BLOCKLY

Tehetséggondozó program
NTP-MTI-13-0049 - NTP-MTI-14-0099

Bécsi Zoltán, Bojda Beáta,
Bubnó Katalin, Domonkos Péter,
Kelemenné Nagy Anikó, Takács Viktor László

Debreceni Egyetem
Kossuth Lajos Gyakorló Gimnáziuma
2014-2015

Tartalom

A blokknyelvek története 4
Blockly 6
Számbontás 10
Dávid anyja 14
Egyenlőtlenség 21
Szállásfoglalás 23
MIT Appinventor2 26
Tanulókártyák 27
Színek beállítása 27
Hangfájlok hozzáadása 28
Telefon rázása 29
Beszédfelismerés 30
Véletlenszerű Sprite elhelyezés 31
Időzített Sprite mozgatás 32
Sprite-ok leállítása időzítővel 33
Mozgatás gombokkal 34
Sprite mozgatása ujjal 35
Suhintás 36
Szenzoros mozgatás 37
Sprite-ok ütközése 38
Pattogó labda 39
Rajzolás a Canvas-ra 40
Több képernyő kezelése 41
Emlékezz rám 42
Bluetooth kommunikáció 43

Mobil alkalmazások 44
MetalPony 44
Forest Gump 48
MagicPaint: húzás, érintés 57
Blocklino - Arduino 70
LED vezérlése 71
Külső LED vezérlése 73
LED fényerejének változtatása 74
Nyomógomb kezelése 76
Soros port használata 78
Adatok küldése soros porton a számítógépre 80
Adatok fogadása soros porton az Arduinótól 82
Analóg bemenet olvasása 85
LED fényerejének állítása potméterrel 87
LED kocka vezérlése telefonnal 89

A blokknyelvek története

Az első, fiatalok számára írt és szélesebb körben elterjedt blokknyelv az Massachusetts Institute of Technology Mitchell Resnick vezette Media Lab által fejlesztett és támogatott Scratch programozási környezet volt 2003-ban. A Scratch sikeres volt több országban, elsősorban angolszász nyelvterületen, bár sok más nyelv mellett magyarra is lefordították és honosították. Magyarországon igazán széles körben azonban mégsem tudott elterjedni. Nem tudta leváltani a közoktatási gyakorlatban a LOGO programozási nyelvet, holott a Scratch-et úgy tartjuk nyilván a nemzetközi szakmai gyakorlatban, mint a LOGO egyfajta folytatását, már csak azért is, mert a mögötte lévő ideológia ugyanaz: kreatív, kísérletező módon, cselekedtetve tanítani. A Scratch grafikus elemei révén sokkal felhasználóbarátabb, és jobban követi a felmerülő új infokommunikációs trendeket, mint például a közösségi informatika elemeinek használata. A LOGO és a Scratch közt volt más grafikus LOGO kísérlet is, ez volt a StarLOGO TNG, szintén Mitchell Resnick csapata fejlesztette. Ennek változatai is élnek ma is, a legújabbal 3D-s modellezés és szimuláció, illetve 3D grafikát felvonultató játékok is tervezhetők.

Blokknyelvek szép számban születtek az MIT jóvoltából, több területen is, szak specifikus nyelveket alkottak főként az MIT-n végzős diákok, sokszor diplomadolgozatként.

Igazi áttörést a blokknyelvek elterjedésében a Google Inc. hozott, melynek egy fejlesztő csapata közös munkában az MIT kutatóival 2008-ban egy olyan blokknyelv fejlesztésén kezdett dolgozni, mely kifejezetten az Android operációs rendszerrel futó mobileszközökre való programok, alkalmazások készítését tette lehetővé érdeklődő fiatalok számára. Egészen addig mobileszközökre csak professzionális programozók tudtak szoftvereket írni, most azonban az App Inventor for Android nevű nyelv segítségével megnyílt a lehetőség bárki számára, az egyszerűbbtől a komolyabb alkalmazások elkészítéséig, és ehhez nem volt szükség előzetes programozói tudásra.

Az App Inventor for Android fejlesztését végül teljes egészében átvette az MIT 2012-ben. Második verziója fut jelenleg, mint felhőszolgáltatás. Ez azt jelenti, hogy senkinek nem kell telepítenie semmit a számítógépére, aki használni szeretné, csak egy Google fiókba való bejelentkezéssel be tud lépni az online fejlesztői környezetbe, ahol a saját projektjein dolgozhat. Az elkészült alkalmazás webcíméről egy kódgenerátor segítségével QR-kódot (quick response, vagyis gyors válasz) készíthetünk, melyet leolvasva okostelefonunkkal (vagy táblagépünkkel) telepíthetjük saját szoftverünket a mobileszközre, és azonnal használhatjuk azt a telefonunkon.

A blokknyelvek története

Az App Inventor sikerét látva a Google elkészítette és nyilvánossá tette az App Inventor 2 és további hasonló blokknyelvek motorjaként szolgáló Blockly nevű programkönyvtárat, mely felhőszolgáltatásként online, mint általános programozási nyelv is használható, illetve segítségével bárki készíthet saját blokknyelvet is akár (Blockly Code). A nyilvános közzétételnek robbanásszerű hatása lett. Csomagológép programozását lehetővé tévő blokknyelvtől kezdve intelligens eszközöket, digitális áramköröket vezérlő különböző szakterületi nyelvek születtek, egyik szerzőnk is készített egy webes prezentáció készítést lehetővé tévő blokknyelvet, a BlockImpress-t.

A blokknyelvek grafikus programozási nyelvek. Ez azt jelenti, hogy szöveges parancsok megtanulása és begépelése helyett a grafikus környezetben egyszerű drag-and-drop technikával összerakhatjuk a megadott parancskészletből a programunk forráskódját. Az egész egy puzzle-játékhoz hasonlít online módon. A blokknyelvek kiküszöbölik azt a kezdő programozókat gyakran érő kudarcélményt, mely a nyelv nem megfelelő elsajátításából fakad.

A jórészt szintaktikai nehézségek tehát minimálisra csökkenthetők, és a lényeg a problémamegoldás lesz, a helyes algoritmus kiválasztása és implementálása, vagyis a számítógépes gondolkodás elsajátítása. Ezért oktatásban hamarabb sikerrel alkalmazhatók a hagyományos programozási nyelveknél.

Az informatika tantárgy kevés óraszámmal rendelkezik, ezen belül pedig igen kevés a programozásra fordítható idő. Azt szeretnénk, ha ebben a kevés órában nem rossz élményeket, hanem sikerélményt, a valódi alkotás élményét adhatnánk meg diákjainknak digitális környezetben.

A vezető információtechnológiai cégek egyöntetűleg a blokknyelvek, mint alkalmas oktatási eszköz mellett tették le voksukat, amikor 2013 decemberében meghirdették az első Hour of Code rendezvénysorozatot. A kezdeményezéssel a számítógépes programozás tanulás fontosságára hívták fel a figyelmet, melyet véleményük szerint már az általános iskolás korosztály számára megfelelő eszközzel el kell kezdeni és minden szinten folytatni. Az Hour of Code kezdeményezéshez csatlakozók vállalják, hogy megtartják intézményükben is a Kódolás óráját, melynek során egy kijelölt hét egy órájában a lehető legszélesebb közönséggel ismertetik meg a számítógépes programozást.

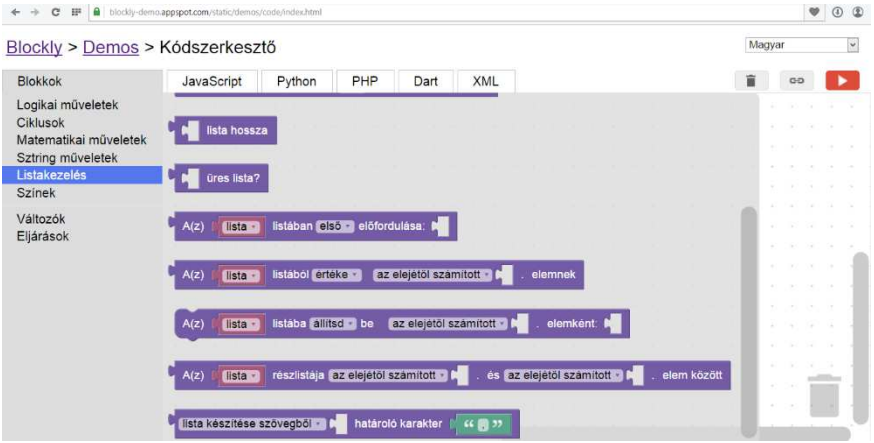
A program első nyelveként a Blockly programkönyvtárat, mint általános programozási nyelvet használjuk szekvenciális programozásra. A szekvenciális programozás azt a hagyományos programkészítési technikát jelenti, amikor a gép sorról sorra haladva értékeli ki és hajtja végre a programot. Ezzel a programozási nyelvvel mi matematikai problémákat oldunk meg. Ennek a modulnak a célja egyrészt a blokknyelvekre való „ráhangolódás”, a blokknyelvi fejlesztő környezettel való megismerkedés, másrészt a programozáshoz szükséges matematikai alapok és a két tudomány közti kapcsolatok kiépítése gyakorlatorientált módon.

A fejlesztő környezetről

A Blockly Code kódszerkesztő és programkönyvtár magyar nyelvű változata jelenleg az alábbi linken található:

https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hu

A képernyő bal oldalán menüsort találunk, mely funkciók szerint csoportosítja a blokkokat. Az egyes blokkcsoportok a vizuális képességek kihasználása érdekében különböző színűek lesznek, így könnyebb lesz megjegyezni, mit hol találunk.



Az egyes blokkok puzzle-darabkákhoz hasonlítanak, és drag-and-drop technikával mozgathatjuk őket a munkaterületre. Ha valamire nincs szükség, kukázhatjuk. A gombbal futtathatjuk a programunkat.

A vízszintes menüsorban lévő egyéb programozási nyelvekre automatikus fordítást tud végezni a Blockly Code generátor, vagyis azok, akik szeretnének megismerkedni ezekkel a nyelvekkel, komoly segítséget kaphatnak hozzá ezen funkción keresztül.

A Blocklyval való munkához nem szükséges bejelentkeznünk sehová, munkánk pedig automatikusan mentésre kerül a Google felhőbe. Ahhoz, hogy hozzáférhessünk ehhez később, célszerű a képernyő jobb felső sarkában lévő jelre kattintva hozzáférést biztosító linket létrehozni, amit elmenthetünk magunknak.

Matematikai feladatmegoldás Blocklyval

Ismert tény, hogy az első számítógépeket nagy számítási igényű számítási feladatok megoldására alkalmazták. De ezen túl mi a kapcsolat a matematikai feladatmegoldás és a számítógép programozás között? A feladatmegoldás módszertana. Ahogyan a kisiskolás kortól kezdve tanuljuk megoldani az alkalmazói (szöveges) feladatokat, ugyanúgy oldjuk meg a feladatokat számítógéppel is, amikor programot írunk az adott problémára. A módszer Pólya György matematikus által meghatározott sok helyen máig alkalmazott módszertanon alapul.

A következő táblázatban összefoglaltuk ennek a módszertannak a lépéseit, és párhuzamot vontunk a matematika és a programozás egyes fázisai közt.

Analógiák

MATEMATIKA	PROGRAMOZÁS
Nem csak a számolás tanításáról szól	Nem csak a programkód írásáról szól
1. A feladat megértése, értelmezése	
2. Tervkészítés	
Ismeretlen (változók) azonosítása	Változók deklarálása
Matematikai modell (a megoldási terv)	Algoritmizálás és implementálás (a program terve)
3. A terv végrehajtása	
Megoldáskeresés	Fordítás és futtatás
Válaszadás a szöveges feladat kérdésére	
4. Diszkutálás (elemzés, visszatekintés)	
Kezdőfeltételek változtatása A feladat általánosítása Más megoldási módok keresése	Tesztelés Hibakeresés Általánosítás (kód újrahasznosíthatósága) Kódoptimalizálás

Egy fontos dolgot le kell szögeznünk, ahogy azt majd látjuk a példákon keresztül is: nem mindig a matematikailag legszebb megoldás az, ami legkönnyebben implementálható!

1. A feladat megértése, értelmezése

A szöveges feladat szövegének értelmezése, a szövegben szereplő releváns adatok azonosítása, szövegből való kiemelése, rejtett információk feltárása, nagyvonalú feltételrendszer meghatározása.

2. Tervkészítés

Meghatározzuk az ismeretlent, és azt az utat, ahogyan eljutunk a kiszámításához. Vagyis algoritmust készítünk és implementáljuk a kívánt programozási nyelven.

3. Terv végrehajtása

Esetünkben ez a futtatást jelenti, melynek során meg kell válaszolni a feladat kérdését.

4. Diszkutálás

A szöveges matematikai feladatok egy konkrét szituációt, konkrét kezdő adatokkal mutatnak be, amit meg kell oldanunk. A matematikában azonban fontos kérdés, hogy a talált megoldási terv milyen körülmények közt volt alkalmazható, milyen feltételek mellett lehet újra hasznosítani. Ezt nevezzük általánosításnak.

Egy számítógépes programot nem gazdaságos létrehozni, ha csak egy konkrét feladatot tud megoldani, és más feltételek fennállása esetén ugyanez a program nem alkalmazható. Ezért fontos a feladat elemzése, és a lehető legszélesebb körű általánosítása.

Az általánosításnak négy szintjét különböztetjük meg:

1. Általánosításra, diszkutálásra nem alkalmas a program, csak az adott feladatot oldja meg. Egyszerű példán bemutatva: induló adatok: 3, 2. Megoldandó feladat: x=3+2.
    - Változók: az adott feladatban szereplő konkrét értékkel deklarált változóink lesznek.
    - Programtörzs, megoldási algoritmus: a konkrét adatokkal dolgozik.
    - Válasz: Az adott feladatra ad választ.
2. Az általánosítás első szintjének tekintjük a probléma formalizálásaként említett szintet a piramisban. Minden adat kigyűjtésre kerül a program elején, olyan módon, hogy változó1=konstans1, változó2=konstans2, stb., és a program többi részében csak a változóneveken keresztül hivatkozunk ezekre a konstansokra. Általánosan még nem oldja meg a problémát a program, „kézi beavatkozás" segítségével viszont már könnyen módosíthatók az induló adatok. Példánk szerint: a=3, b=2, x=a+b.
    - Változók: konkrét értékkel deklarált változóink lesznek – amelyek „kézzel" (a programba való belenyúlással) módosíthatók.
    - Programtörzs, megoldási algoritmus: a változónevekkel dolgozik.
    - Válasz: A változónevekkel dolgozik, vagyis az induló adatok módosításával a program képes választ adni a módosított problémára.
3. Az általánosítás egy magasabb szintje az, amikor a program képes arra, hogy bekérjen adatokat, vagyis egyfajta „interaktív", kommunikációra képes alkalmazást hozunk létre. Példánk szerint: a=?, b=?, x=a+b.
    - Változók: nincsenek konkrét értékkel deklarált változók, adatokat kér be a program.
    - Programtörzs, megoldási algoritmus: a változónevekkel dolgozik.
    - Válasz: A változónevekkel dolgozik, vagyis diszkutálás esetén a program az aktuálisan bekért induló adatokkal oldja meg a problémát és arra ad választ.
4. Ennek a szintnek a következménye egy újabb, negyedik szint a további diszkutálás, aminek a bejövő adatok minőségének vizsgálata nevet adtuk. Matematikai szempontból a feladat szövegéből, valóság közeliségéből adódó, valamint a feladatmegoldás során előbukkanó „kikötésekből" álló feltételrendszernek a programba való beépítéséről van szó, illetve az ennek való megfelelés vizsgálatáról. Informatikai szempontból ezek szoftverminőségi kérdésekhez vezetnek.

1. Feladat: Bontsuk szét a 72-t négy részre úgy, hogy ha az első részhez 5-öt hozzáadunk, a második részből 5-öt kivonunk, a harmadik részt 5-tel megszorozzuk, a negyedik részt 5-tel elosztjuk, akkor mindig ugyanazt a számot kapjuk eredményül!

Matematikai megoldás:

Adatok: Alapszám: 72, az aktuális műveletet, amivel manipuláljuk a bontáskor keletkezett részeket, mindig 5-tel végezzük.

Ismeretlen: Az egyforma mennyiséget keressük, vagyis a részekkel és az 5-tel való műveletvégzés eredményét. Ez lesz az ismeretlen változónk, x.

Megoldási terv: Visszafelé gondolkodunk:

5. az első esetben 5-öt hozzáadtunk az első részhez, és így kaptuk meg x-et, vagyis az ellentétes műveletben gondolkodva az elő részt úgy kapjuk meg, ha x-ből kivonunk 5-öt: $x - 5$;
6. a második részt hasonlóképp gondolkodva úgy kapjuk meg, hogy az ismeretlen végeredményhez hozzáadunk 5-öt: $x + 5$;
7. a harmadik rész előállítása az ismeretlen végeredmény segítségével: $\frac{x}{5}$;
8. a negyedik részé: $x \cdot 5$.

Összeadva a négy részt 72-höz jutunk:

$$x - 5 \; + \; x + 5 \; + \; x/5 \; + \; x \cdot 5 \; = \; 72 \qquad (*)$$

Ezt rendezve:

$$2x + x/5 + 5x = 72$$

$$\frac{36}{5}x \; = \; 72$$

$$x \; = \frac{76}{\frac{36}{5}} = 10$$

Ebből a részek:

$$x - 5 = 5, \qquad x + 5 = 15, \qquad x/5 = 2, \qquad x \cdot 5 = 50.$$

Implementálás Blocklyban:

Amire szükségünk lesz: Matematikai műveletek, listakészítés.

Kulcskérdése a feladatnak, hogy ki tudjuk fejezni az ismeretlent. Láthatjuk a felírt (*) összefüggésből, hogy bármi is a szám egyszer hozzáadva, egyszer kivonva azt a baloldalon nem lesz konstans tag. Csak az ismeretlen valamilyen szorzóval. Ezzel

a szorzóval kellett csak leosztanunk, s ez jelenleg $\frac{36}{5}$ De vajon hogyan fejezzük ki ezt a szorzót, hogy általánosabban fel tudjuk írni a programot?

Ha a választott számot paraméternek tekintjük, jelöljük p-vel, akkor így nézne ki (*) összefüggés:

$$x + p + x - p + x \cdot p + \frac{x}{p} = 72$$

Ezt rendezve:

$$2x + x \cdot p + \frac{x}{p} = 72$$

$$(2 + p + \frac{1}{p})\, x = 72$$

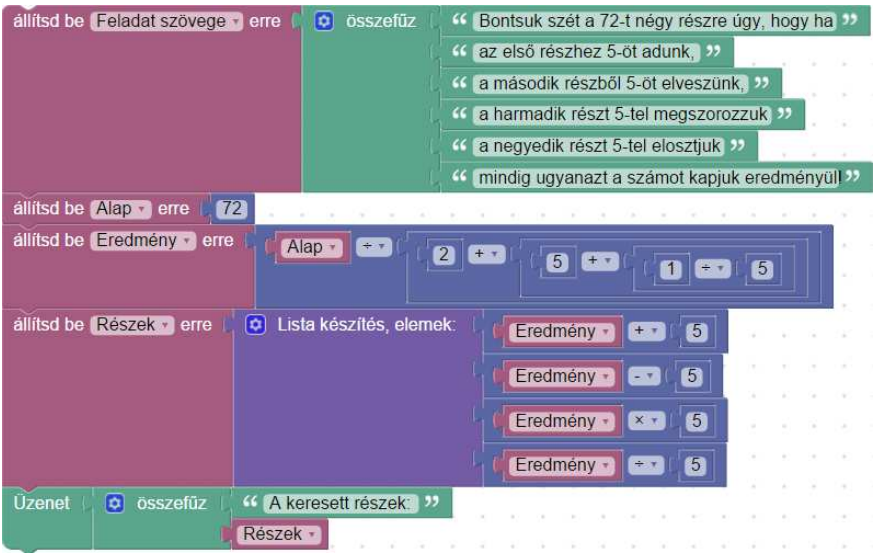
$$x = \frac{72}{(2 + p + \frac{1}{p})}$$

Az alapot mindig azzal a számmal kell leosztani, amit a nevezőben látunk, ahol p helyére aktuálisan beírjuk azt a számot, mellyel a négy alapműveleti manipulációt elvégezzük. A példánkban ez 5.

Máris kész a megoldásunk arra, hogy kicsit általánosabban implementálhassuk a feladatot.

