Veszprémi Szakképzési Centrum Ipari Technikum

**Szakképzés neve: Szoftverfejlesztő**

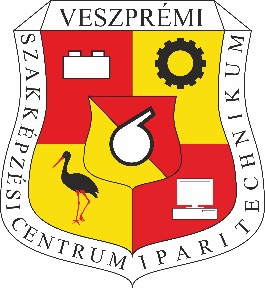
**OKJ száma: 54 213 05**

SZAKDOLGOZAT

ShootingStar

Farkas Gábor Urbán Tamás

témavezető 13.C



Veszprém, 2021

[**1** 3](#_Toc71539067)

[**2** **Felhasználói dokumentáció** 4](#_Toc71539068)

[**2.1** **Általános specifikáció:** 4](#_Toc71539069)

[**2.2** **Rendszerkövetelmények:** 5](#_Toc71539070)

[***2.2.1*** ***Hardverkövetelmények*** 5](#_Toc71539071)

[***2.2.2*** ***Szoftverkövetelmények*** 5](#_Toc71539072)

[**2.3** **A Program használata:** 6](#_Toc71539073)

[**3** **Fejlesztői dokumentáció** 16](#_Toc71539074)

[**3.1** **Témaválasztás indoklása:** 16](#_Toc71539075)

[**3.2** **Az alkalmazott fejlesztői eszközök:** 17](#_Toc71539076)

[***3.2.1*** ***Unity Játékmotor*** 17](#_Toc71539077)

[**3.3** **Definíciók:** 20](#_Toc71539078)

[**3.4** **Feladatspecifikáció, algoritmusok:** 21](#_Toc71539079)

[***3.4.1*** ***Karaktermozgás*** 21](#_Toc71539080)

[***3.4.2*** ***Fegyverek, lövés és annak effektjei*** 23](#_Toc71539081)

[***3.4.3*** ***Életerő és ellenfelek*** 24](#_Toc71539082)

[***3.4.4*** ***Menü*** 25](#_Toc71539083)

[**4** **Tesztelési dokumentáció** 26](#_Toc71539084)

[**5** **Továbbfejlesztési lehetőségek** 28](#_Toc71539085)

[**6** **Összegzés** 29](#_Toc71539086)

[**7** **Forrásmegjelölés** 29](#_Toc71539087)

**1** **Bevezető**

A mai világban egyre nagyobb figyelmet kap az e-sport, valamint a tripla A-s (AAA - A Triple-A játék kifejezés főleg az angol kultúrában létezik, bár néhányan már itthon is hallhatták. Ezek a játékok olyan nagy nevek hordozói, mint a Call of Duty, Battlefield, Star Wars és társai, és főleg az új részekre, új megjelenésekre vonatkoznak.)  
A Triple-A játékokat főleg a blockbuster, avagy kasszasiker filmekhez lehet hasonlítani, csak a játékiparban.

Nagy a büdzsé, rengeteg ember dolgozik rajtuk és nagy körülöttük a hype, avagy az izgalom) játékok megjelenése.

Külön kategóriák szerint csoportosítva, különböző díjakat bezsebelve ezzel. Ennek tudatában én is beleástam magam a játékkészítés világába, a C# programozási nyelven működő Unity játékmotor segítségével. Az FPS (First-Person Shooting) műfaj vonz a leginkább, így arra az elhatározásra jutottam, hogy egy sztori alapú, belsőnézetes lövöldözős játékot fogok készíteni, mely a jövőben játszódik és karakterünkkel különböző pályákat kell teljesíteni különböző ellenfelek kiiktatásával.

Tervben van későbbiekben egy esetleges továbbfejlesztés vagy egy második rész fejlesztése, amennyiben “sikert arat”.

Nagy kedvvel tanulok minden újdonságot Unityvel kapcsolatban, így mindig találok valami kis újítást.

Az informatika, azon belül a szoftverfejlesztés ezen területén tudnám elképzelni a jövőmet.

# **2** **Felhasználói dokumentáció**

## **2.1** **Általános specifikáció:**

A program, jelen esetben a játék, a közeljövőben játszódik, ahol egy robotot kell irányítani, név szerint Acélfejet, aki egy ellenséges frakció robotjait támadja meg. A játékos feladata, hogy sikerrel végig vigye mind a három pályarészt, ami annyit takar, hogy minden egyes ellenfelet ki kell iktatnia az adott pályarészeken. Minden pályarész végén van egy boss harc, ha az utolsó pályán lévő boss-t (fő ellenséget) megöli a játékos akkor sikeresen végig vitte a játékot és nyert. Három féle fegyvere lehet karakterünknek melyekkel különböző mértékű sebzést tud okozni.

