

Debreceni Egyetem
Informatikai Kar
Könyvtárinformatika Tanszék

VIZUÁLIS PROGRAMOZÁS OKTATÁSA KÖZÉPISKOLÁKBAN

Témavezető:

Dr. Csernoch Mária
egyetemi docens

Készítette:

Csapó Gábor
informatika-könyvtárpedagógia tanár MA

Debrecen
2014

Tartalomjegyzék

Tartalomjegyzék.....	2
Bevezetés.....	4
Jelenleg oktatásban használt módszerek.....	7
Oktatási tervezet a Construct 2 szoftverhez.....	10
Bevezetés, játékfejlesztési lehetőségek bemutatása.....	10
Előkészítés, egyszerű programok készítése.....	10
Feladat 1: Békás autóút.....	11
Feladat 2: Visszatérő béka.....	12
Feladat 3: Ugrálás a párkányokon.....	12
Egyszerű feltételes események kezelése.....	13
Feladat 4: Tájképek.....	13
Feladat 5: Hazaút.....	14
Objektumok és viselkedések komplexebb alkalmazása.....	16
Feladat 6: Kenyerek a ládába.....	16
Feladat 7: Almaszedés.....	18
Feladat 8: Keverőpult.....	19
Összetett feltételes események kezelése.....	21
Feladat 9: Lisztes zsákok megszerzése.....	21
Feladat 10: Űrhajó követése.....	23
Feladat 11: Csillagos éjszaka.....	24
Feladat 12: Labirintus.....	26
Feladat 13: Erre csörög a dió.....	28
Feladat 14: Valahol a sötétben.....	31
Feladat 15: Köszönés.....	31
Egyéni projektmunkák.....	36
Összegzés.....	38
Hivatkozások, irodalomjegyzék.....	42
Saját munka bemutatása.....	47
Köszönetnyilvánítás.....	48

1. Melléklet: Képernyőképek.....	49
2. Melléklet: A Construct 2 ismertetése.....	55
A program leírása.....	55
Licenszek.....	55
Kezelőfelület.....	56
Az alkalmazások felépítése.....	59
Objektumok.....	60
Viselkedések.....	61
Eseménylapok, vizuális programozás.....	62
Hálózatkezelés.....	64
Előnézet, tesztelés.....	65
Exportálás.....	65
Kiegészítések, kódolás.....	66
Korlátok, hátrányok.....	67
Dokumentáció, leírások.....	68
3. Melléklet: Tartalmi kiegészítések.....	70
4. Melléklet: Feladatok technikai kiegészítése.....	71
Feladat 1: Békás autóút.....	71
Feladat 3: Ugrálás a párkányokon.....	71
Feladat 4: Tájképek.....	72
Feladat 5: Hazaút.....	72
Feladat 6: Kenyerek a ládába.....	72
Feladat 7: Almaszedés.....	73
Feladat 8: Keverőpult.....	73
Feladat 9: Lisztes zsákok megszerzése.....	73
Feladat 10: Űrhajó követése.....	74
Feladat 11: Csillagos éjszaka.....	74
Feladat 12: Labirintus.....	75
Feladat 13: Erre csörög a dió.....	75
Feladat 14: Valahol a sötétben.....	76
Feladat 15: Köszönés.....	76

Bevezetés

Dolgozatom témája a vizuális programozás elméleti hátterének ismertetése, oktatási lehetőségeinek felmérése, valamint egy gyakorlati módszer kidolgozása.

Manapság egyre több fiatal érdeklődik a számítógépes szoftverek fejlesztése iránt, szeretnék ötleteiket gyakorlatba átültetni, megvalósítani őket. Sajnos azonban többségük számára, a megfigyelések alapján nem járható út a különböző programozási nyelvek kellő mélységű elsajátítása, mivel leginkább bonyolultnak, semmitmondónak, vagy átláthatatlannak találják és nem látják a lehetséges utat, ami majd a kész produktumukhoz vezet. Ennek ellenére azonban nem elhanyagolható a programozás oktatása, hiszen az általa fejlődő készségekre szüksége van a tanulónak a mindennapokban (W34, 2009).

Korunkban egyre nagyobb teret hódítanak azok a fejlesztői környezetek, amelyek segítségével a felhasználó anélkül a képernyőre viheti ötleteit, hogy akár egy sor programkódot írnia kellene. A kód kézi írása helyett egy grafikus felületet alkalmazva párosítja össze az objektumokat az eseményekkel és az azokhoz kapcsolódó utasításokkal. Az ilyen szoftverek segítségével a fejlesztők könnyedén és gyorsan valósíthatják meg programbeli elképzeléseiket még akkor is, ha számukra a programozási nyelvek többsége átláthatatlan; vagy nem áll rendelkezésükre idő, lehetőség azok megtanulására. Ilyen szoftverek közé tartozik a dolgozatom alapjául szolgáló Construct 2 (W24, 2011); vagy például a Mark Overmars, egyetemi oktató által fejlesztett Game Maker (W1, 2007), valamint számos konkurens alkalmazás is: Stencyl (W2, 2011), GameSalad (W3, 2009), Game Editor (W4, 2011).

Ezen programok oktatásának bevezetése megkönnyítené sok diák számára a szoftverfejlesztés mechanizmusának átlátását és emellett pozitívabb kép alakulna ki bennük a programozást illetően, így a későbbiekben (például felsőfokú tanulmányaik során) tanult programozási módszerek könnyebben elsajátíthatóak lennének. Mivel ezek a szoftverek vizuális felületet alkalmaznak a programozás lebonyolításához, tökéletesen alkalmasak arra, hogy könnyű átláthatóság mellett alakuljon ki és fejlődjön az algoritmikus készség. A programozás során az egyes utasításokat „fogd és vidd” módszerrel választhatják ki a diákok, így a kód szintaktikai pontossága helyett a figyelem az algoritmizáláson van.

Célom, hogy ismertessem ezt a szoftverkategóriát, valamint napjaink egyik leghatásosabb és legdinamikusabban fejlődő vizuális nyelvét és annak fejlesztőkörnyezetét. Emellett egy olyan oktatási terv kidolgozása, amely a vizuális programozást széles körben prezentálja a tanárok, a diákok számára, amellyel az alapvető programozási módszereket elsajátítják, amellett, hogy saját munkájuk eredményét is kézzel foghatónak érzik.

A munkámhoz kutatásokat végeztem oktatási intézményekben. Olyan iskolai intézményeket kerestem fel, melyekben kutattam a jelenleg oktatásban lévő programozási módszereket, valamint felmértem az ezt oktató informatika tanárok tapasztalatait, megfigyeléseit, véleményeit, többek közt az oktatásról, annak hatékonyságáról és a diákok hozzáállásáról.

Az általam választott szoftver a Microsoft Windows környezetben futó Construct 2, melynek hivatalos forrásaiból is merítettem információkat (W35, 2013), valamint az ezt támogató közösségi bázis is segítségemre volt (W22, 2013). A program segítségével alkalmazások és játékszoftverek készíthetők előzetes programozási nyelvek ismerete nélkül, a manapság egyre népszerűbb HTML5 technológiára támaszkodva.

Számomra igen fontos szerepet játszik, hogy a munkám eredményét minél hamarabb működés közben láthassam. Fiatalabb koromban bár érdeklődtem a szoftver- és játékfejlesztés iránt, még nem volt meg a lehetőségem és képességem, hogy mélyebb szinten programozni tanuljak. A programozási nyelvek nehézsége miatt fordultam az „egyszerűbb utat” nyújtó alternatívák felé (bár akkor még csak a fent említett Game Maker alkalmazás volt elérhető). Az elmúlt években felkerestem és teszteltem több kiemelkedő programot ebből a kategóriából, míg végül több éves használatuk és váltogatásuk után választásom a Scirra cég által fejlesztett Construct 2 alkalmazás lett, annak letisztultsága és potenciális lehetőségei miatt. Egy gyakran frissülő szoftverről beszélhetünk, amely lépést tart a legújabb technológiákkal, figyelembe veszi a programmal dolgozó felhasználók igényeit, a konkurencia közül a legkevésbé korlátozott, valamint elérhető árkategóriában van az átlag felhasználó számára. Sajnos általános hiba a hasonló szoftverek tekintetében, hogy (ellentétben a hagyományos programozási nyelvek alkalmazásával) alapvető korlátokba ütközünk, amelyet nem tudunk áthidalni a fejlesztőkörnyezet és annak motorja okából. Más szavakkal megfogalmazva: olyannyira le van egyszerűsítve a fejlesztési folyamat, hogy nem nyílik lehetőségünk bizonyos funkciók implementálására. A másik véglet ezzel szemben a túlbonyolított környezetek, melyeket egyszerűen felfoghatunk úgy is, mint egy hagyományos

programozási nyelv kiszépitett és maximálisan felhasználóbaráttá tett fejlesztői környezetét. A Construct 2 ezen két véglet között képviseli a középutat egyszerű és könnyen érthető fejlesztési folyamattal és minimális korlátozásokkal.

Azért is választottam ezt a szoftvert feldolgozásra, mert tudom, hogy egy-egy programkód megírása igen hosszadalmas művelet, különösképp ha egy nagyobb projektről van szó. Nem is beszélve az adott programozási nyelv elsajátításáról, valamint a gyakorlati tapasztalat megszerzésére fordított időről. Célom egy olyan alternatíva bemutatása, amely segítségével bármely kicsit tapasztaltabb számítógép felhasználó fejleszthet alkalmazásokat programozási nyelvek ismerete nélkül viszonylag rövid idő alatt, látványosan. Illetve egy ilyen lehetőségeket felsorakoztató szoftver oktatásának megtervezése, hogy ennek a lehetősége már fiatal korban eljusson az érdeklődőkhöz.

Jövőbeli célom ezen oktatási terv hatékonyságvizsgálata elsősorban középiskolai környezetben.

Jelenleg oktatásban használt módszerek

Az informatikai eszközök és az internet széles körű elterjedésével szükségessé vált először az informatikai alapismeretek, később már mélyebb szintű informatikai témakörök oktatása. Jelenleg a modern informatika tananyagok szerves része a programozás általános- és középiskolákban egyaránt (W40, 2012).

A vizuális programozáshoz legközelebb álló, hasonló kategóriába sorolható szoftver a Comenius Logo, vagy annak tovább fejlesztett változata, az Imagine Logo. Ennek a szoftvernek a használatát számos magyar és külföldi iskolában is oktatják. A Comenius Logo segítségével a diákok a Logo nyelv utasításait használva rajzolnak ki a képernyőre különböző alakzatokat, ezzel elősegítve az alapvető algoritmikus gondolkodás kialakulását, valamint a programozási nyelvek átláthatóságát.

Egy másik elterjedőben lévő, a programozás logikáját szemléltető megközelítés a Lego robotok programozása. A robot megépítése után (amely körülményektől függően az osztály vagy a tanár feladata) egy szoftver segítségével könnyedén és egyértelműen logikai blokkok beszúrásával adhatóak utasítások a robotnak. Ahol az intézmény erőforrásai megengedik, külön Lego robot építő/programozó szakkörök is szerveződhetnek.

Középiskolákban már a hagyományos programozási nyelvek vannak elterjedve (például Pascal, C, C++), de előfordulhat, hogy az algoritmikus gondolkodást komplexebb táblázatkezelő függvények segítségével oktatják.

A dinamikus fejlődő technológiával lépést tartva (W45, 2012) (W46, 2014) egyre jobban elterjedőben van a mobil programozás is, amely során a diákok saját mobil alkalmazásokat fejlesztenek a népszerűbb mobil operációs rendszerekre.

Kutatásom során a jelenleg alkalmazott programozásoktatási módszerek oktatásbeli szerepét, hatékonyságát és elért eredményeit is vizsgáltam. Valamint áttekintettem a Logo nyelv oktatásához használt tankönyvek feladatait (K1, 2009) (K2, 2009).

A programozás oktatásának egyértelmű célja az algoritmikus gondolkodás fejlesztése és annak későbbi tanításának megkönnyítése, valamint egy alapvető tudás átadása, amelyet a tanulók a későbbiekben alkalmazni tudnak.

A Logo nyelvek esetében a diákok könnyen szembesülnek a munkájuk eredményével grafikus módon. A beírt utasítások okozta változásokat azonnal láthatják a képernyőn, és ez pozitívan befolyásolja őket. A szoftver egy játékos és grafikus környezetet alkalmaz, így a fiatalabb diákok számára ideális, mert nem bonyolódik bele, zavarodnak össze komplexebb

menüpontok és bonyolult felhasználói felületek láttán. Korábban a legtöbb általános iskolában Pascal programozási nyelvet oktattak bevezetésképpen a programozás világába a diákoknak. Azonban a Comenius Logo használata és tanítása gyorsabb eredményeket mutatott.

