**ZÁRÓDOLGOZAT**

**SimplePlatformer**

készítette:

**Balatoni Attila**

**2/14.S**

TODO: Iskola cím stb.

Nyilatkozat

Alulírott, Balatoni Attila kijelentem, hogy ez a záródolgozat saját tudásom, önálló munkám terméke.

Kijelentem, hogy a záródolgozat beköttetett és elektronikus formában leadott példányai mind formátumban, mind tartalomban egyezik, eltérést nem tartalmaznak.

Dátum:

Hallgató aláírása

Bevezetés

A szakdolgozatom témájának egy játékot választottam. A program egy kétdimenziós platformer. A játék célja, hogy egyszerű mozgással végig jusson a pályákon a játékos, melyet a pálya kialakítása és az ellenfelek jelenléte nehezít. A játékosnak eközben figyelnie kell az életére, melyet a pályákon elhelyezett életekből nyerhet vissza. Pontokat hasonlóképp szerezhet, kellő ügyességgel minden pénzérmét összeszedhet. A játék futtatása nem igényel erős számítógépet, sem internetkapcsolatot. A szoftver a Unity nevű motorban készült, mely C# kóddal kiegészítve működteti, alkotja a kész játékot. Adatbázis kezeléshez Json-t használtam, mely lokális fájlba mentett adatbázisként szolgál.

Témaválasztás indoklása

Egész életemben szerettem videójátékokkal játszani, ez az elsőszámú és legfontosabb hobbim, ami életem jelentős részét kitölti a jelenben és ez a jövőben semmit sem fog változni. Szoftverfejlesztőnek is azért jelentkeztem, mert hosszútávú célom, hogy a játékfejlesztői iparban el tudjak helyezkedni, mint játékfejlesztő. A játékok, ha csak egy kis időre is, de megszabadítanak a hétköznapi stressztől, elmenekülhetünk a világ gondjai elől. Lehetünk bennük bárkik és bármik, egy olyan világban, melynek határait és szabályait játékfejlesztők alkották azért, hogy legyen mivel kikapcsolódnunk. Ki ne akarna ilyen világot alkotni?

Kétdimenziós platformer mellett döntöttem, mert egy háromdimenziós játék túl nagy kihívást jelentett volna. A fejlesztés kezdetekor semmi tapasztalatom nem volt játékfejlesztésben, illetve egy játékfejlesztő motort sem ismertem. Abban biztos voltam, hogy egy motorban fogom elkészíteni a játékot, mert egy mai világba illő programot akartam készíteni, amit egy jó motor segítségével az akkor tudásommal is el kellett tudjak készíteni. Ehhez végül a Unity-t választottam, ami C# scriptekkel működik kód oldalon, melyet a legjobban kedvelek az iskolában tanult nyelvek közül.

A fejlesztés kezdete, nulladik lépés

Abban biztos voltam, hogy ha törik, ha szakad, én egy játékot le fogok tenni az asztalra. Azt viszont nem tudtam, hogy pontosan, hogy fog kinézni, működni a játékom. Nem tudtam mi legyen a célja, lehessen benne harcolni, legyen benne okos ellenfél, legyen szerepjáték szerű, hosszú vagy rövid, könnyű vagy nehéz…

Végül megszületett az lapkoncepció, miszerint egy „pixel” játékot szeretnék csinálni, mint a Super Mario volt a maga korában. Ezzel a külső és a játékmenet legfőbb részei tisztázva lettek; nem lesz sok ellenfél, sem harc vagy komplikált rendszerek, de modernek legyen modern, legalábbis nem elavult. Ekkor jött egy elég nagy akadály; nem tudom hogyan működik a Unity. Ezt orvosolandó, vettem egy online kurzust a udemy.com-on. A kurzus a programozás alapjaitól indult, a végén pedig elég tudást szedtem össze egy egyszerű játék elkészítéséhez. Szerencsére az ott használt textúra csomag nagyon megtetszett, ezért azt megvásároltam a Unity asset strore-ban (Super Platformer Assets) és neki álltam fejleszteni. Alapos átgondolás, tervezés hiányában azonban ez nem volt egyszerű, így számos nehézség adódott a fejlesztés alatt.

Fejlesztési dokumentáció

Rövid leírás

A program egy egyszerű kétdimenziós platformer, mely ehhez az elképzeléshez hűen a SimplePlatformer nevet kapta. A játék Unity-ben készült, a kód Microsoft Visual Studio-ban C# nyelven lett írva. Adatbázishoz Json-t használtam, a GitHub működtetését Visual Studio Code segítségével oldottam meg.

Menü

A játék indítása után ez a képernyő fogadja a játékost:



A menü 3 gombot tartalmaz. Minden gomb egy funkcióval van ellátva.

A „Start Game” gomb elindítja a játékot.

Az „Extras” átviszi a játékost egy külön ablakra, ahol az eddig elért statisztikáit láthatja, mely egy adatbázisból tölt be.

A „Quit game” bezárja a játékot és menti az adatbázist. Az adatbázis mentéséhez nem kötelező a menüből kilépni a gomb használatával, a program bármilyen nem folyamat megsemmisítő bezárással végre hajtja a mentést, például az alt+f4 billentyűkombináció bármikor használható kilépéshez.

Játékmenet I. – Alapok egyszerűen

A játék menete egyszerű; a játékosnak limitált mennyiségű eszköze van a pálya teljesítéséhez. Ezen eszközök a mozgás és egy íj, mellyel nyilakat lőhet az ellenfelekre.

