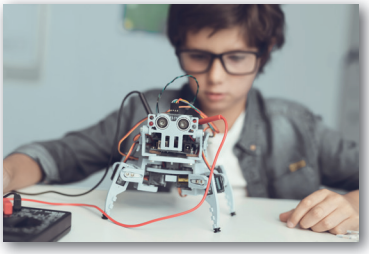


Bevezetés a robotikába

Ebben a fejezetben robotok irányításával, illetve programozásával foglalkozunk. Megismerkedünk a kapcsolódó fogalmakkal, illetve átismételjük ezeket.

Bizonyára már te is láttál robotot, vagy hallottál róluk, láttad a filmekben, híradásokban. De vajon minden olyan eszköz robotnak tekinthető, amelynek a nevében szerepel a robot szó? Robot-e pl. a konyhai robotgép? Egyáltalán melyek azok a tulajdonságok, amelyek alapján azt mondhatjuk egy eszközről, hogy robot? A következőkben ezzel foglalkozunk.

Feladat

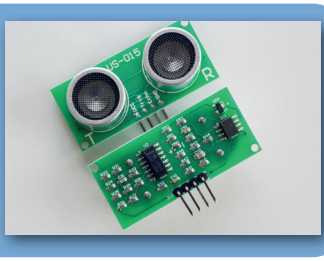
Gyűjtsük össze, hogy eddigi tanulmányaink és tapasztalataink alapján mit tudunk a robotokról! Írjuk rá egy cetlire egy (tényleg létező) robot nevét! Egy másik cetlire pedig írjuk fel azt, hogy ez a robot milyen tevékenységet végez! Helyezzük el a cetliket a tábla megfelelő részein, és mondjuk el hangosan a társainknak a választásunkat! Ha mindenki végzett, beszéljük meg, hogy milyen szempontok szerint lehetne csoportosítani az összegyűjtött robotokat, valamint a cselekvéseket!

A közös ötletelés során bizonyára sokféle eszközt és tevékenységet sikerült összegyűjteni. Ezen tevékenységek lehetnek egyszerűek, de akár nagyon bonyolultak is. Ennek oka, hogy a technológia fejlődésével a robotok is egyre többféle feladatot tudnak ellátni. Az alapján, hogy a robotok milyen jellegű feladatok ellátására képesek, különböző generációkba sorolhatjuk őket.

Robotgenerációk

Az **első generációs robotok** az 1960-as években jelentek meg, és egyszerű, például tárgyak mozgatásával kapcsolatos feladatokat oldottak meg. Ezeknek a robotoknak még nem voltak érzékelőik, így még nem tudtak a környezetük jellemzőire, változásaira reagálni. A **második generációs** robotok az 1970-es évekre jellemzőek, és ezek a robotok már érzékelőkkel lettek felszerelve.

Az **érzékelők** (más néven **szenzorok**) teszik lehetővé azt, hogy a robot érzékelje a környezetét, és az információk kiértékelése alapján döntéseket hozhasson, másmás cselekvéseket hajthasson végre.



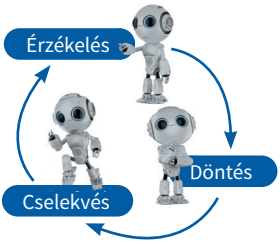
► Egy távolságérzékelő szenzor

Feladatok

1. Gyűjtsük össze közösen, milyen külvilágból származó információkat érzékelhetnek a szenzorok! Pl. hőmérséklet, testhő stb.
2. Ötleteljünk azon, hogy a saját iskolánkban milyen feladatot lehetne rábízni egy második generációs robotra! Alkossunk csoportokat, és tervezzünk meg egy robotot! Gyűjtsük össze, hogy milyen érzékelőkkel kellene felszerelni a robotot, hogy elvégezhesse a feladatát!

A **harmadik generációs robotok** már igen fejlettek, nemcsak szenzorokkal rendelkeznek, hanem sokkal okosabbak, mint az elődeik. Akár a korábbi tapasztalataik alapján is képesek tanulni (**gépi tanulás**), szabályszerűségeket megállapítani. Azt, hogy ilyen bonyolult feladatokra is képessé váltak a robotok, a **mesterséges intelligencia** területén elvégzett kutatások tették lehetővé.

Napjaink korszerű robotjaival szemben tehát **alapvető elvárás**, hogy **érzékeljék** a környezetük jellemzőit, a környezetben bekövetkezett változásokat. Ezek kiértékelése alapján pedig **döntést** kell hozniuk, és az alapján **cselekedniük** kell.

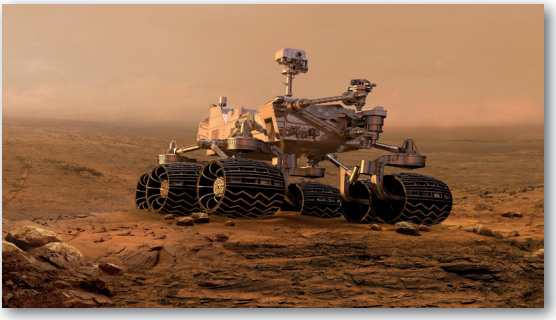


Milyen méretű lehet egy robot?

A robotok nemcsak a tudásuk, hanem a méretük szerint is nagyon változatosak lehetnek. Vannak egészen pici (mikro- és nano-) robotok, amelyeket például az orvostudományban alkalmaznak. Képesek lehetnek az emberi test keringési rendszerének feltérképezésére, vagy akár gyógyszer eljuttatására a test megfelelő pontjaiba.

A nagyobb robotok közé tartozik például az az autó méretű marsjáró robot, amelyet a NASA 2011-ben bocsátott fel a Mars felületének megvizsgálására. A robot annyira fejlett volt, hogy a fúrójával mintát tudott venni a kőzetekből, azok összetételét megvizsgálta, és az adatok kiértékelése után megállapította a kőzetek korát.

► Ipari robot egy süteménygyárban



► Robot a Mars bolygón (Curiosity marsjáró)

Feladatok

1. Vitassuk meg, hogy miért nagyon fontos elvárás az a marsjárótól, hogy önállóan képes legyen a kutatási feladatok elvégzésére? Miért nem elegendő az, hogy a Földről irányítják a robot mozdulatait a mérnökök és kutatók?
2. Gyűjtsük össze, hogy melyik volt a legkisebb, illetve legnagyobb létező robot, amit saját szemünkkel vagy akár dokumentumfilmekben láttunk!

Készítsünk algoritmust!

Ahhoz, hogy a robotokat irányítani, vezérelni tudjuk, ismernünk kell, hogy a robot milyen cselekvésekre képes, illetve milyen utasításokat képes végrehajtani.

Az irányítás kipróbálásához használhatunk egy *virtuális* (fizikai értelemben nem létező, számítógépes programban elérhető) robotot, de akár valós, kézzel fogható oktatási robotot is.

Kezdetben csak arra van szükségünk, hogy a robot képes legyen előrehaladni megadott távolságot, és balra, illetve jobbra fordulni 90 fokkal. A robot lehet akár egy virtuális, számítógépes programban megvalósított robot is.

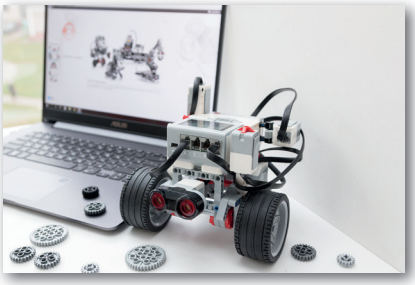
A **robot vezérlése** leegyszerűsítve azt jelenti, hogy a robot pontosan az általunk kiadott utasításokat hajtja végre. A robot mozgása ilyenkor csak a kiadott parancsoktól függ, és a robot nem végez semmilyen korrekciót. Ha például egy hibás utasítás miatt egy falnak vezetjük a robotot, a kerekei nem állnak le az akadályt érzékelve, hanem folyamatosan működnek tovább. Emiatt akár könnyen meg is sérülhetnek. Ezért is fontos a megoldandó feladat alapos tervezése!

Tervezzük meg, mit fog csinálni a robotunk!

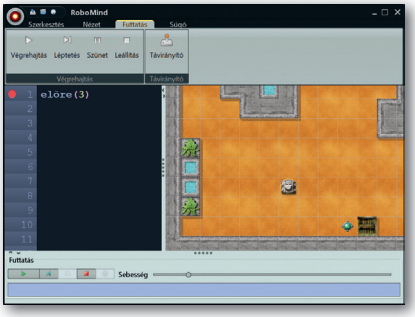
Kezdjünk egy egyszerű példával! A robotot ki kell vezetnünk egy labirintusból. Azt tudjuk, hogy a robot képes előremenni egy távolsággal, jobbra és balra fordulni. A robot kezdetben észak felé néz, vagyis a felülnézeti képen felfelé irányban fog elmozdulni.

Milyen utasításokat kell kiadnunk ahhoz, hogy a robot a kezdeti helyéről eljusson a jobb felső sarokban lévő kijáratig?

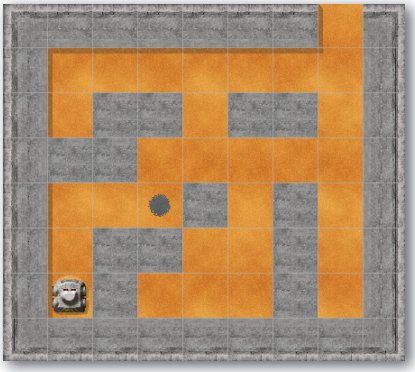
A feladat megoldásához először meg kell terveznünk az egyes lépéseket. Ekkor még nem kell egészen pontosan tudnunk azt, hogy a robot milyen parancsokat tud végrehajtani. Egyszerűen, **mondatszerű elemekkel** írjuk le, hogy milyen lépésekkel oldható meg a feladat.



▶ LEGO Mindstroms EV3 robot



▶ Egy robotszimulátor program (RoboMind) felülete



▶ A labirintus, amelyből ki kell juttatni a robotot

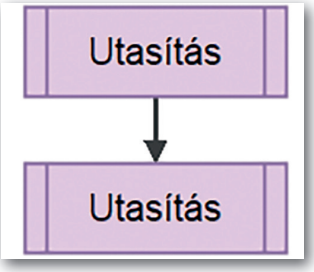
Ha például a szürke ponttal jelölt mezőre akarnánk eljuttatni a robotot, gondolkodhatnánk így:

```
Program
  Menj előre 2 lépést
  Fordulj jobbra
  Menj előre 2 lépést
Program vége
```

Amit most készítettünk, az egy **algoritmus**. Ez nem más, mint a feladat megoldásához szükséges elemi utasítások sorozata.