A célréteg elsősorban az általános iskolák hatodik és hetedik évfolyamos diákjai (van olyan iskola, ahol már negyedik előtt is tanítják). Ebből adódóan a legtöbb intézményben két évfolyamos bontás alakult ki.

A tanulói réteg korosztályából kiindulva a diákok egy része nem hajlandó komolyan venni a feladatokat, vagy nem képes az ehhez szükséges algoritmikus gondolkodásra. Az önálló feladatok megoldása (természetesen azok nehézségi szintjétől függően) nehézségekbe ütközik, esetenként szövegértési problémák is felmerülnek. A tanultakat gyakorlatban alkalmazni csupán egy kisebb réteg képes önállóan. A diákok érdeklődésének növelésével növelhető a tanórák teljesítménye, ezért a tanár feladata olyan munkák összeállítása, amelyek a csoporthoz igazodva megragadják a tanulók figyelmét és ezért érdeklődéssel hajtják végre azokat. Ezek ellenére azonban a szoftver és a nyelv egyszerűsége miatt már az első órán szembesülnek a diákok munkájuk eredményével: egyszerű mozgatást és rajzolást valósítanak meg.

A megkérdezett tanárok szerint a fiatalok szeretik a szoftver használatát, mivel színes és látványos megjelenítést alkalmaz a kezelőfelületet és forráskód futtatásának eredményét illetően. A későbbiekben, amikor a feladatok nehezednek, illetve a programnyelv komolyabb utasításainak és komplexebb algoritmikus gondolkodás alkalmazásának igényének fellépésekor ez az érdeklődés, lendület fokozatosan csökken. A tanulók elsősorban szórakozási eszközként tekintenek a számítógépekre, így főként játszani szeretnének. Az irányított munkát kevésbé preferálják.

Középiskolában már egy fokkal könnyebb a programozási nyelvek oktatása. A tanulók fejlettebb és érettebb gondolkodással rendelkeznek és könnyebb őket ráhangolni az algoritmikus feladatmegoldásra. Ennek ellenére az érettségit tett tanulók által nyújtott teljesítmény mégis elmarad az elvárttól.

Ennek oka elsősorban a szűk óraszámra tehető, ugyanis a programozás oktatására rendelkezésre álló óraszám mellett (az éltanulók kivételével) még a nyelv alapjai sem sajátíthatóak el az előírt mélységben. A másik oka a diákok alulteljesítésének a programozási nyelvek érdektelensége számukra. Egy átlagos tanuló nem találja kellően érdekesnek egy parancssoros felületen programkódot írni, különösképp ha az informatika nem is tartozik a

kedvelt tantárgyai közé. Az így kialakuló lelki eltávolodás eredményeképp még kevesebb algoritmikus gondolkodást fog elsajátítani, holott arra szüksége van a mindennapi élet során. Bizonyos szakiskolákban felmerülhet az a helyzet is, hogy az informatika olyan minimális hangsúllyal bír az iskola szakterülete mellett, hogy a tanárok és tanulók is közös véleményen vannak, miszerint annak oktatása majdhogynem már felesleges. A kerettantervben a szakiskoláknál csak választható tantárgyként szerepel az informatika (W47, 2013)

A felsorolt problémák egyik megoldása lehet a vizuális programnyelvek bevezetése középiskolai oktatásba. A tanulók egy egyszerűen elsajátítható és látványos (a munkájuk eredményét azonnal visszatükröző) módszert tanulnának, amely nemcsak alkalmas az algoritmikus készségek fejlesztésére, de visszatükrözi a hagyományos programozási nyelvek esetében tanult elemeket is, így a későbbiekben erre a tudásra támaszkodva szintén könnyebb más nyelvek elsajátítása.

Ha pedig a tanuló a vizuális nyelvekkel kíván a későbbiekben is foglalkozni, akkor az alapok mellett akár önképzéssel is megtanulhat komplexebb alkalmazásokat fejleszteni, hiszen maga a nyelv és a környezet alkalmas arra, hogy piacképes programok készüljenek a segítségével.

Oktatási tervezet a Construct 2 szoftverhez

A Construct 2 egyes modulok oktatásának hosszát a csoport teljesítőképessége, valamint a rendelkezésre álló tanórák száma alapján érdemes eldönteni. Jelen dolgozat feladatai középiskolás diákok számára lettek összeállítva, így „kiegészíti” és továbbviszi az általános iskolákban oktatott Logo szoftverek tananyagát.

Bevezetés, játékfejlesztési lehetőségek bemutatása

Első lépésként röviden ismertetjük a diákokkal a szoftver célját és különböző látványos mintaprogramok bemutatásával célunk elnyerni az érdeklődésüket iránta (T1).

- Ebben a fázisban fontos, hogy kialakuljon a diákok érdeklődése és láthassák, hogy saját maguk mire lehetnek képesek a Construct 2-t használva. Bemutatunk a későbbiekben elkészítésre kerülő egyszerű, ám látványos játékszoftvereket, majd ezt követően olyan komplex és nem csak látványban, de játékmenetben is összetett alkotásokat, amelyek elkészíthetők egy olyan fejlesztő által, aki aktívan gyakorolja otthon is a programfejlesztést ezzel a környezettel. További nagyszerű forrás a szoftverrel készített produktumok bemutatására a fejlesztő cég publikációs oldala, a Scirra Arcade (W29, 2013).

Előkészítés, egyszerű programok készítése

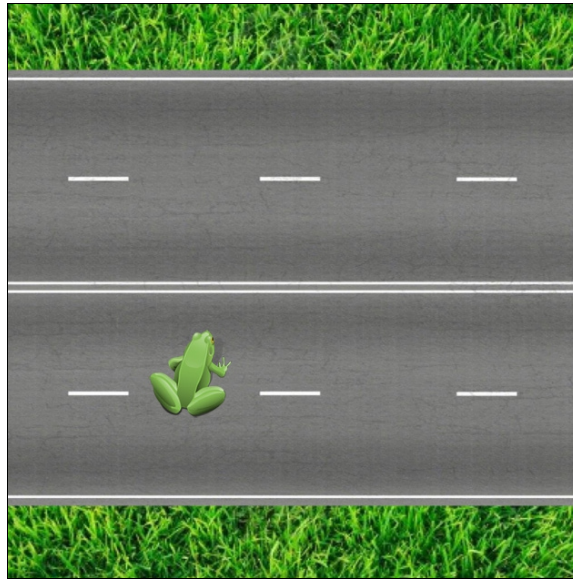
Minden egyes alkalommal (kivéve amennyiben az előző munkát folytatják a diákok) új projektet kell létrehozni és azt egyértelműen az előzőektől elkülönített helyre kell menteni (T2). Első lépésként mindenkinek önállóan új projektet kell indítania. A tanár szolgálgatóként, melyet az első pár példaprogramnál a diákok követnek, hogy megszokják a program kezelését és a funkciók elhelyezését. A feladatok megoldásánál elég az új tananyagot prezentálnia a tanárnak, az önálló munka elvárt a csoport részéről. Ezt követően meg kell ismerkedjenek az alkalmazások készítésének egy egyszerű módjával. Figyelní kell arra, hogy minél több területét érintsék a programnak, főként azokat, amelyekkel a későbbiekben is majd nem minden alkalommal dolgozni fognak. Egy olyan egyszerű program készítése a cél, amely interakciót tesz lehetővé és látványos a diákok számára (T3).

Egy új projektet minden esetben annak elnevezésével és alapbeállításával kezdünk. A projekt elnevezését a projekt tulajdonságainál a Name mezőben tehetjük meg. Az alapbeállítások pedig a következők: A szerzőhöz (*Author*) minden diák írja be a saját nevét.

Az ablak mérete (*Window Size*) 640×480 pixel legyen (F1). Az elrendezés (*Layout 1*) méretét és margóit igazítsuk az alkalmazás ablakméretéhez (F2). A feladatok részletezésénél csak az ettől eltérő beállításokra térünk ki.

Feladat 1: Békás autóút

Segítsünk a békának átkelni az autóúton. A probléma megoldásához egy úttest háttérre és egy 8 irányba mozgatható béka objektumra van szükségünk (1. ábra, *autout.capx*).



1. ábra, A béka objektum az úttesten.

Az autóút elkészítéséhez hozzunk létre egy új üres projektet *autout* néven. Nevezzük el a projektünket „Autóút”-nak. Ennél a projektnél az alapbeállításoktól eltérően az ablak és az elrendezés, valamint a margójának mérete 640×640 pixel legyen.

Elsőként készítsük el a sztrádát, amelyhez a *sztrada.jpg* képet használjuk fel.

Szúrjunk be egy új *Tiled Background* objektumot, melyet nevezzünk el „sztrada” néven. Forrásképként nyissuk meg a *sztrada.jpg* képet, majd a szerkesztőfelületet bezárva, az objektum tulajdonságainál állítsuk a méretet szintén 640×640 pixel nagyságúra, a pozícióját pedig (0;0) koordinátákra (F3).

Adjuk hozzá a projektünkhöz a békát, ami egy *Sprite* típusú objektum. Az objektum neve legyen „beka”. Képként importáljuk be a *beka.png* átlátszó háttérű képfájlt. A békát a sztrada alsó sávjában helyezzük el.

A „beka” nevű, *Sprite* típusú objektumhoz rendeljük hozzá a *8Direction* viselkedést (14. ábra) (F4).

Futtassuk le az elkészült programot, amit a *Run layout* gombra kattintva tehetünk meg. A böngészőben futó programunkban a billentyűzet kurzorvezérlő nyíljaival mozgatható a béka.

Feladat 2: Visszatérő béka

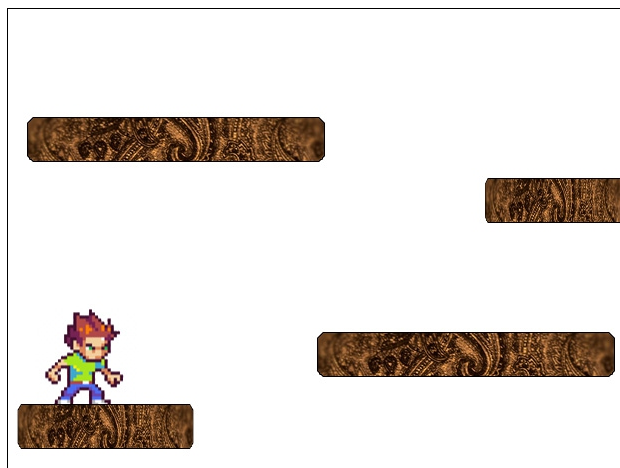
Kezeljük le, hogy az „autout” projektben amennyiben a felhasználó kimozzgatja a békát a képernyőről, az az ellentétes oldalon ismét megjelenjen.

A szerkesztőfelületre visszatérve rendelünk a „beka” objektumhoz egy újabb viselkedést, a Wrap-ot.

Teszteljük le a változásokat egy újabb előnézet indításával és navigáljuk ki a békát a képernyőről. Láthatjuk, ahogy megjelenik a szemközti oldalon.

Feladat 3: Ugrálás a párkányokon

Juttassuk fel a fiút a legmagasabb pontra. Hozzunk létre egy fiú objektumot, amelyhez platform mozgási viselkedést rendelünk, illetve egy talaj elemet, amelyen nem mehet keresztül a fiú (2. ábra, *magaslat.ccapx*).



2. ábra, Az ugráló fiú elhelyezve a legalsó párkányon.

Hozzunk létre egy üres projektet „magaslat” néven és hajtsuk végre az alap beállításokat:

Adjunk hozzá egy *Sprite* objektumot a projekthez, amit „talaj”-nak nevezzünk el. Képként importáljuk a *talaj.png* fájlt.

A „talaj” objektum kapja meg a *Solid* viselkedést, hogy ne legyen átjárható (F5).

Pozicionáljuk az elrendezésen a „talaj” objektumot a képhez (2. ábra) hasonló módon. Az elemet a projektfából többször is illesszük be az elrendezésbe és ahol szükséges, méretezzük át (F6).

Adjunk hozzá az elrendezéshez egy új *Sprite* objektumot „fiu” néven. Képként nyissuk meg a *fiu.png* képfájlt. Pozicionáljuk a legalsó talaj elem felső határára.

A „fiu” objektumhoz rendeljük hozzá a *Platform* viselkedést (F7).

Futtassuk le a programunkat, amelyben a kurzorvezérlő nyilak segítségével lehet mozgatni a „fiu” objektumot.

