Pannon Egyetem

Műszaki Informatikai Kar

Villamosmérnöki és Információs Rendszerek Tanszék

Programtervező Informatika BSc

**SZAKDOLGOZAT**

**Különböző generációk információfeldolgozási és tanulási képességeinek statisztikai alapú tesztelése**

**Csernák László**

Témavezető: Szabó Patrícia

2025

Témakiírás

A szkennelt formában megkapott témakiírás beillesztése a dolgozatba.



Hallgatói nyilatkozat

Alulírott <<név>> hallgató (Neptun kód: <<SAJÁT NEPTUN KÓD>>) kijelentem, és a dolgozat feltöltésével egyidejűleg nyilatkozom, hogy a <<dolgozatcím>> című <<záródolgozatot/szakdolgozatot/diplomadolgozatot>> (a továbbiakban: dolgozat) a Pannon Egyetem Műszaki Informatikai Kar <<szervezeti egységében (tanszékén)>> készítettem a <<végzettség>> oklevél megszerzése érdekében.

Kijelentem, hogy a dolgozatban csak a megadott és hivatkozott forrásokat használtam fel, és ezekre a vonatkozó idézési szabályok szerint hivatkoztam.

Nyilatkozom, hogy a dolgozat érdemi része saját szellemi alkotásom eredménye, és azt más intézményben, szakon, vagy felsőfokú képesítés megszerzésére nem nyújtottam be. Tudomásul veszem, hogy a plágium vagy szerzői jogsértés esetén a dolgozatom elutasításra kerülhet, és ellenem fegyelmi eljárás indulhat. Tudomásul veszem továbbá, hogy szerzői jogsértés esetén az Egyetem jogosult a dolgozat elérhetőségét korlátozni, valamint eltávolítani a dokumentumot a dolgozatok tárolására szolgáló, a témát vezető szervezeti egység által meghatározott elektronikus zárt rendszerből.

Tudomásul veszem továbbá, hogy a Pannon Egyetem a dolgozat eredményeit saját céljaira eltérő írásbeli megállapodás hiányában a Pannon Egyetem Szellemi Tulajdon Kezelési Szabályzatában foglaltaknak megfelelően szabadon felhasználhatja.

Nyilatkozom, hogy a dolgozat elkészítése során mesterséges intelligencia eszközöket *használtam /nem használtam[[1]](#footnote-1).*

Nyilatkozom, hogy a dolgozat elkészítése során az alábbi táblázatban feltüntetett mesterséges intelligencia eszközöket kizárólag a kutatási, illetve fejlesztési feladat támogatására használtam fel, az érdemi munka, elemzés és következtetések teljes mértékben saját szellemi alkotásomat képezik.

Példa a táblázat kitöltésére:

|  |  |  |  |
| --- | --- | --- | --- |
| **Alkalmazott technológia** | **Alkalmazás módja** | **Előállított tartalom** | **MI használat aránya** |
| GPT-4o (OpenAI) | szöveges összefoglaló generálása | 2.2 fejezet | 80% |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Dátum: Veszprém/Nagykanizsa/Zalaegerszeg, 20.. <<hónap neve>> <<nap>>.

*<<hallgató neve>>*

Témavezetői nyilatkozat

Alulírott <<témavezető neve>> témavezető kijelentem, hogy a <<dolgozatcím>> című dolgozatot *<<*hallgató neve*>>* a Pannon Egyetem <<tanszék neve>>én készítette <<végzettség>> végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védésre bocsátását engedélyezem.

Dátum: Veszprém/Nagykanizsa/Zalaegerszeg, 20.. <<hónap neve>> <<nap>>.

*<<témavezető neve>>*

Köszönetnyilvánítás

Tartalmi összefoglaló

Abstract

Abstract in English

**Keywords:** [list 4-6 keywords]

Tartalomjegyzék

[Jelölésjegyzék 10](#_Toc195004379)

[1. Bevezetés 11](#_Toc195004380)

[1.1. Téma bevezetése 11](#_Toc195004381)

[1.2. Pszichológiai elemzés 12](#_Toc195004382)

[1.3. Statisztikai elemzés 13](#_Toc195004383)

[2. Fejlesztési elemek és metódusok meghatározása 14](#_Toc195004384)

[2.1. Fejlesztőkörnyezet és verzió kiválasztás oka 14](#_Toc195004385)

[2.2. Fejlesztés dinamikájának lépései 16](#_Toc195004386)

[2.2.1. Ötlet és koncepció kidolgozása 17](#_Toc195004387)

[2.2.2. Játékmenet tervezése 17](#_Toc195004388)

[2.2.3. Technológia és eszközök kiválasztása 19](#_Toc195004389)

[2.2.4. Prototípus készítése (Alfa fázis) 20](#_Toc195004390)

[2.2.5. Béta tesztelés és adatbázis felállítása 20](#_Toc195004391)

[2.2.6. Kiadás 21](#_Toc195004392)

[2.2.7. Statisztikai elemzés 22](#_Toc195004393)

