

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Szakdolgozat

Programozzuk micro:biteket Python nyelven

Témavezető:

Dr. Abonyi-Tóth Andor

egyetemi adjunktus

Készítette:

Szűcs Norbert

angol nyelv és kultúra tanára –
informatikatanár
osztatlan tanári mesterszak

2019

Eredetiségi nyilatkozat

Alulírott Szűcs Norbert (név), DIL46L (Neptun-kód) ezennel kijelentem és aláírással megerősítem, hogy az ELTE angol nyelv és kultúra tanára, informatikatanár osztatlan tanári mesterszakján írt jelen diplomamunkám saját szellemi termékem, melyet korábban más szakon még nem nyújtottam be szakdolgozatként és amelybe mások munkáját (könyv, tanulmány, kézirat, internetes forrás, személyes közlés stb.) idézőjel és pontos hivatkozások nélkül nem építettem be.

Budapest, 2019. április 11.

.....

a hallgató aláírása

Témavezetői nyilatkozat

Alulírott Dr. Abonyi-Tóth Andor (témavezető neve) ezennel kijelentem és aláírással megerősítem, hogy Szűcs Norbert (hallgató neve) DIL46L (Neptun-kód) osztatlan tanárképzésbeli szakdolgozata megítélésem szerint benyújtható, annak leadásához hozzájárulok.

Budapest, 2019. április 11.

.....

a témavezető aláírása

Tartalom

Bevezető.....	5
1. alkalom (információ megjelenítése a LED mátrixon, animációk).....	10
2. alkalom (a kijelző további lehetőségei, események kezelése ciklussal).....	17
3. alkalom (pinek kezelése, LED használata, véletlenszámok)	22
4. alkalom (zene).....	28
5. alkalom (mozgás, gesztusok).....	32
6. alkalom (írástű, rádió).....	36
7. alkalom (távíró készítése)	44
8. alkalom (fájlkezelés).....	48
9. alkalom (hétköznapi tárgyak használata).....	55
10. alkalom (játékkészítés)	60
11-12. alkalom (saját projektek megvalósítása)	68
Összefoglalás	69
Irodalomjegyzék	70

Bevezető

Ez a dokumentum szakköri segédanyagként szolgál arra az esetre, hogyha a programozás tanítását érdekes, játékos, és szórakoztató módon szeretnénk elkezdni, vagy a már meglévő alaptudást elmélyíteni. Ma már gyakorlatilag nem kérdés, hogy programozást minden diáknak tanulnia kell, hiszen olyan nélkülözhetetlen készségeket fejleszt, mint a logikus gondolkodás, a figyelem, a türelem, és az önálló problémamegoldás. A tanítást már egészen fiatal korban el lehet kezdeni, erre remek eszközök állnak rendelkezésre, többek között az itt használt BBC micro:bit.

Az Egyesült Királyságban sikeresen felismerték, hogy a 21. században felnövő generációnak elengedhetetlen lesz a programozás tanulása, ezáltal a technológia és a tudomány közelebből való megismerése. Ennek érdekében küldött szét a BBC és 31 szervezet az országban 1 millió micro:bitet, hogy azokat az iskolai oktatás keretében használják¹. Ezeket az eszközöket a gyerekek saját tulajdonba kapták meg, így használatuk az iskolán kívül is megoldott. Ezen kívül az iskolák is kaptak eszközöket, és a hivatalos tantervbe is bekerült az eszköz használatának oktatása. A tanárok természetesen megfelelő felkészítést is kaptak az eszközök használatához.

Maga az eszköz tehát a BBC micro:bit nevet viseli, ami egy kisméretű ún. mikrovezérlő. Mérete ellenére rendkívül sok funkciót sikerült belepakolni². Található rajta 25 darab, egyenként programozható LED, egy 5*5-ös mátrixba rendezve. Ezeken szöveget, számokat, képeket jeleníthetünk meg. Rögtön szembetűnik még a két gomb, aminek a lenyomására tudunk reagálni programjainkban. Az eszköz alján található „fogak” a pinek, ezekre különféle eszközöket, szenzorokat, vagy akár hétköznapi tárgyakat csatlakoztathatunk. Az eszközön található LED mátrixot bemenetként is használhatjuk, ilyenkor fényerősség-mérőként szolgál. A hőmérővel a lapka aktuális hőmérsékletét tudhatjuk meg, a gyorsulásmérővel a különböző mozgásokra tudunk reagálni. Az iránytű segítségével megtudhatjuk, hogy melyik irányba nézünk, a beépített rádióval pedig vezeték nélkül kommunikálhatunk az eszközök között. Lehetőségünk van a kommunikációra számítógéppel, mobiltelefonnal, vagy egyéb eszközzel is, ezt a beépített

¹ <https://www.bbc.co.uk/mediacentre/latestnews/2016/bbc-micro-bit-schools-launch> Elérés dátuma: 2019. 02. 16.

² <https://microbit.org/guide/features/> Elérés dátuma: 2019. 02. 16.

Bluetooth segíti. Ennek hiányában USB-n keresztül tudjuk csatlakoztatni a micro:bitet a számítógéphez.

Az eszköz használatának tapasztalatait több kutatás is vizsgálta. Az egyik ilyen kutatás kiemelte, hogy a micro:bit felhasználásának több előnye is lehet a korábbi, kizárólag számítógépen történő programozáshoz képest. A motiváció növekedhet, mivel a munka eredménye nem csak a virtuális térben látható, az eszköz kézzel foghatósága segít a diákok és a téma egymásra találásában, az eszközökkel való közös munka miatt pedig a társas készségeik is fejlődnek, segíti a kooperáció kialakulását. Mivel fizikailag is létrejön egy produktum, ezért a diákok is sokkal kreatívabbak az alkotás folyamata során. Az előnyök azonban nem állnak meg az informatikaoktatásban, a többi STEM tantárgyban is megteszi hatását az effajta programozás. A diákok és a tanárok megkérdezéséből álló kutatás többek között olyan eredményekre jutott, hogy az eszköz használatának egyszerűsége miatt a diákok szívesen dolgoztak vele, mivel nem érezték lehetetlennek a programozását. Kiemelték továbbá, hogy hatalmas előny, hogy láthatják, és megfoghatják az alkotott programok eredményét, az eszköz kézzel foghatósága kulcsszerepet játszott az érdeklődés kialakításában, valamint a tanulnivaló megértésében. Élvezték, hogy lehetőségük volt kreatívnak lenni, miközben a programozás alapjait is sikeresen elsajátították [1].

Egy másik kutatás szintén az eszköz fogadtatását vizsgálta a diákok körében, két észak-írországi általános iskolában. A legszembetűnőbb eredmény a diákok pozitív reakciója volt az eszközök használatával kapcsolatban. Majdnem minden résztvevő diák azt mondta, hogy a micro:bit használata könnyű, élvezetes és hasznos készségeket fejleszt mind a programozás, mind a problémamegoldás terén. A diákok egy része megjegyezte továbbá, hogy a micro:bit által csapatként tudtak dolgozni, és együtt tudták megoldani a problémákat, ezáltal a csapatmunka terén is fejlődtek a készségeik. Szintén majdnem minden diák nagyfokú érdeklődést mutatott a programozás iránt, és megjegyezték, hogy rengeteget tanultak, nagyon rövid idő alatt. Hozzá tették, hogy nagy lelkesedéssel használták az eszközöket, és szeretnék, ha többet használnák, mint az iskolán belül, mind az iskolán kívül. A kutatás elvégzői szerint a micro:bit hatékony eszköz lehet a programozás népszerűsítésére, valamint a STEM tárgyak iránti érdeklődés felkeltésére is [2].

Egy harmadik kutatás szintén a micro:bithez való viszonyulást, valamint az alkalmazhatóságot vizsgálta, 3-8. osztályos tanulóknál. Az itt megkérdezett tanulók 90%-a jól, vagy nagyon jól érezte magát a micro:bit programozása közben. A nehézségre vonatkozó kérdésnél már jobban megoszlottak a válaszok, a diákok 57%-a találta könnyűnek az eszköz programozását, míg 32%-uk közepes nehézségűnek értékelte azt. Ebből is látszik, hogy a kihívást jelentő feladatok közben is jól érezhetik magukat a diákok. A diákok olyan feladatokat oldottak meg, amik hozzájárultak a problémamegoldó gondolkodásuk fejlesztéséhez. A feladatokat 80%-uk hasznosnak találta, ebből látszik, hogy a megfelelő feladatokkal a kompetenciafejlesztés úgy valósulhat meg, hogy azt a diákok is hasznosnak tartják. A tanulók 88%-a válaszolta azt, hogy szívesen használnák később is a micro:bitet, valamint 42%-uk el tudná magát képzelni programozóként. Ez természetesen ilyen korú gyerekeknél még csak jelzésértékű, ám azt mutatja, hogy az eszközzel való munka felkeltette az érdeklődésüket eziránt a pálya iránt [3].

Láthatjuk, hogy a micro:bit egy nagyon jó eszköz a programozás elkezdésére, vagy akár a folytatására, mondjuk egy Logo után. A következő kérdés azonban, hogy milyen nyelven programozzuk az eszközt. Alapvetően kétféle lehetőségünk van, az egyik a blokkos szerkesztőfelület, a másik, pedig valamilyen klasszikus programozási nyelv, kóddal, például a Javascript vagy a Python. Előbbinek a kisebb korosztállynál van létjogosultsága, egyszerűsége miatt. Azonban nagyobbaknál, talán már a 8. évfolyamtól felfelé is át lehet térni a kód alapú programozásra, a micro:bit ebbe a világba is tökéletes bevezető. Ebben az anyagban a Python nyelven történő programozás lesz bemutatva.

A Python napjaink egyik legnépszerűbb programozási nyelve. A TIOBE listáján³ 2019 februárjában a harmadik legnépszerűbb programozási nyelvként szerepel, a Java és a C mögött. Megállapítható tehát, hogy egy, a szakmában is használt nyelvről van szó. Természetesen a középiskolákban nem feltétlenül programozókat képzünk, így a nyelv választása nem lehet pusztán az iparági népszerűség függvénye. Meg kell tehát vizsgálnunk, hogy oktatási szempontból hogyan szerepel a Python, érdemes-e ezt tanítani egy középiskolában, akár első programozási nyelvként.

Legelső érvként a Python mellett általában az egyszerűségét szokták felhozni. Hasonlítsuk össze például a legnépszerűbb nyelvvel, a Java-val, egy Hello, World! programon keresztül!

³ <https://www.tiobe.com/tiobe-index/> Elérés dátuma: 2019. 02. 16.

Java-ban ez így néz ki:

```
1. public class HelloWorld {  
2.  
3.     public static void main(String[] args) {  
4.         System.out.println("Hello, World");  
5.     }  
6.  
7. }
```

Pythonban ugyanez mindössze ennyi:

```
|print("Hello, world!")
```

A különbség szemmel látható. Míg Java-ban rengeteg a körítés, a plusz információ, ami nagy mértékben lelassítja a tanulást, addig a Pythonban ez az egyszerű program a lehető legegyszerűbben megoldható. Nem kell egy hatalmas „csomagot” magunkkal cipelni a kezdeti lépésekhez, rögtön nekiállhatunk a kódolásnak, különösebb előismeretek vagy elméleti háttér nélkül.

A következő lényeges elem a típusok hiánya. A Pythonban nem szükséges előre megmondani, hogy egy változóba milyen típusú értékek kerülhetnek, emiatt is jóval egyszerűbbé válik a tanulás folyamata. Ellenérvként fel lehet hozni, hogy aki a későbbiekben is fog programozást tanulni, az szinte biztosan egy típusos nyelvre tér át, és így talán nehezebb dolga lesz vele. Azonban ez még nem feltétlenül indokolja azt, hogy azzal is kellene elkezdni!

A Python az egyes blokkok, összetartozó elemek jelölését egészen egyedi módon oldja meg, behúzással. A kapcsos zárójelek ugyan szépen tagolják a programot és a fejlett fejlesztőkörnyezetek igyekeznek figyelni az igazításokra, azonban az a tapasztalatom, hogy sok diák simán kikerüli, hogy a kódja formázott legyen, amiből sokszor nehezen olvasható kódok születnek. A Pythonban ez nem kerülhető ki, hiszen az igazításokkal megváltozik a program működése is. Ezáltal a diákok rá vannak kényszerítve arra, hogy szép, jól olvasható kódot írjanak.

Természetesen nem csak előnyei, hanem hátrányai is vannak a nyelvnek. Az egyik, hogy mivel szkriptnyelvről van szó, a rövidebb programok futásakor időt spórolhatunk, azonban hosszabb programok esetén már teljesítményvesztéssel kell számolnunk. Ez azonban középiskolás szinten nem kerül elő. A másik, hogy az információ nincs elrejtve,

az adattagok nyilvánosak. Ez addig nem probléma, amíg az objektumorientáltság nem jön képbe.

Megemlítenék egy kutatást, ami a Python használatát vizsgálta, mint első programozási nyelv, középiskolában [4]. A kurzus előtt felmérték a diákok korábbi tapasztalatait. Azokat a diákokat, akiknek volt már programozási tapasztalata korábban, megkérdezték a Pythonhoz való hozzáállásukról. Az eredmények a Python mellett szóltak, a diákok könnyebbnek és kellemesebbnek ítélték meg a Pythont, mint a korábban használt nyelvet. Erőteljes egyetértés mutatkozott abban, hogy a Pythont könnyebb volt megtanulni, mint a korábbi nyelvet, valamint abban is, hogy a Python szórakoztatóbb volt. Kiemelték, hogy a Python egyszerűbb, kompaktabb, egyértelműbb, mint a többi nyelv. A kurzus előtti elvárások a tanárok részéről tehát teljesültek, ezek miatt választották a Pythont, mint első nyelvet. Azonban azt a tanulmány szerzői is megjegyzik, hogy nem a nyelvet kell megtanítani, hanem a programozást, a nyelv csak egy eszköz. Mi is törekedjünk tehát erre!

A szakkör 12, egyenként 90 perces alkalomból fog állni, de természetesen az időkeretek szabadon variálhatók. Minden alkalom elején található egy áttekintő táblázat az adott óra anyagával, a javasolt munkaformával, valamint az ajánlott időkerettel. Ezután következnek az egyes egységek teljes leírással, példaprogramokkal. Külön táblázatokban, „Feladat a diákok számára” fejléccel találhatóak olyan feladatok, amik az addig tanítottak ismeretében a diákok számára önálló munkában is kiadhatók. Az utolsó két alkalmon a diákok egy saját projekten fognak dolgozni, amit aztán be is mutatnak egymásnak. A kész produktumok, a kreativitás kiélése, az eszközök kézzel foghatósága reméljük minden diákot örömmel tölt majd el!

1. alkalom (információ megjelenítése a LED mátrixon, animációk)

Tematikai egység	Alkalmazott módszerek, munkaformák	Időtartam
Az eszköz bemutatása, munkavédelmi előírások	Frontális tanári bemutató	8 perc
A szerkesztőfelület bemutatása	Frontális tanári bemutató, a szoftver legfontosabb jellemzőinek megismerése	6 perc
Hello, World! program	Közös programírás, tanári magyarázattal	8 perc
Képek megjelenítése	Tanári bemutató, majd egyéni kísérletezés Egyéni képek megjelenítése	5 perc
Animációk megjelenítése	Tanári magyarázat a listáról, animáció beépített listákkal, animáció saját képekkel Egyéni, kreatív feladat, tanári segítséggel, a munkák bemutatása egymásnak	30 perc
Gombok kezelése	Tanári bemutató és magyarázat	30 perc

1. Munkavédelmi előírások, az eszköz bemutatása

Az eszközök kiosztása előtt fontos felhívni a diákok figyelmét pár szabályra, főleg, ha még nem dolgoztak hasonló eszközzel. A munkavédelmi előírások megtalálhatóak a Programozzunk micro:biteket! kiadvány 21. oldalán [5].

Ezek után mutassuk be az eszközt a diákoknak, kitérve a rajta található gombokra, csatlakozókra, érzékelőkre. A számítógéphez való csatlakozás Bluetooth-on keresztül, vagy USB kábel segítségével történhet meg. Az utóbbi mód egyszerűbb, hiszen ilyenkor az áramellátás is megoldódik, nem kell elemet csatlakoztatnunk az eszközhöz. A csatlakozás módja hasonló pl. egy okostelefonéhoz, így a diákok számára biztosan ismerős lesz. Hívjuk fel a figyelmet arra, hogy az eszköz egy külön meghajtóként jelenik

meg, akárcsak egy pendrive. Erre a meghajtóra programokat másolva tudjuk azokat futtatni a micro:biten.

2. A szerkesztőfelület bemutatása

Az eszköz megismerése után írjuk meg közösen az első programunkat! Ehhez először szükség van a szerkesztőfelület megismerésére. Python alapon is több lehetőségünk van a micro:bit programozására. Egyik lehetőségként választhatjuk a böngészőben használható szerkesztőt⁴. Itt azonnal, telepítés nélkül neki lehet állni kódolni, ami bizonyos körülmények között jó választásnak bizonyulhat. Azonban hamar hiányérzetünk támadhat, mivel a felület jelenleg sem automatikus kódkiegészítést, sem hibakeresési lehetőségeket nem támogat. Az esetleg szintaktikus hibáinkat is csak a micro:bit kijelzőjén futó szövegből hámozhatjuk ki, ami jelentősen lelassítja a munkát. Ezen okok miatt inkább egy asztali szerkesztő javasolt. A támogatott választás a Mu névre hallgató program⁵, ami beépített micro:bit üzemmóddal segít nekünk. Ez automatikus kódkiegészítést és magyarázatot tartalmaz, emellett képes a kódot egy kattintással a micro:bitre másolni, valamint ellenőrizni tudja a kódunk helyességét is. Haladó lehetőségei közül érdemes lehet megismerni még a REPL-t és a plotter-t is, ám ezekkel ebben az anyagban nem foglalkozunk.

