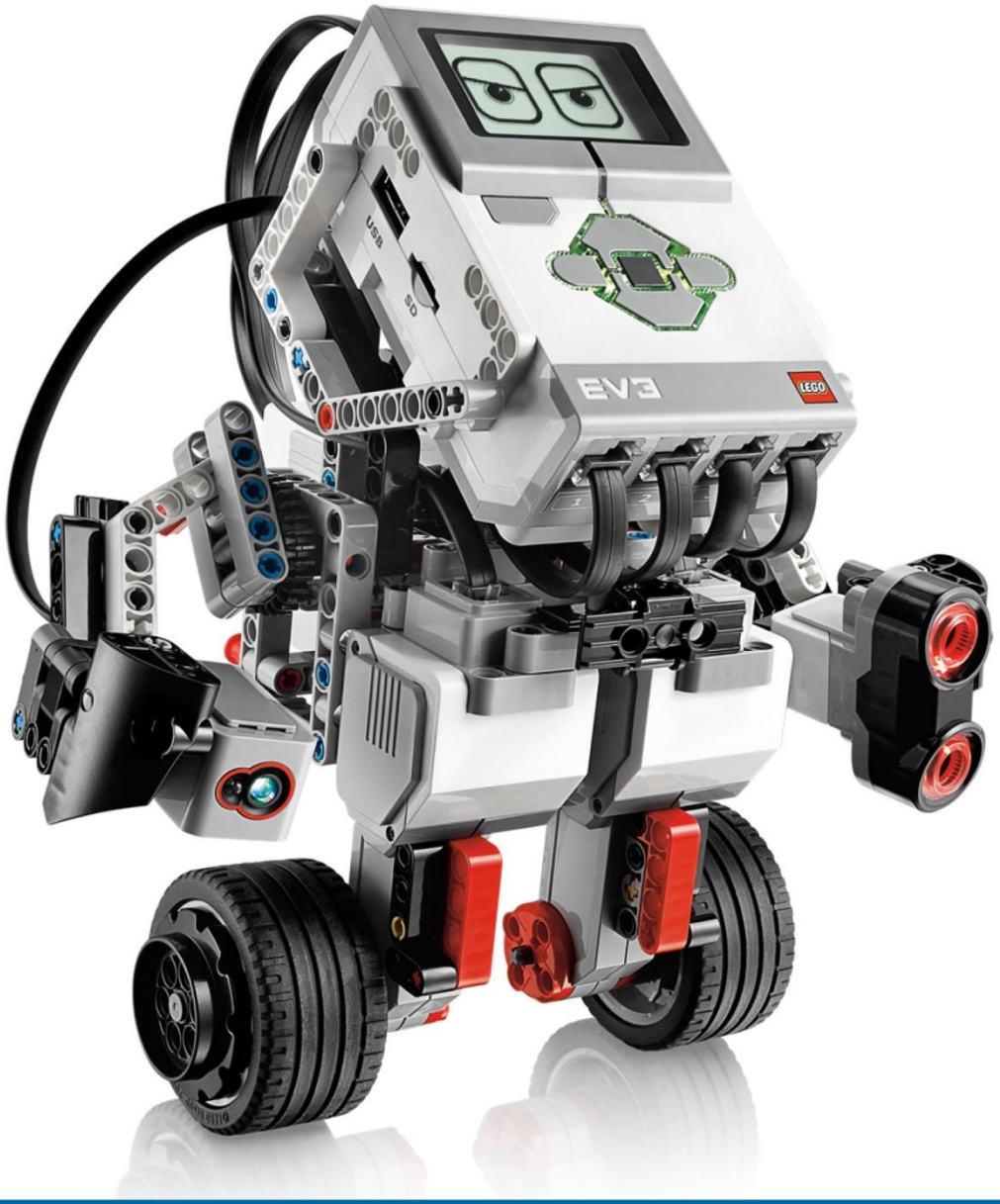
KISS RÓBERT

A MINDSTORMS® EV3 robotok programozásának alapjai

A könyv elektronikus változatának kiadása a National Instruments Hungary Kft.
és a H-Didakt Kft. jóvoltából jöhetett létre.



2014

ni.com

NATIONAL INSTRUMENTS

Szerző: Kiss Róbert
Kiadás éve: 2014
Szakmai lektor: dr. Pásztor Attila

A könyv tartalmának közel 30%-a az „Egyszerű robotika A MINDSTORMS® NXT robotok programozásának alapjai" (2010) című kiadvány átdolgozott változata, így külön köszönet Badó Zsoltnak, hogy lehetővé tette az ott általa leírtak átvételét.

TARTALOMJEGYZÉK

Bevezető **5**
1. Robothardver **6**
1.1. Történeti bevezető 6
1.2. Az „intelligens” tégla és egyéb hardver elemek (MINDSTORMS EV3) 8
1.3. Input eszközök: alapszenzorok 9
1.4. Input eszközök: egyéb szenzorok 10
1.5. Output eszközök: szervomotorok 12
1.6. A programozás során használt tesztrobot 13
2. Keretprogram **14**
2.1. Általános bemutatás 14
2.2. A használt programváltozat és összetevői 15
2.3. A programozási környezet alapelemei 17
2.4. Programírás, első lépések 23
3. A robot képernyőmenüje **27**
4. Egyszerű mozgások **29**
4.1. Motorok vezérlése 29
4.2. Gyakorló feladatok 32
5. Szenzorok használata **34**
5.1. Szín- vagy fényérzékelő (*Colour Sensor*) 36
5.2. Várakozás megadott ideig (*Timer*) 40
5.3. Ütközésérzékelő (*Touch sensor*) 41
5.4. Ultrahangos távolságérzékelő (*Ultrasonic sensor*) 41
5.5. Giroszkóp (*Gyro sensor*) 42
5.6. Gyakorló feladatok 43
6. Vezérlési szerkezetek **44**
6.1. Ciklusok 44
6.2. Elágazások 49
6.3. Gyakorló feladatok 55
7. Paraméterek, változók és a paraméterátadás **56**
7.1. Alapvető paraméterek, adattípusok a programnyelveknél 56
7.2. Paraméterátadás a programon belül 58
7.3. Szenzorok paraméterezése 60
7.3.1. Colour szenzor blokk 60
7.3.2. Ultasonic szenzor blokk 61
7.3.3. Touch szenzor blokk 62
7.3.4. Timer szenzor blokk 62
7.3.5. Motor Rotation blokk 63
7.3.6. Infrared szenzor blokk 63
7.3.7. Brick Buttons blokk 63
7.3.7. Gyro szenzor blokk 64
7.4. Változók 67
7.5. Tömbök 72
7.6. Konstansok 74
7.7. Gyakorló feladatok 75
8. Képernyőkezelés **78**
8.1. A képernyő programozása 79

8.1.1. Szöveg kiíratása ... *80*
8.1.2. Alakzat rajzolása ... *81*
8.1.3. Képek rajzolása ... *83*
8.2. Gyakorló feladatok ... 87
9. Matematikai és logikai műveletek ... 89
9.1. Műveletek csoportosítása ... 89
9.2. Számértéket visszaadó műveletek ... 89
9.3. Logikai értéket visszaadó műveletek ... 91
9.4. Gyakorló feladatok ... 99
10. Többszálú programok – taszkok ... 101
10.1. Többszálú programok létrehozása ... 101
10.2. Gyakorló feladatok ... 104
11. Saját utasításblokk ... 106
11.1. Saját blokkok létrehozása ... 106
11.2. Gyakorló feladatok ... 118
12. Mérésnaplózás ... 120
12.1. Mérési paraméterek beállítása ... 120
12.1.1. Első kísérlet ... *121*
12.1.2. Második kísérlet ... *125*
13. Kommunikáció ... 128
14. Egyebek ... 135
14.1. Hanglejátszás ... 135
14.2. Fájlkezelés ... 136
14.3. Gyakorló feladatok ... 142
15. Versenyfeladatok és megoldásaik ... 143
16. Összetett feladatok ... 156
17. A NATIONAL INSTRUMENTS ... 164
17.1. A National Instrumentsről ... 164
17.2. Küldetésünk ... 164
17.3. Megoldásaink ... 165
17.4. Alkalmazási területek, iparágak ... 165
17.5. Szolgáltatások ... 166
17.6. A LabVIEW grafikus fejlesztői környezet ... 166
17.7. A National Instruments az oktatásban ... 168
17.8. NI myDAQ ... 168
Tulajdonságok ... *169*
Specifikáció ... *169*
17.9. NI myRIO ... 170
17.10. NI ELVIS II ... 171
17.11. A National Instruments és a LEGO ® együttműködése ... 172
17.12. Elérhetőségek ... 174

BEVEZETŐ

A közeljövő technikai fejlődésének domináns irányai az informatika specializálódásán keresztül válnak valósággá. A XX. század *science fiction* regényeinek elképzelései közül egyre több inkább a tudomány, mintsem a fikció kategóriájába sorolható. A robotika az Isaac Asimov által megálmodott kitalációból hétköznapi gyakorlattá vált.
Az oktatás szükségessége nehezen vitatható, hiszen a háztartásokban jelenleg is számos olyan elektronikus eszköz működik, amely processzorvezérelt technológiára épül, a szórakoztató és kommunikációs elektronikától a fejlettebb háztartási gépekig. Ezeknek az eszközöknek az aránya egyre nő, és a jövő generációjának a gombok nyomogatásán túlmutató kompetenciákkal kell rendelkeznie, hogy ne váljon mindez a „varázslat" és „misztikum" eszközévé, hanem az egyszerű eszközhasználó is lássa és tudja a mögöttes algoritmusok emberi kreativitásban rejlő működését.
Az informatikai fejlesztések egyik legmeghatározóbb XXI. századi irányának tehát a robotika fejlődése tűnik.

Könyvünk azt a célt tűzte maga elé, hogy mindenki számára elérhető, egyszerű eszközökön keresztül mutassa be a robotok programozásának lehetőségeit, helyenként játékosnak tűnő formában. A bemutatott példák mögött olyan programozástechnikai eszközök és működési elvek húzódnak meg, amelyek alapjai lehetnek a robotika robbanásszerű fejlődésének, a jövő generáció mérnöki tudásának.

A programok felépítésének megértése, a robotok működésének jelenlegi elvei és korlátai a bemutatott eszközökkel már 10-12 éves kortól lehetővé válnak. A könyv fejezetei fokozatosan nehezedő példákon keresztül vezetnek végig egy egyszerűen használható grafikus programnyelv lehetőségein. A fejezetek végén szereplő feladatok lehetőséget biztosítanak a megtanultak elmélyítésére, és további ötleteket adnak a robotok felhasználásához.
A könyv elsősorban a robotok programozásáról szól, a konstrukciók építésének, tervezésének bemutatása nem volt célunk. A programozói eszköztudás birtokában mindez a következő lépés lehet. Ha már tudjuk, hogy a robotok mire képesek (amit a programjuk előír), akkor adott feladat megoldásához tudunk eszközöket létrehozni. A kreativitás nélkülözhetetlen. Az elérhető eszközök mind a programozás, mind a hardver oldaláról rendelkezésre állnak.

A könyv első hat fejezete a programozást most kezdőknek szól és akár 10 éves kortól használható. A további fejezetek, már komplexebb programozási ismereteket igényelnek.

A kidolgozott példáknál a legtöbb esetben egy lehetséges megoldást közöltünk, sok esetben a megadottól eltérő programozási ötletek is használhatók.

Kecskemét, 2010., 2014.

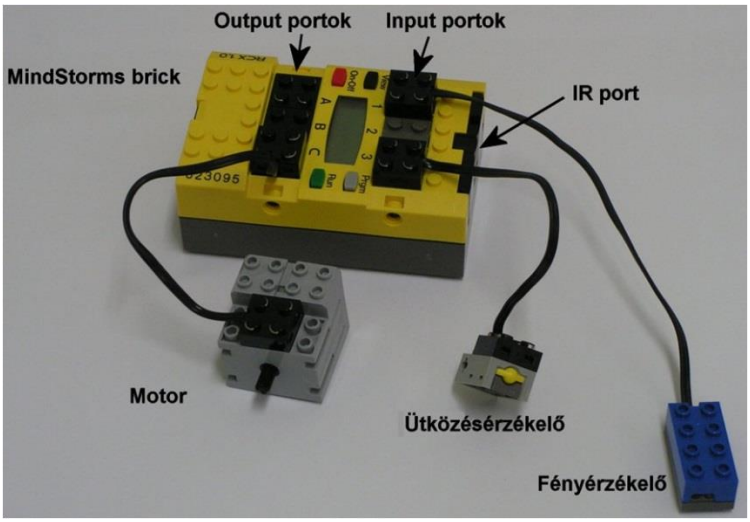
A szerző(k)

1. ROBOTHARDVER

1.1. Történeti bevezető

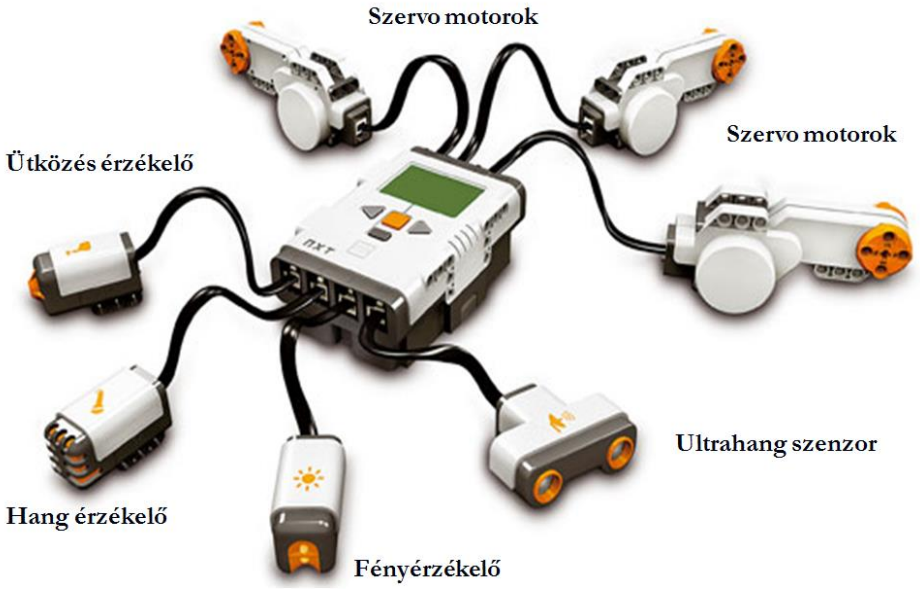
A LEGO a hagyományos konstrukciós játékok készítése mellett már több mint két évtizede foglalkozik robotokkal. Az első általuk készített robotok egyike a 90-es években piacra került *Spybotics* fantázianévre keresztelt processzorvezérelt és programozható egység volt. Nem okozott átütő sikert (legalábbis a hazai piacon), pedig a koncepció alapjai már jól látszottak. A robot két beépített motorral és egy szintén fixen beépített ütközésérzékelővel rendelkezett. Soros porton lehetett a számítógéphez csatlakoztatni, ahol már igen komoly programokat lehetett rá írni (akár C nyelven is), amelyeket áttöltve a robotra, az önállóan hajtotta végre az utasításokat. A termék alapvetően kerekekkel rendelkező konstrukciók építését támogatta. Ezt segítette egy, az eszközhöz tartozó rádiós távirányító is. Gyakorlatilag a LEGO által készített robotok nulladik generációjának tekinthető.

A koncepció első oktatási célra készült változata az 1998-ban megjelent *MINDSTORMS RCX* fantázianevű robot volt. Ez tekinthető az első generációnak.
Ennél a típusnál a modularitás már sokkal komolyabb volt, hiszen a motorok és a szenzorok külön egységként kerültek az eszköz mellé. A robot agyát egy intelligens tégla (*brick*) alkotta, amely infra porton keresztül kommunikált a számítógéppel és akár a többi robottal is. A roboton három bemeneti és három kimeneti csatlakozási pont volt, amelyre motorok és szenzorok kapcsolódhattak. Szenzorként rendelkezésre állt pl.: fényérzékelő, ütközésérzékelő, hőmérsékletmérő, elfordulás érzékelő.



A sikert kihasználva, 2006-ban jelent meg az eszköz második generációs robotja a *MINDSTORMS NXT*. A LEGO a robotot 2006-ban mutatta be a Nürnbergi Játékkiállításon, ahol megnyerte az egyik

nagydíjat. Magyarországon a 2007-es év elejétől kapható kereskedelmi forgalomban. 2008-ban megjelent a *MINDSTORMS NXT 2.0*-ás változata. Elsősorban a szenzorok köre és a keretprogram bővült. A terméknek létezett oktatási és kereskedelmi változata is. A lényeges elemeket tekintve nem volt különbség a két változat között. Sok cég meglátta a lehetőséget az alkalmazásban így LEGO-n kívül mások is elkezdtek különböző szenzorokat gyártani a robothoz.



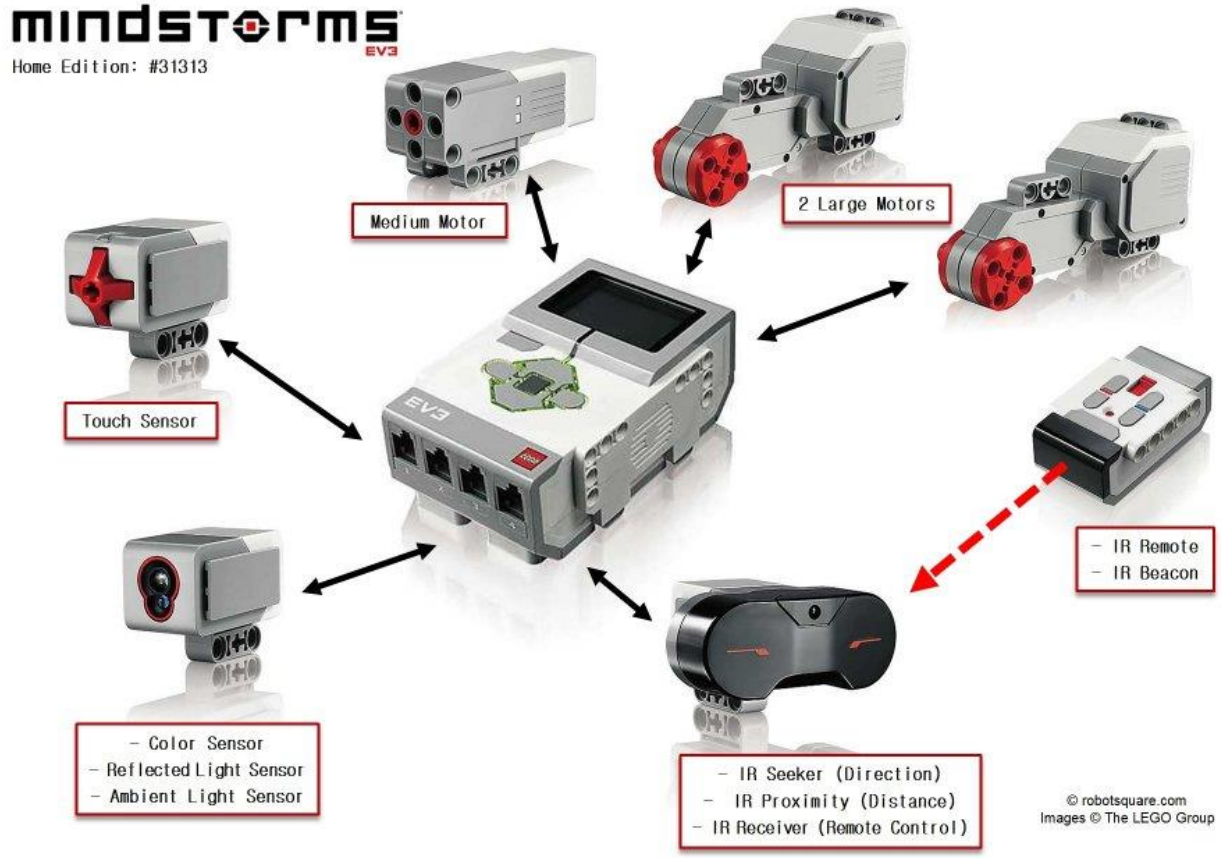
Az eszközhöz gyártott szenzorok köre így jelentősen bővült. A központi egység is sokkal nagyobb teljesítményűvé vált.

A robotkészlet „agya" egy 32 bites AMTEL ARM7 mikrovezérlő volt. A miniszámítógép tartalmazott még 64 Kbyte RAM-ot, 256 Kbyte flash memóriát, 2.0-as USB portot, beépített bluetooth kommunikációs adaptert, 4 db bemeneti és 3 db kimeneti portot a szenzorok és motorok csatlakoztatására, valamint beépített hangszórót és egy 100x64 képpontos grafikus kijelzőt. Az áramforrása 6 db AA típusú elem, vagy saját akkumulátor.

Természetesen az intelligens tégla a készletben található LEGO alkatrészeken kívül bármely egyéb LEGO kompatibilis alkotórésszel összeépíthető volt, így a konstrukciók bonyolultságának csak a kreativitás hiánya szabott határt.

2013-ban jelent meg a 3. generációs robot *MINDSTORMS EV3* fantázianéven. Mind a hardver, mind pedig a vezérlő szoftver jelentős átalakuláson esett át. Az eddigi három motor helyett már négy csatlakoztató az eszközhöz. A készlet alapfelszereléseként jelent meg a giroszenzor, valamint egy új típusú motor, amelynek forgástengelye párhuzamos a motor hosszanti tengelyével (*Medium Motor*). A szenzorok precizitása is nőtt.

A bluetooth kapcsolat mellett a wireless kommunikáció is megjelent. A memória bővíthetőségét mikro SDHC kártyás technikával oldották meg, a számítógép és a robot közötti kommunikációt pedig a vezeték nélküli lehetőségeken kívül egy mikro USB-s kábeles kapcsolaton keresztül is meg lehet valósítani.



Forrás: http://mindstormsev3robots.com/wp-content/uploads/2013/02/EV3Hardware-large.jpg

1.2. Az „intelligens" tégla és egyéb hardver elemek (MINDSTORMS EV3)

A robotkészlet agya egy 300 MHz-es, Linux alapú ARM9-es mikrokontroller. A központi memóriája 64 MB RAM-ot, és 16 MB Flash memóriát tartalmaz, amely a beépített mikro SDHC kártyahelynek köszönhetően tovább bővíthető. Kijelzőként egy 178x128 pixel felbontású monokróm grafikus képernyő tartozik az eszközhöz. Négy bementi és négy kimeneti csatlakozási port található téglán (*brick*), amely a motorok és a szenzorok illesztésére alkalmas.

A bemeneti portokat 1-től 4-ig számozták és jelölték a tégla alsó részén. A három kimeneti portra elsősorban szervomotorok kapcsolhatók, esetleg ledek. A kimeneti portokat A-tól D-ig betűzték a tégla felső részén. Itt kapott helyet a mikro USB csatlakozási pont is, amelynek segítségével pl. számítógéphez csatlakoztatható, és a programok feltöltésére használható. A tégla oldalán találjuk az SDHC kártya bővítő helyet és egy USB csatlakozási pontot.

A működés elve továbbra is az, hogy a megépített robotkonstrukció tartalmazza a téglát, valamint az ahhoz csatlakoztatott szenzorokat és motorokat. Számítógépen elkészítjük a célnak megfelelő programot, amely a szenzorok által érzékelt adatok alapján döntéseket hoz a szükséges tevékenységről, amelyet a robot a motorjai segítségével végrehajt. A programot USB kábelen,

bluetooth-on vagy wifi-n keresztül a robotra töltve az már önállóan viselkedik a programja utasításai alapján.

A bluetooth-os, illetve wireless kommunikáció miatt a robotok egymással is kommunikálhatnak, és programjuk alapján csoportos viselkedésre is képesek.

A robot érzékszervei tehát a szenzorok, amelyek képesek a környezet mérhető adatainak észlelésére, figyelésére. Először röviden ezek közül mutatjuk be a leggyakoribbakat. Mivel a második generációs NXT robot néhány szenzora is használható, ezért azok is szerepelnek az alábbiakban.

1.3. Input eszközök: alapszenzorok

Az eszközhöz több cég is gyárt szenzorokat, így egyre bővülő eszközkészlet áll a rendelkezésre. A szenzorok tehát a robot érzékszervei.

Ütközésérzékelő


NXT változat *EV3 változat*

Az ütközésérzékelő, mint egy kétállású kapcsoló működik. A szenzor érzékeli, amikor a gombot benyomják vagy kiengedik. Ennek megfelelően 0 vagy 1 értéket továbbít a robot a szoftveren keresztül a programnak. A benyomás mértékétől függően a 0-1 közötti érték is differenciálható

Színérzékelő

Valódi színlátást biztosít a robot számára. Az alapszíneket képes megkülönböztetni egymástól.
Több különböző változata is forgalomban van.
A programkörnyezetben beállítható, hogy fényszenzorként működjön.

NXT változat *EV3 változat*

Fényszenzorként a világos és sötét közötti különbséget érzékeli, tehát a fényintenzitás mérhető vele. A visszaadott érték nemcsak a színtől, hanem a felület fényviszonyaitól is függ. Tehát nem a felület színét határozza meg, hanem egy vörös színű fényforrással megvilágított felületről visszaverődő fényintenzitást. A szenzor a programkörnyezetben egy 0-100 közötti értéket szolgáltat a felülettől függően. Használható megvilágítás nélkül is, ekkor a környezet fényét képes érzékelni.

Távolságérzékelő


Az érzékelőt *ultrasonic* szenzornak is nevezik. Az ultrahangos távolságérzékelő a távolságot centiméterben és hüvelykben méri, 0 – 250 cm tartományban, +/-1 cm pontossággal.

NXT változat *EV3 változat*

Az ultrahangos távolságérzékelő ugyanazt a mérési elvet használja, mint a denevérek: a távolságot úgy méri, hogy kiszámolja azt az időt, amely alatt a hanghullám a tárgynak ütközik és visszatér, ugyanúgy,

mint a visszhang. Kemény felületű tárgyak távolságát pontosabban adja vissza, mint a puha felületűekét. Az EV3 változat esetén szoftverkörnyezetben beállítható az un. csendes mód, amelynél nem bocsát ki ultrahangot, csupán figyeli, hogy van-e a környezetében ultrahangforrás.

Gyroszenzor

EV3 változat

Gyakorlatilag egy giroszkóp, amelynek segítségével a robot elfordulási szögét tudjuk megmérni fokokban.

1.4. Input eszközök: egyéb szenzorok

Hangérzékelő

NXT változat

A hangérzékelő méri a hangok intenzitását decibelben (dB) vagy korrigált decibelben (dBA). A decibel a hangnyomás mért értékét fejezi ki. A helyesbített decibelek (dbA) mérésekor a hangérzékelő méri az emberi fül által is hallható hangokat, a standard (helyesbítés nélküli) decibelek érzékelésekor az összes hangot azonos érzékenységgel méri.

Így ezek a hangok tartalmazhatnak olyan összetevőket, melyek az emberi hallás számára túl magasak vagy alacsonyak. Viszonyításként a hangérzékelő maximum 90 dB hangnyomás szintig tud mérni, amely hozzávetőleg egy fűnyíró zajszintjének felel meg. Ennek 4-5%-a egy csendes nappalinak a zajszintje. A programkörnyezet számára egy 0-100 közötti értéket ad vissza.

Iránytű

NXT változat

Az északi iránytól való eltérés szögét adja vissza eredményül fokban (±1 fok pontossággal). Érzékeny a mágneses és elektromos terekre.

Gyorsulásmérő

NXT változat

A szenzor 3 tengely mentén (x, y, z) képes a gyorsulás mérésére -2g és +2g tartományban. Érzékenysége 200 egység/g. Mintavételezési sebessége 100/s.

Infravörös szenzor és jeladó

EV3 változat (érzékelő)

EV3 változat (jeladó)

A szenzor az infravörös szenzor távolságmérésre alkalmas (50-70 cm). A jeladó jelét akár 200 cm-ről is képes érzékelni és meghatározni a helyzetét.
Az NXT változathoz szintén létezik a megfelelője. Alkalmassá teszi a robotot az infra kommunikációra is.

Hőmérsékletmérő

NXT változat

A környezet hőmérsékletét képes mérni és a keretprogram számára visszaadni. A digitális hőmérséklet-érzékelő segítségével mind Celsiusban, mind pedig Fahrenheitben mérthetünk. A mérés értékhatárai: - 20 – + 120 °C vagy - 4 – + 248 °F tartomány.

1.5. Output eszközök: szervomotorok

Az EV3 készlet része két interaktív szervomotor. Nem csak kiviteli eszköz, hanem a beépített érzékelőinek köszönhetően információkat képes visszaadni a keretprogram számára a motor pillanatnyi állapotáról. A beépített forgásérzékelő a motor forgását fokokban vagy teljes fordulatokban méri (±1 fok pontossággal).

További speciális motorként jelent meg a készletben az un. közepes motor (*Medium Motor*), amely specialitása, hogy a forgási tengelye párhuzamos a motor tengelyével.

Szabadon használhatók a második generációs NXT robothoz készült szervo motorok is.

A felsorolás nem teljes. Hosszan lehetne folytatni a robothoz fejlesztett hardver eszközök listáját. Sok olyan átalakító adapter is létezik, amelyet a robothoz illesztve a konstrukció alkalmas lesz más csatlakozási elven működő eszközök kezelésére, vagy akár különböző mérőműszerek helyettesítésére.

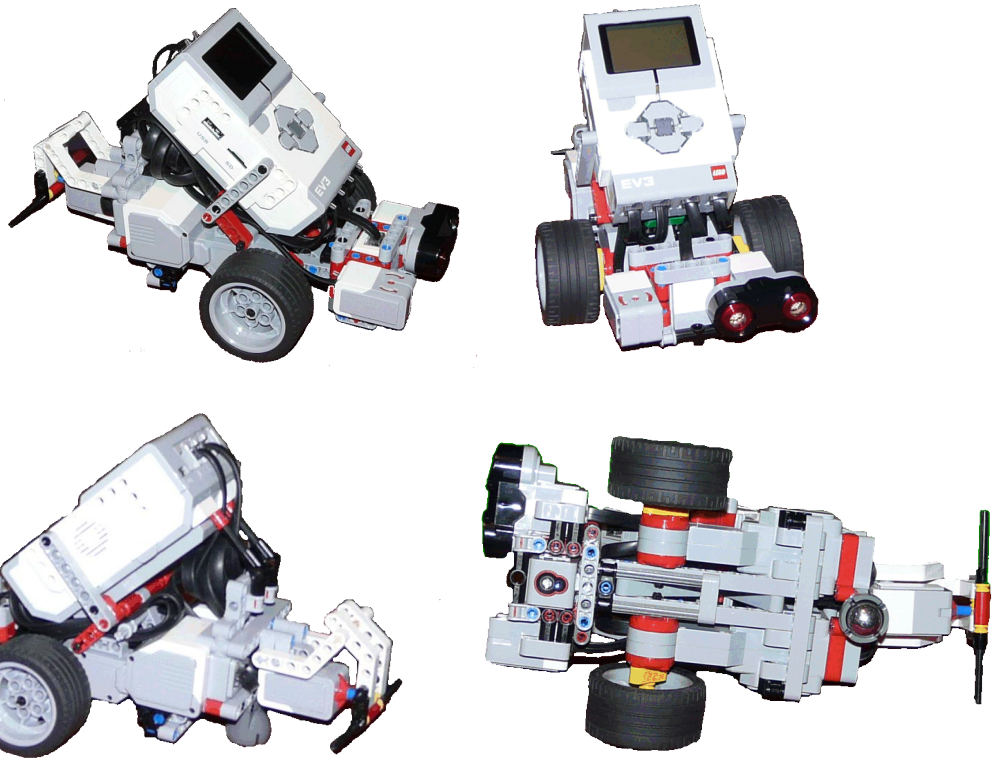
A szenzorok és motorok 6 pólusú RJ12-es (RS485 kábel) csatlakozókábelen keresztül illeszthetők a téglához, amely a gyári készletben különböző hosszméretben találhatók meg.

1.6. A programozás során használt tesztrobot

A következő fejezetekben bemutatjuk a robothoz gyárilag mellékelt programkörnyezet használatát sok példával és néhány feladattal. A bemutatott programnyelv a LEGO és National Instruments közös fejlesztése, amely LabWiev alapokra épül. Ezen kívül sok más programnyelven is lehet a robotot vezérelni. Ezek közül jó néhány ingyenesen hozzáférhető az interneten: pl. leJOS (java alapú) vagy BricxCC és RobotC (C alapú) karakteres programkörnyezetek.
A bemutatott programokhoz használt robot felépítése:

- Két motor csatlakozik a B illetve C portra.
- Egy ütközésérzékelő, amely a robot hátuljára szerelve az 1-es portra csatlakozik.
- Egy giroszenzor, amely a robot 2-es portjára csatlakozik.
- Egy színszenzor a 3-as portra kötve, amely a robot elején található és lefelé irányított helyzetű.
- Valamint egy ultrahangos távolságérzékelő a 4-es portra csatlakoztatva, amely a robot elejére szerelve előre néz.

Minden olyan konstrukció alkalmas a bemutatott programok megvalósítására, amely ennek a felsorolásnak eleget tesz.
A könnyebb érthetőség kedvéért néhány kép az általunk használt robotról:



2. KERETPROGRAM

2.1. Általános bemutatás

A szoftver a LEGO Group és a National Instruments közös fejlesztése, amely a LabVIEW alapokhoz illeszkedően különösebb előképzettség nélkül lehetővé teszi a programozást.
Az EV3 robotkészlet nem tartalmazza a szoftvert, amellyel az összeépített konstrukció programozható, de a http://www.lego.com/en-us/mindstorms/downloads/software/ddsoftwaredownload/ webcímről ingyenesen letölthető a *Home* változat, amely teljes funkcionalitással biztosítja a programok elkészítését. Ugyaninnen a készlethez további szenzorokat kezelő programozási modulok is letölthetők (pl.: ultrahangszenzorhoz, giroszkóphoz, hőmérőhöz, vagy hangszenzorhoz).
Megvásárolható a programozási környezet egy *Education* bővített változata, amely további segédeszközöket tartalmaz és pl. *Teacher* módban is telepíthető. Ez a változat lehetővé teszi komplex oktatási anyagok összeállítását videókkal, forráskódokkal, kommentekkel, ...

A szoftverben egyszerű, zömében egérhasználattal megoldható programírás az ikonok egymás után illesztését és a megfelelő paraméterek beállítását jelenti. A grafikus EV3-G programnyelv használata egyszerű, és vizualitása miatt igen szemléletes. Az egyes hardver elemeket és a legfontosabb programozástechnikai eszközöket egy-egy ikon reprezentálja. Ezeknek az objektumoknak az egymás után fűzéséből vagy akár elágazásokat tartalmazó láncaiból épül fel a program. Megvalósítható segítségükkel nemcsak a lineáris programfutás, hanem a többszálú programozás is.
A programhoz biztosított *Help* (angol nyelvű) részletes leírást tartalmaz az egyes ikonok (programmodulok) felépítéséről és paraméterezési lehetőségeiről, így programozási kézikönyvként jól használható.

Az alábbiakban a keretprogram használatát, legfontosabb funkcióit és a grafikus EV3-G programnyelven készült programok általános szerkezetét mutatjuk be. A fejezetek, *„tutorial"*-ként használhatók, és a kezdeti lépéseket könnyítik meg. Mindenképpen azt javasoljuk a programozást most tanulóknak, hogy a grafikus környezet nehezebb szöveges interpretációja miatt a következő fejezetek anyagát a keretprogrammal együtt, a közölt forráskódok kipróbálásával kövessék végig. A sok grafikus elem és paraméter teljes körű ismertetésére ugyanis nincs lehetőség. Az egyes programozástechnikai elemek használatának megértéséhez szükséges lehet a program *help*-jének használata és a forráskódok kipróbálása.

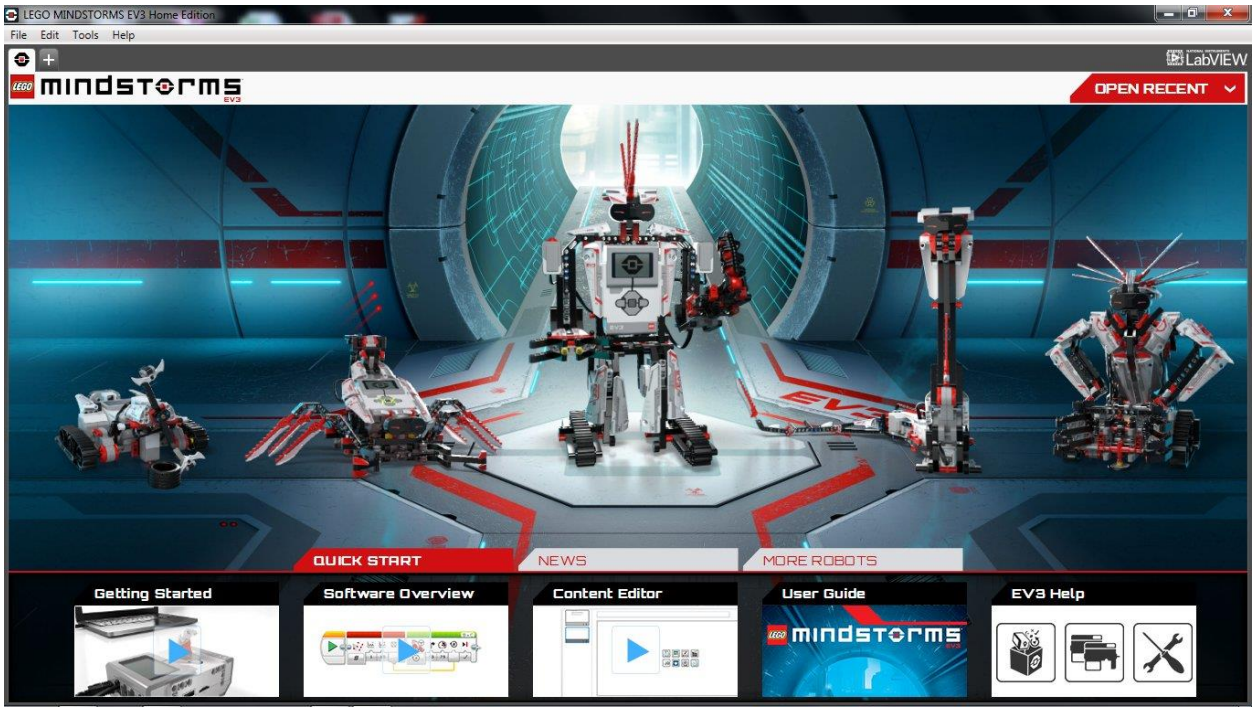
Az EV3-G-hez hasonló moduláris programnyelvek nagymértékben leegyszerűsítik a programírást, hiszen a szintaktikai (pl. gépelési) hibákkal az esetek jelentős részénél nem kell foglalkozni. Az egyes modulok paraméterezése különböző elektronikus űrlapokon használt beviteli eszközök segítségével történik (szövegdoboz, legördülő lista, jelölő négyzet, rádiógomb, stb.). Így az adatbeviteli korlátozások nem engedik meg az értéktartományon kívüli adat beírását. Az ilyen típusú programírásnál a programozási idő jelentős része az algoritmus megalkotására, és nem a gépelésre vagy a szintaktikai hibák javítására fordítódik. Különösen kezdő programozók esetén előnyös mindez, de haladó szinten is a kreatív gondolkodásé a fő szerep. Ezzel a programozási technikával már akár kisgyermekkortól kezdve lehet fejleszteni az algoritmikus gondolkodást, és a programozói kompetenciákat, a karakteralapú programírástól a sokkal kreatívabb, a lényeget jobban megragadó módszerekkel, a globális algoritmusra koncentrálva és nem elveszve a mechanikus szintaktikai részletekben.

A tömör grafikai megvalósítás miatt a program áttekinthető marad, és az egyes programszálak vizuálisan is könnyen követhetők, értelmezhetők. Az EV3-G programnyelv az ikonalapú programelemek miatt bármely programnyelvi specifikáció analógiájaként működhet, és nemcsak ilyen szempontból nyelvfüggetlen, hanem a beszélt kommunikációs nyelvi határok is egyszerűen átléphetők.
A fejlesztőkörnyezet folyamatosan bővíthető, hiszen az internetről letölthetők további modulok, amelyek lehetővé teszik új hardver elemek, programozási eszközök forráskódba építését, vagy az elkészített saját modulok publikálásával, kódba illesztésével a lehetőségek korlátlanná válnak.
A programkörnyezet használatával a programozás oktatás és algoritmikus gondolkodás egy új fejlesztőeszköze jelent meg, amely forradalmasíthatja a programozásról eddig szerzett tapasztalatainkat. Olyanok számára is elérhetővé téve a programozást, robotikát, akik eddig motivációjuk vagy lehetőségeik hiányában a területen kívül maradtak.

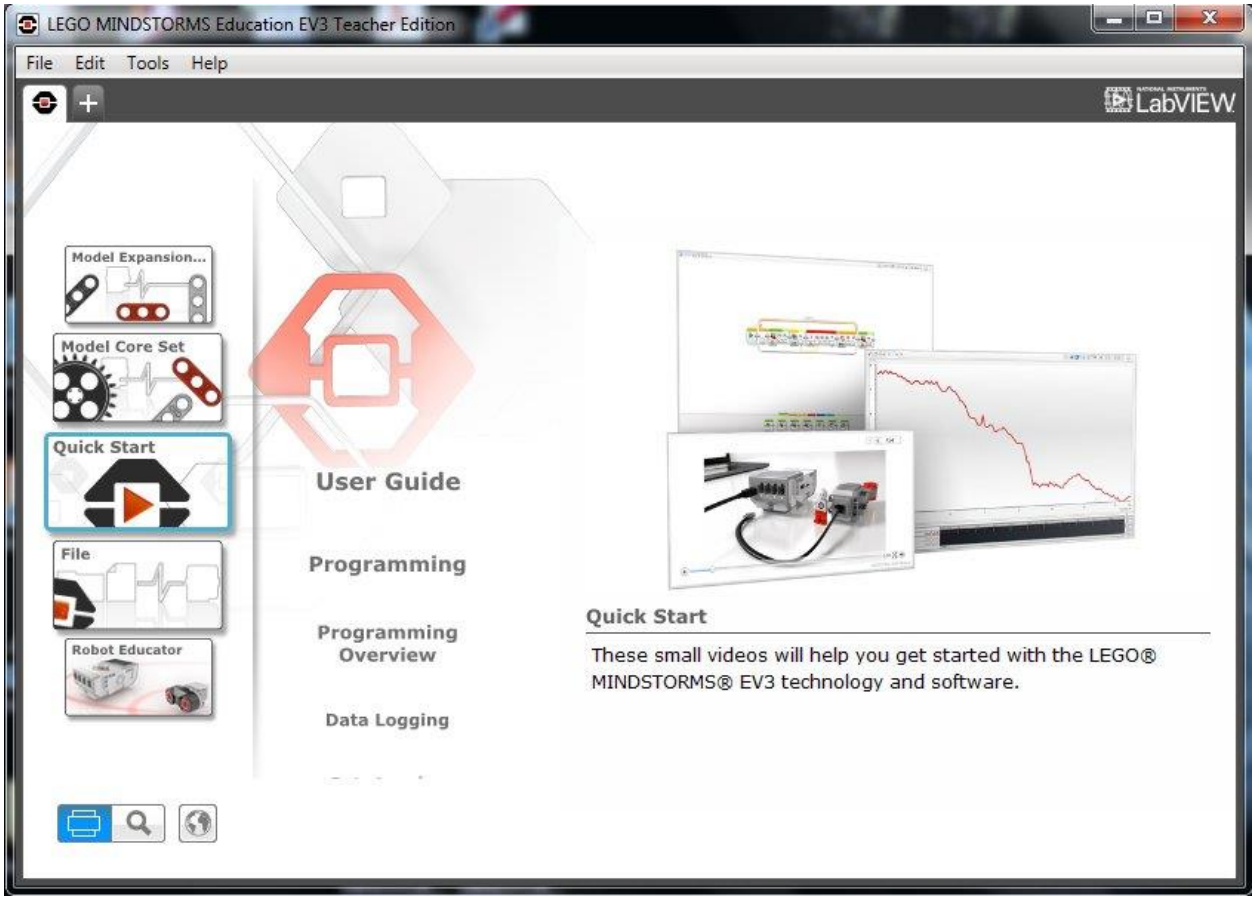
2.2. A használt programváltozat és összetevői

A bemutatott képek a *LEGO® MINDSTORMS® Education EV3 Software Teacher* változatából származó képernyőprintek átszerkesztett változatai, a szoftver 1.0.1-es változatából.
A korábbi NXT1-es illetve NXT2-es készletekhez készült szoftverváltozatokhoz képest jelentős a különbség, mind a vizuális megjelenésben, mind a programmodulok paraméterezésében. A roboton működő saját „operációs rendszer" (*firmware*) több lényeges újítást is tartalmaz, amelyhez a programozói környezet is igazodik. Pl. lehetőségünk van a lebegőpontos számok (tizedes törtek) kezelésére, megjelent a tömb típusú összetett változó. Alapfelszereltségként jelent meg a hardverben a színszenzor és a giro-szenzor.
Az új szoftverváltozat kompatibilis a régebbi NXT téglákkal, ami azt jelenti, hogy az EV3-G programnyelven írt programjaink lefuttathatók az NXT robotokon is. A régebbi típusú robotot csatlakoztatva a számítógéphez az EV3-as szoftver automatikusan felismeri, hogy NXT-ről van szó, így a programozás során néhány blokk nem használható. Pl.: az EV3-nál megjelent közepes motorok vezérlésére létrehozott modul, vagy a képernyő kezelésnél néhány funkció. Egyébként az új szoftverkörnyezet tökéletesen működőképes programokat állít elő az NXT robotokra is. Így néhány funkciótól eltekintve lényegesen bővült az NXT programozási repertoárja is.

A továbbiakban az EV3-as robotszoftvert az EV3-as robottal mutatjuk be.



Az EV3 Home szoftverváltozat nyitó képernyőképe.

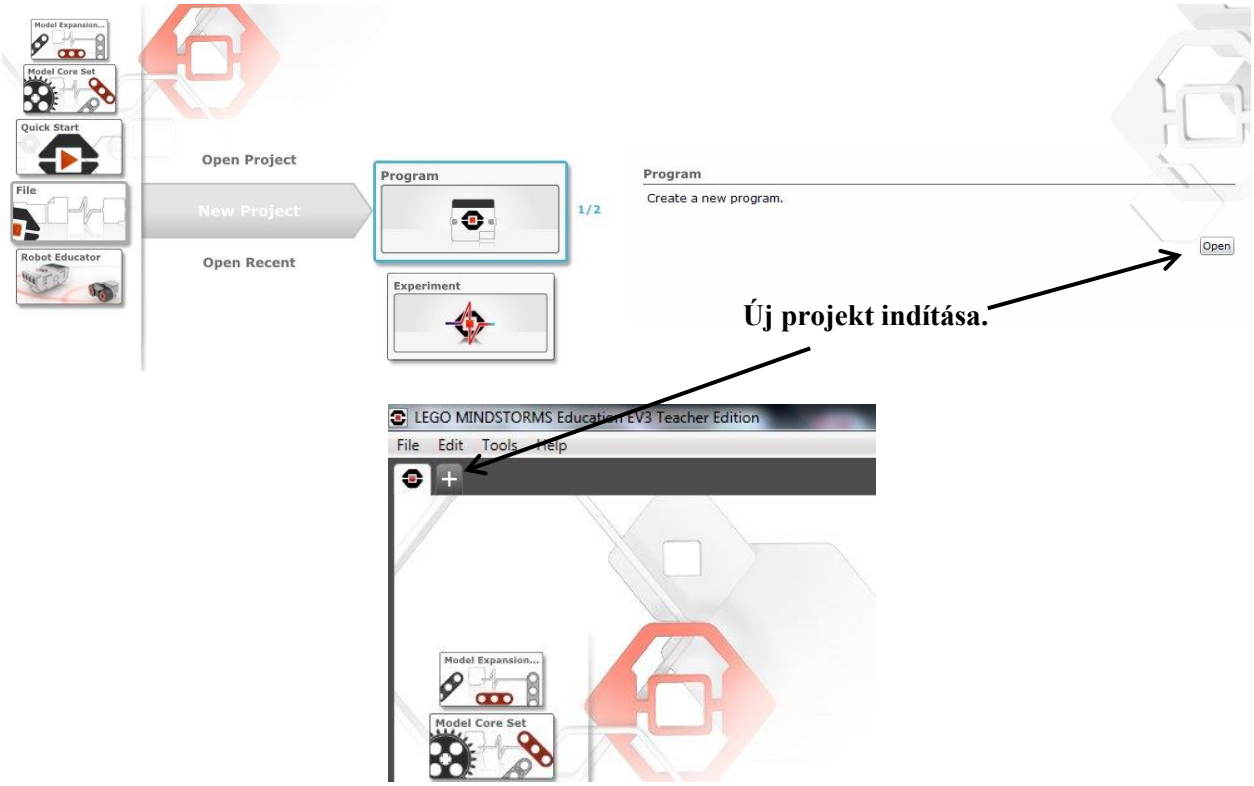


Az EV3 Education Teacher szoftverváltozat nyitó képernyőképe.

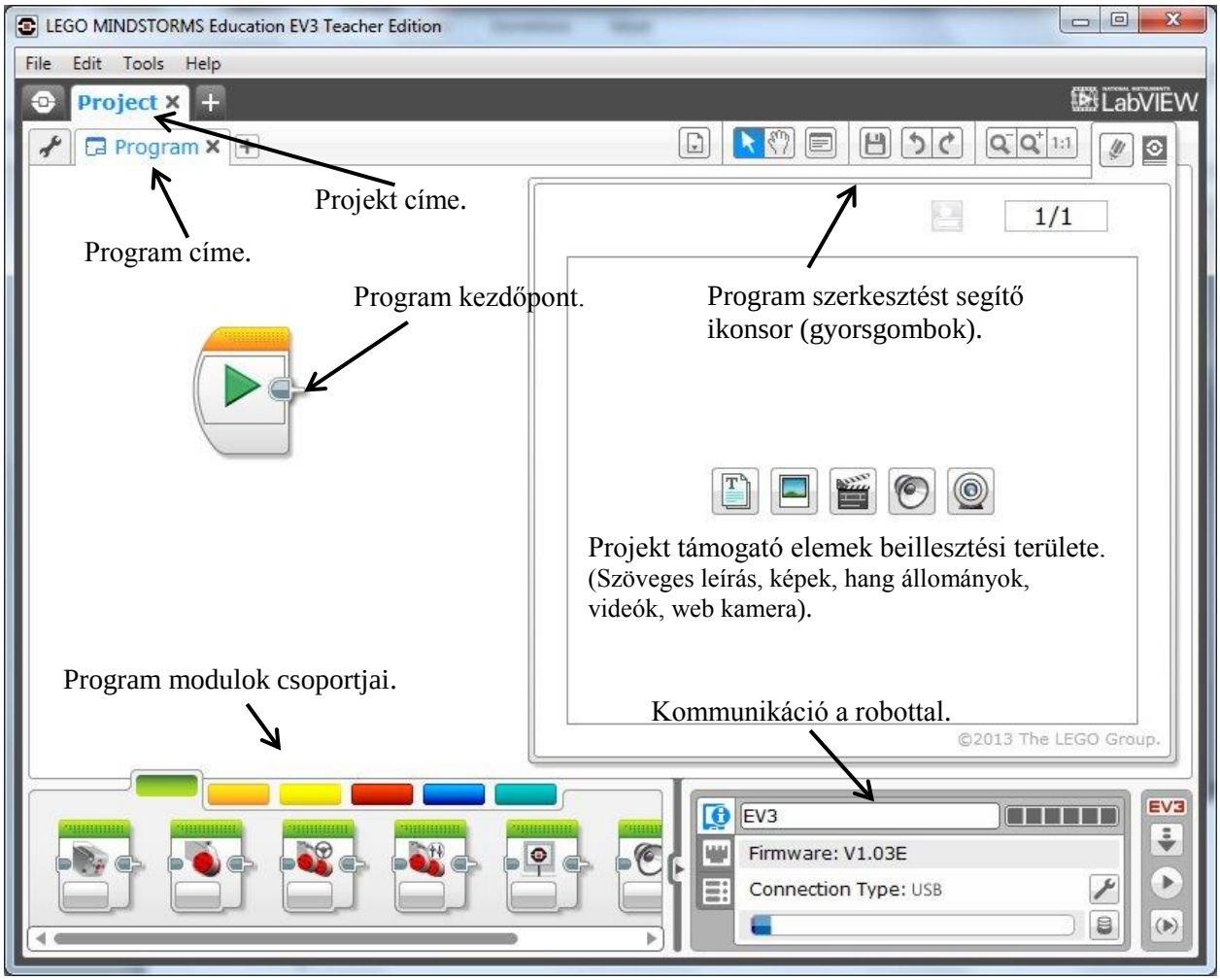
A beillesztett képernyőprintek a *Teacher* szoftverváltozatból származnak.

2.3. A programozási környezet alapelemei

A program indítása után a nyitó képernyőfelülethez jutunk, ahonnan a legfontosabb funkciók ikonalapú menüből választhatók ki. A továbbiakban elsősorban a programírást és az ehhez kapcsolódó funkciók használatát részletezzük. Egy új program írását elkezdeni a *File* menü → *New Project* → *Program* menüpontján keresztül, vagy a képernyő bal felső sarkán látható „+" (*Add Project*) jelen kattintva lehet. Ekkor megjelenik a képernyőn egy új programfelület, *Project* néven, ahol már közvetlenül összeállíthatjuk a programunkat.
A szoftverkörnyezet a szokásos projekt szemléletet támogatja. Tehát programjainkat egy több részből álló komplex projekt keretén belül hozhatjuk létre. A projektbe beilleszthetünk szöveges magyarázatokat, leírásokat, képeket, hangfájlokat, videókat, grafikus program forráskódokat. Mindez együtt alkotja a projektet. Egy projektnek több program is lehet a része, így a logikailag összetartozó programjainkat egy egységbe rendezhetjük. A különböző programjaink forráskódja különálló lapokon helyezhető el. A mentés során egyetlen fájlba kerül be a teljes projekt (*ev3* kiterjesztéssel), de az egyes részei külön-külön kiexportálhatók és más projektekbe beilleszthetők.



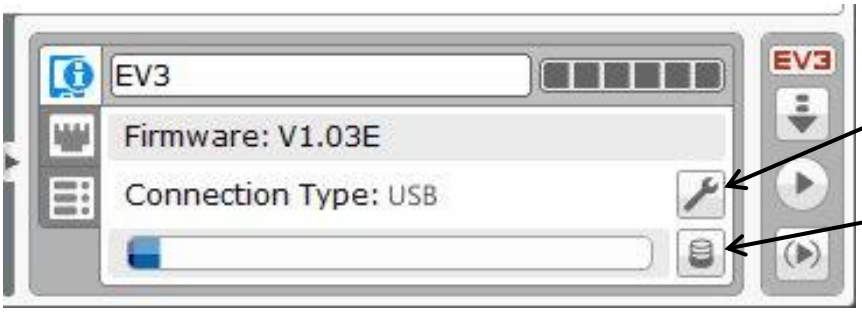
Ha már létező projektet szeretnénk megnyitni, akkor a *File* menü *Open Project…* menüpontját kell választanunk, amelyen keresztül létező *ev3* kiterjesztésű fájlok nyithatók meg.
A nyitó képernyőről elérhető néhány már kész „gyári" projekt, amely tartalmazza az építési útmutatót, a program forráskódját és a működésről készült videó animációt. A nyitó képernyőn találjuk meg a használathoz szükséges gyorstalpalót („*Quick Start*"), amely a programozási eszközök használatának leírását, valamint a szoftverkörnyezet használatát mutatja be, általában szövegesen, képekkel, videó animációkkal támogatva.
Ha egy új program megírásába kezdünk, akkor az ábrán látható felülethez jutunk.



Első lépésként a számítógéphez csatlakoztatott robotot érdemes a rendszerrel megkerestetni. A kapcsolat USB kábelen, bluetooth-on vagy wifi-n keresztül valósítható meg. Ha nincs robot csatlakoztatva a számítógéphez, a program forráskódja akkor is elkészíthető, de a tesztelésre nem lesz lehetőség.

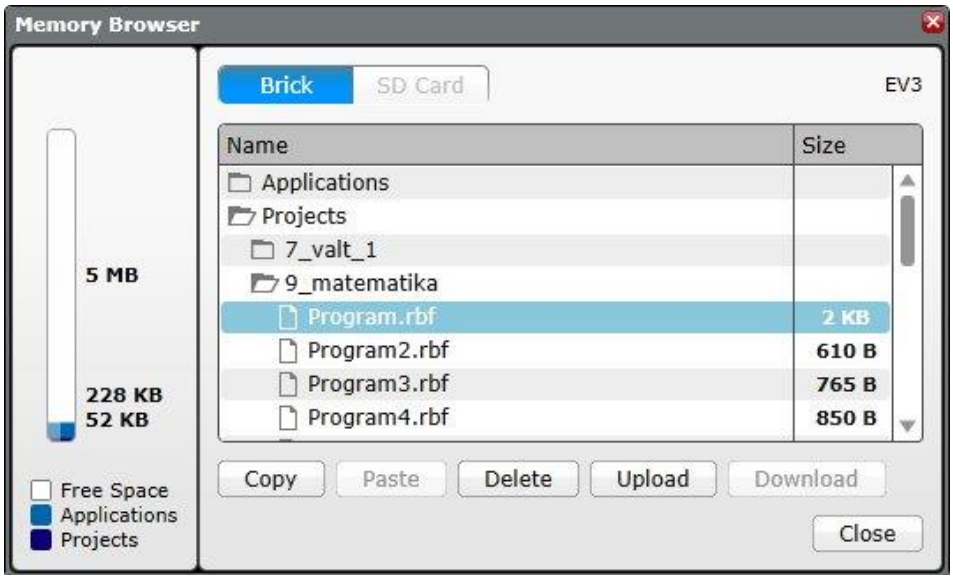
A képernyő jobb alsó részén található a kommunikációs blokk, amely három almenüre tagolódik.

Az első az aktuális kapcsolatot mutatja, kiírva a roboton található *firmware* verziószámát, az akkumulátor töltöttség szintjelzőjét, a robot fantázianevét is (a név itt változtatható meg). Itt található két további ikon, amely a *wireless* beállításokat tartalmazza, valamint a robot memóriájában található programok menedzselő felületére navigál.

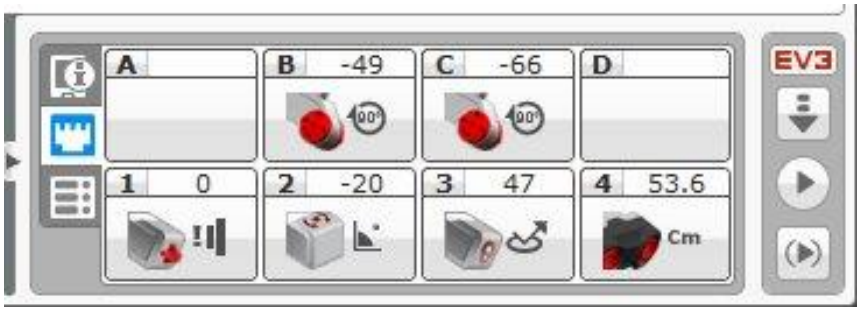


A robotra áttöltött programok, fájlok közül törölni tudjuk a saját programjainkat (*Delete*), vagy visszatölthetjük a számítógépre például a mérési adatokat, képeket tartalmazó fájlokat (*Upload*).

A programok a projekt nevét viselő mappákban kerülnek az EV3-as tégla belső memóriájába. Néhány a rendszer működéséhez szükséges mappa nem törölhető (*Applications*).

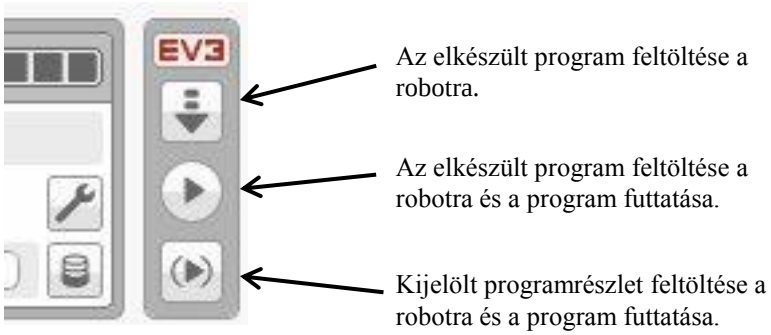


A második menüpontban az aktuálisan csatlakoztatott motorok és szenzorok éppen mért értékeiről kapunk tájékoztatást. A szenzorokat és motorokat automatikusan felismeri a tégla, így azok beállítására nincs szükség. Ha kétszer kattintunk a megfelelő szenzort jelképező területen, akkor beállíthatók a szenzor működési módjának tulajdonságai és a mért érték rögtön leolvashatók a képernyőről.



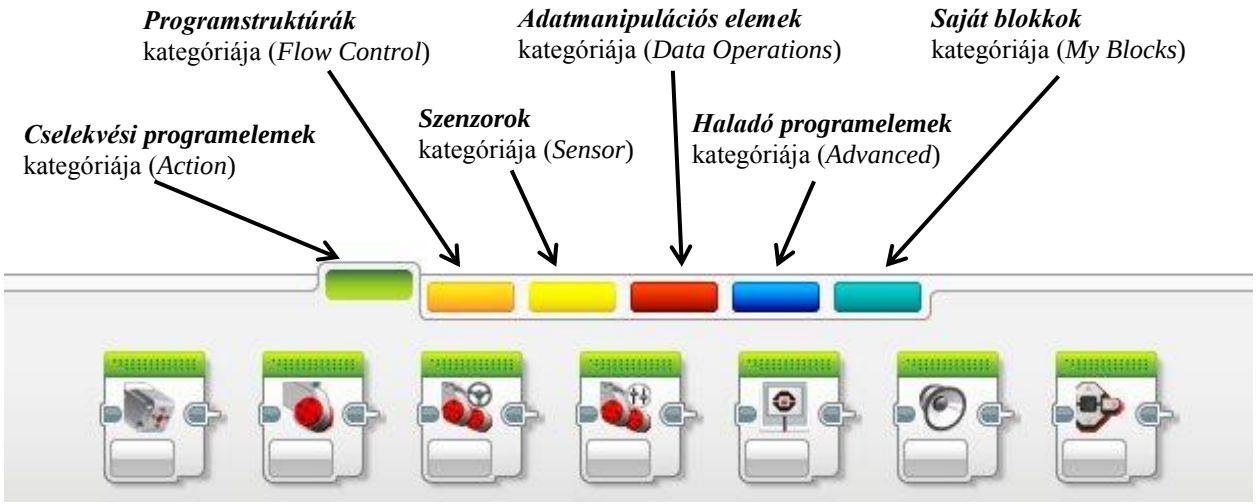
A képen két motor csatlakozik a téglához a B és C portokon keresztül. A csatlakoztatás óta -49°-ot és -66°-ot fordult el a tengelyük. Az egyes portra egy ütközésérzékelő van kötve, amely jelenleg nincs benyomott állapotban. A 2-es porton egy giroszkóp, a 3-as porton *colour* szenzor található, míg a 4-es porton egy ultrahangszenzor. A színszenzor jelenleg fényszenzor üzemmódban, 47% értéket mér.
A harmadik menüpont a számítógép és a tégla közötti kapcsolatról ad felvilágosítást. Jelenlegi élő kapcsolat USB-n keresztüli, az EV3 nevű robottal.

A robot és a számítógép közötti adatcserét megvalósító funkciók a programozói felület jobb alsó sarkában található ikoncsoporton keresztül érhetők el.



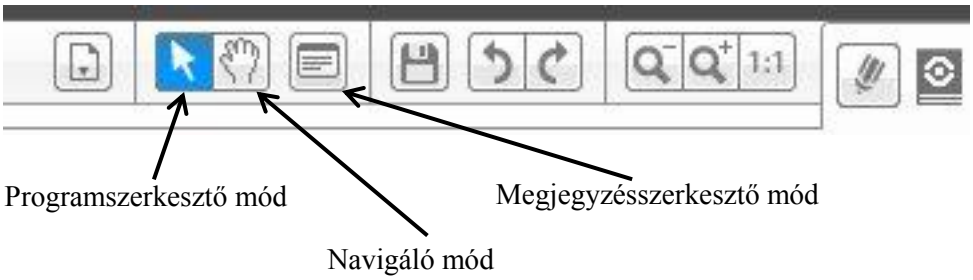
A konkrét programok bemutatása előtt néhány olyan általános felépítésbeli, nevezéktani fogalommal érdemes megismerkedni, amelyek segítik a programozási környezetben való általános tájékozódást és a továbbiak megértését.

A programírás az egyes utasításokat szimbolizáló ikonok egymás után illesztését jelenti. A programikonok funkcióik szerint csoportokba rendezve érhetők el a szoftverkörnyezetben. A képernyő alsó részén tudunk az egyes programcsoportok között navigálni. A megfelelő programcsoportot választva a benne található programikonok, blokkok elérhetők és egérrel a programozási területre húzhatók.

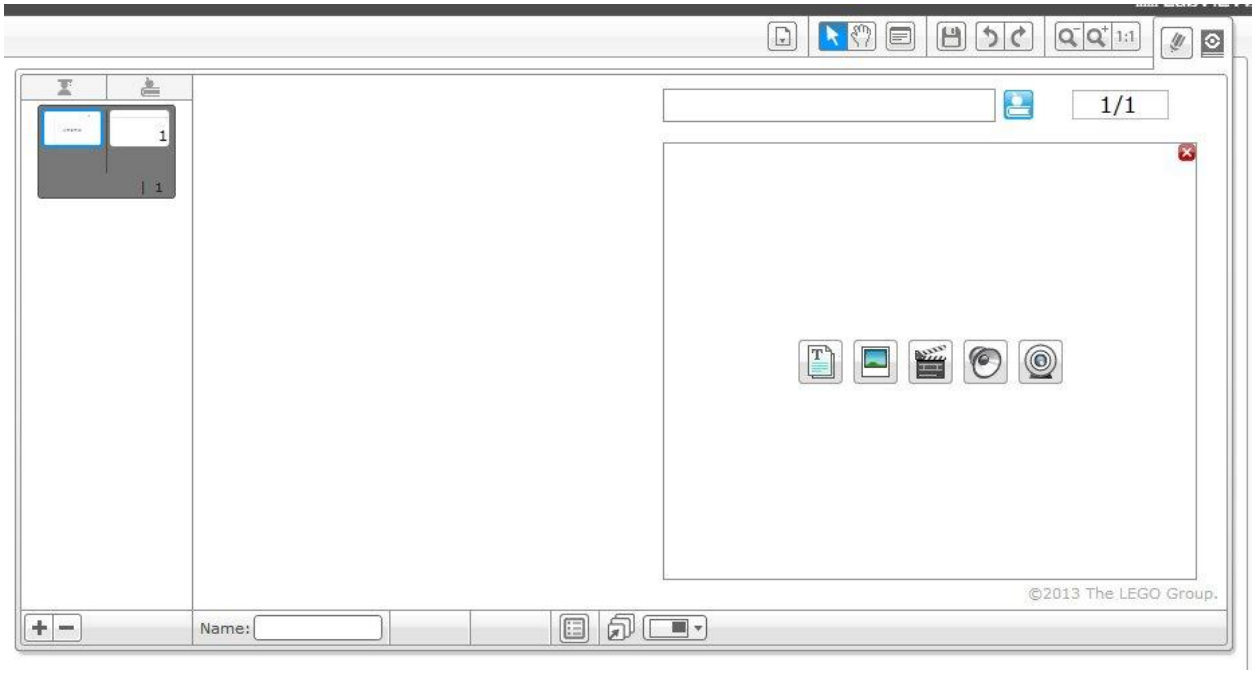


Az egyes programcsoportokat színkódok különböztetik meg. A csoportot jelentő megfelelő színű téglalapra kattintva megjelennek csoportba tartozó ikonok, amelyek közül választani tudunk. Egy-egy ilyen ikon reprezentálja a program utasításait. A programelemekre (menüpontokra) programcsoport néven fogunk hivatkozni, legtöbbször zárójelben megadva az angol elnevezését. A programcsoporton belüli ikonokat sokszor moduloknak vagy blokkoknak fogjuk hívni.

A keretprogramban található hasznos funkciók egyike a szöveges megjegyzések, kommentek elhelyezésének lehetősége. A későbbi programértelmezést nagyban megkönnyíti, ha ilyen címkékkel látjuk el utasításainkat. A szoftver jobb felső részén megjelenő ikonsoron található megfelelő szimbólumra kattintva lehet váltani például a programszerkesztő mód és a megjegyzésszerkesztő mód között.

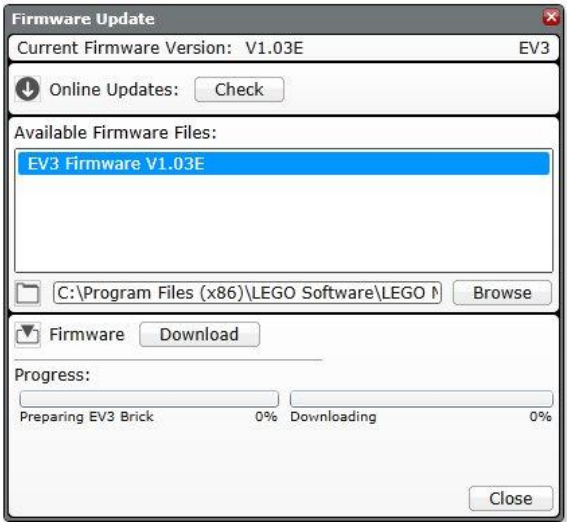


A navigáló mód akkor lehet szükséges, ha a programunk túl nagy méretű, és nem fér el teljes egészében a képernyőn. Ilyenkor a képernyőn nem látható programrészlet a nyíl billentyűkkel vagy navigáló módban kereshető meg, a képernyő görgetésével (egérhúzással). A könnyebb áttekintést segítik az ikonsoron található nagyító ikonok, amelyek segítségével kicsinyíthetjük, nagyíthatjuk, vagy eredeti méretűre állíthatjuk vissza az elkészített program vizuális megjelenítését.
A mentés és visszavonás ikonok szintén az ikonsoron kaptak helyet.
A két utolsó ikonnal a projekthez csatolandó egyéb fájlokat tudjuk feltölteni, vagy szerkeszteni az elkészült munkát.

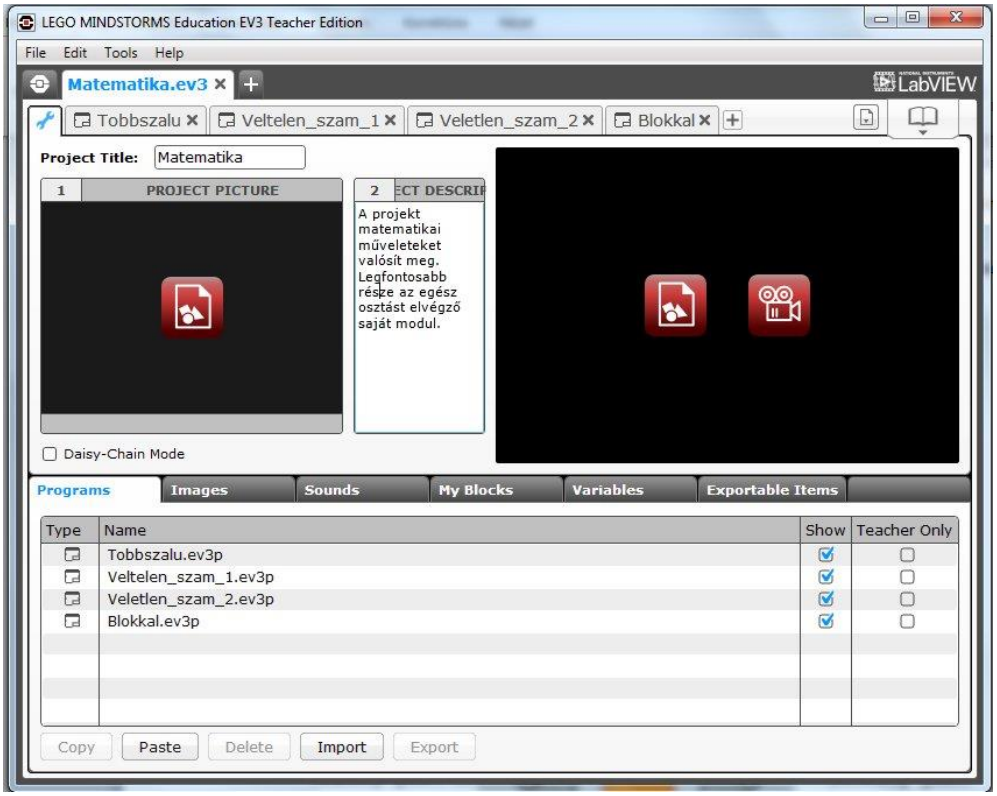


Az elkészült programot a szokásos módon, a *File* menü *Save Project* vagy *Save Project As…* menüpontján keresztül tudjuk menteni és elnevezni.

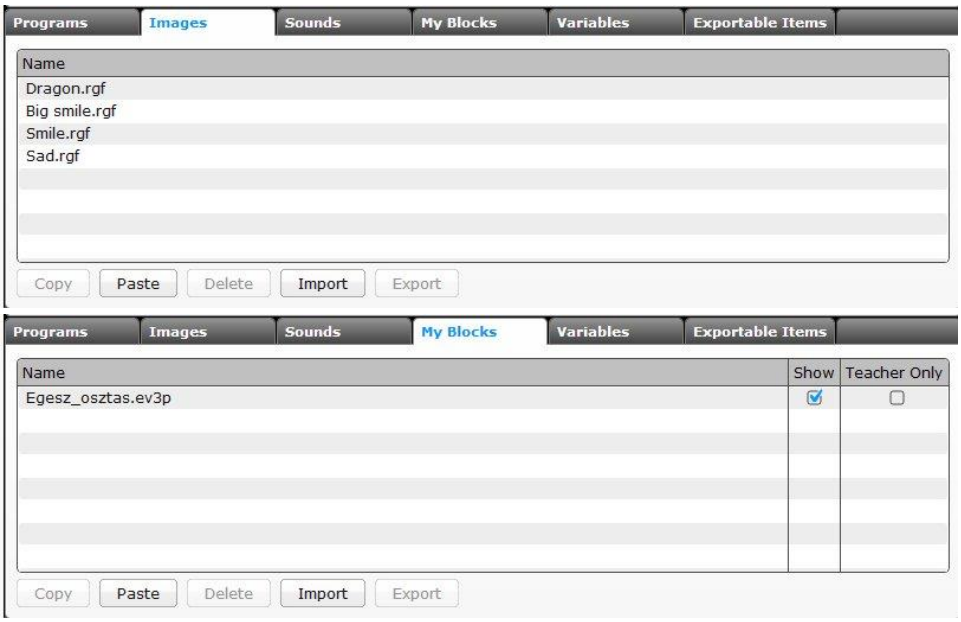
További lehetőségként a *Tools* menüben áll rendelkezésünkre néhány hasznos funkció. Például a téglán elhelyezett *firmware* frissítése (akár interneten keresztül), vagy letöltött új kiegészítő modulok programba importálása.
A *firmware* a robot operációs rendszerének tekinthető. Ez szükséges ahhoz, hogy a robot a megírt, és feltöltött programjaink utasításait megértse, és végre tudja hajtani. Jelen könyv írásakor a *V1.03E firmware* változatot használtuk. Érdemes a legfrissebb *firmware* változatot választani, hiszen ettől függ, hogy milyen utasításokat ért meg a robot és azokat hogyan hajtja végre.



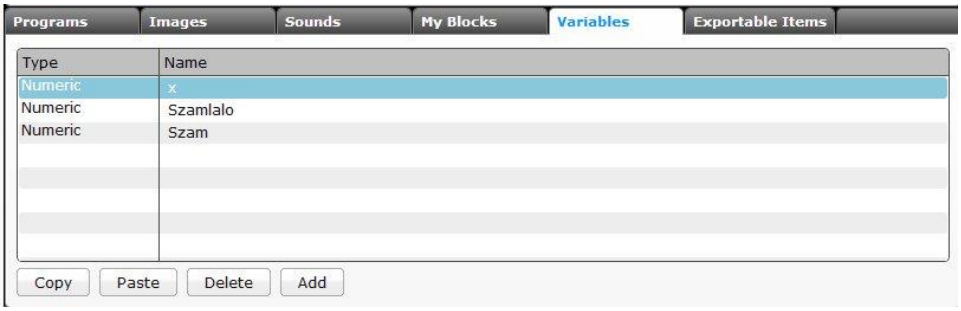
A programozási projekt filozófia egyik legfontosabb előnye, hogy az adott programhoz tartozó fájljainkat együtt tudjuk menedzselni, egyetlen fájlba, egységbe szervezve menteni, visszatölteni. Ez szükségessé teszi, hogy legyen egy olyan felület a szoftverben, ahol át tudjuk tekinteni a projektet alkotó elemeket és az esetleg fölöslegessé váltakat törölni tudjuk, vagy éppen exportálva elérhetővé tegyük más projektek számára.
Az adminisztrációs felülethez a szoftver bal felső szélén látható villáskulcs szimbólumon kattintva juthatunk.



Itt kapunk egy listát a projektben lévő programjainkról, képeinkről, hangállományainkról, saját blokkokról, változókról és az esetleges további exportálható elemekről.
Az egyes lapokat kiválasztva a megjelenő elemeket törölhetjük (*Delete*), exportálhatjuk (*Export*), vágólapra helyezhetjük (*Copy*), vagy éppen beilleszthetjük a vágólap tartalmát (*Paste*).



Nagyon hasznos a változók adminisztrációs felülete. Minden olyan változó látszik, amelyet létrehoztunk a projekt valamelyik programjában. Ha a programot közben töröltük, a változó továbbra is a rendelkezésünkre áll, így azt használhatjuk, vagy külön szükséges törölni. A változók aktualizálása azért fontos, mert ugyanazon a néven nem hozhatunk létre még egy változót.



A projekt menedzselő felületen írhatunk rövid leírást a projekthez, beilleszthetünk képet vagy éppen videót.

A szoftver *Teacher* változatában minden program esetén eldönthetjük, hogy a *Home* vagy *Student* változatot használók láthatják-e a projektben lévő programot. Így komplett oktatási anyagokat összeállítva a diákok csak a nekik szánt feladatrészt láthatják, mégha a projekt tartalmazza is a teljes változatot.

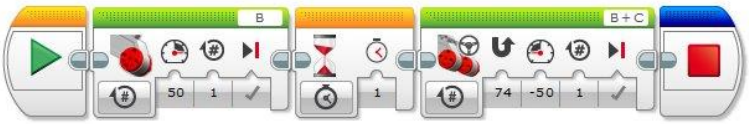
Mindezzel a projektjeink aprólékosan menedzselhetők és komplex oktatási anyagok állíthatók össze.

2.4. Programírás, első lépések

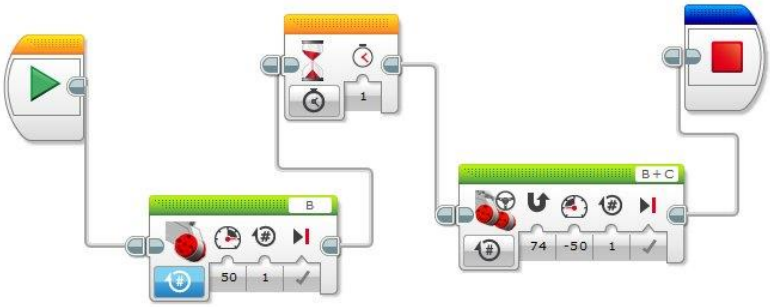
A programírás (bármilyen programnyelven) azt jelenti, hogy utasítások sorozatát készítjük el valamilyen szövegszerkesztő programmal (ezt hívják forráskódnak). Ezt az utasítássort fordítja le egy alkalmas program a számítógép nyelvére, majd a lefordított programot futtatva sorban végrehajtódnak a benne található utasítások.

