Kiss Róbert

ROBOTIKA FELADATGYŰJTEMÉNY

111 feladat LEGO® MINDSTORMS® EV3 és NXT robotokhoz

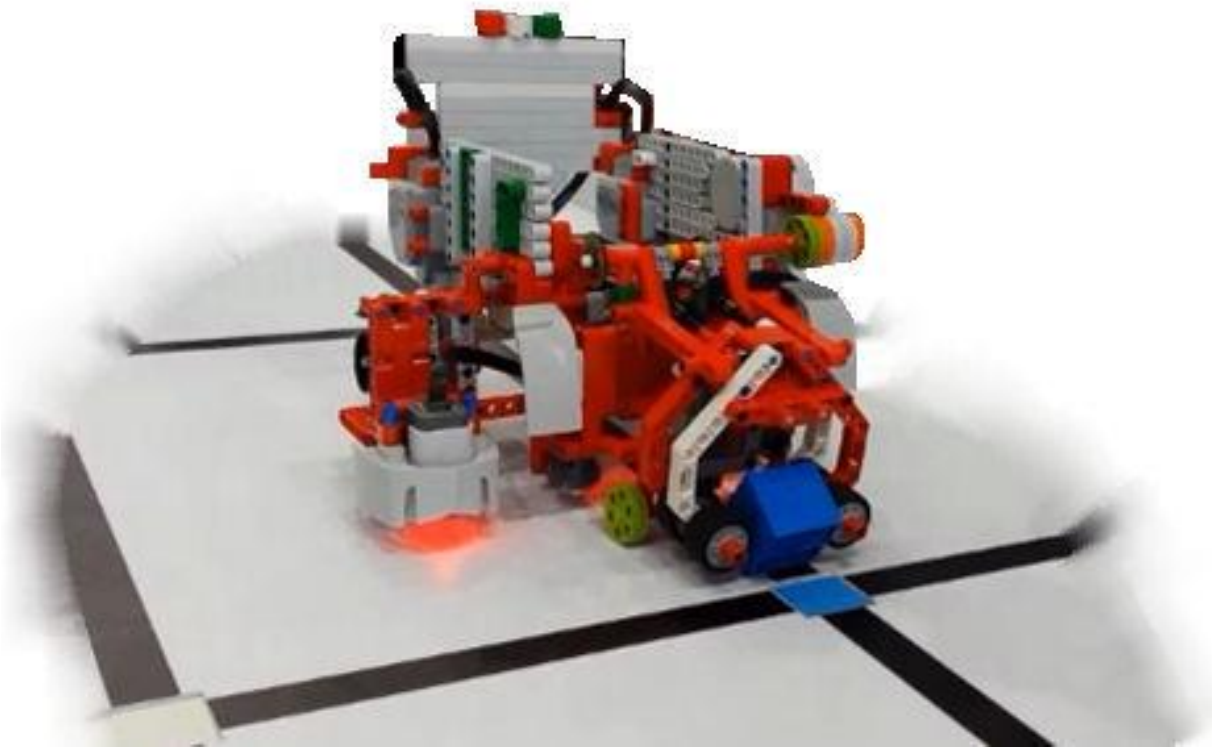
A könyv elektronikus változatának kiadása a H-Didakt Kft. jóvoltából jöhetett létre

2016

H-Didakt Kft
1162 Budapest, Délceg u. 32.
http://www.hdidakt.hu

Szerző: Kiss Róbert

Kiadás éve: 2016



A FELADATGYŰJTEMÉNY FELÉPÍTÉSE ÉS HASZNÁLATA

A feladatgyűjtemény elméleti hátterét a http://www.hdidakt.hu/adat/dw_anyagok/dw_74.pdf címről letölthető elektronikus könyv adja (*A MINDSTORMS EV3 robotok programozásának alapjai*).

A feladatgyűjtemény alapvetően a MINDSTORMS robotok programozását helyezi a középpontba, ezért a robotépítés csak ritkán jelenik meg. A feladatok megoldhatók NXT és EV3 típusú robotokkal is, de a megoldások forráskódjai és a magyarázatok az EV3-G szoftverkörnyezetben készültek. Mindkét típusú robothoz a melléklet tartalmaz egy-egy olyan építési útmutatót, amely az alaprobotot képezi a könyv feladataihoz. Célszerű ezeket vagy ezekhez hasonló robotokat építeni kezdésként.

Törekedtünk arra, hogy a robotprogramozásban különböző szinten tartók is találjanak a feladatgyűjteményben kihívást jelentő feladatokat. Ezért három fő fejezetre tagoltuk a kötetben szereplő feladatokat, és minden feladatot egy ötfokú skálán szinteztünk. A szintek a feladat bonyolultságától, a megoldáshoz szükséges ötletek kreativitásától, a használandó programozási eszközök összetettségétől függően nőnek. A feladatok melletti színes ikonok jelzik a szinteket

1. szint (kezdő): 
2. szint (közép haladó): 
3. szint (haladó): 
4. szint (profi): 
5. szint (versenyző): 

Az első fejezet a programozást most kezdőknek szól. Hat projekttel kezdődik a könyv, amely egy meseszerű történetbe építve ismertet meg az alapokkal. Minden projekt esetén szerepel a becsült időkeret, valamint az elméleti könyv vonatkozó fejezeteire történő hivatkozás. A 7. alfejezet vegyesen tartalmaz egyszerű feladatokat.

A második fejezet már a haladóknak szól. Itt feltételezzük, hogy az elméleti könyv első kilenc fejezetében szereplő programozási elemeket a feladatmegoldók alapszinten ismerik.

A harmadik fejezet versenyszintű feladatokat tartalmaz, amelyek egy része már kitűzésre került különböző magyarországi programozói versenyeken.

Minden feladat esetén, az alfejezetek végén található programozási ötletek címszó alatt olyan, a feladatokhoz kapcsolódó algoritmizálási példák olvashatók, amelyek segítenek a megoldásban. Az első fejezetben még részletesebb útmutatások, de később csak a lényeges, újszerű ötletek bemutatása szerepel.

Valamennyi feladat megoldásának forráskódja letölthető a http://www.hdidakt.hu címről.

Előkészületben van egy videó tár elkészítése, amely a feladatspecifikáció megértését segíti, és letölthető videón is megtekinthető a robot működése. Mindez segíti a kisebbek számára a feladat megértését.

A kötetben több, mint 100 programozási feladat szerepel.

Az egyes feladatokhoz készítendő tesztpályák sematikus rajzai szintén bekerültek a kötetbe. Ezek elkészítéséhez kartonpapírt és szigetelő szalagot javaslunk, amelyek elég variábilisak ahhoz, hogy újabb ötletek nyomán könnyen átalakíthatók legyenek.

Jó programozást!

TARTALOMJEGYZÉK

A FELADATOK SZINTENKÉNTI LISTÁJA

A feladatgyűjtemény feladatai nem feltétlenül nehézségi sorrendben szerepelnek. A használatot könnyíti meg a feladatok szintenkénti listája.

1. szint:

1.1.1.
1.2.1.
1.2.2.
1.2.3.
1.3.1.
1.3.2.
1.3.4.
1.4.1.
1.5.1.
1.6.1.
1.6.2.
1.6.3.
1.6.4.
1.7.1.a
1.7.1.b
1.7.1.c
1.7.2.a
1.7.2.b
1.7.2.c
1.7.3.b
1.7.3.c
1.7.4.a
1.7.4.d
1.7.6.a
1.7.7.a
1.7.7.c
1.7.8.a
1.8.1.
1.8.4.
2.2.3.a

2. szint:

1.2.4.
1.2.5.
1.4.3.
1.4.4.
1.5.2.
1.6.5.
1.6.6.
1.6.7.
1.7.1.d
1.7.3.a
1.7.4.c
1.7.4.e
1.7.5.a
1.7.5.b
1.7.5.e
1.7.5.h
1.7.6.b
1.7.6.c
1.7.7.b
1.7.7.d
1.7.7.e
1.7.8.b
1.7.9.a
1.8.2.
1.8.3.
1.8.5.
2.2.1.
2.3.1.
2.3.3.
2.3.4.
3.3.3.a

3. szint:

1.3.3.
1.4.2.
1.4.5.
1.4.6.
1.7.4.b
1.7.5.b
1.7.5.c
1.7.5.f
1.7.6.d
1.7.6.e
1.7.7.f
1.7.7.g
1.7.7.h
1.7.9.b
1.7.9.c
1.7.10
2.1.1.a
2.1.1.b
2.1.2.
2.1.3.
2.1.4.
2.2.3.b
2.3.2.
2.3.5.a
2.3.10.
2.3.11.
3.1.2.a
3.1.2.b
3.1.3.b
3.1.4.b
3.2.5.
3.3.1.a
3.3.2.a
3.3.2.b

4. szint:

1.7.5.g
2.1.1.c
2.2.2.
2.3.5.b
2.3.7.
2.3.9.
3.1.1.
3.2.1.
3.2.2.
3.2.6.
3.3.3.b

5. szint:

2.3.6.
2.3.8.
3.1.3.a
3.1.4.a
3.2.3.a
3.2.3.b
3.2.4.
3.3.1.b
3.3.1.c
3.3.3.c

1. BEVEZETŐ PROJEKTEK A ROBOTPROGRAMOZÁST MOST KEZDŐKNEK

1.1. PROJEKT – ROBOTÉPÍTÉS

Időigény

1-2 óra

Elméleti anyag

A projekthez a MINDSTORMS® robotok hardver elemeinek ismerete szükséges. Ehhez leírás a bevezetőben megadott elektronikus könyv *1. Robothardver* fejezetében szerepel. Az építés előtt ajánlott áttanulmányozni a *1.2. Az „intelligens" tégla és egyéb hardver elemek* alfejezettől a *1.5. Output eszközök: szervomotorok* alfejezetekig terjedő részt (8-13. oldal).

F.1.1.1. Feladat – Alaprobot építése

Szint: 

A feladatgyűjtemény bevezető feladata egy olyan egységes szerkezetű robot megépítése, amely alkalmas a további programozási feladatok megoldására. A mellékletben szereplő építési útmutatók alapján érdemes a robotokat megépíteni. Az *1. mellékletben* az EV3 típusú alaprobot építési útmutatója található, a *2. mellékletben* az NXT típusú roboté.

Az építési útmutatók egyszerű lépéseken keresztül vezetnek el a kész konstrukciókig. Az építőelemek ábrái nem méretarányosak, de az útmutató tartalmazza a méretekre vonatkozó utalásokat.

Nem feltétlenül szükséges a minta szerinti robot megépítése, ettől eltérő konstrukció is létrehozható. A szenzorok elhelyezése és a csatlakozási portok kiosztását nem célszerű megváltoztatni, mert a feladatok megoldásinak forráskódjai ezek alapján készültek.

Az első feladat tehát, hogy építsd meg az alaprobotot!

1.2. PROJEKT – „KÖRHINTA”

Időigény

2-3 óra (1 óra építés + 1-2 óra programozás)

Elméleti anyag

A projekt programozási részéhez az alapszenzorok használata és a vezérlési szerkezetek alapismerete szükséges. Ehhez leírás a bevezetőben megadott elektronikus könyv *4. Egyszerű mozgások*, *5. Szenzorok használata* és *6. Vezérlési szerkezetek* fejezeteiben szerepel. A programozás elkezdése előtt ajánlott áttanulmányozni a *4.1. Motorok vezérlése*, *5.3. Ütközésérzékelő*, *6.1. Ciklusok* és *6.2. Elágazások* fejezeteket.

F.1.2.1. Feladat – Körhinta építése

Szint: 

Készíts körhintát, amelyet egy MINDSTORMS NXT/EV3 robot működtet!

Biztosan láttál már majálison, vásárokban, játszótereken olyan körhintát, amelyen ülések sorakoztak egy kör mentén és motor forgatta őket körbe-körbe. A feladatod, hogy készítsd el egy ilyen körhinta működő modelljét.

A LEGO motor tengelyére kell a forgó hintát felépítened. A motort csatlakoztathatod az alaprobot szabad motor portjára (NXT robot esetén pl.: A-port; EV3 robot esetén pl.: A vagy D port).

Néhány valós hinta-minta:



További alkalmazási példák:

Repülőgép pilóták centrifugája

Centrifuga (mosógép)

F.1.2.2. Feladat – Körhinta program (egyszerű)

Szint: 

Írj olyan programot, amelyet a robotra feltöltve működteti a körhintát!

A bekapcsoláskor a körhinta motorja lassan forogjon és forgassa a rá szerelt szerkezetet is. (A lassú forgás lehet 20-as erősségű.)

Az ütközésérzékelő egyszeri benyomására a forgási sebesség növekedjen kétszeresére. (Legyen például 40-es.)

Az ütközésérzékelő újabb egyszeri benyomására a forgási sebesség ismét növekedjen. (Legyen a kezdeti háromszorosa, 60-as sebesség.)

Az ütközésérzékelő újabb benyomására álljon meg a hinta.

F.1.2.3. Feladat – Körhinta program (kétféle forgási sebesség ütközésérzékelő hatására)

Szint: 

Írj olyan programot, amelyet a robotra feltöltve működteti a körhintát!

Az 1-es portra kötött ütközésérzékelő benyomva tartása mellett a körhinta forogjon gyorsabban (pl.: 40-as sebességgel), az ütközésérzékelő felengedett állapotában forogjon a hinta lassabban (pl.: 20-as sebességgel).

A körhinta a program leállításával álljon meg!

F.1.2.4. Feladat – Körhinta program (forgásirány változtatása ütközésérzékelő hatására)

Szint: 

Írj olyan programot, amelyet a robotra feltöltve működteti a körhintát!

Használj két ütközésérzékelőt! Az egyiket az 1-es, a másikat a 2-es portra csatlakoztasd! Az 1-es portra kötött ütközésérzékelő benyomva tartása mellett a körhinta forogjon gyorsabban (pl.: 40-es sebességgel), az ütközésérzékelő felengedett állapotában forogjon a hinta lassabban (pl.: 20-as sebességgel).

A 2-es portra kapcsolt ütközésérzékelő benyomása után a körhinta az ellenkező irányba forogjon! (Minden gombnyomásra váltson forgásirányt!)

F.1.2.5. Feladat – Körhinta program (forgási sebesség növelése egyenletesen)

Szint: 

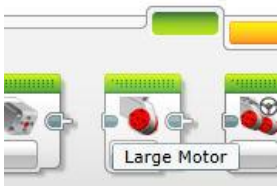
Írj olyan programot, amelyet a robotra feltöltve működteti a körhintát!

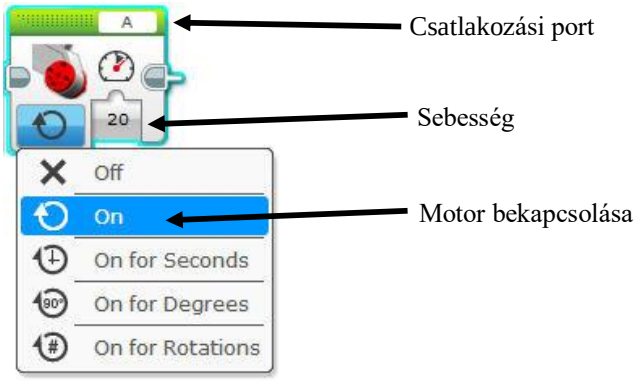
Az 1-es portra csatlakoztatott ütközésérzékelő minden benyomására a hinta forgási sebessége 10-zel növekedjen. Ha elérte a 100-as fordulatot, akkor álljon meg!

Programozási ötletek

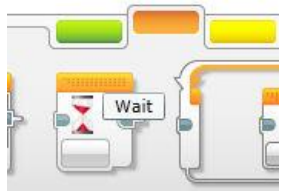
F.1.2.2. Körhinta program (egyszerű)

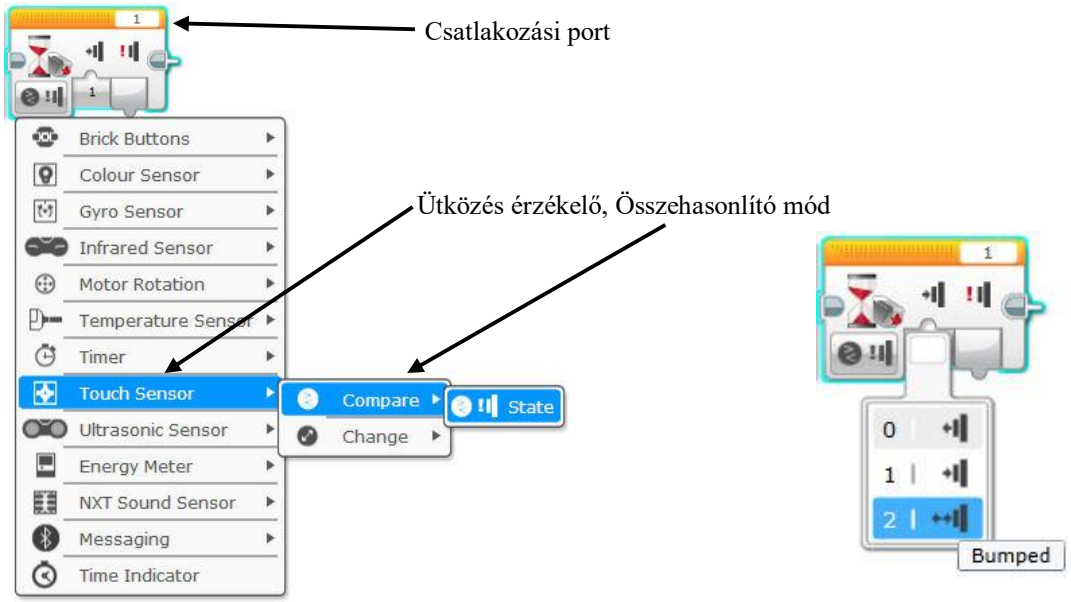
A motor vezérlésére használd a *„Large motor”* ikont! Az ikon a programhoz kapcsolva az *„On”* állapotot kell választani (ezzel bekapcsolod a motor forgását). Ennél a beállításnál csak a csatlakozási portot kell beállítani (A) és a sebességet (20). Ügyelj arra, hogy a motor tényleg az A portra csatlakozzon!





Az ütközésérzékelő kezelésére a *„Wait”* ikont használd! A programhoz kapcsolva az ütközésérzékelő paramétert kell választanod (lásd az alábbi ábra). Figyelj arra, hogy a megfelelő port legyen beállítva (a példánál ez az 1-es, az ikon jobb felső sarkában). A programba illesztett ütközésérzékelőnél a *„Bumped”* módot kell választani. Ebben az esetben a gomb megnyomása utáni felengedés esetén hajtja végre a robot a program következő utasítását. Ha nem ezt választjuk, akkor a program gyorsan végigfut és szemmel nem követhető a sebességnövekedés. (Próbáld ki és megérted!)





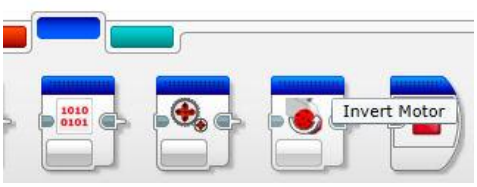
F.1.2.3. Körhinta program (kétféle forgási sebesség ütközésérzékelő hatására)

A feladat megoldásához egy ciklusban (*Loop*) elhelyezett elágazást (*Switch*) kell használnod. Az elágazás két ágán a különböző sebességű motorok vezérlő ikonjai találhatók. Az elágazás két ága között az 1-es ütközésérzékelő benyomott vagy felengedett állapota jelenti a váltást.

F.1.2.4. Körhinta program (forgásirány változtatása ütközésérzékelő hatására)

A feladat megoldásához használhatod az előző feladatban elkészített ciklusban (*Loop*) elhelyezett elágazást (*Switch*).

A ciklusból a 2-es ütközésérzékelő benyomására kell kilépni, majd megváltoztatni a motor forgási irányát. A forgási irány megváltoztatására rendelkezésre áll egy speciális blokk az *Advanced* csoportban az *„Invert Motor”*.

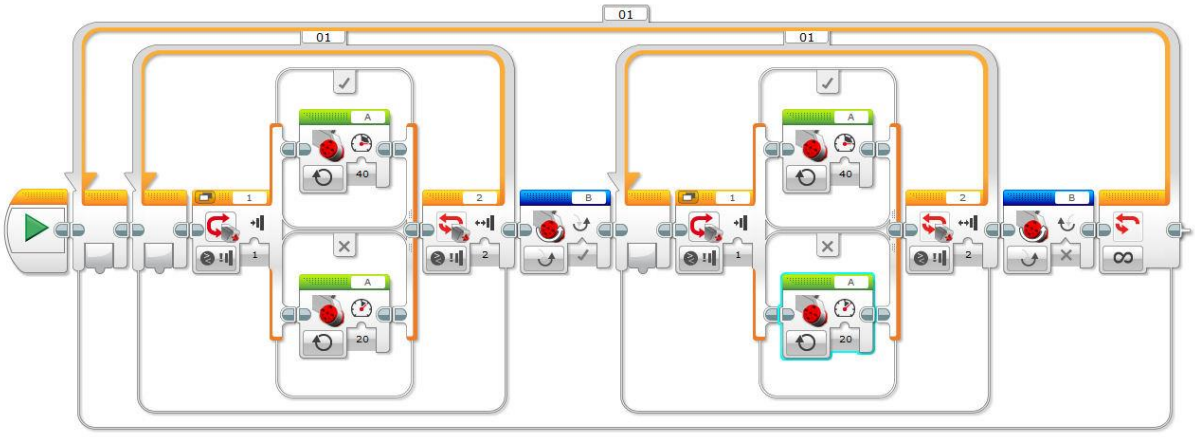


Paraméterként a motor azonosítóját kell megadni (csatlakozási port), valamint hogy milyen irányba forogjon a motor.

Az utasítás végrehajtásakor a beállított forgásirány lesz az érvényes. Tehát, ha korábban is ebben az irányban forgott a motor, akkor nem történik változás, ha ellentétes irányban forgott, akkor megváltozik a forgásirány.

(A blokk valójában az alapértelmezett forgásirányt a „*Large motor*” esetén változtatja meg, illetve állítja vissza eredeti állapotára. A parancsot csak a „*Large motor*” esetén lehet használni.)

További magyarázatok nélkül egy lehetséges megoldás forráskódja:



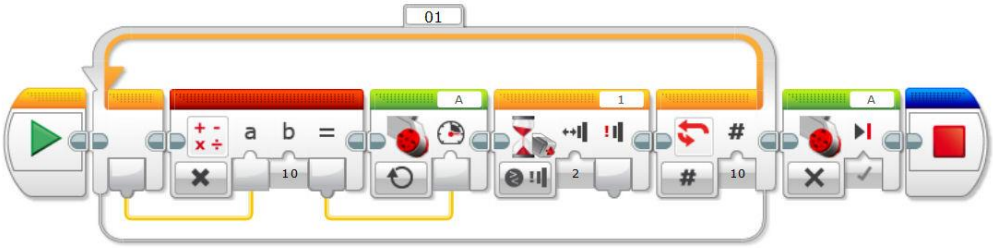
F.1.2.5. Körhinta program (forgási sebesség növelése egyenletesen)

A ciklusokban (*Loop*) van egy beépített számláló, amely értéke eggyel nő, ha ciklus utasításai végrehajtódtak és elölről kezdődnek. Az első utasítás végrehajtáskor a számláló értéke nulla. Ha ennek a számlálónak az értékét minden újrakezdéskor 10-zel megszorozzuk, akkor sorban a következő számokat kapjuk: 0x10=0; 1x10=10; 2x10=20; 3x10=30; …

A kapott értékek éppen a hinta forgási sebességét jelentik.

A szorzás elvégzésére van a programban egy utasításblokk, amelyről az elektronikus tankönyv *9.2. Számértéket visszaadó műveletek* fejezetében olvashatsz.

A megvalósításhoz a blokkokat összekell kötni az adatáramlásnak megfelelően. Részletesebb magyarázat a tankönyv *7.2. Paraméterátadás a programon belül* című fejezetében található.



1.3. PROJEKT – „KIRÁLYLÁNY SZABADÍTÁS”

Időigény

2-3 óra

Elméleti anyag

A projekt programozási részéhez az alapszenzorok használata szükséges. Ehhez leírás a bevezetőben megadott elektronikus könyv *4. Egyszerű mozgások*, *5. Szenzorok használata* fejezeteiben találhatók.

A programozás elkezdése előtt ajánlott áttanulmányozni a *4.1. Motorok vezérlése*, *5.1. Szín- vagy fényszenzor*, *5.3. Ütközésérzékelő* és *5.4. Ultrahangos távolságérzékelő* fejezeteket.

Bevezető történet

Egyszer volt (kétszer nem), hogy hol azt nem tudni, de volt egyszer egy félszemű királylány. Nem is fél szeme volt, hanem egy, de az olyan gyönyörű volt, mintha négy lett volna. Az egész világról csodájára jártak. Ez a népszerűség azonban nemcsak jó, hanem gonosz dolgokat is eredményezett.
Élt az országban egy gonosz és morcos jegesmedve, aki a népszerűségét elveszítette, amikor a királylány csodájára jártak az alattvalók. Ettől lett olyan morcos.
Telt múlt az idő és a királylány eladó sorba lépett. Ebben a sorban többen is álltak, de a királylány állt legelöl. A király férjhez akarta adni lányát, ahhoz a kérőhöz, aki … Ezt még nem találta ki, de erősen gondolkodott rajta.
A gonosz jegesmedve megelégelte a lány népszerűségét és elhatározta, hogy elrabolja. Bár ésszerű érveket nem tudott a rablás indokaként felhozni, hiszen nem evett királylányokat, csak nyers halat, de mivel unatkozott ezért jó bulinak tűnt. El is ment az udvarba kérőnek álcázva és mikor senki nem figyelt oda felkapta a királylányt és elfutott vele.
A király nagy szomorúságba esett, hiszen egyetlen lányát veszítette el. Egész nap búslakodott és azon törte a fejét, hogyan kaphatná vissza. Mígnem hírnököket menesztett szerte az országba, hogy tegyék közhírré, annak adja lánya kezét (no meg az egész lányt is), aki kiszabadítja a szörnyeteg karmai közül. A fele királyságát ugyan nem ajánlotta fel, hiszen szerette az országát és akárkinek nem adta volna oda, de lánya keze ért annyit, mint egy királyság.
Gyülekeztek a lovagok és közrendű népek, hogy szerencsét próbáljanak. Sokan elindultak a királylány keresésére, de senki sem tért vissza, vagy mert elfáradtak és megunták a keresést, vagy mert inukba szállt a bátorságuk. Teltek-múltak a hónapok. Néhány vándor híreket hozott arról, hogy a Pókkany-on és a Sötét tavon is túl, ott ahol a jegesmedve az úr, él egy búslakodó lány, aki arra vár, hogy megmentsék.
A szerencsét próbálók kudarcát látva egyre kevesebben jelentkeztek a királylány megmentésére, és eljött az a hét, amikor már senki sem akart útra kelni. A király nagyon elbúsult és már azt fontolgatta, hogy maga indul útnak, pedig már tíz éve ki sem mozdult a palotájából.
Ekkor előállt egy fiatal és tudós ember, aki épített egy gépet. Robotnak nevezte, bár a nevet a király számára megmagyarázni nem tudta. Valami munkáról beszélt, hogy a gépet eredetileg erre tervezte, de ezt a király nem értette. A gép látszólag önállóan működött és úgy látta a környezetét, mint a denevérek. A tudós megpróbálta elmagyarázni a működés részleteit az embereknek, de ők kinevették, mivel egy szót sem értettek az egészből.
Más jelentkező a királylány megmentésére azonban nem volt, így a király kénytelen volt a robot-lovagot megbízni a szabadítással. A tudósnak volt azonban egy feltétele. Ha sikerrel jár, akkor nem tart igényt a

királylány kezére (sem a vele járó egyebekre), hanem csak annyit kér, hogy a királyi udvarban élete végéig dolgozhasson, és a robotjának mindig legyen feltöltött akkumulátora. A király ebbe örömmel bele is egyezet, annál is inkább, mert fogalma sem volt arról, mi az az akkumulátor (de biztosan nem lehet valami nagy dolog, ha egy ilyen tudósfélének is van belőle).
A tudós tehát munkához látott. Összehívta a vándorokat, akiknek elbeszéléseiből egy térképet rajzolt a királylányhoz vezető útról. Ez alapján elkészítette a robot algoritmusát, majd útnak eresztette. Most már csak várnia kellett az eredményre. És várt, és várt, és várt, …

F.1.3.1. Feladat – Ultrahang szenzor használata akadály észlelésére

Szint: 

Írj programot, amelyet végrehajtva a robot egyenesen halad előre egy akadályig, majd azt 15 centiméterre megközelítve jobbra fordul kb. 90 fokkal.

F.1.3.2. Feladat – Fény/szín szenzor használata eltérő színű felület érzékelésére

Szint: 

Írj programot, amelyet végrehajtva a robot egyenesen halad előre egy fekete vonalig, majd azt elérve jobbra fordul kb. 90 fokkal.

F.1.3.3. Feladat – Egyenes haladás giroszenzorral

Szint: 

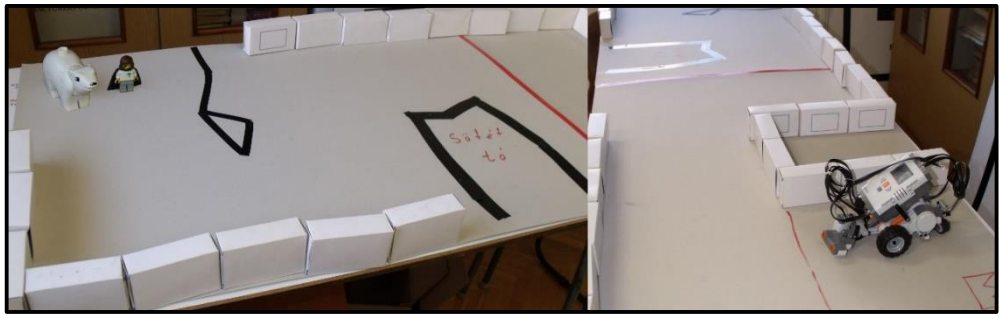
Írj programot, amelyet végrehajtva a robot egyenesen halad előre! Tapasztalatból lehet mondani, hogy a robot felépítéséből, vagy a felület minőségéből következően előfordulhat, hogy a robot nem halad egyenes vonalban, akkor sem, ha a programjában ezt állítottuk be (valamelyik irányba „húz”). Ennek oka lehet, hogy a felépítés nem szimmetrikus, vagy a motorok tengelyei közül valamelyik szorul, súrlódik, esetleg a felület tapadása eltérő. Néha nehéz az okot megtalálni, és ha a hardverben nincs magyarázat a kanyarodásra, akkor a programba építve kell a problémát megoldani. A robot giroszenzorát használva írj olyan programot, amely a robotot akkor is egyenesben tartja, ha a fenti jelenség bekövetkezik!

F.1.3.4. Feladat – „Királylány szabadítás”

Szint: 

Írj programot, amelyet végrehajtva a robot a „Királyi vár”-tól indulva eljut a királylányig! Mozgása közben nem érintheti meg a hegyeket szimbolizáló dobozokat és nem mehet keresztül teljes terjedelmében a tavakat szimbolizáló szigetelő szalagokon. Ha hozzáér a dobozokhoz, vagy teljes terjedelmében áthalad a szigetelő szalag tavak bármelyikén, akkor újra kell kezdenie a mozgását a királyi vártól.

Térkép



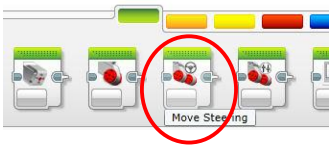
Programozási ötletek

Az alábbi leírások segítenek a program felépítésének tervezésében kivitelezésében.

A robotnak a mozgása során több helyen fordulnia kell a megfelelő irányba. Ehhez, lesz amikor tapasztalati úton meghatározott szöggel kell elfordítani és lesz, amikor a szenzorai által mért értékek alapján kell a fordulást elkezdeni. A három programozási ötlet segít ezeknek a kódrészleteknek az elkészítésében.

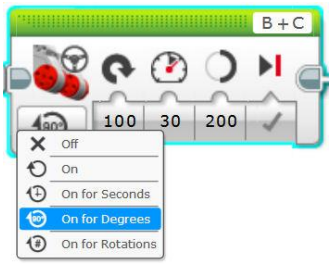
Fordulás adott tapasztalati szöggel

A robot fordulásához használd az *Action* menü (zöld színű) *Move steering* (kormányzott mozgás) utasításblokkot.

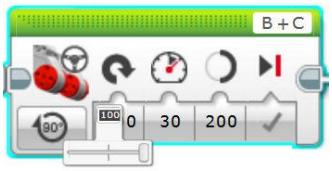


A robot fordulását a motorok tengelyének elfordulási szögével érdemes szabályozni. Első lépésben válaszd az *On for Degrees* beállítást!

A fordulás irányát a *Steering* (kanyarodás) paraméter csúszkájának beállításával szabályozhatod. Ha valamelyik irányban elhúzod a csúszkát, akkor a robot kerekei eltérő sebességgel forognak, így a robot kanyarodni vagy fordulni fog. A csúszka iránya a kanyarodás irányát jelenti, ha a motorokat nem fordítva szerelted be.



A motorok forgási sebességét ne állítsd nagy értékre, mert ha gyorsabban mozog a robot, akkor pontatlanabb lesz. Jelenleg ez az érték a képen szereplő utasításblokknál: 30.



Már csak az elfordulás mértékét kell meghatároznod. A képen ez az érték 200 fok. Ez egy tapasztalati érték, ami azt jelenti, hogy próbálgatással határozd meg milyen beállításnál fog a robot a kívánt szöggel elfordulni (pl.: 90 fokkal).

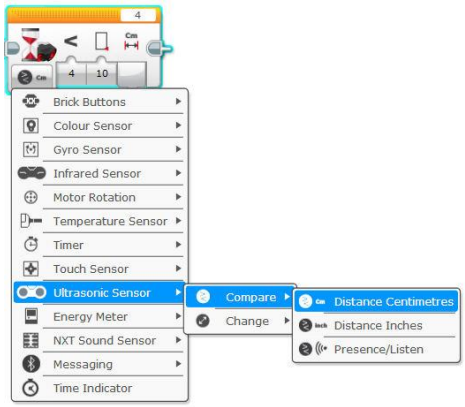
A beállított érték nem a robot elfordulási szögét jelenti, hanem a motorok tengelyének elfordulási szögét. (Ha nem érted a kettő közötti különbséget, akkor tanárod segít a megértésben. Kérj segítséget tőle!)

Ultrahangszenzor használata távolság érzékelésre

Az ultrahang szenzor képes megmérni a robot előtti akadálytól való távolságot centiméterben. Ezt a mért értéket használhatjuk fel arra, hogy ha a távolság egy beállított számértéknél kisebbre csökken, akkor reagáljon erre a robot. A programblokk *Flow Control* (Program Vezérlés) csoportban (narancssárga) található *Wait* (Várakozás) utasítás.

Ennek hatására az összeépített programban csak akkor hajtódik végre a következő utasítás, amikor a beállított feltétel teljesül.

Első lépésben az ultrahangszenzort (*Ultrasonic Sensor*) kell kiválasztani a beillesztett blokkban, annak is az összehasonlító (*Compare*) működési módját és a centiméter skálát (*Distance Centimeters*). Ennek hatására be lehet állítani, hogy milyen mért érték esetén kezdje meg a robot a következő utasítás végrehajtását.



A képen ez az érték < 10. Tehát, ha a robot 10 cm-en belül akadályt érzékel, akkor a következő utasítást kezdi végrehajtani, ami lehet például egy fordulás.

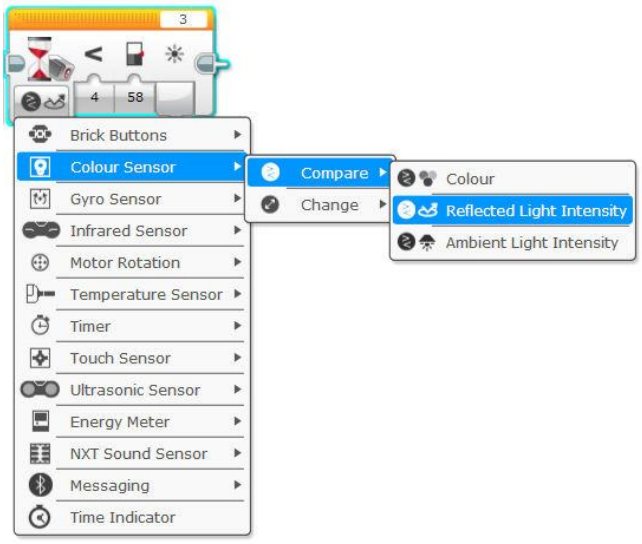
Színszenzor használata eltérő színű felület érzékelésére

A robot színszenzora kétféle módon képes a felület színét érzékelni. Egyrészt képes az alapszínek meghatározására (nem pontos a határ az egyes színek között, tehát a piros nem minden árnyalatát ismeri fel). Másrészt képes fényszenzorként viselkedni, amikor egy 0-100 közötti értéket mér a különböző fényességű és színű felületeken. Attól függően, hogy a projektben a tesztpálya milyen színű és milyen szín jelöli a „tavak” határát, lehet a különböző módok közül választani. Például egy fehér alapú pályán, ha a „tó” határa fekete, akkor a színszenzor is alkalmas a megoldásra. Az alábbi példában a fényszenzor használatát mutatjuk be.

Ehhez először meg kell mérni, hogy a szenzor milyen értékeket érzékel a fehér és fekete felületen. Ehhez érdemes a képernyőmenüt használni. A tankönyv *3. A robot képernyőmenüje* fejezetében olvashatsz erről többet.

A két mért érték számtani közepe (a két szám összegének a fele) alkalmas lesz arra, hogy a robot megkülönböztesse a fekete és fehér színt. Például: ha fehér felületen 82-t mér, feketén pedig 34-et, akkor ezek számtani közepe: (82+34)/2 = 58. Tehát ha robot szenzora 58-nál nagyobb értéket mér, akkor azt fehérként értelmezi, ha kisebbet, akkor feketeként.

A programban az ultrahangszenzornál már bemutatott *Wait* blokkot kell használni. Ezt a programba illesztve először a *Colour Sensor* (Színszenzor), *Compare* (Összehasonlítás), *Reflected Light Intensity* (Megvilágított felületű mérés) beállítást végezd el.

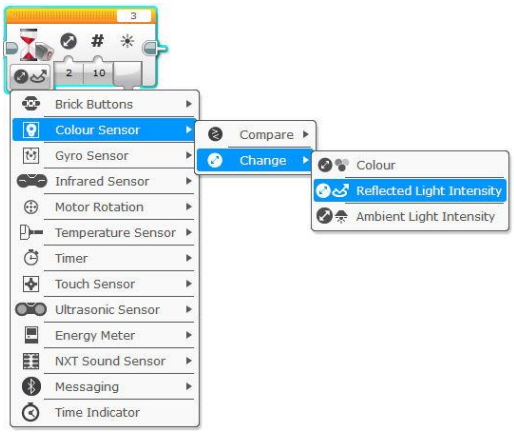


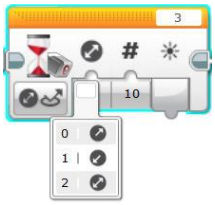
A mérési adataidnak megfelelően állítsd be a határt. A képen ez < 58.

A programba illesztve az utasítást a robot akkor fogja a soron következő parancsot végrehajtani, ha a feltétel teljesül, tehát ha színszenzora 58-nál kisebb értéket mér (vagyis elérte a fekete vonalat).

Ekkor reagálnia kell például egy megfelelő irányú fordulással, amit a következő utasítás adhat meg.

A színszenzort fényszenzorként használva (*Reflected Light Intensity*) van egy másik lehetőség is a felület színváltozásának érzékelésére. Ebben az esetben nem kell méréseket végezned és számtani átlagot számolnod, így a használata talán könnyebb. Válaszd ki a *Flow Control* (narancssárga) csoport *Wait* blokkját és a működési módot állítsd be *Colour Sensor/Change/Reflected Light Intensity* állapotra!



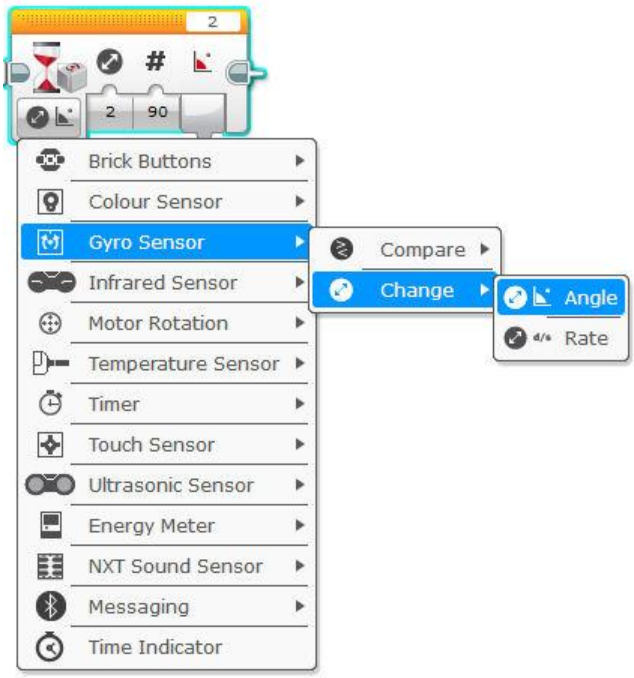


Ekkor a blokk csak a változás mértékét figyeli, ami három féle lehet: a mért érték nő, csökken vagy bármilyen irányban, de változik. A változás mértékét a második paraméterként tudod számszerűen beállítani. Az ikonon ez az érték 10, ami azt jelenti, hogy a fényszenzor által mért érték 10-zel megváltozik (a beállított irányban), akkor teljesül a feltétel és hajtja végre a robot a következő utasítást. A fekete és fehér színek között ritkán fordul elő, hogy 10-nél kisebb lenne a mért különbség. Tehát a robot a változást előzetes mérések nélkül is észlelni tudja. A fekete színű felületen mért érték kisebb, mint a fehér színű felületen mért.

Elfordulási szög megadása giroszenzor használatával

A robotkészletben hardver elemként rendelkezésre áll egy giroszenzor, amely képes a robot tényleges elfordulását mérni fokokban. A szenzor pontossága a gyári adatok szerint ±1 fok. Ez a tapasztalatok szerint függ attól, hogy a szenzort hová szerelted fel a robotra. Ha közvetlenül a motorok tengelyéhez közel, vagy azoktól távolabb, akkor a fordulás szögét eltérően érzékeli, tehát szükség van tesztelésre, és ez alapján az elfordulási szög tényeleges módosítására.

Ideális esetben a 90 fokos elforduláshoz ténylegesen 90 fokos giroszenzorral mért érték tartozik.



A szenzor az elfordulási szöget előjelesen méri, ezért nem mindegy, hogy melyik irányba fordul a robot. A motorok robotra szerelésének irányától függően ±90 fokot mérhet a szenzor.

Az fordulás megkezdése előtt a szenzort érdemes nullázni (*resetelni*), mivel a robot indításától függ a nulla irány és ehhez képest mér a szenzor.

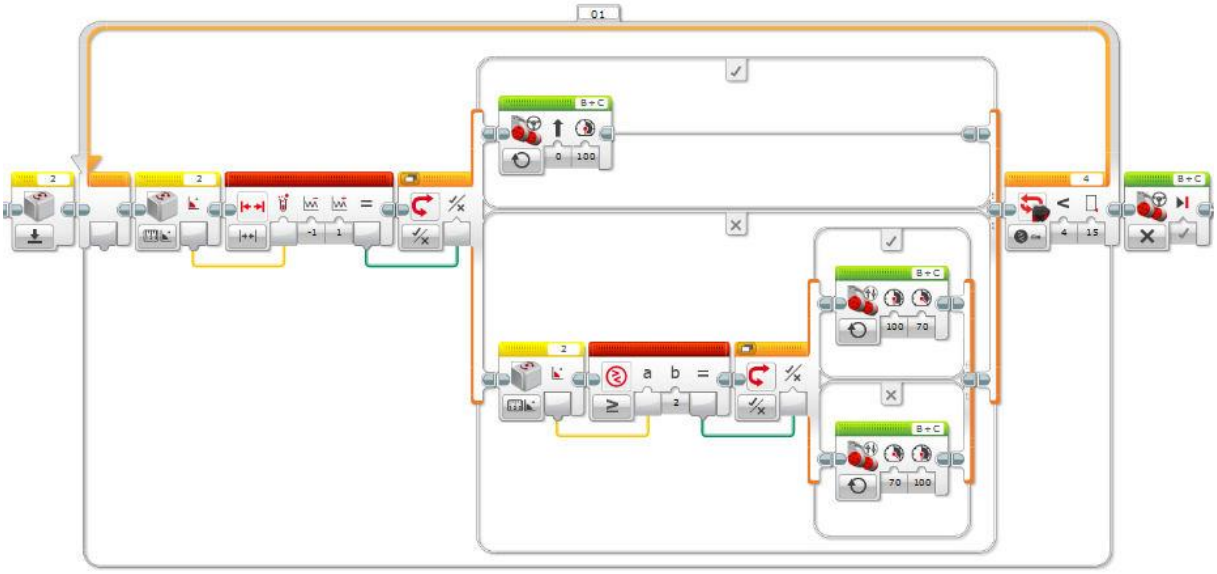
A korábban már látott *Wait* blokkot érdemes használni, és beállításként a *Gyro Sensor/Change/Angle* értéket választani. Ebben az esetben a szenzor a robot fordulása során mért változást figyeli (fokban). Ha ez meghaladja a 90 fokot, akkor hajtja végre a programszál következő utasítását. A fordulás irányától függően a változás lehet $+90^{o}$ vagy -90^{o}. Mindkét esetben teljesül a feltétel.

Egyenes haladás giroszenzorral

A programírás során a giroszenzort használjuk. A szenzor képes a robot elfordulását érzékelni ±1 fokos pontossággal. Az algoritmus lényege, hogy ha a szenzor ennél nagyobb eltérést mér, akkor a motorok különböző sebességű forgatásával korrigáljuk a mozgást, így a robot egyenesben tartható.

A program megírása már összetettebb programozási ismereteket feltételez. Mindenképpen érdemes megírni az algoritmust, de későbbi időpontra is halasztható.

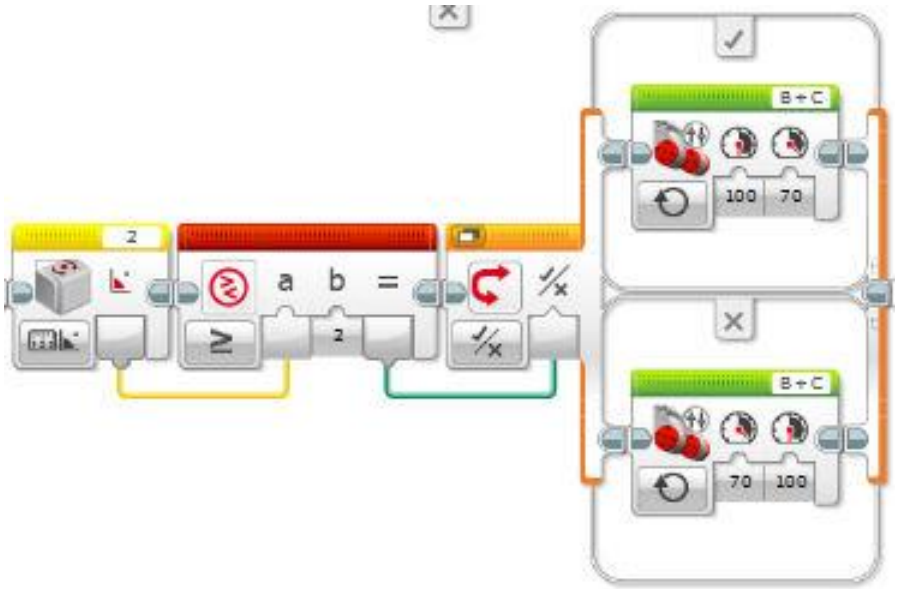
A teljes program forráskódját mutatja a következő ábra.



A program a giroszenzor alaphelyzetbe állításával indul. Ennek hatására nulla értékről kezd el ismét mérni. A cikluson belül folyamatosan vizsgáljuk, hogy a mért érték ±1 fokos határon belül van-e. Ehhez a *Data Operation* csoport *Range* blokkját használjuk. A működési mód *Inside*, ami azt jelenti, hogy akkor ad vissza igaz értéket a modul, ha a mért érték a beállított két határérték között van (beleértve a határokat is). A határok jelen esetben +1 illetve –1. Ebben az esetben a motorok azonos sebességgel forognak, és a robot egyenesen előre halad.

Ha giroszenzor által mért érték kívül esik a ±1-es határokon, akkor azt kell eldönteni, hogy pozitív vagy negatív irányban esik-e kívül. Ebben az esetben ugyanis a robot eltért az egyenes haladási iránytól és ez fogja eldönteni, hogy melyik irányban. Megvizsgáljuk tehát, hogy a mért érték nagyobb vagy egyenlő-e, mint 2 vagy sem. A két esetben a két motort különböző sebességgel működtetve vissza tudjuk terelni az egyenes haladási irányhoz.

A belső elágazásban szereplő feltétel és az utasítások:



1.4. PROJEKT – „KÖVETÉS”

Időigény

2-3 óra

Elméleti anyag

A projekt programozási részéhez az alapszenzorok használata és a vezérlési szerkezetek alapismerete szükséges. Ehhez leírás a bevezetőben megadott elektronikus könyv *4. Egyszerű mozgások, 5. Szenzorok használata* és *6. Vezérlési szerkezetek* fejezeteiben szerepelnek. A programozás elkezdése előtt ajánlott áttanulmányozni a *4.1. Motorok vezérlése, 5.1. Szín- vagy fényszenzor, 5.4. Ultrahangos távolságérzékelő, 6.1. Ciklusok* és *6.2. Elágazások* fejezeteket. A nehezebb feladatokhoz a *7.2. Paraméterátadás a programon* belül, valamint a *7.3. Szenzorok paraméterezése* fejezetekben található hasznos információ.