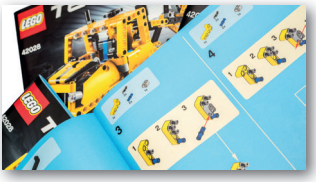
Ebben az algoritmusban egyszerű tevékenységek, utasítások sorozatát írtuk le, amelyek egymás után hajtódnak végre. Ezt más néven **szekvenciának** nevezzük.

Az algoritmust kezdetben szövegesen, mondatszerű leírással adjuk meg. Az algoritmusokat nagyon gyakran grafikusan, **folyamatábra** segítségével is le szokták írni.



► Folyamatábra részlete egy szekvenciával (utasítások sorozatával)

Fontos tudni, hogy az algoritmusokkal nemcsak az informatika területén találkozhatunk, hanem a hétköznapi életben is. Amikor például egy útmutató alapján összerakunk egy építőjátékot, akkor is egy algoritmus szerint dolgozunk.



Feladatok

1. Szerveződjünk három-négy fős csoportokba. Minden csoport kap egy fogalmat, amelyhez kapcsolódóan egy hétköznapi tevékenységhez kell algoritmust készítenünk és ismertetnünk. A fogalmak: *ital/étel automata, sport, társasjáték, tisztálkodás, utazás, állattartás, szülinapi buli, osztálykirándulás, hírek megosztása, útbaigazítás*. Az egyes algoritmusok ismertetésekor beszéljük meg, hogy milyen előfeltételek szükségesek ahhoz, hogy az algoritmus valóban működjön.
2. Készítsük el önállóan azt az algoritmust, amellyel a robot kivezethető a labirintusból! Párokban ellenőrizzük le egymás megoldásait Valóban sikerül a megadott lépésekkel kijuttatni a robotot az útvesztőből? Mi lehet az oka annak, ha nem egyezik meg (egyébként helyes eredményt adó) két megoldás?

Mennyire egyértelműek az utasítások?

Csoportos feladat

Menjünk olyan helyszínre, ahol nagyobb szabad tér van! Ezen a téren jelöljünk meg egy kiindulópontot és egy végpontot!

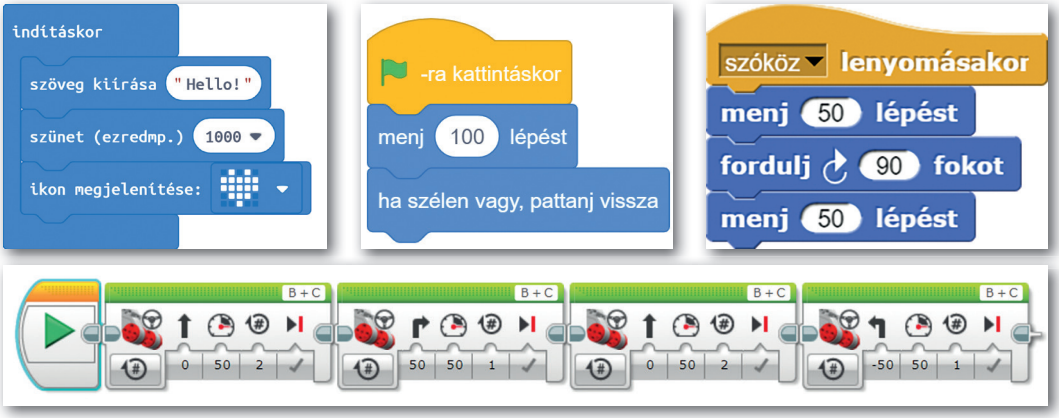
Alkossunk 4-5 fős csoportokat!

Minden csoportban legyen egy olyan diák, akinek bekötjük a szemét. Készítsük el azt az algoritmust, amiről úgy gondoljuk, hogy elvezeti a társunkat a kiindulási ponttól a célig! Adjuk ki az algoritmusnak megfelelő szóbeli utasításokat a társunknak! Mit tapasztalunk? Tényleg eljutott a célhoz? Most adjuk ki az utasításokat úgy, hogy a társunk pontosan visszafelé járja be az útvonalat. Visszajutott a kezdőpontig?

Vitassuk meg, hogy a játék során milyen fontos tanulságokat szereztünk az irányítással kapcsolatban! Hogyan lehetett volna egyértelműbbé tenni az utasításokat? Ötleteljünk arról, hogy az igazi robotok irányítása mennyiben lesz más, illetve hasonló, mint amit ebben a játékban tapasztaltunk?

Ismerkedés a blokkprogramozási környezettel

Egy robot irányításának kipróbálására, szimulálására sokféle környezet alkalmas. Elsőként a blokkprogramozási környezetekkel ismerkedünk meg, amelyekben az egyes utasításokat és vezérlőszerkezeteket blokkok jelképezik. Ezeket a blokkokat nagyon egyszerűen egymáshoz illeszthetjük, így előállítva a programot.



► Blokkok különböző programozási környezetekben

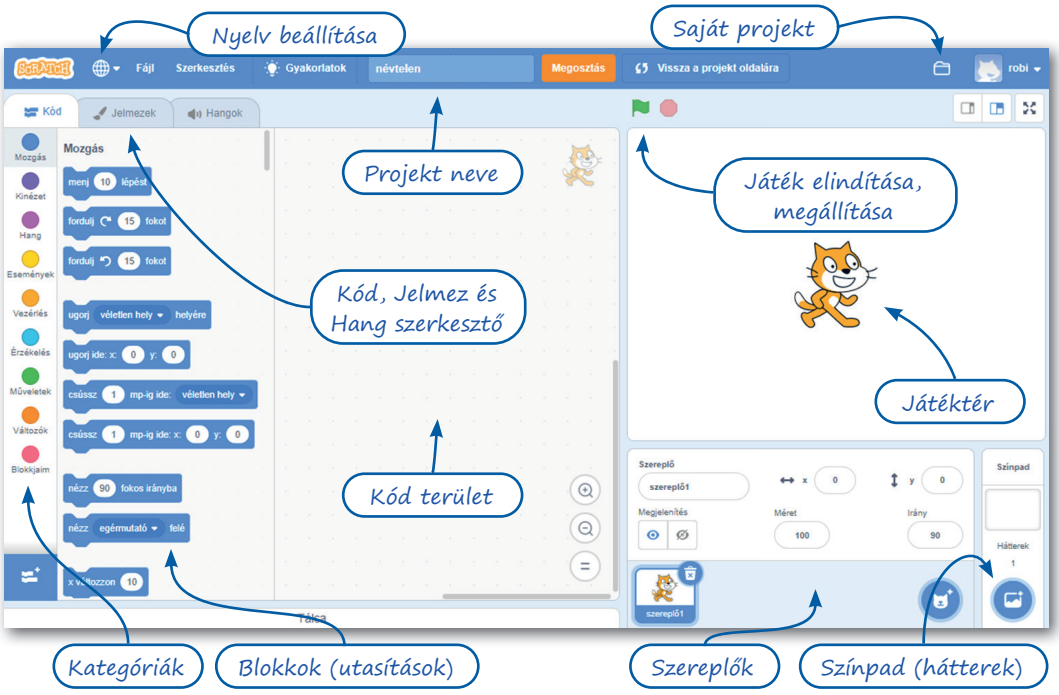
Feladat: Beszéljük meg, hogy az itt szereplő blokkoknak mi lehet a jelentésük? Vajon milyen esetben fognak ezek végrehajtódni? Mi lehet a kiadott utasítások eredménye?

A **blokkprogramozási környezetek** között találunk olyanokat, amelyeknek van letölthető, **számítógépre telepíthető** változata, vagy önálló alkalmazásként használhatók okostelefonon/tableten, de sok esetben nem is szükséges telepítenünk az alkalmazást, mert online módon, böngészőprogramban is használhatjuk azt. Egyes környezetek pedig többfajta lehetőséget is biztosítanak a felsoroltak közül, vagyis akár telepíthetőek is, de böngészőprogramban is használhatóak.

A **blokkprogramozási fejlesztői környezet** minden olyan eszközt biztosít számunkra, amelyre szükség van a programozás során. Ebbe beletartoznak maguk a blokkok, a különböző beállítási lehetőségek (pl. milyen nyelvű legyen a felület), futtatási és nyomonkövetési (tesztelési) lehetőségek, a programok elmentésének és betöltésének lehetőségei, sőt egyes esetekben az elkészült munkákat másokkal is egyszerűen megoszthatjuk a felület segítségével.

A programozási környezet részei

A blokkprogramozási környezetek felületében közös, hogy a programot egy **munkaterületen** vagy **kódterületen** kell összeállítani a különböző blokkokból. A blokkok kategóriák szerint csoportosítva vannak. A képernyőn megjelenő szereplő kinézetét testre lehet szabni, vagy úgy, hogy mi rajzolunk meg egy alakot, vagy úgy is, hogy felhasználunk egy már létezőt. Szintén beállíthatjuk azt, hogy a háttérben milyen kép legyen elhelyezve. Az összeállított programot le tudjuk futtatni, és akár meg is tudjuk állítani.

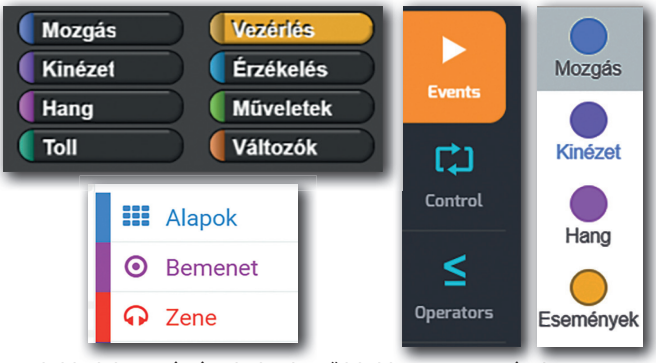


► Példa egy blokkprogramozási fejlesztői környezetre (Scratch)

Kategóriák a blokkok csoportosításához

A blokkok különböző **kategóriákba** vannak besorolva attól függően, hogy milyen célra lehet használni őket.

Ilyen kategóriák lehetnek például a mozgás, kinézet, zene, események, műveletek, toll, stb.