Az EV3-as robothoz készült programnyelv ettől a folyamattól abban tér el, hogy az utasítások sorozatát ikonok helyettesítik. Ezeket az ikonokat egérrel mozgathatjuk a programban a megfelelő helyre. Egy-egy ilyen ikon/modul hatására fog a robot mozogni vagy megmérni egy akadály távolságát maga előtt. Az ikonokat a végrehajtásuk sorrendjében kell „felfűzni" balról jobbra haladva, a programozási felületen látható start ikontól kezdve (zöld háromszög szimbólum). Az ikonok, és ezen keresztül az

utasítások egymásutániságát, összekötését kábelszimbólumok mutatják, amelyek alapértelmezésben nem látszanak. Ha a két ikont összekötő csatlakozási pontra kattintunk, akkor a két ikon közötti távolság megnő és láthatóvá, mozgathatóvá válik az összekötő kábel. Így a képernyőn tetszőleges pozícióba helyezhetők az egyes blokkok. A végrehajtási sorrendet a kábellel történt összekötés sorrendje határozza meg.

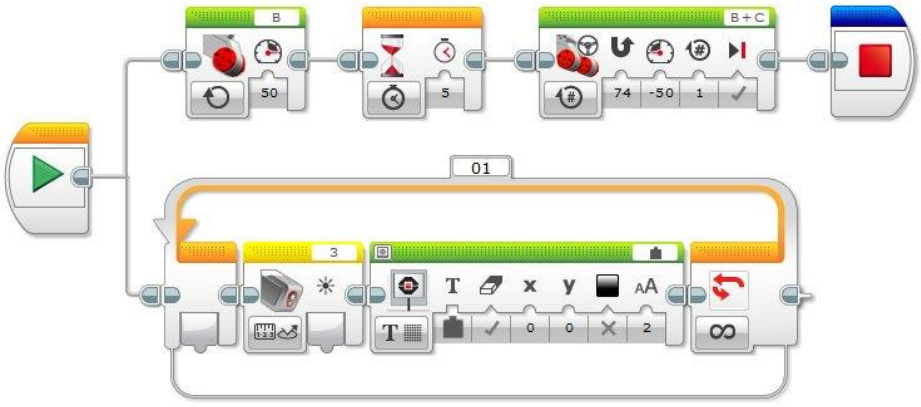


Egymáshoz illesztett utasítássor. Balról jobbra hajtja végre a rendszer az utasításokat.



Kábellel összekapcsolt utasítássor. A végrehajtás sorrendjét a kábelezés határozza meg.

Lehetőségünk van arra is, hogy az utasítás-végrehajtás ne lineáris legyen. Ilyen esetben a program látszólag több szálon fut, egymással párhuzamosan hajtódnak végre a parancsaink.
A párhuzamos szálakat bármely két blokk között kezdeményezhetjük, de két blokk csak egyetlen kábellel köthető össze, tehát a program nem tartalmazhat hurkokat.



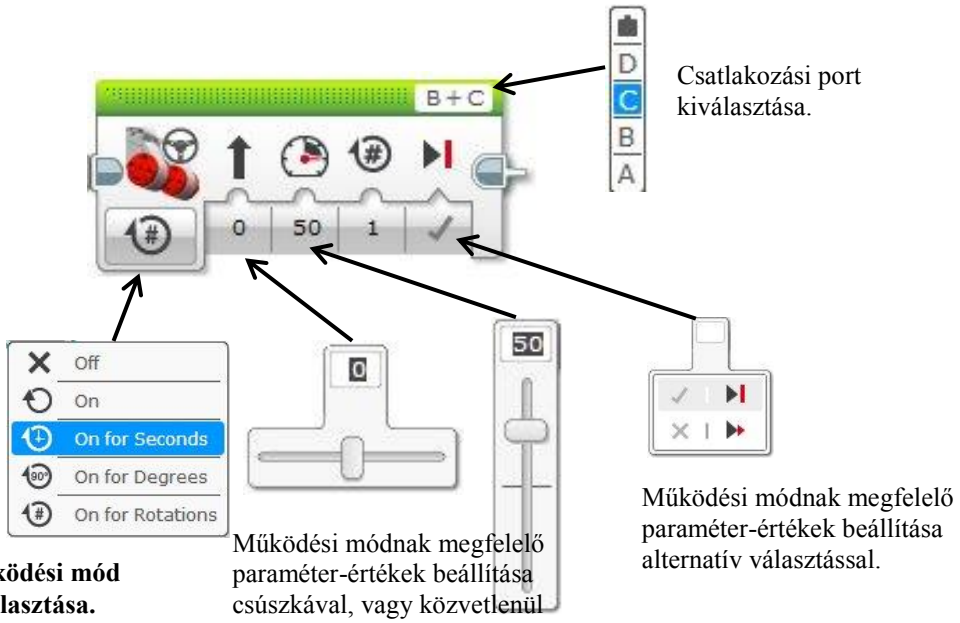
Párhuzamos utasítás-végrehajtás. A program két szálon fut.

A legtöbb ikonnak van egy paraméterlistája, amely értékei az ikonon elhelyezett beviteli objektumok segítségével állíthatók be. Itt adhatjuk meg az ikonnal szimbolizált utasítás tulajdonságait. (Pl.: milyen gyorsan forogjon a motor, mennyi ideig működjön, milyen mértékegységben szeretnénk távolságot mérni, stb.) A paraméterlistán kétféle szimbólum fordulhat elő: beviteli és kiviteli értékek. Ezek vizuális megjelenésüket tekintve is különböznek egymástól.

Beviteli paraméterek vizuális megjelenése:

Kiviteli paraméterek vizuális megjelenése:

A paraméterek az ikon alsó részén láthatók és a beviteli paraméterekbe értéket is írhatunk: közvetlenül a szövegdobozba történő írással, vagy listából történő választással, esetleg valamilyen csúszka használatával. A kiviteli paraméterekbe a felhasználó nem tud értéket beírni, ezeket az adott eszköz „tölti" meg tartalommal. Minderről a Paraméterátadás fejezetben lesz bővebben szó.
Az ikonon látható paraméterlista attól függ, hogy milyen működési módot választottunk az adott blokkal szimbolizált eszköznek. A működési mód beállítása mindig az ikon bal alsó sarkán lévő szimbólumon történő kattintással állítható be, egy listából történő választással. Először ezt kell kiválasztani. Ennek megfelelően az ikon alsó részén megjelennek a lehetséges paraméterek, amelyek értékeit beállíthatjuk.
Az egyes hardver eszközök a tégla portjaira csatlakoznak. A-tól D-ig a motorok, míg 1-től 4-ig a szenzorok. A helyes működés szempontjából fontos megadni, hogy melyik portra milyen eszközt csatlakoztattunk. Ezt a szoftver automatikusan felismeri, ha a robot csatlakoztatva van a számítógéphez. Előfordulhat azonban, hogy meg kell változtatni a port beállítását. Ezt, az utasításblokk jobb felső részén található szimbólumra kattintva tehetjük meg. A portok listája minden esetben a szimbólummal kezdődik (*Wired*) ezt választva a paraméter értékét egy másik blokk által szolgáltatott érték fogja meghatározni. Ennek technikai kivitelezéséről a Paraméterátadás fejezetben lesz szó.
Valamennyi blokk esetén használhatók a felsorolt funkciók, de természetesen az utasításnak megfelelően különböző tartalommal. Az alábbi ábra a motorvezérlő utasításblokk esetén mutatja be a lehetőségeket.



Csatlakozási port kiválasztása.

Működési mód kiválasztása.

Működési módnak megfelelő paraméter-értékek beállítása csúszkával, vagy közvetlenül a szövegdobozba írva.

Működési módnak megfelelő paraméter-értékek beállítása alternatív választással.

A számítógépen tehát a programírás az egyes blokkok egymáshoz fűzését, és minden blokk esetén a megfelelő működési mód és paraméterek beállítását jelenti. Az így összeállított programot kell a robotra feltölteni. Ha jól állítottuk össze a program utasításait, akkor a működés során megoldja a kitűzött feladatot.
A feltöltés elindítása után megtörténik a program „robotnyelvre" fordítása, így az utasítások értelmezhetők lesznek a robot számára is. A robotra már a lefordított program kerül. A feltöltés történhet kábeles, wifi-s vagy bluetooth-os kapcsolaton keresztül. A feltöltés után már nincs szükség a robot és számítógép összekapcsolására, a robot önállóan hajtja végre a lefordított utasításokat (a kapcsolatot nem kell megszakítani, ha a kábel nem zavarja a robot mozgását). A robotra feltöltött és lefordított programok nem tölthetők vissza értelmezhető módon a számítógépre.
A programírás tehát még egyszer összefoglalva azt jelenti, hogy:

- a kitűzött feladatot (a robottól várt viselkedést) lebontjuk egyszerű utasítások sorozatára, amelyek eredményeként a robot a kívánt módon fog viselkedni,
- az utasításoknak megfelelő ikonokat egymás után „felfűzzük" a programszálra,
- minden utasításnál beállítjuk a megfelelő paraméterértékeket.

Ha elkészültünk, a programunkat feltöltjük a robotra és ott elindítjuk.
Ha a robot nem úgy viselkedik, ahogy elterveztük, ennek általában nem a robot az oka, hanem valószínűleg valamilyen programtervezési hibát vétettünk, vagy hardvert rosszul állítottuk össze.
A következő fejezetekben az egyes utasításokat, ikonokat mutatjuk be sok példán keresztül, de a teljesség igénye nélkül.
A szoftverben szereplő programutasítások, ikonok tovább bővíthetők például az internetről letölthető modulokkal. Egy letöltött modult a *Tools* menü *Block Import Wizard…* menüponton keresztül tudunk a szoftverbe építeni.

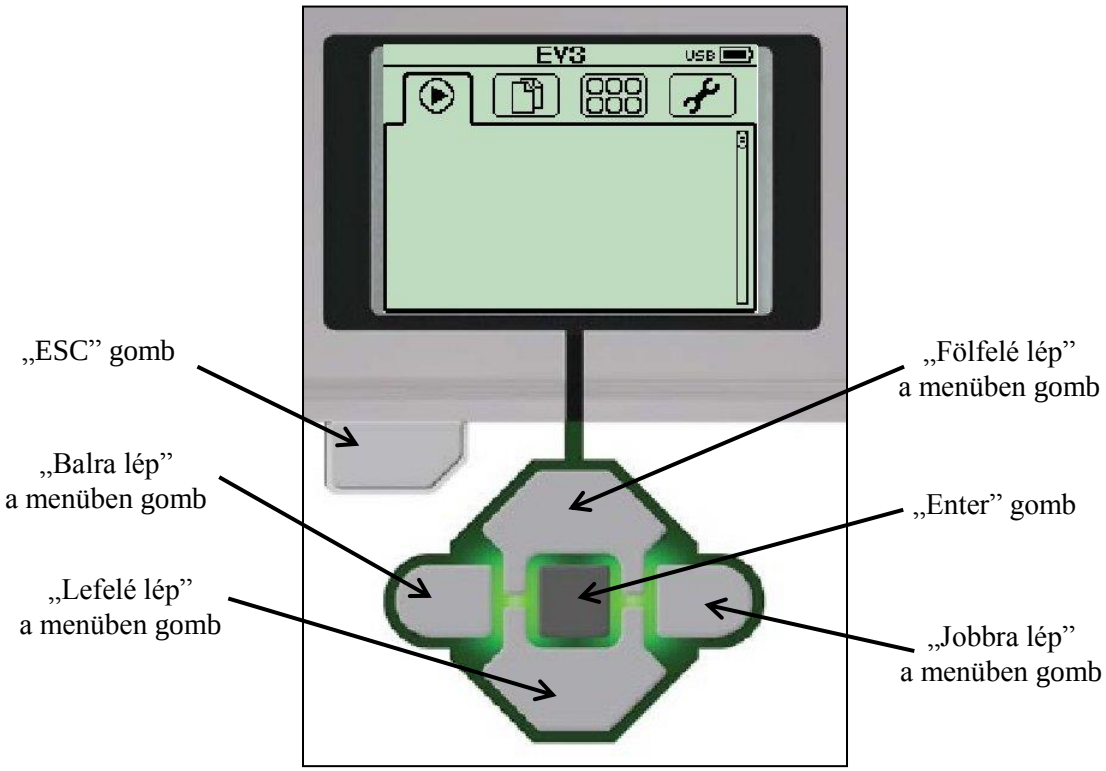
A további fejezetek közül az első néhány egyszerű programokat és programozási ötleteket tartalmaz, de a könyv vége felé már komolyabb tudást igénylő feladatok is szerepelnek. Az első néhány fejezetben a programokhoz alaposabb magyarázat tartozik. Bemutatjuk az egyes paraméterek részletes beállítási értékeit, de később a programozói tudás és a rutin növekedésével már csak utalunk ezekre, és csak a szokatlan, egyedi vagy az új beállításokat részletezzük.

Nem írható olyan programozói könyv, amelyben az első betűtől az utolsóig haladva lineáris rendszerben tanulható meg a programozás. Az egyes komplexebb feladatoknál előfordulhatnak olyan elemek, amelyek későbbi fejezetben szerepelnek részletesen. Ezért a könyv egyes bonyolultabb feladatainál előfordulhat, hogy előre kell lapozni. Javasoljuk, hogy a bemutatott példákat próbálják ki és a paraméterek megváltoztatásával többször is teszteljék. Így lehet olyan programozói tudást szerezni, amellyel már önálló, kreatív programfejlesztés is végezhető.

Az elkészült könyv nem törekedett a teljességre. Néhány fontos programozási elemmel nem is foglalkoztunk. A cél a programozási alapok és lehetőségek bemutatása volt, sok olyan algoritmus ötlettel, amelyek megértése lehetővé teszi a kreatív programozást és rutinos szoftverhasználatot.

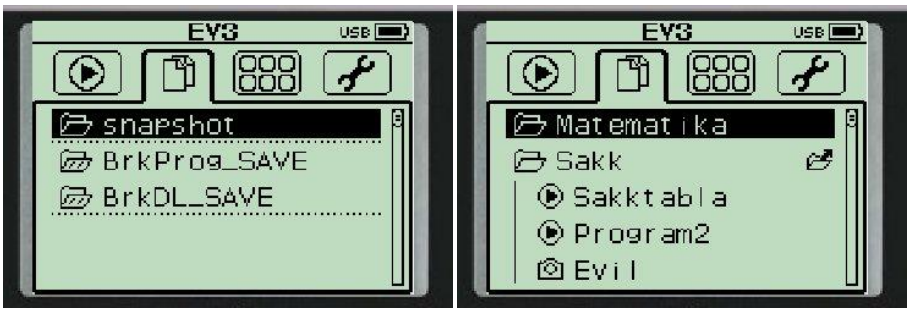
3. A ROBOT KÉPERNYŐMENÜJE

A programírás megkezdése előtt érdemes megismerkednünk a roboton lévő *firmware* által biztosított menürendszerrel és néhány funkciójával.
A tégla bekapcsolása után egy menürendszerhez jutunk, amelyben a roboton elhelyezett hat nyomógomb segítségével lehet navigálni.



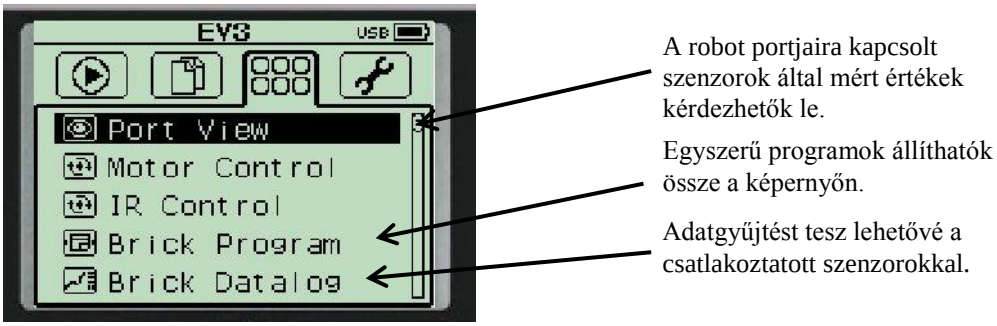
A képernyőn négy vízszintesen elhelyezett menüpont közül választhatunk.
A bal oldali, a robotra töltött és már futtatott programok listáját tartalmazza. Innen egyszerűen kiválasztva a szükséges programot újra elindíthatjuk.

A második menüpont tartalmazza a robotra töltött teljes projektlistát, valamennyi állományával együtt. Kezdetben, a *firmware* feltöltése után, mikor saját programot még nem töltöttünk a robotra, akkor is van néhány alapértelmezett állomány a téglán. Ezek a rendszer működéséhez szükségesek, így nem törölhetők, vagy nem érdemes törölni őket. Később a saját programjaink is itt jelennek meg, mappákba rendezve, a projektek neveivel, mint mappanevekkel. A listából kiválasztva a megfelelő programot, az elindítható az „*Enter*” gomb megnyomásával.

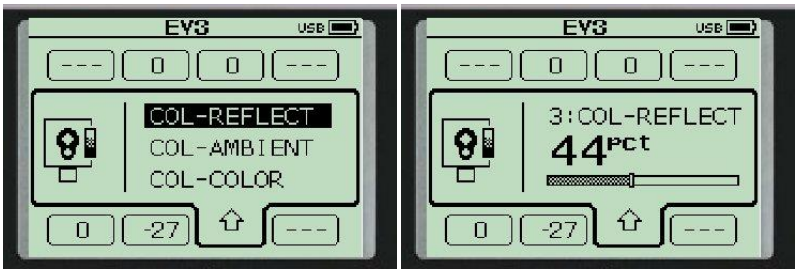


A Projektek mappaszerkezete a tégla képernyőmenüjében (alapértelmezett és saját fájlok).

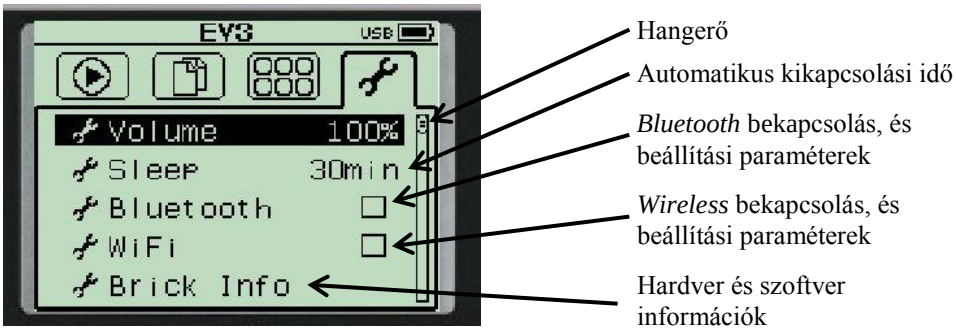
A képernyőmenü harmadik menüpontja alatt több funkció közül is választhatunk, amelyek a csatlakoztatott szenzorok és motorok kezelését segítik.



A szenzorok által mért értékek a szoftverben is megjelennek, de ehhez a téglát csatlakoztatni kell a számítógéphez. Ha erre nincs lehetőségünk, akkor közvetlenül a képernyőmenün keresztül is lekérdezhetjük a négy bementi porton az aktuálisan csatlakoztatott szenzorokkal mért értéket. A szenzorok felismerése automatikusan történik, azok típusát nem szükséges beállítani, de a működési módok közül választhatunk. A képeken a *Colour* szenzor vörös fénnyel történő megvilágítási módban mért értéke látható.



A képernyőmenü negyedik menüpontja elsősorban a hardver beállításokat és információkat tartalmazza.



4. EGYSZERŰ MOZGÁSOK

4.1. Motorok vezérlése

A robot a vezérlőegységhez kapcsolt motorok segítségével valósítja meg a különböző mozgásokat. A robothoz négy motor csatlakoztatható, melyek csatlakozási helyeit A, B, C és D betűkkel jelölték a téglán.
A készletben két különböző felépítésű motor található. A közepes motor forgási tengelye párhuzamos a motor hosszanti tengelyével (*Medium Motor*), míg a nagy motor forgási tengelye merőleges a hosszanti tengelyre (*Large Motor*).

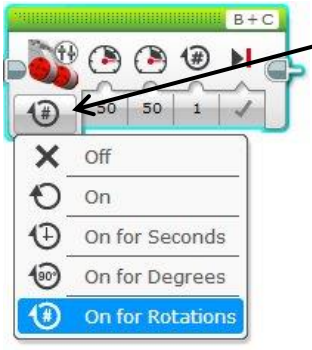
Közepes motor *Nagy motor*

A motorok vezérlésére négy modul is alkalmas. Ezek a *Action* kategóriában található *Large Motor*, *Move Steering* és a *Move Tank* modulok, valamint a speciális közepes motorhoz tartozó *Medium Motor*.

Nagy motor blokk (*Large Motor*) | *Kormányvezérelt motor blokk* (*Move Steering*) | *Sebességvezérelt motor blokk* (*Move Tank*)

A motorok irányítása, és ezeken keresztül a robotkonstrukció mozgatása különböző paraméterek beállítását jelenti. A motorok esetén beállítható:
- Működési mód: Azt jelenti, hogy a motor működésének időtartamát mi szabályozza. Ez lehet idő (másodperc), elfordulási szög (a tengely fordulási szöge fokban), tengelyfordulatok száma, illetve a motort lehet be illetve kikapcsolni.

A működési mód beállítását az ikon bal alsó sarkán lévő módválasztó ikonnal lehet elvégezni.



Működési mód választó ikon.

Működési módok:

Motor kikapcsolása
Motor bekapcsolása
Motor működése a beállított másodperc értékig.
Motor működése a beállított tengelyelfordulási szögig.
Motor működése a beállított tengely körülfordulási számig.

A kiválasztott működési módnak megfelelően az ikon alsó részén, a paraméterlistán csak azok a beállítási lehetőségek jelennek meg, amelyek aktuálisan érvényesek.

	A motor forgási sebessége állítható be. –100; +100 közötti értékkel. A negatív érték ellenkező irányú forgatást jelent. Az érték megadható egy csúszkával, vagy a szám beírásával a szövegmezőbe.
	Csak a kormányvezérelt motor blokk (*Steering Motor*) esetén használható. A kormányzás úgy oldható meg, hogy a két motort különböző sebességgel működtetjük. Ekkor a robot a lassabban működő motor irányába elfordul. Ezt teszi lehetővé a *Steering* paraméter. A paraméter értékét a mellette lévő csúszka segítségével állíthatjuk. Ha a csúszka középen van, akkor mindkét motor azonos sebességgel és ugyanabba az irányba forog. Ha a csúszkát eltoljuk az egyik motor irányába, akkor az a motor gyorsabban forog, és a robot nagy ívben elfordul. Ez a paraméter csak két motor vezérlése esetén használható. Ha a paraméter értéke –50 > és <+50 közötti, akkor mindkét motor ugyanabba az irányba forog, de az egyik gyorsabban, míg a másik lassabban, ezáltal a robot íven kanyarodik. –50 és +50-es értéknél az egyik motor áll, a másik pedig forog. Ha a beállított érték kisebb, mint –50 vagy nagyobb, mint +50, akkor a két motor ellentétes irányba forog eltérő sebességgel, míg –100 és +100-as értéknél ellentétes irányba forognak azonos sebességgel, így a robot helyben fordul. A forgási sebesség a gyorsabban forgó motornál annál nagyobb, minél nagyobb abszolút értékű a beállított érték.
	Ha az időtartammal vezérelt működési módot választjuk, akkor az időtartam állítható be másodperc mértékegységben. Ennyi ideig fog a motor működni. Tizedes tört is használható, így a működés időtartama precízebben vezérelhető.
	Ha a tengelyfordulási szög a működési mód, akkor fok egységben lehet megadni az elfordulás szögét. Tizedes törtek használhatók. Nem a robot fordulási szögét adjuk meg, hanem a motor tengelyének elfordulását.
	Ha a tengelyfordulatok száma a kiválasztott működési mód, akkor körbefordulások számát lehet megadni. Tizedes törtek használhatók.
	A motor működésének befejeztével a megállás módja választható ki. A *Break* választása esetén a motor, és egyben a robot is, blokkolva áll le. A *Coast* választása esetén a motorok kikapcsolnak, de a robot nem fékeződik le. Fokozatosan lassulva áll meg.

A paraméter számértékének megadása történhet az aktuálisan látszó alapértelmezett értékre kattintva és a szövegdobozba beírva, vagy a legtöbb esetben a megjelenő csúszkán vagy listaelemen kattintva is. További lehetőség az érték megadására a paraméterátadás, amely valamely másik blokk által visszaadott érték alapján történhet (lásd paraméterátadással foglalkozó fejezet).

A három motor modul közötti különbségek:
- A *Large Motor* ikonnal egyetlen motor irányítható, amely portját a jobb felső sarokban lehet beállítani.
- A *Move Steering* ikonnal két motor irányítható. Egyetlen sebességparamétert lehet megadni, így a fordulást a két motor közötti sebesség elosztásával lehet szabályozni.
- A *Move Tank* esetén az irányítás a tankok vezérléséhez hasonló: a két motor eltérő sebességű forgatása okozza a kanyarodást. Mintha két botkormánnyal vezérelnénk a motorokat. Mindkét motornak külön-külön állatható a sebessége, és ezáltal finomabban hangolható a mozgása és gyorsabb mozgást, forgást is eredményezhet.

A *Medium Motor* beállításai megegyeznek a másik három moduléval, de csak egyetlen motor vezérelhető egyszerre.

A működési módok közül az utolsó hármat választva (másodperc, elfordulási szög vagy tengelyfordulat) a motor bekapcsol és a beállított érték teljesüléséig működik. Ezalatt a program további utasításai nem hajtódnak végre mindaddig, amíg a motor modulon beállított értéke nem teljesül. Ekkor a motor leáll és a következő utasítással folytatódik a program végrehajtása.
A motor szimpla bekapcsolásával (*On* működési mód), a programszál utasításainak végrehajtása folytatódik, miután a motor elindult. Ebben az esetben a motor leállításáról külön utasításblokkal kell gondoskodni, vagy a program befejeződése automatikusan eredményezi ezt.
Ha tehát nem tudjuk előre, hogy mennyi ideig kell a motort működtetni, akkor a bekapcsolás működési módot kell választanunk (*On*), de ilyen esetben általában valamilyen szenzor által mért érték fogja a leállítást megvalósítani. Ha tudjuk, hogy mennyi ideig kell a motoroknak működnie, akkor arra kell figyelemmel lennünk, hogy, amíg ez az idő le nem telt, addig a robot nem lép tovább az utasítássoron, így a további utasításokat nem vizsgálja, és nem hajtja végre, tehát például a későbbi szenzorokat sem figyeli. Ha a szenzorokat is figyelni szeretnénk és a motort fix ideig működtetni, akkor az egyik lehetőség lehet a több szálú programozás (lásd később).

4/P1. Írjon programot, amelyet végrehajtva a robot 50-es sebességgel előre halad 500°-os tengelyfordulásig!

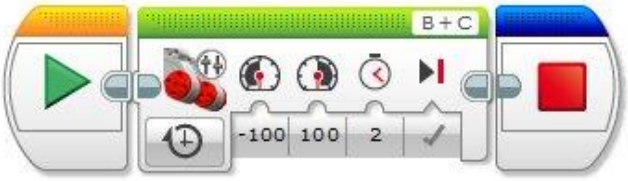
Az ábra a feladat megoldásának a programkódját mutatja be.

A programhoz a kormányvezérelt (*Steering Motor*) blokkot használtuk. A motorok leállítása nem szükséges, hiszen az 500°-os elfordulás után illetve a program végeztével egyébként is leállnak. A motorok leállítása blokkolással történt.

4/P2. Írjon programot, amelyet végrehajtva a robot 50-es sebességgel körbe forog 2 mp-ig!

A feladat megoldásának a programkódja:

A programhoz a kormányvezérelt (*Steering Motor*) blokkot használtuk, amely a B és C portra kötött motorokat 50-es sebességgel, de különböző irányban forgatja 2 mp-ig.
A helyben forgást úgy érjük el, hogy a *Steering* paraméter csúszkáját a C motor irányába toltuk el. Ennek hatására a motorok azonos sebességgel, de különböző irányban kezdenek forogni, és a robot helyben elfordul. A *Seconds* paraméter beállításával adtuk meg a mozgás idejét.
Ha a programhoz a sebességvezérelt (Tank Motor) blokkot használjuk, akkor a forgás gyorsabb is lehet. Mindkét motor sebessége abszolút értékben 100, de a két motornál eltérő az előjel. A többi paraméter beállítása azonos az előző megoldásnál használttal.



4/P3. Írjon programot, amelyet végrehajtva a robot 2 mp-ig tolat, majd balra fordul kb. 90°-ot, végül előre megy a tengely háromszoros körbefordulásig!

A program forráskódja három kormányvezérelt motor ikonból áll.

Az első blokk a B és C motorokat –50-es sebességgel 2 mp-ig forgatja. A negatív sebességérték a hátrafelé mozgást jelenti.
A második motor blokk valósítja meg a kb. 90°-os balra fordulást. Ehhez az összes „nyomatékot" a C (ez a beállítás függ attól, hogy melyik motort melyik portra kötöttük) motorra adjuk és a motorokat 50-es sebességgel 0,5 mp-ig működtetjük. A megfelelő időt célszerű kísérletezéssel meghatározni, mivel ez függhet a robot sebességétől, a kerék átmérőjétől vagy az akkumulátorok töltöttségi szintjétől. A pontosabb fordításra egy lehetőség a giro szenzor használata. Erről a későbbi fejezetekben lesz szó.
Felhívjuk a figyelmet arra, hogy az adott szöggel történő elfordítási mód beállításnál a paraméter „90°, *Degrees*" beállítása nem a robot 90°-kal történő elfordulását eredményezi, hanem a motor tengelyének 90°-os elfordulását.
A harmadik modul a háromszoros tengelyfordulásig történő előre mozgást valósítja meg.

4.2. Gyakorló feladatok

4/F1. Írjon programot, amelyet végrehajtva a robot körbe forog 3 mp-ig, majd előre megy 1000°-os tengelyfordulásig, majd ismét forog 3 mp-ig!

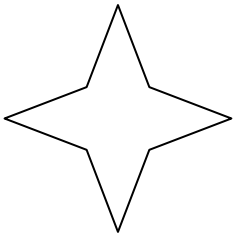
4/F2. Írjon programot, amelyet végrehajtva a robot nagy ívben balra kanyarodik 5 mp-ig, majd tolat 2 mp-ig, végül nagy ívben jobbra kanyarodik 5 mp-ig!

4/F3. Írjon programot, amelyet végrehajtva a robot nagy ívben jobbra kanyarodik 3 mp-ig, majd fordul kb. 180°-ot és nagy ívben balra kanyarodva halad 3 mp-ig!

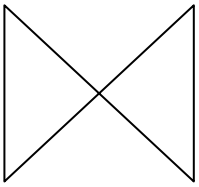
4/F4. Írjon programot, amelyet végrehajtva a robot mozgás közben egy négyzetet ír le!

4/F5. Írjon programot, amelyet végrehajtva a robot egy olyan téglalap mentén mozog, amelynek hosszabbik oldala kétszer akkora, mint a rövidebbik!

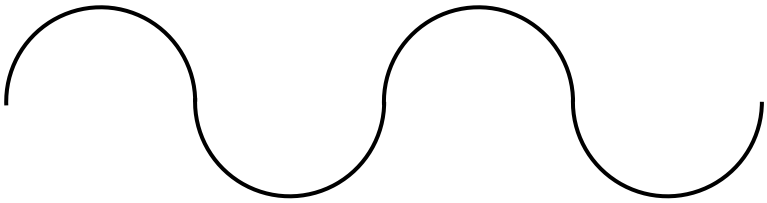
4/F6. Írjon programot, amelyet végrehajtva a robot mozgása során az alábbi alakzatot írja le!



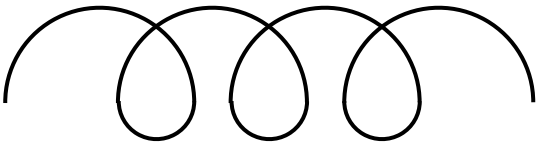
4/F7. Írjon programot, amelyet végrehajtva a robot mozgása során az alábbi alakzatot írja le!



4/F8. Írjon programot, amelyet végrehajtva a robot mozgása során az alábbi alakzatot írja le!



4/F9. Írjon programot, amelyet végrehajtva a robot mozgása során az alábbi alakzatot írja le!



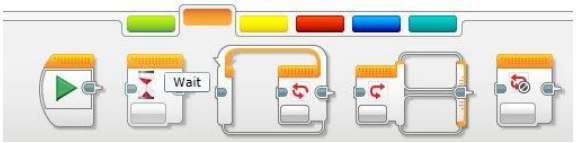
5. SZENZOROK HASZNÁLATA

Az előző fejezetben néhány egyszerű példán keresztül áttekintettük, hogyan mozgathatjuk a robotot. A robot külső érzékelői (szenzorai) segítségével képes érzékelni a „külvilág" jeleit. A szenzorok a tégla 1-4. sorszámozású portjaira csatlakoztathatók.
Az 1. fejezetben bemutatott szenzorok közül leggyakrabban a három „alapszenzort" használjuk, amelyek az EV3 oktatási készletben is megtalálhatók. Ezek a következők:

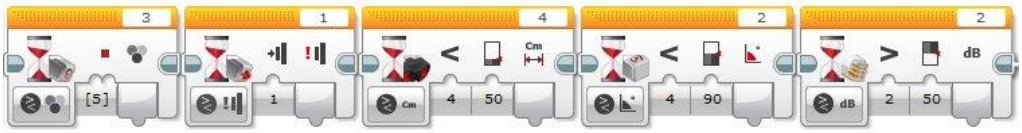
- ütközésérzékelő (*touch sensor*);
- színérzékelő/fényérzékelő (*colour sensor/light sensor*);
- távolságérzékelő (*ultrasonic sensor*)

Ezeken kívül további szenzorok is beszerezhetők, valamint az NXT változathoz készült érzékelőket is képes a szoftver kezelni.
Vannak olyan hardver eszközök is, amelyek ugyan külön külső szenzorként nem jelennek meg, tehát nem csatlakoztathatók az 1-4 bementi portokra, de vezérlésük, vagy funkcióik alapján a szenzorokkal mutatnak hasonlóságot. Például a készletben található szervomotorokba beépítettek egy érzékelőt, amely képes információkat szolgáltatni a motorok állapotáról. Így végeredményben a szervomotorokat is használhatjuk szenzorként. Az érzékelők a keretprogram számára a motor elfordulási szögét adják vissza fokokban vagy teljes tengelyfordulatokban mérve. A tégla nyomógombjai is használhatók ütközésérzékelőhöz hasonló módban.

Az érzékelők számára tehát négy csatlakozási pont (port) található a téglán. A motorokat kivéve ezekre a portokra csatlakoztathatók a környezet különböző értékeit mérő szenzorok.
A robot érzékelőit kétféle módon használhatjuk, ezért a legtöbb szenzorhoz két ikon áll a rendelkezésünkre.

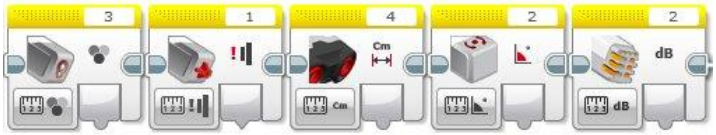


A szenzorok használatának egyik módja, amikor a megfelelő ikon beillesztése a programszálra, felfüggeszti a program további utasításainak végrehajtását, és csak a szenzoron bekövetkezett valamilyen esemény, vagy az érzékelő által visszaadott megfelelő érték hatására folytatódik a programszál utasításainak végrehajtása. Ennél a módnál valamennyi szenzor egy ikonnal a *Flow Control* kategóriában található *Wait*-tel vezérelhető (narancssárga szegélyű ikon). A homokóra jelölés a blokkon utal az utasítások várakoztatására.



A szenzorhasználat másik módja a Szenzor (*Sensor*) programcsoporton belül található ikonok programszálra illesztésével történik. Az ikonok sárga szegéllyel rendelkeznek. Használatukról a paraméterátadással foglalkozó fejezetben lesz szó. A lényeges különbség az előző lehetőségtől, hogy ennél a szenzorhasználati módnál nem áll meg a programszál utasításainak végrehajtása, csupán a program kiolvassa a szenzor által mért értéket és már lép is a következő utasításra. Ha ezt az értéket

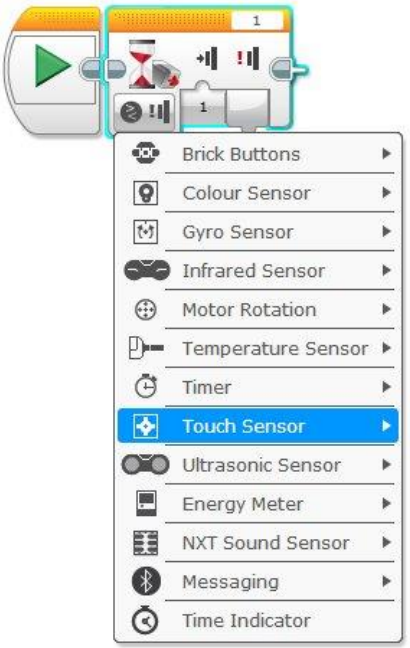
nem használjuk fel a programban, akkor fölösleges a modult használnunk, mert nem lesz hatással a program futására.



A továbbiakban a *Wait* ikon működését részletezzük.

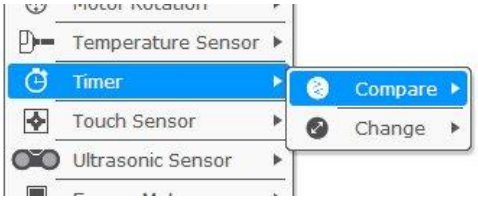
A *Wait* ikon lényegében addig várakoztatja a programot, amíg a szenzoron beállított esemény be nem következik, vagy amíg a szenzor a paraméterként megadott értéknek megfelelő adatot nem mér. Ha a beállított feltétel teljesül, akkor a vezérlés a program végrehajtását a *Wait*-et követő ikonnal folytatja.

Az ikon bal alsó részén lévő szimbólumra kattintva a működési módot tudjuk beállítani. A legördülő lista sok elemet tartalmaz. Nem mutatjuk be valamennyi használatát, mert ezek nagyon hasonlóak egymáshoz, csak az általános elvre utalunk néhány példán keresztül.
Első lépésben a használni kívánt „szenzort" válasszuk ki a listáról. Nem csak hagyományos értelemben vett szenzorok jelennek meg a legördülő listán, hanem minden olyan eszköz, amely képes valamilyen feltétel vizsgálatára (pl.: *Brick Buttons* (a tégla gombjai), *Messaging* (bluetooth üzenetcsere,...)
A táblázatban a listán szereplő eszközök neve, és zárójelben a legjellemzőbb mért érték szerepel.



A téglán szereplő nyomógombok (benyomott/felengedett állapot)
Szín/fény szenzor (%-os fényintenzitás érték vagy szín)
Giroszkóp (elfordulási szög vagy elfordulási arány)
Infravörös szenzor (távolság érték)
Motor elfordulási szög (érték fokban vagy tengelyelfordulásban)
Hőmérséklet szenzor (érték Celsiusban vagy Fahrenheitben)
Időzítő, stopper (eltelt idő másodpercben)
Ütközésérzékelő (benyomott/felengedett állapot)
Ultrahang szenzor (távolság érték cm-ben vagy incsben)
Multiméter (feszültség, áramerősség érték, teljesítmény)
Hang szenzor (decibel érték)
Bluetooth-on kapott jel (szám, szöveg vagy logikai érték)
Idő (másodperc)

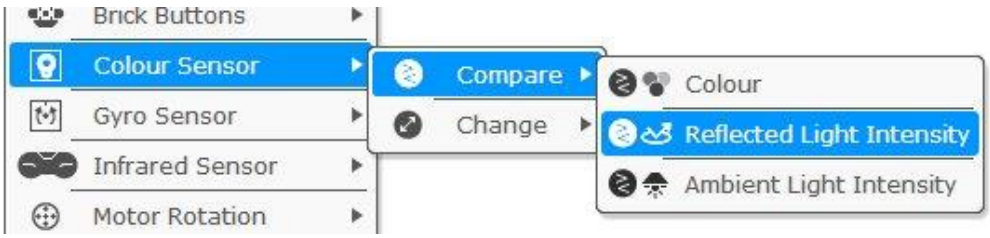
A legtöbb listaelemnek két további eleme, működési módja van. A *Compare* és a *Change*, Összehasonlítás és Változás.



Az alapvető különbség a két elem között, hogy Összehasonlításnál meg kell adnunk azt a határértéket és az ehhez kapcsolódó feltételt, amellyel folyamatosan összehasonlítja a futó program az eszköz által mért, észlelt értéket. Amennyiben a beállított feltétel teljesül, akkor tovább lép a program végrehajtása a következő utasításra, blokkra. A Változásnál azt a mértéket kell megadnunk, amellyel ha megváltozik az eszköz által folyamatosan mért, észlelt érték, akkor igaz értéket kapunk, és a következő utasításra léphetünk.
A fényszenzor esetén mutatjuk be a lehetőségeket.

5.1. Szín- vagy fényérzékelő (*Colour Sensor*)

A fényszenzor (*Light*) már az NXT 1.0 készlet része volt. A felületről visszavert fény intenzitását mérte. A mért érték 1-100 közötti skálán változott, tehát egy % értéket kaptunk vissza. A későbbi generációknál a fényszenzort felváltotta a színszenzor (az EV3-as robot esetén is használhatunk), amely az alapszínek megkülönböztetését is lehetővé tette, de továbbra is működött fényszenzor módban, ahol továbbra is fényintenzitást lehetett vele mérni. A programozás megkezdése előtt érdemes megmérni a visszavert fény intenzitását a különböző felületeken, mivel ez a felület színétől, tükrözési sajátságaitól és a fényviszonyoktól is függ. Ezzel tájékozódhatunk a környezet aktuális fényviszonyairól. A mérést a tégla képernyőmenüjében a *View* funkcióval, vagy ha számítógéphez csatlakoztatott állapotban van az eszköz, akkor a *Port View* funkcióval végezhetjük el.
Mind az Összehasonlító (*Compare*), mind a Változás (*Change*) üzemmódban három lehetőség közül választhatunk. Használhatjuk a fényszenzorként (*Light*) vagy színszenzorként (*Colour*) az eszközt. Fényszenzor üzemmódban azt is eldönthetjük, hogy a szenzor saját fényével (vörös színű) világítsuk-e meg a felületet, vagy a környezet fényét érzékeljük (saját fény nélkül). Ez a *Reflected Light* illetve az *Ambient Light* funkció.



Ha az Összehasonlító módot választjuk, akkor a blokk ikonján megjelenik két beviteli paraméter hely. Az elsővel a relációs jelet választhatjuk ki, míg a második paraméternek beírhatjuk a határértéket.

A képen látott beállítás szerint a megadott feltétel: a fényszenzor által mért fényintenzitás kisebb, mint 50. Ha ez teljesül, akkor lép a program végrehajtása a következő utasításra. Egyébként mindaddig nem lép tovább, míg a beállított feltétel hamis.

A harmadik megjelenő paraméter a listán minden blokknál a szenzor által mért aktuális értéket tartalmazza (részletesebben lásd Paraméterátadás fejezet), és ez kiviteli értékként jelenik meg, tehát nem szerkeszthetjük, csak olvashatjuk a tartalmát.

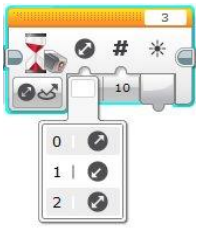
A szenzor csatlakozási portja az ikon jobb felső sarkában látható. Itt tudjuk megváltoztatni, de a rendszer automatikusan kezeli. A fenti példánál ez a 3-as port.

5/P1. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre mindaddig, amíg a fényérzékelője az alapszíntől eltérő színt nem észlel, ekkor álljon meg!

A feladat végrehajtása során homogén fehér felületen mozog a robot. Az eltérő színű csíkot pl. fekete színű szigetelő szalag felragasztásával állíthatjuk elő. A program megírása előtt a képernyő *View* funkciójával a különböző színű felületekről visszavert fény intenzitását megmértük, fehér: 62; fekete: 26. Tehát ha a fényérzékelő 44-nél kisebb értéket mér, akkor a robot elérte a fekete csíkot. A határértéket a két mért adat számtani átlaga alapján határoztuk meg.

A programnál az Összehasonlítás módot és a fényszenzort saját fénnyel (*Reflected Light*) használtuk. Az előre mozgásnál a motorerőt 50-re, a működési módot *On*-ra állítottuk, mivel nem tudjuk előre, hogy mikor ér a robot a fekete színű vonalhoz. A második ikon addig várakoztatja program utasításainak végrehajtását, amíg a fényérzékelő által mért érték kisebb nem lesz 44-nél, ezért a robot a fekete csík eléréséig folyamatosan halad előre. Elérve a fekete csíkot a mért érték kisebb lesz 44-nél, ezért a program a *Wait*-et követő utasítással folytatódik, és a robot megáll.

A program megoldására alkalmas a fényszenzor Változás (*Change*) üzemmódja is. Ebben az esetben nem határértéket állítunk be, hanem azt adjuk meg, hogy mekkora változás szükséges ahhoz, hogy továbblépjen a program végrehajtása a következő utasításra.



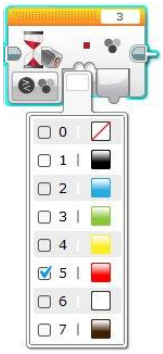
A működési módok megegyeznek az Összehasonlító módnál bemutatottal.

A megjelenő két beviteli paraméter közül az elsőnél állíthatjuk be, hogy a változás milyen irányú legyen: növekvő (0), csökkenő (1) vagy bármilyen (2). A második paraméter a változás mértéke.

Ha az előző feladatot szeretnénk megoldani, akkor a mért fekete (62) és fehér (26) értékeket véve alapul a legnagyobb változás 36 (62 – 26 = 36) lehetne. Ennyit nem

érdemes választani, hiszen a szenzor által mért értéket sok minden befolyásolhatja. Legyen a választott érték 10. Mivel fehér felületen mozog a robot, ezért a szenzorával mért érték csökkenni fog, ha elérte a fekete csíkot. Ez alapján a programja:

A második megoldásként bemutatott program általánosabbnak tekinthető, hiszen fekete és fehér szín helyett bármilyen más színek esetén is működik, csak a két szín közötti intenzitás különbségnek meg kell haladnia a 10-et és világosabbról sötétebbre kell hogy változzon (ez a változás irányát megadó paraméter „bármely – *Any*„ értékre állításával kikerülhető).

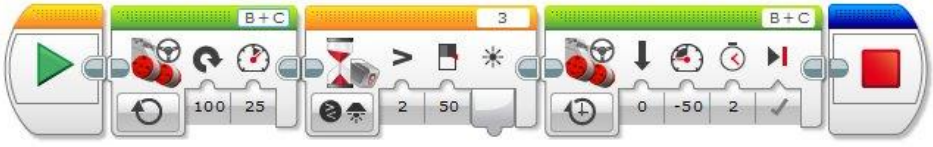


Ha a színszenzor működési módját *Colour*-ra állítjuk, akkor hét szín közül választhatunk, amelyet a szenzor képes felismerni. Ebben az esetben Összehasonlító módban nem kell határértéket megadnunk, mert a választott színtől eltérő mért érték jelenti a továbblépés feltételét, míg Változás üzemmódban semmiféle értéket nem kell megadnunk, mert az induláskor mért színérték megváltozása fogja a továbblépést eredményezni.

A következő példában azt mutatjuk be, hogy mikor érdemes a fényszenzornak a saját fény nélküli üzemmódját (*Ambient Light*) használni.

5/P3. Írjon programot, amelyet végrehajtva a robot forog, míg a fényérzékelőjére rá nem világítunk, ekkor megáll és tolat 2 mp-ig.

A feladat megoldásához az alaprobotot úgy kell átépítenünk, hogy a fényérzékelője előre, vagy felfelé nézzen. A képernyő *View/Ambient light* funkciójával megmértük a fényérzékelő által elnyelt fényintenzitás értékét rávilágítás nélkül (40) és egy zseblámpával történő rávilágítása esetén (57). A képernyőmenü *Wiev* funkciójának *Ambient light* értékénél a szenzorba épített vörös fényű led nem kapcsol be, így nem a felületről visszavert fényintenzitás értékeket mérjük, hanem a környezet fényét.



A folyamatos forgást a motor működési mód *On*-ra állításával valósítjuk meg. Így a robot addig forog, míg a *Wait* paramétereiben beállított feltétel be nem következik. A forgatásnál 25-ös motorerőt használtunk, hogy a forgás ne legyen túl gyors, és a fényérzékelő észlelni tudja a zseblámpa fényét.
Ha a robot érzékeli a zseblámpa fényét, akkor a beállított 50-es értéknél nagyobb fényintenzitást mér és a program végrehajtása a következő utasításra lép, vagyis a robot elkezd tolatni.

5/P4. Írjon programot, amelyet végrehajtva a robot egy fehér felületen egyenesen előre indul fekete csíkok fölött! A második fekete csík fölötti áthaladás után megáll.

A program megoldása során azt használjuk ki, hogy a robot fényszenzora a fehér felületről a fekete csíkra érkezve a fényintenzitás csökkenését, míg a fekete csíkról a fehér felületre érkezve a fényintenzitás növekedését fog mérni. Tehát egy csíkon áthaladva kétszer változnak meg jelentősen a mért értékek. Ennek megfelelően 4 db *Wait* modult fogunk használni (fényszenzor módban, saját fénnyel), kettőt-kettőt a csíkokon való áthaladás érzékelésére, tehát a 4 db színváltás figyelésére.
Mielőtt a programot megírjuk, célszerű a robot képernyőmenüjének *View* funkciójával megmérni a fehér felületen és a fekete csíkon mért fényintenzitás értékét. A mi esetünkben ezek: fekete 34 és fehér 54. Az eltérés 20, ezért *Change* üzemmódot használva, ennél kisebbnek, például 10-nek választjuk a változás mértékét. Mivel fehér felületen kezdődik a mozgás, ezért először csökkeni fog a mért érték (fekete felületre érve), majd növekedni fog (ismét fehér felületre érve), majd még egyszer ugyanez a változás következik be. A program úgy is elkészíthető, hogy a változás irányát nem vesszük figyelembe, csupán a azt a tényt, hogy színváltás esetén megváltozik a mért érték. Ebben az esetben használhatjuk az „*Any*" – bármilyen beállítást is, hiszen így is négy esetben történik majd változás (kétszer csökken, és kétszer nő a mért fényintenzitás).

A motorok mindaddig működnek, amíg a negyedik színváltás is megtörténik. Ekkor állnak le.
A fenti programmal ekvivalens, ha a két mért intenzitás érték számtani közepét vesszük határértéknek, és Összehasonlító (*Compare*) működési módban használjuk a fényszenzort. A határérték tehát: (34+54)/2=44.

Ezzel a programozási ötlettel tetszőleges számú csík után meg tudjuk állítani a robotot, de más szenzorok *Wait* ikonnal történő használatával bonyolultabb „minták" is összeállíthatók a robot vezérlésére.
A következő fejezetben bemutatott vezérlési szerkezetekkel a hasonló programkódok lényegesen rövidíthetők.

5.2. Várakozás megadott ideig (*Timer*)

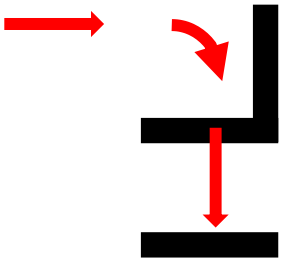
A szenzorokkal történő vezérlés mellett a *Wait* ikon arra is alkalmas, hogy a program végrehajtását adott ideig várakoztassa.

A működési mód listában két időre vonatkozó beállítási lehetőség is van. A *Time Indicator* beállítást választva megadhatjuk másodpercben, hogy mennyi ideig történjen a program futásának várakoztatása. Értékként tizedes törtek is használhatók, így akár milliszekundum mértékegységet is beállíthatunk.

A másik lehetőség a *Timer* működési mód. Ennek használata kicsit komplexebb. A programkörnyezetben rendelkezésünkre áll 8 stopper, amelyet a program bármelyik helyén elindíthatunk. Ezeket a stoppereket képes figyelni a program és az aktuális értéket felhasználni összehasonlításra. A *Timerek* használatáról a későbbi fejezetekben lesz szó.

Az adott ideig várakoztatás alkalmazására egy jó példa ha a robotnak olyan felületen kell mozognia és megkeresnie pl. egy a felülettől eltérő színű vonalat, ahol megtaláláshoz át kell haladnia zavaró színű területek fölött.

5/P5. Írjon programot, amelyet végrehajtva a robot egyenesen előre mozog és megáll egy fekete színű vonalnál, itt jobbra fordul és egyenesen halad az előző vonaltól különálló fekete vonalig. Ezt elérve megáll. A robot mozgását az ábra értelmezi.



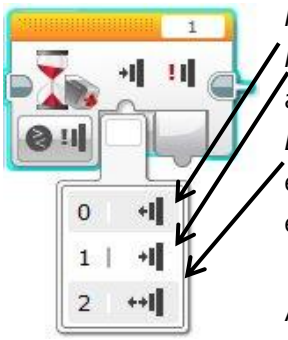
A problémát az jelenti, hogy az első fekete vonalat elérve a robot jobbra fordul kb. 90°-ot és elindul a másik fekete vonal felé. Útközben viszont van egy zavaró fekete vonal, ami fölött át kell, hogy haladjon. A korábban bemutatott ötlettel (lásd 5/P4-es példa) megoldhatónak tűnik a feladat, de elképzelhető, hogy a fordulás után a robot fényszenzora eleve a fekete csík fölé kerül, így rosszul fogja a színváltásokat érzékelni.

Az lenne a jó, ha fordulás után egy darabig „vakon” haladna a robot, tehát nem figyelné a fényszenzorát, csak miután kb. átért a problémás területen, akkor kezdené újra mérni a visszavert fény intenzitását. Ezt a „vakon” haladást úgy fogjuk elérni, hogy a fordulás után elindítjuk az egyenes mozgást, de a fényszenzor figyelése előtt egy *Time Indicator* blokkal várakoztatjuk 2 másodpercig a program továbbfutását, így a fényszenzor csak a 2 másodperc letelte után kezd ismét mérni, figyelni.

A fehér felület és fekete csík megkülönböztetését 10% mért érték csökkenéssel érzékeljük. A derékszögű fordulást tapasztalati úton meghatározott 0,5 másodperces mozgással érjük el.

5.3. Ütközésérzékelő (*Touch sensor*)

Az ütközésérzékelő három állapot megkülönböztetésére alkalmas.



Released – az ütközésérzékelő gombja kiengedett állapotban van.
Pressed – az ütközésérzékelő gombja benyomott állapotban van, ez az alapbeállítás.
Bumped – az ütközésérzékelő gombjának gyors benyomása, egérkattintásszerűen. Az érzékelő gombján állapotváltozás megy végbe, először benyomás, majd felengedés.

Az ütközésérzékelőt szintén a *Wait* blokkban tudjuk kiválasztani. Általában az Összehasonlító mód használata a célszerű.

A *Wait* ikon vezérlésének ütközésérzékelőre állítása után csak az érzékelő portjának a számát kell megadnunk (a rendszer automatikusan kezeli), illetve hogy az érzékelő három állapota közül melyiket figyeljük.

5/P6. Írjon programot, amelyet végrehajtva a robot egyenesen addig tolat, amíg akadálynak nem ütközik, ekkor megáll!

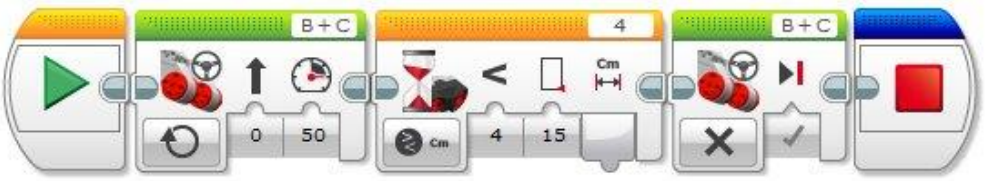
A folyamatos hátra mozgást az első *Steering Motor* ikon biztosítja, *On* működési módban. A *Wait* blokkal megadott utasítás a programot addig várakoztatja, míg az 1-es portra kötött ütközésérzékelő benyomott (*Pressed*) állapotba nem kerül. Ekkor tovább engedi a program futását, és a robot blokkolva megáll.

5.4. Ultrahangos távolságérzékelő (*Ultrasonic sensor*)

Az ultrahangos távolságérzékelő képes az előtte lévő tárgyak távolságát megmérni. A távolságot mindig az érzékelőhöz viszonyítva határozza meg centiméterben vagy hüvelykben mérve. A távolságot 5-250 cm tartományban képes megmérni. A *Wait* ikon ultrahangos távolságmérővel történő vezérlésének működési módjai és programozási filozófiája megegyezik a fényszenzornál bemutatottal.

5/P7. Írjon programot, amelyet végrehajtva a robot egyenesen halad mindaddig, amíg a távolságérzékelője 15 cm-nél kisebb távolságot nem mér! Ekkor álljon meg!

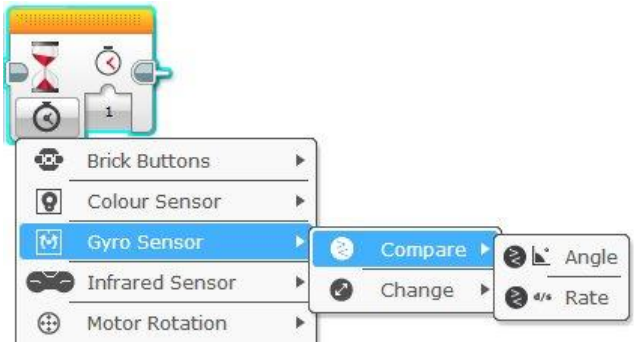
A következő ábra a feladat megoldásának programkódját mutatja:



A távolságérzékelő a 4. portra van kötve. A mértékegységet centiméterre, a továbblépési feltételként mért távolságot 15 cm-nél kisebbre állítottuk.
A feladatot az első fejezetben bemutatott robottal oldottuk meg. Ha ettől eltérő konstrukciójú robotot használunk, és a robot távolságérzékelője túl magasra van szerelve, akkor előfordulhat, hogy a robot nem látja meg az akadályt. Ha az ultrahangos távolságérzékelő a robot elejéhez viszonyítva túlságosan hátra van szerelve, akkor a robot eleje előbb ütközik az akadálynak, mint ahogyan az a beállított távolságon belülre kerülne. Mindkét esetben az akadálynak ütközik a robot és megpróbál továbbhaladni.

5.5. Giroszkóp (*Gyro sensor*)

A giroszenzor képes a robot előjeles elfordulását mérni fokokban. Amiben eltér például a motorok elfordulási szögének mérésétől, hogy míg azok a motor tengelyének elfordulását mérik, és ezáltal csak közvetve kapunk információt a robot tényleges fordulásáról, addig a giroszenzor a robot tényleges fordulását méri. Az előjeles elfordulás értékének viszonyítási pontja a szenzor *Reset* állapota. Tehát ha a programszálra beillesztjük a *Sensor* csoportban található *Gyro Sensor* blokkot és *Reset* módba kapcsoljuk, akkor lenullázzuk az elfordulás mérőt. Innentől kezdve a két irányú elfordulást előjeles értékként kaphatjuk vissza a szenzorból. A *Wait* blokkot használva a programszál utasításai addig nem futnak tovább, amíg a beállított feltétel (Összehasonlítás vagy Változás mód) nem teljesül. Tehát egyszerűen lehet a robot adott szöggel történő elfordulását vezérelni.



Kétféle értéket használhatunk a vezérléshez az elfordulási szöget: *Angle* (szög), illetve egy elfordulási arányt: *Rate*, amely a fordulás szögének időbeli változását adja vissza szög/másodperc mértékegységben.

5/P8. Írjon programot, amelyet végrehajtva a robot 1 másodperces egyenes haladás után megáll és 90°-ot fordul az óramutató járásával megegyezően, majd ismét halad 1 másodpercig! Ezután megáll.

Az 1 másodperces előre mozgás után lenulláztuk a giroszenzort. Elindítva az óramutató járásának megfelelő forgást (*Steering Motor*, *On* működési mód), a *Wait* modul *Gyro Sensor* módjánál a 90°-os változást figyeljük. Ha ez bekövetkezett, akkor ismét egyenesen folytatja a robot a mozgását.

A fordulásnál a két motort ellentétes irányba azonos sebességgel forgattuk. Ha közben figyeljük a motorok elfordulási szögét, akkor 160-170 fok közötti értéket kapunk. Tehát a robot 90°-nyi elfordulását kb. 165° illetve –165° motormozgás eredményezte. Mindez csak akkor igaz, ha a két motor azonos sebességgel és ellentétes irányba forog. Ha ezt a beállítást megváltoztatjuk, akkor más motorfordulási szöget kapunk a 90°-os elforduláshoz. Mindezt tapasztalati úton tudtuk volna

meghatározni, míg a giroszenzor segítségével minderre nem kell tekintettel lennünk, mert a robot fordulását érzékeli a szenzor.

5.6. Gyakorló feladatok

5/F1. Írjon programot, amelyet végrehajtva a robot megáll az asztal szélén! (Az első próbálkozásoknál nem ártanak a jó reflexek!)

5/F2. Írjon programot, amelyet végrehajtva a robot az asztal közepéről indulva a szélénél megáll, majd megfordul (kis tolatás után), és az asztal másik széléig haladva ott megáll!

5/F3. Írjon programot, amelyet végrehajtva a robot egyenesen előre halad a felülettől eltérő színű csík eléréséig, ekkor balra fordul kb. 90°-ot, majd tolat, amíg akadálynak nem ütközik!

5/F4. Írjon programot, amelyet végrehajtva a robot addig tolat, míg a fényérzékelőjére egy zseblámpával rá nem világítunk, ekkor mozogjon előre 75-ös motorerővel 2 mp-ig!

5/F5. Írjon programot, amelyet végrehajtva a robot addig áll, amíg a hangérzékelője 60 dB-nél nagyobb értéket nem mér! Ekkor induljon előre és addig haladjon, míg egy zseblámpával rá nem világítunk a fényérzékelőjére!

5/F6. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre mindaddig, amíg a fényérzékelője az alapszíntől eltérő színt nem észlel! Ekkor forduljon 180°-t, és haladjon előre, amíg az ultrahangos távolságmérője 25 cm-nél kisebb távolságot nem mér!

5/F7. Írjunk programot, amelyet végrehajtva robot előre halad, amíg az ultrahangos távolságmérője 25 cm-nél kisebb távolságot nem mér! Ekkor álljon meg és várakozzon 3 mp-ig! Ezt követően forduljon balra 90°-ot, és haladjon előre a felület színétől eltérő csíkig!

5/F8. Írjon programot, amelyet végrehajtva a robot fehér felületen lévő fekete csíksor fölött halad, és a harmadik fekete csík fölötti áthaladás után megáll!

5/F9. Írjon programot, amelyet végrehajtva a robot fehér felületen lévő fekete csíksor fölött halad, és a második csík fölötti áthaladás után 90°-ot fordul jobbra, majd egyenesen haladva akadálytól 15 cm-re megáll!

5/F10. Írjon programot, amelyet a robot végrehajtva egyenesen halad előre 50-es sebességgel egy fekete csíksor fölött. A második csíkon történt áthaladás után forduljon balra kb. 90°-ot, majd menjen egyenesen, amíg ultrahang szenzora 10 cm-en belül akadályt nem érzékel. Ezután ismét forduljon balra 90°-ot és haladjon előre fekete csíkig. Itt álljon meg és adjon egy másodperc hosszú hangjelzést.

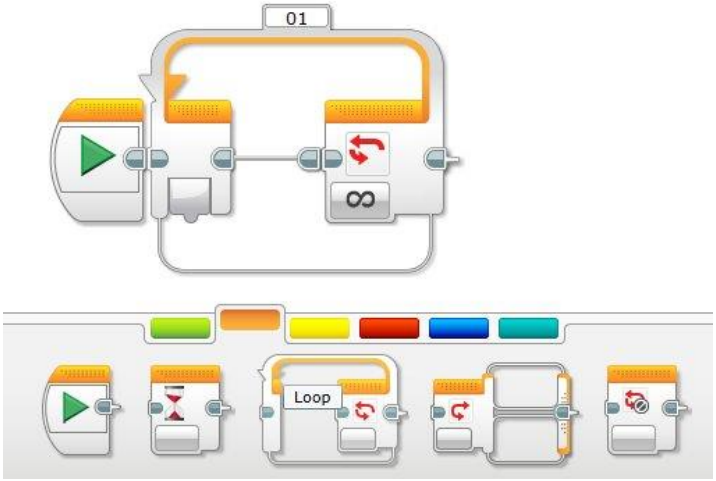
5/F11. Írjon programot, amelyet végrehajtva a robot egyenesen előre halad és akadálytól 10 cm-re megáll. Ekkor megszólaltat egy tetszőleges hangot 1 másodperc időtartamig. Ezután egyenesen tolatni kezd mindaddig, amíg az ütközésérzékelőjét nyomás nem éri. ekkor ismét megáll és megszólaltat két egymástól elkülönülő, de azonos hangot 1-1 másodpercig.

6. VEZÉRLÉSI SZERKEZETEK

Az eddigi példaprogramjaink mind úgynevezett szekvenciális szerkezetűek voltak, vagyis az utasítások sorban, egymás után hajtódtak végre. Sok esetben azonban a feladat megoldásához ez a szerkezet nem elegendő. Előfordulhat, hogy valamely feltételtől függően más-más utasításokat vagy utasítássorozatokat kell végrehajtani. Előfordul, hogy egy utasítást vagy utasítássorozatot többször meg kell ismételni. A programozási nyelvek ezek megvalósításához úgynevezett vezérlési szerkezeteket elágazásokat és ciklusokat tartalmaznak.

6.1. Ciklusok

Ciklusokat akkor használunk, ha egy utasítást vagy utasítássorozatot többször meg kell ismételni. Az ismétlendő utasításokat ciklusmagnak vagy ciklustörzsnek nevezzük. Abban az esetben, ha előre tudjuk, hogy hányszor kell megismételni a ciklusmagot, növekményes ciklusról beszélünk. Ilyenkor a ciklusmag mindig egy előre megadott darabszámszor hajtódik végre. Ha nem ismert előre az ismétlések száma, akkor a ciklusmag végrehajtását valamilyen feltétel teljesüléséhez, vagy nem teljesüléséhez kötjük. Az ilyen ciklusokat feltételes ciklusoknak nevezzük. A feltételes ciklusoknak két fajtáját különböztetjük meg. Az előfeltételes ciklus esetén a feltétel vizsgálata a ciklusmag végrehajtása előtt történik. Utófeltételes ciklusnál a feltételt a ciklusmag végrehajtása után vizsgáljuk.
Az EV3 grafikus nyelvben mind a növekményes, mind a feltételes ciklus megvalósítható. A feltételes ciklusok közül a nyelv csak az utófeltételes ciklust tartalmazza. A ciklusok ikonja a *Loop* ikon, amely a *Flow Control* kategóriában található.



A ciklus tehát ismétli azokat az utasításokat, amelyeket tartalmaz. Akkor ér véget az ismétlés, ha a beállított feltétel igazzá válik. Tehát kilépési feltételt kell megadnunk. A feltételt kell a programírás során először megadnunk, mert ez határozza meg a további beállítási lehetőségeket. A paraméterlista a ciklus jobb oldalán található ikonon kattintva érhető el (alapértelmezés: ∞ - *Unlimited*).
A lista igen hosszú, de öt jól elkülöníthető csoportra oszthatók a feltételek.

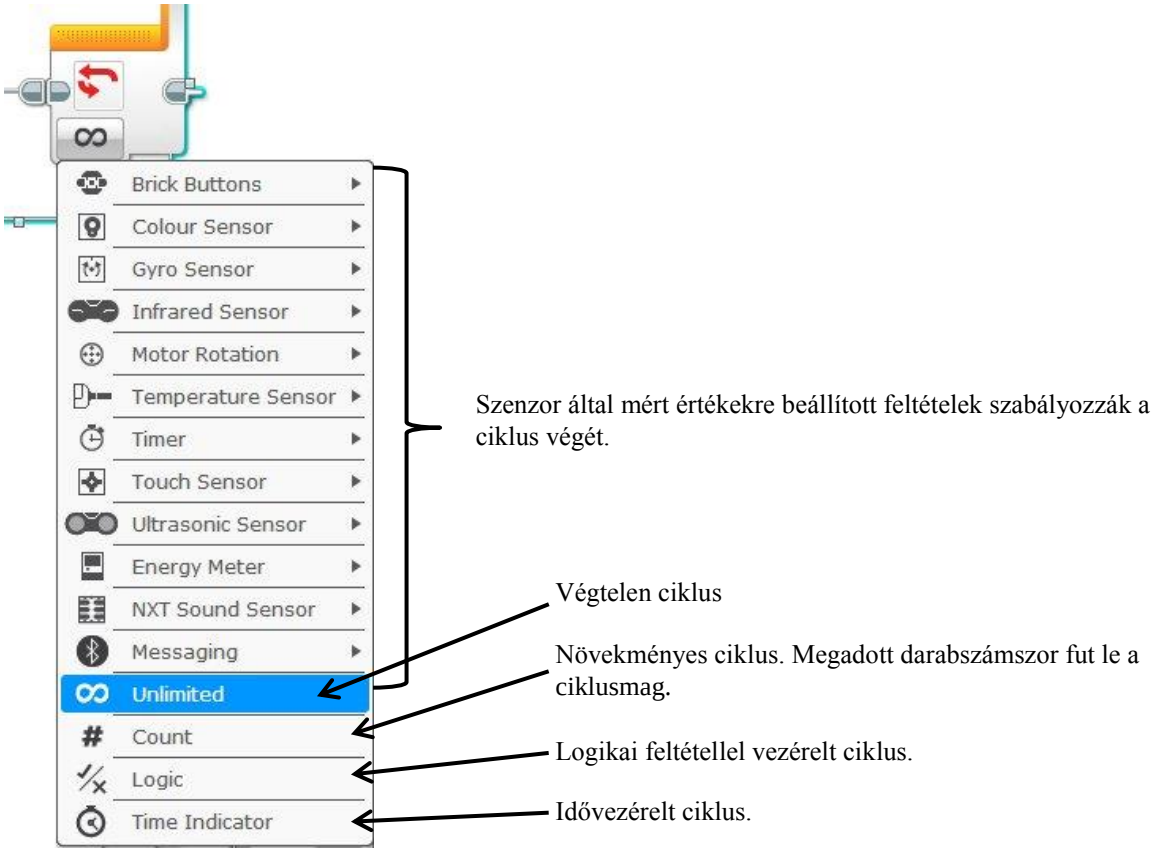
- Az első csoportba taroznak a valamilyen szenzorral vezérelt kilépési feltételek.

- Lehet a ciklus végtelen (*Unlimited*), ami azt jelenti, hogy nem ér véget, csak ha leállítjuk, pl. a tégla „ESC" gombjának megnyomásával. Ez egyben a program végét is jelenti. Le lehet állítani a ismételt utasítások között elhelyezett *Stop Program* ikonnal is (*Advanced* programcsoport), vagy a *Loop Interrupt* (kilépés a ciklusból) ikonnal.

- Vezérelhetjük a kilépést egy nemnegatív egész számmal (*Count*). Ebben az esetben a megadott számnak megfelelően a ciklusmagon belüli utasítások annyiszor hajtódnak végre, amennyi a beállított szám (növekményes ciklus).

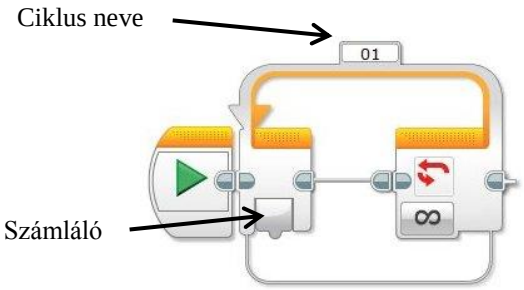
- Vezérelhetünk logikai feltétellel (*Logic*). Erre a későbbi fejezetekben látunk példákat. Olyan kifejezésekről van szó, amelyek kétféle értéket adhatnak vissza (igaz vagy hamis). Alapértelmezésben igaz érték esetén ér véget a ciklus (ezt meg lehet változtatni).

- Végezetül megadhatjuk, hogy hány másodpercig fusson a ciklus. Természetesen ebben az esetben, akkor áll le az utasítások végrehajtása, ha a beállított időértéknél nagyobb értéket értünk el. Előfordulhat, hogy a ciklusmag végrehajtásához több idő szükséges, mint amennyit a ciklus vezérlésénél beállítottunk. Mivel a feltétel teljesülését csak a ciklusmag végrehajtása után ellenőrzi a rendszer, ezért a ciklus futása a megadott időnél tovább is tarthat.



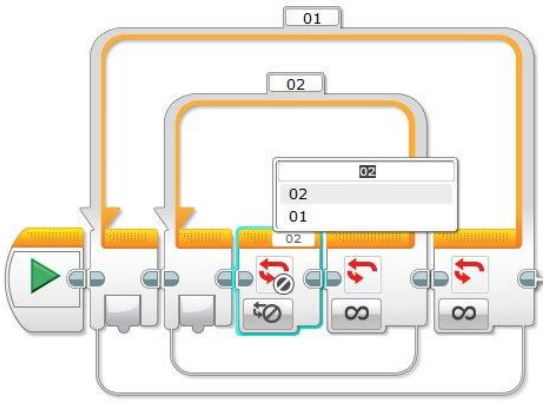
A ciklus ikon bal szélén találunk egy kimeneti paramétert (Lásd a Paraméterátadás fejezetet), amely egy számláló értékét adja vissza. A számláló automatikusan számolja a ciklusmag végrehajtási számát. Nullával kezdődik a számlálás és minden végrehajtás után automatikusan eggyel nő a számláló.

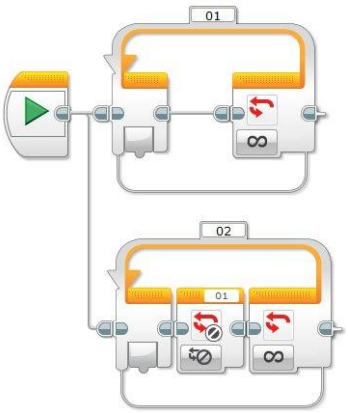
El is nevezhetjük ciklusainkat, ezzel is utalva a funkciójukra és megkönnyítve a későbbi programértelmezést.



Az EV3 szoftverben új elemként jelent meg a ciklusból történő kilépés lehetősége. Korábban ha egy ciklusból a véget érése előtt szerettünk volna kilépni, ezt csak úgy tehettük meg, ha egyben a programnak is vége szakadt. Például a cikluson belül egy feltételhez kötött szálon elhelyeztük a program vége parancsot. A másik lehetőség, hogy a ciklus kilépési feltételét bonyolult logikai feltételekkel vezéreltük, amelyek lehetőséget nyújtottak a különböző eseményekre történő kilépésre.

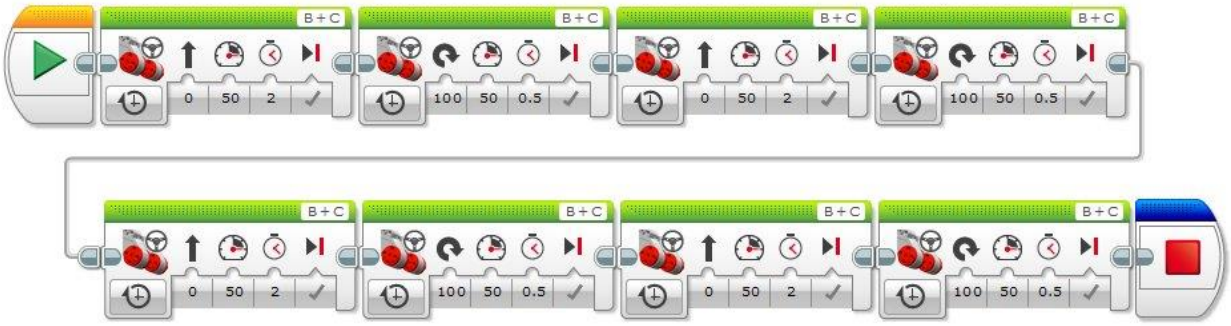
Egyszerűsödött a helyzet az új blokk megjelenésével.
A jobb felső sarokban lévő szövegdobozban választhatjuk ki, hogy melyik ciklusból szeretnénk kilépni. A ciklusból kilépés blokkot a cikluson kívül is használhatjuk. Tehát ki tudunk lépni egy adott ciklusból úgy is, ha másik programszálon helyezzük el a kilépési utasítást. A lenti példánál láthatjuk, hogy a beállított név alapján a két egymásba ágyazott, vagy külön szálon futó ciklus közül a 02-es azonosítójúból lépünk ki.





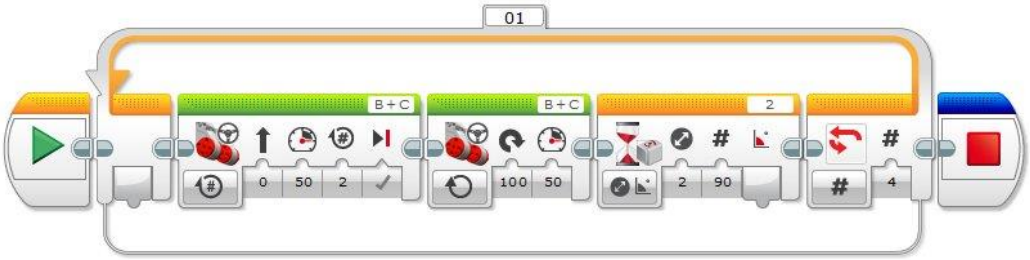
6/P1. Írjon programot, amelyet végrehajtva a robot mozgás közben egy négyzetet ír le!

Azok az olvasók, akik megoldották az *Egyszerű mozgások* című fejezet gyakorló feladatai közül a negyediket, már bizonyára tapasztalták, hogy a feladat két *Steering Motor* ikon négyszeri megismétlésével megoldható. Az első motor blokk egyenes haladást definiál, ez lesz a négyzet egyik oldala. A második motor blokk a kb. 90°-os elfordulást biztosítja, ez lesz a négyzet egyik csúcsa. Mindezt még háromszor végrehajtva kész a négyzet.



Egy növekményes ciklus segítségével a feladatot rövidebben és elegánsabban megoldhatjuk. A ciklusmagot, amelyet négyszer ismétlünk, a két *Steering Motor* ikon alkotja. Az első ikon az egyenes mozgást (négyzet oldala), míg a második ikon a fordulást (négyzet csúcsa) szabályozza.