Bevezető történet

A sikeres királylány szabadítás után a tudós elismertsége jelentősen növekedett. Mindenki szeretett volna megismerkedni vele és messzi tájakról is csodájára jártak a robotjának. Már kicsit elege is lett a népszerűségből és abból, hogy minden második kérdés az volt „Mit tud ez a robot?”. Így a legtöbbször csak annyit válaszolt, hogy amit a beleprogramoztak. Mivel ezt a kérdezők nem értették, de tudatlanságukat nem szerették volna elárulni, ezért a beszélgetés itt többnyire abba is maradt.
A királylány visszatértének megünneplésére nagy vigadalmat rendezett a király, amelynek része volt egy felvonulás is. Azt szerette volna, ha a tudós és a robot is együtt vonul az ünneplőkkel a sorban. Ehhez olyan programot kellett készítenie a tudósnak, amely alapján a robot követni tudja az előtte haladót. Mindez nem kis fejtörést okozott neki, mivel nem tudta, hogy a robot melyik szenzorát használja, pórázon pedig nem vezethette. Szerencsére a felvonulási útvonal hagyományosan ugyanaz volt már évszázadok óta. Több módszerrel is kísérletezett, és úgy gondolta, hogy majd az ünnepség előtt eldönti melyik a leghatékonyabb.

F.1.4.1. Feladat – Követés ultrahangszenzorral (egyszerű)

Szint: 

Írj programot, amelyet végrehajtva a robot képes követni egy előtte haladó tárgyat! A robotra szerelt ultrahangszenzort használd a követéshez! Ha szenzor 15 centiméteren belül akadályt érzékel, akkor a robot induljon el egyenesen felé, egyébként lassan forogjon körbe!

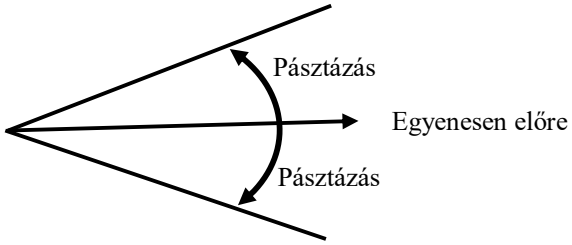
(A program hibája, hogy ha nincs a robot előtt akadály, akkor a körbe forgás irányától függően esetleg sokáig tart, míg újra megtalálja a követendő tárgyat. Ez akkor jelentkezik leginkább észrevehetően, ha követendő tárgy a robot forgási irányával ellentétes irányba mozdul el. Tehát a feladat megoldásában elkészített programmal csak nagyon lassan lehet követni egy nagyon lassan mozgó tárgyat. Kicsit hatékonyabbá tehető az algoritmus az F.1.4.2-es feladat megoldásával.)

F.1.4.2. Feladat – Követés ultrahangszenzorral (összetettebb)

Szint: 

Írj programot, amelyet végrehajtva a robot képes követni egy előtte haladó tárgyat, a robotra szerelt ultrahangszenzor segítségével! Ha a szenzor 15 cm-en belül akadályt érzékel, akkor a robot induljon

egyenesen felé, egyébként pedig adott szöggel (nem teljesen körbe) forduljon jobbra majd balra (váltakozva)! Tehát keresse a céltárgyat jobbra-balra pásztázva.



F.1.4.3. Feladat – Útvonalkövetés egy fény/szín szenzorral

Szint:

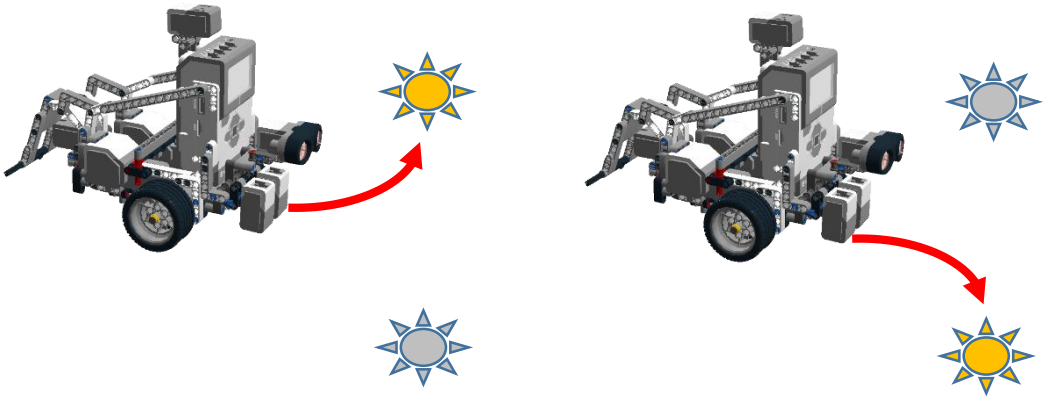
Írj programot amelyet végrehajtva a robot egyetlen fény/szín szenzorával képes követni a világos felületen lévő sötét színű sávot (pl.: fehér felületen a fekete szigetelőszalag csíkot)! A sötét sáv nem egyenes akár bal, akár jobb ívű kanyarokat is tartalmazhat.

F.1.4.4. Feladat – Fénykövetés két fény/szín szenzorral

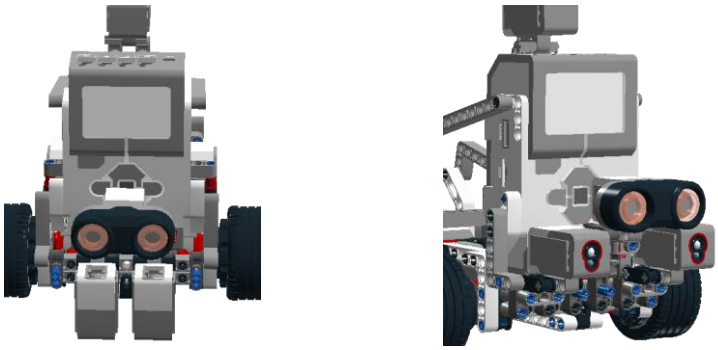
Szint:

Írj programot, amelyet a robot végrehajtva két fény/szín szenzorával képes követni az elemlámpa fényét!

Mindig arra fordul, amerre a lámpafény erősebb.



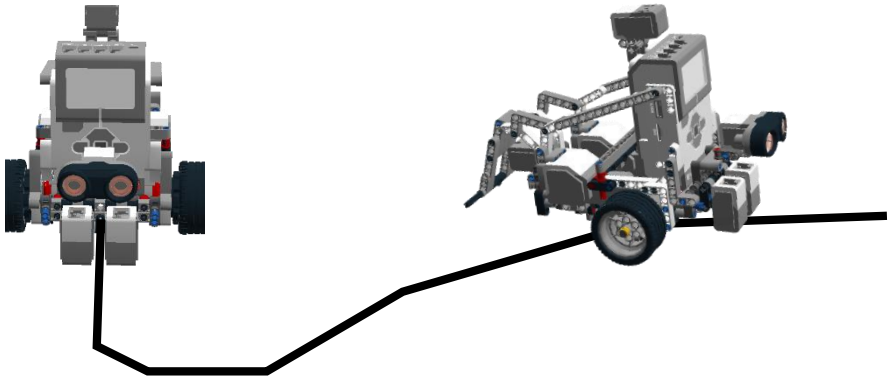
A két fény/szín szenzor lehet akár lefelé néző állapotban, akár előre néző állapotban, de egymás mellet.



F.1.4.5. Feladat – Útvonalkövetés két fény/szín szenzorral

Szint:

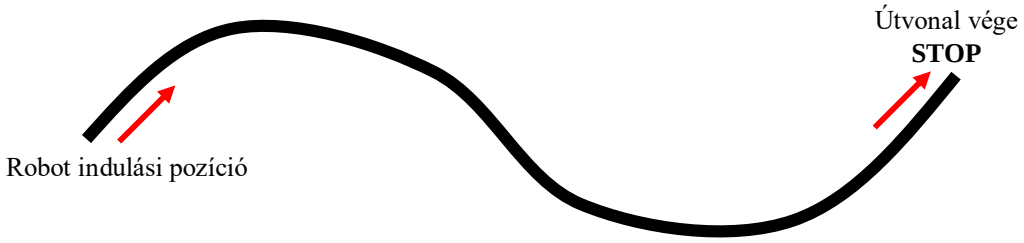
Írj programot, amelyet végrehajtva a robot két fény/szín szenzorával képes követni a világos felületen lévő sötét színű sávot (pl.: fehér felületen a fekete szigetelőszalag csíkot)! A két fény/szín szenzor a sötét sáv két különböző oldalán helyezkedjen el, egymással párhuzamosan, lefelé néző állapotban! A két fény/szín szenzor távolságát úgy érdemes megválasztani, hogy mindkettő a világos színű felület fölött legyen, de közel a sötét színű sávhoz.



F.1.4.6. Feladat – Útvonal végének érzékelése

Szint:

Írj programot, amelyet végrehajtva a robot egyetlen fény/szín szenzorával követi a világos felületen lévő sötét színű útvonalat (pl.: fehér felületen a fekete szigetelőszalag csíkot) és ha elérte az útvonal végét, akkor megáll! Az útvonal végét nem jelzi semmiféle objektum, csak egyszerűen vége lesz.



Programozási ötletek

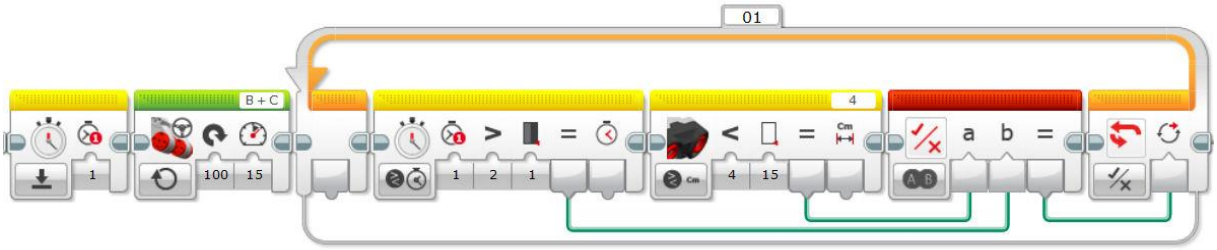
F.1.4.1. Követés ultrahangszenzorral (egyszerű)

A megoldás során egy elágazást használunk (*Switch*), amelyet az ultrahang szenzor vezérel. Ha a mért távolság 15 cm-nél kisebb, akkor a robot egyenesen előre halad, egyébként pedig valamelyik irányba fordul. Mindkét esetben a motorok vezérlése „*On*” állapotú. Ha ezt az elágazást egy végtelen ciklusba tesszük, akkor készen is van a program.

A hátránya, hogy ha elveszíti az előtte haladó tárgyat, akkor lehet, hogy csak egy teljes fordulás után találja meg újra, ha a tárgy nem haladt előre túl sokat. A tárgy újbóli megtalálása függ a robot fordulási irányától és a tárgy kanyarodásától. Ha ezek ellentétesek, akkor áll elő a fenti probléma. Ilyen esetekben a robot körbe-körbe forog, míg meg nem lát 15 cm-en belül valamit.

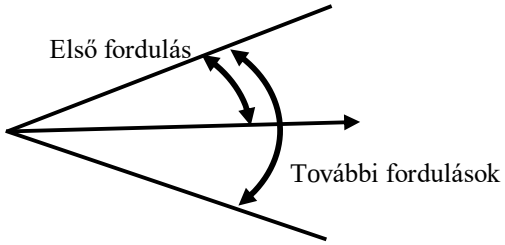
F.1.4.2. Követés ultrahangszenzorral (összetettebb)

Az elsődleges problémát az jelenti, hogy most nem forognia kell a robotnak, hanem csak fordulnia, pl. adott ideig. Ha motor fordulásszabályozását másodpercben adjuk meg, akkor nem tud figyelni közben az ultrahangszenzorra. Tehát egyszerre két dolgot kell figyelnie: az idő múlását és az előtte lévő akadály távolságát. Bármelyik feltétel is következik be (letelt a fordulási idő vagy meglátta 15 cm-en belül a tárgyat) el kell indulnia előre, így a motorok vezérlése csak „*On*" állapotban használható. Az idő múlását célszerűen egy stopperrel mérhetjük (*Timer*). A két feltétel között VAGY logikai kapcsolat áll fenn. Azokban az esetekben, ahol a feltétel teljesülését két vagy több esemény bekövetkezése szabályozza egy általános megoldás adható, az ilyen feltételeket nevezzük összetett feltételnek. Ezekben az esetekben a *Data Operetions/Logic Operetions* blokkjával tudjuk összekapcsolni a két feltételt. A szokásos *Wait* modul helyett egy ciklusba kell helyezni a szenzorokat figyelő blokkokat (sárga színűek és a *Sensor* csoportban találhatók). Eltérően a *Wait* modultól nem várakoztatják a program utasításainak végrehajtását, hanem csak kiolvassák a mért értéket és ezt adják vissza. A várakoztatást itt a ciklus végzi, amely akkor fejeződik be, ha feltételek teljesülnek. A feladatnál a motor vezérlését hagyjuk „*On*" állapotban. és egy cikluson (*Loop*) belül figyeljük az eseményeket. Ezek bekövetkezését a megfelelő logikai művelettel összekapcsolva megkapjuk a ciklus kilépési feltételét.



Az első ikon a stoppert nullázza le, majd bekapcsoljuk a motort helyben forgásra. A cikluson belül VAGY kapcsolattal összekötve a 15 cm-en belüli akadályérzékelés és az 1 másodperc letelte szerepel.

Az oda-vissza irányú mozgást a fenti programrészlet ismételgetésével kapjuk, csak a forgásirányt kell változtatni. Érdemes észrevenni, hogy a jobbra-balra történő, szimmetrikus mozgás első fordulása fele olyan hosszú, mint a többi, ezért ezt külön érdemes kezelni.



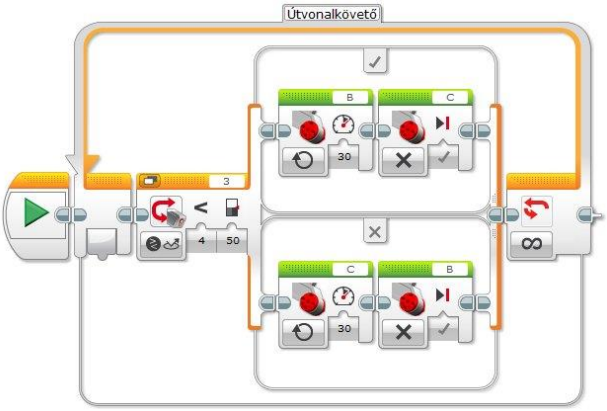
F.1.4.3. Útvonalkövetés egy fény/szín szenzorral

Az alábbi leírások segítenek a program felépítésében kivitelezésében.

A robot mozgása során, fehér felületen kell követnie egy fekete színű útvonalat. Ehhez elegendő egyetlen színszenzor.

Egyszenzoros útvonalkövetés

Első lépésben mérjük meg, hogy fehér illetve fekete színre milyen értéket mutat a robot színszenzora. A színszenzort ismét fényszenzor módban használjuk (*Compare/Reflected Light Sensity*).



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 (azzal, amelyik az útvonalnak azon az oldalán van, amelyiken a szenzor), a másik motor kikapcsolt állapotban van. Ennek hatására a robot ráfordul az útvonalra. Ekkor azonban 50-nél kisebb értéket mér a szenzora. Ebben az esetben a robot a másik motort forgatja előre, az előzőleg működőt leállítja, és lefordul az útvonalról (szenzora nem az útvonal fölött lesz). Ezt egy végtelen ciklusban ismételve a robot kígyózó mozgással követi az útvonalat. A motorokat nem célszerű túl nagy sebességgel működtetni, mivel ekkor a robot nagyokat fordul, és elveszítheti a követendő útvonalat. A feladat megoldásánál 30-as erővel működtettük a motorokat.

A program működése során a robot tulajdonképpen nem az útvonalat követi, hanem a fekete sáv határvonalát. Ha megváltoztatjuk az elágazást vezérlő ikon relációs jelének irányát, akkor a robot a csík másik oldalát követi.

Ha az elágazás két szálán a második motort nem állítjuk le, hanem kisebb sebességgel hátrafelé forgatjuk, akkor ugyan lassabb lesz az útvonalkövetés, de élesebben kanyarodó útvonalat is tudunk követni. A két szálon szereplő motorok esetén a lényeg, hogy különböző sebességgel működjenek. Érdemes tesztelni néhány beállítást.

A fényszenzor mód helyett színszenzor módot is használhatunk az elágazás vezérlésére. Az útvonalkövetés pontatlanabb lesz, mivel a fekete-fehér közötti átmenet nem éles, a vonal határán a szürke valamely árnyalatát érzékeli a szenzor.

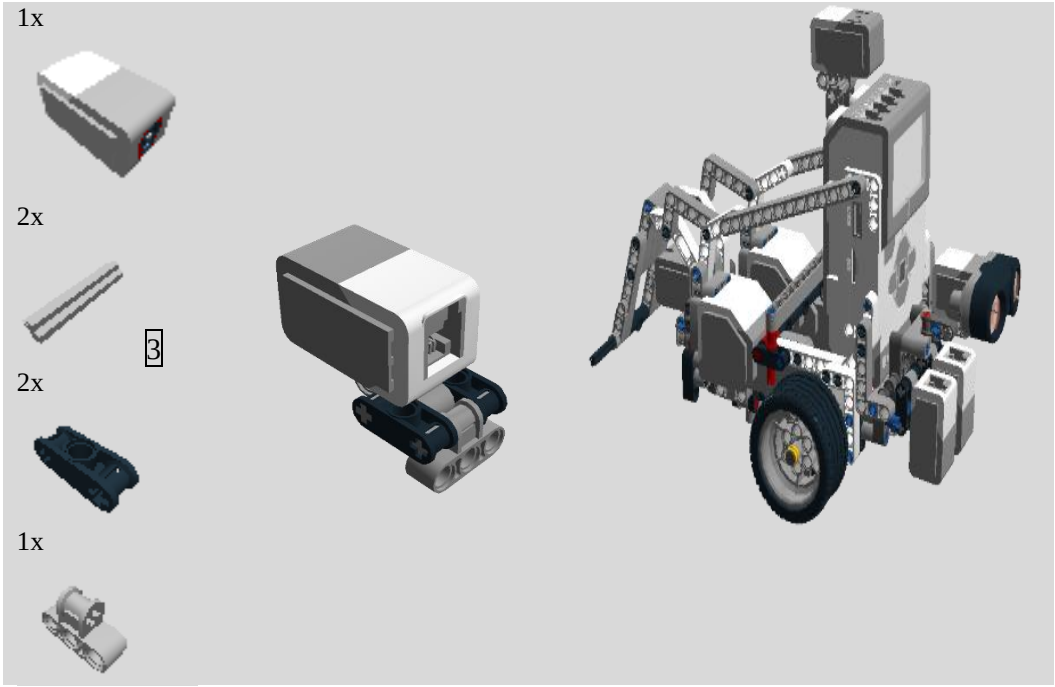
F.1.4.4. Fénykövetés két fény/szín szenzorral

A feladat megoldása azonos ötletet tartalmaz az egyszenzoros útvonalkövetéshez. Az elágazás két szálán a balra illetve jobbra fordulást vezérlő motorikonok vannak. A feltételt pedig a két fény/szín érzékelő által mért értékek nagyságrendje szabályozza. Ha a bal oldali szenzor mér nagyobb értéket,

akkor a robotnak balra kell fordulni, egyébként pedig jobbra. Itt nincs olyan állapot, amikor a robot egyenesen haladna. Továbbra is kígyózó mozgással követi a fényforrást.

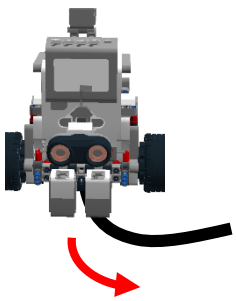
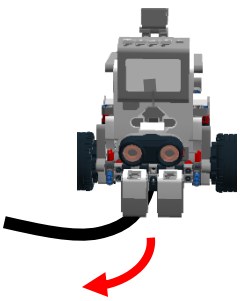
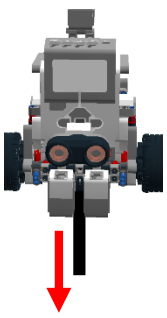
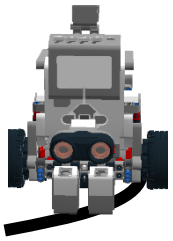
Bonyolultabb elágazás kezeléssel beépíthető az egyenes haladás is. Lásd a következő feladat megoldási ötletét.

A fényszenzor előre néző állapotú építése és rögzítése:

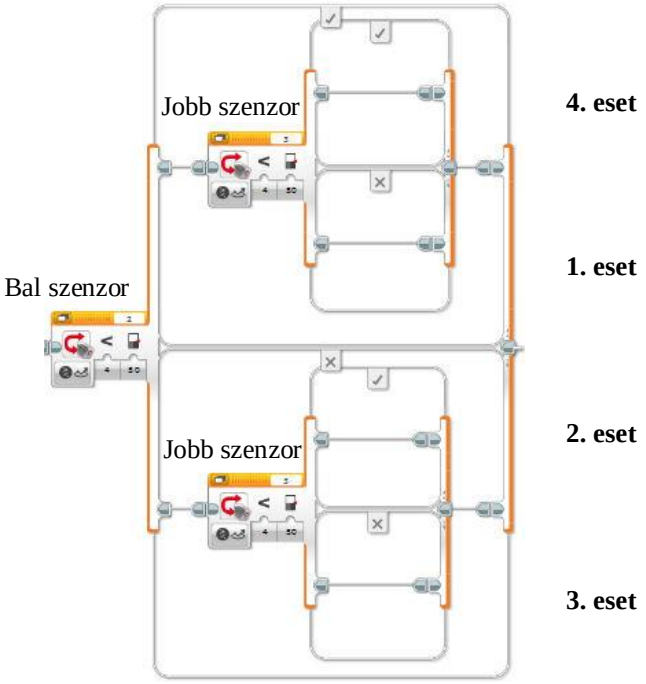


F.1.4.5. Útvonalkövetés két fény/szín szenzorral

Az egy fény/szín szenzorral történő útvonalkövetéshez képest most az a különbség, hogy a két szenzor által mért értéktől függően négyféle esetet különböztethetünk meg. A mért értékek alapján a négy állapotot a következő táblázat foglalja össze:

1. eset Bal szenzor – fekete Jobb szenzor – fehér	**2. eset** Bal szenzor – fehér Jobb szenzor – fekete	**3. eset** Bal szenzor – fehér Jobb szenzor – fehér	**4. eset** Bal szenzor – fekete Jobb szenzor – fekete
Balra kell fordulni 	Jobbra kell fordulni 	Egyenesen kell haladni 	Lassan fordulni kell valamelyik irányba, vagy lassan haladni kell egyenesen előre, amíg az első **két** eset valamelyike be nem következik. 

A négy esetet egymásba helyezett elágazásokkal valósíthatjuk meg. Összesen három elágazás (*Switch*) szükséges hozzá. Az általános szerkezetet a következő ábra értelmezi:



Az elágazások szálain a két motort vezérlő ikonok szerepelnek *On* állapotban, a táblázatban feltüntetett működésnek megfelelően.

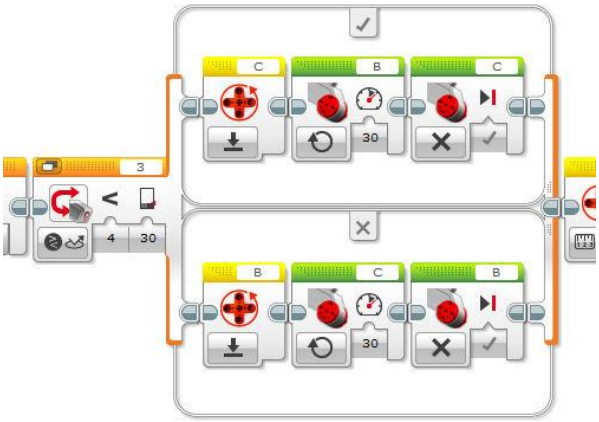
F.1.4.6. Útvonal végének érzékelése

Az útvonal végének érzékelésére azt használjuk ki, hogy egy fény/szín szenzorral történő útvonalkövetés esetén a robot kígyózó mozgással halad a vonal szélén és a motorok váltakozva működnek. Tehát a motorokba beépített elfordulási szögmérők által mért érték egy bekapcsolás időtartama alatt nem változnak nagy értékkel. Ha a robot egyenesen halad, akkor ez a növekedés 10-90 fok között van (nem a robot fordul ennyit, hanem a motor tengelye). Ha a robot kanyarodik, akkor a kanyar mértékétől függően ez az érték lehet nagyobb. Ha a robot elérte az útvonal végét, akkor kanyarodni fog, mégpedig jelentősen, hiszen a tapasztalat szerint visszafordul és a vonal másik oldalát kezdi követni visszafelé.

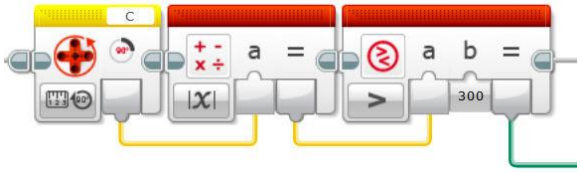
Az útvonal végének érzékelése tehát történhet úgy, hogy ha a motorok közül valamelyik egy működési szakaszban pl. 300 foknál nagyobb értékkel fordul, akkor elértük az útvonal végét és meg kell állni.

Érdemes mindkét motort figyelni, bár az egyszenzoros útvonalkövetés megírt algoritmusa alapján az útvonal végén a robot biztosan mindig ugyanabba az irányba fordul.

A forráskód fontosabb elemei:



Az útkövetés elágazásában vissza kell állítani az elfordulás mérőket alaphelyzetbe, hogy minden működési szakaszban nulláról kezdje mérni az elfordulás szögét. Mindig annak a motornak a szögmérőjét kell visszaállítani, amelyik éppen kikapcsolt állapotban van, így a következő bekapcsoláskor nulláról kezdve fog mérni.



Az elágazásból kilépve az elfordulási szög abszolút értékét kell a kiválasztott határértékhez hasonlítani, mivel a forgásiránytól függően a szög előjelet is kap. Amennyiben a motorokat biztosan csak előre forgatjuk és a motorokat is megfelelően szereltük fel a robotra, akkor biztosak lehetünk benne, hogy csak pozitív szögeket kapunk. Ekkor az abszolút érték használata felesleges. Az összehasonlítás eredményeként kapott logikai értéket használhatjuk a ciklus kilépési feltételeként. Ha a másik motorra is elkészítjük az utasítássort, akkor VAGY kapcsolattal összekötve őket a kapott eredmény lehet a kilépési feltétel.

1.5. PROJEKT – „KINCSBEGYŰJTÉS”

Időigény

2 óra (1 óra építés, tesztpálya készítés, 1 óra programozás)

Elméleti anyag

A projekt programozási részéhez az alapszenzorok használata és a vezérlési szerkezetek alapismerete szükséges. Ehhez leírás a bevezetőben megadott elektronikus könyv *4. Egyszerű mozgások, 5. Szenzorok használata* és *6. Vezérlési szerkezetek* fejezeteiben szerepel. A programozás elkezdése előtt ajánlott áttanulmányozni a *4.1. Motorok vezérlése, 5.1. Szín- vagy fényérzékelő, 6.1. Ciklusok* és *6.2. Elágazások* fejezeteket.

Bevezető történet

A király egyre többször vette igénybe a tudós és a Robot segítségét. Már szinte minden problémával hozzá fordult, tudva, hogy nem ismernek lehetetlent és minden kérést kihívásként kezelnek, amely megoldásáig folyamatosan dolgoznak. A sikeres munkához persze feltöltött akkumulátorok is kellettek. Ezekből nem volt hiány, mert a király egy külön munkacsapatot hozott létre, akik feladata a robot energiaszükségletének biztosítása volt.
A legújabb küldetésben egy kincsvadászatot rendelt meg az uralkodó. Egyébként is fogytán volt a kincstári aranytartalék.
Az ország határán egy széles, gyors sodrású folyó hömpölygött, aminek a sziklás partján kincsek garmadája sorakozott. Azt, hogy ki rakta őket oda már mindenki elfelejtette, de összegyűjtésükkel többen is próbálkoztak, nem sok sikerrel. A probléma az volt, hogy a sziklás partra nem tudtak felmászni, csak a folyón haladva lehetett volna begyűjteni a kincseket. Az erős vízáramlatokat figyelembe véve ez reménytelennek tűnt. Így a kincsek hosszú idő óta a folyó partján nyugodtak.
Megbízta tehát a király a tudóst, hogy építsen és programozzon robotot a kincsek begyűjtésére. Közben a király szókincse is egyre bővült és már a programozni szót is általában jól használta, bár a jelentéséről még nem sokat tudott, csak annyit, hogy ez kell ahhoz, hogy a robot működjön.
A tudós örült az új feladatnak, mert a legutóbbi sikeres küldetése óta már egy hét is eltelt és értelmesnek tűnő célok nélkül csak unatkozott.
Elkezdte tehát a munkát és rájött, hogy itt bizony nemcsak programozni kell, hanem a robotot át is kell építenie (egy kicsit).

F.1.5.1. Feladat – Kincsbegyűjtő robot építése

Szint: 

Az alaprobotra szabadon építhetsz olyan eszközt, amely segítségével a karikákat be tudja gyűjteni és magával tudja vinni (lásd a következő programozási feladatot). Elegendő megfelelő magasságba egy fixen rögzített „L” alakú rúd felszerelése, amely a robot mozgása közben éppen a karikák (kincsek) hurkainak magasságában halad.

A kincseket szimbolizáló karikákat is meg kell építened. Ehhez a *3. mellékletben* találod az építési útmutatót.

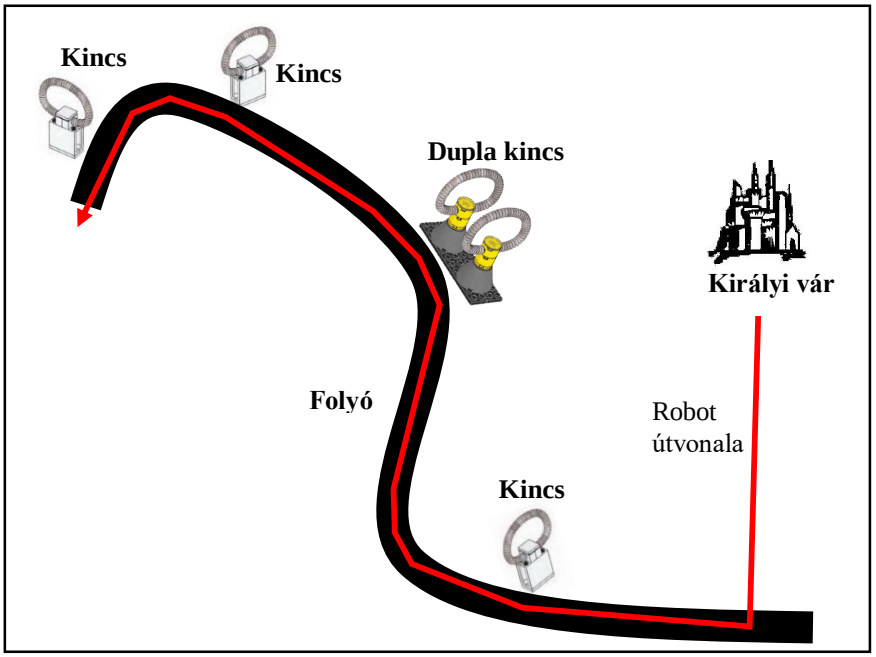
F.1.5.2. Feladat – „Kincsbegyűjtés”

Szint: 

Írj programot, amelyet végrehajtva a robot a „Királyi vár”-tól indulva összegyűjti a kincseket! Mozgása közben a folyó vonalát követi, arról nem térhet le. A kincseket szimbolizáló karikákat

magával kell vinnie, egészen a folyó „végéig”. A cél, hogy a robot valamennyi karikát összegyűjtse és elvigye a folyó „végéig”. A Királyi vár nem a folyó partján fekszik, így először a folyót kell elérnie a robotnak. A robot lehetséges útvonalát az ábra értelmezi.

Térkép



Programozási ötletek

Az útvonal követésére az F.1.4.3. feladatban ismertetett egyszenzoros útvonalkövetést lehet használni.

1.6. PROJEKT – „RAJZ A KÉPERNYŐN”

Időigény

2-3 óra

Elméleti anyag

A projekt programozási részéhez a vezérlési szerkezetek alapismerete, valamint a képernyőkezelésre vonatkozó ismeretek szükségesek. Ehhez leírás a bevezetőben megadott elektronikus könyv *6. Vezérlési szerkezetek* és *8. Képernyőkezelés* fejezeteiben szerepelnek.

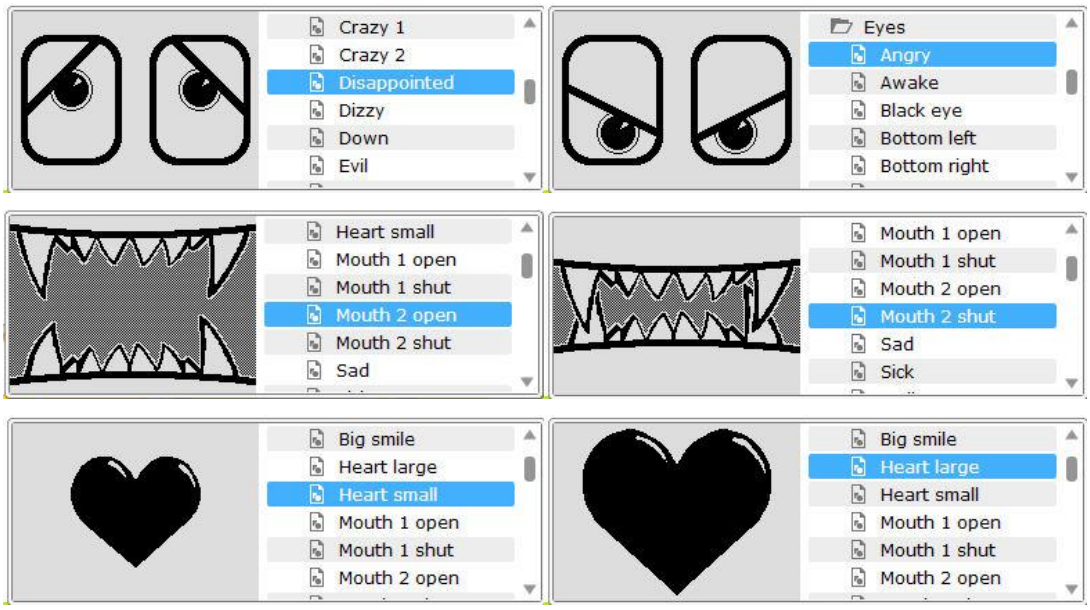
Bevezető történet

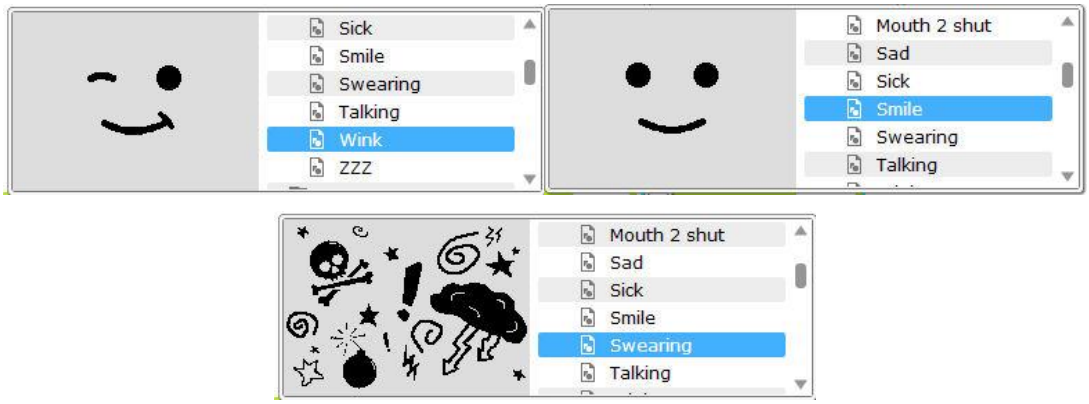
Közelgett a király születésnapja. Ezért a kiszabadított királylány elhatározta, hogy különleges ajándékkal lepi meg az öreg királyt. Meg akart tanulni programozni. Felkereste tehát a Tudóst, hogy tanítsa meg mindarra, amit tud. Persze a királylányok megszokott stílusában Ő arra gondolt, hogy mindezt egy-két nap alatt megtanulja. Hamarosan rájött, hogy tévedett.
A Tudós javaslatára, a közelgő határidő miatt csak néhány egyszerű programozói fogást tanult meg. Mivel egyébként is szeretett rajzolgatni, ezért úgy döntöttek, hogy a robot képernyőjére készít rajzokat, amivel lenyűgözheti apját és a laikus szemlélők számára bizonyíthatja hozzáértését.
A Tudós különböző feladatokat adott neki és segítette a megoldásoknál. Bár a királylány sok mindent nem értett még, de mégis sikerült szép rajzokat készítenie, és a vége felé már komolyabb feladatokat is képes volt megoldani.

F.1.6.1. Feladat – Képregény

Szint:

Írj programot, amely robot képernyőjén a rendelkezésre álló képekből egy történetet jelenít meg! A program automatikusan időzítve játssza le történetet. A megoldás során használhatsz ciklusokat. A képek, amelyekből a történetet össze kell raknod (a képek megfelelő méretben az EV3 robot képernyőjén jelennek meg):

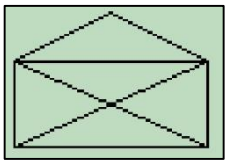




F.1.6.2. Feladat – Boríték rajz

Szint:

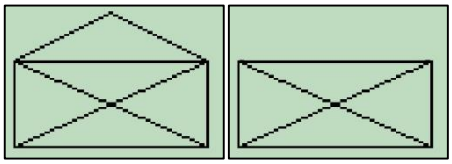
Írj programot, amelyet a robot végrehajtva egy borítékot jelenít meg a képernyőn! A boríték szélessége legyen 150 pixel, magassága (zárt állapotban) 70 pixel, nyitott állapotban 105 pixel. A program a befejezése előtt várakozzon 10 másodpercig!



F.1.6.3. Feladat – Borítékbontás animáció

Szint:

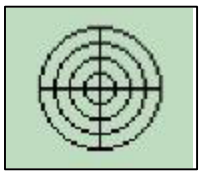
Írj programot, amelyet a robot végrehajtva az előző feladatban elkészített boríték zárt és nyitott változatát a váltogatja a képernyőn! Az ütközésérzékelő benyomott állapotában jelenjen meg a nyitott boríték, egyébként pedig a zárt boríték!



F.1.6.4. Feladat – Céltábla

Szint:

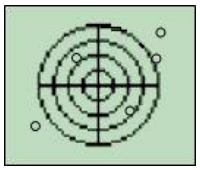
Írjon programot, amelyet végrehajtva a robot a képernyőjére rajzol négy kört, és a körökben egy plusz jelet céltábla szerűen. Lásd ábra! A körök középpontjának koordinátái (50;32) és sugaraik rendre: 20, 15, 10 és 5 pixel hosszúak. A program vége előtt a robot várakozzon 10 másodpercet!



F.1.6.5. Feladat – Célba lövés

Szint:

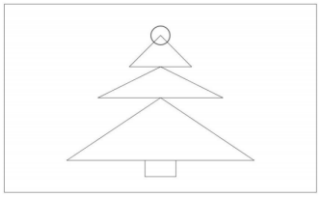
Írj programot, amelyet végrehajtva a robot véletlenszerűen sorsol két számot, amelyet egy kör középpontjának koordinátájaként értelmez, majd a sorsolt koordinátáknak megfelelő középponttal az előző feladatban rajzolt céltáblán megjelenít egy 3 pixel sugarú kört! A sorsolt számok a képernyő



mérettartományába essenek! A céltáblán megjelenő kör szemlélteti a lövés találati pontját. A program több lövést is megjeleníthet a céltáblán.

F.1.6.6. Feladat – Fenyőfa rajz

Szint:

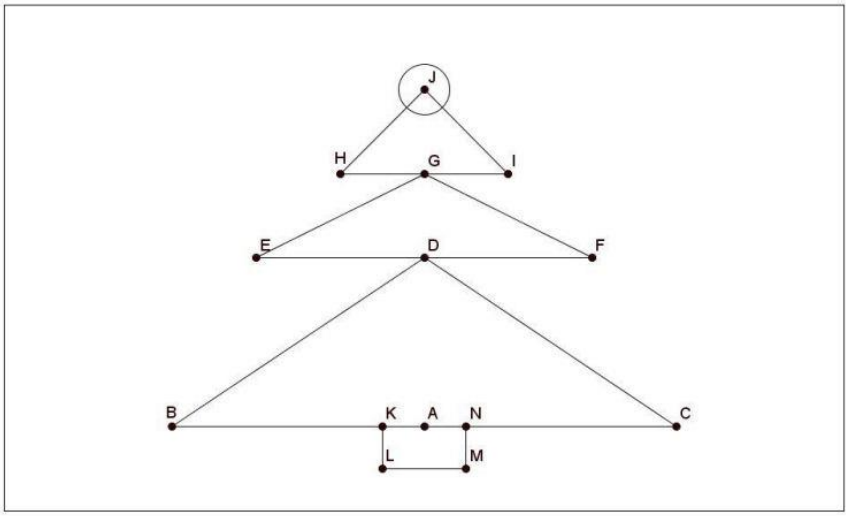


Írj programot, amelyet végrehajtva a robot a képernyőjére rajzol egy fenyőfát! A fenyőfa képe az ábrán látható (NXT robot képernyőjére méretezve). Az ábra néhány pontját megbetűztük és megadtunk néhány méretet képpontban. Az EV3-as robot képernyőjén a fenyőfa fejjel lefelé fog állni.

A rajzot a megadott méretek alapján kell elkészíteni. A program ütközésérzékelő megnyomására álljon le!

Méretek értelmezése:

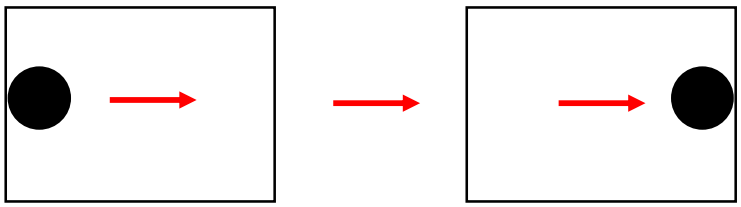
A pont koordinátái: (50;10)
BA szakasz hossza: 30
AD szakasz hossza: 20
ED szakasz hossza: 20
DG szakasz hossza: 10
HG szakasz hossza: 10
GJ szakasz hossza: 10
LM szakasz hossza: 10
LK szakasz hossza: 5
Kör sugara: 3
Kör középpontja: ***J***.
J; G; D; A pontok egy egyenesre illeszkednek, és az ábra szimmetrikus erre az egyenesre.



F.1.6.7. Feladat – Mozgó kör animáció

Szint:

Írj programot, amely egy 20 pixel sugarú kört mozgat a képernyőn vízszintesen! A kör a képernyő bal széléről induljon, függőlegesen kb. középen, folyamatosan mozogjon jobbra, majd álljon meg a képernyő jobb szélét elérve! A program ütközésérzékelő megnyomására fejezze be működését!



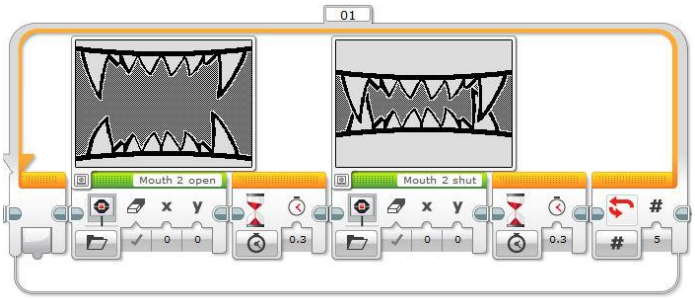
NXT robot esetén a kör nem kell, hogy kitöltött legyen.

Ha a program már működik, akkor próbáld módosítani, hogy a kör függőlegesen, vagy átlósan mozogjon!

Programozási ötletek

F.1.6.1. Képregény

A megadott képekből többféle történet is összeállítható. Ha látványosabbá szeretnéd tenni a lejátszást, akkor mozgó hatást érhetsz el, ha két képet egy ciklusba helyezed és váltakozva játszod le. Mivel a program végrehajtása gyors ezért a képek után rövid várakozást érdemes beállítani, hogy a képernyőn jobban megfigyelhetők legyenek. A bemutatott példánál 0,3 másodperc várakozási idő telik el a két kép megjelenése között és mindezt 5-ször ismétli a ciklus.

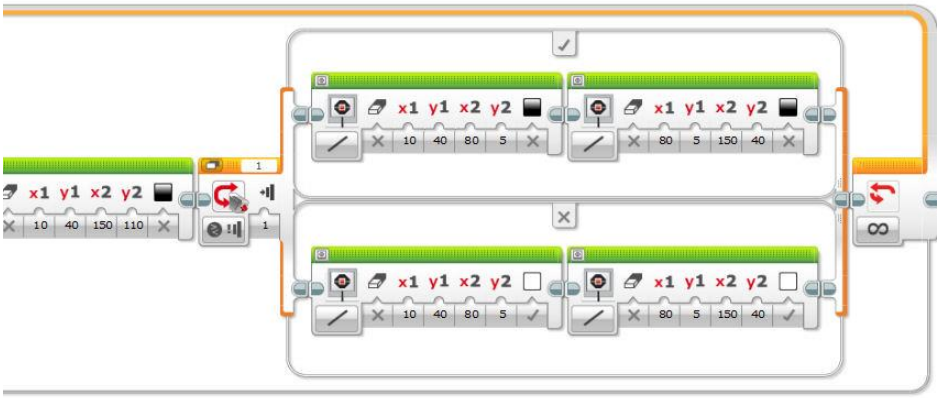


F.1.6.2. Boríték rajz

A boríték megrajzolásához ki kell választani az egyik csúcsát a sokszögnek. Legyen ez a bal alsó csúcs. Ennek koordinátái: (10; 110). Ehhez a viszonyítási ponthoz kell a megadott méretekkel számolni a többi csúcs koordinátáit. Például a jobb alsó csúcs (160; 110), míg a bal oldali felső csúcs (10; 40). A teljes rajzhoz (ha szakaszokból rakod össze), összesen 8 db vonal, tehát 8 db képernyőblokk szükséges. Ne felejtsd el a képernyőtörléseket kikapcsolni!

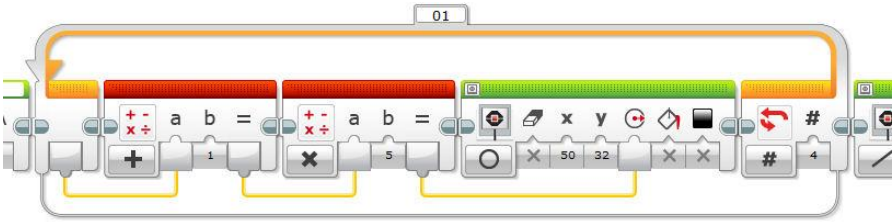
F.1.6.3. Borítékbontás animáció

Az előző feladatban elkészített ábrát és a két felső vonal nélküli ábrát kell egy elágazás két szálára elhelyezni. Az elágazást az ütközésérzékelő vezérli. Az EV3 robot esetén lehetőség van vonalak törlésére (a háttér színével történő újrarajzolásra). Ebben az esetben az elágazás két szálára elegendő a két felső vonal elhelyezése fekete színű rajzolással és háttér színű rajzolással. A többi vonalat rajzoló blokk az elágazáson kívül helyezkedik el.



F.1.6.4. Céltábla

A két szakasz kezdő és végpontjának koordinátáit ki kell számolni a megadott adatokból. A köröket rajzolhatod ciklussal is. Ekkor a ciklus számlálójánál eggyel nagyobb értéket (a számláló 0-tól indul) kell 5-tel szoroznod, hiszen a körök sugarai 5 pixelenként növekednek. A ciklust 4-szer kell lefuttatnod, mert 4 kört kell rajzolnod.



F.1.6.5. Célba lövés

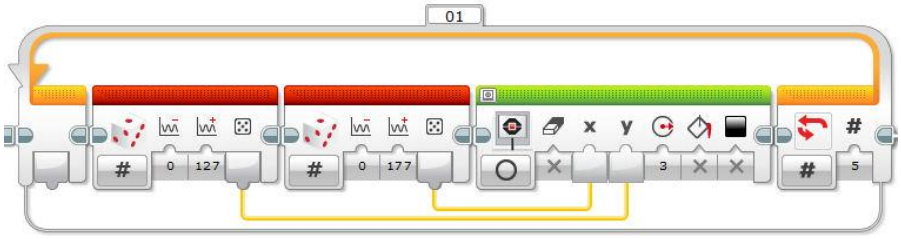
Az előző feladatban megrajzolt céltábla programot kell kiegészíteni két véletlen szám sorsolásával, amely a képernyő mérettartományába esik.

NXT robot esetén a képernyő méretei:

vízszintesen (x koordináta) 0-99; függőlegesen (y koordináta) 0-63

EV3 robot esetén a képernyő méretei:

vízszintesen (x koordináta) 0-177; függőlegesen (y koordináta) 0-127



A program 5 véletlen középpontú kört rajzol az EV3 robot képernyőjére, 3 pixeles sugárral.

F.1.6.6. Fenyőfa rajz

A fenyőfa megbetűzött pontjainak koordinátáit számold ki a megadott értékek alapján! Érdemes egy papírra leírnod a koordinátákat, majd 12 db szakaszból és egy körből megrajzolhatod az alakzatot.

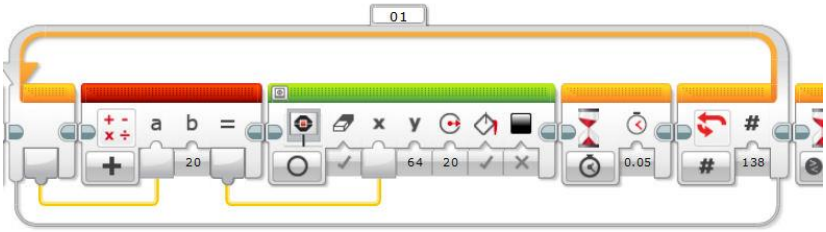
Az NXT robot és az EV3 robot esetén a koordináta-rendszer kezdőpontja nem ugyanott van a képernyőn.

NXT esetén a bal alsó sarok a (0; 0), míg EV3 esetén a bal felső.

F.1.6.6. Mozgó kör animáció

A kör középpontjának indulási koordinátái (20; 64), mivel a sugár 20 pixel. A mozgást úgy éred el, hogy az *x* koordinátát egy cikluson belül folyamatosan eggyel növeled, és minden rajzolás után letörlöd a képernyőt, így mindig csak egy kör látszik, amelynek a középpontja folyamatosan változik. Az y koordináta a program futása során nem változik. (Ha ezt is változtatod, akkor a kör nem vízszintesen fog mozogni.) A képernyő szélessége összesen 178 pixel (EV3 robot esetén), így ahhoz,

hogy a képernyő jobb szélén megálljon a kör összesen 178 – 2x20 = 138 lépés szükséges. A ciklusnak ennyiszer kell lefutnia.

