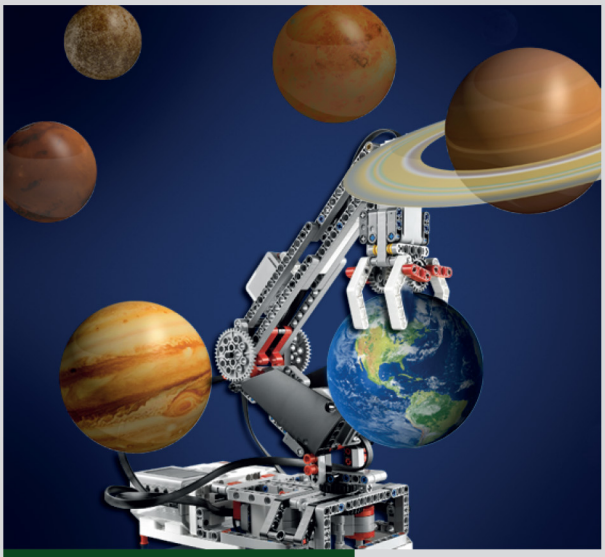


TERMÉSZETTUDOMÁNY

KOVÁCS D. LEHEL ISTVÁN

LEGO ROBOTOK PROGRAMOZÁSA

Sapientia Könyvek

Scientia Kiadó | 2020

KOVÁCS D. LEHEL ISTVÁN

LEGO ROBOTOK PROGRAMOZÁSA

SAPIENTIA KÖNYVEK

SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

LEGO ROBOTOK PROGRAMOZÁSA

KOVÁCS D. LEHEL ISTVÁN

Scientia Kiadó
Kolozsvár · 2020

SAPIENTIA KÖNYVEK

Természettudományok/Informatika

Kiadja a
Scientia Kiadó
400112 Kolozsvár, Mátyás király (Matei Corvin) u. 4.
Tel./fax: +40-364-401454, e-mail: scientia@kpi.sapientia.ro
www.scientiakiado.ro

Felelős kiadó:
Kása Zoltán

Lektorálta:
Darvay Zsolt (Kolozsvár)

Első magyar nyelvű kiadás: 2020

Descrierea CIP a Bibliotecii Naționale a României
KOVÁCS, LEHEL ISTVÁN
LEGO robotok programozása / Kovács D. Lehel István. - Cluj-Napoca : Scientia, 2020
Conține bibliografie
ISBN 978-606-975-041-4

004

TARTALOMJEGYZÉK

CONȚINUT

CONTENTS

BEVEZETŐ

A *robot* egy elektromechanikai szerkezet, amely előzetes programozás alapján képes különböző feladatok végrehajtására.

A robotok lehetnek közvetlen emberi irányítás alatt, vagy önállóan is végezhetik munkájukat, többnyire egy számítógép felügyeletére bízva.

A *robot* szó a szláv *robota* szóból ered, amelynek jelentése: *szolgamunka*, *munka*. A robot szót Karel Čapek (Malé Svatoňovice, 1890. január 9. – Prága, 1938. december 25.) használta először az 1921-ben megjelent *R.U.R.* című játékában.

1953-tól kezdődően Isaac Asimov (Petrovicsi, 1920. január 2. – New York, 1992. április 6.) 38 novellából és 5 regényből álló tudományos-fantasztikus sorozatot írt a pozitronikus robotokról (olyan elképzelt robotok, amelyek pozitronaggyal rendelkeznek. A pozitron az elektron antirészecskéje, vagyis pozitív töltésű elektron). Ő fogalmazta meg a robotika három törvényét is [18].

A robotokkal rendszerint olyan munkákat végeztetnek, amelyek túl veszélyesek vagy túl nehezek az ember számára, vagy egyszerűen túl monoton, de nagy pontossággal végrehajtandó feladat, ezért egy robot sokkal nagyobb biztonsággal képes elvégezni, mint az emberek. Ezeken kívül robotokat szoktak felhasználni katonai célokra is.

A modern robotok általában öt fő alkotóelemmel rendelkeznek:

– egy mozgatható váz,
– egy motorrendszer,
– egy érzékelőrendszer,
– egy energiaforrás és
– egy számítógépes „agy”.

A mozgatható váz részeit motorok irányítják, ezek lehetnek villanymotorok, vagy akár hidraulikus vagy pneumatikus rendszerek.

A motorok működtetéséhez energiaforrásra van szükség, ezt elemről, akkuról vagy hálózatról kaphatják a robotok.

A motorokat, s így a robot mozgását egy számítógépes „agy” irányítja. A legtöbb robot újraprogramozható, viselkedése megváltoztatható egy új program megírásával.

A fejlettebb robotok saját érzékelőrendszerrel, szenzorokkal rendelkeznek. A leggyakoribb szenzor a mozgásérzékelő, melynek segítségével a robot képes saját mozgását nyomon követni.

A robotok nem a modern kor találmányai. Tarentumi Arkhütasz már Kr. e. 2500 évvel egy repülni tudó fagalambot épített. II. Ptolemaiosz egyiptomi fáraónak Kr. e. 200-ban volt egy *android*ja (ember alakját utánzó, emberszabású robot). Egyes leírások szerint az alexandriai Héron egy éneklő madarat, és vele egy

szerkezetben elhelyezve, a zajra mérgesen hátraforduló baglyot készített. Az 1200-as években élt német tudós, Albertus Magnus egy fémből, viaszból, bőrből és üvegből készült mechanikus ember feltételezett alkotója.

Ebü'l-Izz al-Cezeri (Badīʿ az-Zaman Abu l-ʿIzz ibn Ismāʿīl ibn ar-Razāz al-Jazarī) mérnök automatákat, robotokat épített Diyarbakırban a 12. és 13. században.

Mátyás király udvari csillagásza, Regiomontanus egy műsast készített, amely a nürnbergi városkapu felett szárnycsattogtatással és főhajtással üdvözölte az odaérkező Miksa császárt. Kempelen Farkas magyar tudós elsőként készített beszélő gépet 1770 tájékán. Jacques de Vaucanson fizikus, gyártulajdonos és feltaláló két életnagyságú teremtményt alkotott: a fuvolajátékost és a fuvola-dob virtuózt. Ebben az időben a svájci Pierre Jaquet-Droz három életnagyságú automatát készített. A *Művész* például négy különböző arckép megrajzolására volt képes.

A 20. században a technikai fejlődés ugrásszerűen megnövekedett. Ennek nagy hatása volt a robotokra is. Híres volt R. J. Wensley mérnök androidja, melyet füttyjelekkel lehetett irányítani, vagy Harry May 1932-ben gyártott kéttonnás robotja, amely revolverrel lőtt célba, és 20 méterről minden golyóval beletalált egy almába.

1956-ban alapította George Devol és Joseph Engelberger az első ipari robotokra specializálódott céget, 4 évvel később pedig az MIT Servomechanisms Laboratory bemutatta a számítógép-vezérelt gyártást.

Az UNIMATE, az első ipari robot a 60-as évek elején állt munkába a General Motorsnál.

A jelenben robotokat használunk az időjárás előrejelzéséhez, kommunikációs feladatok elvégzésére, a hadászatban, a tűzszerészetben, az űrkutatásban, de jelen vannak utazásoknál, a konyhában és sok más területen is.

Most, a modern korban, a robotika jövőjét a mesterséges intelligenciával rendelkező emberszabású robotok fejlesztése képezi.

A könyvben szereplő programok letölthetők a https://ms.sapientia.ro/~klehel/robot/ honlapról.

1. A LEGO MINDSTORMS EV3-RÓL

A LEGO Mindstorms EV3 harmadik generációs LEGO robot. 2013 szeptemberében e termékcsalád megjelentetésével ünnepelte a népszerű Mindstorms játék- és oktatóeszköz tizenötödik születésnapját a LEGO.

A Mindstorms EV3 a korábbi modellnél gyorsabb processzort (egy ARM9 alapú 32 bites RISC processzort, amelyen *Linux* fut) és nagyobb memóriát kapott, így a rendszer lelkét képező *téglára* (brick) előre megírt programok segítségével komolyabb feladatok is bízhatók [10].

A LEGO Mindstorms EV3 robot önálló életet élhet, elszakadhat a programozásához használt számítógéptől, így fontos szerephez jut a mobil kommunikációs eszközökkel való szorosabb együttműködés.

Természetesen a LEGO Mindstorms EV3 modell legfontosabb eleme az intelligens, programozható *tégla*. Ez az 1. ábrán látható tégla az 1. táblázatban szereplő paraméterekkel rendelkezik [20].

1. ábra. *Az EV3 programozható tégla*

1. táblázat. *Az EV3 programozható tégla technikai jellemzői*

Kijelző	Monokróm LCD, felbontás: 178×128 pixel
Processzor	300 MHz, Texas Instruments, Sitara AM1808 (ARM9 core)
Memória	64 MB RAM, 16 MB Flash, microSDHC általi kiterjeszthetőség
USB-port	Igen
WiFi	Opcionális az USB-porton keresztül
Bluetooth	Igen
Apple eszközök	Kompatibilis az Apple eszközökkel

A *Linux* alapú firmware-nek, az SD kártyaolvasónak és az USB-portnak köszönhetően a LEGO Mindstorms EV3 tégla tetszőlegesen újraprogramozható, így a bővíthetőségnek és az alakíthatóságnak szinte csak a fantázia szab határt.

Az elektronikai rendszerekben és a számítástechnika területén *firmware* alatt azokat a rögzített, többnyire kisméretű programokat és/vagy adatstruktúrákat értjük, melyek különböző elektronikai eszközök vezérlését végzik el. Ez maga a LEGO EV3 robot operációs rendszere.

Ha termékként nézzük, a Mindstorms doboz szinte minden eleme megújult. Kereskedelmi forgalomba két kiszerelésben került a termék [9], [25], [26]:

– *EV3 Home* (kódja: 31313), illetve
– *Education EV3 Core Set* (kódja: 45544)

Az *EV3 Home* doboz tartalma:
– 1 EV3 programozható tégla,
– 2 nagy motor,
– 1 közepes motor,
– 1 érintésérzékelő,
– 1 színérzékelő,
– 1 infravörös érzékelő,
– 1 távirányító,
– 8 kábel,
– 1 USB-kábel,
– valamint 585 LEGO TECHNIC elem.

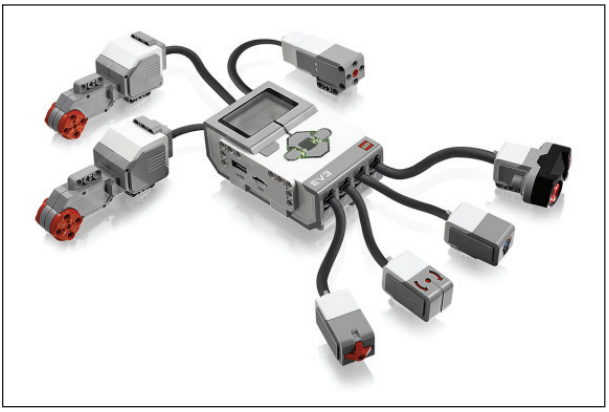
Az *Education EV3 Core Set* doboz tartalma:
– 1 EV3 programozható tégla,
– 2 nagy motor,
– 1 közepes motor,
– 2 érintésérzékelő,
– 1 giroszkópos érzékelő,
– 1 ultrahangos érzékelő,
– kábelek,
– 1 USB-kábel,
– 1 újratölthető elemkészlet,
– valamint LEGO TECHNIC elemek.

Az egyik legfontosabb újdonság a továbbfejlesztett infravörös érzékelő, amely minden korábbinál nagyobb kontrollt tesz lehetővé az EV3 robot működése felett.

A két változat téglái teljesen megegyeznek, a firmware más-más. Ami a LEGO TECHNIC elemeket illeti, külön lehet vásárolni kiegészítő szetteket a Home-hoz, így azt Education kiadássá tudjuk fejleszteni [23].

Javasoljuk az ultrahangos és a giroszkópos érzékelő megvásárlását, valamint a LEGO MINDSTORMS Education Replacement Pack 1 (2000700), 2 (2000701), 3 (2000702) stb. megvásárlását [31], [32].

2. ábra. *Az Education EV3 Core Set rendeződoboza*



3. ábra. *Az EV3 tégla motorokkal és érzékelőkkel*

1.1. Érzékelők

A 2. táblázat a LEGO Mindstorms EV3 érzékelőinek adatait tartalmazza.

2. táblázat. *Az EV3 érzékelői*

Név	Kép	Adatok, tulajdonságok
Infravörös érzékelő 45509 IR Sensor		– A robot környezetének detektálása (50–70 cm), – Infravörös távirányítás érzékelése (2 m), – Jelcsatornák, – Parancsok.
*Ultrahangos érzékelő** 45504 Ultrasonic Sensor		– 1 és 250 cm közötti távolságmérés, – +/– 1 cm pontosság, – Jeladás közben folyamatosan világító, vétel közben villogó LED, – Érték visszatérítése, ha más ultrahangjelet detektál.
*Giroszkópos érzékelő** 45505 Gyro Sensor		– Szögmérés +/– 3 fokos pontossággal, – 440 fok/másodperc sebességgel kimenetelek generálása, – 1 kHz mintavételezési frekvencia.
Színérzékelő 45506 Color Sensor		– Méri a visszavert vörös fény és a környezeti fény intenzitását a sötéttől a világosig (0–100 skálán), – 7 színt ismer fel: fehér, fekete, kék, zöld, sárga, piros, barna, illetve ismeri a „nincs szín” állapotot, – 1 kHz mintavételezési frekvencia.

Név	Kép	Adatok, tulajdonságok
Érintés-érzékelő 45507 Touch Sensor		– Érzékeli, ha a gomb meg volt nyomva vagy el volt engedve, – különbséget tud tenni az egyszeri és a többszöri megnyomás között.

* Az *Education EV3 Core Set*ben

1.2. Motorok

A 3. táblázat a Mindstorms EV3 motorainak adatait és leírásait tartalmazza.

3. táblázat. *Az EV3 motorok*

Név	Kép	Adatok, tulajdonságok
Nagy motor 45502 Large Servo Motor		– 1 fok pontosságú tacho-visszacsatolás, – 160–170 RPM (fordulatszám percenként), – 20 N/cm üzemi nyomaték, – 40 N/cm maximális nyomaték.
Közepes motor 45503 Medium Servo Motor		– 1 fok pontosságú tacho-visszacsatolás, – 240–250 RPM (fordulatszám percenként), – 8 N/cm üzemi nyomaték, – 12 N/cm maximális nyomaték.

Megjegyezzük, hogy a *tachométer* vagy a *fordulatszámmérő* olyan eszköz, amely a motor forgási sebességét méri. A LEGO motorok tachométerként is képesek működni, a tacho-visszacsatolás jelzi az intelligens téglának, hogy milyen szögben fordult el a motor.

1.3. Más eszközök

A 4. táblázat a LEGO Mindstorms EV3 más eszközeinek, kellékeinek adatait tartalmazza.

4. táblázat. *Az EV3 eszközök*

Név	Kép	Adatok, tulajdonságok
Távirányító 45508 IR Beacon		– 4 csatorna, – 1 váltógomb, – 4 gomb, – zöld LED jelzi az aktivitást, – 2 AAA elem, – egy óra inaktivitás után kikapcsol.
Kábelek 45514 Cable Pack		– 4 db 25 cm hosszú, – 2 db 35 cm hosszú, – 1 db 50 cm hosszú kábel.

1.4. Robotok

A LEGO a Mindstorms EV3 mellé tizenhét különböző robot felépítéséhez kínál részletes leírást, útbaigazítást és előre megírt programot.

Az előbbiekből 5 robot, EV3RSTORM, GRIPP3R, R3PTAR, SPIK3R és TRACK3R útmutatója megtalálható az alapcsomagban, 12 továbbié, ROBODOZ3R, BANNER PRINT3R, EV3MEG, BOBB3E, MR-B3AM, RAC3 TRUCK, KRAZ3, EV3D4, EL3CTRIC GUITAR, DINOR3X, WACK3M, valamint EV3GAME pedig a LEGO honlapjáról (*www.lego.com*) tölthető le.

Most először a robothoz háromdimenziós összeépítési segédlet is jár, ezen az összeszerelés lépéseit bemutató ábrák térben forgathatók és nagyíthatók.

Természetesen nagyon sokfajta robotot készíthetünk mi magunk is, vagy az interneten megismerhetjük a mások által megépített robotokat is. Egy érdekes robot például szét tudja válogatni a különböző színű LEGO-kockákat [33].

4. ábra. *GRIPP3R* [40]

5. ábra. *R3PTAR* [40]

1.5. Az intelligens tégla

A LEGO Mindstorms EV3 robotok „agya” és „szíve” az intelligens tégla (45500).

A központi egység 6 gombos világító vezérlője színváltozásával jelzi az egység aktív állapotát. A központi egység nagy felbontású fekete-fehér kijelzője, hangszórója, USB-portja, mini SD-kártyaolvasója, 4-4 ki/bemeneti csatlakozója mutatja sokoldalúságát. A tégla számítógéppel való kommunikációs lehetőségei pedig: Bluetooth és WiFi [14]. Ezen felsorolt kommunikációs csatornákon keresztül nyílik lehetőség az intelligens tégla programozására vagy az adatok kinyerésére, adatkommunikációra. Kompatibilis mobil eszközökkel, a működéséhez szükséges energiát pedig 6 db AA (ceruza) elem biztosítja, vagy lehetőség van az EV3 DC akkumulátoráról való működtetésre is.

Az intelligens tégla (központi egység) főbb jellemzői [13], [27]:

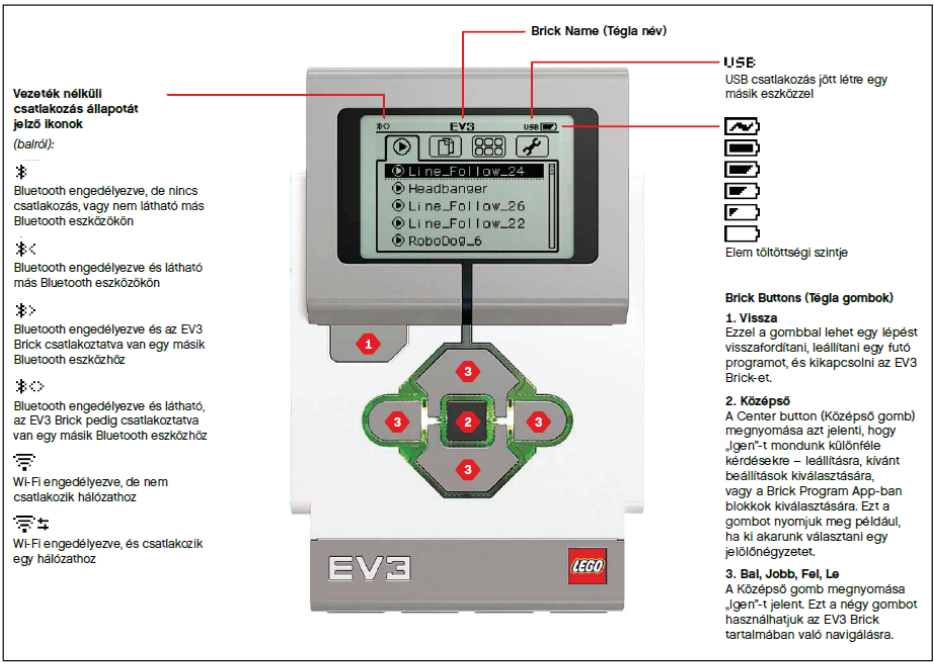
– Processzora: ARM9 Linux-alapú operációs rendszerrel. Ez egy ún. RISC (Reduced Instruction Set Computing), csökkentett utasításkészletű processzor. Ezekre jellemző, hogy nagyon egyszerű utasításkészletük van, így az utasítások végrehajtási ideje alacsony. Huzalozott logikájúak, tehát már ezért is gyorsabbak, mint a CISC (Complex Instruction Set Computing) processzorok, melyek mikroprogramozottak. Hátrányuk, hogy a nagyon egyszerű utasításkészlet miatt a rájuk írt programok nagyon bonyolultak lehetnek, mert míg a mikroprogramozott processzorok egy utasítással több műveletet hajtanak végre, úgy a RISC processzoroknál ezeket a műveleteket több utasításból kell összerakni. Az ARM9-et azonban már ezen a téren is optimalizálták. Vannak pl. olyan összetettebb utasításai, melyek bizonyos esetekben kiválthatják a feltételes elágazás használatát [12], [16].
– Négy bemeneti csatlakozó (port) adatgyűjtéshez akár 1000 minta/sec mintavételezési sebességgel.
– Négy kimeneti csatlakozó a parancsok végrehajtásához.
– Programok tárolására belső memóriája: 16 MB Flash memória és 64 MB RAM.
– Mini SDHC-kártyaolvasó 32 GB kapacitású kártyák olvasására.
– 3 színben világító 6 gombos kezelőfelület (az egység állapotának kijelzésére).
– Nagy felbontású, 178×128 (képpont) kijelző grafikon, grafika és az adatok megjelenítésére.
– Kiváló minőségű beépített hangszóró.
– A központi egység programozhatósága/adatátvitel (mérési adatok kinyerése) EV3 szoftver segítségével.
– Számítógép-intelligens tégla kommunikációs kapcsolat lehetősége USB-n, Bluetooth-on, WiFi-n keresztül.
– Az USB-n keresztül lehetőség van a téglák láncszerű összefűzésére, WiFi kommunikációjára, pendrive stb. csatlakoztatására.

– Tápellátása 6 darab AA elemmel biztosítható, vagy az EV3 tölthető (2050 mAh) akkumulátorokkal.

A 6. ábrán az EV3 téglát látjuk felülnézetből. Az eléggé nagy felbontású (178×128 pixel) kijelző az előlap felső részét foglalja el, alatta 6 darab nyomógomb helyezkedik el, esztétikus designt követve.

A gombokat három csoportba oszthatjuk:

1. *Vissza*: Ezzel a gombbal lehet egy lépést visszafordítani, leállítani egy futó programot, és kikapcsolni az EV3 téglát.
2. *Középső*: A középső gomb megnyomása azt jelenti, hogy „Igen”-t mondunk különféle kérdésekre – leállításra, kívánt beállítások kiválasztására vagy a Brick Program Appban blokkok kiválasztására. Ezt a gombot nyomjuk meg például, ha ki akarunk választani egy jelölőnégyzetet.
3. *Bal*, *Jobb*, *Fel*, *Le*: Ezt a négy gombot használhatjuk az EV3 tégla tartalmában való navigálásra.



6. ábra. *Az EV3 tégla előlapja* [40]

Az úgynevezett *Brick Status Light*, a tégla állapotát jelző fény, amely körülveszi a gombokat, tájékoztat az EV3 tégla aktuális állapotáról:

– *Vörös*: Indítás, Frissítés, Leállítás
– *Villogó vörös*: Foglalt
– *Narancsszínű*: Figyelmeztetés, Kész
– *Villogó narancsszínű*: Figyelmeztetés, Működés
– *Zöld*: Kész
– *Villogó zöld*: Programfutás

7. ábra. *Az EV3 tégla oldallapjai*

A 7. ábrán látható oldallapokon kaptak helyet a Ki/Bemeneti csatlakozó portok, a hangfal, valamint a microSD-kártya helye is.

Az **1**-es, **2**-es, **3**-as és **4**-es bemeneti portokon keresztül érzékelőket csatlakoztathatunk az EV3 téglához.

Az **A**, **B**, **C** és **D** kimeneti portokon keresztül motorok csatlakoztathatók az EV3 téglához.

A Mini-USB PC port a D port mellett található, és ezzel csatlakoztathatjuk az EV3 téglát a számítógéphez.

Az USB gazdaportot felhasználhatjuk például egy USB WiFi hardverkulcs hozzáadására, hogy vezeték nélküli hálózathoz csatlakozhassunk, vagy akár négy EV3 téglát is összekapcsolhassunk lánckapcsolással.

Az SD-kártyaport lehetőséget ad arra, hogy egy SD-kártyával megnöveljük az EV3 tégla felhasználható memóriáját maximum 32 GB-tal.

Hangszóró: minden hang innen érkezik, beleértve a hangeffektusokat is, amelyeket a robotok programozásában használunk. Ha a hangminőség fontos, próbáljuk úgy megtervezni a robotot, hogy a hangszóró ne legyen takarva.

Az EV3 tégla bekapcsolásához a középső gombot (2) kell megnyomni, ezután a tégla állapotjelző fénye piros színűre vált és megjelenik a kezdőképernyő. Amikor a fény zöld színűre vált át, az EV3 tégla működésre kész.

Az EV3 tégla kikapcsolásához nyomjuk a vissza gombot (1) addig, amíg a lekapcsolás képernyő meg nem jelenik. Ezen már ki lesz választva a *megszakítás*t jelentő X. Ha ezt választjuk, a kikapcsolási folyamat leáll. Válasszuk ki az *elfogad*ot jelentő jelölőnégyzetet a jobb gombbal, majd nyugtázzuk ezt a középső gomb megnyomásával. Az EV3 tégla le fog így állni.

Az EV3 tégla felhasználói felülete eléggé egyszerű. Igazából négy ablakból áll:

– Legutóbbi futtatás (Run recent)
– Állomány navigáció (File Navigation)
– Tégla appok (Brick Apps)
– Beállítások (Settings)

A *Legutóbbi futtatás* ablakban mindaddig nincsenek elemek, amíg nem kezdünk el programokat letölteni és futtatni. Itt a legutóbb futtatott programok lesznek láthatók. A listában legfelül lévő program, amely alapértelmezés szerint ki van választva, a legutóbb futott program.

Az Állomány navigáció ablakban érhetjük el és kezelhetjük az EV3 téglán lévő összes állományt, beleértve az SD-kártyán tárolt állományokat is. Az állományok úgynevezett *projektmappák*ba kerülnek, amelyek a tényleges programállományok mellett az egyes projektekben felhasznált hangokat és képeket is tartalmazzák. Itt áthelyezhetjük és törölhetjük az állományokat. A tégla *program app* felhasználásával készített programok tárolása külön, a *BrkProg_SAVE* mappában történik.

A *Tégla appok* ablak már bonyolultabb, több lehetőséget kínál.

Az EV3 téglára négy alkalmazást telepítettek gyárilag és használatra készen. Ezek a következők:

– Port nézet (Port View)
– Motorvezérlés (Motor Control)
– IR-vezérlés (IR Control)
– Tégla program (Brick Program)

A *Port nézet* ablakán könnyen áttekinthetjük azt, hogy melyik porthoz vannak érzékelők vagy motorok csatlakoztatva. Az EV3 tégla gombjaival navigálhatunk a portokhoz, s itt megtalálhatjuk az érzékelőről vagy a motorról visszaküldött aktuális értéket.

A *Motorvezérlés* segítségével irányíthatjuk azon motorok előre vagy hátra mozgását, amelyek csatlakoztatva vannak a négy kimeneti port valamelyikéhez. Együtt irányíthatjuk azokat a motorokat, amelyek az A portra (a Fel és Le gombokat használva) és a D portra (a Bal és a Jobb gombokat használva) vannak csatlakoztatva, illetve a B porthoz (a Fel és Le gombokat használva) és a C porthoz (a Bal és a Jobb gombokat használva) csatlakoztatott motorokat.

Az *IR-vezérlés* a távirányítót adóként, az infravörös érzékelőt pedig vevőként használva, lehetőséget biztosít a négy kimeneti port egyikéhez csatlakoztatott motor előre, hátra mozgására.

A *Tégla program* pont segítségével a számítógépre telepítetthez hasonló, ám leegyszerűsített tervezőprogramot indíthatunk el. Itt lehetőségünk van ve-

zérlőblokkok hozzáadására és programozására, törlésére, programok futtatására, mentésére, megnyitására. Ezt a lehetőséget részletesen *A LEGO Mindstorms EV3 programozása* című fejezetben fogjuk tárgyalni.

A *Beállítások* ablak lehetővé teszi, hogy megtekintsük és módosítsuk a tégla különféle általános beállításait:

– *Hangerő*: az EV3 hangszóróról érkező hang hangerejének beállítása. A Jobb és a Bal gombbal módosíthatjuk a hangerőt, amelynek beállítása a 0% és 100% közötti tartományban lehetséges.
– *Alvó mód*: Ha módosítani akarjuk a tégla alvó módja előtti inaktív időszakát, akkor a Jobb és a Bal gombbal kiválaszthatunk egy rövidebb vagy hosszabb időtartamot, amely 2 perctől végtelenig (*never*) tarthat.
– *Bluetooth*: A Bluetooth kommunikáció beállításait érhetjük el.
– *Wi-Fi*: Itt engedélyezhetjük a Wi-Fi kommunikációt az EV3 téglán és csatlakozhatunk egy vezeték nélküli hálózathoz.
– *Téglainformációk*: Itt találhatóak a tégla aktuális műszaki adatai, a hardware és a firmware verziója, az EV3 operációs rendszerének buildszáma vagy a szabad memória mérete is.

2. A LEGO MINDSTORMS TÖRTÉNETE

Az 1990-es évek elején a LEGO építőkocka-gyártó cég kidolgozott egy rendszert, amelynek segítségével (igénybe véve az eddig már létező LEGO építőelemeket, kockákat, fogaskerekeket, rudakat stb.) robotokat lehet tervezni, építeni és programozni, melyek aztán autonóm módon működhetnek [11].

A rendszer, amit a LEGO elképzelt, nagyon egyszerű. Ahhoz, hogy egy robot egyáltalán programot futtathasson, szüksége van egy központi egységre, amit programozni lehet. Ahhoz, hogy mozogjon, motorokra, illetve ahhoz, hogy a környezetéből információkat kapjon, pedig szenzorokra, érzékelőkre van szüksége. Ezeket kell összekapcsolni egymással, illetve egyéb alkatrészekkel, amelyek a robot vázát alkotják és működésében segítik. Az összekötő elemek már megvoltak, a többit pedig próbálták úgy megtervezni, hogy a kapott eredmény biztonságos, kicsi, strapabíró és kompatibilis legyen a már létező LEGO építőelemekkel. Végül is sikerült egy olyan terméket készíteniük, amely megfelelt mindezen kritériumoknak, és egyben elérhető ára is volt.

Ez a terv kapta a *Mindstorms* kódnevet, melynek ereje, a könnyű újraépíthetőség mellett abban áll, hogy a központi chipjét egy asztali számítógépen megírt és lekompilált programmal lehet feltölteni. A névadást *Seymour Aubrey Papert* (sz. 1928. február 29.) 1980-ban megjelent könyve, a *Mindstorms: Children, Computers, and Powerful Ideas* (*Elmeviharok: gyerekek, számítógépek és erőteljes ötletek*) ihlette.

2.1. Az első generáció

A Mindstorms Robotics Invention termékszett hardver és szoftver gyökere az *MIT Media Lab* által létrehozott programozható tégla, amelyet *Brick Logo* nyelven lehetett programozni.

Az első vizuális programozási környezetet, amelyet 1994-ben hoztak létre a Coloradoi Egyetemen, és amelynek az alapja az *AgentSheets* volt, *LEGOsheets*nek hívták.

Az eredeti Mindstorms Robotics Invention szett két motort, két érintésérzékelőt és egy fényérzékelőt tartalmazott.

Minden első generációs LEGO robot lelke az RCX (*Robotic Command eXplorer*) volt. Hozzá kapcsolódtak a motorok és az érzékelők. Rajta futott a program, amely eldöntötte, hogy mi legyen a következő mozdulat. Az RCX egy infravörös torony segítségével kommunikált a számítógéppel.

8. ábra. *Az RCX* [11]

Az RCX magja egy 32 K RAM-mal rendelkező Hitachi H8-as mikrokontroller volt. Ez a chip irányította a három-három ki- és bemenetet, illetve a sorosan kötött infravörös kommunikációs portot. A chipen levő 16 K-os ROM memóriában egy kis program volt tárolva, amely az első futás alkalmával aktiválódott. Ezt helyettesíthette később egy, a számítógépről letölthető, apró operációs rendszer. Miután ez a program felkerült az RCX-re, a felhasználó programjainak 6 K memória maradt. Ez a kis memória csak kisebb alkalmazások tárolására volt alkalmas, ezért komplexebb programokat nem lehetett futtatni a rendszeren [17].

Ennek az akadálynak az egyik leküzdési módja egy olyan alkalmazás tervezése volt, amely a számítógépen futott, és ott hajtódott végre a számítások nagy része. A számítások eredményét pedig üzenet formájában el lehetett küldeni az RCX-nek, amely lefordította az adatokat a motorok nyelvére, azaz végrehajtotta a megfelelő mozdulatokat.

Így természetesen a kommunikációra tevődött a nagy hangsúly, de ez a mobilitás kárára ment, hisz az infravörös jel nem fogható, csak behatárolt távolságon belül.

5. táblázat. *Az RCX programozható tégla technikai jellemzői*

Processzor	8-bit Hitachi H8/3292, 16 Mhz
ROM	16 Kb
SRAM, chipen	512 byte
SRAM, külső	32 Kb

Kimeneti eszközök	3 motorport, 9 V 500 mA
Bemeneti eszközök	3 érzékelő port
Kijelző	1 monochrom LCD
Hang	1 hangkijelző egység
Időmérő	4 időmérő (8-bit)
Elemek	6×1,5 V
Kommunikáció	IR port (közvetítő + fogadó)

Az intelligens RCX-téglát a következő nyelveken lehetett programozni:
– LEGO által támogatott nyelvek:
  • RCX Code
  • A LabVIEW alapú ROBOLAB, amelyet a Tufts Egyetem fejlesztett ki
– Más népszerű nyelvek:
  • GNAT GPL: Ada alapú
  • LeJos: Java alapú
  • Not eXactly C: (NXC), egy nyílt forráskódú C-szerű nyelv
  • Not Quite C: (NQC)
  • RoboMind: egyszerű didaktikai szkript-nyelv
  • ROBOTC: C alapú nyelv, programozási környezettel
  • Simulink: grafikus nyelv
  • pbFORTH: a Forth kiterjesztése
  • pbLua: egy Lua-verzió
  • Visual Basic: COM+ interfész által

A 6. táblázat a LEGO Mindstorms RCX motorainak, érzékelőinek adatait tartalmazza.

6. táblázat. *Az RCX eszközök*

Név	Kép	Adatok, tulajdonságok
Motor		A motor átlagos forgásszáma teher nélkül 350 RPM, átlagos súly alatt pedig 200/250 RPM. A motor 9 V-os, kevés energiát fogyaszt. Tud forogni előre, hátra, és be lehet állítani a forgási sebességet is.

Név	Kép	Adatok, tulajdonságok
Érintésérzékelő		Ha megnyomjuk az érintésérzékelő gombját, akkor áram halad át a csatlakoztatón. Az RCX képes ezt érzékelni és tudja, hogy mikor van lenyomva az érintésérzékelő gombja és mikor nem.
Fényérzékelő		A fényérzékelő a fény erősségét adja meg a leolvasás pillanatában egy 0 és 100 közötti értékkel.
Kábelek		A kábelek kötik össze az RCX-et a ki- és bemeneti eszközökkel. Nem mindegy, hogy hogyan kötjük össze a kábeleket a motorokkal vagy az érzékelőkkel, hiszen ettől függnek a bemeneti és kimeneti parancsok.
Infravörös torony		Az IR-torony USB-n keresztül kapcsolódik a számítógéphez. Az adatok, programok fény útján, infravörös tartományban jutnak el a toronytól az RCX-hez.
Távirányító		A távirányítóval parancsokat tudunk küldeni az RCX-nek: pl. egy program futása, megadva a program sorszámát; egy program leállítása; egy motor mozgatása előre vagy hátra; üzenetek küldése.

2.2. A második generáció

A 2006 júliusában megjelenő második generáció alapja már a LEGO TECHNIC lett.

Az alapcsomag két verzióban jelent meg:
– Retail Version (8527)
– Education Base Set (9797)

Az intelligens tégla és a szett neve Lego Mindstorms NXT lett. A téglán már négy csatlakozónak került hely szenzorok, érzékelők számára. A vezérelhető motorok száma három maradt, ezen kívül egy USB-csatlakozó is helyet kapott.

A készlet ezen felül tartalmazott:
– 3 szervomotort (amelyekben beépített elfordulásérzékelő volt, így pontosan meg lehetett mérni, hogy hány fokot fordult a motor),
– 1 érintésérzékelőt, amelynek két állása volt (benyomva vagy kiengedve – tehát 0 vagy 1 értéket térített vissza),
– 1 ultrahangos távolságérzékelőt, amely kb. két méterig tudott mérni,
– 1 hangérzékelőt, amely decibelben volt képes mérni a környezete zajszintjét,
– 1 fényérzékelőt, amely a felületről visszaverődő fényt mérte %-ban (volt egy saját LED fényforrása).

9. ábra. *Az NXT-tégla (9841)*

2009. augusztus 5-én a LEGO piacra dobta a LEGO Mindstorms NXT 2.0 verzióját (8547), amely az előbbiekhez képest tartalmazott még:

– egy plusz érintésérzékelőt és
– 1 színérzékelőt, amely 6 színt tudott megkülönböztetni (fekete, fehér, piros, sárga, zöld, kék), de ugyanúgy használható volt fényszenzorként is.

A 7. táblázat az NXT intelligens tégla adatait foglalja össze.

7. táblázat. *Az NXT programozható tégla technikai jellemzői*

Processzor	32-bit Atmel AT91SAM7S256 (256 KB flash memory, 64 KB RAM), 8-bit Atmel ATmega48 microcontroller @ 4 MHz (4 KB flash memory, 512 Bytes RAM)
Kimeneti eszközök	3 motorport
Bemeneti eszközök	4 érzékelő port
Kijelző	1 monochrom LCD, 100×64 pixel
Hang	1 hangkijelző egység 8 kHz hangminőség, 8 bit, 2–16 KHz
Időmérő	4 időmérő (8-bit)
Elemek	6×1,5 V, AA
Kommunikáció	USB-port, Bluetooth Class II V2.0

Eszközök terén már nagyon kiszélesedtek a lehetőségek. A 8. táblázat az NXT-eszközöket foglalja össze.

8. táblázat. *Az NXT-eszközök*

Név	Kép	Adatok, tulajdonságok
Ultrahangos érzékelő 9846 Ultrasonic Sensor		– Érzékeli a távolságot és a mozgást, felismeri a tereptárgyakat. Mértékegysége centiméter vagy inch lehet.
*Fény*érzékelő 9844 Light Sensor		– Képes érzékelni a fényt és a sötétséget, a fény intenzitását méri. Képes mérni a különböző színek intenzitását is.

Név	Kép	Adatok, tulajdonságok
Színérzékelő 9694 Colour Sensor		– Méri a visszavert vörös fényt és a környezeti fényt a sötéttől a csillogó napsütésig, – 6 színt ismer fel, – vörös, zöld, kék lámpaként is működik.
Érintésérzékelő 9843 Touch Sensor		– Érzékeli, ha a gomb meg volt nyomva vagy el volt engedve, – különbséget tud tenni az egyszeri és a többszöri megnyomás között.
Hőérzékelő 9749 NXT Temperature Sensor		– Celsius- és Fahrenheit-fokokat mér, – –20 °C és +120 °C között, vagyis –4 °F és +248 °F között.
Hangérzékelő 9845 Sound Sensor		– Decibelben méri a környezet zaját, hangszintjét. DB- és DBA-mérésre is képes. Beépített hangséma- és hangszínfelismerő rendszere van.
Iránytű szenzor MS1034 Compass sensor		– Iránymeghatározás.

Név	Kép	Adatok, tulajdonságok
Gyorsulásérzékelő MS1040 Accelero-meter sensor		– Ezzel a szenzorral elérhetővé válik a robot számára, hogy merre van felfelé. Az érzékelő 3 tengely mentén *(x, y, z)* képes a gyorsulás mérésére –2g és +2g tartományban. Érzékenysége 200 egység/g. Mintavételezési sebessége 100 minta/sec.
Motor 9842 Interactive Servo Motor		– Forgásérzékelővel ellátott szervomotor, amely fordulat-számot, irányt tud mérni.

Talán ehhez a modellhez készítették a legtöbb érzékelőt, hisz itt jelentek meg a RFID-érzékelők, az elfordulásérzékelők, a mágneses szenzorok, a Vernier-szenzorok, az IR-érzékelők és -keresők, az elektrooptikai távolságérzékelők, a gyorsulás- és dőlésszenzorok stb. [7].

Programozás terén is nagyon kibővültek a lehetőségek. Az NXT-et a következő nyelvekben lehet programozni:

– NXT-G
– LabVIEW Toolkit
– Lego::NXT
– Ada
– Next Byte Codes & Not eXactly C
– ROBOTC
– RoboMind
– NXTGCC
– URBI
– leJOS NXJ
– nxtOSEK
– MATLAB és Simulink
– Lua
– FLL NXT Navigation
– ruby-nxt
– Robotics.NXT

3. A LEGO MINDSTORMS EV3 PROGRAMOZÁSA

Ebben a fejezetben a LEGO Mindstorms EV3 programozását mutatjuk be mind a LEGO saját felülete (LEGO Mindstorms EV3 Home Edition), mind pedig más programozási felületek használata által.

3.1. LEGO MINDSTORMS EV3 Home Edition

Az EV3 LEGO robotok fő szoftverét letölthetjük *a http://www.lego.com/hu-hu/mindstorms/downloads/software/ddsoftwa-redownload/download-software/* honlapról [29].

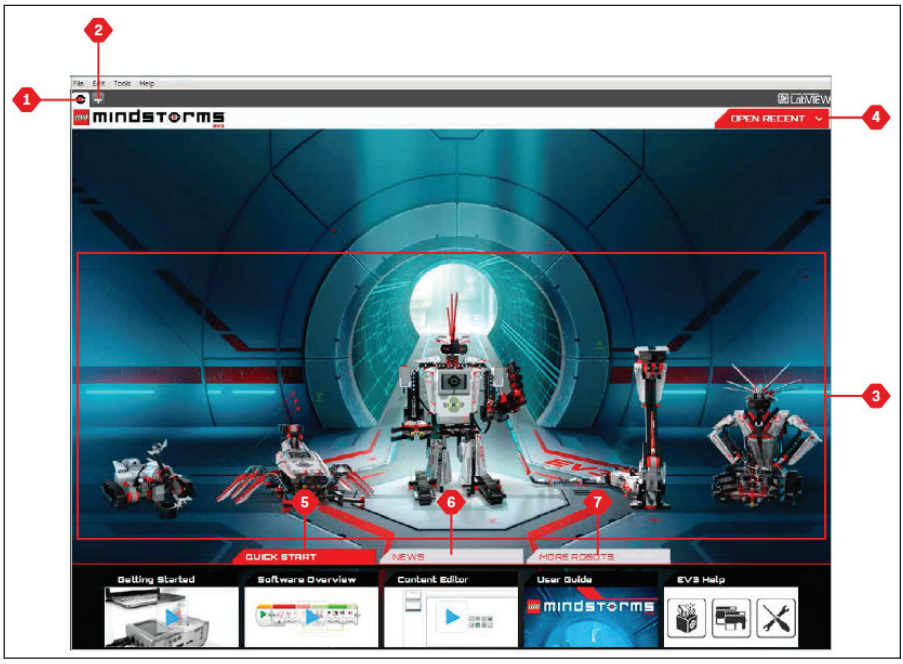
Az EV3 szoftver minimális rendszerkövetelménye: Silverlight 5.0 vagy újabb; Microsoft Dot Net 4.0 vagy újabb; Windows: Windows XP, Vista, Windows 7, vagy Windows 8/10 (+Win RT) (32/64 bit) a legutolsó frissítőcsomagokkal; Macintosh: Mac 10.6, 10.7, and 10.8 a legutolsó javítócsomagokkal; Gépigény: 2 GB RAM vagy több, 5 GHz processzor vagy gyorsabb, legkisebb ajánlott képfelbontás: 1024×600 [28].

Miután ellenőriztük, hogy számítógépünk teljesíti a minimális rendszerkövetelményeket, készen állunk a szoftver telepítésére. Zárjunk be minden más programot, majd kattintsunk duplán az EV3-as szoftver alkalmazásmappájában található telepítőállományra. Ekkor megkezdődik a telepítés.

A LEGO MINDSTORMS EV3 Home Edition egy LabView alapú szoftver. A LabVIEW (**Lab**oratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench) egy grafikus programfejlesztő eszköz (egy G-nyelv, amely először 1986-ban jelent meg a Macintosh gépeken) a National Instrumentstől, amely elsősorban méréstechnikai és a hozzá kapcsolódó jelfeldolgozási feladatok megoldására szolgál, de alkalmas szimulációkra is. A grafikus programozás egy látványos, látszólag könnyen követhető programozási módot jelent, és megtalálhatók benne a hagyományos programozás alapvető jellemzői, mint például a változók, konstansok deklarálása, adattípusok, ciklusok, elágazások szervezése stb.

3.1.1. A szoftver

Amikor elindítjuk a LEGO MINDSTORMS EV3 Home Edition szoftvert, az *előszobá*ban (Lobby) találjuk magunkat. Innen indulva minden lehetőséghez hozzáférhetünk, egyszerűen elérhetjük a szoftver, s ezáltal a robot funkcióit is.



10. ábra. *Az EV3 előszoba* [41]

Az előszobában az alábbi lehetőségeket és erőforrásokat találjuk meg, ezek közül választhatunk:

1. *Előszoba fül* (Lobby Tab): Ez a gomb mindig visszavisz az előszobába.
2. *Projekt hozzáadása* (Add Project): Itt kezdhetünk el egy új projektet, és programozhatjuk a saját robotunkat.
3. *Robotküldetések* (Robot Missions): A LEGO által ajánlott öt fő modell megépítési és programozási utasításai.
4. *Legutóbbi megnyitása* (Open Recent): Könnyű hozzáférés azokhoz a projektekhez, amelyekkel legutóbb dolgoztunk.
5. *Első lépések* (Quick Start): Rövid bevezető videók, EV3 felhasználói útmutató, súgó stb. elérése.
6. *Hírek* (News): Rövid történetek és hírek a LEGO honlapról (internetkapcsolat szükséges).
7. *További robotok* (More Robots): Hozzáférés további modellek építéséhez és programozásához (internetkapcsolat szükséges).

A Projekt hozzáadása (+) gomb megfelel a menüben is megjelenő File / New Project pontnak.

Amikor megnyitunk egy új projektet, az automatikusan létrehoz egy mappát. Minden program, kép, hang, videó, utasítás és egy projekten belül használható bármilyen más eszköz automatikusan ebben a mappában lesz tárolva.

Mindegyik projekt megjeleníthető fül formájában a képernyő felső szélén. Ezen a fülön található egy X gomb is, amely bezárja a fület.

Ha a programfülek mellett, bal oldalon megjelenő fülre kattintunk, amely egy kulcsot tartalmaz jelképként, átkerülünk a *projekt jellemzők* (Project Properties) oldalra. Itt az aktuálisan kiválasztott projektet tekinthetjük meg programjaival, képeivel, hangjával és a többi eszközzel együtt. A projektünkről leírást adhatunk szöveggel, képekkel és videóval, és ezek fogják meghatározni azt is, hogy a projekt hogyan jelenjen meg az előszobában.

A projektünk megjelenő, beállítható jellemzői a következők:

1. *Projektleírás* (Project Description): A projekt címe, leírása, képei, videói.
2. *Projekttartalom előnézet* (Project Content Overview): A projekthez tartozó eszközöket foglalja össze. Itt lesznek a programok, képek, hangok, és a saját blokkok is.
3. *Lánckapcsolás* (Daisy Chain Mode): Ezzel a jelölőnégyzettel kapcsolhatjuk be a lánckapcsolást, vagyis azt, hogy a programunk képes legyen akár négy EV3 téglához is kapcsolódni.
4. *Megosztás* (Share): A LEGO honlapon megoszthatjuk projektünket (internetkapcsolat szükséges).

A Robotküldetések lehetőség a LEGO által ajánlott öt robot, a TRACK3R, SPIK3R, EV3RSTORM, R3PTAR és GRIPP3R megépítési és programozási leírását tartalmazza.

Ha rákattintunk bármelyik robotra, akkor átkerülünk a robot *Küldetés áttekintésére* (Mission Overview). A küldetés kialakításai olyanok, hogy végigvezessenek a programozás legfontosabb elemein, és megismerkedjünk az EV3 rendszerrel. Mindegyik küldetés segít a robot egy részének megépítésében és programozásában. Elkezdjük az első küldetéssel, majd miután teljesítettük azt, továbblépünk a következőre. Ha mindegyik küldetést teljesítettük, befejeződik a robot építése, s kész lesz a parancsaink fogadására.

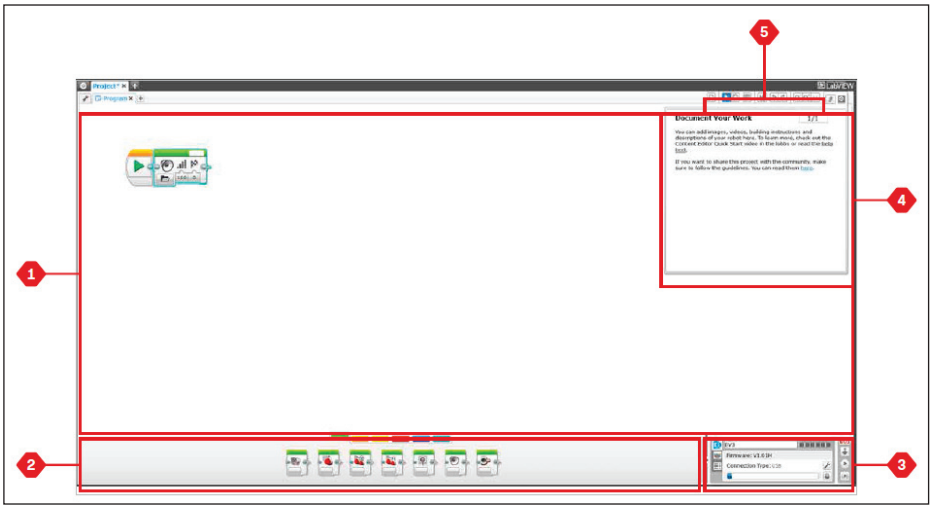
Mindegyik küldetés négy lépésből áll:

– *Cél* (Objective)
– *Létrehozás* (Create)
– *Utasítás* (Command)
– *Rajta* (Go!)

3.1.2. Programozási alapelvek

A robotok programozása grafikus programozási nyelv segítségével történik. Ez a nyelv és felület a következő elemeket tartalmazza:

1. *Programfejlesztői vászon* (Programming Canvas): Ide vázoljuk fel a programunkat.
2. *Programozói paletták* (Programming Palettes): Itt találjuk meg a programunk építőelemeit.
3. *Hardveroldal* (Hardware Page): Itt alakíthatjuk ki és kezelhetjük a kommunikációnkat az EV3 téglával, és itt figyelhetjük meg, milyen motorok és érzékelők vannak csatlakoztatva. Itt tölthetünk le programokat is az EV3 téglára.
4. *Tartalomszerkesztő* (Content Editor): Egy digitális munkafüzet. Itt dokumentálhatjuk a projektünket szövegesen, képekkel és videókkal.
5. *Programozói eszköztár* (Programming Toolbar): Itt találjuk meg a programozási munkánkhoz szükséges alapeszközöket.



11. ábra. *Az EV3 programozói felület* [41]

A hardveroldal információkat közöl az EV3 tégláról. A jobb alsó sarokban elhelyezkedő hardveroldal vezérlő kibontás/összecsukás (Expand/Collapse) segítségével kinagyítható.

A hardveroldal vezérlő funkciói a következők:

1. *Letöltés* (Download): A program letöltése az EV3 téglára.
2. *Letöltés és futtatás* (Download and Run): A program letöltése az EV3 téglára és azonnali indítása, futtatása.
3. *Letöltés és kiválasztott futtatás* (Download and Run Selected): Csak a kijelölt blokkokat tölti le az EV3 téglára, és azonnal el is indítja azokat.

Az EV3 szöveg felül, egy kis ablakban piros színűre vált át, amikor az EV3 tégla csatlakozik a számítógépünkhöz.

A tégla információk (Brick Information) fül fontos információkat jelenít meg az aktuálisan csatlakoztatott EV3 tégláról, például a neve, a telep állapota, a firmware verzió, a csatlakozás típusa és memóriasávja. Hozzáférést biztosít a memóriaböngésző és a vezeték nélküli beállítás eszközökhöz.

A port nézet (Port View) fül az EV3 téglához csatlakoztatott érzékelőkről és motorokról ad információkat. Ha a tégla csatlakoztatva van a számítógéphez, akkor az aktuális értékek lesznek láthatók, ha nincs csatlakoztatva, akkor manuálisan kell beállítani: válasszunk ki egy portot, majd válasszuk ki a megfelelő érzékelőt vagy motort a listából.

Ha úgy írunk programokat, hogy az EV3 tégla nincs csatlakoztatva a számítógéphez, vagy a szoftverből nem programozzuk át másképp, akkor a rendszer alapértelmezett portokat rendel ki a következőképpen [22]:

– 1-es port: érintésérzékelő
– 2-es port: giroszkópos érzékelő vagy hőérzékelő
– 3-as port: színérzékelő
– 4-es port: infravörös érzékelő vagy ultrahang-érzékelő
– A port: közepes motor
– B és C port: két nagy motor
– D port: nagy motor

A *felhasználható téglák* (Available Bricks) fül a rendelkezésre álló téglákat mutatja. Lehetőségünk van eldönteni, hogy melyik téglához csatlakozzunk, és kiválaszthatjuk a kommunikáció típusát.

Az EV3 téglával a következő NXT-érzékelők kompatibilisek:

– *Ultrahangos érzékelő*
– *Fényérzékelő*
– *Érintésérzékelő*
– *Hangérzékelő*

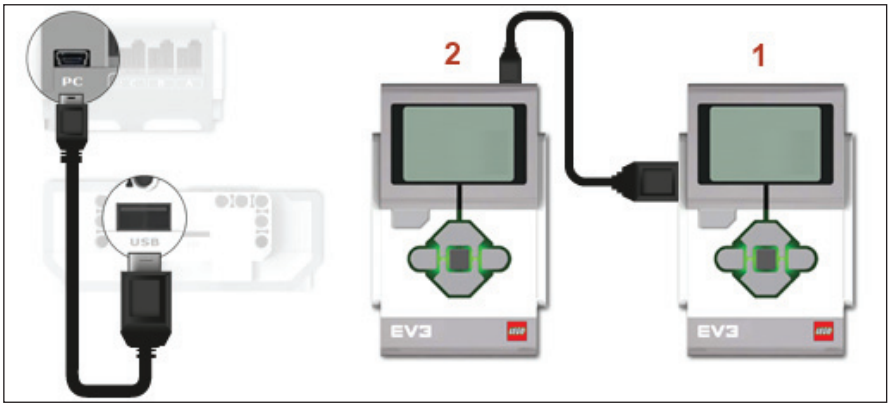
A tartalomszerkesztő kényelmes módot kínál arra, hogy a felhasználók dokumentálhassák projektjeik célját, folyamatát és elemzését. Ide felvehetünk szöveget, képeket, videókat, hangeffektusokat, sőt, akár építési útmutatókat is. Már kész tartalom is helyet kaphat, például a robotküldetések.

Minden egyes oldalon, igény szerint, különféle elrendezések alakíthatók ki, és egy oldal egy sor műveletet képes automatikusan elvégezni, például megadott programok megnyitását vagy egy bizonyos programblokk kiemelését.

A tartalomszerkesztőt a nagy könyv ikonos gombbal lehet megnyitni, ekkor láthatjuk, hogy milyen tartalom készült már el egy projekthez vagy programhoz.

A tartalomszerkesztő főbb területei és jellemzői az alábbiak:

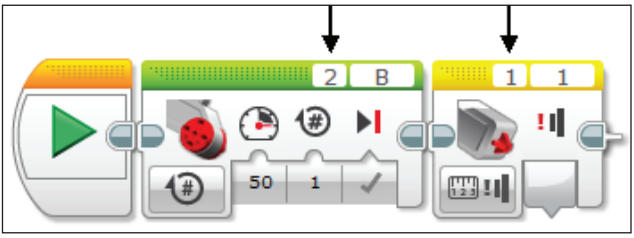
1. *A tartalomszerkesztő megnyitása/bezárása* (Open/Close Content Editor): Itt nyithatjuk meg vagy zárhatjuk be a tartalomszerkesztőt.
2. *Szerkesztési/Megtekintési mód* (Edit/View Mode): A saját oldalainkat itt tekinthetjük meg vagy szerkeszthetjük.
3. *Oldalnavigátor* (Page Navigation): Lépés a következő vagy az előző oldalra.
4. *Oldalcím* (Page Title): Itt címet adhatunk az oldalunknak.
5. *Oldalterület* (Page Area): Itt látható és szerkeszthető a fő tartalom.
6. *Ikonok* (Icons): Itt választhatjuk ki, hogy milyen tartalomtípust szeretnénk felvinni az oldalterületre.
7. *Oldal miniatűrök* (Page Thumbnails): Lépés egy adott oldalra a miniatűr képek alapján.
8. *Oldal hozzáadása/Törlése* (Add/Delete Page): 14 különféle sablonból választhatunk egy új oldal létrehozásánál.
9. *Oldalbeállítás* (Page Setup): Az oldalakon egyedi beállításokat is elvégezhetünk, például beállíthatjuk a formátumot, az oldallal végzett műveletet és a navigálást a következő oldalra.



12. ábra. *Téglák lánckapcsolása* [40]

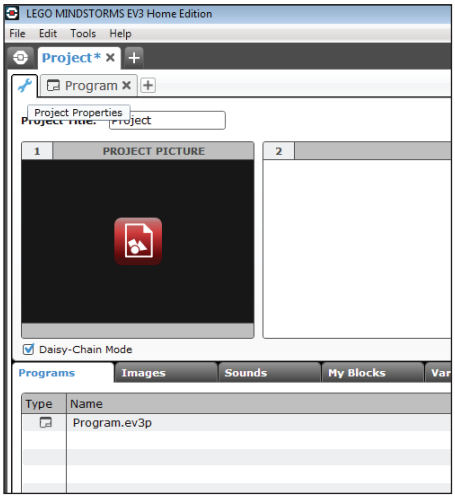
Az EV3 téglákat lánckapcsolással össze lehet kapcsolni a 12. ábrán megadott módon. Legfennebb 4 tégla kapcsolható így össze és programozható egyetlen programban.

Ha lánckapcsolást használunk, akkor megváltoznak a programblokkjaink is, olyan értelemben, hogy a port beállítása mellett megjelenik még egy adatdoboz, amelyben be kell állítani azt is, hogy az adott port melyik téglán értendő (Layer Selector).



13. ábra. *A téglák kiválasztása a portok mellett*

Ha lánckapcsolást akarunk megvalósítani, akkor a projekt tulajdonságainál (Project Properties) be kell állítani a lánckapcsolás módot (Daisy-Chain Mode).



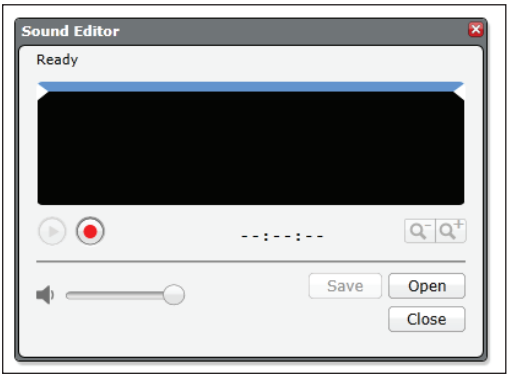
14. ábra. *A lánckapcsolás mód beállítása*

3.1.3. Eszközök

A LEGO MINDSTORMS EV3 Home Edition szoftver *Eszközök* (Tools) menüjében számos eszköz található, amelyek extra funkcionalitást és támogatást nyújtanak az EV3 tégla és szoftver használatához.

A *Hangszerkesztő* (Sound Editor) segítségével ki tudjuk alakítani saját hangeffektusainkat, majd használni tudjuk a szerkesztett hangokat a robotunk programozásában. WAV vagy MP3 formátumú hangállományokat tudunk szerkeszteni, majd a lejátszásra alkalmas RSO kiterjesztéssel a megfelelő formátumban

menteni. Az állomány megnyitása után (Open) a csúszka segítségével jelölhetjük ki a menteni kívánt rész elejét és végét. A kijelölt részt a nagyító ikonra kattintva tovább nagyíthatjuk és vághatjuk. A háromszög alakú ikonra kattintva a kijelölt részt le tudjuk játszani. A mentést a Save gomb segítségével végezhetjük el a név megadása után. Alapértelmezésben az állományt a program telepítési mappájába menti [39].



15. ábra. *Hangszerkesztő*

A *Képszerkesztő* (Image Editor) segítségével képeket, grafikákat tervezhetünk az EV3 tégla kijelzője számára.

A ceruzával szabálytalan alakzatokat rajzolhatunk, ezen kívül vonalat húzhatunk, köröket, téglalapokat rajzolhatunk, festhetünk, törölhetünk, szöveget írhatunk, vagy kiválaszthatunk részeket a rajzból.

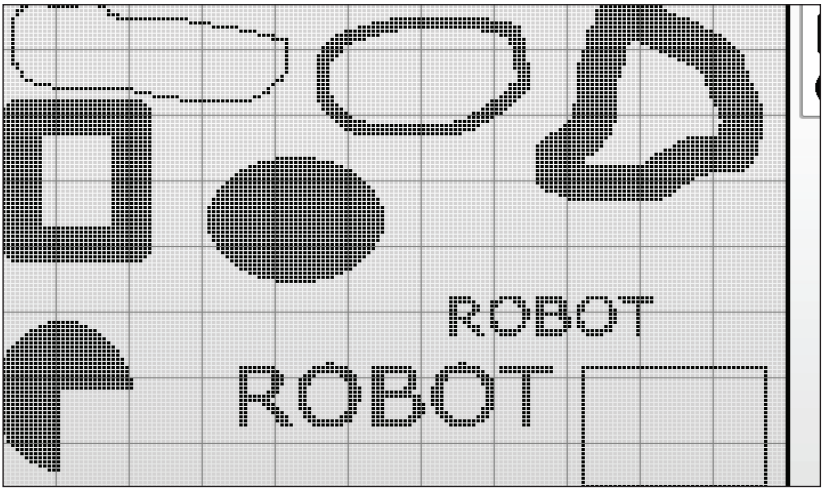
Az alakzatok vonalainak háromféle vastagsága lehet, és kétféle betűtípus közül választhatunk.

Egy előnézet ablakban megtekinthetjük, hogy az ábránk hogyan fog kinézni az EV3 tégla kijelzőjén.

A *saját blokképítő* (My Block Builder) segítségével alprogramokat, saját blokkokat hozhatunk létre és szerkeszthetünk. Elnevezhetjük, ikonnal láthatjuk el, és hozzárendelhetünk olyan paramétereket, amelyek nekünk fontosak. A saját blokkok automatikusan tárolódnak a saját blokkok palettán.

Hasznos eszköz a *Firmware frissítő* (Firmware Update). Időnként frissített firmware-k jelennek meg az EV3 téglához, javasolt ezeknek az új verzióknak a telepítése, amint azok elérhetővé válnak.

A *Vezeték nélküli beállítás* (Wireless Setup) segít abban, hogy vezeték nélküli kapcsolatot állítsunk fel a téglával. Ehhez be kell szereznünk egy Wi-Fi USB adaptert az EV3 téglához, és engedélyeznünk kell a Wi-Fi-kommunikációt a téglán.



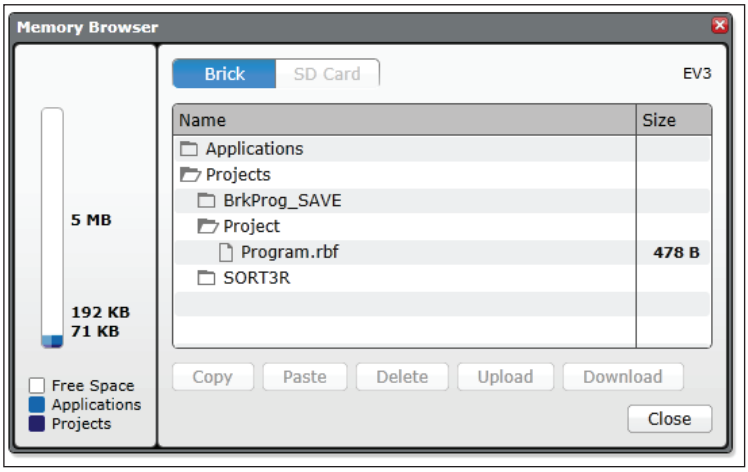
16. ábra. *Képszerkesztő*

A *Blokk importálása* (Block Import) menüpont, eszköz segítségével új blokkot adhatunk a Programfejlesztő palettához. Ez lehet egy új LEGO blokk vagy más gyártók által fejlesztett blokk is, például egy harmadik fél által gyártott érzékelőhöz. Ezeket a blokkokat először le kell töltenünk a számítógépünkre, majd ezt követően importálhatjuk őket a szoftverbe.

A *Letöltés appként* (Download as App) segítségével úgy tölthetjük le programjainkat az EV3 téglára, hogy az a *Tégla appok képernyőn* jelenjen meg az alapértelmezett alkalmazások mellett.

A *Memóriaböngésző* (Memory Browser) áttekintést ad a téglán történő memóriahasználatról (az SD-kártyát is beleértve, ha behelyeztünk egyet). Fel lehet használni programok, hang, grafikus- és egyéb fájlok áthelyezésére az EV3 téglára, és minden olyan állomány másolására és törlésére, amelyek már a téglán vannak.

A *Tégla program importálása* (Import Brick Program) eszköz lehetővé teszi, hogy az EV3 tégla *Tégla programozás app*ban készült programot beimportálhassuk az EV3 szoftverbe. Programunkat így tovább finomíthatjuk a LEGO MINDSTORMS EV3 Home Edition szoftver teljes funkcionalitásának felhasználásával.



17. ábra. *Memóriaböngésző*

3.1.4. Programblokkok

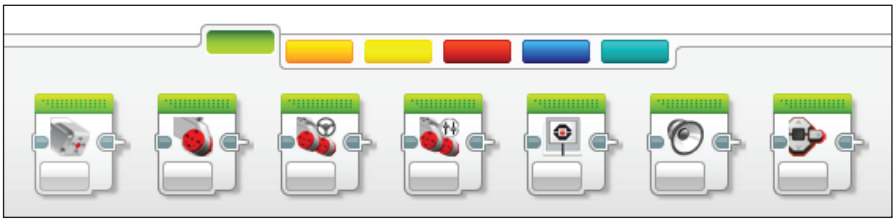
A LEGO robotok programozásához *programblokk*okat (Program Blocks) használunk. Ezek a programfejlesztői vászon (Programming Canvas) alatti palettán (Programming Palettes) vannak elhelyezve.

A blokkokat típus és jelleg alapján kategóriákba sorolták, így könnyebben megtalálhatjuk a szükséges blokkot.

A megfelelő palettáról kiválasztott blokkot az egér segítségével a programfejlesztői vászonra húzzuk, összekötjük őket a megfelelő más blokkokkal, beállítjuk a bemeneti adatait, és máris futtatható programot kapunk [15].

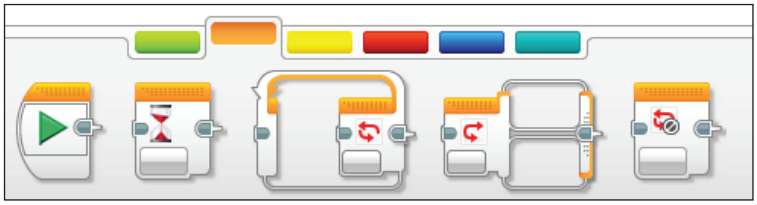
A palettán a blokkok csoportjait színek jelölik: zöld, narancs, sárga, vörös, kék, türkiz. A különböző csoportok a következő blokkokat tartalmazzák:

Zöld – Cselekvő blokkok (Action Blocks):
– Közepes motor (Medium Motor)
– Nagy motor (Large Motor)
– Kormányozás – mozgásvezérlés (Move Steering)
– Tank – mozgástank (Move Tank)
– Kijelző (Display)
– Hang (Sound)
– Tégla-állapotjelző fény (Brick Status Light)



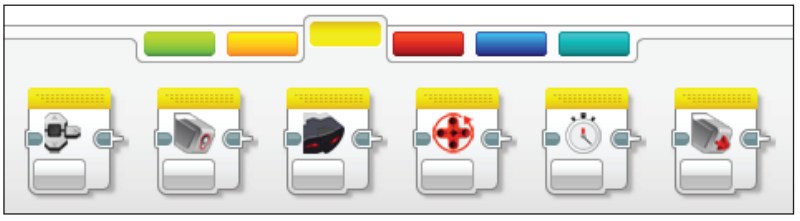
18. ábra. *Cselekvő blokkok*

Narancs – Folyamat blokkok (Flow Blocks)
– Start (Start)
– Várj (Wait)
– Hurok – ciklus (Loop)
– Kapcsoló – elágazás (Switch)
– Hurok, ciklus megszakítás (Loop Interrupt)



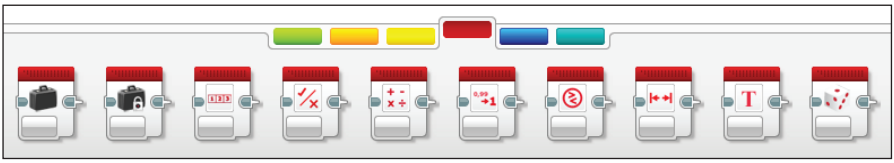
19. ábra. *Folyamat blokkok*

Sárga – Érzékelő blokkok (Sensor Blocks)
– Téglagombok (Brick Buttons)
– Színérzékelő (Color Sensor)
– Infravörös érzékelő (Infrared Sensor)
– Motorforgás (Motor Rotation)
– Időzítő (Timer)
– Érintésérzékelő (Touch Sensor)

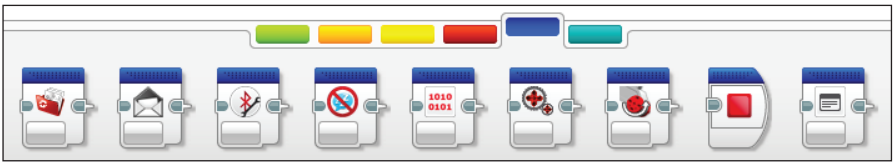


20. ábra. Érzékelő blokkok

Vörös – Adatblokkok (Data Blocks)
– Változó (Variable)
– Állandó (Constant)
– Műveletek tömbökkel (Array Operations)
– Logikai műveletek (Logic Operations)
– Matematika (Math)
– Kerekítés (Round)
– Összehasonlítás (Compare)
– Tartomány (Range)
– Szöveg (Text)
– Véletlenszerű (Random)



21. ábra. *Adatblokkok*

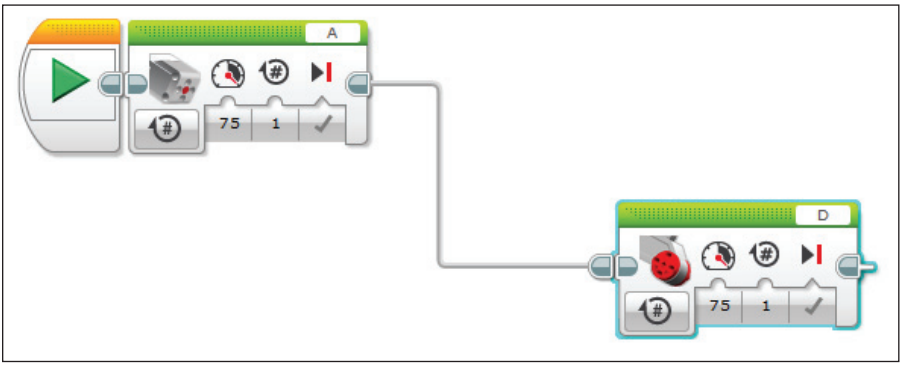


22. ábra. *Speciális blokkok*

Kék – Speciális blokkok (Advanced Blocks)
– Állomány-hozzáférés (File Access)
– Üzenetek (Messaging)
– Kapcsolat (Bluetooth Connection)
– Virrasztás (Keep Awake)
– Nyers érzékelő érték (Raw Sensor Value)
– Szabályozatlan motor (Unregulated Motor)
– Motorinvertálás (Invert Motor)
– Programleállítás (Stop Program)
– Comment (Megjegyzés)

Türkiz – Saját blokkok (My Blocks)
– Kezdetben ez a paletta üres. Ha egy program valamilyen részletét sok más programban fel szeretnénk használni, akkor létrehozhatunk egy saját

blokkot. Ez olyan, mint az eljárás vagy függvény imperatív nyelvek esetén. A létrehozott saját blokkok erre a palettára kerülnek, azután ezeket egyszerűen beszúrhatjuk a későbbi programjainkba, ugyanazon projekten belül.



