Készítsünk labirintust

Következő példánkban a micro:bit segítségével fogunk előállítani véletlenszerűen labirintust, mégpedig úgy, hogy minden előállított labirintus pontosan ugyanannyi (pl. 15) darab kigyújtott pontból álljon.

Először induljunk ki az alábbi állapotból. Az „A" gomb hatására 15 alkalommal véletlen koordinátát generálunk és kigyújtjuk az ahhoz tartozó pontot.

Teszteljük le a megoldást. Kérdezzük meg a diákokat, hogy valójában hány pont jelent meg a kijelzőjükön. Látszik, hogy általában nem mind a 15 pont lesz kigyújtva, mert véletlenszerűen kiválaszthatunk olyan pontokat is, amelyek már ki vannak gyújtva.

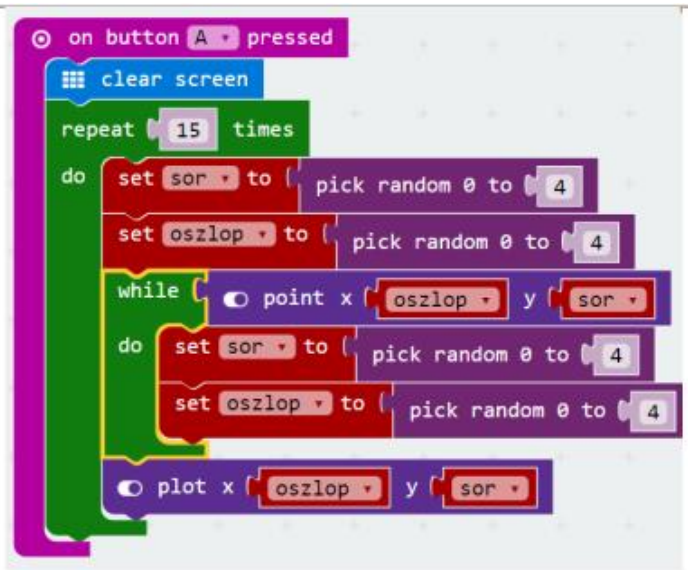
A megoldás az lehetne, hogy addig kellene új véletlenszámot generálni, míg üres képpontot nem találunk. Ezzel garantálhatnánk, hogy pontosan 15 pont legyen kigyújtva.

Azt, hogy egy adott pont ki van-e gyújtva a **Led** kategória `points x y` blokkjával kérdezhetjük le. Igaz (true) értéket kapunk vissza, ha ki van gyújtva a pont, és hamisat (false), ha nem.

Ötleteljünk, hogyan tudnánk ez alapján megvalósítani, hogy pontosan 15 pont legyen kigyújtva véletlenszerűen?

Megoldás lehet, hogy addig generálunk újabb és újabb koordinátákat véletlenszerűen, míg nem kapunk olyan pontot, ami nincs kigyújtva.

Ezt a **Loops** / `While feltétel do` ciklusával tudjuk megtenni az alábbi módon.



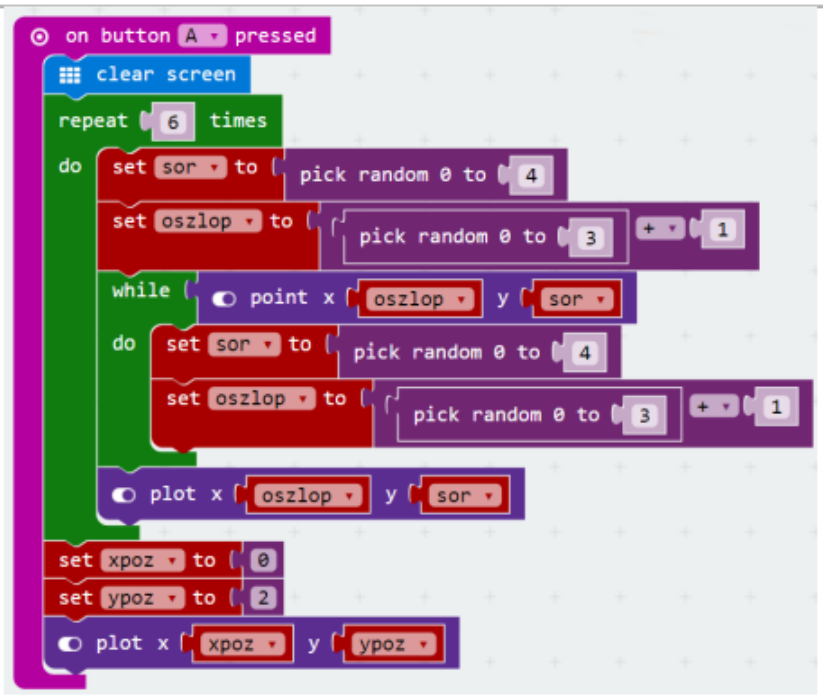
Így már ténylegesen annyi pont lesz kigyújtva, amennyit a `repeat` ciklusban beállítottunk.

Próbáljuk ki a programot 24 ponttal is. Ilyenkor csak 1 pont marad, ami nem lesz kigyújtva.

