Bevezetés és ismétlés

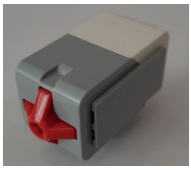
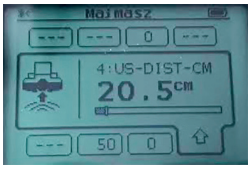
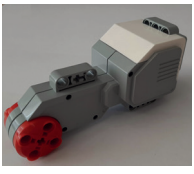
Előzetes tanulmányainkban már megismerkedtünk a micro:bit mikrovezérlővel. Megtanultuk, hogyan tudunk animációt készíteni, a LED-kijelző pontjait koordináta alapján felkapcsolni, lekapcsolni, illetve hogyan készíthetünk olyan játékot, amelyben egy szereplőt irányíthatunk gomblenyomások és gesztusok (például rázás) segítségével.

Megismerkedtünk a robotika alapjaival, és igazi robotjárműveket is programoztunk. Megtanultuk, hogyan haladhat a jármű egyenes vonalban előre és hátra, hogyan foroghat egy helyben, hogyan kanyarodhat kis ívben vagy nagy ívben. A feladatok megoldása során többféle szenzort is használtunk (fényérzékelő, távolságérzékelő, színérzékelő), és önállóan terveztünk programot valós, gyakorlati probléma megoldására.

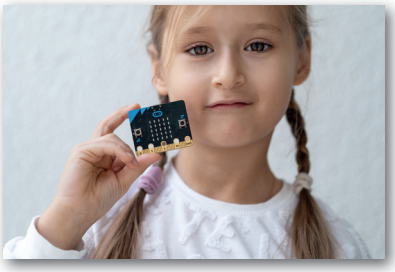
A következőkben elmélyítjük tudásunkat, és új, izgalmas lehetőségekkel is kiegészítjük.

Feladat

Párosítsuk az alábbi fogalmakat a képekkel!

1. logikai műveletek	2. ütközésérzékelő	3. motor
4. szenzor által mért érték	5. gesztusok	6. portok
7. szereplő *x* koordinátájának lekérdezése	8. ultrahangos távolságérzékelő	9. feltételes ciklus
A 	B 	C 
D 	E 	F 
G 	H 	I 

Programozzunk micro:biteket!

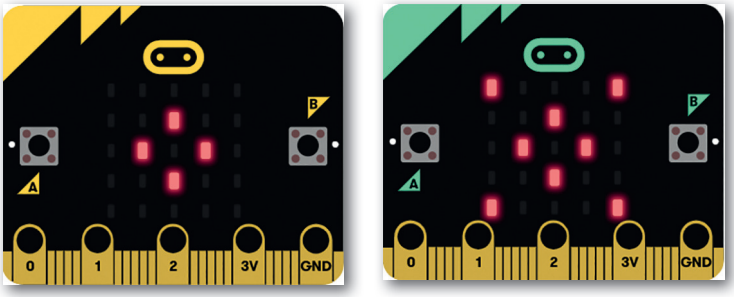


Tudjuk, hogy a micro:bit eszközt többféle blokkprogramozási környezetben is programozhatjuk. A legtöbb funkciót a *MakeCode (https://makecode.microbit.org/)* felülete biztosítja jelenleg, ezért a következőkben ennek használatával mutatjuk be az egyes példákat.

Nyissuk meg a blokkprogramozási felületet, és ismétlésként oldjuk meg az alábbi feladatokat!

Feladatok

1. Jelenítsük meg az alábbi ábrákat az *A*, illetve *B* gombok megnyomásával! A LED-eket a koordinátájuk alapján kapcsoljuk fel! Az *A* és *B* gombok együttes lenyomása után a két ábra felváltva jelenjen meg, ötször ismételve. Ne feledjük, hogy a sorok és oszlopok számozása 0-tól indul. A kijelző bal felső pontjának koordinátája tehát (0;0), jobb felső pontjának koordinátája (4;0).

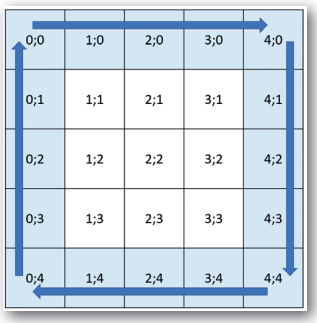


2. Készítsünk olyan programot, amely az *A* gomb lenyomásakor 25 alkalommal véletlenszerűen választ sor- és oszlopkoordinátákat, és felkapcsolja az ott található LED-eket. Az egyes pontok felkapcsolása között teljen el fél másodperc! Az eszköz megrázásakor legyen letörölve a kijelző!
3. Az alábbi események bekövetkezésekor töröljük le a kijelzőt, majd csak azok a pontok legyenek felkapcsolva, amelyek teljesítik a megadott feltételeket! A kirajzoláshoz véletlenszerűen válasszunk ki koordinátákat 100 alkalommal! Jegyezzük fel, hogy milyen ábrák jelentek meg a kijelzőn!

Esemény	**Mely pontok legyenek felkapcsolva?**
A gomb lenyomása	Az *y* koordináta 0 vagy 4, az *x* koordináta pedig tetszőleges.
B gomb lenyomása	(Az *y* koordináta 2, és az *x* koordináta tetszőleges) vagy (az *x* koordináta 2, és az *y* koordináta tetszőleges).
A + *B* gomb lenyomása	Az *y* koordináta 0 vagy 2, és az *x* koordináta 2 vagy 4.

Animáció szomszédos pontok fel- és lekapcsolásával

A LED-ek felkapcsolásában további remek lehetőségek rejlenek. Készíthetünk például figyelemfelhívó fényújságot, animációt, játékokat, megjeleníthetünk grafikonokat.



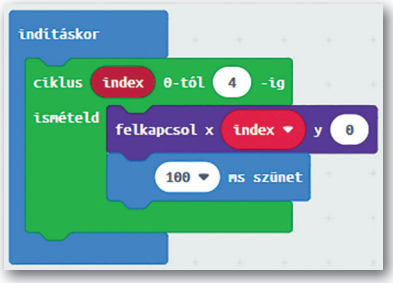
1. példa: Animáció készítése LED-ek felkapcsolásával

A következőkben egy animációt fogunk készíteni. A kijelző bal felső pontjától kezdve folyamatosan kapcsoljuk fel a LED-eket a kijelző jobb felső sarkáig, majd ezt folytassuk a jobb alsó sarokig, onnantól a bal alsó sarokig, majd felfelé, a bal felső sarokig!

Az első sor pontjainak egymás utáni felkapcsolását az alábbi kóddal elvégezhetnénk. Ennek megvalósítása azonban lassú, és a sok blokk használata miatt az esetleges hibázásnak is nagyobb esélye van.

Hogy lehetne másképp?

Láthatjuk, hogy az egyes utasítások ötször ismétlődnek úgy, hogy csak a **felkapcsol** parancs paraméterezésében van különbség.



Az első paraméter 0-tól 4-ig vesz fel értékeket, mivel fel kell kapcsolnunk a 0, 1, 2, 3, 4 sorszámú oszlopokban lévő LED-eket. A második paraméter minden esetben 0, hiszen az első sorban (melynek sorszáma 0) kapcsoljuk fel a pontokat.

A paraméterekben rejlő szabályszerűséget kihasználva a képen látható számlálós ciklus használatával a kód egyszerűbbé válik.

Próbáljuk ki a gyakorlatban a blokk hatását!

Számlálós ciklus ciklusváltozóval

Az imént látott számlálós ciklusban egy úgynevezett **ciklusváltozót** találunk, amely most az *index* nevet kapta. Ez a név azonban tetszőleges lehet, vagyis szabadon megváltoztathatjuk.

A ciklusváltozó kezdeti értéke **nulla**, és a ciklusmag minden egyes végrehajtásakor automatikusan nő az értéke eggyel.

A ciklus addig fut le, amíg a ciklusváltozó értéke kisebb vagy egyenlő, mint a paraméterként megadott szám. Példánkban ez a szám **négy**.

Vagyis a ciklus összesen **ötször fut le**, mivel a változó kezdőértéke nulla volt.

```
index=0
index=1
index=2
index=3
index=4
```

} A ciklusváltozó értéke ötször változik.

