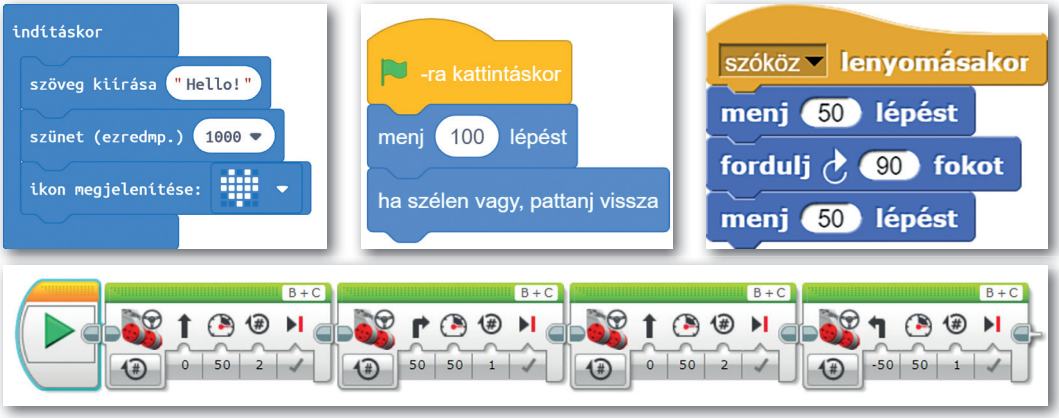
Ismerkedés a blokkprogramozási környezettel

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

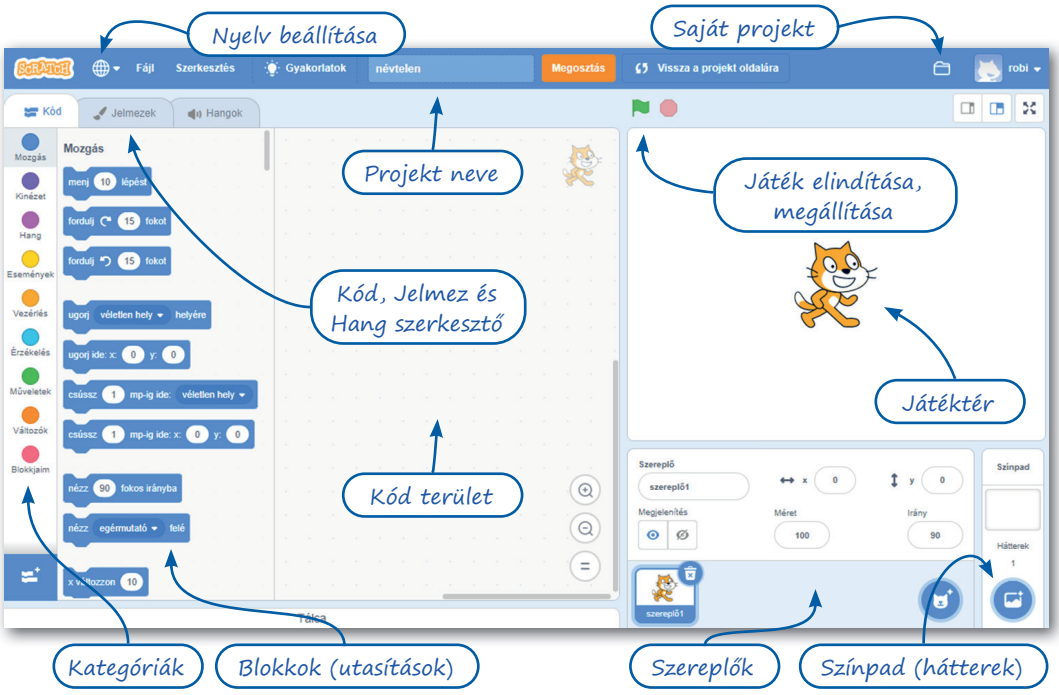
Feladat: Beszéljük meg, hogy az itt szereplő blokkoknak mi lehet a jelentésük? Vajon milyen esetben fognak ezek végrehajtódni? Mi lehet a kiadott utasítások eredménye?

