szerző: Kiss Róbert

LEGO® EDUCATION SPIKE PRIME ROBOT

GYORS KEZDÉS

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

2020

Tartalomjegyzék

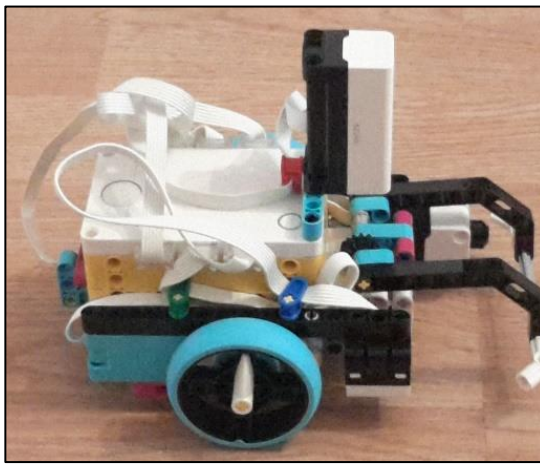
BEVEZETŐ

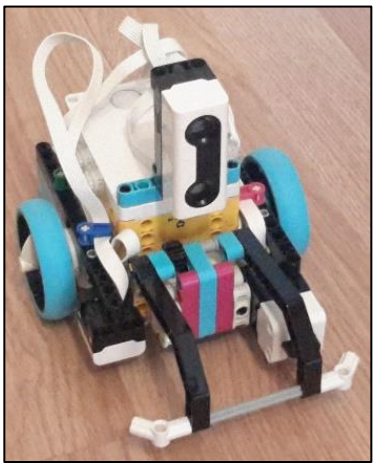
Ez a dokumentum azért jött létre, hogy a LEGO 2020-ban megjelent új eszközéhez a Spyke-hoz nyújtson bevezető programozási segítséget. Nem cél, hogy a programozás lehetőségeit részletesen bemutassa. Csupán kiemel néhány olyan példát, amit végig gondolva a további (bonyolultabb) feladatok megoldása már önállóan tanulható. Tehát az eszköz használatába befektetett idő lerövidítése a cél.

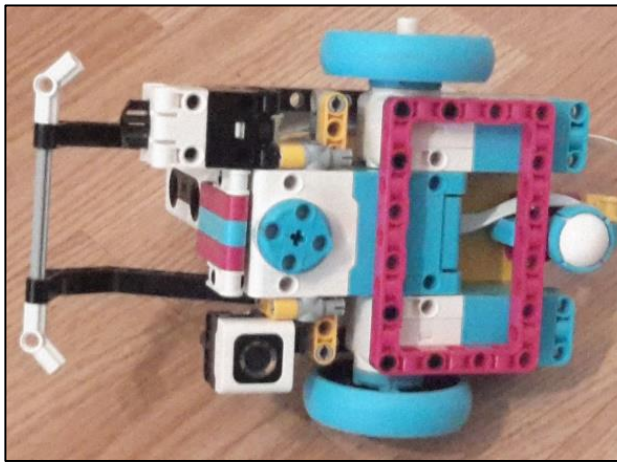
Nem mutatjuk be a Spyke szoftverében szereplő projekteket, mert ezek egyszerűen használhatók, zömében képi anyagra támaszkodnak. Csupán az alapprogramozás, projektektől független néhány eleme szerepel a továbbiakban.

Alaprobot

A programok teszteléséhez egy olyan robotot építettünk, amely guruló mozgásra képes, és amellyel az alapvető programutasítások tesztelhetők. Ennek építési útmutatója szerepel a szoftverben. Az alábbiakban néhány fotó mutatja be az alaprobotot. Természetesen sokféle robot építhető és a szoftver projektjeiben található ezekre sok példa is.







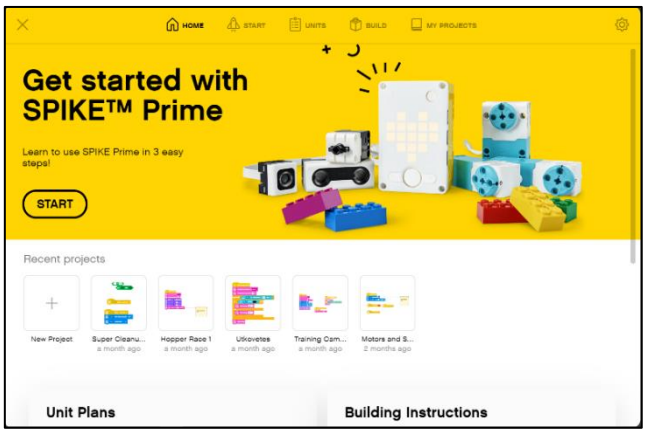
A robot mozgásához két médium motort használtunk, amely egy-egy kerékhez kapcsolódik, valamint egy harmadik motor képes a robot elejére épített „fogókart” mozgatni. Három szenzort helyeztünk el a roboton. Egy ultrahang szenzor található a robot tetején (90 fokkal elforgatva, függőleges helyzetben), egy fény/szín érzékelő jobb oldalt és egy ütközés/nyomás érzékelő a bal oldalon.

A robot központi egységén összesen hat (három-három a két oldalon) be/ki meneti port található. Ezek egyenrangúak, így bármelyikre csatlakozhat motor is és szenzor is.

Szoftverkörnyezet

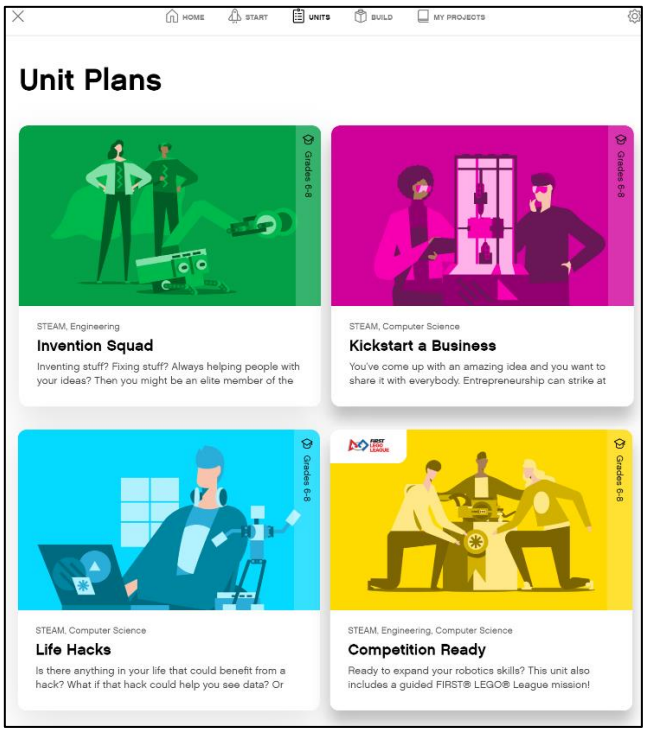
A szoftver ingyenes és letölthető a https://education.lego.com/en-us/downloads/spike-prime/software hivatalos weboldalról.

A szokásos telepítés és programindítás után a következő képernyő fogad.



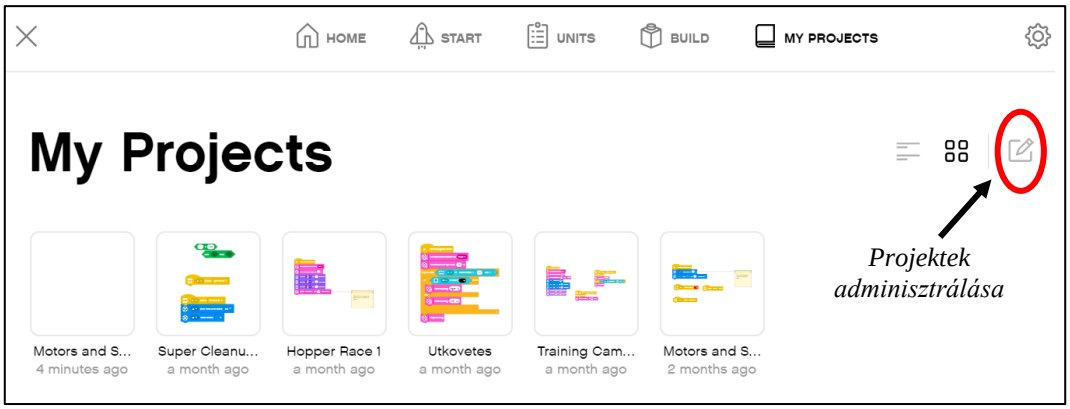
A képernyő alján megjelenő *Unit Plane* és *Building Instructions* menüpontoknál az eszközhöz mellékelt projektek (életkori besorolás + feladatspecifikáció + építési útmutató + programkód) és külön építési útmutatók érhetők el. Az alaprobot a *Driving Base* konstrukciók egy hibrid változata.

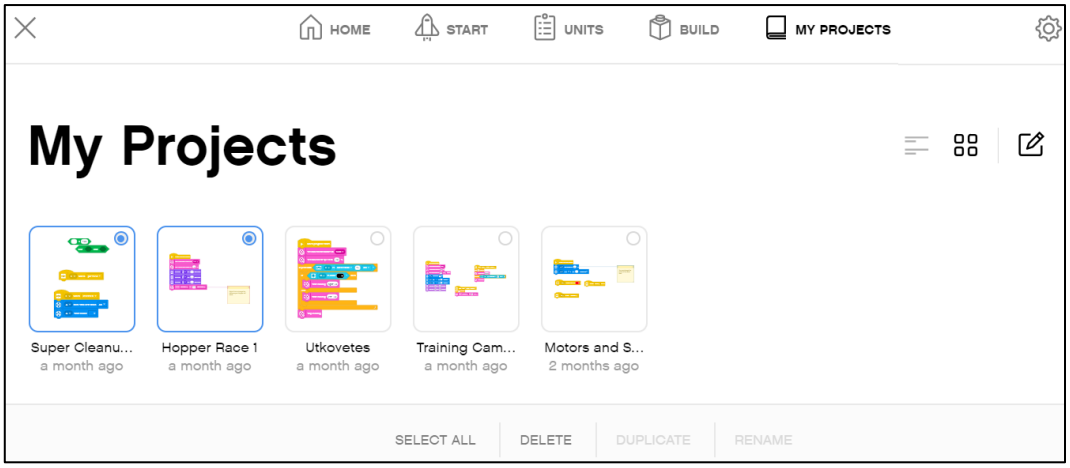
A kész projektek négy kategóriára bontva jelennek meg. Minden kategóriában több projekt is szerepel bonyolultsági sorrendben.

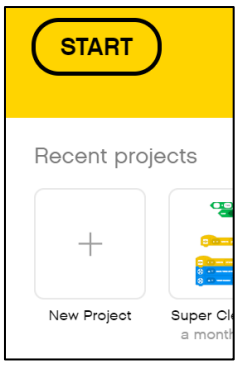


Ugyanezt a két menüt a felső menüsorból is el lehet érni a *Units* illetve *Build* menüpontokon keresztül.

A felső menü *My Projects* menüpontja a saját projektjeink elérését teszi lehetővé. Itt tudjuk adminisztrálni a projektjeinket (pl.: törölni, duplikálni, …).



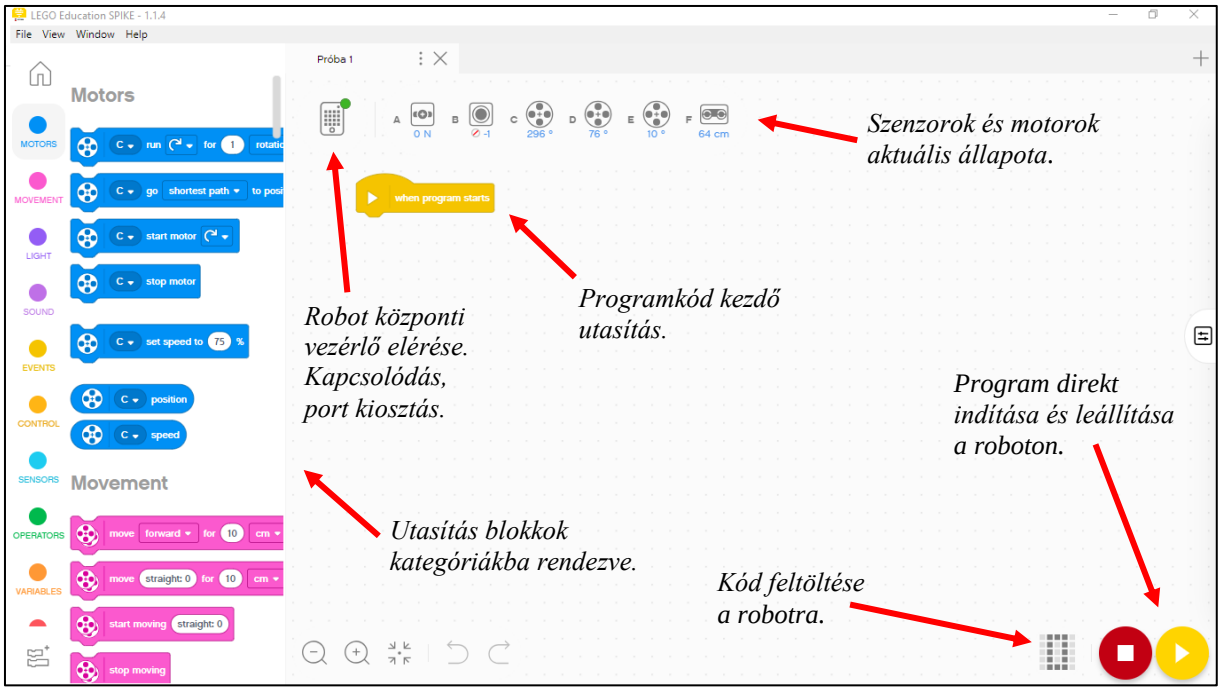




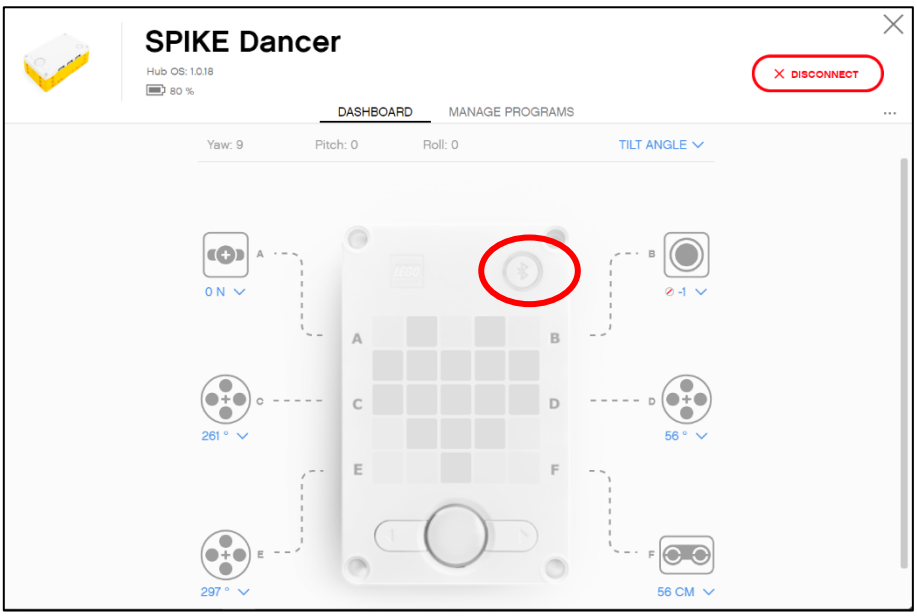
Új projektet létrehozni a nyitólap *Recent projects* területén a *New Project* ikonra kattintva tudunk.