23. ábra. Összeragasztott és összekötött blokkok

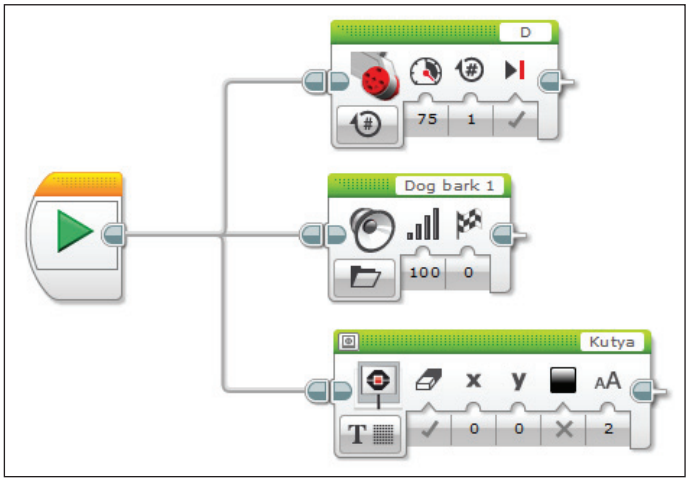
A LEGO robotok programozása úgy történik, hogy a programfejlesztői vászonra előbb felteszünk egy Start blokkot a Folyamat blokkok palettáról, majd a kívánt program létrehozása érdekében a többi blokkot. Ha egy blokkot megfogunk az egérrel és azt a palettáról a vászonra húzzuk, közel egy már meglévő blokkhoz, akkor a két ellentétes oldalon lévő fülecskék révén ezek egymáshoz ragadnak, és a második blokk az első programbeli folytatása lesz. Így egymás mellé több blokkot is feltehetünk, amikor futtatjuk a programot, a blokkok egymásután kapják meg a vezérlést úgy, ahogy a vászonra fel voltak helyezve, balról jobbra. Ez a végrehajtási sorrend.

Ha a blokkot valamivel távolabb helyezzük el az előző blokktól, akkor ezeken nem ragadnak össze, az összekötést a programozó kell megoldja úgy, hogy az első blokk jobb oldali fülecskéjéből egy drótot húz ki az egérrel, és ezt a drótot a második blokk bal oldali fülecskéjével összeköti.

Az összekötő drótokat egyszerűen letörölhetjük úgy, hogy a drót jobb oldali fülecskéjére kattintunk az egérrel.

Az egér segítségével kiválaszthatunk egy adott blokkot (vagy a SHIFT gomb lenyomásával egyszerre többet is), ekkor a blokk körül egy világoskék keret jelenik meg. A kiválasztott blokkot áthelyezhetjük, vagy akár le is törölhetjük.

A folyamatvezérlő blokkok (ciklus, elágazás) négy sarkában négy kis köröcske, az oldalak közepén pedig négy kis négyzet jelenik meg, ezek segítségével tetszőlegesen át tudjuk méretezni a blokkot.



24. ábra. *Párhuzamos blokkok*

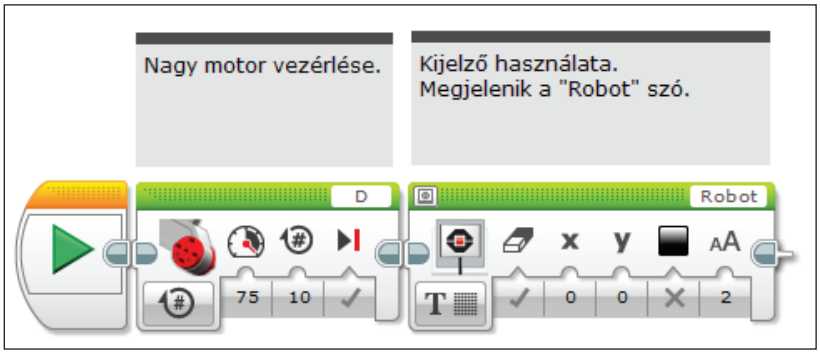
A robotok programozása párhuzamosan is történhet. Ha egy blokk után két vagy több blokkot teszünk fel egymás alá, és ezeket az elsővel úgy kötjük össze, hogy az elsőből kihúzott drótból ágazik el a vezérlés, ezek az utóbbi blokkok párhuzamosan fognak végrehajtódni.

A 24. ábrán látható példában egy nagy motort vezérlünk, közben a hangfalon egy kutyaugatást játszunk le, valamint a kijelzőre kiírjuk a „Kutya” feliratot. E három egymás alatti blokk párhuzamosan fog végrehajtódni.

A párhuzamos programozásnál vigyázzunk az erőforrások megfelelő használatára, hisz könnyű értelmetlen parancsokat kiadni a robotnak! Például ha párhuzamosra állított két blokk segítségével ugyanazt a motort próbáljuk irányítani úgy, hogy az egyik blokkon 10-szer jobbra forgatjuk, a másik párhuzamos blokkon pedig 10-szer balra forgatjuk, akkor nyilvánvaló, hogy a motor működtetésében komoly konfliktushelyzet áll elő.

A vászont, s így a rajta lévő blokkokat nagyíthatjuk, kicsinyíthetjük a vászon fölötti eszközsáv jobb oldali gombjaival, mozgathatjuk ezeket a kéz ikonú gomb segítségével, illetve megjegyzéseket is írhatunk a vászonra. A megjegyzések szövegdobozát tetszőlegesen át lehet méretezni. Ezek nagyon hasznosak lehetnek a program működésének leírására, megértésére.

A programozás során használhatjuk a LEGO MINDSTORMS EV3 Home Edition szoftver gyorsbillentyűit is. Ezeket a gyorsbillentyűket a 9. táblázat foglalja össze.



25. ábra. *Megjegyzések*

9. táblázat. *Gyorsbillentyűk*

Windows	Mac	Eredmény
CTRL+A	Command-A	Mindent kiválaszt
CTRL+B	Command-B	Leállítja az EV3-at
CTRL+C	Command-C	Másolás
CTRL+D	Command-D	Letöltés az EV3-ra
CTRL+H	Command-H	Kontextusfüggő súgó
CTRL+F	Command-F	Képernyő lementése
CTRL+I	Command-I	EV3 memórianavigátor
CTRL+M	Command-M	Hardver oldal ki/be-kapcsolása
CTRL+N	Command-N	Új program
CTRL+E	Command-E	Új kísérlet
CTRL+O	Command-O	Megnyitás…
CTRL+P	Command-P	Nyomtatás
CTRL+Q	Command-Q	Kilépés
CTRL+R	Command-R	Letöltés és futtatás
CTRL+S	Command-S	Mentés
CTRL+Shift+S	Command-Shift-S	Mentés másként…
CTRL+T	Command-T	Előrejelzés
CTRL+U	Command-U	Letöltés az EV3-ról
CTRL+V	Command-V	Beillesztés
CTRL+W	Command-W	Fül bezárása
CTRL+Shift+W	Command-Shift-W	Projekt bezárása
CTRL+X	Command-X	Kivágás

Windows	Mac	Eredmény
CTRL+Y	Command-Y	Helyrehoz
CTRL+Z	Command-Z	Visszavonás
CTRL+G	Command-G	Eszközök közötti váltás
CTRL+Shift+H	Command-Shift-H	Tevékenység elrejtése/megjelenítése
CTRL+Shift+P	Command-Shift-P	Pontelemzés
CTRL+Shift+A	Command-Shift-A	Szekcióelemzés
F1	Command-Option-?	Súgó
1	1	Cselekvőpaletta
2	2	Folyamatpaletta
3	3	Érzékelőpaletta
4	4	Adatpaletta
5	5	Speciális paletta
6	6	Saját blokkok paletta
Left arrow	Left arrow	Balra vivés
Right arrow	Right arrow	Jobbra vivés
Alt+Drag	Alt-Drag	A program mozgatása, átméretezése
CTRL+J	Command-J	Új megjegyzés

3.1.5. Adattípusok

A LEGO MINDSTORMS EV3 Home Edition grafikus programozási nyelv adattípusokat használ a blokkok adatainak ábrázolásához. Ezek a következők:

– Numerikus (Numeric)
– Logikai (Logic)
– Szöveg (Text)
– Numerikus tömb (Numeric Array)
– Logikai tömb (Logic Array)

A *numerikus* adattípus negatív vagy pozitív egész, illetve valós számokat fed. Például: –2, –1.54, 0, 56, 516.2356.

A *logikai* típus egy Igaz (True) vagy Hamis (False) értéket ábrázol.

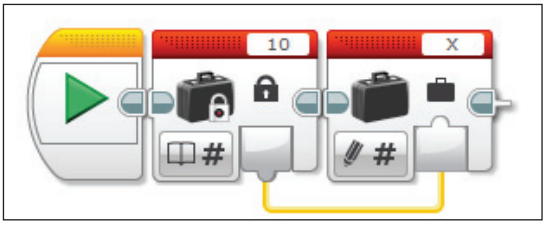
26. ábra. *A Simple Text karakterei*

A *szöveg* típus egy karaktersorozatot (karakterláncot) ábrázol. Az egyes karakterek az úgynevezett *egyszerűsített szöveg* (Simple Text) karakterei lehetnek. Ezeket a 26. ábrán mutatjuk be. Más karaktereket nem tud ábrázolni az EV3 tégla. A karakterek segítségével angol, orosz, egyszerűsített kínai, illetve japán szövegeket tud megjeleníteni, ábrázolni. Természetesen a szöveg szóközöket is tartalmazhat, így nemcsak szavakat, hanem mondatokat is képezhetünk. Például: „Udvozollek a Sapientian!”

A *numerikus tömb* egy negatív vagy pozitív egész, illetve valós számokból álló listát jelent. A listának meghatározott hossza van, amelynek csak az EV3 tégla memóriája szab határt. A lista minden egyes eleme egy numerikus érték, amelyet a megadott sorrendben tárol a rendszer. Mivel a lista nem halmaz, ezért egy érték többször is szerepelhet benne. A lista elemeit pontosvesszővel („;”) választjuk el egymástól, és az egész listát szögletes zárójelek közé tesszük („[]”). Például: [0; –0.25; 345.25; 7; 7; 7]. Az üres tömböt []-el jelöljük, ennek a hossza: 0.

A *logikai tömb* a numerikus tömbhöz hasonló adattípus, azzal a különbséggel, hogy ennek az elemei csak az Igaz (True) vagy Hamis (False) értékek lehetnek.

Az adattípusokhoz szorosan kötődnek az *adatdrót*ok (Data Wire), amelyek segítségével a blokkok kimeneteleit köthetjük a bemenetelekhez, így átadva egymásnak a megfelelő adatokat. A blokkok között így interakció jöhet létre, az adatfolyamok révén pedig összetettebb programok valósíthatók meg, ezáltal a robot viselkedése is komplexebb lehet.



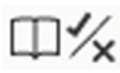
27. ábra. *Adatdrót*

A 27. ábra egy numerikus adatdrótot ábrázol. Az X változó (bemeneti adat) felveszi a konstans 10-es értékét (kimeneti adat). Ez megfelel az X = 10 értékadásnak imperatív programozási nyelvek esetén.

A kimeneti adatot tartalmazó blokk meg kell hogy előzze a bemenetet tartalmazó blokkot.

A 10. táblázat a különböző típusú adatdrótokat és az adattípusok grafikai szimbólumait mutatja be.

10. táblázat. *Adatdrótok és az adattípusok szimbólumai*

Adattípus	Be	Ki	Drót	Jel írásra (bemenet)	Jel olvasásra (kimenet)
numerikus					
logikai					
szöveg					
numerikus tömb					
logikai tömb					

Az adatdrótokat egyszerű „fogd és vidd” (drag and drop) technikával lehet a grafikus felületen kialakítani. Ha az egérrel valamely blokk kimeneti adata fölé megyünk, akkor az egérmutató (kurzor) átvált egy dróttekercset ábrázoló kurzorrá, majd kattintva és megfogva, áthúzhatjuk a drótot egy másik blokk bemenetére.

Egy kimenet egyszerre több blokk bemenete is lehet.

Az adatdrótokat szintén „fogd és vidd” (drag and drop) technikával lehet letörölni: a bemenetről kell lehúzni az adatdrót végét.

A 11. táblázat alapján az egyes adattípusok között automatikus és adatveszteség nélküli konverzió hajtódik végre.

11. táblázat. *Adatkonverziók*

Típusról	Típusra	Eredmény
logikai	numerikus	hamis (false) = 0 igaz (true) = 1
logikai	szöveg	hamis (false) = “0” igaz (true) = “1”
logikai	logikai tömb	egy egyelemű tömb
logikai	numerikus tömb	egy egyelemű tömb (0 vagy 1)
numerikus	szöveg	a szám szöveges ábrázolása (például: “23.65”)
numerikus	numerikus tömb	egy egyelemű tömb
logikai tömb	numerikus tömb	ugyanolyan hosszúságú numerikus tömb 0 vagy 1 elemekkel

Ha az EV3 téglát a számítógéphez csatlakoztatjuk (USB, Bluetooth vagy Wi-Fi), és futtatjuk a programot, az aktívan futó blokk valamely drótja fölé húzva az egeret, egy ablakban megjelenik a dróton lévő érték, vagyis a ki/bemeneti adat. Így könnyen nyomon követhetjük és ellenőrizhetjük adatainkat, s ezáltal a teljes program adatfolyamát, működését.

Ha a portokat adatdrót segítségével adjuk meg, akkor ezek numerikus értékként fognak szerepelni a következőképp:

– A port értéke: 1
– B port értéke: 2
– C port értéke: 3
– D port értéke: 4
– 1 port értéke: 1
– 2 port értéke: 2
– 3 port értéke: 3
– 4 port értéke: 4

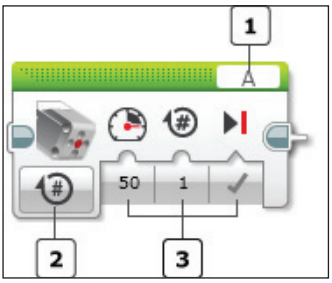
Ha a kormányozási vagy tank üzemmódot választjuk, akkor a két port numerikus értékei:

– B, C: 23
– C, B: 32
– A, B: 12
– A, D: 14

Ha lánckapcsolással kötjük össze az EV3 téglákat, akkor az első tégla portjainak numerikus értékéhez 100-at kell hozzáadni (pl. 101, 102, 103, 104, 123, 132, 112, 114), a második tégla portjainak numerikus értékeihez 200-at, a harmadiknak 300-at, a negyediknek pedig 400-at (pl. 403 a negyedik tégla C portja).

3.1.6. A közepes motor programozása

A közepes motor programozására egy blokk van fenntartva a kezelőfelület zöld eszköztárán. A 28. ábrán látható blokk beállításai a következők:



28. ábra. *A közepes motor blokkja*

1: Az 1-es gomb segítségével a portot választhatjuk ki (port selector). Ezen a porton keresztül fog kommunikálni az EV3 tégla a motorral, itt küldi át a parancsokat. A port az A, B, C vagy D valamelyike lehet.

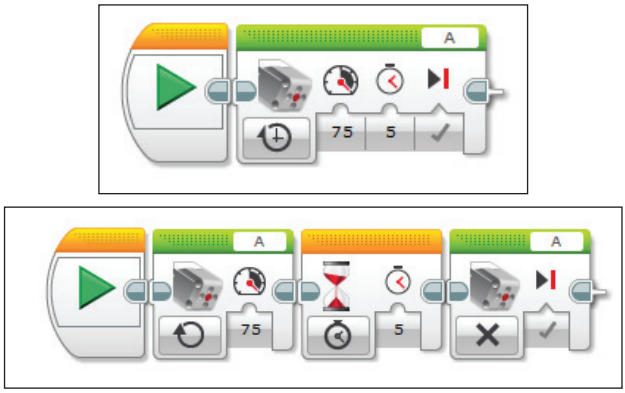
2: A 2-es gomb segítségével egy legördülő menüből kiválaszthatjuk a motor működési módját (mode selector): *Off* (kikapcsol), *On* (bekapcsol), *On for Seconds* (bekapcsol időre), *On for Degrees* (bekapcsol fokokra), *On for Rotations* (bekapcsol fordulatszámra).

3: A 3-as gomb vagy gombok segítségével a bemeneti adatokat adhatjuk meg. Ezek száma, mértékegysége és mérete a módoktól függ.

Az *On* mód bekapcsolja a motort, és ez addig fog működni, míg egy *Off* módú blokk ki nem kapcsolja. A vezérlés a bekapcsolás után azonnal átadódik a következő blokknak. Az egyedüli beállítható bemeneti adat a motor sebessége/ereje, amely egy –100 és 100 közötti érték. Az előjel a forgás irányát jelenti (hátra/előre). A pozitív irány az óramutató járásával megegyező, a negatív irány pedig az óramutató járásával ellentétes.

Az *Off* mód leállít egy előzőleg bekapcsolt motort. Az egyedüli beállítható bemeneti adat a *Brake at End* (leállás a végén). Ez egy logikai érték: *Igaz* (*True*) vagy *Hamis* (*False*) lehet. Az *Igaz* érték azt jelenti, hogy a motor azonnal leáll, és úgy marad abban a pozícióban (*Brake*), a *Hamis* érték pedig azt jelenti, hogy a motor áramellátása kikapcsol, és az addig forog tehetetlenségből szabadon, amíg meg nem áll (*Coast*).

Az *On for Seconds* (bekapcsol időre) mód a megadott ideig működteti a motort. Be lehet állítani a motor erejét, másodpercekben kifejezve (lehet valós szám is) a működés időtartamát, valamint a *Brake at End*et.



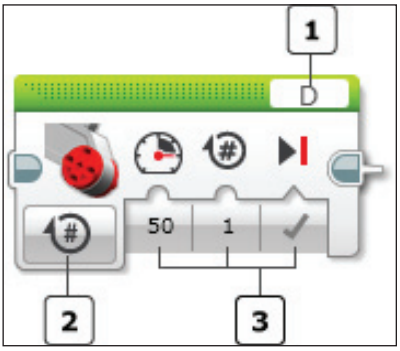
29. ábra. a) *Közepes motor vezérlése egy blokkal*; b) *Közepes motor vezérlése több blokkal*

Az *On for Degrees* (bekapcsol fokokra) mód a megadott fokig forgatja a motort. Be lehet állítani a motor erejét, fokokban kifejezve (lehet valós szám is) a fordulatot (egy teljes fordulat 360°), valamint a *Brake at End*et. A motor belső érzékelője pontosan méri a fokokat, és a motor végrehajtja a megfelelő fordulatot. Ez azt jelenti például, ha működés közben egy akadály nem engedi tovább forogni a motort, ez addig fog várni, míg az akadály elhárul, és pontosan végrehajtódik a megadott foknyi fordulat. Ameddig akadályozva van, a robot programja leáll, a vezérlés nem adódik át a következő blokknak.

Az *On for Rotations* (bekapcsol fordulatszámra) mód a megadott fordulatszámig forgatja a motort. Be lehet állítani a motor erejét, numerikus értékben kifejezve (lehet valós szám is) a fordulatszámot (egy teljes fordulat 1-es, két és fél fordulat: 2,5 stb.), valamint a *Brake at End*et. Ez a mód teljesen megfelel az *On for Degrees* módnak, annyi különbséggel, hogy más a mértékegység: 1 fordulatszám 360°, 2 fordulatszám 720°, fél fordulatszám 180°, 1,25 fordulatszám 450° stb.

A 29. ábra a) és b) programjai ugyanazt csinálják, csak különféleképpen oldják meg a közepes motor vezérlését. Az a) ábrán egy motorvezérlő blokkot láthatunk, amely *On for Seconds* (bekapcsol időre) módban 5 másodpercig működteti 75%-os erővel az A portra kapcsolt motort, mégpedig úgy, hogy azonnal leáll és megtartja a pozícióját. A b) ábrán pedig elindítjuk az A portra kapcsolt motort 75%-os erővel az *On* mód segítségével, majd egy *Wait* (várakozás) blokk segítségével 5 másodpercet várunk (ez idő alatt működik a motor), és végül az *Off* mód segítségével leállítjuk a motort, úgy, hogy az megtartja pozícióját.

3.1.7. A nagy motor programozása



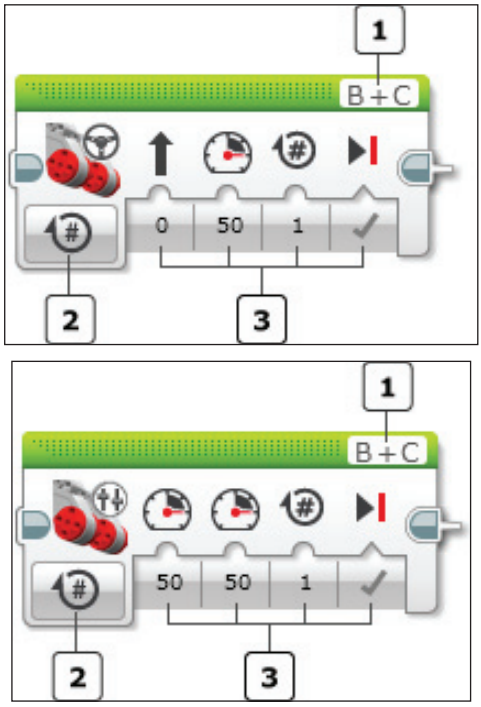
30. ábra. *A nagy motor blokkja*

Blokk szintjén a nagy motor programozása teljesen megegyezik a közepes motor programozásával. Az ott leírtak érvényesek a nagy motor blokkjára is.

Mindazonáltal a nagy motorok annyiban különböznek a közepes motortól, hogy ezeket párba is lehet kötni (például a B + C portokat használva). A párban működtetésre két új blokkot használhatunk: a *Move Steering* (kormányozás blokk), illetve a *Move Tank* (tank blokk) blokkokat.

Az 1-es, 2-es gombok használata teljesen megegyezik a közepes motornál bemutatottakkal, a 3-as gombok (bemeneti adatok) változnak csak némiképp, egész pontosan mindkét blokk esetén bejön egy plusz bemeneti adat.

A kormányozás blokk segítségével a robot előre, hátra, fordulásra vagy megállásra programozható. Beállíthatjuk a kormányozás milyenségét, hogy a robot egyenesen, hajló ívek vagy szűk fordulások mentén kanyarodjon.



31. ábra. a) *Move Steering*; b) *Move Tank*

A kormányozás blokk azt feltételezi, hogy a robot két nagy motorral van felszerelve, egyik motor a jármű (robot) bal oldalán, a másik a jobb oldalon. A kormányozás blokk egyszerre vezérli mindkét motort, s így tudjuk vezetni (kanyarítani) a járművet abba az irányba, amit választottunk.

Az irány miatt mindig vigyázzunk, hogy a bal oldali motor portja legyen beállítva először. Például ha B + C van kiválasztva a portnál, akkor a bal oldali motor legyen a B portra, a jobb oldali motor pedig a C portra kötve.

A kormányozás blokk esetén az első bemeneti adat a kanyarodás mértéke és iránya egy –100 és 100 közötti értékkel van megadva, amely százalékban fejezi ki a kanyarodás mértékét: 0 azt jelenti, hogy egyenesen előre halad, 50 azt, hogy 90°-ban kanyarodik jobbra, –50 azt, hogy 90°-ban kanyarodik balra stb.

Figyeljünk arra, hogy az *On for Degrees* (bekapcsol fokokra) vagy az *On for Rotations* (bekapcsol fordulatszámra) mód esetén a robot által megtett távolság most már nemcsak a beállított foktól vagy fordulatszámtól függ, hanem a kanyarodás ívétől is. Ebben az esetben a beállított értékek mindig arra a motorra értendők, amelyik gyorsabban fordul.

A tank blokk nagyon hasonlít a kormányozás blokkhoz, csak azt feltételezi, hogy a robot lánctalpakon jár, így a fordulatot nem a kanyarodás mértékével lehet megadni, hanem a két motor erejével. Ha ez egyik motornak nagyobb az ereje, mint a másiknak, a jármű elfordul. Így a blokkon két erőbeállítási lehetőség (gomb) van, az egyik a bal oldali motor, a másik pedig a jobb oldali motor erejét állítja be. Például ha a bal oldali motor ereje 100, a jobb oldalié pedig 50, a jármű egy ívben jobbra fog fordulni. A jármű azonnal megfordul, ha például a bal oldali motor erejét 50-re, a jobb oldaliét pedig –50-re állítjuk.

Természetesen a fordulás, kanyarodás íve mindkét esetben függ a kerekek méretétől vagy a kerekek közötti távolságtól, és más faktoroktól is.

3.1.8. A kijelző programozása

A kijelző programozása a *kijelző blokk* (Display Block) segítségével történik. Így fehér-fekete grafikát vagy szöveget jelentethetünk meg az EV3 tégla kijelzőjén.

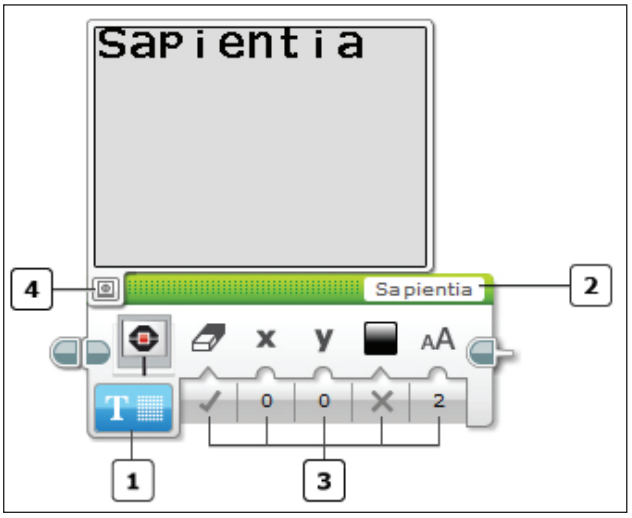
A 32. ábrán látható kijelző blokk részei:

– 1. a blokk módjának kiválasztó gombja (mode selector)
– 2. szövegdoboz
– 3. bemeneti adatok
– 4. kijelzőmegjelenítő gomb

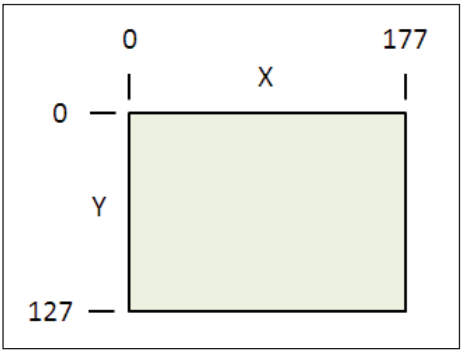
Az 1-es gomb segítségével választhatjuk ki, hogy a kijelzőn szöveget (pixeles vagy rácsos), alakzatot (vonal, kör, téglalap, pont) vagy valamilyen beolvasott képet kívánunk megjeleníteni, illetve itt lehet eredeti állapotába visszaállítani a kijelzőt (Reset Screen).

Ha az 1-es gomb segítségével szöveget választunk, akkor a 2-es szövegdobozba írhatjuk be a megjelenítésre szánt szöveget, vagy itt kiválaszthatjuk azt is, hogy a szöveg egy adatdrót segítségével legyen megadva bemenetként (Wired). Ha az 1-es gomb által egy állományt választunk ki, akkor szintén itt, a 2-es szö-

vegdobozban adhatjuk meg az állomány nevét, vagy választhatunk a LEGO által eleve megadott képek közül.



32. ábra. *A kijelző blokk*



33. ábra. *A kijelző koordináta-rendszere*

Érdekes a 4-es kijelzőmegjelenítő gomb. Amennyiben elkészítettük a grafikánkat vagy szövegünket, ezzel a gombbal egy ablakot hívhatunk elő, amely ugyanúgy tartalmazza a megjelenítendő grafikát vagy szöveget, mint az EV3-as tégla kijelzője. Itt tehát előre láthatunk mindent.

A 3-as gombok segítségével a bemeneti adatokat adhatjuk meg. Természetesen ezek a kiválasztott módtól függnek.

Ha a pixeles szöveg (Pixel) módot választjuk ki, akkor bemeneti adatként a következőket adhatjuk meg:

– A képernyő letörlése (Clear Screen): logikai érték, Igaz vagy Hamis lehet. Ha Igaz, a szöveg megjelenítése előtt a rendszer letörli a képernyőt.
– X és Y: a szöveg kezdetének (bal felső sarok) koordinátái.
– Szín (Color): logikai érték. Ha Igaz, akkor a szöveg fekete alapon fehérrel jelenik meg, ha Hamis, akkor a szöveg fehér alapon feketével jelenik meg.
– Betűtípus (Font): 0, 1 vagy 2 lehet. A 0 normál (Normal) betűtípust, az 1-es félkövér (Bold), a 2-es nagy (Large) betűket jelent.

Ha a rácsos szöveg (Grid) módot választjuk ki, akkor a fentiek annyiban módosulnak, hogy az X és az Y rácspontokat kell megadni, és a rendszer ezekhez a rácspontokhoz igazítja a szöveget. Így a szöveget sorokban és oszlopokban jeleníthetjük meg. Egy oszlop szélessége megegyezik egy karakter szélességével normál és félkövér betűtípus esetén, ez 8 pixelt jelent. A nagy betűtípus pedig kétszer akkora, mint a félkövér (16×16 pixel). Mivel a normál betűtípus 9 pixel magas, a félkövér pedig 8, a sorok mérete 10 pixel. A kijelzőn tehát 12 (0-tól 11-ig) sor és 22 (0-tól 21-ig) oszlop található.

1. feladat

A LEGO robot kijelzőjén valósítsunk meg egy mozgó feliratot!

Írjuk ki az „Udvozollek a Sapientian!” *szöveget úgy, hogy ez jobbról balra gördüljön az EV3 tégla kijelzőjén.*

Nyilvánvaló, hogy ehhez egy ciklus blokkot és matematikai műveleteket megvalósító blokkot is kell használni, de bevezetésként ezeknek a nagyon egyszerű üzemmódját használjuk, így fokozatosan megismerhetjük, megszokhatjuk ezeket a lehetőségeket.

34. ábra. *A mozgó* „Udvozollek a Sapientian!” *felirat*

Ha az alakzatok – vonal módot választjuk, akkor bemenetként, a képernyőtörlés és szín mellett megadhatjuk a vonal bal felső és jobb alsó pontjainak a koordinátáit: X1, Y1, X2, Y2. Egy fehér vonal csak akkor látható, ha nem töröljük le a képernyőt, és az előzőleg feketére volt festve.

Az alakzatok – kör esetén a kör középpontjának X és Y koordinátáit, illetve a kör sugarát kell megadjuk, valamint azt is, hogy a kör legyen-e kitöltve vagy sem.

Az alakzatok – téglalap esetén a téglalap bal felső sarkának az X, Y koordinátáit, valamint a téglalap magasságát és szélességét kell megadjuk bemeneti adatként. Ha a kijelzőnek csak egy részét szeretnénk letörölni, akkor arra a területre rajzolhatunk egy fehér téglalapot.

Az alakzatok – pont egy pixelt rajzol ki a kijelzőre a megadott X, Y koordinátákra.

A kép mód segítségével egy fehér-fekete képállományt jeleníthetünk meg. A kép bal felső sarkának koordinátái a megadott X, Y pontokban lesznek.

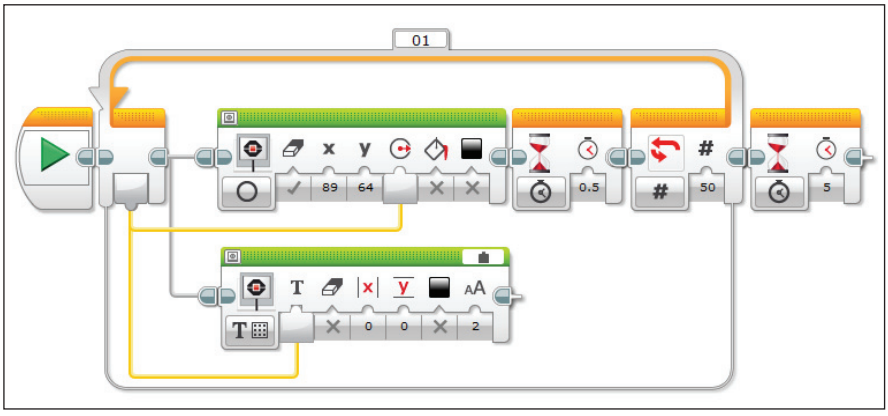
A kijelző visszaállítása mód visszaállítja az EV3 tégla kijelzőjét a normál állapotba, vagyis a program futásáról jelennek meg az információk.

Hogy a megjelenített grafika vagy szöveg ne tűnjön el hamar a képernyőről, a megjelenítés végén használjunk egy várakozás blokkot (Wait).

Ha bármilyen numerikus adatot akarunk megjeleníteni, például egy érzékelő kimeneti értékét, egyszerűen az adatdrót segítségével kössük össze a kimenetet a kijelző blokk szöveges bemenetével. A program automatikusan átalakítja a numerikus értéket szöveggé, és ez megjelenik a kijelzőn.

2. feladat

A 35. ábrán lévő program egy kört rajzol ki a kijelző középpontjába úgy, és animál úgy, hogy a sugara növekszik 0-tól 50-ig. A kör kirajzolása előtt letörli a rendszer a képernyőt. Ezzel párhuzamosan a kör sugarát kiírja a kijelző bal felső sarkába (ekkor már nincs képernyőtörlés). Minden körrajzolás után a rendszer fél másodpercet várakozik, a végén pedig 5 másodperc múlva állítja vissza az eredeti képernyőt.

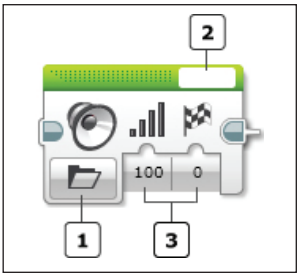


35. ábra. *Növekvő kör és a sugár megjelenítése*

3.1.9. A hangfal programozása

A *hangfal blokk* (Sound Block) segítségével szólaltathatjuk meg az EV3 tégla beépített hangfalát.

A 36. ábrán látható hangfal blokk részei:
– 1. a blokk módjának kiválasztó gombja (mode selector)
– 2. szövegdoboz az állománynevek számára
– 3. bemeneti adatok



36. ábra. *A hangfal blokk*

Az 1-es gomb segítségével választhatjuk ki, hogy a hangfalon a rendszer egy állományt játsszon le, egy hangot szólaltasson meg vagy egy hangjegyet, illetve leállítsa a hangfalat.

Amennyiben a kiválasztott mód egy hangállomány lejátszása, akkor a 2-es szövegdobozban adhatjuk meg az állomány nevét, vagy választhatunk a LEGO által eleve megadott hangok közül. Szintén itt választhatjuk ki azt is, hogy a név egy adatdrót segítségével legyen megadva bemenetként (Wired).

A 3-as gombok segítségével a bemeneti adatokat adhatjuk meg mindegyik mód esetén.

Állomány lejátszása esetén a hangállományt a projekthez kell adni, majd az állomány nevén kívül bemeneti adatként a hangerősséget adhatjuk meg egy 0 és 100 közötti szám segítségével, illetve a lejátszási módot, amely 0, 1 vagy 2 lehet. A 0 azt jelenti, hogy a rendszer megvárja a hangállomány lejátszását, és csak azután adódik át a vezérlés a következő blokknak (Wait for Completion), az 1-es esetében a hangállomány lejátszása megkezdésének pillanatában a vezérlés már át is adódik a következő blokknak, és a rendszer egyszer játssza le az állományt (Play Once). A 2-es esetben a hangállományt addig ismétli a rendszer, míg egy másik hangfal blokk le nem állítja (Repeat). A vezérlés ebben az esetben is azonnal átadódik a következő blokknak.

A hang lejátszása esetén Hz-ben adhatjuk meg a hang frekvenciáját 250-től 10 000-ig, vagy kiválaszthatjuk ezt a szabványos hangok listájából: C 261.63, D

293.67, A 440 stb. A második bemeneti adat a hang hosszát jelenti másodpercekben. Harmadik bemeneti adat a hangerő, negyedik pedig az előbb ismertetett lejátszási mód.

A hangjegy lejátszása nagyon hasonlít a hang lejátszására, azzal a különbséggel, hogy itt első paraméterként egy háromoktávos zongora szabványos hangjegyeit adhatjuk meg.

A paraméter nélküli Stop mód leállít bármiféle hanglejátszást.

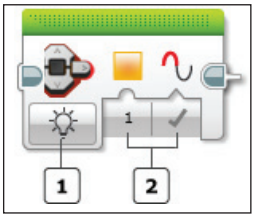
A 12. táblázat a hangjegyeket és ezek frekvenciáját foglalja össze.

12. táblázat. *Hangjegyek*

Betűjel	**Hangjegy**	**Frekvencia / oktáv**			
		4	5	6	7
C	dó	261,63	523,25	1046,5	2093
C#	di	277,18	554,37	1108,73	
D	ré	293,67	587,33	1174,66	
D#	ri	311,13	622,25	1244,51	
E	mi	329,63	659,26	1318,51	
F	fá	349,23	698,46	1396,91	
F#	fi	369,99	739,99	1479,88	
G	szó	392	783,99	1567,98	
G#	szi	415,31	830,61	1661,22	
A	lá	440,0	880	1760	
A#	li	466,16	932,33	1864,66	
H (B)	ti	493,88	987,77	1975,53	

3.1.10. A tégla állapotát jelző fények programozása

Az EV3 tégla állapotát zöld, narancsszínű és vörös villogó vagy folytonos fények jelezhetik. Ezeket a 37. ábrán látható blokkal programozhatjuk, ahol 1. a módválasztó, 2. pedig a bemeneti adatok gombjai.



37. ábra. *Az állapotjelző fények*

Az 1-es mód *Off* (kikapcsolt), *On* (bekapcsolt) vagy *Reset* (visszaállított) lehet. A mód függvényében változnak a bemeneti adatok.

A bekapcsolt mód két adatot kíván: a fények színét, ez 0, 1 vagy 2 lehet. A 0 a zöldet, az 1 a narancsszínűt, a 2 a vöröset jelenti, valamint a villogást jelentő logikai Igaz vagy Hamis értéket. Az állapotjelző fények bekapcsolt állapotban maradnak mindaddig, ameddig nem kapcsoljuk ki őket egy adatokat nem igénylő Off móddal, vagy amíg a program be nem fejeződik.

A visszaállítás mód szintén nem igényel bemenő adatokat, és az állapotjelző fényeket visszaállítja a szabványos program futását jelző villogó zöld állapotjelzésre. Ez a villogás valamivel másabb, mint a programozott zöld villogás.

3.1.11. Az érintésérzékelő programozása

Az érzékelők bemeneti adatokat szolgáltatnak. Az érintésérzékelő a legegyszerűbb érzékelő, tulajdonképpen egy nyomógomb, amelynek három állapota lehet: *benyomott* (Pressed), *felengedett* (Relased) és ütközött (Bumped).

Az érintésérzékelőtől tudhatjuk meg, ha a robotunk valamivel ütközött, de a gomb benyomásával vagy felengedésével különböző cselekvéseket is kiválthatunk. Az érintésérzékelő nem tud adatokat szolgáltatni arról, hogy például a gomb mennyi ideig volt benyomva, vagy milyen erősen volt megnyomva, csak arról, hogy be van-e nyoma vagy sem, illetve az ütközött állapot azt jelzi, hogy a gomb be volt-e nyomva és fel volt-e engedve a közelmúltban, tehát a robot ütközött-e már valamivel. Az ütközött állapot egyszerűen megmondja azt, hogy az érintésérzékelőt nyomógombként használtuk-e vagy sem, vagyis nem kell lekérdezni egymás után azt, hogy le van-e nyomva a gomb, majd fel van-e engedve.

A 13. táblázat azt mutatja be, hogy 7 lépésen keresztül milyen tevékenységekre milyen logikai értéket szolgáltat vissza az érintésérzékelő. Az érintésérzékelő felengedett állapotban lévő gombját kétszer egymás után lenyomjuk, és minden lépés után a program kiolvassa az érintésérzékelő által visszatérített logikai értéket.

Az érintésérzékelő blokk segítségével leolvashatjuk az érintésérzékelőről kapott adatokat, amelyek az érintkező fizikai állapotát tükrözik.

A 38. ábrán látható blokkon az 1-es gomb segítségével a portot választhatjuk ki (port selector). Ezen a porton keresztül fog kommunikálni az EV3 tégla az érzékelővel, innen olvassa be az adatokat. A port az 1, 2, 3 vagy 4 valamelyike lehet. Alapértelmezett portja az 1-es.

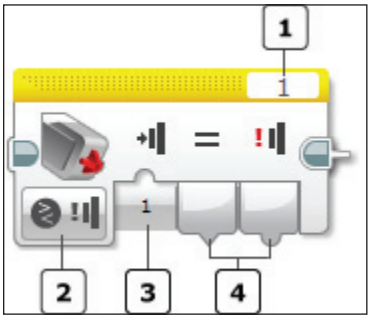
A 2-es gomb segítségével egy legördülő menüből kiválaszthatjuk az érzékelő működési módját (mode selector), ez a *Measure* (mérés) vagy *Compare* (összehasonlítás) lehet.

13. táblázat. *Az érintésérzékelő állapotainak változása*

Lépés	Tevékenység	Benyomott	Felengedett	Ütközött
1	alapállapot: az érintés-érzékelő gombja nincs benyomva	hamis	igaz	hamis
2	benyomjuk a gombot (nyomva van, míg a program beolvassa az adatokat)	igaz	hamis	hamis
3	felengedjük a gombot	hamis	igaz	igaz
4	a gomb fel van engedve, és a program olvassa az érzékelő adatait	hamis	igaz	hamis
5	másodszorra is benyomjuk a gombot	igaz	hamis	hamis
6	felengedjük a gombot	hamis	igaz	igaz
7	a gomb fel van engedve, és a program olvassa az érzékelő adatait	hamis	igaz	hamis

A mérés üzemmódban a visszatérítési értéke egy logikai érték, amely Igaz (True) vagy Hamis (False) lehet. Ezt a 4-es gombon szolgáltatott visszatérítési értéket adatdrót segítségével adhatjuk át más blokknak.

A visszatérési érték Igaz, ha az érintésérzékelő be van nyomva, különben Hamis.



38. ábra. Érintésérzékelő

Az összehasonlítás üzemmódban a 3-as gomb segítségével bemeneti adatként megadhatjuk, hogy a *benyomott* (Pressed), *felengedett* (Relased) vagy ütközött (Bumped) állapotot szeretnénk-e lemérni. A menü segítségével egy numerikus értéket állíthatunk be: 0 – felengedett, 1 – benyomott, 2 – ütközött.

A blokk két értéket térít vissza a 4-es gombon. Az első egy logikai érték, amely megmutatja, hogy az elvárt állapot következett-e be. A második pedig maga a szenzor állapota az előbbi kódolás szerint, numerikus értékként.

A 14. táblázat tevékenységi lépésenként mutatja be, hogy az érintésérzékelő összehasonlítás üzemmódban milyen bemeneti értékre milyen visszatérési értékeket szolgáltat.

14. táblázat. *Az érintésérzékelő összehasonlítás üzemmódban*

Lépés	Tevékenység	Bemenet	Első visszatérési érték	Második visszatérési érték
1	Alapállapot: az érintésérzékelő gombja nincs benyomva.	0	igaz	0
2	Benyomjuk a gombot (nyomva van, míg a program beolvassa az adatokat).	0	hamis	1
3	Felengedjük a gombot.	0	hamis	2
4	A gomb fel van engedve, és a program olvassa az érzékelő adatait.	0	igaz	0
5	Gyorsan benyomjuk és felengedjük a gombot (közben a program nem olvas, csak a felengedés után).	0	hamis	2
6	Alapállapot: az érintésérzékelő gombja nincs benyomva.	1	hamis	0
7	Benyomjuk a gombot (nyomva van, míg a program beolvassa az adatokat).	1	igaz	1
8	Felengedjük a gombot.	1	hamis	2
9	A gomb fel van engedve, és a program olvassa az érzékelő adatait.	1	hamis	0
10	Gyorsan benyomjuk és felengedjük a gombot (közben a program nem olvas, csak a felengedés után).	1	hamis	2
11	Alapállapot: az érintésérzékelő gombja nincs benyomva.	2	hamis	0

Lépés	Tevékenység	Bemenet	Első visszatérési érték	Második visszatérési érték
12	Benyomjuk a gombot (nyomva van, míg a program beolvassa az adatokat).	2	hamis	1
13	Felengedjük a gombot	2	igaz	2
14	A gomb fel van engedve, és a program olvassa az érzékelő adatait.	2	hamis	0
15	Gyorsan benyomjuk és felengedjük a gombot (közben a program nem olvas, csak a felengedés után).	2	igaz	2

Megjegyzés

Az érzékelők programozása azért bonyolultabb, mint például a motoroké, mert a LEGO MINDSTORMS EV3 Home Edition szoftver vezérlőszerkezetei (ciklus, elágazás, várj blokk stb.) is már eleve felhasználhatják az érzékelőkről nyert adatokat. Például egy ciklus tarthat addig, amíg az érintésérzékelő be van nyomva stb. Ezeket a funkciókat a megfelelő blokkoknál fogjuk tárgyalni.

3.1.12. A színérzékelő programozása

A színérzékelő mint digitális érzékelő annak a fénynek a színét vagy erősségét érzékeli, amely bejut a szenzor elején elhelyezett kis ablakon. A színérzékelőnek három különböző üzemmódja van: *színmód* (Color Mode), *visszavert fényerősség mód* (Reflected Light Intensity Mode), valamint *szórt fényerősség mód* (Ambient Light Intensity Mode).

A színérzékelő vagy egy számot térít vissza, és ekkor a numerikus érték a fény intenzitását jelenti, vagy összehasonlíthatjuk a visszatérítési értéket egy bemeneti értékkel, és ekkor logikai Igaz (True) vagy Hamis (False) értéket eredményez.

*Színmód*ban a színérzékelő hét színt képes megkülönböztetni: *fekete*, *kék*, *zöld*, *sárga*, *piros*, *fehér* és *barna*, ezen felül pedig a *nincs szín*t.

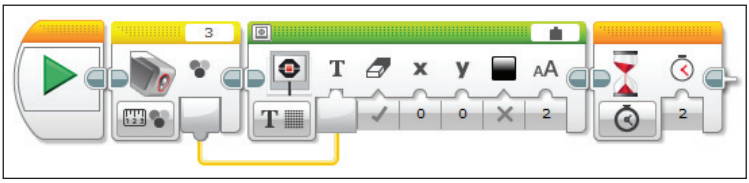
A *visszavert fényerősség mód*ban a színérzékelő egy piros fényt kibocsátó lámpa visszavert fényének az erősségét méri egy 0-tól (nagyon sötét) 100-ig (nagyon világos) terjedő skálán. Robotunk így le tud olvasni például egy színkódolt kártyát.

A *szórt fényerősség mód* segítségével a színérzékelő a környezetéből az ablakba jutó fény erősségét méri. A használt skála itt is 0-tól (nagyon sötét) 100-ig (nagyon világos) terjed. Így a robotunkat be tudjuk programozni például arra, hogy költsön, ha felkel a nap, vagy leálljon, ha kialszik a szobában a fény.

15. táblázat. *A színérzékelő színmódban*

Kód	Jelentés
0	nincs szín
1	fekete
2	kék
3	zöld
4	sárga
5	piros
6	fehér
7	barna

A színérzékelő mintavételezési gyakorisága 1 kHz. A nagyobb pontosság érdekében, a színmód vagy visszavert fényerősség mód esetén úgy építsük meg a robotunkat, hogy a színérzékelő a lehető legközelebb kerüljön a vizsgált felülethez, de ahhoz hozzá nem érve, merőlegesen tudja „letapogatni” a színt.



39. ábra. *A színérzékelő visszatérési értékének kiírása*

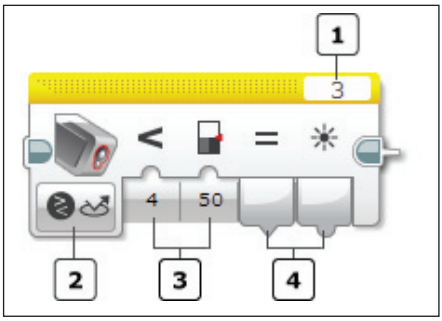
A különböző üzemmódokban a színérzékelő LED-je különböző színű fényt bocsát ki. Színmódban zöld, vörös, kék fénnyel „tapogatja le” a szín RGB (vörös, zöld, kék) komponenseit, visszavert fényerősség módban vörös fényt bocsát ki, szórt fényerősség módban pedig kéket.

A színérzékelő programozása a *színérzékelő blokk* (Color senzor) segítségével történik.

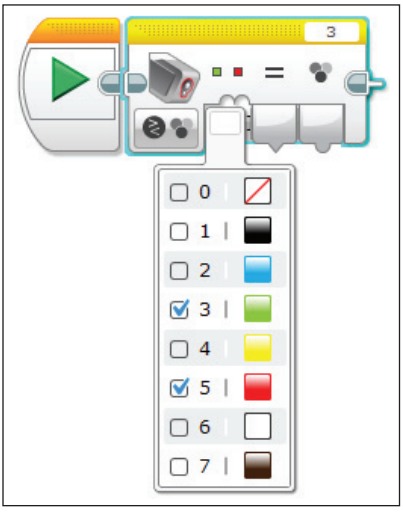
A 40. ábrán látható blokkon az 1-es gomb segítségével a portot választhatjuk ki (port selector). Ezen a porton keresztül fog kommunikálni az EV3 tégla az érzékelővel, innen olvassa be az adatokat. A port az 1, 2, 3 vagy 4 valamelyike lehet. Alapértelmezett portja a 3-as.

A 2-es gomb segítségével egy legördülő menüből kiválaszthatjuk az érzékelő működési módját (mode selector), ez a *Measure* (mérés), *Compare* (összehasonlítás) vagy *Calibrate* (finomhangolás) lehet.

Mérés üzemmódban a blokk a 4-es gombon egy numerikus értéket térít vissza. Színmódban ez 0–7 között jelzi az érzékelt színt, visszavert vagy szórt fényerősség módban pedig 0–100 között a mért fény erősségét.



40. ábra. *Színérzékelő*



41. ábra. *A színérzékelő blokk bemenete az összehasonlítás üzemmód színmódjában*

Összehasonlítás üzemmódban – ha színmódban vagyunk – a blokknak egy bemenete és két kimenete lesz. A bemenet (3-as gomb) egy numerikus tömb. Ha az ebben felsorolt színek közül eggyel is egyezik az érzékelő által mért szín, akkor az első kimeneten a logikai igaz (True) visszatérési érték jelenik meg, különben a logikai hamis (False) érték. A második kimeneten a felismert szín kódja jelenik meg a 0–7 skálán.

Visszavert vagy szórt fényerősség módban a blokknak két bemenete és két kimenete lesz, pont, mint a 40-es ábrán látható 3-as és 4-es gombok. A két bemenet két numerikus érték. Az első a relációs, összehasonlító műveletek kódja a 16. táblázat szerint, a második pedig egy küszöbérték.

16. táblázat. Összehasonlító műveletek kódja

Kód	Szimbólum	Jelentés
0	=	egyenlő
1	≠	nem egyenlő
2	>	nagyobb
3	≥	nagyobb vagy egyenlő
4	<	kisebb
5	≤	kisebb vagy egyenlő

Ha a mért fényerősség a kiválasztott relációban van a megadott küszöbértékkel, akkor az első kimeneten a logikai igaz (True) visszatérési érték jelenik meg, különben a logikai hamis (False) érték. A második kimeneten a mért fényerősség jelenik meg a 0–100 skálán.

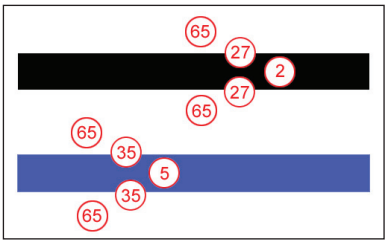
3. feladat

Egy fehér lapra ragasszunk egy matt fekete, valamint egy csillogó kék csíkot például szigetelőszalagból.

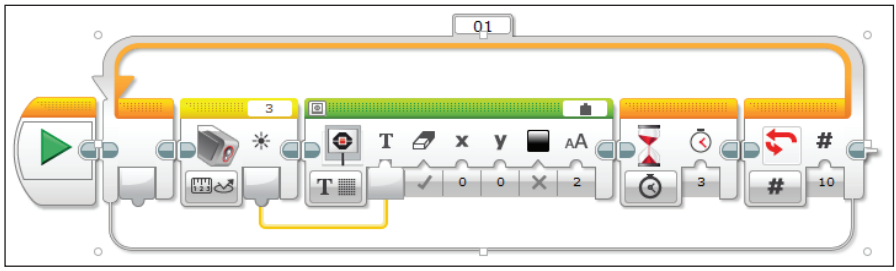
A színérzékelő visszavert fényerősség módját használva mérjük meg a fényerősséget a fehér lapon, a csíkok közelében (az érzékelő fénykúpja félig a csíkon, félig a papíron legyen), valamint a csíkokon.

A színérzékelőt mindvégig 1 cm magasságba tartsuk a lap fölött!

A mért értékeket jegyezzük fel!



42. ábra. *A színérzékelő visszavert fényerősség mérései egy fehér lapra ragasztott matt fekete és fényes kék (nem sötétkék, de nem is világoskék) szalag esetén 1 cm távolságból*



43. ábra. *A 42. ábrán látható méréseket megvalósító egyszerű program*

Finomhangolás üzemmódban a színérzékelő visszavert fényerősség módjának minimum és/vagy maximum értékét állíthatjuk be, illetve visszatérhetünk az eredeti beállításokra (Reset).

Amennyiben megváltoznak a fényviszonyok, például egy sötét szobából kivisszük a LEGO robotot a napfényes szabadba, vagy fordítva, vagy plusz égőket gyújtunk fel, oltunk le, jó finomra hangolni a színérzékelőt. Ehhez egy kis programot kell írnunk, amely megméri a legsötétebb és a legvilágosabb pontban a fényerősséget, majd ezek alapján kalibrálja a szenzort [1].

4. feladat

Írjunk egy programot a színérzékelő finomhangolására!

44. ábra. *A színérzékelő finomhangolása*

Megjegyzés

A *fényerősség* vagy *világosság* a szemünkbe érkező fényenergia mennyiségét jelenti. A mértékegysége: *lumen*. A fényerősség határozza meg, hogy mennyire megfelelően érzékeljük a színeket. Az emberi szem nem érzékeli a 10^{-6} lumen alatti fényt, a 10^{4} lumen fölötti pedig elvakít. Világosság terén a szemünk mintegy 50 fokozatot tud megkülönböztetni. Sötétben (ha nagyon kicsi a fényerősség) csak fekete-fehéren látunk, nem érzékeljük a színeket.

45. ábra. *Fényerősség*

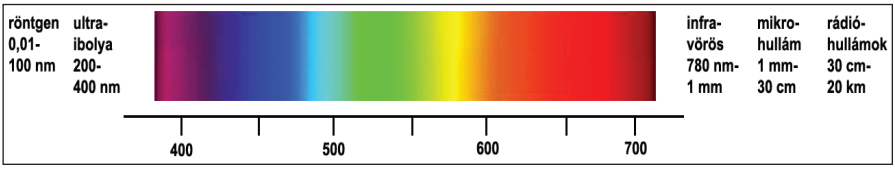
3.1.13. Az infravörös érzékelő programozása

A körülöttünk zajló világról öt érzékszervünk által szerzünk tudomást, azonban az információk legnagyobb részét, több mint 90%-át, a látás során a szemünktől kapjuk.

A *látás* a vizuális információk feldolgozása, amelynek fő célja a tárgyak azonosítása, azok közvetlenül nem észlelhető tulajdonságainak felismerése, illetve a cselekvés vezérlése.

A vizuális információk a *fény* segítségével terjednek, érik el az emberi *szem*et, a látás receptorát.

Az ember számára a fény az elektromágneses sugárzásnak az a része, amelyet a szem érzékelni képes, és amelynek a hatására az agyban képérzet alakul ki. Ez a rész a hullámhossztartomány kb. 380 nm–780 nm közötti intervalluma.



46. ábra. *Elektromágneses hullámok, a fény hullámhossztartománya*

Az *infravörös sugárzás* (Infrared, IR) egy elektromágneses sugárzás, melynek nagyobb a hullámhossza, mint a látható fénynek, de kisebb, mint a mikrohullámnak és a rádióhullámoknak. Az *infra* latin eredetű szó, jelentése: *alatt*.

A legnagyobb hullámhosszal rendelkező látható szín a vörös. Az infravörös sugárzást 780 nm és 1 mm között értjük. Mivel ez már nem látható fény, gyakran alkalmazzák a vezeték nélküli kommunikációban, pozíciók bemérésére, nyomkövetésre stb.

A látható fényhez hasonlóan, az infravörös sugarat is el tudják takarni az objektumok, tehát a vételéhez szabad pálya szükséges az adó és a vevő között. Tehát ha a távirányítóval az infravörös érzékelőt szeretnénk irányítani, vigyázzunk, hogy a távirányítónak legyen is rálátása az infravörös érzékelőre, tehát semmilyen tárgy ne állja útját a kibocsátott infravörös fénynyalábnak. A normális szobai fény nem, de a napfény interferenciát okozhat az infravörös fénnyel.

Az *infravörös érzékelő* egy olyan digitális szenzor, amely képes észlelni a szilárd tárgyakról visszaverődő infravörös fényt, valamint a távirányító által kibocsátott infravörös sugarakat is.

Az infravörös érzékelőnek három különböző üzemmódja van: *közelségi mód* (Proximity Mode), *irányjeladó mód* (Beacon Mode) és *távirányító mód* (Remote Mode).

Közelségi módban az infravörös érzékelő a 0–100 skálán (0 nagyon közel, 100 nagyon távol) megbecsüli egy tárgy távolságát a tárgyról visszaverődő fényhullámok segítségével. Az érzékelő mintegy 70 cm-re lévő tárgyakat képes érzékelni, a tárgy méretétől és formájától függően. A tárgy színe is fontos, a világos színű tárgyak jobban visszaverik az infravörös fényt, mint a sötét színűek.

A másik két üzemmód megértéséhez ismerjük meg először a távirányítót!

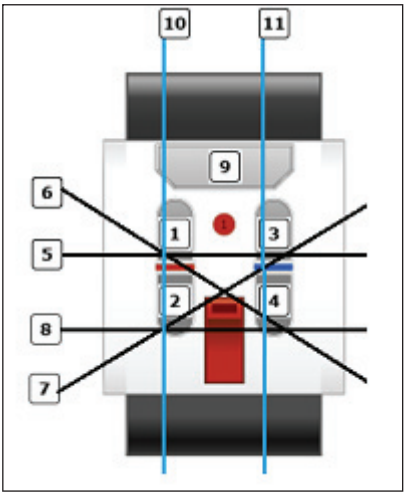
A távirányító, infravörös irányjeladó a LEGO Mindstorms EV3 modell önálló eszköze, amelyet tarthatunk kézben, vagy beépíthetjük egy LEGO modellbe. A működéséhez két AAA méretű alkáli elemre van szükség.

A távirányítónak 4 csatornája, 1 váltógombja, 4 gombja van. A tevékenységet egy zöld LED jelzi. A távirányító egy óra inaktivitás után magától kikapcsol.

Az irányjeladó mód az, amikor a távirányító folyamatos jelt sugároz. Ehhez nyomjuk meg a 9-es gombot. Ekkor a zöld színű LED-kijelző világítani kezd, ez jelzi, hogy az eszköz aktív és folyamatosan jelt sugároz. Ha ismét megnyomjuk a 9-es gombot, akkor kikapcsol. Ha az 1, 2, 3, 4-es gombok valamelyikét nyomjuk meg, akkor a zöld LED csak felvillan, és a jeladás csak a gomb lenyomásáig tart. Ha felengedjük a gombot, megszűnik a jeladás is, ez nem folytonos, mint a 9-es gomb esetében.

A csatornákat a távirányító közepén elhelyezett piros lehúzó gomb segítségével válthatjuk. A gomb fölötti piros mechanikus kijelzőn megjelenik a csatorna száma (1, 2, 3, 4).

Például ha két vagy több robotot szeretnénk vezényelni ugyanazzal a távirányítóval, mindegyik robotnak megfeleltethetünk egy csatornát, így nem keverednek össze a kiadott parancsok. Nem indul el mind a két robot az indulás gomb megnyomására stb. Az infravörös szenzor pontosan a beállított csatorna parancsait érzékeli.



47. ábra. *A távirányító*

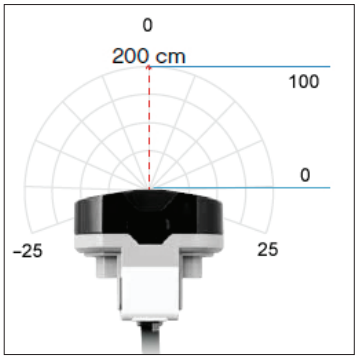
Távirányító módban az infravörös érzékelő képes észlelni, hogy a távirányítón milyen gombok kombinációját nyomtuk meg. Összesen tizenegy gombkombináció lehetséges, ezeket mutatja be a 17. táblázat.

17. táblázat. *A távirányító gombkombinációi*

Kód	**Jelentés**
0	nincs gomb (az irányjeladó mód ki van kapcsolva)
1	1-es gomb
2	2-es gomb
3	3-as gomb
4	4-es gomb
5	az 1-es és a 3-as gomb egyszerre
6	az 1-es és a 4-es gomb egyszerre
7	a 2-es és a 3-as gomb egyszerre
8	a 2-es és a 4-es gomb egyszerre
9	az irányjeladó mód bekapcsolva
10	az 1-es és a 2-es gomb egyszerre
11	a 3-as és a 4-es gomb egyszerre

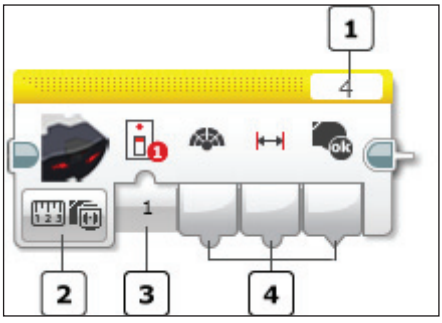
Az *irányjeladó mód*ban válasszuk ki a piros csatornaválasztó gombbal a távirányító négy csatornájának egyikét. Ezen a csatornán fogja az érzékelő a jeleket kapni. Ekkor az infravörös érzékelő maximálisan mintegy 200 cm távolságra észlelni tudja az infravörös fénynyalábot kibocsátó távirányítót abban az irányban, amerre néz. Miután észlelte, az érzékelő meg tudja mérni a jel általános irányát (haladási irány) és távolságát (közelség). Ennek ismeretében például a robotot bújócskázásra tudjuk programozni, amely során meg kell keresse az eldugott távirányítót.

A haladási irány egy –25 és 25 közötti érték lesz, ahol a 0 jelzi, hogy az irányjeladó éppen az infravörös érzékelő előtt van. A távolságot itt is a 0–100 skálán méri (48. ábra) [43].



48. ábra. *Irányjeladás*

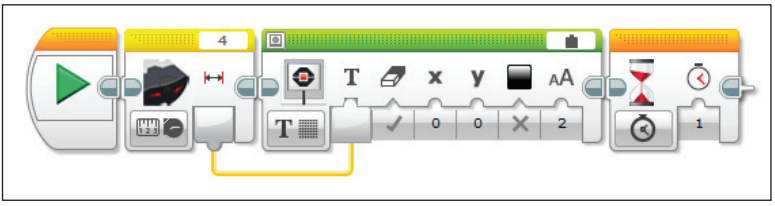
*Távirányító mód*ban az infravörös érzékelő a 17. táblázatnak megfelelően érzékeli a távirányító gombjainak lenyomását.



49. ábra. *Infravörös érzékelő*

Az infravörös érzékelő programozása az *infravörös érzékelő blokk* (Infrared senzor) segítségével történik.

A 49. ábrán látható blokkon az 1-es gomb segítségével a portot választhatjuk ki (port selector). Ezen a porton keresztül fog kommunikálni az EV3 tégla az érzékelővel, innen olvassa be az adatokat. A port az 1, 2, 3 vagy 4 valamelyike lehet. Alapértelmezett portja a 4-es.



50. ábra. *Az infravörös érzékelő által mért távolság (a visszatérési érték) kiírása*

A 2-es gomb segítségével egy legördülő menüből kiválaszthatjuk az érzékelő működési módját (mode selector), ez a *Measure* (mérés) vagy *Compare* (összehasonlítás) lehet.

Mérés üzemmódban, ha a közelségi módot választjuk, a blokk a 4-es gombon (egyetlen kimenet) egy numerikus értéket térít vissza. Ez a 0–100 közötti érték jelzi az érzékelt tárgy távolságát (48. ábra). A 100-as érték azt jelenti, hogy egyáltalán nem érzékelt tárgyat.

Ha az irányjeladó módot választjuk, a blokknak egy bemenete és három kimenete lesz. A bemeneten (3-as gomb) a távirányító csatornáját adhatjuk meg egy 1–4 közötti numerikus értékkel. Csak az ily módon beállított távirányító jeleit fogja érzékelni az infravörös érzékelő. Tehát ha a blokkon például 3-as csatornát állítottunk be, akkor a távirányítót is a 3-as csatornára kell állítsuk.

A 4-es gombon a blokknak 3 kimenete lesz, két numerikus és egy logikai. Az első kimenet (Heading) a távirányító irányát jelzi a –25–25 skálán. A 0-ás azt jelenti, hogy a távirányító éppen az érzékelő előtt van, a negatív értékek a bal oldalt, pozitív értékek a jobb oldalt jelzik. A második kimenet (Proximity) a távirányító távolságát jelzi egy 0–100 közötti numerikus érték segítségével. A harmadik kimenet (Detected) egy logikai érték. Az Igaz (True) azt jelzi, hogy be van kapcsolva az irányjeladó mód (9-es gomb be van nyomva a távirányítón), a Hamis (False) érték pedig azt, hogy az irányjeladó mód nincs bekapcsolva. Ekkor a távolságot jelző kimenet mindig 100, az irányt jelző kimenet pedig 0.

Ha a távirányító módot választjuk, a blokknak egy bemenete és egy kimenete lesz. A bemeneten (3-as gomb) a távirányító csatornáját adhatjuk meg egy 1–4 közötti numerikus értékkel. Csak az ily módon beállított távirányító jeleit

fogja érzékelni az infravörös érzékelő. Tehát ha a blokkon például 3-as csatornát állítottunk be, akkor a távirányítót is a 3-as csatornára kell állítsuk.

A kimeneten (4-es gomb) a blokk visszatérít egy numerikus értéket, a távirányító lenyomott gombjainak a kódját a 17. táblázatnak megfelelően.

Összehasonlítás üzemmódban az infravörös érzékelő összehasonlítja a 16. táblázatban szereplő műveletek valamelyikével a mért adatokat a megadott küszöbértékkel, és egy logikai értéket térít vissza a mért adatok mellett.

Ha a közelségi módot választjuk, a blokknak két bemenete és két kimenete lesz. Az első bemenet a 16. táblázat szerinti összehasonlító művelet kódja, a második bemenet egy numerikus küszöbérték. Az első kimenet az összehasonlítás eredményét tartalmazó logikai érték: igaz (True), ha az összehasonlítás fennáll, ellenkező esetben hamis (False). A második kimenet egy numerikus érték, amely a 0–100 skálán jelzi az érzékelt tárgy távolságát.

Összehasonlítás üzemmódban két irányjeladó módot választhatunk: az *irányjeladó irány*, valamint az *irányjeladó közelségi* módot.

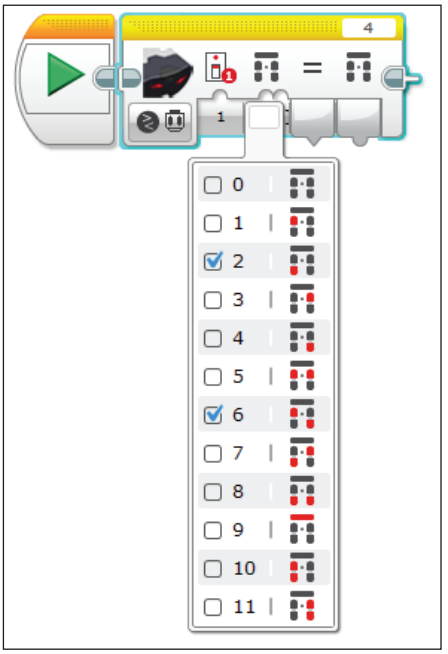
Ha az irányjeladó irány módot választjuk, a blokknak három bemenete és két kimenete lesz. Az első bemeneten a távirányító csatornáját adhatjuk meg egy 1–4 közötti numerikus értékkel. Csak az ily módon beállított távirányító jeleit fogja érzékelni az infravörös érzékelő. A második bemenet a 16. táblázat szerinti összehasonlító művelet kódja, a harmadik bemenet pedig egy numerikus küszöbérték. Az első kimenet az összehasonlítás eredményét tartalmazó logikai érték: Igaz (True), ha az összehasonlítás fennáll, ellenkező esetben Hamis (False). A második kimenet egy numerikus érték, amely a –25–25 skálán jelzi az érzékelt távirányító irányát.

Az irányjeladó közelségi mód annyiban különbözik az irányjeladó irány módtól, hogy a második kimenet most egy numerikus érték lesz, amely a 0–100 skálán jelzi az érzékelt tárgy távolságát.

Ha a távirányító módot választjuk, a blokknak két bemenete és két kimenete lesz. Az első bemeneten a távirányító csatornáját adhatjuk meg egy 1–4 közötti numerikus értékkel. Csak az ily módon beállított távirányító jeleit fogja érzékelni az infravörös érzékelő. A második bemeneten egy numerikus tömbben azokat a távirányító gomb-kódokat kell megadni a 17. táblázat szerint, amelyek lenyomását figyelni szeretnénk. A tömb így több gombkombinációt is tartalmazhat. Ha bármelyik gombot vagy gombkombinációt, amelyik benne van a tömbben, lenyomjuk, akkor a blokk az első kimeneten a logikai Igaz (True) értéket téríti vissza. Ha a lenyomott gomb vagy gombkombináció nincs benne a megadott tömbben, akkor Hamis (False) lesz a visszatérített érték. A második kimeneten numerikus értékként, szintén a 17. táblázatban foglalt kódok szerint a blokk visszatéríti a távirányítón lenyomott gomb vagy gombkombináció kódját.

Természetesen a blokk bemenetei, kimenetei adatdrótok segítségével más blokkokhoz kapcsolhatók.

A távolságjelzésnél láthattuk, hogy az eredményt (a távolságot) nem cm-ben adja meg, hanem a 0–100 skálán. Hasonlóan, habár az érzékelő látószöge nagyobb, mint 180°, az irányt a –25–25 skálán adja meg, így további átalakításokra van szükség a tényleges számításokhoz.



51. ábra. *A távirányító gombjainak megadása*

5. feladat

Az infravörös-érzékelő irányjeladó módját használva forduljon a robotunk a távirányító irányába!

A feladat megoldásához építsünk egy egyszerű robotot. Két nagy motort és az infravörös-érzékelőt használjuk fel hozzá. Két nagy kereke lesz hátul, és elöl középen egy kicsi, amely minden irányban forogni tud, így biztosítva az egyensúlyt és a robot forgását (52. ábra).

A robot forgatásához egy kis mértanfeladatot kell megoldanunk.

1. esetben képzeljük el, amint azt az 53. ábrán bemutatjuk, hogy a robotnak két r sugarú kereke van. A két kerék és a tengely hossza R (a forgásközpont miatt a kerék vastagságának felétől kell mérni). A robot úgy fog megfordulni, hogy az egyik kereke nem forog, áll az O origóban, a másik kereke pedig forog. Így

hasonló fordulást tudunk megvalósítani, mint az evezős csónakkal. Ha csak az egyik evezővel evezünk, a másikkal nem, akkor a csónak megfordul.

52. ábra. *A megépített robot*

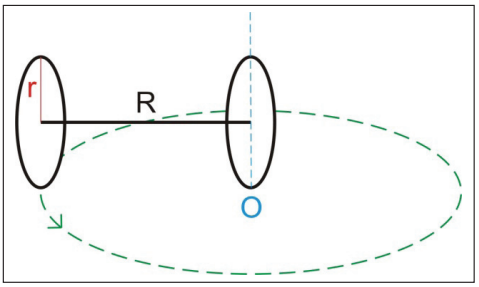
A robotunk tehát az *O* középpont körül fog megfordulni, és ezalatt leírja pont az *R* sugarú kört.

A kérdés az, hogy a kerekek mozgatásához szükséges tank blokkon hány fordulatot állítsunk be a keréknek, hogy a robot pontosan leírja a kört, tehát elforduljon 360°-kal?

A forgó kerék le kell írja a teljes kört, tehát meg kell tegye a kör kerületével megegyező utat. A kör kerülete $2\pi R$. Ha a kerék egyet fordul, a saját kerületével megegyező utat tesz meg. A kerék kerülete $2\pi R$.

