DEBRECENI EGYETEM
GAZDASÁGTUDOMÁNYI KAR
Alkalmazott Informatika és Logisztika Intézet

Alkalmazott informatikai füzetek 1.

TAKÁCS VIKTOR LÁSZLÓ

Mobiltechnológia

DU press

DEBRECENI EGYETEM
GAZDASÁGTUDOMÁNYI KAR
Alkalmazott Informatika és Logisztika Intézet

Alkalmazott informatikai füzetek 1.

TAKÁCS VIKTOR LÁSZLÓ

Mobiltechnológia

Debreceni Egyetemi Kiadó
Debrecen University Press
2017

Lektor:

Dr. Szilágyi Róbert
Domokos Péter

ISBN 978-963-318-639-8

Kiadta: a Debreceni Egyetemi Kiadó, az 1795-ben alapított
Magyar Könyvkiadók és Könyvterjesztők Egyesülésének a tagja
www.dupress.hu
Felelős kiadó: Karácsony Gyöngyi
A nyomdai munkálatokat
a Debreceni Egyetem sokszorosítóüzeme végezte 2017-ben

Tartalomjegyzék

I. Bevezető

Robert W. Sebesta kilenc kiadást megért, folyamatosan bővített, nagy sikerű könyve feltérképezi a számítástechnika/informatika története során megjelenő programozási nyelveket, és vizsgálja őket több szempontból is a kezdetektől, egészen a 20. század végéig.

Jegyzetünkben Sebesta könyve alapján elemezzük a 21. században megjelent új okos eszközök („kütyük") programozását lehetővé tévő fejlesztői környezeteket. Ezek a szempontok időtállóak, hiszen az eszközök továbbra is Neumann-architektúrájú alapgépek, a fejlesztő környezetek mindegyike pedig már meglévő, ismert programozási nyelvekre épül.

A teljesség igénye nélkül néhány példa ezek közül: MIT App Inventor 2, Eclipse, NetBeans, Android Studió.

- Az MIT App Inventor 2 a Google-MIT páros egy újabb vállalkozásának, az Androidnak a fejlesztő környezete, 2008 óta folyamatosan fejlesztett, beta verziós. Egy innovatív, kezdők számára is könnyen használható bevezető fejlesztő környezet, amely a szöveg-alapú komplex nyelvek/fejlesztő környezetek helyett teljesen vizuális, drag-and-drop technikán alapul. Az egyszerű vizuális interfész elősegíti, hogy a kezdő fejlesztők egyszerű alkalmazást készíthessenek jellemzően egy órán belül. Arra lenne hivatott az MIT és aGoogle szándékai szerint, hogy a szélesebb tömegek kezébe adja a modern, új, okos mobileszközök programozásának lehetőségét. Egy olyan programozási környezetet kínál, mely webes felületű, böngészőben fut, segítségével minimális programozási tudás birtokában is működőképes, és igen látványos, érintőképernyőre optimalizált alkalmazásokat készíthetünk okos eszközeinkre (telefonok, tablet PC-k, illetve egyéb, Android operációs rendszerrel futó informatikai eszközök).
- Az Eclipse eredetileg az IBM projektje volt, 2004-től mint önálló, nonprofit alapítvány fogta össze a fejlesztői közösséget, mely mind a mai napig nyílt. Az integrált Java fejlesztő környezet szintje a professzionális programozói színvonal, Java alapismeretekre van szükség a használatához. Egyes vélemények szerint ez lett volna hivatott ingyenes alternatívaként szolgálni a Microsoft Visual Studio-val szemben, de véleményem szerint ezt a célját nem érte el, hogy miért, azt a későbbiekben fejtem ki.
- A NetBeans a Java nyelvhez kapcsolódó grafikus interfészt biztosító programozási környezet, mely beépülő modulként több más nyelvet is támogat. Cseh hallgatói projektként indult 1996-ban, céljuk egy Delphi-hez hasonló Java fejlesztői környezet megalkotása volt. A SUN felvásárolta, majd 2000-ben nyílt forráskódúvá tette. Használata komoly programozói tudást igényel, az Eclipse-hez hasonlóan.
- Az Android Studio első - 0.1-es beta verzióját - 2013 májusában tette közzé a Google. Jelenleg a fejlesztő környezet a 2.2.3-as verziónál jár. A fejlesztő környzet 2014 decemberében értel el az 1.0-ás verziót, így átvehette a hivatalosan használt Eclipse helyét a fejlesztők fegyvertárában. Elődjével ellentétben a népszerű IntelliJ IDEA Java IDE közösségi kiadására épül és kifejezetten a Google mobil operációs rendszerére lett kihegyezve.

A fentiek közül most a Google MIT App Inventor 2 környezet elemzésével foglalkozunk részletesen.

Történeti áttekintés

Mark Friedman (google) és Hal Abelson (MIT Professzor) közösen vezették az App Inventor fejlesztését 2009-ben, amikor Hal alkotói szabdságát töltötte a Googlenél. A fejlesztés korai fázisában közreműködő Google mérnökök: Sharon Perl, Liz Looney és Ellen Spertus. Az App Inventor az MIT's Center for Mobile Learning munkatársainak felügyelete alatt webes szolgáltatásként érhető el. A közös projektben segít még az MIT's Computer Science, az Artificial Intelligence Laboratory (CSAIL) és az MIT Media Lab.

Napjainkban, 2015-től, az MIT App Inventor közösség közel 3 millió felhasználóval büszkélkedhet 195 országból. Több, mint 100 000 aktív heti felhasználó készít 7 millió android alkalmazást!

II. Az MIT App Inventor 2 elemzése

Az elemzés szempontjai a könyv első kézbevételekor szinte készen kapja az olvasó, ahogy belelapoz a tartalomjegyzékbe. Az első fejezet alfejezetei szerint indítanám az elemzést.

Programozási terület (programming domains)

Itt kell bemutatnunk azt, hogy az adott fejlesztőeszköz milyen területen való alkalmazások készítését támogatja elsősorban. Kezdetben a magas szintű programozási nyelveket tudományos célokra fejlesztették ki, majd üzleti felhasználásra, ezt követte a mesterséges intelligenciakutatásokat segítő programozási nyelvek életre hívása.

Az 1960-as években születtek meg először azok a programozási nyelvek, amelyekkel rendszerszoftvereket, operációs rendszereket lehetett készíteni. Innentől kezdve az egyes programozási nyelvek egyre univerzálisabbaká váltak, hiszen az operációs rendszerek sokban megtámogatták az implementációt.

A programozási nyelvek fejlődésének egyik legújabb területe a webprogramozás. Jelenleg a weben a legelterjedtebb programozási nyelvek a szerveroldali PHP-t, Java, ASP.NET és a kliensen futó Java, de a tendencia az, hogy ennek a támogatása megszűnőben van, helyette a Javascript terjed.

A 20. század végére a számítógép háztartási eszközzé vált és a programozás sem csak néhány professzionálisan ezzel foglalkozó ember kiváltsága, hanem tananyag az iskolában. Nyilván fő „lakossági” felhasználása a szórakozás, illetve a weben való kommunikáció, kapcsolatteremtés és kapcsolattartás. A szélessávú Internet elterjedése megadta a végső lökést a webprogramozásra alkalmas nyelvek fejlődésének.

Napjaink új programozási területe az okos mobileszközök.

További tendencia napjainkban az egyes programozási területek összefonódása. Régóta foglalkozom már kisvállalkozások számára gazdasági ügyviteli alkalmazások készítésével. Sokáig igen könnyen el lehetett boldogulni ezen a piacon a Visual Studio-val, mint fejlesztői környezettel. Az irodai alkalmazások legtöbbje táblázatokat, dokumentumokat, kisebb adatbázisokat reprodukál. Ezek programozása asztali számítógépen megoldott az említett környezetben. Viszont az a vállalkozói réteg, mely ilyen üzleteket visz, magánemberként többnyire megengedheti magának, (mi több, sokszor elvárás a körben, melyben mozognak) a divattrendek követését úgy a számítástechnikai vagy mobiltechnológiai eszközökben, mint az élet más színterén. Hamarosan meg fog jelenni részükről az igény – ha nem merült már föl most – hogy az okos kütyüiken intézhessék a vállalkozásuk ügyeit, láthassák, akárhol is vannak, mi történik odabenn az irodában. Természetesen nem szeretnék azt hallani, hogy mindazokat az alkalmazásokat, amit eddig használtak, le kell cserélniük valami teljesen újra emiatt. Az pénz és időveszteség lenne, az alkalmazottaknak új dolgot kell tanulniuk, ami újabb problémákat vet fel.

Nyelvértékelési/osztályozási kritériumok (language evaluation criteria)

Sebesta ennél a fejezetnél egy táblázatot közöl [Sebesta, 26], melyet saját, szabad fordításban a következőképp értelmeztem:

1.1 táblázat Nyelvosztályozási kritériumok, és a jellemzők, melyek meghatározzák őket.

	KRITÉRIUM		
Jellemző	OLVASHATÓSÁG	ÍRHATÓSÁG	FENNTARTHATÓSÁG
Egyszerűség[1]	•	•	•
Ortogonalitás[2]	•	•	•
Adattípusok[3]	•	•	•
Szintaktikai tervezés[4]	•	•	•
Absztrakciók támogatottsága[5]		•	•
Expresszivitás[6]		•	•
Típuskényszerítés[7]			•
Kivételkezelés[8]			•
Alternatív címkézés[9]			•

Egyszerűség: NetBeans és Eclipse esetében nincsenek előre gyártott komponensek a Java nyelvhez képest, az MIT App Inventor 2-ben az Androidos eszközök specialitásainak megfelelően a legegyszerűbb program is tartalmazza az érintőképernyő kezelését, virtuális billentyűzetet, a visual

1 A programozási nyelvben le kell-e programozni mindent, vagy vannak olyan előregyártott komponensek, melyeket elsajátítva a programozó elegánsabb, hatékonyabb kódot tud készíteni.

2 Túl sok beépített komponens megléte egy nyelvben ahhoz vezethet, hogy a programozó nem tudja valamennyit elsajátítani. Jóval hatékonyabb lehet a munka, ha egy kisebb halmaz áll rendelkezésre ezekből, valamint ésszerű szabályok egy csoportja, melyek segítségével az előbbiek összekombinálhatók.

3 A nyelvben meglévő adattípusok.

4 Különös tekintetet kell fordítanunk a szintaktikai tervezés során programunk következő elemeire: azonosítást szolgáló kifejezések, speciális szavak, formulák és jelentéseik. Mindezek nagyban befolyásolják a program olvashatóságát.

5 A programozási nyelvek alapvetően kétféle absztrakciót támogatnak: adat- és folyamatabsztrakciót. Az absztrakció a modern programozási nyelvek tervezésénél kulcsfogalom. Az absztrakció foka az adott programnyelv nagyon fontos jellemzője.

6 Egy adott nyelvben lévő konvenciók sokszínűsége a kód költségének csökkentése érdekében

7 Hogy egy nyelv képes-e és a program futásának mely szakaszában képes típuskényszerítésre, nagyban befolyásolja a program költségét, s így fenntarthatóságát. A futásidejű típuskényszerítés drágább, mint a fordítás ideje alatti.

8 A program azon képessége, hogy futás-idejű hibákat „elcsípjen”.

9 A magyar szakterminológiát nem megfelelően ismerve, ezt a kifejezést így fordítottam. Azt a jelenséget írja le, amikor különböző nevekkel a memóriaterület ugyanazon területére hivatkozunk, és akár dolgozunk is velük. Nyilván ez utóbbi okozhat váratlan, akár nem kívánatos következményeket, hiszen a címnél tárolt érték – pl. egy változó esetében - módosul, és a többi, konkurens név is a megváltozott értéket kapja vissza a továbbiakban.

nyelvből adódó sajátosságok miatt nem kell forráskódot írnunk, egyetlen sort sem, a felhasználói interfészt drag and drop módszerrel rakjuk össze, illetve több csoportba sorolható előregyártott komponenseket tartalmaz:

- Alapkomponensek (User Interface): gomb, címke, szövegdoboz, kép, check-box, jelszó-szövegdoboz, csúszka, dátum választó, idő választó, választó lista, legördülő lista, lista, üzenetkezelő;
- Képernyőelrendezés: vízszintes, függőleges, táblázatos, görgethető vízszintes, görgethető függőleges;
- Médiakomponensek: videó rögzítő, kamera, képválasztó, zene lejátszó, hang, hangrögzítő, hangfelismerő, szövegfelolvasó, video lejátszó, Yandex fordító;
- Animációs komponensek: vászon, ball sprite, image sprite;
- Érzékelők: gyorsulásmérő, vonalkódolvasó, óra, gyroszkóp, helymeghatározó, NFC, irányultságmérő, lépésszámláló, távolságmérő;
- Közösségi komponensek: telefonkönyv, e-mail, telefonhívás, telefonszám választó, megosztás, szöveges üzenet, twitter;
- Tárolás: fájl, FusionTable, TinyDB, TinyWebDB
- Kapcsolat: feladat indító, bluetooth kliens, bluetooth szerver, web
- LEGO® Mindstorms®(Nxt és Ev3 komponensek): színérzékelő, hangérzékelő, tapintásérzékelő, fényérzékelő, ultrahangos érzékelő, közvetlen parancsok, robotvezérlő (motorirányítás – mozgás és forgás);
- Kísérleti: FirebaseDB;
- Extension: ide lehet saját, vagy harmadik féltől származó vezérlőt importálni.

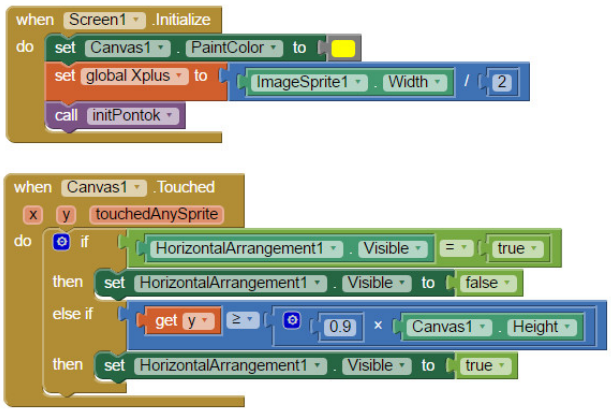
Ortogonalitás: Az MIT App Inventor 2 kevés számú primitív adattípussal rendelkezik. *Lásd adattípusok*. A kifejezéseket jellemzően egyféleképpen lehet felépíteni.

