Matematikai és Informatikai Intézet

Mesterséges Intelligencia a Számítógépes Játékokban

Készítette
Kardos Zsolt
Programtervező Informatikus

Témavezető
Dr. Kovásznai Gergely
Egyetemi Docens

Eger, 2023

Tartalomjegyzék

Bevezetés

Projekt elérhetősége: `https://github.com/O48WRX/Szakdoga`

Már gyermekkorom óta érdekelt a programok működése és mint sok más korombelit a számítógépes játékok is felkeltették a figyelmem. Sokat köszönhetek a videojátékoknak, az Angol nyelvi tudásom és részben, mint előbb is említettem, a programok és ezáltal a programozás iránti érdeklődésem is.

Gimnáziumban már, mint Informatika szakos diák lényegesebben többet tudtam meg az informatikáról mint szakma, köszönhetően az egyetem programjainak és az órán tanult anyagok miatt. Itt már sikerült alapszinten elsajátítani a programozás alapjait, és már *imperatívan*[1] programoztunk *FreePascal*[2]-ban Pascal nyelven, valamint alapvető szinten *C#*-al is foglalkoztunk. *Pascal*-ban megpróbáltam egy *Text-based adventure game*-et létrehozni. „Egy *Text-based adventure game* egy olyan szimuláció, ahol egy 'Agent' kizárolag a természetes nyelv használatával navigálja a játékmenetet."[1]

Az egyetemen eltöltött féléveim alatt már nagyon sok technológiával megismerkedetem. Operációs rendszerek működése és működtetésével, *OOP programozás*[3], a játékok szempontjából fontos számítógépes grafikát és a programozás alapköveit az adatszerkezeteket és jellegzetes algoritmusokat, és még sok mással. Legfontosabbnak az *OOP* programozási tudást tartom, melyet folyamatosan elsajátítottam az egyetem kurzusai alatt, és a szakdolgozatom elkészítésében is lényegesen segített ez a tudás.

A számítógépes játékok már több mint két évtizede az aranykorukat élik, mind eladások és a körülöttük kialakult kultusz miatt. Egyre több ember játszik számítógépes játékokkal, telefonos játékokkal, így a játékfejlesztés több mint releváns téma a mai világban.

Az asztali számítógépes játékok már a 2000-es évek óta uralják a játékoknál a teljesítményt és kinézet kombinációját. A konzolok továbbra is limitáltak a teljesítményükben és általában visszahúzzák a játékok fejlesztését a hardver szabványos konzolok miatt, a játékok így is jól néznek ki, de tény, hogy limitáltak a hardver által, míg a tisztán PC-kre fejlesztett játékok általában jobban néznek ki és az ajánlott hardvereken jobban is futnak.

[1] Az imperatív programozás egy programozási paradigma melyben, a program állapotát változtatni tudjuk egymás után leírt parancsok soraival.

[2] A FreePascal egy fejlesztői környezet a Pascal nyelv számára.

[3] Az OOP programozás, Objektum-orientált programozás, programozási paradigma.

A két dimenziós játékok pedig mind a PC és konzol piacon is jelen vannak. Ilyen címekkel mint a Super Mario, Shovel Knight, Binding of Isaac, Cuphead és még sok más. Mára már inkább az Indie fejlesztők csinálnak két dimenziós játékokat, de sikerük továbbra is tagadhatatlan, az egyik legfrissebb ilyen szenzáció a Cuphead, amivel nagyon sokan játszottak, amikor kijött.

Egyik hallgatótársammal, Riczkó Henrikkel úgy döntöttünk, hogy közösen szeretnénk létrehozni egy működő játékot és hozzá tartozó mesterséges intelligenciát és gépi tanuló algoritmust is.

Mivel közösen dolgozunk a témán így fel kellett osztanunk, hogy ki mivel fog foglalkozni a projektben:

– Riczkó Henrik – Mesterséges Intelligencia, Gépi Tanulás

– Kardos Zsolt – Game Design

Azért választottuk ezt a témát, mert a szétosztott fele a témának kettőnk számára nagyon érdekes. Mindkettőnket kiskorunk óta érdekeltek a számítógépes játékok és belső működésük és, hogy mi tesz egy jó játékot jó játékká. Maga a játékok fejlesztése és a hozzá tartozó mesterséges intelligencia és gépi tanulás nagyon aktuális.

A témán belül is azért választottam két dimenziós játékot, hogy segítsem a hallgatótársam is, és mivel mindkettőnkhöz közel áll ez a játékműfaj.

1. fejezet

Játékfejlesztés háttere és technológiái

1.1. Játékfejlesztés történelme

A számítógépes játékok megszületése egy várható végkifejlet volt, mivel, már rég óta próbálkoztunk a már meglévő játékok, mint például a Sakk, automatizálásával.

Siyuan Xu cikke szerint, az első számítógépes játék 1958-ban vált működővé, a neve *"Tennis for Two"*[1], amit *William Higinbotham*[2] hozott létre. Működése egy analóg számítógép és egy oszcilloszkóp segítségével történt.

Az első konkrét számítógépen futó videojáték, a *Spacewar*. Megjelenésére 1962-ben került sor, *Steve Russel MIT*[3] tag projektjeként. Azért mondjuk, hogy az első tényleges számítógépes játék, mivel egy PDP-1 számítógépen készült.[3]

1.2. Videojátékok aktualitása

A 20. század végén és a 21. század elején terjedtek el a számítógépes játékok, mint egy főbb szórakoztató médium. A '90-es évek és a 2000-es évek mondhatóak a számítógépes játékok technológiai égbetörésének kezdetének és napjainkig folyik ez a folyamat.