A diákoknak röviden mutassuk be a program legfontosabb funkciót: mód választása, új program létrehozása, a mentés és a flash-elés közti különbség, valamint a kód hibaellenőrzése gomb. Ezek után készen is állunk az első programunk megírására!

3. Hello, World! program

Ha már programoztak a diákok, biztosan ismerős számukra a *Hello, World!* kifejezés. Első programunk ezt a szöveget fogja kiírni a micro:bit LED mátrixán. Gépeljük be közösen az alábbi kódot:

```
from microbit import *  
display.scroll('Hello, World!')
```

Magyarázzuk el a diákoknak, hogy minden programot az úgynevezett importálással kell kezdenünk, itt mondjuk meg a programunknak, hogy a micro:bit kódkönyvtárát szeretnénk használni. (Megjegyzés: az importálás az anyagban található többi kódból

⁴ <https://python.microbit.org> Elérés dátuma: 2018.08.03.

⁵ <https://codewith.mu/> Elérés dátuma: 2018.08.03.

helytakarékosági okok miatt kihagyásra kerül, azonban ezt minden programunknál használjuk!) A szöveg kiírásához a `display` osztály `scroll()` metódusát használtuk, aminek az első, és jelen esetben egyetlen paraméterében a kiírandó szöveget kell megadnunk.

Egészítsük ki a programunkat úgy, hogy a szöveget ne csak egyszer írja ki a micro:bit, hanem újra és újra, két másodperces szüneteket tartva az egyes kiírások között. Ehhez egy végtelen ciklusba kell tennünk a kiírást, valamint a `sleep()` függvényt használva szüneteltetni a programot. A kód:

```
while True:
    display.scroll('Hello, World!')
    sleep(2000)
```

Nézzük meg a ciklus szintaktikáját a Pythonban! A `while` kulcsszóval kell kezdenünk, ezután jön a bennmaradási feltételünk, majd kettősponttal jelezzük, hogy a ciklusmag következik. A ciklusmag utasításai új sorba, beljebb kezdve kell írunk. Pythonban más népszerű programozási nyelvekkel ellentétben nem kapcsos zárójelek közé kell tennünk egy-egy ilyen blokkot, hanem behúzásokkal kell jeleznünk az összetartozást. A szép kód írása ezáltal elkerülhetetlen.

4. Képek megjelenítése

Ezek után folytatjuk a megismerkedést a kijelző kezelésével, amihez továbbra is a `display` osztály függvényeit használjuk. A kijelző nem csak szövegek, hanem képek megjelenítésére is használható. A következő programmal jelenítsünk meg egy mosolygó arcot a LED-ek segítségével! Ehhez a `display.show()` metódust fogjuk használni, ami képek megjelenítésére is használható:

```
display.show(Image.HAPPY)
```

A függvény paraméterében a megjeleníteni kívánt képet adjuk meg. Látható, hogy a micro:bit modul beépített képeket is tartalmaz, amiket az `Image` osztály attribútumai segítségével tudunk használni. A teljes lista megtalálható a dokumentáció `Image` osztályt bemutató részében⁶. A listát mutassuk meg a diákoknak is, kísérletezzenek a különböző képekkel, nézzék meg, hogy melyik kép pontosan hogyan jelenik meg, mit ábrázol.

⁶ <https://microbit-micropython.readthedocs.io/en/latest/image.html> Elérés dátuma: 2018.08.03.

Biztosak lehetünk benne, hogy a gyerekek kíváncsisága nem áll meg itt, saját maguk által alkotott képet szeretnének megjeleníteni a kijelzőn. Természetesen erre is van lehetőség. Minden egyes LED-hez tartozik egy 0 és 9 közötti érték, ami azt jelzi, hogy milyen fényerővel világít az adott LED. A 0 jelenti a kikapcsolt állapotot, a 9 a legfényesebbet. Ezek segítségével van lehetőségünk saját képek megjelenítésére, például kirajzolhatunk egy egyszerű repülőt, az alábbi módon:

```
plane = Image("00900:"  
              "09990:"  
              "90909:"  
              "00900:"  
              "09990")  
  
display.show(plane)
```

Hívjuk fel a diákok figyelmét a helyes kódolásra; elsőre talán bonyolultnak tűnhet a zárójelek, idézőjelek, és a kettőspontok használata, főleg, ha még nem programoztak szövegesen.

Talán a legbeszédesebb mód a fenti az egyéni képek létrehozására, azonban miután rutinosabban lettünk ebben, használhatjuk az alábbi, rövidebb formát is:

```
plane = Image("00900:09990:90909:00900:09990")
```

Próbálkozhatunk a különböző fényerősségek hatásával, például átírhatunk minden számot 1-esre.

Feladat a diákok számára

Alkossatok párokat, majd mondjátok meg egymásnak, hogy mit rajzoljon ki a párotok a LED mátrixon, a fenti módszert használva!

5. Animációk megjelenítése

A micro:bit kijelzőjén animációkat – képek sorozatát – is nagyon egyszerűen megjeleníthetjük. Ehhez mindössze a képsorozatra van szükségünk, ami a Pythonban nagyon jól megadható az ún. listával. Ha a diákok még nem találkoztak a lista, vagy a tömb fogalmával, magyarázzuk el nekik! Hozzunk hétköznapi példákat, például egy bevásárlólista, egy dolgozat jegyeinek a listája stb. A lista a Pythonban azonban bármilyen típusú elemekből állhat, így képekből is. Próbáljunk meg egy ilyen listát animáció formájában megjeleníteni. Az eddig használt `Image` osztályban két beépített

lista is van, név szerint az `Image.ALL_CLOCKS`, és az `Image.ALL_ARROWS`. Csináljunk egy olyan programot, ami folyamatosan megjelenít egy járó órát!

```
display.show(Image.ALL_CLOCKS, loop=True, delay=100)
```

A korábban már megismert `show()` függvény egy új formáját használjuk. Az első paraméterben valamilyen iterálhatót kell megadnunk, aminek az egyes elemeit fogja a függvény megjeleníteni. A `loop` paramétert igazra állítva végteleníthetjük az animációt, a `delay` paraméter pedig az egyes állapotok közötti időt adja meg, milliszekundumban. Hívjuk fel a figyelmet rá, hogy a `loop` és a `delay` paramétereknél nem elég csak az értékeket megadni, a nevüket is fel kell tüntetni! Ennek oka, hogy bizonyos paramétereknek alapértelmezetten is van értékük, ezért, ha valamelyiket át szeretnénk állítani, tudnia kell az értelmezőnek, hogy melyik paraméter értékét állítjuk át.

Feladat a diákok számára

Készítsetek animációt, ami a különböző nyilakat jeleníti meg a kijelzőn! Használjátok az `Image` osztály `ALL_ARROWS` listáját! Az animáció ne ismétlődjön, de legyen negyed olyan gyors, mint az órás! A végén töröljétek a kijelzőt a `clear` paraméter segítségével!

Ezután mutassuk meg a diákoknak, hogy hogyan tudnak létrehozni saját maguk listát, általuk megadott értékekkel, hogy aztán az elemeit felhasználják saját animáció készítéséhez. Példaként a korábban már kirajzolt repülő fog „elrepülni”:

```
plane1 = Image("00900:09990:90909:00900:09990")
plane2 = Image("09990:90909:00900:09990:00000")
plane3 = Image("90909:00900:09990:00000:00000")
plane4 = Image("00900:09990:00000:00000:00000")
plane5 = Image("09990:00000:00000:00000:00000")

plane_list = [plane1, plane2, plane3, plane4, plane5]

display.show(plane_list, delay=200, clear=True)
```

Feladat a diákok számára

Készítsetek egy rövid animációs „filmet” saját képekből, esetlegesen szöveggel kísérve! A képeket tároljátok egy listában! Használjátok ki a `display.show()` metódus különböző paramétereit! Ne feledkezzetek meg arról sem, hogy a fényerősség is állítható!

6. Gombok kezelése

A `display` osztály további függvényeinek megnézése előtt tanuljuk meg, hogy hogyan kezelhetjük a `micro:bit` előlapján található két gombot. Ezekkel gyakorlatilag az eseményvezérelt programozás alapjaiba vezethetjük be a diákokat, hiszen a gombok lenyomására fog lefutni egy adott kódrészlet. A másik fontos különbség az eddigi kódokhoz képest, hogy most már nem csak kimentet állít elő a program, hanem reagálnia kell valamilyen bemenetre is. Érdekes erre is felhívni a diákok figyelmét. Gépeljük be a következő, pár soros kódot, majd nézzük át a diákokkal, hogy miket figyelhetünk meg benne.

```
sleep(10000)
display.scroll(str(button_a.get_presses()))
```

Az első sort már a diákoknak is ismerniük kell: 10 másodpercig megállítja a program futtatását. A `display` osztály `scroll()` metódusát már szintén ismerik. Ennek a paraméterében találhatóak az új dolgok. Kezdjük legfelülről! A `button_a` osztály kezeli a `micro:bit` „A” jelű gombját. Ennek az osztálynak az egyik metódusa a `get_presses()`, ami visszaadja, hogy hányszor lett lenyomva az adott gomb a program futásának kezdete óta, emellett lenullázza ezt az értéket. Ugyanígy lehet a „B” gombot is kezelni, a `button_b` osztállyal. A blokkos szerkesztővel⁷ ellentétben itt nincs külön opció a két gomb egyszerre történő kezelésére, ezt logikai operátorokkal kell megoldanunk. Erre egy későbbi feladat során még visszatérünk.

Ha megkaptuk, hogy hányszor lett lenyomva a gomb, már csak az a feladatunk, hogy kiírjuk ezt a kijelzőre. A `scroll()` metódus egy szöveget (stringet) vár paraméteréül, a `get_presses()` pedig egy számot ad vissza, így ezt át kell alakítanunk. Erre szolgál az `str()` függvény, aminek a paraméterében adhatjuk meg a konvertálni kívánt számot. Jelenleg ez a függvény akár el is hagyható, mivel, ha számot adunk meg, a `scroll()` metódus automatikusan konvertál, de a későbbiekben mindenképpen hasznos tudás lehet a típuskonverzió fogalmának bevezetése (vagy lehet, hogy már elő is fordult korábban). A metódusok, függvények egymásba ágyazása elsőre bonyolultnak tűnhet a diákok számára, legyünk biztosak benne, hogy megértették, hiszen ez egy nagyon gyakran használt lehetőség a programozás világában.

⁷ <https://makecode.microbit.org/> Elérés dátuma: 2018.08.03.

Pihenésképpen versenyeztessük meg a diákokat, nézzük, hogy ki tudja a legtöbbször lenyomni a gombot ebben a 10 másodpercben!

Feladat a diákok számára

Módosítsátok a az előző programot úgy, hogy a 10 másodperc letelte után az jelenik meg, hogy összesen hányszor nyomtuk le a két gombot! Ezután módosítsd úgy, hogy csak 5 másodpercünk legyen a gombok nyomkodására!

2. alkalom (a kijelző további lehetőségei, események kezelése ciklussal)

Tematikai egység	Alkalmazott módszerek, munkaformák	Időtartam
Ismétlés	Egyéni munka, majd a megoldás megbeszélése közösen	10 perc
A <code>display</code> osztály további metódusai	Frontális tanári magyarázat, közös programírás, önálló munka, játék	50 perc
Események kezelése ciklussal	Közös programírás, tanári magyarázattal	30 perc

1. Ismétlés

Feladat a diákok számára

Készítsetek programot, ami minden ötödik másodpercben kiírja, hogy hányszor nyomtuk le a „B” gombot az előző öt másodperces szakasz alatt!

A megoldás:

```
while True:
    sleep(5000)
    display.scroll(button_b.get_presses())
```

2. A `display` osztály további metódusai

A második alkalmon folytatjuk az ismerkedést a `display` és az `Image` osztályokkal. Elsőként nézzük meg, hogy milyen további metódusokat kínál a `display` osztály a kijelző kezelésére!

Ennek az osztálynak a segítségével lehetőségünk van a kijelző egyes LED-jeinek külön-külön való kezelésére is. Ebben az esetben minden egyes LED-et koordinátákkal azonosíthatunk, a mátrix típushoz hasonló módon. A diákok a tömb fogalmát már ismerik, ennek alapján magyarázzuk el nekik, hogy mi az a mátrix, kitérve az x és y koordinátákra. A micro:bit kijelzőjén a bal felső sarokban található a 0,0 koordinátájú LED, a jobb alsó sarokban pedig a 4,4 koordinátájú. Ismétlésképpen hívjuk fel a figyelmüket arra is, hogy az indexelés 0-tól kezdődik! Példaként rajzoljunk fel egy 5*5-

ös táblázatot a táblára, ami a micro:bit kijelzőjét szimbolizálja, majd ezen mutassunk rá cellákra, és kérdezzük meg a diákokat, hogy az melyik koordinátájú LED lenne a micro:biten.

Ennek a tudásnak a birtokában máris használni tudjuk a `display` osztály `get_pixel()` és `set_pixel()` metódusait. Előbbi visszaadja a paraméterekben megadott indexű LED világosságértékét (emlékezzünk, hogy ez egy 0 és 9 közötti szám), utóbbival pedig be lehet állítani a paraméterekben megadott LED világosságértékét egy, szintén a paraméterekben megadott értékre. Használjuk most ezeket egy programban, ahol egy új vezérlési szerkezetet is láthatunk! Ha korábban már programoztak a diákok, biztosan szóba került az elágazás fogalma. Röviden ismételjük át ezt velük még a kód leírása előtt, majd gépeljük be az alábbi sorokat, ahol láthatjuk, hogy a Python nyelvben hogyan is kell használni az elágazást:

```
if display.get_pixel(1,1) == 0:
    display.set_pixel(1,1,9)
    sleep(2000)
display.clear()
```

Mint láthatjuk, az elágazás itt is az `if` kulcsszóval kezdődik, ezután kell írunk feltételt, zárójelet nem használva. A feltétel után kettősponttal jelezzük, hogy az elágazás törzse következik; ezt új sorba, beljebb kezdve kell írunk. Vegyük észre, hogy a szintaktika teljesen megegyezik a már használt `while` ciklussal! Amennyiben nem programoztak még C típusú nyelvben a diákok, akkor érdemes felhívni a figyelmüket az egyenlőség vizsgálatára használt dupla egyenlőségjelre, szembe állítva az értékadáshoz használt szimpla verziójával. A behúzásokból láthatjuk, hogy a `sleep()` függvény már nem része az elágazásnak, az mindenképpen lefut, ahogy a kijelző törlésére szolgáló `clear()` metódus is.

Ezek után gyakoroljuk a LED mátrix kezelését. Gondoljunk most úgy a micro:bit kijelzőjére, mintha ablakok lennének! Készítsük el egy emeletes ház szimulációját, amin azt láthatjuk, ahogy az egyes lakásokban a fények kigyúlnak és elalszanak! Illusztrációként bemutathatunk egy videót a jelenségről⁸. Időtakarékossági szempontok miatt használjuk csak a kijelző első három oszlopát! Ezt az 5 pixel magas és 3 pixel széles

⁸ <https://www.youtube.com/watch?v=M-34aBULH8g> Elérés dátuma: 2018.08.03.

területet fogjuk a háznak tekinteni. Az eddigi ismeretek alapján a feladat kiadható a diákoknak önálló megoldásra.

Feladat a diákok számára

A ház földszintjén üzletek találhatók, ezek mindig világítanak. Ezek után kapcsolj fel pár LED-et (szabadon választott koordinátákkal), majd kapcsolj is le közülük néhányat! Módosítsd úgy a programot, hogy először az első emelet lakásaiban kapcsoljanak fel a fények balról jobbra, majd a második emeleten jobbról balra! A lámpák lekapcsolása ugyanilyen sorrendben történjen meg!

A következő két metódus a kijelző ki és bekapcsolására szolgál, ezeket a beszédes `on()` és `off()` nevekkkel látták el. Feldolgozásuk történhet bármilyen önálló feladat keretében, de használható példa lehet egy bombás játék megalkotása. A játék lényege, hogy a diákok körbe állnak, egymásnak adogatva a „bombát”, ami jelen esetben egy `micro:bit` lesz. A bombát akkor lehet továbbadni, hogyha mondtak egy fogalmat a megbeszélt témában. Az érettségi tételek gyakorlására kiváló játék lehet (pl. fogalmak a háttértárak témaköréből). Akinél „felrobban a bomba” (megjelenik a kijelzőn a bomba alakzat), felrobbant, és kiáll a játékból. Az utolsóként életben maradt játékos nyer. A játékhoz szükséges programot a diákok önálló feladat keretében is megcsinálhatják:

Feladat a diákok számára

A program elindulásakor rajzolódjon ki valamilyen bombához hasonló alakzat a kijelzőn, ez villanjon fel háromszor, majd kapcsoljátok ki a kijelzőt a `display.off()` metódussal. Csoportlétszámtól függően körülbelül fél-egy perc elteltével a kijelző kapcsoljon be (a kijelzőn a bomba lesz látható).

Játszható akármilyen témakörben, akár hétköznapi is lehet, akár angolul stb. A lehetőségek száma gyakorlatilag végtelen!

Egy másik példa lehet erre a két metódusra egy zsákbamacska készítése.

Feladat a diákok számára

A micro:bit kijelzőjére rajzoljatok valamilyen egyedi ikont, valami olyan témában, ami rátok jellemző. Az A gombra a kijelző kapcsoljon ki, a B gomb megnyomására pedig kapcsoljon be! Miután a program fut, kapcsoljátok ki a kijelzőt, majd minden tegye bele egy zsákba a micro:bitjét! Ezután a zsákból mindenki húz egy micro:bitet, bekapcsolja a kijelzőt, majd a rajta található ikon segítségével ki kell találnia, hogy kinek a micro:bitjét húzta ki.