Operátorok:

- Matematikai: relációs jelek, összeadás, kivonás, szorzás, osztás, gyökvonás, négyzetgyök, véletlen tört 0 és 1 között, véletlen egész, negálás, minimum, maximum, hányados, egészrész, törtrész, kerekítés (fel/le), hatványozás, *e* hatványai, logaritmus, szögfüggvények, számformázás, szám-e;
- Szöveg: azonosság, konkatenáció, szöveg konstruktor, hossz, szövegvizsgálat, szóközlevágás, kicsi/nagybetűssé alakítás, öt hasonló szövegdarabolási mód, részszöveg keresés, részsztring bányászat;
- Logikai: igaz, hamis, egyenlő, és, vagy, negáció;
- Listák: létrehozás, elemkiválasztás, csere, listából törlés, bővítés, tartalmazás, listák összefűzése, elem pozíciója, véletlen elemválasztás, üres-e a lista, lista-e, lista <-> csv sor átalakítás és lista <-> csv tábla átalakítás, érték keresése kulcs-érték párokból álló listában kulcs alapján;
- RGB színkeverés, illetve szín bontás RGBA listára

A szövegoperátorok között átfedéseket találunk, ez korábban rontotta az ortogonalitást, ám a Google Blockly-val bevezetett mutálódó blokkok lehetősége megoldotta problémát. Az MIT App

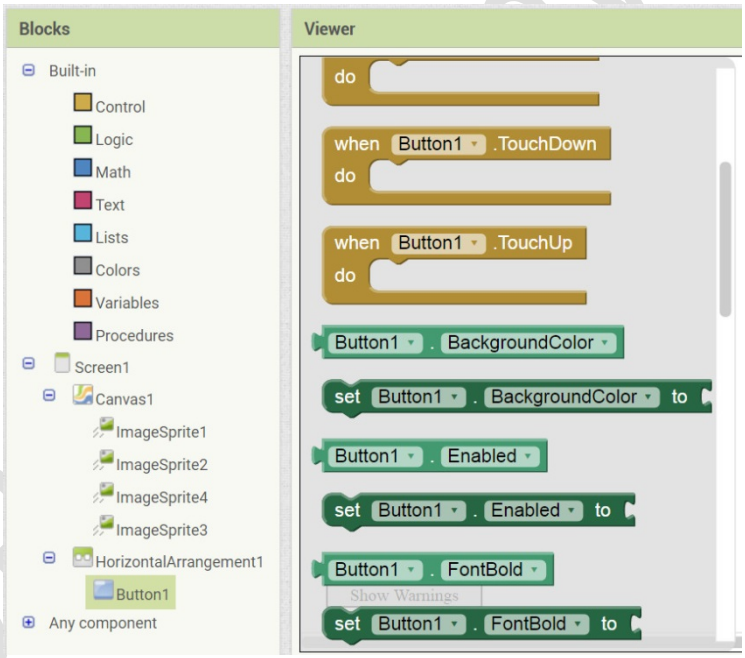
Inventor 2-ben egy kisebb halmaz áll rendelkezésre nyelvi elemekből. Ezek kombinálásához a fejlesztő környezet látványos segítséget ad.



1. ábra Nyelvi elemek kombinálása

Adattípusok: boolean (true,false), szám, szöveg, szín (konstansok).

A **szintaktikai tervezés** során a fejlesztő környezetben választható elemként grafikusan megjelennek a vezérlők eseményei, eljárásai és tulajdonságai. Így ezekkel nem kell foglalkozni, nem kell a szintakszist kitalálni.



2. ábra A vezérlők eseményei, eljárásai és tulajdonságai

Ahogy az ortogonalitásnál beszúrt képen (1. ábra) is látszik, a fejlesztőkörnyezet elkészíti a szükséges paramétereket is.

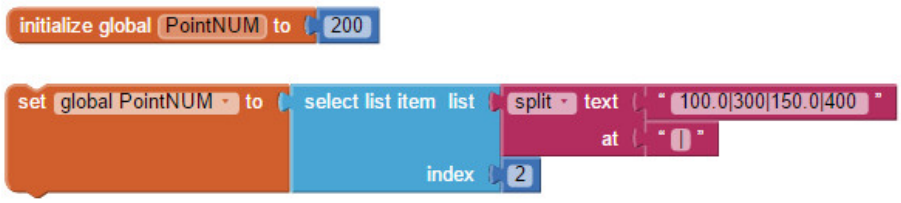
Absztrakciók támogatottsága: Az MIT App Inventor 2-ben definiálhatunk eljárásokat és függvényeket is, illetve a listák felhasználásával magasabb szintű adatabsztrakció (mint a binárisa fa) is megvalósítható, emellett a speciális adattárolók TinyDB és TinyWebDB (kulcs-érték párok tárolására) felhasználásával alapszintű adatbáziskezelés is megvalósítható, de használhatjuk a Google FusionTable szolgáltatását is. Lehetőségünk van a memóriában oszlop alapú adatfeldolgozásra is szöveg fájlokat alapul véve.

Expresszivitás: A nyelvben az alábbi vezérlő blokkok léteznek: if|elseif*|else, speciális if (egy logikai kifejezés értékének megfelelően eltérő program blokkokat lehet végrehajtatni, illetve

ennek megfelelőek a visszatérési értékek is), do result, for each (listaelemekre és számokra), while, close screen (eredménnyel is), open another screen (eredménnyel is).

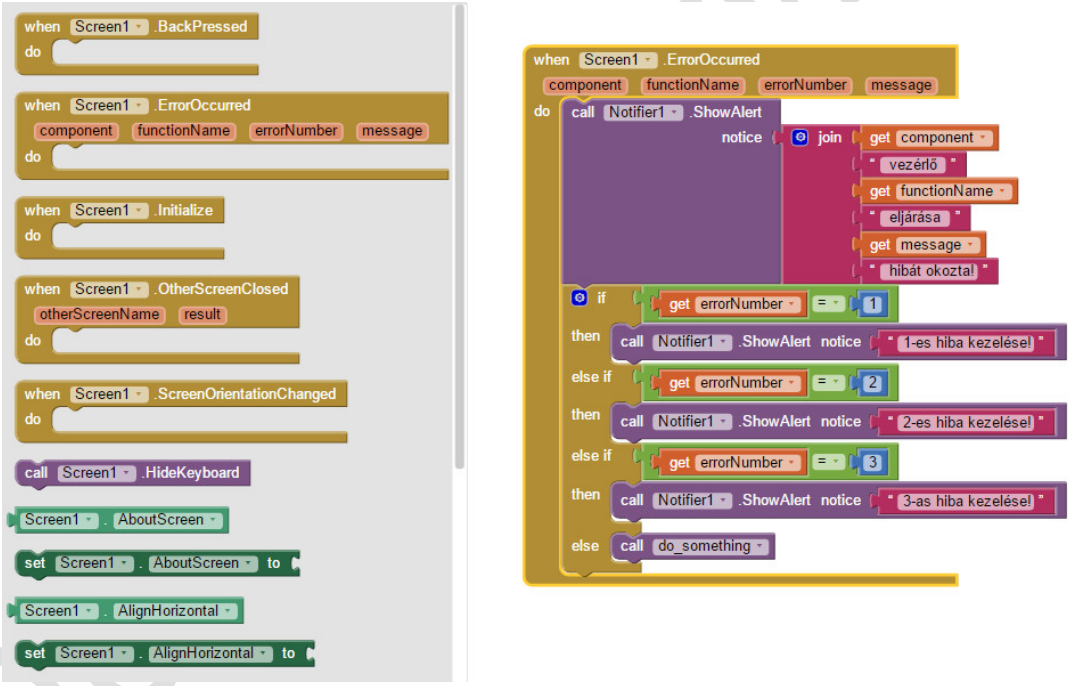
Az MIT App Inventor 2-ben globális (screen) és lokális (procedure) változókat, valamint paramétereket használhatunk.

Típuskényszerítés: Az MIT App Inventor 2-ben megvan a JAVA nyelvben megszokott, azaz helyes az „X=3+12.0” kifejezés, illetve a sztringek létrehozása és darabolása során sem kell figyelni a változók típusára:



3. ábra Típuskényszerítés

Kivételkezelés: Az MIT App Inventor 2-ben képernyő (screen) alapú kivételkezelésre találtam utalást, minden esetben az adott képernyő objektum ErrorOccurred eseménye kerül meghívásra a hibával kapcsolatos értékekkel/környezettel, melyeket megjeleníthetünk, illetve kezelhetünk.



4. ábra Kivételkezelés

Alternatív címkézés: Teljességgel hiányzik a nyelvből.

A nyelv választást befolyásoló tényezők (influences on language design)

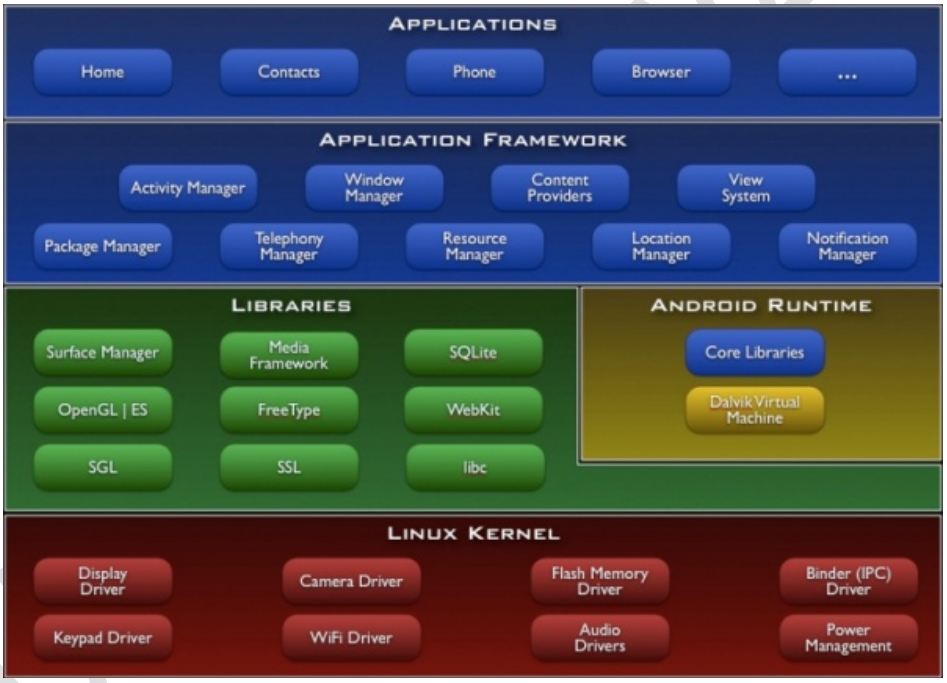
Ezekből a két legfontosabb tényező a számítógép architektúra illetve a programtervezési módszertanok. Előbbi tekintetében elmondhatjuk, az első elektronikus számítógépek óta „egyeduralkodóak” a Neumann-architektúrájú számítógépek. Okos eszközeink is Neumann-gépek.

A szoftverfejlesztési módszertanokról pedig annyit érdemes megjegyeznünk, hogy szinte törvényszerűen fejlődtek, szoros kölcsönhatásban a programnyelvek fejlődésével, sokszor „tyúk és tojás” dilemmát okozva, vajon melyik volt előbb: az első objektumorientált nyelv például, vagy a szemlélet, a módszer?

Az Android operációs rendszert futtató eszközök esetünkben speciális, jól körülhatárolható rendszerek. Az Android platform abból a célból született, hogy egységes nyílt forrású operációs rendszere legyen a mobil eszközöknek (és itt elsősorban a smartphone és tablet kategóriát kell érteni, mintsem egyszerű mobiltelefonokat). Az elképzelés alapja egy Linux alapú operációs rendszer volt, amelyet úgy alakítanak át, hogy képes legyen problémák nélkül kezelni a mobil eszközök integrált hardvereit (érintőképernyő, WiFi, HSDPA, Bluetooth, stb.). Az első lépéseknél nem volt szó Java nyelvről, azonban a Google 2005 júliusában megvásárolta az Android nevű céget, és új irányt adott a fejlesztésnek: a Linux kernel fölé egy virtuális gép került, amely a felhasználói felület kezeléséért és az alkalmazások futtatásáért felelős.

A platform alapját a Linux kernel adja, amely tartalmazza a hardver által kezelendő eszközök meghajtó programjait, a kis méretű kernel adja a memória kezelését, a folyamatok ütemezését és az alacsony fogyasztást elősegítő teljesítmény-kezelést is.

A kernel szolgáltatásait használják a Linux rendszerekben meglévő különféle programkönyvtárak, mint a libc, az SSL vagy az SQLite; ezek C/C++ nyelven vannak megvalósítva, és a Linux kernelen futnak közvetlenül. Részben ezekre épül a Dalvik virtuális gép, amely egyáltalán nem kompatibilis a Sun virtuális gépével, teljesen más az utasítás készlete, és más bináris programot futtat. A virtuális gép más, mint a Java alatti megszokott virtuális gép, vagyis a Java csak mint nyelv jelenik meg!



5. ábra Android rendszer szerkezete [2]

Az 5. ábra kék színnel jelölt részeiben már csak Java forrást találunk, amelyet a virtuális gép futtat, s ez adja az Android lényegét: a látható és tapintható operációs rendszert, illetve a futó programokat. A virtuális gép akár teljesen elrejti a Linux által használt fájlrendszert, és csak az Android Runtime által biztosított fájlrendszert láthatjuk.

Programozási nyelvek csoportosítása (language categories)

A programozási nyelveket legtöbbször a következő négy kategória valamelyikébe soroljuk be: imperatív, funkcionális, logikai vagy objektumorientált. Az egyes kategóriák alkategóriáinak meghatározása vita tárgya sokszor a szakirodalomban.

Az Android platform futtatható állományai egyedi szerkezetű, Java nyelven megírt forráskódot tartalmazó köztes kódok. Míg az Eclipse és NetBeans fejlesztő környezetekbe ez az egyedi

szerkezet és nyelv nyersen került implementálásra az Android SDK különböző verzióin keresztül (jelenleg kilencnél tartunk), addig a Google által rendelkezésünkre bocsájtott MIT App Inventor 2 fejlesztői környezet önállóan tartalmaz egy magasabb szintű, teljes mértékben vizuális nyelvet. Nyilván objektumorientált módszerrel programozzuk Android-os eszközeinket mindhárom fejlesztői környezetben, ugyanakkor az MIT App Inventor 2 objektumorientáltsága mellett grafikus nyelv is.

Kompromisszumok (language design trade-offs)

A nyelvosztályozási kritériumok egymásnak való ellentmondása miatt a nyelvek különböző kompromisszumokat tartalmaznak, melyet a nyelv tervezésekor a nyelvet megalkotók építenek föl, és a programozónak ezekhez alkalmazkodnia kell. Igen erősen befolyásolja ez a programozókat, hogy milyen célból, mikor mely nyelvet választják programozási eszközükként.

