Fajlagos hianykoltseg			System	1 rows	☐
18	Osszes elvesztett mennyiseg			System	0 rows	☐

Double-click here to add a new row.

6.9. ábra: A **Variable** adatmodul párbeszédablaka táblázatnézetben

A modellben fontos szerepet játszó, korábban definiált felhasználói változókat a **Basic Process** panel **Variable** moduljában ellenőrízhetjük, illetve állíthatjuk be. A modulra kattintva megjeleníthető a modul *6.9. ábrán* látható táblázatnézete.

6.1.5 Statisztikák gyűjtése

A termelési-készletezési modell **Statistic** adatmoduljának párbeszédablaka táblázatnézetben a *6.10. ábrán* látható.

Statistic - Advanced Process

	Name	Type	Expression	Report Label
1	Atlagos keszletszint	Time-Persistent	Keszlet	Atlagos keszletszint
2	Folyamat allapot	Frequency		Folyamat allapot
3	Gyartas aktiv	Time-Persistent	Gyartas==1	Gyartas aktiv
4	Elvesztett ugyfel per osszes ugyfel	Output	Elvesztett ugyfel/Osszes ugyfel	Elvesztett ugyfel per osszes ugyfel
5	Atlagos igeny per ugyfel	Output	Osszes igeny/Osszes ugyfel	Atlagos igeny per ugyfel
6	Egysegnyi idore eso gyartasi koltseg per ora	Output	Gyartott mennyiseg*Fajlagos gyartasi koltseg/TFIN	Egysegnyi idore eso gyartasi koltseg per o
7	Egysegnyi idore eso keszlettartasi koltseg per	Time-Persistent	Fajlagos keszlettartasi koltseg*Keszlet	Egysegnyi idore eso keszlettartasi koltseg
8	Egysegnyi idore eso inditasi koltseg per ora	Output	Fajlagos inditasi koltseg*Inditasok szama/TFIN	Egysegnyi idore eso inditasi koltseg per or
9	Egysegnyi idore eso hianykoltseg per ora	Output	Osszes elvesztett mennyiseg*Fajlagos	Egysegnyi idore eso hianykoltseg per ora
10	Egysegnyi idore eso keszletezesi koltseg per	Output	OVALUE(Egysegnyi idore eso inditasi koltseg per	Egysegnyi idore eso keszletezesi koltseg
11	Atlagos ciklus ido ora	Time-Persistent	Ciklusido	Atlagos ciklus ido ora
12	Atlagos igenyszint	Time-Persistent	Pillanatnyi igeny	Atlagos igenyszint
13	Osszes erkezo ugyfel	Output	Osszes ugyfel	Osszes erkezo ugyfel
14	Reszlegesen kiszolgalt ugyfel	Output	Elvesztett ugyfel	Reszlegesen kiszolgalt ugyfel
15	Elvesztett mennyiseg per osszes igeny	Output	Osszes elvesztett mennyiseg/Osszes igeny	Elvesztett mennyiseg per osszes igeny

Double-click here to add a new row.

6.10. ábra: A **Statistic** adatmodul párbeszédablaka táblázatnézetben

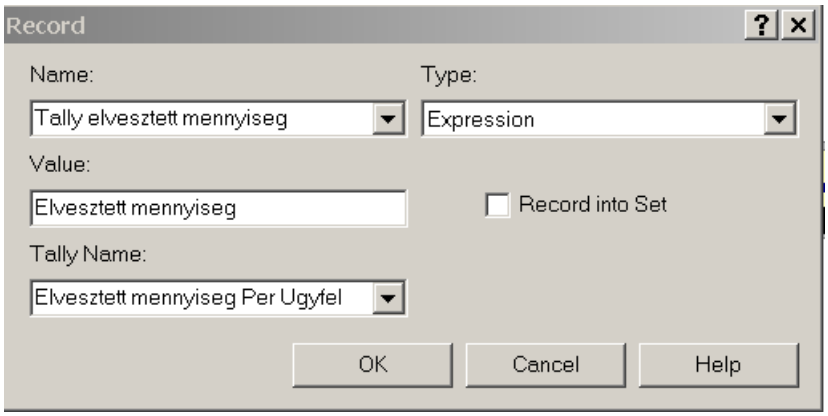
A táblázatban egyebek mellett *Time-Persistent* típusú statisztikák láthatók: („*Atlagos keszletszint*”, „*Gyartas aktiv*”, „*Egysegnyi idore eso keszlettartasi koltseg per ora*”, „*Atlagos ciklus ido ora*”, „*Atlagos igenyszint*”). A készletszintet jellemző statisztikai kimenete 95%-os megbízhatósági szinten tartalmazza a modell futása alatt megfigyelt készletszint átlagos, a minimális és maximális értékeit. A „*Gyartas aktiv*” nevű statisztika annak az időtartamnak a valószínűségét méri, amikor a *Gyartas* nevű változó értéke 1, azaz a gyártás működik. E statisztikának azért van létjogosultsága, mert a gyártás különböző okok miatt leállhat. A készletszinttel arányos egységnyi időre eső készlettartási költség számítható a „*Egysegnyi idore eso keszlettartasi koltseg per ora*” statisztikával. Emlékezzünk arra, hogy ez a *Time-Persistent* eljárás segítségével tetszőleges mennyiség időben átlagolt valószínűségét tudjuk megbecsülni beleértve a közös valószínűséget. A statisztikák gyűjtésére hatással van a *Variable* adatmodul *Report Statistics* oszlopának állapota, amelyben a statisztika gyűjtését be- és kikapcsolhatjuk.

A *6.10. ábrán* látható táblázat tartalmazza a „*Folyamat allapot*” nevű *Frequency* típusú statisztikát is, amely a gyártási folyamat állapotainak a valószínűségét méri, gyakorlatilag a gyártás működik (*busy*), vagy nem működik (*down*) állapotok gyakoriságát. A modellben a tételen (*idle*) állapotot nem kell definiálni.

A „*Elvesztett ugyfel per osszes ugyfel*” *Output* típusú statisztikát (*6.10. ábra*) arra használjuk, hogy kiszámítsuk az összes ügyfélhez viszonyítva azoknak az ügyfeleknek a százalékos arányát, akiknek az igényét nem teljesítettük teljes mértékben. Az *Expression* mezőben ennek megfelelően két változó hányadosa: *Elvesztett ugyfel*/*Osszes ugyfel* szerepel. Ugyancsak *Output* statisztikát használunk a költségkomponensek (egységnyi időre eső indítási, gyártási, készlettartási, hiány költségek), valamint az egységnyi időre eső teljes készletezési költség számítására. A statisztika az összes érkező, a részlegesen kiszolgált ügyfelek számát és az elvesztett mennyiség / összes igény arányát is megjeleníti.

A *6.11. ábra* a „*Tally elveszett mennyiseg*” nevű **Record** modul párbeszédablakát mutatja. Amikor az ügyfél entitás belép ebbe a modulba az *Expression*, esetünkben az „*Elvesztett men-*

nyiseg" nevű attribútum kiértékelődik, amelynek az eredménye számlálás. A modell futásának végén a riportban a Σ*Elvesztett mennyiseg* és a veszteséget szenvedett ügyfelek számának a hányadosa jelenik meg, azaz az átlagos veszteség. Kezeljük óvatosan ezt a statisztikát, mivel számlálás csak akkor történik, amikor egy ügyfelet nem tudtunk teljesen kielégíteni, ezért az átlag (ún. feltételhez kötött átlag) csak azokra az ügyfelekre vonatkozik, akiknél veszteség jelentkezik. Világos, hogy a Σ*Elvesztett mennyiseg*/*Osszes ugyfel* átlag (feltételhez nem kötött átlag) kisebb szám, mint a feltételhez kötött átlag.



6.11. ábra: A „*Tally elveszett mennyiseg*" nevű **Record** modul párbeszédablaka

6.1.6 A szimuláció kimenetei

A *6.12. ábra* szemlélteti a 20.000 óra (több mint két év) hosszúságú szimulációs futás eredményeit. Az egytermékes termelési-készletezési modell működését jellemző megfigyelt eredmények a gyakorlat szempontjából is nagyon tanulságosak.

A *Tally* szekcióban kijelzett *Elvesztett mennyiseg Per Ugyfel* kifejezés szerint a veszteséget szenvedett ügyfelekre vonatkoztatott átlagos veszteség (ki nem elégített mennyiség) 22,1273 termékegység. A *Half Width* oszlopban látható érték (1,47788) a 95%-os konfidencia-intervallum fele.

A *Time Persistent* szekcióban látható *Atlagos keszletszint* változó *Maximum* oszlopából (504,58 egység) az olvasható ki, hogy a pillanatnyi készletszint esetenként meghaladja a célszintet (*S*=500). Ez azonban csak ritkán fordult elő, mivel a *Gyartas* változó (ugyancsak a *Time Persistent* szekcióban) az idő 98,68 %-ban egyenlő volt 1-gyel, vagyis a termelés működött. A készletszint az idő jelentős hányadában alacsony volt, az átlagos készlet 160,54 egység alig magasabb, mint az újrarendelési pont (*s*=150 egység). A *6.13. ábrán* látható, hogy a készletszint változása sokkal nagyobb, mint az 50 és 100 db között változó igényszinté.

Az *Output* típusú *Elvesztett ugyfel per osszes ugyfel* statisztika szerint azoknak az ügyfeleknek az aránya, akik csak részben lettek kielégítve, vagy egyáltalán nem 12,402 %, és az elvesztett mennyiségnek az összes igényhez viszonyított aránya 3,53 %.

A készletezés költségelemei közül az időegységre eső készlettartási költség (321,09 Ft/óra) *Time Persistent* típusú statisztika, az időegységre eső indítási költség (6 Ft/óra), gyártási költség (1459,48 Ft/óra), hiányköltség (4,27 Ft/óra) pedig *Output* típusú. A költségelemek összegeként számított készletezési költség 1790,84 Ft/óra. A költségelemek között a legnagyobb hányadot a gyártási és a készlettartási költségek képviselik. Ezek közül az adott szituációban a gyártási költségre a készletezési politikának nincs hatása, annál inkább a készlettartási költségre, amely az átlagos készletszint függvénye.

Végül, a *Frequencies* szekció mutatja termelési folyamat (BUSY, FAILED és IDLE) állapotainak a valószínűségeit. Ezek rendre, 72,98 % (az erőforrás kihasználtság), 25,98 % (az állásidő valószínűsége) és 1,03 % (a tételen állapot valószínűsége). Vegyük észre, hogy az átlagos működési idő (3,37 óra =202,2 perc) és az átlagos állásidő (1,19 óra =71,4 perc) nagyon közel áll az elméleti értékhez. Az állásidő valószínűsége (25,98 %) egyenlő az állásidő - a ciklushossz várhatóértékének a hányadosával). Emlékeztetőül, a meghibásodások közötti idők exponenciális eloszlásúak 200 perces átlaggal, a javítási időközök normális eloszlásúak 70 perces átlaggal és 30 perces szórással.

19:39:54 **User Specified** november 28, 2010

Egytermékes készletezés Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 20 000,00 Time Units: Hours

Tally

Expression	Average	Half Width	Minimum	Maximum
Elvesztett mennyiseg Per Ugyfel	22.1273	1,47788	0.01385769	75.3473

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Atlagos ciklus ido ora	980.64	(Insufficient)	0	4,140.61
Atlagos igenyszint	75.2076		0	99.98
Atlagos keszletszint	160.54	17,47022	0	504.58
Egysegnyi idore eso keszlettartasi koltseg per ora	321.09	34,94044	0	1,009.17
Gyartas aktiv	0.9868	(Insufficient)	0	1.0000

Output

Output	Value
Atlagos igeny per ugyfel	75.2909
Egysegnyi idore eso gyartasi koltseg per ora	1,459.48
Egysegnyi idore eso hianykoltseg per ora	4.2750
Egysegnyi idore eso inditasi koltseg per ora	6.0000
Egysegnyi idore eso keszletezesi koltseg per ora	1,790.84
Elvesztett mennyiseg per osszes igeny	0.03533718
Elvesztett ugyfel per osszes ugyfel	0.1202
Osszes erkezo ugyfel	4,017.00
Reszlegesen kiszolgalt ugyfel	483.00

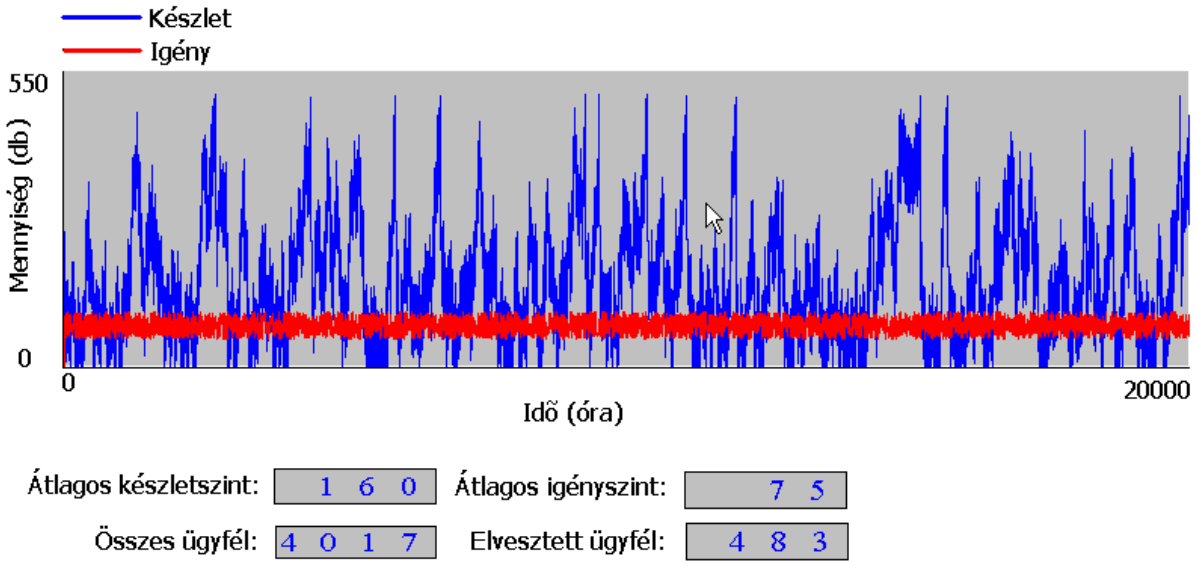
19:29:29 **Frequencies** november 28, 2010

Egytermékes készletezés Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 20 000,00 Time Units: Hours

Folyamat allapot	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	4,332	3.3694	72.98	72,98
FAILED	4,368	1.1897	25.98	25,98
IDLE	60	3.4493	1.03	1,03

6.12. ábra: Az egytermékes termelési-készletezési modell szimulációjának eredményei



6.13. ábra: A készlet- és az igényszint változása az idő függvényében

A *6.13. ábrán* látható statisztikai outputból további információk olvashatók ki, de ezeket nem részletezzük (a szimulációs eredmények analízisének hiányzó részleteit az olvasóra bízzuk).

6.1.7 Modellkísérletek és analízis

Ahogyan korábban említettük az SCM célja egyensúlyt keresni a rendszerköltségek és a vevői elégedettség között. Ismert, hogy maga a szimuláció nem oldja meg ezt a problémát, azonban a bemeneti paramétereket célirányosan változtatva megfigyelhetjük a kimenetek változását. Ugyanakkor előnyt jelent, hogy amíg a klasszikus determinisztikus készletezési modellek csak költség optimalizálásra alkalmasak, addig a szimulációs modellel egyidejűleg vizsgálhatjuk a költségek és a vevői elégedettség változásait. Például közismert, hogy az újrarendelési pont csökkentésétől a készletezési költségek csökkenését várhatjuk. A kérdés természetesen az, hogy meddig csökkenthető az újrarendelési pont anélkül, hogy a vevői elégedettséget jellemző *Elvesztett ugyfel per osszes ugyfel* arány ne romoljon.

6.6. táblázat

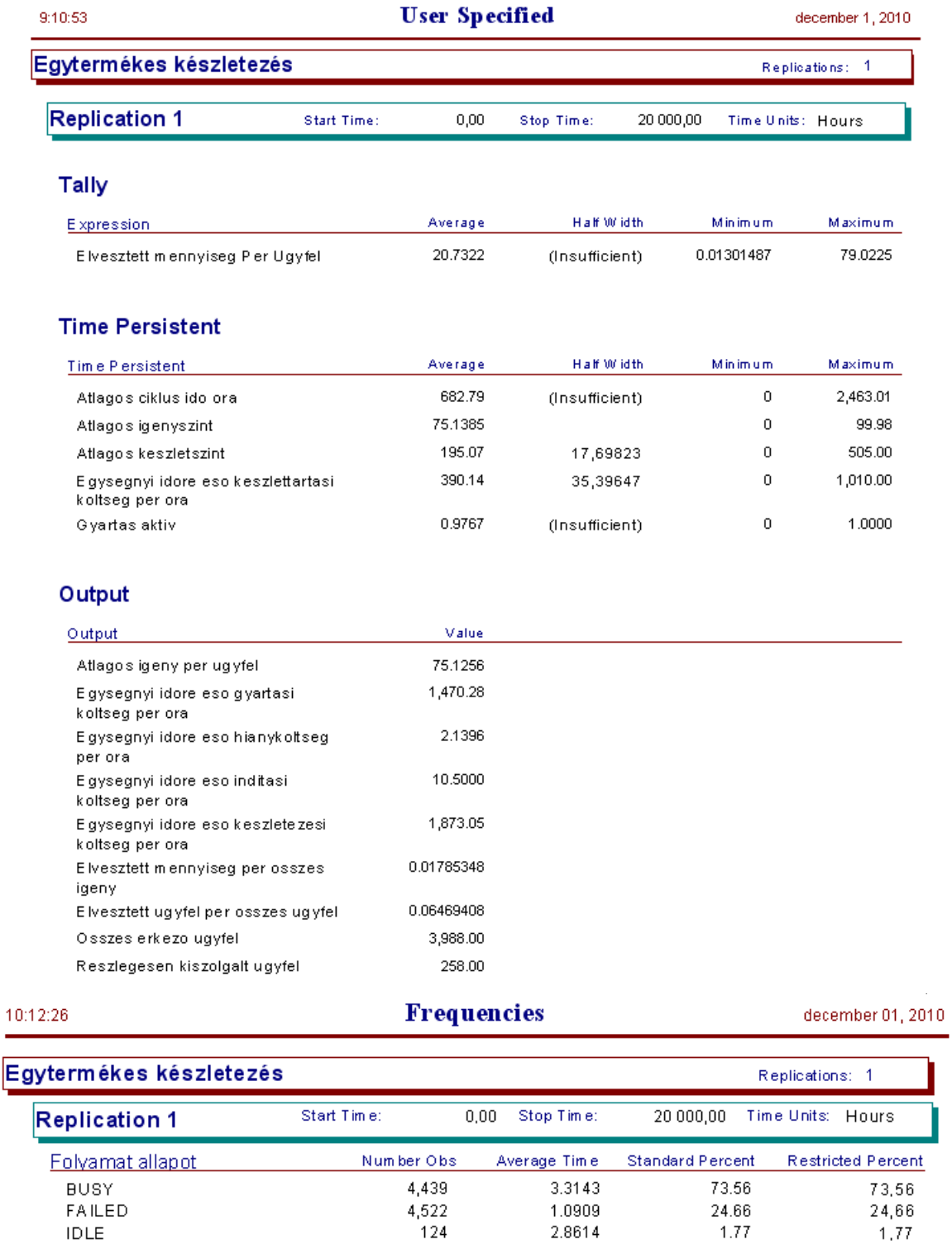
A termelési-készletezési rendszer költséganalízise

Újra-rendelési pont [db]	Készlettartási költség [Ft/óra]	Hiányköltség [Ft/óra]	Indítási költség [Ft/óra]	Készletezési költség [Ft/óra]	Veszteség-arány* [%]	Átlagos ciklusidő [óra]	Gyártás aktív [%]
50	349,87	3,21	6,0	1819,30	9,89	1314,49	98,25
75	288,09	4,20	4,0	1753,72	12,80	1532,36	98,91
100	296,68	3,75	3,5	1763,49	11,63	2477,92	99,02
125	303,74	4,35	4,0	1769,84	12,64	2261,50	98,94
150	321,09	4,27	6,0	1790,84	12,02	980,64	98,68

* Elvesztett ügyfél per összes ügyfél

A kísérletben az újrarendelési pontot 50 és 150 db között változtattuk 25 db-os lépésközzel, és a kapott eredményeket a *6.6. táblázatban* foglaltuk össze. (A vizsgált tartomány felsőhatára az eredeti újrarendelés pont *s*=150 db.) Az újrarendelési pont növelése 50-től 100 db-ig a készletezési költség csökkenését és a gyártókapacitás kihasználtságának a javulást eredményezi, elhanyagolható veszteségarány változás mellett. Megjegyezzük, az eredeti újrarendelési ponthoz tartozó veszteségarány 12,02 % is 11,63 %-ra csökkent, azaz javult a kiszolgálás színvo-

nala. A táblázat alapján az s=100 db nagyságú újrarendelési pont (a táblázat kiemelt sora) kvázi optimumnak tekinthető, mivel s>100 db esetén a költség növekszik, vevőkiszolgálás és a gyártóeszköz kihasználtság pedig romlik. Ebből az egyszerű vizsgálatból azt a következtetést vonhatjuk le, hogy az újrarendelési pont helyes megválasztásával csökkenthetjük a költségeinket és javíthatjuk a vevőkiszolgálás színvonalát.



9:10:53 **User Specified** december 1, 2010

Egytermékes készletezés Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 20 000,00 Time Units: Hours

Tally

Expression	Average	Half Width	Minimum	Maximum
Elvesztett mennyiseg Per Ugyfel	20.7322	(Insufficient)	0.01301487	79.0225

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Atlagos ciklus ido ora	682.79	(Insufficient)	0	2,463.01
Atlagos igenyszint	75.1385		0	99.98
Atlagos keszletszint	195.07	17,69823	0	505.00
Egysegnyi idore eso keszlettartasi koltseg per ora	390.14	35,39647	0	1,010.00
Gyartas aktiv	0.9767	(Insufficient)	0	1.0000

Output

Output	Value
Atlagos igeny per ugyfel	75.1256
Egysegnyi idore eso gyartasi koltseg per ora	1,470.28
Egysegnyi idore eso hianykoltseg per ora	2.1396
Egysegnyi idore eso inditasi koltseg per ora	10.5000
Egysegnyi idore eso keszletezesi koltseg per ora	1,873.05
Elvesztett mennyiseg per osszes igeny	0.01785348
Elvesztett ugyfel per osszes ugyfel	0.06469408
Osszes erkezo ugyfel	3,988.00
Reszlegesen kiszolgalt ugyfel	258.00

10:12:26 **Frequencies** december 01, 2010

Egytermékes készletezés Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 20 000,00 Time Units: Hours

Folyamat allapot	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	4,439	3.3143	73.56	73,56
FAILED	4,522	1.0909	24.66	24,66
IDLE	124	2.8614	1.77	1,77

6.14. ábra: A csökkentett állásidejű egytermékes termelési-készletezési modell szimulációjának eredményei

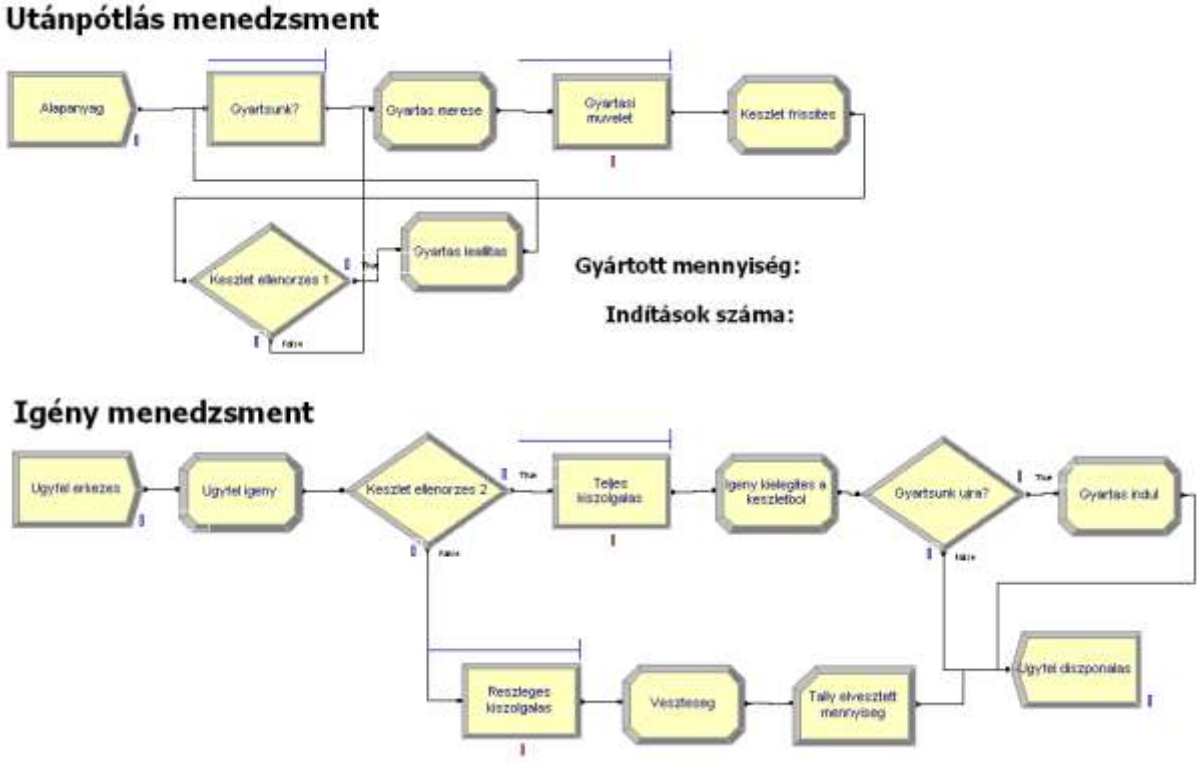
A második kísérletünk a vevőkiszolgálás színvonalának javítására, azaz a teljesen kiszolgált vevői igények valószínűségének a növelésére irányul. Az előző kísérletben megfelelő nagy-

ságúnak talált újrarendelési ponthoz (s=100) tartozó veszteségarány elfogadható (11,63 %), de javításra szorul. A kérdés most, hogyan módosítsuk a rendszert ahhoz, hogy a veszteség valószínűségét elfogadható szintre csökkentsük.

A gyakorlatban, a készletorientált rendszerekben, amilyen az általunk vizsgált rendszer is, a kiszolgálási színvonal javítás egyetlen útjának általában a készletszint növelését tekintik. A *6.6. táblázatból* azonban kiolvasható, hogy az újrarendelési pont növelése a készletezési költség emelkedését eredményezi, ezért a veszteség arány csökkentésére most más utat keresünk. A *6.14. ábrán* a módosított termelési-készletezési modell szimulációs eredményei láthatók, amelyben a normál eloszlású javítási idő várhatóértékét 65 percre és a szórását pedig 25 perc csökkentettük. A karbantartási tevékenység módosításának köszönhető javítási idő csökkenés a meghibásodás valószínűségét 25,98 %-ról 24,66 %-ra csökkenti, ami 5 % körüli változást jelent (lásd a *Frequencies* szekciót a *6.12. ábrák, 6.14. ábrákon*). A részlegesen, illetve teljesen kielégítetlen igények aránya pedig 11,63 %-ról 6,47 %-ra csökken. Ez a csökkenés több mint 44 %-os javulást eredményez a kiszolgálási színvonalban. A bemutatott vizsgálat arra a konklúzióra vezet, hogy a kiszolgálási színvonal szignifikáns javítása inkább a termelési folyamat javításával érhető el, mint az újrarendelési politika módosításával.

6.1.8 Kiszolgálási idővel módosított egytermékes termelési-készletezési rendszer (6.2. modell)

A *6.1. modellben* a rendszerbe érkező ügyfeleket, ha a készletszint megengedte, azonnal, késedelem nélkül kiszolgáltuk, vagyis azt feltételeztük, hogy a kiszolgálási kapacitás végtelen nagy és a kiszolgálási idő elhanyagolható. Sokszor azonban a kiszolgálás időtartama jelentős lehet, amit nem lehet figyelmen kívül hagyni, mivel növeli az ügyfelek rendszerben tartózkodási idejét.



6.15. ábra: Az egytermékes termelési-készletezési rendszer kiszolgálási idővel

A kiszolgálási idő kezelésére átalakított modell folyamatábra nézete a *6.15. ábrán* látható, amely a *6.1. modelltől* annyiban különbözik, hogy az igénymenedzsment szegmensbe két új **Process** modult („*Teljes kiszolgalas*” és „*Reszleges kiszolgalas*”) illesztettünk a „*Keszlet*

ellenorzes 2” nevű **Decide** modul után. E modulok paramétereit a *6.7* és *6.8. táblázatok* tartalmazzák. Mindkét modulban az „*Arukiadó*” nevű erőforrás végzi a kiszolgálást, amelynek a mennyisége 1.

15:48:20 **User Specified** december 1, 2010

Egytermékes készletezés Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 20 000,00 Time Units: Hours

Tally

Expression	Average	Half Width	Minimum	Maximum
Elvesztett mennyiseg Per Ugyfel	8.0241	1,34954	-73	128.32

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Atlagos ciklus ido ora	1,676.66	(Insufficient)	0	6,647.96
Atlagos igenyszint	75.4947		0	99.99
Atlagos keszletszint	140.48	15,49030	-63	504.90
Egysegnyi idore eso keszlettartasi koltseg per ora	280.96	30,98060	-126	1,009.80
Gyartas aktiv	0.9904	(Insufficient)	0	1.0000

Output

Output	Value
Atlagos igeny per ugyfel	75.5100
Egysegnyi idore eso gyartasi koltseg per ora	1,459.23
Egysegnyi idore eso hianykoltseg per ora	3.9446
Egysegnyi idore eso inditasi koltseg per ora	4.0000
Egysegnyi idore eso keszletezesi koltseg per ora	1,748.13
Elvesztett mennyiseg per osszes igeny	0.03267441
Elvesztett ugyfel per osszes ugyfel	0.3075
Osszes elvesztett mennyiseg darab	9,861.57
Osszes erkezo ugyfel	3,997.00
Osszes gyartott mennyiseg darab	291,845.00
Reszlegesen kiszolgalt ugyfel	1,229.00

6.16. ábra: A kiszolgálási idővel kiegészített egytermékes termelési-készletezési modell szimulációjának eredményei

A kiszolgálási idő arányos az igényelt mennyiséggel, ezért a számításához új változót definiáltunk a **Variable** adatmodulban „*Fajlagos kiszolgalasi ido*” néven, amelynek az értéke konstans, 2 perc/db. A kifejezésként (*Expression*) megadott kiszolgálási idő értelemszerűen az ügyfél igényének és a fajlagos kiszolgálási időnek a szorzata (*6.7. táblázat*). A részlegesen kielégített ügyfelek kiszolgálási ideje, tekintettel arra, hogy az igényük csak a pillanatnyi készlet erejéig lesz kielégítve, a készlet és a fajlagos kiszolgálási idő a szorzata (*6.8. táblázat*).

A modell többi input változója megegyezik az eredeti *6.1. modell* változóival, így az eredmények a *6.12. ábrán* közölt eredményekkel hasonlíthatók össze.

6.7. táblázat

A „*Teljes kiszolgalas*” nevű **Process** modul paraméterei

Name Action	Teljes kiszolgalas Seize Delay Release
Resource Type Resource Name Quantity	 Resource Arukiado 1
Delay Type Units Expression	Expression Minutes Igeny * Fajlagos kiszolgalasi ido

6.8. táblázat

A „*Reszleges kiszolgalas*” nevű **Process** modul paraméterei

Name Action	Reszleges kiszolgalas Seize Delay Release
Resource Type Resource Name Quantity	 Resource Arukiado 1
Delay Type Units Expression	Expression Minutes Keszlet * Fajlagos kiszolgalasi ido

A szimuláció futásának eredményei a *6.16. ábrán* látható riportban olvashatók. A leglényegesebb változás, hogy a hosszabb átfutási idő miatt a szimulációs idő alatt kiszolgált ügyfelek száma 4017-ről 3997-re csökkent. Az *Output* típusú *Elvesztett ugyfel per osszes ugyfel* statisztika 12,02 %-ról 30,75 %-ra változott, vagyis a vevő kiszolgálás színvonala jelentős mértékben romlott. Ezen úgy segíthetünk, hogy a kiszolgálási időt az árukiadók számának növelésével csökkentjük, például 1 árukiadó helyett kettőt alkalmazunk (ezt a kísérletet az olvasóra bízzuk). További észrevehető különbség, hogy a készletezési költség 1790,84 Ft/óráról 1748,13 Ft/órára csökkent, ami az alacsonyabb átlagos készletszintnek (140,48 db) köszönhető.

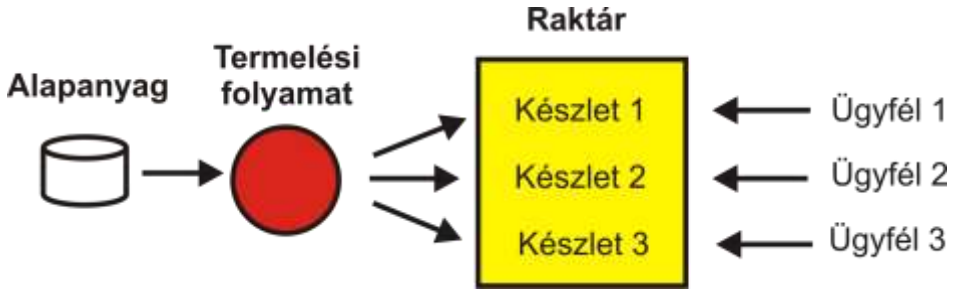
6.2. Többtermékes termelési és készletezési rendszer (6.3. modell)

Ebben a szakaszban továbbfejlesztjük a *6.1 szakaszban* megismert egytermékes termelési és készletezési rendszert többtermékes rendszerré. Az új rendszer is a nem pótolható hiány elvén működik, azaz a kielégítetlen igények elvesznek.

6.2.1 A probléma leírása

A továbbfejlesztett rendszer a *6.17. ábrán* látható. A termelőüzem háromféle (1, 2, és 3) terméket állít elő három megkülönböztetett vonalon érkező ügyfelek ellátására. A terméktípusokat 1, 2, és 3 számokkal jelöljük, illetve különböztetjük meg egymástól. A termelőüzem lotokban meghatározott tételnagyságokat termel, és a készletszinttől függően átáll az egyik terméktípus gyártásáról a másikra. A termelésben az 1 típusú termék rendelkezik a legmagasabb és a 3 típusú a legalacsonyabb prioritással.

A termelési folyamat anyagellátását egy, az ábrán látható, alapanyag-tároló biztosítja, és a végtermékeket a raktárban tárolják. Az ügyfelek a raktárhoz különböző nagyságú igényekkel érkeznek, és ha az igény teljes mértékben nem elégíthető ki, akkor a kielégítetlen hányad veszteséget, pótolhatatlan hiányt jelent. Minden terméktípushoz a *6.9. táblázatba* foglalt egyedi paraméterek tartoznak. A modellezés során a következő feltételezésekkel élünk:



6.17. ábra: Többtermékes termelési-készletezési modell három termékkel

1. A alapanyag tárolóban mindig elegendő anyag áll rendelkezésre, így a termelés anyaghiány miatt soha nem áll le.
2. A termelés lot-okban történik, amelynek a mérete 5 termékegység, és a végtermék lottok a raktárban nyernek elhelyezést. Egy lot előállításának időtartama determinisztikus (*6.9. táblázat*).

6.9. táblázat

A háromtermékes termelési-készletezési rendszer paraméterei

Termék típus	Célszint	Újrarendelési pont	Kezdőkészlet	Műveleti idő [óra]	Igények érkezési időközei [óra]	Igények mennyisége
1	100	50	75	1	Exp(16)	UNIF(4,10)
2	200	100	150	0,6	Exp(8)	UNIF(10,15)
3	300	150	200	0,3	Exp(4)	UNIF(20,30)

A termelési folyamatot véletlen meghibásodás jellemzi, amely akkor jelentkezik, amikor a termelőeszköz működik (*Busy*). A meghibásodási időközök exponenciális eloszlásúak 200 perces középértékkel, a javítási idő normál eloszlású 70 perces középértékkel és 30 perces szórással (megjegyezzük negatív javítási idő is generálódhat, ilyenkor addig generálunk új mintát, amíg nem negatív értéket kapunk).

A raktár (*s*, *S*) készletezési politikát alkalmaz minden terméknél. Megjegyezzük, hogy aktuálisan az erőforrás kapacitása megoszlik a háromféle termék között. Az erőforrás azonban csak akkor rendelhető egy másik termékhez, amikor a termelési folyamat áll. A termékek prioritása következtében az erőforrást a két magasabb prioritású termék többször is lekötheti, mielőtt a 3 típusú termékre sor kerülne. Megjegyezzük, hogy más termékváltási politika is alkalmazható, mint például a pillanatnyi termék gyártásának befejezésekor váltás történik, azaz a pillanatnyi termék gyártása nem folytatódhat.

A következő statisztikákat 100.000 óra szimulációs időre vonatkoztatva szeretnénk megbecsülni:

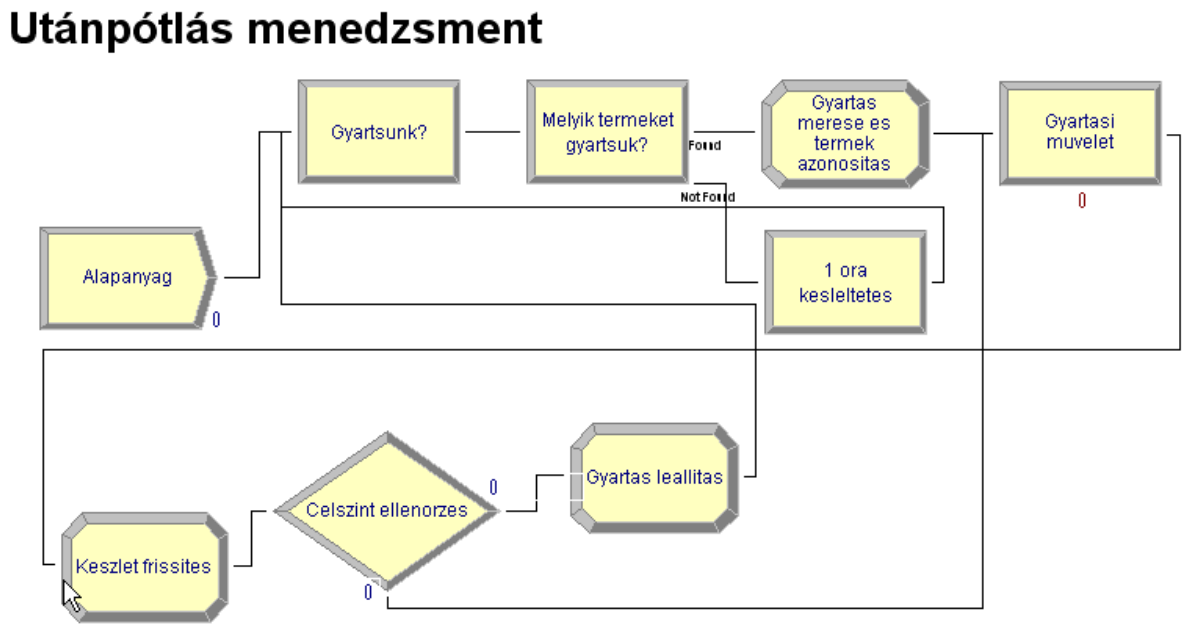
Az erőforrás kihasználását termékenként.
Az erőforrás leállásának valószínűségét.
Az átlagos készletszintet termékenként.
A teljesen ki nem elégített ügyfelek százalékos arányát termékenként.
A teljesen ki nem elégített ügyfelekre vonatkoztatott átlagos veszteség mennyiségét termékenként.

6.2.2 Az Arena modell

A jelenlegi modell a korábban megismert egytermékes modell továbbfejlesztett változata, ezért a felépítése nagyon hasonló a *6.2. ábrán* látható struktúrához. Ennek megfelelően az új modell is két szegmensből az utánpótlás menedzsment és az igénymenedzsment szegmensből épül fel. Az előbbi szegmens úgy módosul, hogy alkalmas több termék megosztott tárolására. A különböző terméktípusok tételeit generáló termékentitás itt is folyamatosan cirkulál a szegmensben, ellenőrzi a készleteket, és amikor szükséges indítja vagy megállítja a gyártást. Az igénymenedzsment szegmens módosítása a különböző típusú ügyfelek és igényeik generálását teszi lehetővé. A szegmens modellezi a készlethiányt és inicializálja a termelést terméktípusonként, amikor az szükséges. A következő szakaszban részletesen megvizsgáljuk az új modell mindkét szegmensének a logikai felépítését.

6.2.3 Az utánpótlás menedzsment szegmens

A termelési (gyártási) tevékenységet modellező módosított utánpótlás menedzsment a *6.18. ábrán* látható.

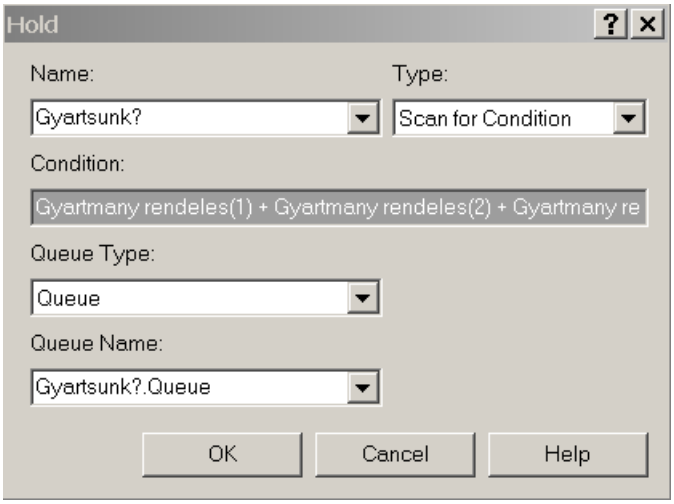


6.18. ábra: Többtermékes termelési-készletezési modell utánpótlás menedzsment szegmense három termékkel

Az „*Alapanyag*” nevű **Create** modul pontosan úgy működik, mint a korábbi modellben, a modulban létrehozott egyetlen entitás vezérli valamennyi terméktípus gyártását. A szegmensben folyamatosan cirkuláló „*Anyag entitas*”-t a „*Gyartsunk?*” nevű **Hold** modul feltartóztatja, és akkor engedi tovább, ha a

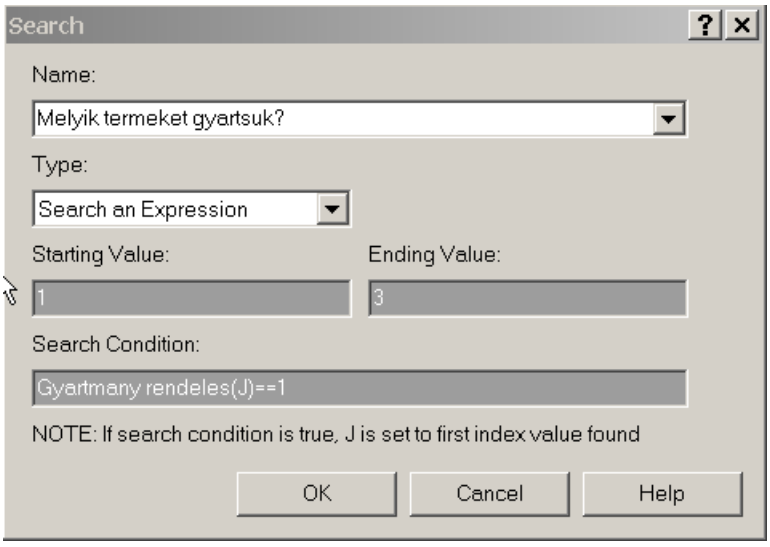
$$Gyartmany\ rendeles(1) + Gyartmany\ rendeles(2) + Gyartmany\ rendeles(3) > 0$$

feltétel teljesül. A feltételt a *6.19. ábrán* látható modulban a *Condition* mezőben írjuk elő. A *Gyartmany rendeles*(k), k = 1,2,3 vektor elemei 1 és 0 értéket vehetnek fel. Az értékek jelzik az adott terméktípus állapotát, azaz, hogy a terméktípus gyártását le kell állítani vagy indítani kell. Pontosabban a *Gyartmany rendeles*(k) = 1 azt jelenti, hogy a készlet az újrarendelési pont alá csökkent és a gyártást indítani kell, *Gyartmany rendeles*(k) = 0 pedig azt, hogy a készlet meghaladta a célszintet és a gyártást le kell állítani. Ha a logikai kifejezés a *Condition* mezőben igaz, akkor az azt jelenti, hogy legalább egy terméktípus gyártására szükség van.



6.19. ábra: A „*Gyartsunk?*” nevű **Hold** modul dialógusablaka

Ha a gyártás aktuálissá vált, akkor arról kell dönteni, hogy melyik terméktípust gyártsuk. A döntés meghozatalához a cirkuláló „*Anyag entitas*” belép a *6.20. ábrán* látható, „*Melyik termeket gyartsuk?*” nevű **Search** modulba. A modulba belépő entitás elindít egy *Search an Expression* típusú keresést. A Search Condition mezőben megadott kifejezés (*Gyartmany rendeles*(*J*)==1) határait és a keresés sorrendjét a Starting Value és az Ending Value mezőkben megadott indexek értékei határozzák meg. A **Search** modul azt a terméktípust fogja választani, amelynél a feltétel elsőként teljesül. Ez azt jelenti, hogy a kisebb indexű terméktípusok prioritást élveznek a nagyobb indexűekkel szemben.



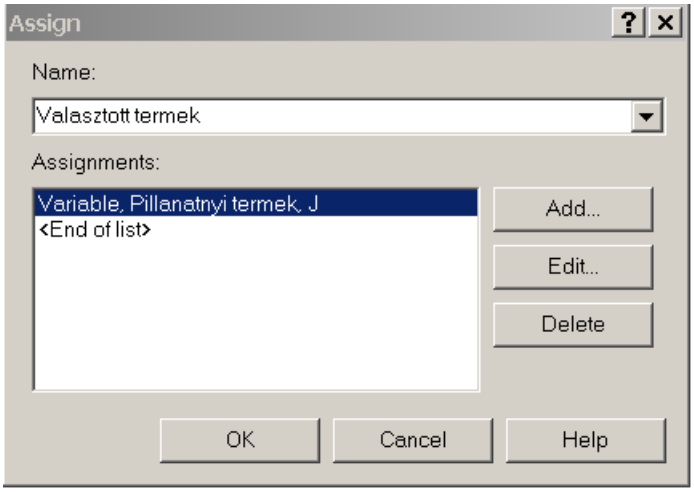
6.20. ábra: A „*Melyik termeket gyartsuk?*” nevű **Search** modul dialógusablaka

A Type mezőben (*6.20. ábra*) többfajta keresés közül választhatunk (*Search a Batch, Search a Queue, Search an Expression*). Például a modellező kereshet egy sorban vagy egy batchben egy entitást, amely valamilyen adott attribútum értékkel rendelkezik, vagy egy kifejezést, amelynek az értéke egyenlő egy adott értékkel. Ha a válaszként kapott **Arena** változó *J* pozitív, akkor a keresés eredményes volt, ha a válasz *J*=0, akkor nem találtunk a feltételnek megfelelő entitást vagy kifejezést.

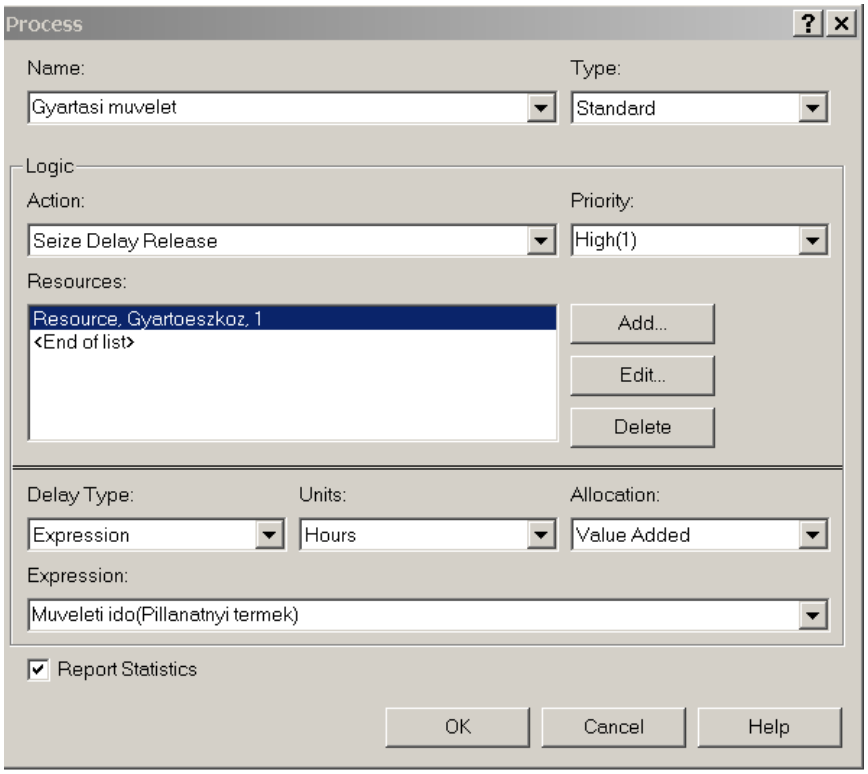
Esetünkben a keresés feltétele *Gyartmany rendeles*(*J*)==1, és a keresés tartománya *k*=1,2, 3. Sikeres keresés eredményeként azt az első **Arena** változót (*J*) kapjuk válaszként, amelynél a *Gyartmany rendeles*(*J*)==1 feltétel teljesül. A *J*= 0 válasz azt jelenti, hogy a keresés eredménytelen volt. Emlékezzünk arra, hogy a probléma leírásban a terméktípusok gyártásának

prioritását úgy definiáltuk, hogy kisebb indexű termékeknek adtunk nagyobb prioritást. Következésképpen a keresésnél először a magasabb prioritású terméktípusokat vizsgáljuk, azaz a kisebb indexszámúakat.

A keresés befejezését követően a cirkuláló entitás belép a „*Valasztott termek*” nevű **Assign** modulba, amelynek a dialógusablakát a *6.21. ábra* szemlélteti. Itt az „*Anyag entitas*” megjegyzi, hogy melyik terméktípust kell gyártani. Ez úgy történik, hogy a „*Pillanatnyi termek*” változó értékéhez hozzárendeljük *J* változó értékét, amely az előző modulban megkeresett terméktípus indexe. Megjegyezzük, a globális tulajdonsággal bíró „*Pillanatnyi termek*” változó használata itt azért megfelelő, mivel az utánpótlás menedzsment szegmensben egyetlen entitás mozog. Ha több entitás keringene a szegmensben, akkor a „*Pillanatnyi termek*” változó helyett az adott entitáshoz rendelt attribútumot kellene definiálni, azért hogy az adott entitás megőrizze a hozzátartozó értéket.

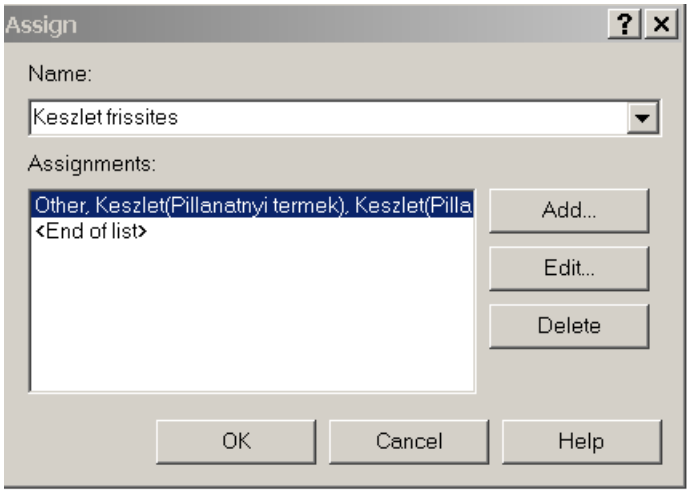


6.21. ábra: A „*Valasztott termek*” nevű **Assign** modul dialógusablaka

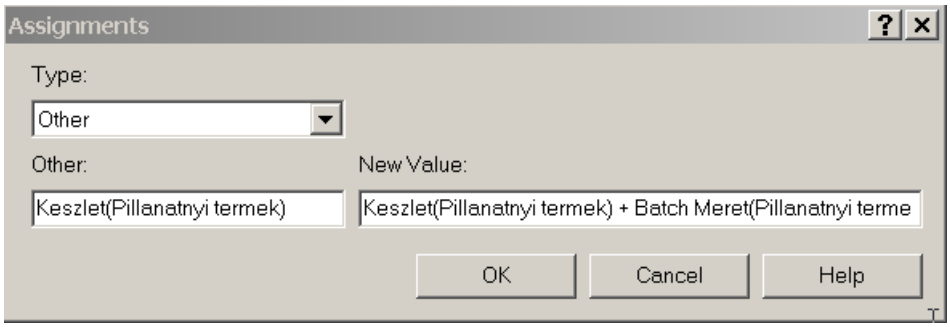


6.22. ábra: A „*Gyartasi muvelet*” nevű **Process** modul dialógusablaka

Ezt követően az „*Anyag entitas*” a „*Gyartasi muvelet*” nevű **Process** modulba és áthalad standard *Seize-Delay-Release* tevékenység sorozaton, amelyek az erőforrás lekötést, a kiválasztott termék gyártását és az erőforrás felszabadítását modellezik. Amíg a *Seize* és *Release* tevékenységek teljesen azonosan működnek, mint az előző modellben, addig a *Delay* tevékenységhez a terméktípustól függő műveleti időt (a *6.9. táblázatban* megadottak szerint) definiáltunk (*6.22. ábra*). A megfelelő műveleti időt a „*Muveleti ido*(*Pillanatnyi termek*)” nevű vektorban tároljuk, amelynek az indexe a terméktípus.



6.23. ábra: A „*Keszlet frissites*” nevű **Assign** modul dialógusablaka



6.24. ábra: A „*Keszlet frissites*” nevű **Assign** modul **Assignments** mezejének dialógusablaka

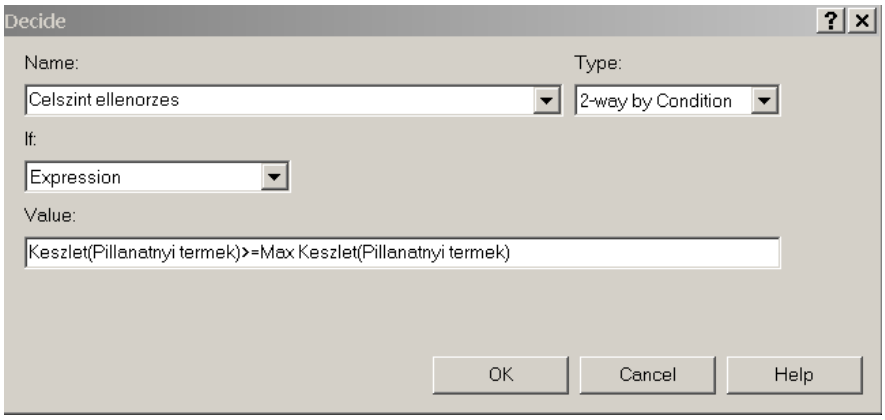
Amikor egy tétel (batch) gyártása befejeződik, akkor a tételt a raktárban kell elhelyezni. Ez úgy valósul meg, hogy az anyagentitás belép a „*Keszlet frissites*” **Assign** modulba, amelynek a dialógusablaka a *6.23. ábrán* látható. A terméktípusok készletszintjeit a „*Keszlet(Pillanatnyi termek)*” nevű vektor tárolja, amelynek az indexe ugyancsak a terméktípus. Hasonló módon, a terméktípusok tételnagyságait a „*Batch Meret(Pillanatnyi termek)*” nevű vektor elemi tartalmazzák. A gyártott terméktípus készletfrissítéséhez a készletet egyszerűen a megnöveljük a terméktípus tételnagyságával. Az *Edit* gombra kattintva (*6.23. ábra*) megjelenik a *6.24. ábrán* látható hozzárendelés:

Keszlet(*Pillanatnyi termek*)= *Keszlet*(*Pillanatnyi termek*) + *Batch Meret*(*Pillanatnyi termek*).

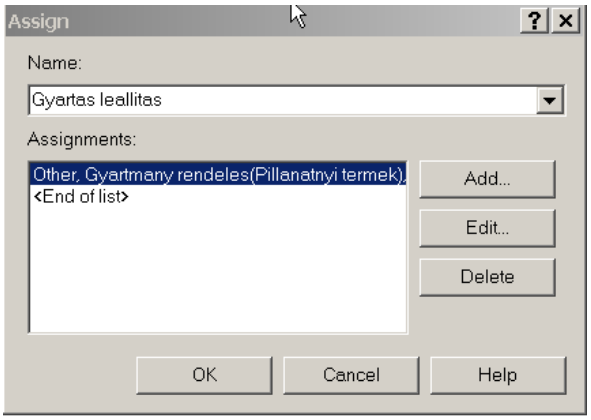
A mező típusa *Other* jelzi, hogy egy vektor elemhez rendelünk értéket. Az **Arena** újabb verzióiban a vektorelemek hozzárendelése más formában is megvalósítható.

Ennél a pontnál az anyagentitásnak ellenőriznie kell, hogy a pillanatnyilag gyártott termék mennyisége elérte-e a maximális készletet jelentő célszintet, amit a „*Celszint ellenorzes*” nevű **Decide** modulban végzünk el (*6.25. ábra*). Megjegyezzük, a terméktípusok célszintjeit „*Max*

Keszlet(*k*)" nevű háromelemű vektorban tároljuk. Az anyagentitás ellenőrzi a pillanatnyi terméktípus célszintjét, és ha az elérte vagy átlépte azt, akkor az anyagentitás a **Decide** modul *True* ágán lép ki a „*Gyartas leallitas*" nevű **Assign** modul felé (*6.26. ábra*). Ebben az anyagentitás leállítja a pillanatnyi terméktípus gyártását olyan módon, hogy a „*Gyartmany rendeles*(*Pillanatnyi termek*)" vektorelemhez 0 értéket rendel.



6.25. ábra: A „*Celszint ellenorzes*" nevű **Decide** modul dialógusablaka



6.26. ábra: A „*Gyartas leallitas*" nevű **Assign** modul dialógusablaka

Ezután az anyagentitás visszatér a „*Gyartsunk?*" nevű **Hold** modulba, hogy ismét meghatározza, melyik terméktípust kell a következő ciklusban gyártani.

Ha azonban a pillanatnyi termék célszintjét nem értük el, akkor az anyagentitás a „*Celszint ellenorzes*" nevű **Decide** modul *False* ágán lép ki és visszatér a „*Gyartasi muvelet*" nevű **Process** modulhoz (*6.18. ábra*) és terméktípus váltás nélkül elkezdődik a pillanatnyi termék újabb tételének a gyártása.

Végül térjünk vissza a gyártóeszköz (erőforrás) tapasztalat szerinti meghibásodásához, amely csak *busy* állapotban fordulhat elő. Ez a megszorítás táblázatnézetben szerkeszthető **Failure** modul dialógustáblájában specifikálható (*6.27. ábra*). Itt az Uptime in this State only mezőben a *BUSY* azt jelzi, hogy a gyártóeszköz csak akkor hibásodik meg, ha működik. Ezzel szemben, ha a mezőt üresen hagyjuk, akkor a meghibásodás bármikor bekövetkezhet, akkor is, ha tétlen, mivel a meghibásodási folyamat folytonos.

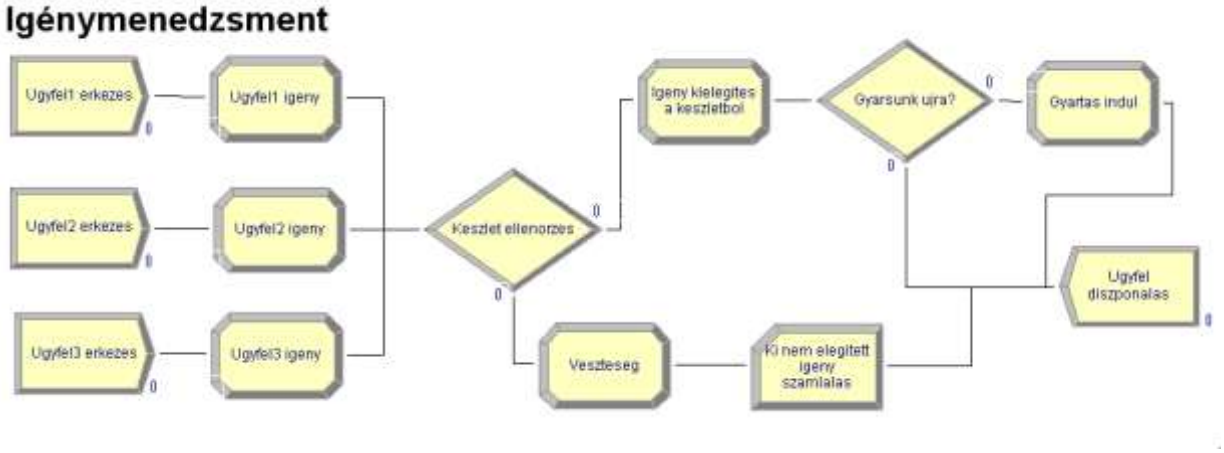
Failure - Advanced Process

	Name	Type	Up Time	Up Time Units	Down Time	Down Time Units	Uptime in this State only
1	Veletlen meghibasodas	Time	EXPO(200)	Minutes	NORM(70, 30)	Minutes	BUSY
	Double-click here to add a new row.						

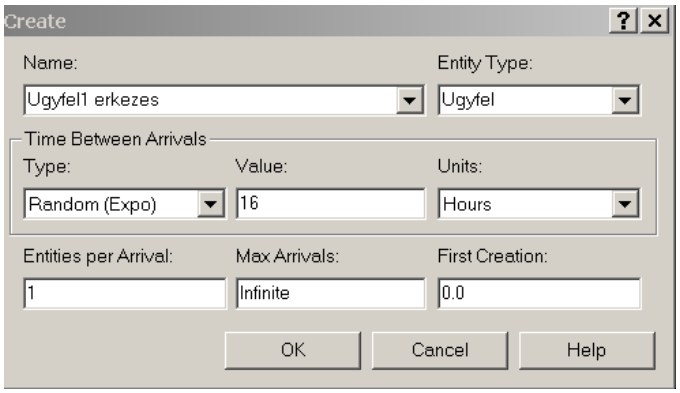
6.27. ábra: A **Failure** modul dialógustáblája

6.2.4 Az igénymenedzsment szegmens

A módosított igénymenedzsment szegmens folyamatábráját a *6.28. ábra* mutatja be, amely az igények érkezését modellezi a termelési és készletezési rendszerben. Ebben a modellben az érkezési folyamatot három **Create** modul reprezentálja az ábra baloldalán.



6.28. ábra: Többtermékes termelési-készletezési modell igénymenedzsment szegmense három termékkel



6.29. ábra: Az „*Ugyfel1 erkezes*” nevű **Create** modul dialógusablaka

Az első **Create** modul dialógus ablaka a *6.29. ábrán* látható, amely az első terméktípusra igényt tartó „*Ugyfel*” entitást generálja. A további **Create** modulok paraméteri a *6.10.* és a *6.11. táblázatokban* találhatók.

