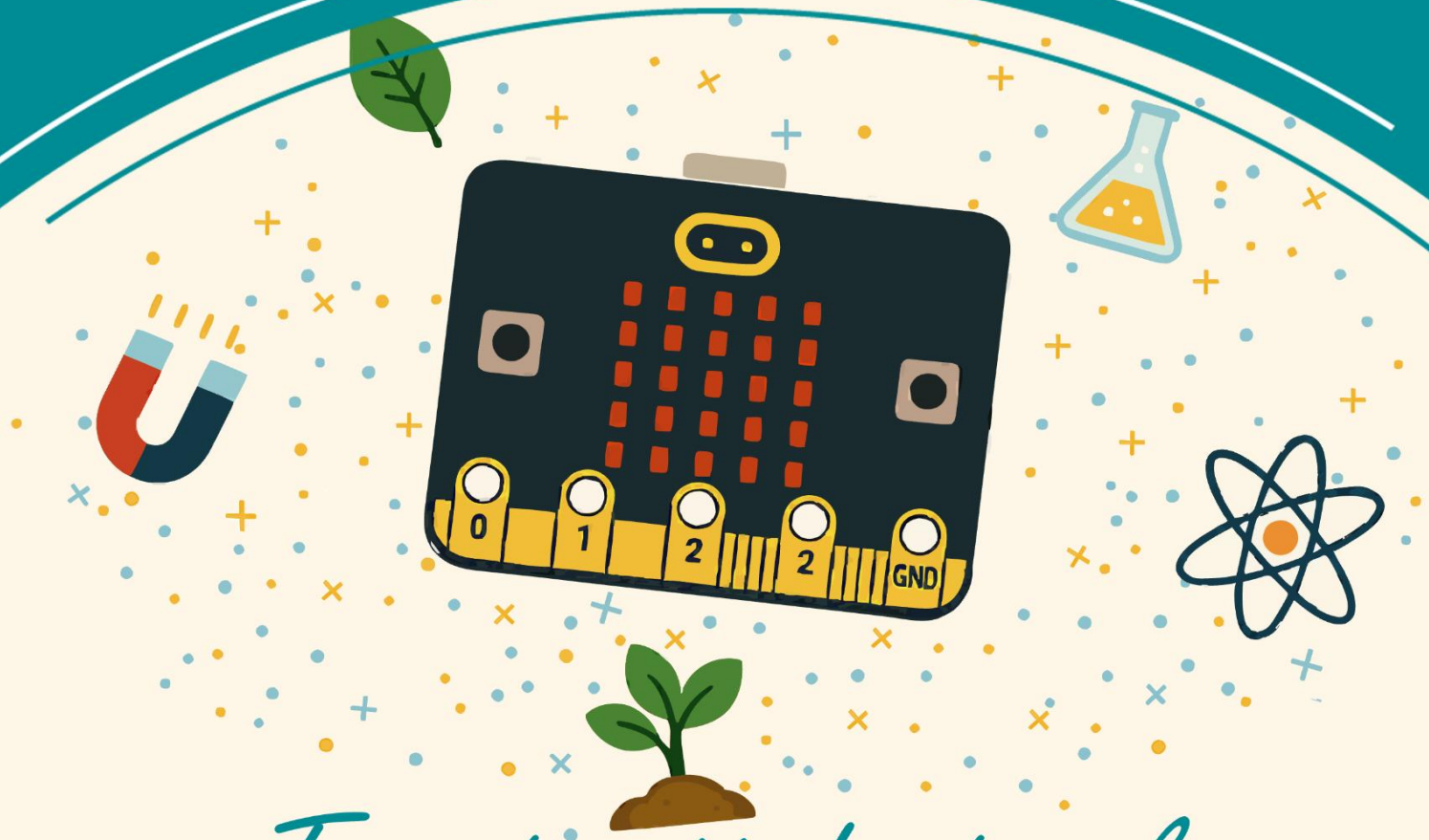
ELTE | IK



Természettudományok micro:bittel

Foglalkozástervek tanárok számára

Gaál Bence
ELTE Eötvös Loránd Tudományegyetem
Informatika Kar

A kiadvány „A KULTURÁLIS ÉS INNOVÁCIÓS MINISZTÉRIUM EKÖP-24 KÓDSZÁMÚ EGYETEMI KIVÁLÓSÁGI ÖSZTÖNDÍJ PROGRAMJÁNAK A NEMZETI KUTATÁSI, FEJLESZTÉSI ÉS INNOVÁCIÓS ALAPBÓL FINANSZÍROZOTT SZAKMAI TÁMOGATÁSÁVAL KÉSZÜLT.”

EKÖP
Egyetemi Kutatói Ösztöndíj Program

Természettudományok micro:bittel

Szerző

Gaál Bence

Felelős kiadó

ELTE Informatikai Kar
1117 Budapest, Pázmány Péter sétány 1/C.

ISBN szám

ISBN 978-963-489-828-3

A kiadvány „A KULTURÁLIS ÉS INNOVÁCIÓS MINISZTÉRIUM EKÖP-24 KÓDSZÁMÚ EGYETEMI KIVÁLÓSÁGI ÖSZTÖNDÍJ PROGRAMJÁNAK A NEMZETI KUTATÁSI, FEJLESZTÉSI ÉS INNOVÁCIÓS ALAPBÓL FINANSZÍROZOTT SZAKMAI TÁMOGATÁSÁVAL KÉSZÜLT.”

EKÖP
Egyetemi Kutatói Ösztöndíj Program

ELTE | IK
INFORMATIKAI KAR

Tartalom

Ábrajegyzék

1 Előszó

A mai diákok digitális eszközök között nőnek fel. Sokan magabiztosan mozognak a technológia világában, és gyakran úgy tűnik, mintha „digitális bennszülöttek” lennének. Tanárként azonban gyakran tapasztaljuk, hogy ez a magabiztosság csupán a felszínt érinti. A valódi digitális kompetencia ennél sokkal többet jelent: kreatív problémamegoldást, tudatos kommunikációt, együttműködést, valamint azt, hogy megértjük a minket körülvevő technológiák működését.

Ugyanilyen fontos, hogy a természettudományokat ne csupán tantárgyként kezeljük, hanem élő, változó, a valósághoz szorosan kapcsolódó tudásterületként. Sajnos sok tanulónál azt látjuk, hogy az érdeklődés e tantárgyak iránt az idő előrehaladtával csökken – különösen a középiskolában. Ennek egyik oka az lehet, hogy a tananyag egyre elvontabbá válik, és eltávolodik a tapasztalati tanulástól.

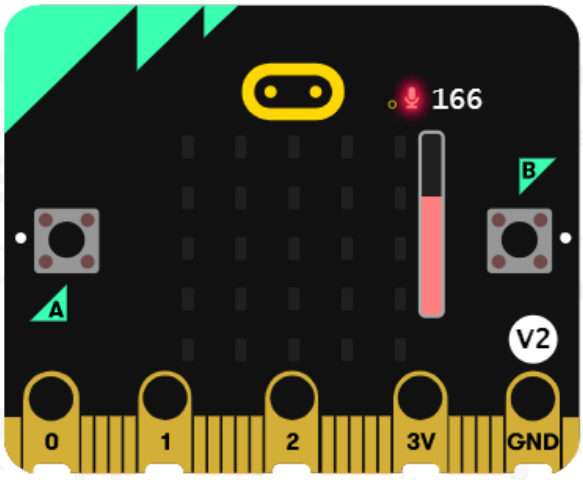
Én magam ezt a szakadékot próbáltam áthidalni, amikor a robotikát – elsősorban a micro:bit eszközt – kezdtem alkalmazni a természettudomány tantárgy tanítása során. Tapasztalataim szerint a diákok sokkal aktívabban és motiváltabban vettek részt az órákon – még azok is, akik korábban nem mutattak különösebb érdeklődést sem a programozás, sem a természettudományok iránt. A robotika kézzelfoghatóvá tette számukra az elvont fogalmakat, és a tanulás élménnyé vált.

Ez a kiadvány is ebben a szellemben született. Alapját a doktori disszertációm adja, amelyben hét, a robotikát a természettudományos oktatásba integráló foglalkozást dolgoztam ki és próbáltam ki a gyakorlatban. A mostani anyag ezt a munkát egészíti ki további micro:bit-alapú foglalkozásötletekkel.

Célom nem csupán az volt, hogy érdekes és játékos foglalkozásterveket kínáljak, hanem az is, hogy megmutassam: akár olcsó eszközökkel és nem csak informatikatanárok számára adott a lehetőség arra, hogy a tanítás élményszerű, alkotó és korszerű legyen. Bízom benne, hogy a kiadványban szereplő foglalkozások és a hozzá kapcsolódó leírások olyan eszközt adnak a pedagógusok kezébe, amely segíti őket abban, hogy a tudományt közelebb hozzák a diákokhoz – és talán saját tanári motivációjukat is új lendülettel töltsék meg.

2 Felhasznált eszközök

2.1 micro:bit v2

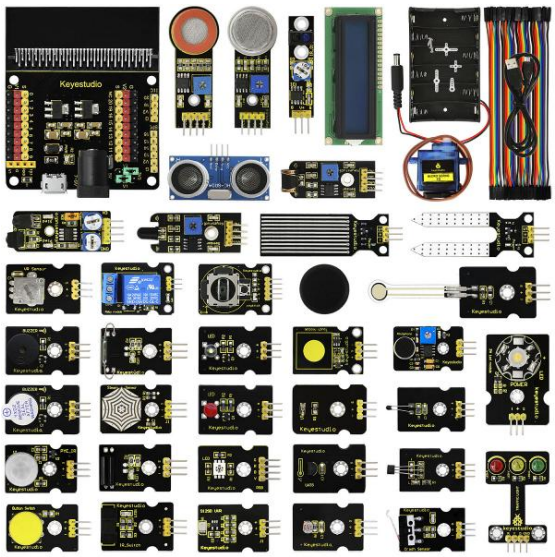


1. ábra micro:bit v2
(*Forrás: http://microbit.inf.elte.hu/2020/11/26/microbit-v2-tobbet-jobban-gyorsabban-tovabbra-is-olcson)*

Bővebb információk a v2-es modellről:

- Micro:bit V2 – többet, jobban, gyorsabban, továbbra is olcsón… – micro:bit botorkálás
- The new micro:bit V2 : Help & Support

2.2 keyestudio 37 in 1 Starter Kit for BBC micro:bit



2. ábra A Kit tartalma
(*Forrás:* KS0361(KS0365) keyestudio 37 in 1 Starter Kit for BBC micro:bit)

A csomag Wiki oldalán bővebb információk és példaprogramok is elérhetők.

3 Foglalkozások

A kiadványban szereplő minden foglalkozás egységes felépítést követ, amely segíti az eligazodást és a gyors áttekinthetőséget. Minden foglalkozás elején egy összefoglaló táblázat található, amely megadja a természettudományos témát, a szemléltető modellt, a szükséges eszközöket, valamint az informatikai alapismereteket. Emellett rögzíti azokat a digitális kultúrához kapcsolódó célokat is, amelyek a tanóra során fejlesztésre kerülnek.

A táblázatot egy rövid bevezető követi, amely segít természettudományi kontextusba helyezni a foglalkozást, és megmutatja, hogyan kapcsolódik az adott témakör a tantervi tartalmakhoz. Ahol szükséges ott pedig kapcsolási rajz[1] is található.

Ezt követi a **konkrét megvalósítási leírás**, amely lépésről lépésre bemutatja az eszközök összeállítását és a micro:bit programozását. A leírásokat úgy kerültek összeállításra, hogy azok **informatika/digitális kultúra végzettséggel nem rendelkező pedagógusok számára is követhetőek legyenek**. A kódokat és az építési lépéseket közérthető magyarázat kíséri, így a foglalkozások biztonsággal beilleszthetők a tanítási gyakorlatba, akár tanórán, akár szakköri keretek között. A felhasznált kódblokkok a szerkesztővel színazonosan kerültek jelölésre. A foglalkozások végén található egy összegző fogalomtár az informatikai fogalmakhoz, amelyekről szó esik az adott foglalkozáson.

A foglalkozásokhoz a példakódok letölthetők az alábbi linken: gaalben/termtud_microbit

A foglalkozások előtt ajánlott Dr. Abonyi-Tóth Andor Programozzunk micro:biteket! szakköri segédanyagának megtekintése, ahol részletes leírást kapunk az eszköz és a kezelőfelület felépítéséről, valamint az eszköz általános programozásáról. Szintén elérhetők további segédanyagok a micro:bit botorkálás[2] weboldalon.

A foglalkozások közül az utolsó kettő haladó szintűnek tekinthető, mivel megvalósításuk nem blokkos, hanem szöveges programozási környezetben, a Python nyelv használatával történik, és több – a kezdő szintű oktatásban nem jellemző – külső könyvtár integrálását is megköveteli.

[1] A képek saját szerkesztésűek, a felhasznált alapképek forrása: KS0361(KS0365) keyestudio 37 in 1 Starter Kit for BBC micro:bit

[2] A kezdeményezés célja, hogy a BBC micro:bit eszköz (ha csak átmeneti használatra is), minél több iskolába eljuthasson, hogy a diákok játékos programozási/kódolási tevékenységeken keresztül megismerhessék, kipróbálhassák az eszközben rejlő lehetőségeket, ezzel is fejlesztve az informatikai gondolkodásuk

3.1 Ciklon vs. Anticiklon

Fő tantárgy, ahol megjelenhet	Földrajz
A foglalkozás helye	A légkör– Ciklonok, anticiklonok, trópusi ciklonok
Szemléltető eszköz	A levegő áramlását bemutató animáció (*8. ábra*)
Felhasznált eszközök a modell elkészítéséhez	• 1 db micro:bit V2
Programozáshoz köthető ismeretek	• elágazások • függvény alapvető használata • feltételes ciklusok • eseményvezérlés alapjai
A foglalkozás digitális kultúrához kapcsolódó céljai	• A diákok képesek legyenek változókat és listákat létrehozni. • Tudjanak egyszerű eljárásokban tárolni kódrészleteket, azokat meghívni. • Váljanak képessé egyszerű feltételmegadásokra, feltételes ciklusok használatára.

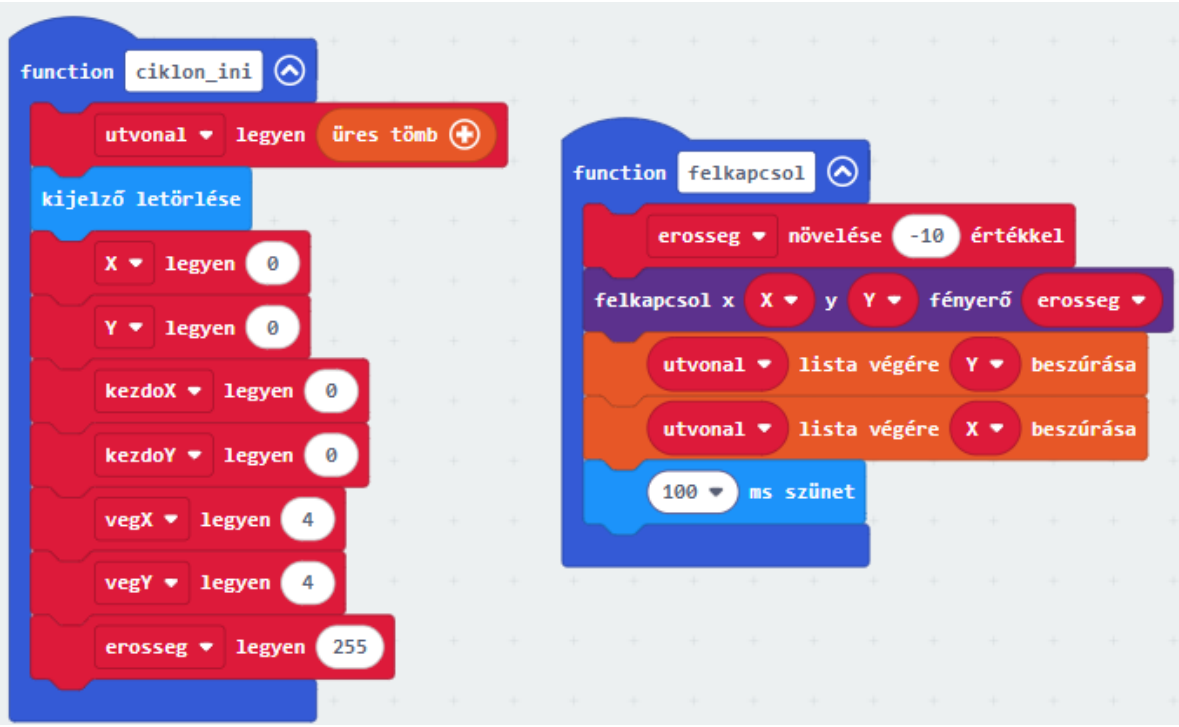
A foglalkozásról röviden

A tanulók egy animált szemléltető eszközt készítenek, amelyen keresztül megfigyelhetővé válik a ciklonok és anticiklonok jellegzetes légmozgása. A micro:bit kijelzőjén megjelenített áramlási irány segíti a magas és alacsony nyomású területekhez kapcsolódó mozgások és időjárási jelenségek megértését. A tanulók ezáltal aktívan dolgozzák fel a légköri folyamatokat, és könnyebben azonosítják a ciklonok és anticiklonokhoz kapcsolódó tipikus mintázatokat.

A program elkészítése

A program működésénél az anticiklon megvalósítása nagyon mély ismeretekre támaszkodna, ezért egy egyszerűsített módot valósítottunk meg, ahol a ciklon mozgását, eltároltuk egy listában/tömbben. Ez azt a korlátot támasztotta elénk, hogy az anticiklon animációjának lejátszására csak a ciklon animációja után van lehetőségünk.

Két saját függvényt hozunk(*3. ábra*) létre ciklon_ini és felkapcsol néven. Az első függvény a kezdeti beállításokért lesz felelős, míg a másik a micro:bit kijelzőjén fogja felkapcsolni a LEDeket megfelelő fényerősséggel és eltárolja az útvonalunkat.



3. ábra A két saját függvényünk programkódja

A ciklon_ini-ben szükséges, hogy létrehozzuk az üres listánkat az utvonal legyen üres tömb parancsal, hogy letöröljük a kijelzőt és hogy megadjuk a változóink kezdőértékét a valtozonev_legyen blokkal. Nézzük, hogy melyik változó mire is jó. Ne feledjük, hogy a ciklon esetében egy spirális mozgást kell megjeleníteni ciklusok segítségével a micro:bit képernyőjén, ezért nem csak az aktuális pozíciót kell tárolni, hanem a határokat is, hogy az animáció egyre kisebb területen haladjon.

1. Pozíció tárolása (x, y):

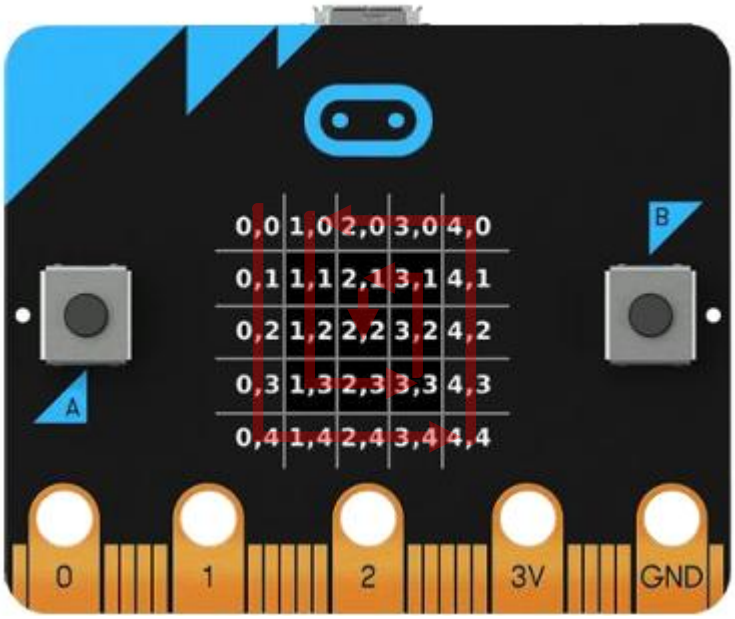
A Micro:bit LED mátrixán (*4. ábra*) egy adott LED pozícióját koordinátákkal (x, y) adjuk meg. Ezek a változók tárolják, hogy éppen melyik LED-et kapcsoljuk be vagy ki.

2. Ciklusok határértékének meghatározása (kezdoX, kezdoY, vegX, vegY):

- **kezdoX és kezdoY (kezdő koordináták):**
    - Ezek jelzik, honnan indul a spirális mozgás aktuális ciklusa.
    - A spirál kezdetén kívülről indul (a teljes kijelző határán: kezdoX=0, kezdoY=0), majd a spirál minden egyes fordulójában ezek az értékek eggyel nőnek, tehát mindig egyre beljebb kerül a kezdő pont.

- **vegX és vegY (végső koordináták):**
  - Ezek határozzák meg, hogy az adott spirális kör meddig tart. Az első körben a teljes külső széléig (vegX=4, vegY=4), majd minden spirális forduló után eggyel csökkennek ezek az értékek, így folyamatosan szűkítve a spirált.

Ezek a változók tehát a mozgási tartomány határait tárolják, így pontosan szabályozható, hogy meddig haladjon a LED a mátrixon (*4. ábra*). Segítenek abban, hogy a mozgás befelé spirálisan haladjon és csökkentésével korlátozni tudjuk a képernyő elérhető méretét.