Egyszerű feltételes események kezelése

A program fő erőssége az eseménylapok kezelése. Ezek a feladatok az eseménylapok használatát mutatják be kezdő szinten. A tanár belátásán múlik, hogy milyen mélységeig kalauzolja el a diákcsoportot ebben a modulban a hozzáadott események számát illetően, azonban célszerű továbbra is a látványos és „kézzel fogható” elemekre szorítkozni.

Feladat 4: Tájképek

Készítsünk egy diavetítő programot, amellyel megnézhetjük a kiránduláson készült képeket, egymás után. A megoldáshoz 3 darab képet fogunk használni, amelyek háttérként szolgálnak, majd a háttérképek között kattintásra tudunk váltani.

A megoldáshoz szükségünk lesz egy több képet tartalmazó háttér objektumra és egy gomb objektumra, amellyel váltogathatunk a képek között (3. ábra, *hattercsere.ccapx*).



3. ábra, A diasorozat első képe és a kamera léptetőgomb megjelenítése.

Hozzunk létre egy üres projektet „hattercsere” néven. Végezzük el az alapbeállításokat:

Adjunk hozzá az elrendezéshez egy „hatter” nevű *Sprite* típusú objektumot.

A „hatter” objektum rendelkezzen három képkockával, melyeket a *hatter01.jpg*; *hatter02.jpg*; *hatter03.jpg* képfájlokkal töltsünk fel (F8).

Állítsuk be, hogy a képek váltása milyen gyorsan történjen, valamint, hogy ismétlődjenek a képek, ha elértünk a képsorozat végére.

A „hatter” objektum alapértelmezett animációját („Default”) konfiguráljuk a következőképpen (17. ábra):

A *Speed* mezőbe 10-es értéket írunk (gyors váltás), illetve a *Loop* opciót állítsuk *Yes*-re.

Az animáció beállításai a kijelölését követően az *Animations* panelben jelennek meg (17. ábra) (F9).

Pozicionáljuk a „hatter” elemet az elrendezés közepére. A kép helye legyen a (320;240) koordinátájú pont.

Hozzuk létre a gomb elemet.

Adjunk hozzá a projektünkhöz egy új *Sprite* típusú objektumot, amelyet nevezzünk el „gomb”-nak. Forrásképként nyissuk meg a *tovabbgomb.png* fájlt, majd pozicionáljuk az elrendezés jobb alsó sarkába.

Adjuk hozzá a projekthez a *Mouse* objektumot „eger” néven (F10).

Az egyik képről a másikra váltás akkor történjen meg, ha rákattintunk a gombra.

Az eseménylapon hozzunk létre egy új eseményt az „eger” objektum *On object clicked* feltételére, amelynek paramétereleme a „gomb” objektum legyen (F11).

Állítsuk be a „hatter” objektum animációkezelését.

Utasítsuk a „hatter” objektumot az animáció elindítására az imént létrehozott eseményblokkon belül (*Animation / Start*).

A képkocka váltása után meg kell állítanunk az animációt. Ehhez egy újabb eseményt kell létrehoznunk.

A „hatter” objektum eseményei közül válasszuk az *Animations* kategóriából az *On any frame changed* feltételt (F12).

Az új eseményblokkhoz határozzuk meg az animációt megállító utasítást (*Animations / Stop*). Az eseménylap felépítése a 18. ábrán megtekinthető.

Futtassuk le a programot.

A „gomb” elemre kattintva a „hatter” objektum animációja elindul, majd egy kép váltása után ismét megáll.

Feladat 5: Hazaút

Kísérjük haza a lányt a gombaházba és amikor hazaér, üdvözljük, játszunk le egy hangfájlt.

Hozzunk létre egy lány objektumot, amelyhez platform mozgást rendelünk, egy ház objektumot, (ami egy a programablaknál nagyobb elrendezésen található), egy talaj objektumot, amin nem mehet keresztül a lány, illetve egy fű objektumot (4. ábra, *hazaut.capx*).



4. ábra, A lány objektum közel a házhoz.

Hozzunk létre egy üres projektet „hazaut” néven. Végezzük el az alapbeállításokat:

Ennél a projektnél az alapbeállításoktól eltérően az elrendezés méretét állítsuk 1400×480 pixel méretűre.

Hozzuk létre a lány objektumot.

Adjunk a projekthez egy új *Sprite* típusú objektumot, amelyet nevezzünk el „lány” néven. Forrásképként nyissuk meg a *lany.png* képet.

A „lány” objektumhoz rendeljük hozzá a *Platform*, illetve a *ScrollTo* viselkedést (F13).

Illesszük be a ház elemet.

Szűrjünk be egy új *Sprite* objektumot, majd nevezzük el „haz”-nak. Forrásképként nyissuk meg a *gombahaz.png* fájlt.

Hozzuk létre a talaj objektumot.

Adjunk hozzá a projekthez egy újabb *Sprite* objektumot „talaj” néven. A grafikai szerkesztőben rajzoljunk egy zöld színű lapos téglalapot, majd vágjuk a képet a téglalap méretére (19. ábra).

A grafikai szerkesztő eszköztárából könnyedén a grafika méretére vághatjuk a képeket (*Crop transparent edges*). Ezzel a beállítással a kép körülvágásához kijelölés nem szükséges.

A „talaj” objektumhoz rendeljük hozzá a *Solid* viselkedést, illetve méretezzük át, hogy szélességben közel kitöltse az elrendezést (F14).

Adjuk hozzá a fűvet a projekthez.

Illesszünk be egy újabb *Sprite* objektumot, melynek legyen a neve „fu”. Képnek nyissuk meg a *fu.png* fájlt. Az objektumot méretezzük át, hogy szélességben közel kitöltse az elrendezést.

Pozicionáljuk az elemeket a képen látható módon (20. ábra) (F15).

Állítsuk be a hangfájlok kezelését.

Adjuk hozzá a projekthez az *Audio* objektumot, aminek adjuk a „hangkezeles” nevet.

Importáljuk be a projekt fába a *hazaer.wav* hangfájlt.

Az eseménylapon rendeljünk eseményt a „lany” objektumhoz, hogy amennyiben érintkezik a „haz” objektummal, a blokk feltétele teljesüljön (*On collision with haz*).

Az esemény definiálásánál ki kell választanunk a „haz” objektumot.

Rendeljünk utasítást az eseményblokkhoz, hogy a „hangkezeles” objektum lejátsza az importált „hazaer” hangot (*Play*).

Futtassuk le a programot.

A kurzormozgató nyilak segítségével lehet mozgatni a „lany” objektumot bal és jobb irányokba, valamint a felfele mutató nyíllal ugrani is tud. A képernyő jobb széléhez közeledve az elrendezés görgetésre kerül.

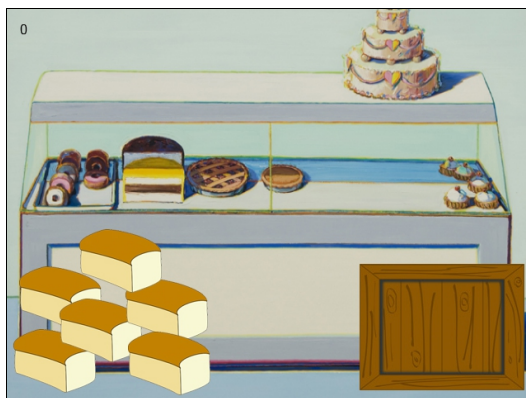
Objektumok és viselkedések komplexebb alkalmazása

Ez a modul olyan feladatok megoldását mutatja be, amelyek megvalósításához összetettebb algoritmusokra van szükség. Túlmegyünk az alapszinten választott viselkedéseken, objektumokon (azok további felhasználása mellett). Néhány nem szükségszerűen a fejlesztések középpontjában lévő, de egyaránt hasznos elemet kell bemutatni a diákoknak, hogy szélesedjen látóterük a programozási folyamatok, rendelkezésre álló eszközök és alkotási lehetőségeik tekintetében.

Feladat 6: Kenyerek a ládába

Pakoljuk be a kenyereket a ládába és közben számoljuk meg, hogy hányat pakoltunk be.

Hozzunk létre egy háttér, egy kenyér és egy láda objektumot, valamint egy szövegdobozt, amibe megszámloljuk az elpakolt kenyerek számát (5. ábra, *pakolas.capx*).



5. ábra, A bal oldalon található kenyereket pakoljuk a jobb oldali ládába (kiindulási helyzet).

Hozzunk létre egy üres projektet „pakolas” néven. Végezzük el az alapbeállításokat.

Adjuk hozzá a projekthez a hátteret.

Illesszünk be egy *Tiled Background* objektumot, amelynek adjuk a „hatter” nevet. Forrásként nyissuk meg a *hatter.jpg* képet. Az objektum mérete 640×480 pixel nagyságú legyen, a pozícióját pedig állítsuk (0;0) koordinátákra.

Adjunk hozzá a projekthez a ládát.

Szúrjunk be egy új *Sprite* objektumot „lada” néven a *lada.jpg* képfájl felhasználásával.

Illesszünk be egy objektumot, ahol meg tudjuk jeleníteni az átpakolt kenyerek számát, egy *Text* objektum felhasználásával, aminek adjuk az „eredmeny” nevet. Szövegnek pedig adjuk meg a „0” kezdőértéket (F16). Az eseménylapon hozzunk létre egy szám típusú globális változót „szam” néven (21. ábra) (F18). A változó kezdőértéke „0” legyen.

Adjunk hozzá egy újabb *Sprite* objektumot a programhoz, melyet mentünk „kenyer” néven. Képként használjuk fel a *kenyer.png* képet.

Az objektumokat pozicionáljuk a képnek megfelelően (5. ábra). A kenyér elemet többször is be kell illeszteni a projektfábol.

A „kenyer” objektumhoz rendeljük hozzá a *Drag & Drop* viselkedést (F17). Definiáljunk egy eseményblokkot a „kenyer” objektumra az *On drop* feltétellel (F19).

Vizsgáljuk meg, hogy a „kenyer” és a „lada” objektumok átfedésben vannak-e.

Adjunk hozzá egy újabb feltételt az előző eseményhez, amelyik a „kenyer” objektum *Is overlapping another object* feltételét vizsgálja a „lada” objektumra beállítva.

Amennyiben mindkét feltétel teljesül, a program növelje meg a „szam” változó értékét 1-gyel (*System / Add to*), majd írassuk ki az „eredmeny” objektumba *Set text* utasítással a „szam” változó értékét úgy, hogy az utasítás paramétereként a változó nevét adjuk meg.

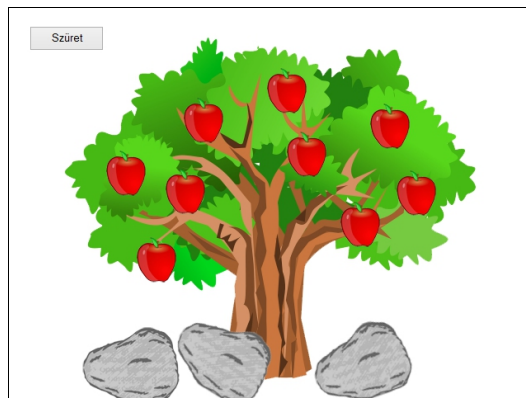
A fenti utasításokat követően a program törölje ki a „kenyer” objektum adott példányát (*Destroy*). Az eseménylap teljes felépítése a 22. ábrán látható.

Futtassuk le a programot.

Az egér bal egérgombjának nyomva tartásával a kenyereket bepakolhatjuk (behúzhatjuk) a ládába.

Feladat 7: Almaszedés

Szüreteljük le a kövek mellé ültetett almafát egy gombra kattintva. Hozzunk létre egy fa objektumot a tövében kövekkel, illetve rajta elhelyezett almákkal. Adjunk hozzá egy gombot, amely megnyomása után az almák a beállított viselkedés miatt leesnek a kövekre, amelyeken fennakadnak vagy legurulnak róluk (6. ábra, *alma.ccapx*).



6. ábra, Almák elhelyezve a fán.

Hozzunk létre egy üres projektet „alma” néven. Végezzük el az alapbeállításokat.

Szúrjuk be a fa elemet.

Adjunk hozzá a projekthez egy új *Sprite* objektumot „fa” néven. Képként használjuk a *fa.png* képet és pozicionáljuk az objektumot az elrendezés közepére.

Adjuk hozzá a projekthez az almát.

Illesszünk be egy új objektumot „alma” néven, a típusa pedig *Sprite* legyen. Forrásképként nyissuk meg az *alma.png* képfájlt. Ezt követően végezzük el az alma tulajdonságainak beállítását:

Az „alma” objektumhoz rendeljük hozzá a *Physics* viselkedést.