## **2.2** **Rendszerkövetelmények:**

#### ***2.2.1*** ***Hardverkövetelmények***

* Minimum:
  + Processzor: 2GHz dual core
  + Memória: 4GB DDR3 RAM
  + Videókártya: 1 GB RAM 128bit GDDR5
  + Háttértár: 6GB HDD szabad tárhely
  + Operációs rendszer: Windows 7 (64-bit)
* Ajánlott:
  + Processzor: 3GHz quad core vagy jobb
  + Memória: 8GB DDR4 Ram
  + Videókártya: 2GB RAM 128bit GDDR5 vagy jobb
  + Háttértár: 6GB SSD szabad tárhely
  + Operációs rendszer: Windows 10 (64-bit)

#### ***2.2.2*** ***Szoftverkövetelmények***

* Windows operációs rendszer
* DirectX 11. Verziója

## **2.3** **A Program használata:**

A program az exe fájllal lesz indítható, miután máris belevetheti magát a felhasználó a játékélménybe. A játékost a főmenü fogadja indításnál. Itt több opció közül lehet választani: **Játék**, **Beállítások**, **Kilépés**.

A játék menüpontra kattintva kezdődik a játékmenet az első pályarészen. A beállítások menüpontra kattintva bedobja a felhasználót egy beállítások almenübe, ahol beállításokat tud eszközölni kényelmére, az egér érzékenységre vonatkozóan. A harmadik, kilépés nevű menüpontra kattintva tudja bezárni a programot a játékos. A játékmenet közben, ha a felhasználó ki szeretne lépni, egyszerűen megteheti ezt, a menü megnyitásával majd a “Kilépés” opcióra kattintva, vagy akár a főmenübe lépve a “Főmenü” opcióra kattintva. A Játék elkezdésekor, a játékos az első pályarészen “ébred”, ahol egy információs asztal, a közepén egy gömbbel várja majd.

Mikor közel ér ehhez a játékos, egy párbeszédablak fog megnyílni, ahol tájékoztat minket a küldetésünkről.

Fontos, hogy a játékos egy **zárt** pályarendszerben van, így nem lehetséges, hogy elhagyja a játékteret. A játéktéren belül a játékos szabadon mozoghat, a cél, mint fentebb ismertetve volt, hogy kiiktassa az összes ellenfelet. A főgonosznál lesz egy “kulcs” melyet, úgy tud megszerezni, ha megöli, valamint megöl minden más ellenfelet. Így esélye lesz a következő pályarészre lépéshez. Ezzel a kulccsal egy nagy üveg ajtót tud majd kinyitni a játékos, amin, ha áthalad, eljuttatja a következő pályarészre.

Minden szinten így tud majd tovább haladni. A második, illetve harmadik szinten nem lesz már “információs asztal”, így párbeszéd sem. Amint átlépett, egyből folytathatja is a játékot, teljesen újra töltődött életerővel. Minden pályarész elején egy checkpoint (A pályának azon része, amit a játékosnak el kell érnie, hogy tovább haladhasson) lesz, melyhez visszatöltődik a játékos, ha meghal.

Ilyenkor szintén újra töltődik az életereje a játékosnak. Ahogy halad a pályarészeken a játékos, úgy lesz egyre nehezebb dolga, ugyanis egyre több ellenféllel kell megküzdenie. Amint a harmadik, egyben az utolsó pályarészt is sikeresen végig játssza, és áthalad az utolsó üvegajtón, egy end credit fogadja, mely jelzi, hogy vége a játéknak, valamint megköszöni és gratulál a játékosnak, hogy sikerrel tudta venni az akadályokat.

**Irányítás**

Az irányítás az FPS játékoknál megszokott módon működik.

* WASD billentyűk a különböző irányokba való mozgást teszik lehetővé.
* R billentyű használatával tudjuk újra tölteni fegyverünket.
* ESC billentyű a menü előhívásáért felelős a játékmenet közben.
* CTRL billentyű nyomva tartásával guggol le a karakterünk, felengedésével feláll eredeti állapotába.
* SPACE billentyű segítségével ugrik a karakterünk.
* Az 123 billentyűkkel fegyvert válthatunk, ha éppen van nálunk több fajta is.

**HUD**

A HUD (Head-up display) egy a játék közben megjelenő felület melyet a játék folyamán végig lát a játékos. Ez a játékosnak fontos karakter adatokat vizualizálja, hogy a játékos tudja mivel rendelkezik a karaktere.

Életerő sáv

Karakterünk rendelkezik egy életerő sávval, amely mutatja aktuális életerejét. A maximális életerő 100 életerőpontnak felel meg, amit a különböző ellenfelek, különböző sebzésértékkel tudnak csökkenteni. Amint egy pályát végig viszünk és egy új pályára érünk, az életerő visszatöltődik maximálisan, ezzel segítve az aktuális pályán lévő megmérettetést. Ez a sáv a képernyőnk jobb alsó sarkában helyezkedik el.

Acélfej emellett egy ammo bar-ral (lőszer sáv) is rendelkezik, amely mutatja, hogy éppen mennyi lőszer van még az adott tárban, illetve mennyi lőszere van még összesen a karakternek. Fegyvertípusonként eltérő lőszermennyiség lehet egyszerre a tárban.

Eltérő ellenfeleinknek eltérő mennyiségű életerejük van. A játékosnak egy beépített célkereszt áll rendelkezésül, ezzel segítve a célzásban és a lövésben.

Amennyiben karakterünknek elfogy az életereje meghal, és az adott pályarész elejéről kell elkezdenie a játékot.

**Fegyverek**

A játékban 3 különböző fegyver játszható, ebből kettővel kezd a karakter, és a harmadikat jutalomként gyűjtheti be egy boss (főgonosz) legyőzésekor.