Hatalmas fogyasztópiacra tettek szert a számítógépes játékok és ezen a piacon nagyobb hanyatlás nem figyelhető meg. Szinte folyamatos igény van az újabb játékokra, mivel egyre többen választják ezt a médiumot.

A PopCap 2011-es kutatása szerint jelentősen gyarapodott a játékokkal játszó emberek száma. Az internet használók 42%a játszott már valamilyen közösségi játékokkal, míg az 1 évvel azelőtti mérésben, ez még csak 28% volt.[2]

[1] A Tennis for Two egy tenisz, vagy asztalitenisz szimulátor.

[2] William Alfred Higinbotham egy amerikai fizikus, aki az első atombombát kifejlesztő csapat része volt.

[3] Massachusetts Institute of Technology, műszaki egyetem

1.3. Játékok és technológiai hozzájárulásaik

Ilyen címekkel indult, mint a *Super Mario*, ami mondhatni, hogy definiálta a két dimenziós platformer játékok sztenderdjét és az egyik inspiráló tényező volt az én projektem elkészítésére is, mivel a Super Mario játéksorozat egy jelentős részét tette ki a gyermekkoromnak. Továbbá a *Wolfenstein 3D*, ami mondható a modern *FPS*[4] játékok atyjának és a hasonlóan klasszikus és kultusz építő *Doom* sorozat első kiadása is.

Az előbb említettek is már grafikailag lenyűgözőek a korukhoz képest, de az első ténylegesen lenyűgöző az a *Quake* nevű játék volt, ami már tényleges három dimenziós modelleket használt. Valamint egy legendás kódrészlet is létrejött a Quake sorozat fejlesztése során. Ez mégpedig a *Fast Inverse Square Root algoritmus* Ez az algoritmus az 1999-ben kiadott, a játéksorozat harmadik kiadása a *Quake III Arena* játék kódjában található.

1.1. ábra. Quake III Arena, kép forrás: id Software[10]

1.3.1. Fast Inverse Square Root algoritmus

Miért van erre szükség?

Shaw: The Legendary Fast Inverse Square Root cikke így beszél az algoritmusról[4]: Az inverz négyzetgyök számítása egy nagyon fontos része a számítógépes játékok fejlesztésében.

- *Pathfinding*[5]
- Grafikus számítások (Fény, visszaverődések stb.)
- Egyéb vektornormalizációs műveletek, melyekhez szükséges a számításban.

[4] First-person shooter, belsőnézetes lövöldözős játékok
[5] A mesterséges intelligencia útkeresése.

```
1   float Q_rsqrt( float number )
2   {
3       long i;
4       float x2, y;
5       const float threehalfs = 1.5F;
6
7       x2 = number * 0.5F;
8       y  = number;
9       i  = * ( long * ) &y; // evil floating point bit
10      i  = 0x5f3759df - ( i >> 1 ); // what the ****?
11      y  = * ( float * ) &i;
12      y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st
13      //  y  = y * ( threehalfs - ( x2 * y * y ) ); //
14      // this can be removed
15
16      return y;
17  }
```

Listing 1.1. Fast Inverse Square root algoritmus, cenzúrázva

Grafikailag intenzív játékokban, pont mint a Quake III Arena, ezek a számítások másodpercenként akár milliószor is megtörténhetnek, így ezen a számításon optimizálást végrehajtva hatalmas teljesítményjavulást tudunk elérni.

De hogyan is csinálták az *id Tech 3* játékmotor programozói, hogy felgyorsították az időben drága inverz négyzetgyökszámítást? Implementáltak egy nagyon gyors és pontos közelítést erre. Lényegében egy *Newton-Raphson iteráció* alapú közelítésről beszélünk.

1.1. Definíció. Newton-Raphson gyökkeresés: Megkapjuk egy szám közelítését, és egy jobb közelítést érünk el:

$y = c/2 * (3^{-}xc^2)$

A Fast Inverse Square root pedig az iterációt egy nagyon okos tippel kezdi. A kód megtekinthető itt[7].

Ez a kód jelentős, és legendás státuszt ért el a játékfejlesztés történelmében. „Fekete mágiának" tartott géniuszi bitmanipulációja és technológiai jelentősége miatt kapott hírnevet. A régebbi processzorok számára ez hatalmas teljesítmény növekedés volt, ám később, 1999-ben az *Intel* által kiadott processzorok számára már elavulttá vált ez a művelet. Újabb műveletek már az idő tizede alatt sokkal kevesebb hibával is el tudták ezt érni. Pontosabb számítások esetére is találtak megoldást ami a *FISR* algoritmus sebességének 80%-ával el tudott érni $< 0{,}01\%$-os hibaszázalékú megoldást.

1.3.2. Half-Life és a realizmus

A Half-Life sorozat mondhatni az egyik leglényegesebb, ha nem a leglényegesebb játéksorozat a ’90-es évek végéről és a 2000-es évek elejéről.

A realizmus pedig rég célja a játékoknak. Bár nem feltétlenül szükséges egy játéknak, hogy realisztikus legyen ahhoz, hogy élvezhető legyen, de lenyűgöző és bevonzó lehet egy játék minél hűebb a valósághoz.

Matthew Byrd cikke[5] alapján: Az első rész a sorozatban, szimplán Half-Life, 1998-ban jelent meg. Az elterjedt közös megegyezés az, hogy nagyon fejledt játék volt az idejéhez képest és sok innovációt hozott be, bár arról már kevesebbet beszélnek manapság.

A *Half-Life* volt az a játék a ’90-es években, ami lefektette az alapjait annak, hogy, hogyan definiálunk, mi tesz egy PC játékot PC játékká, emellett még egy olyan tapasztalatot nyújtott a játékosok számára, amit abban a korban nem sok fejlesztőcsapat tudott volna nyújtani.