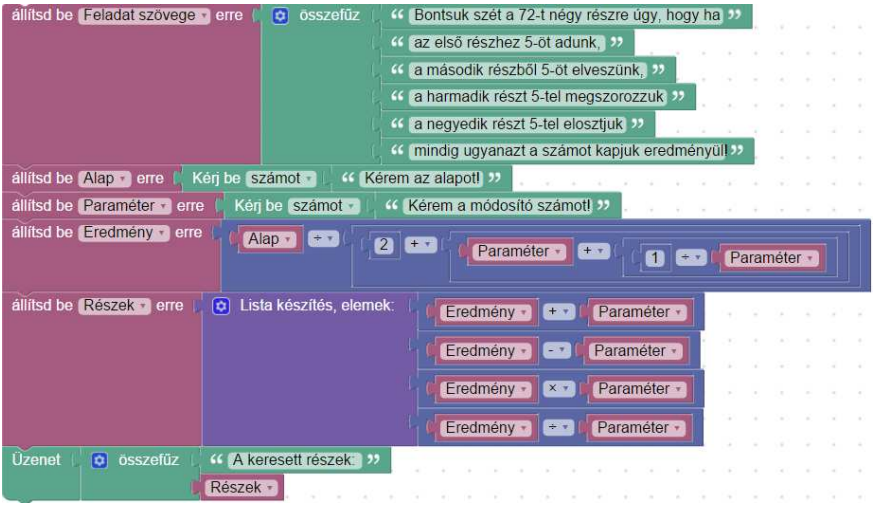
Első implementáció fix adatokkal:

https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hu#7pkucw



Az alapot, és azt a számot, mellyel manipuláljuk a végeredményt, be is kérethetjük, ekkor mindössze annyi módosítást kell végrehajtani, hogy számokat kérünk be a fix értékek helyett, és az 5-ös számot mindenhol lecseréljük egy új változóra, amit Paraméternek fogunk nevezni.

https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hu#u42gkp



Feladat: Dávid anyja 24 évnél idősebb, de 51 évnél fiatalabb. Ha összeadjuk annak az évszámnak a számjegyeit, amikor született, 22-t kapunk. Mikor született Dávid anyja?

Matematikai megoldás 1:

Ismeretlen: egy évszám

Szokásos alakja: x*1000+y*100+z*10+w, ahol x=1, y=9, 6≦z≦9, 0≦w≦9, mert Anya életkora 24 és 51 között van, vagyis ha a jelenlegi dátumból, 2015-ből indulunk ki, akkor születési éve 2015-24=1994 és 2015-51=1961 közé kell essen. Az első két szám tehát biztos, ismeretlen marad z és w, a fenti megszorításokkal.

Az évszám számjegyeinek összege: x+y+z+w=1+9+z+w=10+z+w=22. Ebből z+w=12.

Innentől próbálgatással a következőket kapjuk:

z=6 => w=6, ekkor Anya születési dátuma 1966, kora jelenleg 49 év; megfelel

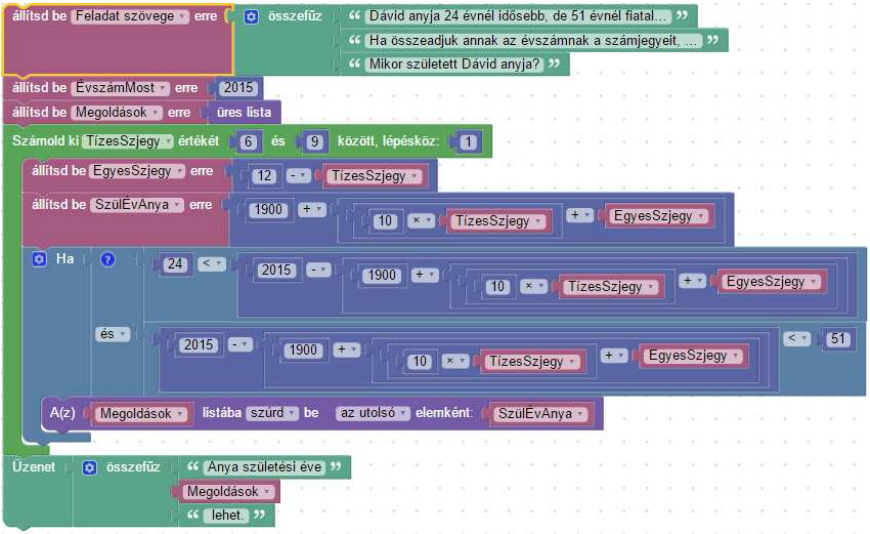
z=7 => w=5, ekkor Anya születési dátuma 1975, kora jelenleg 40 év; megfelel

z=8 => w=4, ekkor Anya születési dátuma 1984, kora jelenleg 31 év; megfelel

z=9 => w=3, ekkor Anya születési dátuma 1993, kora 22 év; ez már nem felel meg a kezdő feltételeknek, vagyis 3 jó megoldás marad.

Implementálás Blockly Code-ban:

https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hu#p99ag5



Matematikai megoldás 2:

Ismeretlen: egy évszám

Szokásos alakja: x*1000+y*100+z*10+w, ahol x=1, y=9, 0≦z, w≦9, ami bővebb feltételrendszer, ezért fontos lépés a feladat általánosításához.

Az évszám számjegyeinek összege: x+y+z+w=1+9+z+w=10+z+w=22.

Anya életkora 25-től 50-ig lehet valamilyen egész szám.

Innentől próbálgatással a következőket kapjuk:

Ha Anya 25 éves, akkor 2015-25=1990-ben született. 1+9+9+0=19, nem felel meg.

Ha Anya 26 éves, akkor 2015-26=1989-ben született. 1+9+8+9=27, nem felel meg.

…

Ha Anya 50 éves, akkor 2015-50=1965-ben született. 1+9+6+5=21, nem felel meg.

A felsorolásban természetesen benne lesz a három megfelelő megoldás.

Általánosan:

Életkor	aktuális év - életkor = = születési év	születési évszám számje-gyeinek összege = 22?	Megoldás
25	2015 - 25 = 1990	1+9+9+0 = 19	nem
26	2015 - 26 = 1989	1+9+8+9 = 27	nem
…	…	…	igen/nem
50	2015 - 50 = 1960	1+9+6+0 = 16	nem

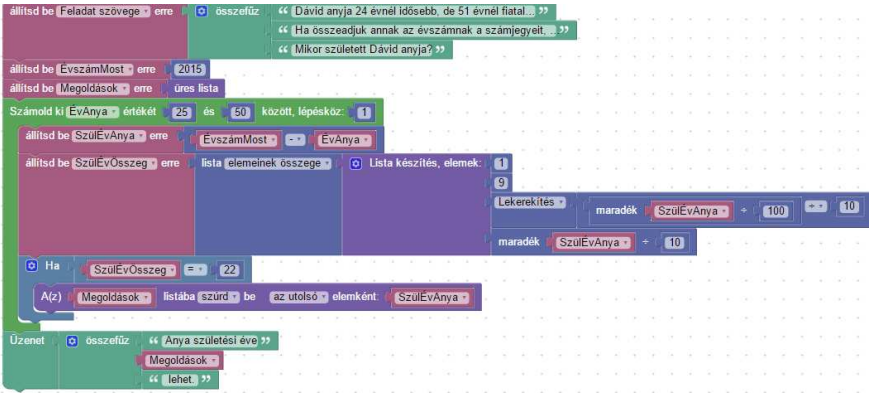
Implementálás Blockly Code-ban:

Kellenek: változók, ciklus, matematikai számítások, lista, feltétel vizsgálat.

A születési évszám számjegyekre bontását is meg kell matematikailag oldani az implementálás előtt, mert a gép nem olyan intelligens önmagától, hogy egy négyjegyű számot számjegyekre bontson! Ezt a következő módon tesszük meg:

- 1 és 9 biztosan szerepel az ezres és százas helyi értékeken. Ezek konstansok lesznek.
- A tízes és egyes helyi értékeken álló számjegyeket átnézve a Blockly matematikai lehetőségeit a következőképp kaphatjuk meg:
  A tízes helyi értéken álló számjegyhez vesszük az évszám 100-zal való osztásakor keletkezett maradékát, ezt elosztjuk 10-zel, és az így kapott számot lefelé kerekítjük a következő egész értékig. (Pl. 1967 esetében: 1967:100=19, maradék 67, 67:10=6,7, lefelé kerekítve 6)
- Az egyes helyi értéken álló számjegyet úgy kapjuk meg, ha 10-zel elosztjuk az évszámot és vesszük a maradékát. (Pl. 1967 esetében 1967:10=196, maradék 7)

https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hu#zn8d3x

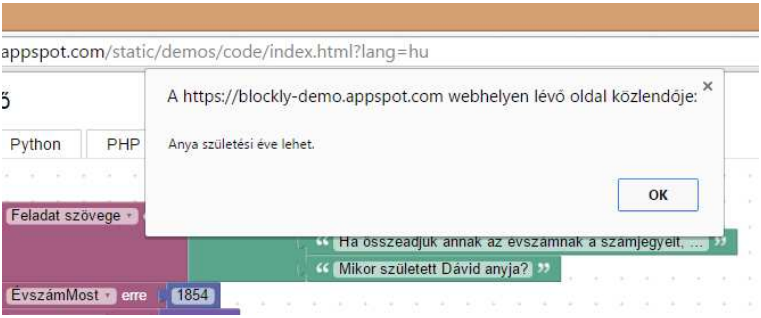


A megoldás:

További általánosítás, interaktívvá tétel

Vegyük észre, hogy bármilyen aktuális dátumot is írunk, a feladat mindig ugyanaz, nem befolyásolja a végeredményt. Próbáljuk ki például, mi történik, ha 2015 helyébe 1986-ot írunk!

Nem működik viszont a programunk, ha 1900 előtti dátumot írunk, hiszen a programba konstansként beírtuk az évszám első két számjegyeként, az 1 és 9 számokat.



Módosítsunk úgy a programon

- 19. századi dátumokra működjön;
- 21. századi dátumokra működjön;

A módosításhoz közvetlenül át kell írnunk bizonyos kezdő adatokat, melyek konstansként szerepelnek a programban.

A tesztelgetés során biztosan lesz olyan kezdő adat kombináció, melynél nem lesz a feltételeknek megfelelő évszám. Ilyen pl., ha 1854-es kezdő dátumunk van, Anya életkorának intervalluma 24 és 51 nyílt intervallum, és a programban átírjuk az évszám második számjegyét 8-ra. Oldjuk meg, hogy ilyen esetben a program írjon ki egy üzenetet, mi szerint nincs megoldása a feladatnak.

Ehhez miután a ciklus lefutott, meg kell vizsgálni a listát, van-e benne elem, vagy üres. Ha üres, akkor más üzenetet kell kiírjon, mint egyébként.

Haladjunk tovább az általánosítással! Most azt szeretnénk, hogy bármilyen négyjegyű szám által meghatározott dátumra működjön a program, vagyis a dátumban a százas helyi értéken lévő számot kell valahogyan visszanyernünk.

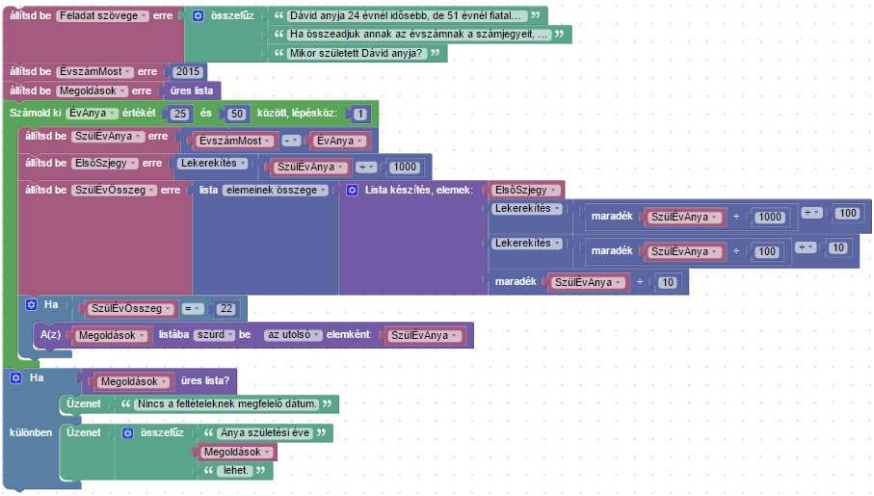
A százas helyi értéken álló számjegyhez vesszük az évszám 1000-rel való osztásakor keletkezett maradékát, ezt elosztjuk 100-zal, és az így kapott számot lefelé kerekítjük a következő egész értékig, teljesen hasonló módon működik, mint a tízeseknél.

Ez így még csak az 1-gyel kezdődő négyjegyű számokat kezeli.

Ahhoz, hogy kezeljen további négyjegyű számokat is, meg kell határoznunk az első számjegyet is, és azt írni az eddig konstansként beírt első számjegy helyére a számjegyeket tartalmazó listába.

Jelenleg a teljes programunk így néz ki:

https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hu#kmh5b9



A folyamatos általánosítás során tehát kiküszöböltük azt a sok matematikai megfontolást igénylő előzetes feltételhalmazt, mely az első matematikai megoldásban szerepel. Míg matematikai szempontból az a megoldás „elegánsabb”, addig a programozás szempontjából ez a feltételhalmaz túlságosan speciálissá teszi a feladatot, ami miatt nem oldható meg emberi intelligencia nélkül.

Végezetül oldjuk meg, hogy a program kommunikáljon velünk, vagyis tegyük alkalmassá a programot arra, hogy a felhasználó maga írjon be aktuális dátumot!

Ennek alapján oldjuk meg, hogy a felhasználó maga állíthassa be

- a születési év összegét,
- az alsó és felső korlátot Anya életkorára vonatkozóan!

3. feladat: Írjunk programot, mely a megadott számok közül kiválogatja azokat, amelyek igazzá teszik a 329+A>457 egyenlőtlenséget! A számok: 128, 113, 228, 129, 369, 215, 342, 127, 214, 217.

A feladat nem nagy kihívás matematikailag. Fogjuk az első számot, behelyettesítjük az egyenlőtlenségbe, ha igazzá tesszük, akkor megjegyezzük, ha nem, akkor „félredobjuk", vesszük a következőt, azzal is elvégezzük ugyanezt. 10 számunk van, ez 10 darab vizsgálat lesz. A végén kiírjuk a megfelelő számokat.

Ugyanezt fogjuk leprogramozni.

Amire szükségünk lesz: lista, ciklus, logikai vizsgálat.

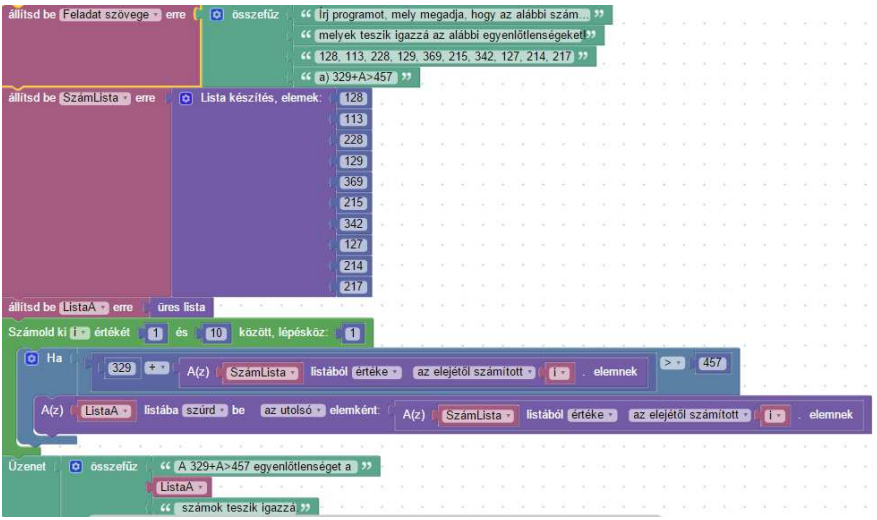
Először feltöltünk egy listát a 10 számmal, majd egy üres listát hozunk létre, melybe gyűjteni fogjuk a megfelelő számokat.

A ciklust arra használjuk, hogy valamennyi listaelemre 1-től 10-es sorszámúig (idegen szóval: index) meg tudjuk fogalmazni a vizsgálat lényegét.

Tehát ciklusváltozó lesz a sorszám (i), ahányadik elemmel az eredeti listában dolgozunk, és a ciklus magjában fogalmazzuk meg formálisan azt, amit ezzel az általános i. elemmel csinálunk. Ezt az elemet hozzá kell adni a 329-hez, és ha 457-nél nagyobb lesz a kapott összeg, akkor a másik, eddig üres listát elkezdjük feltölteni ezekkel a megfelelő elemekkel.

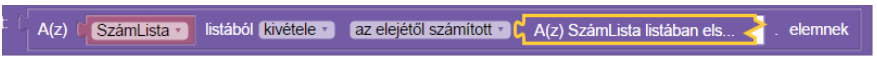
Implementáció Blocklyban:

https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hu#5n8h66



További szépítése lenne a feladatnak, ha a megoldást adó számok rendezve, például növekvő sorrendben lennének kiírva a képernyőre az üzenetben.

Vagy még a kiindulási listánkat kell rendezni, vagy a végeredményként kapott listát. Mivel a rendezési feladat klasszikus programozói feladatok közé tartozik, ezt a kódrészletet is bemutatjuk, a hely szűke miatt több részletben:



Ekkor az eredmény:

4. feladat: Egy kempingben, ahol csak 4 ágyas faházak vannak, 32 diák és 2 kísérő tanár rendel szállást. Egy faházban csak azonos neműek lakhatnak, valamint diák tanárral nem kerülhet össze. Legkevesebb hány faházat kell így fenntartani, hogy biztosan el tudják szállásolni őket?

Matematikai megoldás:

Adatok:

A keresett ismeretlen a faházak darabszáma.

Az ágyak száma faházanként 4.

A kísérő tanárok száma, 2, de nem tudjuk, azonos, vagy különböző neműek. Számukra 1 faházat szükséges fenntartani, amennyiben azonos neműek, 2-t, ha különbözőek.

A diákok száma 32, de itt sem tudjuk a nemek összetételét, holott hasonlóképp befolyással van a keresett ismeretlen értékének alakulására, mint a tanárok esetében.

Táblázatba foglalva a következőket kapjuk: (felkerekítésen a következő egész számig való kerekítést értjük)

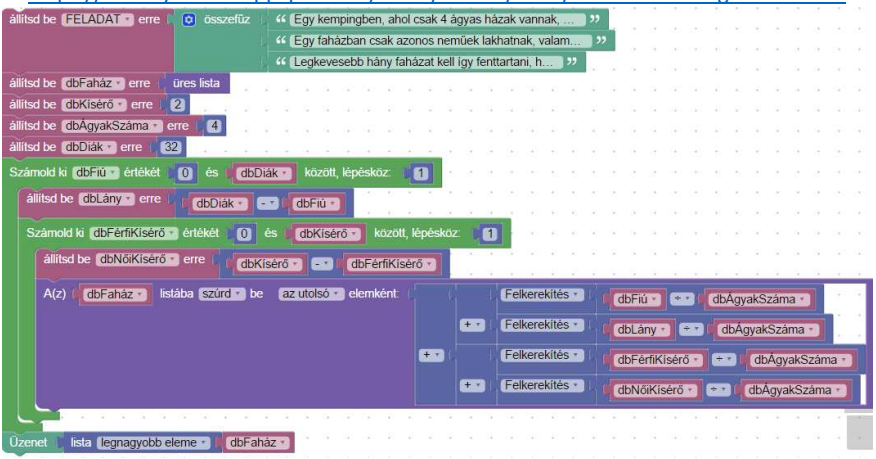
Fiúk	Lányok = Összes diák - Fiúk	Férfi tanár	Női Tanár = Összes tanár – Férfi tanár	Faház szám = felkerekít(Fiúk/4)+felkerekít(Lányok/4)+felkerekít(Ffi tanár/4)+felkerekít(Női tanár/4)
0	32	0	2	0+8+0+1=9
0	32	1	1	0+8+1+1=10
0	32	2	0	i0+8+1+0=9
1	31	0	2	1+8+0+1=10
1	31	1	1	1+8+1+1=11
1	31	2	0	1+8+1+0=10
2	30	0	2	1+8+0+1=10
2	30	1	1	1+8+1+1=11
2	30	2	0	1+8+1+0=10
3	29	0	2	1+8+0+1=10
3	29	1	1	1+8+1+1=11
3	29	2	0	1+8+1+0=10
…	…	…	…	…
32	0	0	2	8+0+0+1=9
32	0	1	1	8+0+1+1=10
32	0	2	0	8+0+1+0=9

Ha felírtuk a teljes táblázatot, csak ki kell választanunk az utolsó oszlopból a legnagyobb elemet. Ennyi faházat kell lefoglalni, hogy biztosan mindenkinek jusson hely a megadott feltételek mellett.

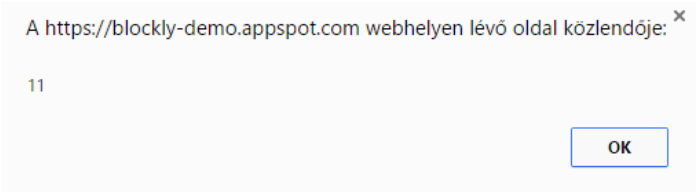
Implementáció Blocklyban:

Ugyanezt hajtjuk végre lista adatszerkezet és egymásba ágyazott ciklusok segítségével programban.

https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hu#f4abe2



A megoldás:



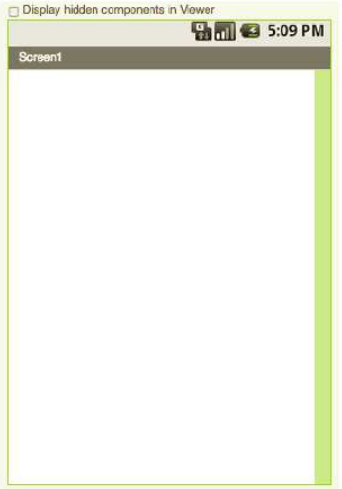
A matematikai megoldás során sok munkától megkíméljük magunkat, ha észrevesszük a szimmetriát a férfi és női kísérők és a fiú és lány diákok számát illetően is, de még így is hosszú lesz a táblázat. A program azonban pillanatok alatt végig tudja ugyanezt vizsgálni.