A béta állapotú MIT App Inventor 2 esetében a szintaktikával keveset kell foglalkozni, ellenben messze nem lehet úgy kihasználni az Android SDK által nyújtott összes lehetőséget, mint a NetBeans vagy az Eclipse esetén.

Implemetációs módszerek (implementation methods)

A számítógép nyilvánvalóan nem tud értelmezni magas szintű programnyelveket. A programozási környezetek, melyekkel a szoftverfejlesztő dolgozik a programozó (az ember) számára elsajátítható nyelv, megfelelő szókinccsel és szabályrendszerrel, melyet a gép nyelvére is le kell tudnia fordítani a programozási környezetnek. A közvetlen gépi kód és a magas szintű nyelv közt azonban vannak különböző rétegek még, melyeket összefoglalóan implementációs rendszernek nevezhetünk, és mint virtuális gép működik a számítógép és a programozó közt. A nyelvimplementációs rendszerek mellett működik minden gépen az operációs rendszer, mely nagyban megsegíti a programozási nyelvet már meglévő primitívjeivel, egyszerű függvényeivel. Ezek lehetővé tesznek olyan alapfunkciókat, mint fájlkezelés, input-output operációk, rendszererőforrások használata. Ezen rétegek alatt van a makroinstrukciók interpretere, és legbelül maga a gép.

Az implementálás háromféle módon történhet.

- Compiler implementation – az implementációs módszerek ezen (egyik extrém) esete, a program gépi kódra való fordítása több lépcsőben történik. Folyamata szerint a forrásprogram (az eredeti nyelven írt program) lexikai, majd szintaktikai elemzésen megy át, a köztes kód generátor és szemantikai ellenőrzés után előálló köztes kódot a gépi kód generátor fordítja végül le a számítógép számára, mely közben megkapja a bejövő adatokat is, és azt feldolgozva közli a futási eredményt. Előnye a gyors futtatás.
- Egyszerű interpretálás – a forrásprogramból és a bejövő adatból az interpreter szolgáltat futási eredményt. Itt nincs fordítás, csak interpretálás, futási időben. Előnye, hogy megkönnyíti a futási időben való hibakeresést. Hátránya viszont a lassú végrehajtás.
- Hibrid implemetációs rendszerek – sok implementációs rendszer működik a fenti két módszer kompromisszumon alapuló egyvelegeként. A forrásprogram lexikai majd szintaktikai analízise után a köztes kód generátor előállítja a köztes kódot, melyet az interpreter kap meg és a bejövő adatokat feldolgozva futási eredményt produkál. A Java környezetek mind hibrid implementációs rendszerűek. A JIT (Just-In-Time) implementációs rendszer egy köztes nyelvre fordítja a forrásprogramot, majd futás közben ezt a köztes nyelvi kódot fordítja gépi kódra. Ezen típusú implementációs

rendszerek esetében a fejlesztőkörnyezet általában lehetőséget biztosít mind a fordításra, mind az egyszerű interpretálásra.

A preprocesszorok olyan programok, amelyek a program fordítása előtt dolgoznak. Ezek a fordítási direktívák beágyazott programrészletek, melyek többnyire külső fájlokra, programokra való hivatkozásokat tartalmaznak.

Programozási környezet (programming environments)

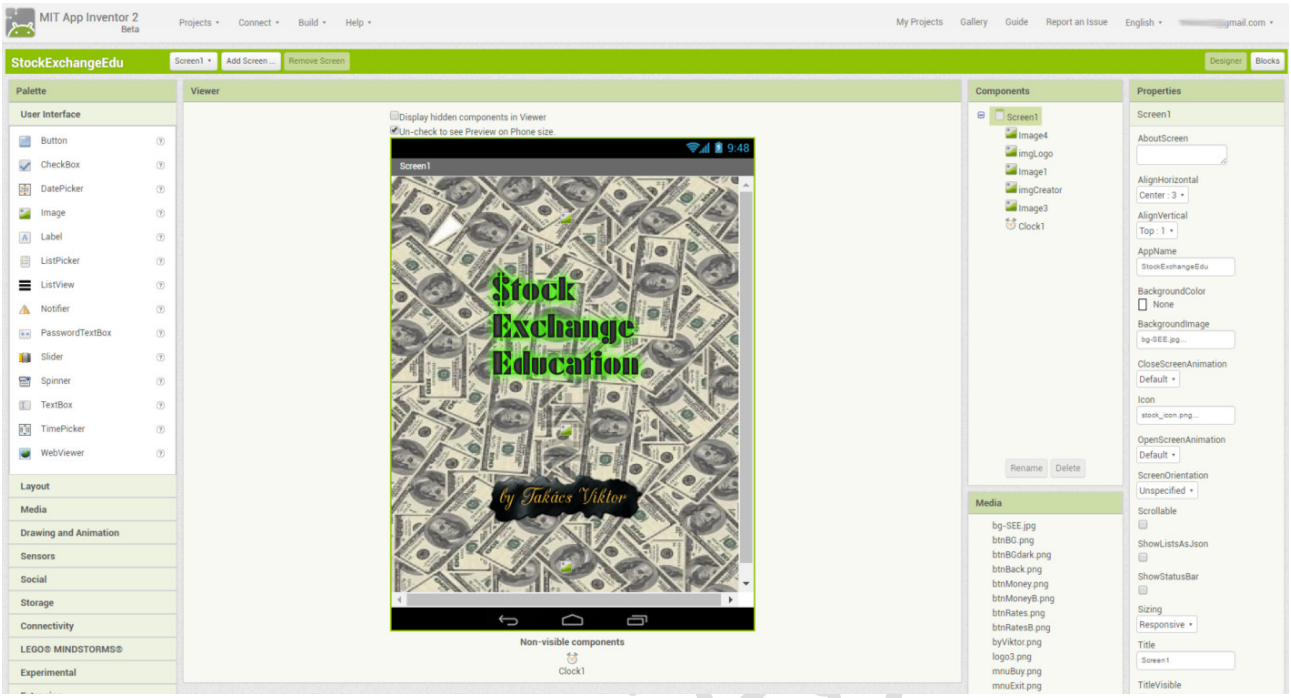
Ez azoknak az eszközöknek a gyűjteménye, melyeket a programozáskor használhat a programozó. Minimálisan tartalmaznia kell egy szövegszerkesztőt, fájlkezelőt, linkert és fordítót. Ahogy a számítógépek egyre inkább felhasználóbarát arcot kezdtek ölteni, a szoftver rendszerek egyre komplexebbé válnak, amely egyrészt mint hajtóerő is szerepet játszik abban, hogy a fejlesztő eszközök megkönnyítsék a fejlesztők munkáját, másrészt ennek a tendenciának mint következményeként megjelent az igény arra is, hogy a programozási környezetek minél tetszetősebbek legyenek és minél több kényelmi funkcióval segítsék, gyorsítsák a programozó munkáját. A modern programozási környezetek nagyban megkönnyítik a szoftverfejlesztést és karbantartást a következő elemek révén: „helyesírás-ellenőrzés”, szintaktikai kiegészítések, grafikus felület, ablakos interfész, univerzalitás – több programozási nyelvet „befogadó” környezetek (pl. NetBeans). Tehát manapság a programozási nyelvek jellemzői mellett egyre fontosabb szerepet kap a programozási környezet minősége is.

A legrégibb programozási környezet a UNIX volt a ’70-es évektől. Hordozhatóság jellemezte és legfontosabb előnye volt, hogy egységes interfészt nyújtott különböző programozási nyelvekhez. Manapság ellátták grafikus interfésszel hasonlóan a Windows illetve Macontish rendszerekhez. A ’90-es években jelentek meg a vizuális fejlesztői környezetek, mint Borland JBuilder, Delphi, Microsoft Visual Studio, NetBeans. Grafikus interfészt nyújtanak, többféle programozási nyelvet támogatnak. Sebesta könyvének 2009-ben megjelent, 9. kiadása szerint relatív legfrissebb evolúciós lépcsőfok a szoftverfejlesztő környezetek evolúciójában a Microsoft Visual Studio .NET. Ezt mostanra túlléptük.

A szoftverfejlesztésben új paradigmák jelentek meg (SOA, Agile), ugyanakkor felgyorsult életünkben az informatikai megoldásokkal, alkalmazásokkal szembeni elvárások, kihívások gyors változása a fejlesztő környezetek gyors evolúcióját is sürgeti. Úgy vélem, ezt ismerhette fel az MIT-Google összefogás is, amikor célul tűzte ki egy olyan környezet megalkotását, melyben a nyelv szinte másodlagossá válva rejtve marad a fejlesztő számára, és minden eddiginél kényelmesebb kódírást tesz lehetővé mobileszközök számára. Gyakorlatilag a programtervezés egyben szoftverfejlesztés is. Ezek a fajta környezetek lehetővé teszik egyéb területen működő szakemberek számára is a gyors programtervezést és készítést, legyen az a mindennapi élet egy területén működő kisebb, vagy a munkához, szórakozáshoz kapcsolódó komolyabb alkalmazás.

III. Az App Inventor 2 fejlesztőkörnyezet kezelőfelülete

fejlesztő környezet 7 fontosabb részből áll, melyet az alábbi ábrán mutatunk be.



6. ábra Az app Inventor 2 fejlesztő környezet

1. A **címsor**ban lehet a projektjeinkkel kapcsolatos tevékenységeket végezni, illetve egy-egy project fordítását az androidos eszközünkre.
2. Az alkalmazás nevével kezdődő **zöld sáv**on lehet az alkalmazás képernyői között váltani, azokat létrehozni, törölni. Minden alkalmazás legalább egy Screen1 képernyőt tartalmaz, mely nem nevezhető át, ám a további képernyők létrehozása és elnevezése a fejlesztőre van bízva. A jobb szélén helyezkedik el a Tervező és a Kódszerkesztő közötti váltást elősegítő gombok.
3. A **Viewer** panelen lehet az alkalmazás kinézetéhez kapcsolódó elemeket vizuálisan összerakni.
4. Bal oldalon **Palette** tartalmazza különböző csoportokban az alkalmazás fejlesztéséhez felhasználható kölönböző vezérlőket/komponenseket. Innen tudjuk a Viwer-be Drag-and-Drop technikával átemelni.
5. A **Components** panel alatt hierarchikus fa struktúrába rendezve (szülő-gyerekek) látszik az összes Viwerben használt komponens, akkor is, ha történetesen az adott képernyőn éppen nem fér ki.
6. A **Media** panel tartalmazza az alkalmazáshoz használt media fájlokat (kép/hang/videó). Ide teljes weboldalakat is fel tudunk tölteni a szükséges javascript kódokkal együtt.
7. A **Properties** panelen tervezési időben tudjuk állítani a kiválasztott komponens tulajdonságainak jelentős részét.

IV. Tanulókártyák

Az továbbiakban egy általunk is kipróbált és jól bevált tanulásmódszertani eszközt adunk az olvasó kezébe.

A tananyagot nem fejezetekre osztjuk, hanem úgynevezett tanulókártyákra.

Egy-egy kártyán egy adott problémát mutatunk be, melyet a mobileszközön le szeretnénk programozni. A tanulókártya segítséget ad egy próbaalkalmazáson keresztül, hogyan valósítjuk meg azt, milyen eszközöket kell ehhez meghívnunk a tervező nézetben, és hogyan programozzuk ezeket az eszközöket a blokkszerkesztőben a cél érdekében.

Némely tanulókártya végén feladatokat, kérdéseket is találunk, mely segítségével tovább gondolhatjuk, vagy saját alkalmazásban adaptálhatjuk az adott problémát.

A módszer, és az eredeti angol nyelvű tanulókártyák az MIT terméke, ők rendszeresen használják a kártyáikat kurzusaikon, illetve honlapjukon közzé is tették Creative Commons Attribution-ShareAlike 3.0 Unported License alatt a kártyákat az önálló tanulás segítésére. Ebben a fejezetben tehát részint az MIT által közzétett kártyák fordításait találjuk, illetve további, saját problémafelvetéseket és ezekre általunk készített kártyákat is.

Az ajánlott módszer kezdőknek az, hogy minden egyes kártyán végighaladjanak, és megpróbálják adaptálni az egyes kártyákon látottakat. Akit azonban csak egy bizonyos probléma megvalósítása érdekel, természetesen kezdheti csak azzal, mert az egyes kártyákon lévő tartalmak csak lazán épülnek egymásra.

Nem adunk közzé kész, teljes programokat, melyekben megtalálhatók lennének ezen kódrészletek. Mi inkább az olvasót szeretnénk arra bátorítani, hogy építse be ezeket az ötleteket saját alkalmazásaiba!

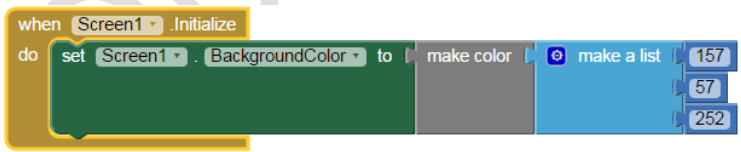
Színek beállítása

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

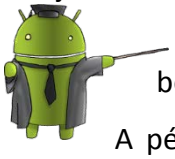
Designer elemek:

Komponens	Csoport	A komponens célja
-	-	-

Blokkszerkesztő:



Mit jelent ez a kód?

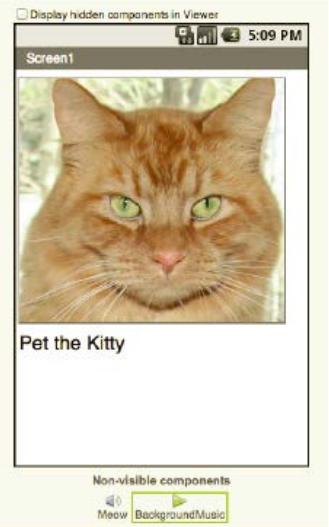


A make color számára egy három számból álló listát kell átadni. A három szám a beállítani kívánt szín RGB kódja.

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.

Tudnánk készíteni türkiz háttérszínt?

Hangfájlok hozzáadása

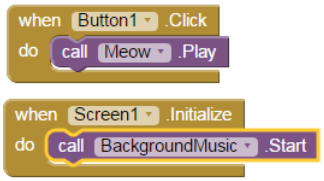


Adjunk rövid hangeffektet, például egy macskanyávogás hangot, amikor megérintjük a macska képét, vagy hosszabb hangfájlt, például háttérzenét a projektünkhöz!