[3. Saját projekt felépítése 22](#_Toc195004394)

[3.1. Szint tervezési szempontok 22](#_Toc195004395)

[3.1.1. Első szint (Simon say’s) 23](#_Toc195004396)

[3.1.2. Második és harmadik szint (Pontossági reakciós teszt) 29](#_Toc195004397)

[3.1.3. Labirintus (türelmi játék) 38](#_Toc195004398)

[3.2. Adatbázis tervezés 42](#_Toc195004399)

[3.2.1. Lokális fájl alapú adatmentés 43](#_Toc195004400)

[3.2.2. Lokális adatbázis alapú adatmentés 44](#_Toc195004401)

[3.3. Szintek közötti váltás 44](#_Toc195004402)

[3.3.1. Szint indítás és teljesítés ellenőrzése 45](#_Toc195004403)

[3.3.2. Eredménymentés a játék végével 46](#_Toc195004404)

[3.3.3. Szintek közötti tovább haladás 46](#_Toc195004405)

[3.4. Alapvető játékos irányítás 47](#_Toc195004406)

[3.5. Mappastruktúra összeállítása 50](#_Toc195004407)

[4. Statisztikai eredmények 51](#_Toc195004408)

[Irodalomjegyzék 52](#_Toc195004409)

[Mellékletek 54](#_Toc195004410)

[Ábrajegyzék 55](#_Toc195004411)

[Táblázatjegyzék 56](#_Toc195004412)

Jelölésjegyzék

|  |  |
| --- | --- |
| KSH | Központi Statisztikai Hivatal |
| EU | Európai Unió |
| Indie | Independent developer |
| bug | hiba |
| Dekstop alkalmazás | asztali alkalmazás |
| Dictionary | könyvtár |
| skill | képesség |
| PEGI | Pan European Game Information |
| target | célpont |
| shoot/shooting | lövés |
| bool/boolian | logikai változó (igaz vagy hamis értékkel) |
| getter | adat lekérő függvény |
| setter | adat beállító függvény |
| script | szkript |
| SVG | Scalable Vector Graphics |
| extrudálás | térbeli kinyomás |
| game object | játéktérben lévő objektum/elem |
| SQL | Structured Query Language |
| scene | jelenet |
| collider | interaktálható terület/elem |
| tutorial | gyakorló anyag |

# Bevezetés

A digitális technológiák gyors fejlődése az elmúlt évtizedekben alapvetően megváltoztatta a társadalom életmódját és az emberekkel való kapcsolattartást. Az olyan eszközök, mint az okostelefonok, a táblagépek, a számítógépek és az internet elérhetősége és használata olyan alapvető szükségletekké vált, amelyek nélkül a modern társadalomban szinte elképzelhetetlen az élet. Ezzel egyidejűleg azonban a generációs különbségek a digitális eszközök és a technológia használatában és a digitális technológiáról való gondolkodásban is egyre hangsúlyosabbá váltak.

## Téma bevezetése

Míg a fiatalabb generációk természetesnek veszik a digitális világot, az idősebb generációknak gyakran nehézséget okoz az új technológiák elsajátítása. Egy magyarországi gerontológiai (időskori élettan) tanulmány szerint „Az internethasználat tekintetében az idősebb korosztályok nagy lemaradásokkal küzdöttek egészen az utóbbi évekig.” [1] A virtuális és digitális készségek generációk közötti különbségeinek kutatására több okból is szükség van. Egyrészt a technológiai készségek nélkülözhetetlenné váltak a munkaerő-piaci versenyképesség megőrzéséhez és a társadalmi egyenlőtlenségek csökkentéséhez. Az új technológiákhoz való alkalmazkodás és azok használatának képessége meghatározó tényezővé vált az egyén gazdasági és társadalmi sikere szempontjából.

Az e készségek elsajátításában mutatkozó nagy generációk közötti különbségek azonban nemcsak a gazdasági lehetőségeket korlátozzák, hanem a társadalmi egyenlőtlenségek új formáit is létrehozhatják. Másrészt a digitális készségek terén mutatkozó hiányosságok vizsgálata segíthet az e téren lemaradó csoportokat célzó hatékonyabb oktatási és képzési programok kialakításában, ahogyan azt több, a témával foglalkozó tanulmány, például a 2018-as „Játékalapú tanulás” című tanulmány is leírja. [2] A digitális készségek felmérése továbbá lehetővé teszi a kormányok és szervezetek számára, hogy jobban megértsék a társadalom igényeit, és intézkedéseket tegyenek a digitális szakadék csökkentése érdekében. E tanulmány célja ezért annak vizsgálata, hogy a digitális készségek (memória, reakcióidő, logika és kitartás) és a technológiához való hozzáállás tekintetében milyen mértékben vannak különbségek a generációk között. Az elemzés felmérési adatokon alapul, amelyek azt mutatják, hogy a digitális technológiában mutatkozó különbségek nemcsak az egyéni használati szokásokra, hanem a munkaerő-piaci lehetőségekre és a társadalmi befogadásra is jelentős hatással vannak. Ezeket a különbségeket egy nagyszabású tesztkörnyezetben fogom megfigyelni egy olyan elemző játékon keresztül, ahol a korcsoportok az alapvető felhasználói adatokra korlátozhatók, és generációs besorolások készíthetők. Ezután a játék eredményei alapján statisztikai meghatározással fogom vizsgálni a felhasználók közötti generációs szakadék arányát, vagyis azt, hogy az idősebb és a fiatalabb generációk között csökkenthető-e.