4. ábra A micro:bit LED-mátrixának koordinátái és a ciklon mozgásának iránya
(Alap kép *forrás: Programozzunk micro:biteket 34.o*)

3. Fényerősség(0-255) beállítása (erosseg):

Ez a változó lesz a felelős azért, hogy a programkód a fényerősséget változtatni tudja, ezzel szemléltetve majd a légnyomás növekedését/csökkenését.

A felkapcsol függvény egyszerűbb lesz. Itt csökkenteni kell a fényerősséget 10-zel, felkapcsolni az aktuális LED-et a felkapcsol x:X y:Y fényerő erosseg paranccsal. Majd a listánk végére eltárolni az Y és X változók értékét az utvonal lista végére Y beszúrása és az utvonal lista végére X beszúrása blokkokkal. Fontos, hogy ebben a sorrendben tegyük ezt meg, mert a későbbiekben fordítva fogjuk kiolvasni az adatokat a listánkból és így megkapjuk majd az X;Y koordináta párokat. Ezután egy 100ms szünettel lassítjuk az animációt.

A ciklon animációja (*9. ábra*) az A gomb lenyomásával fog indulni ezért mindent az, amikor a(z) A gomb lenyomva blokkba építünk. Meghívjuk a ciklon_ini függvényt, majd a felkapcsol függvényt is, hogy az első LEDünk (0;0) is fel legyen kapcsolva.

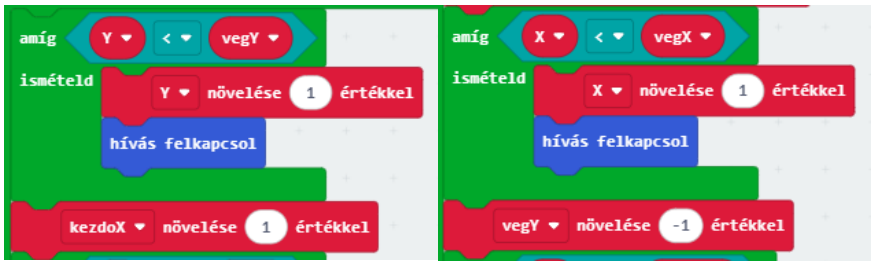
Ezután egy ciklusban addig kell lépkednünk még az aktuális pozíciónk nem a (2;2) koordináta, tehát amíg nem vagyunk a képernyő közepén. Ennek megvalósításához az amíg hamis ismételd ciklust fogjuk használni. A hamis helyére pedig megadjuk a feltételünket (*5. ábra*).

5. ábra A középpontot ellenőrző feltétel

Ezen cikluson belül 4 további ciklusunk lesz, amely a mozgásirányokat reprezentálja. Először lefele mozgunk, aztán jobbra, majd felfelé és végül balra. Minden mozgás után csökkenteni kell a rendelkezésre álló teret a képernyőn.

Lefelé addig tudunk haladni, amíg az Y kisebb lesz, mint a vegY. Ebben az esetben a Y értéket növeljük 1-gyel és a hívás felkapcsol blokkal lefuttatjuk a felkapcsol függvényben lévő kódunkat. Ezzel a képernyő bal oszlopát be is jártuk, ezért az X koordináta mozgástartományának elejét növelni kell, hogy a vízszintes mozgás ne indulhasson az első oszlopból. Ezt a kezdőX növelése 1-értékkel tesszük meg.

Jöjjön most a jobbra mozgás. Az alap struktúra ugyan az, mint a függőleges mozgásnál, csak az Y helyett az X tengelyhez köthető változókat kell használnunk és mivel az alsó sorban haladtunk, ezért a vegY növelése -1-gyel paranccsal tudjuk kivenni a tartományból az alsó sort, hogy a függőleges mozgás folyamán, már ne érintsük majd újra.



6. ábra A le- és jobbra mozgás kódja

A fel és balra mozgás esetében (*7. ábra*) az eltérések a következők lesznek. A feltételnél azt kell vizsgálnunk, hogy a felkapcsolás addig történjen, amíg az Y nagyobb, mint kezdőY és ekkor

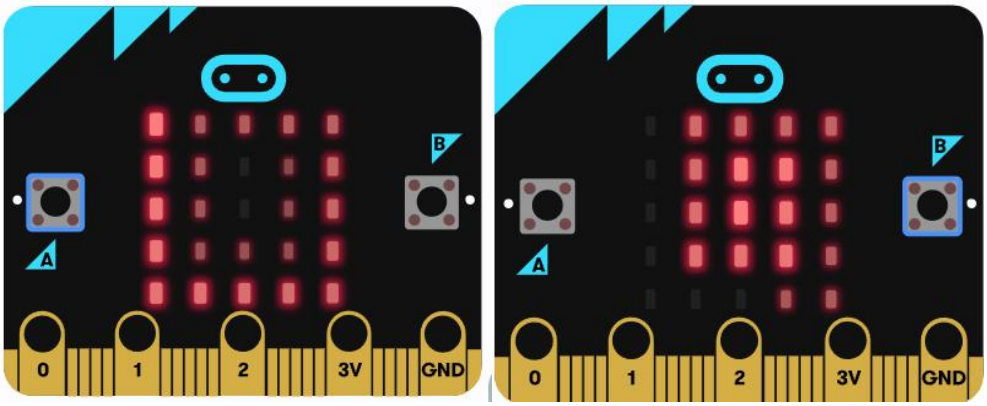
csökkentjük az Y értékét egyel, meghíjuk a felkapcsol függvényt és a kezdőY-t növeljük egyel, ezzel kihagyva az első sort a tartományból.

Balra mozgás esetében a feladatunk ugyan ez, csak írjuk át az X-hez tartozó változókat és a ciklus után a vegX értékét csökkentsük 1-gyel.

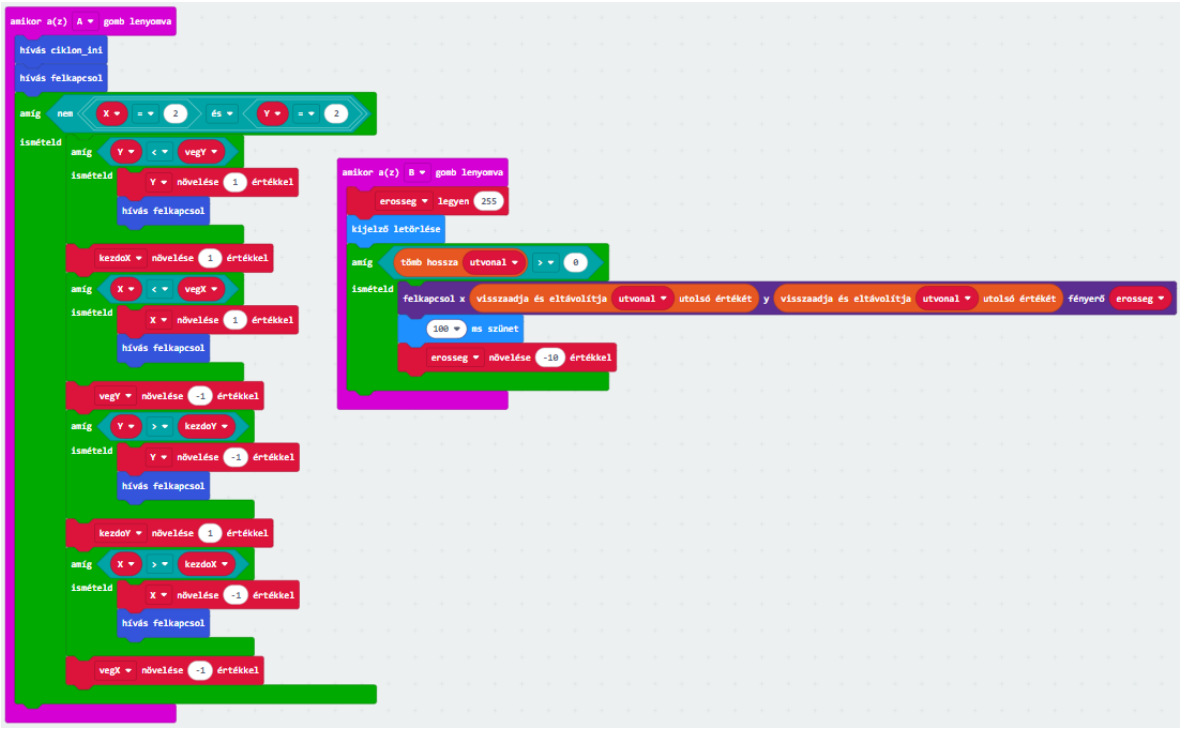
7. ábra A fel- és balra mozgás programkódja

A program nehezével meg is vagyunk, már csak az anticiklon útvonalát (*9. ábra*) kell kirajzolni a B gomb megnyomásával.

Amikor lenyomtuk a B gombot, akkor a fényerősséget visszaállítjuk 10-re, letöröljük a kijelzőt. Ezek után felkapcsoljuk a listában szereplő koordinátákat a megadott erősséggel a lista végéről véve az értékeket és törölve azokat a listából, a visszaadja és eltávolítja utvonal utolsó értékét paranccsal. Ezután 100ms szünetet alkalmazunk és csökkentjük az erosseg valtozot 10-zel. Ezt addig ismételjük, amíg a tömb hossza utvonal nagyobb mint 0.



8. ábra Az elkészült projekt működésének képe a szimulátorban(ciklon: bal, anticiklon: jobb)



9. ábra A ciklon (balra) és anticiklon (jobbra) animációját megvalósító kódrészlet

Továbbfejlesztési lehetőségek/további ötletek

A trópusi ciklonok animációjának megvalósításakor fontos meteorológiai sajátosság, hogy a ciklon közepén egy úgynevezett "szem" alakul ki – ez egy viszonylag nyugodt, csapadékmentes terület, amelyet a forgó felhőkarok vesznek körül. Ennek modellezéséhez a micro:bit LED-mátrixán a középső pozíciót (x:2, y:2) szándékosan ki kell hagynunk a megjelenítésből, hogy hűen tükrözze a természeti jelenség ezen jellemzőjét.

Ez egyszerűen megvalósítható a lekapcsol x:2 y:2 paranccsal, amelyet a fő spirálisan rajzoló ciklus után kell elhelyezni a kódban. Ez biztosítja, hogy még akkor se világítson fel a középső LED, ha azt a spirális algoritmus egyébként bekapcsolná. Így a kijelzőn egy körkörösen rajzolt, de középen üres, forgáshoz hasonló minta jelenik meg, amely jól szemlélteti a trópusi ciklon struktúráját és dinamikáját. Ez a megoldás nemcsak látványos, de kiváló lehetőséget biztosít a tanulók számára az időjárási rendszerek fizikai-geográfiai összefüggéseinek megértésére is.

Fogalomtáblázat

Fogalom, kifejezés	Magyarázat, jelentés
amíg hamis ismételd (ciklus)	Egy programozási szerkezet, amely egy adott feltétel hamissá válásáig ismétli az utasításokat.
eseményvezérlés	A program egy adott esemény (pl. gombnyomás) hatására hajt végre műveleteket.
feltétel	Olyan logikai kifejezés, amely meghatározza, hogy egy ciklus vagy elágazás mikor teljesüljön, vagy mikor érjen véget.
fényerősség (erosseg)	A LED fényerejének értéke (0-255 közötti skálán), amellyel vizuális jelzéseket adhatunk a programban.
függvény (eljárás)	Önálló, újrahasználható programrészlet, amely egy adott feladatot lát el.
kezdoX, kezdoY	A spirál aktuális ciklusának kezdőpontját meghatározó koordináták.
LED mátrix	A micro:bit 5×5-ös kijelzője, amely LED-ekből áll, és koordinátákkal címezhető.
lista/tömb	Adatok tárolására szolgáló eszköz, amelyben elemeket tárolhatunk adott sorrendben.
változó	Olyan tárolóhely, amelyben adatokat (számok, szövegek) tárolhatunk és változtathatunk meg a program során.
vegX, vegY	A spirál aktuális ciklusának végpontját meghatározó koordináták.
x, y	A LED pozíciójának koordinátái a micro:bit 5×5-ös LED-mátrixán (x: vízszintes, y: függőleges).

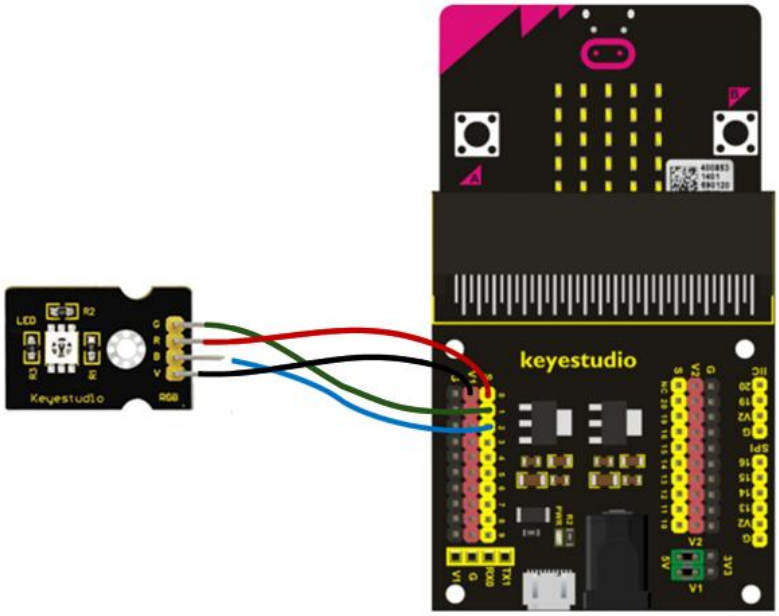
3.2 Gyűjtsd szelektíven

Fő tantárgy, ahol megjelenhet	Földrajz / Természetismeret
A foglalkozás helye	Hulladékkezelés
Szemléltető eszköz	Gépitanulás és AI vezérelt hulladék-felismerő
Felhasznált eszközök a modell elkészítéséhez	• 1 db micro:bit V2 • micro:bit Sensor Shield V2 • RGB LED modul • 4 csatlakozókábel • külső áramforrás a Shieldhez (elem/USB)
Programozáshoz köthető ismeretek	• Gépitanulás és AI modellek • elágazások • soros adatátvitel
A foglalkozás digitális kultúrához kapcsolódó céljai	• A diákok képesek legyenek saját gépitanulási modellek létrehozására • Tudjanak alapvető elágazásokat készíteni • Ismerjék meg a soros portok és adatátvitel fogalmát.

A foglalkozásról röviden

A modell játékos módon tanítja meg, hogyan kell helyesen szétválogatni a szemetet – például felismeri, hogy egy műanyag palack, egy fémdoboz vagy komposztálható hulladék melyik hulladékgyűjtőbe dobható. A különlegessége, hogy gépi tanulást (machine learning) és mesterséges intelligenciát (AI) is használ: a kamera segítségével „megtanulja”, hogyan néznek ki a különféle hulladékfajták, így egyre pontosabban tudja azokat felismerni. Ez nemcsak szórakoztató élményt nyújt a diákok számára, hanem bevezeti őket a jövő technológiáinak világába is.

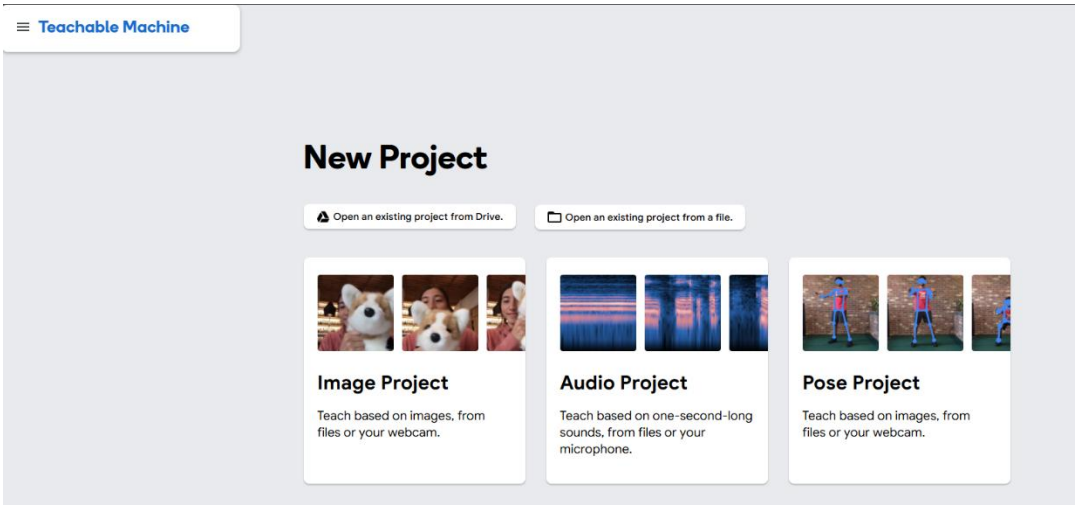
A szenzorok kapcsolása



10. ábra A hulladékfelismerő kapcsolási rajza

A program elkészítése

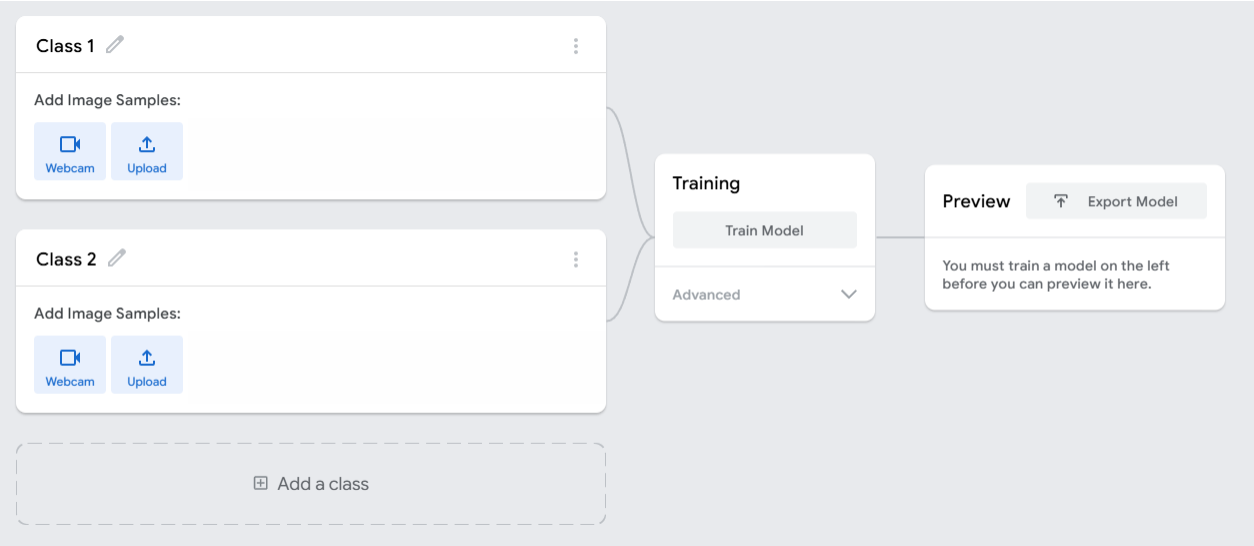
A szemléltetőeszközünk elkészítésének első lépése a képfelismerő modell betanítása. Ehhez a Google Teachable Machine eszközét fogjuk használni. Az oldalon a Get Started-ra kattintva fogjuk tudni kiválasztani, hogy milyen projektet szeretnénk létrehozni (*11. ábra*). Számunkra most az Image Project – Standard modell szükséges, hiszen azt szeretnénk, hogy az eszköz képek alapján ismerje fel a különböző hulladékokat.



11. ábra A projektválasztó felület a Teachable Machine oldalon

Ezek után jutunk a betanító felületre (*12. ábra*), alapból kettő üres osztállyal. Az osztályok feltöltése történhet fájlokból, de a kameránk segítségével is készíthetünk különböző képeket, amik

alapján megtanulja az algoritmus, hogy melyik osztály hogyan is néz ki. Csoportokat szabadon hozzáadhatunk, arra figyeljünk, hogy a név megadásánál ne használjunk ékezetes karaktereket és 12 karakternél rövidebb nevet adjunk meg, hogy ne legyen probléma a karakterkódolással és a túl hosszú üzenetekkel sem.