A fenti feladat jól használható például párokba osztásnál: ilyenkor csak minden második diák micro:bitje kerül bele a zsákba, a diákok másik fele pedig kihúzza azokat. Akinek az eszközt húzta, az lesz a párja.

3. Események kezelése ciklussal

Gyakran van szükségünk arra, hogy a programunk várjon valamilyen esemény bekövetkeztére, például egy gombnyomásra. Ehhez ciklusba szervezünk egy kóddarabot, ami megmondja, hogyan kell reagálni a várt eseményekre. A korábban már használt `while` ciklust fogjuk erre a célra használni, ami a ciklusfeltétel teljesülése esetén futtatja a ciklusmagot. Ennek demonstrálására nézzünk egy egyszerű kódot:

```
while running_time() < 10000:  
    display.show(Image.ASLEEP)  
  
display.show(Image.SURPRISED)
```

A ciklus feltételében a `running_time()` függvénnyel kérdezzük le a program elindítása óta eltelt időt. Természetesen ez is milliszekundumban adja meg az eredményt. Ha még nem telt el 10 másodperc az elindítás óta, a micro:bit „alszik”. Amint letelik ez az idő, a micro:bit meglepődött arccal „felébred”. Ismét hívjuk fel a diákok figyelmét a szintaxisra, különös tekintettel a behúzásokra, hiszen a Python nyelvben csakis ezen múlik, hogy mely sorok alkotnak egy blokkot a kódon belül.

Ezek után áttérhetünk első eseményünkre, a gombnyomásra. Ha azt akarjuk, hogy a micro:bit reagáljon valamit a gombnyomásunkra, egy végtelen ciklus belül kell vizsgálnunk azt, hogy le van-e nyomva éppen a gomb. Erre szolgál a `Button` osztály `is_pressed()` metódusa. Készítsünk most egy kezdetleges virtuális állatot, ami folyamatosan szomorú, kivéve amikor az „A” gomb le van nyomva, ilyenkor vidám lesz.

Amennyiben a „B” gombot nyomjuk le, az állat elalszik (letörlődik a kijelző, és véget ér a program). Az ehhez szükséges kód:

```
1. while True:
2.     if button_a.is_pressed():
3.         display.show(Image.HAPPY)
4.     elif button_b.is_pressed():
5.         break
6.     else:
7.         display.show(Image.SAD)
8.
9. display.clear()
```

Vizsgáljuk meg sorról sorra a programot! Az első sorban indítjuk a végtelen ciklust, ami „hallgatózik” az események (a gombok lenyomása) után. A ciklusmagban elágazással vizsgáljuk, hogy az „A” gomb le van-e nyomva éppen. Amennyiben igen, a kijelzőn egy boldog arcot jelenítünk meg. Az `elif` kulcsszó felel meg a más nyelvekben `else if`-ként megszokott különben ha ágnak. Ha az „A” gomb épp nincs lenyomva, de a „B” igen, lépünk ki a ciklusból, és folytatódjon a program futása. Az éppen futó kódblokkból tehát – más nyelvekhez hasonlóan – a `break` kulcsszóval tudunk kiugrani. Ha egyik gomb sincs lenyomva, a szomorú arcot rajzoljuk ki a kijelzőre. Ezt tehetjük a különben ágba, hiszen ide valóban csak akkor jutunk, ha egyik gomb sincs lenyomva. A ciklus után letörljük a kijelzőt (az állat elalszik), majd véget ér a program.

A második alkalom végén tartsunk egy kis összegzést: tudjuk, hogyan töltsünk rá programot a micro:bitre, milyen lehetőségeink vannak a kijelzőt illetően, hogyan használjuk a `while` ciklust, az elágazást, valamint a gombokat is tudjuk kezelni.

Ha esetleg marad idő a foglalkozás végén, a virtuális állat tetszés szerint továbbfejleszthető!

3. alkalom (pinek kezelése, LED használata, véletlenszámok)

Tematikai egység	Alkalmazott módszerek, munkaformák	Időtartam
Ismétlés	Egyéni munka, majd a megoldás megbeszélése közösen	10 perc
Pinek, ki- és bemenet	Frontális tanári magyarázat, közös programírás, önálló munka	15 perc
LED rákötése az eszközre	Bemutató, önálló munka	10 perc
LED villogtatása	Önálló munka	15 perc
A random modul bemutatása	Frontális tanári magyarázat	5 perc
Sorsolóprogram	Közös programírás	5 perc
Dobókocka készítése	Közös programírás	5 perc
Véletlenszám előállítási módok	Frontális tanári magyarázat, csoportszintű megbeszélés, pármunka	15 perc
A véletlenszámok előállításának elve	Frontális tanári magyarázat, megbeszélés	10 perc

1. Pinek, ki- és bemenet

A micro:bit alján fémcsíkok találhatóak, amiktől az eszköz úgy néz ki, mintha fogai lennének. Ezek az ún. ki- és bemeneti (I/O) pinek. Összesen 19 pinto használhatunk, amikből 5 darab nagyobb, mint a többi. Ezen öt pin mindegyikén található egy jelölés, ezek sorban a 0, az 1, a 2, a 3V, és a GND (ground, földelés). Ezekre krokodilcsipeszek segítségével különféle eszközöket csatlakoztathatunk. Kódszinten minden pin egy objektummal van reprezentálva, aminek a neve `pinN`, ahol N a pin száma. Például a 0-s pinnel való munkához a `pin0` nevű objektumot használhatjuk. Minden ilyen objektumnak vannak metódusai, ám ezek nem egyeznek meg minden pin esetén, hiszen különböző célokra használhatjuk őket. Nézzünk egy rövid példát a bemenet kezelésére!

A legegyszerűbb azt vizsgálunk, hogy meg van-e érintve egy adott pin. Szimuláljuk le azt, hogy megcsikizzuk a micro:bitet!

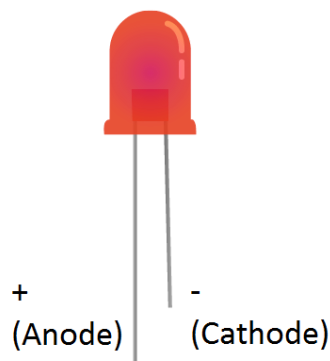
```

while True:
    if pin0.is_touched():
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)

```

A végtelen cikluson belül ezúttal azt vizsgáljuk, hogy a 0-s számú pint érintjük-e éppen. Erre a célra az `is_touched()` metódust használhatjuk. Amikor érintjük a pint, a micro:bit mosolyog, különben szomorú. A program kipróbálása a következőképpen zajlik: egyik kezünkkel fogjuk meg a GND jelű pint, majd a másik kezünkkel érintsük meg a 0-s számút! Ha minden jól megy, látni fogjuk kirajzolódni a mosolygó arcot. Ezzel megvalósítottunk egy nagyon alapvető bemenet-kezelést, ám a pinek segítségével rengeteg egyéb lehetőségünk is van, amire a későbbiekben még nézünk példákat.

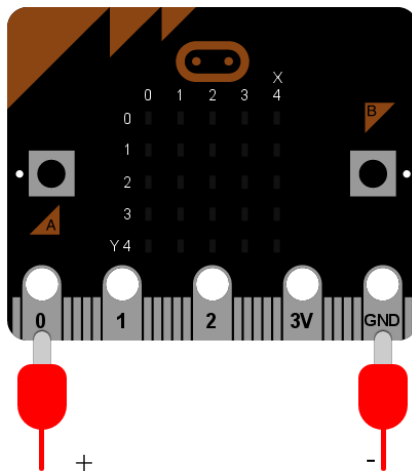
Most pedig nézzük meg a – talán – legegyszerűbb példát a kimenet kezelésére. Ehhez valamilyen eszközt kell csatlakoztatnunk a pinekre, az én választásom ezúttal egy LED-re esett. Ahogy az 1. ábrán láthatjuk, egy LED-nek két kivezetése van, a hosszabb a pozitív (anód), a rövidebb a negatív (katód). A LED kizárólag akkor fog helyesen működni, ha az egyes kivezetéseket jó helyre kötjük be.



1. ábra A LED és kivezetései⁹

A LED-et krokodilcsipeszek segítségével tudjuk rákötni az eszközre. A pozitív kivezetést kössük a 0 pinre, a negatívát pedig a GND jelzésűre, ahogy az az ábrán is látszik!

⁹ <http://www.makerspace-uk.co.uk/wp-content/uploads/2016/07/led-annotated.png> Elérés dátuma: 2018.08.04.



2. ábra A LED összekötése a micro:bittel¹⁰

Ha esetleg nincs krokodilcsipeszünk, alufóliával és ragasztószalaggal is dolgozhatunk. Az alábbi ábrán látható egy megoldás erre a problémára, amin egy fülhallgató van rákötve a micro:bitre. Ez természetesen kényelmetlenebb, mint az előző módszer, ám eszköz hiányában remek helyettesítő megoldás.



3. ábra Alufólia használata krokodilcsipeszek helyett¹¹

¹⁰ http://www.makerspace-uk.co.uk/wp-content/uploads/2016/07/LED-no-wires_bb.png Elérés dátuma: 2018.08.04

¹¹ <https://pxt.azureedge.net/blob/a2f8a1af1a5c4122df2bd13d1e9fd314fd295ef5/static/mb/device/croclips/foilcircuit.jpg> Elérés dátuma: 2018.08.04.

Amint készen vagyunk az összekötéssel, ki is próbálhatjuk azt egy egyszerű programmal. Gépeljük be az alábbi sort a LED felkapcsolásához.

```
pin0.write_digital(1)
```

Ez ugyan csak pár másodpercig izgalmas, ám hamar rá fogunk jönni, hogy a pinek segítségével szinte végtelen számú ötletet meg lehet valósítani!

Gyakorlásképpen bonyolítsuk egy kicsit a fenti programot!

Feladat a diákok számára

Oldd meg, hogy másodpercenként kétszer villanjon fel a LED! Egy felvillanás ideje legyen 20 ms.

A diákok számára ez egy nem feltétlenül gyorsan, de megoldható feladat, a számokkal való kísérletezés során hamar rá lehet jönni a megfelelő értékekre. Segítségképpen álljon itt a megoldás:

```
while True:
    pin0.write_digital(1)
    sleep(20)
    pin0.write_digital(0)
    sleep(480)
```

Ha esetleg nem lenne elég látványos az eredmény, bátran kísérletezzünk az időértékekkel!

Ezáltal készítettünk egy jelzőfényt, ami sok további programnak lehet a kiindulópontja.

2. A random modul, véletlenszámok előállítása, az előállítás elvei

Néha szükségünk van arra, hogy valami ne legyen előre eldöntve, hanem véletlenszerűen történjen meg. Ehhez segítségül hívhatjuk a Pythonban található random modult. Például, ha a csoport tagjai közül akarunk kiválasztani valakit véletlenszerűen, egy ilyen programot kell írunk:

```
import random

names=["Attila", "Bence", "Csilla", "Dani", "Emese", "Fruzszi"]

display.scroll(random.choice(names))
```

Ezúttal nem elég a minden program elején használt általános import utasítás, hanem importálnunk kell a `random` modult is. Az ebben található `choice()` módszerrel tudunk egy lista elemeiből véletlenszerűen kiválasztani egyet. Miután kiválasztottunk egy nevet, a korábban használt `scroll()` módszerrel kiírjuk azt a képernyőre. Ilyen egyszerűen készíthetünk egy sorsolóprogramot.

Legtöbbször a `random` modult inkább számok előállítására használjuk. A véletlenszámok használata nagyon gyakori például a játékokban, elég csak szimplán a dobókockára gondolni, ami számtalan játék alapeleme. Készítsünk most el egyet a `micro:bit`-en!

```
import random  
  
display.show(random.randint(1, 6))
```

Az eszköz minden újraindításnál kiír egy 1 és 6 közötti számot, a `randint()` módszer segítségével. Láthatjuk, hogy ez két számot vár paraméteréül, annak az intervallumnak az alsó- és felső határát, amiből szeretnénk véletlenszámokat visszakapni (a két határ is beletartozik az intervallumba). A függvény ezután, ahogy a neve is jelzi, egy véletlenszerűen kiválasztott egész számot ad vissza. Hívjuk fel a diákok figyelmét a függvény nevében található `int` szóra, és ha korábban nem találkoztak ezzel, mondjuk el nekik, hogy az `int`, vagy `integer` a különböző programnyelvekben az egész szám típusnak felel meg.

Ha esetleg 0 és N közötti számokra lenne szükségünk, használhatjuk a `random.randrange()` módszert is. Ez 0 és a paraméteréül megadott szám közötti véletlen egész számot ad vissza, azonban ennél a paraméterben megadott szám nem lesz része az intervallumnak, amiből a szám kikerül. Erre a különbségre érdemes nyomatékosan felhívni a figyelmet!

Miután az egész számok előállításával már boldogulunk, mindenképpen szót kell ejteni arról az esetről is, amikor valós számokra lenne szükségünk. Hívjuk ezeket a számokat az életkori sajátosságoknak megfelelően így, esetleg tizedestörtnek, vagy csak szimplán írjunk fel a táblára egy példát, hogy demonstráljuk ezeknek a számoknak az alakját. Ilyen számok előállítására a `random.random()` módszert tudjuk használni, ami 0.0 és 1.0 közötti valós számokat ad vissza (a két határ is beleértendő az intervallumba). Amennyiben ennél nagyobb számra volna szükségünk, adjuk hozzá az eredményt a `random.randrange()` módszer által visszaadott számhoz, ahogy ezt az alábbi program

szemlélteti. A program megmutatása előtt a csoport szintjétől függően hagyjuk a diákokat gondolkodni a probléma megoldásán! Akár páros munkát is alkalmazhatunk, hogy egymást segítve jöhessenek rá a megoldásra.

```
import random  
  
answer = random.randrange(100) + random.random()  
display.scroll(answer)
```

A program egy 0 és 100 közötti valós számot állít elő, majd kiírja ezt a kijelzőre.

Érdekes lehet kicsit többet beszélni a véletlenszámok előállításának elvéről, hiszen ez Pythonban egy egyszerű példán keresztül bemutatható. Amit hasznos tudni a véletlenszámokról az az, hogy valójában nem véletlenek, csak úgy tűnhetnek. Ezeknek az előállítása nem más, mint véletlenszerű eredmények számolása egy bizonyos kezdeti (ún. `seed`) értékből. Ez az érték valami olyan, ami nem lehet kétszer ugyanaz, legtöbbször a rendszeridő. Ez kiegészülhet esetleg még pár dologgal, a nagyon nagy valószínűséggel jó eredmény érdekében. Hogyha megismételhetően véletlen eredményt szeretnénk előállítani, mi is beállíthatjuk a `seed` értékét egy tetszőleges számra. Ugyanazzal a `seed` értékkel minden egyes programlefutáskor ugyanazt a számsorozatot kapjuk vissza a `random` metódusait meghívva. Nézzünk egy példát rá a korábbi, dobókockát szimuláló kódot kiegészítve:

```
1. import random  
2.  
3. random.seed(1337)  
4. while True:  
5.     if button_a.was_pressed():  
6.         display.show(random.randint(1, 6))
```

A 3. sorban adtuk meg a kiindulási értéket, ami jelen esetben 1337. Ezután akárhányszor megnyomjuk az „A” gombot, kapunk egy új egész véletlenszámot. A csavar a dologban, hogy akármikor elindítjuk a programot, a kapott számsorozat ugyanaz lesz (2, 3, 5, 1, 1...). Ebből is látszik, hogy a véletlenszám generálás valójában nem teljesen véletlenszerű, hanem csupán valamilyen műveletsorozat segítségével kiszámolt értékeket kapunk vissza.

4. alkalom (zene)

Tematikai egység	Alkalmazott módszerek, munkaformák	Időtartam
A zene modul	Frontális tanári magyarázat	5 perc
Fülhallgató csatlakoztatása	Egyéni munka, tanári monitorozással	5 perc
A beépített dallamok kipróbálása	Egyéni munka, megbeszélés	15 perc
Hangok használata	Frontális tanári bemutató	10 perc
Gyerekdal leírása	Közös programírás	15 perc
Kedvenc dal leírása	Önálló munka	15 perc
Szirána készítése	Közös programírás	10 perc
Kísérletezés, egyéni dallamok létrehozása	Egyéni munka, pármunka	15 perc

1. A zene modul, fülgallható csatlakoztatása

A micro:bit és a MicroPython zenei képességei is kiemelkedőek. Nagyon egyszerűen írhatunk dallamokat, játszhatunk le különböző hangeffekteket. Mivel az eszközön magán nincs hangszóró, ezért ennek, vagy egy fülhallgatónak a csatlakoztatásáról nekünk kell gondoskodnunk. Ehhez ismét a pinekhez kell nyúlnunk. A róluk szóló fejezetben látható módon, a 0 és a GND pinek segítségével csatlakoztathatjuk a hang kibocsátására használt eszközünket oly módon, hogy a krokodilcsipeszeket a jack dugó alsó és felső részéhez csatlakoztatjuk. Segítségünkre lehet az előző fejezetben látható kép, ahol krokodilcsipeszek helyett alufólia csíkokkal lett megoldva a csatlakozás, de a csipeszekkel is ugyanígy járjunk el!

