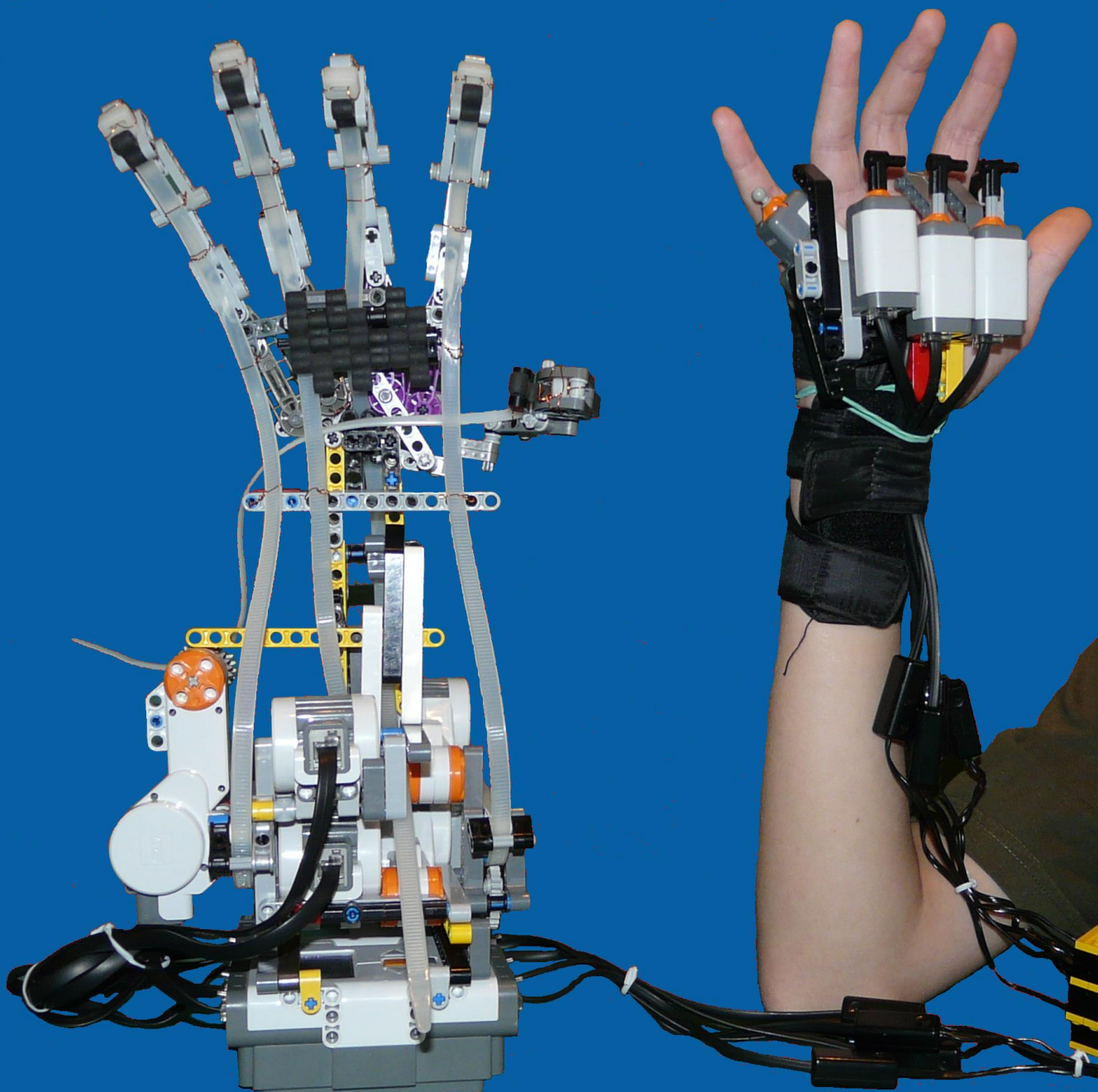
Kiss Róbert - Badó Zsolt

Egyszerű robotika

A Mindstorms NXT robotok programozásának alapjai



Kiss Róbert – Badó Zsolt

Egyszerű robotika

A MINDSTORMS® NXT robotok programozásának alapjai

A könyv elektronikus változatának kiadása a

National Instruments Hungary Kft jóvoltából jöhetett létre.

A letöltési cím: http://www.amcham.hu/download/002/556/Robotkonyv_KR_BZS.pdf

2010

Szerzők: Kiss Róbert, Badó Zsolt
Kiadás éve: 2010
Szerkesztette: Kiss Róbert
Szakmai lektor: Pásztor Attila
Nyelvhelyességi lektor: Jankay Éva

A címmel ellátott képeken szereplő robotkonstrukciók Kasza Dániel, Kiss Ádám, Kiss Máté és Varga György munkái.

TARTALOMJEGYZÉK

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 nyolc fejezete a programozási és robotvezérlési alapok ismertetését tartalmazza. A további fejezetek a haladó programozáshoz adnak ötleteket.

Kecskemét, 2010.

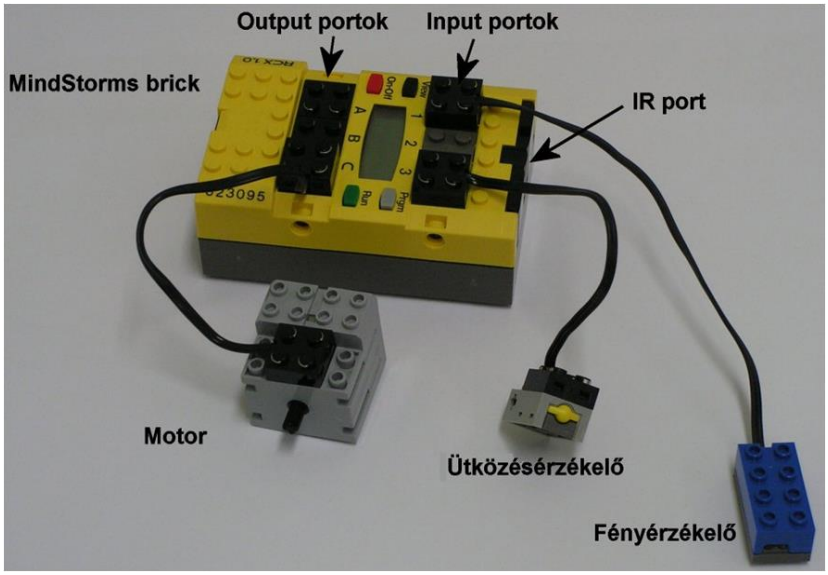
A szerzők

1. ROBOTHARDVER

1.1. Történeti bevezető

A LEGO® Csoport 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.

A koncepció második generációja az 1998-ban megjelent MINDSTORMS® RCX fantázianevű robot.
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, elfordulásérzékelő.



Az NXT a LEGO® Csoport legújabb, 3. generációs programozható robotkészlete, amelyet 2006-ban mutatott 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, amely NXT 1.0 továbbfejlesztése. Elsősorban a szenzorok köre és a keretprogram bővült. A terméknek létezik oktatási és kereskedelmi változata is. A lényeges elemeket tekintve nincs különbség a két változat között. Sok cég meglátta a lehetőséget az alkalmazásban így mások is gyártanak különböző szenzorokat, amelyek illeszthetők a robothoz.

A könyv programjait a MINDSTORMS® NXT 2.0 robottal teszteltük, de változtatás nélkül használhatók az 1.0-ás változaton is.

1.2. A „intelligens” tégla és egyéb hardver elemek

A MINDSTORMS® robotkészlet „agya” egy intelligens, programozható tégla (brick), melynek négy bemenetére szenzorok (érzékelők), kimeneteire pedig szervomotorok csatlakoztathatók. Az eredmények megjelenítésére egy 100 x 64 képpontos LCD grafikus kijelző szolgál.

A tégla agya egy 32 bites ARM7 mikrovezérlő. A miniszámítógép tartalmaz még 64 Kbyte RAM-ot, 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.

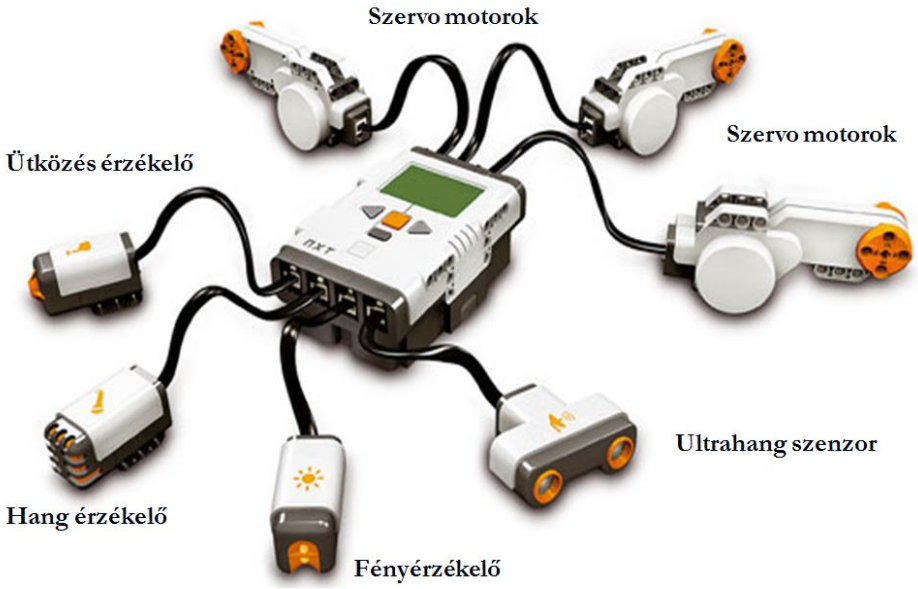
Természetesen az intelligens tégla a készletben található LEGO alkatrészeken kívül bármely egyéb LEGO kompatibilis alkotórésszel összeépíthető, így a konstrukciók bonyolultságának csak a kreativitás hiánya szab határt.

A tégla négy bemeneti portjára szenzorok csatlakoztathatók, amelyek a környezet különböző mérhető értékeiről tudnak adatokat továbbítani a robot agya és a szoftver felé.

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 C-ig betűzték a tégla felső részén. Itt kapott helyet az USB csatlakozási pont is, amelynek segítségével pl. számítógéphez csatlakoztatható, és a programok feltöltésére használható.

A működés elve 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 vagy bluetoothon keresztül a robotra töltve az már önállóan viselkedik a programja utasításai alapján.

A bluetooth-os 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.

1.3. Input eszközök: alapszenzorok

Az eszközhöz több cég is gyárt szenzorokat. Az NXT 2.0-ás készletben három különböző szenzor található: 1 db ultrahangos távolságérzékelő, 1 db színszenzor és 2 db ütközésérzékelő (az NXT 1.0-ás készletben a színszenzor helyett még 1 db fényszenzor és 1 db hangszenzor volt).

Ütközésérzékelő

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.

Fényérzékelő

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. Ez persze függ a megvilágított felület színétől is. A szenzor a programkörnyezetben egy 0-100 közötti értéket szolgáltat a felülettől függően.

Távolságérzékelő

Az érzékelőt *ultrasonic* szenzornak is nevezik. Az ultrahangos távolságérzékelő a távolságot centiméterben vagy hüvelykben méri, 0 – 250 cm tartományban, +/-3 cm pontossággal (gyakorlati tapasztalatként megállapítható, hogy 5 cm-en belüli és 150 cm fölötti értékeket már kevésbé méri pontosan). 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. Több cég is gyárt különböző értéktartományban pontosabban mérő szenzorokat, amelyek csatlakoztathatók a téglához.

Hangérzékelő

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 csak az emberi fül által is hallható hangokat méri, 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.

Színszenzor (az NXT 2.0-ás készlet tartalmazza)

Valódi színlátást biztosít a robot számára. Az érzékelt színinformációk alapján annak RGB (vörös-zöld-kék) összetevőit is képes meghatározni és visszaadni. Több különböző változata is forgalomban van. Az NXT 1.0-ás és az NXT 2.0-ás robothoz különböző változat készült. A programkörnyezetben beállítható,

hogy fényszenzorként működjön. Az alapértelmezett vörös színű megvilágítástól eltérően használhatunk zöld vagy kék színű fényt is.

1.4. Input eszközök: egyéb szenzorok

Néhány fejlesztő nem elégedett meg a készlethez gyártott szenzorokkal, s egyedi fejlesztésű érzékelők gyártásába kezdett. Ezek közül mutatunk be néhányat.

Iránytű

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

Gyorsulásmérő

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 kommunikációs jeladó

Alkalmassá teszi robotot az infrakommunikációra.

Infra labda és infra kereső

Elsősorban a robot foci számára fejlesztett eszközök. A labda jó közelítéssel gömbszimmetrikusan sugároz infra jeleket néhány 10 cm-es hatótávolsággal. Az infra kereső kb. 130 fokos látószöggel tudja meghatározni és a keretprogram számára visszaadni a labda irányát és távolságát.

Gyroszenzor

Gyakorlatilag egy giroszkóp, amelynek segítségével a robot egyensúlyozását lehet javítani, korrigálni.

Hőmérsékletmérő

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.

Szolár panel

Napelem, amely egy 60 W-os izzóval közvetlenül megvilágítva 25 cm távolságból, 5 V 4 mA áramot képes előállítani. Így alkalmas pl. ledek közvetlen működtetésére, vagy a megújuló energiaforrások használatának bemutatására.

Elektorméter és energiatároló egység

A megújuló energiák készlet részeként egy egyszerű elektromos mérőműszer, amely képes feszültség (Volt), áramerősség (Amper), és teljesítmény (Watt) mérésére, valamint a hozzá csatlakoztatható energiatároló egység révén a tárolt energia számszerű (Joule) megjelenítésére. Az NXT téglához csatlakoztatva az adatok átvitele lehetséges. Egy bemeneti és két kimeneti csatlakozóportja van.

Elektromotor, dinamó

A megújuló energiák készlet részeként elsősorban a szélturbina energiatermelő egységeként használható. Maximálisan 800 fordulat/perc sebességre képes.

1.5. Output eszközök: szervomotorok

Az NXT MINDSTORMS® készlet része a három 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).

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 csatlakozókábelen keresztül illeszthetők a téglához, amely a gyári készletben három különböző hosszméretben található meg.

A speciális eszközökhöz további két kábeltípus használható. Egyrészt a régebbi típusú RCX robot szenzorainak csatolására, másrészt egyéb, nem kizárólag az NXT-hez fejlesztett eszközök és szenzorok illesztéséhez.

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® Csoport és a National Instruments közös fejlesztése, amely a LabWiev alapokra épülő grafikus NXT-G nyelv. 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 fényszenzor vagy 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.

Néhány bemutatott program esetén hangszenzort is használtunk, ezt a 2-es portra csatlakoztattuk. 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 Mindstroms NXT robotkészletben a csomag tartozékaként szerepel a működtetéshez, programozáshoz szükséges szoftver. A LEGO® Csoport és a National Instruments közös fejlesztése, amely a LabVIEW alapokhoz illeszkedően különösebb előképzettség nélkül lehetővé teszi a programozást. Az egyszerű, zömében egérhasználattal megoldható programírás ikonok egymás után illesztését és a megfelelő paraméterek beállítását jelenti. 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 a 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 lehetőségéről, így programozási kézikönyvként jól használható.

Az alábbiakban a keretprogram használatát, legfontosabb funkcióit és a 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 ismertetésére ugyanis nincs lehetőség. Az egyes programozástechnikai elemek használatának megértéséhez szükséges lehet a program helpjének használata és a forráskódok kipróbálása.

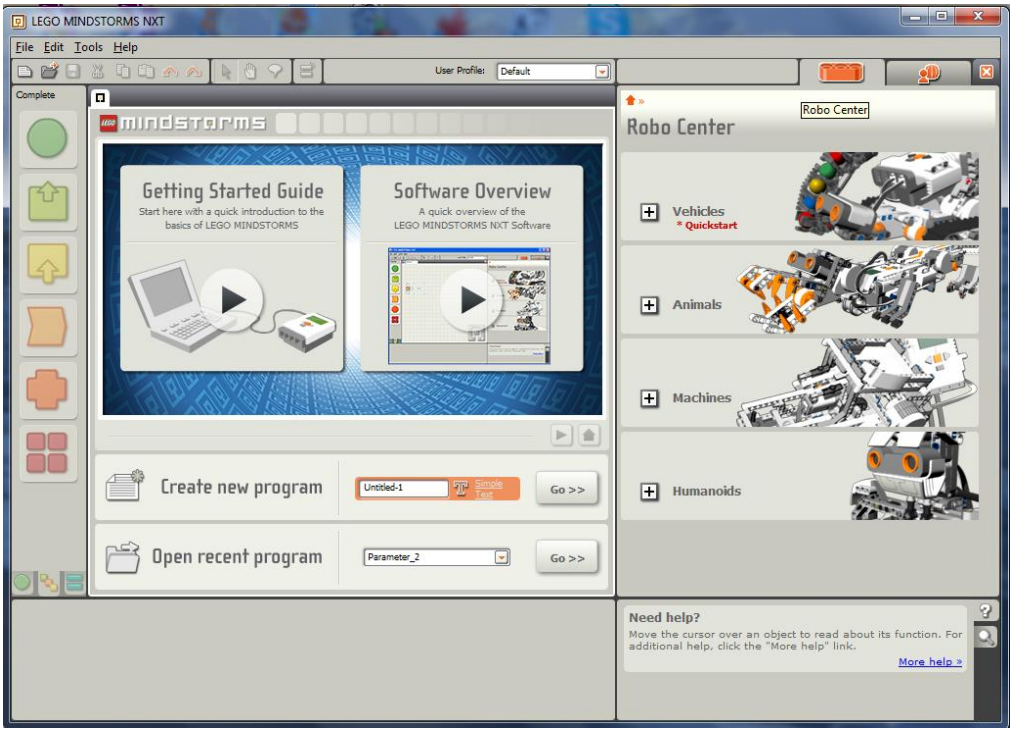
Az NXT-G-hez hasonló moduláris programnyelvek nagymértékben leegyszerűsítik a programírást, hiszen a szintaktikai (pl. gépelési) hibákkal az esetek jelentős részénél nem kell foglalkozni. Az egyes modulok paraméterezése különböző elektronikus űrlapokon használt beviteli eszközök segítségével történik (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 NXT-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 vagy programozási eszközök forráskódba építését.
A programkörnyezet használatával a programozásoktatás 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.

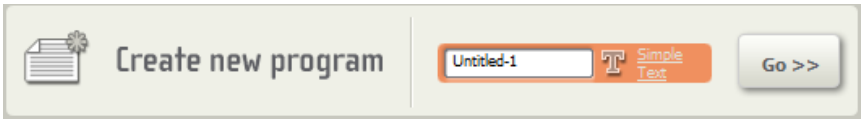
2.2. A használt programváltozat és összetevői

A bemutatott képek a LEGO® MINDSTORMS® 2.0 szoftverváltozatból származó képernyőprintek átszerkesztett változatai. A szoftver az 1.1-es változathoz nagyon hasonlít, de néhány eltérés azért megjelent. A különbségek elsősorban abból adódnak, hogy a szoftver a cég által korábban forgalmazott NXT 1-es robot továbbfejlesztett változatához készült, az NXT 2-eshez. A roboton működő saját „operációs rendszer” (*firmware*) több lényeges újítást is tartalmaz, amelyhez a programozói környezet is igazodik. Pl. a 2-es változatban már lehetőségünk van a lebegőpontos számok (tizedestörtek) kezelésére, míg az NXT 1-es változathoz készült szoftver esetén csak az egész számok voltak használhatók. Alapfelszereltségként jelent meg a hardverben a színszenzor, ami az NXT 1-es változatnál csak kiegészítőként volt beépíthető, így ezeket a változásokat is követi a szoftver.
Összességében azonban elmondható, hogy a szoftver lefelé kompatibilis. Az NXT 1-es robot hardverét ugyanúgy támogatja, mint az újabb változatot. Egyetlen feltétel, hogy a robotra a *LEGO MINDSTORMS NXT Firmware v1.2*-es változata vagy frissebb legyen feltöltve, mint operációs rendszer. Ezt a bemutatott szoftverrel egyszerűen meg tudjuk tenni, akár NXT 1-es, akár NXT 2-es robotkészlettel rendelkezünk.

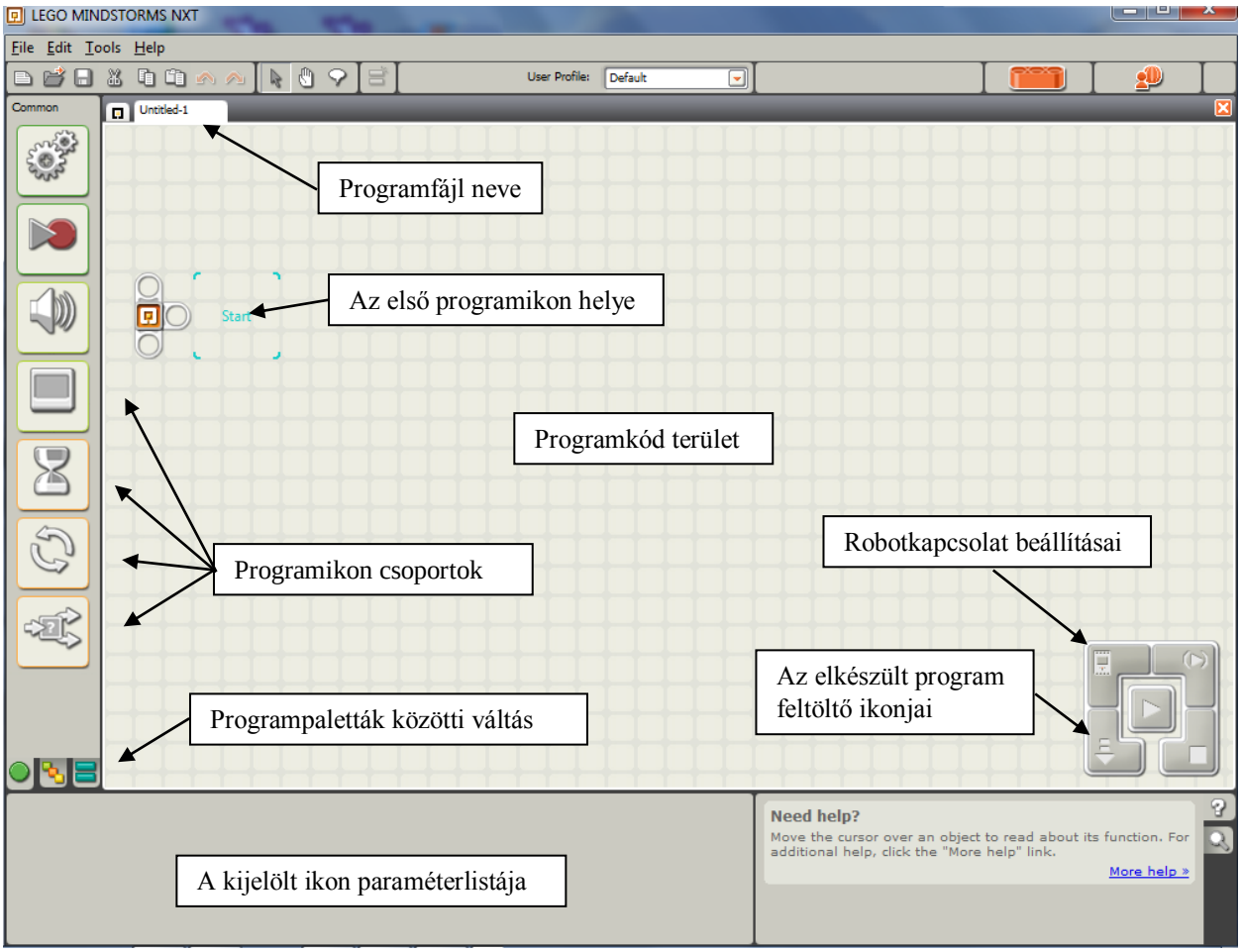


2.3. A programozási környezet alapelemei

A program indítása után a programozói felülethez a megfelelő gombon (*Go*) történő kattintással juthatunk.

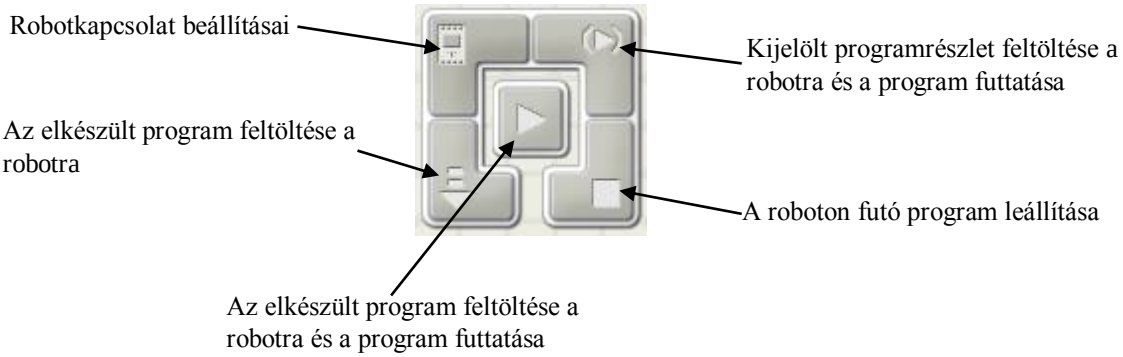


A megjelenő felület már közvetlenül alkalmas a programozás megkezdésére és az első program elkészítésére.



Első lépésként a számítógéphez csatlakoztatott robotot érdemes a rendszerrel megkerestetni. A kapcsolat USB kábelen vagy bluetooth-on keresztül valósítható meg.

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

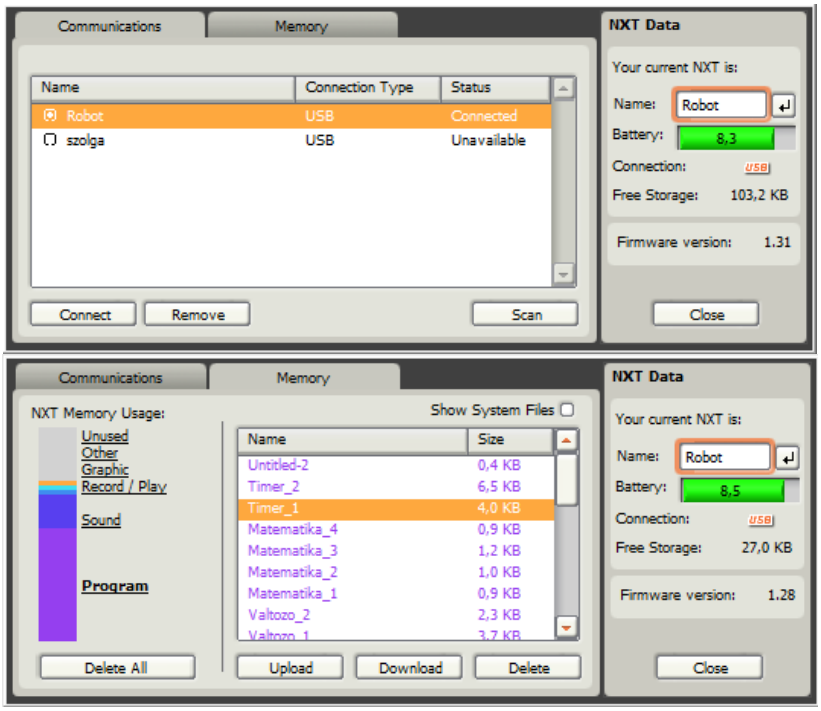


A robotkapcsolat beállításai ikonra kattintva a rendszer automatikusan megkeresi a *Scan* gombon történő kattintás után az elérhető eszközöket (ezeket meg is jegyzi, és később a listából elegendő választani).

A megtalált eszközök közül az *Available* feliratúak az aktuálisan elérhetők, ezek közül választva a *Connect* gombon kattintva történik meg a kapcsolat felépítése. Ha mindezt nem végezzük el, a kapcsolatot akkor is automatikusan felépíti a rendszer az első program robotra töltése során. A robotnak bekapcsolt állapotban kell lennie.

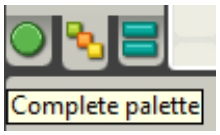
A kapcsolatot nem kell újra felépíteni, ha menet közben leválasztjuk az eszközt, de a programkörnyezetet nem zárjuk be.

A kapcsolódó eszköz esetén néhány információ rögtön a rendelkezésünkre áll. Pl. az elem/akku pillanatnyi töltöttsége, a robot neve (amelyet itt tudunk megváltoztatni) és a *Memory* lapon a roboton lévő fájlok listája (amelyeket pl. itt törölni is tudunk).



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 képernyő bal alsó részén tudunk az egyes programpaletták között váltani. Három ilyen paletta áll rendelkezésre. Az első a legfontosabb, leggyakrabban használt programmodulok ikonjai szerepelnek, a másodikon a teljes modulrendszer megtalálható, míg a harmadikon saját magunk által készített modulokat, rutinokat (függvényeket) tudunk a meglévő modulokból, programokból összeállítani és elérni.



Mivel a teljes programozási elemrendszer a második palettánál áll a rendelkezésre, ezért ezt mutatjuk be.

A képernyő bal oldali részén egy grafikus menü jelenik meg, amely funkciók szerinti csoportosításban tartalmazza a programírás során használható modulokat.

Gyakori programelemek kategóriája (*Common*)

Cselekvési programelemek kategóriája (*Action*)

Szenzorok kategóriája (*Sensor*)

Programstruktúrák kategóriája (*Flow*)

Adatmanipulációs elemek kategóriája (*Data*)

Haladó programelemek kategóriája (*Advanced*)

Az egyes elemekre kattintva egy jobbra gördülő menüből választhatjuk ki a szükséges ikont. A programelemekre (menüpontokra) programcsoport néven fogunk hivatkozni, megadva az angol elnevezését. A programcsoporton belüli ikonokat sokszor moduloknak fogjuk hívni.

A keretprogramban található hasznos funkció a szöveges megjegyzések, kommentek elhelyezésének lehetősége. A későbbi programértelmezést nagyban megkönnyíti, ha ilyen címkékkel látjuk el utasításainkat. Az ikonsoron található megfelelő ikonra kattintva lehet váltani a programszerkesztő mód és a megjegyzésszerkesztő mód között.

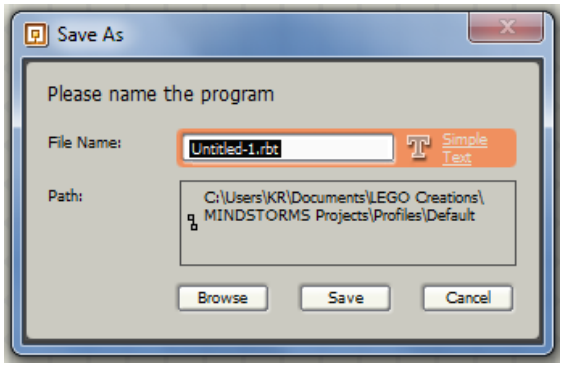
Programszerkesztő mód

Navigáló mód

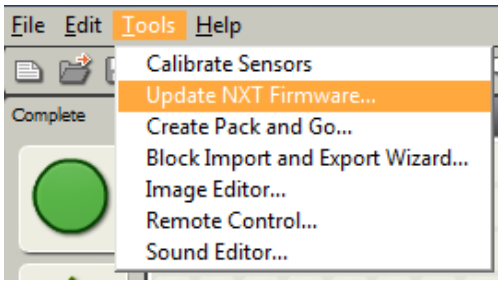
Megjegyzésszerkesztő mód

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.

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

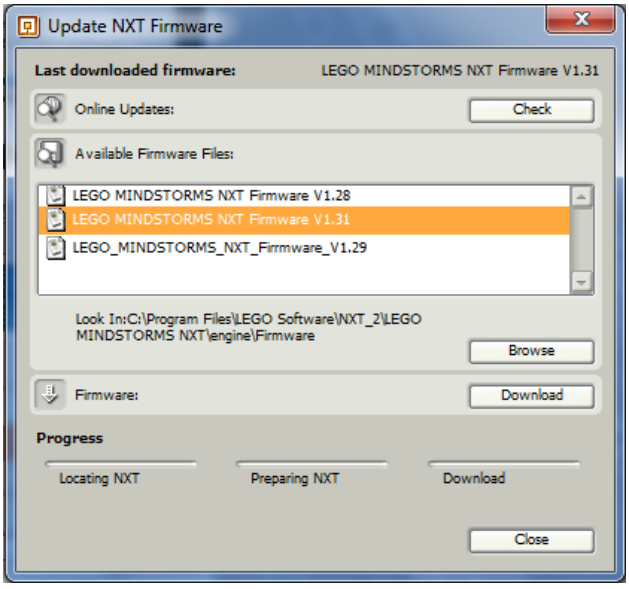


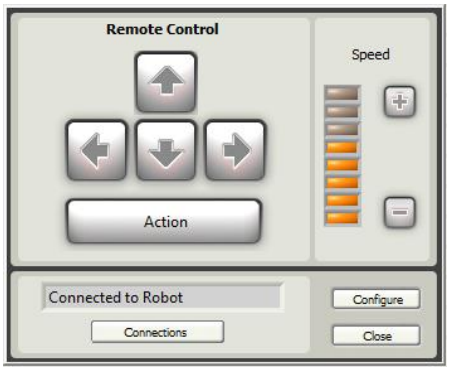
A *Browse* feliratú gombra kattintva lehetőségünk van új mentési mappát választani, ha a beállított alapértelmezés nem felel meg. A programjaink *rbt* kiterjesztést kapnak, és csak a keretprogrammal nyithatók meg.



További lehetőségként rendelkezésünkre áll a téglán elhelyezett *firmware* frissítése (akár interneten keresztül), vagy letöltött új kiegészítő modulok programba importálása. Mindezt a *Tools* menün keresztül tudjuk megtenni.

A *firmware* a robot operációs rendszerének tekinthető. Ez szükséges ahhoz, hogy a megírt és a robotra töltött programjaink utasításait megértse, és végre tudja hajtani. Jelen könyv megírásakor a *LEGO MINDSTORMS NXT Firmware v1.31* változatot használtuk. Érdemes a legfrissebb *firmware* változatot választani, hiszen ettől függ, hogy milyen utasításokat ért meg a robot és azokat hogyan hajtja végre.





A *Tools* menüben található a *Remote Control…* menüpont is, amely a robot közvetlen, távvezérelt irányítását szolgálja. Itt nem írunk programot, amelyet a robot végrehajtana, hanem távvezérléssel működtetjük. Érdemes kipróbálni, egyszerűen használható, és közvetlen, gyors élményt nyújt.

A *Tools* menü további lehetőségeivel a későbbiekben részletesebben foglalkozunk. Egyetlen hasznos eszközt mutatunk még be. Mivel a megírt programjaink viszonylag nagy méretűek, ezért szállításuk egyik gépről a másikra (pl. e-mailben) lassú is lehet. A *Tools* menü *Create Pack and Go…* menüpontján keresztül egy egyszerűen használható tömörítő eszköz áll a rendelkezésünkre. Az elkészült programjainkat kb. 90%-osan tömöríti, és *rbtx* kiterjesztésű állományként menti a megadott helyre. Így könnyebben szállíthatók. A tömörített állomány a keretprogrammal nyitható meg és menthető el ismét, kicsomagolt *rbt* formátumban.

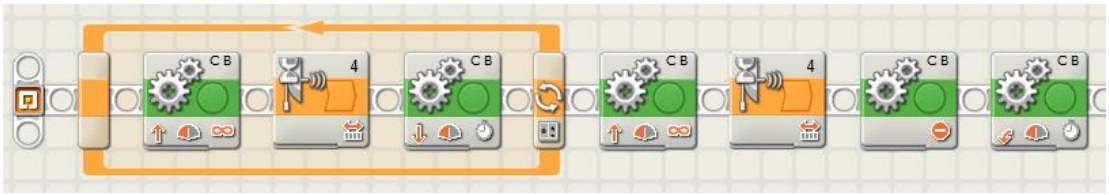
2.4. Programírás, első lépések

A programírás (bármilyen programnyelven) azt jelenti, hogy utasítások egymásutánját 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 NXT-G 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 egy lego rudat szimbolizáló vonalra. A legtöbb ikonnak van egy paraméterlistája, amely a szerkesztőprogram bal alsó részén jelenik meg. Itt állíthatjuk be az ikonnal szimbolizált utasítás tulajdonságait. (Pl.: milyen gyorsan forogjon a motor, mennyi ideig működjön, stb.)

A számítógépen így összeállított programot kell a robotra feltölteni. A feltöltés közben megtörténik a program „robotnyelvre” fordítása, így az utasítások értelmezhetők lesznek a robot számára is. A feltöltés történhet kábeles vagy bluetoothos 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 (forráskód formában) a számítógépre.

Az alábbi ábra egy elkészített program részletét mutatja.



Az utolsó ikon például a B és C portra kötött motorok mozgását vezérli, tehát az általa szimbolizált utasítás hatására a robot saját tengelye körül fog fordulni 50-es sebességgel 0,6 másodpercig.
A mozgás tulajdonságait az ikonhoz tartozó paraméterlistán állítottuk be.

Az NXT-G nyelvű programírás tehát 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.
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. Ilyen pl. a gyorsulásmérőhöz vagy iránytűhöz tartozó modul. Ha rendelkezünk a robothoz ilyen eszközökkel, akkor ezekre a modulokra szükségünk lesz. Egy letöltött modult a *Tools* menü *Block Import and Export 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 új beállításokat részletezzük.

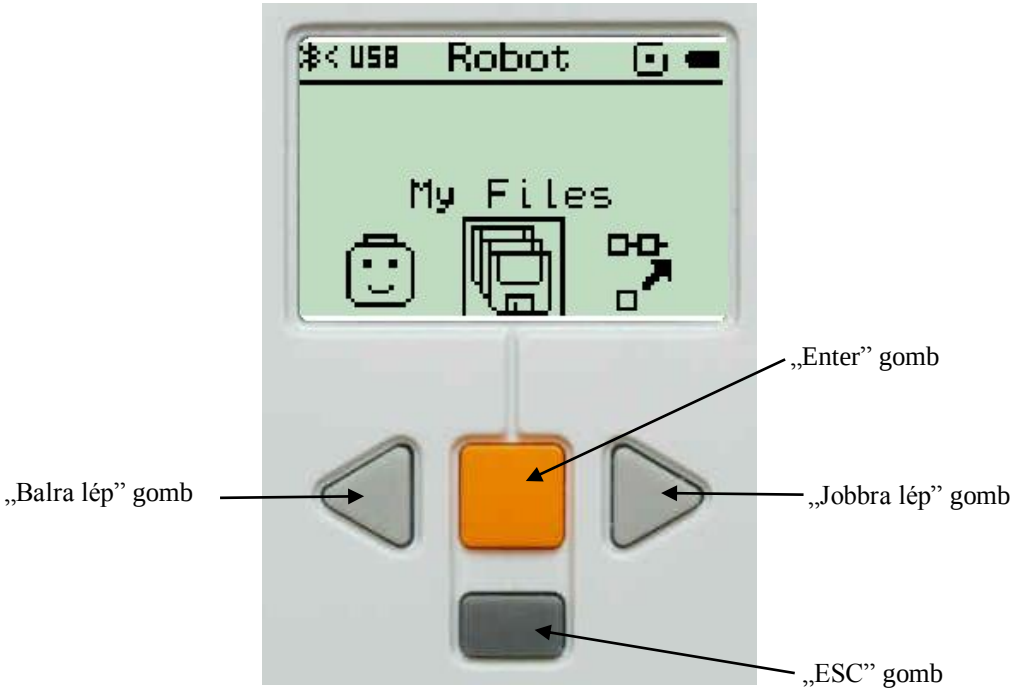
Nem írható olyan programozói könyv, amelyben az első betűtől az utolsóig haladva lineáris rendszerben tanulható meg a programozás. Az egyes komplexebb feladatoknál előfordulhatnak olyan elemek, amelyek későbbi fejezetben szerepelnek részletesen. Ezért a könyv egyes bonyolultabb feladatainál előfordulhat, 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.

Sumo robot erőkaros emelővel.

3. A ROBOT KÉPERNYŐMENÜJE

A programírás megkezdése előtt érdemes megismerkednünk a roboton lévő *firmware* által biztosított menürendszerrel és néhány funkciójával.

A bekapcsolás után egy menürendszerhez jutunk, amelyben a roboton elhelyezett négy nyomógomb segítségével lehet mozogni. Kezdetben a *My Files* nevű menü az aktív.



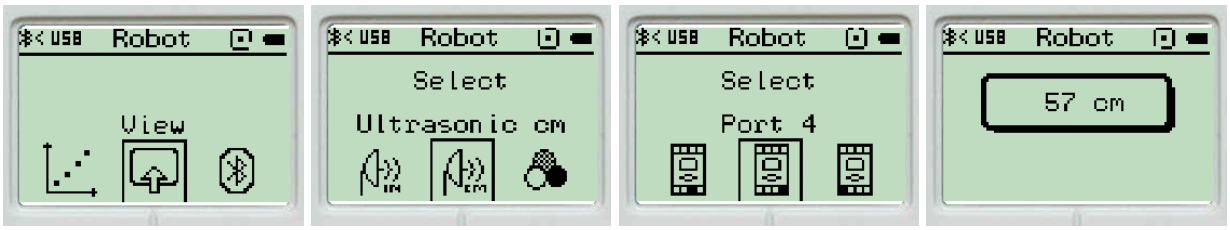
Az egyes menüpontok között balra és jobbra navigálni a két háromszög alakú gombbal lehet. Egy menübe belépni, vagy a kiválasztott funkciót aktiválni a narancssárga négyzet alakú gombbal (*Enter*), míg a menüben eggyel fentebbi szintre lépni a téglalap alakú gombbal (*ESC*) lehet.

Az első funkció, amit érdemes kipróbálni, a *View* funkció. Ennek segítségével a robotra csatlakoztatott szenzorok éppen mért értékei kérdezhetők le és írathatók a képernyőre. Erre akkor is szükségünk lehet, ha a kezdeti programírás során a szenzorok által mért speciális értékeket fel szeretnénk használni a programban, és ezeket inkább saját magunk állítjuk be, nem bízzuk a programra a meghatározásukat.

A *View* menühöz a jobbra gomb néhányszori megnyomása után juthatunk (mivel a menüben lépegetés körkörös, ezért a balra gombbal is eljuthatunk ugyanoda).

A *View* menüpontnál az enterrel tudunk az almenübe lépni. Itt ki kell választanunk azt a szenzort, amelynek az aktuálisan mért értékére kíváncsiak vagyunk. Újabb enter után annak portnak a számát kell kiválasztanunk, amelyre a szenzor csatlakozik. Ezután a képernyőn megjelenik a mért érték, amely folyamatosan frissül.

Például a 4-es portra csatlakoztatott ultrahangszenzor esetén a szenzor előtt 57 cm-re volt akadály, ahogy a képeken látható.

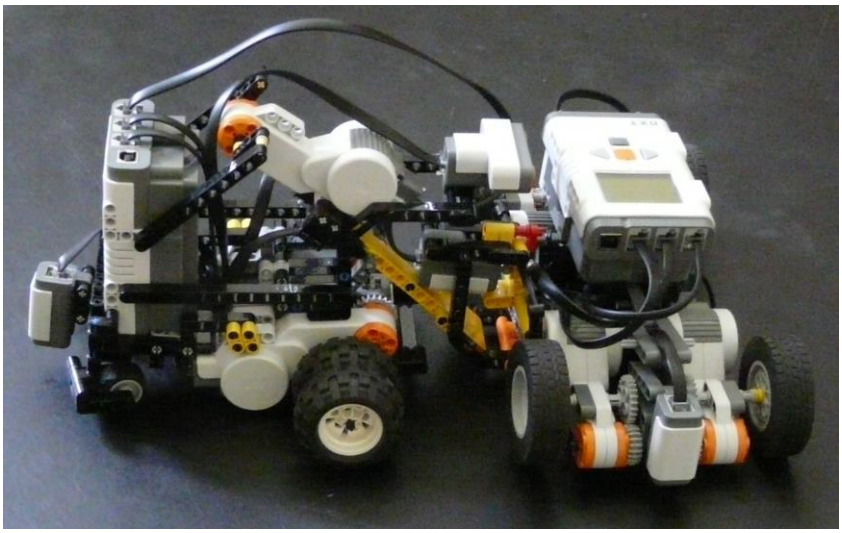


Visszalépni a menürendszerben az *ESC* gomb segítségével lehet.

A *My Files* menüből indulva juthatunk el a robotra a feltöltött programokig. Az enter gomb segítségével léphetünk be a *Softver files* menübe. Itt találhatók a feltöltött programjaink. A bemutatott példánál egy ilyen van, a Kepernyo_1. Az enter gombbal belépve a programba, három ikon jelenik meg. A *Run* feliratút választva az enter gomb hatására indítható a program.

A programot törölni a „kuka” ikon kiválasztásával lehet, míg a boríték ikon a program bluetoothon keresztüli küldését jelenti pl. egy másik robotnak. Ez utóbbi eset csak akkor használható, ha robotjaink között a bloetooth kapcsolatot felépítettük. Erről egy későbbi fejezetben részletesebben írunk. A feltöltött programok listájához visszalépni az *ESC* gombbal lehet.

A robot képernyőmenüjében több funkció is van még, amelyeket érdemes végigpróbálni. A későbbi fejezetekben még néhányról szó lesz.



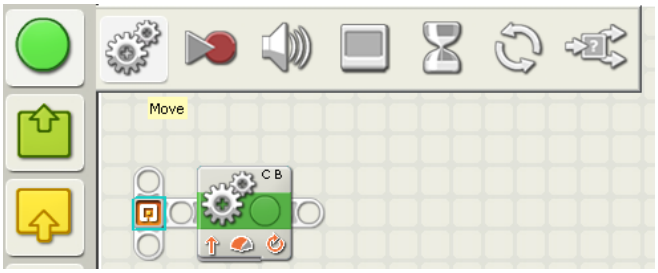
Sumo robotharcosok küzdelme.

