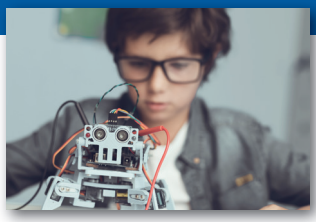
Algoritmizálás, programozás, robotika



Bevezető

Korábbi tanulmányainkban megismerkedtünk a robot fogalmával és a robotok különböző generációival. Algoritmusokat írtunk, amelyek alapján programokat készítettünk blokkprogramozási környezetekben. Az információkat adatként tároltuk el, az adatokkal pedig műveleteket végeztünk. Láttuk, hogy az elvégzendő feladatokat folyamatokkal lehet megoldani, amelyek kisebb részfeladatokból állnak. A folyamatokat folyamatábrán ábrázoltuk.

A robotok viselkedését számítógépes környezetben szimuláltuk. Kezdetben egy virtuális robotot irányítottunk a billentyűzettel, majd később utasítások segítségével vezettük ki a labirintusból. Megvizsgáltuk azt is, hogy a robot hogyan érzékelheti a környezetét. Használtunk szenzorokat (érzékelőket). Készítettünk olyan programot, amely lehetővé tette, hogy a robot önállóan haladjon előre, amíg el nem jut a falig. A feladatok során megismerkedtünk az utasítás, szekvencia, ciklus, változó, elágazás és paraméter fogalmával. Láttuk azt is, hogy paraméterként több adattípus is használható, például megadhattunk számot, vagy akár szöveget is.

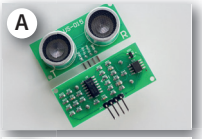
Megismerkedtünk a micro:bit eszközzel, amelyhez érdekes alkalmazásokat és játékokat készítettünk. A következőkben elmélyítjük tudásunkat, és új lehetőségekkel is kiegészítjük. Sőt, betekintést kapunk valódi robotok programozásába is.

Ismétlés

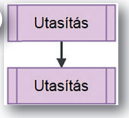
Párosítsuk az alábbi fogalmakat a képekkel!

1. ipari robot
2. számlálós ciklus
3. szenzor
4. változó
5. végtelen ciklus
6. feltételes ciklus
7. kétirányú elágazás
8. szekvencia
9. micro:bit
10. egyszerű elágazás
11. szöveg és szám típusú paraméterek
12. események

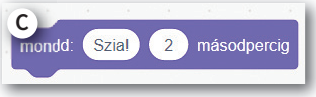
A



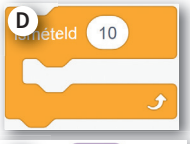
B



C

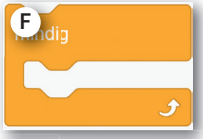


D

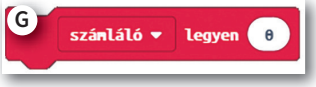


E

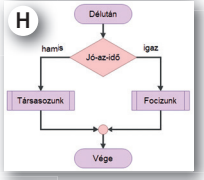
F



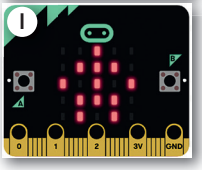
G



H



I

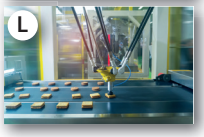


J

K

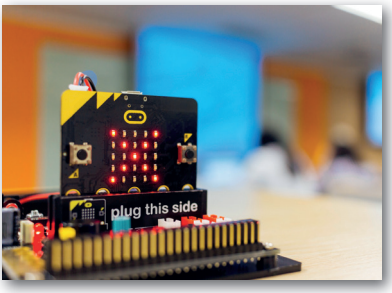
```
Ha feltétel akkor
    utasítás(ok)
elágazás vége
```

L



Programozzunk micro:biteket!

A micro:bit eszközt többféle blokkprogramozási környezetben programozhatjuk (pl. Scratch, Makecode stb.). Nyissuk meg az általunk használt alkalmazást! Ismétlésként készítünk egy, a korábban megvalósított kő, papír, olló játékhoz hasonló alkalmazást.



Gyűjtsünk pontokat!

A pályán az X jelzi a kiindulási helyzetünket. Az egyes négyzetekben különböző pontszámokat láthatunk. Készítsünk olyan programot, amely véletlenszerűen meghatározott útvonalon vezet végig minket a pályán!

A micro:bit megrázásakor törlődjön le a kijelző, majd egy másodperc múlva véletlenszerűen jelenjen meg egy felfelé, jobbra, lefelé vagy balra mutató nyíl!

Rázzuk meg az eszközt, és haladjunk a micro:bit által jelzett irányba egy lépést, de csak akkor, ha még nem értük el a falat! Számoljuk össze, hogy tíz rázás után hány pontot gyűjtöttünk össze! Hasonlítsuk össze a pontjainkat! Ki volt a legszerencsésebb az osztályban? Kísérletezzünk! Hányszor kell megrázni az eszközt ahhoz, hogy olyan mezőre lépjünk, amely 100-as pontszámot rejt?

	40			50		
						100
100		50		20		
			X			50
	20				20	
	100		20			40

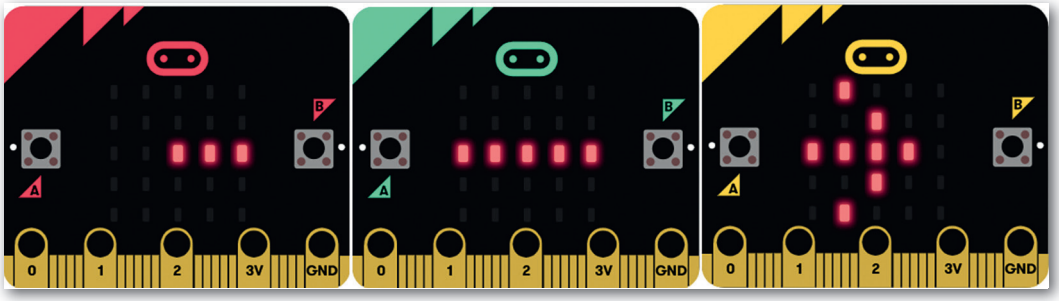
► Pontgyűjtő játék táblája

Készítsünk animációt!

Készítsünk olyan animációt, amely valamelyik kedvenc sportágunkkal kapcsolatos! Ennek során használhatunk ikonokat, számokat és szövegeket is!

Animáció több kijelzőn

Alakítsunk háromfős csoportokat! Készítsünk olyan animációt, amelyhez mindhárom micro:bitre szükség lesz! Az animáció legalább 10 másodperc hosszú legyen! Az animáció induljon el az egyik micro:biten, majd folytatódjon a másikon! Egy ilyen animáció lehet például az, hogy egy nyílvessző balról jobbra átrepül a három micro:bit kijelzőjén.

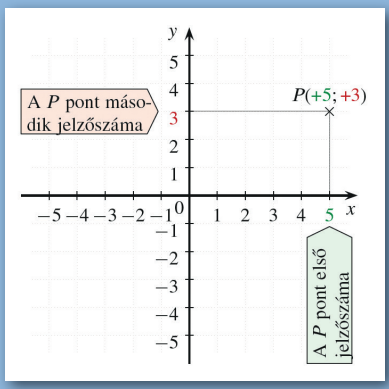


► Az animáció egy részlete három egymás melletti micro:biten

Rajzoljunk a kijelzőre!

A micro:bit kijelzőjét alkotó LED-ek öt sorban és öt oszlopban vannak elrendezve. Korábban csak ikonokat jelenítettünk meg ezen a kijelzőn. Hogyan tudnánk egyenként felkapcsolni, illetve lekapcsolni a LED-eket?

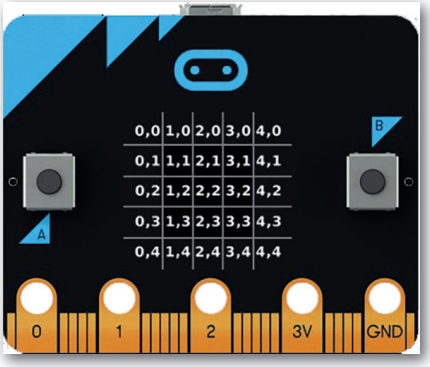
Matematikaórán már tanultunk a derékszögű koordináta-rendszerről, amelyet két, egymásra merőleges számegyenes alkot. Tudjuk, hogy a számegyeneseket x tengelynek, illetve y tengelynek nevezik. A tengelyek közös pontja az origó. Az egyes pontok helyzetét a koordinátáikkal adhatjuk meg. A koordináták sorrendje nem cserélhető fel, először az x tengelyre vonatkozó jelzőszámot, majd az y tengelyre vonatkozó jelzőszámot kell megadnunk. A két jelzőszámot kerek zárójelek közé kell tennünk, és pontosvesszővel kell egymástól elválasztanunk.



A micro:bit LED kijelzőjének pontjait is egy koordináta-rendszer alapján tudjuk azonosítani. Ez a koordináta-rendszer viszont speciális. Itt a bal felső sarok az origó. A pontok jelzőszámai pedig csak a 0, 1, 2, 3 és 4 értékek lehetnek.

Az *x* tengely értékei jobbra nőnek, az *y* tengely értékei pedig lefelé. Az *x* tengely tehát az egyes oszlopokat, az *y* tengely a sorokat jelenti.

A bal felső sarok koordinátája a (0;0), a jobb felső saroké (4;0), a középső ponté (2;2) és így tovább.