2. A beépített dallamok kipróbálása

Ha csatlakoztattuk egymáshoz a két eszközt, teszteljük le, hogy sikerült-e! Gépeljük be az alábbi sorokat:

```
import music  
music.play(music.NYAN)
```

Ha minden jól ment, hallanunk kell a megadott dallamot. Ez a zene modul egy beépített dallama, amiből több is áll rendelkezésünkre. A teljes lista megtalálható a dokumentáció zene modullal foglalkozó oldalán.¹² Vegyük észre, hogy a zene modult a randomhoz hasonlóan külön is importálnunk kell, ha használni szeretnénk, nem elég a már eddig használt import.

Feladat a diákok számára

Próbáld ki a zene modul beépített dallamait! A fenti kódban írt át a dallam nevét, majd hallgasd meg őket! Ezután beszéljétek meg, kinek melyik a kedvence! Hogyan használhatóak ezek a dallamok valamilyen célra, például jelzések, nyomok?

3. Hangok

De nem csak a beépített dallamok állnak rendelkezésünkre, sajátot is nagyon egyszerűen írhatunk. Minden hangnak van egy neve (például C# vagy F), egy oktávja (milyen magasán van az adott hang), és hossza (mennyi ideig tart a lejátszása). Az oktávot számmal jelöljük, 0 a legalacsonyabb oktáv, a 4-esben van a közepső C, a 8-as pedig a legmagasabb amire szükségünk lehet, hacsak nem kutyáknak próbálunk zenét írni. A hosszt szintén egy szám jelöli, minél nagyobb, annál tovább tart az adott hang. Ezek arányosak egymással, például a 4-es hossz pontosan kétszer annyi ideig tart, mint a 2-es. Szünet is rendelkezésünkre áll, ezt R betűvel jelöljük (rest). Minden hang egy stringgel van jelölve, az alábbi módon:

`HANG[oktáv][:hossz]`

Például az `A1:4` az A jelű hang az 1-es oktávban, ami 4 hosszú ideig játszódik le.

Egy dallam lejátszásához ezeket a hangokat listába kell szerveznünk, akárcsak az animációknál tettük azt a képekkel. Példaként itt van a „János bácsi keljen fel” című gyerekdal a Pythonban:

```
import music

tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4",
        "C4:4", "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
music.play(tune)
```

¹² <https://microbit-micropython.readthedocs.io/en/latest/music.html> Elérés dátuma: 2018.08.04.

Elsőre kicsit olvashatatlanak tűnhet a dallam, valamint túl hosszú ideig kell gépelni. De ne aggódjunk, lehetőségünk van az egyszerűsítésre. A Python emlékszik az oktávra és a hossza amíg meg nem változtatjuk azokat. Emiatt a fenti dallam egyszerűbben is leírható, a következő módon:

```
import music

tune = ["C4:4", "D", "E", "C", "C", "D", "E", "C", "E", "F",
        "G:8", "E:4", "F", "G:8"]
music.play(tune)
```

Vegyük észre, hogy az oktáv és az időtartam értékek csak akkor kerülnek feltüntetésre, amikor megváltoznak az előző hanghoz képest. Ezáltal sokkal kevesebbet kell gépelnünk, és a kódunk is egyszerűbben olvasható.

Feladat a diákok számára

Keresd meg interneten az énekórán tanult egyik kedvenc dalod kottáját! Játssz le a dal első pár sorát a micro:biten!

4. Sziréna készítése

De nemcsak zenei hangokat hozhatunk létre, egyéb hanghatásokat is használhatunk. Például egy sziréna hangját így:

```
import music

while True:
    for freq in range(880, 1760, 16):
        music.pitch(freq, 6)
    for freq in range(1760, 880, -16):
        music.pitch(freq, 6)
```

Ebben az esetben a `music` modul `pitch()` metódusát kell használnunk. Paraméteréül egy frekvenciát – például a 440-es érték felel meg az A hangnak –, valamint egy hosszt vár, ami azt jelöli, hogy hány ms-ig kell lejátszani a hangot.

A ciklusban a `range()` függvényt használtunk, amivel számsorozatokot generálhatunk. Ezeket a számokat használjuk a hang frekvenciájaként. A `range()` függvényt három paramétere a kezdeti érték, a végső érték, és a lépésköz. Tehát az első `range()` függvény egy számsorozatot generál 880 és 1760 között, 16-osával. A második pedig 1760 és 880 közötti számokat generál, -16-osával.

Mivel azt szeretnénk, hogy folyamatosan hallatszódjon a sziréna, ezért egy végtelen ciklusban helyeztük el a kódot. Az újdonság ezen belül található, egy újfajta ciklus, a `for`. Ezt úgy fogalmazhatnánk meg, hogy a megadott halmazban minden egyes elemmel csináljunk valamit. Ebben az esetben konkrétan a számsorozat minden egyes elemét adjuk meg frekvenciaként a `pitch()` módszernek, és játsszuk le azt a hangot 6 ms-ig. Vegyük észre az igazítások jelentőségét, amivel már korábban is találkozhattunk.

Feladat a diákok számára

Kísérletezz a `for` ciklussal és a `pitch` módszerrel! Milyen dallamokat tudsz így lejátszani? Mutassátok meg egymásnak a kész dallamaitokat!

5. alkalom (mozgás, gesztusok)

Tematikai egység	Alkalmazott módszerek, munkaformák	Időtartam
A gyorsulásmérő használata	Frontális tanári magyarázat, közös programírás	15 perc
Gyorsulás és zene	Közös programírás	10 perc
A gesztus fogalma, gesztusok fajtái	Frontális tanári magyarázat	10 perc
A 8-as golyó	Közös programírás	20 perc
A többi gesztus kipróbálása	Egyéni programírás	15 perc
A 8-as golyó program módosítása	Egyéni programírás	10 perc

1. A gyorsulásmérő használata

A micro:biten található egy beépített gyorsulásmérő is, ami három tengely mentén képes mérni az elmozdulást. Ezek a következők:

- X – balra és jobbra döntés
- Y – előre és hátra döntés
- Z – felfelé és lefelé mozdítás

Minden tengelyhez tartozik egy metódus, ami visszaad egy pozitív, vagy egy negatív számot, ami a mért érték, ezred g-ben. Amikor a mért érték 0, az a tengely „egyenesben” van. Példaként a következő program kiírja a micro:bit kijelzőjére, hogy merrefelé dől az eszköz:

```
1. while True:
2.     reading = accelerometer.get_x()
3.     if reading > 20:
4.         display.show("J")
5.     elif reading < -20:
6.         display.show("B")
7.     else:
8.         display.show("-")
```

A második sorban leolvassuk az X tengelyen mért értéket, ahol ezt el is tároljuk egy változóba. (A példához hasonlóan létezik a `get_y()` és a `get_z()` metódus is.) Ezután

ettől az értéktől függően kiírjuk, egy-egy betűvel, hogy éppen merre dől az eszköz. Az elmélet alapján elég lenne azt ellenőriznünk, hogy a mért szám 0-nál kisebb, vagy nagyobb-e, azonban ez túl érzékeny volna, gyakorlatilag lehetetlen lenne középen tartani az eszközt, ezért egy kicsit nagyobb határt hagytunk az egyenesen tartásnak. Ha kipróbáljuk a programot így is látni fogjuk, hogy nem olyan egyszerű a kötőjelet a kijelzőn tartani. Mivel azt szeretnénk, hogy az eszköz folyamatosan mérjen és írja ki az éppen aktuális állapotnak megfelelő jelet, ezért az egészet egy végtelen cikluson belülre írjuk.

Kérdezzük meg a diákoktól, hogy honnan lehet ismerős a fenti program működése! Valószínűleg eszükbe fognak jutni az okostelefonok, amik szintén egy ilyen gyorsulásmérő segítségével állapítják meg, hogy éppen merre dől a telefon, hogyan kell tájolniuk a kijelzőt, vagy éppen a játékokban hogyan akarjuk irányítani az autót, szereplőt stb.

Feladat a diákok számára

Készíts programot, ami kiírja a kijelzőre, hogy az eszközt előre, vagy hátra döntjük éppen! Kísérletezz az értékekkel!

Amennyiben nem boldogulnak a tanulók a feladattal, segítségképpen felidézhetjük a téma elején tárgyalt tengelyeket. Kérjük meg a diákokat, hogy emlékezzenek vissza, hogy melyik tengely mentén lehetett mérni az eszköz előre és hátra döntését! Ezek alapján az előző programot néhány helyen módosítva megkapjuk a feladat jó megoldását.

2. Gyorsulás és zene

Eddig mindig csak egy-egy részét használtuk a micro:bitnek. Próbáljuk meg ezúttal az eszköz különféle funkcióit összekapcsolni! Írjunk egy programot, ami a döntés mértékétől függően játszik le hangokat!

```
import music

while True:
    music.pitch(accelerometer.get_y(), 10)
```

Az Y tengelyről beolvasott értéket adjuk át frekvenciaként a `pitch()` metódusnak, ami ezt a hangot 10 ms-ig játssza le, hogy követni tudja a változást a döntésben. A végtelen ciklus miatt ismételten folyamatosan futó programot kapunk. Teszteljük le, hogy

bizonyos mértékű döntéseknél milyen hangot hallunk (ha hallunk egyáltalán)! Ehhez természetesen össze kell kötnünk az eszközöket egy-egy fülhallgatóval.

3. A gesztus fogalma, gesztusok fajtái

Az előző részben megismert gyorsulásmérő „hozadéka” az, hogy a micro:bit különféle gesztusokat is tud érzékelni. Ez azt jelenti, hogy ha az eszközt egy bizonyos módon mozgatjuk (ez a gesztus), képes érzékelni azt, és a hozzá tartozó utasításokat végrehajtani. A Pythonban a következő gesztusok használatára van lehetőségünk: `up`, `down`, `left`, `right`, `face up`, `face down`, `freefall`, `3g`, `6g`, `8g`, `shake`. A gesztusok neveit ugyanúgy használhatjuk, mint a stringeket. A legtöbb gesztus neve magától értetődő, a `3g`, `6g`, és `8g` gesztusok a névnek megfelelő g erőhatás elérésekor aktiválódnak.

A jelenleg érvényben lévő gesztust az `accelerometer.current_gesture()` metódussal tudjuk lekérdezni. Az eredménye valamelyik fenti gesztus lesz. Nézzük meg egy egyszerű példaprogramon keresztül ennek használatát! Írjunk a fentiek alapján közösen egy programot, ami egy boldog arcot rajzol ki a kijelzőre, hogyha az eszköz felfelé néz, egyébként pedig egy ideges arcot:

```
1. while True:
2.     gesture = accelerometer.current_gesture()
3.     if gesture == "face up":
4.         display.show(Image.HAPPY)
5.     else:
6.         display.show(Image.ANGRY)
```

Végtelen ciklusba csomagoljuk a programot, hiszen folyamatos végrehajtásra van szükségünk. A második sorban lekérdezzük a jelenlegi gesztust a korábban már ismertetett metódussal, majd egy elágazás segítségével eldöntjük, hogy az eszköz felfelé néz-e. Ha igen, a beépített boldog arcot használjuk, ha nem, az ideges arcot.

4. A 8-as golyó

Különböző amerikai filmekben, rajzfilmekben találkozhatunk a varázserejű fekete 8-as biliárdgolyóval, ami minden eldöntendő kérdésünkre megadja a választ. Alakítsuk most a micro:bitet varázserejű 8-assá! Tegyük fel a micro:bitnek egy kérdést, rázzuk meg, majd olvassuk el a kijelzőn, hogy mi a válasz rá! Írjunk egy programot, ami megvalósítja ezt a működést:

```

1. import random
2.
3. answers = [
4.     "Biztosan így lesz",
5.     "Kétségkívül",
6.     "Igen, mindenkeppen",
7.     "Számithatsz rá, hogy így lesz",
8.     "Valószínűleg",
9.     "Igen",
10.    "A csillagok szerint igen",
11.    "Később kérdezd meg újra",
12.    "Most nem mondhatom el",
13.    "Jelenleg megjósolhatatlan",
14.    "Koncentrálj, és kérdezd meg újra",
15.    "Ne számíts rá"
16.    "A válaszem nem",
17.    "A forrásaim alapján nem",
18.    "Nem tunik valószínűnek",
19.    "Nagyon kétséges",
20. ]
21.
22. while True:
23.     display.show("8")
24.     if accelerometer.was_gesture("shake"):
25.         display.clear()
26.         sleep(1000)
27.         display.scroll(random.choice(answers))

```

A program nagyrészt az előre megadott válaszok listája teszi ki. Természetesen ezeket megalkothatjuk a gyerekekkel közösen is, sőt egymás programjába is beírhatnak általuk kitalált válaszokat, akár úgy, hogy körbe mennek a gépeken, és mindenki ír pár ötletet a másik programjába. Példaként áll itt néhány, idő hiányában ezeket is nyugodtan használhatjuk. A működés a 22. sorban kezdődik. A végtelen cikluson belül, amennyiben megráztuk az eszközt letöröljük a kijelzőt, majd kis „gondolkodási idő” után kiírunk egy véletlenszerűen kiválasztott választ a listából. Ez a 8-as golyó varázsereje! A diákokkal közösen döntsük el az élet nagy kérdéseit, immáron a micro:bitek segítségével!

Feladat a diákok számára

Készíts programot, ami a különböző gesztusokra mind-mind máshogy reagál! Próbáld ki, hogy melyik gesztus mikor lép életbe!

Módosítsd a 8-as golyó programot! Hogyan lehetne „csalni” a golyó varázserejével, hogy kedvünk szerint pozitív, vagy negatív választ adjon? (Tipp: használd a gombokat!)

6. alkalom (iránytű, rádió)

Tematikai egység	Alkalmazott módszerek, munkaformák	Időtartam
Az iránytű	Frontális tanári magyarázat, közös programírás	20 perc
Rádió	Frontális tanári, magyarázat	5 perc
Szentjánosbogarak	Közös programírás	25 perc
Hideg-meleg játék	Közös megbeszélés, pármunka	40 perc

1. Az iránytű

Az eszközben található egy beépített iránytű, aminek segítségével meghatározhatjuk, hogy merre van észak. A diákokkal közösen írjuk meg az ezt megvalósító programot!

```
compass.calibrate()

while True:
    needle = ((15 - compass.heading()) // 30) % 12
    display.show(Image.ALL_CLOCKS[needle])
```

Mindig, amikor az iránytűt használjuk, kalibrálnunk kell az eszközt, hogy megfelelő értékeket kapjunk az érzékelőből. Erre szolgál a `compass` osztály `calibrate()` metódusa. Amikor lefut ez a metódus, a `micro:bit` megkér minket, hogy „rajzoljuk” tele a kijelzőt, az eszköz különböző irányokba való döntésének segítségével. Amennyiben ez sikerült, egy mosolygó fej a jutalmunk, és használatba vehetjük az iránytűt. Az állandó működést újból egy végtelen ciklussal biztosítjuk. Ezen belül egy kis matematika segítségével kiszámoljuk a `compass.heading()` metódus értékéből, hogy merre kell mutatnia a mutatónak. A `//` az egészosztás, a `%` a maradékos osztás jele a Pythonban. A `heading()` érték 0 és 360 fok között adja meg az aktuális irányt, ahol a 0 az észak. A ciklusmag második sorában a kiszámolt érték segítségével kirajzolunk egy órát, aminek a mutatója nagyjából mindig észak felé fog mutatni. A diákok összehasonlíthatják az egyes eszközök eredményeit, így ellenőrizve, hogy jól működnek-e, illetve, hogy jó programot írtak-e.

2. Rádió

Mindeddig csak egyetlen eszközt használtunk a programjainkhoz. Azonban a micro:bitek rendelkeznek beépített rádió-adóvevővel is, aminek a segítségével kommunikálni tudnak egymással. Ez rengeteg új lehetőséget nyit meg, legyen az akár két eszköz közötti információcsere, többszemélyes, több eszközön futtatható játékok stb.

A rádiózással kapcsolatosan felmerülő probléma, hogy nem lehet közvetlenül egy személynek küldeni a jeleket, azokat mindenki tudja olvasni, aki megfelelő vevővel rendelkezik. Ezért fontos, hogy valamilyen szinten el tudjuk dönteni, hogy kik kapják meg az adásunkat. A micro:bit erre egy meglehetősen egyszerű megoldást alkalmaz: a rádiót különböző csatornákra állíthatjuk be (0 és 83 közt). Ez ugyanúgy működik, mint a walkie-talkie-k esetében. Azok, akik ugyanarra a csatornára állnak, hallják egymást, míg mindenki más nem hallja az adott csatornán folyó beszélgetést.

3. Szentjánosbogarak

Ezúttal egy természetből vett példával demonstráljuk a rádió használatát. A szentjánosbogarak ún. biolumineszcencia segítségével jeleznek egymásnak. Rajzásuknál ezekből a jelzésekből egy nagyon látványos fényjáték áll össze. Valami ehhez hasonlót fogunk létrehozni a micro:bitek segítségével.

Első lépésként be kell importálnunk a `radio` modult a már ismert `import radio` utasítással. A rádió alapesetben energiatakarékossági okok miatt ki van kapcsolva, bekapcsolását a `radio.on()` metódussal érhetjük el. Ilyenkor a rádió alapbeállításokkal indul el, mindenféle konfiguráció nélkül használatba vehetjük, az eszközeink képesek lesznek kommunikálni egymással. Természetesen, amikor több eszközt egymástól függetlenül akarunk ilyen célokra használni, szükség lehet bizonyos beállításokra. A korábban már említett csatorna mellett sok más beállítási lehetőségünk is van, ezeket szükség esetén érdemes a dokumentáció vonatkozó oldalán¹³ tanulmányozni.

Amennyiben megfelelőek számunkra az alapbeállítások, már el is kezdhetjük az üzenetek küldését:

```
radio.send("uzenet")
```

¹³ <https://microbit-micropython.readthedocs.io/en/latest/radio.html> Elérés dátuma: 2019.01.24.