Ahogy láthatjuk, a ciklusváltozó értékére a ciklusmagon belül hivatkozhatunk. A ciklusmag első végrehajtásakor a ciklusváltozó értéke nulla, az utolsó végrehajtásakor pedig négy.

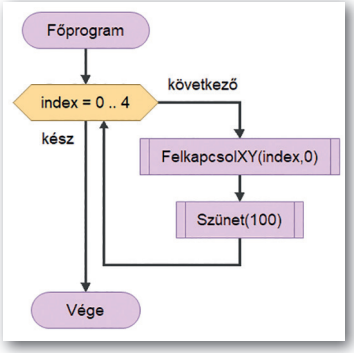
A számlálós ciklust az algoritmus szöveges megfogalmazásakor a következőképpen írhatjuk le:

```
Ciklus változóNév=kezdőérték-től végérték-ig
    utasítás(ok)
Ciklus vége
```

A példánk algoritmusának szöveges leírása és folyamatábrája a következő:

```
Program
 Ciklus index=0-tól 4-ig
    FelkapcsolXY(index,0)
    Szünet(100)
 Ciklus vége
Program vége
```

► Az első sorban lévő LED-ek felkapcsolásának algoritmusa



► Az algoritmus szemléltetése folyamatábrával

2. példa: LED-ek felkapcsolása jobbról balra

A korábbi példában balról jobbra kapcsoltuk fel a LED-eket, így az oszlopok sorszámai 0-tól 4-ig egyesével növekedtek. De mit tehetünk akkor, ha jobbról balra szeretnénk felkapcsolni a pontokat? Ekkor 4-től 0-ig kell visszaszámolnunk. Vagyis a ciklus kezdeti értéke 4 lesz, a végső érték pedig 0, és nem növelni kell a ciklusváltozó értékét eggyel, hanem csökkenteni, vagyis –1 értéket kell hozzáadni.

A feladat megoldásának algoritmusa ekkor a következő:

```
Program
 Ciklus index=4-től 0-ig -1-esével
    FelkapcsolXY(index,0)
    Szünet(100)
 Ciklus vége
Program vége
```

Vannak olyan programozási környezetek, ahol az algoritmusban látott visszaszámlálás egyszerűen megvalósítható. Későbbi tanulmányaink során használunk majd ilyeneket. A *MakeCode* programozási környezetben viszont nincs lehetőségünk annak beállítására, hogy a ciklusváltozó 4-től számoljon visszafelé 0-ig, hiszen a 0 értéket nem tudjuk a blokkban megváltoztatni. Ezért más megoldást kell találnunk. Ehhez pedig a matematikai ismereteinket kell segítségül hívnunk.

A ciklusváltozó *(index)* értéke	A felkapcsolandó pont oszlopának sorszáma
0	4
1	3
2	2
3	1
4	0

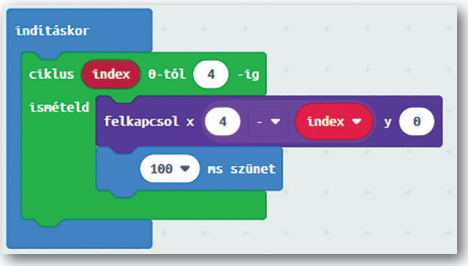
Kérdés

Melyik az a matematikai művelet, amelyet az első oszlop értékein végrehajtva a második oszlopban található számokat kapjuk meg eredményül? Sikerült-e rájönni önállóan a megoldásra?

Ha észrevesszük, hogy a táblázat soraiban lévő számokat összeadva mindig négyet kapunk eredményül, akkor már el is jutottunk a megoldáshoz. Ebből ugyanis következik, hogy a felkapcsolandó pont oszlopának sorszámát úgy kapjuk meg, hogy a négyből kivonjuk a ciklusváltozó aktuális értékét. Vagyis a pontok jobbról balra történő felkapcsolásának algoritmusa és blokkja a következő:

```
Program
 Ciklus index=0-tól 4-ig
    felkapcsolXY(4-index,0)
    szünet(100)
 Ciklus vége
Program vége
```

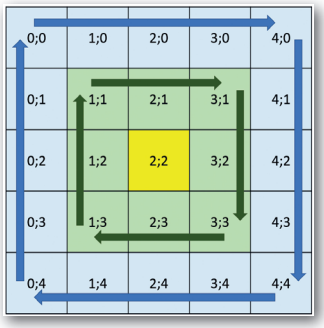
▶ A jobbról balra történő kirajzolás algoritmusa



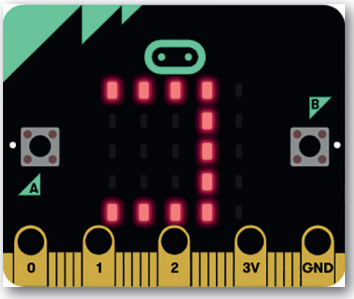
▶ A jobbról balra történő kirajzolás kódja

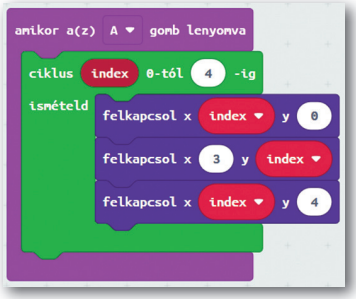
Próbáljuk ki a fenti blokk eredményét!

Feladatok



1. Fejezzük be önállóan a feladat megvalósítását az eredeti terveknek megfelelően! Oldjuk meg úgy is a feladatot, hogy a sarkokban lévő pontokat ne kapcsoljuk fel kétszer!
2. Fejlesszük tovább a feladatot úgy, hogy a belső (a képen zölddel jelölt) pontok is legyenek felkapcsolva egymás után, ugyanolyan sorrendben, mint a szélső sorok és oszlopok pontjai! Legvégül a középső pontot kapcsoljuk be!
3. Készítsünk olyan animációt ciklus segítségével, amely egy futófényt valósít meg! A kijelző középső sorában először balról jobbra, majd jobbról balra kapcsolódjanak fel a LED-ek! Az animáció öt alkalommal ismétlődjön az *A* gomb megnyomásakor!
4. Egyik barátunk az alábbi ábrát szerette volna megjeleníteni a kijelzőn. Az általa elkészített kód viszont hibás, nem ezt az ábrát rajzolja ki. Keressük meg a hibát a kódban, és javítsuk ki! Próbáljuk ki a javított kódot!
5. Készítsük el az alábbi algoritmusnak megfelelő programot, és próbáljuk ki! Milyen ábra





jelenik meg a kijelzőn? Miért? Hogyan lehetne ugyanezt az ábrát kirajzolni az *oszlop* változó használata nélkül?

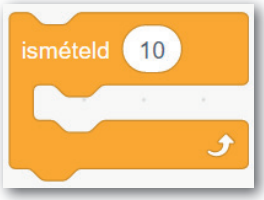
```
Program
oszlop:=0
Ciklus index=0-tól 4-ig
    felkapcsolXY(oszlop,index)
    oszlop:=oszlop+1
Ciklus vége
Program vége
```

Magyarázat:
Az `oszlop:=0` jelzi, hogy az *oszlop* változó értéke nulla lesz.
Az `oszlop:=oszlop+1` azt jelenti, hogy az *oszlop* változó értéke eggyel növekszik.

Számlálós ciklus ciklusváltozó nélkül

Érdemes tudnunk, hogy nem mindegyik blokkprogramozási környezetben használhatunk olyan számlálós ciklust, amely ciklusváltozóval rendelkezik. Sok esetben csak arra van lehetőségünk, hogy megadjuk az ismétlések számát. Erre láthatunk példát itt is.

Ebben az esetben nekünk kell gondoskodnunk a változó létrehozásáról, kezdő értékének beállításáról és értékének növeléséről a ciklusmagon belül. Nézzük a következő példát!