6.10. táblázat

Az „*Ugyfel2 erkezes*” nevű **Create** modul paraméterei

Name	*Ugyfel2 erkezes*
Entity Type	Ugyfel
Type Value Units	Random(Expo) 8 Hours

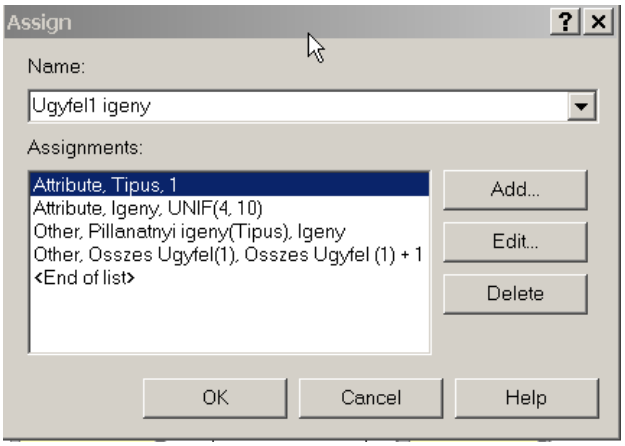
Hasonlóan, a három terméktípus igénylésének folyamatát is három **Assign** modul modellezi, ahol az igényelt terméktípust és az igényelt mennyiséget attribútumaként rendeljük az „*Ugyfel*” entitáshoz. Az attribútumok nevei: „*Tipus*” illetve „*Igeny*”. Az „*Ugyfel1 igeny*” nevű **Assign** modul a *6.30. ábrán* látható, amely az ügyféligényt rendel az 1 jelű terméktípushoz. Ebben a modulban a terméktípust az ügyfélentitás „*Tipus*”, és az igény nagyságát az ügyfélen-

titás „*Igeny*” nevű attribútumaihoz rendeljük. Az igény nagysága egyenletes eloszlású, a *6.9. táblázatban* megadott paraméterekkel.

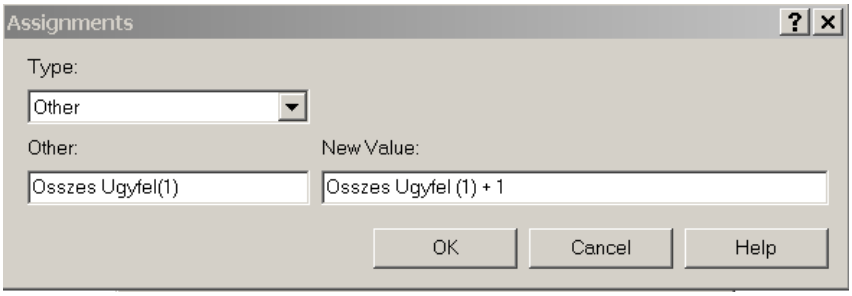
6.11. táblázat

Az „*Ugyfel2 erkezes*” nevű **Create** modul paraméterei

Name	*Ugyfel2 erkezes*
Entity Type	Ugyfel
Type Value Units	Random(Expo) 8 Hours



6.30. ábra: A „*Ugyfel1 igeny*” nevű **Assign** modul dialógusablaka



6.31. ábra: Az „*Osszes Ugyfel*(*k*)” vektort deklarálása az **Assignments** dialógusablakban

6.13. táblázat

Az „*Ugyfel2 igeny*” nevű **Assign** modul paraméterei

Name	Ugyfel2 igeny
Type Attribute Name New Value	Attribute Tipus 1
Type Attribute Name New Value	Attribute Igeny UNIF(10, 15)
Type Other Name New Value	Other Pillanatnyi igeny(Tipus) Igeny
Type Variable Name New Value	Other Osszes Ugyfel (2) Osszes Ugyfel (2) + 1

Ezenkívül, a modell az „*Osszes Ugyfel*(*k*)” nevű vektorban ügyféltípusonként nyilvántartja, illetve számlálja az összes érkező ügyfelet. A megfelelő vektorelemet az érkezés alkalmával 1-el növeli. A harmadik hozzárendelés részletei a *6.31. ábrán* láthatók, ahol „*Osszes Ugyfel*(1)” nevű vektorelem értékét 1-gyel növeljük. Itt a *Type* mezőben ismét az *Other* opciót használtuk az új **Arena** verziókban használható *Variable Array* (1*D*) opció helyett.

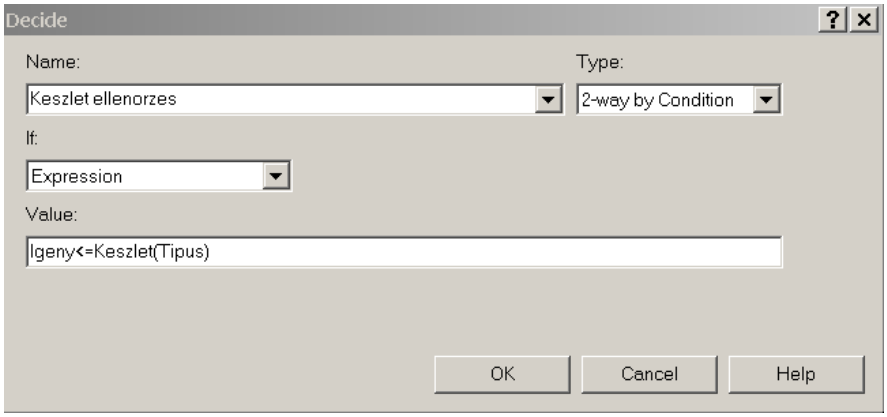
Az „*Ugyfel2 igeny*” é s az *Ugyfel3 igeny* nevű **Assign** modulok paramétereit a *6.13.* és *6.14. táblázatok* tartalmazzák.

Ezután az ügyfélentitás ellenőrzi, hogy a raktárban rendelkezésre áll-e az igényelt terméktípusból az igényelt mennyiség, azaz az ügyfél igénye kielégíthető-e. Az ellenőrzés a „*Keszlet ellenorzes*” nevű **Decide** modulban történik, amelynek a párbeszédablaka a *6.32. ábrán* látható.

6.14. táblázat

Az „*Ugyfel3 igeny*” nevű **Assign** modul paraméterei

Name	Ugyfel3 igeny
Type Attribute Name New Value	Attribute Tipus 1
Type Attribute Name New Value	Attribute Igeny UNIF(20, 30)
Type Other Name New Value	Other Pillanatnyi igeny(Tipus) Igeny
Type Variable Name New Value	Other Osszes Ugyfel (3) Osszes Ugyfel (3) + 1

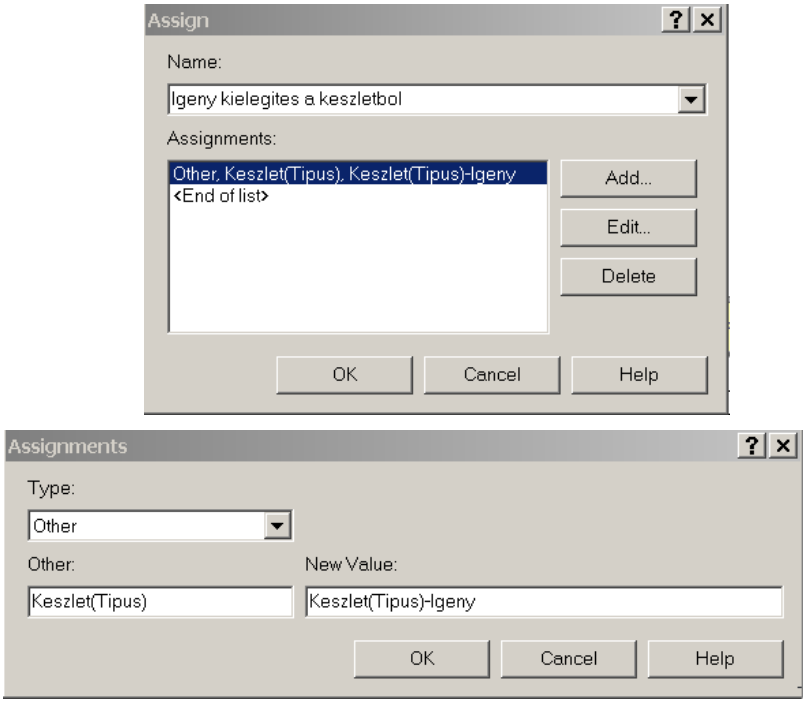


6.32. ábra: A „*Keszlet ellenorzes*” nevű **Decide** modul dialógusablaka

A vizsgálat két lehetséges eredménye: (1) a készletszint nem elegendően nagy a beérkezett igény kielégítéséhez, (2) a rendelkezésre álló készlet elegendően nagy. A továbbiakban leírjuk a két lehetséges esetből levezethető eredményt.

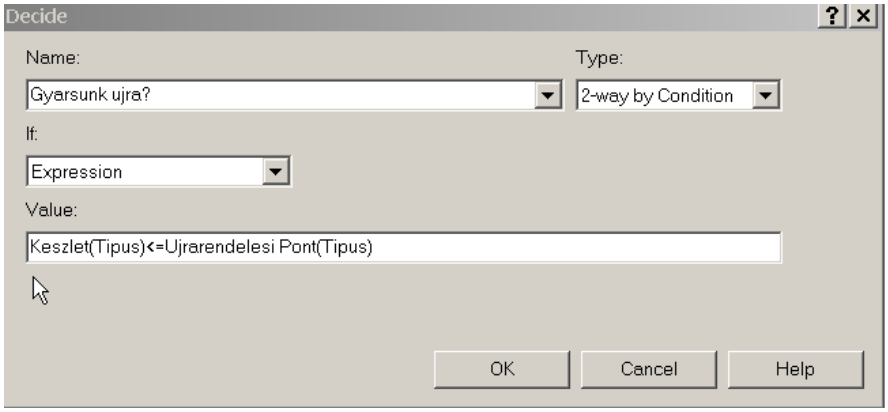
Tekintsük először azt az esetet, amikor a raktári készlet elegendő. Az elegendő termékmennyiség elérhetőségének feltétele a *Value* mezőben látható, ahol vegyük észre, hogy a terméktípusra vonatkozó információt az ügyfélentitás „*Tipus*” nevű attribútumában tároljuk. Ha a raktárban a készlet elegendő nagyságú, akkor a pillanatnyi ügyfél igénye ki lesz elégítve. Ebben az esetben az ügyfélentitás a *True* ágon hagyja el a modult és belép a „*Igeny kielegites a*

keszletbol” nevű **Assign** modulba, ahol a megfelelő készlettípus mennyiségét csökkentjük az igénnyel. Ennek a modulnak a párbeszédablaka a *6.33. ábra* felső részén látható, amelyhez az *Edit* gombbal megnyitható, az ábra alsó részén elhelyezkedő **Assignments** dialógusablak kapcsolódik.



6.33. ábra: Az „*Igeny kielegites a keszletbol*” nevű **Assign** modul dialógusablaka

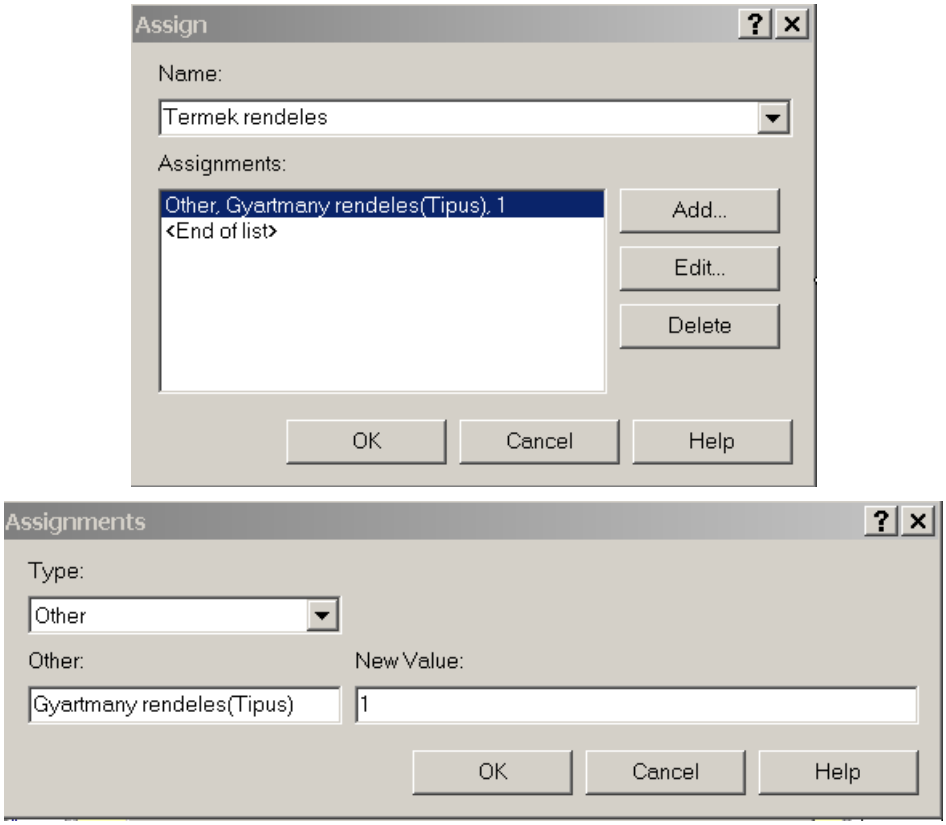
Az ügyfélentitás következő állomása a „*Gyarsunk ujra?*” nevű **Decide** modul, ahol megvizsgáljuk, hogy a készletszint az újrarendelési pont alá csökkent-e, amihez a *6.34. ábrán* a *Value* mezőben olvasható logikai kifejezést használjuk. Vegyük észre, hogy a *Value* mezőben a logikai kifejezés két vektorelem értékét, az igényelt terméktípus készletszintjét és a terméktípus újrarendelési pontját hasonlítja össze.



6.34. ábra: A „*Gyarsunk ujra?*” nevű **Decide** modul dialógusablaka

A *True* ágon távozó ügyfélentitás a „*Termek rendeles*” nevű **Assign** modulba lép (*6.35. ábra*), ahol inicializálja a „*Gyartmany rendeles*(*Tipus*)” vektorelemet (*6.35. ábra* alsó fele). A „*Gyartmany rendeles*(*Tipus*)” inicializálása egyszerűen azt jelenti, hogy a vektorelemhez 1 értéket rendelünk, amely azt jelzi, hogy a terméktípus készlete az újrarendelési pont alá csökkent, azaz hiányos. Az inicializálás eredményeként a terméktípus iránti gyártási igény időben jelezve lesz a gyártás felé, amely a prioritási sorrendnek megfelelően gondoskodik a terméktípus készletének a pótlásáról. Megjegyezzük, a vektorelemekhez a hiányperiódus alatt több-

ször is hozzárendelhetjük az 1 értéket. Mivel ezek a hozzárendelések redundánsak (új információt nem tartalmaznak), ezért ártalmatlanok, a modell állapotának korrekt kezelését nem befolyásolják. Végül a leírt műveletek végrehajtása után az ügyfélentitás belép az „*Ugyfel diszponalas*” nevű **Dispose** modulba és elhagyja a modellt (*6.28. ábra*).



6.35. ábra: A „*Termek rendeles*” nevű **Assign** modul dialógusablaka

A „*Gyarsunk ujra?*” nevű **Decide** modul *False* ágán kilépő ügyfélentitás közvetlen a **Dispose** modulba lép és távozik a szegmensből (*6.28. ábra*).

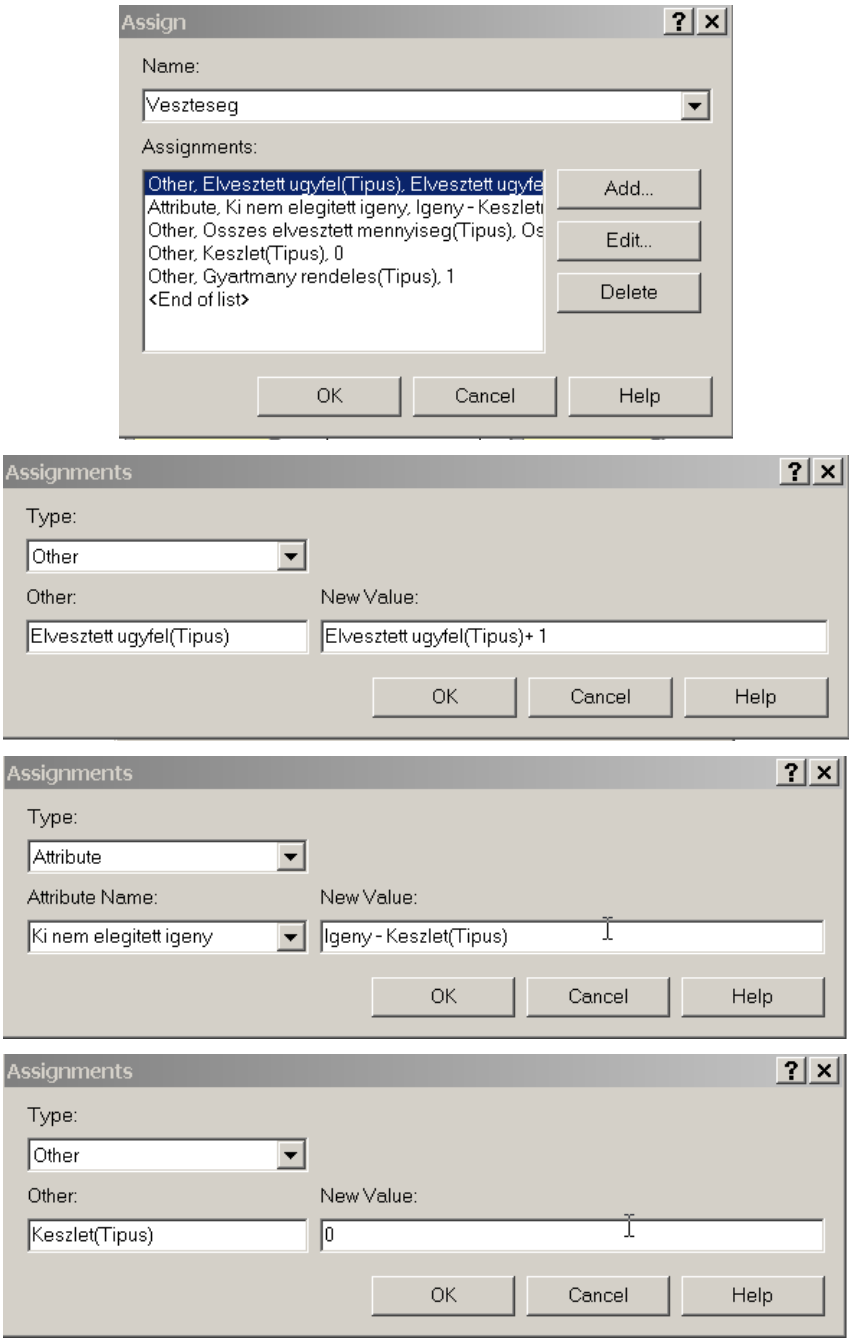
Most tekintsük azt az esetet, amikor a raktárkészlet („*Keszlet*(*Tipus*)”) nem elegendő az ügyféligény kielégítéséhez, és ami azt eredményezi, hogy az ügyfél igényének egy hányada vagy a teljes igény elveszik, ami nemcsak az ügyfélnek, hanem a termelőnek is veszteséget jelent. Az ügyfél entitás ekkor a A „*Keszlet ellenorzes*” nevű **Decide** modul *False* ágán lép ki, és belép a „*Veszteseg*” nevű, öt hozzárendelést tartalmazó **Assign** modulba, amelynek a dialógus ablakát a *6.36 ábra* szemlélteti. Az első hozzárendelés a „*Elvesztett ugyfel(Tipus)*” vektorelem értékét növeli 1-gyel, azaz terméktípusonként számlálja a ki nem elégített ügyfeleket. A második hozzárendelés az elvesztett mennyiséget:

Igeny–Keszlet(*Tipus*)

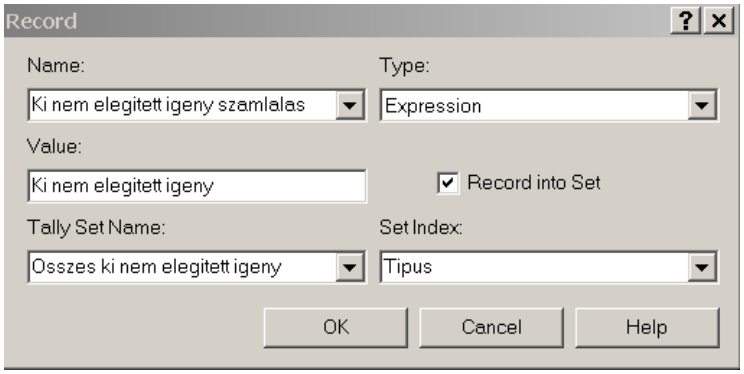
rendeli az ügyfélentitás „*Ki nem elegitett igeny*” nevű attribútumához. A hozzárendelésből az is kiolvasható, hogy az ügyfél a *Keszlet*(*Tipus*) vektorelem értékének megfelelő mennyiséggel távozik. A harmadik hozzárendelés az összes elvesztett mennyiséget kumulálja:

Osszes elvesztett mennyiseg(*Tipus*)=*Osszes elvesztett mennyiseg*(*Tipus*)+*Ki nem elegitett igeny*.

A negyedik hozzárendelés a terméktípus készletszintjét 0-ra csökkenti, és végül az ötödik hozzárendelés a „*Gyartmany rendeles*(*Tipus*)” vektorelemhez 1 értéket rendel, amely azt jelzi, hogy a terméktípus készlete 0-ra, az újrarendelési pont alá csökkent. Megjegyezzük, hogy értelemszerűen a hozzárendelések sorrendje nem változtatható meg.



6.36. ábra: A „*Veszteseg*” nevű **Assign** modul dialógusablaka



6.37. ábra: A „*Ki nem elegitett igeny szamlalas*” nevű **Record** modul dialógusablaka

Az ügyfélentitás ezután „*Ki nem elegitett igeny szamlalas*” nevű **Record** modulba lép be (*6.37. ábra*), ahol a *Value* mezőben megadott „*Ki nem elegitett igeny*” nevű attribútum értékét ügyféltípusonként kumuláljuk a *Tally Set Name* mezőben megadott „*Osszes ki nem elegitett igeny*” nevű háromelemű halmazban. Végül az ügyfélentitás a „*Ugyfel diszponalas*” nevű **Dispose** modulon keresztül távozik a szegmensből.

6.2.5 A modell inputparaméterei és statisztikái

Ebben a szakaszban a modell inputparamétereit és statisztikáit ismertetjük részletesen. Emlékezzünk arra, hogy az összes inputparamétert, beleértve a vektorokat is a **Basic Process** panelen elérhető **Variable** adatmodulban deklarálhatjuk.

A **Variable** adatmodul táblázatnézetben a *6.38. ábrán* látható a „*Keszlet*” változó kezdeti értékeivel. A változók dimenzióját (az elemek számát) az *Initial Values* mezőben a sorok száma jelzi. Például az első sorban a 3 *rows* azt jelenti, hogy a „*Keszlet*” nevű változó egy háromdimenziós vektor. A 0 *rows* azt jelzi, hogy a változónak nincs kezdeti értéke, az 1 *rows* pedig azt jelenti, hogy a változó egy közönséges 1 dimenziós változó. Az *n rows*, (*n*>1) *n* dimenziós vektorváltozót definiál. A *Clear Option* mezőben a *System* azt jelenti, hogy a szimuláció megszakításakor vagy az ismétlések kezdetekor a rendszer visszaállítja a kezdeti értékeket. Az egydimenziós változók statisztikáinak opcionális gyűjtését teszi elérhetővé a *Report Statistics* oszlopban látható négyzet bekapcsolása. A vektorváltozók statisztikáinak gyűjtésére a **Statistic** adatmodul alkalmas, amit később bemutatunk.

Variable - Basic Process

	Name	Rows	Columns	Clear Option	Initial Values	Report Statistics
1	Keszlet	3		System	3 rows	☐
2	Gyartas			System	1 rows	☐
3	Max Keszlet	3		System	0 rows	☐
4	Ujrarendelesi Pont	3		System	3 rows	☐
5	Batch Meret	3		System	1 rows	☐
6	Pillanatnyi termek			System	1 rows	☐
7	Muveleti ido	3		System	3 rows	☐
8	Gyartmany rendeles	3		System	0 rows	☐
9	Osszes Ugyfel	3		System	0 rows	☐
10	Veszteseg	3		System	0 rows	☐

Double-click here to add a new row.

Initial Values

1	75
2	150
3	200

6.38. ábra: A **Variable** adatmodul táblázatnézetben a „*Keszlet*” nevű vektorváltozó kezdeti értékeivel

Set - Basic Process

	Name	Type	Members
1	Osszes ki nem elegitett igeny	Tally	3 rows

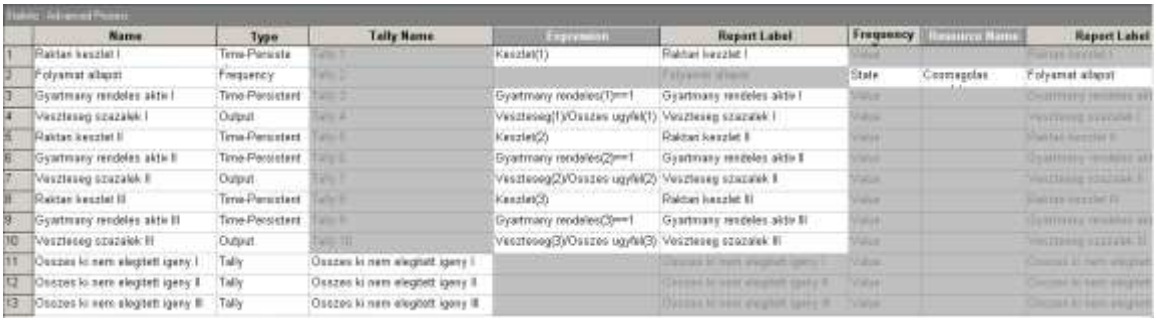
Double-click here to add a new row.

Members

	Tally Name
1	Osszes ki nem elegitett igeny I
2	Osszes ki nem elegitett igeny II
3	Osszes ki nem elegitett igeny III

6.39. ábra: A **Set** (fent) és a kapcsolódó Members (lent) adatmodulok táblázatnézetben a Tally típusú „*Osszes ki nem elegitett igeny*” statisztikához

Emlékezzünk vissza, hogy a „*Ki nem elegitett igeny szamlalas*” nevű **Record** modulban a terméktípusonkénti (1, 2 és 3) összes ki nem elégített igény követésére *Set* (halmaz) típusú változót használtunk. Ehhez a háromdimenziós, „*Osszes ki nem elegitett igeny*” nevű *Set* típusú adatokat a **Basic Process** panelen található **Set** adatmodulban deklaráltuk (*6.39. ábra*).



	Name	Type	Tally Name	Expression	Report Label	Frequency	Resource Name	Report Label
1	Raktari keszlet I	Time-Persists		Keszlet(1)	Raktari keszlet I			
2	Folyamat allapot	Frequency			Folyamat allapot	State	Cosmagolas	Folyamat allapot
3	Gyartmany rendeles aktiv I	Time-Persistent		Gyartmany rendeles(1)==1	Gyartmany rendeles aktiv I			
4	Veszteseg szazalek I	Output		Veszteseg(1)/Osszes ugyfel(1)	Veszteseg szazalek I			
5	Raktari keszlet II	Time-Persistent		Keszlet(2)	Raktari keszlet II			
6	Gyartmany rendeles aktiv II	Time-Persistent		Gyartmany rendeles(2)==1	Gyartmany rendeles aktiv II			
7	Veszteseg szazalek II	Output		Veszteseg(2)/Osszes ugyfel(2)	Veszteseg szazalek II			
8	Raktari keszlet III	Time-Persistent		Keszlet(3)	Raktari keszlet III			
9	Gyartmany rendeles aktiv III	Time-Persistent		Gyartmany rendeles(3)==1	Gyartmany rendeles aktiv III			
10	Veszteseg szazalek III	Output		Veszteseg(3)/Osszes ugyfel(3)	Veszteseg szazalek III			
11	Osszes ki nem elegitett igeny I	Tally	Osszes ki nem elegitett igeny I					
12	Osszes ki nem elegitett igeny II	Tally	Osszes ki nem elegitett igeny II					
13	Osszes ki nem elegitett igeny III	Tally	Osszes ki nem elegitett igeny III					

6.40. ábra: A **Statistic** adatmodul táblázatnézetben a felhasználói statisztikák gyűjteményével

Amint már tudjuk, specifikus, a felhasználó által definiált statisztikai gyűjtemény a **Statistic** adatmodulban hozható létre (*6.40. ábra*). Itt újdonságot jelent, ezért jegyezzük meg, hogy a vektorváltozók statisztikáit elemenként külön-külön kell megadni. A terméktípusonkénti „*Raktari keszlet I, II* és *III*” *Time-Persistent* típusú statisztikák az időben átlagos raktárkészletet jellemzik, ugyanakkor a „*Folyamat allapot” Frequency* típusú statisztika. A „*Veszteseg szazalek I, II,* és *III*” a szimuláció végén számított *Output* típusú statisztika. Végül a „*Osszes ki nem elegitett igeny I, II és III*” a ki nem elégített ügyfelek számára vetített átlagot mutató *Tally* típusú statisztikák.

6.2.6. A szimuláció eredményei

A *6.41. ábra* a 100.000 óra hosszúságú szimuláció eredményeit mutatja. A **Frequencies** riportot vizsgálva a „*Beszerzes muvelet*” nevű erőforrás állapotának statisztikáit. Az erőforrás az idő 64%-ban volt *Busy*, azaz működött, 14%-ban volt *Idle*, azaz várakozott és 22%-ban volt *Failed*, azaz meghibásodás miatt állt.

Megjegyezzük, ha az erőforrás hibája („*Veletlen meghibasodas*”) a bármely állapotban előfordulhatna, nemcsak a *Busy* állapotban, akkor a javítási idő (*Failed* állapot) valószínűsége növekedne, és az *Idle* állapot valószínűsége csökkenne. A *Busy* állapot valószínűsége nem változna, mivel az erőforrás terhelése nem változik.

A terméktípusonkénti várakozási idő, amit a „*Gyartmany rendeles aktiv*” statisztika jellemez, a terméktípus prioritásától függően változik. A várakozási időhányad (várakozási idő/az összes idő) az 1 típusnál 20%, a 2 típusnál 38% és a 3 típusnál 63%. Amint az várható volt, a legkisebb prioritású terméktípus várakozott a legtöbbet, mivel a „*Beszerzes muvelet*” nevű erőforrást ritkábban tudta lekötni, mint a nagyobb prioritású termékek.

A terméktípusonkénti „*Raktári készlet*” mutatja, hogy az átlagos készletszint minden terméktípusnál az újrarendelési ponthoz közeli, ami azt jelzi, hogy az igények csak az erőforrás folyamatosan működésével elégíthetők ki. Ez nagyban a meghibásodásoknak köszönhető, illetve a *Failed* állapot nagy valószínűségének. A jelenséget alátámasztja az is, hogy minimum készletszint minden terméktípusnál 0, amiből arra következtethetünk, hogy időnként hiány jelentkezik. A készlethiány nagyságát terméktípusonként az Output szekcióban a „*Veszteseg szazalek*”, a mennyiségét pedig az Other szekcióban az „*Osszes ki nem elegitett igeny*” jellemzi.

13:23:14 **Frequencies** január 04, 2011

Többtermékes termelési Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 100 000,00 Time Units: Hours

Folyamat allapot	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	19,206	3.2950	63.28	63,28
FAILED	18,322	1.1762	21.55	21,55
IDLE	1,180	12.8522	15.17	15,17

13:21:31 **User Specified** január 4, 2011

Többtermékes termelési Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 100 000,00 Time Units: Hours

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Gyartmany rendeles aktiv I	0.1973	0,008653471	0	1.0000
Gyartmany rendeles aktiv II	0.3711	0,011958004	0	1.0000
Gyartmany rendeles aktiv III	0.6287	0,009541857	0	1.0000
Raktari keszlet I	73.9237		0	105.00
Raktari keszlet II	137.73	1,46919	0	205.00
Raktari keszlet III	180.41	2,53026	0	305.00

Output

Output	Value
Veszteseg szazalek I	0.00095801
Veszteseg szazalek II	0.00905231
Veszteseg szazalek III	0.05890094

Other

None	Average	Half Width	Minimum	Maximum
Osszes ki nem elegitett igeny I	5.0939	(Insufficient)	2.9405	9.2637
Osszes ki nem elegitett igeny II	9.8118	(Insufficient)	0.9008	14.9991
Osszes ki nem elegitett igeny III	19.2961		0.02894693	29.9972

6.41. ábra: A többtermékes termelési-készletezési rendszer szimulációjának eredményei

Az előző példához hasonlóan, a rendszer működése többféle módon javítható. Ismét felhívjuk a figyelmet a karbantartás fontosságára, és a javítási idő csökkentésére. Végül, hogy elkerüljük az alacsony prioritású terméktípus túlzott mellőzését, a terméktípusok közötti váltásokat korlátok előírásával szabályozhatjuk.

6.3. Többlépcsős ellátási-lánc (6.4. modell)

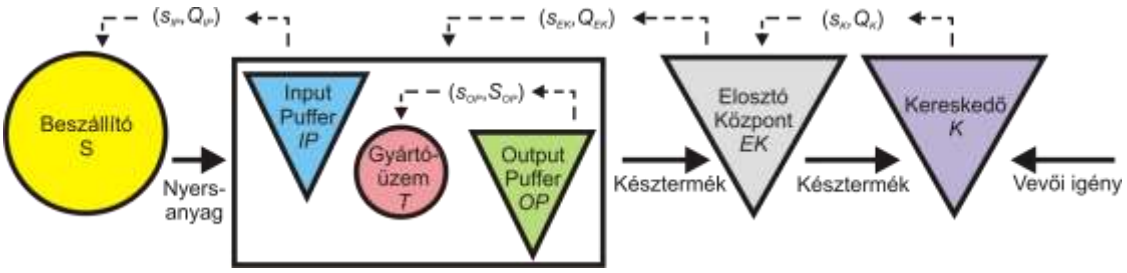
Ebben a részben kiterjesztjük a *6.1.* részben megismert termelési-készletezési rendszert azáltal, hogy több ellátási lépcsőt adunk hozzá. Speciálisan ebben a példában egy négylépcsős ellátási-láncot tanulmányozunk, amely egy beszállítóból, egy gyártóüzemből, egy elosztó-

központból és kereskedőből áll. A rendszer úgynevezett installációs készletezési politikákat használ (az utántöltési politika csak a helyi ellátási installációk információira alapozott). Megjegyezzük, az ilyen politikák csak ún. készletpozíció-adatokat használnak (a rendelkezésre álló készlet, megrendelt készlet és utánrendelés). Ezzel szemben az úgynevezett lépcsős-politikák további információt igényelnek készletpozíció formájában az aktuális installációról, és minden lépcsőről az áramlás irányában.

6.3.1. A probléma ismertetése

Tekintsünk egy egytermékes többlépcsős ellátási-láncot, ami egy kiskereskedőből (*K*), egy elosztó-központból (*EK*), egy gyártóüzemből (*T*), és egy beszállítóból (*S*) áll. A gyártóüzem két egymással összefüggő puffere: az input puffer (*IP*), amely az érkező nyersanyagot, és az output puffer (*OP*), amely a kimenő késztermékeket tárolja. A rendszert vázlatosan a *6.42. ábra* szemlélteti.

A kereskedő a véletlenszerűen érkező vevők (ügyfelek) igényeit elégíti ki, és folyamatos készletfigyelési (s_K, Q_K) politikát alkalmaz, amely a kereskedőnél keletkező információra alapozott. Ismeretes, ez a politika azt jelenti, hogy amikor a készletpozíció (*rendelkezésre álló készlet* + a *megrendelt készlet – az utánrendelés*) s_K szintre vagy az alá csökken, akkor kezdődik az utántöltés, azaz az elosztó-központból Q_K mennyiséget rendelünk. Ha az elosztó-központ elérhető készlete elegendő az igény kielégítéséhez, akkor az utánpótlási idő csak a szállítási időt tartalmazza. Különben az elosztó-központ készlethiányának és a további szállítási késedelemnek köszönhetően várakozás is jelentkezik. A kereskedőnél minden olyan igény, amit nem lehet azonnal a pillanatnyi készletből kielégíteni, nem pótolható, azaz veszteséget jelent.



6.42. ábra: Többlépcsős ellátási-lánc rendszer

Az elosztó-központban jelentkező igények elsődlegesen a kereskedő megrendelései (Q_K). Azonban itt, a kereskedelemtől eltérően, a ki nem elégített igények nem vesznek el, hanem utánrendeléssel teljesülnek. Az elosztó-központ, hasonló módon, a gyártóüzem (*T*) output pufferéből (*OP*) tölti fel a készletét, és ugyancsak folyamatos készletfigyelési politikát (s_{EK}, Q_{EK},) alkalmaz. A ki nem elégített rendelések a gyártóüzemben is utánrendelések lesznek.

6.15. táblázat

A modell készletezési paraméterei

Input puffer	Output puffer	Elosztó központ	Kereskedő
s_{IP} =10	s_{OP} = 10	s_{EK} = 10	s_K= 5
Q_{IP}= 13	S_{OP} = 30	Q_{EK} = 20	Q_K = 10

A gyártóüzem termelési politikája is folyamatos készletfigyelésű (s_{OP}, S_{OP}) politika. Az üzem egyidejűleg egy termékegységet gyárt, és ehhez az input pufferből (*IP*) 1 egységnyi nyersanyag mennyiséget igényel. Megjegyezzük, a nyersanyaghiány az input pufferben a gyártás leállását okozza. Az input puffer viszont egy külső beszállítótól rendel, akinél soha nem jelentkezik készlethiány, így a gyártóüzem rendelése (Q_{IP}) mindig teljesen ki lesz elégítve és a

nyersanyag utánpótlási idő a szállítási idővel egyenlő. Az alkalmazott készletezési politika folyamatos készletfigyelés (s_{IP}, Q_{IP}). A modell készletezési paramétereit a *6.15. táblázat* foglalja össze.

A rendszerben a következő feltételezésekkel élünk:

1. A kereskedőhöz érkező igények a *Poisson* folyamatot követik. Az igényelt mennyiség minden érkezés alkalmával egy darab késztermék, és minden kielégítetlen igény veszteség, azaz pótolhatatlan hiány.

2. A modellezés rugalmassága és általánosíthatósága érdekében feltételezzük, hogy késztermék gyártási idejének az eloszlása és szállítási idejének eloszlása *Erlang* típusú. Pontosabban, minden szállítási késedelem az $Erl(k = 2, \lambda = 1)$ eloszlással írható le, amíg a gyártási idő eloszlása $Erl(k=3, \lambda=5)$.

3. Az ellátási-lánc minden lépcsőjében a rendeléseket olyan sorrendben fogadják, ahogy azokat feladták (nem előzhetik meg egymást). Gyakorlatilag egy rendelés teljesítése akkor kezdődik, amikor az előző rendelés megérkezett a célállomásra.

4. Amikor hiány keletkezik az elosztó központban (*EK*) vagy a gyártóüzemben (*T*), akkor a kielégítetlen rendelés hányad utánrendeléssé válik. Ilyenkor a rendelés teljesítése (szállítása) addig késlekedik (halasztódik), amíg a teljes rendelés mennyiség elérhetővé nem lesz. Röviden, részrendeléseket nem szállítanak.

A modell segítségével a következő működési jellemzőket szeretnénk meghatározni:

A hosszú időre vonatkoztatott, átlagos készletszint a rendszerben.
Az utánrendelések átlagos száma az elosztó központban és az output pufferben.
Az ügyfél kiszolgálási szint minden lépcsőben

Ezeknek a működési jellemzőknek az alakulása segíti a modellezőt a készletezési politikák paramétereinek helyes megválasztásában.

6.3.2 Az Arena modell

A tanulmányozott rendszer sokkal komplexebb annál, hogy azt az előző két mintapélda másolatainak tekinthetnénk. Következésképen a rendszer **Arena** modellje 5 szegmensből áll, és ezek mindegyike egy-egy rendszerlépcső készlet pufferéhez kötődik. Minden ilyen pufferhez a következő események tartoznak: rendelésérkezés, készletfrissítés, a feltöltés indítása és a rendelés szállítása (teljesítése). További szállítási-lánc tevékenységeket modellezünk a szállítási-lánc végpontjain, az igényérkezést az áramlás felső szakaszán és a termék előállítást az áramlás alsó szakaszán. A *6.43. ábra* az **Arena** modell változóit mutatja.

A következőkben részletesen leírjuk a modell szegmenseket, visszafelé haladva, a szállítási-lánc legfelső lépcsőjével kezdünk és innen haladunk a legalsó lépcső felé.

6.3.3 Készletmenedzsment a kereskedőnél szegmens

A kereskedelem készletmenedzsment szegmensének **Arena** modelljét a *6.44. ábra* szemlélteti. Ez a szegmens generálja az igények áramlását, kezeli az igények kielégítését és az elosztóközpont felé indított megrendeléseket.

A *Poisson*-eloszlás szerint érkező ügyfelek igénye mindig egy egységnyi mennyiség, amit az „*Ugyfel igeny a kereskedonel*” nevű **Create** modulban generálunk. Érkezés után az entitás először belép a „*Tally Kereskedo igeny*” nevű **Record** modulba.

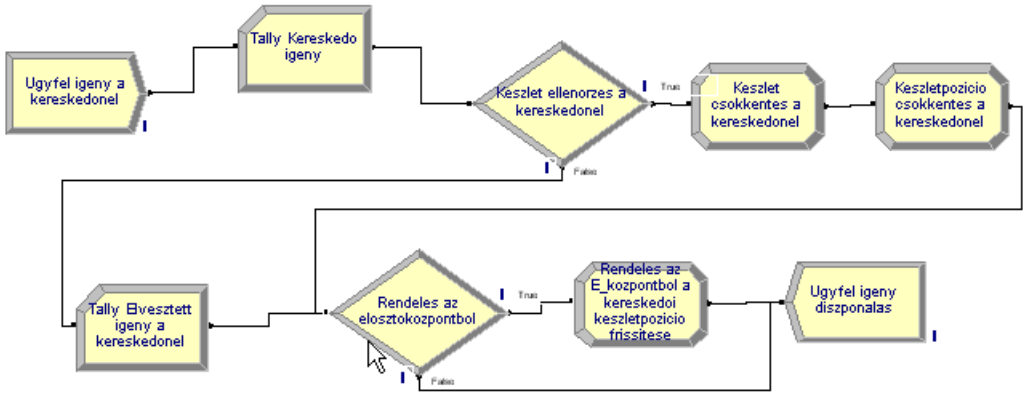
Az ügyfél entitás („*Ugyfel Entity*”) tovább halad a **Decide** modulba (“*Keszlet ellenorzes a kereskedonel*”), ahol megvizsgáljuk, hogy a kereskedő készletéből kiszolgálható-e az ügyfél igénye. A **Decide** modul dialógusablaka a *6.45. ábrán* látható.

Variable - Basic Process

	Name	Rows	Columns	Clear Option	Initial Values	Report Statistics
1	Keszlet_Kereskedo			System	1 rows	
2	Keszlet_EKozpont			System	1 rows	
3	Q_Kereskedo			System	1 rows	
4	s_EKozpont			System	1 rows	
5	Rendeles_OutputPuffer			System	0 rows	
6	Keszlet_OutputPuffer			System	1 rows	
7	Gyartas_Uzem			System	0 rows	
8	KeszletPozicio_InputPuffer			System	1 rows	
9	Keszlet_InputPuffer			System	1 rows	
10	KeszletPozicio_Kereskedo			System	1 rows	
11	s_Kereskedo			System	1 rows	
12	Rendeles_EKozpont			System	0 rows	
13	KeszletPozicio_EKozpont			System	1 rows	
14	Q_EKozpont			System	1 rows	
15	Rendeles_Beszallito			System	0 rows	
16	s_InputPuffer			System	1 rows	
17	Q_InputPuffer			System	1 rows	
18	UtanRendeles_EKozpont			System	0 rows	
19	ElerhetoUtanrendeles_EKozpont			System	0 rows	
20	BigS_Uzem			System	1 rows	
21	s_Uzem			System	1 rows	
22	UtanRendeles_OutputPuffer			System	0 rows	
23	TelKiellgeny_Kereskedo			System	0 rows	
24	TelKiellgeny_EKozpont			System	0 rows	
25	TelKiellgeny_OutputPuffer			System	0 rows	
26	TelKiellgeny_InputPuffer			System	0 rows	
27	ElerhetoUtanrendeles_OutputPuffer			System	0 rows	

Double-click here to add a new row.

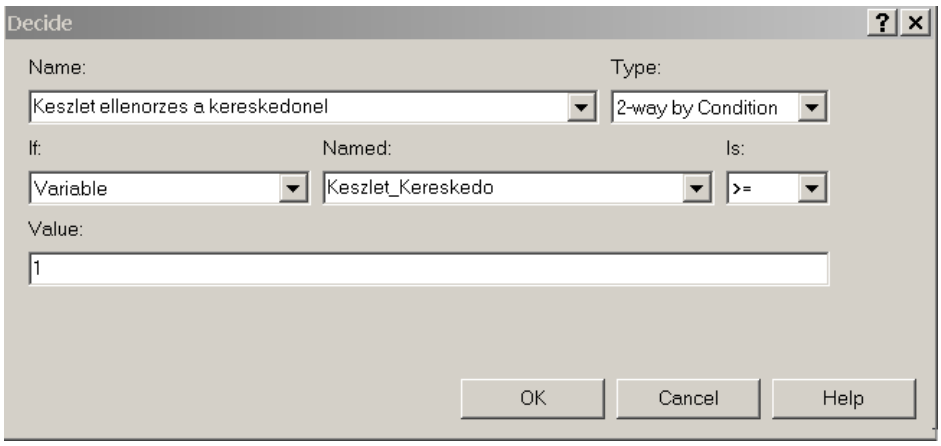
6.43. ábra: A **Variable** adatmodul dialógusablaka táblázatnézetben



6.44. ábra: Készletmenedzsment a kereskedőnél szegmens modellje

A vizsgálatnak két lehetséges kimente van. Az egyik, amikor a feltétel, *Keszlet_Kereskedo*>= 1 igaz, akkor az ügyfél entitás a *True* ágon lép ki a „*Keszlet csokkentes a kereskedonel*” nevű **Assign** modul felé, az utóbbi modulban a kereskedő készletét csökkentjük 1-el, majd az entitás belép a következő „*Keszletpozicio csokkentes a kereskedonel*” nevű **Assign** modulba, ahol a „*KeszletPozicio_Kereskedo*” változó értékét ugyancsak 1-el csökkentjük. A másik kimenet, ha a *Keszlet_Kereskedo* = 0, akkor az ügyfél entitás a *False* ágon lép ki, és a ki nem elégíthető

igényhányad elveszik, Ebben az esetben az ügyfél entitás tovább halad a „*Tally Elvesztett igeny a kereskedonel*” nevű **Record** modulba, ahol az elvesztett igényt és a teljesen ki nem elégített ügyfeleket megszámláljuk.



6.45. ábra: A “*Keszlet ellenorzes a kereskedonel*” nevű **Decide** modul dialógusablaka

Megjegyezzük, a két **Assign** modul a készletszint-információkat tároló két változó értékét tartja karban. A „*Keszlet_Kereskedo*” nevű változó az elérhető készletet tárolja, amely 0 és s_K+Q_K között változik. A „*KeszletPozicio_Kereskedo*” változóban pedig az s_K és s_K+Q_K között változó készletpozíciót tároljuk. Az elsőt arra használjuk, hogy az elérhető készletből kielégítsük az ügyfelek igényét, az utóbbit pedig arra, hogy elindítsuk a megrendeléseket a feltöltés érdekében. Emlékezzünk arra, hogy amikor az készletpozíció eléri az újrarendelési szintet (s_K), vagy az alá csökken, akkor indítjuk a feltöltést, azaz az elosztó-központból Q_K mennyiséget rendelünk. Ekkor a készletpozíció azonnal a Q_K mennyiséggel nő.

A két kimenet „*Rendeles az E_kozpontbol a kereskedoi keszletpozicio frissitese*” nevű **Decide** modulnál találkozik, ahol az ügyfél entitás ellenőrzi, hogy az „*KeszletPozicio_ Kereskedo*” változó elérte-e az újrarendelési pontot:

KeszletPozicio_Kereskedo <= *s_Kereskedo*.

Ha igen, akkor az ügyfél entitás belép a „*Rendeles az E_kozpontbol a kereskedoi keszletpozicio frissitese*” nevű **Assign** modulba, amely két hozzárendelés hajt végre. Az első hozzárendelés *Rendeles_EKozpont* = 1, amely azonnal felszabadítja a „*Készletmenedzsment az elosztó-központban*” elnevezésű szegmens (*6.46. ábra*) „*Indulhat a kereskedoi rendeles?*” nevű **Hold** moduljában várakozó „*Kereskedoi Rendeles Entity*„ entitást. A második hozzárendelés:

KeszletPozicio_Kereskedo = *KeszletPozicio_Kereskedo* + *Q_Kereskedo*,

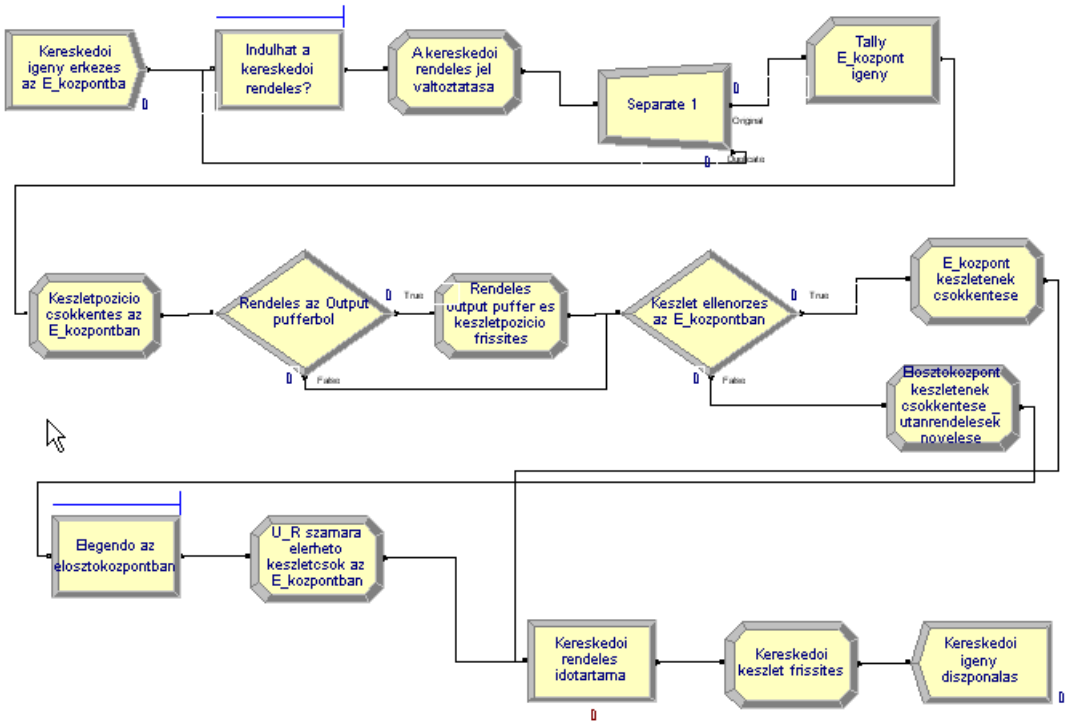
amely azonnal frissíti a kereskedő készletpozícióját. A *False* ágon kilépő ügyfél entitás „*Ugyfel igeny diszponalas*” nevű **Dispose** modulon keresztül elhagyja a rendszert.

6.3.4 Készletmenedzsment az elosztó-központban szegmens

A készletmenedzsment az elosztó-központban szegmens **Arena** modelljét *6.46. ábra* szemlélteti. Ez a modellszegmens generálja az elosztó-központba érkező kereskedői megrendeléseket, frissíti az elosztó-központban a készletszintet, indítja a feltöltést a gyártóüzem output pufferéből, szállítmányokat küld a kereskedőhöz, és frissíti a kereskedő készletszintjét.

A „*Kereskedoi igeny erkezes az E_kozpontba*” nevű **Create** modul létrehoz egy „*Kereskedoi Rendeles Entity*” nevű rendelésentitást 0 időpontban, amely később az elosztó-központból a kereskedő felé irányuló szállításokat generálja. A rendelésentitás először belép a **Hold** modulba („*Indulhat a kereskedoi rendeles?*”), ahol addig várakozik, amíg a

„*Rendeles_EKozpont*” nevű döntési változó értéke 1 nem lesz. A tovább haladó rendelésentitás a „*A kereskedoi rendeles jel valtoztatasa*” nevű **Assign** modulban a „*Rendeles_EKozpont*” változó értékét 0-ra állítja. Ezt követően az entitás belép a *Separate 1* nevű **Separate** modulba, ahol önmagát duplikálja. Az eredeti rendelésentitás folytatja az útját a rendszerben, az entitás másolat viszont egy hurok mentén visszatér a „*Indulhat a kereskedoi rendeles?*” nevű **Hold** modulba, ahol a következő kereskedői rendelést generálja.



6.46. ábra: Készletmenedzsment az elosztó-központban szegmens modellje

A „*Tally E_kozpont igeny*” nevű **Record** modulban megszámláljuk a kereskedői rendelésentitásokat, amelyek ezt követően belépnek a „*Keszletpozicio csokkentes az E_kozpontban*” nevű **Assign** modulba, ahol a „*KeszletPozicio_EKozpont*” változó értéket csökkentjük a „*Q_Kereskedo*” változó értékével. A kereskedői rendelésentitások áramlásának következő állomása a „*Rendeles az Output pufferbol*” nevű **Decide** modul, ahol megvizsgáljuk a *KeszletPozicio_EKozpont* <= *s_EKozpont* feltétel teljesülését, azaz az elosztó-központ készletszintje elérte-e az újrarendelési pontot. Ha a vizsgálat eredménye igen, akkor az entitások a „*Rendeles output puffer es keszletpozicio frissites*” nevű **Assign** modul felé folytatják az útjukat, amelyben két értékadás történik. Az első értékadás: *Rendeles_OutputPuffer*=1, amely a készletmenedzsment az output pufferben szegmens „*Indulhat az E_kozpont rendeles?*” nevű **Hold** moduljában (*6.49. ábra*) pillanatnyilag várakozó entitást felszabadítja. A második értékadás az elosztó-központ készletpozícióját tartalmazó változó értékét növeli az elosztó-központ megrendelésével:

$$KeszletPozicio\_EKozpont = KeszletPozicio\_EKozpont + Q\_EKozpont.$$

A rendelésentitás ezután belép a „*Keszlet ellenorzes az E_kozpontban*” nevű **Decide** modulba, ahol azt vizsgáljuk, hogy a kereskedő készlete elegendő-e. A vizsgálat két lehetséges kimenete: (1) Ha a „*Keszlet_EKozpont*”>=„*Q_Kereskedo*„ feltétel igaz, akkor az entitás a *True* ágon keresztül belép a „*E_kozpont keszletenek csokkentese*” nevű **Assign** modulba, ahol a készletet a kerekedői megrendeléssel csökkentjük:

$$Keszlet\_EKozpont = Keszlet\_EKozpont – Q\_Kereskedo.$$

(2) Ha viszont a „*Keszlet_EKozpont*”<„*Q_Kereskedo*” feltétel teljesül, akkor az entitás a *False* ágon távozik. Ebben az esetben az igény nem elégíthető ki, és a gyártó telepre utánrendelést kell kezdeményezni. Ennek eléréséhez a rendelésentitás belép a „*Elosztokozpont keszletenek csokkentese_utanrendelesek novelese*” nevű **Assign** modulba, ahol három hozzárendelést hajtunk végre:

1. Az utánrendelési szintet növeljük a hiány nagyságával:

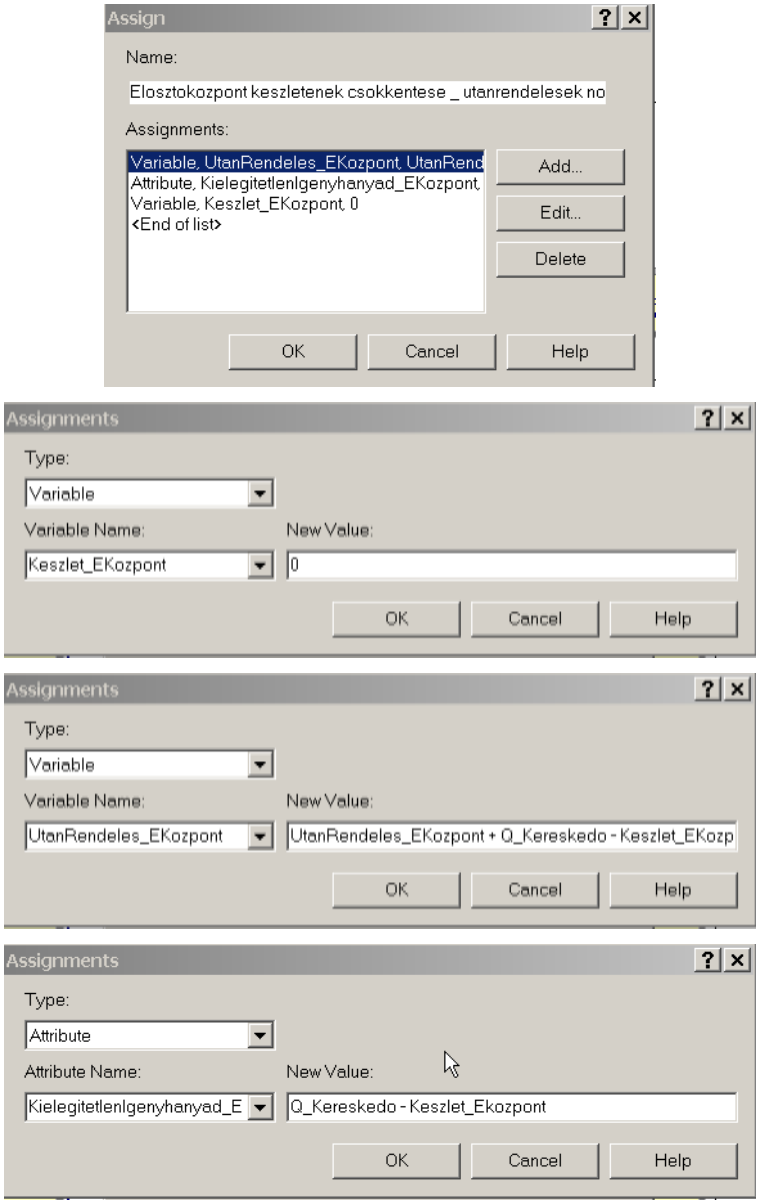
„*UtanRendeles_EKozpont*”=„*UtanRendeles_EKozpont*”+„*Q_Kereskedo*”–„*Keszlet_EKozpont*”

2. A rendelésentitás „*KielegitetlenIgenyhanyad_EKozpont*” nevű attribútumához hozzárendeljük az igény kielégítetlen hányadát:

KielegitetlenIgenyhanyad_EKozpont=Q_Kereskedo–Keszlet_Ekozpont

3. A készletszintet 0-ra csökkentjük:

Keszlet_EKozpont = 0



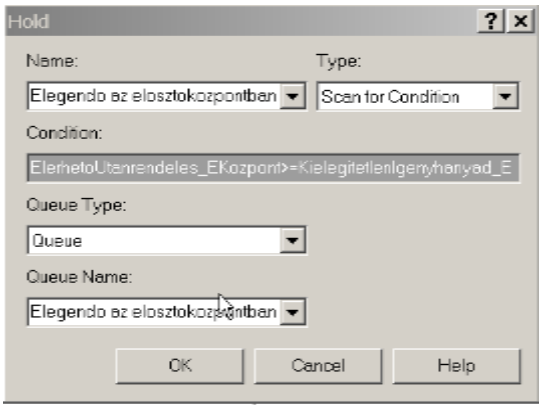
6.47. ábra: Az „*Elosztokozpont keszletenek csokkentese_utanrendelesek novelese*” nevű **Assign** modul dialógusablaka és a hozzátartozó hozzárendelések

A *6.47. ábra* az „*Elosztokozpont keszletenek csokkentese_utanrendelesek novelese*” nevű **Assign** modul és a modulban végrehajtott hozzárendelések dialógusablakait mutatja.

A rendelésentitás az értékadást követően belép az „*Elegendo az elosztokozpont*” nevű **Hold** modulba, ahol addig tartjuk vissza, ameddig az elosztó-központban elegendő mennyiség akkumulálódik a hiány pótlásához. Az entitás felszabadításának feltétele:

ElerhetoUtanrendeles_EKozpont>=KielegitetlenIgenyhanyad_EKozpont,

amint az a *6.48. ábrán* látható.



6.48. ábra: Az „*Elegendo az Output puffer?*” nevű **Hold** modul dialógusablaka

Az „*ElerhetoUtanrendeles_EKozpont*” változót arra használjuk, hogy nyomonkövessük az utánrendelés számára elérhető pillanatnyi készletet a készlethiány időtartama alatt. Ez azt jelenti, hogy a rendelésentitást visszatartjuk a **Hold** modulban addig, amíg a ki nem elégített igények kielégítéséhez elegendő készlet fel nem halmozódik. Megjegyezzük, egyidejűleg akár több rendelésentitást is visszatarthatunk a **Hold** modulban, de azok a modulba érkezés sorrendjében lesznek kielégítve, mivel a **Hold** modul sorában a FIFO elv érvényesül. Ha a feltétel

ElerhetoUtanrendeles_EKozpont >=KielegitetlenIgenyhanyad_EKozpont,

teljesül, akkor az entitás felszabadul és tovább halad az „*U_R szamara elerheto keszletcsok az E_kozpontban*” nevű **Assign** modulba, ahol az „*ElerhetoUtanrendeles _EKozpont*” változó csökken a kielégítetlen igényhányaddal:

ElerhetoUtanrendeles_EKozpont = ElerhetoUtanrendeles_EKozpont – KielegitetlenIgenyhanyad_EKozpont

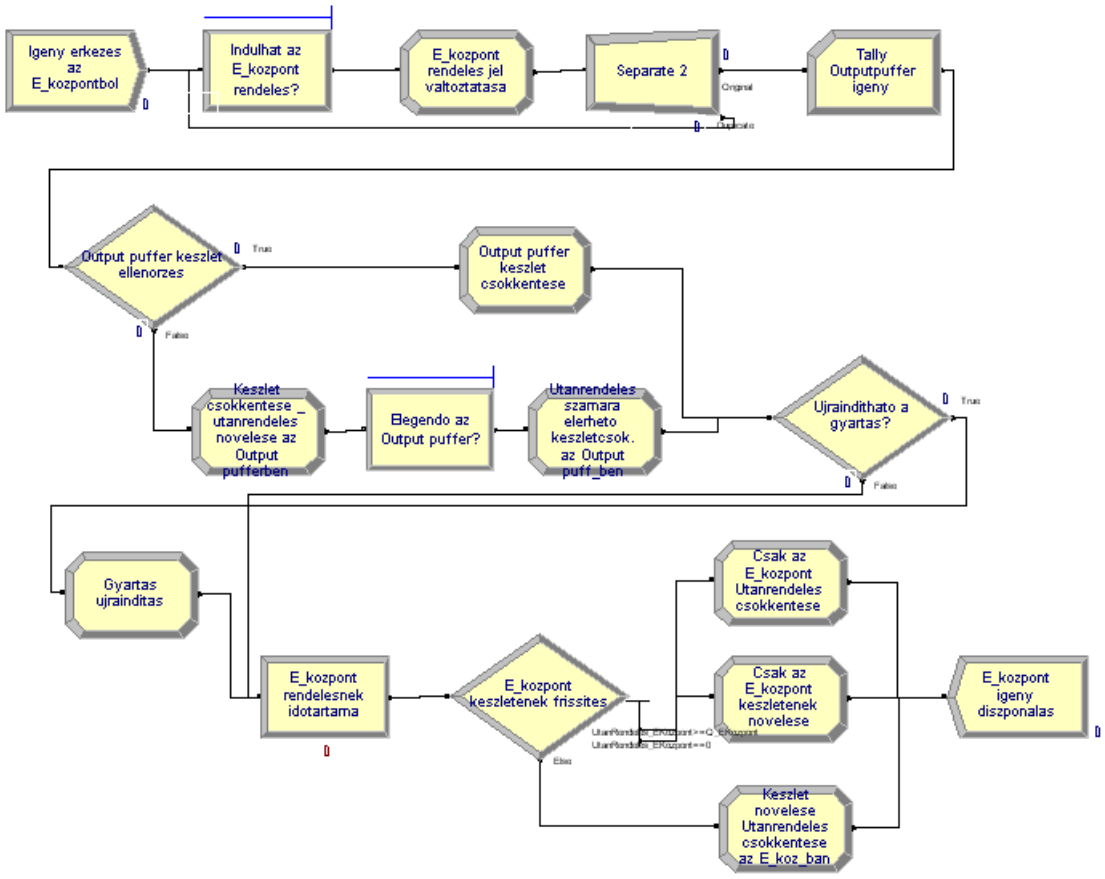
A kereskedői rendelésentitás ezek után kész a szállításra. Emlékezzünk arra, hogy azt feltételeztük, hogy a termékegységek szekvenciálisan áramlanak a szállítási rendszerben, így a szállítmányok nem előzhetik meg egymást. A szabály érvényesítéséhez, a szállítási idő modellezésére az elosztó-központ és a kereskedő között a „*Kereskedoi rendeles idotartama*” nevű **Process** modult használjuk. A rendelésentitás ezután belép a „*Kereskedoi keszlet frissites*” nevű **Assign** modulba, amelyben a kereskedő készletszintjét a kereskedői rendelés nagyságával növeljük:

Keszlet_Kereskedo = Keszlet_Kereskedo + Q_Kereskedo

Vegyük észre, a kereskedői készletpozíciót már akkor frissítettük, amikor elindítottuk a rendelést, ezzel szemben a kereskedő készletét csak akkor, amikor a rendelés megérkezett a kereskedőhöz. Végül a rendelésentitás a „*Kereskedoi igeny diszponalas*” nevű **Dispose** modulon keresztül elhagyja a rendszert.