**Pisztoly**

A pisztoly a második számú fegyvere főhősünknek, amely szinte végtelen mennyiségű tölténnyel rendelkezik, ami 10/800-at (10 golyó fér egyszerre a tárba és 808 tölténnyel rendelkezik összesen) jelent. Ezzel ellentétben elég alacsony sebzésértékkel bír és alacsony a tűzgyorsasága, ami azt jelenti, hogy egy egérkattintásra csak egy lövedéket képes lőni, így azt csak bizonyos szituációkban lesz ajánlatos használni. Ez a fegyver tölt a leggyorsabban újra a játékban.



**AR (gépfegyver)**

Az AR vagyis az Assault Rifle lesz a karakterünk elsőszámú fegyvere, közepes sebzésértékkel bír, valamint közepes tűzgyorsasággal. Ennél a fegyvernél fontos, hogy véges lövedékszámmal bír. Alapjáraton 30/200, azaz 30 golyó van alapból a tárban és még 200 van a karakternél, amit képes betölteni a fegyverbe. Abban az esetben, ha kifogy és nincs nálunk egy harmadik fegyver, csak a pisztolyt képes használni a játékos, ezzel szorult helyzetbe kerülve az ellenfelekkel szemben. Közepes újratöltési idővel rendelkezik.



**MG (géppuska)**

Az MG vagyis a Machine Gun, az epikusan megszerezhető harmadik számú fegyver, mely hatalmas fegyverténnyel bír. Sebzése valamivel kevesebb mint az Assault Rifle-nek ellenben a tűzgyorsasága számottevően jobb. 80/400-as tárral rendelkezik. Az újratöltési ideje ennek a fegyvernek a leghosszabb.

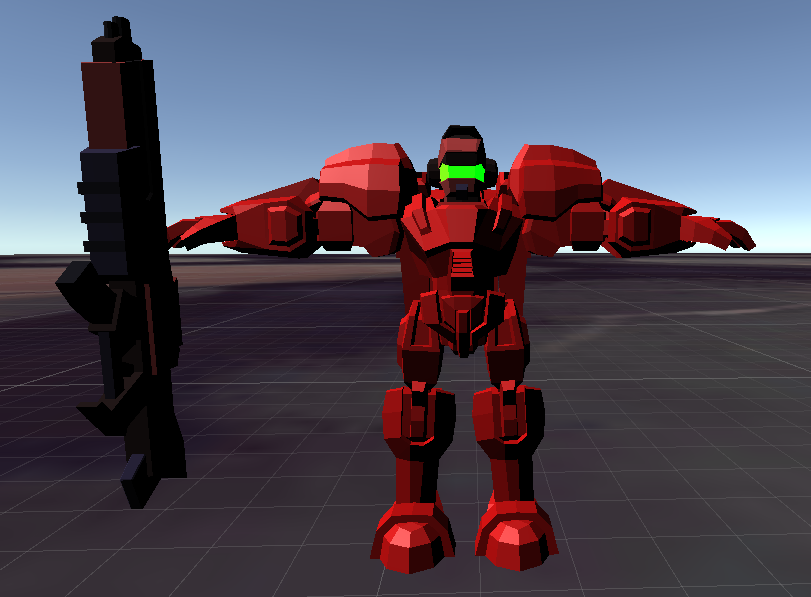


**Ellenfelek**

*Egyszerű ellenfelek*

**#1 Harci robot**

Ez az egység alacsony sebzésértékkel bír, közepes tűzgyorsasága van, valamint az alap ellenfélegységek közül a legtöbb életerővel, ezáltal érdemes folyamatosan tartani a távolságot ellene.



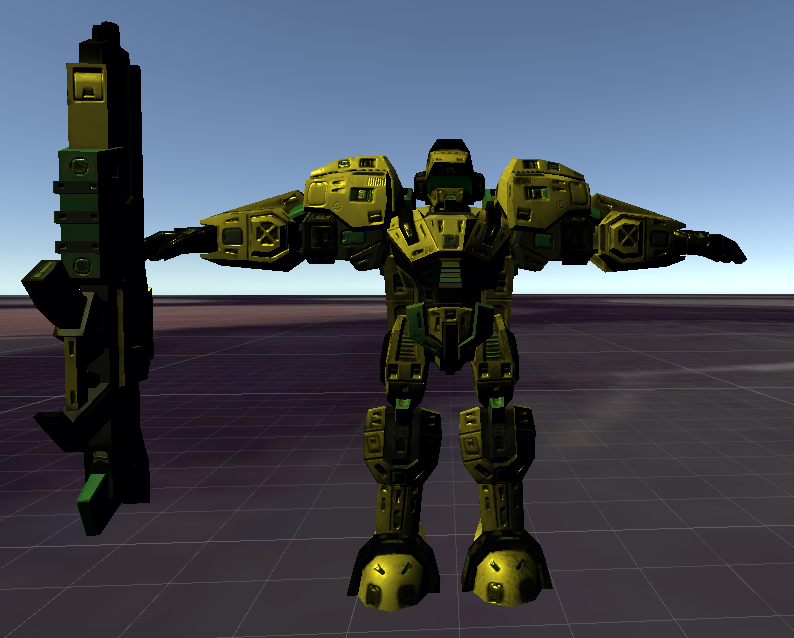
**#2 Harci robot**

A #1 harci robottól eltérően ő közepes sebzésértékkel bír, de ugyan azzal a tűzgyorsasággal. A #2 harci robotnak közepes méretű életereje van.