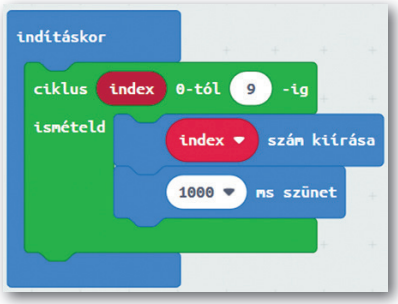


▶ Számlálós ciklus a Scratch környezetben

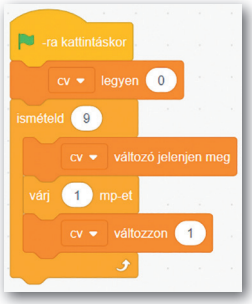
3. példa: Program egész számokra 0-tól 9-ig

Írjunk egy programot, amely 0-tól 9-ig megjeleníti az egész számokat! Minden kiírás között 1 másodperc teljen el!

A megoldást két programozási környezetben is elkészítettük:



▶ 1. megoldás a MakeCode környezetben



▶ 2. megoldás a Scratch környezetben

Az első megoldásban olyan számlálós ciklust használtunk, amely beépített ciklusváltozót tartalmaz. A második megoldásban egy egyszerűbb számlálós ciklust használtunk, amelynél csak az ismétlések számát tudjuk beállítani. Láthatjuk, hogy még a ciklus előtt beállítottuk a ciklusváltozó *(cv)* értékét nullára. A ciklusmagban pedig a ciklusváltozó értékét eggyel növeljük a megfelelő blokk használatával. Így a két példa ugyanazt az eredményt adja.

Feladatok

1. Ha az lenne a feladatunk, hogy 5 és 14 közötti számokat jelenítsünk meg, akkor hogyan kellene megváltoztatni az első, illetve második példában látott programokat?
2. Hogyan érdemes módosítani a programot, ha 2 és 20 közötti páros számokat szeretnénk megjeleníteni?
3. Tapasztalataink alapján fogalmazzuk meg, hogy mikor lehet előnyösebb a második példában használt ciklus használata az első példában láthatóhoz képest!

4. példa: Egymásba ágyazott ciklusok

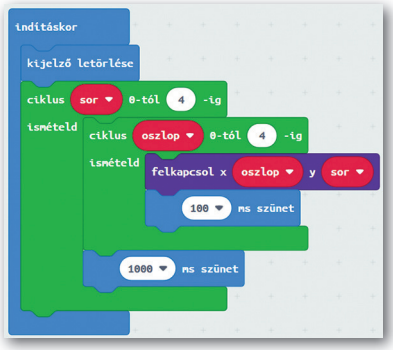
A ciklusokat akár egymásba is ágyazhatjuk. Nézzünk erre egy példát!

Készítsünk olyan programot, amely a micro:bit kijelzőjén lévő pontokat soronként, balról jobbra kapcsolja fel! A pontok felkapcsolása között 100 ms idő teljen el! Az egyes sorok kirajzolása között 1 másodperc szünet legyen!

Ebben az esetben először a nulladik sorszámú sorban kell kirajzolnunk a pontokat, aztán az egyes sorszámúban, egészen a négyes sorszámú sorig.

Egy soron belül a pontokat balról jobbra, vagyis a nulladik sorszámú ponttól a négyes sorszámú pontig kell bekapcsolnunk.

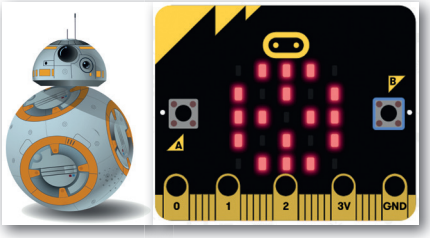
Ezért érdemes a két ciklust egymásba ágyazni, az itt látható módon.

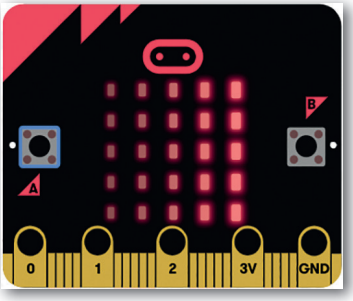


▶ Ciklusok egymásba ágyazása. Próbáljuk ki a kódot!

Feladatok

1. Hogyan változik a kirajzolás akkor, ha a külső ciklus változóját megváltoztatjuk az *oszlop* változóra, a belső ciklusét pedig a *sor* változóra? Próbáljuk ki!
2. Készítsünk olyan programot, amely az *A* gomb megnyomásakor megjelenít egy ikont a kijelzőn! A *B* gomb megnyomásakor soronként, balról jobbra minden pont állapota változzon meg az ellenkezőjére, vagyis ha fel volt kapcsolva, akkor legyen lekapcsolva, ha le volt kapcsolva, legyen felkapcsolva! Ehhez használjuk a *LED* kategória *átvált* nevű blokkját!
3. Az itt látható ábrát (amely egy a mozifilmekből jól ismert robothoz hasonlít) az előző programmal rajzoltuk meg. Mi volt ebben az esetben a kiinduló ikon? Ellenőrizzük a feltevésünket a gyakorlatban is!
4. A *LED* kategóriában egy másik izgalmas blokkot is találunk, amellyel a pont *fényerejét* is be tudjuk állítani.





A paraméter 0 és 255 közötti érték lehet. A 0 azt jelenti, hogy teljesen sötét a pont, a 255 pedig azt, hogy teljesen világos. Módosítsuk úgy az eredeti programot, hogy a pontok fényereje balról jobbra növekedjen!

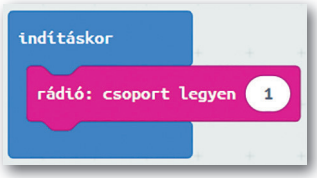
Rádiókapcsolat és fényerősségmérés

Rádiókapcsolat a micro:bitek között

A micro:bitek további izgalmas lehetőségeket is tartogatnak számunkra. Ilyen például az is, hogy ezek az eszközök képesek egymással kommunikálni. Így adatokat is átküldhetünk egyik eszközről a másikra, vagy akár többfelhasználós játékokat is készíthetünk. Sőt, az animációk területén is kihasználhatjuk ezt a funkciót. Nézzünk erre néhány példát!

A rádiókapcsolat beállítását végző blokkokat a *Rádió* kategóriában találjuk.

Két vagy több micro:bit akkor tud kommunikálni egymással, ha azonos rádiócsoportba tartoznak, amelyet egy számmal jelölünk. A csoport számát az itt látható blokk segítségével tudjuk beállítani.

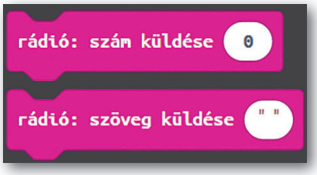


▶ Rádiócsoport beállítása

Ha például egyszerre 20 micro:bitet használunk egy teremben, és pármunkában szeretnénk kipróbálni a rádiókapcsolatot, akkor párosával kell azonos csoportba sorolni az eszközöket.

Természetesen ebben az esetben meg kell állapodnunk az osztálytársainkkal abban, hogy ki melyik csoportszámot fogja beállítani a saját eszközén.

Miután beállítottuk a csoportszámot, már küldhetünk adatokat a micro:bitek között. Szám, illetve szöveg küldéséhez az itt látható blokkokat használhatjuk.



▶ Szám, illetve szöveg küldése

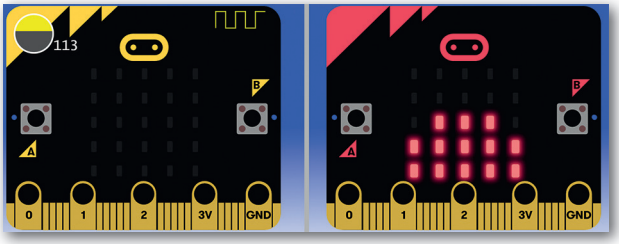
Amikor a micro:bit érzékeli, hogy számot vagy szöveget küldtek neki, akkor különböző utasításokat tud végrehajtani. Ehhez a képen látható blokkok állnak rendelkezésre. Az első blokkot akkor használjuk, amikor a kapott számot *(receivedNumber)*, a másodikat pedig akkor, ha a kapott szöveget *(receivedString)* szeretnénk feldolgozni.