Ha a játékos egy ellenfélbe vagy csapdába lép, életet veszít. Ha minden élete elfogyott, a pálya újraindul. Ekkor az életerő visszatölt, de a próbálkozások száma csökken. Ha a játékos elveszíti az összes próbálkozási lehetőséget, a játékot elölről kell kezdje.

A felhasználót továbbá segítik a pályákon elhelyezett tárgyak, melyekkel interakcióba léphet. Ezek a létra, a felvehető élet, és a gomba.

Továbbá interakcióba léphet a felhasználó a pénzérmékkel is. Minden érme 100 pontot ad a játékosnak.

Minden pálya célja a kincsesládika elérése. Ha a játékos karakter hozzá ér a ládikához, rövidesen átkerül a következő szintre.



Játékmenet II. - Fő játékelemek részletezése

Cél és eszközök

A játék fókuszában a játékos karaktere áll. Ennek a karakternek a játékban nincs neve vagy neme, sem háttértörténete. A dokumentációban „Alex” néven hivatkozok rá.

A játkos célja Alex megóvása és a célba juttatása. Ezt alapszintű mozgással tudja megtenni, ami a futás, és ugrás képessége. Ezen képességek kiegészülnek néhány pályaelem használatával, mint a létra vagy a gomba.

A létrán fel és le lehet mászni, a gomba ruganyos, így a magasba löki Alexet.

A mozgás kiegészül egy fegyverrel. Alex az íját bármikor használhatja.

Pályaelemek

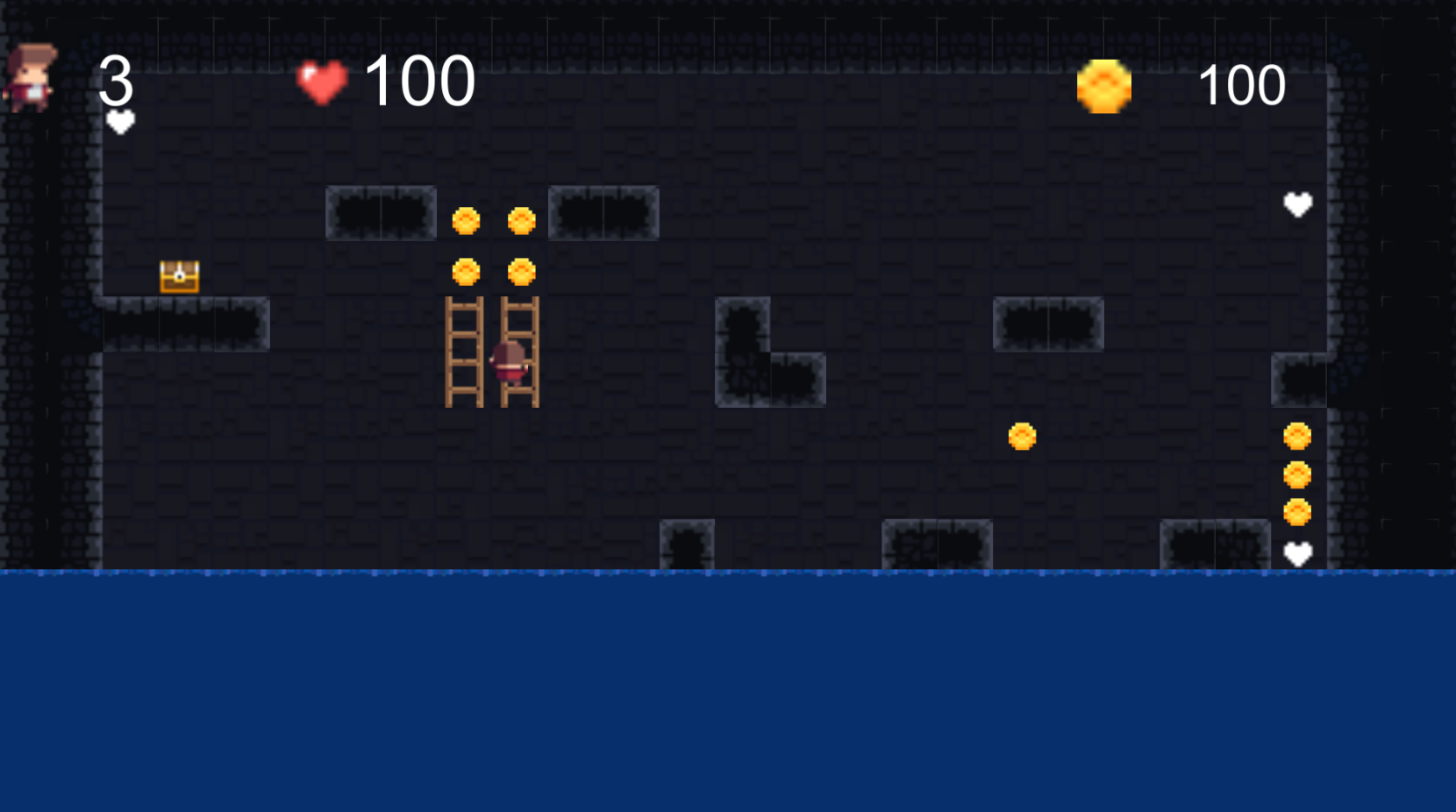
Minden pálya legfőbb eleme maga a járható talaj, az előtér. Ezen tud Alex biztonságosan közlekedni. Ez a fenti képen látható fekete, köves talaj.