A programkód *Scratch* alapú, blokkvezérelt. Tehát előre elkészített utasításblokkok egymásutánját kell felépíteni. Minden utasításblokkon beállíthatók az aktuális paraméterek, amelyek meghatározzák a speciális működést.

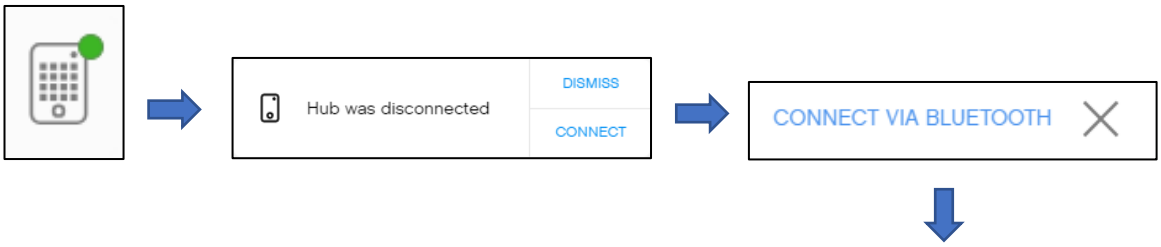
Az utasításblokkok kategóriákba rendezve a programablak bal oldalán találhatók. Minden kategóriának van egy színkódja. Az utasításblokk színe is megegyezik a kategória színkódjával.
Például a motorok egyszerű használatát lehetővé tévő blokkok színe kék (*MOTORS* csoport), míg a gurulni képes robot mozgatásáért felelős utasítások a rózsaszín kódszínt kapták (*MOVEMENT* csoport).
Ha a robotot (pl. USB kábelen keresztül) csatlakoztattuk a számítógéphez, akkor a megnyíló felületen közvetlenül elkezdhetjük írni a kódot és látjuk a képernyő felső részén az aktuális robot különböző portjain lévő eszközök típusát és a szenzorok, motorok aktuálisan mért értékét (a motoroknál az elfordulási szöget).



Ha bekapcsoltuk a robotot és csatlakoztattuk a számítógéphez USB kábelen keresztül, akkor aktívvá válik a projekt felületen a robot kapcsolati vezérlője. Itt egyrészt megnézhetjük a robot port kiosztását, valamint tudunk bluetooth kapcsolatra váltani (lebontva az USB kapcsolatot).

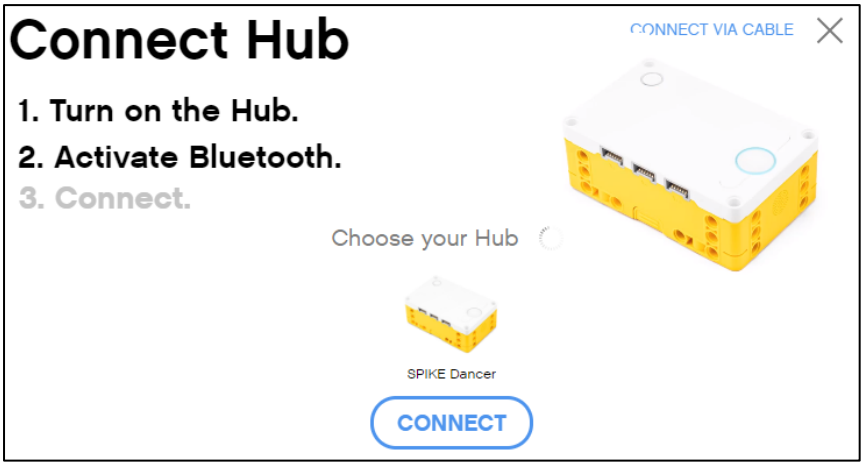


Az bluetooth kapcsolat felépítéséhez meg kell nyomni a robot vezérlő egységének jobb felső sarkában lévő gombot. (Lásd az ábrán pirossal bekeretezve.) Ha robot USB kábelen keresztül kapcsolódik, akkor tölt az akkumulátora, viszont a mozgások során zavaró lehet a kábel, így a vezeték nélküli kapcsolat kényelmesebb.



A sikeres kapcsolódást szöveges üzenet jelzi, illetve a kapcsolati panelen zöld színre vált a korábban piros jelzés.

Ha nem kapcsoljuk ki a robotot, akkor néhány perc használaton kívüli idő után automatikusan kikapcsol, így a bluetooth kapcsolatot is újra fel kell építeni.



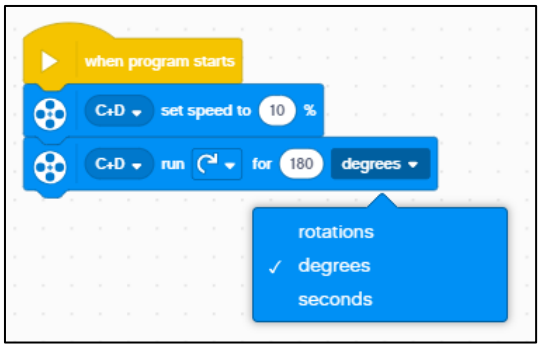
PROGRAMOZÁS

Motorok használata (MOTORS programcsoport)

Az első programunk legyen nagyon egyszerű. Azt szeretnénk, ha a robot a mindkét motorjának tengelyét elforgatná kb. 180 fokkal. Nézzük meg, hogy milyen mozgást végez közben a robot.

A motorok sebességét kell először beállítani. Ehhez külön blokkot kell használni.

Alapértelmezésben a motoroksebessége 75%, és csak egyetlen porton lévő motorra vonatkozik. Ha mindkét motor sebességét szeretnénk egységesen 10%-ra állítani. Kattintsunk a blokkon a motor portját jelző részre. A megjelenő panelen, ha bekapcsoljuk a *Multiple* funkciót, akkor több motornak egyidejűleg tudjuk a sebességét állítani.



A sebesség beállítása után kell megadnunk, hogy a motorok mennyit forogjanak. Ehhez egy újabb blokkot kell beilleszteni a programszálra.

A forgás mértékét szabályozhatjuk másodpercben (*seconds*), tengelykörbefordulások számában (*rotations*) és tengely elfordulási szögben (*degrees*). Mindegyik mértékegységet tudja a robot mérni valamilyen szenzorával. A beépített időzítővel (stopper), vagy a szervomotorban lévő elfordulási szög szenzorral. Itt nem adhatunk meg olyan mértékegységet, amelyet befolyásol a robot geometriája (pl.: centiméter). Ezzel el is készült az első programunk.

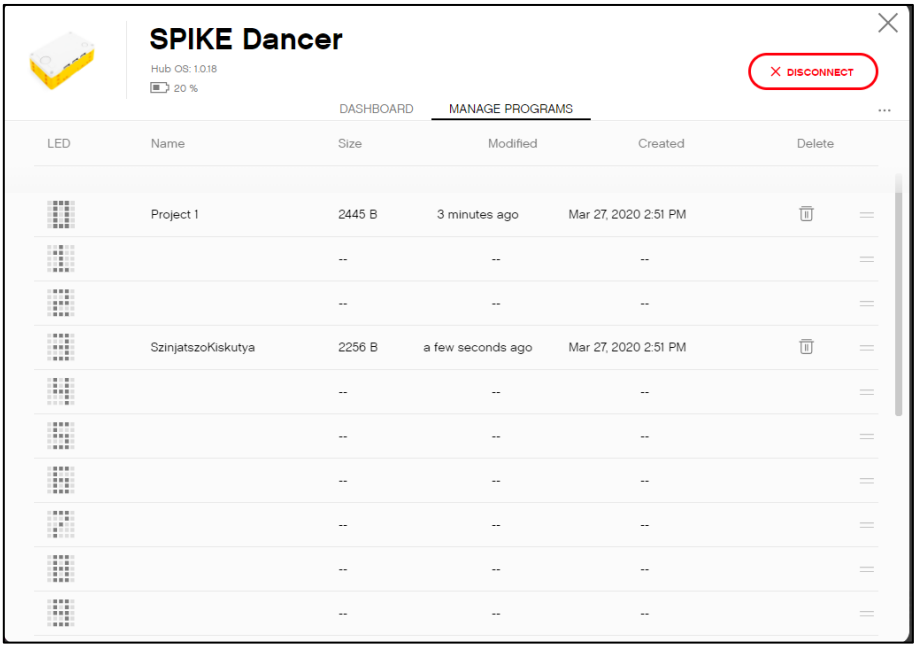
Mielőtt tovább vizsgálnánk a program működését nézzük meg, hogyan tudjuk a programot a robotra tölteni és elindítani.

Ha le szeretnénk futtatni egy programot, akkor a képernyő jobb alsó részén található sárga-fehér háromszög ikonra kell kattintani és a program rákerül a központi egységre és rögtön el is indul. Ezt csak akkor érdemes használni, ha bluetooth-on kapcsolódik a robot, mert egyébként az USB kábel zavaró lehet. Ha kábelt használunk akkor először rá kell tölteni a programot a robotra, majd a vezérlő egységen lévő gombbal elindítani.

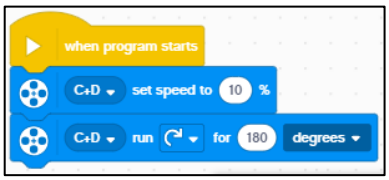
Ha programot rátöltjük a robotra, akkor először ki kell választanunk, hogy melyik tárolóhelyre szeretnénk tölteni. Összesen 20 tárolóhely áll rendelkezésre, 0-tól 19-ig számozva. Az aktuálisan kiválasztott tárolóhely számát láthatjuk a robot központi egységének led-mátrix képernyőjén és a szoftverben is. Az ábrán a „0” tárolóhely az aktuális. Tárolóhelyet váltani a jobbra-balra nyíllal lehet („<”, „>”). Ha a sárga programindító gombbal futtatjuk a programunkat, akkor az rákerül a robot aktuálisan kiválasztott tárolóhelyére és el is indul (ha már volt ott programunk, akkor automatikusan felülíródik). Ha panel felső részén lévő kapcsolóval *Download* funkcióból átkapcsolunk *Streaming*

funkcióba, akkor az indításkor nem töltődik a tárolóba a program, csupán elindítja a rendszer a roboton. Ez programteszteléseknél hasznos.

A robot egyes tárolóhelyein található programok listáját megtekinteni, illetve menedzselni, a szoftver központi egységét szimbolizáló ikonra kattintva, majd a *Manage Programs* lapot választva lehet.

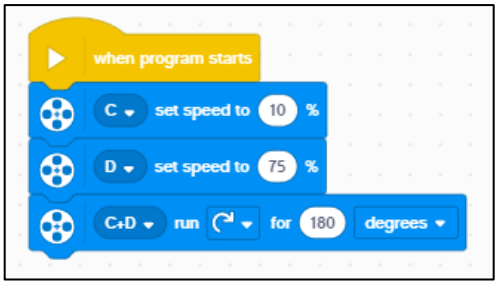


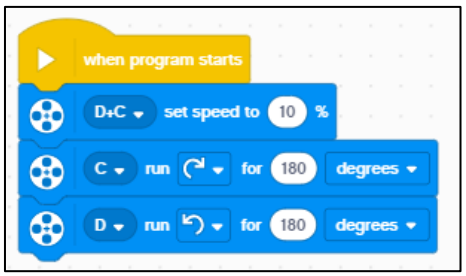
Térjünk vissza az eredetileg megírt első programunkhoz!



A működésnél azt látjuk, hogy a robot kb. 90 fokot helyben fordul. A blokkon be lehet állítani a motorok forgásirányát, de ez a két motor esetén csak azonos lehet, és mivel a két motor tükörszimmetrikusan helyezkedik el, ezért az azonos forgásirány éppen ellentétes forgást jelent. Tehát a két kerék ellentétesen forog, ezért fordult helyben a robot. A másik, amit érdemes észrevenni, hogy kb. 90 fokos volt az elfordulás. A kerekek átmérőjétől és távolságától függ, hogy mekkora tengelyelfordítás, mekkora fordulást fog a valóságban jelenteni. Az alapfelépítést használva a méretek olyanok, hogy a példánál beállított (180 fokos) tengelyfordulat kb. derékszögű fordulást jelentett. Más felépítésnél, vagy keréktípusnál ezt a szöget matematikailag számolni kell, vagy tapasztalati úton meghatározni.

Ha a motorok sebességét eltérőre állítjuk, az elfordulás akkor is kb. derékszögű csak a két motor mozgása eltérő időt vesz igénybe. A tengelyelfordulási szög sebességtől független érték.