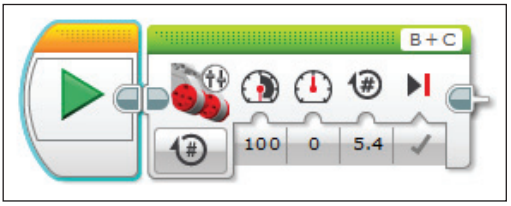
Ha meg akarjuk tudni, hogy hányat kell forduljon a kerék (X), el kell osztanunk a kör kerületét a kerék kerületével, vagyis:

$$X = \frac{2\pi R}{2\pi r} = \frac{R}{r} \quad .$$



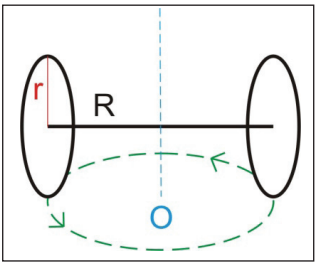
53. ábra. *A robot forgatása – I. eset*

A megépített robotunk esetében a használt kerék sugara (*r*) 2,2 cm, a kerekek közötti távolság (*R*) pedig 11,88 cm (egyik kerék közepétől a másik kerék közepéig), így a fordulatok száma (*X*) 5,4 lesz, ezt kell beállítani a tank blokkon, a program futtatása után pedig a robot körbe fog fordulni.



54. ábra. *A robot forgatása – I. eset: program*

2. esetben a robot úgy is megfordulhat, ha az egyik kereke egy bizonyos erővel előre forog, a másik pedig ugyanakkora erővel hozzá képest fordított irányba. Ekkor a tengely középpontja lesz a forgásközéppont, és a robot az 55. ábrán látható kört írja le.

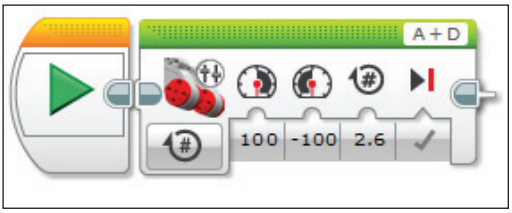


55. ábra. *A robot forgatása – II. eset*

Ebben az esetben az egy kerék által megtett út az előbbi esetbeli fele, a másik felét a másik kerék teszi meg, vagyis

$$X = \frac{R}{2r}.$$

A megépített robotunk esetében a program az 56. ábrán látható.



56. ábra. *A robot forgatása – II. eset: program*

Az előbbi két esetben a robot teljes 360°-os fordulatot tett meg. Nyilvánvaló, hogy az elrejtett távirányító esetében nem ekkorát kell forduljon, hanem akkorát, amekkorát a távirányító és a robot által bezárt szög megkövetel.

Egy tetszőleges szöggel való elforduláshoz szükséges motorfordulat számát nagyon egyszerűen kiszámíthatjuk hármasszabály segítségével. Ha X motorfordulat szükséges a 360°-os forduláshoz, akkor egy tetszőleges α szögű fordulathoz

$$x = \frac{\alpha X}{360}$$

motorfordulat szükséges.

Ha az előbbi I. eset szerinti forgást vesszük, s azt szeretnénk, hogy a robot csak 90°-kal forduljon el, akkor

$$x = \frac{90 \cdot 5,4}{360} = 1,35$$

értéket kell beállítsunk a motor fordulatszámának.

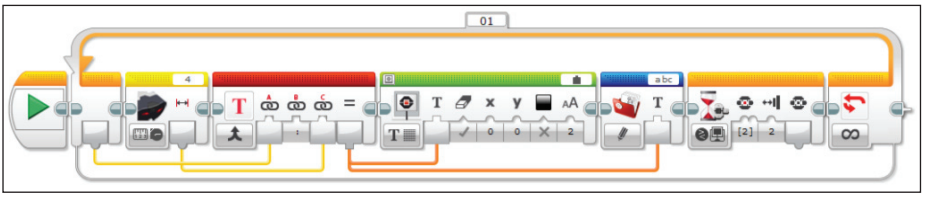
Nézzük meg most, hogyan működik az infravörös érzékelő távolságmérése.

Azt mondtuk, hogy közelségi módban az infravörös érzékelő a 0–100 skálán (0 nagyon közel, 100 nagyon távol) megbecsüli egy tárgy távolságát a tárgyról visszaverődő fényhullámok segítségével. Az érzékelő mintegy 70 cm-re lévő tárgyakat képes érzékelni, a tárgy méretétől és formájától függően.

Végezzünk el egy kísérletet!

Egy 12×8,5×8 cm-es hasáb alakú, világos tárgyat centiméterenként távolítsunk el a közelségi módban lévő infravörös érzékelővel felszerelt robottól, és egy állományba mentsük le a szenzor által mért értékeket!

A program az 57. ábrán látható, a mért adatok pedig a 18. táblázatban, valamint az 58. ábrán.



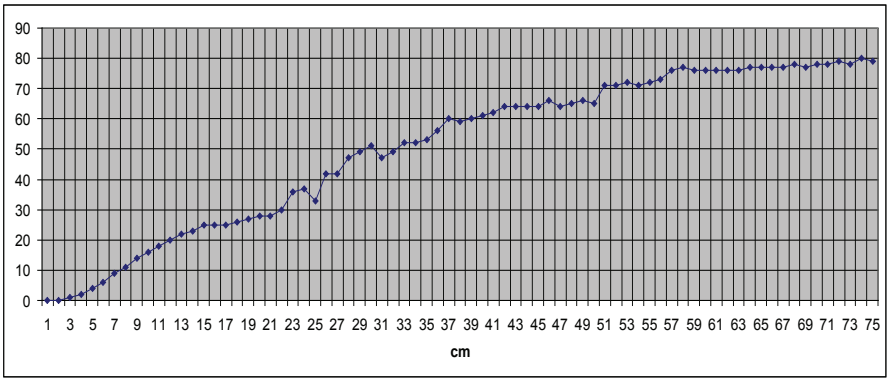
57. ábra. *Program az infravörös érzékelő által mért távolsági adatok kimentésére. Egy ciklusban állományba mentjük a mért adatokat. A tárgyat az infravörös érzékelő elé helyezzük 0 cm-re, és elindítjuk a programot, amely kiírja a centit (a ciklus változója), valamint a mért értéket az állományba. Ezután a tárgyat el kell helyezni az érzékelőtől egy cm-re, és meg kell nyomni a tégla középső gombját, majd így ismételni a méréseket: centinként továbbhelyezni a tárgyat, és megnyomni a középső gombot. Kísérletünkben 100 cm-ig mértük az adatokat.*

A 18. táblázatból látni fogjuk, hogy a 0–5 cm távolságot nem érzékeli jól a szenzor, sem a 70 cm fölöttieket.

18. táblázat. *Az infravörös érzékelővel mért távolságadatok*

cm	mért adat	cm	mért adat	cm	mért adat	cm	mért adat	cm	mért adat	cm	mért adat
0	2	17	25	34	52	51	71	68	78	85	81
1	0	18	26	35	53	52	71	69	77	86	81
2	0	19	27	36	56	53	72	70	78	87	80
3	1	20	28	37	60	54	71	71	78	88	81
4	2	21	28	38	59	55	72	72	79	89	81
5	4	22	30	39	60	56	73	73	78	90	80
6	6	23	36	40	61	57	76	74	80	91	81
7	9	24	37	41	62	58	77	75	79	92	80
8	11	25	33	42	64	59	76	76	80	93	80
9	14	26	42	43	64	60	76	77	79	94	80
10	16	27	42	44	64	61	76	78	80	95	81
11	18	28	47	45	64	62	76	79	80	96	81
12	20	29	49	46	66	63	76	80	81	97	81
13	22	30	51	47	64	64	77	81	81	98	81
14	23	31	47	48	65	65	77	82	80	99	81
15	25	32	49	49	66	66	77	83	80	100	81
16	25	33	52	50	65	67	77	84	81		

Az 58. ábrán ezeket az értékeket jelenítettük meg egy grafikonon.



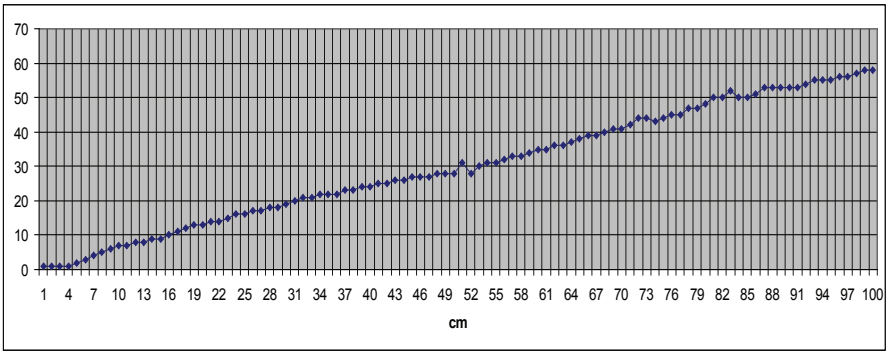
58. ábra. *Az infravörös érzékelővel mért távolságadatok*

Ismételjük meg a távolságmérést úgy, hogy az infravörös érzékelőt irányjeladó módba kapcsoljuk, és a távirányító távolságát mérjük!

A mért adatokat a 19. táblázat, valamint az 59. ábra foglalja össze.

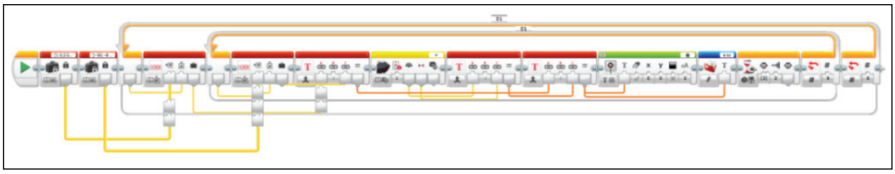
19. táblázat. *Az infravörös érzékelővel mért távolságadatok irányjeladó módban*

cm	mért adat	cm	mért adat	cm	mért adat	cm	mért adat	cm	mért adat	cm	mért adat
0	1	17	11	34	22	51	31	68	40	85	50
1	1	18	12	35	22	52	28	69	41	86	51
2	1	19	13	36	22	53	30	70	41	87	53
3	1	20	13	37	23	54	31	71	42	88	53
4	1	21	14	38	23	55	31	72	44	89	53
5	2	22	14	39	24	56	32	73	44	90	53
6	3	23	15	40	24	57	33	74	43	91	53
7	4	24	16	41	25	58	33	75	44	92	54
8	5	25	16	42	25	59	34	76	45	93	55
9	6	26	17	43	26	60	35	77	45	94	55
10	7	27	17	44	26	61	35	78	47	95	55
11	7	28	18	45	27	62	36	79	47	96	56
12	8	29	18	46	27	63	36	80	48	97	56
13	8	30	19	47	27	64	37	81	50	98	57
14	9	31	20	48	28	65	38	82	50	99	58
15	9	32	21	49	28	66	39	83	52	100	58
16	10	33	21	50	28	67	39	84	50		



59. ábra. *Az infravörös érzékelővel mért távolságadatok irányjeladó módban*

A következő mérés, amit elvégzünk, az irány meghatározására szolgál. Az infravörös érzékelőt irányjeladó módba állítjuk, majd a távirányítót egy olyan megrajzolt papír rácspontjaira helyezzük, amelyen fel vannak tüntetve a távolságok és szögek is, a 47. ábrához hasonlóan, csak sokkal nagyobbak. A méréseket a 60. ábrán látható program segítségével végeztük el, az eredményeket a 20. táblázat foglalja össze.



60. ábra. *Program az infravörös érzékelő által mért irány és távolsági adatok kimentésére irányjeladó módban*

20. táblázat. *Az infravörös érzékelővel mért irány (I), valamint távolság (T) adatok irányjeladó módban*

cm °	10 cm		20 cm		30 cm		40 cm		50 cm		60 cm	
	I	T	I	T	I	T	I	T	I	T	I	T
−90°	−6	8	−11	20	−8	23	−25	30	−25	40	−25	50
−67,5°	−7	7	−16	15	−10	21	−25	26	−25	34	−25	41
−45°	−10	4	−18	13	−12	19	−8	23	−25	30	−25	35
−22,5°	−5	5	−3	14	−6	18	−6	25	−14	32	−13	38
0°	−1	6	−2	13	−3	20	−1	25	−1	31	−3	38

cm / °	10 cm		20 cm		30 cm		40 cm		50 cm		60 cm	
	I	T	I	T	I	T	I	T	I	T	I	T
22,5°	5	5	2	15	4	19	4	26	9	29	10	38
45°	8	6	15	13	12	19	25	26	25	32	25	45
67,5°	6	8	15	16	10	21	25	26	25	33	25	46
90°	4	10	11	20	6	25	25	31	25	40	25	50

A következő lépésben az infravörös érzékelő irányjeladó módját fogjuk használni, hogy megkeressük a távirányítót.

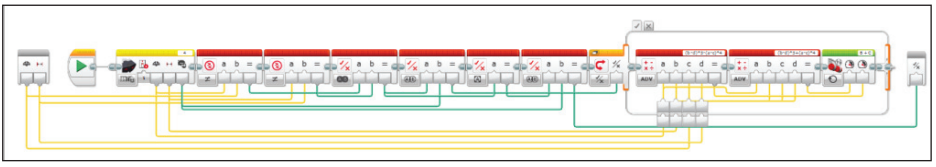
Nyilvánvaló, a legérdekesebb kérdés a 20. táblázatban lévő adatok alapján a valós irány és távolság meghatározása.

Legyen *I* az infravörös érzékelőn mért irány, *T* az infravörös érzékelőn mért távolság, *M* pedig az irányjeladó mód bekapcsolását jelentő logikai Igaz vagy Hamis érték.

Továbbá legyen *I'* a keresett célpont irányának, *T'* pedig a keresett célpont távolságának jellemzői, vagyis milyen irányból és milyen távolságra szeretnénk megközelíteni a távirányítót.

Ha *I* egyenlő *I'*-tel és *T* egyenlő *T'*-tel és az irányjeladó mód be van kapcsolva, tehát *M* igaz, a robot megtalálta a távirányítót, különben egy ciklusban keresi továbbra is azt.

Ha *I* nem egyenlő *I'*-tel vagy *T* nem egyenlő *T'*-tel és az irányjeladó mód be van kapcsolva, tehát *M* igaz, akkor a szakirodalom szerint a robotot tank üzemmódban kell mozgatni úgy, hogy a jobb motort $3\cdot(T - T') - 4\cdot(I - I')$ sebességgel kell mozgatni, a bal motort pedig $3\cdot(T - T') + 4\cdot(I - I')$ sebességgel, mindaddig, amíg meg nem találja a távirányítót.



61. ábra. *A távirányítót kereső robot eljárása*

3.1.14. Az ultrahangos érzékelő programozása

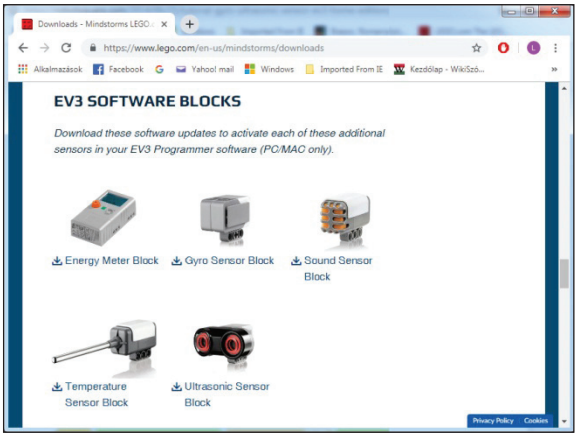
Az ultrahangos és a giroszkópos érzékelők az Education EV3 Core Setben találhatók meg, vagy külön kell őket megrendelni.

Ha LEGO MINDSTORMS EV3 Home Editionben szeretnénk programozni, akkor először telepítenünk kell a megfelelő programblokkokat. Az EV3 tégla

mindkét készletben azonos, lehetővé téve, hogy bármelyik érzékelőt használhassuk, függetlenül attól, hogy melyik alapkészlet van.

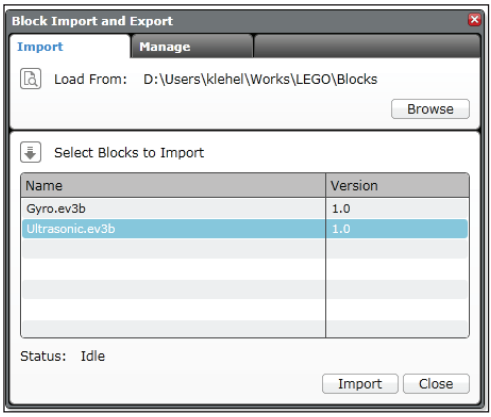
Az alábbi példában bemutatjuk, hogyan telepíthetjük az ultrahangos érzékelő program blokkját. Ismételjük meg ezt az eljárást a giroszkópos érzékelőre vagy a további érzékelőblokkok, például a hőmérséklet-érzékelő blokk és a hangérzékelő blokk telepítéséhez. Ugyanezeket a lépéseket követhetjük egy harmadik fél érzékelő gyártói, például a HiTechnic, a Mindsensors és a Dexter Industries blokkok telepítéséhez is.

1. Töltsük le a telepíteni kívánt blokkot. Ezt megtehetjük a *https://www.lego.com/en-us/mindstorms/downloads* oldal EV3 SOFTWARE BLOCK DOWNLOAD fejezetéből. Itt a 61. ábrán látható blokkokat ajánlja fel a honlap. Mentsük le az állományt (példánkban ez az *Ultrasonic.ev3b*) a számítógépre, és jegyezzük meg a könyvtárat.
2. A letöltött blokkot telepíteni kell, vagyis beimportálni a LEGO MINDSTORMS EV3 Home Editionbe. Indítsuk el a szoftvert, majd a Tools menüből válasszuk ki a Block import menüpontot. Ekkor a 62. ábrán látható párbeszédablak jelenik meg. Itt kattintsunk a Browse gombra, és válasszuk ki a telepíteni kívánt állományt (jelen esetben *Ultrasonic.ev3b*). A Select Blocks to Import részben válasszuk ki a telepíteni kívánt blokkokat, majd nyomjuk meg az Import gombot. Ekkor a rendszer telepíti a blokkot vagy blokkokat, ám a végén figyelmeztet arra, hogy újra kell hogy indítsuk a LEGO MINDSTORMS EV3 Home Editiont azért, hogy a telepítés végleges legyen. Járjunk el a felszólításnak megfelelően, indítsuk újra a szoftvert!



62. ábra. *Letölthető blokkok*

Ha telepítettük a blokkot, és újraindítottuk a szoftvert, akkor használhatjuk is a blokkokat a már eleve telepített blokkokhoz hasonlóan.



63. ábra. *Blokk importálása*

Ultrahangnak a 20 kHz-nél nagyobb frekvenciájú hangot, azaz a nagyfrekvenciás hanghullámot nevezzük. Az ultrahang az emberek számára ugyan nem, de többféle állat számára hallható, közismert, hogy a kutyák reagálnak rá. A denevérek és a delfinek maguk is állítanak elő ultrahangot, amit a tájékozódáshoz használnak fel, s nem csoda, hogy a legtöbb gyermeknek ezek az emlősök jutnak eszébe az ultrahangról, de vannak olyanok is, akik tudják, hogy az orvostudomány is használja különféle vizsgálatokra [2]. Néhány önvezető autó is ultrahangot használ a távolságok érzékelésére, és ultrahang az alapja a szonár működésének is, amely segítségével például a tengeralattjárók feltérképezik a környezetüket. Robotunk érzékelője is a denevérekéhez hasonló módon működik.

Az ultrahangos érzékelőt távolságmérésre használhatjuk.

A kibocsátott ultrahangjel bármiről visszaverődik, ami a levegőnél nagyobb sűrűségű, ezért a kibocsátott jel energiájának egy bizonyos része visszaverődik a vevőbe. A jel oda-vissza terjedési ideje mérhető, és a levegőben terjedő hang sebességének ismeretében távolsággá számítható át. A visszaverő felület típusa nem kritikus, a merőleges beesés viszont hasznos, mivel ez esetben a beeső hullám közvetlenül a vevő felé verődik vissza. A merőlegestől eltérő beesési szög esetén a beeső jelnek kisebb hányada verődik vissza a vevőhöz.

Az ultrahangos érzékelőnek a legmegfelelőbbek a kemény felületek. A lágy tárgyak, például a ruhák elnyelhetik a hanghullámokat, és így nem észlelhetők pontosan. A lekerekített vagy ferde felületű tárgyak is nehezebben érzékelhetők. Az ultrahangos érzékelő nem érzékel olyan tárgyakat, amelyek nagyon közel vannak (közelebb 3 cm-nél vagy 1,5 hüvelyknél). Az érzékelő széles látómezővel rendelkezik, így egy oldalra helyezett közelebbi tárgyat jobban észrevesz, mint egy távolabbit egyenesen előtte.

Az ultrahangos érzékelő adóként és vevőként is működik (kialakítása éppen ezért kettős, egyik az adó, a másik a vevő), ultrahangnyalábot bocsát ki, majd a visszaérkező visszavert hullámokból (visszhangok) számítja ki a távolságot. Képes egy hanghullám kiküldésével szonárként működni vagy a „kósza" ultrahangokat is érzékelni.

Műszaki szempontból az ultrahang használata azért előnyös, mert egyrészt jól irányítható, keskeny sávban sugározható, másrészt pedig könnyen és olcsón előállítható piezo kristályokkal.

A feldolgozás során a távolságot eltelt időként kapjuk meg. Az emberi olvashatóság érdekében az ultrahangos érzékelő átalakítja a kapott értéket távolságra. Ez esetben figyelembe kell venni a hang terjedési sebességét.

A hang terjedési sebessége levegőben a hőmérséklettől, a nedvességtartalomtól és a légnyomástól, tehát a tengerszint feletti magasságtól is függ. Szobahőmérsékleten (0% páratartalom, 20 °C-os környezetben) 343,2 m/s körüli állandónak tételezhető fel.

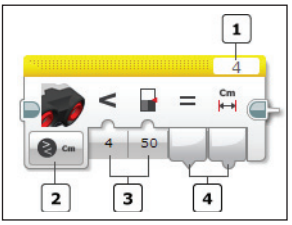
Az ultrahangos érzékelő segítségével a távolságot centiméterben vagy hüvelyk (inch) mérhetjük, így eredményként egy numerikus értéket kapunk. Az érzékelő összehasonlító módban is működik, ekkor a mért értéket összeveti egy küszöbértékkel, és logikai (True vagy False) kimenetet kapunk. Az érzékelő más ultrahangos jeleket is észlelhet *„csak hallás"* módban.

A *távolságmérő* üzemmódot használhatjuk például arra, hogy a robot megálljon egy bizonyos távolságra a faltól.

A *„csak hallás"* üzemmódban az ultrahangos érzékelő segítségével megállapíthatjuk, hogy a közelben működik-e egy másik ultrahangos érzékelő is. Például ezt használhatjuk egy másik „ultrahangos" robot jelenlétének észlelésére.

A 64. ábrán látható blokkon az 1-es gomb segítségével a portot választhatjuk ki (port selector). Ezen a porton keresztül fog kommunikálni az EV3 tégla az érzékelővel, innen olvassa be az adatokat. A port az 1, 2, 3 vagy 4 valamelyike lehet. Alapértelmezett portja a 4-es.

A 2-es gomb segítségével egy legördülő menüből kiválaszthatjuk az érzékelő működési módját (mode selector), ez a *Measure* (mérés) vagy *Compare* (összehasonlítás) lehet.



64. ábra. *Az ultrahangos érzékelő blokkja*

*M*érés üzemmódban a tárgytól mért távolságot kaphatjuk meg numerikus értékként centiméterben (0 és 255 között) vagy hüvelykben (0 és 100 között). Ezt a 4-es gombon szolgáltatott visszatérítési értéket adatdrót segítségével adhatjuk át más blokknak.

A mérés – jelenlét mód (*Presence*) más ultrahangos jeleket keres *„csak hallás”* üzemmódban. Az észlelt ultrahang kimenet igaz (True), ha egy jelet észlel, különben hamis (False).

A fejlett (*Advanced*) üzemmód hasonló a mérés üzemmódhoz, azzal a különbséggel, hogy kiválaszthatjuk, hogy az érzékelő egyetlen ultrahangos jelet (0) vagy folyamatos (1) jelet küld-e a mérési mód bemenetével. A távolságot centiméterben vagy hüvelykben kapjuk meg annak függvényében, hogy melyik módot választottuk.

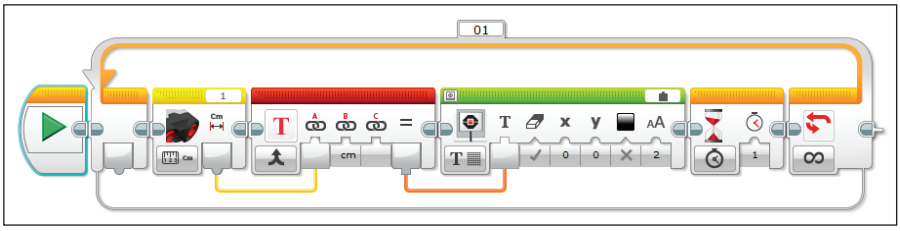
A mérés módban az érzékelő mindig folyamatos ultrahangjelet küld.

Összehasonlítás üzemmódban az ultrahangos érzékelő összehasonlítja a 16. táblázatban szereplő műveletek valamelyikével a mért adatot a megadott küszöbértékkel, és a 4-es gombon egy logikai értéket térít vissza a mért adat mellett.

A 3-as gomb segítségével bemeneti adatként megadhatjuk az összehasonlító műveletet és a küszöbértéket.

Az összehasonlítás – jelenlét mód (*Presence*) más ultrahangos jeleket keres *„csak hallás”* üzemmódban. Az észlelt ultrahang kimenet igaz (True), ha egy jelet észlel, különben hamis (False).

A 65. ábrán látható egyszerű program segítségével meg tudjuk mérni egy tárgy távolságát az ultrahangos érzékelőtől. Az eredmény a tégla kijelzőjén jelenik meg. Ha mozgatjuk a tárgyat, változik a kiírt távolság is.



65. ábra. *Egyszerű ultrahangos távolságmérés centiméterben*

3.1.15. A giroszkópos érzékelő programozása

A digitális EV3 Gyro szenzor (giroszkópos érzékelő) a robotok elfordulását, annak változását képes érzékelni. Segítségével elfordulási szögeket tudunk mérni, megvalósíthatjuk az egyensúlyozó robotot, vagy érzékelni tudjuk, ha a robot leesett.

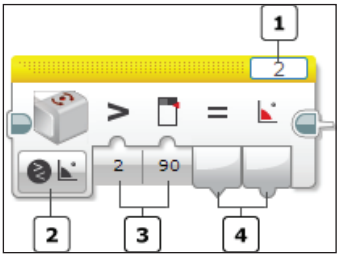
A giroszkóp (más néven pörgettyű vagy szögsebességmérő) a fizikából ismert perdületmegmaradás törvényét bemutató eszköz. A legegyszerűbb változata egy tengely körül szabadon forgó lendkerékből áll. Amikor a kerék forgása közben az eszközt a tengelyre merőleges erőhatás éri, az eszköz a tengelyre és a külső erőhatásra egyaránt merőleges irányban fordul el.

A giroszkópot Léon Foucault francia fizikus találta fel és nevezte el 1852-ben, amikor egy, a Föld forgását igazolandó kísérletén dolgozott (Foucault-inga).

Giroszkópokat gyakran alkalmaznak iránytűk helyett vagy azok kiegészítéseként. Ha ugyanis az eszközt további két tengellyel látjuk el úgy, hogy a három tengely egymásra kölcsönösen merőleges legyen, hogy a giroszkóp tetszőleges irányba szabadon el tudjon fordulni, akkor a pörgő kerék megőrzi forgási tengelyének eredeti irányát, függetlenül attól, hogy a kerete hogyan fordul el.

A giroszkópos érzékelő segítségével meg lehet mérni a forgási sebességet (fordulatszámot) vagy a forgási szöget, így egy numerikus kimenetet kapunk. Az érzékelt adatokat össze is hasonlíthatjuk egy megadott küszöbértékkel, így logikai (igaz vagy hamis) kimenetet kapunk.

A giroszkópos érzékelő csak egy forgási tengely körül érzékeli a mozgást. Ezt az irányt az érzékelőn lévő nyilak jelzik. Mindig győződjünk meg arról, hogy az érzékelőt a megfelelő irányban rögzítettük a robothoz. Ha több forgási tengely körül szeretnénk elmozdulási szögeket mérni, több giroszkópos érzékelőre lesz szükségünk. A szög és a fordulatszám (sebesség) egyaránt lehet pozitív vagy negatív. Az óramutató járásával megegyező forgás pozitív és az óramutató járásával ellentétes irányban negatív.



66. ábra. *A giroszkópos érzékelő blokkja*

A 66. ábrán látható blokkon az 1-es gomb segítségével a portot választhatjuk ki (port selector). Ezen a porton keresztül fog kommunikálni az EV3 tégla az érzékelővel, innen olvassa be az adatokat. A port az 1, 2, 3 vagy 4 valamelyike lehet. Alapértelmezett portja a 2-es.

A 2-es gomb segítségével egy legördülő menüből kiválaszthatjuk az érzékelő működési módját (mode selector), ez a *Measure* (mérés), *Compare* (összehasonlítás) vagy *Reset* (visszaállítás) lehet.

Reset (visszaállítás) üzemmódban lenullázhatjuk az érzékelő értékeit. A szögmérést mindig a legutolsó visszaállításhoz viszonyítva végzi az érzékelő. A forgási szöget úgy számítjuk ki, hogy az idő múlásával ismételten hozzáadjuk a forgási sebességet. A forgási sebesség kisebb pontatlanságai az idő múlásával emelkednek, és a forgási szög „sodródik". A forgási szög 0-ra történő visszaállítása törli a hibát, és új kiindulási pontot határoz meg a jövőbeli szögmérésekhez.

A mérés üzemmódban a visszatérítési érték egy vagy két numerikus érték a 4-es gombon.

Három módja van a mérés üzemmódnak: *Angle* (szög), *Rate* (fordulatszám), valamint *Angle and Rate* (szög és fordulatszám).

A szög üzemmód az elforgatás szögét téríti vissza szögben. A szöget az érzékelő utolsó visszaállításhoz viszonyítva méri.

A fordulatszám mód az elforgatás fordulatszámát adja meg. Pihenő helyzetben a fordulatszám 0. A fordulatszám a forgó testek, alkatrészek, gépek időegység alatti teljes körforgásainak száma. A műszaki gyakorlatban széles körűen elterjedt a percenkénti fordulatszám, amelyet a szögsebesség helyett vagy azzal párhuzamosan használnak a körmozgás sebességének mérőszámául. A LEGO robotoknál az időt másodpercben mérik, így a másodpercenkénti fordulatszám megegyezik a mozgás frekvenciájával. Ha a fordulatszámot megszorozzuk az idővel, akkor megkapjuk, hogy az adott idő alatt hányszor futotta be a test ugyanazt a kört [5].

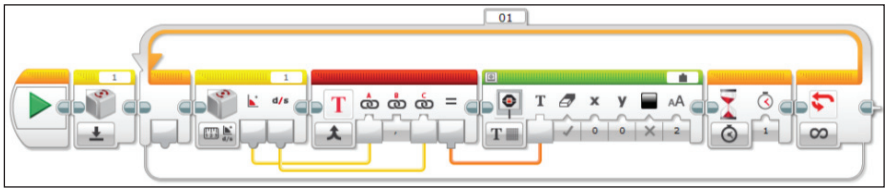
A szög és fordulatszám mód mindkét értéket, a szöget is és a fordulatszámot is visszaadja.

Összehasonlítás üzemmódban a giroszkópos érzékelő összehasonlítja a 16. táblázatban szereplő műveletek valamelyikével a mért adatot a megadott küszöbértékkel, és a 4-es gombon egy logikai értéket térít vissza a mért adat mellett.

A 3-as gomb segítségével bemeneti adatként megadhatjuk az összehasonlító műveletet és a küszöbértéket.

Az összehasonlítás üzemmódnak két módja van, az *Angle* (szög), *Rate* (fordulatszám). Szög módban a szög értékét tudjuk összehasonlítani a megadott küszöbértékkel, fordulatszám módban pedig a fordulatszám értékét.

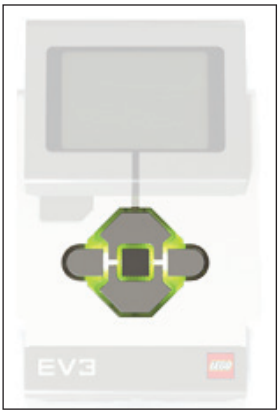
A 67. ábrán látható egyszerű program segítségével az EV3-as téglára rögzített girszkópos érzékelő segítségével megmérjük a tégla elforgatásának szögét, illetve fordulatszámát, és ezeket az adatokat megjelenítjük a tégla kijelzőjén.



67. ábra. *Egyszerű giroszkópos szög- és fordulatszámmérés*

3.1.16. A tégla gombjainak programozása

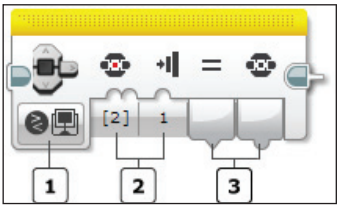
A téglának 5 gombja van, ezek a következők: Bal (Left), Középső (Center), Jobb (Right), Fel (Up), valamint Le (Down). Ezeket a gombokat felhasználhatjuk a programjaink futása során. A téglán ezeken kívül még van egy gomb, a Vissza (Back) gomb, ennek a lenyomása azonban egy futó program befejezését vonja maga után, így ez nem használható fel a programozás során.



68. ábra. *A tégla gombjai*

Az érintésérzékelőhöz hasonlóan a téglagombok is megőrzik azt, ha benyomták és felengedték őket, tehát létezik az ütközött (Bumped) állapot, viszont nem tudják kezelni, ha egyszerre több gombot nyomunk meg. Ekkor valamelyik felülírja az összes többit, és eredményként csak annak a gombnak a lenyomása jelenik meg.

A tégla gombjainak állapota lekérdezhető a tégla gombok (Brick Buttons) blokk segítségével.



69. ábra. *A tégla gombok blokk*

A 69. ábrán látható blokkon az 1-es gomb segítségével egy legördülő menüből kiválaszthatjuk az érzékelő működési módját (mode selector), ez a *Measure* (mérés) vagy *Compare* (összehasonlítás) lehet.

A mérés üzemmódban a visszatérítési érték egy numerikus érték, a lenyomott gomb azonosítója. Ezt a 3-as gombon szolgáltatott visszatérítési értéket adatdrót segítségével adhatjuk át más blokknak. A visszatérési érték jelentését a 21. táblázat foglalja össze.

21. táblázat. *A gombok kódolása*

érték	**gomb**
0	nem történt gomblenyomás
1	bal (left)
2	közép (center)
3	jobb (right)
4	fel (up)
5	le (down)

Az összehasonlítás üzemmódban a 2-es gomb segítségével bemeneti adatként megadhatjuk, hogy a *benyomott* (Pressed), *felengedett* (Relased) vagy ütközött (Bumped) állapotot szeretnénk-e lemérni. A menü segítségével egy numerikus értéket állíthatunk be: 0 – felengedett, 1 – benyomott, 2 – ütközött.

A másik bemeneti érték segítségével a vizsgálni kívánt gombok listáját adhatjuk meg egy legördülő menü segítségével, a 21. táblázatban megadott kódok alapján.

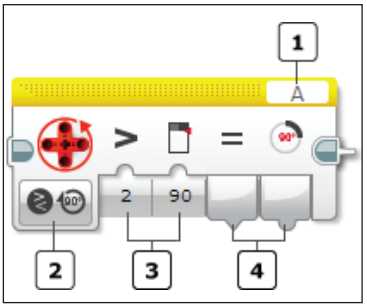
A blokk két értéket térít vissza. Az első egy logikai érték, amely megmutatja, hogy az elvárt állapot bekövetkezett-e valamelyik gombon. A második numerikus érték pedig annak a gombnak az azonosítója, amelyen az elvárt állapot bekövetkezett.

3.1.17. A motor forgásérzékelőjének programozása

A motorok forgásérzékelőjét arra használjuk, hogy megtudjuk, mennyit fordultak el a motorok. Ilyen érzékelőkkel látták el a nagy motort, közepes motort, de az NXT motorokat is. Az érzékelők a motor fordulását tudják mérni fokokban vagy fordulatszámban, a motor egy teljes fordulata 360°.

Az érzékelő segítségével a motor beállított működési erejét (sebességét) is megtudhatjuk.

A motorok forgásérzékelőjét a motor-forgásérzékelő blokk segítségével tudjuk programozni.



70. ábra. *A motor-forgásérzékelő blokk*

A 70. ábrán látható blokkon az 1-es gomb segítségével a portot választhatjuk ki (port selector). Ezen a porton keresztül fog kommunikálni az EV3 tégla az érzékelővel, innen olvassa be az adatokat. Figyelem! A port itt az A, B, C vagy D valamelyike lehet, hisz a motorokat ezekhez a portokhoz kötjük. Alapértelmezett portja az A.

A 2-es gomb segítségével egy legördülő menüből kiválaszthatjuk az érzékelő működési módját (mode selector), ez a *Measure* (mérés), *Compare* (összehasonlítás) vagy *Reset* (visszaállítás) lehet.

Mérés üzemmódban a visszatérítési érték a beállított fok, fordulatszám vagy erő nagyságát jelenti, ezt numerikus értékként a 4-es gombon kapjuk vissza. Minden mérés relatív az utolsó *Reset* (visszaállítás) híváshoz, tehát úgy mérhetünk valamit, hogy először lefuttatjuk a motor-forgásérzékelő blokkot *Reset* (visszaállítás) működési módban, ez lenullázza a számlálókat, majd innen kezdve kérdezhetünk rá arra, hogy mennyit fordult a motor. Az érzékelő értékeinek lenullázása nincs hatással a beállított motor értékekre, tehát a beállított erő és fordulatszám megmarad, csak az olvasott, lekérdezett, visszaszolgáltatott értékeket érinti ez.

Összehasonlítás üzemmódban a motor forgásérzékelő összehasonlítja a 16. táblázatban szereplő műveletek valamelyikével a mért adatokat a megadott küszöbértékkel, és egy logikai értéket térít vissza a mért adatok mellett.

A bemeneti adatokat (az összehasonlító műveletet és a numerikus küszöbértéket) a 3-as gombon kell megadni. A 4-es gombon pedig a két kimenet fog szerepelni.

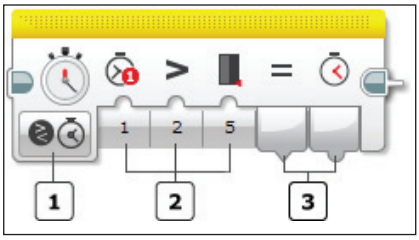
Az első kimenet az összehasonlítás eredményét tartalmazó logikai érték: Igaz (True), ha az összehasonlítás fennáll, ellenkező esetben Hamis (False). A második kimenet egy numerikus érték, amely a ténylegesen mért adatot szolgáltatja vissza.

3.1.18. Az időzítő programozása

Időzítőket (timereket) időmérésre, időintervallumok mérésére használunk. Ezek az EV3 MINDSTORMS tégla belső érzékelőiként is felfoghatók, így működésükhöz külön portbeállítás nem szükséges.

Az időzítő segítségével megmérhetjük, hogy hány másodperc alatt tette meg a robot a leprogramozott utat, mennyi időt vesz fel egy forgás, vagy bármilyen tetszőleges időmérés megvalósítható.

Az EV3 tégla 8 időzítővel rendelkezik, tehát párhuzamosan 8 feladat esetében lehet időt mérni. Az időzítők a program során bármikor lenullázhatók, és akkor csak onnan kezdve mérik az időt. Ha egy időzítőt úgy használunk, hogy előtte nem nulláztuk le, akkor a program indításától méri az időt. Mind a 8 időzítő automatikusan lenullázódik és elindul, amikor a program elkezdődik.



71. ábra. *Az időzítő blokk*

A 71. ábrán látható blokkon az 1-es gomb segítségével egy legördülő menüből kiválaszthatjuk az időzítő működési módját (mode selector), ez a *Measure* (mérés), *Compare* (összehasonlítás) vagy *Reset* (visszaállítás) lehet.

A 2-es gomb segítségével beállítható első érték az időzítő száma. A létező 8 időzítő közül a tégla ezzel hajtja végre a kért műveletet.

A mérés üzemmódban a visszatérítési érték egy numerikus érték, az utolsó lenullázás (vagy a program kezdete óta) eltelt idő. Ezt a 3-as gombon szolgáltatott visszatérítési értéket másodpercben (second, sec) méri a rendszer.

Összehasonlítás üzemmódban az időzítő összehasonlítja a 16. táblázatban szereplő műveletek valamelyikével a mért időt a megadott küszöbértékkel, és egy logikai értéket térít vissza a mért idő mellett.

A bemeneti adatokat (az időzítő számát, az összehasonlító műveletet és a numerikus küszöbértéket) a 2-es gombon kell megadni. A 3-as gombon pedig a két kimenet fog szerepelni.

Az első kimenet az összehasonlítás eredményét tartalmazó logikai érték: Igaz (True), ha az összehasonlítás fennáll, ellenkező esetben Hamis (False). A második kimenet egy numerikus érték, amely a ténylegesen mért időt szolgáltatja vissza.

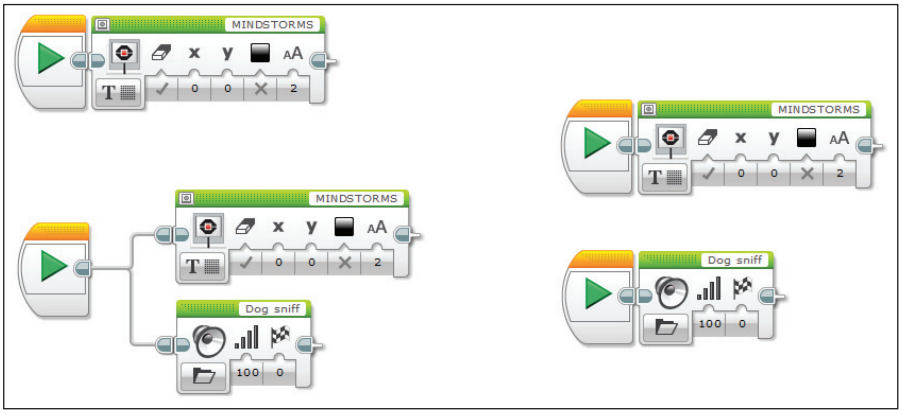
Lenullázás üzemmódban az időzítő nullára állítja számlálóját, és onnan kezdi számolni az eltelt időt.

3.1.19. A Start gomb

A folyamat blokkok közül a Start gomb az első. Szerepe egyszerű, de igen fontos. Minden olyan programszekvenciát, amelyet futtatni akarunk, el kell lássunk egy Start gombbal. Egy program több szekvenciából is állhat, tehát több Start gombot helyezhetünk fel a felületre. Ekkor a szekvenciák automatikusan, egyszerre és párhuzamosan kezdenek futni, ahogy a program elindult. Ha valamely programszekvenciának nincs Start gombja, soha nem fog lefutni.

Ha a megépített robotunk össze van kötve valamilyen kapcsolattal a számítógéppel (USB, Bluetooth, Wi-Fi), akkor a felületen valamelyik Start gombra kattintva csak a megfelelő szekvenciát tudjuk elindítani, és a robot csak ezt fogja végrehajtani, habár a teljes programot lefordítja a rendszer és rátölti a robotra.

72. ábra. *A Start gomb*

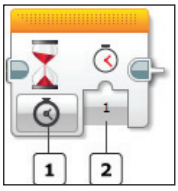


73. ábra. *Program egy és több Start gombbal*

a) *egy szálon futó program*, b) *egy Start gombos elágazó, párhuzamosan futó programszekvenciák*, c) két Start gombos programszekvencia

3.1.20. A Várj blokk

A Várj (Wait) blokk megállítja a program futását, és a következő blokk végrehajtása előtt vár valamire. A Várj blokk nem állítja le a robotot, ha például a Várj blokk végrehajtása előtt a motorok be voltak kapcsolva, ezek forogni fognak a Várj blokkal megadott várakozás közben is.



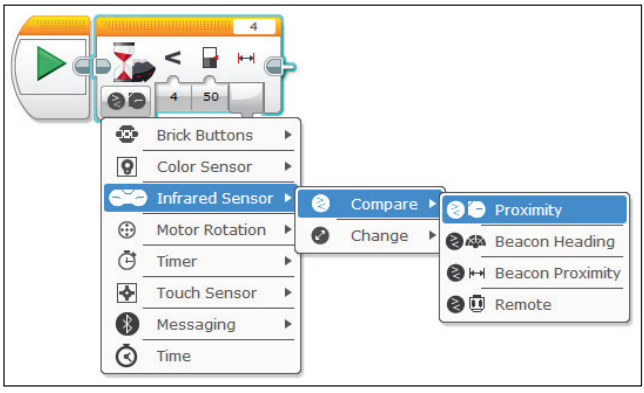
74. ábra. *A Várj blokk*

Az 1-es módszelektor segítségével ki tudjuk választani, hogy mire várjon a blokk. Várhat egyszerűen arra, hogy leteljen a megadott idő, de várhat egy téglagombra, színérzékelőre, infravörös érzékelőre, motor forgásérzékelőjére, egy időzítőre, az érintésérzékelőre vagy egy üzenetre.

A 2-es gomb segítségével a kiválasztott módnak megfelelő adatot állíthatjuk be.

Ha azt választottuk ki, hogy a Várj blokk egy bizonyos ideig várjon, akkor itt, a 2-es gombon kell megadjuk másodpercekben mérve a várakozási időt. Megjegyezzük, hogy az időt valós szám segítségével, tizedes rész használatával is megadhatjuk, például a 2.5 beállítás két és fél másodperc várakozási időt eredményez.

Ha valamilyen érzékelőre kell várjon a blokk, összehasonlítás (compare) vagy *változás* (change) módokat választhatunk ki.



75. ábra. *Az érzékelőkre való várás módjai*

Összehasonlítás módban a blokk folyamatosan olvassa be az érzékelő adatait, és addig vár, míg az érzékelőn meg nem jelenik a beállított érték. Megjegyzendő, hogy ha az összehasonlítás már igaz a Várj blokkba való belépés elején, a program nem fog várni, hanem folytatja működését a következő blokkal.

Változás módban a blokk folyamatosan olvassa be az érzékelő adatait, és addig vár, amíg az érzékelőn meg nem jelenik egy másik érték. Például ha azt állítottuk be, hogy a színérzékelő a piros színen várjon, a program futása mindaddig várakozik, míg az érzékelő a piros színt látja. Ha egy más szín jelenik meg az érzékelő látáskörében, a program ismét futni kezd.

Összehasonlítás módban a blokk a következőkre várhat:
– téglagombokra;
– színérzékelőre;
– infravörös érzékelőre;
– motorforgásra;
– időzítőre;
– érintésérzékelőre;
– üzenetre;
– időre.

Téglagombok esetén beállíthatunk egy vagy több téglagombot, valamint azt, hogy *benyomott* (0 – Pressed), *felengedett* (1 – Relased) és *ütközött* (2 – Bumped) legyen annak a gombnak az állapota, amelyre várunk.

A gombazonosító visszatérési érték (Button ID) azt a gombot azonosítja be, amelyikre a beállított tulajdonság igaznak bizonyul.

Ha a színérzékelőre várunk, akkor a szín, a visszavert fény, valamint a háttérfény erősségének változására várhatunk.

Ha színváltozásra várunk, akkor beállíthatjuk, hogy milyen színre vagy színekre várunk, a kimeneten pedig megjelenik annak a színnek a kódja, amelyet a blokk érzékelt.

Ha visszavert vagy a háttérfény erősségének változására várunk, akkor beállíthatjuk az összehasonlítási műveletet (0 – Egyenlő, 1 – Nem egyenlő, 2 – Nagyobb, 3 – Nagyobb vagy egyenlő, 4 – Kisebb, 5 – Kisebb vagy egyenlő), valamint a küszöbértéket, amire várunk. Ha a reláció igazzá válik, a program befejezi a várakozást, és a kimeneten visszatéríti a mért erősségértéket.

Az infravörös érzékelő esetében is megadhatjuk az erre vonatkozó összes mód (*közelségi mód*, *irányjeladó mód* és *távirányító mód*) szerinti küszöbértéket és összehasonlítási műveletet, amire várakozzon. A várakozás befejezése után a kimeneten megjelenik az effektíven mért érték.

Ha a motor forgására várunk, beállíthatjuk a küszöbértéket, a relációs műveletet, valamint azt, hogy szögre, fordulatszámra vagy erősségre várunk. A várakozás befejezése után a kimeneten megjelenik a ténylegesen mért érték.

Ha egy időzítő értékére várunk, akkor beállíthatjuk az időzítőt (ez az érték 1-től 8-ig változhat; 8 időzítőt tud kezelni a LEGO Mindstorms), a relációs műveletet, valamint a küszöbértéket. Ha a várakozás befejeződött, tehát az időzítő elérte a beállított küszöbértéket, akkor a blokk kimenetén megjelenik az eltelt idő (amennyit várakoztunk) – másodpercekben.

Ha az érintésérzékelőre várunk, akkor beállíthatjuk az érintés módját, hogy *benyomott* (0 – Pressed), *felengedett* (1 – Relased) és *ütközött* (2 – Bumped) legyen az állapota. A visszatérési érték az érintésérzékelő ténylegesen mért állapota.

Ha üzenetre várunk, akkor a beérkezett üzenet szöveges, numerikus vagy logikai lehet. Az üzenet Bluetooth-csatornán érkezik. Szöveges üzemmódban azt állíthatjuk be, hogy a beérkezett üzenet legyen egyenlő vagy nem egyenlő egy megadott szöveggel. A várakozás befejeztével a kimeneten megjelenik a ténylegesen beérkezett szöveges üzenet.

Numerikus üzenet esetében a relációs műveletek (0 – Egyenlő, 1 – Nem egyenlő, 2 – Nagyobb, 3 – Nagyobb vagy egyenlő, 4 – Kisebb, 5 – Kisebb vagy egyenlő) valamelyikét állíthatjuk be, valamint a küszöbértéket, amelyre várakozunk. A várakozás befejeztével a kimeneten megjelenik a ténylegesen beérkezett numerikus üzenet.

Logikai üzenet esetén az IGAZ vagy a HAMIS érték jelenik meg a kimeneten.

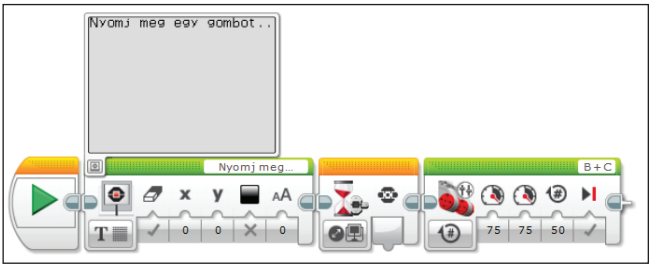
Ha egy adott idő leteltéig várunk, akkor azt kell beállítanunk, hogy hány másodperc teljen el várakozással.

Változás módban a blokk a következőkre várhat:
– téglagombokra;
– színérzékelőre;
– infravörös érzékelőre;
– motorforgásra;
– időzítőre;
– érintésérzékelőre;
– üzenetre.

Változás módban a Várj blokk folyamatosan olvassa az adatokat az esetleges érzékelőkről vagy más komponensekről, és addig vár, amíg értékváltásra kerül sor, vagy egy általunk megadott értéket vesz fel a bemenet.

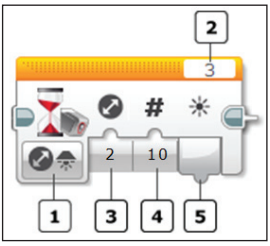
A következő érzékelők esetén a várakozás mód azt jelenti, hogy a program addig vár, ameddig az érzékelő a blokkba való belépés előtti értéktől bármilyen más különböző értéket érzékel, tehát megváltozik valami: téglagombok, színérzékelő szín módban, infravörös érzékelő távirányító módban, érintésérzékelő, üzenet szöveges vagy logikai módban. A Várj blokk visszatéríti a megváltozott – mért – értéket.

Például a 76. ábrán látható programrészen a Várj blokk addig nem indítja el a robot motorjait, ameddig nem nyomtunk le egy akármilyen téglagombot.



76. ábra. *Várakozás téglagomb lenyomására*

Minden más esetben (színérzékelő visszavert és szórt fényerősség módban; infravörös érzékelő közelségi módban vagy irányjeladó haladási és közelségi módban; motor forgásérzékelője fok, fordulatszám vagy erősség módban; időzítő; üzenet numerikus adat módban) be tudjuk állítani a különbözőség irányát (nagyobb, kisebb, bármilyen) és küszöbértékét is. A Várj blokk visszatéríti a megváltozott – mért – értéket.



77. ábra. *A Várj blokk változás módban*

Az 1-es módszelektor segítségével ki tudjuk választani, hogy mire várjon a blokk – változás módban.

A 2-es gomb segítségével a portot állíthatjuk be (portszelektor).

A 3-as gomb segítségével állíthatjuk be a különbözőség irányát:

– 0 = nagyobb (Increase)

– 1 = kisebb (Decrease)

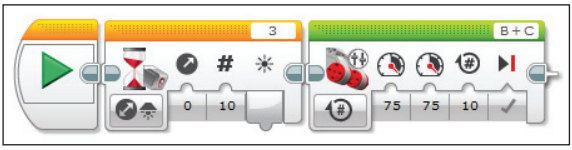
– 2 = bármilyen (Any).

A 4-es gombon tudjuk beállítani a küszöbértéket.

Az 5-ös gombon téríti vissza a blokk a mért (érzékelt) értéket.

A 77. ábrán látható programrészen a Várj blokk addig nem indítja el a robot motorjait, ameddig a szobában a környezeti (szórt) fény erőssége 10 egységgel

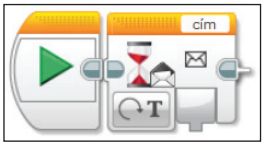
nagyobb nem lesz, mint amekkora volt a blokkba való belépéskor. Tehát ha belépünk a szobába és felkapcsoljuk a villanyt, akkor a robotunk beindul.



78. ábra. *Várakozás a villany felkapcsolására*

Üzenetek esetén a Várj blokknak létezik egy *frissítés* (Update) módja is.

A blokk ekkor addig vár, ameddig be nem érkezik a megfelelő típusú (szöveg, numerikus vagy logikai) üzenet az üzenet címével (fejlécével) együtt. A blokk kimenetén megjelenik a beérkezett üzenet.



79. ábra. Üzenetek frissítés módja

3.1.21. A Ciklus blokk

A programozási nyelvek külön utasításosztályát képezik a ciklusszervező, iteratív számításvezérlő utasítások. Az osztály két lényeges alosztályra bomlik: a *rögzített lépésszámú* és a *változó lépésszámú ciklusok*ra. A rögzített lépésszámú ciklusok az eleve megadott lépésszámig ismétlik a végrehajtandó utasításokat, a változó lépésszámú ciklusok pedig addig ismételnek, ameddig egy megadott feltétel igaz vagy hamis.

Beszélhetünk *elöltesztelős* és *hátultesztelős* ciklusokról.

Elöltesztelős ciklusok esetén már a ciklusmag legelső ismétlését is megelőzi a ciklus vezérlőfeltételének ellenőrzése. Az elöltesztelő ciklusok sajátossága tehát az, hogy amennyiben a legelső tesztelés során a ciklus vezérlőfeltétele nem engedélyezné a ciklusmag ismétlését, úgy az egyszer sem kerül végrehajtásra.

A hátultesztelő ciklus esetén a ciklus magját képező utasítások mindenképpen végrehajtódnak, majd a ciklus végén, hátul következik egy feltételvizsgálat, amely eldönti, hogy bent maradunk-e a ciklusban vagy kilépünk.

Megjegyezzük, hogy *pszeudokód*ban az elöltesztelős ciklusok addig ismételnek, ameddig a megadott feltétel igaz, a hátultesztelős ciklusok pedig addig,

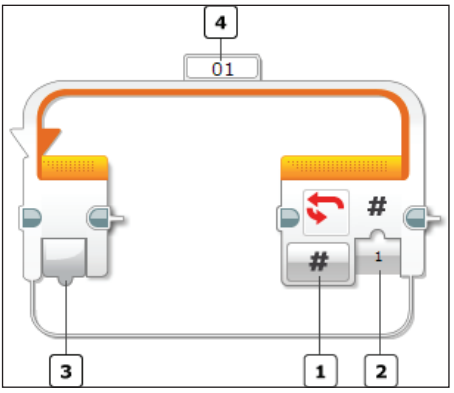
ameddig a megadott feltétel hamis. Az egyes programozási nyelvekre ez a szabály általában nem vonatkozik.

A feltétel logikai értékének módosulása maga után vonja a ciklus befejezését. Amennyiben például egy hátultesztelős ciklus esetében a feltétel mindig hamis, *végtelen ciklus*ról beszélünk, hisz az ismétlés soha nem fog leállni.

A ciklusokban egy *ciklusváltozó* mondhatja meg az ismétlések számát, vagyis azt, hogy éppen a hányadik ismétlésnél tartunk.

A végrehajtandó utasításokat a ciklus *magvá*nak nevezzük. A ciklus magvát el kell határolni a többi utasításoktól. A ciklus befejezése után a mag utasításai többet nem hajtódnak végre, hanem a vezérlés a ciklust követő utasításokkal folytatódik.

*Növekmény*nek vagy *lépés*nek nevezzük a ciklusváltozót módosító értéket.



80. ábra. *Ciklus*

Az 1-es módszelektor segítségével tudjuk kiválasztani a ciklus típusát, megállási feltételét.

A 2-es gomb segítségével a ciklus bemenetét (bemeneteit) tudjuk megadni.

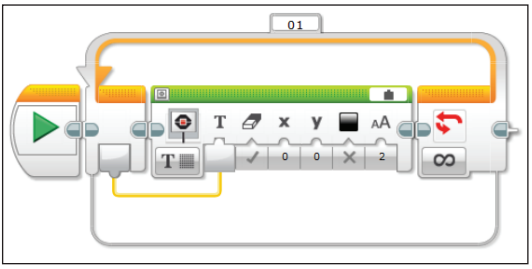
A 3-as gomb a ciklusváltozó értékét adja vissza.

A 4-es gomb segítségével szimbolikus nevet adhatunk a ciklusunknak, így hivatkozási alapot teremthetünk a ciklusra, amelyet később más blokkokban (például ciklusbefejező blokk) felhasználhatunk.

Végtelen ciklus

A végtelen ciklus olyan ciklus, melynek futása külső esemény bekövetkezte nélkül sohasem zárulna le. Egy ilyen külső esemény például a tégla Vissza (Back) gombjának a megnyomása, amellyel kilépünk a programból.

A 81. ábrán egy végtelen ciklust hoztunk létre úgy, hogy a módszelektort *végtelen*re (Unlimited) állítottuk. A ciklus a végtelenségig ismétlődik, és kiírja a robot képernyőjére a ciklusváltozó egyre növekedő értékeit. A ciklust csak a program bezárásával lehet leállítani, ellenkező esetben addig működik, ameddig a robotból ki nem fogy az elem.

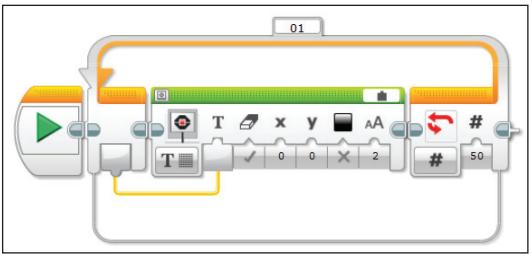


81. ábra. *Végtelen ciklus*

Rögzített lépésszámú ciklus

A rögzített lépésszámú ciklus szervezéséhez a módszelektorban válasszuk ki a *számol* (Count) beállítást. Így megjelenik egy gomb, amely segítségével megadhatjuk, hogy a ciklus hányszor iteráljon.

A 81. ábrán látható végtelen ciklust könnyű átírni rögzített lépésszámú ciklussá. A 82. ábrán látható ciklus 50-szer fogja kiírni a ciklusváltozó értékét, vagyis elszámol 0-től 49-ig. Megjegyzendő, hogy a ciklus automatikus ciklusváltozója mindig 0-tól indul.



82. ábra. *Rögzített lépésszámú ciklus*

Időciklusok

Lehetőség van időciklusok szervezésére is. Ha a módszelektorral az *idő* (Time) beállítást választjuk, akkor a ciklus a másodpercben megadott időegységig fog futni. Az eltelt időt mindig a ciklusmag végrehajtása után teszteli,

és ha az idő kisebb, mint a beállított érték, akkor még egyszer végrehajtja a ciklusmagot.

Változó lépésszámú ciklus

Logikai feltételhez kötött változó lépésszámú ciklust úgy tudunk szervezni, hogy a módszelektor *logikai* (Logic) beállítását választjuk. Így, hátultesztelős ciklus lévén, mindannyiszor végrehajtja a ciklusmagot, ameddig a megadott logikai feltétel hamis. Amint a logikai feltétel igazzá válik, a ciklus leáll. Vigyázat, mert ha a logikai feltételt úgy adjuk meg, hogy az mindig hamis, végtelen ciklusunk lesz!

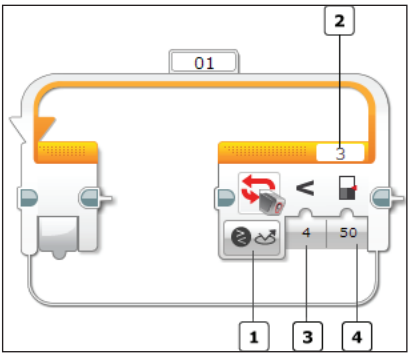
Megfigyelhető – mivel hátultesztelős ciklusunk van –, hogy a ciklusmag egyszer mindenképp végrehajtódik, mert a ciklusunk csak a végén teszteli a logikai feltételt. Elöltesztelős ciklus szervezésére nincs lehetőség, csak ha *elágazás* (Switch) utasítást használunk. Ennek használatával később ismerkedünk meg, de az elv az, hogy teszteljük a logikai feltételt, és ha az már eleve igaz, nem lépünk be a ciklusba.

Logikai feltételhez kötött ciklust kell használnunk akkor is, amikor egynél több érzékelő adataiból következtetve szeretnénk ismételni utasításokat, hisz a ciklusszervezésben csak egy érzékelő által szolgáltatott visszatérési érték felhasználása megengedett.

Érzékelők által vezérelt ciklusok

A ciklus blokk több olyan módot is tartalmaz, amely segítségével be lehet olvasni egy megadott szenzor értékét, és ezt össze lehet vetni (hasonlítani) egy megadott értékkel. A ciklus addig fog tartani, ameddig értékegyezés nem lesz.

Ebben az esetben a 83. ábrán látható a ciklus blokk általános alakja.



83. ábra. Érzékelők által vezérelt ciklusok

Az 1-es módszelektor segítségével tudjuk kiválasztani a ciklus típusát.

A 2-es gomb segítségével tudjuk kiválasztani az érzékelő portját (port szelektor).

A 3-as gombbal adhatjuk meg az összehasonlító műveletet (a 16. táblázat szerint).

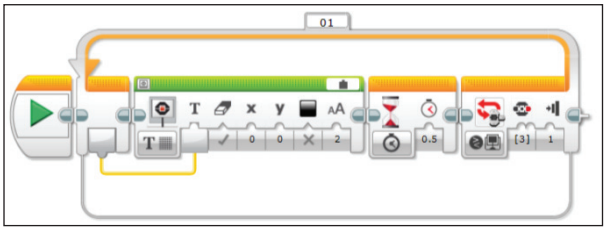
A 4-es gomb segítségével pedig a küszöbértéket állíthatjuk be.

A ciklus blokk a következő érzékelőket ismeri:

– téglagombok;
– színérzékelő;
– infravörös érzékelő;
– motorforgás;
– időzítő;
– érintésérzékelő;
– üzenet.

Értelemszerűen léteznek olyan érzékelők is, amelyeknél nincs összehasonlító művelet, például a színérzékelő szín módja esetén a ciklus akkor állhat le, amikor a színérzékelő egy adott színt érzékelt. Ez az érzékelt szín nem lehet egy küszöbérték, vagyis nincs például a pirosnál nagyobb vagy kisebb szín.

A 84. ábrán látható ciklus addig írja ki a ciklusváltozó értékeit, ameddig a középső téglagombot benyomott állapotban nem találja.

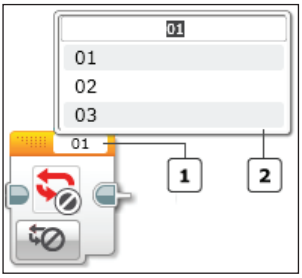


84. ábra. *Gombnyomásig ismétel*

3.1.22. A Ciklusbefejező blokk

A Ciklusbefejező blokk (Loop Interrupt Block) a megadott szimbolikus nevű ciklust fejezi be. Egyszerűen arra kényszeríti a vezérlést, hogy azonnal lépjen ki a ciklusból, és a program a ciklus utáni blokkal folytatódjon. A ciklusbefejező blokk a normális befejezésnél hamarabb, vagy más feltétel beteljesedésekor fejezi be a ciklust akár a cikluson belülről, akár bármilyen más, párhuzamosan futó programszekvenciából.

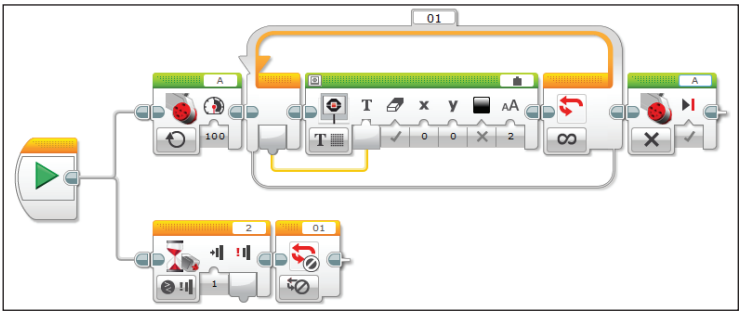
Az 1. gombon állíthatjuk be a ciklus szimbolikus nevét. Ezt kiválaszthatjuk a 2-es listából is, amelyben az összes program által használt ciklus megjelenik.



85. ábra. *A ciklusbefejező blokk*

A 86. ábrán látható program két párhuzamosan futó szekvenciára bomlik. Az első beindítja a nagy motort, majd egy végtelen ciklusban kiírja a ciklusváltozó értékeit a tégla képernyőjére. Elméletileg ez a ciklus végtelen, tehát az elem lemerüléséig forogna a nagy motor.

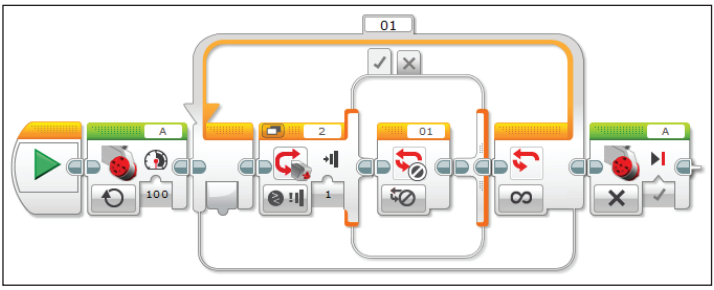
A második szekvencia az érintésérzékelő lenyomására vár. Ha ez az esemény megtörténik, akkor életbe lép a ciklusbefejező blokk, amely leállítja a 01-es végtelen ciklust, vagyis a vezérlés a ciklus utáni utasítással folytatódik, amely megállítja a nagy motort.



86. ábra. *Végtelen ciklus leállítása kívülről*

A 87. ábra azt mutatja be, hogyan használhatjuk a ciklusbefejező blokkot a ciklus belsejében.

A 87. ábrán látható program akkor fejezi be a ciklust, ha megnyomjuk az érintésérzékelőt. Ennek a tesztelésére egy logikai elágazást használtunk, amely beolvassa az érintésérzékelő állapotát, és ha ez le volt nyomva, akkor átadja a vezérlést a ciklusbefejező blokknak. Ha nem volt lenyomva, akkor nem történik semmi.



87. ábra. *Végtelen ciklus leállítása belülről*

3.1.23. Az elágazás

Az elágazás utasítások valósították meg először a futás pillanatában történő döntést bizonyos feltételek függvényében. Ennek a megvalósításnak köszönhető, hogy ugyanaz az algoritmus különböző bemeneti értékek, illetve részeredmények alapján, önmagából, más-más lineáris utasítássorozatot hajtson végre. Ettől az újítástól vált a lineárisan programozható algoritmust végrehajtó gép számítógéppé. Az elágazás megvalósítása Neumann Jánosnak tulajdonítható.

Az egyszerű elágazás egy logikai kifejezés Igaz vagy Hamis értékének függvényében vagy az egyik ágon, vagy a másik ágon hajtja végre a programban található blokkokat.

A többszintű elágazást megvalósító utasítást Wirth és Hoare vezette be 1966-ban. Szemantikai szerepe: több alternatíva közül egynek a kiválasztása.

A végrehajtandó alternatíva kiválasztása egy *szelektor*nak nevezett kifejezés alapján történik, és a szelektor-kifejezés megfelelő **értéke** alapján történik az elágazás. A szelektor-kifejezés megfelelő értékeit *esetek*nek (Case) nevezzük.

Az *elágazás blokk* (Switch block) komplex utasítás blokkja a LEGO MINDSTORMS EV3-nak.

A blokk úgy működik, hogy belépéskor elvégzi a kiválasztott tesztet, majd az eredmény függvényében végrehajtja a megfelelő esetet. Mindig egy és csakis egy eset fog végrehajtódni, ezután a program az elágazás blokk utáni blokkal fog folytatódni.

Ha a blokkban egy esetet üresen hagyunk, akkor azon az ágon a program nem fog csinálni semmit.

A blokkban egy esetet meg kell hogy jelöljünk *alapértelmezett*nek (Default Case), ez akkor fog végrehajtódni, ha a szelektor értéke nem talál egyetlen más esettel sem. Mivel ez a jelölés kötelező, ha azt akarjuk, hogy az alapértelmezett eset ne csináljon semmit, be kell szúrnunk egy üres esetet, és azt kell megjelölnünk alapértelmezettnek.

A 88. ábrán az elágazás blokk egyik változatát látjuk. Hasonló jelenik meg, amikor a palettáról behúzzuk a blokkot a program felületére.

Az 1. gomb a *módszelektor*, a 2. a *port szelektor*, a 3. pedig a megfelelő bemeneti értékek megadására szolgál.

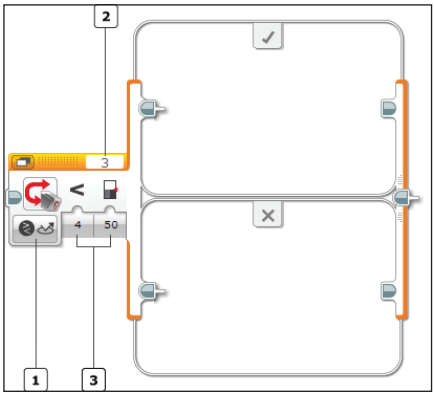
Az 1-es módszelektor segítségével ki tudjuk választani, hogy a blokk milyen tesztet hajtson végre az elején.

Tesztelni tudunk egyszerűen egy szöveges, numerikus vagy logikai értéket, vagy tesztelni tudunk egy érzékelőn mért értéket.

Az érzékelőket *mérés* (Measure) vagy *összehasonlítás* (Compare) módokban tudjuk használni.

Mérés módban a blokk a következőket tudja tesztelni:
– téglagombok;
– színérzékelő;
– infravörös érzékelő.

Összehasonlítás módban a blokk a következőket tudja tesztelni:
– téglagombok;
– színérzékelő;
– infravörös érzékelő;
– motorforgás;
– időzítő;
– érintésérzékelő;
– üzenet.



88. ábra. *Az elágazás blokk*

Megjegyezzük, hogy a Várj blokkal ellentétben az elágazás blokk nem vár addig, ameddig a szenzoron megjelenik a megfelelő érték. A blokk csak a tesztet hajtja végre, a program végrehajtása azonnal tovább is megy. Ha tehát azt

szeretnénk, hogy például a 88. ábrán látható program a „Lenyomva” szöveget jelentesse meg a tégla képernyőjén, akkor jobb, ha az érintésérzékelőt már a program indítása előtt lenyomjuk és nyomva is tartjuk.

Mivel az elágazás blokk grafikailag is nagyon komplex lehet (sok más blokk kerülhet bele), a jobb átláthatóság kedvéért kétfajta megjelenítési mód közül választhatunk.

A *nyílt nézet* (Flat View) a 87. ábrának megfelelően egymás alatt jeleníti meg az összes esetet. Nyilvánvaló, hogy sok esetben nem fog kiférni a képernyőre a blokk, viszont könnyen áttekinthető.

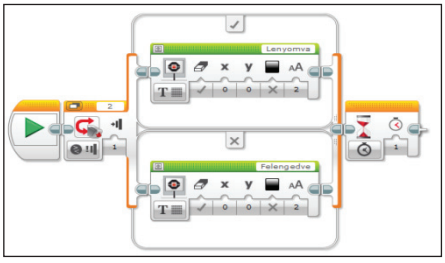
A *füles nézet* (Tabbed View) minden esetnek egy fülecskét hoz létre, és egyszerre csak egy eset látható a képernyőn, ha át akarunk térni egy másik esetre, akkor fülecskét kell váltani. Így a blokk könnyen kifér a képernyőre, csak nem annyira áttekinthető.