▶ Rádiós adat vételekor használható blokkok

5. példa: Micro:bitek kommunikációja egymás között

A rövid bevezető után nézzünk egy példát! Készítsünk olyan programot, amelyben két micro:bit kommunikál egymással! Az első micro:bitet nevezzük *Robi*nak. Ő lesz az adó, amely adatokat küld a másik eszköznek. A második micro:bit lesz a vevő, amely a kapott adatokat feldolgozza. Az ő neve legyen *Bori*.

De milyen adatokat küldjön át *Robi* *Bori* részére? Legyen ez például az a fényerősségérték, amit éppen érzékel. *Bori* pedig jelenítse meg grafikonon a kapott adatot.



▶ Robi az általa érzékelt fényerősségértéket Boriak küldi át, aki azt megjeleníti grafikonon

Összehangolt animáció több micro:biten

A rádiókapcsolat arra is lehetőséget nyújt, hogy egy micro:bit segítségével több más micro:biten egyszerre indítsuk el ugyanazt vagy esetleg más-más animációt. Sőt, az is lehetséges, hogy amikor az egyik eszköz befejezte az animációt, értesíti erről a következő micro:bitet, amely elkezdi a saját animációját, és így tovább.

Feladat

Alkossunk 4-5 főből álló csoportokat! Tervezzünk meg egy animációt, amelyben minden micro:bitnek jut feladat, és a rádiókapcsolat segítségével kommunikálnak egymással! Készítsük el a megfelelő programokat minden micro:bitre, és próbáljuk ki a gyakorlatban is!

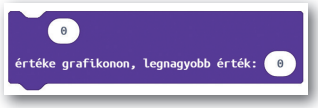
Fényerősség érzékelése és megjelenítése grafikonon

A micro:bit LED-kijelzője egyben fényerősség érzékelésére is használható. Ez a szenzor 0 és 255 közötti értéket ad vissza. A 0 a teljes sötétségnek felel meg, míg a 255 a maximális világosságnak.

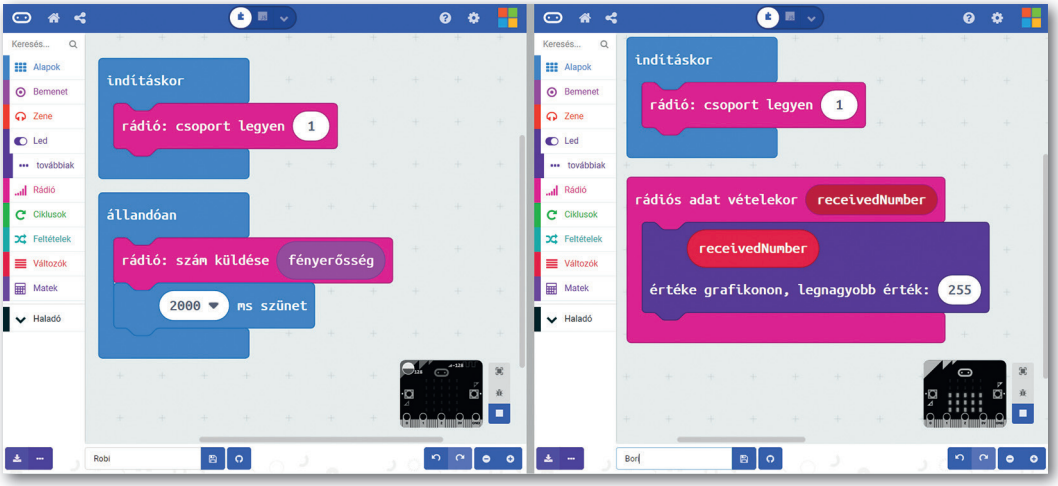
Egy számérték grafikonon való megjelenítésére használhatjuk a képen látható blokkot. Az első paraméternek azt a számot kell beállítani, amelyet ábrázolni szeretnénk. A második paraméter az a legnagyobb érték, amelyhez képest megjelenítjük az értéket.

Robi és *Bori* tehát az alábbi programokat fogja végrehajtani:

▶ A Bemenet kategória Fényerősség blokkja



▶ A LED kategória grafikonábrázolási blokkja



▶ Robi és Bori programjai

A rádiókapcsolatot használó projekt esetén érdemes a *https://makecode.com/multi/* címen elérhető környezetet használnunk. Ahogy a képen is láthatjuk, ilyenkor két részre oszlik a képernyő, így külön-külön létrehozhatjuk az adó és a vevő programját is, sőt ki is próbálhatjuk a szimulátorban.

A fényerősséget a szimulátorban az ikon sárga színű területének megragadásával módosíthatjuk.

Feladatok

1. Próbáljuk ki a fenti programokat először szimulátor segítségével, majd töltsük le a programokat az eszközökre is! Takarjuk el kezünkkel teljesen a LED-kijelzőt, majd ezután világítsuk meg erős fénnyel! Hogyan változik ennek hatására a grafikon a másik eszközön? Milyen körülményeket kell teremteni ahhoz, hogy *Bori* kijelzőjén a grafikon maximális 25 pontjából körülbelül a fele világítson?
2. Képzeljük el, hogy nemsokára egy szabadulószobába leszünk bezárva! Az ajtó csak kívülről nyitható, egy háromjegyű kód megadásával. A szobában ott lesz velünk a *Robi* nevű micro:bit és egy zseblámpa. A szobában egyedül meg kell fejtenünk, hogy mi a zár kódja, és azt a társunknak kell továbbítanunk, hogy kinyithassa a lakatot. Ötleteljünk arról, hogyan lehetne *Robi* és *Bori* segítségével megoldani ezt a kihívást! Játsszuk el ezt a szituációt a micro:bitek segítségével!

Játék a micro:bittel

Riasztóberendezés kerékpárra

Pármunkában tervezzünk meg egy speciális riasztórendszert, amellyel a kerékpárunkat védhetjük meg az illetéktelen használóktól!

- A riasztórendszer két micro:bitből álljon! Az egyiket a kerékpárra fogjuk rögzíteni, ennek neve legyen *Szergiusz!* Ez egy latin eredetű név, amely őrzőt jelent.
- A másik micro:bit neve legyen *Alexia,* amely görög–latin eredetű név, és védőt jelent. *Alexia* feladata, hogy távolról be lehessen kapcsolni, illetve ki lehessen kapcsolni a riasztót.
- *Szergiusz* kijelzőjén legyen látható, hogy éppen élesített (bekapcsolt) vagy kikapcsolt állapotban van-e! *Alexia* kijelzőjén gombnyomással lehessen megjeleníteni azt, hogy *Szergiusz* milyen (bekapcsolt vagy kikapcsolt) állapotban van!
- Amennyiben *Szergiusz* bekapcsolt állapotban van, és mozgást érzékel, akkor a kijelzőjén jelenjen meg egy figyelmeztető ikon vagy animáció, és adjon ki szirénázásra emlékeztető hangot! Riasztás közben küldjön üzenetet *Alexiának,* melynek kijelzőjén jelenjen meg egy figyelmeztető ikon/animáció, és szintén játsszon le egy hangot!
  Van esetleg más hasznos funkció, amelyet érdemes lenne ehhez hasonlóan programozni? Valósítsuk meg azt is!