**#3 Harci robot**

A #3 harci robot, vagyis szuper harci robot a legveszélyesebb ellenfél, ha a boss-okat nem vesszük figyelembe. Nagy sebzéssel és hatalmas tűzgyorsasággal bír, viszont kevés életerővel rendelkezik.



*Fő ellenfelek (Boss-ok)*



**#1Boss**

Az első pályarész végén szereplő boss. Hatalmas életerővel rendelkezik és nagy sebzéssel. Az ő kiiktatásával szerezhetjük meg az Machine Gun-t.

**#2Boss**

A második pályarész végén szereplő boss. Több életerővel rendelkezik, mint az első boss, de ugyanakkora sebzést képes okozni.

**#3boss**

A harmadik pályarész végén szereplő fő boss. Ő a legerősebb a három boss közül. Az ő megölésével a játékos sikeresen teljesíti a játékot.

**Pálya**

Három pályarészen kell végig játszani a történetet. Mindegyik pályarész rendelkezik mind a három egyszerű ellenféllel, valamint egy boss-al. A pályák elemei megegyeznek, így azok csak kialakításban térnek el egymástól. A pályák főbb elemei falak, rámpák. Az ellenfelek akkor támadnak, ha a játékos valamelyik ellenfél látóterébe ér, tehát a pálya azon részén, ahol az ellenfelek képesek detektálni főhősünket.

# **3** **Fejlesztői dokumentáció**

## **3.1** **Témaválasztás indoklása:**

Jelenleg a világban, a játékpiac hatalmas részét képezi az informatikának, valamint foglalkoztatja a szoftverfejlesztők nagyon nagy részét, így számomra nem volt kérdés, hogy ezen területen gondolkozom a szakdolgozatom megírásában. Mivel sok FPS játékkal játszottam életem során, így erre a műfajra van a legnagyobb rálátásom, elképzelésem arról, hogyan is kell kinéznie egy ilyen típusú játéknak. Véleményem szerint a játékok, mint szoftverek hatalmas értéket képviselnek több megközelítés szerint is. Hatalmas reklámerővel bírnak, nagy embercsoportokat képesek megmozgatni, típustól függően képesek kikapcsolni az embert, vagy versenyszellemet adni nekik, versengésre buzdítani őket. Ezen felül képesek fejleszteni a kéz-szem koordinációs képességet, reakció időt, csapatban való dolgozást, valamint a minőségi kommunikáció kialakulását. Ezeket a képességeket remekül be tudja illeszteni az ember a való életbe és profitálni tud belőle majd a későbbi munkahelyén, bármivel is kelljen foglalkoznia.

## **3.2** **Az alkalmazott fejlesztői eszközök:**

Unity 2019.4.14f1 (64-bit) verziójú játékmotor segítségével készítettem a programom. A Unity egy remek fejlesztői környezet játékok készítéséhez, ugyanis a programon belül minden adott hozzá, tulajdonképpen erre lett kitalálva.

A Unity C# programozási nyelv alapú, tehát a scripteket C# nyelven kell megírni hozzá. Ehhez én a Microsoft Visual Studio 2019-et használtam segítségül mely segített az átláthatóságban és a szintaktikai hibák elkerülésében.

A karakter modelleket, a pálya elemeket, valamint a játékban megtalálható fegyvereket szabadon felhasználható free asset-ek ként töltöttem le a Unity hivatalos oldaláról.

A modellekhez különböző események esetén, különböző animációk vannak rendelve, ezek a Mixamo nevű weboldal ingyen hozzáférhető, és használható animációi melyeket az oldalon még testre is szabhat a fejlesztő, gondolok itt az adott animáció gyorsaságára, loopolására, (újrajátszás) vagy akár Framenkénti (képkocka) változtatására. Emellett a már meglévő modelleket feltöltheti, és előre megnézheti hogyan is fog kinzéni az adott modell az adott animációval ezzel rengeteg időt megtakarítva.

#### ***3.2.1*** ***Unity Játékmotor***

A Unity egy cross-platform játékmotor, amit a Unity Technologies fejlesztett. Kezdetben kizárólag csak az Apple OS X-en lehetett használni, de a nagy népszerűsége miatt mára már elérhető a Windowson és Linuxon is. A motor célja a három és kétdimenziós játékok fejlesztése, valamint szimulációk futtatása. A Unity különböző grafikus alkalmazásprogramozási interfészeket is támogat, Windowson elsősorban a Direct3D-t.

**A Unity beépített metódusai**

*Raycast*  
Egy láthatatlan sugarat tudunk meghatározni, amivel érzékelni tudjuk az objektum körüli területet. Ezzel a működési elvvel dolgozik nálam a lövés regisztrálása.

*CompareTag*  
Az objektumoknak különböző címke megnevezéseket tudunk adni, amivel könnyeben tudunk hivatkozni rá, vagy felismerni. A CompareTag metódussal tudjuk ezt összehasonlítani.