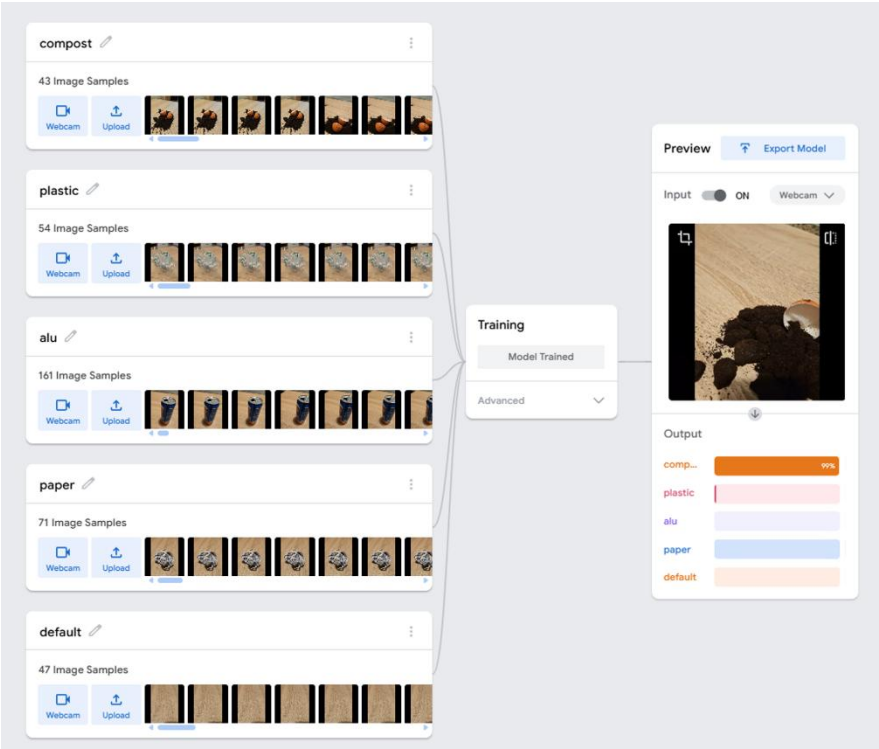
12. ábra A betanító felület

A foglalkozáshoz összesen 5 csoportot hozunk létre, melyek a compost, plastic, alu, paper, és a default neveket kapták. A default osztály létrehozása ajánlott, hogy amikor épp nincs hulladék a kameránk előtt, akkor ne hallucináljon[3] valamilyen hulladékot a képek alapján. A csoportok létrehozása és képekkel való feltöltés után nincs más teendőnk, mint rákattintani a Train Model gombra és megvárni, amíg az algoritmus megtanulja az osztályok jellemzőit a képek alapján.

A betanítás után, már látszik, hogy az asztalra kiöntött kávézaccot és tojáshéjat a betanításnak megfelelően 99%-os magabiztossággal beazonosította komposztként (*13. ábra*). Az osztályterembe érdemes vinni jellemző hulladéktípusokból 1-1 hulladékot és azt megtanítani a modellel. Minnél több képet készítünk, különböző szögekből, annál jobban fog működni a modellünk. A betanítás természetesen működhet fájlfeltöltéssel és az interneten elérhetők különböző képekből álló adatcsomagok, amelyek kifejezetten képfelismerő AI és gépitanulás algoritmusok számára készültek. Egyik ilyen oldal a Kaggle: Your Home for Data Science. Ilyenkor a modellünk jobban használható lesz

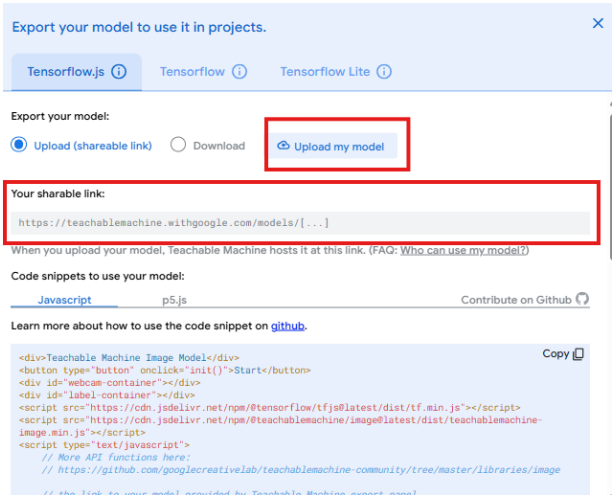
[3] Az AI-hallucináció (angolul: AI hallucination) azt jelenti, amikor egy mesterséges intelligencia valótlan vagy kitalált információt ad válaszként, miközben az teljesen meggyőzőnek vagy helyesnek tűnik.

a tényleges felismerések esetében és nem csak a pár, általunk prezentált képet fogja tudni felismerni.



13. ábra A betanított modell tesztelése komposztra

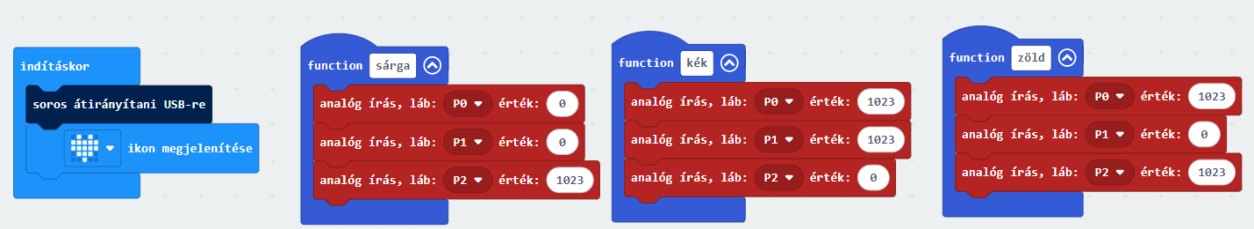
Ha elégedettek vagyunk a betanítás eredményével, akkor exportálhatjuk a modellt a jobb felső sarokban lévő gombbal. Ezek után lehetőségünk a projektünk mentésére és feltöltésére (*14. ábra*). Itt kattintsunk az Upload my model gombra, hogy megkapjuk az elérhetőségi linkjét, amire szükségünk lesz a későbbiekben.



14. ábra A modell megosztható linkjének generálása

Most, hogy a modellünket betanítottuk, nincs más hátra, mint összekapcsoljuk a micro:bitünket és a betanított AI algoritmusunkat. Ezt a MAKE: AI ROBOTS oldalon tudjuk megtenni. Előtte azonban szükséges, hogy elkészítsük a micro:bit működéséhez szükséges kódot.

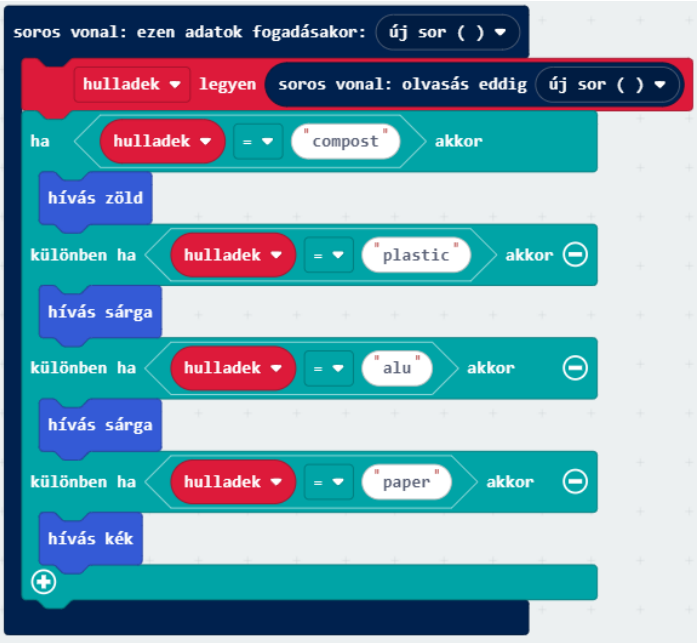
A programunk a számítógép kameráját használva beazonosítja az adott hulladékot és ennek megfelelő színre vált az eszközhöz kapcsolt RGB LED. Sárga szín: műanyag és fém, kék szín: papír, zöld szín: komposzt. Az eszköz bekapcsolásakor, hogy legyen visszajelzés a működésről, egy szív ikon kerül kirajzolásra, illetve a projekt működéséhez szükséges, hogy a soros kommunikációt[4] átirányítsuk az USB-re soros átirányítani USB-re blokkal (*15. ábra*), amely a Soros vonal blokkcsoportban található a Haladó csoportokat lenyitva. Az RGB LED-ünk egy analóg eszköz, ezért az értékei 0-1023 között változnak. 0 a teljes fényerő az adott színből. Érdemes a kód átláthatósága érdekében, külön eljárásokban/függvényekben tárolni a megjelenített színeket. Ezeket az analóg írás, láb: P0/P1/P2 érték: 0-1023 paranccsal tudjuk megtenni (*15. ábra*), amit a csatlakozó lábak blokkcsoportban találunk, szintén a Haladó fület lenyitva. Ezekkel a parancsokkal különböző erősségű analóg jeleket tudunk küldeni a csatlakozólábakon keresztül. *Figyeljünk arra, hogy a kód csak abban az esetben adja vissza a kívánt színeket, ha a bekötés is a leírásnak megfelelő, ellenkező esetben nem biztos, hogy a P0 fog megfelelni a piros értéknek.*



15. ábra A felismerő program kezdeti beállításainak kódja

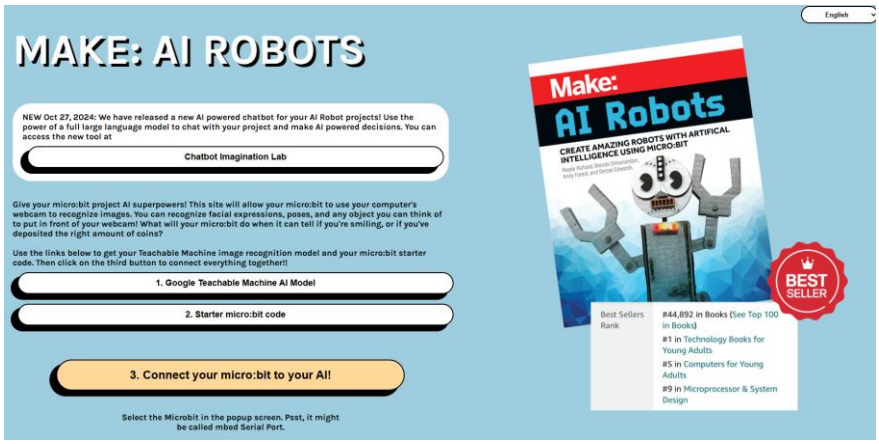
[4] A soros kommunikáció egy olyan kommunikációs mód, amivel a micro:bit és más eszközök(pl.: számítógép) egyszerűen beszélgetni tudnak egymással egy vezetéken keresztül. Esetünkben az USB-n keresztül.

Ezek után már csak azt kell figyelnünk, hogy amikor adatot kapunk(soros vonal: ezen adatok fogadásakor: új sor ( )) meghatározzuk, hogy mi történjen (*16. ábra*). A micro:bit az AI modelltől a felismert osztály nevét fogja megkapni és ennek megfelelően meg tudjuk vizsgálni egy elágazással, hogy a kapott üzenet, amit eltárolunk egy változóban (hulladék legyen soros vonal: olvasás eddig új sor ( )), mit tartalmaz és ez alapján milyen színnel kel a LED-et felkapcsolni (hívás zöld/sárga/kék).



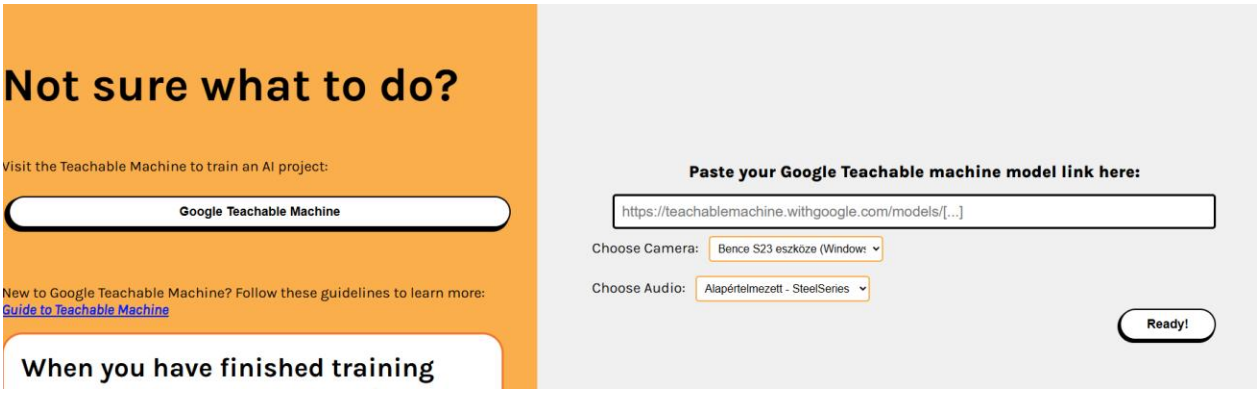
16. ábra A felismerő program legfőbb kódrésze

A kódot mentsük is le a micro:bitre és térjünk vissza a korábban már említett MAKE: AI ROBOTS oldalra. Itt válasszuk a 3. Connect your micro:bit to your AI! gombot (*17. ábra)*. És párosítsuk a micro:bitünket a felugró ablakban (általában az *mbed Serial Port*).



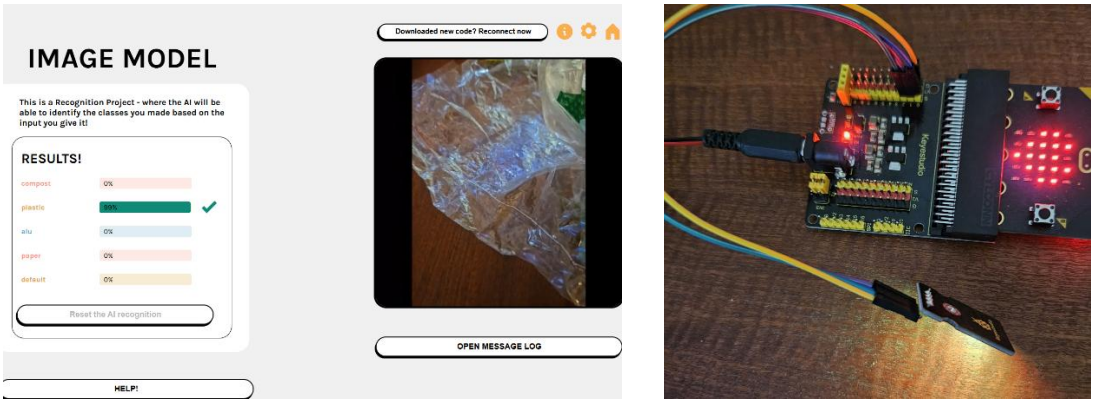
17. ábra Az összekötést megvalósító honlap kezdőlapja

Utána illesszük be az AI modellünk linkjét (*18. ábra*):



18. ábra Az AI modell csatolása a weboldalhoz

Ezután nincs más dolgunk, mint használni a modellt és különböző hulladékokat felismertetni vele. A példánkon látszik (*19. ábra*), hogy a betanított műanyag zacskót sikeresen felismerte a programunk, aminek eredményeképpen felkapcsolta a micro:bithez kapcsolt LED-et sárga fénnyel.



19. ábra A példaprojekt működés közben

Továbbfejlesztési lehetőségek/további ötletek

A modell nagyobb adathalmazzal való betanítása sokkal jobb eredményt ad. A program esetlegesen bővíthető kiírásokkal, amely kiírja a konkrét hulladék típusát is. Eltérő dobozokhoz akár készíthetünk különböző hulladéktípusokra betanított eszközöket, amelyek azt figyelik, hogy ténylegesen megfelelő hulladék került-e bele a dobozba, amelyet egy mosolygós vagy szomorú arccal reprezentál.

Fogalomtáblázat

Fogalom / Kifejezés	Magyarázat
elágazás	Feltételes programutasítás: a program az adott bemenet alapján különböző utasításokat hajt végre.
függvény / eljárás	Önállóan elnevezett kódrészlet, amely újra felhasználható, pl. egy adott szín bekapcsolásához.
gépitanulás (machine learning)	Olyan mesterséges intelligencia-módszer, amely példák alapján „megtanulja” felismerni a mintákat.
lista / osztály (Teachable Machine)	Az AI által megtanulandó kategóriák (pl. „plastic”, „paper”), amelyekbe képeket gyűjtünk.
mbed Serial Port	A számítógép által használt kommunikációs csatorna a micro:bit és az AI rendszer között.
mesterséges intelligencia (AI)	Számítógépes rendszerek képessége olyan feladatok végzésére, amelyeket általában emberi intelligencia igényel.
P0, P1, P2 (csatlakozólábak)	A micro:bit fizikai kivezetései, amelyeken keresztül más eszközöket (pl. LED-et) vezérelhetünk.
RGB LED	Olyan LED, amely piros, zöld és kék fényt külön-külön vezérelve képes különböző színeket megjeleníteni.
soros adatátvitel / soros kommunikáció	Az adatok továbbítása a micro:bit és a számítógép között egy adatcsatornán, pl. USB-n keresztül.
USB-re irányított soros kommunikáció	A micro:bit adatforgalmának áthelyezése USB-re, hogy a számítógép fogadhassa az AI modell válaszait.
változó	Olyan tárolóhely, amelyben adatokat (pl. felismert hulladéktípus neve) tárolhatunk és változtathatunk meg a program során.

3.3 Légköri adatok naplózása

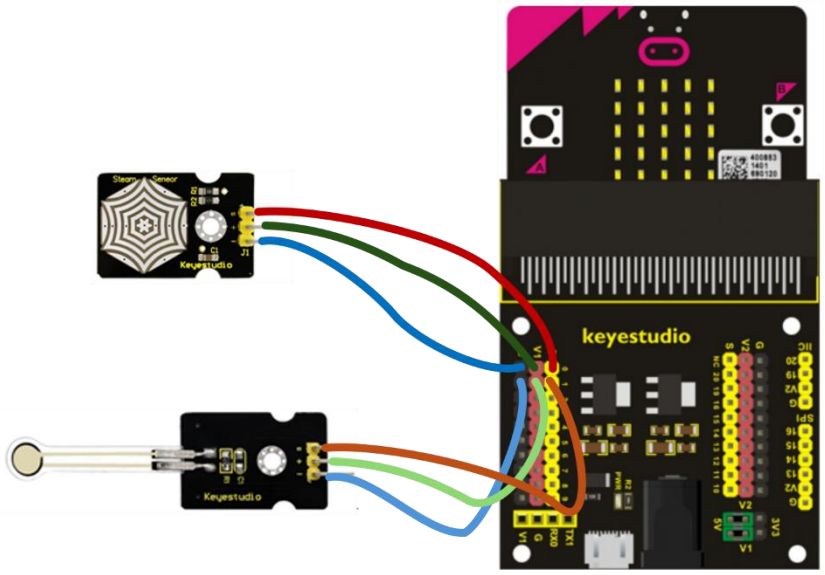
Fő tantárgy, ahol megjelenhet	Fizika / Földrajz / Természetismeret
A foglalkozás helye	Víz és levegő a környezetünkben / A légkör
Szemléltető eszköz	Légköri adatokat mérő és naplózó modelleszköz
Felhasznált eszközök a modell elkészítéséhez	• 1 db micro:bit V2 • micro:bit Sensor Shield V2 • pára- és nyomásmérő szenzor • 6 csatlakozókábel
Programozáshoz köthető ismeretek	• szenzorok használata • adatok mentése és feldolgozása • algoritmikus gondolkodásfejlesztés (maximumkiválasztás)
A foglalkozás digitális kultúrához kapcsolódó céljai	• A diákok képesek legyenek szenzorokkal a külvilágból érkező adatok fogadására • Tudják az adatokat a micro:biten tárolni és később azt feldolgozni

A foglalkozásról röviden

A foglalkozás keretében a tanulók megvizsgálják, hogyan változik a levegő páratartalma és nyomása a környezetükben, miközben érzékelőkkel mérik az adatokat. A gyűjtött értékeket a micro:bit naplózza, így a diákok megtapasztalják, hogyan lehet valós idejű környezeti adatokat rögzíteni és rendszerezni. A program gombnyomásra képes a maximum értékek elmentésére is.

A begyűjtött adatokat később akár táblázatkezelő programban is feldolgozhatják, így a tevékenység erősíti a digitális adatelemzési kompetenciákat, emellett tudatosítja, hogy a környezeti megfigyelések adatvezérelt megközelítése hogyan segíti a természeti jelenségek pontosabb megértését.

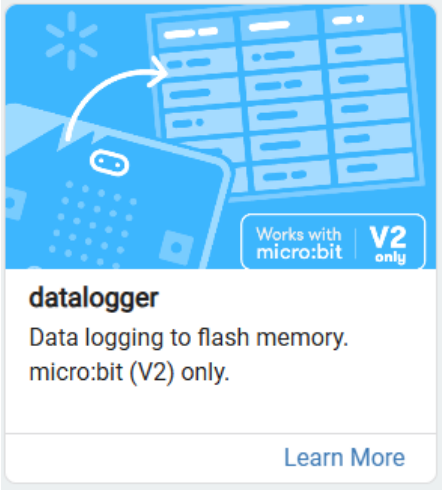
A szenzorok kapcsolása



20. ábra Az adatmérő modell kapcsolási rajza

A program elkészítése

A programunk elkészítéséhez elengedhetetlen lesz a datalogger[5] blokkcsoport (*21. ábra*), amelyet a kiterjesztésekre kattintva tudunk hozzáadni a szerkesztőfelületünkhöz.

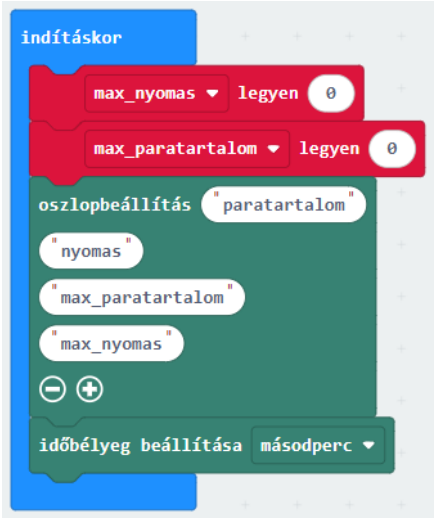


21. ábra A datalogger blokkcsoport borítóképe

A blokkok hozzáadása után az első feladatunk a programkódunkban, a program inicializálása (*22. ábra*). Ahhoz, hogy mindkét értékünk (páratartalom és nyomás) maximumértékét el tudjuk tárolni, ezért indításkor létrehozunk két változót max_paratartalom és max_nyomas néven, amelyeknek

[5] További példaprojektek és részletes áttekintő a blokkcsoportról elérhető a Data logging overview | micro:bit oldalon, angol nyelven.