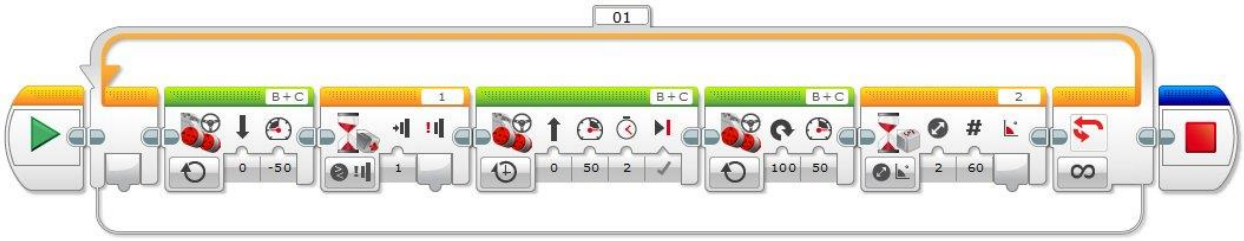
A 90°-os elforduláshoz szükséges 0,5 másodperces motormozgást tapasztalati úton határoztuk meg. Elegánsabb a program, ha a 90°-os fordulást a giroszenzor segítségével mérjük, Változás módban. A fordulást szabályozó motor működési módja ebben az esetben *On* kell, hogy legyen.



6/P2. Írjon programot, amelyet végrehajtva a robot ütközésig mozog hátra, majd előrehalad 2 mp-ig, fordul kb. 60°-ot, majd ezt kikapcsolásig ismétli!

Mivel a robotnak a leírt tevékenységet kikapcsolásig kell ismételnie, ezért az ezeket megvalósító utasításokat egy végtelen ciklusba foglaltuk. A feladat megoldásának a kódja a következő, ha fordulást tapasztalati úton határoztuk meg (0,4 másodperces motorműködés).

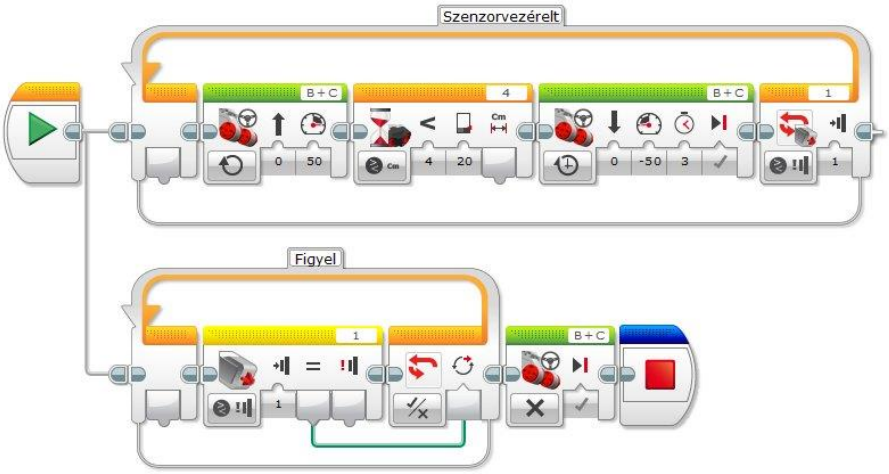
A tapasztalati méricskélést megspórolhatjuk a giroszenzor használatával. Változás mód, 60-as érték.



6/P3. Írjon programot, amelyet végrehajtva a robot addig halad előre, amíg 20 cm-en belül akadályt nem észlel, ekkor tolasson 3 másodpercig, ezt ismételje az ütközésérzékelőjének a megnyomásáig!

A feladat megoldásának a kódja következő lehetne:

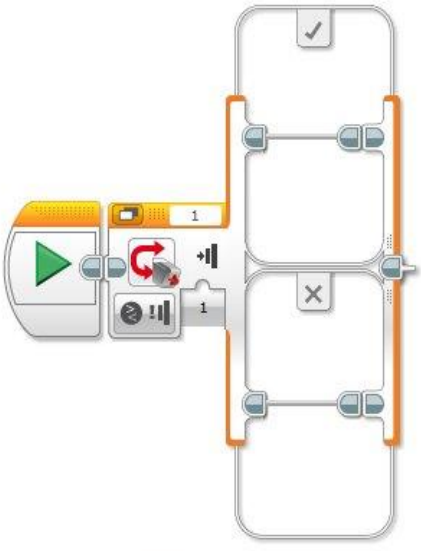
Ha többször is lefuttatjuk a programot, akkor azt tapasztaljuk, hogy nem megfelelően működik. Nem mindegy ugyanis, hogy melyik pillanatban nyomjuk meg az ütközésérzékelőt. A ciklus csak abban az esetben fejeződik be, ha az ütközésérzékelőt a tolatás után nyomjuk meg, egyébként nem érzékeli a program a szenzor megnyomását.
A hibát az okozza, hogy a ciklusmagban lévő utasítások várakoztatást tartalmaznak. Először az akadály észlelésére várakozik a program, majd a hátra mozgás idejének leteltére. Amíg a várakozási utasításokban beállított értékek nem teljesülnek, addig a program nem tudja végrehajtani az utánuk lévő utasításokat. Vagyis a mi esetünkben a ciklusvezérlésére használt ütközésérzékelő olvasása csak akkor történik meg, ha a ciklusmag utasításai végrehajtódtak. Ezért a programunk nem megfelelően működik.
A probléma a többszálú programok segítségével oldható meg például a következő módon.



A feladat ilyen módon történő megoldásához szükséges ismeretek részletes magyarázata a többszálú programokkal foglalkozó fejezetben található.

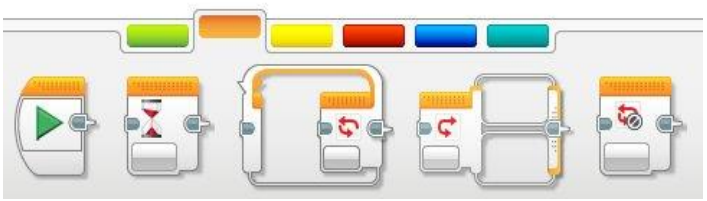
6.2. Elágazások

A másik fontos vezérlési szerkezet az elágazás. Az elágazást mindig valamilyen feltétel segítségével vezéreljük. Az elágazás lehet egyágú, ekkor ha igaz a megadott feltétel, akkor a hozzá tartozó utasítássorozatot végrehajtjuk, egyébként pedig az elágazást követő utasítással folytatjuk. A kétágú elágazás azt jelenti, hogy ha igaz az elágazást vezérlő feltétel, akkor az egyik utasítássorozatot kell végrehajtani, hamis feltétel esetén pedig egy másikat. Valamely utasítássorozat végrehajtása után a program az elágazást követő utasítással folytatódik. Többágú elágazásokat is létre lehet hozni az elágazások egymásba ágyazásával, vagy ha az elágazást olyan feltételekkel vezéreljük, amelyek közül legfeljebb egy teljesülhet.
Az elágazásokat a *Flow Control* kategóriában található *Switch* ikonnal hozhatunk létre.

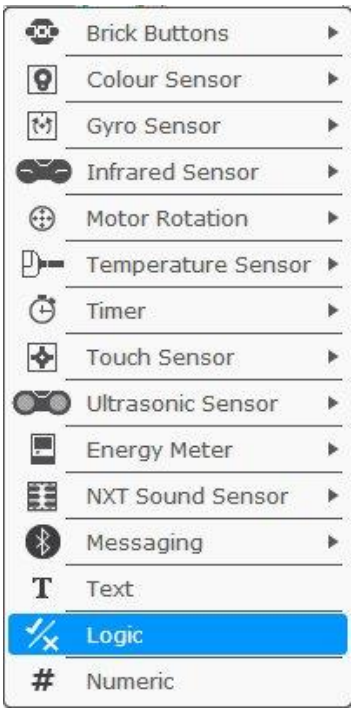


Igaz ág.
Ha a feltétel igaz, akkor kerülnek végrehajtásra a programszálon lévő utasítások.

Hamis ág.
Ha a feltétel hamis, akkor kerülnek végrehajtásra a programszálon lévő utasítások.



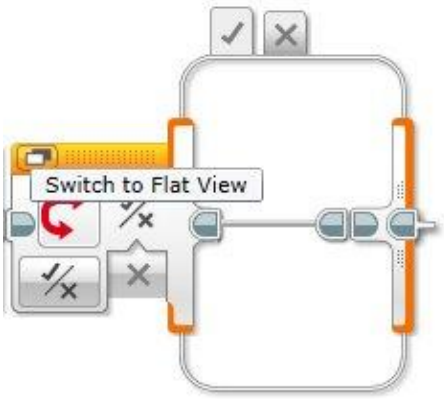
A *Switch* ikon esetében is azt kell először megadnunk, hogy hogyan történjen a vezérlés.



A vezérlési mód egy listáról választható.
Beállítható, hogy a feltételt egy szenzor által mért érték szolgáltassa-e. Ebben az esetben meg kell adnunk a szenzor esetén azt a határértéket, amelyhez a mért értéket hasonlítva igaz vagy hamis lehet a feltételvizsgálat eredménye.
A *Logic* beállítási mód használata esetén egy összetett logikai feltételt adhatunk meg vezérlésként.
Numeric beállítási módnál többszálú ciklusok is használhatók. Megadhatjuk, hogy milyen eredmény esetén milyen utasítássor fusson le.
Text beállítási mód hasonló a *Numeric* beállításhoz, de nem számok, hanem szöveg lesz a különböző programszálak vezérlő értéke.

Ha az elágazást valamely szenzor segítségével vezéreljük, akkor az ikon további paraméterezése természetesen függ a kiválasztott szenzortól is. A paraméterezés lényegében megegyezik a *Wait* ikonnál leírtakkal, de gyakorlatilag csak az Összehasonlító (*Compare*) mód jelenik meg, hiszen ebben az esetben egyébként is egy logikai értéket kapunk eredményül (igaz/hamis a beállított feltétel). A fejezetben bemutatott példákkal szemléltetjük a használatot.

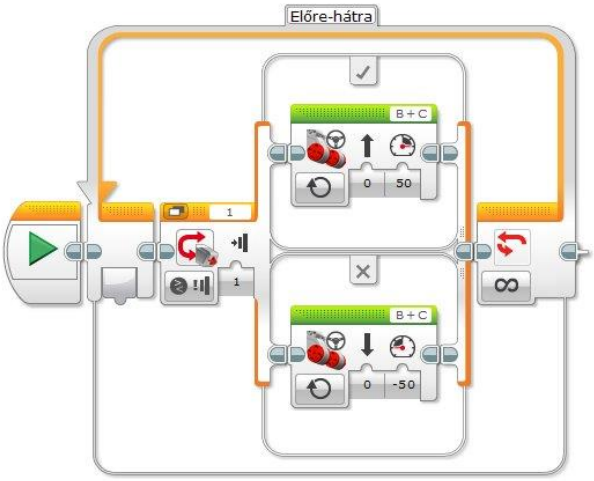
A kettőnél több ágú elágazások létrehozását a 6/P7-es példában mutatjuk be.



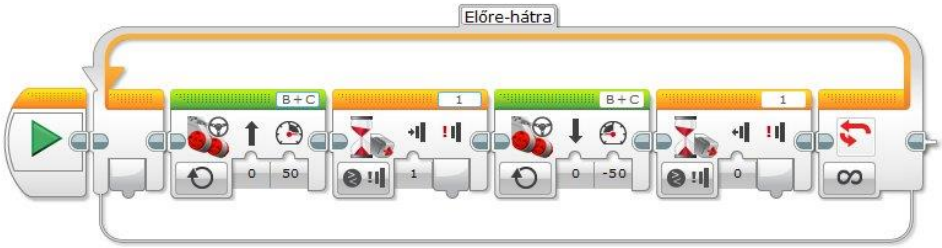
A *Switch* modul bal felső szélén található *Flat View* ikonra kattintva a blokk grafikus megjelenését szabályozhatjuk. Bekapcsolt állapota esetén mindkét ág látszik a képernyőn, egyébként csak az egyik. Ilyen esetben a két ág között a megfelelő fülön történő kattintással válthatunk. Ez a képernyőn tömörebb megjelenést eredményez.
A ciklusok és elágazások képernyőn elfoglalt méretét az informatikában szokásos méretező pontok segítségével is állíthatjuk. Ezek megjelennek a blokkok köré képzelt téglalapok sarkaiban és oldalfelező pontjainál. Így egy képhez hasonlóan a blokkok megjelenési mérete is szabályozható.

6/P4. Írjon programot, amelyet végrehajtva a robot előre mozog, ha az ütközésérzékelője benyomott állapotban van, és hátra mozog, ha kiengedett állapotban van, mindezt kikapcsolásig ismételje!

A program kódja a következő:



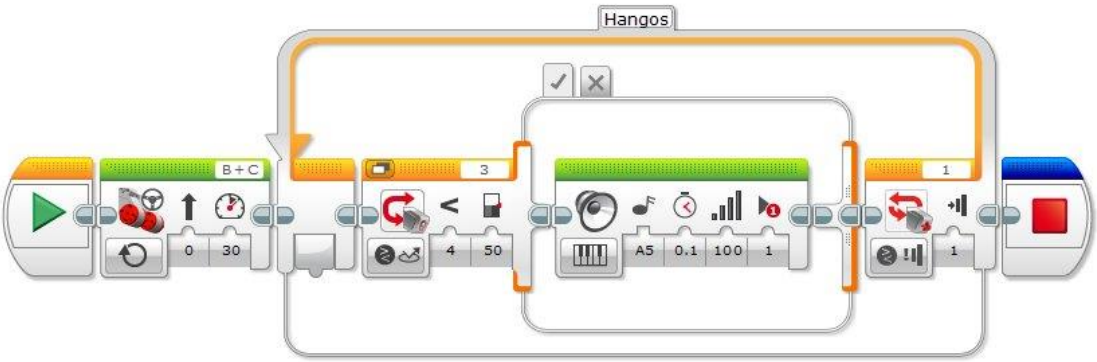
Mivel a tevékenységet a kikapcsolásig kell ismételni, ezért egy végtelen ciklust használunk. A ciklusmagot egy kétágú elágazás alkotja, amelyet az ütközésérzékelővel vezérlünk.
Az 1. portra kötött ütközésérzékelő benyomott (*Pressed*) állapotát figyeljük. Ha az ütközésérzékelőt benyomjuk, akkor a *Switch* feltétele teljesül, ezért a robot a felső igaz ágon szereplő utasítást hajtja végre, és folyamatosan előre halad. Ennek eléréséhez a *Steering Motor* blokk működési módját *Unlimited*-re kell állítani. Ha az ütközésérzékelő kiengedett állapotban van, akkor a feltétel nem teljesül. Ekkor az alsó, hamis ágon szereplő utasításokat hajtja végre a robot, és hátrafelé mozog. A program a tégla „*ESC*" gombjának megnyomásával szakítható meg.
A feladat elágazás használata nélkül is megoldható a következő módon:



A B és C motor bekapcsolása után (*On*) a robot egyenesen előre indul. Mindaddig halad erre, amíg be nem nyomjuk az ütközésérzékelőjét: *Wait* modul, *Touch Sensor* működési mód, 1-es *Pressed* paraméter. Ha benyomtuk, akkor a várakoztatási feltétel igazzá válik, és a következő blokk hatására a robot tolatni kezd. Ezt a mozgást akkor hagyja abba, ha felengedtük az ütközésérzékelőt. Ezután az utasítások végrehajtását kezdi elölről.

6/P5. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre egy az alapszíntől jól megkülönböztethető színű csíkokat tartalmazó felületen! A csíkon áthaladva adjon hangjelzést!

A robot előre haladás közben a fényszenzorával folyamatosan méri az alatta lévő felületről visszavert fény intenzitását. Amennyiben a mért érték eltér az alapszíntől, akkor hangjelzést ad. A program megírása előtt mind az alapszínről, mind pedig a csíkokról visszavert értéket megmértük. A mért értékek alapján meghatározott határértéket a programban konstansként használtuk (ez a példánál 50).

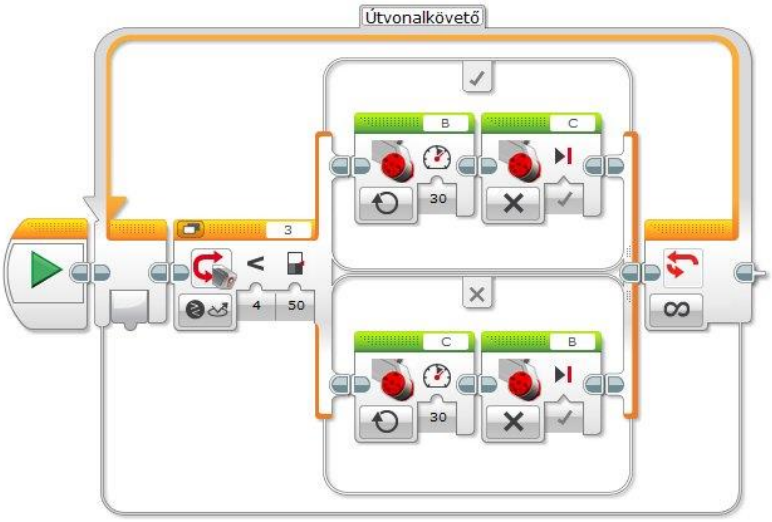


A folyamatos haladást és a hangot adó utasításokat egy végtelen ciklusban helyeztük el. A ciklusmag első utasítása egy *Steering Motor* ikon, amely 30-as motorerővel folyamatosan előre mozgatja a robotot. A motort elegendő a cikluson kívül bekapcsolni, mert a leállításig folyamatosan működni fog. A ciklusban csak egy egyágú elágazás van. Mivel a mérés során az alapszínről visszavert fény intenzitása 65, a fekete csíkról visszavert fényé pedig 35 volt, ezért az elágazás feltételeként azt adtuk meg, hogy a visszavert fény értéke kisebb 50-nél (a 65-ös és 35-ös értékek számtani átlaga: 50). A feltétel teljesülése estén a robot egy normál zenei A hangot ad, 0,1 másodpercig, maximális hangerővel (100). Ha a feltétel nem teljesül, akkor nem kell csinálnunk semmit, ezért a *Switch* alsó ága üresen marad, így azt nem jelenítettük meg. A robot ütközésérzékelőjének megnyomására a program befejeződik.

6/P6. Írjon programot, amelyet végrehajtva a robot a fényszenzora segítségével az alapszíntől jól megkülönböztethető színű útvonalat követ! Az útvonal lehet például egy fehér felületre felragasztott fekete szigetelőszalagból készített vonal.

A fehér alapszínre mért érték 65, a fekete útvonalra pedig 35. Vegyük ezek számtani közepét, ami a mi esetünkben 50. Az útvonal követése úgy történik, hogy ha a fényszenzor 50-nél nagyobb értéket mér, akkor a szenzor nem az útvonal felett van. Ilyenkor az egyik motorjával előre halad, a másik motor kikapcsolt állapotban van. Ennek hatására a robot ráfordul az útvonalra. Ekkor azonban 50-nél kisebb értéket mér. Ebben az esetben a robot a másik motort forgatja előre, az előzőleg működőt leállítja, és lefordul az útvonalról (szenzora nem az útvonal fölött lesz). Ezt egy végtelen ciklusban ismételve a robot kígyózó mozgással követi az útvonalat. A motorokat nem célszerű túl nagy sebességgel működtetni, mivel ekkor a robot nagyokat fordul, és elveszítheti a követendő útvonalat. A feladat megoldásánál 30-as erővel működtettük a motorokat.
A program működése során a robot tulajdonképpen nem az útvonalat követi, hanem a fekete sáv határvonalát.

A megoldás kódja a következő:

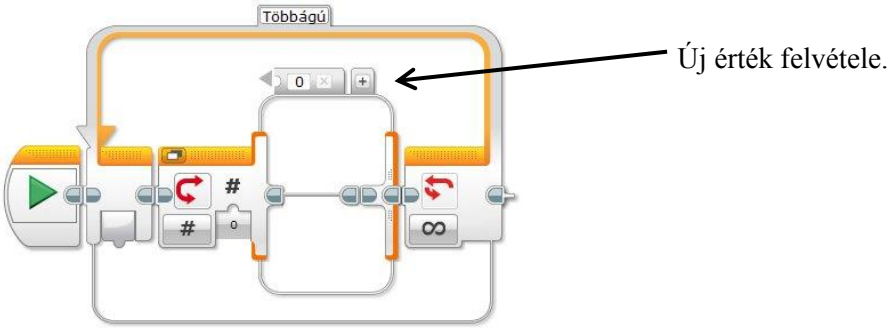


Ha az elágazás két szálán a második motort nem állítjuk le, hanem kisebb sebességgel hátrafelé forgatjuk, akkor ugyan lassabb lesz az útvonalkövetés, de élesebben kanyarodó útvonalat is tudunk követni.

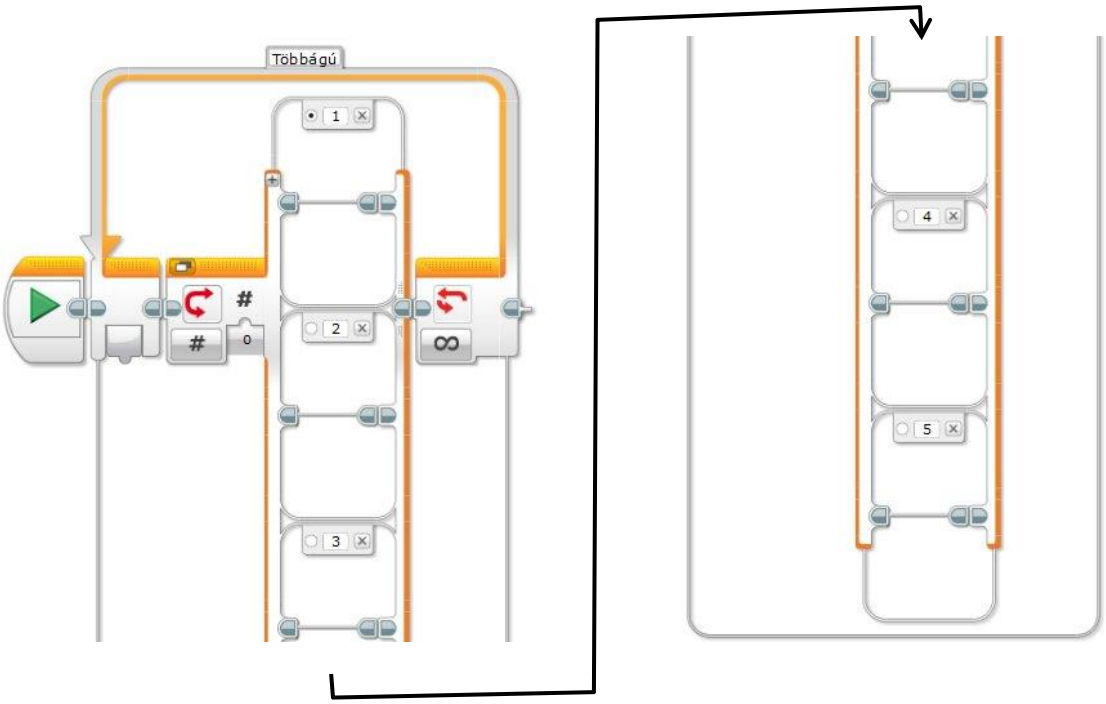
6/P7. Írjon programot, amelyet végrehajtva a robot az ütközésérzékelő megnyomására véletlenszerűen sorsol egy 1 és 5 közötti egész számot, majd a képernyőre írja szövegesen a számnak megfelelő osztályzatot, s ezt kikapcsolásig ismétli!

A kisorsolt értéktől függően az „elégtelen”, „elégséges”, „közepes”, „jó” és „jeles” szavak valamelyikét kell a képernyőn megjeleníteni (ékezet nélküli formában). Ehhez egy többágú elágazást használtunk, amelyet egy szám típusú értékkel vezéreltünk. Ennek a létrehozásához a *Switch* ikont a következő módon kell paraméterezni.

Először is a működési módot állítsuk *Numeric*-ra,. Ekkor egy kétágú elágazás jön létre, melynek első ágához a 1 a másodikhoz pedig az 0 érték van rendelve. Többágú elágazás létrehozásához kattintsunk a *Flat view* megjelenítési módra. Majd a lapozzunk az utolsó értékre (0), amely mellett megjelenik egy **+** jel. Erre kattintva további ágak vehetők fel. Összesen 5 ágú elágazást kell készítenünk.

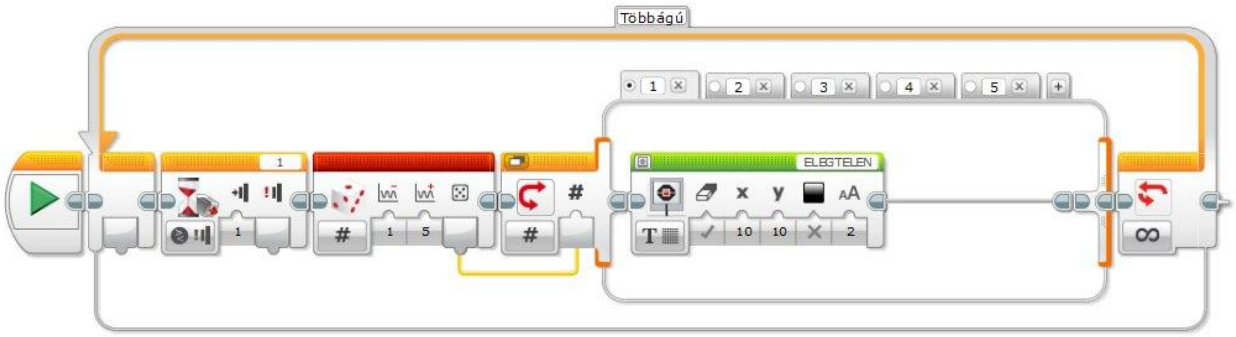


Megjelenítve az elágazás valamennyi szálát, beírhatjuk azokat az értékeket, amelyekre le kell futnia az adott szálon szereplő utasításoknak.



Ezek után minden ágra egy *Display* ikont kell helyeznünk, amely a képernyőre írást teszi lehetővé. Az első ágon a kiíratandó szöveg az „ELEGTELEN”, a másodikon az „ELEGSEGES”, míg végül az ötödiken a „JELES”. Az első ágat akkor hajtja végre a program, ha a kisorsolt szám az 1, a másodikat, ha 2 és végül az ötödik ágat, ha 5. A *Display* blokk, és a véletlen szám előállítására szolgáló modulok működését, és a paraméterátadást a könyv későbbi fejezetei részletesen tárgyalják.

A feladat megoldásra szolgáló program kódja a következő:



6.3. Gyakorló feladatok

6/F1. Írjon programot, amelyet végrehajtva a robot ívben fordul 1 mp-ig balra, majd 1 mp-ig jobbra, mindezt ötször ismételje!

6/F2. Írjon programot, amelyet végrehajtva robot egyenesen halad előre mindaddig, amíg a fényérzékelőjére rá nem világítunk, ekkor forduljon 90°-ot, és kezdje elölről a mozgást! Mindezt kikapcsolásig ismételje!

6/F3. Írjon programot, amelyet végrehajtva a robot egy fehér alapszínű, fekete csíkkal határolt területen belül mozog kikapcsolásig! Ha a robot eléri a terület határát, akkor tolasson, majd forduljon egy bizonyos szöggel! Ezt követően kezdje ismét az előre mozgást!

6/F4. Írjon programot, amelyet végrehajtva a robot addig halad előre, amíg az ultrahangszenzorával 15 cm-en belül akadályt nem észlel, ekkor az ütközésérzékelő benyomásáig haladjon hátra! Mindezt ismételje kikapcsolásig!

6/F5. Írjon programot, amelyet végrehajtva a robot képes az ultrahangos távolságmérőjével egy akadály észlelésére és követésére! A robot egyenletes sebességgel forog, és ha 20 cm-es távolságon belül akadályt észlel, akkor elindul felé. Ha elveszíti az akadályt, azaz 20 cm-en kívülre kerül, akkor forogjon újra! Mindezt kikapcsolásig ismételje!

6/F7. Írjon programot az előző feladat mintájára, amelyet végrehajtva a robot képes egy zseblámpa fényének észlelésére és követésére!

6/F8. Írjon programot, amelyet végrehajtva a robot hatszög alakú pályán mozog!

6/F9. Írjon programot, amelyet végrehajtva a robot egyetlen fényszenzorával követi a fehér felületre ragasztott fekete vonalat! Ha az útkövetés során a robot 20 cm-en belül akadályt észlel az ultrahang szenzorával, akkor forduljon meg és kövesse a fekete vonalat visszafelé!

6/F10. Írjon programot, amelyet végrehajtva a robot 40-es sebességgel forog mindaddig, amíg ultrahang szenzora 10 cm-en belül akadályt érzékel. Ekkor tolatni kezd. A tolatást mindaddig végezze, amíg fényérzékelője fekete vonalat nem érzékel! Elérve a vonalat azt kövesse (balra), és akadálytól 10 cm-re álljon meg! Ekkor a robotot tetszőleges helyre áthelyezve, az ütközésérzékelő megnyomására kezdje újra elölről a korábbi lépéseket. Mindezt kikapcsolásig ismételje!

6/F11. Írjon programot, amelyet a robot végrehajtva startpozícióból indul és követi a fekete vonalat akadálytól 10 cm-ig. Ekkor megáll és 90°-ot balra fordul. A fordulás után elindul egyenesen. Akadálytól 10 cm-re megáll, majd helyben forogni kezd mindaddig, amíg fényszenzora fekete csíkot nem érzékel. Ekkor megáll és sípol 1 másodpercen keresztül, majd befejezi a programját.

7. PARAMÉTEREK, VÁLTOZÓK ÉS A PARAMÉTERÁTADÁS

7.1. Alapvető paraméterek, adattípusok a programnyelveknél

A programozás során szükségünk lehet arra, hogy a bemeneti szenzorok (fényérzékelő, ütközésérzékelő, távolságérzékelő, stb.) által mért értékeket felhasználjuk a robot irányítására. Ezt már eddig is láttuk, hiszen a ciklusok és elágazások feltételeiként többször használtunk ilyen értékeket. Sok esetben azonban ezekkel az értékekkel műveleteket kell végezni, mielőtt használnánk őket, vagy a feltételes elágazások, ciklusok nem jöhetnek szóba a program adott részének megvalósítása során. Ilyenkor szükséges a szenzorok által mért értékeket valamilyen módon átadni, eljuttatni az egyes programmoduloknak. Ezt a célt szolgálja a paraméterátadás technikája. Paraméter alatt a „mért" értéket értjük. A legtöbb esetben ez egy szám, de előfordulhat, hogy szöveges adatról vagy logikai értékről (igaz/hamis) van szó.

A programozási nyelvekben, az informatikában az egyes adatok különböző típusúak lehetnek. Egy adat típusát általában az dönti el, hogy a programíró milyen módon rendelkezett az adat tárolásáról a memóriában. A legtöbb esetben más matematikai elv alapján tárolja például a számítógép a számokat és a szöveges adatokat. A legtöbb esetben az alapján döntjük el a tárolás formáját, és ezáltal az adat típusát, hogy milyen jellegű műveleteket végzünk az adattal. Nem csak a betűket tartalmazó karaktersorozat lehet szöveg. A csupán számjegyekből álló jelsorozatot is tárolhatjuk szövegként, ha nem végzünk matematikai műveleteket vele, csupán a szövegekre jellemző tulajdonságaikat használjuk ki. Pl. a telefonszámok számjegyekből állnak, de nem használjuk ki a matematikai tulajdonságaikat (általában). Nem szorozzuk őket kettővel, nem vonjuk ki őket egymásból, stb. Szükségünk lehet viszont az első két számjegyre (körzetszám), sorba rendezhetjük őket, stb. Tehát inkább szövegekre és nem a számokra jellemző tulajdonságaik vannak. Az elnevezés ellenére (telefonszám), inkább szöveges típusnak tekinthetők.
Az informatika három alapvető adattípusa: szám, szöveg, logikai érték.
Több programnyelv ennél lényegesen több típust különböztet meg, de valamennyi visszavezethető az alaptípusokra.

A számokat további két kategóriára szokás bontani, a különböző memóriabeli tárolási elv alapján. Vannak az egész számok (*integer*), amelyek nem rendelkeznek tizedesrésszel, és vannak a valós számok (*real*), amelyekben szerepel tizedesrész (tizedes vessző/pont utáni számjegyeket tartalmaz). Az informatikai elnevezés tehát különbözik a matematikában megszokottól (ott az egész számok egyben valós számok is).
Az egész illetve valós számok memóriabeli tárolási különbsége mellett programozási szempontból is fontos különbségek vannak. Sok programnyelvben egy osztás esetén, ha csak egész számok között végezzük el a műveletet, akkor az eredmény is egész lesz, függetlenül a matematikai eredménytől. Például 7:2 = 3. Az eredmény tehát a matematikai hányadosból a tizedes rész elhagyásával keletkezik. Lehetne azt gondolni, hogy ez nem túl szerencsés, hiszen félrevezető lehet a kapott eredmény. De ha tisztában vagyunk azzal, hogy az osztás egész számok körében így működik, akkor kihasználhatjuk ennek az előnyeit például az osztási maradékok esetén. Osztási maradékot matematikailag csak egész számok osztásánál kaphatunk. A fenti példánál az osztási maradék 1. Ezt matematikailag megkaphatjuk, ha a hányadost megszorozzuk az osztóval és a szorzatot kivonjuk az eredeti számból: 7–3x2 = 1. A maradék ismerete pedig sokszor hasznos lehet programozási szempontból. Például a maradék dönti el, hogy egy szám páros vagy páratlan (ha 2-vel osztva 1 a maradék, akkor páratlan, ha

0, akkor páros). Ha az egész számoknak egy egyesével növekvő sorozatát tekintjük, akkor a páros és páratlan számok felváltva követik egymást. Tehát egy olyan program esetén, ahol váltakozva kell különböző tevékenységeket végezni, a 2-vel történő osztás utáni maradék vizsgálata segíti a programírást. Ha 2-nél nagyobb számmal osztunk, akkor a ciklikus ismétlődés periódusa hosszabb lesz. Például 5-tel osztásnál ötféle maradékot kaphatunk 0-tól 4-ig. Ezt kihasználva 5 kategóriába tudjuk besorolni az elvégzendő tevékenységeket. Mindezt persze csak akkor lehetséges, ha az adott programnyelvben van ilyen egész osztás, vagy tudunk ilyen algoritmust készíteni.
Ha csak valós számokkal dolgozik a rendszer, akkor minden osztás valós eredményt ad, még akkor is, ha nincs értékes tizedes jegy, így a maradékok sem értelmezhetők (pl.: valós számok osztása esetén 7:2 = 3,5). A maradékok előállításhoz ekkor külön programot kell készíteni.

A logikai típusú adatok kétfélék lehetnek: igaz vagy hamis (*true* vagy *false*). Ha számszerű adatot is rendel a program a logikai értékhez, akkor a hamis a nullának, míg az igaz érték az egynek (vagy néhány programnyelv esetén bármilyen nem nulla számnak) felel meg.

Az adat típusától függ, hogy milyen művelet végezhető vele (matematikai összeadás, karaktersorozatok összefűzése, stb.). Például számként tárolva a 2 és 3 értéket, a 2+3 művelet az 5-öt adja eredményül. Szövegként tárolva a '2' és '3' értéket a '2'+'3' művelet a '23' eredményt adja a legtöbb programnyelvben.
Bizonyos nyelvekben nem keverhetők az egyes adattípusok, tehát nem adhatunk számhoz szöveget, vagy nem fűzhetünk össze logikai értéket számmal. Más programnyelvekben ez megengedett, de ismernünk kell a művelet gépi kiértékelésének szabályait az eredmény értelmezéséhez.

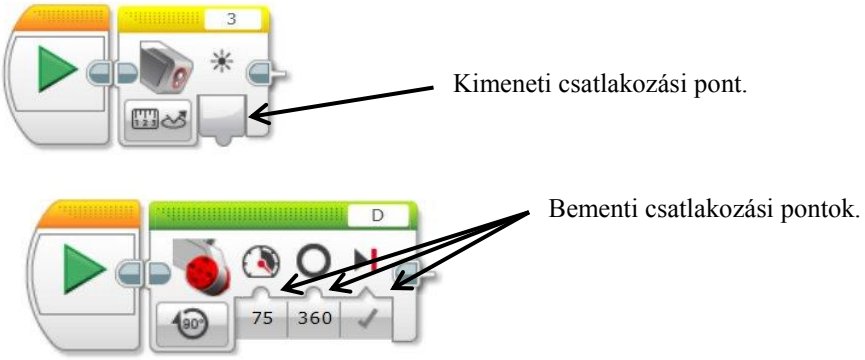
Az EV3-G programnyelvben a változóknak illetve a paramétereknek három típusa különböztethető meg:

- szám (valós)
- szöveg (karaktersorozat)
- logikai (igaz/hamis)

A számok esetében a műveleteknél a maradék nem képezhető, mivel valós osztást végez a rendszer, így két szám hányadosaként is valós számot kapunk. A maradék meghatározására külön algoritmust, programot kell írnunk.

7.2. Paraméterátadás a programon belül

A robot bemeneti szenzorai tehát különböző mért értékeket szolgáltatnak, amelyek típusa az esetek többségében szám. Ezeket az adatokat az egyes programozási modulokat szimbolizáló ikonok át tudják egymásnak adni. A blokkokon kimeneti és bementi csatlakozási pontok találhatók.
Például a fényszenzor illetve a motor esetén:

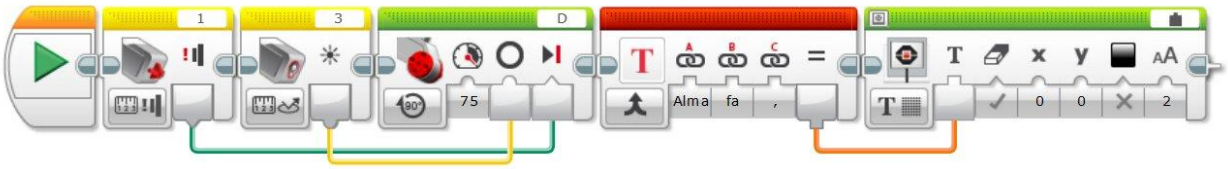


Ezeknek a csatlakozási pontoknak az összekötésével lehet a blokkok között az adatokat átadni. Az összekötés egyszerű egérhúzási művelettel végezhető. A két csatlakozási pont között megjelenik egy kábel (*wire*), amely színe utal az átadott adat típusára is. A csatlakozási pontok vizuális megjelenése is utal az adattípusra, valamint a kimeneti vagy bementi csatlakozási funkcióra is.

Az egyes csatalakozási pontok és kábelek grafikus megjelenését az alábbi táblázat foglalja össze:

Adat típusa	**Bementi** csatlakozási pont vizuális megjelenése	**Kimeneti** csatlakozási pont vizuális megjelenése	Összekötő **kábel** színe
Szám			(sárga)
Szöveg			(narancssárga)
Logikai			(zöld)

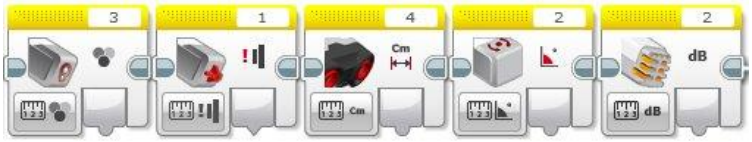
Egy példát láthatunk az alábbi ábrán különböző típusú adatok átadására:



Egy kimeneti csatlakozási pontból több huzal is kivezethető, ezért több modul számára is átadhatjuk ugyanazt az értéket. A bemeneti csatlakozási pontokba viszont legfeljebb egy huzal köthető be. Ez logikus, hiszen két bemeneti adatot ugyanazon paraméter esetén a rendszer nem tudna értelmezni, döntenie kellene, hogy melyiket fogadja el aktuálisnak.
Természetesen kábellel egy kimeneti és egy bemeneti csatlakozási pont köthető össze, két kimeneti, vagy két bementi nem. Az összekötésnek az a funkciója, hogy az egyik blokk kimenetén megjelenő értéket át tudjuk adni egy másik blokknak, értelemszerűen a bementi ponton keresztül, majd ez a másik blokk a megkapott értéket a működéséhez használhatja.

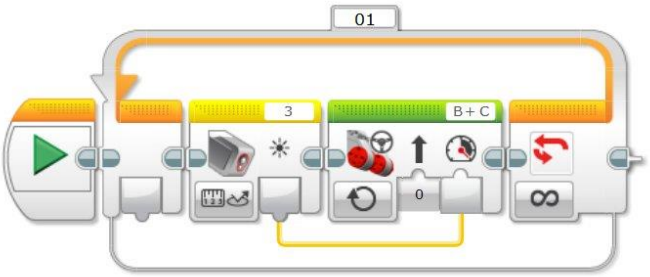
Csak azonos típusú adatokat szimbolizáló pontok köthetők össze. Eltérő adattípusok esetén a rendszer nem jeleníti meg a kábelt, hiszen az adatátadás sem lehetséges. (Ez alól látunk kivételt a képernyőkezeléssel foglalkozó fejezetben.)
A szenzorok használatának tehát van egy olyan módja is, amelynél a mért értéket használjuk fel egy másik modul valamely paramétereként. Ebben az esetben a program utasításainak végrehajtása nem vár a szenzor által mért érték valamely határának elérésére, hanem csupán kiolvassa az értéket, és lép a következő utasításra. Ha a mért értéket nem használjuk fel, akkor a modul használata felesleges, mert nem történik semmi „látható" a végrehajtás során.

A legfontosabb szenzorblokkok (az ikonok szegélye sárga):



Nézzünk egy gyakorlati programozási példát!

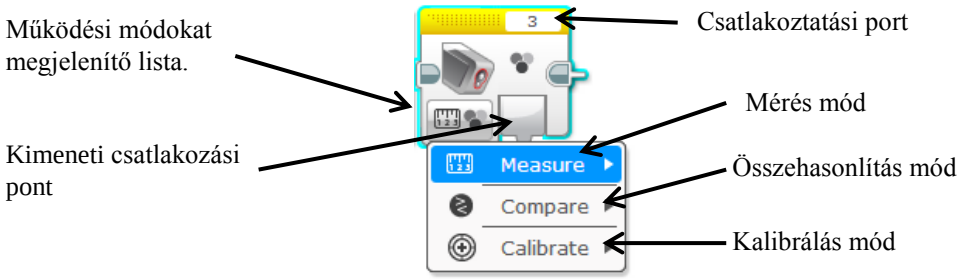
7/P1. Írjon programot, amelyet végrehajtva a robot fehér alapú pályán elhelyezett fekete színű sávok fölött halad! A robot mozgásának sebességét a fényérzékelőjével mért érték határozza meg! Ha tehát a robot fehér színű felület felett halad, akkor gyorsabban mozogjon, míg fekete színű felett lassabban! (A fehér színen mért érték nagyobb, mint a feketén mért.)



A programot kikapcsolásig szeretnénk futtatni, ezért az ikonokat végtelen ciklusba tesszük. A fényszenzor folyamatosan méri a mozgási felület fényintenzitását (*Measure – Reflected Light Intensity*). A robot sebességét kell ezzel az értékkel változtatni, ezért a fényszenzor kimeneti csatlakozási pontját kötjük össze a motorok (B és C port) *Power* bemeneti csatlakozási pontjával. Így a motor pillanatnyi sebességét mindig az határozza meg, hogy a fényszenzor mennyit mér.
Ne felejtsük el a motorok üzemmód paraméterét *On* értékre állítani, mert egyébként töredezett lesz a mozgása.

7.3. Szenzorok paraméterezése

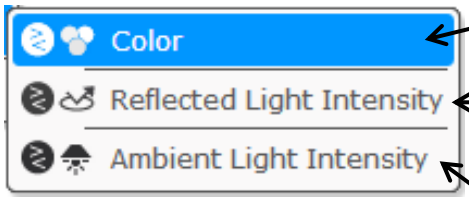
Az alfejezetben csupán néhány alapszenzor paraméterezésére térünk ki. A szoftver *Help*-jében leírtak alapján a többi szenzor működésére vonatkozó lehetőségek is könnyen érthetők.
A rendelkezésre álló szenzorok a *Sensor* programcsoportban találhatók. Legfontosabb paramétereiket a következőkben mutatjuk be.
Minden szenzor esetén először a működési módot érdemes beállítani, mert ettől függ, hogy milyen paraméterei lehetnek. A választható működési módokat a blokk ikonjának bal alsó részére kattintva egy legördülő listából választhatjuk. A kiválasztott működési módhoz további beállítások tartozhatnak, amelyek egy allistán jelennek meg.
Valamennyi szenzor esetén két alapvető működési mód van a *Measure* – Mérés és a *Compare* – Összehasonlítás. Néhány szenzornál további mérési mód is beállítható.
Az blokk jobb felső sarkában állítható be, hogy a szenzor melyik portra csatlakozik, ezt a rendszer automatikusan felismeri, ha a tégla csatlakoztatva van a számítógéphez, így beállítása ritkán szükséges.
A blokk alsó részén pedig a lehetséges kimeneti és bementi csatlakozási pontok látszanak.
Pl. a színszenzor esetén:



A kiválasztott működési mód és a beállított paraméter a blokk ikonjának bal alsó részén szimbólumok formájában is megjelenik, így vizuálisan is ellenőrizhetőek a beállított értékek.
Részletesebben a *Colour* szenzor esetén mutatjuk a választási lehetőségeket. Néhány többi szenzornál csak utalunk a specialitásokra.

7.3.1. Colour szenzor blokk

Az egyes működési módok és a beállítható értékek a *Measure* és *Compare* mód esetén:



Színszenzorként használhatjuk, 7 alapszínt különböztet meg (0-7 közötti értéket ad vissza).
„Tükröző” fényszenzor üzemmód, vörös színnel világítja meg a felületet és a visszatükrözött fény intenzitását méri (0-100 közötti értéket ad vissza).
„Sötét” fényszenzor üzemmód, kék színnel világítja meg a felületet, gyakorlatilag a környezet fényintenzitását méri (0-100 közötti értéket ad vissza).

Színszenzor üzemmódban az alapszíneket egy-egy szám formájában képes visszaadni a szenzor. Így a visszaadott értékek:

0 – nincs szín	3 – zöld	6 – fehér
1 – fekete	4 – sárga	7 – barna
2 – kék	5 – vörös	

Fényszenzor üzemmódban használva a kimeneti csatlakozási ponton 0-100 közötti érték jelenik meg. Minél világosabb vagy tükrözőbb a felület, annál nagyobb a mért érték. A visszaadott érték tehát nem kifejezetten az adott színű felületre jellemző, mert a fényesség és tükrözési sajátságok ezt befolyásolják.

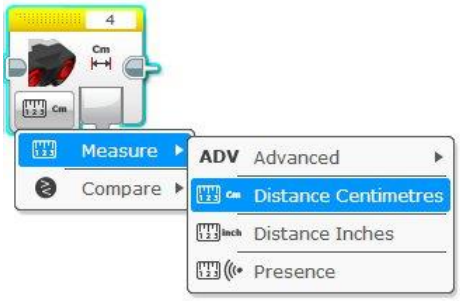
A Mérés (*Measure*) és Összehasonlítás (*Compare*) mód közötti különbség a kimeneti paraméterekben van. A mérési mód esetén egyetlen kimeneti, szám típusú érték adható át más blokknak, mégpedig az aktuálisan érzékelt szín kódja, vagy a mért fényintenzitás számértéke.

Az összehasonlító mód esetén lehetőségünk van beállítani egy feltételt, amelyet a program végrehajtása során a rendszer kiértékel, és egy logikai érték jelenik meg a kimeneti paraméterlistán, amely attól függően igaz vagy hamis, hogy a beállított logikai feltétel teljesül-e a mért értékre vagy sem. Tehát itt nem számot, hanem logikai értéket tudunk átadni más blokkoknak, amelyekkel például ciklusok, vagy elágazások feltételeit helyettesíthetjük.

Például Összehasonlító működési módban a „Tükröző" fényszenzor beállítást választva (*Reflected Light*), az ikonon látható feltétel az, hogy a mért érték kisebb, mint 50. Ennek megfelelően az első logikai kimeneten az igaz (*true*) vagy hamis (*false*) érték jelenik meg. A második kimeneti csatlakozási pontról továbbra is kiolvasható a szenzor által mért érték.

A színszenzor esetén a harmadik működési mód választási lehetőség a Kalibrálás (*Calibrate*) mód. Ezt választva a fényszenzort tudjuk programtól függetlenül kalibrálni, tehát a legkisebb, illetve legnagyobb értéket beállítani egy kiválasztott felület színei és tükrözési sajátságai alapján.

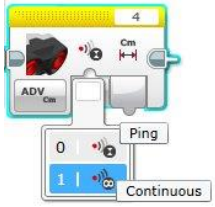
7.3.2. Ultasonic szenzor blokk



A Mérés illetve Összehasonlítás működési mód közül választhatunk.

Mérés módban a felkínált négy lehetőség:

Advanced – visszaadja a szenzor előtti akadálytól mért távolságot (cm-ben, vagy incs-ben). A blokkon megjelenő paraméterlistán, beviteli paraméterként azt is beállíthatjuk, hogy kibocsátott hang, aminek a visszaverődése alapján meghatározza a távolságot csak egyetlen jel legyen-e (*Ping*), vagy folyamatosan sugárzott (*Continuous*).

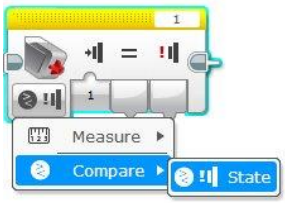


A két *Distance* kezdetű beállítással a mért távolságot cm-ben vagy incs-ben kapjuk meg.

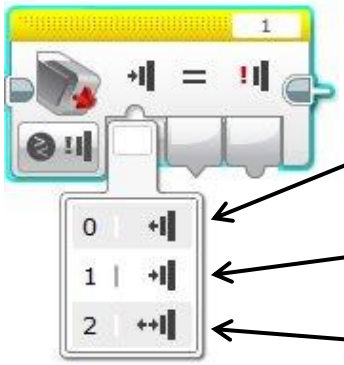
Presence – beállítás esetén nincs kibocsátott hang, a szenzor a környezetét figyeli. Képes érzékelni egy másik ultrahang szenzor (pl. egy másik robot) által kibocsátott jelet. A visszaadott érték logikai

típusú, hiszen a másik robottól mért távolságot nem tudjuk meghatározni, csak a robot jelenlétét (érzékelt-e a szenzorunk más jelet vagy sem).
A *Compare* mód funkciója megegyezik a színszenzornál bemutatottal.

7.3.3. Touch szenzor blokk



Az ütközésérzékelő esetén is kétféle működési mód közül választhatunk: Mérés vagy Összehasonlítás. Mindkét esetben egyetlen paraméterezés választható. A visszaadott érték logikai mindkét esetben. Vagyis ha benyomott állapotú az ütközésérzékelő, akkor igaz, egyébként hamis értéket kapunk.
Az Összehasonlítás módot választva megjelenik az ikonon egy bementi paraméter csatlakozási pont is. Itt szabályozhatjuk azt, hogy az ütközésérzékelő mikor adjon vissza igaz értéket.

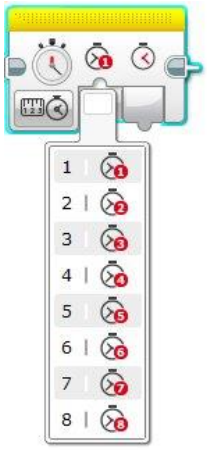


Igaz a visszaadott érték, ha felengedett állapotú az ütközésérzékelő. (*Released*)

Igaz a visszaadott érték, ha benyomott állapotú az ütközésérzékelő. (*Pressed*)

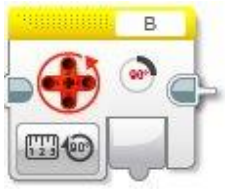
Igaz a visszaadott érték, ha változás történt az ütközésérzékelő állapotában. (*Bumped*)

7.3.4. Timer szenzor blokk



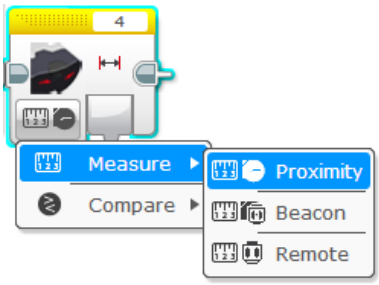
Ha programjaink működése során két esemény között eltelt időt szeretnénk mérni, akkor rendelkezésre állnak a programon belül úgynevezett *Timer*-ek, stopperek. Az események között eltelt idő mérése alkalmas arra is, hogy relatív távolságot mérjünk. Például a robot különböző szélességű sávok fölötti haladása során arra vagyunk kíváncsiak, hogy ezek közül melyik a legszélesebb, akkor állandó sebesség mellett az eltelt idő arányos lesz a megtett távolsággal, így ha megmérjük az áthaladáshoz szükséges időt, akkor a sávok egymáshoz viszonyított szélességére következtethetünk. Itt persze milliszekundum lesz a mértékegység (ami nem hosszúság mértékegység), de a mérőszám nagysága arányos a távolsággal. Tehát relatív hosszt mérhetünk.
Összesen nyolc egymástól független *Timert* használhatunk, amelyet a blokk paraméterlistáján választhatunk ki. A Mérés (*Measure*) és Összehasonlítás (*Compare*) működési mód mellett a *Reset* mód is megjelenik. Ezzel tudjuk a stoppert lenullázni, illetve előkészíteni a használatra.

7.3.5. Motor Rotation blokk



Funkcióját tekintve az ikon jobb felső sarkában beállított porta kapcsolt motort tudjuk figyelni. A beállításoktól függően a *Motor Rotation* blokk indításától (programba helyezés *Reset* állapotban) számítva az adott motor elfordulási szögét előjelesen összegezve, tengelyfordulatok számát, szintén előjelesen összegezve, illetve az aktuális sebességet tudjuk leolvasni. A blokk alkalmas a *Timer*-hez hasonlóan relatív távolság mérésére. A *Reset* funkció a *Timer*-hez hasonlóan működik. Az előjeles összegzés azt jelenti, hogy a motor fordulási szögénél figyelembe veszi a fordulás irányát is, tehát például +200°-os fordulás után, ha megváltozott a forgásirány és -90°-ot fordult a motor tengelye, akkor a blokk által mért érték +110° lesz.

7.3.6. Infrared szenzor blokk



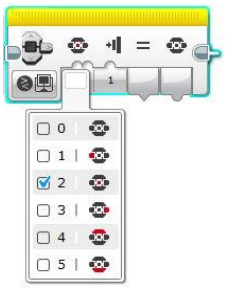
A Mérés illetve Összehasonlítás működési mód közül választhatunk. Az allistában három beállítást kínál a szoftver, amelyek közül a *Proximity* az infra jeladó távolságára utaló 0-100 közötti értéket ad vissza (minél közelebb van a jeladó, annál kisebb az érték). A blokk alkalmas az ultrahang szenzor kezelésére és távolságmérésre is. A *Proximity* beállítást választva az ultrahangszenzor által mért távolság százszorosát kapjuk eredményül.

A másik két lehetőség a *Beacon* és a *Remote* csak abban az esetben választható, ha rendelkezünk infra jeladóval, amelyet képes a szenzor érzékelni.

7.3.7. Brick Buttons blokk

A téglán elhelyezett nyomógombokat is használhatjuk „szenzorként" programjainkban. Működésüket tekintve az ütközésérzékelőhöz hasonlítanak, mivel két állapot megkülönböztetésére alkalmasak (benyomott állapot/felengedett állapot).

Mérés működési módban nem paraméterezhető a blokk, csupán egy számszerű értéket ad vissza a kimeneti csatlakozási ponton, mégpedig a megnyomott gomb sorszámát.



A blokk kimenetén visszaadott szám (nyomógomb azonosítója)	Gomb neve
0	Nincs benyomott gomb
1	Bal gomb
2	Középső gomb (ENTER)
3	Jobb gomb
4	Felfelé gomb
5	Lefelé gomb

Összehasonlítás működés módban beállíthatjuk, hogy melyik gombot figyelje a rendszer és annak állapotától függően igaz vagy hamis értéket adjon vissza. A gomb állapotai megegyeznek az

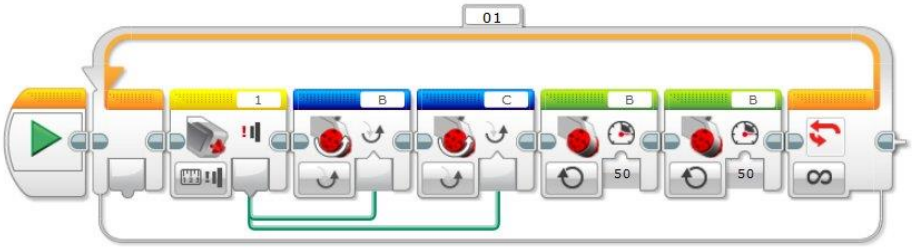
ütközésérzékelőnél látottakkal: benyomott állapot (*Pressed*), felengedett állapot (*Released*), és állapotváltozás (*Bumped*).

7.3.7. Gyro szenzor blokk

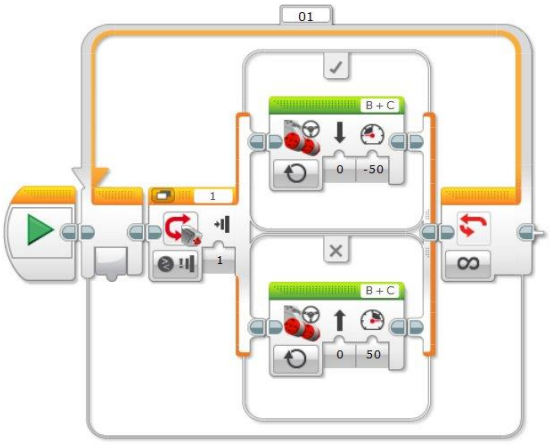
A *Wait* modulnál már bemutattuk a giroszenzor alaphasználatát, tehát hogy a robot elfordulását képes érzékelni. Még egyszer hangsúlyozva, hogy eltérően a *Motor Rotation* blokktól, nem a motorok tengelyfordulási szögét méri, hanem a robot tényleges elfordulási szögét.
Kétféle értéket képes visszaadni a program számára: elfordulási szöget, vagy elfordulási arányt, ami az elfordulás szöge fokokban, osztva az idővel másodpercben.

7/P2. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre 50-es sebességgel, ha nincs benyomva az ütközésérzékelője, és ugyanilyen sebességgel tolat, ha be van nyomva! Mindezt kikapcsolásig ismételje!

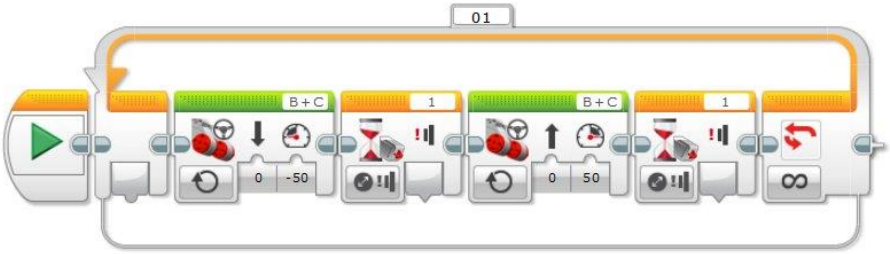
A feladat tehát nem az, hogy az ütközésérzékelő benyomására változtasson irányt, hanem az, hogy amíg be van nyomva, addig tolasson, és ha nincs benyomva, akkor haladjon előre.
A program utasításai végtelen ciklusba kerülnek. Folyamatosan figyeljük és lekérdezzük az ütközésérzékelő állapotát, és ezzel az értékkel szabályozzuk a robot haladási irányát. Az ütközésérzékelő pillanatnyi értéke logikai típusú, hiszen két állapota lehetséges: vagy be van nyomva, vagy nincs (igaz/hamis). Ezt az értéket közvetlenül nem tudjuk a motoroknak átadni, de a motor forgásirány-változtató blokkon keresztül igen (*Invert Motor*). Mivel ez a blokk egyszerre csak egy motor forgásirányát tudja megváltoztatni, ezért két ilyen blokkot kell használni és a két motort is külön-külön blokkal vezérelni. Az ütközésérzékelő kimeneti csatlakozó pontját és a két motor-forgásirány-változtató blokk bementi pontjait kötöttük össze.



A feladat megoldható paraméterátadás nélkül is, egy végtelen ciklusba helyezett feltételes elágazással, amelyet az ütközésérzékelő szabályoz. Az elágazás két szálán a motorvezérlés a sebesség ellentétes előjelű megadásával oldható meg.



Ugyanennek a feladatnak egy harmadik megoldása, amelyben nem használunk elágazást sem. A motor forgásirányát ismét az előjelek szabályozzák. Az ütközésérzékelő figyelést pedig a *Wait* modul valósítja meg. A *Wait* modulnál kiválasztva a *Touch* szenzort, a *Change* értéket állítottuk be. Ennek hatására mindaddig nem lép tovább a program utasításainak végrehajtása, míg meg nem változik az ütközésérzékelő állapota (benyomottról – felengedettre, vagy fordítva).

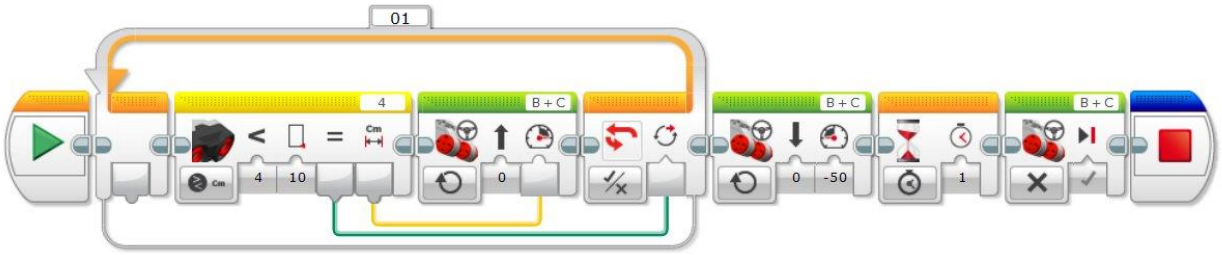


A három bemutatott példa közül bármelyik jó megoldása a feladatnak, és még továbbiak kitalálhatók.

A paraméterátadással megvalósított feladatban a motor forgásirányának feltétele tovább bővíthető, tehát újabb feltételeket építhetünk a programba. A másik két megoldás esetén ez nem lehetséges.

7/P3. Írjon programot, amelyet végrehajtva a robot az ultrahang szenzora által cm-ben mért értékkel arányos sebességgel halad egy akadály felé (folyamatosan lassulva). Ha az akadályt 10 cm-re megközelítette, akkor tolasson hátra, majd ütközésérzékelő benyomására álljon meg!

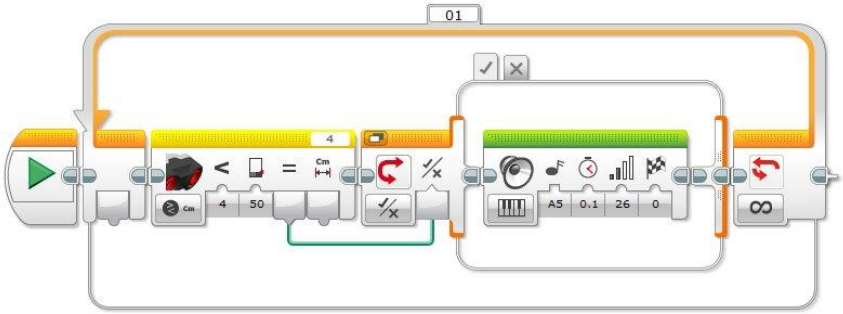
A robot motorjainak sebességét az ultrahangszenzor által centiméterben mért érték határozza meg. Tehát az ultrahangszenzor kimeneti távolságértékét kell a motor sebességparamétereként használni. Mivel a robot közeledik az akadály felé, ezért a távolság, és így a sebessége is folyamatosan csökken. Ha az ultrahangszenzor esetén az összehasonlítás (*Compare*) üzemmódot választjuk, akkor beállíthatjuk, hogy 10 cm legyen a határ. A kimeneti paraméterek közül tehát a logikai érték akkor lesz igaz, ha a robot 10 cm-re megközelítette az előtte lévő akadályt. Mindezt egy ciklusba helyezve a kilépési feltétel lehet a szenzor által visszaadott logikai érték. Ezután a tolatás és az ütközésérzékelőre történő megállás már egyszerű.



7/P4. Írjon programot, amelyet végrehajtva a robot mozdulatlanul áll! Ha az ultrahang szenzora 50 cm-en belül akadályt érzékel, akkor adjon hangjelzést! Ha nincs a távolságon belül érzékelhető akadály, akkor ne! (Mozgásérzékelős riasztó.) Módosítsuk a programot úgy, hogy ha be van nyomva az ütközésérzékelője, akkor ne adjon hangjelzést, míg ha felengedett állapotban van, akkor igen (csak, ha 50 cm-en belül van egy akadály). (A riasztót ki- és bekapcsolhatjuk.)

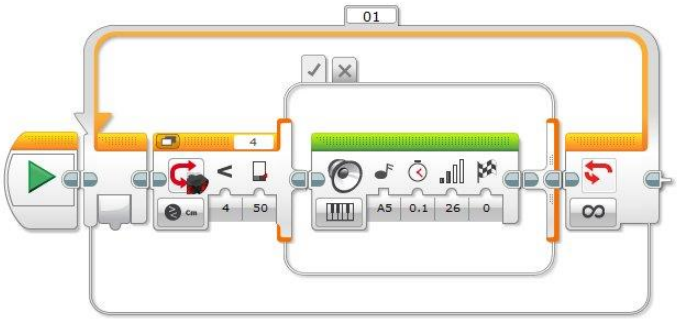
A módosítás előtti program két megoldását mutatjuk be. Az elsőnél paraméterátadással, a másodiknál anélkül.

Mindkét program ugyanúgy működik, az elvi paraméterezése is megegyezik. A programozó döntése, hogy melyik megoldást választja.



Megoldás paraméterátadással.

Az ultrahang szenzort Összehasonlító módban használjuk és határértéknek 50 cm-t állítottunk be. Ha ennél kisebb értéket mér a szenzor, akkor igaz értéket ad és megszólal az elágazás igaz ágában lévő blokkon beállított hang. A feltételes elágazás hamis ága üres, így nem jelenítettük meg.

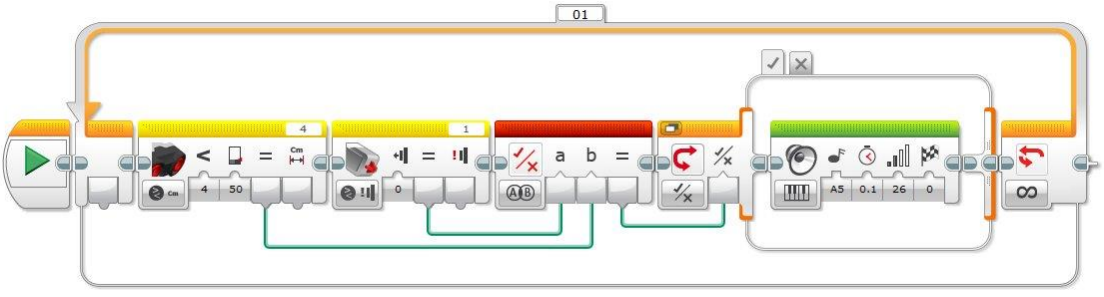


Megoldás paraméterátadás nélkül.

Paraméterátadás nélkül a feltételes elágazás vezérlő feltétele az ultrahang szenzor, amit az első megoldáshoz hasonlóan paramétereztünk.

A második megoldás egyszerűbbnek tűnik. Van azonban egy lényeges hátránya: a második esetben nem használhatunk összetett feltételt, míg az elsőben igen. Tehát ha egy újabb tényezővel szeretnénk bővíteni a hangadás feltételét, akkor az első esetben sokkal egyszerűbben megtehetjük.

A módosított feladat szerint tehát bővítenünk kell a feltételt egy újabbal, ami az ütközésérzékelő állapotát figyeli és csak abban az esetben kell hangjelzést adni, ha 50 cm-en belül észlelünk akadályt és az ütközésérzékelő nincs benyomva. Ez két feltételt jelent, amelyek között „és" kapcsolat van.
A paraméterátadással megoldott eredeti feladatot egyszerűen tudjuk módosítani, míg a másik esetben nem. A megoldásban szereplő piros szegélyű blokk a logikai és kapcsolatot jelenti, tehát a paraméterként megkapott két érték alapján akkor kerül a kimeneti pontra igaz érték, ha mindkét feltétel igaz. (A blokk részletesebb használatát lásd a későbbi fejezetekben.)



Ha a szenzorok által szabályozott feltétel egyszerű (egyetlen mért érték alapján történik a döntés), akkor célszerűbb a második programszerkezetet használni. Ha viszont a feltételt több tényező együttes kapcsolata határozza meg, akkor az első programszerkezet a jó megoldás.

7.4. Változók

A paraméterátadás tehát nagyon sok szituációban használható, azonban vannak olyan esetek, amikor nem tudjuk a mért értéket a megfelelő modulnak közvetlenül átadni. Például előfordulhat, hogy egy szenzor által mért értékre csak jóval később lesz szükség a programban és nem akkor, amikor meg tudjuk mérni. Esetleg egy mért értéket többször is fel szeretnénk használni a program különböző helyein, vagy egy szenzor által mért értékkel valamilyen műveletet kell végezni és az eredményt felhasználni úgy, hogy később szükségünk lehet a korábbi értékre is. Például ha az asztal színétől függetlenül szeretnénk olyan programot írni, ami képes egy eltérő színű felületet megkeresni. Induláskor tudjuk meghatározni az alapszínt, viszont ezt a program futása során végig használnunk kell az összehasonlításra. Ezért szükség van egy olyan eszközre, amellyel megoldhatók többek között a jelzett problémák is, tehát valamilyen módon meg tudjuk jegyezni, el tudjuk tárolni a korábban mért vagy kapott értékeket.
Minden magas szintű programnyelv lényeges elemei az úgynevezett változók. A változók alkalmasak arra, hogy adatokat tároljunk bennük. A memóriának egy kitüntetett területe a változó, aminek saját nevet adhatunk. Erre a memóriaterületre elhelyezhetünk adatokat és bármikor elővehetjük onnan őket az adott változónévvel hivatkozva rájuk. Úgy tudunk tehát programozni, hogy nem is kell ismernünk ezeket az adatokat (elegendő a nevüket megadni), hiszen elég, ha a programban szereplő utasítások „ismerik" az értékeket. A kezdeti (programfuttatás előtti) méricskélés, konstansok meghatározása tehát fölöslegessé válik, mert a robotunk is meg tudja mérni önállóan (program alapján) például az asztal színét, ezt eltárolhatja egy változóban, és el tudja dönteni, hogy egy másik helyen mért szín eltér-e ettől. Mindeközben a programozó nem is találkozik a számszerű értékekkel.
Sok más előnye is van a változók használatának és komolyabb programok esetén nem kerülhető meg a használatuk.
Az EV3-G programnyelvben a programírás során háromféle beépített változótípus használható.

Numeric – előjeles valós szám,
Text – szöveg típus,
Logic – logikai típus (igaz/hamis).

A három beépített típuson kívül további úgynevezett iterált típus is használható a szoftveren belül. A számokból képzett tömb, illetve a logikai értékekből képzett tömb.

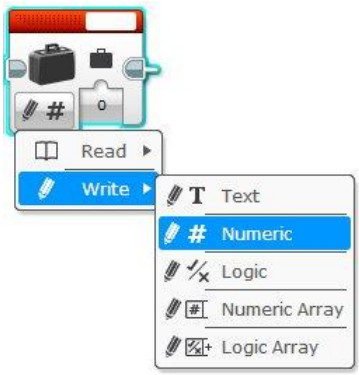
Numeric Array – számértékeket tartalmazó tömb,
Logic Array – Logikai értékeket tartalmazó tömb.

A programnyelvekben sokszor van arra szükség, hogy azonos típusú adatokból többet kell tárolnunk. Természetesen ezt megoldhatjuk úgy is, hogy minden adatnak létrehozunk egy-egy változót, de ha a tárolandó adatok száma nagy, akkor ez sok változót jelent, ami átláthatatlanná teheti a programot egyrészt, másrészt minden változónak külön nevet kell adni, ami a változók egységes kezelését megnehezíti. Például három változó esetén, ha az egyszerűség kedvéért *a*-nak, *b*-nek és *c*-nek nevezzük el őket, akkor sem tudjuk egységesen ciklusba szervezve használni, mert nem tudunk a nevekre egységesen hivatkozni. Ha három értéket szeretnénk beletenni a változókba nem tudjuk ciklussal megoldani, pedig ugyanarról a műveletről van szó (változóba írás), de a ciklusmagon belül

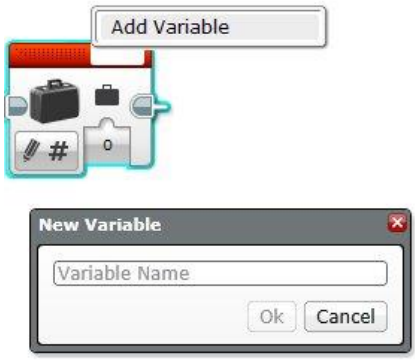
meg kellene változtatnunk a neveket minden lefutáskor. Többek között ezeknek a problémáknak a kiküszöbölésére hozták létre a tömböket. Ez azt jelenti, hogy ugyanolyan típusú változókat tudunk létrehozni tetszőleges számban és nevet is csak egyszer kell adni a tömbnek, innentől kezdve minden változó kap egy sorszámot és a *tömb neve + a sorszám* fogja azonosítani a változó tartalmát. Pl.: legyen a tömb neve *szam*. Az egyes változókra *szam[0]*; *szam[1]*; *szam[2]*; ... néven hivatkozhatunk. A sorszámozás a legtöbb nyelvben 0-val kezdődik. A tömbök szoftveren belüli használatáról még lesz szó a későbbiekben.

Az EV3-G nyelvben használt változók nevében nem használhatunk ékezetes karaktereket és szóközt. Néhány további speciális karakter használata sem megengedett. Az a javaslatunk, hogy az angol ABC kicsi és nagy betűit, valamint tagolásra a „_" karaktert használjuk! Minden esetben érdemes olyan változóneveket választani, amelyek utalnak a tárolt adatra vagy a változó szerepére, így később is könnyebben tudjuk értelmezni a programjainkat.
Egy változó kétféle állapotú lehet a programban betöltött szerepe szerint. Lehet bele írni, vagyis adatot beletenni (tárolni), illetve lehet belőle olvasni, vagyis adatot kivenni és átadni valamely másik programelemnek.
A változók használatához a *Data* csoport *Variable* (bőröndöt szimbolizáló) ikonja használható.

A programokba illesztve első lépésként azt kell eldöntenünk, hogy írni (*Write*) vagy olvasni (*Read*) szeretnénk a változó tartalmát. Erről az ikon bal alsó sarkában szereplő ikonrészletre kattintva dönthetünk. Itt tudjuk kiválasztani a változó típusát is. Általában írással kezdjük a változó használatát, hiszen valamilyen kezdő értéket szoktunk adni. A leggyakrabban a szám típusú változókat használjuk, mert a szenzoraink is ilyen típusú adatokat adnak vissza. Ha egy szám típusú változónak nem adunk kez deti értéket, akkor az alapértelmezésben 0.



Ha kiválasztottuk a változó típusát, akkor a nevét kell megadnunk második lépésként. A későbbiekben ugyanezen a néven tudunk rá hivatkozni és a benne tárolt értéket felhasználni a programban. Egy adott programban ugyanazon a néven csak egy változót lehet létrehozni. A *Variable* ikon jobb felső sarkára kattintva megjelenik az *Add Variable* felirat. Ezt kiválasztva egy párbeszédpanelen beírhatjuk a kívánt nevet (csak akkor kattintsunk rá, ha új változót szeretnénk létrehozni). Ha már több változót is létrehoztunk, akkor a jobb felső sarokra kattintva az *Add Variable* felirat alatt a megfelelő típusú változók nevei is megjelennek, így innen tudunk választani egy már létezőt.

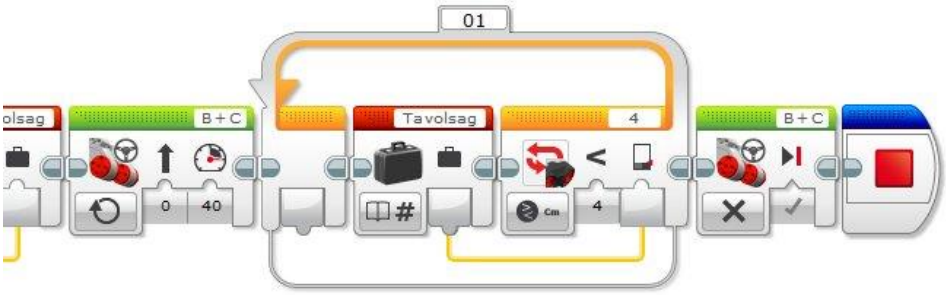


A létrehozott változó tehát állapotát tekintve kétféle lehet: olvasható vagy írható.
Az olvasás (*Read*) során az adat benne marad a változóban, nem törlődik.
Ha írásra (*Write*) állítjuk a változót, akkor értéket úgy kaphat, hogy beírjuk az ikon jobb alsó bemeneti pontjában a *Value* területen, vagy paraméterátadással. (A fentebb létrehozott szám típusú változó jelenlegi értéke 0.)
Ha egy változó állapotát írásra állítottuk, akkor nem lehet belőle olvasni. Az írásnál a változó régi tartalma elvész.
Amennyiben egy adatot eltároltunk egy változóban, akkor a program során azt bármikor elővehetjük onnan, csak arra van szükség, hogy a *Data* csoportból a változót hivatkozó ikont beillesszük a programszálra és kiválasszuk a listából a változó nevét.
Egy változóba csak olyan típusú adat tehető be, amilyen a változó típusa. Tehát például *Numeric* típusú változóba nem tehetünk szöveget.
Néhány egyszerű példán keresztül bemutatjuk a szám (*Numeric*) típusú változók használatát.

7/P5. Írjon programot, amelyet végrehajtva a robot valamekkora távolságra áll egy akadálytól (távolabb, mint 30 cm), majd 20 cm-rel közelebb megy az akadályhoz és ott megáll!

Kezdetben nem tudjuk, hogy milyen messze van a robot az akadálytól, és ez okozza a problémát. Mivel a motorokat nem tudjuk megadott távolsággal vezérelni, ezért úgy oldjuk meg a problémát, hogy kezdésként a robot megméri a távolságát az akadálytól, majd ezt eltárolja a *Tavolsag* nevű változóban. Ebből az értékből levon 10-et, és ezt szintén a *Tavolsag* változóban tárolja. Így ugyan a kezdeti mért érték elveszett (felülírtuk), de nincs is rá szükség. Most már tudjuk, hogy addig kell előre mozognia, amíg az akadálytól mért távolság nagyobb, mint a *Tavolsag* váltózóban tárolt szám.
A programot két részre bontva mutatjuk be.
Az első rész a mérést és az adat tárolását, átalakítását szemlélteti. (A matematikai művelet elvégzését lehetővé tévő modul bemutatását lásd a későbbi fejezetekben!)

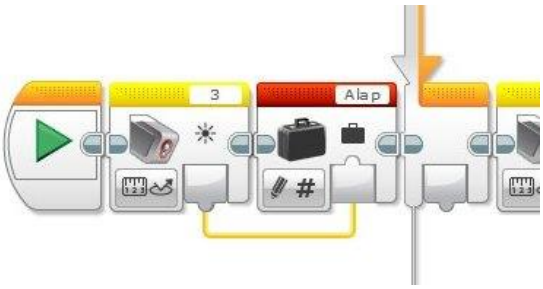
A második programrészletben kezdődik meg a mozgás. A mozgás befejezését egy feltétel fogja szabályozni, tehát *On*-ra kell állítani a motorok működési módját. A cikluson belül folyamatosan lekérdezzük ciklus kilépési feltételében a távolságot, és összehasonlítjuk a *Tavolsag* változó tartalmával.