Pozicionáljuk tetszőlegesen az „alma” elemet a „fa” objektumra, illetve illesszünk be több példányt is belőle (F20).

Adjunk hozzá egy újabb *Sprite* típusú objektumot a projekthez, amelynek „ko” legyen a neve és forrásképként a *ko.png* fájlt használja.

A „ko” elemhez is rendeljük hozzá a *Physics* viselkedést és a tulajdonságainál az *Immovable* opciót állítsuk *Yes*-re (23. ábra) (F21).

Helyezzük el a „ko” objektum több példányát is a „fa” objektum tövébe. Fontos, hogy ne legyen átfedés az „alma” és „ko” objektumok között.

Gátoljuk meg, hogy az „alma” objektumok leessenek a fáról az elrendezés betöltődésekor az alapértelmezett *Physics* viselkedés beállítások miatt.

Az eseménylapon hozzunk létre egy új eseményblokkot, amely a *System* objektum *On start of layout* feltételét vizsgálja (F22).

Rendeljük hozzá egy új utasítást az „alma” elemre vonatkozóan: *Set world gravity*. Az utasítás paramétere 0 legyen (F23).

A szüret megkezdéséhez illesszünk be egy *Button* típusú objektumot „gomb” néven, majd helyezzük el az alkalmazás bal felső sarkába. A gomb felirata „Szüret” legyen.

Definiáljunk egy új eseményblokkot, amelyben a „gomb” objektum *On clicked* feltételét ellenőrizzük.

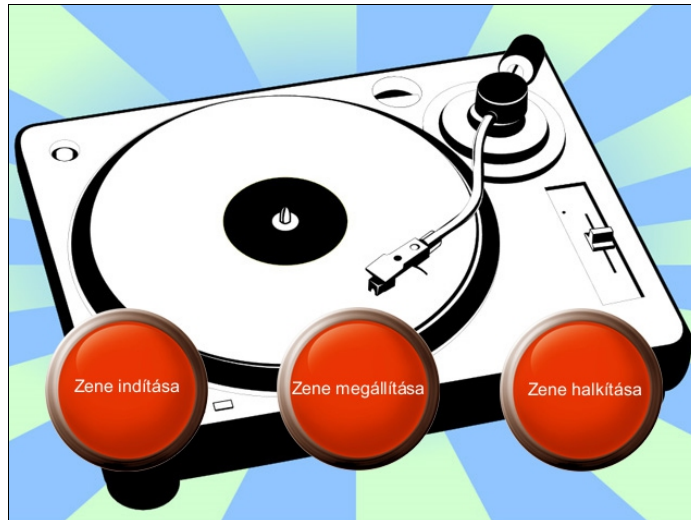
Adjunk hozzá ehhez az eseményblokkhoz egy utasítást az „alma” elemre mutatóan: *Set world gravity*. Az utasítás paramétere 10 legyen (F24).

Futtassuk le a programot.

A Szüret gombra kattintva az „alma” elemek szimulált gravitációja megváltozik és leesnek a fáról, rá a kövekre.

Feladat 8: Keverőpult

Íranyítsuk a zenét egy keverőpultról. A feladat megoldásához szükségünk lesz egy háromgombos keverőpultra, melyen a gombok funkciói a következők: Az első gombra kattintva a program zenét játsszon le ismétlődően, a második gomb állítsa meg a zenét, a harmadik gomb pedig halkítsa le azt (7. ábra, *keveropult.capx*). A megvalósításhoz hozzunk létre egy háttér elemet és 3 darab gomb objektumot felirattal.



7. ábra, A keverőpult és a gombok helyzete.

Hozzunk létre egy üres projektet „keveropult” néven. Végezzük el az alapbeállításokat.

Szűrjük be a program háttérét.

Illesszünk be egy *Tiled Background* objektumot „hatter” néven. Forrásképként nyissuk meg a *hatter.jpg* képfájlt. Az objektum mérete 640×480 legyen, pozícióját pedig (0;0) koordinátákra definiáljuk.

Adjuk hozzá a programhoz a gombokat.

Szűrjük be egy *Sprite* típusú objektumot és adjuk neki az „inditas” nevet. Képként nyissuk meg az *piros_gomb.png* fájlt.

Klónozzuk az „inditas” objektumot kétszer és illesszük be a másolatokat is az elrendezésre. A másolatoknak adjuk a „megallitas” és „halkitas” neveket.

Adjunk hozzá a projekthez egy *Text* típusú objektumot „inditas_felirat” néven. A megjelenített szöveg „Zene indítása” legyen, a szöveg színe pedig fehér (F25).

Klónozzuk az „inditas_felirat” elemet kétszer és adjuk hozzá a duplikációkat is az elrendezéshez. Az új objektumok neve „megallitas_felirat” és „halkitas_felirat” legyen. A másolatok megjelenített szövege „Zene megállítása” és „Zene halkítása” legyen szintén fehér betűszínnel.

Pozicionáljuk az objektumokat a képen látható módon (7. ábra). A gomb elemekre a nevéhez kapcsolódó megegyező felirat kerüljön („inditas” objektumra az „inditas_felirat”).

Adjuk hozzá a programhoz az *Audio* objektumot „hangkezeles” néven, illetve a *Mouse* objektumot „eger” néven.

Importáljuk be a projekt fájlba a *Music* mappába a *zene.flac* audiofájlt.

Az eseménylapon állítsuk be a gombok funkcióit.

Hozzunk létre három új eseményt, amelyek rendre az „eger” objektum *On object clicked* feltételét vizsgálják egyenként az „indítás” majd „megallítás” és végül a „halkítás” objektumokra vonatkoztatva.

Az „indítás” elemre való kattintáskor hívjuk meg a „hangkezeles” objektum *Play* utasítását *Audio file: zene; loop: Looping* és *Volume: -10 dB* beállításokkal (F26).

A „megallítás” objektumra való kattintást vizsgáló eseményhez adjuk meg a „hangkezeles” elem *Stop* parancsát.

A „halkítás” objektumra való kattintást vizsgáló eseményblokkhoz pedig definiáljuk a „hangkezeles” objektum *Set volume* utasítását *-20 dB* értékkel.

Futtassuk le a programot.

A gombokra kattintva elindítható, megállítható a zene, valamint csökkenthető a hangerőssége.

Összetett feltételes események kezelése

Ebben a modulban a diákok elsajátítják a grafikus programozás, más szóval az eseménylapok kezelésének magasabb szintjét és bonyolultabb algoritmikus problémák megoldását. Ki kell térni az események elrendezésének fejlettebb használatára (például alesemények kezelése), illetve a kifejezések (például véletlenszám generálás és objektumok koordinátáinak lekérdezése) alkalmazására is. A kifejezések oktatásához ajánlott kiemelni a programdokumentáció releváns fejezetét (W33, 2013).

Feladat 9: Lisztes zsákok megszerzése

Hullanak a liszteszsákok. Kapjunk el annyit amennyit csak tudunk.

A feladat megoldásához hozzunk létre egy háttér, majd egy zsák objektumot, amelynek példányai véletlenszerűen megjelennek a képernyőn és esnek lefelé (8. ábra, *liszt.ccapx*). Az egérrel rákattintva tudjuk elkapni a zsákokat.



8. ábra, A liszteszsák, amit esés közben el kell kapjunk.

Hozzunk létre egy üres projektet „liszt” néven. Végezzük el az alapbeállításokat.

Adjuk hozzá a projekthez a hátteret.

Illesszünk be egy *Tiled Background* objektumot, amelynek adjuk a „hatter” nevet. Forrásként nyissuk meg a *hatter.jpg* képet. Az objektum mérete 640×480 pixel nagyságú legyen, a pozícióját pedig állítsuk (0;0) koordinátákra.

Adjunk hozzá a projekthez a liszteszsákot.

Szúrjunk be egy *Sprite* típusú objektumot, amelyet „zsak” néven mentsünk el és képként nyissuk meg a *zsak.png* képfájlt.

A „zsak” objektumhoz rendeljük hozzá a *Physics* viselkedést, majd helyezzük el a képen látható helyre (8. ábra) (F27).

Illesszük be a *Mouse* objektumot a programba „eger” néven.

Állítsuk be, hogy a „zsak” objektum minden másodpercben az elrendezés felett véletlenszerűen jelenjen meg.

Az eseménylapon hozzunk létre egy új eseményt, amely a *System* objektum *Every X seconds* feltételét hívja meg. Paraméterként adjunk meg 1 másodpercet (F28).

Kapcsoljuk hozzá a *System* elem *Create object* utasítását, amely hozza létre a „zsak” objektumot a következő koordinátákon (F29):

X: $\text{int}(\text{random}(640))$

Y: -100

Töröljük ki az adott zsákot, amennyiben kiesik az elrendezésen kívülre.

Hozzunk létre egy új eseményblokkot, amelyben a „zsak” objektum *Compare Y* feltételével megvizsgáljuk az elem Y koordinátáját. Konfiguráljuk úgy, hogy 550 értéknél (F3) nagyobb szám esetén aktiválódjon a blokk.

Kapcsoljuk hozzá a „zsak” objektum *Destroy* utasítását.

A program futása során egy idő után számos „zsak” elem kerülne a képernyőn kívülre és folyamatosan növelné az alkalmazás számításigényét. Erre lépésre optimalizációs szempontból van szükség.

Definiáljunk egy új eseményt az „eger” objektum *On object clicked* feltételével, amely a „zsak” elemre hivatkozik.

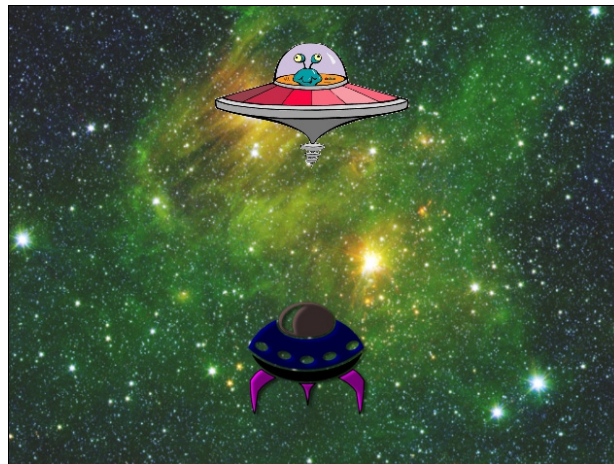
Utasításként adjuk meg a „zsak” objektum *Destroy* parancsát.

Futtassuk le a programot.

1 másodpercenként megjelenik egy új zsák a képernyőn, tehát létrejön a „zsak” objektum egy új példánya, majd a fizikai szimuláció hatására leesik. Az egérrel rákattintva elkaphatjuk a zsákot, tehát kitöröljük az adott példányt.

Feladat 10: Űrhajó követése

Repüljünk el űrhajókkal az üldözőnk elől. Hozzunk létre egy háttér, egy űrhajó és egy idegen űrhajó objektumot. Az űrhajónkat (9. ábra, piros űrhajó) balra és jobbra mozgathatjuk, amit az x tengelyen követ az idegen hajó (9. ábra, lila jármű), azonban a mozgás megkezdését 1 másodperces késéssel veszi csak észre (9. ábra, *urhajo.captx*).



9. ábra, A piros űrhajó mozgását fogja követni a kék űrhajó, mintha a kék árnyéka lenne a pirosnak.

Hozzunk létre egy üres projektet „urhajo” néven. Végezzük el az alapbeállításokat.

Adjuk hozzá a projekthez a háttér.

Illesszünk be egy *Tiled Background* objektumot, amelynek adjuk a „hatter” nevet. Forrásként nyissuk meg a *hatter.jpg* képet. Az objektum mérete 640×480 pixel nagyságú legyen, a pozícióját pedig állítsuk (0;0) koordinátákra.

Saját űrhajó objektum beszúrása

Illesszünk be egy új *Sprite* objektumot „urhajo” néven és képként nyissuk meg az *urhajo_sajat.png* fájlt.

Rendeljük hozzá az „urhajo” objektumhoz a *Bound To Layout* és *8 Direction* viselkedéseket. Az utóbbi esetében a *Directions* opciót állítsuk *Left & right*, illetve a *Set angle* opciót *No* értékekre (24. ábra) (F30).

Pozicionáljuk az elemeket a képen látható módon (9. ábra).

Az eseménylapon definiáljuk az „idegen” objektum mozgását. Hozzunk létre egy eseményblokkot, amelybe megadjuk az „urhajo” elem mozgását vizsgáló *Is moving* feltételt.

Állítsuk be a mozgás érzékelésének késleltetését.