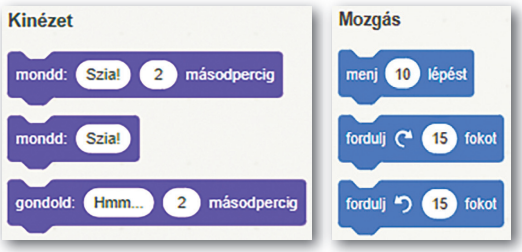


► Blokkok kategóriái a különböző blokkprogramozási környezetekben

A használható blokkok áttekintése

A kategóriákat jellemzően különböző színekkel különböztetik meg. Az azonos csoportokba tartozó blokkok azonos színűek.

Vagyis a blokk színe a blokk szerepére is utal.

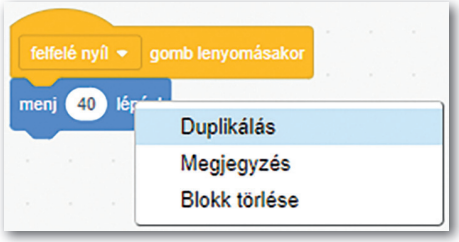


► A blokkok ugyanolyan színűek, mint a kategória (Scratch környezet)

Blokkok másolása, törlése

A munkaterületen elhelyezett blokkokból akár másolatokat is készíthetünk, ha újra fel akarjuk használni őket. Általában ehhez a jobb egérgombbal kell a blokkra kattintani, majd a menüből ki kell választani a duplikálás (megkettőzés) menüpontot.

Ugyanebben a menüben találhatunk törlési lehetőséget is. A törlés általában úgy is működik a programozási környezetekben, hogy a blokkot megragadjuk, és visszahúzzuk arra a területre, ahonnan kiválasztottuk.



▶ Blokk másolatának készítése (duplikálás)

Program végrehajtása, futtatása

A programot végre is tudjuk hajtatni (futtatni tudjuk). Ennek hatására a programozási környezetben láthatjuk a program eredményét. Ez az eredmény lehet például az, hogy elmozdul egy szereplő a képernyőn, de egy igazi robot esetén az is lehet az eredmény, hogy a robot ténylegesen elindul, és megtesz egy bizonyos távolságot.

A program nem biztos, hogy valóban azt a feladatot oldja meg, amit elgondoltunk, mivel az algoritmus megírása és a kódolás során is követhettünk el hibákat. Ezért a programot **tesztelnünk** kell, hogy meggyőződjünk a program helyességéről. A tesztelés során kapott eredményt **elemeznünk** kell annak érdekében, hogy a **hibajavítást** el tudjuk végezni. A hiba lehet magában a kódban is, de már a kigondolt algoritmus is tartalmazhatott hibákat, így vissza kell térnünk a megfelelő algoritmus kitalálásához.

Program mentése, betöltése

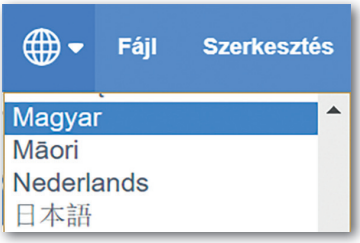
Az elkészült projektjeinket el tudjuk **menteni**, illetve a korábban elmentett projekteket **be tudjuk tölteni** a felületre. Így bármikor tovább tudjuk fejleszteni a korábban elmentett alkalmazásokat.

Tipp: Amikor elmentjük a projektjeinket, adjunk nekik olyan beszédes nevet, amely alapján később is tudni fogjuk, hogy a program milyen célt szolgált.

A felület nyelve

Mivel ezek a környezetek főleg gyermekek számára készültek, a fejlesztők ügyeltek arra, hogy az **alkalmazás nyelve beállítható** legyen, így sok esetben magyar nyelvre is átállíthatóak.

A magyar nyelvű felületen kezdetben jobban kiismerhetjük magunkat, és az önálló felfedezést is segíti.



▶ Nyelv beállításának lehetősége (Scratch)

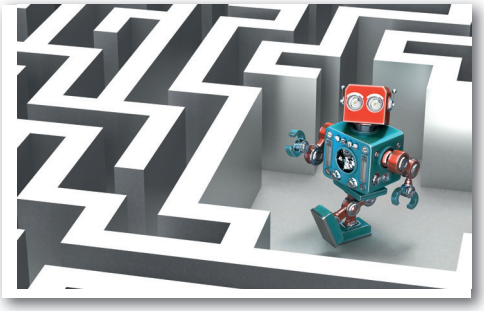
A robot és a pálya elkészítése

Még mielőtt valódi robotokat irányítanánk, érdemes az alap vezérlési és programozási lehetőséggel megismerkedni olyan környezetben, ahol **szabadon kísérletezhetünk**, és tapasztalatokat gyűjthetünk. Egy valódi robotot mindig sokkal nagyobb körültekintéssel kell programozni, mint egy szimulált, virtuális robotot. Az **igazi robot** akár **meg is sérülhet**, vagy **sérülést okozhat**, ha például ráesik a lábunkra vagy nagy sebességgel nekünk ütközik.

A későbbiekben egy olyan projektet készítünk el, amelyben egy robotot fogunk kivezetni egy labirintusból.

Első lépésként készítsük el a pályát, és rajzoljuk meg a robotot!

A robotot olyan módon kell megszemélyesítenünk, hogy lássuk, melyik irányba néz. Így egyértelmű lesz, hogy milyen irányban halad előre a megfelelő utasítás kiadásakor.



Feladatok

1. Tekintsük át, hogy milyen beépített jelmezek állnak rendelkezésre a programozási környezetben!
2. Jelmezként rajzoljunk meg egy olyan egyedi robotalakot, amelyet látva eldönthető, hogy melyik irányba néz. Figyeljünk arra, hogy alapesetben milyen irányba néz a szereplő a környezetben (jobbra, felfelé stb.), és annak megfelelően rajzoljuk meg a robotot.

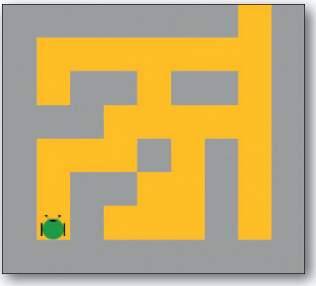
▸ A szereplő jelmezének beállítási lehetőségei a Scratch környezetben

Mi az alábbi robotot rajzoltuk meg, amelynek két kereke van. A kis antennák jelzik, hogy éppen jobbra néz.

▸ Az általunk megrajzolt robot

Ha elkészült a robotunk, folytassuk azzal, hogy elhelyezzük az üres labirintus képét a háttéren. Ez a kép megtalálható a letölthető állományok között.

Feladat: Töltsük le ezt a képet számítógépre! Módosítsuk a robot szereplő helyzetét és méretét úgy, hogy a labirintus bal alsó mezőjére kerüljön, és felfelé nézzen!
A projektet mentsük el a fejlesztői környezetben, hogy később továbbfejleszthessük!



▸ A labirintus beillesztése, a robot elhelyezése és átméretezése utáni állapot

A robot vezérlése

Az algoritmus elkészítése után izgalmas látni azt is, hogy az igazi (vagy a virtuális) robot hogyan hajtja végre a leírt lépéseket. Ehhez meg kell tanulnunk a robot nyelvén beszélni, vagyis kódolnunk kell!

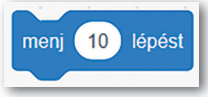
A **kódolás** művelete azt jelenti, hogy az algoritmust egy **programozási nyelv** szabályai, a rendelkezésre álló utasítások szerint átfogalmazzuk.

A kódolás eredménye az adott programozási nyelven megfogalmazott **program**. Az így előállt programot már meg lehet próbálni **végrehajtani**, más néven **futtatni**.

A kódolás nem teljesen ugyanazt jelenti, mint a **programozás**. A programozás egy összetett folyamat. Nemcsak a kódolási tevékenységet foglalja magában, hanem sok más tevékenységet is: a megoldandó feladat precíz leírását (specifikálását), algoritmizálást, tesztelést, hibajavítást, a program továbbfejlesztését és így tovább.

Előzőleg megrajzoltuk a robot, és elhelyeztük a pályát a háttérben. Itt az ideje annak, hogy a robot elkezdjen mozogni a pályán!

Lépjünk előre!



Feladat: Keressünk olyan blokkot a környezetben, amellyel előre lehet léptetni a robotot! Helyezzük el ezt a kódterületen!

Tipp: A blokkprogramozási környezetek többségében, ha duplán kattintunk az elhelyezett blokkra, akkor az végre is hajtódik, így mindjárt látjuk annak hatását.

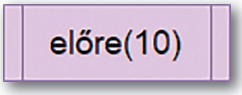
Láthatjuk, hogy az előrelépés mértékét a blokk belsejében elhelyezett ovális mezőben adhatjuk meg. Ide nemcsak egy konkrét szám kerülhet, hanem különböző műveletek eredménye is.

Feladat: Tekintsük át, hogy milyen műveletek érhetőek el a programozási környezetben. Próbáljuk ki azt a blokkot, amely véletlenszerűen meghatározott lépéssel lépteti előre a szereplőt!

▸ Előrelépés véletlenszerűen választott számmal a Scratch környezetben

A paraméter fogalma

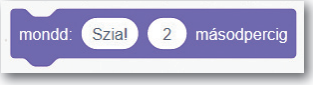
A blokk belsejébe beírt számot **paraméternek** nevezzük. A paraméter értéke most azt befolyásolta, hogy hány lépést tett előre a robot. Ha az algoritmust mondatszerű leírással adjuk meg, a paramétert sokszor **kerek zárójelek** közé tesszük. A folyamatábrában ugyanez a megadási mód látszik.



▸ Folyamatábra az előre utasítással

Feladat: Fedezzük fel önállóan, hogy mekkora számot kell megadni paraméterként ahhoz, hogy a labirintusban egy mezőnyi (egységnyi) távolságot haladjon előre a robot!

A **paraméter** nem mindig számot jelöl, **akár szöveg is lehet**. Jó példa erre az itt látható blokk. A szöveges paraméter adja meg, hogy milyen szöveg jelenik meg a szereplő mellett, a szám pedig azt, hogy hány másodpercig legyen látható a szöveg.



▸ Szöveg és szám is lehet paraméter

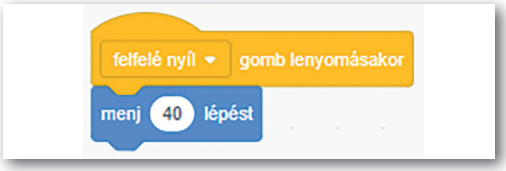
A próbálgatáson kívül máshogy is meghatározhatjuk, hogy mekkora távolságot kell előrelépnünk a megadott háttérképen. Ehhez tudnunk kell, hogy a háttérkép mekkora méretű. Ezt a méretet akár a programozási környezet is jelezheti, de akár az operációs rendszer beépített lehetőségeit használva is kideríthetjük.