6.3.5 Készletmenedzsment az output pufferben szegmens

A *6.49. ábrán* látható az output puffer készletmenedzsmentjének **Arena** modellje. Ez a modellszegmens generálja az elosztó-központ rendeléseit, frissíti az output puffer készletszintjét, amikor szükséges újraindítja a felfüggesztett gyártást, rakományokat küld az elosztó-központba, és frissíti az elosztó-központ készletszintjét.



6.49. ábra: Készletmenedzsment az output pufferben szegmens modellje

Az elosztó-központ rendelés fogadásának a logikája az output pufferben gyakorlatilag azonos az előző szegmensben alkalmazott logikával, ezért itt nem ismertetjük részletesen. Emlékezzünk arra, hogy a felfüggesztett „*E_kozpont Rendeles Entity*” nevű rendelésentitás az „*Indulhat az E_kozpont rendeles?*” nevű **Hold** modulban várakozik.

A rendelésentitás akkor felszabadul, amikor a *Rendeles_OutputPuffer* változó értéke 1 lesz. Hasonló logikával mint az előző szegmensben, ekkor a rendelésentitás belép a „*Separate 2*” nevű **Separate** modulba, ahol duplikálja önmagát. Az eredeti rendelésentitás, mint az elosztó-központ rendelése halad tovább, a másolata pedig egy hurok mentén visszatér a „*Indulhat az E_kozpont rendeles?*” nevű **Hold** modulba azzal a céllal, hogy generálja az elosztó-központ következő felfüggesztett rendelését.

A rendelésentitás következő állomása a „*Tally Outputpuffer igeny*” nevű **Record** modul, amely megszámlálja a rendelés mennyiséget, majd folytatja az útját az „*Output puffer keszlet ellenorzes*” nevű **Decide** modulba, ahol megvizsgáljuk, hogy a rendelkezésre álló készlet elegendő-e az igény kielégítéséhez. Most is két kimenet lehetséges. (1) Ha a feltétel *Keszlet_OutputPuffer*≥ *Q_EKozpont* teljesül, akkor a rendelésentitás a *True* ágon lép ki, és az „*Output puffer keszlet csokkentese*” nevű **Assign** modul felé halad, ahol a rendelkezés álló

készletet *Q_EKozpont* mennyiséggel csökkentjük, a teljesen kielégített igények számlálóját (*TelKielIgeny_OutputPuffer*) pedig 1-gyel növeljük:

$$Keszlet\_OutputPuffer = Keszlet\_OutputPuffer - Q\_EKozpont,$$

$$TelKielIgeny\_OutputPuffer = TelKielIgeny\_OutputPuffer + 1.$$

(2) Ha a $Keszlet\_OutputPuffer < Q\_EKozpont$ feltétel teljesül, akkor a rendelésentitás a *False* ágon lép ki. Ebben az esetben az igény nem elégíthető ki teljesen és utánrendelést kell kezdeményezni az output pufferből, ezért a rendelésentitás belép a „*Keszlet csokkentese_utanrendeles novelese az Output pufferben*” nevű **Assign** modulba, ahol három hozzárendelést hajtunk végre:

1. Az output puffer utánrendelés változóját (*UtanRendeles_OutputPuffer*) növeljük a hiány mértékével:

$$UtanRendeles\_OutputPuffer = UtanRendeles\_OutputPuffer + Q\_EKozpont - Keszlet\_OutputPuffer.$$

2. Az *KielegitetlenIgenyhanyad_OutputPuffer* entitás attribútumhoz hozzárendeljük az igény kielégítetlen hányadát:

$$KielegitetlenIgenyhanyad\_OutputPuffer = Q\_EKozpont - Keszlet\_OutputPuffer.$$

3. Az output puffer készletszintjét 0-ra állítjuk:

$$Keszlet\_OutputPuffer = 0.$$

A rendelésentitás ezután belép a „*Elegendo az Output puffer?*” nevű **Hold** modulba, ahol addig tartjuk vissza, amíg a

$$ElerhetoUtanrendeles\_OutputPuffer \geq KielegitetlenIgenyHanyad\_OutputPuffer$$

feltétel nem teljesül, azaz amíg az output pufferben elegendő nagyságú készlet fel nem halmozódik. Ha feltétel teljesül, akkor a rendelésentitás felszabadul, és az „*Utanrendeles szamara elerheto keszletcsok. az Output puff_ben*” nevű **Assign** modulba érkezik, ahol *ElerhetoUtanrendeles_OutputPuffer* változó értékét csökkentjük a hiány nagyságával (*KielegitetlenIgenyHanyad_OutputPuffer*):

$$ElerhetoUtanrendeles\_OutputPuffer = ElerhetoUtanrendeles\_OutputPuffer - KielegitetlenIgenyHanyad\_OutputPuffer$$

A rendelésentitás tovább halad a „*Ujraindithato a gyartas?*” nevű **Decide** modulba, ahol megvizsgáljuk, hogy a *Keszlet_OutputPuffer* változó elérte-e az újrarendelési pontot (*s_Uzem*) vagy az alá csökkent:

$$Keszlet\_OutputPuffer \leq s\_Uzem.$$

Ha a feltétel igaz, a rendelésentitás belép a „*Gyartas ujrainditas*” nevű **Assign** modulba, ahol a *Gyartas_Uzem* változó értékét 1-re állítjuk, amely azonnal felszabadítja a várakozó gyártásentitást a készletmenedzsment az input pufferben szegmens „*Gyartsunk?*” nevű **Hold** moduljában (*6.50. ábra*), ami gyakorlatilag a gyártás folytatását jelenti.

A rendelésentitás ezek után kész az elosztó-központba történő szállításra. A szállítás időigényét az output puffer és az elosztó-központ között a „*E_kozpont rendelesnek idotartama*” nevű **Process** modul modellezi. A rendelésentitás ezt követően belép a „*E_kozpont keszletenek frissites*” nevű **Decide** modulba, amelynek három lehetséges kimenete van:

1. Ha az $UtanRendeles\_EKozpont \geq Q\_EKozpont$ feltétel teljesül, akkor a rendelésentitás a „*Csak az E_kozpont Utanrendeles csokkentese*” nevű **Assign** modulba lép, ahol az el-

osztó-központ utánrendelését csökkentjük az elosztó-központ rendelésével és az utánrendelés számára elérhető mennyiséget ugyanannyival növeljük:

$$UtanRendeles\_EKozpont = UtanRendeles\_EKozpont - Q\_EKozpont,$$
$$ElerhetoUtanrendeles\_EKozpont = ElerhetoUtanrendeles\_EKozpont + Q\_EKozpont.$$

2. Ha az *UtanRendeles_EKozpont* = 0 teljesül, akkor a rendelésentitás a „*Csak az E_kozpont keszletenek novelese*” nevű **Assign** modulba lép, ahol, mivel az elosztó-központ nincs utánrendelés, az elosztó-központ készletét az elosztó-központ rendelésével növeljük:

$$Keszlet\_EKozpont = Keszlet\_EKozpont + Q\_EKozpont.$$

3. Ha a $0 < UtanRendeles\_EKozpont < Q\_EKozpont$, akkor a rendelésentitás a „*Keszlet novelese Utanrendeles csokkentese az E_koz_ban*” nevű **Assign** modulba lép be, ahol a következő hozzárendeléseket hajtjuk végre:

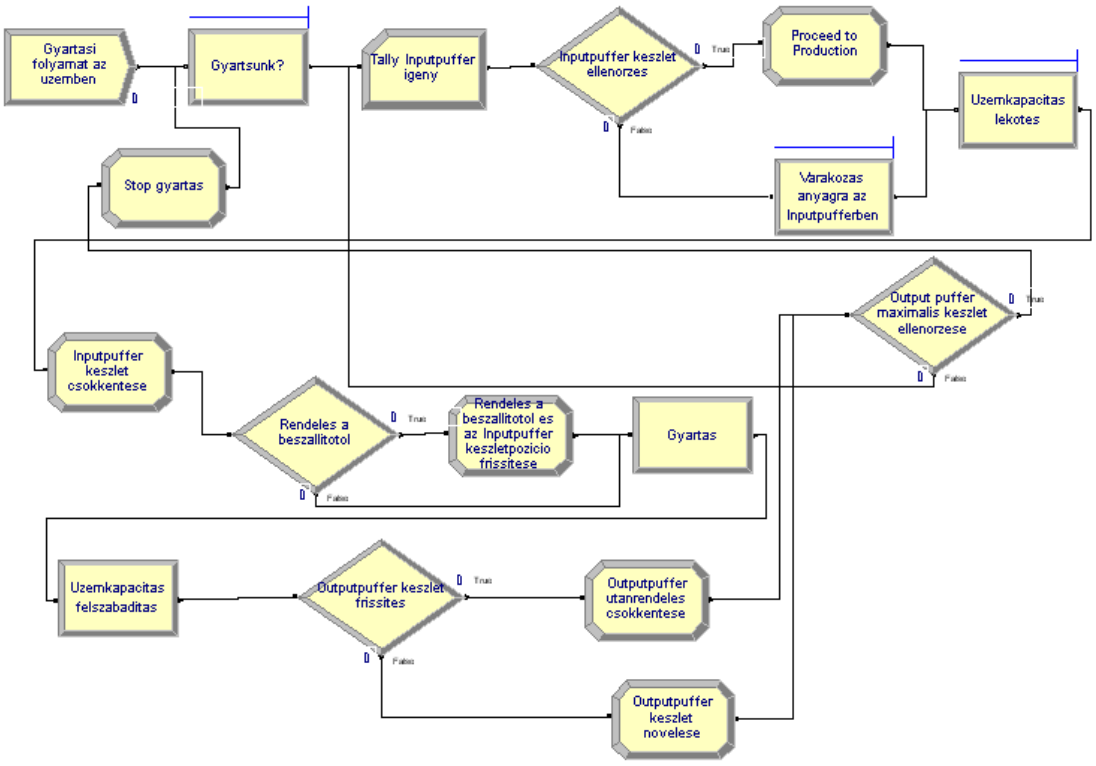
$$Keszlet\_EKozpont = Keszlet\_EKozpont + Q\_EKozpont - Utanrendeles\_EKozpont$$

$$ElerhetoUtanrendeles\_EKozpont = ElerhetoUtanrendeles\_EKozpont + Utanrendeles\_EKozpont,$$

$$UtanRendeles\_EKozpont = 0$$

Ez azt jelenti, hogy az elosztó-központ készlete csak az elosztó-központ rendelése és az utánrendelése közötti különbséggel nő. Az elosztó-központ utánrendelése számára elérhető készlet így az elosztó-központ utánrendelésével nő, és az utánrendelés 0-ra csökken.

Végül a rendelésentitás a „*E_kozpont igeny diszponalas*” nevű **Dispose** modulon keresztül távozik a rendszerből.



6.50. ábra: Gyártás- és készletmenedzsment szegmens az input pufferben

6.3.6 Gyártás- és készletmenedzsment az input pufferben szegmens

Az input puffer gyártás- és készletmenedzsment szegmensének **Arena** modelljét a *6.50. ábra* mutatja. Ez a modellszegmens menedzseli az alapanyag felhasználást és a gyártást olyan módon, hogy egy kontrollentitást cirkulál a modellben, amely szabályozza a gyártás felfüggesztését és folytatását.

Egységnyi termék előállításához a gyártóüzem egy egységnyi nyersanyagot mozgat az input pufferből a termelési folyamathoz és ennek eredményeként ad végterméket az output puffer készletéhez és frissíti annak szintjét. Ha elérjük a célszintet, akkor a gyártás megszakad, és a gyártás akkor indul újra, ha a készlet az újrarendelési pontra csökken. A gyártás leállását eredményezheti a nyersanyag hiány az input pufferben. Az input puffer kimerülése okozta leállás addig tart, amíg a puffert a beszállító fel nem tölti.

A „*Gyartasi folyamat az uzemben*” nevű **Create** modul egy „*OutputPuffer Rendeles Entity*” nevű kontrollentitást generál a 0 időpontban, amely a modellszegmensben cirkulál, és miden ciklus egy gyártási ciklusnak felel meg. A kontrollentitás először belép a „*Gyartsunk?*” nevű **Hold** modulba, és addig várakozik, amíg a gyártást nem engedélyezzük (emlékezzünk arra, hogy ez akkor történik, amikor a készletszint az output pufferben eléri az újrarendelési pontot vagy az alá csökken). Amikor engedélyezzük a gyártás folytatását, a kontrollentitás belép a „*Tally Inputpuffer igeny*” nevű **Record** modulba és számba veszi a következő termékegységet.

A kontrollentitás tovább halad az „*Inputpuffer keszlet ellenorzes*” nevű **Decide** modulba, ahol megvizsgáljuk, hogy van-e nyersanyag az input pufferben.

$$Keszlet\_InputPuffer \geq 1.$$

Ha a vizsgálat eredménye igaz, akkor az entitás a „*A termeles folytatasa*” nevű **Assign** modulban frissíti a teljesen kielégített igények számát:

$$TelKielIgeny\_InputPuffer = TelKielIgeny\_InputPuffer + 1.$$

Ha a feltétel az input puffer kiürülése miatt nem teljesül, akkor a kontrollentitás a *False* ágon távozik a „*Varakozas anyagra az Inputpufferben*” nevű **Hold** modulba, ahol addig várakozik, amíg a $Keszlet\_InputPuffer \geq 1$ feltétel igaz nem lesz. Amikor a feltétel teljesül a kontrollentitás belép a „*Uzemkapacitas lekotes*” nevű **Seize** modulba, ahol azonnal leköti a gyártás „*Uzem*” nevű erőforrását. Az egységnyi nyersanyag felhasználás modellezéséhez a kontrollentitás belép a „*Inputpuffer keszlet csokkentese*” nevű **Assign** modulba és egyszerre csökkenti a *Keszlet_InputPuffer* és a *KeszletPozicio_InputPuffer* változókat 1 egységgel.

$$Keszlet\_InputPuffer = Keszlet\_InputPuffer - 1$$

$$KeszletPozicio\_\ InputPuffer = KeszletPozicio\_InputPuffer - 1.$$

Hasonlóan az előző szegmenshez, a kontrollentitás a „*Rendeles a beszallitotol*” nevű **Decide** modulba lép azért, hogy megvizsgáljuk az input puffer készletszintje elérte-e az újrarendelési pontot:

$$KeszletPozicio\_InputPuffer \leq s\_InputPuffer$$

Ha a feltétel igaz, akkor a nyersanyag feltöltés azonnal megindul azáltal, hogy a készletmenedzsment a beszállítónál szegmensben (*6.51. ábra*), az „*Indulhat az uzem rendelese?*” nevű **Hold** modulban várakozó „*InputPuffer Rendeles Entity*” nevű rendelésentitást felszabadítjuk. A kontrollentitás ezután belép a „*Rendeles a beszallitotol es az Inputpuffer keszletpozicio frissitese*” nevű **Assign** modulba (*6.50. ábra*), ahol frissítjük a *KeszletPozicio*

_InputPuffer változót és a nyersanyag feltöltés indításához a *Rendeles_Beszallito* változó értékét 1-re állítjuk:

$$KeszletPozicio\_InputPuffer = KeszletPozicio\_InputPuffer + Q\_InputPuffer,$$

$$Rendeles\_Beszallito = 1.$$

A kontrollentitás mindkét esetben végül is belép a „*Gyartas*” nevű **Delay** modulba, (ebben egy termék gyártási idejét modellezzük), amit az „*Uzemkapacitas felszabaditas*” nevű **Release** modul követ, ahol felszabadítjuk az „*Uzem*” nevű erőforrást (*6.50. ábra*). Mielőtt egységnyi végterméket adnánk a készlethez, a kontrollentitás a „*Outputpuffer keszlet frissites*” nevű **Decide** modulban megvizsgálja, hogy van-e elintézetlen (függőben lévő) utánrendelés az output pufferben:

$$UtanRendeles\_OutputPuffer \geq 1.$$

Ha van függőben lévő utánrendelés, akkor a kontrollentitás a *True* ágon hagyja el a **Decide** modult, és belép a „*Outputpuffer utanrendeles csokkentese*” nevű **Assign** modulba, ahol a függőben lévő utánrendelést kielégítjük. Az output puffer utánrendelését 1-gyel csökkentjük, és az elérhető utánrendelést az output pufferben 1-gyel növeljük:

$$UtanRendeles\_OutputPuffer = UtanRendeles\_OutputPuffer - 1$$

$$ElerhetoUtanrendeles\_OutputPuffer = ElerhetoUtanrendeles\_OutputPuffer + 1$$

Különben, ha nincs elintézetlen utánrendelés, a kontrollentitás a *False* ágon keresztül belép a „*Outputpuffer keszlet novelese*” nevű **Assign** modulba, ahol frissíti az output puffer készletét:

$$Keszlet\_OutputPuffer = Keszlet\_OutputPuffer + 1.$$

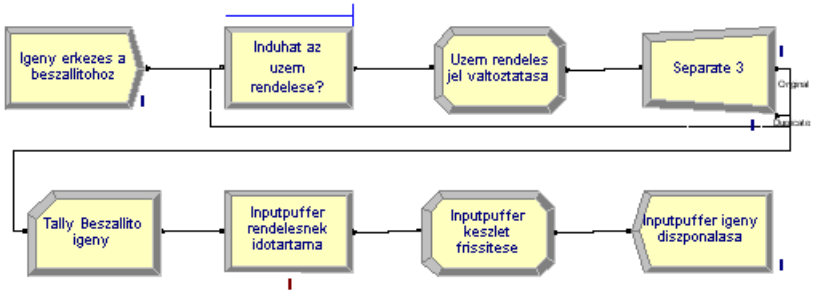
A kontrollentitás tovább halad a „*Output puffer maximalis keszlet ellenorzese*” nevű **Decide** modulba, ahol ellenőrizzük, hogy az output puffer készletszintje elérte-e a célszintet (maximális készletet):

$$Keszlet\_OutputPuffer \geq BigS\_Uzem.$$

Ha a feltétel igaz, akkor a kontrollentitás a *True* ágon át belép a „*Stop gyartas*” nevű **Assign** modulba, és itt a *Gyartas_Uzem* változó értékét 0-ra változtatjuk, ami felfüggeszti a gyártást, amikor az entitás belép a „*Gyartsunk?*” nevű **Hold** modulba, ahol addig várakozik, amíg a következő gyártási ciklus folytatódik. Különben a kontrollentitás a *False* ágon távozik a **Decide** modulból, és belép a „*Tally Inputpuffer igeny*” nevű **Record** modulba, és elkezdődik a következő gyártási ciklus.

6.3.7 Készletmenedzsment a beszállítónál szegmens

A beszállító készletmenedzsment szegmensének **Arena** modellje a *6.51. ábrán* követhető. Ez a modellszegmens generálja az input puffer rendeléseit, rakományokat küld a beszállítótól az input pufferbe és frissíti az input puffer készletszintjét.



6.51. ábra: Készletmenedzsment szegmens a beszállítónál

Az input puffer rendelés generálásának a logikája gyakorlatilag azonos az előző szegmensben használt logikával, ezért nem ismételjük meg a leírást. Megjegyezzük azonban, hogy ez a modellszegmens egy kicsit egyszerűbb, mint a hasonmása, mivel a beszállítónál mindig van elérhető nyersanyag készlet, nincs hiány, és ezért a feltöltési idő megegyezik a szállítási idővel.

6.3.8 Statisztikai gyűjtemény

A többlépcsős ellátási-lánc modell **Statistics** adatmoduljának táblázatnézetét a *6.52. ábra* mutatja. A táblázat tartalmazza a kereskedő, az elosztó-központ, az output puffer és az input puffer készletszintjének, továbbá az elosztó-központ és az output puffer utánrendeléseinek ún. *Time-Persistent* (időben folytonos) statisztikáit. A táblázatban megjelenik a gyártóüzem (a hasznos gyártási idő és az összes idő arányából számított) kihasználtságának mutatója, amely az ugyancsak *Time-Persistent* típusú. Végül, minden ellátási lépcső ügyfélszolgálati szintjét (feltöltési arányát) mutató ún. *Output* típusú statisztika, amely az utánrendelés nélkül, a rendelkezésre álló készletből azonnal kielégített rendelések valószínűsége (hányada):

Inputpuffer ugyfelszolgalati szint=TelKielIgeny_InputPuffer/NC(Tally Inputpuffer igeny)

Oputpuffer ugyfelszolgalati szint=TelKielIgeny_OutputPuffer/NC(Tally Outputpuffer igeny)/Q_EKozpont

Statistic - Advanced Process

	Name	Type	Expression	Report Label
1	Kereskedo atlagos keszlete	Time-Persistent	Keszlet_Kereskedo	Kereskedo atlagos keszlete
2	E_kozpont atlagos keszlete	Time-Persistent	Keszlet_EKozpont	E_kozpont atlagos keszlete
3	Outputpuffer atlagos keszlete	Time-Persistent	Keszlet_OutputPuffer	Outputpuffer atlagos keszlete
4	Inputpuffer atlagos keszlete	Time-Persistent	Keszlet_InputPuffer	Inputpuffer atlagos keszlete
5	E_kozpont atlagos utanrendelese	Time-Persistent	UtanRendeles_EKozpont	E_kozpont atlagos utanrendelese
6	Outputpuffer atlagos utanrendelese	Time-Persistent	Utanrendeles_OutputPuffer	Outputpuffer atlagos utanrendelese
7	Inputpuffer ugyfelszolgalati szint	Output	TelKielIgeny_InputPuffer/NC(Tally Inputpuffer igeny)	Inputpuffer ugyfelszolgalati szint
8	Outputpuffer ugyfelszolgalati szint	Output	TelKielIgeny_OutputPuffer/NC(Tally Outputpuffer igeny)/Q_EKozpont	Outputpuffer ugyfelszolgalati szint
9	Kereskedo ugyfelszolgalati szint	Output	TelKielIgeny_Kereskedo/NC(Tally Kereskedo igeny)	Kereskedo ugyfelszolgalati szint
10	E_kozpont ugyfelszolgalati szint	Output	TelKielIgeny_EKozpont/NC(Tally E_kozpont igeny)/Q_Kereskedo	E_kozpont ugyfelszolgalati szint
11	Uzem kihasznaltsag	Time-Persistent	Gyartas_Uzem==1	Uzem kihasznaltsag

Double-click here to add a new row.

6.52. ábra: A többlépcsős ellátási-lánc modell **Statistics** adatmodulja táblázatnézetben

6.3.9 A szimuláció eredményei

A szimulációs tanulmány hat változó igényű ellátási-lánc változatra terjed ki. A megbízható statisztikai kimenet eléréséhez, a szimulációs futás hosszát minden változatnál úgy választjuk meg, hogy 50.000.000 termékegységet állítsunk elő, ismétlés nélkül.

A hat féle igényérkezési rátához λ (bemenő paraméterhez) tartozó szimulációs kimenet eredményeit a *6.16. táblázat* foglalja össze. A számított statisztika a következő adatokat tartalmazza: az átlagos készletszint, az átlagos utánrendelésszint, és az ügyfél kielégítési ráta mind a négy lépcsőnél (input puffer, output puffer, elosztó-központ és a kereskedő). Az N/H jel rövidítés jelentése: nem használható.

A *6.16. táblázat* vizsgálva kiderül, hogy λ=1 paraméterhez tartozó átlagos készletszint az input pufferben 15,0203, és az ügyfél kielégítési ráta 98.61%. Ez azt jelenti, hogy a kereskedőnél a veszteség az igény 1,39%-ka. Az igényérkezési rátát változtatva az összes készletszint az elvárt minta szerint változik: igényérkezési ráta növekedése az átlagos készletszintek és ügyfél kielégítési ráták csökkenését, továbbá az átlagos utánrendelési szint növekedését eredményezi.

Vegyük észre, hogy a rendszer terhelés növekedése az ügyfél kielégítési ráta romlását legjobban a gyártóüzem input és output pufferében okozza. Ez annak a ténynek köszönhető, hogy a

gyártóüzem termelése állandó, amely növekvő igénnyel néz szembe. Így a növekvő terhelés a legnagyobb hatással a legfelső ellátási-lánc lépcsőre (eltekintve a beszállítótól) van.

6.16. táblázat

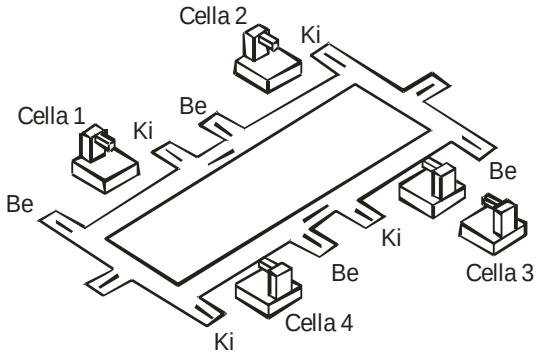
A többlépcsős ellátási-lánc szimulációjának kimenetei

	λ=1,0			λ=1,1		
	Készlet-szint	**Utánrendelési szint**	**Kielégítési ráta (%)**	**Készlet-szint**	**Utánrendelési szint**	**Kielégítési ráta (%)**
Input puffer	15,0203	N/H	99,88	14,8334	N/H	99,87
Output puffer	23,6452	0,0003	99,85	22,9043	0,0013	99,58
Elosztó-központ	23,0257	0,0000	100,00	22,8343	0,0000	100,00
Kereskedő	8,5209	N/H	98,61	8,3328	N/H	98,14
	λ=1,2			λ=1,3		
	Készlet-szint	**Utánrendelési szint**	**Kielégítési ráta (%)**	**Készlet-szint**	**Utánrendelési szint**	**Kielégítési ráta (%)**
Input puffer	14,6480	N/H	99,86	14,4680	N/H	99,85
Output puffer	22,0530	0,0052	98,88	21,0038	0,0199	97,23
Elosztó-központ	22,6296	0,0000	100,00	22,3842	0,0005	99,98
Kereskedő	8,1465	N/H	97,58	7,9646	N/H	96,97
	λ=1,4			λ=1,5		
	Készlet-szint	**Utánrendelési szint**	**Kielégítési ráta (%)**	**Készlet-szint**	**Utánrendelési szint**	**Kielégítési ráta (%)**
Input puffer	14,2924	N/H	99,84	14,1222	N/H	99,82
Output puffer	19,6205	0,0731	93,57	17,6510	0,2507	86,11
Elosztó-központ	22,0235	0,0031	99,90	21,3356	0,0169	99,56
Kereskedő	7,7828	N/H	96,26	7,5964	N/H	95,44

7. Rugalmas gyártási rendszerek modellezése és statisztikai analízise

7.1. A kisméretű rugalmas gyártási rendszerek modellezése

A kisméretű rugalmas gyártási rendszer vázlata a *7.1. ábrán* látható. A modellezett rendszerben négy gyártócellába érkeznek munkadarabok (Cella 1, Cella 2, Cella 3 és Cella 4), illetve távoznak onnan. Az első, a második és a negyedik cellában egy-egy szerszámgép működik, a harmadikban kettő. A Cella 3-ban a két szerszámgép kapacitása különböző, az egyik gép fiatalabb, amelynek a gépi időszükséglete a régi gép időszükségletének a 80%-a. A rendszerben háromféle munkadarabot gyártunk (1, 2, 3), amelyek különböző sorrendben kerülnek a gyártócellákba. A műveletek, illetve a cellák felkeresésének sorrendjét és a műveleti idők paramétereit a *7.1. táblázat* foglalja össze. Minden műveleti idő trianguláris eloszlású. Az 2. sorban és a 4. oszlopban olvasható 2/6, 9, 12 számok azt jelentik, hogy az 2 jelű munkadarab típuson a 4. műveletet a 2. gyártócellában (Cella 2) kell elvégezni, és ennek időigénye TRIA(6, 9, 12). A táblázatban a Cella 3 a régebbi, kisebb teljesítményű szerszámgép paramétereivel szerepel.



7.1. ábra: A kisméretű rugalmas gyártási rendszer vázlata

7.1. táblázat

A műveletek sorrendje és a műveleti idők paraméterei

Munkadarab típus	Cella/ Műveleti idő	Cella/ Műveleti idő	Cella/ Műveleti idő	Cella/ Műveleti idő	Cella/ Műveleti idő
1	1/6, 8, 10	2/5, 8, 10	3/15, 20, 25	4/8, 12, 16	
2	1/11, 13, 15	2/4, 6, 8	4/15, 18, 21	2/6, 9, 12	3/27, 33, 39
3	2/7, 9, 11	1/7, 10, 13	3/18, 23, 28		

A rendszerbe vegyesen belépő munkadarabok érkezési időközei exponenciális eloszlásúak, 13 perces átlagos érkezési időközzel. Az első munkadarab érkezési időpontja 0. A munkadarab típusok egymáshoz viszonyított arányai: munkadarab 1, 26%; munkadarab 2, 48%; munkadarab 3, 26%. A munkadarabok az ábra baloldalán lépnek be, mindig az óramutató járásával megegyező irányban mozognak, és jobboldalon távoznak. Első közelítésben feltételezzük, hogy az anyagmozgatási idő minden cella pár között két perc, függetlenül a távolságtól. Szeretnénk statisztikát készíteni az erőforrások kihasználtságáról, a sorok hosszáról és a sorokban eltöltött időről, továbbá a munkadarab típusonként a ciklusidőkről (a belépési és kilépési pont között eltelt időről). A szimuláció hossza kezdetben legyen 32 óra.

7.1.1. Új Arena fogalmak

A bevezetőben vázolt gyártási probléma néhány jellegzetes vonása új Arena fogalmak megismerést igényli. A korábban tárgyalt modellekben, ha ezt nem is hangsúlyoztuk, minden entitás ugyanabban a sorrendben áramlott az állomásokon keresztül. Ezzel szemben a most megoldandó feladat első jellemzője, hogy a rendszerben háromféle munkadarabot gyártunk különböző művelettervek szerint. Az ilyen típusú feladatoknál a művelettervnek megfelelő automatikus irányításra lesz szükségünk.

A második jellemző, hogy az egyik cellában (Cella 3) két nem egyforma kapacitású szerszámgép működik, az új gép nagyobb teljesítményű, mint a régebbi. A modellben e helyen különbséget kell tenni a két gép között.

A harmadik jellemző az entitások áramlásának a természete a rendszerben. A korábbi modelljeinkben az entitások áramlását a rendszerben a *Connect* opcióval vagy **Route** modul kínálta közvetlen irányítással értük el. Amikor a *Connect* opciót használjuk, az entitást azonnal (késlekedés nélkül) a kapcsolatnak megfelelő, következő modulhoz küldjük, miközben szimulációs idő nem változik. Ha a *Connect* opciót használnánk ebben a modellben, akkor több **Decide** modult kellene a modellbe iktatni, ahhoz, hogy a munkadarabokat a műveleti sorrendnek megfelelő állomásra irányítsuk. Ez a megoldás ugyan biztosíthatná a munkadarabok korrekt áramlását, azonban nem tenné lehetővé a 2 perces szállítási idő figyelembevételét és a munkadarabok áramlásának az animálását sem. Az anyagmozgatási idő reprezentálására használhatnánk a **Delay** modult is, azonban ez nem segítene a munkadarabok animálásában. Végül használhatnánk sok **Decide** modult, amelyeket **Route** modulok követnének két perces mozgatási idővel, annak érdekében, hogy a munkadarabok mozgását be tudjuk mutatni. Amint az elvárható, létezik egy Arena fogalom, a **Sequences**, amely entitások áramlásának könnyű modellezését teszi lehetővé a rendszerben, miközben a munkadarabok mozgását is szemléltetni tudjuk.

Sok rendszer jellemzője, hogy az entitások előre definiált szabályok szerint különböző útvonalakon áramlanak a rendszerben. A legtöbb gyártási rendszerben a munkadarabokhoz művelettervek (az elvégzendő műveleteket és azok sorrendjét tartalmazó listák) tartoznak, és határozzák meg a munkadarabok mozgását. Szolgáltató rendszerekben is léteznek hasonló követelmények. Például egy repülőtéren az utasok különböző útvonalakon közlekednek, attól függően, hogy az utas csomagját ellenőrzik vagy csak kézi poggyásza van, az utas belföldi vagy nemzetközi járaton utazik.

Az Arena az entitásokat az állomások felkeresésének előre definiált szekvenciája (sorrendje) szerint mozgatja. Az **Advance Transfer** panelen elérhető **Sequence** data modul lehetővé teszi, hogy előre definiáljuk az állomások sorrendjét, továbbá az attribútumok és a változók hozzárendelését minden állomáshoz. Ahhoz, hogy egy entitás a megadott szabály szerint áramoljon, a szekvenciát (sorrendet) hozzárendeljük az entitáshoz (a később leírt, beépített szekvencia attribútumot használva) és a **Route** modulban a *Sequential* opciót használjuk, amikor az entitást a következő állomásra mozgatjuk.

Az Arena –az entitás előírt szekvenciájának megfelelő útvonalon– az összes szükséges könyvelést elvégzi, nyilvántartja, hogy hol van az entitás, és hová tart. Ez három speciális, automatikusan definiált Arena attribútum: *Entity.Station* (M), *Entity.Sequence* (NS) és *Entity.JobStep* (IS) segítségével valósul meg. Minden entitás rendelkezik e három attribútummal, azonban az újonnan létrehozott entitások ezen attribútumainak alapértelmezett értéke 0. A *Station* attribútum tartalmazza az entitás pillanatnyi helyét, vagy az állomást ahova az entitás éppen tart. A *Sequence* attribútum tartalmazza az entitás áramlásának sorrendjét, amit minden entitáshoz hozzá kell rendelni ahhoz, hogy a szekvenciának megfelelően mozogjon. A *JobStep* attribútum az entitás pozícióját határozza meg a szekvencián belül.

A **Sequence** adatmodult használva, először definiáljuk és megnevezzük a különböző entitás típusok (példánkban a munkadarab típusok) által felkeresendő állomásokat. Ezt követően, amikor egy új entitás érkezik a rendszerbe, a specifikus szekvenciát társítjuk az entitással, úgy hogy a szekvencia nevét, amely azonos a *Sequence* attribútummal (NS), hozzárendeljük az entitáshoz. Amikor az entitás már kész arra, hogy a szekvencia szerinti következő állomásra mozogjon, akkor a modulban (pl. egy **Route** modul), amely az entitást a következő állomásra irányítja, a Destination Type mezőben a *Sequential* opciót választjuk. A futás alatt, ezen a ponton az Arena először növeli a *JobStep* attribútum (IS) értékét eggyel, majd a *Sequence* és a *JobStep* attribútumok pillanatnyi értéke alapján a szekvenciából lekérdezi a célállomást. Végül az Arena a célállomásra mozgatja az entitást.

Általában egy entitás követi a szekvenciát, miközben halad a befejezés felé, majd kilép a modellből. Bár ez nem követelmény. A *JobStep* attribútum értéke csak akkor nő, ha az entitást a *Sequential* opcióval továbbítjuk. Átmenetileg felfüggeszthetjük a szekvencia szerinti továbbítást, és az entitást közvetlenül más állomásra továbbíthatjuk, majd később visszatérhetünk az eredeti szekvenciához. Ez hasznos lehet akkor, ha például néhány munkadarab újramunkálást igényel a műveletsor néhány pontján. Az újramunkálás befejezését követően visszaállhatunk a normál szekvenciára.

A szekvencia attribútumok szintén bármikor újra hozzárendelhetők a folyamathoz. Például egy munkadarab hiba kezelésekor új *Sequence* attribútum hozzárendelésével és a *JobStep* attribútum 0-ra állításával. Az új *Sequence* a munkadarabot az újramunkálás területén egy új állomássorra irányíthatja. Visszaléphetünk, vagy előreugorhatunk a szekvenciában a *JobStep* attribútum csökkentésével, illetve növelésével. Azonban tanácsos biztonságra törekedni, meggyőződni a *JobStep* attribútum korrekt visszaállításáról, emlékezve arra, hogy az Arena először növeli azt, csak azután keresi a célállomást a szekvenciában.

Mint korábban jeleztük, a szekvenciában miden *JobStep*-nél elvégezhető az attribútumok és a változók hozzárendelése. Például, megváltoztathatjuk az entitás képét vagy hozzárendelhetünk egy műveleti időt a felhasználó által definiált attribútumhoz. A kisméretű rugalmas gyártási rendszer modelljében ezt az opciót arra használjuk, hogy meghatározzuk néhány állomás és munkadarab specifikus művelet idejét.

7.1.2. A modell közelítése

A rendszer közelítő leírására használt szimulációs modell általában függ a rendszer komplexitásától és az elérhető adatok természetétől. Egy egyszerű modellben magától értetődik, hogy mely modulokat használjuk, és azokat milyen sorrendben helyezzük el. A bonyolultabb modellekben azonban a megfelelő közelítés kialakításához gyakran alapos megfontolásokra lehet szükségünk. Az Arena ismereteink szélesedésével majd úgy találjuk, hogy egy rendszer modellezése sokszor többféle felosztással és módon is lehetséges. A gyakorlott modellezőktől számtalanszor hallható, hogy az egyszerű és korrekt modellezés egyidejűleg nem létezik. Azonban bőségesen léteznek rossz megoldások, amelyek nem vezetnek a kívánatos részletességű, korrekt rendszerleíráshoz.

A komplexmodellek tervezésének hajtóereje a modell adatszükséglete, és az, hogy az elérhető adatok mennyire reálisak. A gyakorlott modellezők gyakran sok időt töltenek azzal, hogy miként adják meg, tárolják és használják az adataikat, és ehhez milyen modellkonstrukcióra van szükség. Amikor az adatigény biztosítása egyre megerőltetőbbé válik, ez a közelítés gyakran csak annyi, hogy a pontos modell fejlesztését rövid idő alatt tegye lehetővé. Ez különösen igaz az ellátási rendszerek (raktárak, elosztóhálózatok és szolgáltatóhálózatok) szimulációs modellezésekor. Például egy raktárban több százezer különböző tétel található, amit tárolási egységeknek neveznek. Minden egységhez adatokat rendelünk a tárolás raktáron belüli helyéről, a méretéről, a tömegéről, az újrarendelésről és az új tételek betárolásáról. Ráadásul a rak-

tári adatok meghatározásához információra van szükségünk a vevőkről, a megrendelésekről, a tárolóeszközökről, amelyeken a tárolási egységeket elhelyezzük. Ha a modellünk megköveteli a tárolási egységek helyének változtatási lehetőségét, a tárolási eszközök megválasztását, a készletek figyelését, stb., akkor az adatstruktúra meglehetősen összetett és bonyolult lehet. Bár a könyvben bemutatott modellek nem bonyolultak, mindig tanácsos áttekinteni az adatigényeket még a modellezés megkezdése előtt.

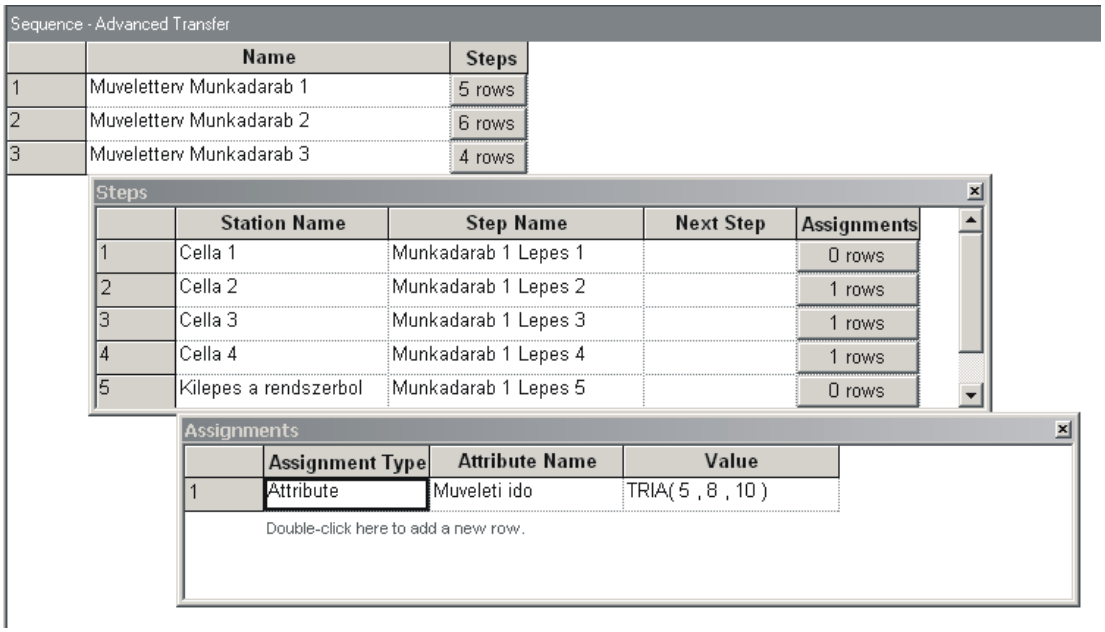
A kisméretű rugalmas gyártási rendszer korlátozott terjedelmű adatstruktúrája hatással lesz a modelltervezésre. A rendszerben áramló munkadarabok irányítására a **Sequence** adatmodult, és a **Sequence** adatmodul opcionális hozzárendelési tulajdonságát fogjuk használni arra, hogy az összes megmunkálási időt attribútumként adjuk meg, kivéve a Cella 1-et. A megmunkálási idő definiálására a Cella 1-ben egy kifejezést használunk. A szállítási idő és a Cella 3-ban működő új gép 80% -os megmunkálási idő szükségletének a kezelésére a *Variables* (változók) fogalmat használjuk. Bár nem normális a munkadarabokra, mint a tervezett adatok halmazára gondolni, a halmazok használata hatással van a modellezési módszerre. A modellben bevezetjük a *Set* (halmaz) fogalmat a felhasználó által definiált indexszel kombinálva, annak érdekében, hogy biztosítsuk a korrekt sorrendet és képet minden munkadarab típushoz.

Először az adatmodulokat töltjük fel, úgy ahogy korábban jeleztük. Aztán megadjuk a főmodell részeit, amely néhány új modult igényel. A következő lépésben animációt adunk a modellhez, használva a CAD-t vagy más rajzoló programot. Végül röviden megbeszéljük a modell verifikáció fogalmát. Már korábban megismerhettük az Arena párbeszédablakok megnyitásának és kitöltésének módszereit, ezért nem töltünk túl sok időt azokkal a földhözragadt részletekkel, hogy miként lássuk el információkkal az Arena-t. Az előző fejezetekben megismert modulok és fogalmak használatakor csak jelezzük az adatok megadásának szükségességét. Mielőtt neki látunk a fejlesztésnek, nézzük meg a *7.6. ábrát*, amely modell a teljes folyamatát szemlélteti.

7.1.3. Az adatmodulok (7.1. modell)

A modell fejlesztését az **Advance Transfer** panelen található **Sequence** adatmodul feltöltésével kezdjük (*7.1. képernyő*). A táblázatnézetében megjelenő modul megnyitása után kattintsunk kétszer a *Double-click here to add a new row* feliratra, hogy új sort adjunk a táblázathoz, és írjuk be az első szekvencia (műveletsor) nevét a Name mezőbe: *„Muveletterv Munkadarab 1”*. (Ez a műveletsor megfelel a *7.1. táblázat* első sorának.) A név beírása után a **Steps** dialógusablakot megnyitva, a műveletterv egyes, az Arena állomásokhoz (gyártócellákhoz) tartozó, lépéseit adhatjuk meg a műveleti sorrendnek megfelelően. Például a *„Muveletterv Munkadarab 1”*-hez szükségünk van a *„Cella 1”*, a *„Cella 2”*, a *„Cella 3”*, a *„Cella 4”* és a *„Kilepes a rendszerbol”* nevű állomásokra. A Step Names mezőben tetszőleges nevet adunk a műveleteknek, pl.: *„Munkadarab 1 Lepes 1”*-től *„Munkadarab 1 Lepes 5”* ig. Gyakori hiba, hogy elfelejtjük megadni az utolsó lépést, nevezetesen az entitás távozását a rendszerből. Ha ezt a lépést elfelejtjük megadni, akkor az Arena *run-time error* hibaüzenet küld, amikor az első entitás feldolgozása befejeződik, és az Arena nem tudja az entitást a következő állomásra irányítani. Amikor definiálunk egy szekvenciát és megadjuk az állomások nevét a Station Name mezőben (*7.1. képernyő*), azok azonnal felkerülnek a Station listára, és a modell dialógusablakaiban megjelenő, lenyíló Station listákon bárhol elérhetőek, kiválaszthatóak.

A munkadarabok műveleti idejét a *Cella* 2, 3 és 4 nevű gyártócellákban attribútumként, hozzárendeléssel adjuk meg az **Assignments** dialógusablakban. Az Arena sokoldalúságának bemutatására a *„Cella 1”*-hez tartozó műveleti időket más módon, az **Expression** adatmodulban fogjuk majd definiálni.



7.1. képernyő: A **Sequence** adatmodul táblázatai

7.2. táblázat

A **Sequence** adatmodul paraméterei

Name	Muveletterv Munkadarab 1
Steps Station Name Step Name	 Cella 2 Munkadarab 1 Lépés 2
Assignments Assignments Type Attribute Name Value	Random(Expo) Attribute Muveleti ido TRIA(5,8,10)

A *7.1. képernyőn* **Sequence** adatmodul táblázataiban, a három munkadarabhoz tartozó szekvenciák nevei, a *„Munkadarab* 1*”* művelettervének a lépései a gyártócellákkal, és *„Munkadarab* 1 *Lépés* 2*”* művelet műveleti ideje olvasható. A képernyőn látható adatokat a *7.2. táblázat* foglalja össze. A további két munkadarab típus „*Muveletterv Munkadarab* 2” és „*Műveletterv Munkadarab* 3” nevű műveletsorait a leírtakhoz hasonlóan adjuk meg. Az adatmodul feltöltéséhez szükséges alapadatok a *7.1. táblázatban* találhatók. Összefoglalva a **Sequence** adatmodulról eddig szerzett ismereteinket, megállapíthatjuk, hogy előnyösen alkalmazható többféle terméket gyártó üzemek irányítására, a művelettervek (műveleti sorrend, megmunkálási idő, stb,) tömör leírására. A modul egyértelműen tartalmazza, hogy egy adott munkadarabot, melyik helyeken, milyen sorrendben és mekkora megmunkálási idővel kell gyártani.

A három munkadarab típus műveleti idejét a *„Cella* 1*”*-ben az előzőektől eltérően kifejezésként adjuk meg, amihez az **Expression** adatmodult használjuk az **Advance Process** panelről. A modul táblázatainak kitöltéséhez a *7.3. táblázatot* használjuk. A kifejezés neve: *„Cella* 1 *ido”*, ami tartalmazza a három munkadarab műveleti idejét jellemző eloszlásfüggvények paramétereit. Ezeket megadhattuk volna korábban a **Sequence** modulban is. Az **Expression** adatmodult csak azért használjuk erre a célra, hogy bemutassuk a műveleti idő hozzárendelés más lehetőségét is. Mivel három féle munkadarabunk van, ezért 3 elemű vektor kifejezést (Rows: 3) definiálunk.

7.3. táblázat

A **Expression** adatmodul paraméterei

Name	Cella 1 ido
Rows	3
Expression Values	
Expression Value	TRIA(6,8,10)
Expression Value	TRIA(11,13,15)
Expression Value	TRIA(7,10,13)

Ezt követően a **Variable** adatmodult használjuk a **Basic Process** panelről, hogy definiáljuk a *„Cella 3”*-ban a szerszámgépek sebesség faktorait és a szállítási időket. Először definiáljuk a *Faktor* változót a következő feltételezés mellett. A *„Cella 3”*-ban a munkadarab műveleti idejét a régi gépre a **Sequence** adatmodulban megadtuk. Legyen az új gép indexe 1 és régi gépé 2. Így az első gép (új) faktora 0,8, a második gépé (régi) 1. Ez lehetővé teszi, hogy később ezt az értéket változtassuk. A **Variable** adatmodul paramétereit a *7.4. táblázat* összegzi.

7.4. táblázat

A **Variable** adatmodul paraméterei

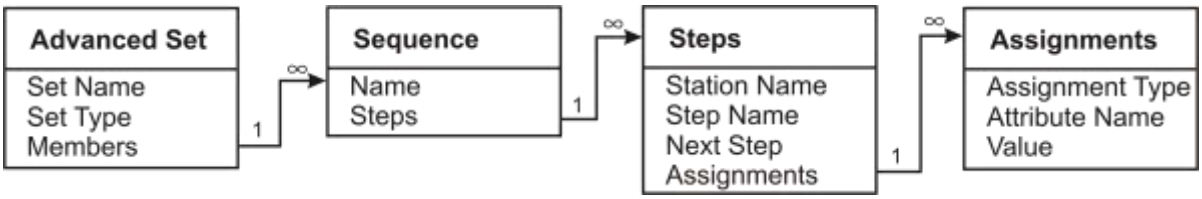
Name	Faktor
Rows	2
Initial Values	
Initial Value	0.8
Initial Value	1.0
Name	Szallitasi ido
Initial Values	
Initial Value	2

A **Basic Process** panelről a **Set** adatmodult használjuk különböző halmazok létrehozásához, többek között a *„Cella 3”*-ban a szerszámgépek, továbbá a munkadarab-képek és az entitás típusok definiálására. A *„Cella 3”*-hoz tartozó halmaz típusa *Resource* (erőforrás), amelynek két eleme: *„Cella 3 uj”* és *„Cella 3 regi”*. Az entitás képek halmazának típusa Entity Picture, a halmaz neve *„Munkadarab kep”*, és a halmaznak három eleme: *„Kep.Munkadarab 1”*, *„Kep.Munkadarab 2”* és *„Kep.Munkadarab 3”*. Végül az entitás típusok halmaza értelemszerűen Entity Type típusú, a halmaz neve *„Entitas tipus”*, és a halmaz három eleme: *„Munkadarab 1”*, *„Munkadarab 2”* és *„Munkadarab 3”*.

A munkadarab szekvenciákat azonosító halmaz elemeit, pl.: *„Muveletterv Munkadarab 1”*, stb. (amelyeket még létre sem hoztunk) a **Sequence** adatmodulban már használtuk (*7.1. képernyő*). Ha e halmaz definiálásához megkíséreljük a **Set** adatmodult használni, azonnal észrevehetjük, hogy az elérhető halmaztípusok között: *Resource* (erőforrás), *Counter* (számláló), *Tally* (jelölő, címkéző), *Entity Type* (entitás típus) és *Entity Picture* (entitás kép) nincs olyan típus, amely alkalmas lenne a szóban forgó halmaz létrehozására.

E probléma megoldásához ezért az **Advance Set** adatmodult használjuk az **Advance Process** panelről (*7.2. ábra*). A Set Name mezőbe beírjuk a műveletterveket tartalmazó halmaz nevét: „*Munkadarab Szekvencia*”. Az adatmodul Set Type mezejét kinyitva, a lista három típust sorol fel: *Queue* (sor), *Storage* (tároló) és *Other* (egyéb). Az *Other* típus egy olyan tároló, amely lehetővé teszi, hogy az Arena objektumokhoz hasonló halmazokat definiáljunk. A *„Munkadarab Szekvencia”* nevű halmaz létrehozásakor ezért az *Other* opciót használjuk. A halmaz három elemét a Members mezőhöz rendeljük: *„Muveletterv Munkadarab 1”*, *„Muveletterv Munkadarab 2”* és *„Muveletterv Munkadarab 3”*. A három munkadarab típus műveleti sor-

rendjének leírására tehát két adatmodult (**Advanced Set** és **Sequence**) használunk. Az adatstruktúrát a *7.2. ábra* szemlélteti.



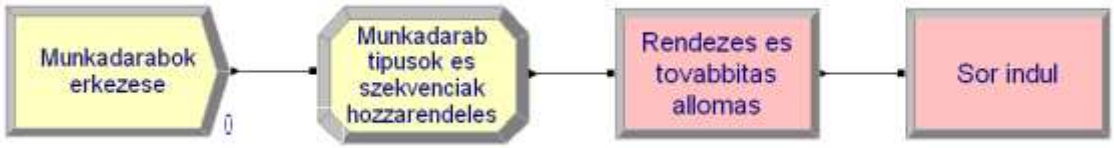
7.2. ábra: A műveleti sorrendet leíró adatstruktúra

Mielőtt a logikai modulokat elhelyeznénk a modellben, nyissuk meg a *Run>Setup* dialógusablakot és állítsuk a Replication Length értékét 32 órára és a Base Time Units paramétert *Minutes*-re. Majd az *Edit>Entity Pictures*–t választva nyissuk meg az **Entity Picture Placement** ablakot és hozzunk létre három különböző képet *„Kep.Munkadarab* 1*”*, *„Kep.Munkadarab* 2*”* és *„Kep.Munkadarab* 3*”* néven. A példában a három munkadarab jelölésére másoljuk le a blue (kék), red (piros) és green (zöld) golyókat, és nevezzük át azokat.

Az adatmodulok definiálása után készen állunk arra, hogy feltöltsük a modellablakot a logikai modulokkal, létrehozzuk a logikai kapcsolatokat, és megadjuk a modulok jellemzőit.

7.1.4. A logikai modulok

A modell motorja, a munkadarabok érkezését, a gyártócellák működését és a munkadarabok távozását reprezentáló, összekapcsolt modulok sorozata (*7.6. ábra*). A munkadarabok érkezését például a *7.3. ábrán* látható modulokkal modellezzük. Az érkezési folyamat modellezésére a **Create** modulban *Random(Expo)* eloszlást használunk 13 perces átlagos érkezési időközzel.



7.3. ábra: A munkadarabok érkezése modulok

Eddig még nem társítottuk a szekvenciákat az érkező entitásokkal. A társítást az **Assign** modulban végezzük el, amelynek a paraméterei a *7.5. táblázatban* találhatók. A hozzárendelések két célt szolgálnak: meghatározzák, hogy milyen munkadarab típus érkezik, és indexet (*Munkadarab Index*) definiálnak a munkadarab halmazok számára, ami lehetővé teszi, hogy minden érkezéssel társítsuk a megfelelő szekvenciát. Először diszkrét eloszlással (DISC() függvény) meghatározzuk a *Munkadarab Index*-et, azaz a munkadarab típusát. Ez a függvény alkalmas arra, hogy valamilyen értékeket generáljunk adott valószínűséggel. A példánkban az értékek a munkadarab típusok: 1, 2, 3 egészszámok, amelyek valószínűségei: 26%, 48% és 26%. Ezeket az értékeket a DISC() függvény argumentumában a kumulatív valószínűségértékükkel együtt, páronként adjuk meg (*7.5. táblázat*). A kumulatív valószínűség utolsó értéke (esetünkben a 3-dik) 1 csak lehet. Az argumentumban az értékeknek nem kell egészszámnak lenniük, felvehetnek bármilyen valós értéket, akár negatív számot is. A *Munkadarab Index* értékek 1, 2 és 3 nemcsak a munkadarab típusának azonosítására használhatók, hanem a korábban definiált *Munkadarab Szekvencia* indexeként is szolgálnak. Ennek érdekében a megfelelő szekvenciát hozzárendeljük az Arena Entity.Sequence nevű attribútumához is (*Entity.Sequence = Munkadarab Szekvencia*(*Munkadarab Index*)), amelyeket a *Munkadarab Szekvencia* nevű halmaz indexként használ.

7.5. táblázat

Az **Assign** modul paraméterei

Name	Munkadarab tipusok es szekvenciak hozzarendeles
Assignments	
Type	Attribute
Attribute Name	Munkadarab Index
New Value	DISC(0.26,1,0.74,2,1.0,3)
Type	Attribute
Attribute Name	Entity.Sequence
New Value	Munkadarab Szekvencia(Munkadarab Index)
Type	Attribute
Attribute Name	Entity.Type
New Value	Entitas Tipus (Munkadarab Index)
Type	Attribute
Attribute Name	Entity.Picture
New Value	Munkadarab Kep(Munkadarab Index)
Name	Szallitasi ido

Hasonlóan társítani kell a megfelelő entitás típus és - kép jellemzőket az újonnan érkező munkadarabokhoz. Ezt megtehetjük úgy, hogy a *Munkadarab Index* attribútumot használjuk az *Entitas Tipus* és a *Munkadarab Kep* halmazok indexelésére. Ezeket a halmazokat úgy képzelhetjük el, mint indexelt egydimenziós vektorváltozókat, ahol az index a *Munkadarab Index* nevű attribútum. Esetünkben ez igaz, mivel ún. *egy az egyhez* kapcsolat van a munkadarab típus (*Munkadarab Index*) és a *Munkadarab Szekvencia*, az *Entitas Tipus* és a *Munkadarab Kep* között. Például az 1 index arra utal, hogy a *Munkadarab* 1, a *Munkadarab Szekvencia*(1) műveletsor szerint lesz megmunkálva. Ez azonos a *Munkadarab Szekvencia* nevű halmaz *„Muveletterv Munkadarab* 1*”* nevű elemével. A jövőbeli modelljeinkben legyünk óvatosak, mert ez nem mindig igaz, csak akkor, ha az adatstruktúra *egy az egyhez* kapcsolatokat tartalmaz.

Esetünkben az **Assign** modul meghatározza a munkadarab típusát és azokhoz hozzárendeli megfelelő entitástípus és - kép szekvenciákat. Vegyük észre, hogy legfontosabb az **Assign** modulba elsőként definiált *Munkadarab Index*, amely többszörös hozzárendelést hajt végre a sorrend listán, mivel az elsőként meghatározott *Munkadarab Index* értéke az amit a későbbi hozzárendeléseknél használunk. Most már készen állunk arra, hogy az első munkadarabot a szekvenciája szerinti első állomásra küldjük.

A modellépítést az **Advanced Transfer** panelen található **Route** modullal fejezzük be. Mielőtt ezt megtesszük, ismernünk kell, hogy az entitásunk pillanatnyi helyzetét (az állomást, ahol pillanatnyilag tartózkodik). Az instrukcióink alapján végzett önálló modellépítés során az **Assign** modulban valószínűleg mindenki találkozott az Attribute Name mezőhöz tartozó lenyíló listában különböző entitás attribútumokkal, köztük az *Entity.Station* nevű attribútummal. Megkísérelhetjük, hogy a munkadarab pillanatnyi helyének meghatározása érdekében hozzáadjuk ezt az attribútumot a modellhez. Sajnos az Arena erre a megoldásra hibaüzenettel válaszol, amikor futtatjuk a modellt, ezért e megoldás helyett a **Station** modult fogjuk használni, amit a következőkben ismertetünk.

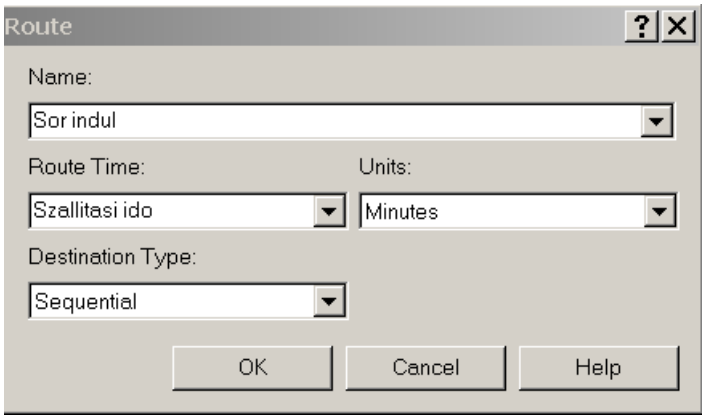
7.6. táblázat

A **Station** modul paraméterei

Name	Rendezes es tovabbitas allomas
Station Type	Station
Station Name	Rendezo allomas

A teljes modellünk 6 állomást fog tartalmazni, ezek: a rendezés és továbbítás, a Cella 1, a Cella 2, a Cella 3, a Cella 4, és a kilépés állomások. Az utolsó öt állomást akkor definiáltuk, amikor információkkal töltöttük fel a **Sequence** adatmodult (*7.1. képernyő*). Az első állomás a „Rendezes es tovabbitas allomas" definiálásához a **Station** modult (**Advanced Transfer** panelről) használjuk (*7.3. ábra*), amely meghatározza a belépő entitások pillanatnyi helyzetét. A modul paramétereit a *7.6. táblázat* mutatja.

Most már valóban készen állunk arra, hogy a munkadarabot egy **Route** modul (Advanced Transfer panel) segítségével az első állomásra küldjük. A **Route** modul entitást továbbít a modulban definiált állomásra, vagy az állomások meglátogatási sorrendjének (szekvenciájának) megfelelő következő állomásra. A szállítási idő a következő állomásra a Route Time mezőben adható meg (*7.6. képernyő*). A modellünkben ez az idő a korábban definiált „*Szallitasi ido*" nevű változó értéke. A Destination Type mezőben a *Sequential* opciót választjuk. Ilyenkor eltűnik a Station Name mező, és amikor futtatjuk a modellt, az Arena az érkező entitásokat a **Sequence** adatmodulban definiált szekvenciának megfelelő helyre továbbítja.



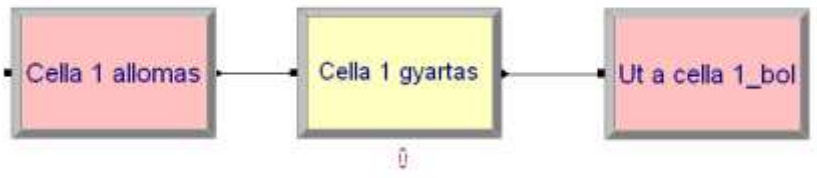
7.6. képernyő: A **Route** modul

Miután az érkező entitásokat a munkadarab szekvenciáknak megfelelő helyekre irányítottuk a modell négy megmunkáló cellájához kapcsolódó folyamatábrával folytatjuk a szerkesztést. Ennek a logikája a négy cellánál lényegében ugyanaz. A cellához érkező munkadarab beáll a sorba, és várakozik a megmunkálásra, majd ezt követően a megmunkálási sorrend szerinti következő állomásra küldjük. Mind a négy cellát **Station**–**Process**–**Route** modul sorozattal könnyen lehet modellezni (*7.4. ábra*).

7.7. táblázat

A *„Sor indul"*nevű **Route** modul paraméterei

Name	Sor indul
Route Time	Szallitasi ido
Units	Minutes
Destination Type	Sequential



7.4. ábra: Folyamatábra modulok a *Cella* 1-hez

A **Station** modul szimbolizálja azt a helyet, ahova a munkadarabot küldhetjük. A modellünkben munkadarab továbbításra a munkadarab típusonként megadott szekvenciát használjuk, így a munkadarab szállítás a szekvenciában megadott következő helyre történik. A *Cella* 1-hez tartozó **Station** modul paraméterei a *7.8. táblázatban* láthatók.

7.8. táblázat

A *„Cella*1 *allomas”* **Station** modul paraméterei

Name	Cella 1 allomas
Station Type	Station
Station Name	Cella 1

A Cella 1 állomásra beérkező munkadarabokat a “*Cella* 1 *gyartas*” nevű **Process** modulba küldjük (a közvetlen *Connect* opciót használva). A Cella 1-ben a műveleti időt már korábban egy kifejezéssel *Cella* 1 *ido* (*Munkadarab Index*) definiáltuk, amelyben a *Munkadarab Index* biztosítja az adott munkadarab típushoz tartozó műveleti idő hozzárendelést. Ez a kifejezés az előre megadott paraméterek alapján egy mintát generál a trianguláris eloszlásból. A modul további inputadatait a *7.9. táblázat* mutatja.