► A micro:bit LED-ek koordinátái

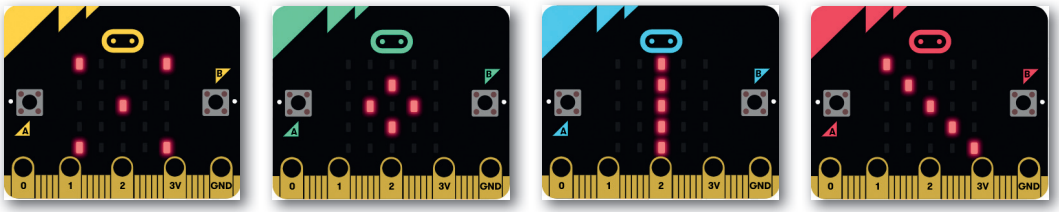
A micro:bit kijelzőjén lévő pontokat felkapcsolhatjuk, lekapcsolhatjuk, illetve átválthatjuk az ellenkező állapotukba. Ez azt jelenti, hogy ha a LED fel volt kapcsolva, akkor lekapcsolódik, ha le volt kapcsolva, akkor pedig felkapcsolódik.

A LED állapotát lekérdezhetjük az *x* és *y* koordináta megadásával. A *„led állapota x y”* blokk *igaz* értéket ad vissza akkor, ha fel van kapcsolva az adott koordinátájú pont, és *hamisat*, ha nem.

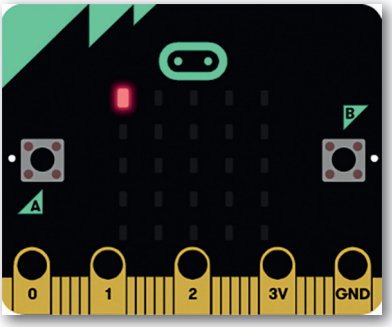
► A LED kategória blokkjai a MakeCode környezetben

Feladatok

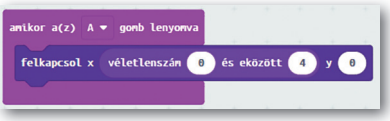
1. Állítsuk elő az alábbi ábrákat! A LED-eket a koordinátájuk alapján kapcsoljuk fel!



2. Készítsünk olyan animációt, amelyben a bal felső sarokban felkapcsolódik a LED, majd jobbra halad, amíg a jobb felső sarokba nem ér!
   A pont eltűnése és a szomszédos helyen való megjelenése között egy másodperc teljen el!
   Hogyan kell módosítani a kódot ahhoz, hogy jobbról balra történjen a mozgás? Mi történik akkor, ha felcseréljük a koordinátákat?

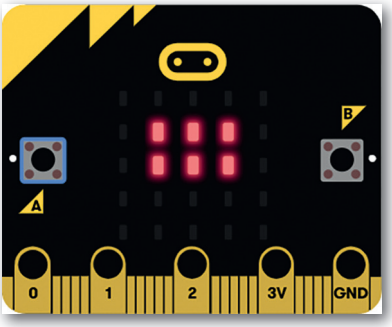


3. Gondoljuk át, hogy mit eredményez az itt látható kód? Mi történik akkor, ha az *y* paraméter értékének a hármas értéket állítjuk be?



4. Készítsünk olyan programot, amely az *A* gomb lenyomásakor a kijelző véletlenszerűen kiválasztott koordinátáján felkapcsol egy LED-et!

5. Változtassuk meg a véletlenszám blokk paramétereit úgy, hogy csak az ábrán jelölt pontok lehessenek felkapcsolva!
   Nyomjuk meg annyiszor a gombot, hogy mind a 6 pont fel legyen kapcsolva!
   Jegyezzük fel, hogy hányszor kellett ehhez lenyomnunk a gombot! Végezzük el többször a kísérletet! Mit tapasztaltunk?
   Beszéljük meg, hogy mi volt a kapott legkisebb, illetve a legnagyobb érték az osztályban a próbálgatások során!



Fontos tapasztalat: Ha véletlenszerűen választunk ki pontokat, akkor előfordulhat, hogy ugyanazt a pontot többször kiválasztjuk.

Szimuláljuk egy parkolóház működését!

Jártál már olyan parkolóházban, ahol a parkolóhelyek felett zöld vagy piros lámpa jelezte, hogy szabad-e a parkoló vagy sem? Ez egy nagyon hasznos segítség az autósok számára, mert már messziről észrevehetik a szabad helyeket.

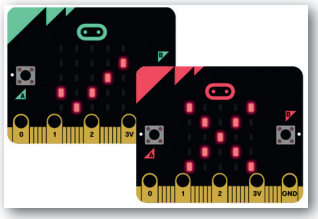
Gondoljuk át, hogy milyen érzékelőkkel lehetne eldönteni, hogy az adott parkolóhelyen áll-e autó vagy sem? (Több helyes megoldás is létezik.)

A következőkben olyan feladatokat oldunk meg a micro:bit segítségével, amelyek az autók parkolásával kapcsolatosak.

Egy áruházhoz szabadtéri parkoló tartozik, amelyben öt sorban és öt oszlopban vannak elrendezve a parkolóhelyek.

Feladatok

1. Készítsünk olyan programot, amely a piros LED-ek felkapcsolásával jelzi, hogy mely helyek foglaltak a parkolóban! Ne felejtsük, hogy a parkolóhelyek öt sorban és öt oszlopban vannak elrendezve!
   Az *A* gomb hatására tizenötször válasszon a program véletlenszerű koordinátát, és kapcsolja fel az ezekhez tartozó LED-eket!



2. Az áruház bejáratához legközelebb eső parkolóhely a mozgáskorlátozott személyek számára van fenntartva. Ennek koordinátája: (0;0). Készítsük el azt a programot, amely a *B* gomb megnyomásakor megjeleníti a ✓ ikont, ha szabad ez a parkoló, és az X ikont, ha foglalt!
   Figyeljünk arra, hogy az ikon megjelenítése előtt töröljük le a kijelzőt!

Keressünk szabad parkolót!

Nem csak a zöld és piros jelzőfény segítheti a parkolást. Használhatnánk mobilalkalmazást a szabad helyek megjelenítésére. Vagy elhelyezhetnénk a parkoló bejáratánál egy gombot, amelyet megnyomva a sofőr láthatná, hogy a parkoló melyik sorának hányadik helyén (oszlopában) van szabad hely.

Hogyan oldhatnánk meg ezt a feladatot az eddigi ismereteink alapján? Gondolkodjunk együtt!

Először válasszunk ki egy helyet véletlenszerűen! Ha ez a hely szabad, akkor jegyezzük fel a koordinátáját, ha nem, akkor válasszunk újra egy helyet véletlenszerűen. Ezt ismételjük addig, míg nem találunk üres helyet!

Játsszuk el az algoritmust!

```
Program
   x:=véletlenszám(0 és 4 között)
   y:=véletlenszám(0 és 4 között)
   ismételd amíg az (x;y) nem szabad!
      x:=véletlenszám(0 és 4 között)
      y:=véletlenszám(0 és 4 között)
   ismétlés vége
   szabad_x:=x
   szabad_y:=y
Program vége
```

Itt láthatjuk a szabad parkolóhely keresésének egyik algoritmusát. Párokban játsszuk végig az algoritmus működését!

Egyikünk rajzoljon fel egy 5 × 5-ös területet a füzetébe tíz szabad hellyel úgy, hogy a másik ne lássa! A másikunk mondjon két véletlenszerűen kiválasztott számot 0 és 4 között. Döntsük el a felrajzolt ábra alapján, hogy az adott helyen szabad-e a parkoló vagy sem. Ismételjük ezt addig, míg a társunk nem mond olyan koordinátákat, ahol szabad hely van! Jegyezzük fel a megtalált üres hely koordinátáit! Hány próbálgatásból sikerült szabad helyet találni?

Beszéljük meg, hogy milyen hátránya van annak, hogy a helyeket véletlenül választjuk ki, nem pedig sorba megyünk végig az egyes helyeken!

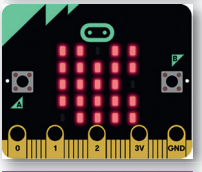
A továbbiakban készítsünk olyan programokat, amelyekben felhasználjuk ezt az algoritmust!

Feladatok

1. Legyen majdnem tele a parkolónk, csak egy üres helyet hagyjunk meg! (Most ne egyenként kapcsoljuk fel a LED-eket, használjuk a *ledek bekapcsolása* blokkot!) Írjunk olyan programot, amelyben a micro:bitnek kell megtalálnia a szabad helyet! Az A gomb megnyomásakor induljon el a szabad hely keresése! Amikor sikerült megtalálni az üres helyet, akkor töröljük a kijelzőt, és jelenítsük meg a kijelzőn a két koordinátát egymás után!

2. Módosítsuk úgy az algoritmust és a programot, hogy nyilván tudjuk tartani azt is, hogy hány próbálkozásból találta meg a micro:bit az üres parkolóhelyet. A koordináták kiírása utána ez a szám is jelenjen meg! Próbáljuk ki ötször egymás után a programot, és jegyezzük fel a próbálkozások számát! Mi volt a legkisebb és a legnagyobb szám?

3. Készítsünk olyan programot, amely mindig pontosan huszonkettő pontot kapcsol fel a kijelzőn!



4. Írjunk olyan programot, amelyben egy autót jobbra és balra mozgathatunk a képernyőn a micro:bit jobbra és balra döntésével. Az autó kezdetben a kijelző közepén legyen!

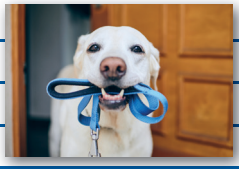
Összetett feltételek megfogalmazása logikai műveletekkel

Korábbi tanulmányainkban már megismerkedtünk az egyszerű és többirányú elágazásokkal. Ezek használatával az utasítás vagy utasítások végrehajtását feltételhez tudjuk kötni. Az elágazásban megadott feltétel akár összetett feltétel is lehet. Ilyeneket nagyon gyakran használunk az algoritmizálás és a programok készítése során. Nézzünk néhány mondatot az összetett feltételek szemléltetésére!

Ha hazaérsz 4 óráig, **és** jó az idő, **(akkor)** vidd el a kutyát sétálni!