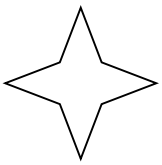


1.7. EGYSZERŰ FELADATOK

F.1.7.1.a-d Feladat – Mozgások

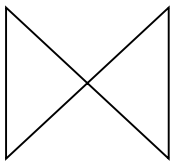
a) Szint: ⚙

Írj programot, amelyet végrehajtva a robot mozgása során az alábbi alakzatot írja le!



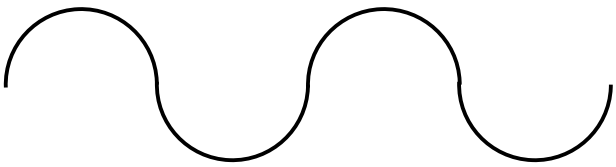
b) Szint: ⚙

Írj programot, amelyet végrehajtva a robot mozgása során az alábbi alakzatot írja le!



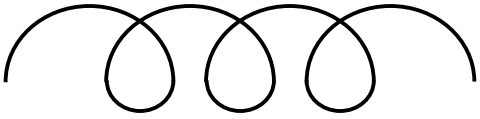
c) Szint: ⚙

Írj programot, amelyet végrehajtva a robot mozgása során az alábbi alakzatot írja le!



d) Szint: ⚙⚙

Írj programot, amelyet végrehajtva a robot mozgása során az alábbi alakzatot írja le!



F.1.7.2.a-c Feladat – Szenzorhasználat

a) Szint: 

Írj programot, amelyet végrehajtva a robot fehér felületen lévő fekete csíksor fölött halad, és a harmadik (vagy tetszőleges, de előre adott) fekete csík fölötti áthaladás után megáll!

b) Szint: 

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

c) Szint: 

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

F.1.7.3.a-c Feladat – Vezérlési szerkezetek

a) Szint: 

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

b) Szint: 

Írj programot, amelyet végrehajtva a robot addig halad előre, amíg az ultrahangszenzorával 15 cm-en belül akadályt nem észlel, ekkor az ütközésérzékelő benyomásáig tolat! Mindezt ismételje kikapcsolásig!

c) Szint: 

Írj programot, amelyet végrehajtva a robot hatszög alakú pályán mozog!

F.1.7.4.a-e Feladat – Paraméterátadás

a) Szint: 

Írj 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!

b) Szint: 

Írj 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.

c) Szint: 

Írj programot, amelyet végrehajtva a robot egy akadálytól tetszőleges távolságra áll és tolatni kezd! A tolatást addig folytassa, amíg az akadálytól kétszer akkora távolságra került, mint amennyire az induláskor volt! Ekkor álljon meg! (A feladatot változó használata nélkül oldd meg!)

d) Szint: 

Írj programot, amelyet végrehajtva a robot folyamatosan lassulva közelít egy akadályhoz! Az ultrahangszenzora által mért aktuális távolság határozza meg a robot pillanatnyi sebességét!

e) Szint: 

Írj programot, amelyet végrehajtva a robot megméri, hogy egy széles csík fölött mennyi ideig tart áthaladnia. Ha a robot egyenletes sebességgel halad a mért idő arányos lesz a csík szélességével. A kapott értéket írja a képernyőre!

F.1.7.5.a-h Feladat – Változók

a) Szint: 

Írj programot, amelyet végrehajtva a robot lassan forog körbe, miközben ultrahang szenzorával a környezetét figyeli! A robot körül két akadály van elhelyezve a robottól különböző távolságban, de 10 cm-nél távolabb és 30 cm-nél közelebb. A két akadály a robot forgása szerinti kör két átellenes pontján helyezkedik el. Miután mindkét akadályt észlelte a robot, induljon el a távolabbi felé!

b) Szint: 

Írj programot, amelyet végrehajtva a robot ütközésérzékelőjének benyomásával tudjuk egy változó tartalmát egyesével növelni. A változó értéke 0-ról induljon és folyamatosan jelenjen meg a képernyőn az értéke. Ha változó értéke elérte az 5-öt, akkor ismét nulláról induljon a számolás. A roboton található „ENTER” gomb megnyomása után a robot annyit sípoljon, amennyi a képernyőn beállított szám. Ha nulla értéknél nyomjuk meg az entert, akkor a robot ne sípoljon. Minden sípolás azonos hangokból áll, amelyek időtartama 1 másodperc és közöttük 0,5 másodperc szünet.

c) Szint: 

Írj programot, amelyet végrehajtva a robot ütközésérzékelőjének benyomásával tudjuk egy változó tartalmát egyesével növelni. A változó értéke 0-ról induljon és folyamatosan jelenjen meg a képernyőn az értéke. Ha változó értéke elérte az 5-öt, akkor ismét nulláról induljon a számolás. A roboton található „ENTER” gomb megnyomása után a robot startpozícióból indulva, egyenesen előre haladjon át annyi fekete csík fölött, amennyi a változóban beállított szám értéke. Ezután álljon meg és várjon 10 másodpercig a program vége előtt. Tehát ha a beállított szám a három, akkor a harmadik csík után álljon meg. Ha nulla értéknél nyomjuk meg az entert, akkor a robot ne induljon el. A csíkok a robot haladási irányára merőlegesek és legalább 1,5 cm szélesek.

d) Szint: 

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

e) Szint: 

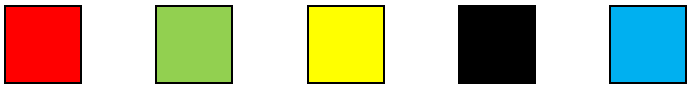
Írj programot, amelyet végrehajtva a robot egyenesen áthalad két különböző szélességű fekete vonal fölött (az alap színe fehér)! A második vonal fölött történő áthaladás után megáll, és ütközésérzékelő megnyomására a képernyőre írja (egymás alatti sorokba) a csíkok fölötti áthaladás idejét másodpercben.

f) Szint: 

Írj programot, amelyet végrehajtva a robot öt különböző színű csík fölött halad át, majd ütközésérzékelő megnyomására a képernyőre írja a csíkok színeit egymás alatti sorokba! A csíkok színe fekete, piros, kék, zöld, sárga lehet, de a sorrendjük változik, az alap színe fehér. A robot feladata, hogy az aktuális sorrendet megállapítsa, és a képernyőre írja.

Például:

A színek sorrendje:



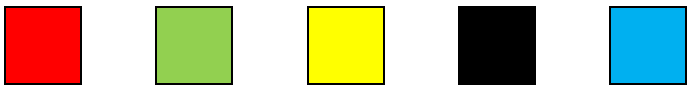
Képernyőkép az EV3 robot képernyőjén:

g) Szint: 

Írj programot, amelyet végrehajtva a robot öt különböző színű csík fölött halad át, majd ütközésérzékelő megnyomására a képernyőre írja, hogy melyik színű csík hányadikként szerepelt a sorban (egymás alatti sorokban)! A csíkok színe fekete, piros, kék, zöld, sárga lehet, de a sorrendjük változik, az alap színe fehér. A robot feladata, hogy az aktuális sorrendet beolvasva, és a képernyőre írja az adott szín pozícióját. (Az előző feladattól az eltérés, hogy most nem a színek sorrendjét kell változtatni a képernyőn, hanem egy adott színsorrend esetén az elfoglalt pozíciót, tehát a képernyőn a szöveg mindig azonos, csak a számok változnak a sorrendnek megfelelően.)

Például:

A színek sorrendje:



Képernyőkép az EV3 robot képernyőjén:

h) Szint:

A robot feladata, hogy megmérjen két fekete színű csík közötti távolságot, tehát azt, hogy mennyi ideig tart a két csík közötti áthaladás. A mérés után a robotot másik helyre helyezve, az ütközésérzékelő megnyomására a korábban mért távolságnak megfelelő időtartamig kell egyenesen haladnia. Tehát egy mért távolságnak megfelelő utat kell megtennie, de másik helyről indulva.

F.1.7.6.a-e Feladat – Képernyőkezelés

a) Szint:

Írj programot, amelyet a robot végrehajtva egyenesen mozog előre és a képernyőjére folyamatosan kiírja a fényérzékelője által mért értéket!

b) Szint:

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

c) Szint:

Írj programot, amelyet végrehajtva a robot különböző szélességű csíkok fölött halad át és megméri a csíkok fölött történő áthaladás időtartamait! A kapott értékeket írja a képernyő egymás alatti soraiba! Összesen 4 csík van. A negyedik után álljon meg!

d) Szint:

Írj programot, amelyet végrehajtva a robot egy játékszimulációt valósít meg! Két ütközésérzékelővel rendelkezik. Az egyikkel a képernyőn megjelenő 10 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.

e) Szint:

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!

F.1.7.7.a-h Feladat – Matematika, logika

a) Szint: 

Írj 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.

b) Szint: 

Írj programot, amelyet végrehajtva a robot véletlenszerűen sorsol két 0-99 közötti számot és kiírja a képernyőjére a két számot, valamint, hogy melyik a nagyobb (első vagy második), esetleg egyenlők-e!

c) Szint: 

Írj 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, …)

d) Szint: 

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

e) Szint: 

Írj programot, amelyet végrehajtva a robot egyenesen előre halad és akkor áll meg, ha elért egy fekete csíkot vagy 10 cm-en belül meglátott egy akadályt!

f) Szint: 

Egy számsorozat első tagja 5. A további tagokat úgy kapjuk, hogy az előző taghoz hozzáadunk 1-et, 2-őt, 3-mat és így tovább.

A sorozat első hét tagja: 5; 6; 8; 11; 15; 20; 26; …

Írj programot, amelyet végrehajtva a robot sorsol egy 1 és 25 közötti számot, majd a képernyőre írja a sorozat azon elemét, amennyi a sorsolt szám.

Például: A sorsolt szám 5, akkor a képernyő tartalma: 5. elem: 15

g) Szint: 

Legyen a sorozat a Fibonacci sorozat. A sorozat első két tagja: 1. A harmadik tagtól kezdve minden tagja a sorozatnak az őt megelőző két tag összege: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

Írj programot, amelyet végrehajtva a robot sorsol egy 3 és 25 közötti számot, majd a képernyőre írja a sorozat azon elemét, amennyi a sorsolt szám.

h) Szint: 

Írj programot, amely sorsol egy 0-99 közötti véletlen számot, amelyet egy sorozat kezdőértékének tekint. A sorozat 3 egymást követő egész számból áll. Tehát ha sorsolt szám a 11, akkor a sorozat 11, 12, 13. A robot háromféle tevékenységet végezhet. Ha a sorozat adott eleme osztható 3-mal, akkor egyenesen halad 1 másodpercig, ha a 3-mal osztási maradék egy, akkor jobbra fordul, ha a 3-mal osztási maradék 2, akkor balra fordul 1-1 másodpercig. A sorozat mindhárom tagjával végezze el az adott tevékenységet. Például ha a sorsolt szám a 11, akkor a robot mozgássora: balra → egyenesen → jobbra (hiszen a 11, 12, 13 számok esetén a 3-mal osztási maradékok 2, 0, 1). A robot írja ki a képernyőre a sorozat három elemét is! A program 10 másodperc várakozás után álljon le!

F.1.7.8.a-b Feladat – Többszálú program

a) Szint: 

Írj programot, amelyet a robot végrehajtva egy sokszög alakú pályán mozog!

Szint: 

Írj programot, amelyet végrehajtva a robot hangjelzést ad, ha ütközésérzékelője be van nyomva!

Szint: 

Írj programot, amelyet végrehajtva a robot egy sokszög alakú pályán halad folyamatosan! Abban az esetben, ha az ütközésérzékelőjét nyomás éri, adjon hangjelzést!

b) Szint: 

Írj programot, amelyet végrehajtva a robot egy sokszög alakú pályán halad folyamatosan! Abban az esetben, ha az ütközésérzékelőjét nyomás éri, álljon meg!

F.1.7.9.a-c Feladat – Összetett feltételek

a) Szint: 

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!

b) Szint: 

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!

c) Szint: 

Írjon programot, amelyet végrehajtva a robot egy adott kereten belül (az alap színétől eltérő színű határvonalon belül) megkeres különböző tárgyakat, és azokat a határvonalon kívülre tolja!

F.1.7.10. Feladat – Robotkommunikáció, távirányítás

Szint: 

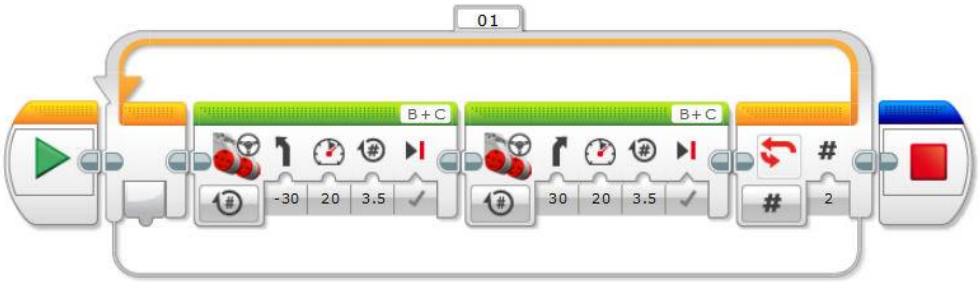
Írj programot, amely két robot kommunikációját valósítja meg! A mester robot mint távirányító működik, és a szolga robot mozgását vezérli. A mozgás irányításához a mesterre szerelt két motor tengelyelfordulási szögének előjeles értékét küldi át a szolga robotnak, amely két motorját működteti velük. A 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.)

Programozási ötletek

F.1.7.1. Mozgások

a-d) A feladatok megoldása során a fordulások vezérlésére konstansokat használhatunk, amelyek meghatározása kísérleti úton történhet. Haladó esetben ciklusokkal lehet a programot általánosabbá tenni.

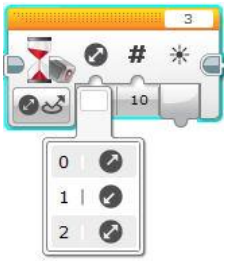
Például a **c**) feladat forráskódja:



F.1.7.2. Szenzorhasználat

a) A feladatot úgy érdemes megoldani, hogy egy cikluson belülre helyezzük az egy csík fölötti áthaladás utasításait, majd a ciklust a csíkok számának megfelelő számszor futtatjuk le. Egy csíkon történő áthaladás során két esemény következik be. Ha az alap színe fehér és a csíkok fekete színűek, akkor először fehér felületről feketére kerül a robot fényszenzora (ráhajtott a csíkra), majd feketéről fehérre változik az érzékelt felület színe (lehajtott a csíkról). Ezt a két utasítást szükséges a cikluson belül elhelyezni.

Az EV3 szoftverkörnyezetben ezt a *Wait* blokk *Colour Sensor/Change* paraméterével állíthatjuk be. Válasszuk a *Reflected Light Intensity* beállítást! Ekkor megadhatjuk, hogy a figyelt esemény a fény növekedése, csökkenése, vagy változása (mindkét irányú) legyen. A változás mértékét számszerűen lehet megadni. Ha a változást (iránytól független) választjuk, akkor a programunk működni fog fekete felületet határoló fehér szegély esetén is.



b) Első lépésben érdemes megmérni az asztal felületén és az asztal szélén visszaadott fényszenzor értékeket. Az asztal szélén úgy mérjünk, hogy a szenzor „lógjon le" az asztal felületéről. Ha különböző értékeket mér a robot, akkor a program egyszerűen elkészíthető. A két mért érték számtani közepe lehet az a határ, ami megkülönbözteti az asztal felületét a szélétől. A robot tehát elindul (*On*). A *Wait* modullal szabályozzuk a várakozást a fényszenzornál megállapított határértékig. Ha a feltétel igazzá válik, akkor a robot megáll. Mérés nélkül a *Change* módot lehet használni.

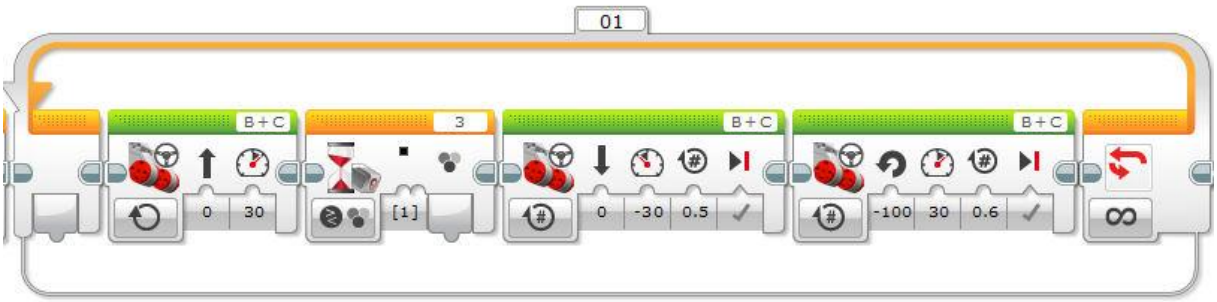
c) A program a „Királylány szabadítás" projektnél bemutatott programozási ötletek alapján megoldható.

F.1.7.3. Vezérlési szerkezetek

a-b) A feladatok megoldása során arra kell ügyelni, hogy ha nem ismerjük a robot mozgásának időtartamát, tehát a mozgást valamilyen szenzorral mért érték fogja megváltoztatni, akkor a motort *On* állapotba kell kapcsolni.

Nem érdemes a robotot gyorsan mozgatni.

Az **a)** feladat kör alakú terület esetén, fekete alapszínnel és fehér szegéllyel a robot szumó versenyek alapprogramja:



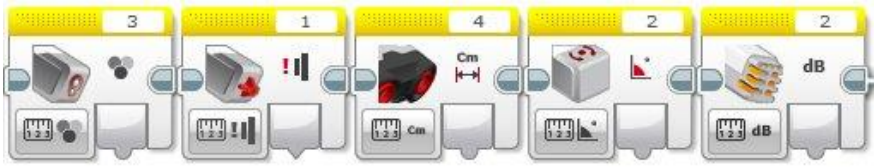
c) Az F.1.7.1. feladatban kitűzött mozgásokhoz hasonlóan oldható meg, de az ismétlődő elemeket ciklusba kell helyezni. A hatszög esetén a cikluson belül két motorvezérlő ikon szerepel. Az első az egyenes haladást biztosítja, míg a második a fordulást. A ciklus ezt ismétli hatszor. A fordulás szöge tapasztalati úton határozható meg, vagy giroszenzor használatával mérhető.

F.1.7.4. Paraméterátadás

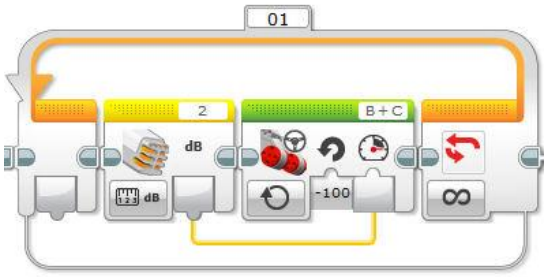
A szenzorok használatának van egy olyan módja is, amelynél a mért értéket használjuk fel egy másik modul valamely paramétereként. Ebben az esetben a program utasításainak végrehajtása nem vár a

szenzor által mért érték valamely határának elérésére, hanem csupán kiolvassa az értéket, és lép a következő utasításra. Ha a mért értéket nem használjuk fel, akkor a modul használata felesleges, mert nem történik semmi „látható” a végrehajtás során.

A legfontosabb szenzorblokkokat mutatja az alábbi ábra. Az ikonok szegélye sárga és a *Sensor* programcsoportban találhatók.



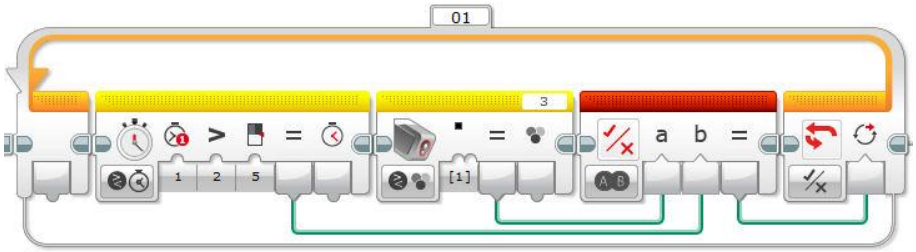
a) A robotra egy hangérzékelő szenzort kell csatlakoztatni, amely az NXT robotkészlet tartozéka (pl.: az alaprobot 2-es bemeneti portjára csatlakoztatható).



A hangszenzor mért értékét használjuk a motor sebességparamétereként. Az összekötő kábel színe sárga, ez arra utal, hogy szám típusú értéket továbbítunk rajta.

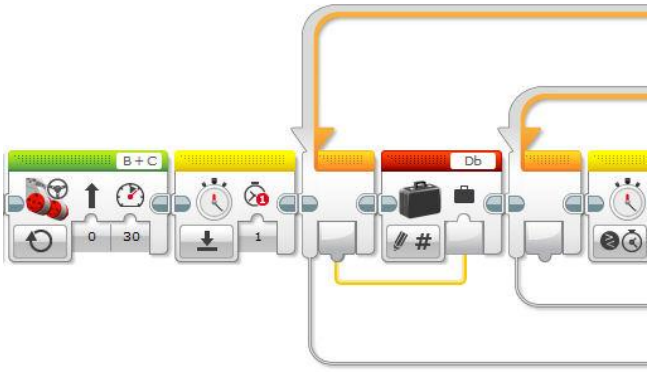
b) A megoldásban a problémát az okozza, hogy két dolgot kell egyszerre csinálnia a robotnak. Egyrészt figyelni az időre, hiszen 5 másodperc elteltével meg kell állnia, másrészt a felületen lévő csíkokat is számolnia kell. A megoldásra két különböző lehetőséget mutatunk.

Ha egyetlen programszálat szeretnénk használni, akkor használhatjuk a korábban már bemutatott ötletet, hogy a fekete csíkon történő áthaladás során két esemény következik be. Az első, hogy fehérről-feketére változik a szenzor által mért szín, a második, hogy feketéről-fehérre változik a szín. A két esemény után lehet eggyel növelni a csíkszámlálás aktuális eredményét. A *Wait* modul használata viszont nem megfelelő a csíkok figyelésére, hiszen amíg a program várakozik, addig nem figyeli a stoppert sem, tehát nem fogja érzékelni az 5 másodperc leteltét. Ezért érdemes írni a *Wait* blokkhoz hasonló utasítássort egy ciklussal, de itt a kilépés már lehet összetett, például egy VAGY kapcsolattal vezérelve.



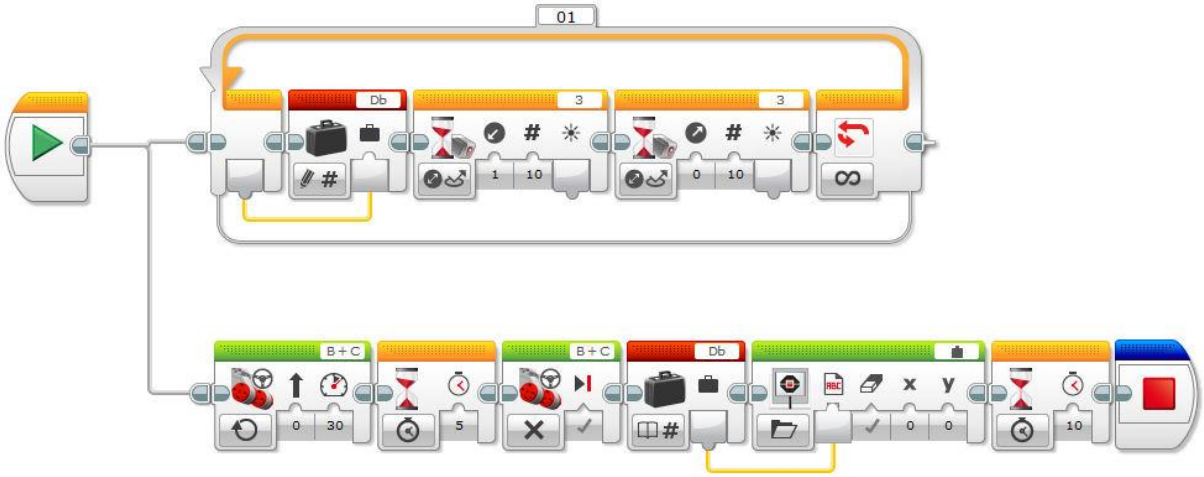
A ciklusból akkor lép ki a program, ha letelt az 5 másodperc vagy fekete színt érzékel a szenzor. Hasonló utasítássor helyettesítheti a második esemény figyelését, amikor feketéről-fehérre érkezik a robot (a színszenzor értékét kell fehérre állítani).

A két ciklust elhelyezve egy nagyobb ciklusban, ami az 5 másodperc elteltéig fut, már kész is van a program lényegi része. Természetesen a stoppert le kell nullázni a ciklusok előtt, és a motorokat is be kell kapcsolni.

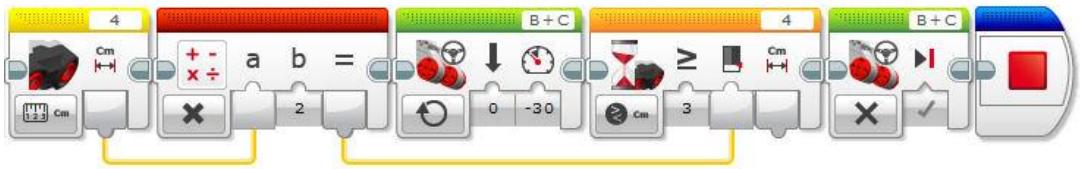


A csíkok számlálásához felhasználhatjuk a ciklusváltozót, amit folyamatosan eltárolunk egy saját változóban, így a kilépve a ciklusból ki tudjuk íratni a képernyőre.

A feladat megoldható két programszál használatával is. Az egyik szálon figyeljük a csíkokat és számoljuk őket, míg a másik szálon mérjük az időt és vezérlejük a motorokat.

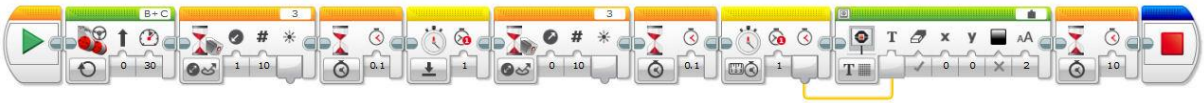


c) A program alapötlete, hogy induláskor a robot megméri ultrahangos távolságérzékelővel az akadálytól való távolságát, ezt az értéket szorozza 2-vel, majd mindaddig tolat, amíg az aktuálisan mért érték nagyobb nem lesz, mint a kezdeti érték kétszerese. Nem érdemes az egyenlőséget használni, mert a mért értékek nem pontosak és csak bizonyos időközönként jutnak el a programhoz. Ha éppen nem az az érték jut el, ami a kezdeti kétszerese, akkor robot nem fog megállni. (Pl.: kezdetben 30 cm-t mért a robot. 60 cm-nél kellene megállnia, de a mérési adatok közül az 59 cm és a 61 cm jut el a programhoz, így nem fog megállni.)



d) Az a) feladattal analóg feladat, csak hagérzékelő helyett ultrahangos távolságérzékelőt kell használni.

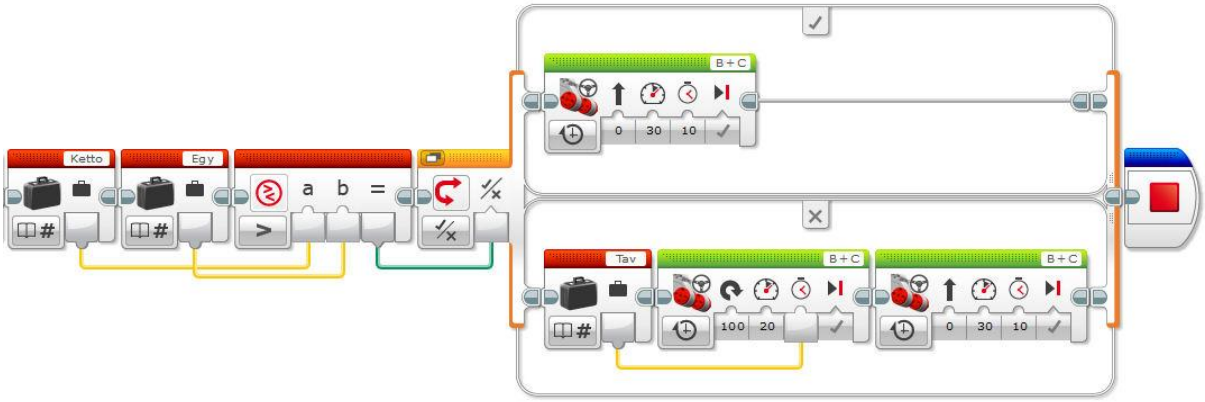
e) A mérésre a stoppert használjuk, amelyet a csík elérésekor lenullázunk, majd a csíkon áthaladva a mért értéket kiíratjuk a képernyőre.



F.1.7.5. Változók

a) A robotnak mindkét akadálytól mért távolságát meg kell mérnie és a kapott adatokat meg is kell jegyeznie, hiszen összehasonlítani őket csak a második mérés után tudja. Ezen kívül, ha az első mérés eredménye a nagyobb, akkor vissza is kell fordulnia az első akadályhoz, így azt is tudnia kell, hogy mennyit forduljon vissza. Ehhez a két akadály észlelése közötti időt mérheti és a visszafordításnál, ha ugyanakkora sebességgel fordul visszafelé is, akkor meg fogja találni az első akadályt. Mindehhez három változót hozunk létre: *Egy*, *Ketto* (a távolságok adatok tárolására), *Tav* (a két akadály között eltelt idő tárolására.

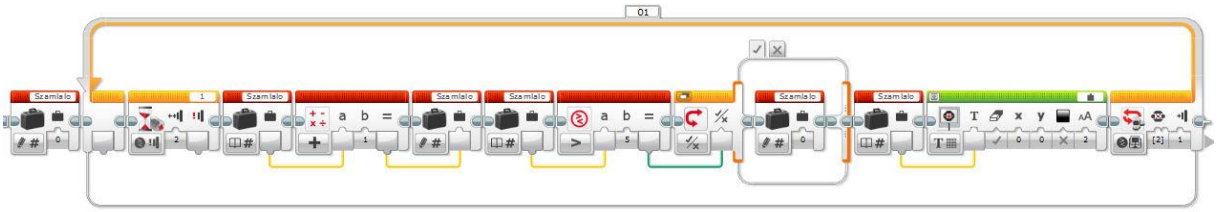
A távolabbi akadály felé történő mozgást egy elágazással oldhatjuk meg. Attól függően, hogy a két változóban tárolt távolságértékek közül melyik a nagyobb.



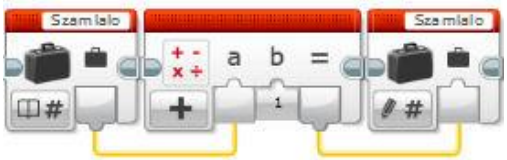
b) A program a megszámlálás alapalgoritmusa. Az elve az, hogy egy nulla kezdő értékű változó tartalmát mindig növeljük eggyel, ha bekövetkezik egy esemény.

A számlálást és a kiíratást a program egyik szálán végezzük, míg az „ENTER” gomb figyelését és a hanglejátszást a másikon. A számlálásnál, ha a változó értéke elérte az 5-öt, akkor újra lenullázzuk.

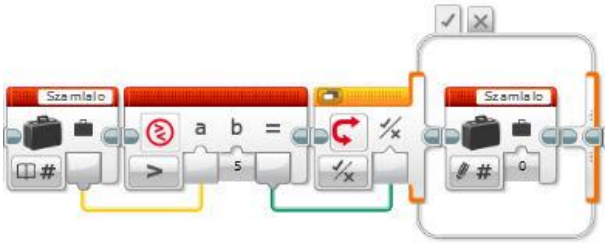
A számlálást végző programszál:



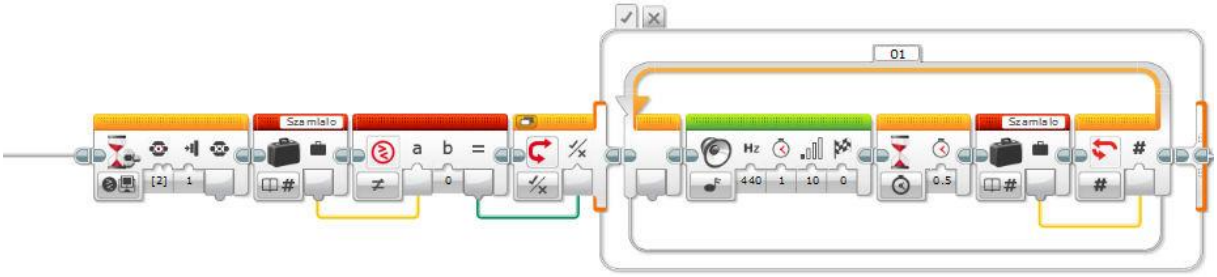
A számlálást végző utasítássor:



A változó nullázását végző utasítássor:



A hanglejátszó programszál:

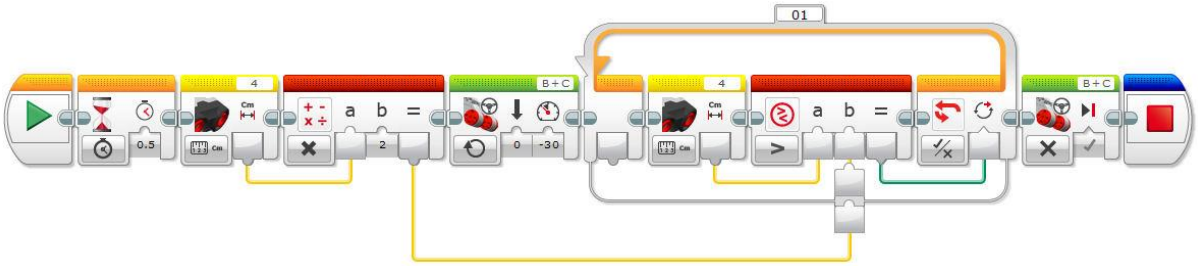


c) A feladat megoldásánál a hanglejátszó programszál helyett a mozgásvezérlést kell használni.

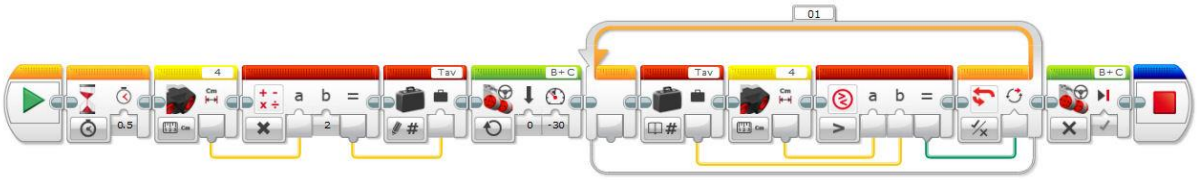
d) A feladat megegyezik az F1.7.4. c) feladatával, de most változók használatával érdemes megoldani. Induláskor a távolságértéket eltároljuk egy változóban, majd ennek kétszereséig tolat a robot.

A mérését a mozgás megkezdésekor folyamatosan kell végezni, tehát egy ciklusba kell helyezni a megfelelő blokkot. A kezdeti mérés viszont még a cikluson kívül történt. A cikluson kívüli értéknek a ciklusba történő bejuttatása történhet a paraméter (kábel) közvetlen becsatolásával is, de egy változón keresztül elegánsabb programozás-technikailag.

A közvetlen becsatlakoztatásra példa:



A változón keresztüli használatra példa:



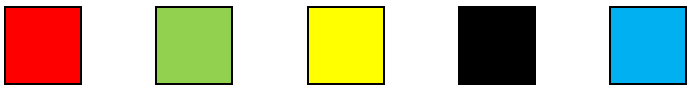
e) A csíkok szélességét a stopperrel érdemes mérni. Mivel a két mérési eredményt csak az ütközésérzékelő megnyomása után kell a képernyőre írni, ezért tárolni kell őket. Létrehozunk tehát két változót a mérési adatok tárolására (Egy, Ketto), majd az ütközésérzékelő megnyomása után a változókban tárolt értékeket írjuk a képernyőre.

Az első mérés és adattárolás forráskódja:

f) Az előző feladathoz képest a különbség, hogy most színeket kell felismerni és tárolni, ráadásul öt különbözőt. A tároláshoz érdemes tömböt létrehozni. Mivel a tömbben csak számokat vagy logikai értékeket tárolhatunk, ezért minden színnek megfeleltetünk egy számadatot, és ha a robot az adott színt érzékelte, akkor az ennek megfelelő számot helyezzük a tömbbe. A színeknek a szoftverben már van egy-egy számkódja.

A színek számkódjai a szoftverben: fekete → 1; kék → 2; zöld → 3; sárga → 4; piros → 5.

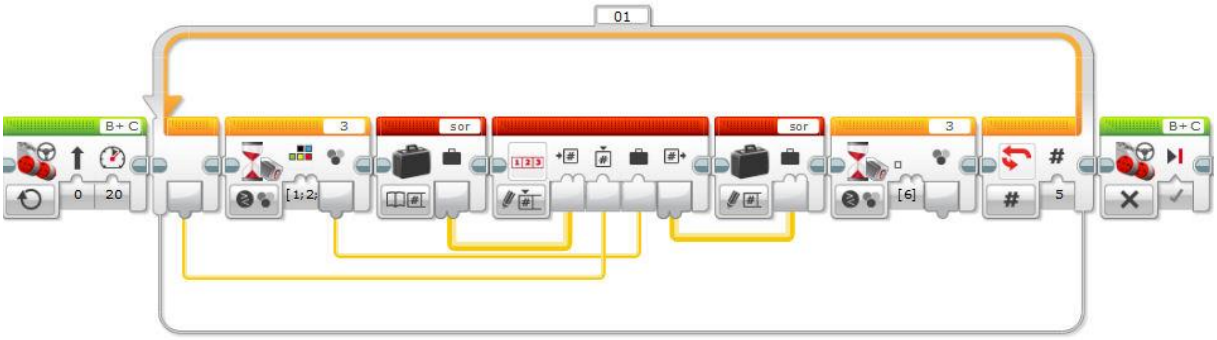
A színek sorrendje:



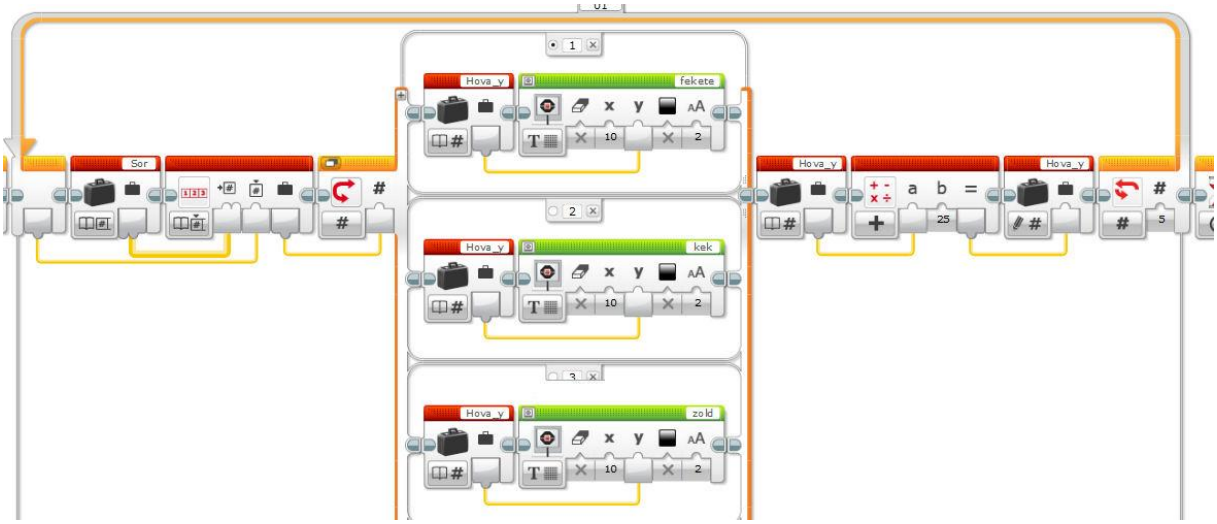
A tömbben tárolt adatok:

A tömbindex	Tárolt érték
0	5
1	3
2	4
3	1
4	2

Létrehozunk egy 5 elemű tömböt, és a színkódokat tároljuk rendre az egyes tömbelemekben. A tömb indexeként használhatjuk a ciklusváltozót.



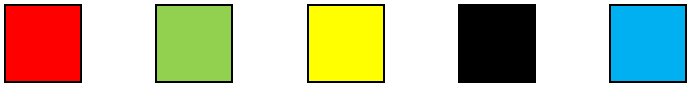
A kiíratáshoz numerikus tömböt használhatunk. Létrehozunk egy változót (*Hova_y*), amely értékét pl. 25 pixellel növeljük minden kiíratás után, így elérhetjük, hogy a színek nevei új sorokba kerüljenek.



g) Az előző feladathoz képest a különbség, hogy most színek sorrendje fix a képernyőn, csak a pozíciót jelző számok változnak a szín neve után. A tárolás algoritmusának egyszerűsítéséhez nem a szín kódját tároljuk, hanem azt, hogy a szín hányadikként fordult elő, mégpedig abban a tömbelemben, amelyet a szín kódja azonosít.

Például:

A színek számkódjai a szoftverben: fekete → 1; kék → 2; zöld → 3; sárga → 4; piros → 5.
A színek sorrendje:

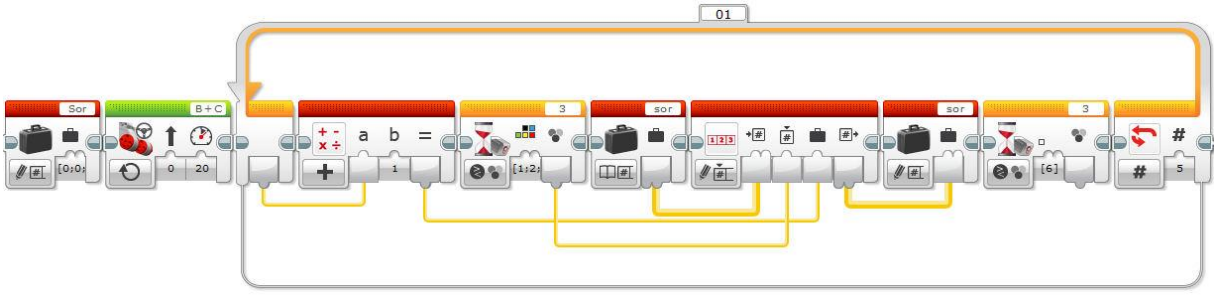


A tömbben tárolt adatok:

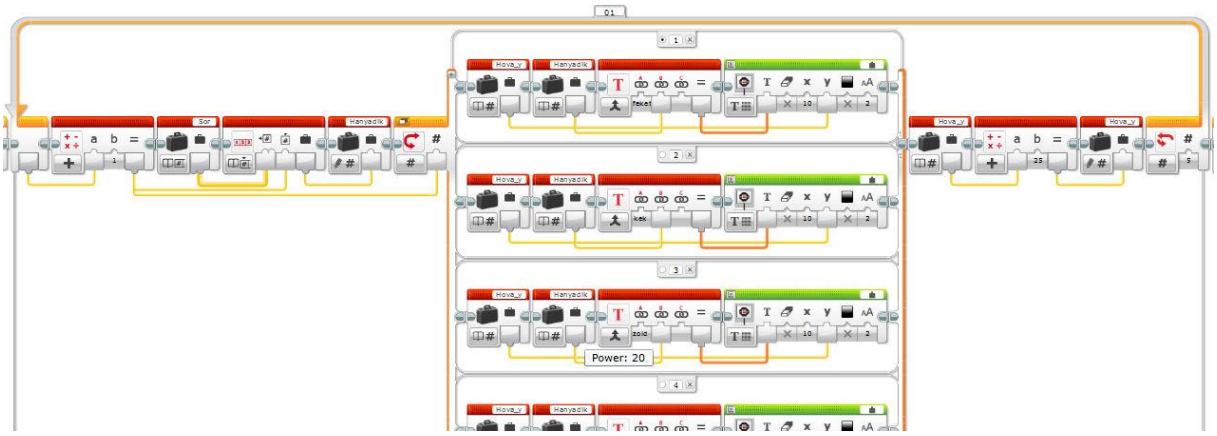
A tömbindex	Tárolt érték
0	Nem használjuk
1	4
2	5
3	2
4	3
5	1

Létrehozunk egy 6 elemű tömböt, de a tömb nulla indexű elemét nem használjuk. Az 1-es indexű elem azért lesz 4, mert az 1-es színkódhoz a fekete szín tartozik és az a 4. a sorban. … A 4-es indexű elem azért lesz 3, mert a 4-es színkódhoz a sárga szín tartozik és az a 3. a sorban. …

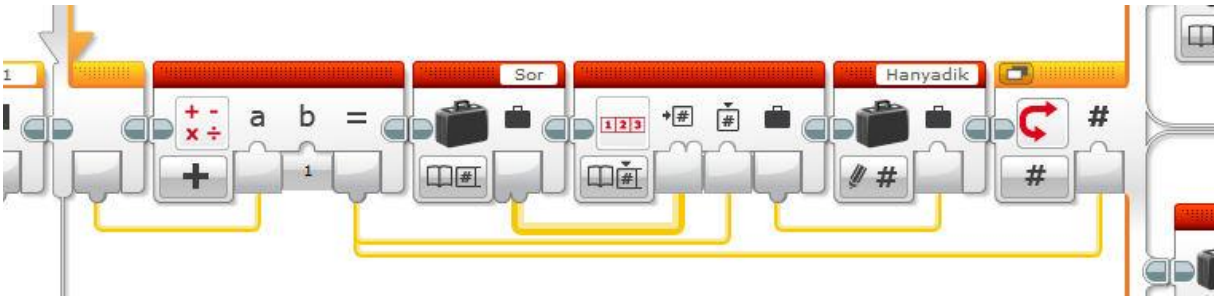
A tárolást és a kiíratást így sokkal egyszerűbb megoldani (rövidebb forráskód tartozik a megoldáshoz).



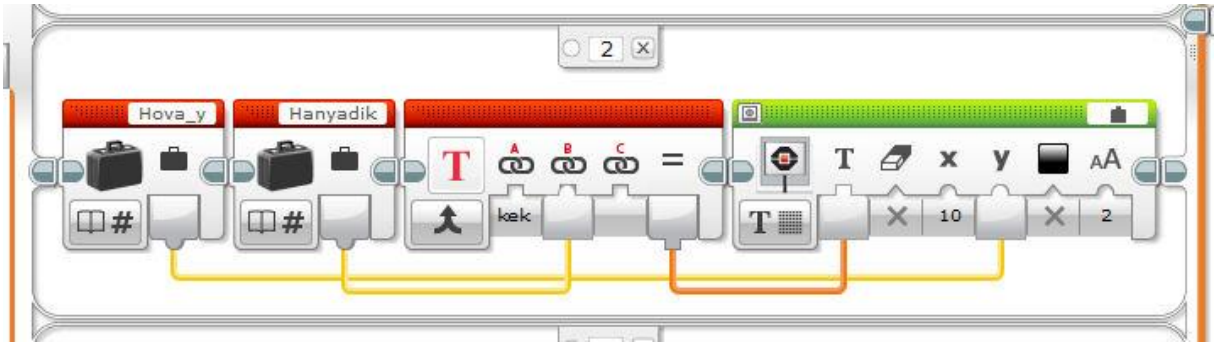
A kiíratást numerikus elágazással készítjük el. Ennek egy részlete:



Az elágazás vezérlő feltétel paraméterezése:



Egy szín sorrendjének kiírása:

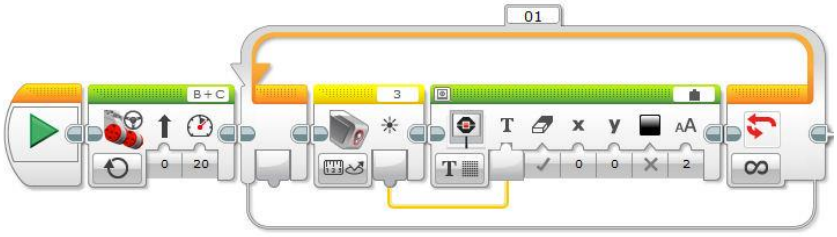


A *Hova_y* változó a képernyő pozíciót határozza meg. Minden kiíratás után 25-tel nő az értéke, így a következő kiírás már 25 pixellel lejjebb történik (EV3 robot esetén).

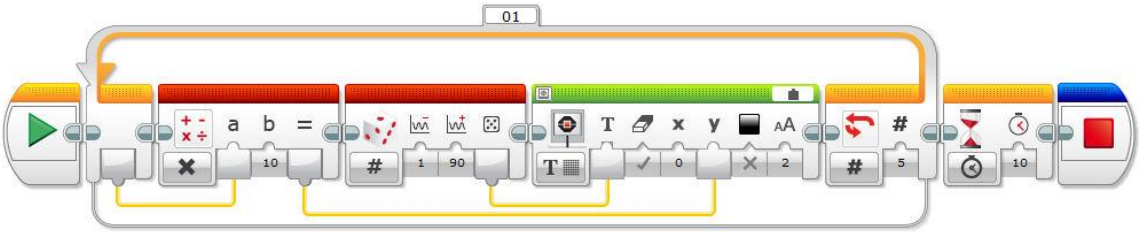
h) A program elkészítéséhez létrehozunk egy változót, amelyben majd eltároljuk a stopperrel mért időtartamot, ami a két csík közötti áthaladás során eltelik. Amire ügyelni kell, hogy a stoppert bekapcsolni (nullázni) az első fekete csíkról történő leéréskor kell bekapcsolni, és a második fekete csík elérésekor kell a mért értéket tárolni. Ütközésérzékelő megnyomására a kezdeti sebességgel megegyező sebességgel, a változóban tárolt időtartamig kell a motort ismét működtetni.

F.1.7.6. Képernyőkezelés

a) A feladat megoldása egy egyszerű paraméterátadásra épül. A fény/szín érzékelő szenzor blokkot kell egy ciklusba helyezni és a mért értéket a képernyőre írni (a képernyőtörlést be kell kapcsolni).