Az üzenetek fogadása is hasonlóan könnyen megy:

```
new_message = radio.receive()
```

Ahogy az üzeneteket megkapjuk, egy sor adatszerkezetbe kerülnek bele. A `receive()` metódus egy sorból műveletnek felel meg, ezáltal helyet csinál az új üzeneteknek. Ha a sor megtelik, az új bejövő üzeneteket figyelmen kívül hagyjuk.

Ezzel a két paranccsal már el is tudjuk készíteni a szentjánosbogár-animációnkat:

```
1. import radio
2. import random
3. from microbit import *
4.
5. flash = [Image().invert()*(i/9) for i in range(9, -1, -1)]
6.
7. radio.on()
8.
9. while True:
10.     if button_a.was_pressed():
11.         radio.send("flash")
12.         incoming = radio.receive()
13.         if incoming == "flash":
14.             sleep(random.randint(50, 350))
15.             display.show(flash, delay=100, wait=False)
16.             if random.randint(0, 9) == 0:
17.                 sleep(500)
18.                 radio.send("flash")
```

Az első három sorban elvégezzük a szükséges importálásokat. Az 5. sorban megadjuk a lejátszandó animációt. Az értelmezést nyugodtan bizzuk rá a tanulókra! Segítségként tanulmányozzák át a dokumentáció `Image`¹⁴ aloldalát! Végül a 7. sorban bekapcsoljuk a rádiót az eszközön. Ezzel végeztünk az előkészületekkel, jöhet a program lényegi fázisa.

Természetesen megint végtelen ciklust használunk, ezzel biztosítva a folyamatos működést. A 10-11. sorban olvasható, hogy az „A” gomb megnyomására tudunk jelet küldeni. A következő sorban kiolvassuk a legrégebben megkapott jelet egy változóba, majd, ha ez megegyezik a „flash” szöveggel, tudjuk, hogy indulhat a villogás. Véletlen időtartamú várakozás után kijelezzük a korábban megadott animációt. Ezek után a 16. sorban eldöntjük, hogy tovább küldjük-e a jelet. Ez véletlenszerűen történik meg, jelen

¹⁴ <https://microbit-micropython.readthedocs.io/en/latest/image.html> Elérés dátuma: 2019.01.24.

esetben 10% eséllyel. Így lehetséges több eszközt közt fenntartani az effektust. A rendelkezésre álló eszközök számától függően lehet a továbbküldés esélyét növelni, vagy csökkenteni. Amennyiben továbbküldésre kerül a sor, hagyjunk egy kis időt az animáció végigfutására, majd elküldjük a „flash” üzenetet.

Végeredményben legalább 9-10 micro:bittel egy nagyon látványos effektust hozhatunk létre, főleg egy sötét teremben. A véletlenszerű villogás, elhalványulás a szentjánosbogarak rajzására hasonlít.

4. Hideg-meleg játék

Bár a rádió funkció használatával egyszerre több eszközre is szükségünk volt a programjaink működéséhez, minden egyes eszközön ugyanaz a kód futott. Ezúttal egy olyan programot fogunk írni, aminél a különböző eszközökre különböző kód is kerül. Kiskorában biztosan mindenki játszott a szüleivel, esetleg testvéreivel, barátaival hideg-meleg játékot. Ennek a lényege, hogy elrejtünk egy tárgyat, majd valaki annak keresésére indul. A tárgyat elrejtő személy a hideg-langyos-meleg szavakkal jelzi a keresőnek, hogy milyen messze, vagy közel jár a tárgy megtalálásához. A micro:bitek segítségével fogjuk újra alkotni ezt a gyerekkori játékot. Jelen esetben az elrejtett tárgy egy micro:bit lesz, ami rádiójeleket fog sugározni magából. A keresőnél lesz egy másik micro:bit, ami ezeket a jeleket fogja, és a jelerősség alapján különböző ikonokat jelenít meg a kijelzőn, attól függően, hogy mennyire vagyunk közel a másik eszközhöz. A teljes megoldás azonnali ismertetése helyett ezúttal lépésenként fogjuk felépíteni az ehhez szükséges programokat, teendőket.

Az első lépés tehát a jeladó kódjának megírása. Ezzel nagyon egyszerű dolgunk lesz, hiszen az eszköz semmi mást nem fog csinálni, mint egy folyamatos rádiójelet sugározni. A szükséges beállítások elvégzése után tehát csak pár sort kell írunk.

```
1. import radio
2.
3. radio.config(power=4)
4. radio.on()
5.
6. while True:
7.     radio.send("0")
```

A rádiónál már megszokott importálás és bekapcsolás között feltűnik egy új művelet, a `radio.config()`. Ennek a metódusnak a paramétereiben állíthatjuk be az eszköz

rádiójának különféle tulajdonságait, például a már korábban tárgyalt csatornát. Ezúttal a jelerősséget állítjuk be, hogy egy kisebb teremben és megfelelő különbségeket lássunk az adó eszköztől távol és közel mért értékek között. Ezt a `power` paraméter értékének változtatásával érhetjük el. Ez egy 0 és 7 között lévő érték lehet, az alapértelmezett a 6-os. Ezt tehát valamivel gyengébbre vettük. Amelyik paraméterekre nem térünk ki, az az alapértelmezett értékekkel fog szerepelni, ezért is szükséges a paraméter nevét is specifikálni, és egy egyenlőségjel után megadni a kívánt értéket. Eddig egyáltalán nem hívtuk meg a `config()` metódust, ami azt jelenti, hogy minden tulajdonságból az alapértelmezett értékek lesznek használva. Azt, hogy melyek ezek, a dokumentációban¹⁵ tudjuk ellenőrizni. Ezután jön a már megszokott végtelen ciklus, hiszen a jel sugárzására folyamatosan szükségünk van. Ezen belül már csak egy üzenetet kell küldenünk, aminek a szövege tetszőleges lehet. A lényeg, hogy ugyanezt az üzenetet kell majd figyelni a vevőkkel. Ebben a pár sorban végeztünk is az adó kódjával, tökéletesen fog működni a projektünk. Azonban célszerű még egy kis szépítést végrehajtanunk! Jelenleg, ha elindítjuk a micro:bitet, csak egy sötét kijelzőt látunk. Érdekes valamiféle animációval kísérni a működést a kijelzőn, hogy megbizonyosodjunk róla, hogy a ciklus valóban fut, és az eszköz folyamatosan küldi a jeleket. Egészítsük hát ki a programot ezzel!

```
1. import radio
2.
3. radio.config(power=4)
4. radio.on()
5.
6. while True:
7.     radio.send("0")
8.     display.show(Image.HEART_SMALL)
9.     sleep(500)
10.    display.show(Image.HEART)
11.    sleep(500)
```

A rádiójel küldése után egy szívdobbanás animációt fogunk szimulálni, ezzel jelezve, hogy épp működésben van az eszköz. Először megjelenítjük a beépített ikonkészlet közül a kis szívet, majd fél másodperc várakoztatás után átváltunk a nagyra. Újabb fél másodperc után újraindul a ciklus, egy új jel küldésével, majd újból a kis szív megjelenítésével. Működés közben a két kép váltakozása olyan animációt hoz létre, mintha dobogna az eszköz „szíve”. Ebben a megoldásban körülbelül egy

¹⁵ <https://microbit-micropython.readthedocs.io/en/latest/radio.html> Elérés dátuma: 2019. 02. 04.

másodpercenként küldjük a jeleket, majd a vevő használatánál ezt figyelembe kell vennünk.

Ezzel az adónk készenáll, töltsük rá a fenti programot, majd tegyük félre. Következhet a vevő programjának a megírása. Át kell gondolnunk, hogy hogyan állapíthatnánk meg egy rádiójelről, hogy mennyire van közel. Szerencsére a MicroPythonban egy üzenet fogadásánál elérhetjük annak a jelnek az erősségét is. A blokkos programozásnál ez egy nagyon egyszerűen elérhető érték volt. Sajnos a Pythonos környezetben nincs rá egy kész függvény, kicsit kutakodni kell a dokumentációban, valamint a Python nyelvet is ismerni kell. Azt már tudjuk, hogy az üzenet fogadása a `radio.receive()` metódussal történik, ebből azonban nem tudjuk megállapítani a jelerősséget. Ehhez egy másik fajta üzenetfogadási módot, a `radio.receive_full()` metódust kell használnunk. Ez egy ún. tuple-t ad vissza, aminek magyar nyelvű megfelelője nem igazán van, talán egy rekordra, vagy listára hasonlít leginkább. Egy tuple-ben több érték található, vesszővel elválasztva. Különbség az előbb felsoroltakhoz képest, hogy értékeit nem tudjuk megváltoztatni. Amennyiben más értékekre volna szükségünk egy tuple-ben, újat kell létrehoznunk. Amiben viszont hasonlít hozzájuk, hogy egyes elemeire indexeléssel hivatkozhatunk, az indexet pedig szögletes zárójelek között adhatjuk meg. További információk a tuple-ről a Python dokumentációjában¹⁶ találhatóak, itt csak a feladathoz szükséges mértékben jelennek meg a részletek.

A `radio.receive_full()` metódus egy három elemű tuple-t ad vissza. Ennek az elemei sorban:

- a következő bejövő üzenet az üzeneteket tartalmazó sorban, bájként
- a jelerősség, ami egy 0 (legerősebb) és -255 (leggyengébb) között lévő egész szám, a jel erőssége dBm-ben (decibel milliwattban)
- egy időbélyeg

Ezek alapján elkezdhetjük írni a vevő kódját. Először teszteljük le, hogy fogjuk-e a jelet, és meg tudjuk-e állapítani annak erősségét. Folyamatosan írjuk ki ezt a kijelzőre!

¹⁶ <https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences> Elérés dátuma: 2019. 02. 02.

```
1. import radio
2.
3. radio.on()
4.
5. while True:
6.     message = radio.receive_full()
7.     if message:
8.         display.scroll(message[1])
```

A program első sorai a már megszokottak. A lényegi részben használjuk a fentebb megismert üzenetfogadási módot. A 7. sorban található ellenőrzésre mindenképpen szükségünk van, enélkül hibát dobna a program, mert nem biztos abban, hogy egy létező értékre hivatkozunk a 8. sorban. Ebben láthatjuk, hogy megjelenítjük a kijelzőn a kapott tuple 2. elemét, ami az 1-es indexű, hiszen a Pythonban is 0-ás kezdőindexet alkalmazunk, akár a legtöbb modern programozási nyelvben.

Ezzel kész az ellenőrző programunk, teszteljük le a működést! Töltsük rá a vevő eszközre a programot, majd kapcsoljuk be az adót, akár a számítógéphez csatlakoztatva, akár elemről működtetve. Az adón megindul a szívdobogás, ezzel párhuzamosan a vevőnek ki kell írnia a kapott jelek erősségét. Végezzünk pár próbamérést, különböző távolságokból. Döntsük el, hogy mik legyenek a küszöbértékek a hideg, a meleg, illetve a forró jelzésekhez, majd jegyezzük fel ezeket! Ezek nagyban függenek a mérés helyétől és az esetleges zavaró körülményektől, így az itt szereplő értékek csupán példaként szolgálnak, minden helyszínen saját mérést kell végezni. Esetemben -75 dBm alatt lesz a hideg, -65 és -75 dBm között a meleg, és -65 dBm felett pedig a forró jelzése.

Ezután módosítsuk a vevők programját! A kalibrációval végeztünk, így a jelerősségek kiírására már nincs szükségünk. Ehelyett a meghatározott küszöbértékek mentén kell egy megfelelő ikont kijelezni a vevő kijelzőjén. Emlékezzünk vissza az első alkalommal kipróbált Image modul képeire! A tanulókkal együtt keressünk három megfelelő ikont, ami viszonylag egyértelműen jelzi, hogy a hideg, a meleg, vagy a forró tartományában járunk! A diákok gondolják át, hogy milyen vezérlési szerkezetre lesz itt szükség! Valószínűleg rá fognak jönni, hogy az elágazásra, de vajon hány feltétel lesz benne? Első válaszként várhatóan hármat fognak mondani, hiszen három intervallumunk van, az ezekbe való beletartozást kell vizsgálnunk. Azonban vezessük rá őket, hogy kettő vizsgálat is elég, hiszen hogyha az első két intervallumba nem tartozik egy érték, akkor biztosak lehetünk benne, hogy a harmadikban benne van! A programot ezek alapján már önálló munkában is el tudják készíteni a diákok. A javasolt megoldás:

```

1. import radio
2.
3. hot = Image("99999:99999:99999:99999:99999")
4.
5. radio.on()
6.
7. while True:
8.     message = radio.receive_full()
9.     if message:
10.         if message[1] < -75:
11.             display.show(Image.DIAMOND_SMALL)
12.         elif message[1] < -65:
13.             display.show(Image.DIAMOND)
14.         else:
15.             display.show(hot)

```

Az egyes értékekhez tartozó képek kiválasztásánál én a kis gyémánt, a gyémánt, és az összes LED felvillantása mellett döntöttem. Ez utóbbit a 3. sorban deklaráltam is, a másik kettő beépített kép, így ezek előzetes felvételére nincs szükség. A rádió bekapcsolása után kezdődhet is a végtelen ciklus, hiszen az éppen aktuális „hőmérséklet” kiírása folyamatosan szükséges. A jelerősséget a kalibrálásnál megismert módon olvassuk ki az üzenetből, majd jöhet az elágazás. Először ellenőrizzük, hogy a hideg tartományban van-e a mért érték. Amennyiben igen, a kijelzőn megjelenítjük a hozzá tartozó képet. Ezután a meleg tartomány következik. Figyeljük meg, hogy a feltételben nem az szerepel, hogy az érték nagyobb mint -75 és kisebb mint -65, elég csak az utóbbit ellenőrizni, hiszen az első feltétel nem teljesüléséből már következik, hogy a jelerősség nagyobb mint -75. Ezért fontos, hogy mindenképpen az `elif` kulcsszót használjuk, különben sosem látnánk a kijelzőn a hideget jelképező kis gyémántot, hiszen ami -75-nél kisebb, az -65-nél is. Legvégül, ha ezek egyike sem teljesül, biztosak lehetünk benne, hogy a jelerősség legalább -65 dBm, így megjeleníthetjük a forró tartományhoz tartozó képet.

A vevő programja ezzel el is készült, eljött a kipróbálás ideje! A diákok páronként az egyikük eszközére az adó, a másikuk eszközére pedig a vevő kódját töltsék fel, ügyelve a csatornák helyes beállítására. Ne legyen egynél több pár ugyanazon a csatornán! Ezután csatlakoztassák az elemeket vagy egy-egy powerbanket a micro:bitekhez, egy rövid tesztel bizonyosodjunk meg róla, hogy működnek a programok, majd az egyik diák dugja el a teremben, a folyosón, vagy az iskola egy meghatározott részén az adót. A másik diák induljon el a vevővel, és próbálja megkeresni azt! Ezután cserélődjenek fel a szerepek, dugja el a másik diák az adót, és a párja keresse meg!

7. alkalom (távíró készítése)

Tematikai egység	Alkalmazott módszerek, munkaformák	Időtartam
Bevezetés, a távíró bemutatása	Frontális tanári magyarázat	5 perc
A program funkcióinak, működésének megbeszélése	Közös megbeszélés	10 perc
A program megírása	Pármunka	45 perc
Tesztelés, üzenetek küldése	Pármunka, csoportmunka	10 perc
Megoldások megbeszélése, bemutatása	Közös megbeszélés, kiselőadás	20 perc

1. Bevezetés, a távíró bemutatása

A rádió jelenléte az eszközben szinte adja a következő program ötletét. Készítsünk távírot, ami morzekódot képes továbbítani! Bevezetőként beszéljünk a diákoknak a morzekód alapjairól, az ábécéről, a rövid és hosszú jelekről, kódolásról, dekódolásról. Kis történeti áttekintést is adhatunk azokról az időkről, amikor ez volt a kommunikáció általános formája, mikor nagy távolságokba kellett üzeneteket gyorsan eljuttatni. Szinte biztos, hogy már valamennyit hallottak róla a diákok, így ez a rész teljesen frontális előadás helyett akár egy közös megbeszélés is lehet.

2. A program funkcióinak, működésének megbeszélése

A diákok feladata egy olyan program megírása lesz, aminek alapvető funkciója az, hogy kettő (esetleg több) micro:bit között tudjon üzeneteket küldeni morzekód segítségével. Beszéljük át közösen velük, hogy hogyan működjön a program, milyen funkciói legyenek még ezen kívül! Az itt bemutatott mintaprogramban a jelek küldésén kívül jelezni tudjuk még azt, hogy a következő szó jön, valamint azt is, hogyha vége van az üzenetünknek. Érdekes lehet egységesíteni a küldött üzenetek tartalmát, hogy a párok később egymás közt is tudjanak üzenetet küldeni.