Mint látjuk, a program már általánosan, változókkal dolgozik az ágyak számát illetően is. További szépítése a programnak az interaktívvá tétel, vagyis ha bekér adatokat a felhasználótól, illetve vizsgálja is azokat, hogy csak pozitív egész számot és nullát fogadjon el a gyerekek vagy a kísérők számaként, különben olyan üzenetet küldjön a felhasználónak, hogy nem megoldható a feladat, mert nem életszerű a megadott szám, mint emberek száma. Ez utóbbi szépítése a programnak egyébként a legtöbb darabszámmal dolgozó feladat esetében kívánatos.

Az Appinventor^2 nyelvvel Android platformú mobil eszközökre készíthetünk alkalmazásokat. A programozási környezet két fontos részre tagolódik. Az egyik a képernyőtervező, a másik a blokkszerkesztő.

A képernyőtervezőn egyszerűen össze tudjuk állítani azokat a látványelemeket illetve vezérlőket, melyeknek programozása révén az alkalmazás az eltervezett módon működni fog. Ezt drag-and-drop technikával, fájlfeltöltéssel, stb. tehetjük meg. Az alkalmazásunk képernyőképeit, fontos szereplőit, illetve azok tulajdonságait módosító vezérlőelemeket, érzékelőket tudunk itt meghívni.

Állítsunk be saját színeket a képernyőn a Make Color blokk segítségével!

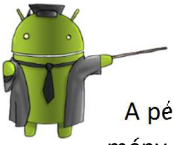


Designer elemek:

Komponens	Csoport	A komponens célja
-	-	-

Blokkszerkesztő:

Mit jelent ez a kód?

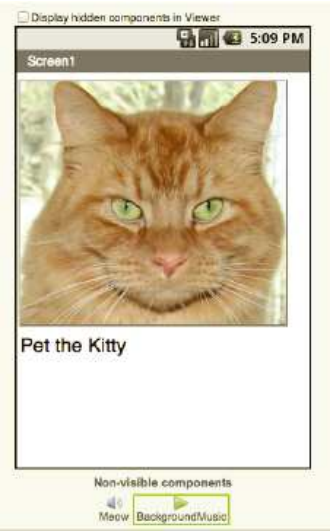


A make color számára egy három számból álló listát kell átadni. A három szám a beállítani kívánt szín RGB kódja.

A példában a Purple (lila) 157, 57 és 252 áll. A Screen1.Initialize esemény meghívásra kerül amikor elindítjuk az alkalmazást, a háttérszín az a szín lesz, amit megalkottunk: a példában ez lila.

Hangfájlok hozzáadása

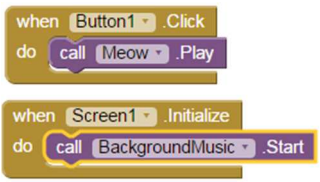
Adjunk rövid hangeffektet, például egy csattanás hangot, amikor két sprite ütközik, vagy hosszabb hangfájlt, például háttérzenét a projektünkhöz!



Designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Button	User Interface	Button1	Érintés esemény kezelése	Image: png,jpg,gif
Sound	Media	Meow	Hangeffekt lejátszáshoz	Source: mp3,wav
Music	Media	BackrgoundMusic	Háttérzene lejátszáshoz	Source: mp3,wav

Blokkszerkesztő:



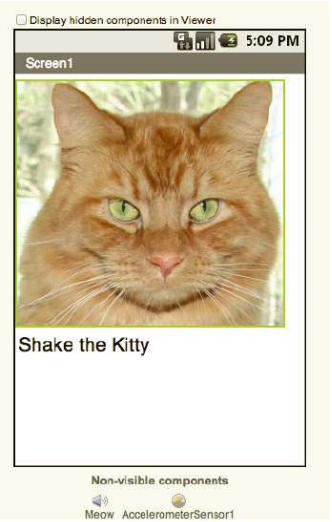
Mit jelent ez a kód?

Amikor a gombot megnyomjuk Button1.Click, a hangeffekt lejátszásra kerül Meow.Play.

Amikor az alkalmazás elindul a mobilon Sreen1.Initialize, akkor a beállított zene elindul BackgroundMusic.Start.

Telefon rázása

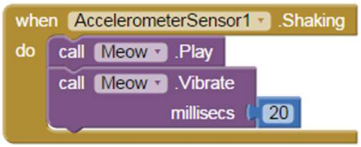
Történjen valami, amikor megrázzuk a telefonunkat!



Designer elemek:

Komponens	Csoport	A komponens célja
Image	User Interface	A macska képéhez
Label	User Interface	szöveg képernyőre írásához
Sound	Media	Hangeffekt lejátszáshoz
AccelerometerSensor	Sensors	A telefon rázásának érzékeléséhez

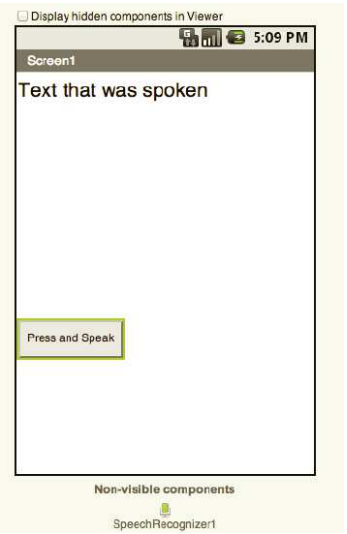
Blokkszerkesztő:



Mit jelent ez a kód?

Az AccelerometerSensor.Shaking esemény észlelésekor (amikor a használó rázza a telefonját), a Meow nevű hangfájl lejátszásra kerül Meow.Play és a telefon vibrálni fog Meow.Vibrate 20 milliszekundumig.

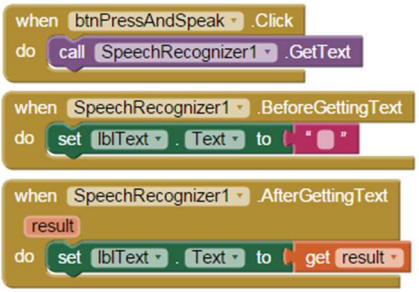
Írassuk ki a képernyőre a mondott szöveget!



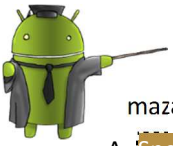
Designer elemek:

Komponens	Csoport	A komponens célja
Label	User Interface	A szöveg kiírásához
Button	User Interface	A beszédfelismerés indításához
SpeechRecognizer	Media	A beszéd feldolgozásához

Blokkszerkesztő:



Mit jelent ez a kód?



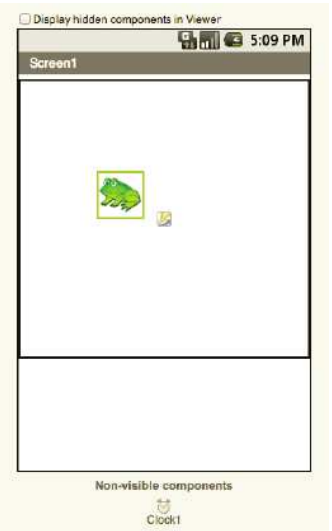
Amikor a gombot megérintjük btnPressAndSpek.Click, a SpeechRecognizer.GetText eljárás meghívásra kerül és az alkalmazás kész a mondott szöveg fogadására.

A SpeechRecognizer.BeforeGettingText esemény azelőtt következik be, mielőtt a szöveget az alkalmazás fogadná és felismerné. Ekkor ürítjük a labelt.

A SpeechRecognizer.AfterGettingText esemény pedig akkor, amikor az alkalmazás a szöveget fogadta és felismerte. Ekkor kiírja a szöveget a labelen a képernyőre.

Véletlenszerű Sprite elhelyezés

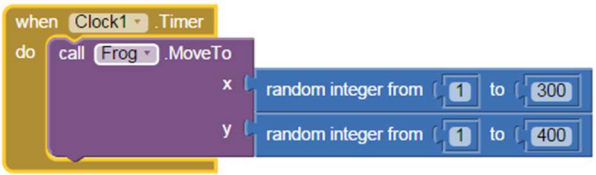
Generáljunk véletlenszerűen számokat, melyekkel, mint (x,y) koordinátákkal Sprite-okat tudunk pozícionálni a képernyőn!



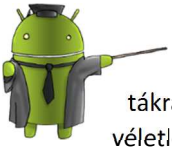
Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a figura elhelyezéséhez
ImageSripte	Drawing and Animation	Figura amit elhelyezünk
Clock	Sensors	Időzítő az elhelyezéshez

Blokkszerkesztő:



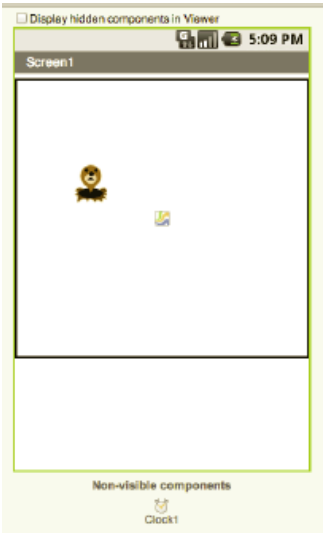
Mit jelent ez a kód?



Amikor a Clock1.Timer esemény aktiválódik, a Frog.MoveTo áthelyezi a béka sprite-ot a véletlen számként generált koordinátákra. A példa szerint x értéke 1 és 300, míg y értéke 1 és 400 közötti véletlen egész szám lehet.

Hogyan tudnánk ezt hasznosítani egy játékprogramban?

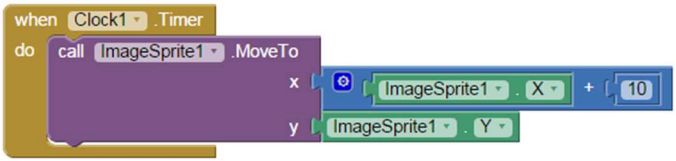
Megadott időközönként mozgassuk el a Sprite-ot!



Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a figura mozgatásához
ImageSripte	Drawing and Animation	Figura amit mozgatunk
Clock	Sensors	Időzítő a mozgatáshoz

Blokkszerkesztő:



Mit jelent ez a kód?

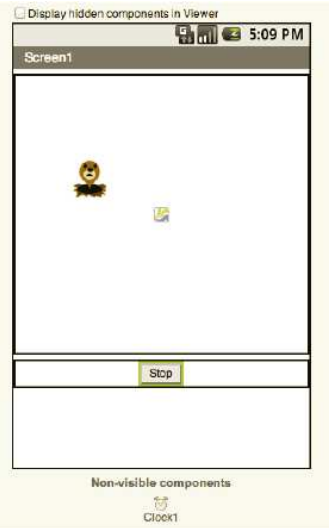
ImageSprite1.MoveTo a figuránkat mozgatja a vászon megadott helyére, a példában a aktuális helyzeténél 10 pixellel jobbra.

Clock1 vezérlőben beállított időközönként a Clock1.Timer esemény *meghívásra* kerül és lefut a beágyazott programkód.

Minden alkalommal, amikor a Clock1.Timer esemény kiváltódik, a Sprite jobbra mozdul (a Sprite x koordinátája változik) 10 pixelt az aktuális helyzetéhez képest.

Sprite-ok leállítása időzítővel

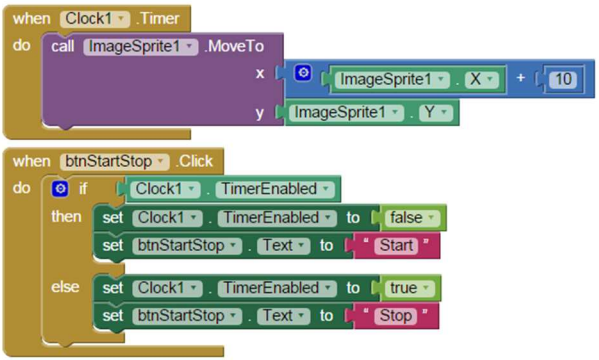
Azt szeretnénk, ha az előző példában szereplő időzítőt leállíthatnánk, amikor mi akarjuk egy Start/Stop gombbal.



Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a figura mozgatásához
ImageSripte	Drawing and Animation	Figura amit mozgatunk
Clock	Sensors	Időzítő a mozgatáshoz
Button	User Interface	Az időzitő ki/be kapcsolásához

Blokkszerkesztő:



Mit jelent ez a kód?

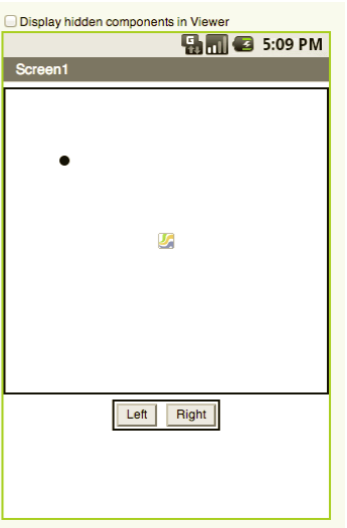
btnStartStop.Click gombot megérintjük, és az időzítő működik, akkor leáll az időzítő és megjelenik a start gomb.

Az ellenkezője történik, ha az időzítő épp nincs működésben.

Próbáljuk meg ellentétesre változtatni az objektum mozgásának irányát minden alkalommal, amikor a Start feliratú gombra klikkelünk!

Mozgatás gombokkal

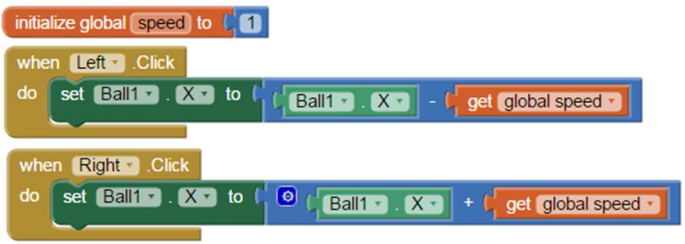
Mozgassunk egy labdát gombokkal!



Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a labda mozgatásához
Ball	Drawing and Animation	Labda amit mozgatunk
Button	User Interface	A jobbra/balra mozgatáshoz

Blokkszerkesztő:



Mit jelent ez a kód?

Adjunk a Speed változónak 1 kezdő értéket, amivel elmozdul minden alkalommal, ha megérintjük a gombot.

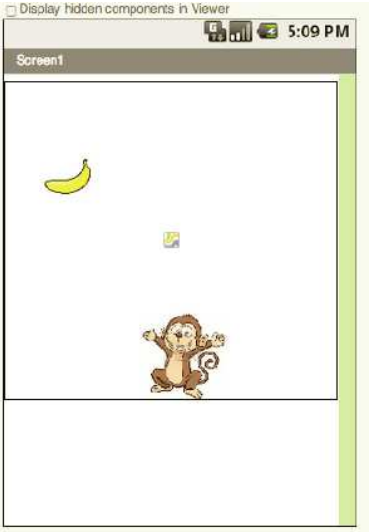
A Left.Click esemény minden alkalommal balra mozdítja a labdát, amikor megérintjük a gombot.

A Right.Click esemény minden alkalommal jobbra mozdítja a labdát, amikor megérintjük a gombot.

Adjunk további gombokat az alkalmazáshoz, amelyek fel illetve le mozgatják a labdát!

Sprite mozgatása ujjal

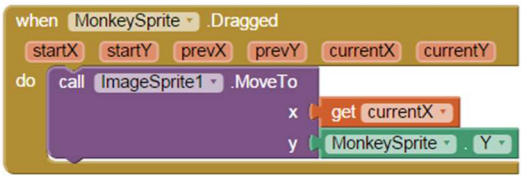
Mozgassuk a sprite-ot egyik oldaltól a másikig vízszintesen az ujjunkkal!



Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a figura mozgatásához
ImageSripte	Drawing and Animation	Figura amit mozgatunk

Blokkszerkesztő:



Mit jelent ez a kód?

Adott sprite esetén, az ujjunk húzásával folyamatosan kiváltódik a spritehoz tartozó dragged esemény, a példa szerint a MonkeySprite.Dragged.

Minden esetben az eseményben megkapjuk az alábbi 6 paraméter értékét:

- startX, startY: a sprite azon pozíciója, ahol megérintettük a képernyőn
- currentX, currentY: a sprite aktuális pozíciója a képernyőn
- prevX, prevY: a sprite előző pozíciója a képernyőn (az esemény előző kiváltásakor kapott currentX, currentY).

A példában a MonkeySprite-ot vízszintesen húzva az csak az x koordináta mentén mozog, y koordinátája ugyanaz marad.

Változtassuk meg egy sprite haladási irányát és sebességét suhintásra!

Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a figura mozgatásához
ImageSripte	Drawing and Animation	Figura amit mozgatunk

Blokkszerkesztő:

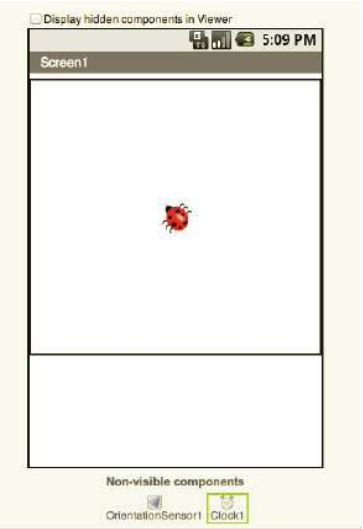
Mit jelent ez a kód?

A PirateSprite.Flung esemény akkor váltódik ki, amikor a telefonhasználó suhintó mozdulatot tesz a sprite-tal a képernyőn.

A használó ezzel beállítja a kalózhajó haladási irányát: PirateSprite.Heading és sebességét: PirateSprite.Speed.

Szenzoros mozgatás

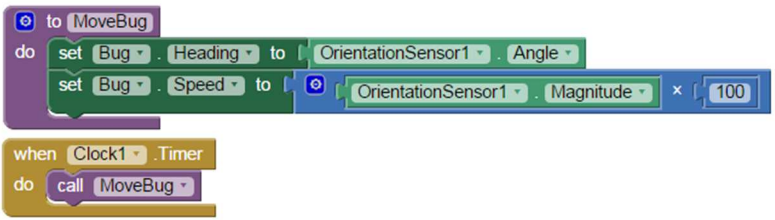
Mozgassunk egy sprite-ot a telefon billegtetésével!



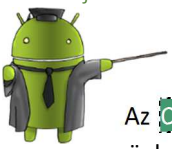
Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a figura mozgatásához
ImageSripte	Drawing and Animation	Figura amit mozgatunk
OrientationSensor	Sensors	Az irány és sebesség meghatározásához
Clock	Sensors	Időzítő a mozgatáshoz

Blokkszerkesztő:



Mit jelent ez a kód?



A MoveBug eljárás során a katica sprite olyan irányba fordul, amilyen irányba a telefont billentjük.

Az OrientationSensor1.Angle mutatja meg a katicának, hogy milyen szögben, a OrientationSensor1.Magnitude pedig, hogy milyen gyorsan kell elmozdulnia annak alapján, hogy hányszor billegtettük a telefont.

A MoveBug eljárás a Clock1.Timer időzítésre aktiválódik.

Sprite-ok ütközése

Történjen valami, ha egyik sprite ütközik egy másikkal!

Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a figura mozgatásához
ImageSripte	Drawing and Animation	Figura amit mozgatunk
Button	User Interface	Gombok a katica mozgatásához

Blokkszerkesztő:

Mit jelent ez a kód?

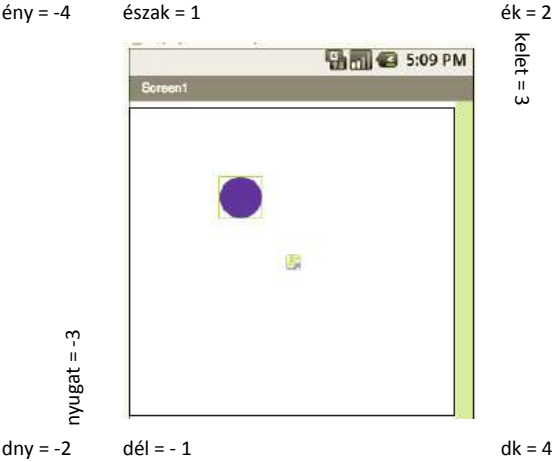
A LadyBug.CollidedWith esemény akkor aktiválódik, amikor a Katica érintkezik a levéltetűvel. Ennek eredménye, hogy a levéltetű eltűnik.

Amennyiben pontosabban akarjuk tudni, hogy mivel ütközött, akkor az other paraméter értékét kell vizsgálnunk.

- Tudnánk több levéltetűt játékba hozni?
- Adjunk hangeffektet ahhoz a részhez, amikor a katica megeszi a levéltetűt!
- A gombokkal való mozgatásról egy másik kártyán olvashatunk.