Tegyük fel, hogy a kép magassága 320 képpont. A kép függőlegesen most 8 mezőt tartalmaz, így egy mező 320 / 8 = 40 képpont magasságú. Vagyis most 40 lépést kell előre lépnie a robotnak, hogy a megfelelő helyre kerüljön.

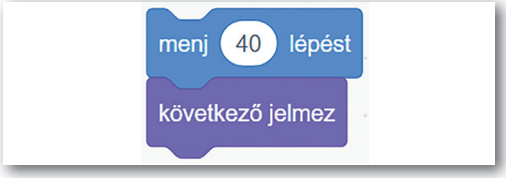
▶ A háttérkép méretének leolvasása (Scratch)

A robot vezérlése billentyűzetről

Most próbáljuk ki, hogy billentyűzet segítségével hogyan vezérelhetnénk a képernyőn a virtuális robotot! A blokkprogramozási környezetekben az ***Események*** *vagy* ***Vezérlés*** *kategóriákban találjuk azokat a vezérlőblokkokat, amelyek lehetővé teszik, hogy az utasítások egy billentyű lenyomásakor hajtódjanak végre.*



▶ A felfelé nyíl hatására előrelép a szereplő 40 egységet (Scratch környezet)



▶ Jelmez váltása (Scratch környezetben)

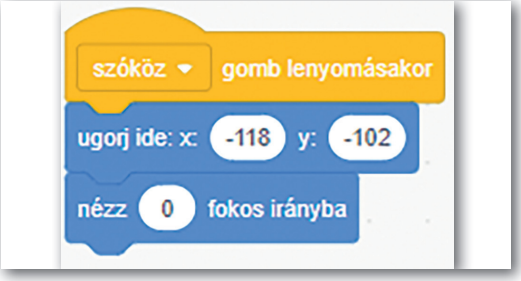
A szereplők helyét (koordinátáját) le lehet olvasni a képernyőről. Az *x* koordináta azt jelöli, hogy vízszintesen melyik pozíción helyezkedik el a szereplő. Az *y* koordináta a függőleges tengelyen lévő pozíciót jelöli.

▶ A szereplő helyzete, mérete, iránya (Scratch és Snap! környezetekben)

Feladat: Készítsük el azt a projektet, amelyben a felfelé nyíl megnyomásakor a robot előrelép 40 egységgel, és balra fordul 90 fokkal a balra nyíl billentyű megnyomásakor. A jobbra nyíl gomb hatására forduljon jobbra 90 fokkal. A szóköz billentyű megnyomásakor pedig kerüljön vissza a kiindulási helyére, a labirintus bal alsó mezőjébe.

Tipp: Ha egy szereplőnek több jelmezt is megrajzolunk, akkor a szereplő mozgás közben váltogatni tudja ezeket a jelmezeket egy megfelelő utasítás kiadásával.

Ha a programban szeretnénk megadni, hogy pontosan milyen koordinátára kerüljön a robot, akkor tipikusan a **Mozgás** kategóriában találjuk meg az erre szolgáló blokkokat.



▶ A szóköz billentyű lenyomásakor a szereplő visszatér a kiindulási állapotba, és felfele néz. (Scratch környezet)

Gyakorlás, saját ötletek megvalósítása

Most már tudjuk azt, hogy hogyan hozhatunk létre szereplőt, hogyan állíthatunk be hátteret, és láttuk azt is, hogy billentyűzetről hogyan irányíthatjuk a szereplőt. Most itt az ideje, hogy önállóan is alkossunk!

Akár egyedül, akár párokban, akár csoportmunkában is sokféle érdekes program készíthető. Ha van kedved, folytasd otthon is, amibe belekezdtél! Mutasd meg a szüleidnek, testvéreidnek is, hogy milyen programokat sikerült el készítened!

Projektmunka

1. Ötleteljünk azon, hogy az eddigi tudásunk alapján milyen táblás társasjátékot tudnánk megvalósítani a tanult programozási környezetben! Tervezzük meg a játékot, majd valósítsuk meg! 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!

Egyéni feladatok

1. A nyúl és teknős versenyéről szóló mesét szinte mindenki ismeri. Most valósítsuk meg ezt egy program segítségével! Rajzoljuk meg a szereplőket! Ha a nyúlra vagy a teknősre kattintunk, akkor lépjen előre egy véletlenszerűen meghatározott értéket! Állítsuk be úgy a véletlen szám paramétereit, hogy nagyobb eséllyel nyerjen a nyúl, mint a teknős! De készítsünk egy turbóteknőst is, amelyet nem győzhet le a nyúl!
2. Készítsünk egy olyan projektet, amelyben egy tetszőleges szereplőt lehet jobbra és balra léptetni a billentyűzet segítségével. A szereplő alakját mi találjuk ki és rajzoljuk meg!
   A szereplőnek legyen több jelmeze is, amit váltogatni tud a lépések során.
   Próbáljuk úgy elkészíteni a jelmezt, hogy úgy tűnjön, ténylegesen halad a szereplő (pl. egy pálcikaember esetén mozog a lába, egy autó esetén forog a kereke, és így tovább).
   Készítsük el hozzá a megfelelő hátteret is, amelyen szívesen elhelyeznénk a szereplőt!

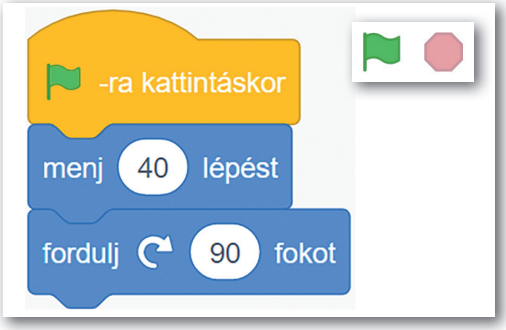
Páros feladatok

1. Találjunk ki közösen egy történetet, amelyet különböző szereplőkkel el lehet játszani. Rajzoljuk meg a szereplőket és a szükséges háttereket is! A szereplők gombnyomásokra reagáljanak!
2. Készítsünk egy *Kérdezz-felelek* programot, amelyben a szereplő alakja mellett az Igen, Nem, Talán szövegek jelennek meg az I, N, T billentyűk lenyomásakor. A szereplő jelmeze is változzon meg a választól függően! A játékot próbáljuk ki párokban! Tegyünk fel egymásnak eldöntendő kérdéseket, de ezekre ne mi, hanem a szereplők válaszoljanak a megfelelő gombok megnyomásával!

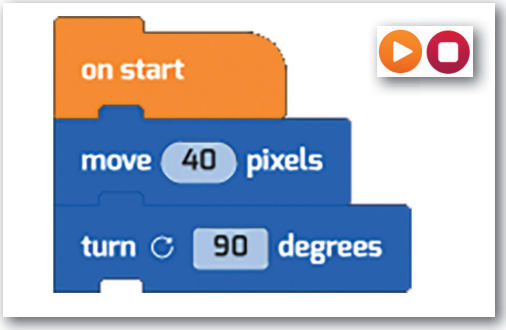
A robot irányítása utasítások segítségével

Korábban a billentyűzettel irányítottuk a robotot. Lépjünk tovább, és írjuk meg a kijutáshoz szükséges programot!

Ha azt szeretnénk, hogy a robot önállóan menjen végig az útvonalon, akkor az utasításokat egy olyan vezérlőblokkban kell elhelyeznünk, amely a program indításakor automatikusan lefut. Ez a vezérlőblokk programozási környezetenként eltérő lehet.



▶ A Scratch és Snap környezetekben a zöld zászlós vezérlőblokk jelenti a főprogramot. A programot a zöld zászló megnyomásával indíthatjuk el, a piros nyolcszöggel pedig leállíthatjuk.



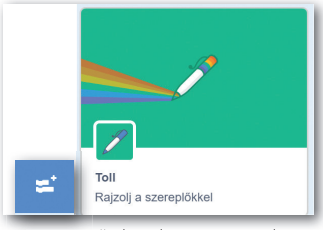
▶ A Tynker környezetben az On start (Induláskor) blokkban kell elhelyeznünk a főprogramot! A programot a háromszög ikonnal tudjuk elindítani, és a négyzet ikonnal tudjuk leállítani.

Feladat: Korábban elkészítettük azt az algoritmust, amely a robotot kivezeti a labirintusból. Most próbáljuk ki a megoldásunkat az általunk használt blokkprogramozási környezetben! A program indításakor a robot kerüljön vissza a kiindulási helyzetébe, nézzen felfelé, majd jelenjen meg mellette a „Szia” szöveg. Utána menjen el a kijáratig úgy, hogy nem érhet hozzá a falakhoz! A robot minden elfordulás után várakozzon 1 másodpercnyi időt, hogy nyomon lehessen követni a haladását. Amikor eljutott a robot a kijáratig, jelenjen meg mellette a „**Kijutottam!**” szöveg!

Merre járt a robot?

Ha meg tudnánk oldani azt, hogy a robot megjelölje, hogy mely mezőkön járt már, akkor a megtett útvonalat akár a program végeztével, utólag is elemezni tudnánk.

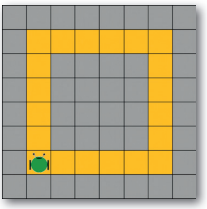
A blokkprogramozási környezetek között sok olyat találunk, amelyekben a szereplők vonalat húzhatnak maguk után. Van olyan környezet is, amelyben egy bővítmény hozzáadása után tudjuk ezt a funkciót használni!



▶ Toll bővítmény használata (Scratch)

Feladat: Derítsük ki, hogy az általunk használt programozási környezetben hogyan lehetne megoldani, hogy a mozgása során egy vonalat húzzon maga után a szereplő! Módosítsuk úgy a korábban elkészült programot, hogy a robot húzzon egy vonalat maga után! A program újbóli elindításakor legyen letörölve az előzőleg meghúzott vonal, és a robot kerüljön vissza a kiindulási helyére!

Ismétlődő utasítások, ciklusok



Nézzünk egy másik pályát, amelyet a letöltendő állományok között megtalálunk! Ez nem egy labirintus, hanem egy négyzet alakú pálya. Most vezessük végig ezen a pályán a robotunkat!
A sárgával jelölt pálya hat egység hosszú és hat egység széles. A robot kiindulópontja legyen ismét a pálya bal alsó pontja, és nézzen a robot észak felé.