Kulcsfontosságú elem továbbá a játékoskarakter, Alex is. Alex a legbonyolultabb objektum az egész játékban. A játék köré lett felépítve. Alex szabadon mozgatható a játékos által. Kellő mennyiségű animációval van ellátva, amik változnak az általa végzett tevékenységek során.

A háttér hasonlít az előtérhez, azonban játékmenet szempontjából nélkülözhető, csak esztétikai hatással bír. A háttér köves külsővel rendelkezik, így sejtetve, hogy valahol a föld alatt lehet a játék helyszíne.

A létrán lehet mászni és ugrani, illetve bárhol meg lehet állni rajta. Ezen objektumok létfontosságúak lehetnek néhány pálya teljesítéséhez, esetenként azonban rövidebb utakat vagy eldugott pályarészeket nyitnak meg.

Számos csapda található a pályákon. Ilynek az sziklatüskék, és a víz. Ezek érintésre sebzik Alexet. Elkerülésükhöz csupán egy jól időzített ugrás szükséges a legtöbb esetben. Ezen objektumok statikusak, így mozgásra nem képesek. Ez alól az árvíz a kivétel, mely bizonyos pályákon lassan elönti a szintet.



Gombára ugráskor az a magasba löki Alexet. A gombát az ugrással kombinálva nagy magasságokba lehet eljutni.

Alex rendelkezik egy íjjal, lőni azonban csak jobbra és balra tud egyenesen. Nem ez az elsődleges megoldás az ellenfelek elkerülésére, csak egy extra eszköz mellyel az ellenfelek végleg eltüntethetők és szükség szerint bármikor rendelkezésre áll.

A szinteken érmék találhatók. Ezen érmék nem szükségesek a pályák befejezéséhez, ha a játékos úgy dönt, hogy figyelmen kívül hagyja őket. Ez növeli a szabadság érzését és így a játékos maga tudja növelni vagy csökkenteni a pályák nehézségét azzal, hogy fel akarja venni őket, vagy sem. Néhány érme megközelítése trükkös lehet, így némi fejtörés esetenként szükséges lehet, ezzel megtörve az állandó rohangálást és ugrálást, mely a platformer játékok végjegye.

Fontos szerepet töltenek be az életerőt töltő szívek. A játkos csak ezek felvételével tud életet visszanyerni. Ha kellő mennyiségű szívet vett fel Alex, életereje 200 pontig nőhet. Ha 200 életerő ponttal rendelkezik és felvesz egy újabb szívet, akkor a próbálkozások száma eggyel megnő. Ez egy kulcsfontosságú mechanika lehet ügyetlenebb játékosoknak. Érdemes az összes ilyen objektumot összeszedni, amennyiben nem tűnik túl veszélyesnek a szív megközelítése.

Az ellenfelek egy célból léteznek; meg akarják gátolni a pályákon való szabad mozgást. Minden alkalommal, amikor Alex hozzá ér egy ellenfélhez (vagy csapdához) életet veszít és felugrik. Az ellenfelek a kis járőröző zselék. Mozgásuk kimerül a horizontális mozgásban, nem üldözik Alexet. A járőrök irányt váltanak, ha Alex a fejükre ugrik, ilyenkor is sebződik a karakter, ezzel megnehezítve az előlük való menekülést.

A kincsesládika a pálya célja, ez viszi tovább a játékost a következő szintre, amint Alexet megérinti a ládát. Egy kismértékű késleltetés van beépítve ebbe a folyamatba, így a felhasználó egy pillanatra megnyugodhat, illetve felveheti a ládika mögötti tárgyakat, ha vannak.

UI

A program valós időben mutatja a játékosnak az összes szükséges információt, amire szüksége lehet. Ezek a(z):