1.2. ábra. Half-Life (1998), kép forrás: Valve[8]

Gránátok dobásától *játék modolhatóságig*[6] nagyon sok olyan fogalmat hozott be, amelyeket a mai napig használnak a modern játékokban.

Mai viszonylatban elég viccesnek tűnik, de a játék használati utasításában az van kiemelve realisztikus ténynek, hogy a játékban található fegyverek logikusan vannak elhelyezve és nem csak lebegnek, mint például a Quake játékokban általánosan.

Az emberi ellenséges entitások mesterséges intelligenciája már képes volt reagálni a gránátok jelenlétére és a saját életüket helyezték előre, míg az űrlények pedig nem futottak el a gránátok elől mert ugye nem tudják mi az.

[6] A modolhatóság annyit jelent, hogy felhasználói módosításokat adhatunk hozzá, akár sajátot vagy másét.

1.3.3. Half-Life és a map design

Előtte kiadott játékok, mint például az *Unreal*, *Doom*, *Hexen* igazából nem szenvedett rossz pályafelépítésekkel, vagyis *map design*-al, de nem igazán adták át a pályáik, hogy egy nagyobb világ részét képeznék.

A Half-Life ezt teljesítette pár szóra érdemes módon:

– A technikai oldalon: A Valve egy eléggé nagy világot teremtett, és ennek a töltése a háttérben történik. Ez azt jelentette, hogy a felhasználóknak jóval kevesebbet kellett töltőképernyők előtt ülniük. Így sokkal jobban bele tudta élni magát a játékos, hogy egy kutatási központot kell navigálnia, és nem egy tradicionális szintről szintre lépkedéses megoldásban történt, mint például a *Doom*-ban.

1.3. ábra. Half-Life (1998) Black Mesa kutatóközpont, kép forrás: Valve, giantbomb.com[**?**]

– A teljes innováció azonban túllő csak a technikai oldalon. A Half-Life csapata, úgy készítette el a pályákat, hogy úgy tűnjön, hogy a kutatási központ minden részének külön feladata, jelentősége van. Ez egy nagyon fontos pályadizjánolási pont, hogy hihető legyen, hogy az a pálya miért van ott, és hogy úgy érződjön, hogy igen, ennek ott kell lennie. Direkt úgy építették össze a pályákat, hogy hihető legyen, hogy az egy kutatási központ:

  - Laborok egymés mellett, amik hasonló anyagokat tanulmányoznak.
  - Pihenőszobák.
  - Raktárok, raktárterületek.

1.3.4. Half-Life modolhatósága

Fontos, hogy emellett a Half-Life másik nagy innovációja, a *modolhatóság*.

1.2. Definíció. Modolás: Modolás alatt modifikációra utalunk, ami a játékok értelmében pályák hozzáadását, teljesen új *kampányt*[7], új fegyvereket és ilyesmi új lelkes felhasználók által hozzáadott tartalmat értünk.

A Valve meghozta azt a döntést, hogy a közösség kezébe ad egy könnyebb hozzáférést a játéktartalom gyártásához a dizájnprogramokkal, amiket használtak a játék elkészítéséhez. Ilyen például a *Hammer*. A *Hammer* egy pályaszerkesztő-készítő program, mellyel GoldSource pályákat tudunk készíteni.

Később a Valve is, de több játékstúdió is rájött, hogy nem csak arra vannak a vásárlók, hogy megvegyék a játékot, hanem modolhatóság által még a játékban bent maradt hibákat is megoldhatják, amennyiben zavarja őket, nem csak textúrákat fognak kicserélni, hanem akár teljesen új tartalmat és tapasztalatokat is generálhatnak a játék alapján.

1.4. Játékmodifikációk jelentősége

A játékmodifikációk nagyon lényegesek a fejlesztők számára, mivel, ha valakit inspirál egy játék arra, hogy változtasson benne valamit, akkor rávezeti arra, hogy tanuljon játékfejlesztést valamilyen szinten.

Mellékhatásként, még a játék körüli közösséget is fenntartja, és élettelibbé tehetik akár évekig az ilyen modok, mivel így a játékosok a saját tetszésüknek megfelelővé tehetik vele a játékokat.

Általában sokkal könnyebb a fejlesztés, ha az ember olyan dologgal foglalkozik, amit szeret úgynevezett *modderek*, akiket fel is vettek az adott céghez dolgozni, például az eredeti *TF Software*-ből *Robin Walker és John Cook*, akik elkészítették eredetileg a Team Fortress mod-ot, még a *Quake*-hez, később a Team Fortress: Classic-ot GoldSource-ra, mégkésőbb a Team Fortress 2-t a Source motorra, az utóbbi kettőt már a Valve szárnyai alatt tették.

Az előbb említettet belevonva, sok olyan modder vagy modoló csapat van, akik átmentek arra, hogy felépítsék a saját különálló játékukat a modjuk alapján.

Ilyen például a *The Forgotten City* mod[13], ami már szimpla mod státuszában is sok díjra lett jelölve a közösség által, melynek készítője később egy stúdióval megalkotta a saját önálló játékát ugyanezen a néven.

Ezekbe a sikerekbe belegondolva, a játékmodifikációk gyártása nem tűnik annyira vesztegetett időnek, mivel sok játékfejlesztő cég erre, úgy figyel mint egy tehetségkutató, mint például a *Bethesda Softworks* a mérhetetlen mennyiségű modifikációkkal a

[7] Kampány alatt a játék fő sztorivonalát értjük.