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:

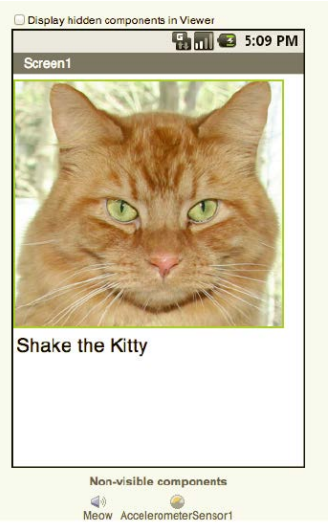


Mit jelent ez a kód?

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

Amikor az alkalmazás elindul a mobilon Screen1.Initialize, akkor a beállított zene elindul BackgroundMusic.Start.

Telefon rázása

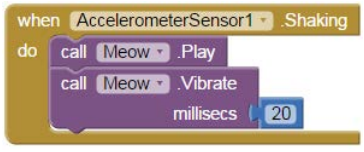


Történjen valami, amikor megrázzuk a telefonunkat!

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

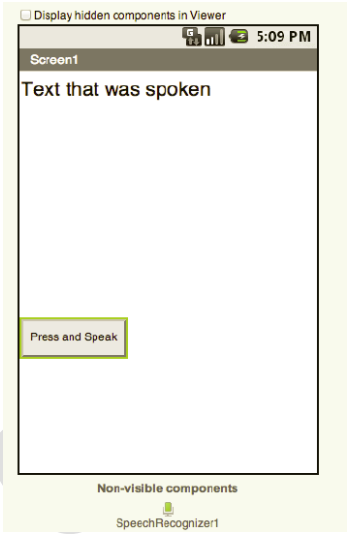
Blokkszerkesztő:



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.

Beszédfelismerés

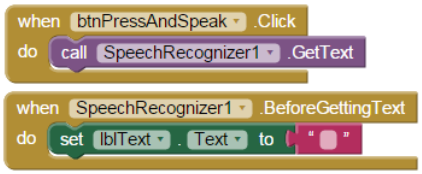


Írassuk ki a képernyőre a mondott szöveget!

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:



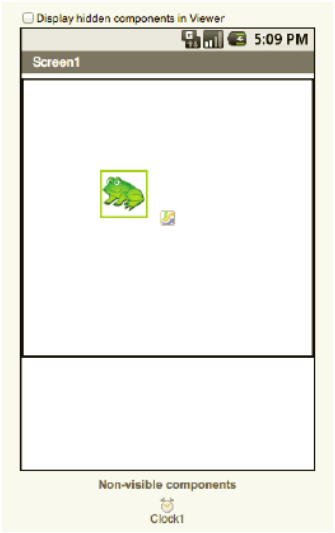
Mit jelent ez a kód?

Amikor a gombot megérintjük btnPressAndSpeak.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

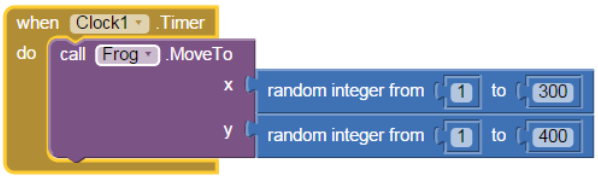


Generáljunk véletlenszerűen számokat, melyekkel, mint (x,y) koordinátákkal Sprite-okat tudunk pozícionálni a képernyőn!

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:

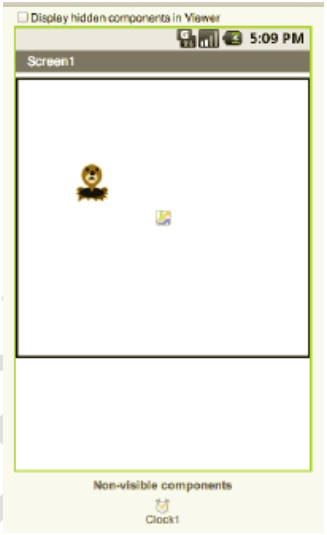


Mit jelent ez a kód?

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?

Időzített Sprite mozgatás

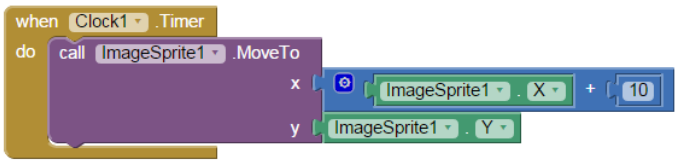


Megadott időközönként mozgassuk el a Sprite-ot!

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:



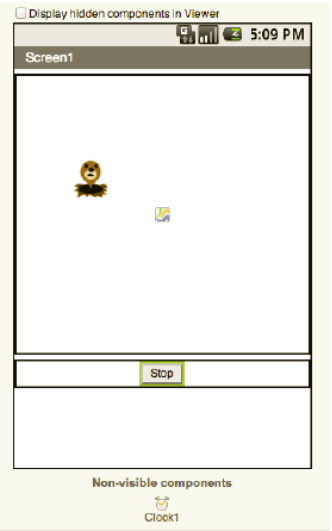
Mit jelent ez a kód?

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

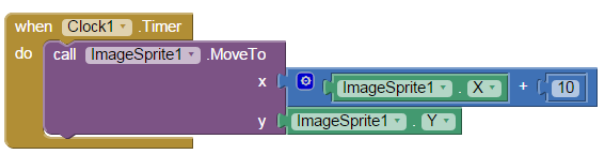


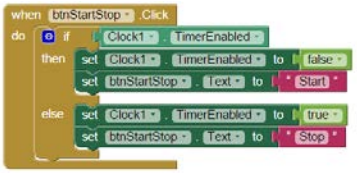
Azt szeretnénk, ha az előző példában szereplő időzítőt leállíthatnánk, amikor mi akarjuk egy Start/Stop gombbal.

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:



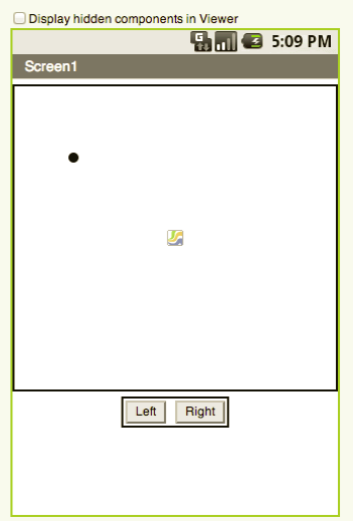


Mit jelent ez a kód?

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.

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:

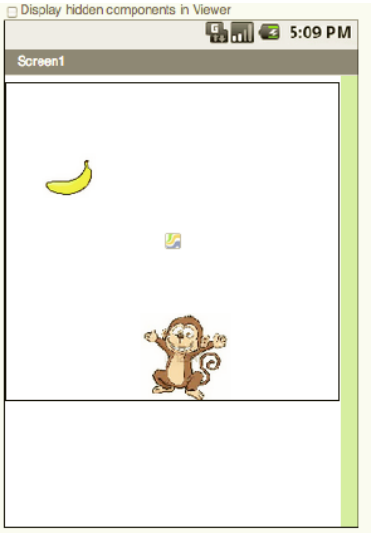
Mit jelent ez a kód?

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.

Sprite mozgatása ujjal

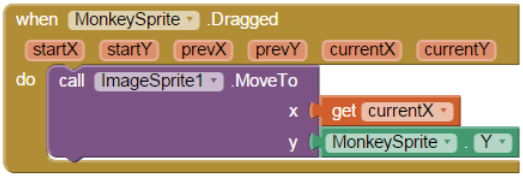


Mozgassuk a sprite-ot egyik oldaltól a másikig vízszintesen az ujjunkkal!

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.

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.

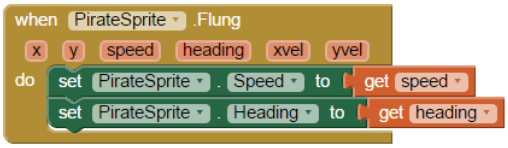
Suhintás

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:

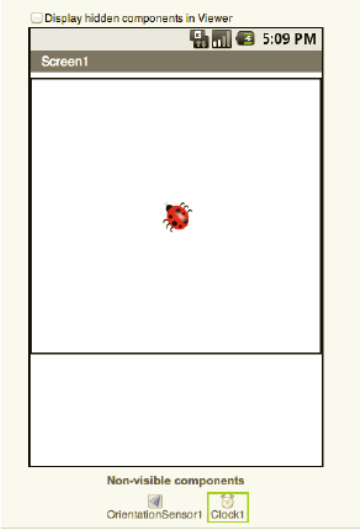


Mit jelent ez a kód?

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

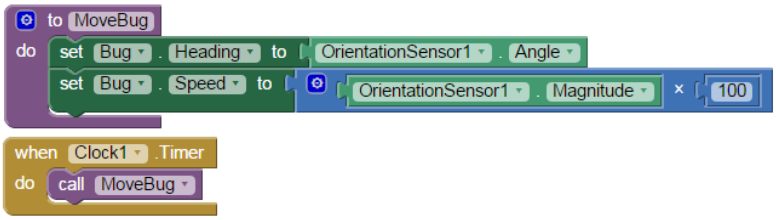


Mozgassunk egy sprite-ot a telefon billegtetésével!

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.

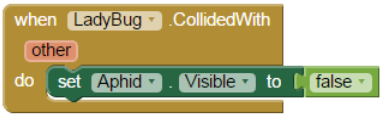
Sprite-ok ütközése

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?

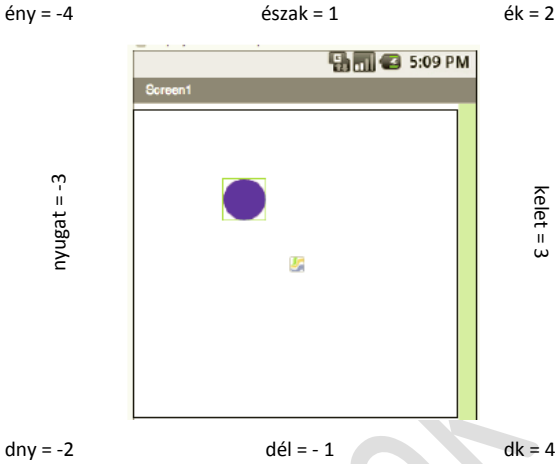
A LadyBug.CollidedWith esemény akkor aktiválódik, amikor a Katica érintkezik a levéltetűvel. Ennek eredménye, hogy a levéltetű eltűnik.

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

Pattogó labda

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

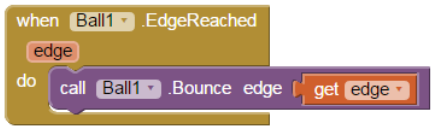
Az szélekhez hozzárendelt értékek (1,2,3,4,-1,-2,-3,-4).



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?

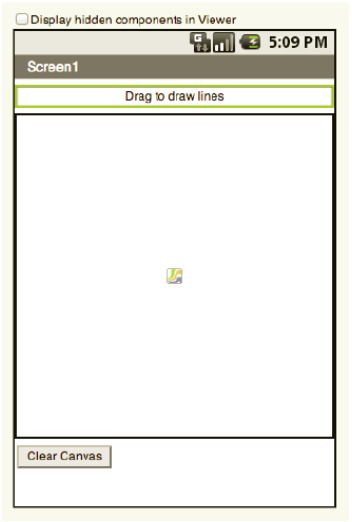
A Canvas vezérlő minden széle (edge) rendelkezik saját numerikus értékkel.

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.

Rajzolás a Canvas-ra

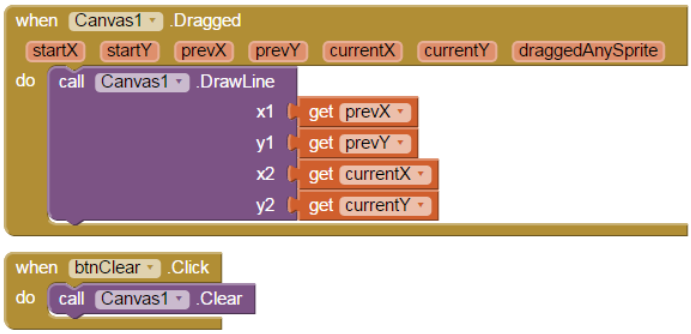
A képernyőn az ujjunk húzásával rajzoljunk görbéket!



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

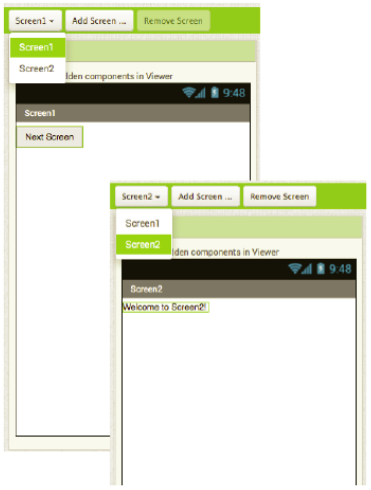
Blokkszerkesztő:



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

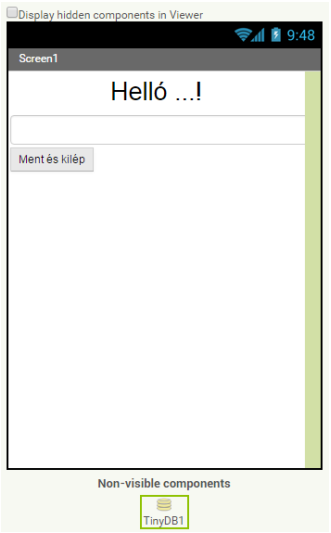
Blokkszerkesztő:

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

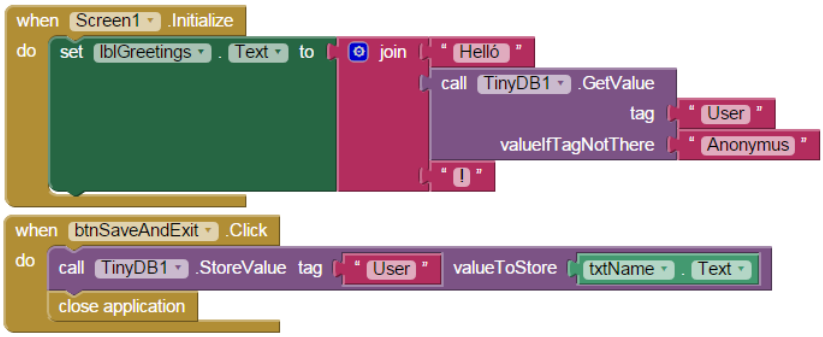


Készítsünk alkalmazást, amelyik emlékszik ránk és legközelebb üdvözöl minket!