* Próbálkozások száma 
* Életerő 
* Összeszedetett pontok 

Unity játékfejlesztői motor

Működési elv

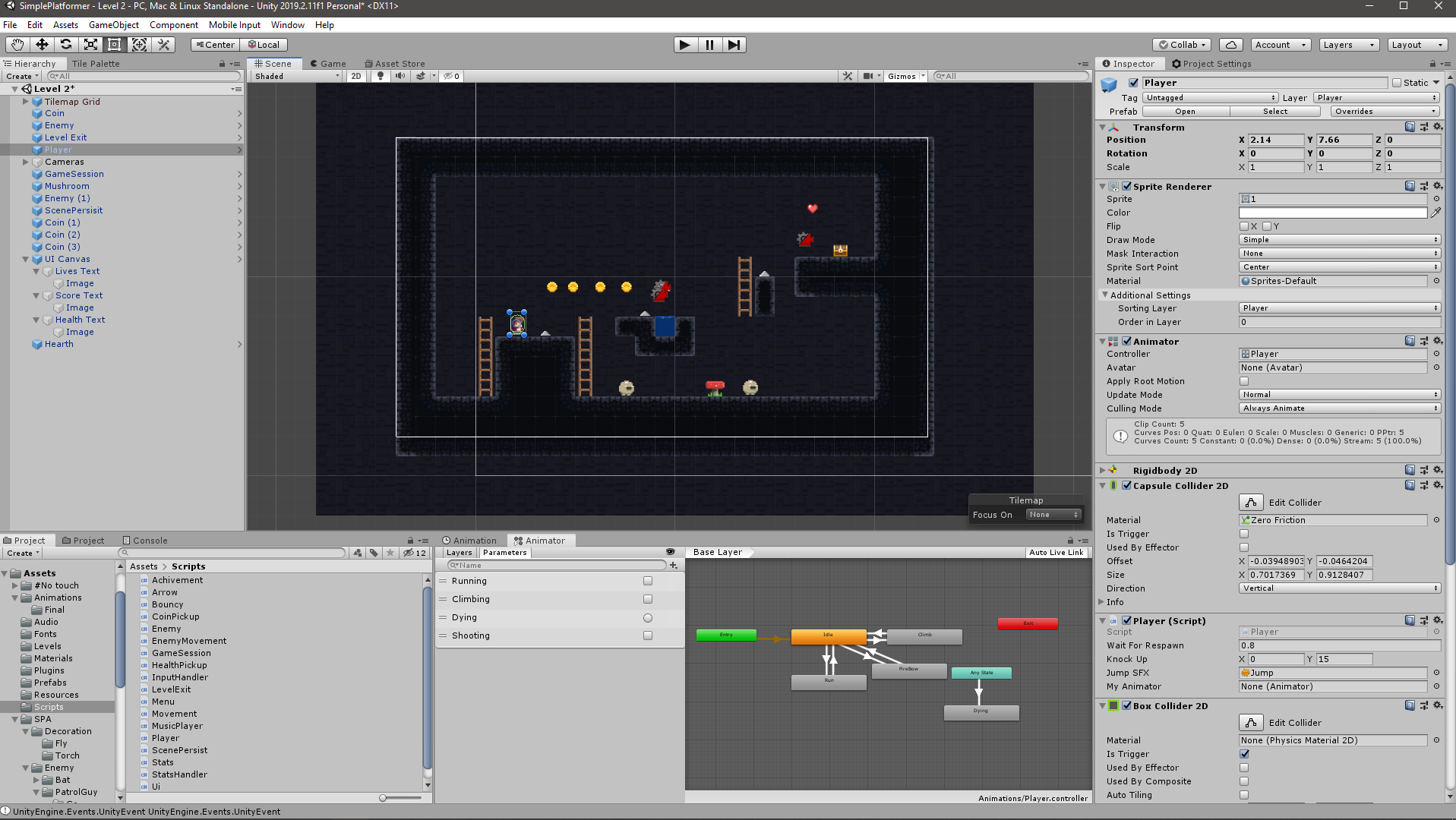
A motor számos beállítást és lehetőséget biztosít a fejlesztőnek a fejlesztés leegyszerűsítéséhez. Rengeteg előre elkészített eszköz, elem, és metódus létezik, melyek használatával sokkal gyorsabban elkészíthetünk egy teljes játékot, mint natív kódolással.

A motorban minden ablak, beállítás szabadon módosítható, ahogy az egy mai modern szoftvertől elvárható. A használatához könnyű hozzá szokni némi idővel.

Unity főbb elemei

A SimplePlatformer elkészítéséhez legtöbbet használt elemeket bemutatom, hogy később a fejlesztés bizonyos lépéseiről tudjak bővebben beszélni.

Ez a Unity motor:

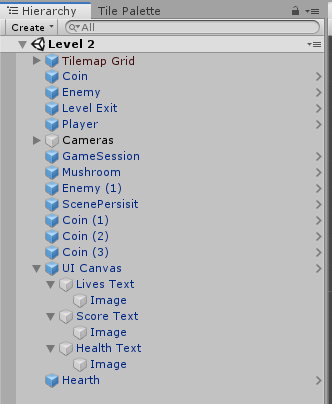


Első ránézésre egy rakás szöveg és ablak. Ezen ablakok közül a legfontosabbak a Hierarchy (bal oldal), az Inspector (jobb oldal), maga a Scene és a Game ablakok (közép). A pályákat Tile rendszerrel építettem meg, melyet később mutatok be.

Hierarchy (Hierarchia) bemutatása

A hierachy-ban vagy hierarchiában található az összes olyan objektumot vagy játékelemet, amit a jelenlegi Scenen azaz, jeleneten használva van. Ha itt nem látunk valamit, az nincs a jeleneten, így az nem fog létrejönni, lefutni és a build-be sem lesz benne.

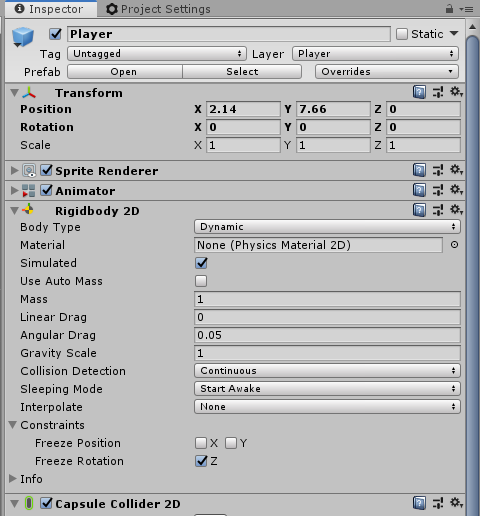
Tesztelés alatt könnyű „null reference exception” hibát kapni. A hibaüzenet nem mutat mindig egyértelműen a hiba forrására. Először érdemes a hierarchiában és az Inspectorban, azaz ellenőrző ablakban keresni a hibát, főleg, ha kódhoz nem nyúlt a közelmúltban a fejlesztő.