4. EGYSZERŰ MOZGÁSOK

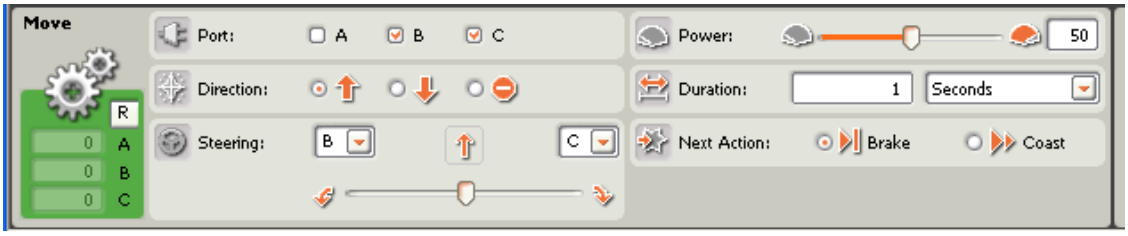
4.1. Motorok vezérlése

A robot a vezérlőegységhez kapcsolt motorok segítségével valósítja meg a különböző mozgásokat. A robothoz három motor csatlakoztatható, melyek csatlakozási helyeit A, B és C betűkkel jelölték. A motorok vezérlésére két vezérlő modul is alkalmas. Ezek a *Common* kategóriában található *Move* és az *Action* kategóriában található *Motor* modulok.

A robot mozgatását leggyakrabban a *Move* modullal valósítjuk meg.



A *Move* modul paraméterezését mutatja be a következő ábra:



Az egyes paraméterek jelentését a következő táblázat tartalmazza.

A paraméter neve	A paraméter jelentése
Port	A *Port* paraméter segítségével adható meg, hogy a robothoz csatlakoztatott motorok közül melyeket szeretnénk vezérelni. A robothoz három motor csatlakoztatható. A könyvben szereplő példákban csak két motort (B és C port) használunk. Jelölő négyzetek segítségével lehet kiválasztani a vezérelni kívánt motorokat. A többi paraméter beállítása ezekre a kiválasztott motorokra lesz érvényes. A fenti ábrán a B és C portokra kötött motorokra érvényesek a beállítások.
Direction	A *Direction* paraméter három különböző értéket vehet fel. A három érték közül rádiógombok segítségével választhatunk, vagyis közülük mindig pontosan egyet lehet kiválasztani. Az első kettő a motor forgásirányát és egyben a robot mozgásának az irányát adja meg. A harmadik érték leállítja a motorokat.

Steering

A robot nem rendelkezik kormányzott kerekekkel. A kormányzás úgy oldható meg, hogy a két motort különböző sebességgel működtetjük. Ekkor a robot a lassabban működő motor irányába elfordul. Ezt a 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 a B motor irányába, akkor a B motor gyorsabban forog, és a robot nagy ívben elfordul. Ha a csúszkát teljesen a B motor irányába toljuk, akkor a motorok azonos sebességgel, de különböző irányba fognak forogni, és a robot helyben fordul. Ez a paraméter csak két motor vezérlése esetén használható.

Power

Ezzel a paraméterrel a motorok sebességét szabályozhatjuk. A paraméter értéke 0 és 100 között változhat. Az értéket a csúszkával, vagy a csúszka melletti szövegdobozva beírva adhatjuk meg.

Duration

Ennek a paraméternek a segítségével állíthatjuk be, hogy mennyi ideig működjenek a motorok. A paraméter melletti legördülő listából négy értéket választhatunk.

Degrees – hány fokot forduljon a motor tengelye. Értéke 0 és 2147483647 közötti lehet.

Rotations – hányszor forduljon körbe a motor tengelye. Három tizedesjegy pontossággal adható meg az érték.

Seconds – mennyi ideig forogjanak a motorok. Az időt másodpercben adhatjuk meg három tizedesjegy pontossággal.

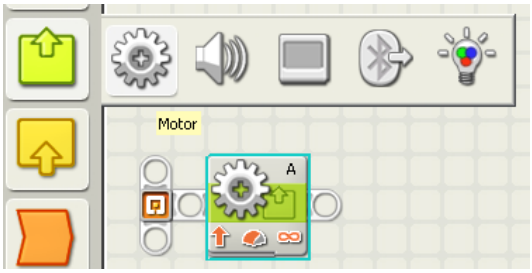
E paraméterek valamelyikét választva a vezérlés addig marad a *Move* modulon, míg a beállított paraméter értéke le nem telik. Például a *Seconds* választása és a paraméter 2-re állítása esetén 2 másodpercig lesz a vezérlés a Move modulon, aztán a következő modulra lép a program végrehajtása.

Unlimited – bekapcsolja a motorokat, majd a következő utasítások kerülnek végrehajtásra. A motorok addig működnek, amíg egy következő motorvezérlő utasítás meg nem változtatja azokat, vagy a program véget nem ér.

Next Action

A motorok leállításának a módját szabályozza. Rádiógombok segítségével két lehetőség közül választhatunk. 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 motorok vezérlésének másik módja az *Action* kategóriában megtalálható *Motor* modul használata.



A *Motor* ikon segítségével egyszerre egy motort tudunk vezérelni. A modul paraméterezése több ponton megegyezik a *Move* modul paraméterezésével.

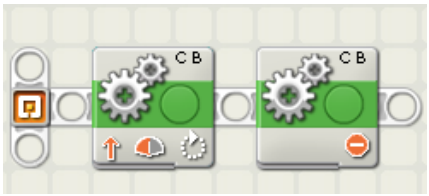
Az egyes paraméterek jelentését a következő táblázat tartalmazza.

A paraméter neve	A paraméter jelentése
Port	Melyik porthoz csatlakoztatott motort szeretnénk vezérelni. A három rádiógomb közül pontosan egy választható.
Direction	Jelentése megegyezik a *Move* modulnál leírtakkal.
Action	A paraméter segítségével azt adhatjuk meg, hogy a robot milyen módon érje el a *Power* paraméterrel beállított sebességet. A paraméter csak abban az esetben használható, ha a *Duration* paraméter értéke nem *Unlimited*. Ekkor három érték közül választhatunk, melyek jelentése a következő: *Constant* – a motor folyamatosan a megadott sebességgel forog (indulástól – leállásig). *Ramp Up* – a motor sebessége folyamatosan nő, amíg el nem éri a megadott értéket. *Ramp Down* – leálláskor a sebesség folyamatosan csökken a megadott értékről nullára.
Power	Jelentése megegyezik a *Move* modul *Power* paraméterénél leírtakkal.
Control	Ennek a paraméternek a bekapcsolásával elérhető, hogy a motor forgása egyenletes legyen. Használata esetén a vezérlés úgy változtatja a sebességet, hogy a másodpercenkénti fordulatszám állandó lesz.

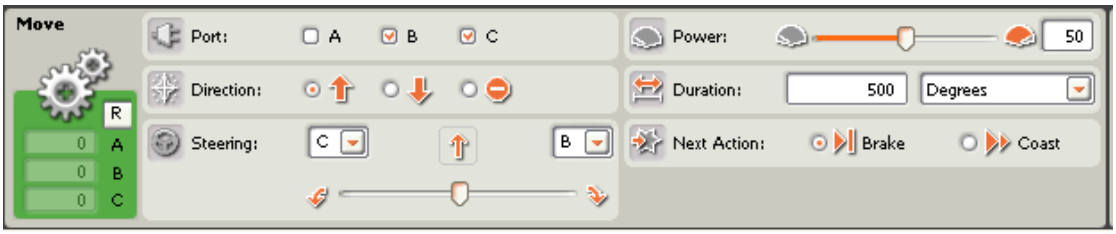
Duration	Jelentése megegyezik a *Move* modulnál leírtakkal.
Wait	A paraméter kikapcsolt állapota esetén a program végrehajtása nem áll meg a *Motor* modulnál. A motor elindul, és a végrehajtás tovább folytatódik a következő utasítással. A motor a beállított érték eléréséig működik. Csak abban az esetben használható, ha a *Duration* paraméternél beállított vezérlés *Degrees* vagy *Rotations*.
Next Action	Jelentése megegyezik a *Move* modulnál leírtakkal.

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 program két *Move* ikonból áll. Az első 500°-os tengelyfordulatig egyenesen előre forgatja a motorokat. Ennek paraméterezését mutatja az alábbi ábra.

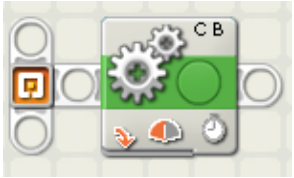


Amint a képernyőképen is látható, a B és C portokra kötött motorokat előre mozgatjuk. Mindkét motor azonos, 50-es sebességgel forog. A motorok működését a *Duration* paraméterrel szabályozzuk. A működés jelen esetben 500°-os tengelyfordulásig tart. A motorok leállása blokkolással történik (*Next Action* értéke *Brake*).

A második *Move* ikon csak kikapcsolja a motorokat, ezért egyedül a *Direction* paramétert kell *STOP*-ra állítani. Ez az ikon akár el is hagyható, hiszen a program végeztével a motorok le fognak állni.

4/P2. Írjon programot, amelyet végrehajtva a robot 50-es sebességgel körbe forog 3 mp-ig!

A feladat megoldásának a programkódja:



A program egy *Move* modulból áll, amely a B és C portra kötött motorokat 50-es sebességgel, de különböző irányban forgatja 3 mp-ig. A paraméterek beállítása a következő:

A helyben forgást úgy érjük el, hogy a *Steering* paraméter csúszkáját a B motor irányába teljesen kihúzzuk. Ennek hatására a motorok azonos sebességgel, de különböző irányban kezdenek forogni, és a robot helyben elfordul. A *Duration* paraméter beállításával adjuk meg a mozgás idejét.

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 *Move* ikonból áll.

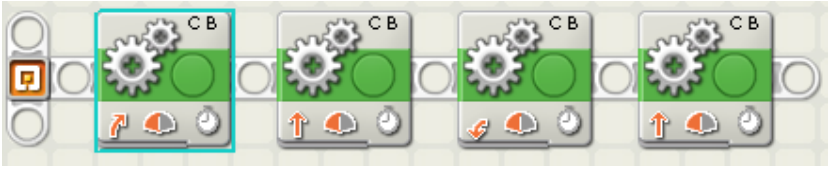
Az első a B és C motorokat 50-es sebességgel 2 mp-ig hátrafelé forgatja. Ennek paraméterezése nagyon hasonlatos az 1. feladat megoldásánál ismertetetthez.

A második *Move* ikon 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 0,6 mp-ig működtetjük. A megfelelő időt célszerű kísérletezéssel meghatározni, mivel ez függhet a robot sebességétől, a kerék átmérőjétől vagy az akkumulátorok töltöttségi szintjétől. A modul paraméterezését a következő ábra mutatja.

Felhívjuk a figyelmet arra, hogy a *Duration* paraméter „90°, *Degrees*” beállítása nem a robot 90°-kal történő elfordulását eredményezi, hanem a motor tengelyének 90°-os elfordulását.

A harmadik *Move* modul a háromszoros tengelyfordulásig történő előre mozgást valósítja meg.

4/P4. Írjon programot amelyet végrehajtva a robot nagy ívben jobbra fordul 3 mp-ig, majd egyenesen halad 2 mp-ig, végül helyben fordul kb. 180°-ot, és ismét egyenesen halad 2 mp-ig!

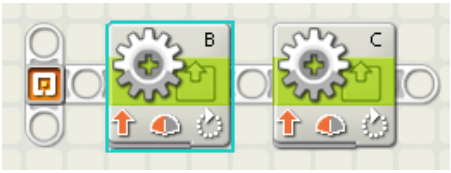


A program négy *Move* ikonnal megvalósítható. Ezek paraméterezését nagyobbrészt az előzőekben már bemutattuk. Az első ikon a nagy ívben való fordulást valósítja meg. Itt a *Steering* paraméter értékét kell úgy beállítani, hogy a B motor (a robot felépítésétől is függ) forogjon gyorsabban. Ezt a beállítást mutatja az alábbi ábra.

A második és negyedik ikon az előremozgásokat valósítja meg.

A harmadik *Move* modul kb. 180°-kal elfordítja a robotot. A modul paraméterezése megegyezik a 3. feladatban a 90°-kal történő elforgatásnál alkalmazottal, azzal az eltéréssel, hogy itt az időkorlát 1,2 mp.

4/P5. Írjon programot, amelyet végrehajtva a robot a Motor ikon felhasználásával halad előre 1440°-os tengelyfordulatig!



A programot két *Motor* ikon felhasználásával oldottuk meg. Paraméterezésükben a *Port* paraméter kivételével teljesen megegyeznek. Az első ikon paraméterezését mutatja az ábra.

A B motort vezéreljük. A mozgás előre történik 50-es motorerővel, 1440°-os tengelyfordulásig. Mivel a *Wait* paraméter kikapcsolt állapotban van, ezért az ikonon nem áll meg a végrehajtás, hanem azonnal továbblép a következő ikonra. Így a két motor egyszerre működik, és a robot egyenesen előre halad. A második modul (C motor) esetén a *Wait for Completion* paraméter értékét hagyjuk bekapcsolva, mert ellenkező esetben a program véget ér és a robot el sem indul.

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!

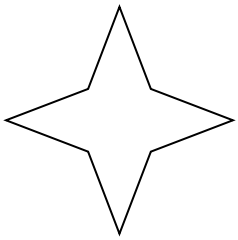
4/F2. Írjon programot, amelyet végrehajtva a robot nagy ívben balra kanyarodik 5 mp-ig, majd tolat 2 mp-ig, végül nagy ívben jobbra kanyarodik 5 mp-ig!

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

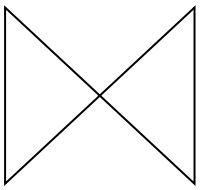
4/F4. Írjon programot, amelyet végrehajtva a robot mozgás közben egy négyzetet ír le!

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!



5. SZENZOROK HASZNÁLATA

Az előző fejezetben néhány egyszerű példán keresztül áttekintettük, hogyan mozgathatjuk a robotot. A robot azonban érzékelői (szenzorai) segítségével képes érzékelni a „külvilág” jeleit.

Az 1. fejezetben bemutatott szenzorok közül leggyakrabban a négy „alapszenzort” használjuk. Ezek a következők:

- ütközésérzékelő (*touch sensor*);
- színérzékelő/fényérzékelő (*color sensor/light sensor*);
- távolságérzékelő (*ultrasonic sensor*)
- hangérzékelő (*sound sensor*)

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.

Az érzékelők számára négy csatlakozási pont (port) található a téglán. Ezek 1-től 4-ig vannak számozva. 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 szenzoron bekövetkezett esemény vagy a szenzor által visszaadott érték a program futását felfüggeszti. Ennél a módnál valamennyi szenzor egy ikonnal a *Common* kategóriában található *Wait*-tel vezérelhető.

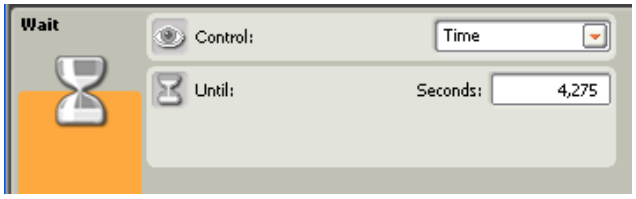
A *Wait* ikon lényegében addig várakoztatja a programot, amíg a szenzoron beállított esemény be nem következik, vagy amíg a szenzor a paraméterként megadott értéknek megfelelő adatot nem mér. Ha a beállított feltétel teljesül, akkor a vezérlés a program végrehajtását a *Wait*-tet követő ikonnal folytatja.

Az ikon paraméterezését a következő ábra mutatja:

5.1. Várakozás megadott ideig

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

A *Control* paraméter segítségével lehet beállítani, hogy hogyan történjen a vezérlés. Ha az értékét *Time*-ra állítjuk, akkor a várakoztatás a megadott ideig történik. Az időt a *Until* paraméternél adhatjuk meg másodpercekben mérve ezredmásodperc pontossággal. Ezt mutatja be a következő példa.

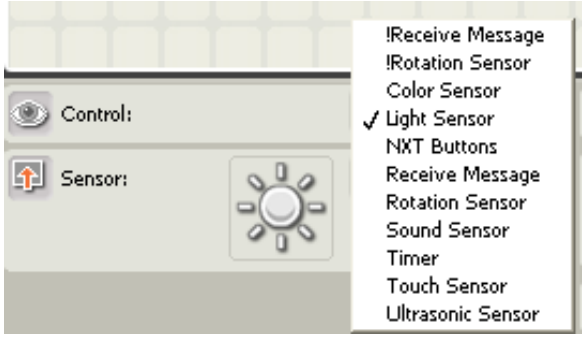


5/P1. Írjunk programot, amelyet végrehajtva a robot előre mozog 2 másodpercig, majd megáll és várakozik 3 másodpercig! Ekkor jobbra fordul kb 90°-ot.

A feladat megoldásának programkódja a következő három ikonból áll.

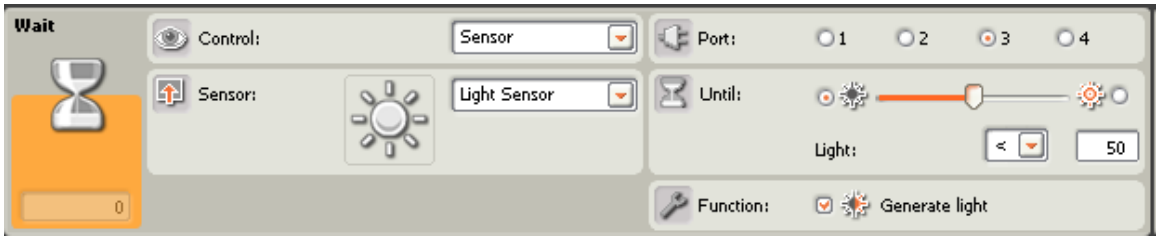
A *Wait* ikon paraméterezésénél a *Control* paraméter értékét *Time*-ra, az *Until* paramétert pedig 3 másodpercre állítottuk.

Szenzorokkal történő vezérlés esetén a *Control* paramétert a *Sensor* értékre állítjuk. Ekkor az megjelenő *Sensor* listából kell kiválasztanunk, hogy melyik szenzorral kívánjuk megvalósítani a vezérlést. A további paraméterezés a kiválasztott szenzortól függ. A *Sensor* paraméter lehetséges értékeit mutatja a következő ábra.



5.2. Fény- és színérzékelő (*Light Sensor, Color Sensor*)

A fényszenzor (*Light*) az NXT 1.0 készlet része. A felületről visszavert fény intenzitását méri. A mért érték 1-100 közötti skálán változhat. A programozás megkezdése előtt érdemes megmérni a visszavert fény intenzitását, mivel ez a felület színétől és a fényviszonyoktól is függ. A mérést a tégla *View* funkciójával végezhetjük el. A robot képernyőjén a következő menüpontokat kell kiválasztanunk: *View\Reflected light\Port*. Ekkor a robot képernyőjén megjelenik az adott pillanatban mért fényintenzitás. Különböző fényszenzorokkal mérve ugyanazon a felületen eltérő értékeket kaphatunk.
A fényszenzor használata estén a következő paramétereket kell beállítanunk.



Először kiválasztjuk, hogy melyik portra kötöttük a szenzort. Az *Until* paraméterrel adhatjuk meg, hogy mekkora mért értékig várakoztassa a program végrehajtását. Értékét a csúszka segítségével vagy az alatta lévő beviteli mezőbe történő beírással állíthatjuk. A megfelelő relációs jelet a legördülő listából választhatjuk ki. Ha a *Function* paraméter *Generate light* jelölőnégyzetét kiválasztjuk, akkor a fényérzékelő vörös fénnyel világítja meg a felületet. Egyébként nincs megvilágítás, de a fényintenzitás értékeket méri a szenzor.

5/P2. 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. szigetelő szalag felragasztásával állíthatjuk elő, ez ebben az esetben fekete volt. 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 feladat megoldása három ikonból áll.

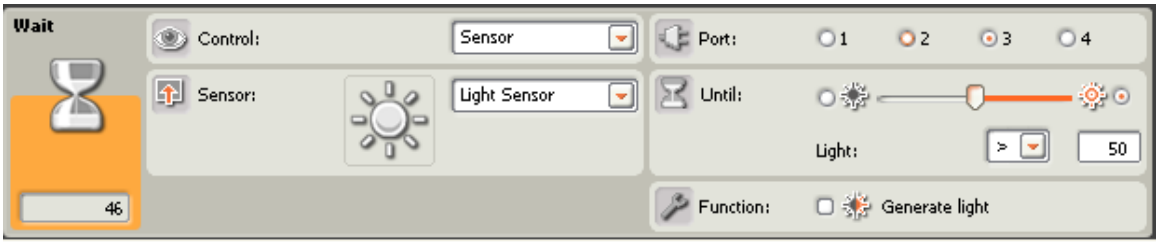
Az előre mozgásnál a motorerőt 50-re, a *Duration* paraméter értékét *Unlimited*-re állítottuk. Mivel a második ikon addig várakoztatja program végrehajtását, amíg a fényérzékelő által mért érték kisebb

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.

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

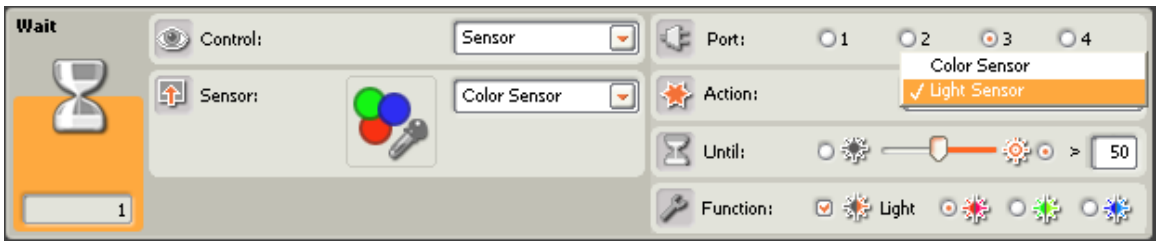
A feladat megoldásához az alaprobotot úgy kell átépítenünk, hogy a fényérzékelője előre, vagy felfelé nézzen. A képernyő *View\Ambient light* funkciójával megmértük a fényérzékelő által elnyelt fényintenzitás értékét rávilágítás nélkül (40) és egy zseblámpával történő rávilágítása esetén (60). A képernyőmenü *Wiev* funkciójának *Ambient light* értékénél a szenzorba épített vörös fényű led nem kapcsol be, így nem a felületről visszavert fényintenzitás értékeket mérjük, hanem a környezetét..

A folyamatos forgást a *Duration* paraméter *Unlimited*-re 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. A *Wait* ikon paraméterezését a következő ábra mutatja:



Mivel itt nem a felületről visszavert fényt kell érzékelni, ezért a *Generate light* opciót nem kapcsoltuk be. A robot akkor fogja a zseblámpa fényét érzékelni, ha a beállított 50-es értéknél nagyobb fényintenzitást mér.

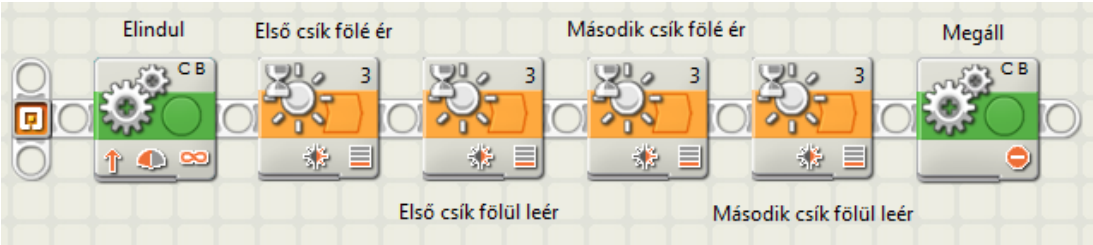
Az NXT 2.0 készletben a fényérzékelő helyett egy színszenzort kapunk. Ez képes színek megkülönböztetésére. Az érzékelni kívánt színtartomány a *Until* legördülő listája és csúszkája segítségével állítható be. Ha a színérzékelő *Action* paraméterét *Color Sensor* helyet *Light Sensor*ra állítjuk, akkor a színérzékelőt hagyományos fényérzékelőként használhatjuk. Az ilyen módon történő használat annyiban tér el az előzőekben leírtaktól, hogy itt a vörös helyet a felület megvilágítására zöld, illetve kék színt is választhatunk. A színérzékelő paraméterezési lehetőségeit mutatja az ábra:



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 fogja mérni. Tehát egy csíkon áthaladva kétszer változnak meg jelentősen a mért értékek. Ennek megfelelően 4 db *Wait* modult fogunk használni (fényszenzor módban), 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. A két érték számtani átlagát használjuk határértékként. Tehát ha a robot fényszenzora 44-nél kisebb értéket mér, akkor fekete, míg nagyobb érték esetén fehér színű felület fölött van.



Az első két *Wait* ikon paraméterezése:

A motorok bekapcsolása után a program az első *Wait* utasításnál addig vár, amíg fekete csík fölé nem ér a fényszenzor (<44), majd ezután a második *Wait*-nél vár addig, amíg fehér felület fölé nem ér (>44), és így tovább. A motorok mindeközben működnek mindaddig, amíg a negyedik színváltás is megtörténik. Ekkor állnak le a motorok.

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.

5.3. Ütközésérzékelő (*Touch sensor*)

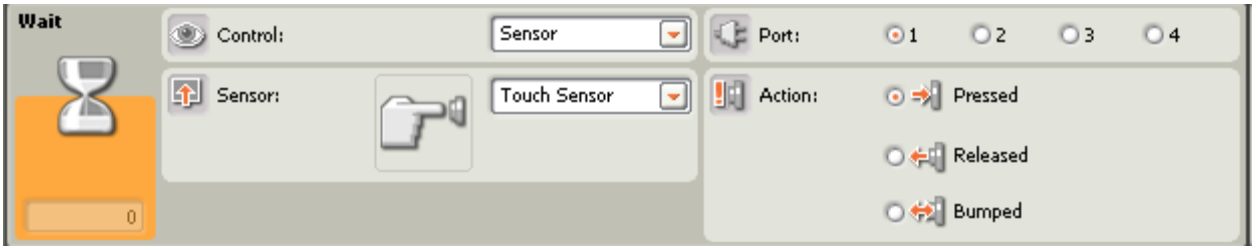
Az ütközésérzékelő három állapot megkülönböztetésére alkalmas.

Pressed – az ütközésérzékelő gombja benyomott állapotban van.

Released – az ütközésérzékelő gombja kiengedett állapotban van, ez az alaphelyzete.

Bumped – az ütközésérzékelő gombjának gyors benyomása és felengedése, egérkattintásszerűen. Az érzékelő gombján állapotváltozás megy végbe, először benyomás, majd felengedés.

Az ütközésérzékelőt szintén használhatjuk a *Wait* vezérlésére. Paraméterezési lehetőségeit a következő ábra mutatja be:



A *Wait* ikon vezérlésének ütközésérzékelőre állítása után csak az érzékelő portjának a számát kell megadnunk, illetve hogy az érzékelő három állapota közül melyiket figyeljük. Ez utóbbit az *Action* paraméter megfelelő rádiógombjának a kiválasztásával tehetjük meg.

5/P5. Írjon programot, amelyet végrehajtva a robot egyenesen addig tolat, amíg akadálynak nem ütközik, ekkor megáll!

A programot a fenti ábra mutatja be. A folyamatos hátra mozgást az első *Move* ikon biztosítja. A *Wait* addig várakozik, míg az 1-es portra kötött ütközésérzékelő benyomott (*Pressed*) állapotba nem kerül. Ekkor tovább engedi a program futását, és a robot blokkolva megáll.

5/P6. Írjon programot, amelyet végrehajtva a robot egyenesen előre halad, majd az ütközésérzékelője megnyomása és felengedése esetén megfordul és tolat 2 mp-ig!

Ebben a programban az ütközésérzékelő megnyomását (*Bumped*) figyeljük. A feladat megoldásához a benyomott (*Pressed*) állapot figyelése is megfelelő volna. A *Bumped* és *Pressed* érték közötti programozási különbségre a későbbiekben mutatunk példát.

5.4. Hangérzékelő (*Sound Sensor*)

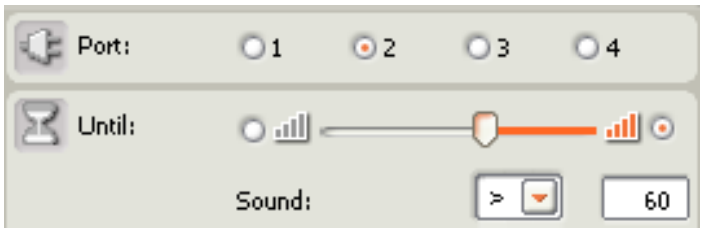
A hangérzékelő a környezet zajszintjét méri decibelben vagy korrigált decibelben. A programkörnyezet számára egy 0-100 közötti értéket ad vissza. Használata előtt érdemes a képernyő *View* funkciójával mérést végezni és a mérés eredményének megfelelő konstansokat használni a programban. A hangérzékelő *Wait* ikonban történő paraméterezését mutatja a következő ábra:

A paraméterezése hasonló a fényérzékelőéhez. A port számának kiválasztása után a továbblépési feltételt az érzékelendő hang nagyságának megadásával végezhetjük el.

5/P7. Írjon programot, amelyet végrehajtva a robot áll mindaddig, amíg a hangérzékelője 60 dB-nél kisebb értéket mér! Ekkor induljon el és haladjon egyenesen előre 3 mp-ig!

A program kódja a következő:

A 2-es portra kötött hangérzékelő 60 dB-nél erősebb hang észleléséig várakoztatja a program végrehajtását. Paraméterezése az ábrán látható:



Ennél erősebb hang észlelése esetén a vezérlés a következő ikonra lép és a robot 3 mp-ig előre mozog.

5.5. Ultrahangos távolságmérő (*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 0-250 cm tartományban képes megmérni. A *Wait* ikon ultrahangos távolságmérővel történő vezérlése esetén a következő paramétereket állíthatjuk be:

A *Port* paraméterrel adjuk meg, hogy a tégla melyik portjához csatlakozik a távolságmérő. A *Show* paraméternél állíthatjuk be a használni kívánt hosszúságegységet (centiméter vagy incs). Végül – hasonlóan a hang- és fényérzékelőhöz – az *Until* paraméterrel beállíthatjuk a továbblépési feltételt a távolság nagyságának megadásával.

5/P8. 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:

Az ultrahangos távolságmérővel vezérelt *Wait* ikont a következő módon paramétereztük:

A távolságérzékelő a 4. portra van kötve. A mértékegységet centiméterre, a továbblépési feltételként mért távolságot 15 cm-nél kisebbre állítottuk.

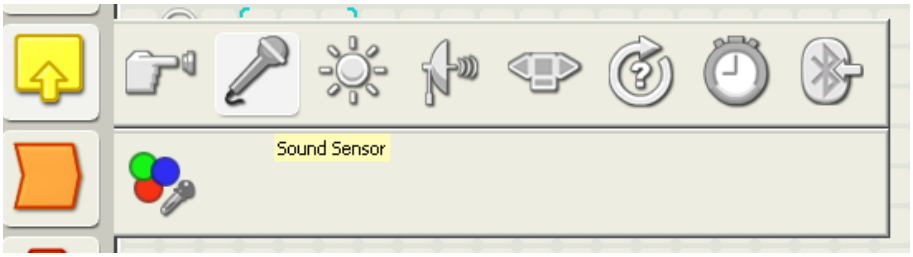
A feladatot az első fejezetben bemutatott robottal oldottuk meg. Ha ettől eltérő konstrukciójú robotot használunk, és a robot távolságérzékelője túl magasra van szerelve, akkor előfordulhat, hogy a robot nem látja meg az akadályt. Ha az ultrahangos távolságérzékelő a robot elejéhez viszonyítva túlságosan

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/P9. Írjon programot, amelyet végrehajtva a robot áll mindaddig, amíg a távolságérzékelője 20 cm-nél kisebb távolságot nem mér, ekkor mozogjon előre 2 mp-ig!

A feladat megoldásának kódja következő:

A szenzorok használatának másik módja, amikor az általuk mért értékeket a programban kiolvassuk. Ezeket a kiolvasott értékeket változókban tároljuk vagy közvetlenül használjuk fel a programvezérléshez. Az ilyen módon történő felhasználáshoz szükséges ikonokat a *Sensor* modulban találjuk.



A szenzorok e módon történő használatát a 7. fejezetben mutatjuk be.

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, és az asztal másik széléig haladva 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 kb. 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 kb. 90°-ot, és haladjon előre a felület színétől eltérő csíkig!

5/F8. Írjon programot, amelyet végrehajtva a robot forogni kezd, ha a fényérzékelőjére rávilágítunk, és az ütközésérzékelő megnyomására megáll!

5/F9. Írjon programot, amelyet végrehajtva a robot fehér felületen lévő fekete csíksor fölött halad, és a harmadik fekete csík fölötti áthaladás után megáll!

5/F10. 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/F11. Írjon programot, amelyet végrehajtva a robot lassan köröz ütközésérzékelő benyomásáig! Ekkor megfordul és az ellenkező irányba köröz tovább mindaddig, amíg ismét nyomás éri az ütközésérzékelőjét.

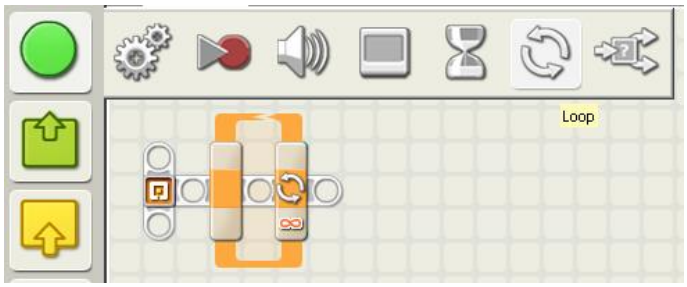
Marsjáró robot hat motorral. Képes 40°-os emelkedőn felmenni, akadályokat kikerülni.

6. VEZÉRLÉSI SZERKEZETEK

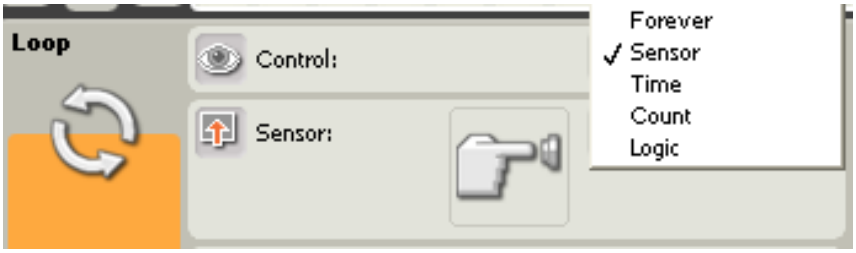
Az eddigi példaprogramjaink mind úgynevezett szekvenciális szerkezetűek voltak, vagyis az utasítások sorban, egymás után hajtódtak végre. Sok esetben azonban a feladat megoldásához ez a szerkezet nem elegendő. Előfordulhat, hogy valamely feltételtől függően különböző utasításokat vagy utasítássorozatokat 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 kötjük. Az ilyen ciklusokat feltételes ciklusoknak nevezzük. A feltételes ciklusoknak két fajtáját különböztetjük meg. Az előfeltételes (elől tesztelő) ciklus esetén a feltétel vizsgálata a ciklusmag végrehajtása előtt történik. Utófeltételes (hátul tesztelő) ciklusnál a feltételt a ciklusmag végrehajtása után vizsgáljuk.
Az NXT-G nyelvben mind a növekményes, mind a feltételes ciklus megvalósítható. A feltételes ciklusok közül az NXT-G nyelv csak az utófeltételes (hátul tesztelő) ciklust tartalmazza. A ciklusok ikonja a *Loop* ikon, amely a *Common* és *Flow* kategóriában egyaránt megtalálható.



A *Loop* ikon használata esetén először azt kell megadni, hogy mi módon történjen a ciklus vezérlése. Ez határozza meg a további beállítási lehetőségeket. A paraméterlistán a *Control* kategóriában öt vezérlési lehetőség közül választhatunk.



Forever – végtelen ciklus. Nincs előre megadott darabszám, sem ciklust vezérlő feltétel. A ciklus a „végtelenségig” fut, megszakítani például a téglán lévő „*ESC*” gombbal lehet. A ciklus megszakítása egyben a program befejezését is jelenti. A ciklus futása megszakítható még a program leállítását eredményező modul ciklusmagban történő elhelyezésével is (*Flow* ikoncsoport *Stop* modul).

Sensor – utófeltételes ciklus. A vezérlés valamely szenzorral történik. A ciklusból való kilépést valamelyik érzékelővel mért értékre beállított feltétel szabályozza. Ezeket a feltételeket a *Wait* ikonnál ismertetetthez hasonlóan állíthatjuk be. Például színszenzor használata estén a *Loop* ikon paraméterlistája a következő:

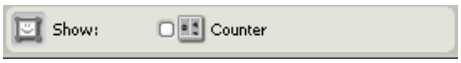
Ebben az esetben a színszenzort fényszenzorként használjuk. Ha a 3-as portra kötött érzékelő 50-nél nagyobb értéket mér, akkor fejeződik be a ciklusmagban szereplő utasítások ismétlése.

Time – a ciklus a megadott ideig fut. Az időkorlátot az *Until* paraméter *Seconds* mezőjének beállításával másodpercben lehet megadni ezred másodperc pontossággal. 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.

Count – növekményes ciklus. A megadott darabszámszor hajtja végre a ciklusmagban elhelyezett utasításokat, amelyet az *Until* paraméter *Count* mezőjében adhatunk meg.

Logic – a ciklust egy logikai kifejezés vezérli. A logikai kifejezés kétféle értéket, igaz (*True*) és hamis (*False*), vehet fel. A logikai kifejezést számokból, változókból, műveleti és relációs jelekből, logikai értékekből és logikai operátorokból állíthatjuk össze. Ilyen vezérlés esetén az *Until* paraméternél rádiógombokkal lehet beállítani, hogy a kifejezés igaz vagy hamis értéke esetén történjen a ciklusból a kilépés. A logikai kifejezésekről a későbbi fejezetekben lesz szó.

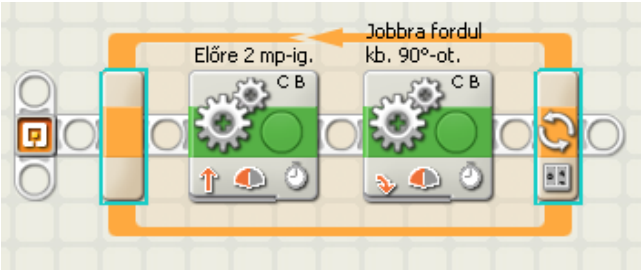
A fenti öt lehetőség közül bármelyiket választjuk, a *Loop* paraméterlistájában találunk egy *Show* paramétert. Ez egy *Counter* feliratú jelölőnégyzet. Bekapcsolt állapotban a rendszer megszámolja, hogy hányszor került végrehajtásra a ciklusmag, és a végrehajtások számát ki tudjuk olvasni és paraméterként átadni. Lásd a paraméterátadás fejezetet!



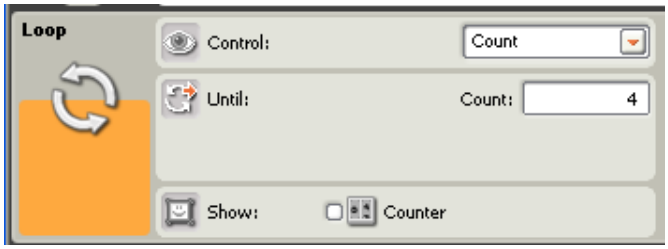
6/P1. Írjon programot, amelyet végrehajtva a robot mozgás közben egy négyzetet ír le!

Azok az olvasók, akik megoldották az *Egyszerű mozgások* című fejezet gyakorló feladatai közül a negyediket, már bizonyára tapasztalták, hogy a feladat két *Move* ikon négyszeri megismétlésével megoldható.

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

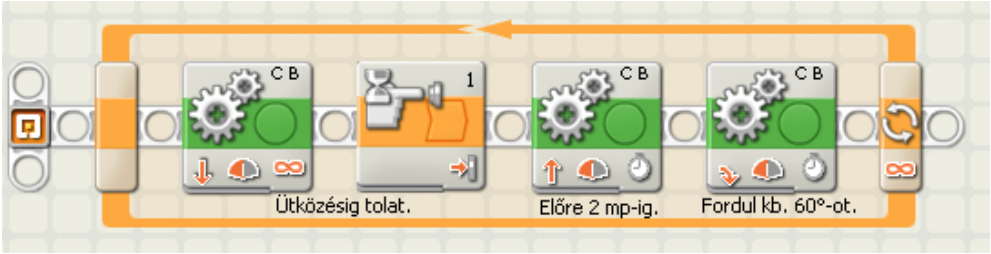


A *Loop* ikon paraméterezését mutatja a következő ábra.



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

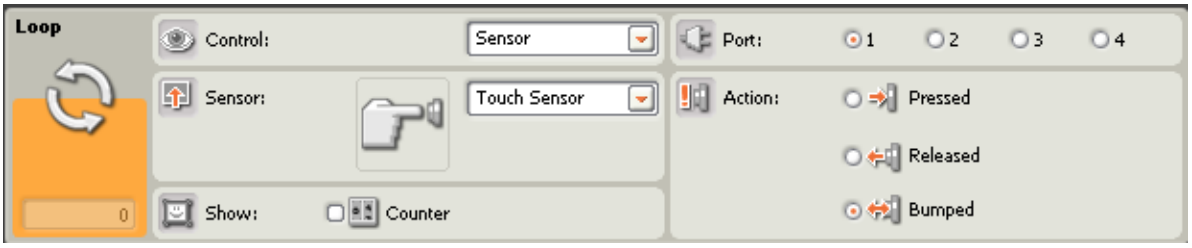
Mivel a robotnak a leírt tevékenységet kikapcsolásig kell ismételnie, ezért az ezeket megvalósító utasításokat egy végtelen ciklusba foglaltuk. A feladat megoldásának a kódja a következő.



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

A feladat megoldásának a kódja következő lehetne:

A ciklus vezérlése:



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.

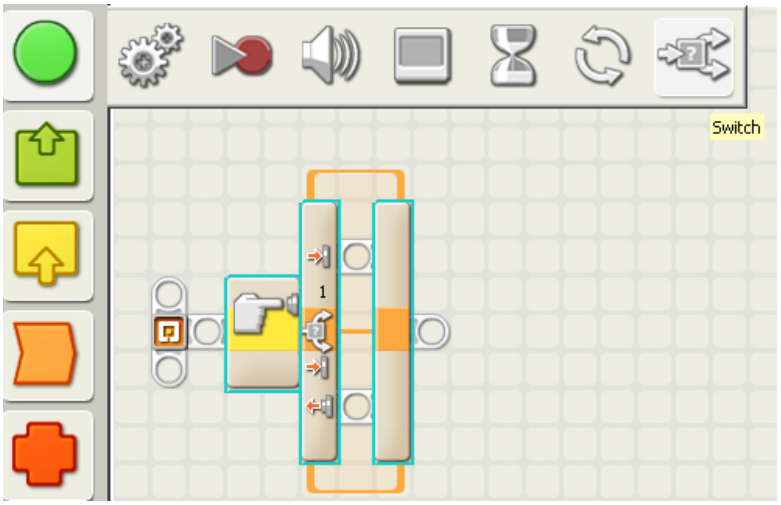
A hibát az okozza, hogy a ciklusmagban lévő utasítások várakoztatást tartalmaznak. Először az akadály észlelésére várakozik a program, majd a hátra mozgás idejének leteltére. Amíg a várakozási utasításokban beállított értékek nem teljesülnek, addig a program nem tudja végrehajtani az utánuk lévő utasításokat. 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.

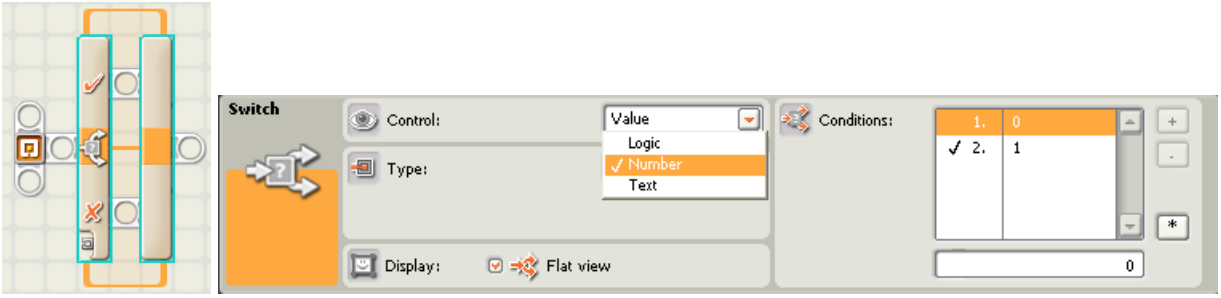
A feladat ilyen módon történő megoldásához szükséges ismeretek részletes magyarázata a többszálú programokkal foglalkozó 10. fejezetben található.

6.2. Elágazások

A másik fontos vezérlési szerkezet az elágazás. Az elágazást mindig valamilyen feltétel segítségével vezéreljük. 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 NXT-G nyelvben elágazásokat a *Common* és a *Flow* kategóriában található *Switch* ikonnal hozhatunk létre.

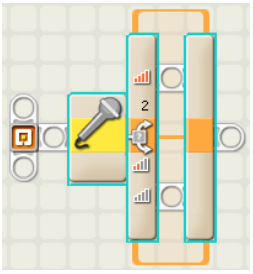


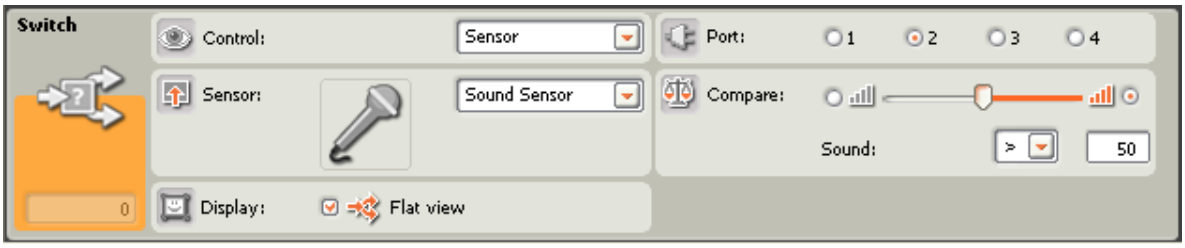
A *Switch* ikon esetében is azt kell először megadnunk, hogy hogyan történjen a vezérlés. Ezt a *Control* paraméter értékének beállításával tudjuk megadni. A *Value* érték választása esetén az elágazást logikai érték, szám vagy szöveges érték segítségével vezérelhetjük. Így többágú elágazásokat is létrehozhatunk. Használatához paraméterátadás szükséges.