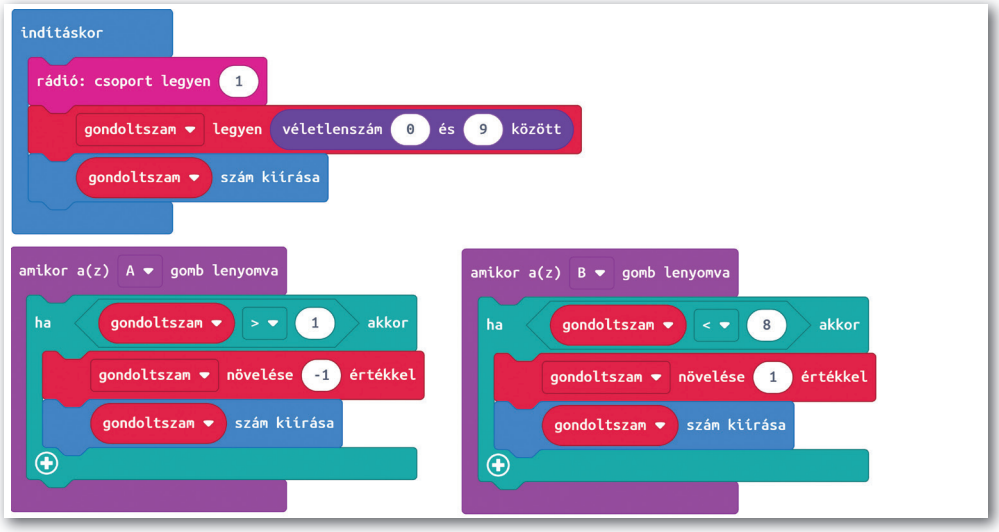
Számkitalálós játék

A következőkben egy egyszerű, párokban játszható számkitalálós játékot valósítunk meg. A játékban két micro:bit vesz részt. Az egyik neve *gondolkodó,* a másiké *gondolatolvasó*. A cél, hogy helyesen tippeljünk meg egy 0 és 9 között kigondolt számot. Ha sikerült, akkor a *gondolatolvasó* nyer, ha nem, akkor a *gondolkodó*.

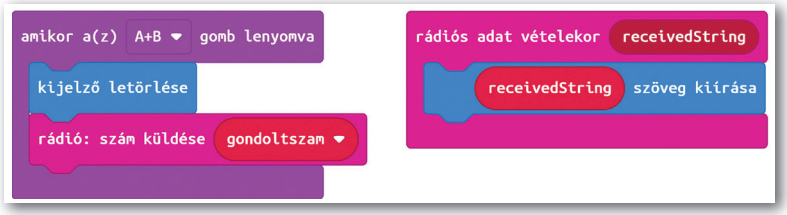
A gondolkodó működése

A *gondolkodó* nevű micro:biten kezdetben egy 0 és 9 közti véletlenszám jelenjen meg a kijelzőn! Az *A* gomb megnyomásával eggyel lehessen csökkenteni a szám értékét, a *B* gomb hatására pedig eggyel növelni! Ügyelnünk kell arra is, hogy a szám ne lehessen 0-nál ki-

sebb, illetve 9-nél nagyobb. Ezért csak abban az esetben csökkentjük az értéket, ha 1-nél nagyobb, és csak akkor növeljük, ha 8-nál kisebb.



A kiválasztott számot az *A* + *B* gombok együttes megnyomásával lehessen véglegesíteni! Ekkor a *gondolatolvasó* micro:bit részére rádiókapcsolaton keresztül át kell küldeni a gondolt számot. A *gondolatolvasó* egy üzenetben vissza fogja küldeni, hogy sikerült-e kitalálnia



a számot. Ezt az üzenetet jelenítsük meg a kijelzőn!

A gondolatolvasó működése

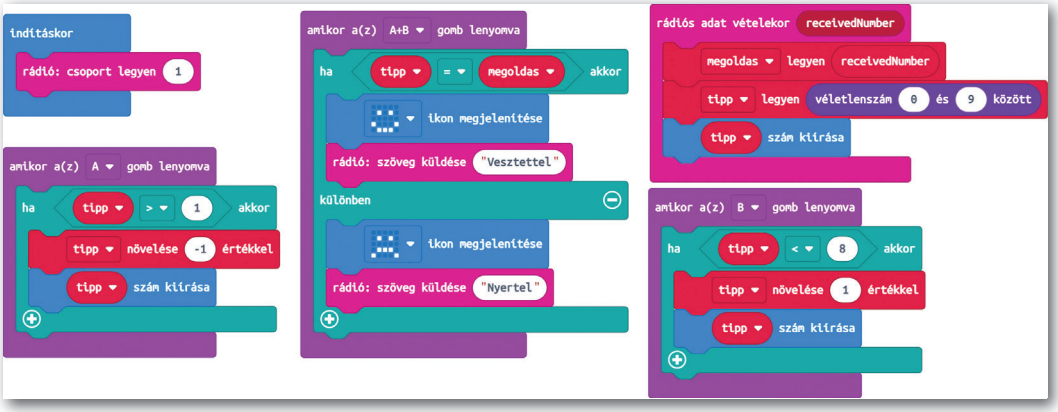
A *gondolatolvasó* micro:biten is ugyanúgy kelljen kiválasztani a számot, mint *gondolkodó* esetén!

A *gondolkodó* által átküldött számot tároljuk el a *megoldas* nevű változóba!

Az *A* + *B* gombok lenyomásakor ellenőrizzük, hogy a tipp megegyezik-e a megoldással! Ha igen, egy vigyorgó fejet, helytelen tipp esetén egy szomorú fejet jelenítsünk meg! A játék végeredménye így már megjelenik ezen a micro:biten.

Ahhoz, hogy a *gondolkodó* micro:bit is értesüljön az eredményről, küldjük el neki a *„Vesztettél”*, illetve *„Nyertél”* szövegeket attól függően, hogy sikerült-e kitalálnia a *gondolatolvasónak* a számot, vagy sem!

Megjegyzés: a magyar ékezetes betűk megjelenítésére nem áll rendelkezésre elég képpont a micro:bit kijelzőjén, ezért a kiírásnál írhatjuk a szöveget ékezetek nélkül is! Például a *„Vesztettél”* helyett a *„Vesztettel”* szöveget jelenítsük meg!



Feladat

Próbáljuk ki a fenti játékot szimulátorban és a micro:bitre letöltve, párokban is! Tíz játékból hányszor nyert a *gondolatolvasó,* illetve a *gondolkodó?*

Gyakorlás, saját ötletek megvalósítása

Most már tudjuk, hogyan teremthetünk rádiókapcsolatot a micro:bitek között. A következőkben engedjük szabadon a képzeletünket, és fejlesszük tovább a megismert alkalmazásokat, vagy találjunk ki saját játékokat!

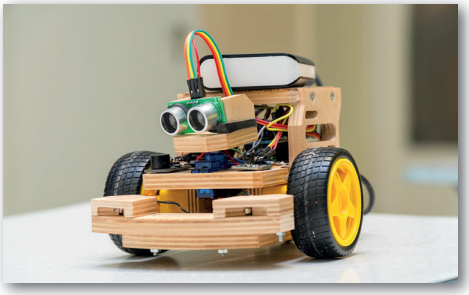
Egyéni feladatok

Gondoljuk át, hogyan tudnánk még izgalmasabbá tenni a számkitalálós játékot! Hogyan lehetne megoldani például, hogy a *gondolatolvasó* valamiképpen segítséget kapjon a gondolt számra vonatkozóan, de úgy, hogy azt mások ne vehessék könnyen észre? Mutassuk be megoldásainkat a többieknek, hátha elhiszik, hogy tényleg gondolatolvasó képességekkel rendelkezünk!

Találjunk ki saját alkalmazást vagy játékot!

1. Alkossunk három-négy fős csoportokat! Tervezzünk olyan alkalmazást, amelyben a micro:bitek egymással kommunikálnak! Ez lehet játék, egy hasznos alkalmazás (mint a korábbi riasztóberendezés) vagy akár egy képzeletbeli szabadulószoba feladványához kapcsolódó alkalmazás. Ötleteinket valósítsuk meg!
2. Ha elkészültünk a munkánkkal, mutassuk be egymásnak az alkalmazásokat, és próbáljuk ki egymás programjait!
3. Gyűjtsük össze, hogy mi tetszett az egyes alkalmazásokban, mi okozott nehézséget a használat során, és hogy milyen módon lehetne még továbbfejleszteni a programokat!

Robotika (ismétlés)

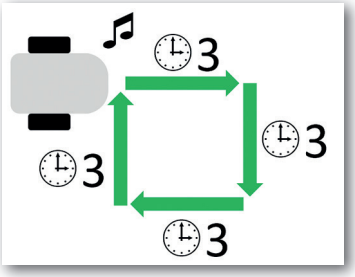


Hamarosan újabb és újabb izgalmas feladatokat oldunk meg a rendelkezésünkre álló robotjárművek segítségével. Előtte viszont ismételjük át azt, hogy hogyan irányíthatjuk a járművet!