Ha végig akarjuk vezetni a pályán a robotot, akkor gondolkodhatnánk így:

```
Program
  Menj előre 5 lépést
  Fordulj jobbra
  Menj előre 5 lépést
  Fordulj jobbra
  Menj előre 5 lépést
  Fordulj jobbra
  Menj előre 5 lépést
  Fordulj jobbra
Program vége
```

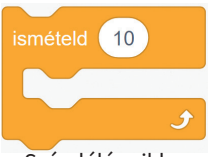
Láthatjuk, hogy az algoritmus ismétlődő elemeket tartalmaz. Négyszer ismételjük meg az öt egységgel történő előrelépést és az elfordulást. Ezt sokkal rövidebben is megfogalmazhatnánk:

```
Program
 Ismételd 4 alkalommal
    Menj előre 5 lépést
    Fordulj jobbra
 Ismétlés vége
Program vége
```

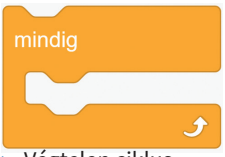
Az ismétlődő tevékenységeket úgynevezett **ciklusokba** érdemes szerveznünk. A ciklus belsejében (ezt nevezzük **ciklusmag**nak) helyezzük el azokat az utasításokat, amelyeket ismételni kell.

Az algoritmus szöveges leírásában az `ismételd` szó jelöli a ciklus kezdetét, és az `ismétlés vége` a befejezését.
Ciklusokból többféle van. Kezdetben ismerkedjünk meg három fajtájával!

Ezeket általában a *Vezérlés* vagy *Ciklusok* kategóriában találjuk.
Számlálós ciklust akkor használunk, amikor ismerjük, hogy az utasításokat pontosan hány alkalommal kell végrehajtani (pl. `ismételd 3-szor … ismétlés vége`).
Végtelen ciklust akkor alkalmazunk, ha az utasításokat állandóan ismételni akarjuk. Ilyenkor a program futását például úgy tudjuk leállítani, hogy megnyomjuk a Leállítás gombot (pl. `ismételd … ismétlés vége`).
A **feltételes ciklusokra** az jellemző, hogy az utasítások ismétlése egy feltétel igaz, vagy hamis voltától függ (pl. `ismételd amíg feltétel … ismétlés vége`).
Programozási nyelvenként különbözhet, hogy a megadott feltétel a ciklus leállításának (kilépésnek) vagy a ciklus végrehajtásának feltétele.



► Számlálós ciklus



► Végtelen ciklus

► *Feltételes ciklus*

Feladatok: Oldjuk meg az alábbi feladatokat:

1. A robot pontosan háromszor menjen végig a körpályán!
2. A robot addig menjen a körpályán folyamatosan, amíg le nem állítjuk a programot.
3. A robot addig körözzön, amíg le nem nyomjuk a „v” billentyűt!

Az utóbbi feladat megoldásához önállóan kell felfedeznünk, hogy a feltételt hogyan kell megadni. Egy biztos: hatszög alakú blokkot kell keresnünk!

A fal érzékelése

Előző projektjeinkben a robot úgy viselkedett, mint egy első generációs robot. Csak a kiadott parancsokat hajtotta végre, nem tudott reagálni a környezetére. Folytatásként egy fejlettebb (második generációs) robot működését fogjuk szimulálni. Ez a robot már érzékeli, hogy szabad terület van-e előtte, így megtaníthatjuk majd arra, hogy elmenjen önállóan a falig.

Feladatok

1. Gondoljuk át, hogy ezen algoritmus alapján milyen útvonalat jár be a robot!
2. Páros munkában keressünk olyan kiindulási helyet a robotnak, amelyre igaz, hogy a fenti parancsok kiadásával szintén kijutna a labirintusból! Keressünk egy olyan kezdőpontot is, ahonnan nem tudna kijutni a robot ezen algoritmus alapján!
3. Van egy olyan útvonal, amely szintén elvezet a kijáratig, de most elhalad mellette a robot. Melyik ez az útvonal? Hogyan kellene módosítani az algoritmust, hogy ezen az útvonalon menjen el a kijáratig?

```
Program
  Menj a falig
  Fordulj jobbra
  Menj a falig
  Fordulj balra
  Menj a falig
  Fordulj jobbra
  Menj a falig
  Fordulj balra
  Menj a falig
Program vége
```

► Az új algoritmus

Ahhoz, hogy továbbléphessünk, meg kell értetnünk a robottal azt, hogy a `menj a falig` művelet az elemi utasítások használatával (előre, jobbra, balra), hogyan hajtható végre. Vagyis ezt a műveletet elemi részekre kell bontanunk. Ezt nevezzük **lépésenkénti finomításnak.**

A **lépésenként finomítás** elve azt mondja ki, hogy a feladatokat olyan kisebb feladatokra (részfeladatokra) kell bontani, amelyeket már nem tudunk tovább bontani, vagy arra a kisebb feladatra már van működő megoldásunk.

Hogyan érzékeli a robot a környezetét?

Ahhoz, hogy a való világban egy robot érzékelje, hogy fal van előtte, szükség van valamilyen szenzorra (pl. távolságérzékelőre).

Az érzékelő által mért adatok kiértékelése után a robot már el tudja dönteni, hogy a vele szemben lévő terület szabad-e, vagy fal van rajta.

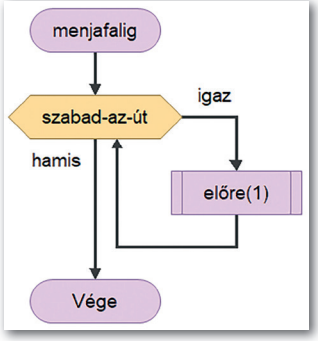
A menj a falig művelet azt jelenti, hogy addig kell előrelépkednie a robotnak, amíg a szemben lévő terület szabad. Ezt folyamatábrán szemléltetve látható, hogy a hatszög alakzat jelenti a feltételt (szabad-e az út?).

Az is szépen látszik, hogy milyen utasítások hajtódnak végre, amikor a feltétel igaz, és mi történik, amikor hamis.

► Az algoritmus leírása folyamatábra segítségével

```
Ismételd amíg szabad az út
        Menj előre 1 lépést
Ismétlés vége
```

► A menj a falig művelet algoritmusa szövegesen



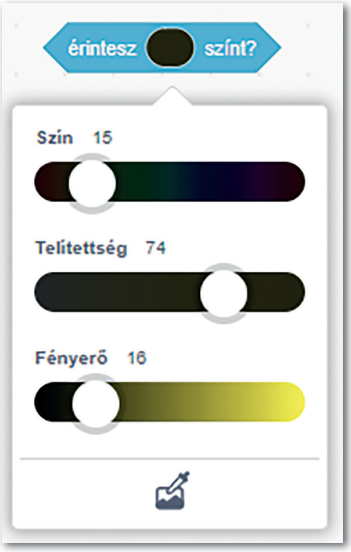
A színérzékelő használata

A példánkban úgy modelleztük a labirintust, hogy a falat szürke színnel jelöltük, a szabad területeket pedig narancssárga színnel. Vagyis most a szín hordozza azt az információt, hogy haladhat-e előre a robot, vagy sem.

Emiatt most érdemesebb színérzékelőt használni a távolságérzékelő helyett. A különböző blokkprogramozási környezetekben esetenként más-más érzékelők használhatók, de a színérzékelő megléte nagyon gyakori.

A színérzékelő akkor ad vissza **igaz értéket**, ha a szereplőnk (robotunk) érint egy megadott színt, ellenkező esetben **hamis** értéket kapunk.

Mivel most a falunk szürke színű, a szürke szín beállításával fogjuk tudni ellenőrizni, hogy elértük-e a falat. Vigyázzunk! Szürke színárnyalatból nagyon sokféle lehet, ezért pontosan azt a színt kell beállítanunk a feltétel megfogalmazása során, amely a labirintus háttérképen megtalálható.



▶ A szín kiválasztása Scratch környezetben. A pipetta ikonnal pontosan beállíthatjuk a vizsgálandó színt.

Feladat

1. Páros munkában fedezzük fel, hogy az általunk használt blokkprogramozási környezetben hogyan valósítható meg a színérzékelés!
2. Készítsük el azt a programot, amelyben a felfelé nyíl megnyomásakor a robot egy ciklus segítségével addig lép előre, míg falba nem ütközik!
3. Mit tapasztalunk? Ha eljutott a falig a robot, tovább tud-e haladni egy másik irányba?

Fordulás előtt lépjünk vissza!

Mivel a ciklus akkor áll le, amikor a szürke színt elérte a robot, ezért valószínűleg akkor is rá fog lógni valamelyik része a szürke színre, amikor elfordul valamelyik irányba és folytatná az útját.

Ezért a ciklus leállása után hátra kell lépnünk valamekkora távolságot, hogy a robot semelyik része ne lógjon rá a szürke területre!

Azt, hogy pontosan mekkora távolságot kell visszalépni, függ attól, hogy milyen robotot rajzoltunk, és hogy például a robotot pontosan a rendelkezésre álló hely közepére rajzoltuk-e vagy sem.

Feladat

Teszteljük le, hogy mi az a legkisebb távolság a visszalépésre, ahol már jól fog működni a programunk, és állítsuk be paraméterként ezt az értéket!

Gyakorlás, saját ötletek megvalósítása

Most már ismerjük azt is, hogy a különböző ciklusokat hogyan lehet használni. Nemcsak billentyűzetről, hanem utasítások alapján is tudjuk irányítani a szereplőket. Sőt, ezek a szereplők akár vonalakat is tudnak húzni, és érzékelhetik azt is, hogy milyen színt érintenek.

Ezek alapján újabb izgalmas programokat lehet megvalósítani.

Alkossunk együtt!

Alkossunk három-négy fős csoportokat! Tervezzünk meg egy olyan játékprogramot, amely azon alapul, hogy egy szereplőt (ami most ne robot legyen!) különböző akadályokon keresztül kell végigvezetni ahhoz, hogy teljesítse a küldetését.

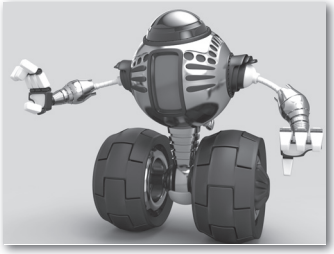
A szereplőket és a hátteret mi választhatjuk ki, illetve rajzolhatjuk meg. A játékprogramban kapjon szerepet a színérzékelő használata! Készítsük el a program algoritmusát, és csak utána álljunk neki a megvalósításnak. Törekedjünk arra, hogy mindenki kapjon olyan részfeladatot, amiért ő a felelős!