Készítsünk egy Labda alakú Sprite-ot, amely visszapattan a Canvas széléről!

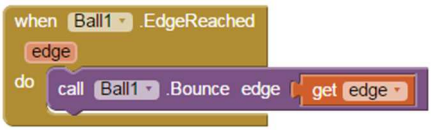
Az szélekhez hozzárendelt értékek (1,2,3,4,-1,-2,-3,-4).



Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a labda pattanásához
Ball	Drawing and Animation	Labda ami pattog

Blokkszerkesztő:



Mit jelent ez a kód?

A Canvas vezérlő minden széle (edge) rendelkezik saját numerikus értékkel.

Amikor a Ball sprite a vászon szélével ütközik, a Ball1.EdgeReached eseményben (edge) lokális változóként megkapjuk, hogy melyik szélével ütközött a sprite. (Visszakapjuk a numerikus értékét.)

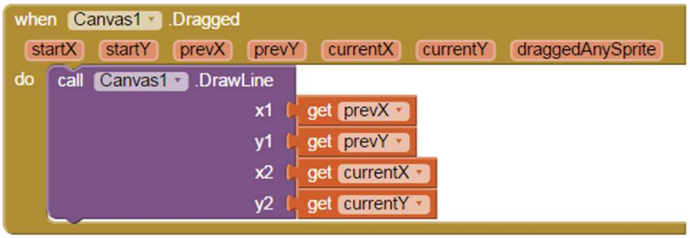
Ezt az értéket adva a Bounce (visszapattanás) eljárás aktuális paramétereként Ball1.Bounce azt okozza, hogy a labda a szélre állított merőlegesre, vagy a sarok szögfelezőjére tükrözve visszapattan.

A képernyőn az ujjunk húzásával rajzoljunk görbéket!

Designer elemek:

Komponens	Csoport	A komponens célja
Canvas	Drawing and Animation	Vászon a rajzoláshoz
Label	User Interface	Felirat a rajzolás módjáról
Button	User Interface	Gomb a vászon törléséhez

Blokkszerkesztő:



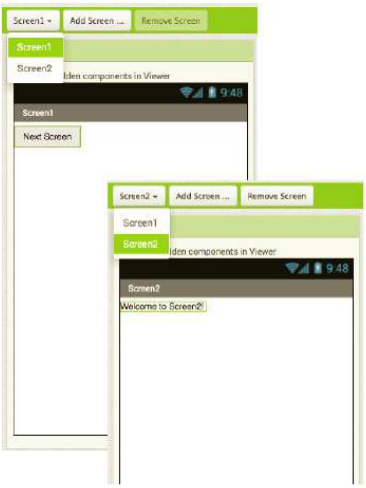
Mit jelent ez a kód?

A Canvas1.Dragged esemény észlelésekor görbe vonalat húz a képernyőre onnantól kezdve, ahol a felhasználó először érintette a képernyőt, egész addig, amíg felemelés nélkül a képernyőn húzza az ujját.

Ha a gombot megnyomjuk btnClear.Click, a vászon törlődik Canvas1.Clear.

Több képernyő kezelése

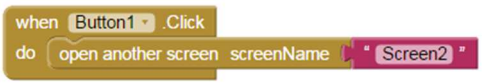
Használjunk több képernyőt az alkalmazásunkban! A második képernyő egy gomb megnyomására aktiválódjon!



Designer elemek:

Komponens	Csoport	A komponens célja
Screen	-	Második képernyő
Label	User Interface	Felirat a második képernyőn
Button	User Interface	Gomb az első képernyőn

Blokkszerkesztő:



Mit jelent ez a kód?

Az open another screen sreenName–hez egy szöveg blokk illeszthető, melyben egy másik képernyő neve szerepel. A Button1 gomb megnyomásával Button1.Click tudjuk ezt a másik képernyőt megnyitni.

Hogyan tudnánk visszatérni a kezdő képernyőre?

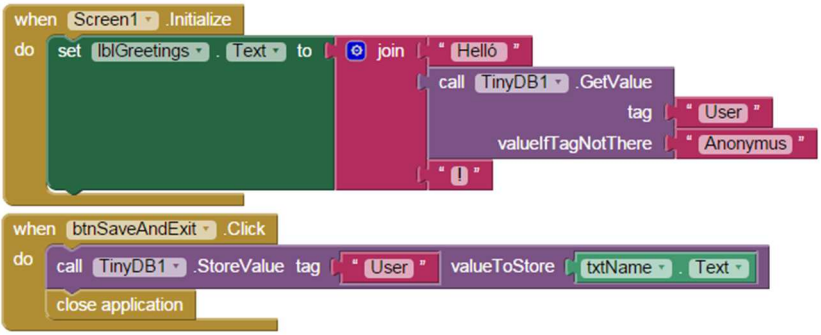
Emlékezz rám

Készítsünk alkalmazást, amelyik emlékszik ránk és legközelebb üdvözöl minket!

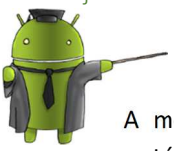
Designer elemek:

Komponens	Csoport	A komponens célja
Label	User Interface	Üdvözlés megjelenítéséhez
TextBox	User Interface	Szövegdoboz a név beírásához
Button	User Interface	Gomb a mentéshez és kilépéshez
TinyDB	Storage	A név tárolásához és olvasásához

Blokkszerkesztő:



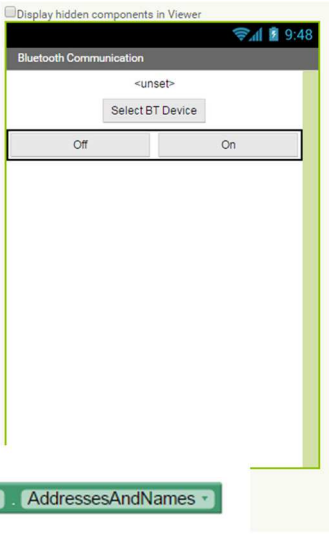
Mit jelent ez a kód?



Amikor az alkalmazás elindul Screen1.Initialize, az üdvözléshez beolvassuk a „User” tag értékét TinyDB.GetValue.

A mentés és kilépés gomb megérintésekor btnSaveAndExit.Click mentésre kerül a nevet tartalmazó szövegdoboz értéke TinyDB.StoreValue és kilépünk close application.

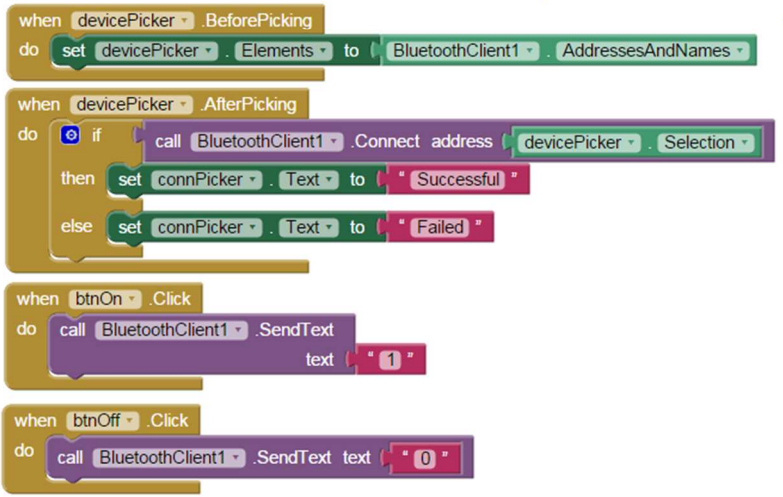
Csatlakozzunk bluetooth-on keresztül egy másik eszközhöz és küldjünk át neki 0-át, vagy 1-et



Designer elemek:

Komponens	Csoport	A komponens célja
Label	User Interface	Csatlakozási információ megjelenítése
Listpicker	User Interface	Lista a BT eszközökhöz
Button	User Interface	Gomb az adatküldéshez
BluetoothClient	Connectivity	Bluetooth kapcsolat létrehozásához

Blokkszerkesztő:



Mit jelent ez a kód?

Amikor a felhasználó megérinti a bluetooth eszköz kiválasztását devicePicker.BeforePicking, feltöltjük a párosított bluetooth eszközök listáját.

A felhasználó választ egyet a párosított eszközökből, a választása után devicePicker.AfterPicking megpróbálunk csatlakozni BluetoothClient.Connect. A kapcsolódás sikerességéről tájékoztatjuk a felhasználót.

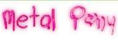
A MetalPony motivációs ereje a „léggitározás” szimulálása a mobilunkon. Az alkalmazás magja mindössze 20 perc alatt elkészíthető.

A megvalósítandó funkciók:

- Üdvözlő képernyő létrehozása
- Játékképernyő megnyitása
- A pengető ujjunkkal mozgatására zene lejátszása

Először hozzuk létre az üdvözlő képernyőt.

Designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Screen1	-	-	Elsődleges képernyő	AlignHorizontal: Center AppName: MetalPony BackgroundColor: Pink Icon:  ScreenOrientation: Portrait Scrollable: - Title: -
Image	User Interface	imgLogo	Metal Pony felirat	Picture: 
Image	User Interface	imgRights	Készítő neve	Picture: 
Image	User Interface	-	Automata távtartó	Height: Fill parent
Button	User Interface	btnStart	Játék indításához	Picture: 
Image	User Interface	-	Automata távtartó	Height: Fill parent
Button	User Interface	btnExit	Kilépéshez	Picture: 
Image	User Interface	-	Automata távtartó	Height: Fill parent

Blokkszerkesztő:

Szabályos kilépés az alkalmazásból:

Elsőként használjuk a btnExit.Click eseményét az alkalmazás szabályosan történő leállításához.

Majd hozzuk létre a játékképernyő megnyitását kezelő btnStart.Click eseményt.

Most pedig kezdődhet a kaland és a „léggitározás” megvalósítása.

A pink pengető megérintésekor kezdjen el zenét lejátszani, illetve a pengető kövesse az ujjunkat.

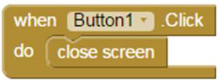
Designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Screen	-	Screen2	Játékképernyő	AlignHorizontal: Center BackgroundColor: Dark Gray ScreenOrientation: Portrait Scrollable: -
Canvas	Drawing and Animation	-	Vászon a „léggitározáshoz”	BackgroundColor: none Height: Fill parent Width: Fill parent
Ball	Drawing and Animation	-	Ez lesz a pengető	PaintColor: Magenta Radius: 7 X: 80 Y: 180
Button	User Interface	-	Visszalépés az üdvözlő képernyőre	Image: 
Player	Media	-	A zene lejátszásához	Loop: ✔ Source: WhiskeyInTheJar.mp3

Blokkszerkesztő:

Visszalépés az üdvözlő képernyőre:

Használjuk a Button1.Click eseményét az előző képernyőre való visszatéréshez.



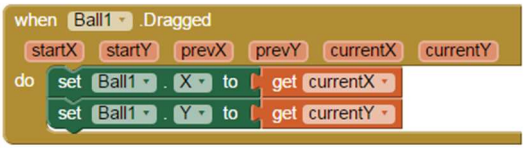
Léggitározás:

Most egy kicsit csalni fogunk, ugyanis nem függ össze a „pengető” mozgatása és a zene lejátszása, ez csak a „pengető” megérintésétől függ.

Amikor megérintjük a „pengetőt” Ball1.TouchDown, akkor induljon el a zene lejátszása Player1.Start.

Amikor elengedjük a „pengetőt” Ball1.TouchUp, akkor megállítjuk a zene lejátszását Player1.Pause.

A harmadik eseménnyel pedig a „pengetőt” mozgatjuk Ball1.Dragged, követve az ujjunkat.



Ezzel készen is van a működő „léggitározós” alkalmazás.

A látvány miatt érdemes vadul „léggitároznunk” az ujjunkkal, hiába lenne elég megérinteni a „pengetőt” Ball1.TouchDown a zene lejátszásához.

Fejlesszük tovább az alkalmazásunkat

Csak akkor játszon le zenét, ha a pengetőt mozgatjuk a vásznon, amennyiben csak mozdulatlanul tartjuk, vagy elengedjük, a zene álljon meg.

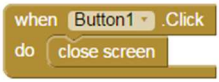
Designer elemek:

Nincs szükség további vezérlőkre.

Blokkszerkesztő:

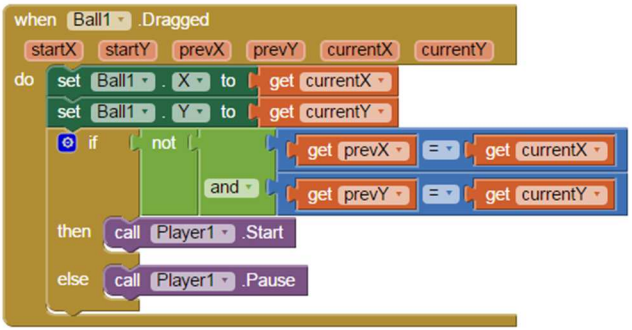
Visszalépés az üdvözlő képernyőre:

Használjuk a Button1.Click eseményét az előző képernyőre való visszatéréshez.



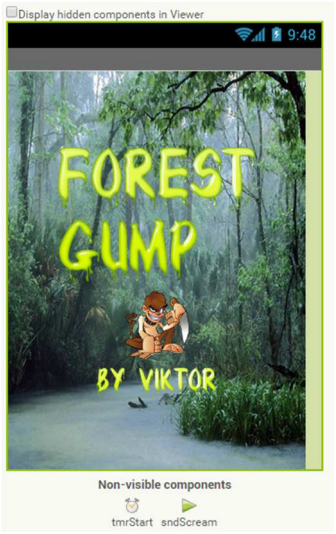
Léggitározás:

Amikor a „pengetőt” mozgatjuk Ball1.Dragged, kövesse az ujjunkat és vizsgáljuk meg, hogy változik-e a „pengető” x, vagy y koordinátája. Amennyiben változik, elindítjuk a zenét Player1.Start, amikor nem mozdítjuk el, akkor megállítjuk Player1.Pause.



Amikor elengedjük a „pengetőt” Ball1.TpouchUp, akkor is megállítjuk a zene lejátszását Player1.Pause.

Készítsünk játékot, amelyben egy majommal kell lehulló banánokat elkapni egy őserdőben.



A játékban 4 képernyőt fogunk létrehozni.

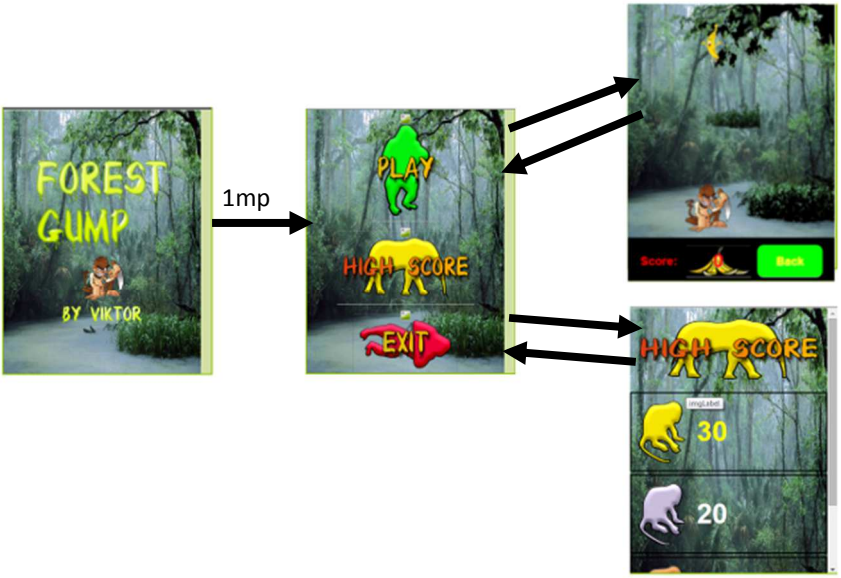
- Üdvözlő képernyő
- Menü
- Játéktér
- Eredmények

A megvalósítandó funkciók:

- Üdvözlő képernyő után automatikusan induljon a menü
- A banánok véletlenszerű helyről hulljanak alá.
- A majmot mozgathassuk vízszintesen az ujjunkkal.
- Pontszámolás és eredménytábla karbantartása (legjobb 3 eredmény).

Képernyőkezelés

A képernyők megnyitási é bezárási láncolata fontos komolyabb alkalmazások esetén.



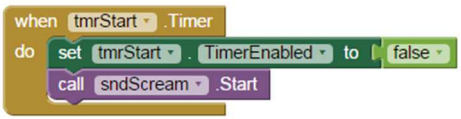
Hozzuk létre az üdvözlő képernyőt!

Designer elemek:

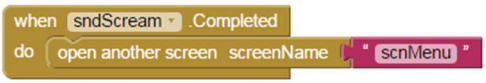
Kompo-nens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Screen1	-	-	Üdvözlő képernyő	AlignHorizontal: Center AlignVertical: Center AppName: ForestGump BackgroundColor: None BackgorundImage:  CloseScreenAnimation:SlideHorizontal Icon:  ScreenOrientation: Portrait Scrollable: - Title: -
Image	User Interface	imgLogo	Forest Gump felirat	Picture: 
Image	User Interface	imgMonkey	A főszereplő	Picture: 
Image	User Interface	imgAuthor	Készítő neve	Image:
Timer	Sensors	tmrStart	Késleltetett prog-ramindításhoz	TimerInterval: 1000
Player	Media	sndScream	Kezdeti hangulat megteremtéséhez	Source: Scream.mp3 volume: 80

Blokkszerkesztő:

Az időzítő tmrStart.Timer eseménye a beállított 1 másodpercenként aktiválódik, ezért az első aktiválódáskor állítsuk le és indítsuk el a sikoltás lejátszását sndScream.Start.



Amint a sikoltás befejeződik sndScream.Completed, jelenítsük meg a menü képernyőt open another screen.



Készítsünk menüt!

A menü képernyő minden más képernyő visszatérési pontja lesz. Minden képernyőt egyszer érdemes megnyitni, itt gyűjtjük össze a program funkcionalitását és nyitunk meg minden más képernyőt.

Designer elemek:

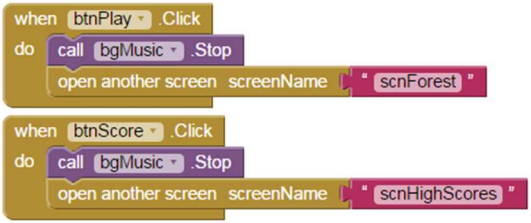
Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Screen	-	scnMenu	Menü képernyő	AlignHorizontal: Center AlignVertical: Center AppName: ForestGump BackgroundColor: None BackgorundImage:  CloseScreenAnimation:SlideHorizontal OpenScreenAnimation:SlideHorizontal ScreenOrientation: Portrait Scrollable: - Title: -
Image	User Interface	imgGap1	Automata távtartó	Height: Fill parent
Button	User Interface	btnPlay	Játéktér indítása	Image: 
Image	User Interface	imgGap2	Automata távtartó	Height: Fill parent
	User Interface	btnScore	Eredménytábla	Image: 
Image	User Interface	imgGap3	Automata távtartó	Height: Fill parent
	User Interface	btnExit	Kilépéshez	Image: 
Image	User Interface	imgGap4	Automata távtartó	Height: Fill parent
Player	Media	bgMusic	Háttérzene indítása	Loop: ✔ Source: rainforest.mp3 volume: 50

Blokkszerkesztő:

Elsőként használjuk a ontExit.Click eseményét az alkalmazás szabályosan történő leállításához.

A képernyő elindításakor scnMenu.Initialize elindítjuk a játék háttérzenéjét bgMusic.Start.

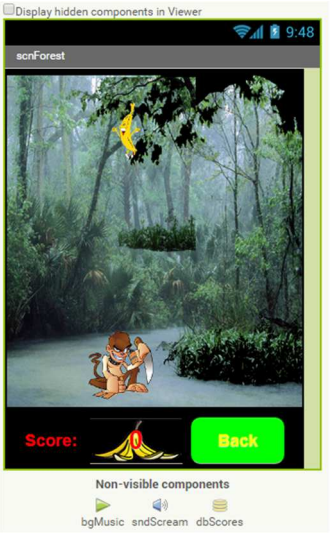
A játéktér btnPlay.Click és az eredménytábla btnScore.Click megnyitásakor először állítsuk le a háttérzenét bgMusic.Stop majd utána nyissuk meg a képernyőket open another screen.



Kapjuk el a banánt!

Mozgassuk a majmot a véletlenszerűen hulló banán alá, hogy elkapjuk. A képernyőn „réteges hátteret” alkalmazunk, hogy érdekesebbé tegyük a játék kinézetét. Ehhez képszerkesztővel két részre lett bontva a háttér. A majom és a banán ezen két réteg között mozog.

A játékos 1 pontot szerez minden banán elkapásával és 2 pontot veszít, ha leesik a földre. A 3 legjobb pontot az alkalmazás adatterületén tároljuk.