3. A program megírása

Ezután a diákok páronként írják meg az előzőleg felvázolt programot. A tanár feladata ezúttal a monitorozás, esetleges problémák esetén segítségnyújtás. Az alábbiakban egy mintamegoldás szerepel, amivel az azonos csatornán lévő micro:bit-ek tudnak egymásnak

jeleket küldeni, valamint a kapott jeleket megjeleníteni a saját kijelzőjükön. Az „A” gomb küld egy rövid jelet, a „B” pedig egy hosszút. Az eszköz megrázásával jelezzük, hogy a következő szó jön, ilyenkor egy nyilat küldünk. A kavarodások elkerülése végett érdemes egy végjelet is küldeni az üzenetünk végén. Ezt a micro:bit lefelé döntésével jelezzük, ilyenkor küldünk egy X-et.

```
1. import radio
2.
3. dot = Image("00000:09990:09990:09990:00000")
4. dash = Image("00000:00000:09990:00000:00000")
5. nextWord = Image("00900:00090:99999:00090:00900")
6. over = Image("90009:09090:00900:09090:90009")
7.
8. radio.on()
9.
10. while True:
11.     if (button_a.was_pressed()):
12.         display.show(dot)
13.         radio.send("dot")
14.         sleep(100)
15.         display.clear()
16.     if (button_b.was_pressed()):
17.         display.show(dash)
18.         radio.send("dash")
19.         sleep(200)
20.         display.clear()
21.     if (accelerometer.was_gesture("shake")):
22.         display.show(nextWord)
23.         radio.send("nextWord")
24.         sleep(1000)
25.         display.clear()
26.     if (accelerometer.current_gesture() == "up"):
27.         display.show(over)
28.         radio.send("over")
29.         sleep(500)
30.         display.clear()
31.
32.     incoming = radio.receive()
33.
34.     if (incoming == "dot"):
35.         display.show(dot)
36.         sleep(100)
37.         display.clear()
38.     elif (incoming == "dash"):
39.         display.show(dash)
40.         sleep(200)
41.         display.clear()
```

```

42.     elif (incoming == "nextWord"):
43.         display.show(nextWord)
44.         sleep(1000)
45.         display.clear()
46.     elif (incoming == "over"):
47.         display.show(over)
48.         sleep(500)
49.         display.clear()

```

A szükséges importálásokról ne feledkezzünk meg ezúttal sem. A 3-6. sorban deklaráljuk a megjelenítendő képeket az első órán ismertetett módon. Ezután bekapcsoljuk a rádiót, majd következik a szokásos végtelen ciklus. Ezen belül megvizsgáljuk, hogy megnyomtuk-e az „A” gombot. Amennyiben igen, kijelesszük a rövidet jelképező pontot, elküldjük a hozzá tartozó üzenetet a rádión, majd kis várakoztatás után letöröljük a kijelzőt. Ugyanígy járunk el a „B” gomb esetén, ezúttal azonban a hosszú üzenetet küldve. Rázással tudjuk jelezni, hogy új szó következik, az üzenetünk végén pedig le kell döntenünk az eszközt. Emlékezzünk vissza, hogy a lefelé döntés megfelelője a gesztusok között az up, vagyis felfelé, hiszen az eszköz tetején található lógó helyzetét kell megadni.

Az üzeneteket tehát már el tudjuk küldeni, azonban minden eszköz egyszerre adó és vevő is, szóval fel kell készülnünk a jelek fogadására is. A 32. sorban kiolvassuk a kapott üzenetet, majd az értékének megfelelően kezeljük azt, a küldésnél látott módon. Ezután nincs más dolgunk, mint megnyitni a morzekódokat tartalmazó táblázatot, és kezdődhet az üzenetítés!

4. Tesztelés, üzenetek küldése

A diákok a párjaikkal teszteljék a programot, próbáljanak megbeszélni a másikkal egy találkozót! Ezután álljanak össze egy másik párral, 4 fős csoportot alkotva. Válasszanak egy közös csatornát, majd hívják meg a másik párt is a találkozóra, a micro:bitek segítségével!

5. Megoldások megbeszélése, bemutatása

Mivel végig önálló feladatmegoldás volt az órán, érdemes az óra végén egy közös megbeszélést tartani. Ehhez válasszunk ki egy párt, akiknél a munka során úgy láttuk, bemutatásra érdemes program készült. Ezt vetítsük ki, és a diákokkal közösen beszéljük meg, hogyan készült el a program, miért pont úgy csinálták, ahogy. Ügyesebb tanulók

akár egy kiselőadás formájában is bemutatthatják a kész programjukat, a többieket megtanítva a készítés folyamatára. Ha esetleg voltak olyan párok, akik másfajta megoldást készítettek el, őket is hallgassuk meg, hasonlítsuk össze a megoldásokat!

8. alkalom (fájlkezelés)

Tematikai egység	Alkalmazott módszerek, munkaformák	Időtartam
A fájlkezelés alapjai	Közös megbeszélés	5 perc
A micro:bit fájlrendszere	Frontális tanári magyarázat	5 perc
Fájlok tartalmának olvasása	Közös programírás	10 perc
Fájlba írás	Közös programírás, önálló munka	15 perc
Fájlmanipulációs műveletek	Közös munka	10 perc
A microfs eszköz	Közös munka, önálló munka, csoportban	20 perc
Önálló feladatmegoldás	önálló munka, pármunka	25 perc

1. A fájlkezelés alapjai

Adatok tárolására rengeteg esetben szükség lehet. Beszéljük meg a diákokkal, hogy hogyan történhet ez! Valószínűleg eljutunk a számítógépes adattárolásig is, ezen belül is ahhoz, hogy hol tároljuk az adatokat, például pendrive, merevlemez. Ám a mi szempontunkból most érdekesebb az, hogy „miben”, milyen formában tároljuk az adatokat. Erre az adatbázis lehet egy logikus válasz a gyerekek részéről, de vezessük rá őket az egyszerűbb megoldásra, a fájlokra. Ezután előkerülhetnek olyan fogalmak, mint a fájlrendszer és a különböző fájlműveletek. Ezekkel fogunk most megismerkedni a MicroPython nyelv segítségével.

2. A micro:bit fájlrendszere

Egy programozási nyelv megismerésénél szinte mindig előjön a fájlkezelés témaköre, nincs ez másképp a MicroPythonnál sem. Az micro:bit beépített tárhelyének köszönhetően megvan a lehetőségünk a fájlok létrehozására és eltárolására. A tárhely mérete ugyan nem nagy, mindössze 30 kilobájt, néhány szöveges fájl tárolására azonban így is megfelelő. Fontos megjegyezni, hogy ez a tárhely nem egyenlő azzal a tárhellyel amit a számítógépen látunk, miután csatlakoztattuk hozzá az eszközt. Ez a mód csak a HEX fájlok másolására szolgál, nem is fogjuk itt látni a létrehozott fájlokat. Tudni érdemes még, hogy a fájlrendszer nem képes mappák kezelésére, minden fájl ugyanazon a helyen van tárolva, valamint, hogy a fájlok kikapcsolás után is megmaradnak. Maga a

Python nyelv rengeteg könnyen és jól használható módot kínál a fájlok kezelésére, ezekből jópár dolog implementálva lett a MicroPythonban is.

3. Fájlok tartalmának olvasása

A fájlok írása és olvasása az `open()` függvény segítségével valósítható meg. Egy fájl megnyitása után annak bezárásáig tudunk vele dolgozni. Minden esetben szükséges a fájl bezárása, hogy a program tudja, már nem akarunk vele dolgozni. A legjobb módszer ennek biztosítására, ha az ún. `with` kulcsszót használjuk, a következő módon:

```
with open("story.txt") as my_file:
    content = my_file.read()
print(content)
```

A `with` kulcsszó után az `open()` függvénnyel megnyitjuk a `story.txt` nevű fájlt, majd a `my_file` nevű objektumhoz rendeljük azt. A következő, beljebb igazított sorban a `content` változóba olvassuk be a fájl tartalmát, a `read()` metódus segítségével.

Figyeljük meg, hogy a következő sor már nincs beljebb kezdve. Ebből láthatjuk, hogy a `with` kulcsszóhoz melyik sorok tartoznak. Jelen esetben csak az a sor, ahol beolvassuk a fájl tartalmát. Amint a `with` blokk végére értünk, a Python automatikusan bezárja a fájlt, hiszen már nincs rá szükségünk. A `with` kulcsszóval jeleztük, hogy ezt a fájlt csak ebben a blokkban szeretnénk használni. Ez a fajta szerkezet ismerős lehet például a Java nyelvből, ahol a `try-with-resources` tölt be hasonló szerepet.

4. Fájlba írás

A fájlokat azonban természetesen nem csak olvasásra, hanem írásra is megnyithatjuk. Ehhez az `open()` függvényben kell jeleznünk a szándékunkat, a következő módon:

```
with open("hello.txt", 'w') as my_file:
    my_file.write("Hello, World!")
```

Figyeljük meg a `'w'` paramétert a függvény hívásában. Ez jelzi, hogy a fájlt `write` módban szeretnénk megnyitni. Akár az `'r'` paramétert is használhattuk volna az előbb, de mivel az olvasási mód az alapértelmezett, ezért ezt el szoktuk hagyni.

Az adatok írása a `write()` metódussal történik, aminek a paraméterébe kerül az a string, amit a fájlba szeretnénk írni. A fenti példában a „Hello, World!” szöveg íródik egy `hello.txt` nevű fájlba. Vigyáznunk kell azonban a fenti metódus működésével!

Amennyiben már létezik az a fájl, amibe írunk, a tartalma felülíródik. Amennyiben hozzáfűzni szeretnénk egy fájlhoz, be kell olvasnunk a tartalmát egy változóba, bezárni a fájlt, hozzáfűzni ehhez a változóhoz a kiírandó szöveget, megnyitni a fájlt írásra, végül pedig kiírni a változó tartalmát a fájlba. Ez eltérés a szokásos Python és a microPython között, mivel a Pythonban található ún. `append` mód is, amivel hozzáfűzhetünk egy fájlhoz. A lehető legegyszerűbb implementáció miatt azonban ez a funkció kimaradt a MicroPythonból. Legyen a diákok feladata az előző fájlhoz még egy mondat hozzáfűzése!

5. Fájlmanipulációs műveletek

Utaljunk vissza az óra eleji bevezetőre a fájlrendszerekről! Most az ezek által felkínált műveletekre fogunk visszatérni.

A fájlok olvasása és írása mellett lehetőségünk van még különböző egyéb manipulációs műveletek végrehajtására is, úgy, mint a fájlok listázása vagy törlése. A számítógépen ezt az operációs rendszer végzi, azonban hasonló funkcionalitás került a Pythonba is az `os` nevű modul által. Ezt a modult használva tudunk egy asztali operációs rendszer működéséhez hasonló műveleteket elvégezni. Alapvetően három műveletet tudunk végezni a fájlrendszerrel: kilistázni a fájlokat, törölni egy fájlt, valamint lekérdezni egy fájl méretét. Ezekhez természetesen mind szükség van az `os` modul importálására.

A fájlok listázása a `listdir()` függvénnyel történik. Ez visszaad egy stringekből álló listát, ami a fájlneveket tartalmazza.

```
import os
my_files = os.listdir()
```

Egy fájl törléséhez a `remove()` függvényt használhatjuk. Egy paramétere van, annak a fájlnek a neve, amit törölni szeretnénk.

```
import os
os.remove("filename.txt")
```

Végül, egy fájl méretének lekérdezésére szolgál a `size()` függvény. Ez egy egész számot ad vissza, a paraméterében megadott fájl által foglalt bájtok számát.

```
import os
file_size = os.size("filename.txt")
```

6. A microfs eszköz

Programjainkból tehát már tudunk létrehozni és olvasni is fájlokat, de mi a helyzet akkor, ha szeretnénk az eszközre másolni, vagy az eszköztől lementeni azokat? Erre szolgál a `microfs` eszköz. Ezt akkor tudjuk használni, ha a Python telepítve van a számítógépünkre. A telepítés és a használat módját részletesen megtekinthetjük a dokumentációban¹⁷, itt csak a legfontosabb funkciókat mutatom be.

Az eszköz telepítéséhez nyissunk parancssort, majd adjuk ki az alábbi utasítást:

```
pip install microfs
```

Ne feledjük, hogy ez csak akkor fog működni, ha a Python telepítve van a számítógépre!

Ezután használatba is vehetjük, a parancsokat minden esetben az `ufs` előtaggal kezdve.

A fájlok kilistázásához a Unixból már ismert parancsot használhatjuk:

```
ufs ls
```

Ha a `micro:bit`-ről szeretnénk lementeni egy fájlt, akkor egy FTP-ből ismerős parancshoz kell nyúlnunk:

```
ufs get story.txt
```

Ez a parancs lementi a `story.txt` fájlt a jelenlegi helyre.

Fájlt az `rm` paranccsal törölhetünk:

```
ufs rm story.txt
```

Végül, ha a számítógépről szeretnénk fájlt másolni a `micro:bit`-re, az alábbi parancs szolgál segítségül:

```
ufs put story2.txt
```

Ezzel a pár egyszerű paranccsal a `micro:bit` egyszerű fájlrendszerét megfelelően tudjuk kezelni.

¹⁷ <https://microfs.readthedocs.io/en/latest/> Elérés dátuma: 2019. 01. 14.

Feladat a diákok számára

Hozz létre egy fájlt a saját neveddel, majd adjátok tovább a micro:bitet a valakinek a teremben! A megkapott micro:bitre másolj rá egy fájlt, szintén a saját neveddel! Adjátok még tovább néhányszor az eszközöket. Az utolsó ember írjon egy programot, ami kiírja az eszközön látható fájlok nevét! Próbáljátok meg rekonstruálni az eszköz által bejárt útvonalat!

7. Önálló feladatmegoldás

Most pedig következzen néhány, a csoport tudásáról függően önállóan, esetleg párban elkészítendő feladat.

Feladat a diákok számára

Hozz létre néhány fájlt különböző tartalommal! Számold meg, hogy hány bájt tárhelyet foglalnak az eszközön található fájlok összesen!

Egy lehetséges megoldás:

```
1. import os
2.
3. with open("hello.txt", 'w') as my_file:
4.     my_file.write("Hello, World!")
5.
6. with open("hello2.txt", 'w') as my_file:
7.     my_file.write("Hello, Hello World!")
8.
9. with open("hello3.txt", 'w') as my_file:
10.    my_file.write("Hello, Hello, Hello World!")
11.
12. files = os.listdir()
13. sizeSum = 0
14. for f in files:
15.     sizeSum += os.size(f)
16. display.scroll(str(sizeSum) + " b")
```

A fájlok létrehozása után a `files` változóba kerül az eszközön található fájlok listája. Az összességét méretnek létrehozunk egy változót, ennek kezdetben 0 az értéke. Ezután egy `for` ciklusra van szükségünk, amit ezúttal azért használunk, mert egy sorozat összes elemén végig kell menni. Az elemeken végigiteráló változót `f`-nek nevezzük el. A cikluson belül a méretet tartalmazó változó jelenlegi értékéhez hozzáadjuk az aktuális fájl méretét. Erre szolgál a több más programnyelvben is használatos `+=` operátor. Ennek

használatára a diákok előzetes ismereteinek megfelelően térjünk ki kisebb-nagyobb részletességgel. Végezetül kiírjuk a kijelzőre az összeget, hozzáfűzve a bájt jelét.

Feladat a diákok számára

Hozz létre néhány fájlt különböző tartalommal! Írd ki a kijelzőre a legtöbb karaktert tartalmazó fájl nevét, és azt is, hogy hány karaktert tartalmaz! Figyelj arra, hogy az eszközön mindig megtalálható `main.py` fájlt ne vizsgáld!

Egy lehetséges megoldás:

```
1. import os
2.
3. with open("hello.txt", 'w') as my_file:
4.     my_file.write("Hello, World!")
5.
6. with open("hello2.txt", 'w') as my_file:
7.     my_file.write("Hello, Hello World!")
8.
9. with open("hello3.txt", 'w') as my_file:
10.    my_file.write("Hello, Hello, Hello World!")
11.
12. files = os.listdir()
13. files.remove("main.py")
14. longestName = ""
15. longest = 0
16. for f in files:
17.     with open(f, 'r') as my_file:
18.         content = my_file.read()
19.         if len(content) > longest:
20.             longestName = f
21.             longest = len(content)
22. display.scroll(longestName)
23. display.scroll(longest)
```

A fájlok létrehozásában a kreativitást a tanulókra bízom. A fájlok listáját ismét változóba mentjük, majd a listából eltávolítjuk a `main.py` fájlt, hogy csak az általunk létrehozott fájlokat vizsgáljuk. Ez a fájl minden eszközön megtalálható, magát az eszközre másolt programot tartalmazza (tehát jelen esetben a fenti kódot). A leghosszabb fájlnevnék és a fájl hosszának létrehozunk egy-egy változót a megfelelő kezdőértékekkel. Ezután az előző feladatban már látott módon bejárjuk a listát. A cikluson belül megnyitjuk az egyes fájlokat, beolvassuk a tartalmukat egy változóba, majd ennek a változónak a hosszát fogjuk összehasonlítani az eddigi legnagyobb értékkel. Erre szolgál a `len()` függvény,

ami megmondja egy stringről, hogy hány karakterből áll. Amennyiben egy új leghosszabb fájl tartalmát találtunk, értékül adjuk a változóinknak a fájl nevét és a hosszát. A ciklus végén kiírjuk ezeket az értékeket.

9. alkalom (hétköznapi tárgyak használata)

A micro:bittel való munka nem csak a számítógép előtt ülést jelentheti. A pinek megléte rengeteg egyéb lehetőséget is teremt. Ezen az alkalmon megnézzük, hogyan köthetjük össze kézügyességünket kódolási képességeinkkel, és alkalmazhatunk hétköznapi tárgyakat a programozásban!