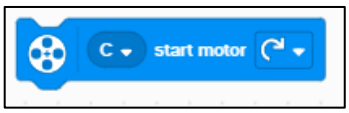




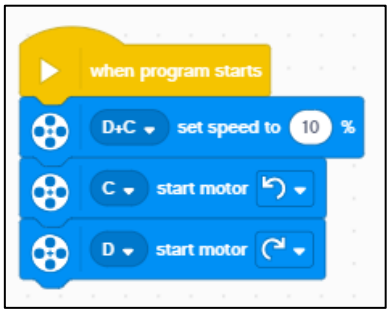
Próbálkozhatunk azzal, hogy a két motort külön-külön vezéreljük, de ellentétes irányú motorforgatással.

A robot ekkor sem megy egyenesen, mert a program végrehajtása során a vezérlés addig nem lép a következő utasításra, amíg a blokkon beállított feltétel be nem következik. Tehát jelen esetben, amíg a C motor ki nem forogja a 180 fokot, addig nem indul el a D motor.

Tehát ha valamilyen feltételt állítunk be a motorok működési „idejének” szabályozására (pl. szög, vagy másodperc, …), akkor nem indul el az utasítással következő parancsa, amíg az előző le nem futott.

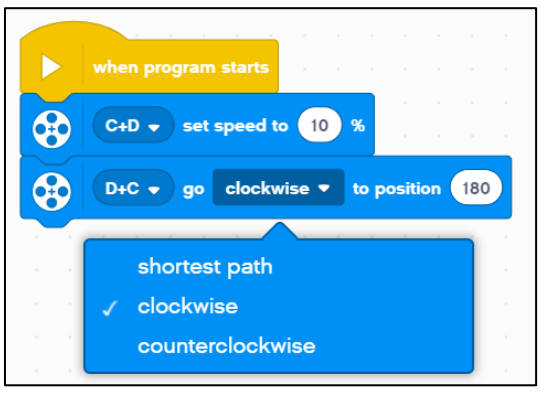


Van azonban egy olyan blokk, amely nem tartalmaz feltételt a motorok vezérlésére, csak bekapcsolja őket. Ennél a blokknál addig forog a motor, amíg direkt le nem állítjuk a szoftverből, vagy a stop motor blokkal le nem állítjuk, vagy le nem merül az akkumulátor. A motor bekapcsolása után rögtön a következő utasítás végrehajtására kerül sor. Ezzel a blokkal tudjuk egyenesen mozgatni a robotot, ha a két motort ellentétes forgásiránnyal mozgatjuk.



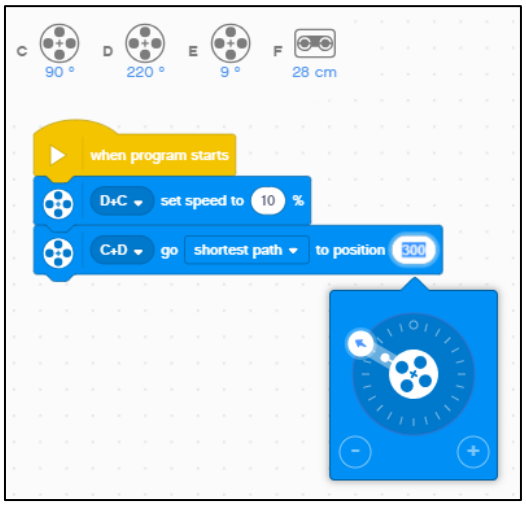
A robot egyenesen előre halad 10%.os sebességgel, amíg le nem állítjuk a programot. Ha a stop motor blokkot használjuk, akkor az indítás és a megállítás közé kell tennünk valamilyen várakozásra kényszerítő feltételt, mert egyébként nem indul el a robot. (Elindulna, de rögtön meg is állítjuk a motorokat.)

A MOTORS programcsoportban van egy további érdekes és hasznos utasításblokk. A robot motorjának van egy 0 fokos pozíciója. Ehhez képest 0-359 közötti skálán tudja mérni, hogy a kerék éppen milyen pozícióban áll. Megadhatjuk azt, hogy forgassa a kereket a 0-359 közötti pozíciók valamelyikébe. Háromféle módon tudja ezt a forgatást elvégezni. A legrövidebb úton (*shortest path*). Ekkor a forgatás

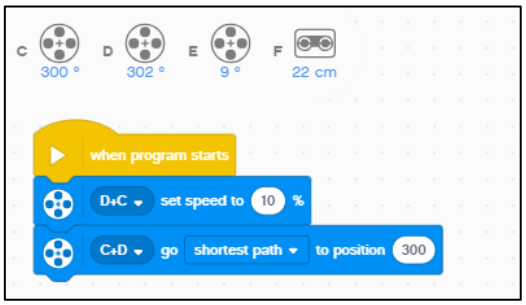


irányát aszerint dönti el, hogy a megadott pozíciót az aktuálishoz képest milyen úton éri el leggyorsabban (óramutató járásával megegyezően, vagy fordítva). Direktben is megadhatjuk a forgatás irányát: óramutató járása szerint (*clockwise*) vagy ellentétesen (*counterclockwise*). Ebben az esetben előfordulhat, hogy 359 fokot fordít a motoron (ha pl. 1 fokon állt és 0 fok a cél pozíció és a forgásirány az óramutató járása szerinti, akkor növekedő irányba forgatja 359 fokig, majd még egy fokot. Ha két motort állítunk be,

akkor mindkét motort addig forgatja, míg el nem éri a célpozíciót. Az alábbi két képen látható a programindítás előtti motorpozíció és a megállás utáni pozíció.



A képen látszik, hogy 300 fokos pozícióba szeretnénk a robot motorjait állítani. A programindítás előtt a C motor 90 fokos, a D motor 220 fokos pozícióban állt.



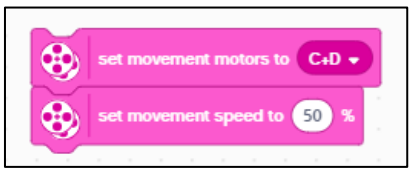
A program lefutása után a C motor 300 fokos pozícióba került, míg a D motor 302 fokos pozícióban állt meg. Mivel a legrövidebb út beállítást használtuk, azért a két motor ellentétesen forgott. A C motor 90 fokról csökkenően 300 fokra, míg a D motor 220 fokról növekvően 302 fokra.

A megépített robot kerekén található egy jelölés, amelyet kezdéskor beállíthatunk a 0 fokos pozícióhoz, így vizuálisan is láthatjuk a motorok pillanatnyi állását.

Ezt a blokkot leginkább nem a robot mozgatására érdemes használni, hanem olyan konstrukciók esetén, ahol a motor valamilyen emelőt vagy forgató kart mozgat. Itt nagy szerepe lehet a motorpozícióknak.

Mozgásvezérlés (MOVEMENT programcsoport)

A rózsaszínnel jelölt blokkok szintén a robot mozgásának irányítását teszik lehetővé. Ezekkel az utasításblokokkal egyszerre két motort tudunk vezérelni. A beállítások részben hasonlóak a MOTORS blokkoknál bemutatottakhoz, így csak néhány speciális lehetőséget emelünk ki.

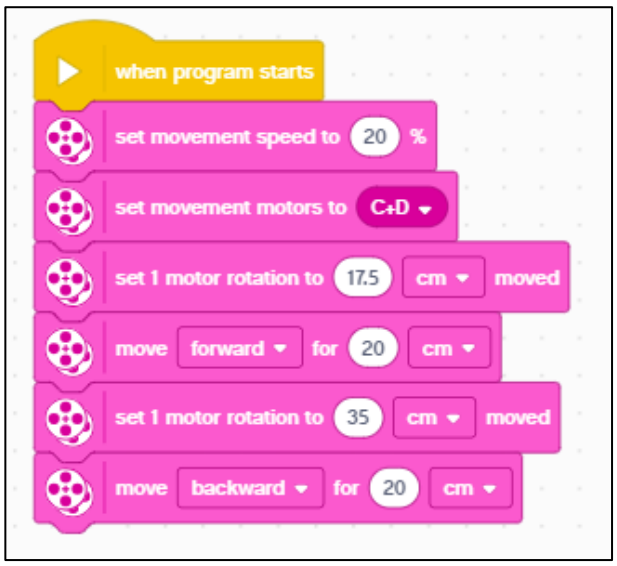


Külön utasításblokkal kell beállítani, hogy melyik két porton lévő motort szeretnénk használni, illetve, hogy mekkora legyen a motorok sebessége. Az utasításblokkon szerepel a „*set*” szó. Ez utal arra, hogy a működést befolyásoló értékek beállítására szolgál. Még egy „*set*” parancs található a listában. A MOTORS programcsoportnál utaltunk arra, hogy a motorok működési „idejét” csak olyan értékkel lehet vezérelni, amelyet a robot mérni tud (elfordulási szög, idő) és értéke nem függ a robot felépítésétől. Ahhoz, hogy például valamilyen távolság

mértékegységben tudjuk megmondani a robotnak, hogy mennyit mozogjon, ismerni kell a kerekek átmérőjét. Egy egyszerű egyenes arányossággal innen kiszámítható a robot haladási távolsága. Ugyanis, ha a robot kerekének átmérője pl. 10 cm, akkor ezt megszorozva π-vel (3.14), megkapjuk a 31.4 cm-es értéket. Ennyit halad előre a robot egy teljes kerékfordulat alatt (a kerék kerületének megfelelő hosszúságot). Ebből már vissza lehet számolni, hogy adott távolság (cm) esetén hány fokot kell fordítani kerekeken. Ezt a számolást végzi el helyettünk egy utasításblokk.

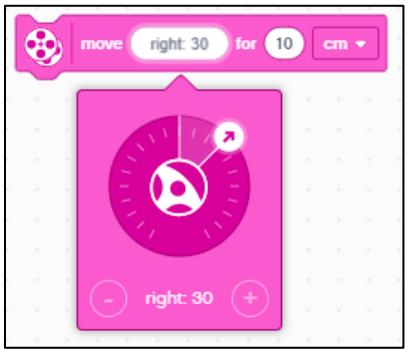
Azt kell megadni, hogy 1 kerékfordulatnál hány cm-t mozog a robot. Tehát a kerék kerületének hosszát kell a szövegdobozba írni. Alapértelmezésben ez 17.5 cm, tehát a standard kerék átmérője: 17.5/3.14 = 5.57 cm. Ha ettől eltérő átmérőjű kereket használunk, akkor a szövegdobozba a megfelelő kerület mértéket kell írni.

Teszteljük le a működést azzal, hogy a robotot 20 cm távolságban előre mozgatjuk, majd átállítjuk a kerék kerületet 35-re (tehát mintha kétszer akkora kerekünk lenne), és megpróbáljuk 20 cm-t hátra mozgatni.



Tapasztalatként azt mondhatjuk, hogy a robot valóban kb. 20 cm-t mozgott előre 20%-os sebességgel, majd elindult hátrafelé és kb. az indulási- és célpozíció között félúton állt meg.

A magyarázat logikus. A tolatás előtt megváltoztattuk a kerékátmérőre vonatkozó paramétert a kétszeresére. Ha kétszer akkora átmérőjű kerék lett volna a roboton, akkor ugyanazon tengelyelfordulás, kétszer akkora távolságot jelentett volna. Ugyanez fordítva: ha a távolság ugyanaz maradt, akkor fele annyit kell fordítani a kereken az eléréséhez. Mivel a kerékátmérő a valóságban nem változott, ezért a fele akkora tengelyelfordulás fele akkora tolatási távolságnak felelt meg. A blokkba tehát mindig az aktuális keréknek megfelelő kerülethosszt kell beírni (ami a kerékátmérőtől csak egy konstans szorzóban különbözik (3.14)).

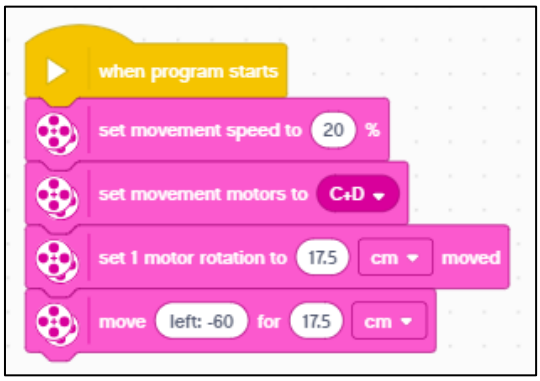


A MOVEMENT csoportban, a kanyarodó mozgás esetén a mozgásvezérlésre van egy új blokk, amelynél egy kormánykerékkel tudjuk beállítani, hogy a két motor sebessége között „mennyi legyen a különbség”.

A haladás távolságát centiméter helyett tengelyelfordulási szögben, másodpercben is megadhatjuk.

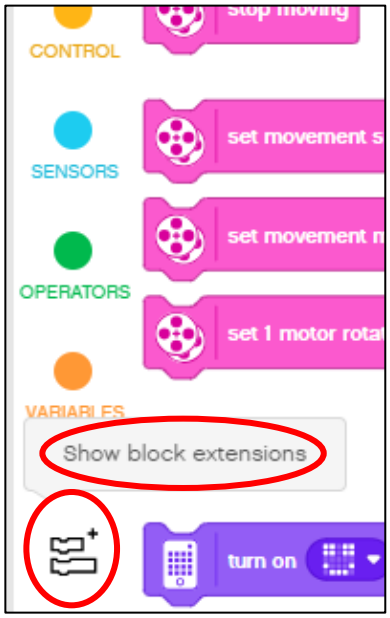
A két kerék eltérő sebességgel fog forogni, ebből következően a robot kanyarodik.

Ha centiméterben adjuk meg a haladás távolságát, akkor a két kerék által megtett távolság átlaga lesz a beállított távolság. Minél nagyobb a megadott elfordulási érték, annál élesebb kanyarban fordul a robot. Ha a beállított érték +100 vagy -100, akkor a robot helyben fordul (a két kerék ellentétes irányban forog).