A hierarchiában a kék színű elemek prefabek, azaz előgyártott elemek. Ezeket az objektumok egyszer már el lettek készítve, csupán fel lettek helyezve a jelenetbe. Ez az egyszerű újrahasznosítás, rengeteg munkát spórol meg. Minden prefab módosítható utólag, ezen módosítások vastagon szedetten jelennek meg az Inspectorban.

Inspector (ellenőrző) bemutatása

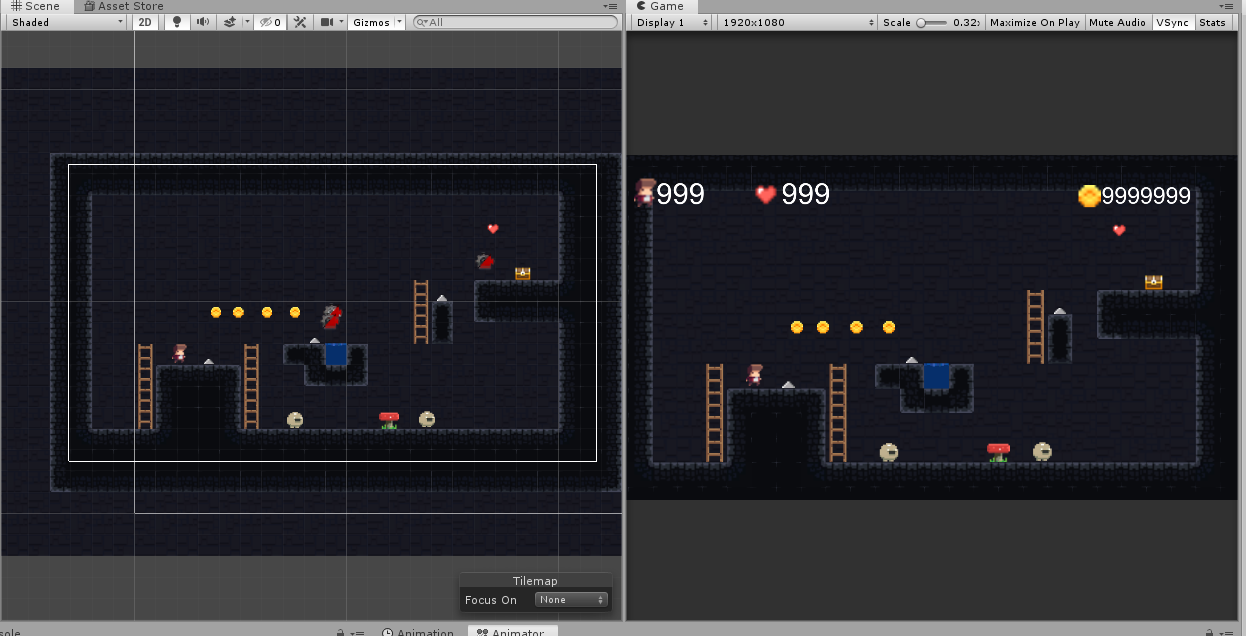
Talán ez a legegyszerűbb és legbonyolultabb ablak mind közül. Itt annak az objektumnak találhatóak az adatait, melyre egérrel kattint a fejlesztő. Egy eleme minden objektumnak van, a Transform. Ez a térbeli elhelyezéséért felel. Minél komplexebb egy objektum, annál több eleme lehet. Alexnak 9 eleme van, így ez az objektum a legkomplexebb.



Minden változtatás itt történik. Az itt lévő elemek a komponensek. Ha a motorban írunk egy scriptet, azt fel kell helyezni az objektumra, hogy működjön. Kódban ezeket a komponenseket módosíthatjuk, egyszerűen hivatkozni kell rájuk, meg kell hívni őket.

Az engine hatékony kezeléséhez sok időt kell befektetni az inspector megismerésébe. Legyen a fejlesztő kezdő vagy profi, inspector beható ismerete nélkül nem lehet Unityben dolgozni. Megismerése egyszerű, el kell olvasni mindent, ami benne van, esetenként belenézni a Unity online dokumentációjába. Ha valami neve nem egy értelmű, az egér rávitelével a motor kiírja mire való pontosan. Ez számomra nélkülözhetetlen volt.

Scene & Game (Jelenet és Játék) bemutatása

A két ablak egymás mellé helyezve nagyon hasonló, eleinte néha összekevertem őket. A jelenet ablakban lehet a játékot megalkotni. Itt kell elhelyezni az objektumokat, amiket aztán futtat az engine. A játék ablakban fut a játék. Csak azt látjuk benne, amit játék közben a játékos látna. Így könnyel lehet tesztelni a játékot build nélkül. 

A játék ablak teszteléshez önmagában kevés. Néha buildben is tesztelni kell, hogy minden lehetséges hiba kiderüljön. Bizonyos metódusok nem futnak a játék ablakban, mint az Application.Quit(); metódus sem.

Tapasztalataim – Unity engine

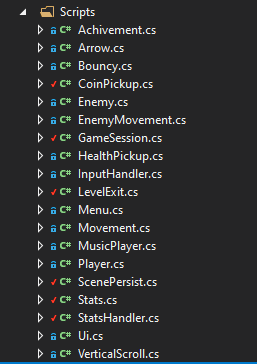
Alapértelmezetten nem a legkényelmesebb a szoftver használata, így hamar átrendeztem néhány ablakot. Rengeteg a megjeleníteni való, ami egy monitoron szűkösen fér el néha. Esetenként nehezen olvasható a sok szöveg, mert méretük elég

kicsi. Növelni rajtuk nem éri meg, hisz így is elég zsúfoltan férnek el bizonyos mezők egymás mellett. Helytakarítás szempontjából sok beállítás össze van csukva lenyíló ablakokba, így a fejlesztés kezdetén sokáig kerestem néhány elemet, mert valami alá be volt csukva.