A két nézet között a blokk bal felső sarkában található  gombbal lehet váltani (Switch to Tabbed View / Switch to Flat View).

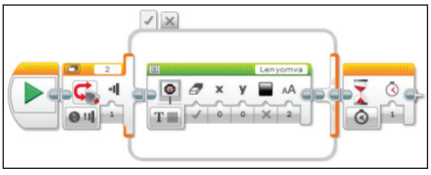
Nyilvánvaló, a program működését nem befolyásolja a két nézet, csak a grafikáját. Annak ellenére, hogy a füles nézetben egyszerre csak egy eset látszik, a többi is ugyanúgy része a programnak.

Szintén a grafikus kinézethez tartozik, hogy a blokkot tetszés szerint át lehet méretezni a kerületén lévő jelölő körök és négyzetek segítségével (90. ábra).

a)



b)

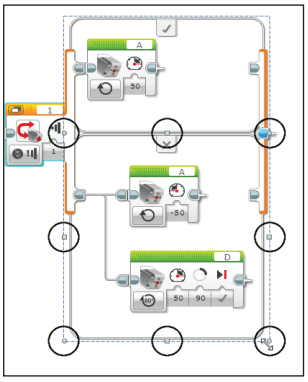


89. ábra. a) *nyílt és* b) *füles nézet*

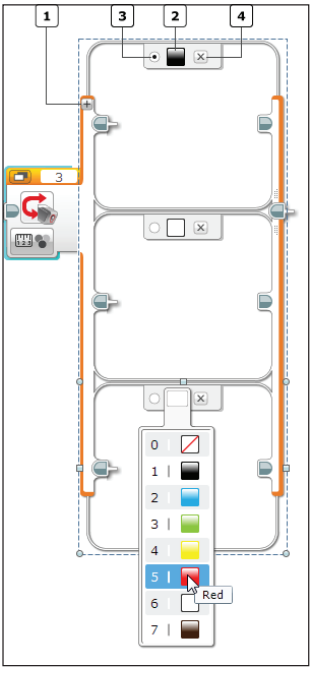
A logikai teszt igaz ágát a „pipa” ☑, a hamis ágát az „X” ☒ jelöli.

A 87. és 89. ábrákon megfigyelhettük, hogyan működik az elágazás egyszerű logikai esetekre. Nézzük most meg, hogyan működik numerikus értékekre.

Alapértelmezetten, ha ráhúzzuk a blokkot a programozási felületre, csak két eset jelenik meg. Például ha numerikus értékek esetén több esetet szeretnénk ágaztatni, akkor a blokkhoz újabb eseteket kell hozzáadnunk. Amint a 91. ábrán is látjuk, a blokk grafikus elemei lehetőséget biztosítanak újabb esetek hozzáadására, esetek kitörlésére, alapértelmezett eset beállítására stb.



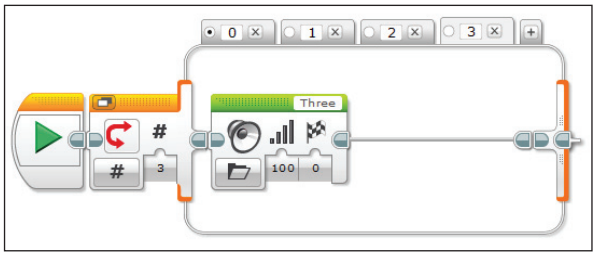
90. ábra. *A blokk átméretezése* [41]



91. ábra. *Esetek hozzáadása, törlése, kezelése* [41]

Az 1-es gomb segítségével új esetet adhatunk hozzá, a 2-es gombbal beállíthatjuk az esetet, a 3-as gomb segítségével mondhatjuk meg, hogy melyik eset legyen az alapértelmezett, a 4-essel pedig kitörölhetjük az esetet.

A 92. ábrán látható blokk a numerikus bemenet függvényében angolul kimondja a számjegyeket 0-ás, 1-es, 2-es vagy 3-as számjegyek esetén. Mivel a 0-ást alapértelmezett esetnek állítottuk be, ezért ha 3-nál nagyobb számjegyet adunk meg, s ez egyik esetre sem talál, csak a nullást fogja angolul kimondani.

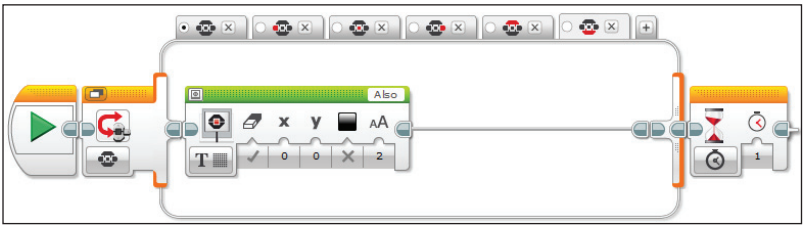


92. ábra. *Numerikus esetek*

Ugyanez történik, ha a blokkot szöveges üzemmódban használjuk. Ekkor a beérkező szöveget hasonlítja össze, s találat esetén lefut a megfelelő esethez tartozó kód.

Mérés módban a blokk tesztelni tudja a téglagombok, színérzékelő vagy az infravörös érzékelő szolgáltatta értékeket.

A 93. ábrán látható program a téglagombokat teszteli le, és kiírja, hogy a bal, a jobb, a középső, a felső vagy az alsó gomb volt lenyomva. Alapértelmezett esetben a képernyőn a „Nincs gomb” felirat jelenik meg.



93. ábra. *Téglagombok tesztelése*

Hasonlóan tudunk eljárni a színérzékelő vagy az infravörös érzékelő által visszaszolgáltatott értékekkel is. Mérés módban az infravörös érzékelő csupán a távirányító gombjainak értékeit tudja visszaszolgáltatni, ezeket lehet tesztelni.

A színeket a 91. ábrán látható módon tudjuk beállítani.

Legösszetettebb tesztek az összehasonlítás módban elérhető tesztek. Ezek a tesztek összehasonlításokon alapulnak, és Igaz vagy Hamis értékeket visszatérítő kétesetes tesztek. Itt a blokk tesztelni tudja, hogy egy megadott téglagomb volt-e megnyomva, vagy több megadott gombot együttesen nyomtunk le. Tesztelni tudja, hogy a színérzékelő adott színt vagy színeket érzékelt-e, a háttér vagy a visszavert fényerősség elért, meghaladott-e egy adott értéket, netán egyenlő vagy nem egyenlő-e egy adott értékkel. Bemenetként megadhatjuk az összehasonlítási műveletet (0 – egyenlő, 1 – nem egyenlő, 2 – nagyobb, 3 – nagyobb vagy egyenlő, 4 – kisebb,

5 – kisebb vagy egyenlő), valamint a küszöbértéket, amihez hasonlítjuk az érzékelő által visszaszolgáltatott értéket.

Az infravörös érzékelő esetén is megadhatjuk most már az erre vonatkozó összes mód (*közelségi mód*, *irányjeladó mód* és *távirányító mód*) szerinti küszöbértéket és összehasonlítási műveletet.

Ha a motor forgását szeretnénk tesztelni, beállíthatjuk a küszöbértéket, a relációs műveletet, valamint azt, hogy szög, fordulatszám vagy erősség értéket tesztelünk.

Ha egy időzítő értékét akarjuk tesztelni, akkor beállíthatjuk az időzítőt (ez az érték 1-től 8-ig változhat; 8 időzítőt tud kezelni a LEGO Mindstorms), a relációs műveletet, valamint a küszöbértéket.

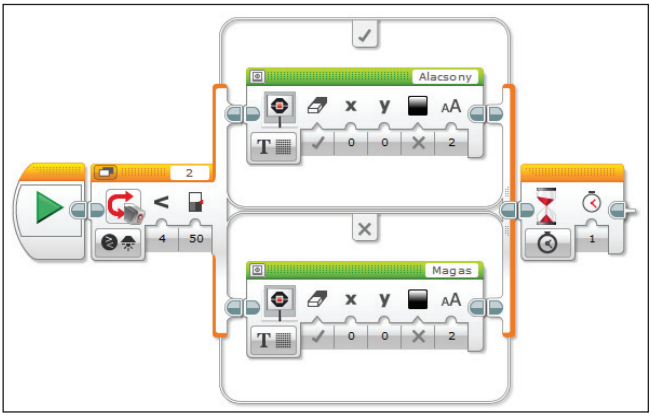
Ha az érintésérzékelőt teszteljük, beállíthatjuk az érintés módját, hogy *benyomott* (0 – Pressed), *felengedett* (1 – Relased) és *ütközött* (2 – Bumped) legyen az állapota.

A fenti esetek mindegyikében, ha valamilyen érzékelőt tesztelünk, akkor a blokkon megjelenik a *portszelektor* is, ahol beállíthatjuk az érzékelő csatlakozási portját.

Ha üzenetet tesztelünk, akkor a beérkezett üzenet szöveges, numerikus vagy logikai lehet. Az üzenet Bluetooth-csatornán érkezik. Szöveges üzemmódban azt állíthatjuk be, hogy a beérkezett üzenet legyen egyenlő vagy nem egyenlő egy megadott szöveggel. Numerikus üzenet esetében a relációs műveletek (0 – egyenlő, 1 – nem egyenlő, 2 – nagyobb, 3 – nagyobb vagy egyenlő, 4 – kisebb, 5 – kisebb vagy egyenlő) valamelyikét állíthatjuk be, valamint a küszöbértéket.

Logikai üzenet esetén közvetlenül a tesztesetek valamelyike jön be az Igaz vagy a Hamis ágon.

A 94. ábrán látható elágazás blokk a színérzékelő által mért háttérvilágítást hasonlítja össze az 50-es értékkel. Ha ennél kisebb, kiírja, hogy „Alacsony” a háttérvilágítás, ha nagyobb vagy egyenlő, akkor pedig kiírja, hogy „Magas” a háttérvilágítás.



94. ábra. *Háttérvilágítás tesztelése*

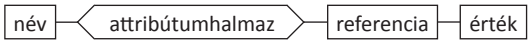
3.1.24. Változók és konstansok

A *változó* fogalma a matematikában egy értelmezési tartománnyal rendelkező, ebből bármilyen értéket felvehető objektum, melynek értéke logikailag határozatlan. Ugyanez a számítástechnikában egy memóriacímen levő memóriazónát jelent, amelynek tartalma mindig létezik, ez egy jól meghatározott érték, és fő jellemzője, hogy csak bizonyos algoritmusok által hozzáférhető és módosítható.

Egy változónak négy alapeleme van:

– *név,*
– *attribútumhalmaz,*
– *referencia,*
– *érték.*

Egy *változó neve* az illető nyelv által lexikálisan megengedett karaktersorozat, ez a változó azonosítója.



95. ábra. *Változók alapelemei*

Az *attribútumhalmaz* jellemzőket tartalmaz a változóról, például a változó típusát, a változó láthatósági területét, a változó élettartamát.

A *referencia* egy információ, amely megadja azt a fizikai vagy logikai helyet, amelynek tartalma a változó értéke.

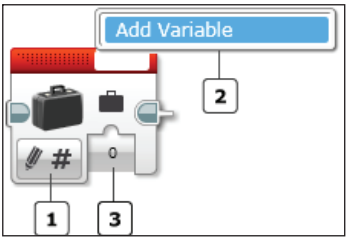
A változó negyedik alapeleme az **érték**: a program futása során a változónak ez a mezője változtatja az értékét. Egy változó értékének a kiolvasása a referencia tartalmának a kiolvasásaként történik. Egy változó értékének a megváltoztatása a referencia tartalmának felülírásaként történik. Az értékadás többnyire egy kifejezés kiértékelésének az eredménye, amely beíródik a változó referenciájának tartalmába.

Vizuális programozási nyelvekben, sajnos, kényelmetlenebb dolgozni változókkal, mint például imperatív nyelvekben.

A LEGO MINDSTORMS EV3-ban a változó a tégla memóriájának egy jól meghatározott helye, amely értéket képes tárolni. Ennek az értéknek a *szöveg*, *numerikus*, *logikai*, *numerikus tömb*, *logikai tömb* EV3 típusok valamelyike lehet a típusa.

A memóriazóna tartalmát, a változó értékét írni vagy *olvasni* lehet.

LEGO MINDSTORMS EV3 Home Editionben a változókat egy bőrönd jelképezi, a blokkon a változó nevét, típusát és értékét lehet beállítani, valamint azt, hogy írni vagy olvasni akarjuk a változót.



96. ábra. *Változó*

Az 1-es *módszelektor* segítségével azt tudjuk beállítani, hogy olvasni (Read) vagy írni (Write) szeretnénk a változót, majd a kiválasztott módban megadhatjuk a változó típusát.

A 2-es gomb segítségével megadhatjuk az új változó nevét, vagy név szerint kiválaszthatunk egy már létező változót a listából.

A változó nevében az angol ábécé nagy- és kisbetűi, valamint a szóköz, aláhúzásjel és mínusz karakterek szerepelhetnek.

Tehát egy lexikálisan elfogadott név a következő karakterekből állhat: „abcdefghiklmnopqrstvxyzABCDEFGHIKLMNOPQRSTVXYZ _-”.

A változó neve a fent említett karakterek bármelyikével kezdődhet, még szóközzel is.

Először mindig az 1-es gomb segítségével adjuk meg a típust, mert a 2-es gomb listájában csak a megfelelő típusú változók nevei jelennek meg!

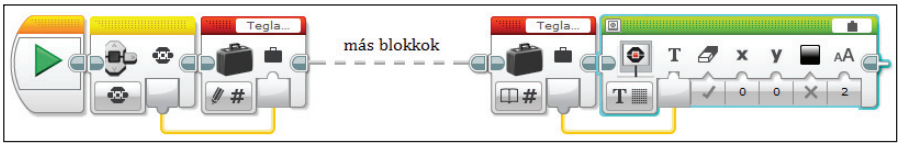
A 3-as gomb a változó értékét jelenti. Írásnál megadhatjuk ezt, olvasásnál innen olvashatjuk ki.

Ha egy változónak nem adunk értéket, a LEGO MINDSTORMS EV3 Home Edition a típusának megfelelő kezdőértékkel látja ezt el: a numerikus értékek kezdőértéke 0, a szövegeké az üres string, a logikai értékeké a false, a tömbök esetében pedig üres tömbbel inicializálja a változókat.

Akárhányszor adhatunk értéket egy változónak a program során, a változó értéke az utoljára beírt érték lesz. Az értéket megadhatjuk közvetlenül beírással vagy adatdrót segítségével is.

Ha létrehoztunk egy változót, a projekt összes programjában látható, használható és párhuzamosan elérhető lesz.

Sok esetben a változók használata megkerülhető az adatdrótok használatával, azonban ha hosszú a program, a változók használata olvashatóbbá teszi ezt, mint egy nagyon hosszú és kusza adatdrót.

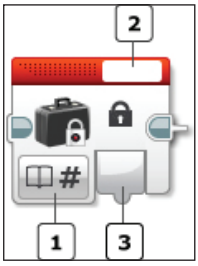


97. ábra. *Változó használata*

A változóktól eltérően a *konstansok* a program futása során megőrzik értéküket. Használatuk egyszerűbbé és kifejezőbbé teszi a programírást.

LEGO MINDSTORMS EV3 Home Editionben a konstansokat a változókhoz hasonló bőrönd jelképezi, ám egy lakat jelzi, hogy értéküket csak olvasni tudjuk. A konstansok értékkonstansok, vagyis külön azonosítóval nem kell ellátni őket, nevük nincs, csak maga az érték jelenti a konstanst.

Ha konstanst használunk, és megváltoztatjuk a blokkon az értéket, akkor a teljes programban mindenhol megváltozik a konstans értéke.

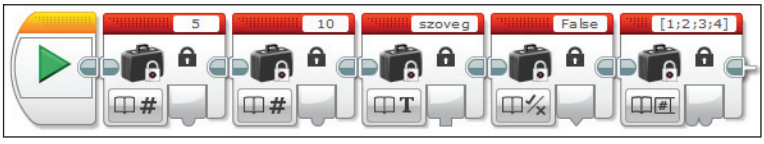


98. ábra. *Konstans*

Az 1-es *módszelektor* segítségével a típust tudjuk beállítani. A változókhoz hasonlóan a típus *szöveg*, *numerikus*, *logikai*, *numerikus tömb* vagy *logikai tömb* lehet.

A 2-es gomb segítségével a konstans értékét adhatjuk meg.

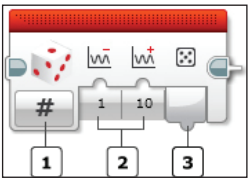
A 3-as gombról pedig az értéket olvashatjuk vissza.



99. ábra. *Konstansok*

3.1.25. A véletlenszám-generátor

A Random blokk véletlen számokat generál. Tulajdonképpen pszeudovéletlen számokról van szó, hisz a processzor működése determinisztikus, és az előállított számok véletlenszerűek, de mégis megfelelnek bizonyos matematikai szabályoknak. Elég sokszor lefuttatva a számítást, előbb-utóbb ismétlésbe botlunk.



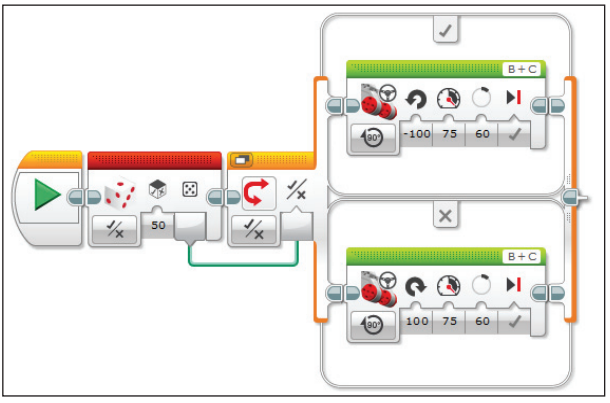
100. ábra. *Véletlenszám-generátor*

Az 1-es *módszelektor* gomb segítségével a véletlen szám típusát választhatjuk ki. Ez numerikus vagy logikai lehet.

Ha numerikus típust választunk, akkor a 2-es gombok segítségével megadhatjuk annak az intervallumnak az alsó és a felső határát, amelyből a véletlen számot kérjük. A megadott tartományon belül minden egyes értéket azonos valószínűséggel választ ki a generátor, tehát a változó normális eloszlású.

Ha logikai típusra kérünk véletlen eredményt, akkor a 2-es gomb segítségével az Igaz (True) válasz valószínűségét adhatjuk meg százalékban.

A 3-as gombról a generált értéket olvashatjuk le.



101. ábra. *Jobbra vagy balra fordul?*

A 101. ábrán látható program 50%-os Igaz valószínűséggel generáltat egy véletlen logikai értéket. Ha az érték Igaz (True), akkor a robot balra, ha Hamis (False), akkor a robot jobbra tér 60 fokos szögben, 75%-os motorerővel.

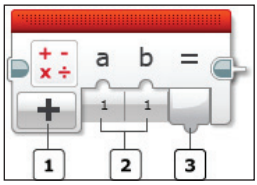
3.1.26. Műveletek

A LEGO MINDSTORMS EV3 Home Edition blokkokat biztosít a következő műveletek elvégzésére:

– matematikai;
– logikai;
– szöveg;
– tömb;
– összehasonlító;
– intervallumteszt;
– kerekítés.

Matematikai műveletek

A matematikai műveletek blokkja (Math) egyszerű matematikai műveleteket végez el a megadott bemeneten, az eredményt pedig megjelenteti a kimeneten.



102. ábra. *Matematikai műveletek*

Az 1-es *módszelektor* gomb segítségével a műveletet választhatjuk ki, ez összeadás, kivonás, szorzás, osztás, abszolút érték, négyzetgyök, hatványozás lehet, illetve az ADV lehetőség kiválasztásával tetszőleges, legtöbb négy változót használó matematikai kifejezést is megadhatunk.

Ha összeadást, kivonást, szorzást, osztást vagy hatványozást választunk, akkor a 2-es gomb segítségével megadhatjuk a műveletekhez szükséges két operandust.

Az abszolút érték és a négyzetgyökvonás egy operandust vár.

A 3-as gombon a művelet eredményét kapjuk meg.

Érdekességként megjegyezzük, hogy a nem értelmezett műveletek (pl. zéróval való osztás, negatív számból gyökvonás stb.) eredménye egy hiba, ám ezt

a hibát, ha bemenetként adjuk meg egy másik blokk számára, akkor az 0-nak értelmezi és veszi.

Érdekes például az is, hogy $-1 \times 0 = -0$.

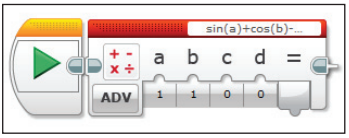
ADV módban legtöbb négyváltozós matematikai kifejezéseket adhatunk meg. Az összeadás, kivonás, szorzás, osztás, maradékképzés, előjelváltás műveletek mellett kerekítő függvényeket (Floor, Ceil, Round), abszolút értéket, tízes és természetes alapú logaritmusokat, szinusz, koszinusz, tangens, arkusz-szinusz, arkusz-koszinusz, arkusz-tangens szögfüggvényeket, valamint négyzetgyökvonást is használhatunk. Zárójelezéssel megváltoztathatjuk a műveletek prioritását is.

Ami a Floor, Ceil, Round kerekítési függvényeket illeti, a következő különbségekről beszélünk:

A Ceil függvény azt a legkisebb egész számot adja vissza, amely nem kisebb az argumentumnál (*ceiling*: mennyezet). Például: Ceil (3,1) = 4; Ceil(5) = 5; Ceil(−3,9) = −3. A függvény tehát felfele kerekít minden esetben.

A Floor függvény azt a legnagyobb egész számot adja vissza, amely nem nagyobb az argumentumnál (*floor*: padló). Például: Floor(3,9) = 3; Floor(5) = 5; Floor(−3,1) = −4. A függvény tehát lefele kerekít minden esetben. Ez a függvény megfelel a matematika *egészrész* függvényének.

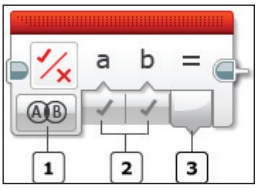
A Round függvény az argumentumként megadott kifejezést a legközelebbi egészre kerekíti, vagyis …,5 alatt lefele, …,5 felett felfele kerekít.



103. ábra. *Tetszőleges matematikai kifejezés*

Logikai műveletek

A logikai műveletek blokk (Logic Operations) az és (And), vagy (Or), kizáró vagy (Xor) és nem (Not) logikai műveletek elvégzésére szolgál.



104. ábra. *Logikai műveletek*

Az 1-es *módszelektor* segítségével a megfelelő műveletet választhatjuk ki. Az És, Vagy, valamint Kizáró Vagy műveletek kétoperandusúak, a Nem egyoperandusú. Az operandusokat a 2-es gombok segítségével lehet megadni, a 3-as gombon pedig megkapjuk a művelet eredményét.

A négy logikai műveletet a következő művelettáblák írják le. Az És eredménye akkor Igaz, ha mindkét operandus Igaz, a Vagy eredménye akkor Hamis, ha mindkét operandus Hamis, a Kizáró Vagy akkor Igaz, ha csak az egyik operandus Igaz, a Nem pedig megfordítja az argumentuma igazságértékét.

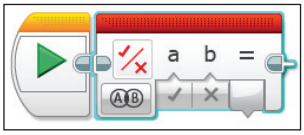
22. táblázat. *Logikai műveletek igazságtáblázatai*

A	B	A És B
igaz	igaz	igaz
igaz	hamis	hamis
hamis	igaz	hamis
hamis	hamis	hamis

A	B	A Vagy B
igaz	igaz	igaz
igaz	hamis	igaz
hamis	igaz	igaz
hamis	hamis	hamis

A	Nem A
igaz	hamis
hamis	igaz

A	B	A Kiz. Vagy B
igaz	igaz	hamis
igaz	hamis	igaz
hamis	igaz	igaz
hamis	hamis	hamis

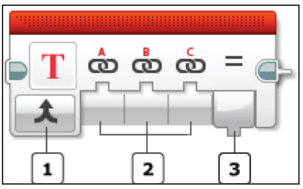


105. ábra. *Az És művelet*

Szövegműveletek

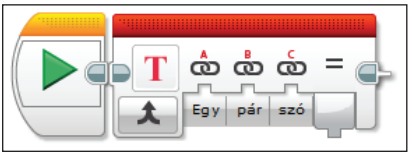
Szövegekkel az összefűzés (konkatenálás) művelete végezhető el.

A Szöveg (Text) blokk legtöbb három szöveget tud egy szöveggé fűzni úgy, hogy egymás után másolja a karakterláncokat.



106. ábra. *A Szöveg blokk*

Az 1-es *módszelektor*nak itt igazán nincs is szerepe, mert más művelet nem választható ki, a 2-es gombokon legtöbb 3 argumentum adható meg, a 3-as gomb pedig az eredményt, az összefűzött karakterláncot, szöveget adja vissza.

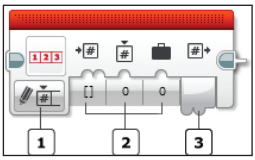


107. ábra. *Szövegek összefűzése*

Tömbműveletek

A Tömbműveletek (Array Operations) blokk segítségével tömbökhöz tudunk elemet adni, egy adott indexű elemet tudunk írni vagy olvasni, illetve tömbök hosszát tudjuk megállapítani.

A műveleteket numerikus vagy logikai elemeket tartalmazó tömbökön tudjuk elvégezni.



108. ábra. *Tömbműveletek*

Az 1-es *módszelektor* segítségével választhatjuk ki a kívánt műveletet (hozzáadás, írás, olvasás, hossz), valamint azt, hogy milyen típusú (numerikus, logikai) tömbökkel dolgozunk.

A 2-es gomb segítségével a bemeneti paramétereket adhatjuk meg.

Például ha a hozzáadást választjuk, akkor meg kell adjunk egy tömböt, valamint egy elemet, amelyet hozzáadunk a tömbhöz. Ha a bemeneti tömböt adatdrót segítségével adjuk meg, akkor az nem változik a hozzáadás során, hanem egy új tömböt hoz létre az eredeti tömb alapján, amelyhez hozzáadja a megadott értéket. Ha egy adott indexű elemet akarunk kiolvasni a tömbből, akkor megadjuk a tömböt, valamint a kívánt indexet, az eredmény pedig az adott indexű elem értéke lesz. Ha írni akarunk egy kívánt indexű elemet, akkor megadjuk a tömböt, az indexet, valamint az új értéket, amit beleír az eredmény tömbbe. A tömb hosszánál megadjuk a tömböt, és a blokk visszatéríti ennek a hosszát.

A 3-as gombon kapjuk meg az eredményt.

A LEGO MINDSTORMS EV3 Home Edition 0 indexalapú tömbökkel dolgozik, vagyis egy *n* elemű tömb utolsó elemének az indexe n–1.

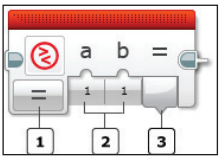
Egy üres tömb hossza 0.

Ha nem létező indexet adunk meg, a tégla hibát jelez.

109. ábra. *Adott indexű elem olvasása*

Összehasonlító műveletek

Az összehasonlító műveletek (Compare) blokk segítségével eldönthetjük, hogy két érték egyenlő, nem egyenlő, kisebb, nagyobb, kisebb vagy egyenlő, nagyobb vagy egyenlő.



110. ábra. Összehasonlítás

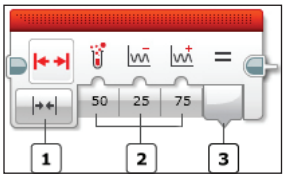
Az 1-es *módszelektor* a hat összehasonlító művelet valamelyike lehet (egyenlő, nem egyenlő, kisebb, nagyobb, kisebb vagy egyenlő, nagyobb vagy egyenlő), a 2-es gomb segítségével a két argumentumot (összehasonlítandó értéket) adjuk meg, a 3-as gomb pedig logikai értékként (Igaz vagy Hamis) visszatéríti az eredményt.

23. táblázat. *Összehasonlító műveletek*

Mód	Jelentés	Bemenet	Kimenet
=	egyenlő	a, b	igaz, ha a = b
≠	nem egyenlő	a, b	igaz, ha a ≠ b
>	nagyobb	a, b	igaz, ha a > b
<	kisebb	a, b	igaz, ha a < b
≥	nagyobb vagy egyenlő	a, b	igaz, ha a ≥ b
≤	kisebb vagy egyenlő	a, b	igaz, ha a ≤ b

Intervallumteszt

Az Intervallumteszt (Range) blokk ellenőrzi, hogy egy megadott szám egy megadott intervallumon belül vagy kívül esik. Az intervallumot az alsó és a felső határának megadásával tudjuk specifikálni.



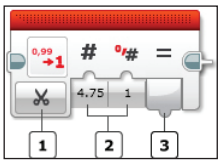
111. ábra. *Intervallumteszt*

Az 1-es *módszelektor* segítségével megadhatjuk, hogy a határokon belül vagy kívül akarunk-e tesztelni. A 2-es gomb segítségével az értéket, valamint az alsó és a felső határt adhatjuk meg, a 3-as gombon Igaz vagy Hamis értékkel megkapjuk a teszt eredményét.

A teszt alul, felül zárt intervallumot vesz. Tehát például az $50 \in [50,75]$ teszt eredménye Igaz (True).

Kerekítés

A Kerekítés (Round) blokk különböző kerekítési módszereket implementál. Egy tizedes számot egésszé kerekíthetünk le, fel vagy a legközelebbi egészhez, illetve megadott tizedesre kerekíthetünk segítségével.



112. ábra. *Kerekítés*

Az 1-es *módszelektor* segítségével a kerekítés módját adhatjuk meg. Ez a legközelebbi egészhez (To Nearest – Round), felkerekítés (Roud Up – Ceil), lekerekítés (Round Down – Floor) vagy tetszőleges tizedesre való kerekítés (Truncate).

A 2-es gomb a bemeneti érték, valamint Truncate esetében a kívánt tizedesek száma is.

A 3-as gombon az eredményt kapjuk meg.

24. táblázat. *Példa a Truncate-ra*

Bemenet	Tizedesek száma	Kimenet
1,253	0	1
1,253	1	1,2
1,253	2	1,25
1,253	6	1,253

3.1.27. Állományok

Az informatikában *adatállomány*nak, állománynak vagy *fájl*nak nevezzük a logikailag összefüggő adatok halmazát, tömbjét. Tárolásuk bármilyen adathordozón történhet.

Minden állomány rendelkezik azonosítóval, mely alapján megkülönböztetjük őket. Ez az azonosító egy névből, egy kiterjesztésből és a kettőt elválasztó karakterből (pont) áll. Az azonosításnak különböző operációs rendszereken eltérő szabályai vannak.

A *szövegállomány* vagy *szöveges állomány* sorokba rendezett, a sorokat a CR/LF karakterek zárják, az állományt pedig Ctrl-Z (például Windows operációs rendszerek alatt). A hozzáférés szekvenciálisan történik, az írás és az olvasás csak külön-külön történhet. Minden állomány megnyitása után rendelkezik egy állomány mutatóval. Ez az aktuális pozíciót mutatja az állományban, írás vagy olvasás után ez elmozdul.

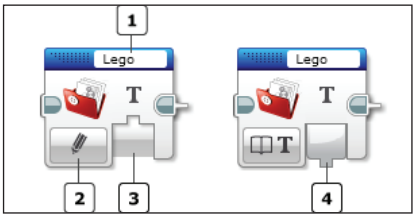
Az EV3 tégla esetén az állomány-hozzáférés blokk teszi lehetővé az adatok állományokból való olvasását és írását.

Használat után az állományt be kell zárni.

Ha egyszer létrehoztunk egy állományt, azután akárhányszor elérhető lesz, és az állomány-hozzáférési blokk vagy a memóriaböngésző segítségével használni is tudjuk.

Az EV3 tégla automatikusan .rtf kiterjesztést ad az állományoknak, így ezeknek más kiterjesztésük nem lehet. Ha kiterjesztést is megadunk az állománynévnél (113. ábra 1-es gomb), akkor a tégla vagy végtelen ciklusba kerül, vagy *FILE NAME ERROR! (Állománynév hiba!)* hibaüzenettel leáll. Állománynévnek lehetőleg csak az angol ábécé betűit, számokat, illetve aláhúzásjelt („_”) adjunk meg. Ha más karaktert adunk, a tégla jobb esetben *FILE NAME ERROR! (Állománynév hiba!)* hibaüzenettel leáll, rosszabb esetben valamilyen átkódolt nevű állományt hoz létre, amit le sem tudunk törölni, csak úgy, ha letöröljük a teljes projekt mappát, amiben a program és az állomány is van.

Ha nem létező nevű állományból akarunk olvasni, vagy törlést, bezárást hajtunk végre, az eredmény egy hibaüzenet lesz.



113. ábra. Állományok

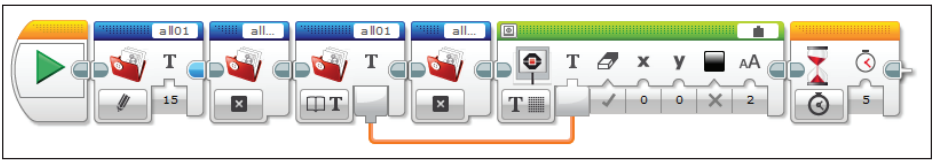
A 113. ábrán látható blokk esetében az 1-es gombon az állomány nevét állíthatjuk be. A 2-es a *módszelektor*, itt az írás, olvasás, törlés, bezárás módokat állíthatjuk be. A 3-as gomb az input, a bemenetel, a 4-es gomb pedig az output, a kimenetel.

A *törlés mód* végérvényesen letörli a háttértárolóról a megadott nevű állományt. Ha egy állományt újra akarunk írni, teljesen új adatokat akarunk felvezetni, akkor először töröljük ki a meglévő állományt, majd írjuk be az új adatokat.

A *bezárás mód* segítségével tudjuk bezárni a megnyitott állományt, ha már nincs szükségünk rá, megtörtént az adatok olvasása vagy írása.

Az *olvasás mód* segítségével szöveges vagy numerikus értéket olvashatunk az állományból. Vigyázzunk, mert a numerikus értéket ki tudjuk bármikor olvasni szövegként, de ha szöveget akarunk kiolvasni numerikus értékként, akkor az operációs rendszer és a processzor kódolásának megfelelő ábrázolási módban kapunk vissza értékeket, legjobb esetben 0-át, rossz esetben pedig az EV3 tégla nem várt módon viselkedhet.

Írás *mód*ban szöveges adatokat írhatunk az állományba. Ha a megadott nevű állomány nem létezik, a blokk létrehozza ezt. Ha az állomány létezik, akkor a végére fogja beírni az adatokat. Az írás mód soha nem törli ki az állományt.



114. ábra. *Példaprogram – állományok*

A 114. ábrán látható példaprogram létrehozza az *all01* nevű állományt, amelynek a neve a Memory Browserben *all01.rtf* lesz, majd beleírja (szövegként) a 15-ös értéket. Az állomány bezárása után szintén szövegként kiolvassa ezt az értéket, majd ismét bezárja az állományt. A kiolvasott 15-ös értéket kiírja a tégla képernyőjére, és 5 másodperc múlva befejezi futását.

3.1.28. Kommunikáció

Az EV3 téglák közötti, illetve a számítógép – EV3 tégla vagy mobiltelefon, tablett – EV3 tégla közötti drót nélküli (wireless) komunikáció Bluetooth vagy WiFi segítségével valósulhat meg.

A WiFi, az IEEE által kifejlesztett vezeték nélküli mikrohullámú kommunikációt (WLAN) megvalósító, széles körűen elterjedt szabvány népszerű neve. A WiFi, az elterjedt nézetekkel szemben, nem az angol Wireless Fidelity kifejezésnek a rövidítése. Az elnevezést egy marketingcég találta ki, játékosan utalva a HiFi szóra, csak később igyekeztek rövidítésként aposztrofálni és úgy reklámozni.

Az EV3 tégla alapból nem tud WiFi-kommunikációt megvalósítani, kell hozzá egy USB-csatlakozású NetGear WNA1100 típusú külső WiFi-modul.

Az EV3 tégla alapból Bluetooth-kapcsolatot tud létesíteni.

A Bluetooth rövid hatótávolságú, adatcseréhez használt, nyílt, vezeték nélküli szabvány. Alkalmazásával számítógépek, mobiltelefonok és egyéb készülékek között automatikusan létesíthetünk kis hatótávolságú rádiós kapcsolatot.

A név Harald Blåtand (I. Harald dán király) nevének angol változata, aki 958-tól, illetve 976-tól 986-ig volt Dánia és Norvégia uralkodója, és nagyon szerette az áfonyát, ezért kékek voltak a fogai. Harald arról volt nevezetes, hogy egyesítette a lázongó dán, norvég és svéd törzseket. Ehhez hasonlóan a Bluetoothot is arra szánták, hogy egyesítsen és összekössön olyan különböző eszközöket, mint a számítógép vagy a mobiltelefon. A Bluetooth logója a H és B betűknek megfelelő skandináv rúnákat, a *Haglaz*t és a *Berkanan*t idézi.

A Bluetooth-kapcsolat árnyoldala, hogy személyes adatainkhoz olyanok is hozzáférhetnek, akiknek nem akartuk ezt megengedni.

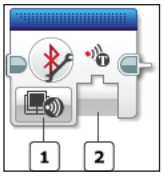
Néhány könnyen betartható biztonsági tanács:

– Csak akkor engedélyezzük a Bluetooth használatát az eszközben (például telefonban), amikor használni akarjuk, és használat után tiltsuk azt le.
– Használjunk hosszú, nehezen kitalálható számkódot az eszközök párosításához (használjunk 8-jegyű vagy még hosszabb számot – az „1234” kód nem jó).
– Párosítás után az eszköz legyen „rejtett” (hidden) állapotban, így is működni fog a már párosított másik eszközzel.
– Utasítsunk vissza minden ismeretlen kapcsolódási kísérletet.
– Engedélyezzük a titkosítást (encryption).
– Időnként nézzük meg a párosított eszközök listáját, nincs-e köztünk olyan, amit nem mi állítottunk be.
– Frissítsük a mobilunk firmware-szoftverét a legújabb verzióra a gyártó honlapjáról.

A Bluetooth-kapcsolat blokk segítségével kapcsolhatjuk be vagy ki a Bluetoothot, csatlakozhatunk egy másik Bluetooth-eszközhöz, vagy lezárhatjuk a kapcsolatot. A Bluetooth-eszközök közé tartoznak az EV3 téglák, mobiltelefonok és számítógépek is, noha nem minden Bluetooth-eszköz támogatja az EV3 téglával való kapcsolatot. Ha már létrehoztuk a Bluetooth-kapcsolatot az EV3 téglagombos menüje segítségével, akkor nem kell a Bluetooth-kapcsolat blokkot használjuk a programban.

Az EV3 rendszer Bluetooth-protokollja Master/Slave (mester/szolga) elven működik. Kiválasztjuk a Master EV3 téglát, és ezt használva csatlakozunk a Slave EV3 téglához. Egy Master EV3 tégla akár 7 Slave EV3 téglához is csatlakozhat. A Master EV3 tégla üzenetet küldhet minden Slave téglának, a Slave téglák azonban csak a Master téglának küldhetnek üzenetet, egymás között a Slave téglák nem tudnak közvetlenül kommunikálni [44].

Ha létrejött a Bluetooth-kapcsolat, akkor üzenetekkel tudunk kommunikálni.



115. ábra. *A Bluetooth-kapcsolat blokk*

A 115. ábrán látható Bluetooth-kapcsolat blokk 1-es gombja a *módszelektor*, 2-es gombja pedig a *bemenet*.

A módszelektor segítségével On (be), Off (ki), Initiate (kezdeményezés), illetve Clear (törlés) módokat tudunk beállítani.

A *be mód* bekapcsolja a Bluetooth-kommunikációt.

A *ki mód* kikapcsolja a Bluetooth-kommunikációt.

A *kezdeményezés mód* segítségével kapcsolatot kezdeményezhetünk egy meghatározott Bluetooth-eszközzel. Az eszköz nevét a bemenet gombon kell megadni. A Bluetooth-kapcsolat mindaddig fennmarad, amíg le nem zárjuk a törlés móddal.

A *törlés mód*dal kapcsolatot zárhatunk le egy megadott nevű Bluetooth-eszközzel.

Üzenetküldés

A 116. ábrán látható üzenetek blokk segítségével Bluetooth-üzeneteket küldhetünk az EV3 téglák között. Üzenet küldéséhez vagy fogadásához először az EV3 téglákat kell csatlakoztatni, akár a tégla Bluetooth menüje, akár a Bluetooth-kapcsolat blokk segítségével.

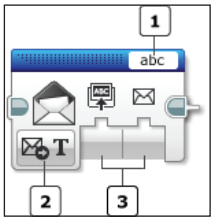
Az 1-es gomb tartalmazza az üzenet *azonosító*ját, a 2-es gomb a *módszelektor*, a 3-as gomb a *bemenetek*.

Az üzenet *azonosítója* egy szöveges címke, amely egyértelműen azonosítja az üzenetet.

Minden üzenetet három összetevő jellemez:

1. Azok a téglák, amelyek között az üzenet átment.
2. Az üzenet azonosítója.
3. Az üzenet értéke. Ez lehet szöveges, numerikus vagy logikai.

Az üzenet azonosítója lehetővé teszi, hogy egyszerre több adatot küldjünk az EV3 téglák között. Például egy tégla a „sebesseg” és a „szog” azonosítójú üzeneteket (adatokat) egyszerre küldi ugyanabban a programban.



116. ábra. *Az üzenetek blokk*

A *módszelektor* segítségével a küldés, fogadás, valamint összehasonlítás üzemmódokat állíthatjuk be. Az üzemmód kiválasztása után kiválaszthatjuk a bemenetek értékeit. A rendelkezésre álló bemenetek az üzemmódtól függően változnak.

Küldés üzemmódban szöveges, numerikus vagy logikai adatot küldhetünk a megadott téglának. Ebben az esetben bemenetként meg kell adni a fogadó tégla nevét, valamint az üzenet értékét (az adatot).

Fogadás üzemmódban a tégla szöveges, numerikus vagy logikai üzenetet kaphat egy Bluetooth segítségével csatlakoztatott EV3 téglától. Ekkor a blokknak egyetlen kimenete van (a bemenet kimenetté válik), mégpedig a kapott üzenet. A kimenet False (hamis) mindaddig, amíg meg nem érkezik az üzenet.

Az ö*sszehasonlítás* üzemmód összehasonlítja a kapott üzenetet (szöveges, numerikus vagy logikai) egy meglévő értékkel. Ha a feltételek teljesülnek, akkor a kimenet igaz (True) lesz, ha a feltételek nem teljesülnek, a kimenet hamis (False) lesz.

A szöveges és numerikus összehasonlítás esetében a blokknak két-két bemenete és két-két kimenete van.

Az egyik bemenet az összehasonlító művelet.

Szövegek összehasonlítására csak az *egyenlő* (0), illetve a *nem egyenlő* (1) relációs műveleteket használhatjuk, numerikus értékek összehasonlítására a 23. táblázatban is szereplő *egyenlő* (0), *nem egyenlő* (1), *nagyobb* (2), *nagyobb vagy egyenlő* (3), *kisebb* (4), *kisebb vagy egyenlő* (5) relációs műveletek állnak a rendelkezésünkre.

A másik bemenet az összehasonlítandó érték.

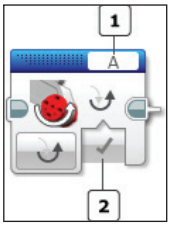
Az egyik kimenet az összehasonlítás eredménye (igaz vagy hamis), a másik kimenet pedig maga az üzenet.

Logikai összehasonlítás esetében a blokknak csak két kimenete van: az összehasonlítás eredménye, illetve maga az üzenet.

3.1.29. Sajátos motorblokkok

A kék fülben két sajátos motorblokk található, a motorinvertálás, illetve a szabályozatlan motor.

A 117. ábrán látható motorinvertálás blokk egyszerűen megváltoztatja a motor forgásirányát. Ha a motor eddig az óramutatóval megegyező forgásirányban haladt, ezentúl az óramutatóval ellentétes lesz a forgásiránya, és fordítva.

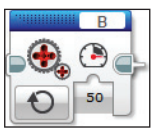


117. ábra. *A motorinvertálás blokk*

Az 1-es gomb segítségével a *port*ot állíthatjuk be (A, B, C, D valamelyike), a 2-es gomb pedig a *bemenet*, ahol megadhatjuk a forgásirányt. Ha az inverz bemenet igaz (True), akkor a kiválasztott motor normál „előre" és „hátra" irányt vált. A motorinvertálás blokk után minden olyan programblokk, amely a motort az óramutató járásával megegyező irányba fordítaná, a motort az óramutató járásával ellentétes irányba fogja fordítani, és fordítva.

Ha egy motor irányát megváltoztattuk, az megváltozva marad mindaddig, amíg egy másik motorinvertálás blokk vissza nem állítja a hamis (False) bemeneti értékkel.

A 118. ábrán látható szabályozatlan motor blokk a közepes motort és a nagy motort is vezérelni tudja. Segítségével bekapcsolhatjuk a megadott porton lévő motort, és ellenőrizni tudjuk a motor teljesítményszintjét.

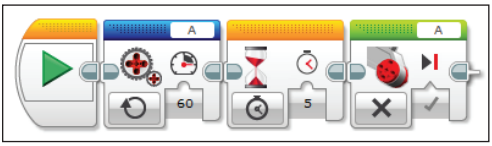


118. ábra. *Szabályozatlan motor blokk*

A szabályozatlan motor blokk nem tartalmazza a motor automatikus vezérlését. A megadott teljesítmény bemenet csak a motor vezérlésére szolgál. A motor sebességét és irányát a tápfeszültség bemenet segítségével szabályozhatjuk. A motor mindaddig forog, amíg meg nem állítja egy másik motorblokk, vagy amíg a program véget nem ér.

A teljesítményvezérlés kompenzálja a motor által tapasztalt bármilyen ellenállást vagy csúszást, az energiagazdálkodás pedig lehetőség szerint megpróbálja kompenzálni az akkumulátor szintjét.

A 119. ábrán látható program bekapcsolja a motort, 5 másodpercig forgatja, majd a nagy motor blokk segítségével megállítja.



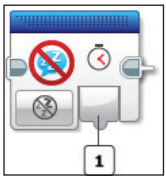
119. ábra. A *szabályozatlan motor blokk használata*

3.1.30. További lehetőségek

A kék fül még három további sajátos lehetőséget tartalmaz: a virrasztás blokkot, a nyers érzékelő érték blokkot, valamint a programleállítás blokkot.

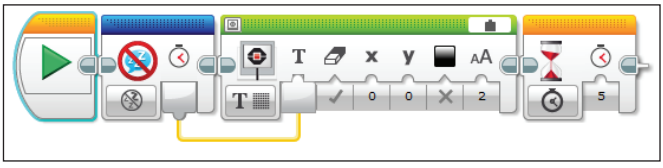
A 120. ábrán látható virrasztás blokk visszaadja az EV3 tégla várakozási idejét. Akkor használjuk ezt a blokkot, ha a programnak hosszabb ideig kell várnia, mint az EV3 tégla várakozási ideje, amelyet a tégla interfésze segítségével tudunk beállítani.

A blokknak egyetlen kimenete van, amely azt mutatja, hogy hány milliszekundum marad az EV3 tégla leállásáig (az alvásig hátralévő idő). Mivel a virrasztás blokk azonnal elindítja a tégla alvási időzítőjét, az alvási idő meg fog egyezni az alvás beállítással.



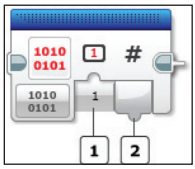
120. ábra. A *virrasztás blokk*

A 121. ábrán látható program kiírja, hogy az EV3 tégla hány milliszekundum múlva fog leállni.



121. ábra. A *virrasztás blokk használata*

A 122. ábrán látható nyers érzékelő érték blokk megadja a fel nem dolgozott érzékelőértéket, ami egy 0 és 1023 közötti szám.



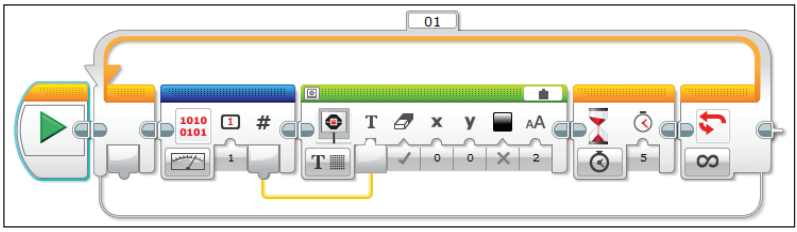
122. ábra. A *nyers érzékelő blokk*

A blokknak egy bemenete és egy kimenete van. A bemenet a *port* (1-es gomb), a kimenet pedig egy érték (2-es gomb).

Minden érzékelő nyers értékeket szolgáltat vissza. A programozási blokkok ezeket az értékeket veszik át, és alakítják át őket több információt szolgáltató számmá.

Például az érintésérzékelő, ha nincs benyomva, egy 160 körüli nyers értéket térít vissza, ha pedig be van nyomva, egy 3360 körüli értéket. Ezt a nyers értéket dolgozza fel az érintésérzékelő programozási blokkja, és adja vissza azt, hogy az érzékelő be volt-e nyomva vagy sem.

A nyers érzékelő érték blokk akkor hasznos, ha egy olyan érzékelőt szeretnénk használni, amelyik nem rendelkezik még programozási blokkal, például egy harmadik fél, vagy akár általunk gyártott érzékelőt. Így használni tudjuk például a LEGO MINDSTORMS EV3 Home Edition szoftvert anélkül, hogy az érzékelőnek programozási blokkja lenne.



123. ábra. A *nyers érzékelő blokk használata*

A 123. ábra egy olyan programot mutat be, amely az 1-es porton lévő érintésérzékelő nyers adatait írja ki a tégla képernyőjére.

A 124. ábra a programleállítás blokkot mutatja be.

124. ábra. *A programleállítás blokk*

A programleállítás blokk azonnal lezárja az összes programozási blokkot és befejezi a programot.

Programleállítási blokkot bárhova tehetünk, elágazásokba, ciklusokba is. Ha a vezérlés elérte ezt a blokkot, a program azonnal leáll, így egy tetszőleges program végére nem érdemes ilyen blokkot tenni, mert ott a futás így is, úgy is leállna.

3.1.31. Saját blokkok

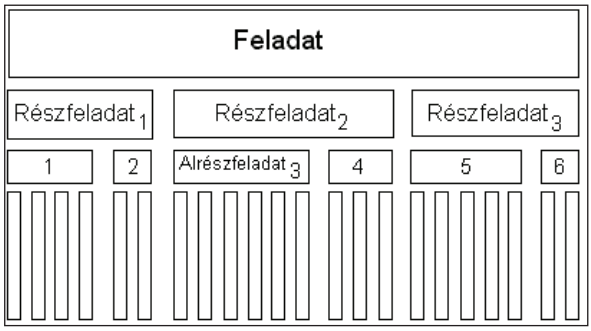
Kezdetben ez a paletta üres. Ha egy program valamilyen részletét sok más programban fel szeretnénk használni, akkor létrehozhatunk egy saját blokkot. Ez olyan, mint az eljárás vagy függvény imperatív nyelvek esetén. A létrehozott saját blokkok erre a palettára kerülnek, azután ezeket egyszerűen beszúrhatjuk a későbbi programjainkba, ugyanazon a projekten belül.

A saját blokkok a procedurális absztrahálást valósítják meg vizuális környezetben.

A programozási feladatok részfeladatokra bonthatók. A részfeladatoktól függően a felosztás lehet:

– minden részfeladat független a többitől és önmagában is egy feladatot képez (pl.: írjunk egy olyan programot, amely 10 adott fraktálfüggvény esetén megrajzolja a fraktál képét a képernyőn);
– a részfeladatok függetlenek, de a megoldásuk kombinációjából alakul ki a feladat megoldása (pl.: írjunk egy olyan rajzolóprogramot, amely rendelkezik a következő rajzoló funkciókkal: vonalrajzolás, téglalaprajzolás, ellipszisrajzolás, satírozott téglalap, satírozott ellipszis rajzolása, adott kerületű sokszög kitöltése stb.);
– létezik néhány alaprészfeladat, ezekre épül néhány komplexebb részfeladat és így tovább (pl.: objektumhierarchia tervezése).

Absztrahálás esetén különböző részfeladatokra egy közös megoldást próbálunk keresni.



125. ábra. *Procedurális absztrahálás*

A procedurális absztrahálás lehet *paraméteres absztrahálás*, amikor egy adott algoritmus alapján megírt részfeladat paraméterek függvényében különbözőképpen viselkedik, illetve lehet *specifikációfüggő absztrahálás*, amikor ismerjük a részfeladat előfeltételeit végrehajtás előtt és az utófeltételeket, amelyeket teljesítenie kell a kontextusnak a részfeladat végrehajtása után. A specifikációfüggő absztrahálás esetén az utófeltételek mindig kell hogy teljesüljenek.

A procedurális absztrahálás három alaptulajdonsága a következő:

– *minimalitás* – az eljárás viselkedését csak egy szükséges keretben kell értelmezni (ritkán fognak egy eljárást mások is használni);
– *általánosság* – a paraméter használata által valósul meg, így nem csak adott nevű változókra lehet alkalmazni, hanem adott típusú változócsoportra;
– *egyszerűség* – vagy *jól-meghatározottság* szükséges a megvalósításhoz, különben nem lehet egy olyan eljárást írni, amelynek eredménye egyértelmű legyen.

A procedurális absztrahálás alprogramok segítségével valósul meg. Alprogramokkal nevet adhatunk egy-egy kódrészletnek, hivatkozhatunk rájuk és paraméterezhetjük a viselkedésüket.

Az alprogramok paraméteres része foglalkozik azzal, hogy a paraméter használata által mennyire lesz általános az illető eljárás, hány helyen lehet alkalmazni, hogyan lehet megtervezni, felhasználni.

Az alprogramok specifikációfüggő része foglalkozik az alprogram viselkedésével, azzal, hogy mit kell csinálnia, nem azzal, hogy hogyan kell csinálnia.

Az alprogramtervezésnél jó, ha több pozitív tulajdonság teljesül:

– a belső algoritmus tervezéséhez ne kelljen ismernünk más alprogramok algoritmusainak működését;
– csak a specifikáció felhasználásával írjuk meg az alprogramot, ne implementáljunk dokumentálatlan saját ötleteket;

– az algoritmus minőségének javításával ne rontsuk el az alprogram specifikációit (paraméterszám, paramétertípus, mellékhatások létrehozása vagy kiküszöbölése).

Ha ezen feltételek teljesülnek, akkor a program több ízben is újraírható a hatékonyság növelésének érdekében vagy a felhasználó érdekeinek megfelelően.

Az alprogramok használatának előnyei:

– *újrafelhasználhatóság* – ugyanazt a kódot (kódrészletet) többször lehet felhasználni;
– *könnyen módosítható forráskód* – az alprogramok törzsét vagy a főprogramot egymástól függetlenül lehet módosítani;
– *karbantarthatóság, továbbfejlesztési lehetőség* – könnyen továbbfejleszthető az alkalmazás az eljárások függetlensége miatt;
– *könnyen olvasható forráskód* – csökken a kód bonyolultsága, áttekinthetőbb lesz, ha jól csengő alprogramneveket választunk (használunk), könnyen megérthetjük, hogy mit csinál az illető alprogram, az egész program.

A procedurális programozás az alprogram működésének leírásán alapszik, illetve ezen leírások betartásain a programozás során. Egy alprogram működésének leírása tartalmazza az alprogram *nevét*, *paraméterlistáját*, *környezetét*, *viselkedését* (törzs).

A programozásban paraméternek hívunk egy olyan értéket, amelytől egy programrész pontos működése függ.

*Formális paramétern*ek hívjuk egy alprogram deklarációjában vagy definíciójában leírt adatokat.

*Aktuális paramétern*ek nevezzük az alprogram hívásakor leírt konkrét értékeket.

Az adatok szerepe szempontjából egy paraméter lehet:

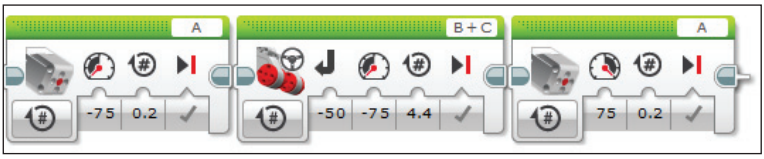
– bemeneti (in, input): a paraméter értéke határozza meg az alprogram futását;
– kimeneti (out, output): az alprogram állítja be a paraméter értékét;
– be- és kimeneti (in-out, input-output): az alprogram függ a paraméter kezdeti értékétől, és módosít(hat)ja is azt.

Az alprogram viselkedése:

– tartalmazza azon feltételeket, leszűkítéseket, amelyek teljesülésével az alprogram működik;
– tartalmazza azt, hogy milyen helyi és környezeti változók módosulnak;
– tartalmazza azt, hogy az alprogram végrehajtása által mi valósul meg.

Első példánkban képzeljük el, hogy egy jobbra és balra forduló autót szeretnénk megvalósítani.

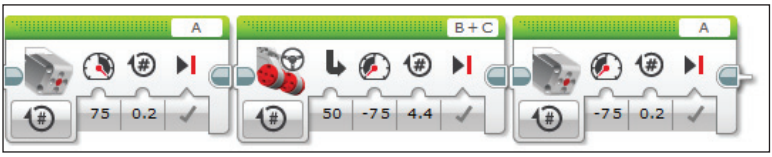
A jobbra térülés programblokkjait a 126. ábrán láthatjuk, a balra térülését pedig a 127. ábrán.



126. ábra. *Jobbra térülés*

Mivel ezt a két blokksorozatot sokszor fogjuk használni a program során, érdemes egy-egy saját blokkot definiálni.

Először a 126. és 127. ábráknak megfelelően tervezzük meg a programot úgy, hogy ne kössük össze a blokkokat a start blokkal.



127. ábra. *Balra térülés*

Válasszuk ki a jobbra térülés blokkjait, majd a Tools menüből válasszuk ki a My Block Builder menüpontot. Ekkor megjelenik a saját blokk varázsló, amely segítségével létrehozhatjuk a blokkunkat:

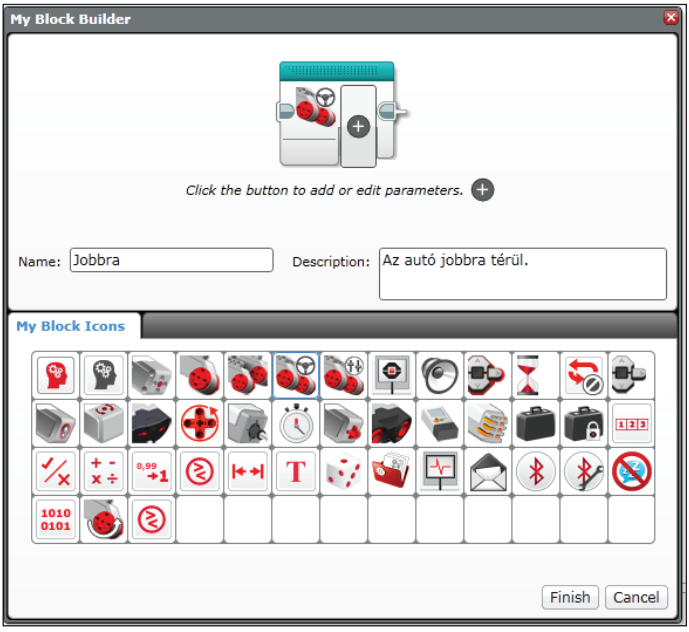
– Kötelezően nevet kell hogy adjunk a blokknak (Name);
– Megadhatjuk a blokk rövid leírását is (Description);
– A felkínált listából kiválaszthatunk egy ikont a blokkunk számára (My Block Icons).

Így a 128. ábrán látható kitöltött varázslóhoz jutunk, nem is marad más hátra, mint a Finish (Vége) gomb megnyomása.

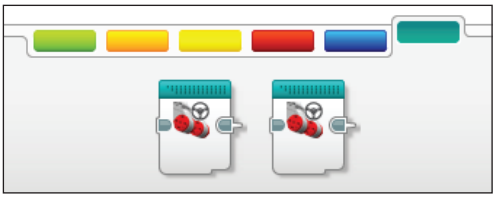
A Finish (vége) gomb megnyomása után az alprogramunk átalakul saját blokká, és megjelenik a saját blokkok (My Blocks) palettán. A 129. ábrán látható palettán csak az adott projekthez tartozó saját blokkok (alprogramok) láthatók.

Ezután a saját blokkokat ugyanúgy használhatjuk, mint a többi palettán lévő blokkokat. Ha meg akarjuk nézni a saját blokkok tartalmát, vagy szerkeszteni

akarjuk ezeket, megtehetjük úgy, hogy a programozási felületre kihúzott blokkra kettőt kattintunk. Ekkor egy új fülben megjelenik a saját blokk tartalma. Azt is észrevehetjük, hogy a felület automatikusan beszúr egy start blokkot a saját blokkok elé.



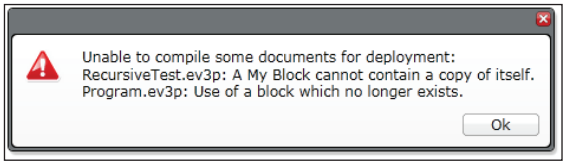
128. ábra. *A Saját blokk varázsló*



129. ábra. *Saját blokkok a palettán*

A LEGO MINDSTORMS EV3 Home Edition felület nem engedi meg a saját blokkok, s így az alprogramok rekurzív hívását. *Rekurzió*nak nevezzük azt az esetet, amikor egy alprogramban szereplő kód **önmagát** (tehát ugyanazt az alprogramot) hívja meg.

Ha úgy tervezzük meg a programot, hogy egy saját blokk tartalmazza önmagának egy példányát (blokkját), akkor a fordítás során a 130. ábrán látható hibaüzenet fog megjelenni.



130. ábra. *A rekurzió nem megengedett*

A saját blokkok paraméterezhetők is.

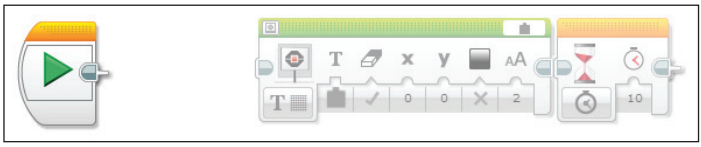
A 128. ábrán láthatjuk, hogy a blokk ikonja után megjelenik egy „+” jel, azzal a magyarázattal, hogy *„Click the button to add or edit parameters.”*, vagyis *„Kattints a gombra a paraméterek hozzáadásához vagy szerkesztéséhez.”*

Nézzünk meg egy egyszerű példaprogramot a saját blokkok paraméterezésére.

6. feladat

Egy saját blokkban írjuk ki a tégla képernyőjére a paraméterben megadott egész számot!

A feladatot úgy oldhatjuk meg, hogy a saját blokkunkat egy bemeneti (input) paraméterrel látjuk el. A 131. ábrán a kiinduló blokkot láthatjuk, a 132. ábrán a paraméter megadását a saját blokk varázslóban, a 133. ábrán pedig a végleges saját blokkot.

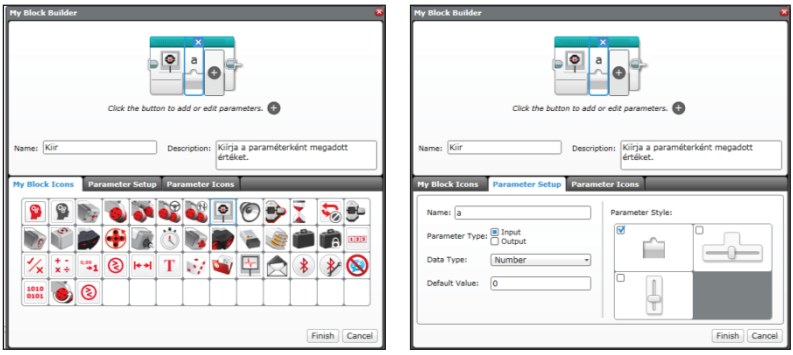


131. ábra. *Kezdeti blokk*

A felületre ráhúzunk egy kijelző blokkot és egy várakozás blokkot, elvégezzük a megfelelő beállításokat, majd kiválasztjuk az egérrel mind a két blokkot. Ezután a Tools menü My Block Builder menüpontja segítségével előhívjuk a saját blokk varázslót.

A blokk nevének, leírásának, ikonjának megadása után kattintsunk az ikon „+” jelére. Ekkor a saját blokk ikonjában megjelenik egy formális paraméter, a

varázsló alsó részén pedig két új fül: a Paraméter Setup (paraméter beállítások), valamint a Parameter Icons (paraméter ikonok).



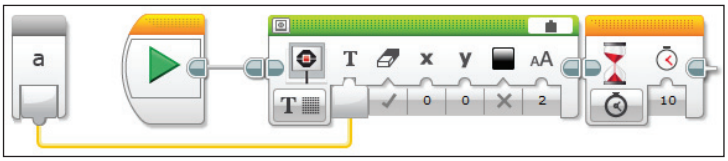
132. ábra. *A paraméter megadása*

A paraméter beállítások fülben beállíthatjuk a paraméter nevét, azt, hogy bemeneti vagy kimeneti paraméter legyen-e (in-out típusú paramétereket nem tud kezelni a vizuális környezet), beállíthatjuk az adat típusát (numerikus, logikai, szöveg, numerikus tömb, logikai tömb), az alapértelmezett értékét, valamint a paraméter vizuális megadási stílusát (csúszka, adatdrót stb.).

A paraméter ikonok fül segítségével egy listából ikont választhatunk a paraméternek.

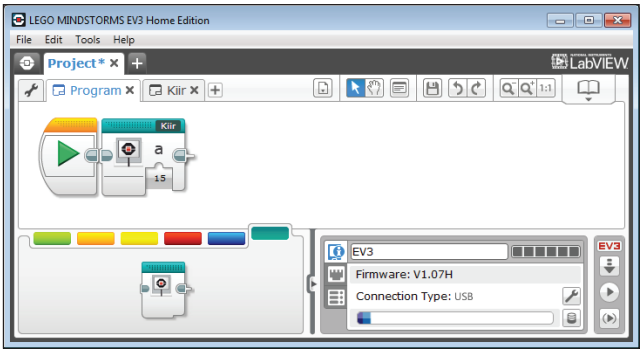
Amint megvagyunk a beállításokkal, a Finish (vége) gomb megnyomásával készíthetjük el a saját blokkunkat.

A paraméter a start blokk előtt jelenik meg, ezt a 133. ábrán látható módon adatdróttal össze kell hogy kössük a kijelző blokkal.



133. ábra. *A végleges saját blokk*

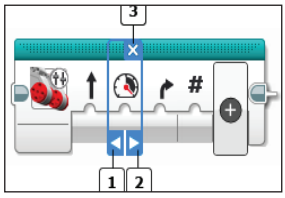
Az így elkészített blokkot bármikor használhatjuk a projektben, egyszerűen úgy, hogy a tervezőfelületre húzzuk a saját blokkok palettáról, majd megadjuk az aktuális paramétert.



134. ábra. *A saját blokk használata*

Megjegyzések

– Maximum 10 paraméter adható meg.
– A blokk paramétereinek sorrendjét módosíthatjuk a varázsló segítségével. Ha megadunk egy paramétert, egy kék téglalap jelenik meg körülötte, ennek segítségével kitörölhetjük a paramétert, vagy a bal, illetve a jobb nyilakkal a kívánt helyre (sorrendbe) mozgathatjuk a paramétert (135. ábra).
– Az 1.0.1. verziójú LEGO MINDSTORMS EV3 Home Edition felületen nincs lehetőség egy saját blokk paramétereinek utólagos módosítására, így már a tervezésnél gondoljuk meg jól, hány és milyen paramétereket szeretnénk.



135. ábra. *Paraméterek törlése, mozgatása*

A következő példaprogram megmutatja, hogyan használjunk kimeneti paramétereket is.

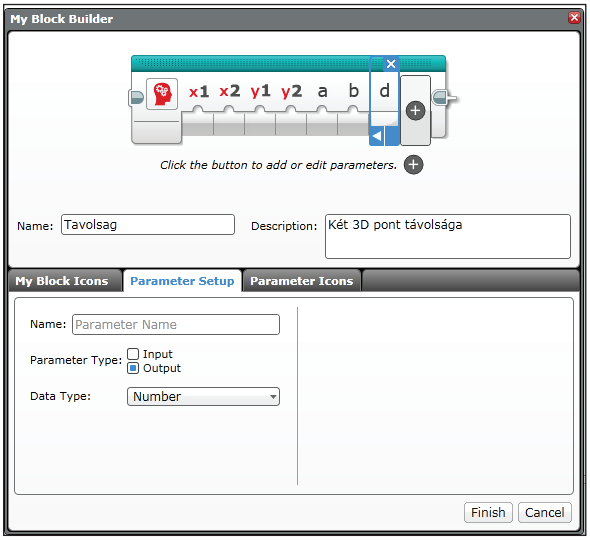
7. feladat

Adjuk meg két pont térbeli koordinátáit, majd egy saját blokkban számítsuk ki a két pont közötti távolságot!

A *távolság* két pont közé eső szakasz hossza. Az euklideszi háromdimenziós térben két pont, $P_1(x_1, y_1, z_1)$ és $P_2(x_2, y_2, z_2)$, távolságát a következő képlet adja meg:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

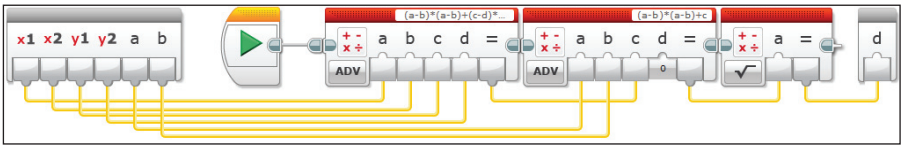
Egy olyan saját blokkra lenne tehát szükségünk, amely megkapja a $P_1(x_1, y_1, z_1)$ és $P_2(x_2, y_2, z_2)$ pontok koordinátáit, majd egy numerikus értékben visszatéríti a köztük lévő távolságot.



136. ábra. *Két pont távolsága*

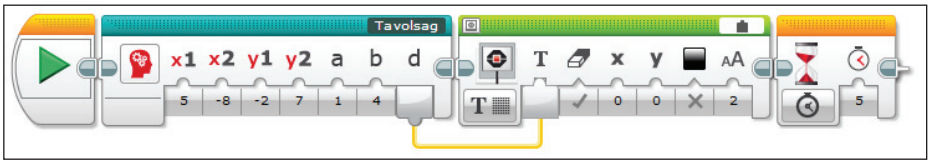
Amint a 136. ábrán is látszik, sajnos nincs lehetőségünk tetszőleges ikonok beállítására, a készletben csak az *x*-re és *y*-ra vonatkozó ikonokat találtunk, így a *z* koordinátákat az *a* és *b* nevű paraméterekben adjuk át. Hat bemeneti és egy kimeneti paraméterünk van.

Mivel a matematikai műveleteket megvalósító blokk is csak négy paramétert tud használni, ezért két blokkra lesz szükségünk. A feladatot megoldó saját blokk a 137. ábrán látható, használata pedig a 138. ábrán.



137. ábra. *Két pont távolságának kiszámítása*

A 137. ábrán megfigyelhetjük, hogy a bemeneti paraméterek a start blokk előtt vannak, a kimeneti paraméter pedig a blokksor után, utolsó elemként jelenik meg. A paramétereket értelemszerűen össze kell kötni a blokkokkal.



138. ábra. *Két pont távolságának kiírása*

8. feladat

Tervezzük meg egy olyan robot programját, amelyik meg tudja oldani a Hanoi tornyai feladatot!

Hanoi tornyainak legendája

Sok ezer évvel ezelőtt Indiában, Fo Hi uralkodása alatt, a benaresi Siva-templom közepén volt egy márványtábla, amelyből három gyémánttű állt ki. Az első tűn 64 vékony aranykorong helyezkedett el, alul a legnagyobb átmérőjű, majd rajta egyre kisebbek. Egyszer Siva isten megparancsolta a templom papjainak, hogy rakják át a korongokat az első tűről a másodikra. Két fontos szabályt azonban be kellett tartaniuk:

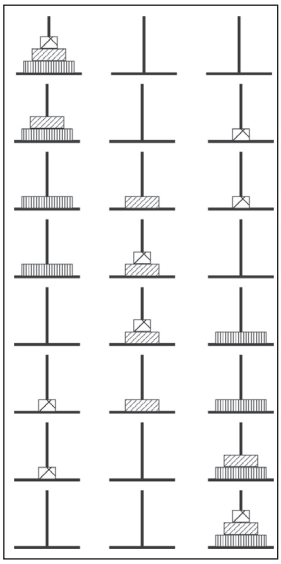
1. a korongok igen sérülékenyek, ezért egyszerre csak egyet lehet mozgatni,
2. valamint nem kerülhet a szent korongokból magasabb értékű alacsonyabb értékű fölé.