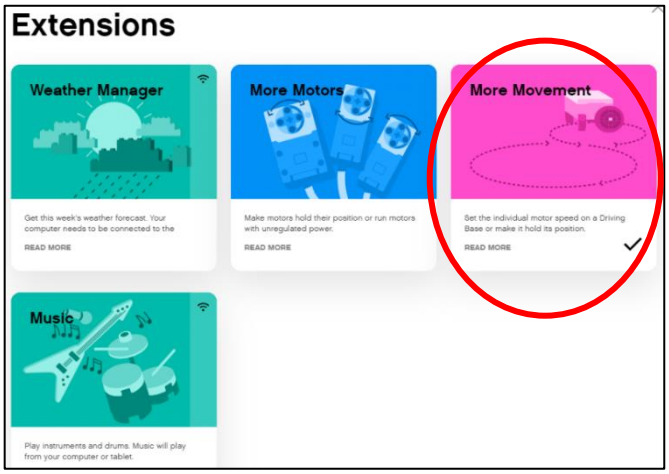


Bár számolással lehet például a fordulás ívét pontosítani, de alapvetően inkább tapasztalati értékek megadására épül a kanyarodás.

A bal oldalt látható programnál a robot balra fordul úgy, hogy a két kerék elfordulási szögének átlaga 360 fok (egy teljes tengelyfordulat 17.5 cm-es előrehaladást jelent). A mozgás olyan, mintha egy autónál a kormányt 60 fokkal fordítottam volna balra. (Az autónál a kerekek fordulnak, itt viszont a forgási sebességük változik.)



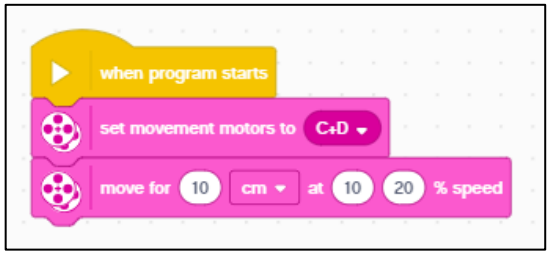
Ha a szoftver bal alsó sarkában megjelenő ikonra kattintunk, akkor további blokkok jelennek meg például a MOTORS és MOVEMENT programcsoporthoz. Ezek a haladó (*advanced*) blokkok összetett motorvezérlést biztosítanak, bár a legtöbbjük megvalósítható az egyszerű blokkokból összeállított kódokkal is.

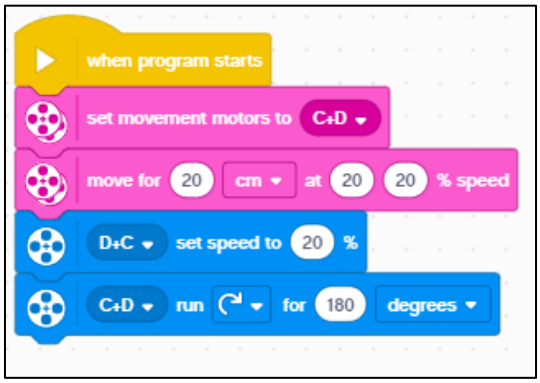


Bekapcsolva a *More Movement* opciót, a programcsoportok alján megjelennek az elérhető összetett vezérlést biztosító blokkok.

Ezek között szerepel egy olyan, amely segítségével a két motor sebességét külön-külön tudjuk állítani. A fordulás ettől sem pontosabban, sem könnyebben nem számítható, de más típusú feladatmegoldást tesz lehetővé (lásd később).

Például egy lehetséges kód, amelynél a robot kanyarodva halad 10 cm-t szerepel a jobb oldali ábrán.

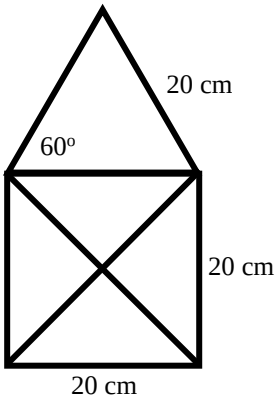




A két programcsoport (MOTORS és MOVEMENT) blokkjait természetesen együtt is tudjuk használni. A példában a robot 20 cm-t halad egyenesen előre, majd derékszögben befordul. (A kódot más módon is el lehet készíteni.)

1. Példa – Alakzat bejárás

Írjon olyan programot, amelyet végrehajtva a robot bejárja az ábrának megfelelő útvonalat (fekete színű szakaszok)! A mozgás során nem haladhat át kétszer ugyanazon az útvonalszakaszon! Az útvonal méreteit az ábrára írt számok értelmezik. A robot az alakzat bármelyik csúcsából indulhat.

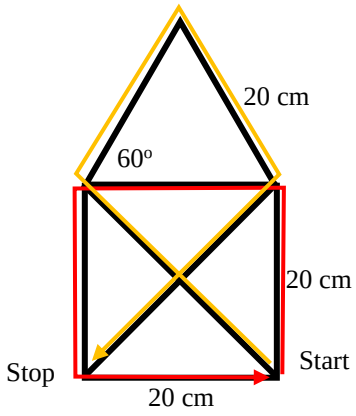


Megjegyzés

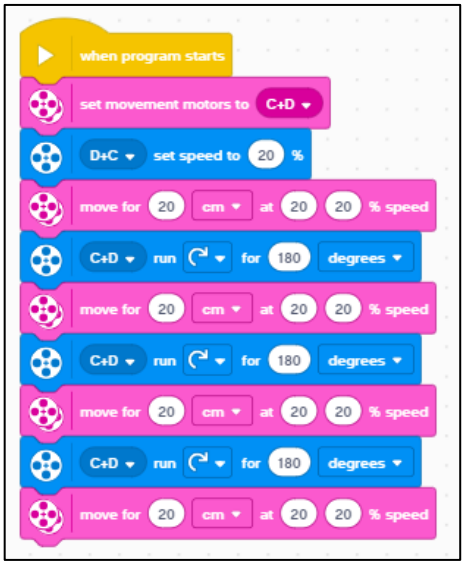
A fekete vonalhálózat egyetlen összefüggő vonallal (a toll felemelése nélkül) megrajzolható úgy, hogy minden szakaszt pontosan egyszer rajzolunk meg. Általános iskolából ismert egyszerű gráfbejárás feladat.

Megoldás

A megoldás forráskódját lépésenként magyarázzuk. A bejárás útvonalát szemlélteti az alábbi ábra.



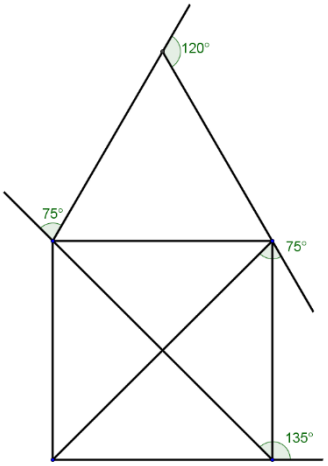
Először a piros színű útvonalon haladunk végig. Tehát egy négyzetet ír le a robot mozgás közben. Ennek a kódja pl.:



A kódot lehet rövidíteni, hiszen vannak benne ismétlődő elemek (egyenesen előre és fordul 90 fokot). Erre megoldást a következő fejezetekben látunk. A sebességeket elegendő egyszer beállítani pl. a program elején. Ha nem állítjuk be a

kerékátmérő mértékét, akkor az alapértelmezettet használja, ami 17.5 cm, így a 20 cm-es előre mozgások távolságát az alaprobotnál helyesen értelmezi.

A piros színű útvonalról (lásd ábra) áttérünk a sárga útvonalra. A négyzet bejárása után, a robotnak fordulnia kell 135 fokot. A robot aktuális iránya és az új haladási irány között éppen 135 fokos fordulóra van szükség ($90^o + 45^o$). Ezután Pithagoras tételével kiszámíthatjuk a 20 cm oldalhosszú négyet átlójának hosszát: $\sqrt{20^2 + 20^2} \approx 28.3$ cm. Ennyit kell egyenesen haladnia, majd fordulnia 75^o-ot ($180^o - 60^o - 45^o$). 20 cm-es előre haladás után egy 120^o-os forduló jön ($180^o - 60^o$). És így tovább … Az alábbi ábra segít a szögek értelmezésében:



A fordulási szögekhez meg kell határozni a tengelyelfordulási szögeket. Egyenes arányossággal számítható. Ha tudjuk, hogy 90^o-os robotelforduláshoz 180^o-os tengelyfordulat szükséges, akkor ez alapján számíthatók a szükséges tengelyelfordulási szögek.

A mozgásnál ügyelni kell a fordulás irányára, ami a program második részében éppen ellentétes a korábbival.

A teljes kód a fenti ábrán látható (az ábra melletti jelölés a piros és sárga útvonalrészt jelenti).

Természetesen a kód egyszerűsíthető, de már a motorok használatával is megoldhatók komplexebb feladatok.

Vezérlési szerkezetek (CONTROL programcsoport), szenzorok egyszerű használata (SENSORS programcsoport)

A programnyelveknél a „vezérlési szerkezetek” kifejezéssel a legtöbb esetben három olyan programozási konstrukciót jelölnek, amelyek az összetevők (utasítások) végrehajtási sorrendjét határozzák meg.

1. Szekvencia

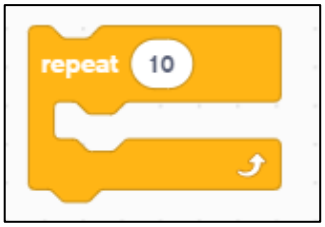
Az utasítások (blokkok) egymás utáni végrehajtása. Gyakorlatilag, amikor a programnyelvben fentről-lefelé egymás után fűzzük az utasításblokkokat, akkor ezzel a végrehajtási sorrendet is megadjuk.

2. Ciklus (iteráció)

Az utasítások egy feltételtől függően többször hajtódnak végre. A feltétel lehet például egy egyszerű számlálás, amikor előre megadjuk, hogy hányszor történjen a végrehajtás, vagy lehet egy esemény bekövetkezése.

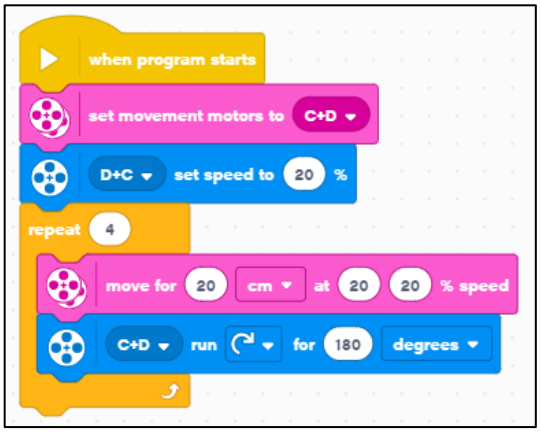
3. Elágazás (szelekció)

Egy feltételtől függően a programnak adott helyzetben mást kell elvégeznie. A feltétel sok esetben egy esemény vagy logikai állítás, amelynek bekövetkezése, vagy igazzá válása esetén a programnak mást kell csinálnia, mint egyébként. Bonyolultabb esetben nem csak kétféle kimenetelt lehet beállítani, hanem megszámlálhatóan sokat.



A ciklusok közül a legegyszerűbb az úgynevezett növekményes ciklus, amelynél előre tudjuk, hogy az utasításszálon szereplő blokkok egy csoportját hányszor kell egymás után végrehajtani. Minden ilyen ciklushoz tartozik egy számláló (ciklusváltozó), amelynek az értéke minden egyes végrehajtás során eggyel nő (0-tól kezdve a számlálást), így elérve a beállított értéket a ciklus futása leáll és a programszál következő utasításával folytatódik a végrehajtás. Az ismétlendő utasítások alkotják a ciklusmagot, amelyet a blokk két vízszintes eleme közé kell helyezni. A ciklusfejléc mutatja az ábrán, hogy jelenleg 10-szer fogja végrehajtani a program a ciklusmagban szereplő utasításokat.

Az előző fejezetben bemutatott 1. példa rövidíthető a növekményes ciklus használatával, mert vannak benne ismétlődő utasítások. Például a négyzet alakú pályán történő mozgás kódját mutatja a jobb oldali ábra.



Sok esetben nem tudjuk előre megmondani, hogy hányszor kell végrehajtani a ciklusmag utasításait. Az ismétlések száma valamilyen feltételtől, eseménytől függ. Ilyen eseményt generálhatnak például a szenzorokkal mért értékek.

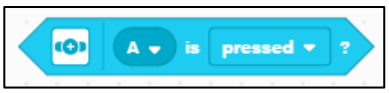
Jelenleg az alaprobotunkra három szenzor csatlakozik:

A port – ütközés/nyomás érzékelő
B port – fény/szín érzékelő
F port – távolság érzékelő

A robot szenzorai kétféle típusú értéket adhatnak vissza a programnak: logikai értéket vagy számot. A logikai érték igaz/hamis lehet, míg a számok akár előjeles valós számok is (de jellemzően nemnegatív egészek).

A SENSORS programcsoportban szereplő blokkok ennek megfelelően az alakjukban eltérnek. A logikai értéket visszadó blokkok hatszög alakúak, míg a számértéket visszaadó blokkok lekerekített végű téglalapok. Ezeket az utasításblokkokat tehát nem közvetlenül a programszálra kell kapcsolni, mint utasítást, hanem az általuk szolgáltatott értékeket lehet felhasználni más blokkok paramétereiként.

Az A portra kötött ütközésérzékelő által visszaadott logikai érték (be van nyomva/nincs benyomva → igaz/hamis).



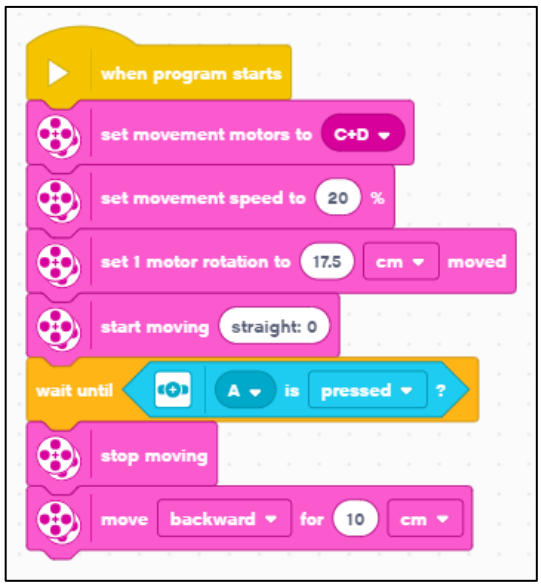
A B portra kötött fényszenzor által visszaadott fényintenzitás érték (a visszavert fény intenzitása → 0-100 közötti szám, minél sötétebb, mattabb a felület, annál kisebb).