1.4. ábra. The Forgotten City (2021), kép forrás: Modern Storyteller[14]

játéksorozataikra, akik *Todd Howard* szavaival élve, már több alkalommal vettek fel új dolgozókat a játékaikat modifikáló felhasználók közül, amennyiben nyilván megfeleltek a munkakör feltételeinek. Ezáltal mondható, hogy a játékmodifikációk gyártása lehet egy egyenes út a professzionális játékfejlesztésbe is.

1.5. Játékmotorok

1.3. Definíció. Játékmotor: A játékmotor lényegében egy keretrendszer a játékok fejlesztésére.

A játékmotorok ma már szerves részét képezik a játékfejlesztésnek, azonban régebben nem voltak külön játékmotorok fejlesztve. A játékmotorok mint, szó a '90-es években kezdett el terjedni, ennek a fő oka a *DOOM* játék fejlesztése és a *DOOM Engine*.

1.5.1. Játékmotorok feladata

A játékmotorok általános feladata az, hogy megkönnyítsék a játékfejlesztést a fejlesztők számára. Vannak komplett játékmotorok, amik a játékfejlesztés legnagyobb pontjait lefedik, de vannak specializálódottabb motorok, mint például grafikai motorok, ilyen az *Unreal*.

Általában úgy fejlesztik ki ezeket a motorokat, hogy alapból rendelkezzenek olyan modulokkal, amik nagyon hasznosak a játékfejlesztésnél. Például manapság már általában, ilyen modulok vannak beépítve:

- Fizikai rendszer (Gravitáció, ütközés stb.)
- Árnyékolási rendszer
- 2D, 3D grafikai rendszer
- Hangrendszer

Gyakran ezek funkcionalitása részben felül is írható, ezzel is egyedivé tehetőek a játékprojektek.

1.5.2. DOOM Engine

Henry Lowood cikke szerint:[6] A játék 1993-as kiadási dátuma előtt id Software kiadott egy olyan kijelentést, hogy a technológiájuk ki fogja terjeszteni az általunk képzelt limitáltságát a számítógépes játékoknak.

Ez a nyilatkozat hozott be egy új fogalmat, a „DOOM Engine" és az „engine" fogalmát. Ez az új fejlemény megformázta a játékfejlesztés jövőjét és újratervezte a játékfejlesztés iparát.

A játékmotor fogalom létrehozása mellett, *John Carmack*, vezető programfejlesztő az id Softwarenél, nem csak egy új szoftvert hozott létre, hanem ezzel a motorral, jól definiáltan, szét is választotta a belső működéseket, mint például a 3D megjelenítő rendszert, játéklogikát a kreatív tartalmaktól mint például a textúráktól.

A DOOM-motor C és Assembly nyelven íródott.

1.5.3. Unreal Engine

Az Unreal Engine (UE) egy 3D számítógépes grafikai játékmotor, melyet az *Epic Games* fejlesztett ki. Először a saját 1998-ban kiadott *Unreal* belsőnézetes lövöldözős játékukban mutattak be. Készítője: *Tim Sweeney* és C++-ban íródott.

1.5. ábra. Unreal (1998), kép forrás: Youtube: Methos és Epic Games[11]

Az Unreal Engine gyorsan kinőtte magát abból, hogy egy játékstúdió motorja, és ma már megszámlálhatatlan mennyiségű játék használja grafikai alapjaként a motort. Például:

- Mass Effect széria
- Life is Strange
- Dishonored

2023-ban már az 5.0-ás verzióról beszélünk az Unreal Engine 5-ről, mely képességéről már nehézség megmondani, hogy a valóságról beszélünk-e, vagy egy gép által renderelt virtuális valóságról.

1.6. ábra. Unreal Engine 5, kép forrás: Youtube: JSFILMZ és Epic Games[12]

1.5.4. Unity Engine

A Unity egy *cross-platform*[8] játékmotor, amit 2005-ben adott ki a Unity Technologies, lényegében arra, hogy „demokratizálja" a játékfejlesztést.

A Unity Engine egy teljesjogú játékmotor, több modulja is van, ami segíti a játékfejlesztést:

- Fizikai rendszer (Gravitáció, Collision, stb.)
- Hang, hang-event rendszer.
- Animációk, animáció recording.
- Felhasználói felület gyártás.
- Pályadizájn segítés
- Mesterséges intelligencia segítség (Navmesh generálás, stb.)

Hatalmas hírnévre tett szert ez a játékmotor, mivel elérhetőbbé tette a játékfejlesztést mindenki számára, még annak is, akinek elérhetetlen lett volna alapból és még meg is könnyítette. Már készen álló, de felülírható beépített moduljai sokban segítik a játékfejlesztés folyamatát.

2005-ben adták ki az 1.0-ás verziót, eredetileg egy Mac OS X játékmotorként. Jelölve is lett az Apple Design Awards díjazásra a legjobb grafikai használatért. Később kapott támogatást Windows-ra, Linuxra és böngészőkre is.

[8] A cross platform annyit jelent, hogy több platformon is futtatható/elérhető az alkalmazás.

A Unity alapjáraton használható két dimenziós és három dimenziós játékok fejlesztésére is, mivel mindkettőre ad akkomodáló modulokat.

A Unity 2.0 2007-ben jött ki és már erősebb támogatást adott a háromdimenziós játékok fejlesztésére, optimizált terraformálás, valós idejű dinamikus árnyékok és még sok más. Ez a verzió azért is fontos, mivel 2008-ban adta ki az Apple az Apple Store-t és a Unity pedig rohamtempóban támogatást adott ki rá. Ezzel elterjedt az *iOS* fejlesztők körében is.

Egy 2012-es kutatás szerint[7] a mobiljátékfejlesztés nagyon az iOS és Unity összeállítás irányába haladt. A fejlesztők, akiket megkérdeztek 94,6%/ban iOS-re fejlesztettek, ezzel az lett a legelterjedtebb platform, az android pedig második helyen 70,7%-ban. A játékmotorokat tekintve pedig a Unity lett a legelterjedtebb 53,1%-os fejlesztői aránnyal, második helyen pedig saját motort használtak 39,8%-al.