*SceneManager*  
A jelenetmenedzser segítségével tudunk a programban utalni a jelenetváltozásokra. Itt tudjuk kezdeményezni egy másik jelenet betöltését. Például, ha a játékból visszalépünk a főmenübe, akkor egy másik jelenetet kell betöltenie, így ez SceneManagerrel működik.

**A Unity editor elemei**

*Scene*  
Leginkább jelentere utal. Minden Unity program legalább egy jelenetből áll. Itt tudjuk megalkotni a kívánt jelenetet, például a főmenüt vagy a játék különböző szintjeit.

*Build Settings*  
A Unity-ben ebben az ablakban tudjuk rendszerezni a különböző jeleneteket, minden jelenethez tartozik egy egész szám (sorszám) amivel tudunk a programon belül utalni rá.

*Camera*  
A kamera segítségével tudjuk megmutatni a játékosnak a játékon belüli területet. Mivel ez egy FPS kategóriájú játék így a kamera a karakterünk “szeme”, így azon keresztül láthatjuk a jelenetet.

*Canvas*  
A vászon az a terület, ahol elhelyezhetjük a kezelőfelülethez szükséges gombokat, szövegeket. Minden kezelőfelületi elem gyerekobjektuma a vászonnak. Egy jelenetben több vászon is lehet.

*Grid*  
A rács az az alaposztály, ami az egyenletesen elosztott pontok és vonalak elrendezésének ábrázolásához szükséges. A rács komponens tárolja a rács elrendezésének dimenziós adatait. Segítségével le tudjuk kérdezni a különböző cellák helyzetét. A rács XY síkban van elhelyezve és mindig a (0,0) koordinátától kezdődik. Az X és Y koordinátái csak pozitív értékeket vehetnek fel.

*Sprite*  
A Sprite egy kétdimenziós grafikai objektum. A grafikák bitképes képekből származnak. A Sprite osztály elsősorban azonosítja a kép egy részét. Ezt a képet a játékobjektum egy SpriteRenderer komponens segítségével tudja megjeleníteni.

*Sprite Sheet*  
A Sprite Sheet egy olyan kép, amelyen több kisebb kép, vagy animáció áll. A kis képek egy nagy képpé történő kombinálása javítja a játék teljesítményét, csökkenti a memóriahasználatot, és felgyorsítja a játék indítási idejét.

*Tilemap*  
A “csempetérkép” tárolja a Sprite objektumokat a Grid által meghatározott elrendezésben.

*TilePalette*  
A csempe palettán tároljuk a programban felhasználni kívánt csempéket

**MonoBehaviour fontosabb metódusai**

*MonoBehaviour*  
Az összes Unity script a MonoBehaviour ősosztályból ered. Amikor C# nyelvet használunk, kifejezetten a MonoBehaviour-ből kell származtatni.

*Start*  
A Start metódus a script meghívása után csak egyszer hívódik meg, még az Update metódus előtt. Itt érdemes kezdeti értékeket adni a változóknak. Nem minden MonoBehaviour scriptnél fontos a Start metódus használata.

*Update*  
Az Update metódus a script meghívása után minden frame-en (Képkocka) lefut. Az egyik leggyakrabban használt funkció, amikor játékokat implementálunk. Nem minden MonoBehaviour scriptnél szükséges az Update metódus használata.

*Send Message*  
Ezzel a metódussal tudunk kommunikálni az objektumok között. Három paraméteres üzenetet is továbbíthatunk. Az elsőben a meghívni kívánt metódus nevét kell megadni, a másodikban paraméter értéket adhatunk át a meghívott metódusnak, a harmadikban pedig kérhetünk hibát, ha a meghívott metódus nem létezik.

## **3.3** **Definíciók:**

**GameObject**

A játékobjektumok alapvető egységei a Unity-nek, amik különböző komponenseket és tulajdonságokat tárolnak. Mindig van egy Transform komponense, amit nem lehet eltávolítani.

**Prefab**

Komponensekkel és tulajdonságokkal kiegészített játékobjektumokat tárol. A prefab gyakorlatilag sablonként működik, amivel bármikor egy ugyanolyan objektumot tudunk létrehozni a scene-ben (jelenet). Bármi változás, amit végrehajtunk a prefabon, azonnal látszódik a sceneben elhelyezett példányokon is.

**Transform**

A Transform komponens határozza meg az objektumok pozícióját, forgását és a méretarányát a scene-ben. Minden játékobjektumnak van Transform komponense.

* Pozíció: Az objektum helye az X, Y és Z koordinátákon
* Forgás: Az objektum X, Y és Z tengely körül forgása fokban mérve
* Méretarány: Az objektum méretaránya az X, Y és Z tengelyen, ahol az eredeti méret értéke 1.

**EndCredit**

Egy lista a játék végén mely bemutatja, hogy ki/kik dolgoztak az adott játékon

## **3.4** **Feladatspecifikáció, algoritmusok:**

#### ***3.4.1*** ***Karaktermozgás***

Ehhez két script is tartozik, MouseLook.cs illetve PlayerMovement.cs

**MouseLook.cs**

Az FPS játékok alapja, hogy az egér mozgásával fordul a kamera, azaz a karakter látószöge, valamint maga a karakter is. Ez a script ezt a fajta mozgást teszi lehetővé.