A mért értékek felhasználhatók a vezérlési szerkezeteknél a ciklusok, elágazások feltételeinek megadására.

2. Példa – Akadály észlelés

Írjunk olyan programot, amelynél a robot egyenesen előre halad mindaddig, amíg az ütközésérzékelőjét be nem nyomták! Ekkor tolasson vissza 10 centimétert!

Megoldás



A cikluson belül most nincsenek végrehajtandó utasítások, mert a mozgást a ciklus előtt el lehet indítani. A ciklusnak csak annyi a szerepe, hogy figyelje az ütközésérzékelőt és várjon mindaddig, amíg be nem következik az az esemény, hogy megnyomták a szenzoron lévő gombot. Erre alkalmas a „*wait until*” szerkezet. Ennek hatására addig várakozik a blokknál a programszál utasításainak végrehajtása, amíg a beállított feltétel teljesül. Ezután a végrehajtás a következő utasítással folytatódik.

A motort a „*wait until*” blokk előtt kell elindítani (a mozgás „időtartamát” ne adjuk meg, hiszen nem tudjuk mikor fogják megnyomni az ütközésérzékelőt). A „*wait until*” blokk után pedig állítsuk meg a mozgást (*stop moving*). Az utolsó utasítás a 10 cm-es tolatás.

3. Példa – Adaptív sebesség radar

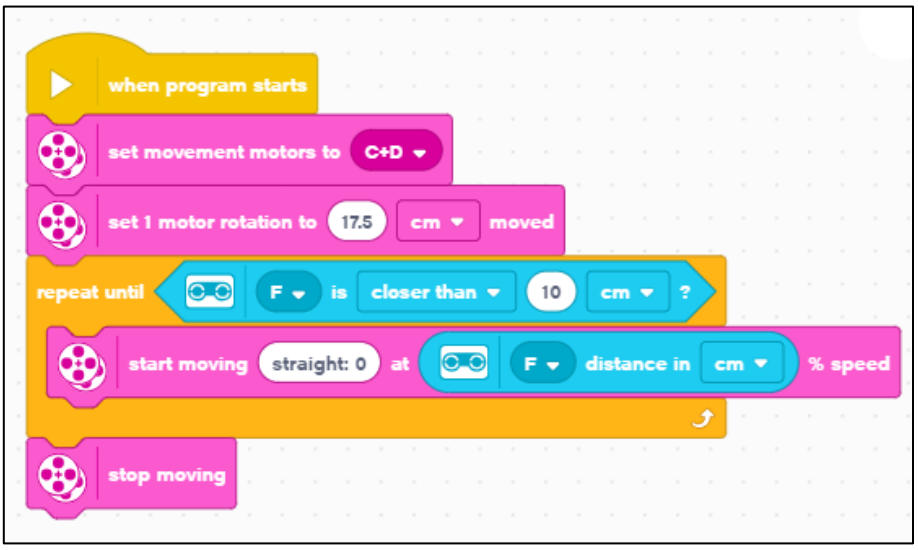
Írjon programot, amelyet a robot végrehajtva egy adaptív sebességtartó automatikát szimulál! Az autókba beépített adaptív sebességtartásnak a lényege, hogy ha akadály kerül (pl. lassabb jármű) az autó elé, akkor csökkenti a sebességet az ütközés elkerülése érdekében.

A feladat szerint a robot haladjon állandó sebességgel (ez legyen 100%), és az ultrahang szenzorával mért távolság függvényében változtassa ezt a sebességet! Ha 10 cm-re megközelítette az akadályt, akkor álljon meg!

Megoldás

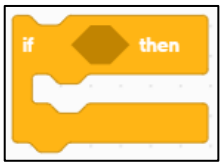
A szokásos „*set*” beállítások után egy úgynevezett előltesztelő ciklust használunk. Itt nem tudjuk előre, hogy hányszor kell végrehajtani a ciklusmagot. A leállítást egy szenzor által adott feltétel fogja garantálni. A ciklus futása során folyamatosan figyeljük a távolságérzékelő aktuális értékeit, és ha 10 cm-nél kisebb értéket érzékel a rendszer (*closer than 10 cm*), akkor a feltétel igaz lesz és a ciklus leáll. A ciklusmagban egyetlen utasítás szerepel. A MORE MOVEMENT csoportban található blokknál a mozgás sebességét lehet megadni. Szerencsére a motor sebessége nem címezhető túl. Ezt azt jelenti, hogy motorsebességként 0-100 közötti érték adható, de ha nagyobb számot kap a rendszer, akkor nem ad hiba üzenetet, hanem a maximális 100%-os sebességet tartja. Tehát a távolságmérő pillanatnyi értéke fogja meghatározni a robot sebességét. Egy fal felé közeledve ez egyre kisebb érték lesz, így a robot is folyamatosan lassul, míg eléri a 10 cm-es értéket. Ekkor megáll, mivel leáll a ciklus is.

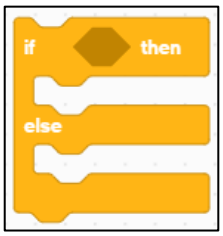
Az előltesztelő jelző a ciklusnál azt jelenti, hogy a ciklusmag utasításainak végrehajtása előtt megvizsgálja a feltételt. Ha az már a kezdéskor is igaz, akkor egyszer sem hajtja végre az utasításokat. Tehát ha a robot 10 cm-en belül áll az akadály előtt, akkor el sem indul.



Alapvetően megjegyezhető, hogy a blokkokon belül minden hatszög alakú szövegdobozba (logikai érték) betehető logikai feltétel, míg a kerek szélű téglalap alakú szövegdobozokba (szám típusú érték) olyan blokkok, amelyek számokat adnak eredményül.

A harmadik vezérlési szerkezet az elágazások. Itt egy logikai feltételtől függően más-más utasítást tud a robot végrehajtani. Lehet olyan elágazást készíteni, amelynél a feltételtől függ, hogy végrehajtsa-e az utasítást a robot, de ha nem teljesül a feltétel, akkor ne csináljon semmi mást, és lehet olyan elágazást létrehozni, ahol van egyébként ág, tehát ha a feltétel nem teljesül, akkor megmondhatjuk, hogy mi legyen a végrehajtandó utasítás.





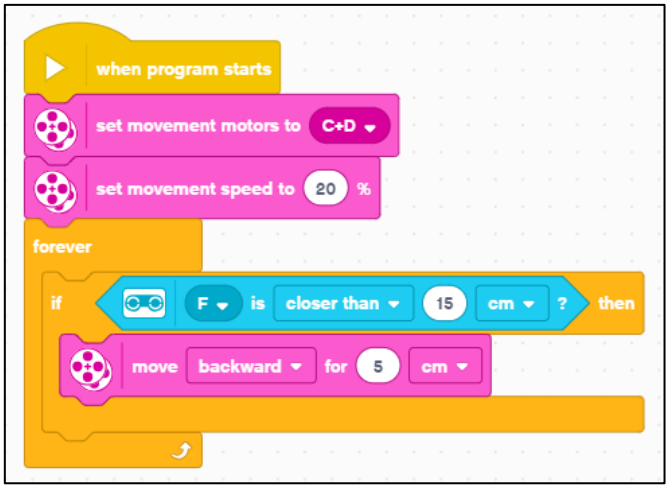
4. Példa – Menekülő robot

Írjon olyan programot, amelyet végrehajtva a robot tolat 5 cm-t, ha 15 cm belül észlel maga előtt valamilyen akadályt! Ha nincs ilyen akadály, akkor álljon egyhelyben.

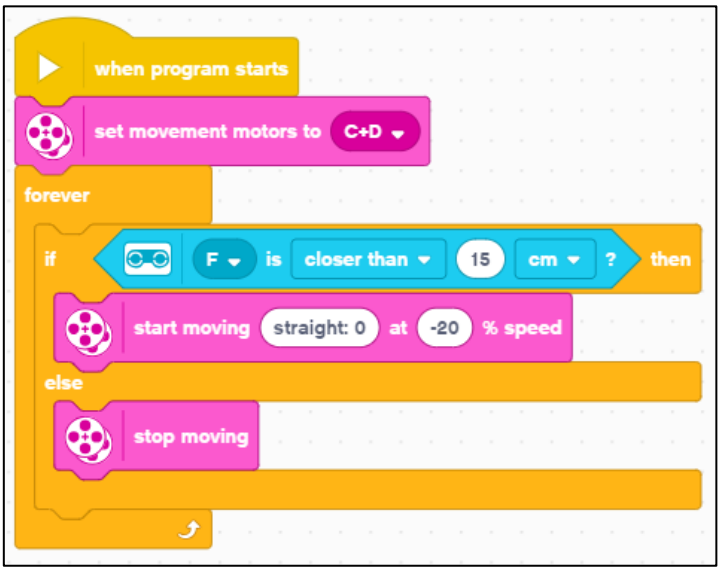
Megoldás

A feladatspecifikáció szerint csak akkor kell mozognia a robotnak, ha 15 cm-en belül akadályt lát. Így egy egyszálú elágazást lehet használni.

A feltételt az előző feladatnál már látott módon állítjuk be. Az utasítás, amit a feltétel igaz állapota esetén végre kall hajtani, egy 5 cm-es tolatás. Ha ebben a formában illesztenénk a kódot a programszálra, akkor csak egyszer hajtaná végre a robot. Ha 15 cm-en belül állna egy akadály előtt a programindítás pillanatában, akkor tolatna és utána véget is érne a programja. Ha úgy indítanánk a programot, hogy nincs a robot előtt akadály, akkor nem mozdulna, de mivel a feltételt megvizsgálta, ezért le is állna a programja. Amennyiben azt szeretnénk, hogy a feltételt folyamatosan figyelje és reagáljon, akkor egy ciklusba kell helyezni az elágazást. Ez lehet most egy végtelen ciklus (elvileg csak akkor áll le, ha direkt leállítjuk, vagy lemerül az akkumulátor). Így már működik.



A program futása nem szép, mert a robot nem folyamatosan tolat (menekül), hanem 5 cm-es szakaszokban. Ha azt szeretnénk, hogy a mozgása folyamatos legyen, akkor kétszálú elágazást érdemes használni. Az egyébként (*else*) ágra a *stop moving* blokkot tehetjük, az igaz ágon pedig csak bekapcsoljuk a motorokat.



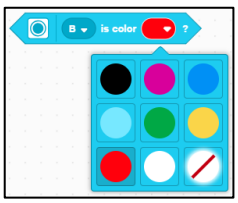
A *start moving* blokk esetén a negatív sebesség a tolatást eredményezi (ellentétes forgásirány a motoroknál).

5. Példa – Útvonalkövetés

Írjon programot, amelyet a robot végrehajtva fényszenzorával követi a fehér alapon lévő fekete színű vonalat!

Megoldás

A feladat bevezetéseként írjunk egy olyan programot, amelynél a robot balra fordul, ha a fényszenzora az indításkor fekete színt érzékel és jobbra egyébként.

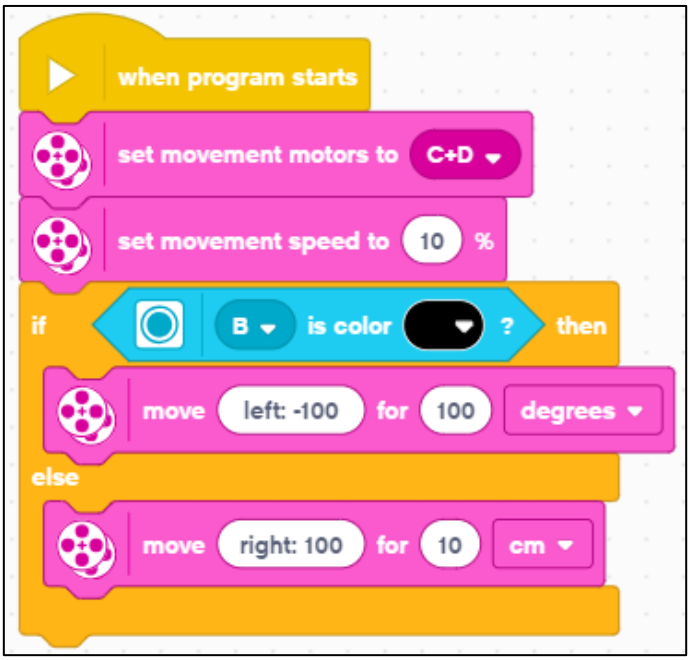


A fényszenzort a programban kétféle üzemmódban tudjuk használni. Színt képes érzékelni vagy a felületről visszavert fény intenzitását. Színérzékelés esetén nyolc különböző színt képes megkülönböztetni (a kilencedik az egyéb szín).

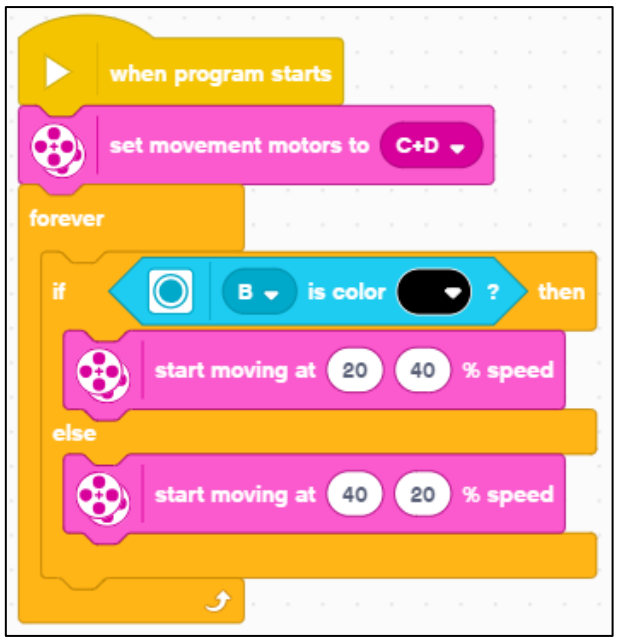
Fény üzemmódban egy 0-100 közötti értéket ad eredményül, a felület színétől, fényességétől függően. A sötét matt felületeknél alacsonyabb az érték.