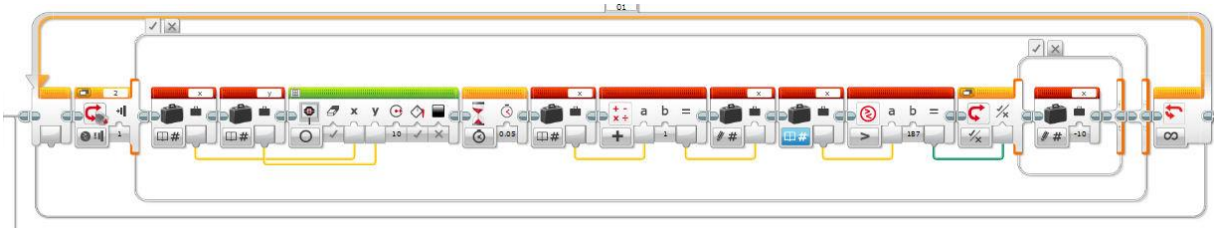
b) A sorsolt számok képernyőre íratása egyszerű. Az egymás alatti sorokba megkötés jelent újdonságot. A megoldáshoz használjuk fel a ciklus beépített számlálóját. A ciklusunk ötször fut le, közben a ciklusváltozó értéke 0-ról 4-re növekszik. A ciklusváltozó értékét szorozzuk 10-zel, akkor 0-tól 40-ig 10-esével növekedő sorozatot kapunk. Ezek a számok alkalmasak a képernyőre írt szöveg függőleges (*y*) koordinátáinak megadására.



c) A feladat az F.1.7.1.5. e) részének az előző feladatnak a kombinációja.

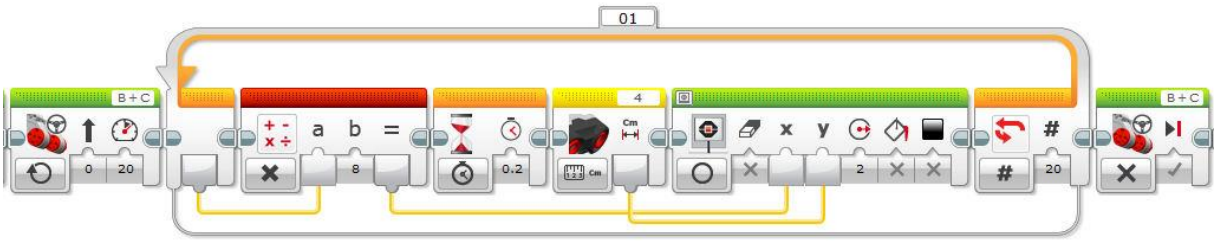
d) A második ütközésérzékelő csatlakoztassuk a 2-es portra. Két változót hozunk létre, amely a kör középpontjának *x* illetve *y* koordinátája lesz. A mozgatás algoritmusa hasonlít a korábban megírt programra. Az ütközésérzékelő benyomott állapotánál a változó értékét folyamatosan növeljük eggyel, így a középpont koordinátája folyamatosan növekszik. A túl gyors mozgás miatt egy lassítást érdemes beépíteni a programba (a példánál ez 0,05 másodperc). Az x illetve y koordináta változtatását két párhuzamos programszálon végezzük.

Az *x* koordinátát változtató programszál (az *y* koordinátára hasonlóan):

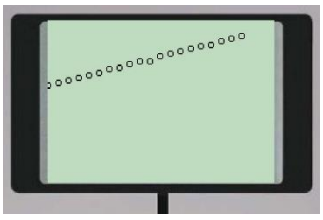


Ha az *x* koordináta elérte a 187-et (177 a képernyő széle +10 a kör sugara), akkor visszaállítjuk az értékét –10-re (10 a kör sugara). EV3 robotnál látványosabb a program, ha a kör fekete színnel kitöltött.

e) Az ábrázolást valós időben végezzük, tehát a mért értékeket nem tároljuk el. A 0,2 másodperces mintavételhez (kb. 4 másodpercre tervezve a mozgást), 20 mintát veszünk. Ebből következően a vízszintes tengelyen 8 pixel fog megfelelni az egymást követő mintavételeknek. A pontok helyett 2 pixel sugarú köröket rajzolunk a képernyőre.



A képen a képernyőkép látható egyenletesen mozgó robot esetén, álló akadály felé. A koordináta rendszerek tengelyei nem szerepelnek az ábrán, illetve EV3 robot esetén az origó a képernyő bal felső sarka (NXT robot esetén a bal alsó).

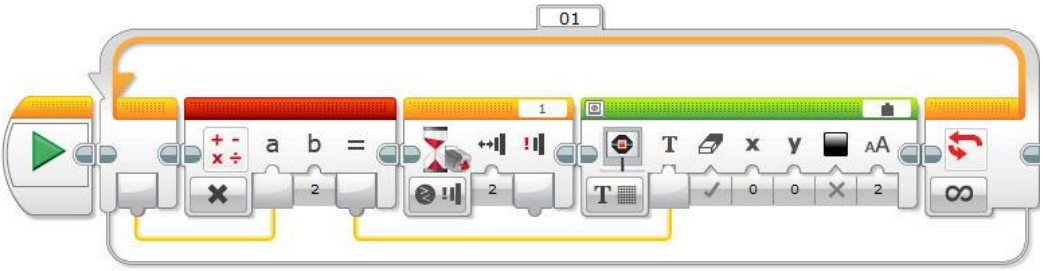


A program továbbfejlesztéseként érdemes a mintavételek számát növelni. Ebben az esetben a robot mozoghat egy adott távolság eléréséig. Természetesen ilyenkor nehezebb meghatározni a mintavételek számát, mivel a robot sebességétől függ, hogy mikor fogja elérni a beállított határtávolságot. Pontosabb és könnyebb a megoldás, ha a mérési adatokat tároljuk, például egy fájlba írjuk és a robot megállása után a fájlban szereplő adatokat jelenítjük meg a képernyőn. A fájlba írás során számolhatjuk egy változóban a mintavételek számát, így az adatok

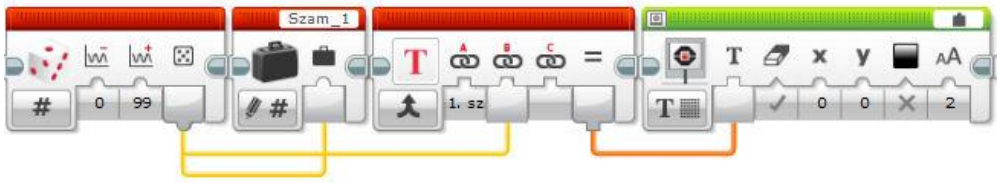
számával elosztva a képernyő szélességét (EV3 robotnál 178, NXT robotnál 100), megkapjuk a *x* tengelyen vett egységet.

F.1.7.7. Matematika, logika

a) A feladat megoldása során felhasználjuk az egyesével növekvő sorozat megjelenítésére írt programot (lásd: F.1.7.5. a) feladat), de most a ciklusváltozó értékét szorozzuk kettővel a képernyőre íratás előtt.

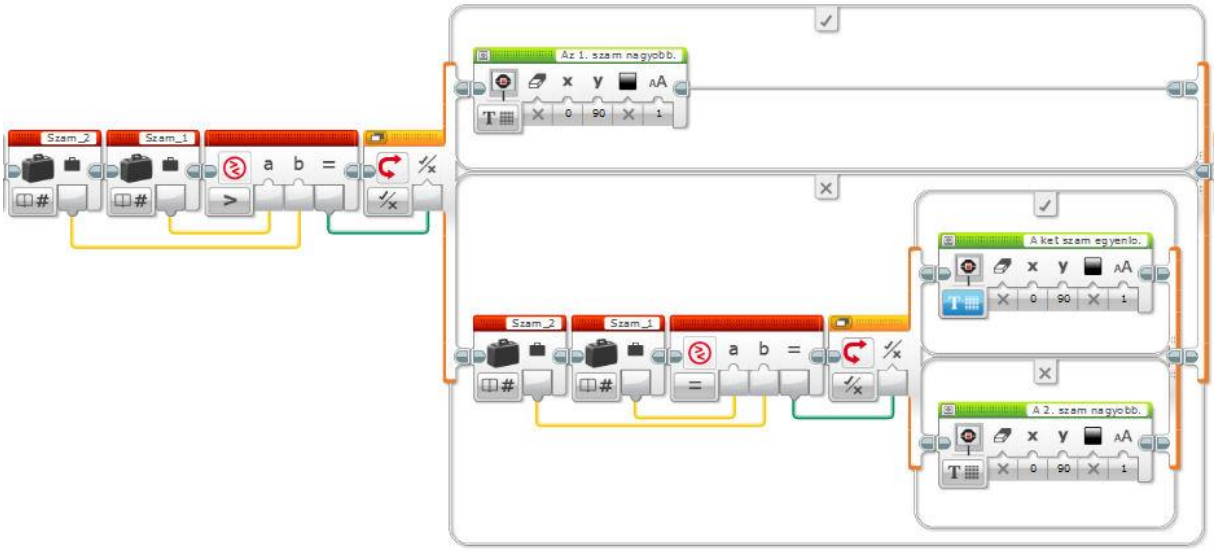


b) A két számot sorsoljuk, eltároljuk egy változóban és kiírjuk a képernyőre.



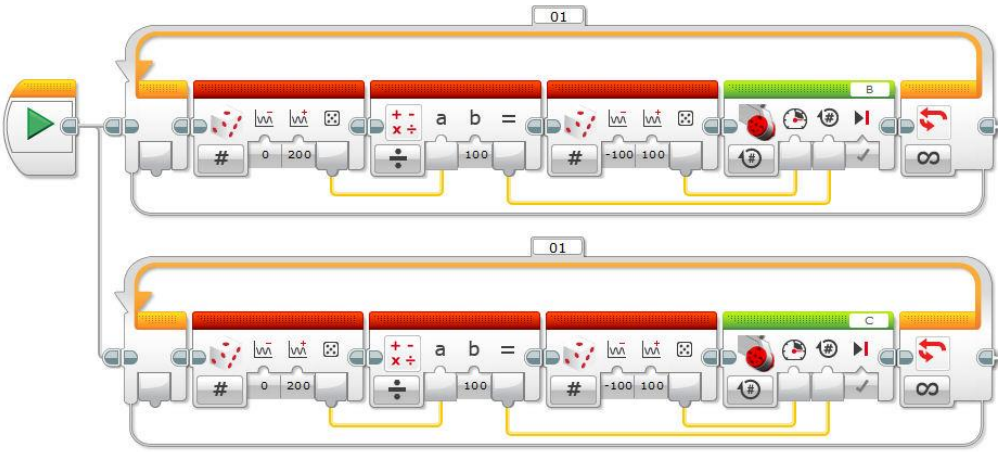
A második szám esetén hasonlóan járunk el, de a kiíratás *y* koordinátája pl.: 30. A szöveget („Az 1. szam:”) és a sorsolt számot összefűzzük a *Data Operation/Text* blokk segítségével, így egységesen tudjuk kezelni a szöveget.

Az összehasonlításhoz két elágazás szükséges, mert összesen három eredmény lehet: az 1. szám a nagyobb, a 2. szám a nagyobb, vagy egyenlők.

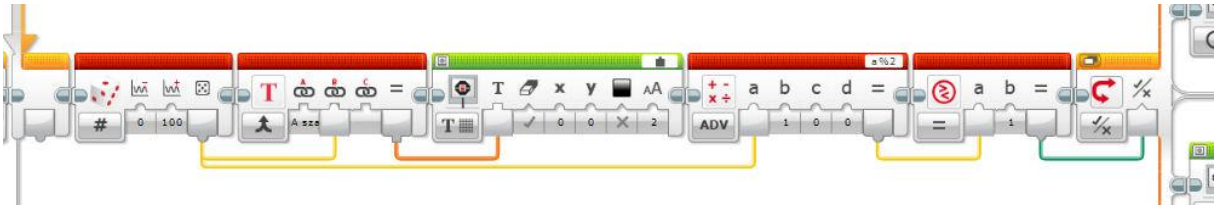


c) A programhoz használjuk a *Move Tank* motort. Itt a két motor sebességét külön tudjuk szabályozni. A motorok forgási irányát a sebesség előjele határozza meg. Tehát két –100 és +100

közötti értéket sorsoltatva lehet a mozgást szabályozni. A működési módot idővel vezérleve egy harmadik véletlen szám adhatja meg az időtartamot. Mivel a véletlen szám sorsoló csak egész számot tud előállítani, ezért ha 0-200 közötti tartományból sorsolunk és a kapott értéket osztjuk 100-zal, akkor 0-2 közötti számokat kapunk, de két tizedes jegy pontossággal. Ha mindezt egy végtelen ciklusba helyezzük, akkor készen is van a véletlen mozgás. Ha a *Move Tank* motor helyett *Large Motor* blokkot használunk, akkor a két motor működési idejét külön-külön tudjuk szabályozni. Ekkor a két motor vezérlését külön programszálon érdemes elhelyezni, mert a hosszabb működési idejű motor mozgásának letelte után sorsol a program új értéket, így a másik motor addig áll.

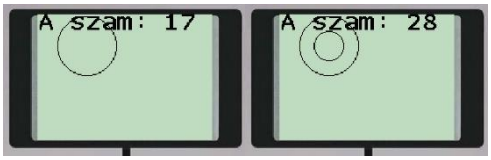


d) A szám sorsolása után el kell dönteni, hogy az páros vagy páratlan. A szoftverben rendelkezésre áll egy matematikai modul, amely az osztási maradékot képes visszaadni. Ez a *Data Operation/Advenced* blokkján keresztül érhető el. Az ikon jobb felső sarkában szereplő beviteli mezőre kattintva a legördülő listából lehet választani a *Modulo* (%) funkciót. Egy szám esetén a 2-vel történő osztás utáni maradék dönti el, hogy a szám páros-e. Ha a maradék 0, akkor páros, ha 1, akkor páratlan.



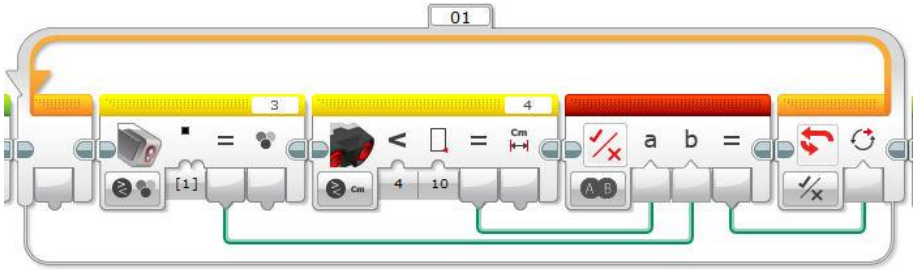
A körök rajzolása már egyszerű, egy elágazás két szálára helyezve a megfelelően paraméterezett *Display* blokkokat.

A képernyőkép:

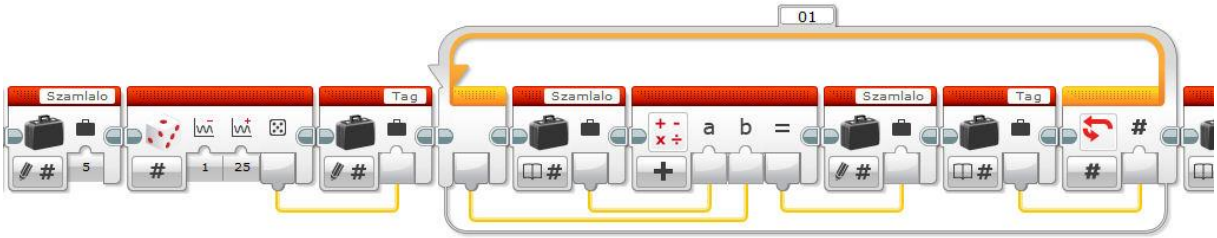


e) A program az összetett feltételek használatára mutat példát. Nem használhatjuk a *Wait* blokkot, mivel itt csak egy feltétel állítható be. A megoldás a szenzorok mért értékeinek figyelése egy

ciklusban. A két szenzort *Compare* módban használjuk és beállítjuk a feltételeket. A blokk akkor ad vissza igaz értéket, ha a feltétel teljesült, de eltérően a *Wait* blokktól nem áll meg a végrehajtás a blokknál, hanem a szenzorérték kiolvasása után a következő utasítás végrehajtására lép a program. A két logikai értéket vagy kapcsolattal összekötve kapjuk a ciklus kilépési feltételét.

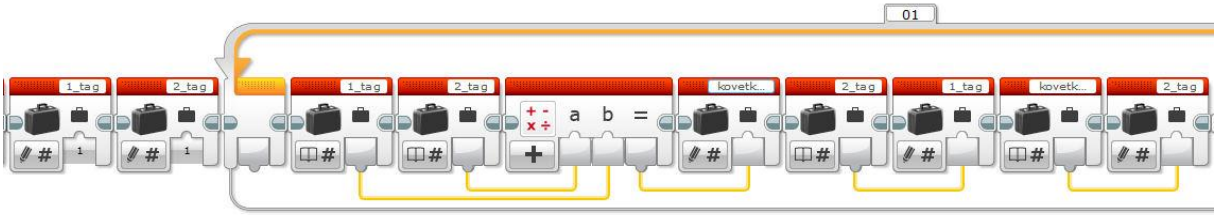


f) A program lényegi részét mutatja az alábbi forráskód részlet:

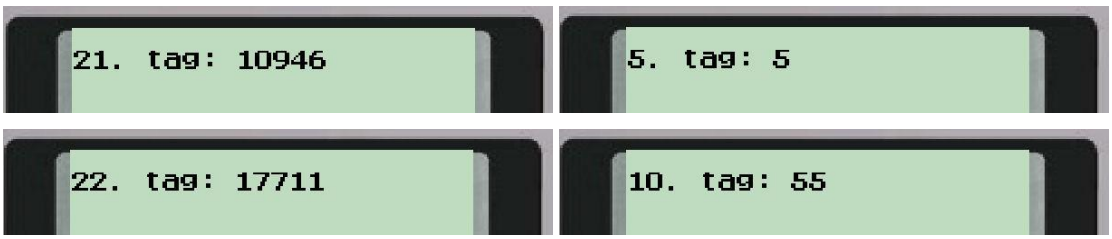


Az eredmények kiíratása hiányzik még.

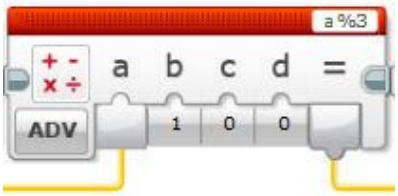
g) A program lényegi része:



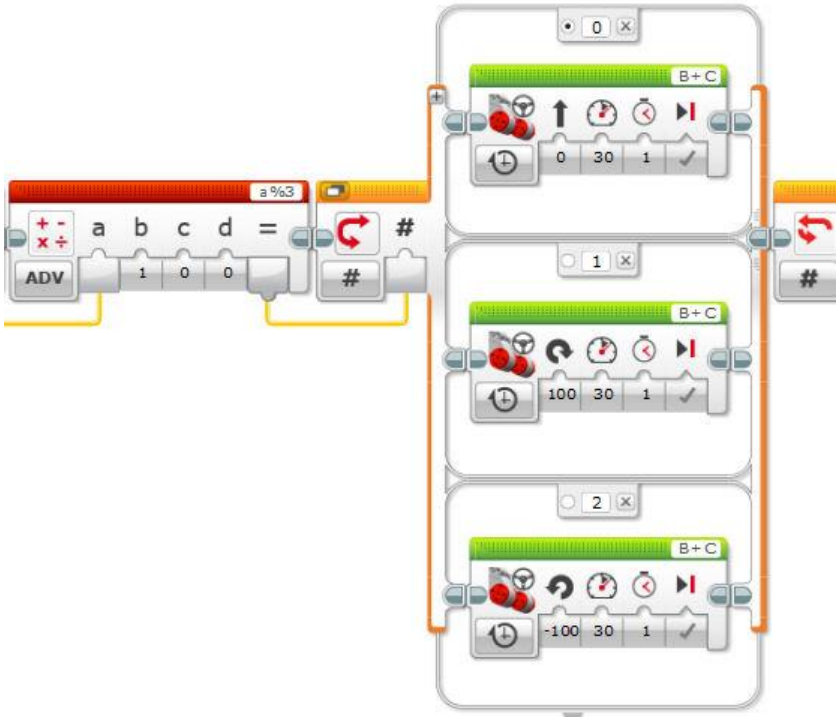
A ciklus a kisorsolt számnál kettővel kisebb értékig fut (hiszen a sorozat első két tagja adott).
A képernyőkép:



h) A matematikai egész osztás keletkező osztási maradékot a *Data Operations/Advanced* blokk jobb felső sarkának beviteli mezőjén található *Modulo* függvénnyel tudjuk képezni. Tehát ha a sorsolt szám a 11, akkor 11%3 művelet a 11 hárommal osztás utáni maradékát, a 2-t adja vissza eredményül. Ha a sorsolt számot az a jelű bemenetre kötöttük, akkor ez a programban a következőképpen néz ki:

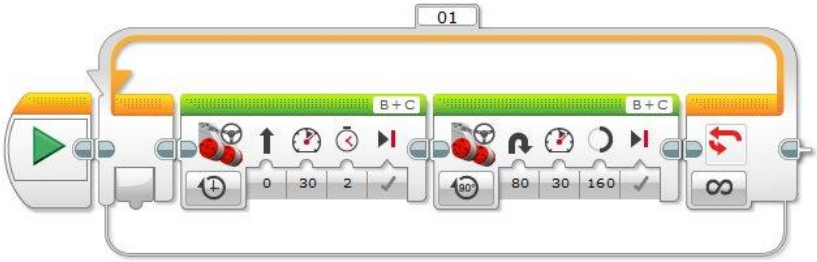


Mivel a maradékos osztás kimenete háromféle lehet, ezért egy Numerikus értékkel vezérelt elágazásban célszerű kezelni. Az elágazás értékei az osztási maradékok, az egyes szálakon szereplő utasítások a feladatleírásnak megfelelő tevékenységek.

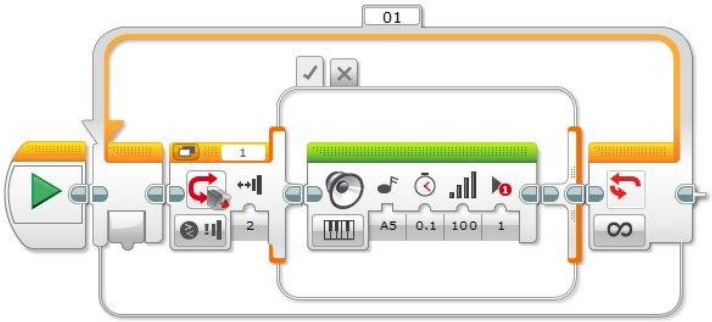


F.1.7.8. Többszálú program

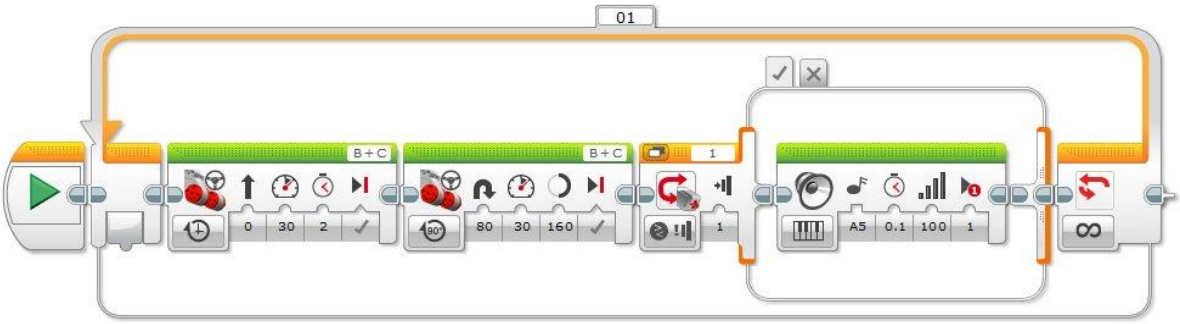
a) A sokszög alakú pályán történő haladást úgy érhetjük el legegyszerűbben, hogy a két motort adott ideig működtetjük (egyenes haladás), majd rövid ideig ellentétes irányban forgatjuk (kanyar). Mindezt ismételjük egy végtelen ciklusban. A sokszög töréspontjainak a száma a fordulás nagyságától függ.



Az ütközésérzékelőre történő hangjelzés programozása egyszerű feladat. Az elágazás alsó szála nem tartalmaz utasítást. A hangjelzés 0,1 másodpercig szól.



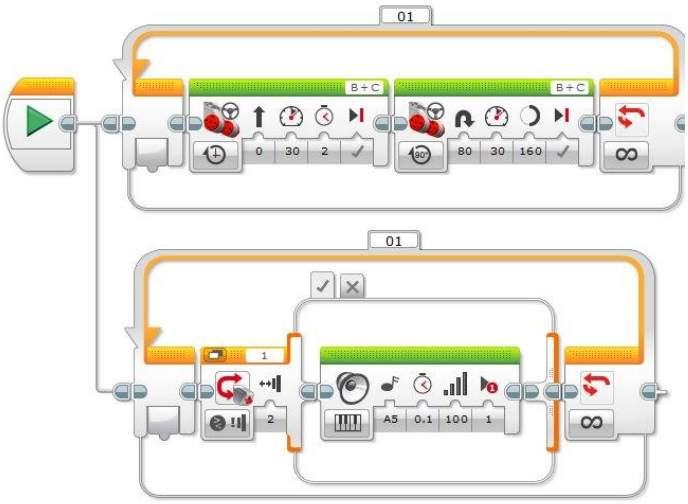
Az előző két program összeillesztése más programozási ötletet igényel. Ha ugyanabba a végtelen ciklusba betesszük egymás után a motorok vezérlését és az ütközésérzékelő figyelését is, akkor a program nem működik helyesen.



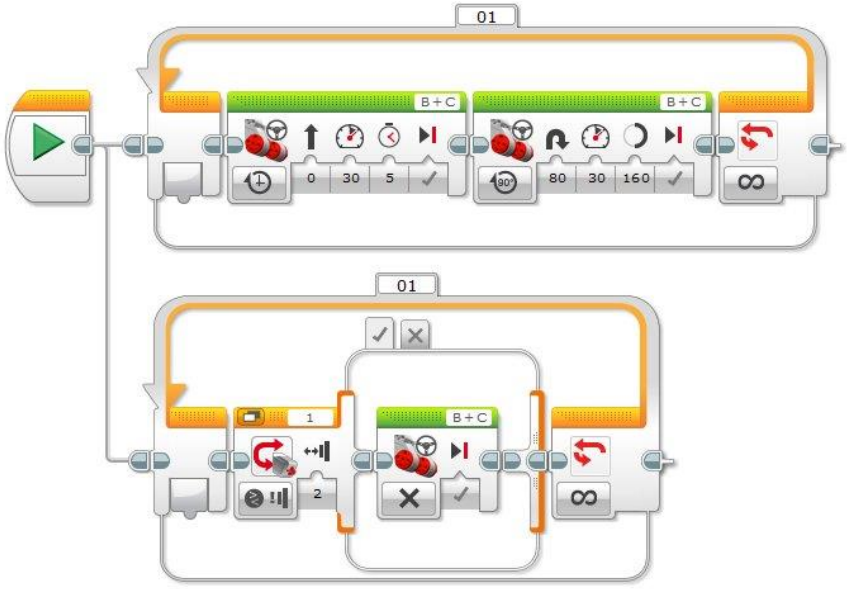
Hiába nyomjuk meg az ütközésérzékelőt, a hangjelzés csak bizonyos esetekben szólal meg. A programot tesztelve észrevehető, hogy ha az ütközésérzékelőt folyamatosan nyomva tartjuk, akkor a kanyarodás után szólal meg rövid ideig a hang (0,1 mp-ig), majd folytatódik a mozgás. Az ok a lineáris programvégrehajtásban kereshető. A két motorvezérlő modulhoz érve az utasítás-végrehajtás addig nem lép a következő modulra, amíg le nem telt a motoroknál beállított idő illetve elfordulási érték. Utána vizsgálja meg a program, hogy éppen benyomott állapotban van-e az ütközésérzékelő. Ha éppen nincs, akkor lép tovább és kezdi a ciklust, tehát a mozgást elölről. 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 helyes működéshez vezető megoldás az lehet, hogy a sokszög alakú pálya mentén történő mozgást végző részt és a szenzorfigyelést megvalósító részt folyamatosan egymással párhuzamosan hajtjuk végre. Ehhez két, látszólag egymással párhuzamosan futó szálat használunk.

A program így már helyesen működik.



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

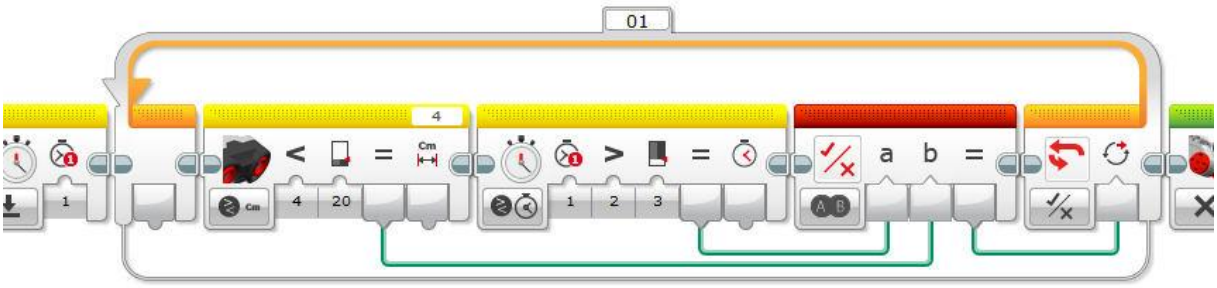


Az ütközésérzékelő felengedésére a robot újra elindul. Ha azt szeretnénk, hogy álljon meg és ne induljon el újra, akkor az alsó szál végtelen ciklusát módosítani kell és a ciklusból kilépve (ütközésérzékelőre) a program vége ikonnal le tudjuk zárni a mozgást.

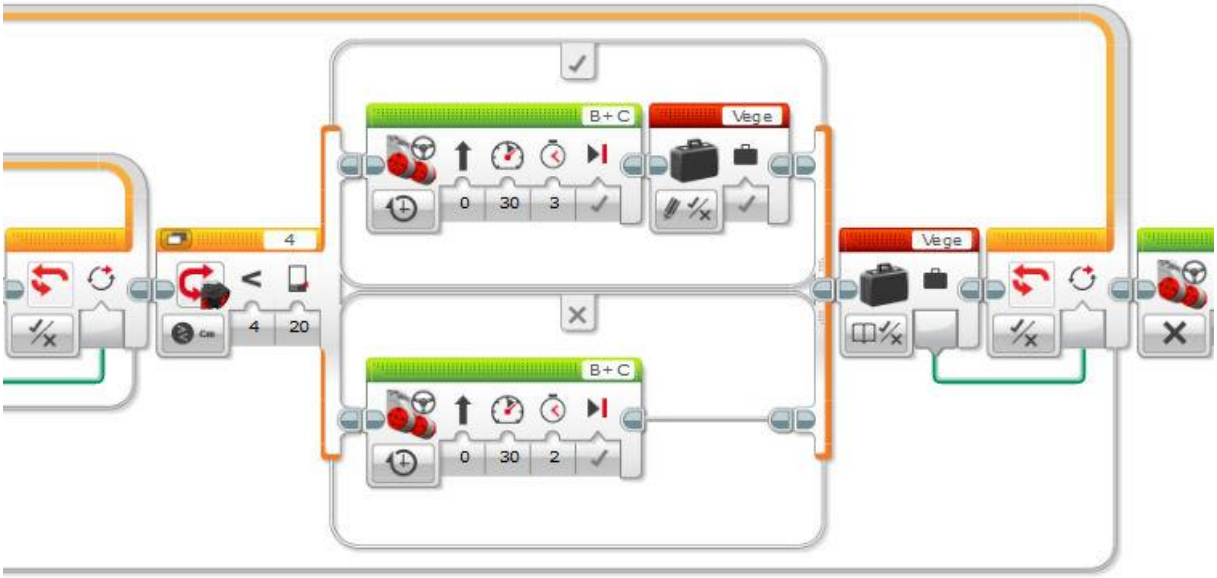
A feladat más elven is megoldható.

F.1.7.9. Összetett feltételek

a) A feladatban egy összetett feltétel szerepel, amely között vagylagos kapcsolat áll fenn. Tehát a robotnak addig kell forognia, amíg meglát valamit vagy le nem telt a 3 másodperc. Ilyen esetkeben a *Data Operations* programcsoport *Logic Operations* blokkját kell használni, az *Or* (Vagy) logikai operátort választva. Egy cikluson belül stopperrel mérve az időt és folyamatosan figyelve az ultrahag szenzor által mért távolságadatokat a vagy kapcsolat lesz a ciklus kilépési feltétele.



b) Az a) feladatrészben megírt kódot lehet módosítani. Ha a ciklusból az ultrahang szenzor mért értéke miatt léptünk ki, akkor egyenesen kell előre haladnunk és vége a programnak, egyébként egyenes előrehaladás után újra kell kezdeni a keresést. Ezt úgy oldhatjuk meg, ha a teljes utasítássort egy újabb ciklusba tesszük, amelynek kilépést egy logikai változó szabályozza. Ennek az értékét akkor állítjuk igazra, ha az ultrahang szenzor miatt léptünk ki a forgásból. Így nincs újra ismétlés, hanem vége a programnak, egyéb esetben a forgás újraindul, mert a logikai változó értéke hamis maradt.



c) Az előző feladatok alapján egyszerű a programozása. A határvonal figyelése szín/fény szenzorral történhet.

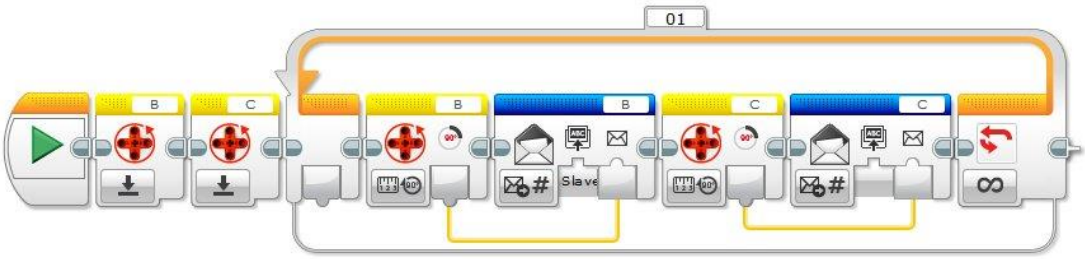
F.1.7.10. Robotkommunikáció, távirányítás

A feladat megoldásához két robot szükséges. Az egyik felépítése (szolga) lehet az alaprobottal megegyező, míg a másikra (mester) elegendő két motort csatlakoztatni, amelyek tengelyeit könnyen lehet forgatni. Például rászerelt kerekek, vagy karok segítségével.

A mester robotra két motor csatlakozik a B és C porton keresztül. A motorok elfordításának előjeles szögét küldjük át a szolga robotnak. A szolga robot két motorjának sebességértékeként használja fel a megkapott számokat. Tehát, ha a mester robot B motorjának tengelyét elfordítjuk 60°-os szöggel, akkor az átküldött érték a +60. Ezt a szolga robot megkapva, szintén a B motorját forgatja 60-as

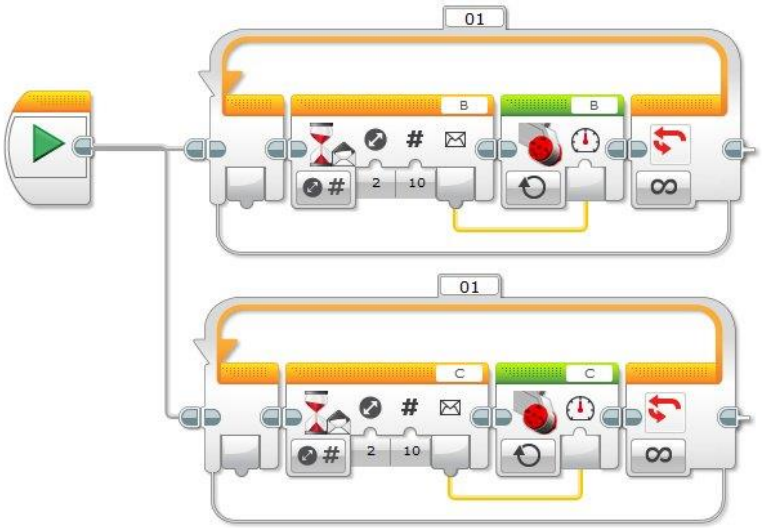
sebességgel. Ha mester robot C motorjának tengelyét elforgatjuk –40°-os szöggel, akkor az átküldött érték a –40, és a szolga robot –40-es sebességgel forgatja a C motorját, ami ellentétes irányú forgatást jelent. Gyakorlatilag, mint két botkormánnyal tudjuk irányítani a szolga robotot a mesteren keresztül. Minél jobban elforgatjuk a motorokat, annál gyorsabban mozog a szolga, illetve az ellentétes irányú forgatás a mesteren a szolga robot ellentétes irányú mozgását eredményezi. A kanyarodáshoz a két motort ellentétesen kell forgatni a mester roboton, így a szolga kanyarodni fog.

Mester robot programja:



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

A szolga robot programja:



A szolga robot két külön szálon vezérli a két motorját. A B motort a „B” azonosítójú üzenet tartalma alapján, míg a C motort a „C” címkéjű üzenet értéke alapján. A motorok *On* módban vannak és a sebességük lesz a megkapott érték. A motorok csak akkor kapnak új bemenő sebesség értéket, ha 10-zel megváltozik az üzenet értéke. Ez azt jelenti, hogy a távirányítón (mester robot) legalább 10°-kal valamelyik irányban el kellett forgatni a motor tengelyét ahhoz, hogy változás történjen a mozgásban. Amíg a megfelelő mértékű elforgatás nem történik meg a szolga robot a korábbi sebességgel mozog. A 10-es érték állítható kisebbre is, így elvileg finomabb irányítás valósítható meg.

1.8. PROJEKT – MEGÚJULÓ ENERGIA

Időigény

3-4 óra (1 óra építés + 2-3 óra programozás)

Elméleti anyag

A projektben szereplő feladatok megoldásához szükség lesz a LEGO Megújuló energiák kiegészítő csomagra. 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/EV3 téglához, így programból is lekérdezhetők a mért adatok, valamint felhasználhatók programjaink vezérlésére.

A készlet legfontosabb tartozékai:

Szolár panel

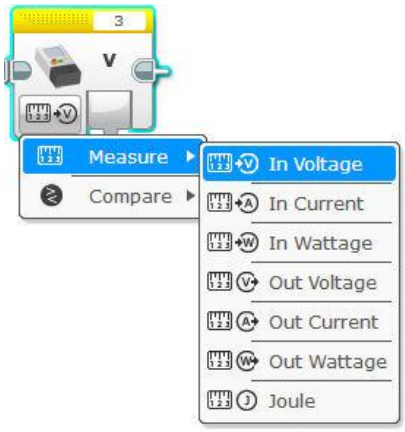
Napelem, amely egy 60 W-os izzóval közvetlenül megvilágítva 25 cm távolságból, 5 V 4 mA áramot képes előállítani. Így alkalmas pl. ledek közvetlen működtetésére, vagy a megújuló energiaforrások használatának bemutatására.

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.



A programozáshoz szükséges az *Energy Meter* modul.

A modul segítségével az energiaforrás négy paraméterét kérdezhetjük le a beállításának megfelelően: feszültség (*In Voltage*), áramerősség (*In Current*), teljesítmény (*In Wattage*), és megtermelt energia (*Joule*). 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.

Bevezető történet

Sajnos a király országában fogytán volt az energia. Az emberek nem is gondolták végig, hogy a mindennapi tevékenységeik során mennyi energiát használnak fel a munkájukhoz, a főzéshez, a világításhoz, … Főleg mióta a Tudós felfedezte az elektromosságot és sikerült az ország szén, gáz és olaj tartalékaiból elektromos energiát előállítani, azóta vált a pazarlás országnyi méretűvé. Az utcákon szinte éjszaka is nappali világosság volt. A bevásárló központokban jégpályákon és fűtött uszodákban szórakozhattak a vásárlástól megfáradt járókelők. Mivel ezekből a létesítményekből annyi volt, hogy a látogatók száma alig haladta meg 2-3 főt naponta. De mivel ez volt a divat és úgy érezték az emberek, hogy akinek nincs jégpályája az hátrányba kerül (persze senki sem tudta, hogy mit is jelent ez a hátrány), ezért mindenki építkezett.
Így megdöbbenve vették tudomásul, hogy elfogyott a szén, a gáz a kőolaj. Nincs miből előállítani az elektromos energiát, ami mindent működtetett.
A király szokásához híven hozott hát egy bölcs döntést és kiadta a Tudósnak, hogy oldja meg az energiaválságot, mert a jégpályák kellenek, hiszen a szomszéd országok mit szólnának, ha néhány (egyébként fölösleges) energiafogyasztó létesítményt bezárnának.
A Tudós sokat gondolkodott a probléma megoldásán, míg végül előállt néhány olyan ötlettel, ami először mindenkit meglepett. Azokat a természeti jelenségeket kellene felhasználni energiatermelésre, amelyek nem fogynak el, mert folyamatosan megújulnak. Ilyen például a szél, a folyók vízáramlása, vagy a napfény. Az ötlet egy idő után mindenkinek tetszett, de a megvalósítást ismét csak a tudósra bízták, aki mint már annyiszor, most is a robotokat hívta segítségül.

F.1.8.1. Feladat – Szolár panel építés

Szint:

Építsd meg a készletben szereplő leírás szerint a szolár panelt működtető konstrukciót!

F.1.8.2. Feladat – Fénykereső

Szint:

Írj programot, amelyet végrehajtva a robot fény/szín szenzora segítségével meghatározza azt az irányt, amely felől a legnagyobb fényintenzitás mérhető! A mérést egy kör kerülete mentén mozgatott fényszenzor segítségével végezze!

F.1.8.3. Feladat – Napkövetés

Szint:

Írj programot, amely a szolár panelt arra fordítja, amerre a legnagyobb fényintenzitás mérhető!

F.1.8.4. Feladat – Szélturbina építés

Szint:

Építsd meg a készletben szereplő leírás szerint a szélturbinát működtető konstrukciót!

F.1.8.5. Feladat – Szélirányba állás

Szint:

Írj programot, amelyet végrehajtva a robot kiírja a képernyőjére, hogy adott irányból fúj-e a szél vagy sem, és szélcsend esetén hangjelzést is ad!

Programozási ötletek

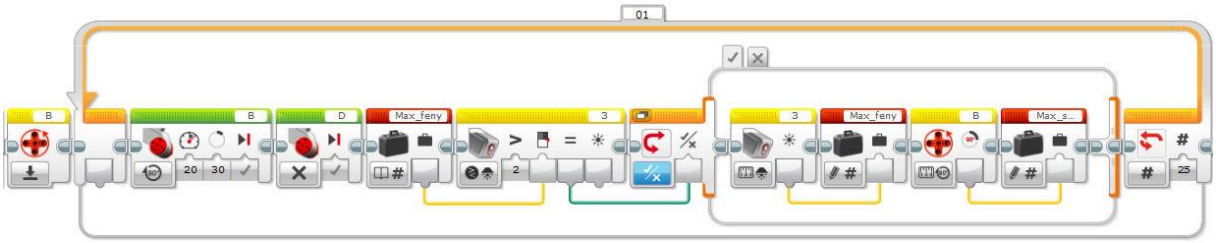
F.1.8.2. Fénykereső

A szolár panel nap felé fordítására azt a módszert érdemes választani, 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 megoldási ötletet két részletben mutatjuk be. Az első részben történik a mérés és a legnagyobb fényintenzitáshoz tartozó elfordulási szög meghatározása. Két szám típusú változót hozunk 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 határozzuk meg, hogy az adott robotkonstrukció esetén a teljes kör mekkora elfordulást jelent (pl.: 750 fok). Így 25-ször lefutó növekményes ciklust használhatunk (30x25 = 750). A cikluson belül a *Max_feny* változó tartalmát (kezdetben 0) összehasonlítjuk az aktuálisan mért fényintenzitással (a fényszenzor lámpáját érdemes kikapcsolni). 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 ezután egy egyszerű paraméterátadással megoldhatjuk.



F.1.8.3. Napkövetés

Az előző feladatban elkészített algoritmust használjuk a szolár panel forgatására. A panelt a B portra csatlakoztatott motor forgatja a fény/szín szenzorral együtt.

F.1.8.5. Szélirányba állás

A bemutatott *Energy Meter* 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 állítsuk 1-re 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 szolár panelnél bemutatott algoritmus ötletet felhasználva a szélturbinát a megfelelő irányba is forgathatjuk.

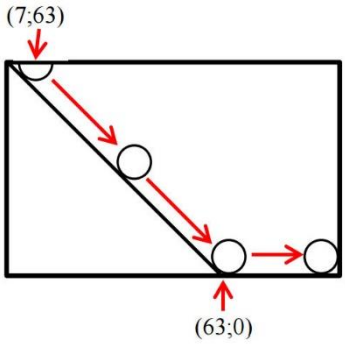
2. HALADÓ FELADATOK

2.1. DINAMIKUS RAJZOK

F.2.1.1.a-c Feladat – Guruló golyó

a) Szint:

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

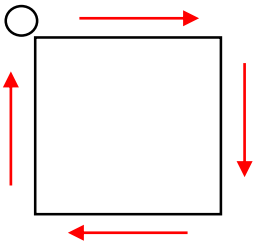


A kör középpontjának kezdő koordinátája NXT robotnál (7;63), EV3 robotnál (7;0).

A lejtő egyik végpontjának koordinátái NXT robotnál (63;0), EV3 robotnál (127;127), és a lejtő 45° -os. A kör egyenletes sebességgel haladjon végig az útvonalon, és a képernyő szélét elérve álljon meg! A program a befejezés előtt 10 másodpercig várakozzon!

b) Szint:

A golyó egy négyzet kerületén mozogjon körbe-körbe. A négyzet bal felső sarkának koordinátái EV3-as robot esetén (20; 20), a négyzet oldala 60 pixel. A kör sugara 10 pixel. A kör útvonalát az ábra értelmezi.



c) Szint:

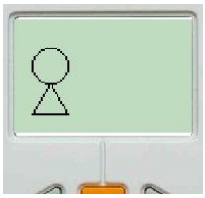
A 10 pixel sugarú kör egy 40 pixel sugarú kör kerületén mozogjon körbe-körbe.

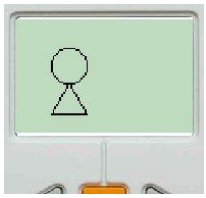
F.2.1.2. Feladat – Alakzat mozgatása

Szint:

Írj programot, amelyet végrehajtva a robot a képernyőjére rajzol egy szabályos háromszögből és egy körből álló ábrát a képnek megfelelően! A háromszög bal alsó csúcsának koordinátái: (10;10), az oldalának hossza 20 pixel. A kör sugara 10 pixel. A kör érinti a háromszög felső csúcsát és középpontja a háromszög szimmetria tengelyére esik. Ütközésérzékelő benyomására a teljes ábrát (háromszög és kör) tolja el 10 pixellel jobbra úgy, hogy a régi ábra már ne látszódjon a képernyőn! Mindezt hatszor ismételje! Az új ábra mindig az ütközésérzékelő megnyomására jelenjen meg! A hatodik rajz után a program várjon 5 másodpercet, majd álljon le!

Az ábrák NXT robot képernyőjén (EV3 robot esetén az ábra fejjel lefelé jelenik meg):

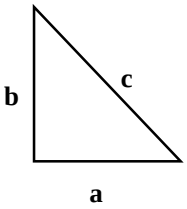

Nyitó kép


1. elmozdulás


2. elmozdulás

... 
... Utolsó kép

Matematikai segítség a háromszög csúcskoordinátáinak meghatározásához (Pitagorasz tétele):
Derékszögű háromszögben az átfogó hosszának négyzete egyenlő a befogók hosszainak négyzetösszegével.



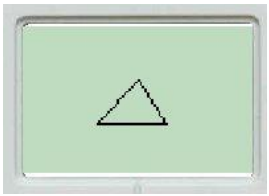
$$c^2 = a^2 + b^2$$

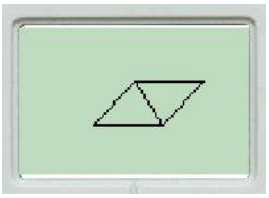
F.2.1.3. Feladat – Tükrözések

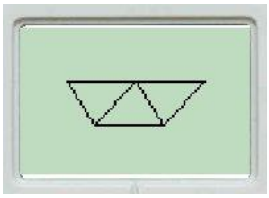
Szint:

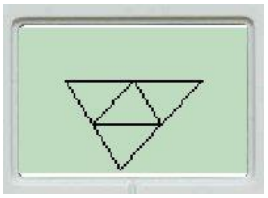
Írj programot, amelyet végrehajtva a robot a képernyőjére rajzol egy háromszöget! A háromszög csúcsának koordinátái: (32;21); (62;21); (50;40). Az ütközésérzékelő benyomására jelenjen meg a képernyőn az eredeti háromszög egyik oldalfelező pontjára vonatkozó tükörképi háromszög is! Az ütközésérzékelő további megnyomásaira jelenjen meg egy másik oldalfelező pontra vonatkozó tükörkép, valamint a harmadik oldalfelező pontra vonatkozó tükörkép is! Minden háromszög látszódjon egy időben képernyőn és a program ütközésérzékelő megnyomására álljon le!

A robot képernyőjén látszó kép (NXT):


Alap képernyőkép


1. tükrözés után


2. tükrözés után

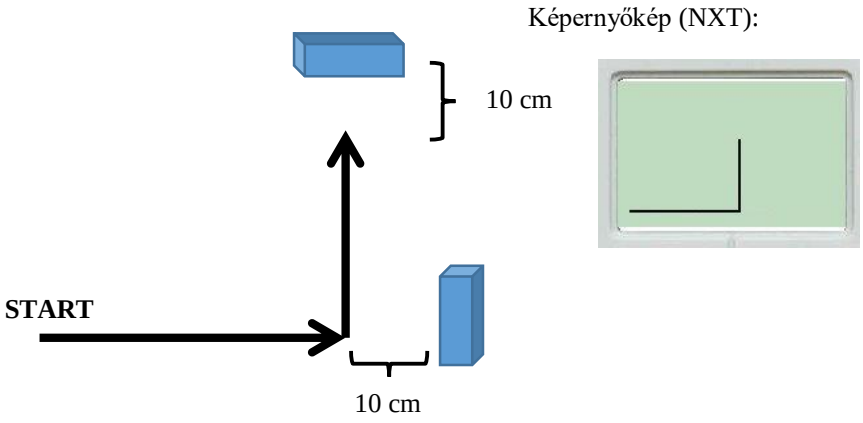

3. tükrözés után

F.2.1.4. Feladat – Útvonal rajzolás

Szint:

Írj programot, amelyet végrehajtva a robot előre mozog akadályig és közben rajzolja a képernyőre a megtett utat! Az akadály előtt 10 cm-rel forduljon kb. 90°-ot balra és ismét mozogjon akadálytól 10 cm-ig, közben a képernyőre rajzolja tovább az útvonalát! A fordulást a képernyőn derékszögként kell értelmezni, függetlenül attól, hogy a robot esetleg nem pontosan 90°-ot fordult. A rajzolást a képernyő (10; 10) koordinátájú pontjánál kezdődjön NXT robot esetén és (120,10) koordinátájú pontján EV3 robot esetén! A második akadály észlelése után a robot álljon meg, és ütközésérzékelő benyomása után álljon le a programja! A rajzolásnál a robot 1 tengelyfordulata feleljen meg a képernyőn 10 pixelnek!