A mondat két feltételt és egy logikai műveletet tartalmaz. Ez alapján a kutyát csak akkor kell elvinnünk sétálni, ha egyszerre teljesül az, hogy 4 óráig hazaérünk, és jó az idő. A bemeneti feltételek teljesülését (igaz) és nem teljesülését (hamis) az alábbi táblázatban összefoglaltuk. A logikai művelet eredménye (a kimenet) az utolsó oszlopban látható.

Bemenet	**Bemenet**	**Kimenet**
Első feltétel: Hazaérünk 4 óráig.	**Második feltétel:** Jó az idő.	**Igaz az első ÉS a második feltétel?** Ha igen, akkor visszük a kutyát sétálni.
igaz	igaz	igaz
igaz	hamis	hamis
hamis	igaz	hamis
hamis	hamis	hamis



▶ Az ÉS logikai művelet táblázata

A fenti táblázatból jól látszik, hogy az ÉS logikai művelet eredménye (a kimenet) csak akkor igaz, ha mindegyik bemeneti állítás igaz. Láthatjuk, hogy a bemenet és a kimenet is igaz vagy hamis értéket vehet fel. Ez tehát olyan adat, amely az úgy nevezett *logikai adattípusba* tartozik.

Most nézzük az alábbi mondatot!

Ha lázad van **vagy** köhögsz, **(akkor)** maradj otthon!

Bemenet	**Bemenet**	**Kimenet**
Első feltétel: Lázunk van.	**Második feltétel:** Köhögünk.	**Igaz az első ÉS a második feltétel?** Ha igen, akkor otthon kell maradnunk.
igaz	igaz	igaz
igaz	hamis	igaz
hamis	igaz	igaz
hamis	hamis	hamis

▶ A VAGY logikai művelet táblázata

Láthatjuk, hogy a VAGY logikai művelet kimenete akkor lesz igaz, ha valamelyik állítás igaz volt. Hamis csak akkor lesz, ha mindegyik állítás hamis volt.

Lássunk egy példát a tagadásra is!

Ha nem esik az eső, **(akkor)** elmegyünk biciklizni!

Bemenet	Kimenet
Feltétel: Esik az eső?	**Igaz a feltétel tagadása?** Ha igen, akkor elmegyünk biciklizni.
igaz	hamis
hamis	igaz

► A NEM (más néven tagadás) logikai művelet táblázata

Láthatjuk, hogy a NEM (tagadás) művelet kimenete akkor lesz igaz, ha a bemeneti állítás hamis volt.

Feladat

Fogalmazzunk meg mi is olyan mondatokat, amelyek az ÉS, VAGY, NEM logikai műveletekre mutatnak példát!

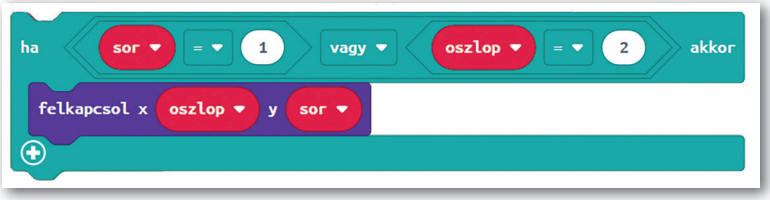
Rajzoljunk ki pontokat összetett feltételek alapján!

A következőkben olyan ábrákat készítünk, amelyekben a LED-eket koordinátáik szerint fogjuk fel- és lekapcsolni. Azt, hogy melyik pont legyen felkapcsolva, különböző feltételek segítségével fogalmazzuk meg.

► Logikai műveletek blokkjai a MakeCode felületen

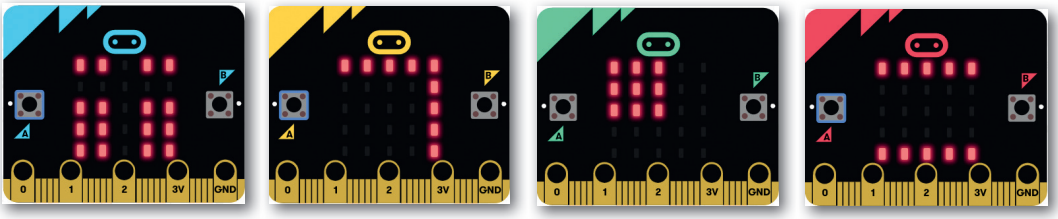
Feladatok

1. Készítsük el azt a programot, amelyet a további feladatok során módosítani fogunk. Az *A* gomb megnyomásakor törlődjön a kijelző! Ezt követően ezer alkalommal válasszunk véletlenszerűen sor- és oszlopkoordinátát nulla és négy között, majd kapcsoljuk fel az ezen a koordinátán lévő LED-et! Mit tapasztalunk a program kipróbálásakor, és miért?
2. Módosítsuk úgy a programot, hogy csak akkor rajzoljuk ki a pontokat, ha bizonyos feltételek teljesülnek! Az alábbi példában azt látjuk, hogy csak akkor rajzoljuk ki a pontot, ha a sor értéke egy, vagy az oszlop értéke kettő. Először gondoljuk át, hogy vajon milyen ábra jelenik meg a kijelzőn! (A továbbiakban tegyük fel, hogy a micro:bit minden pontot kiválaszt véletlenszerűen a rajzolás során!) Próbáljuk ki a programot a gyakorlatban is!



► Összetett feltétel a pont felkapcsolásához

3. Módosítsuk a *„vagy”* logikai műveletet *„és”* logikai műveletre. Mit gondolunk, hogyan változik meg az ábra? Próbáljuk ki a gyakorlatban is!
4. Hogyan tudnánk előállítani az alábbi ábrákat összetett feltételek használatával? Próbáljuk ki a gyakorlatban is az elgondolásunkat!



5. Jelenítsünk meg érdekes ábrákat összetett logikai feltételek alkalmazásával. Alakítsunk párokat, és fejtsük meg, hogy a párunk micro:bitjén látható ábra kirajzolásához milyen logikai feltételeket kellett beállítani!

Irány a világűr!

A továbbiakban is a micro:bit kijelzője lesz a főszereplő. Először megnézzük, hogyan lehetne a hullócsillagok mozgását szimulálni. Ezt a tudást felhasználva pedig egy olyan játékot készítünk, amelyben egy űrhajót kell átvezetnünk a kisbolygók (aszteroidák) között.

Nézzünk hullócsillagokat!

Mielőtt elindulnánk a világűrbe, vessünk egy pillantást az égboltra, hátha látunk hullócsillagokat!

Készítsünk olyan programot, amelyben a micro:bit kijelzőjének felső sorában véletlenszerűen kiválasztott helyen megjelenik egy meteor, ami aztán lefelé mozog a legalsó sorig, majd eltűnik!

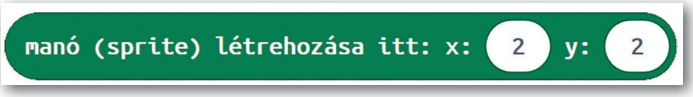
Ezt a programot akár a korábban bemutatott módszerrel, a LED-ek egyenként történő fel- és lekapcsolásával is megvalósíthatnánk. Most azonban egy másik módszert mutatunk be, amellyel egyszerűbben megoldhatjuk ezt a feladatot.

Ha egy grafikus programozási környezetben (pl. Scratch) készítenénk el a programot, akkor megváltoztatnánk a szereplő alakját egy meteor alakra. Beállítanánk a szereplő irányát úgy, hogy lefelé nézzen, és előreléptetnénk megadott lépéssel addig, míg el nem éri a képernyő alját. Vagyis a szereplőt nem a koordináták beállításával mozgatnánk, hanem az előre-hátra parancsok és az irányváltások segítségével.

Ugyanezt megtehetjük a micro:bit kijelzőjén is. A szükséges blokkokat a *Játék* kategóriában találjuk, amely a *Haladó* kategóriák között található a MakeCode (https://makecode.microbit.org/) felületen.

► Játék kategória a MakeCode felületen

A szereplőnket az alábbi blokk segítségével hozhatjuk létre a megadott kezdő koordinátán:

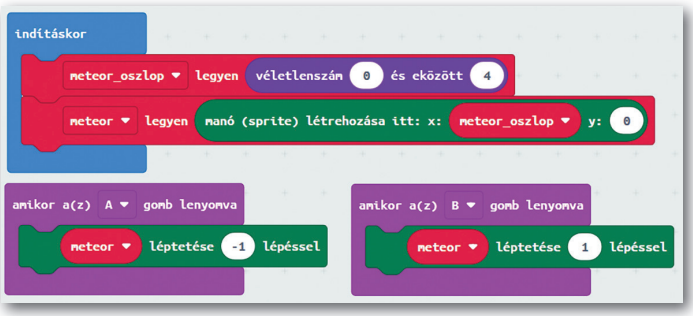


A MakeCode környezetben a szereplőt angolul sprite-nak (kiejtve *szprájt*) hívják, amelyet magyarul manónak is szoktak nevezni a különböző programozási környezetekben. Ez egy olyan pontot jelöl, amelynek van iránya, és az adott irányba el tud mozdulni megadott lépésnyit. Amikor elhelyezzük a pontot, akkor alapesetben jobbra néz, vagyis előrelépéskor jobbra tesz egy lépést.

A szereplőnek nevet kell adnunk, ezért egy változót kell létrehoznunk, amelynek értéke a fenti blokk lesz.

Feladatok

1. Készítsük el az itt látható programot!



▶ Pont mozgatása az *A* és *B* gombokkal

Magyarázat: Programunkban a *meteor_oszlop* változó értéke egy véletlenszerűen kiválasztott szám lesz 0 és 4 között. A *meteor* nevű változóba helyezzük el a szereplő létrehozásához szükséges blokkot. Ennek hatására a meteor a felső sorban, a véletlenszerűen kiválasztott oszlopban jelenik meg. Ha a *B* gombot megnyomjuk, akkor a pont előremozdul az aktuális irányba. Mivel alapesetben jobbra néz a szereplőnk, ezért jobbra fog lépni. A hátralépést úgy tudjuk megvalósítani, hogy negatív számmal lépünk előre. Ezt látjuk az *A* gomb eseményénél.