A játéktér előkészítése

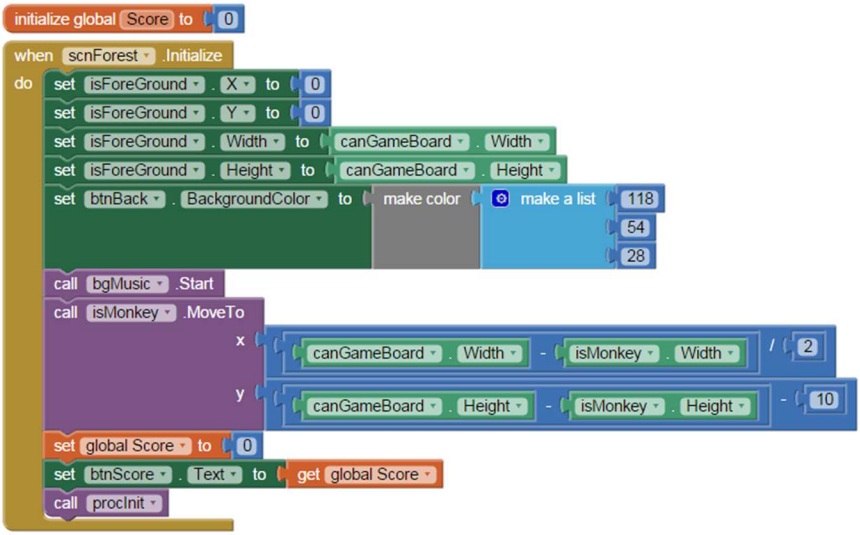
Designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Screen	-	scnForest	Játék képernyő	BackgroundColor: Black CloseScreenAnimation:SlideHorizontal OpenScreenAnimation:SlideHorizontal ScreenOrientation: Portrait Scrollable: - Title: -
Canvas	Drawing and Animation	canGameBoard	Játéktér	BackgroundColor: None BackgorundImage:  Height: Fill parent Width: Fill parent
ImageSprite	Drawing and Animation	isBanana	A hulló banán	Picture:  Z: 1.0
ImageSprite	Drawing and Animation	isForeGround	Réteg a háttér-hez	Height: 200 Width: 200 Picture:  Z: 5.0
ImageSprite	Drawing and Animation	isMonkey	A főszereplő	Picture:  Z: 1.0
Horizontal Alignment	Layout	haMenu	Vízszintes menü	Width: Fill parent
Label	User Interface	lblScore	Felirat	Text: „Score:”
Button	User Interface	btnScore	Pont kijelzéshez (kép és szám)	Fontsize: 30 Height: 50 pixels Width: 100 pixels Image:  Text: „0” textColor: Red
Button	User Interface	btnBack	Vissza a menü-höz	BackgroundColor: Green Height: 50 pixels Width: 100 pixels Text: „Back” textColor: Yellow
Player	Media	bgMusic	Háttérzene	Loop: ✔ Source: rainforest.mp3 volume: 50
Sound	Media	sndScream	Sikoly effekt	Scream.mp3
TinyDB	Storage	dbScores	Eredmény tárolás	-

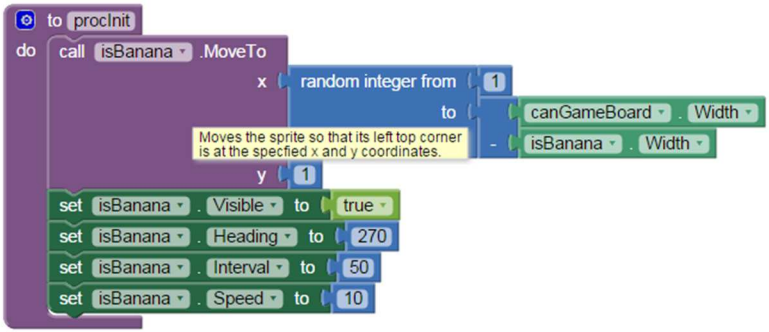
Blokkszerkesztő:

Állítsuk be a képernyő és a játék alapértékeit az scnForest.Initialize esemény-ben.

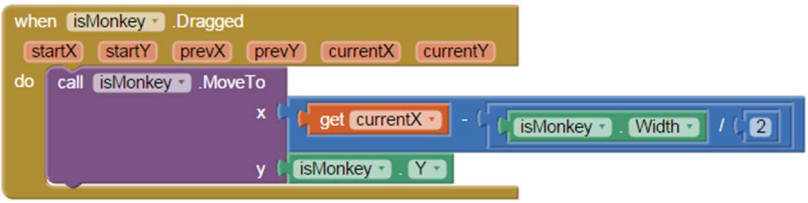
- Beállítjuk a háttér plusz rétegét a vászon területére,
- elindítjuk a háttérzenét,
- és a majmot elhelyezzük alul.



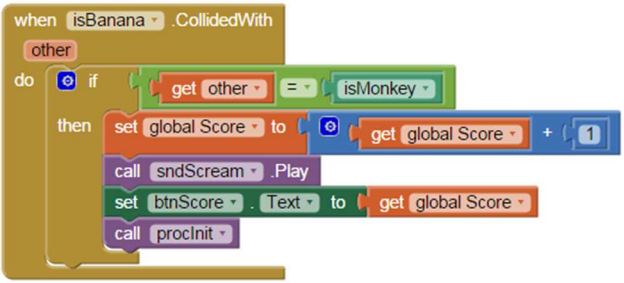
Hozzunk létre procInit eljárást, amellyel a banánt véletlenszerűen elhelyezzük a játéktér tetején. Ezt az eljárást fogjuk meghívni, ha leesik a banán, vagy elkapja a majom.



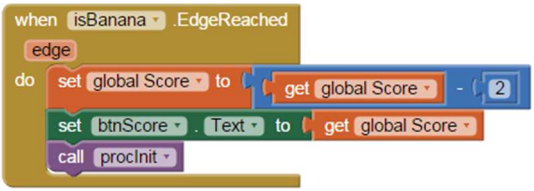
Kezeljük a majom mozgatását isMonkey.Dragged (csak vízszintesen mozoghat).



Amikor a banán ütközik a majommal isBanana.CollidedWith és az other paraméter a majom, növeljük a pontot, lejátszuk a sikoltást sndScream.Play és a banánt véletlenszerűen elhelyezzük a játéktér tetején procInit.



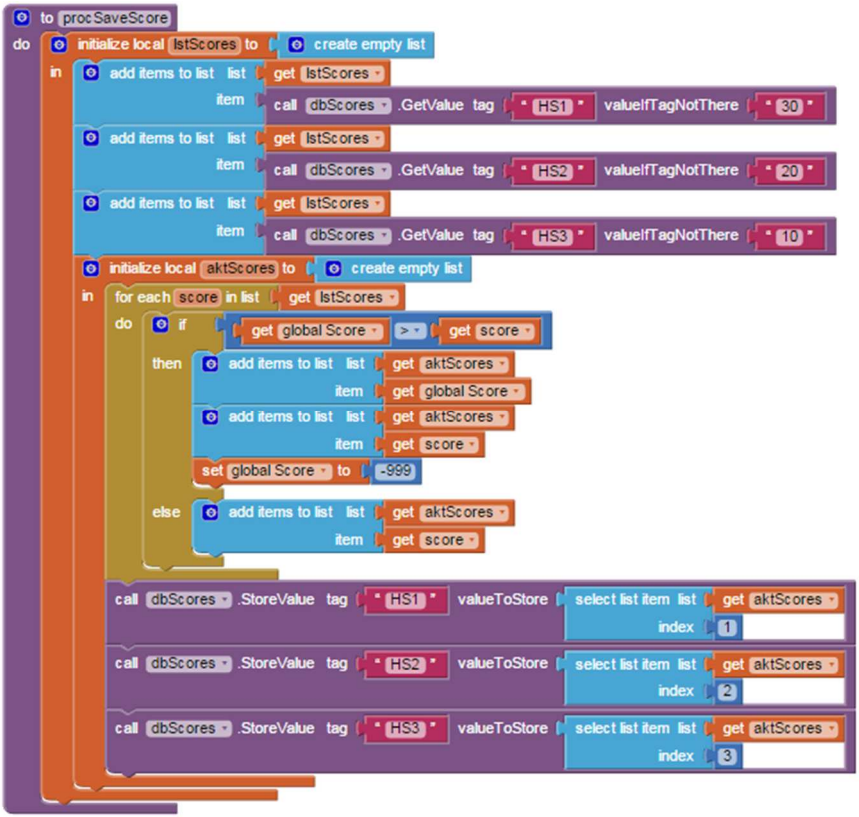
Amennyiben a banán eléri a játéktér alját levonunk 2 pontot és a banánt véletlenszerűen elhelyezzük a játéktér tetején procInit.



Végezetül a vissza gomb megérintésével btnBack.Click befejezhetjük a játékot, leállítjuk a háttérzenét bgMusic.Stop, mentjük az elért pontot procSaveScore és visszatérünk a menühöz close screen.

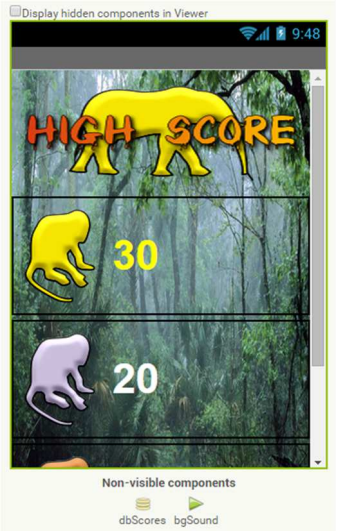
A 3 legjobb eredmény tárolásához:

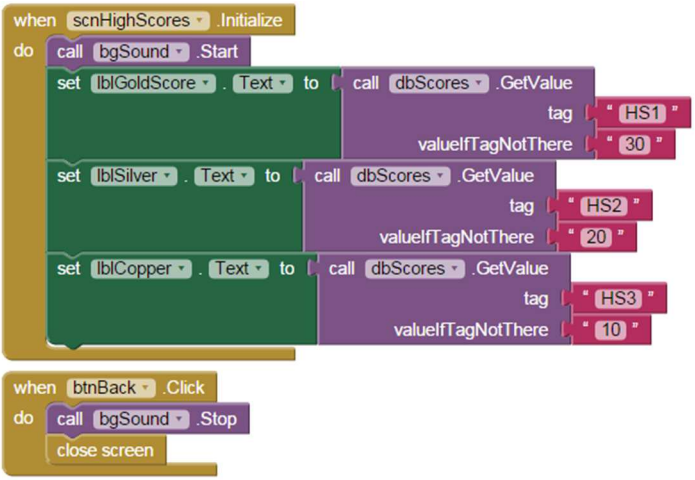
- listába tesszük a HS1, HS2 és HS3 értékeit dbScore.GetValue.
- végigmegyünk a lista értékein és új listát készítünk az elemekből úgy, hogy beszúrjuk az aktuális pontot is. Így 4 elemű rendezett listánk lesz.
- A 4 elemű lista első 3 elemét eltároljuk a HS1, HS2 és HS3 tagekhez dbScore.StoreValue.



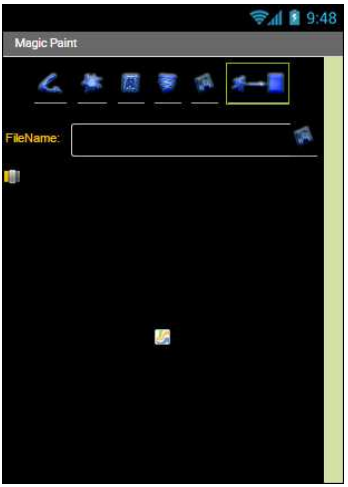
Mutassuk meg a 3 legjobb pontot

Amikor a képernyőt megnyitjuk schHighScores.Initialize megjelenítjük a HS1, HS2 és HS3 értékeit a címkéken dbScore.GetValue.





Végezetül a vissza gomb megérintésével btnBack.Click befejezhetjük a játékot, leállítjuk a háttérzenét bgMusic.Stop, mentjük az elért pontot procSaveScore és visszatérünk a menühöz close screen.



A MagicPaint alkalmazáson keresztül bemutatható a képernyő megérintésén, illetve az ujjunk húzásán alapuló rajzolás vászonra.

A képernyőre rajzoláshoz a felhasználó kiválaszthatja a rajzolás típusát (folyamatos vonal, sugarak, pöttyök), megadhatja a rajzolás színét, illetve a vonal/pötty szélességét. Készíthet fotót a mobileszköz kamerájával, amelyet beállíthat a vászon hátterének, illetve mentheti a kész képet.

A megvalósítandó funkciók:

- Vászon használata rajzoláshoz a húzás és érintés eseménykezelőkkel.
- Kamera használata a vászon hátterének beállításához.
- Csúszka használata a színkeveréshez és vonal vastagság beállításához.
- TinyDB használata a beállítások mentéséhez és visszaállításához.

A vászon képessé tétele az érintés és húzás érzékelésére.

Az alkalmazás készítésekor egyszerű lépésenként haladunk a különböző funkciók megvalósításával a bonyolult több funkciót egyesítő alkalmazás felé.

Designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Screen1	-	-	Elsődleges képernyő	AlignHorizontal: Center AppName: MagicPaint BackgroundColor: Black ScreenOrientation: Portrait Scrollable: - Title: Magic Paint
Canvas	Drawing and Animation	cnvBG	Vászon a rajzoláshoz	BackgroundColor: none Height: Fill parent Width: Fill parent LineWidth: 5 PaintColor: Green
Button	User Interface	btnExit	Kilépés az alkalmazásból	Image: mnuExit.png 

Blokkszerkesztő:

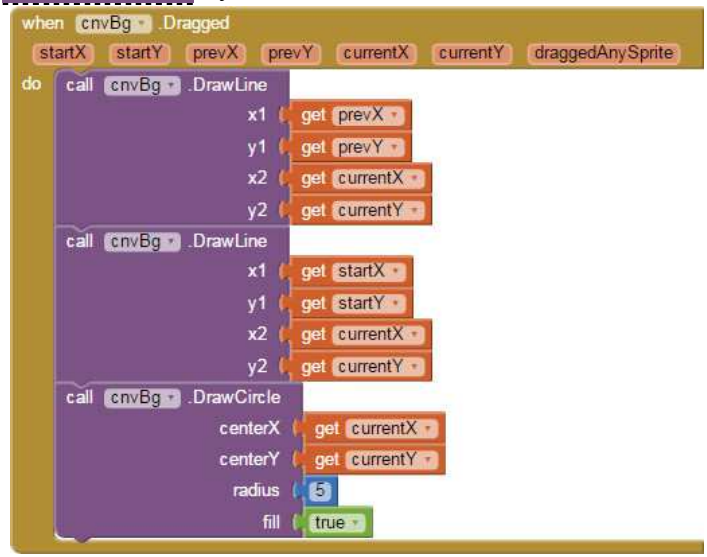
Szabályos kilépés az alkalmazásból:

Elsőként használjuk a btnExit.Click eseményét az alkalmazás szabályosan történő leállításához.

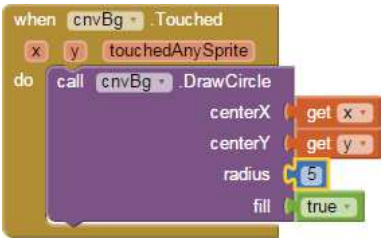
Rajzolás a vászonra:

A cnvBG.Dragged eseménykezelő számunkra 3 fontos (x,y) pozíciót ad vissza, amely kihasználásával tudjuk a rajzolás típusát megadni:

- a (prevX, prevY) és (currentX, currentY) között a vonal rajzolás cnvBG.DrawLine folyamatos lesz
- a (startX, startY) és (currentX, currentY) között a vonal rajzolás cnvBG.DrawLine sugaras lesz
- a (currentX, currentY) pozícióra rajzolhatunk pöttyöket a cnvBG.DrawCircle eljárással



A cnvBG.Touched eseménykezelővel tudjuk kezelni, ha a felhasználó pöttyöket akar rajzolni a képernyő megérintésével a kapott (x, y) paraméterek felhasználásával:



A vászonra rajzolás fejlesztése.

Ahhoz, hogy futásidőben állítani tudjuk, a rajzolás típusát be kell vezetnünk egy globális változót global intMode; létre kell hoznunk egy gombot a rajzolás módjának változtatásához, valamint kelleni fog egy vízszintes elrendezés elem is.

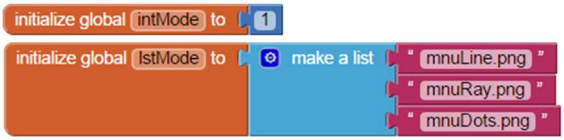
Kiegészítő designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
HorizontalAlignment	Layout	haMenu	Vízszintes elrendezés a menühöz	AlignHorizontal: Center Width: Fill parent
Button	User Interface	btnMode	Rajzolási módok váltása	Image: mnuLine.png 

Helyezzük el a btnMode és a korábban létrehozott btnExit gombokat a haMenu belsejébe.

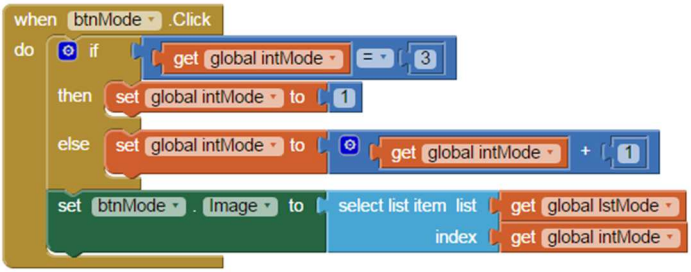
Blokkszerkesztő:

Alapértelmezett rajzolási mód megadásához az előző változatban felsorolt módokat sorszámozzuk be 1-től 3-ig (a folyamatos vonalrajzolás legyen 1) és készítsünk hozzá listát.

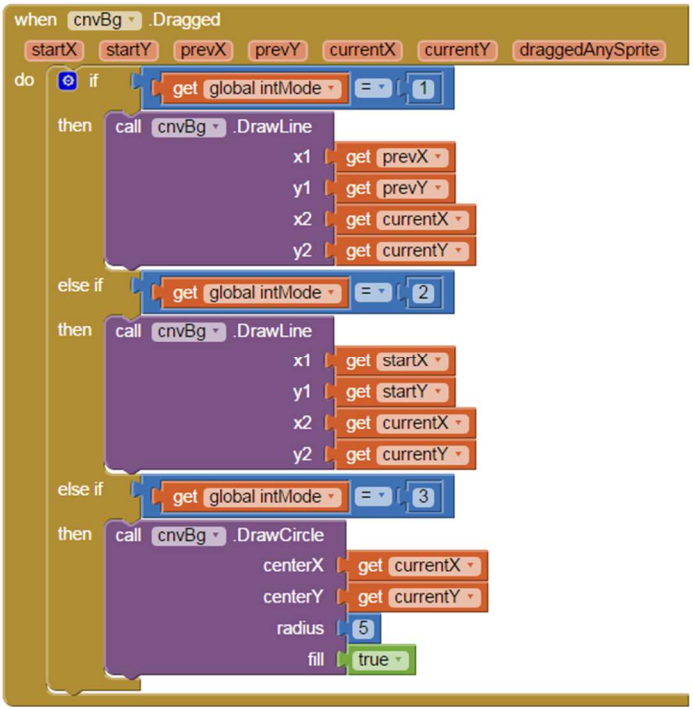


A gombhoz tartozó mindhárom képfájlt töltsük fel a Médiafájlfeltöltő segítségével, ne csak a gombon éppen használtat!

Most jöhet a gomb megérintésének kezelése: btnMode.Click, azaz a rajzolás módjának állítása. A cél, hogy minden érintéskor változzon a rajzolás módja 1, 2, 3, majd újra 1. A btnMode gombunk képét is cseréljük le az aktuális rajzolási módnak megfelelő ikonra.

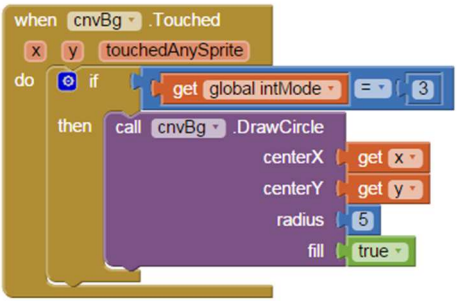


Ahhoz, hogy ennek legyen hatása a rajzolásra is, módosítani kell a korábban létrehozott cnvBG.Dragged eseménykezelő kódját is.



Az intMode globális változó aktuális értékétől függően rajzolunk húzásra folyamatos vonalat, sugarakat, vagy pöttyöket a vászonra.

A cnvBG.Touched érintés kezelő is csak akkor rajzoljon pöttyöt, ha ez az aktuális rajzolási mód.



Teszteljük az alkalmazásunkat!

A vonal szélességének, illetve pötty sugarának állítása.

Miután már többféle technikával is tudunk rajzolni a vásznunkra, érdemes lenne beállítani a rajzolás szélességét is futásidőben. Ehhez újabb gombra és egy csúszkára lesz szükségünk.