Ha kellő odafigyeléssel áttekinti a fejlesztő az ablakokat, hamar észreveszi a kis nyilakat, amik az összecsukott opciókat rejtik. Ha ehhez és a zsúfoltsághoz hozzá szokok az ember szeme, minden könnyen elérhető és alakítható. Hamar ki tud alakulni az emberben egy kép, hogy az ő személyes Unity ablaka hogyan fog felépülni. Több monitor előnyös lehet, főleg játékfuttatás közbeni változások megfigyelésekor bizonyos objektumokon.

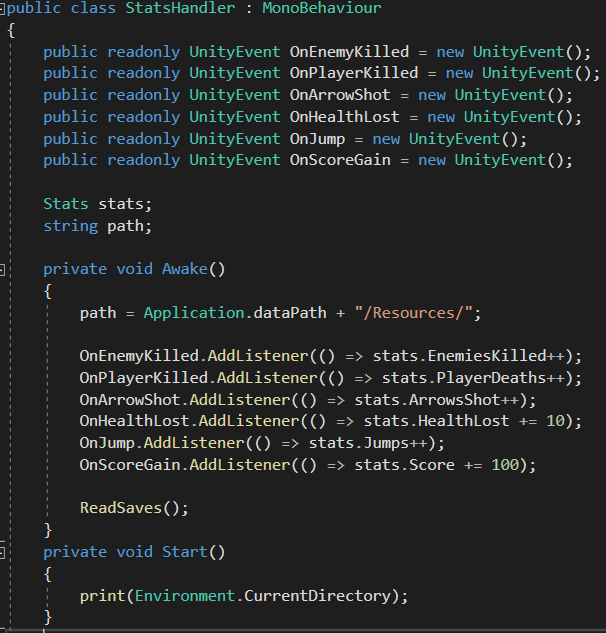
Objektumok és scriptek bemutatása

Elég sok scriptet írtam a fejlesztés végére. Ezekből nem mutatom be mind. A legtöbb néhány egyszerű sor, nem sok a magyarázni való rajtuk vagy nem tartalmaznak olyan kódsorokat, amik máshol hasonló formában nem lennének megtalálhatóak. Ilyenre példa az Arrow.cs, ami a nyílvesszőket eltünteti néhány másodperc múlva vagy a Menu.cs ami a menü gombokra kattintva betölt adott jeleneteket.



Adatbázis

Az adatbázishoz Json kiegészítést használtam. Sajnos nem volt egyszerű hozzá adni a Unityhez, de sikerült. Az adatbázis 3 fájlból áll. Az egyikben eltárolom az adatokat, ez a Stats.cs, a másikban ezeket az adatokat kezelem. Ennek neve StatsHandler.cs.



A kezelés után egy stats.json fájlba lesznek kiírva, így hoztam létre egy fájlba mentett adatbázist.

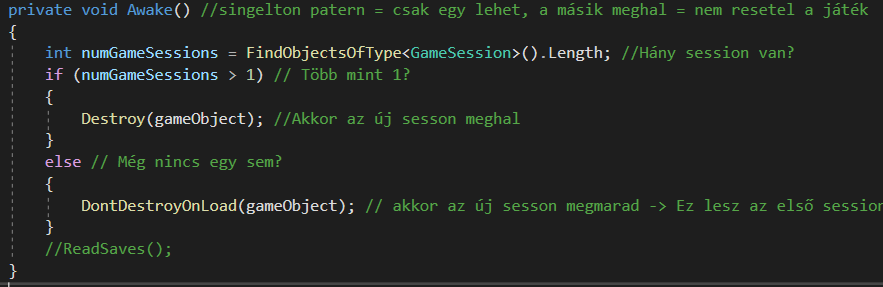
UnityEvent hívásokkal oldottam meg az adatkinyerést. Minden adattaghoz írtam egy Listener (figyelő) metódust, amit a jó helyen meghívok a többi scripten. Ha a listener lefut, az módosítja a Stats.cs adattagjait, melyek a program bezárásakor mentésre kerülnek a stats.json fájlba.

Ennek megírása hatalmas kihívást jelentett, melybe majdnem belebuktam. Ezt a „Majdnem végzetes hibák – Prefab használat” bekezdésben kifejtem.

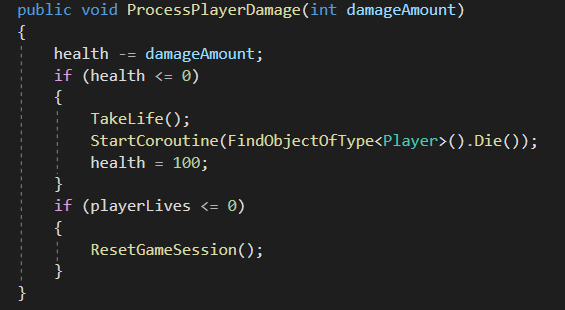
GameSeesion

Az egyik legfontosabb elem a GameSession. Ez a játékelem 2 scriptet tartalmaz. A StatsHandler.cs és GameSession.cs scripteket. A StatsHandler felelős az adatbázis működéséért, így minden jeleneten rajta kell legyen az első pályától az alkalmazás bezárásáig.