7.9. táblázat

A „*Cella* 1 *gyartas*” **Process** modul paraméterei

Name	Cella 1 gyartas
Action	Seize Delay Release
Resources	
Type	Resource
Resource Name	Cella 1 gep
Quantity	1
Delay Type	Expression
Units	Minutes
Expression	Cella 1 ido(Munkadarab Index)

A Cella 1-ben lejátszódó folyamatok befejezéseként az entitást a **Route** modulba küldjük, amelynek a paramétereit a *7.10. táblázat* szemlélteti, ahonnan a műveleti sorrendnek megfelelően a következő állomásra irányítjuk. A neve kivételével ez a **Route** modul azonos a „*Rendezes es tovabbitas allomas*”-on használt „*Sor indul*” nevű modullal (*7.7. táblázat*)

7.10. táblázat

A „*Ut a cella* 1_*bol*” **Route** modul paraméterei

Name	Ut a cella 1_bol
Route Time	Szallitasi ido
Units	Minutes
Destination Type	Sequential

A maradék három gyártócella modellezése nagyon hasonló a Cella 1 modellezéséhez, ezért a részletek ismertetésétől eltekintünk. A folyamatábra szerkesztéséhez másoljuk le a Cella 1-t (*7.4. ábra)* és módosítsuk a paramétereket. Mindenegyes **Station** és **Route** modulban egyszerűen változtassuk a Cella 1-et Cella 2-re, Cella 3-ra vagy Cella 4-re. Hasonlóan szerkesszük át a három **Process** modult és változtassuk meg a műveleti idő kifejezést *Cella* 1 *ido*(*Munkadarab Index*)-ről „*Muveleti ido*”-re. Emlékezzünk vissza, hogy a munkadarab megmunkálási időket a Cella 2-re, Cella 3-ra és Cella 4-re a **Sequences** adatmodulban attribútumként definiáltuk „*Muveleti* ido” néven. Amikor egy munkadarabot valamelyik cellába irányítunk, akkor az Arena automatikusan hozzárendeli ezt az attribútumot az entitáshoz.

Ennél a pontnál vegyük észre, hogy munkadarab megmunkálási időt a Cella 1-ben önkényesen kifejezésként (*Expression*) adtuk meg, és a többi cellában a **Sequences** adatmodulban attribútumként definiált értékeket rendeltünk a megmunkálási időhöz. Ezt a megoldást azért választottuk, hogy bemutassuk, a szimulációs modellben gyakran többféle adatstruktúrát használhatunk ugyanannak a célnak az elérésére. Könnyen átalakíthatnánk a modellünket úgy, hogy a többi cellához hasonlóan, a Cella 1-ben is attribútumot használjunk a megmunkálási időre, és fordítva kifejezéseket a Cella 3-ban és a Cella 4-ben. Azonban, a Cella 2-ben a kifejezés használata nehézségekkel járna, mivel a *Munkadarab* 2 kétszer is bekerül ebbe a gyártócellába, és a műveleti idők különbözőek, amint az a *7.1. táblázatból* kiolvasható. Így tudnunk kellene valahonnan, hogy a *Munkadarab* 2 először vagy másodszor jár-e a Cella 2-ben. Olyan kifejezést kellene definiálni, ami felismeri ezt. Ez egy érdekes feladvány lehet, bár modellezési szempontból nem kétséges, hogy a szekvenciában definiált attribútum alkalmazása előnyösebb. Vegyük észre azt is, hogy a mintamodellünk relatíve kicsi, kevés gyártócellát és munkadarab típust tartalmaz. A gyakorlatban azonban nem ritka, hogy 30-50 gyártócellában folyik több száz féle munkadarab gyártása. Ha a méret probléma megjelenik a modellezett rendszerben, akkor erősen ajánlható, hogy az adatstruktúránkat nagy gonddal tervezzük, mert jelentős hatása lehet a szimulációs projekt sikeres vagy hibás működésére.

A **Process** modul a Cella 3-ban kicsit különböző, mivel a ebben a gyártócellában két gép működik egy új é s egy régi, különböző műveleti időkkel. Ha a gépek azonosak volnának, akkor egy erőforrást használhatnánk dupla kapacitással. Ezt a problémát korábban már úgy oldottuk meg, hogy ezt a két gépet egy „*Cella* 3 *gep*” nevű halmazban egyesítettük. Most a „*Cella* 3 *gyartas*” nevű **Process** modulban ezt a halmazt használjuk, mint erőforrást. A **Process** modul többi paramétere a *7.11. táblázatban* látható.

7.11. táblázat

A „*Cella* 3 *gyartas*” **Process** modul paraméterei

Name	Cella 3 gyartas
Action	Seize Delay Release
Resources	
Type	Set
Set Name	Cella 3 gep
Quantity	1
Save Attribute	Index
Delay Type	Expression
Units	Minutes
Expression	Muveleti ido * Faktor(Index)

A Resource partícióban, a Type lenyíló listából a *Set* opciót választjuk. Ez lehetővé teszi, hogy a Set Name mezőben a „*Cella* 3 *gep*” nevű halmazt adjuk meg erőforrásként. Ahhoz, hogy a halmaz meghatározott elemét rendeljük az entitáshoz, az elemeket az *Index* nevű attribútummal azonosítjuk. Végül egy kifejezést használunk, hogy a kiválasztott géphez tartozó műveleti időt kiszámítsuk. A Resource Set választásakor fogadjuk el az alapértelmezett kiválasztási szabályt (Selection Rule). Ez a *Cyclical*, amely úgy működik, hogy az entitás az első elérhető erőforrást választja, és először az utoljára lekötött erőforrást követő elérhetőségét vizsgálja. Esetünkben az Arena a két erőforrást alternatívan (egymást követően) próbálja kiválasztani, ha azonban az egyik erőforrás foglalt, akkor az elérhető lesz kiválasztva. Nyilvánvalóan ezek a szabályok akkor használhatók, ha egyidejűleg több mint egy erőforrás érhető el. A *Random* kiválasztási szabály véletlen kiválasztást eredményez, a *Preferred Order* szabály az először elérhető erőforrást választja a halmazból. Ha ezt a szabályt használnánk, akkor mindig az új gép lenne kiválasztva, ha elérhető, mert az új gép megelőzi a régit az erőforrás-

halmazban. A többi kiválasztási szabályt csak akkor alkalmazzuk, ha egy vagy több erőforrás kapacitása nagyobb, mint 1.

A *Save Attribute* opció lehetővé teszi az éppen kiválasztott gép indexének a tárolását, amely kiválasztott halmazelemre hivatkozik a felhasználó által definiált attribútumok között. A modellünkben ezt az értéket az *Index* nevű attribútumban őrizzük meg. Ha az új gép van kiválasztva az attribútum értéke 1, és ha a régi, akkor 2. Ez a számozás az erőforrások beírási sorrendjét követi a halmaz definiálásakor. A műveleti idő számításához használt kifejezésben használjuk az *Index* attribútumot: *Muveleti ido* * *Faktor*(*Index*). Emlékezzünk arra, hogy az új gép műveleti ideje a régi gép műveleti idejének a 80%-a. Valószínűleg már világos ennek a kifejezésnek a működése, mégis bemutatjuk. Ha a halmazból az első erőforrást (új gép) választjuk, akkor az *Index* értéke 1 lesz, és a *Faktor*(1) változó értéke 0,8. Ha a második gép (régi gép) a kiválasztott, akkor az *Index* értéke 2, és a *Faktor*(2) változó 1,0, azaz az eredeti műveleti idővel számolunk. Bár a következő a módszer komplikáltnak tűnhet, az Arena utolérhetetlen rugalmasságának illusztrálására bemutatunk egy alternatív megoldást a probléma kezelésére:

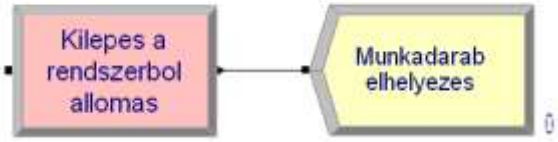
$$Process\ Time * ((Index == 1) * 0.8) + (Index == 2))$$

or

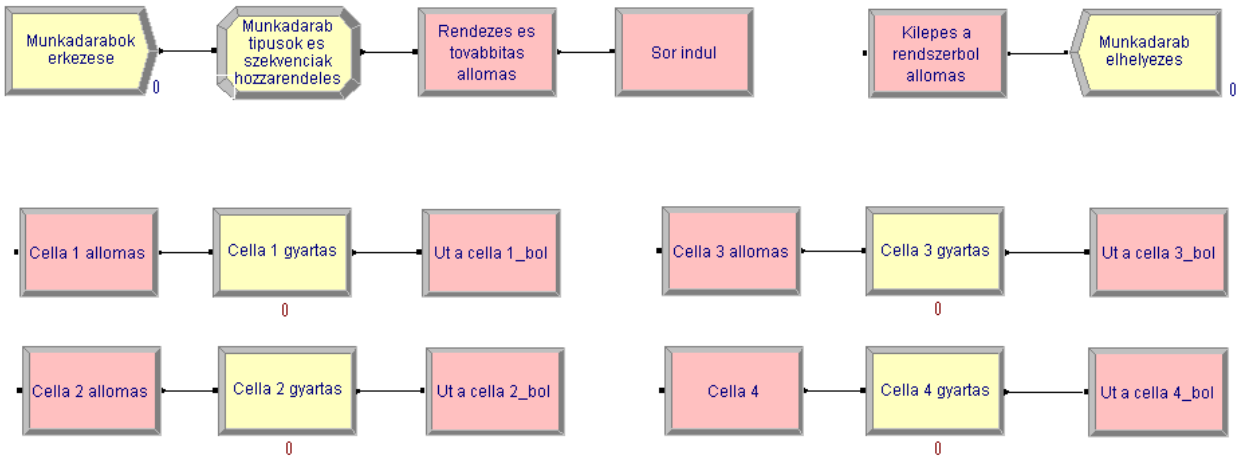
$$Process\ Time *(1 -((Index == 1)*0.2))$$

Az olvasóra bízzuk, hogy kitalálja ennek a kifejezésnek a működését, amihez segítségként, az *a==b* értéke 1, ha *a* egyenlő *b* -vel, különben 0.

A modell helyes működéséhez, illetve az adatok hiánytalan definiálásához, a megmunkálási helyekre érkező, majd a négy gyártócellán áthaladó munkadaraboknak még el kell hagyniuk a rendszert. A két modul, amely megvalósítja ezt a *7.5. ábrán* látható.



7.5. ábra: Folyamatábra modulok a kilépéshez



7.6. ábra: A teljes modell

A „*Kilepes a rendszerbol allomas*” nevű hely definiálásához használjuk a **Station** modult. A „*Munkadarab elhelyezes*” nevű **Dispose** modul eltávolítja a készre munkált munkadarabokat a rendszerből. A teljes modellt az animáció nélkül a *7.6. ábra* mutatja be.

Most már futtathatnánk a modellünket, de a futás alapján nehéz lenne megmondani, hogy a műveletek a megfelelő sorrendben működnek-e, azaz az egymáshoz kapcsolódó szekvenciák helyesen írják le a gyártási műveleteket. Ezért mielőtt analizálnánk a rendszerünket, először kifejlesztjük az animációt, ami sokat segít a modell helyes működésének megítélésében. Feltételezzük, hogy a modellt megmutatjuk a felső vezetésnek, ezért egy kicsit barátságosabb animációt szeretnénk kifejleszteni.

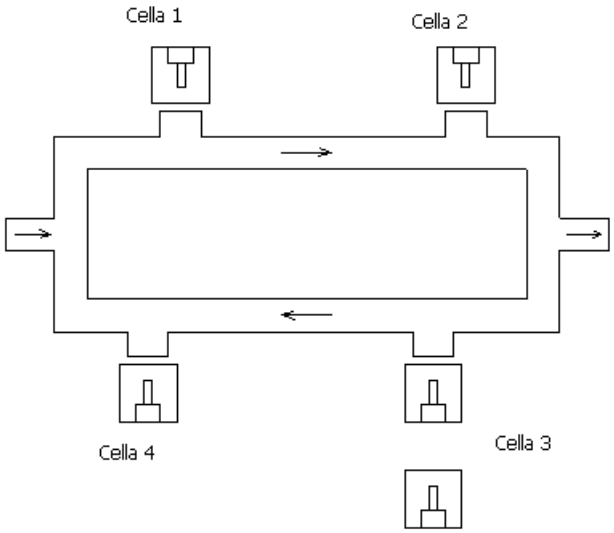
7.1.5. Az animáció fejlesztése

Az animációt hasonló módon fejlesztjük, mint azt az előző fejezetekben tettük, új entitásképeket, erőforrás szimbólumokat és a *7.1. ábra* alapján új alaprajzot szerkesztünk. Azonban felhasználhatunk korábban mások által megalkotott képeket is, ha azok elég pontosan ábrázolják a rendszert. Egy vállalatnál vagy szervezetnél létezhetnek CAD fájlok. Például a *7.1. ábrán* látható grafika egy AutoCAD-del szerkesztett rajz. Az Arena támogatja az AutoCAD-del készített fájlok integrációját. A DXF formátumban mentett AutoCAD fájlok közvetlen importálhatók az Arena-ba. A más CAD vagy rajzoló programokkal (pl. CorelDRAW, Visio®) szerkesztett rajzok is továbbíthatók az Arena-ba, ha azok az AutoCAD standardnak megfelelő DXF formátumban menthetők. Megtehetjük azt is, hogy a rajzoló programmal megszerkesztett ábrát a Clipboard-ba mentjük (Ctrl C) és onnan az Arena folyamatábra nézet ablakában tetszőleges helyre másoljuk (Ctrl V).

Ha az eredeti CAD rajz 2-D-s, akkor egyszerűen mentsük DXF formátumban és ezt követően importáljuk közvetlenül az Arena-ba. A legtöbb CAD objektum (poligon, kör, stb.) az Arena-ban hasonlóan jelenik meg, mint eredetileg. Ha azonban a CAD rajz 3-D-s, akkor először 2-D-be kell konvertálni. A színek ilyenkor elvesznek, azonban az AutoCAD-ben vagy az Arena-ban visszaállíthatók. Ez a konverzió minden objektumot vonallá transzformál, így az importált rajz szerkeszthető egyedi vonalakból áll, amelyeket az *Arrange > Group* paranccsal (gruppolással) összevonhatunk egy objektummá. A parancsot az *Arrange* eszköztárról a *Group* gombbal () is indíthatjuk. Tegyük fel, hogy van egy DXF fájlunk, és szeretnénk többet tudni a témáról. Ehhez az online help-et használva (a DXF File Importation témánál) részleteket találunk a DXF fájlokról és a 3-D konverzióról. Még egy fontos információ: az Arena-ba importált, konvertált fájlok objektumai fehér vonalak, ezért, ha a háttérszín fehér, akkor a vonalak nem látszanak. A láthatóság érdekében változtassuk meg a háttér színét, ehhez használjuk a *Window Background Color* gombot ()a *Draw* eszköztáron. A vonalak színének megváltoztatásához először jelöljük ki a rajzobjektumokat, majd használjuk a *Draw* eszköztáról a *Line Color* gombot ().

A kisméretű rugalmas gyártási rendszer 3-D-s rajzát 2-D-be konvertáltuk és Model7-1.dxf néven DXF formátumban mentettük. Ezt a fájlt importálhatjuk a modellünkbe a *File>DXF import* menü paranccsal. Olvassuk be a fájlt, a kurzort mozgassuk a modell ablakra, ekkor a kurzor kereszt alakúra változik. A kurzort használva rajzoljunk egy téglalapot úgy, hogy az egész importált rajzot tartalmazza. Ha az importált rajz nem megfelelő méretű, jelöljük ki az egész rajzot és méretezzük át a kívánt méretre. Ezt követően a rajzot, az animáció kezdő vagy háttér alakzataként használhatjuk. Az importált DXF rajzot a *7.7. ábra* szemlélteti.

Először az animációnk hátterét jelképező rajzot tegyük látványossá színek helyes megválasztásával. Ezt követően töröljük a rajz szöveges részeit és a nyilakat. Végül mozgassuk a „*Cella 1 gyartas*„ nevű **Process** modul felett elhelyezkedő, az 1-es gyártócella sorát animáló félegyenest (*Cella 1 gyartas.queue*) az animációra, rajzon a Cella 1 közelébe.



7.7. ábra: Az importált DXF rajz

Ezt követően szerkesszük meg az animációban az 1-es gyártócellában működő gépet szimbolizáló alakzatot. Ehhez először másoljuk le az eredeti ábrán látható *T* alakú alakzatot. Vigyük a kurzort az alakzat bal felső pontjára, nyomjuk le az egér bal gombját, és a gombot lenyomva tartva, rajzoljuk körbe az alakzatot. Az így kijelölt alakzatot a *Copy* gombbal ( ) másoljuk a clipboard-ba. Nyissuk meg az **Resource Picture Placement** (erőforráskép elhelyező) ablakot a *Resource* gombbal ( ) az *Animate* eszköztárról. Az erőforráskép elhelyező ablakban kattintsuk kétszer az *Idle* erőforrás szimbólumra. A megnyíló **Picture Editor** (képszerkesztő) ablakban cseréljük a szimbólumot a clipboard-ba másolt képre. A *Draw* eszköztárról használjuk a *Box* gombot ( ) és rajzoljunk két téglalapot a clipboard-ból másolt kép fölé, úgy hogy az kövesse az eredeti alakzatot, majd töröljük az eredeti képet. Az így kapott új alakzat a *Fill Color* gombbal ( ) tetszőleges színnel tölthető ki. Az erőforrás referencia pontját (kis kör a keresztel) az egérrel húzzuk az álló téglalap alsó oldalához. A képszerkesztő ablak bezárása után visszajutunk az erőforráskép elhelyező ablakba. Az új ikonunk azonosításához az Identifier mező lenyíló listájából válasszuk a „*Cella* 1 *gep*” nevű erőforrást. Az OK gombbal zárjuk be az erőforráskép elhelyező ablakot és térjünk vissza az animációnkhoz, ahol megjelenik egy kereszt alakú pointer. Pozícionáljuk a pointert megközelítőleg arra a helyre, ahová az erőforrást el akarjuk helyezni, és klikkeljünk. Esetünkben ez a hely a Cella 1. Az „*Cella* 1 *gep*” nevű erőforrást ezzel az animációba helyeztük. Ha az új erőforrásikonunk nem megfelelő méretű, akkor valamelyik sarkánál megfogva a kívánt mértre kicsinyíthetjük vagy nagyíthatjuk. Ez az erőforrás elhelyező ablakban a *Size Factor* értékének a megváltozását eredményezi. Ezért szerencsésebb az erőforráskép növelését vagy csökkentését a **Picture Editor**-ban (képszerkesztő) elvégezni, úgy hogy a *Size Factor* értéke közel 1 legyen.

Ha az újonnan szerkesztett erőforrásképet megfelelőnek találjuk, akkor mentsük el a korábban létrehozott konyv.plb nevű saját kép-könyvtárunkban. Az erőforráskép elhelyező ablakból nyissuk meg a konyv.plb nevű könyvtárat. Az ablak baloldalán jelöljük ki újonnan elkészített erőforrásikonunkat, kattintsunk **Current Library** terület alatti *Add* (hozzáadás) gombra, majd a jobbra (>>) nyílra. A *Save* (mentés) gombra kattintva mentsük az új ikonnal bővített könyvtárat. Ezt követően az ablak jobboldalán jelöljük ki az új erőforrás szimbólumot és másoljuk a baloldali *Busy* kép helyére.

A következő lépésben az erőforrás szimbólumot körülvevő téglalap (ez jelképezi a gépalapot) fölé rajzoljunk egy azonos méretű téglalapot, majd töröljük az eredetit. Az új téglalapot tölt-

sük ki az erőforráskép színétől eltérő színnel. Az erőforrásképet úgy helyezzük el, hogy referenciapontja a téglalap közepére essen.

A gépalapot jelképező téglalapot és a erőforrásikont ezután másoljuk rá a Cella 2, a Cella 3 és a Cella 4 gyártócellákra, majd másolás után az erőforrás elhelyező ablakban változtassuk meg az azonosítóikat.

Az animáció e fázisának befejezéséhez a még át nem telepített sorokat kell átméretezni és átmozgatni az animáció területére. Miután elkészültünk futtathatjuk a modellt és megfigyelhetjük az animációt. Észrevehetjük, hogy a munkadarabok megjelennek a gyártócellák előtt várakozó sorokban, de mivel még nem építettünk utakat a rendszerbe a munkadarabok mozogása nem látható. Ha közelebbről vizsgáljuk a gyártócellákat az animáció alatt, észrevehetjük, hogy a munkadarabok a gyártócella tetején helyezkednek el. Ez nem egészen az, amit látni szeretnénk. Az ideális az lenne, ha munkadarabok az erőforrás mögött tartózkodnának a megmunkálás alatt. A változtatás nem jelent problémát. Válasszuk ki az erőforrást (ezt megtehetjük szerkesztő üzemmódban vagy a szimuláció átmeneti megszakításával) és hozzuk a gyártócella elé az *Arrange>Bring to Front* menü opcióval vagy a *Bring to Front* gombbal () az *Arrange* eszköztárról. A modell újraindítása után látható lesz a kívánatos hatás.

Az animáció további finomításához láthatóvá tesszük a munkadarabok mozgását. A korábbi animációinkban az entitások egy irányban haladtak a rendszeren keresztül. A kisméretű rugalmas gyártási rendszerben az entitások több útvonalon haladnak, ezért gondosan és figyelmesen kell eljárni, annak érdekében, hogy az összes lehetséges utat hozzáadjuk a rendszerhez. Például a Cella 2 állomást elhagyó munkadarab mehet a Cella 1, 2 és 3 állomásra. Először az állomásokat (*7.7. képernyő*) (összesen hatot) kell elhelyezni az *Animate* eszköztáron elérhető *Station* gomb () segítségével.

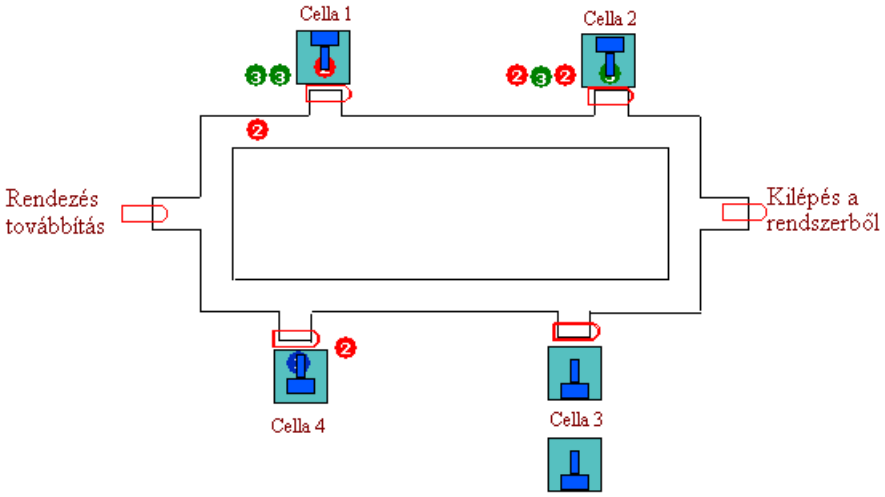
Station - Advanced Transfer

	Name	Station Type	Station Name	Report Statistics
1	Rendezes es tovabbitas allomas	Station	Rendezo allomas	☑
2	Cella 1 allomas	Station	Cella 1	☑
3	Cella 2 allomas	Station	Cella 2	☑
4	Cella 3 allomas	Station	Cella 3	☑
5	Cella 4 allomas	Station	Cella 4	☑
6	Kilepes a rendszerbol allomas	Station	Kilepes a rendszerbol	☑

7.7. képernyő: A modellben definiált állomások

Ezt követően az állomásokat összekötő utakat szerkesztjük az animációba a *Route* gomb () ismételt használatával. Ennek technikáját a 4. fejezetben már megismertük. Az útvonalak szerkesztéséhez segítségünkre lehet a *7.1. táblázat*, amely a munkadarabokon elvégzendő műveleteket és azok helyeit is tartalmazza. Ne felejtsük el azt sem, hogy a munkadarabok mindig az óramutató járásával megegyező irányban mozognak. Ha nem építenénk utakat az animációba, a szimuláció akkor is helyesen működne, mivel az entitások állandó 2 perces szállítási idővel mozognának az állomások között. Azonban ez azzal járna, hogy az entitások mozgása nem lenne látható az animációban. Figyeljünk arra is, hogy az utak kétirányúak is lehetnek. Például feltételezzük, hogy definiáltunk egy utat a Cella 1-ből a Cella 2-be és a párja, Cella 2-ből Cella 1-be hiányzik. Továbbá a Cella 2-ben egy munkadarab megmunkálása befejeződött, amit Cella 1-be kell szállítani. Ekkor az Arena először egy olyan utat keres, amely Cella 2-ből Cella 1-be vezet. Mivel ez az út hiányzik, az Arena az Cella 1-ből a Cella 2-be vezető utat fogja használni. Ez azt jelenti, hogy a munkadarab a Cella 2 belépési pontja és a Cella 1 kilépési pontja között az óramutató járásával ellentétes irányban fog mozogni, ami animációs hibát okoz.

Ha megfigyeljük a modell új animációját, akkor észrevehetjük, hogy időnként a munkadarabok megelőzik egymást, illetve áthaladnak egymás felett. Ez egyrészt az alapadatok véletlen kombinációjának, másrészt az animáció általunk választott módjának köszönhető. Emlékezzünk arra, hogy azt feltételeztük, hogy minden munkadarab szállítása bármely két állomás között 2 perc időt igényel. Ugyanakkor az Arena a mozgó entitások animációs sebességét a szállítási időből és az animációs út fizikai hosszából számítja. A modellünkben az animációs utak különböző hosszúságúak, például a Cella 1-ből a Cella 2-be nagyon rövid, és a Cella 2-ből a Cella 1-be nagyon hosszú. Tekintettel arra, hogy a szállítási idők azonosak, az utak pedig különböző hosszúságúak a sebességek is nagyon nagy különbségeket mutatnak, aminek köszönhetően időnként a munkadarabok megelőzhetik egymást. Ha ez nagyon zavaró, akkor az animációs utak figyelembevételével megadhatunk olyan szállítási időket, amelyek közel azonos sebességű mozgásokat eredményeznek a pálya bármely szakaszán. A megoldás legkönnyebb útja az lenne, ha törölnénk a „*Szallitasi ido*” változót, és a **Sequences** adatmodulban az új szállítási időket új attribútumhoz rendelnénk. További potenciális problémát jelent, ha egy munkadarab akkor lép be, amikor egy másik mozgó munkadarab éppen a belépés helyén jár, akkor a munkadarabok egymást fedik. Ez egy nagyon nehezen kiküszöbölhető jelenség, aminél rosszabbat el sem lehet képzelni. A következő fejezetben olyan anyagmozgatási megoldásokat fogunk megismerni, amelyekkel ez a jelenség elkerülhető.



7.8. ábra: A kisméretű rugalmas gyártási rendszer animációja

E rövid kitérő után nézzük az animációnkat, ami a *7.8. ábrához* hasonló képet mutat.

7.1.6. A modell verifikációja

Verifikáció (ellenőrzés) csak azt vizsgálja, hogy az Arena modell úgy viselkedik-e, ahogyan az a modellezési feltételek alapján elvárható. Ezzel szemben a validáció (megerősítés, jóváhagyás) sokkal nehezebb feladat, ami annak megállapítására irányul, hogy a modell úgy viselkedik-e, mint a valódi rendszer. Most itt röviden a modell ellenőrzésével foglalkozunk.

A verifikáció nyilvánvaló (magától értetődő) és nem nyilvánvaló problémákkal foglalkozik. Például, ha megpróbáljuk futtatni a modellt, és az Arena hibaüzenetet küld, amely azt jelzi, hogy elfelejtettük definiálni a Cella 3-ban a erőforrást, akkor a modell magától értetődően nem a szándékaink szerint működik. Ilyen jellegű hibák keresését a *Run>Check Model* menüopcióval egyszerűen elvégezhetjük. Feltételezve, hogy ilyen típusú hibák nem maradnak a modellben, a továbbiakban csak nem nyilvánvaló problémákkal foglalkozzunk.

Ellenőrzés meglehetősen könnyű, amikor kicsi, iskolapélda méretű problémákat fejlesztünk, amilyenek e könyvben is előfordulnak. Azonban, úgy fogjuk találni, hogy amikor a fejlesztés

sokkal realisztikusabb méretű modellekre irányul, ez a folyamat sokkal nehezebb, mint gondolnánk. Nagyon nagy modellek helyes működésben pedig 100%-ig soha nem lehetünk biztosak.

Egy egyszerű ellenőrzési módszer: csak egyetlen entitás belépését engedélyezzük a rendszerbe, majd figyeljük, hogy az entitás követi-e a modell logikát, és az adatok helyesek-e. A modell lépésenkénti futtatásához és az entitás követéséhez használjuk a *Run* eszköztáról a *Step* gombot ( ). A modell ellenőrzéshez a **Create** modulban a Max Arrivals mező értékét állítsuk 1-re. Az entitás típus kontrolálása érdekében az **Assign** modulban a diszkrételoszlást, amely meghatározza a munkadarab típusát (a „*Munkadarab Index*”-et), cseréljük le egy meghatározott munkadarab típusra. Például, ha a *Munkadarab* 1-t szeretnénk megfigyelni, akkor a „*Munkadarab Index*” értéke legyen 1. Így a munkadarab típusok útját (szekvenciáját) különkülön ellenőrizhetjük. Egy másik közismert módszer, hogy néhány vagy az összes valószínűségi adatot a modellben konstans adatokra cseréljük. Determinisztikus adatok használatával pontosan megjósolhatjuk a rendszer viselkedését.

Ha a modellünket reális döntések meghozatalához szeretnénk használni, akkor ellenőrizni kell a modell viselkedését extrém feltételek esetén. Például csak egy munkadarab típussal működtetjük a rendszert, vagy növeljük/csökkentjük az érkezési időközt. Ha a modell ilyenkor hibásan működik, akkor a hibák valószínűleg eljönnek futás közben is, ha véletlenszerűen hasonló hatások érik. Az animáció tényleges használatakor is gyakran jelentkeznek problémák, amit csak a rendszer működésében jártas személy vesz észre.

Gyakran jó ötlet hosszú futtatásokat végezni különböző bemeneti adatokkal és megfigyelni a potenciális problémák eredményeit. Tapasztalatok bizonyítják, hogy a verifikáció hasznos eszköze a hosszú futások megfigyelésén alapuló **teljesítménybecslés**. Nagyon, nagyon régen, mielőtt a számológépeket kifejlesztették, és a személyi számítógépek megjelenése előtt a mérnökök logarlécet hordtak magukkal, amit bőrtokkal védtek és büszkén csatolták a nadrágszíjukhoz. Ezek a varázslatos eszközök egyszerű kivitelű, mechanikus működésű analóg számítógépek, amelyek képesek különböző matematikai műveletek gyors, 3-4 számjegy pontosságú elvégzésére. Működésük alapelve, hogy a számok szorzatát a számok logaritmusának összegzésével, a számok hányadosát a számok logaritmusának különbségével számítják ki. A logarlécek kiválóan működtek, a használatukat csak az nehezítette, hogy a lécről leolvasható számok nem tartalmaztak tizedes vesszőt, a tizedesvessző helyét a mérnöknek kellett kitalálni, megbecsülni. Ezért a logarléc használata a mérnökökben olyan képességeket fejlesztett ki, hogy képesek voltak a problémákra durva becsléssel jó válaszokat adni. Például, ha valaminek a helyes eredménye 364,2, és a mérnöki becsült érték 400, akkor ez egy jó, elfogadható becslés. Azonban, ha a becsült érték 900, akkor az már hibás. Itt álljunk meg és döntsük el, hogy a leírtaknak van-e valami köze a becsléshez, vagy csak logarléckezelési problémáról van szó. Gyanítható, hogy a fiatalabb olvasókban két kérdés merül fel: (1) mire jó ez a hosszú az irreleváns mese, (2) ki használt valaha logarlécet? Nos a válaszok: (1) arra, hogy érzékeltessük a becslés problémáját (2) csak egy valaki, a szerző.

Hogyan használhatjuk a teljesítménybecslést, mint eszközt a modellünk ellenőrzésére. A szimulációs modellünkhöz definiáljunk induló feltételeket, és becsüljük meg a várható eredményeket, futtassuk a modellt és az eredmények alapján döntsük el, hogy a várt eredményeket kaptuk-e. (Sokszor lehetőségünk van olyan induló feltételek megadására, aminek ismert a végeredménye.) Ha a megérzéseink helyesek voltak, akkor próbálkozzunk más feltételekkel is. Ha nem azt kaptuk, amit vártunk, próbáljuk kitalálni az okát. A rossz becslés oka lehet a rendszer hiányos ismerete, vagy a hibás modell. Néha a nem túl nyilvánvaló, de reálisnak tűnő eredményeket váratlan kölcsönhatások idézik elő. Általában mielőtt egy modellt valós

döntések megalapozására használnánk, alaposan teszteljük. Az előzetes próbafuttatások és vizsgálatok kevésbé fájdalmasak, és nem járnak káros következményekkel.

7.2. A stacionárius szimuláció kimeneteinek statisztikai analízise

Korábban már érintettük a véges (terminating) és a stacionárius (steady-state) szimuláció közötti különbséget és megmutattuk, hogyan használhatók a **PAN** és az **Output Analyzer** az Arena riportok statisztikai analízisére véges szimuláció esetén. Ebben a szakaszban bemutatjuk, hogyan készíthetünk statisztikát a stacionárius szimulációról.

Mielőtt bár mit is tennénk, először győződjünk meg arról, hogy a stacionárius szimuláció megfelel-e annak a célnak, amit szeretnénk elérni. Gyakran egyszerűen azt feltételezzük, hogy a hosszú futás azonos a stacionárius szimulációval, ami akár igaz is lehet. De ha a kezdeti és a megállási feltételek a modellünk működésének a lényeges részét alkotják, a véges analízis valószínűleg megfelelőbb. A stacionárius szimulációtól való idegenkedés oka, amint azt látni fogjuk, hogy a stacionárius szimulációban sokkal nehezebb valamilyen statisztikai analízisre alapozott megbízható közelítést elérni, mint véges esetben, különösen akkor, ha azt szeretnénk, hogy valamilyen mennyiség becsült középértéke a 95%-os konfidencia-intervallum felett legyen. Ezért ha nem szükséges, akkor ne alkalmazzuk.

Még egy figyelmeztetés mielőtt belemerülünk az anyagba. Könnyen belátható, hogy a stacionárius szimuláció futási ideje egy kicsit hosszabb. Ennek köszönhetően több esélye van arra, hogy az Arena belső műveletei egy kicsit különböző eredményeket produkáljanak, amit a véletlen számok árama okoz. Ez nem teszi a modellünket minden értelemben rosszá vagy pontatlanná, de hatással van a numerikus eredményekre, különösen olyan modellben, amelyben sok a statisztikai változó. Így, ha a mintamodelleket a komputerünkön hosszú szimulációs idővel futtatjuk, akkor esély van arra, hogy olyan numerikus válaszokat kapjunk, amelyek különböznek a könyvben közölt riportok eredményeitől. Ennek oka, hogy egy szimuláció kimeneti adatainak változékonysága nemcsak a modell tulajdonságainak természetéből, hanem belső számítások véletlenszerűségéből is fakad. Emiatt ne essünk pánikba, ez van, fogadjuk el. Ugyanakkor ez a jelenség még inkább indokolja, a kimenti adatok alapos statisztikai vizsgálatát.

A 7.2.1 szakaszban a modell felmelegedési (warm up) és futási hosszának a meghatározásával foglalkozunk. A 6.7.2 szakaszban az analízisben használható csonkított-ismétlés stratégiát ismertetjük, amely távol áll az egyszerű közelítéstől, ezért bizonyos értelemben a leghatékonyabb. Egy másik közelítés a hatching, amit a 6.7.3 szakaszban mutatunk be.

7.2.1 A felmelegedési szakasz és a futási hossz (7.2. modell)

A mintapélda, a kisméretű rugalmas gyártási rendszer modelljének fontos jellemzője, hogy induláskor a rendszer üres és tétlen állapotban van. Ez azt jelenti, hogy a modell indulásakor nincs entitás a rendszerben és az erőforrások tétlen (*Idle*) állapotban vannak. Egy véges rendszerben az indításnak ez a szokásos módja. A stacionárius szimulációban ezzel szemben nincsenek kezdeti feltételek, azt feltételezzük, hogy a működés körülményei változatlanok.

Természetesen a stacionárius szimulációt is inicializálni kell, és a futását valahogy meg is kell állítani. Mivel első alkalommal csinálunk ilyen szimulációt, nagy eséllyel fogadhatnának arra, hogy nem túl sokat tudunk a tipikus stacionárius rendszer állapotról, vagy arról, hogy az állapot eléréséhez milyen hosszú futás elegendő. Így valószínűleg hajlunk arra, hogy olyan állapotot inicializáljunk, ami a stacionárius állapot szempontjából a kicsit szokatlan, és hosszú futási időkkel próbálkozunk. Ha például üres és tétlen állapotot inicializálunk egy szimulációban, amelyben az események véletlenszerűen torlódnak, akkor az inicializálást követően azt

tapasztaljuk, hogy a kimeneti adatok néhány periódus után csökkenő torlódást mutatnak, vagyis az adataink kevésbé torzulnak.

A probléma orvosolása érdekében, megpróbálhatjuk az üres és tétlen állapot helyett másként inicializálni a stacionárius rendszert. Például a 0 időpontban néhány entitást helyezünk a modellbe és így indítjuk a rendszert. Bár ez egy lehetséges módszer, az alkalmazása azonban kényelmetlen. Több mint problémás, hogy általában fogalmunk sincs arról, hány entitásnak kellene a 0 időpontban a rendszerben lennie. Ez egy kérdés, amire a szimuláció választ vár.

Egy másik útja az inicializáció miatti torzulás kezelésének, hogy olyan sokáig futtatjuk a modellt, hogy bármilyen kezdeti torzulás is van benne, az a nagyszámú adat hatására eloszlik. Ez a módszer eredményesen használható, ha a kezdeti feltétek okozta torzulási effektus gyorsan megszűnik.

Azonban amit rendszerint teszünk, üres és tételen állapotot inicializálunk, tudva azt, hogy ez nem reprezentálja a stacionárius állapotot, majd hagyjuk a modell felmelegedni (*warm up*), amíg a mesterséges kezdeti feltételek hatása el nem kopik. Ekkor töröljük a statisztikai akkumulátorokat (nem a rendszer állapotát) és innen újból indítjuk a statisztikai adatok gyűjtését az analízishez. Ténylegesen ez egy másik állapotot inicializál, ami különbözik az üres és tétlen állapottól, és a modellre bízza, mennyi entitás legyen a rendszerben, amikor elkezdjük megfigyelni a stacionárius működést. A felmelegedés időtartama lehet hosszú, vagy kevésbé hosszú, attól függően, hogy a kezdeti torzulások mennyi idő alatt kopnak el.

Az Arena-ban nagyon könnyű megadni a kezdeti felmelegedési periódust. Nyissuk meg a *Run> Setup> Replication Parameters* dialógust és a Warm-up Period mezőbe írjunk be egy megfelelő értéket (ne felejtsük el, ellenőrizni az időegységet). A modellünk minden ismétlésnél ugyanúgy indul, mint korábban, de a felmelegedési periódus végén törli a statisztikai akkumulátorokat, és így a riportjaink (és minden más Output típusú adat a **Statistic** adatmodulban) csak a felmelegedési periódus után keletkező adatokat fogják tartalmazni. Ilyen módon megtisztíthatjuk az adatainkat a kezdeti feltételek okozta torzulásoktól.

A feladat legnehezebb része a felmelegedési periódus hosszának a meghatározása. Valószínűleg a legpraktikusabb eljárás, hogy a kulcsfontosságú kimeneti adatokról grafikonokat készítünk, és ezeken megfigyeljük, hogy mikor állandósulnak. Ennek bemutatására, a 7.1. modellen a következő módosításokat hajtjuk végre és Modell7-2 néven mentjük.

Az átfogó kimeneti teljesítmény mérés előkészítéséhez, használjuk a **Statistic** adatmodult (*7.9. ábra*), amellyel kiszámítjuk a háromféle munkadarab összegzett WIP paraméterét. Emlékeztetőül, a WIP (work in process) az egyidejűleg feldolgozás alatt vagy előtt álló entitások számát mutatja. Nyissuk meg a **Statistic** adatmodult és a Name mezőbe írjuk be a statisztika nevét: "*Osszes WIP*", a Type mezőben válasszuk a *Time-Persistent* opciót. Az Expression mezőben szerkesszük meg a kifejezést az **Expression Builder** segítségével. A jobb egérgombbal az Expression mező felett nyissuk meg a legördülő listát, válasszuk a *Build Expression* opciót, ami megnyitja az **Expression Builder** dialógusablakát. A *Basic Process Variables* > *Entity* > *Number in Process* opciókat használva a Current Expression mezőben megjelenik az *EntitiesWIP*(*Entity 1*) kifejezés, amelynek az argumentumát kell "*Munkadarab* 1"-re változtatni. Ehhez az Entity Type legördülő listájából válasszuk a "*Munkadarab* 1" nevű entitást, átírja az argumentumot. Ezt követően az előző lépéseket ismételve, adjuk a kifejezéshez a "*Munkadarab* 2" és a "*Munkadarab* 3" entitásokhoz tartozó függvényeket. A kifejezés végleges alakja a következő:

EntitiesWIP(*Munkadarab* 1)+*EntitiesWIP*(*Munkadarab* 2)+*EntitiesWIP*(*Munkadarab* 3).

Ez a kifejezés a **Category Overview** > User Specified > *Time-Persistent* szekciójában megjeleníti az „*Osszes WIP*” nevű statisztikát, az egyidejűleg megmunkálás alatt vagy előtt álló munkadarabok összesített számáról.

Statistic - Advanced Process

	Name	Type	Expression	Report Label	Output File
1	Osszes WIP	Time-Persistent	EntitiesWIP(Munkadarab 1)+EntitiesWIP(Munkadarab 2)+EntitiesWIP(Munkadarab 3)	Osszes WIP	OsszesWIP.dat
	Double-click here to add a new row.				

7.9. ábra: A **Statistic** adatmodul

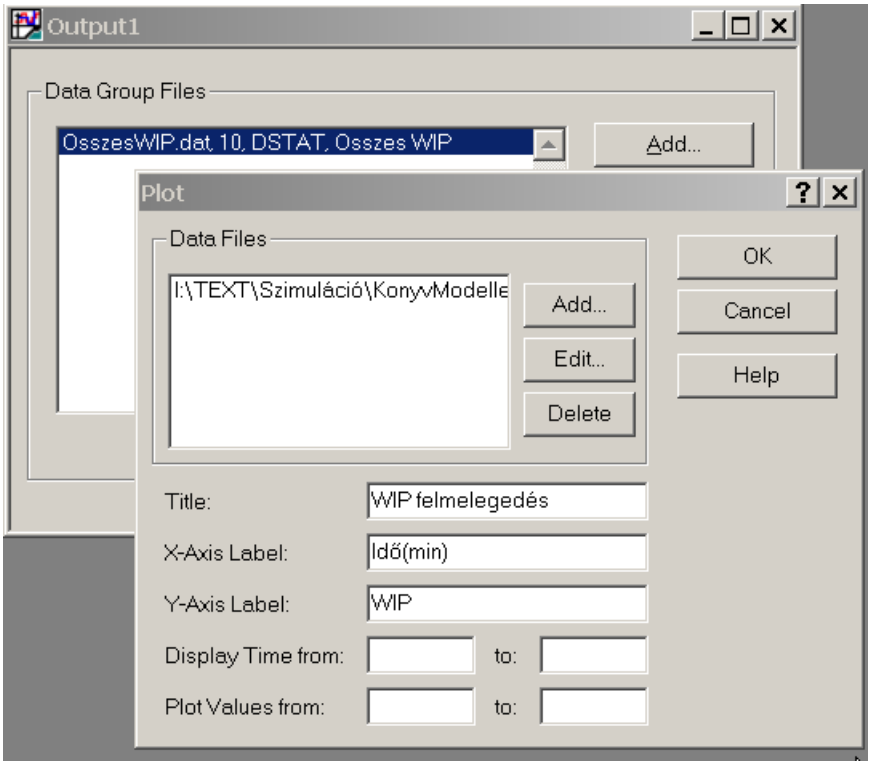
Azonban a riportban megjelenő „*Osszes WIP*” statisztika (átlag, minimum és maximum) helyett a változás dinamikáját szeretnénk követni az idő függvényében, azért hogy megállapíthassuk a felmelegedési periódus végét. Az „*Osszes WIP”* statisztikát ábrázoló animált grafikont elhelyezhetnénk a modell ablakban is, azonban ez a modell futásának leállításakor azonnal eltűnne, és talán előnyösebb lenne egy nagyobb grafikont rajzolni, amely segítené a stabilizációs pont megkeresésében. A grafikon adatait (a görbe történetét) elmentjük és az ismétlések számának a növelésével további görbéket rajzoltatunk, azért hogy csillapítsuk az eltérések zaját. Ehhez a **Statistic** adatmodulban az Output File mezőbe beírjuk a fájl nevét: OsszesWip.dat, amelybe elmentjük az adatainkat (*7.9. ábra*). Az adatokat később beolvashatunk az Arena **Output Analyzer** –be és ott grafikon formájában megjeleníthetjük azokat. A futás hosszától és az ismétlések számától függően ez a fájl elég nagy lehet, ezért kérdés, hogy szükségünk van-e nagyon részletes futási információra (az általunk mentett fájl mérete 176 Kb az alább közölt adatok esetén).

Mivel most nem vagyunk érdekeltek az animációban a *Run>Run Control > Batch Run* (*No Animation*) bekapcsolásával felgyorsítjuk a program futását. Tovább gyorsíthatjuk a programot a *Run>Setup>Project Parameters* ablakban és a **Dispose** modulban kikapcsoljuk a statisztikák gyűjtését. Ahhoz, hogy elegendő adatot kapjunk az ismétlések számát növeljük 10-re a *Run > Setup > Replication Parameters* ablakban a Number of Replication mezőben. Továbbá érdekeltek vagyunk a hosszú futásban, ezért futás hosszát (Replication Length) 32 óráról 120 órára változtatjuk. Ezeket az adatokat szabadon vehetjük fel. Az önkényes beállításokkal néhány próba és hiba után véglegesíthetjük a futási paramétereinket. A futtatás néhány másodpercet vesz igénybe.

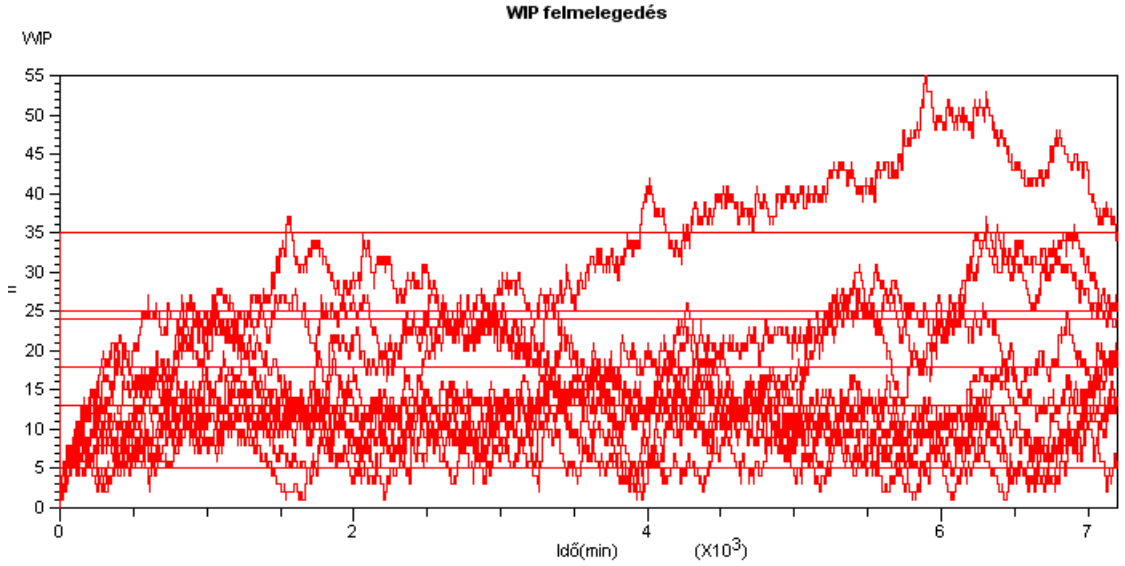
A WIP grafikon megrajzolásához indítsuk el az **Output Analyzer**-t. Zárjuk be az Arena programot és a Start menüből *Rockwell Software>Arena X.X > Output Analyzer* indítsuk el az **Analyzer**-t. Hozzunk létre egy új adatcsoportot (data group). A *File>New* menü opciókkal vagy a *Ctrl N* billentyű kombinációval nyissuk meg az **Output** ablakot (*7.10. ábra*), és az *Add* gombbal adjuk hozzá az OsszesWIP.dat adatfájlt. Az adatcsoportot későbbi felhasználásra mentsük a *File> Save As* menü opciókkal OsszesWIP.dgr néven. A *Graph>Plot* menü opciókkal vagy a () ikonnal nyissuk meg a **Plot** dialógusablakot. Először az *Add* gombbal adjuk a Data Files mezőhöz a OsszesWIP.dat adatfájlt, a Replications mezőben válasszuk a *Lumped* opciót majd töltsük ki a dialógusablak további mezőit: a címet (Title), a grafikon vízszintes és függőleges tengelyének a címkéit, az *OK* gombbal zárjuk be az ablakot. Az ablak bezárása után azonnal megjelenik a *7.11. ábrán* látható diagram.

A *7.11. ábra* mutatja az „*Osszes WIP*” statisztikából készített eredmény görbéket, az ismétlések számának megfelelően 10 egymást átfedő görbét. A grafikonból világosan látszik, hogy a futási hossz, 7200 perc elegendő a folyamat felmelegedéséhez, és az adatok állandósulásához. Az is egyértelmű, hogy a modell nem telítődik (nem robban fel), ami akkor következne be, ha a megmunkálás nem tudna lépést tartani az érkezésekkel. A felmelegedési periódusra az első néhány száz percben hatnak a kezdeti feltételek (az üres és tételen indítás), ezt követően egyértelműen egyenletessé válnak az ismétlések görbéi. Az állandósult, stacionárius állapot már

2000 perc után bekövetkezik, azonban legyünk egy kicsit konzervatívok, és válasszuk a felmelegedési periódus hosszát 2880 perc-re, ami 48 órának, illetve 2 napnak felel meg.



7.10. ábra: Az **Output Analyzer** és a **Plot** dialógusablak



7.11. ábra: A 7.2 modell WIP grafikonja

Ha futás (és a grafikon) hosszabb időperiódusra készül, vagy a modell túl gyorsan felmelegszik, nehézséget okozhat a felmelegedési periódus végének a meghatározása a görbe baloldalán, mert az adatpontok túl sűrűn követik egymást. Ha ez előfordul, akkor nagyítsuk fel a görbe bal oldalát amennyire csak tudjuk. Az adatokat exportálhatjuk szöveg fájlba, és importálhatjuk más adatfeldolgozásra alkalmas programokba, például az Excel-be.

Olyan modellekben, amelyeknél több egymástól eltérő kimeneti adatsort szeretnénk megjeleníteni és megismerni, minden kimenetre külön-külön kell a *7.11. ábrához* hasonló grafikont

készíteni. A kimenetek a felmelegedési periódust illetően különbségeket mutathatnak. Ilyenkor biztonságból az egyedi felmelegedési periódusok közül a maximálist választjuk.

7.2.2 A csonkított ismétlések (7.3. modell)

Ha meghatároztuk a felmelegedési periódust, és ez periódus indokolhatóan rövid a tervezett futási hosszhoz viszonyítva, akkor a stacionárius statisztikai analízis nagyon egyszerű. Futtassuk az IID (independent and identically distributed) ismétléseket, ahogyan azt korábban a véges szimulációknál tettük, azzal a különbséggel, most megadjuk a felmelegedés (Warm-up Period) hosszát *Run>Setup>Project Parameters* ablakban a Warm-up Period mezőben) minden ismétlésre. A megfelelő felmelegedési idő és futási hossz értékekkel egyszerűen futtassuk el a független ismétléseket, amelyek már nem tartalmazzák a felmelegedési periódus alatt keletkező statisztikákat. A kimeneteken megjelenő statisztikák tisztán a stacionárius működés alatt gyűjtött adatokat összegzik.

Ez az elv használható összehasonlításra, az alternatívák kiválasztására és az optimumkeresésre, és egyszerű rendszeranalízisre. A különböző alternatívák felmelegedési és futási hossza lényegesen eltérő lehet. Az OptQuest használatakor például nincs lehetőségünk arra, hogy kontroláljuk a különböző alternatívák közötti eltéréseket. Ilyenkor egyet tehetünk, hogy futtatjuk a programot a lehetséges input paraméterekkel és megpróbáljuk megkeresni a leghosszabb felmelegedési periódust.

A 7.2 modellben a futási idő 120 óta és 7.2.1 szakaszban ismertetett vizsgálat alapján a felmelegedési időt 2880 perc-re (ami 48 órának felel) választjuk és az ismétlések számát változatlanul hagyjuk. A **Statistic** adatmodulban az Output File mezőből törölhetjük a fájl nevét, mivel továbbiakban nincs szükségünk az „*Osszes WIP*” nevű statisztika részletes történetére. Ezt követően mentsük a modellt Modell7-3 néven. A 7.3. modellben a **Category Overview** > **User Specified** riport *Time Persistent* szekciójából kiolvasható eredmények a 95%-os konfidencia szinten: átlag (average) 16,39 ± 6,51, ugyanez az eredmény a 7.2. modellből 15,35 ± 4,42. A különbség jelzi a kezdeti feltételek torzító hatását. Tekintettel a konfidencia-intervallum kicsi szélességére annak érdekében, hogy a különbség érzékelhetőbb legyen, növeljük az ismétlések számát 110-rő 100-ra. Ekkor a következő eredményeket kapjuk: a 7.3. modellnél 15,45 ± 1,18, a 7.2. modellnél 14,42 ± 0,86. Vegyük észre, hogy azonos számú ismétlés esetén a konfidencia-intervallum szélesebb a 7.3. modellnél, mint a 7.2.-nél, ami annak köszönhető, hogy amíg a 7.3. modell minden ismétlésben csak az utolsó 72 óra adatait gyűjtötte, addig a 7.2. modell a teljes futási idő (120 óra) adatait. Az eredmények megfelelnek annak, amit a 2.13.4 szakaszban már jeleztünk, a csökkentett mintaszám, a kevesebb válasz általában növeli a középérték szórásnégyzetének becsült értékét. Ez az ára a jobb középértéknek.

Ha nagyobb pontosságot és keskenyebb konfidencia-intervallumot szeretnénk elérni, akkor két lehetőségünk van. (1) Változatlan felmelegedési és futási idő mellett növeljük az ismétlések számát, vagy (2) változatlan ismétlésszám mellett növeljük a futás hosszát. (Feltételezve, hogy az eredeti felmelegedési idő hossza adekvát volt.) Valószínűleg az egyszerűbb megoldás az ismétlések számának a növelése, amivel arányosan növeljük a mintavételek számát. El kell még mondani, hogy a futási hossz megnyújtásakor és az ismétlések számának a befagyasztásakor a növekvő pontosság nem a mintavételek számának (statisztikailag ez a szabadságfok) a növekedéséből, hanem az ismétlések átlagértékének csökkenő szórásnégyzetből adódik. A hosszabb futási idő a szórásnégyzetet csökkenti. Továbbá a futási idő növelésekor biztosabbak lehetünk abban, hogy a futás alatt a stacionárius szakaszból több adatot nyerünk.

Ha megfelelő futási hosszúságot és felmelegedési periódust határoztunk meg és a felmelegedési periódus nem túl hosszú, akkor a csonkított ismétlések stratégiája szimpatikus választás. Más stacionárius analízis stratégiákhoz képest egyszerű, és megbízhatóan független megfi-

gyeléseket biztosít (eredményeket a csonkított ismétlésekből), amelynek előnyei a statisztikai analízis készítésekor domborodnak ki. Ezek az előnyök nemcsak az egyszerű konfidencia-intervallum készítésekor jelentkeznek, hanem az alternatívák összehasonlításakor és a más statisztikai célok elérésekor is.

8. Az entitások szállítása

Eddig az entitásáramlások irányításának két különböző módszerét tekintettünk át. Az első módszer szerint, a modulokat összekapcsoljuk, és az entitások utazási idő nélkül mozognak az egyik modulból a másikba. A második módszer a *Route* és a *Station* fogalmakat használja, és a modellben az entitásokat előre definiált utazási időnek megfelelő késleltetéssel mozgatja. Az entitások mindkét esetben kötöttségek nélkül haladnak, a szállítópályákon egyidejűleg annyi kapacitás áll rendelkezésre, amennyi csak szükséges.

Természetesen a dolgok nem mindig ilyen egyszerűek. Gyakran előfordul, hogy az egyidejűleg szállítható entitások száma korlátozott, mint például a kommunikációs rendszerekben, ahol az entitások információcsomagok. A sávszélesség korlátozott, ami miatt egyidejűleg csak meghatározott számú csomag szállítható. Hasonló korlátozás előfordulhat akkor is, ha egy villástargonca, vagy egy dolgozó vesz fel és szállít egy entitást. A különböző konvejor típusokkal szállított entitások konkurenseknek tekinthetők, amelyek versenyeznek a szállítópályára kerülésért. E fejezetben bemutatunk néhány modellezési ötlet és képességet e problémák megoldására. Az entitásszállítás pontos modellezése nagyon fontos, mert a várakozás és a hatékonyság hiánya a műveletekben indokoltnál több időveszteséget okozhat. Például amikor csak járunk körbe-körbe, ahelyett, hogy az aktuális dolgunkat végeznénk.

A 8.1. alfejezetben részletesen tárgyaljuk a különböző entitásmozgásokat és szállításokat, valamint ezek modellezhetőségét. A 8.2. alfejezetben megmutatjuk, hogyan használhatók az **Arena** modellezési eszközök, amikor az egyidejűleg mozgatható entitások száma korlátozott. A 8.3. alfejezetben olyan szállítóeszközöket (villástargonca, vasúti kocsi, emberek, stb.) mutatunk be, amelyek felvesznek és szállítanak valamit. A különböző típusú konvejorok modellezését a 8.4. alfejezet ismerteti.

8.1. Entitás transzfer típusok

Az entitások modulok közötti mozgatására először a **Connect** opciót használtuk (3. fejezet), amely az entitásokat késedelem (időigény) nélkül szállítja az egyik modulból a másikba. A 4. fejezetben bevezettük a **Route** fogalmát, amely az entitások állomások közötti mozgatását teszi lehetővé, megengedve a mozgás időigényével megegyező késleltetést. Megmutattuk, hogyan használható a **Route** modul az entitások egy specifikus állomásra mozgatására, majd a 7. fejezetben általánosítottuk ezt a fogalmat a **Sequences** fogalom használatával.

Bár ezek az eszközök képesek az előforduló helyzetek többségének a modellezésére, néha azonban találkozunk olyan problémákkal, amikor a mozgatások számát bizonyos pontokon és időpontokban korlátozni kellene. Például egy kommunikációs hálózat modellezésekor a kapcsolatok (élek) kapacitásai korlátozottak lehetnek. Ezért szükségünk van egy olyan módszerre, amely korlátozza a hálózat egyes élein vagy a teljes hálózaton egyidejűleg mozgó üzenetek számát. A megoldás meglepően egyszerű, úgy képzeljük el a hálózat éleit, mint erőforrásokat, és amelyeknek a kapacitása egyenlő a megengedett párhuzamosan áramló üzenetek számával. Ha a szükséges kapacitás függ az üzenet nagyságától, akkor az üzenet nagyságának megfelelő nagyságú erőforrás kapacitást egy kifejezésben definiáljuk, és miden üzenettől megköveteljük, hogy továbbítás előtt kösse le az igényelt, az üzenet nagysága által meghatározott számú erőforrás kapacitást. Ezt a fajta entitásszállítást **erőforrás-korlátos entitás transzfernek** nevezzük, és amit a 8.2. részben részletesen megtárgyalunk.

Az erőforrással korlátozott egyidejű szállítás egy kommunikációs hálózaton megfelelően működik, de az anyagmozgatás körébe tartozó entitásmozgások valamennyi osztályának és kategóriájának pontos modellezését nem teszi lehetővé. A modellezés megköveteli az alkalmazkodást a különböző, nagymértékben eltérő anyagmozgató rendszerekhez és a különböző típu-

sú anyagmozgató gépekhez. Nem véletlen, hogy könyvek sora foglalkozik ezzel a témával (*Kulwiec*, 1985). Ennek ellenére a modellezési követelményeik alapján ezeket az eszközöket két általános kategóriába sorolhatjuk.

Az **első kategória** az egyidejű szállítások számát az egyedileg elérhető anyagmozgató gépek számával korlátozza. A kategóriába tartozó anyagmozgató gépek: a szállítókocsik, a targoncák, a villástargoncák, a vezetőnélküli targoncák, az emberek, stb. E módszer lényeges eleme, hogy az eszközökhöz fizikai helyeket rendelünk, és értelemszerűen az aktuális szállítás végrehajtása előtt, amikor a szállításigény jelentkezik, akkor először az eszközt arra a helyre kell mozgatni, ahol az entitás tartózkodik. A modellezés szemszögéből úgy képzelhetők el ezek az **Arena** környezetben **Transporter**-nek nevezett eszközök, mint mozgatható erőforrások.

A **második kategóriában** a szállítás korlátozásának alapja az elérhető hely. Ez szintén lehetővé teszi a két pont közötti egyidejű szállítások korlátozását, de ez a korlát tipikusan a helyigényen alapul. E kategóriába sorolt anyagmozgató eszközök a folyamatos üzemű anyagmozgatógépek: a konvejorok, felsőpályás kocsik, hajtott és gravitációs görgőpályák, vontató kötélpályák, stb. Például a mozgólépcső egy közismert anyagmozgatógép ebben a kategóriában. A szállításigény jelentkezésekor, mielőtt megkezdjük a szállítást, először az eszközön, azon a ponton, ahol éppen várakozunk, lekötjük a kívánatos mennyiségű elérhető, nem foglalt helyet. Ezek az **Arena** környezetben **Conveyor**-oknak nevezett eszközök az eddigiektől lényegesen különböző, sajátos modellezési képességekkel rendelkeznek.

Az Arena **Transporter** és **Conveyor** fogalmak segítségével gyakorlatilag majdnem mindenféle anyagmozgatási rendszer könnyen modellezhető. Bár létezik néhány anyagmozgató eszköz, amelyek egyedi tulajdonságai kihívást jelentenek a modellező számára. A portáldaruk és híddaruk klasszikus példái az ilyen eszközöknek. Egy önálló daru könnyen modellezhető transzporterként. Ha azonban egynél több daru mozog ugyanazon a pályán, a daruk pontos, kontrolált mozgatásának módja nagyon bonyolult. Szerencsétlenségünkre, majdnem minden rendszer, amelyben több daru is működik, általában a darun elhelyezett fülkében tartózkodó darukezelők által vezérelt. A darukezelők látják egymást és kommunikálnak egymással, és a döntéseik nem szükségszerűen megjósolhatók. Ebben az esetben a darut ugyan könnyű modellezni, de nagyon nehéz feladat szimulálni azt, hogy az emberi logika hogyan működik annak érdekében, hogy elkerüljék az összeütközést.

Ilyen módon háromféle entitástranszfer korlátozás definiálható: **erőforrással, transzporterrel és konvejorral** korlátozott entitásmozgatás. Először röviden az erőforrással korlátozott mozgatást tárgyaljuk a 8.2. alfejezetben, majd a 8.3 és 8.4 alfejezetekben transzportereket és konvejorokat alkalmazunk a 7. fejezetben megismert kisméretű rugalmas gyártási rendszerben.