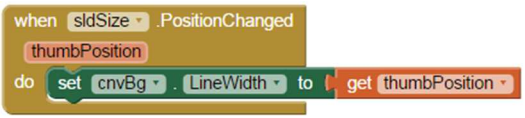
Kiegészítő designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Button	User Interface	btnSize	Csúszka megjelenítése és elrejtése	Image: mnuSize.png 
Slider	User Interface	sldSize	Ecsetszélesség állítása	Width: Fill parent MaxValue: 25 MinValue: 2 ThumbPosition: 5 Visible: False

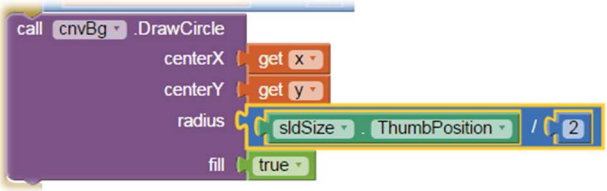
Blokkszerkesztő:

Az ecsetméret állító gomb megérintésekor btnSize.Click állítsuk a csúszka láthatóságát az ellentétére, azaz láthatóból rejtett, rejtettből látható lesz.

Az ecsetméretet a csúszka mozgatásakor sldSize.PositionChanged állítjuk. Ezt természetesen csak látható vezérlő esetén tudjuk megtenni.



Módosítsuk mindkét pöttyrajzoló cnvBg.DrawCircle eljárásunkat úgy, hogy a pötty sugara a vonalrajzolás szélességének a fele legyen.



Az ecsetméret a vászon tulajdonsága, minden vonal rajzolására egységesen vonatkozik, elég akkor állítani ha más mérettel akarunk rajzolni!

Rajzolás színének beállítása

Hozzunk létre egy új képernyőt, melyre elhelyezünk 3 csúszkát egymás alatt és egy gombot amin mutatjuk a beállított színt és visszatérhetünk az előző képernyőre. Ahhoz, hogy az első képernyőt használni tudjuk a beállított színt, szükségünk lesz egy TinyDB komponensre is.

Designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Screen	-	scnColor	Rajzolás színének beállítása	AlignHorizontal: Center BackgroundColor: Black ScreenOrientation: Portrait Scrollable: - Title: scnColor
Slider	User Interface	sldRed	Az RGB vörös komponensének beállításához	ColorLeft: Red Width: Fill parent MaxValue: 255 MinValue: 0 ThumbPosition: 55
Slider	User Interface	sldGreen	Az RGB zöld komponensének beállításához	ColorLeft: Green Width: Fill parent MaxValue: 255 MinValue: 0 Thumbposition: 55
Slider	User Interface	sldBlue	Az RGB kék komponensének beállításához	ColorLeft: Blue Width: Fill parent MaxValue: 255 MinValue: 0 Thumbposition: 55
Button	User Interface	btnSetColor	Szín beállítás és visszatérés az előző képernyőre	Height: Fill parent Width: Fill parent Text: üres
TinyDB	Storage	dbSettings	Beállított szín mentésére	-

Blokkszerkesztő:

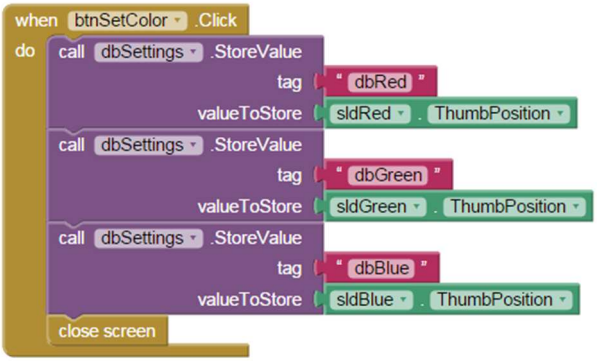
Bármelyik csúszka mozgatásakor módosítanunk kell a gomb háttérszínét, ezért hozzunk létre számukra egy közös setButtonColor eljárást, amit csak meg kell hívni.

A make color blokk számára egy három számból álló listát kell átadni. A három szám a beállítani kívánt szín RGB kódja, jelen esetben a három csúszkánk ThumbPosition értéke.

Következő lépésben hozzuk létre a csúszkáink .PositionChanged eseménykezelőit és hívjuk meg a setButtonColor eljárást.

Ezzel a btnSetColor gombunk háttérszíne folyamatosan változik a csúszkák pozíciójának megfelelően.

A beállított színnel való rajzoláshoz a btnSetColor.Click eseményében mentsük el a rajzolás színét az alkalmazás TinyDB adattárolójában dbSettings.Storevalue. Majd zárjuk be a képernyőt close screen, ezzel visszatérünk a vásznunkhoz és folyatathatjuk a rajzolást.



A beállított szín felhasználása.

Ahhoz, hogy felhasználhassuk az eltárolt színt a Screen1 képernyőnkön vissza kell olvasnunk az alkalmazáshoz tartozó adattárolóból a dbSettings.GetValue függvénnyel.

Kiegészítő designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
TinyDB	Storage	dbSettings	Beállított szín visszaolvasására	-

Blokkszerkesztő:

A rajzolás színének visszaolvasásához dbSettings.GetValue és beállításához cnvBG.PaintColor használjuk a Screen1.OtherScreenClosed eseményét, ez az esemény akkor váltódik ki, amikor egy másik képernyő bezárásával visszajutunk a Screen1 képernyőre. Mivel csak a Screen1 és scnColor képernyőink vannak így az eseménykezelő által biztosított otherScreenName paraméter csak az scnColor lehet.

Vászon törlése

Ahhoz, hogy töröljük a vászon tartalmát, hozzunk létre egy gombot btnClear néven és helyezzük el a haMenu menüsorban.

Kiegészítő designer elemek:

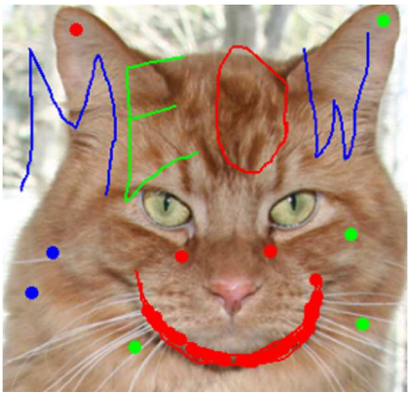
Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Button	User Interface	btnClear	Vászon törléséhez	Image: mnuClear.png Text: nincs

Blokkszerkesztő:

Az újonnan létrehozott gomb feladata, hogy megérintésekor btnBG.Click törölje a vászon tartalmát cnvBg.Clear.

Kamera használata a vászon hátterének beállításához.

Következő lépésünk, hogy a mobiltelefon kamerája által készített képet beállítsuk a rajzoló alkalmazásunk hátteréként és a művészi hajlamainkat ezen éljük ki a továbbiakban.



Kiegészítő designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Button	User Interface	btnBg	Képkészítés elindításához	Image: mnubg2.png Text: nincs
Camera	Media	cam	Fénykép készítéshez	-

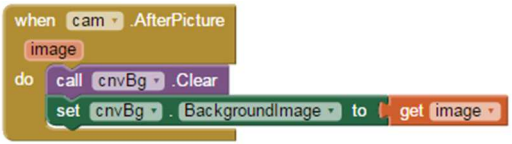
Helyezzük el a btnBg gombot a haMenu elrendezés vezérlőben és állítsuk be a tulajdonságai a fentieknek megfelelően.

Blokkszerkesztő:

Az újonnan létrehozott gomb feladata, hogy megérintésekor btnBG.Click indítsa el a telefon alapértelmezett kamera alkalmazásának képkészítés funkcióját cam.TakePicture.

Ilyenkor a rajzoló alkalmazásunk a háttérbe szorul és a kamera alkalmazás lesz aktív, a kép elkészítése után lesz újra aktív az alkalmazásunk.

Szükségünk van még a visszakapott kép beállítására, melyet a cam.AfterPicture esemény image paraméterével tudunk megtenni. Előtte érdemes lehet a vászon törlése cnvBg.Clear.



Elkészült kép mentése

Utolsó előtti lépésünk, hogy az elkészült képet el tudjuk menteni a telefonra.

Kiegészítő designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
Button	User Interface	btnSave	Mentés almenü megjelenítéséhez	Image: mnuSave.png Text: nincs
HorizontalAlignment	Layout	haSave	Vízszintes elrendezés a mentés almenühöz	AlignHorizontal: Center Width: Fill parent
Label	User Interface	Label1	Címke a fájlnévhez	Text: FileName:
TextBox	User Inerface	txtFileName	a kép nevének megadásához	Width: Fill parent Text: üres
Button	User Interface	btnSave2	Kép mentéséhezz	Image: mnuSave.png Text: nincs
Notifier	User Interface	msgBox	Üzenet megjelenítése a mentésről	NotifierLength: Long

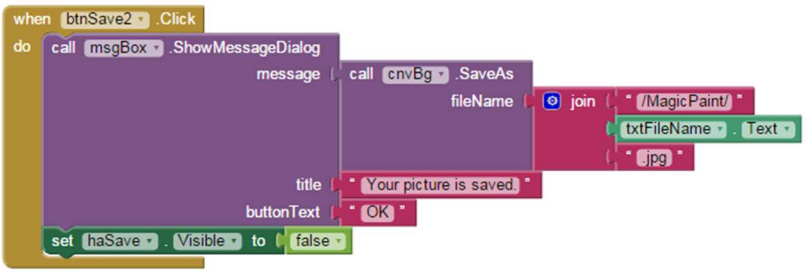
Helyezzük el a btnSave gombot a haMenu elrendezés vezérlőben és állítsuk be a tulajdonságai a fentieknek megfelelően.

Ez után hozzuk létre a haSave elrendezést és helyezzük el benne a Label1 címkét, a txtFileName szövegdobozt és a btnSave2 gombot.

Blokkszerkesztő:

A menüben elhelyezett gomb feladata, hogy megérintéskor btnSave.Click megjelenítse/elrejtse az almenüt.

Amennyiben a kép mentése almenü látszik, a felhasználó tud írni a szövegdobozba és tudja menteni cnvBg.SaveAs a képet. Az eredményről tájékoztatjuk a felhasználót msgBox. ShowMessageDialog. Mentés után elrejtjük a képmentés almenüt.



Végső simítások

Oldjuk meg a különböző változók inicializálását, illetve mentését és beolvasását. Ehhez használjuk a dbSettings.SetValue és dbSettings.GetValue funkciókat, illetve a Screen1.Initialize és scnColor.Initialize eseményeket.

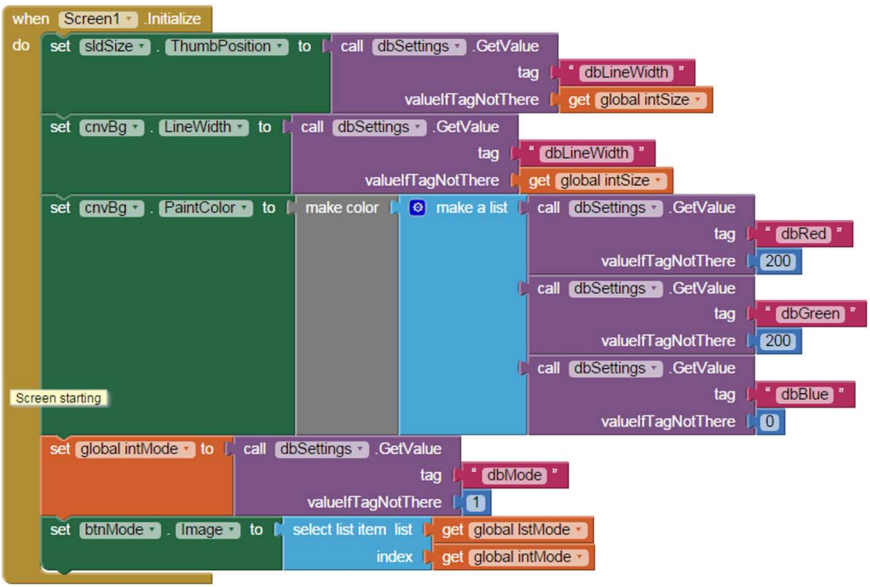
Kiegészítő designer elemek:

Nincs szükség további elemekre.

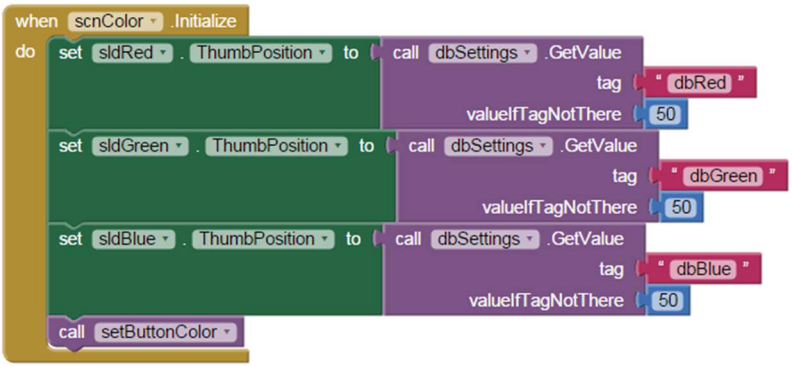
Blokkszerkesztő:

A végleges globális változók a Screen1 képernyőn a következők

Az alkalmazás indításakor be kell olvasni az alkalmazás adatterületéről a globális változók legutóbbi értékeit és beállítjuk a korábban létrehozott vezérlők tulajdonságait.



Az scnColor képernyő megnyitásakor beolvassuk a vörös, zöld és kék színkomponensek korábbi értékeit a dbSettings.GetValue funkciókkal, és beállítjuk a csúszkák Thumb értékeit, valamint meghívjuk a setButtonColor eljárást a btnSetColor gomb háttérszínét.



Az Arduino egy nyíltforrású hardver, ami azt jelenti, hogy az interneten elérhetők az eszközök tervei, és az alapján bárki legyártathatja, illetve forgalomba hozhatja. Az Arduino gyakorlatilag egy kicsi számítógép, amire programot írva különböző érzékelők jeleit olvashatjuk le, illetve eszközöket vezérelhetünk - motorokat mozgathatunk, eszközöket kapcsolhatunk be- vagy ki.

Programozás

Az Arduinót alapvetően C/C++ nyelven programozzák. Mi megpróbáltuk ezt a platformot közelíteni a programozásban még járatlan fiatalokhoz, ezért készítettük el hozzá a Blocklinot. Ez egy blokk alapú fejlesztő környezet, mellyel néhány perc alatt egy teljesen laikus is elkészítheti első programját az eszközre.

A fejlesztőkörnyezet a blocklino.org oldalon érhető el.

A környezet felépítése

Programunk különböző részeit az alábbi fő blokk megfelelő részébe kell megírni.

Ennek 5 része van:

- **Variables:** Ebben a szekcióban definiálhatjuk változóinkat.
- **Functions:** Itt hozhatunk létre eljárásokat, amelyeket főleg akkor használunk, ha bizonyos műveletsort a programunkban sokszor kell végrehajtanunk. A bonyolultabb, összefüggő műveletsorokat is megírhatjuk, és névvel elláthatjuk itt, hogy a programunk könnyebben érthető legyen.
- **Timers:** Ebben a részben az időzítőket állíthatjuk be. Ezeket olyan feladatok végrehajtására használjuk, amelyeket gyakran, ismétlődve kell végrehajtanunk, de tipikusan a főprogramunktól függetlenül.
- **Setup:** A program ezen része indul el először az eszközön, az itt található parancsok egyszer hajtódnak végre.
- **Loop:** A programunk magja, az itt található kódot hajtja végre az eszköz működése közben, folyamatosan ismételve azt.

A LED egy világító dióda, ami áram hatására világít. Az első arduniós programunk egy ilyen eszköz villogtatása lesz. A programunk célja, hogy egy LED-et felkapcsoljunk, az 1 másodpercig világítson, majd ezután 1 másodpercig ne világítson, és ez folyamatosan ismétlődve hajtódjon végre.

Hardver:

A feladat első megoldásához nincs szükségünk különösebb hardver megépítésére, ugyanis az Arduinón találhatunk egy LED-et, ami az eszköz 13-as lábára van kötve. A programunkban először ezt a LED-et fogjuk vezérelni.

Szoftver:

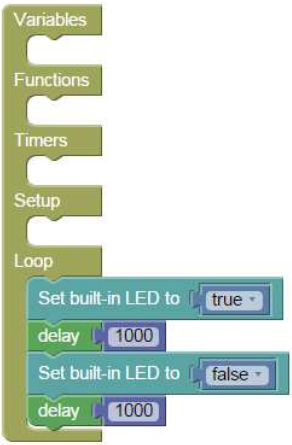
A programunk első verziója egyszerű lesz, mindössze két blokkra lesz szükségünk.

Setup:

Ennél a programnál erre a szekcióra nem lesz szükség, mert nem kell semmi kezdeti beállítást elvégeznünk.

Loop:

- Szükségünk lesz egy Set built-in LED to blokkra, amivel a beépített LED-et vezérelhetjük egyszerűsített módon. Ennek az egyetlen egy bemenete a LED kívánt állapota, azaz, hogy be (true) vagy ki (false) szeretnénk kapcsolni. A blokkot a Devices fül alatt találjuk.
- Emellett szükségünk lesz egy Delay blokkra, amellyel a program futását blokkolhatjuk egy általunk meghatározott ideig. Ezt az időt adhatjuk meg a bemenetén ezredmásodpercben (milliszekundumban) mérve (1 sec = 1000 ms). A blokkot a Control fül alatt találjuk.



Egy hasonló eredményt produkáló, több blokkból álló változatot is elkészítünk a következőkben, ennek elkészítése hasznos lehet a későbbiekben.

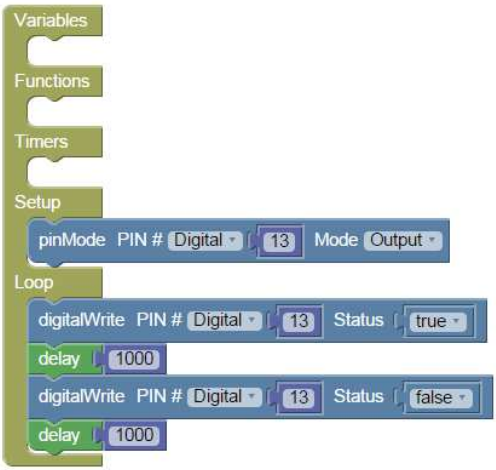
Setup:

Ebben a szekcióban állítjuk be az eszköz 13-as lábának módját, azaz hogy a továbbiakban kimenetként vagy bemenetként szeretnénk használni. Ehhez a pinMode blokkot használjuk, aminek meg kell adnunk, hogy ezt a módot melyik lábra szeretnénk beállítani, ez jelen esetben a digitális 13-as láb lesz, mivel a beépített LED erre van kötve. Ezt azért tettük ebbe a szekcióba, mert a programunk futása során az adott lábat mindig kimenetként fogjuk használni, így azt elég egyszer, a program futása előtt beállítani. A blokkot az *I/O* fül alatt találjuk.

Az előző példában erre azért nem volt szükség, mert a Set built-in LED to blokk ezt elvégezte helyettünk. Azonban jó tudni, hogy mi történik a háttérben, mielőtt továbblépünk, hiszen általános esetben, mikor nem a beépített LEDet vezéreljük, magunknak kell majd erről gondoskodnunk.

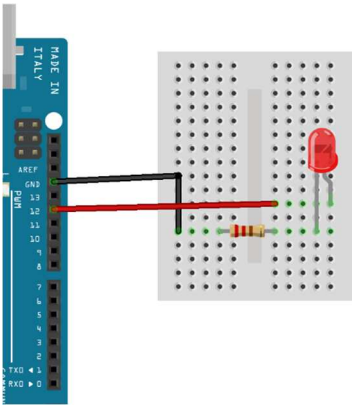
Loop:

Ez a szekció nagyban hasonlít a programunk az előzőekben elkészítetthez, azonban itt egy digitalWrite blokkot használunk, amellyel a bemenetén megadott lábat – jelen esetben a 13-ast – vezérelhetjük. A várakozás itt is a delay blokkal történik.



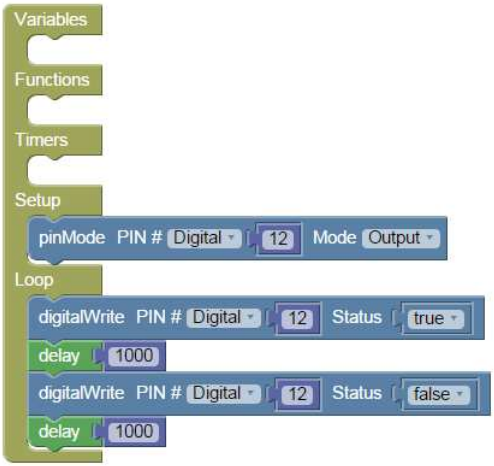
Hardver:

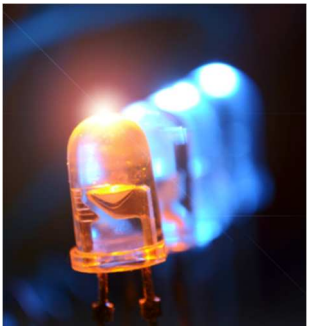
A feladat második megoldásához bekötünk egy LED-et az Arduinóhoz. Ehhez szükségünk lesz egy ellenállásra is, amit a LED negatív (rövidebb) lábára kötünk. A LED pozítív lábát (a hosszabikat) a 12-es kivezetésre kötjük az Arduinón, a szoftverben majd ezt a lábat vezérelve állíthatjuk a LED állapotát. A negatív lábat az ellenálláson keresztül a GND felíratú kivezetésre, a földre kötjük.