Próbáljuk ki a fenti kódot a gyakorlatban is! Mit tapasztalunk, mi történik, ha a szereplő eléri a kijelző szélét? Eltűnik? Visszapattan? Nem mozog tovább abban az irányban?

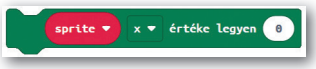
2. Változtassuk meg a programot úgy, hogy a szereplő ne jobbra induljon el az előreléptéskor, hanem lefelé! Fedezzük fel önállóan, hogy az elforduláshoz melyik blokkot kell használni és milyen paraméterrel!

3. Tudjuk, hogy az állandóan nevű blokkban elhelyezett utasítások folyamatosan ismétlődnek. Illesszük be az alábbi kódot a programunkba!
Ennek hatására a meteor lefelé mozog, míg el nem éri a legalsó sort. A szünet beállítása azért szükséges, hogy ne legyen túl gyors a pont mozgása. Próbáljuk ki a kódot a gyakorlatban is!

▶ Meteor előreléptetése

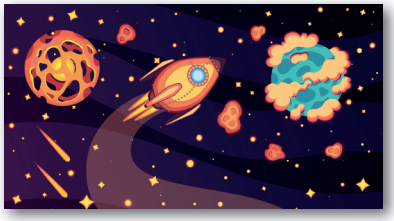
4. Fejlesszük tovább az alkalmazást úgy, hogy ha a meteor eléri a legalsó sort, akkor kerüljön vissza a legfelső sorba, egy véletlenszerűen kiválasztott oszlopba!
   A szereplő koordinátáinak lekérdezéséhez használhatjuk az itt látható blokkot! Természetesen nemcsak az *x*, hanem az *y* koordinátát is lekérdezhetjük.
   A szereplő koordinátájának megváltoztatásához is találunk megfelelő blokkot a *Játék* kategóriában.
   Ha sikeresen megoldottuk ezt a feladatot is, akkor elkészültünk a hullócsillag-szimulációval. Láthatjuk, hogy a meteorok egymás után jelennek meg, és hullanak le a kijelzőn.

► A szereplő *x* koordinátájának lekérdezése



► A szereplő *x* koordinátájának beállítása

Vágjunk át az aszteroidamezőn!



Most képzeljük el, hogy a korábban megvalósított hullócsillagok az űrhajónk előtti akadályokat (kisbolygókat) jelölik, amelyeket ki kell kerülnünk az utunk során! Fejlesszük tovább ennek megfelelően az alkalmazást!

Feladat

Hozzunk létre egy szereplőt *űrhajó* névvel a legalsó sor középső pontján. Oldjuk meg, hogy az *A* gombbal az űrhajót balra lehessen léptetni, a *B* gombbal pedig jobbra!

Ha ezt a feladatot is megoldottuk, már csak néhány kisebb fejlesztés szükséges ahhoz, hogy egy valódi játékprogramot hozzunk létre.

A játék célja az, hogy ne ütközzünk neki a kisbolygónak. Ha ez megtörténik, akkor álljon le a játék.

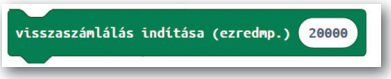
Ezt a feltételt az itt látható blokkok segítségével könnyen megvalósíthatjuk a programunkban. A blokk közvetlenül a meteor léptetése után kerüljön! Próbáljuk ki a programot!

► Vége a játéknak, ha az űrhajó hozzáér a meteorhoz

Izgalmasabb lenne a játékunk, ha pontokat kapnánk azért, ha elkerültük az ütközéseket. A játék kategóriában a pontszámok beállítására és növelésére is találunk blokkokat. A játék indulásakor (az *indításkor* blokkban) nullázzuk le a pontszámot. Amikor pedig sikeresen kikerültünk egy kisbolygót, növeljük meg a pontszámot! Az elért pontszám automatikusan meg fog jelenni a micro:bit kijelzőjén akkor, amikor véget ér a játék. Amikor pedig a pontszámunk nő, akkor a kijelző felvillan.

► Pontszám beállítása, illetve növelése

Állítsuk be azt, hogy a játék maximum 20 másodpercig tartson! Az ehhez szükséges *visszaszámlálás* blokkot az *indításkor* blokkban kell elhelyeznünk.



► Visszaszámlálás indítása

Feladat

Végezzük el ezt a fejlesztést önállóan az itt látható blokkok megfelelő helyre való beillesztésével! Ezek után próbáljuk ki a játékot! Hány pontot sikerült összegyűjteni a játék során? Hasonlítsuk össze a kapott pontokat a társainkéval!

Gyakorlás, saját ötletek megvalósítása

Most már tudjuk, hogyan hozhatunk létre a Játék kategória elemeivel egyszerű, de mégis izgalmas játékokat.

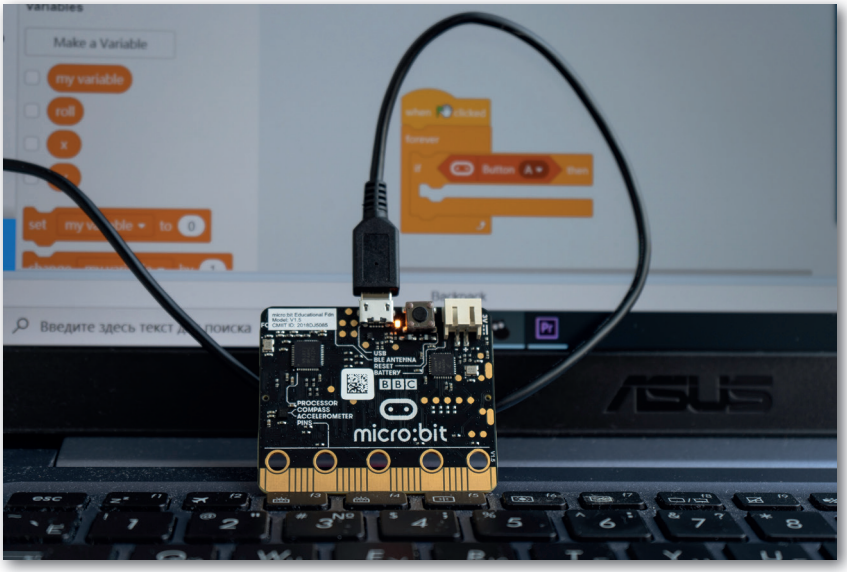
A következőkben engedjük szabadon a képzeletünket! Fejlesszük tovább a játékot az elképzeléseinknek megfelelően, illetve találjunk ki új játékokat!

Egyéni feladatok

1. Gondoljuk át, hogyan tudnánk még izgalmasabbá tenni az űrhajós játékot! Hogyan lehetne megoldani például, hogy az idő múlásával egyre nehezedjen a feladat? Valósítsuk meg az ötleteinket!
2. Fejlesszük tovább úgy az alkalmazást, hogy ne csak gombok megnyomásával lehessen irányítani az űrhajót, hanem más gesztusok segítségével is!
3. Milyen új funkciót tudnánk elképzelni, amelyhez összetett feltétel megfogalmazása szükséges? Valósítsuk meg a gyakorlatban!

Találjunk ki saját játékot!

1. Alkossunk három-négy fős csoportokat! Tervezzünk olyan játékot a micro:bitre, amelyben egy vagy több szereplőt kell irányítanunk egy adott cél eléréséhez! A játék során lehessen pontokat szerezni! Oldjuk meg azt is, hogy a játék egyre nehezedjen! Előbb készítsük el a program algoritmusát, és csak utána álljunk neki a megvalósításnak! Törekedjünk arra, hogy mindenki kapjon olyan részfeladatot, amiért ő a felelős!
2. Ha elkészültünk a munkánkkal, mutassuk be egymásnak a játékokat, és próbáljuk ki egymás programjait! Gyűjtsük össze, hogy mi tetszett az egyes játékokban, mi okozott nehézséget a használat során, és hogy milyen módon lehetne még továbbfejleszteni a programokat!



Valódi robotok programozása

Folytassuk valódi robotok programozásával!

Napjainkban rengeteg, a gyerekek számára készített oktatási célú robot áll rendelkezésre. Tankönyvünkben néhány olyan robotot mutatunk be, amelyek ideálisak lehetnek a robotikával való ismerkedéshez. De természetesen más (hasonló tudású) robotok is használhatók a foglalkozások során.

Kezdjük a Lego Mindstorms EV3 robotkészlet rövid bemutatásával!

► LEGO Mindstorms EV3 szett

A Lego Mindstorms EV3 készlet

A Lego cég már több évtizede gyárt olyan készleteket, amelyekkel betekintést nyerhetünk a robotika alapjaiba. Napjaink egyik legnépszerűbb robotika készlete a Lego Mindstorms EV3. A készletből több fajta vásárolható, a különböző csomagok esetenként más-más érzékelőket (szenzorokat) és alkatrészeket tartalmaznak.
Nézzük, melyek ezek!

EV3 tégla. Ez a robot irányítóközpontja. Ehhez tudjuk csatlakoztatni a többi alkatrészt, a motorokat és a szenzorokat.

A tégla tetején egy kijelző található, amelyen szövegeket, számokat vagy akár rajzokat is megjeleníthetünk. A kijelző alatt gombok találhatók, amelyek körül egy jelzőfény világíthat, illetve villoghat piros, zöld, vagy narancsszínben.

Ez az eszköz tartalmazza a működéshez szükséges elemeket, vagy a tölthető akkumulátort. A tégla elején és hátulján négy-négy portot (vagyis csatlakozó aljzatot) találunk. Az egytől négyig sorszámozott csatlakozókba az érzékelőkhöz csatlakozó kábeleket dughatjuk be. Az A, B, C és D jelű portokhoz pedig a motorokat lehet csatlakoztatni a kábelekkel.