Fejlődése során, mivel alapelve az inkluzivitás a platformok felé, kibővítette a támogatott platformok listáját. Mára már mobil és pc támogatás mellett támogat webes, konzol és virtuális valóság platformokat is.

- Mobil
  - iOS
  - Android
  - tvOS
- Asztali
  - Microsoft Windows
  - Mac
  - Linux
- Web paltform: WebGL
- Konzol platformok
  - Playstation
  - Xbox
  - Nintendo Switch
  - Stadia

- VR,ER
  - Oculus
  - Playstation VR
  - ARCore
  - ARKit
  - Windows Mixed Reality
  - Magic Leap
  - SteamVR
  - Google Cardboard

2. fejezet

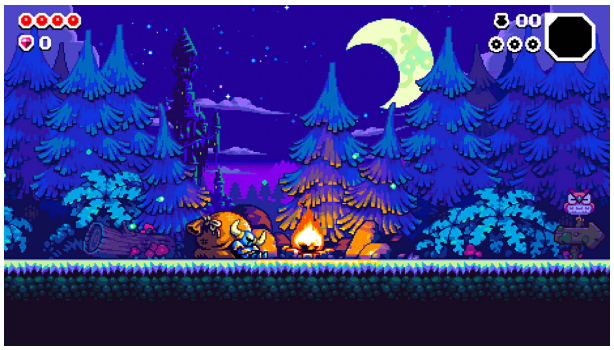
Unity Játékfejlesztés menete és kivitelezése

A téma kiválasztásánál fontosnak tartottuk, hogy a kettőnk feladata könnyen összeilleszthető legyen és, egy olyan játékmotort válasszunk, ami kellően támogatja mindkettőnk tevékenységét. Ezért választottuk a Unity-t.

Kiskorom óta szerettem a két dimenziós platformer játékokat, így biztosra tudtam, hogy azt szeretnék csinálni erre a projektre.

2.1. Tervezés

Először inspirációt gyűjtöttem más hasonló játékokból. Ilyen például a *Super Mario*, ami a számítógépes játékokba az egyik bevezető játékom volt, a második pedig a *Shovel Knight*



2.1. ábra. Shovel Knight (2014), Yacht Club games forrás: https://www.eurogamer.net/shovel-knight-dig-a-great-series-safe-in-the-hands-of-the-very-bests

A *Shovel Knight* a modernebb két dimenziós platformer[1], amely egész nagy sikereket

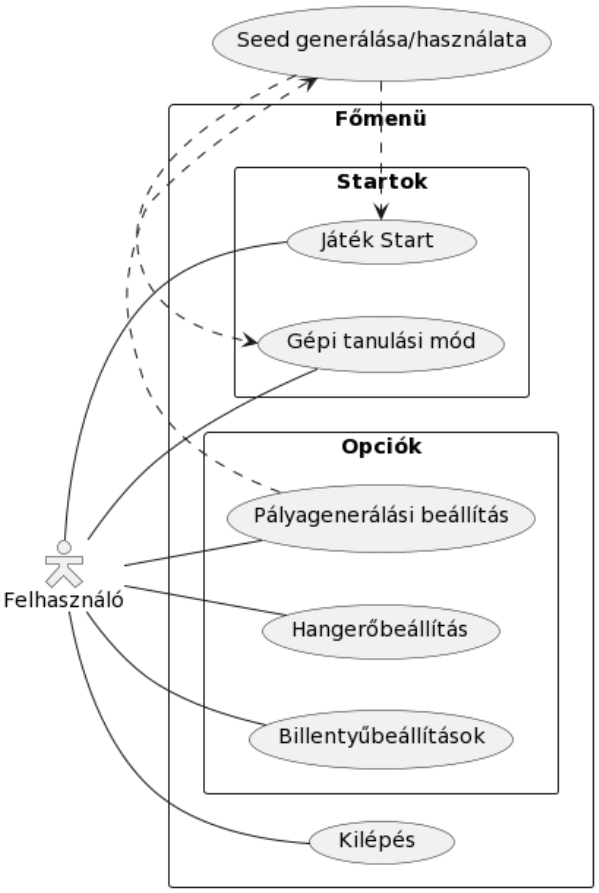
[1] A Platformer játékok, olyan játékok, melyek pályak pálya alapú manőverezési problémákat tesznek fel.

ért el a piacon.

Lényegében ez a két játék inspirált engem arra, hogy én is egy két dimenziós platformer játékot készítsek.

Az előző nyár alatt sokat kutattam a Unity játékfejlesztéssel kapcsolatban. Elveket a pályatervezéssel kapcsolatban, hogy milyen nehézségűnek kell a pálya bejárásának lenni, hogy élvezhető legyen. Hogyan kell a játékost mozgatni és a többi.

Először, a tervezéseink szerinti menüt terveztük ki.



2.2. ábra. Főmenü Use-case

Úgy terveztük, hogy eredetileg randomizált legyen a pályagenerálás, de ezt az ötletet eldobtuk. A játékot két módban lehet elindítani:

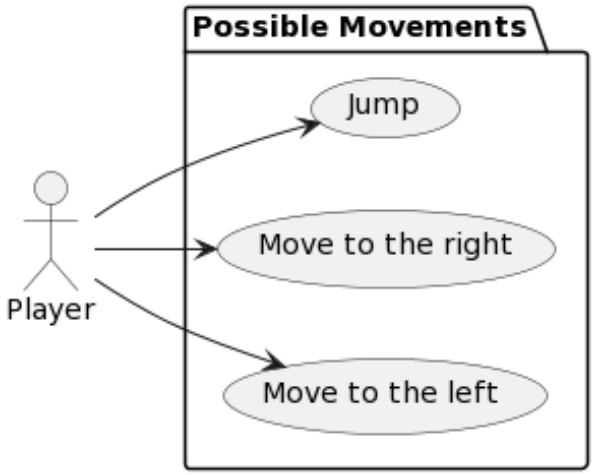
– Alapjáték – Játékos irányítja a karaktert.