A program elkészítéséhez a motorok sebességét állítsuk kicsire (pl. 10%)! Az elágazás vezérléséhez most a színszenzor logikai értéket visszaadó blokkját használjuk. A forráskódot az ábra szemlélteti.

A robot összesen 100 fokot fordít a kerekein. Az irány attól függ, hogy az indításnál fekete vagy nem fekete színt érzékelt-e a szenzora.



Ha a programunkat úgy alakítjuk át, hogy ne csak egyszer hajtsa végre az elágazás kiértékelését, hanem folyamatosan és ennek megfelelően reagáljon jobbra vagy balra fordulással, akkor egy olyan általánosan használható programhoz jutunk, amely képes követni egy fekete vonal jobb vagy bal szélét, kígyózó mozgással. Javítanunk kell még a programon annyit, hogy a forgások ne konstans értékűek legyenek, hanem csak különböző sebességgel forgassuk a motorokat. Ha konstanst állítunk be fordulási szögnek, akkor addig nem vizsgálja újra az elágazás feltételét, míg le nem forogta a beállított szöget.
Ciklusként használhatunk végtelen ciklust.



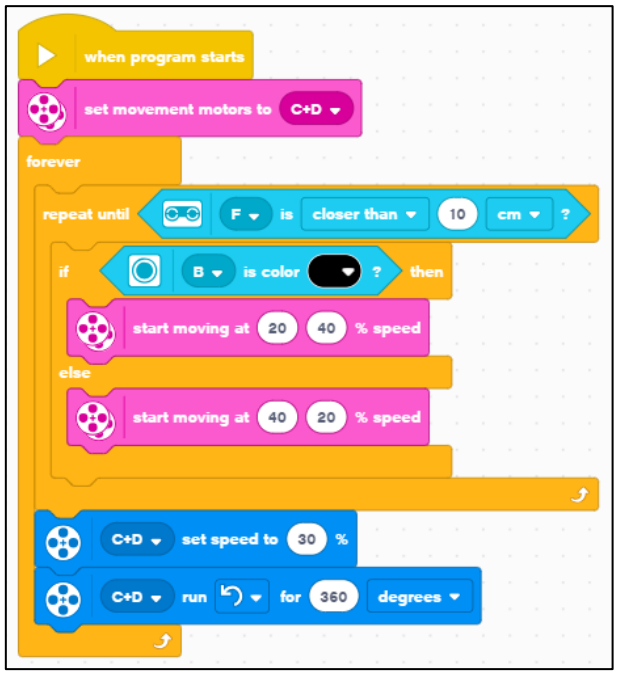
Érdemes a robot mozgásán megfigyelni az algoritmus működését. A jelenlegi programmal az útvonal menetirány szerinti bal szélét követi a robot, hiszen fekete szín érzékelésénél a jobb oldali motor forog gyorsabban, tehát balra fordul. Ha a szenzora lekerült a fekete felületről, akkor pedig jobbra fordul.

Ha az útvonal másik oldalát szeretnénk követni, akkor fel kell cserélni az elágazás két ágát, vagy meg kell fordítani a motorok sebességének előjelét.

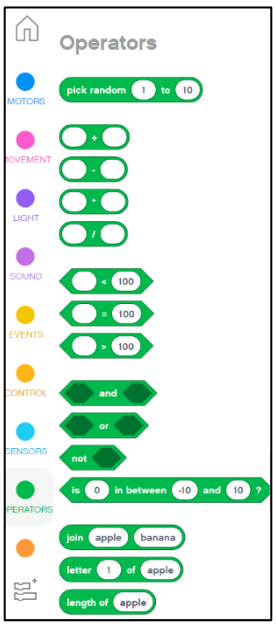
Finomítani lehet az útvonalkövetés pontosságát, ha a motorok sebeségei közötti különbséget változtatjuk. A robot reakcióideje miatt, ha kicsi a sebességkülönbség, akkor nagy ívű lesz a fordulás és könnyen átkerül a szenzor az út rossz oldalára és elveszíti az útvonalat (megfordul). Tehát éles kanyarokat vagy keskeny útvonalat lassú sebességekkel és nagyobb sebességkülönbséggel érdemes követni. Mindig az adott útvonal dönti el a beállítandó értékeket.

Ha a végtelen ciklus helyett például a távolságérzékelőt használjuk az útvonalkövetés befejezésének érzékelésére, akkor komplex pályákat is be lehet járni a robottal. Például akadályig kövesse az útvonalat, majd ott forduljon meg és induljon el visszafelé. …

Az alábbi program két akadály között útvonalkövetéssel mozgatja a robotot amíg le nem állítjuk a programot. Akkor működik helyesen a program, ha a színszenzor kb. a robot hosszanti tengelyén van. Az alaprobotnál a jobb oldalon található, így az akadály észlelésekor, a 180 fokos fordulásnál a szenzor messze kerül az útvonaltól és nem fog rátalálni a visszafelé követésnél. Ezt úgy korrigálhatjuk, hogy az akadály elérésekor nem 180 fokos fordulót állítunk be, hanem kisebbet vagy nagyobbat a fényszenzor helyzetétől függően. A lényeg, hogy a fordulás után a fényszenzor az útvonal „jó” oldalán álljon, így, ha ismét elindul az útvonalkövetési ciklus, akkor automatikusan rá fog állni az útra.



Operátorok (OPERATORS programcsoport)



A programozásban operátoroknak nevezzük azokat az elemeket, amelyek valamilyen műveletet végeznek el egy vagy két operandus, elem között. Ezek lehetnek például matematikai műveletek, amelyek összeadnak, kivonnak, szoroznak, osztanak, vagy maradékosan osztanak számokat és az eredményt adják vissza értékül. Lehetnek logikai értéket visszaadó műveletek, például a relációs operátorok (<, >, =), vagy logikai műveletek (és-and, vagy-or, tagadás-not). Ide tartozhatnak a karaktersorozatokat (string) összefűző, vagy daraboló műveletet végzők, de a jelen programkörnyezetben a véletlen számokat előállító műveletet is az operátorok kategóriába sorolták. A szoftverben zöld szín jelöli a csoportot. Ezek a blokkok önállóan nem használhatók utasításként, de az egyes blokkoknál felhasználhatók a feltételek, bemeneti paraméterek megadására, számítására, átalakítására.

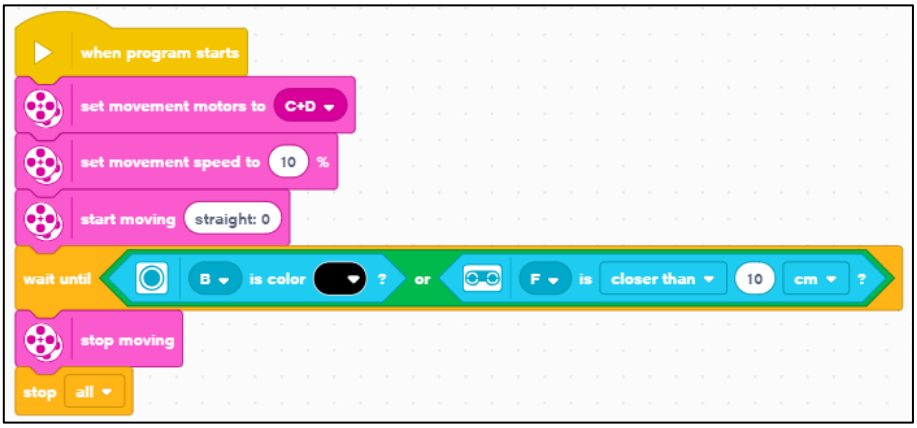
A végzett művelettől függően logikai vagy szám típusú értéket adnak vissza. Használatukkal például az elágazások vagy ciklusok feltételei sokkal komplexebbé tehető.

6. példa – Összetett feltételű mozgásvezérlés

Írjon olyan programot, amelynek végrehajtása során a robot állandó sebességgel halad egyenesen előre és megáll, ha fekete színű felületet érzékel a színszenzora vagy 10 cm-nél közelebb került egy akadályhoz! Amelyik esemény előbb következik be, az állítsa meg a robot mozgását!

Megoldás

Talán a legegyszerűbb megoldás, ha a kezdeti értékadások után elindítjuk a robotot és egy *wait until* szerkezettel várunk egy esemény bekövetkezésére. Az esemény most két feltételből áll, amelyek között vagylagos a kapcsolat: vagy fekete színt érzékel, vagy akadályt.



Az esemény bekövetkezte után leállítjuk a mozgást. (A program végén a stop utasítás el is hagyható.)

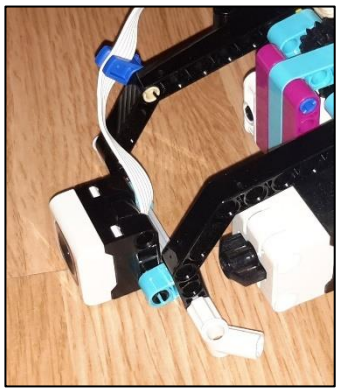
7. példa – Színválogató

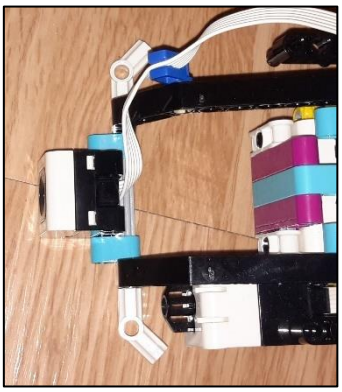
Írjon programot, amelyet végrehajtva a robot egyenesen előre mozog mindaddig, amíg egy színes tárgyat nem érzékel (pl.: LEGO kockát)! A tárgy színétől függően végezze a további tevékenységét! Ha

a tárgy piros vagy sárga, akkor az elejére szerelt fogókar segítségével „fogja” meg és vonszolja hátra 30 cm-t, egyébként ne csináljon semmit, tehát hagyja abba a működését!

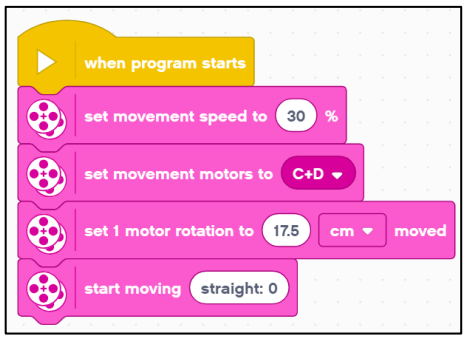
Megoldás

A programírás előtt egy kicsit át kell alakítani a robotot. A feladat megoldása szempontjából az lenne az ideális, ha a fényszenzor a fogókar közepén helyezkedne el és mindenképpen előre néző pozícióban. Az alábbi képek mutatják a szükséges szerkezeti módosításokat.



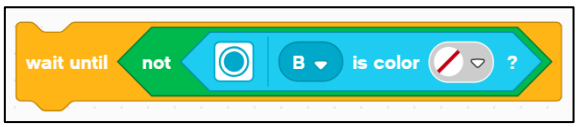


A programot a szokásos mozgásra vonatkozó paraméterek beállításával kezdjük.

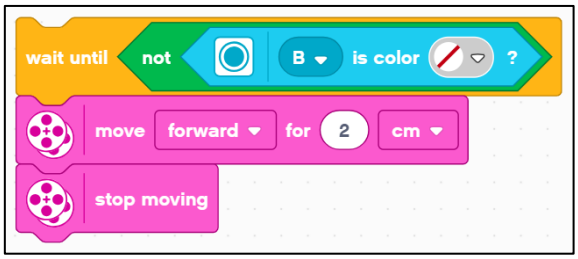


A mozgást mindaddig kell folytatnia, amíg be nem következik az esemény (érzékel egy színes tárgyat). Ezt a *wait until* szerkezettel vizsgálhatjuk.

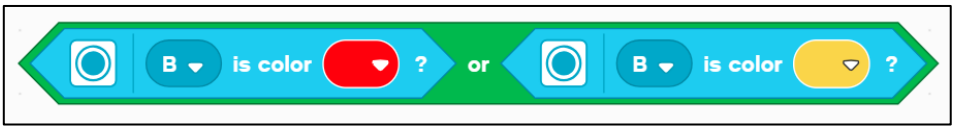
Feltételként a fényszenzornak azt az értékét választjuk, amely az egyéb szín érzékelését jelenti és ezt egy tagadás műveletet végző operátoron keresztül kapcsoljuk a ciklushoz.



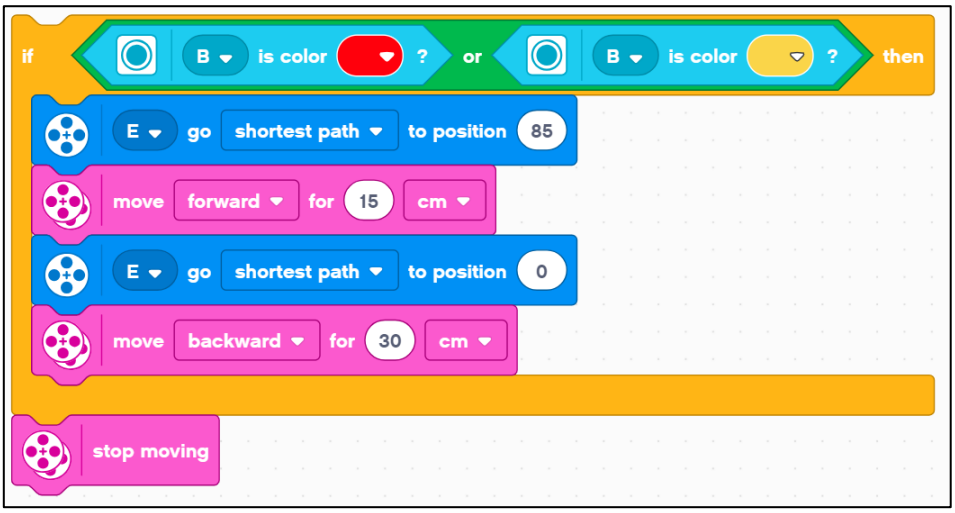
Tehát akkor folytatódik a programszál, ha a színszenzor a robot számára is értelmezhető színű tárgyat észlel (nem egyéb színű a tárgy). Mivel a robot a színérzékelésre már 7-8 cm távolságból is képes, de ilyenkor még bizonytalanabb a színfelismerése, ezért mielőtt megállítanánk 2 cm-t még mozogjon előre.