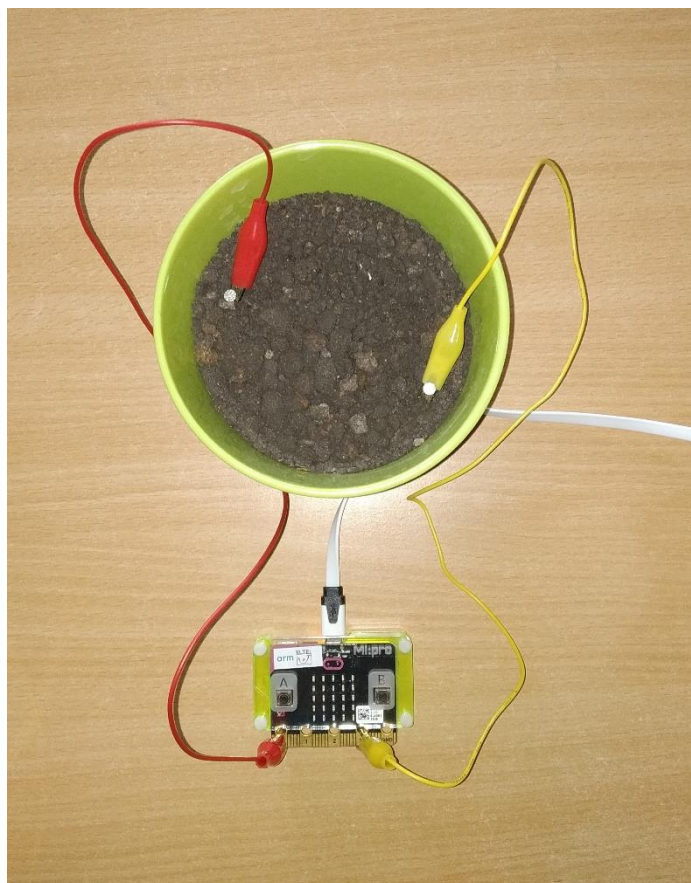
1. Nedvességmérő

Elsőként egy olyan projektet fogunk elkészíteni, ami bármely háztartásban hasznos lehet. Ez nem más, mint egy nedvességmérő, amivel a növényeink földjének nedvességtartalmát tudjuk mérni¹⁸. Amennyiben túl alacsony ez az érték, valamilyen figyelemfelkeltő üzenetet, képet jelenítünk meg a micro:bit kijelzőjén. A projekt elkészítéséhez a következő eszközökre lesz szükségünk:

- 1 micro:bit, elemtartóval és elemekkel, vagy egyéb áramforrással (pl. powerbank)
- 2 hosszabb szög
- 2 krokodilcsipesz
- virágföld
- 1 cserép, vagy kisebb edény a földnek

A kód elkészítése előtt tegyük meg a következő lépéseket! A cserepet töltsük meg virágfölddel. Ügyeljünk rá, hogy mindenképpen száraz legyen, akár pár napot száríthatjuk is ezelőtt. Majd kössük rá az egyik szöget az eszköz 3V pinjére egy krokodilcsipesz segítségével, és szúrjuk bele a földbe. Ezután a másik szöget kössük rá a 0-s pinre, és szintén szúrjuk a földbe (4. ábra). A föld elektromos ellenállását fogjuk mérni a szögek segítségével. Ez a benne található víz és tápanyag mennyiségétől függ. Minél több víz van a földben, annál kisebb az ellenállása. Először megmérjük a száraz földét, majd a földet meglocsoljuk, így a nedves körülményeket is mérni tudjuk. Ezáltal kapunk két küszöbértéket, amikhez a továbbiak folyamán viszonyítani tudunk.

¹⁸ Alapötlet: <https://makecode.microbit.org/projects/soil-moisture> Elérés dátuma: 2019.02.15.



4. ábra A szögek csatlakoztatása

A programunkban a 0-s pinen lévő feszültséget kell leolvasnunk. Ezt a következőképpen tehetjük meg:

```
while True:
    if button_a.was_pressed():
        reading = pin0.read_analog()
        display.scroll(reading)
```

Energiatakarékossági okokból csak az „A” gomb megnyomására végzünk olvasást. A 0-s pinről analóg értéket fogunk leolvasni, ez 0 és 1023 között lehet. Majd ezt az értéket ki is írjuk.

Végezzünk tehát pár mérést száraz földben, majd nedvesben is, és jegyezzük fel őket. A száraz érték körülbelül 250-300 között lesz, míg a nedves érték 1000 felett.

Ezután készítsük el a programot, ami a micro:bit kijelzőjén jelzi az aktuális nedvességszintet! Frissen locsolt föld esetén a teljes kijelző világítson, száraz föld esetén csak a LED mátrix alsó sora! A köztes állapotokban a leolvasott értéknek megfelelően világítson 2, 3, vagy 4 sor!

A megoldás:

```
1. level5 = Image("55555:55555:55555:55555:55555")
2. level4 = Image("00000:55555:55555:55555:55555")
3. level3 = Image("00000:00000:55555:55555:55555")
4. level2 = Image("00000:00000:00000:55555:55555")
5. level1 = Image("00000:00000:00000:00000:55555")
6.
7. while True:
8.     reading = pin0.read_analog()
9.     if reading > 1000:
10.         display.show(level5)
11.     elif reading > 760:
12.         display.show(level4)
13.     elif reading > 530:
14.         display.show(level3)
15.     elif reading > 300:
16.         display.show(level2)
17.     else:
18.         display.show(level1)
```

Először is deklaráljuk az egyes szintekhez tartozó képeket. Érdekes nem a maximális fényerőt alkalmazni, energiatakarékosági okok miatt. A végtelen ciklusban a kalibráló programban látottaknak megfelelően beolvassuk a pinről az értéket, majd egy elágazás segítségével kirajzoljuk az értéknek megfelelő képet. Korábbi méréseinkből tudjuk, hogy 300 alatt lesz a száraz föld, 1000 felett pedig a nedves. A kettő közti intervallumot nagyjából egyenlő részekre osztjuk. Így ahogy szárad ki a föld, fokozatosan egyre alacsonyabb szintet fog jelezni a micro:bit.

Feladat a diákok számára

Oldd meg, hogy száraz föld esetén egy felkiáltójel villogjon a kijelzőn!

Alakítsd át a programot úgy, hogy még energiatakarékosabb legyen!

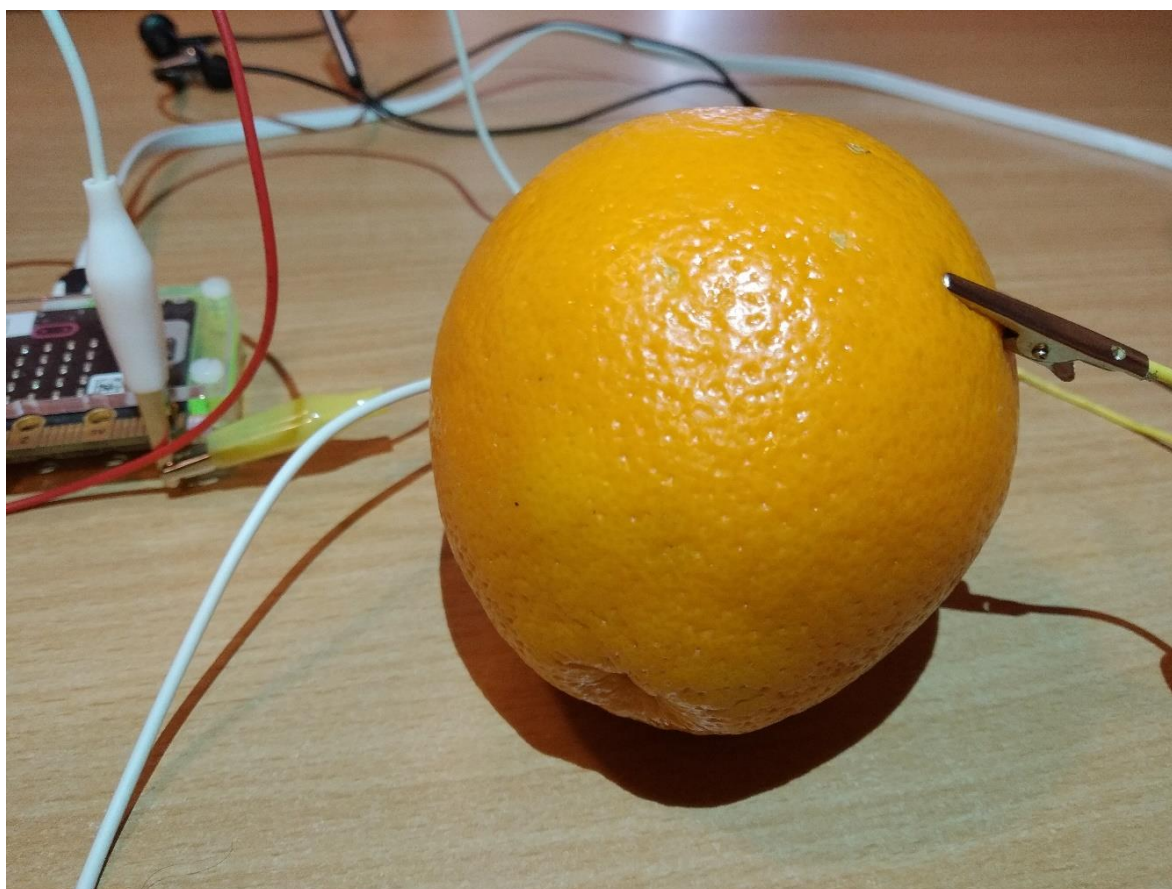
2. Gyümölcszongora

Ebben a projektben egy igazi különlegességet fogunk elkészíteni: banánok és egy narancs segítségével fogunk zenélni! A titok ismét a pinekben rejlik, ezekre fogjuk csatlakoztatni a gyümölcsöket, majd a megérintésükre fogunk reagálni a micro:bittel. A projekt elkészítéséhez ismét szükségünk lesz pár eszközre:

- 1 micro:bit, elemtartóval és elemekkel, vagy egyéb áramforrással (pl. powerbank)
- 1-2 banán

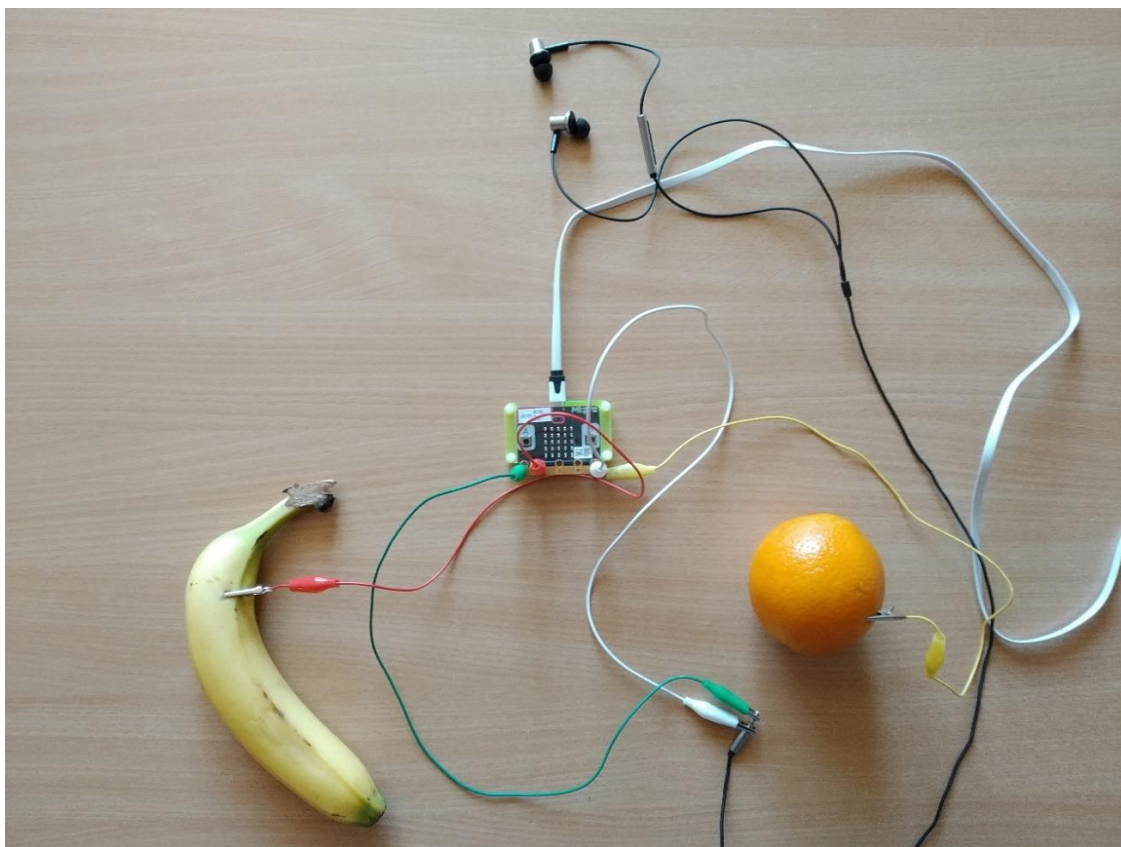
- 1 narancs
- legalább 4 krokodilcsipesz
- fülhallgató vagy hangszóró

Ismét az eszköz összeállításával kell kezdenünk. A már ismert módon csatlakoztassuk a fülhallgatót a micro:bithez, a 0-s és a GND pineket használva. Ezután csatlakoztassuk a narancsot a földeléshez! Ehhez vegyünk egy krokodilcsipeszt, majd az egyik végét csatlakoztassuk a GND pinre csatlakoztatott csipeszhez, a csipesz másik végét pedig „csatlakoztassuk” a narancshoz, az 5. ábrán látható módon.



5. ábra A narancs csatlakoztatása

Végül a negyedik krokodilcsipeszt felhasználva kössük rá a banánt az 1-es pinre, a 6. ábrán látható módon. Ezzel a zongoránk kezdetleges verziója el is készült, próbáljuk ki!



6. ábra A teljes szerkezet

Programunkban az 1-es pinen érkező bemenetet kell vizsgálnunk. Amennyiben érkezik, megszólaltatunk egy hangot:

```
import music

while True:
    if pin1.is_touched():
        music.play("C4:4")
        display.show(Image.HAPPY)
```

A kódból már minden elem ismerős a korábbi alkalmakról, most egyszerűen összegyűrtük őket. Amennyiben az 1-es pinen bemenetet érzékelünk, lejátszunk egy megadott hangot. Opcionálisan egy ikon megjelenítésével is kísérhetjük ezt.

Fogjuk meg a narancsot az egyik kezünkben, majd a másik kezünk egyik ujjával érintsük meg a banánt! Az ujjainkat váltogatva úgy játszhatunk le hangokat, mintha csak egy zongorán játszanánk. Persze ez egy banánon nem sokáig izgalmas. Kössünk rá tehát további banánokat a többi pinre! Mindegyikre programozunk más hangot, lehetőleg magasság szerint sorba állítva. A banánok egy-egy zongorabillentyűt fognak jelképezni. Több micro:bit használatával egy egész zongorát is megalkothatunk, amin egy hozzáértő gyerek akár el is tud játszani valamit!

10. alkalom (játékkészítés)

Ezúttal a diákok saját projektjei előtt közösen fogunk elkészíteni egyet, méghozzá egy olyan témában, ami eddig nem került szóba: a játékkészítés. A micro:biten lehetőségünk van jónéhány ismert játék elkészítésére, emellett a saját ötleteink megvalósítására is van mód. Ebben a fejezetben egy jól ismert okostelefonos játék, a Flappy Bird micro:bit-es verzióját fogjuk leprogramozni. Ebben a játékban egy kis madárral kell csövek közt átrepülnünk. A haladást nehezíti, hogy a madár folyamatosan zuhan, egy-egy kattintással csap egyet a szárnyával, aminek hatására egy kicsit emelkedik. Ez a játék sokáig a legnépszerűbb volt mindkét okostelefonos platformon, ám azóta az alkotó törölte azt¹⁹, így jelenleg hivatalos forrásból az eredeti programot nem tudjuk beszerezni. Szerencsére létezik egy böngészőben játszható verziója²⁰, így ki tudjuk próbálni az eredeti verzió mását a programozás előtt. Természetesen szükségünk lesz apróbb változásokra, hiszen a grafikus felület helyett csupán egy 5x5-ös LED mátrix áll a rendelkezésünkre. Alkossuk meg a játékot, lépésről lépésre haladva!²¹

1. Üdvözlő üzenet, a játék elkezdése

Nagyon nehézé válna a játék, ha egyből a micro:bit áram alá helyezésénél elindulna, ezért célszerű valamilyen üdvözlő üzenetet, esetleg visszaszámlálást megjeleníteni a játék kezdete előtt. A szokásos import után jelenítsük meg ezt:

```
display.scroll("3...2...1...GO!", 75)
```

A második paraméterként szereplő számmal tudjuk a szöveg görgetésének sebességét változtatni. Ennek az alapértelmezett értéke 150, így itt az alap sebesség duplájával pörög a szöveg a kijelzőn. Ezt az értéket személyes preferenciáinknak megfelelően szabadon változtathatjuk, akár csak a megjelenő szöveget.

2. A madár megjelenítése

Kezdjük el a kijelzőn megjeleníteni a játék elemeit, elsőként a madarat! Mivel a madár vízszintesen nem, csak függőlegesen mozog, ezért csak azt kell eltárolnunk, hogy az y tengelyen hol található. Habár csak 5 soros a kijelző, a madár pozícióját 0 és 99 között

¹⁹ <https://twitter.com/dongatory/status/432227971173068800> Elérés dátuma: 2019.02.05.

²⁰ <https://flappybird.io/> Elérés dátuma: 2019.02.05.