Ha elkészültünk a munkánkkal, mutassuk be egymásnak a játékokat, és próbáljuk ki egymás programjait!

Egyéni feladat

1. Némelyik robot hangot is képes kiadni. Fedezzük fel önállóan, hogy milyen lehetőségek vannak hangok megszólaltatására az általunk használt környezetben! A robot induláskor és megérkezéskor más-más hangot adjon ki! Van-e esetleg olyan bővítmény a programhoz, amit érdemes lehet telepíteni?
2. Készítsünk olyan programot, amelyben ha egy robot a kék színt érzékeli, akkor jobbra fordulva folytatja az útját, ha pedig sárgát, akkor balra fordul. Piros szín esetén forduljon vissza!

Páros feladat



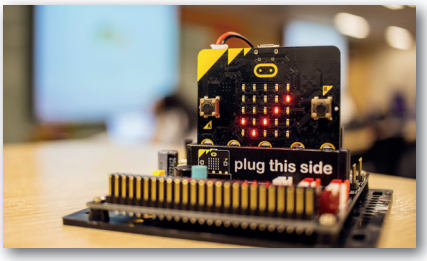
Az igazi robotok kerekeit motorok mozgatják. Sajnos néha előfordulhat üzemzavar. Tegyük fel, hogy a robotunknak két kereke van, és az egyik kerekét forgató motort nem lehet kikapcsolni, ezért a kerék állandóan a maximális fordulattal forog.

A másik motor teljesítményét tudjuk állítani, de így a robot vagy csak egyenesen tud haladni, vagy jobbra tud fordulni, balra nem.

Pármunkában vitassuk meg, hogy egy hibás működésű (balra fordulásra képtelen) robot el tudna-e jutni a kijáratig a korábban bemutatott pályán? Ha igen, hogyan változik az algoritmus és a programkód ebben az esetben? Próbáljuk ki a gyakorlatban!

Programozzunk micro:biteket!

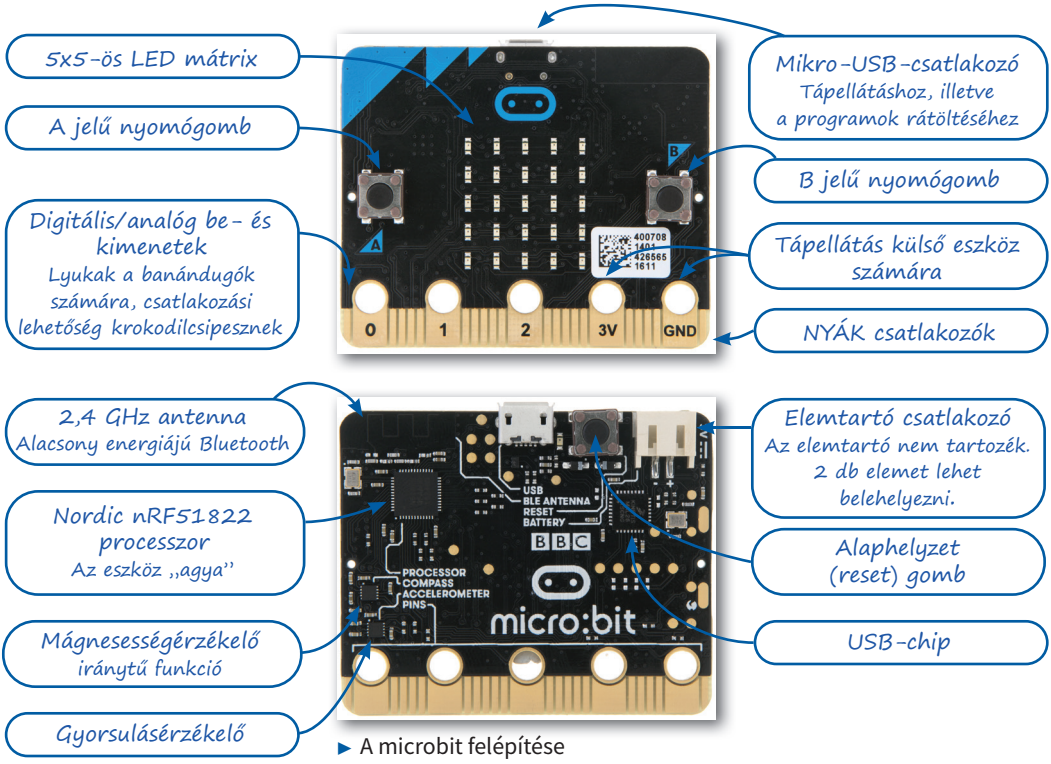
A következőkben egy olyan eszközt fogunk megismerni, amely valódi szenzorokkal van felszerelve, és blokkprogramozási környezetben is programozhatjuk. Ez nem más, mint a micro:bit.



A micro:bit egy oktatási célra kifejlesztett, egyetlen lapkán megvalósított miniszámítógép. Található rajta egy 5x5-ös LED kijelző, amelyen számokat, szövegeket és különböző ikonokat, animációkat jeleníthetünk meg.

Sokféle érzékelővel el van látva. Rendelkezik gyorsulásérzékelővel, hőmérséklet-érzékelővel, fényérzékelővel, irányérzékelővel.

Van két gombja (A és B jelű), amelyek megnyomására reagálni tud. A be- és kimeneti csatlakozói lehetővé teszik, hogy más eszközökkel is össze lehessen kötni. A Bluetooth kapcsolatnak köszönhetően pedig a micro:bitek egymással is képesek kommunikálni.



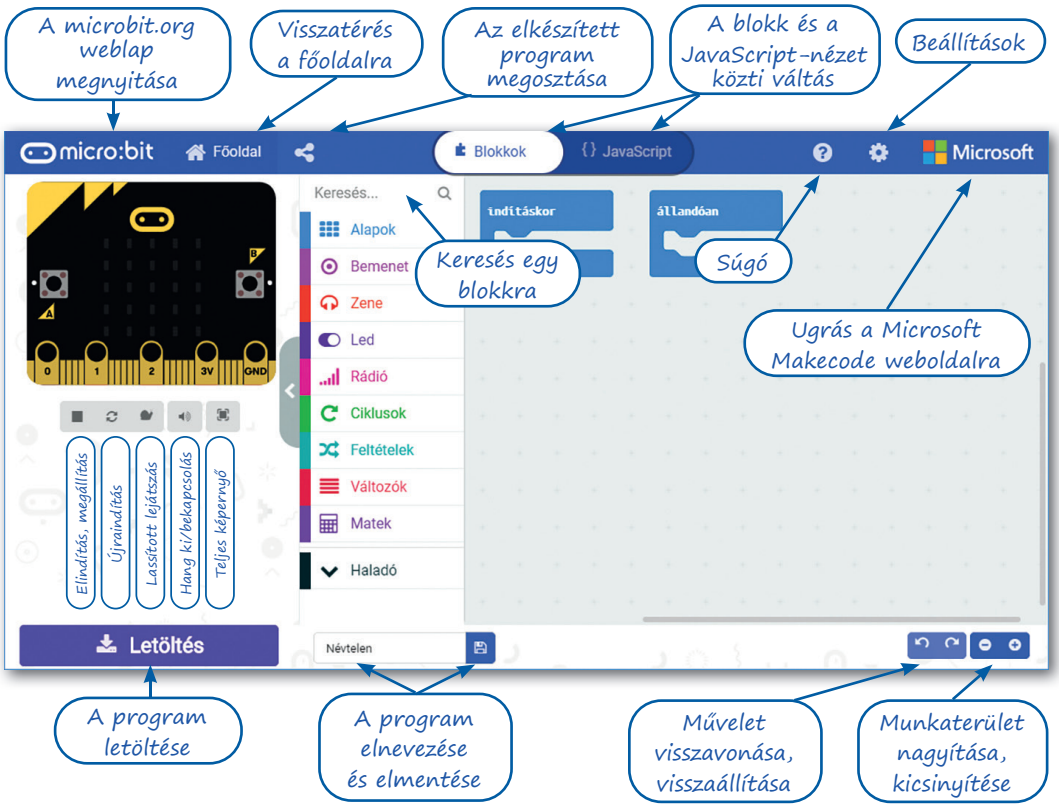
► A microbit felépítése

Az eszközt sokféle programozási nyelven lehet programozni, köztük több blokkprogramozási nyelven is. De nagy különbség van abban, hogy az egyes környezetekben milyen funkciókat érhetünk el. A legtöbb funkciót a MakeCode környezet biztosítja. Ennek van online elérhető (https://makecode.microbit.org/) és letölthető változata is.

Ismerkedés a felülettel

A *MakeCode* alkalmazás felülete nagyon hasonlít a korábban látott blokkprogramozási környezetekhez. A fő különbség, hogy most nem egy szereplőt irányítunk a képernyőn, hanem egy fizikailag is létező eszközre készítünk programokat.

Az elkészült programot egy szimulátor segítségével ki is próbálhatjuk, de ha rendelkezünk micro:bittel, akkor a programot az eszközre letölthetjük, és maga az eszköz fogja végrehajtani azt.



► A Makecode programozási felülete

A beépített szenzorok miatt a micro:bit képes reagálni a környezetből érkező hatásokra. Ilyen lehet, amikor valamelyik irányba megdöntjük az eszközt, megnyomjuk valamelyik vagy mindkét gombját, ha sötét helyre visszük, ha észak felé fordítjuk, ha megrázzuk, és így tovább.

► Néhány esemény, amelyekre az eszköz reagálni képes

Mire ügyeljünk?

Amikor a micro:bitekkel dolgozunk, ügyeljünk a következőkre!