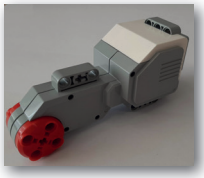
► Az 1-es, 2-es, 3-as és 4-es számú portok

► Az A, B, C, D jelű portok

Az érzékelők

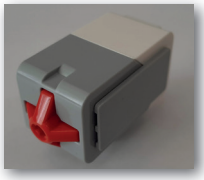
A készlethez az alábbi érzékelők használhatók. Ezek vagy részei az adott csomagnak, vagy külön szerezhetők be.

Színérzékelő: Ez a szenzor hét különböző szín (fekete, kék, zöld, sárga, piros, fehér és barna) megkülönböztetésére képes. Emellett használható fényerősség mérésére is. Ha a környezet nagyon sötét, akkor 0 értéket, ha nagyon világos, akkor pedig 100-as értéket ad vissza.



Nagy motor: Ezzel biztosíthatjuk a robot meghajtását, például hajthatja a jármű kerekét. Percenként 160-170 fordulatra képes.

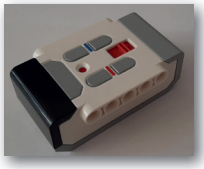
Közepes motor: Ez a motor kisebb és gyorsabban reagál, mint a nagy motor. Percenként 240-250 fordulatra képes.



Ütközésérzékelő: Azt érzékeli, hogy a rajta elhelyezett piros nyomógomb benyomott vagy kiengedett állapotban van-e. Benyomott állapotban 1-es értéket, kiengedett állapotban 0 értéket ad vissza. 2-es értéket akkor ad vissza, ha a lenyomás után nem sokkal fel lett engedve a gomb.

Ultrahangos távolságérzékelő: Ahogy a nevében benne van, távolságérzékelésre használható, 1 cm és 250 cm távolság között. Hasonló a működési elve ahhoz, mint ahogy egy denevér tájékozódik a sötétben. Ultrahangot bocsát ki, és a tárgyakról visszavert hullámokat érzékeli.

Infravörös érzékelő: Ez a szenzor több üzemmódban használható. Egyrész működik távolságérzékelőként kb. 70 cm távolságig. A tárgy és az érzékelő közti távolságot nem centiméterben adja meg, hanem 0 jelenti a nagyon közeli távolságot, és 100 a távolit. Képes a távirányító jelének észlelésére kétméteres távolságig. Így például beprogramozható arra a robot, hogy az infravörös távirányító (jeladó) felé forduljon, majd haladjon az irányába.



Távirányító infravörös irányjeladó: Segítségével távirányítható robotot építhetünk, illetve ahogy már említettük, jeladóként is működik.

Gyroszenzor: Segítségével a robot elfordulásának szögét lehet megmérni. Ez az érzékelő viszont csak a speciálisabb készletekben érhető el.

Milyen alaprobotot építsünk a kísérletezéshez?

A készletből több fajta robot összeállítható. Az alaprobotok összeszerelési útmutatói elérhetőek a Lego weboldalán (http://tiny.cc/ev3robotok), de ha már gyakorlatot szereztünk az építésben, saját elképzelésünknek megfelelő robotokat is összeállíthatunk.

A robotikával való ismerkedéshez olyan járművet célszerű építeni, amelynél:

- nagy motorok hajtják meg a jármű bal és jobb oldali kerekét,
- a jármű elejére egy lefelé néző színérzékelő van felszerelve,
- egy ultrahangos távolságérzékelő (vagy infravörös érzékelő) van elhelyezve a jármű elején úgy, hogy az előrenézzen.

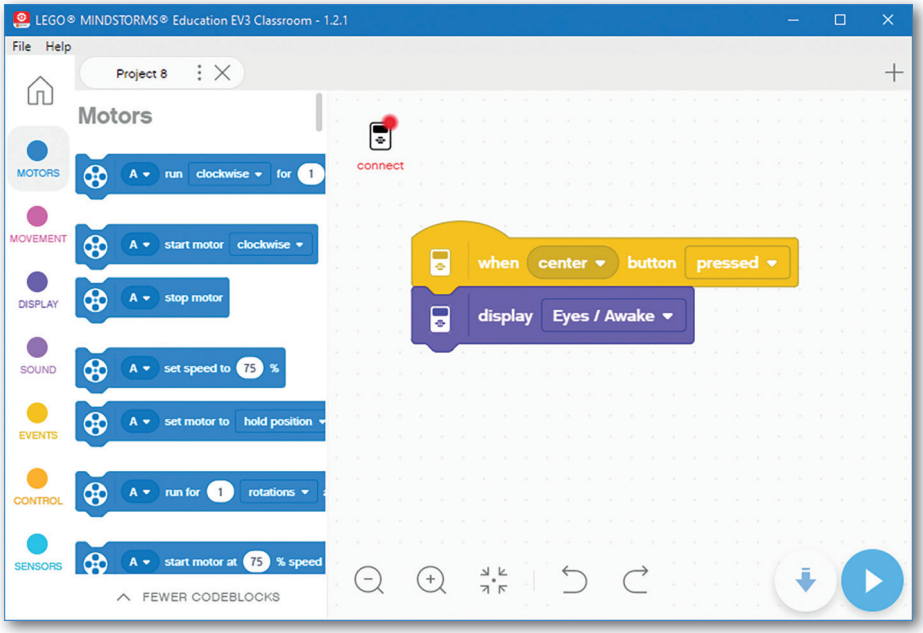
A készlet programozási lehetőségei

A készlet irányítóközpontját (a téglát) többféle módon is elérhetjük a számítógépről, ahol majd a programokat fogjuk elkészíteni. Egyrészt összeköthetjük USB-kábellel, másrészt használhatunk vezeték nélküli (Bluetooth, wifi) technológiákat is.

A tégla programozása több programozási környezetben is lehetséges. Ezek közül kettőre térünk ki. Mind a két környezetre igaz, hogy egy blokkprogramozási felület áll a rendelkezésünkre. Az egyes blokkok kategóriák szerint vannak rendezve, épp-úgy, mint a korábban használt környezetekben.

EV3 Classroom környezet

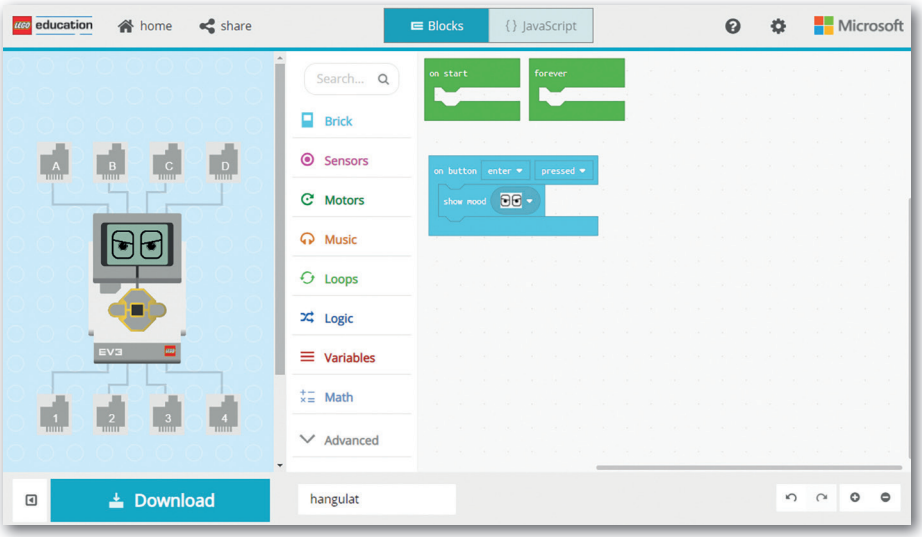
A programok készítésére használhatjuk az *EV3 Classroom* szoftvert, amely ingyenesen letölthető a Lego weboldaláról. A programok összeállítására az alábbi felület szolgál. Bal oldalon láthatjuk a kategóriákat a használható blokkokkal.



► Az EV3 Classroom szoftver felülete

Az EV3 - Blokk / Javascript szerkesztőfelület

A programozáshoz használhatunk online elérhető felületet is (https://makecode.mindstorms.com/). Mivel ez a környezet webböngészőből használható, program telepítésére nincs szükség.



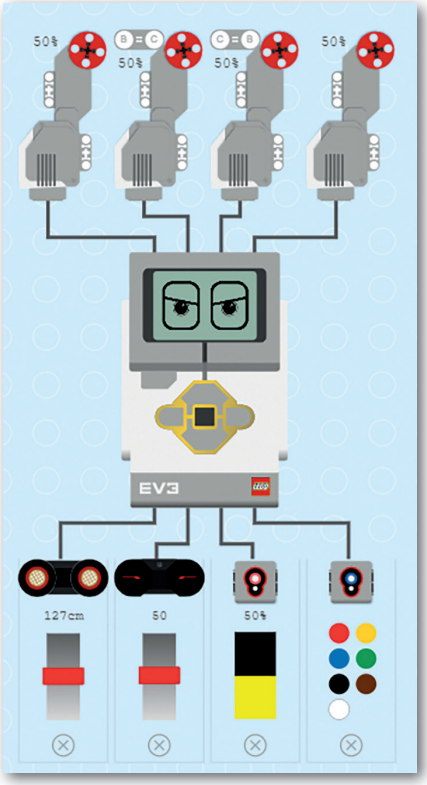
► Az online szerkesztői felület

A *Makecode Mindstorms* felülete azért különleges, mert ebben egy szimulátort is használhatunk.

Itt éppúgy lenyomhatjuk a gombokat, mint a valódi EV3 téglán. A kijelzőn pedig láthatjuk, hogy milyen ábra jelenne meg a megírt kód hatására.