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.

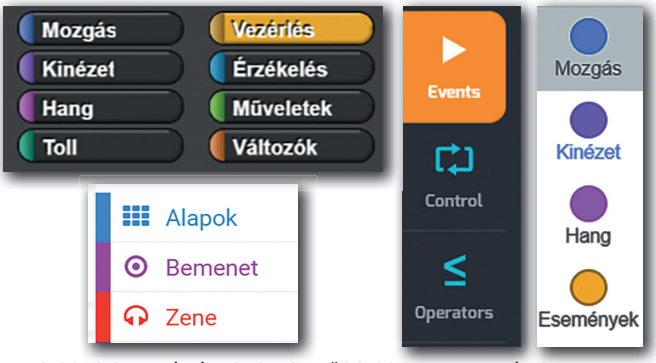


► Példa egy blokkprogramozási fejlesztői környezetre (Scratch)

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.

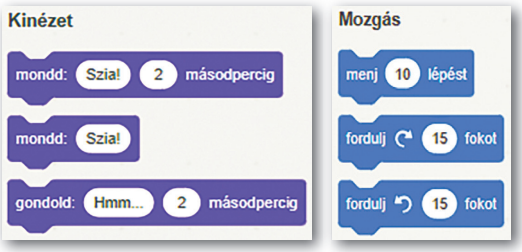


► Blokkok kategóriái a különböző blokkprogramozási környezetekben

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.

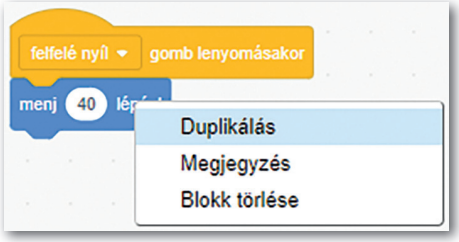


► A blokkok ugyanolyan színűek, mint a kategória (Scratch környezet)

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

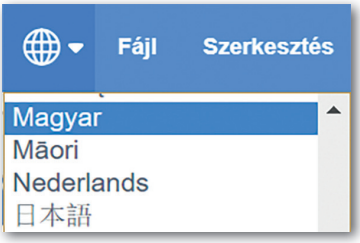
Az elkészült projektjeinket el tudjuk **menteni**, illetve a korábban elmentett projekteket **be tudjuk tölteni** a felületre. Így bármikor tovább tudjuk fejleszteni a korábban elmentett alkalmazásokat.

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)

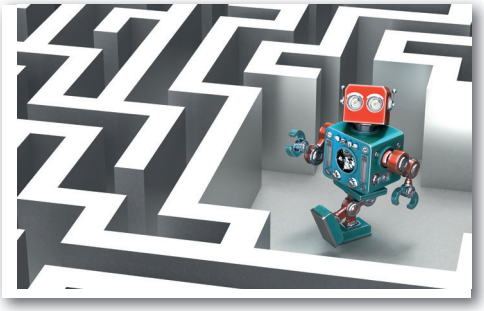
A robot és a pálya elkészítése

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.

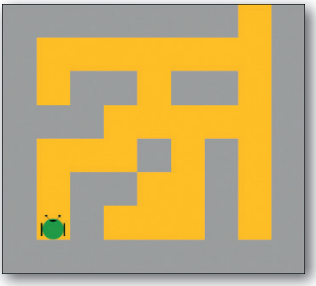
▸ A szereplő jelmezének beállítási lehetőségei a Scratch környezetben

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!

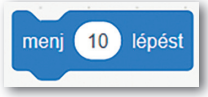
A **kódolás** művelete azt jelenti, hogy az algoritmust egy **programozási nyelv** szabályai, a rendelkezésre álló utasítások szerint átfogalmazzuk.