Az elágazást vezérelhetjük valamely szenzor segítségével is, ekkor a *Control* paraméter értékét *Sensor*-ra kell állítani. Az ikon további paraméterezése természetesen függ a kiválasztott szenzortól is. A paraméterezés lényegében megegyezik a *Wait* ikonnál, illetve ciklusoknál leírtakkal.

Például a hangszenzor esetén:



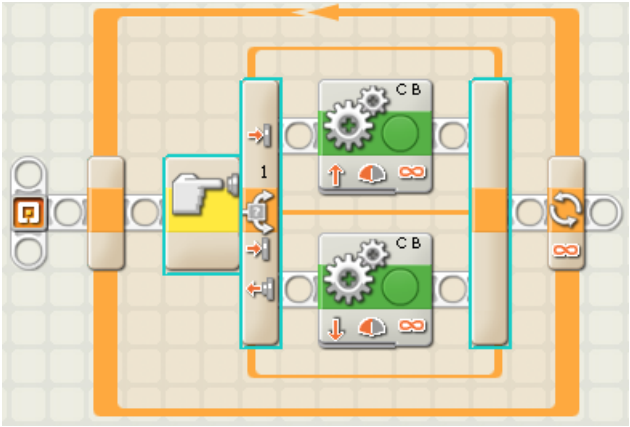


A két végrehajtandó utasítássorozat közül a felső szálon lévő hajtódik végre, ha a beállított feltétel igaz, egyébként pedig az alsón lévő. Ha egyágú elágazásra van szükségünk, akkor az egyébként ág (alsó szál) üresen hagyható.

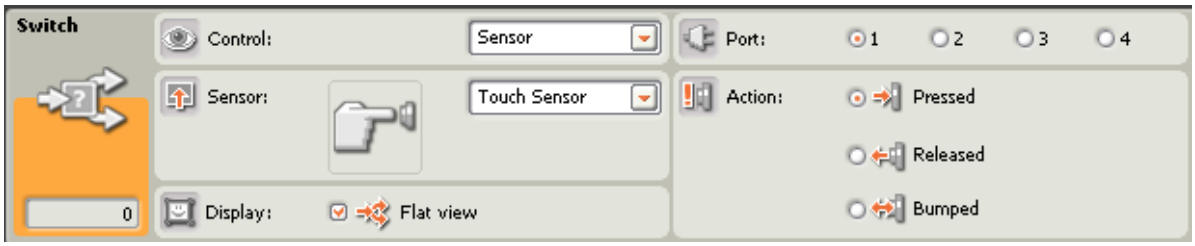
A paraméterlistában található egy *Display* nevű paraméter, amely egy *Flat view* nevű jelölőnégyzetből áll. Ezzel az ikon grafikus megjelenését szabályozhatjuk. Bekapcsolt állapota esetén mindkét ág látszik a képernyőn, egyébként csak az egyik. Ilyen esetben a két ág között a megfelelő fülön történő kattintással válthatunk. Nagyobb méretű programok esetén a jelölőnégyzet kikapcsolásával a programunk nagyobb része válik láthatóvá a képernyőn (mivel az elágazás moduljai kisebb helyen jelennek meg).

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:

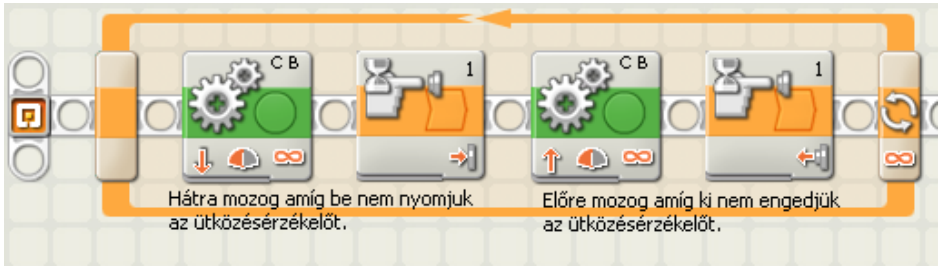


Mivel a tevékenységet a kikapcsolásig kell ismételni, ezért egy végtelen ciklust használunk. A ciklusmagot egy kétágú elágazás alkotja, amelyet az ütközésérzékelővel vezérelünk. Ennek paraméterezését mutatja az ábra.



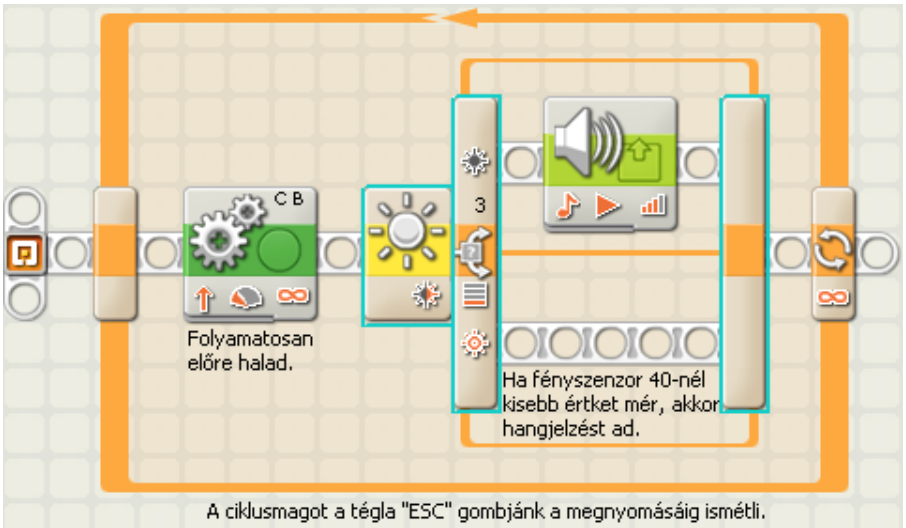
Az 1. portra kötött ütközésérzékelő benyomott (*Pressed*) állapotát figyeljük. Ha az ütközésérzékelőt benyomjuk, akkor a *Switch* feltétele teljesül, ezért a robot a felső igaz ágon szereplő utasításokat hajtja végre, és folyamatosan előre halad. Ennek eléréséhez a *Move* ikon *Duration* paraméterét *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 (szintén *Unlimited* a *Duration* értéke). A program a tégla „*ESC*” gombjának megnyomásával szakítható meg.

A feladat elágazás használata nélkül is megoldható a következő módon:



6/P5. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre egy az alapszíntől jól megkülönböztethető színű csíkokat tartalmazó felületen! A csík fölött haladva 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 a *View* funkcióval megmértük. A mért értékeket a programban konstansként használtuk.



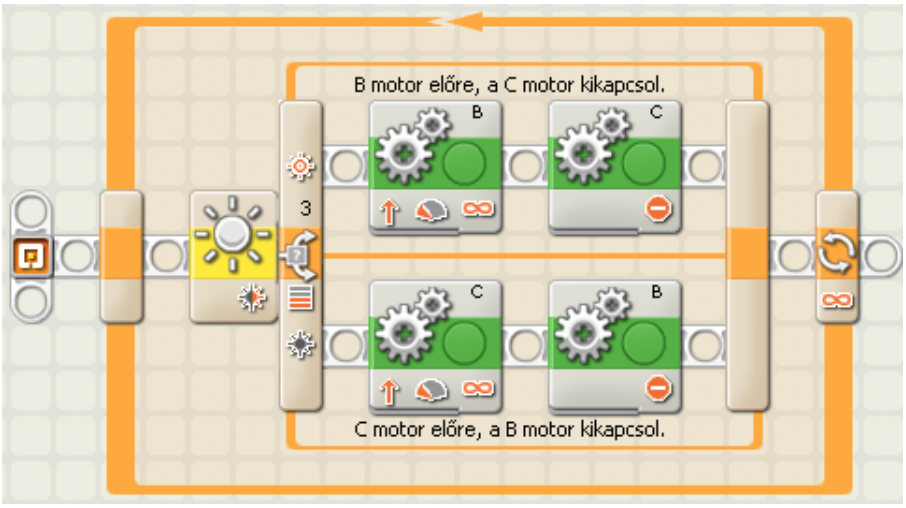
A folyamatos haladást és a hangot adó utasításokat egy végtelen ciklusba helyeztük el. A ciklusmag első utasítása egy *Move* ikon, amely 30-as motorerővel folyamatosan előre mozgatja a robotot. Ezt egy

egyágú elágazás követi. 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. Ha a feltétel nem teljesül, akkor nem kell csinálnunk semmit, ezért a *Switch* alsó ága üresen marad. A tégla „ESC” gombjának a megnyomására a program befejeződik. A programkörnyezet hangkezelő moduljáról a későbbiekben részletesebben lesz szó.

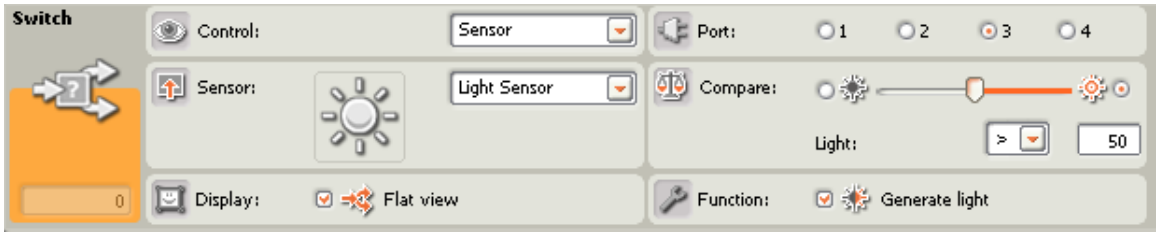
6/P6. Írjon programot, amelyet végrehajtva a robot a fényszenzora segítségével az alapszíntől jól megkülönböztethető színű útvonalat követ! Az útvonal lehet például egy fehér felületre felragasztott fekete szigetelőszalagból készített vonal.

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 előre forgatja, az előzőleg működőt leállítja, és lefordul az útvonalról. Ezt egy végtelen ciklusban ismételve a robot kígyózó mozgással követi az út határvonalát. 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 megoldás kódja következő:



Az elágazás paraméterezését a következő ábra mutatja:

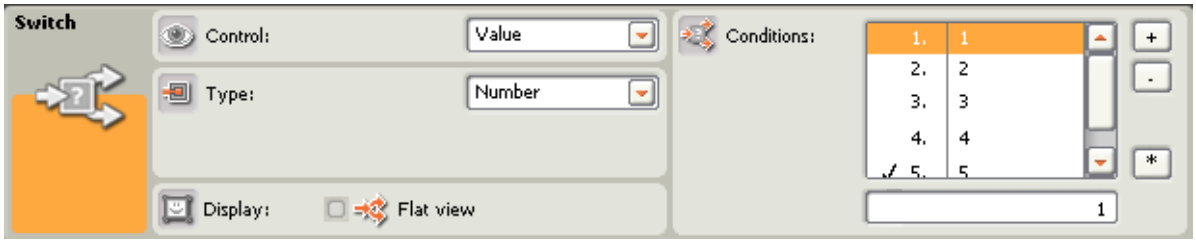


6/P7. Írjon programot, amelyet végrehajtva a robot az ütközésérzékelő megnyomására véletlenszerűen sorsol egy 1 és 5 közötti egész számot, majd a képernyőre írja a számnak megfelelő osztályzatot, s ezt kikapcsolásig ismétli!

A kisorsolt értéktől függően az elégtelen, elégséges, közepes, jó és jeles szavak valamelyikét kell a képernyőn megjeleníteni. 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 *Control* paramétert állítsuk *Value*-ra, a *Type*-nál pedig válasszuk a *Number* értéket. Ekkor egy kétágú elágazás jön létre, melynek első ágához a 0 a másodikhoz pedig az 1 érték van rendelve. Többágú elágazás létrehozásához kapcsoljuk ki a *Display* paraméter *Flat view* jelölőnégyzetét. A *Conditions* paraméternél a 2. ág melletti értéket írjuk át 2-re, majd az első mellettit 1-re. Ezt a paraméter alatti beviteli mezőben tehetjük meg. A *Conditions* paraméter melletti + jelre kattintva újabb értékeket adhatunk a listához és egyben új ágakat a *Switch* ikonhoz. Ezek az ágak egy-egy fülként jelennek meg a *Switch* grafikus nézetében. A – jel segítségével csökkenthetjük az ágak számát. A mi példánkban öt ágra van szükség. Ezek után minden ágra egy *Display* ikont kell helyeznünk. Az első ágon a kiíratandó szöveg az „ELEGTELEN”, a másodikon az „ELEGSEGES”, míg végül az ötödiken a „JELES”. Az első ágat akkor hajtja végre, ha a kisorsolt szám az 1, a másodikat, ha 2 és végül az ötödik ágat, ha 5. A *Display* ikon a véletlen szám előállítására szolgáló ikon működését és a paraméterátadást a könyv későbbi fejezetei részletesen tárgyalják.

A feladat megoldásra szolgáló program kódja a következő:

A *Switch* ikont a következő módon paramétereztük:



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 kb. 90°-ot! 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 folytassa az előre mozgást!

6/F4. Írjon programot, amelyet végrehajtva a robot addig halad előre, amíg az ultrahang szenzorával 15 cm-en belül akadályt nem észlel, ekkor az ütközésérzékelő benyomásáig halad 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/F6. Írjon programot az előző feladat mintájára, amelyet végrehajtva a robot képes egy zseblámpa fényének észlelésére és követésére!

6/F7. Írjon programot, amelyet végrehajtva a robot hatszög alakú pályán mozog!

6/F8. 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!

7. PARAMÉTERÁTADÁS ÉS VÁLTOZÓK

7.1. Paraméterek, adattípusok

A programozás során szükségünk lehet arra, hogy a bemeneti szenzorok (fényérzékelő, ütközésérzékelő, ultrahangszenzor, 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. Más 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 NXT-G nyelv három alapvető adattípusa: szám, szöveg, logikai érték.

Több programnyelv ennél lényegesen több típust különböztet meg, de valamennyi visszavezethető az alaptípusokra. A számokat is kétfelé szokás bontani, a különböző memóriabeli tárolási elv alapján. Vannak az egész számok, amelyek nem rendelkeznek tizedesrésszel, és vannak a valós számok, amelyekben szerepel tizedesrész (tizedes határoló utáni számjegyeket tartalmaz). Az informatikai elnevezés tehát különbözik a matematikában megszokottól (ott az egész számok egyben valós számok is).

Az NXT-G nyelv a v1.2-es változattól kezdve a tizedesrésszel rendelkező számokat is tudja kezelni. A korábbi *firmware* és az alacsonyabb verziószámú programváltozatok csak az egész számok kezelésére voltak alkalmasak. Érdemes tudnunk, hogy az általunk használt programváltozat képes-e a tizedesrésszel rendelkező számok kezelésére. Ha nem, akkor pl. az osztásaink eredménye is minden esetben egész szám lesz, mégpedig a tizedes rész elhagyásával képzett szám. Pl.: 7 : 2 = 3 (ha nem kezeli a programunk a tizedes törteket), és 7 : 2 = 3,5 (ha kezeli a programunk a tizedes törteket).

A típusa mellett a másik fontos információ egy adatról, hogy milyen értékhatárok között képes a rendszer kezelni, tehát mekkora lehet pl. a maximális értéke. A számok esetében a memóriabeli adattárolás 32 biten történik. Ez azt jelenti, hogy a legkisebb szám $-2^{31} = -2147483648$, míg a legnagyobb $2^{31} - 1 =$

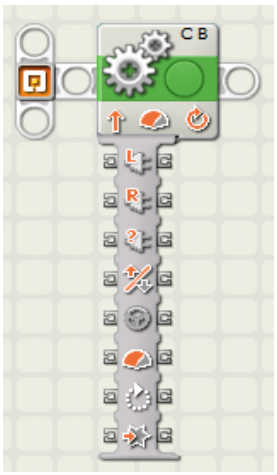
2147483647. Néhány esetben a használható számtartomány korlátozott, a paraméter jellegének megfelelően pl.: a motor sebessége – 100 és 100 között változhat (a negatív érték ellentétes irányú forgást eredményez).

Mivel a robot programozása során szöveges adattípusokat ritkábban használunk, így ezek mérete is erősen korlátozott. Néhány esetben fordulhatnak csak elő, pl.: a képernyőre íráskor. Ezekben az esetekben általában 58 karakter a maximális hossza a használható karaktersorozatoknak. A fájlnevek esetén 15 karakter.

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 felel meg.

Az adat típusától függ, hogy milyen művelet végezhető vele (matematikai összeadás, karaktersorozatok összefűzése, stb.). Nem keverhetők az egyes adattípusok, tehát nem adhatunk számhoz szöveget, vagy nem fűzhetünk össze logikai értéket számmal.

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 legtöbb modul bal alsó részére kattintva legördül egy csatlakozási pontokat tartalmazó rész. Ezeken keresztül történhet meg az értékek modulok közötti cseréje.

A *Move* ikon esetén láthatjuk ezt az ábrán.

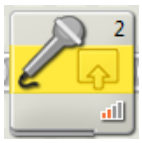
Vannak olyan modulok, amelyeknek nincs ilyen paramétercsatlakozási listája. Pl. a *Common* csoportba tartozó *Wait* modul.

Az eddigi programozás során a szenzorokat a *Wait* modulon keresztül, vagy a ciklusok, elágazások közvetlen bementi beállításaként használtuk. A szenzorok használatának van egy másik módja is (a szenzorhasználatot bemutató fejezetben már utaltunk rá).

Valamennyi alapszenzorhoz tartozik egy-egy modul a *Sensor* csoportban is. Ezek azok az ikonok, modulok, amelyek segítségével a mért értékek lekérdezhetők és továbbíthatók más programrészek felé.

Paraméterátadáson tehát a programban használt értékek (szám, szöveg, logikai) modulok közötti átadását értjük. A magas szintű programnyelvekkel foglalkozó szakirodalom helyenként szűkebb értelemben említi a paraméterátadás fogalmát, és több különböző esetet is megkülönböztet. Az NXT-G nyelv szempontjából mi egyféle elnevezést használunk, és a modulok közötti adattovábbítást értjük alatta.

Néhány alapszenzor ikonjait tartalmazza a következő táblázat.

Szenzor típusa	Alapszenzorok ikonja a *Common* csoport *Wait* moduljánál	Alapszenzorok ikonja a *Sensor* modulban
Fényszenzor		
Ütközésérzékelő		
Hangszenzor		
Ultrahangszenzor		
Színszenzor		

A kétféle ikoncsoportba tartozó modulok funkciója között alapvető különbség van. A *Wait* blokk ikonjain egy-egy homokóra is látható. Ez arra utal, hogy a megírt program futtatása során nem lép tovább a program végrehajtása, amíg a beállított feltétel nem teljesül, míg a *Sensor* csoport ikonjai a szenzorok által mért értékek pillanatnyi lekérdezését teszik lehetővé, itt tehát nincs várakozás. A különbségre a homokóra jelölésen kívül az eltérő szín is felhívja a figyelmet.

Például az alábbi két program működése között alapvető különbség van.

Az első esetben a motorok (B és C) elindulnak, és a második ikonnál vár a program mindaddig, amíg az ütközésérzékelőt be nem nyomjuk, majd fordul a robot, és vége a programnak.

A második esetben a motorok (B és C) elindulnak, de mivel a második ikonnál nem várakozik a program, rögtön elkezd fordulni, és utána leáll. Az egyenes mozgást nem is látjuk a kipróbálás során, az elég gyors végrehajtás miatt, tehát látszólag rögtön a fordulással kezdődik a program. Ebben az esetben az első két ikonnak nincs is szerepe, hiszen a végtelen előrehaladásnál értelemszerűen nincs várakozás, míg a második ikon esetében ugyan lekérdeztük az ütközésérzékelő állapotát, de mivel nem használtuk fel a kapott értéket, ezért szintén nincs szerepe.

A *Sensor* csoport ikonjai tehát visszaadják a szenzorok által aktuálisan mért értékeket, amelyek továbbíthatók más programmodulok felé. Az értékvisszaadás azt jelenti, hogy az aktuálisan mért érték a modul legördíthető paraméterlistájának a megfelelő csatlakozási pontjára kerül, ahonnan azt ki tudják olvasni más modulok. Természetesen ha az értéket nem íratjuk ki a képernyőre, akkor a felhasználó nem is ismeri ezt. Programozási szempontból nem is kell, hogy ismerje, elég ha a robot fel tudja használni a program futása során. Ezzel elkerülhető szenzorok értékeinek programírás előtti vizuális meghatározása például a robot képernyőmenüjének *Wiev* funkcióján keresztül.

Már csak egyetlen kérdés maradt, hogyan lehet a lekérdezett értékeket felhasználni, átadni más moduloknak?

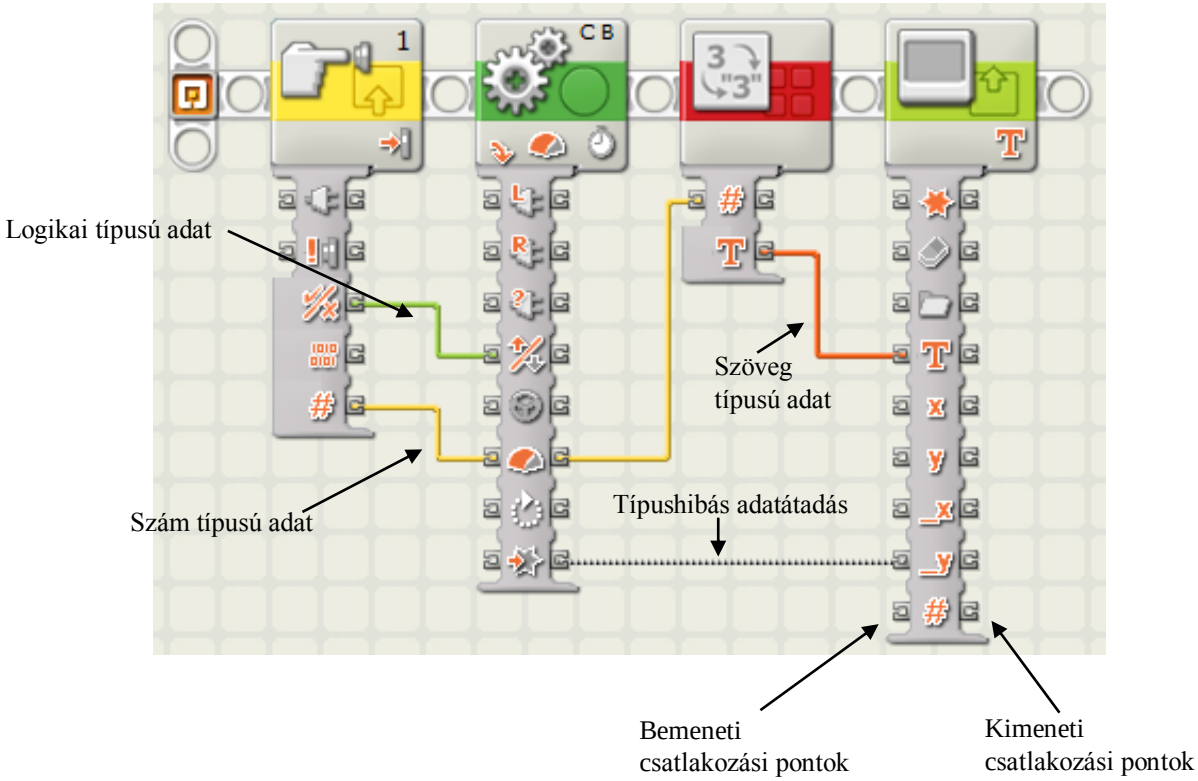
Ezt a célt szolgálja a paraméterátadás. A programjaink megfelelő modulja lekérdezi például a szenzorok értékét, majd a kapott adatot át lehet adni más moduloknak az ikonok bal alsó részére kattintva, a legördülő listák megfelelő csatlakozási pontjait összekötve (az egér bal gombjának lenyomása mellett, egyszerű egérmozdulattal). Ekkor egy huzal (*wire*) jelenik meg a két összekötött csatlakozási pont között.

A huzal színe utal az adat típusára:

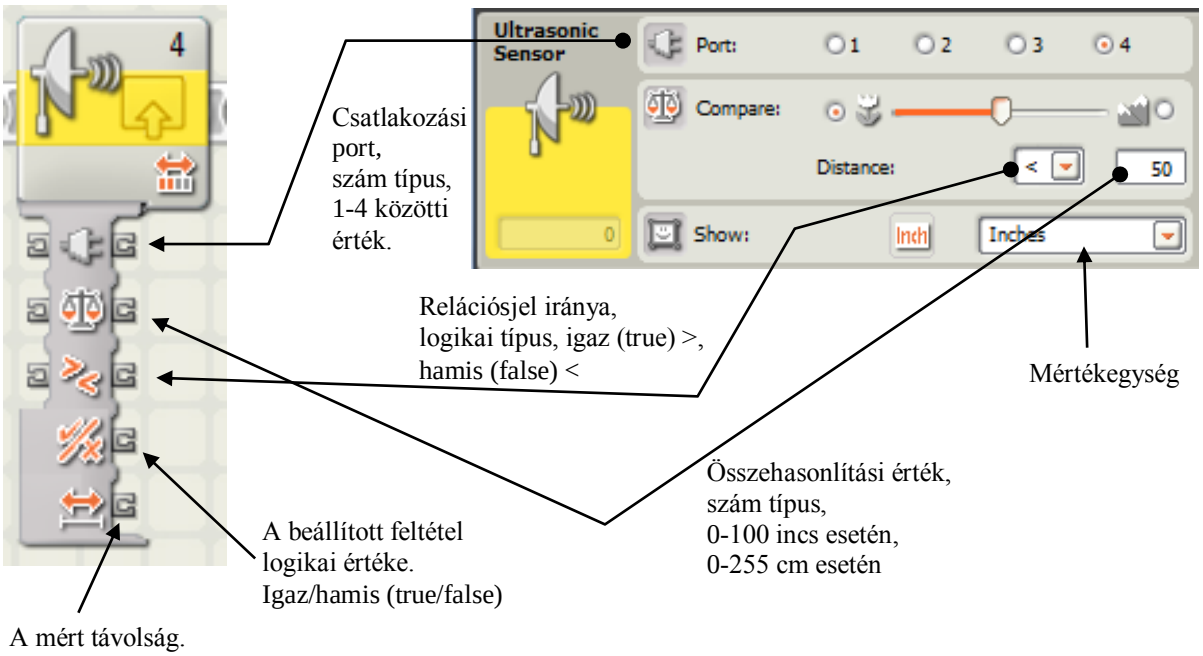
- szám típus esetén: sárga,
- szöveg típus esetén: piros,
- logikai típus esetén: zöld.

Csak a megfelelő típusú adatok átadására alkalmas pontok köthetők össze. Ha rossz (nem azonos) adattípusú pontokat próbálunk összekötni, akkor a huzal színe szürkére vált, és a programunk nem lesz futtatható mindaddig, amíg meg nem szüntetjük a rossz összekötést. A kijelölés után a huzal a DEL billentyűvel törölhető.

Egy kimeneti csatlakozási pontból több huzal is kivezethető, és 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, érvényesnek.



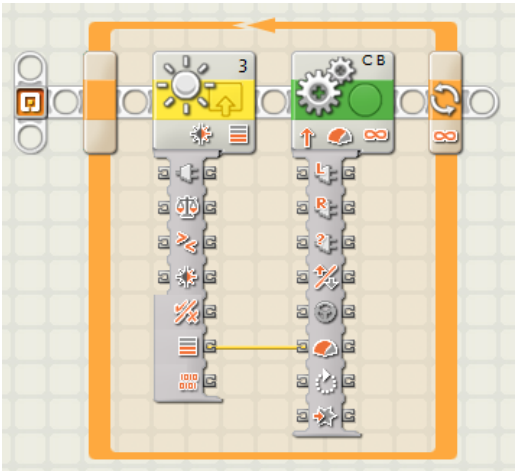
A legtöbb modulnak tehát kétféle paraméterezési lehetősége is van. Egyrészt a modulra kattintva a képernyő bal alsó részén megjelenő paraméterlista, ahol manuálisan (esetleg egérrel) tudjuk beírni, beállítani az értékeket, másrészt a legördíthető paraméterlista, ahová paraméterátadással juthatnak el az értékek. A legördíthető listán valamennyi fontos paraméter csatlakozási pontja megjelenik, amely a bal alsó részen látható (általában ugyanazzal a szimbólummal jelölve), de sok esetben több csatlakozási pont is szerepel. Ha mindkét helyen állítunk be értéket, akkor a legördíthető listán beállított lesz az érvényes. Pl.: az ultrahangszenzor esetén a megfeleltetés:



A mértékegységet tehát csak a bal alsó sarokban megjelenő paraméterlistán lehet beállítani, míg a beállított feltétel logikai értékét (tehát, hogy igaz vagy hamis), illetve az aktuálisan mért távolságot csak a legördíthető paraméterlistán tudjuk lekérdezni. A két listát bonyolultabb programoknál együtt érdemes használni.

A modulok esetén használható legördülő paraméterlista elemeiről, azok adattípusairól, valamint a használható értéktartományokról a keretprogram súgójában találhatunk egyszerű táblázatos leírást.

7/P1. Írjon programot, amelyet végrehajtva a robot egy 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. A lekérdezett értékek a modul legördülő paraméterlistáján az *Intensity* csatlakozási pontnál jelennek meg. Mivel a robot sebességét kell ezzel az értékkel változtatni, ezért az itt megjelenő adatot a *Move* modul legördülő listájának *Power* csatlakozási pontjához kell kötni. Így a motor pillanatnyi sebességét mindig az határozza meg, hogy a fényszenzor mit mér.

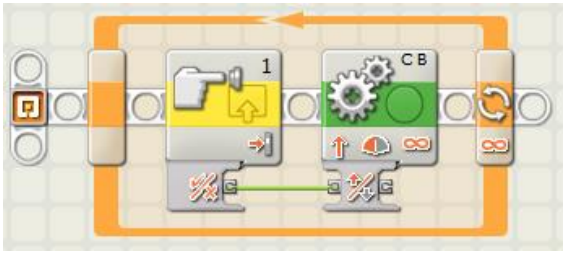
Ha egy modul paraméterátadással kap értéket, akkor a szerkesztő képernyő bal alsó részén, a modulhoz tartozó paraméterlistán nem számít, hogy mit állítottunk be az adott tulajdonságnál. Tehát jelen esetben a *Move* ikonnál figyelmen kívül hagyja a program, hogy a sebesség 75-re van állítva. Ne felejtsük el a motorok *Duration* paraméterét *Unlimited* értékre állítani, mert egyébként töredezett lesz a mozgása.

Reméljük ezzel az egyszerű példával a paraméterátadás technikáját sikerült szemléltetni. A komolyabb programok esetében a paraméterátadás elkerülhetetlen, ezért megértése nagyon fontos.

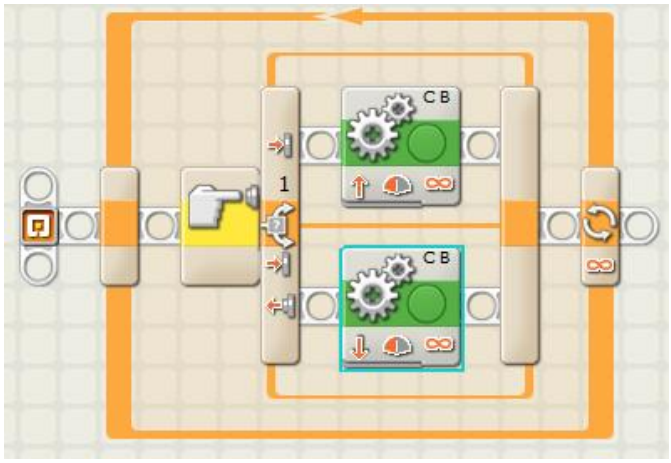
7/P2. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre 50-es sebességgel, ha nincs benyomva az ütközésérzékelője, és ugyanilyen sebességgel tolat, ha be van nyomva! Mindezt kikapcsolásig ismételje! (A feladat megegyezik a 6/P4-es példával.)

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 adjuk a *Move* modul megfelelő csatlakozási pontjára.



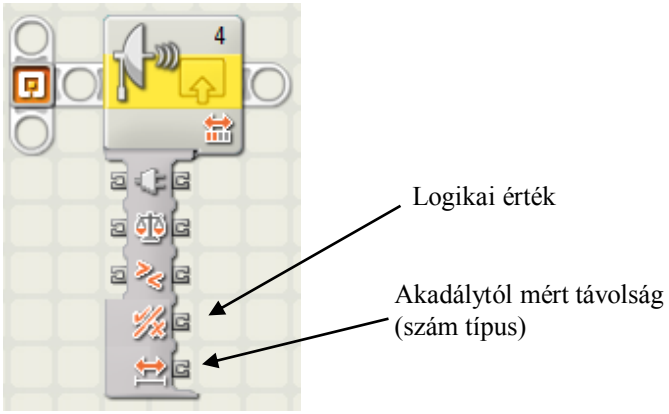
A program paraméterátadás nélkül is megoldható, egyszerű feltételes elágazás segítségével.



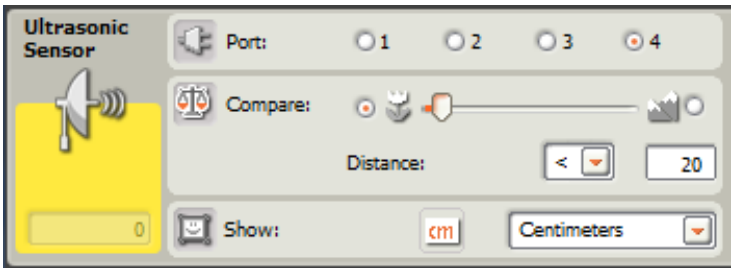
A szöveges típusú adatok átadására leginkább a képernyőre íratáskor van szükség, ezért az ilyen típusú paraméterátadással a képernyőkezelés fejezetben foglalkozunk.

Az előbbiekben hivatkozott *Sensor* ikoncsoport egyes ikonjai esetében több olyan csatlakozási pont is van a legördülő paraméterlistán, amelynek használhatósága magyarázatra szorul. Valamennyit nem tudjuk bemutatni, de egy példán keresztül talán sikerül rávilágítanunk a használatuk lényegére.

Az ultrahangszenzor példáját vegyük alapul. A szenzorral mért érték egy szám, az adott akadálytól mért távolság cm-ben. A legördülő listában mégis van logikai típusú érték lekérdezésére utaló csatlakozási pont. Hogyan adhat vissza a szenzor logikai értéket?



Minden *Sensor* csoportba tartozó modulnak a legördülő paraméterlistáján kívül van paraméterezési lehetősége a szokásos módon is, tehát a keretprogram alsó részén megjelenő területen. Az Ultrahang szenzor esetén ez a következőképpen néz ki:



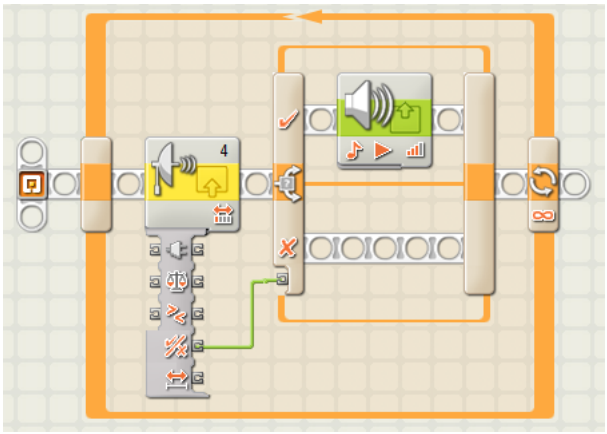
Itt a csatlakozási port mellett a távolságmérés mértékegységét (*Show*: incs vagy cm) is beállíthatjuk. A középső rész (*Compare*) területe szolgál arra, hogy megadjunk egy feltételt. Jelen esetben a mért távolság (*Distance*) kisebb, mint 20 cm. Ez lesz az a feltétel, ami a legördülő paraméterlista logikai értékét meghatározza. Ha a mért távolság 20 cm-nél kisebb, akkor a logikai érték igaz, egyébként hamis. Mindennek csak a logikai érték lekérdezésénél van szerepe. Más esetben a beállított érték nem befolyásolja a program működését.

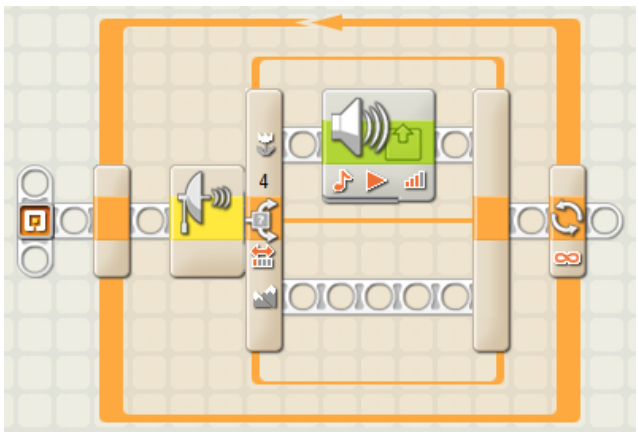
Sok programnál leegyszerűsíti, átalakítja a programszerkezetet, ha ilyen típusú paraméterátadást használunk. Például:

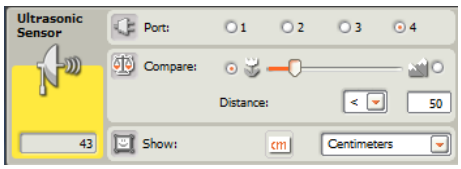
7/P3. Írjon programot, amelyet végrehajtva a robot mozdulatlanul áll! Ha az ultrahang szenzora 50 cm-en belül akadályt érzékel, akkor adjon hangjelzést! Ha nincs a távolságon belül érzékelhető akadály, akkor ne! (Mozgásérzékelős riasztó.)

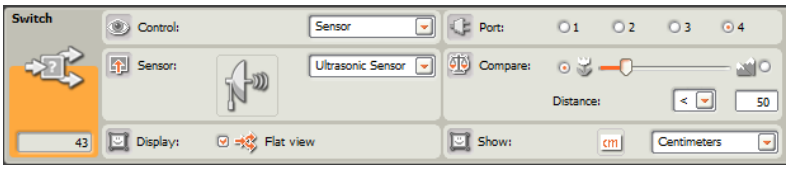
A program két megoldását mutatjuk be. Az elsőnél paraméterátadással, a másodiknál anélkül.

Mindkét program ugyanúgy működik, az elvi paraméterezése is megegyezik. A programozó döntése, hogy melyik megoldást választja. A döntést befolyásolhatja a hozzá kapcsolódó feltétel bonyolultsága, összetettsége. A bemutatott példák egyszerű feltételt tartalmaznak, így mindkét programszerkezet használható.



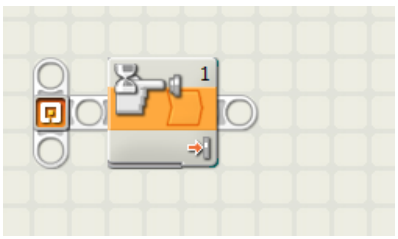






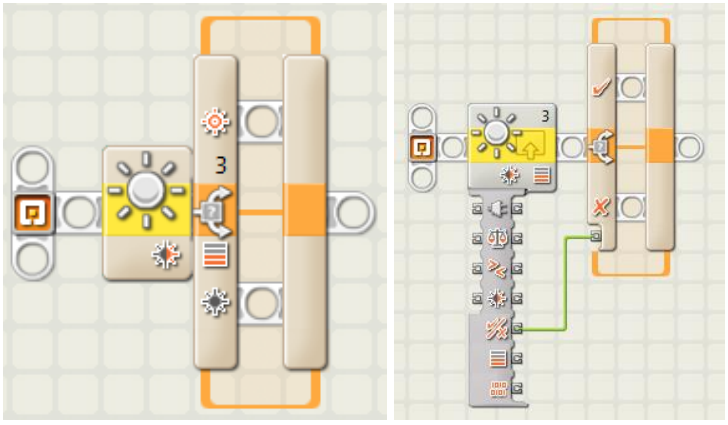
A második tűnik egyszerűbbnek. Van azonban egy lényeges hátránya: a második esetben nem használhatunk összetett feltételt, míg az elsőben igen. Tehát ha egy újabb tényezővel szeretnénk bővíteni a hangadás feltételét, akkor az első esetben sokkal egyszerűbben megtehetjük. Pl. csak akkor adjon hangjelzést, ha az akadály 50 cm-en belül van és a robot hátrafelé mozog. Az ilyen összetett feltételekről egy későbbi fejezetben lesz szó. 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 célszerűbb.

A korábban bemutatott *Wait* modul gyakorlatilag egy ciklus és egy szenzorfigyelést megvalósító modul együttese. A következő két programszerkezet között nincs különbség:

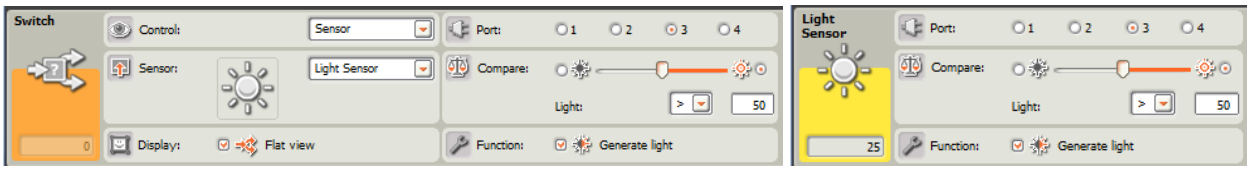


Mindkét program esetén az ütközésérzékelő benyomásáig vár a program a továbblépés előtt. Mivel sokszor kell ilyen utasítást használni programjainkban, ezért készítették el a programfejlesztők az egyszerűbb (egy ikonos) változatot. A többi szenzor esetén is van ugyanilyen megoldás. A feltételes elágazások esetén is létezik hasonló alternatíva.

Például a fényszenzor esetén:



A két megoldás itt is ekvivalens. Mindkét esetben 50-nél nagyobbra állítottuk be a fényszenzor által mért értékhatárt, ami az elágazás különböző szálai közötti választást szabályozza.



Az elágazásokat, illetve a ciklusokat is vezérelhetjük tehát paraméterátadással, logikai feltételekkel. Az elágazás (*Switch*) illetve ciklus (*Loop*) paraméterlistájánál a *Control* paraméter értékét, ciklusoknál – *Logic*, elágazásoknál – *Value* értékre állítva megjelenik a csatlakozási pont, ahová bármilyen logikai értéket (igaz/hamis) szolgáltató kifejezés csatlakoztatható. Ha egyszerű, egyetlen szenzorérték által szabályozott feltétel szükséges, akkor használjuk a rövidített formát (kényelmesebb), ha viszont a feltételeink összetettebbek, akkor a fentebb bemutatott paraméterátadás lehet a helyes megoldás. A paraméterátadással vezérelt ciklusokra és elágazásokra a későbbiekben sok példát fogunk mutatni.

7.3. Változók

A paraméterátadás tehát nagyon sok szituációban használható, azonban vannak olyan esetek, amikor nem tudjuk a mért értéket a megfelelő modulnak közvetlenül átadni. Például nem köthető össze huzallal két modul, ha az egyik cikluson vagy elágazáson kívül, a másik pedig belül van. Előfordulhat az is, 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. 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.

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

névvel 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 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.

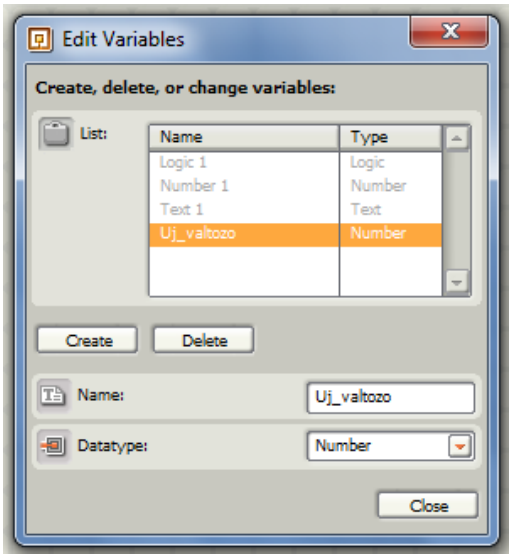
Az NXT-G nyelvben a programírás során háromféle változótípus használható.

Number – előjeles szám

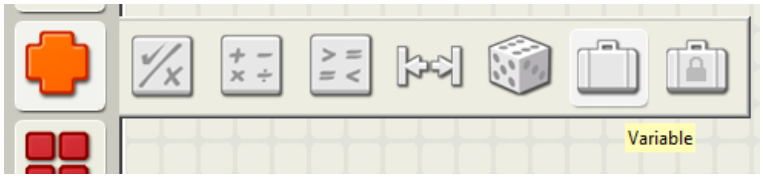
Text – szöveg típus.

Logic – logikai típus (igaz/hamis).

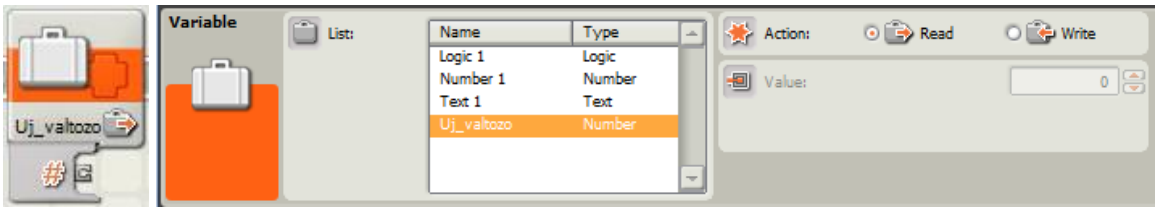
Alapértelmezésben három változó áll rendelkezésre, minden típusból egy-egy. Továbbiakat az *Edit/Define Variables* menüben tudunk készíteni, a név és típus megadása után. Változók nevében nem használhatunk ékezetes karaktereket és szóközt. Néhány további speciális karakter használata sem megengedett. 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.