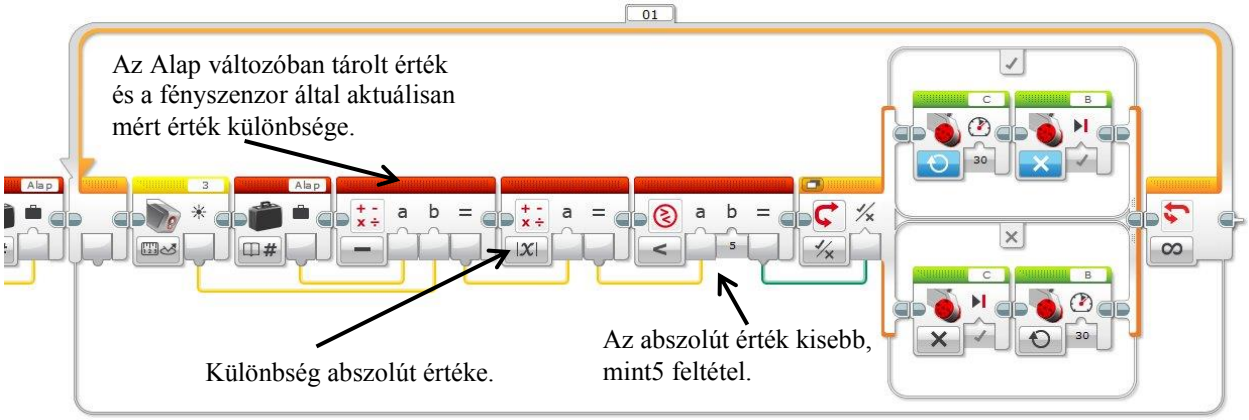
A ciklus akkor ér véget, ha a logikai érték igaz, tehát a mért érték kisebb, mint a változóban tárolt.
A programban arra is láttunk egy példát, hogyan lehet a cikluson kívüli értéket a cikluson belül felhasználni (változóban tárolva, és a változót a cikluson kívül és belül is beillesztve).

7/P6. Írjon programot, amelyben a robot egy az alapfelület színétől jól megkülönböztethető színű (legalább 5 értéknyi az eltérés közöttük) vonalat követ egyetlen fényszenzorával! Előre nem tudjuk, hogy milyen színű az alapfelület, és milyen színű a vonal.

A korábbi fejezetben bemutatott egy szenzorral történő útvonalkövetés algoritmusát olyan módon kell átalakítani, hogy bármilyen színű felületen és vonalon működjön, egyaránt alkalmas legyen fehér alapon fekete vonal vagy fekete alapon fehér vonal követésére. Ehhez még a robot elindulása előtt színmintát veszünk az alapfelület színéből a fényszenzorral, és eltároljuk egy változóban (*Alap*). Az ettől az értéktől 5-tel eltérő színt fogja a robot vonalként értelmezni. Azt viszont nem tudjuk előre, hogy az alap színe vagy a vonal színe lesz-e magasabb fényintenzitású, tehát melyik esetben fog nagyobb értéket visszaadni a fényszenzor. A két érték különbsége így lehet pozitív vagy negatív is, attól függően, hogy melyikből vonjuk ki a másikat (alapszín – aktuálisszín vagy aktuálisszín – alapszín). Ezért a vonal észlelése azt jelenti, hogy a különbségnek vagy 5-nél nagyobbnak vagy –5-nél kisebbnek kell lennie. Mivel nem tudjuk, hogy melyik a nagyobb, de azt tudjuk, hogy a különbség csak előjelben fog eltérni egymástól (hiszen pl.: 5 – 3 = 2 vagy 3 – 5 = – 2), ezért felhasználjuk a matematikai abszolút érték fogalmát. A különbség abszolút értéke (bármelyik tag is a nagyobb), mindig nemnegatív lesz. Az így képzett különbség fogja vezérelni a motorok mozgását és az útvonalkövetést.



Mintavétel az alap színéből, amelyet az Alap változó tárol.

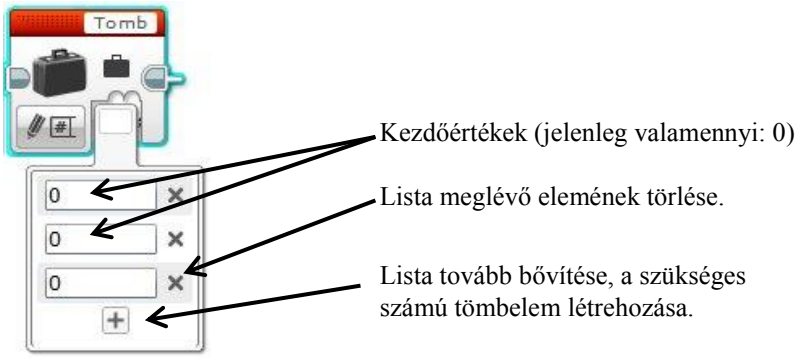


Színfüggetlen útvonalkövetés.

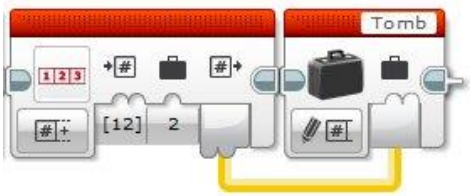
Az útvonalkövetés azon az elven működik, hogy amennyiben a fényszenzor egy határértéknél nagyobb fényintenzitást mér (esetünkben ez a határérték 5), akkor pl. balra fordul úgy, hogy az egyik motorját előre forgatja, míg a másik motor áll. Eltérő esetben a két motor szerepet cserél. Így kígyózó mozgással halad előre a robot, követve az útvonal határát, hiszen az egyik méréskor a határérték alatti, míg a másik méréskor a határérték fölötti fényintenzitás a jellemző, aszerint, hogy a fényszenzora éppen az útvonal fölött van vagy nincs.
A matematikai modulok használatának magyarázatát lásd a későbbi fejezetekben!

7.5. Tömbök

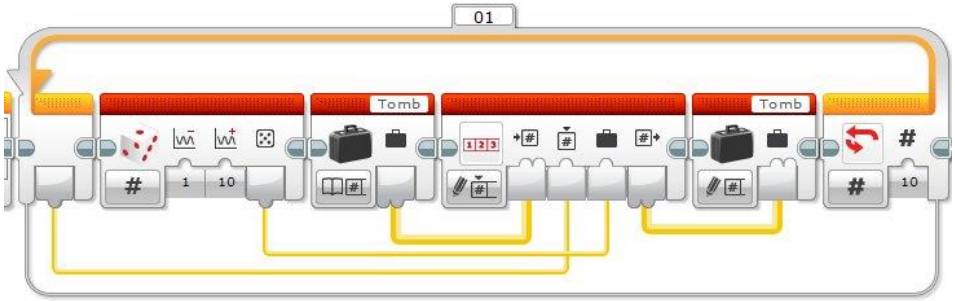
A tömbökről már esett szó a fejezet bevezetőjében. A szerepük tehát az, hogy nagy mennyiségű változót tudunk segítségükkel tárolni anélkül, hogy külön-külön létre kellene hozni őket.
A *Variabale* blokknak válasszuk a *Numeric Array* működési módját és nevezzük el a tömböt *Tomb*-nek!
A létrehozáskor érdemes a tömböt kinullázni, tehát szükséges elemszámnak megfelelően feltölteni nullákkal. A változók mennyiségének elvileg csak az EV3 memóriája szab határt.
A tömb kinullázásakor az ikon jobb alsó sarkán lévő beviteli pontra kattintva írhatjuk be a kezdőértékeket, illetve bővíthetjük a listát a megjelenő „+" jelre kattintva.



Minden tömbbe került értéknek van egy sorszáma (a tömbben elfoglalt helye), amellyel azonosíthatjuk és felhasználhatjuk a programban. Ezt a programozásban indexnek nevezik. A tömb indexelése 0-val kezdődik, tehát a tömb legelső eleme a 0. indexű, míg pl. a 3-as index a tömb 4. elemét jelenti, ...
Értéket tárolni a tömbben egy speciális blokkon keresztül lehet. Ez a *Data* csoport *Array Operations* blokkja. Itt meg kell adni, hogy mennyi a tárolni kívánt érték, illetve, hogy melyik helyre szeretnénk betenni, tehát mennyi legyen a tárolt elem indexe. Ha már volt azon a helyen érték, akkor felülíródik a régi.
A képen szereplő esetben a tömb 2-es indexű helyére (3. elem) a 12-es számot tettük be. A 12-es szám helyett a beviteli ponton (*Array In*) használhatunk változót vagy tömbváltozót is.



Az alábbi programrészlet egy 10 elemű tömböt tölt fel 1 és 10 közötti véletlen számokkal. A tömb indexének léptetésére a ciklusváltozót használtuk.



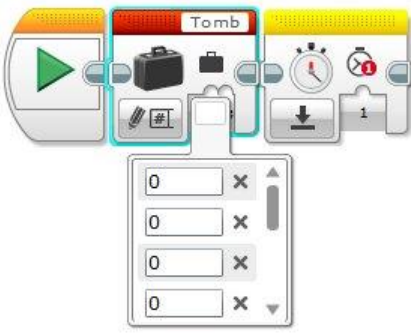
Az *Array Operations* blokknak négyféle működési módja van:

Append	Hozzáfűzés. Hatására az új elem a tömb utolsó eleme utáni helyre kerül.
Read at Index	Olvasás index szerint. Hatására a kimeneti csatlakozó pontra a megadott indexű elem kerül.
Write at Index	Írás index szerint. Hatására a megadott indexű helyre kerül az új elem, az ott lévő felülíródik.
Length	A tömb elemeinek számát adja vissza értékként. A tömb „hossza”.

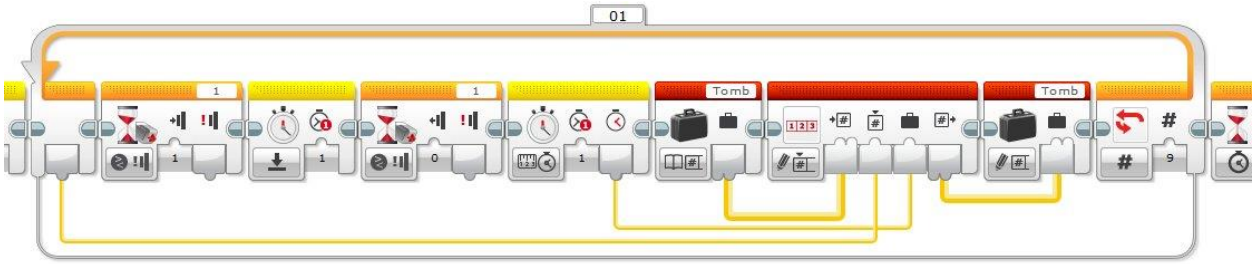
Egy példán keresztül nézzük meg a használatot.

7/P7. Írjon programot, amelyben a robot morze jeleket játszik le (hosszú és rövid hangokat)! A morze jelek időtartamát ütközésérzékelő megnyomásával lehessen beállítani! A megszólaltatott hang olyan hosszan szóljon, mint amennyi ideig az ütközésérzékelő be volt nyomva! Először az ütközésérzékelő segítségével állítsuk be, és tömbben tároljuk a lejátszandó jelsorozatot, majd a teljes jelsorozat tárolása után játssza le a robot a hangsort!

A programot kilenc jelre írjuk meg (teszteléshez: SOS). Első lépésben létrehozunk egy tömböt, amelynek kilenc elemét feltöltjük o-kal. Az ütközésérzékelő lenyomva tartásának időtartamát stopperrel mérjük (kezdésként elindítjuk a stoppert).

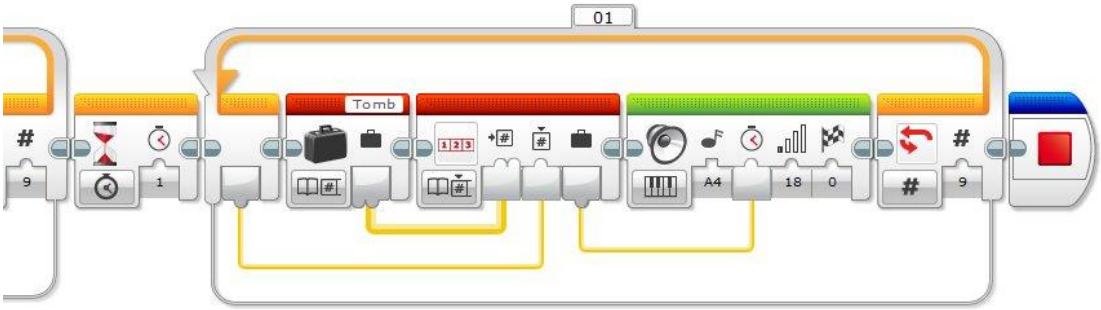


Ezután egy kilencszer lefutó ciklusban az ütközésérzékelő benyomására várunk. Ha ez megtörtént, akkor nullázzuk a stoppert és várunk az ütközésérzékelő felengedésére. A mért időtartamot az elkészített tömbben tároljuk, a tömb indexeként a ciklusváltozót használva. Az *Array Operations* blokk *Write at Index* módban van. Megfigyelhetjük a programnál, hogyan kell paraméterezni az tömbbe írást.



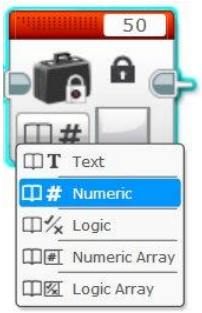
Ha lefutott a ciklus kilencszer, akkor 1 másodperces várakozás után megkezdjük a lejátszást. A tömbben tárolt értékek szolgálnak a *Sound* blokk időtartam paramétereként, a tömb elemein

történő végiglépegetést a ciklusváltozóval vezéreljük. Az *Array Operations* blokk *Read at Index* módban van, így a kimeneti csatlakozó pontján mindig az aktuális tömbelem értéke jelenik meg.



7.6. Konstansok

A változók mellett lehetőségünk van konstansok létrehozására is. A különbség a változóktól programozási szempontból annyi, hogy a konstansokban eltárolt értékek nem változtathatók meg, tehát egyszeri értékadás után a program teljes futási ideje alatt ugyanaz marad az értékük. A konstans értékét csak manuálisan lehet beállítani, programból paraméterátadással nem. Típusai megegyeznek a változókéval. Konstansokat létrehozni a *Data* menü *Constant* moduljának programba illesztésével lehet.
A konstansoknak nincs neve (mint a változóknak). Azonosításuk a beállított értékük alapján lehetséges, ezért az érték beállítása a változók elnevezéséhez hasonló. Miután kiválasztottuk a konstans típusát (ikon bal alsó sarkára kattintva), a jobb felső sarokban lévő üres mezőbe beírhatjuk az értéket.
A képen látható konstans szám típusú és a beállított értéke 50.

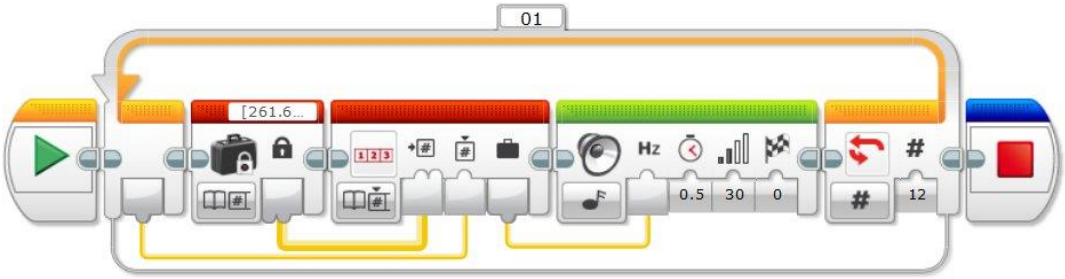


7/P8. Írjon programot, amelyben a robot lejátszik egy oktávnyi hangskálát félhangonként! Minden hangot 0,5 másodpercig megszólaltatva!

A program megírásához azt kell tudni, hogy az oktávnyi hangsor 12 félhangból áll, a C-vel (dó) kezdődően. Nem szeretnénk hosszú forráskódot írni. A hangokat megszólaltató blokkot 12-szer betehetnénk a programba egymás után, így kapnánk egy megoldást. Ha elegánsabban szeretnénk megoldani a feladatot, akkor létrehozunk szám típusú tömbkonstanst, amelyben eltároljuk a zenei hangok frekvencia értékeit.

C → 261.63	E → 329.63	G# → 415.31
C# → 277.18	F → 349.23	A → 440
D → 293.67	F# → 369.99	A# → 466.16
D# → 311.13	G → 392	B → 493.88

A változókhoz hasonlóan a konstans tömbök értékeit is tömb operátorral tudjuk olvasni.
Mindezt egy ciklusba helyezve a ciklusváltozó lesz az a számszerű érték, amely folyamatosan növekedve a tömb indexét szolgáltatja. Az így kapott számértékeket adjuk át a *Sound* blokknak, amely működési mód paraméterét *Play Tone*-ra állítjuk. Ezzel a működési móddal tudunk megszólaltatni frekvenciaértékkel megadott hangokat (Hz). A ciklusmag 12-szer kell, hogy lefusson, mivel 12 hang szerepel az oktávnyi skálán.



Ha egy másik konstans tömbben eltároljuk a hangok lejátszásának időtartamát (ami a példánk esetén minden hangnál 0,5 másodperc), és ezt kötjük a *Sound* blokk megfelelő csatlakozási pontjához (*Duration*), akkor összetett dallamokat is le tudunk játszani.

7.7. Gyakorló feladatok

7/F1. Írjon programot, amelyet végrehajtva a robot a hangérzékelője által mért értéket használja fel a motorok sebességének vezérlésére! Annál gyorsabban forogjon helyben a robot, minél hangosabb a környezete!

7/F2. Írjon programot, amelyet végrehajtva a robot előre halad, és 500 milliszekundumonként mintát vesz a fényérzékelőjével és azt kiírja a képernyőre! Mindezt 10-szer ismételje!

7/F3. Hozzon létre egy szám típusú változót és tárolja el benne a robot elindításakor a fényérzékelő által mért értéket! A robot ezután haladjon előre egyenesen mindaddig, amíg ennél az értéknél 3-mal kisebb értéket nem mér a fényérzékelő, ekkor álljon meg!

7/F4. Írjon programot, amelyet végrehajtva a robot egy az alaptól jól elkülönülő csíksor fölött halad 5 másodpercen keresztül! Öt másodperc múlva megáll és képernyőjére írja a csíkok számát, amelyek fölött áthaladt.

7/F5. Írjon programot, amelyet végrehajtva a robot kezdetben lassan halad előre (10-es sebességgel)! Ha az ütközésérzékelőjét nyomás éri, akkor a sebességét megduplázza. Mindezt addig teszi, amíg a sebessége meg nem haladja a 100-at. Ekkor ismét lecsökkenti a sebességét 10-re, és kezdi elölről a folyamatot.

7/F6. Írjon programot, amelyet végrehajtva a robot egy akadálytól tetszőleges távolságra áll és tolatni kezd! A tolatást addig folytassa, amíg az akadálytól kétszer akkora távolságra került, mint amennyire az induláskor volt! Ekkor álljon meg!

7/F7. Írjon programot, amelyet végrehajtva a robot folyamatosan lassulva közelít egy akadályhoz! Az ultrahangszenzora által mért aktuális távolság határozza meg a robot pillanatnyi sebességét!

7/F8. Írjon programot, amelyet végrehajtva a robot startpozícióból indul az ütközésérzékelőjének megnyomására, és fény szenzorával követi a fekete csíkot, amíg akadályt nem érzékel 10 cm-en belül. Ekkor megfordul és visszafelé követi a fekete vonalat. Az útelágazáshoz érve a jobb oldali vonalat követi tovább, amíg akadályt nem érzékel 10 cm-en belül. Ekkor megáll.

7/F9. Írjon programot, amelyet a robot végrehajtva startpozícióból indul egyenesen előre fekete színű csíksor fölött! A sebessége a fényszenzora által mért értéktől függően változik. Ha 15 cm távolságon belül meglát az ultrahang szenzorával egy akadályt, akkor tolatni kezd és visszafelé is végrehajtja ugyanezt a mozgást. Hátrafelé mozgását az ütközésérzékelő benyomására szakítsa meg!

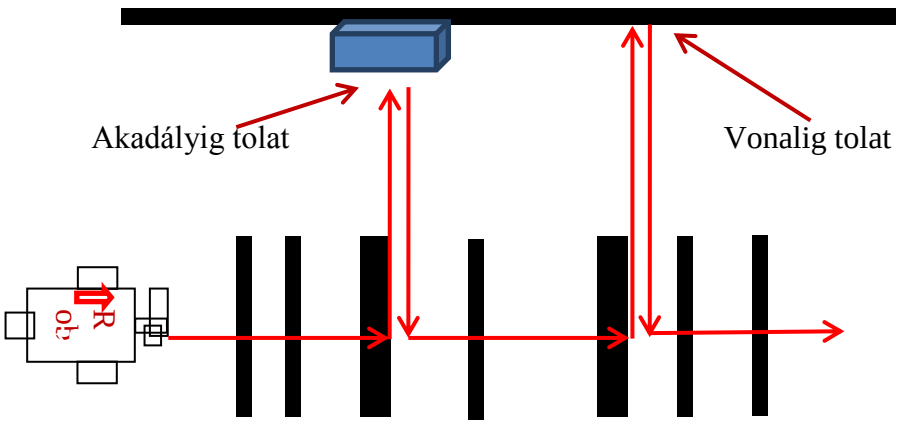
7/F10. Írjon programot, amelyet a robot végrehajtva startpozícióból indul és egyenesen előre halad fekete csíkig. Ekkor 90°-ot balra fordul és ugyanannyi távolságot halad előre, mint amennyit az indulástól a fekete csíkig megtett. Ekkor jobbra fordul 90°-ot és fekete csíkig halad előre. A csíkot elérve azt jobbra követi, míg ultrahang szenzorával 10 cm-en belül akadályt nem érzékel, ekkor megáll.

7/F11. Írjon programot, amelyet végrehajtva a robot ütközésérzékelőjének benyomásával tudjuk egy változó tartalmát egyesével növelni. A változó értéke 0-ról induljon és folyamatosan jelenjen meg a képernyőn az értéke. Ha változó értéke elérte az 5-öt, akkor ismét nulláról induljon a számolás. (Tehát 0 és 5 közötti természetes szám lehet az értéke.) A roboton található „enter” ( ) gomb megnyomása után a robot annyit sípoljon, amennyi a képernyőn beállított szám. Tehát ha a beállított szám a három, akkor a 3-at sípol. Ha nulla értéknél nyomjuk meg az entert, akkor a robot ne sípoljon. Minden sípolás azonos hangokból áll, amelyek időtartama 1 másodperc és közöttük 0,5 másodperc szünet. A programot több értékkel tesztelve is be kell mutatni.

7/F12. Írjon programot, amelyet végrehajtva a robot startpozícióból indul egyenesen előre egy fekete csíksor fölött. A harmadik csíkon történő áthaladás után körülbelül 180°-ot fordul és visszamegy az indulási pozícióba.

7/F13. A robot egyenesen halad a fehér alapszínű pályára felfestett fekete csíkokon keresztül. Kétféle szélességű csík található a pályán: a keskenyebb 2 cm-es és a szélesebb 4 cm-es. A csíkok közötti távolság nem azonos. A robot indulási pozíciójához képest az első csík keskeny. A szélesebb csíkoknál (a csíkon történt áthaladás után) a robotnak fordulnia és tolatnia kell a haladási irányához képest balra. Ezt a tolatást két esemény szakíthatja meg: vagy ha a robot hátra szerelt ütközésérzékelőjét nyomás éri (pl.: nekitolat egy doboznak), vagy ha a fényérzékelője jelzi, hogy a pályán elhelyezett, haladási iránnyal párhuzamos fekete színű csíkot érzékelt. Ezután vissza kell térnie az eredeti útvonalához és folyatatni a csíkokon keresztüli haladást. A pályán két szélesebb csík van, így az előbbi tolatást kétszer kell ismételni (minden szélesebb csík esetén egyszer). A robot tolatásának megszakításához egy esetben az ütközésérzékelőt és egy esetben a fekete csíkot kell használnia. A két eset közül bármelyikben lehet az akadály a doboz illetve a csík. A pálya végén a robotnak nem kell megállnia.

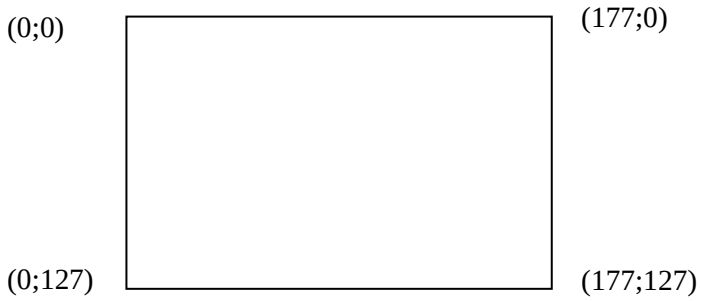
Pl.:



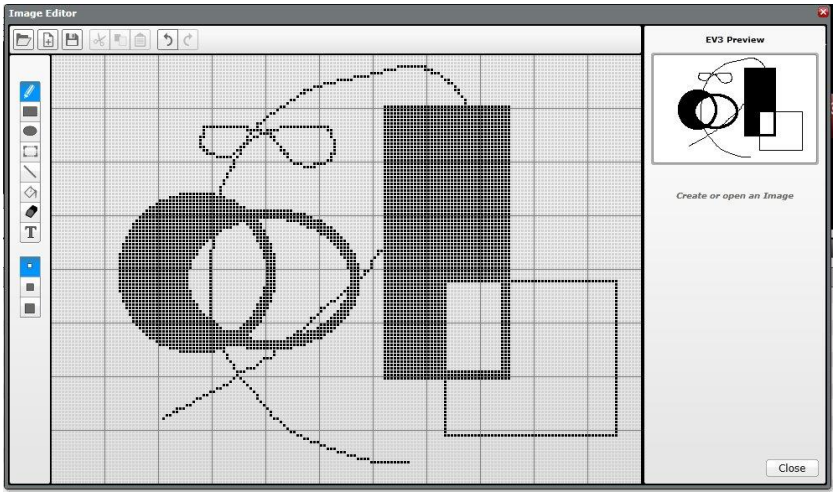
7/F14. Írjon programot, amelyet végrehajtva a robot ütközésérzékelőjének benyomásával tudjuk egy változó tartalmát egyesével növelni. A változó értéke 0-ról induljon és folyamatosan jelenjen meg a képernyőn az értéke. Ha változó értéke elérte az 5-öt, akkor ismét nulláról induljon a számolás. (Tehát 0 és 5 közötti természetes szám lehet az értéke.) A roboton található „ENTER" gomb megnyomása után a robot az ábrán jelölt startpozícióból indulva, egyenesen előre haladjon át annyi fekete csík fölött, amennyi a változóban beállított szám értéke. Ezután álljon meg és várjon 10 másodpercig a program vége előtt. Tehát ha a beállított szám a három, akkor a harmadik csík után álljon meg. Ha nulla értéknél nyomjuk meg az entert, akkor a robot ne induljon el.

8. KÉPERNYŐKEZELÉS

A programírás során szükség lehet adatok, egyszerű rajzok képernyőre írására. Ez nem csak azért fontos, mert látványosabbá teszi programjainkat, vagy mert a mérési adatok eredményeire kíváncsiak vagyunk és szeretnénk őket vizuálisan is látni (nem csak a programban felhasználni), hanem a program készítése közben kreatív képernyőhasználattal nyomon tudjuk követni a mérési-, számítási eredmények alakulását. A grafikus képernyőkezelés olyan új programozási lehetőségeket nyit meg, amely során programjaink nem csak a mérések, szenzorvezérelt mozgások elvégzésére lesznek alkalmasak, hanem a képernyőkezelésen keresztül a felhasználóval való interaktív kapcsolattartás is tovább bővülhet.
Az EV3 hardvere és szoftvere lehetővé teszi egy egyszerű, kétszínű grafikus képernyőt használatát.
A képernyő egy LCD alapú 178x128-as fekete-fehér pontmátrix grafikus megjelenítő. A bal felső sarok koordinátái a 0;0, míg a jobb alsó sarok koordinátái 177;127. Így a képernyő látható mérete 178x128 pixel.

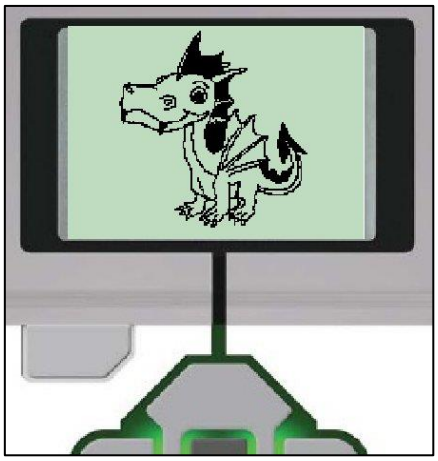


A képernyőre írhatunk szöveget, számokat, rajzolhatunk egyenest, kört, téglalapot és pontot, valamint megjeleníthetünk *rgf* kiterjesztésű képfájlt. Az alapszoftverrel rendelkezésünkre bocsátanak néhány egyszerű piktogramot, amelyet felhasználhatunk programjaink látványosabbá tételéhez (*LEGO Software\LEGO MINDSTORMS Edu EV3\Resources\BrickResources\Education\Images\files* mappában).
Saját magunk is szerkeszthetünk ilyen egyszerű képeket, ábrákat a rendelkezésre álló szerkesztőprogrammal: *Tools/Image Editor…* menüpont aktiválásán keresztül.



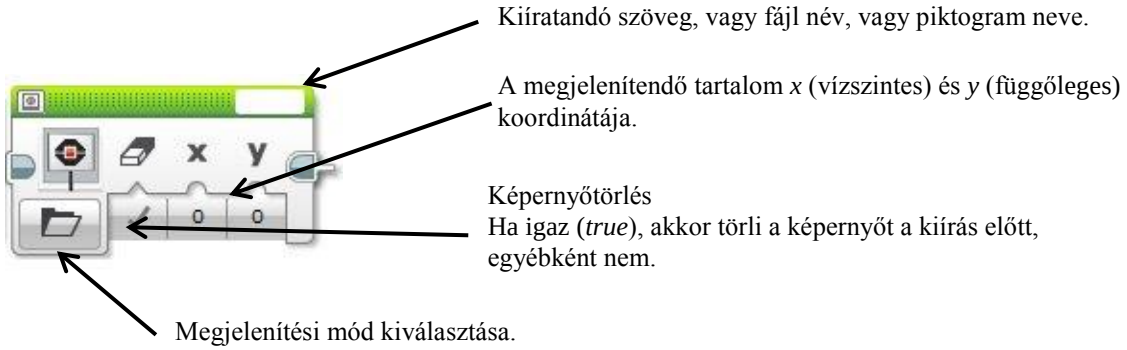
Mivel a program használata nagyon egyszerű, ezért nem mutatjuk be részleteiben. Pixelenként lehet szerkeszteni a képet két színben. Az elkészült ábrát *rgf* formátumban tudjuk menteni. Ha a program alapértelmezett képmappájába mentjük, akkor rögtön fel is használhatjuk a programjainkban.

Arra is van lehetőség, hogy a *Image Editor*-ba betöltsünk egy már létező *jpg* formátumú képet. A rendszer ezt automatikusan átméretezi és kétszínűvé alakítja. Természetesen ezzel romlik az eredeti kép minősége, de így is látványos grafikák készíthetők a képernyőre.

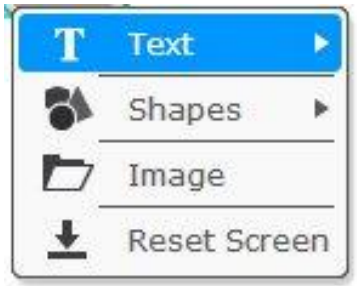


8.1. A képernyő programozása

A képernyő használata az *Action* csoporton belüli *Display* modul programba illesztésével valósítható meg. Alaphelyzetben az ábrán látható funkciók jelennek meg. Beállítva a működési módot a további lehetőségek felkerülnek a blokk ikonjára.



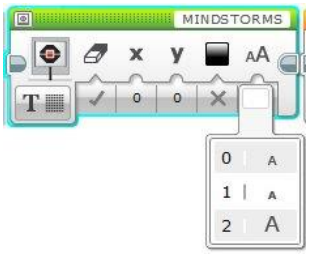
A lehetséges megjelenítési módok:



Szöveg *Pixels* vagy *Grid* beállítással szöveget jelenít meg a képernyőn.
Alakzat Egyenes (*Line*), kör (*Circle*), téglalap (*Rectangle*) vagy pont (*Point*)
Kép *rbf* kiterjesztésű, kétszínű piktogram jeleníthető meg a képernyőn.
A képernyő alaphelyzetbe állítása.

8.1.1. Szöveg kiíratása

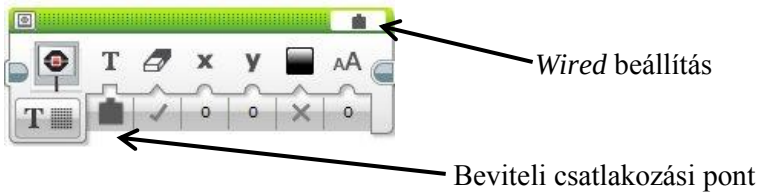
Alapvetően a képernyő szöveg megjelenítésre képes. Háromféle betűméret közül választhatunk, amelyet a *Font* paraméternél tudunk beállítani. A szöveget az ikon jobb felső sarkában lévő szövegdobozba kell beírni.



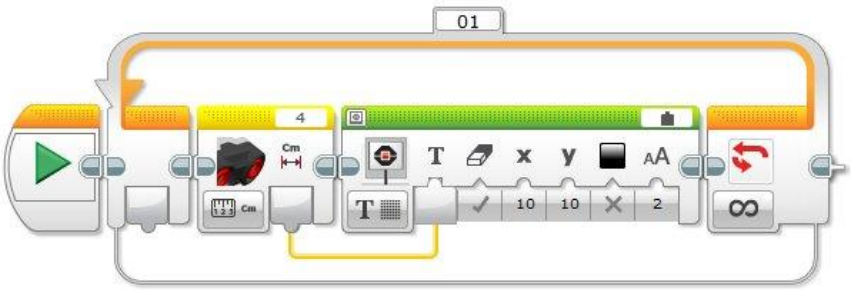
0 (*Normal*) – 9 pixel magas és 8 pixel széles egy-egy karakter.
1 (*Bold*) – 8 pixel magas és 8 pixel széles egy-egy karakter.
2 (*Large*) – 16 pixel magas és 16 pixel széles egy-egy karakter.

A megjelenés képernyőn:

A szöveg üzemmód választása esetén a *Pixels* vagy *Grid* lehetőség kínálkozik. A *Grid* funkciónál a képernyőt egy 22x13 méretű láthatatlan rácsra osztja fel a program és a karakterek az egyes cellákba kerülhetnek. Ennek megfelelően az *x* és *y* koordináták is csak ezeken a határokon belül adhatók meg.
A *Pixels* beállításnál szabadon rendelkezhetünk a kiíratandó szöveg bal felső pixelének koordinátájáról.
Csak szöveges formátumú adatokat tudunk a képernyőre íratni. Tehát a szám típusú adatokat előbb át kell alakítani szöveges típusúvá. A megjelenítés szempontjából egy számot pl. 123 is ki tud írni a rendszer szövegként. Ez nem jelent problémát csak abban az esetben, ha a kiíratandó tartalom egy szenzor által mért érték, vagy egy változóban tárolt szám. A programírás során az eltárolt adataink típusát a tárolási formátum határozza meg. Más módon tárolja a rendszer például a számokat és szöveges típusú adatokat (lásd 7. fejezet), ezért ha számszerű adatokat szeretnénk kiíratni a képernyőre, akkor azokat először szöveges formátumúvá kell alakítanunk. Az átalakítást a rendszer automatikusan elvégzi, ha a megjelenítési módot *Text*-re állítjuk és a jobb felső sarokban lévő szövegdoboznál kiválasztjuk a *Wired* paramétert. Ekkor megjelenik a beviteli csatlakozási pont, ahová paraméterátadással tudjuk az aktuális értéket betenni.



8/P1. Írjon programot, amelyet végrehajtva a robot a képernyőjére folyamatosan kiírja az ultrahangos távolságérzékelője által mért értéket! A robot a programot kikapcsolásig ismételje!



A 4-es portra csatlakoztatott ultrahangos távolságérzékelő által mért értéket kapja meg az Display blokk és a nagy méretben írja a képernyő (10;10) koordinátájú helyétől kezdődően (a kiírt érték bal felső sarka kerül az adott koordinátára). Minden kiírás előtt törlődik a képernyő és a kiíratást fekete színnel végezzük.

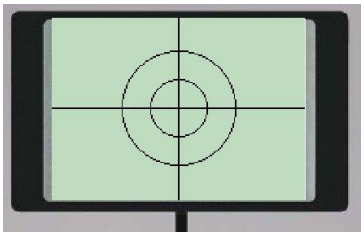
8.1.2. Alakzat rajzolása

Négyféle alakzatot használhatunk a rajzolás során. Mind a négy esetben más-más paramétereket kell megadni a rajz pozíciójának egyértelműségéhez.

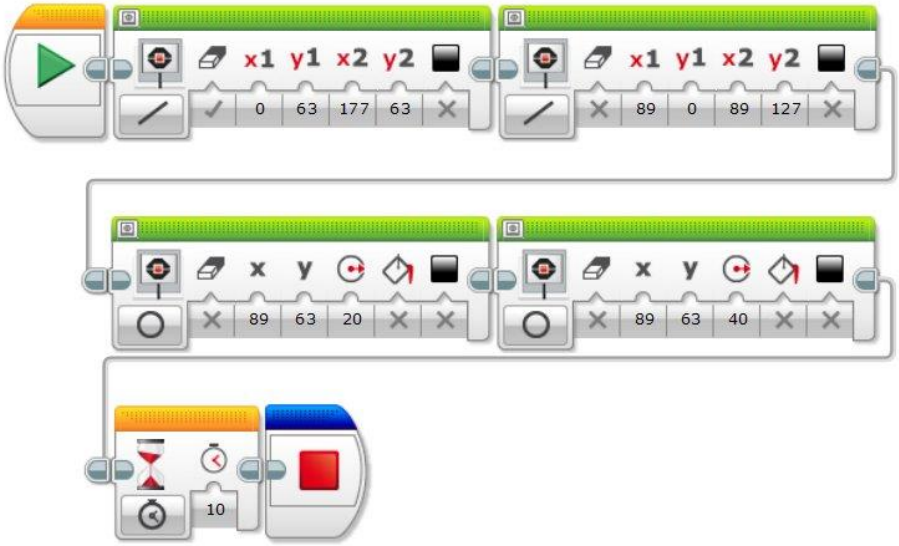
Szakasz (*Line*) – A két végpont megadása szükséges. Mindkét pont *x* illetve *y* koordinátájával.
Kör (*Circle*) – A középpont két koordinátáját és a kör sugarát szükséges megadni.
Téglalap (*Rectangle*) – A bal felső sarok két koordinátáját és a két oldal hosszát kell megadni.
Pont (*Point*) – A pont két koordinátáját kell megadni.

A két zárt alakzat esetén (kör és téglalap) arról is dönthetünk, hogy kitöltött legyen-e a rajz, tehát készíthetünk fekete színnel kitöltött objektumokat is (az egyenes esetén ez nem értelmezhető).
Valamennyi alakzat esetén megválaszthatjuk a rajz vonalszínét is, ami kétféle lehet fehér vagy fekete. Természetesen az alap képernyőszíne „fehérnek" tekinthető, így ezen a háttéren a fehér színű rajzok nem fognak látszani. A rajzeszközök egyszerű használatát mutatja be a következő két egyszerű program.

8/P2. Írjon programot, amelyet végrehajtva a robot az ábrán látható célkeresztet rajzolja a képernyőre!



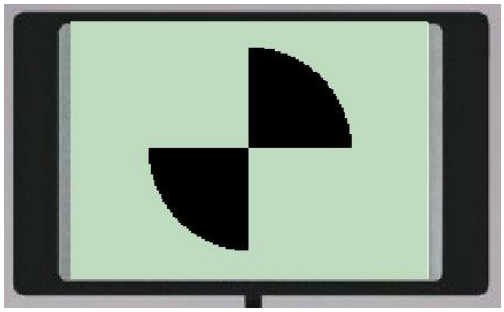
Mindezt négy rajzelemből építi fel a program. Két szakasz (*Line*) és két kör (*Circle*). Mind a négy modul esetében kikapcsoltuk a képernyőtörlést, és ötödik ikonként szerepel egy 10 másodperces várakozás, hogy legyen elég idő az ábra tanulmányozására. (Várakozás nélkül a program befejeződésekor visszaállna az alapképernyő, így az ábra rögtön eltűnne.)
A program forráskódja:



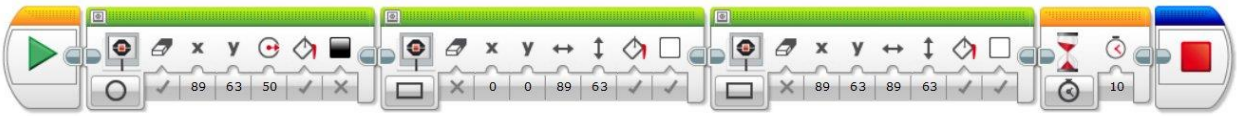
Az egyes rajzelemeknél beállított koordináták:

Vízszintes szakasz: 0;63 → 177;63
Függőleges szakasz: 89;0 → 89;127
Belső kör középpont: 89;63 sugár: 20
Külső kör középpont: 89;63 sugár: 40

8/P3. Írjon programot, amelyet végrehajtva a robot az ábrán látható alakzatot rajzolja a képernyőre!

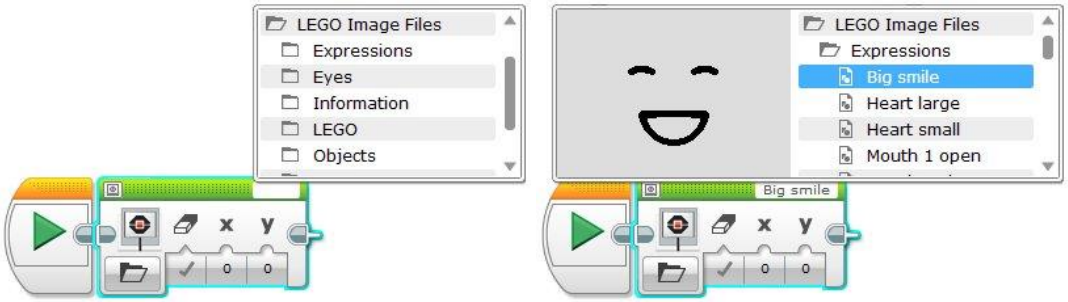


Az ábra úgy készült, hogy egy fekete színnel kitöltött kört rajzoltunk először (középpont: (89;63), sugár: 50), majd két fehér színnel kitöltött téglalapot (1. téglalap: bal felső koord.: (0;0), oldalhossz: 89, illetve 63; 2. téglalap: bal felső: (89;63), oldalhossz: 89 illetve 63).



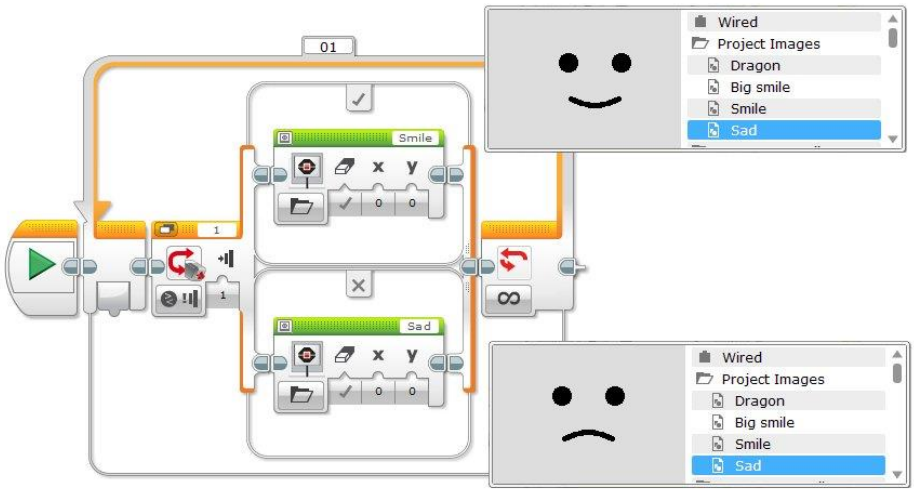
8.1.3. Képek rajzolása

A képernyőre nem csak alakzatok és szöveg kerülhet, hanem egyszerű két színnel elkészített rajzok is. Alapértelmezésben sok előre elkészített piktogram, egyszerű grafika áll a rendelkezésünkre, amelyet a képernyőre rajzolhatunk. A korábban bemutatott rajzoló eszközzel magunk is készíthetünk ilyen *rbf* kiterjesztésű képeket.
A *Display* modul beillesztésével ha a működési módot *Image*-re állítjuk, akkor ezek a rajzok az ikon jobb felső sarkában látható szövegdobozra kattintva egy listából választhatók. Az előre elkészített rajzok témájukat tekintve mappákba rendezve jelennek meg a listán. Valamelyiket kiválasztva a képről egy villámnézetet is kapunk.



A használatot egy egyszerű, de látványos programmal mutatjuk be.

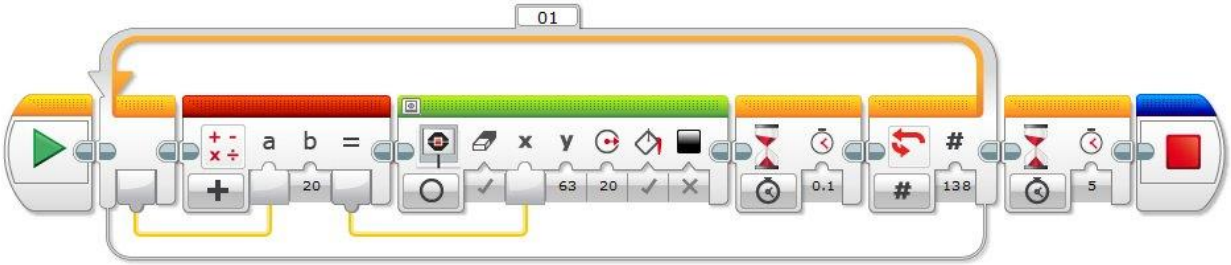
8/P4. Írjon programot, amelyet végrehajtva a robot egy mosolygó smileyt rajzol a képernyőjére, ha az ütközésérzékelője be van nyomva, és egy szomorú smileyt, ha nincs benyomva! Mindezt kikapcsolásig ismételje!



Összetettebb program esetén már komplexebben kihasználhatjuk a korábbi fejezeteknél bemutatott programozási eszközöket.

*8/P4. Írjon programot, amelyet végrehajtva a robot a képernyőjén vízszintesen mozgat egy 20 pi*xel *sugarú fekete színnel kitöltött kört! A kör a bal oldali képernyőszéltől a jobb oldali képernyőszélig* egyenletes sebességgel *(programozói beavatkozás nélkül) haladjon, majd ott álljon meg!*

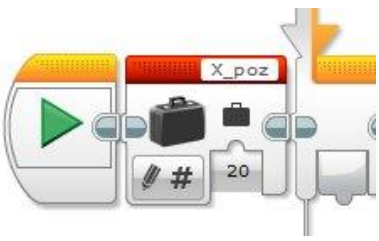
Mivel a kör sugara 20 pixel, ezért a középpontjának a koordinátája vízszintesen 19 és 157 között egyenletesen növekszik, tehát összesen legfeljebb 138 értéket vehet fel. A függőleges koordinátája pedig állandó, pl.: 63.
Egy 138-szor lefutó ciklust hozunk létre, amely minden egyes végrehajtás során eggyel növeli a vízszintes koordinátát. Erre alkalmas a ciklusváltozó, amelyhez 20-at adva (a kör középpontjának indulási pozíciója 20) már meg is kapjuk a kör középpontjának aktuális *x* koordinátáját. Ezt a paramétert adjuk át a képernyő modul bemeneti *x* paraméterének. A képernyőre rajzolás esetén mindig töröljük az előző ábrát, és a ciklus után várakozunk 5 mp-ig a program befejezése előtt. Mivel a végrehajtás nagyon gyors, ezért úgy lassítunk a kör mozgásán, hogy 0,1 mp-ig várakozunk minden egyes rajzolás után.



Természetesen a képernyőn történő mozgást egyéb szenzorokkal is vezérelhetjük. A következő program mutat erre ötletet.

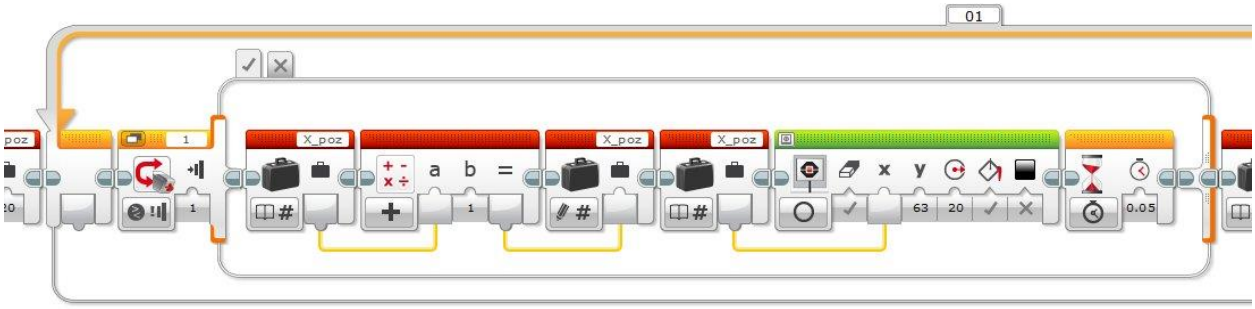
8/P5. Írjon programot, amelyet végrehajtva a robot ütközésérzékelőjének benyomásával szabályozható a képernyőn megjelenő 20 pixel sugarú fekete színnel kitöltött kör vízszintes mozgása! Ha az ütközésérzékelő be van nyomva, akkor a kör x koordinátája folyamatosan 1 pixelenként növekszik (így a kör balról jobbra mozog). Ha a kör elérte a képernyő szélét (teljesen eltűnt a képernyőről), akkor újra jelenjen meg bal oldalon és folytassa a vízszintes balról jobbra mozgást. Ha az ütközésérzékelő nincs benyomva, akkor álljon a kör az aktuális pozícióján. Mindezt kikapcsolásig ismételje!

A megoldást több részletben mutatjuk be:



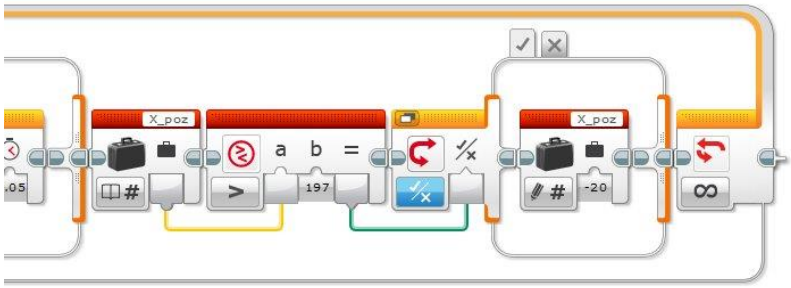
Létrehozunk egy *X_poz* nevű szám típusú változót. Ebben fogjuk a program során tárolni a képernyőn vízszintesen mozgó kör aktuális *x* koordinátáját. Mivel a kör középpontjának vízszintes koordinátája 20-nál kezdődik (így látszik a teljes kör a képernyőn), ezért az *X_poz* változó kezdőértékét a program elején 20-ra állítjuk.

Ezután indul a végtelen ciklus, amely egy elágazást tartalmaz. Ha az ütközésérzékelő be van nyomva, akkor kell változtatni a vízszintes koordinátát. Egyébként nem kell semmit csinálni, így az elágazás alsó szála üres (nem tartalmaz utasítást), az ábrán nem jelenítettük meg.



Ha az ütközésérzékelő be van nyomva, akkor az *X_poz* változó értékét eggyel növeljük. Ezt természetesen el kell tárolni az *X_poz* változóban, hogy a következő ciklusfutáskor az új értéket tudjuk tovább növelni. Az új értéknek megfelelő *x* koordináta-középpontú kört rajzoljuk a képernyőre. Minden rajzolásnál töröljük az előző ábrát. A középpont *y* koordinátája állandó, pl.: 63. A követhetetlenül gyors mozgás elkerülése érdekében egy 0,05 mp-es várakozást állítunk be, minden rajzolás után.

Ezzel a mozgás már megvalósult, de ha azt szeretnénk, hogy abban az esetben, amikor a kör elérte a képernyő jobb szélét (amikor teljesen eltűnt a képernyőről), akkor újra beússzon a bal szélen, akkor meg kell vizsgálnunk az *X_poz* értékét. Ha ez az érték nagyobb, mint 197 (177 a képernyő széle plusz 20 a kör sugara), akkor a változó értékét -20-ra kell állítanunk, így folyamatosan beúszik balról.



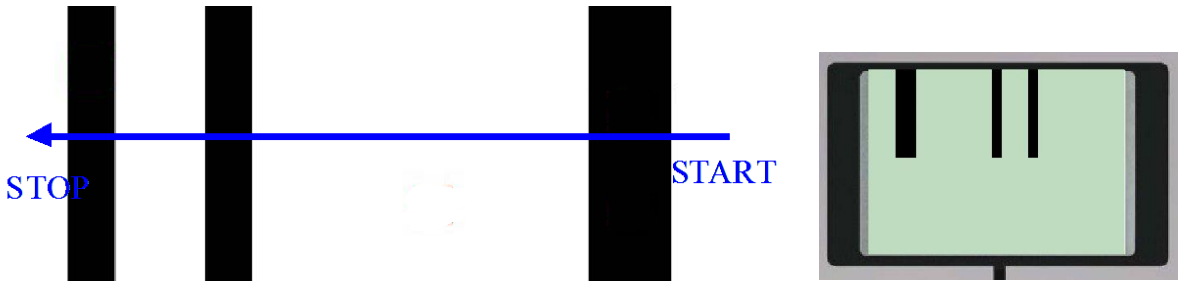
Ezzel az elágazásnak és a ciklusnak is vége.

A programot három részre darabolva mutattuk be, de természetesen ezt csak a könnyebb áttekinthetőség kedvéért tettük.

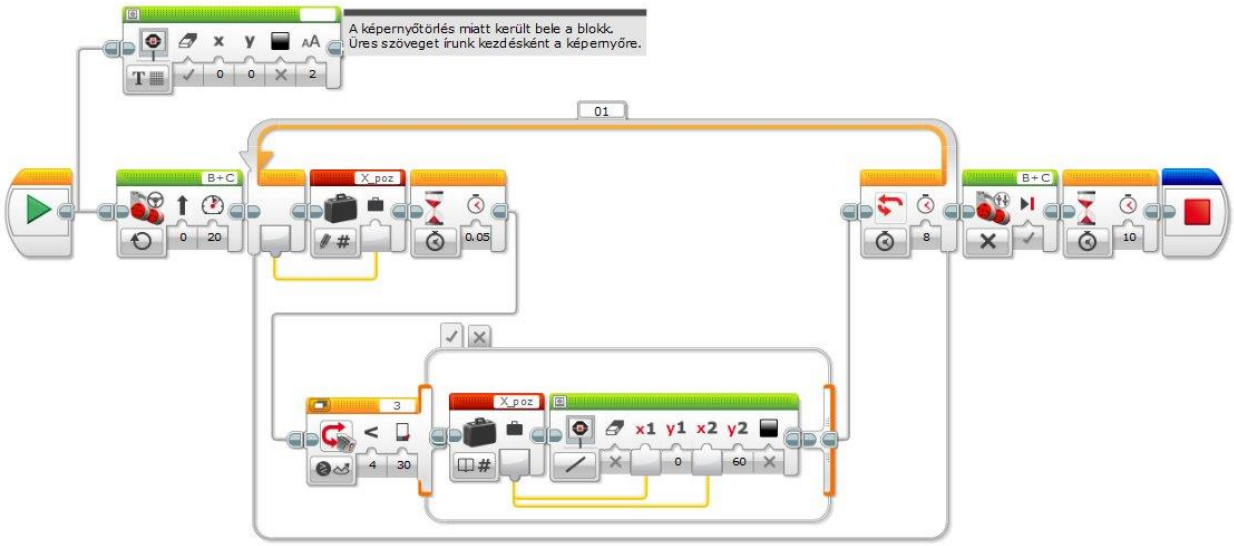
A program továbbfejlesztéseként a függőleges mozgás vezérlésére használhatunk egy másik ütközésérzékelőt, így a kör a képernyőn tetszőleges pozícióba mozgatható.

8/P6. Írjon programot, amelyet végrehajtva a robot egyenesen előre indul egy fehér felületen, és lassan (kb. 20-as sebességgel) halad egy fekete csíksor fölött 8 mp-ig. 0,05 mp-enként színmintát vesz fényszenzorával az éppen aktuális felületről. A képernyőre egy függőleges 60 pixel hosszú szakaszt rajzol, ha az aktuálisan mért szín fekete, és nem rajzol szakaszt, ha fehér. Minden szinmintavétel esetén 1-gyel nagyobb vízszintes koordinátájú ponttól kezdődően rajzolja a függőleges szakaszt. (Az első szakaszt a 0 vízszintes koordinátánál kezdi.) Tehát az időtartamokból következően összesen 160 db mérést végez, és ennek megfelelően rajzol vagy nem rajzol szakaszokat. Az 8 mp letelte után 10 mp-ig várakozik, majd utána ér véget a programja.

A pálya képe és a képernyőkép:



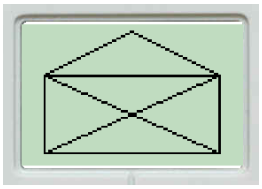
A feladat megoldása nem bonyolult, ha sikerült megértenünk az eddigieket.



A képernyőre egy függőleges, 60 pixel magasságú szakaszt rajzolunk, amelynek *y* koordinátája mindig 0 és 60, a vonal *x* koordinátája pedig minden ciklusfutáskor eggyel nő. Ezt a ciklusváltozó szabályozza és a *X_poz* nevű változóban tároljuk. Ténylegesen rajzolni csak akkor kell, ha fényszenzor által mért érték fekete. A fekete/fehér közötti határértéket a programban tapasztalati méréssel határoztuk meg, és manuálisan 30-nak állítottuk be. A feltételes elágazás hamis (alsó) ága nem tartalmaz utasítást (ha a felület fehér, akkor nem kell rajzolni), így nem jelenítettük meg. A ciklus 8 mp-ig fut. A program látványos eredményt produkál. Gyakorlatilag egydimenziós szkennerként működik.

8.2. Gyakorló feladatok

8/F1. Írjon programot, amelyet végrehajtva a robot a következő ábrát rajzolja a képernyőre:



8/F2. Írjon programot, amelyet végrehajtva a robot kiírja a képernyőre a „Benyomva” kifejezést, ha az ütközésérzékelőjét nyomás éri, és a „Nincs benyomva” kifejezést, ha nem!

8/F3. Írjon programot, amelyet a robot végrehajtva egyenesen mozog előre és a képernyőjére folyamatosan kiírja a fényérzékelője által mért értéket!

8/F4. Írjon programot, amelyet végrehajtva a robot sorsol öt 1 és 90 közötti véletlen számot, és egy más alatti sorokban kiírja a képernyőjére!

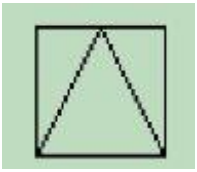
8/F5. Írjon programot, amelyet a robot végrehajtva egy 5 pixel sugarú kört mozgat átlósan a bal alsó saroktól a jobb felső sarok felé beavatkozás nélkül! (Ha a képernyő tényleges téglalap alakját vesszük alapul, akkor a feladat nehezebb. Ha egy négyzet átlója mentén valósítjuk meg a mozgatást, akkor könnyebb.)

8/F6. Írjon programot, amelyet végrehajtva a robot egy játékszimulációt valósít meg! Két ütközésérzékelővel rendelkezik. Az egyikkel a képernyőn megjelenő 2 pixel sugarú kört vízszintes, a másikkal függőleges irányban lehet mozgatni. A mozgatás ciklikus, tehát ha a kör elérte a képernyő szélét, akkor az átellenes oldalon tűnik fel. A képernyőre rajzoló modul esetén a *Clear* paraméter nincs bekapcsolva, így a mozgatással rajzolni lehet a képernyőre.

8/F7. Írjon programot, amelyet végrehajtva a robot egy akadály felé közeledik egyenletes sebességgel és 0,2 másodpercenként meghatározza az akadálytól mért távolságát! Az ultrahagszenzorával az akadálytól mért távolságértékeket jelenítse meg a képernyőre rajzolt koordináta rendszerben! A függőleges tengelyen ábrázolja távolságot, míg a vízszintes tengelyen a mintavétel időpontját!

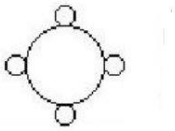
8/F8. Írjon programot, amelyet végrehajtva a robot a képernyőjére rajzol négy kört, és a körökben egy plusz jelet céltábla szerűen. Lásd ábra! A körök középpontjának koordinátái (50;32) és sugaraik rendre: 20, 15, 10 és 5 pixel hosszúak. A program vége előtt a robot várakozzon 10 másodpercet!

8/F9. Írjon programot, amelyet végrehajtva a robot a képernyőjére rajzol egy négyzetet, és a négyzetben egy egyenlő szárú háromszöget. Lásd ábra! A négyzet bal alsó sarkának koordinátái (30;12) és oldala 40 pixel hosszú. A program vége előtt a robot várakozzon 10 másodpercet!

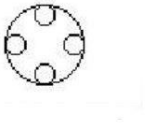


8/F10. Írjon programot, amelyet végrehajtva a robot 5 mp-ig várakozik. Ezt követően rajzolja az 1. ábrát a képernyőre, ha megnyomták az ütközésérzékelőjét és a 2. ábrát, ha nem. A nagy kör középpontja (49; 31), sugara 20 pixel. A kiskörök érintik a nagy kört és sugaruk 5 pixel. Az ábra kirajzolása után várakozzon az ütközésérzékelő megnyomásáig!

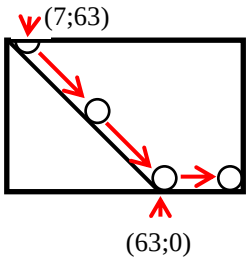
Az ütközés érzékelőt megnyomták



Az ütközés érzékelőt nem nyomták meg



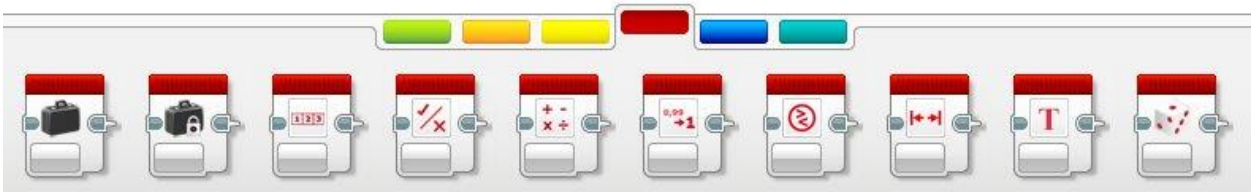
8/F11. Írjon programot, amelyet végrehajtva a robot egy 5 pixel sugarú kört mozgat megadott pályán! A kör útvonala egy 45°-os lejtővel kezdődik, majd a lejtő aljától vízszintesen folytatódik. A kör nem metszheti át a lejtőt szimbolizáló szakaszt és a vízszintes szakaszon is teljes terjedelmében látszania kell. A körvonal képe és a lejtő, illetve a vízszintes szakaszon a képernyő aktív alsó széle között ne legyen szemmel látható hézag. (Tehát a kör „legurul" a lejtőn, majd a vízszintes szakaszon halad tovább, eltekintve a gyorsulástól, egyenletes sebességgel.) Az útvonalat értelmezi a következő ábra:



9. MATEMATIKAI ÉS LOGIKAI MŰVELETEK

9.1. Műveletek csoportosítása

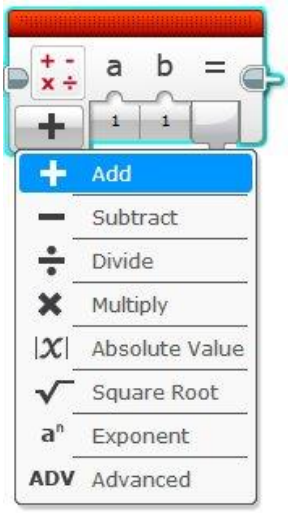
Már az eddigi programok során is előfordult, hogy néhány esetben használtuk az itt bemutatásra kerülő modulokat. Valamennyi a *Data* csoportban található. Alapvetően az adatokkal történő számolási műveletek elvégzését teszik lehetővé, valamint a kapott adatok összehasonlítását a matematikában megszokott logikai relációk segítségével.



Többféle szempont szerint is csoportosíthatjuk a modulokat. Az egyik szempont lehet az, hogy az elvégzett művelet eredménye milyen típusú. Eszerint lesznek olyan műveletek, amelyek eredménye szám pl.: összeadás, abszolút érték, és lesznek olyan műveletek, amelyek eredménye logikai érték (igaz/hamis) pl.: és, vagy, <, =, stb. A másik csoportosítási szempont lehet az, hogy hány bemeneti érték szükséges a művelet elvégzéséhez. Vannak olyan műveletek, amelyek két érték valamilyen eredményét adják vissza pl.: szorzás, >, és, … illetve amelyek egyetlen adaton fejtik ki hatásukat, és egy másik adatot adnak vissza pl.: négyzetgyök, abszolút érték, logikai tagadás. Ez utóbbi csoportosítási szempontot a szakirodalom egy illetve két operandusú műveleteknek nevezi.

9.2. Számértéket visszaadó műveletek

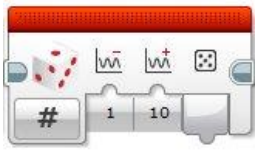
A matematikai műveletek között megtaláljuk a négy alapműveletet: összeadás, kivonás, szorzás, osztás, valamint a hatványozást (a^n *Exponent*). Ezek kétváltozós műveletek, tehát két szám között elvégzett művelet eredményeként egy újabb számot adnak vissza. Ugyanezen a modulon belül az abszolút érték, és a gyökvonás „művelete" is szerepel. Ez utóbbiak egyváltozós, tehát egyetlen számon fejtik ki hatásukat és egy újabb számot adnak eredményül. Valamennyi művelet a *Data* csoport *Math* modulján keresztül érhető el.



A blokk utolsó művelete az *Advanced* nevet viseli, amely legfeljebb négy szám típusú érték összegzését végzi el. Tehát alapértelmezésben A + B + C + D összeget számítja ki, ahol A, B, C, D tetszőleges számok. Nem feltétlenül szükséges minden értéket megadni, mert alapértelmezésben o szerepel valamennyi esetben, így ha nem adunk meg számot, akkor az nem befolyásolja az összeget. A művelet változóit lehet konkrét számok beírásával, vagy paraméterátadással is konkretizálni.

Az ikon bal alsó sarkán kattintva tudjuk egy legördülő listából kiválasztani a szükséges műveletet, és vagy paraméterátadással vagy közvetlenül a szövegdobozokba írt számok segítségével megadni azokat az adatokat, amelyek eredménye a modul kimeneti csatlakozópontjáról olvasható be egy változóba vagy adható át más modulnak.

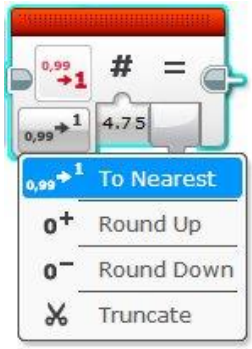
A lehetőségek: összeadás (*Addition*), kivonás (*Substraction*), szorzás (*Multiplication*), osztás (*Division*), abszolút érték (*Absolute Value*), négyzetgyök (*Square Root*), hatványozás (*Exponent*), valamint egy összetett művelet (*Advanced*), amit fentebb bemutattunk.
További számértéket visszaadó „művelet", a véletlen számsorsoló, amelynek kétféle működési módja van.



Numeric módban paraméterátadással vagy a szövegdobozokba beírt számokkal megadhatjuk, hogy milyen tartományon belül állítson elő véletlen számot, amely a blokk kimeneti csatlakozási pontján jelenik meg. A képen látható beállítással 1 – 10 közötti számot sorsol, egyenlő valószínűséggel (az 1-et és 10-et is sorsolhatja).

Logic módban a visszaadott érték logikai (igaz/hamis közül az egyik). Bemenetként 0-100 közötti érték állítható be és ez jelenti az igaz válasz valószínűségét. Tehát ha 25-re állítjuk a bementi értéket, akkor 25% valószínűséggel ad vissza igaz (*true*) és 75% valószínűséggel hamis (*false*) értéket a blokk.

Egy operandusú „művelet", tehát egyetlen bemenő paramétere van kerekítést végző blokknak.



Négyféle működési mód közül választhatunk.
To Nearest – Ez felel meg a tényleges matematikai kerekítésnek, tehát 5 tizedtől felfelé, egyébként lefelé kerekít és mindenképpen egész szám az eredmény.

Round Up – Egész számot ad eredményül, ami a bemeneti értéknél nagyobb vagy vele egyenlő. Akkor nagyobb, ha a bemeneti szám rendelkezik nem nulla tizedes résszel.

Round Down – Egész szám az eredmény, a bementi érték tizedes rész nélküli egészrésze pozitív számok esetén. Negatív számoknál az egész résznél kisebb egész az eredmény.

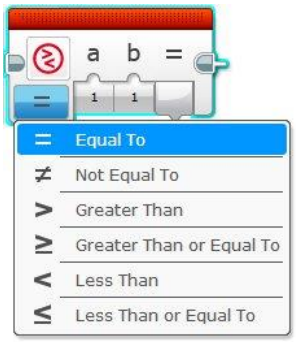
Truncate – Csonkolás, ahol megadhatjuk a tizedes jegyek számát is. Az eredmény a megfelelő tizedesjegy számmal rendelkező szám. A bemeneti értékből a többi számjegy elhagyásával keletkezett.
Egy táblázatban összefoglalva a négy működési mód közötti különbségeket:

Bemenet	**To Nearest**	**Round Up**	**Round Down**	**Truncate (tizedesjegyek száma: 1)**
4,23	4	5	4	4,2
4,5	5	5	4	4,5
4,87	5	5	4	4,8
-4,2	-4	-4	-5	-4,2
-4,83	-5	-4	-5	-4,8

9.3. Logikai értéket visszaadó műveletek

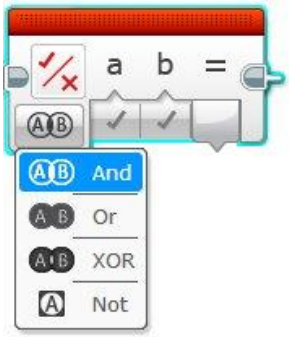
Szintén a *Data* csoporton belül találhatók a logikai értéket visszaadó műveletek, több modulra bontva.

Összehasonlító műveletek (*Compare*)



A legördülő lista elemeit értelmezik a matematikából már megszokott jelölések.
Például a *Greater Than or Equal To* kifejezés a nagyobb vagy egyenlő magyar nyelvi és matematikai megfelelője.
Valamennyi művelet két bemenő adatot igényel, amelyek típusa szám kell, hogy legyen.

Logikai műveletek (*Logic*)



A működési mód listából négy logikai művelet választható ki: ÉS (*And*), VAGY (*Or*), KIZÁRÓ VAGY (*Xor*) és a logikai TAGADÁS (*Not*). A műveletek jelentését a matematika halmazelmélet területéről ismert Venn-diagramos szemléltető ábra is segíti. A *Not* művelet egy bemenő adatot igényel, míg a másik három kettőt-kettőt. A bemenő adatok minden esetben logikai típusúak kell, hogy legyenek, és a visszaadott érték is logikai.

A kétváltozós logikai műveletek két logikai értéket visszaadó feltétel összekapcsolását teszik lehetővé. A két logikai feltétel értékétől függően vagy igaz vagy hamis eredményt szolgáltatnak. A logikai műveletek működését értéktáblázatokkal szokás szemléltetni. Az összekapcsolt két feltételt „A feltétel”-ként és „B feltétel”-ként szerepeltetjük az alábbi táblázatokban. Mindkét esetben igaz vagy hamis lehet a bemeneti érték. Ennek megfelelően az összekapcsolt feltétel eredményét a szürke háttérszínnel kiemelt négy cella tartalmazza.

ÉS (*AND*)		A feltétel	
		Igaz	*Hamis*
B feltétel	*Igaz*	**Igaz**	**Hamis**
	Hamis	**Hamis**	**Hamis**

Az „és” kapcsolat esetén akkor lesz az összetett művelet eredménye igaz, ha mindkét feltétel igaz volt, egyébként hamis.

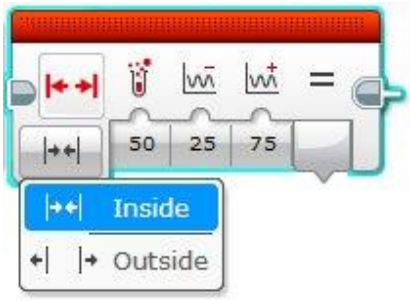
VAGY (*OR*)		A feltétel	
		Igaz	*Hamis*
B feltétel	*Igaz*	**Igaz**	**Igaz**
	Hamis	**Igaz**	**Hamis**

A „vagy" kapcsolat esetén akkor lesz az összetett művelet eredménye hamis, ha mindkét feltétel hamis volt, egyébként igaz.

Kizáró VAGY (*XOR*)		A feltétel	
		Igaz	*Hamis*
B feltétel	*Igaz*	**Hamis**	**Igaz**
	Hamis	**Igaz**	**Hamis**

A „kizáró vagy" kapcsolat esetén akkor lesz az összetett művelet eredménye igaz, ha a kiinduló feltételek értékei különbözőek voltak, egyébként hamis.
A legördülő listából választható negyedik művelet a tagadás (*Not*). Ez egyváltozós művelet, egyetlen bemenő értéket igényel és annak az értékét ellentettjére változtatja.

Intervallum művelet (*Range*)

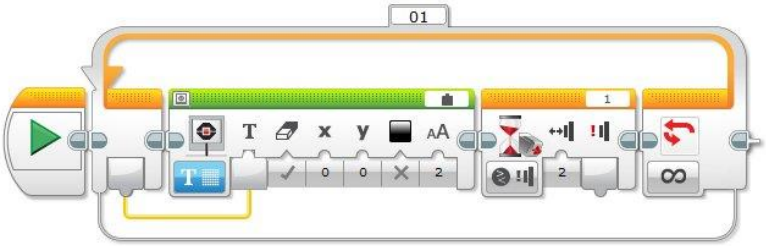


Két bemenő határadatot igényel a használata. Azt vizsgálja, hogy egy paraméterként kapott szám a megadott határadatokon belül (közöttük) vagy kívül van-e. Eredményül logikai értéket ad vissza aszerint, hogy melyik működési módot választottuk. A két lehetőség: *Inside* (belül) vagy *Outside* (kívül).
Inside esetén a visszaadott érték akkor lesz igaz, ha a megkapott paraméter a határadatok között van. A vizsgált számot paraméterátadással vagy közvetlenül a szövegdobozba írva adhatjuk meg. A képen szereplő példánál a visszaadott érték igaz (*True*), mert a 25 < 50 < 75 (a tesztadatként használt 50 a 25 és 75 között van).

Outside esetén éppen fordítva, tehát az ábrán szereplő példánál hamis (*False*) értéket kapunk vissza, mivel a bevitt érték 50, ami nem esik kívül a 25-75 tartományon.
A modul használatával az abszolútértékhez hasonló programszerkezetek is létrehozhatók.

9/P1. Írjon programot, amelyet végrehajtva a robot a képernyőjére egyesével növekvő számokat ír ki (0-tól kezdve)! Az ütközésérzékelő benyomására írjon ki mindig eggyel nagyobb számot! Minden kiírás előtt törölje a képernyőt, és kikapcsolásig ismételje mindezt!

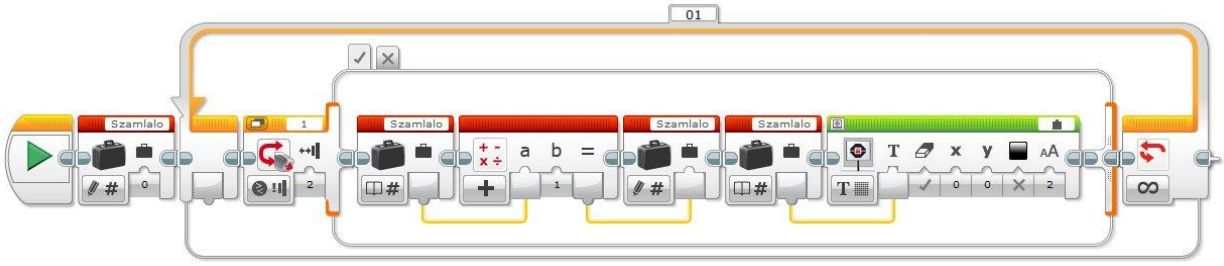
A programot kétféleképpen készítettük el. Az első változatban nem használtunk matematikai műveleteket, hanem a ciklusváltozó értékét használtuk fel a számok egyesével növekedő sorozatának előállításához.



A ciklusváltozó értékét írattuk ki a képernyőre. Az újrakezdés előtt egy ütközésérzékelőre történő várakozást állítottunk be. Ezen a modulon csak akkor lép túl a program, ha benyomjuk az érzékelőt. Az ütközésérzékelő paraméterezésénél figyelni kell arra, hogy a bementi paraméter értékét

Bumped-re (2-es érték) állítsuk. Ennél a paraméterértéknél nem az érzékelő benyomására ad igaz értéket a modul (lép túl rajta), hanem a dupla állapotváltozás hatására: tehát ha benyomott állapotból kiengedett állapotba kerül a szenzor. Ha nem ezt a paramétert használjuk, hanem a szokásos *Pressed* (benyomva, 1-es érték) módot, akkor a képernyőn a számok szemmel követhetetlen sebességgel növekszenek, mert amíg be van nyomva az ütközésérzékelő, addig a ciklus folyamatosan elölről kezdődik (fut), hiszen benyomott állapotban nem vár a program a *Wait* modulnál. A program futási sebessége viszont emberi léptékkel mérve elég nagy ahhoz, hogy ne tudjuk olyan rövid ideig benyomva tartani a szenzort, hogy csak egyszer fusson le a ciklus. (Érdemes kipróbálni, mert a ciklusok futási sebességéről kaphatunk információt.)

Ennél a programnál tehát nem kellett használnunk a matematikai műveleteket. A második megoldás ugyanerre a feladatra már hasonlít a programozásban megszokott egyik alapalgoritmusra. Ez a szabvány megoldási ötlet a megszámlálási algoritmusoknál. Vagyis egy feltétel teljesülése esetén eggyel növeljük egy előre létrehozott változó értékét. Jelen esetben a feltétel az ütközésérzékelő benyomása. Természetesen itt is a *Bumped* módot (2-es érték) kell használni.



Egy változóban (*Szamlalo*) tároljuk folyamatosan a képernyőre kiíratandó számokat (0 kezdőértéktől).