Egy float típusú publikus változót létrehoztam mouseSensitivity néven, mely kap egy értéket 1-100 között, ez befolyásolja az egérnek az érzékenységét. Ezt a menüben lehet egy slideren állítani a játékban, a játékos kedve és kényelme szerint. Létrehoztam egy a unitybe előre beépített Transform osztály példányát playerBody néven, valamint még egy publikus float típusú változót xRotation néven. A Start függvényben lefuttatok egy Cursor.lockState = CursorLockMode.Locked-ot mely eltünteti a kurzort futtatáskor. Az Update függvényben létrehoztam két float típusú változót melyek mouseX és mouseY nevet kaptak. Ezek az Input.GetAxis(“Mouse X”) illetve Input.GetAxis(“Mouse Y”)-t kapják értékként melyek az engine beépített metódusai, ezek számolják a megtett dölési szöget az x illetve y tengelyen. Ezeket megszorzom a mouseSensitvity értékkel így megkapva az egér érzékenységét. Az xRotation változóba -= ként átadom a mouseY változó adatát, majd xRotationnek a Mathf.Clamp segítségével megadom, hogy az y tengelyen csak 90°-ig engedjen fordulni. transform.localRotation = Quaternion.Euler segítségével az y-tengelyt is képesek leszünk érdemben mozgatni az egérrel, tehát a kamera is fordul. Majd a playerBody.Rotate segítségével az x-tengely beállítjuk, hogy a test is forogjon ne csak a kamera.

**PlayerMovement.cs**

A mozgáshoz előre beépített komponenst, a CharaterControllert használom. Ebben alap dolgokat előre be lehet állítani, ezek a magassága ennek kontrollernek, hogy milyen meredek sík dolgokon képes felmenni, illetve, hogy egy adott lépcsőnél milyen magas lépcsőfokokat képes “megmászni” a karakter. Ebben a scriptben van megírva a karakter mozgása a WASD billentyűkkel, a karakter reakciója a környezettel és azok tényezőivel például a gravitáció, valamint a karakter ugrási képessége és guggolási képessége is. Guggolásnál lekicsinyül a CharacterController mérete, így az ellenfeleink kisebb felületen tudnak minket sebezni. Mivel a mozgás a fizikára alapszik így, magasról esésnél a karakter esési sebessége is a fizikával megegyező módon nő. Egy groundCheck nevű objektumot is létrehoztam, mely azért felel, hogy folyamatosan rávizsgáljon arra, hogy a karakter alatt van-e valamiféle layer. Ez segít az ugrásnál abban, hogy a karakter ne tudjon a levegőben tovább ugrálni, hanem csak akkor ha egyszer földet ért.

#### ***3.4.2*** ***Fegyverek, lövés és annak effektjei***

Ide is két scriptet tudok felsorolni, a WeaponSwitch.cs-t illetve gun.cs-t.

**WeaponSwitch.cs**

Ez a script felel a fegyverváltásért, alapértelmezetten az AR van a karakter kezében, emellett megtalálható még a pisztoly kezdésként a karakternél. Mindegyik fegyvernek, amelyet a karakter birtokolhat van egy indexe, ez az index az, amit visszaad a script, ezt lehet az egérgörgővel vagy az 123-mal módosítani. Ez biztosítja azt a játékban, hogy képes legyen a játékos ezekkel a billentyűkkel, illetve az egér görgőjével fegyvert cserélni. Alapvetően minden fegyver egyszerre a kezében van a játékosnak, így ez az indexelés segít abban is, hogy letiltsam azokat a fegyvereket, amelyek nincsenek kiválasztva, tehát, ha a pisztolyt választja a játékos a többit addig letiltja amíg a pisztoly van a kezében így a többi fegyver nem jelenik meg amíg rájuk nem vált a játékos.

**Gun.cs**

A gun.cs egy hosszú és összetett script, amit úgy készítettem el, hogy ne kelljen minden fegyverre külön megírni, paraméterekkel meg lehessen adni a különböző fegyvereknek a saját értékeit. Ez a script magába foglalja a sebzés értékét, fegyver hatótávját, a fegyver tűzgyorsaságát, az eltalált ellenfél hátra lökésének mértékét, a maximum töltény mennyiséget, azt, hogy egy tárban mennyi lehet a maximális töltény, illetve, hogy az adott tárban jelenleg mennyi töltény található. Továbbá megtalálható még az újra töltése a fegyvereknek, a töltények számának kiíratása a képernyőre ammobar formájában, valamint a képernyőre íratás ha kifogytál teljesen a lőszerből egy adott fegyvernél, egy fpsCam, ami arra szolgál, hogy a lövedékek a képernyő közepe felé menjenek, ahol a célkeresztünk is van. Itt található még a muzzleFlash illetve impactEffect, melyek arra szolgálnak, hogy a fegyver elsütésekor és a golyó becsapódásakor egy-egy villanás legyen, ezzel imitálva azt, hogy a fegyver valóban elsült, illetve, hogy a lövedék célba ért. Az újra töltés korutinnal (A korutin egy olyan function mely képes félbeszakítani a saját végrehajtását előre megadott időre.) történik, ami abban segít, hogy a játékos ne legyen képes lőni az újra töltés alatt amíg az újra töltési animáció lefut. A karakter a fegyvert csak akkor képes újra tölteni ha az aktuális tárból fogyott töltény, tehát lőtt a játékos, ekkor az R billentyűvel újra tudjuk tölteni a fegyvert. A fegyver automatikusan újra tölt, ha az aktuális tárból kifogyott, illetve van még nála lőszer. Lövéskor a Target osztályt hívja meg, és rávizsgál, hogy az eltalált játékobjektum rendelkezik-e a Target.cs scripttel. Ha rendelkezik akkor meghívódik benne a TakeDamage függvény melynek a damage paramétert adjuk át. Rávizsgál arra is, hogy az adott objektumon van-e rigidbody komponens, ha van rajta akkor egy hátralökés funkciót hajt végre.