A szerzetesek természetesen használhatták a harmadik tűt is a rakodás közben. Amikor az utolsó korong a helyére kerül, a templom porrá omlik össze, és a világ véget ér.

A Hanoi tornyai játék leírását először egy bizonyos N. Claus de Siam, a Li-Sou-Stian egyetem oktatója publikálta egy párizsi újságban [4]. Később kiderült, hogy az 1883-ban megjelent cikk szerzője valójában Edouard Lucas francia matematikus, a Lyceé Saint-Louis tanára (Az álnév a Lucas d'Amiens név betűiből született). A játékot Lucas Hanoi tornyának keresztelte el.

A megoldás

Mi a játék megoldása? Tételezzük fel, hogy a papok a lehető leghatékonyabban, hiba nélkül dolgoznak. A kérdés tehát a következő: legalább hány lépésben lehet mind a 64 korongot átjuttatni az első tűről a másodikra?



139. ábra. *Hanoi tornyai*

Érdemes a problémát először kevesebb korongra megvizsgálni, hátha tapasztalunk valami összefüggést a korongok és a lépések száma között (egy lépés alatt természetesen egy korong áthelyezését értjük). Magától értetődően egy korong esetén egy, és könnyen végiggondolható, hogy kettő esetén három lépésre van szükségünk. Három korongra már nem ennyire egyszerű a kép, de némi próbálkozás árán megtalálhatjuk azt a hét áthelyezést, amely a leggyorsabb megoldást adja. A korongok számát K-val, a lépésekét L-lel jelölve megsejthetjük a következő összefüggést: $L = 2^K - 1$.

A bizonyítás egy ügyes trükkre épül. Vegyük észre, hogy K értékétől függetlenül biztos lesz egy olyan lépés, amikor a legnagyobb korong átkerül az elsőről a második tűre. Ekkor nyilván az összes többi a harmadik tűn van, mégpedig a szabályokból következően nagyság szerinti sorrendben. A feladat tehát a következőképpen módosul: helyezzük át a harmadik tűről a másodikra a korongokat úgy, hogy az elsőt is felhasználhatjuk. Egy újabb Hanoi-tornyot kaptunk, immár K–1 koronggal. Ezek szerint még annyi lépésre van szükségünk, mintha eggyel kevesebb koronggal kezdtük volna a játékot.

A fentiek segítségével felírhatjuk a következő képletet: $L = L' + 1 + L'$, ahol L' a lépések száma K–1 korong esetén. Az összeadás három tagja a megoldás három lépését jelöli:

1. K–1 korong az első tűről a harmadikra,
2. a legnagyobb korong a helyére,
3. a K–1 korong vissza a másodikra.

Legyen S_K–1 a K darab koronghoz tartozó lépések száma, azaz $L = S_K$–1, illetve $L' = S_{K-1}$–1. A fenti képletet átalakítva: $L = 2 \cdot L' + 1$, azaz $S_K - 1 = 2 \cdot (S_{K-1} - 1) + 1$, amiből a zárójel felbontása és az egyenlet rendezése után $S_K = 2 \cdot S_{K-1}$. A feladat elején már megállapítottuk, hogy S_1–1 = 1, azaz $S_1 = 2$. Mivel a fentiek szerint minden következő S kétszerese az előzőnek, ezért kimondhatjuk, hogy $S_K = 2^K$, ezzel pedig beláttuk a kiinduló állítást, hiszen $L = S_K$–1 = 2^K–1.

A szerzeteseknek tehát 2^{64}–1 (= 18 446 744 073 709 551 615) áthelyezést kell végrehajtaniuk. Ha feltesszük, hogy egy korongot átlagosan egy másodperc alatt tesznek át, munkájuk akkor is több mint 590 000 000 000 évig tartana (összehasonlításképpen, becslés szerint, a világegyetem 13,7 milliárd éves)!

Kétségtelen, hogy kellő ügyességgel megépíthető egy olyan LEGO robot, amely át tudja tenni a korongokat az egyik rúdról a másikra, azonban a robot vezérlésére szükség van a háttéralgoritmusra. Ezt fogjuk a következőkben megvizsgálni.

A Hanoi tornyai feladat a Divide et impera („oszd meg és uralkodj”) technika segítségével oldható meg klasszikus módon.

A probléma itt csupán az, hogy a LEGO MINDSTORMS EV3 Home Edition nem ismeri a rekurziót, tehát más megoldást kell keresnünk.

A kérdés tehát az, hogy létezik-e iteratív algoritmus az optimális lépéssorozat kigenerálásához?

Igen, létezik, és az alábbi észrevételeken alapszik (nevezzük el a három tornyot „kicsi”-nek, „közepes”-nek és „nagy”-nak attól függően, hogy miként viszonyulnak egymáshoz méretük szerint a legfelső korongok):

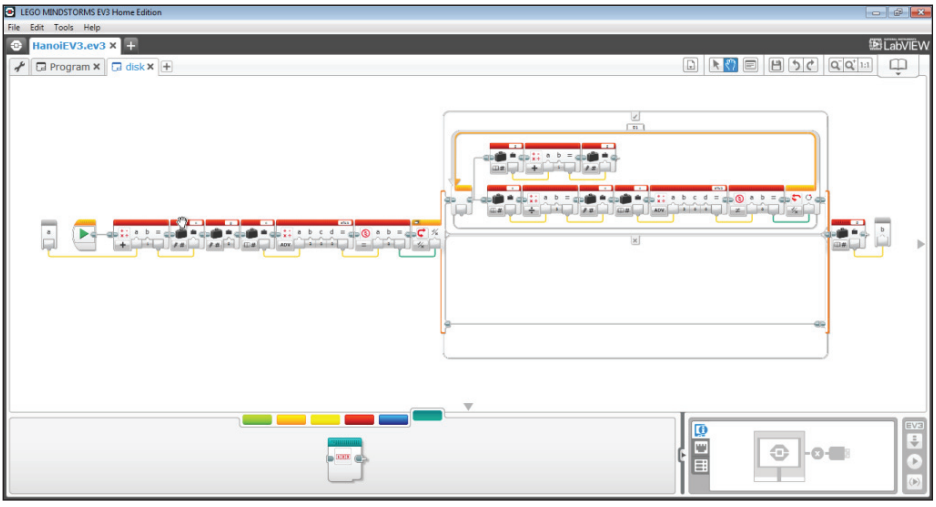
– A három szóba jöhető lépés: kicsi → nagy, kicsi → közepes, közepes → nagy.
– Ha utoljára a kicsi korong lépett, akkor nem léphet újra az, mert vagy visszakerülne oda, ahonnan jött (hurok), vagy ahova lépne, oda direktbe (egy lépésből) is léphetett volna (ami nyilván „optimálisabb” lett volna).
– Tehát a kicsi és közepes korongok felváltva lépnek.
– Ha a közepes korong van soron, akkor egyértelmű, hogy „közepes → nagy” lépést kell tenni.
– Bizonyítható továbbá, hogy attól függően, hogy az n páratlan vagy páros, a kicsi korong vagy az (a, b, c, a, b, c, …), vagy az (a, c, b, a, c, b, …) lépésmintát kell kövesse. Tehát ez esetben is egyértelműen meghatározható, melyik a következő lépés.

A fentieket figyelembe véve a játék megoldását a 140. ábrán látható C nyelven írt programban foglalhatjuk össze.

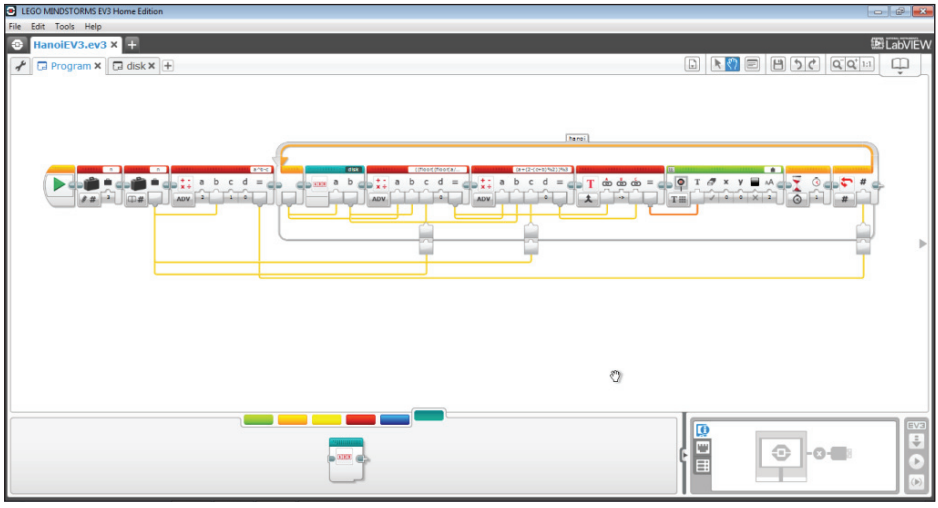
```
  int disk(int i)
  {
    int g=0, x = i+1;
    while(x%2==0)
    {
      ++g;
      x/=2;
    }
    return g;
  }

int main()
{
   int n=3;
   int limit = pow(2, n) - 1;
   for (int i = 0; i < limit; ++i)
   {
     int d = disk(i);
     int source = (((i/(int)pow(2,d)+1)/2)*(2-(n+d)%2))%3;
     int dest = (source + (2 - (n + d)%2))%3;
     printf("disk %d: %c->%c\n", d, source+'a', dest+'a');
   }
   return 0;
}
```

140. ábra. *Hanoi tornyai C nyelven*

Ha vizuális környezetbe szeretnénk átültetni, hogy robotunk is „értse”, először tervezzük meg a 141. ábrán látható disk nevű saját blokkot, majd rakjuk össze a 142. ábrán látható programot.



141. ábra. *A disk saját blokk*



142. ábra. *A Hanoi tornyai program*

3.2. Programozás a téglán

Az I.5. fejezetben már bemutattuk a tégla lehetőségeit, itt most a gyárilag telepített *Tégla program* pontot fogjuk részletezni.

Az EV3 tégla programozási alkalmazása hasonló a számítógépünkre telepített LEGO MINDSTORMS EV3 Home Edition környezethez. Az itt található blokkok megadják nekünk a kezdéshez szükséges alapismereteket. Természetesen bonyolult programokat csakis a számítógépen tudunk megtervezni, de a tégla programok lehetőséget nyújtanak arra, hogy a robotunk alapszinten működhessen számítógép nélkül.

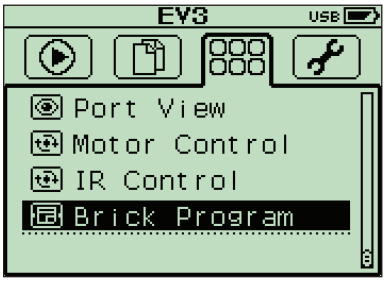
A téglán nem lehet egymásba ágyazott ciklusokat, bonyolult elágazásokat, több szálon futó programokat tervezni, nem lehet adatdrótokkal összekötni a blokkokat, nem lehet változókat, konstansokat használni vagy tömböket programozni, hanem csak nagyon kezdetleges érzékelő és motorműveleteket végrehajtani.

A portok kiosztása is alapértelmezett módon történik, ezen változtatni nem tudunk:

– 1-es port: érintésérzékelő,
– 2-es port: giroszkópos érzékelő,
– 3-as port: színérzékelő,
– 4-es port: infravörös érzékelő,
– A port: közepes motor,

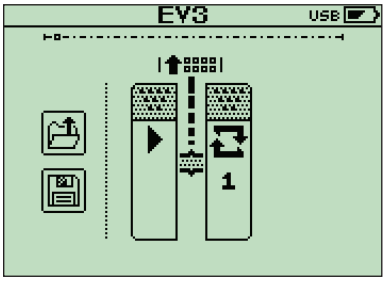
– B port és C port: két nagy motor,
– D port: egy nagy motor.

A 143. ábrán látható módon keressük meg a téglagombok segítségével, és nyissuk meg a Brick Program (Tégla program) alkalmazást.



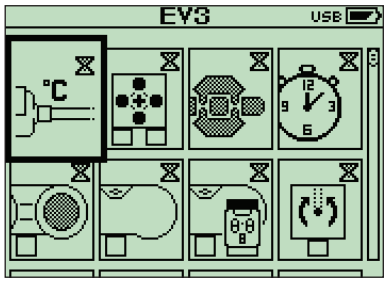
143. ábra. *A Tégla program alkalmazás*

Ha elindítottuk az alkalmazást, megjelenik a 144. ábrán látható környezet, amely nagyon egyszerű módon igyekszik reprodukálni a LEGO MINDSTORMS EV3 Home Edition blokkjait.



144. ábra. *A környezet*

Megfigyelhető a Start blokk és a Hurok blokk – amely segítségével a programot tudjuk megismételni 1, 2, …, 10 alkalommal vagy végtelenszer – egy sorrendi huzallal összekötve. Közöttük megjelenik a függőleges, megtört Blokk hozzáadás vonal. Ez jelzi, hogy további blokkokat adhatunk hozzá a programunkhoz. A tégla Fel gombjának megnyomásával tudjuk ezt megtenni, ekkor megjelenik a 145. ábrán látható blokkpaletta.

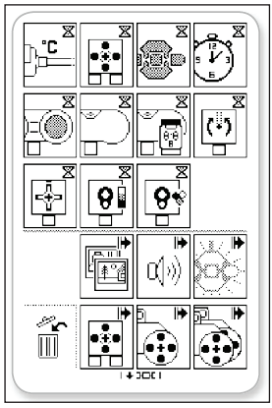


145. ábra. *A blokkpaletta*

A blokkpalettán a Le, Fel, Jobbra, Balra téglagombok segítségével navigálhatunk, itt található az aktuális blokkot kitörlő kuka gomb is.

Ha végignavigálunk az egész palettán, akkor visszajutunk a programhoz.

Általában véve kétféle blokk van: a Cselekvő (Action) és a Váró (Wait). A cselekvő blokkot egy kis nyíl, a váró blokkot egy homokóra jelzi a blokk jobb felső sarkában. Összesen hat különböző cselekvő és tizenegy különböző váró blokk közül választhatunk.



146. ábra. *A teljes blokkpaletta*

A teljes blokkpaletta a 146. ábrán látható, és a következő blokkokat foglalja magában (balról jobbra és fentről le):

– Váró blokkok
  - Hőmérséklet-érzékelő
  - Motorérzékelő
  - Téglagombok
  - Várakozás egy adott ideig
  - Ultrahangos érzékelő
  - Infravörös érzékelő

- Távirányító
- Motorforgás-érzékelő
- Érintésérzékelő
- Fényérzékelő
- Színérzékelő

– Cselekvő blokkok
- Kijelző
- Hang
- Fények a gombok körül
- Közepes motor
- Nagy motor
- Két nagy motor (tank vagy kormányozás)

– Kuka

Ha megtaláltuk azt a blokkot, amelyikre szükségünk van, navigáljunk rá és nyomjuk meg a tégla középső gombját, így visszakerülünk a programunkhoz, és a kiválasztott blokk beszúródik az aktuális helyre.

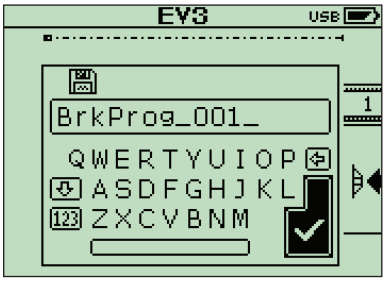
A programunkban a blokkok között a tégla bal és jobb gombjával navigálhatunk. A középső gomb megnyomásával módosíthatjuk a kijelölt blokk beállításait (mindig a képernyő közepén látható blokk), vagy új blokkot vehetünk fel, ha a sorrend vonal van kijelölve és a blokk hozzáadás vonal látható.

A kijelző és a hang blokkok esetében az alapértelmezett képek és hangok állíthatók be. Ezeknek a listáját a IV. fejezetben találhatjuk meg.

A programfuttatáshoz a tégla bal gombjával navigáljunk a program legelején lévő Start blokkra. Nyomjuk meg a középső gombot, és a programunk elindul.

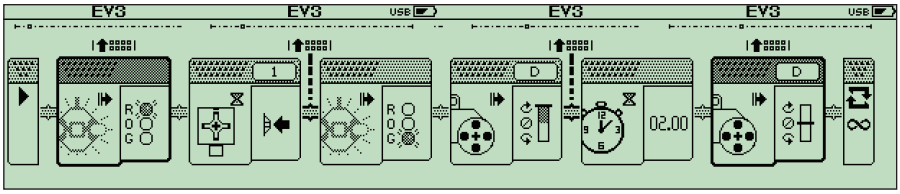
A Start blokk előtt lévő Megnyitás (Open) ikonra kattintva megnyithatjuk a lementett EV3 tégla programjainkat.

A Megnyitás alatti Mentés (Save) ikonra kattintva elmenthetjük a programot. A programnak nevet kell adni a 147. ábrán látható módon, majd az OK-ra kattintva tudjuk elmenteni a programot a *BrkProg_SAVE* mappába, amely az Állomány navigáció képernyőn érhető el.



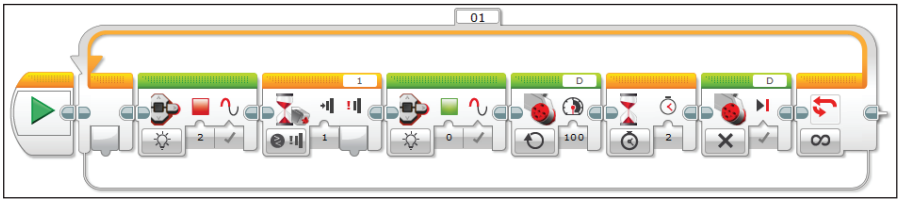
147. ábra. *Névadás és mentés*

A 148. ábrán látható példaprogram bekapcsolja a gombok piros fényét, vár, ameddig meg nem nyomjuk az érintésérzékelőt, ekkor bekapcsolja a zöld fényeket, beindítja a motort 2 másodperc erejéig, majd lezárja ezt. A beállított ciklusnak köszönhetően ezt a programot végtelenszer ismétli az EV3 tégla.



148. ábra. *Példaprogram – programozás a téglán*

Ha a 148. ábrán látható példaprogramot beimportáljuk a LEGO MINDSTORMS EV3 Home Edition *Tools* menüjének *Import Brick Program* parancsával, akkor a 149. ábrán látható programot kapjuk.



149. ábra. *Példaprogram – importálva*

3.3. Programozás más nyelvekben

A bemutatott saját vizuális környezeten kívül a LEGO EV3-as tégla számos más imperatív vagy vizuális nyelvben is programozható.

A Neumann-elvek felhasználásával megépített számítógépek programozása – az alacsony vagy magas szintű programozási nyelvek által – szorosan összefügg az imperatív (*imperative*: parancsoló, utasító) paradigmával.

Az imperatív paradigma tulajdonságai:

– *Algoritmikusság* – a programozó algoritmust kódol (forráskódot, programszöveget ír le), és ez az algoritmus működteti a processzort.
– *Utasítások használata* – a program utasítások sorozatából áll.
– *Változók használata* – legfőbb programozói eszköz a változó, amely a tár közvetlen elérését biztosítja, lehetőséget nyújt a tárban lévő érték közvetlen

megváltoztatására. Az algoritmusok, utasítások változókat használnak, a változók értékeit módosítják, tehát a program a hatását közvetlenül a tárban lévő értékekre fejti ki.
– *Ciklikusság* – lehetséges az utasítások ismételt végrehajtása.
– *Elágazó programszerkezet* – létezik GOTO utasítás, a program végrehajtása több ág valamelyikén futhat.
– *Tükrözés* – a beolvasás és kiírás a memória direkt másolásával történik meg.

Az imperatív programozási nyelvek főbb elemei: a *változók*, *konstansok*, *típusok*, *kifejezések*, *utasítások*, *vezérlési szerkezetek*, *programegységek*.

Az EV3 tégla számos imperatív nyelvben programozható, például:
– Bricx CC: C++ és C (ingyenes) [38]
– Microsoft MakeCode – blokkokkal és JavaScripttel (ingyenes)
– ROBOTC for MINDSTORMS EV3: C (nem ingyenes, firmware csere) [21]
– leJOS JAVA for EV3: Java (firmware csere) [37]
– EV3DEV (Debian, firmware csere): pár nyelv alatta [19], [24], [35]
– MONOBRICK: C# és .NET (firmware csere)
– PYTHON FOR EV3: Python (firmware csere)
– EV3 Basic (firmware csere)

Az imperatív paradigmára épülő nyelvek közül itt az ingyenes *Bricx CC* (*Bricx Command Center*) környezetet mutatjuk be, amely egy C-hez hasonló nyelvre épül. Nyilvánvaló, hogy a fent említett elemek mindegyike a C nyelvből öröklődött, a *Bricx CC* újdonsága csupán az, hogy új függvényeket tartalmaz az EV3 tégla programozásához. Imperatív és vizuális paradigma keveréke a ROBOTC és a MakeCode, amelyek mind vizuális, mind imperatív kóddal tudnak dolgozni. Más vizuális nyelvek közül itt a Scratch 3.0-ra térünk ki.

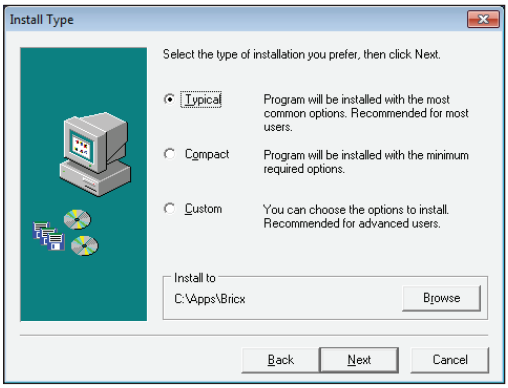
3.3.1. A Bricx CC telepítése

Töltsük le a *Bricx Command Center* (BricxCC) legutolsó verzióját a *http://bricxcc.sourceforge.net/* oldalról [8].

Kezdjük el a BricxCC környezet (IDE) telepítését a letöltött például *bricxcc_setup_33810_20130220.exe* futtatásával, majd kövessük a telepítés egyes lépéseit (Next gomb).

A telepítéshez válasszunk egy rövid, lehetőleg szóközöket és egyéb különleges karaktereket nem tartalmazó nevű mappát, például *C:\Apps\Bricx*, így később nem kell különösebb úgynevezett Escape-szekvenciákkal (vezérlőkarakterekkel) törődnünk a programozás során.

Válasszuk ki a *Typical* (Tipikus) telepítési módot a 150. ábrának megfelelően.



150. ábra. *Telepítés*

Írjuk be a *http://bricxcc.sourceforge.net/test_releases/* címet a böngészőbe, majd töltsük le innen a legutolsó dátummal rendelkező *test_release.zip* csomagot. Legyen ez például a *test_release20131007.zip*.

Csomagoljuk ki a ZIP-et a BricxCC telepítési mappába, a fenti esetben ez a *C:\Apps\Bricx*.

Ugyanebben a mappában keressük meg a *linux_tools.zip* csomagot, hozzunk létre a telepítési mappában egy *linux_tools* nevű mappát, majd csomagoljuk ki ide a ZIP-pet.

Ugyaninnen töltsük le az *lms_api.zip* csomagot is. A telepítési mappában hozzunk létre egy *API* nevű mappát, majd csomagoljuk ki ide az *lms_api.zip*-et.

Így a BicxCC telepítésével megvagyunk.

Telepítsük most a *Sourcery G++ LiteToolchains for the EV3* segédprogramot is, amely letölthető a *http://www.codesourcery.com/sgpp /lite/arm/portal/package4573/public/arm-none-linux-gnueabi/arm-2009q1-203-arm-none-linux-gnueabi.bin* oldalról.

A telepítéshez indítsuk el a letöltött *arm-2009q1-203-arm-none-linux-gnueabi.exe* programot.

Itt is válasszunk egy rövid, lehetőleg szóközöket és egyéb különleges karaktereket nem tartalmazó nevű mappát, például *C:\Apps\GPP*.

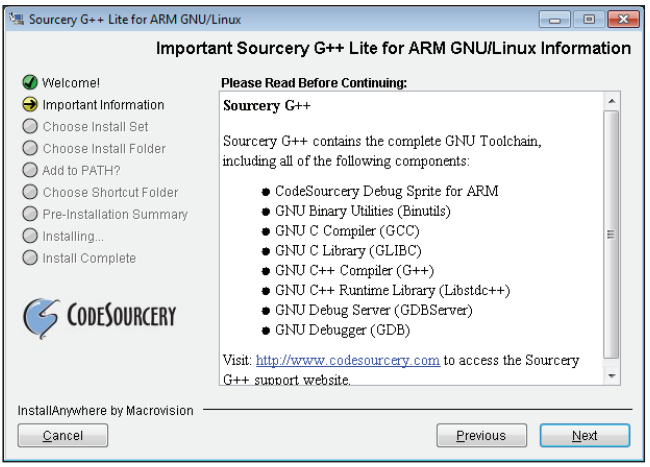
Amint a 151. ábrán láthatjuk, a programcsomag egy teljes G++ fordítóprogramot, függvénykönyvtárakat stb. tartalmaz.

Válasszuk itt is a *Typical* (Tipikus) telepítési módot, állítsuk be a telepítési útvonalat, ez most: *C:\Apps\GPP*, majd telepítsük a csomagot.

Rendszerünk beállításának következő lépése a telepített szoftverek elérési útvonalainak a megadása.

Ehhez menjünk a Windows kezelőpanel *\Control Panel\System and Security\System* lapjára, majd itt a bal oldani menüsorból kattintsunk az *Advanced*

System Settings (Haladó rendszerbeállítások) sorra. A megjelenő párbeszédablak *Advanced* (Haladó) fülecskéjében nyomjuk meg az *Environment Variables* (Környezeti változók) gombot. Ekkor a 152. ábrán látható párbeszédablak fog megjelenni.



151. ábra. *A G++ telepítése*

Itt a *PATH* (elérési útvonal) soron állva nyomjuk meg az *Edit…* (Szerkeszt) gombot.

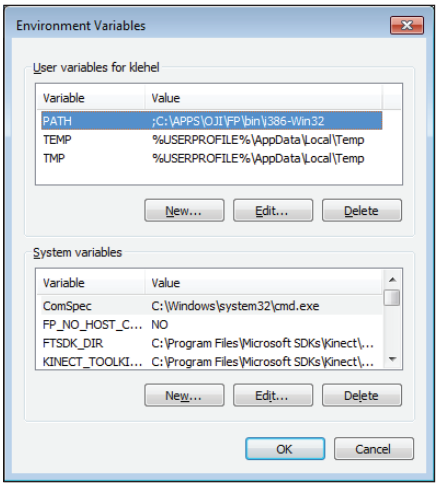
A megjelenő párbeszédablakban a *Variable value* (Változó értéke) sorhoz adjuk hozzá pontosvesszővel elválasztva a BricxCC, a linux_tools, valamint a G++ programkönyvtárait például a következőképpen: *;C:\Apps\Bricx;C:\Apps\Bricx\linux_tools;C:\Apps\GPP\bin*.

Végezetül a rendszerünk beállításának utolsó lépéseként töltsük le és telepítsük a LEGO EV3 tégla legutolsó verziójú firmware-t.

A firmware-t elérhetjük a *https://www.lego.com/en-gb/mindstorms/downloads* oldalon az EV3 MINDSTORMS FIRMWARE DOWNLOAD (PC/MAC) fejezetben. A könyv írásának pillanatában ez a V1.09H verziójú volt.

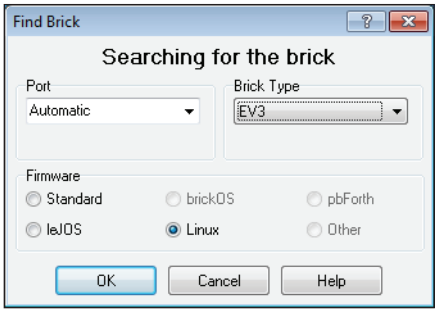
A firmware letöltése után a számítógépről ezt át kell telepítenünk a LEGO EV3 téglára. Ehhez elsősorban arra ügyeljünk, hogy a tégla elemei vagy akkumulátorai ne legyenek kifogyóban, legyen legalább 10 perc működésre elegendő energia bennük.

Kapcsoljuk be a LEGO EV3 téglát, és csatlakoztassuk a számítógéphez az USB kábel segítségével.



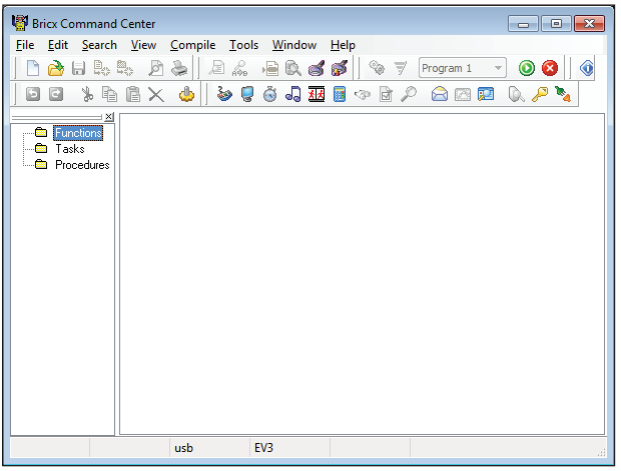
152. ábra. *A környezeti változók beállítása*

Indítsuk el a Bricx Command Center környezetet. Itt először a 153. ábrán látható beállításokkal (Port: Automatic, Brick Type: EV3, Firmware: Linux) keressük meg a téglánkat.



153. ábra. *A tégla megkeresése*

A tégla megkeresése után megjelenik a 154. ábrán látható környezet, amelyben a programozás mellett számos más lehetőség adódik a LEGO EV3 tégla beállításaira, kezelésére.

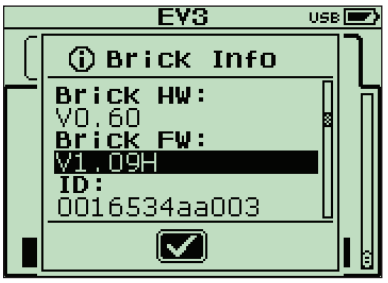


154. ábra. *A Bricx Command Center*

A firmware téglára való telepítése érdekében a *Tools* menü *Download Firmware* (Firmware letöltés) parancsát válasszuk ki. Ekkor egy párbeszédablak jelenik meg, amelyben beállíthatjuk, hogy melyik firmware-t szeretnénk telepíteni. Keressük meg, és adjuk meg a LEGO oldalról letöltött firmware-t (pl. EV3 Firmware V1.09H.bin).

A párbeszédablak bezárása után a rendszer elkezdi telepíteni az új firmware-t, a tégla kijelzőjén is megjelenik az *Updating..* (frissítés) felirat. A firmware letöltése az EV3 téglára körülbelül 5-7 percet vesz igénybe. A letöltés befejezése után a tégla újraindul.

Természetesen a firmware frissítését el tudjuk végezni a LEGO MINDSTORMS EV3 Home Edition *Tools* (Eszközök) menüjének *Firmware Update* (Firmware frissítő) parancsával is.



155. ábra. *A firmware verziószáma*

Ha meg akarunk győződni firmware frissítéséről, az EV3 tégla képernyőjének jobb szélén keressük meg a csavarkulcsot (a jobb téglagomb nyomogatásával), majd itt válasszuk ki a *Brick Info* (tégla információ) lehetőséget a lefele gomb nyomogatásával. Itt a *Brick FW:* sorban megjelenik a firmware verziószáma a 155. ábra szerint.

A rendszerünk kész, használhatjuk.

3.3.2. A Bricx CC környezet

A Bricx Command Center (BricxCC) a LEGO MINDSTORMS, a CyberMaster és a Spybot robotrendszerekkel való könnyebb munkavégzésre készült. A Dave Baum által kigondolt, jelenleg John Hansen által fejlesztett NQC (Not Quite C Compiler) köré épül, amely lehetővé teszi az EV3, RCX, a Scout, a Cybermaster és a Spybot téglák programozását egy C-hez hasonló nyelven [3], [42].

A BricxCC környezetet eredetileg Mark Overmars alkotta meg, ma pedig szintén John Hansen fejleszti.

Az RCX-re fejlesztett NQC nyelv hamarosan kibővült az NXT-re, így született meg az NXT (Not eXactly C).

A programozás elsajátítása előtt ismerkedjünk meg a Bricx CC környezettel!

Napjaink tendenciája, hogy a fordítóprogramokat *környezettel* lássuk el, mely integrálja a különböző elemeket. Legfontosabb kritérium, hogy a környezet egy szövegszerkesztővel rendelkezzen, amelyben meg tudjuk írni a forráskódot, közvetlenül lehessen hívni a fordítóprogramot vagy a szerkesztőt, a környezet tartalmazzon egy jól megírt kontextusfüggő súgórendszert is (*help*), amely a nyelvleírást és az egyes modulok, eljárások, függvények stb. bemutatását tartalmazza lehetőleg sok példaprogrammal.

Ezeket a környezeteket *IDE*-nek (*Integrated Development Environment*), *beágyazott fejlesztési környezetek*nek nevezzük.

Egy modern fordítóprogram környezete a következő elemeket tartalmazza:

– szövegszerkesztő,
– fordítórendszer,
– szerkesztőrendszer (linker),
– futtatórendszer,
– súgó,
– kódkiegészítők, sablonok,
– varázslók, kódgenerátorok,
– tervezőfelület (vizuális tervezés elősegítése: folyamatábrák, UML tervezési lehetőségek stb.),
– projekt kezelése, egyszerre több forráskód-állomány szerkesztése,

– debugger, nyomkövető (töréspontok definiálása, részletes futtatás, változók értékeinek nyomon követése, kifejezések kiértékelése stb.),
– szimbólumkövető,
– verem, regiszterek tartalmának kijelzése, gépi kód,
– adatbázis-tervező (relációk megadása),
– csoport- és nemzetközi programozás támogatása,
– automatikus dokumentációkészítő,
– tennivalók listája (ToDo),
– más környezeti eszközök, beágyazott lehetőségek (pl. ikon rajzolóprogramok stb.).

A 154. ábrán látható Bricx CC környezet a következő főmenüpontokkal rendelkezik:

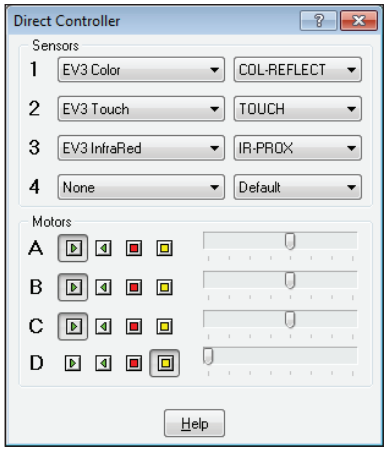
– *Fájl menü (File):* innen létrehozhatjuk, megnyithatjuk, elmenthetjük, bezárhatjuk a forráskódokat tartalmazó állományokat. A BricxCC lehetővé teszi a forráskódok nyomtatását. A menü alján található egy hasznos lista a nemrég megnyitott állományokról.
– *Edit menü (Edit):* Mint bármely szövegszerkesztőben, itt megtaláljuk a visszavonási (Undo) és visszaállítási (Redo) funkciókat, valamint a vágást (Cut), másolást (Copy), beillesztést (Paste) és törlést (Delete). Itt van a mindent kijelöl (Select All) funkció is. A Speciális beillesztés lehetővé teszi, hogy HTML vagy RTF formátumban szúrjunk be szöveget. A Következő mező (Next Field) funkció (F10) kiemeli a következő idézőjelek közé tett szöveget. A sablonokkal (F9) együtt használva, ez a funkció felgyorsítja a programírást. Például ha behozunk egy `for („init"; „condition"; „increment") { „body" }` sablont, akkor F10-et nyomva először az `„init"`-re ugrik a kurzor, majd a `„condition"`-ra, az `„increment"`-re és végül a `„body"`-ra. A menü utolsó pontja a Beállítások (Preferences…), ahol a környezetet, a fordítókat, sablonokat, makrókat szabhatjuk testre.
– *Keresés menü (Search):* innen megtalálhatunk és helyettesíthetünk egy szöveget a programban (mint bármely más szövegszerkesztőben). Egy pontos sorszámra léphetünk vagy megnyithatjuk az eljárások listáját. A kereséshez a *grep* linuxból jól ismert segédprogramot is felhasználhatjuk.
– *Nézet menü (View):* Ezzel a menüvel váltogathatjuk az összes panel és eszköztár láthatóságát. Hasznos ablak a Kód / Hiba lista (F12). Itt láthatjuk a fordított program kódját (ha sikeresen fordítják) vagy a hibás sorokat.
– *Fordítás menü (Compile):* Innen lehet lefordítani, futtatni, letölteni, elindítani, leállítani a programot.
– *Eszközök menü* (Tools)*:* A környezet egyik leghasznosabb menüje, a következő fejezetben részletesen foglalkozunk vele.
– *Ablak menü (Window):* Itt beállíthatjuk a gyerekablakok elhelyezkedését, a pozíciójukat akár le is menthetjük, majd betölthetjük.

– *Súgó menü (Help):* Innen megnyithatjuk az online útmutatót, a régi NQC útmutatót, a Névjegy doboz (About) és a hivatalos BricxCC weboldalt is, ahol frissítéseket, mintákat és dokumentumokat érhetünk el.

3.3.3. A Bricx CC eszközei és segédprogramjai

A John Hansen készítette eszközök nagyon hasznosak a téglával való kommunikálás, az információnyerés, adatfolyam szempontjából.

Közvetlen vezérlés (Direct Control): Innen kapcsolhatjuk be és ki a motorokat mindkét irányban, bármely sebességgel, beállíthatjuk az érzékelők típusait és módjait. Ezt a segédprogramot hibakeresési célokra fejlesztették.

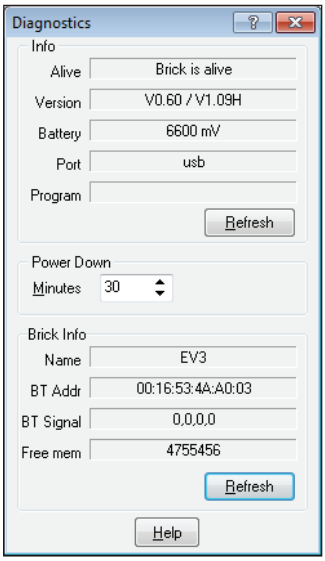


156. ábra. *A közvetlen vezérlés*

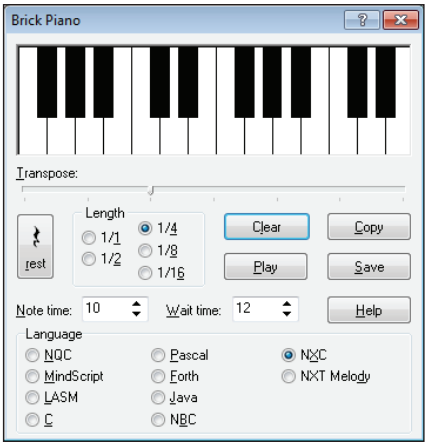
Diagnosztika (Diagnostics): Ez a segédprogram kiírja az összes rendelkezésre álló információt a csatlakoztatott tégláról: a firmware verzióját, az akkumulátor feszültségét, ahogyan a tégla csatlakozik a számítógéphez (USB vagy Bluetooth), annak nevét és Bluetooth-címét, a szabad memória mennyiségét stb.

A tégla követése (Watching the Brick): Ez a segédprogram egy teljesen átfogó párbeszédablak, amelyben információkat kaphatunk a tégla érzékelőiről, a szervomotorok paramétereiről, az üzenetekről stb. A megfigyelt esemény grafikonjait is nyomon követhetjük. Sajnos EV3 tégla esetében ez a funkció az ismertetett verzióban nem működik!

Zongora (Piano): Ezt az eszközt a zenészek használhatják, hogy dallamokat írjanak bármely programozható tégla számára. A legenerált kódot különböző programozási nyelvekbe exportáljuk.



157. ábra. *A diagnosztika*



158. ábra. *A zongora*

25. táblázat. *A zongora generált kódjai*

```
#include <config.h>
#include <dsound.h>
#include <tm.h>

static const note_t music[] = {
  { PITCH_E4, 40 },
  { PITCH_E4, 40 },
  { PITCH_F4, 40 },
  { PITCH_G4, 40 },
  { PITCH_G4, 40 },
  { PITCH_F4, 40 },
  { PITCH_E4, 40 },
  { PITCH_D4, 40 },
  { PITCH_C4, 40 },
  { PITCH_C4, 40 },
  { PITCH_D4, 40 },
  { PITCH_E4, 40 },
  { PITCH_D4, 40 },
  { PITCH_C4, 40 },
  { PITCH_C4, 40 },
  { PITCH_END, 0 }
};

int main(int argc,char* argv[]) {

  dsound_set_duration(10);
  dsound_set_internote(0);
  dsound_play(music);
  wait_event(dsound_finished, 0);
  dsound_set_duration(
  DSOUND_DEFAULT_16th_ms);
  dsound_set_internote(
  DSOUND_DEFAULT_internote_ms);
  return 0;
}
```

A C/C++ kód

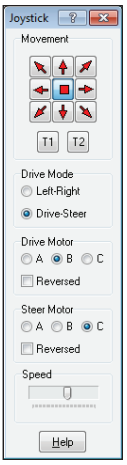
```
task main() {
  PlayTone(330,400);
  Wait(480);
  PlayTone(330,400);
  Wait(480);
  PlayTone(349,400);
  Wait(480);
  PlayTone(392,400);
  Wait(480);
  PlayTone(392,400);
  Wait(480);
  PlayTone(349,400);
  Wait(480);
  PlayTone(330,400);
  Wait(480);
  PlayTone(294,400);
  Wait(480);
  PlayTone(262,400);
  Wait(480);
  PlayTone(262,400);
  Wait(480);
  PlayTone(294,400);
  Wait(480);
  PlayTone(330,400);
  Wait(480);
  PlayTone(294,400);
  Wait(480);
  PlayTone(262,400);
  Wait(480);
  PlayTone(262,400);
  Wait(480);
}
```

Az NXC kód [6]

Joystick: Ezzel az eszközzel szabályozhatjuk a különböző hajtásrobotot (például Tribot, JohnNXT vagy Turtle). A kormányozás vagy tank üzemmódban is vezethetjük a robotot, az egyik motornál vezetve a kerekeket, a másikkal pedig kormányozhatunk.

Távirányító (Remote): Mind az RCX, mind a Scout a Mindstorms távirányítóval vezérelhető. Ebben az ablakban emulálhatjuk a távoli parancsot azzal, hogy a téglával egyenértékű parancsokat küldünk.

Konfigurálható követés (Configurable Watch): Ez az eszköz hasonló a *tégla követése* ablakhoz, kivéve, hogy kézzel kell hozzáadnunk a kiválasztott források monitorjait.



159. ábra. *A joystick*

Értékek beállítása (Set Values): Ezzel a párbeszédablakkal bármely írható forrás/érték kombinációt bármilyen olvasható forrás/érték kombináció értékére állíthatunk be. A források és az értéktartományok listája attól függ, hogy melyik téglát választottuk ki.

Spybot EEPROM: Ezt az eszközt az EPROM értékek követésére használhatjuk.

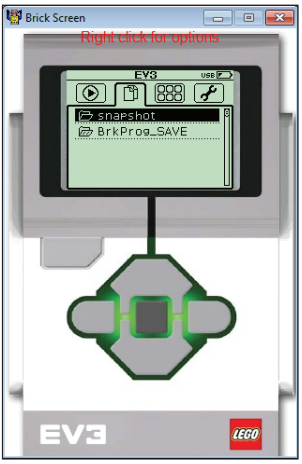
Intéző (Explorer): Ez az eszköz a tégla flash memória állományböngészője. Segítségével állományokat másolhatunk, törölhetünk, indíthatunk. A tégla teljes linuxos állományrendszere látszik.

Képernyőmentő (Screen Capture): Ez a segédprogram lehetővé teszi a tégla-képernyő tartalmának megjelenítését és lementését képként a számítógépen. Hasznos egy nem elérhető tégla képernyőjének megtekintéséhez vagy a tégla távoli vezérléséhez is. Segítségével a tégla nevét is beállíthatjuk. JPEG, PNG, BMP és GIF formátumban tudjuk lementeni a képernyőt, de AVI-ban mozgóképekben is tárolhatjuk ezt. Akár 4-szeres nagyításban is lementhetjük a képeket. Ez az eszköz akkor is nagyon hasznos, ha ki akarjuk vetíteni a tégla képernyőjét.

Követendők listája (Watch List): a debugolás során követendő elemeket és ezek értékeit tartalmazza.

Üzenetküldés (*Send Messages):* A téglák képesek reagálni az üzenetekre, egymásnak üzeneteket küldhetnek, vagy a számítógéppel is kommunikálhatnak

így. Az üzenetküldés azonban eléggé lassú művelet, körülbelül fél másodpercig tart, míg a robot reagál.



160. ábra. *A képernyőmentő*

Datalog: Az RCX az adatokat egy belső adatnaplóba is írhatja, amelyet fel lehet tölteni a számítógépre.

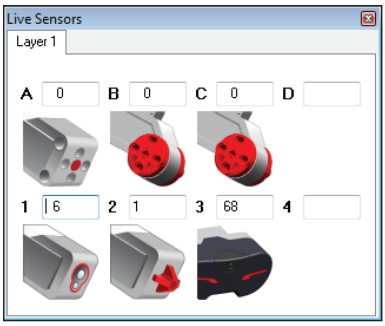
Memóriatérkép (Memory Map): Itt a tégla memóriájával kapcsolatos információkat kaphatjuk meg. Akkor hasznos, ha hibát keresünk, vagy a téglában memóriaproblémák merültek fel. Az eszköz különösen akkor hasznos, ha gyanítjuk, hogy nincs elegendő memória a programjainkhoz.

Memóriatisztítás (Clear Memory): Használjuk ezt a parancsot, ha ki akarjuk törölni a tégla memóriáját! Ez a parancs minden programban eltávolítja az összes feladatot és alprogramot, és kitörli az RCX adatnaplót is. A memória törlése fontos a nagy programok betöltése esetén, mivel a tégla megtartja az összes programot és feladatot akkor is, ha kikapcsoljuk.

MIDI-átalakítás (MIDI Conversion) és Hang-átalakítás (Sound Conversion): Ezek az eszközök egy MIDI- vagy WAV-hangállomány konvertálására szolgálnak egy kódba vagy egy RSO-*állományba. Mi*előtt átalakítanánk egy WAV-*állományt, konvertálnunk kell mono (csak egy csatorna), 8 bites 8 kHz-es formátumra. Az RSO-állományt tömö*ríthetjük, hogy helyet takarítson meg a téglán.

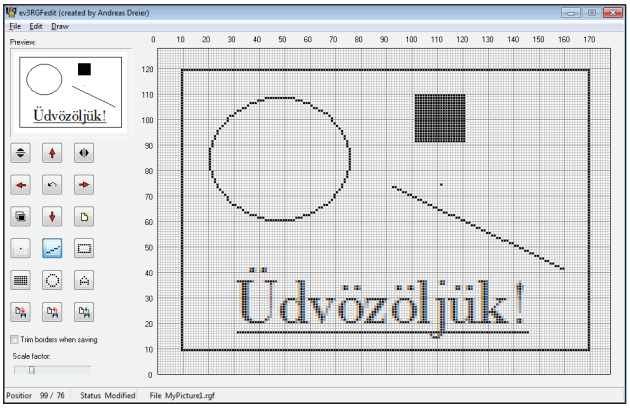
Egyszerű terminál (Simple Terminal): Ez az eszköz egy egyszerű kommunikációs terminálablak.

Élő érzékelők (*Live Sensors):* Az összes ki- és bemeneti – A, B, C, D és 1, 2, 3, 4 – portot tudjuk követni a segédprogram segítségével. Megjelennek a csatlakoztatott eszközök, le tudjuk olvasni ezek értékeit.



161. ábra. Élő érzékelők

A *Képszerkesztő* (Image Editor) segítségével képeket, grafikákat tervezhetünk az EV3 tégla kijelzője számára. A képet .RGF (Robot Graphics File) formátumba menthetjük le. Lehetőség van a kép eltolására, tükrözésére, különböző betűtípusú szövegek írására stb.



162. ábra. *Képszerkesztő*

Téglakereső (Find Brick), Téglalezáró (Turn Brick Off) és Kapcsolatlezáró (Close Communication): Ezen segédprogramok funkcionalitása magától érthető. A téglakereső működését már bemutattuk, a téglalezáró lekapcsolja, lezárja a beindított téglát, a kapcsolatlezáró pedig bontja az aktuális élő kapcsolatot.

Firmware letöltő (Download Firmware): amint már láttuk, ez az eszköz alkalmas arra, hogy frissítsük a tégla firmware-t.

Firmware kinyitása (Unlock Firmware): A firmware biztonsági beállításait adhatjuk itt meg.

Eszközök beállítása (Configure tools): Itt állíthatjuk be a BricxCC-t makrók és külső programok futtatásához.

3.3.4. Az EV3-as tégla programozása Bricx CC környezetben

3.3.4.1. A „Helló, világ!” program

A „Helló, világ!” programok olyan számítógépes programok, amelyek egyszerűen kiírják a megjelenítő eszközre: „Helló, világ!” (angolul: „Hello world!”). Mivel ez a program többnyire a legegyszerűbbek közé tartozik, gyakran használjuk arra, hogy kezdő programozókat megismertessünk a nyelv alapvető szintaxisával, illetve arra, hogy teszteljük a fejlesztői környezet helyes telepítését.

Itt is ezzel fogunk kezdeni a környezettel való ismerkedés után.

A File (Állományok) menüből válasszuk a New (Új) menüpontot.

Ekkor egy *Untitled1 (Névtelen1)* fülecske jelenik meg a szövegszerkesztőben, ide már beírhatjuk a programot.

Mentsük le az üres programot, hogy az állománytípusnak megfelelően beinduljon a környezet szintaxiskiemelője (Syntax highlighting).

A File / Save menüpont segítségével adjunk egy nevet a forráskódot tartalmazó állományunknak, legyen ez pl. *hv.c*, válasszuk ki a C++ Files (*.c, *.cpp, *.hpp) opciót, és mentsük el a forrásszövegünket.

Írjuk be a következő forráskódot:

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „C:\Apps\Bricx\API\ev3_lcd.h”
4. #include „C:\Apps\Bricx\API\ev3_command.h”
5.
6. int main()
7. {
8.   LcdInit();
9.   LcdText(1, 0, 0, „Hello, vilag!”);
10.   Wait(SEC_1);
11.   LcdExit();
12.   return 0;
13. }
```

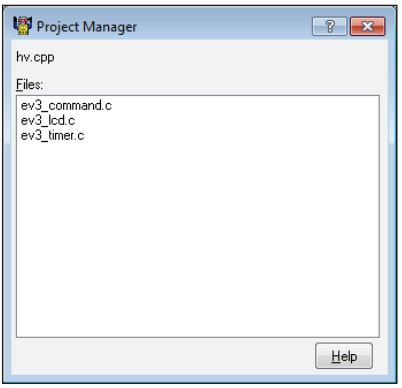
A View / Project Manager menüpont segítségével hívjuk elő a projektmenedzser ablakot. Kattintsunk a jobb egérgombbal, majd válasszuk az Add… (Hozzáadás) lehetőséget. A megjelenő párbeszédablakban navigáljunk a Bricx CC telepítési mappájában az API mappára, innen válasszuk ki az *ev3_command.c*, *ev3_lcd.c*, *ev3_timer.c* állományokat, majd zárjuk be a projektmenedzsert. A le-

mentett forrásszöveget tartalmazó mappában így létrejön egy *hv.prj* nevű szöveges állomány [30].

Sajnos az így létrejött állományba be kell írnunk az elérési útvonalakat is, ezért a File / Open menüpont segítségével nyissuk meg a *hv.prj* nevű állományt, így ez az IDE egy új fülecskéjében fog megjelenni.

Minden sor elé írjuk be az elérési útvonalat is – ahova az API mappát létrehoztuk (*C:\Apps\Bricx\API*), majd mentsük le az állományt:

```
1. C:\Apps\Bricx\API\ev3_timer.c
2. C:\Apps\Bricx\API\ev3_command.c
3. C:\Apps\Bricx\API\ev3_lcd.c
```

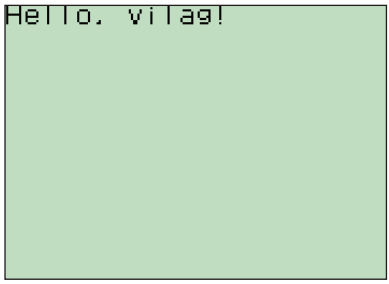


163. ábra. *A projektmenedzser*

Így megvan a fordításhoz szükséges két állomány, a forráskód és a projekt. Vigyázzunk, hogy a fordítás előtt mindig mentsük le a forráskódot, különben a régit fordítja le!

A Compile / Compile (F5) menüpont segítségével fordítsuk le a programunkat. Ez megtörténik hiba nélkül. Ha hibaüzenet jelenne meg, akkor a View / Show Code/Error | Warning Listing (F12) segítségével megnézhetjük a pontos hibaüzenetet és azt a sort, amelyik a hibát okozta.

A fordítás után a Compile / Download and Run (Ctrl+F5) menüpont segítségével tölthetjük le a lefordított programot a téglára és futtathatjuk ott. A Compile / Download (F6) segítségével csak letölthetjük, a Compile / Run (F7) segítségével pedig futtathatjuk a programot, amelynek az eredménye a 164. ábrán látszik.



164. ábra. *Helló, világ!*

3.3.4.2. Konstansok

A konstansokat az *ev3_constants.h* foglalja magában.
Néhány fontosabb konstans a következő:

Általános konstansok:

```
TRUE  1 // igaz érték
FALSE 0 // hamis érték

NUM_INPUTS     4   // bemeneti portok száma
NUM_LEDS       4   // LED-ek száma
LCD_WIDTH      178 // a kijelző vízszintes mérete
LCD_HEIGHT     128 // a kijelző függőleges mérete
TOPLINE_HEIGHT 10  // a felső sor magassága

OWNER_NONE 0x0000 // egy erőforrás tulajdonosa
```

A téglák láncolásának konstansai:

```
LAYER_MASTER 0x00 // mester
LAYER_SLAVE1 0x10 // szolga 1
LAYER_SLAVE2 0x20 // szolga 2
LAYER_SLAVE3 0x40 // szolga 3
LAYER_MASK   0x70 // a réteg maszk
```

A kimenet konstansai:

```
OUT_A    0x01 // A port
OUT_B    0x02 // B port
OUT_C    0x04 // C port
OUT_D    0x08 // D port
```

```
OUT_AB   0x03 // A és B portok
OUT_AC   0x05 // A és C portok
OUT_AD   0x09 // A és D portok
OUT_BC   0x06 // B és C portok
OUT_BD   0x0a // B és D portok
OUT_CD   0x0c // C és D portok
OUT_ABC  0x07 // A, B és C portok
OUT_BCD  0x0e // B, C és D portok
OUT_ABCD 0x0f // A, B, C és D portok
OUT_ALL  0x0f // minden port
OUT_MASK 0x0f // a kimeneti maszk

OUT_FLOAT 0x00  // motorműködés: Coast (amíg meg nem áll)
OUT_OFF   0x40  // motorműködés: Ki
OUT_ON    0x80  // motorműködés: Be
OUT_REV    0x00 // motorműködés: Hátra
OUT_TOGGLE 0x40 // motorműködés: Kapcsolás
OUT_FWD    0x80 // motorműködés: Előre

OUT_POWER_DEFAULT -127 // alapértelmezett erősség

OUT_REGMODE_IDLE  0 // nincs kiegyenlítés
OUT_REGMODE_SPEED 1 // sebesség kiegyenlítés
OUT_REGMODE_SYNC  2 // két motor szinkronizálása

RESET_NONE           0x00 // nincs visszaállítás
RESET_COUNT         0x08 // a belső tachométer visszaállítása
RESET_BLOCK_COUNT    0x20 // a blokk tachométer visszaállítása
RESET_ROTATION_COUNT 0x40 // a fordulatszámláló visszaállítása
RESET_BLOCKANDTACHO  0x28 // a belső és a blokk visszaállítása
RESET_ALL            0x68 // minden visszaállítása

NUM_OUTPUTS 4   // a kimeneti portok száma
```

Gombok konstansai:

```
BUTTON_ID_UP     0x01 // Fel gomb
BUTTON_ID_ENTER  0x02 // Enter
BUTTON_ID_DOWN   0x04 // Le gomb
BUTTON_ID_RIGHT  0x08 // Jobbra gomb
BUTTON_ID_LEFT   0x10 // Balra gomb
BUTTON_ID_ESCAPE 0x20 // Kilépés gomb
BUTTON_ID_ALL    0x3f // Minden gomb

NO_OF_BTNS 6  // EV3 gombok száma
NUM_BUTTONS 6 // A rendszerben lévő gombok száma
```

Színek konstansai:

```
INPUT_BLACKCOLOR  1 // Fekete
INPUT_BLUECOLOR   2 // Kék
INPUT_GREENCOLOR  3 // Zöld
INPUT_YELLOWCOLOR 4 // Sárga
INPUT_REDCOLOR    5 // Piros
INPUT_WHITECOLOR  6 // Fehér
```

Milliszekundumok, másodpercek, percek:

```
MS_1         1
MS_2         2
MS_3         3
MS_4         4
MS_5         5
MS_6         6
MS_7         7
MS_8         8
MS_9         9
MS_10       10
MS_20       20
MS_30       30
MS_40       40
MS_50       50
MS_60       60
MS_70       70
MS_80       80
MS_90       90
MS_100     100
MS_150     150
MS_200     200
MS_250     250
MS_300     300
MS_350     350
MS_400     400
MS_450     450
MS_500     500
MS_600     600
MS_700     700
MS_800     800
MS_900     900
SEC_1     1000
SEC_2     2000
SEC_3     3000
SEC_4     4000
SEC_5     5000
```

```
SEC_6    6000
SEC_7    7000
SEC_8    8000
SEC_9    9000
SEC_10  10000
SEC_15  15000
SEC_20  20000
SEC_30  30000
MIN_1   60000
```

Hangok:

```
TONE_C2      65 // Második oktáv C-hang
TONE_CS2     69 // Második oktáv C-félhang
TONE_D2      73
TONE_DS2     78
TONE_E2      82
TONE_F2      87
TONE_FS2     92
TONE_G2      98
TONE_GS2     104
TONE_A2      110
TONE_AS2     117
TONE_B2      123
TONE_C3      131 // Harmadik oktáv C-hang
TONE_CS3     139
TONE_D3      147
TONE_DS3     156
TONE_E3      165
TONE_F3      175
TONE_FS3     185
TONE_G3      196
TONE_GS3     208
TONE_A3      220
TONE_AS3     233
TONE_B3      247
TONE_C4      262 // Negyedik oktáv C-hang
TONE_CS4     277
TONE_D4      294
TONE_DS4     311
TONE_E4      330
TONE_F4      349
TONE_FS4     370
TONE_G4      392
TONE_GS4     415
TONE_A4      440
TONE_AS4     466
TONE_B4      494
```

```
TONE_C5      523
TONE_CS5     554
TONE_D5      587
TONE_DS5     622
TONE_E5      659
TONE_F5      698
TONE_FS5     740
TONE_G5      784
TONE_GS5     831
TONE_A5      880
TONE_AS5     932
TONE_B5      988
TONE_C6     1047
TONE_CS6    1109
TONE_D6     1175
TONE_DS6    1245
TONE_E6     1319
TONE_F6     1397
TONE_FS6    1480
TONE_G6     1568
TONE_GS6    1661
TONE_A6     1760
TONE_AS6    1865
TONE_B6     1976
TONE_C7     2093
TONE_CS7    2217
TONE_D7     2349
TONE_DS7    2489
TONE_E7     2637
TONE_F7     2794
TONE_FS7    2960
TONE_G7     3136
TONE_GS7    3322
TONE_A7     3520
TONE_AS7    3729
TONE_B7     3951
```

Ütemek:

```
NOTE_WHOLE   1000            // Egész hang
NOTE_HALF    (NOTE_WHOLE/2)  // Félhang
NOTE_QUARTER (NOTE_WHOLE/4)  // Negyed hang
NOTE_EIGHT   (NOTE_WHOLE/8)  // Nyolcad hang
NOTE_SIXTEEN (NOTE_WHOLE/16) // Tizenhatod hang
```

3.3.4.3. A kijelző programozása

Az EV3 tégla LCD (Liquid Crystal Display) folyadékkristályos képernyőjére a következő *ev3_lcd.h* és *ev3_lcd.c* modulokban (kell használni az `#include „C:\Apps\Bricx\API\ev3_lcd.h”`-t) lévő típusok és függvények segítségével írhatunk:

Az *ev3_lcd.h* típusai:

```
typedef byte IMGDATA;
typedef IMGDATA* IP;

typedef struct
{
  int X;
  int Y;
} LocationType;

typedef LocationType* PLocationType;

typedef enum
{
  ifRAW_FB0,
  ifRAW_BUF,
  ifXBM,
  ifP1,
  ifP4,
  ifBMP,
  ifPNG
} ImageFormat;
```

Az *ev3_lcd.h* függvényei:

```
bool LcdText(char color, short x, short y, char* text);
```

A `color` 0 vagy 1 lehet, 0 esetében fekete alapon fehér szöveget, 1 esetében fehér alapon fekete (normális) szöveget ír ki.

Az `x` és az `y` a szöveg kezdőhelye pixelekben megadva. A képernyő mérete 178×128 pixel, így a megadható `x` értékek a 0…177, az `y` értékek pedig a 0…127 tartományból lehetnek.

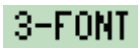
A `text` a kiírandó szöveg.

A függvény **`false`** (hamis) értéket térít vissza, ha a képernyő nem volt inicializálva.

A szöveg betűtípusát a következő függvénnyel lehet megváltoztatni:

```
bool LcdSelectFont(byte FontType);
```
A `FontType` a beállítani kívánt betűtípus a 26. táblázat alapján.

26. táblázat. *Betűtípusok*

Kód	Konstans	Eredmény
0	FONTTYPE_NORMAL	
1	FONTTYPE_SMALL	
2	FONTTYPE_LARGE	
3	FONTTYPE_TINY	

```
bool LcdInit();
```
A kiírás vagy rajzolás előtt a kijelzőt inicializálni kell.

```
bool LcdInitialized();
```
Visszatéríti, hogy a kijelző inicializálva volt-e vagy sem.

```
bool LcdExit();
```
A használat után lezárjuk a kijelzőt.

```
void LcdRefresh();
```
Frissíti a kijelzőt.

```
void LcdSetAutoRefresh(bool bOn);
```
Beállítja a kijelző automatikus frissítését, ha a `bOn` **`true`**.

```
bool LcdUpdate();
```
Frissíti a kijelzőt.

```
bool LcdClean();
```
Letörli a kijelzőt.

```
void LcdClearDisplay();
```
Letörli a kijelzőt még akkor is, ha nem volt inicializálva.

```
bool LcdScroll(short y);
```
Függőlegesen görgeti a kijelzőt.

```
byte* LcdGetDisplay();
```
Visszatéríti a kijelző azonosítóját.

```
byte* LcdGetFrameBuffer();
```
Visszatéríti a kijelző memóriazónájának azonosítóját.

```
void LcdWriteDisplayToFile(char* filename, ImageFormat fmt);
```
Állományba menti a kijelző tartalmát. Állományformátumok (`ImageFormat`): `ifRAW_FB0`, `ifRAW_BUF`, `ifXBM`, `ifP1`, `ifP4`, `ifBMP`, `ifPNG`.

```
void LcdWriteFrameBufferToFile(char* filename, ImageFormat fmt);
```
Állományba menti a kijelző memóriazónájának tartalmát. Állományformátumok (`ImageFormat`): `ifRAW_FB0`, `ifRAW_BUF`, `ifXBM`, `ifP1`, `ifP4`, `ifBMP`, `ifPNG`.

```
bool LcdIcon(char Color, short X, short Y, char IconType,
char IconNum);
```
Az előre definiált ikonokat rajzolja ki a megadott színnel (0 – fekete alapon fehér, 1 – fehér alapon fekete), a megadott `X` és `Y` koordinátákra. Az `IconType` 0 – `ICONTYPE_NORMAL`, 1 – `ICONTYPE_SMALL`, 2 – `ICONTYPE_LARGE`, 3 – `ICONTYPE_MENU`, valamint 4 – `ICONTYPE_ARROW` lehet. Az `IconNum` egy egész szám, a kívánt ikon sorszáma.

```
bool LcdBmpFile(char Color, short X, short Y, char* Name);
bool LcdPicture(char Color, short X, short Y, IP pBitmap);
```
Képet rajzolhatunk ki a megadott színnel, a megadott pozícióktól kezdődően. Az első függvényben a képet tartalmazó BMP-állomány nevét kell megadni szövegként, a második függvényben pedig a kép memóriabufferét kell megadni.

```
bool LcdFillWindow(char Color, short Y, short Y1);
```
Az `Y`-től az `Y1`-ig terjedő sávot festi ki a megadott színnel.

```
char PointOutEx(int x, int y, unsigned long options);
char LineOutEx(int x1, int y1, int x2, int y2,
unsigned long options);
char RectOutEx(int x, int y, int width, int height,
unsigned long options);
char CircleOutEx(int x, int y, byte radius,
unsigned long options);
char EllipseOutEx(int x, int y, byte radiusX, byte radiusY,
unsigned long options);
```

Függvények segítségével grafikus objektumokat tudunk kirajzolni a tégla kijelzőjére. Mindegyik függvénynek megvan az `Ex` nélküli változata is, ekkor nem kell megadni az **`unsigned long`** `options` paramétert, pontosabban ezt a `DRAW_OPT_NORMAL`-nak veszi. Például: **`char`** `PointOut(`**`int`** `x, `**`int`** `y);`

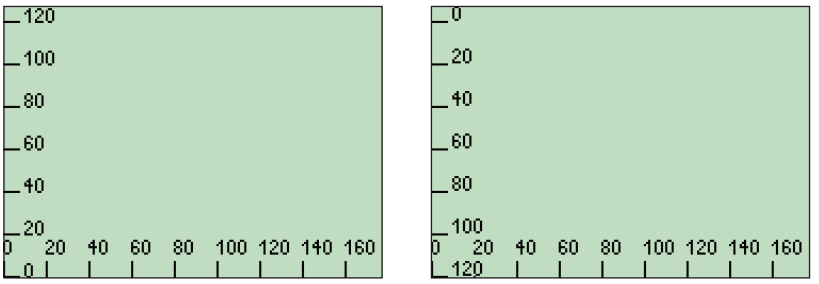
Tárgyaljuk le először az `options` paramétert, amely a kirajzolás módját adja meg. Ezt a tárgyalást a 27. táblázat foglalja össze.

27. táblázat. *Az `options` paraméter*

`DRAW_OPT_NORMAL`	`0x0000`	Normális rajzolás.
`DRAW_OPT_CLEAR_WHOLE_SCREEN`	`0x0001`	Rajzolás előtt törli a teljes képernyőt.
`DRAW_OPT_CLEAR_EXCEPT_STATUS_SCREEN`	`0x0002`	Az állapotsoron kívül törli a teljes képernyőt rajzolás előtt.
`DRAW_OPT_CLEAR_PIXELS`	`0x0004`	Pixeleket töröl rajzolás közben (fehérben rajzol).
`DRAW_OPT_CLEAR`	`0x0004`	Pixeleket töröl rajzolás közben (fehérben rajzol).
`DRAW_OPT_INVERT`	`0x0004`	Invertálja a szöveget vagy a grafikát.
`DRAW_OPT_LOGICAL_COPY`	`0x0000`	A pixelek rajzolása közben logikai MÁSOLÁS műveletet (COPY) hajt végre.
`DRAW_OPT_LOGICAL_AND`	`0x0008`	A pixelek rajzolása közben logikai ÉS műveletet (AND) hajt végre.
`DRAW_OPT_LOGICAL_OR`	`0x0010`	A pixelek rajzolása közben logikai VAGY műveletet (OR) hajt végre.
`DRAW_OPT_LOGICAL_XOR`	`0x0018`	A pixelek rajzolása közben logikai XOR műveletet hajt végre.
`DRAW_OPT_FILL_SHAPE`	`0x0020`	Kitölti az alakzatot (téglalap, kör, ellipszis, sokszög).
`DRAW_OPT_CLEAR_SCREEN_MODES`	`0x0003`	Bit maszk a képernyőtörlés módnak.
`DRAW_OPT_LOGICAL_OPERATIONS`	`0x0018`	Bit maszk a logikai műveletek számára.
`DRAW_OPT_POLYGON_POLYLINE`	`0x0400`	Nem zárja be a sokszöget, hanem törtvonalat rajzol.
`DRAW_OPT_CLEAR_LINE`	`0x0800`	Szöveg kirajzolása előtt törli a teljes sort.
`DRAW_OPT_CLEAR_EOL`	`0x1000`	Szöveg kirajzolása után törli a szöveg utáni teljes sort.

Az első függvény egy adott `x`, `y` koordinátájú pixelt rajzol ki, a második egy vonalat, a harmadik egy téglalapot, a negyedik egy `x`, `y` középpontú, `radius` sugarú kört rajzol ki, az ötödik pedig egy ellipszist, amelynek *x* sugara az `radiusX`, *y* sugara pedig a `radiusY`.

A 165. ábra a kijelző parancsainak grafikus és szöveges koordináta-rendszerét mutatja be. Amint láthatjuk, a két koordináta-rendszer *y* koordinátája egymás fordítottjai. Tehát ha az `LcdText` függvényt használjuk, akkor a 165. b) ábrának megfelelően kell beírnunk az *x* és *y* koordinátákat.



165. ábra. *a) grafikus és b) szöveges koordináták*

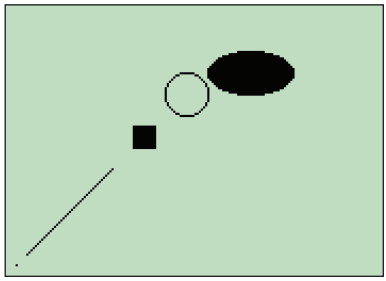
A 165. ábrát megvalósító program a következő:

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „c:\APPS\Bricx\API\ev3_lcd.h”
4. #include „c:\APPS\Bricx\API\ev3_command.h”
5.
6. int main()
7. {
8.   LcdInit();
9.   LcdSelectFont(3);
10.   char s[3];
11.   int x;
12.   for(x = 0; x <= 170; x+= 20)
13.   {
14.      sprintf(s, „%d”, x);
15.      LcdText(1, x, 110, s);
16.      LineOut(x, 0, x, 7);
17.   }
18.
19.   int y;
20.   for(y = 0; y <= 120; y += 20)
21.   {
22.      sprintf(s, „%d”, y);
```

```
23.       //LcdText(1, 10, 127-y-6, s); // grafikus
24.       LcdText(1, 10, y+1, s);       // szoveges
25.       LineOut(0, y, 7, y);
26.    }
27.    Wait(SEC_1);
28.    LcdExit();
29.    return 0;
30. }
```

Az alábbi program grafikus alakzatokat rajzol ki a tégla kijelzőjére úgy, ahogy a 166. ábrán látható.

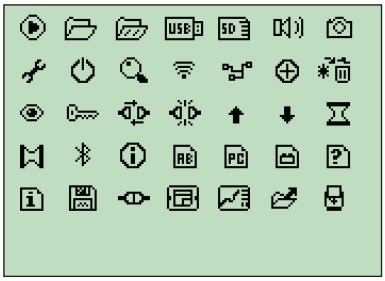
```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „c:\APPS\Bricx\API\ev3_lcd.h”
4. #include „c:\APPS\Bricx\API\ev3_command.h”
5.
6. int main()
7. {
8.    LcdInit();
9.    PointOut(5, 5);
10.    LineOut(10, 10, 50, 50);
11.    RectOutEx(60, 60, 10, 10, DRAW_OPT_FILL_SHAPE);
12.    CircleOut(85, 85, 10);
13.    EllipseOutEx(115, 95, 20, 10, DRAW_OPT_FILL_SHAPE);
14.    Wait(SEC_1);
15.    LcdExit();
16.    return 0;
17. }
```



166. ábra. *A grafikus kijelző ábrái*

A következő program kirajzolja a LEGO tégla összes ikonját, ahogy az a 167. ábrán látható:

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „c:\APPS\Bricx\API\ev3_lcd.h”
4. #include „c:\APPS\Bricx\API\ev3_command.h”
5.
6. int main()
7. {
8.   LcdInit();
9.   int i;
10.   for (i = 0; i < 35; i++)
11.     LcdIcon(1, 0+((i % 7) * 24), 5+((i / 7) * 20),
12.     ICONTYPE_NORMAL, i);
13.   Wait(SEC_1);
14.   LcdExit();
15.   return 0;
16. }
```



167. ábra. *Ikonok*

3.3.4.4. A hangfal programozása

Az EV3 tégla hangfalát a következő *ev3_sound.h* és *ev3_sound.c* modulokban (kell használni az `#include „C:\Apps\Bricx\API\ev3_sound.h”`-t) lévő típusok és függvények segítségével programozhatjuk:

Az *ev3_sound.h* típusa:

```
typedef struct
{
   unsigned short Frequency;
   unsigned short Duration;
} Tone;
```

Az *ev3_lcd.h* függvényei:

```
bool SoundInit();
```
Inicializálja a hangfalat.