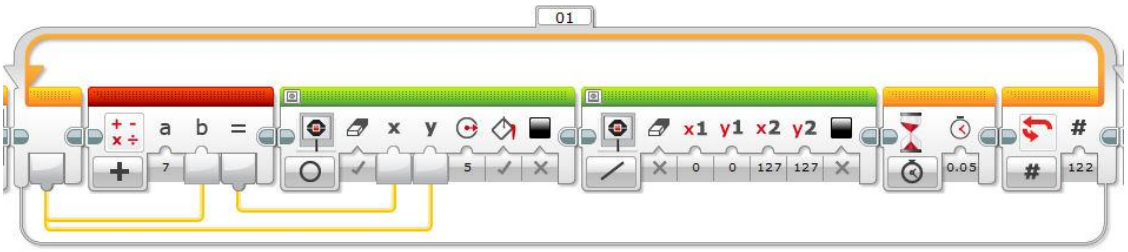
Például:



Programozási ötletek

F.2.1.1. Guruló golyó

a) A golyó mozgása két részből áll. Az első szakaszon, mikor lefelé gurul, a kör középpontjának *x* és *y* koordinátája azonosan növekszik (EV3 robotnál). A növekedés vezérlésére a ciklusváltozót használhatjuk. Az NXT robotnál az *x* koordináta növekszik, míg az *y* ugyanolyan mértékben csökken.



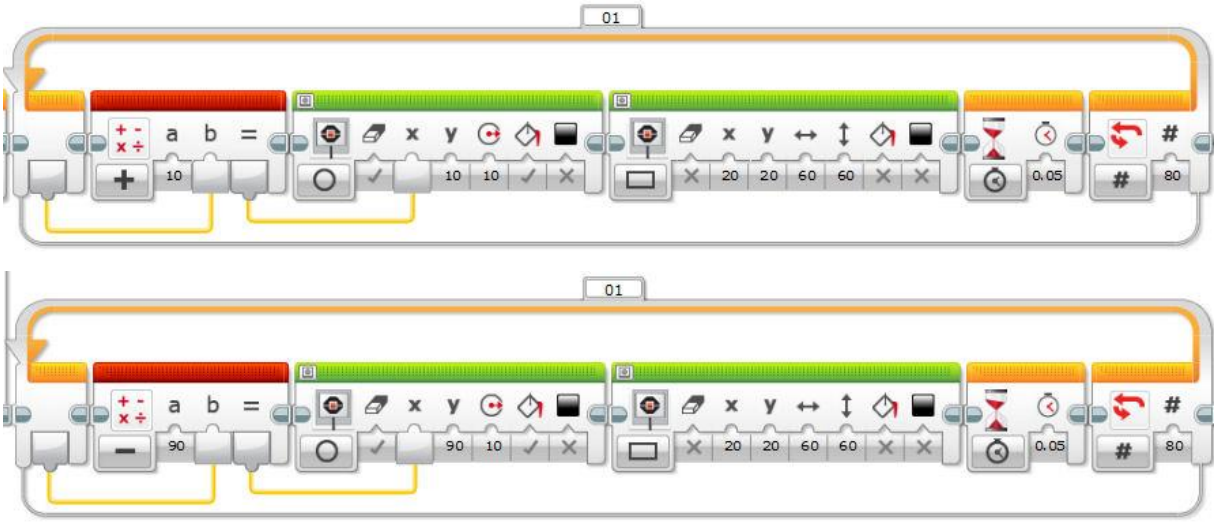
Mivel a golyót folyamatosan újra rajzoljuk, ezért mindig törölni kell a képernyőt, tehát a lejtő felrajzolását is a ciklusba kell tenni. A ciklus 122-szer fut le (EV3), hiszen az y koordináta 0-ról indul és a sugárnyira áll meg a képernyő aljától.

A második szakaszon csak a golyó középpontjának csak az *x* koordinátája változik, ami 129-es kezdőértékről 172-ig változik (43-szor fut le a ciklus EV3 robot esetén).

A mozgás szemmel való követéséhez a ciklusok egy 0,05 másodperces lassítást tartalmaznak.

b) A program magyarázata az EV3 típusú robotra vonatkozik. NXT esetén az elv hasonló, de a koordináta rendszer origójának más a helye, így az *x* illetve *y* koordináta változása eltérő.

A golyó mozgását négy részből érdemes összerakni, a négyzet négy oldalának megfelelően. A kör középpontjának mindig csak az egyik koordinátája változik. A vízszintes mozgások alatt az *x*, míg a függőleges mozgásoknál az *y*. A mozgás kezdőpontját számolni kell (gyakorlatilag egy 80 oldalú négyzet csúcsai). Minden mozgás 80 lépésből áll. Az azonos tájolású mozgások esetén először növelni kell a koordinátákat, majd csökkenteni. Az *x* koordináta esetén a két ciklus:



Az *y* koordinátára hasonló a megoldás. A négy ciklust egymás után fűzve a képernyőn a megfelelő animáció jelenik meg.

c) A kör körvonal mentén történő mozgatásához a kör paraméteres egyenletrendszerét használjuk fel.

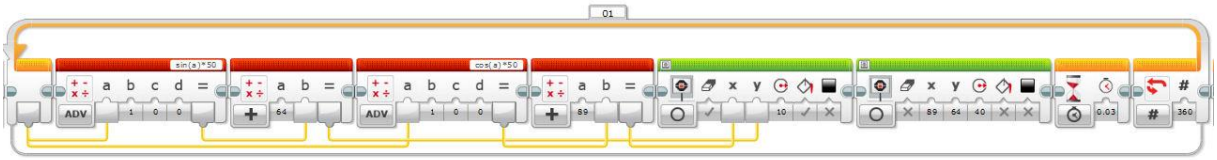
A kör paraméteres egyenlete:

$$x = r \cdot \cos(\alpha)$$
$$y = r \cdot \sin(\alpha)$$

Ahol *r* a kör sugara az α pedig egy forgásszög (a szoftverben fokban megadva). Ha az α 0-360 fok között változik, akkor egy teljes fordulat történik.

A trigonometrikus függvényeket a *Data Operation/Advanced* blokkján keresztül érhetjük el.

A szög megadását a ciklusváltozó jelenti és a kör sugara 50, mivel az eredeti kör 40 és ehhez kell még adni a mozgó kör 10 pixeles sugarát. A kód matematikai lényegét mutatja a következő ábra:



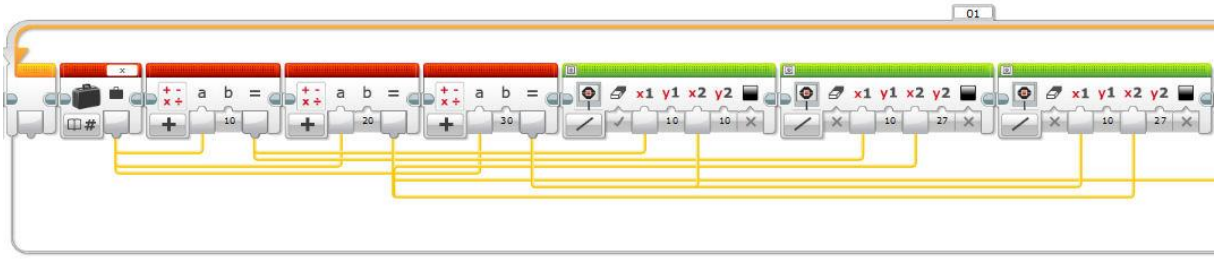
F.2.1.2. Alakzat mozgatása

A háromszög csúcsainak koordinátái: (10;10), (30;10), (20;27).

A kör középpontja: (20;37).

Ciklussal végezhetjük a hatszori eltolást, az *x* koordináták értékét kell minden esetben növelni 10-zel.

Érdemes egy változót létrehozni, amely kezdőértéke 0. Ehhez 10-et, 20-at, 30-at hozzáadva a kapott értékeket használhatjuk a vonalak és a kör *x* koordinátáinak megadására. Ütközésérzékelő benyomására az *x* értékét 10-zel növelve az ábra jobbra mozdul. A forráskód egy részlete:



A feladat egy lehetséges megoldása, ha saját blokkot készítünk a háromszöget és kört tartalmazó ábrából, ami bemeneti paraméterként az *x* koordináta értékét várja.

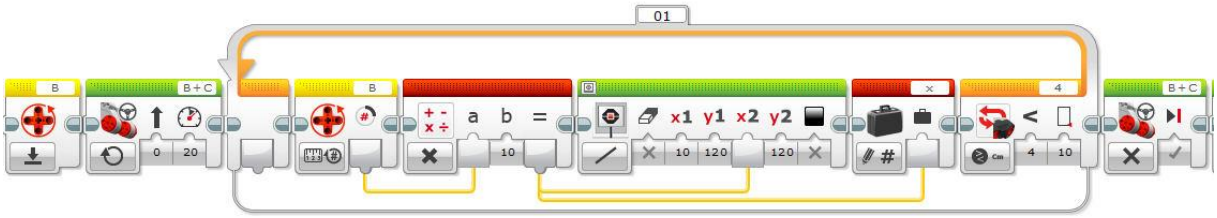
F.2.1.3. Tükrözések

A tükrözött háromszögek koordinátáit ki kell számolni és megrajzoltatni őket. Az új háromszögek esetén elegendő két-két vonalat rajzolni, mert a harmadik fixen marad. A számoláshoz érdemes négyzetrácsos lapon felvett koordinátarendszert használni. Az EV3 robot esetén az ábra fejjel lefelé látszik.

F.2.1.4. Útvonal rajzolás

A tengelyfordulatok számlálását például a B motor esetén végezzük, ami nem lesz egész szám. A megadott aránynak megfelelően az elfordulás tízszeresét feleltetjük meg a képernyő pixeleinek. A kezdő (10,120) koordinátától indulva folyamatosan újrarajzoljuk a vízszintes szakaszt. Ahhoz, hogy a fordulás után a függőleges szakaszt is rajzolni tudjuk, meg kell jegyeznünk, hogy milyen *x* koordinátánál ért véget a vízszintes vonal. Ezt egy változóban tároljuk. A 90 fokos fordulás után ugyanezt az algoritmust használjuk, de EV3 robot esetén a 120-as *y* koordinátából kivonjuk a mért értéket.

A vízszintes rajzolás kódja:



2.2. HANGHATÁSOK

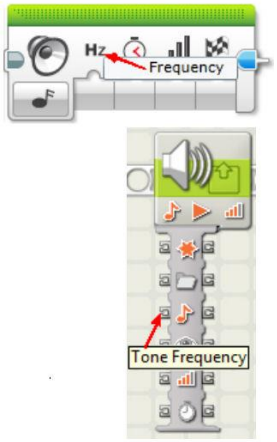
F.2.2.1. Feladat – Hangszer

Szint: 

Írj programot, amelyet végrehajtva a robot egy hangszerként működik! A távolságérzékelőjével mért érték 50-szeresét a hang frekvenciájaként (Hz) értelmezve, egy század másodperc időtartamig játszik le egy hangot, majd ezt ismétli mindaddig, amíg új értéket nem mér a távolságérzékelő. Az ütközésérzékelővel lehessen „oktávot” váltani. Ha be van nyomva az ütközésérzékelő, akkor a mért távolság 100-szorosa legyen a hang frekvenciája (Hz), egyébként az 50-szerese. A program kikapcsolásig működjön!

Tehát minél távolabb lát akadályt maga előtt a robot, annál magasabb hangot szólaltat meg. Pl.: A robot 15 cm-re észlel egy akadályt (ennyit mér az ultrahangos távolságérzékelője), akkor a megszólaltatott hang a 15x50 = 750 Hz frekvenciájú, ha nincs benyomva az ütközésérzékelője. Abban az esetben, ha be van nyomva az ütközésérzékelő, akkor 15x100 = 1500 Hz a megszólaltatott hang frekvenciája.

(Egy hang annál magasabb, minél nagyobb a frekvenciája. Minden hangnak megvan a saját frekvenciaértéke. Pl.: a normál zenei „A” hanghoz a 440 Hz-es frekvenciaérték tartozik.)

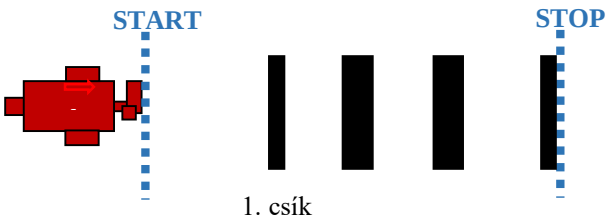


F.2.2.2. Feladat – Morzekód (csíkszélesség alapján)

Szint: 

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

Például:



1. csík

F.2.2.3.a-b Feladat – Véletlen „zene”

a) Szint:

Sorsoljon a robot véletlen számokat, amelyek egy hangfájl hangjainak frekvencia és időtartam adatai lesznek! A kisorsolt számokat használva a *Sound* blokk megfelelő paramétereiként, a robot játssza le a véletlen zenét! A hang frekvenciája 400-2000 Hz-es frekvenciatartományba essen, az időtartam pedig a 0,1 – 1 másodperces tartományba! Sorsolj összesen 20 számpárt!

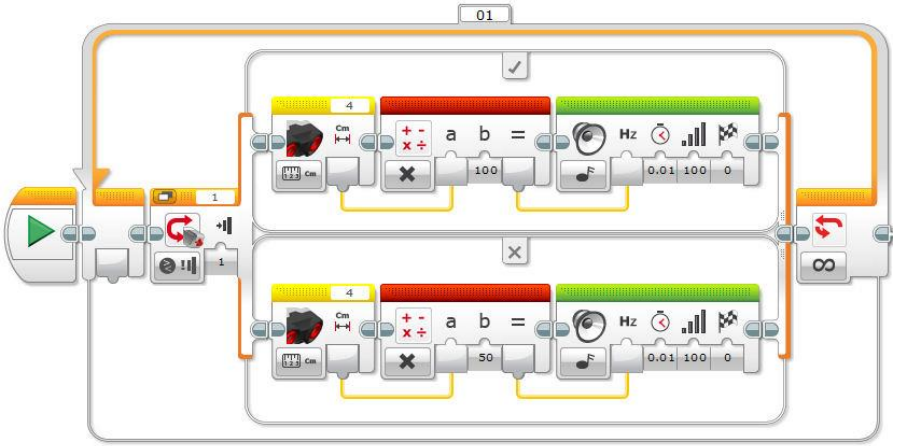
b) Szint:

A sorsolt számokat a robot írja ki egy fájlba, minden számot új sorba írva! A számpárok közül az első legyen a frekvencia, a második az időtartam! Az így elkészült fájlból olvassa be a robot az adatokat és játssza le a „zenét”! A fájlokat lehet cserélni a csoportok között!

Programozási ötletek

F.2.2.1. Hangszer

A feladat megoldása egy egyszerű paraméterátadásra épül. Az ultrahangszenzor által mért érték 50 vagy 100 szorosát kell a hangszóró bemeneti portjára csatlakoztatni egy elágazás két szálán. Az elágazást az ütközésérzékelő vezérli.

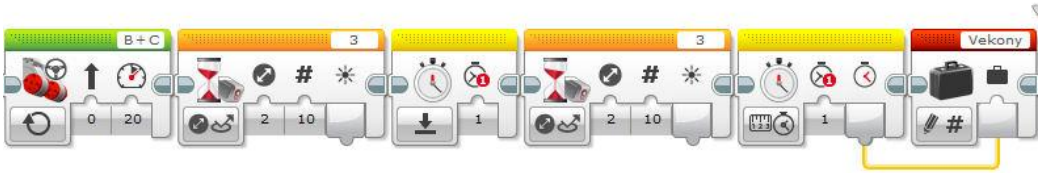


F.2.2.2. Morzekód (csíkszélesség alapján)

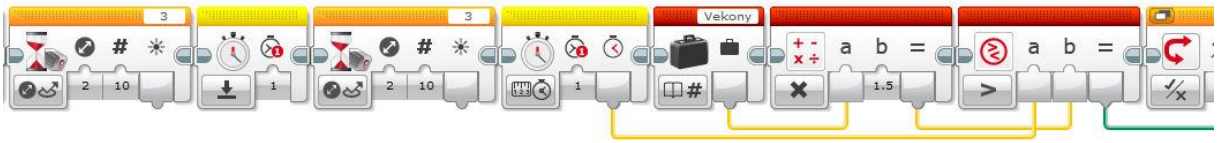
Az algoritmus lépései:

- Célszerű egy 4 elemű tömb típusú változót létrehozni és 0 értéket tárolni a megfelelő tömbelemben, ha a hang rövid és 1 értéket, ha hosszú.

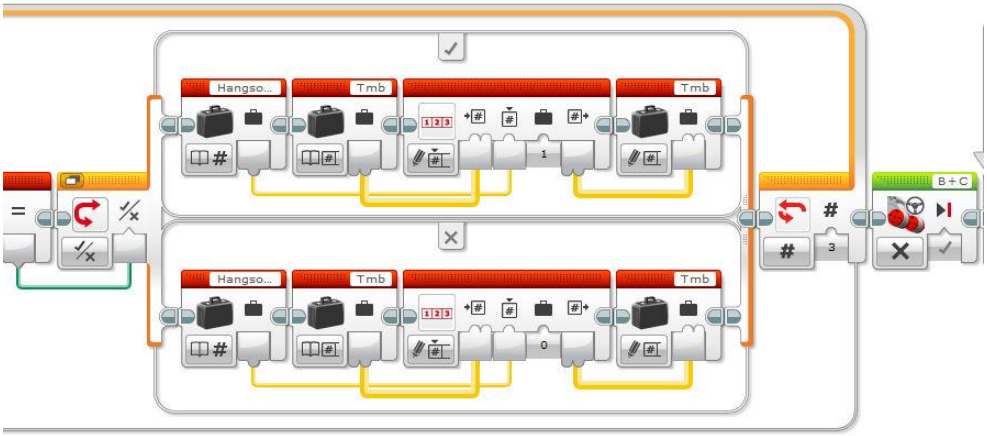
- Az első csík szélességének mérése (stopperrel) és tárolása (lásd feladatgyűjtemény korábbi feladata). Az első morze hang rövid. A 0. tömbelembe 0 kerül, amelyet a létrehozáskor is beírhatunk.



- Egy ciklusban, amely 3-szor fut le, lehet a további csíkok szélességét mérni. Minden csík után a kapott szélességet össze kell hasonlítani az első (rövid) csíkra kapott érték pl. 1,5-szeresével. Ha nagyobb, akkor széles volt a csík egyébként keskeny.



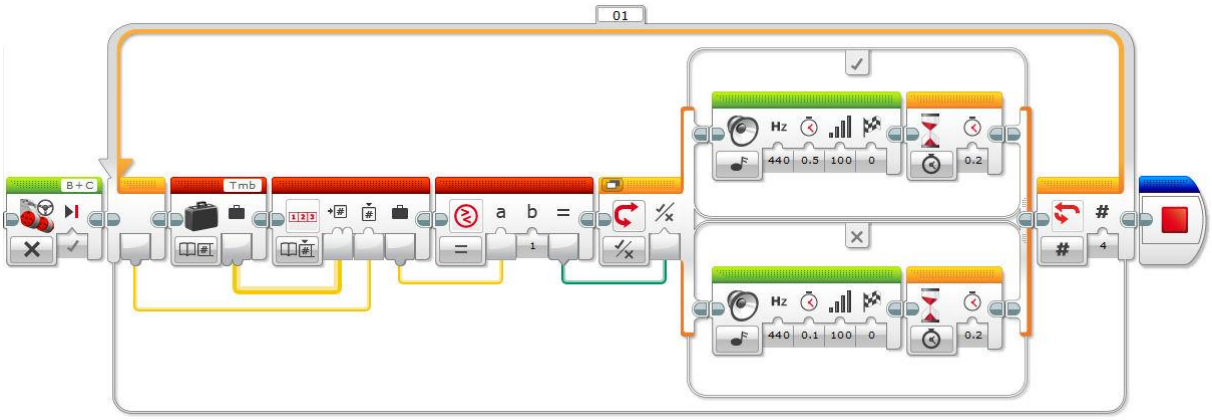
Ennek megfelelően 1 vagy 0 értéket tárolni a megfelelő tömbelemben. Azért érdemes a rövid csík 1,5-szeresével összehasonlítani, mert a csíkok feletti áthaladás időtartama két azonos szélességű keskeny csík esetén sem lesz egyenlő (a mérés időpontja befolyásolja, és apró eltérések lesznek).



A *Hangsorszam* változó adja meg, hogy hányadik hangról van szó (a tömb indexe), 1-től kezdve folyamatosan növekszik a csíkok számával. A ciklusváltozót lehet használni értékként.

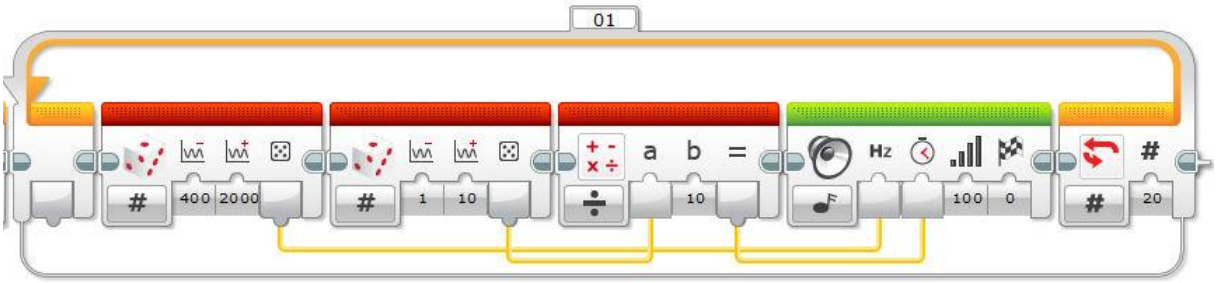
- A negyedik csík után a robot megáll.

- Egy újabb ciklussal, amely 4-szer fut le a tömbben tárolt 0 vagy 1 értékeknek megfelelően megszólaltathatók a rövid vagy hosszú hangokat, köztük a megadott időtartamú szünettel.



F.2.2.3. Véletlen „zene”

a) Egy zenei hangot alapvetően két értékkel jellemezhetünk: a frekvenciájával (milyen magas/mély a hang) és a megszólalás időtartamával. A sorsolt számok ezt a két értéket jelentik. Mivel a véletlen szám sorsoló csak egész számokat tud generálni, ezért 1 és 10 közötti számokat sorsoltassunk és osszuk el a kapott értéket 10-zel. Ezt használjuk fel a hang lejátszási időtartamaként!



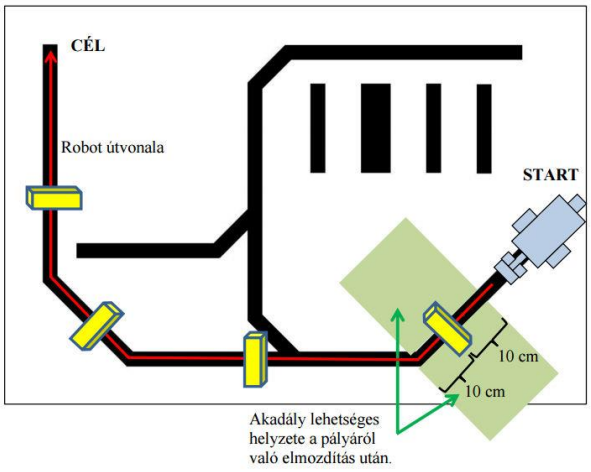
b) Az előző feladat fájlkezelés közbeiktatásával átalakított változata. A fájlok használatával lehetőségünk lesz a „zeneszámok” cseréjére.

2.3. MOZGÁSOK

F.2.3.1. Feladat – Akadály eltávolítás

Szint:

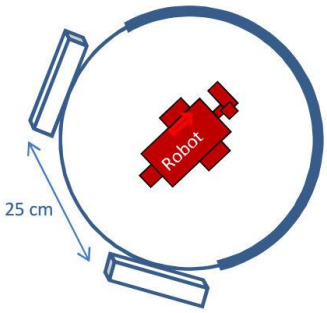
Írj programot, amelyet a robot végrehajtva startpozícióból indul, és követi a fekete színű vonalat! Ha a robot 10 cm-en belül akadályt érzékel, akkor azt tolja a pálya mellé, majd kövesse tovább az útvonalat! A robot a dobozt nem tolhatja maga előtt, azt az útvonal mellé kell juttatnia, vagyis a doboznak az eredeti helyzetéhez viszonyított, ±10 cm-es sávon belül kell lennie (érintkeznie kell a sávval, de a követendő útvonalhoz nem érhet hozzá). Lásd ábra! A dobozok száma és helyzete előre nem ismert. Ha a robot elérte az ábrán jelölt célt, akkor nem kell megállnia, de a mozgását befejezettnek tekintjük a feladat szempontjából.



F.2.3.2. Feladat – Kapukeresés

Szint:

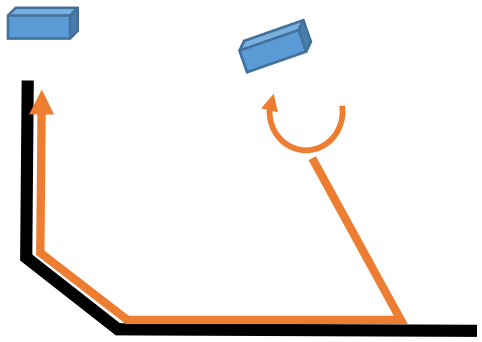
Írj programot, amelyet végrehajtva a robot megkeres egy kaput és áthalad rajta! A kapu két téglatestből áll, amelyek legalább 10 cm x 8 cm x 5 cm méretűek. A robot egyenletes sebességgel helyben forog és körülette két doboz van elhelyezve úgy, hogy a forgási középponttól legfeljebb 25 cm sugarú körvonalra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



F.2.3.3. Feladat – Mozgásvezérlés (összetett szenzorhasználattal)

Szint:

Írj programot, amelyet végrehajtva a robot lassan forog mindaddig, amíg ultrahang szenzora 10 cm-en belül akadályt érzékel. Ekkor tolatni kezd. A tolatást mindaddig végezze, amíg fény/szín érzékelője fekete vonalat nem érzékel! Elérve a vonalat azt kövesse (balra), és akadálytól 10 cm-re álljon meg! Ekkor a robotot tetszőleges helyre áthelyezve, az ütközésérzékelő megnyomására kezdje újra elölről a korábbi lépéseket. Mindezt kikapcsolásig ismételje! Egy lehetséges útvonalat az ábra értelmez.



F.2.3.4. Feladat – Sebességvezérlés

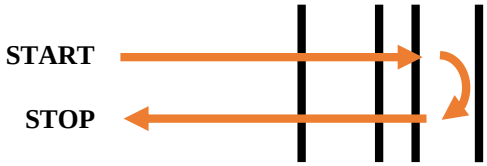
Szint:

Írj programot, amelyet végrehajtva a robot a startpozícióból indulva egyenesen előre halad 20-as sebességgel a fehér színű pályán elhelyezett fekete csíkok felett. Minden feketecsíkon történő áthaladás után 10-zel növelje meg a sebességet. A mozgást addig folytassa, amíg a sebessége eléri a 50-es értéket, vagy 15 cm-em belül akadályt érzékel. A mozgást mindig az előbb bekövetkező esemény állítsa meg. A megállást követően tolasson ütközésig, majd ismét induljon előre 20-as sebességgel és ismételje az előző tevékenységét. Mindezt kikapcsolásig ismételje.

F.2.3.5.a-b Feladat – Mozgásvezérlés (számlálással)

a) Szint:

Írj programot, amelyet végrehajtva a robot startpozícióból indul egyenesen előre egy fekete csíksor fölött! A harmadik csíkon történő áthaladás után forduljon körülbelül 180°-ot és menjen vissza az indulási pozícióba. A robot indulási pozíciója tetszőleges lehet, de a fekete csíksorra merőleges és tőle balra van. Egy lehetséges útvonalat mutat az ábra.



b) Szint:

Írj programot, amelyet végrehajtva a robot startpozícióból indul egyenesen előre egy fekete csíksor fölött! A képernyőjén beállított számnak megfelelő számú fekete vonal után körülbelül 180°-ot fordul és visszamegy az indulási pozícióba. A képernyőjén a számot az ütközésérzékelő megfelelő számú benyomásával lehessen beállítani! A robot az ENTER gomb megnyomására induljon el!

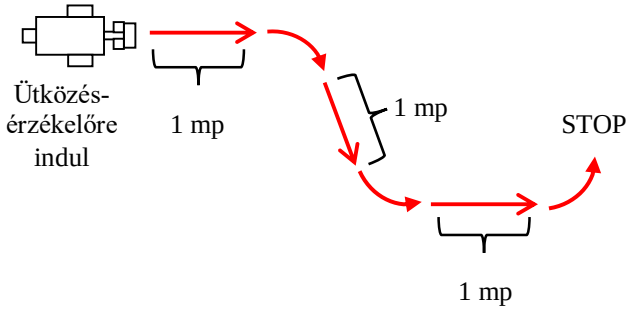
F.2.3.6. Feladat – Mozgásvezérlés (nyomógombokkal) 1.

Szint:

Írj programot, amelyet végrehajtva a roboton lévő „bal” ( ) illetve „jobb” ( ) gombok segítségével lehet beállítani a mozgását, a következő leírásnak megfelelően. A bal gomb megnyomására megjelenik a képernyőn a bal szó, míg a jobb gomb megnyomására a jobb szó. Mindezt háromszor kell ismételni, tehát három egymás alatti sorban látszik a képernyőn a bal vagy jobb szavak valamilyen variációja (lásd ábra). Ezután ütközésérzékelőre elindul a robot és 1 másodpercig halad egyenesen előre, majd 200°-os tengelyfordulattal fordul balra vagy jobbra a beállított első értéknek megfelelően. Ezután újra egyenesen halad 1 másodpercig, majd újra fordul 200°-os tengelyfordulatot a második beállított értéknek megfelelően. Ugyanezt a mozgást elvégzi a harmadik beállított értékre is.

Beállított értékek a képernyőn (NXT):

A robot mozgása:



F.2.3.7. Feladat – Mozgásvezérlés (nyomógombokkal) 2.

Szint:

Írj programot, amelyet végrehajtva a robot 30-as sebességgel halad előre, majd kb. 90°-ot fordul balra és ismét egyenesen előre halad. A két előre mozgás időtartamát a tégla nyomógombjai segítségével lehessen beállítani. Először az elindulás utáni előre haladás időtartamát kell megadni egész másodpercben. Majd a tégla enter gombjának benyomása után a fordulás utáni előrehaladás időtartamát szintén egész másodpercben. A robot a mozgást az ütközésérzékelő benyomására kezdje meg. A téglán lévő jobb oldali háromszög alakú gombbal lehessen az mozgási időtartamot egyesével növelni, míg a balra lévő háromszög alakú gomb segítségével eggyel csökkenteni. A narancssárga gomb jelentse a beállított érték véglegesítését. Mindkét beállított számot jelenítse meg a robot képernyőjén, egymás alatti sorokban!

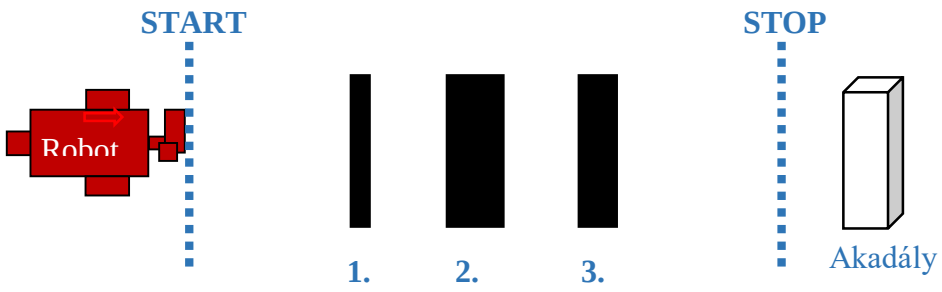
F.2.3.8. Feladat – Sorba rendezés (csíkszélesség alapján)

Szint:

Írj programot, amelyet végrehajtva a robot az ábrán jelzett startpozícióból indul egyenesen előre három egymással párhuzamos, de különböző szélességű fekete csík fölött, a csíkokra merőleges irányban. A pálya alapszíne fehér. Akadálytól 10 cm-re megáll és képernyőjére írja a csíkok

sorszámát a szélességüknek megfelelő csökkenő sorrendben, a képernyő ugyanazon sorába, de térközzel elválasztva egymástól. A csíkok számozása a startpozícióhoz legközelebbivel kezdődik és a robot haladási irányának megfelelően növekszik. A három fekete csík szélessége: 2,5 cm; 5 cm illetve 7,5 cm, hosszuk tetszőleges, de legalább 20 cm. A csíkok sorrendje tetszőleges lehet. A program befejezése előtt 5 másodpercig várakozzon, hogy a képernyőre írt adatok elolvashatók legyenek. A csíkok közötti távolság legalább 10 cm.

Például



A képernyőre írt számsor: 2 3 1

F.2.3.9. Feladat – Mozgásvezérlés (véletlen számok alapján)

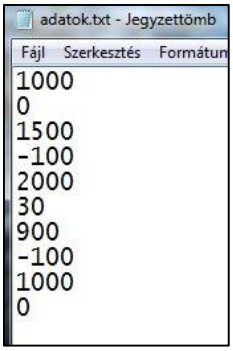
Szint:

Írj programot, amely során a robot sorsol öt darab –5 és +4 közötti véletlen számot! A számokat írja ki a képernyőre, egymás fölötti sorokba! Majd menjen előre 0,5 másodpercig, és forduljon balra kb. 90°-ot ha a sorsolt szám negatív, vagy forduljon jobbra kb. 90°-ot egyébként! Mindezt ismételje meg 5-ször a kisorsolt számoknak megfelelően (előre → jobbra/balra → előre → jobbra/balra → …)! Az ötödik fordulás után a robot álljon meg és várjon 10 másodpercig a program leállása előtt!

F.2.3.10. Feladat – Mozgásvezérlés (fájlban tárolt adatok alapján)

Szint:

Írj programot, amelyet végrehajtva a robot egy szövegfájlban megkapott adatsor alapján mozog! A szövegfájlban 10 db egész szám található. A számok párban értelmezendők. A párok első eleme a robot mozgásának időtartamát határozza meg milliszekundumban, a második eleme pedig a kormányzás (*steering*) értéke.



Tehát a 10 szám, összesen 5 mozgást határoz meg.

A például:

Először a robot 1000 ms-ig mozog egyenesen előre, mert a *steering* értéke 0. Ezután a robot 1500 ms-ig helyben forog, mert a *steering* értéke -100. Ezután a robot 2000 ms-ig kanyarodik az ellenkező irányba (nem helyben forog), mert a *steering* értéke: 30. ...

F.2.3.11. Feladat – Távolságmérés és minimum kiválasztás

Szint: 

Írj programot, amelyet végrehajtva a robot egyenesen előre indul mindaddig, amíg ultrahang szenzora 15 cm-en belül akadályt nem érzékel! Ekkor visszatolat 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 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.

Programozási ötletek

F.2.3.1. Akadály eltávolítás

A feladat megoldásához egyszerű kreatív ötlet szükséges. Ha a robot megközelítette a megadott távolságra az akadályt, akkor közelebb megy és egy 360 fokos fordulattal eltávolítja az útvonalról. Így azt is elértük, hogy nem került messze a követendő útvonalától.

Természetesen más megoldások is lehetségesek. Pl.: Megfelelő fordulókkal az akadály mellé megy és eltolja, majd újra visszatér az útvonalra, de az első megoldás a legegyszerűbb.

Mivel nem tudjuk, hogy hány akadály van ezért az útvonalkövetést és a dobozeltávolítást egy újabb ciklusba kell helyezni.

A cél elérését nem kell a robotnak érzékelnie.

F.2.3.2. Kapukeresés

A robot forgás közben először az első dobozt fogja érzékelni (kb. 20 cm-en belül), majd ha tovább forog, akkor a két doboz közötti rést (kb. 30 cm-nél nagyobb mért értékkel), majd a második dobozt (ismét kb. 20 cm-en belül). Ha megmérjük a két doboz közötti rés kezdete és a második doboz észlelése között eltelt időt, akkor ennek az időtartamnak a felével visszafelé forgatva a robotot, kb. a két doboz közötti rés közepéig fordul. A különböző irányú forgások esetén, ha a sebesség és az időtartam azonos, akkor is lehet különbség. Így elképzelhető, hogy korrekciós mozgásra még szükség lehet, de ez csak tapasztalati úton határozható meg. Eltérést okoz az is, hogy a rés és a második doboz elérésekor a robot már mozgásban volt, míg a visszafordítás álló helyzetből történik, így a gyorsuláshoz szükséges idővel kalkulálni kellene.

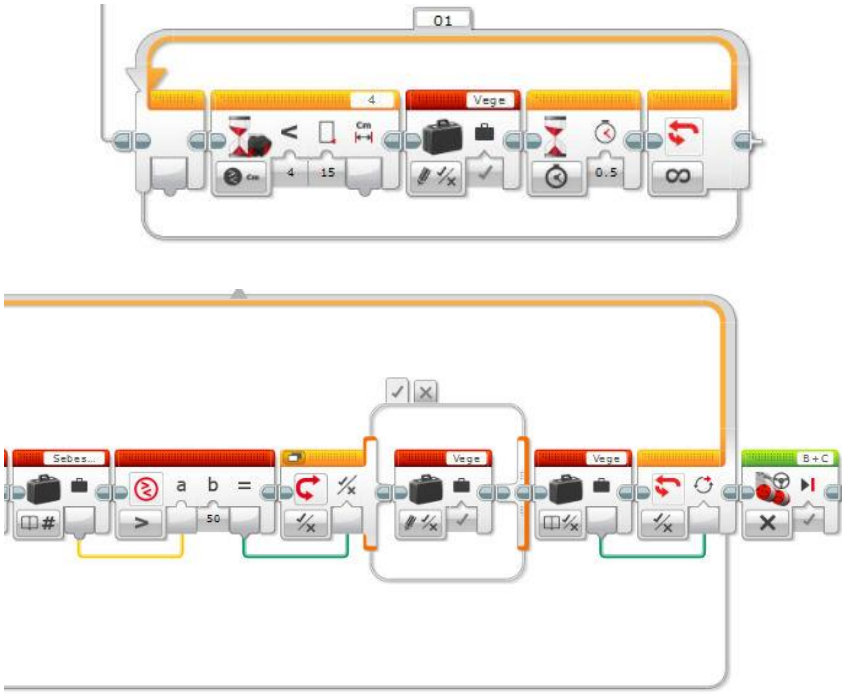
F.2.3.3. Mozgásvezérlés (összetett szenzorhasználattal)

A feladat az előzőekben már megismert algoritmusok összeillesztése. Forgás közben akadály keresése, valamint útvonalkövetés kombinációja. Mindezt többször végrehajtva, tehát ciklusba szervezve.

F.2.3.4. Sebességvezérlés

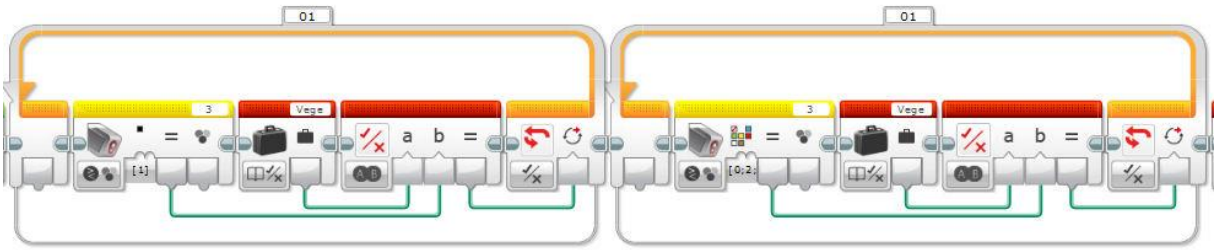
A feladatot többszálú programként célszerű megírni, ugyanis ha a fekete vonalakat a *Wait* utasítással figyeljük, akkor az ultrahangszenzor által mért értéket nem érzékeli a robot. Az egyik szálon figyeljük az ultrahangszenzor értékét, a másik szálon pedig a fekete csíkokat. A csíkfigyelő szálon vezérlejük a motorok sebességét. Minden áthaladáskor növeljük a sebességet 10-zel.

A két szál közötti kapcsolatot egy logikai változó teremti meg. A példánál ez a *Vege* nevet viseli. Kezdetben hamis (*false*) az értéke. Akkor állítjuk át igazra (*true*), ha az ultrahangszenzor 15 cm-en belül akadályt érzékelt (egyik szál), vagy ha a sebesség meghaladta az 50-es értéket (másik szál).



Ha a *Vege* változó értéke igaz, akkor lépünk ki a ciklusból és kezdi meg a robot a tolatást.

A csíkok figyelése is csak abban az esetben történik meg, ha *Vege* változó értéke hamis. Ezt úgy tudjuk elérni, hogy egy összetett kilépési feltétellel szabályozzuk a fekete csík, illetve a csíkon történő áthaladás után a nem fekete terület elérését. A logikai kapcsolat a két feltétel között VAGY.

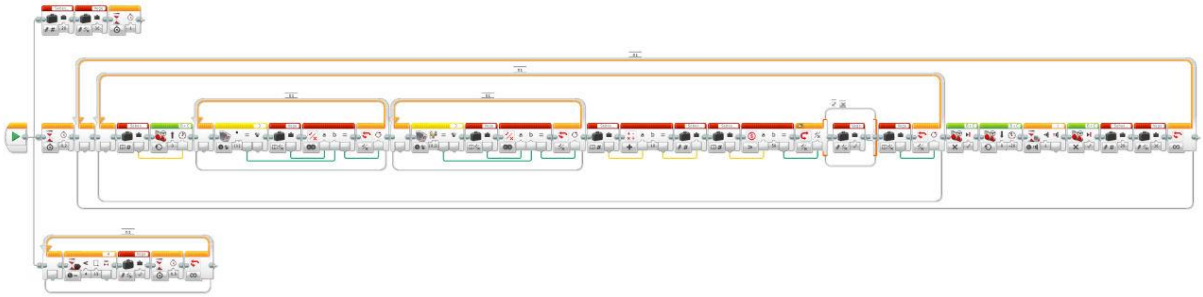


A motor sebességének vezérlésére egy *Sebesseg* nevű változót hoztunk létre, amelynek kezdő értéke 20, és minden csík után 10-zel növeljük az értékét.

Ha a *Vege* változó értéke igazra változik, akkor a robot tolatni kezd ütközésig, majd a változók kezdőértékét visszaállítjuk (*Sebesseg = 20* és *Vege = false*). Mindkét szálon futó utasításokat végtelen ciklusba tesszük.

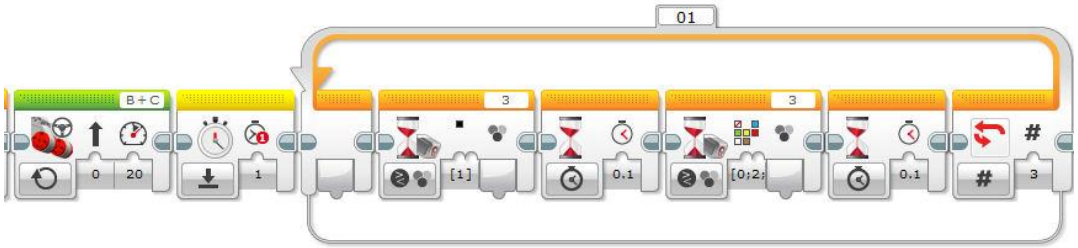
A programban egy harmadik szálon állítjuk be a két változó kezdő értékét.

A teljes forráskód az áttekintés miatt:

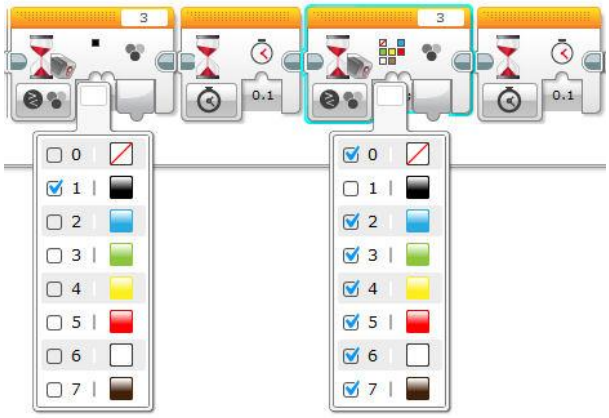


F.2.3.5. Mozgásvezérlés (számlálással)

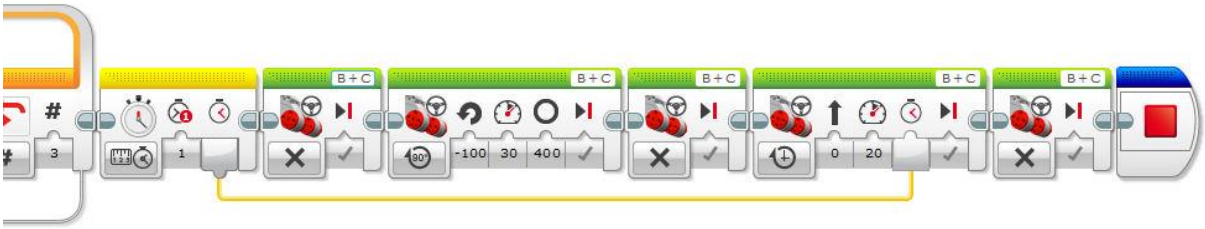
a) A csíkokon történő áthaladás érzékelésére a szokásos algoritmust használjuk a *Wait* blokkokkal, mivel tudjuk előre, hogy 3 csíkon kell áthaladni, ezért egy háromszor lefutó ciklusban.



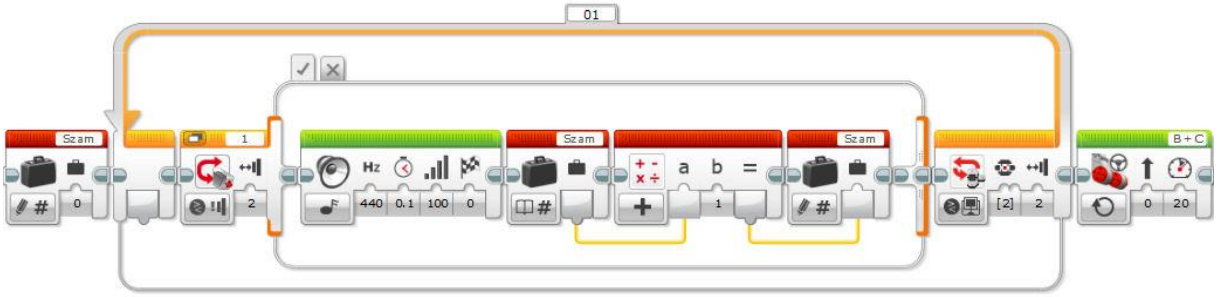
A csík határainak elérését most színszenzorral érzékeljük. A csík kezdetét fekete színként, míg a végét nem fekete színként.



Mivel a robotnak vissza kell térnie a kezdő pozícióba, ezért egy stopperrel mérjük az indulástól eltelt időt, majd a visszafordulás után ennyi ideig működtetjük a motorokat (ugyanazzal a sebességgel).

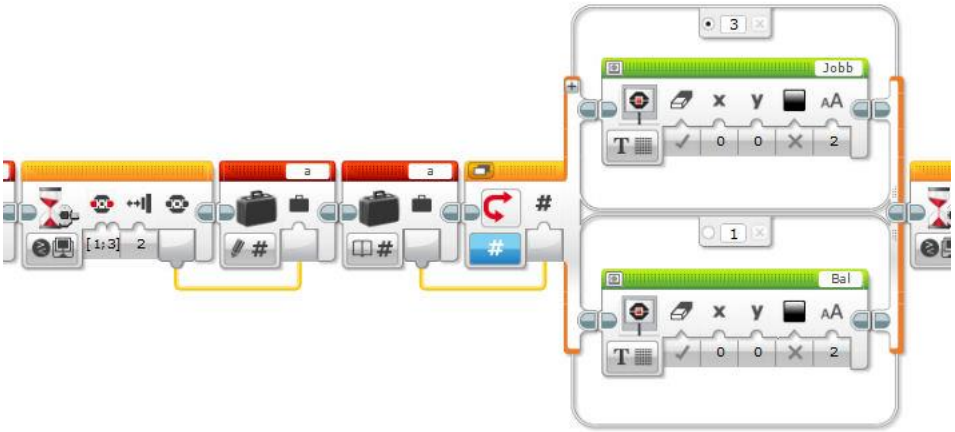


b) Az a) részben megoldott feladattól annyiban különbözik, hogy ütközésérzékelővel kell beállítani a csíkok számát, amelyek fölött a robotnak át kell haladnia. Így az a) részben megírt forráskód módosításával végezzük el a programozást. A beállított számérték tárolására létrehoztunk egy *Szam* nevű változót. Ennek az értékét változtatjuk egy cikluson belül, ha az ütközésérzékelőt benyomtuk. A ciklus ENTER-ig fut. A *Wait* blokkot nem használhatjuk, mert a robot nem venné észre, ha benyomtuk az ENTER-t és a ciklus nem állna le.



F.2.3.6. Mozgásvezérlés (nyomógombokkal) 1.