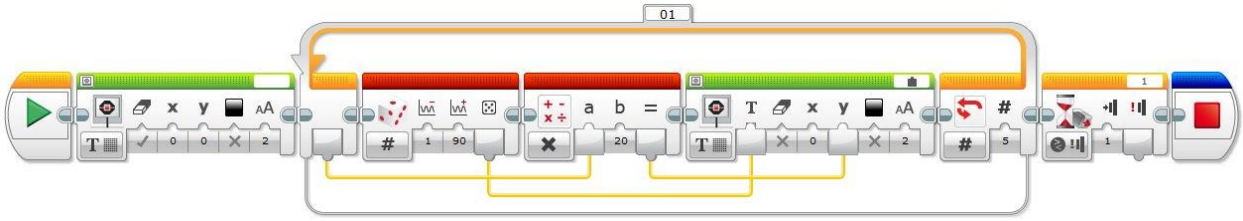
Ha az ütközésérzékelőt benyomtuk és felengedtük, akkor eggyel növeljük a változó értékét és kiírjuk a képernyőre. A megnövelt változó értékét el is kell tárolni a *Szamlalo* változóban, mert a következő ciklus lefutáskor már ezt az új értéket kell tovább növelnünk.

A képernyőn megjelenő első szám így nem a 0, hanem az 1, mivel még kiíratás előtt növeltük a változó értékét. Ha nullával szeretnénk kezdeni a kiíratást, akkor például a ciklus előtti értékadásnál a *Szamlalo* változó kezdőértékét -1-re kell állítani, vagy a kiíratás után növelni az értékét.

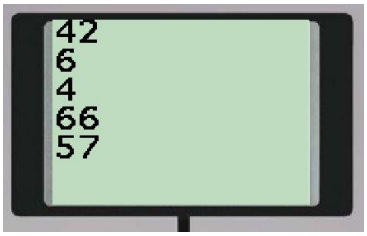
A második megoldás annyival általánosabb, hogyha nem 0-tól kell kezdeni a számok kiírását, akkor a kezdő értékadással ez egyszerűen korrigálható, vagy ha nem egyesével kell növelni a megjelenő számokat, akkor összeadásnál egyetlen paraméter módosításával megoldható a feladat.

9/P2. Írjon programot, amelyet végrehajtva a robot sorsol 5 db 1 és 90 közötti számot, és a számokat egymás fölötti sorokban írja a képernyőre! A program az ütközésérzékelő benyomására álljon le! A kisorsolt számok lehetnek egyenlők is.

A véletlen szám sorsolásánál a számtartomány minimális értéke 1, maximális értéke 90. A sorsolás után a képernyőre írás történik meg, képernyőtörlés nélkül. A ciklus 5-ször fut le.



Az egymás alatti sorokba íratást úgy oldottuk meg, hogy a ciklusváltozó értékét megszoroztuk 20-szal és az így kapott számot használtuk a képernyőírási pozíció y koordinátájaként. Az x koordináta minden esetben 0 maradt. Tehát a megjelenő számok y pozíciói rendre a ciklusváltozó 20 szorosai: 0x20 = 0; 1x20 = 20; 2x20 = 40; 3x20 = 60; 4x20 = 80.
A program a ciklus lefutása után az ütközésérzékelő benyomására várakozik.

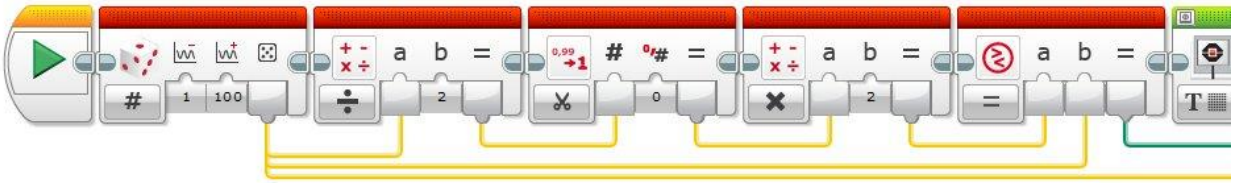


9/P3. Írjon programot, amelyet végrehajtva a robot sorsol egy 1 és 100 közötti véletlen számot, majd a képernyőre írja a számot és alá azt, hogy páros vagy páratlan! A program várjon 5 mp-ig majd álljon le!

Az EV3-as szoftverváltozat már kezeli a tizedes törteket is. Tehát az osztás eredménye nem egész szám. Az NXT 1-es eszközhöz készült *firmware* esetén az osztás után még mindenképpen egész szám volt az eredmény, mert a tárolásnál a tizedes részt a rendszer elhagyta. Így egyszerűen kihasználható volt ez a tulajdonság a páros/páratlanság eldöntésénél (pl.: ha a szám = 9, akkor 9/2 = 4,5, de ebből csak 4-et tárol a rendszer, így 4x2 = 8, ami kisebb, mint az eredeti 9, páros számok esetén a kettővel való osztás utáni visszaszorzás az eredeti számot adja eredményül).
Általános matematikai megfogalmazással:

szám = hányados * osztó + maradék

Tehát az eredeti szám egész osztás esetén éppen a maradékkal tér el a hányados és az osztó szorzatától. Ha ez a maradék 0 (vagyis a szám osztható), akkor nincs eltérés. Ezt használhatjuk ki az oszthatóság vizsgálatánál.
Tehát ha a számot elosztom kettővel, majd a hányadost megszorzom kettővel, akkor a kapott eredmény vagy kisebb lesz, mint az eredeti szám (páratlan eset) vagy vele egyenlő (páros eset). Egyszerű összehasonlítással eldönthető a kiindulási szám paritása.
Az algoritmus működéséhez szükséges, hogy egész osztást végezzünk, tehát ne tizedes tört legyen az eredmény. Hiszen ha csak olyan osztásunk van, ami tizedes törtet ad eredményül, akkor 9/2 = 4,5, amit hiába szorzunk vissza 2-val 2x4,5 = 9. Tehát semmiképpen nem kapunk eltérést.
Az EV3 programkörnyezetben nem áll rendelkezésre olyan modul, ami az osztás utáni maradékot lenne képes kiszámítani. Az osztás eredménye is tizedes tört formában áll a rendelkezésre. Viszont a kerekítés blokkon belül akár a *Truncate* (0 tizedesjegyre állítva), akár a *Round Down* működési mód (pozitív számok esetén), *Round Up* (negatív számok esetén) alkalmas a hányados egész részének meghatározására. Ezt használjuk fel az algoritmus előállításához.



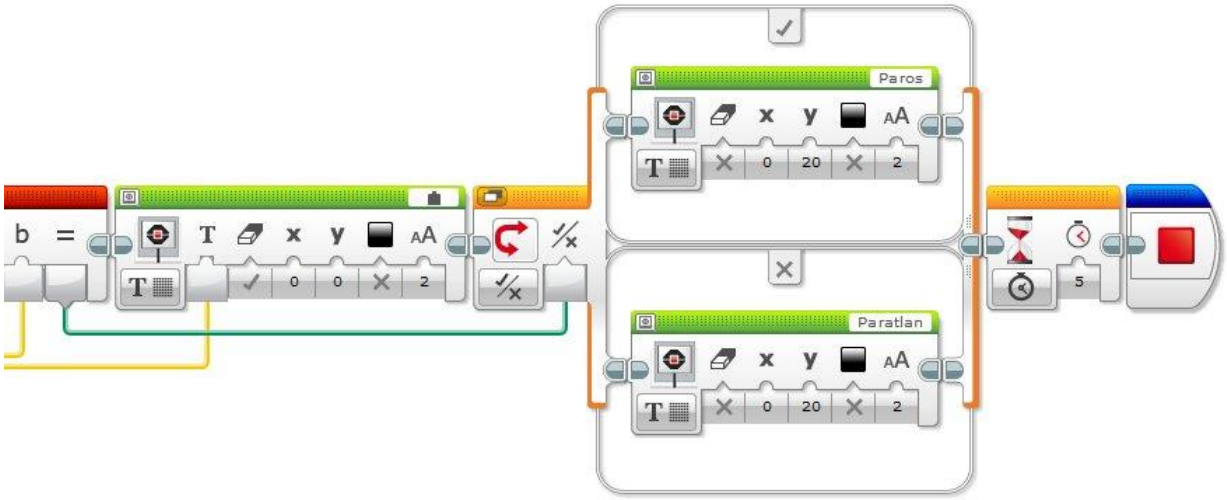
Első lépésben sorsoltunk egy 1-100 közötti számot. Ezt elosztottuk 2-vel. A hányadost csonkoltuk (*Truncate*) 0 tizedes jegyre. Az így kapott értéket megszoroztuk 2-vel, majd az eredményt összehasonlítottuk az eredeti számmal. Ha egyenlők voltak, akkor igaz (*True*) értéket kaptunk vissza. Ez teljesül minden páros szám esetén.

pl.: 8/2 = 4 → a 4-et csonkolva az eredmény 4 → 4x2 = 8 → Ez pedig megegyezik az eredeti számmal.

Páratlan számoknál az összehasonlítás eredménye hamis (*False*).

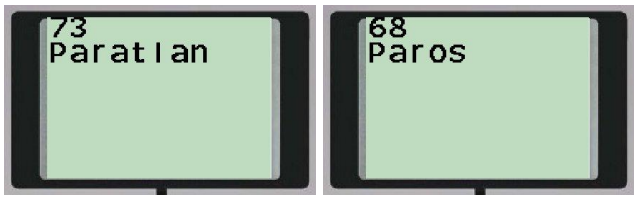
pl.: 9/2 = 4,5 → a 4,5-et csonkolva az eredmény 4 → 4x2 = 8 → Ez pedig nem egyezik meg az eredeti számmal, ami 9 volt.

A megkapott logikai érték használható egy elágazás feltételének.



Az eredeti szám kiíratása után az előbb bemutatott feltételtől függően írjuk a képernyőre a „*Paros*" vagy „*Paratlan*" szavakat.

A képernyőkép:



Érdemes észrevenni, hogy ha a csonkolt eredményt kivonjuk az eredeti számból és a különbséget szorozzuk 2-vel, akkor éppen a maradékot kapjuk.

Pl.:

11/2 = 5,5 → a csonkolt eredmény: 5 → 5,5 – 5 = 0,5 → 0,5x2 = 1.

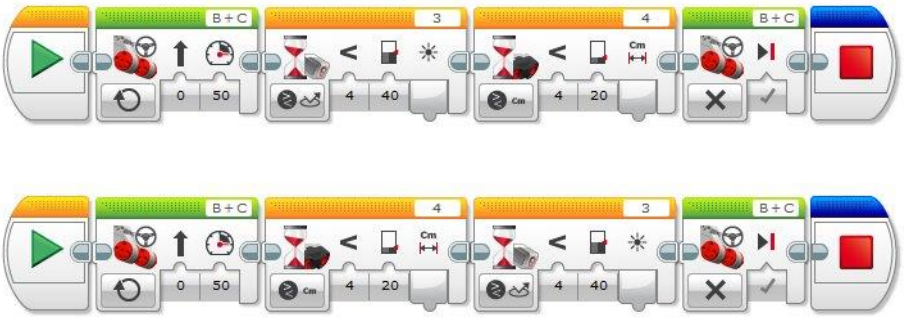
vagy

16/2 = 8 → a csonkolt eredmény : 8 → 8 – 8 = 0 → 0x2 = 0.

Természetesen ez az algoritmus nemcsak a kettővel való osztás utáni maradékot képes meghatározni, hanem analóg módon bármely számmal történő osztás utánit is.

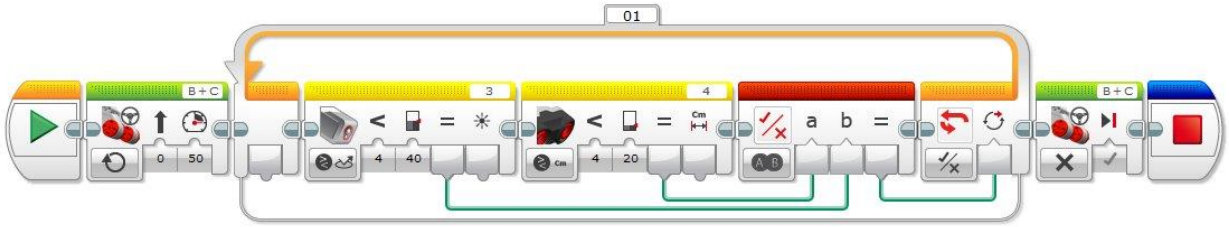
9/P4. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre egy fehér színű felületen, és megáll, ha fényszenzora fekete színű csíkot észlel, vagy ultrahangszenzora által mért érték alapján 20 cm-nél közelebb került egy akadályhoz!

A robot mozgását egy összetett feltétel vezérli. Két szenzor (fény és ultrahang) értékét kell folyamatosan figyelni és akkor kell abbahagyni a mozgást, ha legalább az egyik esetén teljesül a példában megadott feltétel. A 7. fejezetben már utaltunk az ilyen összetett feltételekkel történő programvezérlésre. Ha a *Wait* modult használjuk a szenzorok figyelésére, akkor a programunk nem fog helyesen működni. Kezdeti mérések alapján a fehér-fekete közötti határértéknek a fényszenzor által mért 40-es értéket állítottuk be (fekete 28, fehér 52, a két szám számtani közepe 40). Ha 40-nél kisebb értéket mér a szenzor, akkor a felület fekete.



A problémát mindkét esetben az okozza, hogy a beillesztett két *Wait* modul sorrendje kötött és mindkettőnek teljesülni kell (ráadásul a megadott sorrendben), ahhoz, hogy a program futása során továbblépjen a végrehajtás. Pl.: Az első esetben, ha a robot nem halad át fekete csík fölött, de 20 cm-nél jobban megközelít egy akadályt, akkor nem áll meg, hiszen a program a fényszenzor által mért megfelelő értékre vár.

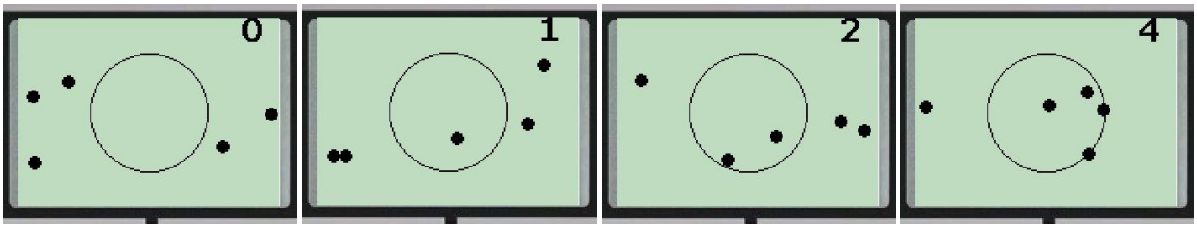
Az összetett feltételből következően a *Data* csoport *Logic* modulját kell használnunk. A két szenzor figyelését megvalósító modulokat „VAGY" (*Or*) feltétellel összekötve, az eredményül kapott logikai értéket használjuk egy ciklus kilépési feltételének vezérlésére.



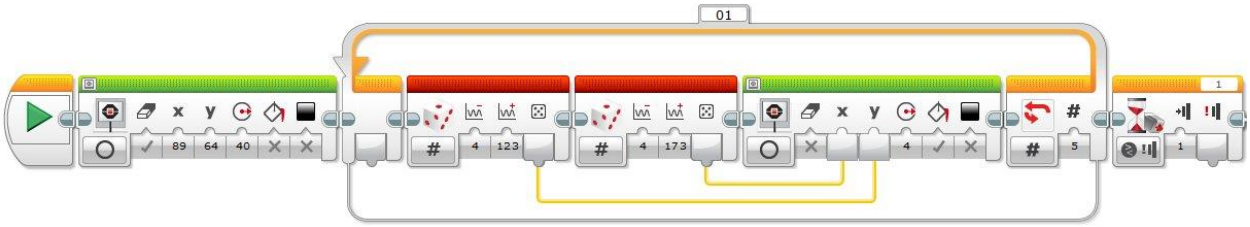
A ciklusból akkor lépünk ki, és áll meg a robot, ha vagy a fényszenzor mér 40-nél vagy az ultrahangszenzor mér 20 cm-nél kisebb értéket, vagy mindkettő együtt következik be.

9/P5. Írjon programot, amelyet végrehajtva a robot egy célba lövést szimulál véletlen számok sorsolásával! A képernyő közepére rajzoljon 40 pixel sugarú kört, ez lesz a céltábla! Sorsoljon két véletlen számot, amely a lövés találati helyének x és y koordinátáját jelenti! Jelenítse meg a találat helyét a képernyőn, egy fekete színű 4 pixel sugarú körrel! Összesen 5 lövést jelenítsen meg a képernyőn (minden lövés látszódjon). Ha a találati helyet jelző 4 pixel sugarú kör a céltáblát jelölő körön belül van, vagy legalább érinti, akkor a lövés célba talált. Számolja meg, hogy hány találatot ért el és írja ki a számot a képernyő jobb felső sarkába! A program ütközésérzékelő megnyomására álljon le!

A képernyőkép különböző találatszámok esetén:



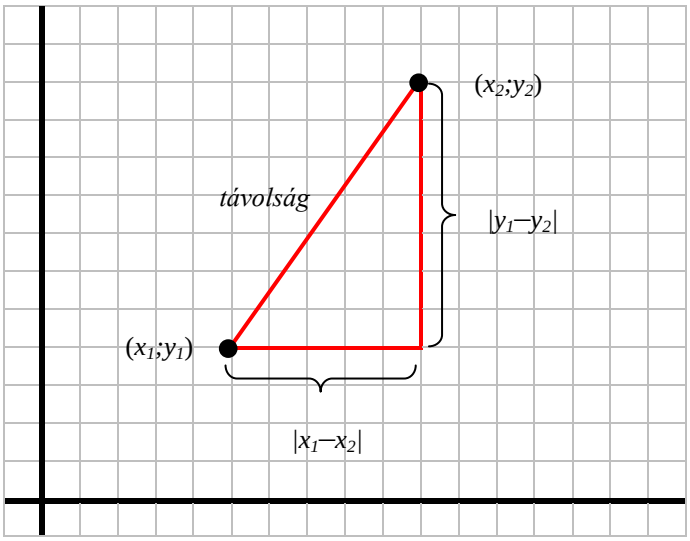
A program megírását kezdjük azzal, hogy a céltáblát felrajzoljuk a képernyőre. Ennek középpontja legyen a (89; 64) koordinátájú pont, sugara 40 pixel. Ezután sorsoljunk két véletlen számot! Mivel a találati pontokat 4 pixel sugarú körök jelölik, ezért ha azt szeretnénk, hogy a találat teljes egészében látszódjon a képernyőn a középpont vízszintes koordinátáit a 4-173; a függőleges koordinátáit a 4-123 tartományból kell sorsolni. A megjelenítést egy *Display* blokkal végezzük, amelyet alakzatrajzoló módban, *Circle* (kör) funkcióval használunk. A kör sugara 4 pixel, a képernyőtörlést kikapcsoltuk és beállítottuk az alakzat fekete színnel történő kitöltését.
Mivel öt lövést kell megjelenítenünk, ezért a sorsolást és megjelenítést egy ötször lefutó ciklusba tettük. A program az ütközésérzékelő megnyomására áll le.
Mielőtt folytatnánk a program írását teszteljük le a jelenlegi változatot. A megjelenítés már működik.



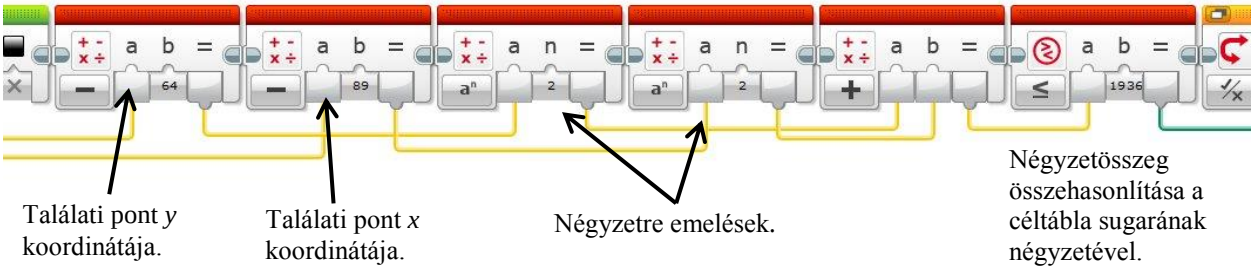
A következő probléma amit meg kell oldani, hogy hogyan határozza meg az algoritmus, hogy a találati pont benne van-e a körben. Ehhez Pitagórasz tételét fogjuk használni. Ha a koordináta rendszerben két pont távolságára vagyunk kíváncsiak, akkor a matematikai összefüggés, amelyet használnunk kell:

$$távolság = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2},$$

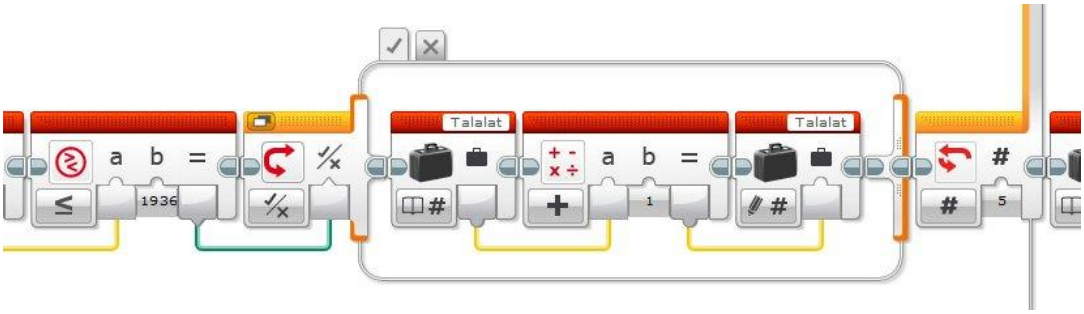
ahol az egyik pont koordinátái (x_1;y_1), a másik ponté (x_2;y_2). A kivonás sorrendje a négyzetre emelés miatt mindegy (csak előjelben különbözne az x_1-x_2 az x_2-x_1-től).



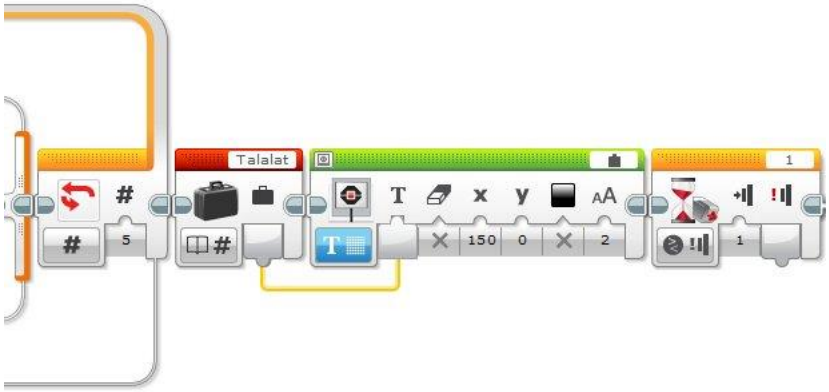
A találati pont tehát akkor van a céltábla körén belül, ha kör középpontjától mért távolsága nem nagyobb a kör sugaránál. Ugyanezt úgy is megfogalmazhatjuk, hogy a kör középpontjának és a találati pont megfelelő koordinátái különbségének négyzetösszege nem nagyobb, mint a sugár négyzete.
Mivel a találati pontot 4 pixel sugarú kör jelzi, ezért ha azt is találatként szeretnénk számolni, amikor a lövés széle érinti a kör szélét, akkor a céltábla kör sugarát 4 pixellel nagyobbnak kell választani. Így a 44 pixeles kör sugarának négyzete: 1936.
A matematikai műveleteket kiszámító kódrészlet tehát:



Az összehasonlítás eredménye egy logikai érték. Hamis, ha a pont kívül van a céltábla körén, egyébként igaz.
A következő lépés a találatok számlálása. A programozásban használt megszámlálás algoritmusának ötletét használjuk. Egy nulla kezdőértékű változó, értékét növeljük eggyel, ha egy feltétel teljesül, egyébként nem csinálunk semmit. A feltétel most az előző matematikai számolás eredményeként kapott logikai változó. Ez vezérli a feltételes elágazást, amelynek a hamis ágában nem szerepel utasítás.



A megnövelt értéket el kell tárolnunk újra az eredeti (*Talalat*) változóban, hogy a következő növeléskor már az új értékkel tudjunk számolni.
A ciklus ötszöri lefutása után a *Talalat* változóban a találatok számának megfelelő érték lesz, amelyet a képernyő megfelelő pozíciójára írathatunk.



9.4. Gyakorló feladatok

9/F1. Írjon programot, amelyet végrehajtva a robot a képernyőn kettesével növekedő számokat jelenít meg (a régi számot mindig törli a képernyőről)! A számok növekedését az ütközésérzékelő benyomása szabályozza.

9/F2. Írjon programot, amelyet végrehajtva a robot a képernyőn egyesével növekedő számokat jelenít meg tízig (a régi számot mindig törli a képernyőről), majd utána egytől újra kezdi a számlálást! A számok növekedését az ütközésérzékelő benyomása szabályozza.

9/F3. Írjon programot, amelyet végrehajtva a robot sorsol 1 és 100 közötti véletlen számot, majd egymás alatti sorokba kiírja a számot, az 5-tel történő osztás utáni hányados egész részét és az osztás utáni maradékot!

9/F4. Írjon programot, amelyet végrehajtva a robot véletlenszerűen sorsol 1 és 100 közötti véletlen számot! Ha páros számot sorsolt, ütközésig tolat, és visszatér a kiindulási fekete vonalig! Ha páratlan számot sorsolt, akkor előremegy 2 mp-et, majd visszatér a kiindulási fekete vonalig. Mindezt végrehajtja összesen 6-szor, és a kisorsolt számokat a képernyőre írja egymás fölötti sorokba!

9/F5. Írjon programot, amelyet végrehajtva a robot véletlenszerűen sorsol két 0-100 közötti számot és kiírja a képernyőjére a két számot, valamint, hogy melyik a nagyobb (első vagy második), esetleg egyenlők-e!

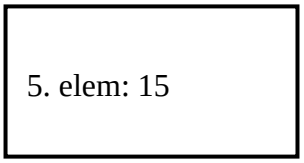
9/F6. Írjon programot, amelyet végrehajtva a robot teljesen véletlenszerűen mozog! Mozgásának minél több paraméterét határozzuk meg véletlen számok sorsolásával! (Sebesség, mozgási idő, irány, motorok be- és kikapcsolása, …)

9/F7. Írjon programot, amelyet végrehajtva a robot sorsol 1-100 közötti véletlen számot. Ha a kisorsolt szám 25 és 75 között van, akkor 1 másodpercig egyenesen haladjon előre. Ha szám nem ilyen, akkor sorsoljon új számot 1 és 2 között. Ha az új szám 1, akkor helyben forduljon balra 1 másodpercig, ha az új szám 2, akkor helyben forduljon jobbra 1 másodpercig. Mindezt ismételje kikapcsolásig. A robot sebessége legyen végig 50. A képernyőjére folyamatosan írja ki az aktuálisan sorsolt számot!

9/F10. Írjon programot, amelyet végrehajtva a robot startpozícióban állva sorsol 1-100 közötti véletlen számot, amelyet kiír a képernyőjére. A szám kiírása után várakozik 5 másodpercig. Ha a szám nagyobb mint 50, akkor előre indul el és egyenesen halad fekete csíkig és ott megáll. Ha a szám nem nagyobb 50-nél, akkor tolatni kezd és fekete csíknál megáll. Miután megállt sípol 1 másodpercig, majd ugyanabban az irányban halad tovább, mint korábban és a következő fekete csíkot elérve megáll. A program vége előtt a robot 5 másodpercig várakozik. A kiírt szám végig maradjon a képernyőjén!

9/F11. Írjon programot amelyet végrehajtva a robot sorsol egy 0-100 közötti számot, amelyet kiír a képernyőjére. Ha sorsolt szám páratlan, akkor a képernyőre rajzol egy kört, amelynek a középpontja (50;32) és sugara 30 pixel. Ha a kisorsolt szám páros, akkor a képernyőre rajzol két koncentrikus kört (*azonos középpontú körök*), amelyek középpontja (50;32) sugaraik pedig 30 illetve 15 pixel. Ütközésérzékelő megnyomására kezdje újra a sorsolást. Mindezt kikapcsolásig ismételje.

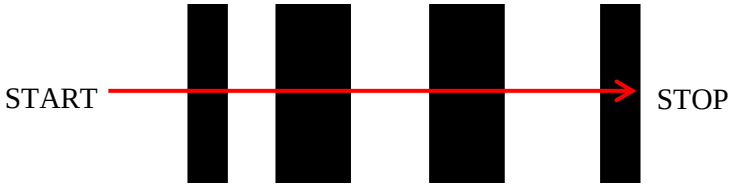
9/F12. Egy számsorozat első tagja 5. A további tagokat úgy kapjuk, hogy az előző taghoz hozzáadunk 1-et, 2-őt, 3-mat és így tovább. A sorozat első hét tagja: 5; 6; 8; 11; 15; 20; 26; … Írjon programot, amelyet végrehajtva a robot sorsol egy 1 és 25 közötti számot, majd a képernyőre írja a sorozat azon elemét, amennyi a sorsolt szám.
Például: A sorsolt szám 5, akkor a képernyő tartalma:



9/F13. Írjon programot, amelyet végrehajtva a robot sorsol egyetlen 1 és 10000 közé eső számot, amelyet a képernyőjére is kiír. A képernyőre írja továbbá a szám tízesre és százasra kerekített értékét az alábbi minta szerint!
Pl.:

9/F14. Az előző feladat átalakított változata. A robot írja képernyőjére a korábban sorsolt szám jelenleg érvényes forint kerekítési szabály szerint kerekített értékét. Tehát 1-re vagy 2-re végződő számok esetén 0-ra kerekítünk lefelé, 3-ra vagy 4-re végződő számok esetén 5-re kerekítünk felfelé, 6-ra vagy 7-re végződő számok esetén 5-re kerekítünk lefelé, míg 8-ra vagy 9-re végződő számok esetén 0-ra kerekítünk felfelé.
Pl.:

9/F15. Írjon programot, amelyet végrehajtva a robot az ábrán jelzett startpozícióból indul egyenesen előre négy egymással párhuzamos, de különböző szélességű fekete csík fölött, a csíkokra merőleges irányban. A pálya alapszíne fehér. A vékonyabb csíkok szélessége legalább 2 cm, a vastagabb csíkok szélessége legalább kétszerese a vékonyabb csíkokénak. A csíkok között legalább 2 cm távolság van. Az első csík mindenképpen vékony, a többi csík tetszőlegesen lehet vékony és vastag is. A negyedik csíkon történő áthaladás után a robot megáll és egy morzekódot játszik le a csíkok szélességének megfelelően. A morzekód rövid illetve hosszú hangjelzések sorozata. A rövid hangjelzés 0,1 mp (másodperc) időtartamú és utána 0,2 mp szünet következik. A hosszú hangjelzés 0,5 mp időtartamú és utána 0,2 mp szünet következik. Az ábrának megfelelő morzekód: *rövid-hosszú-hosszú-rövid* (ti tá tá ti). A megszólaltatott hang azonos legyen a rövid és hosszú esetben is!



10. TÖBBSZÁLÚ PROGRAMOK – TASZKOK

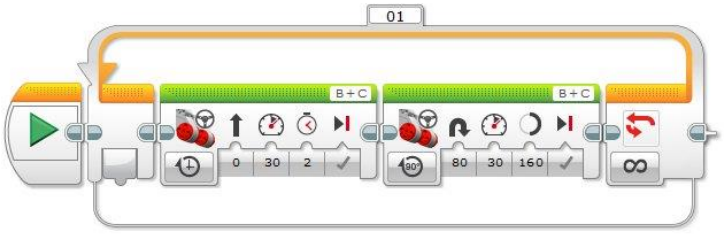
A hagyományos procedurális programozási környezetek esetén megszokott, hogy a programok egy szálon futnak. Ez azt jelenti, hogy egy lineáris utasítás-végrehajtási rend érvényes, az első utasítástól kezdve sorban az utolsóig. A lineáris programszerkezetből következően programjaink egy utasítást csak akkor fognak végrehajtani, ha az előző utasítások már lefutottak.
A mindennapi életben viszont az emberi érzékszervek egyszerre több szálon is képesek figyelni a környezet eseményeire. Szem, fül ...
A bonyolultabb programozási rendszerekben ez a többszálúság (*multitask*) már alapértelmezett funkció, hiszen a számítógép operációs rendszere lehetővé teszi, hogy látszólag egyszerre több programot is futtassunk. Pl. internetezés közben zenét is hallgathatunk a számítógépen egy médialejátszó program segítségével.
Erre a többszálú programozásra nyújt lehetőséget az EV3 szoftverkörnyezet és a robot. Szükség is van rá, hiszen a robot szenzoraira sok esetben egymással párhuzamosan kell figyelni és a motorokkal történő mozgás mellett esetleg egy speciális szenzorérték esetén valamilyen eseményt kezdeményezni.

10.1. Többszálú programok létrehozása

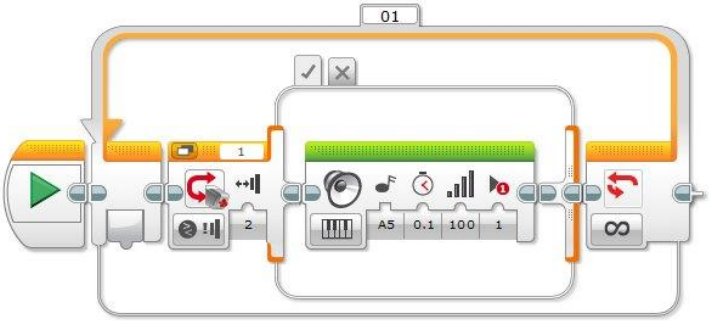
A programunkban tehát létrehozhatunk egymással látszólag párhuzamosan működő, futó taszkokat (szálakat), így egy időben tudjuk a motorokat vezérelni és a szenzorokkal mért adatokat kezelni.
Egyszálú program esetén is lehet a motorok vezérlése mellett a szenzorok értékeit figyelni, de ha a programunk várakozási utasítást tartalmaz, akkor mindaddig, amíg az ott beállított érték nem teljesül az utána következő utasítások nem hajtódnak végre, tehát például a további szenzorok figyelése sem történhet meg.
Egy egyszerű példával szemléltetve a szituációt:

10/P1. Írjon programot, amelyet végrehajtva a robot egy sokszög alakú pályán halad folyamatosan! Abban az esetben, ha az ütközésérzékelőjét nyomás éri, adjon hangjelzést!

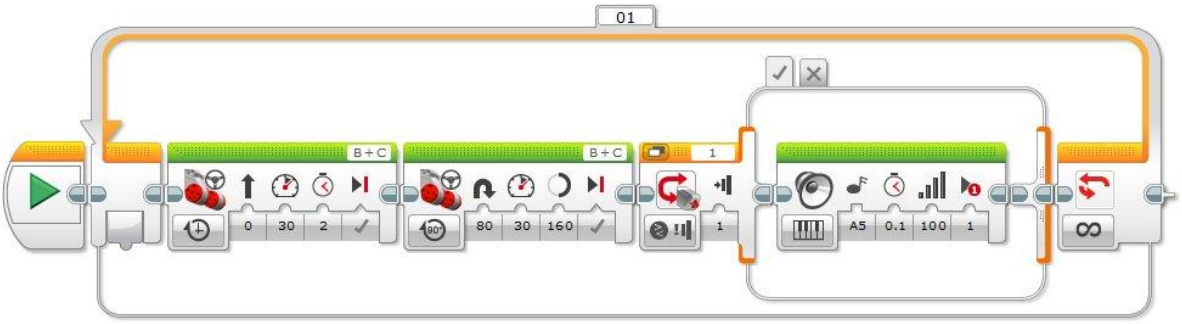
A program egyes elemeinek elkészítése nem okoz különösebb nehézséget. A sokszög alakú pályán történő haladást úgy érhetjük el legegyszerűbben, hogy a két motort adott ideig működtetjük (egyenes haladás), majd rövid ideig ellentétes irányban forgatjuk (kanyar). Mindezt ismételjük egy végtelen ciklusban. A sokszög töréspontjainak a száma a fordulás nagyságától függ.



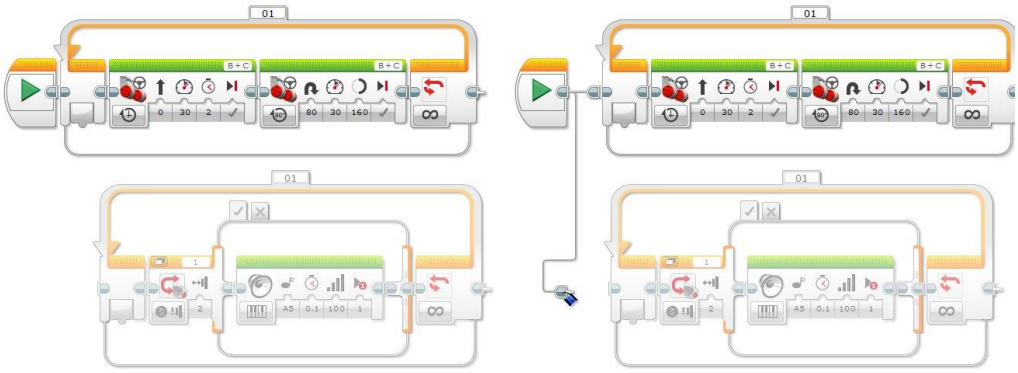
Motorok vezérlő blokkjai közül a kormányvezérelt blokkot használtuk. A két motor először 30-as sebességgel halad 2 másodpercig előre, majd balra fordul 160°-ot szintén 30-as sebességgel.
Az ütközésérzékelőre történő hangjelzés programozása szintén egyszerű feladat. Az elágazás alsó szála nem tartalmaz utasítást, ezért nem jelenítettük meg. A hangjelzés 0,1 másodpercig szól.



Hogyan lehet a két programot összeilleszteni? Az első próbálkozásként a legcélszerűbbnek tűnik, ha ugyanabba a végtelen ciklusba betesszük egymás után a motorok vezérlését és az ütközésérzékelő figyelését is.



Érdemes kipróbálni mindhárom programot. A harmadik esetben (amikor a két előzőt egymás után helyeztük el egy ciklusban) nem működik helyesen a program. Hiába nyomjuk meg az ütközésérzékelőt, a hangjelzés csak bizonyos esetekben szólal meg. A programot tesztelve észrevehető, hogy ha az ütközésérzékelőt folyamatosan nyomva tartjuk, akkor a kanyarodás után szólal meg rövid ideig a hang (0,1 mp-ig), majd folytatódik a mozgás. Az ok a lineáris programvégrehajtásban kereshető. A két motorvezérlő modulhoz érve az utasítás-végrehajtás addig nem lép a következő modulra, amíg le nem telt a motoroknál beállított idő illetve elfordulási érték. Utána vizsgálja meg a program, hogy éppen benyomott állapotban van-e az ütközésérzékelő. Ha éppen nincs, akkor lép tovább és kezdi a ciklust, tehát a mozgást elölről. Ha éppen nyomás alatt van az ütközésérzékelő, akkor kiadja a rövid hangjelzést, és már lép is tovább. Tehát ha éppen nem akkor nyomjuk meg az érzékelőt, amikor a program ezt vizsgálja, akkor programunk észre sem veszi, hogy történt valami esemény, amire neki reagálnia kellett volna. Javítható a hatékonyság egy kicsit, ha az ütközésérzékelő feltételét *Bumped*-re állítjuk. Ekkor mindenképpen megszólal a hangjelzés, de csak miután a két mozgás (egyenes irányú és kanyarodás) befejeződött. A rendszer tehát megjegyezte (rövid időre), hogy történt egy ütközésérzékelő benyomás, ami alapján az elágazás igaz ágán szereplő utasítást végre is hajtja, ha a ciklusban odaért az blokkhoz.
A helyes működéshez vezető megoldás az lehet, hogy a törött vonal mentén történő mozgást végző részt és a szenzorfigyelést megvalósító részt folyamatosan egymással párhuzamosan hajtjuk végre. Ehhez két, látszólag egymással párhuzamosan futó szálat használunk. Mindkét programmodult elhelyezzük a szerkesztő területen. A motorok vezérlését szolgáló részt a szokásos helyre, míg az érzékelőt figyelő ciklust alá mozgatjuk. Az érzékelőt figyelő rész halványabb színnel látszik, mert ez jelenleg nem része az aktív programnak.

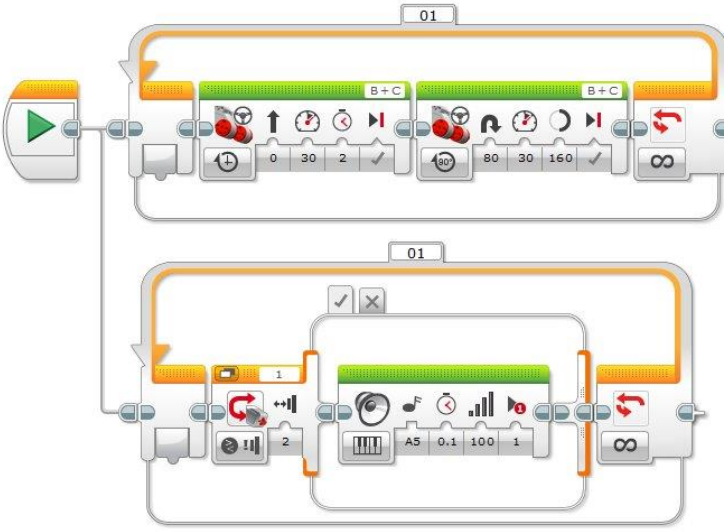


Rákattintva a START ikon és a motorokat tartalmazó ciklus csatlakozási pontjára, eltávolodik egymástól a két ikon és hozzáférhetővé válik egy szürke „kábeles csatlakozás". A STRAT ikonból az egérrel megfogva a csatlakozási pontot újabb szál húzható ki és ezt csatlakoztathatjuk a másik ciklus bementi pontjához. Ezzel az is aktívvá válik és a programunknak már két szála van. A két szálon lévő utasítások egymással látszólag párhuzamosan futnak, tehát a robot egyszerre fog mozgást végezni és figyelni az ütközésérzékelő állapotára.

A látszólagosságot azért hangsúlyozzuk, mert a tégla egyetlen processzort tartalmaz, amely sorban egymás után hajtja végre az utasításokat, de a két szálon szereplő modulok esetén gyorsan váltakozva. Hol az egyik, hol a másik szál utasítását hajtva végre. Ha ez a váltogatás elég gyors, akkor a felhasználó a robot működésében a párhuzamosságot látja.

A szálakat a program bármelyik helyén csatlakoztathatjuk egymáshoz (cikluson belül és elágazásban nem). A programszálak összekötése nem tartalmazhat zárt hurkot.

A program így már helyesen működik.

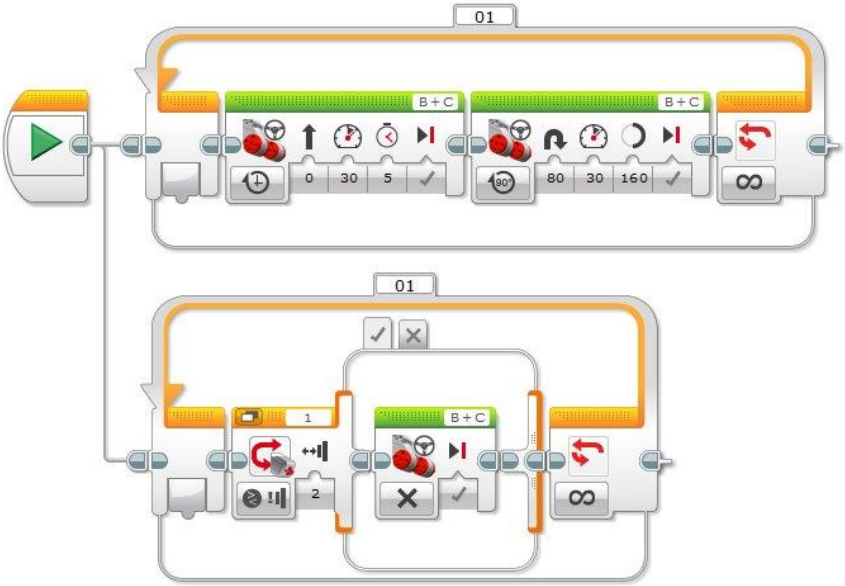


A többszálú programok használatánál problémát okozhat, ha egyszerre több szál is ugyanazt a motort vezérli. Ilyenkor alaposan végig kell gondolni, hogy melyik szál érvényesül éppen. Esetleg eredményként nem is történik mozgás.

Ennek szemléltetésére az előző feladat egy módosított változatát érdemes kipróbálni.

10/P2. Írjon programot, amelyet végrehajtva a robot egy sokszög alakú pályán halad folyamatosan! Abban az esetben, ha az ütközésérzékelőjét nyomás éri, álljon meg!

Az előző program mintájára próbálkozzunk a következővel, de módosítsuk úgy az első motorvezérlő blokk működését, hogy 5 másodpercig mozogjon egyenesen előre a fordulás megkezdése előtt.



Első lépésben teszteljük úgy a programot, hogy az ütközésérzékelő *Bumped* állapotban van. A program érdekes viselkedést mutat. Elindul a robot, majd ha megnyomjuk az ütközésérzékelőt, akkor nem történik semmi csak ha felengedjük (az ütközésérzékelő beállítása miatt). Ekkor az éppen aktuális blokk végrehajtása megszakad és a következő blokk végrehajtása kezdődik. Tehát ha egyenes mozgás közben nyomtuk meg és engedtük fel a szenzort, akkor fordulni kezd, míg ha éppen fordult, akkor egyenesen halad tovább előre a felengedés után. Mivel mindkét szálon egy-egy végtelen ciklus fut, ezért az egyik szálon kezdeményezett mozgásmegszakítás csak időszakosan állítja le a másik szálon futó utasításokat (aktuális blokk).
Ha *Pressed* állapotban használjuk az ütközésérzékelőt, akkor a mozgás rögtön leáll és csak akkor indul újra, amikor felengedjük az érzékelőt. Az idő viszont közben is telik a motor számára, így ha akkor nyomtuk le, amikor még 4 másodperc volt hátra az egyenes mozgásból, két másodpercig tartottuk lenyomva, akkor felengedés után még további 2 mp-ig egyenes halad a fordulás előtt.

A fenti programokat párhuzamos szálak nélkül, lineáris szerkezettel is meg lehet írni, hogy helyesen működjenek, de sokkal bonyolultabban. Arra kell ügyelni, hogy a használt utasítások, modulok ne tartalmazzanak olyan várakoztatást, amely közben nem történik meg a szenzorok figyelése. Összetett feltételekkel és az időzítők (*Timer*) használatával mindez kikerülhető, de nem érdemes bonyolult programszerkezeteket létrehozni, ha egyszerűbben is megoldható a feladat.
Tapasztalatként megfogalmazhatjuk, hogy ha párhuzamos szálakat használunk, akkor alaposan meg kell tervezni, hogy a motorok mozgását éppen melyik szál vezérli. Általában érdemes ezt egy programszálra bízni, és a többi szálat a szenzorok figyelésére, vagy egyéb műveletek elvégzésére használni.

10.2. Gyakorló feladatok

10/F1. Írjon programot, amelyet végrehajtva a robot egy sokszög alakú pályán mozog! Ha az ultrahang szenzorával 15 cm-es távolságon belül akadályt érzékel, akkor megáll. (Írja meg a programot úgy is, hogy ha a megállás után az akadályt eltávolítjuk, akkor ne induljon el újra a robot!)

10/F2. Írjon programot, amelyet végrehajtva a robot elindul egyenesen előre a haladási irányára merőleges fekete vonalak fölött (az alap színe pl. fehér)! A harmadik vonal fölötti áthaladás után elkezd tolatni az ütközésérzékelő benyomásáig. Miközben mozog, két hangból álló dallamot játszik folyamatosan (1. hang: 440 Hz – zenei A, 200 ms időtartamig, 2. hang: 528 Hz – zenei C, 200 ms időtartamig).

10/F3. Írjon programot, amelyet végrehajtva a robot elindul egyenesen előre! Az ultrahang szenzora által jelzett akadálytól 15 cm-re megáll, majd tolat egyenesen a fekete csíkig és újra indul előre, az akadálytól 15 cm-ig. Mindezt ismétli háromszor. A képernyőre folyamatosan kiírja az indulástól eltelt időt.

10/F4. Írjon programot, amelyet végrehajtva a robot egyetlen fényszenzorával egy nem egyenes útvonalat követ! Mozgás közben a fényszenzora által mért értéket folyamatosan a képernyőre írja. (Írja meg a programot taszkok segítségével, és anélkül is!)

10/F5. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre egy fehér színű felületen, és megáll, ha fényszenzora fekete színű csíkot észlel vagy ultrahangszenzora által mért érték alapján 20 cm-nél közelebb kerül egy akadályhoz! A szenzorok figyelésére használja a *Wait* modult! (A feladat a *Wait* modul használatában különbözik a 9/P4-től.)

10/F6. Írjon programot, amelyet végrehajtva a robot váltakozva két-két másodpercig balra, majd jobbra forog kikapcsolásig ismételve! A különböző irányú forgás közben más-más hangból álló hangjelzést adjon!

11. SAJÁT UTASÍTÁSBLOKK

11.1. Saját blokkok létrehozása

Az összetettebb programok írása során előfordulhat, hogy ugyanazon műveletsort a program több különböző helyén is meg kell ismételni. Más programnyelvekben ilyen esetekben célszerű az ismétlődő kódot külön függvényként, eljárásként, szubrutinként megírni, amit egyszerű hivatkozással lehet beilleszteni a megfelelő helyre. Az EV3-as szoftverben is van erre lehetőség. Készíthetünk saját modulokat, amelyeket a program tetszőleges helyére akárhányszor beilleszthetünk egyetlen blokként. A végrehajtás során a saját modulhoz érve, a rendszer végrehajtja azokat az utasításokat, amelyeket a létrehozás során a blokkba beépítettünk. A blokkot lehet paraméterezni, tehát kaphat külső értékeket, illetve a műveletek eredményeit vissza tudja adni kimeneti csatlakozási pontokon keresztül.
A saját modul készítése azzal kezdődik, hogy az utasításait elkészítjük, mint egy különálló programot. Ezután a szükséges utasításokat kijelöljük (az egér bal fülét lenyomva tartva és bekeretezve az utasításokat, vagy egyenként kattintva rájuk a SHIFT gomb lenyomva tartása mellett). Ekkor az *Tools* menü *My Block Builder* menüpontját választva tovább szerkeszthető a modul.
A használathoz az egyszenzoros útvonalkövetést választottuk, mivel ez sok programnál használható, így a saját blokk elkészítése megkönnyíti a programok írását.

11/P1. Írjon programot, amelyet végrehajtva a robot egyetlen fényszenzorával követi az alaptól különböző színű vonalat.

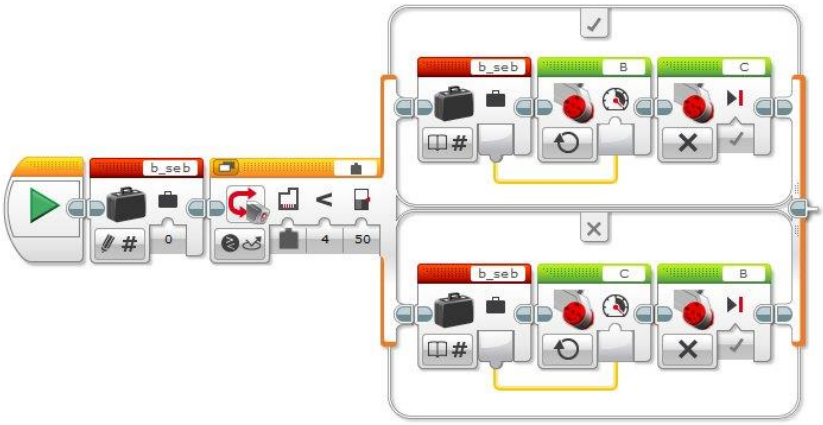
Mielőtt hozzá kezdenénk a saját blokk elkészítéséhez érdemes megterveznünk, hogy milyen adatok szükségesek a program működéséhez, tehát milyen értékeket kell majd a blokknak átadnunk, illetve milyen visszatérési értékeket (kiviteli paraméterek) szolgáltat a működés során a blokk.
Az általános használathoz négy bementi paramétert választunk (ezek köre tovább bővíthető):

- A motorok sebessége.
- A fényszenzor határértéke, ami alapján megkülönbözteti az alap és a vonal színét.
- A fényszenzor csatlakoztatási portjának száma.
- Az útvonalkövetés „iránya”, ami alatt azt értjük, hogy a vonal melyik szélét követi a robot. Szükség lehet arra, hogy a vonalat bal vagy jobb oldalán kövesse. Ezt a feltételes elágazás relációs jelének iránya határozza meg (< vagy >).

A felsorolt négy paraméteren kívül továbbiak is adhatók. Pl.: a motorok csatlakozási portja, a motorok forgási iránya, a motorok sebességét külön-külön is vezérelhetjük, ...

Kimeneti paraméterként a blokknak nem kell semmit visszaadni.

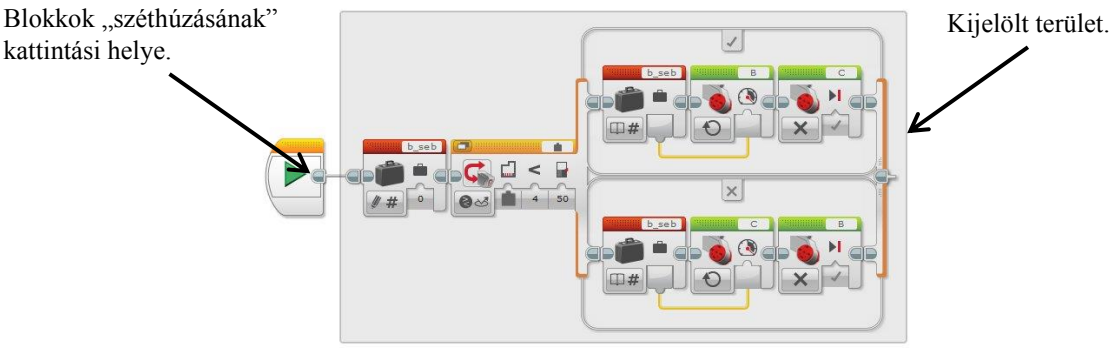
Miután tisztáztuk a bemeneti és kimeneti paramétereket, első lépésben el kell készíteni az útvonalkövető algoritmust.

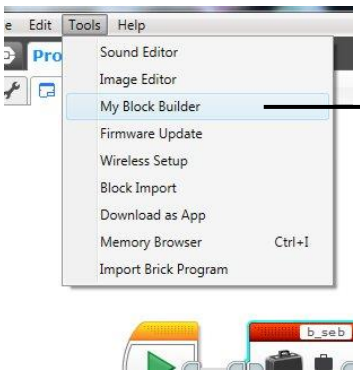


Emlékeztetőül: az algoritmus úgy működik, hogy egy fényszenzorral vezérelt elágazás két szálára két-két motort vezérlő blokkot helyezünk. Minden blokk csak egyetlen motort fog vezérelni. Az egyik motor forog, míg a másik áll. A két szálon éppen ellentétesen. Tehát ha a fényszenzor a határértéknél nagyobb intenzitást mér, akkor az egyik szálon lévő utasítások futnak pl. a bal motor fordul a jobb pedig áll, míg ellenkező esetben a másik szálon lévők: a jobb motor fordul, a bal pedig áll. Mindeközben a robot kígyózó mozgással halad előre, és emiatt a fényszenzora hol a vonal fölött, hol pedig mellette lesz, így váltakozva hol a bal, hol a jobb motor fordul picit előre. Ez eredményezi a kígyózó mozgást, amellyel a vonal szélét követi.

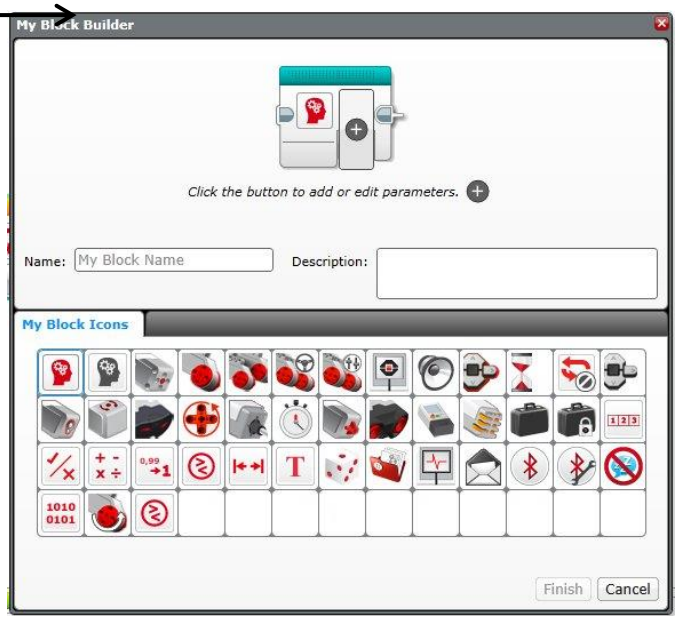
Az összeállított program nem tartalmazza a ciklust, csupán egy mozdulatnak az algoritmusát. Ciklusba majd akkor kerül, ha ténylegesen használni szeretnénk a blokkot, hiszen előre nem tudjuk, hogy a ciklusnak meddig kell futnia, így nem célszerű beépíteni a saját blokkba, mert túl bonyolult paraméterezést igényelne az általános megoldás.
A beállításoknál több dologra is figyelni kell. A motorok sebességét külső értékkel szeretnénk megadni. Mivel a motorok az elágazáson belül vannak, ide csak úgy tudunk értéket bevinni, ha létrehozunk egy változót, amely majd megkapja a sebesség paramétert, és az elágazáson belül ebből a változóból olvassa ki a motor a sebesség értékét. A programban ez a változó a *b_seb*. A fényszenzor csatlakozási portjának a számát is külső paraméterrel szeretnénk vezérelni, így a fényszenzor blokk jobb felső sarkában a port beállításnál a *Wired* opciót kell választanunk, ezzel megjelenik bementi csatlakozási pontként a port beállítási paramétere.

Az alap algoritmus elkészülte után a blokkba szánt utasításokat ki kell jelölni. Ehhez az egér bal gombját lenyomva tartva egyszerű bekeretezéssel ki tudjuk jelölni a szükséges részt. Ezt megkönnyítheti, ha a blokkok közötti csatlakozási pontra kattintunk az egérrel, mert ekkor megnő a blokkok közötti távolság (egy kábeles összekötés jelenik meg).

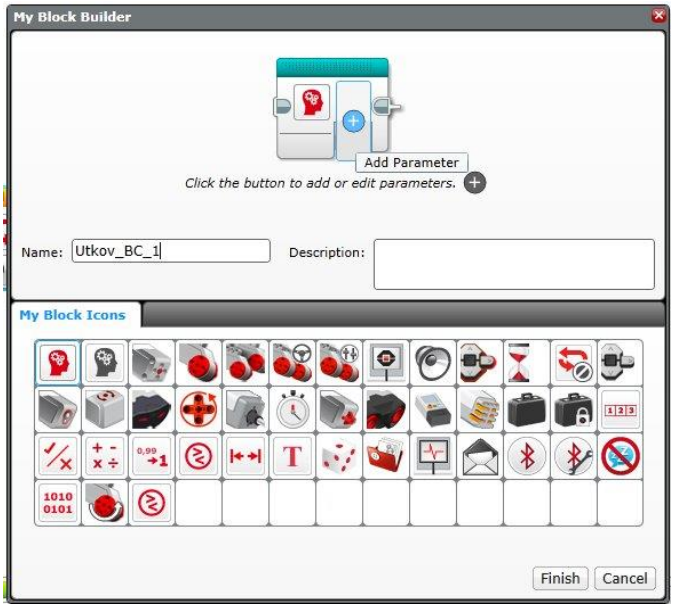




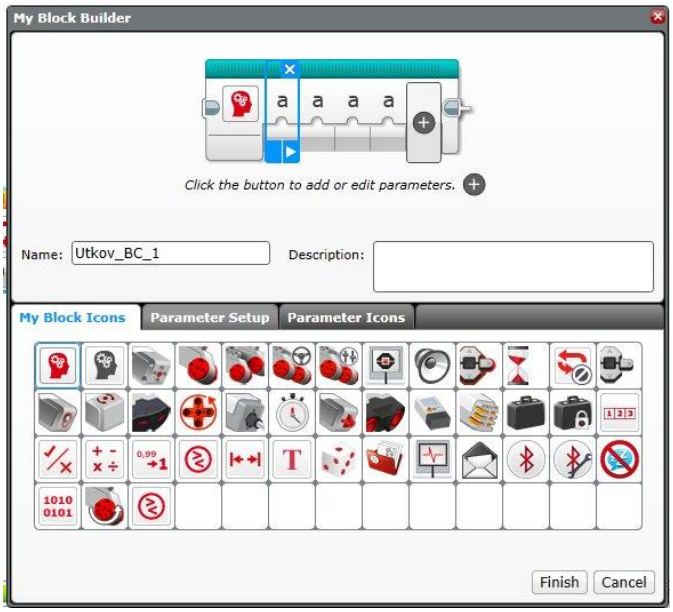
A kijelölés után a *Tools* menü *My Block Builder* menüpontjával érjük el a további szerkesztési lehetőségek párbeszédpanelét.



Első lépésben választhatunk ikont a blokkunknak (a felkínált listából), illetve itt kötelezően meg kell adnunk a blokk nevét (*Name*), és írhatunk rövid leírást is (*Description*).
A blokkunknak az *Utkov_BC_1* nevet adtuk. Az elnevezés utal az útkövetésre, a használt motor portokra és arra, hogy egyetlen fényszenzorral működik. A felkínált ikonon (alapértelmezés) nem változtattunk.

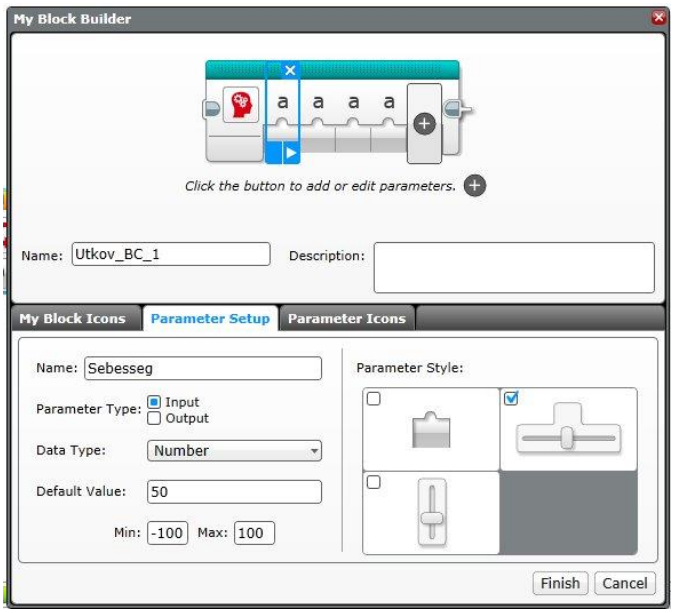


Ezután a szükséges paramétereket tudjuk hozzáadni a blokkhoz. Ezekből négyre van szükségünk. A panel felső részén látható sablonon megjelenő kék színű ⊕ jelre kattintva lehet paramétereket hozzáadni.

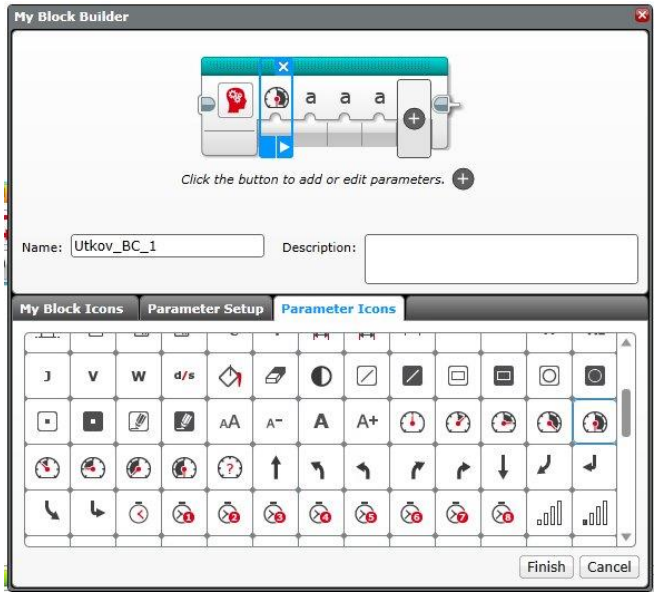


Miután megvan a négy paraméter két újabb lap jelenik meg a panelen: *Parameter Setup* és *Parameter Icons* néven. Itt lehet beállítani külön-külön valamennyi paraméter tulajdonságait.

Első lépésként a paraméterek tulajdonságait állítsuk be, majd ikont is rendelhetünk hozzájuk. Ez az ikon fog megjelenni a blokkunkon a csatlakozási pont fölött.

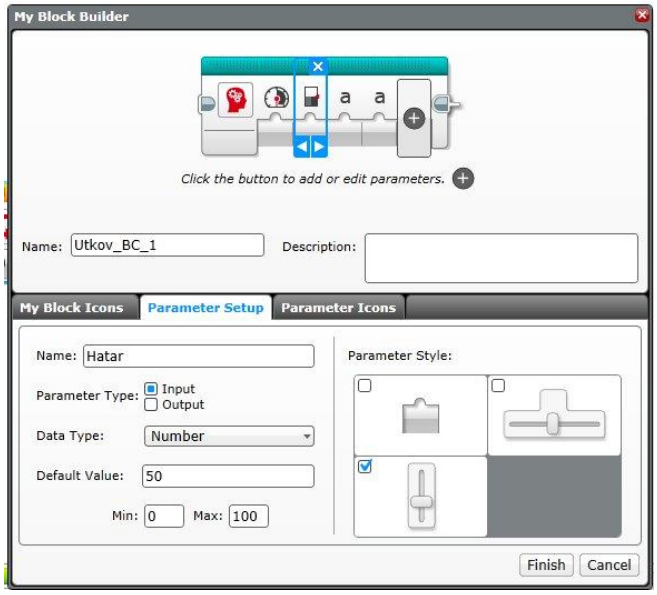


Legyen az első paraméter a sebesség.
- Meg kell adni a nevét: *Sebesseg* (*Name*)
- A paraméter típusát: *Input* (bemeneti)
- A bele kerülő adat típusát: *Number*
- Alapértelmezett értékét: *50* (ha nem adjuk meg o). Érdemes beállítani, mert egy rossz adatátadás során el sem fog indulni a robot, ha o marad az érték.
- A paraméter megjelenési módját az elkészülő blokkon. Ez lehet egyszerű beviteli mező, vagy vízszintes ill. függőleges csúszka. Ez utóbbiak közül választva valamelyiket a minimális és maximális értéket is beállíthatjuk. Ha egyszerű beviteli mezőt választunk, akkor a felhasználó akármekkora értéket beírhat, ami gondot jelenthet néhány esetben, így a szélsőértékek megadásával ezt a problémát elkerülhetjük.



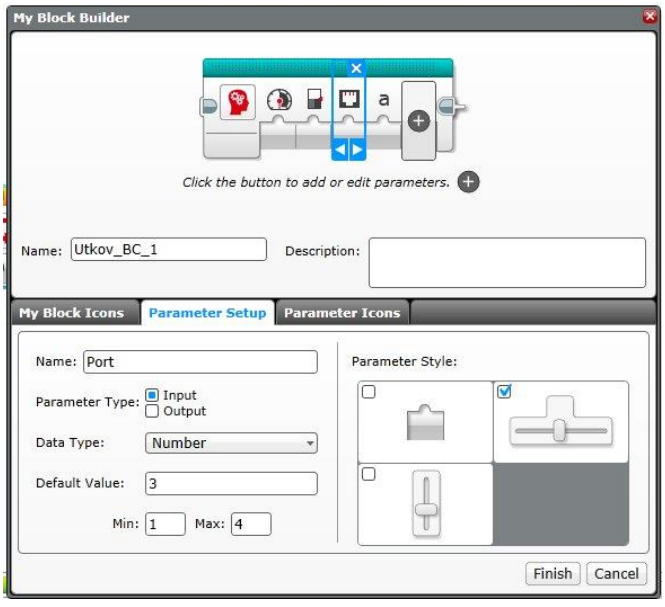
Végezetül egy listából választhatunk ikont is a paraméterhez.
A programkörnyezetben szokásos kilométeróra szimbólumot választottuk.

A többi paraméter beállítása hasonlóan történik.

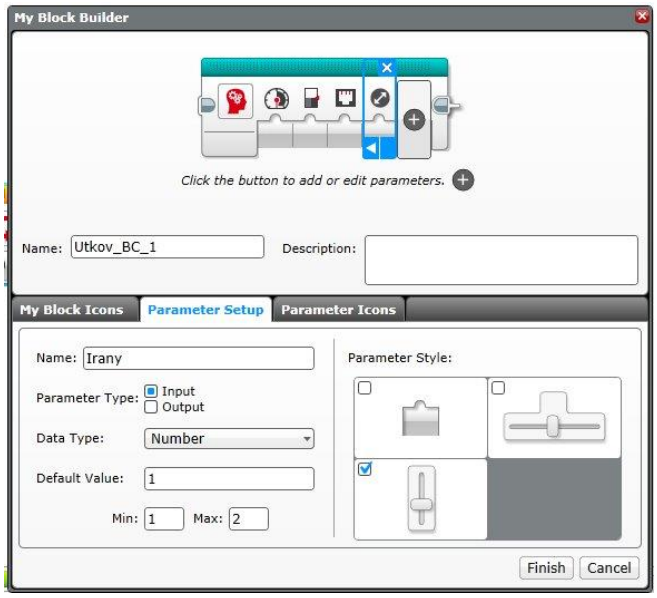


Második paraméter legyen a fényszenzor határértékét meghatározó bementi pont.
A megadott értékek:
- Név: *Hatar*
- Típus: *Input*
- Adattípus: *Number*
- Alapértelmezett érték: *50*
- Stílus: *függőleges csúszka* (*0-100* közötti értékkel).

Harmadik paraméter a fényszenzor csatlakozási portja.

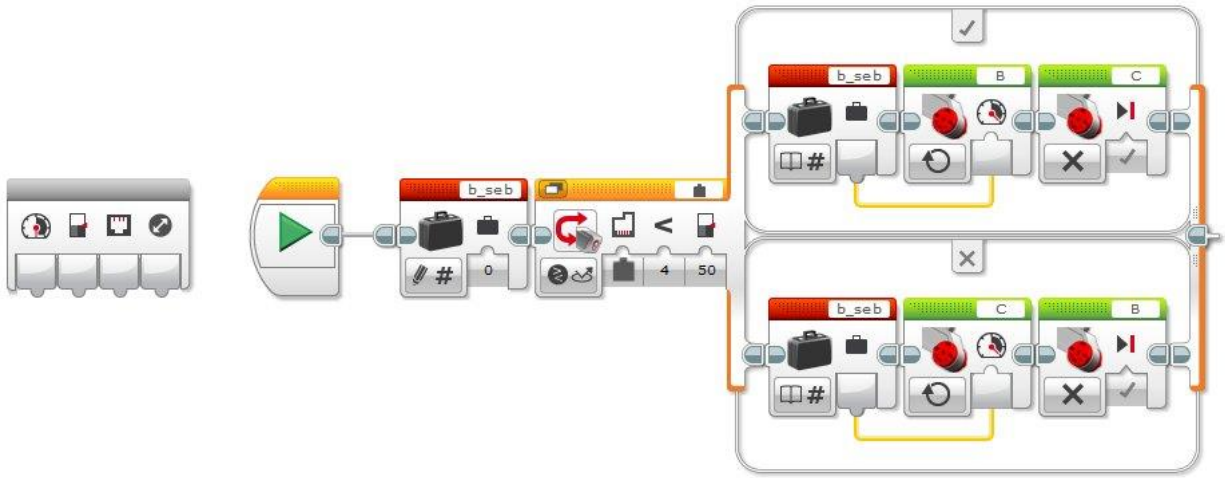


- Név: *Port*
- Típus: *Input*
- Adattípus: *Number*
- Alapértelmezett érték: *3* (érdemes beállítani, mert egyébként 0 lenne, ami nincs értelmezve, egyébként a fényszenzor szokásos csatlakozási portja a 3)
- Stílus: *vízszintes csúszka* (*1-4* közötti értékkel)

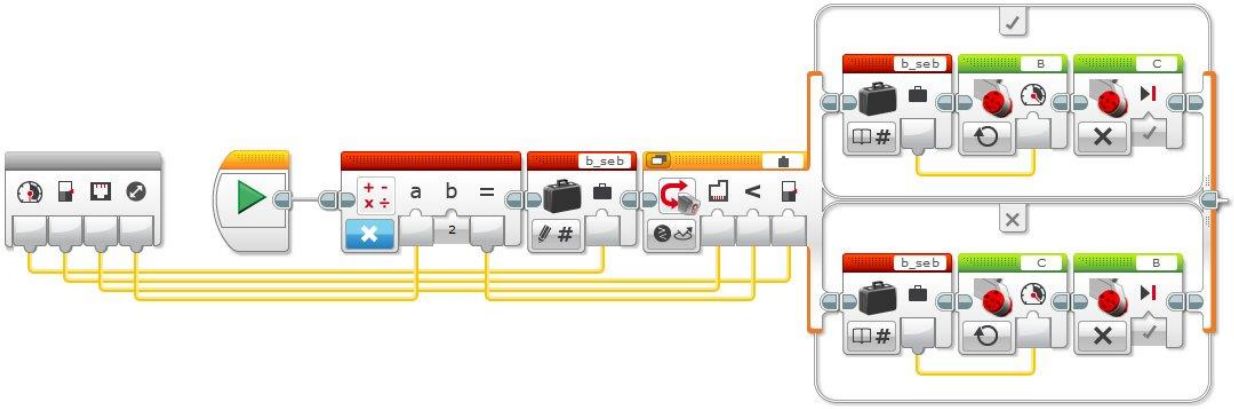


Negyedik paraméter az irány, ami az útvonalkövetésnél azt határozza meg, hogy a vonal bal vagy jobb szélét követi-e a robot.
- Név: *Irany*
- Típus: *Input*
- Adattípus: *Number*
- Alapértelmezett érték: *1*
- Stílus: függőleges csúszka (*1-2* közötti értékkel)

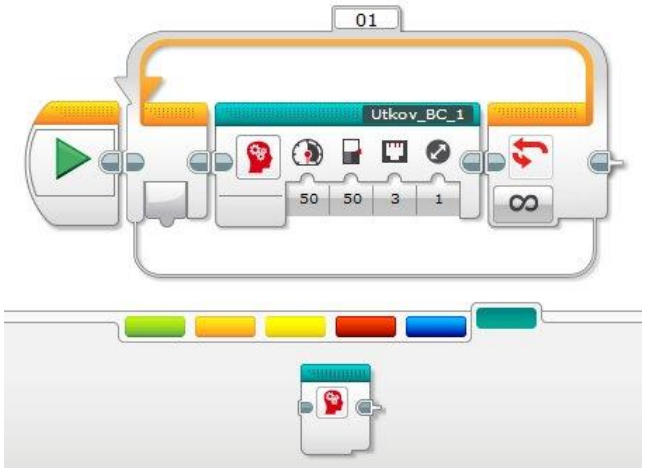
Az útvonal követési irányát a feltétel relációs jelének állása határozza meg: „<" vagy „>". Minden relációs jelhez egy számot rendelt a szoftver. A „>" jelhez a 2-est, míg a „<" jelhez a 4-est. Mivel nem akarjuk, hogy a felhasználó a 3-as számot is megadhassa értékként, ezért állítottuk be minimális értéknek az 1-est és maximálisnak a 2-est. Ebből úgy fogunk 2-est és 4-est csinálni, hogy mielőtt átadnánk a fényszenzor blokknak a bejövő számot, megszorozzuk 2-vel. Így az 1x2 = 2 és 2x2 = 4 értékekhez jutunk.
Ha készen vagyunk a paraméterezéssel, akkor a *Finish* gomba kattintva elkészül a blokkunk szerkezete.



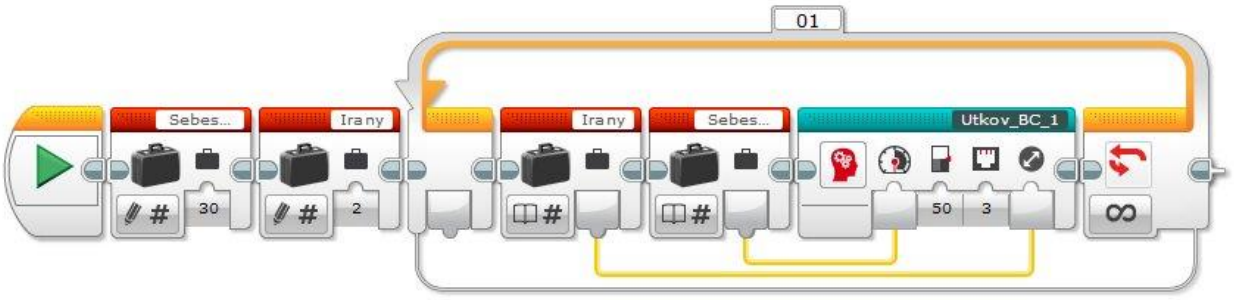
Az eredeti program mellett bal oldalon megjelent a négy paramétert szimbolizáló ikonsor, amelyet most a megfelelő helyre kell kábelezni, illetve még el kell helyezni a programban a 2-vel szorzó blokkot (magyarázatot lásd fentebb).



Ezzel a blokk kész van a használatra. Az ikonja megjelenik a *My Blocks* csoportban és programba illesztve már tesztelhető is az alapértelmezett beállításokkal: 50-es sebesség, 50-es fényszenzor határ, 3-as portra csatlakoztatott fényszenzor, és „>" relációjel a feltételvizsgálatnál. Ahhoz, hogy az útvonalkövetés működjön ciklusba kell tenni a blokkot. A teszteléshez egy végtelen ciklus is megteszi. Konkrét feladat esetén a kilépési feltételt itt majd be tudjuk állítani.

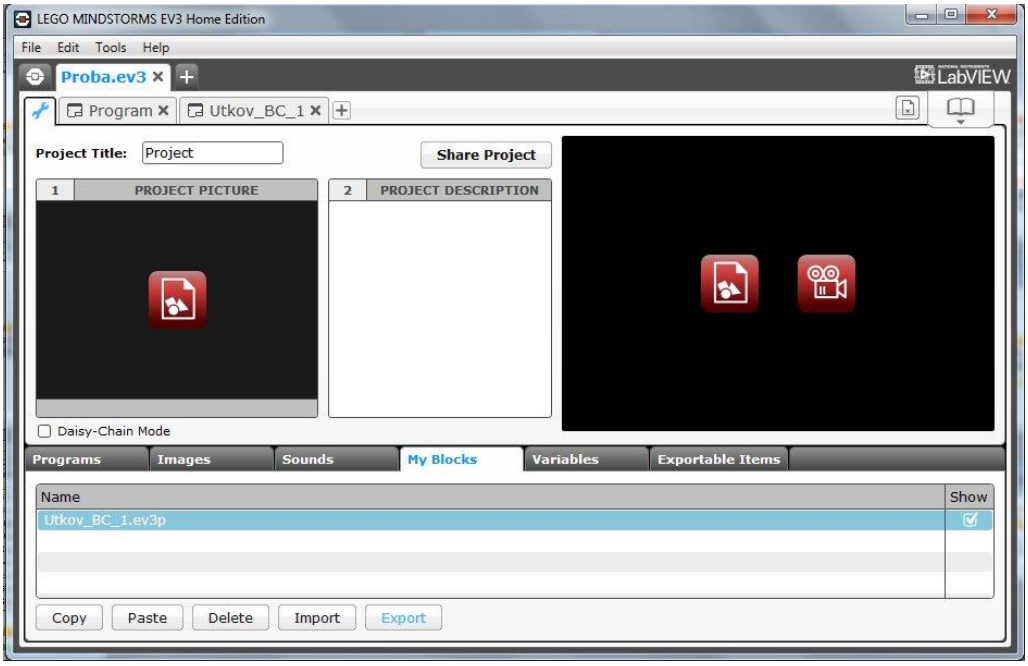


Ha szeretnénk a paramétereket változtatni, azt változókon keresztül, vagy közvetlenül a blokk szövegdobozaiba írva tudjuk megtenni. Az ábrán látható példán a sebességet állítottuk 30-ra és megváltoztattuk az útvonalkövetés irányát.