A GameSession.cs egy singelton alapú script. Ez azt jelenti, hogy belőle egyszerre csak egy lehet, és a jelenetek között átjárása van. A legtöbb scripttel ellentétben a GameSession tartalmaz egy metódust, ami lehetővé teszi számára, hogy a jelenet végén ne pusztuljon el, hanem menjen át a következőre és ott ne jöjjön létre.



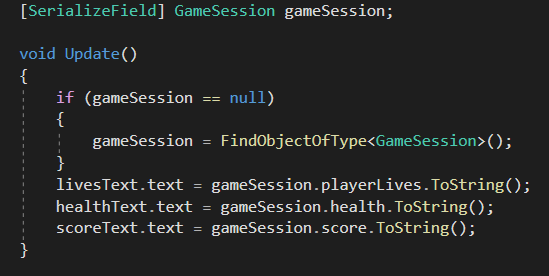
Ez a singelton alapú script felelős minden olyan játékelemért, amit a játékon végig kell vigyen a játékos, tehát Alex „adatait”. Ezek a próbálkozások száma, életerő és a pontok. Ezek szintről szintre szükségesek és nem szabad, hogy a jelenet végével megsemmisüljenek majd a következőn alapértékekkel létre jöjjenek. Ha a játékos minden próbálkozási ehetőséget eljátszik, visszakerül a menübe és a GameSession elpusztul.



A GameSession.cs tartalmazza Alex „adatait”. Ezek az adatok azért nem Alexon vannak, mert ha meghal, a jelenetet újra kell tölteni, így Alexot is. Rajta így nem lehet singeltont alkalmazni. Ezért jött létre a GameSession objektum.

UI

A UI a GameSession.cs scriptből kapja meg az adatokat. Kezdetben ezt a GameSession csinálta, de végül külön szedtem, hogy a kettő ne függjön egymástól.

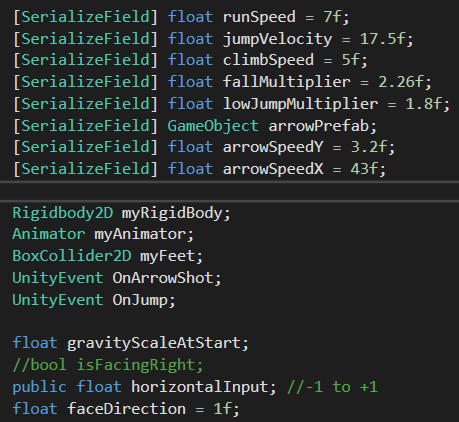


A metódus egyszerűen szöveggé alakítja a GameSession-ben tárolt értékeket, hogy azt egy text objektumba kiírva lássa a játékos ezen értékeket.

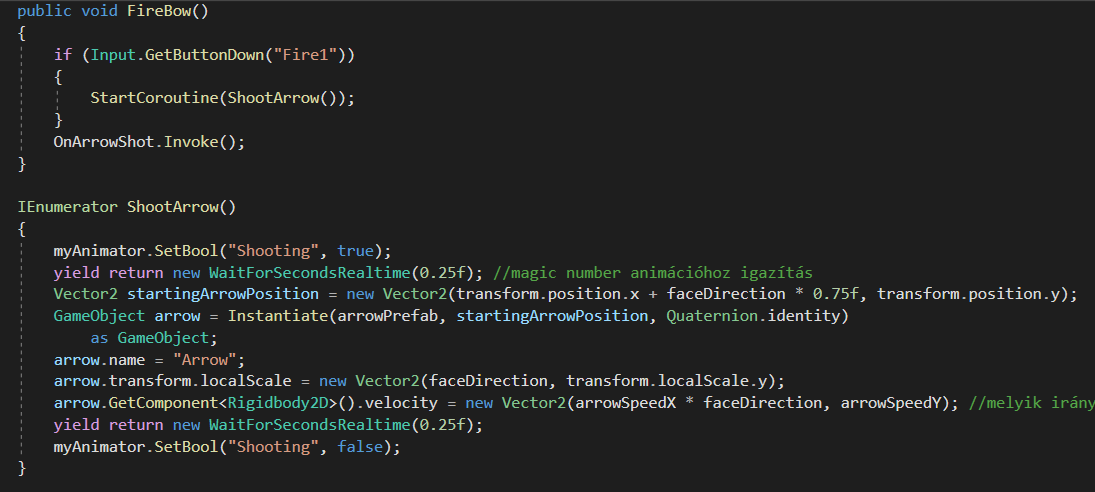
Alex – Player & Movement

A Player.cs és Movement.cs scriptek a legnagyobbak és csak Alex irányításához köthető metódusokat tartalmaznak.

A Movement.cs script tartalmaz minden olyan változót és függvényt, ami a karakter mozgatását teszik lehetővé. Mivel elég sok minden tartozik a mozgás kategóriába, így elég sok változót kellet létre hoznom. Ezeken SerializeField-et használtam, hogy a motorból közvetlen hozzájuk férhessek és akár tesztelés közben is tudjam változtatni az értékeket. A SerializeField privátként tartja a változókat.