- Mindig tiszta asztalon dolgozzunk! Különösen veszélyesek az eszköz számára a rövidzárlatot okozó tárgyak, mint pl. gemkapocs, tűzőgépkapocs, körző, egy ceruza kitört grafithegye, kifolyt üdítőital stb.
- Az elektrosztatikus kisülés akár tönkre is teheti az eszközt, ezért fontos, hogy mielőtt az eszközhöz hozzáérnénk, érintsük meg a számítógép házát, vagy a tanteremben lévő radiátor vagy fűtési csőhálózat felületét!
- Az eszköz úgy legyen elhelyezve az asztalon, hogy ne eshessen le, illetve véletlenül se lehessen lesodorni azt!
- Az egyes gesztusok kipróbálása során (pl. rázás) figyeljünk arra, hogy biztosan tartsuk az eszközt a kezünkben, ne ejtsük azt le!
- A tevékenységek során se az eszközben, se a társainkban ne tegyünk kárt!
- Elemről csak addig működtessük az eszközt, ameddig feltétlenül szükséges, egyébként használjuk az USB-kábelt!
- Ne kössünk az eszköz kivezetéseihez olyan külső eszközöket, amelyek pl. nagyobb tápfeszültséget igényelnek, mint amit az eszköz biztosítani tud! Ez akár tönkreteheti az eszközt, és az esetleges túlmelegedés miatt égési sérüléseket is okozhat!
- Az eszköz érzékelőit megzavarhatják külső tényezők, pl. erős mágneses tér, fémtárgyak stb. Ha a szenzorok nem megfelelő értékeket mérnek, vizsgáljuk meg, hogy nincs-e a környezetben valamilyen zavaró tényező!

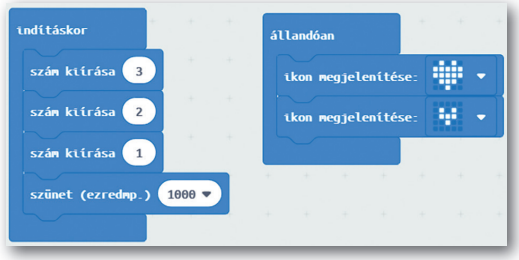
Készítsünk animációt!

Kezdjük az eszközzel való ismerkedést azzal, hogy a LED kijelzőn jelenítsünk meg különböző ábrákat és animációkat!

A munkaterületen alapesetben két blokkot is találunk. Az *indításkor* blokk tartalma egyszer, a program elindulásakor hajtódik végre. Az *állandóan* blokk tartalma állandóan ismétlődik, csakúgy, mint a korábban látott végtelen ciklus esetén.

Mindkét blokk kék színű, ez egyben arra is utal, hogy melyik kategóriába tartoznak. Ez most az Alapok kategória.

Feladatok

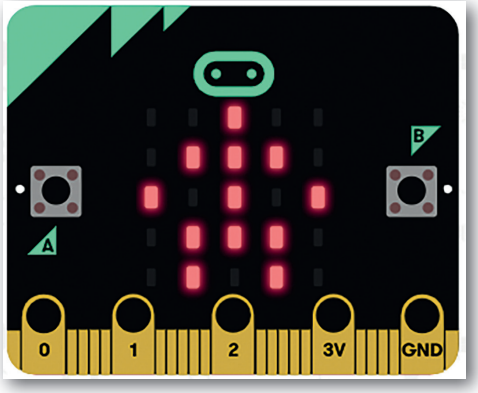


▶ Mit csinálnak a blokkok?

1. Próbáljuk ki, hogy az itt látható blokkok hatására mi fog történni a szimulátorban/eszközön! Mit tapasztalunk?
2. Bővítsük úgy a kódot, hogy indításkor az eszköz írja ki a keresztnevünket! Az ehhez szükséges blokk szintén az Alapok kategóriában található. Próbáljuk ki a programot!
3. Láthattuk, hogy a programban nemcsak szívecskét, hanem sok más ikont is meg lehet jeleníteni. Cseréljük le a szív ikonokat több, általunk választott ikonra. Bővítsük az állandóan blokk tartalmát úgy, hogy az animáció legalább öt ikonból (fázisból) álljon!
4. Nincs olyan ikon, amire szükségünk van? Semmi gond, rajzoljunk egyet kedvünk szerint! Készítsünk olyan animációt, amelynek minden fázisát mi rajzoltuk meg. Az ehhez szükséges blokkot szintén az Alapok kategóriában kell keresni.
5. A micro:bit logója pont a LED kijelző tetejénél van elhelyezve, és úgy néz ki, mintha egy robot feje lenne, két szemmel. Használjuk ki ezt, készítsünk egy robotot, amely tud integetni!

A *Bemenet* kategóriában találunk olyan vezérlőblokkokat, amelyekkel megoldható, hogy a blokkok akkor hajtódjanak végre, amikor megnyomtuk az *A* vagy a *B* gombot, vagy mindkettőt egyszerre.

Készítsünk olyan animációt, amelyben az *A* gombot lenyomva a robot az egyik karjával integet, a *B* gomb hatására pedig a másikkal! Ha mindkét gombot egyszerre megnyomjuk, akkor pedig csináljon valami egyedi mozgást, amit mi találunk ki. Az integetés legalább kétszer ismétlődjön. Keressük meg a megfelelő blokkot ehhez a Ciklusok kategóriában. Az integetés után a robot vegye fel a kiindulási helyzetét! Próbáljuk ki a projektet! Mentsük el a programot, hogy következő alkalommal folytathassuk!

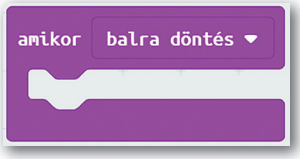


Használjuk az érzékelőket!

Korábban megismerkedtünk azzal, hogy a LED kijelzőn hogyan jeleníthetünk meg ikonokat, animációkat, és már az *A* és *B* gombokat is használtuk. Most továbblépünk úgy, hogy a micro:bit érzékelőiben rejlő lehetőségeket is kihasználjuk.

Feladat

Az *A* és *B* gombokkal működtetett animációs projektet fejlesszük tovább úgy, hogy ne csak a gombokra reagáljon a micro:bit. Az eszköz balra döntésekor integessen a robot az egyik kezével, jobbra döntéskor pedig a másikkal! Amikor megrázzuk az eszközt, akkor pedig mindkét karját emelje fel, majd engedje le!



► A Bemenet kategória egyik blokkja

Tipp: Ezeket a gesztusokat akár a szimulátorban is kipróbálhatjuk a virtuális micro:bit mozgatásával, illetve a Rázás (shake) gomb megnyomásával.

Készítsünk számlálót, ami egyben dobókocka is!

A micro:bitet akár elemről is lehet működtetni, a rátöltött programot pedig egész addig képes végrehajtani, míg egy másik programot rá nem töltünk. Ez azt jelenti, hogy a számítógéptől függetlenül, akár a szabadban is használni tudnánk.

A következőkben egy olyan alkalmazást készítünk, amellyel egyszerűen megszámolhatunk különböző dolgokat, sőt akár dobókockaként is működhet. Ezt az alkalmazást sokféle célra felhasználhatjuk:

- Megmérhetjük, hogy hány autó haladt el az iskola melletti útszakaszon adott idő alatt. Ha látunk egy autót, növeljük meg a számlálót.
- Egy meccsen megmérhetjük, hogy hány labdaérintés kellett ahhoz, hogy gól szülessen. A gól után lenullázhatjuk a számlálót, és folytathatjuk a mérést.
- Bármilyen társasjátéknál nyilvántarthatjuk, hogy hányszor nyertünk. Sőt, igazi dobókocka helyett használhatjuk a micro:bitet is.
- A tanórai feladatoknál véletlenszerű csoportbeosztást tudunk csinálni (nem csak digitáliskultúra-órán!). Akik azonos számot dobnak a virtuális dobókockával, azok egy csoportba kerülhetnek egy feladat megoldásakor.

Feladat

Gondoljuk át, hogy milyen más célra tudnánk használni egy ilyen eszközt, a saját érdeklődési körünkhöz, hobbinkhoz kapcsolódóan! Beszéljük meg, hogy kinek milyen ötlet jutott eszébe!

Hogyan tároljuk a számláló értékét?

A terv az, hogy egy olyan alkalmazást készítsünk, amelyben ha megnyomjuk a *B* gombot, akkor mindig növekedjen eggyel a számláló a micro:bit kijelzőjén. Az *A* + *B* gomb hatására pedig nullázódjon le a számláló. Az *A* gomb megnyomásakor véletlenszerűen jelenjen meg egy szám 1 és 6 között, mintha egy dobókockával dobtunk volna.

Azt már megismertük, hogy az *A* és *B* gombok lenyomását figyelő vezérlőblokkot hogyan kell használni. De hogyan tudnánk eltárolni, sőt növelni egy számot, valamint azt megjeleníteni a kijelzőn?

Változók használata

Ahhoz, hogy számokat, szövegeket el tudjunk tárolni egy program során, meg kell ismerkednünk a változókkal.

Az adatokat a program végrehajtásakor úgynevezett **változókban** tárolhatjuk. Az adat lehet szám, szöveg, sőt más, összetettebb adat is. A változó onnan kapta a nevét, hogy a program végrehajtása során a tartalma **változhat, módosulhat**.

A változóknak egyedi nevet kell adnunk. A változó értékére pedig a változó neve alapján hivatkozhatunk.

Nézzük meg a számláló alkalmazás kapcsán, hogy a gyakorlatban hogyan használhatnánk a változókat.

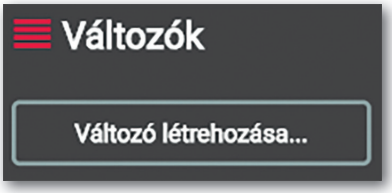
A programunkat azért készítjük el, hogy megszámoljunk különböző dolgokat. Ezt az adatot egy változóban fogjuk eltárolni.

Amikor **elnevezzük a változót**, akkor érdemes olyan nevet adni neki, amely alapján később is jól tudjuk majd azonosítani. Most *számláló* néven hozzunk létre egy változót!

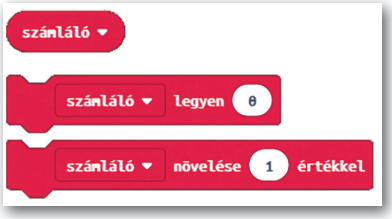
Miután ezt megtettük, a *Változók* között megjelenik egy ovális alakzat, benne a megadott névvel. Ezt a blokkot tudjuk használni akkor, ha le akarjuk kérdezni, hogy éppen milyen értéket tartalmaz a változó.

De nem csak ez a blokk jött létre, hanem a változó értékének beállítására szolgáló blokk is. Az érték szintén egy ovális alakzatban van elhelyezve. Ide számot is írhatunk majd, de akár szöveg is lehet benne, sőt különböző matematikai műveletek eredménye is.

Szintén létrejön egy blokk, amely a számláló értékét eggyel megnöveli. (Ha negatív előjelet használunk, akkor akár csökkenthetjük is az értéket.)