– Gépi tanulási mód – Gépi tanulás algoritmus kezeli a karaktert.

Hangbeállításoknál a zene és zajok hangosságának átállítására van esélyünk. Billentyűbeállításoknál pedig a támadás és ugrás gombját tudjuk átállítani. A kilépés pedig szimplán bezárja az alkalmazást.

A főmenü megtervezése után pedig a játékos mozgatását kezdtem el kutatni, hogy, hogyan kellene megoldani.

Először megterveztem egy UML-t arról, hogy mit is szeretnék elérni. Alapjáraton, mivel egy két dimenziós játékról beszélünk, így balra, jobbra mozgást kell nyújtanunk, valamint ugrásra lehetőséget.



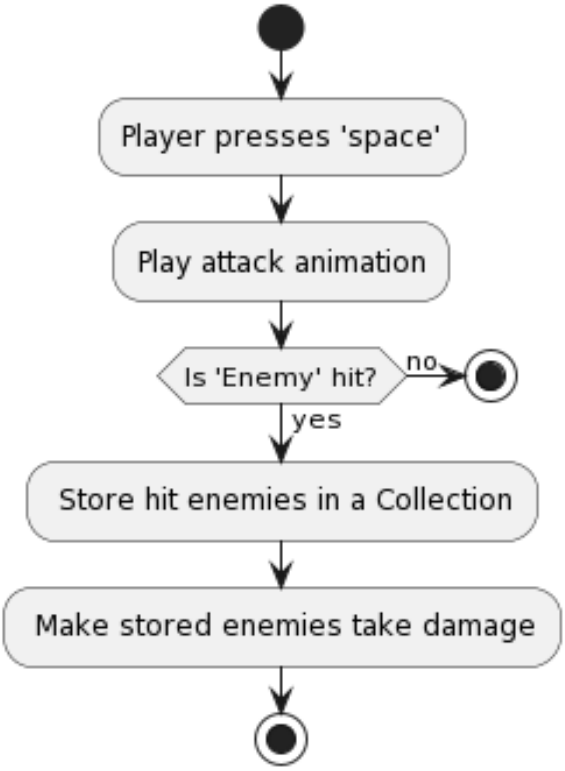
2.3. ábra. Use-case ábra a mozgásról való elképzelésről.

Ezek mellett természetesen le kell majd kezelnünk, olyan helyzeteket, mikor a játékos modelljét meg kell fordítanunk, hogy a haladás irányába legyen fordulva.

Ennek a segítségére felhasználtam *Pandemonium* Unity játékfejlesztési tutorialját használtam fel.[15] ettől a videótól teljesen megértettem, hogy, hogyan kellene megvalósítanom az eltervezett mozgást a játékos számára és le is kezelte a model megfordítási problémám.

A mozgások mellett szükség lenne persze az animációkra is a mozgásokhoz, szerencsére alapvető tudást megadta ez a videó is, és ebből képes voltam tovább építkezni az ötleten.

Lényeges alapja lenne a játékomnak a támadás is, és erre is keresnem kellett egy jó megoldást. Én *Brackeys* videóját találtam a legtalálóbbnak és számomra szimpatikus megoldásnak.[16] A videó informatívan és követhetően magyarázza el a mögötte álló technológiát, és, hogy azt, hogyan kell használni általánosságban. Kellően jó megoldást adott a tipikus hibákra mint több ellenfél eltalálásának a működése, és hogy alapjáraton mivel detektáljuk, hogy eltaláltuk-e az ellenséget. Ezáltal el is tudtam készíteni egy aktivitás diagramot, hogy, hogyan szeretném, hogy működjön a játékos támadása.



2.4. ábra. Aktivitás diagram a játékos támadásáról.

Felvehető tárgyakat is terveztem, mivel ezzel érdekesebbé is tudjuk tenni a játékot a játékos számára.

Tesztelésképp az elején, csak egy olyan tárgyat tervezek, ami csak visszadob a pálya elejére. Később viszont, már olyanokat is tervezek, amiktől a játékos többet fog sebezni, több életereje lesz és esetleg egy értéket képező felvehető tárgy, ami növeli a játékos pontszámát. A játékos pontszáma pedig, csak annyi funkciót lát el, hogy visszajelzés a játékosnak, hogy mennyire fedezte fel a pályákat.

Fontos része a tervezésnek még, hogy össze kellett gyűjteni olyan „asset"-eket, amelyeket használni fogok a játék fejlesztésekor. Mikor ezeket összegyűjtöttem, figyeltem arra, hogy kizárólag ingyenesen használható tartalmakat használjak fel.

Erre a *Unity Asset Store*-t használtam fel. Ez a Unity saját *asset* megosztó platformja, ahol a készítők akár értékesíthetik az általuk gyártott tartalmakat, melyek felhasználhatók a játékok fejlesztéséhez.