8.2. Kisméretű rugalmas gyártórendszer erőforrás-korlátos szállítással (8.1 modell)

A korábban tárgyalt 7.1. modellben feltételeztük, hogy a rugalmas gyártási rendszerben minden szállítási művelet időtartama 2 perc. Ha a szállítási művelet függ az anyagmozgató eszközök elérhetőségétől, a tényleges szállítási idő ettől lényegesen eltérhet. A korábbi modellben a rendszer potenciális kapacitását egy ésszerű becsléssel adtuk meg, ami azonban nagy valószínűséggel nem adott pontos képet a rész ciklusidőkről. A pontosság javítására a legegyszerűbb módszer az **erőforrás-korlátos szállítás** bevezetése. Feltételezzük, hogy a szállítókapacitásunk kettő, minden esetben ugyanazzal a kétperces szállítási idővel. Azonban a szállításra kész entitásoknak addig kell várakozniuk, amíg a folyó szállítás be nem fejeződik, azaz a szállítóeszköz szabaddá nem válik.

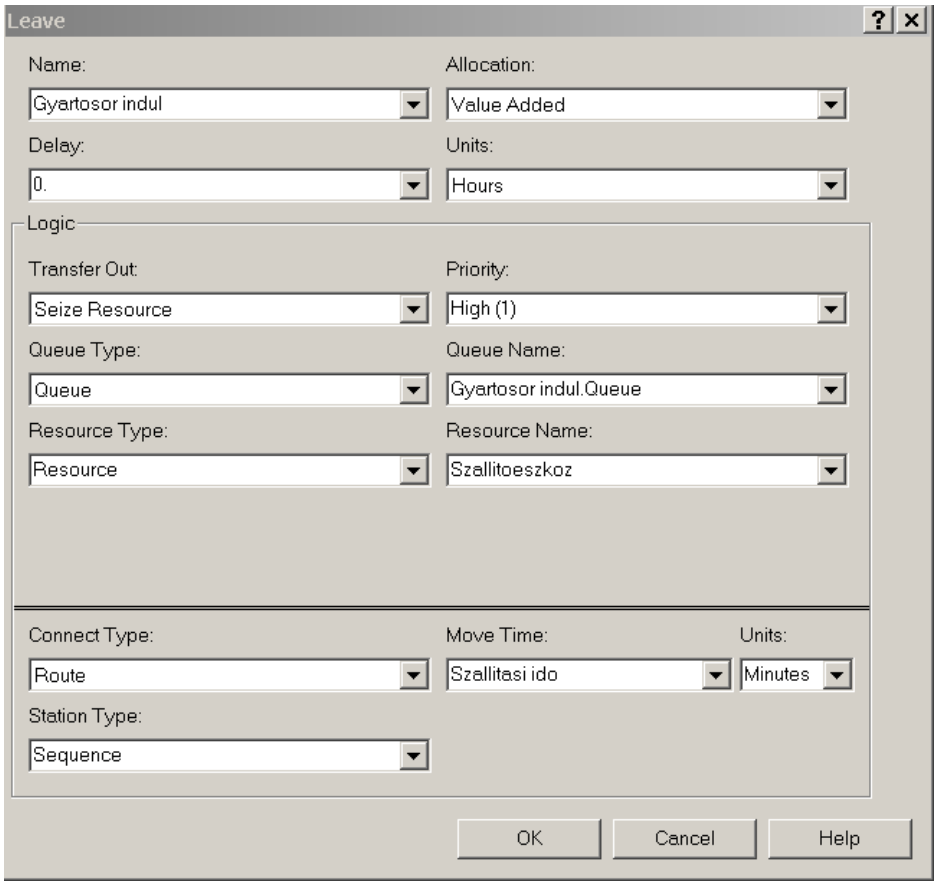
A modellben az egyidejű entitásszállítások korlátozása erőforrással relatíve egyszerű. Az újfajta **szállító erőforrás** kezelés menetét a következők szerint definiáljuk: először lekötjük az erőforrás egy egységét, mielőtt a helyváltoztatást célzó szállítást megkezdjük, majd felszabadítjuk az erőforrást, amikor a célállomásra vagy - helyre érkezünk. Ez egy lehetséges módja az **Arena** erőforrás kezelésnek, de elképzelhető ettől eltérő szállításmodellezés is.

A modellünkben a gondolatmenet megvalósításának két különböző megoldása létezik. Kézenfekvőnek látszik, hogy az **Advance Process** panelről egy **Seize** modult illesztünk a modellbe miden **Route** modult elé (5 ilyen modul található a jelenlegi modellünkben.) Minden **Seize** modul a szállító erőforrásból egy egységet igényel. Az új erőforrás, amelynek a neve legyen *„Szallitoeszkoz”*, megjelenik a **Resource data** modulban, ahol a kapacitását 2-re kell állítani. Létezik egy további fogalom is, amit figyelembe kell lenni, amikor a módosításokat elvégezzük. A **Seize** modulokhoz tartozik egy önálló sor. Ha minden újonnan beillesztett **Seize** modulban *Shared Queue* (osztott sort) használunk, akkor a szállító erőforrás a FIFO elv szerint allokálódik. Ha viszont minden **Seize** modulban önálló sorokat specifikálunk, akkor elérhetjük, hogy prioritást adjunk egy kiválasztott folyamatnak. Például egy újonnan érkező entitásnak prioritást akarunk adni, lehetővé téve, hogy az újonnan érkező munkadarabokat azonnal az első megmunkáló helyre küldjük, annak érdekében, hogy a munkadarab megmunkálása azonnal elkezdődjön, mert ez a munkadarab ciklusidejének a csökkenését eredményezi. (Ha időt szentelve rá, a folyamatot alaposan végig gondoljuk, akkor a konklúzió: ez egy hibás gondolkodás). Ha ugyanis minden prioritás egyforma, akkor az eredmény a FIFO elvhez vezet vissza, azaz a két közelítés ugyanaz. Ez nem egészen igaz, ha a várakozó munkadarabokat az animációban vizsgáljuk, ahogyan ezt meg is tesszük. Ha az önálló sort használjuk (ez az alapértelmezett) minden úthoz, akkor a várakozó munkadarabokat könnyen megjeleníthetjük az animációban.

Minden egyes munkadarab adott útvonalon való szállítása előtt lefoglalunk egy szállító erőforrást, ezt követően, amikor az erőforrás megérkezik a célállomásra, felszabadítjuk. Ezt úgy érjük el, hogy minden **Station** modul után beillesztünk egy **Release** modult a modellbe (kivétel az „*Rendezes es tovabbitas allomas*” amelyet csak arra használunk, hogy meghatározzuk az újonnan érkező munkadarabok helyét). Mint látni fogjuk, a **Release** modulok egyszerűen felszabadítják a szállító erőforrást.

A második megoldásban a **Route** modulokat **Leave** modulokkal és a **Station** modulokat **Enter** modulokkal (**Advanced Transfer** panelről) helyettesítjük. A **Leave** modul egy modulban teszi lehetővé egy erőforrást lefoglalását és a munkadarab utaztatását. Hasonlóan, az **Enter** modul a **Station** és a **Release** modulok tulajdonságait kombinálja. A továbbiakban azt a második módszert választjuk, mert ez a modell átalakításán kívül lehetőséget biztosít két új modul bevezetésére és más szállítási tulajdonságok bemutatására.

A **Leave** modul lehetővé teszi, hogy már az állomás elhagyása előtt lekössük a szállító erőforrást. Az is definiálható, hogy melyik egyedi erőforrás legyen lekötve. Az egyedi erőforrás lehet egy erőforrás halmaz (resource set) specifikus tagja, vagy a kiválasztás alapja lehet egy kifejezés kiértékelése is. A Transfer Out (kiszállítás) logika szintén definiálható, amely lehetőséget biztosít arra, hogy megadjuk, hogy melyik entitás legyen hozzáadva a szállító erőforráshoz, ha több mint egy entitás várakozik. A kisebb számú, a nagyobb prioritású, stb. A rendszerünkben mi egyszerűen a *Seize Resource* opciót választjuk és definiáljuk a szállító erőforrás nevét. Az utat jellemző további információkat Connect type, Station Type, Move Time, stb. ugyancsak meg kell adni.



8.1. képernyő: „*Gyartosor indul*” nevű **Leave** modul

8.1. táblázat

A „*Gyartosor indul*” nevű **Leave** modul paraméterei

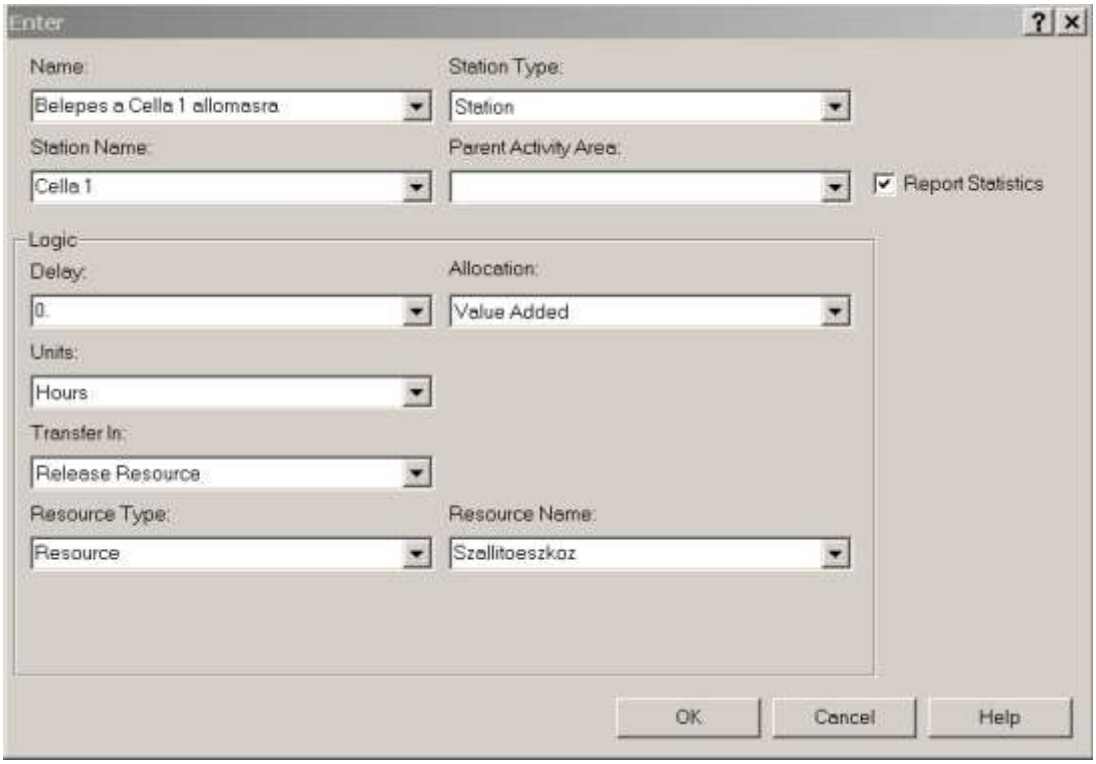
Name	Gyartosor indul
Logic Transfer Out Resource Name	 Seize Resource Szallitoeszkoz
Connect Type Move Time Units Station Type	Route Szallitasi ido Minutes Sequence

Kezdjünk azzal, hogy a „*Gyartosor indul*” nevű **Route** modult egy **Leave** modulra cseréljük, ami a *8.1. képernyőn* látható. A modul nevét nem változtatjuk („*Gyartosor indul*”), a Transfer Out mezőhöz a *Seize Resource*–t választjuk és a Resource Name mezőbe beírjuk a „*Szallitoeszkoz*” az erőforrás nevet. Végül a Connect Type mezőben a *Route* opciót választjuk, és a Move Time mezőbe „*Szallitasi ido*” kifejezést írjuk, aminek a mértékegysége min, és a Station Type mezőben a *Sequence* opciót választjuk. A többi mezőben meghagyjuk az alapértelmezett értékeket. Vegyük észre, hogy ezek az adatok a szállító erőforrásra várakozó munkadarabokat individuális sorokba rendezik. A másik megjegyzés, a Transfer Out erőforrás logikával kapcsolatban, a sor és a sor neve (*Gyartosor indul.Queue*) automatikusan adódik a modulhoz, amikor a dialógusablakot az *OK* gombbal bezárjuk. Ezután a sort megjelenítő alakzatot az animációban a megfelelő helyre mozgathatjuk.

Ezt követően a négy megmaradt **Route** modult is cseréljük **Leave** modulokra. A korábbi modul neveket elfogadjuk (ugyanazokat a modulneveket használjuk, mint a **Route** modulokban) a további adatok megegyeznek a *8.1. képernyőn* látható és a *8.1. táblázatban* közölt adatok-

kal.

Az **Enter** modul lehetővé teszi a szállító erőforrás felszabadítását és lehetőséget ad a lerakási idő megadására is a Delay mezőben. Helyettesítsük először a „*Belepes a Cella 1 allomasra*” nevű **Station** modult egy **Enter** modullal (*8.2. képernyő*). Töröljük a meglévő **Station** modult és illesszük be az új **Enter** modult. A modul nevét hagyjuk változatlanul, azaz válasszuk a lenyíló listából a „*Belepes a Cella 1 allomasra*” nevet. A logic területen a Transfer In (beszállítás) mezőhöz a *Release Resource,* a Resource Type mezőhöz a *Resource* opciót válasszuk, végül a Resource Name a **Leave** modulban lekötött „*Szallitoeszkoz*” legyen, amit ebben modulban szabadítunk fel.



8.2. képernyő: „*Belepes a Cella 1 allomasra*” nevű **Enter** modul

Cseréljük a maradék négy **Station** modult **Enter** modulokra (kivétel a „*Rendezes es tovabbitas allomas*” nevű állomás). A **Enter** modulok paraméterei a modul nevek kivételével megegyeznek a *8.2. képernyőn* látható értékekkel. A modulok nevei legyenek azonosak a lecserélt **Station** modulok neveivel.

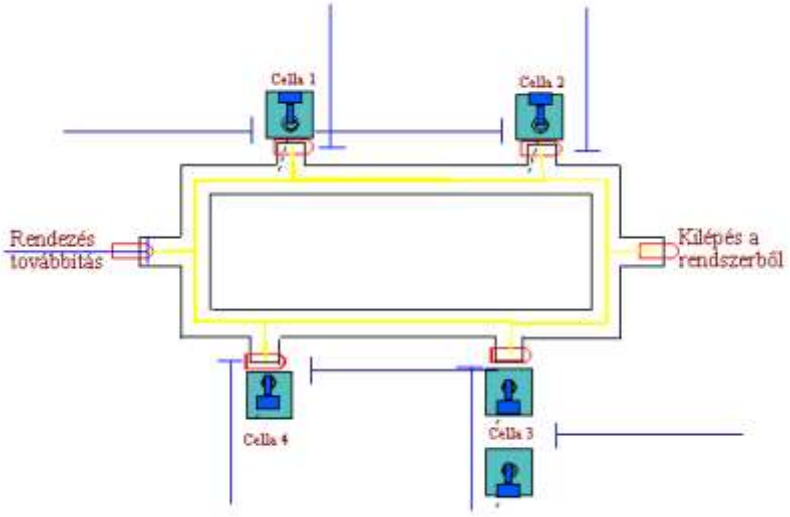
8.2. táblázat

A „*Belepes a Cella 1 allomasra*” nevű **Enter** modul paraméterei

Name Station Name	Belepes a Cella 1 allomasra Cella 1
Logic Transfer In Resource Type Resource Name	 Release Resource Resource Szallitoeszkoz

Miután gondoskodtunk arról, hogy várakozó sorok az animációba kerüljenek, majdnem készen állunk a modell futtatására. Mielőtt ezt megtesszük, tudnunk kell, hogy még nem definiáltuk a korlátozott kapacitású szállító erőforrás kapacitását, amely meghatározza az egyidejűleg maximálisan szállítható munkadarabok számát. Bár már a **Leave** modulban a Resource Name megadásakor („*Szallitoeszkoz*”) definiáltuk az erőforrást, amelyhez az Arena az alapér-

telmezett 1 kapacitást rendelte. A változtatáshoz nyissuk meg a **Resource data** module a **Basic Process** panelen, és az erőforrás kapacitását változtassuk 2-re. Az eredményül kapott modell hasonlóan néz ki, mint a 7.1 modell, mert a modulcsere során a modulok száma nem változott, ráadásul az új modulok nevei is megegyeznek a lecserélt modulok neveivel. Az animáció is hasonló a hozzáadott az öt várakozó sorral. Ezek a sorok a *8.1. ábrán* láthatók.



8.1. ábra: Az átalakított kisméretű gyártórendszer animációja

Ha a modell futása közben, az animációt figyeljük, gyorsan észrevehetjük, hogy a munkadarabok várakozása a szállítóeszközre ritkábbá vált. Ha kételkednénk a modell korrektségében, változtassuk a szállítókapacitást 1-re, és figyeljük meg újra az animációt. Ha összehasonlítjuk az új modell eredményeit (2 erőforrás kapacitással) a *7.1. modell* eredményeivel, akkor azt tapasztaljuk, hogy egy munkadarabra eső összes idő (*Total Time*) kicsit növekszik, köszönhetően a szállító erőforrás sorban eltöltött idejére. A különbség 3 és 9 perc között változik a munkadarab típusától függően.

8.3. Kisméretű rugalmas gyártórendszer transzporterekkel

Feltételezzük, minden anyagmozgatási feladat elvégezhető valamilyen eszközzel, pl. szállítókocsival, kézi emelőkkel, villástargoncával, stb. (a modellben nevezzük ezeket kézikocsiknak). Továbbá feltételezzük, hogy ezek a kézikocsik 50 m/min sebességgel mozognak teherrel és üresen egyaránt. Minden kézikocsi egyidejűleg 1 munkadarabot tud szállítani, valamint a fel- és lerakási idő 0,25 perc a kezdő és a célállomásokon. Ezenkívül, ismerjük az állomások közötti távolságokat, amelyekre szükségünk lesz, amikor a kocsikat beépítjük a modellünkbe.

Az **Arena**-ban két transporter (szállítóeszköz) típus létezik: a kötetlenpályás (*Free-Path*) és a kötöttpályás (*Guided*). A kötetlenpályás transzporterek szabadon mozoghatnak a rendszerben a torlódások okozta késés nélkül. Az utazási idő egyik pontból a másikba a szállítóeszköz sebességétől és a megtett távolságtól függ. A kötöttpályás szállítóeszközök csak az előre definiált hálózaton mozoghatnak. Ezeknél az utazási idő függ a szállítóeszköz sebességétől, hálózaton kijelölt útvonaltól, és az útvonalon előforduló torlódásoktól. A legismertebb kötöttpályás szállítóeszköz a vezetőnélküli targonca. A kézikocsik a rendszerünkben a kötetlenpályás kategóriába sorolhatók.

A munkadarabok szállítása transzporterekkel három tevékenységet igényel. A szállítóeszköz igénylését, a munkadarab szállítást és a szállítóeszköz felszabadítását. A kulcs szavak: **Request** (igénylés) **Transport** (szállítás) és **Free** (felszabadítás). A **Request** tevékenység analóg a **Seize** (erőforrás lekötés) tevékenységgel, amely allokál egy elérhető szállítóeszközt

az igénylő entitáshoz és az allokált szállítóeszközt az entitás helyéhez mozgatja, hacsak nem tartózkodik ott. A **Transport** (szállítás) tevékenység az entitást a célállomásra mozgatja. Ez analóg a **Route** tevékenységgel, de szállítás esetén a szállítóeszköz és az entitás együtt mozognak. A **Free** tevékenység szabaddá teszi a szállítóeszközt a következő igénylés számára, ez nagyon hasonlít a **Release**, erőforrás felszabadításhoz. Ha több szállítóeszköz van a rendszerben, akkor az entitásokhoz való hozzárendeléssel kapcsolatban két kérdéssel szembesülhetünk. Az első, előfordulhat olyan szituáció a futás alatt, amikor egy entitás szállítóeszközt igényel és egyszerre több szállítóeszköz is elérhető. Ebben az esetben a *Transporter Selection Rule* (szállítóeszköz kiválasztási szabály) határozza meg, hogy melyik szállítóeszköz legyen a kiválasztott. A legtöbb modellben használt *Smallest Distance* (legkisebb távolság) szabály azt a szállítóeszközt allokálja, amely a legközelebb van az igénylő entitáshoz. Más szabályok, beleértve a *Largest Distance* (legnagyobb távolság) szabályt is, kreatív gondolkodásra késztetnek. Az adott esetben a legérzékenyebb szabály kiválasztása nem könnyű feladat. A második kérdés a szállítóeszköz allokációval kapcsolatban merül fel, amikor a szállítóeszköz szabad és arra több entitás várakozik. Ebben az esetben az Arena a prioritást használva, a szállítóeszközt ahhoz a várakozó entitáshoz allokálja, amelynek a prioritása legnagyobb (ez a legalacsonyabb prioritási számú entitás). Ha a prioritás egyforma, akkor a szállítóeszköz a legközelebbi várakozó entitáshoz allokálódik.

8.3.1. A transzporterekkel módosított modell (8.2. modell)

Ahhoz hogy a szállítóeszköz megjelenjen a kisméretű gyártórendszerben, először definiáljuk a kézikocsit. Ezt megtehetjük az **Advanced Transfer** panelen található **Transporter data** modul megnyitása után. Először nyissuk meg a *8.1 modellt* tartalmazó Modell8-1 nevű fájlt és a módosítások előtt mentsük Modell8-2 néven. A **Transporter** adatmodul megfelelő mezőibe, Name (név), Capacity (kapacitás=a kocsik száma) és Velocity (sebesség) írjuk be az értékeket (*8.3. képernyő*). Az **Advanced Transfer** panelen a szállítóeszközt definiáló **Transporter** adatmodul kötetlenpályás szállítást feltételez. A Distance Set név mezőben, továbbá az Units (időegység), Initial Position (szállítóeszköz kezdeti pozíciója) és a Report Statistics mezőkben elfogadjuk az alapértelmezett nevet és értékeket (*8.3. képernyő*).

A sebesség definiálásakor figyelni kell arra, hogy az megfeleljen az általunk megadott idő és távolság egységeknek (pl. ha az idő percben, a távolság méterben adott, akkor a sebesség mértékegysége m/min).

Transporter - Advanced Transfer

	Name	Capacity	Distance Set	Velocity	Units	Initial Positions	Report Statistics
1	Kezi kocsi	2	Kezi kocsi.Distance	50	Per Minute	0 rows	☑

8.3. képernyő: A **Transporter** adatmodul

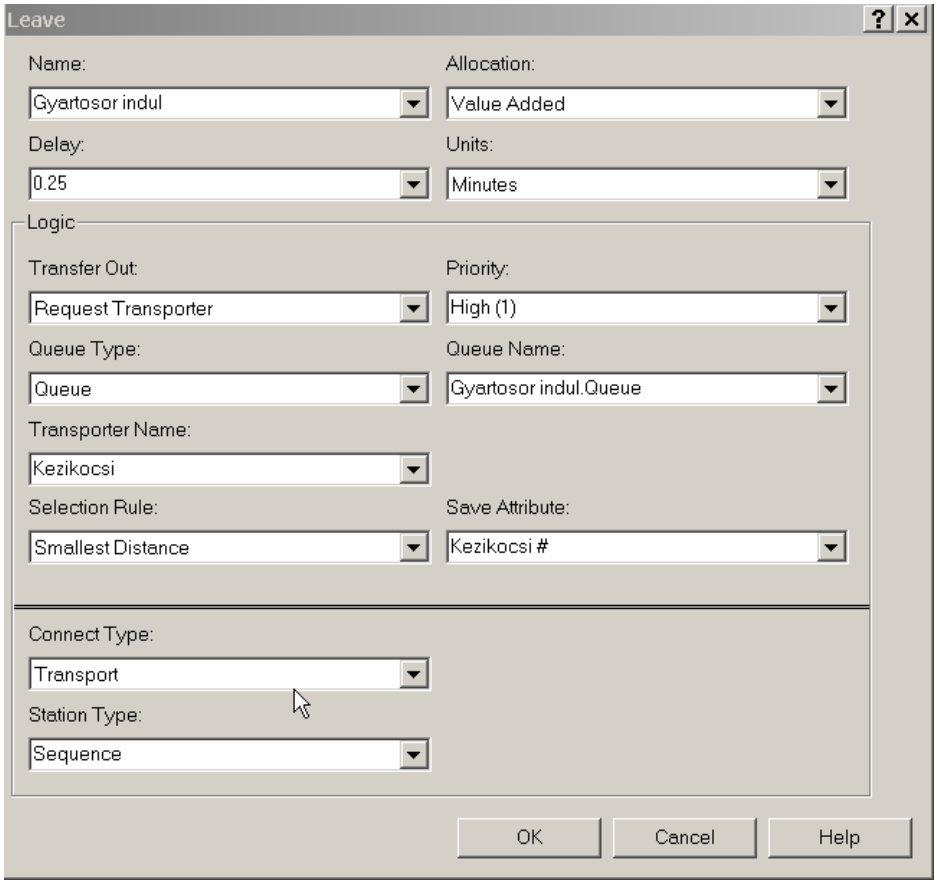
8.3. táblázat

A **Transporter** adatmodul paraméterei

Name	Kezikocsi
Capacity	2
Velocity	50

A transzporter definiálás után a kézikocsi megjelenítéséhez szerkesszünk egy képet. Ez majdnem hasonló módon történik, mint az entitás és erőforrásképek szerkesztése. Klikkeljünk a *Transporter* gombra ( ) az *Animate Transfer* eszköztáron, és nyissuk meg a **Transporter Picture Placement** (Transzporter képelhelyező) ablakot. Az alapértelmezett képet, cseréljük egy zöld vonallal határolt fehér négyzetre (a kerethez 5 pont szélességű vonalat használjunk), amikor a kézikocsi *Idle* (tétlen) és kék vonallal határoltra, amikor a kézikocsi *Busy* (foglalt).

Amikor a kocsi szállít egy munkadarabot az **Arena**-nak tudnia kell, hogy a kézikocsi *Busy* képén hová pozícionálja a munkadarabot. Ez a viszonyítási pont hasonló, mint az erőforrás *seize* (megfogási) pontja. A transzportereknél ezt a pontot *ride* (utazási) pontnak nevezzük. A *ride* pont csak a *Busy* képre szerkeszthető. A *ride* pontelhelyező parancs az **Arena Picture Editor** ablakában érhető el az *Object* menüben, amikor a *Busy* képet szerkesztjük. Kiválasztva az *Object >Ride Point* parancsot a kurzor keresztre változik. Mozgassuk a keresztet a kézikocsit szimbolizáló négyzet közepére és klikkeljünk. A *ride* pont a *Busy* állapotú kézikocsin ⊠ szimbólumként jelenik meg, és azt eredményezi, hogy az entitás hivatkozási pontja a transzporter *ride* ponthoz lesz pozícionálva.



8.4. képernyő: „*Gyartosor indul*” nevű **Leave** modul

8.4. táblázat

A „*Gyartosor indul*” nevű **Leave** modul paraméterei

Name Delay Units	Gyartosor indul 0,25 Minutes
Logic Transfer Out Transporter Name Selection Rule Save Attribute	 Request Transporter Cart Smallest Distance Kezikocsi #
Connect Type Station Type	Transport Sequence

Amikor befejezzük a képszerkesztést, a változásokat elfogadjuk, bezárjuk a **Transporter Picture Placement** ablakot a pointer keresztre változik és a kereszt az animáció közelében pozí-

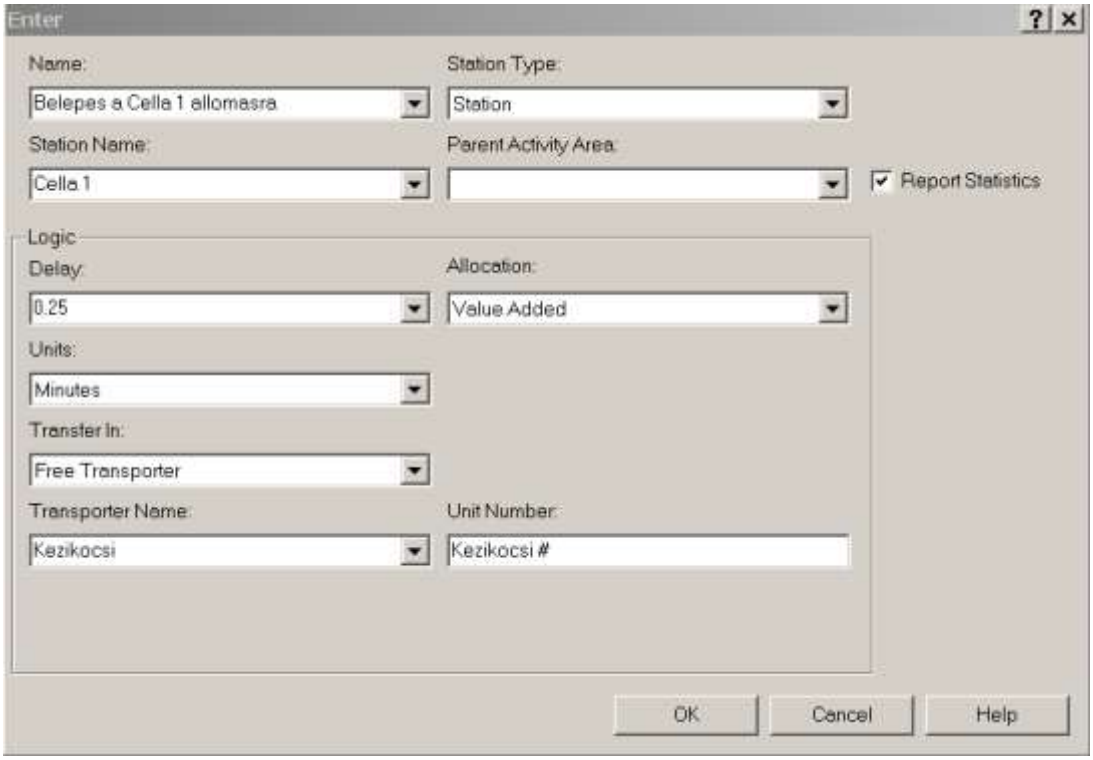
cionálható. Ez az művelet az *Idle* kézikocsi képét a modellablakba helyezi. A modellablakba helyezett transzporter kép később, ha szükséges, újraszerkeszthető. A modell futása alatt a kép rejtett lesz, és csak a replikációi mozognak az animációban, mint önálló transzporterek.

A modell logikában a kézikocsi igényléshez ugyanazokat a modulokat használjuk, amelyeket az előző modellben az erőforrás lekötéshez, vagyis a **Leave** modulokat. A *8.4. képernyő* mutatja a „*Gyartosor indul*" nevű **Leave** modulba beírt értékékeket. A további **Leave** modulok paraméteri is azonosak a Name (név) kivételével. A Delay mezőben a 0,25 perc a rakodási időt jelenti. A Transfer Out mezőben a *Request Transporter* opciót választottuk. A Selection Rule opció meghatározza, melyik kézikocsi legyen allokálva, ha az adott időpontban több kézikocsi szabad. A választható opciók: *Cyclic* (ciklikus), *Random* (véletlen), *Preferred Order* (elsőbbségi rendben), *Largest Distance* (legnagyobb távolság), és *Smallest Distance* (legkisebb távolság). *Cyclic* (ciklikus) szabály a transzporterek közel egyenletes kihasználását próbálja elérni, olyan módon, hogy a transzportereket körforgásszerűen allokálja. A *Preferred Order* (elsőbbségi rendben) szabály mindig a legkisebb sorszámú, elérhető transzportert próbálja kiválasztani. A modellben a *Smallest Distance* (legkisebb távolság) szabályt választjuk, amely azt eredményezi, hogy a szállítást igénylő entitáshoz legközelebb eső kézikocsi lesz allokálva. A Save Attribute mezőhöz egy új attribútumot, (*Kezikocsi #*) definiáltunk és használtunk, azért hogy megőrizzük az allokált kézikocsi számát (erről később többet szólunk). A Connect Type mezőhöz a *Transport* opciót választottuk. Vegyük észre, amikor ezt az opciót választjuk értelemszerűen a Move Time mező eltűnik, mivel az aktuális utazási idő a megtett távolságból és a transzporter sebességéből lesz kiszámítva.

A **Leave** modul négy műveletet végez: allokálja a transzportert, az üres transzportert a szállítást igénylő munkadarabhoz mozgatja, végrehajtja a rakodási időnek megfelelő késleltetést és a munkadarabot a célállomásra mozgatja. E műveletet elvégezésének vannak alternatív megoldásai. Például, használhatjuk a **Request–Delay–Transport** modulok kombinációját. A **Request** modul (**Advanced Transfer** panel) elvégzi az első kettő műveletet, a **Delay** modul a rakodást és a **Transport** module (**Advanced Transfer** panel) az utolsó műveletet. Bár ez a megoldás három modult igényel, előnye, hogy növeli a modellezés rugalmasságát. Például az üres és a terhet szállító transzporter sebessége különböző lehet. Tulajdonképpen a szállítási sebességet egy kifejezéssel is meghatározhatjuk, amely értéke a munkadarab valamelyik attribútumától függ, például a sebesség függhet a munkadarab súlyától. A **Delay** modult egy **Process** modullal is helyettesíthetjük, ami lehetővé teszi, hogy az rakodási (anyagmozgatási) művelethez erőforrást (operátort, gépet, stb.) rendeljünk. Végül a **Request** modult egy **Allocate–Move** modulkombinációra cserélhetjük. Az **Allocate** modul (**Advanced Transfer** panel) allokálja a transzportert, a **Move** modul (**Advanced Transfer** panel) pedig az üres transzportert a szállítást igénylő munkadarabhoz mozgatja. Ilyenkor e két művelet közé további tevékenységeket illeszthetünk, ha valamilyen logika vagy a modellezés pontossága ezt igényli.

Amikor egy munkadarab megérkezik a célállomásra, a kézikocsit felszabadítjuk és elérhetővé tesszük más munkadarabok számára. A kézikocsi felszabadítására ugyanazt az **Enter** modult használjuk, mint amelyikkel az előző modellben felszabadítottuk az erőforrást. A *8.5. képernyő* és a *8.5 táblázat* mutatja a modulba beírt és a választott paramétereket. Az **Enter** modulok adatai a nevek kivételével teljesen azonosak. A lerakodási időt percben adjuk meg a Delay mezőben, és a *Free Transporter* opciót választottuk a Transfer In mezőben. A modellben a Transporter Name a korábban definiált „*Kezikocsi*", és a *Unit Number* a kézikocsi attribútuma (*Kezikocsi #*), amely a kézikocsi sorszáma. Az utóbbi két mezőt üresen is hagyhatnánk. Az **Arena** ugyanis számon tartja, hogy melyik kézikocsi van allokálva egy entitáshoz és melyik szabad. Ha azonban egynél több transzporter működik a rendszerben és megadjuk a transzporter nevét, akkor a Unit Number mező sem maradhat üres. Amikor megadjuk egy

transzporter nevét az **Arena** mindig az első transzportert próbálja szabaddá tenni.



8.5. képernyő: „*Belepes a Cella 1 allomasra*” nevű **Enter** modul

8.5. táblázat

A „*Belepes a Cella 1 allomasra*” nevű **Enter** modul paraméterei

Name	Belepes a Cella 1 allomasra
Station Name	Cella 1
Logic	
Delay	0,25
Units	Minutes
Transfer In	Free Transporter
Transporter Name	Kezikocsi
Unit Number	Kezikocsi #

Az **Enter** modul is, úgy mint a **Leave** modul, több műveletet integrál: definiálja az állomást, a lerakodási időnek megfelelő késleltetést végez, és felszabadítja a transzportert. Ezeknek a funkcióknak a modellezésére itt is léteznek alternatívák. Az **Enter** modul helyett használhatjuk a **Station–Delay–Free** modulkombinációt. A **Station** modul definiálja az állomást, a **Delay** modul a lerakodás okozta késleltetést, és a **Free** modul (**Advanced Transfer** panel) felszabadítja a transzportert. A három művelet szétválasztása további műveletek beillesztését teszi lehetővé, ha az szükséges. Például egy **Delay** modult **Process** modulra cserélhetünk, amelyben operátort rendelhetünk a lerakodási művelethez.

Eddig a pontig definiáltuk a kézikocsit és a *Request* (igénylés), *Transport* (szállítás) és *Free* (felszabadítás) műveletek bevezetésével megváltoztattuk a modell logikát, amely lehetővé teszi a munkadarabok szállítását a rendszer egyik pontjáról a másikra.

Az aktuális utazási idő függ a transzporter sebességétől (a kézikocsival együtt definiáltuk a **Transporter** adatodulban) és a szállítási távolságtól. Amikor definiáltuk a kézikocsit (8.3. képernyő), elfogadtuk az *Distance Set*-hez (távolsághalmaz) rendelt alapértelmezett nevet: „*Kezikocsi.Distance*”. Most a halmaz elemeit kell megadni, amelyeket az Arena akkor hasz-

nál, ha szállítás történik. Tekintsük a munkadarabok mozgását az egyik állomásról a másikra, amint áthaladnak a rendszeren (*7. 1. táblázat*). E táblázat felhasználásával készült a *8.6. táblázat*, amelyben az állomás vagy hely adatpárok és az állomások közötti távolságok (m) láthatók. Például az első adat 37 m, a „*Rendezo allomas*” és a „*Cella 1*” állomások közötti távolság. Az üres cellák azt jelzik, hogy a relációban nincs munkadarabmozgás.

8.6. táblázat

A munkadarab-mozgások távolságai

Honnan/Hová	Cella 1	Cella 2	Cella 3	Cella 4	Kilépés a rendszerből
Rendező állomás	37	74			
Cella 1		45	92		
Cella 2	139		55	147	
Cella 3				45	155
Cella 4		92			118

A távolságokat az **Advanced Transfer** panelen található **Distance** adatmodulba lehet beírni. A 11 adat a *8.6. képernyőn* látható. Az adatbevitel során csak a távolságokat kell beírni, mert az állomások nevei a lenyíló listákból kiválasztatók. Vegyük észre, hogy mozgás iránya *Beginning Station→Ending Station*, például az első sorban „*Rendezo allomas*”*-ról→*”*Cella 1*”*-be* (*8.6. képernyő*). Definiálhatjuk a „*Cella 1*”*-ből→* „*Rendezo allomas*”*-ra* távolságot is, ha az oda-vissza út különbözik (de ezt az utat a modellben nem használjuk, ezért a megadása nem szükséges).

Stations

	Beginning Station	Ending Station	Distance
1	Rendezo allomas	Cella 1	37
2	Rendezo allomas	Cella 2	74
3	Cella 1	Cella 2	45
4	Cella 1	Cella 3	92
5	Cella 2	Cella 1	139
6	Cella 2	Cella 3	55
7	Cella 2	Cella 4	147
8	Cella 3	Cella 4	45
9	Cella 3	Kilepes a rendszerbol	155
10	Cella 4	Cella 2	92
11	Cella 4	Kilepes a rendszerbol	118

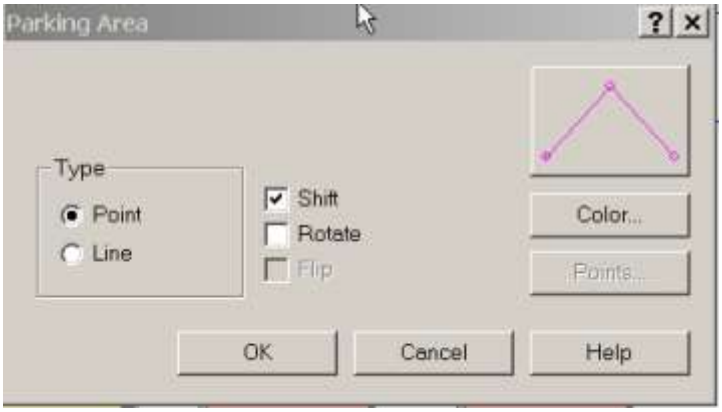
8.6. képernyő: A **Distance data** modul a munkadarabmozgások távolságaival

Most vessünk egy pillantást a távolságokhoz tartozó animációs komponensekre. Már korábban definiáltuk a transzporter (kézikocsi) képét, de az animációhoz meg kell adni a távolságokat is. Eljött az ideje annak, hogy az animációból töröljük az utakat (az állomások maradnak) mielőtt a távolságokat beillesztjük. Használjuk a *Distance* gombot () az *Animate Transfer* eszköztáron, klikkeljünk rá, és adjuk a távolságot az animációhoz. Ezt annyiszor ismételjük, ahány távolság szerepel a listában (a modellben 11). Amikor egy munkadarabot szállítunk, a kézikocsi és a munkadarabok követik az animációba rajzolt az útvonalakat, vagyis a távolságokat. Amikor a *Distance* gombra klikkelünk megjelenik a dialógusablak, amely lehetővé teszi, hogy módosítsuk az út karakterisztikát úgy, ahogy az a *8.7. képernyőn* látható. Azt szeretnénk, hogy sem a kézikocsi, sem a munkadarab, amely a kézikocsin utazik, ne forogjon vagy pattogjon mialatt az útvonalon mozog. Ezek a tulajdonságok nem szükségesek, mivel a képek könnyen jól azonosítható négyzetek és körök. A munkadarabokra számokat helyezünk

a könnyebb azonosíthatóság érdekében. A munkadarab képek átalakítását az olvasókra bízzuk.

8.7. képernyő: A **Distance** dialógusablak

Miközben a 11 távolságot megrajzoljuk, megfigyelhetjük a *View* eszköztáron elérhető *Grid* and *Snap* parancsok működést. Tanácsos a távolságokat olyan sorrendben rajzolni, ahogy azok a **Distance** adatmodul listában követik egymást. Így elkerülhetjük a keveredést és csökkentjük annak lehetőségét, hogy valamelyik távolság kimarad az animációból. Ha úgy találjuk, hogy valamelyik útvonal nincs a helyén, akkor könnyen újra szerkeszthetjük. Klikkeljünk a szóban forgó *Distance* vonalra (útvonalra), amely fényesre változik és megjelennek az alakzatformálást szolgáló töréspontok a vonalon. Mozgassuk óvatosan az pointert valamelyik pont fölé, ekkor a pointer alakja keresztre változik. Ezután az egér balgombját lenyomva tartva, fogjuk meg a pontot és mozgassuk a kívánatos helyzetbe. Ha a pointer alakja nyíl, akkor az egérgomb lenyomásakor a vonal belső pontjai elmozdulhatnak. Ha ez történne, akkor használjuk az *Undo* parancsot. További pontok hozzáadásához, vagy pontok törléséhez klikkeljünk kétszer az útvonalra, ami megnyitja a dialógusablakot, ahol a #Point mezőben tetszőlegesen változtathatjuk a pontok számát. Amikor az útvonal pontjait növeljük vagy csökkentjük, akkor mindig a végpont után illeszkedik az új pont, illetve a végpontot töröljük.

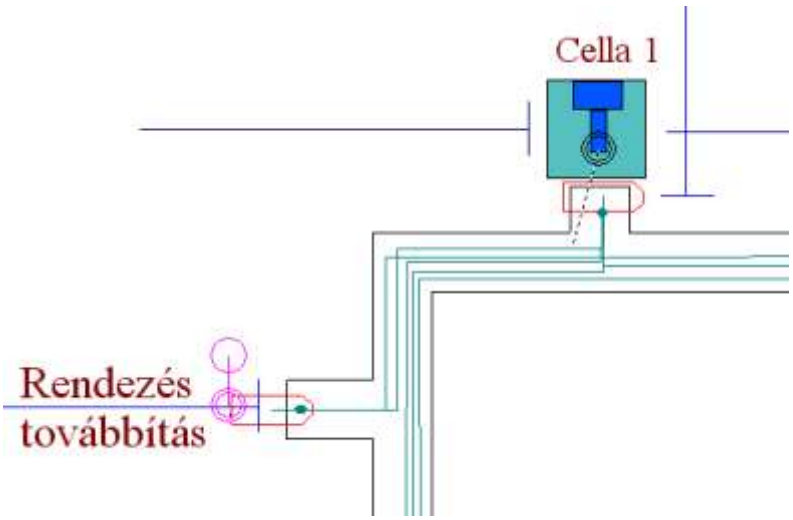


8.8. képernyő: A **Parking Area** dialógusablak

A modell futásakor megfigyelhetjük, hogy amikor az animációban a munkadarabok és a kézikocsi együttmozognak, valamint a kézikocsi szabaddá válik, akkor eltűnnek az animációból egészen addig, amíg a kézikocsi újra nem allokálódik. Ez azért van, mert nem mondtuk meg az **Arena**-nak, hogy a transzporterek hol tartózkodjanak, amikor *Idle* (tétlen) állapotban vannak. Ezt a problémát kiküszöbölhetjük, ha parkoló területet adunk az animációhoz. Klikkeljünk a *Parking* gombra () az *Animate Transfer* eszköztáron, amely megnyitja a **Parking** dialógusablakot (*8.8. képernyő*). A dialógusablak elfogadása (*OK*) után a kurzor keresztre változik. Pozícionáljuk a pointert az animáción egy állomás balalsó sarkához és klikkeljünk. Amint mozgatjuk a pointert, láthatjuk, hogy egy összekötővonal rajzolódik az állomás és a

pillanatnyi pointer pozíció között. Ha nem ez történik, akkor klikkeljünk újra. Miután a pointert a parkolási területnek megfelelő helyre pozícionáltuk klikkeljünk, és mozgassuk a pointert a második transzporter parkolási területére és klikkeljünk kétszer. A dupla klikkelés hatására befejeződik a parkolási terület szerkesztése, és a kurzor alakja a normális nyílra változik. Ha akaratunk ellenére több vagy kevesebb pontot hoztunk létre klikkeljünk duplán a parkolási területre, ami újra megnyitja a **Parking Area** dialógusablakot. Használjuk a *Points* gombot a pontok számának vagy a parkolási terület szerkesztéséhez. A modellben a pontok száma kettő, mivel a kézikocsik száma kettő.

A szerkesztés befejezése után a „*Rendezo allomas*” és a „*Cella 1*” állomások végső helyzetének a *8.2 ábrához* hasonlónak kell lennie.



8.2. ábra: Az útvonalak és a parkolási terület elhelyezése

Most már futtathatjuk a modellt és az animációt. Azt fogjuk tapasztalni, hogy a futás nagyon gyorsan megszakad és meg jelenik az **Arena** *Error*/*Warning* ablak. A kijelzett figyelmeztetés azt jelzi, hogy néhány állomáspár között a távolság nincs megadva, ami arra kényszeríti az **Arena**-t, hogy a hiányzó távolságot 0-nak tekintse, tanácsos a problémát javítani. (Az **Arena** meggondolatlanul azt feltételezi, hogy elfelejtettünk megadni egy értéket.) Az ablak bezárása után, a program folytatásakor az üzenet újra megjelenik. A problémát a leggyakrabban a „*Kilepes a rendszerbol*”, a „*Cella 3*” és a „*Cella 4*” helyeken szabaddá váló kézikocsi okozza, amikor a kézikocsit a „*Rendezo allomas*”-on tartózkodó munkadarab igényli. Az Arena megpróbálja a kézikocsit a „*Rendezo allomas*”-ra mozgatni, de nem talál a „*Rendezo allomas*”-ra vezető utat ezért hibát jelez. Ha a kézikocsi a „*Cella 1*” és a „*Cella 2*” állomásokon szabadul fel, és „*Rendezo allomas*”-on jelentkezik az igény, akkor a kézikocsi üresen az óramutatóval ellentétes irányban fog mozogni, pedig az óramutató járásával megegyező irány lenne kívánatos. Ezt a problémát úgy kerülhetjük el, hogy minden lehetséges üres és rakott utat megadunk a **Distance** adatmodulban.

Általában, ha a modellben n állomás van, akkor a lehetséges távolságok feszítőfát alkotnak és az élek száma $n(n-1)$. Ez egy $n \times n$ méretű mátrix, amelynek a főátlójában 0 elemek vannak. E feltételezés szerint minden állomáspár között lehetséges a mozgás, és távolság függ az utazás irányától. A modellünk hat állomást tartalmaz és 30 lehetséges távolságot (a modellben a távolság nem függ az utazás irányától). Ha a távolság független az utazás irányától, akkor a távolságok száma $n(n-1)/2$. Gyakran előfordul, hogy bizonyos relációkban soha nem történik szállítás. A modellben „*Kilepes a rendszerbol*”, állomásról „*Rendezo allomas*” állomásra üres kézikocsi mozog, de ellentétes irányban nincs forgalom. A *8.6. táblázat* csak a rakott kézikocsi mozgásokat tartalmazza. Ha számba vesszük az összes lehetséges üres kézikocsi mozgá-

sokat és eliminálják a duplikációkat az első halmazból, akkor a *8.7. táblázatban* megadott távolságokat kapjuk, összesen 25 távolságot.

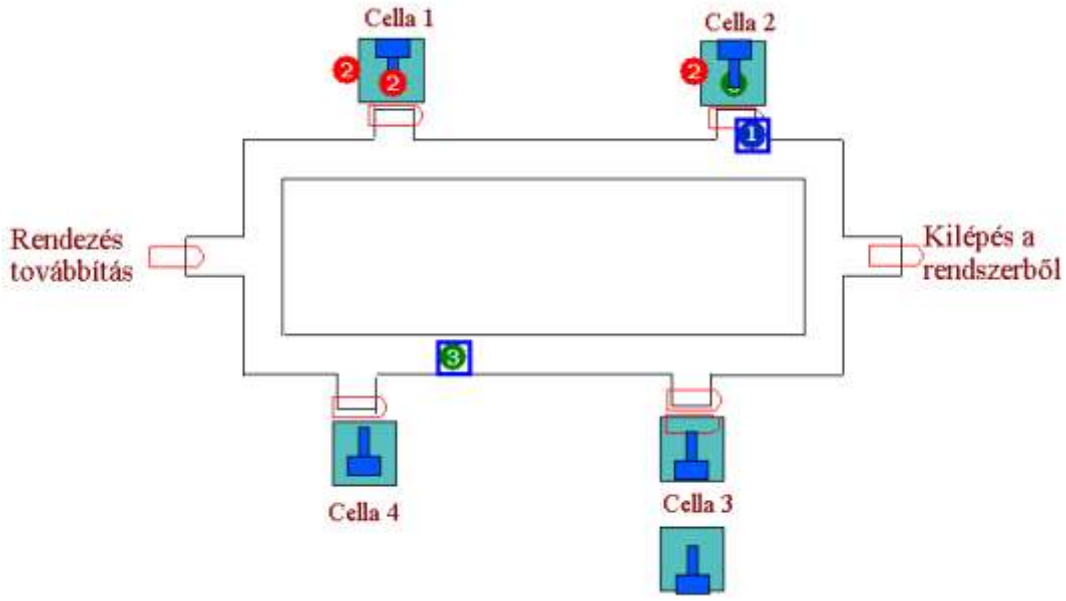
8.7. táblázat

Az összes lehetséges munkadarab-mozgás távolságai

Honnan/Hová	Rendező állomás	Cella 1	Cella 2	Cella 3	Cella 4	Kilépés a rendszerből
Rendező állomás		37	74			
Cella 1	155		45	92	129	
Cella 2	118	139		55	147	
Cella 3	71	92	129		45	155
Cella 4	34	55	92	139		118
Kilépés a rendszerből	100	121	158	37	74	

Ha a távolságok száma nagyon megnő, akkor a kötöttpályás transzporter alkalmazását javasoljuk, amely hálózatot használ az állomáspárok közötti távolság helyett. A kötöttpályás transzporterekről a 9. fejezet ad bővebb áttekintést. Részletes ismereteket Pegden, Shannon, and Sadowski: Advanced Manufacturing Features, (1995) című forrás tartalmaz.

Végül *8.2 modellben* az animáció nagyon hasonlít a *8.1. modellben* látottakhoz. Az észrevehető különbség, hogy a kézikocsik együtt mozognak a munkadarabokkal. A kötetlenpályának köszönhetően lehetséges, hogy egy éppen látható kézikocsi egy másik kézikocsi felett mozog. Bár csak két kézikocsi van a rendszerben, ez megtörténhet kisebb nagyobb gyakorisággal, többször mint a korábbi modellben, amely korlátozás nélküli utakat tartalmazott. A *8.3. ábra* egy pillanatfelvételt mutat az animációról.



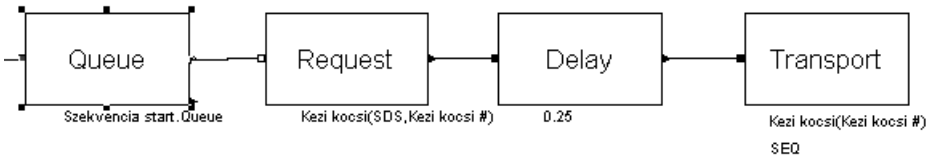
8.3. ábra: Kisméretű gyártórendszer transzporterekkel

8.3.2 A transzporter animáció finomítása (8.3. modell)

Ha futtatjuk a saját szerkesztésű vagy a gyakorló 8.2 modellünket, és alaposan megfigyeljük az animációt, akkor észrevehetjük, hogy a munkadarabok rendszeresen eltűnnek, miközben a kézikocsikra várakoznak. A jelenség csak átmeneti, mivel a munkadarabok később, a rakodási hely elhagyása előtt újra megjelennek a kocsikon. A jelenség kapcsán feltehetjük magunknak

a kérdést, hogy egyáltalán helyesen működik-e a modellünk. Átgondolva az észlelt jelenséget, arra a következtetésre jutunk, hogy a modellünk pontos, és a jelenség oka csak az animáció folytonossági hiánya. Amikor egy új munkadarab érkezik, vagy a munkadarab megmunkálása befejeződik egy cellában, akkor a munkadarab belép egy *request* (igény) sorba, ahol az igényelt kocsira várakozik. Miután egy munkadarab (entitás) allokál (leköt) egy kocsit a kocsi elindul a munkadarab tartózkodási helyéhez, ezzel egyidejűleg az entitás azonnal kilép a *request* sorból és eltűnik a képernyőről. (Az entitás pontos helyét illetően nem szükséges belemenni a részletekbe.) A jelenség a legegyszerűbb módon a **Storage** (tárolás) tulajdonság alkalmazásával szüntethető meg. A **Storage** modul egy olyan hely a szállítóeszközre várakozó entitások számára, ahol az entitások az üres transzporter mozgási ideje alatt tartózkodhatnak, és ahol az entitások folyamatosan látszanak az animációban. Minden esetben, amikor egy entitás belép a tárolóba, akkor a *storage* változó értéke 1-gyel növekszik, és amikor egy entitás elhagyja azt, akkor 1-gyel csökken. Ez lehetővé teszi, hogy statisztikákat készíthessünk a tárolóban tartózkodó munkadarabok számáról.

Sajnos a *storage* (tárolás) az **Advanced Transfer** panel moduljai számára nem érhető el, azonban a *Blocks* panel moduljai képesek használni ezt az opciót (a **Blocks** vagy az **Elements** panelről választott modulok esetében viszont az adattábla nézet nem érhető el.) A következőkben röviden bemutatjuk, hogy milyen változtatásokat kell elvégeznünk a modellünkön. Nyissuk meg a 8.2. modellt (Modell8-2) és mentsük el Modell8-3 néven. Először töröljük mind az öt **Leave** modult és helyettesítsük a **Blocks** panelről a **Queue–Request–Delay–Transport** modulok kombinációjával (*8.4. ábra*). (A *8.4. ábrában* látható modulok helyettesítik a *„Gyartosor indul”* nevű **Leave** modult.) Az említett négy modul a **Blocks** panelen található.



8.4. ábra: Queue–Request–Delay–Transport modulok a **Blocks** panelről

Storage - Advanced Process

	Name
1	Rendezes tovabbitas felszedes
2	Cella 1 felszedes
3	Cella 2 felszedes
4	Cella 3 felszedes
5	Cella 4 felszedes
6	Rendezes tovabbitas varakozas
7	Cella 1 varakozas
8	Cella 2 varakozas
9	Cella 3 varakozas
10	Cella 4 varakozas

Double-click here to add a new row.

8.9. képernyő: A **Storage** adatmodul

Ezt követően nyissuk meg a **Queue** adatmodult (**Basic Process** panel) és adjunk öt sort (*Queue*) a modellhez, ahol a munkadarabok várakoznak a kézikocsira: *Gyartosor in-*

dul.Queue, Ut a cella 1_bol.Queue, Ut a cella 2_bol.Queue, Ut a cella 3_bol.Queue, és *Ut a cella 4_bol.Queue*. Ennél a pontnál felmerülhet, hogy ezeket a sorokat már korábban definiáltuk. Ez valóban igaz, azonban, amikor a **Leave** modulokat töröltük, akkor ezeket a **Leave** modulokhoz tartozó sorokat is töröltük, ezért kell újra definiálni azokat. (Vegyük észre, hogy a sorok nemcsak a **Queue** adatmodulból, hanem az animációból is eltüntek) Miután a sorokkal kapcsolatos teendőinket befejeztük, az **Advanced Process** panel **Storage** adatmoduljában tíz tároló helyet adunk a modellhez (*8.9. képernyő*).

A hiányzó sorokhoz hasonlóan, a törölt **Leave** modulokban definiált „*Kezikocsi#*" attribútumot is pótolni kell. Ezt az attribútumot a „*Munkadarab tipusok es szekvenciak hozzarendeles*" nevű **Assign** modulban adjuk meg. A hozzárendelés a *8.8. táblázabant* látható. Amikor az **Assign** modulban csak az attribútum nevét definiáljuk, akkor a Value mezőnek nincs jelentése. Attribútumok definiálására használhatjuk az **Elements** panel **Attributes** modulját is.

8.8. táblázat

A „*Kezikocsi#*" attribútum definiálása

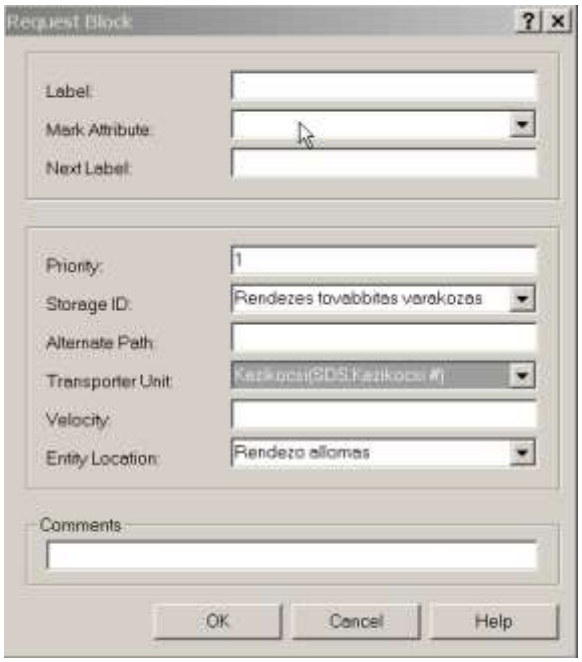
Type	Attribute
Name	Kezikocsi#
Value	0

Végül a *8.4. ábrán* látható **Queue–Request–Delay–Transport** modulkombináció elemeit szerkesztjük. Megmutatjuk, hogyan adjuk meg a négy cellába belépő munkadarabok számát. Kezdjünk **a Queue** adatmodullal, amelyben megadjuk a sor nevét (*8.9. táblázat*). Ebben a sorban a munkadarabok addig várakoznak, ameddig nem allokálnak egy kézikocsit. A sort akár el is hagyhatnánk, és ekkor az **Arena** egy belső sort használna a várakozó entitások tárolására. Ez a megoldás is pontos modellt eredményezné, azonban nem tudnánk animálni a sort vagy láthatóvá tenni a szállításra várakozó munkadarabokat.

8.9. táblázat

Gyartosor indul.Queue definiálása a **Queue** adatmodulban

Queue ID	Gyartosor indul.Queue



8.10. képernyő: A **Request Block** modul

A **Queue Block** modult a **Request Block** modul követi (*8.10. képernyő*). Ez a modul allokálja a tétlen kézikocsit és a munkadarab tartózkodási helyéhez mozgatja. A modul lehetővé teszi azt is, hogy tároló helyet specifikáljunk, amelyet arra használunk, hogy a munkadarabot az alatt az idő alatt is animálhassuk, ameddig a kézikocsi a munkadarabhoz érkezik.

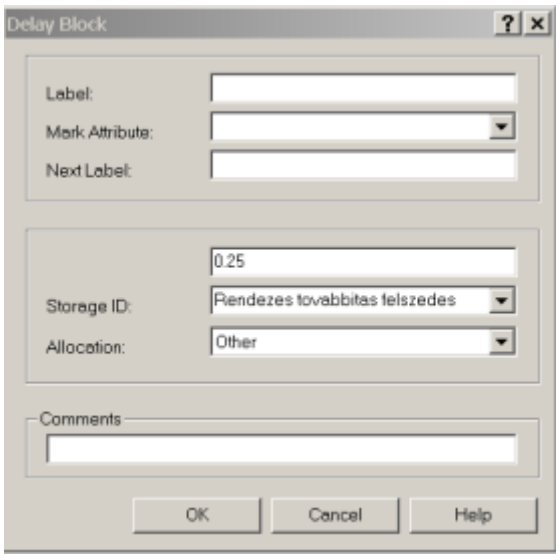
8.10. táblázat

Adatok definiálása a **Request Block** modulban

Storage ID	Rendezes tovabbitas varakozas
Transporter Unit	Kezikocsi(SDS,Kezikocsi #)
Entity Location	Rendezo allomas

A Storage ID és az Entity Location mezők adatait a *8.10. táblázatban* adtuk meg. A Transporter Unit mező további magyarázatot igényel. Először vegyük észre, hogy a mező árnyékolt. Ez azt jelzi, hogy a modulban ez egy kötelezően kitöltendő mező. A transzporter specifikálására három különböző formát használhatunk. Az első forma a *Transporter ID*(*Number*), ahol a *Transporter ID* a transzporter neve, esetünkben a „*Kezikocsi*". A *Number* az igényelt transzporter egységszámát jelöli. A példában két kézikocsit használunk, ezért az egységszám 1 vagy 2 lehet. Így ezzel a formával egyedileg azonosított transzportert igényelhetünk. Ha egységszámot elhagyjuk, akkor az **Arena** a *default* 1 egységszámú transzportert igényli. A második forma: *Transporter ID*(*TSR*), ahol a *TSR* (transporter selection rule) a transzporter kiválasztási szabály. A példánkban ez a legrövidebb távolság szabály, SDS (*Smallest Distance to Station*), amely az állomáshoz legkisebb távolságra elhelyezkedő transzportert választja. Az utolsó forma a *Transporter ID* (*TSR, AttributeID*), ahol az *AttributeID* az attribútum neve, amelyben az **Arena** a kiválasztott transzporter egységszámát tárolja. A példában az attribútum neve „*Kézikocsi#*". A Transporter Unit mezőhöz megadandó érték így *Kezikocsi*(*SDS,Kezikocsi #*).

Amikor a kézikocsi a feladó állomáshoz érkezik, a munkadarab kilép a **Request Block** modulból, és belép a **Delay Block** modulba (*8.11. képernyő*), amelynek a paramétereit a *8.11. táblázat* tartalmazza. A rakodási idő (*Duration*) 0,25 másodperc és a tároló hely azonosítója (Storage ID) „*Rendezes tovabbitas felszedes*". Később, amikor módosítjuk az animációt megmagyarázzuk, hogy miért definiáltuk a második tárolót ().