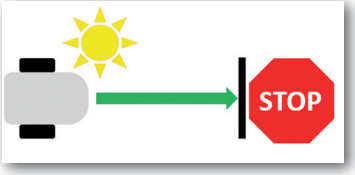
Csoportmunkában oldjuk meg a következő feladatokat!

Csoportos feladatok

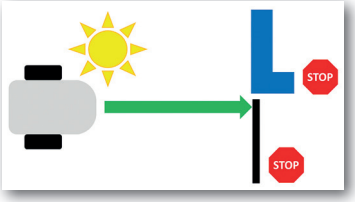
1. A robot indulás előtt adjon ki egy hangot! Ezután négyszer ismételje a következőket:
   a) haladjon előre egyenes vonalban 3 másodpercig,
   b) majd forduljon jobbra 90°-ot!
   Megálláskor szintén szólaltasson meg egy hangot!



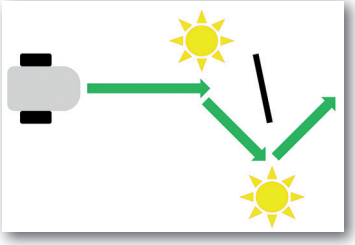
2. Ragasszunk le a padlóra (vagy más biztonságosan használható sík felületre) egy sötét csíkot! Helyezzük el tőle körülbelül fél méterre a robotot!
   Írjunk olyan programot, amely erős fény hatására elindítja a robotot előre, majd automatikusan leállítja azt, ha sötét csíkot érzékel!



3. Módosítsuk úgy az előző programot, hogy ne csak a csíkot érzékelje a jármű, hanem akkor is álljon meg, ha akadályt érzékel maga előtt!
   Helyezzünk el a csík előtt egy akadályt, és próbáljuk ki, hogy helyesen működik-e a programunk!
4. Készítsünk olyan programot, amelyben a robot előrehalad, és felváltva jobbra, illetve balra kanyarodik abban az esetben, ha erős fényt érzékel!
   Ragasszunk le egy csíkot a pályára, helyezzük el előtte a robotot!
   Világítsuk meg erős fénnyel a robotot úgy, hogy az kikerülje a leragasztott csíkot!



5. Előzetes tanulmányainkból tudjuk, hogy a vezérlés azt jelenti, hogy a robot pontosan az általunk kiadott utasításokat hajtja végre, nem vizsgálja a környezetét, nem hajt végre korrekciót a mért adatok alapján. Adjuk meg, hogy a fenti feladatok közül melyekre igaz az, hogy vezéreltük a robotot!



Feladatok megoldása virtuális robotjárművel

A valós robotjárművekkel való munka nagyon izgalmas, már csak azért is, mert a robot reagálhat a környezetének jellemzőire, az elhelyezett akadályokra, a színes csíkokra, a fényviszonyok változására és így tovább.

A programok tesztelése viszont hosszadalmas lehet, mivel minden módosítás után újra fel kell töltenünk a programot a robotra, és a futtatás során meg kell vizsgálnunk azt is, hogy a módosított program megfelel-e az elvárásainknak.

Az is előfordulhat, hogy szívesen dolgoznánk egy probléma megoldásán otthon (valós robot nélkül), majd ha elkészültünk a programmal, azt valós robottal is kipróbálnánk. Sőt, lehetnek olyan feladatok is, amelyeket a valós robottal nem tudunk megoldani, a szimulált környezet viszont lehetőséget biztosít erre.

Ebben segítenek minket azon környezetek, amelyekben a robotjárműveket szimulációs környezetben használhatjuk. A következőkben egy ilyen felülettel ismerkedünk meg.

A VEX VR felület használata

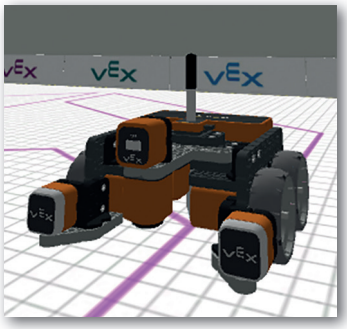
A *https://vr.vex.com/* webcímen egy olyan blokkprogramozási környezetet találunk, amelyben egy robotjárművet programozhatunk. Tanulmányaink során már többféle blokkprogramozási környezetet használtunk, így ezen a felületen is hamar ki fogjuk ismerni magunkat.

A virtuális robotjármű előre, illetve lefelé néző távolságérzékelővel és színérzékelővel, fordulásérzékelővel, helymeghatározóval, illetve bal és jobb oldali ütközésérzékelő szenzorokkal van felszerelve.

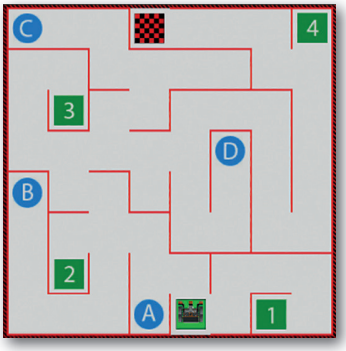
Emellett képes arra, hogy a tollát letegye, illetve felemelje. Így a bejárt útvonalat is megrajzolhatjuk a segítségével.

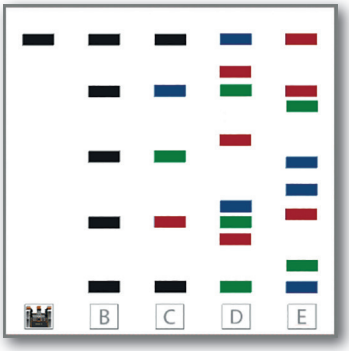
A robot rendelkezik egy elektromágnessel is, melynek segítségével a pályán elhelyezett fémkorongokat fel tudja emelni, el tudja szállítani egy másik helyre, és ott le tudja tenni.

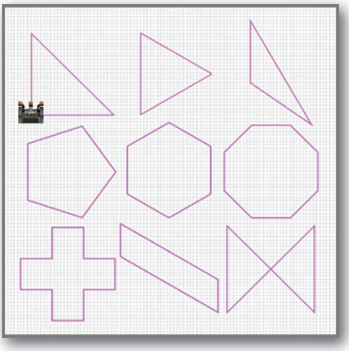
A jármű többféle színtéren (úgynevezett játszótéren) is elhelyezhető.



► A virtuális robotjármű képe

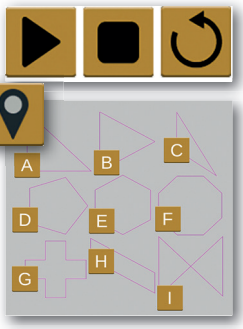






► Néhány választható játszótér (Fallabirintus, Vonalérzékelő, Alakkövető)

Program futtatása, nézetek kiválasztása



A programot elindíthatjuk, megállíthatjuk, illetve újrakezdhetjük. Utóbbi esetben a robot a kiindulási helyzetébe kerül.

Elképzelhető, hogy az adott pályán (játszótéren) több kiindulási pozíción is elhelyezhető a robot. Ekkor megjelenik a pozícióválasztó ikon, amelyet megnyomva kiválaszthatjuk a kezdőpontot.

A kezdő pozíciók betűkkel vannak ellátva, amelyekre kattintva a robot az adott helyre kerül, és ott kezdi a programjának végrehajtását.

A jármű haladását többféle nézetben is nyomon követhetjük. Használhatunk felülnézeti képet, háromdimenziós nézetet és olyan nézetet is, mintha mi magunk ülnénk a járműben, és előrenéznénk.

A nézetek váltásához a különböző irányokba néző kamera ikonokat használhatjuk.

A műszerfal használata

A valós járművekben is van olyan műszerfal, amelyről leolvashatjuk a jármű állapotára vonatkozó adatokat. Ilyen például a sebesség, a fordulatszám és az üzemanyagszint. A virtuális robotunk is rendelkezik műszerfallal, amelyet a *Műszerfal* ikon megnyomásával jeleníthetünk meg.