A változók használatához a *Data* csoport *Variable* (bőröndöt szimbolizáló) ikonja használható.



A programokba illesztve első lépésként válasszuk ki a szerkesztőterület alsó részén megjelenő paraméterlistából a változó nevét. Itt minden olyan változó szerepel, amit létrehoztunk (*Edit* menü), valamint a kezdetben rendelkezésünkre álló három alapértelmezett is.



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 két lehetőség közül az *Action* paraméterterületen választhatunk.

Az olvasás során az adat benne marad a változóban, nem törlődik.

Ha írásra (*Write*) állítjuk a változót, akkor kaphat értéket úgy, hogy beírjuk a *Value* területen aktívvá váló szövegdobozba, vagy paraméterátadással. 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.

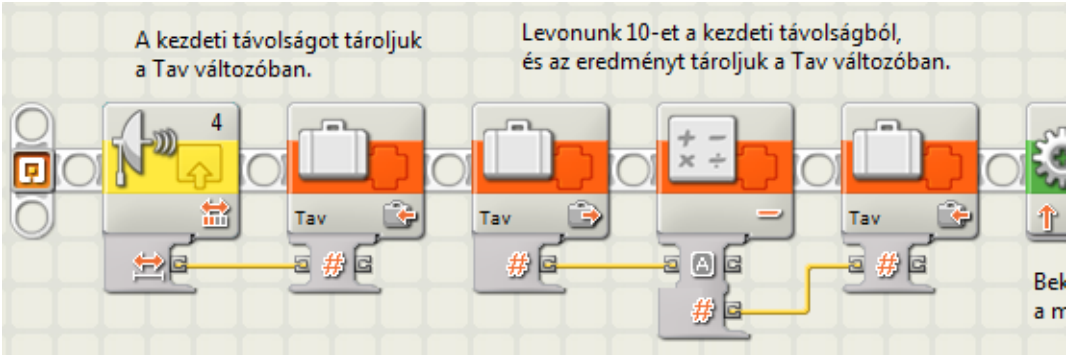
Egy változóba csak olyan típusú adat tehető be, amilyen a változó típusa. Tehát például *Number* típusú változóba nem tehetünk szöveget.

Néhány egyszerű példán keresztül bemutatjuk a szám (*Number*) típusú változók használatát.

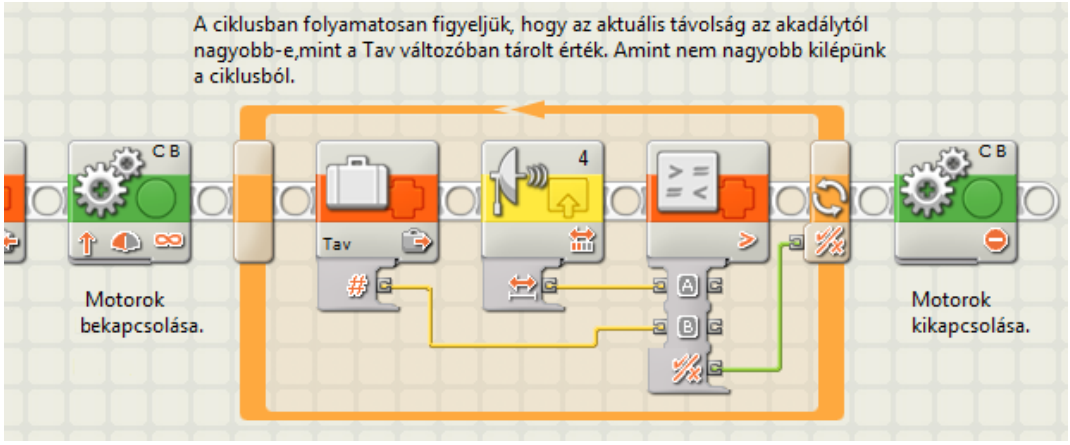
7/P4. Írjon programot, amelyet végrehajtva a robot valamekkora távolságra áll egy akadálytól (távolabb, mint 20 cm), majd 10 cm-rel közelebb megy az akadályhoz és ott megáll!

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 Tav nevű változóban. Ebből az értékből levon 10-et, és ezt szintén a Tav 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 Tav változóban tárolt szám. A programot két részre bontva mutatjuk be.

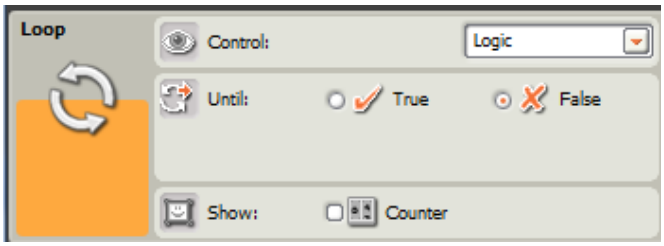
Az első rész a mérést és az adat tárolását, átalakítását szemlélteti. A matematikai művelet elvégzését lehetővé tévő modul bemutatását lásd a 9. fejezetben!



A második programrészletben kezdődik meg a mozgás (*Move* ikon). A mozgás befejezését egy feltétel fogja szabályozni, tehát *Unlimited*-re kell állítani a *Duration* paramétert. A cikluson belül folyamatosan lekérdezzük a távolságot, és összehasonlítjuk a Tav változó tartalmával.



A ciklus működésének végét egy logikai feltétel szabályozza. A feltétel: (Mért érték > Tav). Ennek a feltételnek vagy igaz (*true*) vagy hamis (*false*) az értéke. A ciklust úgy paraméterezzük, hogy a szerkesztőprogram alsó részén megjelenő paraméterlistájának *Until* területén a *False* értéket állítjuk be.



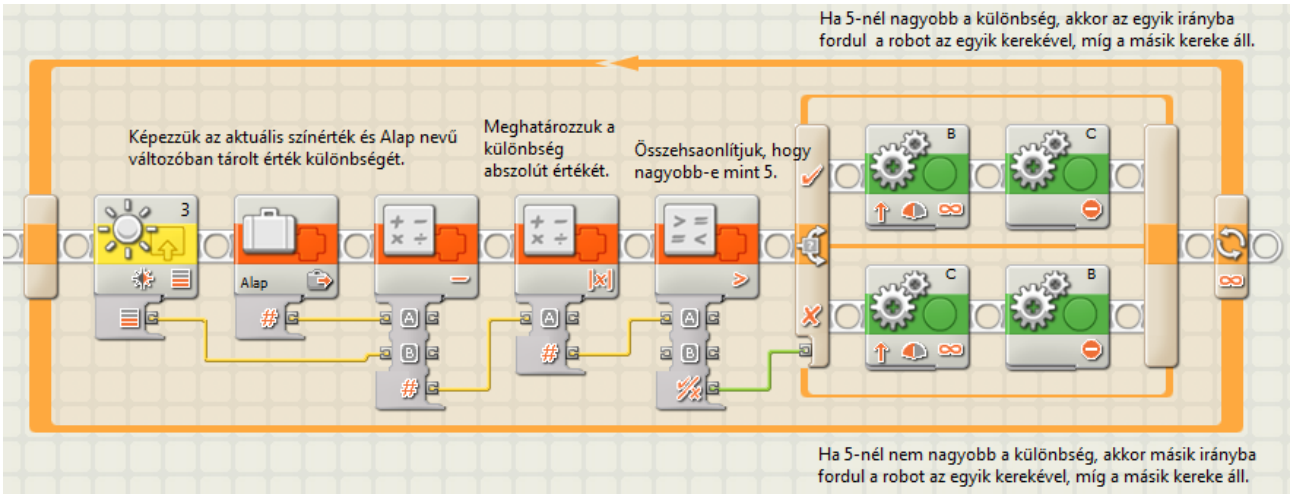
Az NXT-G programnyelvben használatos ciklusoknál ez a kilépési feltétel, tehát akkor marad abba az ismétlődés, ha a beállított feltétel hamis (*false*) lesz. Ebben az esetben ez azt jelenti, hogy mért érték már nem nagyobb, mint a Tav változóban tárolt, vagyis a robot a kezdeti távolsághoz képest 10 cm-rel közelebb került az akadályhoz.

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/P5. 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 és fekete alapon fehér vonal követésére. Ehhez még a robot elindulása előtt színmintát veszünk (a robot automatikusan végzi a mintavétel) 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 vonom 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.



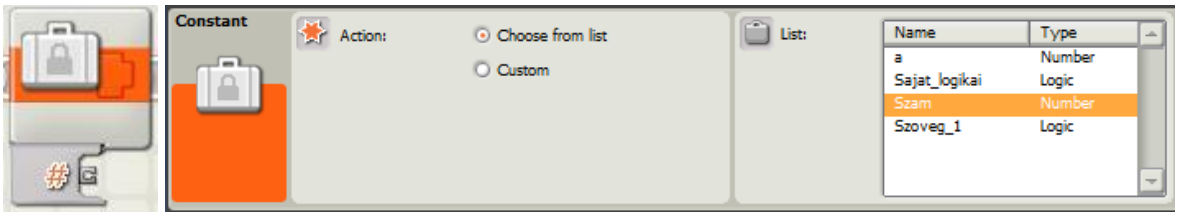
Az útvonalkövetés azon az elven működik, hogy amennyiben a kezdeti és az aktuálisan mért fényintenzitás egy határértéknél nagyobb mértékben eltér (esetünkben ez a határérték 5), akkor pl. balra fordul úgy, hogy az egyik motorját előre forgatja, míg a másik motor áll. Eltérő esetben a két motor szerepet 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 eltérés a jellemző, aszerint, hogy a fényszenzora éppen az útvonal fölött van vagy nincs.

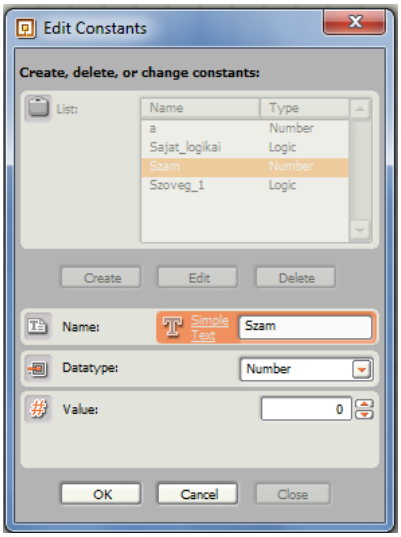
A matematikai modulok használatának magyarázatát lásd a 9. fejezetben!

7.4. 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 az *Edit* menü *Define Constants* menüpontjánál lehet, és használatukhoz a *Data* menü *Constant* modulját kell a programba illeszteni. Ha az *Edit* menüben létrehozott konstansokat szeretnénk használni, akkor az *Action* paraméter értékét *Choose from list*-re kell állítani. Ebben az esetben megjelenik egy lista a rendelkezésre álló konstansok nevével.





A konstansok létrehozásakor a *Data Type* paraméternél választhatjuk ki a típust, és a *Value* paraméternél állíthatjuk be az értéket.

7.5. 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!

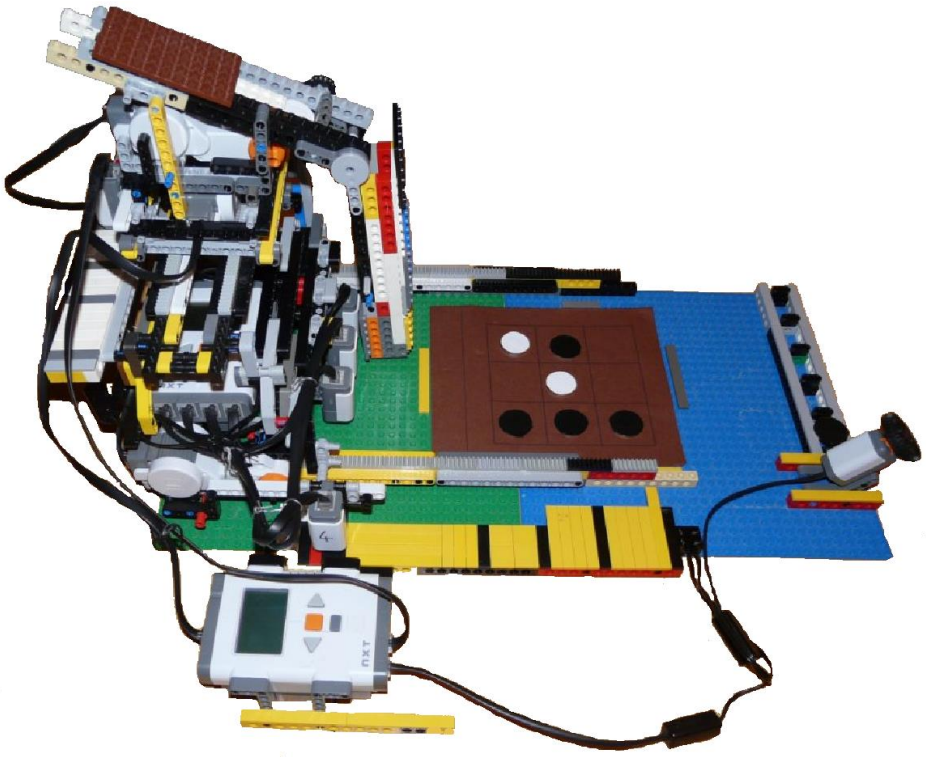
7/F3. Hozzon létre egy szám típusú változót és tárolja el benne a robot elindításakor a fényérzékelő által mért értéket! A robot ezután haladjon előre egyenesen mindaddig, amíg ennél az értéknél 3-mal kisebb értéket nem mér a fényérzékelő, ekkor álljon meg!

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 az 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!



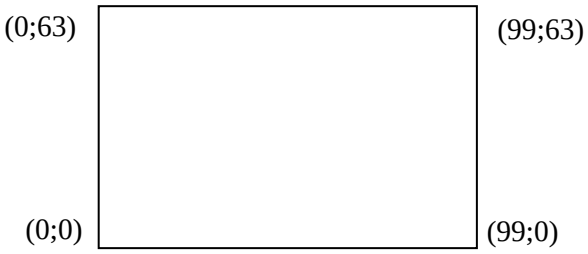
Emberi ellenfél ellen TicTacToe játékot játszó robot.

8. KÉPERNYŐKEZELÉS

A programozás során az adatok és információk megjelenítésre használhatunk egy egyszerű, 6400 képpontból álló, kétszínű grafikus képernyőt.

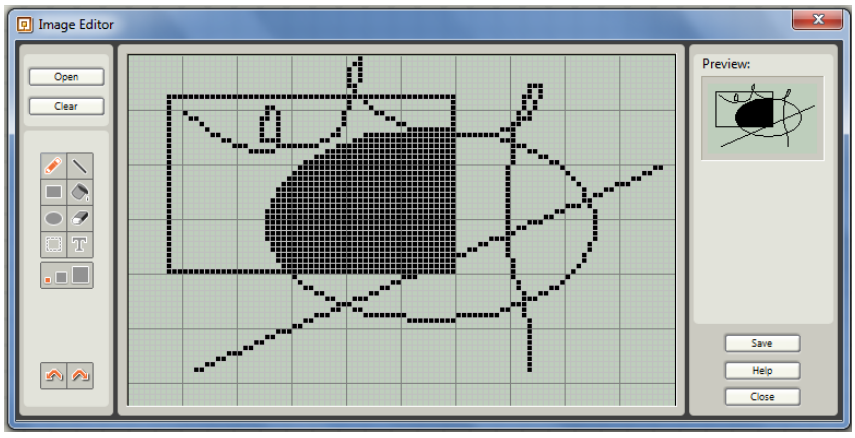
Sok programozási környezetben rendelkezésre állnak nyomkövetési lehetőségek, amelyek segítségével a programunkat lépésenként futtathatjuk, és a képernyőn figyelhetjük az egyes változók tartalmának módosulását. Az NXT-G programkörnyezetben ilyen lehetőség nincs, de helyettesíthetjük a kreatív képernyőkezeléssel.

Nagyon hasznos lehet például a programok tesztelése során az egyes változók értékének kiíratása, de mivel grafikus képernyőről van szó egyszerű ábrák, rajzok készítésére is alkalmas. A képernyő egy LCD alapú 100x64-es fekete-fehér pontmátrix grafikus megjelenítő. A bal alsó sarok koordinátái a 0;0, míg a jobb felső sarok koordinátái 99;63.



A képernyőre írhatunk szöveget, számokat, rajzolhatunk egyenest, kört, téglalapot és pontot, valamint megjeleníthetünk *ric* kiterjesztésű képfájlt. Az alapszoftverrel rendelkezésünkre bocsátanak néhány egyszerű piktogramot, amelyet felhasználhatunk programjaink látványosabbá tételéhez (*LEGO MINDSTORMS NXT/engine/Pictures* mappában).

Saját magunk is szerkeszthetünk ilyen egyszerű képeket, ábrákat a rendelkezésre álló szerkesztőprogrammal: *Tools/Image Editor...* menüpont aktiválásán keresztül.

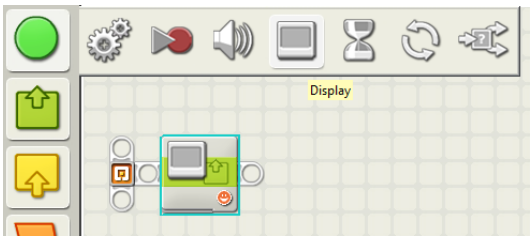


Mivel a program használata nagyon egyszerű, ezért nem mutatjuk be részleteiben. Pixelenként lehet szerkeszteni a képet két színben. Az elkészült ábrát *ric* formátumban tudjuk menteni. Ha a program alapértelmezett képmappájába mentjük, akkor rögtön fel is használhatjuk a programjainkban.

8.1. A képernyő programozása

A képernyőre kerülő karakterek 8 pont magasak, tehát a képernyőn összesen 8 sornyi szöveget lehet megjeleníteni (egymásra írás nélkül).

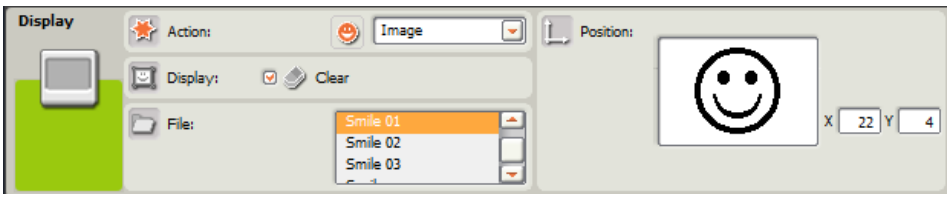
A képernyőkezelést megvalósító modul a *Common* utasításcsoportban található *Display* ikonon keresztül érhető el. Alapértelmezésben egyszerű képek megjelenítését tudjuk beállítani.



Az egyes paraméterek jelentését tartalmazza a következő táblázat.

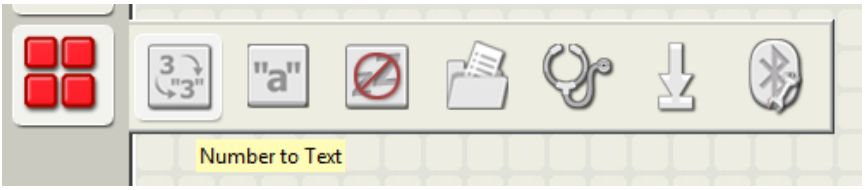
A paraméter neve	A paraméter jelentése
Action	Kiválasztható, hogy a képernyőn milyen típusú adatot szeretnénk megjeleníteni. *Image* – *ric* formátumú és kiterjesztésű piktogram. *Text* – szöveg. *Drawing* – geometriai alakzat. Ebben az esetben a megjelenő *Type* paraméternél állíthatjuk be, hogy pont (*Point*), szakasz (Line), vagy kör (*Circle*) legyen-e a grafikus ábra. *Reset* – képernyőtörlés.
Display	Egy jelölőnégyzettel tudjuk megadni, hogy a következő rajzolási művelet előtt letöröljük-e a képernyőt? Ha a *Clear* paramétert bejelöljük, akkor a beállított képernyőtartalom üres képernyőn fog megjelenni. Ez a paraméter teszi lehetővé, hogy több grafikus ábra vagy szöveg is megjeleníthető legyen egyidőben a képernyőn. Ha a *Clear* paramétert nem jelöljük be, akkor képernyőtörlés nélkül jelennek meg a beállított tartalmak.
File vagy Text vagy Type	A paraméter megjelenése az *Action* opciónál választott típustól függ. Ha ott *Image* a beállított érték, akkor itt tudjuk kiválasztani a megjelenítendő *ric* kiterjesztésű fájlt (*File*). Ha az *Action* értékét *Text*-re állítottuk, akkor itt adhatjuk meg a megjelenítendő szöveget (*Text*). Ha a beállított érték *Drawing*, akkor itt választhatjuk ki a rajzolni kívánt grafikus objektum típusát. Lásd később. Ha az *Action* opciónál *Reset* a beállított érték, akkor ez a paraméter nem jelenik meg.
Position	Itt adhatjuk meg a megjeleníteni kívánt objektum képernyő-koordinátáit. Valamint egy mintaképet is láthatunk a leendő képernyőről.

Ha az *Action* paraméternél az *Image* funkciót választjuk, akkor a *File* paraméterlistáról választott kép megjelenik a paraméterlistán szereplő minta ablakban is. A minta ablak mellett állíthatjuk be a kép megjelenési pozícióját. Ez a bal alsó sarok koordinátáinak megadását jelenti, akár beírva az X illetve Y jelölésű szövegdobozokba, akár az egérrel megfogva a képet és a megfelelő helyre húzva. A kép minden esetben téglalap alakú, és a vízszintes koordináta az X, a függőleges koordináta az Y.



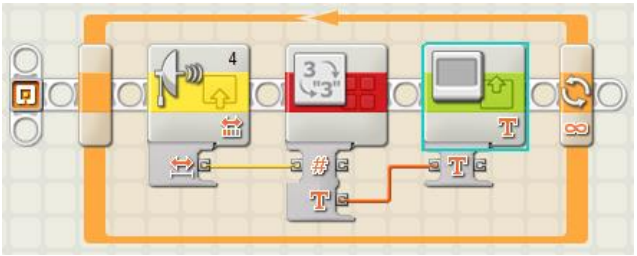
A bemutatott esetben a kép bal alsó sarkának koordinátái vízszintesen 22, függőlegesen 4.

A felsorolt megjeleníthető tartalmakból látszik, hogy csak szöveges formátumú adatokat tudunk a képernyőre íratni. Tehát a szám típusú adatokat előbb át kell alakítani szöveges típusúvá. 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. Ehhez rendelkezésre áll egy átalakító, konvertáló programelem (*Advanced* programcsoport, *Number to Text* ikon).

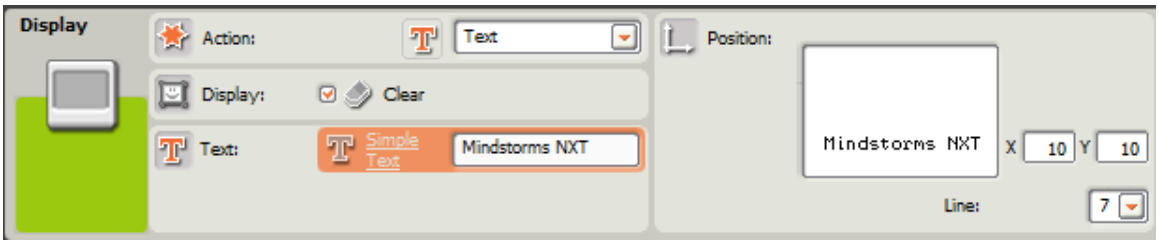


A szám típusú adatot minden esetben ehhez a programelemhez kell kapcsolni (paraméterátadás), majd a kimenetét lehet a képernyőikon bemeneteként használni.

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



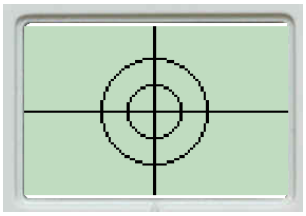
A 4-es portra csatlakoztatott ultrahangos távolságérzékelő által mért értéket kapja meg az átalakító modul, amely a számértéket szöveges formátumúvá alakítja és átadja a képernyőkezelést megvalósító modulnak. A képernyőkezelő modul paraméterezése:



A megjelenítés Text formátumú, a képernyő 10;10 koordinátájú pontjától kezdve írja ki az adatot (szemléltetésként a „*Mindstorms NXT*” szöveg jelenik meg a paraméter-képernyőn). Minden kiíratás előtt képernyőtörlés történik (bekapcsolt *Clear* paraméter), mivel a kevesebb helyiértékből álló számok után továbbra is a képernyőn maradnának a további számjegyek. A 10;10 koordinátájú hely közelítőleg a 7. karaktersornak felel meg (egy sor 8 pixel magas, és a képernyő legfelső 8 pixelnyi sora az 1-es sorszámú, tehát összesen 8 sornyi szöveg jeleníthető meg a képernyőn.

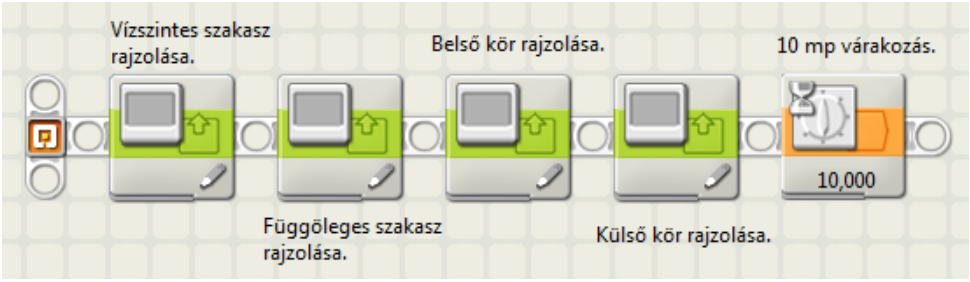
A képek és szöveg megjelenítése mellett egyszerű grafikai objektumokból összeállított rajzok is megjeleníthetők a képernyőn.

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



Mindezt négy rajzelemből építi fel. 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:

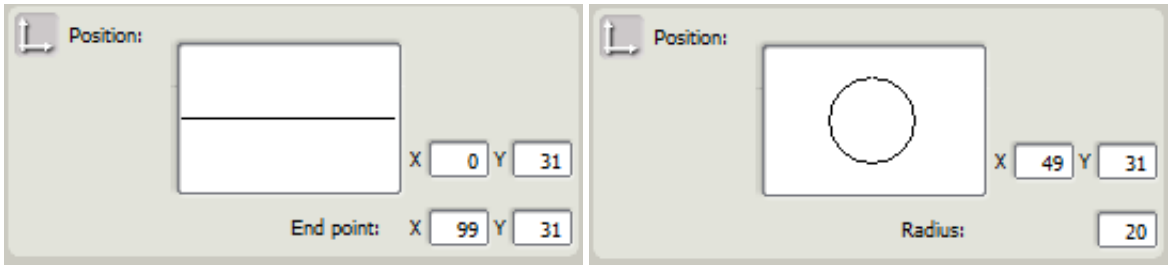
Vízszintes szakasz: 0;31 → 99;31

Függőleges szakasz: 49;0 → 49;63

Belső kör középpont: 49;31 sugár: 10

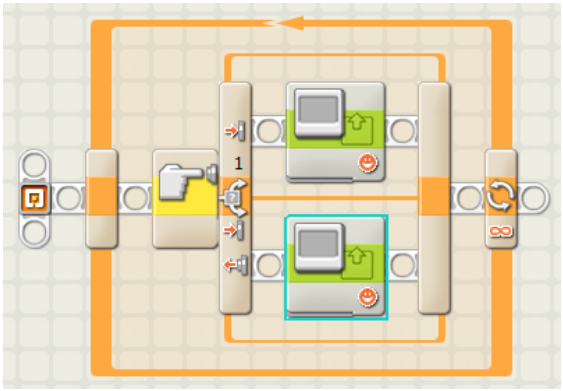
Külső kör középpont: 49;31 sugár: 20

Pl.:



Egy érdekes, de nem túl bonyolult program készíthető korábban bemutatott programelemek felhasználásával.

8/P3. Írjon programot, amelyet végrehajtva a robot egy mosolygó smileyt rajzol a képernyőjére, ha az ütközésérzékelője nincs benyomva, és egy szomorú smileyt, ha be van nyomva! Mindezt kikapcsolásig ismételje!



Az ütközésérzékelő az 1-es portra van csatlakoztatva. Az elágazás felső szálán szereplő képernyőmodul paraméterezése:

Az alsó szálon szereplő modul paraméterezése hasonlóan történik. A felhasznált két smiley a keretprogramban rendelkezésre álló *ric* típusú kép.

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 10 pixel sugarú kört! A kör a bal oldali képernyőszéltől a jobb oldali képernyőszélig egyenletes sebességgel (programozói beavatkozás nélkül) haladjon, majd ott álljon meg!

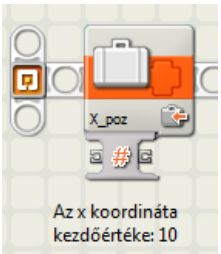
Mivel a kör sugara 10 pixel, ezért a középpontjának a koordinátája vízszintesen 10 és 89 között egyenletesen növekszik, tehát összesen legfeljebb 80 értéket vehet fel. A függőleges koordinátája pedig állandó, pl.: 31.

Egy 80-szor lefutó ciklust hozunk létre, amely minden egyes végrehajtás során eggyel növeli a vízszintes koordinátát. Erre alkalmas a ciklusváltozó, amelyhez 10-et adva (a kör középpontjának indulási pozíciója 10) már meg is kapjuk a kör középpontjának aktuális x koordinátáját. Ezt a paramétert adjuk át a képernyő modul legördülő listáján található *x* paraméternek. 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.

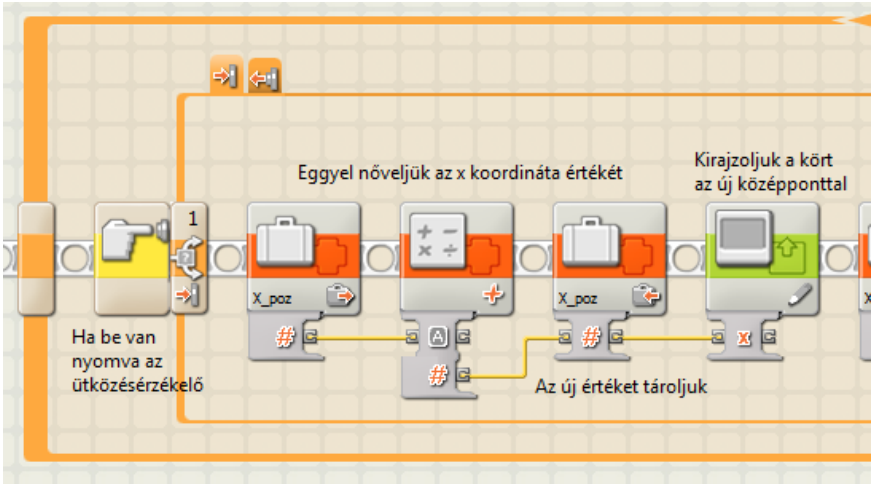
8/P5. Írjon programot, amelyet végrehajtva a robot ütközésérzékelőjének benyomásával szabályozható a képernyőn megjelenő 10 pixel sugarú kör vízszintes mozgása! Ha az ütközésérzékelő be van nyomva, akkor a kör x koordinátája folyamatosan 1 pixelenként növekszik (így a kör balról jobbra mozog). Ha a kör elérte a képernyő szélét (teljesen eltűnt a képernyőről), akkor újra jelenjen meg bal oldalt 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:



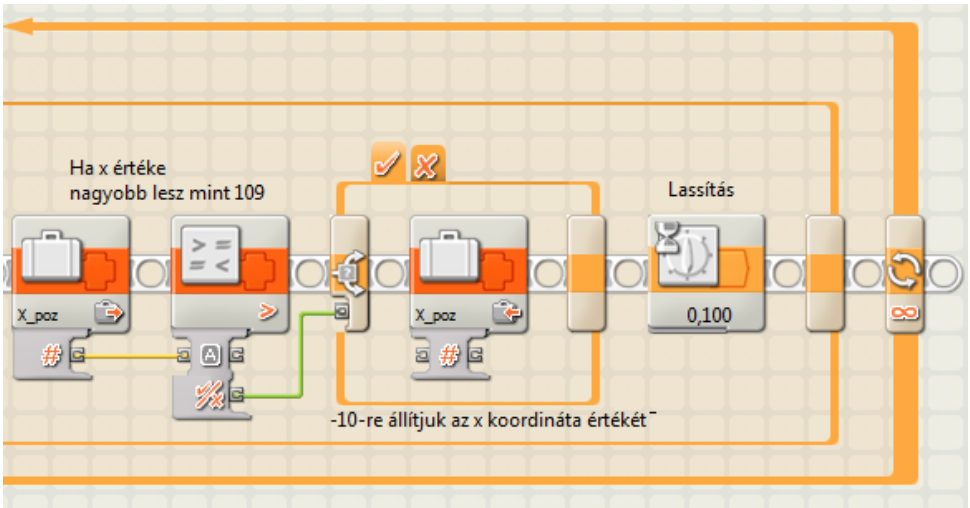
Létrehozunk egy X_poz nevű numerikus típusú változót. Ebben fogjuk a program során tárolni a képernyőn vízszintesen mozgó kör aktuális x koordinátáját. Mivel a kör középpontjának vízszintes koordinátája 10-nél kezdődik (így látszik a teljes kör a képernyőn), ezért az X_poz változó kezdőértékét a program elején 10-re 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 cikluslefutáskor az új értéket tudjuk tovább növelni. Az új értéknek megfelelő x koordináta-középpontú kört rajzoljuk a képernyőre. Minden rajzolásnál töröljük az előző ábrát és a középpont y koordinátája állandó, pl.: 31.

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 109 (99 a képernyő széle plusz 10 a kör sugara), akkor a változó értékét -10-re kell állítanunk, így folyamatosan beúszik balról.



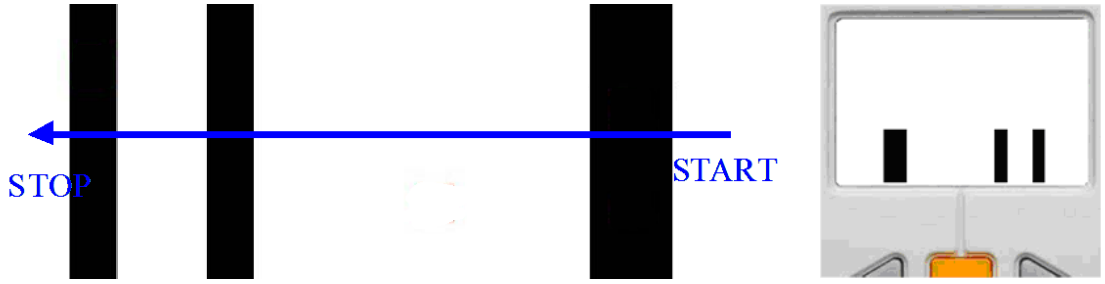
Végezetül a követhetetlenül gyors mozgás elkerülése érdekében egy 0,1 mp-es várakozást állítunk be, minden rajzolás után. Ezzel az elágazásnak vége és a ciklus újraindul.

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

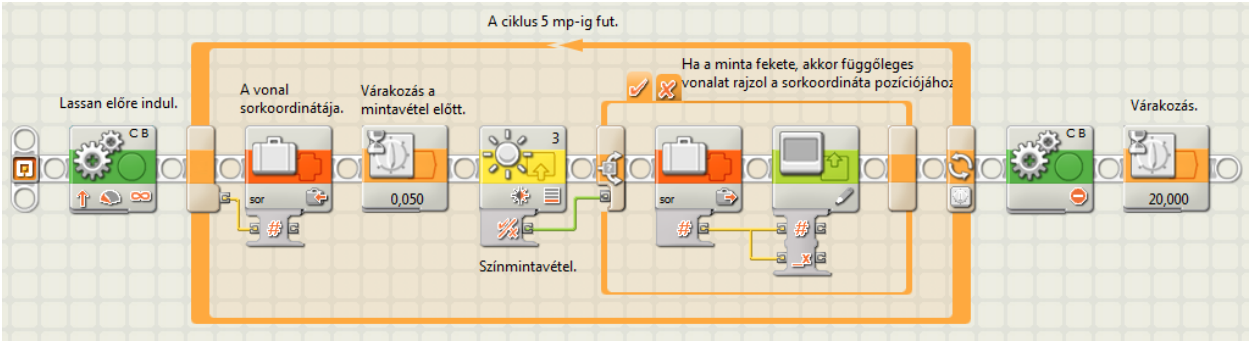
A program továbbfejlesztéseként a függőleges mozgás vezérlésére használhatunk egy másik ütközésérzékelőt, így a kör a képernyőn tetszőleges pozícióba mozgatható.

8/P6. Írjon programot, amelyet végrehajtva a robot egyenesen előre indul egy fehér felületen, és lassan (kb. 40-es sebességgel) halad egy fekete csíksor fölött 5 mp-ig. 0,05 mp-enként színmintát vesz fényszenzorával az éppen aktuális felületről. A képernyőre egy függőleges 20 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 100 db mérést végez, és ennek megfelelően rajzol vagy nem rajzol szakaszokat. Az 5 mp letelte után 20 mp-ig várakozik, majd utána ér véget a programja.

A pálya képe és a képernyőkép:



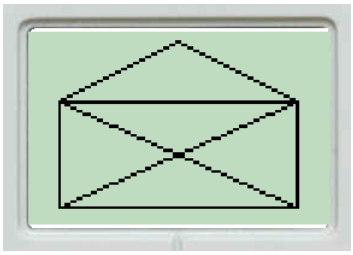
A feladat megoldása az eddigiek alapján nem bonyolult, ha sikerült megértenünk a korábbiakat.



A képernyőre egy függőleges, 20 pixel magasságú szakaszt rajzolunk, amelynek y koordinátája mindig 0 és 20, a vonal x koordinátája pedig minden ciklusleftáskor eggyel nő. Ezt a ciklusváltozó szabályozza és a Sor nevű változóban tároljuk. Ténylegesen rajzolni csak akkor kell, ha fényszenzor által mért érték fekete. A határértéket a programban tapasztalati méréssel határoztuk meg, és manuálisan 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 5 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/F!4. 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.)

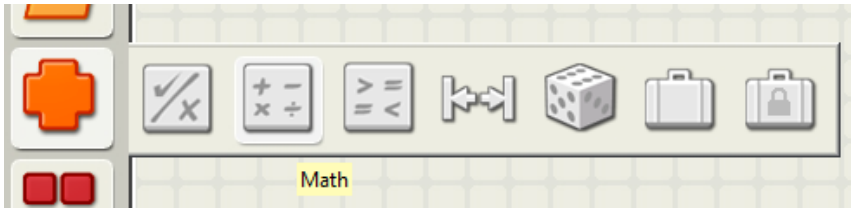
8/F6. Írjon programot, amelyet végrehajtva a robot egy játékszimulációt valósít meg! Két ütközésérzékelővel rendelkezik. Az egyikkel a képernyőn megjelenő 2 pixel sugarú kört vízszintes, a másikkal függőleges irányban lehet mozgatni. A mozgatás ciklikus, tehát ha a kör elérte a képernyő szélét, akkor 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!

9. MATEMATIKAI ÉS LOGIKAI MŰVELETEK

9.1. Műveletek csoportosítása

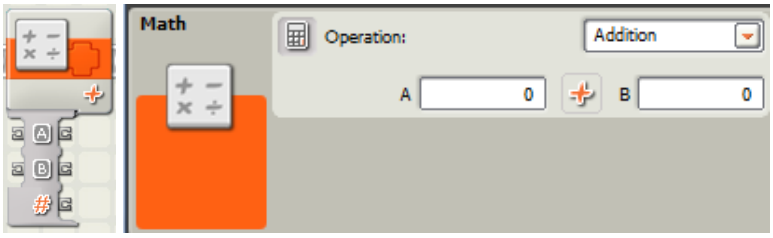
Már az eddigi programok során is előfordult, hogy néhány esetben használtuk az itt bemutatásra kerülő modulokat. Valamennyi a *Data* csoportban található. Alapvetően az adatokkal történő számolási műveletek elvégzését teszik lehetővé, valamint a kapott adatok összehasonlítását a matematikában megszokott 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, stb, 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. 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 utóbbi kettő egyváltozós, tehát egyetlen számon fejti ki hatását és egy újabb számot ad eredményül. Valamennyi művelet a *Data* csoport *Math* modulján keresztül érhető el.



Az *Operation* paraméternél 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 legördülő paraméterlistájának #-tel jelölt 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*).

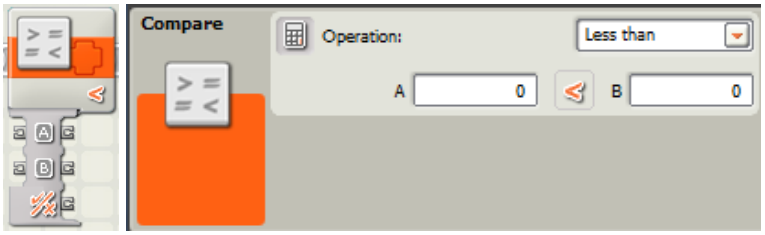
További számértéket visszaadó „művelet”, a véletlen számsorsoló.

Vagy paraméterátadással vagy a szövegdobozokba beírt számokkal megadhatjuk, hogy milyen tartományon belül állítson elő véletlen számot. Ha paraméterátadással vagy a szövegdobozba beírva adjuk meg a lehetséges számtartomány, 0 – 32767. A véletlen számsorsoló negatív számokat is tud sorsolni, de vagy csak negatív vagy csak pozitív számokat. A tartomány tehát megadható -32768 – 0 vagy 0 – 32767 formában, de nem lehet -32768 – +32767 formát használni. A paraméterlistán szereplő csúszkát használva 0-100 közötti érték állítható be.

9.3. Logikai értéket visszaadó műveletek

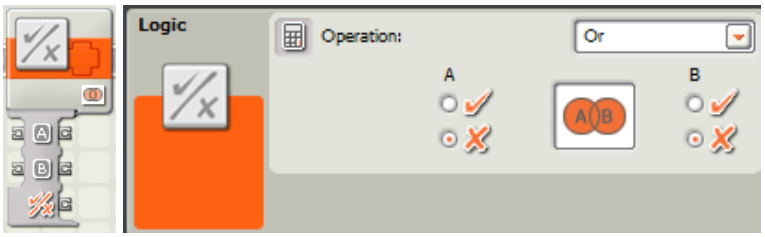
Szintén a *Data* csoporton belül találhatók a logikai értéket visszaadó műveletek, de több modulra bontva.

Összehasonlító műveletek (*Compare*)



Az *Operation* területen található legördülő lista elemei: *Less than* (<), *Greater than* (>), illetve *Equals* (=). A „kisebb egyenlő” illetve „nagyobb egyenlő” relációknak nincs megfelelő listaelem, de mivel ezek a nagyobb illetve kisebb relációk ellentétei, így használatuk megoldható ezek tagadásával, ellentettként. Valamennyi művelet két-két bemenő adatot igényel, amelyek típusa szám (*Number*) kell, hogy legyen.

Logikai műveletek (*Logic*)



Az *Operation* legördülő listából a 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 három logikai művelet 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 feltételek értéke. 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**

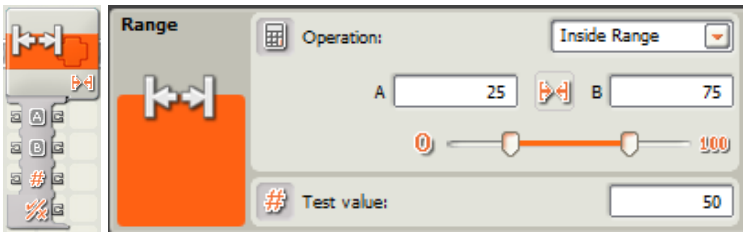
A „vagy” kapcsolat esetén akkor lesz az összetett művelet eredménye hamis, ha mindkét feltétel hamis volt, egyébként igaz.

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 adatot igényel és annak az értékét ellentettjére változtatja.

Intervallum művelet (*Range*)

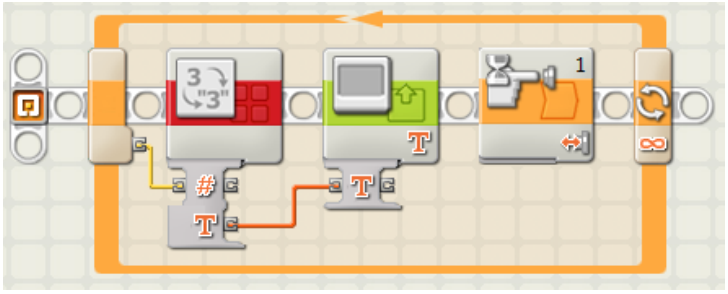


Két bemenő határadatot igényel a használata. Azt vizsgálja, hogy egy paraméterként kapott szám a megadott határadatokon belül (közöttük) vagy kívül van-e. Eredményül logikai értéket ad vissza aszerint, hogy az *Operation* paraméterterületen mit állítottunk be. A két lehetőség: *Inside Range* (belül) vagy *Outside Range* (kívül). *Inside Range* esetén a visszaadott érték akkor lesz igaz, ha a megkapott paraméter

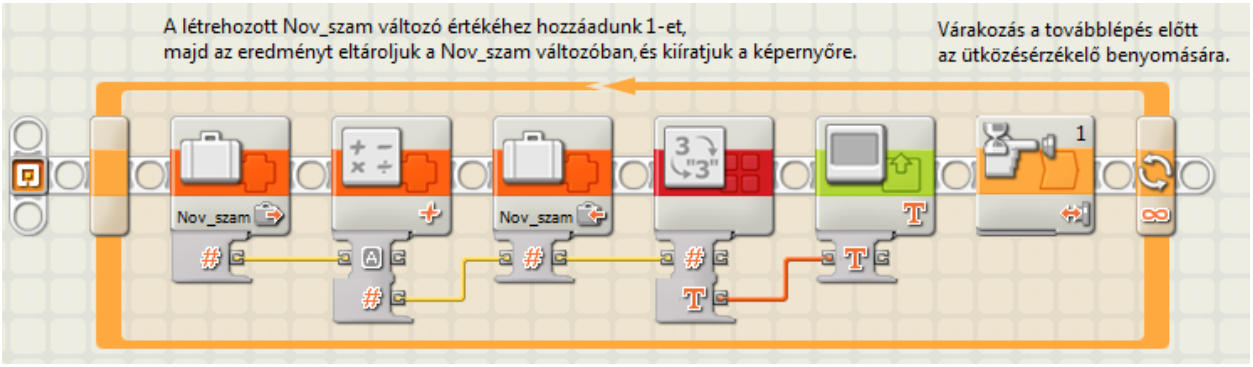
a határadatok között van. A vizsgált számot paraméterátadással vagy a *Test value* szövegdobozba írva adhatjuk meg. A képen szereplő példánál a visszaadott érték igaz (*True*), mert a 25 < 50 < 75 (a tesztadatként használt 50 a 25 és 75 között van). A modul használatával az abszolútértékhez hasonló programszerkezetek 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.