²¹ <https://blog.withcode.uk/2016/05/flappy-bird-microbit-python-tutorial-for-beginners/> Elérés dátuma: 2019. 02. 05.

fogjuk eltárolni, majd ebből számoljuk ki, hogy melyik sorban legyen. Ezáltal realisztikusabb mozgás érhető el, mivel nemcsak 5 féle értékkel tudunk számolni. Egészítsük ki a kódot a madár megjelenítésével!

```
y = 50
led_y = int(y / 20)
display.set_pixel(1, led_y, 9)
```

Az `y` változóba megadjuk a madár kezdőpozícióját, a képernyő közepén, hiszen itt kezd a játékot. A `led_y` változóban fogjuk tárolni, hogy melyik LED-en kell szerepelnie a madárnak. Ehhez az `y` értékét 20-szal osztjuk, mivel egy 0-99 közti értékből szeretnénk egy 0 és 4 közötti előállítani (tehát 100 féle értékből 5 félért). Az `int()` függvénnyel ezt a számot egészen kerekítjük, hiszen a LED koordinátája csak egész szám lehet. Végül, az utolsó sorban felvillantjuk a madár helyét a LED mátrixon. Az `x` koordináta 1 lesz, így a második oszlopban lesz a madarunk, jobban követhető lesz, mintha a képernyő legszélén lenne. Az `y` koordinátát korábban kiszámoltuk, a világosság értéke pedig a maximális, 9 lesz. A csövek ennél halványabbak lesznek, így könnyű lesz megkülönböztetni a szereplőket a pálya részeitől.

3. A madár mozgatása

Sikeresen megjelenítettük a madarunkat, azonban ez még csak egy égő pont a LED-ek közt. Kezdjük el hát mozgatni! Első lépésként a gravitációt fogjuk szimulálni, hogy a madár elkezdjen zuhanni a föld felé. Ehhez elő kell vennünk a végtelen ciklust, hiszen a játék folyamatosan fog játszódni. Vezessük be egy változót a sebesség tárolására, ennek az értéke legyen 1. A cikluson belül a madár pozícióját mindig a sebességnek megfelelően változtassuk! Ha így lefuttatjuk a programot, két hibát vehetünk észre: az egyik, hogy a madár ugyan esik a föld felé, azonban az előző pozíciója is felvillantva marad, így egy vonal rajzolódik ki. A másik hiba pedig az, hogy a madár „ki tud esni” a kijelzőről, így az `y` értékét maximalizálnunk kell. Kérdezzük meg a tanulókat arról, hogy hogyan javítanák ki ezeket a hibákat! A jó válaszok megtalálása után kódoljuk is le őket. A végeredménynek így kell kinéznie:

```
1. display.scroll("3...2...1...GO!",75)
2.
3. y = 50
4. speed = 1
5.
```



```

6. while True:
7.     display.clear()
8.     y += speed
9.     if y > 99:
10.        y = 99
11.    led_y = int(y / 20)
12.    display.set_pixel(1, led_y, 9)
13.    sleep(20)

```

Ezzel a madár már esik ugyan, azonban nem túl valósághű, hiszen konstans 1 sebességgel teszi ezt. A valóságban a tárgyak esés közben egyre gyorsabbak. Növeljük hát esés közben a sebességet! Kezdetben állítsuk be 0-ra, a ciklus első lépéseként növeljük, és ne hagyjuk, hogy 2-nél több legyen. A feladat önálló munka keretében megoldható.

A madár lefelé mozgásával elkészültünk, azonban játszhatatlan lenne a játék, ha nem tudna csapkodni a szárnyaival, vagyis készítsük el a felfelé mozgatást! A madár az „A” gomb lenyomására csapjon egyet a szárnyával, ezáltal kis löketet kapva felfelé!

```

1. display.scroll("3...2...1...GO!",75)
2.
3. y = 50
4. speed = 0
5.
6. while True:
7.     display.clear()
8.     if button_a.was_pressed():
9.         speed = -8
10.
11.    speed += 1
12.    if speed > 2:
13.        speed = 2
14.    y += speed
15.    if y > 99:
16.        y = 99
17.    if y < 0:
18.        y = 0
19.    led_y = int(y / 20)
20.    display.set_pixel(1, led_y, 9)
21.    sleep(20)

```

A 8. sortól kezdődően egészítettük ki a programot: amennyiben az „A” gombot lenyomtuk, a sebességet állítsuk -8-ra! Ezáltal a madár felrepül, de a gravitáció máris elkezd dolgozni rajta, és megkezd a zuhanást. Mivel most már felfelé is tud menni a madár, meg kell akadályozni, hogy felül kirepüljön a kijelzőről, ezt a 17-18. sorban tettük meg. Ezzel a madár megjelenítésével és mozgásával el is készültünk.

4. A csövek megjelenítése

Most pedig térjünk át a játék másik szereplőjére, a csövekre. Ezek fognak majd egyre közeledni a madárhoz, aminek az ezeken lévő lyukakon kell keresztül repülnie. A programon belüli megvalósításuk úgy fog történni, hogy a kijelző jobb szélére kirajzolunk egy függőleges vonalat, amibe véletlenszerűen egy 2 pixel magas lyukat fogunk elhelyezni. Kérdezzük meg a diákoktól az ehhez szükséges modul nevét!

Ezúttal egy új funkcióval is megismerkedünk a programozás során, a saját függvény készítésével. Beszéljünk a tanulókkal arról, hogy ez miért célszerű a programozás során! A Pythonban ezt a `def` kulcsszóval érhetjük el, ezután adhatjuk meg a függvény nevét és a paramétereit, a visszatérési érték típusa természetesen nem szükséges. Nézzük a kódot:

```
1. import random
2.
3. display.scroll("3...2...1...GO!",75)
4.
5. y = 50
6. speed = 0
7.
8. def make_pipe():
9.     pipe = Image("00003:00003:00003:00003:00003")
10.    gap = random.randint(0,3)
11.    pipe.set_pixel(4, gap, 0)
12.    pipe.set_pixel(4, gap+1, 0)
13.    return pipe
14.
15. pipe = make_pipe()
16.
17. while True:
18.     display.show(pipe)
19.     if button_a.was_pressed():
20.         speed = -8
21.
22.     speed += 1
23.     if speed > 2:
24.         speed = 2
25.     y += speed
26.     if y > 99:
27.         y = 99
28.     if y < 0:
29.         y = 0
30.     led_y = int(y / 20)
31.     display.set_pixel(1, led_y, 9)
32.     sleep(20)
```

A programba importáltuk a `random` modult. A csőkészítő függvényünk a 8-13. sorban található. Először létrehozuk a vonalat a kijelző jobb szélén, a teljes fényerősség harmadán. Ahogy már korábban említettük, erre azért van szükség, hogy a cső könnyen megkülönböztethető legyen a madártól. Ezután választunk egy 0 és 3 közti véletlenszámot. Azért 3 a felső határ, mert a választott számú és az eggyel nagyobb koordinátájú LED-et fogjuk lekapcsolni, így ha a 4-et is megengednénk, már a kijelzőn kívülre hivatkoznánk. A választás után a `set_pixel()` metódussal lekapcsoljuk (0 fényerősségre állítjuk) a kiválasztott LED-eket. Végül a `return` utasítással visszaadjuk a kész képet, ami majd a csőként fog funkcionálni. További módosításként még a ciklus előtt „elkérünk” egy csövet a függvénytől, majd a cikluson belül a képernyő törlése helyett kirajzoljuk a csövet. A törlésre azért nincs szükség, mert a cső kirajzolása letöröl mindent a kijelzőről, így a madarunk előző pozíciója sem fog látszódni már. Minden szereplőnk ott van a kijelzőn, így jöhet a következő lépés.

5. A csövek mozgatása, az ütközés vizsgálata

Ebben a lépésben tehát elkezdjük mozgatni a csöveket, és megvizsgáljuk, hogy a madarunk beleütközött-e valamelyikbe.

Először is szükségünk lesz arra, hogy számolni tudjuk, hányszor futott már le a játék fő ciklusa. Erre bevezetjük a `frame` (képkocka) nevű változót, ami 0-ról indul, és minden futásnál eggyel növekszik. Ezeknek a számolásával tudjuk majd beállítani a játék sebességét, nehézségét. Ehhez természetesen konstansokra is szükségünk van. Ezen felül a ciklust ki kell egészítenünk a cső mozgatásával, valamint az új cső készítésével. A kód kezd kicsit meghízni, az új sorokat félkövéren jelöljük, továbbá a főciklus nagysága miatt kommenteket is bevezetünk (a saját függvények fontossága máris megmutatkozik!):

```
1. import random
2.
3. DELAY = 20
4. FRAMES_PER_PIPE_MOVE= 20
5. FRAMES_PER_NEW_PIPE = 100
6.
7. display.scroll("3...2...1...GO!",75)
8.
9. y = 50
10. speed = 0
11. frame = 0
12.
```



```

13. def make_pipe():
14.     pipe = Image("00003:00003:00003:00003:00003")
15.     gap = random.randint(0,3)
16.     pipe.set_pixel(4, gap, 0)
17.     pipe.set_pixel(4, gap+1, 0)
18.     return pipe
19.
20. pipe = make_pipe()
21.
22. while True:
23.     frame += 1
24.
25.     display.show(pipe)
26.
27.     # szarnycsapas a gombra
28.     if button_a.was_pressed():
29.         speed = -8
30.
31.     # gravitacio
32.     speed += 1
33.     if speed > 2:
34.         speed = 2
35.
36.     # madar mozgatasa, de maradjon a kijelzon
37.     y += speed
38.     if y > 99:
39.         y = 99
40.     if y < 0:
41.         y = 0
42.
43.     # madar kirajzolasa
44.     led_y = int(y / 20)
45.     display.set_pixel(1, led_y, 9)
46.
47.     # cso mozgatasa balra
48.     if (frame % FRAMES_PER_PIPE_MOVE == 0):
49.         pipe = pipe.shift_left(1)
50.
51.     # uj cso létrehozasa
52.     if (frame % FRAMES_PER_NEW_PIPE == 0):
53.         pipe = make_pipe()
54.
55.     sleep(20)

```

A konstansok a 3-5. sorban találhatóak. A Python konvencióinak megfelelően ezek csupa nagybetűvel, a szavak közt alulvonással írandóak. A DELAY mondja meg, hogy két frame közt mennyi idő teljen el, a FRAMES_PER_PIPE_MOVE értéke azt mutatja, hogy hány frame teljen el amíg a cső eggyel balra mozdul, míg a FRAME_PER_NEW_PIPE értéke a

két cső előállítás között eltelt frame-ek számát mutatja. Ezeket az értékeket használjuk a ciklusban. a 48. sorban eldöntjük, hogy szükséges-e ebben a frame-ben mozgatni a csövet. Ezt a maradékos osztás műveletével egyszerűen megtehetjük. Az `Image` osztály `shift_left()` metódusával a balra mozgatást könnyedén megoldhatjuk. Hasonlóképpen járunk el az új cső létrehozásánál is. Ha már sikeresen mozognak a csövek, vizsgáljuk meg az ütközést a madárral! Ehhez egészítsük ki a főciklust, a madár kirajzolása után, a következőkkel:

```
if pipe.get_pixel(1, led_y) != 0:
    display.show(Image.SAD)
    break
```

Amennyiben a csövet tartalmazó kép 1. oszlopában (a kijelző 2. oszlopában, tehát a madarat tartalmazó oszlopban), a `led_y` sorban, azaz a madár aktuális helyén nem 0 a világosságérték, akkor ott bizony a csőnek egy darabja van. Ilyenkor sajnos ütköztünk a csővel. Ezt kísérje egy szomorú arc megjelenítése a kijelzőn, majd álljon le a játék! Ezzel elkészült a saját Flappy Bird-ünk, ami a micro:bit-en játszható. Azonban egy dolog még hiányzik egy jó játékból. Kérdezzük meg a tanulóktól, hogy mi lehet az!

Feladat a diákok számára

Egészítsd ki a játékot pontszámítással! Minden cső, amin átrepültünk érjen egy pontot!
A játék végén kerüljön megjelenítésre az elért pontszám!

Az elkészült játék teljes kódja:

```
1. from microbit import *
2. import random
3.
4. DELAY = 20
5. FRAMES_PER_PIPE_MOVE= 20
6. FRAMES_PER_NEW_PIPE = 100
7.
8. # display.scroll("3...2...1...GO!",75)
9.
10. y = 50
11. speed = 0
12. frame = 0
13. score = 0
14.
15. def make_pipe():
16.     pipe = Image("00003:00003:00003:00003:00003")
```

```

17.     gap = random.randint(0,3)
18.     pipe.set_pixel(4, gap, 0)
19.     pipe.set_pixel(4, gap+1, 0)
20.     return pipe
21.
22. pipe = make_pipe()
23.
24. while True:
25.     frame += 1
26.
27.     display.show(pipe)
28.
29.     # szarnycsapas a gombra
30.     if button_a.was_pressed():
31.         speed = -8
32.
33.     # gravitacio
34.     speed += 1
35.     if speed > 2:
36.         speed = 2
37.
38.     # madar mozgatasa, de maradjon a kijelzon
39.     y += speed
40.     if y > 99:
41.         y = 99
42.     if y < 0:
43.         y = 0
44.
45.     # madar kirajzolasa
46.     led_y = int(y / 20)
47.     display.set_pixel(1, led_y, 9)
48.
49.     # utkozes vizsgalata
50.     if pipe.get_pixel(1, led_y) != 0:
51.         display.show(Image.SAD)
52.         sleep(500)
53.         display.scroll(score)
54.         break
55.
56.     # cso mozgatasa balra
57.     if (frame % FRAMES_PER_PIPE_MOVE == 0):
58.         pipe = pipe.shift_left(1)
59.
60.     # uj cso létrehozasa
61.     if (frame % FRAMES_PER_NEW_PIPE == 0):
62.         pipe = make_pipe()
63.         score += 1
64.
65.     sleep(20)

```

11-12. alkalom (saját projektek megvalósítása)

A diákok a szakkör utolsó két alkalmán párokban, esetleg 3 fős csoportokban készítsenek el egy közös projektet. A különféle ötleteknek csak a diákok kreativitása és a rendelkezésre álló eszközök szabhatnak határt. Amennyiben az előző szakkör végén felhívjuk rá a figyelmet, akár előre is átgondolhatják a tanulók, hogy mit szeretnének csinálni, így erre az alkalomra már tudnak magukkal különféle eszközöket hozni, legyen az alufólia, kartondoboz, ragasztószalag, vagy valami teljesen más, ahogy a gyümölcshangszernél is láthattuk. A szakkörön tanult funkciók segítségével programok nagyon széles skálája készíthető el. Összegzőképpen álljon itt egy lista a tanult elemekből:

- LED-ek felvillantása, képek, animációk megjelenítése a kijelzőn
- gombnyomások kezelése
- PIN-ek használata különféle eszközök csatlakoztatására
- véletlen választás, véletlenszám-generálás
- dallamok, hangok lejátszása
- gyorsulásmérő használata, gesztusok
- iránytű
- rádió, kommunikálás az eszközök között
- fájlkezelés
- játékkészítés

Természetesen az internet is tele van projektötletekkel. Kiindulásképp bátran meríthetünk a MakeCode²² oldalról, de egyéb forrásokból is rengeteg izgalmasabbnál izgalmasabb ötletet szerezhetünk.

Ezen a két alkalmon lehetőség van a projektek elkészítésére, finomítására. Eközben a tanár monitorozó szerepet tölt be, segít a csapatoknak, akár ötletekkel, akár megvalósítással kapcsolatban. Az utolsó alkalom végén a diákok tartsanak bemutatót, majd próbálják ki egymás alkotásait. A gyerekeknek garantált élmény lesz, hogy kezükbe foghatják az általuk alkotott „művet”.

²² <https://makecode.microbit.org/> Elérés dátuma: 2019. 02. 05.

Összefoglalás

A kész projektek bemutatásával szakkörünk végére értünk. Most már elmondhatjuk, hogy sikeresen megismerkedtünk a micro:bit és a MicroPython szinte minden funkciójával, miközben megismerkedtünk a programozás alapjaival is. Természetesen, ha lehetőségünk van rá és kedvünk tartja, nyugodtan folytathatjuk az eszközzel való kísérletezést további alkalmakon. Reményeim szerint sokkal szórakoztatóbb volt ilyen módon, egy kézzel fogható eszköz segítségével megtanulni ezt a sok új ismeretet, mintha csak a számítógép monitorán, egy virtuális térben történtek volna az események. Ne álljunk meg azonban itt! Mutassunk utat az érdeklődő diákoknak, akik szeretnének tovább foglalkozni akár a micro:bittel, akár a programozással. A tanulóknak lehetősége van a microPython további lehetőségeivel is megismerkedni, mert bár a legtöbb funkció bemutatásra került, azért bőven akadnak még izgalmas, felfedezésre váró funkciói mind a nyelvnek, mind az eszköznek. Remélhetőleg az itt megszerzett ismereteket, készségeket a tanulók más tantárgyaik, sőt akár a mindennapjaik keretében is fel tudják majd használni.

Irodalomjegyzék

- [1] Sentance, S., Waite, J., Hodges, S., MacLeod, E., & Yeomans, L. E. (2017). "Creating Cool Stuff" - Pupils' experience of the BBC micro:bit. In Proceedings of the 48th ACM Technical Symposium on Computer Science Education: SIGCSE 2017 DOI: 10.1145/3017680.3017749
- [2] Gibson, S., Bradley, P. (2017) 'A study of Northern Ireland Key Stage 2 pupils' perceptions of using the BBC micro:bit in STEM education', The STeP Journal, 4(1), pp.15-41.
- [3] Czékmán, B. & Kiss, J. "Digitális eszközök használata az osztálytermekben. Egy BBC micro: bites projekt tapasztalatai." Educatio 27.1 (2018): 111-120.
- [4] Mannila, Linda & Peltomäki, Mia & Back, Ralph-Johan & Salakoski, Tapio. (2006). Why Complicate Things? Introducing Programming in High School Using Python. Conferences in Research and Practice in Information Technology Series. 52.
- [5] Dr. Abonyi-Tóth Andor: Programozzunk micro:biteket!, ELTE Informatikai Kar, 2018, ISBN 978-963-284-992-8. <http://microbit.inf.elte.hu/wp-content/uploads/2018/05/Programozzunk-microbiteket-2018.pdf> Elérés dátuma: 2018.07.14.