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:



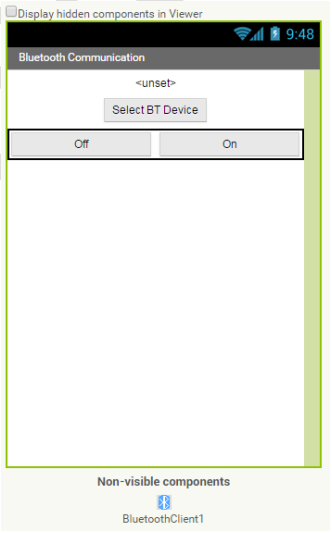
Mit jelent ez a kód?

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.

Bluetooth kommunikáció

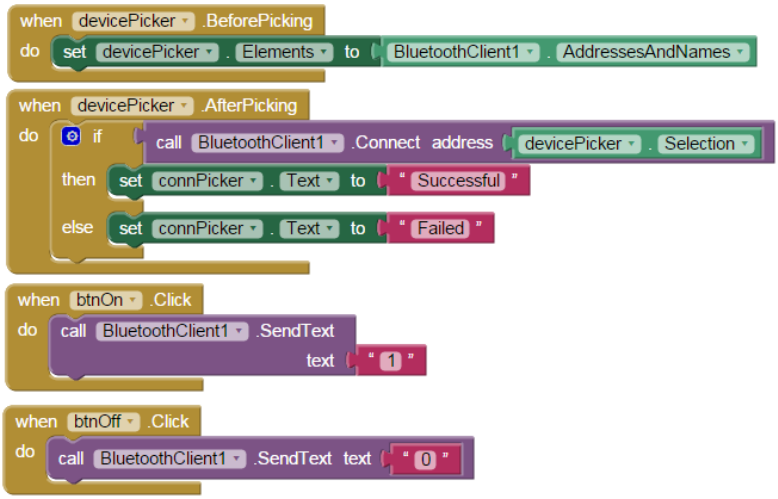
Csatlakozzunk bluetooth-on keresztül egy másik eszközhöz és küldjünk át neki 0-át, vagy 1-et



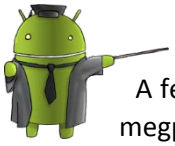
Designer elemek:

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.

V. Adatkezelés

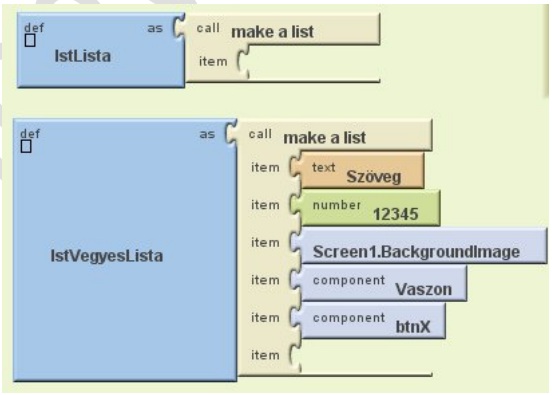
Az adatok kezelését az AppInventor jelenleg 5 alapvető nyelvi elemmel támogatja:

- **Lista**: a program futása alatt használhatjuk, tetszőleges nyelvi elem tárolására;
- **TinyDB:** helyileg, az eszközön tárolhatóak adatok „Kulcs – Érték” párokban;
- **TinyWebDB:** a Google felhőszolgáltatásában tárolhatóak adatok „Kulcs – Érték” párokban, de a blokk lehetővé teszi, hogy saját szervert üzemeltessünk a másik oldalon (JSON arrays);
- **Web:** a szerver oldalon tárolt adatok kérdezhetőek le több féle formátumban, nekünk a JSON kommunikációs forma a megfelelő;
- **Fusion Tables:** a felhasználói fiókhoz kapcsolódóan, a Google felhőszolgáltatásában tárolhatóak adatok táblákban.

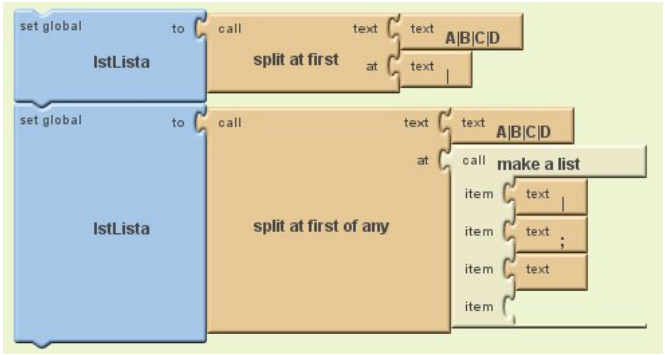
Listakezelés

Az AppInventor rendelkezik egy beépített, preferált adatszerkezettel. Ez a Lista, melyet a program futása alatt használhatunk. A listakezeléshez számos blokk tartozik önálló ***Lists*** csoporttal, ebből is látható, hogy kiemelt eleme az AppInventor blokknyelvnek.

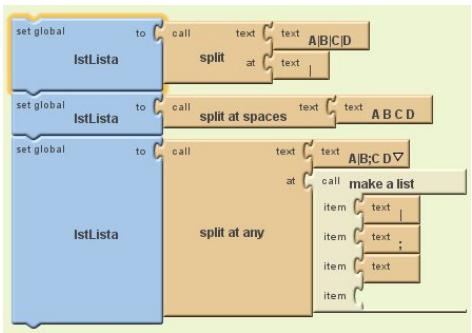
*Lista definiálás*ára szolgál a **<make a list>**[10] blokk, mellyel tervezésidőben eredményét változó definícióban felhasználva akár üresen, akár tetszőleges összetételű elemekkel feltöltve definiálhatunk listát.



E mellett *lista létrehozására* további megoldások a ***Text*** csoport blokkjai, melyek segítségével futásidőben szövegek darabolásával készíthetünk listákat.



- A **<splitatfirst>** blokkal egy szöveget darabolhatunk egy másik első előfordulása alapján 2 elemű listába,
- a **<splitatfirst of any>** blokkal a szöveget egy szöveg-lista bármelyik elemének első előfordulása alapján 2 elemű listába.
- a **<split>** blokkal egy szöveget darabolhatunk egy másik összes előfordulása alapján n elemű listába,

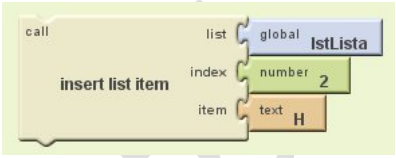


[10] A blokkok jelölésére a <> jeleket használom

- a **<splitatspaces>** blokk az előzőhöz hasonlóan darabol, de itt a szóközök mentén készül az n elemű lista,
- a **<splitatany>** blokk pedig a szöveget egy szöveg-lista bármelyik elemének összes előfordulása alapján n elemű listába darabolja.

Listabővítés módjai:

- Az **<add items to list>** blokkal tudunk futásidőben létező listához hozzáadni új elemet; a lista végén bővítünk, míg az **<insert list item>** blokk szolgál a lista megadott helyen (index) való bővítésére.



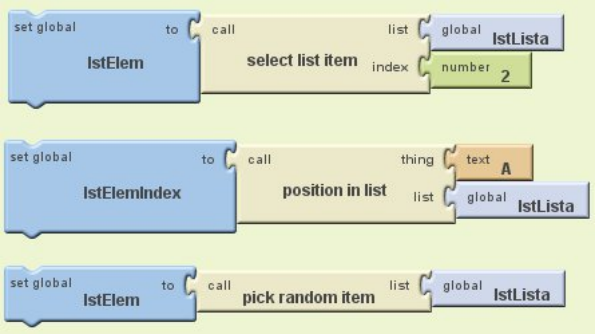
- Futásidőben az **<append to list>** blokkal van lehetőségünk lista bővítésére másik listával úgy, hogy a list2 elemet a list1 után fűzi list2 változatlansága mellett.

További műveletek a listával:

A **<copy list>** blokkal a lista *másolata* hozható létre egy másik változóba.

A lista *adott elemének értékét* megkaphatjuk

- a **<select list item>** blokk használatával, az elem indexe alapján,
- illetve a **<pick random item>** blokk segítségével véletlenszerű elem értékéhez jutunk.



A *lista elemének indexét* értékének ismeretében a **<position in list>** blokk segítségével kaphatjuk meg.

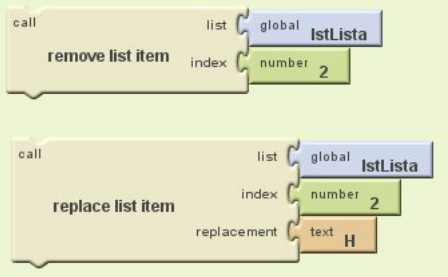
Felhasználási ötlet: Tegyük fel, hogy rendelkezünk két listával *lstMagyar* (magyar kifejezések listája), illetve *lstAngol* (angol kifejezések listája), úgy, hogy az *lstMagyar* lista i-edik elemének megfelelő angol kifejezés az *lstAngol* lista i-edik elemében van tárolva.

Ekkor egy **[Button]**[11] nyomógomb komponens szövege futásidőben állítható az *lstMagyar*, vagy az *lstAngol* lista megadott elemének értékével attól függően, hogy a felhasználó milyen nyelvet választott, vagy állított be programunkban.

[11] Vezérlők jelölésére a [ ] zárójeleket használom

Másik lehetőségünk, hogy egy **[ListPicker]** komponenst feltöltjük az *lstMagyar* lista elemeivel, majd a felhasználóval kiválasztatunk az elem szövege alapján egy elemet a **[ListPicker]** komponensben, a választott érték indexét meghatározzuk az *lstMagyar* listában, majd ezen index szerint vesszük az *lstAngol* lista elemét.

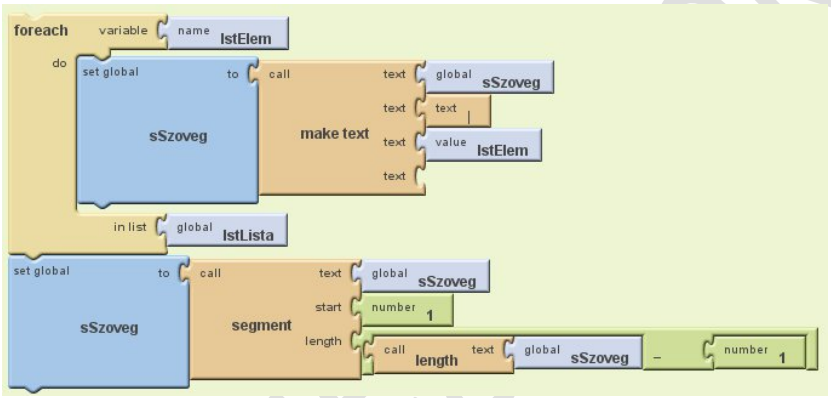
Listából való törlés/módosítás:



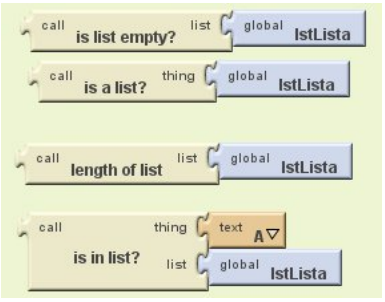
- Lista elemet tudunk eltávolítani a **<remove list item>** blokk használatával az indexe alapján,
- elem cseréjére a **<replace list item>** blokk szolgál.

Lista bejárása:

A listák összes elemén végig tudunk menni (bejárni) „megfogva” a listaértékeket a **<for each>** blokkal. A példában összefűzzük a lista elemeit szövegnek.



A listákkal kapcsolatban még a klasszikusnak számító ellenőrzések, műveletek végezhetőek, mint



- **<is list empty?>**, azaz üres-e a lista,
- **<is a list?>** lista-e,
- **<length of list>** a lista hossza,
- illetve még egy érdekes blokk, az **<is in list?>,** mellyel megtudjuk, hogy a keresett dolog (szöveg, kép, komponens, stb...) benne van-e a listában.

A lista egy igen hatékony adatszerkezete az MIT App Inventor 2 blokknyelvnek. A listakezelést kibővítették egy érdekes blokkcsoporttal, mellyel jelenleg egy listába elhelyezett azonos típusú komponens **[Button]**, **[Canvas]**, **[ImageSprite]**, **[Clock]**, **[Image]**, **[HorizontalArrangement]**, **[TableArrangement]**, **[VerticalArrangement]**, **[Screen]** tulajdonságai elegánsan kezelhetőek a **<for each>** blokk segítségével.

Példa: Tegyük fel, hogy szeretnénk egy többképernyős szoftvert készíteni, melyre jelenleg az alábbi szerkezetben van csak lehetőségünk, mivel az AppInventor jelenlegi állapotában nem kezel több képernyőt **[Screen]:**

[Screen$_1$][12]

[VA$_{start}$][13]

[HA$_{menü}$][14]

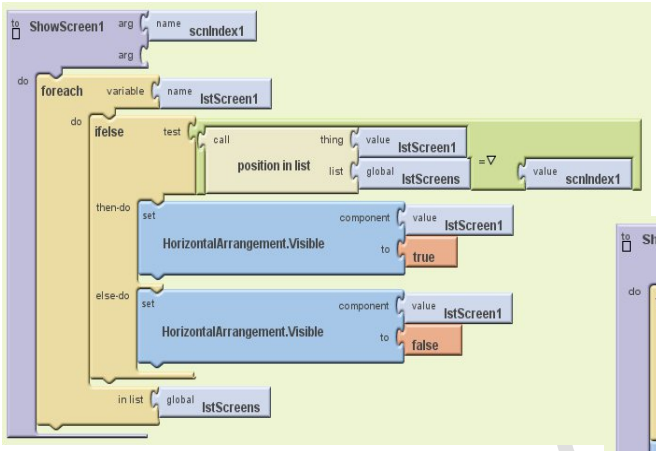
[HA$_{scn1}$]

[HA$_{scn2}$]

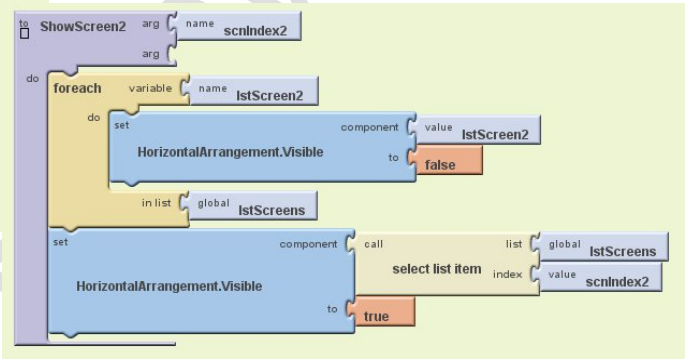
[HA$_{scn…}$]

[HA$_{scnn}$]