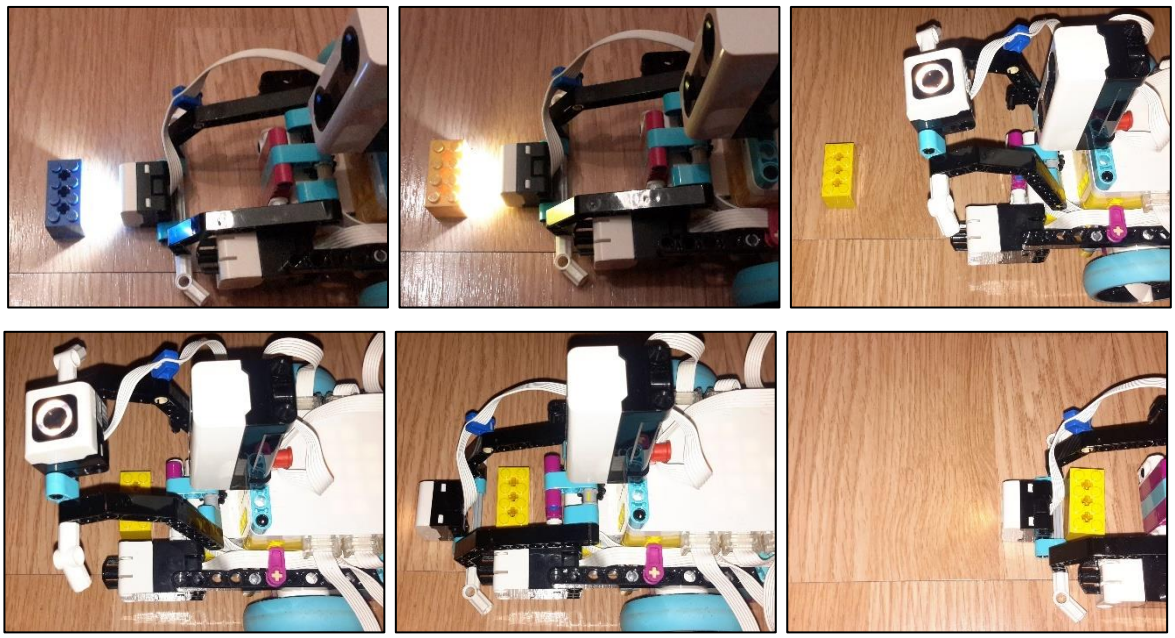
Elvileg most már elég közel van az akadályhoz, hogy helyesen ismerje fel a színét. A feladatspecifikáció szerinti két feltételt (piros vagy sárga tárgyat lát) egy vagy típusú operátorral kódolhatjuk, használva a színszenzor pillanatnyi értékét lekérdező blokkokat.



Ha ennek a feltételnek a logikai értéke igaz, akkor fel kell emelnie a fogókarját, előre kell mozognia (pl.: 15 cm-t), le kell engednie a fogókarját és tolatnia kell 30 centimétert. A fogókarok adott pozícióba mozgatásához használjuk a korábban említett MOTORS csoporton belüli, adott pozícióba mozgató blokkot. Így a kar felemelése és visszaengedése pontosan elvégezhető, akár többször egymás után is.



A részletekben bemutatott kódokat összekapcsolva megkapjuk a teljes programot hiszen, ha az érzékelt szín nem piros és nem sárga, akkor a robotnak nem kell semmit csinálnia, tehát a programja véget érhet. A robot mozgását viselkedését mutatja be a következő képsorozat.



Eseménykezelés (EVENTS programcsoport)

A hagyományos procedurális programozási környezetek esetén megszokott, hogy a programok egy szálon futnak. Ez azt jelenti, hogy egy lineáris utasítás-végrehajtási rend érvényes, az első utasítástól kezdve sorban az utolsóig. A lineáris programszerkezetből következően programjaink egy utasítást csak akkor fognak végrehajtani, ha az előző utasítások már lefutottak. Erre láttunk példát a motorok mozgatását bemutató fejezetben.

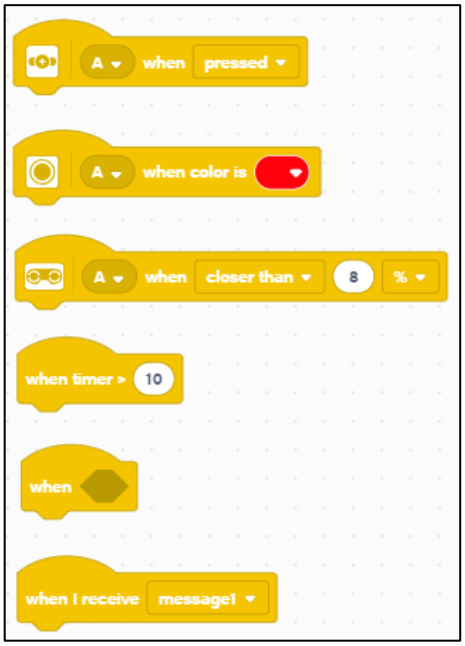
A mindennapi életben viszont az emberi érzékszervek egyszerre több szálon is képesek figyelni a környezet eseményeire. Szem, fül …

A bonyolultabb programozási rendszerekben ez a többszálúság (*multitask*) már alapértelmezett funkció, hiszen a számítógép operációs rendszere lehetővé teszi, hogy látszólag egyszerre több programot is futtassunk. Pl. internetezés közben zenét is hallgathatunk a számítógépen egy médialejátszó program segítségével.

Erre a többszálú programozásra nyújt lehetőséget, amelyre robotprogramozásnál nagy szükség van. A robot szenzoraira sok esetben egymással párhuzamosan kell figyelni és a motorokkal történő mozgás mellett esetleg egy speciális szenzorérték esetén valamilyen eseményt kezdeményezni.

Az eseményvezérelt programozásnál egy külön szálon figyelhetjük a robot szenzorait és a beállított esemény bekövetkezésekor automatikusan elindul az a parancssor, amelyet az eseményhez rendeltünk.

A szoftver EVENTS programcsoportjában találhatók az eseményindító blokkok. Maga főprogram is egy eseményindítóval kezdődik: „*when program starts*”. A robothoz tartozó három szenzorhoz (szín-, nyomás-, távolságszenzor) is tartoznak eseményindítók. Ezeken kívül köthető a párhuzamos szál indítása időhöz, összetett feltételhez és üzenethez is.



A párhuzamos szálú programozásnál a két szálon elhelyezett utasításokat látszólag egyszerre (egyidőben) hajtja végre a robot, de valójában gyors váltások történnek a két szál utasításainak végrehajtása között.

A párhuzamos szál csak akkor lép életbe, ha az indító feltétel teljesül.

Az 1. példánál már láttuk, hogy a robotot könnyű egy négyzet (sokszög) alakú pályán mozgatni. Alakítsuk át a programot és most azt szeretnénk, ha a robot egy négyzet alakú pályán mozogna folyamatosan és az ütközésérzékelő megnyomására megállna.

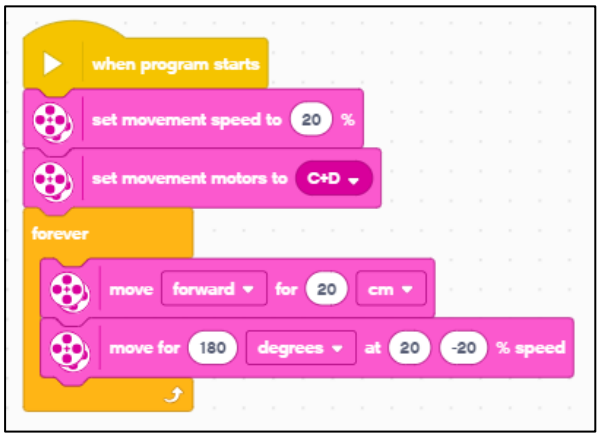
8. példa – Mozgásvezérlés párhuzamos szálú eseményvezérléssel

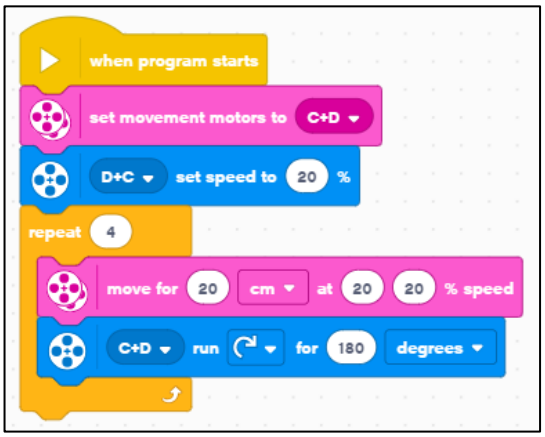
Írjon programot, amelyet végrehajtva a robot egy négyzet alakú pályán mozog és ha megnyomják az ütközésérzékelőjét, akkor azonnal megáll!

Megoldás

Az eredeti program látható a jobb oldali képen.

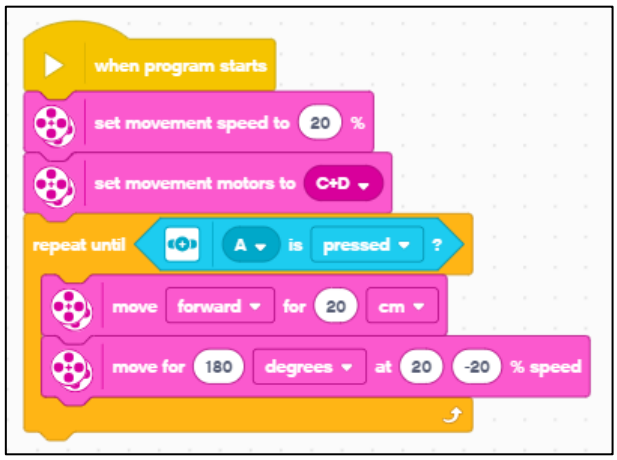
A program egy alternatív megoldása, amely hatására a robot ugyanúgy viselkedik szerepel a bal oldali képen.





A két kód között annyi a különbség, hogy a bal oldali programnál a robot nem áll meg a négyzet bejárása után, hanem folyamatosan mozog kikapcsolásig, egy négyzet alakú pályán.

Az első gondolat a példa feladatának megoldására, hogy cseréljük le a végtelen ciklust egy előltesztelő ciklusra, ahol a feltétel az ütközésérzékelő figyelése.

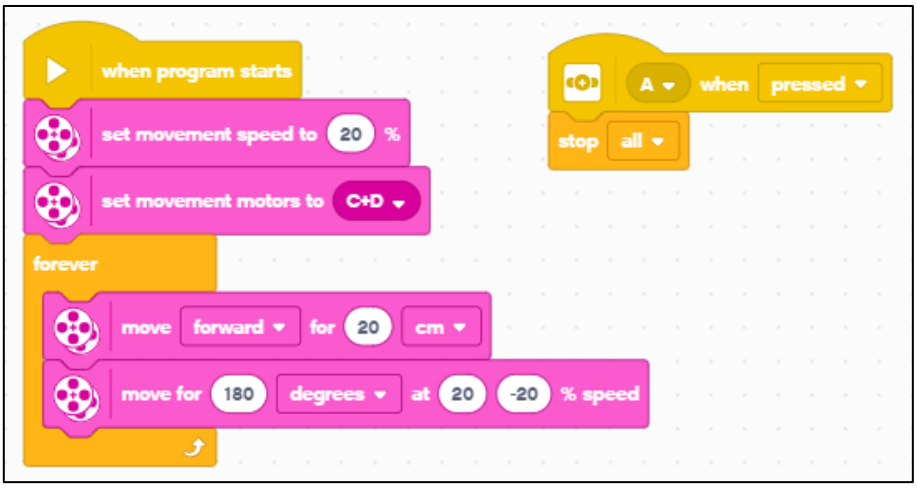


A programot letesztelve azt tapasztaljuk, hogy nem működik helyesen, a robot nem áll meg az ütközésérzékelő megnyomására. Ha türelmesek vagyunk és az ütközésérzékelőt folyamatosan nyomva tartjuk, akkor egy idő után viszont megáll.

A viselkedés oka, hogy a ciklus feltételét akkor vizsgálja a robot, amikor végrehajtotta a ciklusmagban szereplő két utasítást. Tehát lemozogta a 20 centimétert és elfordult kb. 90 fokkal (180 fokos tengelyfordulat). Ha éppen abban a pillanatban nyomjuk meg az ütközésérzékelőt, akkor megáll.

Korábban is említettük, hogy ha a mozgást olyan blokkal szabályozzuk, amelynél beállítottunk valamilyen mozgás időtartamára vonatkozó feltételt (pl.: 20 cm-es előrehaladás vagy 180 fokos tengelyfordulat), akkor addig, amíg ezek a tevékenységek be nem fejeződtek a program végrehajtása az adott utasításnál áll (nemlép túl rajta a vezérlés).

A megoldás a párhuzamos szál indítása és az ütközésérzékelőn bekövetkezett esemény figyelése lehet.



A második szálon egyetlen utasítás szerepel: „*stop all*”. Ennek hatására a robot meg fog állni, mert leállítjuk a végtelen ciklust is. Annyi késleltetés lesz a dologban, hogy még befejezi az éppen futtatott parancs végrehajtását és csak utána áll meg.

Az eseményvezérelt programindítására nézzünk egy másik feladatot. A 3. példánál korábban bemutatott adaptív sebesség radar programot alakítsuk át!

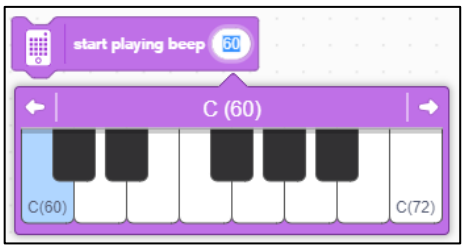
9. példa – Adaptív sebesség radar hangjelzéssel

Írjon programot, amelyt a robot végrehajtva egy akadály felé közeledik folyamatosan csökkenő sebességgel! A sebessége az akadálytól mért távolságtól függően csökkenjen! Az akadálytól 10 cm távolságban álljon meg, de 20 cm-en belüli távolságnál már adjon hangjelzést! Ha a robot megállt, akkor a hangjelzés is álljon le!