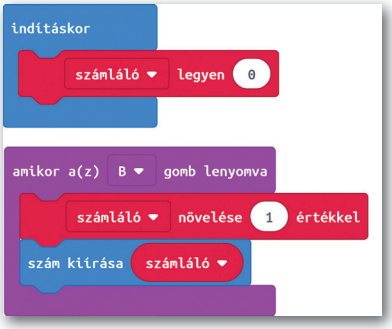
► Változó létrehozása a MakeCode felületen (Változók kategória)



► A létrehozott változóhoz tartozó blokkok

Feladatok

1. Készítsük el az itt látható programot, és próbáljuk ki a működését!
2. Módosítsuk úgy a programot, hogy az *A* + *B* gomb hatására is nullázódjon le a változó értéke!
3. Fejlesszük tovább az alkalmazást úgy, hogy az *A* gomb hatására egy véletlenszerűen meghatározott szám kerüljön bele a dobókocka nevű változóba, majd jelenítsük meg a változó értékét. Nézzünk körül a Matek kategóriában, hogy melyik blokkal lehetne véletlenszámot létrehozni!



► A számláló program egy részlete

Készítsünk egy játékot!

Van egy eszközünk, amelyen különböző ikonokat jeleníthetünk meg, illetve reagál arra, ha megrázzuk, megdöntjük valamelyik irányba. Ezen egyszerű elemek felhasználásával már akár játékokat is készíthetünk.

Tegyük fel, hogy igazságosan akarjuk eldönteni, hogy mi vagy a testvérünk ehesse meg az utolsó szelet csokit. Vagy a véletlenre szeretnénk bízni, hogy kinek kell kivinnie a szemetet a kukába. Ilyenkor akár kő, papír, olló játékot is játszhatunk. De erre a játékra akár a micro:bitet is megtaníthatjuk.

Ennek kapcsán pedig olyan új dolgokkal is megismerkedünk, amelyeket más játékok készítésénél is fel tudunk használni.

► A kő, papír, olló játék kézjelei

Gondoljuk át, hogy mi lehet egy kő, papír, olló játék algoritmusa! Valójában annyi történik, hogy véletlenszerűen kiválasztjuk, hogy három lehetőségből (kő, papír, olló) melyiket fogjuk a kezünkkel mutatni. Ez nagyon hasonló ahhoz, mintha egy 1 és 3 közötti számra gondolnánk, és a szám értékétől függően más-más kézjeleket mutatnánk.

Azt, hogy egy változó értékétől függően más-más dolog történjen, azt elágazással tudjuk leírni. De mi az az elágazás?

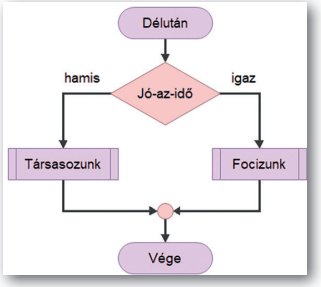
Elágazások a hétköznapi életben

Elágazásokkal a hétköznapi életben is gyakran találkozunk. Az alábbi mondatok mind ezt szemléltetik:

Ha hazaérsz 4 óráig, (**akkor**) vidd el a kutyát sétálni!
Ha tüzet észlelsz, (**akkor**) nyomd meg a tűzjelzőt!
Ha látsz egy hullócsillagot, (**akkor**) kívánj valamit!

Ezek a mondatok egyszerű elágazásra mutatnak példát. Csak arra az esetre adtunk meg utasításokat, ha a feltétel igaz. Ha hozzátesszük azt is, hogy különben mi történjen, akkor már kétirányú elágazásról beszélhetünk:

Ha délután jó az idő, (**akkor**) focizunk, **különben** társasozunk.
Ha jobban érzed magad, (**akkor**) menj át a nagyihoz, **különben** maradj itthon pihenni.



► Kétirányú elágazás folyamatábrán szemléltetve

Az **elágazásban** egy utasítás vagy egy utasításcsoport végrehajtását feltételhez tudjuk kötni. **Egyszerű elágazás** esetén az utasítások akkor hajtódnak végre, ha a megadott feltétel **igaz**.

Kétirányú elágazásban már azt is megadjuk, hogy milyen utasítások hajtódjanak végre akkor, ha a feltétel nem igaz, vagyis hamis.

Feladat

Fogalmazzunk meg mi is olyan mondatokat, amelyek egyirányú, illetve kétirányú elágazásnak felelnek meg!

Elágazások és változók az algoritmusokban

Amikor algoritmizálunk, érdemes a változókkal és elágazásokkal kapcsolatban az itt látható megfogalmazást használni.

A *kő, papír, olló* játékunk algoritmusa így alakul:

```
Program
   gondoltszám:=véletlenszám(1 és 3 között)
   Ha gondoltszám=1 akkor kő kirajzolása
       különben
         Ha gondoltszám=2 akkor papír
         kirajzolása
             különben olló kirajzolása
         Elágazás vége
   Elágazás vége
Program vége
```

Láthatjuk, hogy az 1 és 3 között véletlenszerűen meghatározott számot egy olyan **változóban tároljuk el**, amelynek neve: gondoltszám. Amikor a változónak értéket adunk, akkor nemcsak egyenlőségjelet használunk, hanem egy kettőspontot is teszünk az egyenlőségjel elé.

Ha a program algoritmusának első sorát fel kellene olvasnunk, akkor így tehetnénk meg: *„A gondoltszám változó értéke legyen egyenlő egy 1 és 3 közötti véletlen számmal.”*

Az elágazásokat a következő mondatszerű leírásokkal adhatjuk meg:

Egyszerű elágazás esetén:

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

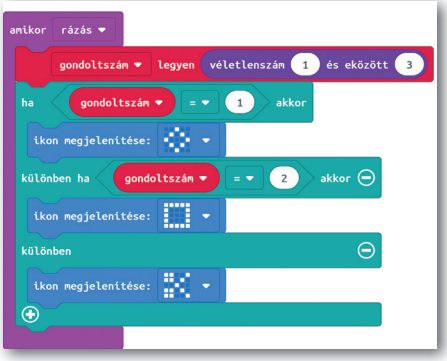
Kétirányú elágazás esetén:

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

Háromirányú elágazás esetén:

```
Ha feltétel akkor utasítás(ok)
   különben ha feltétel2 akkor utasítás(ok)
                        különben utasítás(ok)
            elágazás vége
elágazás vége
```

Játékra fel!



► A programkód a MakeCode környezetben

Feladatok

1. Próbáljuk ki az itt látható kódot! Játsszunk le pár játszmát párokban!
2. Fejlesszük tovább a játékot úgy, hogy nyilván lehessen tartani, hogy hány játszmát nyertünk! A *B* gombbal lehessen növelni a számlálót, az *A* gombbal lehessen nullázni!
3. Játsszunk tojás, fióka, sas, főnix játékot! Kezdetben mindenki tojás állapotból induljon. Aki megnyer egy játszmát, fióka lesz. Ők egymás ellen játszanak párokban, majd aki nyer, sassá változik. A sasok egymás ellen fognak játszani. Aki nyer a sasok közül, az főnixmadárrá változik. Az nyer az osztályban, aki először eléri a főnixmadár szintet!
4. Hogyan lehetne továbbfejleszteni a játékot? Ötleteljünk párokban, és készítsük el a továbbfejlesztett alkalmazást!

Gyakorlás, saját ötletek megvalósítása

Most már egy új, kézzelfogható eszközt is megismertünk, amelynek kijelzőjén számokat, szöveget, ikonokat és animációkat is meg tudunk jeleníteni. Az eszköz szenzorai segítségével érzékelni tudja a környezetét, és akár gesztusokkal (pl. rázás) is irányítani lehet.

Ezek alapján újabb izgalmas programokat lehet megvalósítani.

Alkossunk együtt!

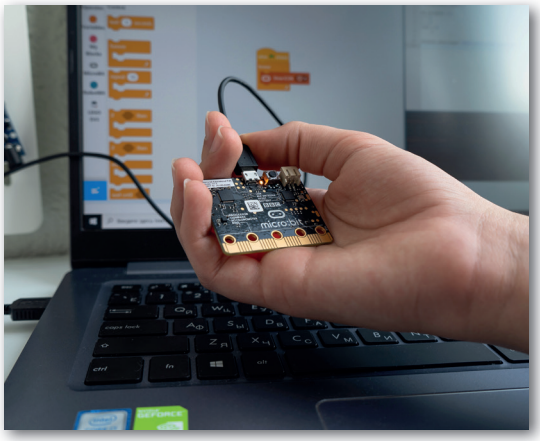
Alkossunk három-négy fős csoportokat! Gondoljuk át, hogyan lehetne továbbfejleszteni a korábban megvalósított számláló és dobókocka alkalmazást úgy, hogy minél több társasjátéknál lehessen használni segédeszközként! Gyűjtsük össze, hogy az általunk játszott és kedvelt társasjátékoknál milyen segédeszközöket kell használni, és azokat hogyan lehetne kiváltani micro:bittel!

Tervezzük meg az alkalmazást, majd valósítsuk is meg! Törekedjünk arra, hogy mindenki kapjon olyan részfeladatot, amiért ő a felelős!

Ha elkészültünk a munkánkkal, mutassuk be egymásnak a fejlesztéseteket!

Otthon pedig, amikor összeül a család egy jó kis társasjátékra, lepjük meg őket azzal, hogy a micro:bitek is szerepet kapnak a játékban!

Egyéni feladat



1. Készítsünk olyan animációt, melynek szereplője egy általunk kiválasztott állat. Az *A* és *B* gombok megnyomásakor, valamint a különböző események hatására (pl. rázás, döntés) más-más animáció játszódjon le (pl. menjen balra, jobbra, ugorjon egyet stb.).
2. Készítsünk egy programajánló alkalmazást, amely különböző tevékenységeket ajánl! Ha unatkozunk, csak megrázzuk a micro:bitet, és a megjelenő ikon alapján már akár cselekedhetünk is (pl. zenehallgatás, sport, kirándulás stb.)!
3. Készítsünk egy időjárás-előrejelző alkalmazást. Az *A* gomb hatására véletlenszerűen napsugár, esőfelhő vagy hópihe alak jelenjen meg. A *B* gomb hatására pedig jelenjen meg számként, hogy a micro:bit hőmérséklet-érzékelője hány fokot érzékel.

Páros feladat

Készítsünk egy számkitalálós játékot! A játék során azt kell megtippelnünk, hogy a micro:bit milyen számra fog gondolni. Aki eltalálja, az kap egy pontot. A játékot felváltva lehet játszani. A program úgy működjön, hogy a micro:biten a *B* gomb megnyomásával lehessen növelni a kijelzőn megjelenő szám értékét, az *A* gomb hatására pedig egy véletlen szám jelenjen meg 1 és 9 között.