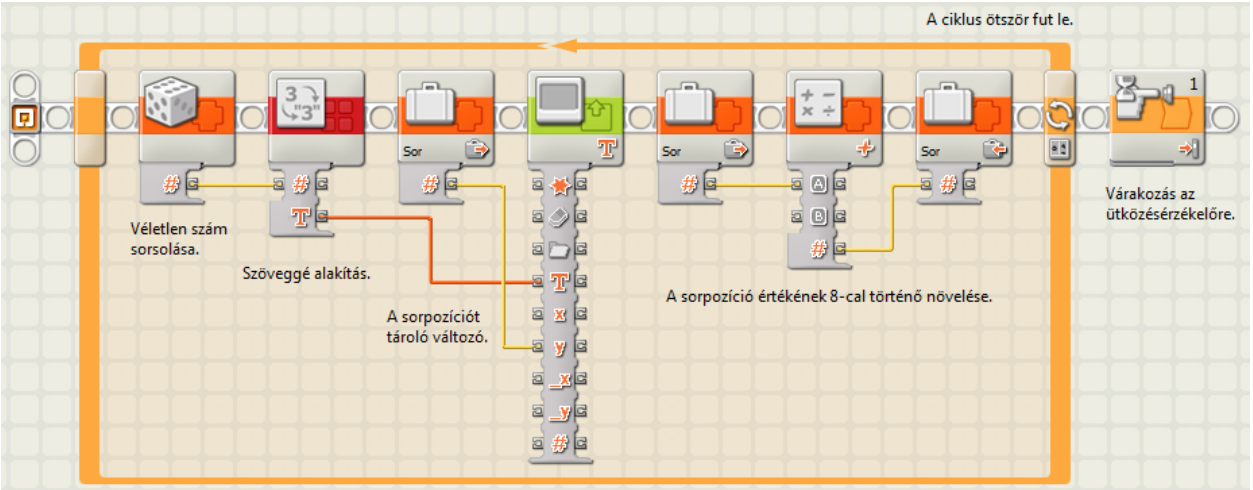
A ciklusváltozó értékét (minden cikluslefutáskor eggyel növekvő számsor) első lépésben szöveggé konvertáltuk, majd ezt írattuk ki a képernyőre. Az újrakezdés előtt egy ütközésérzékelőre történő várakozást állítottunk be. Ezen a modulon csak akkor lép túl a program, ha benyomjuk az érzékelőt. Az ütközésérzékelő paraméterezésénél figyelni kell arra, hogy az *Action* paraméter értékét *Bumped*re állítsuk. Ennél a paraméterértéknél nem az érzékelő benyomására ad igaz értéket a modul (lép túl rajta), hanem az állapotváltozás hatására: tehát ha benyomott állapotból kiengedett állapotba kerül a szenzor. Ha nem ezt a paramétert használjuk, hanem a szokásos *Pressed* (benyomva) értéket, 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.



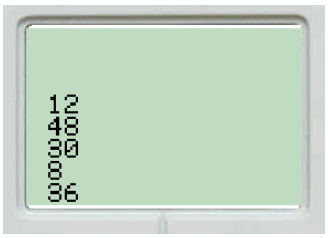
Egy változóban tároljuk folyamatosan a képernyőre kiíratandó számokat (0 kezdőértéktől). Minden ciklus lefutáskor eggyel növeljük a változó értékét, amit el is tárolunk ugyanabban a Nov_szam nevű változóban. Ezzel ugyan elveszítjük az előző értéket, de nincs rá tovább szükségünk. A számot ezután szöveggé alakítjuk, és kiíratjuk a képernyőre. Az ütközésérzékelőre történő várakozás ugyanaz, mint az első programnál. 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 előtt érdemes a Nov_szam változó kezdőértékét -1-re állítani, vagy a kiíratás után növelni az értékét.

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 szöveggé alakítás után a képernyőre írás történik meg, képernyőtörlés nélkül.

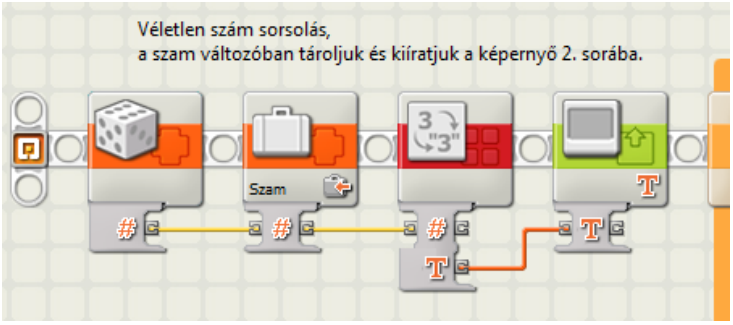


A Sor nevű változó tartalmazza a kiíratás helyének függőleges koordinátáját (y). Ezt az értéket adtuk át a képernyőkezelő modul megfelelő (y) csatlakozási pontjára. Ez kezdetben nulla. Mivel a sorok magassága a képernyőn 8 pixel (karakterek magassága), ezért a kiíratás után a változó értékét 8-cal növeltük. A ciklus 5-ször fut le, majd várakozik az ütközésérzékelő benyomására. A képernyőkép:



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!

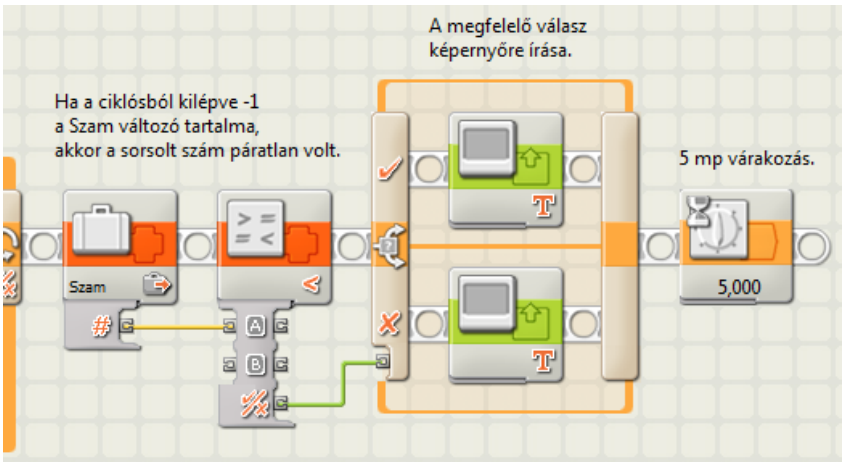
Az NXT 2-es szoftverváltozat már kezeli a tizedes törteket is. Tehát az osztás eredménye nem feltétlenül egész szám. Az NXT 1-es változat esetén az osztás után még mindenképpen egész szám volt az eredmény, mert a tárolásnál a tizedes részt a rendszer elhagyta. Így egyszerűen kihasználható volt ez a tulajdonság a páros/páratlanság eldöntésénél (pl.: ha a 2*szám/2 < a szám, akkor az eredeti szám páratlan volt, kicsit konkrétabban: ha a szám = 9, akkor 9/2 = 4,5, de ebből csak 4-et tárol a rendszer, így 2*4 = 8, ami kisebb, mint az eredeti 9, páros számok esetén a kettővel való osztás utáni visszaszorzás az eredeti számot adja eredményül). Tehát ha a számot elosztottam kettővel, majd a hányadost megszoroztam kettővel, akkor a kapott eredmény vagy kisebb lett, mint az eredeti szám (páratlan eset) vagy vele egyenlő (páros eset). Egyszerű összehasonlítással eldönthető volt a kiindulási szám paritása. A tizedes törtek kezelése ugyan hiányzott korábban a programból, de azzal, hogy ezeket már korrekt módon használja a program (NXT-G 2-es), a fenti páratlanság vizsgálat lehetőségét elveszítettük, hiszen a kettővel való osztás utáni hányados kétszerese mindenképpen meg fog egyezni az eredeti számmal. Sajnos a programkörnyezetben nem áll rendelkezésre sem olyan modul, amivel levághatnánk a tizedes részt a szám végéről, sem olyan modul, amely az osztás utáni maradékot lenne képes kiszámolni. Így egy „ravasz” algoritmust használunk a páratlanság eldöntésére, amely kivonásokkal határozza meg a kettővel való osztás utáni maradék -1 szeresét, és ez alapján dönt. A programot három részben mutatjuk be. Az első részben sorsolunk egy 1 és 100 közötti véletlen számot, ezt eltároljuk a Szam változóban, és kiíratjuk a képernyő 2. sorába (a legfelső az 1. sor). Nincs képernyőtörlés.



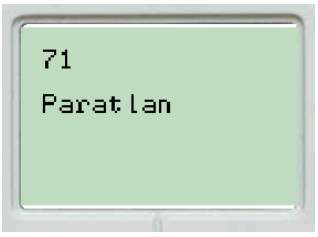
A 2. részben azt használjuk ki, hogy ha a kisorsolt számból elég sokszor kivonjuk a 2-t, akkor eredményül előbb-utóbb 0-t vagy -1-et kapunk aszerint, hogy páros vagy páratlan volt-e. Tehát a kivonásokat egy ciklussal addig végezzük, amíg az eredmény nullánál nagyobb. Amint elértük a nullát vagy a –1-et, abbahagyjuk a kivonásokat. A ciklus kilépési feltételét (*Until* paraméter) *False*-ra

állítottuk. A kivonás eredményét szintén a Szam változóban tároljuk, így ugyan elveszíthetjük a korábbi értékeket, de azokra most nincs szükségünk.

A program harmadik részében azt vizsgáljuk, hogy a ciklusból kilépés után a Szam változó tartalma 0 vagy -1. Ha -1, akkor az eredeti szám páratlan volt, egyébként páros. Ezt kiíratjuk képernyőtörlés nélkül a 4. sorba. Az 5 mp-es várakozás után a program befejeződik.



A képernyőkép:



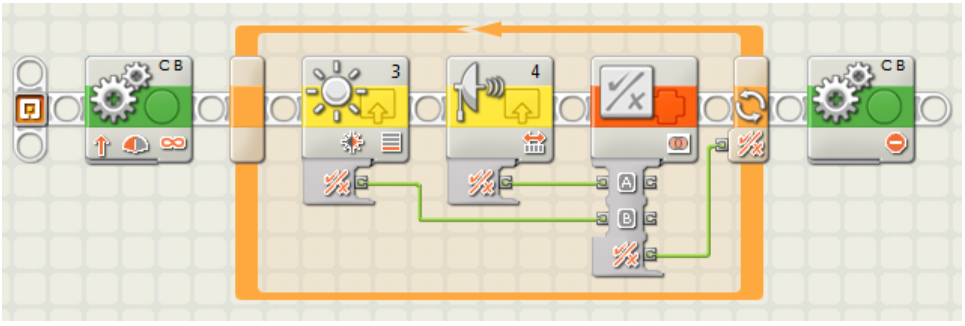
Érdemes észrevenni, hogy a ciklus végrehajtásainak száma, éppen a kettővel való osztás utáni hányados egész része, míg a ciklusból történő kilépés utáni Szam változó abszolút értéke a kettővel történő osztás utáni maradék komplementere (osztó – |Szam| = maradék). Természetesen ez az algoritmus nemcsak 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 kerül egy akadályhoz!

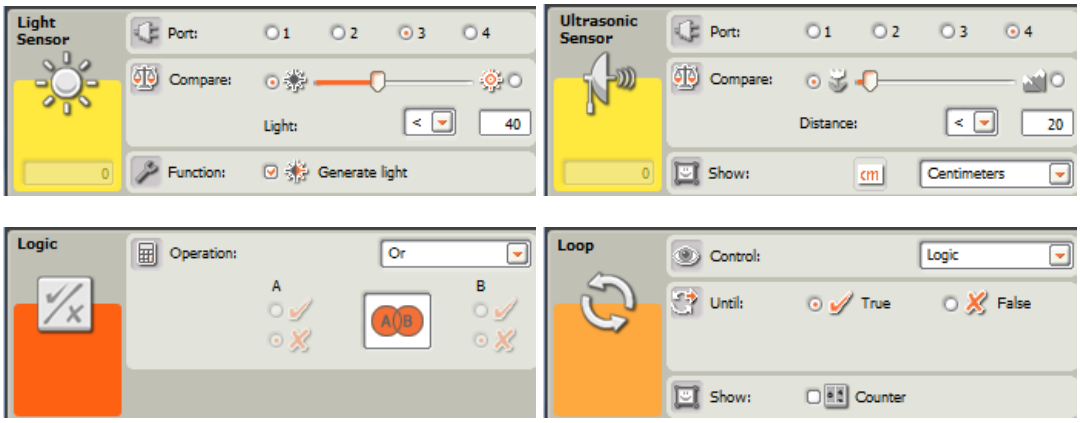
A robot mozgását egy összetett feltétel vezérli. Két szenzor (fény és ultrahang) értékét kell folyamatosan figyelni és akkor kell abbahagyni a mozgást, ha legalább az egyik esetén teljesül a példában megadott feltétel. A 7. fejezetben már utaltunk az ilyen összetett feltételekkel történő programvezérlésre. 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 ciklus és a benne szereplő modulok legfontosabb paramétereinek beállítását láthatjuk az alábbi ábrákon.



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.

9.4. 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, …)

10. TÖBBSZÁLÚ PROGRAMOK – TASZKOK

A hagyományos 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. Természetesen programjaink tartalmazhatnak pl. függvényhívásokat (lásd a saját blokkok létrehozását bemutató 14.5. fejezetet), de ezekben az esetekben is igaz, hogy a végrehajtott utasítások egy lineáris láncot alkotnak. 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 NXT-G programnyelv 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

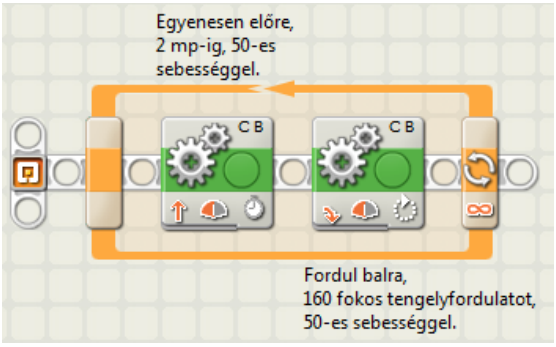
A programunkban tehát létrehozhatunk egymással látszólag párhuzamosan működő, futó taszkokat (szálakat), így egy időben tudjuk a motorokat vezérelni és a szenzorokkal mért adatokat kezelni.

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 nem történhet egyidőben.

Egy egyszerű példával szemléltetve a szituációt:

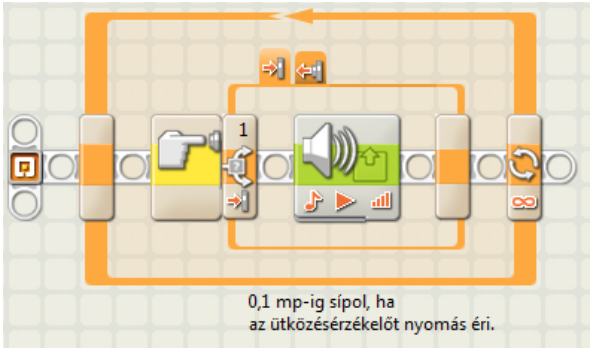
10/P1. Írjon programot, amelyet végrehajtva a robot egy sokszög alakú pályán halad folyamatosan! Abban az esetben, ha az ütközésérzékelőjét nyomás éri, adjon hangjelzést!

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, de ez most nem fontos.

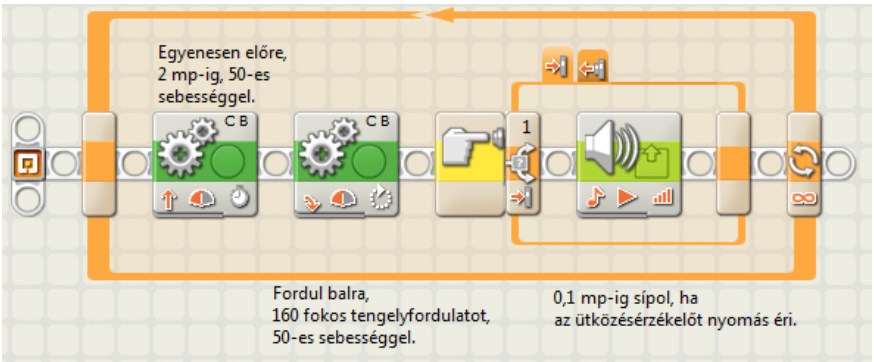


A két motor paraméterezésének fontos részleteit mutatja az alábbi ábra.

Az ütközésérzékelőre történő hangjelzés programozása szintén egyszerű feladat. Az elágazás alsó ága nem tartalmaz utasítást, ezért nem jelenítettük meg.



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.



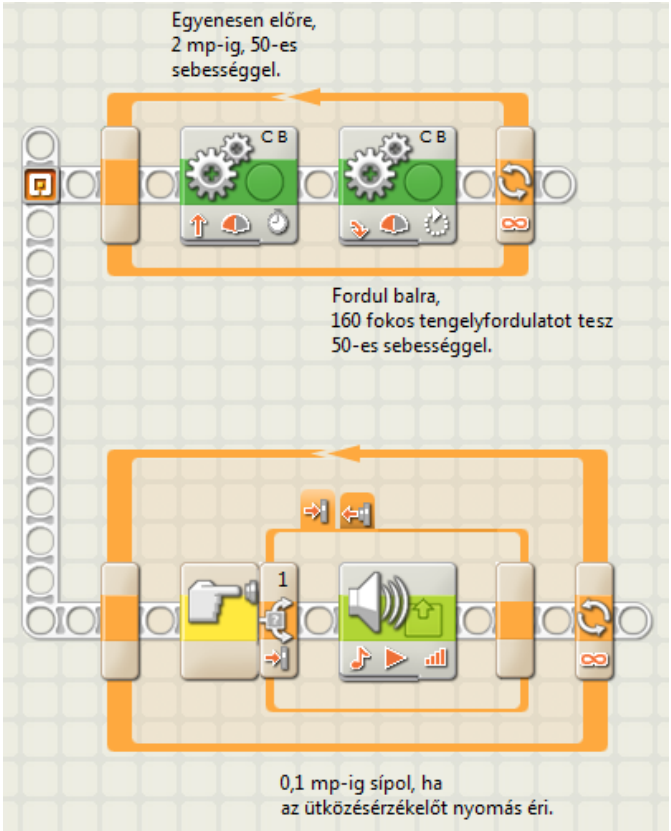
Érdemes kipróbálni mindhárom programot. A harmadik esetben (amikor a két előzőt egymás után helyeztük el egy ciklusban) nem működik helyesen a program. Hiába nyomjuk meg az ütközésérzékelőt, a hangjelzés elmarad, vagy csak rövid ideig hallható. A programot tesztelve észrevehető, hogy ha az ütközésérzékelőt 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, 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.

A megoldás az, hogy a törött vonal mentén történő mozgást végző részt és a szenzorfigyelést megvalósító részt folyamatosan egymással párhuzamosan kellene végrehajtani. Ehhez két, látszólag egymással párhuzamosan futó szálat használunk. Mindkét részprogramot elhelyezzük a szerkesztő területen. A motorok vezérlését szolgáló részt a szokásos helyre, míg az érzékelőt figyelő ciklust alá mozgatjuk. Az összekötést a SHIFT billentyű lenyomása mellett az egérrel lehet elvégezni. Az így összekötött két szálon lévő utasítások egymással látszólag párhuzamosan fognak végrehajtódni. 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. Ebből a technikából következően, ha programjaink sok szálat tartalmaznak, akkor a processzor leterhelése miatt lassabban fognak futni.

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

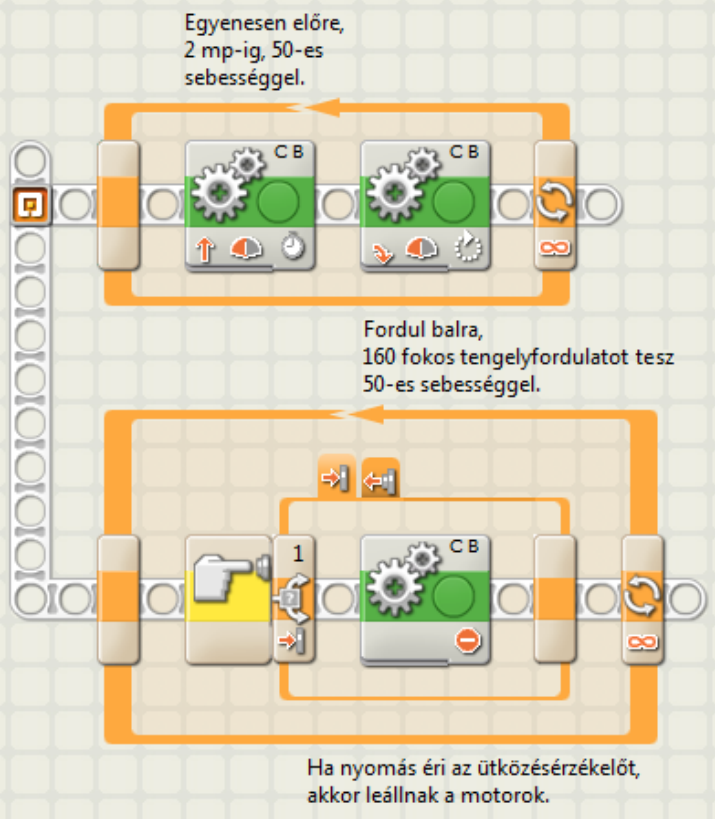
A program így már helyesen működik.



A többszálú programok használatánál problémát okozhat, ha egyszerre több szál is ugyanazt a motort vezérli. Ilyenkor esetleges, hogy melyik szál érvényesül éppen. Esetleg 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:



A program érdekes viselkedést mutat, ha elkezdjük tesztelni. Elindul a robot, majd ha az egyenesen haladó szakaszban nyomjuk meg az ütközésérzékelőt, akkor megáll. Ha letelt a 2 mp (az előre haladás ideje), akkor elkezd fordulni, majd mintha mi sem történt volna, folytatja haladását a sokszög alakú pályán. Az ok egyszerű, hiszen az első szál bekapcsolta a motorokat, amelyeknek 2 mp-ig kellett volna működni, tehát elkezdte mérni a 2 mp-et. Közben a második szál leállította a mozgást, de miután letelt a 2 mp, ismét az első szálon lévő modul kezdte irányítani a két motort. Ennek következtében fordult a robot, majd kezdte az első szál utasításait elölről.
Ha az ütközésérzékelőt akkor nyomjuk meg, amikor a fordulási szakaszban van a robot, akkor megáll és nem is indul el újra. Az oka ismét logikus, hiszen az első programszál fordulást vezérlő modulját nem idővel, hanem tengelyelfordulási szöggel vezéreljük, tehát elindul a fordulás, és elkezdi a robot mérni a tengelyelfordulási szöget. Mivel a második szálon lévő utasítással megállítottuk, így a beállított 160 fokot soha nem fogja elérni a tengelyek elfordulási szöge, tehát az első szálon nem is fog továbblépni, így mozdulatlan marad.

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.

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!

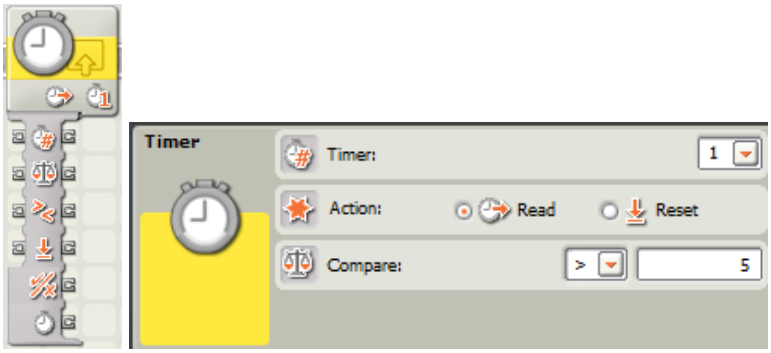
11. IDŐZÍTŐ, ELFORDULÁSMÉRŐ

11.1. Időzítők, stopper

A programírás során már többször volt szükség arra, hogy a robot bizonyos idő eltelte után hajtson végre valamilyen műveletet. Az idő mérése minden esetben csak úgy valósulhatott meg eddig, hogy amíg a beállított időtartam le nem telt, addig a program utasításainak végrehajtása állt. Szükségünk lehet olyan időmérő eszközre is, amely a háttérben képes működni, tehát amíg a robot a program utasításait végrehajtja, folyamatosan méri az időt, és a pillanatnyi állapota bármikor lekérdezhető.

A programozási környezetben rendelkezésünkre áll 3 db stopper a *Sensor* csoporton belül, amelyet a *Timer* ikon beillesztésével a program bármelyik helyén elindíthatunk, és onnantól kezdve milliszekundumban méri az időt. Ebből az időfolyamból bármikor vehetünk mintát (pl. egy változóban eltárolva), így lehetőségünk van a program egyes részeinek időzített elindítására.

Másik felhasználási lehetőség a „távolságmérés”. Abban az esetben, ha a robot állandó sebességgel halad, akkor a két pont megtétele között eltelt idő egyenesen arányos lesz a két pont távolságával. Így ezeket az időtartamokat mérve lehetőségünk lesz különböző pontok közötti távolságok „hosszának” összehasonlítására.



Összesen 3 stopper (*timer*) áll a rendelkezésünkre, amelyek közül a paraméterlista legelső legördülő eleménél választhatunk. Ez lehetővé teszi, hogy egyidőben három különböző időpontban elindított időfolyam pillanatnyi értékeit tudjuk lekérdezni. A stoppernek két állapota van (*Action*). A *Read* funkció választásakor az elindított stopper aktuális értékét olvashatjuk ki és tárolhatjuk a változóban, adhatjuk át paraméterként valamelyik modulnak, esetleg írathatjuk ki a képernyőre. Az értéket a legördülő paraméterlista órával jelölt csatlakozópontján lehet lekérdezni. A *Reset* funkció választásával lenullázhatjuk és újraindíthatjuk a stoppert. A *Compare* paraméter használata megegyezik a *Sensor* csoport ultrahangos távolságérzékelő moduljánál ismertetett funkcióval.

A stopper akkor indul, amikor a programszálon a végrehajtás a *Timer* modulra kerül és mindaddig folyamatosan méri az időt (milliszekundumonként eggyel nő az értéke), amíg a *Reset* funkcióval újra nem indítjuk, vagy véget nem ér a program.

A kreatív használatot példákon keresztül mutatjuk be.

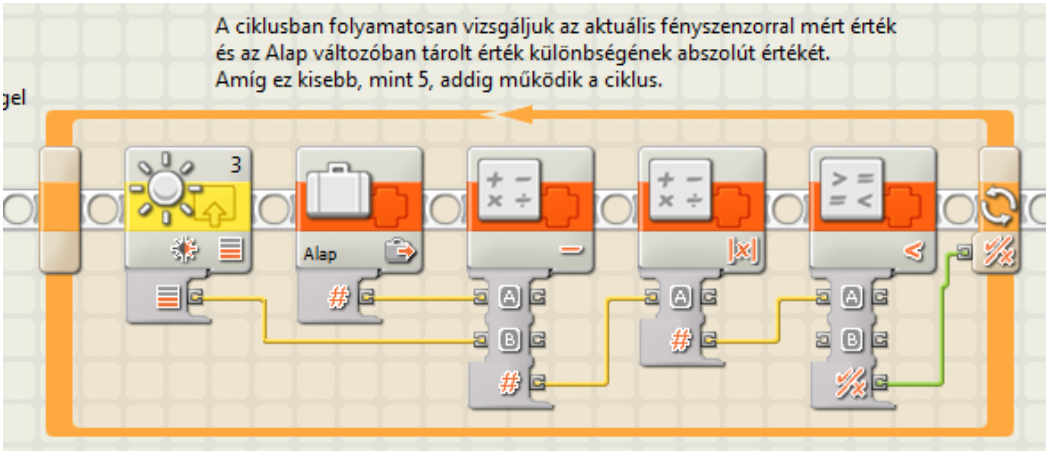
11/P1. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre az alapfelület színétől eltérő színű vonalig (legalább 5 egységgel eltér a szín), majd onnan visszatolat a kiindulási pontig!

A programírás során nem akarunk konstansokat használni, így kezdésként színmintát veszünk az alapfelületről. Ezt tároljuk az Alap nevű változóban. Ezután a robot addig megy előre, amíg a fényszenzora által mért érték abszolút értékben 5-nél nagyobb mértékben eltér az Alap változóban tárolttól, ekkor megáll. A mozgás megkezdésekor elindítunk egy stoppert, amelynek értékét a megálláskor kiolvassuk és eltároljuk az Ido nevű változóban. Ez az érték megadja, hogy mennyi ideig tartott, amíg a robot elérte a vonalat. Ezután az előre haladás sebességével megegyező sebességgel tolatást kezdünk, amelyet az Ido változóban tárolt milliszekundum időtartamig kell végezni.

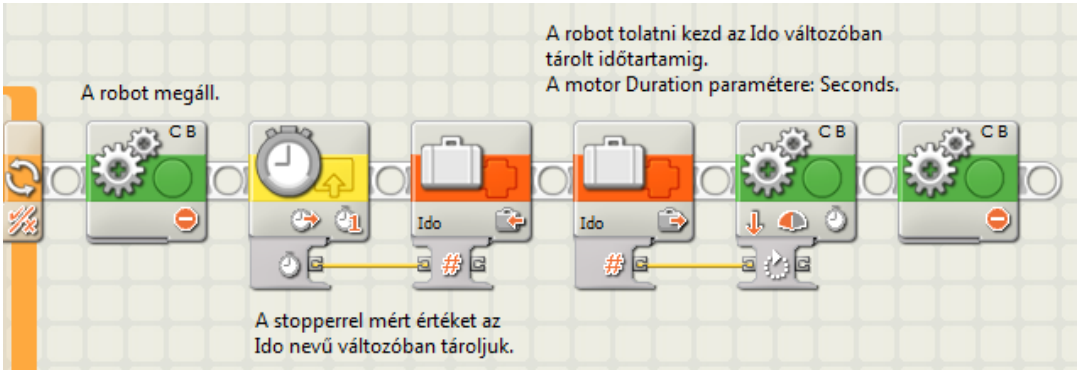
A programot három részre bontva mutatjuk be.

Az első programrészben a szín mintavétel, a stopper és a robot indítása történik. A robot 50-es sebességgel halad egyenletesen. Mivel nem tudjuk, mennyi ideig kell mozognia, ezért a *Duration* paramétere *Unlimited*.

A második programrészletben szereplő ciklus vizsgálja, hogy a robot elérte-e az eltérő színű vonalat. Ennek elvét a színfüggetlen útvonalkövetés példánál már magyaráztuk. A lényeg, hogy a fényszenzorral mért aktuális érték és a kezdeti érték (Alap nevű változó tartalma) különbségét vizsgáljuk. Mivel nem tudjuk, melyik érték a nagyobb, ezért a különbség akár negatív is lehet. Az előjelbeli eltérés kiküszöbölésére a különbség abszolút értékét használjuk. Amennyiben ez kisebb, mint 5, akkor a ciklus újra lefut. Ha a különbség abszolút értéke legalább 5, akkor kilépünk a ciklusból (a ciklus feltétel: *False*).



A harmadik programrészletnél történik a tolatás vezérlése. Először megáll a robot, majd a stopper által mutatott aktuális időt tároljuk az Ido nevű változóban. Mivel a stoppert a robot indulásakor indítottuk, ezért ez az idő a robot menetideje. Ugyanennyi ideig kell visszafelé tolatni, ügyelve, hogy a sebesség megegyezzen az előre haladás sebességével. Fontos, hogy a tolatást vezérlő *Move* modul *Duration* paraméterét *Seconds* értékre állítsuk. Az Ido változó kihagyható a programból, ekkor a stopper által mért idő értékét közvetlenül a tolatást vezérlő *Move* modulnak adjuk át.

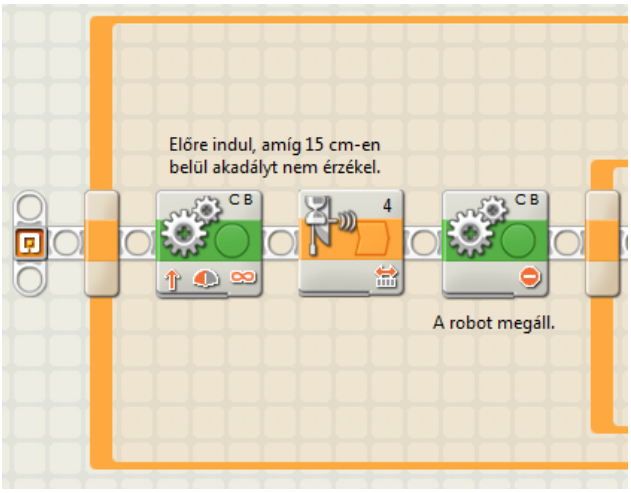


Az eltérő színű vonaltól tetszőleges távolságból indíthatjuk a robotot, az mindig visszatolat a kiindulóponthoz. Persze az eszköz pontosságának megfelelő mértékben.

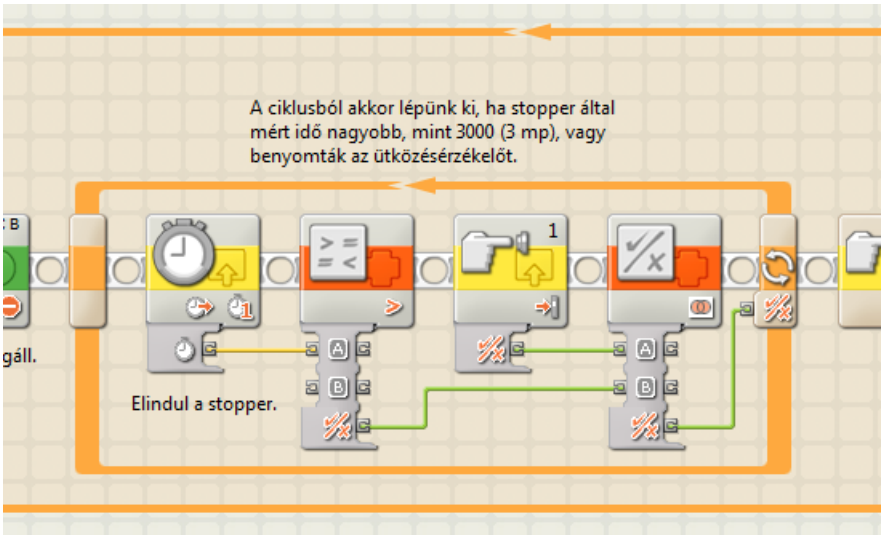
11/P2. Írjon programot, amelyet végrehajtva a robot egyenesen előre indul és egy akadálytól 15 cm-re megáll! Itt várakozik 3 mp-ig. Ha eközben nyomás éri az ütközésérzékelőjét, akkor 160 fokos tengelyfordulatot tesz, majd befejezi programját. Ha 3 mp-en belül nem éri nyomás az ütközésérzékelőjét, akkor 2 mp-ig tolat hátra, majd ismét elindul előre, és megáll az akadálytól 15 cm-re. Mindezt addig ismétli, amíg a 3 mp-es várakozás közben meg nem nyomják az ütközésérzékelőjét.

A program egyes elemei nem bonyolultak. Az egyetlen újdonság az összetett feltételrendszer, ami az akadálynál történő várakozáskor szerepel. A mozgást ciklusba helyezett utasítások vezérlik akadályig és vissza. Az akadálynál történő várakozáskor 3 mp-en keresztül figyelni kell az ütközésérzékelő állapotát. Ha nyomás éri eközben, fordulni kell és befejezni a programot, egyébként tolatni, majd újrakezdeni a ciklust.

A programot három részletben mutatjuk be. Az első programrészletben az akadályig történő mozgás lebonyolítását végző utasítások szerepelnek. A robot 50-es sebességgel elindul előre. Mivel nem tudjuk, hogy mennyit kell előre mennie, ezért a *Move* ikon *Duration* paramétere: *Unlimited*. A várakoztató modulon (*Wait*) akkor lép túl a végrehajtás, ha az ultrahang szenzorral mért távolság 15 cm-nél kisebb. Ekkor megáll a robot.

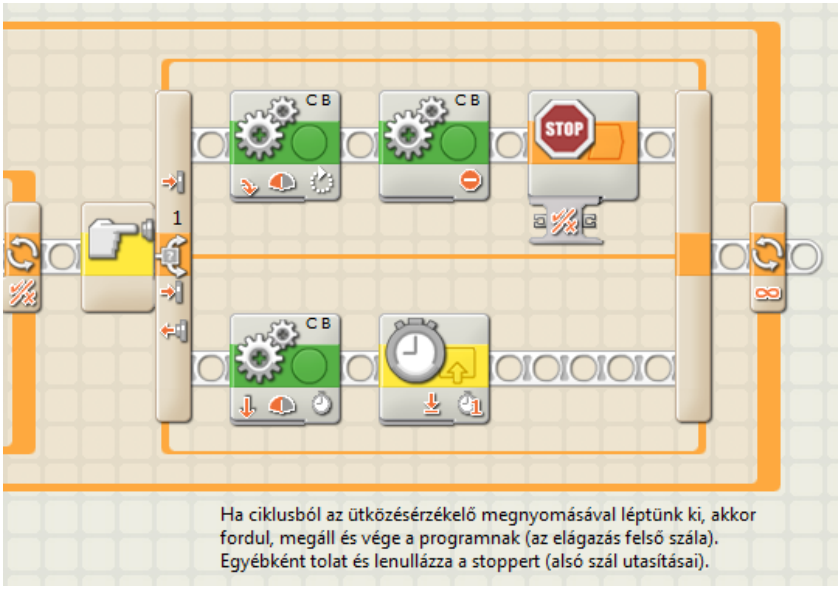


A második programrészlet végzi a 3 mp-es várakoztatást, miközben az ütközésérzékelőt figyeli. Ezt egy ciklussal oldottuk meg, aminek a kilépési feltétele összetett. Két feltétel összekapcsolásával hoztuk létre. Az elindított stopper méri az időt. A ciklusból akkor fogunk kilépni, ha a stopper által mért idő nagyobb lesz, mint 3000 milliszekundum (3 másodperc), vagy megnyomták az ütközésérzékelőt. A két feltételt a logikai „vagy” modul köti össze. A ciklus kilépési feltételét *True*-ra állítottuk. Ez azt jelenti, hogy akkor lépünk ki, ha a feltétel igaz. A vaggyal összekötött két feltétel alapján mindez akkor következik be, ha az összetett feltétel valamelyik tagja igaz lesz. Tehát vagy letelt a 3000 ms, vagy megnyomták az ütközésérzékelőt (természetesen egyszerre is bekövetkezhet a dolog, de ennek kicsi az esélye).



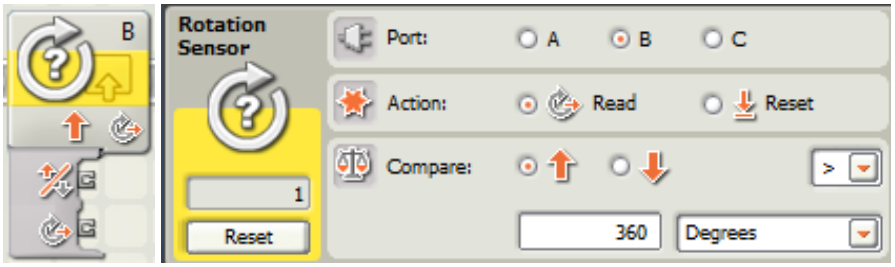
A harmadik programrészlet utasításai vezérlik a ciklusból történő kilépés utáni robottevékenységet, a kétféle kilépési lehetőségnek megfelelően. Megvizsgáljuk, hogy az ütközésérzékelő megnyomásával történt-e a kilépés. Mivel a program elég gyorsan fut, így gyakorlatilag arra nincs esély, hogy a ciklusból a 3 mp letelte miatt léptünk ki, és a következő ezredmásodpercben nyomták meg az ütközésérzékelőt, így az itteni feltétel kiértékelése úgy viselkedik, mintha az ütközésérzékelő miatt léptünk volna ki. Ez szemmel egyébként sem lenne látható. A feltétel kiértékelése tehát az ütközésérzékelő megnyomását vizsgálja. Ha emiatt történt a ciklusból kilépés, akkor a felső szál

utasításai kerülnek végrehajtásra, vagyis a robot fordul, megáll és befejeződik a program. Ha nem az ütközésérzékelő megnyomásával léptünk ki a ciklusból (hanem letelt a 3 mp), akkor az alsó szál utasításai az érvényesek. Ekkor tolat a robot, majd kezdi elölről a külső ciklust (elindul előre akadályig, stb.). Mielőtt a robot a külső ciklus utasításainak végrehajtását újra kezdené a stoppert fontos lenullázni (*Reset*), mert magától nem áll meg, és az újbóli futás során a mért idő már jóval 3000 milliszekundum fölött lenne.



11.2. Elfordulásmérő

A *Sensor* csoportban található *Rotation Sensor* modul alkalmas arra, hogy a kiválasztott motor tengelyének elfordulási szögét visszaadja. A motor bekapcsolásakor 0-ról indul a számláló, és az elfordulás irányának megfelelően pozitív vagy negatív értéket ad vissza. Beállítható, hogy az elfordulás szögét fokban vagy a teljes körülfordulások számával mérje-e. Ez utóbbi esetben két tizedesjegy pontossággal adja vissza az értéket. A motor elindulását követően folyamatosan méri a kiinduló helyzethez viszonyított elfordulást a program leállításáig vagy a lenullázásig (*Reset*). Az elfordulási szög értékét összegezve adja vissza. Ez azt jelenti, hogy ha a motor elindítását követően előre fordult a tengelye 180 fokot, majd hátra 50 fokot, akkor a visszaadott érték 130 fok lesz. A modul a stopperhez hasonlóan alkalmas arra, hogy a két pont közötti „távolságot” megmérjük. Ebben az esetben a mértékegység a szögelfordulás lesz fokban vagy a körülfordulások számában megadva. Míg a stopperrel történő mérés nem függ a tényleges terepviszonyokon történő mozgástól, addig az elfordulásmérő a kerék tapadását, csúszását nem hagyja figyelmen kívül. Alkalmas lehet több együttesen működő motor szinkronizálására. Ha azt szeretnénk, hogy minden motor által hajtott kerék azonos utat tegyen meg, figyelmen kívül hagyva esetleg a kerék csúszásából adódó eltéréseket, akkor a motorok elfordulási szögeit külön-külön figyelve a különbségek alapján korrigálható a forgás és ezen keresztül a robot mozgása.



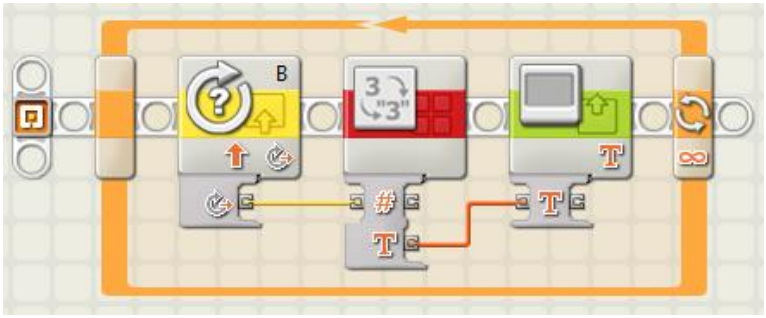
Az egyes paraméterek jelentése:

A paraméter neve	A paraméter jelentése
Port	Annak a portnak a betűjele, amelyre a mérni kívánt motor csatlakozik.
Action	*Read* paraméter esetén a modul visszaadja az összegzett elfordulás mértékét, amely tárolható változóban, vagy felhasználható más modulok bemenő paramétereként. *Reset* paraméter esetén lenullázza az elfordulás mérő számlálóját.
Compare	Beállíthatunk egy olyan feltételt, amelynek teljesülése esetén a modul legördülő paraméterlistájának megfelelő csatlakozási pontján igaz érték jelenik meg. Ciklusok vagy elágazások esetén például feltételvezérlésre használható. A legördülő lista két értéket tartalmaz: *Degrees* vagy *Rotation*. Az itt beállított érték fogja meghatározni, hogy milyen mértékegységben mér a szenzor.

Példaként egy egyszerű programot írunk, amely a visszaadott érték nagyságát szemlélteti.

11/P3. Írjon programot, amelyet végrehajtva a robot a képernyőre írja a B portra kötött motorjának elfordulási szögét fok mértékegységben!

A program nem tartalmaz a robot mozgását eredményező utasítást. Legegyszerűbben tesztelni úgy lehet, ha kézzel forgatjuk a B portra kötött motor tengelyét (pl. a rászerelt kerék segítségével). Mindkét irányú forgatást próbáljuk ki!



11.3. Gyakorló feladatok

11/F1. Írjon programot, amelyet végrehajtva a robot az alap színétől jól megkülönböztethető és különböző szélességű csíksoron egyenletes sebességgel halad keresztül. Minden csík után írja képernyőre a csík szélességét milliszekundumban (a csík fölötti áthaladás időtartama).