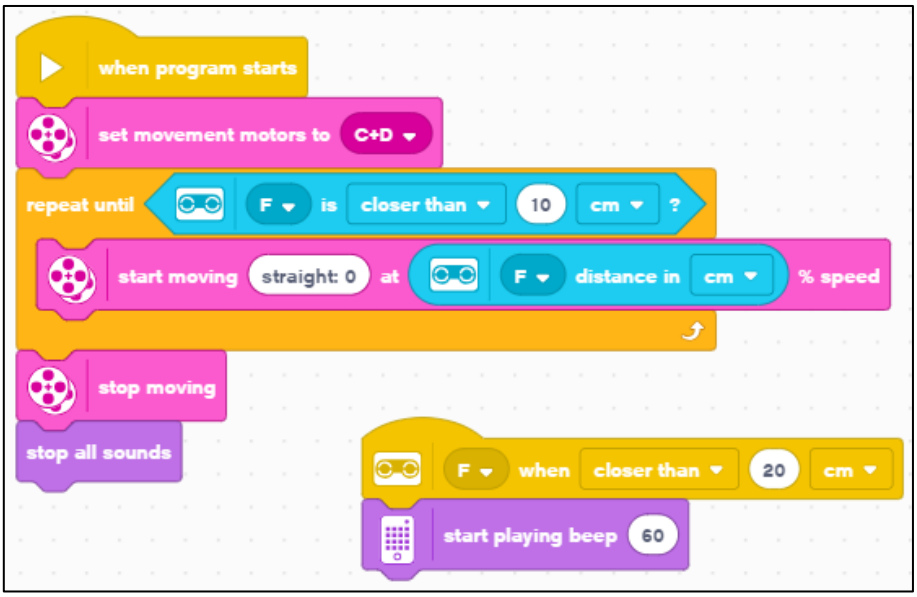
Megoldás

A 3. példánál bemutatott kódot vegyük alapul és egészítsük ki egy eseményindítóval. Ha a robot távolság szenzora 20 centiméteren belül akadályt észlel, akkor indítsunk el egy új programszálat, ami egy hangjelzést generál. Amikor a robot megállt, akkor állítsuk le a hangjelzést!

A programkörnyezet SOUND (lila színű) programcsoportjában találhatók a hangkezelő blokkok. Itt be lehet állítani különböző hangeffektek lejátszását (a szoftverben több ilyen is van), de egyszerű hangot is le tudunk játszani, amelyet egy zongora stilizált billentyűzetéről választhatunk ki.



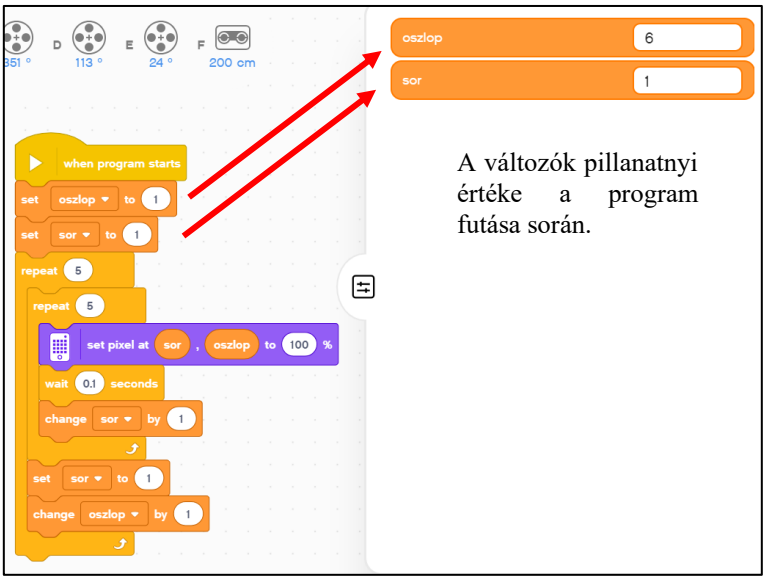
A feladat megoldásához tehát indítunk a távolságérzékelő mért értékére alapozott eseményszálat, egy hang lejátszásával. A fő programszál végére pedig beillesztjük a hanglejátszás megállítását. Az eseményszálra helyezett utasítás úgy viselkedik, mintha egy végtelen ciklus része lenne. Amíg az eseményszál feltétele igaz (tehát a mért távolság kisebb, mint 20 cm), addig folyamatosan ismétli a kiválasztott hangot. Ezért kell leállítani a fő programszál utolsó utasításával.



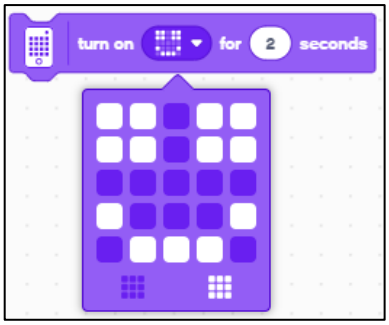
Képernyőkezelés (LIGHT programcsoport) és változók használata (VARIABLES programcsoport)

Minden magas szintű programnyelv lényeges elemei az úgynevezett változók. A változók alkalmasak arra, hogy adatokat tároljunk bennük. A memóriának egy kitüntetett területe a változó, aminek saját nevet adhatunk. Erre a memóriaterületre elhelyezhetünk adatokat és bármikor elővehetjük onnan őket az adott változó névvel hivatkozva rájuk. Úgy tudunk tehát programozni, hogy nem is kell ismernünk ezeket az adatokat (elegendő a nevüket megadni), hiszen elég, ha a programban szereplő utasítások „ismerik” az értékeket.

A programkörnyezetben van lehetőség változók létrehozására. A változóink számokat tárolhatnak. A VARIABLES programcsoport „*Make a variables*” feliratú gombjára kattintva meg tudjuk adni a változó nevét. Az elkészített változóink megjelennek a programcsoport listájában egy-egy jelölőnégyzettel. A változót használó program futása során a képernyőn meg tudjuk figyelni az értékeik változását. Ez egyfajta nyomkövetésre ad lehetőséget, hiszen az értékeket figyelve tudunk következtetni a programunk belső működésére vagy esetleges hibáira. Azok a változók jelennek meg a megfigyelési listában, amelyek előtt bekapcsoltuk a jelölőnégyzetet. A megfigyelési ablak a képernyő jobb szélének közepén található szimbólumra kattintva hívható elő, illetve rejthető el ( ).



A változók használatát egy példa segítségével mutatjuk be. Ehhez a robot vezérlő egységének képernyőjét használjuk. A képernyő egy 5x5-ös led mátrix megjelenítő, ahol a 25 db ledet tudjuk ki-be kapcsolni, illetve a fényerőt szabályozni. Egyszerű képeket, ábrákat tudunk így megjeleníteni pl.: egy-egy betű vagy számjegy képét. A rendelkezésre álló blokkok a LIGHT programcsoportban találhatók. Van a blokkok között olyan, amely segítségével egy ábrát tudunk grafikusan összeállítani, amelyet meg tudunk jeleníteni a kijelzőn. Programozási szempontból talán a leghasználhatóbb az a blokk, ahol a képernyőkoordináták megadásával lehet egyetlen ledet ki-be kapcsolni.

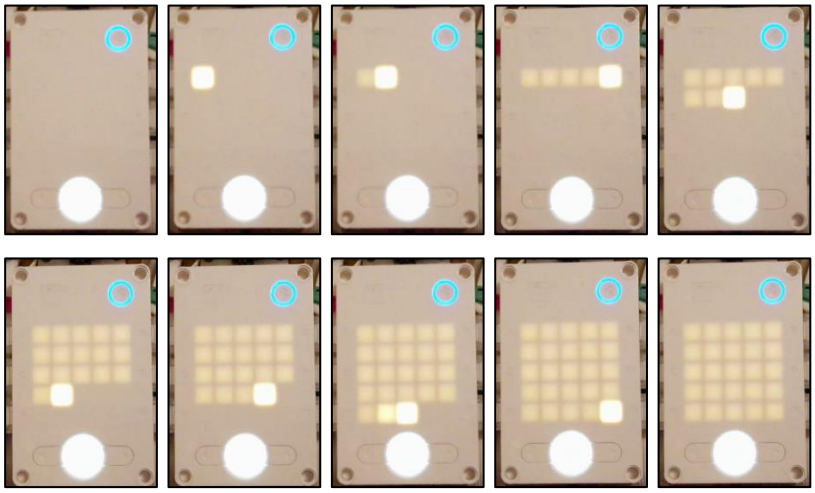


Természetesen a koordináták helyére, vagy fényerő paraméterének helyére lehet szám típusú értéket visszadó blokkot is tenni.

10. példa – Futófény a képernyőn

Írjon programot, amelyt végrehajtva a robot a vezérlőegységének képernyőjén egy futófényt jelenít meg! Az egyetlen négyzetből álló fény a bal felső saroktól induljon, 0.1 másodperces várakozás után haladjon balról-jobbra és fentről lefelé a 25. pixelig! A led megjelenésekor teljes fényerővel világítson, de mielőtt a soron következő led felvillan halványuljon el 40%-ra! Kezdetben teljesen sötét képernyővel induljon a program és a befejezése után minden led 40%-osan világítson!

Az alábbi fotósorozat értelmezi a feladatot.

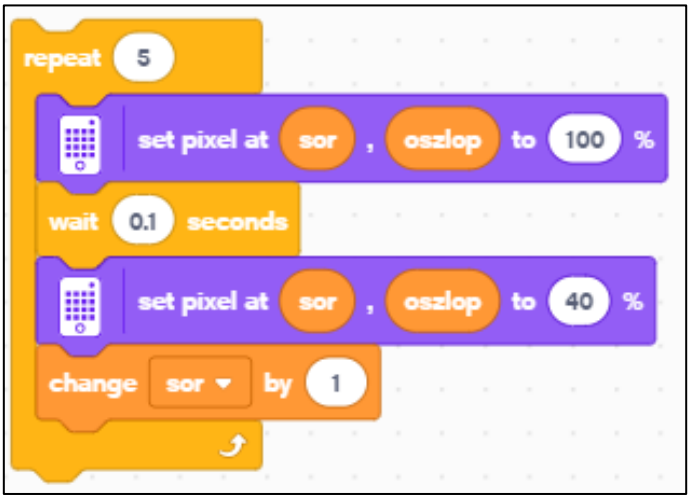


Megoldás

A képernyőn szereplő 25 pont bejárásához a pontok vízszintes és függőleges pozíciójának folyamatos változtatása szükséges. Készítsünk két változót *sor* és *oszlop* néven, amelyben majd mindig az aktuális sor és oszlop koordinátát tároljuk, amelyik pontnak éppen 100%-kon kell felvillannia. Mindkét változó kezdőértéke legyen 1, mivel a ledek indexelése a bal felső led esetén (1; 1), a jobb alsó-é pedig (5; 5).

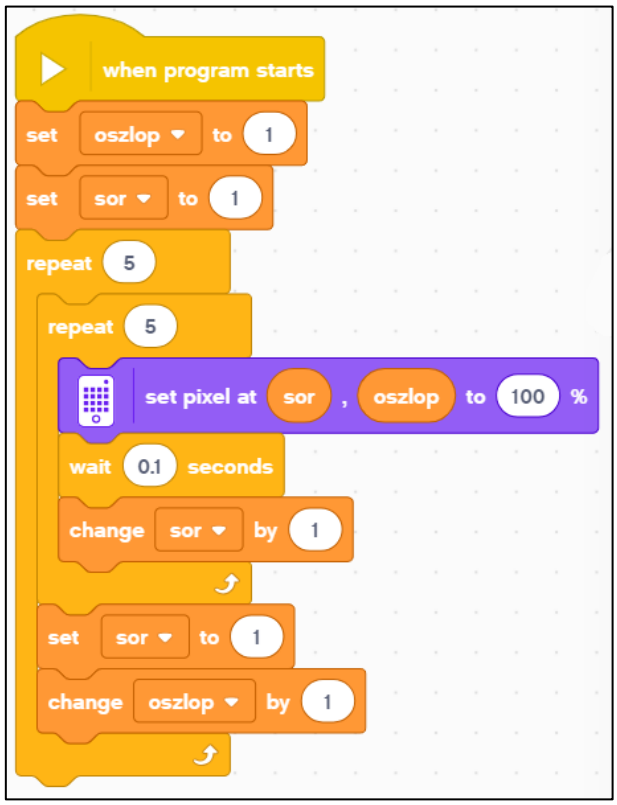
A két változót felhasználva a felvillanó led pozícióját a következő paraméterezett blokkal tudjuk megadni.

Elindítunk egy 5-ször lefutó ciklust, amely egy soron fog végig menni. Az adott pozíciójú led felvillantása után várakozik 0.1 másodpercet, majd 40%-osra állítja a fényerőt és növeli eggyel a *sor* változó értékét.



A ciklus lefutásának hatására a futófény egy soron tud végig menni. Miután ötször végrehajtotta a robot a ciklusmagot, eggyel meg kell növelni az *oszlop* változó értékét (következő sorra lépünk), viszont vissza kell állítani 1-re a *sor* változó értékét, hogy ismét az első leddel kezdje a felvillantást a soron belül.

Mindezt 5-ször kell megismételni a képernyő teljes bejárásához. A teljes program kódja szerepel az alábbi ábrán.



ZÁRSZÓ

Ennek a rövid jegyzetnek a megírásánál nem törekedtünk teljességre. A célunk a kezdő lépések megkönnyítése volt. A bemutatott 10 példát próbáltuk úgy összeállítani, hogy minden fontosabb programozási elemre szerepeljen egy használatot bemutató lehetőség. Az eszközhöz tartozó szoftverben nagyon sok kidolgozott projekt szerepel, amelyhez a robotkonstrukciók megépítése mellett forráskódok is tartoznak. Ezek tanulmányozása sokat segít a profi programozóvá válásban.

A bemutatott példákra más megoldások is adhatók, néha egyszerűbbek, rövidebbek, kreatívabbak. Az eszköz tulajdonságainak megértésénél viszont ezek a példák segíthetnek leginkább.