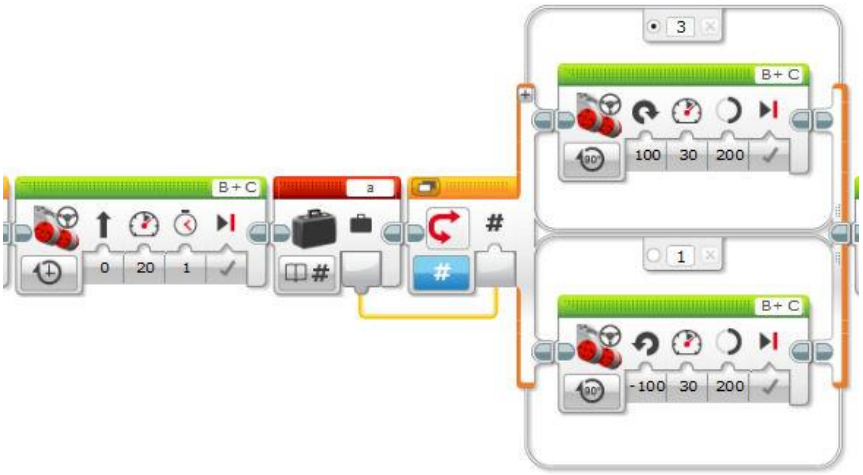
A nyomógombok és a feliratok kezelését a következő programrészlet szemlélteti:



Minden nyomógombnak van egy számszerű értéke. A „jobbra” gomb a 3-as, míg a „balra” gomb az 1-es értékkel rendelkezik. Ezt az értéket tároljuk el egy változóban. Egy numerikus értékkel vezérlet elágazásban be tudjuk állítani, hogy milyen szám esetén hajtódjanak végre az utasítások, így a változóban tárolt értéknek megfelelően íratjuk a képernyőre a „Jobb” vagy „Bal” szavak valamelyikét. A három gombnyomásnak összesen három változót hoztunk létre, így a

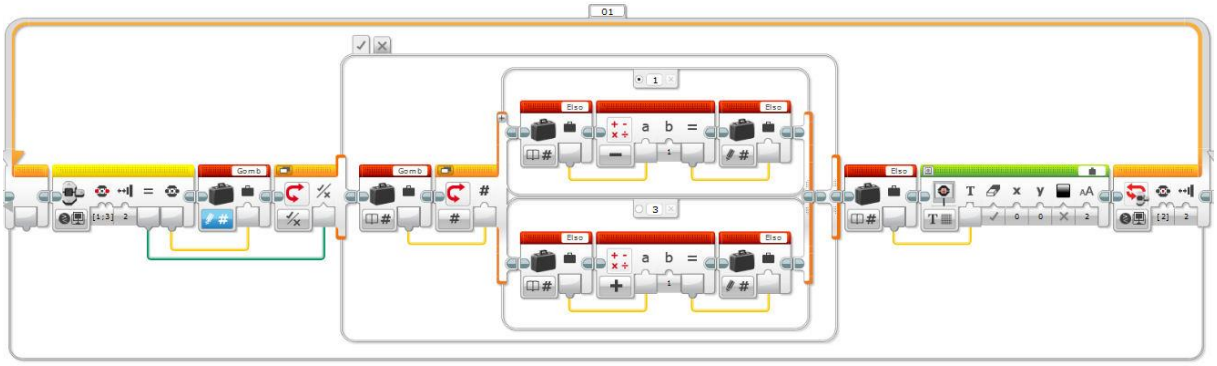
programrészletet háromszor ismételjük (nem ciklusban) a három változóra. Ha tömböt használunk, akkor ciklussal is megoldható a feladat, és a ciklusváltozó lehet a tömb indexe.

A mozgássor vezérléséhez ugyanezt a technikát használhatjuk.



F.2.3.7. Mozgásvezérlés (nyomógombokkal) 2.

A nyomógombokkal történő számlálás algoritmusához kihasználjuk, hogy minden nyomógombnak van egy számértéke. A jobb oldali gomb az 1-es, míg a bal oldali gomb a 3-as. Egy ciklust futtatunk ENTER benyomásáig. A cikluson belül azt vizsgáljuk, hogy történt-e 1-es vagy 3-as gomb benyomás. Ha igen, akkor eggyel növeljük vagy csökkentjük a létrehozott változó értékét.



Az ENTER benyomása után hasonló ciklust futtatunk a másik mozgás idejének beállításához, de természetesen más változót használva.

A mozgás vezérlése már egyszerű:

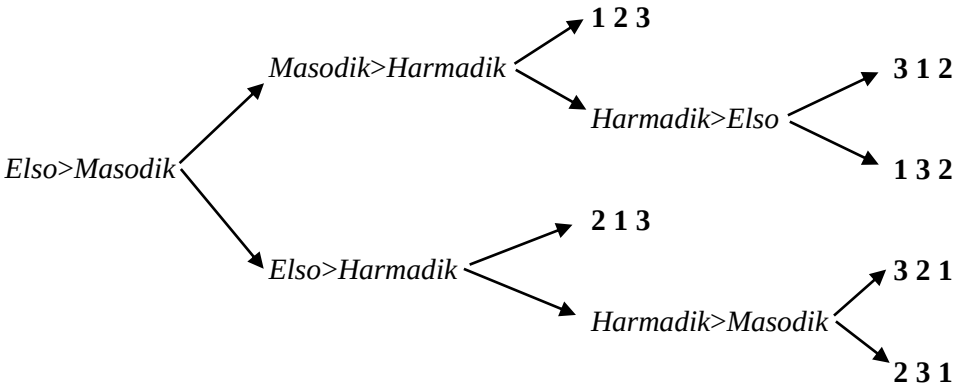
F.2.3.8. Sorba rendezés (csíkszélesség alapján)

A csíkok szélességének tárolásához létrehozunk három változót (tömböt is használhatunk, de lényegesen nem rövidíti le a programot).

A szélessége mérése a szokásos módon a stopperrel történik:

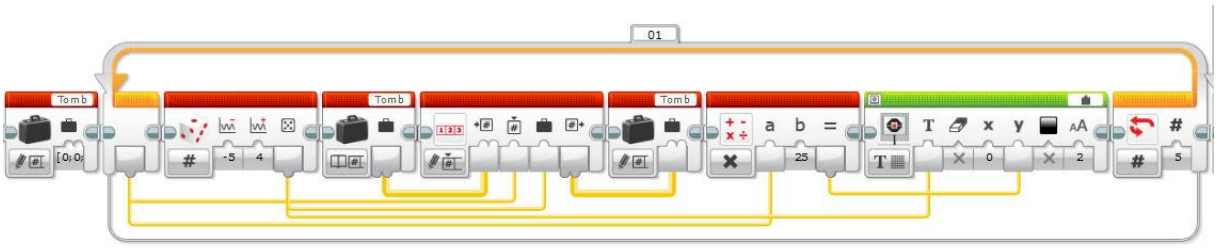
A másik két csíkra hasonlóan, de más változókban tárolva az értéket.

A három csík sorrendjére összesen 6 féle lehetőség van. Ehhez összesen 5 egymásba ágyazott elágazást használunk. A felépítést szemlélteti a következő táblázat, ha a változók rendre *Elso, Masodik, Harmadik*. A felső nyíl jelenti mindig az igaz ágat.

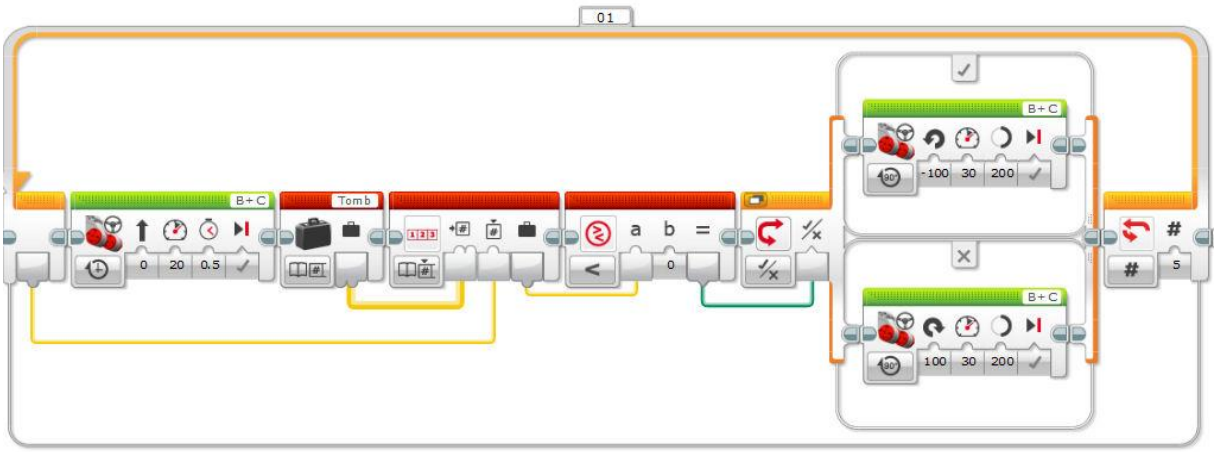


F.2.3.9. Mozgásvezérlés (véletlen számok alapján)

A számok tárolását egy 5 elemű tömbben végezzük egy cikluson belül. A tömb indexe (hányadik helyen tárolja az adott értéket a tömbben a program) a ciklusváltozó lesz (a tömb indexelése 0-val kezdődik). A cikluson belül rögtön ki is íratjuk a sorsolt számot, egymás alatti, 25 pixel magas sorokba.



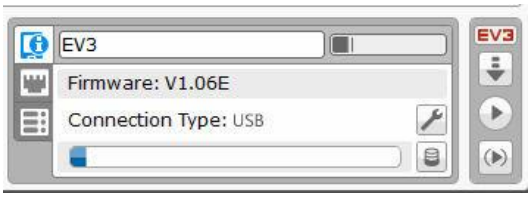
Egy újabb 5-ször lefutó ciklusban sorban kiolvassuk a tömbben tárolt számokat, és előjelüktől függően balra vagy jobbra fordulunk a robottal (a 0-t esetén szintén jobbra).



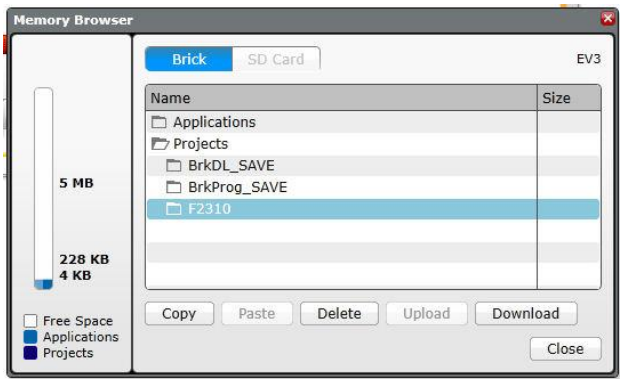
F.2.3.10. Mozgásvezérlés (fájlban tárolt adatok alapján)

Legyen a fájl neve *adatok.txt* illetve *adatok.rtf*. A fájlt bármilyen szövegszerkesztő programmal létrehozhatjuk, de normál szövegfájlként (*txt*) kell mentenünk. EV3 robot esetén az elkészített szövegfájl kiterjesztését *rtf*-re módosítva használhatóvá válik.

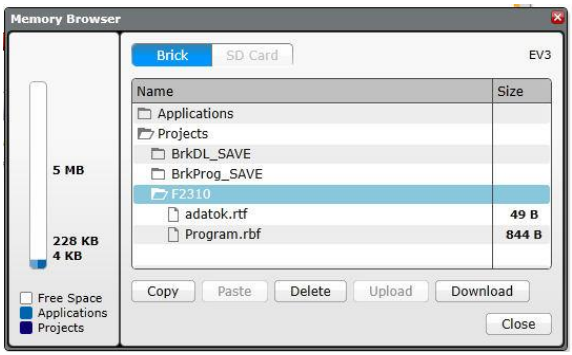
A fájl feltöltése a robotra. Válasszuk ki a szoftver jobb alsó sarkában lévő panel *Brick Information* lapján az *Open Memory Browser* funkciót (hordó szimbólum)!



A megjelenő panelen válasszuk ki az aktuális projektet! Ekkor aktív lesz a *Download* feliratú gomb.



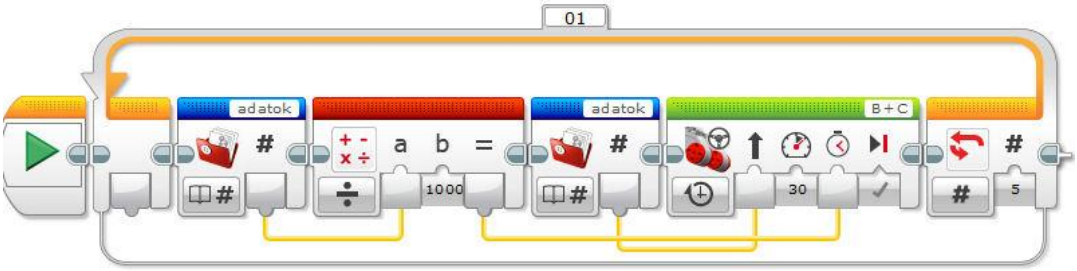
Erre kattintva a megnyíló fájlkezelőben tallózással kiválaszthatjuk a feltöltendő fájlt, ami a sikeres művelet után meg is jelenik az aktuális projektben a roboton.



Az EV3 robot esetén a motorvezérlés másodpercben értelmezi a megkapott adatot, tehát ha a fájlban milliszekundumban adtuk meg az előrehaladás időtartamát, akkor a beolvasott értéket osztanunk kell 1000-rel.

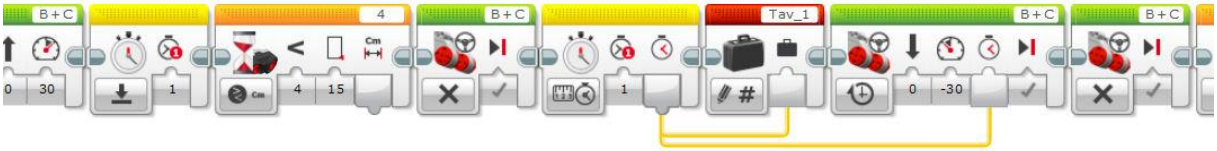
Ügyeljünk arra, hogy az *Advanced/File Access* blokk esetén a kiterjesztést ne írjuk be a modul jobb felső sarkában lévő szövegdobozba, csak a nevet.

A program forráskódja:



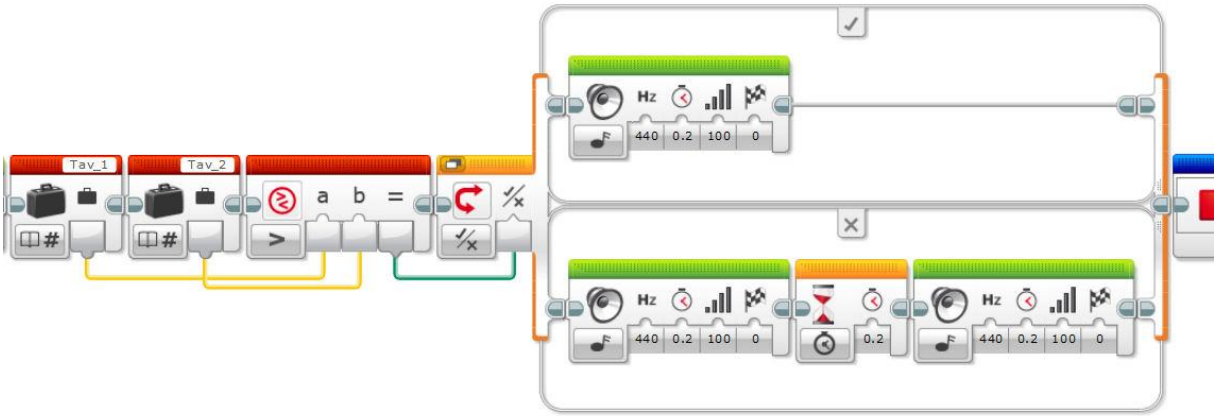
F.2.3.11. Távolságmérés és minimumkiválasztás

A távolság mérése történhet a stopperrel, ha a robot egyenletes sebességgel halad, akkor az eltelt idő és a megtett út egyenesen arányos lesz. Az akadály észlelésekor mért időt tároljuk egy változóban, majd a hátrafelé mozgást is ugyanezzel az értékkel szabályozzuk.



Mindezt 5 másodperc várakozás után megismételjük és egy másik változóban tároljuk az új mért értéket.

Miután a robot visszatolatott a két változó tartalmát összehasonlítva egy elágazás két szálán tudjuk a lejátszandó hangokat utasításokkal kódolni.



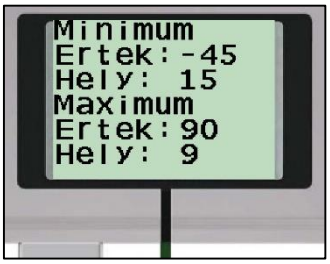
3. VERSENYFELADATOK

3.1. FÁJLKEZELÉS

F.3.1.1. Feladat – Szélsőérték keresés

Szint: 

Írj programot, amelyet végrehajtva a robot egy fájlban tárolt egész számok közül kiírja képernyőre a legkisebb és a legnagyobb számot! A program azt is írja ki, hogy a fájlban hányadik szám a legkisebb illetve legnagyobb! Ha több azonos szám is van, akkor elegendő egyet kiírnia.



A fájlban minden szám külön sorban szerepel. A fájl első adata a tárolt és vizsgálandó számok száma. A megjelenítés az ábra szerinti legyen! A fájl neve: *Adatok_1.rtf* vagy *Adatok_1.txt*.

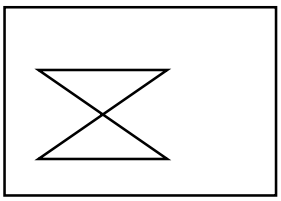
F.3.1.2.a-b Feladat – Rajzolás tárolt koordináták alapján

a) Szint: 

Adatok:

5
10
10
50
50
10
50
50
10
10
10

Képernyőkép:



Írj pont összekötögető játékot! Egy állományban az összekötendő pontok *x* és *y* koordinátáit tároljuk. Olvasd be a pontok koordinátáit, és a fájlban tárolt sorrendnek megfelelően kösd össze őket szakaszokkal! Minden pont a robot képernyőjének belső pontja.

A fájl első adata a tárolt pontok száma. Minden koordináta külön sorban szerepel *x*; *y* sorrendben. A fájl neve: *Rajz_1.txt* illetve *Rajz_1.rtf*. A képernyőre rajzolt kép ütközésérzékelő benyomásáig maradjon látható! Egy példát láthatsz az ábrán.

b) Szint: 

A kapott képet nagyítsd fel a kétszeresére! A kép területe 4-szerese legyen az eredetinek. (Ha a képet egy téglalapba foglalnád, akkor a sor és oszlop mérete is kétszerese legyen az eredetinek.)

F.3.1.3.a-b Feladat – Rajzolás futamhossz kódolás alapján

a) Szint: 

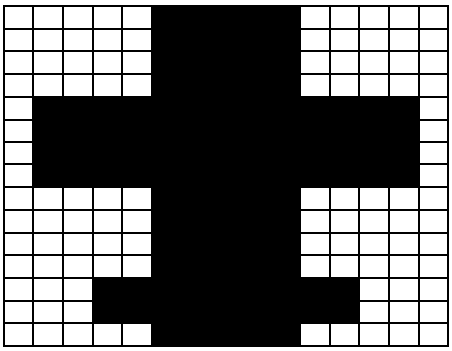
A robot feladata, hogy egy szövegfájlban (*Rajz_2.txt* vagy *Rajz_2.rtf*) tárolt képet megjelenítsen a képernyőjén. A kép pixeleinek színét (fekete-fehér) tartalmazza a fájl. A kép négyzetes méretű, tehát a sorok és oszlopok száma megegyezik. Ezt a számot a fájl első adata tartalmazza. (A mellékelt példafájl esetén ez a szám 50. Tehát a kép 50x50 pixel méretű.)

Az adattárolás kódolási algoritmusa:

A kép bal felső sarkától indulva a fehér illetve fekete színű összefüggő pixelek száma váltakozva van tárolva a fájlban, soronként újra kezdve a leírást. A fájlban minden szám külön sorban szerepel. Minden képsor fehér színű pixellel kezdődik.

A kép maradjon a képernyőn ütközésérzékelő benyomásáig!

Például:

> A bal felső saroktól indulva a fehér és fekete színű összefüggő pixelek száma váltakozva, soronként újra kezdve a leírást. A kép (15x15):
>
> 
>
> A fájl tartalma (a fájlban az adatok külön sorokban szerepelnek):
>
> 15 5 5 5 5 5 5 5 5 5 5 5 5 1 13 1 1 13 1 1 13 1 1 13 1 5 5 5 5 5 5 5 5 5 5 5 3 9 3 3 9 3 5 5 5

b) Szint: 

A kapott képet nagyítsd fel a kétszeresére! A kép területe 4-szerese legyen az eredetinek, tehát 100x100-as. A pontok helyett 2 pixel sugarú köröket rajzolj a képernyőre!

F.3.1.4.a-b Feladat – Rajzolás tárolt pozíció alapján

a) Szint: 

A robot feladata, hogy egy szövegfájlban (*Rajz_3.txt* vagy *Rajz_3.rtf*) tárolt képet megjelenítsen a képernyőjén. A fájlban pozitív egész számok találhatók. Minden szám új sorban van. A programírás során ismert a fájlban található adatok száma és a kép mérete. Ezek az adatok a fájl első három sorában szerepelnek, adatok száma, sorhossz (pixel) és sorok száma (pixel). A számok a fekete pixelek helyét jelölik sorfolytonos számozással. A kép bal felső pixele kapta az 1-es sorszámot.

Például:

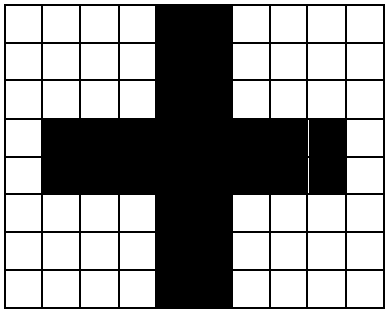
A fekete pixelek száma a példánál (a fájlban található adatok száma): 28

A sorok hossza: 10

A sorok száma: 8

A fájl tartalma és a kódolt kép a példánál (a fájlban minden szám külön sorban szerepel):

28 10 8 5 6 15 16 25 26 32 33 34 35 36 37 38 39 42 43 44 45 46 47 48 49 55 56 65 66 75 76



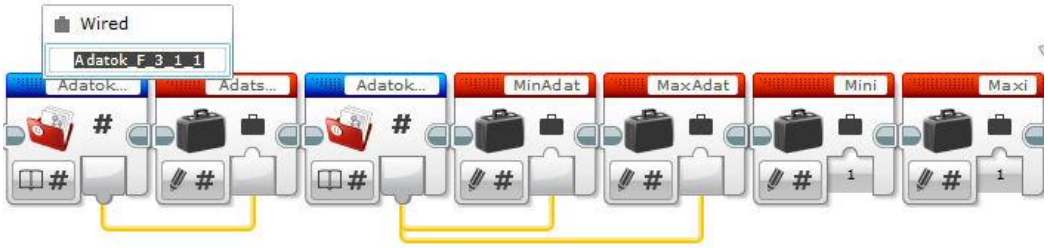
b) Szint:

A kapott képet nagyítsd fel a kétszeresére! A kép területe 4-szerese legyen az eredetinek, a téglalap alakú kép sorainak és oszlopainak a hossza is legyen az eredeti kétszerese! A pontok helyett 2 pixel sugarú köröket rajzolj a képernyőre!

Programozási ötletek

F.3.1.1. Szélsőérték keresés

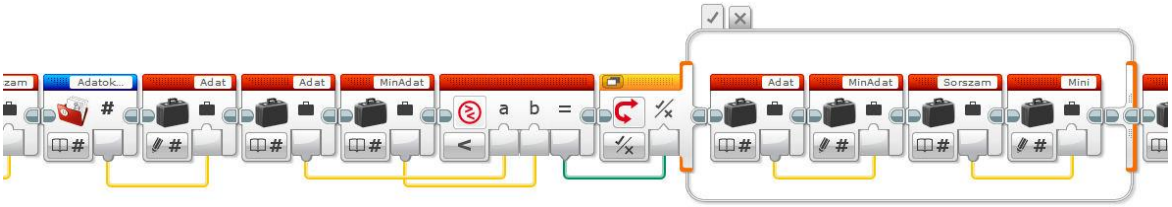
Az első lépésekben a megfelelő változókat hozzuk létre és a fájlból olvassuk be a kezdő adatokat!



Létrehoztunk egy *Adatszam* nevű változót, ebbe olvastuk be a fájl első adatát, mivel ez jelentette az adatok számát. Kezdetben a legkisebb és legnagyobb elem is az fájlban tárolt első adat. Ezért létrehoztunk egy *MinAdat* és egy *MaxAdat* nevű változót, amelybe beolvastuk a fájl második adatát. A *Mini* és *Maxi* változók tárolják majd azt az értéket, hogy a legkisebb illetve legnagyobb elem hányadik volt a fájlban. A kezdő értékük 1, hiszen aktuálisan az első elem a szélsőérték (a fájl legelső adatát, ami a számok számát jelenti nem számoljuk az adatok közé).

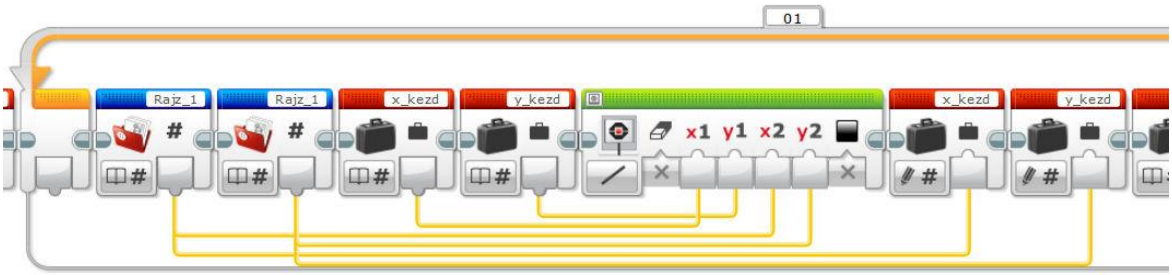
Ezután következhet az algoritmus, amelynek a legkisebb kiválasztására vonatkozó ötlete, hogy beolvasunk egy adatot, majd összehasonlítjuk az aktuálisan legkisebbel (*MinAdat*), ha kisebb, akkor ezt tároljuk a *MinAdat* változóban, egyébként nem csinálunk semmit. Ha a beolvasott adat kisebb volt, mint az aktuálisan legkisebb, akkor a *Mini* változó értékét is módosítani kell. Ehhez a ciklusváltozót használhatjuk (a ciklusváltozó értékénél kettővel nagyobb értéket kell tárolni, hiszen

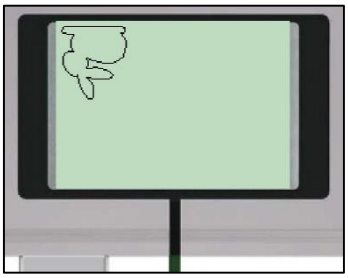
a ciklusban a 2. adatot olvastuk be először). A ciklus kilépési feltételét az *Adatszam* változó határozza meg (eggyel kisebb).



F.3.1.2. Rajzolás tárolt koordináták alapján

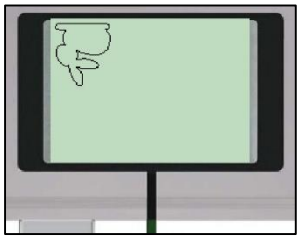
a) Első lépésben olvassuk be a fájlból az adatok számát (*Adatszam* változó), valamint az első szakasz kezdőpontjának koordinátáit, tehát az első koordinátapárt (*x_kezd* illetve *y_kezd* változók).
Innentől kezdve egy ciklusban tudjuk beolvasni a következő képpont koordinátáit, amelyek a szakasz végpontját adják meg. Felrajzoltatva a szakaszt a beolvasott értékekkel felül kell írni az *x_kezd* és *y_kezd* változók értékeit.

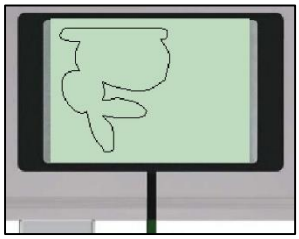




Mindezt az *Adatszam* változóba beolvasott értéknél eggyel kevesebbszer kell végrehajtani, hiszen az első pont koordinátáit a cikluson kívül olvastuk be.
A mellékelt fájlban tárolt adatok alapján a képernyőképet az ábra mutatja.
EV3 robot esetén a kép fejjel lefelé látható, mert a képernyő koordináta rendszerének origója a bal felső sarok.

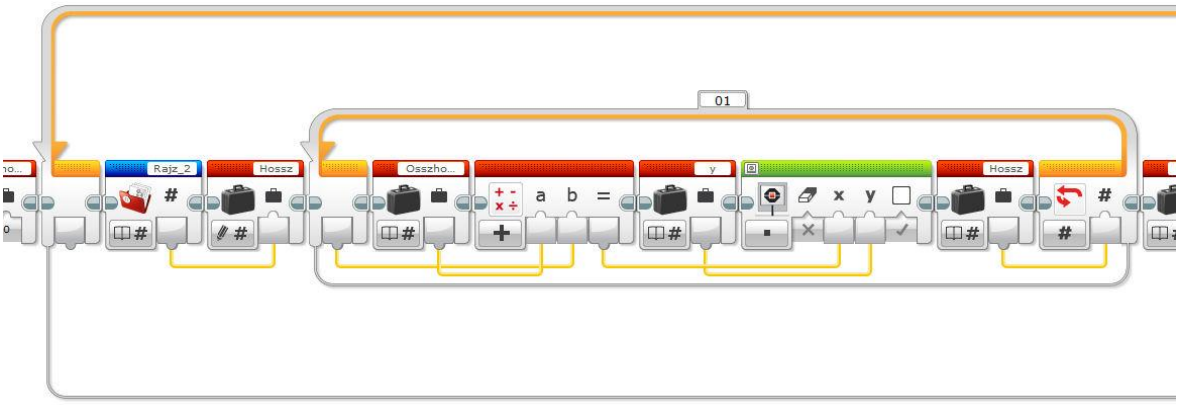
b) A kétszeres nagyításhoz a beolvasott koordinátákat szorozzuk 2-vel a rajzolás előtt.
Az eredeti és a nagyított kép képernyőn megjelenő képei:



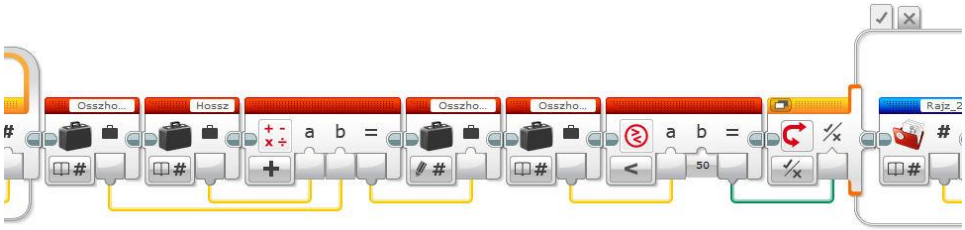


F.3.1.3. Rajzolás futamhossz kódolás alapján

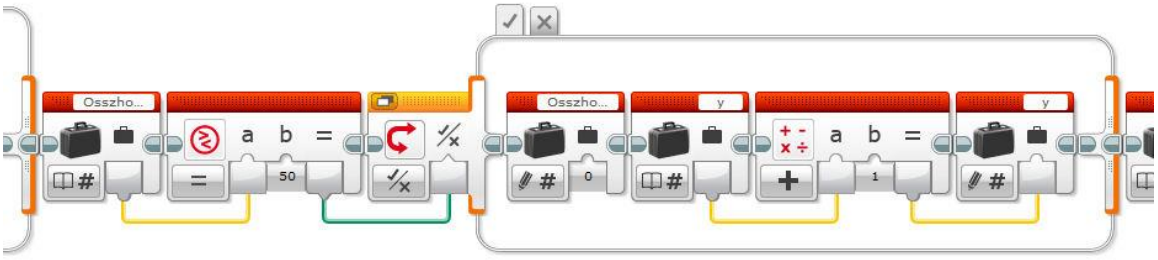
a) A megjelenítéshez beolvassuk a fájl első adatát a *Hossz* változóba, majd egy ciklust futtatunk a beolvasott értéknek megfelelően és pixelenként fehér (háttér színe) színnel rajzolunk pontokat a képernyőre. Közben szükséges azt is tárolni, hogy a soron belül éppen hányadik pixelnél tartunk a kirajzolással, így az *Osszhossz* nevű változóban ezt folyamatosan összegezzük.



Az *y* változó a rajzolás sorának koordinátája, amelyet majd eggyel növelünk, ha elértük a sor végét. A ciklusból kilépve az *Osszhossz* változó értékét és az aktuális *Hossz* értékét összegezve meg kell vizsgálni, hogy elértünk-e már a sor végére (kisebb-e az *Osszhossz* értéke mint 50). Ha még nem, akkor jöhet a fekete színnel történő rajzolás. Teljesen analóg módon, mint a fehér színnel történő.

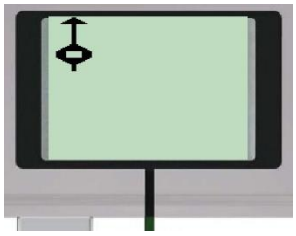


Ha az *Osszhossz* változó értéke elérte az 50-et (ez volt a sorhossz), akkor növelni kell eggyel az *y* értékét, hogy a további rajzolás már új sorban kezdődjön és vissza kell állítani az *Osszhossz* változó értékét 0-ra, hogy a sor elejétől rajzoljon a program.



A ciklus addig fut, amíg az y változó eléri az 50-et.

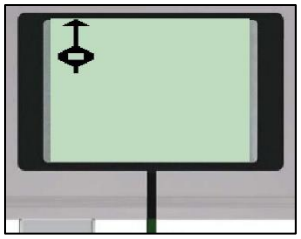
A feladat megoldható szakaszok rajzolásával is, ekkor a szakasz hosszát jelenti a beolvasott adat és váltakozva kell fehér illetve fekete színű szakaszokat rajzolni. A sorok végének elérését hasonlóan kell számolni, és ha elértük az 50-et, akkor új sorban kell folytatni a rajzolást.

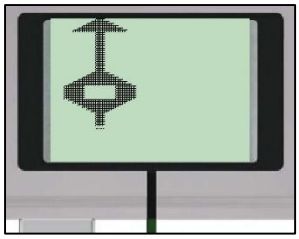


A mellékelt fájl adatai alapján, a képernyőn megjelenő képet az ábra mutatja egy EV3 robot képernyőjén.

b) A kétszeres nagyításhoz a beolvasott koordinátákat szorozzuk 2-vel a rajzolás előtt.

Az eredeti és a nagyított kép képernyőn megjelenő képei:





F.3.1.4. Rajzolás tárolt pozíció alapján

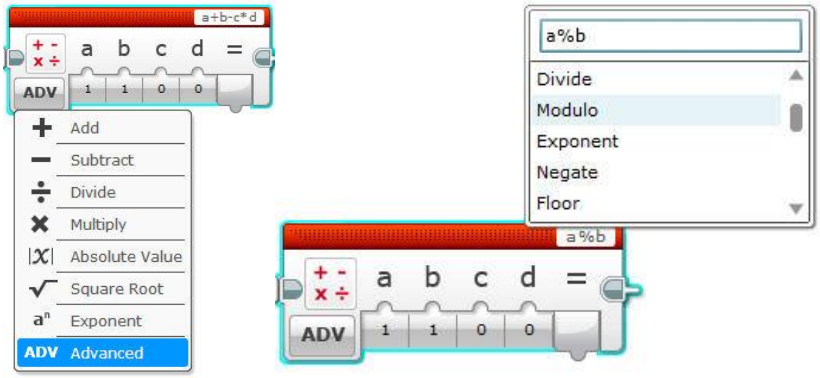
a) A megoldást a bevezető értékek beolvasásával kell kezdeni, ezek rendre kerülhetnek az *Adatszam*, *Oszlopszam*, *Sorszam* nevű változókba. Majd ezek után egy ciklussal történik a rajzolás.

A fájlban a fekete pixelek sorszámát rögzítettük. Ebből egyszerű matematikával kiszámolható a sor illetve oszlop koordináta. A sorszámot elosztva a sor hosszával a maradék megadja a soron belüli *x* koordinátát, míg a hányados egész része az *y* koordinátát.

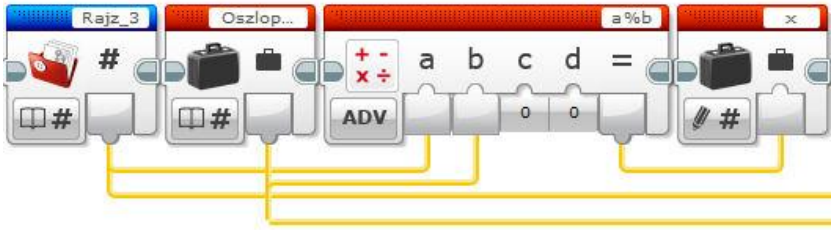
Például ha a fekete pixel sorszáma 2063 és a sorhossz 81 pixel, akkor 2063 : 81 → hányados: 25; maradék: 38. Tehát a fekete pixel a 38. oszlop 25. sorában van. Koordinátái: (38; 25).

Mivel a számozás 1-gyel kezdődik és a 81-gyel történő osztás 0…80 közötti maradékot adhat, ezért ha a maradék 0, akkor azt a pixelt a 81. oszlopba kell rajzolni és nem a nulladikba (a képernyő oszlopainak a számozása is 0-val kezdődik).

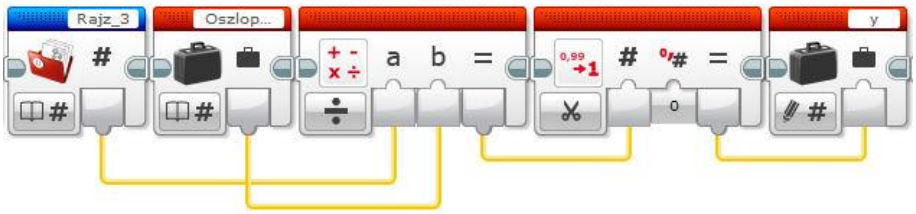
Az osztási maradék képzésére a programban rendelkezésre áll egy speciális blokk. A *Data Operetion* csoport *Math* blokkjának *Advanced* beállítása.



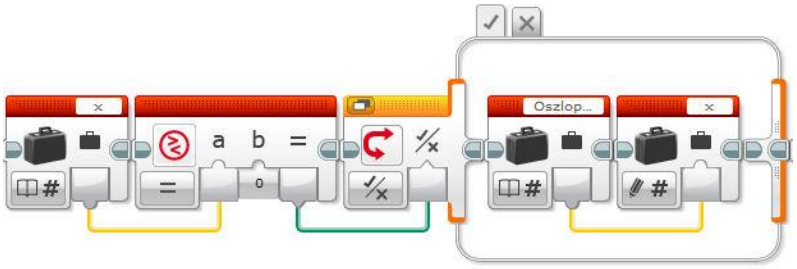
Az *Advanced* funkciót kiválasztva a blokk jobb felső sarkára kattintva a legördülő listából választhatunk a matematikai függvények közül. A *Modulo* függvény az osztás utáni maradékot adja vissza. A blokknak négy bemeneti (szám típusú) paramétere lehet. Ezek közül kettőre van szükségünk. Az *a* csatlakozási pontba kötjük az osztandót (a fájlból beolvasott számot), míg a *b* csatlakozási pontba az osztót (a sorok hosszát, a fájlban szereplő második számot).



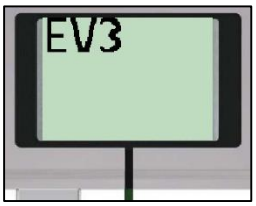
Az osztás utáni hányados egészrészének meghatározásához végezzük el az osztást, majd a hányadosból vágjuk le a tizedesrészt a *Truncate* funkcióval, amelyet 0 tizedes jegyre paraméterezünk.



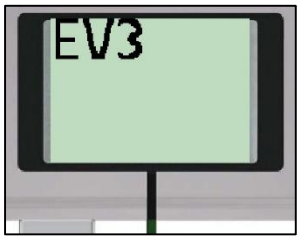
Az így megkapott koordináták helyére rajzolunk fekete színű pontokat. Ne felejtsük el beállítani a rajzolás előtt, hogy ha a maradék 0, akkor az *x* értéke legyen egyenlő az *Oszlopszam* változóba beolvasott értékkel. Egyébként az ábra pixelhibásan jelenik meg.

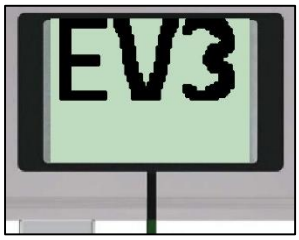


A mellékelt fájlban található adatok alapján, a képernyőn megjelenő képet az ábra mutatja egy EV3 robot képernyőjén.



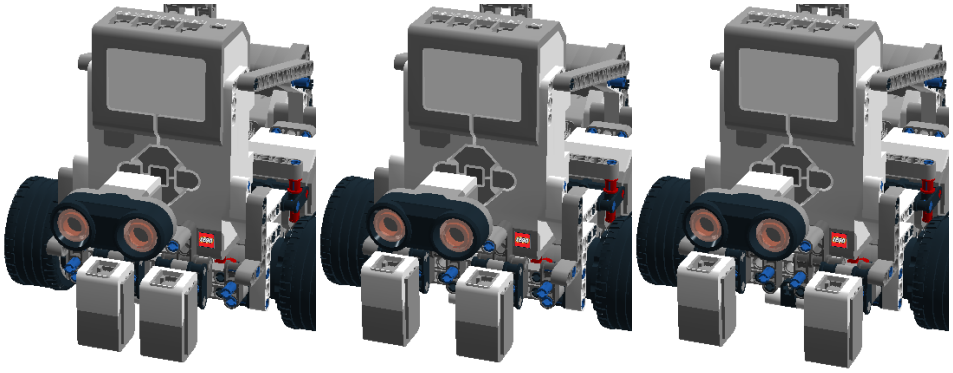
b) A kétszeres nagyításhoz a beolvasott koordinátákat szorozzuk 2-vel a rajzolás előtt.
Az eredeti és a nagyított kép képernyőn megjelenő képei:





3.2. TÁJÉKOZÓDÁS

A fejezetben található feladatok megoldásához az alaprobot átépítése szükséges. Két fény/szín szenzort kell rajta elhelyezni, egymással párhuzamosan, lefelé néző állapotban. Az adott feladat függvényében a fény/színszenzorok távolsága változtatható.



F.3.2.1. Feladat – Koordináta rendszer 1.

Szint:

A robot feladata, hogy egy koordináta-rendszerben bejárjon egy adott útvonalat. Az útvonal jellemzőit fájlban tárolt számok formájában kapja meg. A fájl neve: *Utvonal.txt* illetve *Utvonal.rtf*.

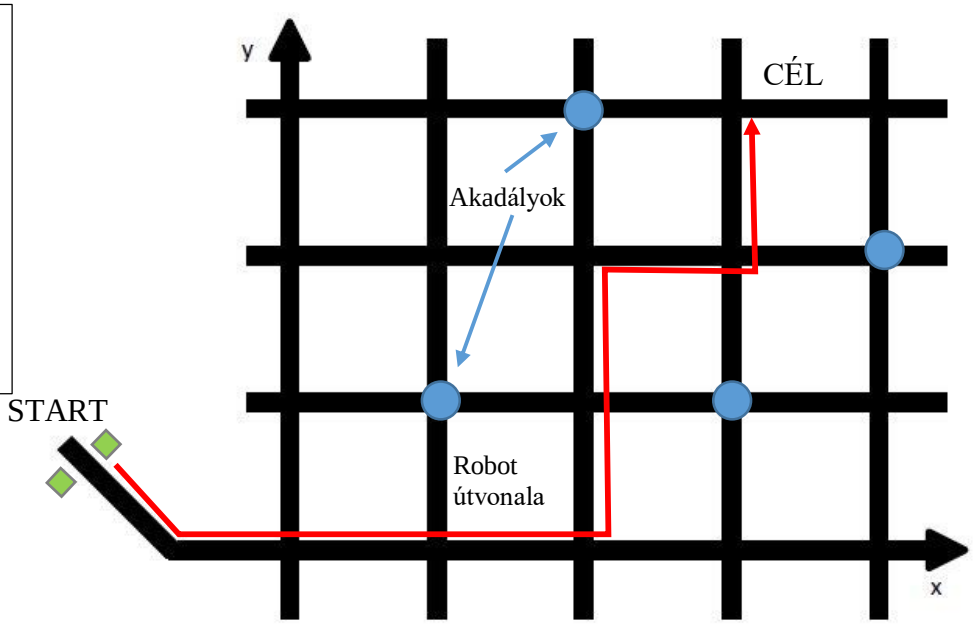
A fájl felépítése: A fájlban egyjegyű számok szerepelnek, minden szám külön sorban. A számokat kettesével kell értelmezni. Az első szám megadja az előre haladás csomópontjainak számát (tehát hányadik kereszteződés után kell fordulni), a második szám megadja a fordulás irányát. A fordulás irányának kódjai: 0 → a robot álljon meg (program vége); 1 → balra forduljon 90 fokot; 2 → jobbra forduljon 90 fokot.

A robot startpozícióból indul. A start pozíció egy fekete vonal, amely a koordináta-rendszer *x*-tengelyéhez vezet, de egyik tengellyel sem zár be 90 fokos szöget (lásd ábra). A robot két színszenzora a vonal két oldalán helyezkedik el (az indulás után nem szükséges a vonalat úgy követni, hogy a két színszenzor a vonal két oldalán helyezkedjen el).

A koordináta rendszer bizonyos koordinátáinál (vonalkereszteződéseinél) akadályok vannak elhelyezve (tartógyűrűn álló golyók). Ha a robot a koordináta rendszerben történő mozgása során lelöki a tartógyűrűről a golyót, akkor hibapontot kap. Ha a fájlban tárolt útvonalat helyesen értelmezi a robot és az ott meghatározott utat járja be, akkor a golyókat nem fogja érinteni.

Például:

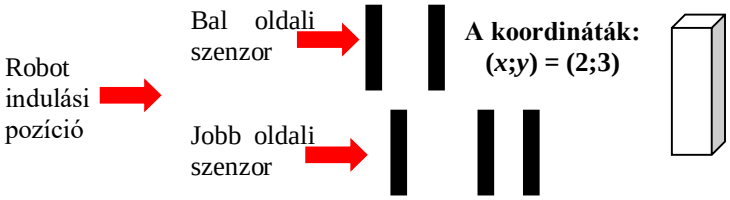
A fájlban tárolt számok:
3
1
2
2
1
1
1
0



F.3.2.2. Feladat – Koordináta rendszer 2.

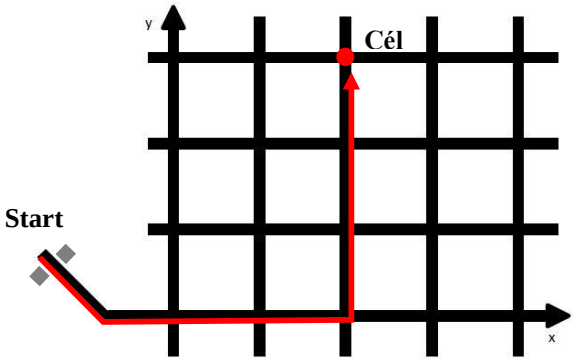
Szint:

A robot feladata, hogy egy koordináta-rendszerben megtalálja a megfelelő koordinátájú pontot. A pont (*x*; *y*) koordinátáját egy-egy fekete színű vonalsor kódolja. A robot áthalad a vonalak felett úgy, hogy a két fény/szín szenzora egy-egy vonalsor fölött mozog. A bal oldali szenzor alatti vonalsor felel meg az *x*, míg a jobb oldali az *y* koordinátának. A vonalak száma adja meg az elérendő pont *x* illetve *y* koordinátáját. A mozgás végét egy akadály jelzi, amely legalább 10 cm-re van az utolsó vonaltól. Például:



A leolvasott koordinátákat a robot írja ki a képernyőjére, egymás mellé, *x; y* sorrendben!

A leolvasott koordináták alapján kell a robotnak a koordináta-rendszer adott pontjára mozognia. A koordináták meghatározása után a robotot áthelyezve a kezdő pozícióba, ütközésérzékelő megnyomására kezdheti meg mozgását, amelyet a leolvasott koordinátáknak megfelelő



helyen kell befejeznie. A megadott pozíció egy fekete vonal, amely a koordináta-rendszer x-tengelyéhez vezet, de egyik tengellyel sem zár be 90 fokos szöget (lásd ábra). A robot két színszenzora a vonal két oldalán helyezkedik el. A megfelelő koordinátát elérve a robotnak meg kell állnia a pozíció felett.

A robot egy lehetséges útvonalát szemlélteti az ábra. Az origóig a robot útja kötött, az ábrán jelzett útvonalon kell haladnia. Az origótól a robot szabadon mozoghat.

F.3.2.3.a-b Feladat – Térképezés (egyenes szakaszok)

a) Szint:

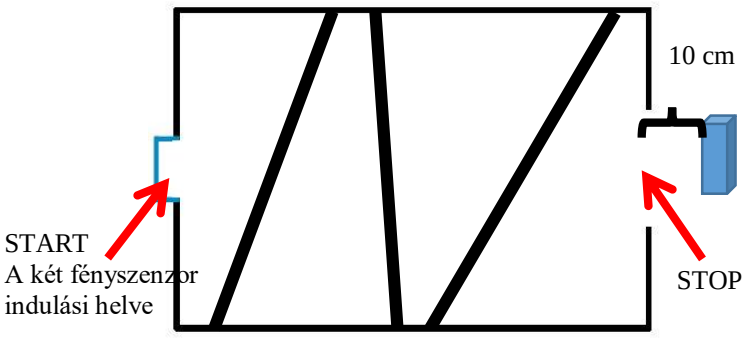
A robot feladata, hogy a képernyőjére rajzolja egy megadott pályaszakasz térképét. A robot a pályán egyszer haladhat végig, a megadott pozícióból indulva egyenesen előre, majd a végighaladás után kell a képernyőjére rajzolnia. A pálya végét akadály jelzi, attól 10 cm-re kell megállnia. A pályán a robot egyenes vonalú haladását keresztező fekete színű egyenes vonalak találhatók (a pálya alapszíne fehér). A vonalak száma három. A vonalak nem feltétlenül merőlegesek a haladási irányra, de töréspontot nem tartalmaznak, és teljes egészében keresztezik a pályát (pályaszéltől-pályaszélig). A vonalak keresztezhetik egymást is, de nem a robot haladási útvonalán. A roboton két fény/szín szenzor található. A két szenzor távolsága 5 cm. A robot a pályán hosszanti irányban haladhat, annak középvonalán és egyszeri végighaladása során vehet mintát a vonalak hálózatából.

A robot képernyőjére rajzolja fel azokat a pontokat, ahol a fényszenzorai először érzékelték a fekete vonalakat. A teljes haladási időtartam felel meg a teljes képernyőszélességnek. A pontok jelölésére 3 pixel sugarú körök használhatók, amelyek középpontja legyen a vonal észlelésének pozíciója. A program ütközés érzékelő benyomására álljon le.