A keresett asset-ek főleg:

- Protagonista modell
- Ellenség modell
- Textúrák
  - *Skybox*[2]
  - Talaj, falak
  - Opcionális szépítések.
- Felhasználói felület

2.2. Verziókövetés

A Unity projektek számára az ajánlott a *Unity Version Control*, régebbi nevén *Plastic SCM*. Mi azonban, egy különlegesen erre összerakott *gitignore* fájllal ellátott *Git*-et használtunk, projektünket *GitHub*-ra tettük fel, ahol publikusan elérhető *repository*-ban elérhető: `https://github.com/048WRX/Szakdoga`

Azért választottuk a Git és GitHub kombinációt, mivel ez számunkra a megszokott, és mindenféleképpen a zökkenőmentes, stabil együttműködést szerettük volna elérni.

A Git nagy előnye, hogy könnyen kezelhető terminálból, de nagyon sok *IDE* [3] alapjáraton támogatja a Git-et.

Tudja segíteni a csapatmunkát azzal is, hogy beépített feladatkövető rendszere is van, mellyel folytonossá és könnyűvé teszi a csoportos munkákat.

2.3. Csapatmunka

Programfejlesztők folyamatosan csoportos munkával fognak találkozni a munkahelyen, így nagyon örültem a lehetőségnek, hogy a konzulensem mellett csapatként is dolgozhatok hallgatótársammal ezen a témán.

Egyetemen is már többször kellett csoportos munkában részt vennünk, de ebben az esetben ez a téma túlfed egy félévet a súlyával, így teljesen kiteljesedve tudunk dolgozni a témán, érezve a végeredmény komolyságát is.

Nagyon fontos csapatmunkánál a kommunikáció, így törekedtünk arra, hogy legalább hetente kétszer konzultáljunk egymással a helyzet állásáról.

Első elképzelésekkel ellentétben nehezebbnek bizonyult a csapatmunka, mivel még nem dolgoztunk ekkora projekten, így sok személyes faktorhoz hozzá kellett szoknunk.

[2] Az eget szemléltető kép.

[3] Integrated Developer Environment, integrált fejlesztési környezet.

2.4. Csapatbeli kommunikáció

Mivel csoportos munkánál a kommunikáció kifejezetten fontos szerepet kap, így mi is nagy fókuszt raktunk erre a részére a csapatmunkának.

Gyakran téma volt nálunk személyesen is, amikor találkoztunk egyetemen vagy egyéb eseményeknél is, hogy, hogyan kellene továbbfejleszteni a projektünket. A tervezés nagy része viszont, főleg a *Discord* alkalmazáson történt, vagy a *Meta Messenger* alkalmazásán történt.

Messengeren a kisebb változásokat beszéltük meg, vagy kisebb betervezett *feature*-öket. A főbb tervezések Discord-on történtek, ott beszéltük meg az eredeti műfaját a játéknak és, hogy milyen funkciókat fog tartalmazni például.

2.5. Feladatok megosztása

A tervezésünk egyik legfontosabb része volt, mivel arra kellett odafigyelnünk, hogy fair módon osszuk el egymás között a feladatokat. Mivel ketten voltunk egészen könnyen el tudtuk felezni a felelősségeket egymás között.

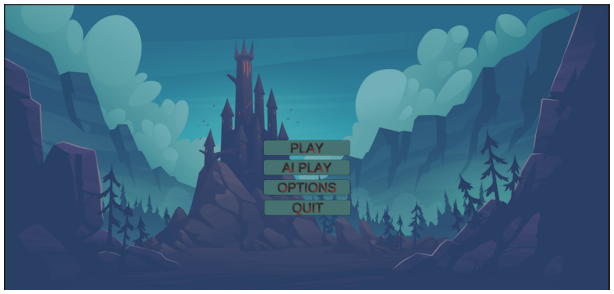
Mivel egészen hamar eldöntöttük, hogy ki melyik részét fogja csinálni a projektnek, én magát a játékot, Henrik pedig a mesterséges intelligenciát és a gépi tanuló algoritmust, így ezzel a részével nem kellett küzdenünk.

Viszont a nehézségek a feladatok heti leosztásánál voltak, mivel ebbe külső hatások is belekavartak, mint a családi és egyéb programok és maga az egyetem is. Ezért gondoltam szükségesnek, hogy megemlítem ezt a részét is a fejlesztésnek.

2.6. Fejlesztés

2.6.1. Főmenü

A legelső amit elkészítettem aza főmenü volt. Elemi része a játékoknak a főmenü, hogy el lehessen indítani a játékot, tetszőleges beállításokat végezni a játék működésén vagy működtetésén.



2.5. ábra. A játék főmenüje.

Fő feladat az volt a menünél, hogy ez legyen az első *scene*[4] és itt lehessen beállítani a hangerőt, valamint, hogy az indítógombbal az első pályára kerüljünk át. Ezt egy szimpla sor kóddal el lehet érni:

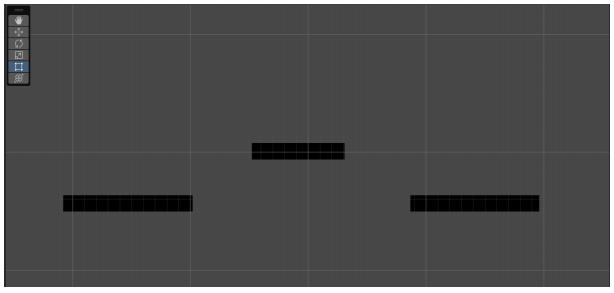
```
1 public void playGame()
2 {
3         SceneManager.LoadScene(SceneManager.GetActiveScene().
              buildIndex + 1);
4 }
```

Listing 2.1. A játék Scene léptető kódsora.