értékét 0-ra állítjuk. Ezután az adattáblánkat kell beállítanunk az oszlopbeállítás („ ”) paranccsal. Itt a pluszjelre kattintva további 3 oszlopot hozzáadunk és elnevezzük az oszlopokat sorba (paratartalom, nyomas, max_paratartalom, max_nyomas). Ékezetes karakterek esetében a naplózás nem fog megvalósulni, ezért kerüljük az oszlopnevek esetén az ékezetes betűket. Ezek után pedig beállítjuk, hogy a táblázatunkban időbélyegek is megjelenjenek, amit az időbélyeg beállítása másodperc paranccsal tehetünk meg. *A másodperc opció az időbélyeg formátumát adja itt meg, nem pedig a mérési intervallumot.*



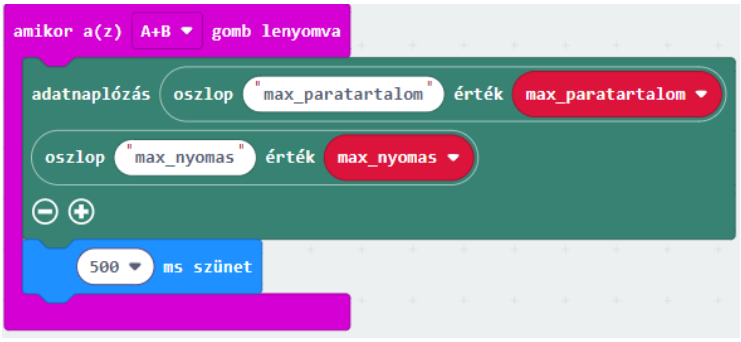
22. ábra A mérőeszköz inicializálásának kódja

A program fő gerincét (*23. ábra*) egy speciális ciklus fogja adni, ami adott időközönként (esetünkben másodpercenként) fut le és hajtja végre a benne lévő kódokat. A beállított idő lesz a mérés gyakorisága. A cikluson belül tehát először is be kell olvasnunk az adatokat két változó(paratartalom, nyomas) segítségével. Ezeket a változókat beállítjuk az analóg olvasás, láb: P0 és P1 parancsokkal a már korábban is használt csatlakozó lábak blokkcsoportból.

Az adatok beolvasása után, következik a maximumértékek vizsgálata. Amennyiben az aktuálisan mért érték nagyobb, mint a maximum érték, akkor az új maximum érték legyen a beolvasott értékünk. Ezt mindkét érték esetében meg kell tennünk. A maximumvizsgálat után az adatnaplózás (oszlop „paratartalom” érték paratartalom) (oszlop „nyomas” érték nyomas) blokkal naplózzuk az aktuálisan mért eredményeket. A maximumértékek tárolását gombnyomásra valósítjuk meg, amelyre mindjárt kitérünk, de előtte létrehozunk egy visszajelző ikont, amely 100ms ideig látható a képernyőn, majd letöröljük a képernyőnket a kijelző törlése blokkal. Ennek célja, hogy vizuálisan is láthassuk, amikor a mérés megtörténik.

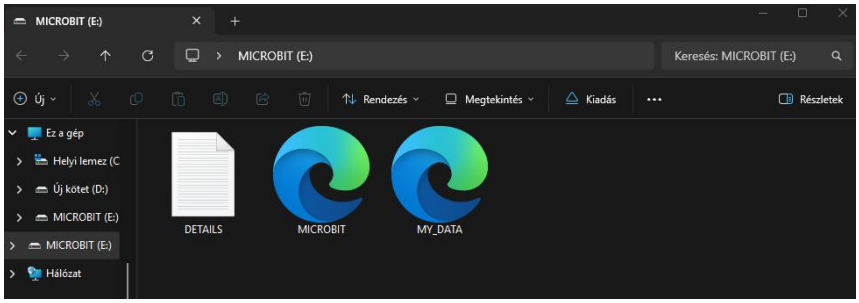
23. ábra Az adatgyűjtő programunk fő programkódja

A maximumértékek tárolása értelmetlen és tárhelypazarló lenne minden időpillanatban, ezért ezeket az értékeket csak az A+B gomb megnyomásakor tároljuk el (*24. ábra*). A várakozás szerepe, hogy gátolja a többszöri tárolást egy lenyomásra.



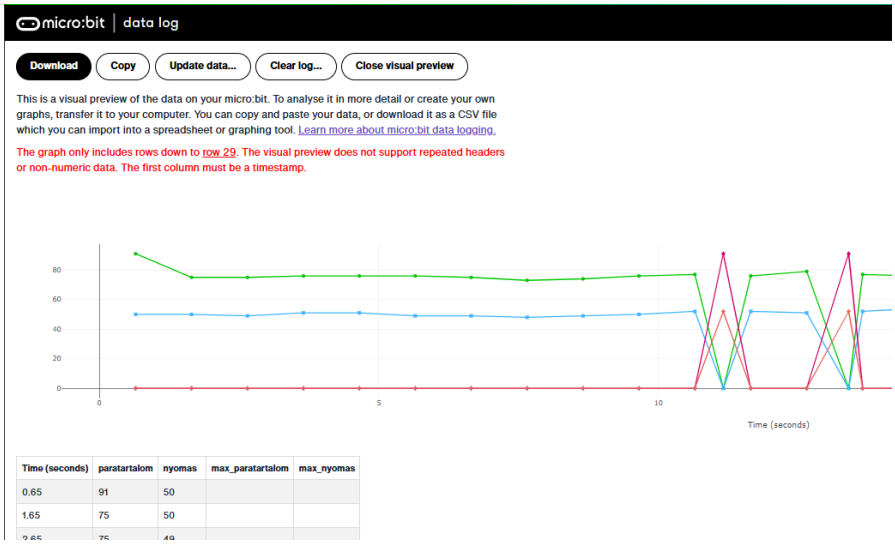
24. ábra A maximumértékek tárolásának programkódja

A tárolt adatokhoz a micro:bit csatlakoztatásával tudunk hozzáférni. A tárolt adatok MY_DATA.HTM fájlban találhatók, amit a micro:bit meghajtóján érünk el (*25. ábra*).



25. ábra A naplófájl elérési helye

A fájlt megnyitva egy elég intuitív felülettel találkozhatunk (*26. ábra*), ahol lehetőségünk van letölteni[6] és lemásolni az adatokat, törölni a naplót és egy korlátozott (29 érték megjelenítése) adatvizualizációs eszköz[7] is a rendelkezésünkre áll. Az adatok frissítése a micro:bit újracsatlakoztatásával történik és ekkor a fájlt is újra meg kell nyitni, vagy frissíteni a böngészőben. A fájl



26. ábra A micro:bit data log felülete

Értelemszerűen a maximumértékek tárolása miatt az adatokat tisztázni szükséges a mentés után, hogy a tényleges vizualizációt megkapjuk.

A micro:bit belső memóriája másodpercenként rögzített egész számokból álló, négyoszlopos adatnaplózás esetén körülbelül 6–7 óra mérési adat tárolására elegendő. Ha azonban a mintavételezési gyakoriságot percenkénti rögzítésre, a maximumértékek naplózását pedig óránkénti gyakoriságra állítjuk, a memória kapacitása akár egy teljes hónapnyi adat tárolását is lehetővé teszi.

[6] A letöltés .csv formátumú fájlba menti az adatokat. A fájl megnyitásához segédlet elérhető a Microsoft honlapján: CSV-fájl megnyitása vagy importálása - A Microsoft ügyfélszolgálata

[7] Offline módban nem érhető el.

Továbbfejlesztési lehetőségek/további ötletek

A micro:bit nem képes önmagában rendszeridő tárolására, márpedig a méréseknél főleg, ha azok meteorológiai célból születnek egyáltalán nem lényegtelen a tényleges időbélyegek használata. Ebből kifolyólag az egyik továbbfejlesztési lehetőség, hogy az időt a program indításakor a gombok segítségével beállítjuk és onnantól kezdve a saját időbélyegünket használjuk egy külön oszlopban, vagy csak az idő eltárolásával később már az adatfeldolgozás során könnyen kiszámíthatjuk az időbélyegek alapján a tényleges mérési időpontokat.

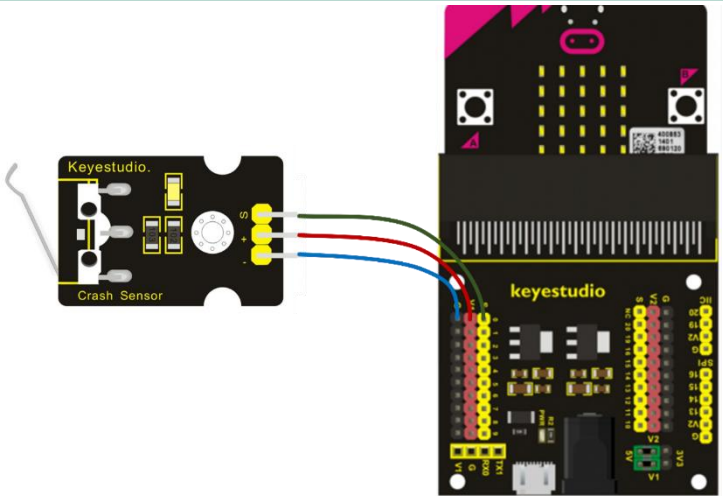
Fogalomtáblázat

Fogalom / Kifejezés	Magyarázat
adatnaplózás (datalogging)	A mért értékek automatikus, időbélyeggel ellátott tárolása egy adatfájlba.
analóg olvasás, láb: P0/P1	A micro:bit bemenetén található lábakon keresztül érkező szenzorértékek beolvasása.
belső memória	A micro:bit tárolókapacitása, amely korlátozott mennyiségű adat elmentését teszi lehetővé.
ciklus	Ismétlődő programutasításokat végrehajtó szerkezet, például adott időközönként futtatott mérés.
időbélyeg	Az adatnaplózás során egy adott mért értékhez kapcsolt időpont.
kijelző törlése	A kijelző LED-jeinek lekapcsolása.
maximumkiválasztás	Algoritmikus művelet: megállapítja, hogy egy új érték nagyobb-e, mint az eddigi legnagyobb.
mintavételezési gyakoriság	Megadja, milyen időközönként történik adatgyűjtés (pl. másodpercenként, percenként).
MY_DATA.HTM fájl	A micro:bit belső memóriájában létrejövő fájl, amelyben a naplózott adatokat találjuk.
program inicializálása	A program indításkori előkészítése: változók beállítása, kijelző törlése stb.
táblázatkezelő program	Olyan szoftver (pl. Excel), amely lehetővé teszi az adatok feldolgozását és elemzését.
változó	Olyan tárolóhely, amelyben adatokat (mért értékeket, maximumokat) tárolhatunk és változtathatunk meg a program során.
várakozás (delay)	Rövid késleltetés a program futásában, amely segít az események szabályos végrehajtásában.

3.4 Morse jeladója

Fő tantárgy, ahol megjelenhet	Fizika
A foglalkozás helye	Mágneses mező. Az áram mágneses mezője / Készítsünk elektromágnest!
Szemléltető eszköz	Légköri adatokat mérő és naplózó modelleszköz
Felhasznált eszközök a modell elkészítéséhez	• 1 db micro:bit V2 • micro:bit Sensor Shield V2 • ütközésszenzor • 3 csatlakozókábel
Programozáshoz köthető ismeretek	• karakterműveletek • listaműveletek • műveletek futási idővel
A foglalkozás digitális kultúrához kapcsolódó céljai	• A diákok képesek legyenek adatokat listákban tárolni és műveleteket végrehajtani velük • Szöveges adatok összefűzése és felhasználása, adattípusok megismerése

A szenzorok kapcsolása



27. ábra A Morse-távíró modell kapcsolási rajza

A foglalkozásról röviden

Ez a foglalkozás a Morse-kód működésének megértésére és modellezésére épül. A tanulók egy ütközésérzékelőt használnak "távíróként": rövid és hosszú megnyomásokkal pontokat és vonásokat küldenek, amelyeket a micro:bit leolvassa és szöveggé alakít. A program idő alapján különbözteti meg, hogy az adott jel pont (rövid) vagy vonás (hosszú), és ezeket egy előre beállított táblázat alapján betűkké alakítja.

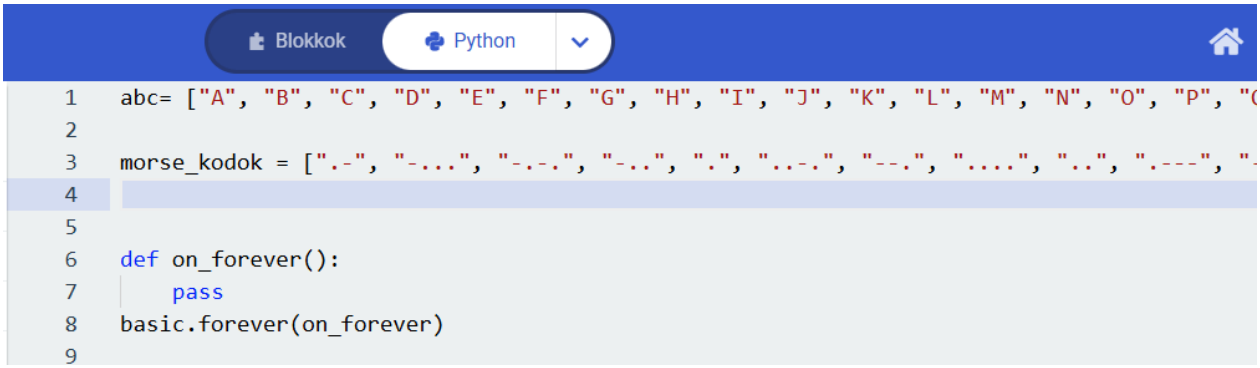
A foglalkozás során a diákok megtapasztalják, hogyan lehet az elektromágneses jelenségeket (pl. Morse-jelek továbbítása) egyszerű szenzoros eszközökkel modellezni. A rendszer működése közben a tanulók visszajelzést kapnak fény- és hanghatásokkal, és a dekódolt szavak a micro:bit kijelzőjén is megjelennek. A tevékenység ráirányítja a figyelmet az adatközlés fizikai alapjaira és az elektromágnesesség gyakorlati alkalmazásaira.

A program elkészítése

A program elkészítése előtt érdemes előre megalkotnunk a két listát, amely az angol ABC, valamint a hozzá tartozó Morse ABC kódjait tartalmazza. Ehhez másoljuk ki az alábbi két listát és a micro:bit felületén váltsunk át a python nyelvre és a kód elejére illesszük be őket (*28. ábra*). Ezzel a művelettel az indításkor blokkal készen is vagyunk. Érdemes ezt a programot lementeni és a diákok számára már előzetesen biztosítani.

abc= ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]

morse_kodok = [".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.."]



28. ábra A micro:bit programozási felülete python környezetben

A program megvalósításához két eljárásra is szükségünk lesz. Az egyik esetben az eljárás egy szöveges paramétert is tartalmazni fog. Ezért amikor létrehozzuk a karakter függvényünket, kattintsunk a szöveg elemre miután elneveztük a függvényünket (*29. ábra*). A másik függvényünk pedig a betuiras lesz, amelyet paraméterek nélkül hozunk létre.

Nézzük első körben a karakter függvény felépítését (*30. ábra*). A karakterfüggvény fogja kijelezni, hogy éppen melyik Morse karaktert nyomtuk meg és a paraméterben kapott érték alapján fog

nekünk visszajelzést adni, illetve itt fogjuk a Morse karaktereket összefűzni, hiszen egy normál betű az több Morse karakterből áll.

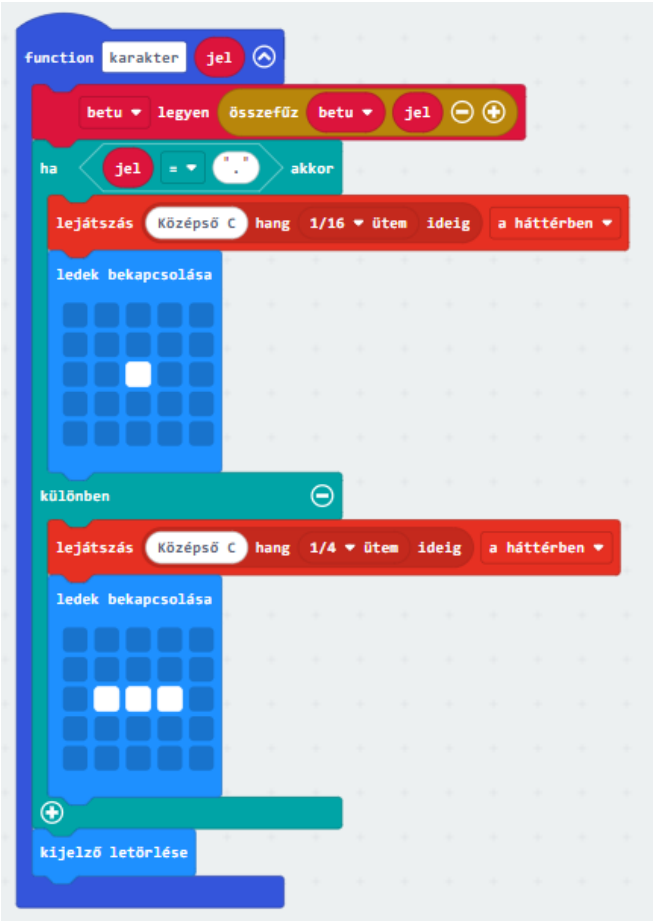
29. ábra Paraméteres függvény létrehozása

Ezek segítségéül hozzunk létre egy jel- és egy betu változót, előbbit lokálisan fogjuk használni a függvényen belül, ezért illesszük be a függvény paraméteréhez. A függvényben az első dolgunk, hogy a meglévő karakterekhez hozzáfűzzük az éppen lenyomott Morse jelet, ami meghatározza nekünk a betűt majd. Ezt a betu legyen összefűz (betu) (jel) paranccsal tudjuk megtenni.

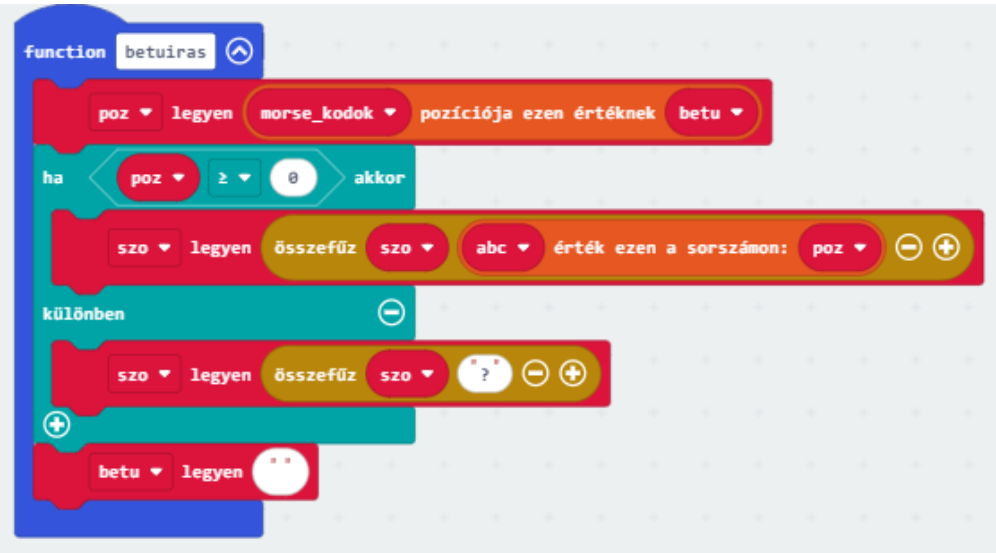
Ezek után meg kell vizsgálni, hogy a kapott értékünk az pont vagy vonás. Ha a jel "." akkor a lejátszás középső C hang 1/16 ütem ideig [a háttérben] paranccsal lejátszunk egy csipogást és kirajzolunk a képernyőre egy pontot a ledek bekapcsolásával. Ellenkező esetben, tehát különben pedig 1/4 ütemig történik a lejátszás le a hangot és egy vonalat rajzolunk ki. Végül pedig a kijelző törlése blokkal eltűntetjük az aktuális visszajelzést a kijelzőről.