Menjünk végig a labirintuson

Most már tudunk véletlenszerűen labirintus létrehozni. Készítsünk olyan alkalmazást, amelyben végig tudunk vinni egy pontot a labirintuson. Az eszköz 4 irányba döntésével fogjuk megváltoztatni a pont pozícióját úgy, hogy csak üres pontra léphetünk.

Kezdetben a pontunk az X=0, Y=2 koordinátán legyen elhelyezve, és a jobb alsó sarokba kelljen eljutnunk az akadályok kikerülésével. Hogy nagyobb eséllyel végig tudjunk menni a pályán, 6 pontból álló labirintus generáljunk úgy, hogy az első oszlopba ne tegyünk akadályt.

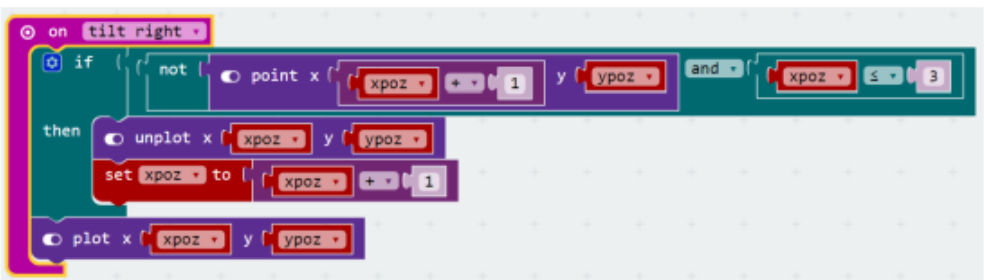


Beszéljük át közösen, hogy minek kell történnie akkor, ha jobbra döntjük az eszközt!

Egyrészt vizsgálnunk kell, hogy a jobb oldali szomszédos mező üres-e. Ha igen, akkor az eredeti helyéről törölnünk kell a pontot, és az új helyen meg kell jelenítenünk azt, és ezt a koordinátát el kell tárolnunk. Emellett azt is néznünk kell, hogy legfeljebb az utolsó előtti oszlopban lehetünk a jobbra lépés előtt, mivel nem szabad kilépnünk a területből (xpoz<=3).

Az üres hely vizsgálatánál használjuk a `not` blokkot, így a feltételünk akkor lesz igaz, ha üres az adott pont.

A megoldás tehát:



Most a diákokon a sor, hogy mind a 4 irányú döntésre elkészítsék a megfelelő kódot.

Menjünk végig a labirintuson (döntés 4 irányba)

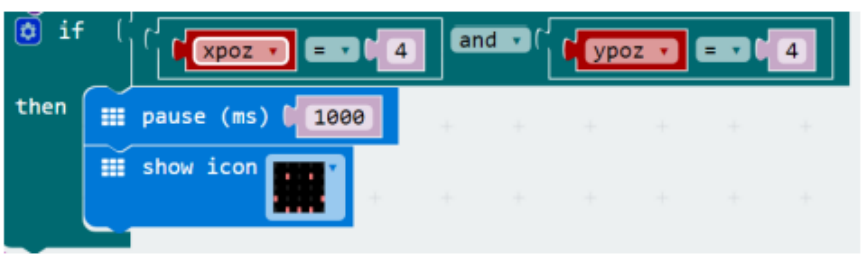
Feladat a diákok számára

Alakítsd át az előbbi programot úgy, hogy a pontot az eszköz 4 irányban történő döntésével (`tilt right, title left, logo up, logo down`) lehessen irányítani.

Labirintus, eredmény kijelzés

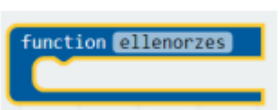
Jó lenne, ha kapnánk egy szép visszajelzést arról, hogy sikerült a feladatot megoldani, vagyis eljutottunk a jobb alsó sarokba! Ilyenkor 1 másodperc múlva jelenjen meg egy vigyorgó fej!

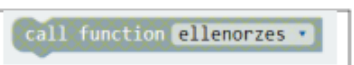
Ezt megoldhatjuk úgy, hogy az alábbi blokkot elhelyezzük mind a 4 blokkunk végén.



De ha ezt tesszük, és úgy döntünk, hogy például ne vigyorgó fej jelenjen meg, hanem egy animáció, esetleg egy szöveg, akkor innentől 4 helyen kell módosítani a kódunkat. Ez nagyon nem hatékony. Ezért azon kódrészleteket, amelyek többször is felhasználásra kerülnek érdemes függvényekbe szervezni.

Ehhez a `functions` kategória `make a function` gombját kell megnyomnunk és meg kell adnunk a függvény nevét. A név legyen a következő: ellenorzes.

Ekkor megjelenik a következő blokk: 

Ebben elhelyezhetjük a megfelelő kódot. A függvényt a 

blokkal hívhatjuk meg, amely automatikusan létrejön a függvény létrehozásakor.

Mind a 4 blokk végére helyezzük el a függvény hívást.

A teljes projektünk elérhető a https://makecode.micro:bit.org/_Tcxb2Riuf776 címen.

Megjegyzés: mivel háttérfolyamat használatára is lehetőségünk van a `forever` blokk használatával, azt is megtehetjük, hogy a `forever` blokkban helyezzük el az „ellenorzes” függvényhívást. Ekkor a többi blokkból törölhetjük is ezt a függvényhívást.