#### ***3.4.3*** ***Életerő és ellenfelek***

HealthBar.cs script tartozik ehhez, valamint a Target.cs.

**HealthBar.cs**

Ez a script tartalmazza a healthbar-t amely mutatja, hogy karakterünknek vagy egy ellenfélnek éppen mennyi életereje van. Ez egy slider segítségével van megvalósítva, ami játék indításakor mindig maximalizálva van, tehát 100 életerőpontja van a karakterünknek, az ellenfeleknek pedig típustól függően a maximális. Ha sebzést kap a karakter a healthbar vagyis a karakterünk élete csökken, valamint a zöldből átmenettel elkezd pirosra váltani ezzel felhívva a játékos figyelmét, hogy mennyire van veszélyben.

**PlayerHealthbar.cs**

A PlayerHealthbar.cs tartalma maga a karakterünk életerejének pontja, amit átad a HealthBar.cs-nek. Amíg a HealthBar.cs csak a vizuális megjelenítésre alkalmas itt történnek a tényleges számolások melyek alapján állítja a vizuális megjelenítést. A karakterünknek sebzés után 10 másodperccel, folyamatosan elkezd visszatöltődni az életereje amíg el nem éri a maximumot, ha a 10 másodperc nem telik el az újabb találat között akkor a számlálás újraindul.

**Target.cs**

Ebben a scriptben van megírva az ellenfelek életereje, valamint megölése és annak animációja. Ez a script paraméterekkel dolgozik, így minden külön ellenfélnek különböző nagyságú életerőpontot tudok adni. Itt létrehoztam egy függvényt TakeDamage néven és adtam neki egy amount paramétert, amely a fegyver sebzésének értékét veszi át. Ebben a függvényben az életerő annyival csökken, amennyi a sebzés értéke, ha az életerő eléri a nullát akkor meghívódik a Die függvény melyben lefut a halál animáció, majd törlődik a gameObject.

#### ***3.4.4*** ***Menü***

A MainMenu.cs sorolható ide, illetve a PauseMenu.cs.

**MainMenu.cs**

Ez a script felel a főmenü vezérléséért, a Játék gombra nyomva a menüben meghívódik a PlayGame függvény, amely betölti a következő indexű Scene-t. A Kilépés-re kattintva pedig a QuitGame függvényt hívja meg mely lefuttatja az Application.Quit()-et és ezzel bezárja a programot.

**PauseMenu.cs**

A PauseMenu.cs felel a játékon belüli pause menüért, ami tartalmazza a Játék folytatása, Beállítások, Kilépés a főmenübe, illetve a Kilépés a játékból opciókat. Ezt a játékos az ESC billentyű megnyomásával éri el a játékban, ilyenkor a kurzor ismét láthatóvá válik a Cursor.lockState = CursorLockMode.None segítségével. Létrehoztam egy bool típusú változót GameIsPaused néven melynek alapból false értéket adtam. Létrehoztam egy Resume és egy Pause nevű függvényt melyekben a a PauseMenu UI-ját aktiválom vagy deaktiválom, a GameIsPaused változónak értéket állítok Resume függvényben false-t Pause függvényben true-t, valamint a timeScale-t a Pause függvényben 0-ra, a Resume függvényben 1-re állítom. Ezeket az Update függvényben meghívom rávizsgálva arra hogy az ESC gomb megnyomásra került-e, ezen belül pedig hogy a GameIsPaused-nak milyen értéke van, ha true akkor a Resume függvény fut le, ha false akkor pedig a Pause függvény. A Játék folytatására kattintva szintén a Resume függvény fut le. A Kilépés a főmenübe opcióra nyomva lefut a BackToMainMenu függvény amely betölti a 0. indexű Scene-t vagyis a főmenüt. A Kilépés a játékból opcióra kattintva hívódik meg a QuitGame függvény mellyel bezárja a programot.

# **4** **Tesztelési dokumentáció**

Minden script megírása után teszteltem a játékot, hogy az éppen aktuálisan megírt újdonság a játékban rendben funkcionál-e. A lövés mechanizmusnál a találatot konzolosan kiiratom, így visszajelzés is jön az eltalált objektumról a fejlesztői környezetbe. Logikai bukkanók akadtak néhányszor a fejlesztés során, ilyen volt a töltényszám helyes menedzselése újra töltéskor, melyet néhányszor újra kellett gondolnom és írnom.