Szoftver:

A programunk gyakorlatilag azonos lesz az előző feladat során elkészítettel, csak most a 13-as láb helyett a 12-est kell vezérelnünk.





Az eddigi feladatokban a LED-et csak be-, illetve kikapcsoltuk, emellett azonban a fényerejének állítására is van mód. Ezt elérhetjük például a rajta átfolyó áram növelésével vagy csökkentésével, vagy úgy, ha - nagyon gyorsan - ki- és bekapcsolgatjuk a LED-et.

Utóbbi esetben - ha elég gyorsan csináljuk - a szemünk nem látja a vibrálást, hanem kisebb fényerőként érzékeli ezt. Minél nagyobb a kikapcsolt idő aránya a bekapcsoltéhoz képest, annál halványabbnak látjuk a fényt. Ezt a módszert hívják PWM-nek (Pulse Width Modulation, impulzusszélesség moduláció - azt változtatjuk, milyen "széles" az impulzus, amit a LED-re kapcsolunk). Az Arduinó ez utóbbi módszerhez ad segítséget.

Ebben a feladatban a LED fényerején először a 0-ról a maximumra növeljük, majd a maximumról a 0-racsökkentjük, kis lépésekben haladva.

Hardver:

A feladat első megoldásához nincs szükségünk hardver megépítésére, ugyanis az Arduinón található beépített LED-et fogjuk használni, ez a 13-as lábra van kötve.

Szoftver:

A programunk első verziója egyszerű lesz, mindössze két blokkra lesz szükségünk:

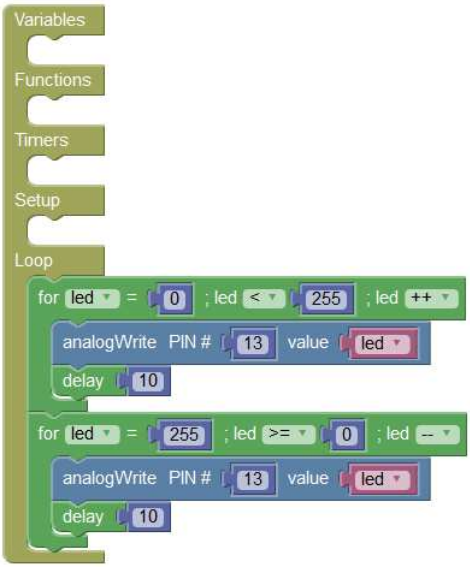
Setup:

Amennyiben digitális kimenetként vezérlünk egy lábat, akkor be kell állítsuk a pinMode blokkal, hogy azt kimenetként szeretnénk használni. Amennyiben azonban az analogWrite blokkal vezéreljük a lábat, nem szükséges azt előtte kimenetként beállítani, így ez a szekció üresen maradt ebben a példában.

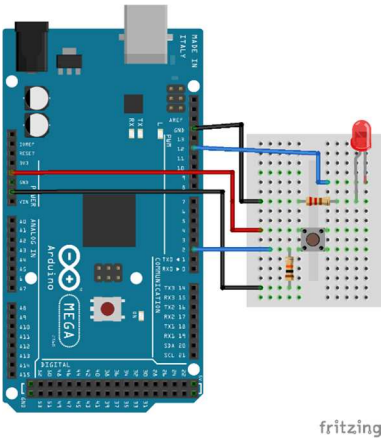
Loop:

- A loop szekcióban két nagyon hasonló For ciklust találunk, ezek értékei 0-tól 254-ig, illetve 255-től 0-ig változnak. Ennek oka az, hogy a lábra egy 0 és 255 közötti számot írva állíthatjuk be az állapotát, a 0 a legalacsonyabb szint (folyamatosan kikapcsolva), a 255 pedig a legmagasabb (folyamatosan bekapcsolva).
- A ciklusokban mindössze annyit teszünk, hogy kiküldjük a ciklusváltozó értékét. Ehhez az analogWrite blokkot használjuk, amely az I/O szekcióban található. Ezen kívül a korábban már használt Delay blokkal várakozunk 10 milliszekundumot. Erre azért van szükség, hogy lássuk a LED fényerejének változását.

A feladatot meg lehet oldani külső LED-del is, amelyet a korábban ismertetett módon kell bekötni. A szoftverben ebben az esetben a 13-as port helyett azt kell használni, amelyikre a LED-et kötöttük.



A feladat során egy nyomógomb bejövő jelét fogjuk használni egy LED vezérlésére. A cél az, hogy amíg a gombot lenyomva tartjuk, a LED világítson.



Hardver:

A LED-et ezúttal is a 12-es lábra, az előzőekben már megismert módon fogjuk kapcsolni. A nyomógombunknak 4 lába van, ezekből a 2-2 szemközti van összekötve alaphelyzetben, a nyomógomb benyomásakor pedig az egy oldalán lévő lábak lesznek összekötve. A nyomógomb bemeneti jelének olvasásakor a két, egy oldalon lévő lábat kell figyelnünk. Az egyiket az Arduino 5V lábára kötjük, a másikat pedig egy ellenálláson keresztül a GND-re, a földre. Az ellenállás másik lábát fogjuk bemeneti jelként használni, így azt az Arduino 2-es lábára kötjük.

Amikor a nyomógombot nem tartjuk nyomva, olyankor az szakadást jelent az áramkörben. A 2-es lábon ilyenkor logikai alacsony szintet (*false*) érzékelünk.

Ha a nyomógombot megnyomjuk, akkor záródik, ilyenkor az ellenállásra +5V kerül. Mivel más alkatrész nincs bekötve, az ellenálláson fog esni az 5V, így a nyomógomb felőli végén - az Arduinó 2-es lábán - +5V-ot, tehát logikai igaz (*true*) szintet mérhetünk.

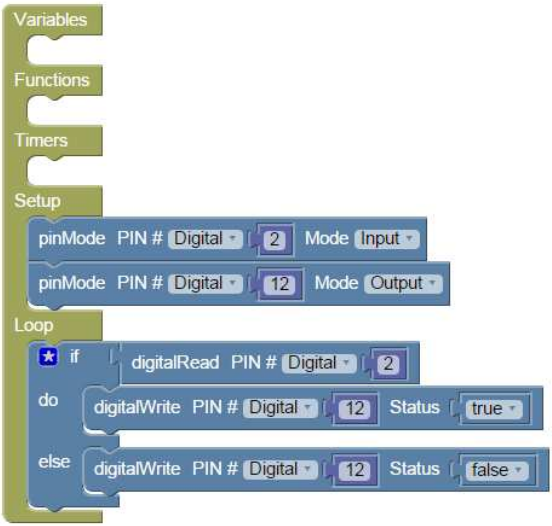
Szoftver:

Setup:

Ebben a szekcióban a korábban már használt pinMode blokkal beállítjuk a LED és a nyomógomb lábainak módját. A LED-et itt is *kimenet*ként állítjuk be, a gombot viszont *bemenet*ként, ugyanis a jelét szeretnénk olvasni.

Loop:

A Loop gyakorlatilag egy elágazásból áll, aminek a gomb értéke a feltétele. Ezt az értéket a digitalRead blokkal tudjuk beolvasni. A LED-et akkor kapcsoljuk be, ha a gombról olvasott érték *igaz*, tehát a nyomógombot benyomták, egyéb esetben a LED-et kikapcsoljuk.

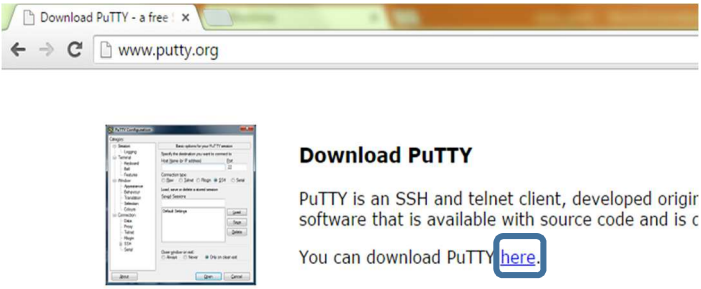

Variables
Functions
Timers
Setup
pinMode PIN # Digital 2 Mode Input
pinMode PIN # Digital 12 Mode Output
Loop
if digitalRead PIN # Digital 2
do digitalWrite PIN # Digital 12 Status true
else digitalWrite PIN # Digital 12 Status false

Az Arduinó és a számítógép kommunikálhat egymással, adatokat küldhetnek egymásnak. Ezt kihasználva megjeleníthetjük a számítógépen az Arduinó által küldött adatokat - ez lehet valami, amit kiszámít, vagy egy érzékelőn, pl. hőmérőn vagy fényerősség mérőn mért értékek, - valamint adatokat, vezérlést is küldhetünk az Arduinónak (pl. állítsd a szervó motort 90 fokhoz).

A kommunikáció az USB porton keresztül valósul meg, amelyen a rendszer egy soros portot fog emulálni. Erről több információra nincs szükségünk, elég annyit tudni, hogy amit az egyik oldalon (pl. az Arduinón) beleírunk, azt a másik oldalon (a számítógépen) kiolvashatjuk, és fordítva.

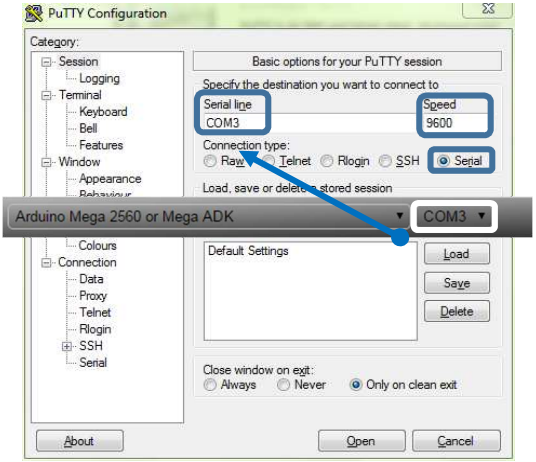
A PuTTY letöltése:

Egy PuTTY nevű alkalmazásra lesz szükségünk ahhoz, hogy kommunikáljunk az Arduinóval. Ezt a *putty.org* oldalról tölthetjük le.

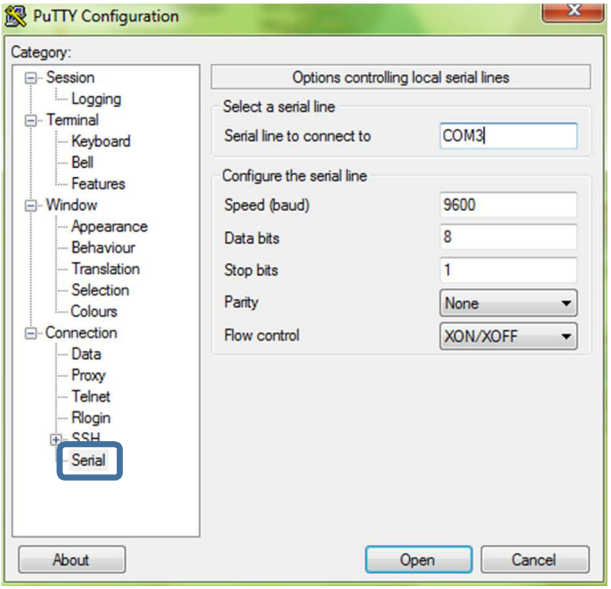


A PuTTY beállítása:

A soros port használatához meg kell adnunk a port sorszámát. Itt is azt a portot kell használnunk, amit a kód letöltésekor, a böngészőben használtunk.



Ezután a *Serial* fülön ellenőrizzük a beállítások helyességét.



Majd az Open gombbal megnyitjuk a csatornát.

A fenti képernyő jelenik meg előttünk, ahol az Arduinóról érkező adatokat láthatjuk, illetve mi is kommunikálhatunk az Arduinóval.

Az Arduino és a számítógép között soros porton tudunk kommunikálni. Az első feladat során az Arduino egyesével elküldi a számokat 10-ig soros porton a számítógépre.

Hardver:

Ehhez a feladathoz nincs szükségünk különösebb hardverre, mindössze az Arduinót fogjuk használni.

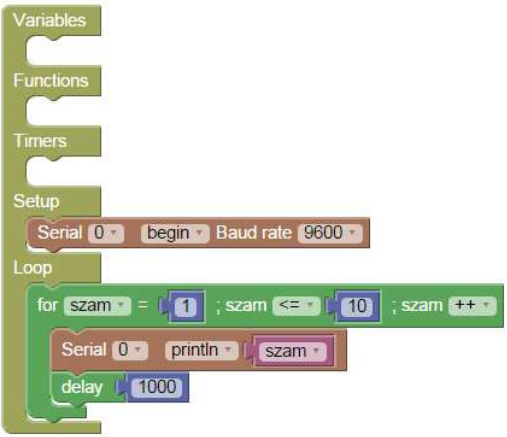
Szoftver:

Setup:

A soros portot meg kell nyitnunk, hogy használjuk, mint kommunikációs csatornát, ehhez a Serial blokkon a *begin* opciót kell választanunk. A *port sorszámát* nem változtatjuk meg, a 0 sorszámút használjuk, és a *kommunikáció sebességét* (Baudrate) is 9600-on hagyjuk.

Loop:

- Az egymás után lévő számokat egy For ciklussal hozzuk létre, ahol a **szam** változó értékét növeljük minden egyes lefutáskor, és mindezt addig csináljuk, amíg az el nem éri a 10-et.
- A szám elküldéséhez szintén a Serial blokkot használjuk, és a *println* opcióval küldjük el a **szam** változó értékét. A blokk opciói közt találunk még egy *print*-et is, ez abban különbözik a *println*-től, hogy az elküldött adat után nem viszi át a kurzort a következő sor elejére, így a legközelebb elküldött adat egy sorba kerül a korábban *print*-tel elküldött adattal.
- Hogy ne egyszerre küldjük el a 10 számot, az elküldés után a korábban már használt delay blokkal várakozunk 1 másodpercet.


Variables
Functions
Timers
Setup
Serial 0 begin Baud rate 9600
Loop
for szam = 1 ; szam <= 10 ; szam ++
Serial 0 println szam
delay 1000

Az eddigiekben csak az Arduinóról küldtünk adatot a számítógépre, azonban a kommunikáció két oldalú, azaz a számítógépről fogadhat is adatot. Ebben a feladatban ezt fogjuk használni. Ha 1-est fogadott az eszköz, felkapcsoljuk a LED-et, ha pedig 0-ást, akkor lekapcsoljuk azt.

Hardver:

A feladat megoldásához először szintén csak az Arduinóra lesz szükségünk.

Szoftver:

Declare:

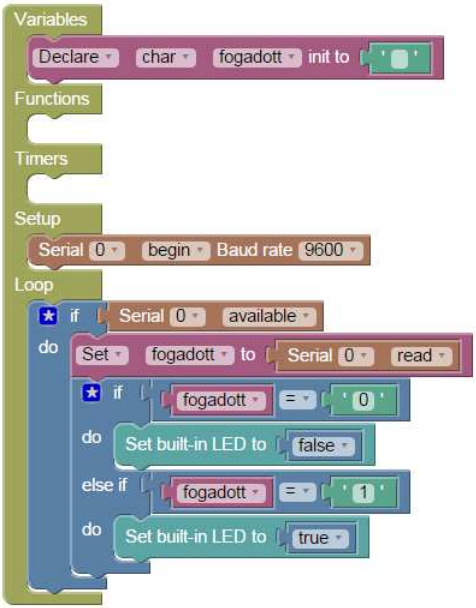
Deklarálnunk kell egy *char* típusú változót, amelybe majd a későbbiekben a soros csatornáról beolvasott karaktert tároljuk el. Ennek kezdőértékül egy üres karaktert adunk meg. A karakter konstanst a *Text* fül alatt találjuk, a beviteli mező két oldalán egy-egy aposztrófal.

Setup:

A soros portot ugyanúgy meg kell nyitnunk, mint az előző feladat során.

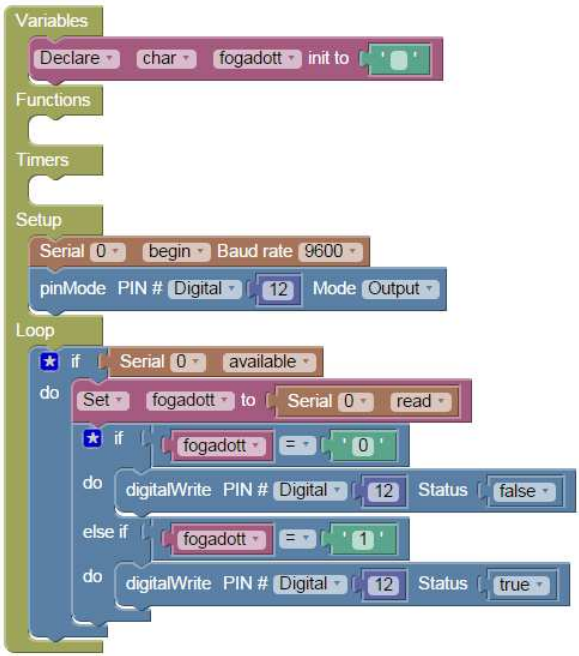
Loop:

- Ahhoz, hogy kiolvassuk és feldolgozzuk a fogadott adatot, tudnunk kell, hogy mikor érkezett adat a csatornán. Ezt a Serial *available* opciójával tudjuk meg, ami akkor igaz, ha van kiolvasásra váró adat.
- Ha van ilyen adat, akkor azt kiolvassuk a fogadott változóba a Serial *read* opciójával. Ezzel az olvasással egyszerre csak egy karaktert tudunk fogadni, ez azonban ebben a példában nem okoz gondot, mert csak egy karaktert várunk.
- Ezután a **fogadott** értéket vizsgáljuk, és amennyiben 0 értékű karaktert kaptunk, lekapcsoljuk a LED-et, ha pedig egy 1-est kaptunk, felkapcsoljuk. **Figyelem:** fontos, hogy a fogadott adatot *karakterként* kezeljük, tehát a zöld, egyszeres aposztrófal határolt blokkot használjuk, nem pedig a kék számblokkot vagy az idézőjellel határolt karakterlánc blokkot!


Variables
Declare
char
fogadott
init to
' '
Functions
Timers
Setup
Serial 0
begin
Baud rate
9600
Loop
if
Serial 0
available
do
Set
fogadott
to
Serial 0
read
if
fogadott
=
'0'
do
Set built-in LED to
false
else if
fogadott
=
'1'
do
Set built-in LED to
true

Ha nem a beépített LED-et akarjuk állítani, hanem egy külső LED-et, akkor egy kisebb módosítással megtehetjük ezt is. Az érkezett adat feldolgozása hasonlóképpen történik, mint az előző esetben, mindössze a vezérlésben különbözik a két program.

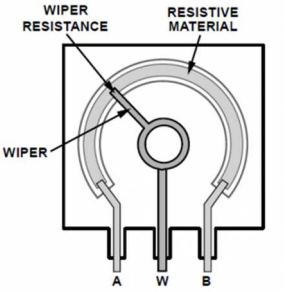
A példában a 12-es lábra kötöttük a LED-et, ezért a Setup-ban ezt állítjuk be kimenetként, a vezérlés során pedig ezt állítjuk igazra vagy hamisra a fogadott értéknek megfelelően.

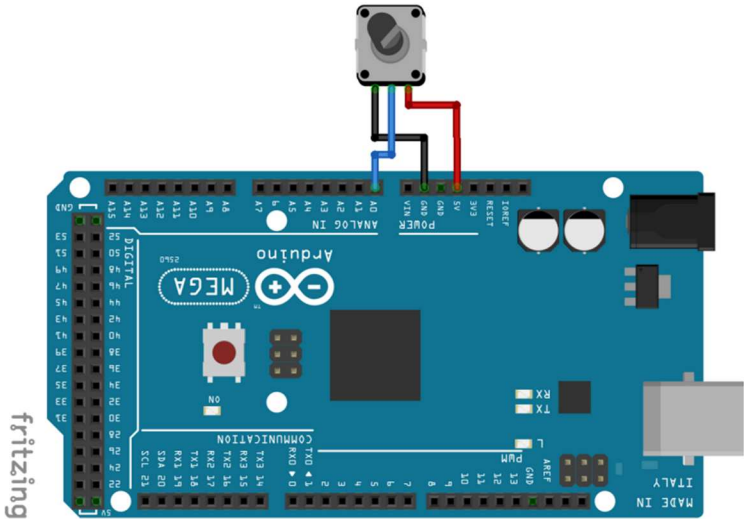


A potenciométer (vagy általánosabb nevén a potméter) egy változtatható ellenállású áramköri egység. Ezt fogjuk a feladat során bemenetként használni. Az első feladatban kiolvassuk az értékét, és azt soros porton továbbítjuk a számítógépre.

Hardver:

A feladat során be kell majd kötnünk egy potmétert. Ennek 3 db kivezetése van. Úgy kell elképzelni, mint két, sorba kötött ellenállást (R1 és R2), ahol a középső láb a két ellenállás összekötött lábának a kivezetése. A potméter tekergetésével R1 és R2 értéke változik úgy, hogy az összegük változatlan marad. Így a két szélső láb feszültségszintje közötti feszültséget állít a középső lábon. Ennek megfelelően az egyik szélső lábat a földre kötjük, a másik szélsőt az 5V-ra, a középsőt pedig az Arduino egyik *Analog* lábára.