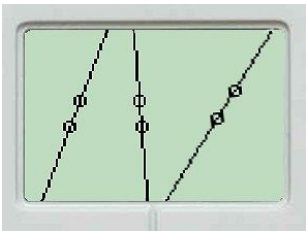
b) Szint:

Ezen pontok alapján az észlelt vonalakat szintén rajzolja a robot képernyőjére 1 pixel vastagságban. A program ütközés érzékelő benyomására álljon le.

A pálya például:



A képernyőkép:



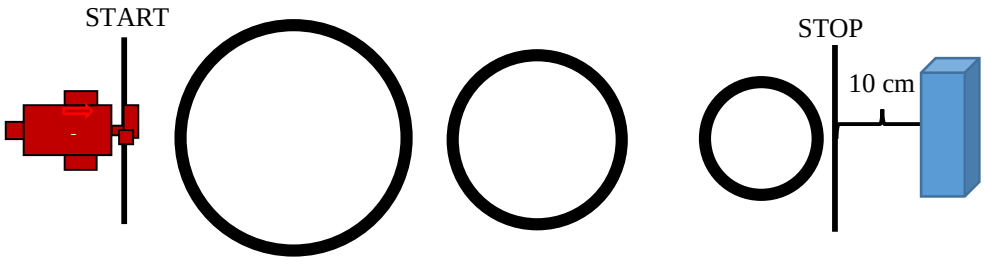
F.3.2.4. Feladat – Térképezés (körök)

Szint:

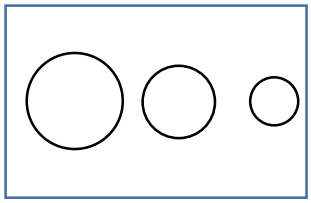
Írj programot, amelyet végrehajtva a robot egyenes vonalban halad egy adott pontról indulva különböző sugarú fekete színű körök fölött! A körvonalak vastagsága legalább 1,5 cm, és a körök nem metszik át és nem is tartalmazzák egymást. A körök középpontjai a robot haladási pályáján helyezkednek el. A robot az egyenes vonalú mozgását akadálytól 10 cm-re fejezze be! Ezután a képernyőjére rajzolja a köröket, a megtett útjának megfelelően arányosan. Három darab kör van a pályán. A program ütközésérzékelő benyomására álljon le! A robot nem fordulhat vissza, csak az egyszeri, egyenes vonalú pályán történő áthaladás során gyűjthet adatokat a körökről. A képernyőre rajzolt körvonalak 1 pixel vastagságúak legyenek.

A feladatot egyetlen fény/szín szenzor használatával kell megoldani.

Például:



Képernyőkép:



F.3.2.5. Feladat – Nem folytonos útvonalkövetés akadályokkal

Szint:

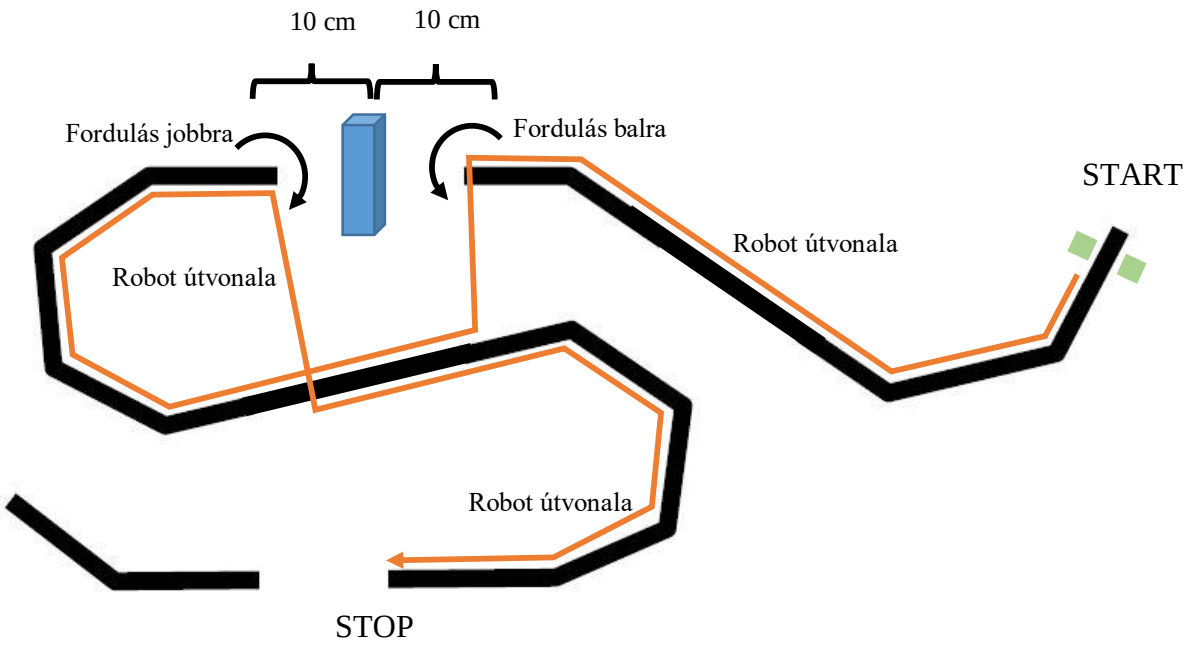
A robot feladata, hogy a startpozícióból indulva kövesse a fehér alapon lévő fekete színű útvonalat (csíkot). A vonalat úgy kell követnie, hogy a két színszenzora az útvonal két különböző oldalán legyen. A robotnak a pályán a haladási irány szerint kell mozognia, nem fordulhat vissza és nem hagyhat ki fekete csíkkal jelölt útvonalrészeket. A haladási irányt az ábra értelmezi.

Az útvonal egy helyen megszakad. A szakadásnál (az útvonal folytatásának vonalában) egy akadály helyezkedik el. Ha a robot ultrahang szenzorával 10 cm távolságon belül érzékeli az akadályt, akkor fordulnia kell, majd áttérnie a másik útvonalra (az ábra értelmezi) és ezt követnie a szakadásig. Itt ismét fordulnia kell és az új útvonal megtalálása után azt követnie a célig. A célt szintén egy akadály jelzi, amelytől kb. 10 cm-re kell megállnia a robotnak.

Az útvonalkövetés során tehát két esetben kell fordulnia és új útvonalra térnie a robotnak. Első alkalommal balra, második alkalommal jobbra történik a forduló.

Az indulási pozícióban a robot két színszenzora az útvonal két oldalán helyezkedik el, a fekete vonal kezdetétől kb. 4-5 cm-re (lásd az ábrán).

A robot útvonalát és haladási irányát értelmező ábra:



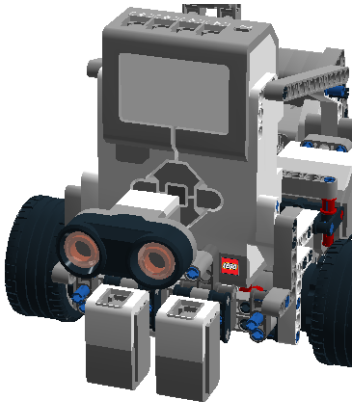
F.3.2.6. Feladat – Radar

Szint:

Írj programot, amely végrehajtása során a robot lassan forog körbe (pl.: 10-es sebességgel) és radarszerűen feltérképezi környezetét! Egy teljes kört forduljon! A fordulás során giroszkóppal mérje az elfordulási szöget és ultrahangszenzorával az előtte lévő tárgyak távolságát. A képernyőn jelenítse meg 1 pixel sugarú körökkel a tárgyak helyzetét. A robot pozíciója a képernyő közepére essen! 1 pixel feleljen meg 1 cm-es távolságnak! Tehát ha tárgyat észlel, akkor tegyen a képernyő megfelelő helyére egy 1 pixel sugarú kört! 0,02 másodpercenként vegyen mintát a környezetéből! A program ütközésérzékelő megnyomására álljon le!

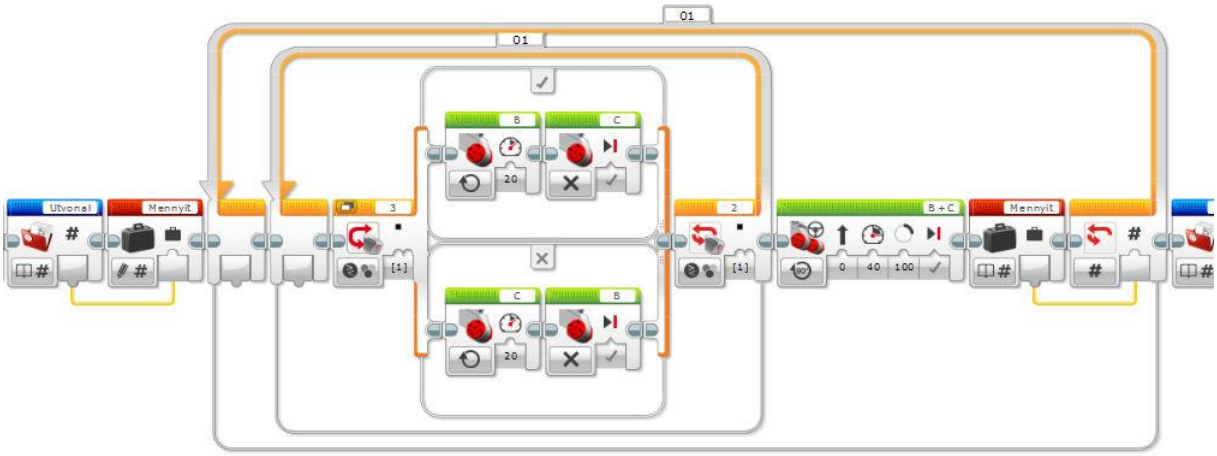
Programozási ötletek

F.3.2.1. Koordináta rendszer 1.

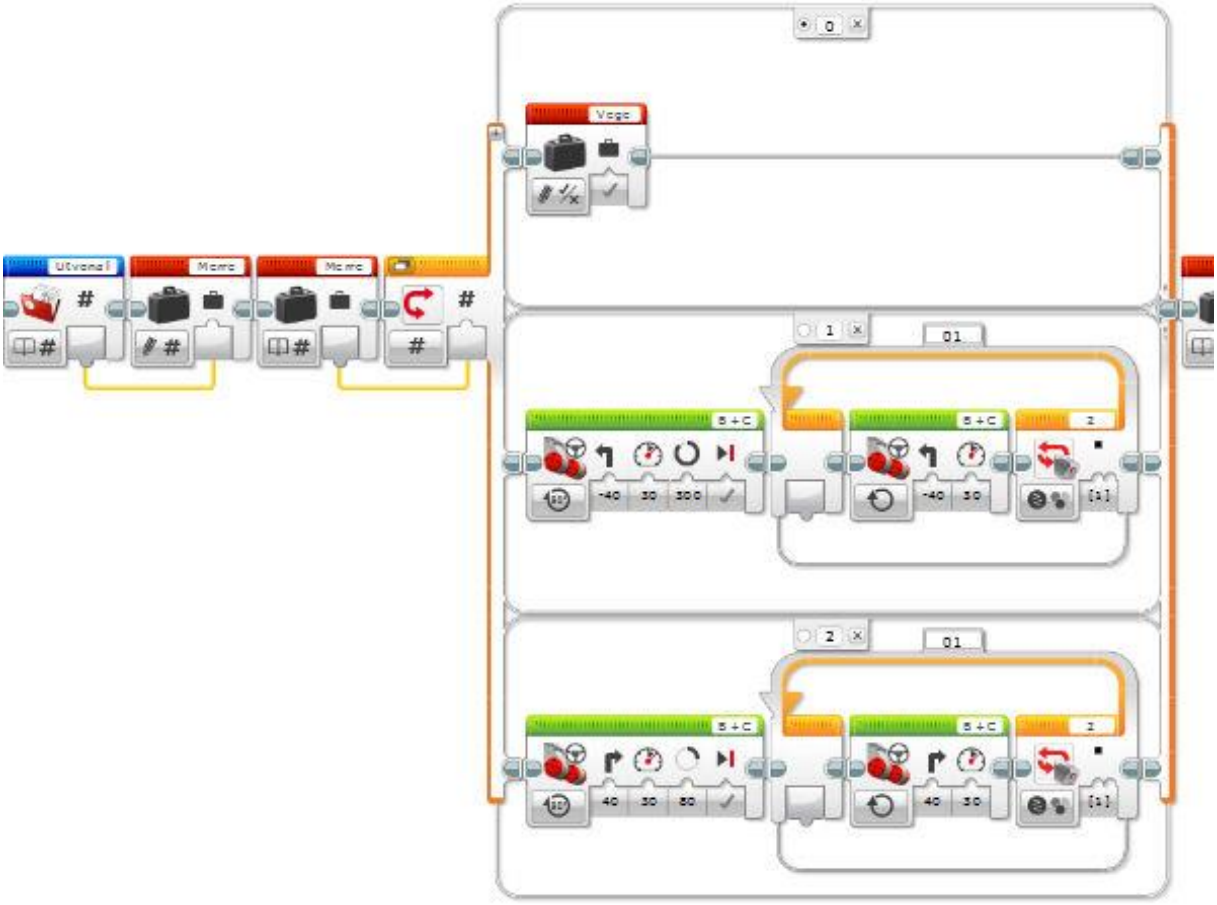


Az alaprobotra két fény/szín szenzor felhelyezése szükséges, egymással párhuzamosan, lefelé néző irányban. Az egyikkel fogja az útvonalat követni, míg a másikkal a kereszteződéseket számolni. A program elkészítésekor két fő lépésre kell bontani a feladatot. Az első a koordináta rendszerben történő megadott hosszúságú haladás, a második a robot fordulása a megadott irányba. Mindkét részt a fájlban tárolt számok alapján lehet vezérelni. Célszerű létrehozni két változót, amelybe a fájlból beolvassuk az adatpárok értékeit. A *Mennyit* változóba az előre haladás koordinátaszámát, a *Merre* változóba a fordulás irányát.

Egyszenzoros útvonalkövetéssel követjük a fekete vonalat (pl.: a robot 3-as szenzorával, míg a 2-es szenzor az útvonal mellett halad és a fekete kereszteződéseket számolja. Mikor elért egy kereszteződést az útvonalkövetés véget ér és a robot egyenesen halad előre egy kicsit, hogy a kerekek tengelye kb. a kereszteződés fölé kerüljön. A fordulásnál így fog tudni optimálisan kanyarodni. Mindezt a beolvasott koordinátaszámszor ismételjük.



Ha elérte a robot a megfelelő kereszteződést, akkor a fájlból beolvassuk a fordulás irányát. Itt három féle értéket kaphatunk. Ha 1-es akkor balra, ha 2-es akkor jobbra fordulunk. Először konstanssal egy kicsit, hogy a fényszenzor átforduljon a fekete vonal fölött, majd addig, amíg a 2-es szenzor el nem éri a fekete vonalat a fordulás során. Itt szükség lehet tapasztalati konstansokkal történő korrekcióra. Ha a beolvasott érték 0, akkor meg kell állni. Ezt egy logikai változó hamis értékről igazra történő megváltoztatásával fogjuk elérni. Az elágazásnak tehát három ága van, ezért Numerikus vezérlő feltételt állítunk be, ahol az egyes számadatok fogják eldönteni, hogy melyik szálon szereplő utasítások hajtódnak végre (0; 1; 2), a *Merre* változóba beolvasott érték alapján.

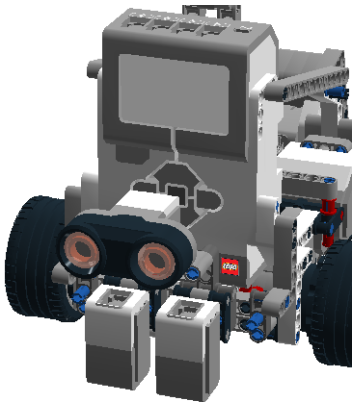


A két lépésnél bemutatott utasítássorozatot egy ciklusba szervezve a program már működőképes. A ciklus kilépési feltétele a logikai változó igaz értéke, amelyet az elágazás 0 értékű szálán állítottunk be.

F.3.2.2. Koordináta rendszer 2.

A feladat megoldása az előző feladat ötleteit felhasználva nem nehéz. A célkoordináták meghatározásához külön programszálon érdemes leolvasni a bal illetve jobb oldali fény/szín szenzor alatti vonalak számát. A célkoordináták ismeretében az *x* értéknek megfelelő koordináta vonalon áthaladva jobbra fordulás után *y* vonal felett kell még áthaladni.

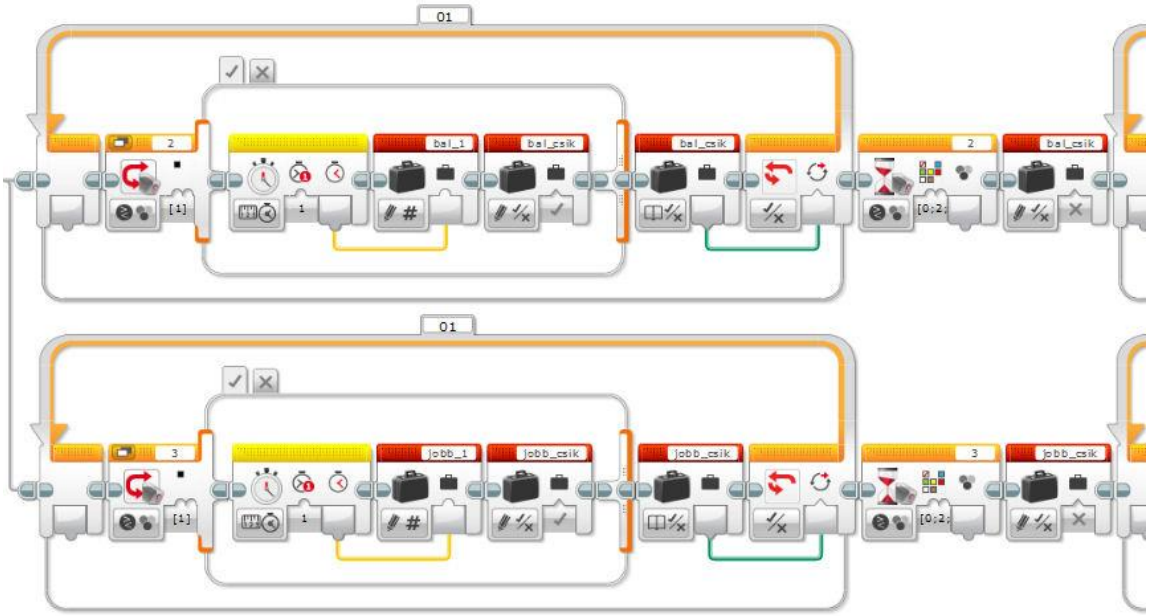
F.3.2.3. Térképezés (egyenes szakaszok)



Az alaprobotra két fény/szín szenzor felhelyezése szükséges, egymással párhuzamosan, lefelé néző irányban. A két fény/szín szenzor közötti távolság biztosítja, hogy a pályán keresztben futó vonalak dőlésszögét érzékelni tudja. Az indulástól számítva (egyenes haladás esetén) különböző időtartam alatt éri el a két fényszenzor a haladási irányra nem merőleges szakaszokat.

a) A feladatot két részre lehet bontani. Az első részben a fényszenzorokkal érzékeljük a vonal elérését és egy stopperrel

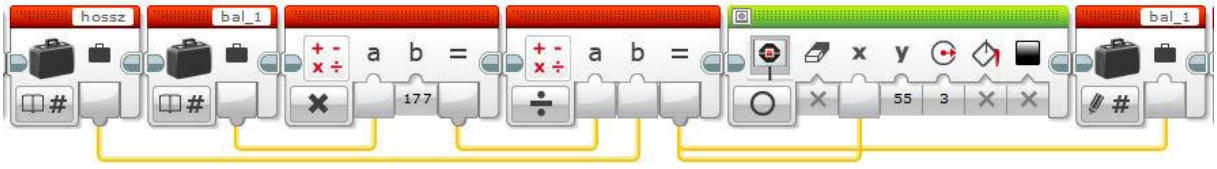
mérjük, hogy ez az indulás után mennyi idővel következett be. Ezeket az időtartamokat tárolni kell, hiszen a rajzoláshoz szükséges a pálya hossza, amit csak a megállás után fogunk ismerni. EV3 robot esetén használhatunk tömböt a tároláshoz, míg NXT robot esetén a két fényszenzor három-három méréséhez összesen 6 db változó szükséges. A két szenzor méréseit két külön programszálon futtatjuk. Egy-egy változó értékének tárolását mutatja az alábbi programrészlet.



Az adattárolást akkor kell elvégezni, amikor a robot szenzora fekete színt érzékel. Ekkor ki kell lépni a szenzort figyelő ciklusból, ezt egy logikai változó értékének megváltoztatásával érjük el (*bal_csik* illetve *jobb_csik*). Ilyenkor a robot szenzora még fekete terület fölött van, ezért a következő mérést még nem lehet elkezdeni. Meg kell várni, amíg a szenzor leért a fekete területről és az újabb mérés előtt a logikai változó értékét is vissza kell állítani hamisra.

A program második részében történik meg a mérési adatok feldolgozása (a robot megállása után).

A pálya hosszának ismeretében egyszerű egyenes arányossággal át tudjuk számítani a mért időtartamokat koordinátákra. A képernyő szélessége EV3 robot esetén 178 pixel.



Ezt a számolást mind a 6 változó esetén meg kell tenni. Érdemes az utasításokból egy saját blokkot létrehozni, ez rövidít a programkódon. Tömbök használata esetén ciklussal is végezhető a számolás.

b) A feladat megoldásához a koordináta geometriában megismert összefüggések alapján egy egyenes két pontjának ismeretében felírható az egyenlete, és ebből számítható a képernyő széleire eső képpontok koordinátái. Ezeket szakaszokkal összekötve megrajzolhatók az „egyenesek”.

F.3.2.4. Térképezés (körök)

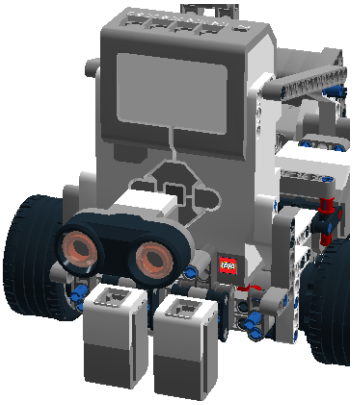
A feladat megoldása az előző feladattal analóg módon végezhető el.

A robotnak most csak egyetlen fényszenzora van. Az útvonala a körök középpontjai fölött halad. Két mérést kell végeznie minden kör esetén: a bal oldali kerületi pont kezdetét és a jobb oldali kerületi pont kezdetét kell megmérni. A méréshez ismét lehet stoppert használni. Az induláskor lenullázva a fekete vonalakig eltelt idő arányos lesz a koordináta-rendszerbeli pozíció *x* koordinátájával.

A mozgás befejeztével át kell számolni a mért időtartamokat koordinátákra. A teljes megtett út időtartama felel meg a képernyő szélességének, így egyenes arányossággal megkaphatók az *x* koordináták. Egy kör esetén a két mérési pont számtani közepe lesz a középpont *x* koordinátája, a különbség fele pedig a sugár.

A mérési pontok koordinátáit tárolhatjuk tömbben, így ciklusok használatával egyszerűsödik a program. Saját blokkokat is készíthetünk az ismétlődő utasítássorozatokból.

F.3.2.5. Nem folytonos útvonalkövetés akadályokkal

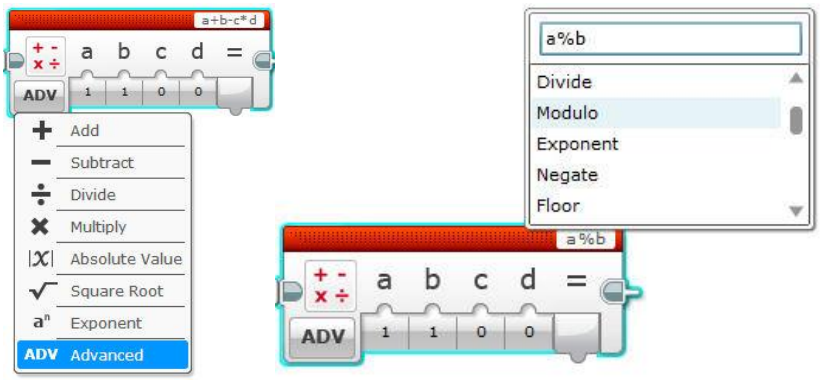


Az alaprobotra két fény/szín szenzor felhelyezése szükséges, egymással párhuzamosan, lefelé néző irányban. Az fekete színű útvonal a két szenzor között lesz.

A feladat megoldásához elegendő a két színszenzorral történő útvonalkövetés algoritmusa, amely az ultrahang szenzorral mért akadálytól 10 cm-re ér véget itt a megfelelő irányú fordulás után egyenes előrehaladás következik. Ezt az újabb fekete színű útvonal elérése szakítja meg, amelyet például fény/szín szenzorral lehet figyelni. Ezek a lépések ismétlődnek. Az akadály elérésekor a fordulási irányok különbözők, így ciklussal csak akkor oldható meg a feladat, ha a fordulási irányt például változóval szabályozzuk. Az irányok csak előjelben térnek el.

F.3.2.6. Radar

A feladat megoldásában a matematikai háttér ismerete okozhat problémát. Az *Data Operation* ikoncsoport *Math* blokkjában az *Advanced* beállításnál van lehetőségünk matematika függvények használatára.

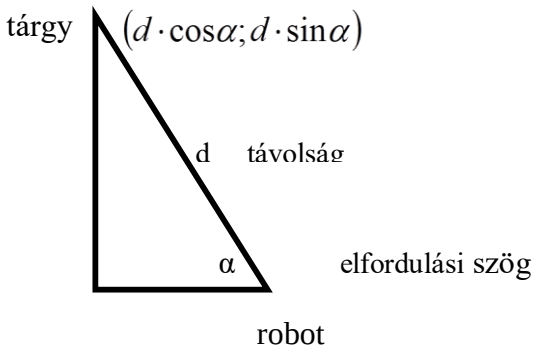


Az *Advanced* funkciót kiválasztva a blokk jobb felső sarkára kattintva a legördülő listából választhatunk a matematikai függvények közül.

A mért elfordulási szög és távolság adatokból (polár koordináták) ki tudjuk számítani a koordináta rendszer Descartes koordinátáit.

Az összefüggést az ábra szemlélteti.

A trigonometrikus számítás eredménye alapján megkapjuk a rajzolandó kör középpontjának koordinátáit.

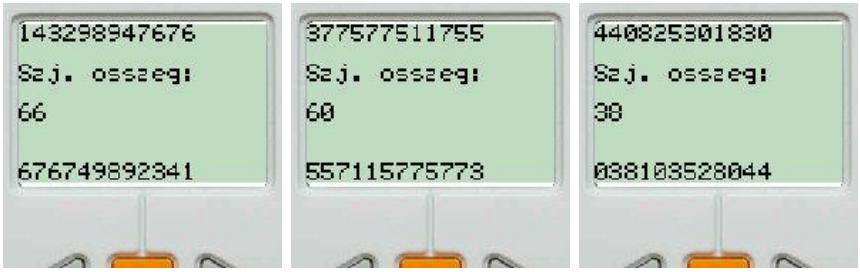


3.3. MATEMATIKA, VÉLETLEN SZÁMOK

F.3.3.1.a-c Feladat – Nagy számok

a) Szint: 

Írj programot, amelyet végrehajtva a robot a képernyőjére ír egy 12 jegyűnek „látszó” véletlen számot (a képernyőre írt véletlen szám kezdődhet 0-val is)! A képernyőre írja a számjegyek összegét, és a 12 jegyű számot visszafelé is. A kiírás formátumát az ábra értelmezi. A program az ütközésérzékelő benyomására álljon le!



b) Szint: 

Írj programot, amelyet végrehajtva a robot a képernyőjére ír két 12 jegyűnek „látszó” véletlen számot (a képernyőre írt véletlen szám kezdődhet 0-val is)! Majd egy vízszintes vonal alá kiírja a két szám összegét is.

c) Szint: 

Írj programot, amelyet végrehajtva a robot fájlba ír két 200 jegyű véletlen számot, majd a fájlba írja az összegüket is! A három szám külön sorban szerepeljen egymás alatt. Első két sorban a két véletlen szám, majd a harmadik sorban az összegük.

F.3.3.2.a-b Feladat – Kerekítés

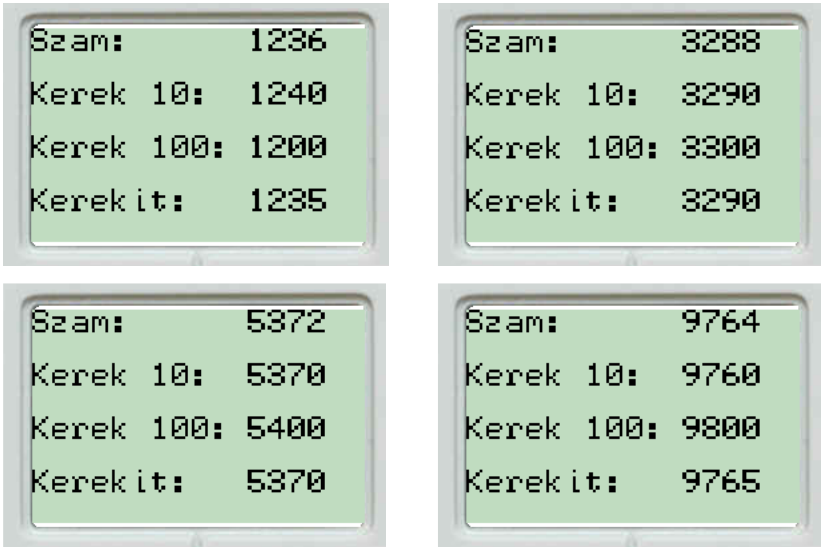
a) Szint: 

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

Például:

b) Szint: A robot írja a képernyőjére a korábban sorsolt szám jelenleg érvényes forint kerekítési szabály szerint kerekített értékét. Tehát 1-re vagy 2-re végződő számok esetén 0-ra kerekítünk lefelé, 3-ra vagy 4-re végződő számok esetén 5-re kerekítünk felfelé, 6-ra vagy 7-re végződő számok esetén 5-re kerekítünk lefelé, míg 8-ra vagy 9-re végződő számok esetén 0-ra kerekítünk felfelé.

Például:



F.3.3.3.a-c Feladat – Célba lövés

a) Szint:

Írj programot, amelyet a robot végrehajtva képernyőjére rajzol egy

- NXT robot esetén: 40 pixel oldalú négyzetet (céltábla), amelynél a bal alsó csúcs koordinátái (20;10)!
- EV3 robot esetén: 80 pixel oldalú négyzetet (céltábla), amelynél a bal felső csúcs koordinátái (20;10)!

A robot sorsoljon véletlenszerűen két számot, amely koordinátákhoz tartozó pont körül a képernyőre rajzol egy 2 pixel sugarú kört (lövés találati pontja). A sorsolást úgy végezze, hogy a rajzolandó körvonalnak legalább a negyed része látsszon a képernyőn. A sorsolások alatt a céltábla végig látszódjon a képernyőn! Ha a sorsolt pont a négyzeten belül van, akkor sípoljon egyet a robot, egyébként ne. Egy másodperc várakozás után ismételje a sorsolást, mindaddig, amíg az ütközésérzékelőt nyomás nem éri. Valamennyi pont maradjon a képernyőn! Az ütközésérzékelő megnyomása után a program álljon le.

b) Szint: 

Módosítsd a programot úgy, hogy az ütközésérzékelő megnyomása után írjon a képernyőre egy statisztikát a következő formában:

Például:

```
Lovesek szama: 40
Talalatok szama: 3
Talalati arany: 7,5 %
```

c) Szint: 

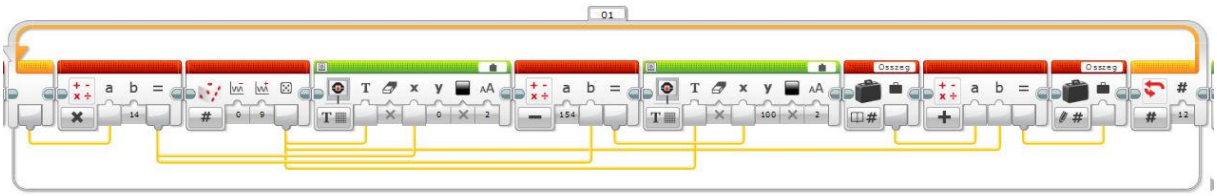
A feladatban a céltábla alakja legyen kör (és ne négyzet)! A képernyő jobb felső sarkába a találatok számát is írjuk ki folyamatosan! A kör középpontja képernyő közepe legyen, sugara 40 pixel.

Programozási ötletek

F.3.3.1. Nagy számok

A feladatban a lényeges elem, hogy olyan nagy számokkal hogyan tudunk számolni, amelyek már kívül esnek a változó tárolási méretén. Ha egy szám típusú változót a rendszer 4 byte-on tárol, akkor a legnagyobb szám, amit még kezelni képes a 2^{31}-1 = 2147483647 (ez 10 jegyű). A 12 jegyű számokat így számjegyenként, vagy szövegként tudja csak értelmezni. A feladatban a „látszólag” 12 jegyű számot számjegyenként sorsoltatjuk és íratjuk a képernyőre. Azért csak 12 jegyű számokat használ a feladat, mert a képernyőre nem fér ki több számjegy, egyébként tetszőleges hosszúságú „számok” is előállíthatók a bemutatott módszerekkel.

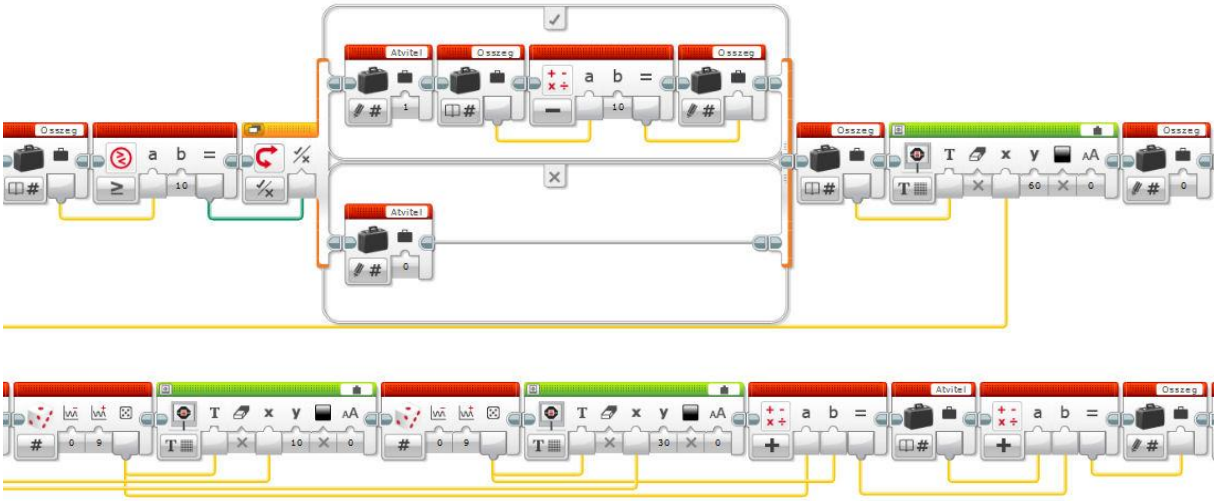
a) Egy 12-szer lefutó ciklusban számjegyeket sorsoltatunk, amelyeket a képernyőre íratunk, egymástól 14 pixel távolságra. A képernyőre írás pozíciójának *x* koordinátáját a ciklusváltozó 14-szerese adja. Ugyanebben a ciklusban a számot már visszafelé is kiíratjuk a (154;100) pozíciótól kezdve 14-gyel csökkentve az *x* értékét számjegyenként. A ciklusban elvégezzük a jegyek összeadását is, az *Osszeg* nevű változóban.



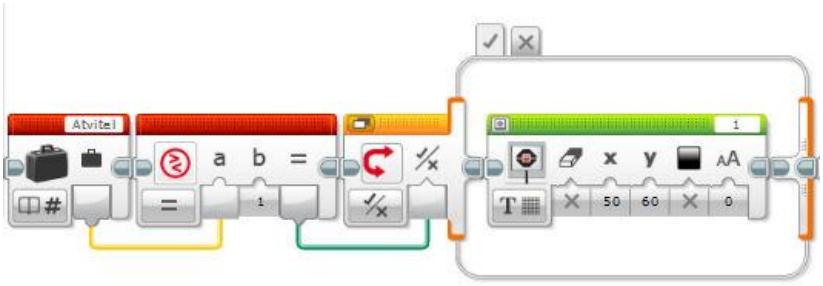
A ciklus lefutása után már csak a számjegyek összegének képernyőre írása maradt.

b) A feladat a) részéhez képest lényeges nehezítés a két 12 jegyű szám összeadása. A számokat a legkisebb helyiértéktől kezdve sorsoltatjuk, így az összeadás ugyanabban a ciklusban, tárolás nélkül elvégezhető. A kiíratás az a) feladatrészben leírtnak megfelelően a 170-es *x* koordinátától kezdődően visszafelé történik. Minden számjegyre 10 pixel vízszintes helyet hagyunk. Az összegzésnél az írásbeli összeadás matematikai szabályát használjuk. Ha két azonos helyiértéken álló számjegyet összeadunk és az összeg 10 vagy annál nagyobb, akkor keletkezik egy átvitel (1), amit az eggyel

nagyobb helyiértéken álló számjegyek összegéhez kell adni. A programban ezt az átvitelt az *Atvitel* változó tárolja. Két számjegy összeadásánál, ha van átvitel, akkor a két számjegy összegénél 10-zel kisebb számot írunk ki összegként a képernyőre, ha nincs átvitel, akkor a két számjegy összege lesz a kiírt szám, persze az előző átvitel összegével növelve.

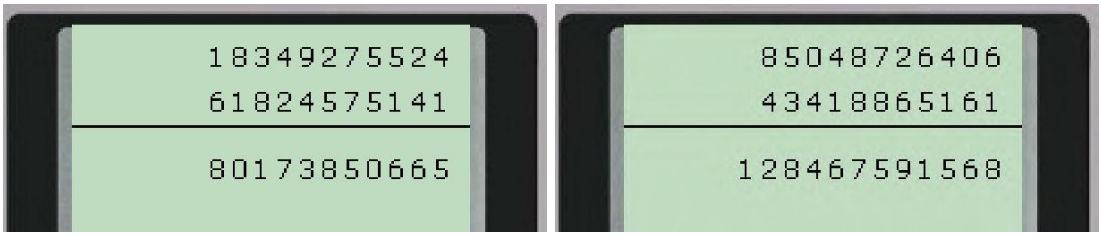


A ciklus lefutása után, ha maradt átvitel, akkor 13. számjegyként az 1-est még a szám elé kell írni.



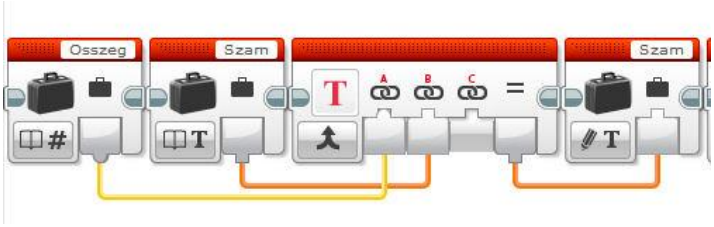
A program tetszőleges hosszúságú számra is működik, de az eredményt a képernyő korlátozott mérete miatt nem tudjuk kiíratni. Fájlba viszont igen.

A program által generált képernyőkép:

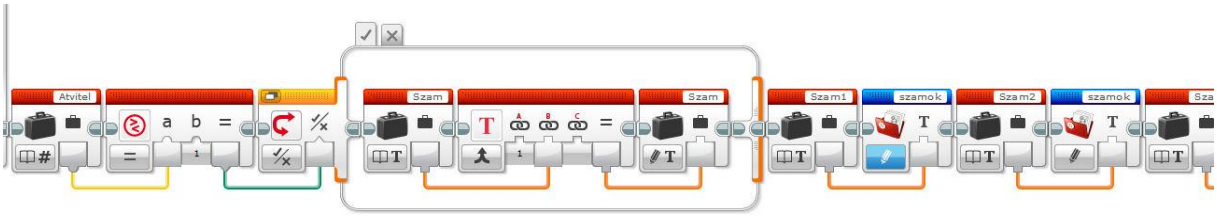


c) Az előző program algoritmusa alkalmas a feladat megoldására. A képernyőre írás helyett azonban fájlba írjuk a számokat. Mivel a számjegyek sorsolása a legkisebb helyiértékűvel kezdődik (írásiránnyal ellentétesen), ezért a *Data Operations/Text* blokkjával a cikluson belül összefűzzük a számokat egy-egy szöveges változóba, majd a ciklus lefutása után írjuk a fájlba.

Például az összeg esetén:



A ciklus lefutása után az átvitelt is hozzáfűzzük az *Osszeg* változóhoz, majd a két számot (*Szam1*; *Szam2*) valamint az összegüket (*Szam*) fájlba íratjuk. Minden írási művelet új sorba kerül a fájlban.



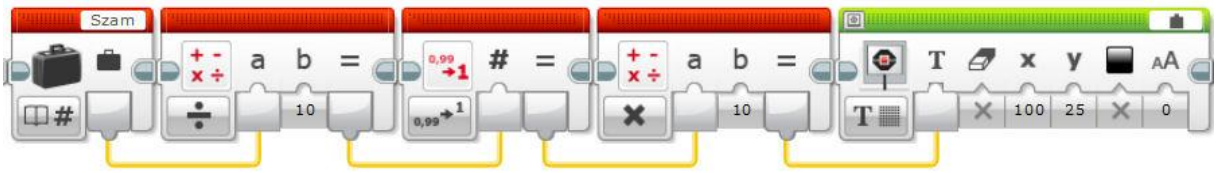
Ha a ciklust 200-szor futtatjuk, akkor a fájlba 200 jegyű számok kerülnek.

A program futásának eredménye a fájlban 200 jegyű számok esetén:

```
166110458201339985062757991714554060124528533705267582343888043246297553543919298527846526815823566119724936251335833361754028939806749333132752717435811261835453828362160375382977738689244704276126054461656658743855756575953236822448677686627510858716749853635283364916819538771883043448858221869967644884625376715331322723578388540585421865153617314547780698768281596865171255014608581487831865776561224075725560720353315396798927893191284791139257329251571582789235497796486832191412638010562884213398789007366494026386778660807875319268079167266039905330656521846892508479199547393487462703704
```

F.3.3.2. Kerekítés

a) Az EV3-as szoftverben rendelkezésre áll a kerekítés művelete *Data Operations/Round*. A működési módok közül a *To Nearest* funkciót választva a matematika szabályainak megfelelően kerekít egészre a program. Ha 10-esekre szeretnénk kerekíteni, akkor a trükk az, hogy osztjuk a számot 10-zel, kerekítjük egészekre, majd az eredményt ismét megszorozzuk 10-zel.



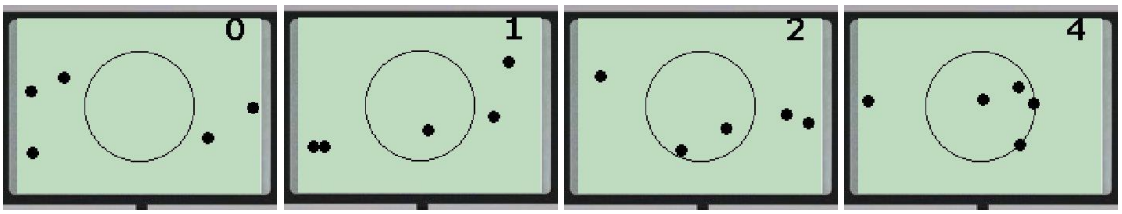
A százasra kerekítésnél a 10 helyett 100-zal osztunk, kerekítünk, majd 100-zal szorzunk.

b) A forint kerekítési szabályok az a) feladat mintájára kezelhetők, ahol matematikailag 5-ös kerekítési szabály van. Tehát a sorsolt számot 5-tel kell osztani, egészre kerekíteni, majd szorozni 5-tel.

F.3.3.3. Célba lövés

a-b) A feladat megoldását a nehezebb c) eset alapján könnyű kódolni.

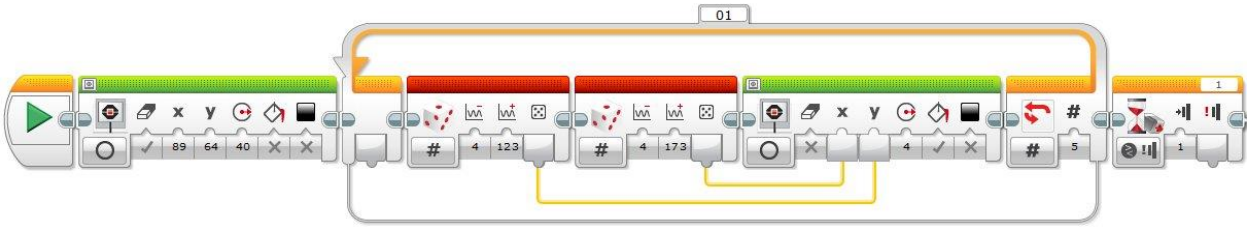
c) A képernyőkép különböző találatszámok esetén:



A program megírását kezdjük azzal, hogy a céltáblát felrajzoljuk a képernyőre. Ennek középpontja legyen a (89; 64) koordinátájú pont, sugara 40 pixel. Ezután sorsoljunk két véletlen számot! Mivel a találati pontokat 4 pixel sugarú körök jelölik, ezért ha azt szeretnénk, hogy a találat teljes egészében látszódjon a képernyőn a középpont vízszintes koordinátáit a 4-173; a függőleges koordinátáit a 4-123 tartományból kell sorsolni. A megjelenítést egy *Display* blokkal végezzük, amelyet alakzat rajzoló módban, *Circle* (kör) funkcióval használunk. A kör sugara 4 pixel, a képernyőtörlést kikapcsoltuk és beállítottuk az alakzat fekete színnel történő kitöltését.

Mivel öt lövést kell megjelenítenünk, ezért a sorsolást és megjelenítést egy ötször lefutó ciklusba tettük. A program az ütközésérzékelő megnyomására áll le.

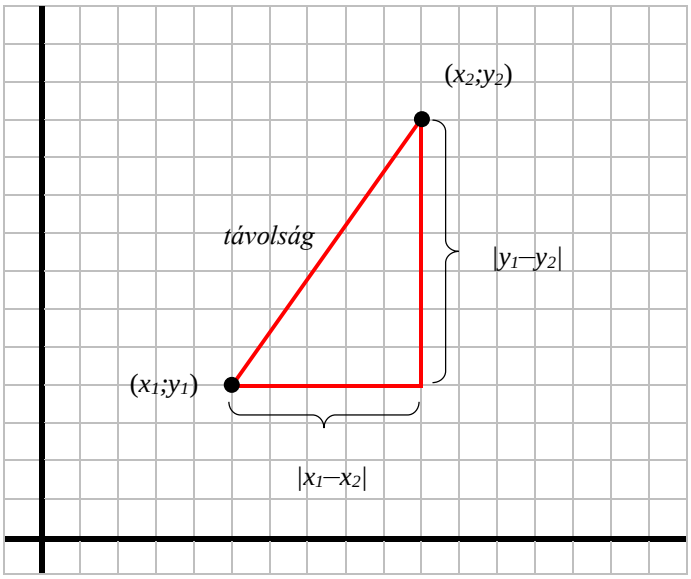
Mielőtt folytatnánk a program írását teszteljük le a jelenlegi változatot. A megjelenítés már működik.



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

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

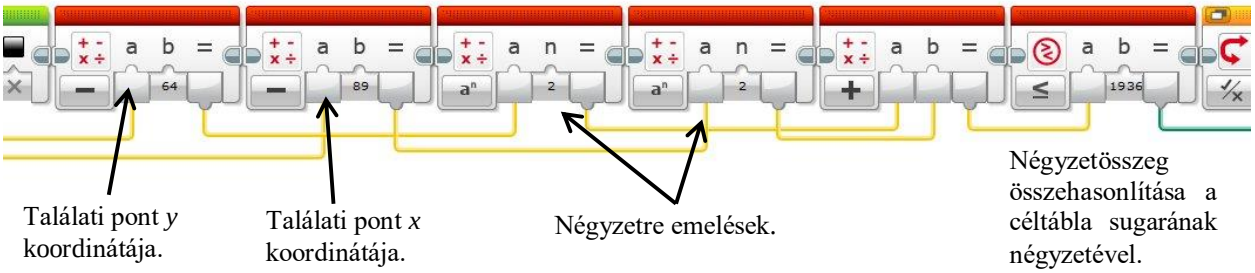
ahol az egyik pont koordinátái (x_1;y_1), a másik ponté (x_2;y_2). A kivonás sorrendje a négyzetre emelés miatt mindegy (csak előjelben különbözne az x_1-x_2 az x_2-x_1-től).



A találati pont tehát akkor van a céltábla körén belül, ha kör középpontjától mért távolsága nem nagyobb a kör sugaránál. Ugyanezt úgy is megfogalmazhatjuk, hogy a kör középpontjának és a találati pont megfelelő koordinátái különbségének négyzetösszege nem nagyobb, mint a sugár négyzete.

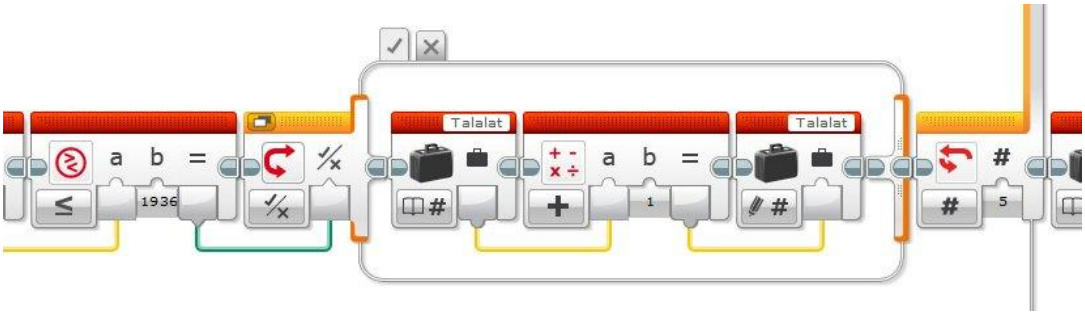
Mivel a találati pontot 4 pixel sugarú kör jelzi, ezért ha azt is találatként szeretnénk számolni, amikor a lövés széle érinti a kör szélét, akkor a céltábla kör sugarát 4 pixellel nagyobbnak kell választani. Így a 44 pixeles kör sugarának négyzete: 1936.