Amennyiben a projektünkben motorokat is használtunk, akkor azok is megjelennek, sőt amikor működésbe lépnek, akkor a képernyőn láthatjuk, hogy elkezdenek forogni. A motorok melletti szám jelenti a motor sebességét. Minél nagyobb számot látunk, annál nagyobb sebességgel forog a tengely.

Amennyiben érzékelőket használunk, azok is megjelennek a szimulátorban. Az ábra alsó részén egy ultrahangos távolságérzékelőt, egy infravörös érzékelőt, egy fényerősség-érzékelőt és egy színérzékelőt láthatunk. Az alattuk lévő csúszkák és ikonok segítségével módosíthatjuk az értéküket. Így a programunkat még azelőtt tesztelhetjük, hogy azt az eszközre feltöltenénk és futtatnánk.



A program feltöltése, futtatása

A programokat USB-kábel vagy vezeték nélküli (Bluetooth, wifi) kapcsolat segítségével tudjuk a robotra feltölteni. A feltöltés után a programok az EV3 tégla menürendszerében elérhetőek, és kiválasztás után futtathatóak lesznek.

Az eszköz menürendszerében a gombok segítségével tudunk lépkedni balra, jobbra, fel és le. Középen találhatunk egy kiválasztó (*Enter*) gombot, a bal felső sarokban pedig egy Kilépés (*Escape*) gombot. A középső gomb hosszú lenyomásával tudjuk az eszközt bekapcsolni.

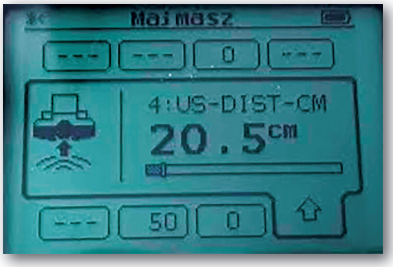
A tégla menürendszerében olyan funkciót is találunk, amellyel a csatlakoztatott szenzorok értékeit is megtekinthetjük. Az alábbi képen azt látjuk, hogy a távolságérzékelő szenzor 20,5 cm távolságra érzékel egy tárgyat maga előtt. Az is látszik, hogy a szenzor a 4-es számú porthoz van csatlakoztatva.

Az EV3 Classroom programozási környezet előnye, hogy abban a valós robot szenzorai által mért adatok a képernyőn is megjelennek. Ehhez természetesen a robotot csatlakoztatnunk kell a számítógéphez.

► A robot érzékelőinek állapota

A fenti ábrán azt láthatjuk, hogy a robot jelenleg csatlakoztatva van a számítógéphez. Ezt a zöld kör jelzi az első ikon jobb felső sarkában. A 2-es számú porthoz csatlakozó fényérzékelő most 49%-os fényerősségszintet jelez. A 3-as portra kötött ütközésérzékelő 0 értéket ad vissza, vagyis most nincs benyomva az ütközésérzékelő gombja. A 4-es porthoz csatlakozó távolságérzékelő előtt 11,4 cm-rel van egy tárgy. A C portra kötött nagy motor pedig 70°-ot fordult el az indítás óta.

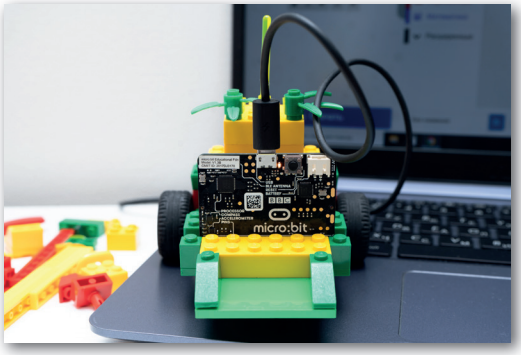
► Az EV3 tégla kijelzője a menüvel



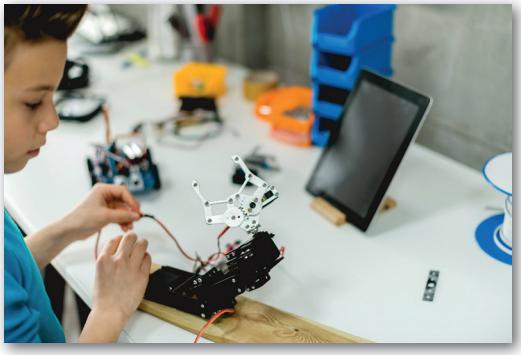
► A távolságérzékelő szenzor által mért érték

Micro:bitekre épülő robotok

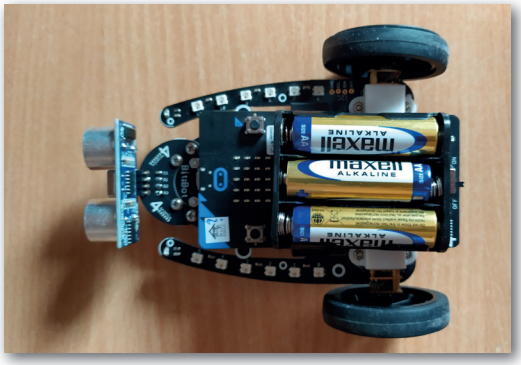
A micro:bit eszközhöz is számos olyan kiegészítőt szerezhetünk be, amelyek segítségével robotokat építhetünk. Ekkor a micro:bitet el kell helyeznünk az adott eszköz csatlakozójában. Ebben az esetben tehát a micro:bit lesz az eszköz agya, amelyet programozni fogunk.



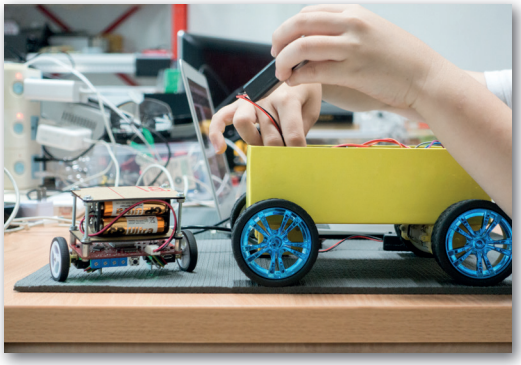
▶ Micro:bit egy Lego járműben



▶ Robotkar



▶ Bit:bot jármű, levehető érzékelőkkel



▶ Vezeték nélküli kapcsolattal irányítható jármű

Robotjármű építésére több készlet is alkalmas (pl. bit:bot, micro:Maqueen). Ezek a készletek is olyan motorokat tartalmaznak, amelyek majd gumiborítású kerekeket fognak meghajtani.

Emellett olyan érzékelők is helyet kaptak a csomagban, amelyek alkalmasak lehetnek a szín-, illetve fényerősség érzékelésére, illetve távolságérzékelésre. Így készíthetünk olyan programokat, amelyek segítségével a jármű kikerülheti az akadályokat, illetve akár vonalat is követhet.

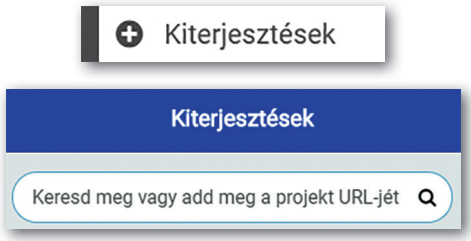
Az eszközökre akár különböző színű LED-eket is felszerelhetünk, illetve egyszerűbb hanghatások megszólaltatására is alkalmasak lehetnek.

A készletek programozási lehetőségei

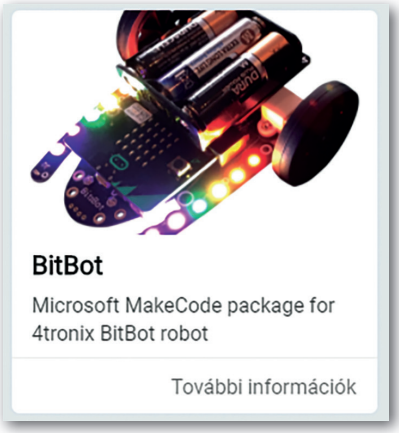
Ezen készletek előnye, hogy ugyanazt a programozási felületet használhatjuk, mint amelyet a micro:bit programozása során már megszokhattunk (https://makecode.microbit.org/).

A különböző készletekhez tartozó kiterjesztéseket a *Kiterjesztések* gombra kattintva jeleníthetjük meg. A készlet nevére akár rá is kereshetünk.

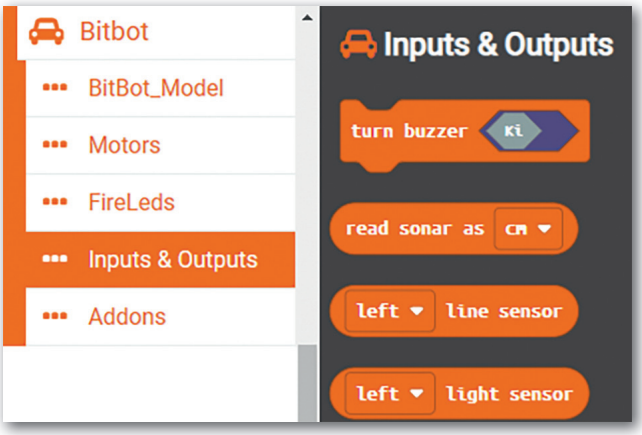
Ezek után már csak a készlet blokkjára kell kattintanunk ahhoz, hogy az eszköz használatához szükséges kategóriák és blokkok betöltődjenek a szerkesztőablakba.