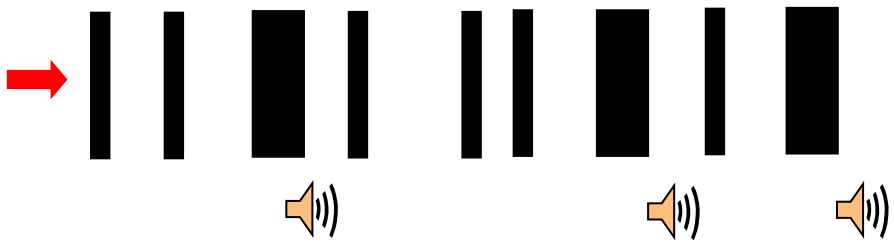
11/F2. Írjon programot, amelyet végrehajtva a robot addig forog, amíg ultrahang szenzorával meg nem lát 20 cm-es távolságon belül valamit, vagy 3 másodpercig. Ezután álljon meg!

11/F3. Írjon programot, amelyet végrehajtva a robot addig forog, amíg ultrahang szenzorával meg nem lát 20 cm-es távolságon belül valamit, vagy 3 másodpercig. Ha meglátott egy akadályt akkor induljon el felé és tolja 3 mp-ig, ha nem látott meg akadályt, de letelt a 3 mp, akkor menjen előre 2 mp-ig, majd kezdje elölről a forgást! Mindezt egy akadály észleléséig folytassa!

11/F4. Írjon programot, amelyet végrehajtva a robot egy adott kereten belül (az alap színétől eltérő színű határvonalon belül) megkeres különböző tárgyakat (gömb vagy téglatest), és azokat a határvonalon kívülre tolja!

11/F5. Írjon programot, amelyet végrehajtva a robot kétféle szélességű csíkokból álló csíksor fölött halad, majd a szélesebb csík fölött áthaladva hangjelzést ad! Az első csík nem lehet széles.

Pl.:

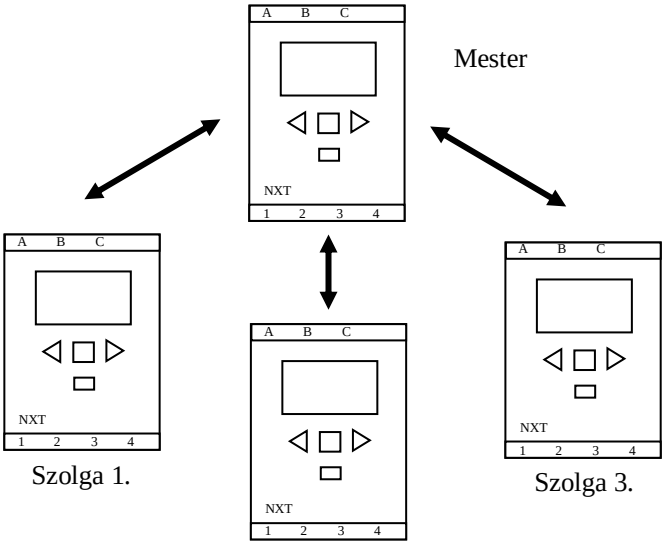


11/F6. Írjon programot, amelyet végrehajtva a robot egyenesen előre indul mindaddig, amíg ultrahang szenzora 15 cm-en belül akadályt nem érzékel! Ekkor visszatolat kb. startpozícióig. Várakozik 5 mp-et, majd újra elindul előre, az ultrahangszenzora által jelzett akadályig (15 cm-en belül). Az akadály a második esetben az előzőtől eltérő távolságra van (vagy közelebb, vagy távolabb). Ismét tolat kb. a startpozícióig, majd egyet vagy kettőt sípol (hang: 440 Hz – zenei A, 200 ms időtartamig, utána 200 ms szünet) aszerint, hogy melyik esetben volt az akadály messzebb a startpozíciótól.

12. 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 NXT bluetooth protokollját úgy készítették el, hogy egy mester robot, és három szolga kapcsolódhat össze. Alapesetben tehát összesen négy robotot kapcsolhatunk össze bluetooth alapú hálózattá. Haladó esetben ez a korlát bővíthető, például két tégla I2C porton történő kábeles összekötésével, és a kábelen történő kommunikációval (a robot 4-es bemeneti portja). Így mindkét kábellel összekötött tégla további három-három szolgával kommunikálhat bluetoothon keresztül. Más technikákkal bonyolultabb hálózati topológiák is kialakíthatók.

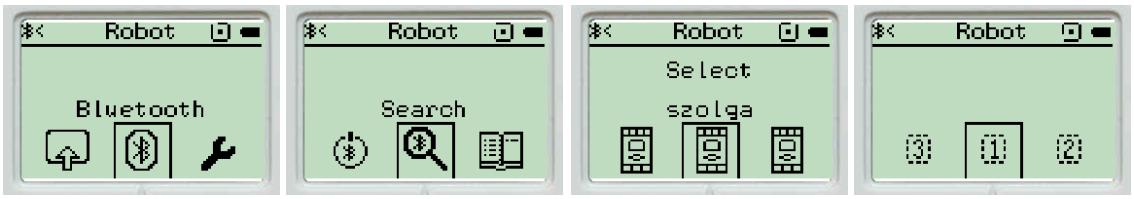


A szolga robotok csak a mesterrel kommunikálhatnak, így egymásnak szánt üzeneteiket is csak a mesteren keresztül küldhetik. Egy robot egyszerre vagy mester vagy szolga lehet.

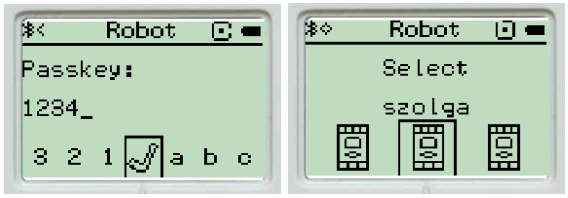
A kommunikáció megkezdése előtt fel kell építeni a bluetooth kapcsolatot. A kapcsolat felépítését mindig a mesternek kijelölt roboton kell kezdeményezni, és általában az NXT tégla képernyőmenüjének megfelelő funkciójával, de programból is elvégezhető. A kapcsolat mindaddig megmarad, amíg le nem bontjuk, 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

Az első kapcsolat felépítésekor a mester robot képernyőmenüjének *Bluetooth/Search* menüpontjával érdemes megkerestetni a hatókörön belüli kommunikációra képes eszközöket (ez eltarthat néhány percig). Ezután a listából kiválasztható az eszköz (pl. egy másik robot), amellyel a kommunikációt szeretnénk felépíteni. A megjelenő lehetőségek közül a csatlakozási portot kell kiválasztani, és a kapcsolat kiépül. Csatlakozási portként 1-3 lehetőségek közül választhatunk. A kiválasztott porttal tudjuk a programon belül később a szolga robotot azonosítani.



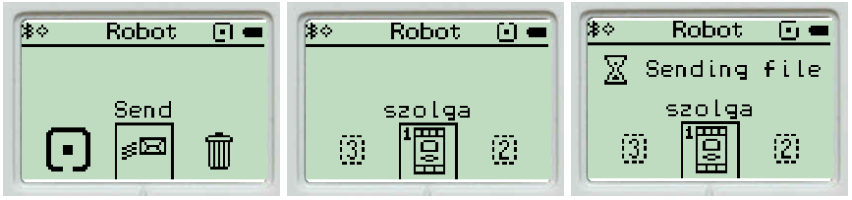
Az első kapcsolatépítés során egy kódcsere is megtörténik, amely mindkét kapcsolatban részt vevő eszköz képernyőjén megjelenő kód beállítását és elfogadását jelenti. A csatlakozás állapotát a bal felső sarokban lévő ikon megváltozása is jelzi.



Ha a kapcsolat felépült, akkor a rendszer megjegyzi a kódokat, és a továbbiakban a keresés helyett a *My contacts* menüpontban is megjelenik az eszköz, amelyet kiválasztva csatlakozást lehet kezdeményezni.

Ha a kapcsolat kiépült, akkor a programok már feltölthetők a robotokra. Ezt elvégezhetjük külön-külön a számítógéphez csatlakoztatva őket, vagy lehetőségünk van a mester roboton keresztüli programfeltöltésre. Ekkor a mester robotra töltünk fel minden programot, majd azt, amelyiket a szolga robotra szeretnénk áttölteni kiválasztjuk a képernyőmenüben, majd a *Send* almenüpontot választva a megadott bluetooth portra csatlakoztatott robotnak át tudjuk küldeni.

Az áttöltés csak akkor sikeres, ha a szolga roboton nincs ugyanazon a néven program, ha ilyen van, akkor azt először törölni kell.



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 moduljainak kiválasztásával és programba illesztésével.

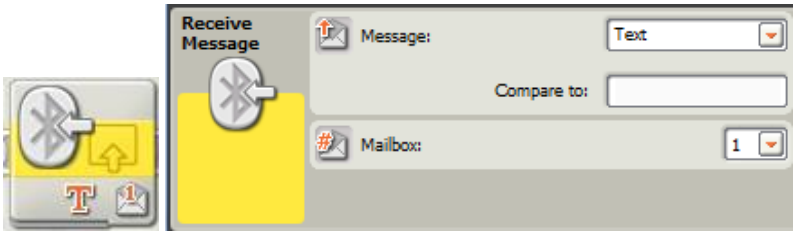
A mesterről történő adatküldéshez minden esetben meg kell adni annak a szolgának a kommunikációs portszámát (1-3), amelynek az üzenet szól (a csatlakozáskor állítottuk be). A küldés után az adat egy pufferbe (*mailbox*) kerül a fogadó roboton, ahonnan kiolvashatjuk és pl. eltárolhatjuk egy változóban. Az üzenet pufferből történő kiolvasása után törlődik onnan, tehát újbóli kiolvasása nem lehetséges. Összesen 10 ilyen puffer áll a rendelkezésre, tehát 10 db adatot tud fogadni a robot adatvesztés és kiolvasás nélkül.

Az üzenetküldés és üzenetfogadás utasításait egy-egy modullal tudjuk megvalósítani. Az üzenetküldés az *Action* csoport *Send Message* blokkjával lehetséges.

A paraméterek jelentése:

A paraméter neve	A paraméter jelentése
Connection	A kommunikációs port száma. A kapcsolat felépítésénél kell megadni. A 0 érték a mestert, míg az 1-3 érték a szolgákat azonosítja.
Message	Az elküldendő üzenet típusa, amely lehet szöveg (*Text*), szám (*Number*), vagy logikai érték (*Logic*). A típusbeállítás után adhatjuk meg az üzenet értékét a szövegdobozban.
Mailbox	Annak a puffernek a száma, amelyikbe az üzenet kerül. Az értéke 1-10 lehet.

Az üzenetek fogadása a *Sensor* csoport *Receive Message* blokkjával lehetséges.



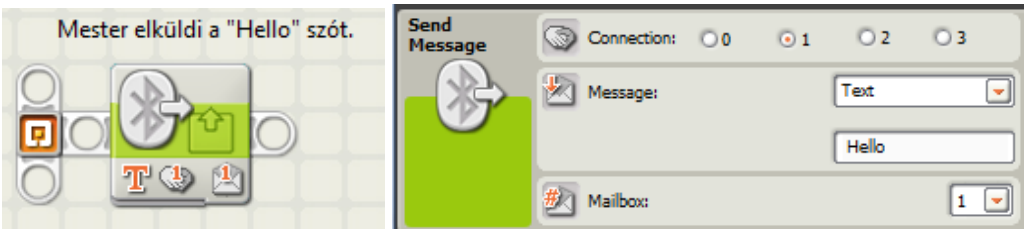
A paraméterek jelentése:

A paraméter neve	A paraméter jelentése
Message	A fogadott üzenet típusa, amely lehet szöveg (*Text*), szám (*Number*) vagy logikai érték (*Logic*). A típus megadása utáni *Compare to* szövegdobozban a beállított típusnak megfelelő értéket adhatunk meg. Ha a megkapott üzenet megegyezik az itt megadottal, akkor a modul legördíthető paraméterlistájának *Logic out* kimenetén megjelenik egy igaz (*True*) érték, egyébként hamis (*False*). Mindez alkalmas ciklusok, elágazások feltételeinek vezérlésére, ha nincs szükségünk a konkrét üzenet tartalmára, csak arra, hogy megegyezik-e egy bizonyos értékkel.
Mailbox	Annak a puffernek a száma, amelyikbe az üzenet került. Az értéke 1-10 lehet.

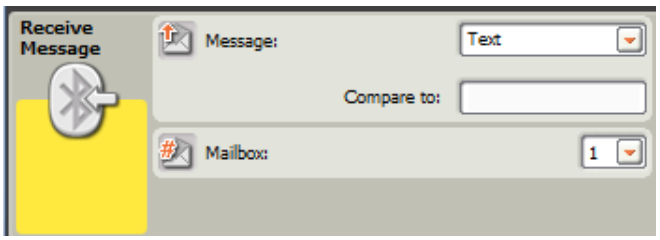
Az üzenetek küldésénél és fogadásánál a kommunikációs csatorna és a *mailbox* számát ugyanúgy kell megadni a két roboton, ellenkező esetben az üzeneteinket nem tudjuk olvasni.
A kommunikációs programoknál mindig figyelni kell arra, hogy az üzenetek küldése és fogadása szinkronban legyen. Ez azt jelenti, hogy az üzenet olvasását végző modul egyszer nézi meg a *mailbox* tartalmát 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. A másik, gyakrabban használt megoldás, ha egy ciklusba tesszük be az üzenetolvasási utasítást, így folyamatosan tudjuk figyelni a *mailbox*ot.
Nézzünk egy-egy példát mindkét esetre.

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, egyetlen utasítást tartalmaz, az üzenetküldést, amely az 1-es kommunikációs csatornára kapcsolódó szolgának szól, és az üzenet az 1-es *mailbox*ba kerül. Az üzenet egyszeri elküldése után a program le is áll. 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 az 1-es *mailbox*át, és képernyőre írja a benne megjelenő szöveges üzeneteket. Az üzenet kiolvasása után a *mailbox* kiürül, de az üres puffer tartalma nem szöveg, ezért az üzenet a képernyőn marad.

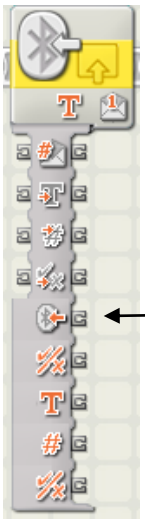


A szolga robot esetében úgy oldottuk meg a szinkronizálási problémát (tehát azt, hogy csak az üzenet megérkezése után van értelme azt a pufferből olvasni), hogy egy végtelen ciklussal folyamatosan olvassuk a puffer tartalmát.

Egy másik lehetséges megoldás a szolga robot programjára:

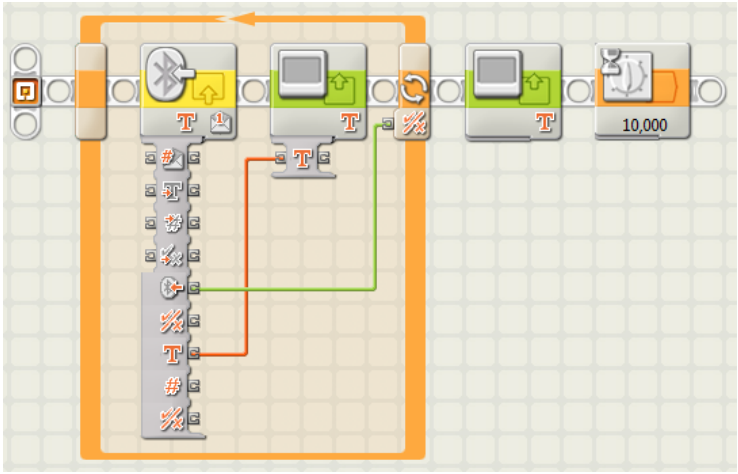
Ennél a programnál 2 mp-et vár a *mailbox* olvasásával a szolga robot, és utána csak egyszer olvassa annak tartalmát. A mesternek tehát 2 mp áll a rendelkezésére, hogy elküldje az üzenetet. Ha ez nem történik meg ennyi idő alatt, akkor a kommunikáció eredménytelen.

Az első megoldás tűnik biztosabbnak. A ciklussal figyelés esetén azonban jó lenne valamilyen feltétellel vezérelnünk a futását, hogy ne kellejen végtelen ciklust használni. Tehát meg kell adnunk, hogy meddig figyelje a *mailbox* tartalmát. Az időtartammal történő szabályozás nem tűnik célravezetőnek, hiszen ugyanúgy lekéshetjük az üzenetet. Egy logikai feltételt kell megadnunk arra vonatkozóan, hogy mikor lépjünk ki a ciklusból. A kérdésre kézenfekvőnek tűnik a válasz: ha megkaptuk az üzenetet. Erre a célra az üzenetfogadó modul legördülő paraméterlistáján szerepel egy csatlakozási pont, ami logikai típusú, és akkor ad igaz értéket, ha a *mailbox*ba üzenet érkezett.



Ha a beállított *mailbox*ba üzenet érkezett, akkor igaz az értéke, egyébként hamis.

Paraméterátadással a ciklus kilépési feltételeként használjuk a csatlakozási ponton megjelenő logikai értéket. A ciklusból kilépve a képernyőre íratjuk a „*Vege a ciklusnak.*” szöveget, hogy a program futása során is lássuk a ciklusból kilépés megtörténtét. A megkapott üzenet és a fenti szöveg gyakorlatilag egy időben jelenik meg a képernyőn.

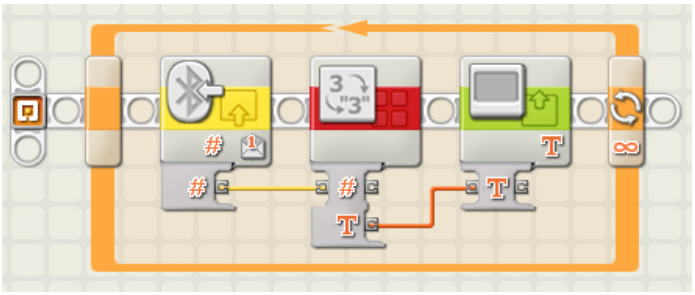


Azt láttuk, hogy a szöveges típusú adatküldés esetén a puffer kiürülése nem zavarta a megjelenítendő információt. Szám típusú adatok küldése esetén azonban ez kicsit bonyolultabb. Az üres puffer tartalmát ugyanis a rendszer 0-ként érzékeli, így a nulla küldött adat, vagy az üres puffer összetéveszthető, ha nem figyelünk erre a programírás során. A következő példával szemléltetjük ezt.

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 szolgának az 1-es kommunikációs csatornán, az 1-es *mailbox*ba. Mindezt folyamatosan, tehát végtelen ciklusba illesztve. Az adatküldés modul *Message* paraméterénél *Number* típust kell beállítatni.

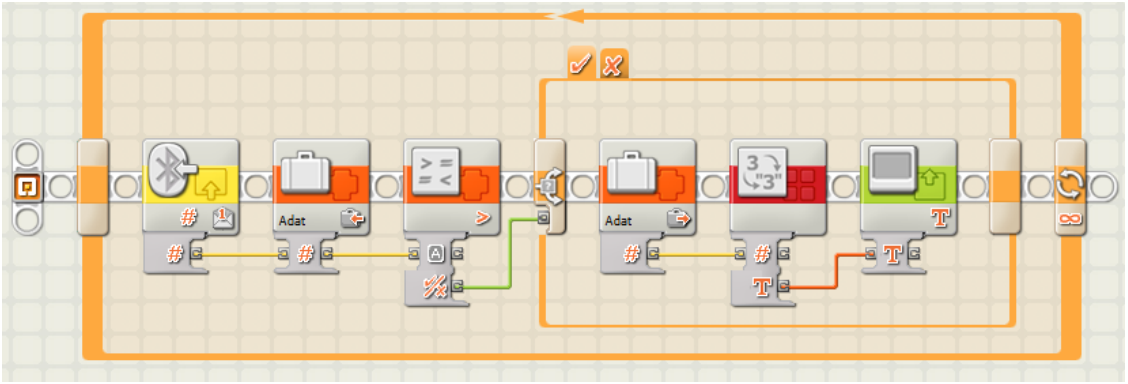
A szolga robot esetén a megkapott szám típusú adatot először szöveggé kell alakítani, majd megjeleníteni a képernyőn. A képernyőtörlés (*Clear*) be van kapcsolva.



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

A programokat tesztelve azt látjuk, hogy a szolga képernyőjén megjelennek az adatok, de váltakozva hol egy nagyobb szám, hol pedig a nulla. Ennek az oka, hogy az adatolvasás és megjelenítés gyorsabban hajtódik végre, mint a bluetoothon keresztüli adatátvitel. Így a megérkezett adat kiolvasása után a puffer még az új adat megérkezéséig üres, és ezt jeleníti meg a rendszer nullaként a képernyőn. Mivel ez zavaró, ezért jó lenne korrigálni.

Egy lehetséges megoldás, ha a puffer tartalmát nem közvetlenül a képernyőre íratjuk ki, hanem először eltároljuk egy változóban (Adat). Ebből a változóból csak akkor írjuk a képernyőre az értéket, ha az nagyobb, mint nulla. Minden új kiíratás előtt töröljük a képernyőt. Ezzel az algoritmussal megoldottuk, hogy a nullák ne jelenjenek meg a képernyőn, viszont így a nullát mint adatot elveszítettük kommunikációs szempontból. Mivel az elágazás alsó szála nem tartalmaz utasítást, ezért azt nem jelenítettük meg.

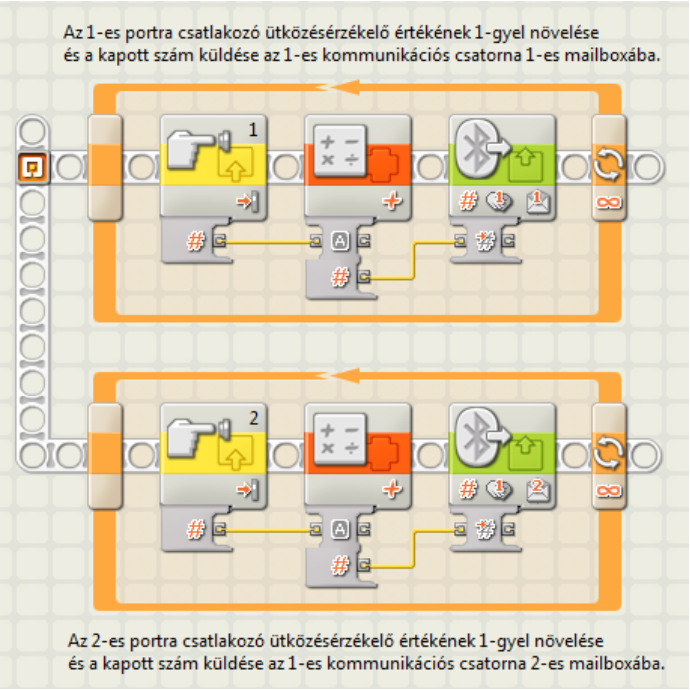


Az eddig bemutatott példák két robot közötti egyirányú kommunikációt valósítottak meg. A mester robot küldött üzeneteket, amelyeket a szolga fogadott és reagált rájuk. Ha a feladat során kétirányú kommunikációt szeretnénk megvalósítani, akkor a szolga is küldhet válaszként üzeneteket a mesternek. A szolgák a mestert a nullás kommunikációs csatornán keresztül érhetik el teljesen analóg módon, mint az

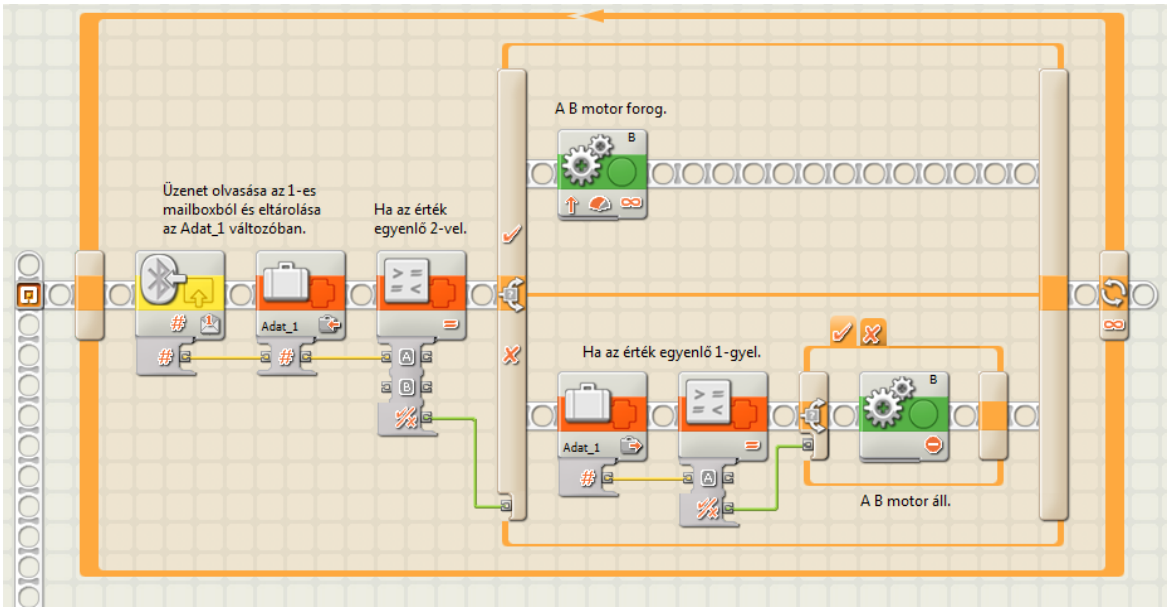
eddig bemutatott példákban. Ha egy mesterre két szolga kapcsolódik, és a két szolga egymásnak szeretne üzenetet küldeni, akkor azt a mesteren keresztül tehetik meg. Az egyik szolga elküldi a nullás csatornán az üzenetet a mesternek, majd az a másik szolga kommunikációs csatornáján küldi tovább.
A következő program az eddig bemutatott ötletek együttes használatát szemlélteti egy összetettebb példán, de továbbra is egyirányú kommunikációt megvalósítva.

12/P3. Írjon programot, amely két robot közötti kommunikációt valósít meg! A mester robotra két ütközésérzékelő csatlakozik az 1-es és 2-es porton keresztül. A mester az ütközésérzékelők állapotáról küld át információt a szolgának (be van nyomva/nincs benyomva). A szolga robotra szerelt két motor aszerint indul el, vagy áll meg, hogy a megkapott adat milyen. Ha a mester robot 1-es ütközésérzékelője be van nyomva, akkor elindul a szolga robot B motorja, egyébként áll. Ha a mester 2-es ütközésérzékelője van benyomva, akkor elindul a szolga robot C motorja, egyébként áll.

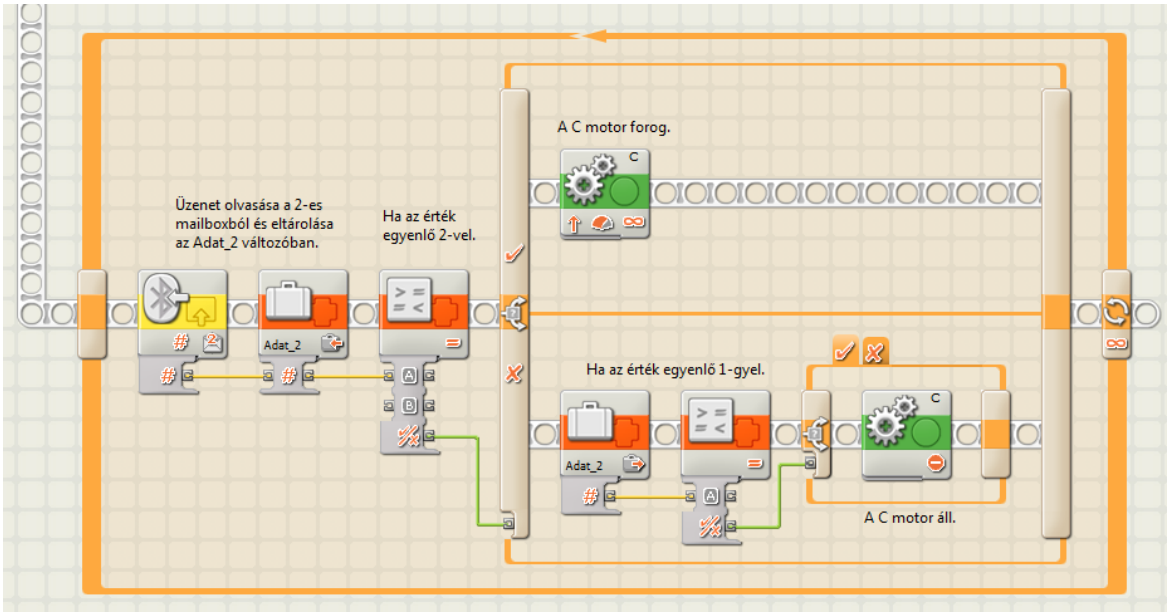
A program egy kezdetleges bluetoothos távirányító. A szolga robot mozog, a mesteren keresztül lehet irányítani. A robot nem tud tolatni, ahhoz, hogy az ellenkező irányba haladjon, meg kell fordulni vele (az egyik kereke forog, a másik nem).
A megoldásnál a mester robot 1-es értéket küld át a szolgának, ha nincs benyomva az ütközésérzékelő és 2-est, ha igen. A nullát elkerüljük, mert az üres puffer esetén is 0 a tartalma. Az 1-es portra csatlakoztatott ütközésérzékelő állapotát jelző 1-est vagy 2-est az 1-es *mailbox*ba, míg a 2-es portra csatlakoztatott szenzor állapotát jelző hasonló értékeket a 2-es *mailbox*ba küldjük. A két adatküldést folyamatosan végezzük (végtelen ciklusban) és párhuzamosan futó szálakon (taszk).
Az egyes és kettes értéket úgy állítjuk elő, hogy az ütközésérzékelő legördíthető paraméterlistájának *Logical Number* csatlakozási pontján megjelenő értékhez 1-et hozzáadunk. Mivel az érték nulla, ha érzékelő nincs benyomva, és egy, ha igen, az eggyel növelés éppen a megfelelő számokat állítja elő. Az így kapott számokat küldjük a szolgának.



A szolga robot a megkapott értéket kiolvassa a *mailbox*okból és eltárolja őket az Adat_1 vagy Adat_2 változóban aszerint, hogy melyik *mailbox*ból származnak. A korábban látott ötlet alapján a program megvizsgálja a változó tartalmát, hogy egyenlő-e kettővel, ekkor kell a motort bekapcsolni. Ha a változó tartalma nem kettő, akkor lehet még 1, ebben az esetben meg kell állítani a motort, illetve lehet 0, ekkor üres a puffer, így semmit sem kell csinálni. Tehát még egy elágazás szükséges, az első elágazás alsó (hamis) szálára. A második elágazásnak a hamis (alsó) szála nem tartalmaz utasítást, így azt nem jelenítettük meg. Az 1-es *mailbox*ba kapott érték a B, míg a 2-es *mailbox* értéke a C motort vezérli. A két programszerkezet teljesen analóg egymással, és futhat párhuzamos programszálon (taszk). Az 1-es *mailbox* B motort vezérlő programszála:

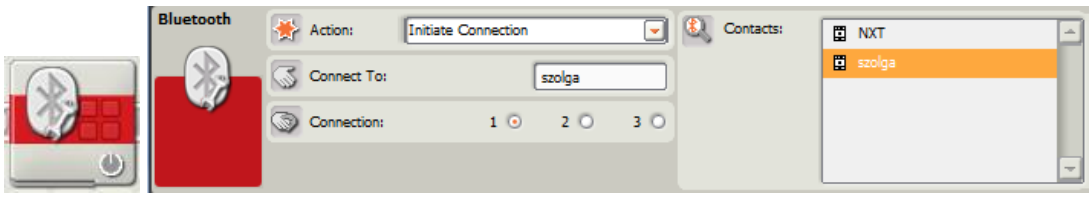


A 2-es *mailbox*, C motort vezérlő programszála:



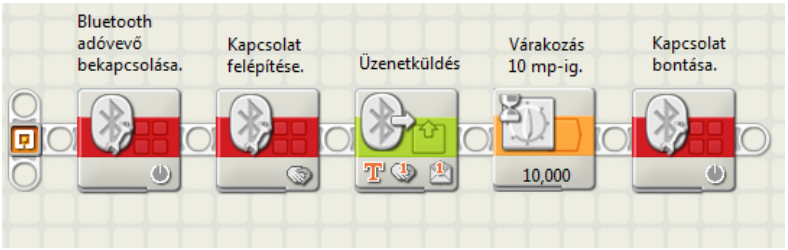
Az itt bemutatott távirányítónál komolyabb eszközök is építhetők. Például az interneten több ötlet és lehetőség is szerepel a konstrukciókról.

A kommunikációs kapcsolatot programból is fel lehet építeni a manuális felépítésnél szükséges információk megadásával. Az utasítást reprezentáló modul az *Advanced* csoport *Bloetooth Connection* ikonja.



A paraméter neve	A paraméter jelentése
Action	A kiválasztott tevékenységet lehet megadni. A lehetőségek: *Turn On* – a bluetooth adóvevő bekapcsolása *Turn Off* – a bluetooth adóvevő kikapcsolása *Initiate Connection* – kapcsolat felépítése a kiválasztott eszközzel. *Close Connection* – a kiépített kapcsolat bontása.
Connect To	Csak az *Initiate Connection* (kapcsolatépítés) esetén jelenik meg. Itt adhatjuk meg annak az eszköznek a nevét, amellyel kapcsolatot szeretnénk felépíteni. Segítségül a *Contacts* ablakban megjelenik azoknak az eszközöknek a neve, amelyek a tégla *My Connections* képernyőmenüjében is szerepelnek.
Connection	A kommunikációs csatorna száma. Mivel a kapcsolatépítést mindig a mester kezdeményezi, ezért itt a szolgák azonosítására fenntartott számok szerepelnek 1-3 (a 0 nem).

A következő program egy egyszerű kapcsolatépítést mutat be a mester programján keresztül.
Első lépésben megtörténik az adóvevő bekapcsolása, majd a kommunikációs csatorna kiépítése a „szolga” nevű robottal az 1-es csatornán. Ezután a mester elküld egy szöveges üzenetet és 10 mp várakozás után lebontja a kapcsolatot.



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 kommunikációját valósítja meg! A mester robot mint távirányító működik, és a szolga robot mozgását vezérli. A mozgás irányításához a mesterre szerelt két motor tengelyelfordulási szögének előjeles értékét küldi át két különböző *mailbox*ba, a két motornak megfelelően. A szolga robot a megkapott értékeket a két motorjának a sebesség paramétereként használja fel. (Ha valamelyik motor esetén negatív értéket adunk meg sebességként, akkor az ellentétes irányú és a kapott szám abszolút értékével megegyező nagyságú forgást eredményez. A motorokat nem lehet túlhajtani, tehát ha egy motor nagyobb értéket kap, mint 100, akkor is 100-as sebességgel forog tovább. A konstrukciót mégis érdemes úgy megépíteni, hogy elfordulási szögként ne lehessen 100 foknál nagyobb szöggel egyik irányba sem elfordítani a mester motorjának tengelyeit.)

12/F4. Írjon programot, amely két robot kommunikációját valósítja meg! Mindkét robot véletlenszerűen sorsol egy-egy 1-10 közötti számot. Információt cserélnek egymással a kisorsolt számokról. A melyik robot kisebb számot sorsolt, az veszített és tolat egy kicsit. Azonos kisorsolt számok esetén egyikük sem mozdul. Mindezt 10 másodpercen keresztül ismétlik.

12/F5. Írjon programot, amely három robot kommunikációját valósítja meg! Az előző feladatban ismertetett játékot játssza a két szolga robot, azzal a különbséggel, hogy a kisorsolt számokat a mesternek küldik el, és a mester robot dönt a vesztesről. A mester üzenetének megfelelően tolat a vesztes robot, majd kezdik elölről a játékot.

12/F6. Írjon programot, amely két robot kommunikációját valósítja meg! A mester robot egyenesen halad fehér felületen lévő fekete színű csíkig. A csíkot elérve 90°-ot fordul jobbra, majd ismét fekete csíkig halad egyenesen. A mozgását jellemző paramétereket küldi bluetoothon keresztül a szolga robotnak, amely fényszenzor nélkül bejárja a mester által megtett útvonalat. (Nehezítésként bonyolultabb útvonal is kijelölhető.)

13. FÁJLKEZELÉS

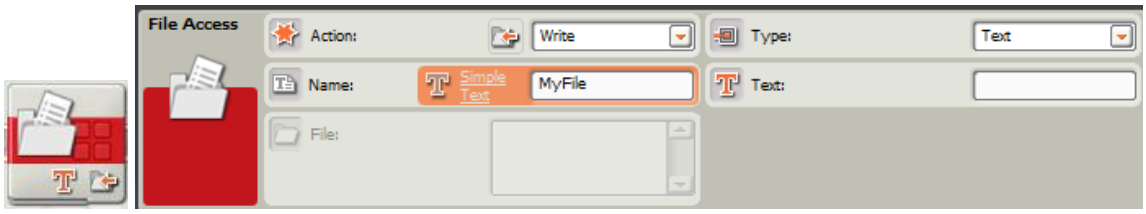
Nagy mennyiségű mérési adat tárolására nem elegendő az a változómennyiség, amit létre tudunk hozni a keretprogramban. Ezen kívül a változókban tárolt adatok csak a memóriában léteznek, így azok programon kívüli felhasználására (esetleg más programokkal történő feldolgozására) nincs lehetőség. A különböző 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 (szövegszerkesztő, 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 az NXT-G 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.

Az első esetben a megírt programkódban a megfelelő modulok beillesztésével adhatjuk meg a tárolás egyes lépéseit, míg a második esetben programtól függetlenül, tehát „manuálisan”, a képernyőmenün keresztül állíthatjuk be a kiválasztott szenzorok figyelését és a mért adatok tárolását.

13.1. Hagyományos fájlkezelés

A programvezérlő modul az *Advanced* ikoncsoport *File Access* ikonjának beillesztésével érhető el.

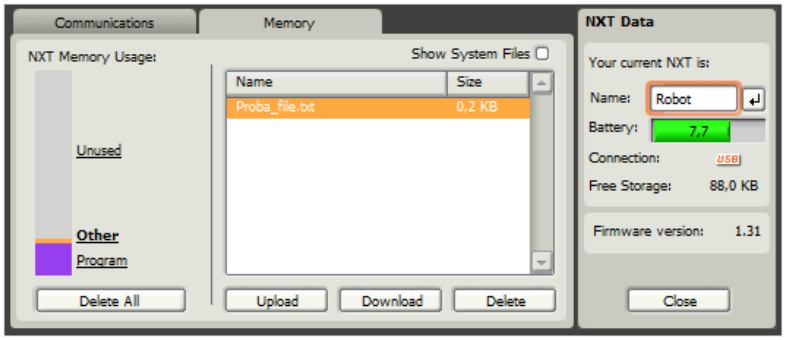


A paraméterek jelentését az alábbi táblázat foglalja össze:

A paraméter neve	A paraméter jelentése
Action	A végrehajtani kívánt fájlműveletet választhatjuk ki. A lehetőségek: *Read* – olvasás, egy létező fájlból történő adatok beolvasása. *Write* – írás, adatok írása a kiválasztott fájlba. *Close* – bezárás, fájl bezárása (a második oszlop paraméterei nem elérhetők). *Delete* – törlés, kiválasztott fájl törlése (a második oszlop paraméterei nem elérhetők).

Name	A fájl nevét lehet megadni. Ha még nem létezett ilyen fájl, akkor a rendszer létrehozza, és megjelenik a listában (*File* paraméternél).
File	A létező fájljaink listája. Rákattintva a listában a fájl nevére a *Name* paraméternél is megjelenik a fájlnév. A fájloknak a robot memóriájában kell lenniük.
Type	A fájl típusa adható meg. *Text* – szöveg vagy *Number* – szám választható. Mivel a szenzorok számokat adnak vissza eredményül, ezért a *Number* funkció a gyakoribb, de ebben az esetben is szövegfájlt hoz létre a rendszer.
Text/Number	A *Type* paraméternél választott beállítástól függ. Ha direkt akarunk a fájlba írni szöveget vagy számot (tehát nem a szenzorok által mért vagy a program által előállított értéket).

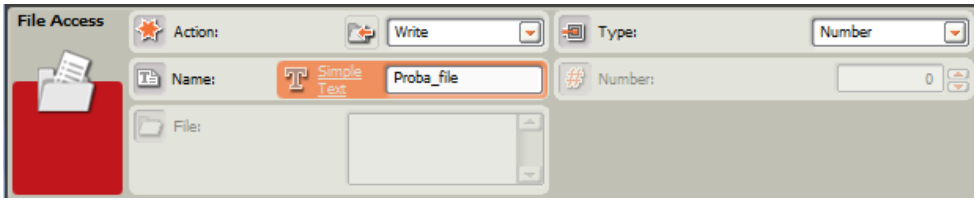
A programozásban a fájlkezelés több lépésből áll. A használat előtt a fájlt meg kell nyitni (ez a művelet különböző programnyelvekben két lépésből is állhat, az első lépés lehet egy logikai név, valamint a fájl használati módjára utaló kód megadása, majd a második lépés a tényleges megnyitás). A megnyitás után végezhetjük el a manipulációkat az adatokkal, pl. fájlba írás, olvasás. Majd a használat végén a fájlt le kell zárni. Ha lezárás nélkül lépünk ki, az adataink megsérülhetnek vagy elveszhetnek.
Az NXT-G nyelvben az ikon beillesztésével a rendszer automatikusan megnyitja a fájlt a kiválasztott műveletnek megfelelően. Ha írásra (*Write*) nyitottuk meg a fájlt, akkor létrehozza, ha még nem létezett. Ha már volt ilyen néven fájl, akkor az írási művelet hozzáfűzést jelent, vagyis a létező fájl végére kerülnek az új adataink, megtartva a korábbi tartalmat is. Ha üres fájlban szeretnénk adatokat tárolni, akkor egy még nem létező fájlnevet kell megadnunk, vagy le kell törölnünk (*Delete*) a már létező fájlt. Ezután következhet az adatok tárolása a fájlban, majd a használat után a lezárás (*Close*). Ha csak olvasni szeretnénk egy már létező fájlból, akkor használhatjuk a *Read* funkciót. Ezzel a fájl legelső adatához tudunk hozzáférni.
A fájl a létrehozása és lezárása után a robot memóriájában jön létre, a megadott néven. Innen tudjuk manuálisan átmásolni tetszőleges mappába az *Upload* gombon kattintva. A már számítógépen létező *txt* fájlunkat a robot memóriájába töltve (*Download*) elérhetővé válik programjaink számára.



Egy egyszerű használati módot mutat be a következő program.

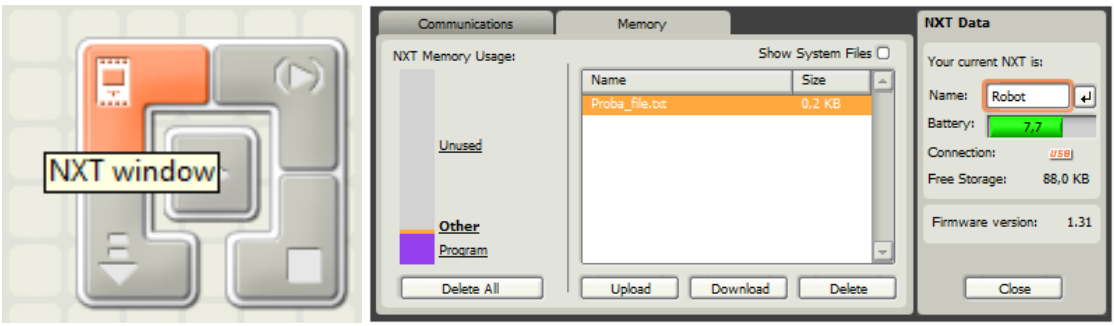
13/P1. Írjon programot, amelyet végrehajtva a robot állandó sebességgel halad előre egy akadály felé! Az ultrahang szenzorával mért értékeket 0,1 másodpercenként rögzíti egy fájlban (Number típusként). Az adatok rögzítését a robot 4 mp-ig végezze!

Az első ikon letörli az adott nevű fájlt. Természetesen ha még nem volt ilyen fájl, akkor ez nem szükséges. Elindul előre a robot 30-as sebességgel (nem túl gyorsan). A ciklus 4 mp-ig fut, és ebben történik meg a mérés és fájlban rögzítés. A fájlkezelés modul paraméterezését mutatja a következő ábra:



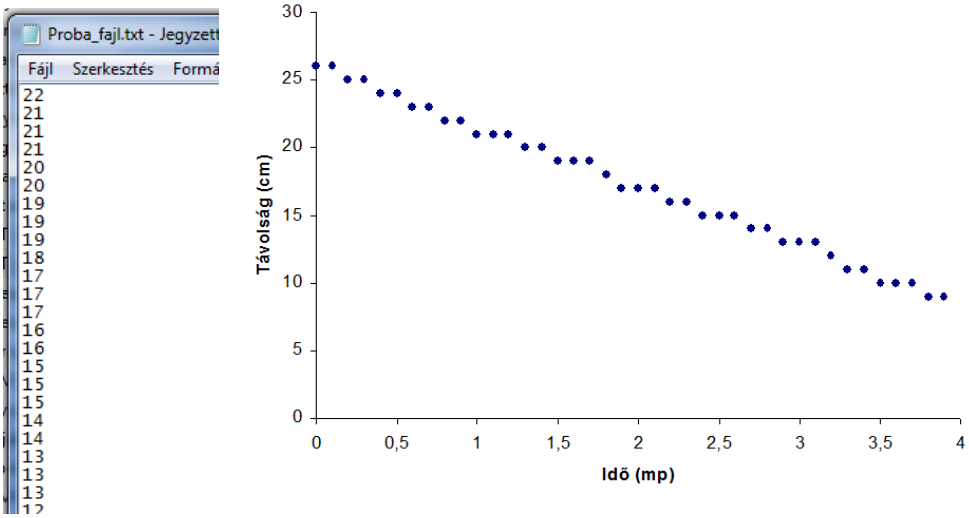
Az adatok paraméterátadással kerülnek a modulhoz. A 0,1 másodperces várakozás után a ciklus újra indul. A ciklus után a robot megáll, és a fájlt lezárjuk (*Close*).

A fájl ezután elérhető a robot memóriatérképét megjelenítő ikonnal, és az *Upload* gombbal átmásolható tetszőleges mappába.