Ha egy változóba olyan értéket írunk, ami a blokkban beállított határokon kívül esik, akkor nem kapunk hibaüzenetet, hanem a rendszer a hozzá legközelebb álló, még megengedett paramétert fogja használni a futtatás során. Tehát ha irányként pl. 10-et írunk be a változóba, akkor az értékátadás során a rendszer 2-t fog értelmezni, mert az 1 illetve 2 megengedett érték közül a 2 esik közelebb a 10-hez. Ezért fontos, hogy a blokkok létrehozása során állítsuk be a minimális és maximális értékhatárokat.

Az elkészült blokk megjelenik a projekt tulajdonságlapján is. Innen exportálhatjuk tetszőleges mappába, ahonnan később bármelyik projektbe vissza tudjuk importálni, így általánosan használhatóvá tehetjük. Az exportálás során *ev3s* kiterjesztést kap a fájl.



A bemutatott saját blokk készítése majdnem minden lehetőségre kitért. Egyetlen elem maradt ki: a blokk által kiszámított értékek visszaadását hogyan lehet megoldani. Ezt egy másik több helyen használható algoritmus segítségével mutatjuk be.

11/P2. Írjon programot, amely két pozitív egész számot kap bemeneti paraméterként, az első számot elosztja a másodikkal és eredményül visszaadja a hányadost és az osztási maradékot.

Mivel a programnyelvben nem áll rendelkezésre maradékos osztást végző modul, ezért készítsünk sajátot, ami általánosan bármilyen nemnegatív számra működik.
A matematikai műveletekkel foglalkozó fejezetben bemutattuk, hogyan lehet a 2-vel osztás esetén a maradékot és a hányadost képezni. Utaltunk arra, hogy az algoritmus tetszőleges pozitív osztó esetén is működik. Az algoritmust nem csak osztások elvégzésére használhatjuk, hanem a megoldás végén egy olyan komplex feladatot is bemutatunk, amelyhez az elkészített saját blokk használható.
Ismét kezdjük azzal a munkát, hogy megtervezzük milyen bemenetei és kimenetei lesznek a blokknak. Ezt a feladatspecifikáció elég egyértelműen megadja. Két szám típusú bemenet (szám és osztó), és két szám típusú kimenet szükséges (hányados és maradék).
Az algoritmus lényege az egész osztás matematikai formulájában keresendő (valamennyi szereplő érték égész és az osztó nem lehet 0):

$$\text{szám} = \text{hányados} * \text{osztó} + \text{maradék}$$

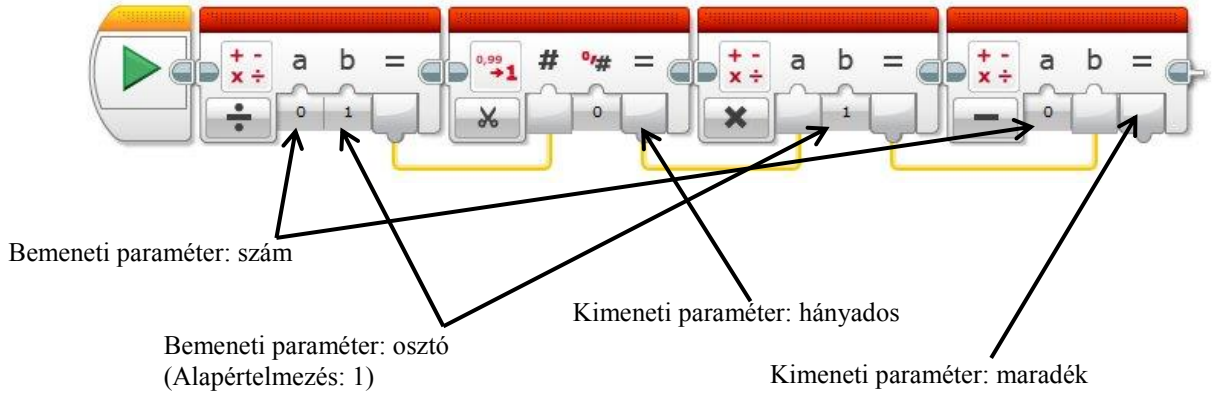
Az egyenletet átrendezve kapjuk a kimeneteket:

$$\text{maradék} = \text{szám} - \text{hányados} * \text{osztó}$$

Mivel az EV3-as szoftver valós osztást tud elvégezni, tehát egy osztás eredménye valós szám lesz (pl.: 9:2 = 4,5), ezért a hányados o tizedes jegyre csonkolt változata lesz az egész hányados.

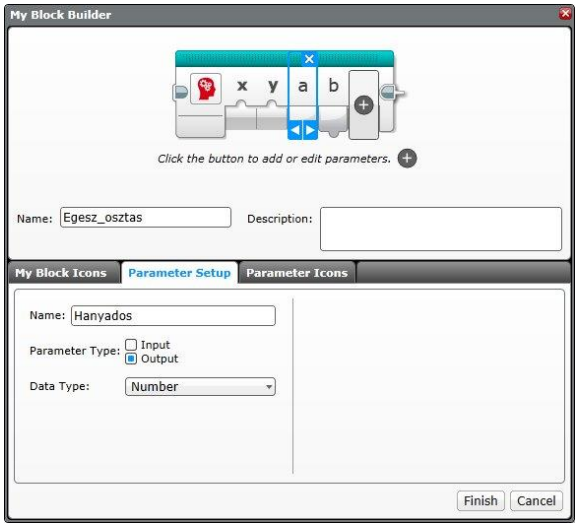
$$\text{hányados} = [\text{szám} : \text{osztó}]$$

Ez alapján az algoritmus magja a következő:

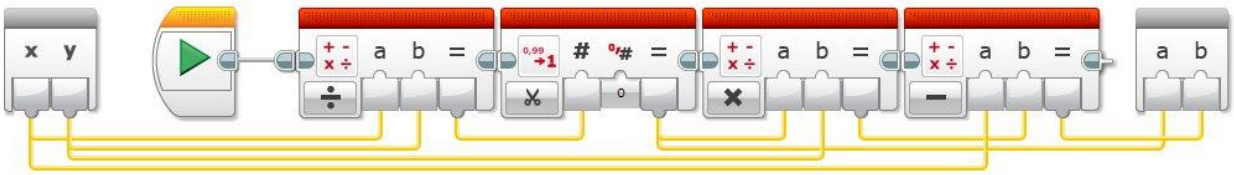


Az első blokk végzi az osztást. A második blokk csonkolja a hányadost o tizedes jegyre, ez lesz az osztás egész hányadosa. A harmadik blokk végzi az egész hányados és az osztó szorzását. A negyedik blokk végzi a szorzat kivonását az eredeti számból, kimenete lesz a maradék.
A négy blokkból a korábban bemutatott módon elkészítjük a saját blokkot, ügyelve, hogy az osztónál az alapértelmezett értéket állítsuk 1-re. Mindkét bemeneti paraméter szövegdoboz stílusú, mivel nem szeretnénk korlátozást bevezetni az értékekre. Így ugyan az osztó esetén manuálisan a o is bekerülhet értékként (a felhasználó felülírja az alapértelmezett számot o-val), de ezt egy feltételvizsgálattal, amit beépíthetünk a blokkba ki lehet védeni. Esetleg bízhatunk a felhasználó matematikai kompetenciáiban is. ☺

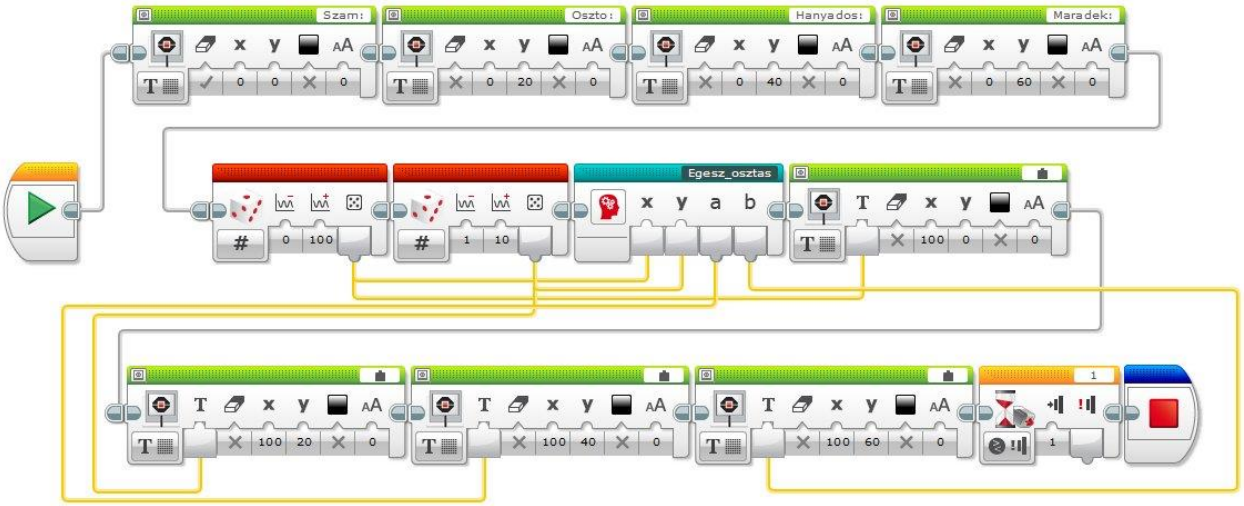
A kiviteli paramétereknél a hányados esetére vonatkozik a következő ábra. Ha a paraméterlistán az *Output* típust választjuk, akkor nincs is több beállítási lehetőségünk, mint az adattípus, ami ebben az esetben szám (*Number*).



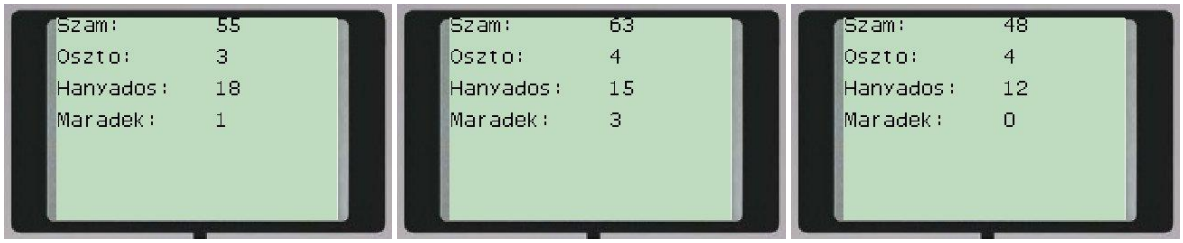
A blokkot *Egesz_osztas*-nak neveztük el. A *Finish* gombra kattintva elkészül a blokk, csak a paramétereket kell a megfelelő módon csatlakoztatni.



Teszteléshez írjunk egy programot, amely sorsol két számot és ezt használja bemeneti adatként. A számot a 0-100 intervallumból, míg az osztót az 1-10 intervallumból.
Írassuk ki a képernyőre, egymás alatti sorokban a számot, osztót, hányadost és maradékot.
A program első ikonsora a szöveges kiíratás a képernyőre (*Szam:* , *Oszto:* , *Hanyados:* , *Maradek:*) egymás alatti sorokba, amelyek között 20 pixel a távolság. A középső rész tartalmazza a tényleges algoritmust, majd a kezdeti értékek és eredmények kiíratása történik meg a vízszintes 100 pozíciótól kezdve. A program ütközésérzékelő benyomására áll le.

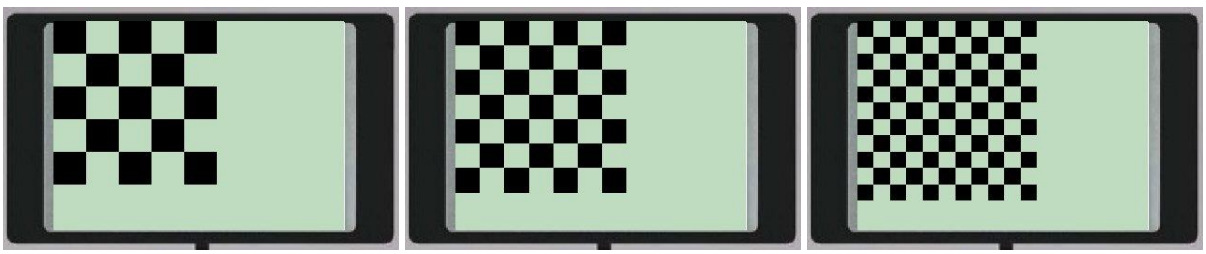


Teszteredmények:



A maradékos osztást megvalósító saját blokk felhasználásával készítünk egy programot, amely szemlélteti a matematikán kívüli alkalmazás lehetőségét.

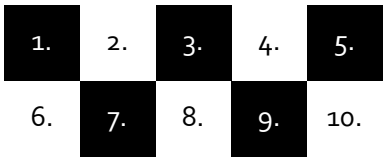
11/P3. Írjon programot, amely egy sakktábla szerű ábrát rajzol a képernyőre! A sakktábla mezői váltakozva fekete és fehér színűek legyenek. A programot páratlan mezőszámú sakktáblára készítsük el! Továbbfejlesztési lehetőségként páros mezőszámú sakktáblára is elkészíthetjük. Minden esetben négyzet alakú legyen a sakktábla. Például az 5x5-ös; 7x7-es és 11x11-es tábla képe.



A programot mérete miatt részletekben mutatjuk be. Az egyes változókat többször is beillesztettük a programba, annak ellenére, hogy egy változóból az érték többször is kiolvasható. Ennek oka, hogy a részekre darabolás során áttekinthető maradjon a forráskód.

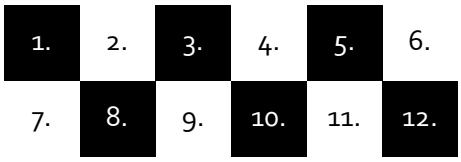
Két bementi változót hoztunk létre, amelyben a sakktábla egy mezőjének oldalhosszát adhatjuk meg pixelben, illetve az egy oldalon található mezők számát. A képernyő megjelenítési mérete határt szab ezeknek a változóknak, de a program nem ad hibát, ha túlcímezzük a képernyőt.
A két változó: *Oldalhossz* és *Mezoszam*. A fenti három példánál ezek rendre (*Oldalhossz*; *Mezoszam*):
(20; 5) (15; 7) (10; 11).
A *Display* ikon a kezdeti képernyőtörlés miatt szerepel, *Text* típusként üres karaktersorozattal.

Az előkészítés után egy ciklus indul. A sakktábla mezői felváltva fekete és fehér színűek a bal felső saroktól indulva jobbra, majd a sor végére érve ugyanez a rend folytatódik a következő sor elején. Tehát minden páros sorszámú mező fekete, míg minden páratlan sorszámú fehér. Ezért fogalmaztuk meg a feladatot páratlan mezőszámú sakktáblára. Páros mezőszámú esetén a következő sor első mezőjének színe megegyezik az előző sor utolsó mezőjének színével, így a bemutatott algoritmust át kell alakítani, hogy páros mezőszámú sakktábla esetén is működjön. Ez megoldható egy további feltétel beiktatásával.



...

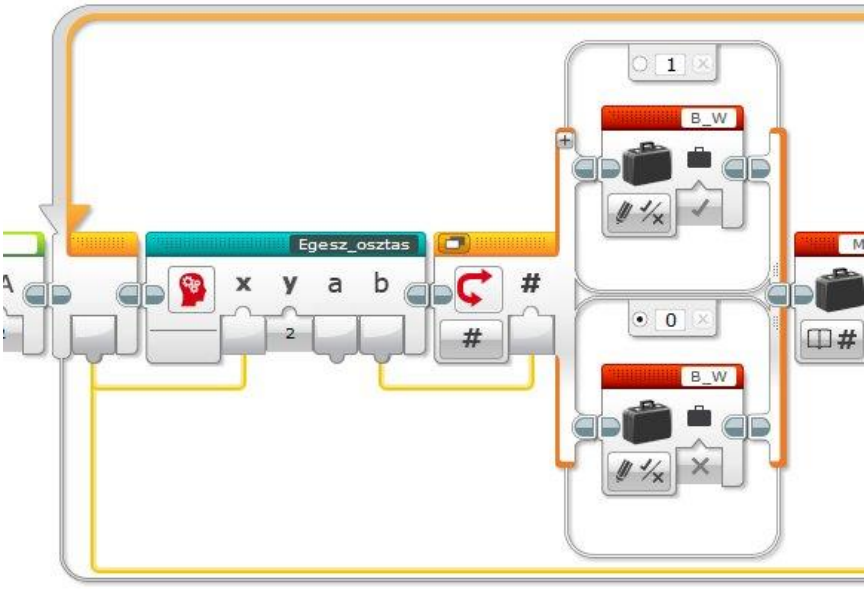
Páratlan oldalszám esetén minden azonos paritású mező színe megegyezik.
(pl.: 5x5)



...

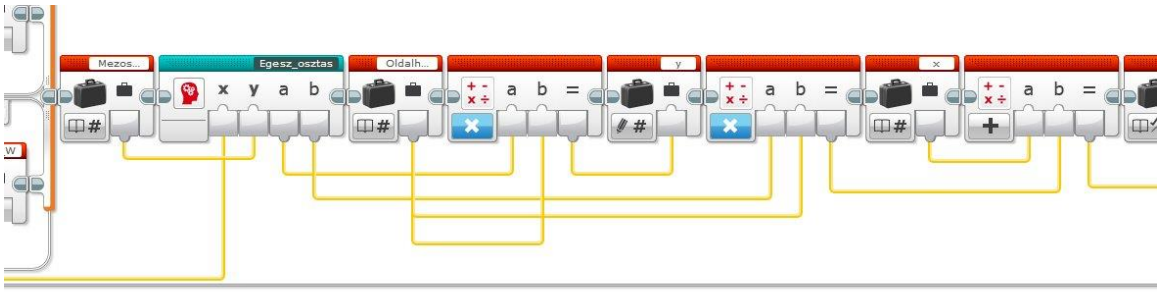
Páros oldalszám esetén sorátlépésnél az azonos színű mezők sorszáma paritást vált.
(pl.: 6x6)

A cikluson belül használunk egy *B_W* logikai típusú változót, ami aszerint vesz fel igaz vagy hamis értéket, hogy a ciklusváltozó értéke páros vagy páratlan. A változót használjuk a későbbiek során a rajzolásnál a négyzet színének beállításához. A ciklusváltozó paritását az előző feladatnál elkészített maradékos osztás blokkal vizsgáltuk. Elosztottuk az értéket 2-vel és a kapott maradékot használtuk az elágazás feltételénél. Ha 1-et kaptunk maradékul, akkor igazra állítottuk a *B_W* értékét, egyébként hamisra. A saját blokk hányados paraméterét nem használtuk és az elágazást egy numerikus értékkel vezéreltük.

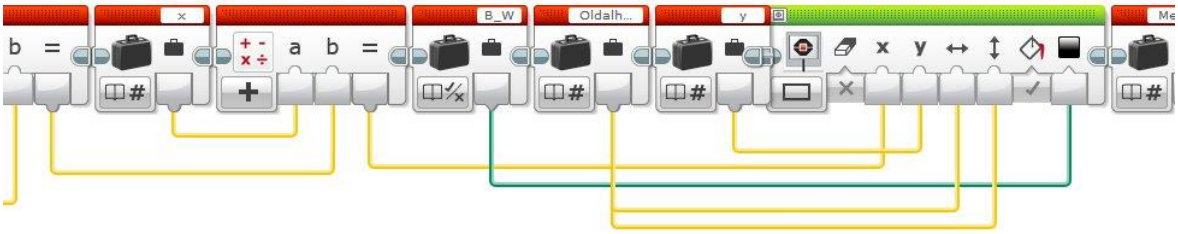


A rajzolandó négyzet bal felső sarokkoordinátáinak meghatározásához ismét a maradékos osztás blokkot használtuk. Ha a ciklusváltozót elosztjuk az egy soron található mezők számával (*Mezoszam*) és az így kapott egész hányadost megszorozzuk egy négyzet oldalhosszával (*Oldalhossz*), akkor megkapjuk a mező oszlopkoordinátáját (*y*). Ha pedig az osztás utáni maradékot szorozzuk az oldalhosszal, akkor az sorkoordinátát (*x*). A következő táblázat konkrét számoknál mutatja be az ötlet helyességét. Legyen az egy soron lévő mezők száma 5. Ekkor összesen 25 mező található a sakktáblán, minden sorban 5, és összesen 5 oszlopban. Az oldalhossz pedig legyen 20 pixel.

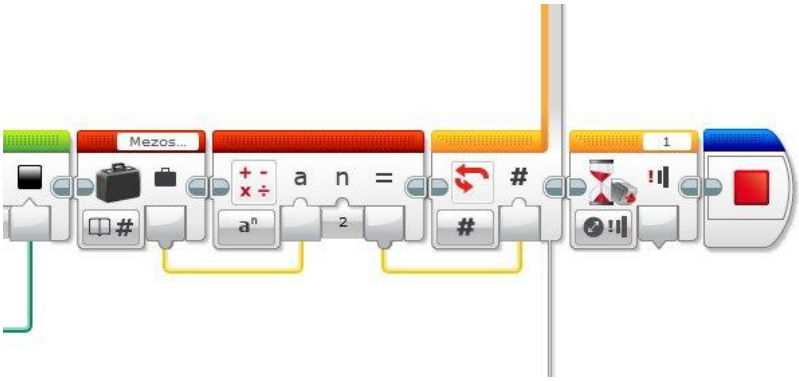
Ciklusváltozó	5-tel osztva a maradék	5-tel osztva a hányados	Maradék szorozva 20-szal	Hányados szorozva 20-szal	A mező bal felső sarkának *x* koordinátája	A mező bal felső sarkának *y* koordinátája
0	0	0	0	0	0	0
1	1	0	20	0	20	0
2	2	0	40	0	40	0
3	3	0	60	0	60	0
4	4	0	80	0	80	0
5	0	1	0	20	0	20
6	1	1	20	20	20	20
7	2	1	40	20	40	20
8	3	1	60	20	60	20
9	4	1	80	20	80	20
10	0	2	0	40	0	40
11	1	2	20	40	20	40
…						



Ha mindez megvan, akkor már csak rajzolni kell a képernyőre a négyzeteket. A bal felső sarok koordinátái (*x*; *y*), az oldalhossz a kezdeti *Oldalhossz* változóban van, és mivel négyzetet rajzolunk a téglalap mindkét oldala ugyanilyen hosszú, a színét pedig a *B_W* változó szabályozza.



A ciklusnak annyiszor kell lefutnia, ahány négyzet van a sakktáblán, ez nem más, mint a *Mezoszam* változó négyzete. A program az ütközésérzékelő benyomására áll le.



Feladatként adódik, hogy módosítsuk a programot úgy, hogy páros mezőszám esetén is működjön.

11.2. Gyakorló feladatok

11/F1. Írjon saját blokkot, amely kiszámítja két koordinátáival adott pont távolságát a koordináta-rendszerben! A blokk bemenő paraméterként megkapja a két pont koordinátáit (egész számok, összesen négy darab: $(x_1;y_1)$ és $(x_2;y_2)$), és visszaadja a két pont távolságát (valós szám).

11/F2. Írjon saját blokkot, amely két szín- vagy fényszenzor segítségével követi az alap színétől különböző színű útvonalat! Az útvonal a két fényszenzor között legyen. A blokk bemenő paraméterként megkapja a motorok sebességét, a fényszenzorok csatlakozási portjainak számát és nincs visszaadott értéke.

11/F3. Írjon saját blokkot, amely egy logikai értéket (igaz/hamis) ad vissza aszerint, hogy a fényszenzorával adott időközönként mért két egymást követő érték egy adott határértéknél (pozitív egész) nagyobb vagy kisebb mértékben tér-e el! Bemenő paraméterként a blokk megkapja a mintavételek közötti időtartamot (valós szám, másodperc), az eltérés mértékét (pozitív egész szám), vagyis a határértéket. Visszaadott értéke egy logikai igaz/hamis érték.

11/F4. Írjon saját blokkot, amely visszaadja egy alap színétől eltérő színű csík szélességén történő áthaladás időtartamát másodpercben (valós szám). A blokk csak akkor működjön ha a robot a csík színétől eltérő színű felületről indul. Bemenő paramétere nincs, visszatérési értéke egy valós szám.

11/F5. Írjon saját blokkot, amely a képernyőre rajzolja az $f(x)=ax+b$ hozzárendelési szabályú függvény képét! A koordináta rendszer origója a képernyő bal alsó sarka legyen! Bemenő paraméterként a blokk megkapja az a és b paraméterek értékét (valós számok), visszatérési értéke nincs.

11/F6. Írjon saját blokkot, amely a képernyőre rajzolja az $f(x)=ax^2+bx+c$ hozzárendelési szabályú függvény képét! A koordináta rendszer origója a képernyő bal alsó sarka legyen! Bemenő paraméterként a blokk megkapja az a; b és c paraméterek értékét (valós számok), visszatérési értéke nincs.

12. MÉRÉSNAPLÓZÁS

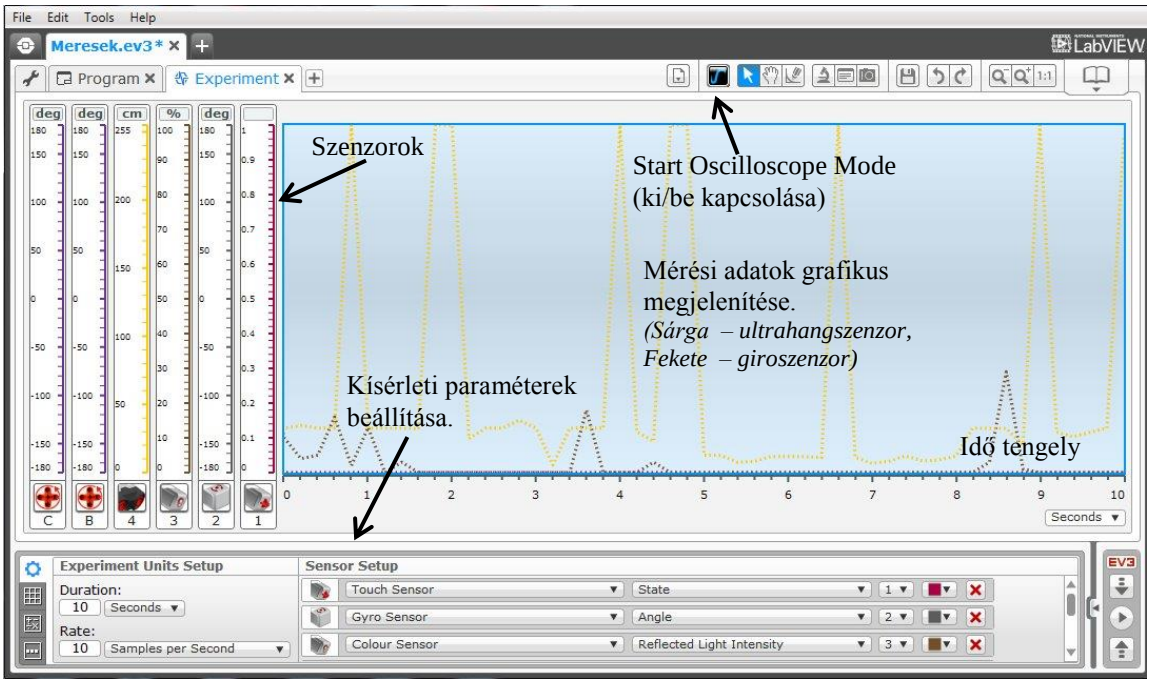
Az EV3 szoftver egyik jelentősen továbbfejlesztett része a szenzorokkal végzett mérések adatainak naplózása, elemzése. Nem csak a mérési adatokat képes a szoftver rögzíteni, hanem online grafikus megjelenítésre is képes abban az esetben, ha robot csatlakoztatva van a számítógéphez. A grafikonok elemezhetők, különböző típusú görbék illeszthetők rá, így a fizikai törvények teljesülése is vizsgálható.
A rendelkezésre álló eszközök és lehetőségek sokszínűsége nem teszi lehetővé a teljes bemutatást, ezért két egyszerű példán keresztül érzékeltetjük a mérések grafikus megjelenítésének és elemzésének egyszerű módját.

12.1. Mérési paraméterek beállítása

A mérés munkafolyamatának kezdeteként létre kell hozni egy „kísérlet” (*Experiment*) lapot.

A projekt *Add* (+) jelére kattintva a listából a *New Experiment* funkciót választva a programozói felület helyett egy kísérlet beállítási felülethez jutunk.

Első ránézésre bonyolultnak tűnhet, hiszen alapértelmezésként a rendszer minden téglához csatlakoztatott szenzort megjelenít, és el is kezdi az online mérést a szenzorok aktuális értékeinek grafikus megjelenítésével.
A képernyő bal oldali részén láthatók a szenzorok: a két motor (C és B) elfordulási szögeit mérő szenzor, az ultrahangszenzor, a színérzékelő, a giroszenzor és az ütközésérzékelő. Az oszlopokon a mérési skála, az oszlopok tetején a beállított mértékegység látszik. A képernyő alján kaptak helyet a mérési paraméterek beállítási lehetőségei.



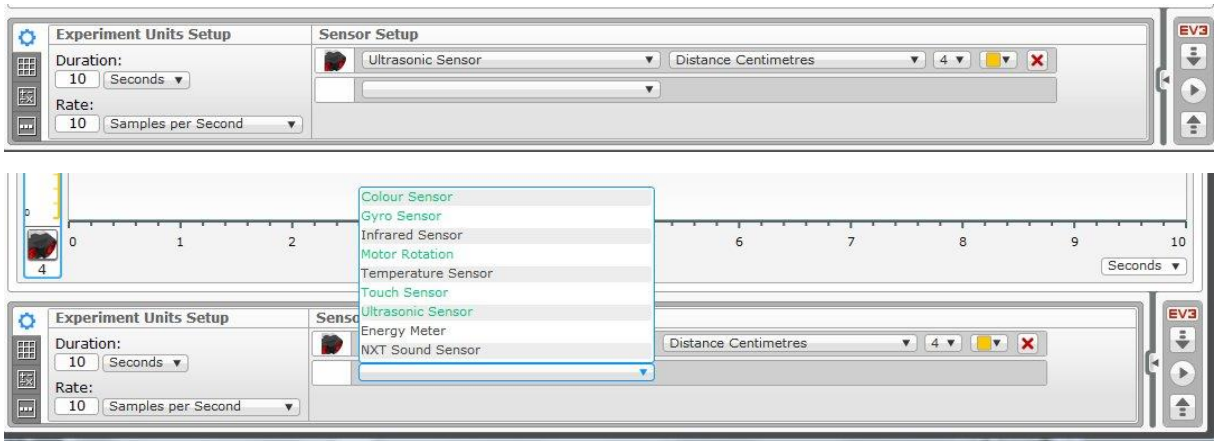
A kék hátterű részen az adatok grafikus képét látjuk, a különböző szenzorok értékeit különböző színnel ábrázolva. A vízszintes tengelyen az idő értékek látszanak. A grafikon folyamatosan változik az idő

teltével, mint egy oszcilloszkóp képernyőjén. A mérést leállítani a jobb felső sarok ikonjai közül a *Start Oscilloscope Mode* gombon kattintva lehet.
A tényleges mérés megkezdése előtt a kísérleti beállításokat végezzük el.

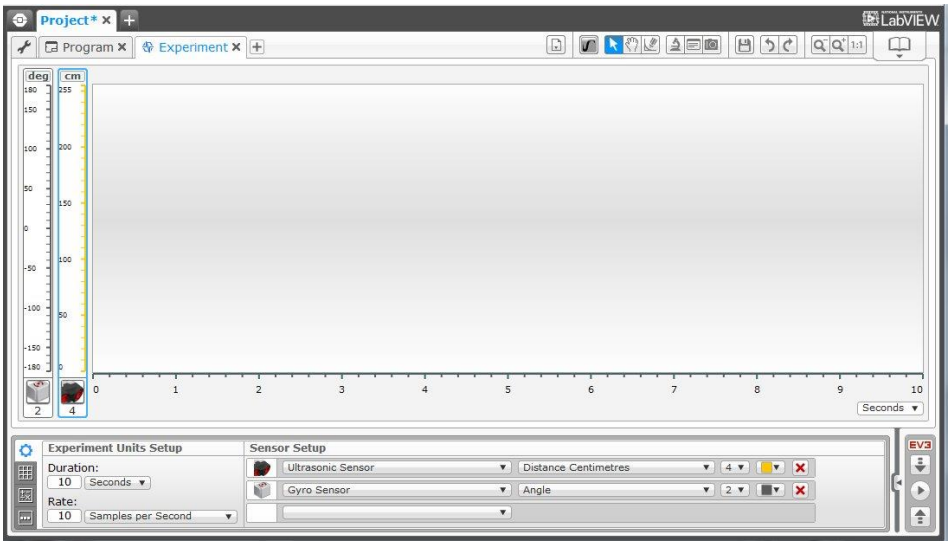
12.1.1. Első kísérlet

A kísérletben két szenzort használunk. Az ultrahangos távolságérzékelővel mérjük a robot előtti akadályok távolságát, miközben a robot lassan körbe forog. A giroszenzorral pedig mérjük a robot elfordulási szögét. Így a két mérési adatsort egy grafikonon megjelenítve mintegy szonár, szkenner funkcionál. Meg tudjuk állapítani, hogy a robot környezetében, tőle milyen távolságra voltak akadályok. Az elfordulási szög mérésével a kiindulási pozícióhoz képest az akadályok helyzetét is meg tudjuk adni.

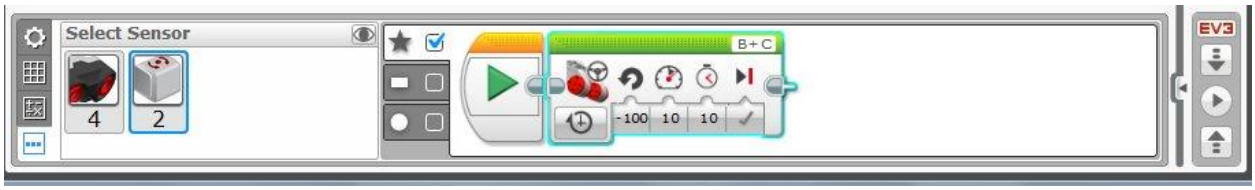
Első lépésben állítsuk be, hogy melyik két szenzort fogjuk használni. A paraméterterületen válasszuk ki a megjelenő listából a két szenzort (a piros x-szel tudjuk a szükségtelen szenzorokat eltávolítani).



Beállíthatjuk a mérési gyakoriságot és az adatrögzítés időtartamát is.
Duration (időtartam): 10 másodpercre állítottuk.
Rate (mérési gyakoriság): 10 mintavétel másodpercenként, tehát 0,1 másodperces gyakoriság.



Az giroszenzor az elfordulási szöget, míg az ultrahangszenzor a távolságot méri centiméterben. A grafikonok színe narancssárga (ultrahangszenzor) és szürke (giroszenzor) lesz.
A kísérlet során a robotot helyben szeretnénk forgatni, ezért ehhez egy programot kell írnunk. A paraméterezési felület negyedik lapján (*Graph Programming*) erre lehetőségünk van.

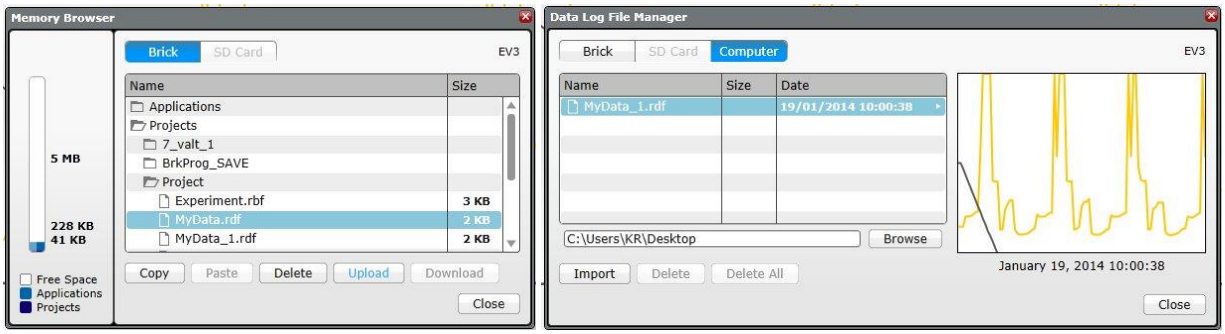


A programozási felületen megszokott *Action* csoport moduljait használhatjuk a program előállításához. A példánál a robot 10 másodpercig forog 10-es sebességgel a saját tengelye körül.

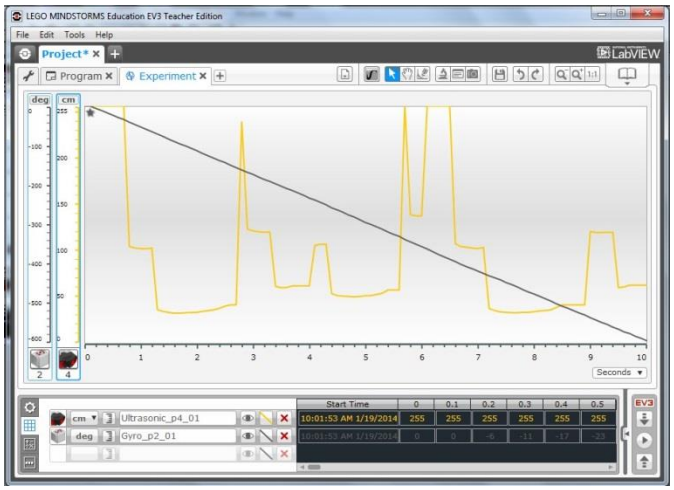
Ahhoz, hogy a mérés során a program elinduljon a programsor bal felső sarkánál lévő jelölőnégyzetet be kell jelölni (a példánál a csillag szimbólum mellett). Összesen három különböző programot készíthetünk.

A mérést elkezdeni a jobb szélen lévő Start gombon kattintva lehet, vagy a robotra a szokásos módon áttöltve a képernyőmenüben is kiválasztható és elindítható (*Experiment* néven).

A mérés elindítása után, ha a robot csatlakoztatva van a számítógéphez, akkor automatikusan megjelenik a mérési grafikon a képernyőn. Ha nem csatlakozik, akkor a mérési adatok a robot belső tároló memóriájába kerülnek, ahonnan áttölthetők a számítógépre (*rdf* kiterjesztésű fájlként), majd importálhatók a projektbe és a grafikus megjelenítés az áttöltés után automatikus.



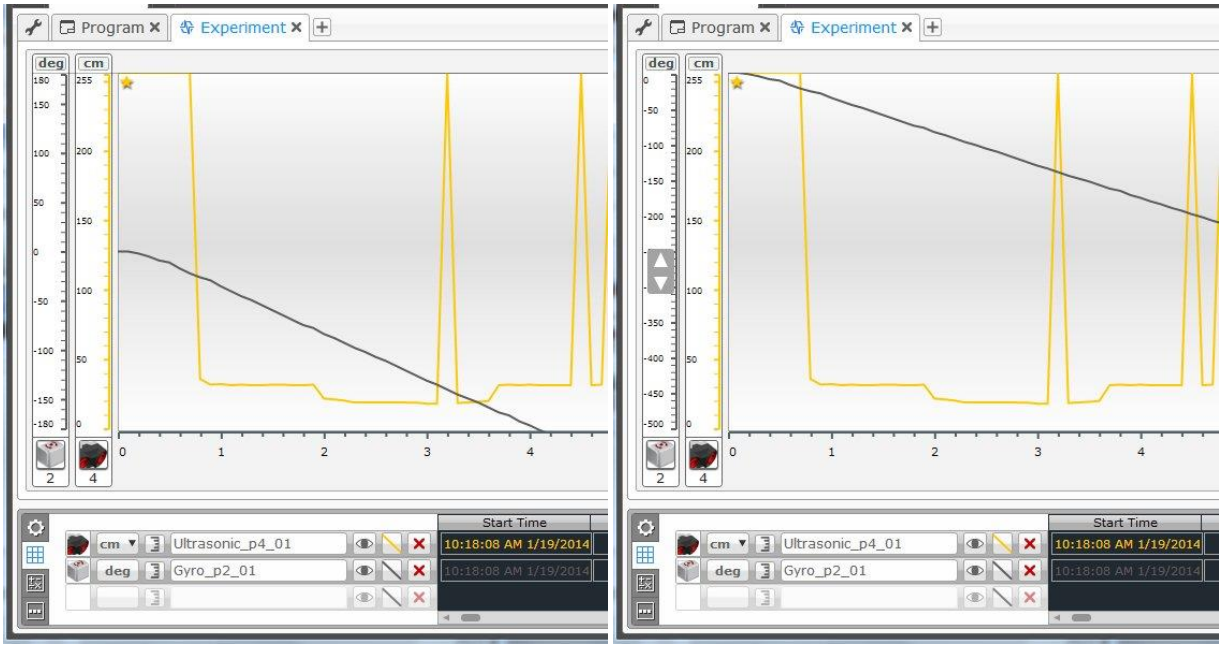
A fenti kísérlet elvégzése után a mérési adatok grafikonja:



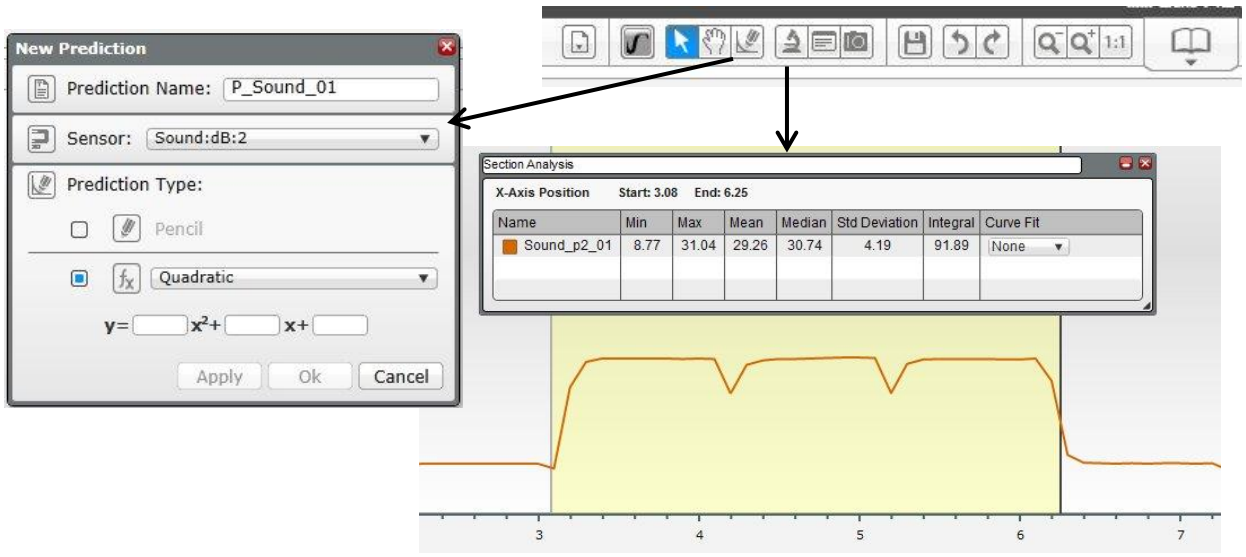
A sárga színű görbe az ultrahangszenzor által mért távolságadatokat, míg a szürke görbe a giroszenzor által mért elfordulási szög adatokat mutatja.

A függőleges tengely skálatartományán érdemes változtatni. Van olyan lehetőség, hogy a mérési adatoknak megfelelően a rendszer automatikusan átskálázza a tengelyt. Az alábbi két példán jól látható a különbség a giroszenzor adatain. Az első esetben a skálatartomány -180 – +180 közötti. Mivel a robot egy irányba forgott, így a forgásszögeknél csak a negatív tartományra van szükség. Az adott

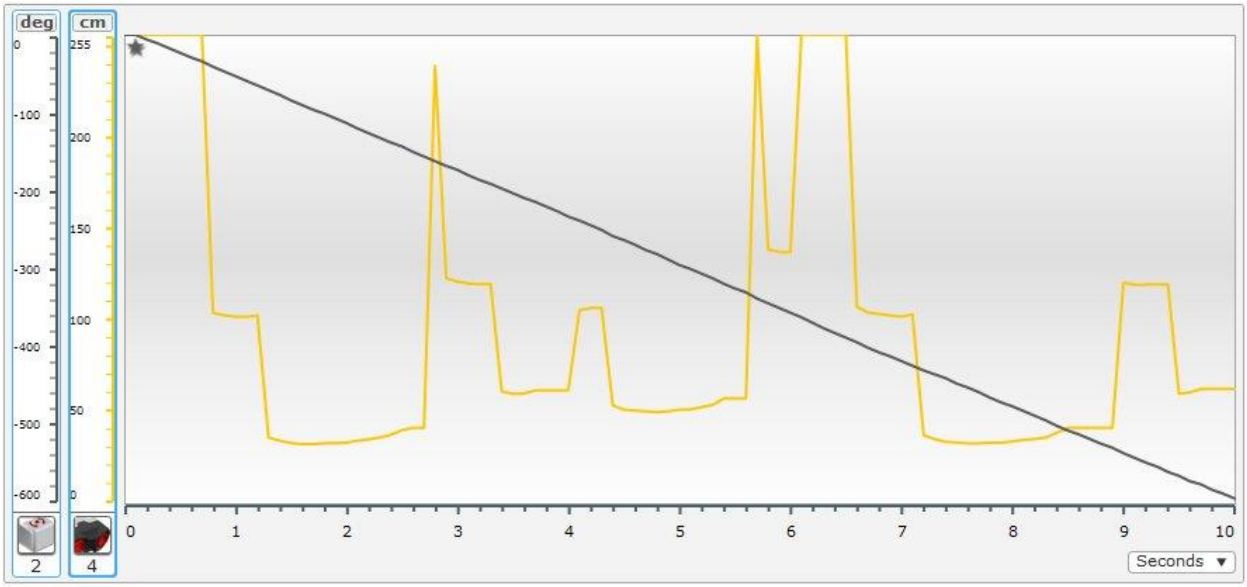
skálára kattintva lehet az optimalizált skálabeosztást létrehozni. A második grafikon esetén a skála o és -500 fok közötti.



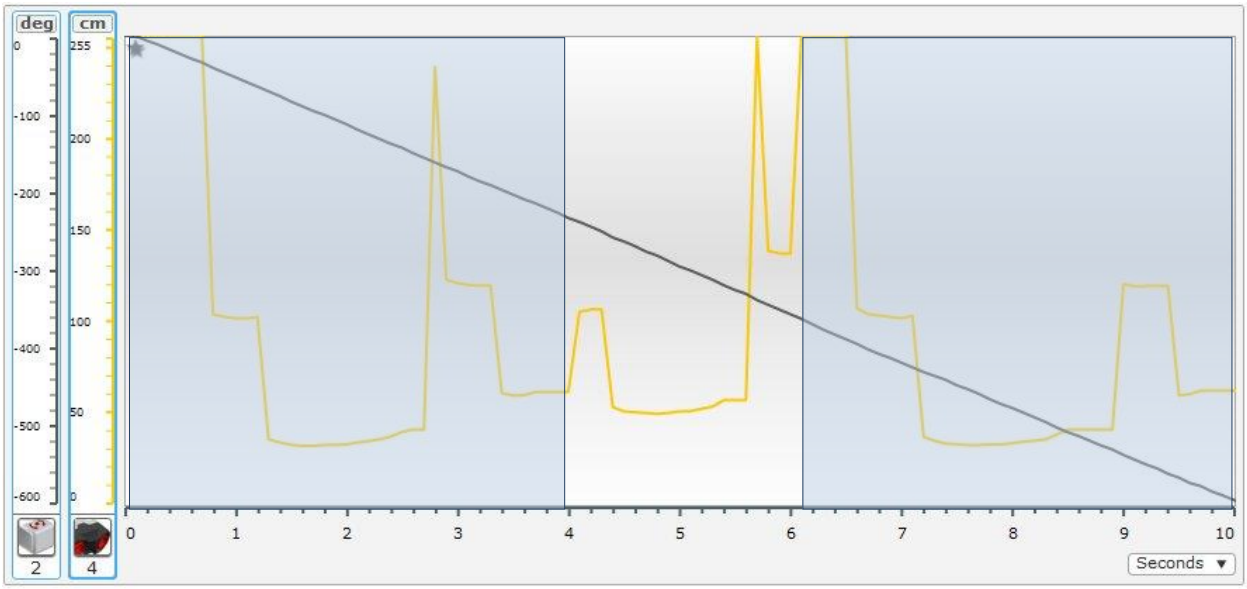
Ha elvégeztük a mérést, akkor az adatok értelmezése a következő lépés. Ehhez sok hasznos segédeszközt biztosít a rendszer. Lehet adott hozzárendelési szabállyal különböző grafikontípusokat illeszteni a képernyőre. Ezzel vizsgálható a mért értékek fizikai törvényekhez illeszkedése. A mérési adatok számszerűen is leolvashatók a grafikon alatti területen, kiemelhetünk mérési pontokat vagy intervallumokat, amelyekhez tartozó értékekről egy gyors statisztika jelenik meg a képernyőn. A jobb felső sarokban lévő ikonsoron találhatók az elemző eszközök ikonjai.



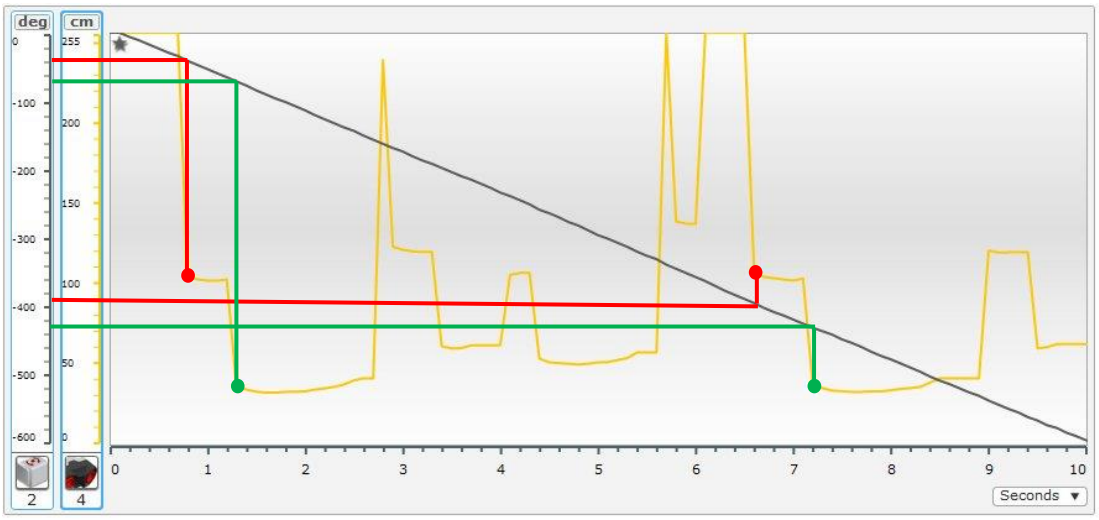
Sok lehetőségünk van, amelyek használatához a szoftver *Help*-je nyújt további segítséget.
Mindez csak egy eszközkészlet, amely segít az elemzésben, de a tényleges interpretációt a felhasználónak kell elvégeznie.



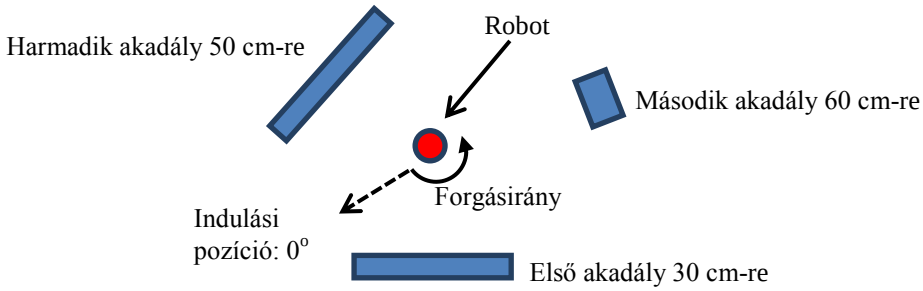
Ha megnézzük a grafikont, akkor láthatjuk, hogy a robot kb. 600°-ot fordult. A fordulásnál tehát kell találnunk ismétlődő távolságértékeket, hiszen volt olyan rész a környezetében, amelyet kétszer is vizsgált. A szürke színű grafikon közel egyenes, hiszen a forgás egyenletes volt. Ha megkeressük a 360°-os értéket a függőleges tengelyen, akkor az ismétlődés kezdetét is megtaláljuk. Ez kb. 6 másodpercnél van. Ha eltekintünk a 2,8 másodpercnél lévő ultrahangszenzor által mért kiugró távolságértéktől (valószínűleg mérési hiba), akkor a sárga grafikon valóban ismétlődő mintát mutat.



Azt is leolvashatjuk a grafikonról, hogy mekkora forgásszögnél milyen távolságot mért a robot, tehát ez alapján megrajzolható vázlatosan a környezete.
Például kb. 36°-nál (396°-nál) érzékelt először akadályt, kb. 100 cm-re. Ezután kb. 65°-nál (425°-nál) az akadályt már 30 cm-nél érzékelte.
320° után már nem volt újabb akadály.



Ha ez alapján szeretnénk megrajzolni a robot körüli képet, akkor a következő ábrát kaphatjuk.



12.1.2. Második kísérlet

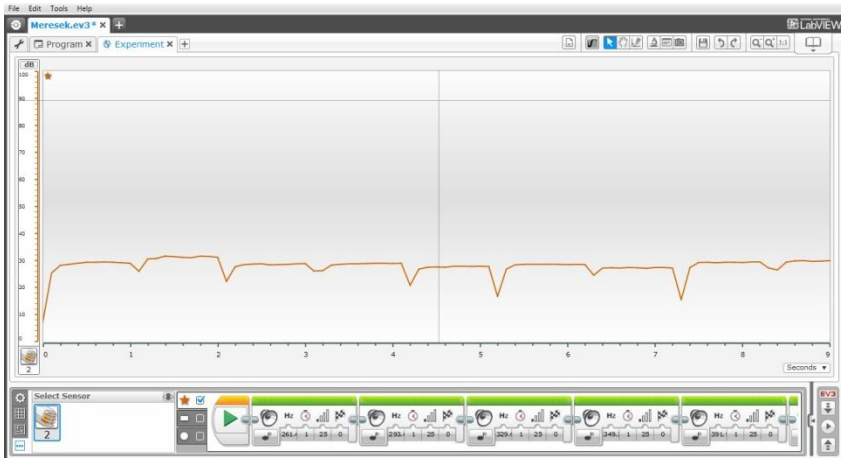
A kísérlet során a robot hangszenzorát használjuk (NXT szenzor) hangnyomás (decibel, hangerő) mérésére. A robot hangszórójával különböző frekvenciájú hangokat szólaltatunk meg, és ugyanakkor mérjük a hangszenzor értékeit.

A hangszenzor csatlakoztatása után válasszuk ki a listából, majd készítsük el a programot, amely 9 hangot szólaltat meg egymás után. A frekvenciák a zenei egész hangok frekvenciáival megegyezőek: C, D, E, F, G, A, H, C, D.

Minden hang 1 másodpercig szól, a rögzítés időtartama így 9 másodperc, a mintavételi arány 10 minta/másodperc.

Azt várjuk, hogy a különböző frekvenciájú hangok azonos decibel értékeket szolgáltatnak, hiszen a hangerő egységesen 25%.

A várakozásnak megfelelően a kapott grafikon is ezt mutatja.



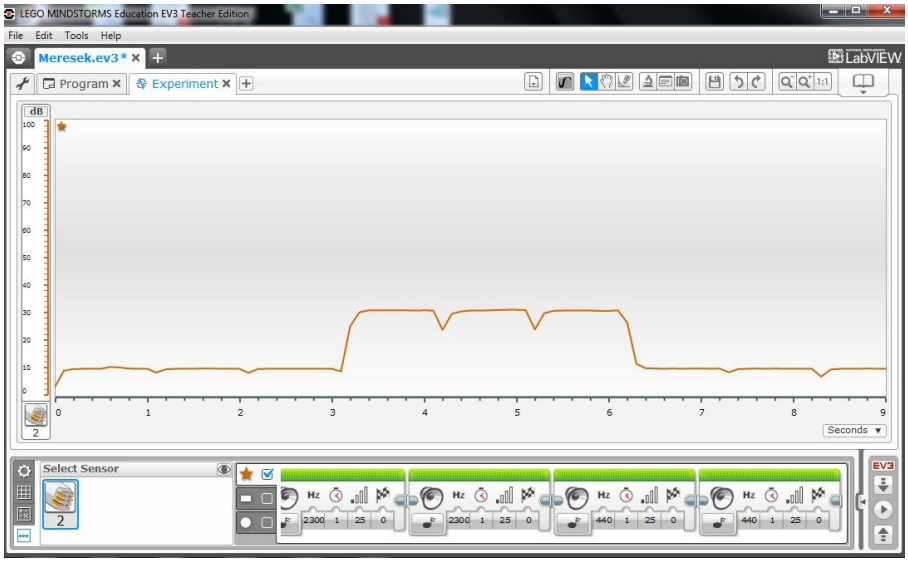
A grafikonon jól megfigyelhető az egyes hangok közötti váltás során a felfutó és lefutó szakasz, tehát a hang erősödése és halkulása. Mivel ez nagyon rövid időtartam, ezért emberi füllel nem észlelhető.

Ha a hangok frekvenciáit úgy módosítjuk, hogy az első 5 hang jóval magasabb legyen, akkor a mért értékek szórást mutatnak.

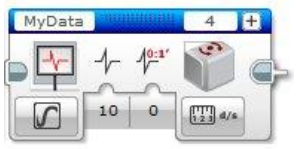
Valahol a 2200 Hz-től a mért értékeket a hangszenzor kisebb hangnyomás értékkel azonosítja. Mindez a mikrofon (hangszenzor) karakterisztikája miatt van így. A magas hangok esetén a mért érték 10 db körül van, míg a mélyebb hangok esetén 30 db körül.

Ez adhat egy ötletet. Ez alapján meg tudunk különböztetni hangokat a hangszenzor segítségével. Legalább két olyan értéket találhatunk, amely esetén mást fog mérni a szenzor. Ez alkalmas lehet a robotok közötti adatcserére, kódolásra. Ha az egyik hangot tekintjük az egyik jelnek, míg a másik hangot a másik jelnek, máris van egy kettes számrendszerbeli kódunk, amivel kommunikálni tudunk.

Jól kell megválasztani a frekvencia értékeket: pl.: 2300 Hz az egyik jel, míg 440 Hz a másik. Az egyik robot váltakoztatva a frekvenciákat megszólaltatja a kódsort. A másik robot a hangszenzorával méri a decibel értékeket. Eltárolja őket, majd kellő tűréshatárral (pl.: 20 db alatti mért értéknél magas, míg egyébként mély hangként azonosítja) átkódolja. A kódolt jelsorozat alkalmas például a robot motorjainak vezérlésére. Magas jel esetén tolat, mély jel esetén előre mozog. Csak a tárolás és átkódolás után érdemes a mozgást elindítani, mert a motorok működési zaja befolyásolja a mért értékeket. Ha kódsort nagyobb csoportokban értelmezzük, pl. két egymás utáni hang felel meg egy kódnak, akkor már négy különböző állapotot tudunk megkülönböztetni: magas-magas → 0; magas-mély →1; mély-magas →2; mély-mély →3.
A fenti beállításokkal a segélykérés SOS jelzésének „hangképe" az alábbi grafikont eredményezi:



A bemutatott két példa csak ízelítő az eszköz kreatív használatához. Például fizikai törvények mérésekkel igazolásához is alkalmas lehet, megfelelő szenzorok használatával.



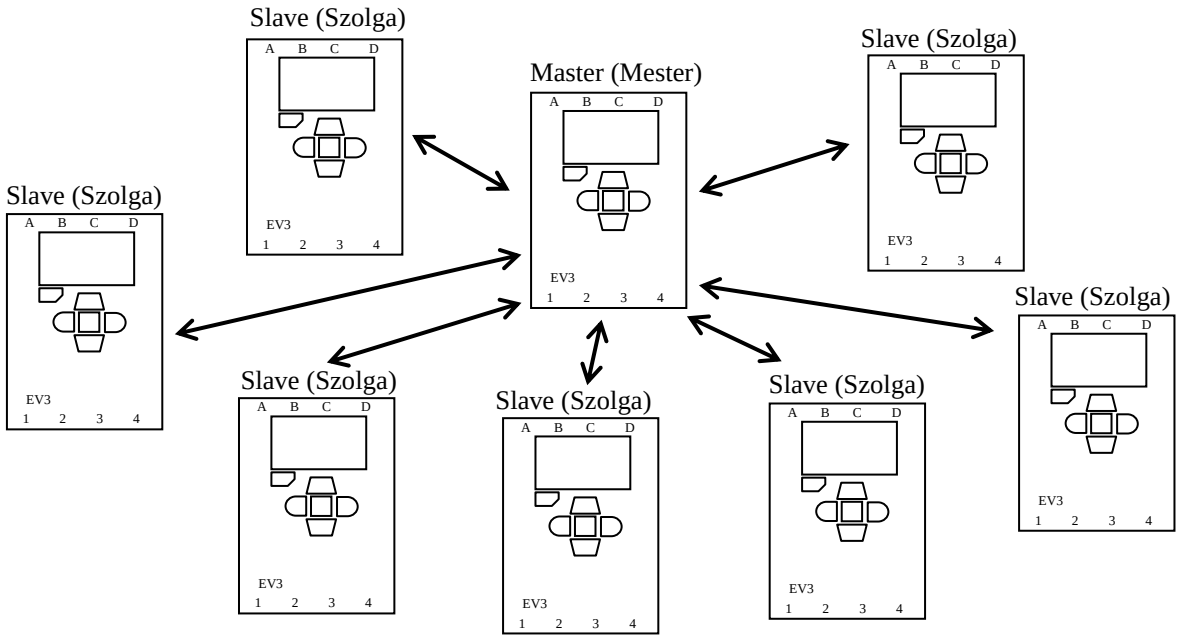
A mérési adatok számítógépes kapcsolat nélküli rögzítéséhez egy programozási modul is rendelkezésre áll, amit használva a mérési adatok a robot memóriájában tárolódnak, ahonnan számítógépre tölthetők és szövegfájlként importálva táblázatkezelő programban feldolgozhatók.

A modul az Advanced csoportban található. Paraméterezése egyszerű, több szenzor adatainak egyidejű rögzítése is beállítható.

13. KOMMUNIKÁCIÓ

A robotok a beépített bluetooth technika miatt képesek egymással is kommunikálni. Természetesen más bluetoothos kommunikációra alkalmas eszközzel is, például mobiltelefonnal, PDA-val vagy számítógéppel. Ezekre az egyéb eszközökre először telepíteni kell a megfelelő robotkezelő alkalmazást (letölthetők pl. az internetről), majd ezután képesek a kommunikációra.

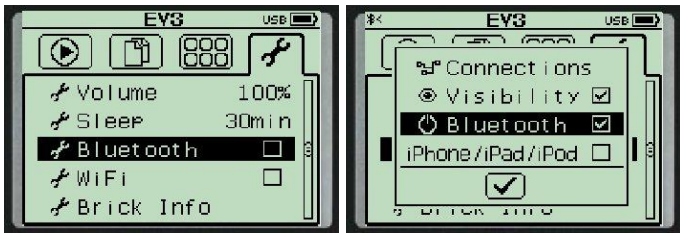
A robotok közötti kommunikáció *master-slave* (mester-szolga) alapú. Ez azt jelenti, hogy az egymással kommunikációs kapcsolatban álló robotok között van egy kitüntetett szerepű, amelyen keresztül az adatok továbbítása folyik (*master*). Az EV3 bluetooth protokollját úgy készítették el, hogy egy mester robot, és hét szolga kapcsolódhat össze. Összesen tehát nyolc robotot kapcsolhatunk össze bluetooth alapú hálózattá.



A szolga robotok csak a mesterrel kommunikálhatnak, így egymásnak szánt üzeneteiket is csak a mesteren keresztül küldhetik. Egy robot egyszerre vagy mester vagy szolga lehet.
A kommunikáció megkezdése előtt fel kell építeni a bluetooth kapcsolatot. A kapcsolat felépítését mindig a mesternek kijelölt roboton kell kezdeményezni, és általában az EV3 tégla képernyőmenüjének megfelelő funkciójával, de programból is elvégezhető. A kapcsolat mindaddig megmarad, amíg le nem bontjuk, vagy valamelyik robotot ki nem kapcsoljuk. Ha nem használjuk a beépített bluetooth adóvevőt érdemes kikapcsolni, mert az akkumulátort használja, így annak feltöltöttségi szintje gyorsabban csökken. A képernyő bal felső sarkában lévő ikon jelzi, hogy be van-e kapcsolva a bluetooth eszköz.

12.1. A bluetooth kapcsolat felépítése

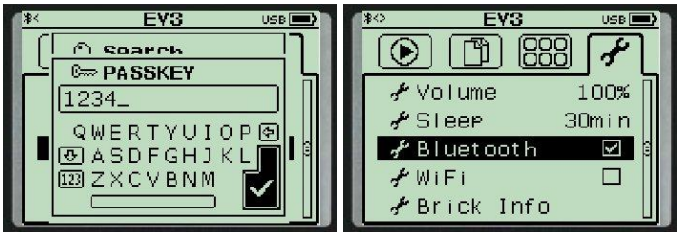
Első lépésként a bluetooth adaptert kell a roboton bekapcsolni. Ezt célszerű a képernyőmenün keresztül megtenni.



A bekapcsolt állapotot a képernyő bal felső sarkában megjelenő „<" szimbólum jelzi.
Az első kapcsolat felépítésekor a mester robot képernyőmenüjének *Connections/Search* menüpontjával érdemes megkerestetni a hatókörön belüli kommunikációra képes eszközöket (ez eltarthat néhány másodpercig).

Ezután a listából kiválasztható az eszköz (pl. egy másik robot), amellyel a kommunikációt szeretnénk felépíteni. A robot a kapcsolódó partnert a neve alapján fogja azonosítani a kommunikáció során.

Az első kapcsolatépítésnél egy kódcsere is megtörténik, amely mindkét kapcsolatban részt vevő eszköz képernyőjén megjelenő kód beállítását és elfogadását jelenti.
A csatlakozás állapotát a bal felső sarokban lévő ikon megváltozása is jelzi („<>").



Ha a kapcsolat felépült, akkor a rendszer megjegyzi a kódokat, és a továbbiakban a keresés helyett a *Connections* menüpontban is megjelenik az eszköz, amelyet kiválasztva csatlakozást lehet kezdeményezni.
Ha a kapcsolat kiépült, akkor a programok már feltölthetők a robotokra.
A kapcsolatot programból is lehet kezdeményezni az *Advanced* csoport *Bluetooth Connections* blokkjával.
Az alábbi kódrészlet felépíti a kapcsolatot az előző példánál használt „Slave" fantázianevű robottal.

12.2. Robotok közötti kommunikáció programozása

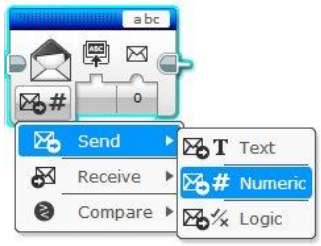
A robotok közötti kommunikáció azt jelenti, hogy a mester a szolgáknak, és a szolgák a mesternek tudnak átküldeni adatokat (számokat, szöveget vagy logikai értékeket) a keretprogram megfelelő moduljának kiválasztásával és programba illesztésével.
Az üzenetcseréhez három dolgot kell megadni:

- A robotokat, amelyek között a kommunikáció zajlik. Ezek közül az egyik, amelyiken a program fut, a másik robot nevét pedig a megfelelő ikonon kell beállítani. A robotot tehát mindig a neve alapján azonosítjuk.

- Az üzenet egyedi azonosítóját. A kommunikációs modul jobb felső sarkában szereplő szövegdobozba lehet beírni. Mind a küldő, mind a fogadó roboton ezzel az azonosítóval lehet az üzenetre hivatkozni.

- Az üzenetet, amely lehet szám, szöveg vagy logikai érték.

Az üzenetküldés és fogadás is egyetlen blokkal megvalósítható, mind a mesteren, mind a szolgán azonos módon.
A modul az *Advanced* programcsoport *Messaging* blokkja.



A működési mód beállítása szerint lehet üzenetet küldeni (*Send*), fogadni (*Receive*), és használható a blokk Összehasonlító módban (*Compare*). Ez utóbbi azt jelenti, hogy a megkapott üzenetet össze tudjuk hasonlítani egy értékkel és a kimenet aszerint lesz igaz vagy hamis, hogy egyezés van-e.

Az üzenet három féle lehet: szöveg, szám vagy logikai érték.

A kommunikációs programoknál mindig figyelni kell arra, hogy az üzenetek küldése és fogadása szinkronban legyen. Ez azt jelenti, hogy az üzenet olvasását végző modul egyszer nézi meg a *mailbox* tartalmát, vagyis az üzenetettárolót akkor, amikor éppen rá kerül a végrehajtás sora. Ha akkor éppen nincs még ott az üzenet, akkor a kommunikáció sikertelen. Ezt vagy úgy tudjuk kivédeni, hogy az üzenet olvasását késleltetjük, tehát olyankor olvassuk az üzenetet, amikor már biztosan ott van. Ez nehezen kiszámítható egy bonyolultabb program esetén. Egy másik, gyakrabban használt megoldás, ha egy ciklusba tesszük be az üzenetolvasási utasítást, így folyamatosan tudjuk figyelni a *mailbox*ot.
Egy harmadik lehetőségünk is van, és ez tűnik a legegyszerűbb megoldásnak. A *Wait* modulnak van egy *Messaging* működési módja, amelynél be tudjuk például azt állítani, hogy addig ne lépjen tovább a program a következő utasításra, amíg meg nem változott (*Change* mód) az üzenettároló tartalma.

Az ikonon például az „abc" azonosítójú szám típusú üzenet értékének 10-zel történő megváltozása esetén lépünk a programszál következő utasítására. Maga az üzenet a blokk kimeneti paraméterén olvasható ki, és a blokk folyamatosan figyeli üzenet érkezését.

12/P1. Írjon programot, amelyben két robot kommunikációja valósul meg! A mester robot a „Hello" szót küldi át a szolgának, amely ezt megjeleníti a képernyőjén.

A mester robot programja nagyon egyszerű, Első lépésben felépíti a kapcsolatot a „Slave" nevű robottal, majd az ütközésérzékelő benyomására átküldi a „Hello" szót.

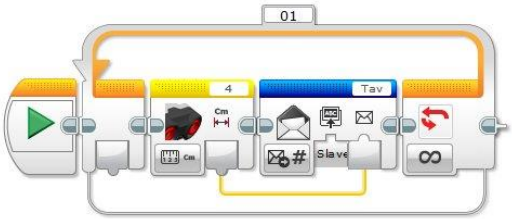
Az üzenet egyszeri elküldése után a program le is áll. Az üzenet azonosítója „Szia”. A fogadó roboton ugyanezzel a névvel lehet hivatkozni az üzenetre.

Mivel a mester programja csak egyszer küldi el az üzenetet, ezért a szolga programját kell előbb elindítani, hogy már felkészült állapotban várja az üzenet megérkezését.

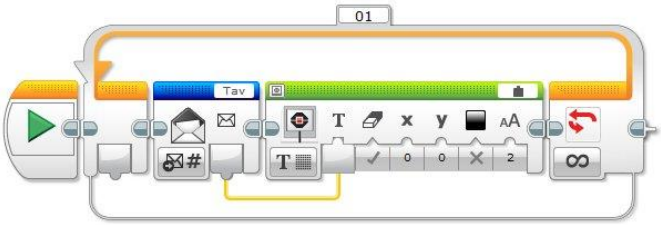
A szolga robot folyamatosan figyeli a kapott üzeneteket (*Wait* blokk *Messaging*), és amint változás következett be, kiírja az üzenetet a képernyőre, majd 5 másodpercig várakozik, és leáll.
A szolga robot esetében úgy oldottuk meg a szinkronizálási problémát (tehát azt, hogy csak az üzenet megérkezése után van értelme azt a pufferből (üzenettárolóból) olvasni), hogy a *Wait* blokkal folyamatosan figyeltük az üzenetpuffer tartalmát. Természetesen használhattunk volna ciklust is a figyelésre.

12/P2. Írjon programot, amely két robot közötti kommunikációt valósít meg! A mester robot folyamatosan küldi a szolgának az ultrahangszenzora által mért értékeket, a szolga robot pedig megjeleníti ezt a képernyőjén. A programok kikapcsolásig fussanak!

A mester robot programja egy végtelen ciklusba helyezett két utasításból áll. Az ultrahang szenzor által mért adatot küldjük át a „Slave” nevű szolgának. Mindezt folyamatosan, tehát végtelen ciklusba illesztve. Az adatküldés modul (*Messaging)* paraméterénél *Number* típust kell beállítatni. Az üzenet azonosítója „Tav”.



A szolga robot esetén a megkapott szám típusú adatot jelenítjük meg a képernyőn. A képernyőtörlés be van kapcsolva. Ügyelni kell arra, hogy az üzenet azonosítója ugyanaz legyen, mint a küldő félnél, jelen esetben „Tav”.

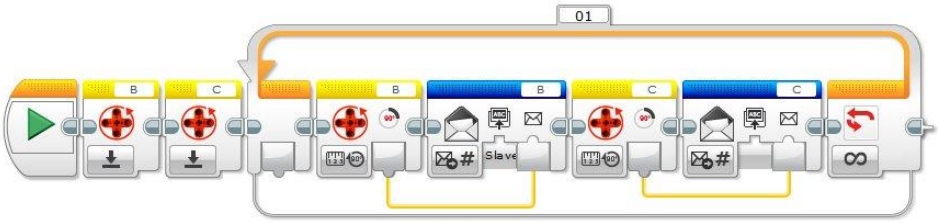


Mivel mindkét program végtelen ciklust tartalmaz, ezért mindegy, hogy melyiket indítjuk először.

12/P3. Írjon programot, amely két robot közötti egyirányú kommunikációt valósít meg! A mester robot mint távirányító, a kommunikációs csatornán küldjön jeleket a szolga robotnak, amely a megkapott jelek alapján mozogjon!

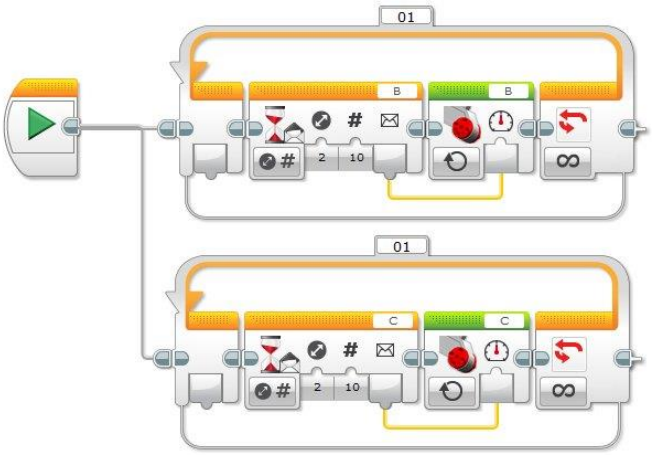
A feladat első olvasatra nehéznek tűnik és nincs jól meghatározva, hiszen nem írtuk le, milyen jeleket küldjön a mester, a szolga mozgását mennyire részletesen szeretnénk vezérelni, ...
Sokféle megoldás lehetséges. Az egyik legegyszerűbbet mutatjuk be.
A mester robotra két motor csatlakozik a B és C porton keresztül. A motorok elfordításának előjeles szögét küldjük át a szolga robotnak. A szolga robot két motorjának sebességértékeként használja fel a megkapott számokat. Tehát, ha a mester robot B motorjának tengelyét elfordítjuk 60°-os szöggel, akkor az átküldött érték a +60. Ezt a szolga robot megkapva, szintén a B motorját forgatja 60-as sebességgel. Ha mester robot C motorjának tengelyét elforgatjuk –40°-os szöggel, akkor az átküldött érték a –40, és a szolga robot –40-es sebességgel forgatja a C motorját, ami ellentétes irányú forgatást jelent. Gyakorlatilag mint két botkormánnyal tudjuk irányítani a szolga robotot a mesteren keresztül. Minél jobban elforgatjuk a motorokat, annál gyorsabban mozog a szolga, illetve az ellentétes irányú forgatás a mesteren a szolga robot ellentétes irányú mozgását eredményezi. A kanyarodáshoz a két motort ellentétesen kell forgatni a mester roboton, így a szolga kanyarodni fog.
Az elv egyszerű, a megvalósítás is.

Mester robot programja:



Kezdésként kinullázzuk a motorok fordulatszámlálóit (*Reset* mód *Motor Rotation* blokk). Ezt csak egyszer kell megtenni, mert a ciklus indulása után már folyamatos az adatküldés. A motorok elfordulás érzékelőinek értékeit folyamatosan küldjük a Slave nevű robotnak, mégpedig a „B" illetve „C" címkéjű üzenetben.

A szolga robot programja:



A szolga robot két külön szálon vezérli a két motorját. A B motort a „B" azonosítójú üzenet tartalma alapján, míg a C motort a „C" címkéjű üzenet értéke alapján. A motorok *On* módban vannak és a sebességük lesz a megkapott érték. A motorok csak akkor kapnak új bemenő sebesség értéket, ha

10-zel megváltozik az üzenet értéke. Ez azt jelenti, hogy a távirányítón (mester robot) legalább 10°-kal valamelyik irányban el kellett forgatni a motor tengelyét ahhoz, hogy változás történjen a mozgásban. Amíg a megfelelő mértékű elforgatás nem történik meg a szolga robot a korábbi sebességgel mozog. A 10-es érték állítható kisebbre is, így elvileg finomabb irányítás valósítható meg.

Az eddig bemutatott példák két robot közötti egyirányú kommunikációt valósítottak meg. A mester robot küldött üzeneteket, amelyeket a szolga fogadott és reagált rájuk. Ha a feladat során kétirányú kommunikációt szeretnénk megvalósítani, akkor a szolga is küldhet válaszként üzeneteket a mesternek. A mester robotot az üzenetküldés során szintén a nevével tudjuk azonosítani. Ha egy mesterre két szolga kapcsolódik, és a két szolga egymásnak szeretne üzenetet küldeni, akkor azt a mesteren keresztül tehetik meg. Az egyik szolga elküldi az üzenetet a mesternek, majd az a másik szolga felé küldi tovább. Minden robot csak annak küldhet tovább üzenetet, amelyikkel felépítette a bluetooth-os kapcsolatot. A *Messaging* blokkban a címzett robot nevének helyén egy legördülő listában megjelennek azoknak a robotoknak a nevei, amelyek elérhetők a kommunikációban.