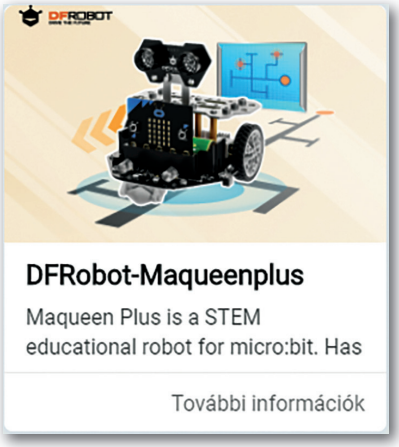
► Keresés a kiterjesztések között



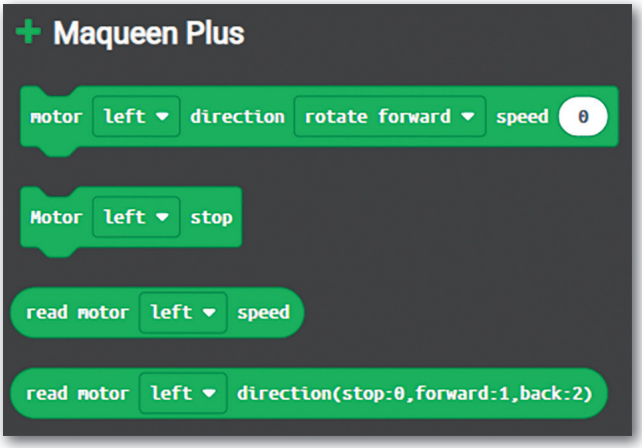
► BitBot jármű a kiterjesztések között



► A BitBot járműhöz tartozó kategória és a használható blokkok



► Maqueen plus robot blokkja a kiterjesztések között



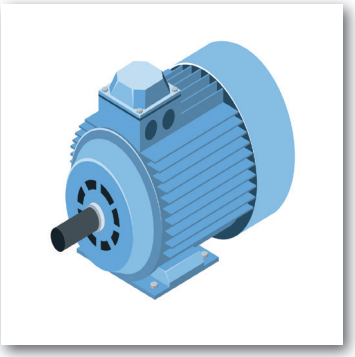
► A Maqueen plus robot működtetéséhez szükséges blokkok egy részlete

Hasznos tanácsok a járművek irányításához

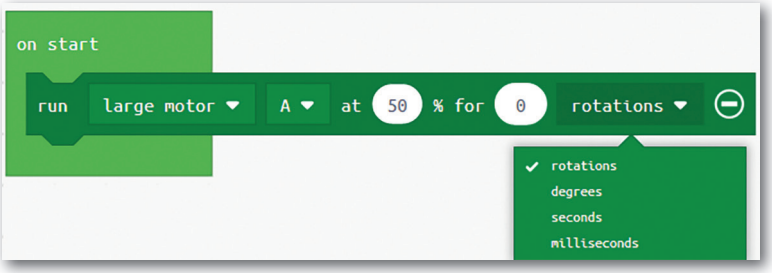
Először olyan feladatokat fogunk megoldani, amelyekben a robotjárművünk különböző útvonalakat jár be. Ezért a következőkben a motorok működtetéséhez szükséges tudnivalókat tekintjük át. Mivel jellemzően angol nyelvű parancsokat használunk a felületeken, ezért a fogalmak angol nyelvű változatait is feltüntetjük.

A motorok akár többféle üzemmódban is működhetnek, az adott robot típusától függően. Általában a következő lehetőségek állnak rendelkezésre.

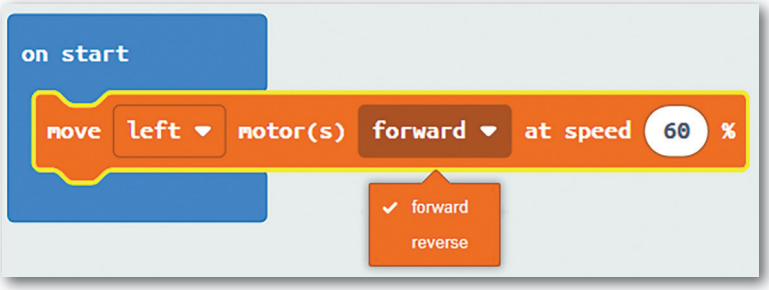
- A motort működtethetjük megadott ideig (pl. 2 másodpercig). A másodpercek angolul: *seconds*.
- A motort bekapcsolhatjuk (*start*), és az folyamatosan működhet addig, amíg le nem állítjuk (*stop*).
- A motor működhet addig, míg megadott szögértéket (pl. 180°) nem fordul el a tengelye. A szögeket angolul *degrees* néven találjuk a programozási felületeken.
- A motort működtethetjük addig, amíg a tengelye valahány fordulatot meg nem tesz (pl. 10 fordulat). A fordulatokat angolul a *rotations* jelenti.



Emellett megadhatjuk azt is, hogy a motor mekkora sebességgel (*speed*) működjön.



► Nagy motor működtetése 50%-os sebességgel, megadott forgási szög, másodperc, ezredmásodperc értékkel
(Lego Education EV3 Makecode környezet)



► A bal oldali (left) motor működtetése előre (forward) irányba, 60%-os sebességgel (speed).
(Bitbot jármű vezérlése a Makecode környezetben)

Hajtási irány és fékezés

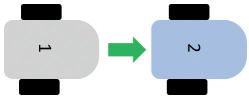
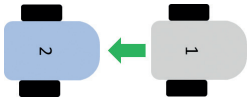
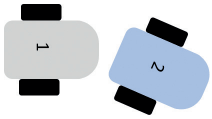
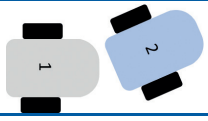
A motor hajtási iránya is beállítható. Egyes robotoknál ezt az előre- (*forward*) és hátramenet (*reversed*) paraméterekkel tehetjük meg. Más robotok esetén ugyanezt a motor forgási sebességének előjelével jelezhetjük. Pozitív érték esetén a motor előrehajtja a kereket, negatív érték hatására pedig hátrafelé.

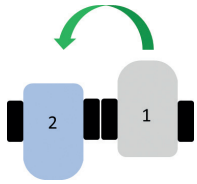
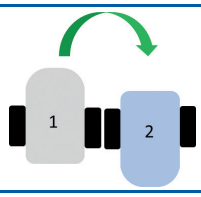
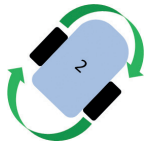
Azt, hogy a motor fékezve álljon le vagy sem, szintén beállíthatjuk. Ha fékezve áll le a motor (*break*), akkor a megállás hirtelen történik. Ha nincs fék beállítva, akkor a megállás elnyújtottan, lassulva megy végbe. Egyes felületeken ezt a funkciót nem fékezésnek hívják, hanem úgy fogalmaznak, hogy a robot megálláskor tartsa meg a helyzetét (*hold position*).

A jármű kanyarodása

Egyszerű oktatási célú robotoknál nem jellemző, hogy kormányzott kerekekkel irányíthatnánk a robotot. Ezért a kanyarodást úgy tudjuk elérni, hogy a jármű két oldalán lévő motorokat eltérő sebességgel működtetjük.

Az alábbi táblázatban néhány példát látunk a jármű haladására vonatkozóan. A sebességet most százalékos értékkel jelezzük. Pozitív érték esetén a motor előrehajtja a kereket, negatív érték esetén pedig hátra.

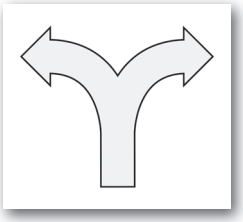
	Bal oldali motor sebessége	Jobb oldali motor sebessége	Haladási irány	
1.	50%	50%	Mivel mind a két motor azonos sebességgel forog, a jármű egyenes irányba halad előre.	
2.	–30%	–30%	Mind a két motor azonos sebességgel forog, de ellenkező irányban. Ezért a jármű hátrafelé halad egyenes vonalban.	
3.	60%	50%	Most a bal oldali motor gyorsabban forgatja a tengelyt. Ezért a jármű jobbra kanyarodik. Mivel nem sok különbség van a két kerék sebessége között, a jármű lassan, nagy ívben kanyarodik jobbra.	
4.	100%	30%	Most is jobbra kanyarodik a jármű, de kisebb ívben. Ennek oka, hogy nagyobb a különbség a két motor sebessége között.	
5.	30%	100%	Most balra kanyarodik a jármű, mert a jobb oldali motor gyorsabban hajtja a kereket.	

6.	0%	50%	A bal oldali kerék most nem mozog, a jobb oldali viszont igen. Ezért a jármű kis ívben fordul a bal oldali kereke körül.	
7.	30%	0%	Most a jobb oldali kerék áll egy helyben. Ezért a jármű jobbra kanyarodik, a jobb oldali kereke körül fordul.	
8.	30%	–30%	Most a bal oldali motor előre-, a jobb oldali motor hátrafordul ugyanakkora sebességgel. Ezért a jármű egy helyben forog, mégpedig jobbra.	
9.	–50%	50%	Most a bal oldali motor hátra-, a jobb oldali motor előrefordul ugyanakkora sebességgel. Ezért a jármű egy helyben forog, mégpedig balra.	

A valóságban lehet, hogy majd azt tapasztaljuk, hogy picit máshogy működik a jármű a táblázatban leírtakhoz képest. Például hiába állítjuk be mindkét motor sebességét ugyanakkora értékre, a jármű mégsem egyenes vonalban halad. Ennek több oka is lehet. Lehet, hogy a két kerék eltérő felületen halad, vagy el van ferdülve a kerék. Ilyenkor kísérletezés útján állíthatjuk be a megfelelő értékeket.

Válaszoljunk az alábbi kérdésekre!

1. Milyen valódi járművekre jellemző az, hogy egy helyben meg tudnak fordulni? Ezeket jellemzően hogyan lehet irányítani? Kormánykerékkel? Esetleg más módon? Nézzünk utána!
2. Ha a bal oldali kerék előreforog 70%-os teljesítménnyel, a jobb oldali kerék előreforog 68%-os teljesítménnyel, akkor hogyan fog haladni a jármű?
3. Az autóvezetők sokszor Y (ipszilon) megfordulással fordulnak meg egy széles úton. Ezt például úgy lehet megvalósítani, hogy hátratolatunk úgy, hogy a kormányt jobbra tekerjük, majd előremegyünk úgy, hogy a kormányt balra tekerjük. Ekkor az autó egy Y betűhöz hasonló alakot ír le. Hogyan lehetne ezt a mozgást egy robotjárművel megcsinálni? Készítsük el a program algoritmusát!