Rendeljük hozzá a *System* objektum *Wait* utasítását 1 mp-es időintervallum beállítással (F31).

Határozzuk meg az idegen űrhajó mozgását egy olyan eseményblokkban, amely csak a késleltetés után kerül meghívásra.

A blokkhoz definiáljunk egy aleseményt, amely a *System* elem *Trigger once while true* feltételét hívja meg (F32).

Definiáljunk egy utasítást, amely megadja az „idegen” objektum koordinátáit.

Hívjuk meg az „idegen” objektum *Set position* parancsát a következő paraméterekkel (F33):

X: urhajo.X

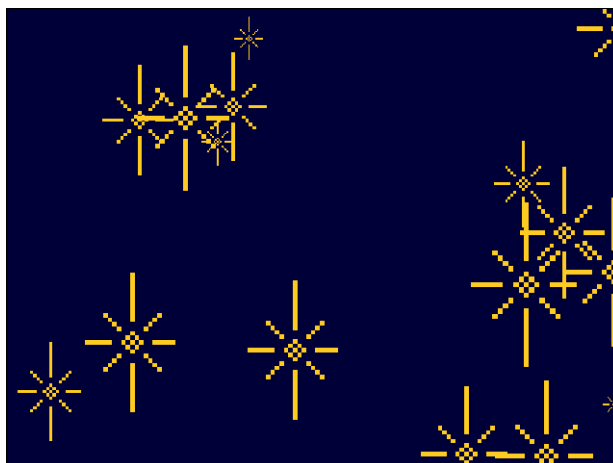
Y: 370

Futtassuk le a programot.

A kurzormozgató bal és jobb nyilakkal mozgathatjuk az „urhajo” objektumot. A mozgás kezdetétől számítva 1 mp várakozást követően az „idegen” objektum elkezd követni az űrhajót, tehát felveszi az „urhajo” elem x koordinátáját.

Feladat 11: Csillagos éjszaka

Töltsük fel az égboltot különböző méretű csillagokkal. A megoldáshoz szükségünk lesz egy csillag objektumra, amelyet másodpercenként véletlenszerű pozícióba és különböző méretben több példányban megjelenítünk a képernyőn (10. ábra). A csillagok létrehozását a billentyűzet bármelyik gombjának megnyomásával megszakíthatjuk (*csillagos_ejszaka.ccapx*). A feladat célja, hogy megismerkedjünk a hagyományos programozási nyelvekben már közismert *while* ciklussal a Construct 2 környezetében.



10. ábra, Csillagok égbolt, véletlenszerű csillagokkal.

Üres égbolt létrehozásához hozzunk létre egy üres projektet „csillagos_ejszaka” néven, majd végezzük el az alapbeállításokat.

Állítsuk be a projekt hátterét.

Válasszuk ki a *Layers* panelen az alapértelmezett, *Layer 0* réteget, majd a megjelenő beállítások közül a *Background color* opciót állítsuk sötétkékre – a mintában megadott háttér RGB színkódja: (0;0;51) (F34).

Elsőként egy csillagot hozunk létre, tehát beillesztjük a Csillag objektumot.

Illesszünk be egy új *Sprite* objektumot „csillag” néven és képként nyissuk meg a *csillag.png* fájlt.

Helyezzük el a „csillag” objektumot az elrendezésen kívül (25. ábra).

Sok csillag megjelenítéséhez állítsuk be, hogy minden másodpercben megjelenjen egy csillag a háttéren. A csillagok ismételt megjelenítéséhez egy while ciklust használunk.

Az eseménylapon hozzunk létre egy while ciklus eseményblokkot a *System* objektum *While* elemét kiválasztva. Adjuk hozzá ehhez az eseményblokkhoz a *System* objektum *Every X Seconds* feltételét 1 mp-es intervallumra beállítva (F35).

Csillagok véletlen pozíciójának meghatározásához rendeljük a *System* objektum *Create object* utasítását a blokkhoz a „csillag” objektumra hivatkozva. Az utasítás a következő koordinátákon hozza létre az objektumot (F36):

X: $\text{int}(\text{random}(640))$

Y: $\text{int}(\text{random}(480))$

Csillagok méretezéséhez illesszük be a blokkba a „csillag” objektum *Set scale* utasítását, amely a $\text{random}(1)+0.1$ paraméter szerint átméretezi a „csillag” objektumot (F37).

Csillagok folyamatos megjelenítésének megállításához állítsuk be, hogy a billentyűzet bármelyik gombjának megnyomására a csillagok létrehozása megszakadjon.

Adjuk hozzá a projekthez a *Keyboard* objektumot „billentyuzet” néven.

Az eseménylapon hozzunk létre egy globális szám változót „megallitva” néven, amelynek az alapértelmezett értéke 0 legyen.

A korábban létrehozott eseményblokkhoz adjuk hozzá a *System* objektum *Compare variable* feltételét, amely ellenőrizze, hogy a „megallitva” változó értéke egyenlő-e 0-val (F38). Ezt követően hozzunk létre egy új eseményblokkot, amely a „beillentyuzet” objektum *On any key pressed* feltételét vizsgálja.

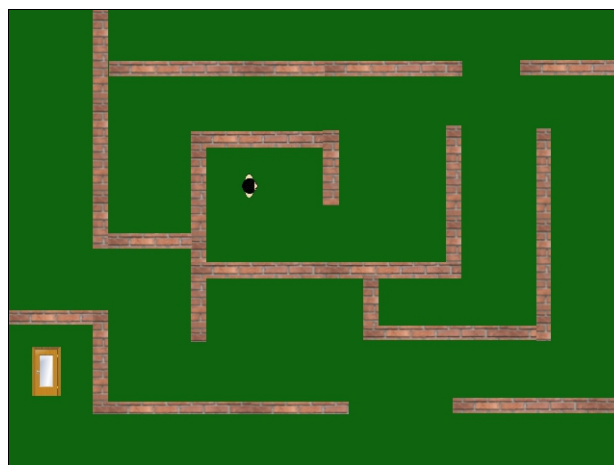
Rendeljük ehhez a blokkhoz a *System* objektum *Set value* utasítását, amely a „megallitva” változó értékét állítsa 1-re (26. ábra) (F39).

Futtassuk le a programot.

A képernyőn másodpercenként megjelenik egy csillag különböző méretben és pozícióban. A csillagok megjelenését a billentyűzet bármelyik gombjának megnyomásával megszakíthatjuk.

Feladat 12: Labirintus

Juttassuk el a fiút a labirintus kijáratánál található ajtóhoz. Építsünk fel egy labirintust egy fal objektumot használva, majd hozzunk létre egy ajtó objektumot. Ezt követően helyezzünk el egy fiú karakter objektumot a labirintus tetszőleges pontján (11. ábra). A fiú mozogjon, automatikusan megtalálva az útvonalat az ajtóhoz. A fiú mozgását úgy indítjuk el, hogy az ajtóra rákattintunk (*labirintus.capx*).



11. ábra, A labirintus egyféle elrendezése, kijárat ajtóval (bal alsó sarok) és a kijáratot kereső fiúval (a labirintus közepén).

Hozzunk létre egy üres projektet „labirintus” néven, majd végezzük el az alapbeállításokat:

Állítsuk be a projekt hátterét.

Válasszuk ki a *Layers* panelen az alapértelmezett, *Layer 0* réteget, majd a megjelenő beállítások közül a *Background color* opciót állítsuk sötétzöld színűre, (12;94;12) RGB színkódra.

Fal objektum beillesztéséhez illesszünk be egy új *9-patch* objektumot „fal” néven és képként nyissuk meg a *fal.png* fájlt (F40).

Állítsuk be, hogy a megnyitott képet az objektum ismétlődően jelenítse meg margók nélkül.

Az objektum tulajdonságainál állítsuk be a *Left*, *Right*, *Top* és *Bottom margin* opciókat 0 értékre, valamint az *Edges* és *Fill* beállításokat *Tile* értékre (27.ábra) (F41).

Állítsuk be, hogy a falak átjárhatatlanok legyenek.

Állítsuk be a „fal” objektumnak a *Solid* viselkedést (F42).

Építsük meg a labirintust.

A labirintus megépítéséhez illesszük be többszörösen a „fal” objektumot (11. ábra). Figyelve arra, hogy az egymás mellett lévő „fal” objektumok vége összeérjen.

Adjunk hozzá a projekthez az ajtó objektumot.

Szűrjünk be egy *Sprite* típusú objektumot, amelyet „ajtó” néven mentsünk el és képként nyissuk meg az *ajto.png* képfájlt.

Pozicionáljuk az „ajtó” objektumot a labirintus szélére (11.ábra).

Illesszük be a projektbe a fiú objektumot.

Szűrjünk be egy *Sprite* típusú objektumot, amelynek a neve „fiú” legyen és képként nyissuk meg az *fiu.png* képfájlt.

Pozicionáljuk a „fiú” objektumot a kiindulási pontba, a labirintus egy tetszőleges pontjára (11.ábra).

Állítsuk be a „fiú” objektumra az útvonalkereső viselkedést és ne engedjük, hogy elhagyja a képernyőt.

Adjuk hozzá a *Pathfinding* viselkedést a „fiú” objektumhoz, majd a viselkedés beállításainál állítsuk a *Cell border* opciót 2-es értékre.

Ezt követően adjuk hozzá az objektumhoz a *Bound to layout* viselkedést is (F43).

Adjuk meg, hogy a „fiu” objektum arra a pozícióra mozogjon, ahova az egér bal egérgombjával kattintunk.

Illesszük be a projektbe a *Mouse* objektumot „eger” néven.

Az eseménylapon hozzunk létre egy új eseményblokkot az „eger” objektum *On click* feltételével. A feltétel az egyszeri bal egérgombbal történő kattintást vizsgálja.

Az eseményblokkhoz rendeljük a „fiu” objektum *Find path* utasítását, amely a következő koordinátákra hivatkozzon (F44):

X: eger.X

Y: eger.Y

Szúrjunk be egy új eseményblokkot a „fiu” objektum *On path found* eseményével. A blokkhoz illesszük be a „fiu” objektum *Move along path* parancsát.

Az *On path found* feltétel akkor teljesül, amikor az útvonalkereső algoritmus végzett az útvonalkereséssel és talált útvonalat (F45).

Kezeljük le, hogy amennyiben egy falra kattintunk, akkor ne keressen útvonalat a program.

Az egér kattintását vizsgáló eseményblokkhoz adjuk hozzá az „eger” objektum *Cursor is over object* feltételét a „fal” objektumra hivatkozva.

A feltételt állítsuk tagadásra (28. ábra) (F46).

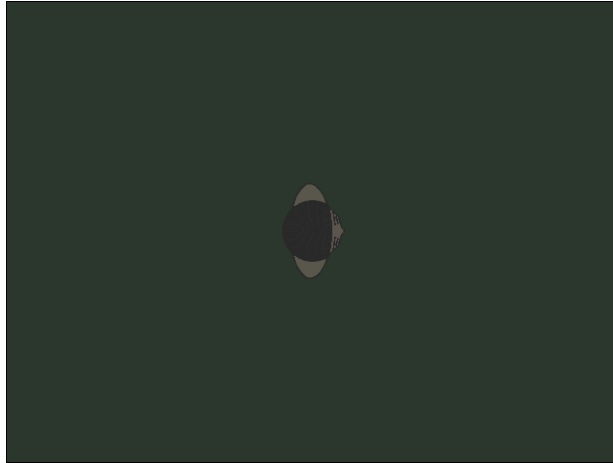
Futtassuk le a programot.

A labirintusban elhelyezett ajtóra kattintva a fiú objektum odamozog.

Feladat 13: Erre csörög a dió

Keressük meg a zene forrását! A feladat a tradicionális „Erre csörög a dió” játék számítógépes megvalósítása. Egy hangforrással – rádió – és egy gyerekkel játszunk. A rádió folyamatosan játssza a zenét, amely aszerint halkuljon vagy erősödjön, hogy a gyerek milyen messze van a rádiótól.

A megvalósításhoz szükségünk van a következő objektumokra: egy egyszínű háttérrel rendelkező rétegen egy gyerek és egy rádió objektumra. A gyerek objektumot mozgathassuk a kurzormozgató nyilak segítségével (12. ábra, *erre_csorog_a_dio.ccapx*).



12. ábra, A gyerek objektum elsötétített megjelenítéssel.

Megvalósítás:

Hozzunk létre egy üres projektet „erre_csorog_a_dio” néven, majd végezzük el az alapbeállításokat:

Az alapbeállításokon túl az előnézeti böngészőprogram a *Chrome* legyen. Az elrendezés margóját igazítsuk az ablak méretéhez, méretét pedig állítsuk 1500×1500 pixel nagyságúra (F47).