Minden adat új sorba (bekezdésbe) kerül. 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 létrejött *txt* kiterjesztésű fájl egy részletét, valamint az adatokból készült grafikont mutatja az alábbi ábra:



A pontsorra illesztett egyenes meredeksége a robot sebességét adja meg cm/s mértékegységben. Ebben az esetben ez 4,25 cm/s = 0,153 km/h. A programban beállított érték 30-as volt. Ez a sebesség persze függ a kerekek átmérőjétől, az akku töltöttségétől, a tapadástól, stb., tehát csak konkrét robot esetében tudjuk a sebességet a szokásos mértékegységekben megbecsülni.

Egy már létező fájlból adatokat beolvasni és a robot képernyőjén megjeleníteni a *File Access* modul *Action* paraméterének *Read* beállítása mellett tudunk. Ebben az esetben mindig a fájlban szereplő legelső adatot olvassa be a program. Ha a modult szimbolizáló ikont újra beillesztjük ugyanabba a programba, akkor a következő beolvasás már a második adatot adja eredményül, és így tovább. Ha adott sorszámú adatra van szükségünk, akkor például ciklusba helyezhetjük a modult és így érhetjük el a szükséges adatot. A fájl lezárása és újbóli megnyitása után ismét elölről kezdhetjük az adatok olvasását. Minden adatunk új sorban kell, hogy elhelyezkedjen a fájlban, mert a beolvasásnál a program a sorvégjelet (enter) figyeli az adatok elkülönítésére.

Az adatok írásának és olvasásának fájlbeli pozícióját (sorszámát) úgy határozza meg a rendszer, hogy minden fájlhoz tartozik egy virtuális mutató, amely valamelyik sorára mutat. A következő adat írása és olvasása mindig abból a sorból történik, amelyre ez a mutató hivatkozik. Amikor egy írási vagy olvasási műveletet elvégzünk, akkor a mutató automatikusan a következő sorra pozícionál. A fájl olvasásra történő megnyitásakor automatikusan a legelső sorra, míg írásra történő megnyitásakor a legutolsó utáni sorra mutat. Így a beolvasás a fájl elején, míg az írás a fájl végén kezdődhet.

Ha számokat olvasunk be egy textfájlból, akkor a beolvasás addig történik, amíg a rendszer a sorban található karaktereket számként tudja értelmezni. Tehát ha egy sorban szóközzel elválasztott számok találhatók, akkor csak az első számot olvassa be. Tizedes határolóként a pontot kell használni.

Szöveg beolvasásánál mindez nem játszik szerepet, mert a fájlban eleve szövegként tároltuk az adatokat (a számokat is). A számok memóriába történő visszaolvasásánál viszont egy átalakítást (konverziót) is végre kell hajtani, ezért a rendszer csak a számmá alakítható adatokat tudja visszaolvasni.

A számok beolvasásának értelmezésében az alábbi példák segítenek:

A szövegfájl egy sora	Számként beolvasva a memóriába kerül	Magyarázat
12 3 5	12	A beolvasásánál a 2-es számjegy utáni karaktert (szóköz) már nem tudta számjegyként értelmezni a rendszer.
29w13	29	A beolvasásánál a 9-es számjegy utáni karaktert (w) már nem tudta számjegyként értelmezni a rendszer.
45,8	45	A beolvasásánál az 5-ös számjegy utáni karaktert (vessző) már nem tudta számként értelmezni a rendszer, mivel a tizedes határoló a pont.
67.23	67.23	Értelmezhető számként.

Ezek a problémák leginkább akkor léphetnek fel, ha olyan fájlokat próbálunk a robottal beolvastatni, amelyek valamilyen külső szerkesztőprogrammal készültek, hiszen ha a robot írja fájlba az adatokat, akkor azok a szintaktikai szabályoknak megfelelően kerülnek tárolásra.

13/P2. Írjon programot, amelyet végrehajtva a robot sorsol 10 db 1 és 100 közötti véletlen számot, ezeket egy fájlban tárolja! A fájl lezárása után, írassuk a képernyőre a hetediknek kisorsolt számot!

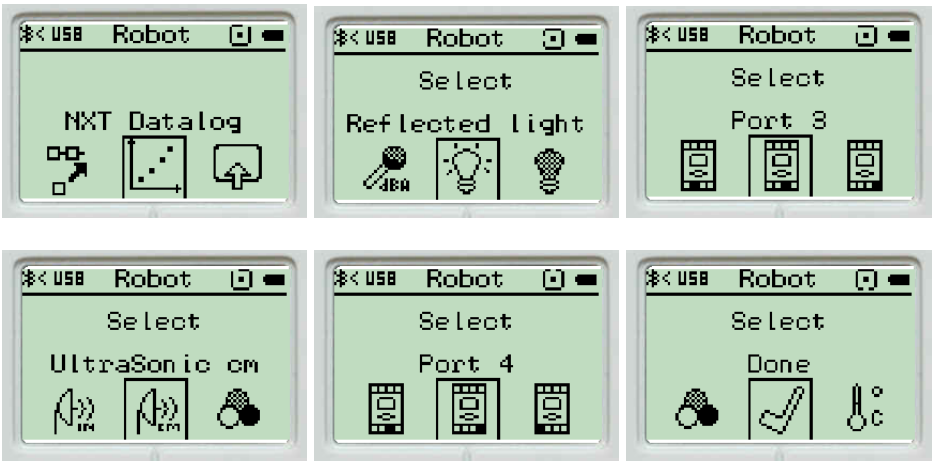
A megoldás forráskódját két részletben mutatjuk be. Az első részben az adatok tárolása történt meg. Létrehozott szövegfájlként az alapértelmezett *MyFile.txt* állományt használtuk. Mivel lehet, hogy ilyen néven már létezett állományunk, ezért első lépésben a fájl törlésével kezdtük a programot. Ezután egy 10-szer lefutó ciklusban generáltuk a véletlen számokat és írtuk őket a fájlba. Minden kisorsolt szám új sorba került (automatikusan). A fájl lezárását követően az olvasási művelet a fájl első bejegyzésével kezdődik.

A második programrészletben történik meg a számok beolvasása a memóriába. Mivel csak a hetedik számra van szükségünk, ezért egy hatszor lefutó ciklus tartalmazza a *File Access* modult. Így a fájlmutató hatot lép előre és a következő művelet a hetedik sorra fog vonatkozni. Az újabb olvasási művelet, egy konverzió után megjeleníti a számot a képernyőn. A fájlból képernyőre írás során két konverzió történt. Egyrészt a szövegként tárolt számjegyeket a rendszer számmá alakította a memóriába történő beolvasás során, másrészt a számot visszaalakította szöveggé a megjelenítéshez. A fájl lezárását követő várakozás a képernyőolvasást teszi lehetővé.

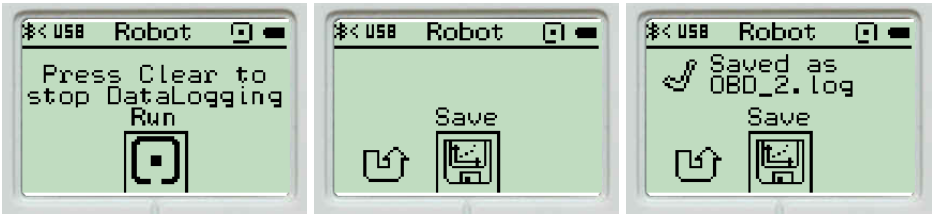
Érdemes ellenőrizni, hogy valóban jó érték jelent meg a képernyőn. A *MyFile.txt* állományt a számítógépre másolva (*Upload*), és ott megnyitva.

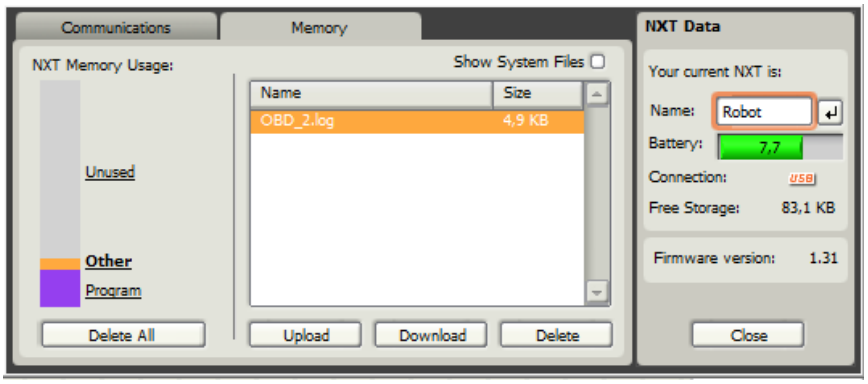
13.2. Datalog fájlok

A fájlkezelés egy másik módja a *Datalog* fájlok használata. A szerepe az, hogy ha nem akarunk programot írni, csak a szenzorok által mért értékeket rögzíteni és fájlban tárolni, akkor erre legyen lehetőségünk. A rögzítést a robot képernyőmenüjében kell kezdeményezni, az NXT *Datalog* menüben.
Itt kiválaszthatjuk, hogy melyik szenzor értékeit szeretnénk folyamatosan tárolni (természetesen a portot is meg kell adni). Akár több szenzor értékeinek egyidejű figyelésére és rögzítésére is van lehetőség. Miután ezeket beállítottuk, indíthatjuk az adatok rögzítését.



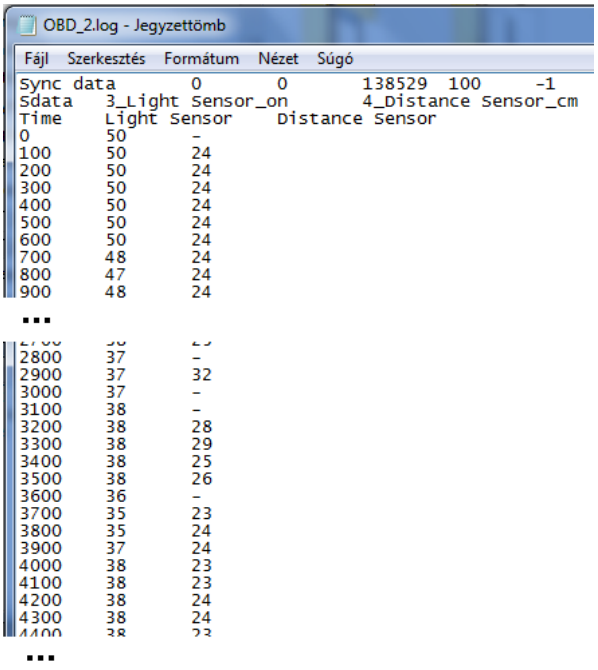
A rögzítés mindaddig fut, amíg a menüben le nem állítjuk. Mivel az adatokat tartalmazó fájl a robot memóriájában kerül tárolásra, ezért a memória kapacitása határt szab a rögzítés időtartamának. A rögzítés során nem tudunk kilépni a *Datalog* menüből, így nem indítható el program. Tehát a robot és szenzorainak esetleges mozgását manuálisan kell elvégeznünk.
A rögzítés leállítása után a memóriából, a létrejött fájl az *Upload* gombbon kattintva másolható a számítógép megadott mappájába.





A fájl *log* kiterjesztést kap, és szabványos szövegfájlként jön létre, tehát bármilyen szövegszerkesztő programmal megnyitható és olvasható.

Példaként egy olyan fájl tartalmát mutatjuk meg, amelynél a 3-as portra kötött fényszenzort és a 4-es portra kötött ultrahang szenzort egyidejűleg figyeltük, és tároltuk a mért értékeit. A robotot és szenzorait időnként kézzel mozgattuk.



Az első oszlop számadatai a mintavétel időpontját mutatják a rögzítés indítását tekintve 0-nak, milliszekundum mértékegységben (0,1 másodpercenként történt a mintavétel). A második számoszlop adatai a fényszenzor adott időpillanatbeli értékei, míg a harmadik számoszlop az ultrahangszenzor értékei. Az ultrahangszenzor „lassabb” működése következtében nem minden esetben adott vissza értéket, így ezeken a helyeken „-” jel látható.

Az adatrögzítés bemutatott két módja alkalmas arra, hogy a robothoz kapható szenzorokkal akár természettudományos méréseket végezzünk. A mérési pontosság nem tudományos érvényű, de arra alkalmas, hogy a fizikai mennyiségek közötti kapcsolatok tendenciáit vagy az egyes szenzorok által mérhető paraméterek jellegét szemléltessük.

13.3. Gyakorló feladatok

13/F1. Egyszerű szövegszerkesztő programmal készítsen egy textfájlt (*txt* 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!)

13/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.

13/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!

13/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!

13/F5. A 12/F4-es feladatot oldja meg úgy, hogy a két robot által kisorsolt számokat a mester robot egy fájlban is rögzíti. A számpárok elválasztására használja a 0 értéket, és egy sorsoláspár eredménye kerüljön a fájl egy sorába!

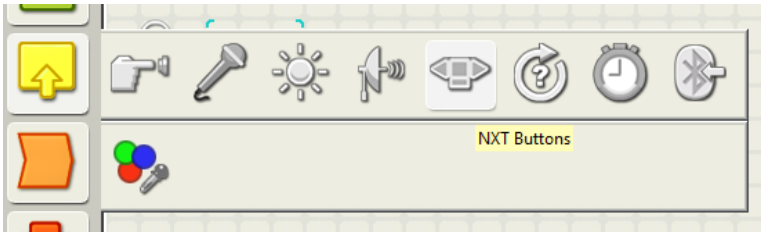
13/F6. Írjon programot, amelyet végrehajtva a robot egy enyhe lejtőn gurul lefelé egy akadály felé! A motor kerekei szabadon forognak, nincsenek a motorhoz csatlakoztatva. A robot ultrahagszenzorával mért értékeket, az akadálytól való távolságot, 0,1 másodpercenként rögzítse egy fájlba! A mérés végeztével a fájlt töltse át a számítógépre, és táblázatkezelő program segítségével készítsen a mért adatokkal grafikont!

13/F7. Írjon programot, amelyet végrehajtva a robot két különböző szövegfájlban tárolt egy-egy szót, egy harmadik szövegfájlba másol! A két szót egy szóköz válassza el, és a fájl ugyanazon sorába kerüljenek! (A feladat továbbfejlesztéseként olyan állományok összemásolását végezze el, amelyek több sorban elhelyezett szavakat tartalmaznak!)

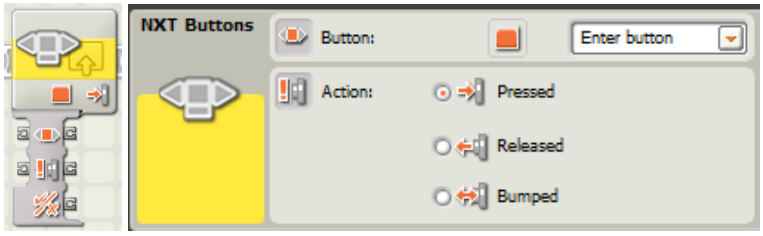
14. EGYÉB MODULOK

14.1. Nyomógombok használata

Az NXT tégla tartalmaz négy darab nyomógombot, amelyek közül három programban is használható: a balra illetve jobbra mutató háromszögek, illetve a középső (narancssárga) „*Enter*” gomb. A szürke „*ESC*” funkciójú gomb nem használható programban, hiszen ez biztosítja pl. az elindított programjaink leállítását. Így ennek átdefiniálásával a programjainkat nem lehetne leállítani.
A vezérlést biztosító modul a *Sensor* csoporton belül található *NXT Buttons*.



A modul használata az ütközésérzékelőére hasonlít. A gombok esetében megadhatjuk, hogy mikor adjanak vissza igaz vagy hamis értéket: benyomott állapotban (*Pressed*), felengedett állapotban (*Released*), vagy akkor, ha átmenet történik két állapot között: benyomott állapotból felengedett állapotba kerül (*Bumped*).
A tégla nyomógombjai feltételes elágazások, ciklusok vezérlésére is használhatók.

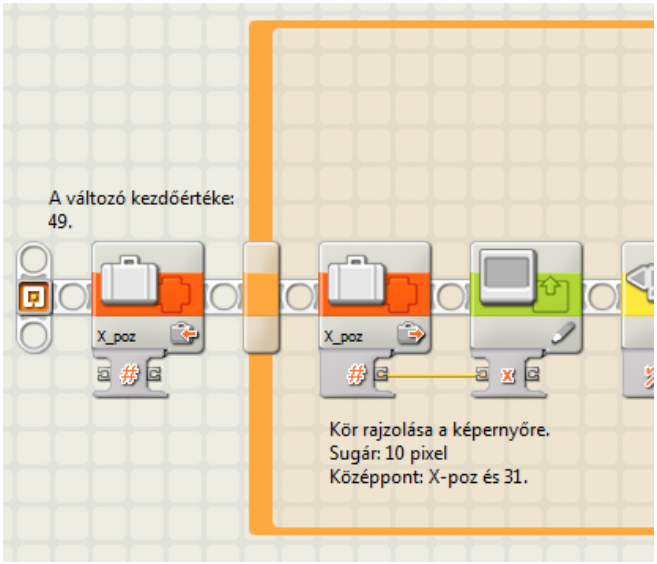


A legördülő listában három elem szerepel, annak megfelelően, hogy melyik gombot szeretnénk használni a programban: *Enter button* – középső, narancssárga gomb, *Left button* – balra mutató háromszög alakú gomb, *Right button* – jobbra mutató háromszög alakú gomb.
Egy példán keresztül mutatjuk be a használatukat.

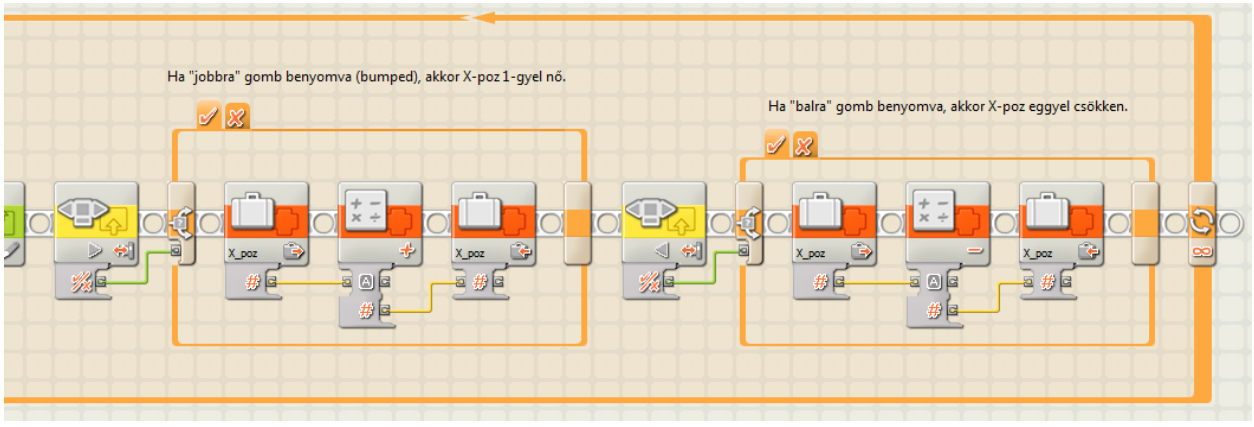
14/P1. Írjon programot, amelyet végrehajtva a robot a képernyő közepén megjelenít egy 10 pixel sugarú kört, amelyet a „balra” és „jobbra” gombok segítségével lehet 1 pixelenként mozgatni!

A programot két részletben mutatjuk be. Az első programrészletben történik az előkészítés és a képernyőre rajzolás. Létrehoztunk egy X_poz nevű változót, amelyben a képernyőn megjelenő kör aktuális helyzetének vízszintes koordinátáját (x) tároljuk. A ciklus előtt 49-es kezdőértéket állítottunk be, körülbelül ez felel meg a képernyő közepének. A végtelen ciklus elindítása után felrajzoljuk a képernyőre a kört, amelynek sugara 10 pixel, a középpontjának függőleges (y) koordinátája: 31, ami a

program során nem változik. A középpont x koordinátáját az X_poz változóban paraméterátadással állítottuk be.



A program második részében, a végtelen cikluson belül programoztuk le a gombok benyomására bekövetkező eseményeket. A két gombhoz tartozó programszerkezet hasonló. Mindkét gomb esetén egy-egy elágazást vezérel a visszaadott érték. Ha a visszaadott érték hamis, akkor nem kell utasítást végrehajtani, így az elágazás alsó szálát nem jelenítettük meg a példaprogramban. Igaz értékre (a megnyomás hatására) a felső szálon lévő utasítások futnak le. A jobbra mutató háromszög alakú gomb benyomásakor eggyel növeljük az X_poz változó tartalmát, míg a balra mutató gomb esetén eggyel csökkentjük. Természetesen az X_poz megváltoztatott értékét eltároljuk a változóban, így felülírva a régi értéket. Ezzel vége is a ciklusmagnak, hiszen kezdődhet elölről, a rajzolással. A gombok esetében a vezérlő feltételt *Bumped*-re állítottuk, így csak abban az esetben történik meg a változók értékének növelése vagy csökkentése, ha a gomb állapotában változás lép fel: benyomott állapotból felengedett állapotba kerül. A *Pressed* funkciót választva – a ciklus futási sebességének megfelelően – egy gombnyomás ideje alatt sokszor megtörténne a változó értékének módosulása, hiszen amíg a rendszer benyomott állapotúnak érzékeli a gombot, addig többször is lefut a ciklus.

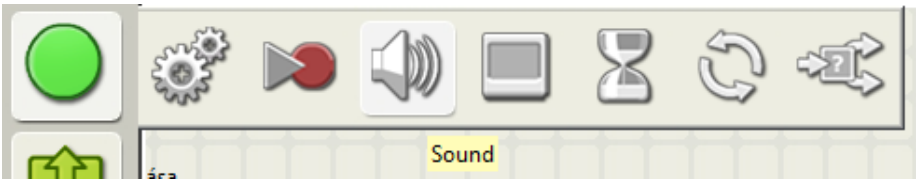


A program nem figyeli a képernyő szélének elérését, ez továbbfejlesztési lehetőségként adódik. Szintén kiegészíthető a program egy olyan funkcióval, amely az *enter* gomb benyomására visszaállítja a kiinduló helyzetbe a kört.

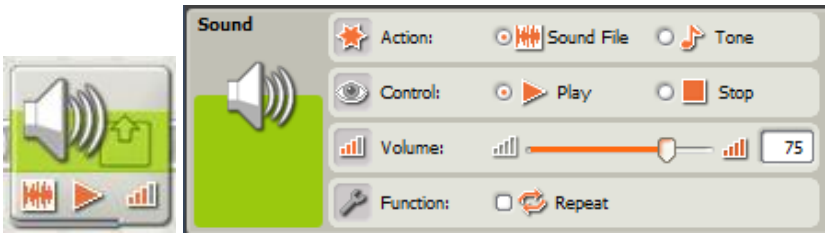
A program elvi hibája, hogy ha nem történik gombnyomás, a ciklus akkor is folyamatosan fut, és a kört valamennyi lefutás során újrarajzolja (a képernyőtörlés be van kapcsolva: *Clear*). Ezen pl. úgy lehetne változtatni, hogy a rajzoló modult a két feltételes elágazásba tesszük (mindkettőbe), így a tényleges újrarajzolás csak akkor történik meg, ha változás van az `X_poz` értékében.

14.2. Hanglejátszás

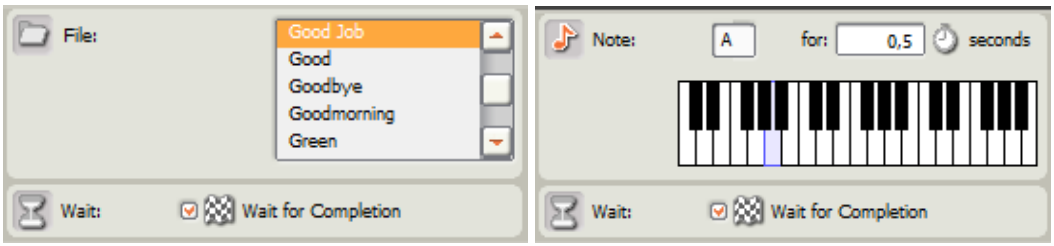
Az NXT robot rendelkezik egy egyszerű beépített hangszóróval, ami dallamok, hangok lejátszására alkalmas. A modul a *Common* csoportban található *Sound*.



Beillesztve a megfelelő programhelyre, a képernyő alján megjelenő paraméterterület tartalma aszerint változik, hogy a kétféle üzemmód közül melyiket választjuk. Lehet előre elkészített hangfájlokat lejátszani (néhányat kapunk a szoftverrel), vagy adott frekvenciájú hangot megszólaltatni.



További paraméterek hangfájl esetén illetve szóló hang esetén:

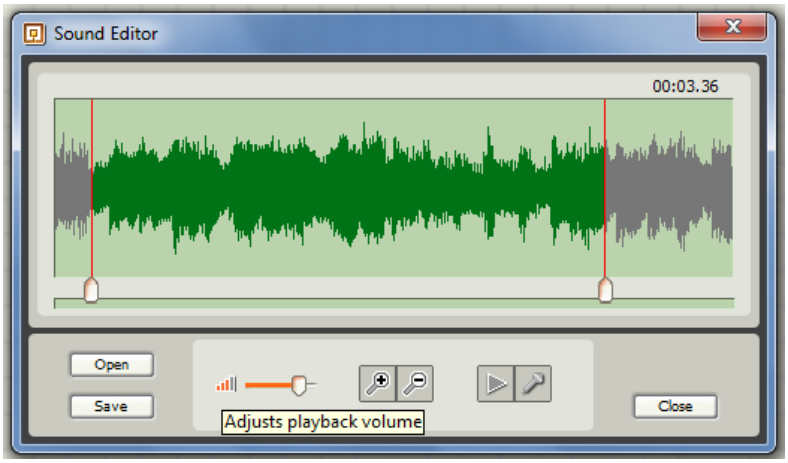


Hangfájl esetén a listában megjelenő zenék közül választhatunk, hang esetén pedig a zongorabillentyűzeten választhatjuk ki a megszólaltatható hangot. Ez utóbbi esetben azt is megadhatjuk, hogy milyen időtartamig szóljon a hang (három tizedesjegy pontossággal másodpercben). A hangfrekvencia számszerű megadására közvetlenül nincs lehetőség, de paraméterátadással a modul legördíthető paramétereinél szerepel a csatlakozási pont, ahol ezt meg lehet tenni.

A további paraméterek jelentése:

A paraméter neve	A paraméter jelentése
Action	Kiválaszthatjuk, hogy hangfájlt szeretnénk lejátszani (*Sound File*) vagy szóló hangot (*Tone*).
Control	Kiválaszthatjuk, hogy lejátszani szeretnénk a hangot (*Play*), vagy leállítani a lejátszását (*Stop*)
Volume	A lejátszás hangerejét állíthatjuk be 0-100 közötti skálán. A hangerőt befolyásolja a robot képernyőmenüjében beállított hangerő is.
Function	A hang lejátszásának ismétlését lehet bekapcsolni.
Wait for Completion	Ha a jelölőnégyzet be van kapcsolva, akkor a programban a végrehajtás során nem lépünk tovább, amíg a hang teljes időtartamát le nem játszotta, egyébként (kikapcsolt állapotban) a hang a háttérben szól, tehát a rendszer elkezdi lejátszani a hangot, és továbblép a következő utasításra.

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 képernyő bal és jobb oldalán található 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 program színkóddal jelzi, ha a kijelölt rész mérete megfelelő. A hangfájl a robotra töltve a programjaink elől foglalja a memóriát, és mivel a memória mérete korlátozott, ezért nem érdemes nagyméretű fájlokat használni. Zöld színnel jelenik meg a hanghullám a képernyőn, ha már elegendően kicsi a mérete a memóriában való tároláshoz, egyébként piros színnel. A mentést a *Save* gombon történő kattintással végezhetjük el a név megadása

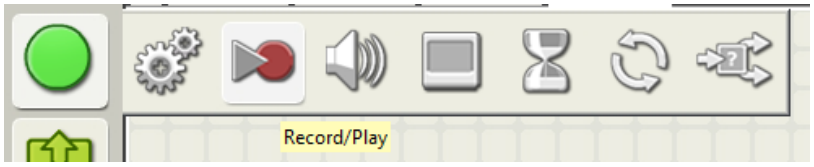
után. Alapértelmezésben a program telepítési mappájába történik a mentés, azon belül az *engine/Sounds* mappába.

14.3. Programleállítás

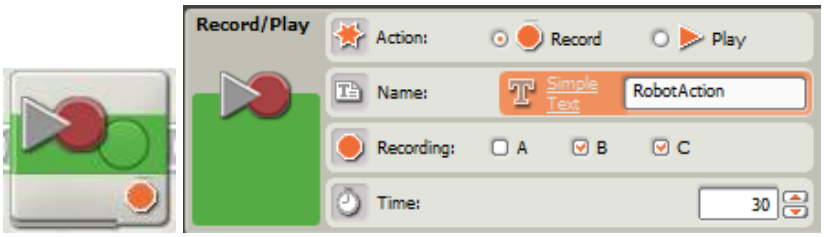
Bizonyos esetekben szükség lehet arra, hogy adott szituációban leállítsuk a programot. Így például ciklusok belsejében vagy elágazásokban is használhatjuk a leállító utasítást, amelynek hatására a program úgy ér véget, hogy esetleg nem is hajtott végre minden benne szereplő utasítás. A modul a *Flow* csoportban található és a programszál adott helyére illesztve, végrehajtásakor leállítja annak futását, és visszatér a nyitó képernyő.

14.4. Mozgásrögzítés

Bár kevésbé hasznos, de látványos eszköz áll a rendelkezésünkre a robot mozgásának rögzítéséhez, a *Common* csoport *Record/Play* modulján keresztül.



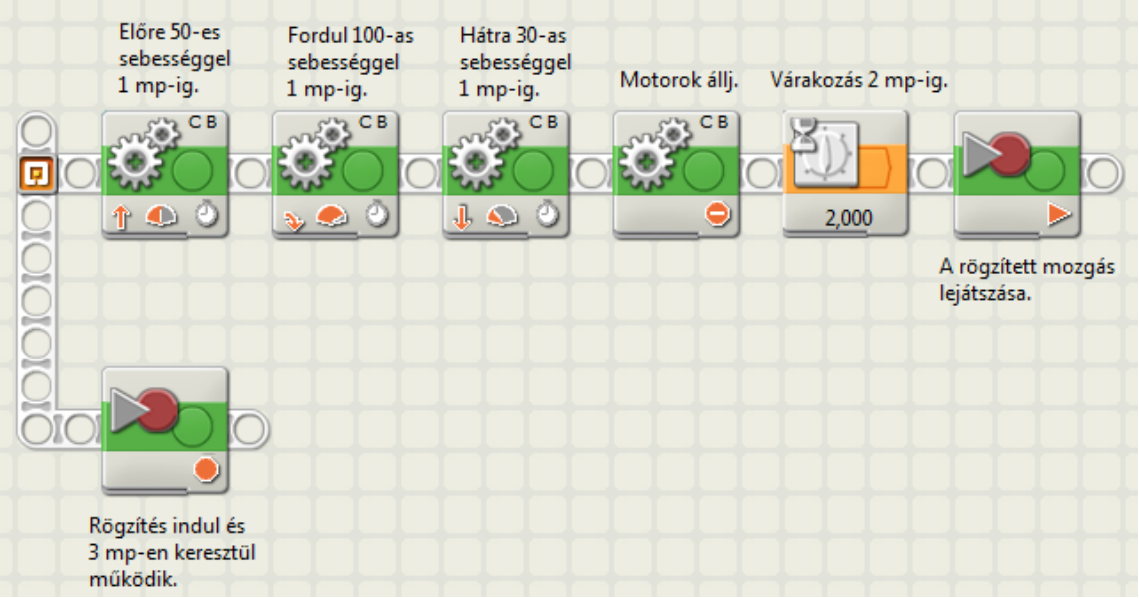
A programmodul alkalmas arra, hogy a robot mozgását, annak minden egyes lépését eltárolja és később visszajátszva megismételje.



A paraméterek és jelentésük:

A paraméter neve	A paraméter jelentése
Action	Azt választhatjuk ki, hogy a robot mozgását rögzíteni szeretnénk (*Record*), vagy a rögzített mozgást lejátszani (*Play*).
Name	A rögzített mozgássor nevét, azonosítóját adhatjuk meg. *Play* funkció esetén egy listából választhatjuk ki az adott nevű lejátszandó rögzítést.
Recording	A rögzíteni kívánt motorok portjait jelölhetjük be. A rögzítés során csak azokat a motormozgásokat jegyzi meg, amelyek a kiválasztott portokra vannak csatlakoztatva. (*Play* funkció esetén nem aktív.)
Time	A rögzítés ideje másodpercben. (*Play* funkció esetén nem aktív.)

A rögzítés használata során külön programszálon kell elhelyezni a mozgásrögzítő modult. Egyéb esetben a program utasításainak végrehajtása addig áll, amíg le nem telik a rögzítésre beállított idő.
A következő program a használatot szemlélteti. Három lépésből álló mozgássort rögzítünk. Minden lépés 1 mp időtartamú, és a B illetve C portra csatlakoztatott motorok végzik, így ezeket rögzítjük. Az egymást követő lépésekben előre mozgás, majd gyorsabb fordulás, és végül lassabb tolatás szerepel. A mozgássor végén 2 mp-es várakozás után lejátsszuk a rögzített adatsort, és a robot megismétli a mozgást.



Szerepe mindennek akkor lehet, ha egy bonyolultabb programvezérelt mozgássort rögzítünk. Például a robot feladata, hogy megkeressen egy adott helyet az érzékelői által szolgáltatott adatok alapján (pl. fekete sáv egy fehér alapon). Ezután ugyanaz a mozgássor megismételhető a lejátszás során szenzorhasználat nélkül. A rögzített fájl áttölthető a számítógépre és onnan egy másik robotra, amely azt lejátszhatja. Így egy szenzorokkal ellátott robot feltételvezérelt mozgását egy szenzorok nélküli robot is meg tudja ismételni.

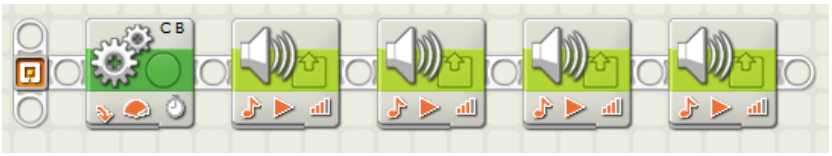
14.5. Saját utasításblokk

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, taszkként megírni, amit egyszerű hivatkozással lehet beilleszteni a megfelelő helyre. Az NXT-G nyelvben 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 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 *Edit* menü *Make A New My Block* menüpontját választva menthető az elkészített modul.

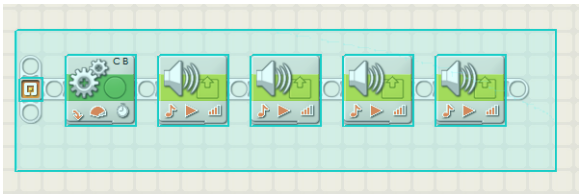
Egy egyszerű példán mutatjuk be a használatot.

14/P2. Írjon programot, amelyet végrehajtva a robot egyenesen mozog előre, amíg 15 cm távolságra nem kerül egy akadálytól, ekkor ütközésig tolat! Ütközéskor gyorsan forog 3 mp-ig, és lejátszik egy 4 hangból álló dallamot. Ezután mindezt kezdi elölről.

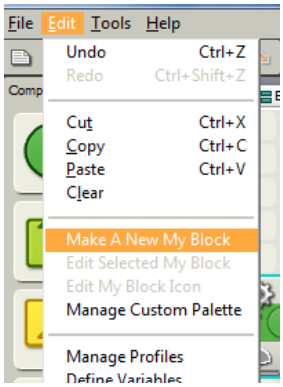
Az ütközés utáni pörgést és dallamjátszást fogjuk elkészíteni egy saját „Hurra” nevű modulként.
Először azokat az utasításokat programozzuk le, amelyek a modulba kerülnek.



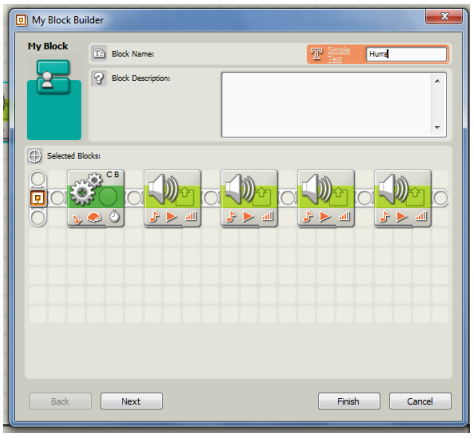
A program első utasítása egy 3 mp-es forgást tartalmaz 100-as sebességgel. A következő négy modul egy-egy különböző hang 0,5 mp-es lejátszását eredményezi.
A modulokat kijelölve:



az *Edit* menüben van lehetőségünk a mentésre.

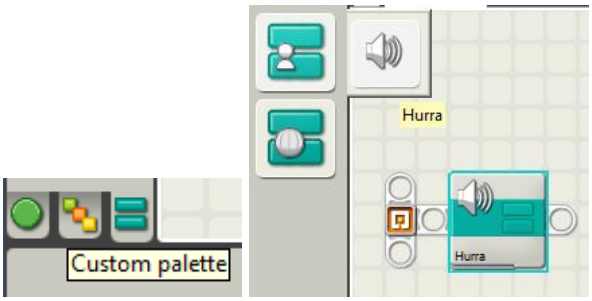


Ekkor elindul egy varázsló, amelynek első lépésében megadhatjuk a modul nevét és egy rövid leírást a működéséről.

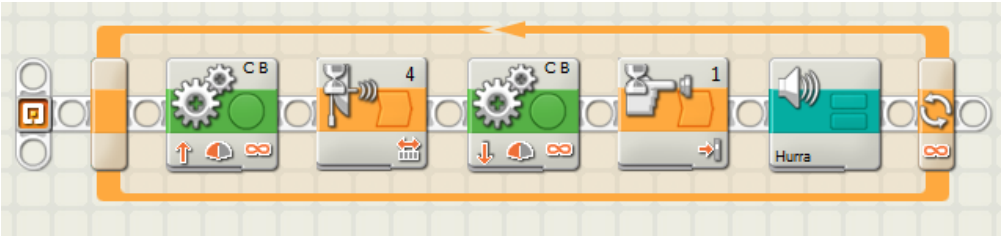


Második lépésként ikont választhatunk a saját modulnak.

A *Custom palette* lapon, a *My Blocks* csoportban jelenik meg az elkészített modulunk.



Ha készen van a saját modul, akkor ugyanúgy használhatjuk, mint az eddigieket. Szabadon beilleszthetők programjainkba. A fentebb megadott feladat megoldása a létrehozott saját modul beillesztésével is történhet.



A modulokat a rendszer ugyanúgy *rbt* kiterjesztéssel menti el, mint a programjainkat. Ez azt is jelenti, hogy bármelyik már elkészített programunk használható modulként. Az *Edit* menü *Manage Custom Palette* menüpontján keresztül másolhatjuk, törölhetjük a már elkészült programjainkat, moduljainkat. A menüpont kiválasztásakor az operációs rendszer fájlkezelője indul el, így minden ott használható eszköz rendelkezésünkre áll. Ezen a menüponton keresztül nézhetjük meg, hogy a rendszer melyik mappába mentette az elkészült modulunkat (*My Blocks*).

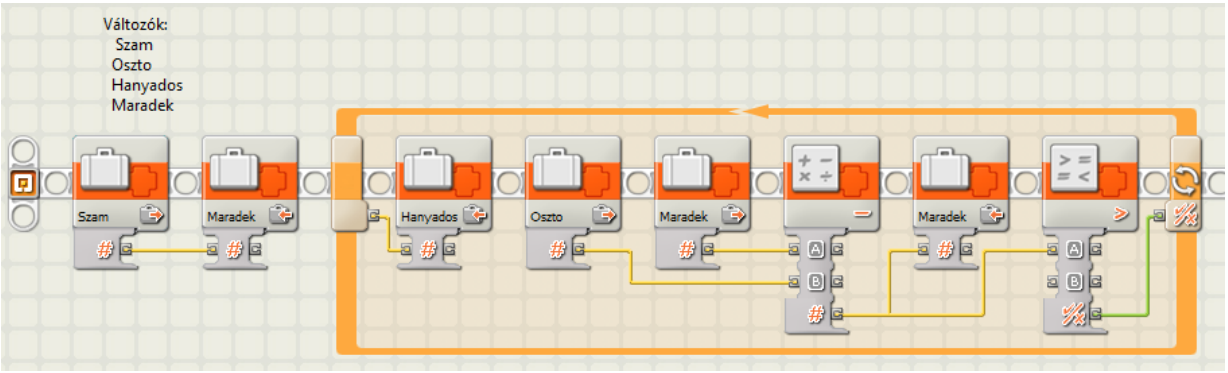
Sajnos az elkészített moduljainknál nem hozhatunk létre legördülő paraméterlistát, így a blokk és az őt tartalmazó program közötti adatcsere csak változókon keresztül történhet meg. Ügyelni kell arra, hogy ugyanazokat a változóneveket és típusokat használjuk a modulon belül és kívül. Ha elfelejtettük volna, hogy milyen utasításokat, változókat tartalmazott a saját modulunk, vagy módosítani akarjuk a tartalmát, akkor elegendő kettőt kattintani a programba beillesztett modul ikonjára, hogy egy külön lapon megnyissa a rendszer, ahol szabadon szerkeszthetjük a benne szereplő utasításokat.

A fenti példánál egy kicsit bonyolultabb feladaton keresztül mutatjuk be, hogyan lehet olyan modult létrehozni, amely adatokat tud fogadni és visszaadni az őt tartalmazó programnak.

14/P3. Írjon programot, amely, egy 0 és 100 közötti véletlen számot eloszt hárommal, és a képernyőre írja az osztási maradékot, valamint a hányadost! A program 5 mp várakozás után fejeződjön be! (Ezt lehet úgy továbbfejleszteni, hogy nem hárommal osztunk, hanem az osztó is véletlen szám.)

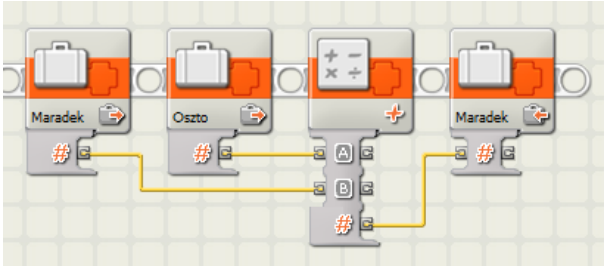
Mivel a programnyelvben nem áll rendelkezésre (egyelőre) maradékos osztást végző modul, ezért készítsünk sajátot, ami általánosan bármilyen nemnegatív számra működik.

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. Négy szám típusú változót hozunk létre: Szam, Oszto, Hanyados, Maradek néven. A változók neve utal a bennük tárolandó későbbi tartalomra.



Első lépésként a Szam változó tartalmát áttöltjük a Maradek nevű változóba, mert az algoritmus során a kezdeti számot folyamatosan változtatnunk kell, és azt szeretnénk, hogy megmaradjon az eredeti szám is változatlanul. Az osztást kivonások egymásutánjával valósítjuk meg. A Maradek változóból (aminek kezdőértéke megegyezik az eredeti számmal) elkezdjük kivonni az osztót mindaddig, amíg az eredmény nullánál nagyobb. Ha elértük a nullát vagy valamely negatív értéket, akkor kilépünk a ciklusból. A kivonás eredményét folyamatosan eltároljuk a Maradek nevű változóban. Ahányszor lefutott a ciklus, annyiszor tudtuk kivonni az osztót, tehát a ciklusváltozó értéke egyben a hányadost fogja megadni. Ezt tároljuk a Hanyados nevű változóban.

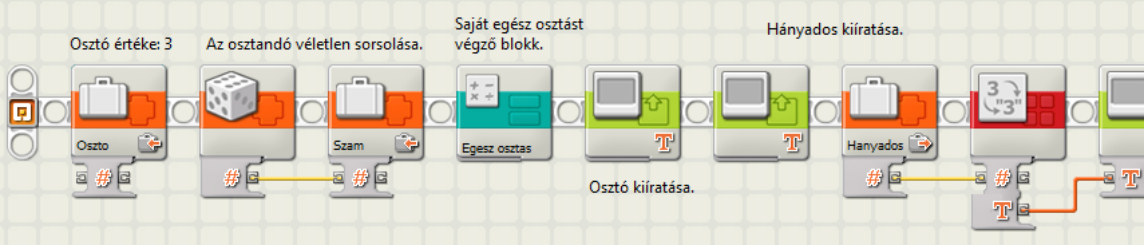
A ciklusból kilépve a Maradek nevű változó tartalma 0 vagy negatív szám. Mindenképpen nagyobb, mint az osztó -1-szerese. A tényleges maradékot úgy kapjuk, hogy a Maradek változóhoz hozzáadjuk az osztót (a maradék komplementere szerepel a ciklusból kilépés után a Maradek változóban).



Miután az összegzést elvégeztük, az eredményt a Maradek változóban tároljuk, így matematikailag helyes értékek kerültek minden változóba.

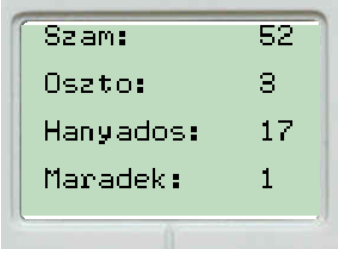
A modul használatához két bemenő adatra van szükség: a számra és az osztóra. Eredményül az osztás utáni maradékot és a hányados egész részét adja vissza. Mindezt persze csak akkor, ha a programban, ahová beillesztjük, ugyanezeket a változóneveket használjuk.

A program egy részlete, amely eleget tesz a feladatkiírásnak:



A változók létrehozása után az osztó értékét 3-ra állítottuk, míg a Szam változóba egy véletlen szám került. Ezután kerül sor a saját blokk utasításainak futtatására. A program további részeiben a képernyőre írattuk valamennyi adatot, amit előállítottunk.