2.6.2. Platformok

A főmenü után, első feladatom az volt, hogy előállítsak egy navigálható tesztpályát. Először *platformokat*[5] kellett létrehoznom. Erre felhasználtam a Unity-be beépített *Square* objectet, melyet transzformálva elértem a szükséges kinézetet. Ezeknek a platformoknak a pozíciója abszolút és nem megváltoztatható a játékos által. A transzformálására, a beépített *Transform* komponenst használtam fel, láthatóságára/színezésére pedig a *Sprite Renderer* komponenst. Itt még fennállt az a probléma, hogy a játékos átesett az objektumon, ezért még a *Box Collider 2D*-t is felhasználtam, ezzel kiküszöböltem a játékos és egyéb *Actor*-ök átesését.

A *Box Collider 2D* az egyik legfontosabb eszköz, amit használtam a játék fejlesztése során. Megoldja, hogy a karakterek ne tudjanak átesni a *Collision Box*-án, ami skálázható méretű, valamint a *Material* property-jével megoldható az *Actor*-ök falon való fennakadása. Ez úgy történik, hogy mind a platformok és a játékos és az NPC-k[6] is el vannak látva *Collision Box*okkal, skálázva a modellekhez, és a kettő doboz nem tud átesni egymáson.



2.6. ábra. Az első tesztpályán (TestMap01) lévő platformok

[4] A scenek, az, az jelenetek, a Unity-ben az elkészített pályák technikailag, amik között váltogatni tudunk.
[5] A navigálható felületeket, amivel ütközik a játékos.
[6] Non-Player Character, nem a játékos által irányított karakterek.

2.7. Játékos

A játék egyik legfontosabb része, minden bizonnyal, hogy legyen egy játékos által irányítható objektum, amivel játszhatja a játékot. Alapjáraton, egy a platformokhoz hasonlító fekete négyzet volt a játszható karakter.

2.7.1. Mozgás

A játékos mozgása elemi része az ilyen játékoknak, mivel az élvezhetőségének az egyik alapja. Legelőször a gravitációt kellett implementálni.

Gravitáció

A gravitációt, úgy oldottam meg, hogy a játékos objektumnak adtam, egy *Box Collider 2D*-t, hogy ne essen át a platformokon, ezután felhasználtam a *Rigidbody 2D* komponenst, melyben a „Body Type"-ot dinamikusra állítjuk és ez alapjáraton rendelkezik egy „Gravity Scale" property, mely alapjáraton ellátja az objektumunkat egy Y tengelyen való mozgással.

Kétirányú mozgás

A kétirányú mozgás, a futás szintén egy alapja, amihez már az X-tengelyen való mozgatás szükséges.

Emellett még a modelltranszformáció is egy probléma amit meg kellett oldani. Ez annyit jelent, hogy az adott irányba való mozgásnál, az *Actor*-ök modeljét az adott irányba kell elforgatni.

Ugrás

Az ugrás szintén, mivel platformer játékról beszélünk, alapköve a műfajnak, hogy a nehezebben elérhető, esetlegesen elrejtett részekre is el tudjon érni a játékos.

Ezt a mozdulatot már az Y-tengelyen kell végezni, és egy folyamatosan csökkenő sebesség alapján kell számítani. Amikor az adott sebesség pedig eléri a 0-át, akkor pedig a beépített gravitáció vissza fogja húzni az objektumot.

3. fejezet

Tesztelés

Irodalomjegyzék

[1] Sahith Dambekodi, Spencer Frazier, Prithviraj Ammanabrolu, Mark O. Riedl, Cornell University, arXiv:2012.02757: *Playing Text-Based Games with Common Sense*, 2020.12.04

[2] infostart.hu, PopCap:*https://infostart.hu/eletmod/2011/11/27/kik-jatszanak-a-szamitogepes-kozossegi-jatekokkal-468037*

[3] Siyuan Xu: *History of AI design in video games and its development in RTS games https://sites.google.com/site/myangelcafe/articles/history_ai*

[4] Shaw:*The Legendary Fast Inverse Square Root, https://medium.com/hard-mode/the-legendary-fast-inverse-square-root-e51fee3b49d9*, 2017.09.17

[5] Matthew Byrd:*Half-Life: The Forgotten Innovations of the PC Shooter, https://www.denofgeek.com/games/half-life-innovations-history/*, 2020.02.12

[6] Henry Lowood, Kinephanos, Stanford Libraries, Palo Alto:*Game Engines and Game History, https://www.kinephanos.ca/2014/game-engines-and-game-history/*

[7] gamedeveloper.com, Gamasutra:*Mobile game developer survey leans heavily toward iOS, Unity, https://www.gamedeveloper.com/audio/mobile-game-developer-survey-leans-heavily-toward-ios-unity*, 2012.05.24

[8] Valve:*Half-Life (1998) kép, https://store.steampowered.com/app/70/HalfLife/?l=hungarian*

[9] Valve:*Half-Life (1998) kép, https://www.giantbomb.com/black-mesa-research-facility/3035-64/*

[10] id Software:*Quake III Arena (1999) kép, https://store.steampowered.com/app/2200/Quake_III_Arena/?l=hungarian*

[11] Epic Games, Methos:*Unreal (1998) kép, https://www.youtube.com/watch?v=WTRmZ6aFE*

[12] Epic Games, JSFILMZ:*Unreal Engine 5 kép, https://www.youtube.com/watch?v=1uXzaY749Ag*

[13] TheModernStoryteller: *The Forgotten City, https://www.nexusmods.com/skyrimspecialedition/mods/1179*, 2016.10.29

[14] Modern Storyteller, Dear Villagers: *The Forgotten City kép (2021), https://store.steampowered.com/app/874260/The_Forgotten_City/*

[15] Pandemonium: *Unity 2D Platformer for Complete Beginners – #1 PLAYER MOVEMENT, https://www.youtube.com/watch?v=TcranVQUQ5U*

[16] Brackeys: *MELEE COMBAT in Unity, https://www.youtube.com/watch?v=sPiVz1k-fEs*