Állítsuk be a projekt hátterét.

Válasszuk ki a *Layers* panelen az alapértelmezett, *Layer 0* réteget, majd színezzük át a hátteret sötétzöldre. Ez a *Background color* opció beállításával tehetjük meg. Állítsuk a háttér színét (11;68;13) RGB színkódra.

Gyerek objektum beillesztéséhez illesszünk be egy új *Sprite* objektumot „gyerek” néven és képként nyissuk meg a *gyerek.png* fájlt.

Helyezzük el a gyereket a (400;355) koordinátájú pontra.

Állítsuk be a „gyerek” objektumnak az *8 Direction*, *Bound to layout* és *Scroll To* viselkedéseket.

Adjuk hozzá a projekthez a rádió objektumot.

Illesszük be egy *Sprite* objektumot „radio” néven és forrásképnek nyissuk meg a *radio.png* képfájlt.

Helyezzük az objektumot az elrendezésen tetszőleges pozícióba. Pontos pozicionálásra majd az eseménylapon kerül sor.

Adjunk egy szürke sötétítő réteget az elrendezéshez.

A *Layers* fülön adjunk hozzá egy új réteget a meglévő fölé. A réteg beállításait a következőképp végezzük el (F48):

Background- color: (45;45;45) RGB színekód

Transparent: No

Opacity: 80

Importáljuk be a projektbe a lejátszandó zenét és definiáljuk a hangkezelést.

Adjuk hozzá a projekthez az *Audio* objektumot „hangok” néven.

Importáljuk be a Music mappába a zene.flac audiofájlt.

A „hangok” objektum beállításainál a *Distance model* opciót állítsuk *Linear* értékre, a *Reference distance* opciót *100* értékre és a *Maximum distance* opciót *1300* értékre (29. ábra). Ezzel a beállítással az elrendezésünk méretéhez igazítjuk a hang halkulását, annak a szélére mozogva pedig már nem is lesz hallható.

A *Distance model* beállítás segítségével adhatjuk meg, hogy a hangkezelés milyen algoritmust használjon a hangforrástól való távolság kiszámítására. Esetünkben a legegyszerűbb, lineáris távolságszámításra van szükség (F49).

Kérdezzük le az elrendezés méretét és pozicionáljuk a rádiót a közepére.

Az eseménylapon adjunk hozzá egy új eseményblokkot a *System* objektum *On start of layout* feltételével.

Rendeljük a „radio” objektum *Set position* utasítását a következő argumentumokkal (F50):

X: LayoutWidth/2

Y: LayoutHeight/2

Állítsuk be a zene automatikus lejátszását és forrását.

Az eseményblokkhoz rendeljük hozzá a „hangok” objektum *Play at object* utasítását. A lejátszandó hangfájl a „zene” legyen, a *Loop* beállítás *Looping* értéket vegyen fel, az objektum pedig a „radio” objektum legyen (F51).

Adjuk hozzá az eseményblokkhoz a „hangok” objektum *Set listener object* utasítását, amely a „gyerek” objektumra hivatkozzon (F52).

Futtassuk le a programot.

A gyerek objektumot a kurzormozgató nyilak segítségével tudjuk irányítani. A rádió folyamatosan zenét játszik le, és a gyerek rádiótól való távolsága határozza meg, hogy milyen hangerővel halljuk a zenét.

Feladat 14: Valahol a sötétben

Kezeljük le, hogy az „erre_csorog_a_dio” játékban a program futtatásakor a „gyerek” objektum véletlenszerű pozícióban jelenjen meg, de ne kerülhessen rá a rádióra.

Helyezzük el véletlenszerű pozícióba a „gyerek” objektumot a program futtatásakor.

A szerkesztőfelületre visszatérve illesszünk be az eseménylapra az elrendezés betöltődését vizsgáló eseményblokkhoz a „gyerek” objektum Set position utasítását a következő argumentumokkal (F53):

X: `int(random(1466)+34)`

Y: `int(random(1448)+51)`

Kezeljük le, ha a véletlenül az elrendezés betöltődésekor a „gyerek” objektum rákerülne a „radio” objektumra.

Illesszünk be egy új eseményblokkot a System objektum While eseményét használva.

Rendeljük ehhez az eseményblokkhoz a „gyerek” objektum Is overlapping another object feltételét a „radio” objektumra hivatkozva (F54).

Amennyiben átfedésben van a két objektum, helyezzük új pozícióra a gyereket.

Illesszük be az eseményblokkhoz a „gyerek” objektum Set position utasítását a következő argumentumokkal (30. ábra):

X: `int(random(1466)+34)`

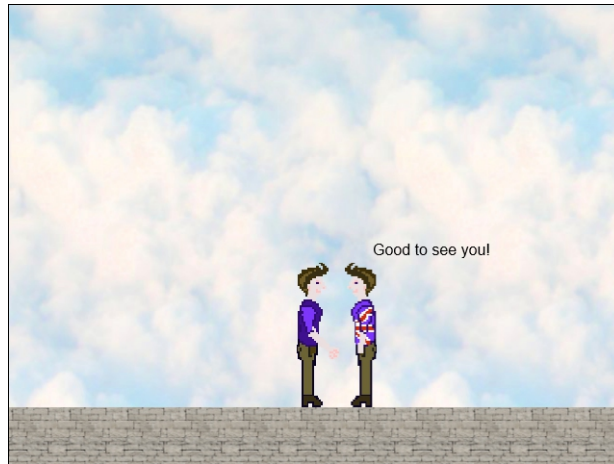
Y: `int(random(1448)+51)`

Teszteljük le a változásokat egy újabb előnézet indításával, és láthatjuk, hogy a program minden egyes futtatásánál a „gyerek” objektum más pozícióba kerül.

Feladat 15: Köszönés

Sétáljunk végig egy turistával az úton és köszönjünk az embereknek, akikkel találkozunk. Egy vízszintes úton jobbra haladva találkozunk sorban a következő nyelven beszélő emberekkel: magyar, angol és francia. Ha a közelükbe megyünk, a Szóköz billentyű megnyomásával köszöntenek bennünket a nyelvükön egy véletlenszerűen kiválasztott módon és a fejük fölött megjelenik a köszönés szöveges változata is.

A megvalósításhoz szükségünk van a következő objektumokra: egy felhős háttérrel rendelkező rétegen egy talaj, egy turista, három ember és egy szöveg objektumra. A turistát mozgathassuk a kurzormozgató nyilak segítségével (13. ábra, *koszones.capx*).



13. ábra, A turista találkozik az angol nyelvű emberrel, aki angolul köszön neki.

Megvalósítás:

Hozzunk létre egy üres projektet „koszones” néven, majd végezzük el az alapbeállításokat:

Állítsuk be a projekt hátterét.

Adjunk hozzá a projekthez egy új *Tiled Background* objektumot „hatter” néven és forrásképnek nyissuk meg a *hatter.png* fájlt.

Az objektumot pozicionáljuk a (0;0) koordinátákra és méretét állítsuk 2000×480 pixel nagyságúra.

Talaj objektum beillesztéséhez illesszünk be egy új *9-patch* objektumot „talaj” néven és képként nyissuk meg a *talaj.png* fájlt.

Végezzük el rajta a forráskép ismétléséhez és a margók kihagyásához szükséges beállításokat:

Left margin: 0

Right margin: 0

Top margin: 0

Bottom margin: 0

Edges: Tile

Fill: Tile

Helyezzük el a „talaj” objektumot a (0;420) koordinátájú pontra, méretét pedig állítsuk 2000×60 pixel nagyságúra. Végül definiáljuk, hogy az emberek meg tudjanak állni a talajon.

Rendeljük a „talaj” objektumhoz a *Solid* viselkedést.

Illesszük be a projektbe a három különböző nyelvű embert.

Adjunk hozzá a projekthez három új Sprite objektumot „ember_magyar”, „ember_angol” és „ember_francia” néven.

Az „ember_magyar” objektum forrásképének nyissuk meg az *ember_magyar.png* fájlt, az „ember_angol” objektumnak az *ember_angol.png* képet, az „ember_francia” objektumnak pedig az *ember_francia.png* állományt.

Helyezzük el az ember objektumokat a következő koordinátákra:

„ember_magyar”: (897;320)

„ember_angol”: (1288;320)

„ember_francia”: (1635;320)

Ezt követően állítsuk be, hogy az emberek megálljanak a talajon.

Mindhárom ember objektumnak állítsunk be a *Platform* viselkedést és a beállításai közül a *Default controls* opciót definiáljuk „No” értékre.

A *Platform* viselkedés használata esetünkben azt a célt szolgálja, hogy az ember objektumok a talaj objektummal érintkezve (amely rendelkezik a *Solid* viselkedéssel) megálljanak rajta (F55).

A turista objektum hozzáadása

Illesszünk be egy új *Sprite* objektumot „turista” néven és forrásképnek nyissuk meg a *turista.png* fájlt.

Helyezzük el az objektumot a (29;338) koordinátákra.

Határozzuk meg a „turista” objektum következő viselkedéseit: mozgathassuk a kurzormozgató nyilak segítségével, kövesse a kamera és ne hagyhassa el a képernyőt.

Rendeljük a „turista” objektumhoz a *Platform*, *Scroll To* és *Bound to layout* viselkedéseket.

Készítsük elő a köszönések szöveges kiírását.

Adjunk hozzá a projekthez egy új *Text* objektumot „szoveg” néven, majd helyezzük el az elrendezésen kívülre.

Készítsük elő a köszönések hangjainak lejátszását.

Adjuk hozzá a projekthez az *Audio* objektumot „hangok” néven.

Importáljuk be a következő hangfájlokat a projektbe:

koszones_angol_1.flac

koszones_angol_2.flac

koszones_angol_3.flac

koszones_francia_1.flac

koszones_francia_2.flac

koszones_francia_3.flac

koszones_magyar_1.flac

koszones_magyar_2.flac

koszones_magyar_3.flac

Definiáljuk a köszönések kezeléséhez szükséges változókat.

Az eseménylapon illesszünk be egy szám alapú globális változót „melyik_koszones” néven, 0 alapértelmezett értékkel, illetve egy globális szöveg alapú változót „nyelv” néven, üres string alapértelmezett értékkel.

Valósítsuk meg a köszönést mindhárom nyelven függvény segítségével.

Adjuk hozzá a projekthez a *Function* objektumot „fuggveny” néven (F56).

Hozzuk létre a „koszones” függvényt.

Az eseménylapon hozzunk létre egy új eseményblokkot a „fuggveny” objektum *On function* feltételével. A feltétel a „koszones” függvény futását vizsgálja (F57).

Kezeljük le, hogy egyszerre csak egy szöveg objektum jelenjen meg és válasszuk ki véletlenszerűen a köszönést.

Rendeljük az eseményblokkhoz a „szoveg” objektum *Destroy* utasítását.

A *System* objektum *Set value* utasításával adjunk a „melyik_koszones” változónak értékül egy véletlenszerűen generált egész számot a [0;2] intervallumban. Ezt az *int(random(3))* függvényekkel tehetjük meg.

Kezeljük le a magyar nyelvű köszönéseket.

A „fuggveny” objektum „koszones” függvényét definiáló eseményblokkhoz illesszünk be egy alesemény blokkot, amely a *System* objektum *Compare variable* utasításával megvizsgálja, hogy a „nyelv” változóban tárolt érték a „magyar” karaktersorozat-e.

Jelenítsük meg a szöveg objektumot a magyar ember feje felett.

Rendeljük az eseményblokkhoz a *System* objektum *Create object* parancsát, amely hozza létre a „szoveg” objektumot a következő koordinátákon:

X: ember_magyar.X

Y: ember_magyar.Y-70

Írjuk ki a kiválasztott köszönés szövegét és játsszuk le a hozzá kapcsolódó hangot.

A „nyelv” változó „magyar” értékét vizsgáló eseményblokkhoz adjunk hozzá három aleseményt. Az események rendre a *System* objektum *Compare variable* utasításával a „melyik_koszones” változó értékét teszteljék 0; 1; 2 értékekre (32. ábra).

Mindhárom blokkhoz rendeljük a „szoveg” objektum *Set text* utasítását, illetve a „hangok” objektum „Play” utasítását.

A „melyik_koszones” változó értékét 0-ra tesztelő blokk a "Jó napot kívánok!" szöveget írassa ki a „szoveg” objektummal és a „koszones_magyar_1” hangot játssza le a „hangok” objektummal.

A „melyik_koszones” változó értékét 1-re tesztelő blokk az "Isten hozta!" szöveget írassa ki a „szoveg” objektummal és a „koszones_magyar_2” hangot játssza le a „hangok” objektummal.