A betuiras függvényben (*31. ábra*) meg kell néznünk, hogy a korábban létrehozott betu változóban tárolt Morse jelek validak és ténylegesen lefordíthatók-e. Ezt úgy tudjuk megtenni, hogy először eltároljuk, hogy a betu változó hányadik helyen szerepel a morse_kodok listában a poz legyen a morse_kodok pozíciója ezen értéknek (betu) blokkal. Ezek után, ha megnézzük, hogy a poz nagyobb vagy egyenlő, mint nulla. Ha ennek eredménye nem egy szám, akkor azt jelenti, hogy nincs a listában az elem, tehát nem valós kódunk van. Ha ez a feltétel igaz, akkor a program megkeresi, hogy a kapott Morse-kód hányadik helyen szerepel a Morse-kódokat tartalmazó listában. Mivel a Morse-kódok és az angol ábécé betűi ugyanabban a sorrendben szerepelnek, ez alapján ki tudja választani a megfelelő betűt az ábécélistából. Ezután ezt a betűt hozzáfűzi az eddig összegyűjtött szöveghez, amely így folyamatosan bővül a lefordított karakterekkel. Ezt a műveletet a szo legyen (összefűz (szo) ((abc érték ezen a sorszámon: (poz)). Amennyiben a feltétel nem teljesül, úgy a szo

soron következő karaktere legyen egy „?”-jel. A függvény végén pedig állítsuk üresre a betu változót, amely szöveges változók esetében üres idézőjelekkel oldható meg



30. ábra A karakter függvény programkódja



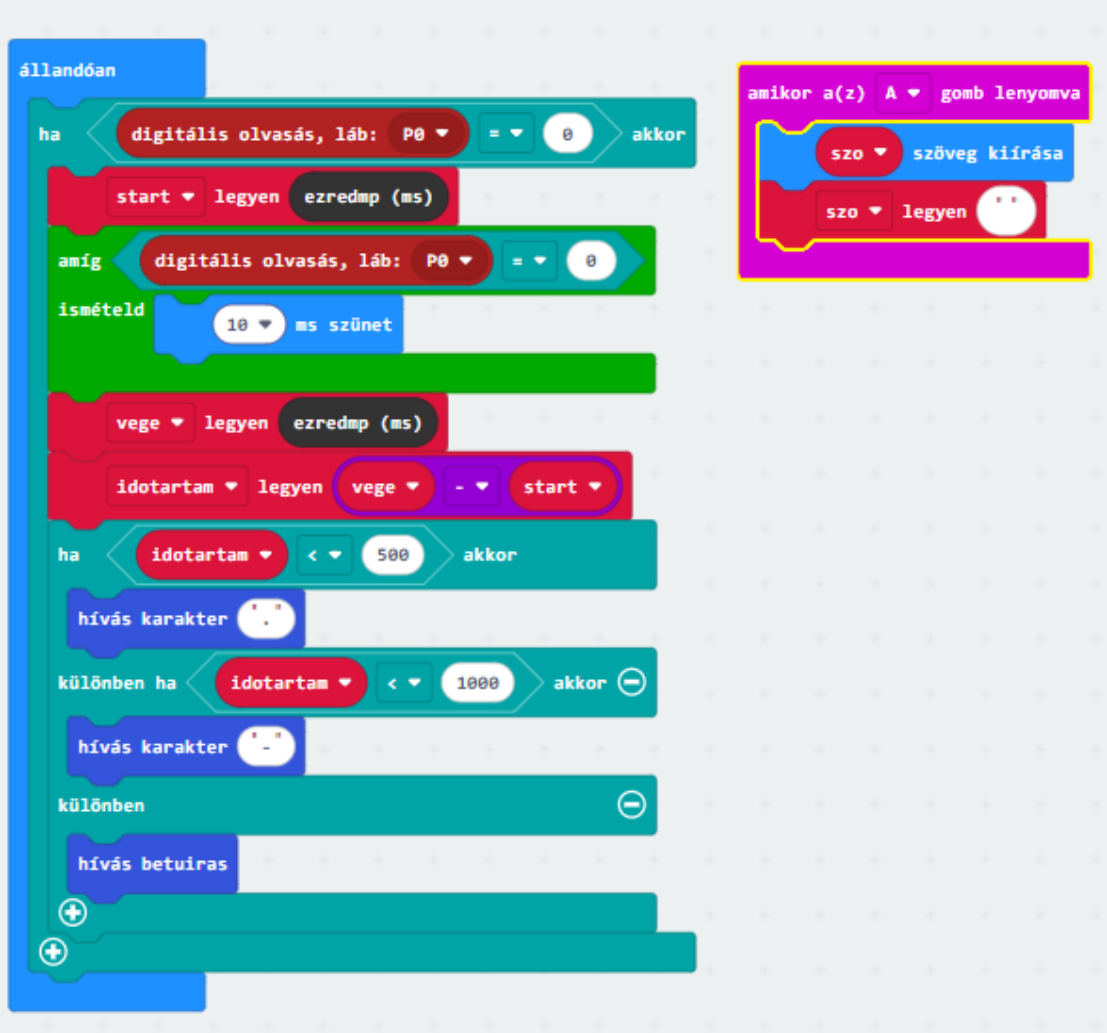
31. ábra A betuiras függvény programkódja

A program elkészítésének nehezén túl vagyunk. A következő rész az állandóan blokk elkészítése lesz *(32. ábra*). Itt meg kell vizsgálnunk, hogy az ütközőszenzorunk le van-e nyomva, ezt a ha digitális olvasás, láb P0 egyenlő 0 feltétellel tudjuk megtenni. Ha ez igaz, akkor a start változó legyen egyenlő ezredmp (ms)-el. Ezzel a futási időt tudjuk megvizsgálni és gyakorlatilag a lenyomás időpillanatát mentettük el, hogy később meg tudjuk nézni, hogy mennyi ideig volt lenyomva a gomb. Az ezredmp (ms) blokk a Haladó fület lenyitva, a Vezérlés blokkcsoportban található meg.

Az eltárolás után az amíg digitális olvasás: P0 egyenlő 0 ismételd 10ms szünet blokkokkal várakoztatjuk a programot, amíg le van nyomva a szenzorunk. Ezután a starthoz hasonlóan a vege változóban is eltároljuk az aktuális futási időt. Ezután kiszámoljuk, hogy mennyi ideig volt lenyomva a gomb az idotartam változóban úgy, hogy a vege változóból kivonjuk a start változót.

Ha az idotartam kisebb, mint 500 akkor a hívás karakter ”.” blokkal hívjuk a karakter függvényt, különben, ha idotartam kisebb, mint 1000 akkor pedig hívás karakter ”-” blokkot használjuk, különben pedig a betuiras függvényt hívjuk meg. Ez utóbbi azt jelenti, hogy ha 1 másodpercnél tovább tartjuk lenyomva a szenzort, akkor lezárjuk az aktuális betű kódolását. A karakter függvény paraméterébe beírt pont és vonás pedig átadásra kerül és a függvény ezt fogja eltárolni a korábban létrehozott jel változóban.

A szavaink kiírásához még szükség lesz egy apró kis blokkra. Az, amikor az A gomb lenyomva blokk segítségével írjuk ki a szo szöveget, majd a szo legyen ”” paranccsal állítsuk üresre a szo változót.



32. ábra A Morse-jeladó fő programkódja

Továbbfejlesztési lehetőségek/további ötletek

A program továbbfejleszthető úgy, hogy ne csak egy micro:bit tudjon Morse-jeleket feldolgozni lokálisan, hanem több micro:bit is részt vehessen a kommunikációban. Ehhez a micro:bit beépített rádiókapcsolatát lehet felhasználni, amely lehetővé teszi az üzenetküldést egymás között vezeték nélkül. Minden micro:bit egyedi azonosítót kapna, és amikor egy tanuló Morse-jelet küld, az üzenet tartalmazná annak a micro:bitnek az azonosítóját, akinek szánja. A fogadó micro:bit csak akkor dolgozná fel az üzenetet, ha az valóban neki szól. Így a rendszer nemcsak két eszköz közötti kommunikációra lenne alkalmas, hanem egy egész osztálytermi hálózaton belül is lehetővé tenné az üzenetküldést, mintha egy egyszerű Morse-alapú chat működne több résztvevővel.

Fogalomtáblázat

Fogalom / Kifejezés	Magyarázat
azonosító (ID)	Minden micro:bit egyedi azonosítóval rendelkezik, amivel megkülönböztethető és címzhető az üzenetküldés során.
digitális olvasás, láb: P0	Beolvasás a micro:bit egyik digitális bemenetéről, pl. gombnyomás érzékeléséhez. Értéke 0 vagy 1 lehet.
ezredmp (ms) blokk	Időmérésre szolgáló blokk, amely ezredmásodperces pontosságot ad vissza a futás idejéből.
feltételvizsgálat (if/else)	A program különböző utasításokat hajt végre attól függően, hogy egy logikai kifejezés igaz vagy hamis.
függvény (eljárás)	Előre definiált utasításblokk, amit név alapján többször is végrehajthatunk (pl. karakter(), betuiras()).
indexelés	Annak meghatározása, hogy egy adott adat hányadik helyen van egy listában (pl. Morse-kód alapján betű kiválasztása).
ismételd amíg **(while)**	Addig ismétlődő utasításblokk, amíg egy adott feltétel igaz (pl. gomb nyomva tartása alatt).
karakterműveletek	Szövegeken végzett műveletek, például karakterek összehasonlítása, fűzése vagy kiolvasása.
listaműveletek	Több adat tárolása egy sorozatban (pl. tömb/lista) és azok keresése, indexelése, összehasonlítása.
összefűzés	Két vagy több karakterből, vagy szövegből egy új szöveg (string) létrehozása.
paraméteres függvény	Olyan függvény, amelynek működését egy bemeneti adat (pl. Morse karakter) befolyásolja.
üres szöveg ("" idéző-jelek között)	Üres szöveges érték beállítása, a változó „törlése” programozásban.
változó	Olyan tárolóhely, amelyben adatokat (pl. jel, betu, szo, idotartam) tárolhatunk és változtathatunk meg a program során.

3.5 Újraélesztés

Fő tantárgy, ahol megjelenhet	Biológia
A foglalkozás helye	Az ember szervezete és működése
Szemléltető eszköz	Újraélesztés ritmikusságának gyakorlására szolgáló eszköz
Felhasznált eszközök a modell elkészítéséhez	• 1 db micro:bit V2 • (Vízpumpa és gumicső) • Relé modul
Programozáshoz köthető ismeretek	• feltételkezelés • műveletek változókkal • futási idő kezelés
A foglalkozás digitális kultúrához kapcsolódó céljai	• A diákok képesek legyenek a program számára fontos és összetett feltételek meghatározására és alkalmazására • Változókban tárolt adatokkal műveletek végrehajtása.

A foglalkozásról röviden

A micro:bit segítségével készült modelleszköz célja, hogy a diákok játékos formában gyakorolhassák az újraélesztéshez szükséges megfelelő ritmust. A program folyamatosan figyeli, milyen gyakorisággal nyomják le a tanulók a micro:bit egyik gombját, és összehasonlítja azt az optimális, percenként körülbelül 100–120 nyomásból álló ritmussal, amely a szakszerű mellkaskompresszió alapja. Ha a tanulók tempója megfelelő, a kijelzőn egy mosolygó arc (smiley) jelenik meg, visszajelzésként a helyes kivitelezésről. Ez az eszköz nemcsak segít a ritmusérzék fejlesztésében, hanem felhívja a figyelmet az újraélesztés jelentőségére is, miközben digitális eszközhasználatra és problémamegoldásra is ösztönöz.

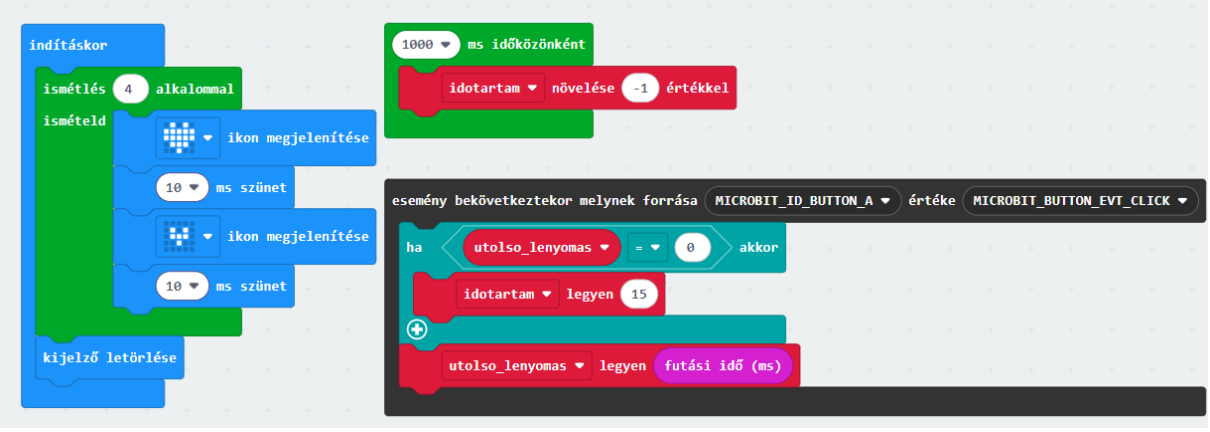
A program elkészítése

A program indításakor egy egyszerű szívdobogás animációt fogunk megjeleníteni (*33. ábra*). Ezt egy ismétléssel és benne a nagy szív és kis szív ikon megjelenítésével oldunk meg. Közöttük 10ms szünetet alkalmazunk. Az animáció után letöröljük a kijelzőt. A program fő része előtt hozzunk létre még két blokkot. Az egyik egy speciális ciklus lesz, amely 1000ms időközönként csökkenti az idotartam változót 1-gyel. Erre azért van szükség, hogy a későbbiekben az idotartam változó értékét vizsgálva meg tudjuk adni és figyelni az „újraélesztés” időtartamát.

A másik blokkunk egy eseményvezérelt blokk, amelyhez szükség lesz a Haladó blokkokon belül a Vezérlés blokkcsoportra(*33. ábra*). Innen válasszuk az esemény bekövetkeztekor melynek forrása

() értéke () blokkot. Az első paramétert hagyhatjuk az alapból beállított értéken, hiszen így azt figyeli, hogy történik-e valami a micro:bit A gombjával. A második értéket állítsuk át arra, hogy akkor hajtsa végre az ebben a blokkban lévő utasításokat, ha az A gomb lenyomásra került, vagyis klikkeltünk vele. Ezt akkor tudjuk figyelni, ha a MICROBIT_BUTTON_EVT_CLICK opciót választjuk. Azért van szükség ennek a blokknak a használatára, hogy csak a lenyomás és klikkelés pillanatát tudjuk vizsgálni és ne legyen gond azzal, ha sokáig lenyomva tartjuk a gombot. Utóbbit például a bevitelnél lévő, amikor az A gomb lenyomása blokk nem tudja megkülönböztetni.

A most létrehozott eseményvezérelt blokkunkon belül meg kell nézni, hogy történt-e már gomblenyomás egyáltalán, elkezdődött-e már az újraélesztés. A „kompressziók" idejének tárolásához létrehoztunk egy utolso_lenyomas változót. Az indításkor akár be is állíthatjuk az értékét nullára, de alapból is nulla lesz, amíg nem adunk meg neki mást. Amennyiben, ha utolso_lenyomás egyenlő 0-val akkor az időtartamot beállítjuk a kívánt értékre az idotartam legyen () blokkal. A példában 15 másodpercig fogja a program a kompressziók ritmusát vizsgálni. Ezt csak az első kompresszió esetén kell megtenni. A lenyomások esetén pedig az utolso_lenyomas legyen futási idő (ms)[8]. Ezzel tudni fogjuk, hogy két lenyomás között mennyi idő telt el, amihez majd viszonyítani tudjuk, hogy a 100-120 lenyomás megtörténne-e egy perc alatt. *A program kiegészíthető lenne annak figyelésével, hogy a diákok nem kezdték-e túl korán a gomb lenyomását. Ebben az esetben egy logikai változó használata javasolt, amelyet az indításkor blokk végén igazra állítunk és a gombnyomások naplózásánál feltételként adjuk meg, hogy csak abban az esetben naplózunk bármit, ha a változónk igaz értékkel rendelkezik.*

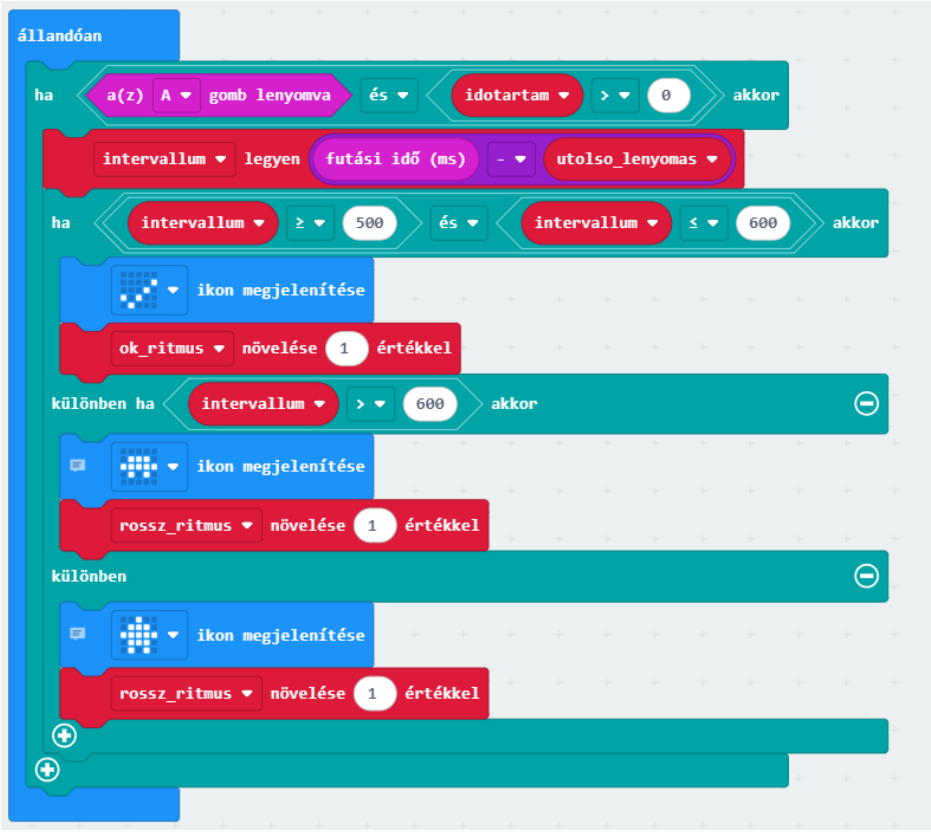


33. ábra A program inicializálásának és a két speciális blokk programkódja

[8] Az előző anyagban a futási időt a Vezérlés blokkok használatával mértük, itt most egy másik példát alkalmaztunk, a Bevitel blokkcsoportból

A következő lépés az állandóan blokkok létrehozása, ugyanis az átláthatóság kedvéért két, párhuzamosan futó állandóan blokk fogja alkotni a programunk fennmaradó részét. Ha a diákok számára könnyebb, akkor ezeket egymás után egy állandóan blokkban is meg lehet valósítani.

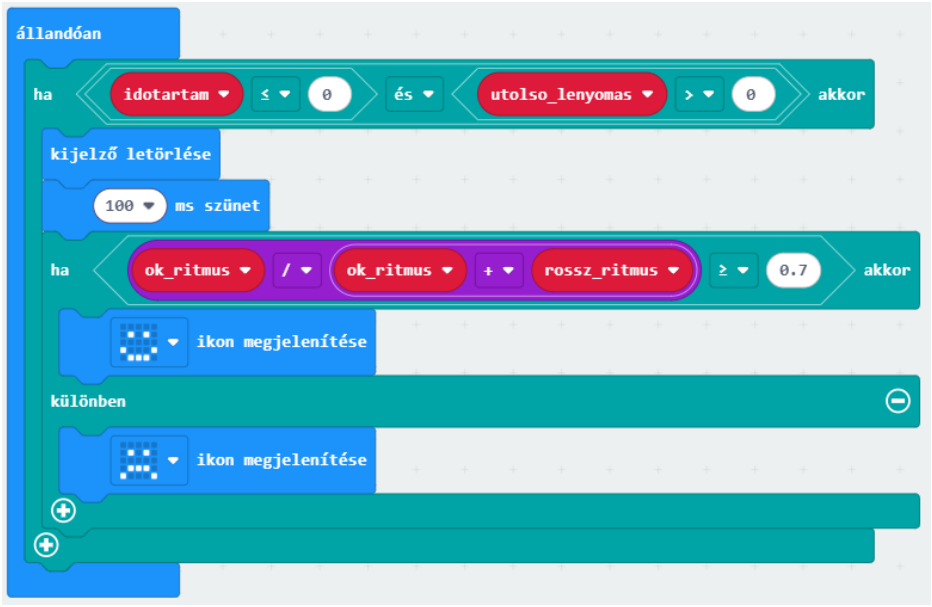
Az első állandóan blokkunkban (*34. ábra*) egy fő feltételünk lesz, amely megnézi, hogy az A gomb le van-e nyomva és az idotartam változó nagyobb-e, mint nulla. Ezt a ha az A gomb lenyomva és idotartam nagyobb, mint 0 akkor blokkal tudjuk megoldani. Utána, a ha blokkban, az intervallum legyen futási idő(ms) – utolso_lenyomas blokkal lemérjük és eltároljuk a két gombnyomás közötti időt. Ezt az eltárolt időt meg kell vizsgálni, hogy megfelel az újraélesztés elfogadott ritmusának. Ehhez percenként 100-120 nyomás kell, azaz két kompresszió között 500-600 ms telhet el. Amennyiben, ha az intervallum nagyobb egyenlő mint 500 és kisebb egyenlő mint 600, akkor pipa ikon jelzi és az ok_ritmus változót növeljük 1-gyel. Ebben a változóban tároljuk el a megfelelő ritmusú lenyomásokat. Egyéb esetben, különben ha az intervallum nagyobb mint 600, akkor egy ikon jelezze, hogy lassúak vagyunk és a rossz_ritmus változó értékét növeljük 1-gyel, különben pedig egy másik ikonnal jelezzük, hogy túl gyorsak voltunk és ismételten növeljük a rossz_ritmus változót.