Egyszerre természetesen csupán egyetlen [HA$_{scni}$] „képernyőt” szeretnénk látni a „menüszalag” alatt, ezért az eddig leírtakat alkalmazva egy *lstSCN* listába rendezzük a „képernyőinket”, majd a <**foreach>** blokkal végigmenve az *lstSCN* lista elemein a **<HorizontalArrangement.Visible>** blokkal



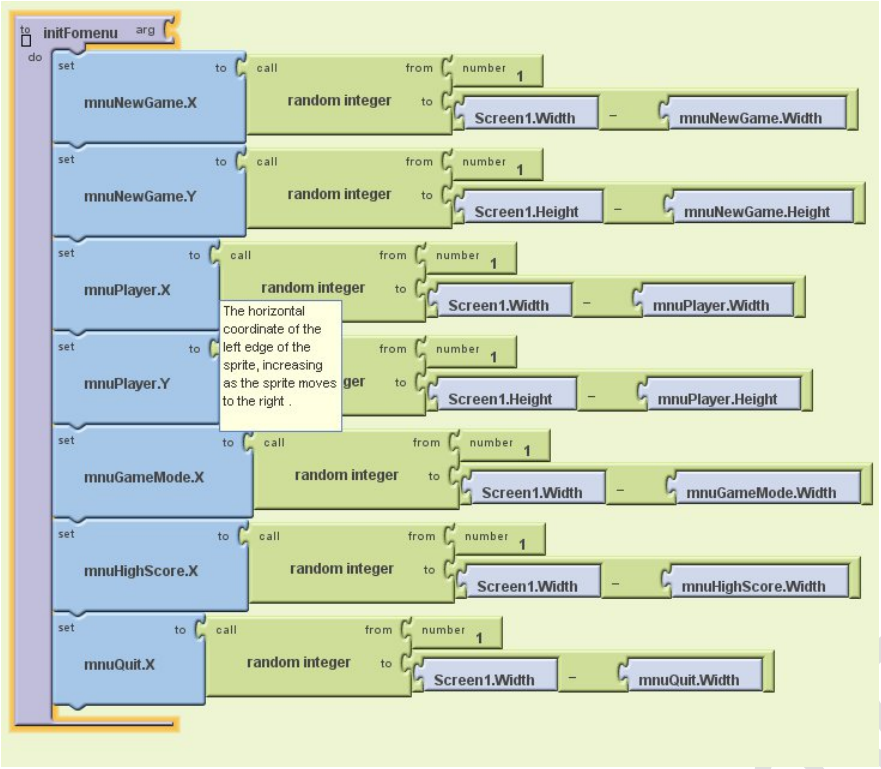
1. nem láthatóra állítjuk akkor, ha nem az i-edikről van szó, egyébként láthatóra, vagy
2. mindenképpen nem láthatóra állítjuk és a végén az i-edik láthatóságát igazra.



[12] Az AppInventorban korábban csak 1 képernyőt lehetett létrehozni

[13] VA: [VerticalArrangement] konténer, melyben egymás alatt vannak elrendezve automatikusan az elemek

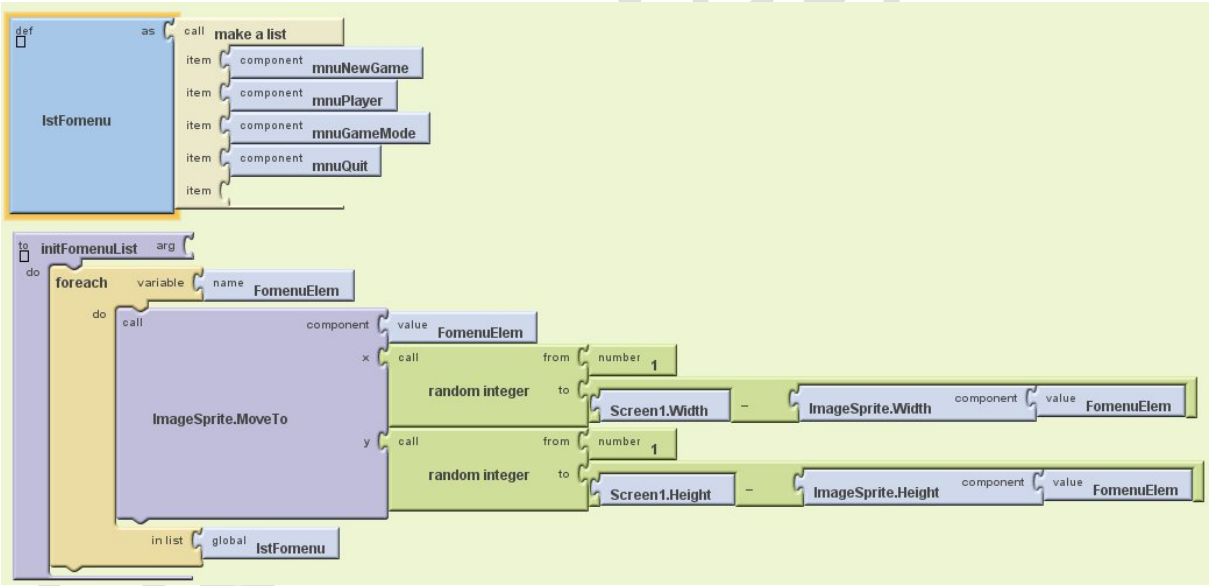
[14] HA: [HorizontalArrangement] konténer, melyben egymás mellett vannak elrendezve az elemek



Amennyiben a programunk egyik képernyőjén a menüpontokat szeretnénk elszórtan megjeleníteni, akkor az új lehetőségeknek megfelelően ezt már megtehetjük dinamikus módon lista használatával, kevesebb hibázási lehetőséggel.

A mellékelt kódrészleten látható, hogy minden egyes **[ImageSprite]** gombunknál sorban állítani kellett a pozícióját.

Az új lehetőséggel az alábbi módon néz ki a programunk ezen eljárása.



Még elegánsabbá és újrafelhasználhatóbbá tehetjük a kódot akkor, ha az *InitFomenuList* eljárásnak paraméterként adjuk át az *lstFomenu* listát, vagy bármely más listát, ami jelen esetben **[ImageSprite]** komponensekből állhat.

Összegzésképpen a lista, mint adatszerkezet az alábbi tulajdonságokkal rendelkezik:

- A program futása során dinamikus adattárolást tesz lehetővé. Tárolhatunk listában:
  - értékeket (szöveg, szám, szín, igaz/hamis),
  - listá(ka)t,

- különböző objektum-tulajdonságokat, mint pl. az **[Image]** komponens Picture tulajdonsága, ami felhasználható animációs képkockák listában való tárolására,
- valamint különböző komponenseket. (A blokkszerkesztő Any component csoportjába tartozó, dinamikusan megjelenő **[Any Button]**, **[Any Canvas]**, **[Any HorizontalArangement]**, stb komponensek esetében van lehetőség az ilyen listák felhasználására.)

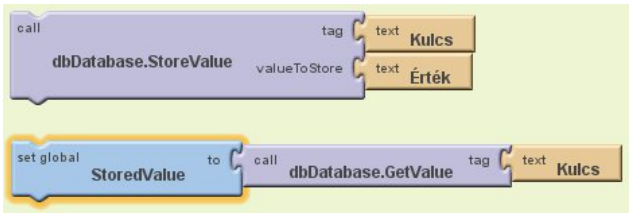
- A lista elemei indexük alapján közvetlen elérésűek **<select list item>**,
- elemei az elsőtől az utolsóig bejárhatóak **<for each>**,
- bővíthető egyaránt a végén **<add items to list>** vagy tetszőleges helyen **<insert list item>**,
- elemei egyszerűen törölhetőek az indexük ismeretében **<remove list item>**,
- elemei cserélhetőek **<replace list item>**,
- a különböző listák összefűzhetőek **<append to list>**,
- másolhatóak **<append to list>**,
- ellenőrizhető, hogy listáról van-e szó**<is a list?>**,
- üres-e a lista **<is list empty?>**,
- a keresett dolog (szöveg, kép, komponens, stb...) benne van-e a listában **<is in list?>**,
- illetve lekérdezhető a lista hossza **<length of list>**.

TinyDB

Az AppInventor blokknyelv a perzisztens, lokális adattároláshoz rendelkezik a **[TinyDB]** komponenssel, amelyet az alap komponenscsoportban találunk.

A **[TinyDB]** nem látható komponens, ennek megfelelően nincsenek sem írható, sem olvasható tulajdonságai.

Pusztán arra szolgál, hogy az okos eszközünkön a programunkhoz csatoltan tárolni tudjunk kulcs–érték párokat. A kulcs–érték pár tárolására a komponens ***StoreValue*** eljárása szolgál, míg visszaolvasásra a ***GetValue*** függvény.



Mint az látható, pusztán tárolni és visszanyerni tudjuk a kulcsokhoz tartozó értékeket, így a megszokott adattárolási funkcióknak csak korlátozottan tesz eleget:

- *Létrehozás*: egyszerűen a ***StoreValue*** eljárás használatakor létrejön a programhoz kapcsolódó tárterületen a kulcs és az érték is.
- *Módosítás*: a ***StoreValue*** eljárás során a megadott kulcs szerint az értékkel felülírjuk az előzőt.
- *Törlés:* kulcsot nem, értéket lehet. Az értéket a ***StoreValue*** eljárással üres szövegre, vagy valamely kinevezett alapértékre állíthatjuk. Ebből következik, hogy egy kulcsnak három állapotát tudjuk kezelni (létező, nem létező, törölt).
- *Hozzáférés:* Közvetlenül a kulcs ismeretében a ***GetValue*** függvénnyel megkapjuk az értéket.
- *Bejárás*: nem értelmezhető, nincs nyelvi elem a programhoz tartozó kulcsok kinyerésére a kulcsok ismerete nélkül.
- *Rendezés*: nem értelmezhető, mivel nincs a halmazon kívül a kulcs-érték pároknak az AppInventor nyelvi elemivel hozzáférhető struktúrája.

A **[TinyDB]** komponens kulcs-érték párjainak elsődleges felhasználási területeként a programbeállítások futtatások közötti tárolására gondolhatunk. Ezekben az esetekben közös, hogy ismerjük a használt kulcsokat.

- Első példában a felhasználónak megengedjük, hogy beállítsa, hogy fekvő vagy álló helyzetben szeretné használni a programunkat. A kulcs legyen ’Orientation’, a lehetséges értékek pedig {’’,’Álló’,’Fekvő’,’Szenzor’}, a program indulásakor beolvassuk az ’Orientation’ kulcs értékét és ennek megfelelően állítjuk álló, fekvő, vagy automatikus pozícióba a kpernyőt.

- Felhasználónév és jelszó kell a programunk funkcióinak eléréséhez és megengedjük a felhasználónak, hogy az első futáskor megadja ezeket. A kulcsok legyenek ’usr’ és ’pwd’ a program indításakor, ha üres az ’usr’ és ’pwd’ kulcs értéke, akkor a felhasználó beállítja a kulcs-érték párokat, egyébként a beírt értékeket ellenőrizzük csupán.

- Használhatjuk a különböző vizuális komponensek **[ImageSprite]**, **[Button]**, stb. tulajdonságainak tárolására is oly módon, hogy a kulcs legyen a komponens neve, míg az érték szöveget a komponens tulajdonságaiból megadott sorrendben képezzük valamely elválasztó karakter kiválasztásával. A komponens tulajdonságokat lista blokkba olvassuk vissza valamelyik **<split …>** blokk használatával.

A kulcsok képzése során érdemes megfontolnunk, hogy bármi, amit tárolni akarunk ugyanabba a „halmazba” kerül és bonyolult programok esetében könnyen előfordulhat, hogy ugyanazzal a kulccsal tárolnánk különböző dolgok értékét. Ennek elkerülése érdekében a kulcsok képzése során a különböző típusok számára különböző kulcs-prefixeket alkalmazhatunk. Ilyen kulcs-prefix változók esetében lehetne a ’var_’, programbeállítások esetében ’set_’, felhasználónév és jelszó esetén ’sec_’, stb. Ezen technika alkalmazásával jelentősen csökkenthető a kulcs-érték párok véletlen felülírásának veszélye.

A kulcs-érték párok önmagukban csak az előzőekben bemutatott szűk felhasználási lehetőséggel rendelkeznek, ezért érdemes különböző adatszerkezeteket létrehozni a segítségükkel. Ez minden esetben a kulcs és/vagy érték rész meghatározott szerkezetű felépítésével elérhető.

1. Példa: A programban elért **n** legjobb eredményt szeretnénk tárolni **[TinyDB]** komponenssel. Ennek érdekében létrehozzuk az alábbi kulcs-érték párokat (a kulcs-prefix ’***eredm_***’, az érték elemeit a ’**|**’ jel felhasználásával fűzöm össze, az alapérték a ’***null***’ szöveg, **n** pozitív természetes szám):

   - <eredm_játékos><j_1|j_2|…|j_n>
   - <eredm_eredmény><e_1|e_2|...|e_n>

   Az értékeket ’játékos’ és ’eredmény’> n-esekben tároljuk, a használathoz futásidőben egy-egy listába olvassuk a ***Text*** csoport **<split>** blokkja segítségével, amellyel egy szöveget darabolhatunk egy másik összes előfordulása alapján n elemű listába. Ez az adatszerkezet a továbbiakban egy-egy n elemű csökkenő módon rendezett, közvetlen elérésű lista. Ekkor a **[TinyDB]** komponenssel statikus állapotokat tárolunk csak, minden adatszerkezeti műveletet futásidőben végzünk a két listán, majd a műveletek végén letároljuk a két listát a megfelelő kulcsokkal.

   A módszer hátránya, hogy **n** nem lehet tetszőlegesen nagy (5-10 eredmény tárolásához használható biztonsággal), valamint kötött a kulcsok elnevezése, illetve a kulcs csak a hozzá tartozó értékek elérését szolgálja.

2. Példa: A programban elért **n** legjobb eredményt szeretnénk tárolni **[TinyDB]** komponenssel. Ennek érdekében egyszer létrehozzuk az alábbi kulcs-érték párokat (a kulcs-prefix ’***tbl_***’, a kulcs és az érték elemeit a ’**|**’ jel felhasználásával fűzöm össze, az alapérték a ’***null***’ szöveg, **n** pozitív természetes szám):

   - <tbl_eredmény|1><null>
   - <tbl_eredmény|2><null>
   - <tbl_eredmény|...><null>
   - <tbl_eredmény|n><null>

   Az értékeket <játékos|elért eredmény> párokban tartjuk nyilván, ez az adatszerkezet egy n elemű csökkenő módon rendezett, közvetlen elérésű lista.

   Műveletek az adatszerkezettel:

   - *Bejárás*: az n kulcs, illetve ennek tetszőlegesen kiválasztott k eleme tetszőleges irányban egymás után megfogható, értékük kinyerhető.