A „melyik_koszones” változó értékét 2-re tesztelő blokk az "Üdvözlöm!" szöveget írassa ki a „szoveg” objektummal és a „koszones_magyar_3” hangot játssza le a „hangok” objektummal.

Kezeljük le az angol és francia nyelvű köszönéseket. Hasonlóan járjunk el, mint a magyar köszönés esetén, csak az objektumok változnak meg, amelyekhez ismét be kell állítani a tulajdonságokat.

Definiáljuk a „koszones” függvény meghívását, ha a turista a különböző emberek közelében áll és a felhasználó megnyomja a Szóköz billentyűt.

Adjuk hozzá a projekthez a billentyűzet bemenet használatát.

Illesszük be a *Keyboard* objektumot „billentyuzet” néven.

Illesszünk be az eseménylapon egy, az eddigiektől független új eseményblokkot a „billentyuzet” objektum *On key pressed* feltételével, amely a Szóköz billentyű lenyomását vizsgálja.

Vizsgáljuk meg, hogy a turista a közelében áll-e az adott nyelvű embernek, vagy átfedésben vannak-e.

A Szóköz billentyű lenyomását vizsgáló blokkhoz illesszünk be három alesemény blokkot. Mindhárom blokk a „turista” objektum *Is overlapping another object* feltételét vizsgálja rendre az „ember_magyar”, „ember_angol” és „ember_francia” objektumokra.

Mindhárom eseményblokkot állítsuk *vagy* logikájú vizsgálatra. Ebben az esetben az eseményblokk bármely benne foglalt feltétel teljesülésekor meghívásra kerül.

Ahhoz, hogy az eseményblokkokba elhelyezett feltételek közötti logikai összefüggést vagy kapcsolatra állítsuk, az eseményblokk helyi menüjéből válasszuk ki a *Make 'Or' block* opciót.

Adjuk hozzá mindhárom blokkhoz a „turista” objektum *Is overlapping at offset* feltételét rendre az „ember_magyar”, „ember_angol” és „ember_francia” objektumokra hivatkozva, *Offset X: 50* argumentummal (31.ábra).

Az *Is overlapping at offset* feltétel azt vizsgálja, hogy a két objektum a beállított koordináta értékek által megadott eltolással számolva átfedésben vannak-e (F58).

Adjuk meg, hogy a turista milyen nyelvű emberrel kerül kapcsolatba.

Az „ember_magyar”, „ember_angol”, „ember_francia” objektumokkal való átfedéseket vizsgáló blokkhoz rendeljük a *System* objektum *Set value* utasítását, amely a „nyelv” változónak adja a „magyar”, „angol”, illetve „francia” értéket.

Hívjuk meg a „koszores” függvényt.

Mindhárom átfedést vizsgáló eseményblokkhoz adjuk hozzá a „fuggveny” objektum *Call function* utasítását, amely a „koszores” függvényt hívja meg (F59).

Futtassuk le a programot.

A turista objektumot a kurzormozgató nyilak segítségével tudjuk mozgatni. Amennyiben közel mozgatjuk egy emberhez és megnyomjuk a Szóköz billentyűt, az adott ember a saját nyelvén egy véletlenszerűen kiválasztott módon köszönti a turistát és a köszönés szövege megjelenik a feje felett.

Egyéni projektmunkák

Fontos hogy a tanulók kreativitásukkal önállóan is programot alkossanak. Ezért, amikor már kellőképpen elsajátították a fejlesztői környezet használatát, otthoni feladatra is kell kötelezni őket akár egyénileg, akár csoportosan (a feladat nehézségétől és összetettségétől függően). Ez a feladat beadandó programok elkészítését jelentené a kiindulási irányok meghatározásával.

Le kell fektetni néhány alapkövetelményt, hogy mit tartalmazzon a projekt és milyen funkcionalitást implementáljanak bele a diákok. Azonban ellentétben az órai feladatokkal (melyek ettől a modultól teljesen függetlenül érinthetők) az otthoni feladat lényege a szabadság és kiterjesztett kreativitás kell, hogy legyen. Ezért elsődlegesen nem tudásfelmérési szempontból kell elvégeztetni a tanulókkal a feladatot, hanem, hogy önállóságuk és egyéni

problémamegoldásuk algoritmikus környezetben is fejlődjön. A tanárnak az önálló munkák elkészítése alatt is rendelkezésre kell állnia, hogy a diákok felmerülő kérdéseit megválaszolhassa, de vigyáznia kell, hogy ne tegyen konkrét javaslatokat a projektek elkészülését illetően (a befolyásolás elkerülése végett).

Példaprojekt: Készítsenek egy játékprogramot, amelyben a játékos irányíthat egy platform mozgással ellátott objektumot és pontokat szerezhet pénz elemek gyűjtésével.

A tanulók szabadon készíthetnek bármilyen alkalmazást, amely a fenti feltételeknek eleget tesz. Amennyiben úgy látják jónak, beillesztenek egy háttér elemet, vagy hangokat is lejátszhatnak az egyes cselekvések mellé. Az ez és ehhez hasonló kiegészítő funkciók kitalálása az egyénekre vannak bízva.

Összegzés

A 2013-as kerettanterv szerint az általános- és középiskolás diákok rendkívül szűk keretek között vehetnek részt informatikai képzésben Magyarországon. Ennek a képzésnek része az algoritmikus gondolkodás fejlesztése, a programozás tanítása, de az alacsony óraszámok miatt rendkívül alacsony hatékonysággal. Az algoritmikus gondolkodás kialakítására és fejlesztésére általánosan négy módszert alkalmaznak:

- A Logo szoftvereket (Comenius és Imagine Logo), amelyek a Logo programozási nyelvre épülve oktatási céllal lettek kifejlesztve és kezdő szinten alkalmasak a programozási folyamatok bemutatására. Elsősorban általános iskolákban alkalmazzák, melyből a diákok gyorsan tanulnak és látványosan. Egyszerűsége miatt idősebb korosztályok számára nem ideális. A Logo másik hátránya, hogy szinte kizárólagosan iskolai felhasználásra készült, az algoritmikus készség és kreativitás fejlesztésére. Iskolán kívül, a valós programozásban nem játszik szerepet, és mint ilyen a diákok nem tekintik olyan programozási nyelvnek, amit érdemes megtanulni.
- A Lego robotokat, amelyek megépítésük után egy egyszerű felületen, logikai blokkok segítségével vezérelhetőek. Ez a módszer is főként általános iskolákban terjedt el a korosztálybeli érdeklődés miatt.
- Hagyományos programozási nyelveket, amelyek már kifejtettebb logikai készségeket igényelnek. Ezért főként középiskolai tananyagban találkozhatunk velük. Sajnos azonban ez helyenként idejétmúlt nyelv oktatását jelenti (K3, 2000) (K4, 2002), amelyet a diák szinte biztos, hogy nem fog alkalmazni a későbbiekben. Emellett a programozási nyelveket sok diák nem érti, mivel bonyolultnak találja és elsődleges célja nem annak élvezete, vagy lehetőségeinek megtanulása, kiaknázása, hanem a tantárgy teljesítése.
- Mobil programozást, amely során nem hagyományos PC-re, hanem a trendeknek megfelelően, mobil eszközökre készítenek alkalmazásokat a diákok és ezen keresztül sajátítják el az algoritmizálást. Ennek hátránya, hogy a legtöbb esetben hagyományos programozási nyelveket használnak, amely továbbra sem a legkönnyebben befogadható a diákok számára. Előnye azonban, hogy ezek az alkalmazások már látványosnak készülnek, így a tanulók könnyebben meg tudják ragadni a munkájuk

eredményét. Megjegyezném, hogy a Construct 2 segítségével is lehetőségünk van mobil alkalmazásokat fejleszteni a napjainkban elterjedt mobil operációs rendszerekre is.

További lehetőség még az algoritmikus gondolkodás fejlesztésére a táblázatkezelői programok funkcionális nyelve, de ezzel a lehetőséggel alig élnek az informatikatanárok.

- Táblázatkezelő programok szolgálhatnának bevezető nyelvként a hagyományos programozási nyelvekhez. Ennek a módszernek nagy előnye az időtakarékoság és a programozási készségek kialakításával párhuzamosan járó táblázatkezelés elmélyülése.

A vizuális programozás a hagyományos programozási nyelvek kategóriáját egészítené ki, a Construct 2 szoftver oktatásával. Egy innovatív és új módszerről van szó, amely a programozást egy alternatív oldalról közelíti meg. Olyan szerkesztőfelületet alkalmaz, amelyben „fogd és vidd” módszerrel összeállíthatunk „bármilyen” 2D-s játékszoftvert forráskód hagyományos gépelése nélkül. Ezért oktatásbeli hatékonysága is magas, hiszen a tanulóknak nem kell bonyolult kódelemeket elsajátítaniuk ahhoz, hogy továbbfejlődjön az algoritmikus készségük, valamint kedvezőbb számukra a tény, hogy játékszoftvert fejlesztenek a nem túl népszerű iskolához között problémák megoldása helyett. Emellett nem elhanyagolható a HTML5 technológia modernitása sem, hiszen egy dinamikusan fejlődő, nyílt és cross-platform rendszerről van szó, valamint, hogy a Construct 2 ingyenesen letölthető és használható az oktatásban.

A Scirra cég programjával szerzett tudást a későbbiekben is felhasználhatják a diákok. Akár az oktatott fejlesztőkörnyezetet használva is piacképes szoftverek készíthetők több platformra is, de a szerzett algoritmikus készség mindenképp kamatoztatható (különösen napjainkban ahol szinte minden ember érintkezik valamilyen digitális eszközzel).

A Construct 2 környezethez kidolgozott oktatási terv fejezetekre bontva vezeti be a tanulókat a szoftver használatába és ezzel együtt a komplexebb projektek elkészítésébe is. Kialakításakor külön figyelmet fordítottam arra, hogy a tanórák szűk keretein belül minél több területet és programot lefedjen és látványos maradjon. A modulokon végighaladva a csoport először megismerkedik a szoftver használatával, majd már az első feladatnál grafikai felületet épít fel és alapvető mozgást valósít meg minimális beállításokkal. Ezt követi a mélyebb

elmerülés a program lehetőségeiben és folyamatosan haladva egyre komplexebb projekteket valósítanak meg. Bár a csoport részéről elvárt az önálló munka, mégis ez gyakorlatban nem mindig kivitelezhető, ezért a tanár szolgál vezetőként a diákok számára.

A kezdeti bevezetést követik a már vizuális programozást is igénylő feladatok. Az egyszerűség kedvéért itt még csupán csak néhány soros kódot készítünk és az egyes utasítások vagy feltételek nem kerülnek kombinálásra. Emellett fokozatosan érintik a program újabb, kevésbé szembeötlő funkcióit (például az animációkezelés vagy grafikai szerkesztés).

Az ezt követő modul már alapértelmezettnek veszi az eseménylapok használatát és igyekszik komplexebb eseményblokkokat vagy utasításcsomagokat kialakítani további új lehetőségek bemutatásával (például: a *Drag & Drop* vagy *Physics* viselkedés).

Az elmélyülés és közeledés a hagyományos programozási nyelvek utasításaihoz folyamatos. A következő részben a vizuális programozás további elmélyítése kapja a hangsúlyt. A tanulók kihasználják annak lehetőségeit: aleseményeket hoznak létre, illetve megismerkednek a különböző kifejezések használatával (például véletlenszám generálás, vagy koordináta lekérdezés).

Ezek mellett figyelembe véve a játékfejlesztés kreatív oldalát is, a diákok egy önálló (vagy csoportos) beadandó programot is elkészítenek. Minimális irányadás mellett a tanulók fantáziájára van bízva, hogy hogyan építik fel az órákon (és egyénileg) szerzett tudásukat felhasználva.

A Construct 2 tanulása közben elsajátított tudást a későbbiekben többszörösen is kamatoztathatják a diákok. A hasonló kategóriában elterjedt szoftverek száma folyamatosan nő és ezzel párhuzamosan csökken az úr a köztük és hagyományos programozási nyelvvel készített alkalmazások között. Így nemcsak az algoritmikus készség fejlődéséről beszélhetünk, hanem egy olyan módszerről is, amelyet felhasználva (akár a Construct 2 környezetnél maradván) piacképes szoftverek fejleszthetőek.