A játékom, “béta” tesztelésen is átesett, mely azt takarja, hogy egy közel kész, kibuildelt verziót átküldtem néhány osztálytársamnak, hogy próbálják ki, valamint tájékoztassanak az esetleges hibákról. Ezzel olyan hibák kiküszöbölése is sikerült, melyek nálam nem jelentek meg, vagy nem tudtam elérni.

**Az elvégzett használati tesztek**

*Mozgás, irányítás*  
A karakterünk irányításának ellenőrzése. A WASD, Space, Ctrl gombok, illetve az egér reagálása.

*Falakba ütközés*  
Ellenőriztem, hogy a karakter nem-e akad be semmilyen tereptárgyba, illetve, hogy nem tud átmenni a falakon.

*Menürendszer*  
Ellenőriztem, hogy a gombok megfelelő helyre vezetnek, illetve, hogy a pause menü megállítja-e az időt játékon belül. (Time.timeScale=0f;)

*Sebzés*  
Ellenőriztem, hogy a sebzés a fegyveren előre beállított sebzéssel megegyezik-e. Ellenőriztem, hogy a fegyverek tűzgyorsasága optimális-e az adott fegyvertípusokhoz, valamint azt is, hogy milyen távolságból képesek eltalálni az ellenfeleket.

*Ellenfelek*  
Ellenőriztem az ellenfelek detektálási képességét, életerejük mennyiségét, sebzésük mértékét, és lövéseik között eltelt időt. Továbbá ellenőriztem hogy a NavMeshAgent-ek alatt a talaj megfelelően van bake-elve, ezáltal megfelelően tudnak mozogni a pályán.

# **5** **Továbbfejlesztési lehetőségek**

Továbbfejlesztési lehetőségnek látnám ebben a játékban a multiplayer létrehozását, melyhez különálló pályák tartoznának. Ezeken a játékosok összemérhetnék tudásukat egymás ellen. Ahhoz, hogy ez megvalósítható legyen kell egy szerver kapcsolatot létesíteni a játékosoknak, hogy online játszhassanak. Emellett a játékosok karaktereinek hitboxát (A karakter körül lévő váz, mely regisztrálja a találatokat) átkellene alakítani, finomítani teljes mértékben, hogy a sebzés értéke testrészenként változzon.

A történet kibővítés is nagy lehetőség a későbbiekben ebben a játékban, új fajta ellenfelek behozása, akik nagyobb fegyverténnyel bírnak. A pályák kibővítését is jó opciónak tartom, majd új pályarészek létrehozását.

A játék továbbfejlesztését csak úgy tudom elképzelni, hogyha az csapatmunkában és nem önálló munkában folytatódik. Legalább 3-4 fős team-et hívnék össze, ahol mindenki elvállal egy részt a fejlesztésben. Gondolok itt a modellezésre, kódolásra, level design-ra. Ha úgy látjuk, hogy a végtermék jól sikerülne, akkor piacra is dobnánk.

A multiplayer egy másik ágán is tovább lehetne fejleszteni a játékot, ez pedig a cooperative ág (maximum 4 játékos összedolgozva játssza a játékot, és így együtt haladnak a történettel) mely elég érdekes és bevonzó lenne.

# **6** **Összegzés**

Véleményem szerint az elképzeléseim javarészét sikerült megvalósítani, amelyeket témaválasztás során kitűztem magamnak. Elképzelésem szerint olyan, hogy teljesen kész legyen valami, ebben a szakmában nem létezik. Mindig van olyan dolog, amin lehet javítani, továbbfejleszteni. Ha az ember folyamatosan benne van ebben az úgynevezett mókuskerékben, mindig van olyan rész, ahol tud egyszerűsíteni.

A játék megírása során jöttem rá, hogy az informatikán belüli játékfejlesztés mennyire szerteágazó, hogy milyen sok szakember kell egy komplett kész projekt létrehozásához. A kóderek, akik csak a szükséges scriptek kódjait írják, a grafikusok, akik lemodellezik, valamint meganimálják az összes játékban lévő objektumot, és a level designer-ek, akik felépítik a grafikusok által előre legyártott elemekből a játékot. Sok apró részletes munkából áll össze a végén a kép.

A játék készítése számomra elég nehézkesen indult háttértudás nélkül, viszont a belefektetett energia kifizetődött, ugyanis bár hosszú kutatómunkák során, de rengeteg új ismerettel gazdagodtam és ez a fejlődésemet szolgálta.

# **7** **Forrásmegjelölés**

<https://docs.unity3d.com/ScriptReference/index.html>

<https://www.youtube.com/user/Brackeys>

<https://hu.wikipedia.org/wiki/Unity_(j%C3%A1t%C3%A9kmotor>)

<https://www.youtube.com/channel/UCRMXHQ2rJ9_0CHS7mhL7erg>

[https://www.youtube.com/channel/UCNJvwJ6daLmw4\_gUKTw4cSg https://www.mixamo.com/](https://www.youtube.com/channel/UCNJvwJ6daLmw4_gUKTw4cSg)