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.

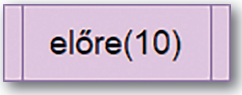
Láthatjuk, hogy az előrelépés mértékét a blokk belsejében elhelyezett ovális mezőben adhatjuk meg. Ide nemcsak egy konkrét szám kerülhet, hanem különböző műveletek eredménye is.

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

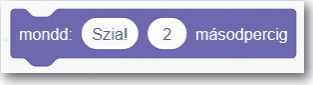
A blokk belsejébe beírt számot **paraméternek** nevezzük. A paraméter értéke most azt befolyásolta, hogy hány lépést tett előre a robot. Ha az algoritmust mondatszerű leírással adjuk meg, a paramétert sokszor **kerek zárójelek** közé tesszük. A folyamatábrában ugyanez a megadási mód látszik.



▸ 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

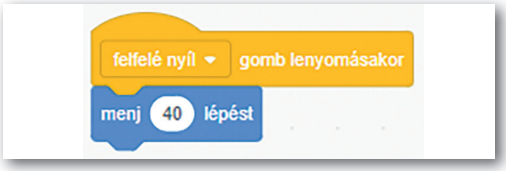
A próbálgatáson kívül máshogy is meghatározhatjuk, hogy mekkora távolságot kell előrelépnünk a megadott háttérképen. Ehhez tudnunk kell, hogy a háttérkép mekkora méretű. Ezt a méretet akár a programozási környezet is jelezheti, de akár az operációs rendszer beépített lehetőségeit használva is kideríthetjük.

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.

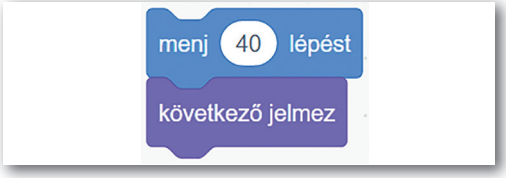
▶ A háttérkép méretének leolvasása (Scratch)

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)

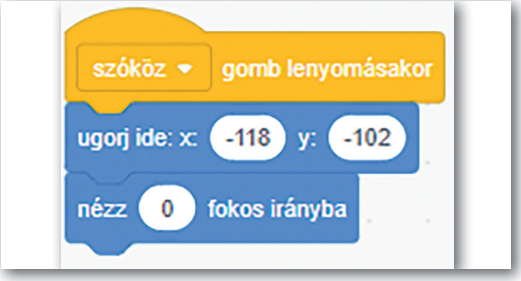
A szereplők helyét (koordinátáját) le lehet olvasni a képernyőről. Az *x* koordináta azt jelöli, hogy vízszintesen melyik pozíción helyezkedik el a szereplő. Az *y* koordináta a függőleges tengelyen lévő pozíciót jelöli.

▶ 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!

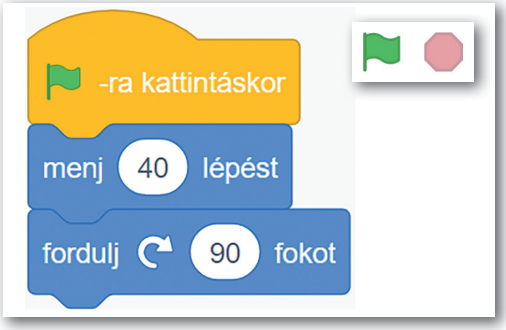
Páros feladatok

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!

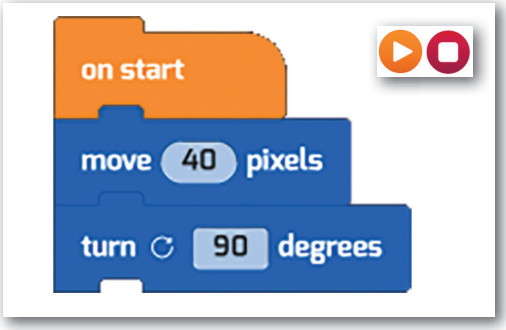
A robot irányítása utasítások segítségével

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.