Heading	Rotation	Front Eye	Down Eye	Location	Location Angle	Bumper	Distance
267°	87°	Object: True Color: None	Object: False Color: None	X: -910 mm Y: -266 mm	267°	Left: False Right: False	35 mm

► A megjelenő műszerfal

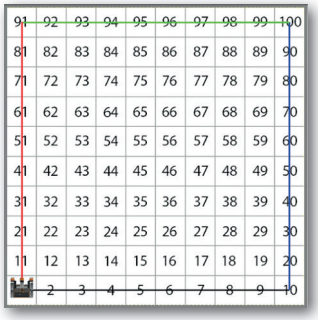
A műszerfalról a következő értékeket olvashatjuk le:

- **Irány** *(heading)*: Ez a szög mutatja, hogy aktuálisan mi az iránya a robotnak.
- **Fordulás** *(rotation)*: Azt mutatja meg, hogy a program futtatása során összesen mekkora szögértékkel fordult el a robot. Ez pozitív, ha a robot többet fordult el az óramutató járásával megegyező irányban, mint ellenkező irányban. Ellenkező esetben negatív.
- **Előrenéző szem** *(front eye)*: Itt láthatjuk, hogy az előrenéző szem érzékel-e maga előtt tárgyat *(object)*. Amennyiben igen, akkor az igaz *(true)* érték jelenik meg, ellenkező esetben a hamis *(false)*. Amennyiben színt *(color)* érzékel a szenzor, akkor a szín nevét is látjuk angol nyelven.
- **Lefelé néző szem** *(down eye)*: Ugyanazon értékeket látjuk, mint az előbb, csak a lefelé néző szem esetén.
- **Pozíció** *(location)*: Itt látható a robot aktuális *x* és *y* koordinátája.
- **Pozíciószög** *(location angle)*: A programban beállíthatjuk, hogy mi legyen a robot alapértelmezett menetiránya. Alapesetben ez a 0°. Ha megváltoztatjuk, akkor az irány *(heading)* szögét ehhez az alapértelmezett szöghöz képest mutatja. Ritkán van arra szükség, hogy ezt a szöget megváltoztassuk.

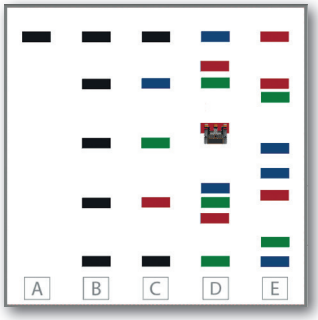
- **Ütköző** *(bumper)*: Az ütközésérzékelő állapotát jelzi. A bal *(left)* és jobb *(right)* érzékelő esetén külön látjuk, hogy igaz-e, hogy le van nyomva *(true)*, vagy nem *(false)*.
- **Távolság** *(distance)*: Ez az érték azt mutatja, hogy mekkora távolságra van akadály a robot előtt.

Feladatok szimulátor segítségével

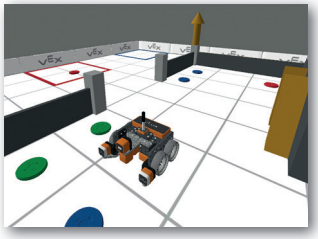
Oldjuk meg az alábbi feladatokat! Keressük meg önállóan a megoldáshoz szükséges blokkokat, és teszteljük az eredményt a szimulátorban! A megoldásokat minden esetben mentsük el a saját gépünkre vagy külső adathordozóra!



1. Válasszuk ki azt a játszóteret, amelyben a rács cellái sorszámozva vannak!
   Rajzoljuk meg a képen látható ábrát a robot tollának leengedésével és színének megváltoztatásával!
   Próbáljuk ki a különböző nézeteket a robot haladása közben!
   Jegyezzük fel, hogy milyen koordinátán volt a robot a kiindulási helyzetében, illetve a többi három sarokban! Hányas számú cellába kerülne a robot, ha a [700;-100] koordinátára vezetnénk el?



2. Válasszuk ki azt a pályát, amelyen különböző színű csíkok vannak elhelyezve! A robot kiindulása helyzete legyen a D jelű cella!
   Készítsünk olyan programot, amelyben a robot addig megy előre, míg a lefelé néző szeme a kék csíkot nem érzékeli! Megállás után várjon 2 másodpercet, majd folytassa útját addig, míg nem érzékeli a piros csíkot! Ekkor álljon meg a robot!



3. Válasszuk ki azt a pályát, amelyben egy színes korongokkal körbevett kastélyt látunk!
   Mozgassunk át minden négyzetbe egy, a négyzet színével megegyező színű korongot!
   Ügyeljünk arra, hogy a pályán a korongok a cellák közepén vannak elhelyezve, az elektromágnes viszont a robot elejére van felszerelve, ezért megfelelő pozícióra kell állnunk ahhoz, hogy fel tudjuk venni a korongot.
   A robot egyszerre csak egy korongot tud szállítani.

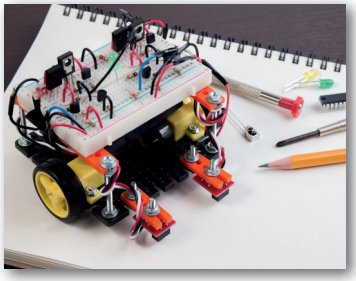
Útvonalkövetés valós robotok segítségével

A korszerű raktárakban, összeszerelő üzemekben sok esetben már szorgos robotok szállítják az árut, illetve az alkatrészeket egyik helyről a másikra.

Azt, hogy a robotoknak milyen útvonalat kell bejárniuk, jelezhetik a padlóra festett különböző színű csíkok, sávok, de más megoldások is léteznek erre. Az sem biztos, hogy az útvonalak az emberi szem által látható módon vannak kialakítva.

De hogyan lehet megtanítani a robotoknak, hogy egy vonalat kövessenek? Azt gondolhatnánk, hogy erre csak a programozók, illetve mérnökök képesek. Dehogy! Kis gondolkodással és megfelelő eszközöket használva mi is megoldhatjuk ezt a problémát.

Hogyan épülhet fel egy vonalkövető robot?



Egy valódi, vonalkövetési feladatra alkalmas robot többféleképpen is megépíthető. A mozgáshoz szükség van kerekekre és az azokat meghajtó motorokra. A járműre érzékelőket is kell szerelni, amelyek lehetnek színérzékelők vagy a visszavert fény erősségét érzékelő szenzorok is.

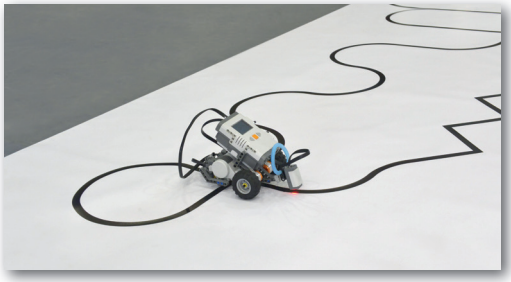
Vannak olyan robotok, amelyek csak egy érzékelővel vannak ellátva, de lehetnek olyanok is, amelyeken több érzékelő található. Sőt ma már az sem ritka, hogy egy kamera van a robotra szerelve.

Természetesen szükség van egy olyan alkatrészre is, amely a robot „agyaként” viselkedik, vagyis képes arra, hogy a szenzorok adatait feldolgozza, végrehajtsa a vonalkövetéshez szükséges programot, be- és kikapcsolja a motorokat, vagy a kamera képét elemezze.

A pálya

Nagyon fontos, hogy a csík, amelyet követni kell, jól érzékelhető legyen a robot számára. Világos felületen sötét (például fekete), sötét felületen világos (például fehér) csíkot érdemes elhelyezni.

A vonal lehet egy, a padlóra szigetelőszalaggal felragasztott csík is, mivel az könnyen eltávolítható, és újabb útvonalak is könnyedén készíthetők.



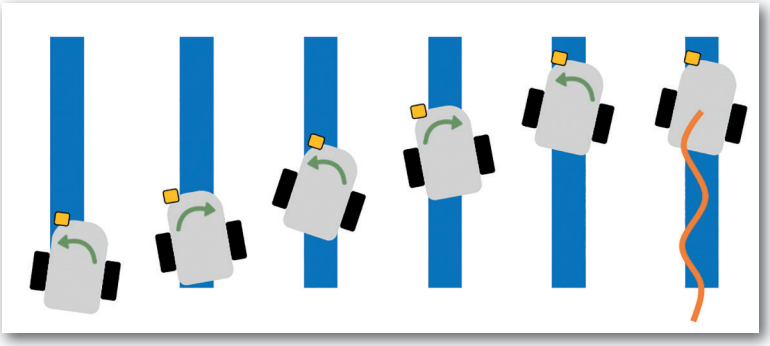
Vonalkövetés egy, illetve több érzékelővel

A következőkben több vonalkövetési megoldással is megismerkedünk.