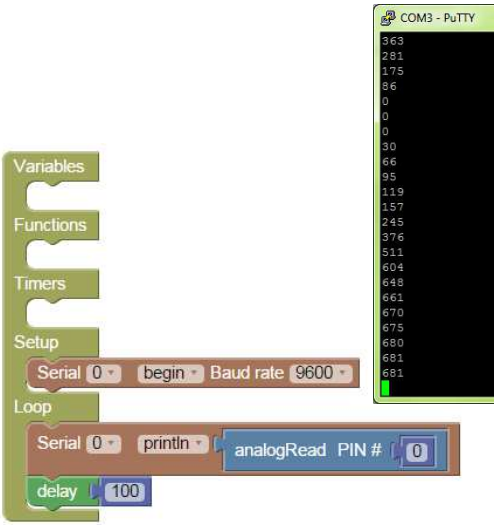
Szoftver:

Setup:

A soros portos kommunikációhoz szükségünk lesz annak inicializálására, amit a korábban már látott módon teszünk meg.

Loop:

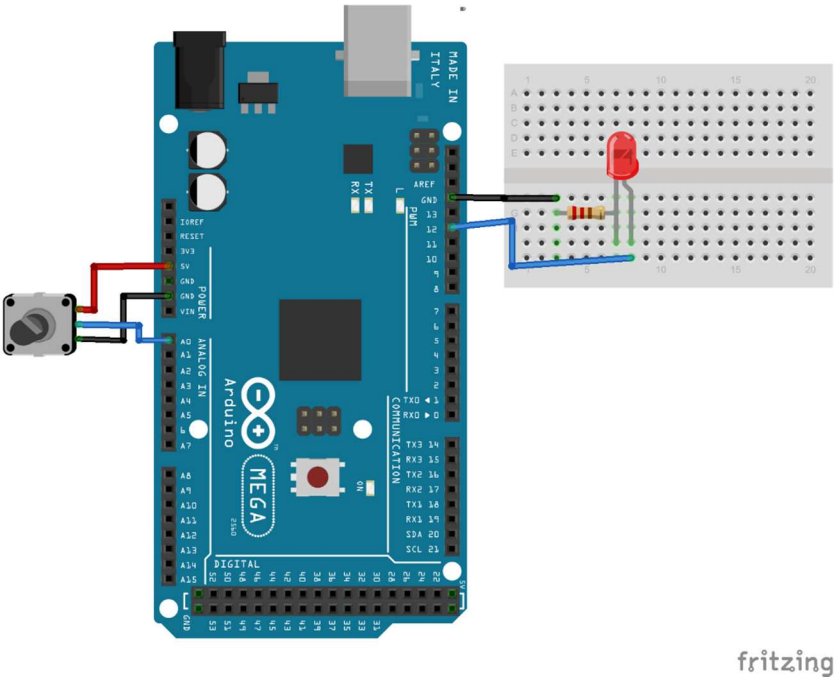
A soros portra kiírjuk az értéket, amit az A0-ás lábon olvasunk. Az olvasáshoz az analogRead blokkot használjuk. A beolvasás értéke 0-1023 közé esik. A kommunikáció után a delay blokkal várakozunk 100 milliszekundumot, erre azért van szükség, hogy a számítógépen is olvasható legyen az átküldött adat.



A feladat során két, korábban már használt eszközt használunk fel: egy potenciométert és egy LED-et. A feladat az, hogy a potméterrel állíthassuk a LED fényerejét.

Hardver:

A potmétert most is a 0-s analóg lábra kötjük, LED-et pedig a korábban már leírt módon a 12-es digitális lábra.



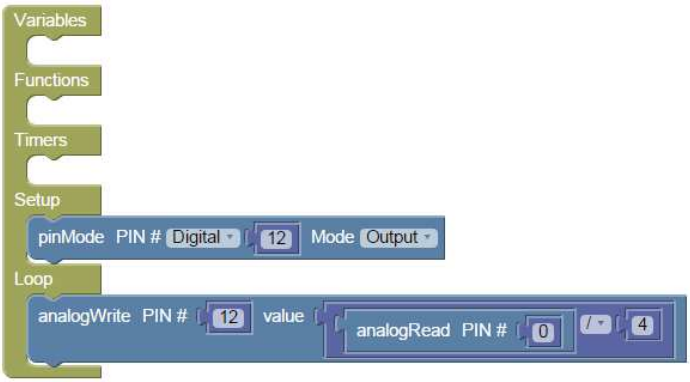
Szoftver:

Setup:

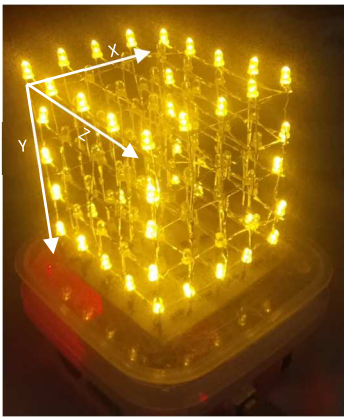
A LED kivezetését beállítjuk kimenetként.

Loop:

Egyetlen analogWrite blokkot használunk, hogy a LED fényerejét állítsuk, ennek bemenetére pedig a potméterről beolvasott érték negyedét írjuk. Az osztásra azért van szükség, mert a bemenetről 0-1023 közötti értéket olvasunk, a kimenetre azonban 0-255 közötti értéket kell írnunk. Ezzel a leképezéssel egyenlő részekre osztjuk a tartományt.



Hardver:

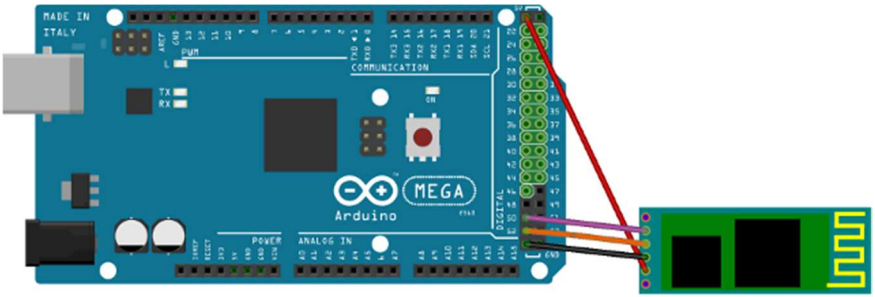


Az eddigi feladatokban önállóan programoztunk különböző blokknyelveken. Ehhez a feladathoz építettünk egy célhardvert, egy 5x5x5-ös LED kockát, illetve a hozzá kapcsolódó vezérlőpanelt un. „arduino shield”-et.

A kocka vezérlése (x,y,z) koordináták alapján lehetséges 0-4 közötti egész értékekkel.

A célunk az, hogy telefonról vezérelve interaktív animációval bemutassuk a kocka részeit.

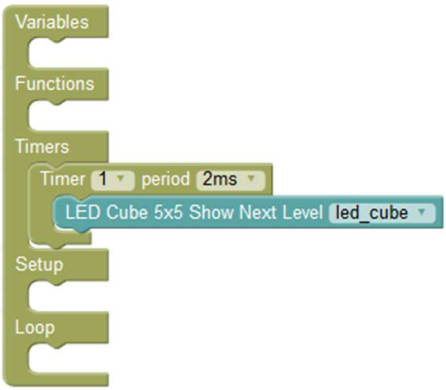
A LED kocka „csatlakozója” az arduino 22-46-ig számozott digitális lábakba kerül, míg a Bluetooth modul az 50 és 52-esbe.



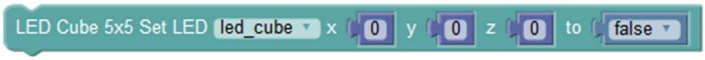
Szoftver:

A LED kocka úgy lett elkészítve, hogy egyszerre mindig egy lap világít, ezért létre kell hozni egy időzítőt, hogy gyorsan váltogassuk a világító lapokat.

A Timer 1 period értékét ezredmásodpercben (ms) adjuk meg, a 2ms-os időzítés az emberi szem számára folyamatosan világító kockát jelent, 100ms mellett már látható, ahogy a lapokon levő világító ledeket sorban kapcsolja fel az alsó lappal kezdve.



A ledeket bekapcsolni a LED Cube 5x5 Set LED blokkal lehet, paraméterei az x, y, z koordináták és a true érték, kikapcsolni a false értékkel lehet.

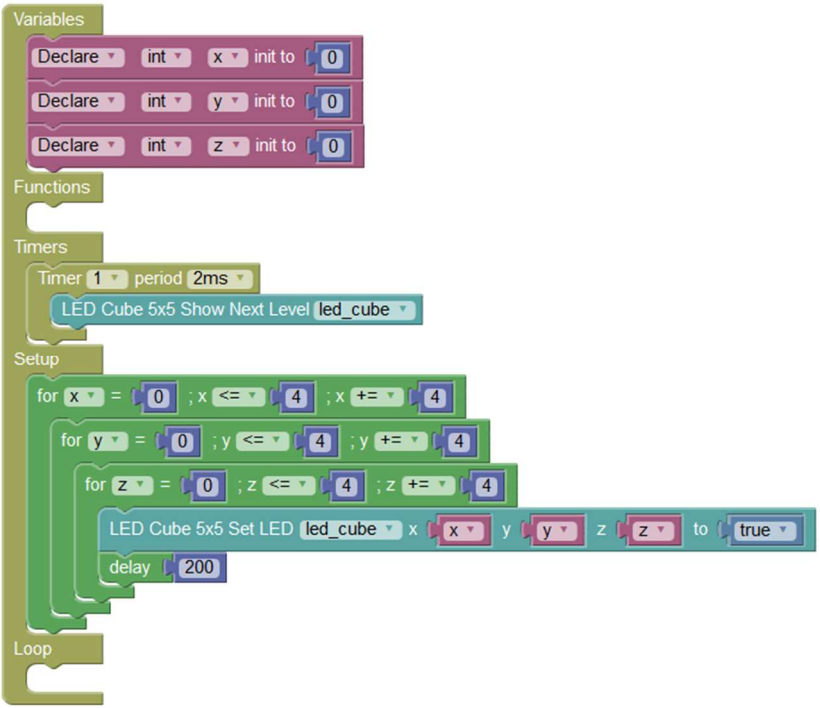


Variables:

A kocka részeinek bemutatásához érdemes ciklusokkal végigmenni a szükséges ledeken (néha az összesen), ehhez deklaráljunk x, y és z változót int típussal 0 alapértékkel.

Setup:

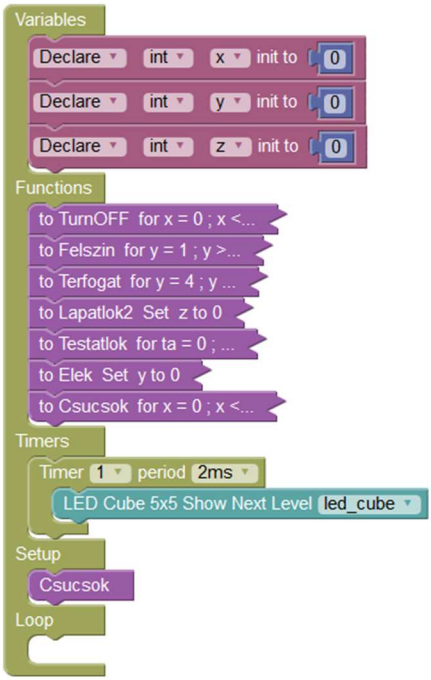
A kocka csúcsain levő ledek bekapcsolásához foglaljuk ciklusba a LED Cube 5x5 Set LED blokkot, a ciklusokat 0-4 között 4-es lépésközzel hozzuk létre.



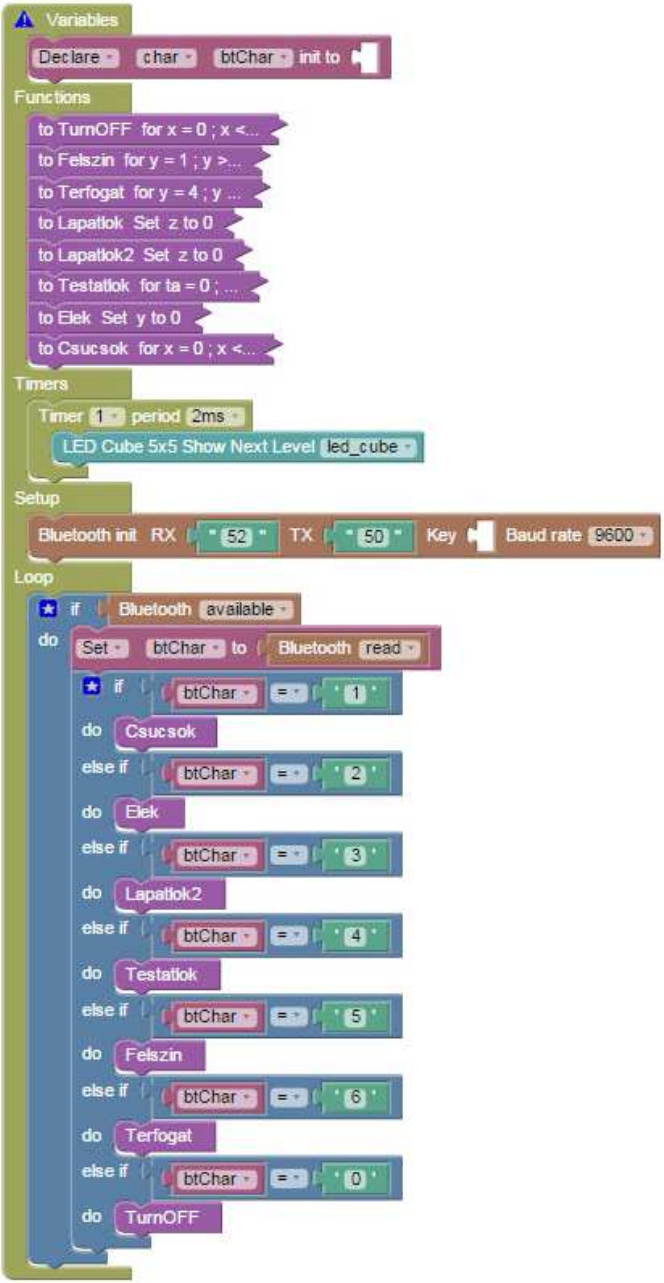
Szemléletes animációt kapunk a fenti kódban szereplő 200ms késleltetés hozzáadásával a LED-ek sorban egymás után kezdenek el világítani.

Hozzunk létre eljárásokat a kocka csúcsaihoz, éleihez, lapátlóihoz, testátlóihoz, felszínéhez és térfogatához tartozó ledek bekapcsolásához, illetve az összes led kikapcsolásához. Az eljárásokban csupán a For ciklus, a LED Cube és delay blokkok felhasználásával meg tudjuk alkotni a kódunkat.

Az eljárásokat helyezzük el a Functions részbe.

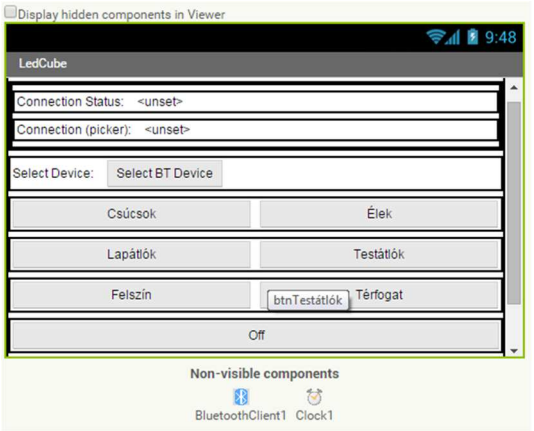


Végezetül helyezzük el a bluetooth modulhoz kapcsolódó inicializáló blokkot, amelyben megadjuk, hogy az arduino 50-es és 52-es digitális lábain akarjuk használni, valamint helyezzünk el egy kommunikációt figyelő programrészletet a Loop részbe.


Variables
Declare char btChar init to
Functions
to TurnOFF for x = 0 ; x <...
to Felszin for y = 1 ; y >...
to Terfogat for y = 4 ; y ...
to Lapatlok Set z to 0
to Lapatlok2 Set z to 0
to Testatlok for ta = 0 ; ...
to Elek Set y to 0
to Csucsok for x = 0 ; x <...
Timers
Timer 1 period 2ms
LED Cube 5x5 Show Next Level led_cube
Setup
Bluetooth init RX " 52 " TX " 50 " Key Baud rate 9600
Loop
if Bluetooth available
do Set btChar to Bluetooth read
if btChar = ' 1 '
do Csucsok
else if btChar = ' 2 '
do Elek
else if btChar = ' 3 '
do Lapatlok2
else if btChar = ' 4 '
do Testatlok
else if btChar = ' 5 '
do Felszin
else if btChar = ' 6 '
do Terfogat
else if btChar = ' 0 '
do TurnOFF

AppInventoros vezérlő alkalmazás

A LED kocka vezérléséhez először is kapcsolódni kell hozzá bluetooth-on keresztül, majd a megfelelő gombra kattintva elküldeni egy 0-6 közötti számot, amelyet az arduino feldolgoz és megjeleníti a számhoz tartozó animációt



Bluetooth kapcsolódás

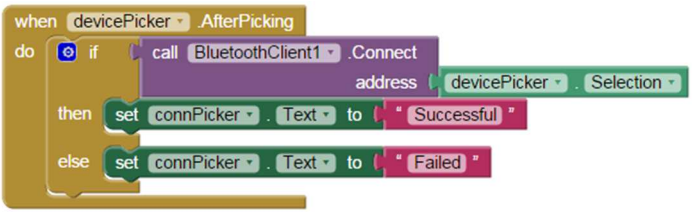
Designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
HorizontalAlignment	Layout	-	Csatlakozás állapotnak megjelenítéshez	Width: Fill parent
Label	User Interface	-	-	Text: Connection status:
Label	User Interface	connClock	Státuszinformáció	Text: <unset>
HorizontalAlignment	Layout	-	Csatlakozás sikerességének megjelenítéshez	Width: Fill parent
Label	User Interface	-	-	Text: Connection (picker):
Label	User Interface	connPicker	Sikerességi információ	Text: <unset>
HorizontalAlignment	Layout	-	Eszköz kiválasztáshoz	Width: Fill parent
Label	User Interface	-	-	Text: Select Device:
ListPicker	User Interface	devicePicker	Csatlakoztatott eszközök listájának megjelenítése	Text: Select BT Device
BluetoothClient	Connectivity	-	Csatlakozáshoz és kommunikációhoz	-
Clock	Sensors	-	Csatlakozás ellenőrzéséhez	-

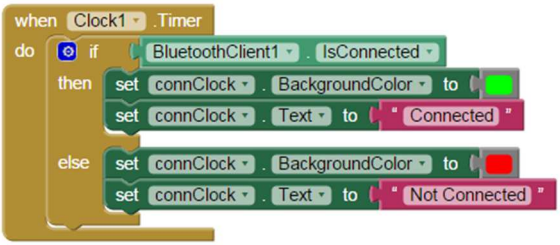
Blokkszerkesztő:

Amikor a felhasználó megérinti a bluetooth eszköz kiválasztását devicePicker.BeforePicking, feltöltjük a párosított bluetooth eszközök listáját.

A felhasználó választ egyet a párosított eszközökből, a választása után devicePicker.AfterPicking megpróbálunk csatlakozni BluetoothClient.Connect. A kapcsolódás sikerességéről tájékoztatjuk a felhasználót.



Hozzuk létre az időzítőnket is, amely másodpercenként ellenőrzi a bluetooth kapcsolatunkat.



Kocka vezérlése

Végezetül ha már rendelkezésre áll a kapcsolat, csak át kell küldenünk a LED kockát vezérlő arduinonak a megfelelő 0-6 közötti kódot.

Kiegészítő designer elemek:

Komponens	Csoport	Elnevezés	A komponens célja	Tulajdonságok
HorizontalAlignment	Layout	-	első sor gombhoz	Width: Fill parent
Button	User Interface	btnCsucsok	LED kocka vezérléshez	Text: Csúcsok Width: Fill parent
Button	User Interface	btnElek	LED kocka vezérléshez	Text: Élek Width: Fill parent
HorizontalAlignment	Layout	-	második sor gombhoz	Width: Fill parent
Button	User Interface	btnLapátlók	LED kocka vezérléshez	Text: Lapátlók Width: Fill parent
Button	User Interface	btnTestátlók	LED kocka vezérléshez	Text: Testátlók Width: Fill parent
HorizontalAlignment	Layout	-	harmadik sor gombhoz	Width: Fill parent
Button	User Interface	btnFelszín	LED kocka vezérléshez	Text: Felszín Width: Fill parent
Button	User Interface	btnTérfogat	LED kocka vezérléshez	Text: Térfogat Width: Fill parent
Button	User Interface	btnOff	LED kocka lekapcsolásához	Text: Off Width: Fill parent

Blokkszerkesztő:

A minden btn.Click eseményben hívjuk meg a BluetoothClient.SendText eljárást.