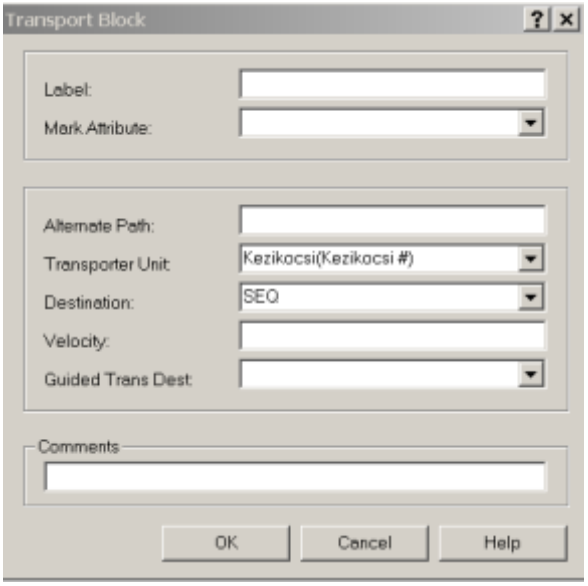
8.11. képernyő: A **Delay Block** modul

8.11. táblázat

Adatok definiálása a **Delay Block** modulban

Duration	0.25
Storage ID	Rendezes tovabbitas felszedes

Miután egy munkadarabot felraktunk a kézikocsira, azt a következő **Transport Block** modulba (*8.12. képernyő*) küldjük, amelynek a paraméterei a *8.12. táblázatban* láthatók. Az igényelt kézikocsi *Kezikocsi*(*Kezikocsi #*) és a célállomás (*Destination*) SEQ, ami a *Sequence* rövidítése az **Arenában**. A példában az allokált kézikocsit szándékosan követtük, így biztosak lehetünk abban, hogy a célállomásra érkező kézikocsi lesz felszabadítva.



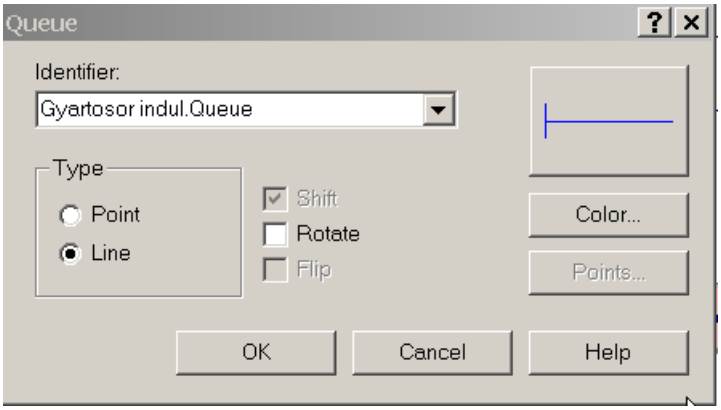
8.12. képernyő: A **Transport Block** modul

8.12. táblázat

Adatok definiálása a **Transport Block** modulban

Transporter Unit	Kezikocsi(Kezikocsi #)
Destination	SEQ

A további modulok paraméterei a leírtak mintájára adhatók meg, kivéve az *Entity Location* és a *Storage ID* értékeket, amelyek az adott helyen specifikusak.



8.13. képernyő: A **Queue** dialógus ablaka

Ezek után módosíthatjuk az animációnkat is. Először vegyük észre, hogy a korábban törölt **Leave** modulok tartozó a várakozó sorok (a *8.1. ábrán* még láthatók) az animációból is eltűn-

tek. A törölt **Leave** modulok részét képező sorokat a **Queue** adatmodulban már újra definiáltuk, most ezeket a megfelelő helyre az animációba is be kell illeszteni. A sorok beillesztéséhez használjuk a *Queue* gombot () az animációs eszköztárról. A megnyíló dialógus ablakban (*8.13. képernyő*), az Identifier mezőhöz tartozó listából először válasszuk a *Gyartosor indul.Queue* sort. Az *OK* gombbal zárjuk be az ablakot, és a megjelenő keresztet mozgassuk az animáción a megfelelő helyre, majd rajzoljuk meg a sort szimbolizáló egyenest. Az eljárást ismételve illesszük be többi újradefiniált sorokat (*Ut a cella 1_bol.Queue, Ut a cella 2_bol.Queue, Ut a cella 3_bol.Queue*, és *Ut a cella 4_bol.Queue*) is.

Ezt követően a *8.9. képernyőn* definiált tíz tároló helyet (*Rendezes tovabbitas felszedes, Rendezes tovabbitas varakozas, Cella 1 felszedes, Cella 1 varakozas, Cella 2 felszedes, Cella 2 varakozas, Cella 3 felszedes, Cella 3 varakozas, Cella 4 felszedes, Cella 4 varakozas*) is az animációhoz kell adni. Ehhez használjuk a *Storage* gombot () az *Animate Transfer* eszköztárról. A **Storage** modulban definiált kétpozíciós tároló helyeket az animációban úgy helyezzük el, hogy a felszedő hely és a várakozási hely közötti távolság érzékelhető legyen. Például az 1 jelű gyártócellánál a két tároló: *Cella 1 felszedes* és *Cella 1 varakozas*. Az egyik tárolót (*Cella 1 felszedes*) közvetlenül a parkolási terület (*Cella 1* nevű állomás) fölé, oda, ahol a kézikocsik a rakodás ideje alatt tartózkodnak, a másikat (*Cella 1 varakozas*) az állomástól távolabb, az *Ut a cella 1_bol.Queue* nevű sor alá helyezzük. Közvetlen a rakodás helyén csak egy tárolót (*Cella 1 felszedes*) használunk, annak ellenére, hogy egyszerre, ugyanazon a helyen akár két munkadarab rakodása is történhet. Ilyen esetekben a második munkadarab a hozzátartozó kocsival együtt az első fölé kerül. Ha kíváncsiak vagyunk az animáció pontos működésére, akkor nyissuk meg a *Modell8-3* nevű fájt, futtassuk a modellt és figyeljük meg, mi történik az 1 jelű gyártócellánál. A futás alatt a megmunkált munkadarab először a transzporterre várakozó sorban látható (*Ut a cella 1_bol.Queue*). Mihelyt egy kézikocsi elérhetővé válik és allokálódik a munkadarabhoz, a munkadarab az első tároló hely pozícióra kerül (*Cella 1 varakozas*), és addig marad ott, ameddig a kézikocsi megérkezik az állomásra (*Cella 1*). A kézikocsi érkezésekor a munkadarab helyzete megváltozik, a **Delay** modulban megadott tároló helyen (*Cella 1 felszedes*) jelenik meg, és a rakodás befejezéséig helyben marad. Ez a tároló a *Cella 1* nevű állomás fölött helyezkedik el, ahova a transzportert mozgattuk, ezért a munkadarab az álló transzporterren is látható. (Emlékezzünk arra, hogy a *Cella 1 varakozas* nevű tárolót a **Request**, a *Cella 1 felszedes* nevű tárolót pedig a **Delay** modulban definiáltuk.) Ezt követően a munkadarab együtt mozog a transzporterrel.

Létezik a bemutatott módszernél egyszerűbb út is annak az animációs problémának a megoldására, hogy a munkadarabok a *move* és a *delay* tevékenységek alatt nem láthatók az animációban. Ez a módszer a **Leave** modulok helyett a **Store - Request - Delay - Unstore – Transport** modulkombinációt használja. A **Request** és **Transport** modulok az **Advanced Transfer** panelen, amíg a **Delay, Store** és **Unstore** modulok az **Advanced Process** panelen találhatók. Az utóbbi két modul lehetővé teszi a tároló tartalmának növelését és csökkentését. Ha ezt a módszert használjuk, akkor a várakozó sort nem tudjuk animálni. Minden munkadarab a tárolóba (**Store**) kerül arra az időre, amíg a kézikocsi allokálására és az állomásra érkezésére, valamint a rakodás befejezésére várakozik. A módszer részleteit nem mutatjuk be, azonban javasoljuk a kipróbálását.

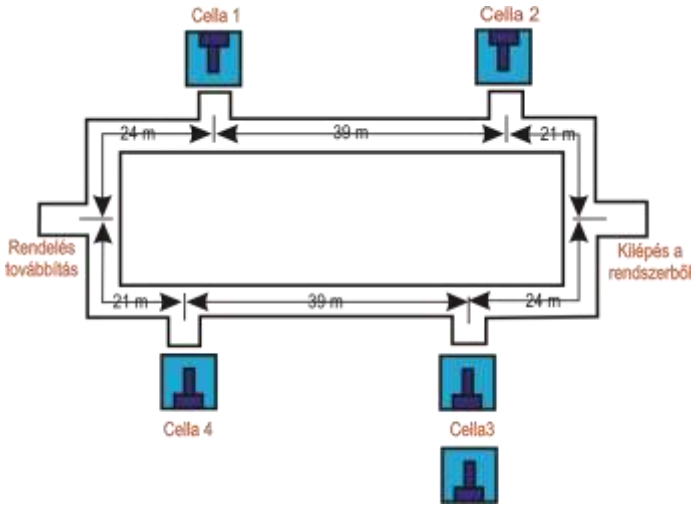
A modell futtatása előtt használjuk a *Run > Setup* parancsot és a Statistics Collection mezőben jelöljük meg a *Transporters* opciót is. Ha megfigyeljük az animációt, akkor néha láthatjuk a kézikocsira várakozó munkadarabokat, és úgy érezzük, hogy nincs túl nagy különbség az eredeti 8.1. modell futásához képest. A kézikocsik csak az idő 60%-ban használtak. Felhívjuk a figyelmet arra, hogy tévedés lenne azt állítani, hogy a két modell között nincs különbség. A relatíve rövid futási idő miatt nem érjük el a stacionárius működés állapotát (a tranzi-

ens indítás állapotában vagyunk), és ilyenkor a statisztikai változók is elhomályosítják az összehasonlítást.

Ha a kézikocsi sebességét 25 m/percre (az eredeti sebesség felére) változtatjuk, és futtatjuk a modellt, azt tapasztaljuk, hogy a két futás között nincs túl nagy különbség. Érdemes elgondolkodni azon, hogy mi ennek a magyarázata. Vegyük figyelembe, hogy a kézikocsik által megtett átlagos távolság kevesebb, mint 100 m, és amikor a kézikocsik gyakran válnak foglalttá, akkor az üresjáratok valószínűsége csökken. Emlékezzünk arra is, hogy amikor egy felszabaduló kézikocsira több igény is jelentkezik, akkor a kézikocsi a legközelebbi munkadarabot fogja szállítani. Az átlagos munkadarab mozgatási idő így csak 2 perc körüli érték. A felrakási és lerakási idő pedig változatlan, 0,25 perc. A rendszer érzékenységének elemzéséhez változtassuk a modell input paraméterit: a kézikocsi sebességét, a felrakási és lerakási időt, a transzporter kiválasztás szabályát, a kézikocsik számát, a szimuláció időtartamát, és figyeljük meg, hogy ezek miként hatnak a rendszer működésére.

8.4. Konvejorok

A gyártási rendszerünkben az anyagok mozgatására eddig egymással együttműködő kézikocsikat használtunk, amelyeket most egy konvejor rendszerrel szeretnénk helyettesíteni. A rendszert amennyire csak lehet, leegyszerűsítjük (természetesen a korrekt működés nem szenvedhet sérülést), és inkább a modellezési technikára koncentrálunk. Az új modellben a transporterek útvonalát követő, az óramutató járásával megegyező irányban mozgó, végtelenített, ún. hurok-konvejort alkalmazunk. A munkadarabok hat helyen léphetnek be a rendszerbe és hagyhatják el azt. A konvejor sebessége 20 m/perc. A szállítási távolságok m-ben a *8.5. ábrán* vannak megadva. Azt is feltételezzük, hogy 0,25 perc rakodási idő elegendő a fel és a lerakodásra, továbbá a miden munkadarab 4 m hosszú és a sérülések elkerülése végett 6 m helyet igényel a pályán.



8.5. ábra: A szállítási távolságok a konvejoron

Az **Arena** konvejorok működése azon az elven alapul, hogy a szállítandó entitásnak, mielőtt a konvejorra kerülne, addig kell várakoznia, amíg az adott helyen a belépéshez szükséges szabad hely elérhető nem lesz a konvejoron. Az **Arena konvejor** állandóan mozgó, **egyenlő hosszúságú cellákat** tartalmaz. Ezért amikor egy entitás megpróbál a konvejorra jutni, akkor addig kell várakoznia, amíg az entitás által igényelt, definiált számú, egymáskövető, szabad (nem foglalt) cella elérhető lesz az adott ponton. A konvejort úgy képzelhetjük el, mint egy mozgólépcsőt, amelynek a lépcsőfokai megfelelnek a celláknak, és a különböző utasok eltérő számú lépcsőfokot igényelnek. Például egy utas sok csomaggal kettő vagy három lépcsőfokot is igényelhet, ugyanakkor egy csomagnélküli személy csak egy lépcsőfokot. A mozgólépcső

tekintetében a cellaméret egyszerűen értelmezhető. Azonban ha egy szállítószalagot, vagy egy görgős pályát tekintünk, akkor a cellaméret egyáltalán nem magától értetődő.

Annak érdekében, hogy a konvejor tulajdonságait rugalmassá tegyük, az Arena lehetővé teszi a cellaméret definiálását. A konvejort hossza mentén feloszthatjuk egymástkövető, egyenlő méretű cellákra. Minden cella csak egy entitást hordozhat, de egy entitás több cellát is igényelhet. Ez meglehetősen érdekes dilemmát okoz, mert első közelítésben azt szeretnénk, hogy a cella olyan kicsi legyen, amennyire csak lehet, azért, hogy nagyobb modellezési pontosságot érjük el, ugyanakkor azt is szeretnénk, hogy a cella olyan nagy legyen, amennyire csak lehet, hogy nagyobb számítási sebességet érjünk el. A probléma illusztrálására tekintsünk egy egyszerű példát. Legyen egy 100 m hosszú konvejorunk, amelyen 2 m hosszú munkadarabokat akarunk szállítani. Mivel a konvejor hossza méterben adott, az első gondolatunk, legyen a cellamérete 1 m. Ez azt jelenti, hogy 100 cellánk lesz, és minden munkadarab 2 cella kapacitást köt le. Könnyen beállíthatjuk a cellaméretet 2 m -re (50 cella), vagy 1 cm-re (1000 cella) is. Napjainkban a számítógépek olyan gyorsak, hogy lényegtelennek tűnik mennyi cellára, 50-re vagy 1000-re osztjuk fel a konvejort. Ennek a hatása a modell számítási sebességére elhanyagolható. Azonban vannak olyan modellek, amelyekben a konvejor hossza több kilométer is lehet és ilyenkor a cellák számának érzékelhető hatása lehet a számítási sebességre. Ilyenkor az 1 cm és a 100 m közötti különbség lényegesen befolyásolhatja a modell futási idejét.

Kérdés, akkor miért nem használunk olyan nagy cellát, amekkorát csak lehet? Nos, amikor egy entitás megpróbálja elérni a konvejort, a belépés csak akkor lehetséges, amikor az előző cella vége, vagy a következő cella eleje éppen az entitás tartózkodási (belépési) helyéhez érkezik. A mozgólépcső analógiában, ez megfelel annak, hogy a belépés helyén arra várunk, hogy a következő lépcsőfok megjelenjen. Ha a cellaméretét 100 m hosszúra választjuk, és az utolsó cella vége 0,5 cm-rel éppen áthaladt a belépési helyen, akkor az entitásnak addig kell várnia, amíg a konvejor 99,5 m-t halad előre. Ha cellamérete 1 cm, akkor hasonló szituációban az entitásnak csak 0,5 cm-t kell várnia.

Az előzőek alapüzenete, hogy a cellaméret megválasztásakor egyaránt figyelembe kell venni a cellaméret hatását és a modellezett folyamat jellemzőit. Ha a konvejor terhelése nem túl nagy, vagy az időbeni késedelem hatása potenciálisan csekély a rendszer működésére, akkor használjunk nagyobb méretű cellát. Ha ezek közül valamelyik kritikusan hat a rendszer működésére, akkor használjunk kicsi cellaméretet. Amikor döntést hozunk, akkor két dolgot tartsunk szem előtt. Az első, az entitás mérete cellaszámban kifejezve csak egészértékű lehet, pl. egy entitás nem igényelhet 1,5 cellát. A másik, egy konvejor szegmensben a cellák számának (konvejor hossza osztva a cellamérettel) ugyancsak egészszámúnak kell lennie, például ha a konvejor 100 m hosszú, akkor a cellaméret nem lehet három.

Mielőtt konvejorokat adunk a modellünkhöz, tekintsünk át néhány alapfogalmat, amelyek remélhetőleg hasznosak lesznek, amikor saját modellt építünk konvejorokkal. Ugyanúgy, mint az erőforrásokhoz és a tanszporterekhez, a konvejorokhoz is tartozik néhány kulcsszó: *Access* (elérés), *Convey* (szállítás) és *Exit* (kilépés). Az Arena konvejoron egy entitás szállításához először a konvejoron helyet kell elérni, majd az entitást a célállomásra kell szállítani és végül az entitást ki kell léptetni a konvejorról, és ezzel egyidejűleg szabaddá tenni az elfoglalt teret. A fizikailag az Arena konvejor a teljes konvejort alkotó, egymáshoz kapcsolódó cellák sorozata. Minden szegmens egy Arena állomásnál kezdődik és fejeződik be. A teendők csak annyi, hogy a szegmenseket kapcsoljuk össze egy vonal- vagy egy hurok-konvejorrá. Egy egyedülálló konvejornak nem lehet elágazási pontja, ami miatt a konvejor folyásirányú árama nem osztható szét kettő vagy több áramra, és fordítva nem lehet gyűjtőpontja, ahol kettő vagy több ellentétes irányú áram összefolyik. Azonban az áram szétosztást vagy gyűjtést igénylő rendszerekben definiálhatunk több konvejort. Például, egy elágazó rendszerben az entitást

konvejorral szállítjuk az elágazási pontig, ahol kilépünk a konvejorról (*Exit*) és a következő konvejoron teret igénylünk (*Access*) az entitás számára, amely a célállomásra szállítja azt. A kisméretű rugalmas gyártási rendszerben mi egyedülálló hurok-konvejort fogunk használni.

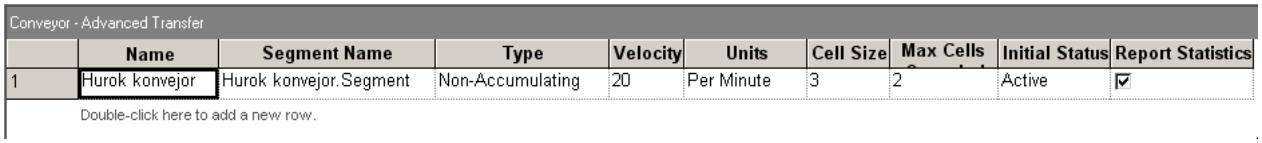
Az Arena-ban két konvejor típust különböztetünk meg: non-accumulating (nem-akkumuláló) és accumulating (akkumuláló) konvejorokat. Minkét típus csak egy irányban mozoghat, azaz nem lehetnek reverzálóak. A konvejortípus nevek utalnak a funkciójukra. Például a serleges elevátor, a szállítószalag vagy a mozgólépcső **nem-akkumuláló** konvejorok. Ezeken a konvejorokon a az entitások közötti távolság nem változik szállítás közben, hacsak az entitás ki nem lép, és újra teret igényel. A nem-akkumuláló a konvejorok e szabály követve azonos módon működnek. Abban a pillanatban, amikor a nem-akkumuláló konvejoron egy entitás éppen teret nyer, lényegében a konvejor áll, vagyis kikapcsolt állapotban van. Az entitás szállításának megkezdésekor a konvejor bekapcsolt állapotban kerül, és addig mozog, amíg a következő célállomást el nem éri. A célállomás elérésekor a konvejor megáll, és az entitás kilép, majd a konvejor újraindul. Ha a konvejor elérése és a szállítás indítása között, vagy a célállomás elérése és a kilépés között nem telik el idő, akkor a konvejor úgy működik, mintha soha nem állna meg. Ha azonban az említett helyeken pozitív fel- és lerakási idő jelentkezik (mint pl. a mi rendszerünk esetében), akkor a konvejor átmenetileg a fel- és lerakodás időtartamáig megáll.

Az **akkumuláló** konvejorok abban különböznek, az előzőtől, hogy soha nem állnak meg. Amikor egy entitás megáll az akkumuláló konvejoron, akkor más entitások folytatják az útjukat a konvejoron. Törvényszerűen ezért az álló entitás előbb vagy utóbb blokkolja a mögötte haladó entitásokat, azaz a továbbmozgó entitások az álló entitás mögött torlódnak. Amikor az álló entitás kilép, a mozogó konvejoron az akkumulált entitások felszabadulnak és tovább mozognak a célállomásuk felé. Azonban az entitások mindegyike nem tud egyidejűleg újraindulni. A modellben definiált helyszükséglettől függően időeltolódással indulnak. Gondoljunk a gépkocsikra, amelyek az autópályán vagy egy közlekedési lámpa előtt akkumulálódnak (torlódnak). Amikor a gépkocsik megállnak, akkor biztonságos távolságot tartva, de a normál követési távolságnál szorosabban egymás mögé sorakoznak (megakadályozva, hogy néhány szemtelen vezető besoroljon közéjük). A blokkolás megszűnésekor a gépkocsik időeltolódással indulnak el, megvárják, hogy a kocsik között biztonságos követési távolság alakuljon ki. A fejezetben később leírjuk ennek a jelenségnek az adatigényét.

8.4.1 Kisméretű rugalmas gyártórendszer nem-akkumuláló konvejorokkal (8.4. modell)

A bevezető után készen állunk arra, hogy alkalmazzuk az ún. nem-akkumuláló konvejorokat a gyártási rendszerünkben. A modell építésekor a szükséges változásokra felhívjuk a figyelmet. Miután elhelyeztük a konvejorokat és futtatjuk a modellünket úgy találjuk, hogy nehéz szemmel követni, hogy a vajon konvejorunk helyesen működik-e, mivel a felrakási és a lerakodási idő nagyon kicsi a többi időhöz viszonyítva. (A modell futásakor csak sejtjük, hogy mi történik.) Szeretnénk néhány kísérletet elvégezni, azért hogy megtudjuk, a rakodási műveletek miként hatnak a rendszer működésére. Azt javasoljuk ezért, hogy előrelátóan definiáljunk két új változót: „*Felrakasi ido*”, „*Lerakasi ido*”, és állítsuk ezek kezdeti értékét 0,25 percre. Ezt követően a **Leave** és az **Enter** modulokban a Delay mezőben helyettesítsük a konstans időket a megfelelő változó nevekkel, ami lehetővé teszi, hogy később egy helyen változtassuk ezeket az időket. Ennek a módja már ismert, ezért nem ismételjük a részleteket (ha nem akkor olvassa újra az 5.2.3 alfejezetet).

Nyissuk meg a 8.1. modellt, és ismét töröljük az útvonalakat. A **Conveyor** adatmodult (**Advanced Transfer** panelről) használva, definiáljuk a konvejort, amit a *8.14. képernyő* és a *8.13. táblázat* mutat.

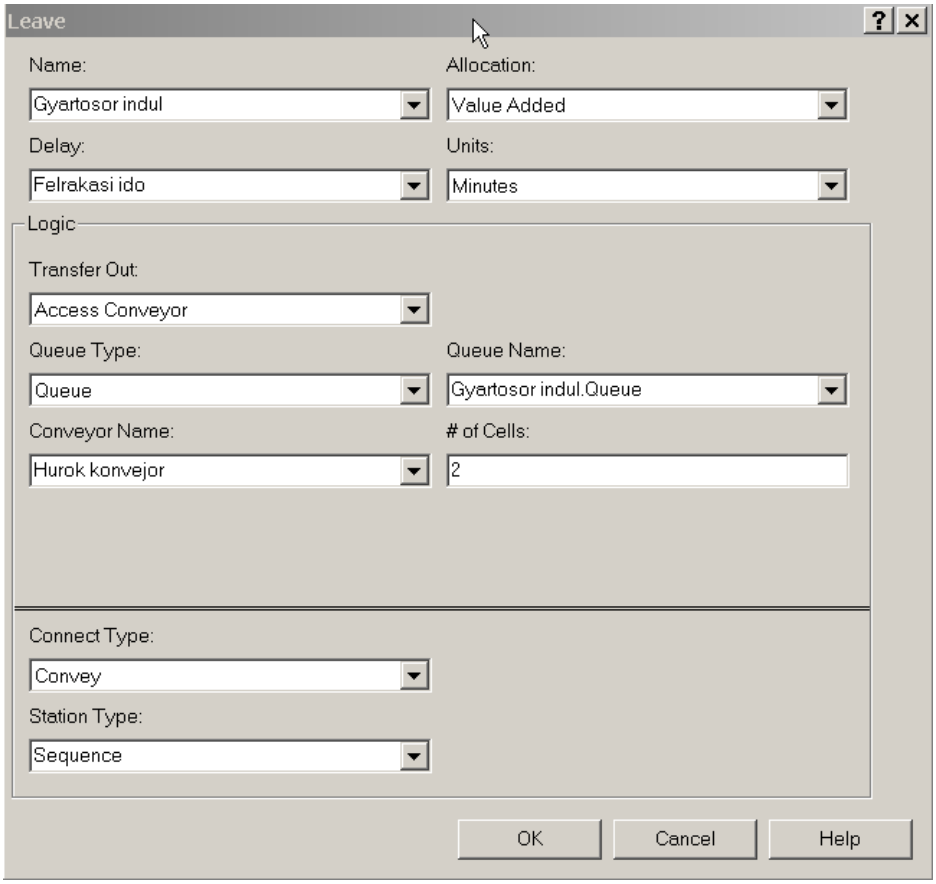


8.14. képernyő: A **Conveyor** adatmodul dialógusablaka

8.13. táblázat

Adatok definiálása a **Conveyor** adatmodulban

Name	Hurok konvejor
Segment Name	Hurok konvejor.Segment
Type	Nonaccumulating
Velocity	20
Cell Size	3
Max Cells Occupied	2



8.15. képernyő: A „*Gyartosor indul*” nevű **Leave** modul dialógusablaka

A definiált adatok közül kettő, a *Cell Size* (cellaméret) és a *Max Cells Occupied* (maximálisan elfoglalt cellák száma) további magyarázatot igényel. Az előző modell eredményei alapján teljesen világos, hogy a rendszerben a forgalom nem túl nagy. A bevezetőből ismert az is, hogy ha a munkadarab várakozási ideje kicsi mielőtt a konvejorra kerül (az igényelt tér elérhető lesz), akkor a várakozás hatása a munkadarab ciklusidejére is csekély. Ezért úgy döntünk, hogy olyan nagy cellát használunk, amekkorát csak lehet. Legyen a cellamérete 3 m, mivel ez egészszámú cellát ad a teljes konvejor hosszra. A konvejor teljes hossza 168 m (*8.5. ábra*), és a konvejor hossza egészszámú többszöröse a 3-nak, azaz 168/3=56 cella. Minden munkadarab 6 m helyet igényel, ezért egy munkadarab két cellát foglal el (*Max Cells Occupied*=2). Erre az információra azért van szüksége az Arena-nak, hogy elegendő helyet

tudjon biztosítani a belépő entitás számára, amikor az megérkezik a konvejor végéhez. Mivel a mi konvejorunk egy hurok-konvejor, amelynek nincs vége, ennek nincs hatása a modellre. De más esetekben, általában meg kell adni azt a legnagyobb cellaszámot (*Max Cells Occupied*), amit az entitás elfoglalhat a szállítás alatt.

A konvejorok beépítése a modellbe nagyon hasonló a transzportrek alkalmazásához, ezért nem részletezzük. A **Leave** és az **Enter** modulokat kell megváltoztatni, ugyanúgy, ahogy a tarnszportereknél tettük. A „*Gyartosor indul*” nevű **Leave** modul paraméterei a *8.15. képernyőn* és a *8.14. táblázatban* láthatók. Figyeljünk arra, hogy minden munkadarab a konvejorra kerüléskor két üres cellát (minden cellamérete 3 m) igényel, és a Delay mezőben a „*Felrakasi ido*” nevű változót kell megadni, ahogy azt korábban megbeszéltük.

8.14. táblázat

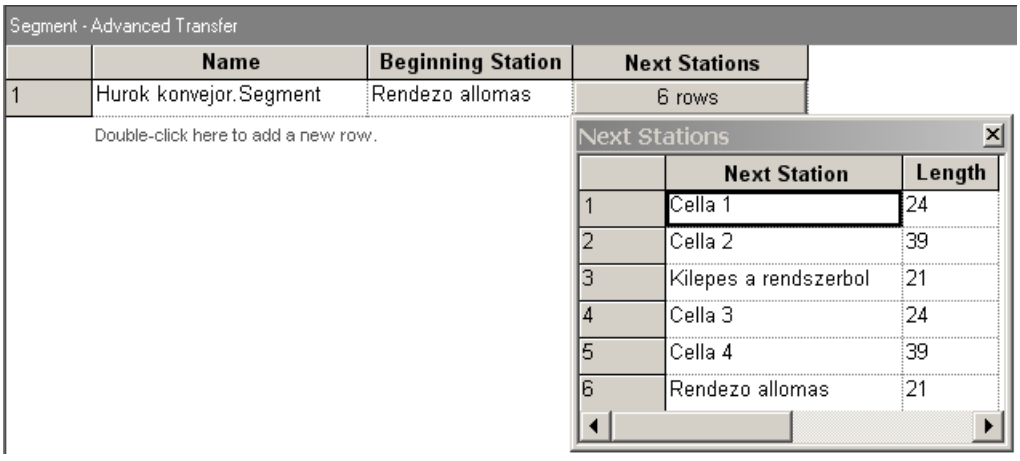
Adatok definiálása a „*Gyartosor indul*” nevű **Leave** modulban

Name Delay Units	Gyartosor indul Felrakasi ido Minutes
Logic Transfer Out Conveyor Name # of Cells	 Access Conveyor Hurok konvejor 2
Connect Type	Convey
Station Type	Sequence

8.15. táblázat

Adatok definiálása a „*Belepes a Cella 1 allomasra*” nevű **Enter** modulban

Name Station Type Station Name	Belepes a Cella 1 allomasra Station Cella 1
Logic Delay Units Transfer In Conveyor Name	 Lerakasi ido Minutes Exit Conveyor Hurok konvejor



8.16. képernyő: A **Segment** adatmodul dialógusablaka

A „*Belepes a Cella 1 allomasra*” nevű **Enter** modul paramétereit a *8.15. táblázat* foglalja össze. A paraméterek megadása után készen állunk arra, hogy elhelyezzük a konvejor szeg-

menseket az animációban. Az előző modellben alkalmazott **Distance** (távolság)-gal analóg módon, először a modell és az animáció működéséhez egyaránt szükséges szegmens adatokat definiáljuk. Ehhez a **Segment** adatmodult használjuk, amely ugyancsak az **Advanced Transfer** panelen található. A *8.16. képernyő* a **Segment** adatmodult mutatja táblázat nézetben az első adatsorral és a Next Station lenyíló ablakba beírt adatokkal. A hurok konvejor első pontjának (Beginning Station) önkényesen a "*Rendezo allomas*"-t választottuk.

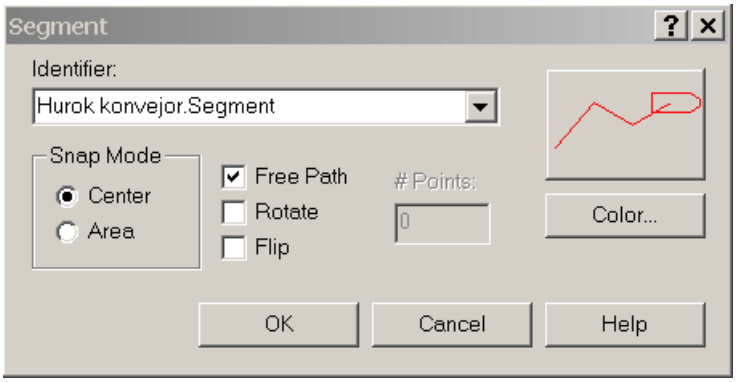
A *8.16. képernyő* jobb alsó részén a hat szegmens adatait láthatjuk. Megjegyezzük, a Length oszlopban a számok a megelőző és az aktuális állomás közötti távolságot jelentik, és nem a cellák számát. Például a „*Cella 1*" és a „*Cella 2*" állomások közötti távolság 39 m. Ezek a távolságok közvetlenül leolvashatók a *8.5. ábráról*.

8.16. táblázat

Adatok definiálása a **Segment** adatmodulban

Name	Hurok konvejor.Segment
Beginning Station	Rendezo allomas

Miután befejeztük a szegmensek definiálását, beszerkeszthetjük azokat az animációba. Először az állomásokat szimbolizáló alakzatokat ( ) (ezeket a 8.1. modellből örököltük) mozgassuk a hurok nyomvonalára, arra a helyre, ahol a munkadarabok belépnek a konvejorra, illetve elhagyják azt. A szegmensek beillesztéséhez használjuk a szegmens gombot ( ) az **Animate Transfer** eszköztárról. Általában annyi szegmenset kell beilleszteni, amennyi a szakaszok (résztávolságok) száma, mert általában a szegmensek, mint élek, feszítőfát alkotnak. A mi modellünk kivétel (a hurok-konvejor szegmensei zárt kört alkotnak), amelyben a kör záródásához hat szegmenset kell elhelyezni. A Segment dialógusablak a *8.17. képernyőn* látható, amelynek paramétereit nem változtattuk, elfogadtuk az alapértelmezett értékeket.



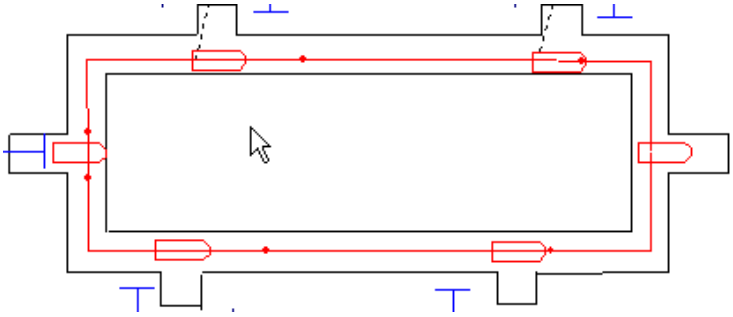
8.17. képernyő: A **Segment** dialógusablak

Egy szegmens beillesztésének menete a következő: kattintsunk az **Animate Transfer** eszköztáron a szegmens gombra ( ), a megjelenő dialógusablakot (*8.17. képernyő*) az OK gombbal zárjuk be, a bezárás után megjelenő keresztet mozgassuk a szegmens kezdőállomása fölé és kattintással pozícionáljuk a kezdőpontot az állomásra, az egér mozgatásával és kattintással rajzoljuk meg a szegmens nyomvonalát, végül a szegmens végpontját dupla kattintással helyezzük a szegmens végét szimbolizáló következő állomásra. Ezt az eljárást ismételjük meg hatszor.

A modellbe beillesztett hat szegmenset a *8.6. ábrán* mutatja. A beillesztés során, ha szükséges, pozícionáljuk újra az állomásokat szimbolizáló alakzatokat úgy, hogy a szegmensek kövessék a hurok nyomvonalát és a *8.6. ábrához* hasonló képet kapjunk.

Az animáció befejezése után készen állunk a modell futtatására. Először a *Run* > *Setup* paranccsal és a *Project Parameters* füllel megnyitott ablakban kapcsoljuk be a konvejor statisz-

tika (*Statistics Collection* > *Conveyors*) gyűjtését. A modell futtatása után észrevehetjük, hogy a riportban a *Conveyor* szekcióban két új, a konvejorra vonatkozó statisztika jelenik meg. Az első statisztika (Blocked) azt mutatja, hogy a konvejor a teljes idő 17,65 százalékában állt. Első látásra ez a szám túl nagy értéknek látszik, mégis reális, ha figyelembe vesszük, hogy minden alkalommal, amikor egy munkadarab a konvejorra kerül vagy arról kilép, jelentkezik a fel és lerakodási idő. Ez idő alatt a konvejor áll vagy blokkolt (blocked). A mozgatások és a munkadarabok számát tekintve ez az érték valószínűnek látszik.



8.6. ábra: A hurok-konvejor szegmensei az állomásokkal

A második statisztika (*Utilization*) arról tájékozat, hogy a konvejor időben átlagosan csak az 6,6 százalékosan kihasznált. Ez nem az, az idő, amit a munkadarabok a konvejoron töltöttek, hanem az, az átlagosan elfoglalt tér, amit a munkadarabok lefoglaltak a konvejoron. Ezt könnyen ellenőrizhetjük. Használjuk a *8.5. ábrát*, és számítsuk ki a konvejor teljes hosszát, ami 168 m-re adódik. Azt is tudjuk, hogy minden cella mérete 3 m és a cellák száma 56. Minden munkadarab szállítása két cellát igényel, ami azt jelenti, hogy a konvejor egyidejűleg 28 munkadarabot szállíthat. Ha a 28 munkadarabot szorozzuk az átlagos kihasználtsággal (0,066), akkor azt kapjuk, hogy a konvejoron átlagosan egyidejűleg 1,848 munkadarabot szállítottunk. Ha megfigyeljük az animációt, akkor meggyőződhetünk az eredmény korrektségéről.

Ha összehasonlítjuk a munkadarabok rendszerben töltött idejét a 8.1. modell azonos adataival, akkor úgy találjuk, hogy azok nagyobb értékek. Ez indokolt a konvejor blokkolás és a sebesség miatt.

Ha az animációban nem látszik a fel- és lerakási idő alatt a konvejor leállása (még akkor sem, ha lecsökkentjük a szimuláció sebességét, változtatjuk az Animation Speed Factor értékét), akkor növeljük a „*Felrakasi ido*” és a „*Lerakasi ido*” változók értékét 2 vagy 3 percre. A magasabb rakodási időkkel futtatva a modellt, a leállás láthatóvá és nyilvánvalóvá válik.

A szimuláció alatt a változók értékének megváltoztatására több alternatív módszer is létezik. Ha gyakran szeretnénk változtatni, akkor alkalmazzuk az Arena Visual Basic® for Applications (VBA) interface-t. A második egyszerűbb módszer, használjuk az Arena parancs driver-ét a **Run Controller**-t. A modell indítása után várjunk körülbelül 445 percet és használjuk a *Run>Pause* parancsot, vagy a *Pause* gombot ( ) a **Standard** eszköztárról, hogy megszakítsuk átmenetileg a program végrehajtását. Ezt követően a *Run>Run Control>Command* paranccsal vagy *Command* gombbal ( ) a **Run Interaction** eszköztáron nyissuk meg a *Command* ablakot. Ez a szöveges ablak mutatja a pillanatnyi szimulációs időt, amit a ">" prompt követ, amely után beírhatjuk a saját parancsunkat. Használjuk a SHOW parancsot, hogy megjelenítsük a „*Felrakasi ido*” változó értékét, majd az ASSIGN parancsot, hogy megváltoztassuk az értékét (*8.7. ábra*). Ezt követően ismételjük meg ezt a két lépést a „*Lerakasi ido*” értékének a változtatásához. Zárjuk be az ablakot és a *Run* > *Go* menü opcióval, vagy a *Go* gombbal ( ) a **Standard** eszköztáron folytassuk a szimulációt a változók új értékeivel.

```
SIMAN Run Controller.
449.03333>SHOW Felrakasi ido
FELRAKASI IDO = 0.25
449.03333>ASSIGN Felrakasi ido = 2
449.03333>SHOW Lerakasi ido
LERAKASI IDO = 0.25
449.03333>ASSIGN Lerakasi ido = 2
449.03333>
```

8.7. ábra: Változók értékének a változtatás a **Run Controller**-rel

Ha nagyon rövid ideig figyeljük az animációt, akkor azt tapasztaljuk, hogy a konvejor mozgása szaggatott, és gyorsan feltöltődik tíz munkadarabbal. Bármikor megállíthatjuk a szimuláció futását, és megváltoztathatjuk ezeket az értékeket. Nem ronthatunk el semmit, mert a **Run Controler**-rel végrehajtott változtatások nem permanensek, csak átmenetiek. Az Arena változásokat nem őrzi meg, ezért a következő alkalommal a modell az eredeti értékekkel fut.

Bár az animáció közbeni változtatások kiváló lehetőséget teremtenek az ellenőrzésre, a modell kezdeti feltételeit futás közben nem tudjuk véglegesen megváltoztatni. Ne felejtsük azonban, amikor a modellt kiértékelési céllal használjuk, ez nagyon félrevezető működési értékeket eredményezhet és a beavatkozásokat lényegében lehetetlen megismételni.

8.4.2 Kisméretű rugalmas gyártórendszer akkumuláló konvejorokkal (8.5. modell)

A modellben a konvejor típusának a megváltoztatása nagyon könnyű. A Modell8-4 modellt mentsük Modell8-5 néven, és ehhez használjuk a *File* > *Save As* parancsot. A modell átalakításához a **Conveyor** adatmodulban két adatot kell megváltoztatni. A Type mezőben válasszuk az *Accumulating* opciót és Accumulation Length mezőbe pedig írjunk 4 m (*8.18. képernyő*). A megadott akkumuláló hosszúság azt jelenti, hogy torlódáskor a munkadarab 4 m helyet igényel a konvejoron. Megjegyezzük, az álló entitásra megadott értéknek (4 m) nem kell egészszámú cellának megfelelni. Például esetünkben a 4 m 1,5 cellának felel meg. A blokkolás megszűnésekor a munkadarabok újra rendeződnek és ismét 6 m, azaz kétcellányi helyet foglalnak el a konvejoron.

Conveyor - Advanced Transfer

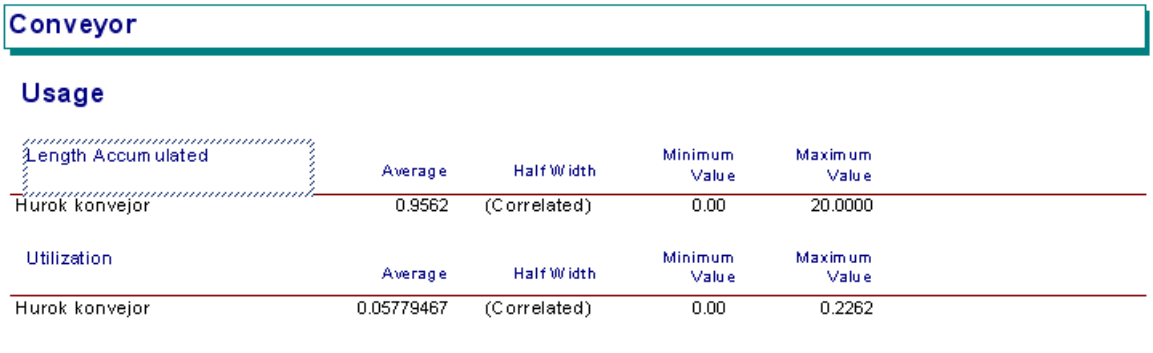
	Name	Segment Name	Type	Velocity	Units	Cell Size	Max Cells Occupied	Accumulation Length	Initial Status
1	Hurok konvejor	Hurok konvejor.Segment	Accumulating	20	Per Minute	3	2	4	Active
	Double-click here to add a new row.								

8.18. képernyő: A **Conveyor** adatmodul dialógusablak

8.17. táblázat

Adatok definiálása a **Conveyor** adatmodulban

Name	Hurok konvejor
Segment Name	Hurok konvejor.Segment
Type	Accumulating
Velocity	20
Units	Per Minute
Cell Size	3
Max Cell Occupied	3
Accumulation Length	4



Conveyor

Usage

Length Accumulated	Average	Half Width	Minimum Value	Maximum Value
Hurok konvejor	0.9562	(Correlated)	0.00	20.0000

Utilization	Average	Half Width	Minimum Value	Maximum Value
Hurok konvejor	0.05779467	(Correlated)	0.00	0.2262

8.7. ábra: A **Category Overview** riport **Conveyor** szekciójának eredményei

A változtatások végrehajtása után ha futtatjuk a modellt azt tapasztaljuk, hogy nagyon kicsi akkumuláció (torlódás) történik. Erről meggyőződhetünk, ha az összegző riportban tanulmányozzuk a konvejor statisztikát (*8.7. ábra*). Az átlagos akkumuláció kevesebb, mint 1 m és az átlagos kihasználtság pedig kisebb mint 6%. Az akkumuláció növeléséhez és jobb érzékeltetéséhez változtassuk a fel- és lerakási időt 4 vagy 5 percre. Ennek a hatása futás közben már jól érzékelhető.

A riportokat összehasonlítva, a nem-akkumuláló konvejorhoz viszonyítva, az eredmények nagyon hasonlóak, kivéve a munkadarab rendszeridőt, amely sokkal nagyobb az akkumuláló konvejor esetében. Valószínűleg ez a rövid futási időnek köszönhető. A gyanú igazolására, az érdeklődő olvasónak javasoljuk a szimulációs idő növelését és az ismételt összehasonlítást.

8.5. Összefoglalás

Az entitások mozgatása a szimulációs modell nagyon fontos része. Ebben a fejezetben illusztráltuk azokat az Arena az eszközöket, amelyek a mozgatások valósághű modellezése érdekében különböző körülmények között alkalmazhatók.

Ez a fejezet az utolsó általános, bevezető az Arena modellezésbe, amit néhány mélyebb és részletesebb, alacsonyabb szintű modellezési képesség bemutatása követ. Például a hibakeresés és az animáció finomítása. Kitérünk arra is, hogyan érhető el és keverhető a SIMAN szimulációs nyelv, bár a nyelv teljes bemutatására nem vállalkozunk. A részletek megismeréséhez ajánljuk Pegden, Shannon és Sadowski (1995) munkáját, amely az SIMAN alkalmazásával foglalkozik. Az eddig megszerzett ismeretek birtokában is, már félelmetes modellezési arzenállal rendelkezünk ahhoz, hogy különböző bonyolultságú rendszereket valósághűen modellezzünk. Azonban felhívjuk a figyelmet arra, hogy még számos további modellezési konstrukció és trükk létezik. Ezek közül néhányat a 9. fejezetben ismerhetünk meg.

9. További modellezési problémák és módszerek

Az előző fejezetekben egyre bonyolultabb példák bemutatásával átfogó képet adtunk a különböző rendszerek modellezéséről. Ezeket a példákat tudatosan, az alkalmazás realitását és fontosságát szem előtt tartva úgy választottuk ki, hogy különböző modellezési kérdéseket érintsenek, továbbá megmutassuk, milyen módszerek segítségével lehet az **Arena**-val korrekten, könnyen és gyorsan modellezni. Ezzel a tudással felvértezve, képesek leszünk arra, hogy sokszínű, de emellett valóságos és hatékony szimulációs modelleket hozzunk létre.

Ezekből a feladatokból semmilyen ésszerű válogatás nem tudja feltárni a modellezési kérdéseknek az összes rejtett zugát, trükkjeit, amiket a felhasználónak időnként használnia kell. Hasonlóképpen, képtelenség az Arena összes lehetőségét, képességét megmutatni. Ezért erre mi sem törekszünk. Ennek ellenére szeretnénk kiemelni és megmutatni néhány fontosnak tartott módszert (trükköt), és elmondani hogyan kell ezeket használni.

Ezt több példa bemutatásával érjük el, amelyekben leginkább a specifikus modellezési módszerekre és Arena tulajdonságokra fogunk koncentrálni, ezért a példák jóval lényegre törőbbek lesznek az eddigieknél. A 9.1. fejezetben tovább fejlesztjük a 8. fejezetben konstruált konvejor modelljeinket; a 9.2. fejezetben pedig megbeszélünk néhány modellfinomítási lehetőséget a 8. fejezetbeli transzporterekkel összefüggésben. A kiszolgáló-rendszerekben, különösen azokban, amelyekben emberek fontos szerepet játszanak, gyakori esemény a vásárló „visszalépés” (más szóval, amikor valaki kiugrik a sorból). Ezt a problémát a 9.3. fejezetben tárgyaljuk, az „akadályoztatás” fogalmával együtt. A 9.4. fejezet bevezetés azokba a módszerekbe, amelyek munkadarabok tárolását (az eddig megismert sorokon túl), valamint az entitások csoportosítását (batch képzés) teszik lehetővé, azzal a lehetőséggel, hogy később szétválasszuk azokat. A 9.5. fejezetben bemutatjuk, hogyan lehet egy szorosan párosított rendszert, amelyben az entitások a pillanatnyi pozíciójukból erőforrásokat allokálnak a mozgásuk irányában mielőtt továbbhaladhatnak, ezeket az entitások szempontjából *átfedett erőforrásoknak* nevezik. Végül a 9.6. fejezetben röviden megemlítünk néhány más specifikus témakört, a vezetett transzportereket, a párhuzamos sorokat, továbbá a komplex döntési logika és a hurkolás lehetőségét.

Ez a fejezet a korábbiaktól eltérően, amelyekben a különböző fejezetek (témakörök) sorrendje nem feltétlen határozta meg az olvasás sorrendjét, máshogy van felépítve. Inkább azt a célt szolgálja, hogy az alkalmazott projekt változatokon keresztül mintákat mutasson a fontosnak tartott modellezési technikák és Arena tulajdonságok köréből.

9.1. Konvejorok modellezése az Advanced Transfer Panel használatával

Ebben a bekezdésben a 8. fejezetben leírt konvejor alapmodellek finomítási lehetőségeit mutatjuk be.

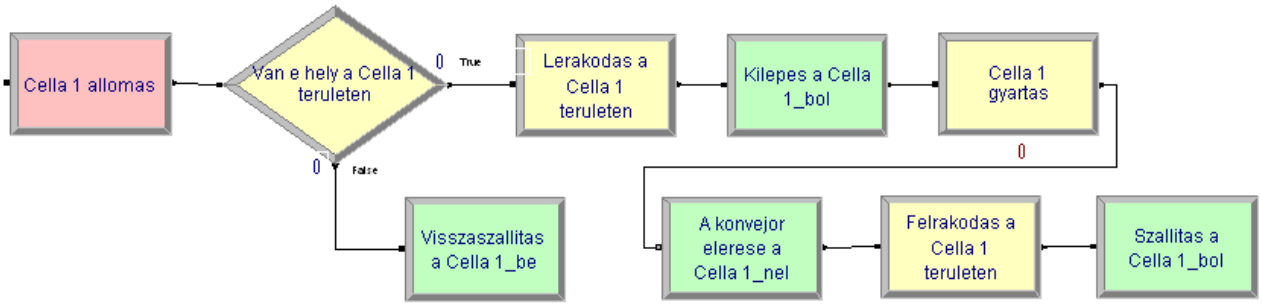
9.1.1. Véges tárolók az állomásokon (9.1. modell)

A 8. fejezetben bemutattuk az Arena konvejorokat. A 8.4.1. bekezdésben a kisméretű gyártórendszer egy olyan modelljét (8.4. modell) fejlesztettük ki, amelyben nem-akkumuláló konvejorokat használtunk a munkadarabok szállítására. A modell fejlesztésekor feltételezzük, hogy minden gyártócella előtt végtelen (korlátlan) kapacitású tároló található a megmunkálásra várakozó munkadarabok tárolására. Ez a feltételezés megengedte, hogy az **Enter** és a **Leave** modulok által meghatározott igények szerint használjuk a konvejor képességeit. A konvejor definiálásához az **Advanced Transfer Panel**-ről a modellhez adtuk a **Conveyor** és a **Segment** adatmodulokat.

Most módosítjuk a feltételezésünket annyiban, hogy korlátozzuk a feldolgozásra váró munkadarabok tárolására rendelkezésre álló tárolókapacitást a „*Cella 1”* és a *„Cella 2*” gyártócelláknál. Pontosabban, feltételezzük, hogy mindkét gyártócellánál csak egy-egy munkadarab várakozhat. Az ilyen típusú modellben definiálni kell, hogy mi történjen azzal a munkadarabbal, amely az 1. vagy 2. cellához ér, de ott már egy másik munkadarab foglalja el a véges tároló teljes kapacitását. Feltételezve, hogy az **Enter** modult használatával egyidejűleg megoldást találunk (nem találunk) a tároló korlátozására, csábító lenne egyszerűen várakoztatni az érkező darabot addig, amíg a tárolóban lévő munkadarab továbbhalad a gyártócellájához. Természetesen ez jelentős torlódást okozhatna a gyártócellák előtt. Nemcsak az jelent gondot, hogy a munkadarabok nem képesek belépni a cellába, hanem a más cellákhoz tartó munkadarabok is torlódnának a várakozó darabok mögött, újabb problémát okozva ezzel. Ezzel egyidejűleg a megmunkált munkadarabok is megpróbálják elhagyni a gyártócellát, és helyet igényelnek a konvejoron, hogy tovább haladjanak a következő célpontjuk felé. A hely, amelyet a megmunkált munkadarabok megpróbálnak elérni, azonban foglalt a feldolgozandó és a gyártócellába belépésre várakozó munkadarabok által. Ezt a megoldhatatlan helyzetet nevezik *deadlock*-nak vagy *gridlock*-nak (zsákutcának vagy forgalmi dugónak).

A probléma megoldására, az 1. és 2. cellához érkező munkadarabokra, használjuk a következő módszert. Ha a tároló éppen üres, az érkező darab beléphet oda, különben pedig továbbszállítjuk a végtelenített (hurokot alkotó) pályán. A munkadarab így, újra elérve a gyártócellát, többször is próbálkozhat a belépéssel. Ahhoz, hogy ezt elérjük, úgy kell megváltoztatnunk a modellünket, hogy irányítani tudjuk, mikor hagyja el a munkadarab a konvejort.

Az **Advanced Transfer Panel** további öt új modullal: **Access**, **Convey**, **Exit** és **Stop** gondoskodik a konvejorok sokkal pontosabb és részletesebb modellezhetőségéről. Az **Exit** modul kilépteti az entitást a konvejorról, felszabadítva az általa foglalt cellá(ka)t. Alapvetően ez történik akkor, amikor az **Enter** modul párbeszédablakában a Transfer In paraméterhez az „*Exit Conveyor*” (konvejor elhagyása) opciót rendeljük. Az **Access** modul segítségével egy entitás a konvejoron, egy specifikus helyen (általában a pillanatnyi helyzetében) pozíciót vagy hozzáférést igényel. A **Convey** modult arra használjuk, hogy elszállítsa a munkadarabot a célállomására, miután sikeresen hozzáfért a szükséges helyéhez a konvejoron. Amikor a **Leave** modul párbeszédablakában a Transfer Out paraméternek az “*Access Conveyor*” opciót választjuk, gyakorlatilag arra utasítjuk az Arena-t, hogy az entitás érje el a konvejort és mozgassa (továbbítsa) a Station name paraméter által meghatározott helyre. A **Start** és **Leave** modulok elindítják/megállítják a konvejort. Ez a két modul felhasználható a hibaesemények kezelésére, és általában a konvejor állapotok (*Busy*, *Idle*, *Inactivate*) modellezésére.



9.1. ábra: Új logikai hálózat a *„Cella 1”*-nél

Az új modellünk kifejlesztéséhez, felhasználjuk a 8.4. modellt, amelyben nem-akkumuláló konvejort alkalmaztunk és amelyben a „*Belepes a Cella 1 allomasra*” nevű **Enter** és az „*Ut a cella 1_bol*” **Leave** modulokat az új modulokra cseréljük, ahogy azt a *9.1. ábra* mutatja.

A lecserélt **Enter** modulban definiáltuk a „*Cella 1*” nevű állomást, és kiléptünk a konvejorról. Az állomás definiálása azért fontos, mert az Arena-nak tudnia kell, hová küldje az entitást, amit esetünkben a Cella 1 gyártócellába kell mozgatni. Ezért azzal kezdünk a modulcserét, hogy az **Advanced Transfer** panelről egy **Station** modult választjuk, azzal a céllal, hogy meghatározzunk a Cella 1 belépési pontját. Használhattuk volna az **Enter** modult is erre a célra, de alternatívaként bemutatjuk a **Station** modult. A **Station** modul egyszerűen csak a logikai belépés helyét definiálja az állomáshoz érkező entitások számára. Esetünkben, az egyetlen megadott érték a párbeszédablakban az állomás neve, ahogy a *9.1 táblázatban* látható.

9.1. táblázat

A *Station* modul paraméterei

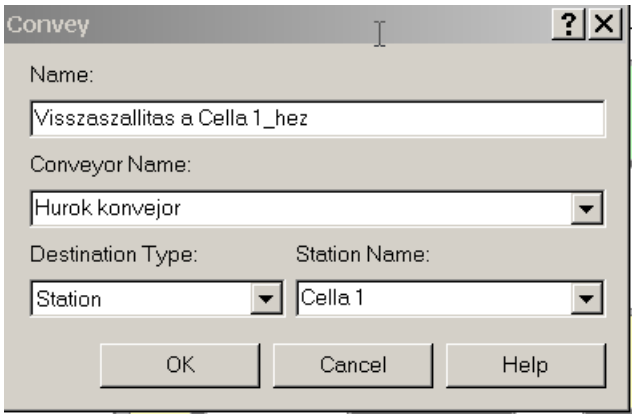
Name	Cella 1 allomas
Station Name	Cella 1

Amikor egy entitás, vagy munkadarab megérkezik a *„Cella 1”*-hez, ellenőrizni kell a várakozó sor állapotát, mielőtt határoznánk a további sorsáról. Ezt úgy érjük el, hogy az entitást egy **Decide** (döntés) modulba küldjük, ahol megvizsgáljuk *NQ*(*Cella 1 gyartas.Queue*) == 0 feltételt. Tudni szeretnénk, hogy az 1 gyártócellához tartozó sor üres-e (*9.2. táblázat*). Ezt a sort a 8.4. modellből örököltük, amely a *„Cella 1 gyartas”* nevű **Process** modulban lett definiálva. Ha a sor az adott pillanatban foglalt, akkor az érkező munkadarab a *False* (hamis) ágon távozik a **Decide** modulból.

9.2. táblázat

A *Decide* modul paraméterei

Name	Van e hely a Cella 1 teruleten
Type	2-way by Condition
If	Expression
Value	NQ(Cella 1 gyartas.Queue) = = 0



Name	Visszaszallitas a Cella 1_hez
Conveyor Name	Loop Conveyor
Station Name	Cella 1

9.1. képernyő: A **Convey** (mozgatás) modul paraméterei

Ebben az esetben az entitás nem hagyja el a konvejort, hanem továbbmozog, és a végtelenített pályán (hurokon) ismét a Cella 1 állomáshoz vezetjük. Ezt úgy érjük el, hogy az entitást a

„*Visszaszallitas a Cella 1_hez*” nevű **Convey** modulhoz küldjük, amelyben a mozgatás célállomása a *„Cella 1”* állomás (*9.1 képernyő*).

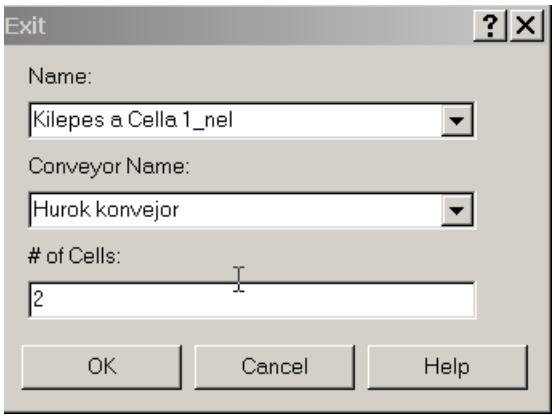
Ennél a pontnál gondolkozzunk el azon, hogy mi történik akkor, ha egy entitás már a Cella 1 állomáson van. Az Arena azt feltételezi, hogy a szándékunk az entitás mozgatása, ezért elküldi a megadott úton. Természetesen feltételezve azt is, hogy fizikailag a mozgatás a kijelölt célállomásra megvalósítható, ami esetünkben igaz. A **Convey** modul ezért az entitást a megadott konvejorral a kijelölt állomásra mozgatja. Ha az entitás még nincs a konvejoron, az Arena megáll, és ezt egy hibaüzenettel jelzi ezt.

9.3. táblázat

A **Delay** modul paraméterei

Name	Lerakodas a Cella 1 teruleten
Delay Time	Lerakasi ido
Units	Minutes

Ha a *„Cella 1”* sora üres, azaz az *NQ(Cella 1 gyartas.Queue)* == 0 feltétel teljesül, akkor a munkadarab belép a *True* (igaz) ágon kapcsolt „*Lerakodas a Cella 1 teruleten*” nevű **Delay** modulba (*9.3. táblázat*), ahol az entitást késleltetjük a *„Lerakasi ido”*-nek megfelelő időtartamig. Ezután az entitást a „*Kilepes a Cella 1_bol*” nevű **Exit** modulhoz (*9.2. képernyő*) küldjük, amely eltávolítja azt a konvejorról, és felszabadítja az általa foglalt cellákat.



Name	Kilepes a Cella 1_nel
Conveyor Name	Hurok konvejor
# of Cells	2

9.2. képernyő: Az **Exit** (kilépés) modul

Az entitást a **Exit** modulból a *9.1. ábrán* látható *„Cella 1 gyartas”* nevű **Process** modulba küldjük, ami az aktuális gyártási folyamatot reprezentálja. Miután a művelet, a munkadarab megmunkálása befejeződött, a munkadarab kilép a **Process** modulból, és a folyamatban következő „*A konvejor elerese a Cella 1_nel*” nevű **Access** modulba kerül, hogy helyet igényeljen a szállítószalagon (*9.3. képernyő*).

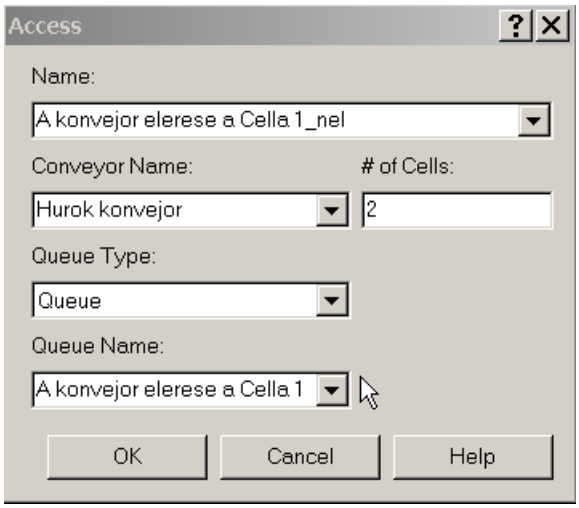
Az **Access** modul cellákat allokál a konvejoron, így egy entitást akkor lehet a következő állomására mozgatni, ha a konvejor éppen nem mozgat más entitást. Ezért, ha azonnal nem kezdeményezzük az entitás mozgatását, akkor az entitás az egész konvejort megállásra kényszeríti a mozgatás kezdetéig.

Abban az esetben, amikor az igényelt cellák nem elérhetőek, az entitás a sorban (*A konvejor elerese a Cella 1_nel.Queue*) marad, amíg a konvejoron nem érhető el hely. Ez egy módja

annak, hogy az entitás megjelenjen az animációban, amikor a konvejor elérésére kell várakoznia.

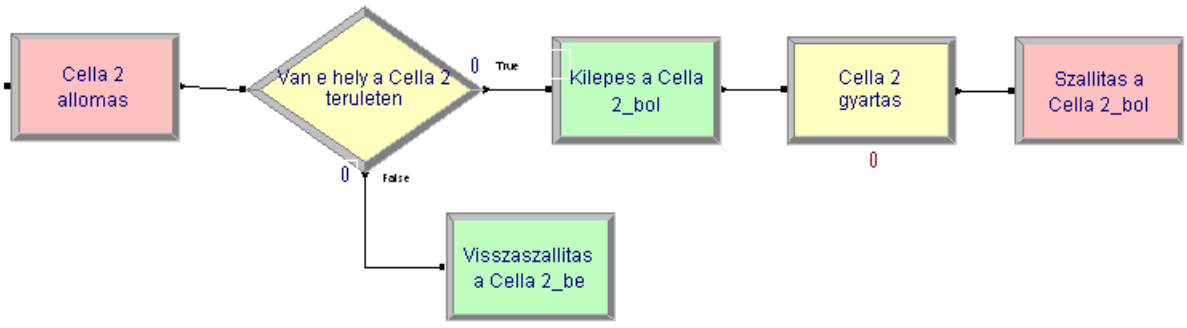
Miután az entitás elérte a konvejort, az entitás a *„Felrakodas a Cella 1 teruleten”* nevű **Delay** modulban megadott a *„Felrakasi ido”* idejéig késleltetve lesz. Ezt követően a *„Szallitas a Cella 1_bol”* nevű **Convey** modulba kerül, ahonnan meghatározott sorrend (szekvencia) szerint mozgatjuk tovább. Ezzel befejeztük a modulcserét a *8.4. modellben*, a *„Cella 1”* állomáson.