Vonalkövetés egy érzékelővel

Ha csak egy darab fény-/világosság- vagy színérzékelő áll rendelkezésre, akkor alkalmazhatjuk az alábbi módszert. Helyezzük el a szenzort a jármű bal oldalán úgy, hogy lefelé nézzen! A járművet kissé fordítsuk jobbra úgy, hogy az érzékelő a csík felett legyen!

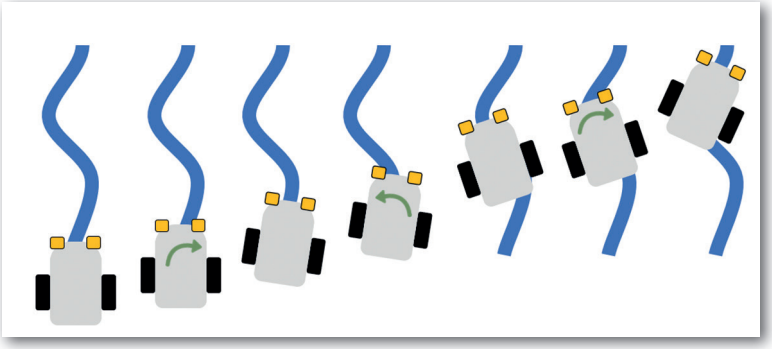
Ha a jármű előrehaladása során a szenzor a sötét színt érzékeli, akkor balra kell kanyarodnia, ha pedig a világosat, akkor jobbra. Ezzel a jármű a csík bal oldali szélét próbálja követni. Ahogy az ábrán is látszik, a robot cikcakkos (vagy kígyózó) mozgást fog végezni, még egy egyenes útvonal követése során is.



► Egy érzékelővel felszerelt jármű cikcakkos (kígyózó) mozgása

Vonalkövetés két érzékelővel

Ha van több fény-/világosság- vagy színérzékelőnk is, akkor akár kettőt is felszerelhetünk a jármű elejére. Ebben az esetben, **ha a bal oldali szenzor érzékeli** a sötét csíkot, az azt jelenti, hogy az útvonal balra kanyarodik. Emiatt a robotjárműnek is balra kell kanyarodnia. Hasonlóan kell végrehajtani a jobbra kanyarodást is, vagyis **ha a jobb oldali szenzor érzékeli** a csíkot, akkor jobbra kell kanyarodnia a robotnak.



► Két érzékelővel ellátott robot haladása

A vonalkövetés egy lehetséges algoritmusa

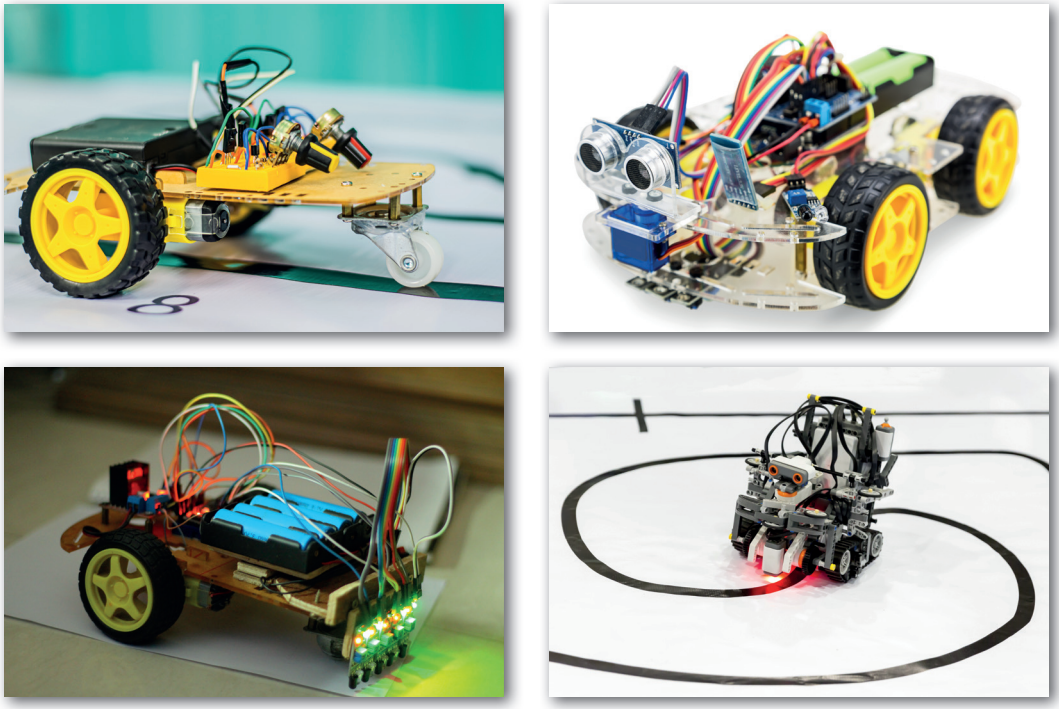
Ezek után lássuk a vonalkövetés egy algoritmusát arra az esetre, ha a robot két fény-/világosság- vagy színérzékelővel van felszerelve. A vonalkövetés egy lehetséges algoritmusa:

```
ismételd
    menj előre 2 lépést
    ha a jobb oldali érzékelő az útvonal színét érzékeli
          fordulj a csík mentén jobbra
    elágazás vége
    ha a bal oldali érzékelő az útvonal színét érzékeli
          fordulj a csík mentén balra
    elágazás vége
    ha elérte a kijáratot
          álljon le a program
    elágazás vége
ismétlés vége
```

Magyarázat: Láthatjuk, hogy a ciklusban nincs megadva az ismétlések száma, illetve a leállási feltétel, vagyis egy végtelen ciklust fogalmaztunk meg.

A robot minden lépésben előrelép, és megvizsgálja, hogy el kell-e kanyarodnia valamelyik irányba a csík mentén.

A program futása akkor áll le, ha a robot eljutott a kijáratig.



▶ Vonalkövető robotok

Feladatok

1. Alakítsunk 2–4 fős csoportokat! Építsünk olyan robotot a rendelkezésre álló eszközök felhasználásával, amely képes vonalkövetési feladatok megoldására! Amennyiben szükséges, módosítsuk a korábban látott algoritmust, majd készítsük el a vonalkövető programot!
2. Alakítsunk ki egy körpályát, például szigetelőszalagok padlóra ragasztásával, vagy használhatunk akár olyan asztalterítőt is, amelyre filccel rajzoltuk meg a pályát. Ügyeljünk arra, hogy a felület ne legyen csúszós, mert akkor a robot nem tud megfelelően haladni rajta! Indítsuk el a járművet, és mérjük meg, hogy mennyi idő alatt tesz meg egy kört! Módosítsuk úgy a programot, hogy a jármű minél gyorsabban teljesítse a távot!
3. Hogyan kell módosítani az algoritmust akkor, ha csak egy érzékelő van a jármű bal oldalára felszerelve? Próbáljuk ki az elgondolásunkat a gyakorlatban is! Mit tapasztalunk, melyik megoldással ér gyorsabban körbe a pályán a robot? Mekkora különbség volt a megtett időben az egy, illetve két érzékelővel felszerelt jármű között?
4. A vonalkövetési algoritmusok működését akár valós robotok nélkül, blokkprogramozási környezetekben is ki lehet próbálni. Készítsünk olyan programot (például a Scratch környezetben), amellyel a robot követni tud egy megadott színű (például fekete) vonalat! Rajzoljunk olyan robotot, amelynek különböző színű csápjai (például kék és piros) vannak a bal és jobb oldalon! Rajzoljuk meg a pályát! Használjuk a Scratch színérzékelő blokkját annak eldöntésére, hogy merre kell kanyarodnia a robotnak!