```
bool SoundOpen();
```
Megnyitja a kommunikációt a hangfallal.

```
bool SoundClose();
```
Lezárja a kommunikációt a hangfallal.

```
bool SoundExit();
```
Kilép a hangfal üzemmódból.

```
bool SoundInitialized();
```
Visszatéríti, hogy a hangfal inicializálva volt-e vagy sem.

```
void PlayFileEx(char* pFileName, byte volume, bool loop);
```
Lejátszik egy .rmd, .wav, .rso hangállományt. A paraméterek közül `pFileName` az állomány neve, `volume` jelöli, hogy mennyire legyen hangos, a `loop` pedig azt, hogy ismételje-e vagy sem.

```
void PlayFile(char* pFileName);
```
Alapértelmezett módon (`volume = 100`, `loop = false`) játssza le a `pFileName` nevű hangállományt.

```
void PlayToneEx(unsigned short frequency, unsigned short
duration, byte volume);
```
Lejátszik egy adott hangot. A paraméterek közül a `frequency` a hang frekvenciáját, a `duration` a lejátszás hosszát, a `volum` a lejátszás erejét adja meg.

```
void PlayTone (unsigned short frequency, unsigned short
duration);
```
Alapértelmezett módon (`volume = 100`) játszik le egy adott hangot.

```
void PlaySound(byte aCode);
```
Lejátszik egy, az `aCode`-dal azonosított rendszerhangot.

```
void PlayTonesEx(Tone tones[], size_t size);
```
A `tones[]` tömbben megadott hangokat játssza le, `size` a tömb mérete, vagyis a lejátszandó hangok száma.

```
void PlayTones(Tone tones[]);
```

Lejátssza a `tones[]` tömbben megadott hangokat.

```
int SoundState();
```

Visszatéríti a hangfal állapotát.

```
void StopSound();
```

Leállítja a lejátszást, üresjáratra állítja a hangfalat.

```
bool SoundTest();
```

Teszteli, hogy a hangfal foglalt-e vagy sem, vagyis éppen lejátszik-e valamit vagy sem.

```
void SoundReady();
```

Készenléti állapotba helyezi a hangfalat. Ha a hangfal foglalt (épp lejátszik valamit), várakozik, míg fel nem szabadul.

```
void MuteSound();
```

Elnémítja a hangfalat.

```
void UnmuteSound();
```

Beindítja az elnémított hangfalat.

```
void ClearSound();
```

A `StopSound` szinnonimája.

A következő program bemutatja a hangfal működését, lejátszik egy hangot (`TONE_A4` – negyedik oktávú, 440 Hz frekvenciájú egyvonalas *a* kamarahangot, vagyis a *Lá*-t), alapértelmezett rendszerhangot, valamint egy tömbben megadott zenét (Örömóda).

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „c:\APPS\Bricx\API\ev3_lcd.h”
4. #include „c:\APPS\Bricx\API\ev3_command.h”
5. #include „c:\APPS\Bricx\API\ev3_sound.h”
6.
7. Tone music[] = {
8.   {TONE_E4, MS_500},
9.   {TONE_E4, MS_500},
10.   {TONE_F4, MS_500},
11.   {TONE_G4, MS_500},
12.   {TONE_G4, MS_500},
```

```
13.   {TONE_F4, MS_500},
14.   {TONE_E4, MS_500},
15.   {TONE_D4, MS_500},
16.   {TONE_C4, MS_500},
17.   {TONE_C4, MS_500},
18.   {TONE_D4, MS_500},
19.   {TONE_E4, MS_500},
20.   {TONE_D4, MS_500},
21.   {TONE_C4, MS_500},
22.   {TONE_C4, MS_500}
23. };
24.
25. int main()
26. {
27.   SoundInit();
28.   PlayTone(TONE_A4, MS_500);
29.   Wait(SEC_1);
30.   PlaySound(SOUND_FAST_UP);
31.   Wait(SEC_1);
32.   PlayTones(music);
33.   Wait(SEC_1);
34.   return 0;
35. }
```

3.3.4.5. Parancs

Az *ev3_command.h* egyetlen parancsot (függvényt) tartalmaz, mégpedig a következőt:

```
void Wait(unsigned long ms);
```

A megadott `ms` milliszekundumig várakozik.

3.3.4.6. A gombok programozása

Az EV3 tégla gombjait a következő *ev3_button.h* és *ev3_button.c* modulokban (kell használni az `#include „C:\Apps\Bricx\API\ev3_button.h”`-t) lévő függvények segítségével programozhatjuk:

```
bool ButtonLedInit();
```

A függvény inicializálja a tégla gombjait. Hamis értéket térít vissza, ha a folyamat nem volt sikeres.

```
bool ButtonLedOpen();
```

A függvény a gombok állapotával inicializál a nulla értékek helyett. Ez megakadályozza a hamis gombnyomási eseményeket a program indításakor.

```
bool ButtonLedClose();
```

Lezárja a munkafolyamatot, újrainicializálja a LED-eket.

```
bool ButtonLedExit();
```

A függvény lezárja a tégla gombjait, megszakítja a kommunikációt.

```
bool ButtonLedInitialized();
```

Visszatéríti, hogy a tégla gombjai inicializálva voltak-e vagy sem.

```
float HardwareVersion();
```

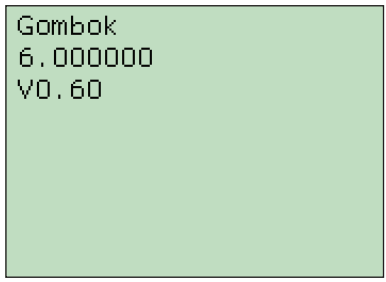
Visszatéríti a hardver verziószámát. A mi tesztesetünkben ez 6.0 volt. Részletek a 168. ábrán.

```
char* HardwareVersionString();
```

Visszatéríti a hardver verziószámát szöveges formátumban. A mi tesztesetünkben ez V0.60 volt.

A következő program a 168. ábrán látható kimenetet eredményezi:

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „C:\Apps\Bricx\API\ev3_lcd.h”
4. #include „C:\Apps\Bricx\API\ev3_command.h”
5. #include „C:\Apps\Bricx\API\ev3_button.h”
6.
7. int main()
8. {
9.   LcdInit();
10.   LcdText(1, 5, 5, „Gombok”);
11.   ButtonLedInit();
12.   ButtonLedOpen();
13.   Wait(SEC_1);
14.   float hv = HardwareVersion();
15.   char v[15];
16.   sprintf(v, „%f”, hv);
17.   LcdText(1, 5, 20, v);
18.   LcdText(1, 5, 35, HardwareVersionString());
19.   Wait(SEC_1);
20.   LcdClean();
21.   LcdExit();
22.   return 0;
23. }
```



168. ábra. *Verziószám*

```
void SetLedWarning(bool Value);
```

Bekapcsolja vagy kikapcsolja (a `TRUE` vagy `FALSE` paraméter szerint) a tégla figyelmeztető (narancssárga) LED-jeit.

```
byte LedWarning();
```

Visszatéríti, hogy a figyelmeztető (narancssárga) LED-ek be vannak-e kapcsolva (1) vagy sem (0).

```
void SetLedPattern(byte Pattern);
```

Beállítja a LED-ek mintázatát.

Az `ev3_constants.h`-ban lévő következő konstansokat tudjuk használni:

```
LED_BLACK         0
LED_GREEN         1
LED_RED           2
LED_ORANGE        3
LED_GREEN_FLASH   4
LED_RED_FLASH     5
LED_ORANGE_FLASH  6
LED_GREEN_PULSE   7
LED_RED_PULSE     8
LED_ORANGE_PULSE  9
NUM_LED_PATTERNS  10
```

Amint látjuk, a LED-eknek 10 mintázata lehetséges: nincsenek bekapcsolva, zöld, piros, narancssárga, felvillanó zöld, felvillanó piros, felvillanó narancssárga, villogó zöld, villogó piros, villogó narancssárga.

A következő program a LED-ek mindegyik mintázatát bemutatja:

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „C:\Apps\Bricx\API\ev3_lcd.h”
```

```
4. #include „C:\Apps\Bricx\API\ev3_command.h”
5. #include „C:\Apps\Bricx\API\ev3_button.h”
6.
7. int main()
8. {
9.   LcdInit();
10.   LcdText(1, 5, 5, „Gombok”);
11.   ButtonLedInit();
12.   ButtonLedOpen();
13.   int i;
14.   for(i = LED_BLACK; i < NUM_LED_PATTERNS; ++i)
15.   {
16.     SetLedPattern(i);
17.     Wait(SEC_1);
18.   }
19.   SetLedPattern(LED_BLACK);
20.   LcdClean();
21.   LcdExit();
22.   return 0;
23. }
```

```
byte LedPattern();
```

Visszatéríti, hogy a LED-ek éppen milyen mintázatot mutatnak.

```
word ButtonWaitForAnyEvent(unsigned int timeout);
```

A megadott `timeout` ideig vár egy akármilyen gombesemény bekövetkezésére. Visszatéríti a lenyomott, felengedett gomb, gombok értékét.

```
word ButtonWaitForAnyPress(unsigned int timeout);
```

A megadott `timeout` ideig vár egy akármilyen gomblenyomás bekövetkezésére. Visszatéríti a lenyomott gomb, gombok értékét.

Az alábbi program az Escape gomb lenyomásáig vár gombnyomásra, majd kiírja a kijelzőre, hogy melyik gomb volt lenyomva:

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „C:\Apps\Bricx\API\ev3_lcd.h”
4. #include „C:\Apps\Bricx\API\ev3_command.h”
5. #include „C:\Apps\Bricx\API\ev3_button.h”
6.
7. int main()
8. {
9.   LcdInit();
10.   ButtonLedInit();
11.   ButtonLedOpen();
```

```
12.   char v[20];
13.   sprintf(v, „Gombok”);
14.   while (true)
15.   {
16.     LcdClearDisplay();
17.     LcdText(1, 5, 5, „Gombok”);
18.     byte but = ButtonWaitForAnyPress(1000);
19.     if (but == 0) sprintf(v, „None”);
20.     if ((but & BUTTON_ID_ENTER) != 0)
21.         sprintf(v, „Enter”);
22.     if ((but & BUTTON_ID_LEFT) != 0)
23.         sprintf(v, „Left”);
24.     if ((but & BUTTON_ID_RIGHT) != 0)
25.         sprintf(v, „Right”);
26.     if ((but & BUTTON_ID_UP) != 0)
27.         sprintf(v, „Up”);
28.     if ((but & BUTTON_ID_DOWN) != 0)
29.         sprintf(v, „Down”);
30.     if ((but & BUTTON_ID_ESCAPE) != 0)
31.         sprintf(v, „Escape”);
32.     LcdText(1, 5, 15, v);
33.     Wait(SEC_1);
34.     if (but == BUTTON_ID_ESCAPE) break;
35.   }
36.   LcdClean();
37.   LcdExit();
38.   return 0;
39. }
```

```
bool ButtonIsUp(byte Button);
```

Jelzi, hogy a megadott `Button` gomb fel van-e engedve. A `Button` a következők egyike lehet: `BUTTON_ID_UP`, `BUTTON_ID_ENTER`, `BUTTON_ID_DOWN`, `BUTTON_ID_RIGHT`, `BUTTON_ID_LEFT`, `BUTTON_ID_ESCAPE`, `BUTTON_ID_ALL`.

```
bool ButtonIsDown(byte Button);
```

Jelzi, hogy a megadott `Button` gomb le van-e nyomva. A `Button` a következők egyike lehet: `BUTTON_ID_UP`, `BUTTON_ID_ENTER`, `BUTTON_ID_DOWN`, `BUTTON_ID_RIGHT`, `BUTTON_ID_LEFT`, `BUTTON_ID_ESCAPE`, `BUTTON_ID_ALL`.

```
void ButtonWaitForPress(byte Button);
```

Várakozik a megadott `Button` gomb lenyomásáig.

```
void ButtonWaitForPressAndReleae(byte Button);
```

Várakozik a megadott `Button` gomb lenyomásáig és felengedéséig.

Megjegyzés: Az eljárás neve `ButtonWaitForPressAndRelease` kellene hogy legyen, de minden valószínűség szerint elírták.

```
bool ButtonPressedEx(byte btn, bool resetCount);
```

NXC-stílusú API-függvény. Nincs lehetőség a gombnyomások megszámlálására, rövid vagy hosszú nyomás- vagy felengedés-időtartam érzékelésére. A `resetCount` paraméter is figyelmen kívül marad.

A `btn` paraméterben a lekérdezendő gombot kell megadni, ha ez le volt nyomva, akkor a függvény `TRUE` értéket térít vissza.

Tulajdonképpen megegyezik a `ButtonIsDown(btn)` függvénnyel.

```
bool ButtonPressed(byte btn);
```

A fenti függvény egyszerűsített változata, csak a `btn` paramétert kell megadni.

```
char ReadButtonEx(byte btn, bool reset, bool* pressed, word*
count);
```

NXC-stílusú API-függvény. Tulajdonképpen a `pressed` paraméterben visszatéríti, hogy a `btn` gomb le volt-e nyomva vagy sem.

```
byte ButtonState(byte btn);
```

Ha a `btn` gomb le van nyomva, akkor visszatérít egy `BTNSTATE_PRESSED_STATE | BTNSTATE_PRESSED_EV` értéket, különben egy `BTNSTATE_NONE` értéket.

3.3.4.7. Az időzítő programozása

Vannak olyan esetek, amikor az EV3 robotok időben kell hogy reagáljanak a bekövetkező eseményekre, vagy előre megadott ütemezés szerint kell hogy végezzék feladataikat (pl. azonos időközönként megmérik a fényértékeket). Ebből következően az EV3 tégla képes kell legyen az alábbiakra:

– Időtartam mérése
– Időalapú események generálása, ami lehet egyszeri vagy ismétlődő
– Megfelelő sebességgel reagálni az előre nem meghatározható időben bekövetkező eseményekre

Ebből következik, hogy szükségünk van olyan eszközökre és módszerekre, amelyek lehetővé teszik a hatékony időalapú tevékenység végzését. Főbb elemei ennek az eszköztárnak az időzítők vagy időszámlálók, amelyek lehetővé teszik számunkra az idő mérését, illetve a feladatok ütemezését.

Az EV3 tégla időzítőit az *ev3_timer.h* és *ev3_timer.c* modulokban (kell használni az `#include „C:\Apps\Bricx\API\ev3_timer.h”`-t) lévő típusok és függvények segítségével programozhatjuk.

Az időt centiszekundumban (CS – 100 ütem, ketyegés másodpercenként), milliszekundumban (MS – 1000 ütem, ketyegés másodpercenként) vagy mikroszekundumban (US – 1 000 000 ütem, ketyegés másodpercenként) mérhetjük.

Mindhárom időzítőből (CS, MS, US) négy-négy állhat rendelkezésünkre.

Az `ev3_constants.h`-ban lévő következő konstansokat tudjuk használni:

```
CS_TIMER_1    0
CS_TIMER_2    1
CS_TIMER_3    2
CS_TIMER_4    3
NUM_CS_TIMERS 4
```

```
MS_TIMER_1    0
MS_TIMER_2    1
MS_TIMER_3    2
MS_TIMER_4    3
NUM_MS_TIMERS 4
```

```
US_TIMER_1    0
US_TIMER_2    1
US_TIMER_3    2
US_TIMER_4    3
NUM_US_TIMERS 4
```

Az *ev3_timer.h* típusai:

```
typedef enum {
  ti10ms,
  ti50ms,
  ti100ms,
  ti250ms,
  ti500ms,
  ti1sec
} TimerInterval;

typedef void (*TimerCallback)(int sig);
```

Az *ev3_timer.h* függvényei:

```
void TimerInit();
```

Inicializálja az időzítőket.

```
void ClearTimer(byte Timer);
```

Törli a CS alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy a `CS_TIMER_1`, `CS_TIMER_2`, `CS_TIMER_3`, `CS_TIMER_4` konstansok valamelyike.

```
void ClearTimerMS(byte Timer);
```

Törli az MS alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `MS_TIMER_1`, `MS_TIMER_2`, `MS_TIMER_3`, `MS_TIMER_4` konstansok valamelyike.

```
void ClearTimerUS(byte Timer);
```

Törli az US alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `US_TIMER_1`, `US_TIMER_2`, `US_TIMER_3`, `US_TIMER_4` konstansok valamelyike.

```
void SetTimer(byte Timer, unsigned long Value);
```

Beállítja a CS alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `CS_TIMER_1`, `CS_TIMER_2`, `CS_TIMER_3`, `CS_TIMER_4` konstansok valamelyike. A `Value` a beállítandó érték.

```
void SetTimerMS(byte Timer, unsigned long Value);
```

Beállítja az MS alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `MS_TIMER_1`, `MS_TIMER_2`, `MS_TIMER_3`, `MS_TIMER_4` konstansok valamelyike. A `Value` a beállítandó érték.

```
void SetTimerUS(byte Timer, unsigned long Value);
```

Beállítja az US alapú időzítőt. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `US_TIMER_1`, `US_TIMER_2`, `US_TIMER_3`, `US_TIMER_4` konstansok valamelyike. A `Value` a beállítandó érték.

```
unsigned long Timer(byte Timer);
```

Egy lassú, másodpercenként 10 ütemű, ketyegésű időzítőértéket térít vissza.

```
unsigned long FastTimer(byte Timer);
```

Visszatéríti a CS alapú időzítő értékét. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `CS_TIMER_1`, `CS_TIMER_2`, `CS_TIMER_3`, `CS_TIMER_4` konstansok valamelyike.

```
unsigned long TimerMS(byte Timer);
```

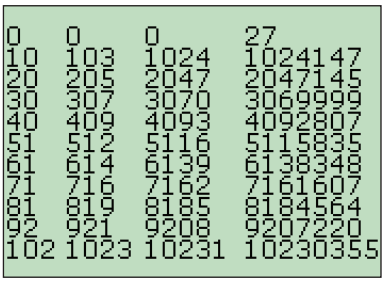
Visszatéríti az MS alapú időzítő értékét. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `MS_TIMER_1`, `MS_TIMER_2`, `MS_TIMER_3`, `MS_TIMER_4` konstansok valamelyike.

```
unsigned long TimerUS(byte Timer);
```

Visszatéríti az US alapú időzítő értékét. A `Timer` az időzítő száma lehet 0 és 3 között vagy az `US_TIMER_1`, `US_TIMER_2`, `US_TIMER_3`, `US_TIMER_4` konstansok valamelyike.

A következő példaprogram az időzítők inicializálása után kiírja a lassú, gyors (CS), MS, valamint US időzítők első 11 értékét. A program eredményét a 169. ábrán láthatjuk. Megfigyelhetjük, hogy a grafikus képernyőre való írás eléggé időigényes, az időzítők értékeit késlelteti.

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „C:\Apps\Bricx\API\ev3_lcd.h”
4. #include „C:\Apps\Bricx\API\ev3_command.h”
5. #include „C:\Apps\Bricx\API\ev3_button.h”
6. #include „C:\Apps\Bricx\API\ev3_timer.h”
7.
8. int main()
9. {
10.   LcdInit();
11.   LcdClearDisplay();
12.   TimerInit();
13.   ClearTimer(0);
14.   ClearTimerMS(0);
15.   ClearTimerUS(0);
16.   unsigned long t, ft, ms, us;
17.   int i;
18.   char s[10];
19.   for (i = 0; i <= 10; ++i)
20.   {
21.     t = Timer(0);
22.     ft = FastTimer(0);
23.     ms = TimerMS(0);
24.     us = TimerUS(0);
25.     sprintf(s, „%d”, t);
26.     LcdText(1, 0, 10 * (i+1), s);
27.     sprintf(s, „%d”, ft);
28.     LcdText(1, 28, 10 * (i+1), s);
29.     sprintf(s, „%d”, ms);
30.     LcdText(1, 65, 10 * (i+1), s);
31.     sprintf(s, „%d”, us);
32.     LcdText(1, 112, 10 * (i+1), s);
33.     Wait(SEC_1);
34.   }
35.   LcdClean();
36.   LcdExit();
37.   return 0;
38. }
```



169. ábra. *Időzítők*

```
unsigned long long TimerGetCS();
```

A `gettimeofday(&tv, 0);` C függvény segítségével visszatéríti a Unix Epoch (0:00:00 January 1, 1970 GMT) óta eltelt időt centiszekundumokban (CS).

```
unsigned long long TimerGetMS();
```

A `gettimeofday(&tv, 0);` C függvény segítségével visszatéríti a Unix Epoch (0:00:00 January 1, 1970 GMT) óta eltelt időt milliszekundumokban (MS).

```
unsigned long long TimerGetUS();
```

A `gettimeofday(&tv, 0);` C függvény segítségével visszatéríti a Unix Epoch (0:00:00 January 1, 1970 GMT) óta eltelt időt mikroszekundumokban (MS).

```
unsigned long TimerWait(unsigned long Time);
```

Visszatéríti a `TimerGetMS()` értékét, hozzáadva a Time értéket.

```
void TimerReady(unsigned long Timer);
```

Vár, ameddig a `Time` paraméterrel milliszekundumban megadott idő lejár, vagyis `Timer > TimerGetMS()`.

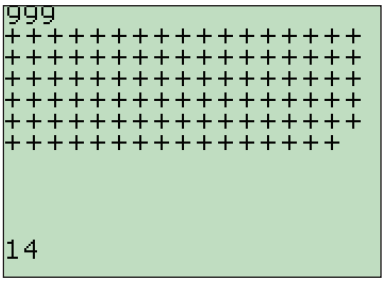
```
void SetTimerCallback(TimerInterval interval, TimerCallback
callback);
```

A függvény segítségével az időzítők használatának másik nagy lehetőségét tudjuk leprogramozni.

Gyakran szükségünk lehet bizonyos időközönként (periódusként) megszakítani a program normális futását, elvégezni valamilyen műveletet, majd visszatérni a program normális futásához. Ezt is időzítővel tudjuk megoldani, mégpedig úgy, hogy a `SetTimerCallback` függvény segítségével megadunk egy időintervallumot (`interval` – milliszekundumokban), amely eltelte után újra és újra meghívódik a második paraméterként megadott függvény (`callback`).

A következő példaprogram 100 milliszekundomonként kirajzol egy „+” jelt a képernyőre. 10 ezer milliszekundum futás után a program eredményét a 170. ábrán láthatjuk.

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „C:\Apps\Bricx\API\ev3_lcd.h”
4. #include „C:\Apps\Bricx\API\ev3_command.h”
5. #include „C:\Apps\Bricx\API\ev3_button.h”
6. #include „C:\Apps\Bricx\API\ev3_timer.h”
7.
8. int x=0, y=10;
9.
10. void Handler(int sig)
11. {
12.   LcdText(1, x, y, „+”);
13.   x+=10;
14.   if (x > 177)
15.   {
16.    X = 0;
17.    Y += 10;
18.   }
19.   char s[10];
20.   sprintf(s, „%d”, sig);
21.   LcdText(1, 0, 110, s);
22. }
23.
24. int main()
25. {
26.   LcdInit();
27.   LcdClearDisplay();
28.   TimerInit();
29.   SetTimerCallback(ti100ms, &Handler);
30.   int i = 0;
31.   char s[10];
32.   while (i < 1000)
33.   {
34.     sprintf(s, „%d”, i);
35.     LcdText(1, 0, 0, s);
36.     Wait(10);
37.     ++i;
38.   }
39.   LcdClean();
40.   LcdExit();
41.   return 0;
42. }
```



170. ábra. *Megadott időközönként ismétlődő esemény*

3.3.4.8. A kimenet programozása

Az EV3 tégla kimeneteit, vagyis a motorokat az ev3_output.h és ev3_output.c modulokban (kell használni az #include „C:\Apps\Bricx\API\ev3_output.h”-t) lévő függvények segítségével programozhatjuk [34].

```
bool OutputInit(void);
```
Inicializálja a kimenetet.

```
bool OutputOpen(void);
```
Megnyitja a kimenetet.

```
bool OutputClose(void);
```
Lezárja a kimenetet.

```
bool OutputExit(void);
```
Kilép az output modulból. Lezárja az összes kimenetet.

```
bool OutputProgramStop(void);
```
Megállítja a motorokat.

```
bool OutputInitialized(void);
```
Visszatéríti, hogy a kimenet inicializálva volt-e vagy sem.

```
bool OutputStop(byte Outputs, bool useBrake);
```
Megállítja a megadott kimeneteken található motorokat (`Outputs`), be lehet állítani, hogy azonnal megálljanak (brake), vagy hogy a motorok áramellátása kikapcsoljon, és azok addig forogjanak tehetetlenségből szabadon, amíg meg nem állnak (coast).

```
bool OutputSetType(byte Output, char DeviceType);
```

Beállítja a megfelelő kimenet (`Output`) típusát (`DeviceType`).

```
bool OutputSetTypesArray(char* pTypes);
```

Egy tömb segítségével állítja be a kimenetek (motorok) típusait.

```
bool OutputSetTypes(char OutputA, char OutputB, char OutputC,
char OutputD);
```

Egyenként beállítja a kimenetek típusait.

```
bool OutputReset(byte Outputs);
```

Újraállítja (reszetálja) a kimeneteket, vagyis nullára állítja a pozícióikat.

```
bool OutputSpeed(byte Outputs, char Speed);
```

Beállítja a megfelelő kimenetek sebességét (a polaritáshoz viszonyítva). Lehetővé teszi a szabályozást, ha a kimenet fordulatszámmérővel (tachométerrel) rendelkezik.

```
bool OutputPower(byte Outputs, char Power);
```

Beállítja a megfelelő kimenet teljesítményét, erejét. Felfüggeszti a szabályozást és a pozicionálást.

```
bool OutputStartEx(byte Outputs, byte Owner);
```

A megadott kimeneteket indítja el.

```
bool OutputStart(byte Outputs);
```

`OWNER_NONE` beállítással indítja el az `Outputs`-szal megadott kimeneteket.

```
bool OutputPolarity(byte Outputs, char Polarity);
```

Beállítja a megadott kimenetek polaritását. Ha a `Polarity` 1, a motor előre, ha –1, akkor hátra fog forogni, a 0-val pedig vált, megfordítja az irányát.

```
bool OutputRead(byte Output, char* Speed, int* TachoCount,
int* TachoSensor);
```

A megadott kimenet adatait olvassa ki. A `Speed` a sebesség, a `TachoCount` tachométer fordulatainak száma, a `TachoSensor` pedig a tachoérzékelő értéke. Ezeket az értékeket az utolsó visszaállítástól (reszetálástól) számolja.

```
bool OutputTest(byte Outputs, bool* isBusy);
```

Ellenőrzi, hogy melyik kimenet foglalt.

```
bool OutputState(byte Outputs, byte* State);
```

Beállítja az összes foglalt kimenet bitmaszkját (`State`).

```
bool OutputClearCount(byte Outputs);
```

Ha a motort érzékelő üzemmódban használjuk, törli a tachoszámot, vagyis lenullázza a fordulatszámmérőt.

```
bool OutputGetCount(byte Output, int* Tacho);
```

Ha a motort érzékelő üzemmódban használjuk, visszatéríti a fordulatszámmérő értékét (`Tacho`).

```
bool OutputGetTachoCount(byte Output, int* Tacho);
```

Ha a motort érzékelő üzemmódban használjuk, visszatéríti a fordulatszámmérő értékét az utolsó visszaállítás óta (`Tacho`).

```
bool OutputGetActualSpeed(byte Output, char* Speed);
```

Visszatéríti a motor aktuális sebességét (`Speed`).

```
bool OutputStepPowerEx(byte Outputs, char Power, int Step1,
int Step2, int Step3, bool useBrake, byte Owner);
```

Beállítja a kimenetek teljesítményét (`Power`), a rámpalépéseket (fel, állandó, le), valamint a megállási módot (`useBrake`).

```
bool OutputStepPower(byte Outputs, char Power, int Step1, int
Step2, int Step3);
```

Az előbbi függvényt hívja alapértelmezett `useBrake TRUE` és `Owner OWNER_NONE` beállításokkal.

```
bool OutputTimePowerEx(byte Outputs, char Power, int Time1,
int Time2, int Time3, bool useBrake, byte Owner);
```

Az `OutputStepPowerEx` függvényhez hasonló beállító függvény, csak nem a lépéseket, hanem az időt állítja be.

```
bool OutputTimePower(byte Outputs, char Power, int Time1, int
Time2, int Time3);
```

Az előbbi függvényt hívja alapértelmezett `useBrake TRUE` és `Owner OWNER_NONE` beállításokkal.

```
bool OutputStepSpeedEx(byte Outputs, char Speed, int Step1,
int Step2, int Step3, bool useBrake, byte Owner);
```

Az `OutputStepPowerEx` függvényhez hasonló beállító függvény, csak nem a teljesítményt, hanem a sebességet állítja be.

```
bool OutputStepSpeed(byte Outputs, char Speed, int Step1, int
Step2, int Step3)
```

Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
bool OutputTimeSpeedEx(byte Outputs, char Speed, int Time1,
int Time2, int Time3, bool useBrake, byte Owner);
```

Az `OutputStepSpeedEx` függvényhez hasonló beállító függvény, csak nem a lépéseket, hanem az időt állítja be.

```
bool OutputTimeSpeed(byte Outputs, char Speed, int Time1, int
Time2, int Time);
```

Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
bool OutputStepSyncEx(byte Outputs, char Speed, short Turn,
int Step, bool useBrake, byte Owner);
```

Kormányzási módot valósít meg két motor segítségével. Az `Outputs` paraméter tehát csak `OUT_AB`, `OUT_AC`, `OUT_AD`, `OUT_BC`, `OUT_BD` vagy `OUT_CD` lehet. Beállítja a sebességet (`Speed`), a fordulás szögét (`Turn`), a lépést (`Step`), a megállási módot (`useBrake`), valamint a tulajdonost (`Owner`).

```
bool OutputStepSync(byte Outputs, char Speed, short Turn, int
Step);
```

Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
bool OutputTimeSyncEx(byte Outputs, char Speed, short Turn,
int Time, bool useBrake, byte Owner);
```

Kormányzási módot valósít meg két motor segítségével, tehát az `OutputStepSync` függvényhez hasonló beállító függvény, ám nem a lépést, hanem a motor működési idejét (`Time`) állítja be.

```
bool OutputTimeSync(byte Outputs, char Speed, short Turn, int
Time);
```

Az előbbi függvényt hívja alapértelmezett `useBrake` `TRUE` és `Owner` `OWNER_NONE` beállításokkal.

```
void SetOutputEx(byte Outputs, byte Mode, byte reset);
```

Beállítja a kimenet módját (`Mode`). Ez a mód `OUT_FLOAT`, `OUT_OFF` vagy `OUT_ON` lehet. Az `OUT_ON` bekapcsolja a motort, az `OUT_OFF` kikapcsolja, az `OUT_FLOAT`

pedig kikapcsolja az áramellátást, de hagyja a motort szabadon megállni. A `reset` segítségével a fordulatszámmérőt állíthatjuk vissza, reszetálhatjuk.

```
void SetOutput(byte Outputs, byte Mode);
```
Az előbbi függvényt hívja alapértelmezett `reset` `RESET_NONE` beállítással.

```
void SetDirection(byte Outputs, byte Dir);
```
Beállítja a motor irányát. Ha a `Dir` 1 (`OUT_FWD`), a motor előre, ha –1 (`OUT_REV`), akkor hátra fog forogni, a 0-val (`OUT_TOGGLE`) pedig vált, megfordítja az irányát.

```
void SetPower(byte Outputs, char Power);
```
Beállítja a motor teljesítményét.

```
void SetSpeed(byte Outputs, char Speed);
```
Beállítja a motor sebességét.

```
void OnEx(byte Outputs, byte reset);
```
Bekapcsolja a motort.

```
void OffEx(byte Outputs, byte reset);
```
Kikapcsolja a motort.

```
void FloatEx(byte Outputs, byte reset);
```
Kikapcsolja a motor áramellátását, és hagyja magától megállni.

```
void On(byte Outputs);
```
Az `OnEx` függvényt hívja alapértelmezett `RESET_NONE` paraméterrel.

```
void Off(byte Outputs);
```
Az `OffEx` függvényt hívja alapértelmezett `RESET_NONE` paraméterrel.

```
void Float(byte Outputs);
```
A `FloatEx` függvényt hívja alapértelmezett `RESET_NONE` paraméterrel.

```
void Coast(byte Outputs);
```
A `FloatEx` függvényt hívja alapértelmezett `RESET_NONE` paraméterrel.

```
void Toggle(byte Outputs);
```
Megváltoztatja a motor forgási irányát.

```
void Fwd(byte Outputs);
```
Beállítja a motor előreforgását.

```
void Rev(byte Outputs);
```

Beállítja a motor visszaforgását.

```
void OnFwdEx(byte Outputs, char Power, byte reset);
```

Beállítja a motor előreforgását, teljesítményét, valamint a fordulatszámmérőjét. Bekapcsolja a motort.

```
void OnRevEx(byte Outputs, char Power, byte reset);
```

Beállítja a motor visszaforgását, teljesítményét, valamint a fordulatszámmérőjét. Bekapcsolja a motort.

```
void OnFwd(byte Outputs);
```

Bekapcsolja a motort előre irányba, `OUT_POWER_DEFAULT`, valamint `RESET_NONE` alapértelmezett paraméterekkel.

```
void OnRev(byte Outputs);
```

Bekapcsolja a motort vissza irányba, `OUT_POWER_DEFAULT`, valamint `RESET_NONE` alapértelmezett paraméterekkel.

```
void OnFwdRegEx(byte Outputs, char Speed, byte RegMode, byte reset);
```

Bekapcsolja a motort előre irányba, beállítja a sebességét és a fordulatszámmérőt. Ebben a fejlesztésben a `RegMode` paraméter nem játszik szerepet.

```
void OnRevRegEx(byte Outputs, char Speed, byte RegMode, byte reset);
```

Bekapcsolja a motort vissza irányba, beállítja a sebességét és a fordulatszámmérőt. Ebben a fejlesztésben a `RegMode` paraméter nem játszik szerepet.

```
void OnFwdReg(byte Outputs, char Speed);
```

Az `OnFwdRegEx` függvényt hívja `OUT_REGMODE_SPEED` és `RESET_NONE` alapértelmezett paraméterekkel.

```
void OnRevReg(byte Outputs, char Speed);
```

Az `OnRevRegEx` függvényt hívja `OUT_REGMODE_SPEED` és `RESET_NONE` alapértelmezett paraméterekkel.

```
void OnFwdSyncEx(byte Outputs, char Speed, short Turn, byte reset);
```

Kormányzási módot valósít meg két motor segítségével előre irányba, beállítja a sebességet, a fordulatot, valamint a fordulatszámmérőt.

```
void OnRevSyncEx(byte Outputs, char Speed, short Turn, byte
reset);
```

Kormányzási módot valósít meg két motor segítségével vissza irányba, beállítja a sebességet, a fordulatot, valamint a fordulatszámmérőt.

```
void OnFwdSync(byte Outputs, char Speed);
```

Az `OnFwdSyncEx` függvényt hívja 0 fordulási szöggel és `RESET_NONE` alapértelmezett paraméterrel.

```
void OnRevSync(byte Outputs, char Speed);
```

Az `OnRevSyncEx` függvényt hívja 0 fordulási szöggel és `RESET_NONE` alapértelmezett paraméterrel.

```
void RotateMotorNoWaitEx(byte Outputs, char Speed, int Angle,
short Turn, bool Sync, bool Stop);
```

Bekapcsolja a motort párhuzamos módon, vagyis a következő utasítás is végrehajtódni kezd. Megadhatjuk a sebességet (`Speed`), a fordulat szögét (`Angle`), a fordulást (`Turn`), ha a `Sync` `true`, akkor kormányzási módba kerülünk, ha nem, akkor csak egy motort indít, megadhatjuk a megállás módját (`Stop`).

```
void RotateMotorNoWait(byte Outputs, char Speed, int Angle);
```

Az előbbi függvényt hívja `Turn = 0`, `Sync = true`, `Stop = true` alapértelmezett értékekkel.

```
void RotateMotorEx(byte Outputs, char Speed, int Angle, short
Turn, bool Sync, bool Stop);
```

A `RotateMotorNoWaitEx` függvényhez hasonló függvény, ám míg a motor működik, más utasítás nem hajtódik végre.

```
void RotateMotor(byte Outputs, char Speed, int Angle);
```

Az előbbi függvényt hívja `Turn = 0`, `Sync = true`, `Stop = true` alapértelmezett értékekkel.

```
void OnForSyncEx(byte Outputs, int Time, char Speed, short
Turn, bool Stop);
```

Az `OutputTimeSyncEx` függvényt hívja.

```
void OnForSync(byte Outputs, int Time, char Speed);
```

Az előbbi függvényt hívja `Turn = 0`, `Stop = true` alapértelmezett értékekkel.

```
void OnForEx(byte Outputs, int Time, char Power, byte reset);
```

A megadott ideig (`Time`) működteti a motort a megadott teljesítménnyel (`Power`). Beállíthatjuk a fordulatszámmérőt is.

```
void OnFor(byte Outputs, int Time);
```

Az előbbi függvényt hívja `Power = OUT_POWER_DEFAULT`, `reset = RESET_NONE` alapértelmezett értékekkel.

```
void ResetTachoCount(byte Outputs);
```

Reszetálja (visszaállítja) a fordulatszámmérőt vagy -mérőket.

```
void ResetAllTachoCounts(byte Outputs);
```

Reszetálja (visszaállítja) az összes fordulatszámmérőt.

```
void ResetBlockTachoCount(byte Outputs);
```

A `ResetTachoCount` függvény szinonimája.

```
void ResetRotationCount(byte Outputs);
```

Reszetálja (visszaállítja) a fordulatszámmérőt vagy -mérőket.

```
void ResetCount(byte Outputs, byte reset);
```

Reszetálja (visszaállítja) a fordulatszámmérőt vagy -mérőket. A reset paraméter `RESET_COUNT`, `RESET_BLOCK_COUNT`, `RESET_ROTATION_COUNT`, `RESET_BLOCKANDTACHO` vagy `RESET_ALL` lehet, s ennek függvényében hívja az előbbi függvények valamelyikét.

```
int MotorTachoCount(byte Output);
```

Visszatéríti a megadott motor fordulatszámmérőjének értékét.

```
int MotorBlockTachoCount(byte Output);
```

Az előbbi függvény szinonimája.

```
char MotorPower(byte Output);
```

Visszatéríti a motor aktuális sebességét.

```
char MotorActualSpeed(byte Output);
```

Az előbbi függvény szinonimája.

```
int MotorRotationCount(byte Output);
```

Visszatéríti a motor fordulatszámát.

```
bool MotorBusy(byte Output);
```

Visszatéríti, hogy a motor foglalt-e vagy sem.

A következő program segítségével a motorokat teszteljük.

```
1. #include „c:\APPS\Bricx\API\ev3_command.h”
2. #include „c:\APPS\Bricx\API\ev3_output.h”
3.
4. int main()
5. {
6.   OutputInit();
7.   ResetAllTachoCounts(OUT_ABCD);
8.   SetPower(OUT_A, 90);
9.   SetSpeed(OUT_B, 40);
10.   SetPower(OUT_C, 60);
11.   SetPower(OUT_D, -60);
12.   On(OUT_ALL);
13.   Wait(SEC_5);
14.   Float(OUT_ALL);
15.   OutputClose();
16.   OutputExit();
17.   return 0;
18. }
```

Sajnos John Hansen fejlesztései itt leálltak, az érzékelőket nem programozta le, ám saját fejlesztésű programjainkban jól tudjuk ezeket a modulokat használni, a többit pedig leprogramozzuk mi.

9. feladat

Tervezzük meg egy olyan robot programját, amelyik meg tudja oldani rekurzívan a Hanoi tornyai feladatot!

Mivel most imperatív nyelven programozunk, nyugodtan használhatunk rekurzív hívást is. A program így a következő lesz:

```
1. #include <stdio.h>
2. #include <unistd.h>
3. #include „C:\Apps\Bricx\API\ev3_timer.h”
4. #include „C:\Apps\Bricx\API\ev3_lcd.h”
5. #include „C:\Apps\Bricx\API\ev3_command.h”
6. #include „C:\Apps\Bricx\API\ev3_output.h”
7.
8. char mes[6];
9.
```

```
10. void hanoi(int k, char s, char d, char h)
11. {
12.   if (k == 1)
13.   {
14.     sprintf(mes, „%c -> %c\n”, s, d);
15.     LcdText(1, 0, 0, mes);
16.     LcdRefresh();
17.     Wait(SEC_1);
18.   }
19.   else
20.   {
21.     hanoi(k-1, s, h, d);
22.     sprintf(mes, „%c -> %c\n”, s, d);
23.     LcdText(1, 0, 0, mes);
24.     LcdRefresh();
25.     Wait(SEC_1);
26.     hanoi(k-1, h, d, s);
27.   }
28. }
29.
30. int main()
31. {
32.   LcdInit();
33.   LcdOpen();
34.   int n = 3;
35.   hanoi(n, ‚0’, ‚1’, ‚2’);
36.   LcdExit();
37.   LcdClose();
38.   return 0;
39. }
```

3.3.5. A Microsoft MakeCode

A Microsoft MakeCode egy olyan webes felület, amely a számítógépek és külső eszközök (pl. micro:bit [36], LEGO® MINDSTORMS® Education EV3, más áramköri és robotikai hardverek stb.) kódolását, programozását segíti elő. Az oktatási céloknak tökéletesen megfelel, a változatos alkalmazások robotika klubok működését, tudományos óratartást, gyakorlatok levezetését teszi lehetővé nagyon egyszerűen, egységes keretben.

Rengeteg tutorial, oktatóvideó, bevezető tanfolyam segíti az egyéni vagy csoportos projektek megvalósítását.

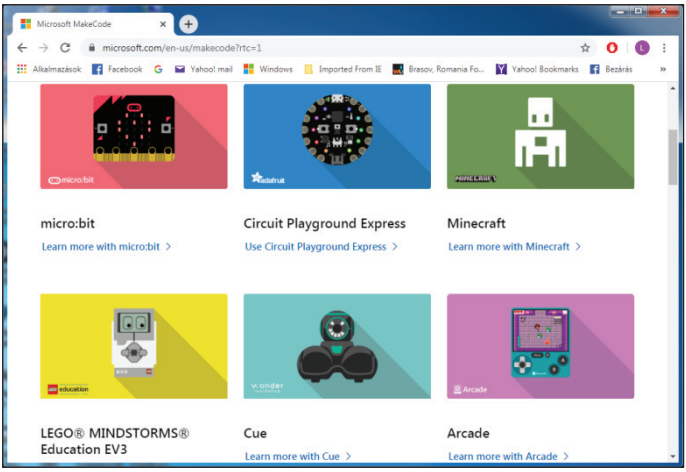
A felület érdekessége, hogy vizuális paradigmára épülve grafikusan tudjuk összerakni a programokat, ám a háttérben JavaScript kódot is generál, tehát meglesz a teljes imperatív program is, így a felület két legfontosabb része a Blocks és a JavaScript.

Így a Microsoft MakeCode felülete ideális kezdők számára, hiszen blokkokból állíthatjuk össze az alkalmazásokat (mint például a Scratch nyelv esetén), a képernyő bal oldalán látható szimulátorban pedig ellenőrizni lehet a kód eredményét anélkül, hogy azt az eszközre rátöltenénk, de a haladók számára is ideális, mert ők válthatnak egy teljes funkcionalitású JavaScript szerkesztőre. Az ide-oda konvertálást (blokkokból JavaScriptbe és vissza) a felület megoldja.

A megírt vagy megtervezett program futtatása a roboton úgy történik, hogy a kódot (bináris, futtatható változatban – *.UF2 állomány lementjük a honlapról a számítógépre, majd ezt az állományt egyszerűen átmásoljuk a LEGO robotra. Ennek érdekében nyilván arra van szükség, hogy a számítógép meghajtóként, lemezegységként ismerje fel a LEGO téglát, és így lehessen rá állományokat másolni.

3.3.5.1. Telepítés

A MakeCode felületet nem kell telepíteni, egyszerűen elérhető akármilyen böngészőből a *https://www.microsoft.com/makecode* honlapon (171. ábra).



171. ábra. *A MakeCode honlapja*

Amint már említettük, ahhoz, hogy a MakeCode környezetet használni tudjuk robotunk programozására, a téglára telepíteni kell legalább a 1.10E verziójú Education firmware-t. Ha meg szeretnénk nézni a robotunk firmware verzióját, megtehetjük úgy, hogy a tégla beállításainál kiválasztjuk a Brick Info menüpontot (172. ábra).

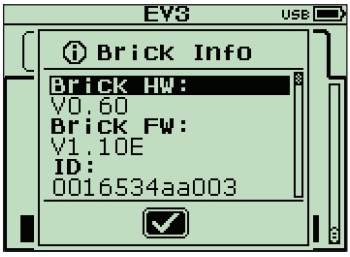
Ha a Brick FW-nél nem a V1.10E vagy ennél nagyobb verzió jelenik meg, akkor a *https://education.lego.com/en-us/support/mindstorms-ev3/firmware-update* honlapon található leírás alapján telepítenünk kell az új firmware-t.

Ezt kétféleképpen tehetjük meg:

– Az online EV3 Device Manager-t használva,

– A LEGO® MINDSTORMS® Education EV3 szoftver segítségével.

Ha át szeretnénk állni ennek a firmware-nek a használatára, mindenképpen javasoljuk letölteni és telepíteni a LEGO MINDSTORMS Education EV3 szoftvert, mert így újabb lehetőségek nyílnak meg a robotprogramozásban (nem sokban tér el a LEGO MINDSTORMS EV3 Home Edition szoftvertől, de vannak új lehetőségek benne).



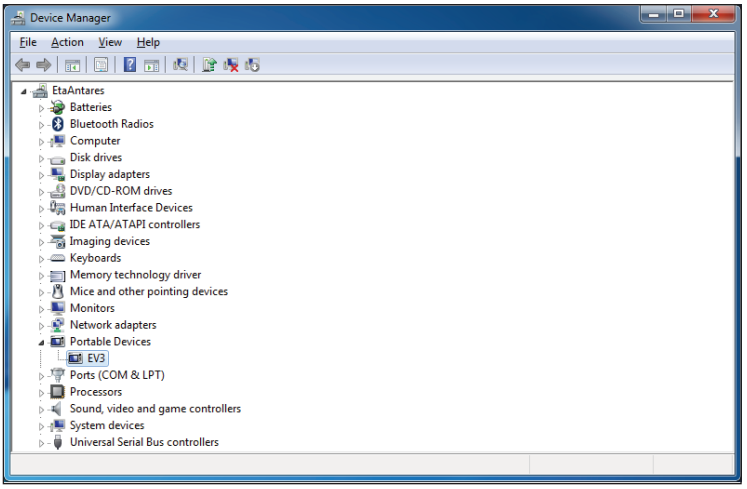
172. ábra. *A firmware verzió*

Mindenképpen jó, ha a fent említett honlapról letöltjük a saját számítógépünkre az új verziójú firmware-t, így könnyebben tudjuk cserélni, például ha vissza szeretnénk térni a Home Editionra.

Mentsük tehát le a *LME-EV3_Firmware_1.10E.bin* állományt vagy ennek egy újabb verzióját.

Sajnos megtörténhet, hogy a firmware telepítése után sem ismeri fel a Windows a LEGO téglát meghajtóként, ebben az esetben a Device Manager segítségével (Scan for hardware changes) próbáljuk meg felismertetni a LEGO téglát, és automatikusan telepíteni a szükséges drivereket. A 173. ábra a telepített drivereket mutatja a Device Managerben – ehhez hasonló kell hogy megjelenjen nálunk is (Portable Devices – EV3).

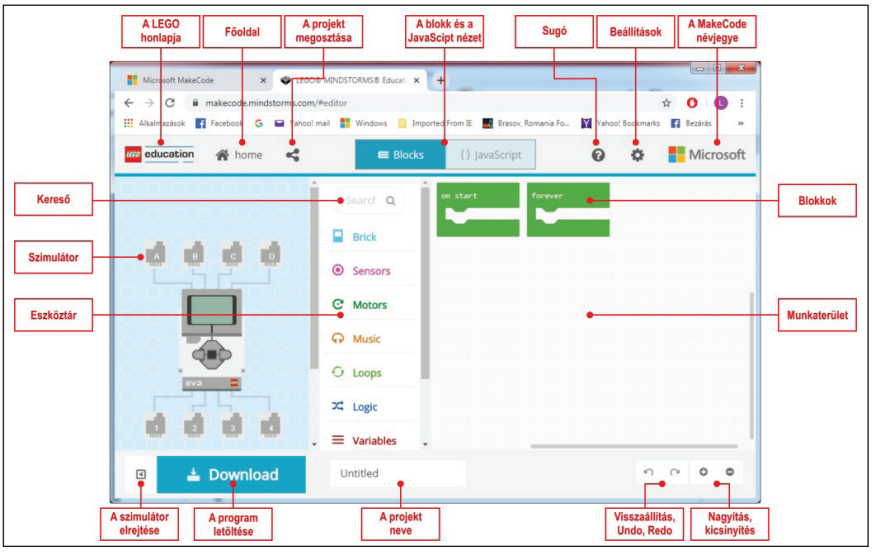
Ha sikeresen települtek a driverek, és az USB-porton keresztül a tégla össze van kötve a számítógéppel, akkor új meghajtóként megjelenik a LEGO robot. Látjuk a könyvtárszerkezetet, állományokat tudunk rámásolni vagy lemásolni róla, így már semmi akadálya annak, hogy használni tudjuk a MakeCode környezetet.



172. ábra. *Telepített driverek*

3.3.5.2. A MakeCode használata

A *https://www.microsoft.com/makecode* honlapon válasszuk ki a LEGO® MINDSTORMS® Education EV3 lehetőséget (*https://makecode. mindstorms. com/*), majd a megjelenő új oldalon hozzunk létre egy új projektet (+ New Project). Ekkor a 173. ábrán látható felületet kapjuk, és itt már megírhatjuk programjainkat.



173. ábra. *A MakeCode felülete*

A MakeCode felületen is, mint ahogy azt már a LEGO MINDSTORMS EV3 Home Edition esetében is megszoktuk, a programozás azt jelenti, hogy a működés szerint csoportosított *Eszköztár*ból blokkokat választunk ki, majd azokat a *Munkaterület*en összerakjuk más blokkokkal. Így építjük fel a programot.

Különbségek:

– nincs adathuzal, az adatokat változók segítségével adhatjuk át a blokkok között;
– nincs Start, Stop blokk, eseményvezérelt programozásról beszélhetünk, eseményblokkok vannak (on start, forever, run in parallel, on button, on touch, on color sensor stb.);
– szimulátorban azonnal látjuk az eredményt;
– a rendszer a háttérben JavaScript kódot generál;
– kézzel kell rámásolni a LEGO téglára a kódot.

A szimulátor esetében lehetőségünk van:

– a szimulátor leállítására;
– a szimulátor újraindítására;
– a szimulátor lelassítására (Slow Motion);
– a szimulátor hangjának elnémítására;
– a szimulátor kinagyítására teljes képernyős üzemmódba.

Az eszköztár nagy csoportjai a következők:

– Tégla (Brick)
    • Gombok (Buttons)
    • Kijelző (Screen)
    • Elem (Battery)
– Érzékelők (Sensors)
    • Érintésérzékelő (Touch Sensor)
    • Színérzékelő (Color Sensor)
    • Ultrahang-érzékelő (Ultrasonic Sensor)
    • Giroszkópikus érzékelő (Gyro Sensor)
    • Infravörös érzékelő (Infrared Sensor)
    • Távirányító (Remote Infrared Beacon)
    • Kalibrálás (Calibration)
– Motorok (Motors)
    • Mozgás (Move)
    • Számlálók (Counters)
    • Tulajdonságok (Properties)
– Zene (Music)
– Ciklusok (Loops)
– Logika (Logic)
    • Feltételek (Conditonals)

  - Összehasonlítások (Comparition)
  - Logikai (Boolean)
- Változók (Variables)
- Matematika (Math)
- Függvények (Functions)
- Tömbök (Arrays)
- Szöveg (Text)
- Konzol (Console)
- Kontroll (Control)

A fentieken kívül lehetőség van új kiegészítők (Extensions) telepítésére is a rendszerbe.

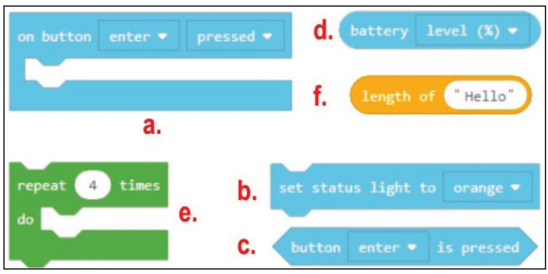
A blokkok nagy része hasonló a LEGO MINDSTORMS EV3 Home Editionban lévő blokkokhoz, a funkcionalitásuk, beállításaik teljesen megegyezők, legfennebb a nevük más, vagy ugyanazt a műveletet több blokk segítségével lehet leírni.

A következő blokkokat tudjuk megkülönböztetni (174. ábra):

- Események (a.);
- Utasítások (b.);
- Logikai értékek (c.);
- Numerikus értékek (d.);
- Utasítás blokkok (e.);
- Szöveges értékek (f.).

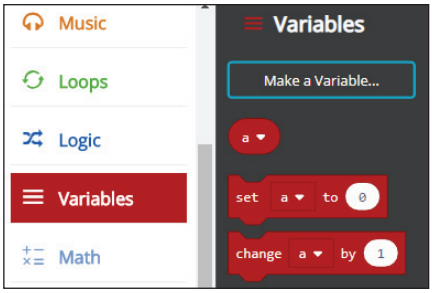
Gyakorlatilag ezekből a típusú blokkokból épül fel a teljes program, az eszköztáron ezen kívül létezik saját használatra létrehozandó blokkoknak is fenntartott hely, ezek a Változók és a Függvények.

Ha egy saját változót akarunk létrehozni, akkor a Változók (Variables) lehetőséget kell válasszuk, majd innen a Változó létrehozása… (Make a Variable…) gombot. Ekkor a megjelenő párbeszédablakban meg kell hogy adjuk a változó nevét. Sem típust, sem semmi más tulajdonságot nem kell megadni, csak egy nevet.



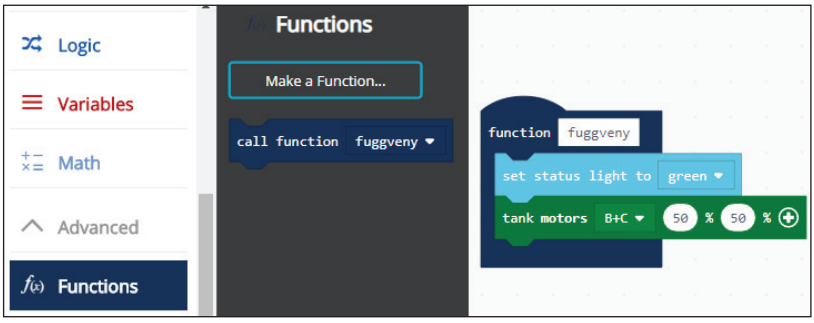
174. ábra. *Blokktípusok*

Ha létrehoztunk egy változót (175. ábra), akkor az Eszköztár Változók fülében automatikusan létrejön egy numerikus blokk a változó nevével, valamint két utasításblokk, az egyik a változó értékének beállítására (set), a másik a változó értékének módosítására (change).



175. ábra. *Változók*

Ha egy új függvényt szeretnénk létrehozni, akkor az Eszköztár Függvények fülében lévő Függvény létrehozása... (Make a Function...) gombot kell megnyomni. A párbeszédablakban meg kell adni egy nevet, majd a munkaterületen megjelenik egy új lila függvényblokk. Ide helyezhetjük el a függvény utasításait. Paraméterek, visszatérési érték megadására nincs lehetőség, ezeket változók segítségével állíthatjuk be (például a micro:bit programozásánál erre van lehetőség, az EV3-nál viszont nincs). Rekurzív függvények írására sincs lehetőség. A függvény csupán az utasításokat zárja egybe. A Függvények fülben automatikusan létrejön egy függvényhívási utasítás is (176. ábra).



176. ábra. *Függvények*

A Beállítások gomb segítségével testre szabhatjuk a munkafelületünket. Nevet adhatunk a projektnek (Project Settings), kezelhetjük a kiterjesztéseket

(Extensions), kinyomtathatjuk a projektet (Print…), lementhetjük a munkánkat (Save Project), kitörölhetjük azt (Delete Project), valamilyen visszaélést, hibát jelenthetünk be (Report Abuse), beállíthatjuk a felület nyelvét (Language). Itt jegyezzük meg, hogy míg például a micro:bit esetén 27 nyelv közül választhatunk, köztük a magyar nyelvet is, addig a LEGO EV3 esetén egyelőre csak 5 nyelv érhető el (magyar nincs), a honlap kezelői fel is szólítanak, hogy segítsünk a fordításban.

A fenti beállítások mellett lehetőség van magasabb kontraszt beállítására (High Contrast – On, Off), ekkor a képernyő háttere feketére vált, valamint zöld háttér beállítására (Green Screen – On, Off) is. Ez igen hasznos, ha videót szeretnénk készíteni vagy képeket szeretnénk lementeni.

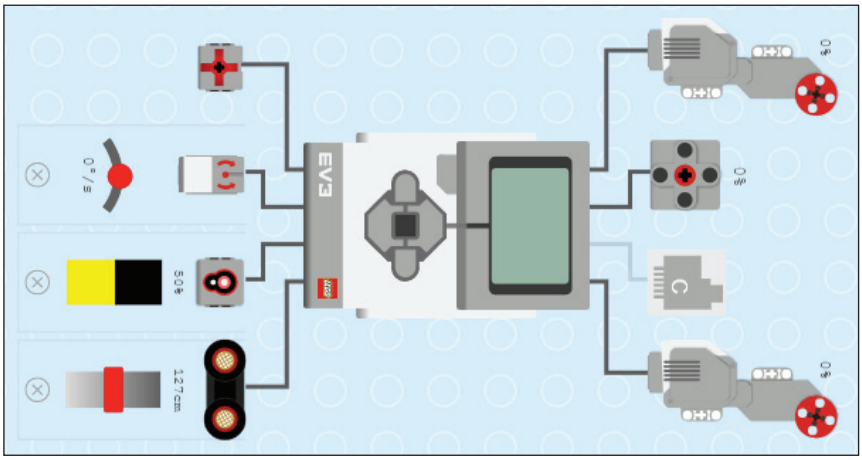
A Visszaállítás (Reset) menüpont törli az összes projektünket (ennek a törlésnek nincs visszavonás művelete), a Névjegy (About…) menüpont segítségével pedig a LEGO és a MakeCode verziószámát, felhasználási feltételeit tekinthetjük meg.

Ha a Blokk Munkaterületen a jobb egérgombbal kattintunk, az előjövő környezetérzékeny menü a következő lehetőségeket kínálja fel:

– Megjegyzés hozzáadása (Add Comment) – segítségével megjegyzésblokkokat helyezhetünk el a Munkaterületen;
– Az összes blokk törlése (Delete All Blocks) – töröljük a Munkaterületre helyezett összes blokkot;
– A kód formázása (Format Code) – szépen, automatikusan rendezi, elhelyezi a blokkokat;
– Képernyőfotó mentése (Download Screenshot) – képként elmenti a Munkaterületet.

Ha a JavaScript Munkaterületen a jobb egérgombbal kattintunk, az előjövő környezetérzékeny menü a következő lehetőségeket kínálja fel:

– Mentés (Save) – lementi a dokumentumot (programot);
– Szimulátor futtatása (Run Simulator) – futtatja az EV3 szimulátort;
– Letöltés (Download) – lementi, letölti a programot bináris, futtatható változatban (*.UF2);
– Dokumentum formázása (Format Document) – szépen, automatikusan elrendezi a kódot;
– Kivágás (Cut) – vágólapra helyezi és kitörli a kijelölt szöveget;
– Másolás (Copy) – vágólapra helyezi a kijelölt szöveget;
– Parancs paletta (Command Palette) – könnyen kereshető változatban, egy listában megjeleníti a parancsokat.



177. ábra. *A szimulátor*

Érdekes a szimulátor működése is (177. ábra).

A LEGO MINDSTORMS Education EV3 szimulátor azonnali visszajelzést ad a programozó számára, hogy milyen érzékelők és motorok vannak csatlakoztatva a tégla melyik portjához. Ez egy nagyszerű módszer a programok tesztelésére és a hibakeresésre is.

Megváltoztathatjuk az érzékelők bemeneti értékeit, nyomogathatjuk a gombokat, megnézhetjük a motorok viselkedését, és láthatjuk a kijelzőn megjelenő információkat.

Választhatjuk a teljes képernyős, vagy a nem teljes képernyős működését, a tégla kijelzőjét fel is nagyíthatjuk működés közben.

3.3.5.3. Programozás MakeCode segítségével

A MakeCode felület a LEGO tégla programozását eseményorientáltan, eseményvezérléssel oldja meg.

Az eseményvezérelt programozás lényege, hogy a teljes program vagy ennek részei, ágai nem szekvenciálisan, előre meghatározott sorrendben futnak le, hanem a vezérlést bizonyos külső vagy belső események határozzák meg, indítják el.

A program így nem más, mint az események bekövetkezésére válaszul végrehajtott eljárások (úgynevezett *eseménykezelők*) halmaza, amelyek nagyrészt egymástól függetlenül dolgoznak.

Az eseményvezérelt architektúrák ez utóbbi okból kifolyólag rendkívül robusztusak és könnyen átláthatók.

A *külső események* a felhasználói bevitel (gomb, billentyű lenyomása, egérművelet, menü kiválasztása stb.), valamint a hardvertől származó események (például az időzítő áramkör vagy periféria által kiváltott megszakítás, visszahívás).

A *belső események* a program más részeitől vagy más programoktól kapott üzenetek.

Az események mindig egy komponenshez kapcsolódnak, ahhoz, amelyen ezek bekövetkeznek. Például ha az egérrel egy gombra kattintunk, akkor az esemény a gomb komponensen keletkezik. Ez egy objektum, és ezt nevezzük az *esemény forrásobjektumá*nak.

Programozás során természetesen nem minden eseményre kell reagálnunk, csak azokra, amelyek a program szempontjából elengedhetetlenek. Így a programozó mondja meg, hogy melyek azok az események, amelyek bekövetkezésére egy forrásobjektumon a programnak reagálni kell, vagyis a programozó készítse el az egyes eseményekhez az eseménykezelőket.

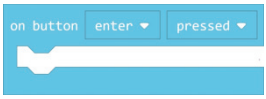
Az eseményvezérelt programok esetén a főprogram egy ciklusban figyeli a bekövetkezett eseményeket (szekvenciálisan), begyűjti, majd szétosztja (dispatch) ezeket. Így hívódnak meg az eseménykezelők.

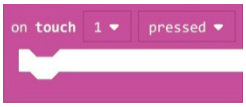
A működés során maguk az eseménykezelők is válhatnak további események kiváltójává és forrásává.

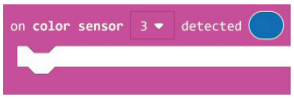
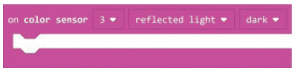
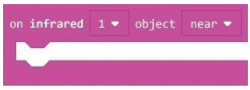
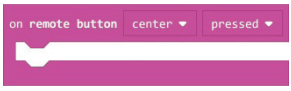
A MakeCode eseménykezelőit a 28. táblázat foglalja össze. A portokat itt is ugyanúgy be kell állítanunk, mint LEGO MINDSTORMS EV3 Home Edition esetén. Érdekesség, hogy a portok JavaScriptben nem paraméterként jelennek meg, hanem a függvények nevében (például `touch1`, `color3` stb.).

28. táblázat. *A MakeCode eseménykezelői*

Eszköz	Blokk	JavaScript	Jelentés

Tégla		`brick.buttonEnter.onEvent(ButtonEvent.Pressed, function () {`
		
		`})`
			Ha lenyomjuk, felengedjük a tégla valamelyik gombját.

Érzékelők		`sensors.touch1.onEvent(ButtonEvent.Pressed, function () {`
		
		`})`
			Ha az érintésérzékelőt lenyomjuk, felengedjük.

Eszköz	Blokk	JavaScript	Jelentés
		`sensors.color3.onColorDetected(ColorSensorColor.Blue, function () {})`	Ha a színérzékelő valamilyen színt érzékel.
		`sensors.color3.onLightDetected(LightIntensityMode.Reflected, Light.Dark, function () {})`	Ha a színérzékelő valamilyen fényerősséget érzékel.
		`sensors.ultrasonic4.onEvent(UltrasonicSensorEvent.ObjectDetected, function () {})`	Ha az ultrahangérzékelő valamilyen tárgyat érzékel.
		`sensors.infrared1.onEvent(InfraredSensorEvent.ObjectNear, function () {})`	Ha az infravörös érzékelő valamilyen tárgyat érzékel.
		`sensors.remoteButtonCenter.onEvent(ButtonEvent.Pressed, function () {})`	Ha lenyomtuk, felengedtük a távirányító valamelyik gombját.
Ciklusok		`forever(function () {})`	Mindig végrehajtódik, folyamatosan fut.
		Nincs külön függvénye, maga a program.	A program kezdetekor hajtódik végre. Inicializáló rész.

Eszköz	Blokk	JavaScript	Jelentés

Kontroll		`control.onEvent(0, 0, function () {`
		
		`})`
			Amikor egy regisztrált esemény történik.

Először tehát kiválasztjuk a programunkhoz szükséges eseménykezelőket, ezeket felhelyezzük a Munkaterületre, majd beletesszük a szükséges utasításokat. Ha Blokk nézetben vagyunk, akkor egyszerűen az egérrel belehúzzuk, ha JavaScript nézetben vagyunk, vagy beírjuk, vagy behúzzuk az egérrel. Ekkor vigyázzunk a zárójelekre!

Ha egy blokkot a Munkaterületre húzunk, és lenyomjuk rajta a jobb egérgombot, a megjelenő menüből lehetőségünk nyílik másolatot készíteni a blokkról (Duplicate), megjegyzéssel ellátni a blokkot (Add Comment), törölni a blokkot (Delete Block), valamint segítséget kérni a blokkról (Help).

178. ábra. *Blokkok megjegyzéssel való ellátása*

JavaScriptben a következőképpen láthatjuk el megjegyzésekkel a forráskódot:

```
// Megjegyzés
```

vagy:

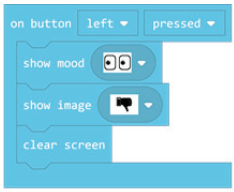
```
/**
 * Többsoros
 * megjegyzés
 */
```

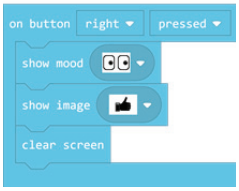
10. feladat

Készítsünk egy olyan programot, amelyik figyeli a tégla gombjait. Ha megnyomjuk a bal gombot, akkor rajzoljon ki egy balra néző szempárt, majd egy „nem tetszik” jelt, ha viszont a jobb gombot nyomjuk meg, akkor egy jobbra néző szempárt, majd egy „tetszik” jelt rajzoljon ki! Mindkét kirajzolás után törölje a kijelzőt!

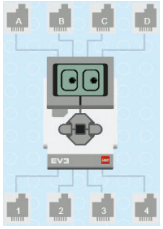
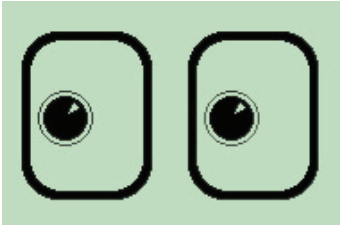
29. táblázat. *Program MakeCode-ban*

A program blokkokkal	A program JavaScriptben

	`brick.buttonLeft.onEvent(ButtonEvent.Pressed, function () {`
	`  brick.showMood(moods.middleLeft)`
	`  brick.showImage(images.informationThumbsDown)`
	`  brick.clearScreen()`
	`})`

	`brick.buttonRight.onEvent(ButtonEvent.Pressed, function () {`
	`  brick.showMood(moods.middleRight)`
	`  brick.showImage(images.informationThumbsUp)`
	`  brick.clearScreen()`
	`})`

30. táblázat. *Futtatás MakeCode-ban*

Futtatás szimulátoron	Futtatás a LEGO EV3 téglán
	

A teljesség igénye nélkül, mivel a blokkok nagyon hasonlítanak a már letárgyalt LEGO MINDSTORMS EV3 Home Edition blokkokhoz, itt csak egy pár érdekességet mutatunk be.

A 31. táblázatban látható kódrészlettel le tudjuk kérdezni a LEGO tégla elemeinek állapotát, töltődöttségét, áramerősségét, feszültségét.

31. táblázat. *Elemek állapota*

Blokk	JavaScript
	`brick.showNumber(brick.batteryInfo(` `BatteryProperty.Level), 1)`

Egyszerre, egy utasítással ki tudjuk íratni a portok állapotát (`brick.showPorts()`).

Nem tudunk betűméretet megadni.

Motorok esetén az azonnali leállást vagy a tehetetlenségből továbbforgást két külön utasítás segítségével tudjuk megadni. Az első a `run`, a második a `ramp`.

Az összes motort le tudjuk állítani egyszerre a `stopAll` utasítással, és olyan tulajdonságokat is be tudunk állítani, amelyeket a LEGO MINDSTORMS EV3 Home Editionban nem. Ilyenek például a motorok gyorsulása, lassulása.

Számos lehetőség van hangok lejátszására.

Ciklusok esetében lehetőség van while, repeat és kétfajta for ciklus létrehozására. A while esetében feltételt lehet megadni. A repeat blokkot a JavaScript for ciklusra fordítja (32. táblázat).

A matematika blokkok között két érdekességre is bukkanhatunk, az egyik a `constrain` (`Math.constrain(0, 0, 0)`), a másik a `map` (`Math.map(0, 0, 1023, 0, 4)`) blokk.

Az első blokk esetében egy megadott szám értéke nem lehet kisebb és nem lehet nagyobb, mint a megadott két másik szám. Az egyik határérték beállítja a legalacsonyabb értéket, amelyet a tesztelt szám felvehet. A másik határérték pedig a legmagasabb értéket, amelyet a tesztelt szám felvehet. Ha a tesztelt érték e két határérték tartományán belül van, akkor maga az érték kerül visszatérítésre, ha viszont az érték kívül esik a két határérték tartományán, akkor a visszatérített érték az a határérték lesz, amelyik a legközelebb áll a vizsgált értékhez. Például ha a 15-öt lekorlátozzuk a 2–10 tartományra, akkor a visszatérési érték a 10 lesz. A `Math.constrain(3, 6, 11)` visszatérési értéke 6 lesz. Természetesen az 5-ös érték, amelyet a 2–10 tartományra korlátozunk, 5-ös is marad.

A `constrain` szintaxisa: `function constrain(value: number, low: number, high: number): number;`

A `map` blokk átkonvertálja az adott tartományból vett értéket egy másik adott tartományban lévő értékre. Például ha egy kutya 17 évig él, egy ember pedig 85 évig, akkor egy kutyaév hány emberévnek felel meg? Ehhez és hasonló leképezésekre jó használni a `map` blokkot.

A `map` szintaxisa: `function map(value: number, fromLow: number, fromHigh: number, toLow: number, toHigh: number): number;`

Tömbök esetén érdekes az, hogy utasítás szintjén tudunk megfordítani egy listát: `list.reverse()`.

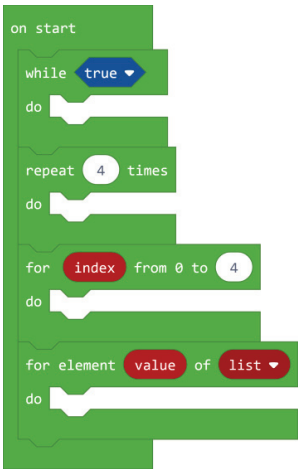
A kontrollblokkok között találjuk a következőket:

- `panic`: megjelenít egy hibakódot és leállítja a programot (`function panic(code: number);`);
- `assert`: megjelenít egy hibakódot és leállítja a programot, ha a feltételként megadott állítás (`cond`) hamis (`function assert(cond: boolean, code: number);`);

– `run in paralel`: a főprogram futtatásával egyidejűleg futtatja a blokkban megadott kódot (`function runInParallel(a: () => void): void;`);
– `raise event`: Kivált egy eseményt, bejelenti, hogy valami történt egy eseményforrásnál (`function raiseEvent(src: int32, value: int32): void;`).

32. táblázat. *Ciklusblokkok és JavaScript-fordításuk*

Blokk	JavaScript

	`let list: number[] = []`
	`while (true) {`
	`}`
	`for (let i = 0; i < 4; i++) {`
	`}`
	`for (let index = 0; index <= 4; index++) {`
	`}`
	`for (let value of list) {`
	`}`

11. feladat

Az ultrahangos érzékelőt felhasználva készítsünk egy olyan programot, amely meg tudja mérni egy, a tégla felé közeledő tárgy sebességét!