34. ábra Az első állandóan blokk programkódja

A második állandóan (*35. ábra*) blokk fogja vizsgálni, hogy mi történjen akkor, ha az idotartam kisebb egyenlő mint 0 és az utolso_lenyomas nagyobb, mint nulla. Ilyen esetben a kijelzőt letöröljük és egy 100ms várakozást rakunk bele a programunkba, hogy ne azonnal váltson át a sebességjelző ikon a végső visszajelzésre.

Ezután megnézzük, hogy a megfelelő ritmusok aránya eléri-e a 70%-os sikerességet, a ha ok_ritmus / ok_ritmus + rossz ritmus nagyobb egyenlő mint 0.7 blokkal. Ha ez igaz, akkor egy mosolygós smiley jelzi az újraélesztés sikerességét, ellenkező esetben pedig egy szomorú smiley jelzi azt, hogy nem sikerült.



35. ábra A második állandóan blokk programkódja

Továbbfejlesztési lehetőségek/további ötletek

A program továbbfejleszthető a fentebb említett feltételvizsgálaton felül azzal, hogy egy pumpát rácsatlakoztatunk a micro:bitre, például a Sensor Shield V2 segítségével. Amennyiben sikerül az újraélesztés, a pumpa, a szívet szimulálva elindul és tudunk vele folyadékot pumpáltatni, szemléltetve a vérkeringés beindítását. Ehhez egy relé modulra is szükségünk van, amely sikeres újraélesztésnél bekapcsolja a vízpumpánkat.

Fogalomtáblázat

Kifejezés	Magyarázat
arányszámítás	A helyes ritmusarány kiszámítása: ok_ritmus / (ok_ritmus + rossz_ritmus)
ciklus (állandóan blokk)	Folyamatosan ismétlődő kódrészlet, amely valós idejű ellenőrzést végez
eseményvezérlés	A program csak akkor hajt végre műveletet, ha adott esemény (pl. gombnyomás) történik
feltételvizsgálat (if/else)	A program különböző utasításokat hajt végre a feltételek függvényében
futási idő	A program indulása óta eltelt idő ezredmásodpercben (ms)
időalapú mérés	Gombnyomások közti idő mérése, a ritmus ellenőrzéséhez
intervallum számítása	Két gombnyomás közti idő kiszámítása
MICROBIT_BUTTON_EVT_CLICK	Kifejezetten a "klikkelés" eseményt figyelő eseménytípus, nem reagál hosszan tartott nyomásra
változó	Olyan tárolóhely, amelyben adatokat (pl. idotartam, intervallum) tárolhatunk és változtathatunk meg a program során.
visszajelző ikonok	A kijelzőn mosolygó, szomorú arc vagy pipa, X ikon jelenik meg a teljesítmény alapján
logikai változó	Igen/nem érték tárolása (pl.: elindult-e már az újraélesztés)

3.6 Vírusfertőzés

Fő tantárgy, ahol megjelenhet	Biológia
A foglalkozás helye	Az élet eredete és szerveződése – A vírusok
Szemléltető eszköz	Vírusfertőzés valószínűségének szimulálása
Felhasznált eszközök a modell elkészítéséhez	• 2 db micro:bit V2
Programozáshoz köthető ismeretek	• feltételkezelés • függvénykészítés • véletlenszámok gyakorlati használata
A foglalkozás digitális kultúrához kapcsolódó céljai	• A diákok képesek legyenek a program számára fontos és összetett feltételek meghatározására és alkalmazására • Változókban tárolt adatokkal műveletek végrehajtása. • Véletlenszámok előállítása és felhasználása esélyszámításnál

A foglalkozásról röviden

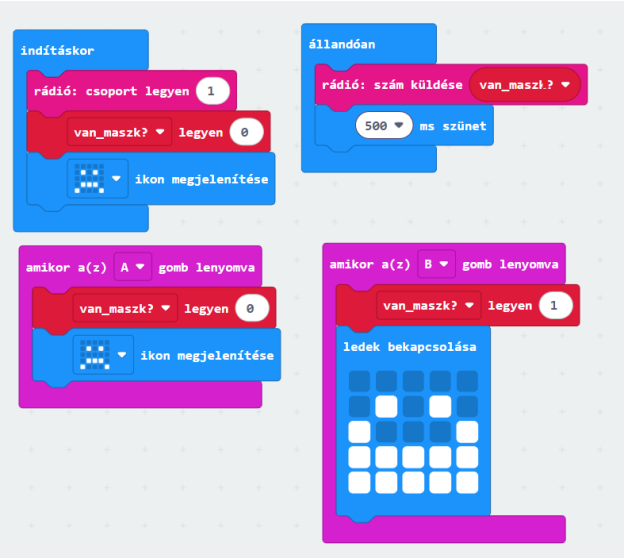
A foglalkozás célja, hogy a tanulók párokban dolgozva, két micro:bit eszköz segítségével modellezzék egy vírusfertőzés terjedésének folyamatát. Az egyik eszköz egy fertőzött személyt szimulál, amely rádiós jelet sugároz a környezetébe, jelezve a potenciális fertőzést. A másik eszköz érzékeli ezeket a jeleket, és annak alapján, hogy visel-e „maszkot” (azaz védelmet jelző állapotban van-e), valamint mekkora a távolság a két micro:bit között (rádiójel erőssége alapján), kiszámítja a fertőzés valószínűségét. A tanulók a gombok segítségével állíthatják be a maszkviselés állapotát, és valós időben tapasztalhatják meg, hogy a védekezés és a távolságtartás hogyan csökkenti a megfertőződés kockázatát. A tevékenység során nemcsak biológiai ismereteik mélyülnek, hanem algoritmikus gondolkodásuk és logikai következtetéseik is fejlődnek.

A program elkészítése

Első lépésként kezdjük a vírushordozó micro:bit programozásával (*36. ábra*). A feladatnak ez a könnyebbik része. Indításkor szükséges a rádiócsoportok beállítása. Itt figyeljünk arra, hogy minden pár saját rádiócsoportot használjon, hogy elkerüljük azt, hogy egyik csoport bezavar a másik csoport eszközeinek működésébe. A rádiócsoportot a rádió: csoport legyen 1 blokkal állíthatjuk be. Ezután létrehozunk egy változót, amit 0-ra állítunk a van_masz? legyen 0 paranccsal. Ez a változó fogja tárolni, hogy az eszközünk viseli-e a képzeletbeli maszkját. Logika feltételt is

használhatnánk, de a rádión keresztül számot és szöveget tudunk csak továbbítani, ezért az Igaz Hamis-t az 1-0 számpárral váltjuk ki. Ezután az eszközön egy szomorú smiley jelenik meg ezzel szemléltetve, hogy az eszköz beteg.

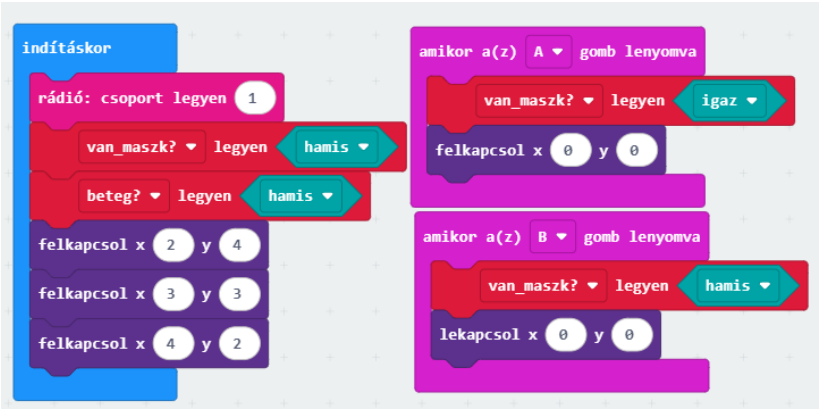
Az állandóan blokkban annyi dolgunk lesz, hogy a rádió: szám küldése van_maszk? paranccsal elküldjük a rádión a változónk értékét, hogy viseli-e a maszkot vagy sem. Ezt 500ms szünettel lassítjuk. Két eseményvezérlést is létre fogunk hozni. Amikor az A gomb lenyomva van, akkor a van_maszk? legyen 0 és jelenjen meg a szomorú smiley. Ezzel tudják a diákok majd levenni a maszkot. Az amikor B gomb lenyomva blokkal pedig a maszk felvételét oldjuk meg, tehát a van_maszk? legyen 1 parancsot kell használnunk. Ezután a kijelzőn egy maszkos smiley jelenik meg a ledek bekapcsolása blokk segítségével. Ezzel el is készültünk a fertőző micro:bit program-kódjával.



36. ábra A vírushordozó micro:bit programkódja

Második lépésként hozzuk létre a még vírusmentes micro:bit programkódját. Indításkor (*37. ábra*) itt is szükséges a rádiócsoport beállítása, valamint itt is használjuk a van_maszk? változót és a kezdeti értékét hamisra állítjuk. Ezen felül lesz egy beteg? változó, aminek kezdeti értéke szintén hamis lesz. Az indításkor blokkban kapcsoljuk fel a felkapcsol x: y: parancssal a 2;4, 3;3, 4;2 ledeket. Később ennek a fertőzöttség visszajelzésében lesz szerepe. Az A és B gomb megnyomásával (*37. ábra*) szintén a maszk viselését tudjuk beállítani. A kijelző korlátai miatt azonban itt csak a bal felső leddel fogjuk jelezni, hogy van-e vagy nincs maszk az eszközön, mert szükséges lesz valahogy majd azt is jelezni, hogy a fertőzés fennáll. Tehát amikor az A gomb lenyomva, a

van_maszk? legyen igaz, majd a felkapcsol x0 y0 paranccsal kapcsoljuk is fel a bal felső ledet. A B gomb lenyomására a van_maszk? változót állítsuk hamisra és a ledet kapcsoljuk le.



37. ábra A nem fertőzött micro:bit inicializációjának kódja

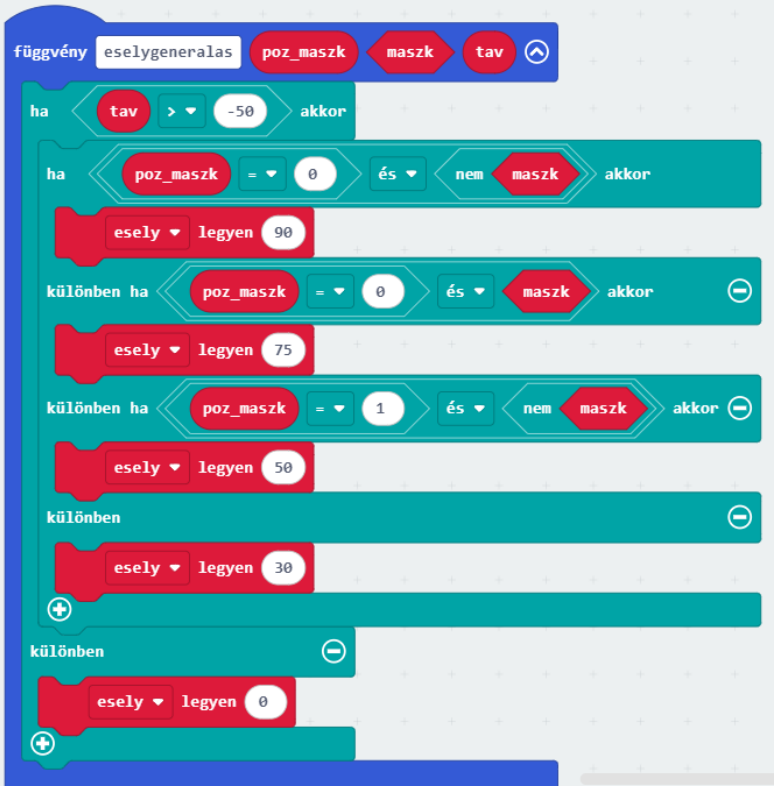
Következő lépésben határozzuk meg mi történjen akkor, amikor rádiós jelet kap a micro:bit. Az eszköz képes figyelni ezt a rádiós adat vételekor receivedNumber (*40. ábra*) blokkal és végrehajtja a benne lévő kódokat abban az esetben, ha kap rádión keresztül egy számot. A blokkon belül egy függvény fogja megvizsgálni azt, hogy mekkora az esélye a megbetegedésnek. A megbetegedés esélyére program a következő logikát alkalmazza:

- Ha **a fertőzött nem visel maszkot**, és **a fogadó sem**, akkor az esély 90%.
- Ha **a fertőzött nem visel maszkot**, de **a fogadó igen**, akkor 75%.
- Ha **a fertőzött maszkot visel**, és **a fogadó nem**, akkor 50%.
- Ha **mindketten viselnek maszkot**, akkor 30%.
- Ha egymástól messze vannak (nem lépnek kontaktba), 0%.

A függvény létrehozásakor (*38. ábra*) adjunk meg neki legalább 2 szám és egy logikai paramétert, amelyek helyére a rádión kapott számot (a másik micro:bit visel-e maszkot), a mi micro:bitünk maszkviselési állapotát és a fogadott csomag jelerősségét fogjuk behelyettesíteni.

38. ábra A kívánt függvény struktúra

A példában a függvény eselygeneralas néven fut. A tartalma (*39. ábra*) pedig a következőképpen épül fel a függvény eselygeneralas poz_maszk <maszk> tav blokkon belül[9]. Az első változó a rádión kapott értéket fogja jelölni, a második, hogy az eszközünkön van-e maszk, míg a tav a rádiójel erősségét[10]. Ha a tav nagyobb mint -50, akkor az azt jelenti, hogy a micro:bitek közel vannak egymáshoz, ilyenkor kell megvizsgálnunk a fentebb említett lehetőségeket, hogy melyik eszközökön van maszk. Tehát ha az első feltételünk igaz, akkor ha poz_maszk = 0 és nem maszk akkor esely legyen 90. Különben ha poz_maszk = 0 és maszk akkor esely legyen 75, különben ha poz_maszk = 1 és nem maszk, akkor esely legyen 50, különben pedig esely legyen 30. Amennyiben a micro:bitek messze vannak egymástól az esely legyen 0.

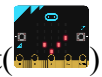
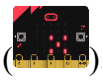


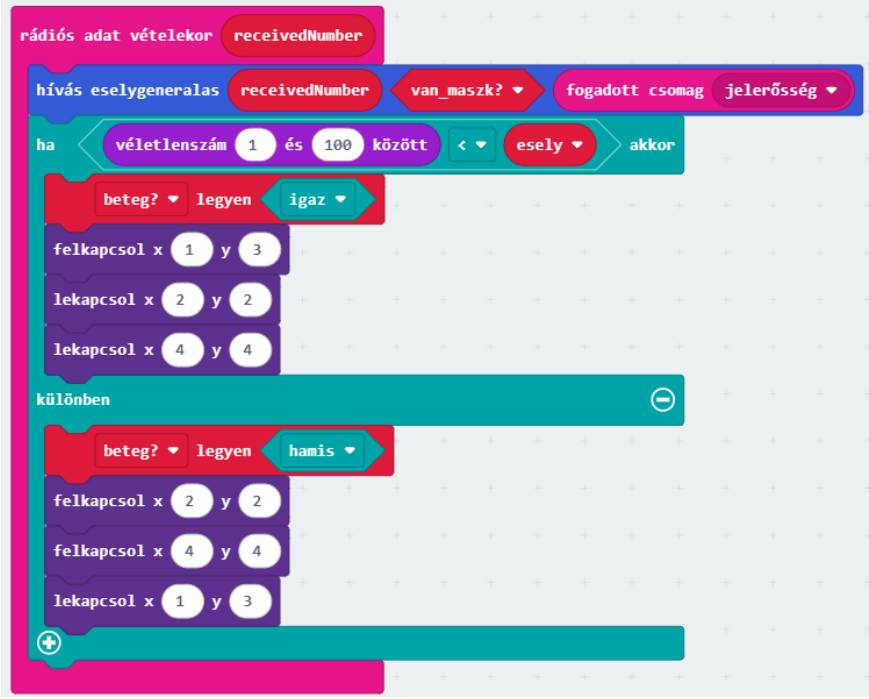
39. ábra Az esélygeneráló függvény felépítése

Ha ezzel megvagyunk, akkor térjünk vissza a rádiós adat vételekor blokkunkhoz (*40. ábra*). Itt hívjuk meg a függvényünket a hívás eselygeneralas receivedNumber <van_maszk?> fogadott csomag jelerősség blokkal. A paraméterek átadásra kerülnek a függvénynek, amiket a függvény fel fog tudni dolgozni és a korábban létrehozott függvény blokkban tud velük dolgozni. Ezután

[9] A változók itt a helyi, lokális változók, amiket a függvény tud használni saját magán belül és ezeknek kívülről értékeket adhatunk.
[10] Jelerősség: az érték -128-tól -28-ig terjed (-128 gyenge, -28 pedig erős jelet jelent).

generálunk egy véletlenszámot és eldöntjük, hogy megtörtént-e a megbetegedés. Ezt a ha véletlenszám 1 és 99 között kisebb, mint esely feltétellel tesszük meg. Ha igaz, akkor a beteg? legyen igaz, különben a beteg? legyen hamis.

Ami még hátra van, hogy jelezzük, hogy az eszköz a fertőzött() vagy sem (). Ehhez már felkapcsoltunk korábban 3 ledet, amely mindkét esetben fel kell, hogy kapcsolva legyen, így megspóroljuk azt, hogy fel kelljen kapcsolni őket feleslegesen. Ezek alapján a feltételünk igaz ágát kiegészítjük azzal, hogy felkapcsoljuk az 1;3 ledet és lekapcsoljuk a 2;2 és 4;4 ledeket, míg a hamis ágon pedig a 2;2 és 4;4 ledeket kell felkapcsolni az 1;3-at pedig le. *Megjegyzés: A rádiójel fogadásáig a kijelzőn csak az indításkor felkapcsolt ledek fognak világítani.*



40. ábra A vírusfertőzöttség vizsgálata rádiós adat vételekor

Továbbfejlesztési lehetőségek/további ötletek

A jelenlegi modell páros helyzetet szimulál, de könnyedén kiterjeszthető egész osztályra is. Ebben az esetben minden tanuló micro:bittel vesz részt a szimulációban, és csak egyetlen micro:bit van kezdetben „fertőzött” állapotban. A többi eszköz rádión keresztül érzékeli a környezetéből érkező jeleket, és minden egyes fogadott jel után – a maszkviselés és a távolság figyelembevételével – kiszámolja, hogy bekövetkezik-e a fertőzés. Ha igen, a tanuló eszköze maga is fertőzővé válik, és onnantól kezdve ő is sugározni kezdi a jelet mások felé.

Ez a változat egy valós járványterjedési modellt valósít meg, ahol láncszerűen terjed a fertőzés, és a tanulók dinamikusan tapasztalják meg, milyen gyorsan tud egy vírus végigsöpörni egy közösségen. Az eszközök kijelzőin vizuális visszajelzés is megjelenhet (pl. ikonváltás, villogás), amiből következtetni lehet a fertőzöttek arányára. A foglalkozás végén a diákok közösen értékelhetik, hogyan változott volna a járvány lefutása, ha mindenki viselt volna maszkot, vagy ha nagyobb távolságot tartanak egymástól.

Ez a változat lehetőséget nyújt összetettebb hálózati szimulációkra, és bevezetőként szolgálhat az epidemiológiai modellezés és a valószínűségi rendszerek fogalmába is.

Fogalomtáblázat

Kifejezés	Magyarázat
felkapcsol x: y: / lekapcsol x: y:	Egy adott LED fel- vagy lekapcsolása a kijelzőn, koordináták alapján (pl. x:2 y:4).
feltételvizsgálat (if/else)	A program különböző utasításokat hajt végre a feltételek függvényében.
függvény (eljárás)	Előre definiált utasításblokk, amit név alapján többször is végrehajthatunk (pl. eselygeneralas()).
jelerősség	A rádión érkező jel „erőssége”, amely a két micro:bit közötti távolságra utal. Az érték -128-tól -28-ig terjed (-128 gyenge, -28 pedig erős jelet jelent).
paraméteres függvény	Olyan függvény, amelynek működését bemeneti adatok (pl. maszk állapota, távolság) befolyásolják.
rádiós adat vételekor	Eseményvezérelt blokk, amely akkor fut le, ha a micro:bit rádión keresztül adatot (számot) kap.
rádiós adatátvitel / rádiókapcsolat	A micro:bit eszközök közötti vezeték nélküli kommunikáció, amely lehetővé teszi adatok (pl. számok, szövegek) továbbítását.
változó	Olyan tárolóhely, amelyben adatokat (pl. maszkviselés, fertőzöttség) tárolhatunk és változtathatunk meg a program során.
véletlenszám	Olyan szám, amelyet a program véletlenszerűen választ meg egy megadott tartományból. Esélyszámításnál a fertőzés bekövetkeztét szimulálja.