Ugyanezeket a műveleteket a *„Cella 2”* állomáson is el kell végezni. Egyszerűen ismételjük meg az előző lépéseket, vagy másoljuk le ezeket a modulokat, vagy szerkesszük meg önállóan. A Cella 2-nél az **Enter** modult lecseréltük és a **Leave** modult meghagytuk (*9.2. ábra*). Feltételezzük, hogy az eddigiek alapján már mindenki könnyedén el tudja végezni a szükséges változtatásokat.



Name	A konvejor elerese a Cella 1_nel
Conveyor Name	Hurok konvejor
# of Cells	2

9.3. képernyő: Az **Access** modul



9.2. ábra: Az új modell folyamatábrája a *„Cella 2”*-nél.

Észrevehetjük, amikor kitöröljük az **Enter** és **Leave** modulokat, akkor az animáció egy része is törlődik. Ennek magyarázata, hogy a **Leave** modulhoz egy “szabad” animációs sor tartozik. Amikor egy ilyen modellt szerkesztünk, vagy ilyen módon megváltoztatunk gyakran könnyebb az új tulajdonságokat az **Animate** eszköztár használatával felépíteni. Így amikor a mo-

dellünket fejlesztjük, az új modulokkal automatikusan hozzáadott animációs tulajdonságokat töröljük, és magunk gondoskodunk az új tulajdonságok animálásáról. Modellünkben töröltük a sorokat és a *„Cella 1”* tárolót és egyszerűen hozzáadtuk az új access queue-t. Az új modell futtattatásakor azt láthattuk, hogy a munkadarabok a belépéstől a Cella 1-ig blokkolva voltak, és a 275 perc eltelte után indultak el.

9.1.2. A munkadarabok a szállítószalagon maradnak a feldolgozás alatt (9.2. modell)

Most, miután megismertük ezeket az új konvejor modulokat, tekintsünk meg egy másik problémát is. Kezdjük az akkumuláló konvejor modellel (8.5. model, a 8.4.2. bekezdésből). Feltételezzük, hogy egy új elrendezést próbálunk ki, amelyben a *„Cella 2”* műveleteit úgy végezzük el, hogy a munkadarab a művelet időtartama alatt a konvejoron marad. Pontosabban, a darabok, amelyeket a *„Cella 2”*-höz szállítunk, nem hagyják el a konvejort, hanem megállnak a *„Cella 2”* állomáson. Az aktuális művelet végrehajtása közben a munkadarab a konvejoron van, majd a művelet elvégzése után továbbszállítjuk a következő állomásra. Az akkumuláló konvejoron a munkadarabok folyamatosan haladnak, kivéve ha a *„Cella 2”*-ben feldolgozás alatt álló munkadarab ebben nem akadályozza azokat.

Megvalósíthatnánk ezt az a módosítást oly módon is, hogy a jelenlegi **Enter** és **Leave** modult lecseréljük a *„Cella 2”*-nél egy olyan modulcsomaggal, ami hasonló ahhoz, amit a *9.1. modellhez* készítettünk. Azonban, ha figyelembe vesszük, hogy a *„Cella 2”*-höz érkező entitások, nem hagyhatják el a konvejort, van egy jóval egyszerűbb megoldás is. Módosítsuk **Transfer In** opciót az **Enter** modulban úgy, hogy a *None* (egyik sem) opciót választjuk. Ebben az esetben az entitás nem fogja elhagyni a konvejort, ez által ott marad addig, amíg a kiszolgáló által definiált műveletsor befejeződik. Viszont, ha csak ezt az egy módosítást végezzük el, hibaüzenetet kapunk, amikor az entitás megpróbálja elhagyni a *Cell 2* állomást. A **Leave** modul párbeszédablakában az korábban kiválasztott **Transfer Out** opció, az *Access Conveyor* arra kényszeríti az entitást, hogy próbáljon helyet foglalni a konvejoron, és mivel az entitás a konvejoron marad, az Arena összezavarodik, és leállítja a folyamatot runtime error (futtatási hiba) üzenettel. Ezt egyszerűen javíthatjuk: a **Leave** modulban a **Transfer Out** párbeszédabla **Logic** részében válasszuk a *None* opciót. Ezek a szükséges modell átalakítások megoldják a problémánkat.

Kipróbálhatjuk az új modellt, ha megnézzük az animációt. Javasolt a szimulációt először a fast-forward sebességgel futtatni, mielőtt elkezdjük az animációt figyelni. A szimulációnak ezen a pontján a változtatások már elég látványosak.

9.2. További ismeretek a transzporterekről

A 8. fejezetben bemutattuk Arena transzporterekkel és konvejorokkal kapcsolatos modellezési fogalmakat a **Basic Process** panelen található magas-szintű modulok funkcióit használva. A 9.1. bekezdésben kiterjesztettük ezeket a funkciókat az **Advanced Transfer Panel**-en elérhető modulok segítségével azért, hogy módosítsuk és finomítsuk a 8. fejezetben bemutatott modelleket. Ehhez hasonló lehetőségek a transzporterek esetében is léteznek. Habár nem fogunk kifejleszteni teljes modelleket e modulok segítségével, de rövid áttekintést adunk a különböző szituációhoz szükséges funkciókról és modulok sorozatáról. Ez elegendő lehet ahhoz, hogy később e konstrukciókat sikeresen használjuk a saját modelljeinkben. Emlékezzünk arra, hogy az **online help** is mindig elérhető.

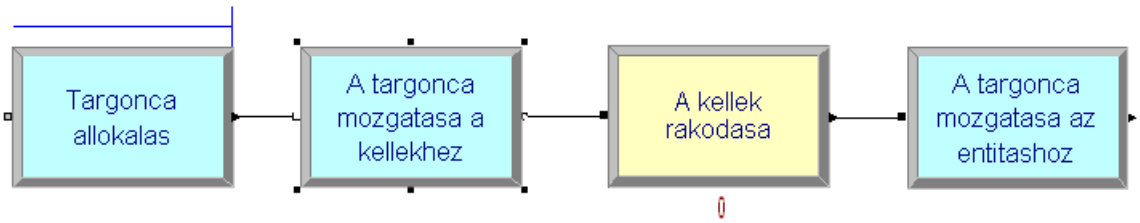
Kezdjünk az alap lehetőségekkel, amelyeket a 8.3. bekezdésben már érintettünk. Az alapvető transzporter modellezési szerkezetek megvalósíthatók az **Enter** és **Leave** modulok *Transfer In* és *Transfer Out* opcióival. Tekintsünk egy transzporter igényű folyamatot és egy ezt követő

entitásszállítást a transzporterrel egy megadott állomásra. Az **Advanced Transfer Panel**-en található **Request** és **Transport** modulok is alkalmasak e feladat megoldására.

A **Request** modul helyettesíti a **Leave** modul **Transfer Out** *Request Transporter* opcióját, és majdnem azonos azzal. A **Request** modulban nyereségként lehetőséget kapunk az alapértelmezett sebesség (Velocity) felülírására, de elvesztjük a rakodási idő (Delay) megadásának a lehetőségét. Ezt a veszteséget úgy pótolhatjuk, hogy az entitást egy **Delay** modulhoz irányítjuk és abban adjuk meg a rakodási időt. A **Request** modul valójában két műveletet végez: egy transzportert rendel az entitáshoz, és az üres transzportert az entitáshoz mozgatja, feltéve, hogy a transzporter még nincs az entitásnál. A **Transport** modul elvégzi a **Transfer Out** következő műveletét, azáltal, hogy elindítja a transzportert és az entitás szállítását a megadott helyre. Most tekintsünk egy olyan modellezési szituációt, amikor kívánatos lenne ezt a két funkciót külön választani. Feltételezzük, hogy amikor a transzporter megérkezik az entitáshoz, az entitás tartózkodási helyén rakodásra van szükség, ami egy operátor (rakodómunkás) közreműködését igényli. Az operátor, mint erőforrás, egyértelmű modellezése az új modulokkal valósítható meg. A szükséges modulsor (**Request**–**Process**–**Transport**) a *9.3. ábrán* látható.

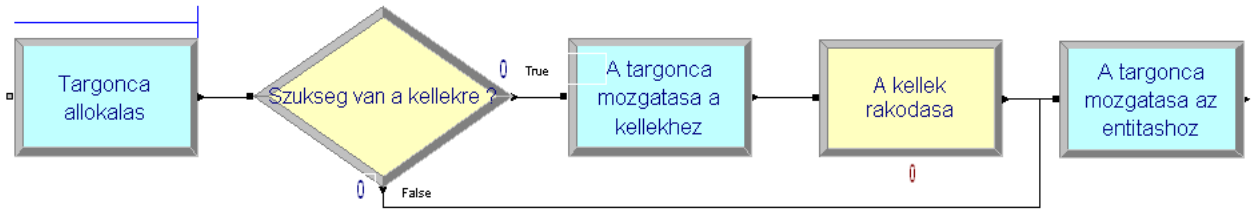
9.3. ábra: Rakodómunkás közreműködésével végzett rakodás

Hasonlóképpen a **Request** modul két aktivitása is szétválasztható az **Allocate** és **Move** modulokkal. Az **Allocate** modul hozzárendeli a szállítóeszközt az entitáshoz, de a szállítóeszköz helyben marad. A **Move** modul lehetővé teszi, hogy az entitás a hozzárendelt transzportert a modellen belül bárhova mozgassa. A **Request** modul használatakor a transzporter automatikusan az entitás pozíciójához kerül. Képzeljünk el egy olyan helyzetet, amikor az üres transzporternek fel kell szednie egy, az entitás szállításához szükséges kelléket (pl. raklapot) adott munkaterületen. Ebben az esetben először allokálni kell a transzportert, majd azt arra a munkaterületre kell küldeni, ahol felszedni a kelléket, végül a transzportert el kell küldeni az entitás helyéhez. E tevékenységek modellezése az **Allocate**–**Move**–**Process**–**Move** modulsorozattal végezhető el (*9.4. ábra*).



9.4. ábra: Kellék (pl. rakodólap) szükséges az entitásszállításhoz

Természetesen, sokféle módon lehet modellezni az ilyen folyamatokat. Például feltételezzük, hogy csak az entitások egy részének szállítása igényel kelléket. Ilyenkor a feltétel ellenőrzésére egyszerűen egy **Decide** modult alkalmazhatunk (*9.5. ábra*).



9.5. ábra: A kellékigény ellenőrzése

Az **Allocate** és **Move** modulokkal szerkesztett két példánkban mindkét eset azt eredményezte, hogy a transzporter az entitás pozíciójához mozgott. Most feltételezzük, hogy a transzporter felszabadulásakor azt szeretnénk, hogy a transzporter a rendszerben bolyongva keressen munkát magának, természetesen egy előre definiált útvonalon. Ez az útvonal hasonló állomások sorozatához, azzal a különbséggel, hogy zárt hurokot alkot. Minden olyan esetben, amikor a le nem kötött transzporter eléri a következő állomást az útvonalon, ellenőriznie kell, hogy van e szállítási igény az állomáson. Ha nincs, akkor továbbhalad, de ha van, akkor azonnal kérvényező helyre mozog.

Ennek modellezéséhez egy egyedi entitást kreálnunk (ezt nevezzük hurokentitásnak), ami nagyon alacsony prioritással (nagy szám a prioritásra) próbálja allokálni a transzportert. Amint ez megtörtént, az üres transzporter a következő állomásra mozog az útvonalán. Az állomásra érkezésekor a transzporter felszabadul, és készen áll az új igények teljesítésére. A hurokentitás a kezdeti **Allocate** modulba irányítódik, ahol újra megpróbálja a transzportert allokálni. Ez azt feltételezi, hogy minden pillanatnyi transzporter igény magasabb prioritású, mint a hurokentitás. Ne felejtsük, hogy az alapértelmezett prioritásérték 1, és a legkisebb érték jelenti legnagyobb prioritást.

Amint egy transzporter megérkezik a célállomására, fel kell szabadítani. A **Free** (felszabadítás) modul gondoskodik erről a funkcióról. A modul paramétereinek megadásához csak a transzporter nevét kell beírni a párbeszédablakba, és néhány esetben a transportert azonosító nevet is. Két további modul, a **Halt** és az **Activate** modul az aktív vagy elérhető transzporterek irányítását teszik lehetővé. A **Halt** modul egy transzporter egységet inaktivál, vagy tesz elérhetetlenné. Az **Activate** modul egy egység inaktív transzportert helyez aktív vagy elérhető állapotba.

9.3. Entitás visszalépés

9.3.1. Entitás akadályozottság és visszalépés

Az **entitások akadályozottsága** akkor lép fel, amikor az entitás vagy vásárló nem csatlakozik a sorhoz mert nincs hely. A vásárló lemond a kiszolgálásról és elmegy, vagy valahol máshol keres kiszolgálót. **Entitás visszalépésről**, viszont akkor beszélünk, amikor az entitás vagy a vásárló érkezéskor csatlakozik a sorhoz, de később úgy dönt, hogy kilép a sorból, valószínűleg bánva, hogy nem így döntött azonnal. Tekintsük a jóval bonyolultabb esetet, amikor mindkét esemény egyszerre jelentkezik. Először is, definiáljuk a lehetőségeket, ahogy az akadályok és a visszalépések történhetnek.

A legtöbb szolgáltatási rendszer rendszerkapacitása véges, ami gyakorlatilag a várakozási helyek mennyiségét jelenti. Amikor az összes hely foglalt, a vásárló nem képes belépni a rendszerbe, azaz akadályozott. Ez a legegyszerűbb formája az entitás akadályozottságnak. Sajnos a legtöbb rendszer, ahol ez felmerül az akadályozottság, lényegesen bonyolultabb. Tekintsünk egy egyszerű kiszolgáló vonalat elméletileg végtelen sorral vagy a várakozási

kapacitással. A legtöbb esetben a véges kapacitás korlátja a rendelkezésre álló hely, de elképzelhető, hogy a kiszolgálósor kilép az épületből, és az utcán, a blokk körül kígyózik. Ilyen esetekben, nincs számszerűen megadható kapacitás korlát. Ilyenkor a vásárlók döntései (belép vagy elmegy) a helyzet kiértékelésén alapulnak, azaz az akadályozottság vagy a kapacitás gyakran entitásfüggő. Az egyik vásárló úgy dönt, hogy hosszú kiszolgálási sor esetén nem várakozik, a másik pedig belép ugyanebbe a sorba.

Az entitás visszalépés még bonyolultabb kérdéskör, mind modellezési, mind megjelenítési szempontból a szoftverben. Minden entitás vagy vásárló, aki belép a sorba, különböző toleranciaküszöbbel rendelkezik, abban a tekintetben, hogy ki mennyit hajlandó várni, mielőtt elhagyná a sort. A döntés gyakran azon múlik, hogy az ügyfél mennyire akarja igénybe venni az adott szolgáltatást. Van olyan ügyfél, aki a várakozási idő hosszától függetlenül is hajlandó várakozni, és van olyan, aki viszont egy idő után elmegy, mert rájön, hogy az általa elvárt időn belül nem lesz kiszolgálva. A vásárlók döntését gyakran az befolyásolja legjobban, hogy mennyi ideje várnak, és hol állnak a sorban. A vásárló beállhat a sorba úgy is, hogyha tíz percen belül nem szolgálják ki, akkor elmegy, de miután eltelt a tíz perc, dönthet úgy is, hogy még marad, ha már közel áll a kiszolgáláshoz.

A sorváltás, vagy "zsokizás" egy még bonyolultabb formája a visszautasításnak, ami a szupermarketek pénztárainál, gyorséttermekben és bankoknál fordul elő, tipikusan ott, ahol több sort "üzemeltetnek". A vásárló kiválaszt egy sort, majd később átértékeli a döntését, a pillanatnyi sorhosszúságok függvényében. Mindenesetre választástól függetlenül, mindig a leglassabb sorban állunk, és ha váltunk, akkor hirtelen felgyorsul az elhagyott sor. A sorváltás logikáját itt nem tárgyaljuk.

9.3.2. Egy szolgáltatási modell visszalépéssel és akadályozottsággal (9.3. modell)

Vizsgáljuk meg az akadályozottságot és a visszalépést egy egyszerű modell segítségével. A vásárlók érkezése EXPO(5) érkezési időközzel írható le a egy kiszolgálós kiszolgálási rendszerben, amelyben a kiszolgálási idő EXPO(4,25). Minden idő mértékegysége percben adott. Annak ellenére, hogy a sor kapacitása végtelen, minden érkező vásárló összeveti a sor hosszúságát a saját várakozási toleranciaküszöbével. Ha a sor hossza nagyobb, mint a tűrőképessége, nem lép be a sorba és elhagyja a rendszert. Az ügyfelek tűréshatárát háromszög eloszlással jellemezzük TRIA(3,6,15). Mivel a mintánkat folytonos (exponenciális) eloszlásból generáljuk, a minta nem lesz egész szám. Használhatnánk az Arena beépített matematikai függvényeit, hogy egész számmá konvertáljuk, de minket csak az érdekel, hogy ez a szám nagyobb-e mint a generált tűréshatár.

Két módszer létezik az akadályozottság modellezésére. Tegyük fel, hogy létrehozzuk az érkezéseinket, generáljuk a tűréshatár értékét a háromszöges eloszlásból, és hozzárendeljük ezt az értéket, mint attribútumot az entitáshoz. Elküldhetnénk az érkezéseinket egy **Decide** modulba, és összevethetnénk a minta értékét a pillanatnyi sor hosszával, használva az NQ Arena változót. Ha a tűréshatár kisebb vagy egyenlő, mint a pillanatnyi sor hossza, akkor az ügyfél akadályozottság miatt nem lép be a rendszerbe. Különben pedig belép a várakozó sorba. Alternatív megoldásként a tűréshatárt hozzárendelhetjük egy változóhoz, és a kiszolgálósor kapacitáshoz egy hasonló változót használhatunk. Aztán az érkezéseinket közvetlenül a kiszolgálóhoz küldhetjük. Ha a pillanatnyi sor hossza nagyobb vagy egyenlő, mint a tűréshatár, ami egyenlő a sor kapacitásával, az érkezők automatikusan nem állnak be a sorba. Az utóbbi módszert használva, lehetséges, hogy a sor olyan teherbíró képesség értéket vegyen fel, ami kisebb, mint az emberek száma a sorban. Ez a módszer azért működik, mert az Arena csak a sor kapacitást ellenőrzi, amikor egy új entitás próbál belépni a sorba. Ezért a pillanatnyi entitások

biztonsággal a sorban maradnak, függetlenül az új sor-kapacitás értékétől. A modellünk fejlesztésekor a második módszert fogjuk választani.

A visszalépés bemutatásához feltesszük, hogy az érkező vásárlók, akik nem mennek el azonnal, hajlandók véges ideig várakozni, mielőtt kilépnek a sorból. Ezt a visszalépési tolerancia-időt egy Erlang eloszlásból generáljuk ERLA(15,2), aminek a középértéke 30, és attribútumként hozzárendeljük a **Create** modulban létrehozott entitásokhoz. A visszalépési aktivitás mechanizmusának a modellezése kihívás lenne. Ha az érkezőket hagynánk belépni a sorba, a visszalépés tűrésidejének elérésekor meg kellene találnunk és el kellene távolítanunk az entitásokat a sorból. A problémaleírás e pontjánál, érdemes elgondolkozni a helyzetkezelés más, alternatív megoldásain. Például, definiálhatnánk egy változót, ami képes követni, hogy kiszolgáló egység mikor lesz legközelebb elérhető. Először az entitás-feldolgozási időt generáljuk, és azt attribútumként hozzárendeljük az entitáshoz egy **Assing** modulban. Aztán elküldjük az entitásunkat a **Decide** modulhoz, ahol ellenőrizzük akadályozottságot. Ha nincs akadályozva, ugyanebben a modulban ellenőrizzük, hogy az entitás hozzájut-e a szolgáltatásához a visszalépési tűréshatár elérése előtt. Ha a visszalépési tűréshatárt előbb éri el, mint a kiszolgálást, akkor eltávolítjuk a sorból. Különben pedig elküldjük az entitást egy **Assign** modulhoz, ahol frissítjük a szerver legkorábbi elérhetőségének idejét leíró változót, és aztán az entitást a sorba küldjük. Ez a logika komplikáltnak tűnhet, amit a következőképpen lehet összefoglalni:

```
Define Available Time = Time in the future when server will be available
        Create arrival
        Assign Service Time
Assign the Time in the future the activity would renege, which is equal to Tolerance Time + TNOW
Assign Balk Limit
Decide
If Balk Limit > Number in queue Balk entity
If Renege Time < Available Time Renege entity
Else
Assign Available Time = MX(Available Time , TNOW) + Service Time
Send entity to queue
```

10. Kanban-szabályozású gyártási rendszerek modellezése

A kanban-elvű gyártásirányítás a 70-es évek végén és a 80-as évek elején került a nemzetközi érdeklődés középpontjába. A kanban rendszer elemzése és fejlesztése ezekben az években vált a kutatóintézetek népszerű témájává, és erre az időre tehető, amikor a gyártással foglalkozó amerikai és európai vállalatok a termelési folyamataik irányítására elkezdték használni a Japánban sikeresnek bizonyult módszert. Az eredeti rendszert a Toyota Motor Corporation fejlesztette ki, és az alapelveit már meglehetősen régen, 1954-ben lefektette. Az eljárás hosszú kísérleti szakasz után, 1970-re vált technikailag megalapozottá. Azóta a gyakran Toyota gyártási rendszerként (TPS) is emlegetett rendszer, amelynek csak egyik eleme a kanban, világszerte ismertté vált. A kanban rendszer a megkülönböztetett figyelmet elsősorban annak köszönheti, hogy teljesen különbözik a tradicionális toló elvű gyártásirányítási rendszerektől. A kanban rendszerben a húzóelv érvényesül, amelyben a munkadarabok haladását ellentétes irányú információs folyamatok szabályozzák.

A kanban-elvű irányításban rejlő lehetőségeket, illetve az ennek köszönhető hatékonyságnövekedést számos elméleti és empirikus tanulmány dokumentálja. A termelésmenedzsment témakörében megjelent, a kanban rendszerrel foglalkozó tanulmányok jelentős hányada a feltöltési ciklusidő, a kanban méret és egyéb rendszerparaméterek meghatározására irányuló kutatásokról számol be. Ezekben módszerként leggyakrabban a matematikai programozást, a sorbanállási elméletet és a szimulációt alkalmazzák. Az analitikus (matematikai) elemző módszerek a kulcsfontosságú rendszermutatók gyors meghatározásához szükségesek, de alkalmazhatóságuk korlátozott, ezzel szemben a számítógépes szimuláció általánosan használható módszer a rendszerműködés analizálására.

Ebben a fejezetben egy olyan modulsorozatot mutatunk be, amelyek segítségével a kanban-elvű többlépcsős gyártási rendszerek szimulációs modelljei gyorsan és könnyen kifejleszthetők. A modulsorozat egytermékes, többlépcsős, egy- és kétkártyás kanban rendszerek építésére használható.

10.1. A kanban rendszerek csoportosítása

A kanban vezérelt gyártási rendszerek legáltalánosabb változatai, többféle terméket gyártanak ugyanazokkal, de többféle termék megmunkálására alkalmas gyártóeszközökkel. A gyártóeszközök mellett a rendszer magában foglal egy gyártásütemező részleget, egy kimeneti tárolót, konténereket a munkadarabok szállításához és tároláshoz, valamint termékenként egy-egy kanban kártya garnitúrát.

Tradicionálisan a kanban egy információhordozó kártya. A kimeneti tárolóban minden terméktároló konténerhez kapcsolódik egy-egy kanban kártya. A kártyák száma limitált, ami korlátozza a termékek maximális mennyiségét a rendszerben. Amikor egy konténer elhagyja a kimeneti tárolót, a kísérő kanban kártya elszakad a konténertől (aktívvá válik) és visszakerül a gyártásütemező részleghez. Abban az esetben, ha a konténer a gyártónál marad, a kanban kártya akkor szakad el a konténertől, amikor abból az utolsó tételt kiemeltük, ami ekvivalens azzal, hogy kötött számú konténert használunk a maximális készletszint korlátozására. Előfordul az is, hogy a kiemelt kanban kártyákat egy, a kimeneti tárolóban elhelyezett ládában gyűjtjük, mielőtt azokat visszajuttatjuk az ütemező részleghez. A visszajuttatás történhet akkor, amikor meghatározott számú kártya összegyűlt, vagy akkor, amikor előre meghatározott idő eltelt az utolsó visszajuttatás óta. A konténertől elszakított ún. aktív kanban kártya a kártyán található információk alapján felhatalmazza a termelést egy konténer mennyiségű munkadarab legyártására. Miután a konténer megtelik az előírt számú munkadarabbal a konténer-

hez csatoljuk a kártyát, amely ilyen módon inaktívvá válik, és a konténert a kártyával együtt a kimeneti tárolóba továbbítjuk.

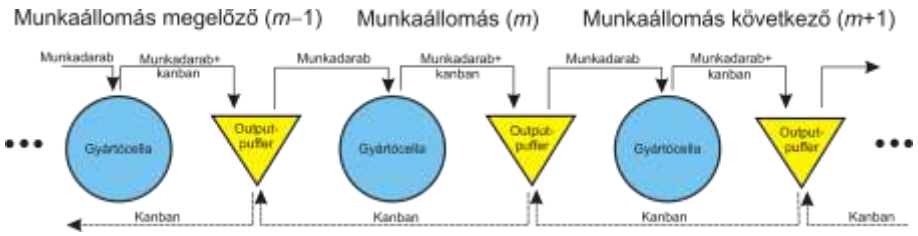
A többlépcsős kanban rendszerek a konténer- vagy anyagtranszfer szabályai szerint csoportosíthatók. Az anyagtovábbítás egy gyártási lépcső (*m*) output tárolójából a következő gyártási lépcső (*m*+1) input tárolójába történik. Az irodalom legalább négy különböző szabály található. Egyes rendszerekben a gyártási lépcső output tárolója a következő gyártási lépcső input tárolója. Következésképpen ilyenkor nincs szükség fizikai mozgatásra. Más rendszerekben, ahol az output- és az inputtárolók elkülönülnek egymástól a konténerek továbbítása az outputtárolóból az inputtárolóba különböző időpontokban hajtható végre. A négy különböző anyagtovábbítási sémát az *10.1. táblázat* összegzi. A továbbiak megértése érdekében ezekről az anyagtranszfer típusokról rövid áttekintést adunk.

10.1. táblázat

Az anyagtranszfer sémák osztályozása

m-edik lépcső Outputtároló = *m* + 1-edik lépcső Inputtároló	*m*-edik lépcső Outputtároló ≠ *m* + 1-edik lépcső Inputtároló		
Típus I Átszállítás közvetlen a termelés indítása előtt	**Típus 2** Átszállítás azonnal a kanban aktivizálódása után	**Típus 3** Kötött mennyiség, változó átszállítási ciklusidő	**Típus 4** Kötött átszállítási ciklusidő, változó mennyiség
Egykártyás rendszer		Kétkártyás rendszer	

1 típusú anyagtranszfer. A gyártási lépcső outputtárolója egyidejűleg a következő gyártási lépcső inputtárolója, és az anyagtovábbítás az outputtárolóból közvetlen a gyártás indítása előtt kezdődik (*10.1. ábra*). Ezt a sémát *késleltetett anyagtranszfernek* nevezik.

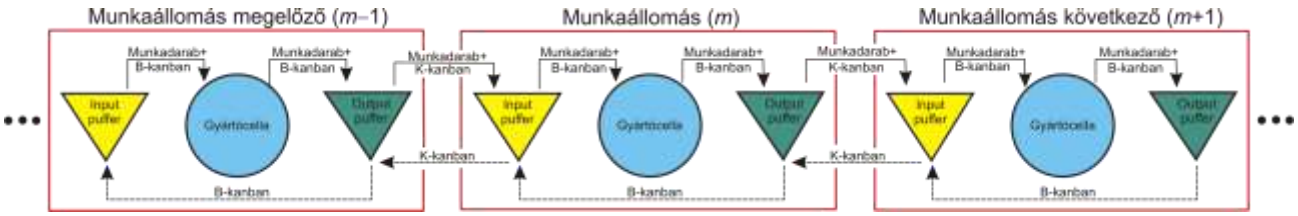


10.1. ábra: Egykártyás kanban rendszer (1 típusú)

2 típusú anyagtranszfer. A gyártási lépcső output tárolója fizikailag elkülönül a következő gyártási lépcső inputtárolójától. Az anyagot az *m*-edik lépcső outputtárolójából *m*+1-edik lépcső inputtárolójába szállítjuk azonnal, amikor a kanban aktivizálódott (az aktív kanban engedélyezi egy inputanyagot tartalmazó konténer átszállítását). A kanban ezután a konténerhez kapcsolódik (a kanban inaktívvá válik), és együttmozogva belépnek az *m*+1-edik lépcső gyártóeszközei előtti sorba. Ha az *m*+1-edik lépcsőben a kanban aktív, és az *m*-edik lépcső outputtárolója üres, akkor a transzfer addig késlekedik, amíg az *m*-edik gyártási lépcsőben be nem fejeződik a konténer feltöltése az igényelt munkadarabokkal. Ezt a sémát *azonnali anyagtranszfernek* nevezik.

3 típusú és 4 típusú anyagtranszfer. A gyártási lépcső outputtárolója fizikailag elkülönül a következő gyártási lépcső inputtárolójától. A lépcsők közötti mozgatások szervezésére ezek a transzfertípusok további, az angol nyelvű irodalomban átszállító mozgató, továbbító vagy szállító kanbannak nevezett kártyakészleteket használnak. A tanulmányban ezeket a kártyakészleteket **külső** kanbannak, a gyártási lépcsőn belüli mozgatásokat szervező kanbanokat pedig **belső** vagy gyártó kanbannak fogjuk hívni (*10.2. ábra*). Tekintettel a kétféle kártyára, ezeket a rendszereket **kétkártyás** kanban rendszereknek nevezik, megkülönböztetve az egyféle kártyát használó **egykártyás** kanban rendszerektől. A külső kanban a gyártási lépcső input-

tárolójában nem üres konténerhez csatolt, azaz inaktív. Amikor a konténer tartalmát felhasználtuk a külső kanban elszakad a konténertől (aktívvá válik), és egy kanbangyűjtő ládába kerül. Végül a külső kanbanokat kiszedik a ládából és megelőző gyártási lépcső outputtárolójába viszik. Itt minden aktív külső kanbanhoz hozzárendelünk egy feltöltött konténert. Ezekből eltávolítjuk a korábban hozzájuk csatolt belső kanbanokat és azokat külső kanbanokkal helyettesítjük. Ezt követően a konténereket átszállítjuk az *m*+1-edik lépcső inputtárolójába. Az eltávolított belső kanbanokat összegyűjtjük, és végül azokat visszajuttatjuk az *m*-edik lépcső ütemező részlegéhez.

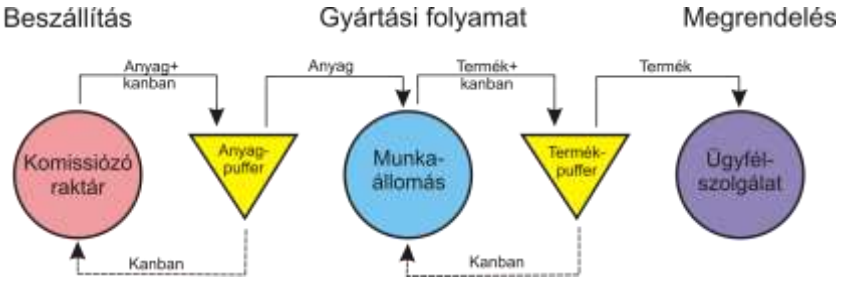


10.2. ábra: Kétkártyás kanban rendszer

A külső kanbanok kiemelésének és visszaszállításának időpontja a gyűjtőládából kétféle módon definiálható: (1) adott állandó mennyiség változó átszállítási ciklusidővel (3 típusú anyagtranszfer), (2) állandó átszállítási ciklusidő változó mennyiséggel (periodikus anyagkezelés, 4 típusú anyagtranszfer). Az első sémánál akkor távolítjuk el a külső kanbanokat, amikor előre meghatározott számú kártya akkumulálódott a gyűjtőládában. Ebben az esetben az átszállítási ciklus hossza, az egymást követő anyagtranszferek között eltelt idő változó lehet. A második sémánál a kártyákat periodikusan, az előre definiált ütemet követve emeljük ki a gyűjtőládából. Ennél a típusnál a ciklusidő állandó és a kártyák száma változhat. Megjegyezzük a 3 típusú anyagtranszfer azonos a 2 típusú anyagtranszferrel, ha az állandó, előre meghatározott kártyaszám egy.

10.2. Az egylépcsős, egykártyás kanban rendszer modellje

Ebben a pontban egy egytermékes, egylépcsős és egykártyás kanban rendszer modelljét mutatjuk be, amely három, a megrendeléseket generáló, a gyártási folyamatot szimuláló és a gyártás alapanyag ellátását biztosító szegmensre vagy állomásra bontható (*10.3. ábra*).



10.3. ábra: Egykártyás kanban rendszer

10.2.1. A probléma ismertetése

A kanban rendszer által gyártott termékeket igénylő ügyfelek véletlenszerűen, exponenciális eloszlás szerint érkeznek, 10 perces átlagos érkezési időközzel és egy konténer mennyiségű terméket igényelnek. A rendelés előkészítés időtartama 5 perc. Az ügyfélszolgálat az ügyfeleket a termék pufferből szolgálja ki, és ennek a puffernek a nagysága 2 konténer. Ez a puffer a gyártási folyamatot realizáló munkaállomás output puffere. Az ügyféligény kielégítésével egyidejűleg aktívvá váló kanban kártyát a munkaállomásra küldjük, azaz kezdeményezzük a gyártást, illetve a puffer feltöltését.

A munkaállomáson a gyártás indításának az egyik feltétele az aktív kanban kártya jelenléte, a másik feltétele pedig, hogy a gyártáshoz szükséges alapanyagok rendelkezésre álljanak. A munkaállomás az alapanyagok tárolására alkalmas anyag pufferrel rendelkezik, amely egyidejűleg a raktár output puffere, és a kapacitása 2 konténer (*10.3. ábra*). Így a munkaállomáson az aktív kanban kártya érkezésekor a gyártás azonnal megkezdődhet, ha az anyag puffer nem üres.

Ha a termelés feltételei adottak, akkor megkezdődik egy konténer mennyiségű termékek gyártása és ezzel egy időben gondoskodunk az anyag puffer feltöltéséről. A termeléshez szükséges gyártóeszközök mennyisége egy, és egy konténernyi termék előállításának műveleti ideje 10 perc.

Az alapanyag-ellátást biztosító raktárban a komissiózási idő 5 perc. A raktár kapacitását végtelen nagynak tekintjük, ami azt jelenti, hogy a folyamat működését veszélyeztető anyaghiány soha nem jelentkezhet.

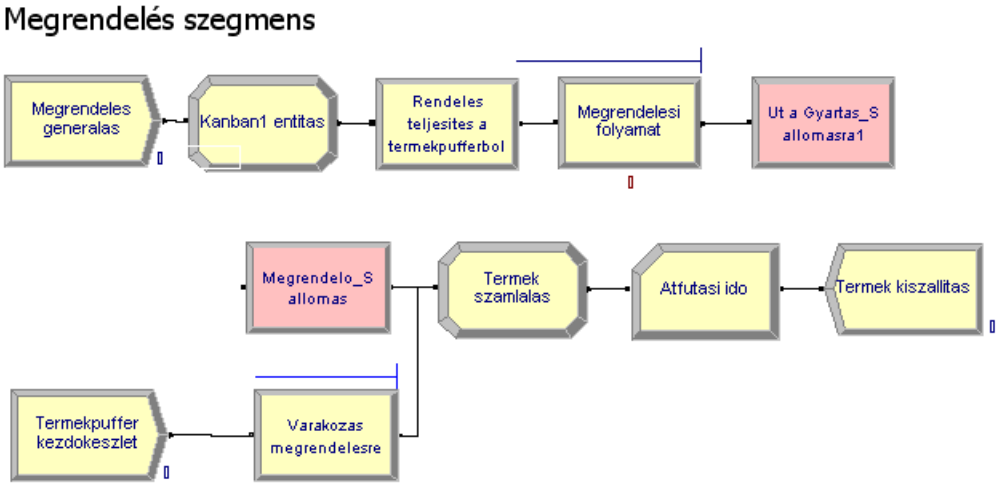
A szállítási idő a raktár és a munkaállomás, illetve a munkaállomás és az ügyfélszolgálat között egyaránt 5 perc. A kanban kártyák továbbításának időtartama minden relációban 2 perc. A szimulációs idő hossza legyen 100 óra.

A modellezés eredményeként szeretnénk megismerni:

- A folyamatok állapotainak valószínűségeit (idle, busy).
- A várakozó kanbanok és megrendelések átlagos számát.
- Az alapanyag puffer átlagos készletét.
- A megrendelések, a gyártott termékek és az anyagbeszállítások számát.
- A teljesített megrendelések számát.

10.2.2. A probléma Arena modellje (10.1. modell)

A probléma Arena modelljének a felépítése a valóságos folyamatot követve három szegmensből áll: megrendelés, gyártás és beszállítás.



10.4. ábra: Az egykártyás kanban rendszer megrendelés szegmense

A megrendelés szegmens

A megrendelés szegmens (*10.4. ábra*) „*Megrendeles generalas*” nevű **Create** moduljában létrehozzuk a *Kanban1* nevű entitást. A modul paramétereit a *10.1. táblázat* foglalja össze. A „*Kanban1 entitas*” nevű **Assign** modulban az entitás tulajdonságait, az entitás képét, az állomás nevét adjuk meg, valamint a „*Megrendelesek szama”* nevű változóban számláljuk a beérkezett megrendeléseket (*10.2. táblázat*). A „*Rendeles teljesites a termekpufferbol*” nevű **Signal** modul egy jelet küld a „*Varakozas megrendelesre*” nevű **Hold** modulba (*10.5. ábra*),

ahol a „*Termekpuffer kezdokeszlet*” nevű **Create** modulban létrehozott *Termek* entitások várakoznak megrendelésre. A modulban a Type mező értéke *Wait for Signal*, és egy jel érkezésekor egy várakozó entitás továbbhaladását engedélyezzük, amit a Limit mezőben korlátozunk. A „*Termekpuffer kezdokeszlet*” **Create** modulban a termék puffer méretének megfelelő számú entitást hozunk létre egy alkalommal, a szimuláció indításakor (*10.3. táblázat*)**.** A termék puffer méretét a „*Termekpuffer merete*” nevű változó definiálja, aminek az értéke a problémaleírás szerint 2 konténer.

10.1. táblázat

A „*Megrendeles generalas*” nevű **Create** modul paraméterei

Name	Megrendeles generalas
Entity Type	Kanban1
Time Between Arrivals	
Type	Random(Expo)
Value	10
Units	Minutes
Entity per Arrival	1
Max Arrivals	Infinite
First Creation	0

10.2. táblázat

Az „*Kanban1 entitas*” nevű **Assign** modul paraméterei

Name	Kanban1 entitas
Type	Entity Picture
Entity Picture	Picture.Blue Page
Type	Attribute
Attribute Name	Entity Station
New Value	Megrendelo_S
Type	Attribute
Attribute Name	ErkIdo
New Value	TNOW
Type	Variable
Variable Name	Megrendelesek szama
New Value	Megrendelesek szama + 1

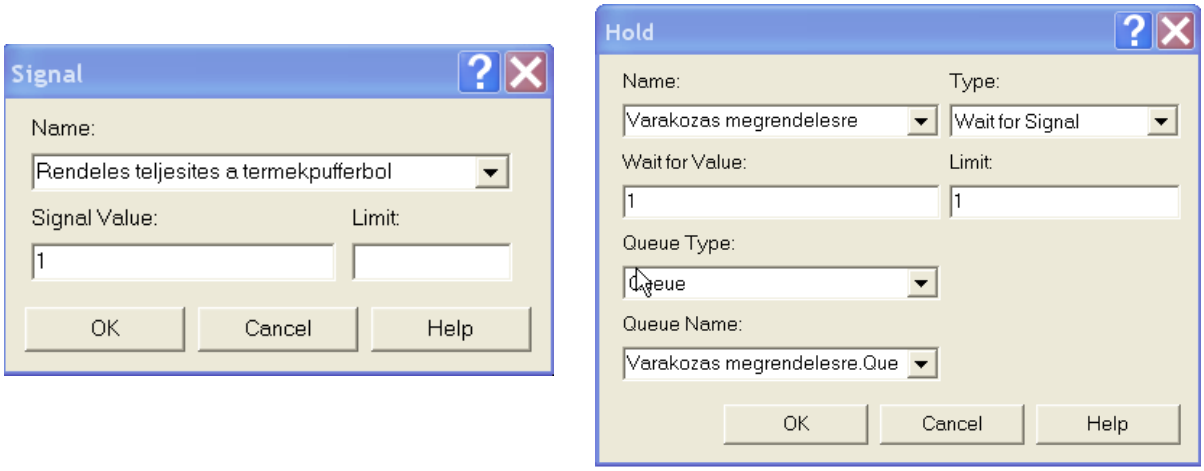
10.3. táblázat

A „*Termekpuffer kezdokeszlet*” nevű **Create** modul paraméterei

Name	Termekpuffer kezdokeszlet
Entity Type	Termek
Time Between Arrivals	
Type	Constant
Value	1
Units	Minutes
Entity per Arrival	Termekpuffer merete
Max Arrivals	1
First Creation	0

A *Kanban1* entitás a **Signal** modul után belép a „*Megrendelesi folyamat*” nevű **Process** modulba (*10.4. ábra*), amely a megrendelés előkészítést modellezi. A modul paraméteri a *10.4. táblázatban* olvashatók. A „*Rendeles elokeszitesi ido*” nevű kifejezés értéke 5 perc. A *Kan-*

ban1 entitást ezt követően a „*Ut a Gyartas_S allomasra1*” nevű **Route** modullal (*10.6. ábra*) a munkaállomást azonosító „*Gyartas_S*” nevű állomásra küldjük.

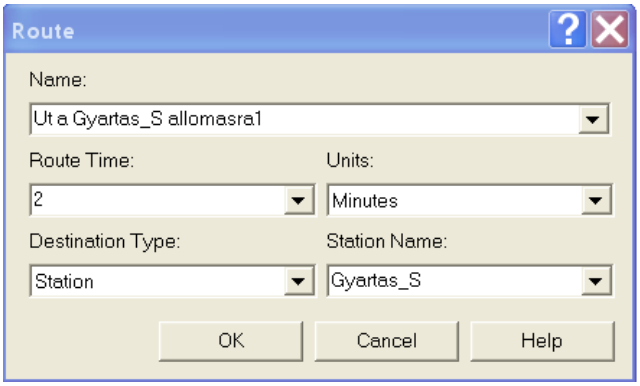


10.5. ábra: A „*Rendeles teljesites a termekpufferbol*” nevű **Signal**, és a „*Varakozas megrendelesre*” nevű **Hold** modul dialógusablakai

10.4. táblázat

A „*Megrendelesi folyamat*” nevű **Process** modul paraméterei

Name Action	Megrendelesi folyamat Seize Delay Release
Resource Type Resource Name Quantity	 Resource Megrendelo_R 1
Delay Type Units Expression	Expression Minutes Rendeles elokeszitesi ido



10.6. ábra: Az „*Ut a Gyartas_S allomasra1*” nevű **Route** modul párbeszédablaka

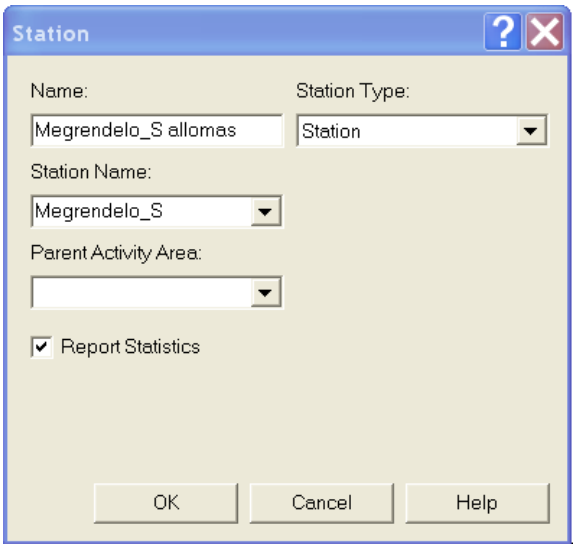
A *Termek* entitás a „*Termek szamlalas*” nevű **Assign** és a „*Atfutasi ido*” nevű **Record** modulokon áthaladva, a „*Termek kiszallitas*” nevű **Dispose** modulon keresztül elhagyja a rendszert (*10.4. ábra*). A „*Termek szamlalas*” nevű **Assign** modulban az áthaladó entitáshoz hozzárendeljük a *Temek* entitás attribútumait, továbbá megszámláljuk a teljesített megrendeléseket.

A szegmens, illetve az ügyfélszolgálat helyét, az **Assign** modult megelőző „*Megrendelo_S allomas*” nevű **Station** modul, azonosítja (*10.7. ábra*).

10.5. táblázat

A „*Termek szamlalas*” nevű **Assign** modul paraméterei

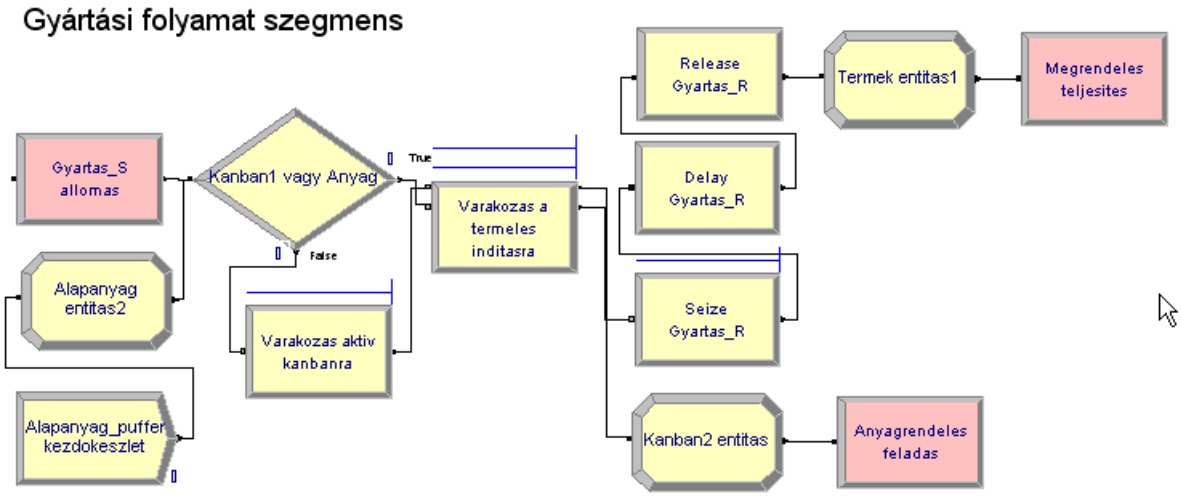
Name	Termek szamlalas
Type Entity Type	Entity Type Termek
Type Entity Picture	Entity Picture Picture.Blue Ball
Type Variable Name New Value	Variable Teljesitett megrendelesek szama Teljesitett megrendelesek szama + 1



10.7. ábra: Az „*Megrendelo_S allomas*” nevű **Station** modul párbeszédablaka

A gyártás szegmens

A gyártás szegmens (*10.8. ábra*) „*Gyartas_S*” nevű **Station** moduljába egyrészt a megrendelés szegmensből, a „*Megrendelo_S*” állomásról, másrészt a beszállítás szegmensből a „*Beszallito_S*” állomásról érkeznek *Kanban1* és *Anyag* entitások. A szegmens egy anyag pufferrel is rendelkezik, amelyet az „*Alapanyag_puffer kezdokeszlet*” nevű **Create** modullal (*10.6. táblázat*) töltünk fel az „*Alapanyag entitas2*” nevű **Assign** modul érintésével (*10.7. táblázat*).



10.8. ábra: Az egykártyás kanban rendszer gyártási folyamat szegmense

A „*Gyartas_S*” nevű állomásra érkező *Kanban1* és *Anyag* entitásokat a „*Kanban1 vagy Anyag*” nevű **Decide** modulban (*10.9. ábra*) válogatjuk szét. Így érjük el, hogy a **Decide** modult követő „*Varakozas a termeles inditasra*” nevű **Match** modulba a *Kanban1* és az *Anyag* entitások szeparáltan lépjenek be. Az *Anyag* entitások a **Decide** modul *True* ágán, a *Kanban1* entitások pedig a *False* ágán lépnek ki. A **Match** modulhoz két sor tartozik: a „*Varakozas a termeles inditasra.Queue1*” és a „*Varakozas a termeles inditasra.Queue2*”. Az elsőben a *Kanban1*, a másodikban az *Anyag* entitások várakoznak. Tulajdonképpen a második sor az anyag puffernek felel meg, az első sor pedig a kiszolgálásra várakozó megrendeléseket reprezentálja. A forgalomban lévő kanbanok számát úgy korlátozzuk, hogy a **Decide** modul *False* oldala és a **Match** modul közé egy „*Varakozas aktiv kanbanra*” nevű **Hold** modult (*10.11. ábra*) illesztünk. A **Hold** modul a *Kanban1* entitást csak akkor engedi tovább, ha a „*Varakozas a termeles inditasra.Queue1*” sorban a *Kanban1* entitások száma kisebb vagy egyenlő, mint az „*Anyagpuffer merete*” nevű változó értéke. Ez azt jelenti, hogy az aktív kanbanok száma megegyezik az anyag puffer méretével.

10.6. táblázat

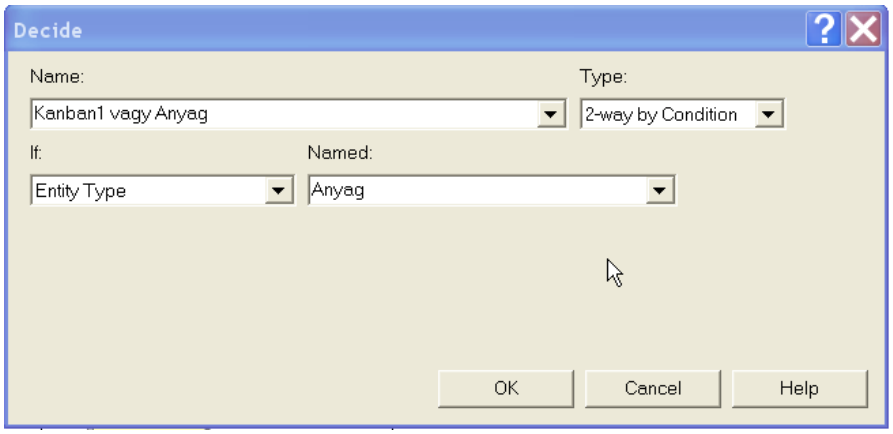
Az „*Alapanyag_puffer kezdokeszlet*” nevű **Create** modul paraméterei

Name	Alapanyag_puffer kezdokeszlet
Entity Type	Anyag
Time Between Arrivals	
Type	Constant
Value	1
Units	Minutes
Entity per Arrival	Anyagpuffer merete
Max Arrivals	1
First Creation	0

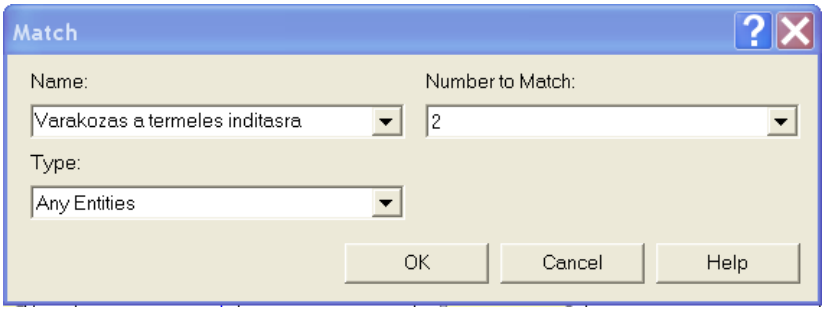
10.7. táblázat

Az „*Alapanyag entitas2*” nevű **Assign** modul paraméterei

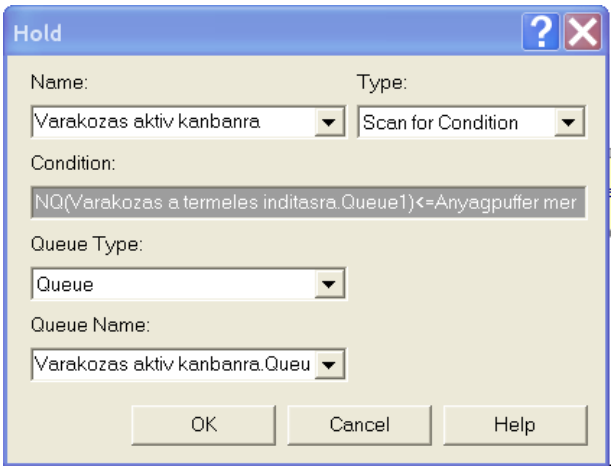
Name	Alapanyag entitas2
Type	Entity Picture
Entity Picture	Picture.Red Ball
Type	Variable
Variable Name	Anyag beszallitasok szama
New Value	Anyag beszallitasok szama+ 1



10.9. ábra: Az „*Kanban1 vagy Anyag*” nevű **Decide** modul párbeszédablaka



10.10. ábra: Az „*Varakozas a termeles inditasra*” nevű **Match** modul párbeszédablaka



10.11. ábra: Az „*Varakozas aktiv kanbanra*” nevű **Hold** modul párbeszédablaka

10.8. táblázat

A „*Seize Gyartas_R*” nevű **Seize** modul paraméterei

Name Allocation	Seize Gyartas_R Wait
Resource Type Resource Name Quantity	 Resource Gyartas_R 1
Queue Type Queue Name	Queue Seize Gyartas_R.Queue

10.9. táblázat

A „*Delay Gyartas_R*” nevű **Delay** modul paraméterei

Name Allocation Delay Time Units	Delay Gyartas_R Value Added Gyartasi ido Minutes

10.10. táblázat

A „*Release Gyartas_R*” nevű **Release** modul paraméterei

Name	Release Gyartas_R
Resource Type Resource Name Quantity	 Resource Gyartas_R 1

Visszatérve a „*Varakozas a termeles inditasra*” nevű **Match** modulhoz, abból egyidejűleg egy-egy egymáshoz rendelt *Kanban1* és *Anyag* entitás léphet ki. Az *Anyag* entitás belép a „*Seize Gyartas_R*” nevű **Seize** modulba (*10.8. táblázat*), ahol leköti a „*Gyartas_R*” nevű erőforrást, feltéve hogy az erőforrás szabad, különben várakozik a „*Seize Gyartas_R.Queue*” nevű sorban. Az erőforrás lekötése után az *Anyag* entitás belép a „*Delay Gyartas_R*” nevű **Delay** modulba (*10.9. táblázat*), és ott a „*Gyartasi ido*” értének megfelelő időt tölt. A gyártás befejezése után a továbbhaladó entitás a „*Release Gyartas_R*” nevű **Release** modulban (*10.10. táblázat*) felszabadítja a „*Gyartas_R*” erőforrást. A gyártás eredményként az anyag termékké válik, ezért a „*Termek entitas1*” nevű **Assign** modulban (*10.11. táblázat*) megváltoztatjuk az *Anyag* entitás tulajdonságait és növeljük a „*Gyartott termekek szama*” nevű változó értékét eggyel. A *Termek* entitást ezt követően a „*Megrendeles teljesites*” nevű **Route** modul segítségével a „*Megrendelo_S*” nevű állomásra küldjük.

10.11. táblázat

A „*Termek entitas1*” nevű **Assign** modul paraméterei

Name	Termek entitas1
Type Entity Type	Entity Type Termek
Type Entity Picture	Entity Picture Picture.Blue Ball
Type Attribute Name New Value	Attribute Entity Station Gyartas_S
Type Variable Name New Value	Variable Gyartott termekek szama Gyartott termekek szama + 1

10.12. táblázat

A „*Megrendeles teljesites*” nevű **Route** modul paraméterei

Name	Megrendeles teljesites
Route Time	5
Units	Minutes
Destination Type	Station
Station Name	Megrendelo_S

10.13. táblázat

A „*Kanban2 entitas*” nevű **Assign** modul paraméterei

Name	Kanban2 entitas
Type Entity Type	Entity Type Kanban2
Type Entity Picture	Entity Picture Picture.Red Page
Type Attribute Name New Value	Attribute Entity Station Gyartas_S

A **Match** modulból az *Anyag* entitással együtt kilépő *Kanban1* entitás attribútumait a „*Kanban2 entitas*” nevű **Assign** modulban (*10.13. táblázat*) változtatjuk meg. Az entitás típusa *Kanban2*, képe *Picture.Red Page,* az Entity.Station attribútuma pedig „*Gyartas_S*” lesz. Az új tulajdonságokkal felruházott *Kanban2* entitást az „*Anyagrendeles feladas*” nevű *Route* modul-

lal (*10.14. táblázat*) a beszállító szegmensbe, pontosabban a „*Beszallito_S*” állomásra irányítjuk.

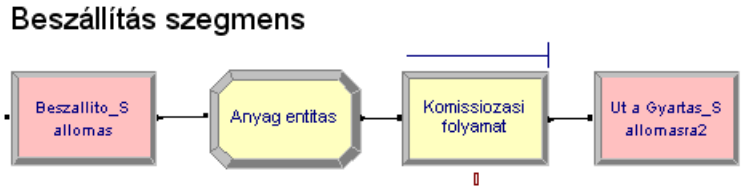
10.14. táblázat

Az „*Anyagrendeles feladas*” nevű **Route** modul paraméterei

Name	Anyagrendeles feladas
Route Time	2
Units	Minutes
Destination Type	Station
Station Name	Beszallito_S

A beszállítás szegmens

A beszállítás szegmens (*10.12. ábra*) „*Beszallito_S*” állomására belépő *Kanban2* entitások tulajdonságait az „*Anyag entitas*” **Assign** modulban (*10.15. táblázat*) változtatjuk meg. A tovább haladó *Anyag* entitás belép a „*Komissiozasi folyamat*” nevű **Process** modulba (*10.16. táblázat*), ahol leköti a „*Beszallito_R*” erőforrást, és végrehajtja a komissiózást. Az „*Ut a Gyartas_S allomasra2*” nevű **Route** modul *10.17. táblázat*) az *Anyag* entitást a „*Gyartas_S*” állomásra küldi.



10.12. ábra: Az egykártyás kanban rendszer beszállítás szegmense

10.15. táblázat

Az „*Anyag entitas*” nevű **Assign** modul paraméterei

Name	Anyag entitas
Type Entity Type	Entity Type Anyag
Type Entity Picture	Entity Picture Picture.Red Ball
Type Variable New Value	Variable Anyag beszallitasok szama Anyag beszallitasok szama + 1

10.16. táblázat

A „*Megrendelesi folyamat*” nevű **Process** modul paraméterei

Name Action	*Komissiozasi folyamat* Seize Delay Release
Resource Type Resource Name Quantity	 Resource Beszallito_R 1
Delay Type Units Expression	Expression Minutes Komissiozasi ido

10.17. táblázat

Az „*Ut a Gyartas_S allomasra2*” nevű **Route** modul paraméterei

Name	Ut a Gyartas_S allomasra2
Route Time	2
Units	Minutes
Destination Type	Station
Station Name	Gyartas_S

10.2.3. Változók, kifejezések és statisztikák

A modell változóit a *10.13. ábra* foglalja össze. Az „*Anyagpuffer merete*” és a „*Termekpuffer merete*” inputváltozók, amelyeknek az értékét a szimuláció futása előtt kell a **Variable** adatmodulban megadni. A „*Megrendelesek szama*”, a”*Teljesitett megrendelesek szama*”, a „*Gyartott termekek szama*” és az „*Anyag beszallitasok szama*” outputváltozók, amelyek kezdeti értéke 0, és amelyek a szimuláció végén a *User Specified* statisztikában jelennek meg.

Variable - Basic Process

	Name	Rows	Columns	Clear Option	Initial Values	Report Statistics
1	Anyagpuffer merete			System	1 rows	☐
2	Termekpuffer merete			System	1 rows	☐
3	Megrendelesek szama			System	0 rows	☐
4	Teljesitett megrendelesek szama			System	0 rows	☐
5	Gyartott termekek szama			System	0 rows	☐
6	Anyag beszallitasok szama			System	0 rows	☐

Double-click here to add a new row.

10.13. ábra: A **Variable** adatmodul párbeszédablaka táblázatnézetben

Expression - Advanced Process

	Name	Rows	Columns	Expression Values
1	Komissiozasi ido			1 rows
2	Rendeles elokeszitesi ido			1 rows
3	Gyartasi ido			1 rows

Double-click here to add a new row.

10.14. ábra: Az **Expression** adatmodul párbeszédablaka táblázatnézetben

	Name	Type	Expression	Report Label	Frequency	Resource Name	Report Label
1	Megrendelo_R allapota	Frequency		Megrendelo_R allapota	State	Megrendelo_R	Megrendelo_R allapota
2	Gyartas_R allapota	Frequency		Gyartas_R allapota	State	Gyartas_R	Gyartas_R allapota
3	Beszallito_R allapota	Frequency		Beszallito_R allapota	State	Beszallito_R	Beszallito_R allapota
4	Az alapanyag_puffer atlagos keszlete	Output	DAVG(Varakozas a termeles inditasra.Queue2.NumberInQueue)	Az alapanyag_puffer atlagos keszlete	Value		Az alapanyag_puffer atlagos keszlete
5	A varakozo Kanbanok atlagos szama	Output	DAVG(Varakozas a termeles inditasra.Queue1.NumberInQueue)	A varakozo Kanbanok atlagos szama	Value		A varakozo Kanbanok atlagos szama
6	Megrendelesek szama osszesen	Output	Megrendelesek szama	Megrendelesek szama osszesen	Value		Megrendelesek szama osszesen
7	Gyartott termek szama osszesen	Output	Gyartott termekek szama	Gyartott termek szama osszesen	Value		Gyartott termek szama osszesen
8	Teljesitett megrendelesek szama osszesen	Output	Teljesitett megrendelesek szama	Teljesitett megrendelesek szama	Value		Teljesitett megrendelesek szama
9	Anyag beszallitasok szama osszesen	Output	Anyag beszallitasok szama	Anyag beszallitasok szama osszesen	Value		Anyag beszallitasok szama osszesen
10	A varakozo megrendelesek atlagos szama	Output	DAVG(Megrendelesi folyamat.Queue.NumberInQueue)	A varakozo megrendelesek atlagos szama	Value		A varakozo megrendelesek atlagos szama

10.15. ábra: A **Statistics** adatmodul párbeszédablaka táblázatnézetben

A modellben, pontosabban a **Process** és **Delay** modulokban használt kifejezéseket „*Komissiozási ido*”, „*Rendeles elokeszitesi ido*” „*Gyartasi ido*” (5, 1 és 10 perc) az **Expression** adatmodulban definiálhatjuk (*10.14. ábra*).

A statisztikai adatmodul (*10.15. ábra*) első három sorában a modell erőforrásairól készül *Frequency* típusú statisztika, amelyek az erőforrások állapotát (*Busy, Idle, Inactive*) mutatják. A 4-5 sorokban az anyag és a termék puffer átlagos készletét számítjuk ki. A 6-9 sorok statisztikái az összes megrendelések, teljesített megrendelések, gyártott termék és anyag beszállítások számáról adnak tájékoztatást. Az utolsó sorban az időben változó várakozó megrendelések számát átlagoljuk.