- *Keresés*: a kulcs és az érték ’elért eredmény’ része rendezett, ezek esetében alkalmazhatóak hatékonyabb kereső algoritmusok, az érték játékos része rendezetlen, erre csak a teljes keresés alkalmazható.
- *Rendezés*: kulcsra nem értelmezett, az értékek ’játékos’ és ’elért eredmény’ alapján is rendezhető (a példában nincs jelentősége, a célnak megfelelő rendezettséggel kerülnek be az adatok).
- *Törlés*: a törlés során a törlendő érték kulcsának számrészénél egyel nagyobb kulcs értékével felülírom, míg el nem jutok az n számrészű kulcsig, melynek értékét null-ra állítom.
- *Beszúrás*: bonyolult, ha megtaláltuk hova kell beszúrni (k-adik kulcs), akkor az n-ediktől a k+1-dik kulcsig fölülírjuk a kulcsok értékeit a megelőző értékével, majd a k-adik kulcs értékének megadjuk a beszúrandó értéket.

A módszer hátránya, hogy a kulcsok továbbra is kötöttek, de **n** tetszőlegesen nagy lehet és a tárolt adatok részleteikben is használhatók, nem csak egyszerű tárolásra használjuk a **[TinyDB]** komponenst.

3. Példa: A programban elért összes eredményt szeretnénk tárolni **[TinyDB]** komponenssel. Ennek érdekében létrehozzuk az <**Eredmények**><null|null|null>kulcs-érték párt, mint fix belépő pontot , ahol a kulcs-prefix '***eredm_***', az érték első két eleme hivatkozást tartalmaz a megelőző és a rákövetkező kulcsra, a harmadik pedig a listaelemszáma. Az értékeket a '**|**' jel felhasználásával fűzöm össze, az alapérték a '***null***' szöveg, **n**ismeretlen pozitív természetes szám).

   Kétirányú láncolt eredmény-lista:

   $\boldsymbol{Val}^{15}(\mathbf{Eredmények}^{16}) = \{\boldsymbol{Key}^{17}(E_1)|\boldsymbol{Key}(E_n)|n\}$

   Amennyiben még nincs egyetlen eredményünk sem

   $$\boldsymbol{Val}(\mathbf{Eredmények}) = \{\mathbf{null}|\mathbf{null}|\mathbf{0}\}$$

   Egyetlen eredménnyel rendelkezünk:

   $\boldsymbol{Val}(\mathbf{Eredmények}) = \{\boldsymbol{Key}(E_1)|\boldsymbol{Key}(E_1)|1\}$ és
   $\boldsymbol{Val}(E_1) = \{\boldsymbol{Key}(E_1)|\boldsymbol{Key}(E_1)|Játékosnév|Pontszám\}$

   Több eredmény esetén fennáll:

   $\boldsymbol{Val}(\mathbf{Eredmények}) = \{\boldsymbol{Key}(E_1)|\boldsymbol{Key}(E_n)|n\}$
   $\boldsymbol{Val}(E_i^{18}) = \{\boldsymbol{Key}(E_{i-1})|\boldsymbol{Key}(E_{i+1})|V_{játékosnév}|V_{pontszám}\}$

   Töröltnek tekinthetjük az eredményt, ha teljesülnek az alábbi feltételek:

   $\boldsymbol{Key}(E_i) \cap \boldsymbol{List}^{19}\{\mathbf{Eredmények}\} = \emptyset$
   $\boldsymbol{Val}(E_i) = \{\mathbf{null}|\mathbf{null}|\mathbf{null}|\mathbf{null}\}$

   Az értékeket <játékosnév|pontszám> párokban tartjuk nyilván, ez az adatszerkezet egy n elemű csökkenő módon rendezett, kétirányú láncolt lista. Kiegészítésképpen az Eredményekhez tartozó kulcsokat megtisztítva a kulcs-prefixtől beolvassuk egy *lstEredmények* listába, mellyel közvetlen kulcsalapú hozzáférést biztosítunk a program futásideje alatt az Eredményekhez.

[15] A ***Val*** függvénnyel jelölöm a zárójelben megadott nevesített kulcshoz tartozó értéket.

[16] Az Eredmények kétirányú láncolt lista nevesített kulcsa, fix belépőpont.

[17] A ***Key***függvénnyel jelölöm a zárójelben megadott kulcs-érték párból a kulcsot.

[18] Az E_i jelölést használom az eredménylistához tartozó kulcs-érték párok jelölésére, ahol 1 <= i <= n

[19] A ***List*** függvénnyel jelölöm a zárójelben megadott nevesített kulcs által meghatározott kétirányú láncolt listát.

Műveletek az adatszerkezettel:

- *Bejárás*: az n kulcs, illetve ennek tetszőlegesen kiválasztott k eleme tetszőleges irányban egymás után megfogható, értékük kinyerhető.
- *Keresés*: ebben az implementációban nincs értelme.
- *Rendezés*: kulcsra nem értelmezett, az értékek alapvetően'Pontszám' alapján rendezetten kerülnek be).
- *Beszúrás*: nagyon egyszerű, megkeressük az elem helyét és beállítjuk a környezete hivatkozásait a beszúrandó elemre, illetve a beszúrandó elem hivatkozásai a környezetére. Minden esetben $\boldsymbol{Key}(E_{\mathrm{B}}) = 'eredm\_' + (\boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{3}} + \mathbf{1})$

  - Beszúrás üres eredmény-listába:

    $\boldsymbol{Val}(E_{\mathrm{B}}) = \{\boldsymbol{Key}(E_{\mathrm{B}})|\boldsymbol{Key}(E_{\mathrm{B}})|\mathrm{V}_{játékosnév}|\mathrm{V}_{pontszám}\}$
    $\boldsymbol{Val}(\mathbf{Eredmények}) = \{\boldsymbol{Key}(E_{\mathrm{B}})|\boldsymbol{Key}(E_{\mathrm{B}})|1\}$

  - Beszúrás az eredmény-lista elején:

    $\boldsymbol{Val}(E_{\mathrm{B}}) = \{\boldsymbol{Key}(E_{\mathrm{B}})|\boldsymbol{Key}(E_1)|\mathrm{V}_{játékosnév}|\mathrm{V}_{pontszám}\}$
    $\boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{1}} = \boldsymbol{Key}(T_{\mathrm{B}})$
    $és\ \ \boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{3}} = \boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{3}} + \mathbf{1},$
    $ezáltal\ \boldsymbol{Val}(\mathbf{Eredmények}) = \{\boldsymbol{Key}(T_{\mathrm{n+1}})|\boldsymbol{Key}(T_{\mathrm{n}})|n+1\}$
    $\boldsymbol{Val}(E_1)_{\mathbf{1}} = \boldsymbol{Key}(E_{\mathrm{B}})$

  - Beszúrás az eredmény-listavégén:

    $\boldsymbol{Val}(E_{\mathrm{B}}) = \{\boldsymbol{Key}(E_n)|\boldsymbol{Key}(E_B)|\mathrm{V}_{játékosnév}|\mathrm{V}_{pontszám}\}$
    $\boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{2}} = \boldsymbol{Key}(T_{\mathrm{B}})$
    $és\ \ \boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{3}} = \boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{3}} + \mathbf{1},$
    $ezáltal\ \boldsymbol{Val}(\mathbf{Eredmények}) = \{\boldsymbol{Key}(E_1)|\boldsymbol{Key}(E_{\mathrm{n+1}})|n+1\}$
    $\boldsymbol{Val}(\mathrm{E_n})_2 = \boldsymbol{Key}(\mathrm{E_B})$

  - Beszúrás az eredmény-listaközepén, az i-edik eredmény után:

    $\boldsymbol{Val}(E_{\mathrm{B}}) = \{\boldsymbol{Key}(E_{\mathrm{i}})|\boldsymbol{Key}(E_{\mathrm{i+1}})|\mathrm{V}_{játékosnév}|\mathrm{V}_{pontszám}\}$
    $\boldsymbol{Val}(E_{\mathrm{i}})_{\mathbf{2}} = \boldsymbol{Key}(E_{\mathrm{B}})$
    $\boldsymbol{Val}(E_{\mathrm{i+1}})_{\mathbf{1}} = \boldsymbol{Key}(E_{\mathrm{B}})$
    $\boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{3}} = \boldsymbol{Val}(\mathbf{Eredmények})_{\mathbf{3}} + \mathbf{1}$

Törlés: a törlés egyszerű, melynek során a megelőző kulcs rákövetkezőjét a törlendő kulcs rákövetkezőjére állítjuk, a rákövetkező kulcs megelőzőjét pedig a törlendő kulcs megelőzőjére cseréljük, majd a törlendő kulcs értékét alapértékre állítjuk. Az eredmények számát nem változtatjuk, mivel továbbra is **n** eredményünk marad, csupán egy értékét „töröltre" állítjuk.

- Az egyetlen eredmény törlése:

  $\boldsymbol{Val}(E_1) = \{\mathbf{null}|\mathbf{null}|\mathbf{null}|\mathbf{null}\}$
  $\boldsymbol{Val}(\mathbf{Eredmények}) = \{\mathbf{null}|\mathbf{null}|\mathbf{0}\}$

- Az első Eredménytörlése:

  $\boldsymbol{Val}(E_1) = \{\mathbf{null}|\mathbf{null}|\mathbf{null}|\mathbf{null}\}$
  $\boldsymbol{Val}(E_2)_\mathbf{1} = \boldsymbol{Key}(E_2)$
  $\boldsymbol{Val}(\mathbf{Eredmények})_\mathbf{1} = \boldsymbol{Key}(T_2)$

- Az utolsó Eredménytörlése:

  $\boldsymbol{Val}(E_n) = \{\mathbf{null}|\mathbf{null}|\mathbf{null}|\mathbf{null}\}$
  $\boldsymbol{Val}(E_{\mathrm{n}-1})_\mathbf{2} = \boldsymbol{Key}(E_{\mathrm{n}-1})$
  $\boldsymbol{Val}(\mathbf{Eredmények})_\mathbf{2} = \boldsymbol{Key}(E_{\mathrm{n}-1}).$

- Az i-edik Eredménytörlése:

  $\boldsymbol{Val}(E_{\mathrm{i}-1})_\mathbf{2} = \mathrm{V}\boldsymbol{al}(E_i)_2$
  $\boldsymbol{Val}(E_{\mathrm{i}+1})_\mathbf{1} = \mathrm{V}\boldsymbol{al}(E_i)_1$
  $\boldsymbol{Val}(E_i) = \{\mathbf{null}|\mathbf{null}|\mathbf{null}|\mathbf{null}\}$

Ennek a módszernek a jellemzői:

- Tetszőleges számú kulcs-érték párunk lehet, egyedül az okos eszközünk tárolókapacitása szab határt.
- A Kulcsok tartalma kötött, explicit módon a program készítése során határozzuk meg a kulcsképzés módszerét.
- A kulcs-prefixek segítségével könnyen elkerülhetőek a kulcsütközések.
- Dinamikus adatszerkezetben tároljuk a kulcs-érték párokat, a kulcs-értékek adathalmazában akár egyesével is kezelhetünk ilyen párokat.

A kétirányú láncolt listára épülően egyéb adatszerkezeteket is képesek vagyunk megvalósítani:

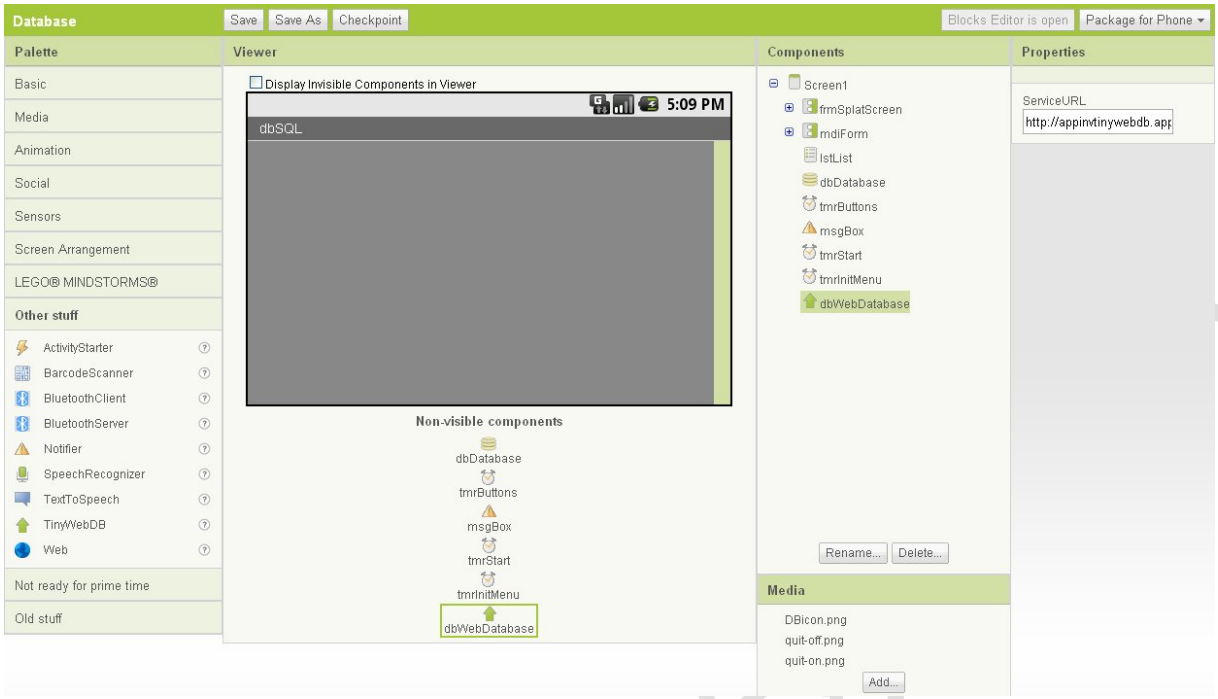
- **Sor** esetében a lista elején *bővítünk*, a végén *törlün*k és *kiolvasunk értéket*. *Keresés, rendezés* nincs értelmezve.
- **Verem** esetében a verem tetején levő adatot manipulálhatjuk; azaz a lista elejéhez férünk csak hozzá, az elején *bővítünk és törlünk*. *Keresés, rendezés* nincs értelmezve.

A bemutatott adattár megoldással már egyszerűbb feladatok megoldhatóak, ilyenek a teljesség igénye nélkül:

- Kvízjátékok;
- távoktatási vizsgalapok;
- kidolgozható a fenti adattárolási módszeren alapuló elektronikus tananyag (amennyire sikerült utána járnom az Android operációs rendszer alatt közvetlenül nem hozzáférhetőek a programokhoz társított adatok, sok mindent elrejt a rendszer a felhasználó elől);
- jegyzetek tárolása;
- TinyWebDB komponenssel kiegészítve szinkronizáltan csoportmunka alkalmazások létrehozása.