3.7 Hőmérséklet mérése valós időben (Python alapú foglalkozás)

Fő tantárgy, ahol megjelenhet	Földrajz, Fizika
A foglalkozás helye	A légkör földrajza, Hőtani folyamatok
Szemléltető eszköz	Digitális, valós idejű hőmérőeszköz készítése, időjárásállomások szimulálása
Felhasznált eszközök a modell elkészítéséhez	• 1 db micro:bit V2
Programozáshoz köthető ismeretek	• adatvizualizáció • soros kommunikáció és hardver interfész
A foglalkozás digitális kultúrához kapcsolódó céljai	• A diákok képesek legyenek valós időben egy hardver által mért adatot feldolgozni • Tudják a feldolgozott adatokat megfelelően, vizuálisan ábrázolni

A foglalkozásról röviden

A foglalkozás során a tanulók egy valós idejű hőmérsékletmérő rendszert készítenek micro:bit és Python segítségével. A micro:bit beépített hőmérő szenzora folyamatosan méri a környezeti hőmérsékletet, és az adatokat soros porton keresztül továbbítja a számítógépre. A számítógépen egy Python program fogadja és feldolgozza ezeket az értékeket, majd valós időben megjeleníti azokat egy grafikonon. A tanulók így megtapasztalják, hogyan lehet egy fizikai szenzor adatát digitálisan megjeleníteni, megismerik az adatok beolvasásának és vizualizációjának folyamatát, valamint betekintést kapnak a valós idejű adatkezelés alapjaiba. A projekt szemlélteti a szenzoros mérés, adatgyűjtés és informatikai feldolgozás kapcsolatát, miközben fejleszti a digitális kompetenciákat és a természettudományos gondolkodást is.

A program elkészítése

A program elkészítésénél Visual Studio Code[11]-t (VS Code) fogunk használni. A fájl létrehozása után az első dolgunk a külső könyvtárak telepítése és importálása lesz (*41. ábra*). A serial könyvtár fog felelni a soros porton történő kommunikációért, míg a matplotlibbel tudunk majd grafikonokon adatokat megjeleníteni. A telepítés a terminálon keresztül a **pip install pyserial matplotlib** paranccsal fog történni.

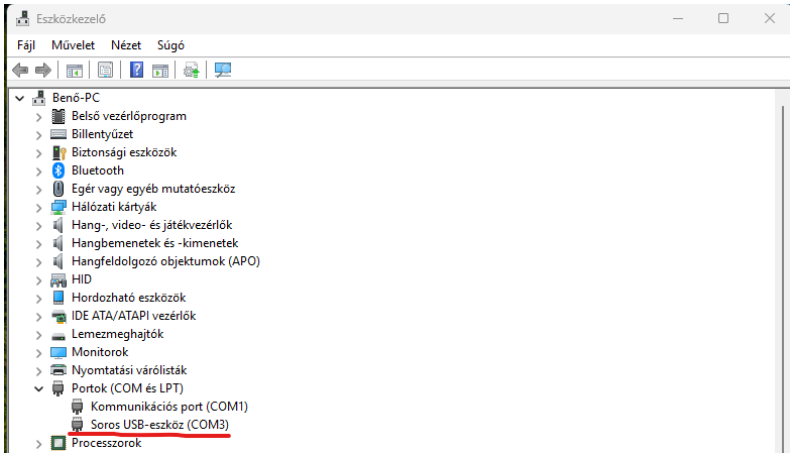
[11] Gyorstalpaló python fileok létrehozásához VS-Codeban.

```
1   import serial
2   import matplotlib.pyplot as plt
3   #külső könyvtárak importálása
4   #serial: soros port kommunikációhoz, matplotlib.pyplot: grafikonok rajzolás
5
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\gaalb> pip install pyserial matplotlib
```

41. ábra A könyvtárak telepítése és importálása a programkódunkba

Ezután állítsuk be a soros port tulajdonságait. Ehhez az Eszközkezelőben (*42. ábra*) nézzük, meg hogy a micro:bit melyik porthoz csatlakozik. A soros port és baudrate[12] beállítása után inicializálni kell az ábránkat is, amelyen megjelenik majd a két grafikon, valamint hozzuk létre a változóinkat is és állítsuk be a kezdőértéküket (*43. ábra*).



42. ábra A micro:bit kommunikációs portja és száma az Eszközkezelőben

```
6   # Soros port beállítása, meg kell nézni, hogy melyik porton van a microbit és a makecode felületet bezárni
7   SERIAL_PORT = 'COM3'
8   BAUDRATE = 115200
9
10  # A soros port beállítása, a microbit által használt port és a baudrate megadásával
11  ser = serial.Serial(SERIAL_PORT, BAUDRATE, timeout=1)
12
13  # Az ábra létrehozása két alábrával: oszlopdiagram és vonaldiagram
14  fig, (ax_bar, ax_line) = plt.subplots(2, 1, figsize=(8, 6))
15
16
17
18  mablak = [] # Ez a lista fogja tárolni az utolsó 50 értéket, amit a vonaldiagramon mutatunk
19  global_min = None # Globális minimum érték, ami a teljes futás alatt változik
20  global_max = None # Globális maximum érték, ami a teljes futás alatt változik
21
```

43. ábra A programunk inicializálása

[12] A baudrate a soros kommunikáció egyik legfontosabb jellemzője. Azt adja meg, hogy másodpercenként hány jelváltás (bit) történik egy kommunikációs vonalon. A micro:bit v2 alapértelmezett értéke 115200 és fontos, hogy minden résztvevő azonos értékre legyen beállítva.

A program futtatása során a makecode felületet mindenképpen zárjuk be, mert gondot okozhat, ha két program is el akarja érni az eszközt.

A 14. sor (*43. ábra*) egy matplotlib függvényhívás, amely létrehozza a grafikon alapstruktúráját:

1. **plt.subplots(2, 1, figsize=(8, 6))** - Létrehoz egy ábrát (figure) két részábrával (subplot)
    - 2, 1 = 2 sor, 1 oszlop elrendezés (egymás alatt)
    - figsize=(8, 6) = az ábra mérete: 8 inch széles, 6 inch magas
2. **Tuple unpacking** történik:
    - fig = a teljes ábra objektum
    - (ax_bar, ax_line) = a két részábra objektum

Az inicializálás után következik az „állandóan” blokkunk (*44. ábra*), amelyet egy **while True** fog megvalósítani. Elhelyezünk egy **try – except KeyboardInterrupt** blokkot, hogy a terminálon leütött Ctrl+C paranccsal be tudjuk zárni a programot. Az except ágban egy print segítségével kiírjuk, hogy kilépés történik és a **break-**kel megszakítjuk a futást. Ezután a **ser.close()** függvénnyel lezárjuk a soros porton történő kommunikációt. Ez lesz tehát a programunk váza, innentől kezdve csakis a **try** ágban fogunk dolgozni.

```
23  while True:# Végtelen ciklus, amíg a program fut
24  >   try: ···
72
73
74      except KeyboardInterrupt: # A program leállítása Ctrl+C-vel
75          print("Kilépés...")
76          break
77
78  ser.close()
```

44. ábra Az állandóan blokk megvalósítása pythonban

Először is a beérkező adatokat eltároljuk egy **line** változóban. Az adatokat tisztítjuk és dekódoljuk UTF-8 formátumra a **ser.readline().decode().strip()** függvényekkel (*45. ábra*).

Meg kell vizsgálnunk utána, hogy van-e már beolvasott sorunk, hiszen ellenkező esetben semmit nem kell csinálni. Ezt figyelembe venni minden további kód a feltételünk igazságában fog megvalósulni. Első lépésként lekerekítjük és nem egész számokra (float) konvertáljuk a beolvasott értékeket, amelyeket innentől kezdve a **value** változóban tárolunk és hozzáadjuk az **mablak** listánkhoz. Ez a lista fogja tárolni az 50 legutolsó mérésünket, ugyanis a példában egy mérési ablakot használunk, amely 50 időpont eredményeit tárolja csak, ezért meg kell nézni azt is, hogy a listánk hossza nagyobb-e mint 50. Ebben az esetben az első elemet az **mablak.pop(0)** függvénnyel

eltávolítjuk. Ezután történik az egész mérés alatt mért abszolút szélsőértékek tárolása és felülbírálata. Amennyiben a global_min None vagy a value, ami az aktuális érték kisebb, mint a korábbi global_min, akkor a global_min-t beállítjuk a value értékére. A maximum esetében a relációs jel megfordul értelemszerűen.

```
while True:# Végtelen ciklus, amíg a program fut
    try:
        line = ser.readline().decode().strip()
        # Beolvassa a soros portból érkező adatokat, dekódolja és eltávolítja a felesleges szóközöket
        if line:
            value = round(float(line), 2)# A beolvasott érték kerekítése két tizedesjegyre
            mablak.append(value)    # Hozzáadja az értéket a mablak listához


            if len(mablak) > 50:  # Csak az utolsó 50 értéket tartjuk meg
                # Ha több mint 50 érték van, eltávolítjuk a legrégebbit
                # Ez segít a memóriahasználat csökkentésében és a diagramok frissítésében
                mablak.pop(0)

            # Frissítjük a globális minimumot és maximumot, klasszikus max-min kiválasztással
            if global_min is None or value < global_min:
                global_min = value
            if global_max is None or value > global_max:
                global_max = value

```

45. ábra A kapott adatok feldolgozása, tisztítása és eltárolása

Most következik az oszlopdiagram frissítése és tulajdonságainak beállítása (*46. ábra*). Itt fogjuk látni az aktuális hőmérsékletet egy szimpla oszlopdiagramon.

```
            # Oszlopdiagram frissítése
            ax_bar.cla() # Töröljük az előző adatokat az oszlopdiagramról, ezzel frissítjük
            ax_bar.bar(['Hőmérséklet'], [value], color='red')   # Oszlopdiagram frissítése, a hőmérséklet értékkel, piros színnel
            ax_bar.set_ylim(10, 40) # Az oszlopdiagram y-tengelyének beállítása 10 és 40 között, szobahőmérséklet körüli tartományban
            ax_bar.set_title('Valós idejű hőmérséklet') # Diagram címének beállítása
            ax_bar.set_ylabel('Érték') # Y-tengely címének beállítása
            ax_bar.text(0, 39, f"{value} °C", ha='center', va='top', fontsize=14, color='black')
            # A hőmérséklet értékének kiírása az oszlopdiagram tetejére a 39-es értékhez
```

46. ábra Az oszlopdiagramunk frissítése és beállításai

Az oszlopdiagram beállítása után az alsó vonaldiagram beállítása következik (*47. ábra*), amely az 50 legutóbbi hőmérsékleti adatokat fogja megjeleníteni.

```
            # Vonaldiagram frissítése
            ax_line.cla() # Töröljük az előző adatokat a vonaldiagramról, ezzel frissítjük
            ax_line.plot(mablak, color='orange', marker='o', label="Aktuális hőmérséklet")
            # Vonaldiagram frissítése, az utolsó 50 értékkel, narancssárga színnel és kör alakú jelölőkkel
            ax_line.set_ylim(10, 40) # A vonaldiagram y-tengelyének beállítása 10 és 40 között, szobahőmérséklet körüli tartományban
            ax_line.set_title('Hőmérséklet változása') # Diagram címének beállítása
            ax_line.set_xlabel('Idő(ms)') # X-tengely címének beállítása
            ax_line.set_ylabel('Érték') # Y-tengely címének beállítása
```

47. ábra A vonaldiagramunk frissítése és beállításai

Ha ezekkel is megvagyunk, akkor már csak annyi dolgunk van, hogy amennyiben a minimum és a maximum változók már rendelkeznek mért értékkel, akkor megjelenítjük őket is, szaggatott

vonalak formájában az alsó diagramon. A minimum esetében kék, míg a maximum esetében piros szöveggel. Egyben fel is címkézzük ezeket f-string használatával (48. ábra).

Az f-string egy speciális string literál, amely az f vagy F előtaggal kezdődik, és lehetővé teszi változók és kifejezések beágyazását közvetlenül a stringbe {} kapcsos zárójelekkel.

Példa a kódodból:

ax_bar.text(0, 39, f"{value} °C", ha='center', va='top', fontsize=14, color='black')

Itt az f"{value} °C" egy f-string, ahol:

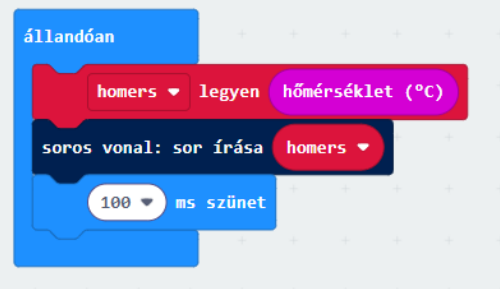
- f = jelzi, hogy ez egy formatted string
- {value} = a value változó értékét helyettesíti be
- °C = sima szöveg, ami változatlan marad

Ezután pedig hozzáadjuk a jelmagyarázatot a jobb felső sarokba, majd a plt.pause(0.1) függvénynel egy 100ms-es várakozást állítunk be, amely a frissítések gyakoriságáért fog felelni.

```
63              # Teljes futás alatti minimum és maximum megjelenítése
64              if global_min is not None and global_max is not None:
65                  ax_line.axhline(global_min, color='blue', linestyle='--', label=f"Futás min: {global_min} °C")
66                  # Kék színű vízszintes vonal a globális minimum értéknél
67                  ax_line.axhline(global_max, color='red', linestyle='--', label=f"Futás max: {global_max} °C")
68                  # Piros színű vízszintes vonal a globális maximum értéknél
69                  ax_line.legend(loc='upper right') # Jelmagyarázat hozzáadása a vonaldiagramhoz
70
71              plt.pause(0.1)
72
73
74      except KeyboardInterrupt: # A program leállítása Ctrl+C-vel
```

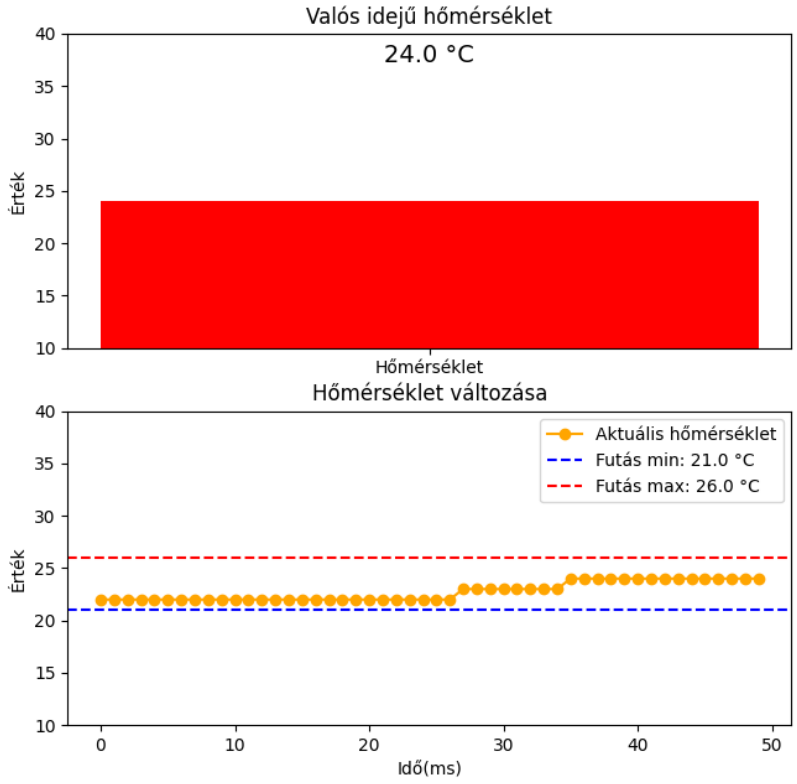
48. ábra A min/max értékek megjelenítése a diagramon, jelmagyarázat hozzáadása és a frissítési gyakoriság beállítása

A programunk python része elkészült, a nehezén túl is vagyunk, hiszen a micro:biten használt kód rendkívül rövid és egyszerű lesz (*49. ábra*). Csak az állandóan blokkra lesz szükségünk, amelyben egy homers változóban eltároljuk a hőmérsékletet, amelyet a micro:bittel mérünk, majd a soros vonal: sor írása homers blokkal átküldjük az adatokat a soros portra és 100ms szünetet rakunk ide is. Figyeljünk arra, hogy ez a szünet megegyezzen a python-ban alkalmazott szünettel, az esetleges hibák elkerülése érdekében.



49. ábra A projekt micro:bithez tartozó programkódja

Ha lementettük a kódot a micro:bitre, akkor a Makecode felületet **mindenképpen zárjuk be**, ahogy azt korábban említettük. Ezután a VS Code-ban futtassuk a python scriptet és egy felugró ablakban meg fog jelenni a két diagram (*50. ábra*).



50. ábra Az elkészült projekt adatvizualizációs ablaka

Továbbfejlesztési lehetőségek/további ötletek

A projekt továbbfejleszthető azzal, hogy a rendszer a hőmérsékleti adatok mellett más környezeti paramétereket is (pl. páratartalom, légnyomás) mér és jelenít meg párhuzamosan. Ehhez további szenzorokat lehet csatlakoztatni a micro:bithez, majd a Python-kódot módosítani, hogy a soros porton keresztül többféle adatot tudjon fogadni és azokat külön-külön grafikonokon megjeleníteni. Így egy valódi időjárás-állomást szimuláló rendszer hozható létre, amely komplex környezeti adatokat is kezel és elemez.

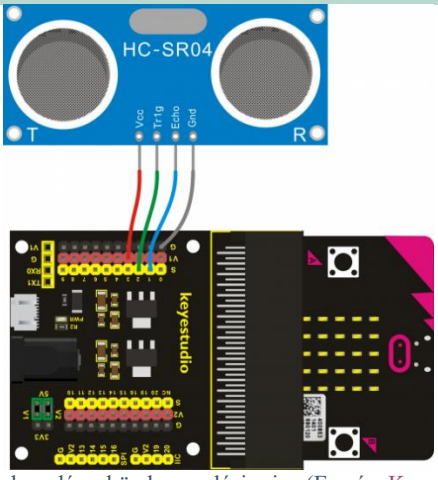
Fogalomtáblázat

Kifejezés	Magyarázat
.decode().strip()	A kapott byte típusú adat szöveggé alakítása és megtisztítása
adatablak (mablak lista)	Egy lista, amely az utolsó 50 hőmérsékleti értéket tárolja
fig, ax = plt.subplots()	Függvény, amely egy rajzfelületet és alá tartozó grafikonokat hoz létre
f-string	Formázott szöveg, amelyben változók értékeit közvetlenül lehet megjeleníteni
függvény (eljárás)	Előre definiált utasításblokk, amit név alapján többször is végrehajthatunk.
global_min, global_max	A mérési időszak alatt rögzített legkisebb és legnagyobb értékek tárolása
matplotlib könyvtár	Adatvizualizációs csomag grafikonok, diagramok létrehozásához
oszlopdiagram (ax_bar)	Aktuális hőmérséklet vizuális megjelenítése egy oszloppal
plt.pause(0.1)	100 ms szünet, amely biztosítja a folyamatos, de nem túl gyors frissítést
pop(0)	A lista első elemének eltávolítása, ha elértük az 50 elem hosszúságot
ser.readline()	A soros porton érkező adat beolvasása
serial könyvtár	A Python modul, amely lehetővé teszi a soros port kommunikáció kezelését
soros kommunikáció	Adatátvitel micro:bit és számítógép között USB-kábelen keresztül
try – except szerkezet	Hibakezelési forma: kilépés Ctrl+C-vel, hiba esetén biztonságos megszakítás
vonaldiagram (ax_line)	Az utolsó 50 érték időbeli változásának megjelenítése
while True:	Végtelen ciklus, amely folyamatosan új adatokat olvas be és jelenít meg

3.8 Radar szimulátor (Python alapú foglalkozás)

Fő tantárgy, ahol megjelenhet	Fizika
A foglalkozás helye	A hullámok szerepe a kommunikációban
Szemléltető eszköz	Digitális, valós idejű hőmérőeszköz készítése, időjárásállomások szimulálása
Felhasznált eszközök a modell elkészítéséhez	• 2 db micro:bit V2 • micro:bit Sensor Shield V2 • ultrahangos távolságmérő • 4 csatlakozókábel • külső áramforrás a Shieldhez
Programozáshoz köthető ismeretek	• adatvizualizáció • soros kommunikáció és hardver interfész
A foglalkozás digitális kultúrához kapcsolódó céljai	• A diákok képesek legyenek a micro:bit által mért távolságadatokat valós időben Python segítségével beolvasni és értelmezni • Megtanulják a különböző típusú diagramok (pl. scatter plot, polárdiagram) készítését a mért adatok szemléltetésére

A szenzorok kapcsolása



51. ábra A radar adóeszköz kapcsolási rajza (Forrás: Keyestudio Wiki)

A foglalkozásról röviden

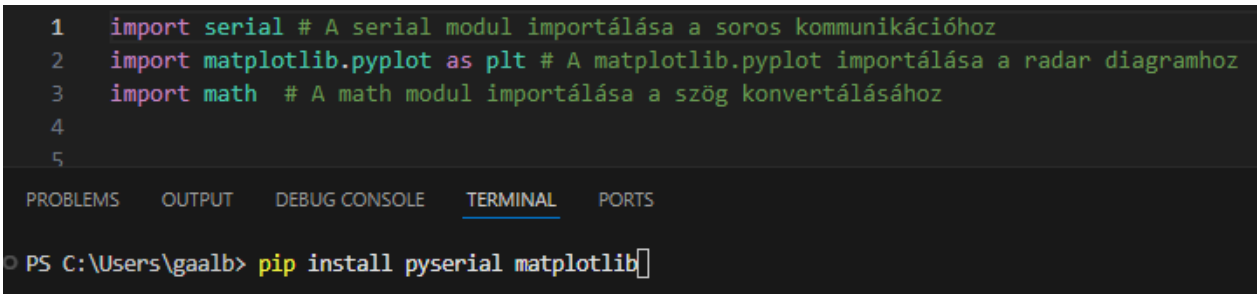
A foglalkozás során a diákok egy egyszerű radar szimulációt készítenek két micro:bit és egy ultrahangos távolságmérő szenzor felhasználásával, Python programozással. A méréseket végző micro:bithez csatlakozik az ultrahangos szenzor, és a beépített iránytű segítségével meghatározza,

milyen irányban történt az adott mérés. A mért adatokat vezeték nélküli rádiókapcsolaton keresztül továbbítja egy második micro:bithez, amely USB-n keresztül továbbküldi azokat a számítógép felé. Így az érzékelő nem helyhez kötött, szabadon mozgatható marad.