A képernyőkép:

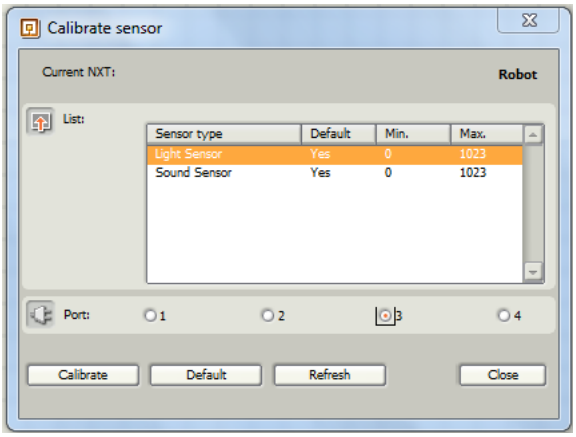


14.6. Szenzorok kalibrálása

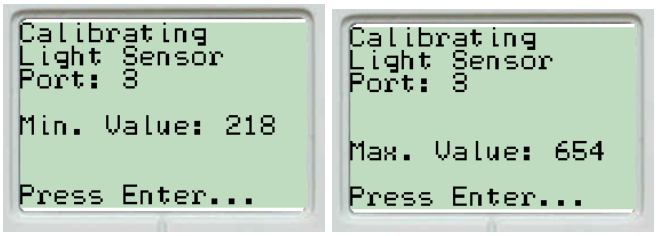
A robot fény- és hangszenzora csak relatív értékeket képes visszaadni. Ez azt jelenti, hogy amíg például az ultrahang szenzor az akadály távolságát képes mérni centiméterben, addig a fényszenzor egy alapértékhez viszonyított 0-100 közötti, mértékegység nélküli értéket ad vissza. Ez nem a színt jelenti, hanem a felület fényintenzitásától függő és a megadott értékhatárok közé eső értéket. Csak azt tudjuk állítani, hogy a visszaadott érték arányos a felületről visszavert fény intenzitásával. Az arányosság mindig feltételez egy kezdőértéket. Ezt a kezdőértéket tudjuk a kalibrálással beállítani. Megadhatjuk a kezdőérték minimumát és maximumát. A megadott minimum lesz a 0 érték, míg a maximum a 100. Minden, a minimálisnál kisebb fényintenzitást produkáló felületre nullát fog a szenzor mérni és hasonlóan 100 lesz a kalibrálás során megadott értéknél nagyobb fényintenzitású felületen mért érték. Ennek akkor lehet szerepe, ha a felületünk nem homogén színű, hanem kisebb-nagyobb eltérések váltakozása figyelhető meg, esetleg a megvilágítás miatt ingadozik az azonos színű felületeken visszaadott fényszenzor érték. Ilyenkor a kiválasztott alapszín értékét 0-nak vagy 100-nak választva a tőle lefelé (nulla esetén) vagy felfelé (száz esetén) eltérő értékeket a robot nem érzékeli.

A fényszenzorhoz hasonlóan a hangszenzort is kalibrálhatjuk. A mérési skálája hasonlít a fényszenzoréhoz (0-100 közötti relatív érték). A kalibrálást menüből és programból is el lehet végezni. A beállított minimum és maximum értékek mindaddig érvényesek lesznek, amíg vissza nem állítjuk a kezdeti, kalibrálás előtti alapértelmezett paramétereket (*Default*). A menüből történő kalibrálás a *Tools*

menü, *Calibrate Sensors* menüpontján keresztül érhető el. Első lépésben a szenzor típusát kell kiválasztanunk és a csatlakozási portot. Majd a *Calibrate* feliratú gombon kattintva elkezdődik a mérés. A folyamatot a fényszenzor esetén mutatjuk be.



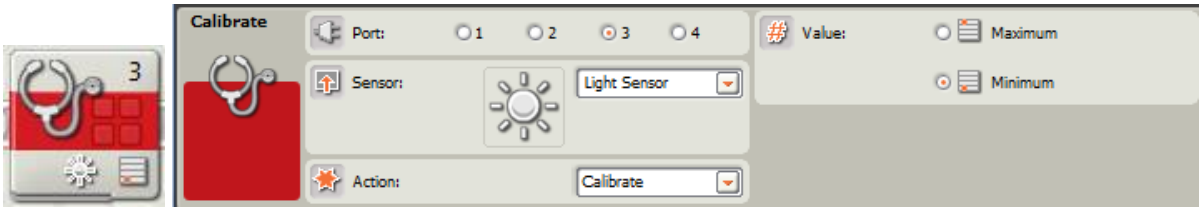
Az ábrán látható értékeket beállítva és a *Calibrate* gombra kattintva elindul a kalibrálás folyamata. Mindez a robot képernyőjén keresztül követhető és vezérelhető. Mind a minimális, mind a maximális értéket be kell állítanunk. Célszerű a robotot olyan pozícióba helyezni, ahol a fényszenzora a megfelelő színű felület fölött van. A téglán lévő vezérlőgombokkal lehet a kijelzett értéket elfogadni és továbblépni.



A robot fényszenzora a valóságban 0-1023 közötti skálán mér és ezt az adatot alakítja 0-100 közé egy lineáris transzformációval. A példán bemutatott esetben a minimális érték a 218 (fekete), míg a maximális érték a 654 (fehér) lesz. Ezt tekinti a robot 0-nak, illetve 100-nak. Az eredeti értékeket visszaállítani a *Default* gombra kattintva lehet.

A kalibrálás elvégezhető programból is. Így lehetőségünk van arra, hogy akár menet közben mért értéket állítsunk be minimálisnak vagy maximálisnak.

A modul ikonja az *Advanced* csoportban található.



A paraméterek jelentése:

A paraméter neve	A paraméter jelentése
Port	A szenzor csatlakozási portja.
Sensor	A kiválasztott szenzor. Kétféle lehet: fényszenzor (*Light Sensor*) vagy hangszenzor (*Sound Sensor*).
Action	Kétféle érték közül választhatunk. *Calibrate* esetén a kalibrálást végezhetjük el, míg *Delete* esetén visszaállíthatjuk az eredeti értékeket.
Value	Kiválaszthatjuk, hogy a minimális vagy maximális értéket szeretnénk-e beállítani. Egyszerre csak az egyiket lehet, tehát ha mindkettőt meg szeretnénk változtatni, akkor kétszer kell használnunk a modult.

A kalibrálás modulban nem lehet az értéket manuálisan, beírt értékkel megadni, csak paraméterátadással. Így a robot a programja során a környezetében aktuálisan mért értéket állíthatja be 0-nak vagy 100-nak.
A kalibrálás programon keresztüli használata során ügyelni kell arra, hogy megfelelően különböző érték legyen a minimum és a maximum. Ha a két érték megegyezik vagy nagyon kicsi az eltérés, akkor a robot szenzora mindig ugyanazt fogja visszaadni.
Természetesen a kalibrálással nem változik meg a szenzor érzékenysége, így a mérési pontosság nem lesz nagyobb, de alkalmas arra, hogy kizárjuk a zavaró hatásokat (a skála két szélsőértékén kívülieket). A mérési pontosság persze látszólag növekedhet, hiszen a szenzorok a 0 – 1023 tartományban mérnek, és ezt az értéket jelenítik meg közel egy nagyságrenddel kisebb számként. Így a mérési tartomány szűkítésével látszólag nő a mérési pontosság a megkülönböztethető értékeket tekintve. Pl. míg korábban a 345 – 354 között mért értékeket a robot 35-ként adta vissza, addig leszűkítve a mérési tartományt ezek között az értékek között is lesz számszerű különbség.

14.7. Motorok szinkronizálása

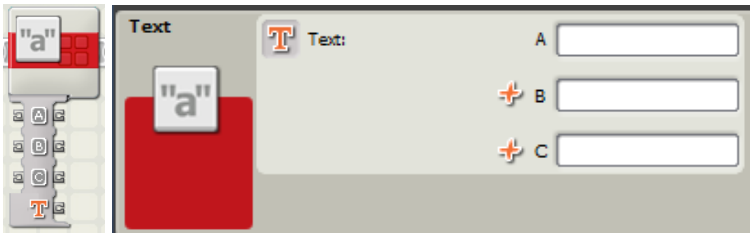
A robot szervomotorjai szoftveres, beépített szinkronizációt tartalmaznak. Ez azt jelenti, hogy ha két motort működtetünk egyszerre, és nincs beállítva kanyarodás (a nyomaték elosztása a motorok között azonos), akkor a vezérlés automatikusan szinkronizálja őket: a két motor elfordulási szögeit figyelve korrigálja az esetleges csúszásokból, tapadási egyenetlenségekből eredő eltéréseket. Így a robot jó közelítéssel egyenesen fog haladni. Ez a szinkronizáció automatikus, azonban tapasztalataink szerint nagyon alacsony sebességeknél nem működik (40 fölötti sebességnél már igen).
A motorok működésében egy másik szinkronizáció is szerepet kap. Abban az esetben, ha több *Move* ikont használunk egymás után, és az ikonok *Next Action* paraméterét *Coast*ra állítjuk (az utolsót kivéve), akkor a lassulva megállásból következően a *Move*-okkal vezérelt mozgás összességében időben elcsúszhat, hiszen a motor leállását követően a robot még lassulva gurul. Ezt korrigálja a másodlagos szinkronizáció

azzal, hogy az utolsó *Move* esetén (*Next Action* paraméter: *Break*), a beállított működési időtartamot lerövidíti, így a teljes mozgássor időtartama jól közelíti a *Move* ikonoknál beállított mozgási időtartamok összegét.

Amennyiben ezt nem szeretnénk, tehát a másodlagos szinkronizációt szeretnénk kikapcsolni, akkor ezt megtehetjük az *Advanced* csoport *Reset Motor* moduljával.
Paraméterként azoknak a motoroknak a csatalakozási portjait kell megadni, amelyeknél a szinkronizációt ki szeretnénk kapcsolni.

14.8. Szövegösszefűzés

Az *Advanced* csoport *Text* moduljával lehetőségünk van arra, hogy legfeljebb három szöveges típusú értéket egyetlen karakterlánccá fűzzünk össze.

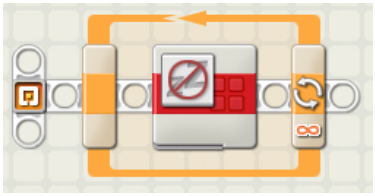


A szövegeket megadhatjuk a szövegdobozba írva, vagy paraméterként. Ha változókban tárolt szövegeket fűzünk össze, akkor a kapott karakterlánc részleges tartalma aszerint változhat, hogy mi volt a változó értéke. Így a program eseményeitől függően különböző tartalmú szövegek (üzenetek) írhatók például a robot képernyőjére.

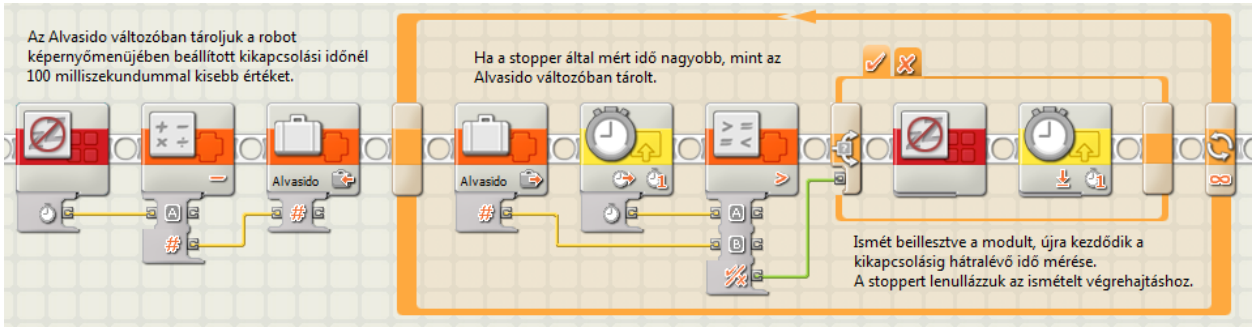
14.9. Kikapcsolási idő

A robot képernyőmenüjének *Settings/Sleep* funkciójával beállítható, hogy a robot energiatakarékossági okokból hány perc múlva kapcsoljon ki automatikusan. A kikapcsolás akkor is megtörténik, ha program fut a roboton. Mivel a megírt programjaink futási ideje viszonylag rövid (általában egy percen belüli), ezért ritkán okoz mindez problémát. Hosszabb futási idejű programok esetén azonban figyelni kell a beállításra (pl. hosszabb fizikai méréssorozat, amelyet nem kísérünk állandó figyelemmel, hanem a robot önállóan dolgozik). A robot képernyőmenüjében ugyan beállítható a *Never* érték, amely használata esetén nem kapcsol ki a robot, de így az akkumulátor merül fölöslegesen a programozási holtidőben. A képernyőmenüben beállított értéket lekérdezhetjük programból az *Advanced* csoport *Keep Alive* moduljával. A modul csak olvasható, és milliszekundumban adja vissza a beállított értéket.
Ha a program egy adott helyére beillesztjük az ikont, akkor végrehajtása során onnantól kezdi mérni a képernyőmenüben beállított kikapcsolásig hátralévő időt. Például a képernyőmenü *Settings/Sleep* funkciójánál 2 percet állítottunk be, és a megírt programunkba egy percnyi várakozás után (*Wait*) illesztjük be a *Keep Alive* modult, akkor a robot 3 perc után fog kikapcsolni.

Ez lehetőséget teremt arra, hogy a képernyőmenüben beállított értéktől függetlenül garantálni tudjuk, hogy programjaink futása során ne kapcsoljon ki a robot. A modult egy végtelen ciklusba helyezve, és egy külön programszálon futtatva, a robot a program futása alatt nem fog kikapcsolni, hiszen folyamatosan újrakezdődik a kikapcsolásig hátralévő idő mérése.



Elegánsabb megoldás, ha a modult használva először lekérdezzük a beállított értéket, majd stopperrel mérjük a programindítás óta eltelt időt, és ha megközelítjük a kikapcsolási időként beállított értéket, csak akkor használjuk a *Keep Alive* modult. Egy lehetséges programkódot mutat az alábbi ábra.



Az Alvasido változóban tároltuk a képernyőmenüben beállított értéknél 100 milliszekundummal kisebb értéket. Ehhez fogjuk egy ciklusban folyamatosan hasonlítani a stopperrel mért időt. Ha a mért idő nagyobb lesz, mint a változóban tárolt, akkor ismét beillesztjük a *Keep Alive* modult, tehát újrakezdjük a kikapcsolásig hátralévő idő mérését. Ha a stoppert lenullázzuk, és végtelen ciklust használunk, akkor a programunk nem fog futási időben leállni a képernyőmenüben beállított készenléti idő letelte miatt. A programkódot külön programszálra helyezve, bármely programban használható.

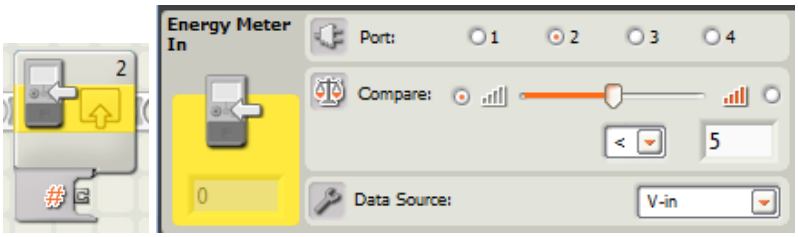
14.10. Speciális szenzorok használata

A könyv *1.4. Input eszközök: egyéb szenzorok* fejezetében több olyan hardver eszközről is írtunk, amelyek kiegészítőként csatlakoztathatók a téglához. Ezek teljes körű bemutatása nem volt célunk. Használatukat a „Megújuló energiák” (*Renewable Energy*) készletben található szolár panel, és szélturbina eszközökön keresztül szemléltetjük.

Valamennyi kiegészítő eszközhöz a hardveren kívül szükséges a vezérlésüket lehetővé tévő programmodulok telepítése. A modulok letölthetők az internetről. A telepítésüket lásd a 2. fejezetben.

A megújuló energiák készlet tartalmaz egy szolár panelt, egy megépíthető szélturbinát, egy elektrométert, amely önálló mérések elvégzését is lehetővé teszi, valamint egy energiatároló egységet. Ugyanakkor az eszközök csatlakoztathatók az NXT téglához, így programból is lekérdezhetők a mért adatok, valamint felhasználhatók programjaink vezérlésére.

A programozáshoz szükséges az *Emeter In* modul telepítése. A telepítés során adható meg, hogy melyik programcsoportba kerüljön az ikonja.

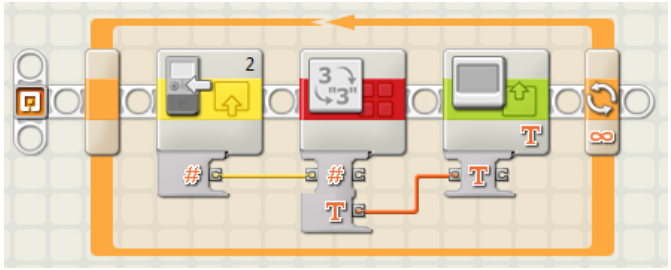


A modul segítségével az energiaforrás négy paraméterét kérdezhetjük le a *Data Source* beállításának megfelelően: feszültség (*V-in*), áramerősség (*A-in*), teljesítmény (*W-in*), és megtermelt energia (*J-in*). Természetesen ez utóbbi nem a pillanatnyi értéket jelenti, hanem a készletben található energiatároló egység töltöttségét, amely a megújuló energiaforrásokhoz csatlakoztatva, a megtermelt energia tárolását teszi lehetővé.

A szolár panel esetében, egy 60 W-os izzótól 20 cm távolságban mért adatokat mutatja a következő ábra.

Az energiatároló ebben az esetben még üres volt.

Az értékek képernyőre íratása a korábban már bemutatott paraméterátadás technikájával valósítható meg.



Az eszköz alkalmas arra is, hogy a tégla output portjaira kiadott fizikai jellemzőket mérje. Az alábbi ábra az A porton mért feszültség-, áramerősség-, és teljesítményadatokat mutatja.

A korábban bemutatott programozástechnikai eszközök és a szolár panel segítségével egy gyakorlati probléma megoldása is lehetővé vált. A napelemek egyik problémája, hogy az optimális energiatermeléshez, követniük kell a nap járását. Ehhez egy olyan mechanikai rendszer megépítése

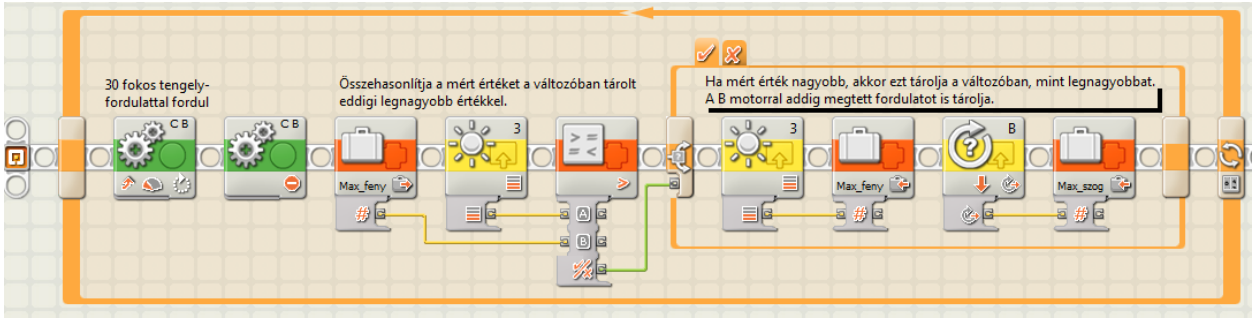
szükséges, amely folyamatosan és automatikusan a nap felé fordítja őket. Ennek a modellezését szemlélteti a következő példa.

14/P4. Írjon programot, amelyet végrehajtva a robot fényszenzora segítségével meghatározza azt az irányt, amely felől a legnagyobb fényintenzitás mérhető! A mérést egy kör kerülete mentén mozgatott fényszenzor segítségével végezze!

A szolár panel nap felé fordítását úgy is elvégezhetnénk, hogy a nap járását beprogramozzuk a robotnak, és az bizonyos időközönként fordul a megfelelő irányba.

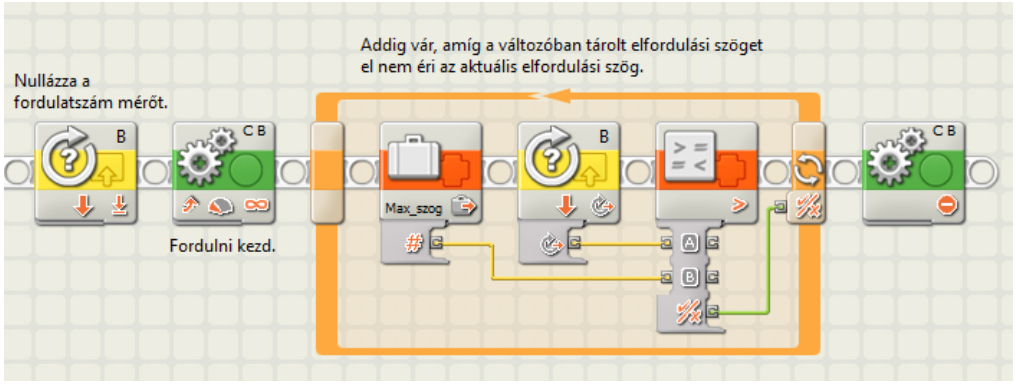
Ehelyett azt a módszert választottuk, hogy egy teljes körbefordulás során, bizonyos időnként mintát veszünk a fényszenzorral a környezetben mérhető fényintenzitásból. A legnagyobb mért értékhez tartozó elfordulási szöget megjegyezzük, és miután a teljes kör mentén végeztünk a méréssel, a robotot az adott pozícióba forgatjuk. Ehhez a robot szerkezetét módosítani kell, és a fényszenzort előre néző pozícióban felszerelni.

A megoldást két részletben mutatjuk be. Az első részben történik a mérés és a legnagyobb fényintenzitáshoz tartozó elfordulási szög meghatározása. Két szám típusú változót hoztunk létre. A Max_feny változóban tároljuk az aktuálisan legnagyobb mért értéket, míg a Max_szog változóban a hozzá tartozó motorelfordulási szöget. A mintavételezést 30 fokonként végezzük (a motor tengelyének elfordulási szöge). A program indítása előtt méréssel meghatároztuk, hogy az adott robotkonstrukció esetén a teljes kör kb. 750 fokos elfordulást jelent. Így 25-ször lefutó növekményes ciklust használtunk (30x25 = 750). A cikluson belül a Max_feny változó tartalmát (kezdetben 0) összehasonlítjuk az aktuálisan mért fényintenzitással (a fényszenzor lámpáját kikapcsoltuk, *Generate light*). Amennyiben az aktuálisan mért érték nagyobb, mint a változóban tárolt, akkor a Max_feny változóba betesszük az új értéket (ezzel a régi, fölösleges értéket felülírtuk), és a Max_szog változóban eltároljuk a B motor tengelyének indulási pozíciótól megtett elfordulási szögét.



A ciklus lefutása után a Max_szog változó tartalmazza a kiindulási pozíciótól szükséges tengelyelfordulási szög mértékét. Ügyelni kell arra, hogy a mérésnél, és a robot pozícióba állításánál ugyanazt a sebességet használjuk. A B motor elfordulási szögét mérő modul 750°-nál nagyobb értéket tartalmaz a kiindulási pozícióba visszaérkezve, így először nullázzuk (*Reset*) azt. A pozícióba állítást egy *Unlimitedre* állított *Move* modullal végezzük. A modul utáni ciklus mindaddig fut, amíg a B

motor elfordulási szöge nagyobb nem lesz, mint a Max_szog változóban eltárolt érték. A bemutatott algoritmusötlet a programozás szélsőérték keresés programozási tételével ekvivalens.

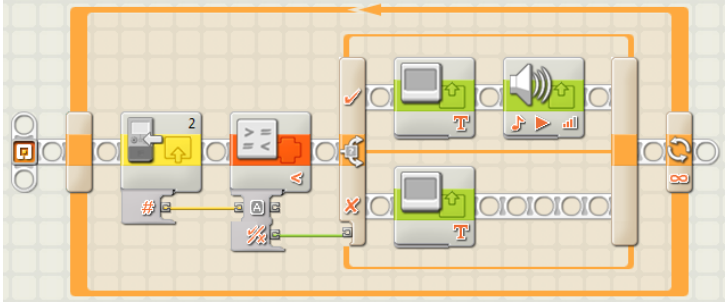


A bemutatott program alkalmas arra, hogy megkeresse a legnagyobb fényintenzitás értéket adó irányt. Egyetlen kérdés maradt nyitva. Automatizálni kellene, hogy mikor induljon el a forgatási algoritmus. Ennek egy lehetséges vezérlésére a szélturbina esetében mutatunk be egy példát.

14/P5. Írjon programot, amelyet végrehajtva a robot kiírja a képernyőjére, hogy adott irányból fúj-e a szél vagy sem, és szélcsend esetén hangjelzést is adjon!

A bemutatott *Emeter-in* modul alkalmas arra, hogy visszaadja az energiatermelő egység által pillanatnyilag szolgáltatott feszültség értéket. A szélkerék által forgatott turbina (dinamó) által termelt elektromosság arányos a turbina forgási sebességével, és ezen keresztül a szélerősséggel. Természetesen sok más tényező is befolyásolja mindezt, például a szélkerék lapátjainak dőlésszöge. Egy adott szerkezeti konstrukció esetén azonban ezek a hatások jó közelítéssel konstansnak tekinthetők, így az arányosság fennáll. A programozási ötlet a szélturbina esetén az, hogy ha a termelt elektromos feszültség értéke egy adott szint alá csökken, akkor a szél erősségének kellett csökkennie, vagy irányának megváltoznia. (Az irány megváltozását az előbbi példánál használt algoritmus segítségével tudjuk ellenőrizni, a fényintenzitás helyett az adott irányban mérhető megtermelt elektromos feszültség értékét felhasználva).

A programban 1-re állítottuk azt a határt (1 Volt aktuálisan termelt feszültség), amely alapján megkülönböztetjük a szélcsendet a szeles állapottól. Egy végtelen ciklusban az elektrométer modullal mért feszültségértéket hasonlítjuk az 1-hez. Ha a termelt feszültség ennél kisebb, akkor a képernyőre írjuk a *„Szelcsend”* szót és hangjelzést adunk, míg ellenkező esetben kiírjuk a *„Fuj a szel”* szöveget.



A bemutatott példák csak ízelítőt adtak a MINDSTORMS® NXT robothoz beszerezhető hardver eszközök és programozási modulok használatáról. Az *1. fejezetben* ismertetett hardver elemek további sok új, és kreatív felhasználási lehetőséget biztosítanak.

14.11. Gyakorló feladatok

14/F1. Írjon programot, amely segítségével a téglán található nyomógombokkal állíthatjuk be a robot mozgását! Hozzon létre négy változót, és a téglán lévő nyomógombok segítségével a változók tartalmát állítsa be rendre igaz vagy hamis értékre! A robot 1 másodperces előre mozgás után, aszerint forduljon kb. 90º-ot balra vagy jobbra, hogy az első változó értéke igaz vagy hamis. Ezután ismét 1 másodperc előremozgás után a második változóban tárolt érték határozza meg a fordulás irányát, és így tovább. A robot a mozgását a téglán lévő enter gomb megnyomására kezdje!

14/F2. 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/F3. 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 zeneszámrészletet!

14/F4. Készítsen egy saját blokkot (modult), amelyet a programba illesztve a robot egyetlen fényszenzorával követ fehér alapon egy fekete csíkot! A blokkot úgy paraméterezze, hogy a motorok sebességét és a fekete-fehér színek határértékét változón keresztül lehessen beállítani!

14/F5. Készítsen egy saját blokkot (modult), amelyet a programba illesztve a robot változókon keresztül kap három számértéket és a képernyőre írja közülük a legnagyobbat!

14/F6. Írjon programot, amelyet végrehajtva a robot megméri egy előtte elhelyezkedő akadálytól a távolságát és fájlba írja! A mérőszám és a mértékegység is a fájl ugyanazon sorába kerüljön, egy szóközzel elválasztva. A mértékegységet az ultrahangszenzor beállításai alapján határozza meg a program!

15. VEGYES FELADATOK

15/F1. Írjon programot, amelyet végrehajtva a robot egy az alapszíntől jól elkülöníthető csíksor fölött halad és megáll a megadott számú csík után! (A csíkok párhuzamosak, de távolságuk és szélességük nem féltetlenül azonos.)

Pl.:

15/F2. Írjon programot, amelyet végrehajtva a robot egyenesen halad előre mindaddig, amíg a fényérzékelőjére rá nem világítunk, ekkor forduljon kb. 180°-ot! Mindezt ismételje kikapcsolásig!

15/F3. Írjon programot, amelyet végrehajtva a robot ívben fordul 1 mp-ig balra, majd ívben fordul jobbra 1 mp-ig, mindezt 5-ször hajtsa végre! A program megírása során egyetlen *Move* ikont használjon!

15/F4. Írjon programot, amelyet végrehajtva a robot egy ultrahang szenzorral képes egy adott távolságon belül lévő akadály észlelésére és követésére. A robot egyenletes sebességgel forog és ha egy adott távolságon belül észlel valamit (az ultrahangszenzora jelzi), akkor elindul felé mindaddig, amíg az adott távolságon belül van az észlelt akadály. Ha elveszítette (kikerült az akadály a távolságon kívülre), akkor újra forog. Mindezt kikapcsolásig ismétli.

15/F5. Írjon programot, amelyet végrehajtva a robot egy fényérzékelővel képes egy lámpa fényének észlelésére és követésére! A robot egyenletes sebességgel forog, és ha egy lámpa fényét észleli, akkor elindul felé. Ha elveszítette, akkor újra forog. Mindezt kikapcsolásig ismétli.

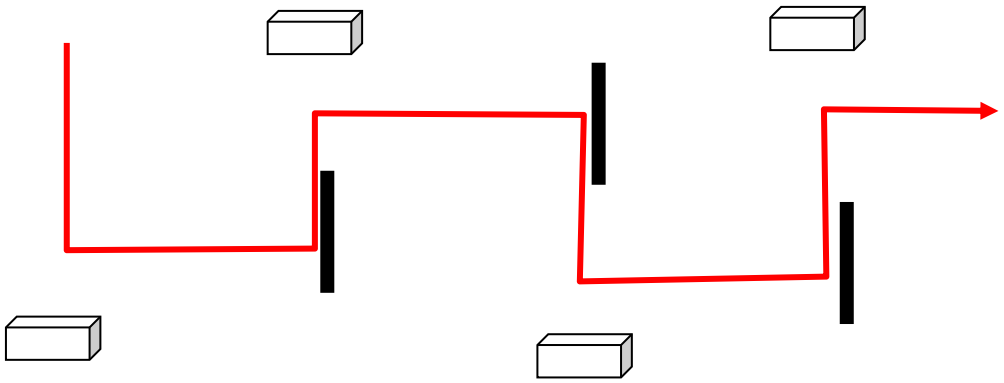
15/F6. Írjon programot, amelyet végrehajtva a robot véletlenszerűen sorsol 1 és 5 közötti számot! A kisorsolt számot kiírja az LCD képernyőjére, és ki is mondja angolul vagy magyarul!

15/F7. Írjon programot, amelyet végrehajtva a robot az alap színétől jól megkülönböztethető és különböző szélességű csíksoron egyenletes sebességgel halad keresztül. Minden csík után írja képernyőre a csík szélességét milliszekundumban (a csík fölötti áthaladás időtartama).

15/F8. Írjon programot, amelyet végrehajtva a robot egy az alaptól jól elkülönülő színű csíksor fölött halad 5 másodpercen keresztül! Öt másodperc múlva megáll, és képernyőjére írja a csíkok számát, amelyek fölött teljes egészében áthaladt.

15/F9. Írjon programot, amelyet végrehajtva a robot az ultrahang szenzorát és fényérzékelőjét használva az ábrán látható akadálypályán a jelölt útvonalon halad végig!

Pl.:



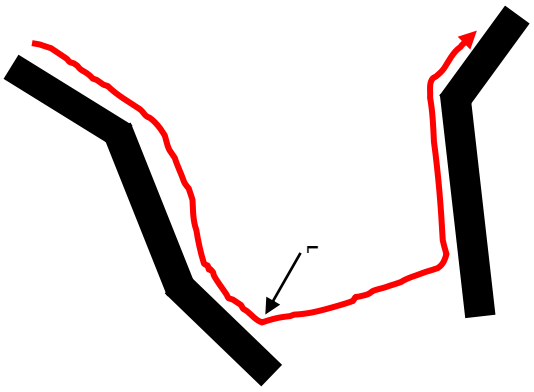
15/F10. Írjon programot, amelyet végrehajtva a robot egy spirál alakú pályán mozog 30 másodpercig!

15/F11. Írjon programot, amelyet végrehajtva a robot váltakozva egy fekete-fehér színű vonalat követ egyetlen fényszenzorával! Az alap színe jól megkülönböztethetően különbözik a feketétől és fehértől is.

Pl.:

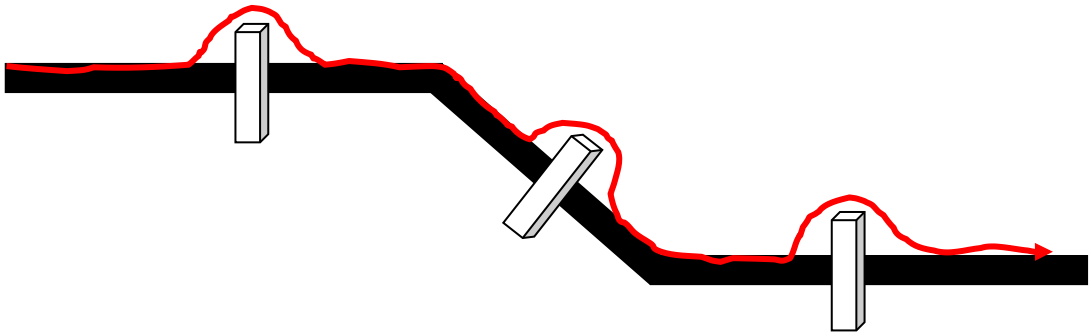
15/F12. Írjon programot, amelyet végrehajtva a robot egyetlen fényszenzorával 5 másodpercig követ egy az alaptól jól megkülönböztethető színű csíkot, majd adott pozícióba mozog, ahol ismét követni kezdi a csíkot! Lásd az ábrát!

Pl.:



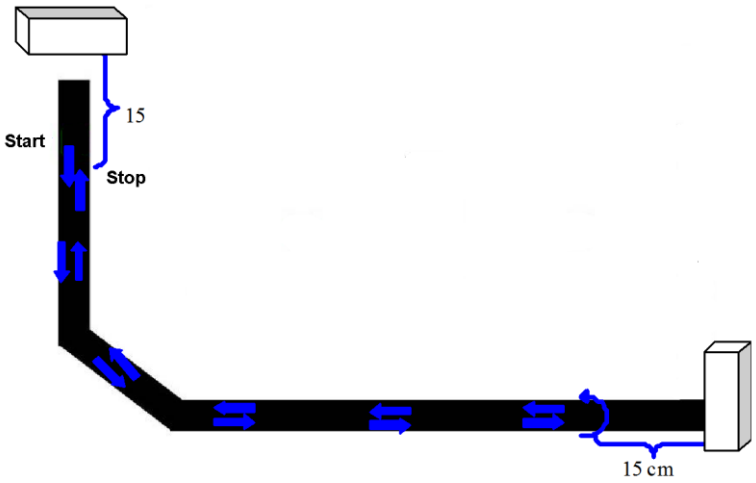
15/F13. Írjon programot, amelyet végrehajtva a robot egyetlen fényszenzorával követi az alaptól jól megkülönböztethető színű csíkot (nem feltétlenül egyenes), amelyen az ultrahang szenzorral észlelhető akadályok vannak. Egy ilyen akadályhoz érve a robotnak – valamelyik oldaláról megkerülve az akadályt – a csíkra visszatérve kell folytatnia az útvonalkövetést. Lásd ábra!

Pl.:



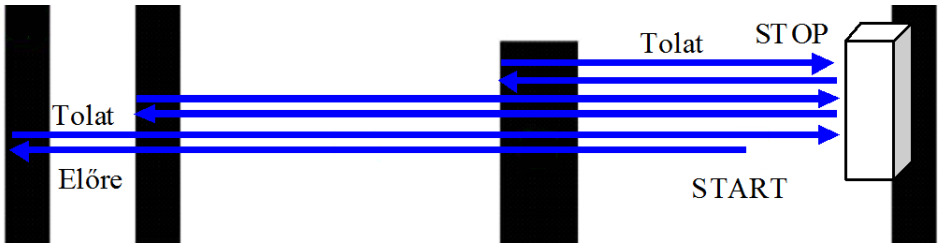
15/F14. Írjon programot, amelyet végrehajtva a robot startpozícióból indul előre, és egyetlen fényszenzorával követi a fekete színű vonalat! Ha ultrahang szenzorával 15 cm-en belül akadályt észlel, akkor megfordul, és az ellenkező irányban követi tovább mindaddig, amíg ultrahang szenzora 15 cm-en belül akadályt nem észlel. Ekkor megáll.

Pl.:



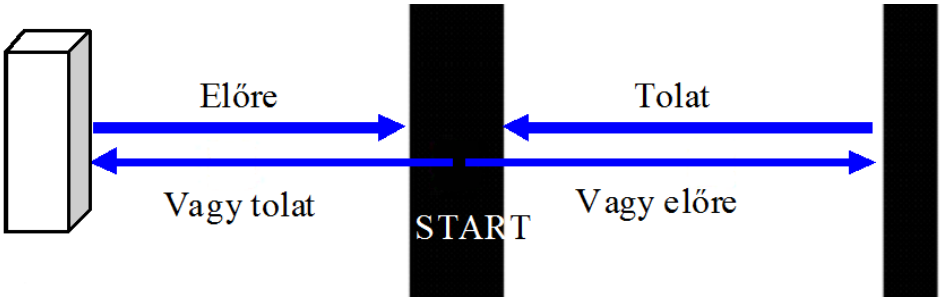
16/F15. Írjon programot, amelyet végrehajtva a robot elindul egyenesen előre a haladási irányára merőleges, különböző vastagságú fekete vonalak fölött! A harmadik csík fölötti áthaladás után elkezd tolatni, az ütközésérzékelő benyomásáig. Ezután újra előre indul, majd a második csík után ismét tolatni 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 (1. hang: 440 Hz - zenei A, 200 ms időtartamig, 2. hang: 528 Hz – zenei C, 200 ms időtartamig).

Pl.:



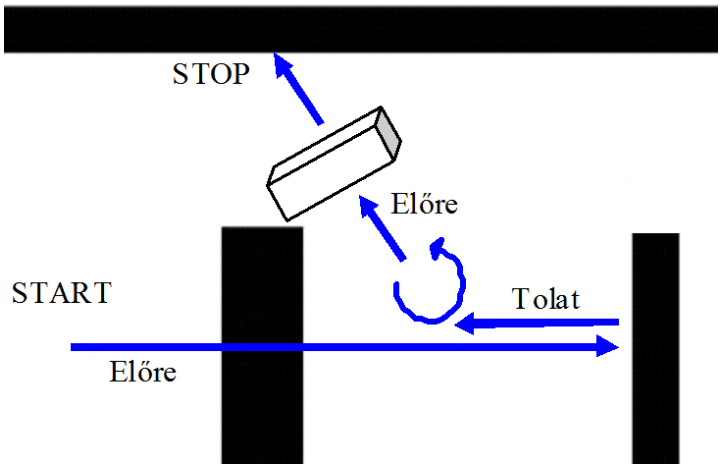
16/F16. Írjon programot, amelyet végrehajtva a robot véletlenszerűen sorsol nullát vagy egyet! Ha egyet sorsolt, akkor sípol egyet (hang: zenei A – 440 Hz, időtartam: 0,5 mp), majd ütközésig tolat és visszatér a kiindulási fekete vonalig. Ha nullát sorsolt, akkor elmegy a fekete csíkig, majd visszatér a fekete kiindulási vonalig (ne sípoljon). Mindezt hajtsa végre összesen 4-szer!

Pl.:



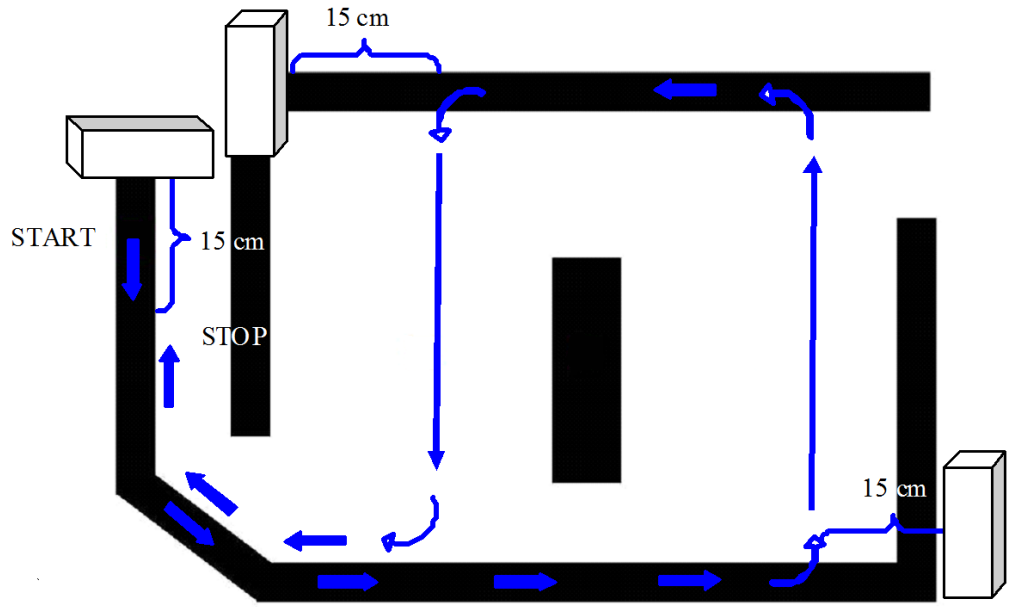
15/F17. Írjon programot, amelyet végrehajtva a robot startpozícióból indul, és a második fekete csíknál megáll! Tolat körülbelül a két csík közötti távolság közepéig, majd helyben forogni kezd mindaddig, amíg az ultrahang szenzora akadályt nem érzékel 20 cm-es távolságon belül. Ekkor sípol egyet (Hang: zenei A – 440 Hz, időtartam: 0,5 mp), elindul a doboz felé, és a fekete csík eléréséig tolja a dobozt (elegendő, ha hozzáér a dobozhoz). A tolatásnál a két csík közötti távolság közepét ne konstans beállításával, hanem valamilyen algoritmussal határozza meg a robot!

Pl.:



15/F18. Írjon programot, amelyet végrehajtva a robot elindul, és egyetlen fényszenzorával követi a fekete vonalat mindaddig, amíg ultrahang szenzora 15 cm-en belül akadályt nem érzékel! Ekkor balra fordulva áttér a másik fekete vonalra. Ezt követi mindaddig, amíg az ultrahang szenzora 15 cm-en belül akadályt nem érzékel. Ekkor balra fordul, és áttér a következő fekete vonalra. Ezt követi mindaddig, amíg ultrahag szenzora 15 cm-en belül akadályt nem érzékel. Ekkor megáll. A haladási irányok értelmezéséhez lásd a nyilak által jelzett útvonalat!

Pl.:



AJÁNLOTT IRODALOM

B. Erwin: Creative Projects with LEGO Mindstorms, Addison-Wesley Professional (USA), 2001

D. Astolfo, M. Ferrari, G. Ferrari: Bulding Robots with LEGO Mindstorms NXT, Syngress Publishing, Inc. (USA), 2007

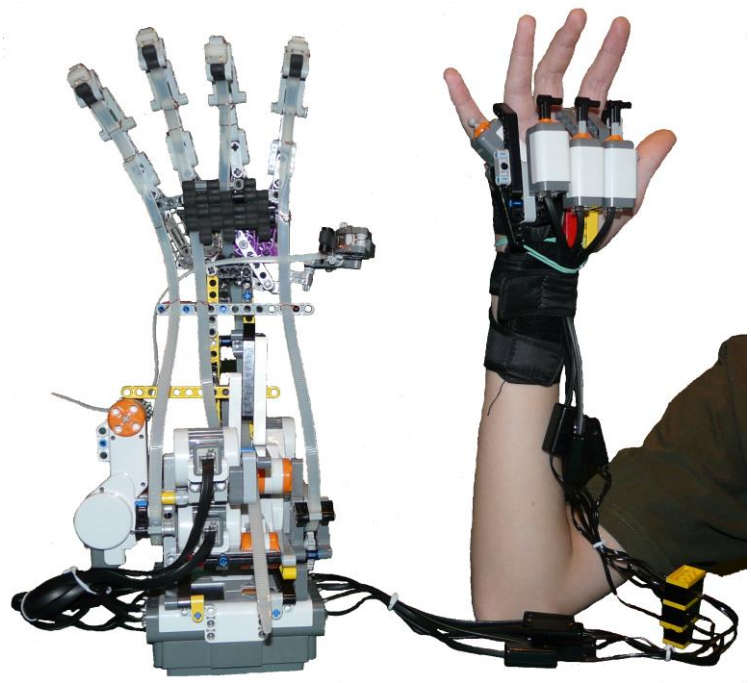
D. Astolfo, S. Cavers, K. Clague, Cs. Soh, L. Whitman, T. Witherspoon: LEGO Mindstorms Ultimate Builder Projects, Syngress Publishing, Inc. (USA), 2002

J. Floyd Kelly: Mindstorms NXT-G Programming Guide, Apress, Inc. (USA), 2007, 2009

Kiss Róbert, Pásztor Attila: Mobil robotok programozása NXC és NXT-G nyelven, Kecskeméti Főiskola, 2009

M. Gasperi, P. „Philo" Hurbain, I. Hurbain: Extreme NXT, Extended the LEGO Mindstorms NXT to the Next Level, Apress, Inc. (USA), 2007

Seymour A. Papert: Mindstorms: Children, Computers, And Powerful Ideas, Basic Books (USA), 1993



Ütközésérzékelőkkel vezérelt robotkéz. Képes könnyű tárgyak megfogására.