A matematikai műveleteket kiszámító kódrészlet tehát:



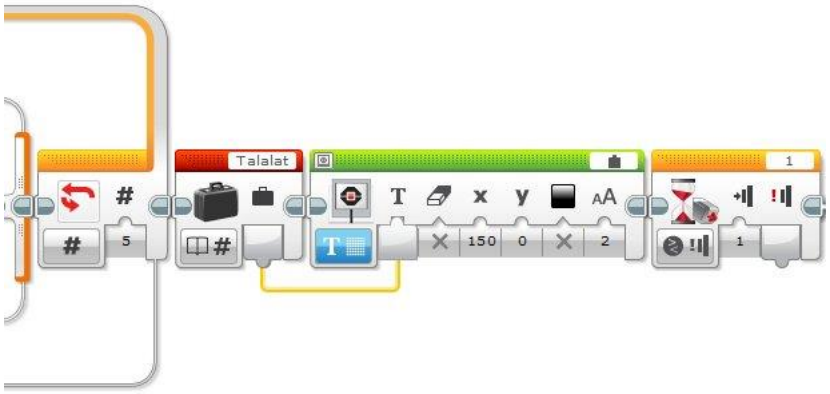
Az összehasonlítás eredménye egy logikai érték. Hamis, ha a pont kívül van a céltábla körén, egyébként igaz.

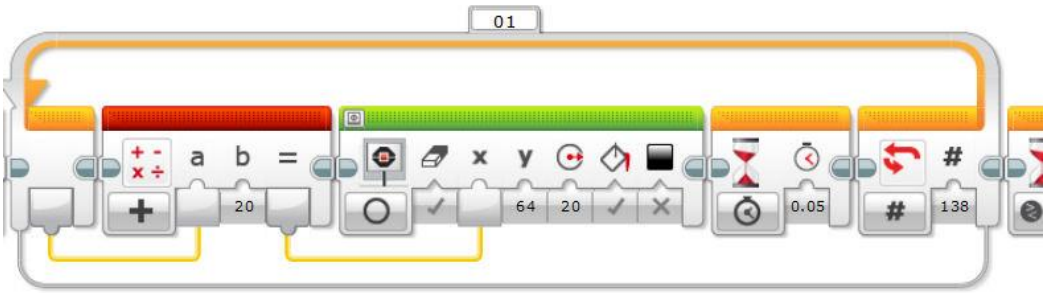
A következő lépés a találatok számlálása. A programozásban használt megszámlálás algoritmusának ötletét használjuk. Egy nulla kezdőértékű változó, értékét növeljük eggyel, ha egy feltétel teljesül, egyébként nem csinálunk semmit. A feltétel most az előző matematikai számolás eredményeként kapott logikai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



A megnövelt értéket el kell tárolnunk újra az eredeti (*Talalat*) változóban, hogy a következő növeléskor már az új értékkel tudjunk számolni.

A ciklus ötszöri lefutása után a *Talalat* változóban a találatok számának megfelelő érték lesz, amelyet a képernyő megfelelő pozíciójára írathatunk.



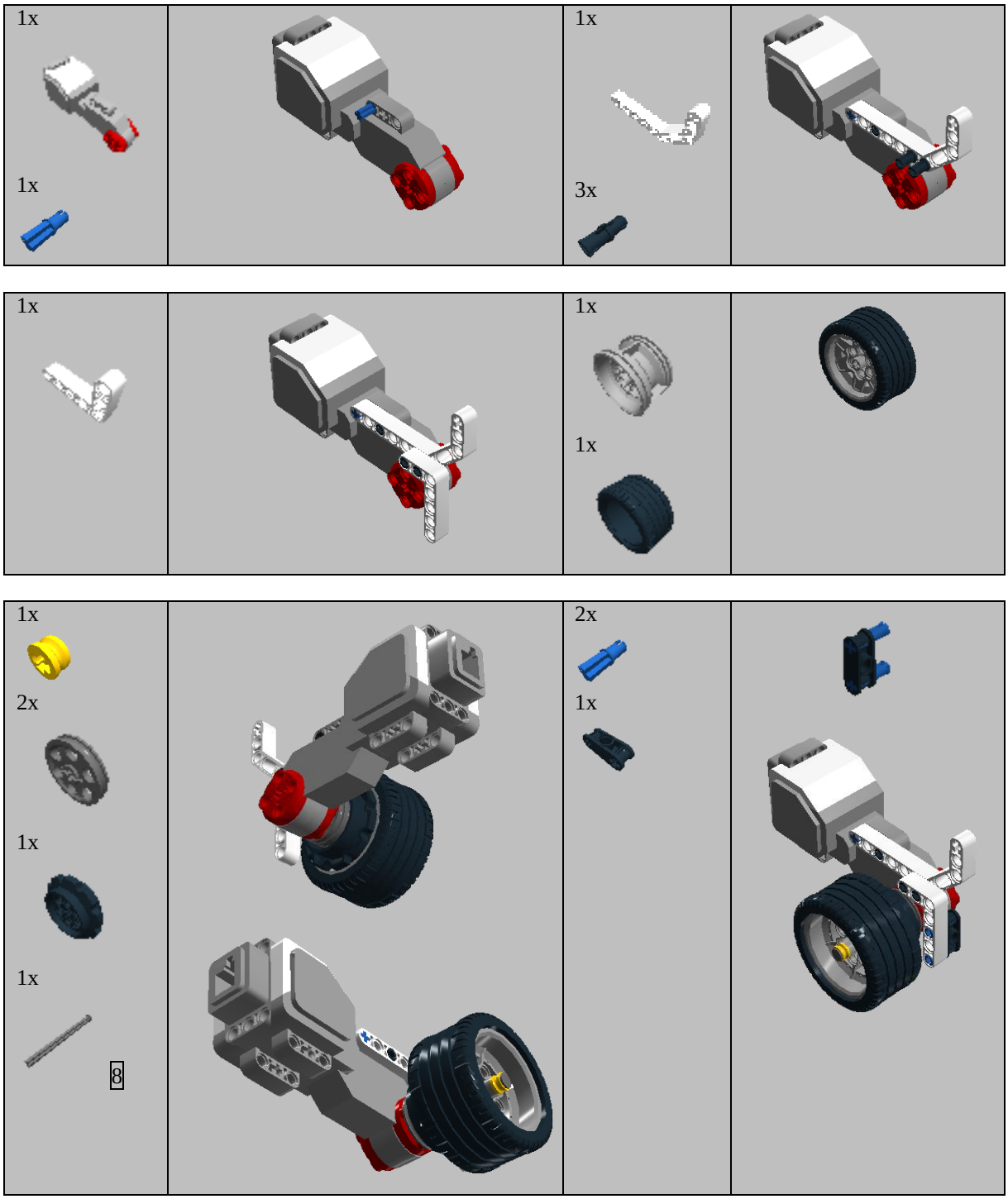


MELLÉKLETEK

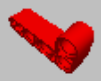
1. MELLÉKLET: ÉPÍTÉSI ÚTMUTATÓ – EV3 ALAPROBOT

A portok javasolt kiosztása:

Motorok: B és C port
Ütközésérzékelő: 1-es port
Gyroszenzor: 2-es port
Színszenzor: 3-as port
Ultrahangszenzor: 4-es port



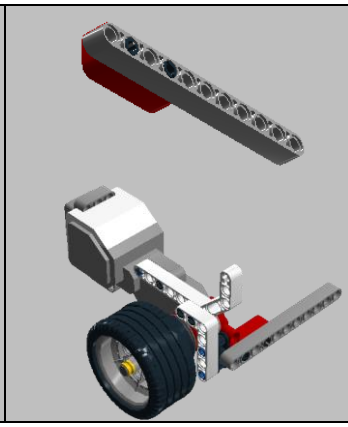
1x



1x

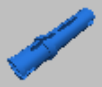
11

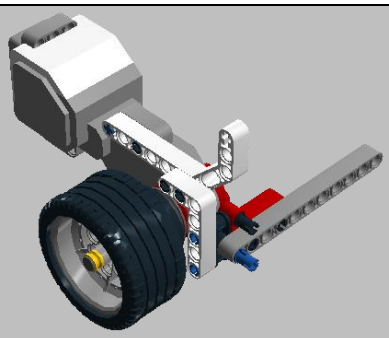
2x



1x

1x



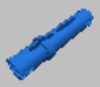


1x

11

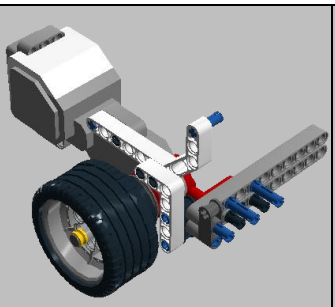
2x

3x

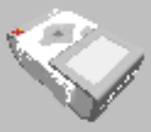


1x

1x

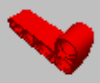


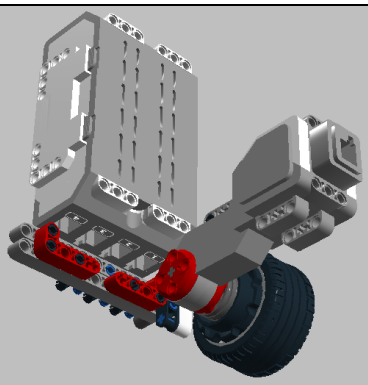
1x



3x

1x

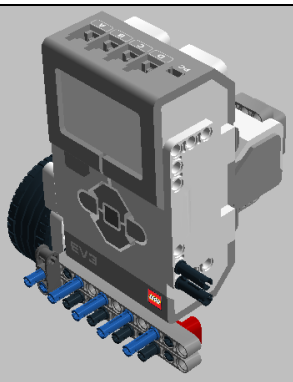




1x

2x

2x

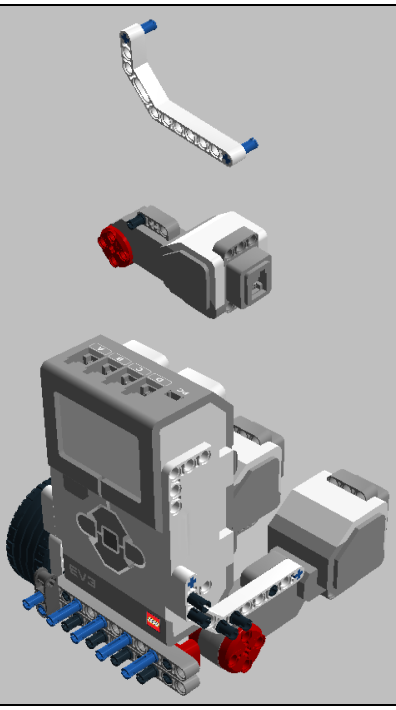


1x

3x

1x

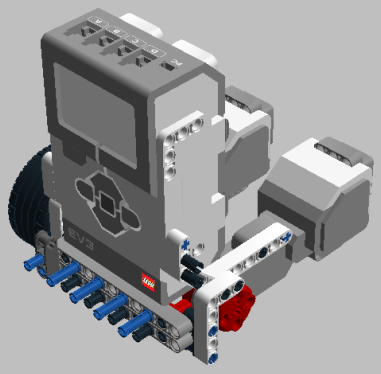
2x



1x

2x

1x



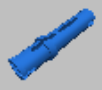
1x

1x

2x

1x

2x



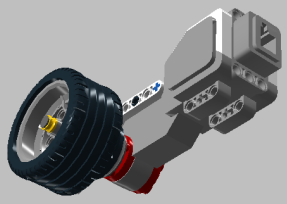
1x

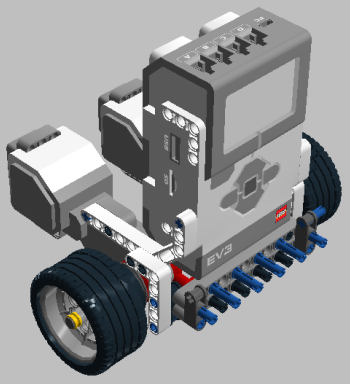
1x

8

1x

1x





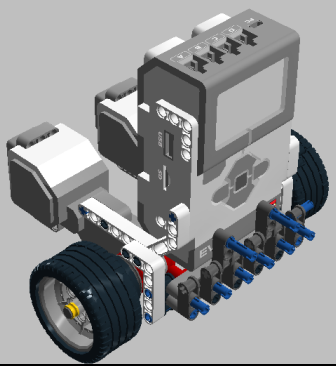
2x

3

6x

2x

2x

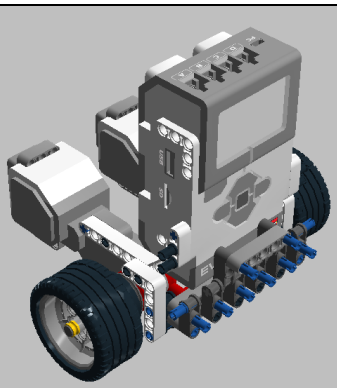


1x

5

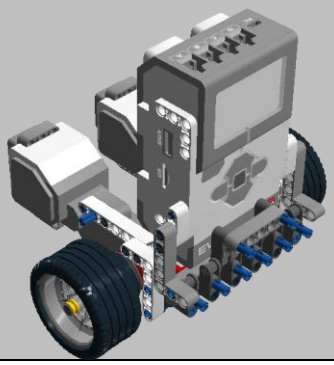
2x

4x



2x

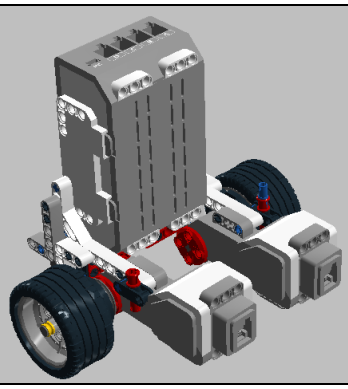
2x



2x

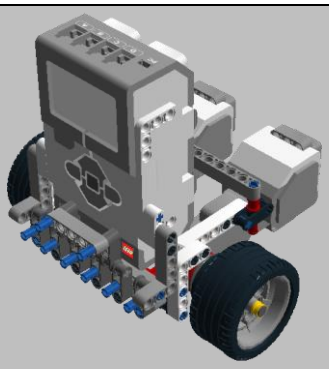
2x

2x



1x

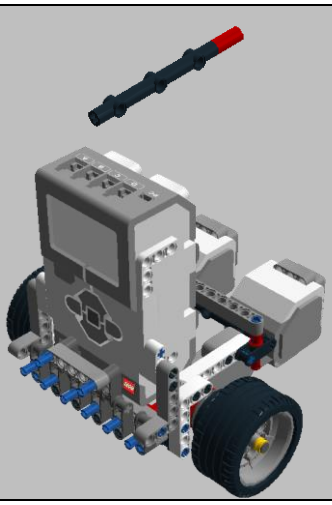
13



3x

3x

1x



2x

2x

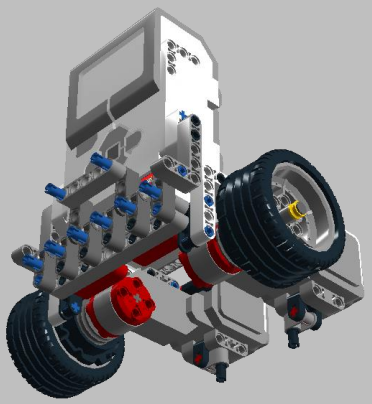
2x

2x

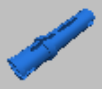
2x

3

2x

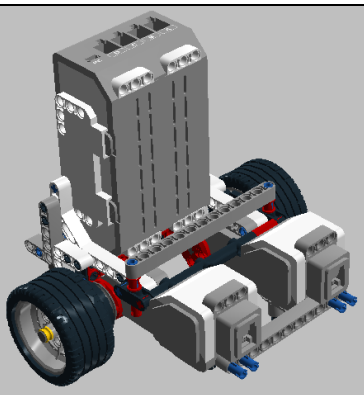


4x



1x

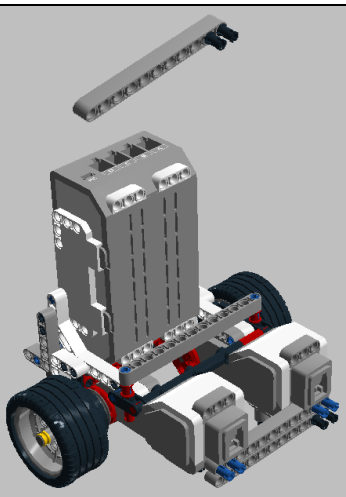
11



1x

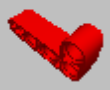
11

2x



1x

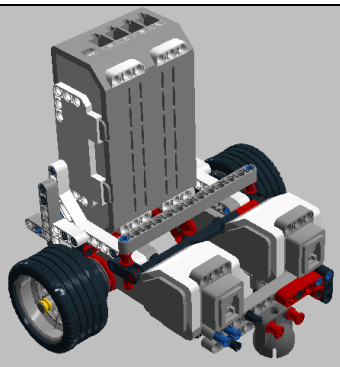
1x



3x

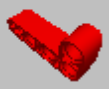
1x

2x



1x

1x

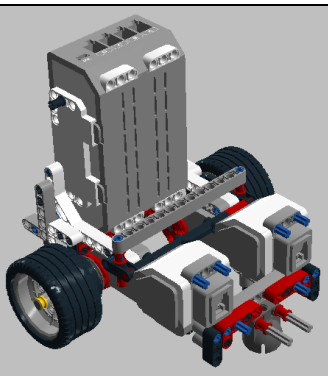


2x

6x

2x

3

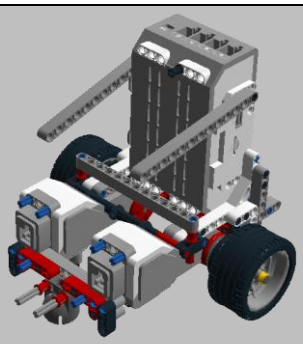


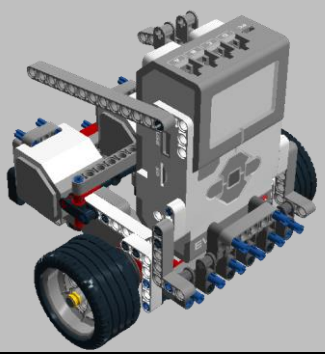
2x

2x

13

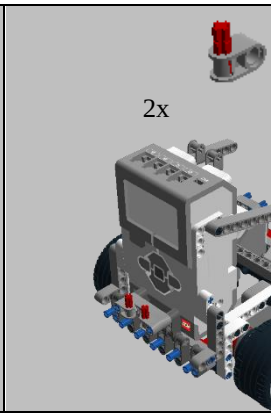
2x





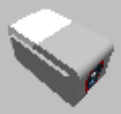
2x

2x



2x

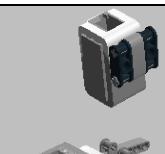
1x

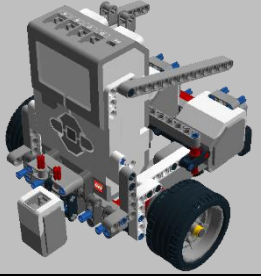


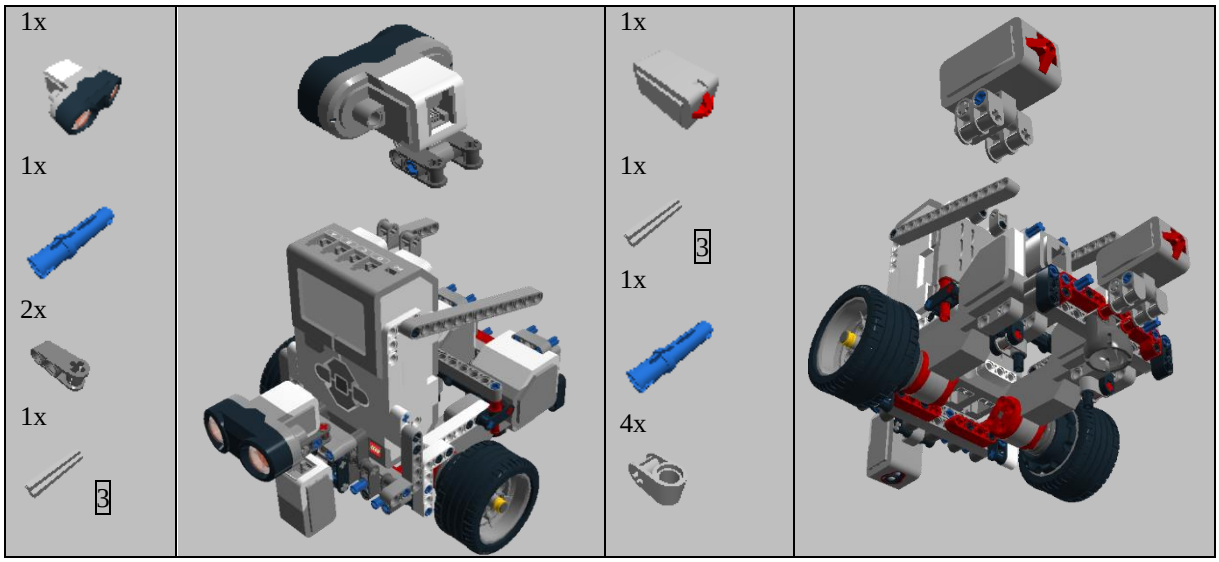
2x

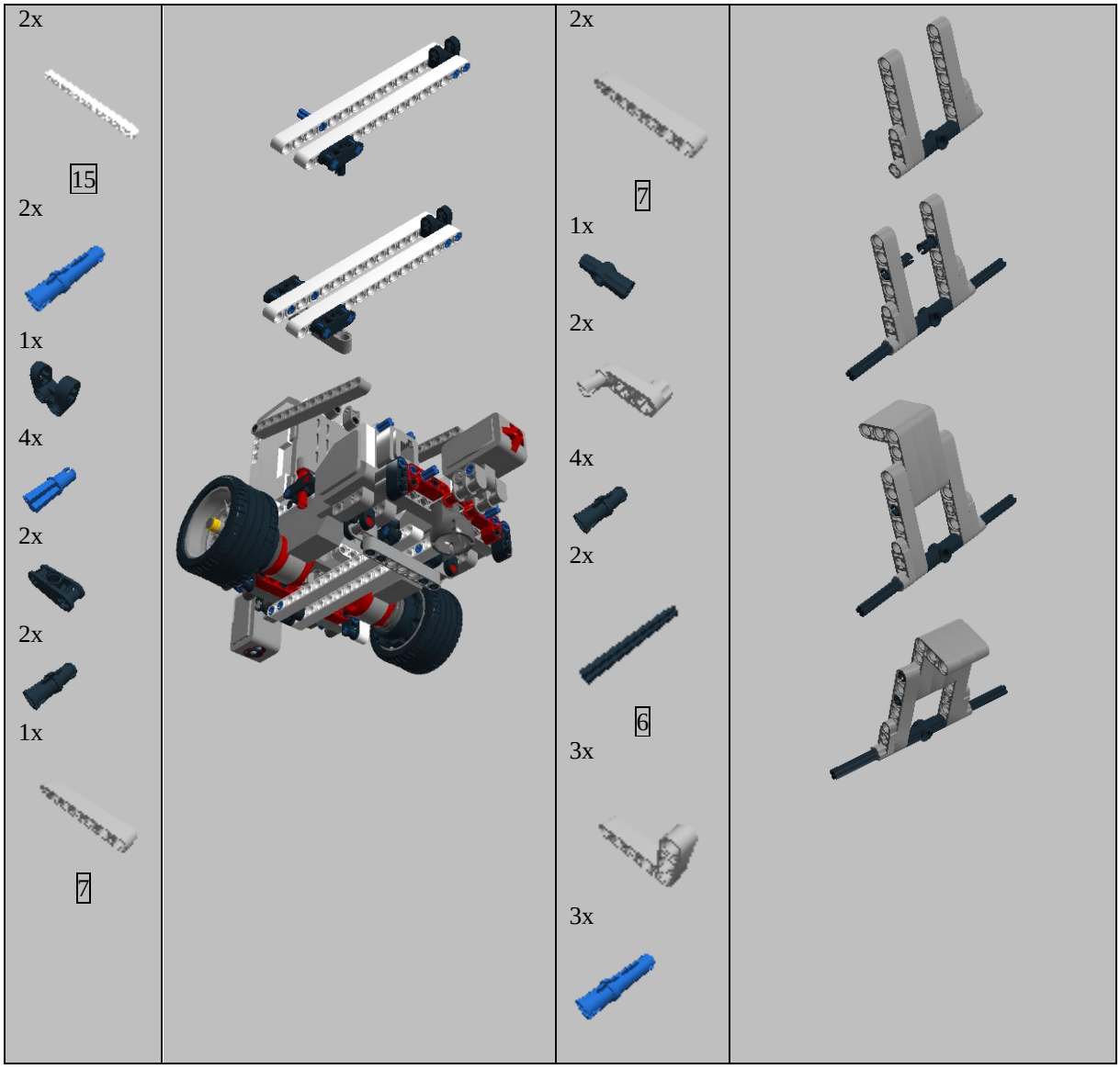
3

2x






1x
1x
2x
1x
3
1x
1x
3
1x
4x


2x
15
2x
1x
4x
2x
2x
1x
7
2x
7
1x
2x
4x
2x
6
3x
3x

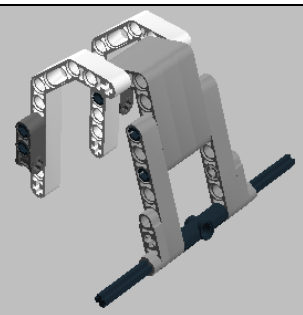
2x

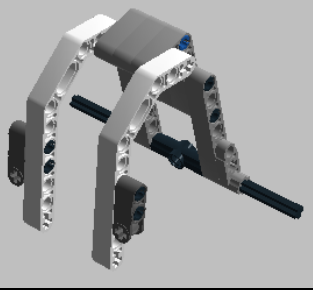
4x

2x

1x

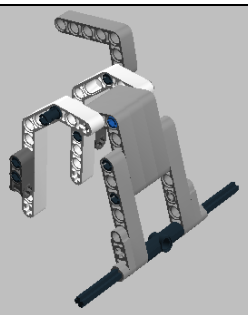
5

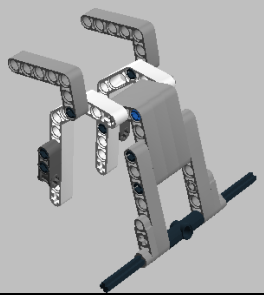




2x

2x

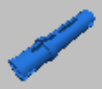




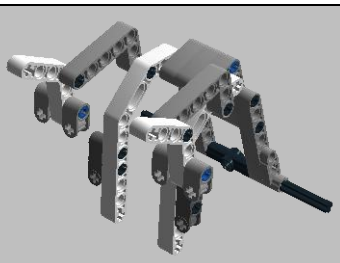
2x

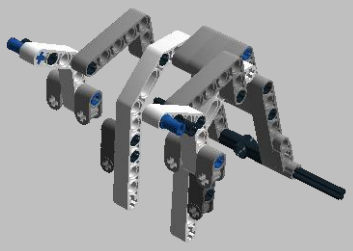
4x

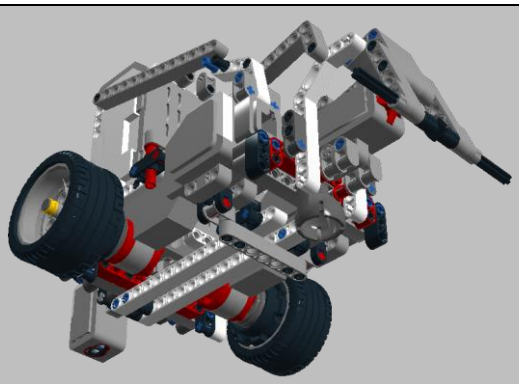
4x

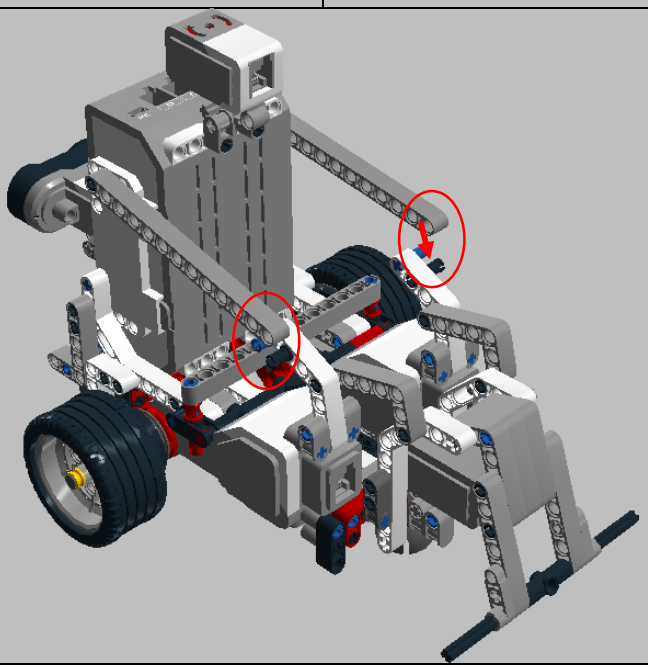


4x







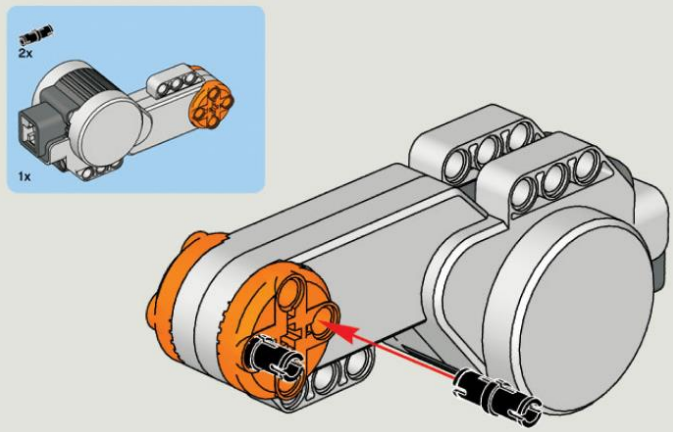


2. MELLÉKLET: ÉPÍTÉSI ÚTMUTATÓ – NXT ALAPROBOT

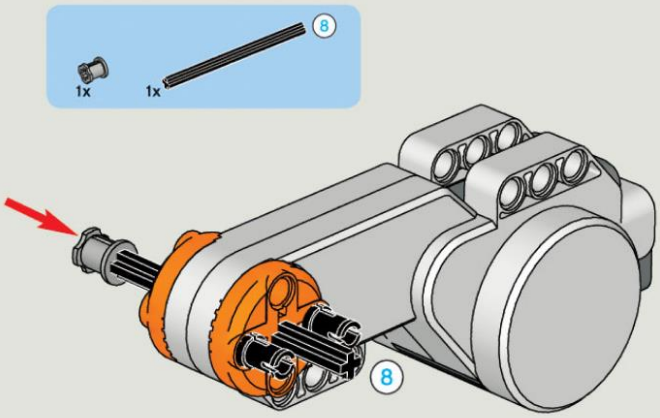
A portok javasolt kiosztása:

Motorok: B és C port
Ütközésérzékelő: 1-es port
Fény- vagy színszenzor: 3-as port
Ultrahangszenzor: 4-es port

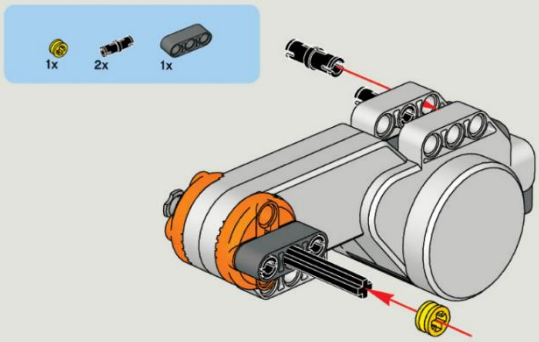
1.



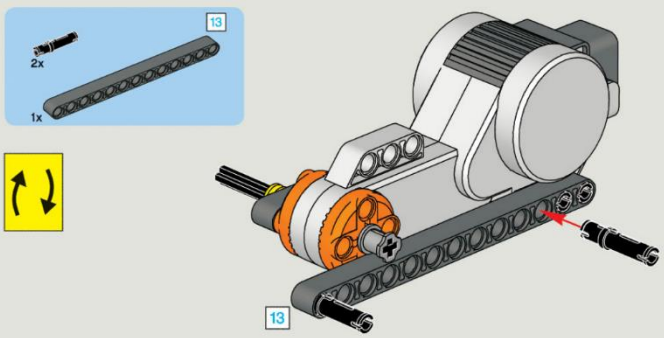
2.



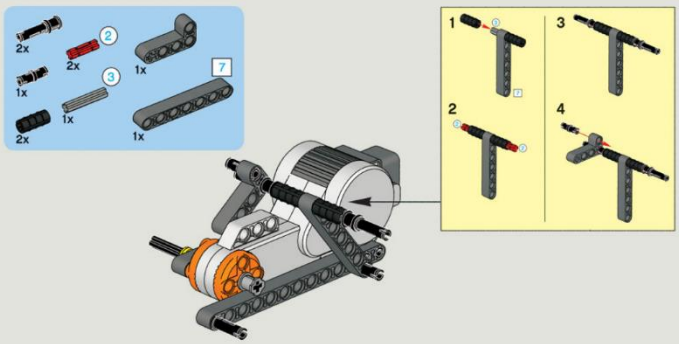
3.



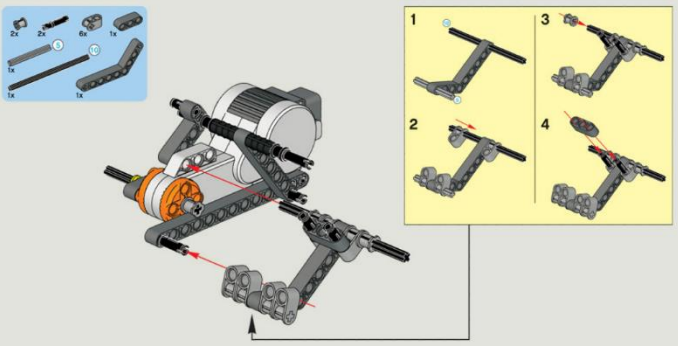
4.



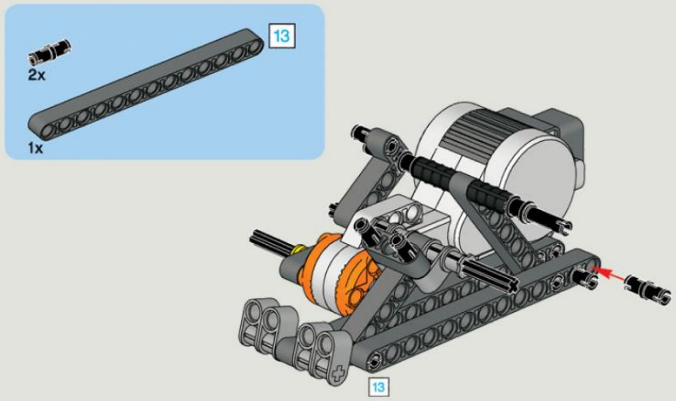
5.



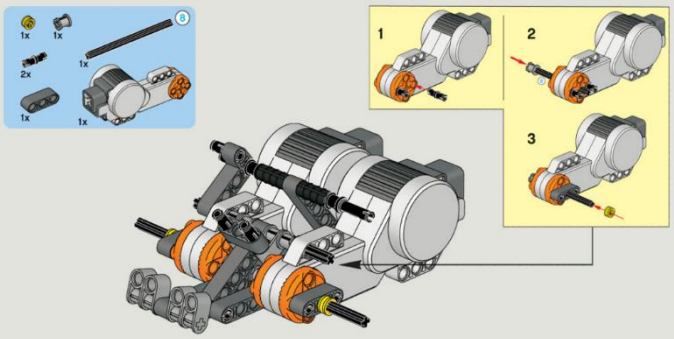
6.



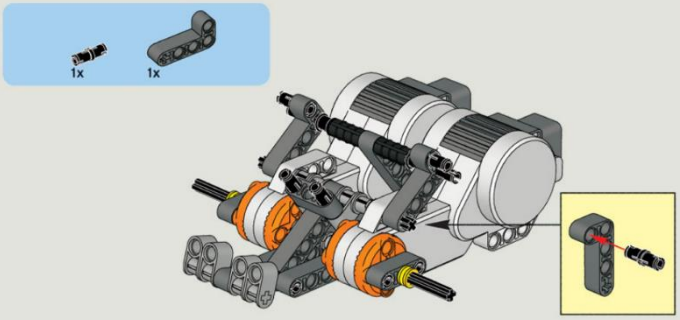
7.



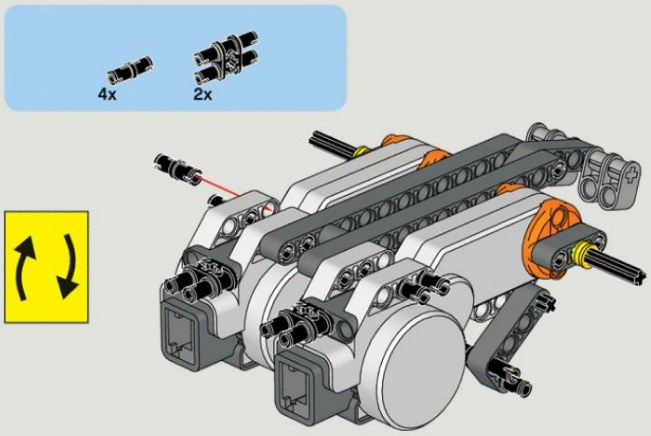
8.



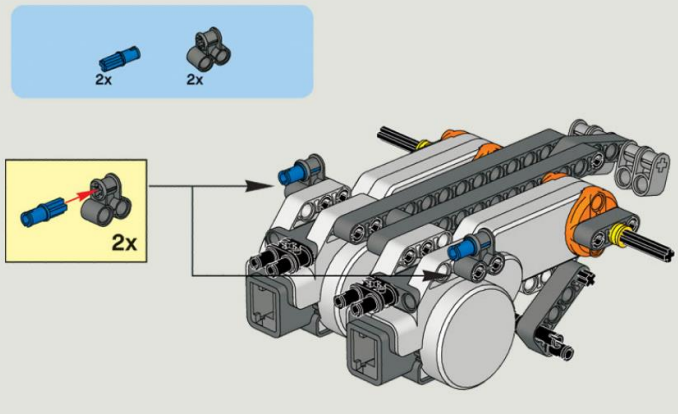
9.



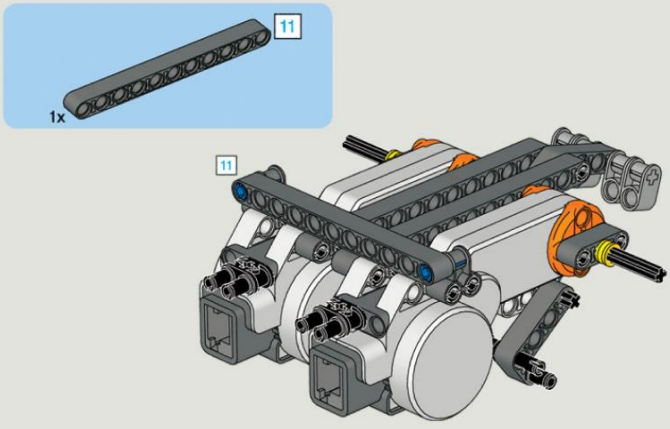
10.



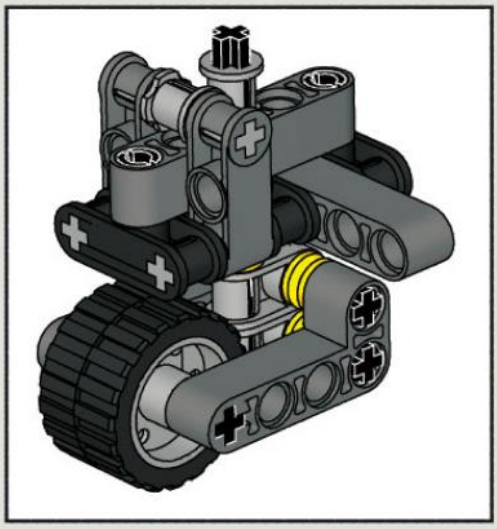
11.



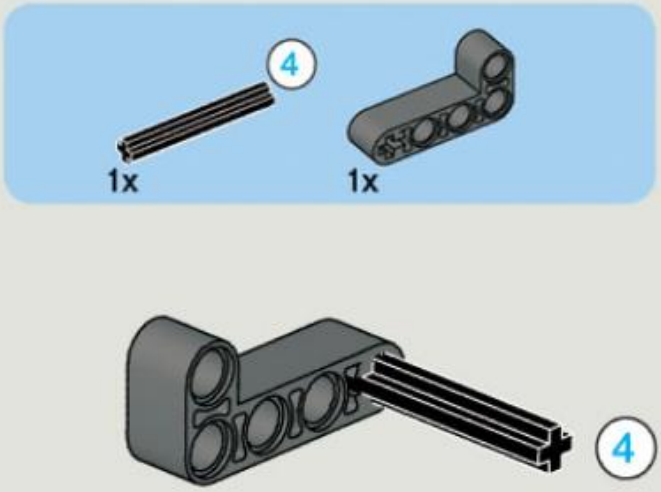
12.



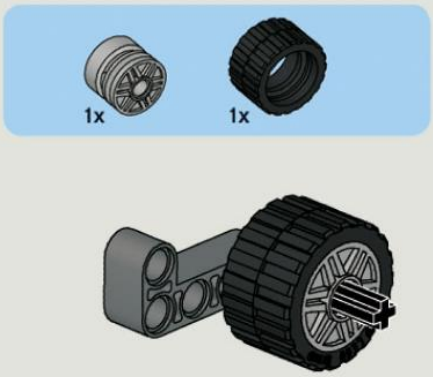
13.



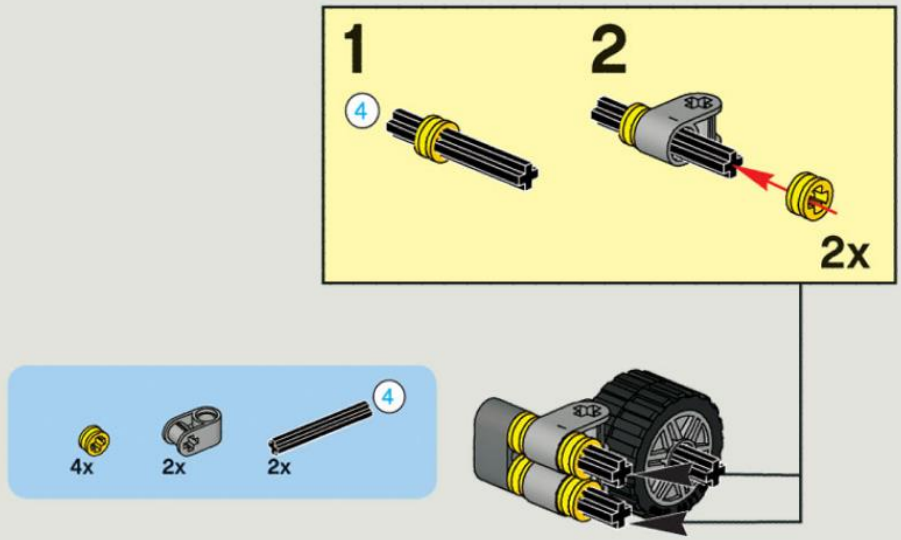
14.



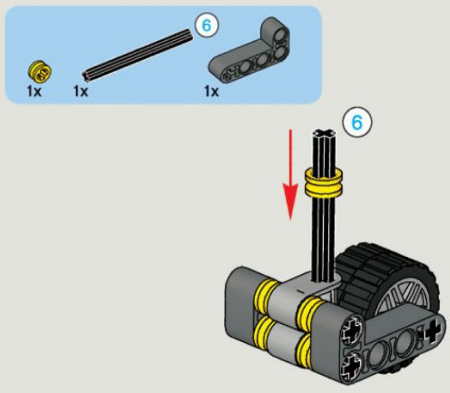
15.



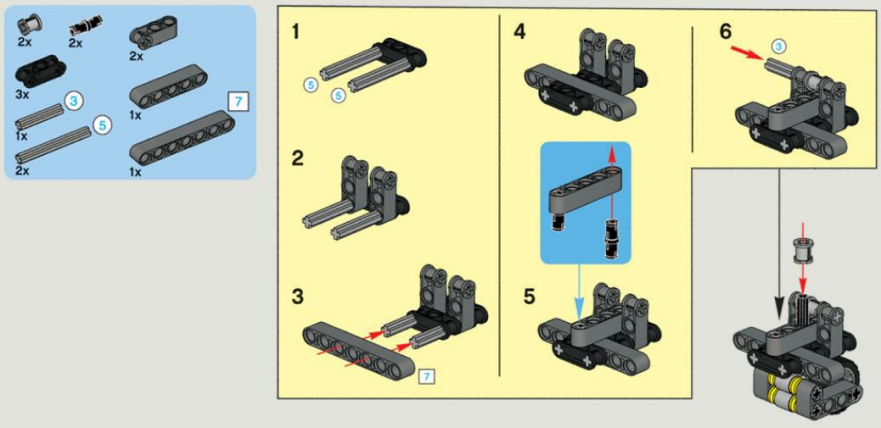
16.



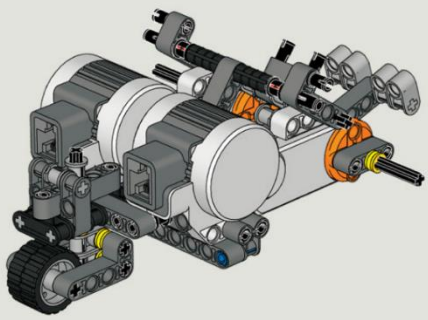
17..



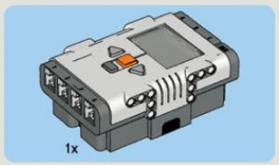
18.

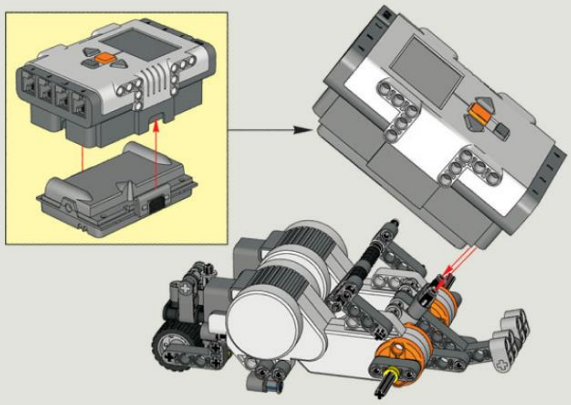


19.

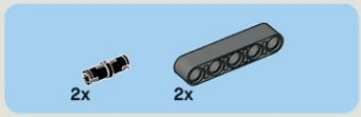


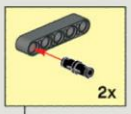
20.

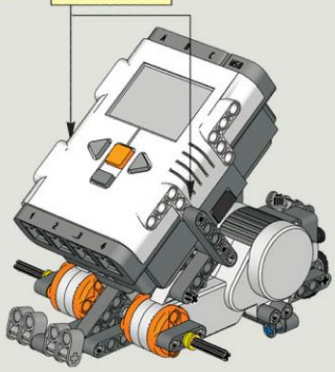




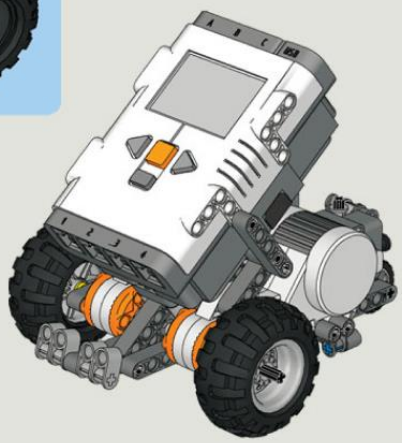
21.



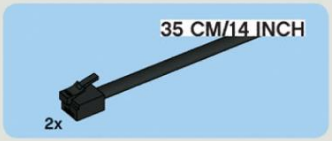


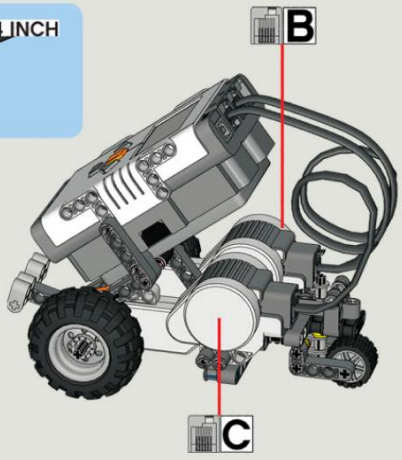


22.



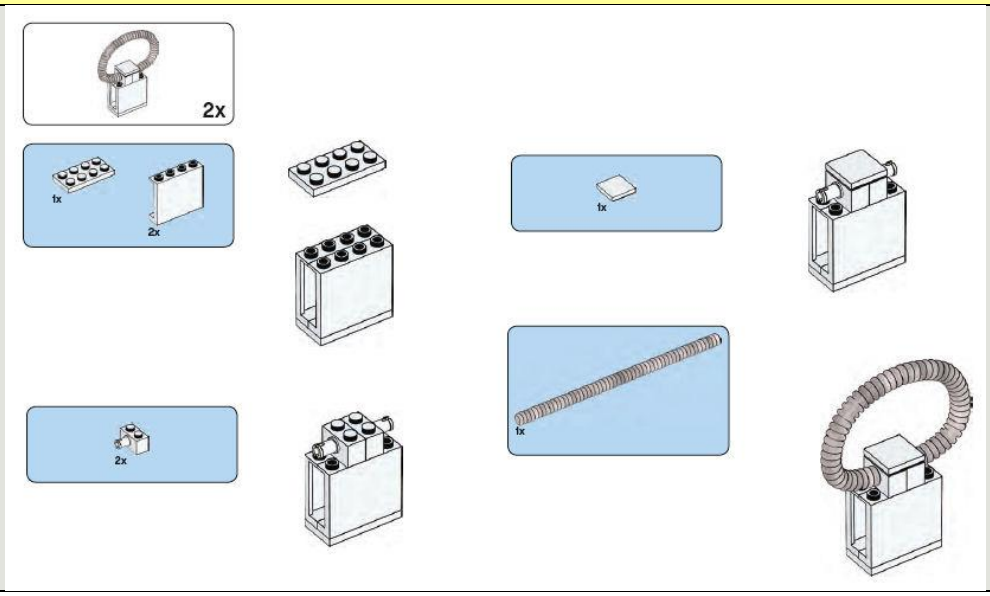
23.





3. MELLÉKLET: ÉPÍTÉSI ÚTMUTATÓ – KINCSEK

1. Kincs



2. Dupla kincs