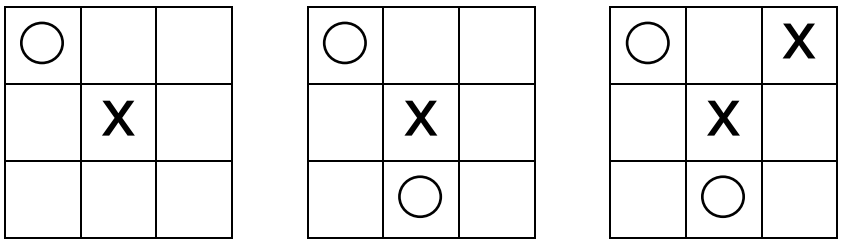
12.3. Gyakorló feladatok

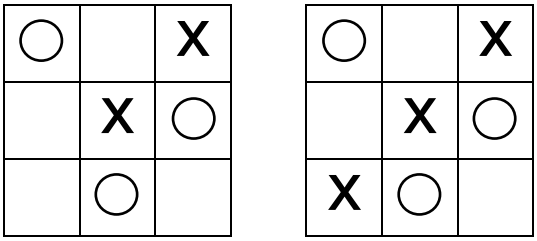
12/F1. Írjon programot, amely két robot kommunikációját valósítja meg! A mester robot sorsol egy véletlen számot, és átküldi szolga robotnak, amely ezt a számot kiírja képernyőjére, majd egy üzenetet küld vissza a „megkaptam" szöveggel. A mester, miután vette a szolga üzenetét, azt kiírja a képernyőre, és a programja néhány másodperc várakozás után leáll.

12/F2. Írjon programot, amely két robot kommunikációját valósítja meg! A mester robot egyenesen halad állandó sebességgel és folyamatosan küldi a fényszenzora által mért értékeket a szolga robotnak, amely a képernyőjére írja azokat.

12/F3. Írjon programot, amely két robot kommunikációján keresztül egy egyszerű játékprogramot valósít meg! Mindkét robot a képernyőjére rajzol egy 3x3-as rácsot, amely játéktáblaként funkcionál. Az egyik roboton azt lehessen beállítani, hogy az emberi játékos a kilenc mező közül melyikre lép. A kiválasztott mezőn jelenjen meg egy kör! A másik robot véletlen sorsolással határozza meg, hogy az üresen maradt mezők közül Ő hová lép. Itt jelenjen meg egy „X". A lépéseket addig folytassák felváltva, amíg van üres mező, vagy valamelyik játékosnak vízszintesen, függőlegesen vagy átlósan három azonos formájú jelből álló sora kialakult (Ő lesz a győztes). A játék a 3x3-as amőba (TicTacToe). A robotjátékos stratégia alapján is játszhat, nem csak véletlen sorsolással.

Mindkét robot képernyőjén jelenjen meg a tábla és az állás!





12/F4. (*nehezebb feladat*) Írjon programot, amely a hagyományos torpedó játékhoz hasonló játékot valósít meg! Az egyik robot véletlen sorsolással elhelyez pl. 10x10-es négyzetrácsos hálón 1, 2 és 3 négyzetből álló síkidomokat, amelyek kezdetben nem látszanak. A játékosok felváltva adják meg a két roboton a lövéseik koordinátáit (sor; oszlop). Mindkét játékos képernyőjén látszik a saját lövéseik helye és hogy volt-e ott síkidom (hajó) vagy sem. Találat esetén a játékos kap egy pontot, egyébként nem. A cél minél hamarabb (kevesebb lövésből) eltalálni a síkidomok helyét. A kommunikáció során a pályán elhelyezett síkidomok helyzete mindkét robot számára ugyanaz, tehát a kisorsolt pozíciókat mindkét robot ismeri. A játékosok lövéseit is ismeri mindkét robot, de csak a saját képernyőn jelennek meg. A játéknak akkor van vége, ha valamelyik játékos valamennyi hajót eltalálta. A játékosok felváltva „lőnek”. A játék végén mindkét robot kiírja az elért pontszámokat.

14. EGYEBEK

14.1. Hanglejátszás

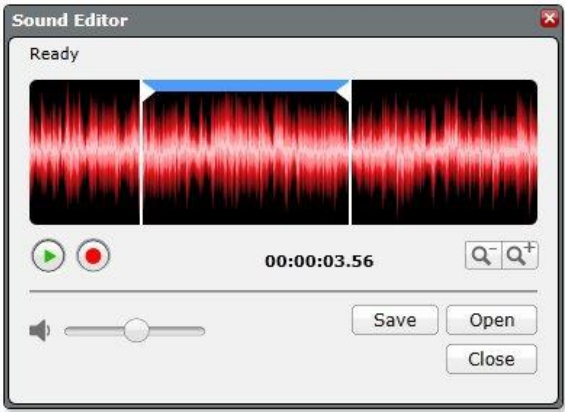
Az EV3-as tégla rendelkezik egy egyszerű beépített hangszóróval, ami dallamok, hangok lejátszására alkalmas. A modul a *Action* csoportban található *Sound*.

Beillesztve a megfelelő programhelyre, három működési mód közül választhatunk. Lehet előre elkészített hangfájlokat lejátszani (*Play File*), vagy adott frekvenciájú hangot (*Play Tone*), esetleg alap zenei hangokat (*Play Note*) megszólaltatni.
A három esetben más-más lesz a paraméterezési mód.
A hangfájlok esetén lejátszandó fájl nevét a blokk jobb felső sarkában lehet megadni, ekkor a szoftverrel kapott néhány zenei effekt mappákba rendezetten megjelenik és választhatunk.

A zenei hangok lejátszásánál egy zongora billentyűzetéről választhatunk hangot, míg a frekvenciás megadás esetén beírhatjuk a szövegdobozba a hang frekvenciáját (Hz). A rendszer felkínálja az alaphangok frekvencia táblázatát, így abból is választhatunk. (A zenei skála nem lineáris, hanem logaritmikus.)

Mindhárom beállítás esetén megadhatjuk a megszólalás hangerősségét, valamint egy listából választható értékkel, hogy egyszer játssza le a rendszer a hangot vagy folyamatosan, illetve a lejátszás ideje alatt más modul ne induljon el. A hangfájlok lejátszását kivéve a hang megszólalásának időtartama is beállítható.

Hangfájlok saját szerkesztésére is van lehetőség. Ehhez rendelkezésre áll egy egyszerű hangszerkesztő felület, ahol *wav* vagy *mp3* formátumú hangfájlokat tudunk szerkeszteni, majd a lejátszásra alkalmas *rso* kiterjesztéssel a megfelelő formátumban menteni.



A szerkesztőpanel a *Tools* menü *Sound Editor...* menüpontján keresztül érhető el.

A fájl 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 ikonon kattintva a kijelölt rész le is játszható. A mentést a *Save* gombon történő kattintással végezhetjük, el a név megadása után. Alapértelmezésben a program telepítési mappájába történik a mentés.

14.2. Fájlkezelés

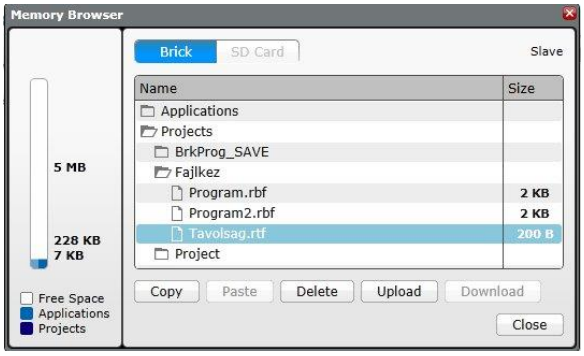
Nagy mennyiségű mérési adat tárolására nem elegendő az a változómennyiség, amit létre tudunk hozni a keretprogramban. Ezen kívül a változókban tárolt adatok csak a memóriában léteznek, így azok programon kívüli felhasználására (esetleg más programokkal történő feldolgozására) nincs lehetőség. A különböző programnyelvekben a fájlkezelés oldja meg ezt a problémát. A mérési adatainkat tehát fájlokban tárolhatjuk. Ezek általában szövegfájlok, amelyek bármely szövegszerkesztő vagy textfájlokat megnyitni képes programmal olvashatók. Mivel a *txt* fájlokban tárolt adatokat nagyon sok keretprogram képes kezelni (táblázatkezelő, adatbázis-kezelő programok), így további feldolgozásukra számos mód nyílik, ugyanakkor ezek a fájlok platformfüggetlenek, tehát nem csak a *windows* rendszerek képesek kezelni őket. A textfájlokban tárolt adatok specialitása, hogy a számok is szövegként tárolódnak. A fájlokból az adatokat vissza is tudjuk olvasni a memóriába, a szöveges tárolású számjegyek számmá konvertálását a keretprogram a beolvasás során automatikusan elvégzi.
Mivel a robotra csatlakoztatott szenzorok adatai egyszerű mérésekre is alkalmasak, így a nyelv és a rendelkezésre álló keretprogram támogatja a mérési adatok fájlban tárolását. Minderre kétféle lehetőség adódik. Az egyik módszer a hagyományos struktúrának megfelelő fájlkezelés, míg a másik a robotra jellemző úgynevezett *datalog* fájlok létrehozása. Ez utóbbiról korábban már volt szó.
Az első esetben a megírt programkódban a megfelelő modulok beillesztésével adhatjuk meg a tárolás egyes lépéseit, míg a második esetben programtól függetlenül, állíthatjuk be a kiválasztott szenzorok figyelését és a mért adatok tárolását.

A hagyományos fájlkezelés programvezérlő modulja az *Advanced* ikoncsoport *File Access* ikonjának beillesztésével érhető el.

A legördülő listán szereplő fájlkezelési módok a más programnyelvekben megszokott funkciókat tartalmazzák. Egy fájlt olvasásra vagy írásra lehet megnyitni. Az első esetben a fájlnak léteznie kell, a második esetben a rendszer létrehozza, ha még nem létezik. Ha egy létező fájlba írunk, akkor adataink a fájl végére kerülnek (hozzáfűzés).
A törlés funkció törli a fájlt, míg a bezárás funkciónál nem törlődik az állomány. Ha egy nyitott fájlt a használat végén nem zárunk be, akkor megsérülhetnek a benne tárolt információk.
A fájl nevét az ikon jobb felső sarkában lévő szövegdobozba írva kell megadni (jelenleg: *„abc.rtf”*).

A programozásban a fájlkezelés több lépésből áll. A használat előtt a fájlt meg kell nyitni, létre kell hozni (ez a művelet különböző programnyelvekben két lépésből is állhat, az első lépés lehet egy logikai név, valamint a fájl használati módjára utaló kód megadása, majd a második lépés a tényleges megnyitás). A megnyitás után végezhetjük el a manipulációkat az adatokkal, pl. fájlba írás, olvasás. Majd a használat végén a fájlt le kell zárni.
Az EV3-G programnyelvben az ikon beillesztésével a rendszer automatikusan megnyitja a fájlt a kiválasztott műveletnek megfelelően (*Write*-írás vagy *Read*-olvasás).

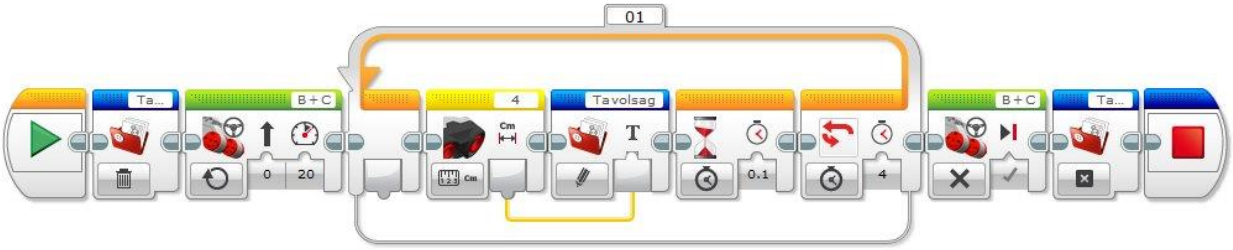
A fájl a létrehozása és lezárása után a robot flash memóriájában jön létre, a megadott néven. Innen tudjuk manuálisan átmásolni tetszőleges mappába a *Upload* gombon kattintva. A már számítógépen létező szövegfájlunkat a robot memóriájába töltve (*Download*) elérhetővé válik programjaink számára. A fájl kiterjesztése *rtf*.



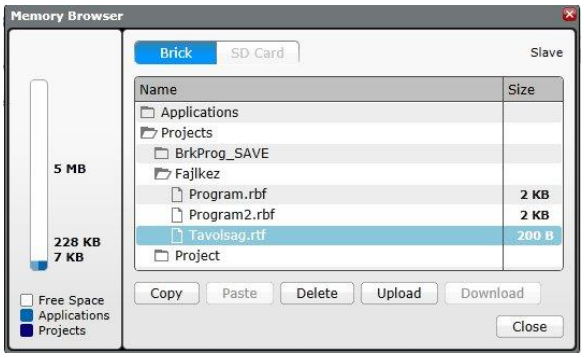
Egy egyszerű használati módot mutat be a következő program.

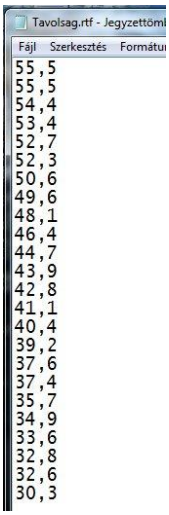
14/P1. Írjon programot, amelyet végrehajtva a robot állandó sebességgel halad előre egy akadály felé! Az ultrahang szenzorával mért értékeket 0,1 másodpercenként rögzíti egy fájlban. Az adatok rögzítését a robot 4 mp-ig végezze!

A 4 másodperc alatt a robot 40 mérési adatot fog rögzíteni. Ha a 0,1 másodperces mintavételi sebességet nem adjuk meg, akkor két ezred másodpercenként mér az ultrahangszenzor, ami most fölöslegesen sok adatot eredményez. Az adatokat a *Tavolsag.rtf* fájlba rögzíti, ami a robot memóriájába kerül.
Első lépésként töröljük a fájlt, hogyha lett volna ilyen néven már állományunk, akkor ne zavarjanak a benne lévő adatok. A motorok bekapcsolása után a 4 másodpercig futó ciklusban a fájlba írás történik meg (*Write* mód). A mért adatok szám típusúak, de a fájlban szövegként rögzíti őket a rendszer (automatikus az átalakítás). A ciklus után a robotot leállítjuk és bezárjuk a fájlt.

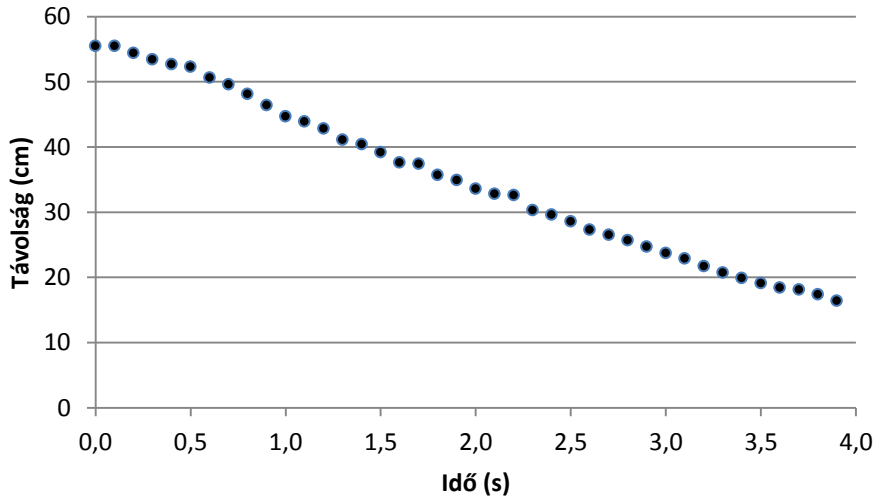


A program befejezését követően a memóriából az állomány áttölthető a számítógépre, ahol tetszőleges szövegszerkesztő programmal olvasható a tartalma, vagy akár táblázatkezelőbe importálhatók a mérési adatok.
A táblázatkezelőbe történő importnál arra kell ügyelni, hogy az operációs rendszer beállításaitól függően a tizedes határoló ugyanaz legyen a mérés rögzítésekor és az importálásnál (a pont és vessző, mint tizedes határoló eltérése azt eredményezheti, hogy az adatainkat dátumként vagy szöveges adatként veszi át a táblázatkezelő).
Fájl áttöltése a számítógépre és tartalma a Jegyzettömb szövegszerkesztőben:

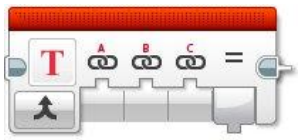




Minden adat új sorba (bekezdésbe) került. Az adatokon jól megfigyelhető, ahogy a robot közeledett az akadály felé és egyre kisebb távolságokat mért a szenzora. Ha az adatokat importáljuk egy táblázatkezelő programba, és ott egy grafikont illesztünk rájuk, akkor mindez vizuálisan még jobban nyomon követhető.



A pontsorra illesztett egyenes meredeksége a robot sebességét adja meg cm/s mértékegységben. A programban beállított érték 30-as volt.

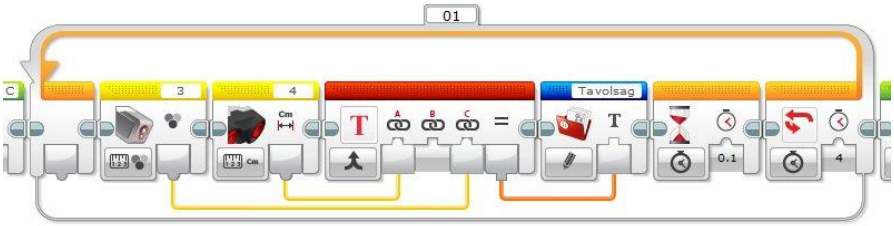


Ha több szenzor mérési adatait egyszerre szeretnénk fájlba rögzíteni, akkor használhatjuk a *Data Operations* csoport *Text* blokkját.

Segítségével három bemenő paraméter adatait tudjuk összefűzni. A bemenet, a jelölése alapján szöveg típusú kell, hogy legyen, de szám típust is használhatunk. Ilyenkor a rendszer automatikusan átalakítja a tárolási formát.

Az előző példa kiegészítéseként, ha a fényszenzor által mért adatokat is szeretnénk a fájlba rögzíteni a távolságadatokkal együtt, akkor a két mérési adat elválasztására például a szóközt használhatjuk. (Ha nem adunk meg elválasztó karaktert, akkor a számok közvetlenül egymás után íródnak és nem tudjuk szétválasztani őket.) A *Text* modullal össze tudjuk fűzni a karaktersorozatokat egyetlen szövegláncca: 'távolság érték' + 'szóköz' + 'fényszenzor érték'. Így táblázatkezelőbe importálva az adatsorokat,

beállítható, hogy külön oszlopba kerüljenek a számok, a szenzoroknak megfelelően. Az előző példa ciklusa kiegészítve:



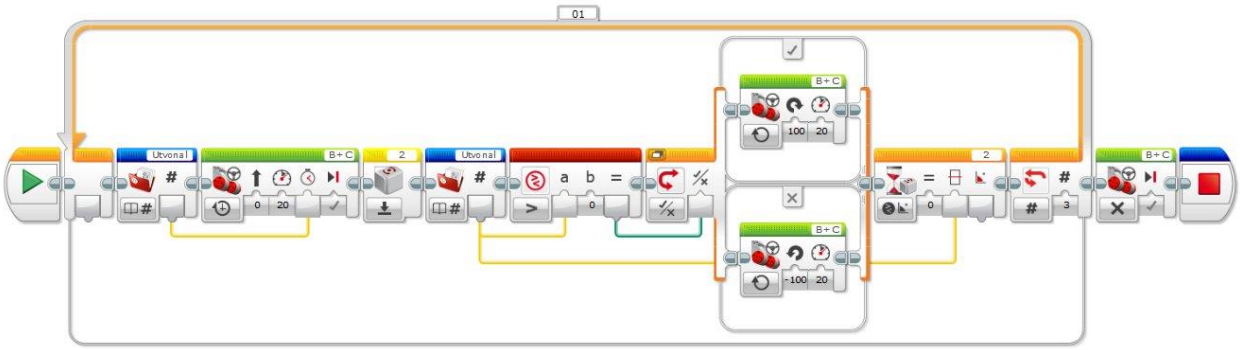
A fájlból történő olvasás alkalmas arra, hogy a robot mozgását vezéreljük. Első példaként hozzunk létre egyszerű szövegszerkesztő programmal egy szövegfájlt, amelybe manuálisan számokat írunk, majd ezeket beolvasva használjuk fel a robot motorjainak irányítására.

14/P2. Írjon programot, amelyet végrehajtva a robot egy fájlban tárolt adatsor alapján végzi a mozgását!

A szövegfájlba a robot egyenes vonalú mozgási idejére és fordulási szögére vonatkozó számokat írunk. Minden adat külön sorba kerül. A fájlt manuálisan hozzuk létre. Az adatok beolvasása után az első adattal a motor mozgási idejét paraméterezzük, míg a második adat a giroszenzor által mért fordulási szöget határozza meg. Így a mozgást két-két adat írja le, abban az értelemben, hogy minden egyenes vonalú mozgást egy fordulás követ. Az *Utvonal.rtf* nevű fájlba a következő adatokat írtuk:

0.5
90
1
-60
1.5
30

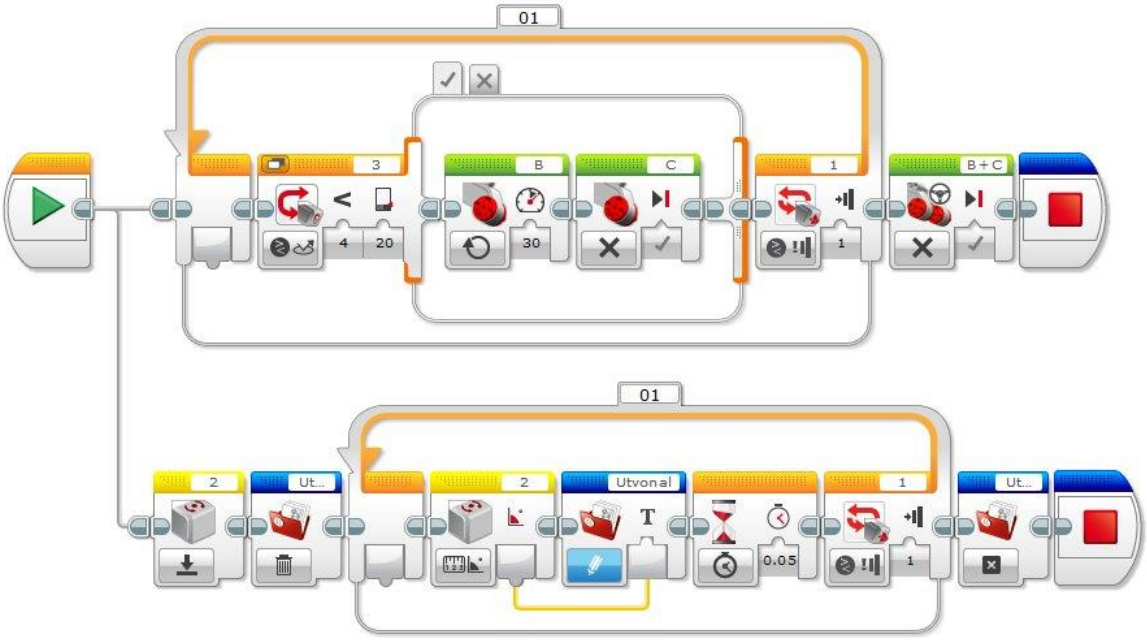
A mozgás ezek szerint: egyenesen előre 0,5 másodpercig, majd fordulás jobbra 90°-ot, egyenesen előre 1 másodpercig, fordulás balra 60°-ot, egyenesen előre 1,5 másodpercig és fordulás jobbra 30°-ot.



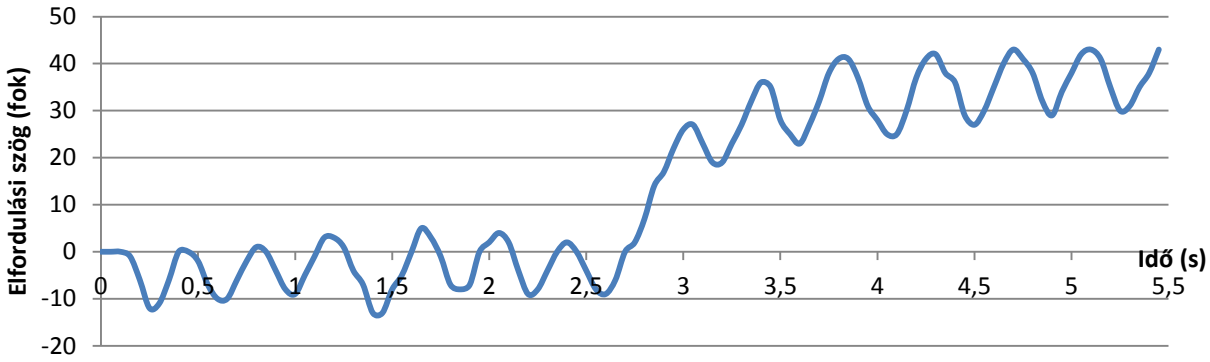
Mivel három pár adatunk van, ezért a ciklus 3-szor fut le. A cikluson belül az először kiolvasott adatot a motorok működési idejét meghatározó paraméter helyére kötjük be. A motorok sebessége állandó (20). A giroszenzor nullázása után (*Reset* mód), el kell dönteni, hogy melyik irányba kell a robotnak fordulnia. Ezt a fájlba írt fordulási szög adatok előjele dönti el. Ha a szög negatív, akkor balra, egyébként jobbra. Ezt vizsgálja a feltételes elágazás blokk, és ennek megfelelően indítja el a motorok forgásirányát, míg a szöget a *Wait* modul *Gyro Sensor* beállítása vizsgája Összehasonlító (*Compare*) módban.

14/P3. Írjon programot, amelyet végrehajtva a robot egy fekete színű útvonalat követ egyetlen fényszenzorával. A mozgás során a giroszenzorral mért adatokat rögzítse fájlba! Megfigyelhető-e a mérési adatokon a robot „kígyózó" mozgása?

A program két szálon fut. A felső szál a hagyományos egyszenzoros útvonalkövetés (lásd korábban). Az alsó szál a fájlkezelés rész. A giroszenzor alaphelyzetbe állítása (*Reset*), és a fájl biztonsági törlése után a ciklus 0,05 másodpercenként az *Utvonal.rtf* fájlba rögzíti a giroszenzor pillanatnyi mérési értékét fok mértékegységben.

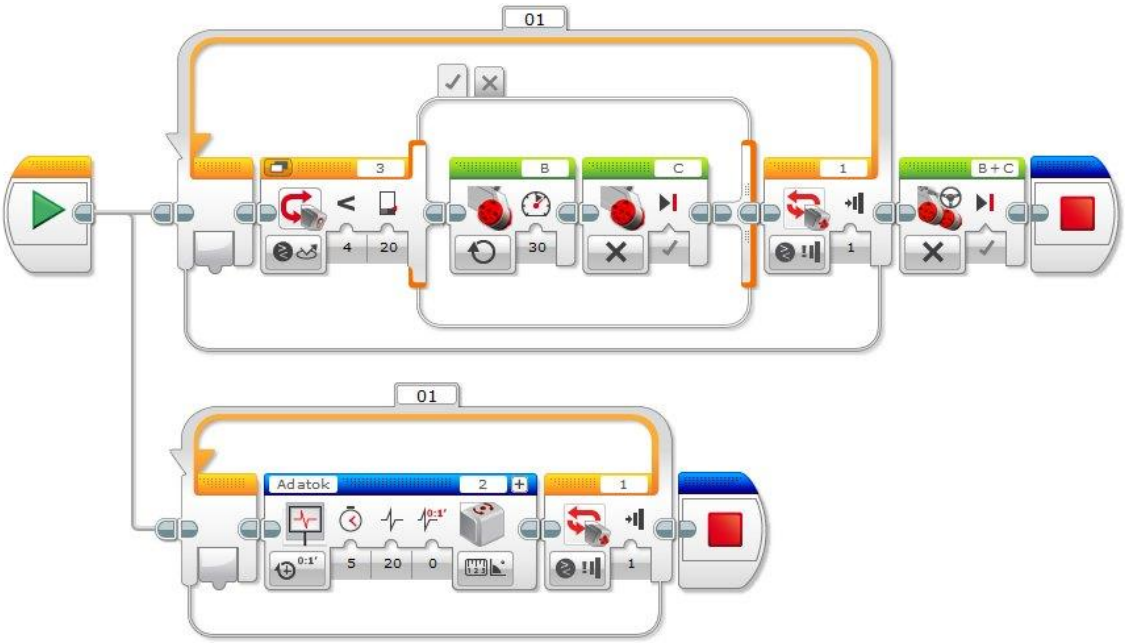


A programot ütközésérzékelővel leállítva és a létrejött fájl adatait táblázatkezelő programba importálva a grafikonon jól megfigyelhető a robot kígyózó mozgásából adódó elfordulási szög változása.

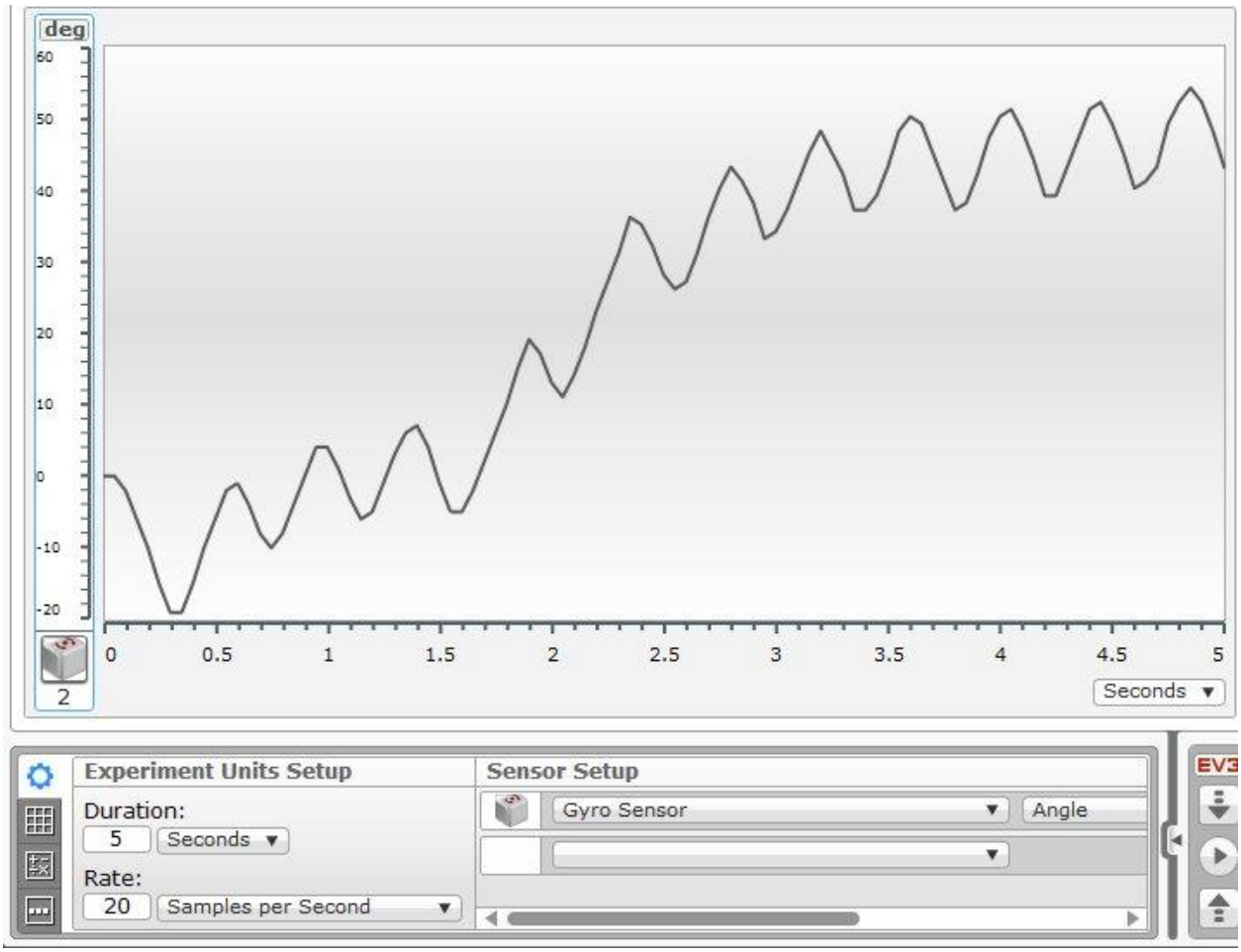


Több útvonalkövetésre vonatkozó következtetés is levonható. Egyrészt a robot az út jobb oldalát követte. Ez abból látható, hogy negatív irányú elfordulással kezdett (balra fordult), ez az óramutató járásával ellentétes. Másrészt kb. 2,5 másodpercnyi egyenes vonalú út után egy jobbkanyar volt az úton, majd utána ismét egyenesen folytatódott.
A programot kicsit átalakítva *Data Logging* modul segítségével rögzítettük fájlba (*Adatok.rdf*), és importáltuk be a szoftver *Experiment* felületére.

A módosított program:



A grafikus kép a mérési adatokról:



A fenti következtetések ismét levonhatók, a mérési adatokra illesztett grafikon alapján.

14.3. Gyakorló feladatok

14/F1. Egyszerű szövegszerkesztő programmal készítsen egy szövegfájlt (*rtf* kiterjesztés), amelybe írjon egy rövid szöveget! Töltse fel a fájlt a robot memóriájába, majd írassa ki a szöveget a robot képernyőjére! (A szöveg ne tartalmazzon ékezetes karaktert!)

14/F2. Írjon programot, amelyet végrehajtva a robot lassan halad előre, és fényszenzora által mért értékeket egy fájlban tárolja! 0,1 másodpercenként rögzítse a fájlban a fényszenzor által mért értéket! A mérés végeztével (kb. 5 mp) a fájlt töltse át a számítógépre! Készítsen egy táblázatkezelő program segítségével grafikont a mért adatokkal! Pl.: Készítsen oszlopdiagramot, amelynél az oszlop magassága arányos a mért értékkel.

14/F3. Írjon programot, amelyet végrehajtva a robot a hangszenzora által mért értékeket egy szövegfájlban tárolja! A mérés végeztével (kb. 15 mp) a fájlt töltse át a számítógépre! Készítsen egy táblázatkezelő program segítségével grafikont a mért adatokkal! Pl.: Egy zenelejátszó eszközön megszólaltatott zeneszám 15 másodperces részletén mért adatokat rögzítse!

14/F4. Írjon programot, amelyet végrehajtva a robot 1 és 90 közötti véletlen számokat sorsol! Minden ötödik szám kisorsolása után írja a fájlba a 0 értéket! Sorsoljon tíz számötöst!

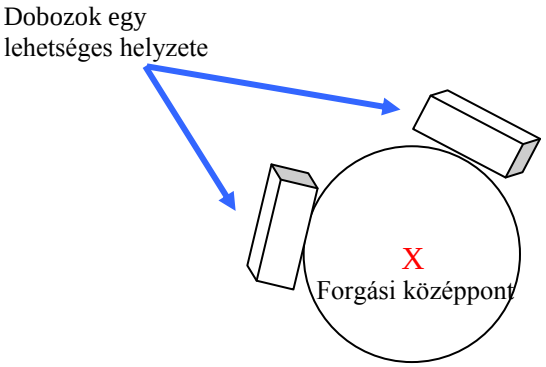
14/F5. Írjon programot, amelyet végrehajtva a robot egy enyhe lejtőn gurul lefelé egy akadály felé! A motor kerekei szabadon forognak, nincsenek a motorhoz csatlakoztatva. A robot ultrahagszenzorával mért értékeket, az akadálytól való távolságot, 0,05 másodpercenként rögzítse egy fájlba! A mérés végeztével a fájlt töltse át a számítógépre, és táblázatkezelő program segítségével készítsen a mért adatokkal grafikont!

14/F6. Mikrofon segítségével rögzítse 1-től 5-ig a számok kiejtett hangalakját! A rögzített hangfájlokat szerkessze, és mentse *rso* formátumban, a rendelkezésre álló szerkesztő program segítségével! Írjon programot, amelyet végrehajtva a robot 1 és 5 közötti véletlen számot sorsol és lejátssza a számhoz tartozó hangfájlt!

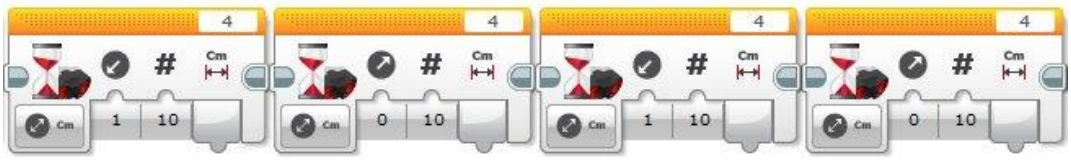
14/F7. Egy ismert könnyűzenei szám 5 másodperces részletét szerkessze és mentse *rso* formátumban, a rendelkezésre álló szerkesztő program segítségével! Írjon programot, amelyet végrehajtva a robot tolatva halad, amíg ütközésérzékelőjét nyomás nem éri. Ekkor játssza le a rögzített zenerészletet!

15. VERSENYFELADATOK ÉS MEGOLDÁSAIK

15/P1. Írjon programot, amelyet a robot végrehajtva egyenletes sebességgel helyben forog! A robot körül két doboz van elhelyezve úgy, hogy a forgási középponttól legfeljebb 25 cm sugarú körvonalra leghosszabb oldalukkal érintőlegesen helyezkednek el (lásd ábra). A robot feladata, hogy a forgásirányát figyelembe véve a második dobozt megtalálja és fekete csíkig tolja. A robot indulási pozíciója olyan, hogy nem dobozzal szemben áll.



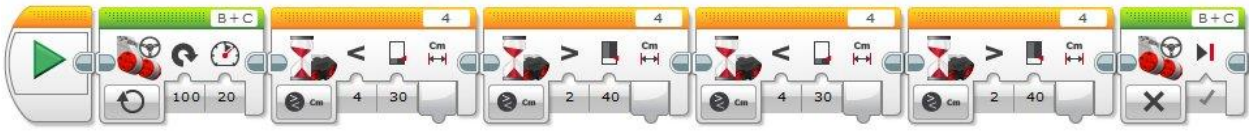
A program alapötlete, hogy az ultrahangszenzort figyelve a forgás során először nem érzékel a robot 25 cm-en belül akadályt, majd ha a doboz a látóterébe kerül, akkor a szenzora 25 cm-nél kisebb távolságot mér. Ha a doboz kikerül a látóteréből, akkor ismét nagyobb értéket kapunk vissza, mint 25 cm. Tehát az ultrahangszenzor változása alapján egy doboz esetén először csökken pl. 10 cm-t a mért érték, azután pedig nő 10 cm-t. Ebben az esetben az ultrahangszenzor Változás (*Change*) működési módját használjuk. Mivel két dobozt kell érzékelni, ezért a blokkpár kétszer szerepel.



Ugyanezt az eredményt érhetjük el akkor is, ha doboz észlelésére az ultrahangszenzor Összehasonlító (*Compare*) módját választjuk pl. 30 cm-nél kisebb észlelést választva a doboz érzékeléséhez és 40 cm-nél nagyobbat a forgás soráni elhagyásához. Szintén dupla blokkpárt használva.

Mivel a szenzoron kívül semmire sem kell figyelnie a robotnak, ezért használhatjuk a *Wait* blokkot, így csak abban az esetben lép túl az adott blokkal meghatározott parancson a program, ha beállított feltétel teljesül.

A blokksor előtt be kell kapcsolnunk a motorokat forgásra állítva őket és a második doboz után le kell állítanunk a forgást. A forgás sebessége ne legyen túl nagy, mert az ultrahangszenzor észlelési sebessége miatt esetleg nem lesz pontos a mérés. A példánál a sebességet 20-ra állítottuk.



A második doboz elhagyása után a robotot egy kicsit vissza kell fordítanunk (a példánál ez az érték 0,4 másodperces visszafelé irányú forgás), hogy szemben álljon a dobozzal, és bekapcsolva motorokat egyenes haladást beállítva a fényszenzor által érzékelt fekete vonalig mozgatni. A fekete vonal elérését akár előzetes méréssel meghatározott fényintenzitás beállításával, akár *Change* üzemmódban a változást figyelve detektálhatjuk (a példánál ez a változás 10). A vonal elérését követően a motorokat leállítva a feladatot megoldottuk.

A visszafordítás mértékét a dobozok mérete határozza meg. A cél, hogy kb. szembe kerüljön a robot a dobozzal. Ezt számolással is meghatározhatjuk, ha eltároljuk a doboz két szélének érzékelését jelző időpontokat egy-egy változóban, és ezek különbségének fele lehet a visszafordítás időtartama. A számítás ugyan nem lesz pontos, mert az észlelésnél a motorok mozgásban voltak, míg a visszafordításnál fel kell őket gyorsítani, így a gyorsulásra szánt időt is figyelembe kellene venni. Ha a mérést nem az időpontokat mérni tudó stopperrel (*Timer*) végezzük, hanem pl. az motor elfordulási szögértékeit tároljuk, akkor a visszafordítás pontosabb lehet.

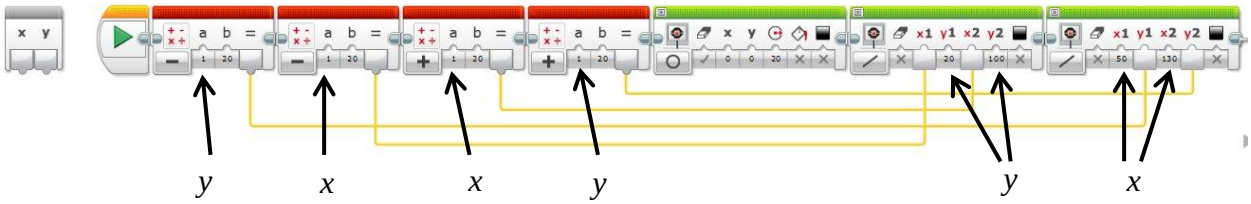
15/P2. Írjon programot, amelyet a robot végrehajtva a képernyőjén egyenletes sebességgel mozgat balról-jobbra egy alakzatot a képernyő középvonalán. Az alakzat elérve a képernyő jobb szélét ismét a bal szélen bukkanjon fel és induljon jobbra. Ezt a vízszintes mozgást addig ismételje, amíg az ütközésérzékelőjét nyomás nem éri. Ekkor a vízszintes mozgás helyett az aktuális pozícióból indulva függőlegesen mozogjon az alakzat tovább, lentről-felfelé. A képernyő tetejét elérve alul bukkanjon föl újra. Az ütközésérzékelő ismételt benyomására újra vízszintes mozgást kezdjen, az aktuális pozíciótól, balról-jobbra irányban. Mindezt kikapcsolásig ismételje. Tehát az ütközésérzékelő minden egyes benyomásának hatására a vízszintes mozgás függőleges mozgásra váltson és fordítva.

Az alakzat:

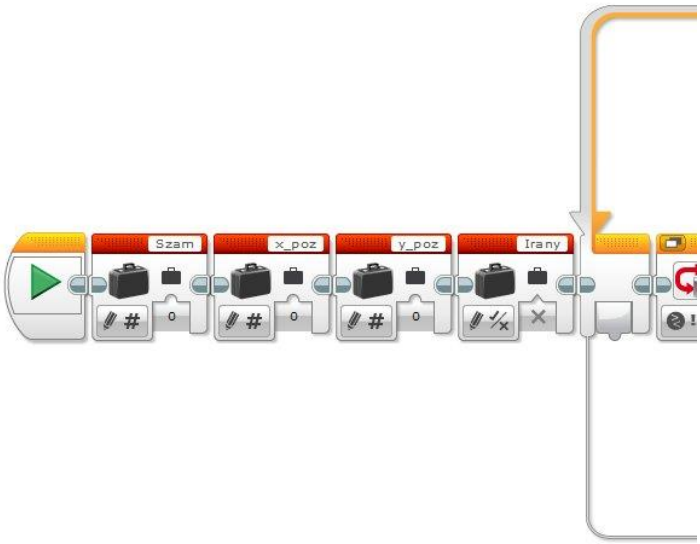
A kör sugara 20 pixel.

Első lépésben készítsük el az alakzat rajzát. Mivel sokszor kell majd használnunk ezért érdemes külön blokkban elkészíteni. A rajzhoz egy kört és két szakaszt kell rajzolni. A blokk paraméterként a kör középpontját kapja meg. A kör sugarát fix 20 pixelnek választjuk. (Ezt is lehetne paraméterrel szabályozni, de a feladat megoldása szempontjából nincs rá szükség.) A saját blokk nem ad vissza értéket.

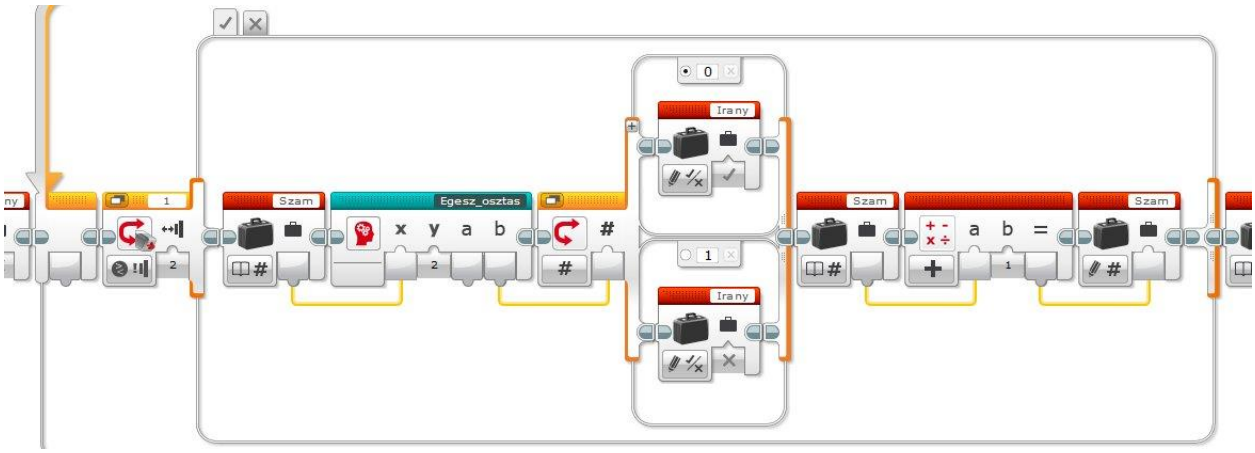
A szakaszokat úgy rajzoljuk fel, hogy a megkapott körközéppontból levonunk 20-at, illetve hozzáadunk 20-at a kezdő és végpont koordinátáinak számításához. A ±20 műveletet mindkét koordinátával el kell végezni. Így ha a kör középpontjának koordinátái: (x;y), akkor a vízszintes szakasz: (x–20;y) → (x+20;y) és a függőleges szakasz: (x;y–20) → (x;y+20) végpontokkal jellemezhető.



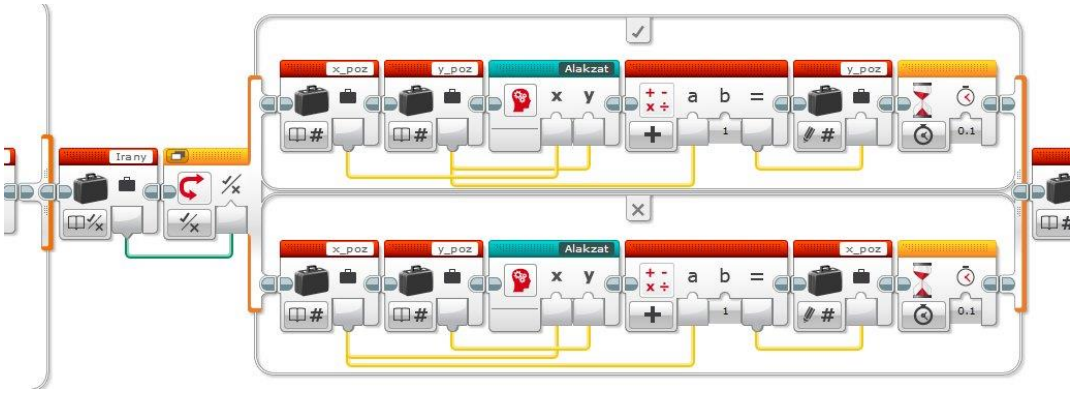
Az áttekinthetőség kedvéért a blokkon belül nem kötöttük be a bemeneti paramétereket, de valamennyi esetben jelöltük a bekötés helyét.
A blokk alkalmas arra, hogy a megkapott koordinátákkal, mint középponttal felrajzolja a feladatban kért alakzatot.
A megoldás további algoritmusát több részletben mutatjuk be.
A választott megoldási ötlethez négy változót hozunk létre. Az *x_poz* és *y_poz* változóban tároljuk az alakzat középpontjának aktuális koordinátáit. A *Szam* változóban számláljuk az ütközésérzékelő benyomásainak számát. Ez fogja a mozgás irányát meghatározni. Minden benyomás után eggyel növeljük az értékét. Mivel váltakozva kell vízszintes vagy függőleges irányban mozgatni, így a kettővel történő osztási maradékot használjuk az irány megváltoztatására. Egy folyamatosan növekvő számsor esetén a 2-es maradék váltakozva 0 vagy 1.
Az *Irany* logikai változó értéke attól függően lesz hamis vagy igaz, hogy a *Szam* változó 2-es maradéka 0 vagy 1. Ezt az egyszerűbb feltételes elágazás vezérlés miatt vezettük be.
A változók létrehozása és a kezdőértékek beállítása után egy végtelen ciklust indítunk el.



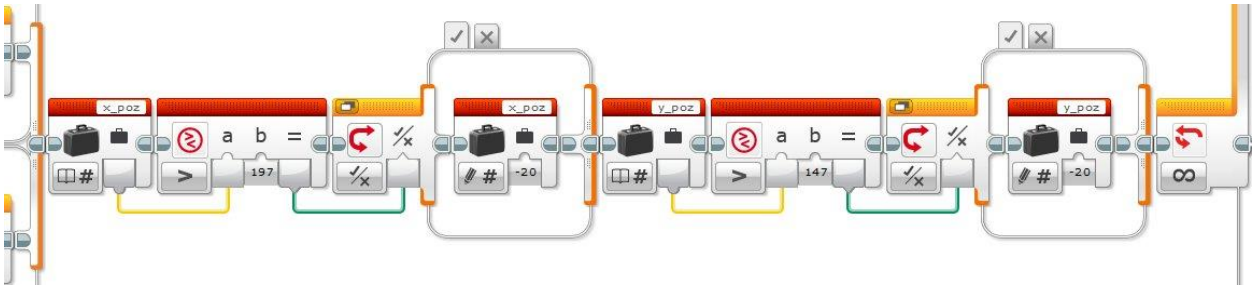
A cikluson belül állítjuk be egy számvezérelt elágazásban a logikai változó értékét. Ebbe az elágazásba csak akkor jut be a program, ha benyomtuk az ütközésérzékelőt (*Bumped*). A feltételes elágazáson belül növeljük eggyel a *Szam* változó értékét is, hiszen a következő ütközésérzékelő benyomásra már eggyel nagyobb értéknek kell venni a 2-es maradékát (így kapunk váltakozó 0, 1, 0, 1, 0, 1, … maradéksorozatot).



Az elágazásban szereplő utasítások felelősek az irány váltakoztatásáért.
A következő programrészlet szabályozza az elkészített alakzat képernyőre rajzolását. Az előzőekben létrehozott *Irany* változó értékétől függően (igaz/hamis) vagy az *x_poz*, vagy az *y_poz* változó értékét növeljük eggyel és a saját alakzatrajzoló blokkba ezeket a koordinátákat kötjük be.



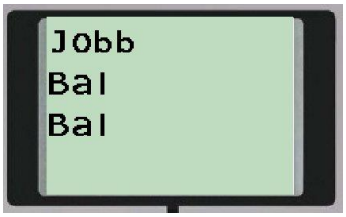
A rajzolás mindig abba az irányba történik, amely változó értékét növeljük. Mivel mindig csak az egyik változó értéke nő, a másik állandó, így az alakzat vízszintesen vagy függőlegesen mozog.
A program már működik, de a képernyő szélét még nem kezeltük. Tehát, ha az alakzat elérte a képernyő szélét, akkor a másik oldalról kell ismét felbukkannia. Ezt két feltételvizsgálattal érjük el. Figyeljük az *x_poz* illetve *y_poz* változó értékét, és ha képernyő szélének megfelelő koordinátájánál 20-sszal nagyobb lesz az értéke (20 a kör sugara), akkor a változó értékét –20-ra állítjuk vissza.



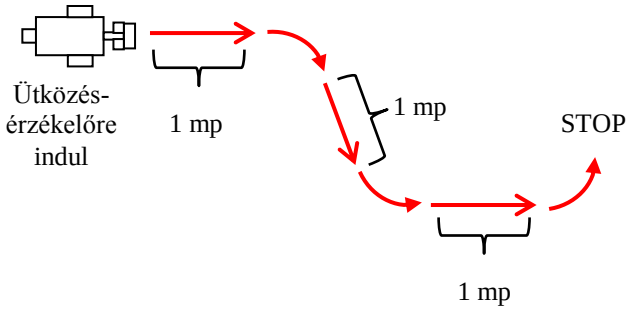
15/F3. Írjon programot, amelyet végrehajtva a roboton lévő „bal” ( ) illetve „jobb” ( ) gombok segítségével lehet beállítani a mozgását, a következő leírásnak megfelelően. A bal gomb megnyomására megjelenik a képernyőn a bal szó, míg a jobb gomb megnyomására a jobb szó. Mindezt háromszor kell ismételni, tehát három egymás alatti sorban látszik a képernyőn a bal vagy jobb szavak valamilyen variációja (lásd ábra). Ezután ütközésérzékelőre elindul a robot és 1 másodpercig halad egyenesen előre, majd 2000-os tengelyfordulattal fordul balra vagy jobbra a beállított első értéknek megfelelően. Ezután

újra egyenesen halad 1 másodpercig, majd újra fordul 200°-os tengelyfordulatot a második beállított értéknek megfelelően. Ugyanezt a mozgást elvégzi a harmadik beállított értékre is.

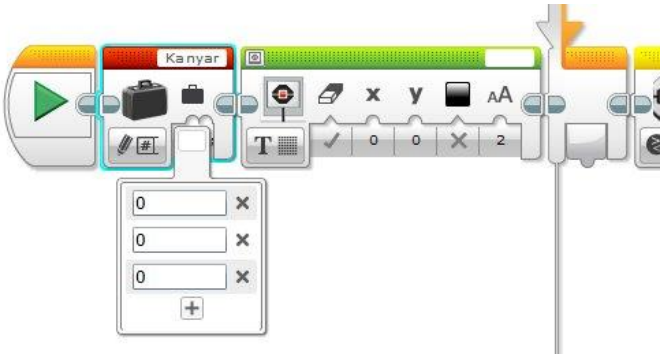
Beállított értékek a képernyőn:



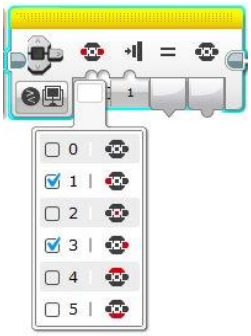
A robot mozgása:



A programban a tégla nyomógombjai segítségével kell három értéket tárolni (az értékek száma tetszőlegesen növelhető). Ezeket az értékeket kétféleképpen fogjuk használni. Egyrészt a képernyőre kell írni a „Bal" illetve „Jobb" szavakat, másrészt a robot mozgásánál a kanyarodás irányát is ezek határozzák meg. Az értékeket változóban tároljuk, célszerűen egy szám típusú tömböt hozunk létre (*Kanyar*). Kezdésként a három elemű tömböt kinullázzuk (feltöltjük az elemit 0-kal). A *Display* modul a képernyőtörlést végzi (*Text* üzemmódban üres karaktersorozatot írva).

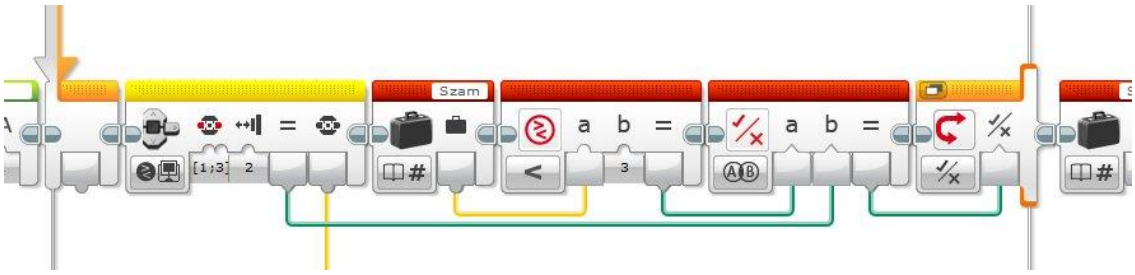


Mivel csak három értéket szeretnénk a tömbben tárolni, ezért valahogy ki kell védeni azt az esetet, hogy a felhasználó többször nyomja meg a tégla gombjait (csak az első három gombnyomás értékét tároljuk). A gombnyomások számlálására hoztuk létre a *Szam* változót, amely a tárolást megvalósító tömb indexe lesz. A szenzor csoportban szereplő *Brick Buttons* modulnál a *Compare* módot használva beállíthatjuk, hogy a tégla melyik gombjainak benyomását figyelje a program.



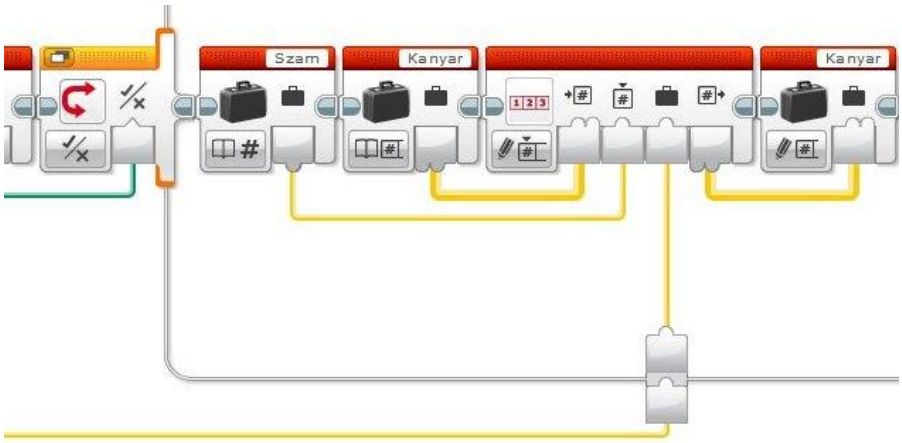
Az 1-es és 3-as számérték jelöli a balra illetve jobbra nyomógombot. A blokk akkor ad vissza igaz értéket, ha a két lehetőség közül valamelyik teljesül (vagylagos kapcsolat). A benyomott gombazonosító számértékét szintén visszaadja a blokk a megfelelő kimeneti csatlakozási ponton.

A gombnyomások tárolását tehát egy összetett feltétel szabályozza. Csak az első három gombnyomás esetén (*Szam* változó értéke kisebb, mint 3 → 0; 1; 2), és a téglán az 1-es vagy 3-as azonosítójú gombot nyomták be (és kapcsolat a két feltétel között), akkor eltároljuk a tömbben a benyomott gomb azonosítóját.



Nyomógomb azonosítója

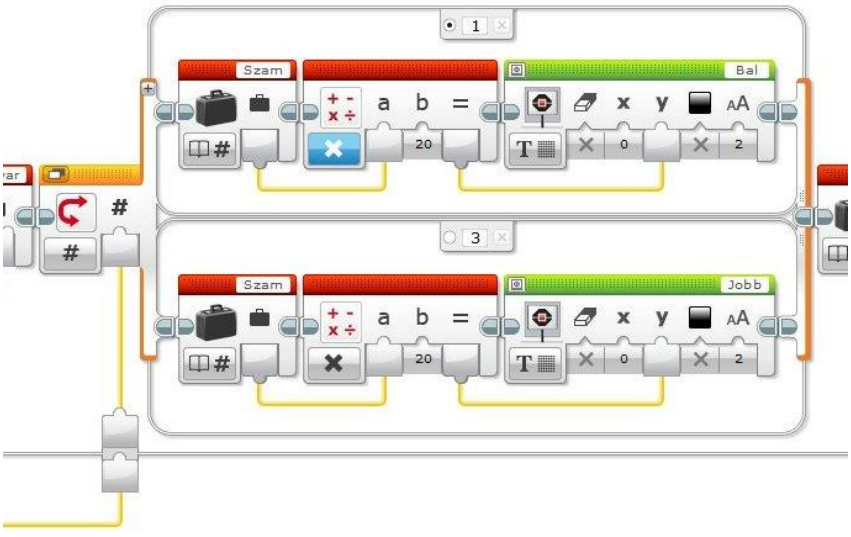
A tárolást egy feltételes elágazás végzi, amely hamis ágán nem szerepel utasítás (nem jelenítettük meg). A *Kanyar* nevű tömbben a nyomógomb azonosítóját tároljuk a *Szam* változó által meghatározott helyen (ez kezdetben 0, ez lesz a tömb indexe).



Nyomógomb azonosítója

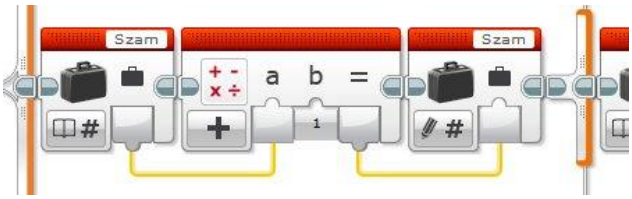
A képernyőre a gombnyomást tényét szövegesen is ki kell írni. Ha ezt a gombnyomás megtörténte után rögtön meg szeretnénk megtenni, akkor egy feltételes elágazást kell használnunk a két értéknek megfelelően. Ha egy számvezérelt elágazást használunk, akkor a tégla által visszaadott nyomógomb azonosító (1 vagy 3), alkalmas erre. Az elágazás két ágán szereplő utasítások tehát az 1 illetve 3 megkapott érték alapján futnak le.

A kiírás képernyőn megjelenő pozíciójának helyét is vezérelhetjük a *Szam* változó tartalmával. Mivel ez 0, 1, vagy 2, ezért ha megszorozzuk 20-szal, és ezt használjuk az *y* képernyő koordináta megadására, akkor egymás alatti sorokban jelennek meg a „Bal" illetve „Jobb" szavak, 20 pixeles eltéréssel. Az *x* koordináta mindhárom esetben 0.

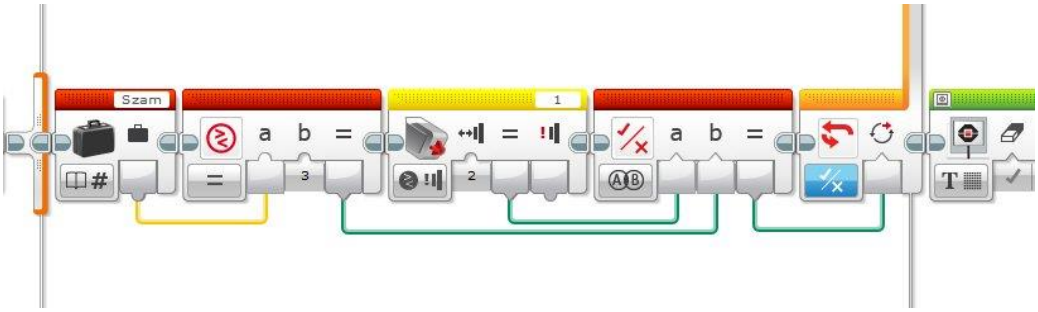


Nyomógomb azonosítója

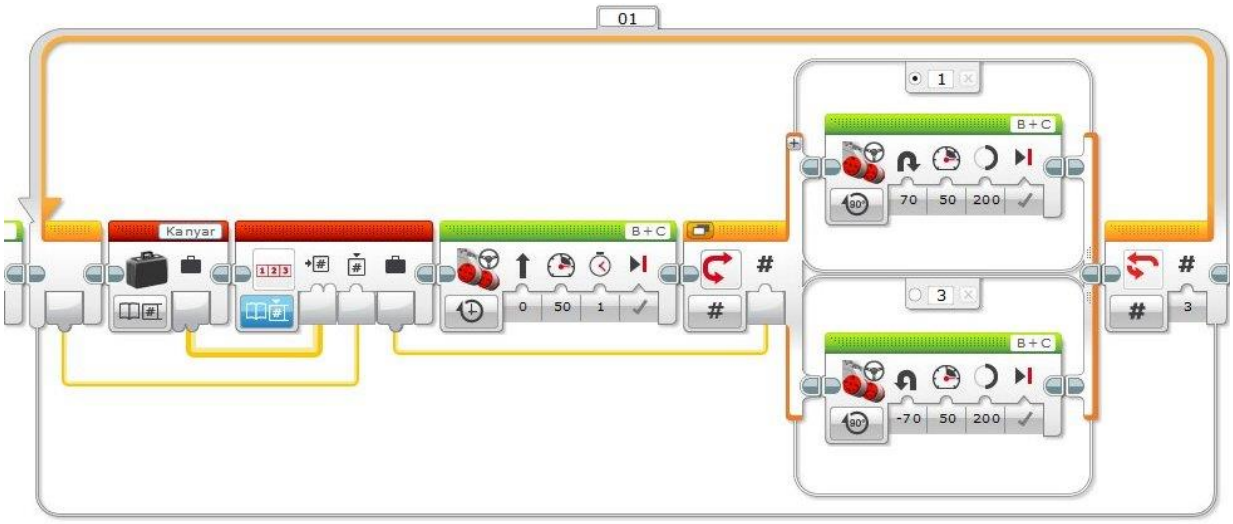
Mielőtt véget érne az adattárolási és kiíratási ciklusunk a *Szam* változó értékét meg kell növelni 1-gyel, hogy a következő gombnyomás, kiíratás esetén már új tömbelembe történjen a mentés, illetve a következő sorba a kiíratás.



A feladat leírása alapján a robot a mozgást ütközésérzékelő megnyomására kezdi meg. A ciklusnak tehát abban az esetben kell véget érnie, ha megtörtént a három gombnyomás (*Szam* változó értéke 3, és megtörtént az ütközésérzékelő megnyomása is (*Bumped*).



Ezután elindul a robot mozgása. A Kanyar tömbben tárolt értékeknek megfelelően balra vagy jobbra kell fordulnia a robotnak, egy egyenes mozgást követően.
Mivel három értéket tároltunk a tömbben, így az egyszerűség kedvéért egy háromszor lefutó ciklust használunk, amely tartalmazza az egyenes mozgást szabályozó blokkot (*Steering Motor*, 1 mp.), valamint egy szám vezérelt elágazást, amely a jobbra vagy balra kanyarodást irányítja. Az elágazást vezérlő számokat a *Kanyar* tömbből olvassuk ki és a tömb indexeként a ciklusváltozót használjuk.

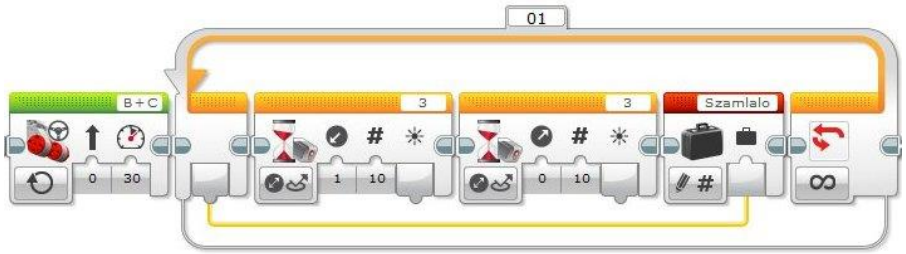


A háromszori lefutás után a motorokat leállítjuk, és várakozunk az ütközésérzékelő megnyomására.

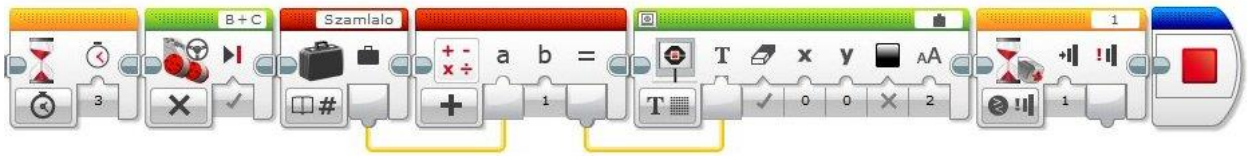
A programot általánosabban is meg lehet írni, több gombnyomás tárolására, vagy használva a tégla többi gombját, esetleg bonyolultabb mozgások leírására.

15/P4. Írjon programot, amelyet végrehajtva a robot egy fehér alapon fekete színű csíksor fölött halad 3 másodpercen keresztül! Három másodperc múlva megáll, és képernyőjére írja a csíkok számát, amelyek fölött teljes egészében áthaladt.

A csíkok számlálását a korábbi fejezetekben bemutatott elv alapján végezzük. Amikor a robot áthalad egy fekete csíkon kétszer változik meg a fényszenzora által mért érték. Először csökkenni fog, amikor a fehér felületről feketére ér, majd növekedni fog, amikor a fekete felületről fehérre kerül. Ha változás mértékét 10%-ban határozzuk meg, akkor a *Wait* modul fényszenzor módjának *Change* beállítása alkalmas a két változás észlelésére. Ezután kell egy változó értékét eggyel növelni. A változóban így a csíkok száma fog szerepelni és csak azoké a csíkoké, amelyeken teljes egészében áthaladt a robot. Mivel több csík is lehet, ezért a két *Wait* blokkot ciklusba helyezzük és a ciklusváltozó értéke lesz amit a csíkok számlálásánál eltárolunk. Ez ugyan eggyel kevesebb, mint a tényleges csíkok száma (mivel o-tól indul a számlálás), de a kiíratásnál ezt majd figyelembe vesszük.

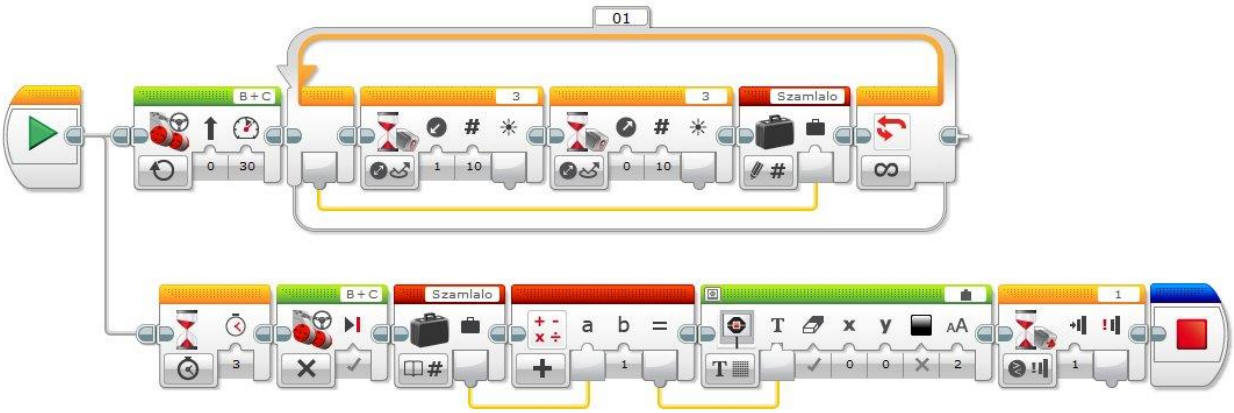


A *Wait* blokk használatával viszont nem tudjuk figyelni az időt. Tehát hiába állítjuk be a ciklust 3 másodperc futási időre, csak a két várakozás után értékeli ki a rendszer az időt, így előfordulhat, hogy jóval a 3 másodperc letelte után fog csak megállni a robot. A megoldás lehet az, hogy a *Wait* blokk helyett a Szenzor csoport fényszenzor modulját használjuk és egy stoppert, amely méri az időt. A két blokk által visszaadott értékekből egy összetett feltételt hozunk létre, amely szabályozza a ciklus futását. A másik lehetséges megoldás, hogy az időt egy külön szálon mérjük és ha letelt a 3 másodperc, akkor leállítjuk a motort és kiírjuk a megfelelő értéket a képernyőre. Ez utóbbi egyszerűbb megoldásnak tűnik.
A másik szálon tehát az időt mérjük egy *Wait* blokkal, és ha letelt a 3 másodperc leállítjuk a motorokat és a képernyőre írjuk a *Szamlalo* aktuális értékénél eggyel nagyobb értéket (a számlálás o-tól indult).

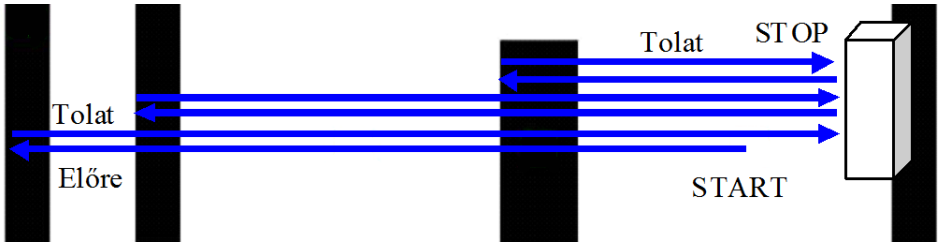


Az első szálon futó ciklus ugyan nem áll meg, de a robot igen. Mivel a megállást követően rögtön a képernyőre írjuk a változó értékét, ezért az a teljes egészében áthaladt csíkok számát fogja tartalmazni. A másik szálon a végtelen ciklus eközben valamelyik *Wait* blokk feltételének teljesülésére vár, de mivel nincs mozgás, ezért nem változik meg a *Szamlalo* értéke. A program az ütközésérzékelő megnyomására áll le.

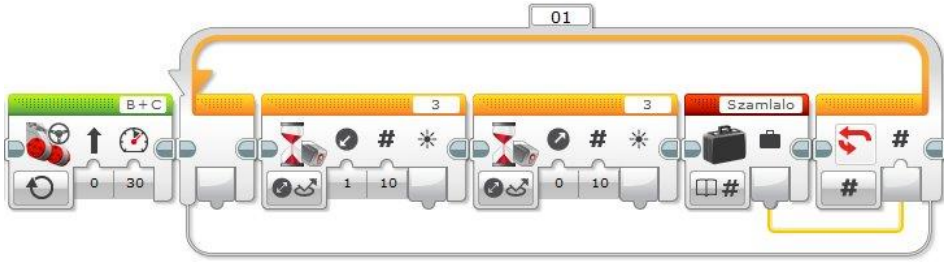
A teljes program:



15/P5. Írjon programot, amelyet végrehajtva a robot elindul egyenesen előre a haladási irányára merőleges, különböző vastagságú fekete vonalak fölött! A harmadik csík fölötti áthaladás után elkezd tolatni, az ütközésérzékelő benyomásáig. Ezután újra előre indul, majd a második csík után ismét tolatni kezd az ütközésérzékelő benyomásáig. Újra előre indul, majd az első csík után kezd tolatni az ütközésérzékelőig. Ezután megáll. Miközben mozog, két hangból álló dallamot játszik folyamatosan.
Pl.:

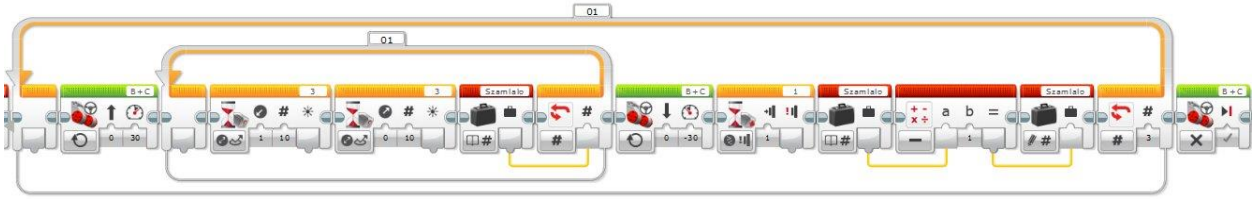


A program elvénél a csíkokon történő áthaladás érzékelése hasonlít a korábban bemutatott technikára. A motorok bekapcsolása után beillesztünk a programszálra két *Wait* blokkot, fényszenzor *Change* üzemmódban, amelyek közül az első csökkenést, a második növekedést figyel. Első lépésben három csíkon kell áthaladnia a robotnak, ezért a ciklusnak, amely a két fényszenzor blokkot tartalmazza háromszor kell lefutnia, második alkalommal már csak kétszer, és a végezetül egyszer. Szükségünk van tehát egy változóra (*Szamlalo*), amely ciklus futásainak számát tartalmazza.

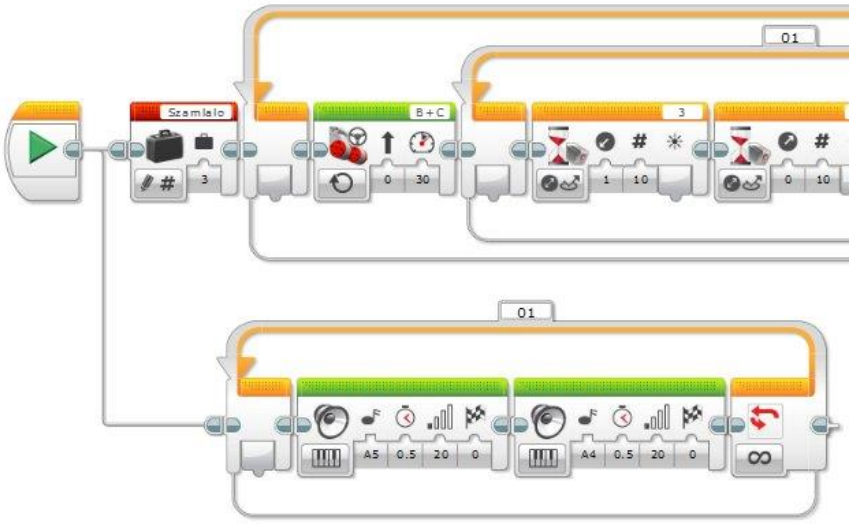


A ciklusból kilépve a *Szamlalo* változó tartalmát eggyel csökkentjük, így ha újra elindítanánk az utasítássort, akkor már csak két csíkon haladna át a robot. A változó értékének csökkentése mellett a motorok mozgását hátramenetbe kell kapcsolni és ütközésérzékelő benyomásáig működtetni.

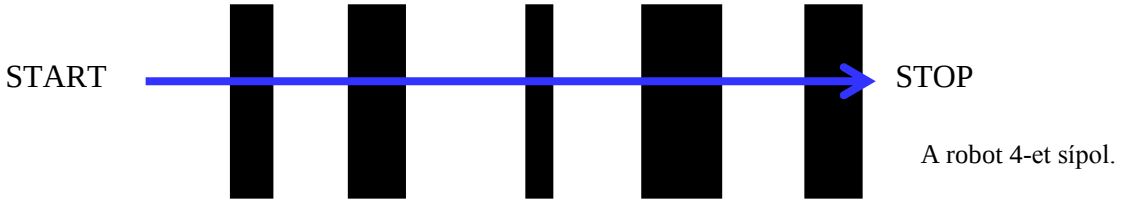
Ha a fenti két utasítássort egy háromszor lefutó ciklusba helyezzük, akkor a programunk már működik. A ciklus után a motorokat ki kell kapcsolni, mert egyébként az utolsó mozgássor irányába halad tovább a robot.