A Construct 2 oktatási tervnek nem célja, hogy kiszorítsa a hagyományos programozási nyelveket az oktatásból. Ehelyett annak újragondolását szorgalmazza. A programozási nyelvek előnyei vitathatatlanok, azonban nem megfelelőek és nem is szükségesek minden diák számára. Az alapvető informatikai oktatás egyik célja, hogy az algoritmikus készség kialakuljon, amelyre tökéletesen alkalmas a Construct 2 szoftver egy egyszerű és látványos módon tanítva azt. A programozási nyelvek ellenben az informatika iránt mélyebben érdeklődőknek, illetve a későbbiekben ilyen irányba elinduló tanulók számára hasznosak, így

azoknak csupán fakultációs lebontásban való tanítását javasolnám. Az otthonában nagyon kevés tanuló próbálja ki, amit az iskolában tanult, nem írnak parancssori ablakban futó programokat. Ellenben képzeljük el, hogy ez a szám mennyivel nőne ha játékokat fejleszthetne nyelvi akadályok nélkül. A játékszoftverek szerves részei különböző generációk életének és messze túlmutatnak a hagyományos „játék” fogalmán. Egy többdimenziós médiumról beszélhetünk, amely jelenleg nagyobb piacot tudhat magáénak, mint a filmipar. Ha a látványosság, egyszerűség és a siker élménye párosítható a fejlesztési folyamattal, úgy a tanulók nagyobb eséllyel kezdenek majd a későbbiekben is érdeklődni további programozási nyelvek iránt.

Hivatkozások, irodalomjegyzék

- K1 Fenyő Zoltán, Fenyősné Kircsi Amália (2009) Informatika, hatodikos, Pedellus Tankönyvkiadó, 62–91
- K2 Fenyő Zoltán, Fenyősné Kircsi Amália (2009) Informatika, hetedikes, Pedellus Tankönyvkiadó, 94–113
- K3 Katona Endre (2000) Bevezetés az informatikába, Panem Könyvkiadó, 155–159
- K4 Rozgonyi-Borus Ferenc, Kokas Károly (2002) Informatika középiskolásoknak, Mozaik Kiadó, 168–172
- W1 YoYo Games (2007) Welcome to YoYo Games <http://www.yoyogames.com/> Letöltés dátuma: 2014. április 14.
- W2 Stencyl (2011) Create iOS, Android and Flash Games with Stencyl <http://stencyl.com/> Letöltés dátuma: 2014. április 14.
- W3 GameSalad (2009) Game Design Engine, Make Games for iPhone & Android - GameSalad <http://gamesalad.com/> Letöltés dátuma: 2014. április 14.
- W4 Game Editor (2011) Game Editor <http://game-editor.com/> Letöltés dátuma: 2014. április 20.
- W5 Scirra (2011) About Us - Scirra.com <https://www.scirra.com/about> Letöltés dátuma: 2014. április 20.
- W6 Ashley (2012) Construct Classic r2 released - Scirra Forum <http://www.scirra.com/forum/topic48884.html> Letöltés dátuma: 2012. február 5. Letöltés dátuma: 2014. április 8.
- W7 Ashley (2013) Scirra has retired Construct Classic - Scirra Forum <http://www.scirra.com/forum/topic66664.html> Letöltés dátuma: 2013. április 20. Letöltés dátuma: 2014. április 21.
- W8 Scirra (2013) Official Construct 2 Manual - Construct 2 Manual <https://www.scirra.com/manual/1/construct-2> Letöltés dátuma: 2014. április 10.

- W9 Scirra (2013) Top game making tutorials - Scirra.com <https://www.scirra.com/tutorials/>
Letöltés dátuma: 2014 április 12.
- W10 Ludodesign (2012) What a Game Designer Needs to Know - Scirra.com
<https://www.scirra.com/tutorials/249/what-a-game-designer-needs-to-know> Feltöltés dátuma: 2012. január 20. Letöltés dátuma: 2014. április 15.
- W11 Ludodesign (2012) The Five and a Half Filters of Game Creation - Scirra.com
<https://www.scirra.com/tutorials/341/the-five-and-a-half-filters-of-game-creation>
Feltöltés dátuma: 2012. július 1. Letöltés dátuma: 2014. április 15.
- W12 Ludodesign (2011) 6 Steps to Play with Players' Minds - Scirra.com
<https://www.scirra.com/tutorials/188/6-steps-to-play-with-players-mind> Feltöltés dátuma: 2011 december 17. Letöltés dátuma: 2014. november 07.
- W13 Joshua "Loaf" L. (2013) 5 Ways To Save Your Game From An Early Grave | Game Maker Blog <http://gamemakerblog.com/2013/03/30/5-ways-to-save-your-game-from-an-early-grave/> Feltöltés dátuma: 2013. március 30. Letöltés dátuma: 2014. április 15.
- W14 HasGraphics (2013) Free Graphics for Flash and Indie Games <http://hasgraphics.com/>
Letöltés dátuma: 2014. április 15.
- W15 WP Clipart (2013) WPClipart Home Page <http://www.wpclipart.com/index.html> Letöltés dátuma: 2014. november 6.
- W16 OpenGameArt (2013) OpenGameArt.org <http://opengameart.org/> Letöltés dátuma: 2014. április 15.
- W17 Bfxr (2011) Bfxr. Make sound effects for your games <http://www.bfxr.net/> Letöltés dátuma: 2014. április 15.
- W18 Maciek (2008) cgMusic - Can computers create music? | Casual Games Harmony
<http://codeminion.com/blogs/maciek/2008/05/cgmusic-computers-create-music/> Feltöltés dátuma: 2008. május. 19. Letöltés dátuma: 2014. április 15.
- W19 Freesound (2013) Freesound.org - Sounds browse <http://www.freesound.org/browse/>
Letöltés dátuma: 2014. április 15.

- W20 Free-Loops (2007) Free-Loops.com | Free Drum Loops Wav MP3 Aif and Midi Sound Loops <http://free-loops.com/> Letöltés dátuma: 2014. április 15.
- W21 Opsound (2013) opsound: free love, free music <http://opsound.org/> Letöltés dátuma: 2014. április 15.
- W22 Scirra (2013) Scirra Forum <https://www.scirra.com/forum/> Letöltés dátuma: 2014. április 18.
- W23 Scirra (2013) Scirra Forum <https://www.scirra.com/forum/default.asp?C=12&SID=28de4721-a5az-27a597f7-41393-5872685185&title=construct-2-general> Letöltés dátuma: 2013. április 18.
- W24 Scirra (2011) Create Games with Construct 2 - Scirra.com <https://www.scirra.com/> Letöltés dátuma: 2014. április 18.
- W25 Clay.io (2013) HTML5 Game Engines - Find Which is Right For You <http://html5gameengine.com/> Letöltés dátuma: 2014. április 20.
- W26 Scirra (2013) Plugins for Construct 2 - Scirra Forum https://www.scirra.com/forum/plugins-for-construct-2_forum47.html Letöltés dátuma: 2014. április 20.
- W27 Scirra (2013) Bugs - Scirra Forum https://www.scirra.com/forum/bugs_forum43.html Letöltés dátuma: 2014. április 20.
- W28 Valve (2013) Construct 2 a Steamen <http://store.steampowered.com/app/227240/> Letöltés dátuma: 2013. január 25. Letöltés dátuma: 2014. április 20.
- W29 Scirra (2013) Addicting Games - Scirra Arcade <http://www.scirra.com/arcade> Letöltés dátuma: 2014. április 21.
- W30 Dropbox (2014) Dropbox <https://www.dropbox.com/> Letöltés dátuma: 2014. április 21.
- W31 Google (2014) Google Drive <https://drive.google.com/> Letöltés dátuma: 2014. április 21.
- W32 Microsoft (2014) Microsoft OneDrive <https://onedrive.live.com/> Letöltés dátuma: 2014. november 7.

- W33 Scirra (2013) Expressions - Construct 2 Manual
<https://www.scirra.com/manual/78/expressions> Letöltés dátuma: 2014. április 21.
- W34 Szlávi Péter, Zsakó László (2009) Az informatika oktatás téveszméi
<http://lenarda.hu/files/Az-informatikaoktatas-teveszmei.pdf> Letöltés dátuma: 2013. április 25.
- W35 Scirra (2013) Remember not to waste your memory - Scirra.com
<https://www.scirra.com/blog> Letöltés dátuma: 2014. április 28.
- W36 Scirra (2013) Maunal <https://www.scirra.com/manual.zip> Letöltés dátuma: 2014. november 8.
- W37 Tom (2013) Improved Tutorial Pages - Scirra.com
<https://www.scirra.com/blog/109/improved-tutorial-pages> Feltöltés dátuma: 2013. április 4. Letöltés dátuma: 2014. április 6.
- W38 YoYoGames (2013) YoYo Games Ltd. : Learn
<http://help.yoyogames.com/categories/20139196-Learn> Letöltés dátuma: 2014. április 27.
- W39 Stencyl (2013) Stencylpedia <http://www.stencyl.com/help/> Letöltés dátuma: 2014. április 27.
- W40 Oktatáskutató és Fejlesztő Intézet (2012) Kerettanterv - Oktatáskutató és Fejlesztő Intézet, OFI <http://kerettanterv.ofi.hu/> Letöltés dátuma: 2014. április 27.
- W41 Scirra (2013) Construct 2 Javascript SDK documentation - Construct 2 Manual
<https://www.scirra.com/manual/15/sdk> Letöltés dátuma: 2014. április. 28.
- W42 Scirra (2014) Make online multiplayer games with Construct 2 - Scirra.com
<https://www.scirra.com/blog/142/make-online-multiplayer-games-with-construct-2>
Feltöltés dátuma: 2014. április 24. Letöltés dátuma: 2014. november 05.
- W43 Scirra (2014) The Debugger - Construct 2 Manual
<https://www.scirra.com/manual/169/debugger> Letöltés dátuma: 2014. november 05.
- W44 Scirra (2014) Construct 2 r187 (Beta) Download - Scirra.com
<https://www.scirra.com/construct2/releases/r187> Letöltés dátuma: 2014. november 09.

- W45 Michael Gove (2012) Michael Gove speech at the BETT Show 2012 - Speeches – GOV.UK <https://www.gov.uk/government/speeches/michael-gove-speech-at-the-bett-show-2012> Feltöltés dátuma: 2012. január 13. Letöltés dátuma: 2014. november 13.
- W46 Michael Gove (2014) Michael Gove speaks about computing and education technology - Speeches – GOV.UK <https://www.gov.uk/government/speeches/michael-gove-speaks-about-computing-and-education-technology> Feltöltés dátuma: 2014. január 22. Letöltés dátuma: 2014. november 13.
- W47 Oktatókutató és Fejlesztő Intézet (2013) Kerettanterv - Oktatókutató és Fejlesztő Intézet, OFI http://kerettanterv.ofi.hu/08_melleklet_szakiskola/index_szi.html Letöltés dátuma: 2014. november 14.

Saját munka bemutatása

Munkám során felmértem a napjainkban rendelkezésre álló legnépszerűbb vizuális fejlesztőkörnyezeteket és ezek közül kiválasztottam a három legjobbnak ítélt nyelvet. Ezt követően további vizsgálatok és gyakorlati munka során kiválasztottam a legmegfelelőbb és legdinamikusabban fejlődő környezetet, amelynek még részletesebb elemzésébe kezdtem.

Kutatásom másik ágaként az általános- és középiskolai informatika oktatásban végeztem felmérést (informatika tanárokat megkérdezve), amelyben azt vizsgáltam, hogy jelenleg milyen programozás oktatási módszerek vannak érvényben és, hogy ezekkel milyen hatásokkal képesek a tanulók az algoritmikus gondolkozást elsajátítani.

A választott nyelv mély megismerése után kidolgoztam egy oktatási tantervet, amely alkalmas lehet a középiskolai programozásoktatás leváltására úgy, hogy a tanulók nagyobb hatásokkal sajátíthassák el az algoritmikus készségeket. A tantervhez példafeladatokat készítettem, amelynek forrásait és kész projektfájljait egyaránt csatolom a dolgozathoz.

Köszönetnyilvánítás

Szeretnék köszönetet mondani elsősorban témavezetőmnek és tanáromnak, Dr. Csernoch Máriának, aki felügyelte és segítette kutatásomat és munkámat a tanterv és a dolgozat kialakítása során.

Köszönöm a megkérdezett informatika tanárok segítségét, akik megosztották velem oktatásbeli tapasztalataikat és véleményüket, hogy tisztább képet kaphassak az informatika oktatásról.

Köszönöm még a Scirra felhasználói közösség segítségét, ahol mindig választ kaphattam a Construct 2 szoftverrel kapcsolatos kérdéseimre és természetesen a Scirra cégnek, hogy létrehozták ezt a vizuális fejlesztőkörnyezetet és támogatják az oktatásbeli felhasználását.