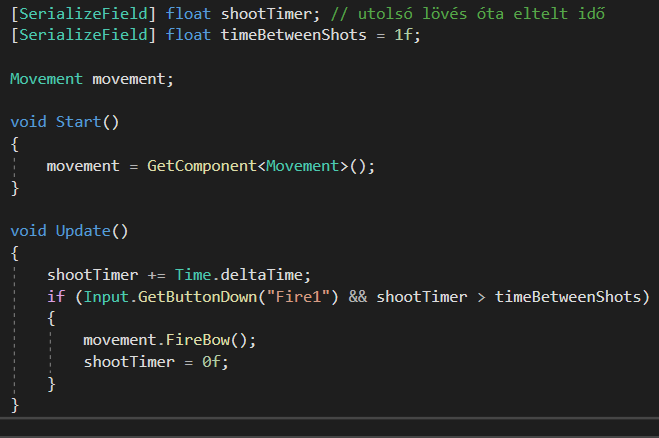


Kihívást jelentett a lövés elkészítése. A lövésnél nem volt elég meghívnom egy általam a motor animátorában elkészített animációt. Itt létre is kellett hoznom egy lövedéket, ami képes volt adott irányba mozogni és ütközni más objektumokkal.



A cél itt az volt, hogy a lövés az animációhoz legyen igazítva és ne lehessen megállás nélkül lőni. Ehhez egy coroutine kellett. A coroutine kötelezően tartalmaz egy yield return sort. Ez lehetővé teszi, hogy a metódus mielőtt lefutna ismét, várjon arra, ami a yield return után van. Nálam ez valós időben számolt másodpercek voltak, amiket teszteléssel állapítottam meg. A coroutine magától nem fut, el kell indítani a StartCoroutine metódussal.

A lövés metódus elindítja az animációt, ami egy bool érték alapján indul, majd vár egy kicsit a metódus. Utána megkeresi, hogy hova kell majd létrehozni a lövedéket. A lövedék helye Alex előtt van kicsivel. Ehhez tudni kell merre néz Alex. Utána létrejön a lövedék „Arrow” néven és elindul arra, amerre Alex néz. Ezek után a metódus vár kicsit, hogy az animáció befejeződjön, majd az animációból kilép a metódus a bool érték hamissá téttelével.

Ez azonban nem elég ahhoz, hogy ne lehessen esztelenül nyilakat lőni. Ez nem gátolja a „Fire” gomb bármikori aktiválását. Ehhez külön scriptet írtam, egy InputHandler.cs-t. 

Ez a script számolja a lövések közti időt és nem engedi lőni a játékost, ha nem telt el egy másodperc. Ezt az időt találtamoptimálisnak az animáció végig játszásához. Így biztos nem tud a felhasználó úgy lőni, hogy az animáció ne játszódjon le, de a lövés nem függ az animációtól.

Tapasztalataim – Scriptek írása

Kezdetben elég nehezen állt rá az agyam a kódolásra. Az objektum orientált programozási ismereteim hiányosak voltak, amit elég hamar ki kellett egészítenem. OOP nélkül nem lehet Unityben dolgozni, hisz egy motorban létező objektum egy komponensének egy változójának elérése lehetetlen OOP ismeretek nélkül. Ebből rengeteget tanultam, bár elég nehéz kezdet volt, ezért is sok a komment a kódokban. A fejlesztés vége felé is néha belezavarodtam a komponensek elérésbe, de kis idővel mindent meg tudtam oldani.

Majdnem végzetes hibák – Prefab használat

Felhasznált eszközök

Fejlesztési platform

Microsoft Windows 10 Pro 2019 ver. 1903

OS Build: 18362.1139

Motor (engine)

Unity – personal licensz

2019.2.11f1 verzió

Programnyelvek és keretrendszerek

Microsoft Visual Studio - 2019

C# programnyelv a scriptekhez

Json programnyelv az adatbázishoz

Visual Studio Code

Verziókezelő

GitHub

Visual Studio Code segítségével kezelve

Egyéb eszközök és tájékozódási helyek

Windows

Snip & sketch a dokumentáció képeihez

Sticky notes gyors jegyzetekhez

Unity motor online dokumentáció

Unity motor online fórumok

Udemy.com

GameDev.tv a unity alapok elsajátításáért

Rendszerkövetelmény

Zárógondolatok

Játékot fejleszteni csak azért, mert motort használ az fejlesztő, nem lesz könnyű. Nagyon sok kihívással találtam magam szemben, néhányat idővel le tudtam küzdeni, sajnos nem mindet. Sokszor éreztem reménytelennek a helyzetet. Rengeteg olyan próbléma adódik egy ekkora méretű fejlesztés során, amikre nem álltam készen. A legnagyobbat a „Majdnem végzetes hibák – Prefab használat” bekezdésben részleteztem. Az iskolai rövid programok írásától nagyon eltért ez a projekt, így már értem, miért fontos az alapos tervezés.

A tiszta, egyértelmű koncepció hiányában sokszor újra kellett írnom már kész kódokat, mert nem voltak elég objektum orientáltak, sem újrahasznosíthatóak. Ezek sosem hagytak kellemes élményeket maguk után, és mégis elég pozitív gondolatok jutnak eszembe, ha a fejlesztésre gondolok. Nagyon jó tapasztalatszerzés volt ez a projekt, rengeteget tanultam mind a Unity használatról, mind a programozásról.

Előismeret nélkül nem volt bölcs döntés belevágni ebbe a projektbe, de nem bántam meg. Nem lett olyan, mint amilyet megálmodtam, de egy működő, egészet alkotó szoftver lett a végére. Ez számomra hatalmas teljesítmény és egy nagy lépéssel közelebb kerültem ahhoz, hogy a jövőben programozóként hivatkozhassak magamra!