A számítógépen futó Python program a matplotlib könyvtár segítségével vizuális megjelenítést biztosít egy polárdiagramon (radar nézet) az irány és a távolság alapján. A tanulók valós idejű szenzoros mérést végeznek, adatokat dolgoznak fel, és fejlesztik adatvizualizációs és informatikai gondolkodásukat.

A program elkészítése

A program elkészítéséhez itt is, csak úgy, mint az előző foglalkozásban a VS Code-t[13] fogjuk használni. Első lépésként a programunk során használt könyvtárak telepítésére és importálására lesz szükség (*52. ábra*). Ebben a projektben a serial, matplotlib és math könyvtárakat fogjuk felhasználni. A telepítés az előző foglalkozástervhez hasonlóan a pip install pyserial matplotlib terminálparanccsal fog történni. A math könyvtár az alapcsomag része, nem kell telepíteni. Ha ezzel megvagyunk, akkor az import paranccsal beimportáljuk a könyvtárakat az alábbi ábrán (*52. ábra*) látható módon.



52. ábra A könyvtárak telepítése és importálása

Következő lépésként keressük meg, hogy a vevő micro:bit melyik porthoz csatlakozik (*42. ábra*), majd állítsuk be a soros portunkat (*53. ábra*).

```
7  # Soros port beállítása, meg kell nézni, hogy melyik porton van a microbit és a makecode felületet bezárni
8  SERIAL_PORT = 'COM7' # A soros port neve, amely a microbit-hez csatlakozik
9  BAUDRATE = 115200  # A soros port sebessége, amely megegyezik a microbit beállításával
10 ser = serial.Serial(SERIAL_PORT, BAUDRATE, timeout=1) # Soros port megnyitása a megadott sebességgel
```

53. ábra A soros kommunikáció inicializálása

A megjelenő adatvizualizációt egy poláris diagramon megjelenő pont fogja adni. Ehhez először létre kell hozni egy ábrát **a plt.figure()** függvénnyel, majd létrehozni a diagramot (*54. ábra*). A

[13] Gyorstalpaló python fileok létrehozásához VS-Codeban.

diagram y tengelyét 400-as távolságra állítjuk, mert a micro:bithez kapcsolt ultrahangos távolságmérő 400 cm-ig tud működni. Ezek után pedig beállítjuk, hogy a diagram északi oldalán legyen a 0 pont és az óra járásával megegyező irányban haladjon a számozás. Kelleni fog nekünk két változó is, amelyeknek értékét **None-ra** állítunk. Az egyik a **kezdo_szog**, míg a másik a radar változó lesz. Az elsőre azért lesz szükség, hogy a micro:bitről küldött első iránytű irányt a 0 pozícióba tegye és a későbbiekben relatívan ehhez fogjuk az objektum irányát vizsgálni, mintha egy hajón lennénk. A második változóban pedig a diagramon megjelenítendő pont objektumot fogjuk eltárolni majd.

```
13  # A radar diagram beállítása
14  plt.figure() # Új ábra létrehozása, amelyen a diagram megjelenik
15  ax = plt.subplot(polar=True)  # Poláris diagram létrehozása
16  ax.set_ylim(0, 400)  # A maximális távolság beállítása 400 cm-re, a vonalak közötti távolság automatikusan igazodik
17  ax.set_theta_zero_location("N")  # Az északi irány (felső) beállítása 0 fokra
18  ax.set_theta_direction(-1)  # Az irány beállítása óramutató járásával megegyezőre
19
20  # Változó az első szög tárolására
21  kezdo_szog = None
22
23  # Változó a radar pontok tárolására
24  radar = None
```

54. ábra A poláris diagram és változók létrehozása

Az inicializálás után, csak úgy mint az előző foglalkozásnál, következik az „állandóan” blokkunk (*55. ábra*), amelyet egy **while True** fog megvalósítani. Elhelyezünk egy **try – except KeyboardInterrupt** blokkot, hogy a terminálon leütött Ctrl+C paranccsal be tudjuk zárni a programot. Az except ágban egy print segítségével kiírjuk, hogy kilépés történik és a **break**-kel megszakítjuk a futást. Ezután a **ser.close()** függvénnyel lezárjuk a soros porton történő kommunikációt. Ez lesz tehát a programunk váza, innentől kezdve csakis a **try** ágban fogunk dolgozni.

```
26  # Adatok olvasásának indítása
27  while True: # Végtelen ciklus, amíg a program fut
28      try: ...
53
54      except KeyboardInterrupt: # A program leállítása, amikor a felhasználó megnyomja a Ctrl+C-t
55          print("Kilépés...")
56          break
57  # A soros kapcsolat bezárása
58  ser.close()
```

55. ábra Az állandóan blokk megvalósítása pythonban

A try-ban szereplő kód a soros porton érkező adatok beolvasásával kezdődik a line változóba (*56. ábra*). Az előző foglalkozáshoz képest eltérés, hogy itt több adatot használunk így a **ser.readline().decode().strip()** függvény kiegészül egy **.split(’,’)** függvénnyel is, amely az adatokat feldarabolja a vessző mentén. Így a line változó gyakorlatilag egy lista lesz. Amennyiben van beolvasott adatunk, akkor a tav változóban eltároljuk a line lista első elemét (**line[0]**) a szog változóban pedig

a line lista második elemét (**line[1]**). Utána pedig, ha a **kezdo_szog** még **None** értékkel bír, akkor a **kezdo_szog** egyenlő lesz a **szog** változó értékével. Szükségünk lesz majd a korrigált relatív szögre is, ezért az **igazitott_szog** változóban meghatározzuk a relatív szögértékünket a **szog – kezdo_szog** művelettel.

```
28      try:
29          # Olvasson be egy sort a soros portból
30          line = ser.readline().decode().strip().split(',')
31          if line:
32              #print("Nyers adatok:", line)  # Nyers adatok megjelenítése hibakereséshez
33
34              # Az adatok számokká konvertálása
35              tav = float(line[0])
36              szog = float(line[1])
37
38              # Az első szög beállítása elülső szögként
39              if kezdo_szog is None:
40                  kezdo_szog = szog
41
42              # A szög beállítása az első szöghöz képest
43              igazitott_szog = szog - kezdo_szog
44
45              # A scatter plot frissítése az egész diagram törlése nélkül
46
```

56. ábra A soros portról érkező adatok beolvasása, tárolása és a szögértékek kiszámítása

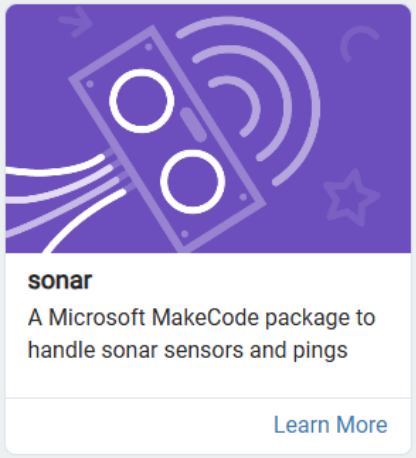
Nem maradt sok hátra a python kódunkból, csak annyi, hogy a radar változóban eltároljuk és ezzel létrehozzuk a pontunkat a diagramon. Ezt a radar = ax.scatter(math.radians(igazitott_szog), tav, color='red')

Ez a sor egy **scatter plot pontot** ad hozzá a radar diagramhoz. Részletesen:

- **ax.scatter()** - Egy pontot rajzol a poláris koordináta-rendszerben
- **math.radians(igazitott_szog)** - A szöget fokból radiánba konvertálja (matplotlib poláris diagramok radiánokat várnak)
- **tav** - A távolság értéke, ami meghatározza, hogy milyen messze legyen a pont a középponttól
- **color='red'** - A pont színe piros lesz
- **radar** = - A létrehozott pont objektumot eltárolja a radar változóban

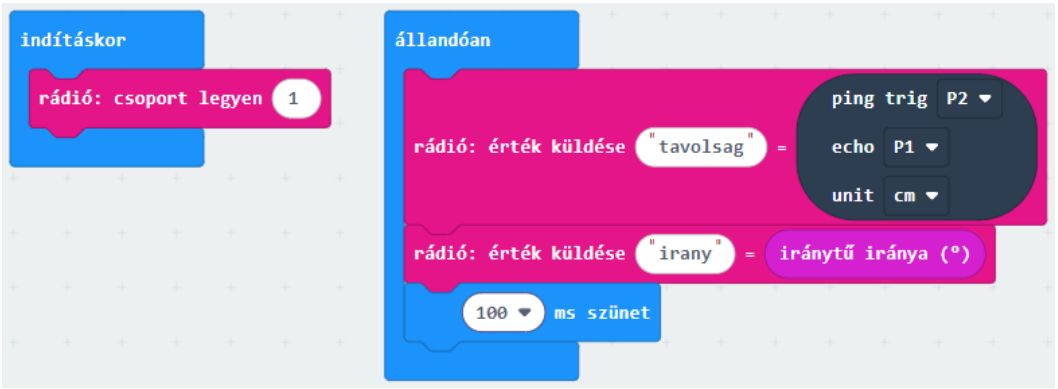
Ez a sor felelős azért, hogy a radar szenzor által mért adatok (szög és távolság) **vizuálisan megjelenjenek** a radar diagramon. Minden mérési ciklusban egy új piros pontot rajzol oda, ahol a szenzor akadályt észlel. Ezután az ábránkat frissítjük a **plt.pause(0.1)** kóddal majd a **radar.remove()** függvénnyel töröljük az aktuális pontot, hogy a ciklus következő futásakor már az új jelenjen meg. A program python kódja ezzel elkészült.

Következő lépésként a micro:bitek közül az adóeszköz kódját fogjuk áttekinteni. Az indításkor blokkban állítsuk be a rádiócsoportot. Figyeljünk, hogy mindkét eszköz ugyan abban a rádiócsoportban legyen. A szenzor működtetéséhez hozzá kell adnunk a sonar (*57. ábra*) kiterjesztést a micro:bithez, ehhez kattintsunk a Kiterjesztések blokkra és írjuk be a keresőbe a „sonar” kifejezést.



57. ábra A sonar kiterjesztés, amely szükséges a szenzor működéséhez.

Ha ez megtörtént, akkor álljunk neki a program fő részének elkészítéséhez (). Az állandóan blokkban a rádió: érték küldése „tavolsag” = ping trig (P2) echo (P1) unit (cm) blokkal tudjuk elküldeni a távolságértéket, amit a micro:bit a szenzorral mér. A szenzorhoz kapcsolódó blokk a sonar blokkcsoportban található. Ezután a rádió: érték küldése „irány” = iránytá iránya (°)[14] blokkal pedig elküldjük a fokot. Alkalmazzunk egy 100ms várakoztatást is.



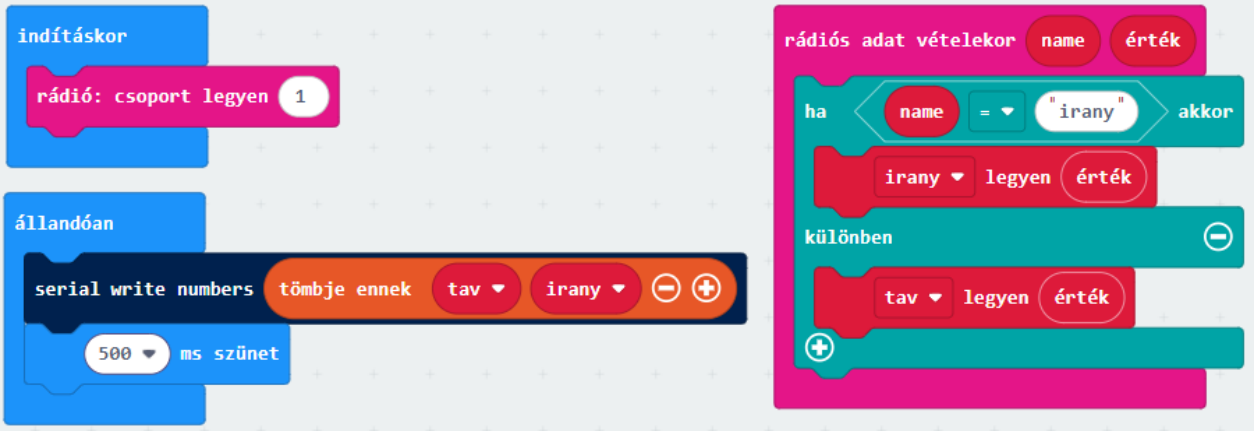
58. ábra Az adóeszköz programkódja

A vevőeszköz programkódja szintén nem lesz bonyolult (*59. ábra*). Az indításkor blokk megegyezik az adó programkódjával. Eseményvezérléssel tudjuk megvalósítani a kapott adatok fogadását,

[14] Az iránytű használatához a micro:bitet az indítás után döntögetni kell addig, amíg a villogó pont kiszínezi az egész kijelzőt. Ezzel történik a az iránytű kalibrációja.

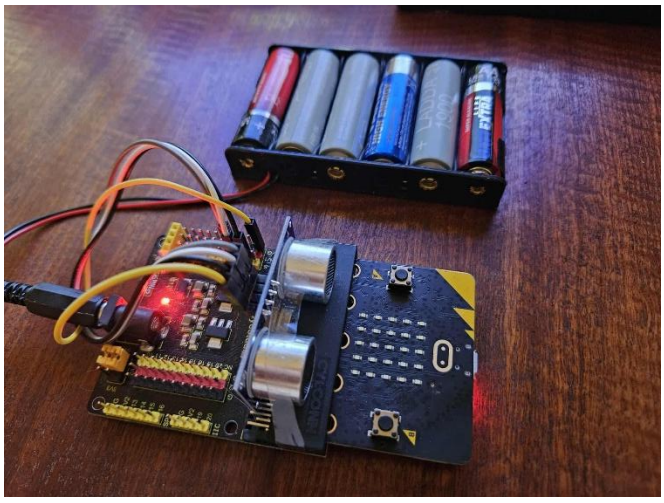
méghozzá a rádiós adat vételekor name érték blokkal. Ezzel lehetőségünk van a kapott adatokat szortírozni és megvizsgálni, hogy milyen név-érték párosban érkeznek. Ezt egy ha feltétel vizsgálattal tudjuk megtenni, ugyanis, ha name egyenlő „irany” akkor irany változó legyen érték, különben tav legyen érték.

Az állandóan blokkunkban pedig a serial write numbers (tömbje ennek (tav) (irany)) blokkal el tudjuk küldeni az adatokat a soros porton keresztül 500ms szünetet alkalmazva.



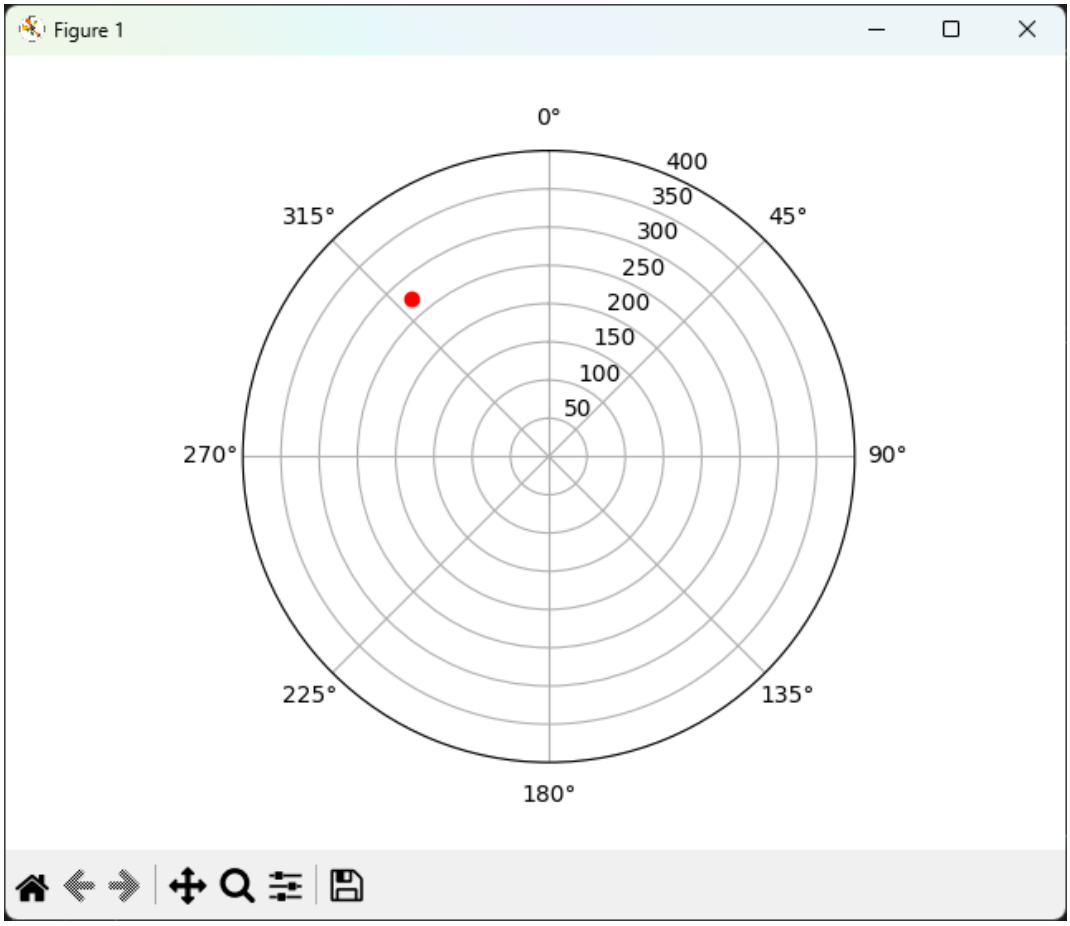
59. ábra A vevőeszköz programkódja

Érdemes lehet az adóeszköz esetében a szenzort valahogy rögzíteni az eszközre (*60. ábra)*, hogy az együtt forduljon a micro:bittel, így a valós irányt fogjuk látni a megjelenő grafikonon. A példánkban egy sima cellux segítségével történt meg a rögzítés.



60. ábra Az eszköz és a rá rögzített szenzor képe

Ha elkészültünk, akkor nincs más dolgunk, mint futtatni a python scriptet és megfigyelni a grafikus felületen, hogy miként jelenik meg az objektum a radarunkon (61. ábra).



61. ábra Képernyőkép a program futásáról

Továbbfejlesztési lehetőségek/további ötletek

A projekt bővíthető több objektum egyidejű megjelenítésével, azaz ne csak az utolsó mért pont legyen látható a radar nézetben, hanem a legutóbbi mérésekből álló „radarkép” rajzolódjon ki. Ehhez a program módosítható úgy, hogy a pontokat egy listában tároljuk, és minden ciklusban a teljes listát újrarajzoljuk. További lehetőségként implementálható egy távolságalapú szűrés, amely például más színnel jeleníti meg a veszélyesen közeli tárgyakat. A mérések naplózása CSV-fájlba is megvalósítható, így az adatok később elemezhetők vagy újra megjeleníthetők.

Fogalomtáblázat

Kifejezés	Magyarázat
baudrate	Soros kommunikáció sebessége, bit/másodperc egységben
függvény (eljárás)	Előre definiált utasításblokk, amit név alapján többször is végrehajthatunk (pl. karakter(), betuiras()).
math.radians()	Fokban megadott szöget átvált radiánra – szükséges matplotlib polárdiagramhoz
plt.pause(0.1)	100 milliszekundumos frissítésidő – ezzel lehet animáltan megjeleníteni az adatokat
polar diagram (polárdiagram)	Kördiagram, ahol a távolságot a sugár, az irányt a szög jelöli – a radar nézet alapja
radar.remove()	A korábbi pont törlése a diagramról, hogy csak az aktuális pozíció látszódjon
radio: érték küldése	A micro:bit Makecode parancsa rádión keresztüli adattovábbításra
relatív szög	Az aktuális mért szög és az első mért irány különbsége – a mérés irányának beállításához
scatter plot	Szórásdiagram – pontokból álló grafikon, amely adott adatpárokat jelenít meg
scatter()	Matplotlib függvény, amellyel pontokat lehet megjeleníteni egy grafikonon
serial (soros) kommunikáció	Adatátvitel egy vezetéken keresztül – például micro:bit és számítógép között
serial write numbers	Makecode blokk, amely számokat küld a micro:bit USB-n keresztül a számítógép felé
try – except blokk	Hibakezelési szerkezet Pythonban – biztosítja a program stabil leállítását
ultrahangos távolságmérő	Szenzor, amely hanghullámok visszaverődésének ideje alapján számítja ki a távolságot