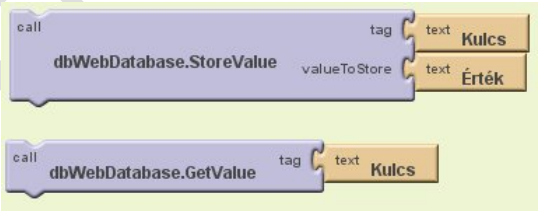
TinyWebDB

Az AppInventor blokknyelv a perzisztens, szerver oldali adattároláshoz rendelkezik a **[TinyWebDB]** komponenssel, amelyet az „egyéb cuccok”[20] komponenscsoportban találunk.

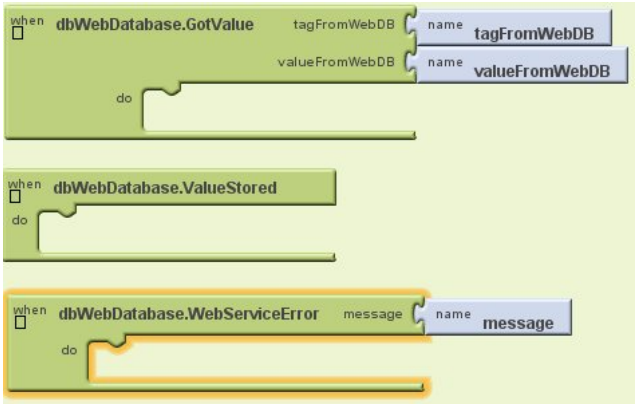


A **[TinyWebDB]** nem látható komponens, de a **[TinyDB]** komponenssel ellentétben egy Service URL tulajdonsággal rendelkezik, melyet futásidőben is állíthatunk.

Pusztán arra szolgál, hogy a programunkkal egy web szerveren tárolni tudjunk kulcs–érték párokat. A kulcs–érték pár tárolására a komponens ***StoreValue***, míg vissza-olvasásra a ***GetValue*** eljárás szolgál.



Ezen túl kapunk még három eseménykezelő blokkot:



- a **WebServiceError** esemény-blokkban tudjuk a hálózati hibát elcsípni,
- a **ValueStored** esemény-blokk a ***StoreValue*** eljárás sikeres végrehajtása után aktivizálódik,
- míg a **GotValue** eseménnyel a ***GetValue*** eljárás sikeres végrehajtása után jutunk a kulcs-érték párunk értékeihez.

Alapvetően tárolni és visszanyerni tudunk kulcsokhoz tartozó értékeket, így önmagában a megszokott adattárolási funkcióknak csak korlátozottan tesz eleget:

[20] Eredetileg is Other stuff a csoport neve, nem saját szleng

- *Létrehozás:* egyszerűen a ***StoreValue*** eljárás használatakor létrejön a szerveren a kulcs és az érték is.
- *Módosítás:* a ***StoreValue*** eljárás során a megadott kulcs szerint az értékkel felülírjuk az előzőt.
- *Törlés*: kulcsot nem, értéket lehet. Az értéket a ***StoreValue*** eljárással üres szövegre, vagy valamely kinevezett alapértékre állíthatjuk. Ebből következik, hogy egy kulcsnak három állapotát tudjuk kezelni (létező, nem létező, törölt).
- *Hozzáférés*: Közvetlenül a kulcs ismeretében a ***GetValue*** eljárás meghívásával a **GotValue** eseményben megkapjuk az értéket.
- *Bejárás*: nem értelmezhető, nincs nyelvi elem a programhoz tartozó kulcsok kinyerésére a kulcsok ismerete nélkül.
- *Rendezés*: nem értelmezhető, mivel nincs a halmazon kívül a kulcs-érték pároknak az AppInventor nyelvi elemivel hozzáférhető struktúrája.

A **[TinyWebDB]** komponens kulcs-érték párjainak elsődleges felhasználási területeként a programbeállítások futtatások közötti szerver oldali tárolására gondolhatunk. Ezekben az esetekben közös, hogy ismerjük a használt kulcsokat. Alapvetően a **[TinyWebDB]** komponensre igaz minden, amit a **[TinyDB]** komponens esetében írtunk, csak felhasználási területét tekintve más: a program több felhasználóját globálisan érintő adatok szerveren való tárolására és olvasására vonatkoztatjuk mindazokat.

A két komponens hasonló felépítéséből adódóan felmerül a lokális adatok szinkronizálása aktív internet kapcsolat esetén a szerver oldalon tárolt adatokkal. Például:

- Többnyelvű program esetében a különböző nyelvi készleteket szerver oldalon tároljuk a megszokott kulcs-érték párokban „időbélyegezve”. A felhasználónak a programhoz kapcsolódóan a lokális adattárába csak az általa választott nyelv kulcs-érték párjait juttatjuk el.
- Szerver oldalon tárolunk különböző teszteket eLearning „keretrendszerben” és a felhasználó eszközén csak az általa szükségesnek vélt teszteket töltjük le kérésre.
- A szerverről letöltött teszt megoldását visszatöltjük az eszközünkről, hogy a Tanár ellenőrizhesse.
- Csoportkommunikáció céljából a tagok által írt üzeneteket témánként a szerver oldalon tároljuk, de a felhasználó aktív netkapcsolat esetén szinkronizálhatja a lokális adattárába.
- A játékprogramunkhoz szerver oldali eredménytáblát tartunk fent, amelyet szinkronizálhatunk a lokális adatokkal.

A **[TinyWebDB]** komponens használatához a Google felhőjében a program számára saját webes adattárat is létrehozhatunk[21].

[21] Bővebben a http://ai2.appinventor.mit.edu/reference/other/tinywebdb.html linken van információ

Alapértelmezés szerint a TinyWebDB komponenssel az adatokat tárolni lehet a Google által nyújtott megosztott teszt szolgáltatás[22] keretében. Ez hasznos a program teszteléséhez, de 1000 bejegyzéses a tárolási kapacitása.

Teljesen saját felügyelet alatt álló adattárait is készíthetünk, illetve ahhoz hozzáférhetünk az AppInventor **[TinyWebDB]** komponensével, de ehhez fel kell készítenünk a weboldalunkat a *storevalue* és *getvalue* paraméterek kezelésére.

A **[TinyWebDB]** komponens *GetValue* eljárása elküldi paraméterben a kulcsot, amelyet a $tag =trim($_POST["tag"]); PHP-kóddal csípünk el, majd jöhet a kulcsra vonatkozó kezelő eljárás és a válasz JSON tömb formátumban, melyet a **[TinyWebDB] GotValue** eseményében fel tudunk dolgozni.

```
<?php
$postUrl=$_SERVER["REQUEST_URI"];
require_once('inc/function.php');
foreach (glob('classes/class_*.php') as$file) {
require_once($file);
}

if(strpos($postUrl,'getvalue')) {
    // TinyWebDB.GetValue
$tag =trim($_POST["tag"]);
   //$tag = "KeyHead";
if ($tag<>""){
$MySQL = newNewMySQL(MYSQL_HOST, 3306, MYSQL_USER, MYSQL_PWD, MYSQL_DB, true);
$theData = $MySQL->tinydbGetValue("SELECT Value FROM ".TBL_PF."tinywebdb WHERE TAG='".$tag."'");
echo '["VALUE", "'.$tag.'", "'.$theData.'"]';
unset($MySQL);
}else {
echo "";
    }
}else {
  // TinyWebDB.StoreValue
$tagToStore =trim($_POST["tag"]);
$valueToStore =trim($_POST["value"]);

if ($tagToStore<>""){

$MySQL = newNewMySQL(MYSQL_HOST, 3306, MYSQL_USER, MYSQL_PWD, MYSQL_DB, true);
$theData = $MySQL->tinydbStoreValue("INSERT INTO `".TBL_PF."tinywebdb` (`TAG` ,`Value`) VALUES
('".$tagToStore."', '".$valueToStore."')");
echo '["STORED", "'.$tagToStore.'", "'.$valueToStore.'"]';
unset($MySQL);

}else {
echo "";}

}
```

[22] http://appinvtinywebdb.appspot.com/

```
?>
```

Ekkor konkrét MySQL alapú adatbázist dolgoztathatunk a háttérben és oda kell figyelnünk a *storevalue* paraméter esetén, hogy meglévő kulcs esetén az **UPDATE**, míg új kulcs esetén **INSERT INTO** utasítással kell az adatbázissal kommunikálni; a fenti példában csak egyszerű **INSERT INTO** van.

Összetett kulcsokkal meg tudjuk adni a szerverünknek, hogy milyen adatokra van szükségünk. Ilyenkor nem feltétlenül csak a kulcshoz tartozó szerver oldali kulcs-érték párt kaphatjuk vissza, hanem megoldható a kérés –szerverválasz (JSON tömb) klasszikus kommunikáció is.

```
<?php
$postUrl=$_SERVER["REQUEST_URI"];
require_once('inc/function.php');
foreach (glob('classes/class_*.php') as$file) {
require_once($file);
}
if(strpos($postUrl,'getvalue')){
$tag =trim($_POST["tag"]);
if ($tag == 'viktor'){
$MySQL = newNewMySQL(MYSQL_HOST, 3306, MYSQL_USER, MYSQL_PWD, MYSQL_DB, true);
$theData = $MySQL->mQuery("SELECT * FROM ".TBL_PF."employees WHERE active='yes' ORDER
byemployeeNumber ASC LIMIT 10", array(1, 2, 3, 4, 5));
$resultData = array("VALUE",$tag,$theData);
$resultDataJSON = json_encode($resultData);
echo$resultDataJSON;
unset($MySQL);
echo$tag;
  }
}
?>
```

A fenti példában a ’viktor’ kulcsot elküldve a honlapnak, válaszul megkapjuk az employees tábla első 10 aktív rekordjának adatai közül az 1..5 mezők tartalmát JSON tömb formátumban.

Például egy ’pwd|név|jelszó|r’ kulcsot elküldve az adatbázisunkban tárolt users táblában ellenőrizni tudjuk, hogy a felhasználó megfelelő jelszót állított-e be, illetve válaszként visszaküldhetjük a felhasználó jogait [”VALUE”,”pwd|név|jelszó|r”,”r|w|a|c”] JSON tömb formátumú választ küldhetünk.

További adatkezelést támogató vezérlők

- A **[Web]** komponens adatkezelésben való használata hasonló a **[TinyWebDB]** komponenséhez, azzal a különbséggel, hogy még általánosabb felhasználást tesz lehetővé az adatokhoz való hozzáférés terén. Így többek között xml formátumú adatok fogadására is alkalmas, illetve jobban támogatja az utf-8 karakterkódolást is.
- A **[Fusion Tables]** esetében teljesen önálló eljáráscsoportok vannak.

VI. Záró gondolatok

Sebesta könyve nyilvánvalóan olyan szemszögből vizsgálja a programozási nyelvek evolúcióját, mely gondolkodásmód a programozás „hőskorának” sajátja. Adott volt egy programozó(csoport), aki(k) megalkot(nak) egy új nyelvet, és minden fejlesztés azon múlik, ők továbbgondolják, továbbfejlesztik-e majd azt. Azonban jelen társadalmunkban, az információs társadalomban – annak is web2.0-ás verziójában – a közösség visszahat az informatika fejlődésére. Az informatika egyes elemei, legyen az egy új hardvereszköz, vagy programozási nyelv egyfajta olyan „árucikk”, melynek fenntarthatósága, fejlődése a „fogyasztók” érdeklődésétől függ. Esetünkben a fogyasztók természetesen a programozói közösség, akik használni fogják az MIT App Inventor 2-t. Ez a közösség nyílt, az alkotók újabb ötleteket várnak a „rajongói tábortól”, hagyják, hogy ők szabják meg a fejlesztés irányát. Erre be is mutattam jó néhány példát előzőleg. Egyszóval egy programozási nyelv fenntarthatóságára sokszor nemcsak a programozási nyelvben lévő elemek hatnak, hanem külső tényezők is. Nemsokára tehát újra kell gondolni azt, hogy ezeket a külső tényezőket hogyan lehetne az elemzés szempontjai közé felvenni, egyáltalában hogyan mérhető ezen külső szempontok hatása adott nyelv evolúciójára.

Mindenesetre ennek a nyelvnek a fejlődése nagyon gyors, azt hiszem, a programozói közösség sokkal kedvezőbben fogadta, mint azt akár az alkotói remélték volna, és ha mindkét félben megmarad ez a lelkesedés, még sok lehetőség rejlik benne elsősorban az informatika oktatáson belül, az okos eszközök programozásának oktatása területén, mely jelenleg nem szerepel közoktatásunkban sem az informatika tananyagban, de lassan kikerülhetetlen lesz.

VII. Idézett forrásmunkák

[1] Bhagi, A. (2016). *AI2 FlashCards.* Forrás: MIT App Inventor 2: http://appinventor.mit.edu/explore/sites/all/files/Resources/AppInventorFlashCards.pdf

[2] Eclipse Foundation. (2011). *About the Eclipse Foundation*. Letöltés dátuma: 2010, forrás: Eclipse: http://www.eclipse.org/org/

[3] Gábor, A. (2008. December 12). *Java Forum*. Letöltés dátuma: 2010, forrás: Java Forum: http://wiki.javaforum.hu/pages/viewpage.action?pageId=19365990

[4] Google. (2011. 01 14). *Issues - app-inventor*. Letöltés dátuma: 2011. 01 14, forrás: Issues - app-inventor: http://code.google.com/p/app-inventor-for-android/issues/list?can=2&q=&colspec=ID+Status+Summary+Reporter+Stars&sort=&mode=grid&y=&x=Status&cells=tiles&nobtn=Update

[5] Google. (dátum nélk.). *Reference Documentation*. Letöltés dátuma: 2010, forrás: AppInventor: http://appinventor.mit.edu/explore/content/reference-documentation.html

[6] MIT. (2014). *App Inventor 2.* Forrás: AI2 Concept Cards: http://appinventor.mit.edu/explore/sites/all/files/ConceptCards/ai2/AI2_ConceptCards.pdf

[7] Oracle. (2011). *A Brief History of NetBeans*. Forrás: NetBeans: http://netbeans.org/about/history.html

[8] Sebesta, R. W. (2010). *Concepts of programming languages* (9. kiad.). Upper Saddle River: Pearson.

[9] Wolber, D. (2016). *App Inventor 2 Book: Create Your Own Android Apps.* Forrás: MIT App Inventor 2: http://www.appinventor.org/book2