A feladat megoldása érdekében csatlakoztassuk az érintésérzékelőt a tégla 1-es portjára, az ultrahangos érzékelőt pedig a LEGO EV3-as tégla 2-es portjára.

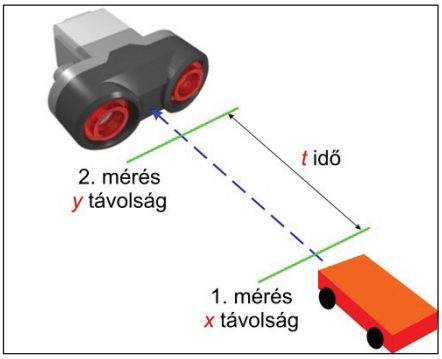
A sebesség mérésének érdekében az ultrahangos érzékelőt tartsuk a közeledő tárgy felé úgy, hogy szemben legyen a tárggyal, és merőleges legyen a mozgásirányra, amint a 179. ábra is mutatja.

Az érintésérzékelő megnyomása után (így indul a program) két mérést végzünk t időegység alatt. Tegyük fel, hogy az első méréskor a közeledő tárgy x távolságra volt, a második méréskor pedig y távolságra. Ekkor a tárgy sebessége a megtett út osztva a beállított idővel, vagyis

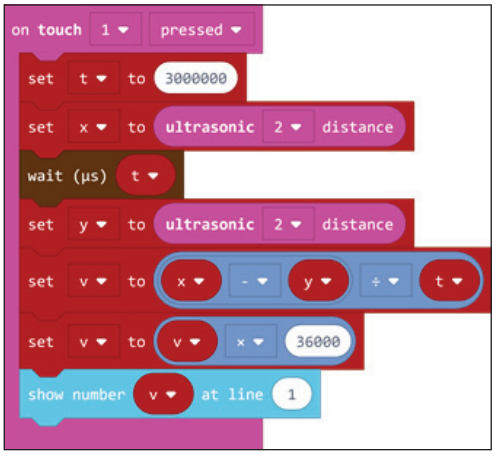
$$v = \frac{x - y}{t}.$$

Mivel a távolságot centiméterben, az időt pedig mikroszekundumokban kell megadni, ezért a sebességet centiméter/mikroszekundumban kapjuk meg.

Ha ezt át szeretnénk alakítani km/órává, akkor a távolságot meg kell szoroznunk 0,00001-gyel, és figyelembe kell vennünk, hogy 1 óra = 3 600 000 000 mikroszekundum, vagyis 1 centiméter/mikroszekundum = 36 000 km/h.



179. ábra. *Sebességmérés ultrahangos érzékelővel*



180. ábra. *A megoldás blokkokkal*

A megoldás JavaScriptben:

```
let v = 0
let y = 0
let x = 0
let t = 0

sensors.touch1.onEvent(ButtonEvent.Pressed, function () {
```

```
    t = 3000000
    x = sensors.ultrasonic2.distance()
    control.waitMicros(t)
    y = sensors.ultrasonic2.distance()
    v = (x - y) / t
    v = v * 36000
    brick.showNumber(v, 1)
})
```

3.3.6. A ROBOTC környezet

A ROBOTC egy C alapú programozási nyelv, könnyen kezelhető fejlesztőkörnyezettel, elsődlegesen a robotika programozási nyelve, amely oktatási célokat tűzött ki önmaga számára. A nyelv ideális robot közti platformok programozására, robotikai rendszerek megvalósítására, így a robotversenyek közkedvelt szereplőjévé vált.

A nyelv az azonosítók, literálok és megjegyzések területén teljes egyezést mutat a C-vel, míg a kulcsszavakban van némi bővítés. A nyelv folyamatos változás alatt áll, időről időre új hardverekhez készül támogatás, illetve bővül a nyelvi készlet is [21].

A különböző LEGO robotokra (RCX, NXT, EV3) a ROBOTC nyelvnek 3 különböző változata létezik, és ezek különböző beépített alprogramokkal rendelkeznek, sőt külön változat van a VEX Robotics, Arduino, CORTEX, PIC rendszerek számára is.

3.3.6.1. Telepítés

A ROBOTC nyelvet és környezetet a *http://www.robotc.net/ download/lego/* honlapról lehet letölteni.

Az egyedüli probléma a ROBOTC környezettel az, hogy csak 10 napos ingyenesen használható próbaverziója van, azután meg kell vásárolni.

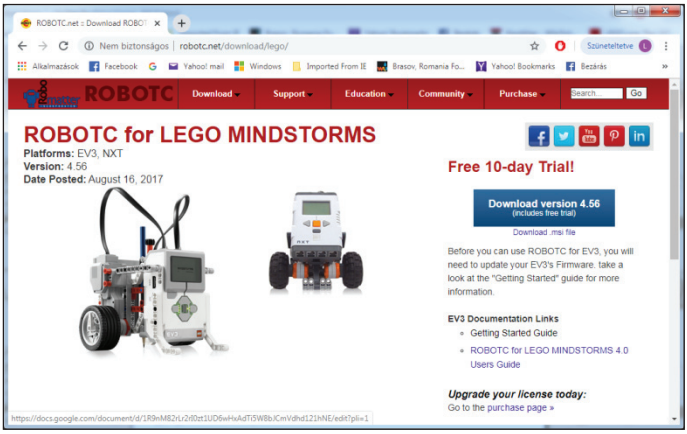
Ha letöltjük a kívánt verziót, például a ROBOTCforLEGOMind-storms_456Release.exe állományt, akkor telepíthetjük ezt.

A szokásos telepítési lépéseket kell végigkövetni, a telepítő a teljes (Complete) módot ajánlja fel, és a szükséges drivereket is telepíti.

A sikeres telepítés után az Asztalon több ikon is megjelenik:

– ROBOTC for LEGO Mindstorms 4.X – szöveges módú programozást lehetővé tevő környezet;
– Graphical ROBOTC for LEGO Mindstorms 4.X – grafikus módú programozást lehetővé tevő környezet;
– Robot Virtual Worlds - LEGO 4.X – szöveges módú programozást lehetővé tevő környezet, az eredményt virtuális roboton lehet tesztelni;

– Graphical Robot Virtual Worlds - LEGO 4.X – grafikus módú programozást lehetővé tevő környezet, az eredményt virtuális roboton lehet tesztelni.



171. ábra. *A ROBOTC honlapja*

Az elindítás után a környezet automatikusan frissíti önmagát, mindenből letölti és telepíti az utolsó verziókat.

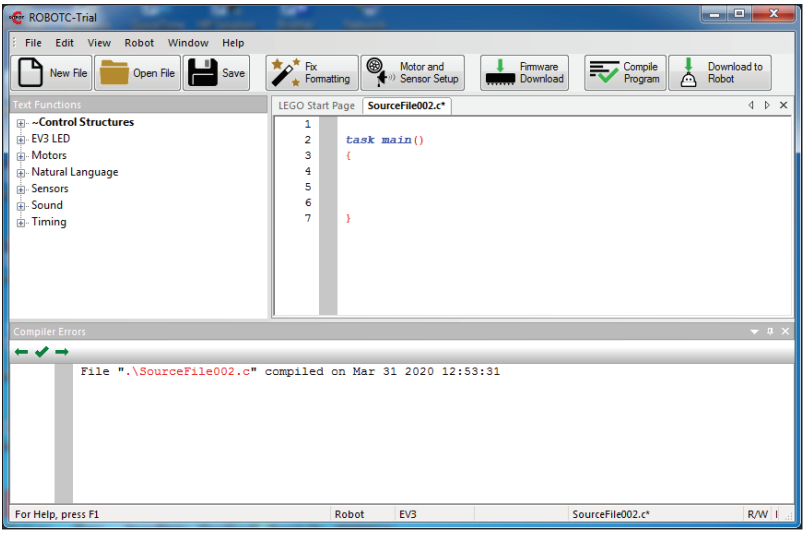
172. ábra. *A ROBOTC környezete*

3.3.6.2. A ROBOTC használata

ROBOTC esetében dönthetünk arról, hogy grafikusan, blokkok segítségével vagy szöveges üzemmódban, C nyelven szeretnénk programozni. A ROBOTC felülete mindkét esetben hasonló, csak az értelem szerinti részek (grafikus vagy szöveges) változnak. A kétfajta felületet a 173., illetve a 174. ábrán mutatjuk be.

Mindkét esetben ugyanaz a menüsor: File, Edit, View, Robot, Window, Help.

A File menüpont a forráskódok, programok megnyitását, mentését, nyomtatását, valamint az alkalmazásból való kilépést tartalmazza. Érdekessége, hogy léteznek Open and Compile, Open and Compile All Files in Directory, Open and Compile All Selected Files / Folders menüsorok, parancsok, amelyek segítségével megnyithatunk és lefordíthatunk egy forrásállományt, az összes forrásállományt egy könyvtárból vagy az összes kiválasztott forrásállományt.



173. ábra. *A ROBOTC szöveges felülete*

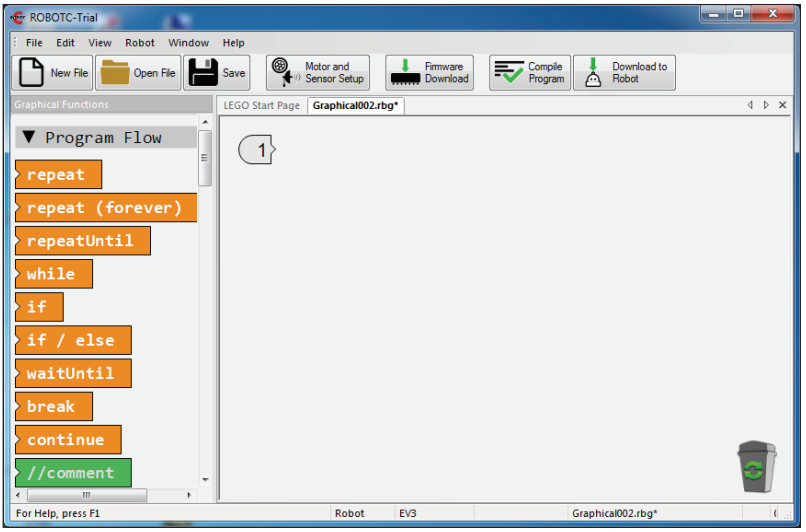
Az Edit menü a szokásos szerkesztési műveleteket tartalmazza, külön kitérve a forráskód formázására és könyvjelzők használatára.

A View menü az alkalmazás kinézetét, a látható elemeket határozza meg. A felület beállításain túl a forrásszövegre vonatkozó látványelemekre is kitér: megtekinthetjük a teljes forráskódot, minden beállítással, az assembly kódot, a Windows Registry infókat, a szimbólumtáblákat, a portokat stb. Ugyanitt lehetőségünk van a betűméret növelésére, csökkentésére, beállíthatjuk az automata kódkiegészítést, de külön menüpontként tartalmazza a felület és a rendszer teljes beállításait is.

A Robot menü a forráskód fordítására, futtatására, nyomkövetésére, a téglán elvégezhető műveletekre, a motorok és érzékelők beállításaira, a firmware és a Linux kernel letöltésére, a robottal való kommunikálás tesztelésére tartalmaz menüpontokat.

A Window menü a menük beállításait, a joystick beállításait, valamint egy hangokat forráskóddá alakító eszközt tartalmaz.

A Help menü a ROBOTC súgórendszerét, a licenszek és frissítések kezelőjét, valamint a névjegyet tartalmazza. Az alkalmazásokhoz külön súgórendszer tartozik, ez a MadCap Help Viewer.



174. ábra. *A ROBOTC grafikus felülete*

A menüsor alatt az eszköztár található, amely gyors elérésű gombokat tartalmaz. Ezek segítségével új forrásállományt hozhatunk létre vagy nyithatunk meg, lementhetjük az állományt, formázhatjuk a kódot, elérhetjük a motor- és szenzorbeállításokat, letölthetjük a firmware-t, lefordíthatjuk a programot, és letölthetjük ezt a robotra.

Az eszköztár alatti felület két részre oszlik, a jobb oldali a munkaterület, ahol a kódot írhatjuk, vagy a blokkokat rakhatjuk össze. A bal oldalon csoportosítva találhatjuk a szöveges üzemmódban használható eljárásokat, utasításokat, grafikus üzemmódban pedig a blokkokat.

Szöveges üzemmódban a felület alján található ablakrészben jelennek meg a fordítási hibák.

Szöveges üzemmódban a következő függvények, eljárások, utak érhetők el:

– vezérlési szerkezetek (Control Structures),
– elem és áramellátás (Battery & Power Control),
– gombok (Buttons),
– naplózás (Datalog),
– nyomkövetés (Debug),
– kijelző (Display),
– meghajtás (Drive Train),

– EV3 LED,
– EV3 különfélék (EV3 Misc),
– állományelérés (File Access),
– postaládák (Mailboxes),
– matematika (Math),
– különfélék (Miscellaneous),
– motorok (Motors),
– érzékelők (Sensors),
– hangok (Sounds),
– párhuzamosság (Task Control),
– időzítés (Timing),
– felhasználói (User Defined).

Például a vezérlési szerkezetek közé sorolhatjuk a taszkokat és alprogramokat, a változókat, az elágazásokat, a különböző ciklusokat. Az EV3 LED az intelligens tégla LED-jeit, gombjait kezeli, a Motorok csoport a LEGO motorokat vezérli. Az Érzékelő csoport az érzékelőkhöz szükséges változókat és magukat az érzékelőket kezelő eljárásokat tartalmazza. A Hangok csoport az audiokezelést, az Időzítés csoport az időzítőkkel kapcsolatos feladatokat látja el.

Az egyes csoportokban változók és parancsok is találhatók.

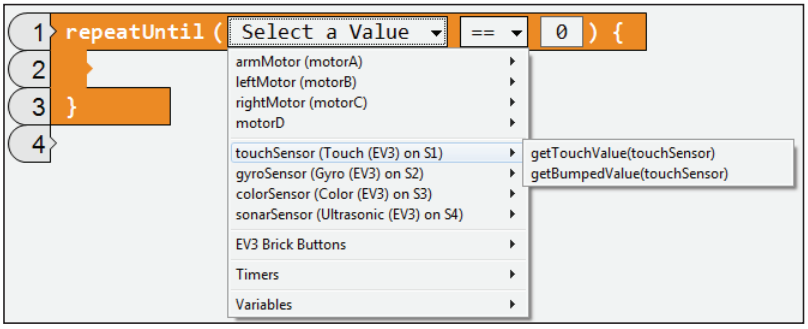
Szöveges módban is működik a „fogd és vidd” (drag and drop) technika. Az eljárásokat megfoghatjuk, és ráhúzva a munkaterületre, megjelennek a megfelelő sorban.

Grafikus üzemmódban a következő nagy csoportok vannak:
– vezérlés (Program Flow),
– változók (Variables),
– egyszerű viselkedés (Simple Behaviors),
– motorvezérlés (Motor Commands),
– távirányító (Remote Control),
– időzítés (Timing),
– vonalkövetés (Line Tracking),
– naplózás (Datalog),
– kijelző (Display),
– EV3 LED,
– giroszkópikus érzékelő (Gyro Sensor),
– érintésérzékelő (Touch Sensor),
– hangok (Sounds).

A blokkok használatának érdekessége, hogy a munkaterületen a rendszer sorszámokkal látja el a ráhúzott blokkokat.

A programozás során rengeteg legördülő lista segíti az értékek beírását, az adatok, feltételek megadását stb.

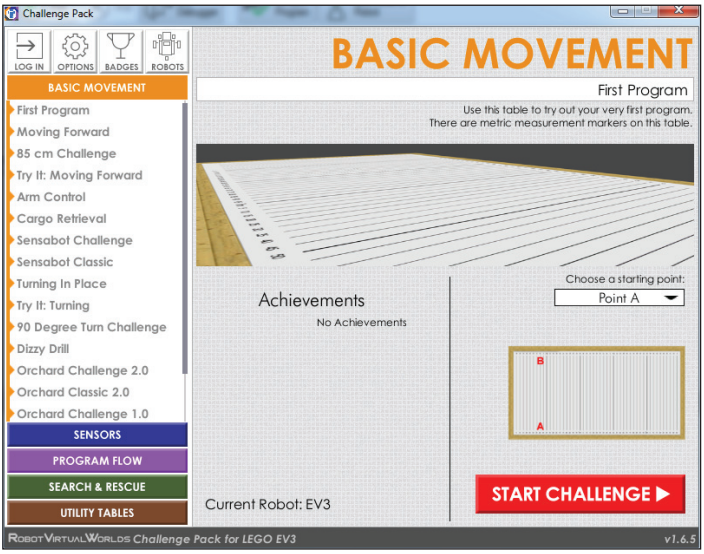
A Help menü a ROBOTC súgórendszerét, a licenszek és frissítések kezelőjét, valamint a névjegyet tartalmazza. Az alkalmazásokhoz külön súgórendszer tartozik, ez a MadCap Help Viewer.



175. ábra. *Legördülő listák*

Hogyha nincs EV3-as robotunk, virtuális környezetben próbálhatjuk ki programunk működését. A Robot Virtual Worlds egy csúcskategóriás szimulációs környezet, amely lehetővé teszi a robotok nélküli programozást.

A Robot Virtual Worlds szimulálja a népszerű valós VEX®, LEGO® és TETRIX® robotokat 3D-s környezetben. A szimulációs környezet tökéletes otthoni, tantermi és versenyek alatti használatra is.



176. ábra. *Virtuális környezet*

A virtuális környezet mind a szöveges, mind a grafikus üzemmódban megírt programok futtatását lehetővé teszi, sőt saját tutoriálokkal, kihívásokkal is rendelkezik, számos izgalmas lehetőséget biztosít az oktatásra.

A ROBOTC használatakor a programozók válthatnak a fizikai és a virtuális robotok között, hogy például gyors hibakeresést végezzenek, és teszteljék a kódjukat virtuális környezetben, majd egy valódi robotra telepítsék azt.

A virtuális világok robotjai tökéletesen szimulálják a motorokat és az érzékelőket.

A szimulátor mobil eszközökre is elérhető.

176. ábra. *Robot működésének szimulálása*

3.3.6.3. Programozás ROBOTC segítségével

A ROBOTC felület a LEGO tégla programozását procedurálisan, imperatívan oldja meg. A ROBOTC firmware-je tartalmazza a C standard library egy részhalmazát, minden C kulcsszó itt is kulcsszó, a C és a ROBOTC tulajdonságai is nagyjából megegyeznek, ezért itt mi csak az eltéréseket tárgyaljuk.

Elsősorban elő kell készítenünk az intelligens EV3 téglánkat.

A Robot menüpontból válasszuk a Download EV3 Linux kernel menüpontot. Itt töltsük le a Standard File-t, majd várjuk meg, míg telepíti a téglára, és ez újraindul.

Ezután szintén a Robot menü Download Firmware menüpontjával töltsük le, és telepítsük a Standard File-t.

Firmware-cserére van tehát szükség, hogy robotunk jól működjön.

Lehetőség van párhuzamos programozásra, a párhuzamos alapegység a taszk, így egy ROBOTC főprogram:

```
task main()
{

  return;
}
```

A nyelv a statikus párhuzamosságot támogatja. Ez abból áll, hogy csak a forráskód szintjén hozhatunk létre folyamatokat, a folyamatok száma fordítási időben meghatározható.

A taszkokhoz teljes hozzáférésünk van, nincsenek korlátozások. Bármelyik taszk leállíthatja vagy szüneteltetheti a többi taszkot.

Műveletek taszkokkal:

- `startTask(taskname)` – elindítja az adott nevű taszkot;
- `stopTask(taskname)` – leállítja az adott nevű taszkot;
- `stopAllTasks()` – leállítja az összes taszkot, a main() kivételével;
- `hogCPU()` – felfüggeszti az összes taszkot a hívó taszk kivételével, így övé lesz a teljes processzor;
- `releaseCPU()` – folytatódik az összes taszk, ami addig függesztve volt.

Beállíthatjuk a taszkok prioritását, de ez nem arányos a kapott időszelettel. Egy prioritásos sor alapján mindig a legnagyobb prioritású taszk kapja meg az ütemezést, ha pedig ezekből több van, a round robin ütemezést használja a futtató rendszer. Az azonos prioritáson lévő taszkok teljesen aszinkron módon futnak, szinkronizálási lehetőségünk csak a `hogCPU()` és `releaseCPU()` parancsokon keresztül van.

A ROBOTC programok változói egy globális térben vannak, tehát párhuzamos programozás esetén a programozó feladata ügyelni arra, hogy helyesen történjenek az értékadások, változóhasználatok.

```
int i;

task task1()
{
 i = 2;
 return;
}
```

```
task main()
{
  startTask(task1);
  i = 1;
  moveMotorTarget(i, 10, 50);
  return;
}
```

A fenti példaprogram esetén a főprogram (main) és a task1 nevű eljárás párhuzamosan hajtódnak végre. Mind a két taszk módosítja az `i` globális változót. Mivel a párhuzamos végrehajtás nemdeterminisztikus, nem tudjuk, hogy melyik motor fog elindulni, illetve többszöri végrehajtás után nem biztos, hogy a motorok elindulási sorrendje ugyanaz lesz. Kerüljük az ilyen helyzeteket!

A ROBOTC nyelvben nincsenek mutatótípusok, és a referenciatípus is csak korlátozottan jelenik meg. A nyelvben nincs dinamikus memóriafoglalás, nincs heap.

A motorokhoz és ezek beállításaihoz tömbökön keresztül férhetünk hozzá. Legfontosabb ilyen tömb a `motor` tömb. A `motor` tömb indexei a különböző motorok azonosítói (például `motorA`, `motorB` stb.), az egyes elemek értékei (–100 és 100 között) a megfelelő motor aktuális sebességét jelenti.

Az egyes motorok pozíciói is elérhetők, vagyis hogy mennyit fordultak a legutóbbi inicializálás óta. A fordulást fokban mérjük, és az `nMotorEncoder` tömb megfelelő indexű elemei tartalmazzák ezeket. Az `nMotorEncoderTarget` tömbbel megadhatjuk azt az fordulatszámot, amelyet ha elér egy motor, akkor automatikusan leáll. E két tömb segítségével precíz mozgások programozhatók le.

Lehetőségünk van motorokat szinkron módon összekötni, s így ha elindítjuk az egyik motort, automatikusan vele együtt mozog a másik. A szinkronizáció beállítása az `nSyncedMotors` változó értékének megadásával történik (például `nSyncedMotors = syncBC`;).

A robothoz kötött szenzorokat is tömbök segítségével tudjuk elérni. A szenzorokra az `S1`, `S2`, `S3`, `S4` konstansokkal tudunk hivatkozni.

A ROBOTC nem támogatja a rekurziót, és nem támogatja az absztrakt adattípusokat vagy a kivételkezelést sem.

12. feladat

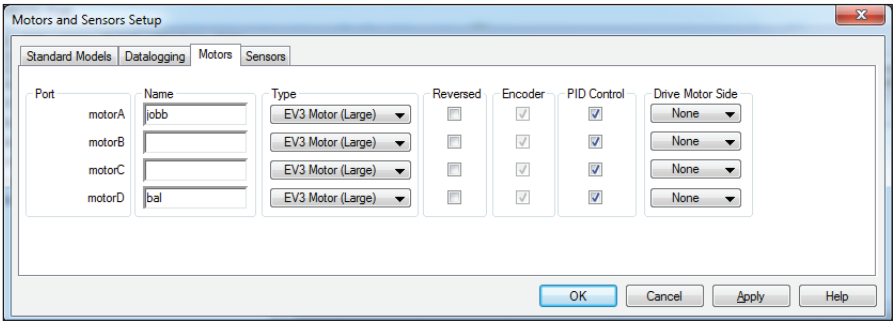
Egy színérzékelővel ellátott robot menjen előre addig, ameddig egy fekete sávot nem érzékel. Menés közben a robot LED-jei pirosan villogjanak, megállás után váltsanak zöldre!

A feladat megoldása érdekében csatlakoztassuk a színérzékelőt a tégla 1-es portjára, és a robotunkat lássuk el két motorral.

ROBOTC-ben lehetőségünk van arra, hogy a motoroknak saját nevet adjunk, így nem a `motorA`, `motorB` konstansokkal hivatkozhatunk rájuk, hanem az általunk adott nevekkel.

Legyen az A portra kötött motor neve `jobb`, a D portra kötött motor neve pedig `bal`.

Az átnevezés úgy történik, hogy a Robot menü Motors and Sensors Setup menüsorát választjuk ki, s az innen előjövő párbeszédablak Motors fülében beállítjuk a motor A és a motor D nevét, a 177. ábra szerint.



177. ábra. *Motorok nevének beállítása*

Ekkor a forráskódban automatikusan legenerálódnak a következő sorok:

```
#pragma config(Motor, motorA, jobb, tmotorEV3_Large, PIDControl,
 encoder)
#pragma config(Motor, motorD, bal, tmotorEV3_Large, PIDControl,
 encoder)
//*!!Code automatically generated by ‘ROBOTC’ configuration
 wizard
```

A Sensors fülben az érzékelőknek is nevet adhatunk, a motorok neveinek megváltoztatásához hasonló módon.

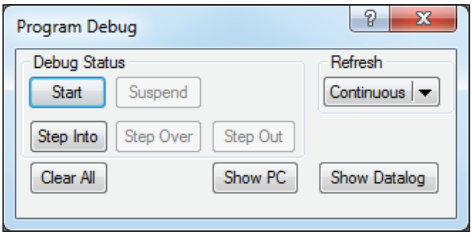
A program a következő:

```
task main()
{
  setLEDColor(ledRedFlash);
  motor[bal] = 50;
  motor[jobb] = 50;
  waitUntil(getColorName(S1) == colorBlack);
  motor[motorA] = 0;
  motor[motorD] = 0;
```

```
    setLEDColor(ledGreen);
}
```

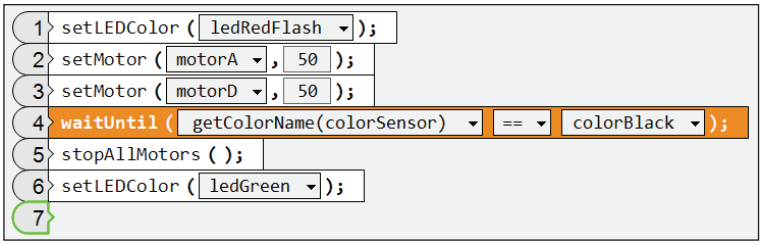
Fordítás előtt a programot le kell menteni, így szöveges üzemmódban egy *.C szöveges állományt, grafikus üzemmódban pedig egy *.RBG bináris állományt kapunk eredményül.

Ha lefordítjuk a programot (Compile program gyorsgomb az Eszköztáron), majd letöltjük a robotra (Download to Robot gyorsgomb), akkor a 178. ábrán látható futtató, nyomkövető párbeszédablak jön elő. Innen működtethetjük ténylegesen a robotunkat (Start).



178. ábra. *Futtató ablak*

Ha a feladatot vizuálisan szeretnénk megoldani, akkor a 179. ábrán látható kódot kapjuk.



179. ábra. *A feladat vizuális megoldása*

3.3.7. A Scratch 3.0

A Scratch egy ingyenesen letölthető vizuális, objektumorientált, interpretált és dinamikus programozási környezet, amelyet elsősorban a gyermekeknek hoztak létre, azzal a céllal, hogy vonzóvá, érdekessé, elérhetővé tegyék

számukra programozást. A Scratch szereplőközpontú, dinamikus, támogatja a játékok, animációk készítését. A különféle médiaelemek – kép és hang – vegyes használatával teret kap a programozó képzelőereje, és olyan interaktív programok születnek, ahol megmutatkozik a kreativitás.

A Scratch nyelvet és környezetet Mitchel Resnick tervezte, és a Lifelong Kindergarten csoport fejleszti az MIT Media Lab keretein belül. A számunkra fontos 3.0-ás verzió 2019. január 2-án jelent meg. Ez a verzió tartalmazza az EV3-as tégla vezérlési lehetőségeit.

A Scratch programkód tulajdonképpen a szereplők viselkedését írja le. A blokkokat az egér segítségével húzhatjuk át az eszköztárból a munkafelületre. A blokkok parancsok, változók, állapotkomponensek, logikai kifejezések, elágazás- és ciklusszervező utasítások, és csak szintaktikailag helyes módon illeszkednek egymáshoz. A programozás eseményvezérelt és többszálú.

Mivel a blokkok egyértelműek, használatuk nyilvánvaló, itt csak az EV3-mal kapcsolatos blokkokat mutatjuk be.

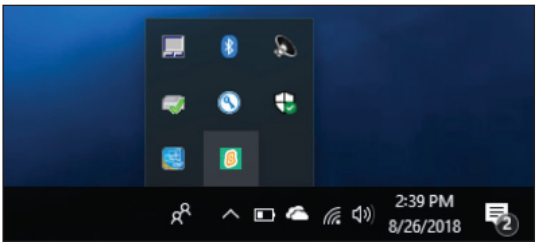
3.3.7.1. Telepítés

Töltsük le és telepítsük a 3.0-ás Scratch-et (Scratch Desktop). A letöltéshez menjünk a *https://scratch.mit.edu/download* honlapra! Megjegyezhető, hogy a Scratch online is használható webes, sőt okostelefonra szóló verziója is van.

Indítsuk el a Scratch Desktop alkalmazást!

Ahhoz, hogy a LEGO EV3 téglát elérjük, telepítenünk kell a Scratch Linket is, amelyet a *https://scratch.mit.edu/ev3* honlapról tölthetünk le.

Indítsuk el a Scratch Linket, és győződjünk meg róla, hogy fut! Meg kell jelennie az eszköztárunkon.



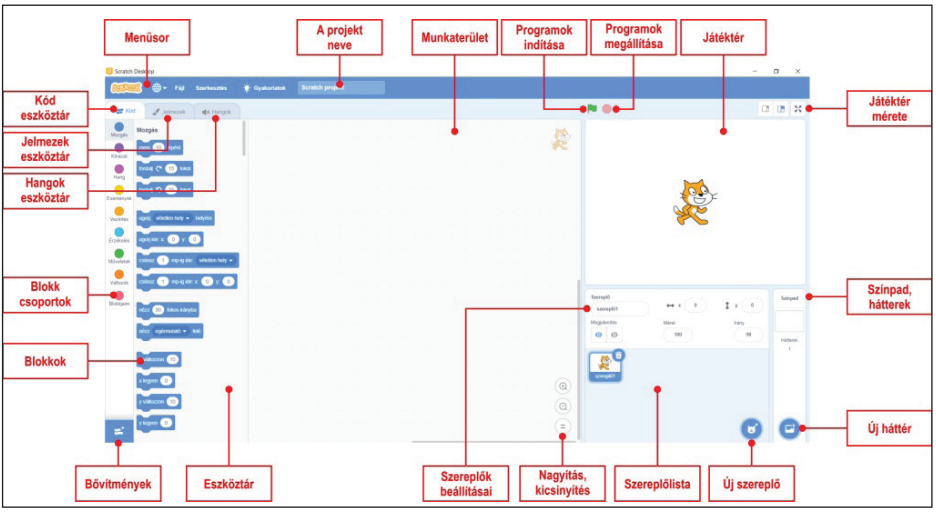
180. ábra. *A telepített Scratch Link*

Kapcsoljuk be az EV3 téglát! A kapcsolat megteremtéséhez az EV3 firmwarejének az 1.10E vagy újabb verziójára van szükség.

Indítsuk el a Scratch Desktopot!

3.3.7.2. A Scratch 3.0 használata

A Scratch felülete (181. ábra) barátságos, áttekinthető, a kiadható parancsok közvetlen elérésűek.



181. ábra. *A Scratch 3.0 felülete*

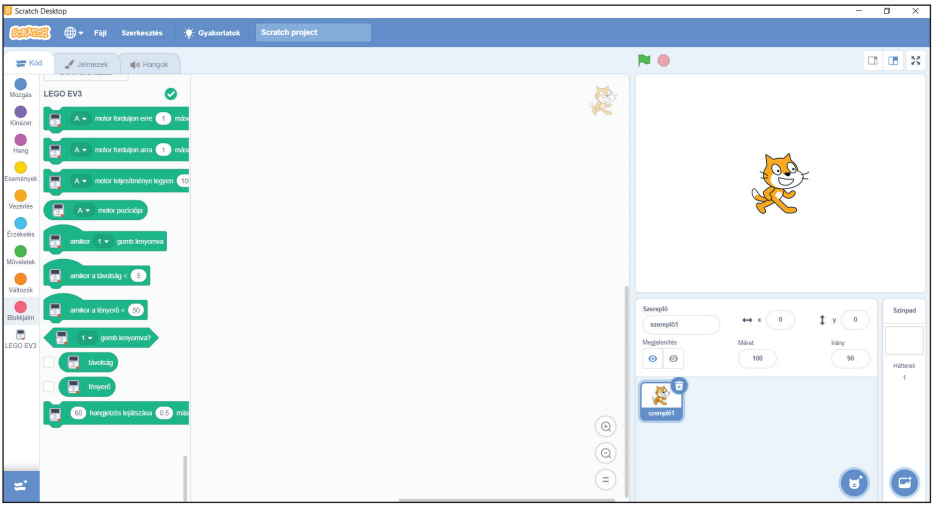
A bal alsó sarokban lévő *Bővítmény hozzáadása* gombbal adjuk hozzá a LEGO MINDSTORMS EV3 bővítményt!

182. ábra. *A micro:bit és a LEGO MINDSTORMS EV3 bővítmény*

Ekkor automatikusan a rendszer csatlakozni próbál a bekapcsolt EV3 téglához.

183. ábra. *Csatlakozás a téglához*

Ha a *Csatlakozás* gombra kattintunk, a Scratch felveszi a kapcsolatot az EV3 téglával. Ha az automatikus csatlakozás nem működik, ellenőrizzük, hogy az EV3 téglán engedélyezve van-e a Bluetooth kapcsolat (lásd IV.3. alfejezet). Ha korábban már csatlakoztunk, és most nem tudunk újra csatlakozni, próbáljuk meg kézzel megszüntetni a társítást az EV3 és a számítógép között. Nyissuk meg a Bluetooth-beállításokat, keressük meg az EV3-at, és távolítsuk el. Az is gond lehet, ha a Scratch több példányban van elindítva. Zárjuk be a többi futó példányt. Csak egy számítógép csatlakozhat egyszerre az EV3-hoz. Ha egy másik számítógép is csatlakozik az EV3-hoz, szüntessük meg azt a kapcsolatot!



184. ábra. *Az EV3 blokkjai*

Ha sikeresen kapcsolódott, nyomjuk meg a *Menj a szerkesztőbe* gombot! Így már a Scratch blokkok mellett (amelyeknek ismertetése itt most nem tisztünk) elérhetők lesznek az EV3 kódblokkok, valamint a LEGO EV3 felirat mellett megjelenik a „minden rendben”-t jelző zöld pipa.

A Scratch 3.0 felületen is, mint ahogy azt már a MakeCode esetében is megszoktuk, a programozás azt jelenti, hogy a működés szerint csoportosított *Eszköztár*ból blokkokat választunk ki, majd azokat a *Munkaterület*en összerakjuk más blokkokkal. Így építjük fel a programot.

A Scratch 3.0 blokkjai mind külalak, mind funkcionalitás szempontjából nagyon hasonlítanak a MakeCode blokkokra, így ezek teljes bemutatását itt mellőzzük, most csak az EV3 blokkjaira koncentrálunk.

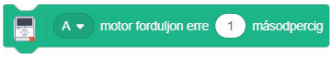
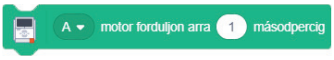
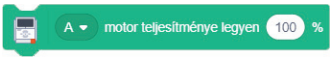
3.3.7.3. Programozás Scratch 3.0 segítségével

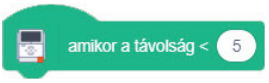
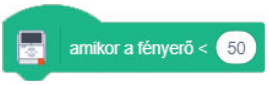
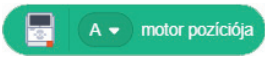
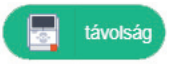
A Scratch felület a LEGO tégla programozását, mint minden más programozási feladatot, eseményorientáltan, eseményvezérléssel – ha kell párhuzamosan – oldja meg.

A Scratch 3.0 EV3 blokkjait a 33. táblázat foglalja össze. A portokat itt is ugyanúgy be kell állítanunk, mint LEGO MINDSTORMS EV3 Home Edition esetén.

A motorokat csak másodperc beállítással irányíthatjuk, nincs fordulatszám vagy szög. Az előre vagy hátra forgást az „erre”, „arra” szavakkal illetik. A távolság és a fényerőn kívül más érzékelőket nem használhatunk.

33. táblázat. *Az EV3 blokkok*

Blokk	Jelentés
	A megadott motor a megadott ideig megy hátra.
	A megadott motor a megadott ideig megy előre.
	Beállíthatjuk a megadott motor teljesítményét / sebességét.
	A megadott másodpercig hangjelzést ad.
	Logikai érték: visszatéríti, hogy a megadott gomb le volt-e nyomva.
	Esemény: akkor hajtódik végre, ha a megadott gomb le van nyomva.

Blokk	Jelentés
	Esemény: akkor hajtódik végre, ha a távolság kisebb a megadott értéknél.
	Esemény: akkor hajtódik végre, ha a fényerő kisebb a megadott értéknél.
	Változó: a megadott motor pozíciója.
	Változó: a fényerő értéke.
	Változó: a távolság értéke.

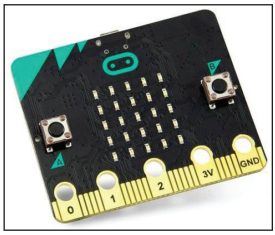
13. feladat

Építsünk meg egy, az 52. ábrán látható robothoz hasonló robotot, amelyet micro:bit segítségével irányítunk. Ha a micro:bitet balra döltjük, a robot forduljon balra, ha a micro:bitet jobbra dőltjük, a robot forduljon jobbra, ha a micro:bitet előre döltjük, a robot haladjon előre, illetve ha a micro:bitet hátra dőltjük, a robot menjen hátrafelé!

A BBC micro:bit egy kifejezetten oktatási célra létrehozott, egylapkás mikrovezérlő, amely 4×5 cm-es méretével, 5×5-ös LED kijelzőjével, gyorsulásérzékelő, hőmérséklet-érzékelő, fényérzékelő, irányérzékelő szenzoraival, be- és kimeneti csatlakozóival, 2 gombjával, bluetooth/rádió-kapcsolódási lehetőségével igen sokrétű alkalmazást tesz lehetővé, legyen az (akár többfelhasználós) játék fejlesztése, viselhető eszközök (pl. okosóra, lépésszámláló, okosruha) tervezése és megvalósítása, kísérletezés a szenzorok által mért adatok felhasználásával, vagy éppen külső eszközök vezérlése/irányítása. Mivel az eszköz egy mikrovezérlő, ezért a programozásához szükséges egy számítógép (asztali, notebook, vagy akár tablet és okostelefon), amelyhez vagy USB-kábellel, vagy Bluetooth-kapcsolaton keresztül kapcsolódhatunk. PC-ről vagy mobilról is elérhető webes felületen (*https://makecode.microbit.org/*) írhatunk programokat, amelyek USB-n vagy akár bluetoothon keresztül tölthetők fel az eszközre. A programot egyszerűen fel kell másolni a micro:bit virtuális meghajtójára, és már működni is kezd.

A Scratch 3.0 kiváló felület micro:bitek programozásához is.

Ha micro:bitet szeretnénk programozni, először kössük össze USB-kábellel a számítógéppel, töltsük le a *https://scratch.mit.edu/microbit* oldalról a Scratch Micro:bit HEX állományt, majd másoljuk rá a micro:bitre. Ezután kihúzhatjuk az USB-kábelt, majd az EV3 robothoz hasonló módon a Scratch Link segítségével csatlakoztathatjuk a micro:bitet a Scratch 3.0-hoz.



185. ábra. *A micro:bit*

Adjuk hozzá a Scratch 3.0-hoz a 182. ábrán látható micro:bit bővítményt, majd az EV3 téglához hasonlóan programozhatjuk a micro:bitet is.

A megvalósított program a 186. ábrán látható.

186. ábra. *A micro:bit és az EV3 robot együttműködése*

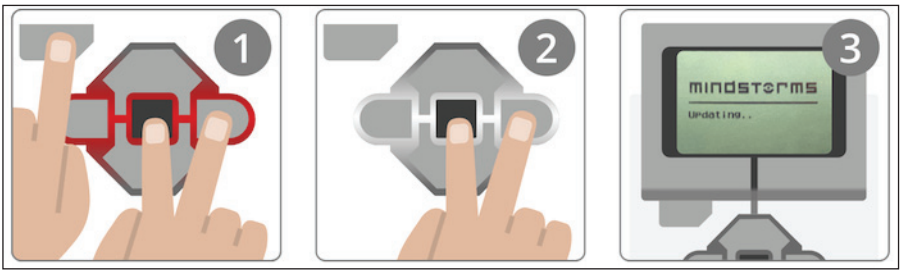
4. HASZNOS TUDNIVALÓK

4.1. Az intelligens tégla resetje

Ha az EV3 tégla hirtelen leáll vagy lefagy, és nem lehet lezárni a bevált kikapcsolási módszer szerint, akkor vissza kell állítani, resetelni kell a processzort. Az EV3 tégla visszaállítása nem törli a memóriában lévő, korábbi munkamenetekből származó fájlokat és projekteket. A meglévő munkamenetből származó fájlok és projektek viszont el fognak veszni.

Elsősorban meg kell győződni arról, hogy az EV3 tégla be van kapcsolva. Ezután a visszaállítás menete a következő:

1. Tartsuk egyszerre lenyomva a Vissza, Közép és Jobb gombot az EV3 téglán.
2. Amikor a képernyő világítása kialszik, engedjük el a Vissza gombot.
3. Amikor a képernyőn megjelenik az indulási kép, engedjük el a Közép és a Jobb gombot.



187. ábra. *Az EV3 tégla visszaállítása*

4.2. Az intelligens tégla firmware-cseréje

A firmware az EV3 tégla gyári szoftvere, operációs rendszere. Firmware nélkül az EV3 tégla nem működik. Időnként a LEGO új firmware-verziót ad ki, amely bővíti a funkciókat vagy hibajavításokat tartalmaz.

A firmware frissítéséhez USB-kapcsolat szükséges a számítógép és az EV3 tégla között, a számítógép pedig internetes kapcsolattal kell hogy rendelkezzen.

A firmware-frissítés előtt győződjünk meg arról, hogy az elemek, akkuk nincsenek lemerülve. Ha a tégla frissítés közben leáll, problémák származhatnak belőle.

A firmware frissítése a tégla memóriájának törlésével jár, tehát a programok, projektek, állományok el fognak veszni!

1. Kapcsoljuk be az EV3 téglát, és csatlakoztassuk USB-kábel segítségével a számítógéphez.
2. Az Eszközök menüben válasszuk ki a firmware-frissítés lehetőséget.
3. Kattintsunk az Ellenőrzés gombra, hogy megtudjuk, elérhető-e új firmware-frissítés.
4. Válasszuk ki a legfrissebb firmware-verziót az *Available firmware* fájlok közül.
5. Ha már letöltött, számítógépen található firmware-verziót szeretnénk használni, nyomjuk meg a Tallózás gombot, majd keressük meg és válasszuk ki a megfelelő firmware-állományt.
6. A Letöltés gombra kattintva elindul a frissítés.
7. A frissítés befejezése után az EV3 tégla magától újraindul.

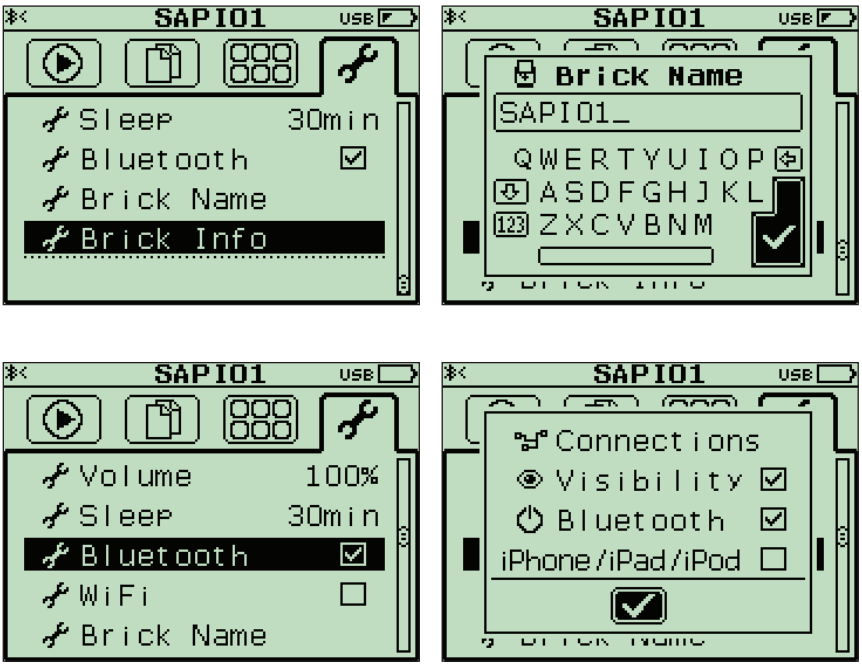
Ha valamilyen oknál fogva az EV3 tégla nem működik a firmware frissítésének folyamat alatt, manuálisan kell frissíteni a firmware-t. A számítógéppel USB-kábelen keresztül csatlakoztatott téglán:

1. Tartsuk lenyomva a Vissza, Közép és Jobb gombokat.
2. Amikor az EV3 tégla újraindul, engedjük fel a Vissza gombot.
3. Amikor a képernyőn a „Frissítés” (Updating) felirat látható, engedjük el a Középső és a Jobb gombot, majd kattintsunk a Letöltés gombra a Firmware frissítő eszközön.

Ha a kézi firmware-frissítés sem teszi az EV3 téglát először működőképessé, ismételjük meg a kézi frissítési folyamatot.

4.3. Bluetooth beállítása

Ha Bluetooth-kapcsolatot szeretnénk létrehozni egy EV3 tégla és egy másik eszköz között, először is adjunk nevet a téglánknak. Ezt megtehetjük a tégla Beállítások (Settings) fülecskéjében.



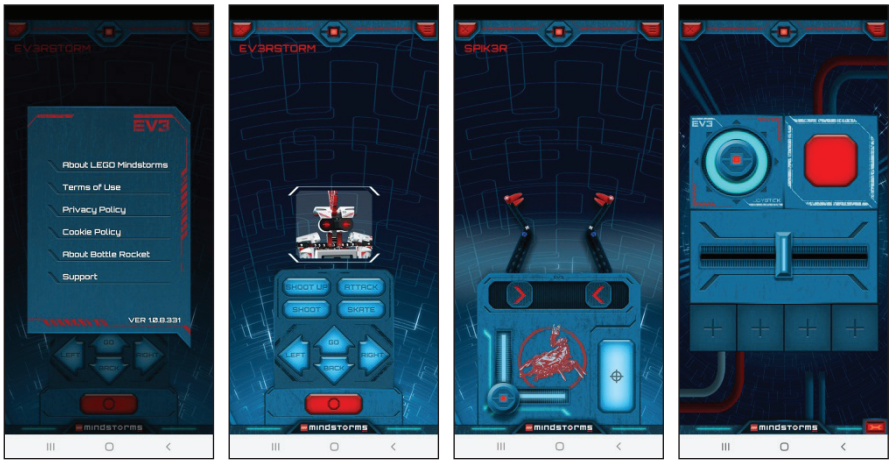
188. ábra. *A Bluetooth beállítása az EV3 téglán*

Kapcsoljuk be a Bluetoothot: EV3 > Settings > Bluetooth > Bluetooth. A láthatóságot (Visibility) és a Bluetoothot is kapcsoljuk be.

Ezután, ha például egy androidos okostelefonról akarunk kapcsolódni a LEGO EV3 téglához, akkor kapcsoljuk be az okostelefonon is a Bluetoothot, a megjelenő listában válasszuk ki az elnevezett téglát (SAPI01), majd végezzük el a párosítást. A telefon is, az EV3-as tégla is felismerik egymást, be kell ütni a megadott kódot (alapértelmezésben ez 1234), majd a két eszköz kapcsolatot tud létesíteni.

A Play Áruházból töltsük le például a LEGO® MINDSTORMS® Commander ingyenes alkalmazást, amellyel irányíthatjuk a LEGO EV3-as robotokat.

Az alkalmazás Bluetoothon keresztül veszi fel a kapcsolatot az EV3 téglával, majd segítségével utasításokat tudunk adni. Lehetőség van a LEGO saját robotjait (EV3RSTORM, R3PTAR, TRACK3R, SPIK3R, GRIPP3R) vezérelni, de önálló kontrollert is készíthetünk a felületen.



189. ábra. *Az EV3 tégla irányítása okostelefonról*

190. ábra. *Program az EV3 téglán*

5. FELADATOK

5.1. Robotos projektek

A közelmúlt, a jelen és a közeljövő egyaránt az informatika, számítástechnika rohamos fejlődéséről tesz tanúbizonyságot. A múlt nagy tudományos-fantasztikus, sci-fi regényei mára már jórészt tudományos ténnyé váltak. Ezt az utat járta be a robotika is. Az asimovi állam hétköznapi valósággá vált, világunk részévé váltak a robotok, mindenütt körülvesznek bennünket.

A külső hardverek, a processzorvezérelt technológiára épülő elektronikus eszközök, az okostelefonok és más mobileszközök, a robotok programozása napjaink nagy kihívása, s a megállíthatatlan fejlődés egyértelmű útmutatóként határozza meg a távolabbi jövőt is: a 21. század a robotokról fog szólni.

Ebben a fejezetben a robotok programozásának gyakorlati ismérveit fogjuk áttekinteni egyszerű robotok segítségével. A robotok megépítése és programozása a mi feladatunk lesz, így a kreativitás a fejezet nélkülözhetetlen kelléke.

A robotok építése és programozása az alábbi szakaszokat követi:

– tervezés,
  - a robot tervezése,
  - a program tervezése,
  - a kommunikáció tervezése,
– építés,
– programozás,
– tesztelés,
  - a robot tesztelése,
  - a program tesztelése,
  - a kommunikáció tesztelése,
– módosítás és továbbfejlesztés.

5.1.1. A tervezés

A feladat megfogalmazása után az első lépés a tervezés. Legelőször a megoldandó feladatot kell pontosan definiálni (specifikáció). Ebből tudjuk meghatározni, hogy mit is kell tudjon a robot. Nyilván egy bonyolultabb robot megtervezéséhez precíz mérnöki tudásra van szükség, hisz figyelembe kell venni a fellépő erőket, az anyagminőséget, a szerkezetet stb. LEGO robotok esetében arra kell ügyelni, hogy minden egyes álló alkatrész legalább két helyen legyen rögzítve, a rögzítések ne essenek szét, a robot legyen stabil, ne ingadozzon, ne dőljön fel hamar. A robusztusság és stabilitás fontos tulajdonságok.

A feladatból derül ki, hogy milyen érzékelőkkel, milyen motorokkal kell ellátni a robotot, hogy nézzen ki, kerekei vagy lánctalpai legyenek, meg kell-e változtassa a haladás irányát, hogyan nézzen ki az adatátadás, kábelezés stb.

A feladatok nagy része olyan, hogy több lehetséges megoldásuk is van, az egyik egyszerűbb, a másik bonyolultabb, az egyik olcsóbb, a másik költségesebb, az egyik jobban programozható, a másik nehezebben stb., tehát mindig a megfelelő megoldást kell kiválasztani. Hasonlóképpen láthattuk azt is, hogy számtalan programozási nyelv, felület, eszköz áll rendelkezésünkre a robotok programozására, itt is a számunkra, a feladat számára épp legalkalmasabbat, legkényelmesebbet kell kiválasztani.

Például ha a feladat az, hogy egy vonalkövető robotot kell megépítenünk, akkor a vonal érzékelésére használhatunk például egy kamerát is. Ekkor a kamera által készített képek feldolgozásával és elemzésével határozhatjuk meg a vonal helyét. Nyilván ez a módszer elég bonyolult, ezért inkább egyszerűbb megoldásokban kell gondolkodni. A vonalkövető robotok általában a visszaverődött fény mérésével érzékelik a vonalat. A fekete vonal ugyanis kevesebb fényt ver vissza, a világos pedig többet. Így a színérzékelőt használhatjuk kamera helyett. Meggondolandó kérdés a színérzékelők száma is. Ha egy színérzékelőt használunk, a robotunk nem lesz annyira pontos, tévedhet, két vagy három színérzékelő használatával sokat nyerünk pontosságban.

Hasonlóképpen a feladat meghatározza azt is, hogy milyen programozási környezetben gondolkodjunk, miben a legegyszerűbb megoldani a feladatot. Nyilván ha színérzékelőt használunk, a legegyszerűbb megoldás a robot saját nyelvében keresendő.

5.1.1.1 A robot tervezése

A robotok tervezése magában foglalja a mozgó és álló részek megtervezését, az összekapcsolást, a kinézetet, designt, vagyis a robot teljes szerkezetének a megtervezését.

Ha úgy szeretnénk megtervezni a robotot, hogy közben elkészüljön a megépítési kézikönyve is, használhatjuk a LEGO Digital Designer vagy a BrickLink Studio szoftvert.

A LEGO Digital Designer (LDD) egy ingyenes számítógépes tervezőprogram, amelyet a LEGO Group készített a LEGO Design byME részeként, és amelyben 3D-s környezetben alkothatunk kedvünk szerint. MacOS és Windows operációs rendszerekre telepíthető. A szoftver lehetővé teszi a felhasználóknak, hogy modelleket építsenek virtuális LEGO elemek felhasználásával.

A legújabb, 4.3.12 verziót 2019-ben töltötték fel, és jelenleg az egyetlen elérhető verzió.

A szoftver különböző színű LEGO elemeket és darabokat tartalmaz, amelyek felhasználhatók bármilyen elképzelhető modell felépítésére. Az átfogóbb LDD

Extended mód lehetővé teszi, hogy bármilyen építőelemet tetszőleges színre állítsunk. A tervezőprogram segítségével képernyőképeket lehet készíteni, és automatikusan létrehozhatjuk az építési útmutatót is.

A LEGO Digital Designer volt a fő modellező program a *Lego Movie* franchise létrehozásához.

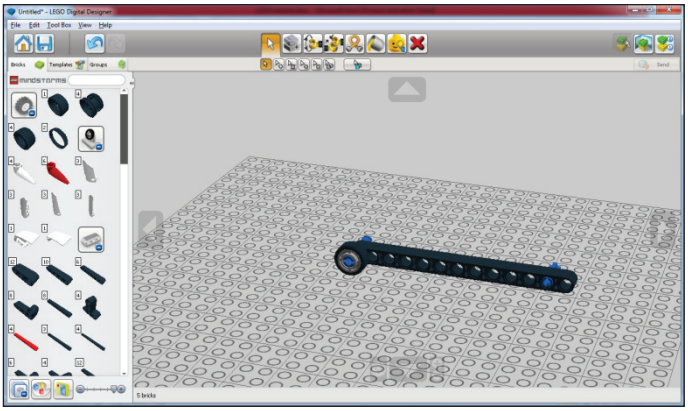
A program a LEGO holnapjáról letölthető (*https://www.lego.com/en-us/ldd*). Ha internethez kapcsolódtunk, és egy korábbi verziót használunk, akkor felkínálja a frissítés lehetőségét, ami gyakorlatilag a teljes telepítőcsomag letöltését és újratelepítést jelent.

A tervezés során a LEGO-alkatrészek egyszerű „fogd és vidd" módszerrel vonszolhatóak át a tervezőfelületre, ahol kurzornyilakkal forgathatók, bal egérgombbal megfogva pedig mozgathatók is. Az alkatrészeket összeilleszthetjük, csoportosíthatjuk.

Az építőelemeket kategóriákra osztották, így viszonylag egyszerűen tudunk válogatni.

A tervezés során felhasznált alkatrészekről generál egy táblázatot, amelyben pontosan látszanak a felhasznált elemek gyári azonosítói, képei, darabszámai is. Ezt a File menü Export BOM pontja teszi.

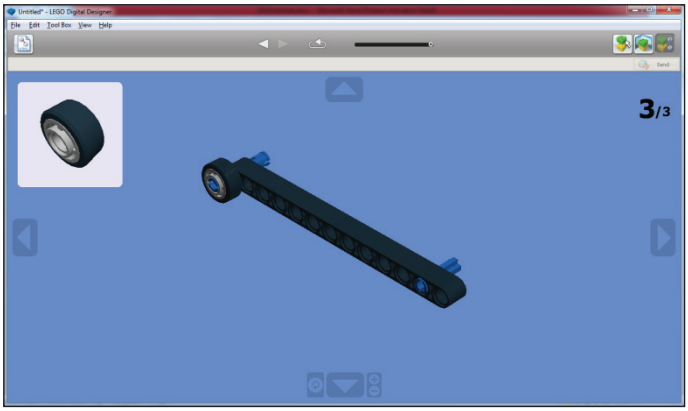
A végeredményt, a tervezett robotot exportálhatjuk 3D modellként (így felhasználhatjuk más szoftverekben), nyomathatjuk képként, vagy generáltathatunk komplett összeszerelési útmutatót is, tetszőleges böngészővel megnyitható HTML formátumban.



191. ábra. *Robot tervezése*

A robot megtervezése csak az első lépése a folyamatnak, számtalanszor megeshet, hogy a program megírása vagy a működés tesztelése során változtatni kell a roboton (például túl magasra tettük a színérzékelőt, beakad a kerék egy

nagyobb kanyarban stb.), ezért jó, ha a robotunkat digitális eszközzel terveztük meg, és így később is könnyen tudunk a terven módosítani, az építési útmutató pedig automatikusan generálódik a módosítások után is.



192. ábra. *Megépítési útmutató generálása*

5.1.1.2. A program tervezése

Mint minden program, alkalmazás, szoftver, a LEGO robotunk programjának a megtervezése is klasszikus szoftvertervezési folyamat, amelynek során a jól bevált és nem egyszer magától a programozási nyelvtől is függő tervezési módszereket kell használni. Ezeknek az ismertetése nem e könyv célja, itt csak egy pár konkrét, fontos elemet szeretnénk kiemelni.

Mint az esetek nagy részében, általában, itt is jó, ha a programunk moduláris felépítésű, saját eljárásokban, blokkokban csoportosítjuk az odaillő, újrafelhasználható részeket. Az általánosság és az absztrakció kedvéért ezeket az eljárásokat, blokkokat paraméterezzük!

Mielőtt hozzákezdenénk a saját blokk elkészítéséhez, érdemes megtervezünk, hogy milyen adatok szükségesek a működéséhez, tehát milyen értékeket kell majd a blokknak átadnunk, illetve milyen visszatérési értékeket szolgáltat majd vissza a működése során a blokk.

Tervezzük tehát meg az adatok forgalmát is.

Mivel általában nem megszokott, de LEGO robotok esetén jól ki lehet használni az osztott és párhuzamos paradigmát, tervezzük meg, hogy hány LEGO EV3 téglára van szükségünk, hogyan kötjük össze ezeket, melyikre milyen szenzorok, motorok kerülnek stb. A kapcsolásról, szerkezetről készítsünk diagramot!

Ha párhuzamos szálakat használunk, akkor alaposan meg kell tervezni, hogy a motorok mozgását éppen melyik szál vezérli. Általában ezt érdemes egy szálra

bízni, és a többi szálat a szenzorok figyelésére vagy egyéb műveletek elvégzésére használni. A párhuzamos végrehajtásról is készítsünk diagramot!

5.1.1.3. A kommunikáció tervezése

Amennyiben a robotunk működés közben kommunikál is a számítógéppel vagy más robottal, hardverrel, meg kell terveznünk a kommunikáció protokollját is.

A protokoll magába foglalhatja a következőket:
– adatátvitel kezdeményezése és befejezése;
– küldők és fogadók szinkronizálása;
– küldési hibák észlelése és kijavítása;
– adatok formázása és kódolása;
– a szolgáltatást, amit a protokoll nyújtani fog;
– üzenetek szótárát, amely a protokollt implementálja;
– az üzenetek kódolását;
– eljárási szabályokat, amelyek a stabil üzenetváltást felügyelik.

A szabályok tervezéséhez állapotdiagramokat, folyamatábrákat, algebrai kifejezéseket vagy program formájú leírásokat használhatunk.

El kell döntenünk, hogy melyik fél kezdeményezi az adatküldést, illetve melyik fél fejezi be azt.

Egy protokoll tervezésekor a következő szabályokat kell figyelembe vennünk:

1. A protokoll legyen egyszerű – a protokoll legyen jól strukturált, legyen felbontható kisszámú, jól érthető részekre.
2. Modularitás – az összetett protokollokat bontsuk fel modulokra úgy, hogy minden rész külön fejleszthető, tesztelhető, kezelhető legyen.
3. A protokoll legyen jól formált – a protokoll ne legyen hiányos, ne tartalmazzon elérhetetlen kódot, ismerje a határait, tudja stabilizálni magát.
4. Robusztusság – a protokollt úgy kell megtervezni, hogy a nem előre látható esetekben is képes legyen helytállni. Ne függjön a környezetétől.
5. Konzisztencia – az állapotátmenetek legyenek világosak, kerüljük az események ismétlődését.

5.1.2. Építés

A robot megépítése a megtervezett modell alapján már elég egyszerű, ám kézügyességet igénylő feladat.

Alakítsunk ki magunk körül egy munkaterületet, a LEGO-alkatrészeket osszuk szét, vigyázzunk, hogy ezek ne vesszenek el. Lehetőség szerint rendezzük be őket egy tálcára.

Ha úgy gondoljuk, hogy a robotunkhoz más eszközöket is használni kell (például egy kamerát a filmezésre), akkor pászítsuk össze ezeket az eszközöket a LEGO-alkatrészekkel valamilyen módon.

Amint megépítettük a robotot, szedjük össze a megmaradt alkatrészeket, tegyük be a dobozba.

5.1.3. Programozás

Amint már említettük, a feladatnak megfelelően válasszuk ki a legkényelmesebb programozási nyelvet. Ha úgy gondoljuk, hogy imperatívan könnyebb (például a rekurzív hívások miatt), akkor imperatív nyelvet válasszunk, ha a vizuális, blokkokra épülő nyelv a megfelelő, akkor arra essen a választásunk.

A programozást az elméleti résznél leírtak alapján végezzük, arra törekedve, hogy a programunk legyen minél egyszerűbb, átláthatóbb, hatékony, moduláris.

A forráskódot lássuk el megjegyzésekkel, magyarázzuk meg és dokumentáljuk az algoritmusokat, függvényeket, eljárásokat, blokkokat.

5.1.4. Tesztelés

A megépítés és a programozás után a robotunkat alaposan le kell tesztelni az alábbiak alapján:

– a robot tesztelése,
– a program tesztelése,
– a kommunikáció tesztelése.

Ha a robot nem úgy viselkedik, ahogy elterveztük, ennek általában nem a robot az oka, hanem valószínűleg valamilyen építési vagy programozási hibát követtünk el.

Mindig vigyázzunk arra, hogy a szélső értékekre is jól teszteljük le a robotot minden szempontból.

Ha rendellenességet észlelünk, azonnal javítsuk ki, majd teszteljük az egészet újra.

Robotunk akkor lesz végleges, ha minden funkcionalitás szerint jól működik.

5.1.5. Módosítás és továbbfejlesztés

Gyakran később is jönnek jó ötletek arra vonatkozóan, hogy a robotot hogyan kellene kibővíteni, milyen plusz funkcionalitással kellene ellátni. Vigyázzunk arra, hogy a robotot úgy építsük meg és úgy programozzuk, hogy könnyen bővíthető, továbbfejleszthető legyen.

Ha megépítettünk és kipróbáltunk egy robotot, akkor ne szedjük szét azonnal, inkább gondolkozzunk, mire tudjuk még felhasználni, hogyan fejlesszük tovább.

5.2. A vonalkövető robot

14. feladat

Építsünk és programozzunk le egy olyan robotot, amely követni tud egy világos (fehér) felületen lévő sötét (fekete) zárt vonalat (görbét)!

5.2.1. A robot

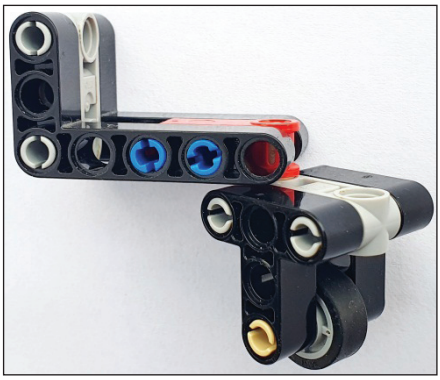
Ahhoz, hogy egyáltalán követni tudja a robot a vonalat, először is érzékelnie kell azt. Szükségünk lesz egy olyan szenzorra, ami érzékeli a vonalat, és olyan válaszjelet ad, amit a robotot vezérlő elektronika értelmezni tud. Erre való a színérzékelő. Az első felmerülő nagy kérdés az, hogy hány színérzékelőt használjunk? Az itt található megoldás során kipróbáljuk a robotot úgy, hogy egy, kettő vagy három színérzékelőt fogunk használni, a megfelelő alfejezetben megmagyarázzuk és letárgyaljuk a különbségeket [45].

A lényeg az, hogy úgy tervezzük meg a robotunkat, hogy rendre fel tudjuk szerelni rá a három színérzékelőt.

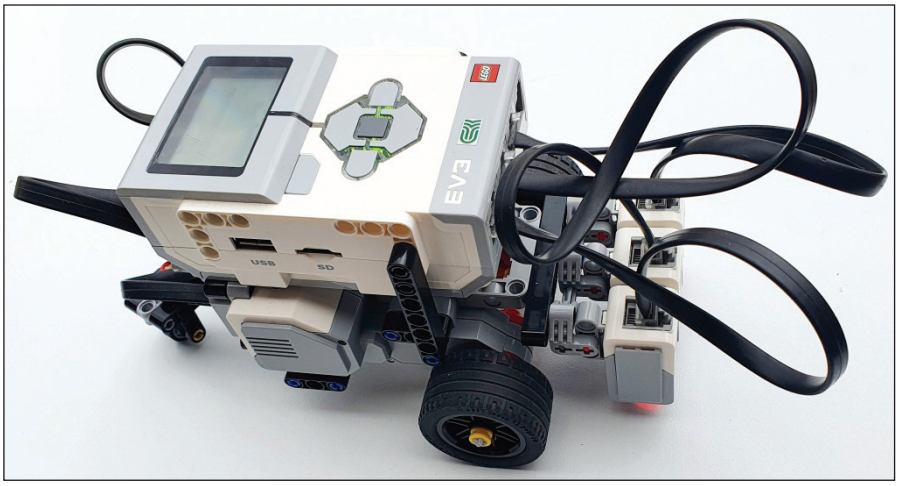
193. ábra. *Bolygókerék az Education készletből*

Szükséges fő alkatrészek: természetesen a legfontosabb a LEGO EV3 intelligens tégla, a színérzékelők, két motor, két nagy kerék, valamint hátul egy bolygókerék, amely megtámasztja a robotot, körbe forogva pedig nem akadályozza a mozgását.

Ha LEGO® MINDSTORMS® Education EV3 készletünk van, akkor ebben már található bolygókerék. Home készlet esetén vagy megvásároljuk a Replacement Pack LME 3 (2000702), vagy a 194. ábra alapján készítünk egy egyszerű bolygókereket. A forgó részeknél (kerék és függőleges tengely) használjunk sárga alkatrészeket (pl. 4514554), mert ezek biztosítják a gördülékenységet.



194. ábra. *Bolygókerék*



195. ábra. *Vonalkövető robotunk*

A vonalkövetésehez a robotnak haladnia kell tudni. Ehhez valamilyen motorra (motorokra) lesz szükség. Mi most egy differenciális meghajtású robotot fogunk építeni.

A differenciális meghajtású robot két különböző oldalán lévő kerekei egymástól függetlenül vannak meghajtva. Ezek biztosítják a meghajtást és egyben a kormányzást is, ezért nem szükséges az első kerekek elfordítása, mint például egy hagyományos autó vagy kerékpár esetén. A két oldalon lévő kereket azonos sebességgel hajtva, a robot az adott irányba egyenesen halad. A robot kormány-

zásához elegendő a jobb és bal oldali kereket különböző sebességgel meghajtani. Például ha a jobb oldali kerék gyorsabban forog, mint a bal oldali kerék, akkor a robot balra fordul. A fordulás ívenék nagysága a kerekek forgási sebességének a különbségétől függ. Ha az egyik motor áll, akkor az álló kerék körül elfordulva kanyarodik a robot. Megvalósítható a helyben fordulás is, ha a motorokat ellentétes irányba forgatjuk.

A megépített robotot a 195. ábra mutatja be. Ezt a robotot később még más feladatok megoldásánál is fel fogjuk használni.

Programozás szempontjából megemlítjük már most, hogy mindhárom program esetében az első lépés a színérzékelő finomhangolása a 4. feladat és 44. ábra alapján.

5.2.2. Vonalkövetés egy színérzékelővel

A feladatot a MakeCode segítségével fogjuk megoldani. Egy színérzékelőt használunk – a középsőt – a robotra felszerelt háromból.

A megoldás alapötlete a visszavert fényerősség mérésén alapszik.

A tesztelés során kialakult fényállapot esetén 7 volt a fekete, 60 a fehér részről visszavert fény erőssége, a feketéhez közel álló fehér részen pedig 50-et mért.

196. ábra. *Fényerősség-mérések*

Ha egy színérzékelőt használunk, a robot nagyjából azon a vonalon fog végigmenni, amely elválasztja a fekete sávot a fehér háttértől. Tulajdonképpen nem is igen számít így a fekete sáv vastagsága.

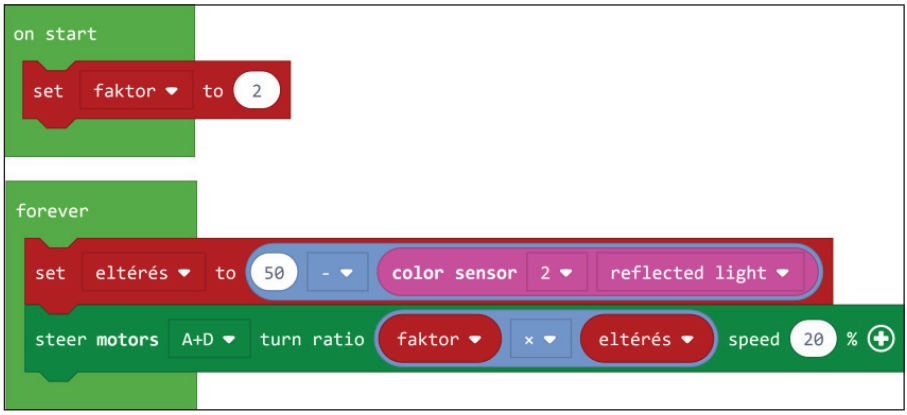
Az alapötlet az, hogy eltérést számítunk a mért fehér/fekete erősség (50) és a visszavert fény erőssége között (*eltérés* = 50 – *visszavert fény erőssége*). Ha a visszavert fény erőssége a feketéhez áll közelebb, akkor ez az eltérés pozitív lesz, ha a fehérhez áll közelebb, akkor az eltérés zéró vagy negatív lesz.

Ezt az eltérést beszorozzuk egy faktorral (legyen ez 2 – empirikus alapon, mert így nem nagy a robot kilengése, eléggé egyenletesen halad), és így megkapjuk azt a szöget, amellyel a robotunkat balra vagy jobbra térítjük el.

Ha a színérzékelő intenzív fehéret érzékel, a robotot balra téríti el, vagyis visszatéríti a fekete sávra.

Ha a színérzékelő feketét érzékel, a robotot jobbra téríti el, vagyis a fehér felé közelíti, így pontosan követni tudja a fehér és fekete közötti vonalat.

A faktorral beszorzott eltérést beállítjuk a két motor kormányzás blokkján (steer motors), s így a bekapcsolt motorok mindig a kért irányba fognak térülni, a robot pedig előre fog haladni.

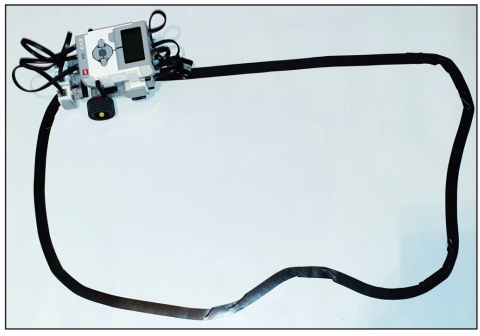


197. ábra. *Vonalkövetés egy színérzékelővel*

Megjegyzendő, hogy a pontosság kedvéért ezeket az értékeket minden fényhelyzetben újra kellene számolni, és ezekkel finomra hangolni, inicializálni a színérzékelőt. Itt a kódnak a rövidebb változatát közöljük.