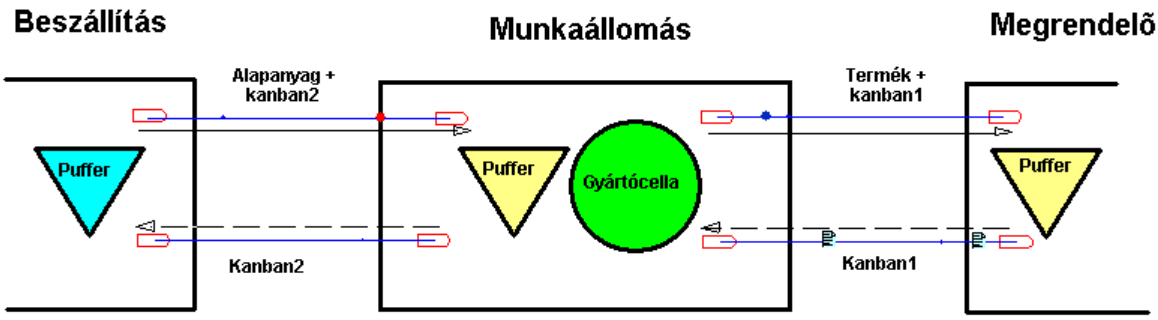
10.2.4. Az egykártyás kanban rendszer animációja

Az animációhoz viszonylag egyszerű, az előző fejezetekben megismert eszközöket használunk (*10.17. ábra*). A rendszerben mozgó entitások („*Kanban1*”, „*Kanban2*”, „*Anyag*” és „*Termek*”) megkülönböztetéséhez különböző alakú és színű entitásképeket (*Picture.Blue Page, Picture.Red Page, Picture.Report, Picture.Blue Ball*) használunk. Ezeket már korábban, az entitások létrehozásakor definiáltuk. Az entitásokhoz rendelt képek az **Entiy** adatmodulban (*10.16. ábra*) tekinthetők meg. A *10.17. ábra* háttérgrafikáját (háromszögek, körök, téglalapok, vonalak) az Arena rajzeszközeivel rajzolhatjuk meg. Az entitások mozgásának megjelenítéséhez az *Animate Transfer* eszköztár *Station* eszközével ( ) az animációban elhelyezzük a korábban definiált állomásokat („*Beszallito_S allomas*”, „*Gyartas_S allomas*” és „*Megrendelo_S allomas*”) a megfelelő helyekre, majd a *Route* eszköz ( ) segítségével, a *10.17. ábrát* követve összekötjük azokat. Az animáció végső eredménye a *10.17. ábrán* látható, amelyen jól követhető a kanban kártyák, az anyagok és a termék mozgása.

Entity - Basic Process

	Entity Type	Initial Picture
1	Kanban1	Picture.Blue Page
2	Termek	Picture.Blue Ball
3	Kanban2	Picture.Red Page
4	Anyag	Picture.Report

10.16. ábra: Az **Entity** adatmodul dialógustáblája



10.17. ábra: Az egykártyás kanban rendszer animációja

10.2.5. A szimuláció kimenetei

Az egykártyás kanban rendszer modelljének 100 órás futásának a végén, a *Riport panelen* elérhető jelentések közül a sorokról (*Queues*) készített jelentést az *10.18. ábra* mutatja be.

A jelentésben minden sorhoz *Waiting Time* (várakozási idő) és *Number Waiting* (várakozó entitások száma, azaz a sor hossza) statisztika tartozik. A „*Seize Gyartas_R*” nevű sorban átlagosan 7,88 termékegység várakozik, és a sorban átlagos várakozási idő 77,79 perc. A meg-

előző „*Varakozas a termeles inditasra*” nevű **Match** modulhoz tartozó „*Varakozas a termeles inditasra.Queue1*” és a „*Varakozas a termeles inditasra.Queue2*” sorokban a várakozási idő kisebb 3,55 illetve 7,87 perc. Az előbbi sorban a „*Kanban1*”, az utóbbiban pedig az „*Anyag*” entitások várakoznak. A számokból kiolvasható, hogy az anyagellátás intenzitása kicsit nagyobb, mint a megrendelések érkezési intenzitása, vagyis inkább az „*Anyag*” entitások várakoznak a „*Kanban1*” entitásokra, mint fordítva. Ugyanakkor a szűkkeresztmetszetet a „*Gyartas_R*” erőforrás jelenti, amelynek az elérésére várakozó entitások átlagos várakozási ideje sokkal nagyobb, 77,79 perc.

6:08:20 **Queues** november 6, 2011

Egykártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Queue Detail Summary

Time

	Waiting Time
Komissiozasi folyamat.Queue	0.33
Megrendelesi folyamat.Queue	0.06
Seize Gyartas_R.Queue	77.79
Varakozas a termeles inditasra.Queue1	3.55
Varakozas a termeles inditasra.Queue2	7.87
Varakozas aktiv kanbanra.Queue	6.18
Varakozas megrendelesre.Queue	5.69

Other

	Number Waiting
Komissiozasi folyamat.Queue	0.03
Megrendelesi folyamat.Queue	0.01
Seize Gyartas_R.Queue	7.88
Varakozas a termeles inditasra.Queue1	0.35
Varakozas a termeles inditasra.Queue2	0.78
Varakozas aktiv kanbanra.Queue	0.02
Varakozas megrendelesre.Queue	0.00

10.18. ábra: Jelentés a sorokról

6:14:46 **Resources** november 6, 2011

Egykártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Resource Detail Summary

Usage

	Inst Util	Num Busy	Num Sched	Num Seized	Sched Util
Beszallito_R	0,49	0,49	1,00	592,00	0,49
Gyartas_R	0,95	0,95	1,00	568,00	0,95
Megrendelo_R	0,10	0,10	1,00	592,00	0,10

10.19. ábra: Jelentés az erőforrásokról

A **Riport** panel erőforrásokról (*Resources*) szóló jelentését a az *10.19. ábra* jeleníti meg. Az *Inst Util* oszlopban látható, hogy az erőforrások közül a „*Gyartas_R*” kihasználtsága 95%-os. Ugyanakkor a „*Beszallito_R*” és a „*Megrendelo_R*” csak az idő 49, illetve 10%-ban foglalt. Az erőforrásokról szóló riport oszlopai további statisztikákat is tartalmaznak, amelynek az elemzését az olvasóra bízzuk.

6:19:55 **Frequencies** november 06, 2011

Egykártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Beszallito_R allapota	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	409	7.2372	49.33	49,33
IDLE	410	7.4146	50.67	50,67

Gyartas_R allapota	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	31	182.98	94.54	94,54
IDLE	31	10.5702	5.46	5,46

Megrendelo_R allapota	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	535	1.1065	9.87	9,87
IDLE	535	10.1084	90.13	90,13

10.20. ábra: Jelentés az erőforrások állapotvalószínűségeiről

Az előző jelentéshez hasonló megállapításokra juthatunk a *10.20. ábrán* megjelenített *Frequencies* statisztikából. A „*Gyartas_R*” erőforrás az idő 94,64 %-ban, „*Beszallito_R*” erőforrás az idő 49,33 %-ban és „*Megrendelo_R*” erőforrás az idő 9,87 %-ban a foglalt (*Busy*).

6:31:29 **User Specified** november 6, 2011

Egykártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Tally

Interval	Average	Half Width	Minimum	Maximum
Atfutasi ido	118.76	(Correlated)	3.4615	322.06

Output

Output	Value
A varakozo Kanbanok atlagos szama	0.3503
A varakozo megrendelesek atlagos szama	0.00558280
Anyag beszallitasok szama osszesen	594.00
Az alapanyag_puffer atlagos keszlete	0.7838
Gyartott termek szama osszesen	567.00
Megrendelesek szama osszesen	592.00
Teljesitett megrendelesek szama osszesen	568.00

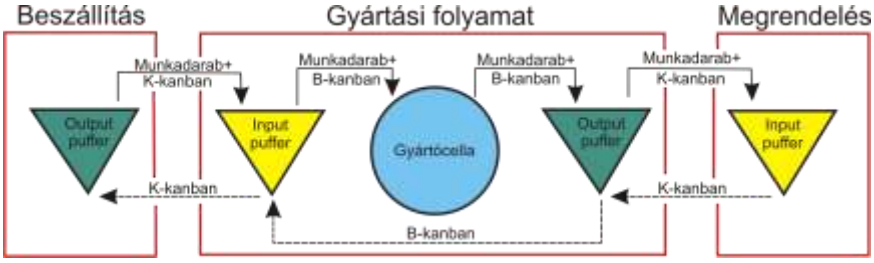
10.21. ábra: Jelentés a felhasználó által definiált statisztikákról

A felhasználó által definiált statisztikákat összefoglaló jelentés az *10.21. ábrán* tanulmányozható. Ezeket a statisztikákat a **Statistic** és a **Record** modulokban definiáltuk. Az ábrán a *Tally* szakaszban található az „*Atfutasi ido*” nevű statisztika. Az átfutási idő becsült középértéke 118,76 perc, a minimum 3,46 a maximum 322,06 perc.

Az *Output* szakasz a rendszer a várakozó kanbanok átlagos számáról (0,35 db), a várakozó megrendelések átlagos számáról (0,0056 db), az alapanyag puffer átlagos készletéről, az összes megrendelések számáról (592 db), az összes anyag beszállítások számáról (594 db), az összes gyártott termék számáról (567 db) és a teljesített megrendelések számáról (568 db) közöl becsült információt.

10.3. Az egylépcsős, kétkártyás kanban rendszer modellje

Ebben a pontban egy egytermékes, egylépcsős és kétkártyás kanban rendszer modellezését mutatjuk be, amely a *10.2 pontban* ismertetett problémához hasonlóan három szegmensre osztható fel (*10.22. ábra*).



10.22. ábra: Kétkártyás kanban rendszer

10.3.1. A probléma ismertetése

A probléma annyiban különbözik a *10.2.1. pontban* leírt gyártási rendszertől, hogy a gyártási folyamat input és output pufferei fizikailag elkülönülnek a raktár output és az ügyfélszolgálat input pufferétől. Az ügyfélszolgálat és a gyártás, valamint a raktár és a gyártás közötti anyag- és információáramlást a *K-kanbannak* nevezett külső kanban kártyák generálják (*10.22. ábra*). A gyártási folyamaton belül pedig a *B-kanbannnak* nevezett belső kanban kártyák működnek.

A kanban rendszer által gyártott termékeket igénylő ügyfelek itt is véletlenszerűen, exponenciális eloszlás szerint érkeznek, 10 perces átlagos érkezési időközzel, és egy konténer mennyiségű terméket igényelnek. A rendelés előkészítés időtartama 5 perc. Az ügyfélszolgálat az ügyfeleket a saját input pufferéből szolgálja ki, és ennek a puffernek a nagysága 2 konténer. Az ügyféligény kielégítésével egyidejűleg aktívvá váló *K-kanbant* a munkaállomás output pufferébe küldjük, ahol egyrészt anyagáramot generálunk az ügyfélszolgálat input pufferének feltöltése érdekében, másrészt az ezzel egyidejűleg aktívvá váló *B-kanbant* a munkaállomás input pufferébe továbbítjuk, azaz kezdeményezzük a gyártást.

A munkaállomáson a gyártás indításának egyik feltétele az aktív *B-kanban* kártya jelenléte, a másik feltétele pedig, hogy a gyártáshoz szükséges alapanyagok rendelkezésre álljanak. A munkaállomáson az alapanyagokat az input pufferben tároljuk, amelynek a kezdőkészlete 2 konténer (*10.22. ábra*). Így a munkaállomáson az aktív *B-kanban* kártya érkezésekor a gyártás azonnal megkezdődhet, feltéve, hogy az input puffer nem üres. Ha a termelés feltételei adottak, akkor megkezdődik egy konténer mennyiségű termékek gyártása, és ezzel egy időben gondoskodunk az input puffer feltöltéséről, azaz, az aktív *K-kanban* kártyákat a raktár output pufferébe küldjük. A munkaállomáson a termeléshez szükséges gyártócellák mennyisége egy, és egy konténernyi termék előállításának műveleti ideje 10 perc.

A raktár output pufferébe érkező aktív *K-kanbanok* kezdeményezik az alapanyag beszállítást. Az alapanyag ellátást biztosító raktárban a komissiózási idő 5 perc. A raktár kapacitását vég-

telen nagynak tekintjük, ami azt jelenti, hogy a folyamat működését veszélyeztető anyaghiány soha nem jelentkezhet. A szállítási idő a raktár és a munkaállomás, illetve a munkaállomás és az ügyfélszolgálat között egyaránt 5 perc. A *K-kanban* és a *B-kanban* kártyák továbbításának időtartama minden relációban 2 perc. A szimulációs idő hossza legyen 100 óra.

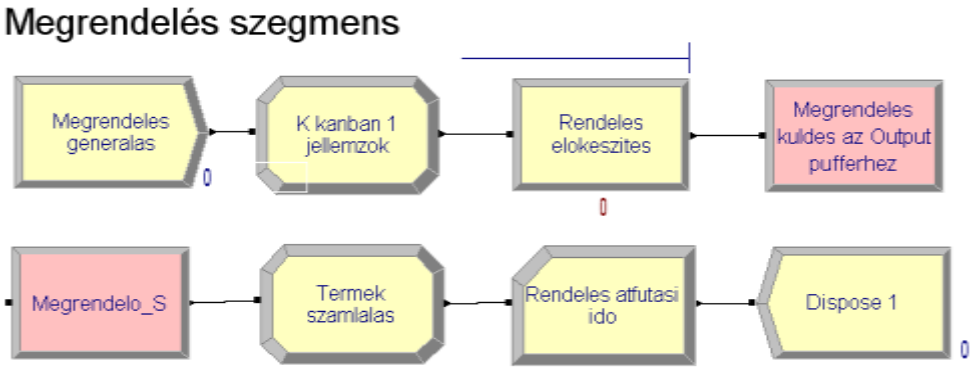
A modellezés eredményeként szeretnénk megismerni:

A folyamatok állapotainak valószínűségeit (idle, busy).
A külső és a belső kanbanok aktivitását.
A megrendelések, a gyártott termékek és az anyagbeszállítások számát.
A teljesített megrendelések számát.
A nem teljesített megrendelések számát.
A gyártósor kapacitását db/óra.
Az egy termékre eső gyártási időt óra/db.
Az átlagos rendelésteljesítési időt min per db.

10.3.2. A probléma Arena modellje (10.2. modell)

A megrendelés szegmens

A megrendelés szegmens (*10.23. ábra*) „*Megrendeles generalas*” nevű **Create** moduljában létrehozzuk a „*K kanban 1*” nevű entitást. A modul további paramétereit a *10.18. táblázat* foglalja össze. A „*K kanban 1 jellemzok*” nevű **Assign** modulban az entitás tulajdonságait, az entitás képét, az állomás nevét adjuk meg. A modulban ezenkívül a „*Rendeles erkezes*” nevű attribútumhoz rendeljük a megrendelés érkezésének az időpontját, és a „*Megrendelesek szama”* nevű változóban számláljuk a beérkezett megrendeléseket (*10.19. táblázat*).



10.23. ábra: A kétkártyás kanban rendszer megrendelés szegmense

10.18. táblázat

A „*Megrendeles generalas*” nevű **Create** modul paraméterei

Name	Megrendeles generalas
Entity Type	K kanban 1
Time Between Arrivals	
Type	Random(Expo)
Value	10
Units	Minutes
Entity per Arrival	1
Max Arrivals	Infinite
First Creation	0

A „*Rendeles elokeszites*” nevű **Process** modul a beérkező megrendelések előkészítési műveletit szimulálja (*10.23. ábra*). A modul paramétereit a *10.20. táblázat* foglalja össze. Az Action mezőben a *Seize Delay Release* opciót választjuk, és a folyamat erőforrásának neve

„*Megrendelo_R*”. Előkészítés után a „*K kanban 1*” nevű entitást a „*Megrendeles kuldes az Output pufferhez*” nevű **Route** modullal a gyártási folyamat szegmenshez, pontosabban az „*Output puffer*” nevű állomására küldjük (*10.24. ábra*).

10.19. táblázat

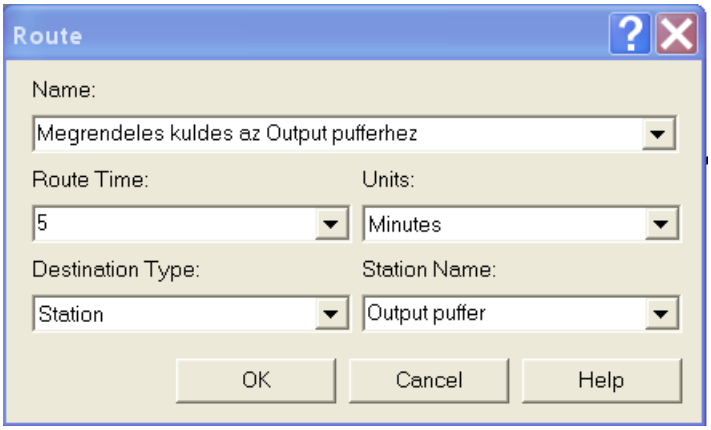
A „*K kanban 1 jellemzok*” nevű **Assign** modul paraméterei

Name	K kanban 1 jellemzok
Type Entity Picture	Entity Picture Picture.Blue Page
Type Attribute Name New Value	Attribute Entity Station Megrendelo_S
Type Attribute Name New Value	Attribute Rendeles erkezes TNOW
Type Variable Name New Value	Variable Megrendelesek szama Megrendelesek szama + 1

10.20. táblázat

A „*Rendeles elokeszites*” nevű **Process** modul paraméterei

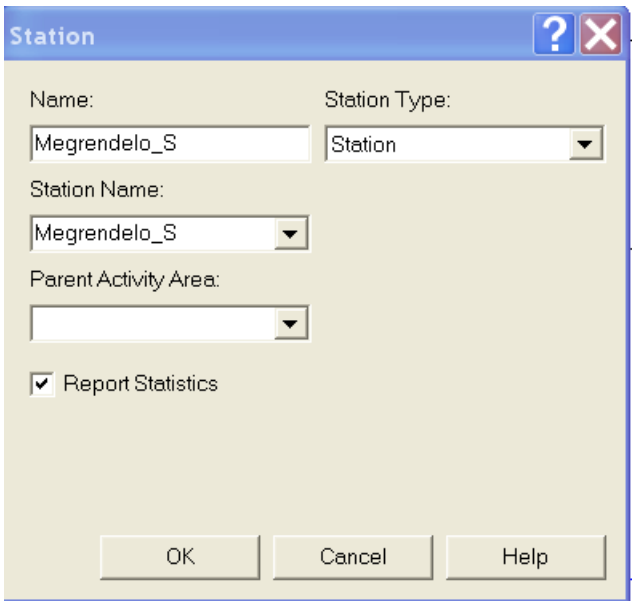
Name Action	Rendeles elokeszites Seize Delay Release
Resource Type Resource Name Quantity	 Resource Megrendelo_R 1
Delay Type Units Expression	Expression Minutes Rendeles elokeszitesi ido



10.24. ábra: A „*Megrendeles kuldes az Output pufferhez*” nevű **Route** modul párbeszédablaka

A megrendelés szegmens helyét a „*Megrendelo_S*” nevű **Station** modul azonosítja (*10.25. ábra*). A gyártási folyamatot követően a „*Megrendeles teljesites*” nevű **Route** modulból (*10.26. ábra*) ide érkeznek „*Termek*” entitások, amelyek a „*Termek szamlalas*” nevű **Assign** (*10.21. táblázat*) és az „*Atfutasi ido*” nevű **Record** (*10.26. ábra*) modulokon áthaladva a

„*Dispose 1*” nevű **Dispose** modulon keresztül távoznak a rendszerből. A „*Termek szamlalas*” nevű **Assign** modulban a „*Rendelesteljesitesi ido*” nevű változóhoz rendeljük a rendelés érkezése és a termék távozása között eltelt időt. Az „*Atfutasi ido*” nevű **Record** modulal a rendelés érkezése és a termék távozása közötti átlagos átfutási időt határozzuk meg.

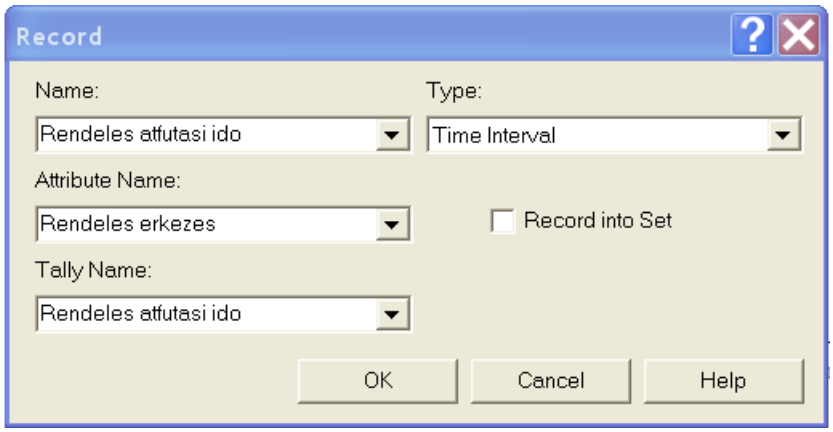


10.25. ábra: Az *Megrendelo_S*” nevű **Station** modul párbeszédablaka

10.21. táblázat

A „*Termek szamlalas*” nevű **Assign** modul paraméterei

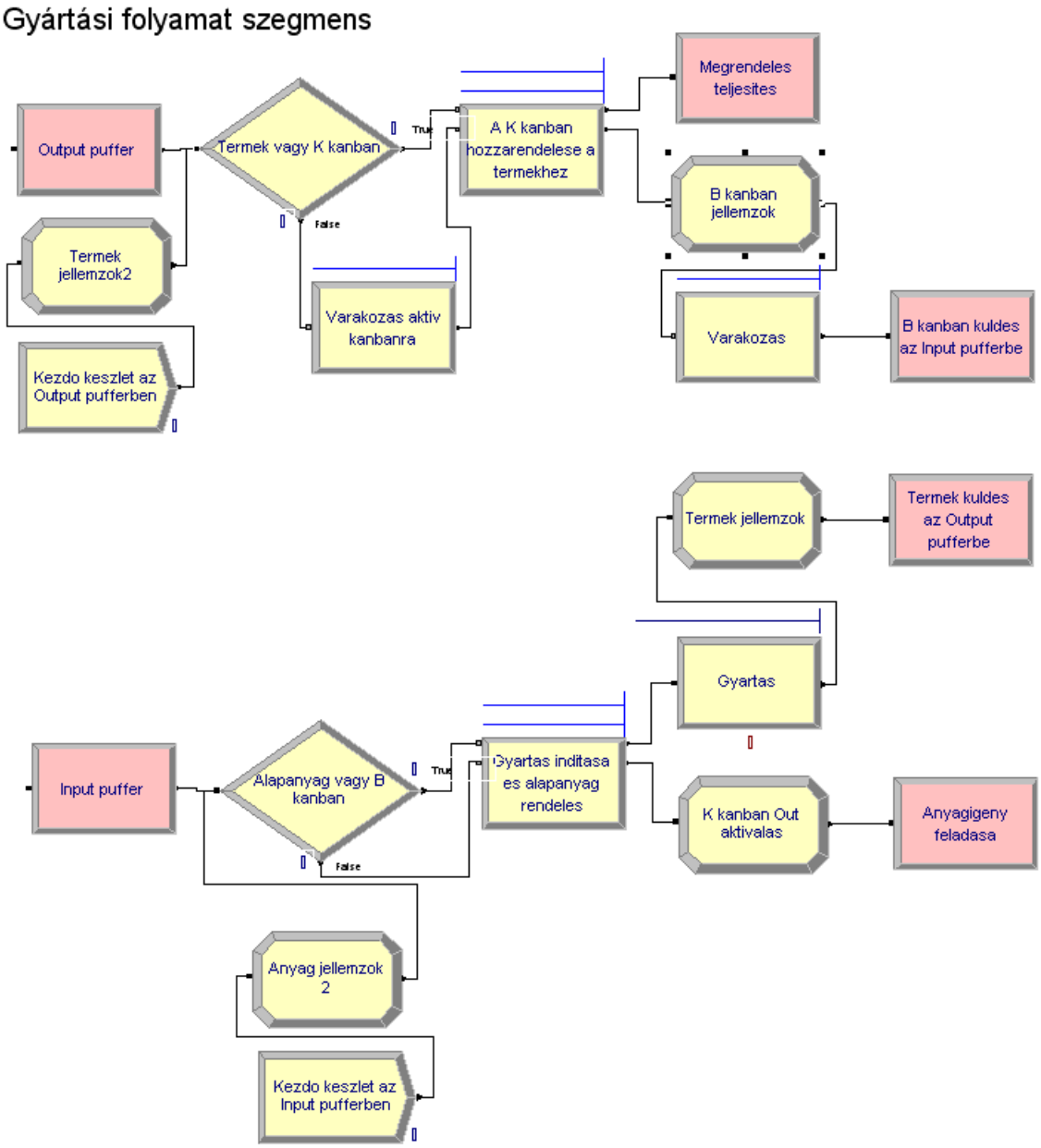
Name	Termek szamlalas
Type Entity Type	Entity Type Termek
Type Variable Name New Value	Variable Rendelesteljesitesi ido TNOW - Rendeles erkezes
Type Variable Name New Value	Variable Teljesitett rendelesek szama Teljesitett rendelesek szama +1



10.26. ábra: A „*Rendeles atfutasi ido*” nevű **Record** modul párbeszédablaka

A gyártás szegmens

A gyártás szegmens (*10.27. ábra*) „*Output puffer*” nevű **Station** moduljába (*10.22. táblázat*) a megrendelés szegmensből megrendelések „*Kanban 1*” entitások, illetve a „*Termek kuldes az Output pufferbe*” nevű **Route** modulból a szegmensben előállított termékek, „*Termek*” entitások érkeznek. A modulba érkező entitásokat az entitástípus alapján a „*Termek vagy K kanban*” nevű **Decide** (*10.28. ábra*) modulban választjuk szét. A *True* ágon kilépő „*Termek*” entitások közvetlen, a *False* ágon kilépő „*K Kanban 1*” entitások pedig a „*Varakozas aktiv kanbanra*” nevű **Hold** modulon keresztül (*10.23. táblázat*) lépnek be a „*A K kanban hozzarendelese a termekhez*” nevű **Match** modulba. A **Match** modulban a hozzárendelés eredményeként a „*K Kanban 1*” entitások inaktívvá válnak.



10.27. ábra: A kétkártyás kanban rendszer gyártási folyamat szegmense

10.22. táblázat

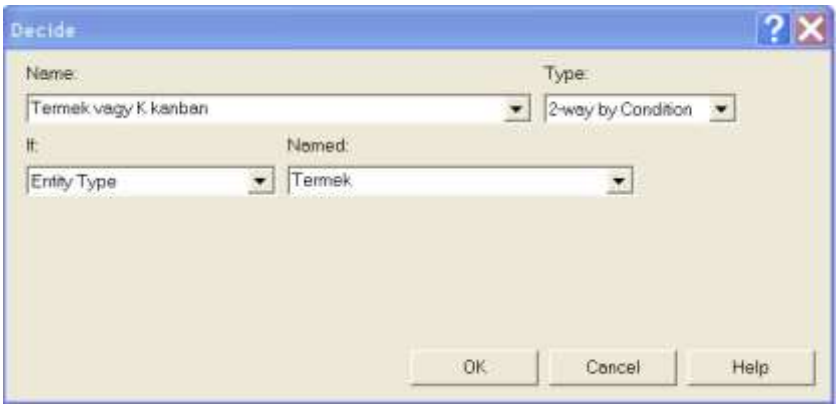
Az „*Output puffer*” nevű **Station** modul paraméterei

Name	Output puffer
Station Type	Station
Station Name	Output puffer

A **Hold** modulban az entitások továbbhaladását a

NQ(*A K kanban hozzarendelese a termekhez.Queue2*) <= *Termekpuffer meret*

feltételhez kötjük, ami azt jelenti, hogy megrendelés teljesítést a pufferből csak akkor kezdeményezzük, ha a termék pufferben az előírt, a „*Termekpuffer meret*”-ben meghatározott számú termékkel egyenlő vagy annál kevesebb található. A feltétel teljesüléséig a „*K kanban 1*” entitások a **Hold** modul „*Varakozas aktiv kanbanra.Queue*” nevű sorában várakoznak. Tulajdonképpen a **Hold** modul szabályozza a termék puffer méretét.



10.28. ábra: A „*Termek vagy K kanban*” nevű **Decide** modul párbeszédablaka

10.23. táblázat

A „*Varakozas aktiv kanbanra*” nevű **Hold** modul paraméterei

Name	Varakozas aktiv kanbanra
Type Condition	Scan for Condition NQ(A K kanban hozzarendelese a termekhez.Queue2) <= Termekpuffer meret
Queue Type Queue Name	Queue Varakozas aktiv kanbanra.Queue

10.24. táblázat

Az „*Megrendeles teljesites*” nevű **Route** modul paraméterei

Name	Megrendeles teljesites
Route Time Units	5 Minutes
Destination Type Station Name	Station Megrendelo_S

A **Match** modulból az entitások párosával (egy termék és egy kanban) akkor haladhatnak tovább, ha a két entitástípus egyidejűleg jelen van. A **Match** modulhoz két sor tartozik, a „*K kanban hozzarendelese a termekhez.Oueue1*” és a „*K kanban hozzarendelese a termekhez.Oueue2*”. Az elsőben a *True* ágon érkező termékek, a másodikban pedig a *False* ágon érkező K kanbanok várakoznak. A „*Termek*” entitás a „*Megrendeles teljesites*” nevű **Route**

(*10.24. táblázat*) modulon keresztül a megrendelés szegmens „*Megrendelo_S*” nevű **Station** (*10.25. ábra*) moduljába távozik.

A „*B kanban jellemzok*” nevű **Assign** modulba érkező „*Kanban 1*” entitások tulajdonságai a *10.25. táblázat* szerint változnak meg. Az entitás típusa belső kanbanra, azaz „*B kanban*”-ra, az entitás képe „*Picture.Red Page*”-re és az entitás állomása “*Output puffer*”-re változik. A statisztikák gyűjtéséhez a kanbanok aktivitását a 0 és 1 értékű változókkal jellemezzük. A 0 érték az inaktív, az 1 érték az aktív állapotot jelzi. A „*B kanbanok szama*” nevű változó az aktívvá vált „*B kanban*”-okat számlálja.

10.25. táblázat

A „*A B kanban jellemzok*” nevű **Assign** modul paraméterei

Name	A B kanban jellemzok
Type Entity Type	Entity Type B kanban
Type Variable Name New Value	Variable K kanban 1 aktivitas 0
Type Attribute Name New Value	Attribute Entity.Picture Picture.Red Page
Type Attribute Name New Value	Attribute Entity.Station Output puffer
Type Variable Name New Value	Variable B kanban aktivitas 1
Type Variable Name New Value	Variable B kanbanok szama B kanbanok szama + 1

10.26. táblázat

A „*Varakozas*” nevű **Hold** modul paraméterei

Name	Varakozas
Type Condition	Scan for Condition STATE (Gyartocella)== −1
Queue Type Queue Name	Queue Varakozas.Queue

10.27. táblázat

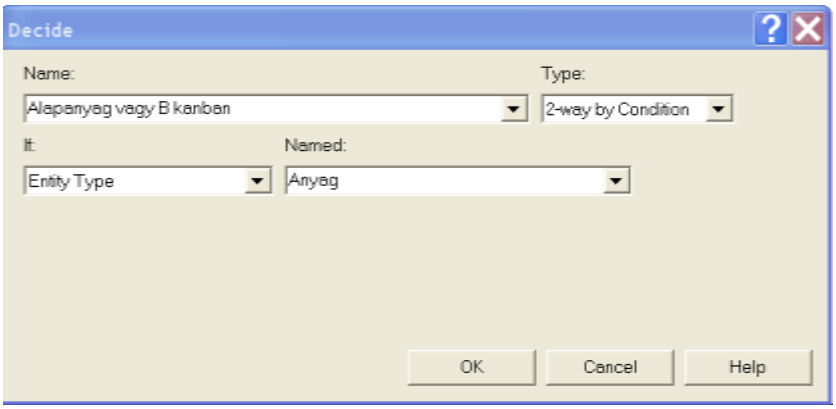
A „*B kanban kuldes az Input pufferbe*” nevű **Route** modul paraméterei

Name	B kanban kuldes az Input pufferbe
Route Time Units	5 Minutes
Destination Type Station Name	Station Input puffer

A hozzárendeléseket követően az új „*B kanban*” entitások a „*Varakozás*” nevű **Hold** modulban (*10.26. táblázat*) a gyártás „*Gyartocella*” nevű erőforrásának a felszabadítására várakoznak. A *STATE* (*Gyartocella*) függvény −1 értéke azt jelenti, hogy az erőforrás szabad (*Idle*)

állapotban van, vagyis, amikor a függvény értéke −1, akkor engedélyezzük az entitás továbbhaladását.

Az entitások ezután a „*B kanban kuldes az Input pufferbe*” nevű **Route** modul közvetítésével az „*Input puffer*” nevű **Station** modulba (*10.27. táblázat*) kerülnek. A beszállítás szegmensből a gyártás alapanyagai, „*Anyag*” nevű entitások is ugyanide érkeznek. Az input puffer kezdőkészletét a szimuláció indításakor egy alkalommal a „*Kezdo keszlet az Input pufferben*”nevű **Create** modullal töltjük fel „*Anyag*” entitásokkal. A **Create** modulban az Entities Per Arrival mezőbe az „*Anyagpuffer meret*” nevű változót írjuk be, amelynek az értéke a feladatleírás szerint 2. Az „*Anyag jellemzok 2*” nevű **Assign** modulban az entitásokhoz rendeljük az attribútumaikat: Entity.Picture == „*Picture.Red Ball*”, Entity.Station ==”*Input puffer*”. Ezután az entitásokat az „*Alapanyag vagy B kanban*” nevű **Decide** modulban (*10.29. ábra*) az entitás típus alapján válogatjuk szét. A *True* ágon távozó „*Anyag*” entitások és a *False* ágon kilépő „*B Kanban*” entitások belépnek a „*Gyartas inditasa es alapanyag rendeles*” nevű **Match** modulba, ahonnan a modul működési szabályai szerint párosan (egy „*Anyag*” és egy „*B Kanban*”) távoznak.



10.29. ábra: Az „*Alapanyag vagy B kanban*” nevű **Decide** modul párbeszédablaka

10.28. táblázat

A „*Gyartas*” nevű **Process** modul paraméterei

Name Action	Gyartas Seize Delay Release
Resource Type Resource Name Quantity	 Resource Gyartocella 1
Delay Type Units Expression	Expression Minutes Gyartasi ido

10.29. táblázat

A „*K kanban Out aktivalas*” nevű **Assign** modul paraméterei

Name	K kanban Out aktivalas
Type Entity Type	Entity Type K kanban 2
Type Variable Name New Value	Variable K kanban 2 aktivitas 1
Type	Attribute

Attribute Name New Value	Entity.Picture Picture.Red Page
Type Attribute Name New Value	Attribute Entity.Station Beszallito_S
Type Variable Name New Value	Variable Anyagrendelesek szama Anyagrendelesek szama + 1

A **Match** modul kilépő oldalán a „*B Kanban*” entitások a „*Gyartas*” nevű **Process** modul (*10.28. táblázat*), a „*Anyag*” entitások a „*K kanban Out aktivalas*” nevű **Assing** modul (*10.29. táblázat*) felé haladnak. Az új „*K kanban 2*” entitásokat az „*Anyagigeny feladasa*” nevű **Route** modullal (*10.30. táblázat*) a beszállítás szegmens „*Beszallito_S*” nevű **Station** modul-jába irányítjuk.

10.30. táblázat

Az „*Anyagigeny feladasa*”nevű **Route** modul paraméterei

Name	Anyagigeny feladasa
Route Time Units	5 Minutes
Destination Type Station Name	Attribute Entity.Station

A „*Gyartas*” nevű **Process** modulban realizáljuk a gyártást, amelynek a műveleti idejét a „*Gyartasi ido*” nevű kifejezés tartalmazza, és aminek az értéke 10 perc. A gyártás után az entitásokat a „*Termek jellemzok*” nevű **Assing** modulban új tulajdonságokkal ruházzuk fel (*10.31. táblázat*). Végül a „*Termek*” entitásokat a „*Termek kuldes az Output pufferbe*” nevű **Route** modullal (*10.32. táblázat*) az „*Output puffer*” nevű állomásra küldjük, ahonnan a ko-rábban leírtak szerint a megrendelő szegmensbe jutnak.

10.31. táblázat

A „*Termek jellemzok*” nevű **Assign** modul paraméterei

Name	Termek jellemzok
Type Entity Type	Entity Type Termek
Type Variable Name New Value	Variable B kanban aktivitas 0
Type Attribute Name New Value	Attribute Entity.Picture Picture.Blue Ball
Type Attribute Name New Value	Attribute Entity.Station Output puffer
Type Variable Name New Value	Variable Gyartott termekek szama Gyartott termekek szama + 1

10.32. táblázat

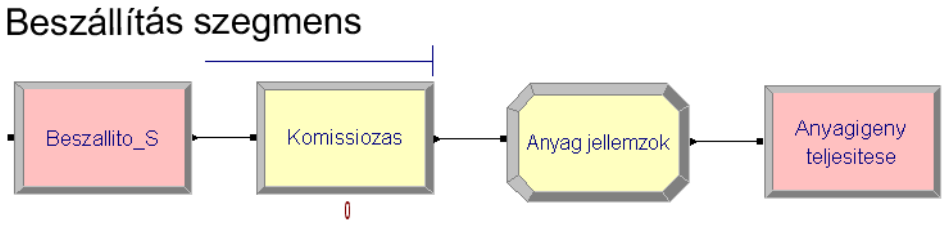
A „*Termek kuldes az Output pufferbe*”nevű **Route** modul paraméterei

Name	Termek kuldes az Output pufferbe
Route Time Units	5 Minutes
Destination Type Station Name	Attribute Entity.Station

Megjegyezzük az „*Anyagigeny feladasa*” és a „*Termek kuldes az Output pufferbe*”nevű **Route** modulokban a Destination Type és Station Name mezők értékei, a korábbiaktól eltérően, *Attribute*, illetve *Entity.Station*. Az *Entity.Station* attributumokat („*Beszallito_S*”, illetve „*Output puffer*”) a **Route** modulok előtti **Assing** modulokban rendeltük az entitásokhoz (*10.29.* és *10.32. táblázatok*).

A beszállítás szegmens

A szegmens logikáját a *10.30. ábra* szemlélteti. A „*Beszallito_S*” állomásra az anyagigények, a „*K kanban 2*” entitások, az „*Anyagigeny feladasa*”nevű **Route**-ból érkeznek, és azonnal belépnek a „*Komissiozas*” nevű **Process** modulhoz tartozó sorba. A modul paraméterit a *10.33. táblázat* foglalja össze. A művelet erőforrása a „*Beszallito_R*” nevű resource. A komissiózás műveleti idejét pedig a „*Komissiozasi ido*” nevű kifejezéssel adjuk meg, amelynek a kezdeti értéke 5 perc.



10.30. ábra: A kétkártyás kanban rendszer beszállítás szegmense

10.33. táblázat

A „*Gyartas*” nevű **Process** modul paraméterei

Name Action	Komissiozas Seize Delay Release
Resource Type Resource Name Quantity	 Resource Beszallito_R 1
Delay Type Units Expression	Expression Minutes Komissiozasi ido

A komissiózást követően az entitások attribútumait (entitás típusa, képe, állomása) a „*Anyag jellemzok*” nevű **Assign** modulban (*10.34. táblázat*) változtatjuk meg. A modulban megszüntetjük a „*K kanban 2*” aktivitását, és növeljük az „*Anyagbeszallitasok szama*” nevű változó értékét.

10.34. táblázat

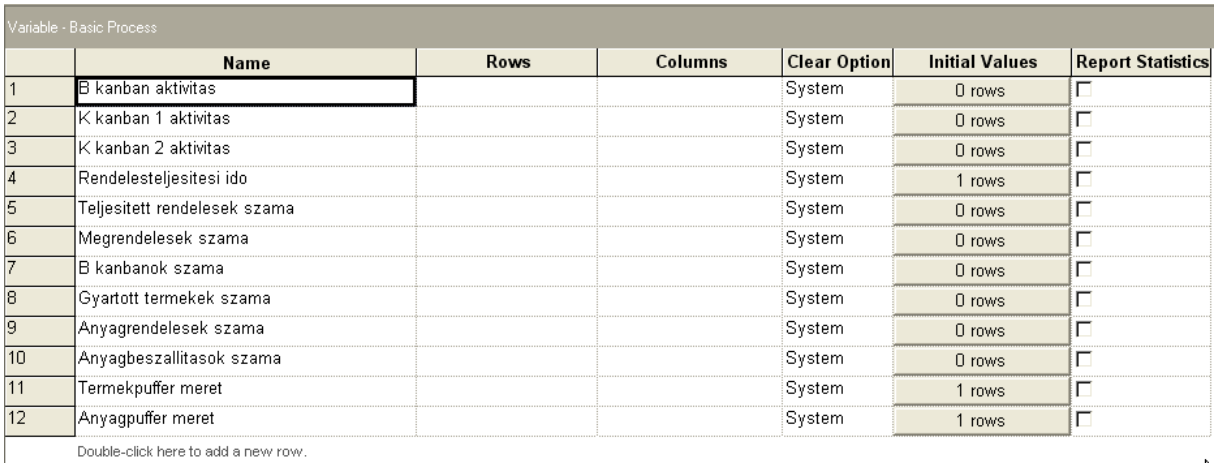
A „*Anyag jellemzok*” nevű **Assign** modul paraméterei

Name	Anyag jellemzok
Type Entity Type	Entity Type Anyag
Type Variable Name New Value	Variable K kanban 2 aktivitas 0
Type Attribute Name New Value	Attribute Entity.Picture Picture.Red Ball
Type Attribute Name New Value	Attribute Entity.Station Input puffer
Type Variable Name New Value	Variable Anyagbeszallitasok szama Anyagbeszallitasok szama + 1

Végül az „*Anyag*” entitásokat a „*Anyagigeny teljesitese*” nevű **Route** modullal a gyártás szegmens „*Input puffer*” nevű állomására küldjük.

10.3.3. Változók, kifejezések és statisztikák

A modell táblázatba foglalt változóit a *10.31. ábrán* láthatjuk. A modell inputváltozói az „*Anyagpuffer merete*” és a „*Termekpuffer merete*”. Ezek értékét a szimuláció futása előtt a **Variable** adatmodulban kell megadni. A „*Megrendelesek szama*”, a”*Teljesitett megrendelesek szama*”, a „*Gyartott termekek szama*”, az „*Anyagrendelesek szama*”, az „*Anyagbeszallitasok szama*”, a „*B kanbanok szama*” és a „*Rendelesteljesitesi ido*” outputváltozók, amelyek kezdeti értéke 0, és amelyek a szimuláció végén a *User Specified* statisztikában jelennek meg. A 0 és 1 értékű „*B kanban aktivitas*”, „*K kanban 1 aktivitas*” és „*K kanban 2 aktivitas*” nevű változók a kanbanok aktivitását mutató *Time Persistent* típusú statisztika készítéséhez szükségesek.


Variable - Basic Process

	Name	Rows	Columns	Clear Option	Initial Values	Report Statistics
1	B kanban aktivitas			System	0 rows	☐
2	K kanban 1 aktivitas			System	0 rows	☐
3	K kanban 2 aktivitas			System	0 rows	☐
4	Rendelesteljesitesi ido			System	1 rows	☐
5	Teljesitett rendelesek szama			System	0 rows	☐
6	Megrendelesek szama			System	0 rows	☐
7	B kanbanok szama			System	0 rows	☐
8	Gyartott termekek szama			System	0 rows	☐
9	Anyagrendelesek szama			System	0 rows	☐
10	Anyagbeszallitasok szama			System	0 rows	☐
11	Termekpuffer meret			System	1 rows	☐
12	Anyagpuffer meret			System	1 rows	☐

Double-click here to add a new row.

10.31. ábra: A **Variable** adatmodul párbeszédablaka táblázatnézetben

Expression - Advanced Process

	Name	Rows	Columns	Expression Values
1	Gyartasi ido			1 rows
2	Komissiozasi ido			1 rows
3	Rendeles elokeszitesi ido			1 rows
	Double-click here to add a new row.			

10.32. ábra: Az **Expression** adatmodul párbeszédablaka táblázatnézetben

A **Process** modulokban használt, a műveleti időket tároló kifejezéseket „*Komissiozási ido*”, „*Rendeles elokeszitesi ido*” „*Gyartasi ido*” (10, 5 és 10 perc) az **Expression** adatmodulban definiáljuk (*10.32. ábra*).

A statisztikai adatmodul (*10.33. ábra*) első három sorában találhatók a belső és a külső kanbanok aktivitását mutató *TimePersistent* típusú statisztika. A 4-10 és 12 sorok statisztikái az összes megrendelések, a teljesített megrendelések, a gyártott termék, az anyagrendelések és az anyag beszállítások számáról adnak tájékoztatást. Az 11 sorban az átlagos rendeléstelje-sítési időről, ami a rendelés érkezése és kiszolgálása között eltelik, gyűjtünk információt. Végül az adatmodul 13-15 soraiban a modell erőforrásairól készül *Frequency* típusú statisztika, amelyek az erőforrások állapotát (*Busy, Idle, Inactive*) mutatják.

Statistic - Advanced Process

	Name	Type	Expression	Report Label	Frequency Type	Resource Name	Report Label
1	K kanban 1 aktivitasa	Time-Persistent	K kanban 1 aktivitas	K kanban 1 aktivitasa	Value		
2	K kanban 2 aktivitasa	Time-Persistent	K kanban 2 aktivitas	K kanban 2 aktivitasa	Value		
3	B kanban aktivitasa	Time-Persistent	B kanban aktivitas	B kanban aktivitasa	Value		
4	Megrendelesek szama osszesen	Output	Megrendelesek szama	Megrendelesek szama osszesen	Value		
5	Teljesitett megrendelesek szama osszesen	Output	Teljesitett rendelesek szama	Teljesitett megrendelesek szama osszesen	Value		
6	Gyartott termekek szama osszesen	Output	Gyartott termekek szama	Gyartott termekek szama osszesen	Value		
7	Anyagrendelesek szama osszesen	Output	Anyagrendelesek szama	Anyagrendelesek szama osszesen	Value		
8	Anyagbeszallitasok szama osszesen	Output	Anyagbeszallitasok szama	Anyagbeszallitasok szama osszesen	Value		
9	Az egy termekre eso gyartasi ido ora per db	Output	TFIN/60/Teljesitett rendelesek szama	Az egy termekre eso gyartasi ido ora per db	Value		
10	A gyartosor kapacitasa db per ora	Output	Teljesitett rendelesek szama * 60 /TFIN	A gyartosor kapacitasa db per ora	Value		
11	Atlagos rendeles teljesitesi ido min per db	Time-Persistent	Rendelesteljesitesi ido	Atlagos rendeles teljesitesi ido min per db	Value		
12	Nem teljesitett megrendelesek szama	Output	Megrendelesek szama - Teljesitett rendelesek szama	Nem teljesitett megrendelesek szama	Value		
13	Megrendelo_R allapota	Frequency		Megrendelo_R allapota	State	Megrendelo_R	Megrendelo_R allapota
14	Gyartocella allapota	Frequency		Gyartocella allapota	State	Gyartocella	Gyartocella allapota
15	Beszallito_R allapota	Frequency		Beszallito_R allapota	State	Beszallito_R	Beszallito_R allapota

10.33. ábra: Az **Statistics** adatmodul párbeszédablaka táblázatnézetben

10.3.4. A kétkártyás kanban rendszer animációja

Az animáció készítésekor az egykártyás kanban rendszernél megismert módszert követjük (*10.35. ábra*). A rendszerben mozgó entitások („*Termek*” „*B kanban*”, „*K kanban 1*”, „*K kanban 2*”, és „*Anyag*”) megkülönböztetéséhez különböző alakú és színű entitásképeket (*Picture.Blue Ball, Picture.Red Page, Picture.Blue Page, Picture.Red Ball*) használunk. Ezeket már korábban, az entitások létrehozásakor definiáltuk. Az entitásokhoz rendelt képek az **Entiy** adatmodulban (*10.34. ábra*) tekinthetők meg.

A *10.35. ábra* háttérgrafikáját az Arena rajzeszközeivel rajzolhatjuk meg. Az entitások mozgásának megjelenítéséhez pedig az *Animate Transfer* eszköztár *Station* eszközével ( ) a megfelelő helyekre helyezzük a modellben definiált állomásokat („*Beszallito_S allomas*”, „*Output puffer*”, „*Input puffer*” és „*Megrendelo_S allomas*”), majd a *Route* eszköz ( ) segítségével összekötjük azokat. Az animáció egy pillanatfelvétele a *10.35. ábrán* látható, amelyen

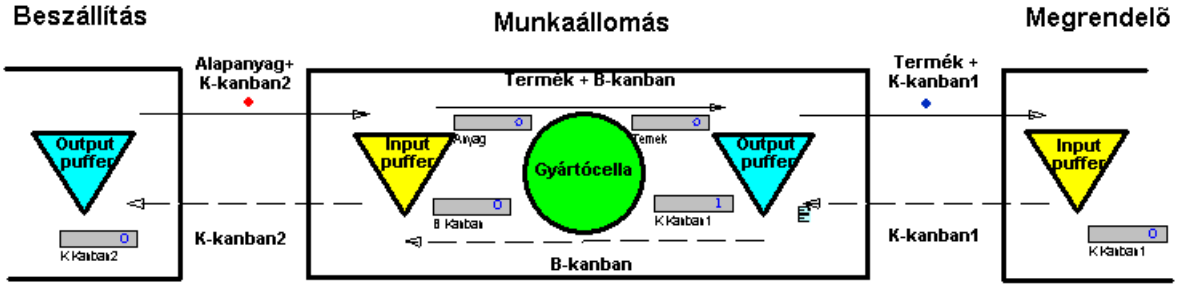
a kanban kártyák, az anyagok és a termékek mozgása mellet a pufferek készletváltozása is követhető.

Entity - Basic Process

	Entity Type	Initial Picture
1	Termek	Picture.Blue Ball
2	B kanban	Picture.Red Page
3	K kanban 1	Picture.Blue Page
4	K kanban 2	Picture.Red Page
5	Anyag	Picture.Red Ball

Double-click here to add a new row.

10.34. ábra: Az **Entity** adatmodul dialógustáblája



10.35. ábra: A kétkártyás kanban rendszer animációja

10.3.5. A szimuláció kimenetei

A kétkártyás kanban rendszer modelljének 100 órás futásának a végén, a *Riport panelen* elérhető jelentések közül a sorokról (*Queues*) készített jelentést az *10.36. ábra* mutatja be. A jelentésben minden sorhoz *Waiting Time* (várakozási idő) és *Number Waiting* (várakozó entitások száma, azaz a sor hossza) statisztika tartozik. A megelőző „*A K kanban hozzarendelese a termekhez*” nevű **Match** modulhoz tartozó „*A K kanban hozzarendelese a termekhez.Queue1*” és a „*A K kanban hozzarendelese a termekhez.Queue2*” sorokban a várakozási idő 0,17 illetve 36,54 perc. Az előbbi sorban a „*Termek*”, az utóbbiban pedig a „*K Kanban 1*” entitások várakoznak. A számokból kiolvasható, hogy a megrendelések érkezési intenzitása nagyobb, mint a gyártás intenzitása, vagyis inkább a „*K Kanban 1*” entitások várakoznak a „*Termek*” entitásokra, mint fordítva. A „*Varakozas aktiv kanbanra.Queue*” sorban az átlagos várakozási idő 389,41 perc. Ez azt is jelenti, hogy a szűkkeresztmetszetet az anyagellátás és a gyártás jelenti. Az eredmények magyarázata, hogy a rendelések exponenciális eloszlás szerint érkeznek 10 perces átlagos érkezési időközzel, ugyanakkor a gyártást megelőző komissiózás és a gyártás műveleti ideje egyaránt 10 perc, amihez még mozgatási idők is adódnak.

A **Riport** panel erőforrásokról (*Resources*) szóló jelentését a az *10.37. ábra* jeleníti meg. Az *Inst Util* oszlopban látható, hogy az erőforrások közül a „*Gyartocella*” kihasználtsága 80%-os. Ugyanakkor a „*Beszallito_R*” és a „*Megrendelo_R*” csak az idő 79, illetve 49%-ban foglalt. Az erőforrásokról szóló riport további oszlopaiban a statisztikák az igénybevett erőforrások számát (Num Sched), az igénybevételek számát (Num Seized) tartalmazzák.

Az előző erőforrásokról szóló jelentéshez hasonló megállapításokra juthatunk a *10.38. ábrán* megjelenített *Frequencies* statisztikákból. A „*Gyartocella*” erőforrás az idő 79,55 %-ban, „*Beszallito_R*” erőforrás az idő 79,46 %-ban és „*Megrendelo_R*” erőforrás az idő 49,33 %-ban a foglalt (*Busy*).

16:50:23 **Queues** február 27, 2012

Kétkártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Queue Detail Summary

Time

	Waiting Time
A K kanban hozzarendelese a termekhez.Queue1	0.17
A K kanban hozzarendelese a termekhez.Queue2	36.54
Gyartas inditasa es alapanyag rendeles.Queue1	2.68
Gyartas inditasa es alapanyag rendeles.Queue2	0.00
Gyartas.Queue	2.48
Komissiozas.Queue	2.47
Rendeles elokeszites.Queue	2.53
Varakozas aktiv kanbanra.Queue	389.41
Varakozas.Queue	5.00

Other

	Number Waiting
A K kanban hozzarendelese a termekhez.Queue1	0.01
A K kanban hozzarendelese a termekhez.Queue2	2.92
Gyartas inditasa es alapanyag rendeles.Queue1	0.21
Gyartas inditasa es alapanyag rendeles.Queue2	0.00
Gyartas.Queue	0.20
Komissiozas.Queue	0.20
Rendeles elokeszites.Queue	0.25
Varakozas aktiv kanbanra.Queue	39.51
Varakozas.Queue	0.20

10.36. ábra: Jelentés a sorokról

16:50:58 **Resources** február 27, 2012

Kétkártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Resource Detail Summary

Usage

	Inst Util	Num Busy	Num Sched	Num Seized	Sched Util
Beszallito_R	0,79	0,79	1,00	477,00	0,79
Gyartocella	0,80	0,80	1,00	478,00	0,80
Megrendelo_R	0,49	0,49	1,00	592,00	0,49

10.37. ábra: Jelentés az erőforrásokról

16:51:29 **Frequencies** február 27, 2012

Kétkártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Beszallito_R allapota	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	241	19.7834	79.46	79,46
IDLE	241	5.1129	20.54	20,54

Gyartocella allapota	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	241	19.8042	79.55	79,55
IDLE	241	5.0921	20.45	20,45

Megrendelo_R allapota	Number Obs	Average Time	Standard Percent	Restricted Percent
BUSY	296	10.0000	49.33	49,33
IDLE	296	10.2703	50.67	50,67

10.38. ábra: Jelentés az erőforrások állapotvalószínűségeiről

18:12:57 **User Specified** február 27, 2012

Kétkártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Tally

Interval	Average	Half Width	Minimum	Maximum
Rendeles atfutasi ido	459.02	(Correlated)	15.0000	1,141.14

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Atlagos rendeles teljesitesi ido min per db	456.18	(Correlated)	5.0000	1,141.14
B kanban aktivitasa	0.5988	0,003805324	0	1.0000
K kanban 1 aktivitasa	0.4245	0,022154582	0	1.0000
K kanban 2 aktivitasa	0.4946	0,018463778	0	1.0000

Output

Output	Value
A gyartosor kapacitasa db per ora	4.7800
Anyagbeszallitasok szama osszesen	476.00
Anyagrendelesek szama osszesen	478.00
Az egy termekre eso gyartasi ido ora per db	0.2092
Gyartott termekek szama osszesen	477.00
Megrendelesek szama osszesen	592.00
Nem teljesitett megrendelesek szama	114.00
Teljesitett megrendelesek szama osszesen	478.00

10.39. ábra: Jelentés a felhasználó által definiált statisztikákról

A felhasználó által definiált statisztikákat összefoglaló jelentés az *10.39. ábrán* tanulmányozható. Ezeket a statisztikákat a **Statistic** és a **Record** modulokban definiáltuk. Az ábrán a *Tally* szakaszban található a „*Rendeles atfutasi ido*” nevű statisztika. Az átfutási idő becsült középértéke 459,02 perc, a minimum 15 a maximum 1141,14 perc. Ez meglehetősen nagy érték, ami a már említett anyagellátási és gyártási folyamatok kicsi kapacitásának köszönhető.

18:38:46 **User Specified** február 27, 2012

Kétkártyás kanban rendszer Replications: 1

Replication 1 Start Time: 0,00 Stop Time: 6 000,00 Time Units: Minutes

Tally

Interval	Average	Half Width	Minimum	Maximum
Rendeles atfutasi ido	138.45	(Correlated)	15.0000	339.06

Time Persistent

Time Persistent	Average	Half Width	Minimum	Maximum
Atlagos rendeles teljesitesi ido min per db	133.58	(Correlated)	5.0000	339.06
B kanban aktivitasa	0.6145	0,026563577	0	1.0000
K kanban 1 aktivitasa	0.4174	0,031357746	0	1.0000
K kanban 2 aktivitasa	0.6067	0,020722549	0	1.0000

Output

Output	Value
A gyartosor kapacitasa db per ora	5.6600
Anyagbeszallitasok szama osszesen	565.00
Anyagrendelesek szama osszesen	566.00
Az egy termekre eso gyartasi ido ora per db	0.1767
Gyartott termekek szama osszesen	566.00
Megrendelesek szama osszesen	592.00
Nem teljesitett megrendelesek szama	26.0000
Teljesitett megrendelesek szama osszesen	566.00

10.40. ábra: Jelentés a felhasználó által definiált statisztikákról a növelt kapacitású modellben

A *Time Persistent* szakasz a kanbanok (*B kanban*, *K kanban 1* és *K kanban 2*) aktivitásáról ad tájékoztatást. (Amint az ismert, egy kanban akkor aktív, amikor elszakad a terméktől, vagy az anyagtól.) A statisztikák szerint a *B kanban* az idő 59,88 %-ban, a *K kanban 1* az idő 42,45 %-ban és a *K kanban 2* az idő 49,46 %-ban aktív. Az „*Atlagos rendeles teljesitesi ido min per db*” azonos a „*Rendeles atfutasi ido*” nevű *Tally* statisztikával.

Az *Output* szakaszból megismerhetjük a gyártósor kapacitását, ami 4,78 db/óra, és az egy termékre eső gyártási időt: 0,21 óra. Tájékozódhatunk az összes megrendelések számáról (592 db), az összes anyag beszállítások számáról (476 db), az összes gyártott termék számáról (477 db) és a teljesített megrendelések számáról (478 db). A megrendelések száma és teljesített megrendelések száma közötti különbség (114 db) is magyarázatot ad a hosszú átfutási időre.

Az rendelés átfutási idő csökkentése érdekében növeljük a „*Gyartocella*” és a „*Beszallito_R*” erőforrások kapacitását 1-ről 2-re. Ennek eredményét a *10.41 ábrán* láthatjuk. A kapacitásnövelésnek köszönhetően a rendelés átfutási idő 459 percről 138 percre csökken. Ez azonban azzal jár, hogy mind a „*Gyartocella*”, mind a „*Beszallito_R*” erőforrások kihasználási mutatói egyaránt 47%-ra romlanak.

Ajánlott irodalom

1. **Altiok, T.- Melamed B.:** Simulation Modeling and Analysis with Arena. Academic Press is an imprint of Elsevier, ISBN 13: 978-0-12-370523-5, 2007.

2. Arena Standard Edition User's Guide. Rockwell Software.

3. Arena Professional Edition Reference Guide. Rockwell Software.

4. **Banks, J. and J. S. Carson**: Discrete-Event System Simulation, Prentice-Hall, 1984. 2. Conway, R. W., B. M. Johnson, and W. L. Maxwell, "Some Problems of Digital Systems Simulation," Management Science, Vol. 6, 1959, pp. 92-110.

5**. Barnes, R. M.:** Motion and Time Study: Design and Measurement of Work, Sixth Edition, John Wiley, 1968.

6. **Bratley, P.-Fox B. L.-Schrage L. E.:** A Guide to Simulation, 2d ed., Springer-Verlag, New York, NY., 1987.

7. **Benkő J.:** Logisztikai tervezés. (mezőgazdasági alkalmazásokkal) Dinasztia Kiadó, Budapest, 2000.

8. **Benkő J.:** A termény betakarítás és szállítás modellezése az Arena szimulátorral. Logisztikai évkönyv 2006. (Szerk.: Szegedi Z.), Magyar Logisztikai Egyesület, Budapest, 2006. 125-133 p.

9. **Benkő J.:** Logisztika I. (A felsőfokú logisztikai tanfolyam tankönyve.), LOKA, Gödöllő, 2009.

10. **Benkő J**.: Anyagmozgatási műveletek modellezése szimulációval Arena környezetben. Logisztikai évkönyv 2010. (Szerk: Szegedi Z.), MLE, Budapest 2010. 99-105 p., ISSN 1218-3849

11. **Benkő J**.: Ellátási láncok modellezése szimulációval. Logisztikai évkönyv 2011. (Szerk: Bokor Z.), MLE, Budapest 2011. 13-18 p., ISSN 1218-3849

12. **Benkő J.:** Kanban-rendszerű gyártás modellezése szimulációval. Magyar Minőség, XIX. évfolyam, 3. szám, 2010. március, ISSN 1789-5502 (CD-ROM), 16-31 p.

13. **Benkő J.:** Modeling Supply Chain with Simulation. I. Central European Conference on Logistics, University of Miskolc 26. november 2010.

14. **Benkő J:** Kanban-szabályozású gyártási rendszerek modellezése szimulációval. Logisztikai évkönyv 2012. (Szerk: Bokor Z.), MLE, Budapest 2012. 13-18 p., ISSN 1218-3849

15. **Devroye, L.:** Non-Uniform Random Variate Generation, Springer-Verlag, New York, NY., 1986.

16. **Devore, J. L.:** Probability and Statistics for Engineering and the Sciences, 6th ed., Wadsworth Inc, Belmont, CA., 2003.

17. **Diananda, P. H.:** "Some Probability Limit Theorems with Statistical Applications." Proceedings, Cambridge Phil. Soc., Vol. 49, 1953, pp. 239-246.

18. **Emshoff, J. R. and R. L. Sisson**: Design and Use of Computer Simulation Models Macmillan, 1970.

19. **Feller, W.:** An Introduction to Probability Theory and Its Applications, John Wiley, Vol. II, 1972.

20. **Fishman, G. S.:** Principles of Discrete Event Simulation, John Wiley, 1978.

21. **Giffin, W.**: Transform Techniques for Probability Modeling, Academic Press, 1975.

22. **Hogg, R. V. and A. T. Craig:** Introduction to Mathematical Statistics, Macmillan, 1970.

23. **Kelton, W. D.-Sadowski, R. P.-Sturrock, D. T.:** Simulation with Arena. Mc Graw Hill Higher Education, International Edition, ISBN 0-07-121933-1, 2004.

24. **Law, A. M.- Kelton, W. D.:** Simulation Modeling and Analysis, 3d ed., McGraw-Hill, New York, NY., 2000.

25. **Meszéna Gy.-Ziermann M.:** Valószínűségelmélet és matematikai statisztika. Közgazdasági és Jogi Könyvkiadó, Budapest, 1981.

26. **Mihram, G. A.:** Simulation: Statistical Foundations and Methodology, Academic Press, 1972.

27. **Papoulis, A.:** Probability, Random Variables, and Stochastic Processes, McGraw-Hill, 1965.

28. **Prezenszki J.** (szerk): Logisztika. (Bevezető fejezetek.) BME, Mérnöktovábbképző Intézet, Budapest, 1995.

29. **Prezenszki J.**(szerk): Logisztika II. (Módszerek, eljárások.), LFK, Budapest, 1999.

30. **Pritsker, A. A. B.:** The GASP IV Simulation Language, John Wiley, 1974.

31. **Pritsker, A. A. B.:** Modeling and Analysis Using Q-GERT Networks, Halsted Press and Pritsker & Associates, Inc., 1977.

32. **Pritsker, A. A. B.:** Introduction to Simulation and SLAM II. A Halsted Press Book, John Wiley & Sons, New York, 1986.

33. **Pritsker, A. A. B.:** "Compilation of Definitions of Simulation," SIMULATION, Vol. 33, 1979, pp. 61-63.

34. **Pritsker, A. A. B.:** "Models Yield Keys to Productivity Problems, Solutions," Industrial Engineering, October, 1983, pp. 83-87.

35. **Shannon, R. E.:** Systems Simulation: The Art and Science, Prentice-Hall, 1975.

36. **Szegedi Z.- Prezenszki J.:** Logisztika menedzsment. Kossuth Kiadó, Budapest, 2003.

37. **Wilson, J. R.:** Statistical Techniques for Simulation Practitioners. Course Notes, Pritsker & Associates, 1985.