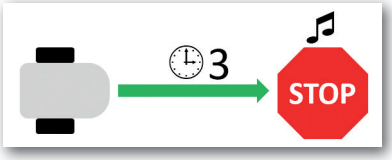
Hasznos tanácsok a robotokkal való munkához

- A robotok érzékeny és drága eszközök. Vigyázzunk rájuk, hogy minél több ideig használhassuk őket!
- Védjük egymás testi épségét! Figyeljünk arra, hogy a robot ne ütközhessen neki embereknek! Az ember éppúgy megsérülhet, mint az eszköz.
- Csak olyan felületen használjuk a robotokat, amely biztonságos! Ne legyen lejtős a felület, mert lecsúszhat róla a robot! Ne tegyük az asztal szélére, mert véletlenül lesodorhatjuk azt!
- Biztos kézzel, de ne túl erősen fogjuk meg az eszközt, ha odébb szeretnénk mozdítani. Így elkerülhetjük azt, hogy letörjünk egy érzékelőt vagy más alkatrészt.
- Ne gurítsuk a robotot, illetve ne blokkoljuk a kerekeit a motorok működése közben!
- Ne ejtsük le az eszközt, mert könnyen megsérülhet és tönkremehet!
- Csak olyan elemeket, akkumulátorokat használjunk, amelyeket a gyártó javasol az eszközhöz!
- Ha vezeték nélküli kapcsolatot használunk, legyünk figyelmesek, hogy valóban a mi eszközünkhöz kapcsolódunk, nem pedig egy társunk robotjának programját írjuk felül!
- Csak olyan eszközökkel bővítsük a robotot (ha van erre lehetőség), amelyet a gyártó javasol!

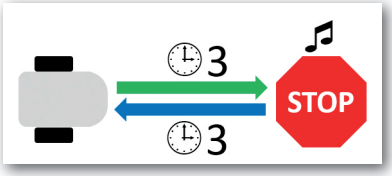
Feladatok megoldása robotokkal

Attól függően, hogy milyen robot áll rendelkezésünkre, ismerkedjünk meg a hozzá tartozó programozási felülettel, valamint a legfontosabb blokkok jelentésével! Ha nincs magyarítva a felület, és nem ismerjük az angol alapszavak jelentését, a parancsok fordításához használjunk online szótárat (pl. http://szotar.sztaki.hu/).

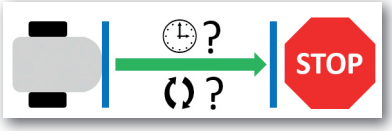
Ezek után oldjuk meg az alábbi feladatokat!



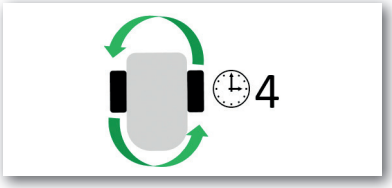
1. A robot haladjon előre egyenes vonalban 3 másodpercig, majd álljon meg! Megálláskor adjon ki egy hangot!



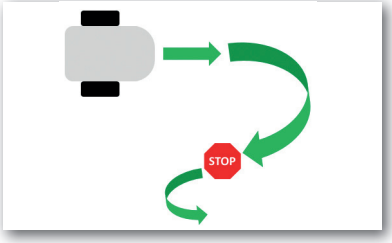
2. Módosítsuk az előbbi programot úgy, hogy a robot a megállás után egyenes vonalban 3 másodpercig hátrafelé haladjon! Jelöljük meg a kiindulási állapotot és a visszaérkezési állapotot ceruzával vagy ragasztószalaggal. Próbáljuk ki a programot fék alkalmazásával és fék nélkül is, amennyiben ezt az adott robot támogatja. Mit tapasztalunk? Pontosan oda érkezett vissza a robot, ahonnan elindult? Ha nem, akkor mi lehet annak az oka?



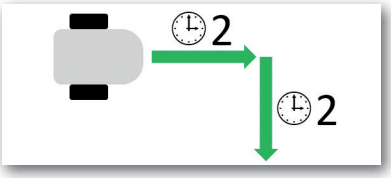
3. A padlón jelezze egy vonal a startvonalat! Vele párhuzamosan legyen egy célvonal is! Kísérletezzük ki, hogy hány másodpercig vagy tengelyfordulatig kell haladnia a robotnak ahhoz, hogy pontosan a cél előtt álljon meg! Ha az szeretnénk elérni, hogy különböző sebességbeállításoknál is ugyanakkora távolságot tegyen meg a jármű, akkor azt hogyan érhetjük el? Ha esetleg nagyobb átmérőjű kereket szerelnénk a robotra, akkor hogyan változna a megtett távolság?



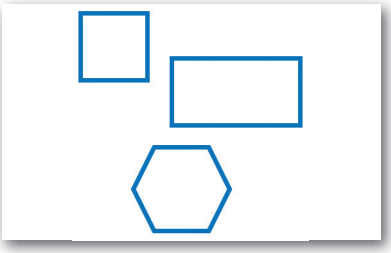
4. A robot forogjon 4 másodpercig egy helyben az óramutató járásával ellenétes irányban!



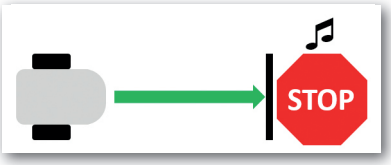
5. A robot haladjon előre, majd forduljon jobbra nagy ívben! Álljon meg, majd balra forduljon kis ívben!



6. A jármű haladjon előre 2 másodpercig, majd forduljon jobbra 90°-ot, és újból menjen előre 2 másodpercig!

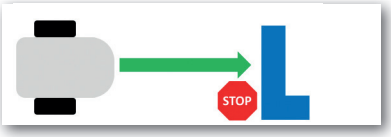


7. A robot járja be a következő síkidomoknak megfelelő útvonalakat: négyzet, téglalap, hatszög! Rögzítsünk a robothoz egy filctollat, amelynek a hegye érjen le! Terítsünk le egy nagy kartonlapot a földre! Vizsgáljuk meg, hogy pontosan milyen alakzatokat rajzol a robotunk! Visszatért-e a robot pontosan a kiindulási helyére? Mérjük meg szögmérővel, hogy azonos szögekkel fordult-e el a robot a rajzolás során? Ha nem, akkor ennek mi lehet az oka? Mi segíthetné azt, hogy pontosabb legyen a fordulási szög?

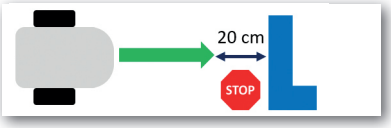


8. A robot addig haladjon előre, míg egy sötét csíkot nem érzékel! Megállás után adjon ki egy hangjelzést!

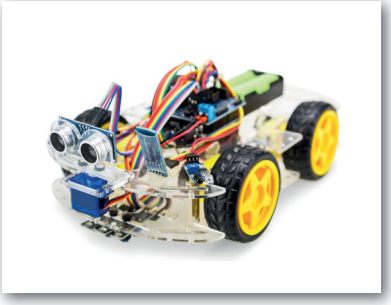
9. A robot akkor induljon el egyenes vonalban, ha a fényérzékelője erős fényt érzékel! Világítsuk meg a robotot zseblámpával vagy az okostelefon lámpájával, hogy tesztelhessük a programot!



10. Készítsünk olyan programot, amely a robotot lelassítja, majd megállítja, ha a robot akadályt érzékel maga előtt!



11. Írjunk olyan programot, amely a robotot pontosan 20 cm-rel az akadály előtt állítja le! Mit tapasztalunk? Mennyire sikerült pontosan az előírt távolságon megállni? Használjunk vonalzót a távolság méréséhez!



Alkossunk együtt!

Most már tudjuk, hogyan irányhatunk egy robot járművet. Emellett a különböző szenzorok használatába is betekintést nyertünk a gyakorlófeladatok megoldása során. Itt az ideje, hogy önállóan tervezzünk és valósítsunk meg egy programot egy gyakorlati probléma megoldására!

Alkossunk három-négy fős csoportokat! A feladatunk az lesz, hogy egy előre nem ismert terepen a leggyorsabban végigvezessük a robotjárművünket úgy, hogy az egy adott tárgy elé érkezzen, és előtte megálljon. A pályán fixen ki lesz jelölve a robot indulási pontja, és el lesz helyezve a célt jelentő akadály.

1. A robot előrehaladásának ideje/mértéke ne legyen fixen elhelyezve a programban! Irányváltást csak akkor végezhet a robot, ha a távolságérzékelője vagy a fényérzékelője (színérzékelője) egy megadott értéket ad vissza. Az irányváltás iránya viszont elhelyezhető a programban (pl. az első csík után jobbra fordul, a második csík után balra).
2. A pályán a csapatok elhelyezhetnek csíkokat, akadályokat, esetleg fényforrásokat a robot útvonalának módosításához.
3. Minden csapatnak korlátozott ideje lesz arra, hogy a pályán elhelyezze az irányításhoz szükséges tárgyakat és kipróbálja a robot működését, és esetleg módosítsa a programot. Ezért is fontos, hogy előre gondoljuk át az egyes irányítási lehetőségek előnyeit vagy hátrányait.
4. A robot elindulása és megérkezése közti időt mérjük meg stopperórával! Az a csapat nyer, amelyiknek a robotja a legrövidebb idő alatt jut el a startvonaltól a célig.
5. Beszéljük meg, hogy melyik csapat milyen megoldást választott a probléma megoldására! Melyik elgondolás bizonyult a leghatékonyabbnak? Milyen funkció vagy szenzor lenne szükséges ahhoz, hogy a problémát egyszerűbben, hatékonyabban megoldhassuk?