JavaScriptben a fenti kód a következőképpen néz ki:

```
let eltérés = 0
let faktor = 0
faktor = 2

forever(function () {
    eltérés = 40 - sensors.color2.light(LightIntensityMode.Reflected)
     motors.largeAD.steer(faktor * eltérés, 20)
})
```



198. ábra. *Vonalkövető robotunk működés közben*

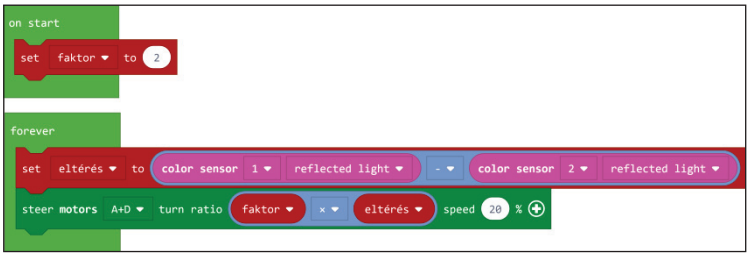
5.2.3. Vonalkövetés két színérzékelővel

Ha két színérzékelőt használunk, akkor a köztük lévő távolság akkora legyen, mint a fekete sáv szélessége. Mindkét színérzékelő a fekete sáv és a fehér rész közötti vonalat fogja figyelni.

Ekkor eltérést számítunk a két színérzékelő által visszaszolgáltatott fényerősségértékek között. Ez az eltérés lesz pozitív vagy negatív, és ez fogja meghatározni, hogy a robot balra vagy jobbra térüljön.

Az eltérés értékét itt is beszorozzuk a faktorral, majd az új értéket beállítjuk a két motor kormányzás blokkján (steer motors), s így a bekapcsolt motorok mindig a kért irányba fognak térülni, a robot pedig előre fog haladni.

Két színérzékelőt használva a robotunk jóval pontosabban követi a fekete vonalat a fehér lapon.



199. ábra. *Vonalkövetés két színérzékelővel*

JavaScriptben a fenti kód a következőképpen néz ki:

```
let eltérés = 0
let faktor = 0
faktor = 2

forever(function () {
     eltérés = sensors.color1.light(LightIntensityMode.Reflected) -
    sensors.color2.light(LightIntensityMode.Reflected)
    motors.largeAD.steer(faktor * eltérés, 20)
})
```

5.2.4. Vonalkövetés három színérzékelővel

A megoldás alapötlete az, hogy az egy színérzékelős vonalkövetőt kombináljuk a két színérzékelős vonalkövetővel, s így egy sokoldalú vonalkövetőt kapunk eredményül.

A harmadik érzékelő tulajdonképpen a „sürgősségi” érzékelő, amely segítségével jóval élesebb kanyarokat is be tud venni a robot.

A programunk első részében tehát a középső színérzékelő programja fut, vagyis a vonalkövetés egy színérzékelővel program.

A program második felében a két szélső színérzékelőt programozzuk le. Mindkettő a fekete vonalat kell hogy keresse, tehát két egymásba ágyazott elágazásban azt nézzük, hogy ha az érzékelt fényerősség kisebb, mint mondjuk 15, mindkét esetben egy ciklusban addig ismétli a balra vagy jobbra fordulásokat egy adott szöggel, amíg az érzékelt fényerősség kisebb, mint 15.

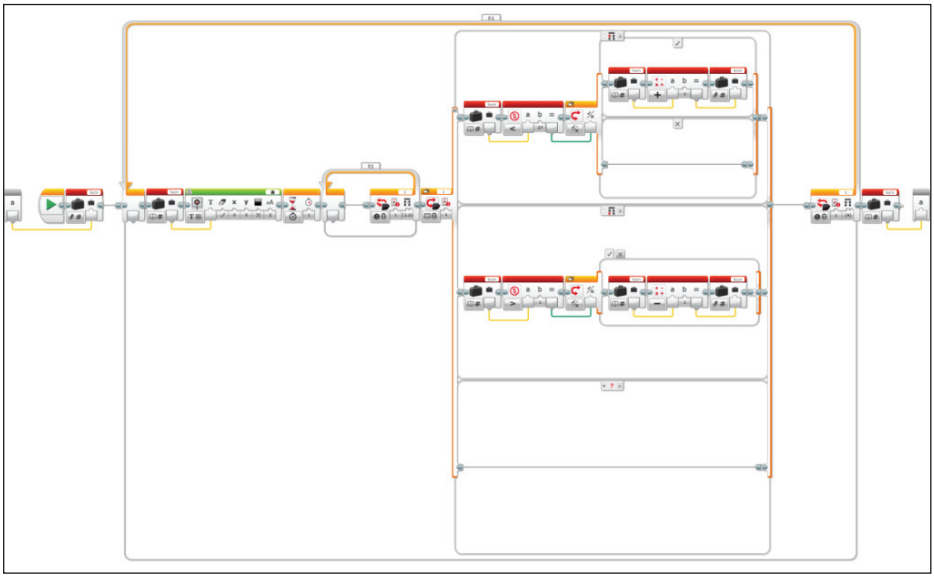
Ha a robot nem reagál elég érzékenyen, a középső színérzékelőt szereljük előbbre, ne legyen egy síkban a másik kettővel.

Ennek a megoldásnak a leprogramozását az olvasóra bízzuk!

5.3. Számbeolvasó

15. feladat

Az infravörös érzékelő és a távirányító segítségével valósítsunk meg egy számbeolvasót!



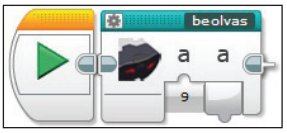
200. ábra. *Számbeolvasó*

A számbeolvasó 1 és 27 között tudjon számokat beolvasni úgy, hogy a képernyőn először jelenjen meg a kezdőérték, majd ha a távirányító felfele gombját

nyomjuk, egyesével nőjön a szám értéke 27-ig, ha a távirányító lefele gombját nyomjuk, egyesével csökkenjen a szám értéke 1-ig. Ha a távirányító középső gombját nyomjuk meg, akkor az éppen látott érték lesz a beolvasott szám. Vigyázzunk, mert a távirányító középső gombját, ha megnyomtuk, benyomva marad, tehát egymás után kétszer kell megnyomni, hogy visszaálljon eredeti állapotába!

Valósítsuk meg a számbeolvasót saját blokk formájában. A kezdőértéket adjuk át paraméterként, majd szintén paraméterként kapjuk meg a beolvasott számot!

A feladatot LEGO MINDSTORMS EV3 Home Editionban fogjuk megoldani.



201. ábra. *A számbeolvasó blokk*

5.4. Kártyatrükk

16. feladat

A kártyatrükkhöz 27 kártyalapra van szükségünk. Gondoljunk egy számra 1 és 27 között. Keverjük meg a paklit, majd hátlappal felfele vegyük ki belőle a gondolt számú kártyalapot, jegyezzük ezt jól meg, tegyük vissza a pakliba, és jól keverjük össze ezt. Ezután előlappal felfelé osszuk le a kártyalapokat három oszlopba. Jegyezzük meg, hogy a kiválasztott kártyalap melyik oszlopba került (bal, jobb vagy középső), ezt tudassuk a robottal. Ezután a robot algoritmusa által diktált módon vegyük fel előlappal felfelé a kártyaoszlopokat (például: középső, bal, jobb vagy bal, középső, jobb), majd előlappal felfelé osszuk le a kártyalapokat három oszlopba. Ismételjük meg az előbbi műveletet, majd még egyszer osszunk, és vegyük fel az oszlopokat (összesen három osztás és három oszlopfelvétel). Fordítsuk meg a kártyapaklit úgy, hogy hátlappal felfele legyenek a kártyalapok, majd számoljunk le annyi kártyalapot, amilyen számra gondoltunk az elején. Az utoljára leszámolt kártyalap a megjegyzett, előre kiválasztott kártyalap kell hogy legyen.

A 2019-ben megszervezett IX. Kiss Elemér Tábor nagy sztárja volt a kártyatrükköt végző robot.

Nyilván a trükk a kártyaoszlopok felemelésének sorrendjében van.

202. ábra. *A kártyák leosztása*
(http://www.mathaware.org/mam/2014/calendar/27card.html)

Például ha a 8-as számra gondoltunk, és az első leosztás után a kiválasztott kártyalap a BAL oldali oszlopban van, akkor a következőképpen kell felvenni az oszlopokat: Középső, BAL, Jobb. A második osztás után a kiválasztott kártyalap szintén a BAL oldali oszlopba kerül, ekkor a felvétel sorrendje: BAL, Jobb, Középső. A harmadik osztás után a kártyalap szintén a BAL oszlopba fog kerülni, és így vesszük fel az oszlopokat: BAL, Középső, Jobb.

Ha a gondolt szám 12, az első osztás után a kártyalap a JOBB oszlopba fog kerülni, a felvétel sorrendje: Bal, Középső, JOBB. A második osztás után a kártya szintén a JOBB oszlopba kerül, a sorrend pedig: Bal, JOBB, Középső. A harmadik osztás után a kártyalap a BAL oszlopba kerül, és az oszlopokat így kell felvenni: Középső, BAL, jobb.

Ha a gondolt szám a 23, a kiválasztott kártyalap az első leosztás után a JOBB oldali oszlopba fog kerülni, a felvétel: Középső, JOBB, Bal. A második leosztás után szintén a JOBB oszlopba kerül, és így kell felvenni: Bal, JOBB, Középső. A harmadik leosztás után a kártyalap a BAL oszlopban lesz, és így kell az oszlopokat felvenni: Középső, Jobb, BAL.

A trükk matematikai háttere a következő:

Egy tetszőleges szám oszlopának a felemelési sorrendjét a három leosztási körben a $(27-a)\%9\%3$, $|2-\lfloor((27-a)\%9)/3\rfloor|$, valamint a $\lfloor(27-a)/9\rfloor$ képletek adják meg.

Például a 8-as szám esetén:

1. körben	2. körben	3. körben
1 – középsőként vesszük fel az oszlopot	2 – elsőként vesszük fel az oszlopot	2 – elsőként vesszük fel az oszlopot

A 12-es szám esetén:

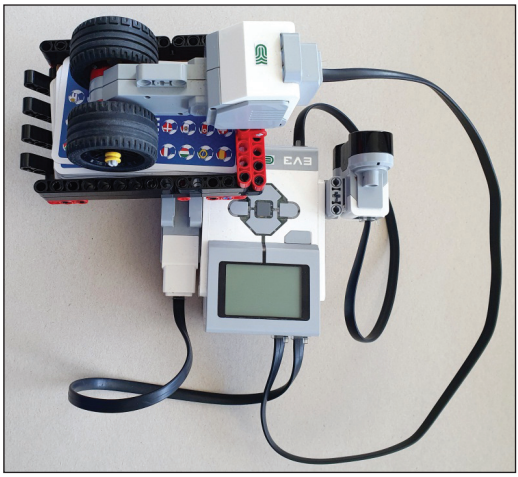
1. körben	2. körben	3. körben
0 – utolsóként vesszük fel az oszlopot	0 – utolsóként vesszük fel az oszlopot	1 – középsőként vesszük fel az oszlopot

A 23-as esetén:

1. körben	2. körben	3. körben
1 – középsőként vesszük fel az oszlopot	1 – középsőként vesszük fel az oszlopot	0 – utolsóként vesszük fel az oszlopot

Igazából nem számít, hogy más oszlopokat milyen sorrendben veszünk fel, de mivel a robot kell mondjon valamit, így azokat véletlenszerűen állítjuk elő.

A feladatban elhangzó felszólításokat a robot felvett hangállományok segítségével mondja el. A játékos a kívánt számokat az V.3. alfejezetben megvalósított számbeolvasó segítségével a távirányítóval adja meg. A 195. ábrán látható módon megépítjük a robotot. A nagy motor felemelhető, lecsukható. A kártyák legyenek vastagabbak, vékony kártyákat nehéz így leosztani.



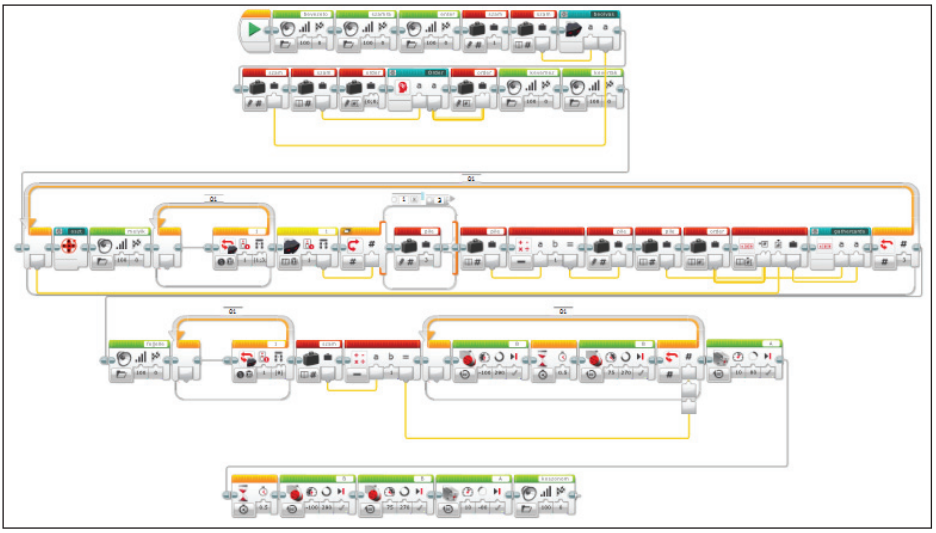
203. ábra. *A trükkös robot*

Hangok:

– bevezeto.rsf: *„Ehhez a trükkhöz 27 kártyára van szükséged. Kövesd a robotot és el fogsz ámulni!”*
– szamra.rsf: *„Gondolj egy számra 1 és 27 között! A távirányító bal gombjaival vezetheted be a számot, figyelj a képernyőre!”*
– enter.rsf: *„A távirányító középső gombja az ENTER.”*
– kevernez.rsf: *„Keverd jól össze a kártyákat, majd nézd meg a gondolt számú pozíción lévő kártyát! Jegyezd meg ezt jól!”*
– keverrak.rsf: *„Keverd ismét jól össze a kártyákat, majd előlappal felfele tedd be a robotba, és nyomd meg a távirányító középső gombját!”*

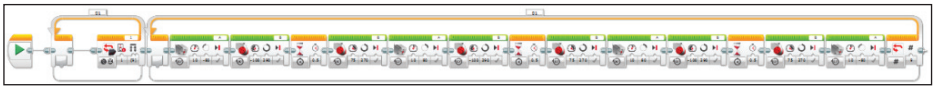
– melyik.rsf: *„Melyik oszlopban volt a kiválasztott kártya? A bal, közép vagy jobb? A távirányító ugyanezen gombjait nyomd meg!”*
– koszonom.rsf: *„Köszönöm, hogy velem játszottál!”*
– jobb.rsf: *„Jobb.”*
– kozepso.rsf: *„Középső.”*
– bal.rsf: *„Bal.”*
– pakli.rsf: *„Vedd fel a következő pakli kártyát:”*
– tetejere.rsf: „És helyezd a kezedben lévő tetejére!”
– oszto.rsf: *„Tedd be a kártyákat az osztóba, majd nyomd meg a középső távirányító gombot!”*
  • fejjelle.rsf: *„Fejjel lefelé tetted be őket? Na úgy tedd be, és nyomd meg a középső gombot!”*

Főprogram:
– Program.ev3p – főprogram, 204. ábra.



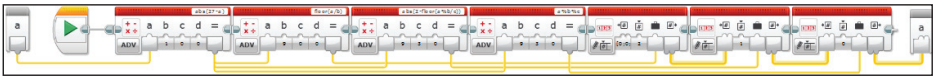
204. ábra. *A kártyatrükk főprogramja*

Saját blokkok:
– beolvas.ev3p – számbeolvasó, lásd V.3. alfejezet;
– oszt.ev3p – így oszt a robot két motor használatával, 205. ábra;
– Order.ev3p – kezdeti rendezés, 206. ábra;
– randomorder.ev3p – véletlenszerű rendezés, 207. ábra;
– gathercards.ev3p – a kártyagyűjtés modulja, 209. ábra;
– pile.ev3p – az oszlopok modulja, 208. ábra.



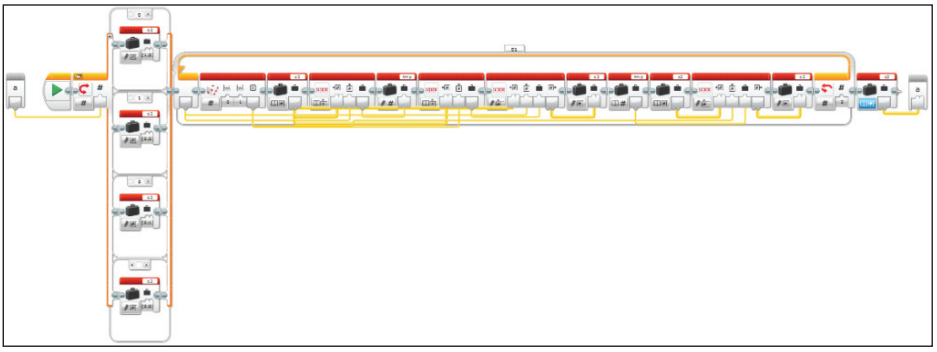
205. ábra. *A kártyaosztás modulja*

A kártyaosztás modulja mozgatja a két motort ahhoz, hogy a robot három oszlopba tudja szétosztani a kártyákat. A nagy motorral a kártyalapokat osztjuk le, a középső motorral a megfelelő szöget állítjuk be, hogy jobbra, középre, balra menjenek a kártyák.



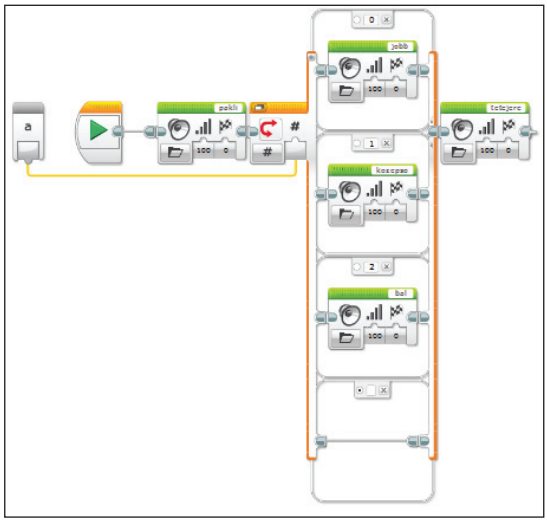
206. ábra. *A rendezés modulja*

A matematikai háttérszámításokat a rendezés modulja végzi, míg az oszlopok tömbjét a véletlen rendezés modulja állítja elő.



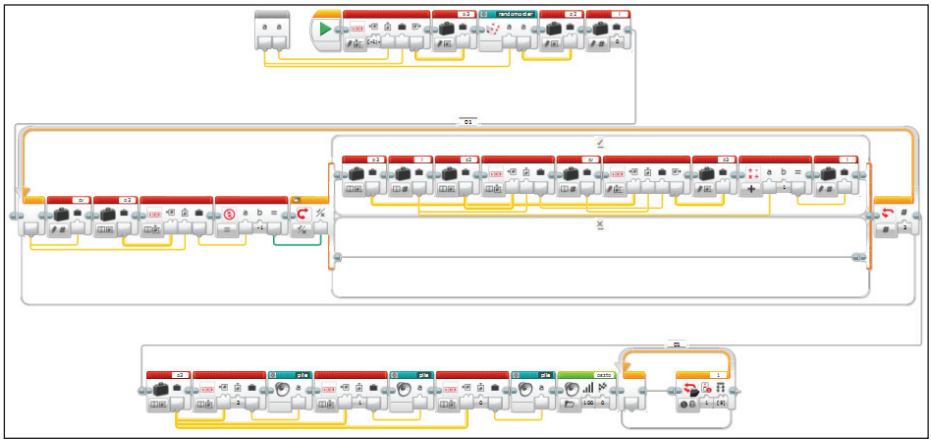
207. ábra. *A véletlen rendezés modulja*

Az oszlopok modulja dekódolja az oszlopok neveit, tehát a 0 a jobb, 1 a középső, 2 pedig a bal oszlop lesz, és a neveket ki is mondja.



208. ábra. *Az oszlopok modulja*

A kártyagyűjtés modulja dönti el a matematikai számításokat elvégző modulok segítségével, hogy milyen sorrendbe vegyük fel az oszlopokat.



209. ábra. *A kártyagyűjtés modulja*

Mivel a blokkok nem teljesen olvashatók a méret miatt, képzeljük el a robot programját úgy, hogy a következő C++ nyelvű programot írtuk át valamilyen, az EV3 robot által felismert nyelvre.

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

string pile_names[3] = {„BAL”, „KÖZÉP”, „JOBB”};

struct order
{
    int r[3];
};

order compute_order(int n)
{
  order o;
  n = 27 - n;
  o.r[2] = n / 9;
  o.r[1] = 2 - n % 9 / 3;
  o.r[0] = n % 9 % 3;
  return o;
}

void wait_for_keypress()
{
  cout << „<ENTER>, ha kész!” << endl;
  cin.ignore();
}

void random_order(int exclude, int o[2])
{
  o[0] = o[1] = exclude;
  while (o[0] == exclude)
    o[0] = rand()%3;
  while ((o[1] == o[0]) || (o[1] == exclude))
    o[1] = rand()%3;
}

void gather_cards(int pile, int ord)
{
  string p[3];
```

```
42    int o[2];
43    random_order(pile, o);
44    p[ord] = pile_names[pile];
45    int j = 0;
46    for (int i = 0; i < 3; ++i)
47      if (i!=ord)
48        p[i] = pile_names[o[j++]];
49    cout << „1. Vedd fel a „ << p[2] << „ oszlopot, előlappal
            felfelé!” << endl;
50    cout << „2. Tedd a „ << p[1] << „ oszlopot rá!” << endl;
51    cout << „3. Végül tedd a „ << p[0] << „ oszlopot is a
            tetejére.” << endl;
52    wait_for_keypress();
53  }
54
55  int main()
56  {
57    srand(time(0));
58    int n = 0;
59    order o;
60    cout << „Ehhez a trükkhöz 27 kártyára van szükséged.” << endl;
61    cout << „Kövesd a robotot és el fogsz ámulni!” << endl;
62    wait_for_keypress();
63    while ((n<1) || (n>27))
64    {

65      cout << „Gondolj egy számra 1 és 27 között: „;
66      cin >> n;
67      cin.ignore();
68    }
69    o = compute_order(n);
70    cout << „Keverd össze a kártyákat!” << endl;
71    wait_for_keypress();
72    cout << „Jegyezd meg a „ << n << „. helyen lévő kártyát!”
          << endl;
73    wait_for_keypress();
74    cout << „Keverd össze újra a kártyákat!” << endl;
75    wait_for_keypress();
76    for (int i = 0; i < 3; ++i)
77    {
78      cout << „Előlappal felfelé oszd szét a kártyákat ciklikusan
              jobbra, középre, balra!” << endl;
```

```
79        wait_for_keypress();
80        int pile = 0;
81        while ((pile<1) || (pile>3))
82        {
83          cout << „Melyik oszlopban van a kártyád?
                  1 - BAL, 2 - KÖZÉPSŐ, 3 - JOBB: „;
84          cin >> pile;
85          cin.ignore();
86        }
87        gather_cards(pile-1, o.r[i]);
88      }
89      cout << „Számolj le a hátlappal felfelé „ << n <<
                „kártyát, a felső a TIÉD!” << endl;
90      wait_for_keypress();
91      cout << „Köszönöm, hogy velem játszottál!” << endl;
92      return 0;
93    }
```

5.5. Sávszámoló

17. feladat

Írj programot, amely megszámlálja, hogy hány piros, sárga, zöld, kék és fekete vonal van egy adott pályán, majd kiírja a képernyőre a következőképpen:

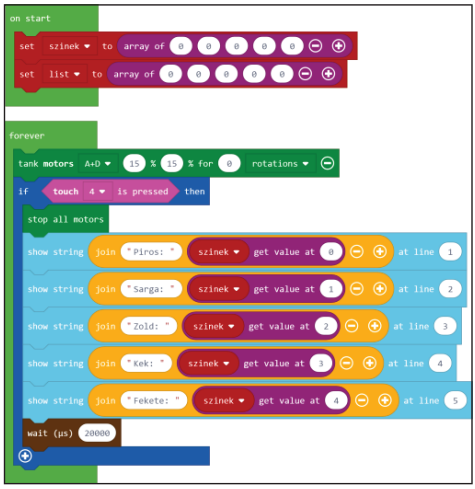
PIROS: … db.
SARGA: … db.
ZOLD: … db.
KEK: … db.
FEKETE: … db.

Az V.2. alfejezetben bemutatott színérzékelős robotot használjuk fel, most természetesen csak egy színérzékelővel.

A robotot ellátjuk egy érintésérzékelővel is, mert azzal fogjuk leállítani.

A robot lassan kell haladjon, hogy érzékelni tudja a színes sávokat.

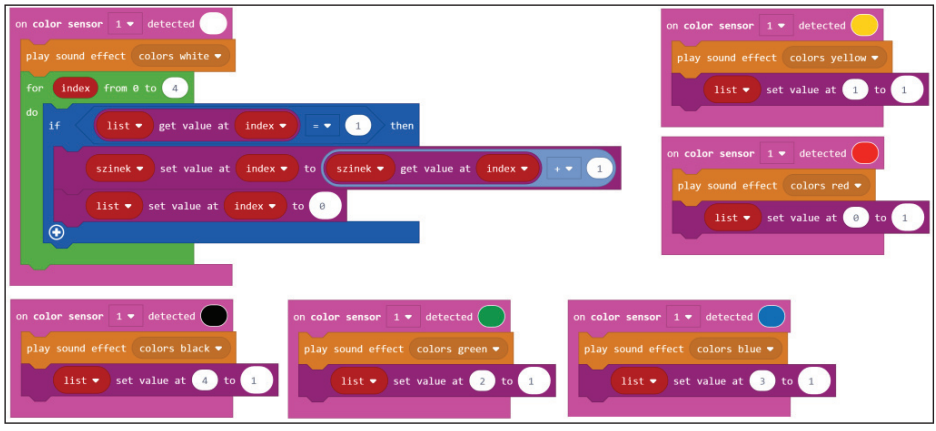
A megoldás alapötlete az, hogy két tömböt fogunk használni. Az egyik azért kell, hogy a program független legyen a sávok szélességétől. Ha egy széles sávon kétszer vagy többször is mér a színérzékelő, azt egy mérésnek vegye. A sávok között fehér csíkok vannak (háttér), így a fehérnél aktualizálja a tömböket. A második tömbben számolja a sávokat. Mindkét tömb ötelemű, mert öt színt kell hogy felismerjen a robot.



210. ábra. *Az on start és forever blokkok*

Az *on start* eseménykezelő nullás kezdőértékekkel látja el a tömböket. A színek tömb a sávokat számolja, a list tömb pedig, hogy milyen színű sávon van a robot.

A *forever* eseménykezelő futtatja a motorokat mindaddig, ameddig le nem nyomjuk az érintésérzékelőt. Ekkor leállítja a motorokat és kiírja a színek számát.



211. ábra. *A színérzékelő eseménykezelői*

A színérzékelő eseménykezelői (a megfelelő színre) kimondják a felismert színt, majd módosítják a sávok tömbjét. Fehér sáv érzékelésekor napra készre hozzuk a tömböket.

JavaScriptben a kód:

```
let list: number[] = []
let szinek: number[] = []

sensors.color1.onColorDetected(ColorSensorColor.Yellow, function
() {
    music.playSoundEffect(sounds.colorsYellow)
    list[1] = 1
})

sensors.color1.onColorDetected(ColorSensorColor.Black, function ()
{
    music.playSoundEffect(sounds.colorsBlack)
    list[4] = 1
})

sensors.color1.onColorDetected(ColorSensorColor.Green, function ()
{
    music.playSoundEffect(sounds.colorsGreen)
    list[2] = 1
})

sensors.color1.onColorDetected(ColorSensorColor.Blue, function ()
{
    music.playSoundEffect(sounds.colorsBlue)
    list[3] = 1
})

sensors.color1.onColorDetected(ColorSensorColor.Red, function () {
    music.playSoundEffect(sounds.colorsRed)
    list[0] = 1
})

sensors.color1.onColorDetected(ColorSensorColor.White, function ()
{
    music.playSoundEffect(sounds.colorsWhite)
    for (let index = 0; index <= 4; index++) {
        if (list[index] == 1) {
```

```
            szinek[index] = szinek[index] + 1
            list[index] = 0
        }
    }
})

szinek = [0, 0, 0, 0, 0]
list = [0, 0, 0, 0, 0]

forever(function () {
    motors.largeAD.tank(15, 15, 0, MoveUnit.Rotations)
    if (sensors.touch4.isPressed()) {
        motors.stopAll()
        brick.showString(„Piros: „ + szinek[0], 1)
        brick.showString(„Sarga: „ + szinek[1], 2)
        brick.showString(„Zold: „ + szinek[2], 3)
        brick.showString(„Kek: „ + szinek[3], 4)
        brick.showString(„Fekete: „ + szinek[4], 5)
        control.waitMicros(20000)
    }
})
```

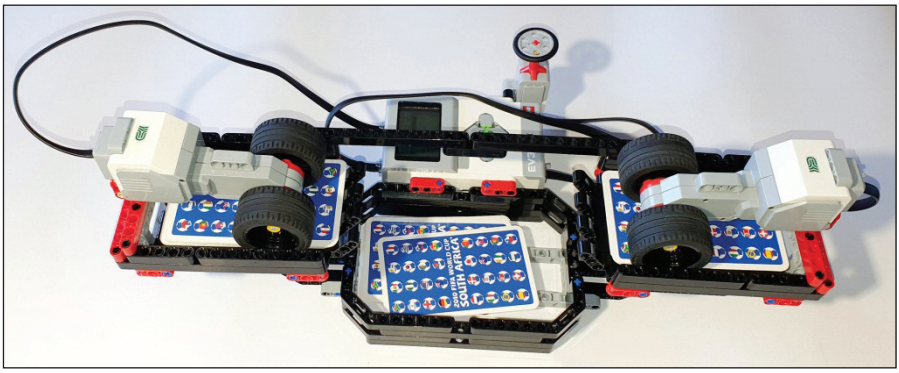
5.6. Kártyakeverő

18. feladat
Építsünk egy kártyakeverő robotot!

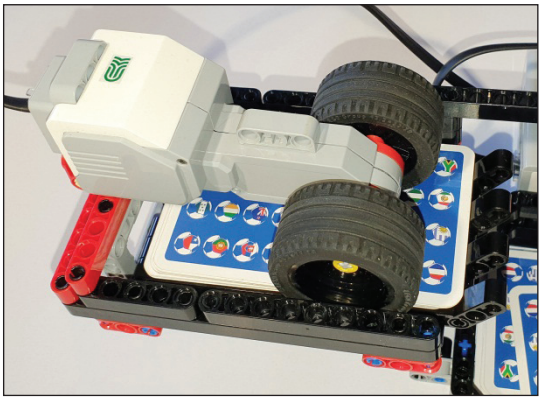
A robot működési elve az, hogy a pakli kártyát kettéválasztjuk két közel egyenlő vastagságú részre, majd mindkét részt külön-külön belehelyezzük a robot megfelelő tárolójába, ahonnan a robot egyenként, felváltva egy harmadik tárolóba osztja le a lapokat, s így összekeveri ezeket (212. ábra).

A robot megépítésénél fel tudjuk használni az V.4. fejezetben bemutatott kártyaosztó robot tárolóját. Még egy ugyanilyent kell készíteni a másik oldalra is (213. ábra).

A kártyakeverő robot programozása egyszerű, csak a motorokat kell meghajtani, viszont azért, hogy egyszerre ne több lap essen le, a motor előre kell hogy lökje a felső lapot, míg az leesik, de a vele együtt elmozduló alsó lapot vissza kell hogy húzza egy kicsit. Ez így volt megoldva az V.4. fejezetben is. Tehát a motrok 330 fokot fordulva lökik a felső kártyát, 270 fokot fordulva pedig visszahúzzák az alsót.



212. ábra. *Kártyakeverő robot*

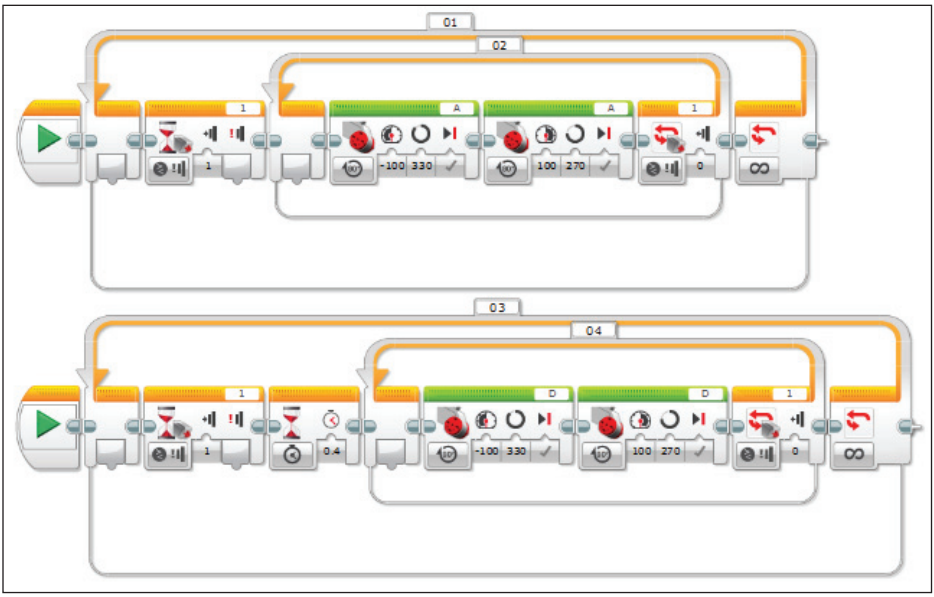


213. ábra. *Kártyatároló a roboton*

A robot úgy van elképzelve, hogy akkor kezdi keverni a kártyákat, amikor benyomtuk az érintésérzékelőt, és addig keveri, ameddig az érintésérzékelő be van nyomva. Ennek érdekében a keverést egy végtelen ciklusba programozzuk (01 és 03 ciklusok), amelyen belül egy másik ciklus addig működik, ameddig az érintésérzékelő be van nyomva (02 és 04 ciklusok).

Azért, hogy a keverés gyorsabban működjön, amiután elindult az egyik motor, rövid időre már indulhat a másik motor is, így a két motor működését egymástól függetlenül, párhuzamosan programozzuk le. Nyilván, hogy ne működjenek egyszerre, a D motort 0,4 másodperccel késleltetjük az A motorhoz viszonyítva.

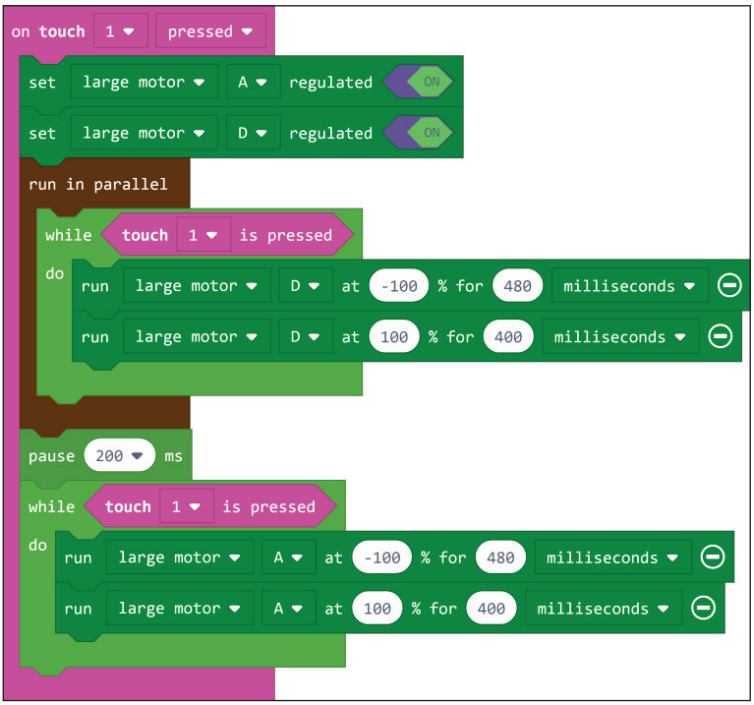
A LEGO MINDSTORMS EV3 Home Edition környezetben összerakott program a 207. ábrán látható.



214. ábra. *A kártyakeverő robot programja*

Ha ugyanezt a programot MakeCode környezetben szeretnénk összerakni, megírni, akkor könnyebb dolgunk van, hisz itt van eseménykezelés, viszont a tapasztalatunk azt mutatta, hogy a motorok forgása nem igazán működött pontosan, ha szögeket adtunk meg, csak akkor, ha milliszekundumokban fejeztük ki, hogy mennyit forogjanak. Itt is párhuzamosan programoztuk le a két motort. A pontosság érdekében a program elején mindkét motort szabályoztuk, regularizáltuk.

A MakeCode blokkok a 208. ábrán láthatók.

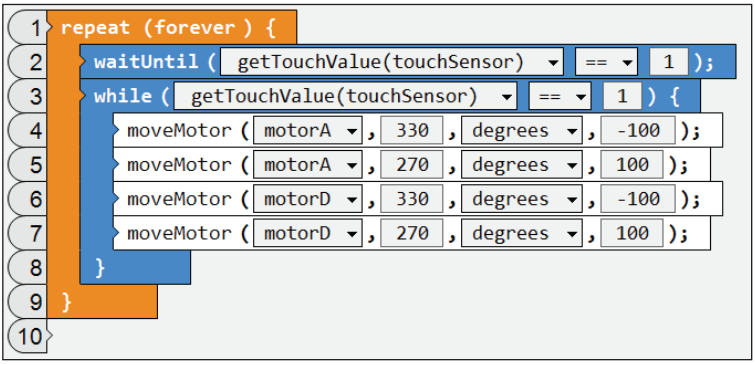


215. ábra. *A kártyakeverő robot MakeCode programja*

A JavaScript kód:

```
sensors.touch1.onEvent(ButtonEvent.Pressed, function () {
    motors.largeA.setRegulated(true)
    motors.largeD.setRegulated(true)
    control.runInParallel(function () {
        while (sensors.touch1.isPressed()) {
            motors.largeD.run(-100, 480, MoveUnit.MilliSeconds)
            motors.largeD.run(100, 400, MoveUnit.MilliSeconds)
        }
    })
    pause(200)
    while (sensors.touch1.isPressed()) {
        motors.largeA.run(-100, 480, MoveUnit.MilliSeconds)
        motors.largeA.run(100, 400, MoveUnit.MilliSeconds)
    }
})
```

Az összehasonlító elemzés megvalósításának érdekében a fenti programot ROBOTC nyelvre is átültetjük, mégpedig mind szöveges, mind grafikus üzemmódban. A grafikus környezetben megvalósított programot a 216. ábra mutatja.



216. ábra. *A kártyakeverő robot ROBOTC programja*

A ROBOTC grafikus környezete nem ismeri a párhuzamos programok fogalmát, nem indítható egyszerre több taszk, így szekvenciálisan valósítottuk meg a programot. Mind szöveges, mind grafikus üzemmódban vigyázzunk arra, hogy a ROBOTC függvényei esetén a paraméterek sorrendje nem éppen a más nyelvekben megszokott sorrend, például a moveMotor esetén is először a szöget adjuk meg, azután a sebességet. A grafikus üzemmódban akkor működnek ezek a parancsok, ha a Robot menü Platform Type menüpontjában ki van választva a Natural Language (természetes nyelv).

A szöveges program esetén két változat lehetséges, az egyik a grafikus átírása párhuzamos programmá. Ezt akkor tehetjük meg, ha a Robot menü Platform Type menüpontjában ki van választva a Natural Language (természetes nyelv). Ez a program a következő:

```
task first()
{
  while(getTouchValue(S1) == 1)
  {
    moveMotor(motorA, 330, degrees, -100);
    moveMotor(motorA, 270, degrees, 100);
  }
}

task second()
{
  wait(0.3);
```

```
  while(getTouchValue(S1) == 1)
  {
    moveMotor(motorD, 330, degrees, -100);
    moveMotor(motorD, 270, degrees, 100);
  }
}

task main()
{
  while(true)
  {
    waitUntil(getTouchValue(S1) == 1);
    startTask(first);
    startTask(second);
    waitUntil(getTouchValue(S1) == 0);
  }
}
```

Megfigyelhetjük, hogy tulajdonképpen három párhuzamosan futó programrészünk van. Az egyik a főprogram, ez egy végtelen ciklusban addig várakozik, ameddig le nem nyomjuk az érintésérzékelőt, majd elindítja a két párhuzamosan futó `first` és `second` nevű taszkokat. Ezután addig várakozik, ameddig fel nem engedjük az érintésérzékelőt. A `first` és `second` taszkok az A és a D motorokat kezelik, a második taszk 0,3 másodperc késleltetéssel indul az elsőhöz képest, hogy a kártyák egymásra kerüljenek, és ne akadjanak össze. Mindkét taszkban egy ciklus hajtja a motorokat mindaddig, ameddig az érintésérzékelő be van nyomva.

Nyilván úgy is felépíthettük volna a programot, hogy csak két taszkunk legyen, a főprogram és még egy, ekkor a főprogramba is került volna egy ciklus, de az érdekesség kedvéért itt most a háromtaszkos változatot mutatjuk be.

Ha a Robot menü Platform Type menüpontjában nincs ki választva a Natural Language (természetes nyelv), akkor a program felépítése ugyanaz, de néhány utasítás más. Például nincs meg a `moveMotor` eljárás, helyette a `resetMotorEncoder` eljárással lenullázzuk a motor fordulatszámmérőjét, a `setMotorTarget` eljárással beállítjuk, hogy mennyit menjen (szögben), a `waitUntilMotorStop` eljárással pedig várunk, ameddig a motor leáll, vagy nincs meg a `wait`, helyette a `sleep` eljárás van, és itt a paramétert milliszekundumban kell megadni.

A program a következő:

```
task first()
{
  while(getTouchValue(S1) == 1)
  {
    resetMotorEncoder(motorA);
    setMotorTarget(motorA, 330, -100);
    waitUntilMotorStop(motorA);
```

```
    resetMotorEncoder(motorA);
    setMotorTarget(motorA, 270, 100);
    waitUntilMotorStop(motorA);
  }
}

task second()
{
  sleep(300);
  while(getTouchValue(S1) == 1)
  {
    resetMotorEncoder(motorD);
    setMotorTarget(motorD, 330, -100);
    waitUntilMotorStop(motorD);
    resetMotorEncoder(motorD);
    setMotorTarget(motorD, 270, 100);
    waitUntilMotorStop(motorD);
  }
}

task main()
{
  while(true)
  {
    waitUntil(getTouchValue(S1) == 1);
    startTask(first);
    startTask(second);
    waitUntil(getTouchValue(S1) == 0);
  }
}
```

Ha Bricx CC környezetet kívánunk használni, akkor figyelembe kell vennünk azt, hogy az EV3-as tégla érzékelőit még nem támogatja a rendszer, így mellőznünk kell ezt, csak a tégla gombjait használhatjuk, vagy ha tudjuk, hogy hány kártyalapunk van, egy ciklusban programozzuk le a keverést.

Ciklussal így valósíthatjuk meg:

```
1. #include „C:\APPS\Bricx\API\ev3_command.h”
2. #include „C:\APPS\Bricx\API\ev3_output.h”
3.
4. int main()
5. {
6.   OutputInit();
7.   int i;
8.   for(i=0; i<16; ++i)
9.   {
10.     RotateMotorEx(OUT_A, -100, 330, 0, false, true);
```

```
11.     RotateMotorNoWaitEx(OUT_A, 100, 270, 0, false, true);
12.     RotateMotorEx(OUT_D, -100, 330, 0, false, true);
13.     RotateMotorEx(OUT_D,  100, 270, 0, false, true);
14.   }
15.   OutputClose();
16.   OutputExit();
17.   return 0;
18. }
```

Megfigyelhetjük, hogy a párhuzamos futtatást úgy oldottuk meg, hogy az első `RotateMotorEx` hívás után egy `RotateMotorNoWaitEx` hívás következik, amely nem várja már meg, hogy befejeződjenek a többi motormozgások, hanem aszinkron módon, párhuzamosan forgatja a motort.

A bal téglagomb lenyomásáig pedig így működik a robot:

```
1. #include „C:\APPS\Bricx\API\ev3_command.h”
2. #include „C:\APPS\Bricx\API\ev3_output.h”
3. #include „C:\Apps\Bricx\API\ev3_button.h”
4. #include „C:\Apps\Bricx\API\ev3_timer.h”
5.
6. int main()
7. {
8.   OutputInit();
9.   ButtonLedInit();
10.   ButtonLedOpen();
11.   while(!ButtonIsDown(BUTTON_ID_LEFT))
12.   {
13.     RotateMotorEx(OUT_A, -100, 330, 0, false, true);
14.     RotateMotorNoWaitEx(OUT_A, 100, 270, 0, false, true);
15.     RotateMotorEx(OUT_D, -100, 330, 0, false, true);
16.     RotateMotorEx(OUT_D,  100, 270, 0, false, true);
17.   }
18.   OutputClose();
19.   OutputExit();
20.   return 0;
21. }
```

A kísérletek során megfigyelhettük, hogy ahány programozási nyelvet választottunk, annyiféleképp kellett gondolkozni, kihasználni a nyelv előnyeit, lehetőségeit, figyelni a nyelv jellegzetességeire. Nincs két egyforma program, két egyforma megoldás. Végkövetkeztetésként azt mondhatjuk el, hogy az EV3 robot, ha nem is a legkényelmesebben, de minden bizonnyal a legprecízebben, legpontosabban a saját LEGO MINDSTORMS EV3 Home Edition vagy LEGO MINDSTORMS Education EV3 környezeteiben programozható. Az imperatív nyelvek gyors programírási lehetőségei sem vehetik fel a versenyt a saját környezet blokkjainak pontosságával.

5.7. Javasolt feladatok

5.7.1. A robot

Ő **Szaffi**, a robotunk. Lánctalpas, két motorja van és négy érzékelője: hátul egy érintésérzékelő, elöl egy színérzékelő, amely lefele néz, valamint egy infravörös és egy ultrahangos érzékelő, amelyek előre néznek. Szeret játszani, s csak úgy falja a jobbnál jobb programokat. A robotot mindig ráhelyezzük a pálya szélére, majd a pálya másik szélén automatikusan megáll. Az eredményeket a kijelzőre írja ki. **Szaffi** várja az utasításaidat, programjaidat. Játsszatok együtt! Legyetek kreatívak!

A fenti leírás alapján építsük meg a lánctalpas robotot! Ha nincs ultrahangos vagy infravörös érzékelőnk, akkor csak azt kell felszerelni, amelyik van.

5.7.2. Vonalkövetés

Kövesd a fekete vonalat. Mérd meg a hosszát – számítsd ki abból, hogy a kerekek hányat fordultak. Közelítsd meg képlettel az ellipszis kerületét, írd ki a robot képernyőjére a mért és a számított értéket!

5.7.3. Akadályok

Adott két akadály és a köztük lévő robot. Mérd meg az ultrahang-érzékelő segítségével, hogy milyen távol van egymástól a két akadály. Mérd le ezt a távolságot a kerekek fordulatszámából is. Tolass hátra addig, ameddig az érintésérzékelő megállítja a robotot!

5.7.4. Szonár

Készíts olyan programot, amelynek eredményeképp a robot arányosan akkora sugarú körlapot rajzol a kijelző közepére, amilyen távol észlel magától menet közben egy akadályt, majd kikerüli az akadályt, és tovább halad a fekete vonalig!

5.7.5. Robotfoci

A pályán egy labda és egy kapu található. A kapu helyét a bekapcsolt infravörös távirányító jelzi. A robot keresse meg az ultrahangos érzékelője segítségével a labdát, majd juttassa be a kapuba!

5.7.6. Háború

A kék, a zöld, a sárga, a fehér, a barna és a piros városállamok között egyre jobban elmérgesedett a viszony, majd kitört a háború. Mindenki mindenki ellen harcolt keményen, az egyedüli engedmény csupán az volt, hogy a városállamok között robotok biztosíthatják szabadon a postát, a robotokra nem támadnak az ellenséges csapatok, viszont a robotok a lehető legrövidebb utakon közlekedhettek a városállamok között. A robot egy tömbbe sorfolytonosan építse fel a városállamok gráfját, majd számítsa ki az egyes települések közötti legrövidebb utakat! A megoldás során a robot bluetoothon keresztül kommunikálhat is a számítógéppel. A robot jusson el a startból a stopba!

5.7.7. Morse

A távirányító segítségével adjunk meg egy max. háromjegyű számot, majd a robot „énekelje el” a szám Morse-kódját!

5.7.8. Véletlen

Írj programot, amely során a robot véletlenszerűen sorsol egy számot 1 és 100 között. Ha páros számot sorsolt, ütközésig tolat, majd visszatér az első fekete vonalig, ha páratlan számot sorsolt, akkor előremegy az első zöld vonalig, majd visszatér, ahonnan jött. Mindezt végrehajtja összesen 5-ször, és a kisorsolt számokat a képernyőre írja egymás fölötti sorokba!

6. ALAPÉRTELMEZETT HANGOK ÉS KÉPEK

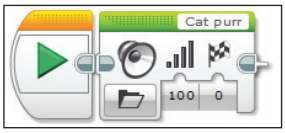
6.1. Hangok

A következőkben felsoroljuk a szabványos LEGO EV3 hangállományokat, amelyek *.RSF állományformátumban találhatók meg.

Ezekre a nevükkel hivatkozhatunk a különböző blokkokban.

Ne felejtsük el, hogy az EV3 tégla állomány- és könyvtárnevei érzékenyek a kis- és nagybetűkre, mivel a téglán Linux operációs rendszer fut [12]. Nem mindegy tehát, ha „Motor start”-ot vagy „Motor Start”-ot írunk. A másodikat nem fogja felismerni a rendszer.

Az állományneveket kiterjesztés nélkül kell megadni.



217. ábra. *A hangfal blokk*

6.1.1. Állatok

– Cat purr (macskamorgás)
– Dog bark 1 (kutyaugatás 1)
– Dog bark 2 (kutyaugatás 2)
– Dog growl (kutyamorgás)
– Dog sniff (kutyaszimatolás)
– Dog whine (kutyanyafogás)
– Elephant call (elefánthívás)
– Insect buzz 1 (rovarzümmögés 1)
– Insect buzz 2 (rovarzümmögés 2)
– Snake hiss (kígyósziszegés)
– Snake rattle (kígyócsörgés)
– T-rex roar (t-rex-ordítás)

6.1.2. Színek

– Black (fekete)
– Blue (kék)
– Brown (barna)

– Green (zöld)
– Red (piros)
– White (fehér)
– Yellow (sárga)

6.1.3. Kommunikáció

– Bravo (Brávó!)
– EV3
– Fantastic (Fantasztikus!)
– Game over (Játék vége)
– Go (Menj!)
– Good job (Jó munka!)
– Good (Jó!)
– Goodbye (Viszlát!)
– Hello (Helló!)
– Hi (Szia!)
– LEGO
– MINDSTORMS
– Morning (Reggel)
– No (Nem!)
– Okay (Oké!)
– Okey-dokey (Oké-zsoké!)
– Sorry (Bocsánat!)
– Thank you (Köszönöm!)
– Yes (Igen!)

6.1.4. Kifejezések

– Boing (pattogás)
– Boo (hurrogás)
– Cheering (éljenzés)
– Crunching (ropogás)
– Crying (sírás)
– Fanfare (harsonaszó)
– Kung fu
– Laughing 1 (nevetés 1)
– Laughing 2 (nevetés 2)
– Magic wand (varázspálca)
– Ouch (Jaj!)
– Shouting (kiabálás)
– Smack (cuppanás)

– Sneezing (tüsszentés)
– Snoring (horkolás)
– Uh-oh (O-ó!)

6.1.5. Információk

– Activate (aktiválás)
– Analyze (elemzés)
– Backwards (visszafelé)
– Color (szín)
– Detected (észlelés)
– Down (le)
– Error alarm (hibariasztás)
– Error (hiba)
– Flashing (villogtatás)
– Forward (előre)
– Left (bal)
– Object (tárgy)
– Right (jobb)
– Searching (keresés)
– Start (Rajt!)
– Stop (Állj!)
– Touch (érintés)
– Turn (kanyarulás)
– Up (fel)

6.1.6. Mechanikus

– Air release (levegőkieresztés)
– Airbrake (légnyomásos fék)
– Backing alert (tolatáshang)
– Blip 1 (csipogás 1)
– Blip 2 (csipogás 2)
– Blip 3 (csipogás 3)
– Blip 4 (csipogás 4)
– Horn 1 (kürt 1)
– Horn 2 (kürt 2)
– Laser (lézer)
– Motor idle (motor alapjáraton)
– Motor start (motorindítás)
– Motor stop (motorleállítás)
– Ratchet (racsni)

– Sonar (hanglokátor)
– Tick tack (tik-tak)
– Walk (séta)

6.1.7. Mozgások

– Arm 1 (kar 1)
– Arm 2 (kar 2)
– Arm 3 (kar 3)
– Arm 4 (kar 4)
– Drop load (leeső teher)
– Lift load (emelő teher)
– Servo 1 (szervo 1)
– Servo 2 (szervo 2)
– Servo 3 (szervo 3)
– Servo 4 (szervo 4)
– Slide load (csúszó teher)
– Snap (csattanás)
– Speed down (lassulás)
– Speed idle (sebesség alapjáraton)
– Speed up (felgyorsulás)
– Speeding (gyors)

6.1.8. Számok

– Zero (zéró)
– One (egy)
– Two (kettő)
– Three (három)
– Four (négy)
– Five (öt)
– Six (hat)
– Seven (hét)
– Eight (nyolc)
– Nine (kilenc)
– Ten (tíz)

6.1.9. Rendszer

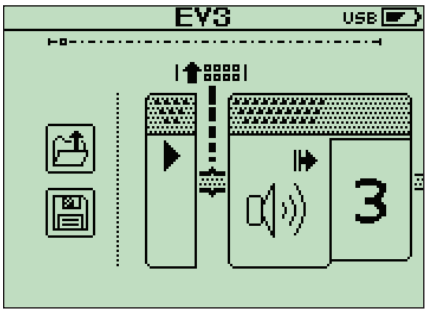
– Click (kattintás)
– Confirm (megerősítés)
– Connect (helyes)

– Download (letöltés)
– General alert (általános riasztás)
– Overpower (túlerő)
– Power down (kikapcsolás)
– Ready (kész!)
– Start up (üzembe helyezés)

6.1.10. Az EV3 tégla program alkalmazásának hangjai

Az alábbiakban azt a 12 hangot soroljuk fel, amelyeket az EV3 tégla program alkalmazásában használhatunk a hangfal blokkon. Ezekre a sorszámukkal hivatkozhatunk, nem a nevükkel (lásd 203. ábra).

– 1. Hello (Helló!)
– 2. Goodbye (Viszlát!)
– 3. Fanfare (harsonaszó)
– 4. Error alarm (hibariasztás)
– 5. Start (Rajt!)
– 6. Stop (Állj!)
– 7. Object (tárgy)
– 8. Ouch (Jaj!)
– 9. Blip 3 (csipogás 3)
– 10. Arm 1 (kar 1)
– 11. Snap (csattanás)
– 12. Laser (lézer)



218. ábra. *Hangfal blokk az EV3 téglán*

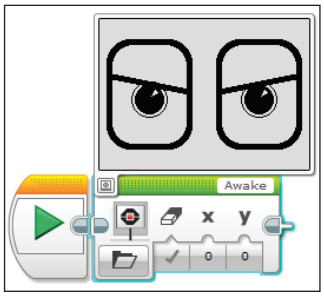
6.2. Képek

A következőkben felsoroljuk a szabványos LEGO EV3 képállományokat, amelyek *.RGF állományformátumban találhatók meg.

Ezekre a nevükkel hivatkozhatunk a különböző blokkokban.

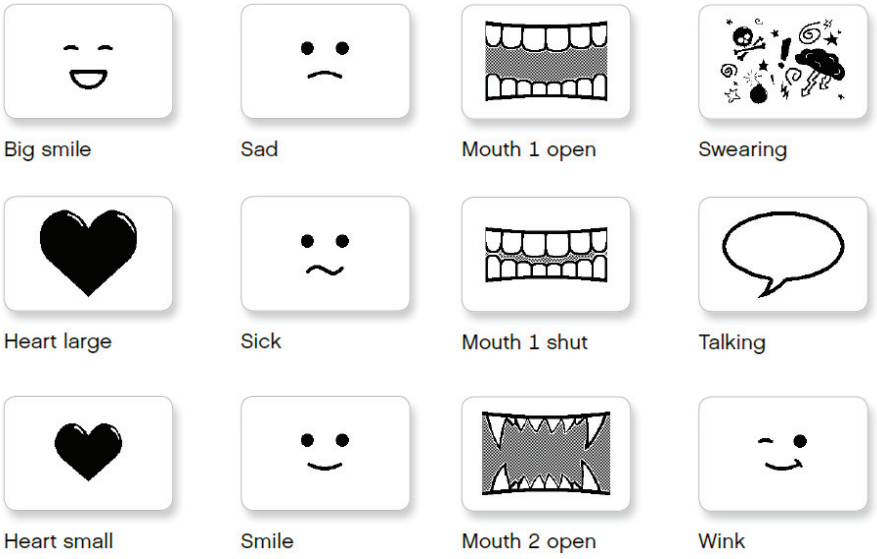
Ne felejtsük el, hogy az EV3 tégla állomány- és könyvtárnevei érzékenyek a kis- és nagybetűkre.

Az állományneveket kiterjesztés nélkül kell megadni.



219. ábra. *A kijelző blokk*

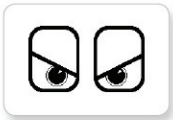
6.2.1. Kifejezések



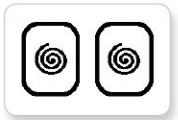
Mouth 2 shut

ZZZ

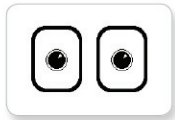
6.2.2. Szemek



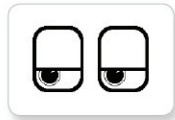
Angry



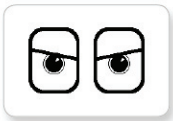
Dizzy



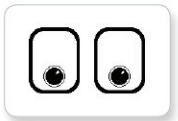
Neutral



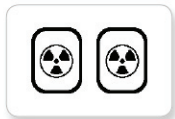
Tired left



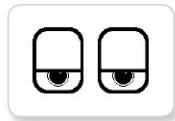
Awake



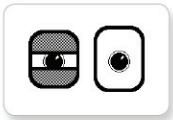
Down



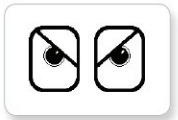
Nuclear



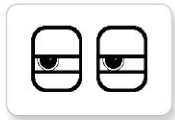
Tired middle



Black eye

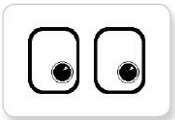


Evil

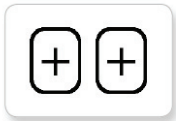


Pinch left

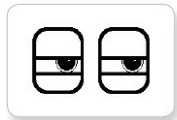
Tired right



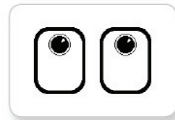
Bottom right



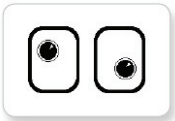
Knocked out



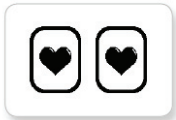
Pinch right



Up

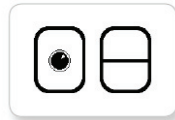


Crazy 1

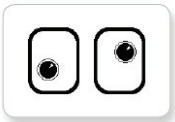


Love

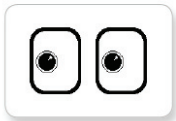
Sleeping



Winking

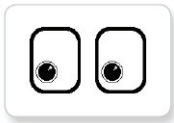


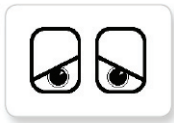
Crazy 2

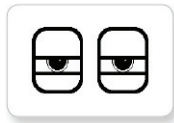


Middle left

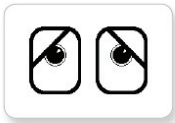
Tear

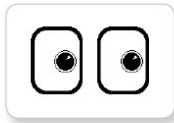

Bottom left


Hurt


Pinch middle


Toxic


Disappointed


Middle right

6.2.3. Információk


Accept


No go


Thumbs down


Forward

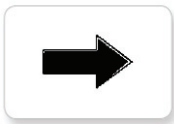

Backward


Question mark


Thumbs up


Left


Decline

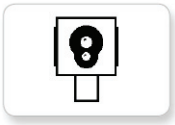

Right


Warning

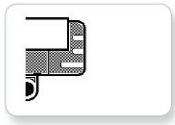
Stop 1

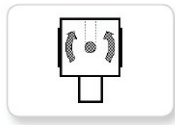
Stop 2

6.2.4. LEGO

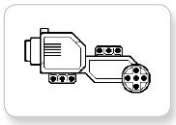

Color sensor

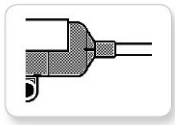

IR sensor


Sound sensor


Gyro sensor


EV3 icon

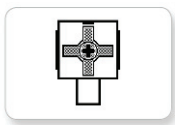

Large motor


Temp. sensor

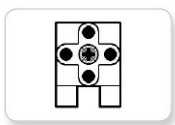

IR beacon

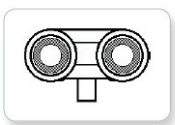
EV3

LEGO


Touch sensor

MINDSTORMS


Medium motor


US sensor

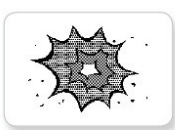
6.2.5. Tárgyak


Bomb


Lightning

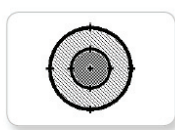

Flowers


Snow


Boom


Night


Forest


Target

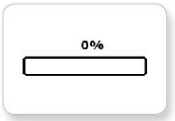

Fire


Pirate


Light off


Light on

6.2.6. Folyamatok

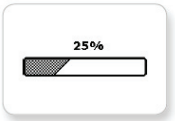


Bar 0


Dial 2


Hourglass 0


Timer 4

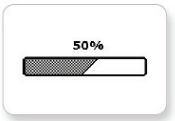


Bar 1

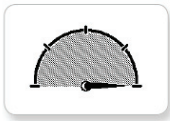

Dial 3


Hourglass 1


Water level 0

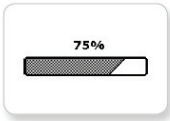


Bar 2


Dial 4


Hourglass 2


Water level 1

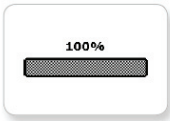


Bar 3

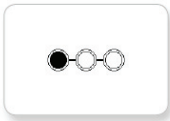

Dots 0

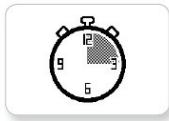

Timer 0


Water level 2

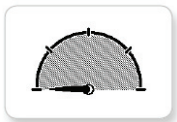


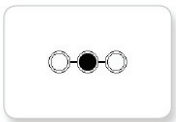
Bar 4


Dots 1

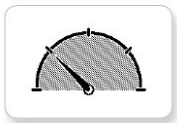

Timer 1

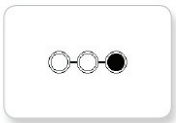

Water level 3


Dial 0


Dots 2


Timer 2


Dial 1


Dots 3


Timer 3

6.2.7. Rendszer


Accept 1


Dot empty


Slider 0


Slider 6


Accept 2


Dot full


Slider 1


Slider 7


Alert

EV3 small

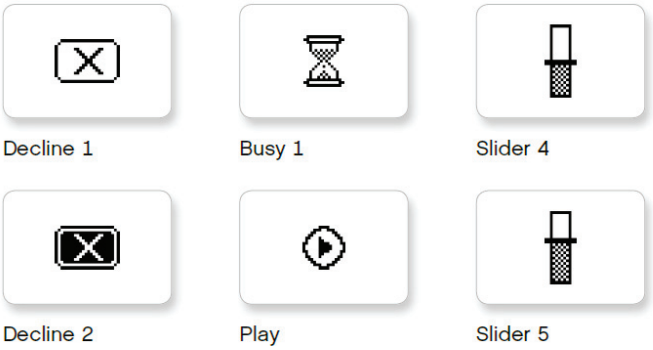

Slider 2


Slider 8


Box

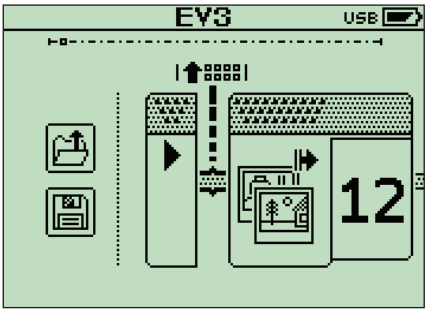

Busy 0


Slider 3



6.2.8. Az EV3 tégla program alkalmazásának képei

Az alábbiakban azt a 12 képet soroljuk fel, amelyeket az EV3 tégla program alkalmazásában használhatunk a kijelző blokkon. Ezekre a sorszámukkal hivatkozhatunk, nem a nevükkel (lásd 220. ábra).



220. ábra. *Kijelző blokk az EV3 téglán*

SZAKIRODALOM

[1] Ayad, Tony: *EV3 Programming Overview for FLL Coaches*, http://www.firstroboticscanada.org/main/wp-content/uploads/2013EV3Programming.pdf
[2] Barbalics Dóra Krisztina; Solymos Dóra: *Lego Mindstorms EV3 robotok programozása*, ELTE, Budapest, 2018.
[3] Baum, Dave; Hansen, John: *NQC Programmer's Guide. Version 3.1 r5*.
[4] Claus, N. (pseudonym for Édouard Lucas): *La Tour d'Hanoï: V'eritable Cassetête Annamite*, Original instruction sheet printed by Paul Bousrez, Tours, 1883.
[5] Griffin, Terry: *The Art of LEGO® Mindstorms® EV3 Programming*, No Starch Press, 2014.
[6] Hansen, John: *Not eXactly C (NXC) Programmer's Guide.*
[7] http://botbench.com/blog/2013/01/08/comparing-the-nxt-and-ev3-bricks/
[8] http://bricxcc.sourceforge.net/
[9] http://education.lego.com/es-es/products
[10] http://en.wikipedia.org/wiki/ARM9
[11] http://en.wikipedia.org/wiki/Lego_Mindstorms
[12] http://en.wikipedia.org/wiki/Linux_kernel
[13] http://ev3.fantastic.computer/doxygen/index.html
[14] http://ev3.fantastic.computer/doxygen-all/index.html
[15] http://ev3lessons.com/
[16] http://hu.wikipedia.org/wiki/ARM_architekt%C3%BAra
[17] http://hu.wikipedia.org/wiki/MOS_Technology_6502
[18] http://hu.wikipedia.org/wiki/Robot
[19] http://in4lio.github.io/ev3dev-c/sensor_8c-example.html
[20] http://mindstorms.lego.com/en-us/Default.aspx?domainredir=lego.com
[21] http://nyelvek.inf.elte.hu/leirasok/RobotC/
[22] http://www.afrel.co.jp/en/archives/848
[23] http://www.ev-3.net/en/archives/850
[24] http://www.ev3dev.org/
[25] http://www.geeks.hu/blog/ces_2013/130108lego_ mindstorms_ev3
[26] http://www.hdidakt.hu/mindstorms.php?csoport=50
[27] http://www.hdidakt.hu/termekek/ev3-tartozekok-es-kiegeszitok/320-lego-mindstorms-education-ev3-intelligens-tegla.html
[28] http://www.lego.com/en-us/mindstorms/support/faq/
[29] http://www.lego.com/hu-hu/mindstorms/downloads/software/ddsoftwaredownload/download-software/

[30] http://www.legomindstormsrobots.com/lego-mindstorms-ev3/programming-ev3-c-bricxcc/
[31] http://www.leg-technic.hu/blog/38/31313-mindstorms-ev3-az-itelet-elso-napja
[32] http://www.leg-technic.hu/blog/39/31313-mindstorms-ev3-az-itelet-masodik-napja
[33] http://www.philohome.com/sort3r/sort3r.htm
[34] http://www.robotnav.com/motors/
[35] https://github.com/in4lio/ev3dev-c
[36] https://github.com/JorgePe/microbit
[37] https://github.com/mindboards/ev3sources
[38] https://stackoverflow.com/questions/28526200/how-to-program-lego-mindstorms-ev3-using-c-language
[39] Kiss Róbert: *A MINDSTORMS® EV3 robotok programozásának alapjai*, Budapest, 2014.
[40] LEGO Mindstorms EV3 Felhasználói útmutató (www.lego.com)
[41] LEGO MINDSTORMS EV3 Home Edition súgó
[42] Overmars, Mark: *Programming Lego Robots using NQC*.
[43] Park, Eun Jung: *Exploring LEGO® Mindstorms® EV3: Tools and Techniques for Building and Programming Robots*, John Wiley & Sons, Inc., Indianapolis, 2014.
[44] Schueller, Albert W.: *Programming with Robots*, Whitman College, Washington, 2009.
[45] Valk, Laurens: *LEGO MINDSTORMS EV3 Discovery Book: A Beginner's Guide to Building and and Programming Robots*, No Starch Press, 2014.

REZUMAT

Cartea *Programarea roboților LEGO* prezintă structura şi programarea roboților LEGO Mindstorms EV3, pornind de la istoria acestora şi generațiile anterioare.

Este prezentată cărămida inteligentă precum şi sunt prezentate dispozitivele auxiliare (senzori, motoare etc.).

Paradigma vizuală şi imperativă, dar şi programarea pe cărămidă este inclusă în carte, care descrie mai multe limbaje de programare, interfețe şi medii de programare.

Este prezentată mediul de programare LEGO MINDSTORMS EV3 Home Edition, platforma nativă a roboților LEGO, dar şi mediile de programare Bricx CC, Microsoft MakeCode şi ROBOTC.

O serie de probleme rezolvate şi propuse închide volumul.

ABSTRACT

The book *Programming LEGO robots* presents the structure and programming of LEGO Mindstorms EV3 robots, looking back at the history and previous generations.

Both the intelligent brick and the auxiliary devices (sensors, motors, etc.) are presented.

The visual and imperative paradigm, but also brick programming, plays an important role in the book, which describes several programming languages and environments.

The LEGO MINDSTORMS EV3 Home Edition programming environment, the native platform of LEGO robots, is presented as well as the Bricx CC, Microsoft MakeCode, and ROBOTC programming environments.

A series of solved and proposed problems closes the volume.

A SZERZŐRŐL

Kovács Lehel István 1975-ben született Brassóban. Elemi iskolai tanulmányait Négyfaluban, a középiskolát Brassóban végezte. 1997-ben szerzett egyetemi oklevelet a Babeş–Bolyai Tudományegyetem informatika szakán. 1998-ban ugyanott magiszteri oklevelet szerzett, majd doktori tanulmányokat folytatott Budapesten és Kolozsváron. Doktori disszertációját 2006-ban védte meg Kolozsváron.

1997-től a kolozsvári Babeş–Bolyai Tudományegyetem, 2007-től a Sapientia Erdélyi Magyar Tudományegyetem Marosvásárhelyi Karának oktatója.

Főbb kutatási területei: fordítóprogramok, értelmezők, programozási nyelvek, számítógépes rendszerek tervezése, számítógépes grafika, virtuális valóságok, robotika.

2020-ig 99 szakmai konferencián vett részt, 9 szakkönyv szerzője, 4 kötet szerkesztője, 37 elismert folyóiratban közölt szakcikk, 168 tudomány-népszerűsítő cikk szerzője, 3 szaklap szerkesztője, 25 kutatási program résztvevője, vezetője.

Scientia Kiadó
400112 Kolozsvár (Cluj-Napoca)
Mátyás király (Matei Corvin) u. 4. sz.
Tel./fax: +40-364-401454
E-mail: scientia@kpi.sapientia.ro
www.scientiakiado.ro

Műszaki szerkesztés:
Metaforma Kft.

Borítóterv:
Tipotéka Kft.

Korrektúra:
Szenkovics Enikő

Tipográfia:
Könczey Elemér

Nyomdai munkálatok:
F&F INTERNATIONAL Kft.
Felelős vezető: Ambrus Enikő igazgató

A robotok a jövő technikáját, technológiáját képezik, programozásuk pedig a jövő informatikusai számára jelent nagy kihívást. A könyv a LEGO Mindstorms EV3 robotok felépítését és programozását mutatja be, visszatekintve az előzményekre, az előző robotgenerációkra is. A vizuális és imperatív paradigma, illetve a téglán való programozás is szerepet kap a könyvben, amely több programozási nyelvet, felületet, környezetet ismertet: LEGO MINDSTORMS EV3 Home Edition, Bricx CC, Microsoft MakeCode, ROBOTC, Scratch 3.0. Megoldott és javasolt feladatok sora zárja a kötetet, amelyek segítségével jobban betekinthetünk a robotok programozásának fortélyaiba, kibővíthetjük gyakorlati ismereteinket. A LEGO robotokkal játszva tanulhatunk, programozhatunk!

ISBN 978-606-975-041-4