A két hangból álló dallam lejátszását a mozgással egy időben kell megvalósítani, ezért érdemes külön programszálra helyezni a két hangot váltakozva lejátszó blokkokat. Mindkét különböző hangot 0,5 másodpercig szólaltatjuk meg. A példánál beállított két hang: A5 és A4 (Tehát két zenei A hang egy oktáv különbséggel). A hangerőt 20%-ra állítottuk és csak az egyik hang lejátszása után szólal meg a második. A hanglejátszást végtelen ciklusba helyeztük, így csak a program vége fogja a dallamot elnémítani.



15/P6. Írjon programot, amelyet végrehajtva a robot fehér felületen különböző szélességű csíksor fölött halad, a csíkokra merőlegesen. A csíkok feketék. Az ötödik csíkon történő áthaladás után a robot megáll, és annyit sípol, mint a legszélesebb csík sorszáma a haladás irányában számolva.
Pl.:



Az elkészített programban tehát meg kell mérnünk a csíkok szélességét és ezeket eltárolni, hiszen csak az ötödik csíkon történő áthaladás után fog a robot arról dönteni, hogy melyik volt a

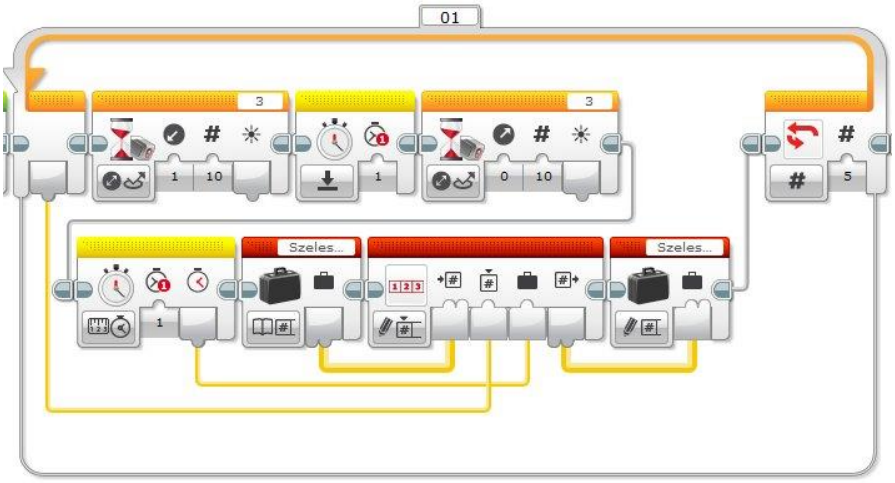
legszélesebb. Miután eltároltuk a csíkok szélességét, meg kell határoznunk, hogy melyik volt a legszélesebb, és ennek a sorszámát kell megjegyeznünk. Ha ez megvan, akkor annyit kell sípolnunk, amennyi ez a sorszám.

Mivel „sok" adatot kell tárolni, ezért szám típusú tömböt használunk az adatok rögzítésére. A csíkok szélességét stopperrel mérjük, tehát ha a robot egyenletesen halad, akkor két időpont között eltelt idő arányos a megtett távolsággal, így a szélességek összehasonlíthatók lesznek, ha megmérjük, hogy mennyi idő szükséges a fölöttük történő áthaladáshoz. A mérést úgy végezzük, hogy elindítunk egy stoppert, amikor a robot elérte a csík szélét, és eltároljuk a stopper értékét, ha áthaladt fölötte.

A csíkok érzékelésére a *Wait* modul, fényszenzor módjának *Change* állapotát használjuk. Amikor elértük a csík elejét a fényszenzorral mért érték 10%-nyi csökkenésével fogjuk ezt érzékelni és ekkor lenullázzuk a stoppert. Amikor elérte a robot a csík végét, akkor eltároljuk a stopper értékét a tömb típusú változó következő helyén.

A program előkészítéseként létrehozzuk a *Szelesseg* nevű számtömböt és kinullázzuk az első 5 elemét, és elindítjuk a motorokat, nem túl gyorsan (30).

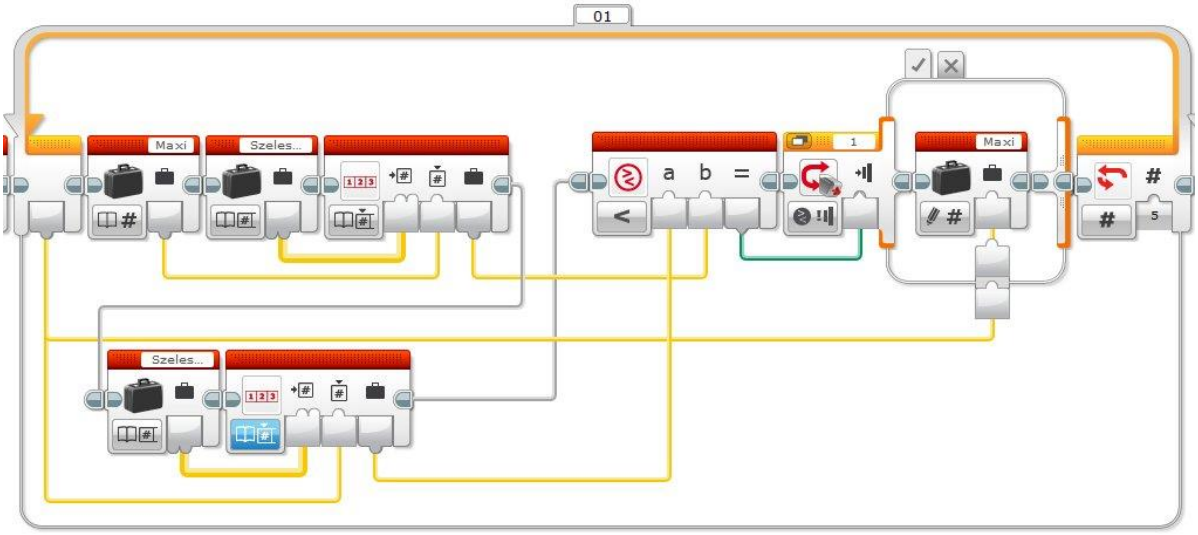
Az adattárolást végző ciklus 5-ször fut le. A fényszenzor első 10%-nyi változása után indul a stopper (*Reset*), majd a második 10%-os változás után a *Szelesseg* tömbbe kerül az értéke. A ciklusváltozót használjuk a tömb indexének eggyel növeléséhez, így az értékek rendre a 0.; 1.; 2.; 3.; 4. tömbelembe kerülnek.



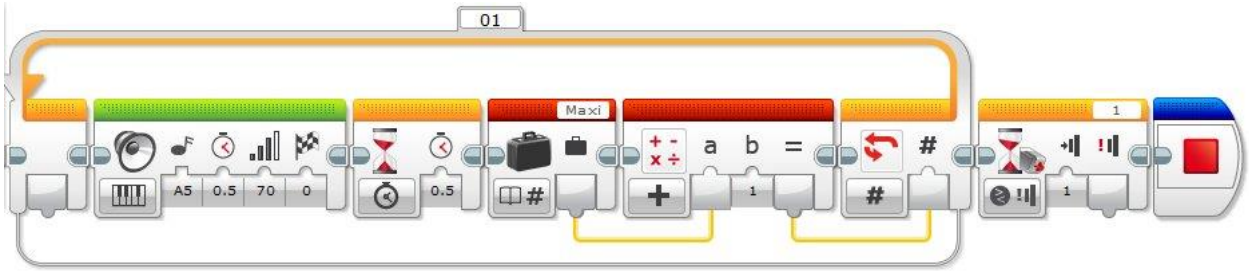
A tárolás után kell meghatároznunk a legnagyobb érték tömbbeli helyét (indexét). Ehhez a programozásban használt egyik alapalgoritmust a szélsőérték kiválasztás (jelenleg maximum) algoritmusát használjuk.

Az algoritmus alapötlete, hogy első lépésben a tömb legelső eleme a legnagyobb. Az aktuálisan legnagyobb elem tömbbeli indexét egy változóban tároljuk (a példánál ez a *Maxi*, és kezdőértéke a tömb első elemének indexe, vagyis 0). Ezután megyünk sorban végig a tömbelemeken és összehasonlítjuk őket az aktuálisan legnagyobbal. Ha valamelyik elem nagyobb, akkor onnantól kezdve a *Maxi* változó értéke a nagyobb elem indexe lesz. A következő elemet, már ezzel az újabb legnagyobb értékkel hasonlítjuk össze. Érdemes a legnagyobb elem keresésekor a tömbben elfoglalt helyét (indexét) megjegyezni egy változóban, és nem az értékét. Tehát nem azt jegyezzük meg, hogy mennyi a legnagyobb, hanem azt, hogy hányadik a legnagyobb. Így bármikor elérhetjük

a legnagyobb értéket is és azt is, hogy ez hányadik, míg fordított esetben (ha az értéket jegyeznénk meg), az érték rendelkezésünkre áll, de azt nem tudjuk, hogy ez hányadik eleme a tömbnek.
A *Maxi* változó tartalmazza az aktuálisan legnagyobb elem indexét és a ciklusváltozót használjuk a többi elem indexeként, amely folyamatosan eggyel növekszik minden ciklusleftatáskor.
Ha az összehasonlítás eredménye alapján, az aktuális érték (a ciklusváltozó alapján meghatározott) nagyobb, mint a jelenleg legnagyobbnak tartott, akkor a *Maxi* változó felveszi a ciklusváltozó értékét.



A ciklus lefutása után a legnagyobb értékű tömbelem indexe kerül a *Maxi* változóba. A hanglejátszást tehát annyiszor kell lefuttatni, amennyi a *Maxi* értéke, illetve annál eggyel többször, mert a ciklusváltozó 0-tól indul, így a *Maxi* értéke 1-gyel kisebb, mint legszélesebb csík sorszáma.



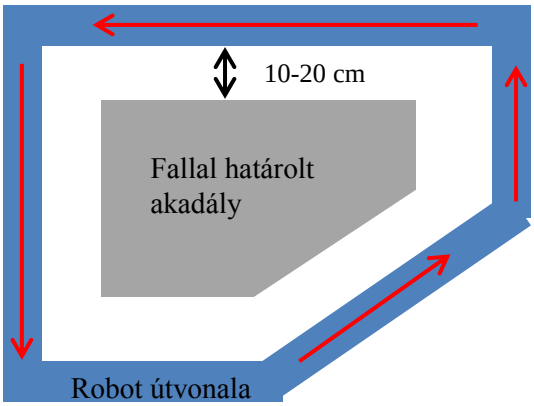
Mivel a maximum kiválasztás algoritmusa az egyik fontos alapalgoritmus, ezért értelmezéséhez a következő számpéldát érdemes áttanulmányozni.
A feladat, hogy a tömbben lévő 5 szám közül a legnagyobbat határozzuk meg.

Tömbelem indexe	Tömbelem	*Maxi* változó értéke	A tömb jelenlegi legnagyobb értéke	Aktuális tömbelem	Össze-hasonlítás	Összehasonlítás eredménye
0.	3	0	3	3	3<3	hamis, nincs változás
1.	5	0	3	5	3<5	igaz, *Maxi* értéke innentől 1
2.	2	1	5	2	5<2	hamis, nincs változás
3.	8	1	5	8	5<8	igaz, *Maxi* értéke innentől 3
4.	7	3	8	7	8<7	hamis, nincs változás

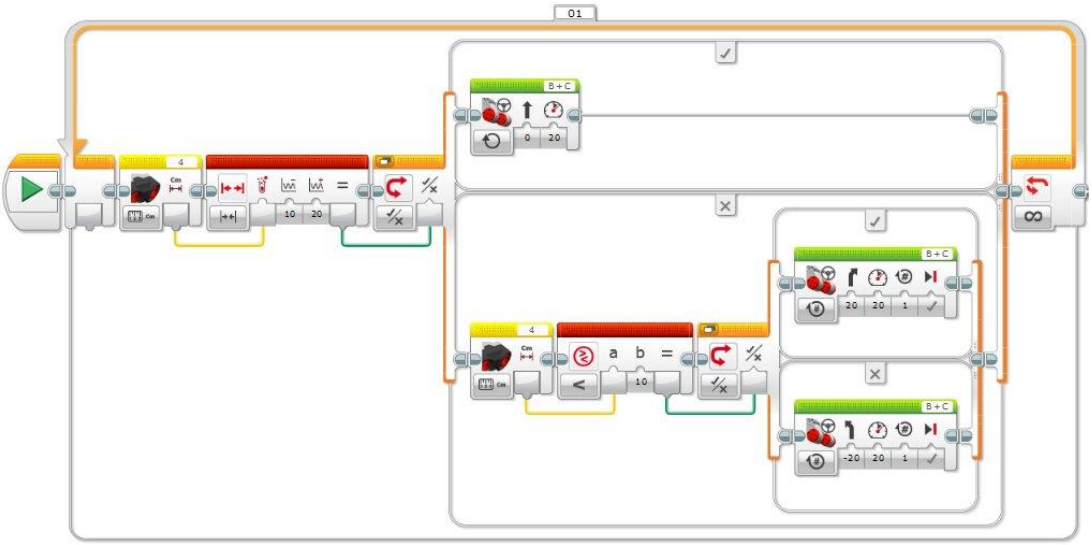
A ciklus végeztével a *Maxi* változó értéke 3, tehát a tömb 3-as indexű eleme a legnagyobb, ami 8.

15/P7. Írjon programot, amelyet végrehajtva a robot ultrahang szenzorával képes követni a „fallal" határolt akadály körvonalát! Tehát valamilyen körüljárást választva a faltól 10-20 centiméteres sávon maradjon belül a robot, annak görbületeinél (sarkok) is beleértve. Az ábra értelmezi a feladatot. A körüljárási irány az óramutató járásával ellentétes (kék sáv).

Pl.:



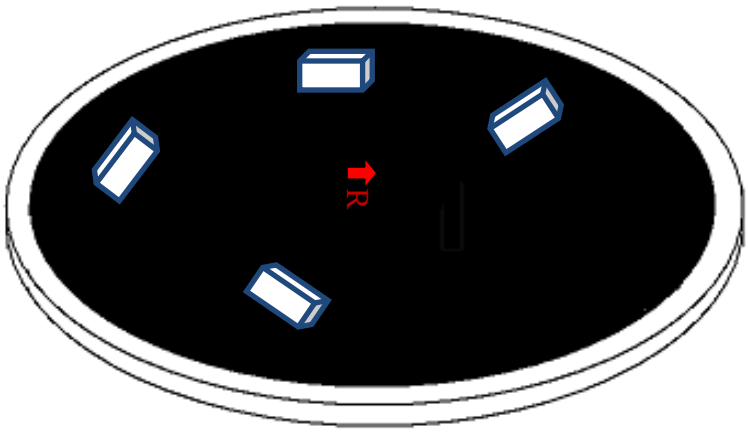
Az algoritmus elve a fényszenzoros útvonalkövetés elvével egyezik meg. Itt azonban három különböző érték szabályozza a mozgást. Ha az ultrahangszenzor 10-20 cm-en belüli távolságot mér, akkor a robot egyenesen halad. Ha a mért távolság kisebb, mint 10 cm, akkor balra fordul, ha nagyobb, mint 20 cm, akkor pedig jobbra. A fordulást a *Steering Motor* blokkal oldottuk meg kormányzás paramétert 20 illetve –20 értékre állítva. Így a robot íven fordul, ezáltal kevésbé éles kanyart ír le. Ha a kanyarodás túl éles, akkor az ultrahangszenzor által mért érték pontatlanná válhat, a visszaverődési szög nagymértékű változása miatt. A falkövetés irányát meg tudjuk változtatni, ha a belső elágazás két ágában szereplő motorvezérlő blokkokat felcseréljük.



A működést tekintve első lépésben az ultrahangszenzorral megmérjük a faltól való távolságot. Ha ez 10-20 cm közötti, akkor egyenesen halad a robot. Ha ettől eltér, akkor újra megmérjük a távolságot és kisebb, mint 10 esetén az egyik irányba fordulunk. Ha a mért érték nem kisebb, mint 10, akkor csak 20-nál nagyobb lehet, mivel 10-20 között a felső szál utasítása érvényes. Tehát 20-nál nagyobb esetben a másik irányba fordulunk. A kétszeri ultrahangszenzoros mérés helyett változót is használhatunk.

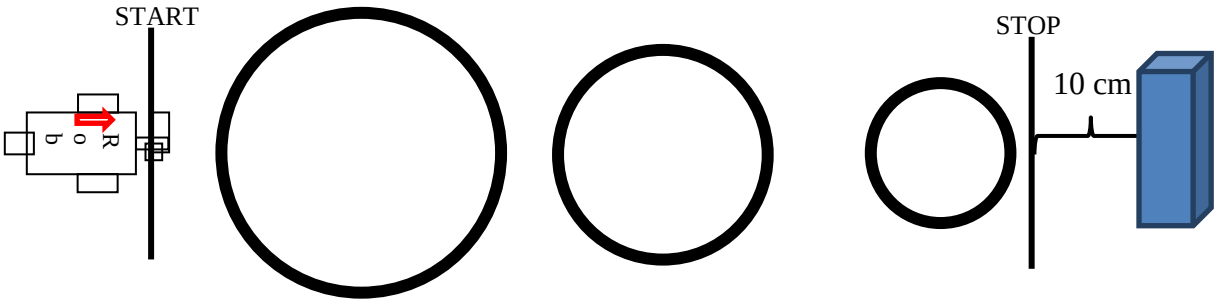
16. Összetett feladatok

16/F1. Írjon programot, amelyet végrehajtva a robot egy adott területen belül megkeres tárgyakat és azokat a terület határáig tolja. A terület kör alakú és kb. 120 cm átmérőjű (robotszumó pálya). A területet a pálya alapszínétől különböző színű sáv határolja. A határoló sáv szélessége kb. 4 cm. A pálya alapszíne lehet fekete, ekkor a határoló sáv fehér színű, de a pálya lehet fehér alapszínű is, ekkor a határoló sáv színe fekete. A robot programját úgy kell elkészíteni, hogy akár fekete, akár fehér alapszínű pályán működjön. A robot a kör közepéről indul, és tetszőleges algoritmussal keresheti a tárgyakat, de nem hagyhatja el a pályát. A tárgyak téglatest alakú dobozok, amelyek alapja 4 cm x 15 cm és magasságuk legalább 8 cm. A tárgyak bárhol lehetnek a pályán belül, és legalább négy darab tárgy van. A robot keresési távolsága legfeljebb 40 cm lehet. Tehát a programot úgy kell elkészíteni, mintha a robot 40 cm-nél messzebbre nem „látna". Ha egy tárgyat észlelt a robot, akkor azt a pálya széléig kell tolnia. Ha tárgy bármely része érinti a pályát határoló sávot, akkor a tárgy levehető a pályáról. A robot programja az indítástól mért 1 perc elteltével automatikusan álljon le. Hibának számít, ha a robot bármelyik része a pályát határoló sávon kívül érinti a talajt (a tesztpálya kb. 0,5 cm magas a talajtól mérve).
Pl.: *(Az ábra nem méretarányos!)*

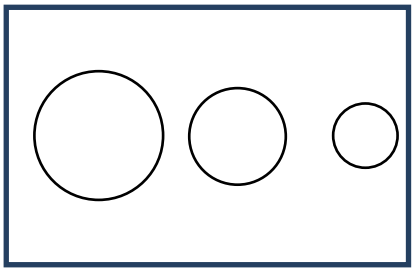


16/F2. Írjon programot, amelyet végrehajtva a robot egyenes vonalban halad egy adott pontról indulva különböző sugarú fekete színű körök fölött. A körvonalak vastagsága legalább 1,5 cm, és a körök nem metszik át és nem is tartalmazzák egymást. A körök középpontjai a robot haladási pályáján helyezkednek el. A robot az egyenes vonalú mozgását akadálytól 10 cm-re fejezi be. Ezután a képernyőjére rajzolja a köröket, a megtett útjának megfelelően arányosan. Legalább kettő, és legfeljebb négy kör lehet a pályán. A program ütközésérzékelő benyomására álljon le, hogy a képernyőképet át lehessen tekinteni. A robot nem fordulhat vissza, csak az egyszeri, egyenes vonalú pályán történő áthaladás során gyűjthet adatokat a körökről. A képernyőre rajzolt körvonalak 1 pixel vastagságúak legyenek.

Pl.:



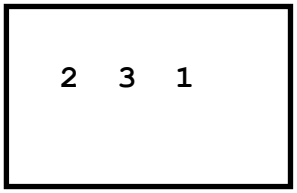
Képernyőkép:



16/F3. Írjon programot, amelyet végrehajtva a robot az ábrán jelzett startpozícióból indul egyenesen előre három egymással párhuzamos, de különböző szélességű fekete csík fölött, a csíkokra merőleges irányban. A pálya alapszíne fehér. Akadálytól 10 cm-re megáll és képernyőjére írja a csíkok sorszámát a szélességüknek megfelelő csökkenő sorrendben, a képernyő ugyanazon sorába, de térközzel elválasztva egymástól. A csíkok számozása a startpozícióhoz legközelebbivel kezdődik és a robot haladási irányának megfelelően növekszik. A három fekete csík szélessége: 2,5 cm; 5 cm illetve 7,5 cm, hosszuk tetszőleges, de legalább 20 cm. A csíkok sorrendje tetszőleges lehet. A program befejezése előtt 5 másodpercig várakozzon, hogy a képernyőre írt adatok elolvashatók legyenek. A csíkok közötti távolság legalább 10 cm.

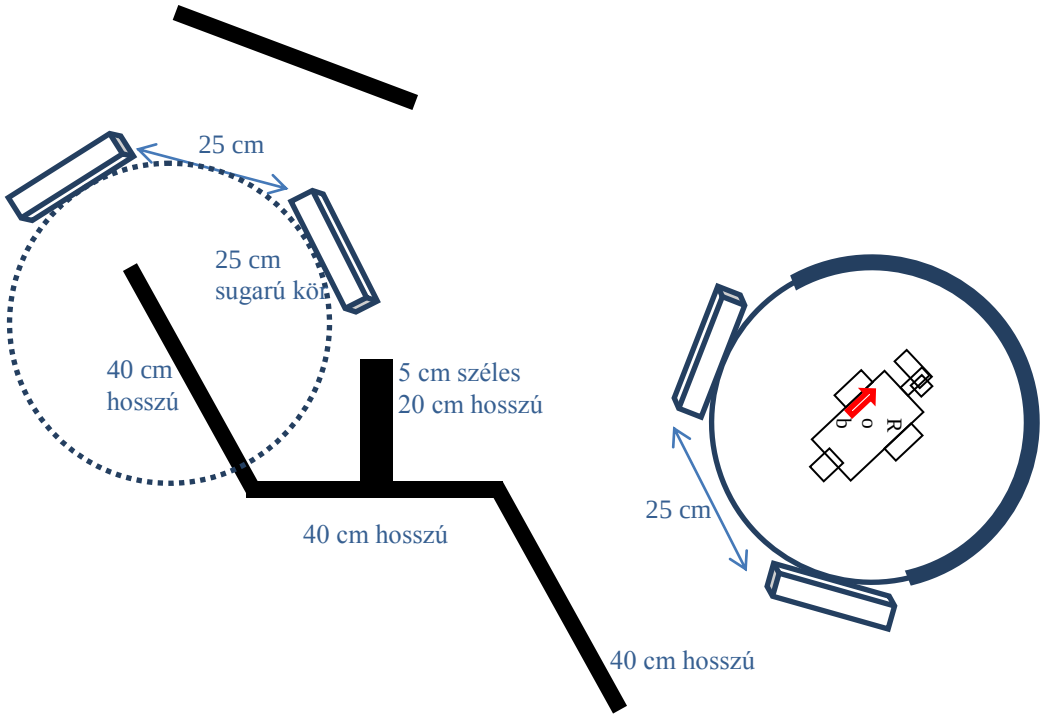
Pl.:

A képernyőre írt szélesség szerinti csökkenő sorrend, egymástól olvashatóan térközzel elválasztva:



16/F4. Írjon programot, amelyet végrehajtva a robot kezdésként megkeres egy kaput és áthalad rajta! A kapu két téglatestből áll, amelyek legalább 10 cm x 8 cm x 5 cm méretűek. A robot egyenletes sebességgel helyben forog és körülette két doboz van elhelyezve úgy, hogy a forgási középponttól legfeljebb 25 cm sugarú körvonalra leghosszabb oldalukkal érintőlegesen helyezkednek el (lásd ábra). A téglatestek egymástól 25 cm távolságban szimmetrikusan vannak elhelyezve. A robotnak át kell haladnia a két téglatest között (kapu) úgy, hogy egyikhez sem ér hozzá. A robot az ábrán vastag vonallal jelölt körív bármely pontja felé nézhet és a forgásiránya is tetszőleges lehet.

A kapuk között áthaladva fekete vonalig halad előre, majd azt elérve követni kezdi a lehető legrövidebb útvonalon mindaddig, amíg a vonal végén lévő dobozt 10 cm-re meg nem közelíti, ekkor megáll. A fekete vonal végén lévő dobozt csak útvonalkövetéssel közelítheti meg. A doboztól 15 cm-re megállva újabb kaput kell megkeresnie és áthaladnia rajta, anélkül, hogy hozzáérne. A kapu pozíciója az ábra szerinti. A robot a kapun áthaladva a fekete csíknál megáll.



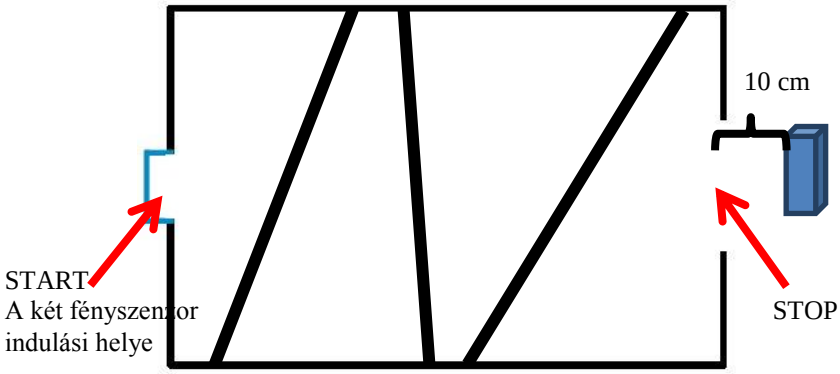
A következő feladatok az elmúlt két év robotprogramozó versenyeinek feladatai közül kerültek ki. A megoldásukhoz használható robot a könyv elején bemutatottól annyiban különbözik, hogy két szín- vagy fényszenzorral van felszerelve, amelyek egymással párhuzamosan, lefelé néző állapotban a robot elején helyezkednek el. A két szenzor közötti távolság kb. 2-3 cm.

16/F5. A robot feladata, hogy a képernyőjére rajzolja egy megadott pályaszakasz térképét. A robot a pályán egyszer haladhat végig, a megadott pozícióból indulva egyenesen előre, majd a végighaladás után kell a képernyőjére rajzolnia. A pálya végét akadály jelzi, attól 10 cm-re kell megállnia. A pályán a robot egyenes vonalú haladását keresztező fekete színű egyenes vonalak találhatók (a pálya alapszíne fehér). A vonalak száma legalább kettő és legfeljebb négy. A vonalak nem feltétlenül merőlegesek a haladási irányra, de töréspontot nem tartalmaznak, és teljes egészében keresztezik a pályát (pályaszéltől-pályaszélig). A vonalak keresztezhetik

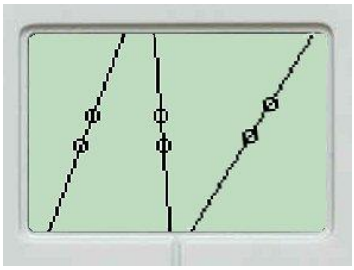
egymást is, de nem a robot haladási útvonalán. A pálya arányai megegyeznek a robot képernyőjének arányával . A pálya hosszabbik éle 50 cm hosszú. A két fényszenzor távolsága 5 cm. A képernyőre rajzolt térkép kb. méretarányos kell, hogy legyen, de egy vonal szélességét elegendő 1 pixel vastagsággal ábrázolni. A robot a pályán hosszanti irányban haladhat, annak középvonalán és egyszeri végighaladása során vehet mintát a vonalak hálózatából.

A robot képernyőjére rajzolja fel azokat a pontokat, ahol a fényszenzorai először érzékelték a fekete vonalakat. A pontok jelölésére 2 pixel sugarú köröket használjanak, amelyek középpontja legyen a vonal észlelésének pozíciója. Ezek a pontok a program végéig maradjanak a képernyőn! Az észlelt vonalakat szintén rajzolja a robot képernyőjére 1 pixel vastagságban. A program ütközés érzékelő benyomására álljon le.

A pálya például:



A képernyőkép:



16/F6. A robot fehér alapú pályán kell, hogy egy fekete vonalakból álló útlabirintuson végighaladjon. A labirintus merőleges útkereszteződéseket tartalmaz. A kereszteződésnél a fordulási irányt egy robot által előzőleg leolvasott kódsor alapján lehet meghatározni.

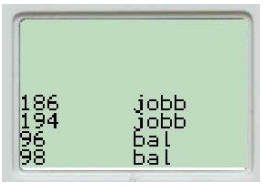
A kódsor a robot haladási irányára merőleges szélesebb vagy keskenyebb fekete vonalakból áll. A beolvasandó vonalak száma négy. A szélesebb vonal szélessége kétszerese a keskenyebbnek. A kódsor kezdődhet keskenyebb és szélesebb vonallal is. A robot áthalad a kódsort jelentő vonalsorozat fölött, majd megáll. Ekkor át lehet helyezni az útvonalhálózat kezdő pozíciójába, majd valamilyen jelzés hatására megkezdi a labirintusban a mozgását. Ez a jelzés lehet ütközésérzékelő benyomása vagy hangjelzés, ...

Egy útkereszteződéshez érve a leolvasott kódsor alapján hozza meg a döntést a továbbhaladás irányáról. Ha az útkereszteződésnek megfelelő sorszámú vonal széles volt, akkor jobbra fordul, egyébként balra. Ezután tovább követi az útvonalat a megfelelő irányban mindaddig, amíg újabb kereszteződéshez nem ér. Ekkor ismét hasonló elvek alapján dönt és folytatja útját tovább. ... Az utolsó fordulás (negyedik) után megáll.

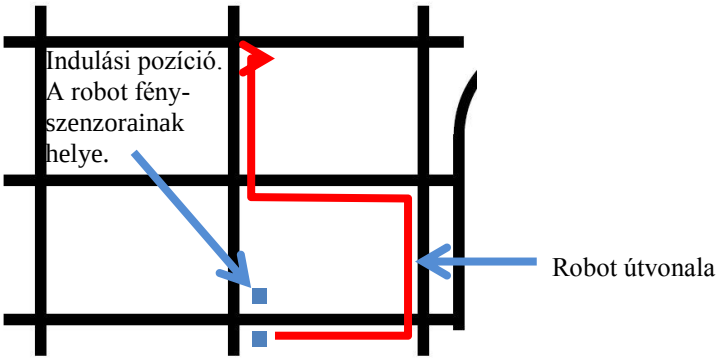
A kódsor beolvasása után írja a robot a képernyőjére egymás fölötti sorokba a csíkok szélességét a programozás során használt mértékegységben (milliszekundum, tengelyfordulási szög). A képernyőre írást a képernyő alján kezdje, és fölfelé haladva jelenítse meg a számokat. A számsor minden egyes értéke mellé írja a „bal" vagy „jobb" szavak közül azt, amelyik a fordulási irányt jelenti. A program befejezéséig maradjon meg a képernyőn a teljes kiírt számsor és szöveg.

A kódsor pl.:

A kódsornak megfelelő képernyőkép:



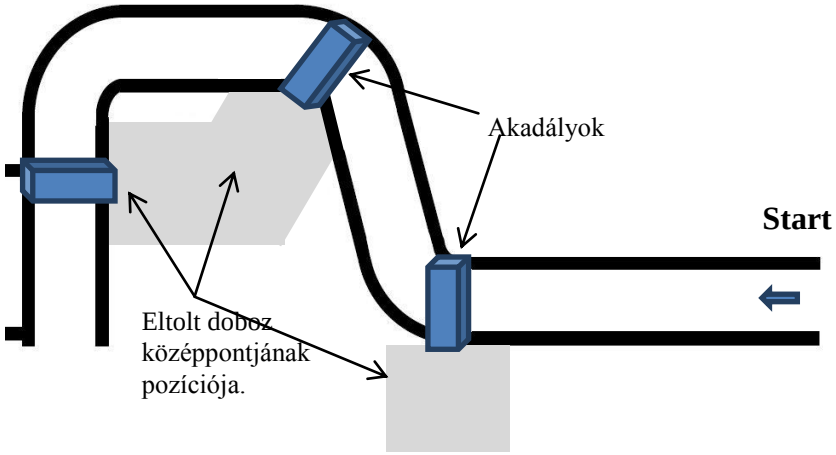
A pálya és a haladási útvonal a kódsornak megfelelően:



16/F7. A robotnak egy 10 cm széles fehér útvonalon kell végig haladnia. Az útvonal széleit fekete csík jelzi. A robot nem térhet le az útvonalról. Az útvonal elhagyásának minősül (letérés), ha a két kerék középpontját összekötő szakasz felezőpontja elhagyja a fekete csíkok által határolt részt. Az útvonalon keresztben akadályok vannak elhelyezve (összesen 3 db). Az akadályok téglatestek, amelyek teljes egészében átérik a 10 cm-es útvonalat. A robot feladata, hogy ezeket az akadályokat eltávolítsa az útról úgy, hogy közben nem tér le róla. Az akadályok bárhol lehetnek az útvonalon. Az akadályokat a robot nem tolhatja maga előtt, azokat az út mellé kell tolnia, az akadály pozíciójától ±10 cm-es sávon belülre úgy, hogy a doboz alapjának középpontja az út mellett legyen. Mindhárom akadályt (a robot haladási iránya szerint) az út bal oldalára kell tolni. A robotnak nem kell megállnia a pálya végén.

A feladatmegoldáshoz felhasználható 1 db 10-es méretű LEGO rúd, amelyet szabadon el lehet helyezni a roboton.

Pl.:



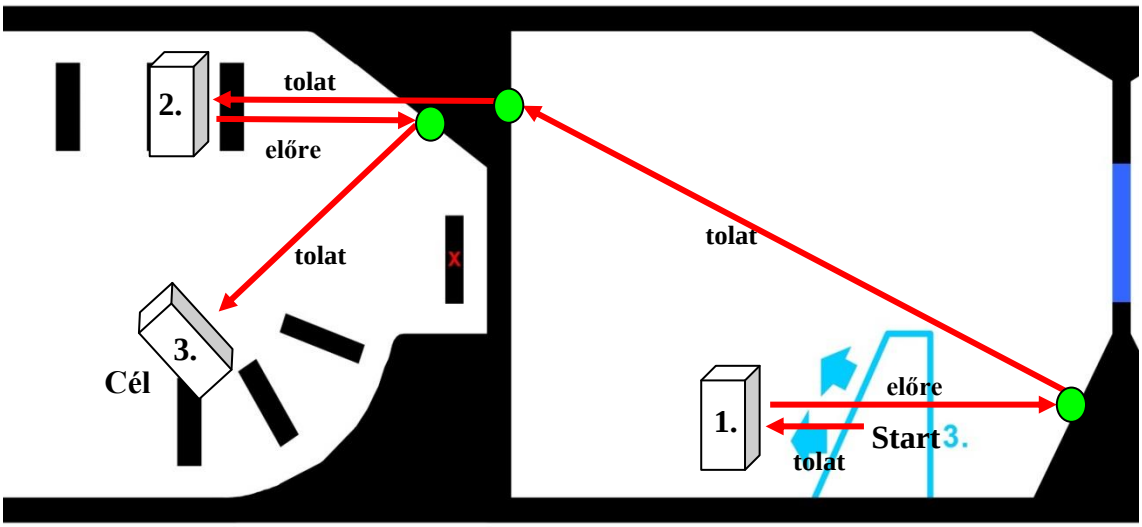
16/F8. A robotnak egy összetett előre-hátra mozgást kell végeznie. Bizonyos helyzeteket elérve tolatnia kell, amelyet egy, a pályán elhelyezett akadály szakít meg, amit a hátra szerelt ütközésérzékelőjével érzékelhet. Ezután előre kell haladnia, amelyet a pályára felfestett csík szakít meg, amit az előre szerelt fényérzékelőivel érzékelhet. A csíkok nem merőlegesek a robot haladási irányára. A következő tolatást mindig a csíkra merőleges irányban kell folytatni. Tehát tolat az akadályig, majd előre halad a csíkig és így tovább. A tolatást mindig az akadály észlelése, az előrehaladást pedig a csík észlelése szakítja meg. Az előre haladás mindig egyenes irányú, de a tolatás irányát a csík és a haladási irány szöge szabályozza. Minden esetben a fényérzékelővel észlelt csíkra merőlegesen kell megkezdeni, vagy folytatni a tolatást.

A robot tolatással indul egyenesen, majd ütközésre egyenesen előre halad a fekete csíkig. Erre merőlegesen tolatni kezd a következő fekete csíkig. Erre merőlegesen folytatja a tolatást ütközésig. Ekkor előre indul egyenesen a fekete csíkig. Végül erre merőlegesen kezd tolatni ütközésig és megáll.

Három akadályt kell az ütközésérzékelőjével a megfelelő sorrendben (a pálya képén a hasábok számozásával jelzett sorrend) megérintenie, valamint három fekete csík esetén kell, rá merőlegesen pozícionálni és folytatni az útját (a pálya képén zöld körrel jelzett).

A robotnak a pálya képén piros színnel jelzett útvonalat kell bejárnia, amelyet az akadályok és a fekete csíkok szabályoznak.

A robot a kékkel jelzett trapéz alakú területről indul (a két fényszenzora a trapézon belül van).



16/F9. A robot fehér alapú pályán, egyik fényszenzorával kell, hogy kövesse a fekete színű útvonalat. Az útvonalra közel merőlegesen, attól néhány cm távolságra fekete csíkok találhatók a pályán. A robot másik fényszenzora ezek fölött a csíkok fölött kell, hogy haladjon.

A robotnak abban az esetben, ha a 2. fényszenzora éppen fekete csík fölött halad, két hangból álló dallamot kell játszania, mindaddig, míg a 2. szenzora a fekete csík fölött van.

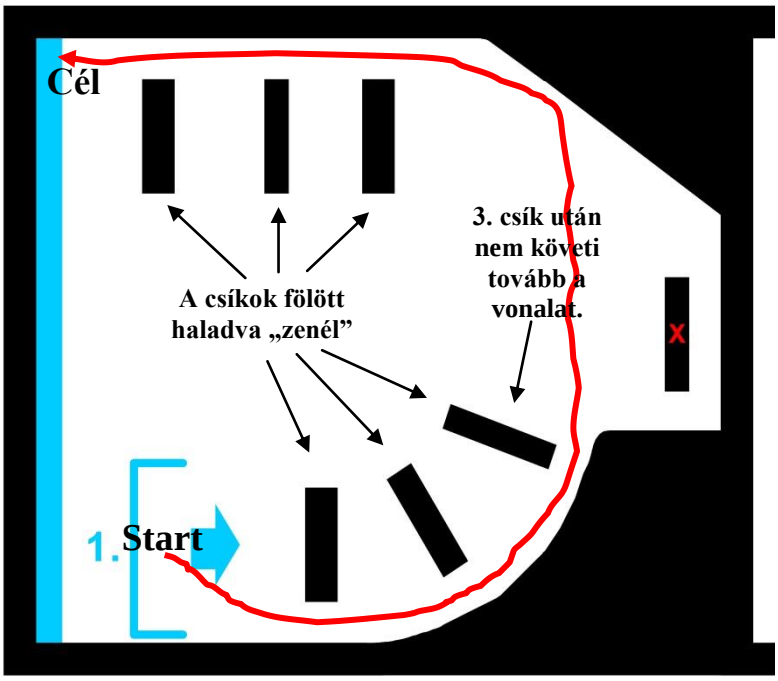
A harmadik fekete színű csíkon áthaladva a robot hagyja abba az útvonalkövetést és a vele „szemben lévő" fekete területig kell haladnia (a szemben lévő kifejezést a pálya képén jelölt útvonal értelmezi).

Itt ismét az egyik fényszenzorával útvonalkövetést végez (a pálya képén jelölt irányba), míg a másik fényszenzora ismét a fekete csíkok fölött halad és a robot ugyanazt a két hangból álló dallamot szólaltatja meg, minden áthaladáskor. Ha a robot túlságosan lassan halad, akkor előfordulhat, hogy a dallamot egy áthaladáskor akár többször is megszólaltatja.

A két megszólaltatandó hang:

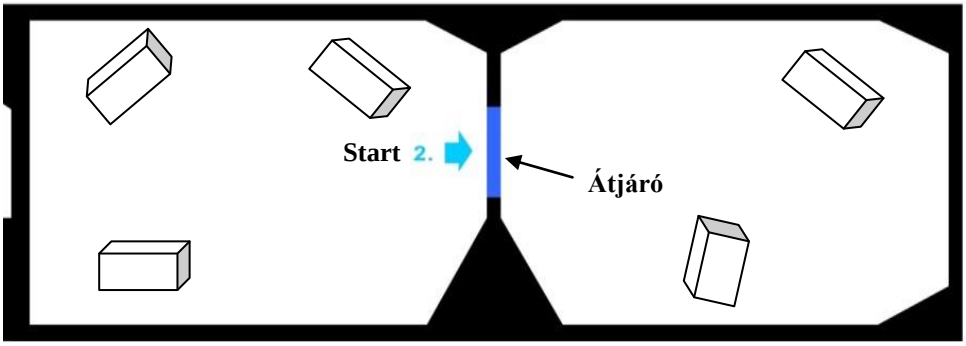
440 Hz – 200 ms időtartamig (zenei A)

528 Hz – 200 ms időtartamig (zenei C)



16/F10. Egy fehér alapú, fekete szegéllyel határolt területen belül kell a robotnak tárgyakat (hasábokat) megkeresnie és a kereten kívülre tolnia (a tárgy a kereten kívülinek számít, ha bármely része érinti a fekete színű falat, vagy teljes terjedelmében azon kívül van). A keret két részre osztott „szobából" áll, amelyek között átjáró található (az átjáró színe kék). A robot a keretet nem hagyhatja el és a két szobát elválasztó falon sem haladhat át. (Áthaladás/elhagyás alatt azt értve, hogy mindkét kereke egyszerre nem haladhat át a falon. Az megengedett, hogy egyik kereke áthaladjon a falon.) A tárgyakat keresheti az ultrahang szenzorával és az ütközésérzékelőjével. Az adott idő alatt a robotnak minél több tárgyat el kell távolítania a pályáról. A tárgyak mindkét szobában lehetnek, így a robotnak mindkét szobát be kell járnia.

A robot úgy indul, hogy mindkét fényszenzora a két szobát elválasztó átjáró (kék színű) fölött van.


Start 2.
Átjáró

17. A NATIONAL INSTRUMENTS

17.1. A National Instrumentsről

A méréstechnika, automatizálás és vezérlés területén mára piacvezető pozíciót betöltő National Instruments Corporationt (NIC) 1976-ban alapították az amerikai Texas állam fővárosában, Austinban. A vállalat ma világszerte több mint 7100 főt foglalkoztat és több mint 50 országban van közvetlenül jelen.

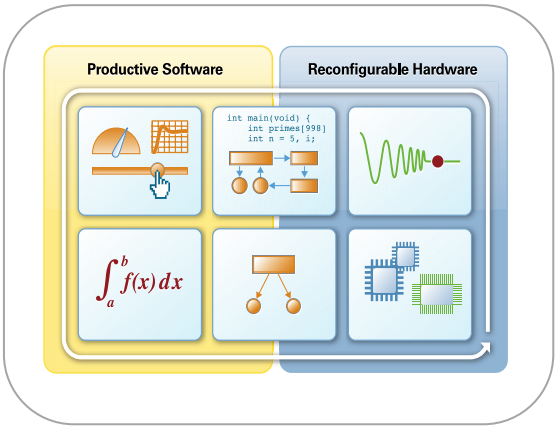
A National Instruments 2001-ben Magyarországon, Debrecenben nyitotta meg első tengeren túli gyárát. Ma a gyár több mint 1000 alkalmazottnak a munkaadója, illetve itt történik a vállalat teljes hardvergyártásának több mint 80%-a. Budapesten található az NI Kelet-Európát kiszolgáló kereskedelmi és marketing központja, amely szintén több mint egy évtizede — a debreceni csapattal szoros együttműködésben — segíti a hazai mérnököket, kutatókat, oktatókat és diákokat munkájuk hatékonyságának fokozásában.

A National Instruments (NI), mint a legújabb technológiák fejlesztését és az innováció megvalósulását elősegítő multinacionális vállalat, a grafikus rendszerfejlesztésen keresztül egy olyan integrált szoftver- és hardver platformot nyújt, amely bármely mérés-, illetve irányítástechnikai rendszer fejlődési idejét lerövidíti. Az NI által fejlesztett számítógép műszerezésű hardver és szoftver termékeket mérnökök, kutatók, oktatók és diákok alkalmazzák szerte a világon. Az NI éves költségvetésének jelentős hányadát, 16-18%-át fordítja évről-évre kutatás-fejlesztésre. Az elmúlt 15 évben a FORTUNE magazin a National Instrumentset minden alkalommal a legjobb 100 munkahely közé választotta Amerikában. Az National Instruments Hungary Kft. 2002 óta minden évben részt vesz a Legjobb Munkahely Felmérésben, ahol minden alkalommal az első 40 helyezett között szerepelt. 2012-ben, az európai Legjobb Munkahely Felmérésben a 6. helyezést érte el a vállalat.

17.2. Küldetésünk

A National Instrumentsnél úgy gondoljuk, hogy a jövő kihívásainak nagy részét, mint például a tiszta víz, hatékony energia előállítás, intelligens közlekedési rendszerek, adatbiztonság, mobil kommunikáció, kiberfizikai rendszerek, kutatók és mérnökök fogják megoldani. Ezen összetett problémák kezelésére, illetve innovatív megoldások létrehozásához hatékony eszközökre van szükségük. A National Instruments olyan eszközöket fejleszt a mérnökök és kutatók számára, amelyek elősegítik a termelékenység, innováció és felfedezések fejlődését.

17.3. Megoldásaink



A grafikus rendszertervezés moduláris eszközökre és a flexibilis grafikus fejlesztői szoftver kombinációjára épül. Ennek alkalmazásával egy magasabb absztrakciós szinten tervezhetjük meg és implementálhatjuk mérő-, vezérlő-, teszt- és automatizálási rendszereinket úgy, hogy a feladatnak legalkalmasabb platformon (PC, ipari PC, beágyazott rendszerek, FPGA, GPU) moduláris elemekből építjük fel a rendszer architektúrát, aminek funkcióját szoftveresen definiáljuk a grafikus rendszertervezés segítségével. A szoftver más alkalmazásfejlesztői környezetben megvalósított komponenseket is képes integrálni (pl: .NET, Open CL, C, C++, C#, .m script, szimulációs modellek stb.) és azokat a kiválasztott célhardveren futtatni, ezzel is növelve a flexibilitást és produktivitást a prototípus-fejlesztéstől az ipari kivitel elkészítéséig. Az kereskedelmi forgalomban is megvásárolható eszközök felhasználásával a teljes fejlesztési folyamat költsége és hosszú távú karbantartási költsége is csökkenthető.

17.4. Alkalmazási területek, iparágak

A grafikus rendszertervezői eszközöket több mint 35000 vállalatnál alkalmazzák világszerte, mely megoldást kínál a műszaki kihívások kezelésére számos iparágban, mint például a következőkben:

- Energetika, megújuló energia
- Közlekedés/autóipar
- Egészségügyi berendezések/orvosi műszerezés
- Kommunikációs rendszerek
- Agrárium/mezőgazdaság
- Oktatás/kutatás
- Repülő és hadi ipar
- Űrkutatás

Alkalmazási területek, ahol már sikeresen alkalmazzák a grafikus rendszertervezői környezetet:

- Intelligens adatgyűjtő, vezérlő és diagnosztikai rendszerek
- Teszt és tesztelés automatizálása
- Gyártási folyamat automatizálása – intelligens centralizált és decentralizált rendszerek
- Minőségellenőrzés – funkcionális és vizuális ellenőrző rendszerek
- Célgépek és beágyazott rendszerek fejlesztése
- Gyors prototípus- és demonstrátor-fejlesztés (mérő, vezérlő, teszt és automatizálási alkalmazások)
- Gyakorlati oktatást segítő rendszerek

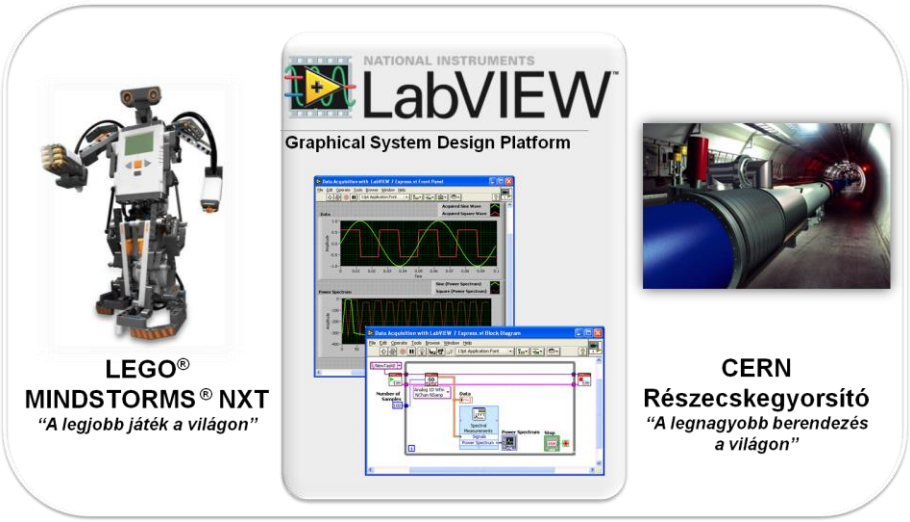
17.5. Szolgáltatások

A National Instruments a világ több mint 50 országában kínál terméktámogatást az eszközeihez és teljes körű oktatást a hatékony használat elsajátításához. Az Alliance Partner Program olyan kis- és középvállalkozásoknak segít a világpiacra való kijutásban az NI értékesítési és marketing csatornáin keresztül, akik az NI technológiákra alapozva kulcsrakész terméket és/vagy szolgáltatást kínálnak a végfelhasználók számára egy adott alkalmazási vagy ipari szegmensben.

17.6. A LabVIEW grafikus fejlesztői környezet

A **LabVIEW** egy grafikus fejlesztő környezet, amelyet ma a mérnökök és kutatók az elsőszámú fejlesztőeszközként tartanak számon mérő-, adatgyűjtő, automatizálási és tudományos kutatási feladatok megoldásához.

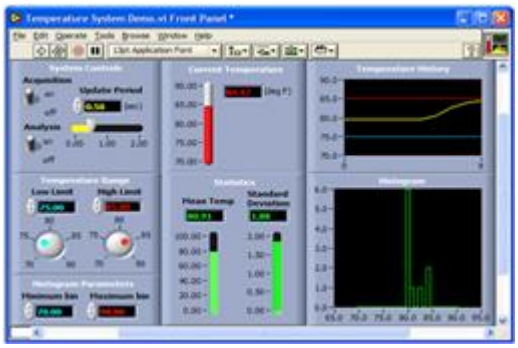
A LabVIEW 1986-os megjelenését követően rohamléptekben hódította meg a mérnökvilágot. A National Instruments által kifejlesztett **LabVIEW** sikere a szoftver sokoldalúságában és hatékonyságában rejlik. A PC alapú adatgyűjtő és automatizálási feladatok megoldásához kifejlesztett LabVIEW egy olyan grafikus fejlesztőkörnyezet nyújt, amely segítségével egyszerű és komplex projekteket is gyorsan elkészíthetünk: a Lego Robot vezérléstől a világ legnagyobb részecskegyorsítójáig (CERN).



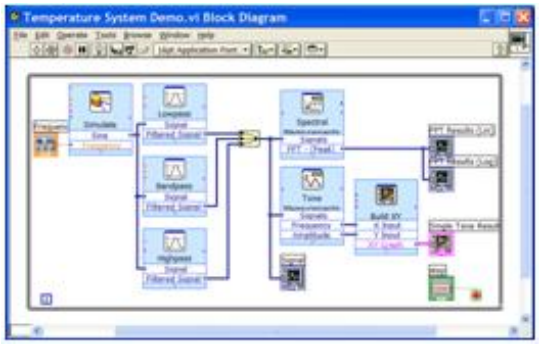
A grafikus rendszertervezés a mérnökök és kutatók számára egy olyan gyorsabb megoldási lehetőség, amivel a valós életben is működő rendszereket tervezhetnek. Eltérően a matematikai modellezést megcélozó modell-alapú tervezéstől, a grafikus rendszertervezés segít a mérnököknek abban, hogy inkább az alkalmazásra koncentrálhassanak, mintsem a számításokon való munkára.

A LabVIEW és a grafikus rendszertervezés segítségével a mérnökök a rendszer elképzelésétől gyorsabban jutnak el a megvalósításig, mert a szoftver elvonatkoztat a rendszer komplexitásától és széles körű I/O és hardver platformmal is integrálható.

A LabVIEW sikerének titka a szoftver grafikus fejlesztőfelületében rejlik. A szöveges programozási nyelvek összes funkcionalitásának megtartása mellett nyújt magas szintű grafikus fejlesztőeszközt mérnöki feladatok gyors megoldásához. A grafikus blokkokból felépülő LabVIEW alkalmazás egy blokkdiagramot és egy hozzá szorosan kapcsolódó egyéni kialakítású vezérlőpanelt tartalmaz, ami a felhasználó által egy palettából kiválasztott függvényekből összeállított algoritmust hajt végre.



1.ábra LabVIEW blokk diagram



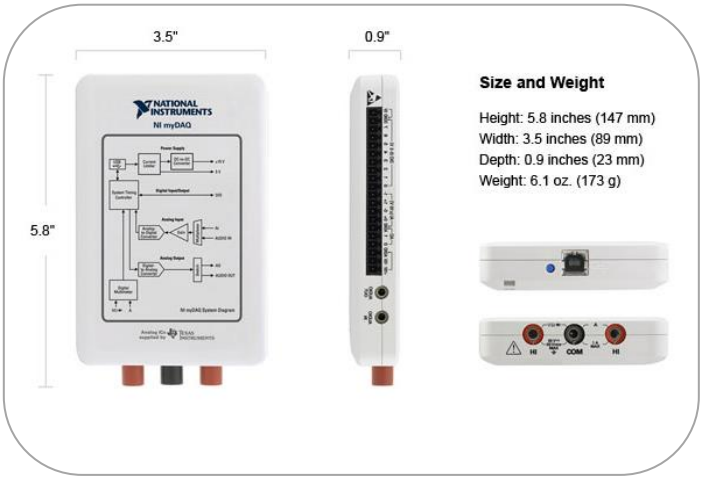
2.ábra LabVIEW felhasználói felület

A LabVIEW-t ma már mérnökök és kutatók a világ minden táján számtalan iparágban használják az autók, telekommunikációs eszközök, félvezetők tervezésén és gyártásán át egészen a biológiai, kémiai és fizikai kutatásokig. Az akadémia területén a LabVIEW grafikus fejlesztői környezet lehetőséget ad a hallgatóknak arra, hogy vizualizálják, majd végrehajtsák mérnöki koncepcióikat. A LabVIEW alkalmazása a tanórákon egy hatékony és dinamikus tanulási környezetet teremt, miközben az NI ELVIS használata során szabadon alakíthatják saját felhasználói felületüket.

A National Instruments minden segédeszközt és oktatási segédanyagot biztosít a hallgatóknak, hogy rövid időn belül magabiztosan használják a LabVIEW grafikus fejlesztőkörnyezet.

17.7. A National Instruments az oktatásban

Az NI elkötelezett annak tekintetében, hogy mind napjaink gyakorló mérnökei, mind a jövő mérnökgenerációjának számára elérhetővé tegye az általa fejlesztett eszközöket, továbbá támogatást nyújtsanak a mérnököknek, kutatóknak, oktatóknak és diákoknak a legújabb eszközök megismerésében, alkalmazásuk elsajátításában és ennek továbbfejlesztésében.



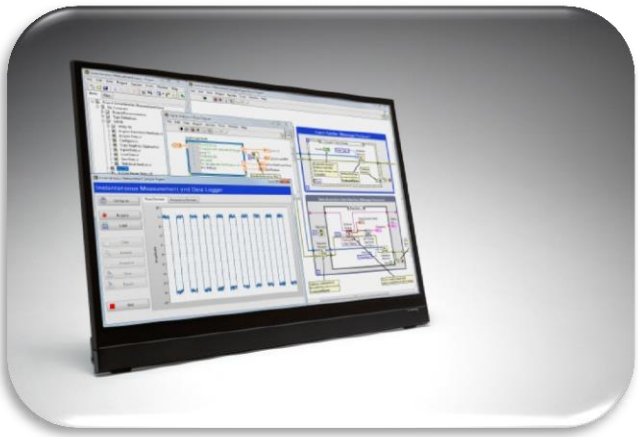
Az NI világszerte elhivatottan segíti a mérnöki és tudományos oktatást olyan szoftverekkel és hardverekkel, amelyekkel az oktatók és a hallgatók egyaránt összeköthetik elméleti tanulmányaikat valós mérésekkel, alkalmazásokkal.

Az NI kiemelt ügyként kezeli a felnövekvő generáció sorsát, a mérnöki pályát választó diákok számának csökkenő tendenciájára reagálva az eddigieknél is hangsúlyosabban tekint erre a területre. Az óvodától kezdve egészen az egyetemig kíséri a tanulókat, olyan eszközöket adva a kezükbe, melyek használatával mélyebben megismerkedhetnek a természettudományokkal, és ennek folyományaként később nagyobb eséllyel választják ezt a pályát.

17.8. NI myDAQ

A National Instruments munkásságát mind az oktatás, mind az innováció területén számos elnyert díj méltatja.

A myDAQ eszköz, mely egyszerre több tradicionális mérőműszer használatától is mentesít, hiszen egy hordozható, multifunkcionális mérő, adatgyűjtő és vezérlő eszköz, két kategóriában nyert innovációs díjat az elmúlt években: a Magyar Kereskedelmi és Iparkamara 2010. évi Innovációs Díját, illetve az Iparfejlesztési Közalapítvány 2010. évi Szervezeti Innovációs Díját.



Az NI myDAQ egy olyan termék, amely a hagyományos oktatási szemléletet maga mögött hagyva az egyéni, személyes labor szemléletet helyezi előtérbe. Az NI myDAQ termékével az elméleti és gyakorlati oktatás ugyanabban az időpontban megosztva történik, így a megszerzett tudás azonnal kipróbálható. A termék képes a legtöbb villamosmérnöki gyakorlatban számításba jöhető mérés elvégzésére. A termék a hagyományos oktatásban használt labor felszereléseket váltja ki, melynek eredményeképpen jelentős megtakarítás jelentkezik a felhasználó oktatási intézményeknél. A

műszerrel eredményesebb, gyakorlatorientáltabb az oktatás, miközben a hallgatóknak a termék elengedhetetlen eszközévé válik.

Az NI myDAQ műszert laboron kívüli gyakorlati kísérletezésre tervezték. A hordozható műszer széleskörű szolgáltatásokkal rendelkezik. Az NI myDAQ valós fejlesztéseket tesz lehetővé, és a diákok - NI LabVIEW és Multisim használatával – rendszerek prototípusait készíthetik el, áramköri analíziseket végezhetnek az előadáson és a laboron kívül. További részletekért látogasson el a www.ni.com/mydaq oldalra!

Tulajdonságok

- Nyolc az egyben: egyetlen eszköz nyolc általános célú, számítógép-alapú Plug-and-Play laborműszert helyettesít
- Az NI myDAQ segítségével a számítógép mérőműszerré változtatható. Az NI ELVISmx hardver driver nyolc egyszerű műszerfunkciót biztosít.
- National Instruments technika csúcsminőségű Texas Instruments analóg integrált áramkörökkel (IC)
- Az eszköz rendszerdiagramja és az NI myDAQ felhasználói útmutatójában szereplő Texas Instruments IC-k listája megmutatja, hogyan alkalmazták a TI adatkonvertereket, erősítőket, interfész és energiaellátás vezérlő áramköröket az NI myDAQ tervezésekor.
- Bárhol és bármikor használható, kompakt és hordozható
- Az NI myDAQ elég kicsi ahhoz, hogy elférjen egy hátizsák zsebében és teljes egészében USB-s tápellátásról működik, így egy laptop segítségével bárhol használható.
- Funkcionalitása kiterjeszthető a LabVIEW és Multisim segítségével
- A házi feladatok megoldásai valódi jelekkel ellenőrizhetőek, Multisim használatával áramköri szimulációhoz és LabVIEW-val a mérések automatizálásához és jelfeldolgozáshoz.
- iPod-kompatibilis 3.5mm-es audio bemenet és kimenet, audio jelek keveréséhez és manipulásához
- Az audio jelek szoftver-műszerek használatával analizálhatóak, vagy manipulálhatóak LabVIEW segítségével, saját grafikus hangszínszabályozó, karaoke gép, stb. létrehozásához.

Specifikáció

- Két differenciális analóg bemeneti és analóg kimeneti csatorna (200 kS/s, 16 bit, +/- 10 Volt)
- Az analóg be- és kimeneti csatornákhoz egyeztetett hozzáférés +/- 10 Volt tartományban a csavaros csatlakozókról vagy +/- 2 Volt tartományban a 3,5mm-es jack csatlakozókon keresztül.
- +5 , +15, and -15 Volt táp kimenetek (maximum 500 mWatt teljesítményig)
- USB-s tápellátású, hogy maximálisan hordozható legyen. A myDAQ elegendő tápellátást biztosít egyszerű áramkörök és szenzorok számára.
- Nyolc digitális be- és kimeneti vonal (3.3 Volt TTL-kompatibilis)
- A szoftveres időzítésű digitális vonalak használhatóak mind alacsony feszültségű TTL (LVTTL), mind 5 Volt TTL digitális áramkörökkel. A vonalak külön-külön bemeneti vagy kimeneti állapotba állíthatóak.

- 60 Voltos digtális multiméter (DMM) feszültség, áram- és ellenállásmérésekhez
- Az izolált DMM AC, DC feszültség és áram, valamint ellenállás, dióda feszültség- és szakadásmérési képességekkel rendelkezik.
- Újrahasználható tároló doboz és rekesz, DMM mérőfejek és audio kábel
- A tároló doboz és rekesz kényelmes tárolást biztosítanak az NI myDAQ műszer és a mellékelt USB kábel, DMM mérőfejek, audio kábel, valamint a saját alkatrészek és tartozékok számára.

17.9. NI myRIO

A National Instruments által fejlesztett NI myRIO beágyazott hardvereszközzel a diákok gyorsabban és költséghatékonyabban tervezhetnek valódi, összetett mérnöki rendszereket, mint valaha.

Az NI myRIO a népszerű NI CompactRIO platform nagyteljesítményű technológiáján alapul, de kisebb és diák-közelibb, mint az iparban használt megfelelője. Az NI myRIO-ban megtalálható Zynq® a Xilinx legújabb, teljes körűen programozható egychipes rendszere (SoC) kétmagos ARM Cortex-A9 processzorral és 28000 programozható logikai egységes FPGA-val. Az NI LabVIEW grafikus programozási környezet segítségével a diákok rendkívül gyorsan programozhatják az FPGA-t, rugalmas prototípusfejlesztési és gyors tervezési iterációkkal tökéletesítve rendszerüket.

Nick Morozovsky, a Kaliforniai Egyetem (San Diego) végzős diák-kutatója elmondta: *"Az NI myRIO az FPGA számítási kapacitását és rugalmasságát párosítja össze, mindezt teszi egy rendkívül praktikus méretben, melynek következtében az NI myRIO a beágyazott robotikai alkalmazások ideális vezérlőegységévé válik".*

Az NI myRIO-n emellett található 10 analóg bemenet, 6 analóg kimenet, hang be- és kimeneti csatorna, valamint akár 40 digitális be- és kimeneti vonal is. A tartós, zárt készülékházban beépített WiFi-t, háromtengelyes gyorsulásérzékelőt és több programozható LED-et is találhatunk.

"Ha fel kellett volna sorolnom egy hordozható be-kimeneti eszköz általam elvárt tulajdonságait, a lista szinte teljesen egybevágott volna az NI myRIO adatlapjával" — nyilatkozta III. Dan Dickrell, a Floridai Egyetem mérnökoktatója. "Ez a parányi eszköz kiemelkedő mérnöki alkotás."

A myRIO-val kiegészített LabVIEW újrakonfigurálható be-kimeneti (RIO) architektúra tovább bővíti az NI által különböző tudásszintekhez kínált megoldások körét, legyen szó akár a mérnöki alapokkal ismerkedő diákok vagy a világ legbonyolultabb rendszereit fejlesztő mérnökök szintjéről. Az egyszerű tantermi és laboratóriumi felhasználhatóság érdekében az NI myRIO-hoz ingyenes oktatóanyagok tölthetők le, továbbá az eszköz együttműködik valamennyi NI miniSystems kiegészítővel és számos, harmadik fél által előállított érzékelővel, illetve beavatkozóval. Az NI myRIO széles körű, hardver ökoszisztémáján túl fontos szempont, hogy az eszköz többféle környezetből is programozható, beleértve a LabVIEW-t és C/C++-t is. Az eszközt az oktatók így könnyedén beilleszthetik a már kidolgozott vezérlést, robotikát, mechatronikát és beágyazott rendszereket tárgyaló tanmenetükbe.

"Küldetésünknek érezzük annak támogatását, hogy a diákok tanulmányaik során is ugyanahhoz a technológiához férjenek hozzá, amelyet később a munkájuk során is használni fognak" — nyilatkozta Dave Wilson, az NI globális akadémiai marketing igazgatója. "Mind a diákok, mind a leendő munkaadóik számára biztosítani szeretnénk, hogy együttműködésük kezdetétől készek legyenek az innovációra."

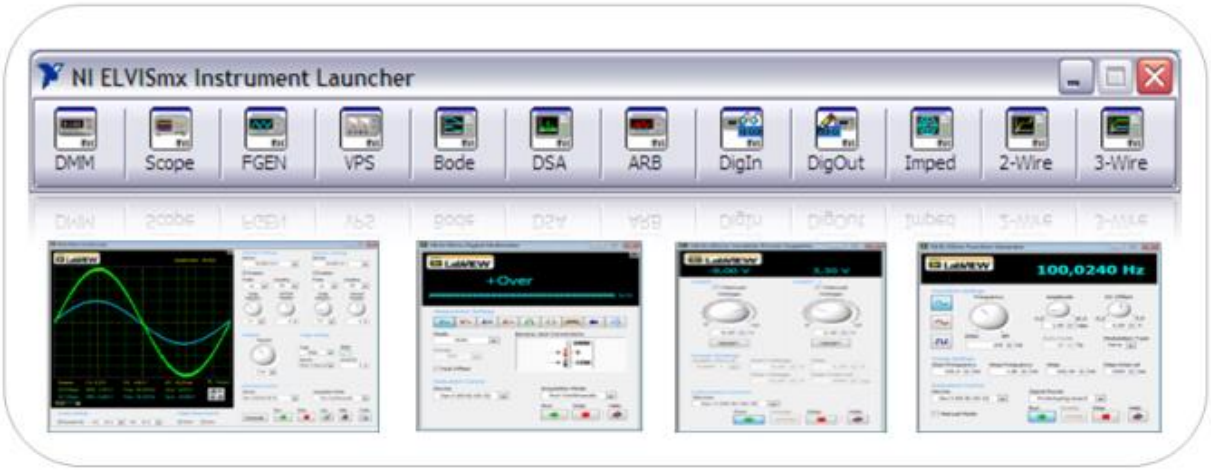
Az NI myRIO-hoz kapcsolódó részletes információk az ni.com/myrio weboldalon találhatók.

17.10. NI ELVIS II

A National Instruments által fejlesztett és gyártott **NI ELVIS II** (**E**ducational **L**aboratory **V**irtual **I**nstrumentation **S**uite) termékcsomag a piacon egyedülálló, kifejezetten a fizika és a műszaki tantárgyak gyakorlati oktatására tervezett számítógép alapú tervező, és prototípus készítő munkaállomás.

A kifejezetten oktatási célokra fejlesztett termék egy eszközben egyesíti az összes olyan méréstechnikai funkciót, amely egy méréstechnikai laboratóriumban elengedhetetlenül szükséges. Az NI ELVIS egy multidiszciplináris, számítógép alapú oktatóeszköz, amely az alábbi területek gyakorlati oktatására használható:

- Elektronika/elektrotechnika
- Áramkör tervezés és szimuláció
- Mechatronika/robotika
- Méréstechnika
- Automatizálás
- Beágyazott rendszerek
- Vezérlés és szabályozás
- Érzékelők és szenzorok
- Megújuló energia
- Telekommunikáció
- Bioelektronika
- Fizika



Az NI ELVIS rendszere a képernyőn megjelenő – a LabVIEW grafikus fejlesztői környezetben gyárilag elkészített - virtuális műszer felületét kapcsolja össze az adatgyűjtő eszköz valós funkcióival, és a próbapanel megfelelő csatlakozásaival. Az NI ELVIS alkalmazásával lecsökkenthetjük a laboratóriumok felszerelésének költségeit, mivel minden olyan eszközt tartalmaz, amelyre az oktatólaborokban

szükségünk lehet. A rendszer egy költséghatékony elektronikai mérőlaboratórium kialakítását teszi lehetővé, mivel az eszköz 12 különböző, nagy értékű mérőműszert tartalmaz:

- Digitális multiméter
- Oszcilloszkóp
- Függvénygenerátor
- Szabályozható tápegység
- Bode analizátor
- Dinamikus jel-analizátor, spektrum-analizátor
- Hullámforma-generátor (+ hullámforma szerkesztő)
- Digitális jelgenerátor
- Digitális vonal író
- Digitális vonal olvasó
- Impedancia analizátor
- 2 vezetékes áram-feszültség analizátor, 3 vezetékes áram-feszültség analizátor

Az eszköz a virtuális és valós világ látványos összekapcsolásával felkelti a fiatal hallgatók érdeklődését a mérnöki tudományok iránt. Az NI ELVIS munkaállomás a cserélhető kiegészítő paneljeinek segítségével számos témakör gyakorlati oktatására alkalmas: a megújuló energia forrásoktól egészen a telekommunikációig.

Az eszköz a LabVIEW grafikus fejlesztőkörnyezettel és a Multisim áramkör tervező és szimuláló szoftverrel együtt, teljes körű adatgyűjtést és prototípus kifejlesztést tesz lehetővé, ideális eszközt jelent egyszerűbb és bonyolult oktatási feladatok megoldásához.

További információ az NI ELVIS-ről a http://www.ni.com/nielvis oldalon található.

17.11. A National Instruments és a LEGO ® együttműködése

Globális szinten bizonyított tény, hogy a diákok napjainkban csökkenő arányban választanak természettudományos pályát és terveznek mérnöki útra lépni. A tudomány, technológia, mérnöki pálya és a matematika, tehát a természettudományok iránti érdeklődés hiánya már jóval azelőtt jelentkezik, hogy a diákok belépnének az egyetemre. Amikor a gyerekek jelentősen több időt töltenek az elméleti tanulással, mint a gyakorlati feladatmegoldással, ami a mai természettudományos oktatást jellemzi, gyakran elvesztik a lelkesedésüket a téma iránt vagy esetleg egyáltalán nem is lesz lehetőségük rá, hogy elkötelezetté váljanak iránta.

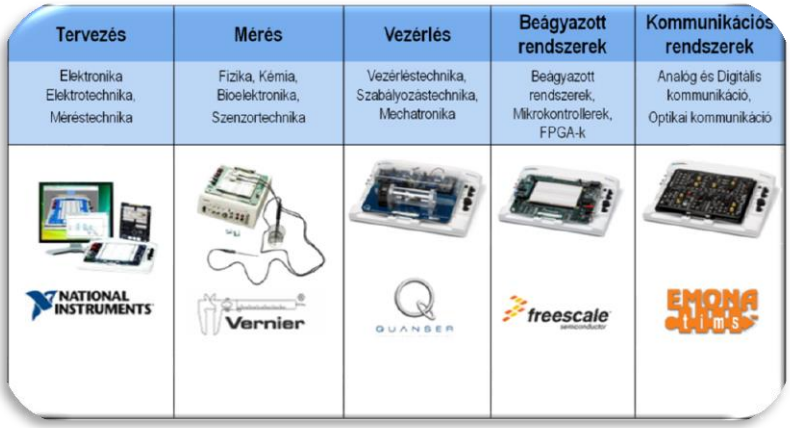
A National Instruments és a LEGO® már több mind 2 évtizede dolgozik együtt azzal a céllal, hogy közös munkával olyan interaktív technológiát fejlesszen, amely ötvözi a gyerekeknek örömöt okozó játékot a professzionális mérnöki eszközökkel, ezáltal pedig fejlődéshez segítse azt a koncepciót, amely mentén ma a természettudományok oktatása történik. A mérnöki feladatok nem csupán elméleti megközelítése, hanem gyakorlati kivitelezése a diákokat már 6 éves korban lenyűgözi és képes érdeklődésüket fenntartani. A gyakorlati feladatmegoldásban való részvétel folyamán a diákok a természettudományok alapvető koncepcióját tanulják meg, amely kritikus jelentőséggel bír gyakorló mérnökként elérhető sikereik kapcsán.

A National Instruments olyan eszközöket fejleszt a mérnökök és tudósok számára az ipar szinte minden területére kiterjedve, mely elősegíti a termelékenység, az innováció és a tudományos felfedezések fejlődését. Az NI továbbá olyan LabVIEW alapú oktatási eszközöket is kínál, amelyek segítenek az oktatási intézményekben áthidalni az elmélet és gyakorlat között húzódó szakadékot. Ez a jelenség olyan egyedi pozícióba helyezi az NI-t, melyben lehetősége nyílik a gyerekeknek alapvető mérnöki koncepciókat megtanítani ipari sztenderd eszközök használatával már fiatal korban.

A LEGO® Education alapvető célja, hogy a gyerekeket kreatív gondolkodásra, rendszerezett érvelésre és a bennük rejlő lehetőségek kiaknázására inspirálja, hogy ez által maguk alakíthassák saját jövőjüket. Amikor a LEGO® Education a 2000-es évek elején technológiai partnert keresett, akivel együttműködve kifejlesztheti a programozható robotika platformjához szükséges szoftvert, az NI ebbe a koncepcióba egyértelműen beleillett. Az együttműködés a high-tech eszközök és a természettudomány oktatásának tökéletes kombinációját hozta létre. A gyerekek ösztönösen vonzódnak a LEGO® elemekkel való gyakorlati interakcióhoz. Ennek a jelenségnek a párosítása az ipar területén is releváns technológiával úttörő jelentőségű, mert lehetővé teszi, hogy ez az ösztönös játékszeretet a természettudományok elsajátítása során tovább fejlődjön és egyfajta természettudományos műveltséggé nője ki magát. A két vállalat együttműködése tehát létrejött és sikeresen működik már közel egy évtizede, amióta folyamatosan olyan oktatási megoldások fejlesztésén dolgoznak, melyek magukkal ragadják a diákokat. Olyan robotikai platformokat hoznak létre, melyek gyakorlatilag rendszert teremtenek a feladatok szisztematikus és kreatív módon történő megoldásához.

A partnerség egyik legfontosabb aspektusa a LabVIEW alapú oktatási eszközök biztosítása, amelynek különleges tulajdonsága, hogy az óvodától egészen a világ legösszetettebb mérnöki kihívásaiig terjedően megoldást nyújt az adott problémára. A LabVIEW grafikus programozási képességei egy olyan intuitív felhasználói felületet nyújtanak a felhasználóknak, amely ideális a mérnöki koncepciók fiatal diákokkal való megismertetésére. A szoftver azon tulajdonsága, mely szerint egy folytonos és széles skálán, az oktatástól az iparig minden felhasználó számára jól alkalmazható, lehetővé teszi a diákok számára, hogy minden egyes alkalmazás eredeti céljára koncentráljanak anélkül, hogy folyamatosan újabb és újabb, egyre bonyolultabb eszközök használatát kelljen megtanulniuk, ahogy ők maguk egyre több tudás birtokába kerülnek.

A LEGO® Education és az NI azt a lehetőséget biztosítja a diákoknak, hogy alkossanak, építsenek, hibázzanak, majd újra próbálkozzanak, vagyis egy problémamegoldásra orientált megközelítést, amely a hagyományos tantermi oktatásban gyakran nem kap elég hangsúlyt. Ezek a platformok lehetőségeket teremtenek a diákok számára, hogy elsajátíthassák azokat a képességeket, amelyek a mai és jövőbeni társadalom legösszetettebb kihívásainak megoldásában segíthetik őket.



A LEGO® MINDSTORMS® az a termék, melynek használatában, az NI LabVIEW szoftverrel való programozása során a LEGO® és az NI együttműködése testet ölt. A robot az építőjáték változatosságát a legmodernebb technológiával ötvözi, melynek révén a felhasználók kedvük szerint építhetik meg és programozhatják a robotokat, hogy azok mozogjanak, beszéljenek, akár gondolkozzanak, mindezt a felhasználó kreativitására és ügyességére bízva.

Az NI kifejezetten a robot programozására kifejlesztette a LabVIEW szoftvernek egy egyedi, oktatás célokat szolgáló verzióját. A LabVIEW for LEGO® MINDSTORMS® a LabVIEW szoftvernek egy egyedi

verziója, amelyet kifejezetten a LEGO® Education robotikai platformjával való használatra terveztek, hogy segítségével egy olyan kifinomult oktatási eszköz álljon a diákok rendelkezésére, amely segíti őket a robot programozásában. A szoftver egy olyan oktatási eszköz, amely a robot irányításában és programozásában segíti a diákokat. A LabVIEW szoftver használatával a robot egy valódi tudományos és mérnöki állomásként funkcionál, mely felkészíti a diákokat felsőfokú tanulmányaikra, illetve a mérnöki pályára, melynek során szintén találkozni fognak a LabVIEW szoftverrel

LabVIEW szoftver legújabb moduljával, a LabVIEW Module for LEGO® MINDSTORMS®-zal a felhasználók a robot legújabb verzióját, a LEGO® MINDSTORMS® EV3® robotikai platformot tudják programozni. A modul segíti a diákokat, hogy olyan programokat hozhassanak létre, melyek kommunikálnak a robottal, illetve irányítani is tudják azt. Mivel a LabVIEW érzékelők, műszerek és rendszerek ezreivel áll kapcsolatban, a felhasználók bármilyen szintű mérnöki ismeret birtokában képesek vele rövid idő alatt összetett robotikai projektek tervezésére. Ez teszi a platformot ideális eszközévé a természettudományos oktatásnak.

17.12. Elérhetőségek

National Instruments Hungary Kft.
1117 Budapest, Neumann János u. 1. E. ép. | hungary.ni.com | ni.hungary@ni.com | Telefon: +36 1 481 1400 |

NI Hungary Kft.
4031 Debrecen, Határ út 1/A. | hungary.ni.com/debrecen | Telefon: +36 52 515 400 |

H-Didakt Kft.
1162 Budapest, Délceg u. 32. | http://www.hdidakt.hu | info@hdidakt.hu | Telefon: +36 (30) 3744-500; +36 (30) 4609-638 | Fax: +36 (1) 409-0765