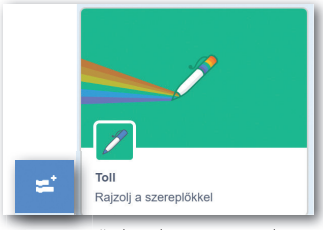
▶ A Tynker környezetben az On start (Induláskor) blokkban kell elhelyeznünk a főprogramot! A programot a háromszög ikonnal tudjuk elindítani, és a négyzet ikonnal tudjuk leállítani.

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.

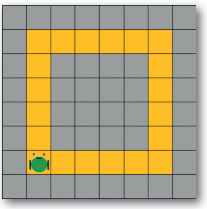
A blokkprogramozási környezetek között sok olyat találunk, amelyekben a szereplők vonalat húzhatnak maguk után. Van olyan környezet is, amelyben egy bővítmény hozzáadása után tudjuk ezt a funkciót használni!



▶ 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
```

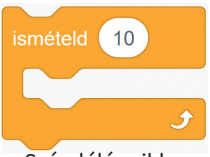
Láthatjuk, hogy az algoritmus ismétlődő elemeket tartalmaz. Négyszer ismételjük meg az öt egységgel történő előrelépést és az elfordulást. Ezt sokkal rövidebben is megfogalmazhatnánk:

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

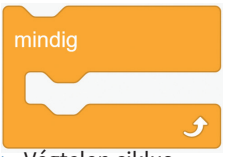
Az ismétlődő tevékenységeket úgynevezett **ciklusokba** érdemes szerveznünk. A ciklus belsejében (ezt nevezzük **ciklusmag**nak) helyezzük el azokat az utasításokat, amelyeket ismételni kell.

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.

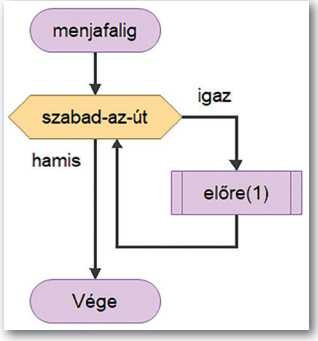
A menj a falig művelet azt jelenti, hogy addig kell előrelépkednie a robotnak, amíg a szemben lévő terület szabad. Ezt folyamatábrán szemléltetve látható, hogy a hatszög alakzat jelenti a feltételt (szabad-e az út?).

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



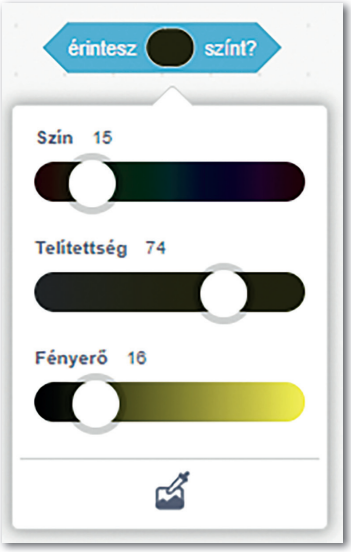
A színérzékelő használata

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.

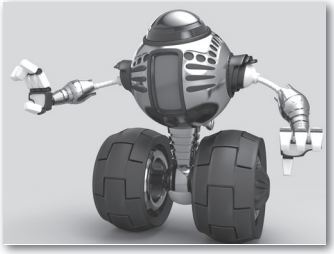
A szereplőket és a hátteret mi választhatjuk ki, illetve rajzolhatjuk meg. A játékprogramban kapjon szerepet a színérzékelő használata! Készítsük el a program algoritmusát, és csak utána álljunk neki a megvalósításnak. Törekedjünk arra, hogy mindenki kapjon olyan részfeladatot, amiért ő a felelős!

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!