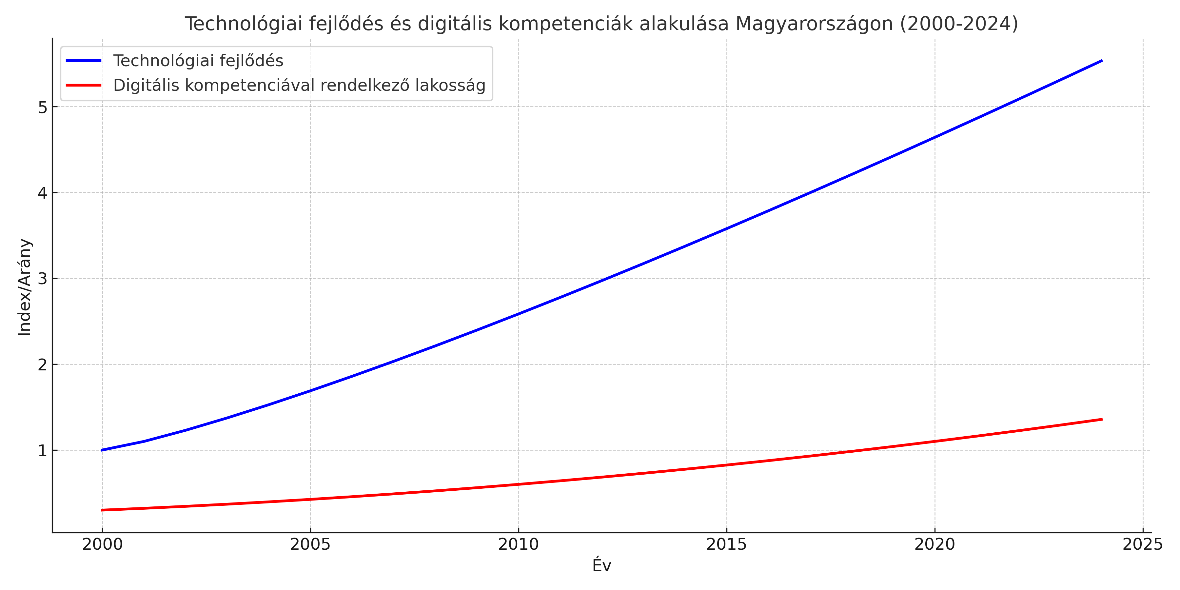
## Pszichológiai elemzés

A generációs eltérések elemzésével konkrét kutatás napjainkig nem létezik csak bizonyos rétegeket és körülményeket vizsgáltak meg, például olyan szituációkat, ahol a becsült eltérés látványos lehet eredmények terén. Az alapvető emberi logikával alapvetően mindenki arra a következtetésre jut, hogy az elöregedő társadalmi forma miatt egy olyan folyamatosan végtelenig tartó egyenlőtlen arányosság létezik, amik mondhatni sosem lesznek párhuzamosak. Az idősek sosem fogják annyira megérteni az egyre modernebb technológiákat, mint a fiatalok, akik ebben nevelkednek. Tehát a fiatalabbak és az idősebbek sose fognak egy technológiai felvilágosultsági szinten tartózkodni. Természetesen most is élnek olyan idősek a társadalomban, akik kulcsfontosságú szerepet töltenek be annak fejlődésében, mint például ilyen Vinton Cerf, aki 1943-as születése ellenére nem csak betölti az „internet atyja” szerepet, hanem jelenleg is a Google-nál dolgozik, ahol az internetes technológiák fejlesztésére és terjesztésére összpontosít. [3] Nem kell nagyon bonyolult példákat keresni ahhoz, hogy ezt pszichológiai vagy érzelmi szinten is érzékeltetni tudjuk.

Elég csak a legközéletibb példákra gondolni. A legjobb példa az, amikor például a nagyszülők, de már a szülők se értik a gyerekeik trendek által befolyásolt mindennapjaikat, folyamatos figyelem és ingerigényét. Az alapvető beszédükbe vagy beszédstílusukba beillesztett szleng szavak már semmitmondónak tűnnek. Nosztalgikussá válnak a korábbi években használt szavak, amik a szülők részéről, megszokottnak számítanak, de a gyereke szintjéről már múlt századi-nak tűnhetnek kisebb-nagyobb túlzásokkal. Ezek csak egyszerű helyzetek, amik bármikor kialakulhatnak és mindenkit frusztrációval tölthetnek el, vagy éppen ingerülté tehetik azt, aki ilyen formában érzékeli először az idő telésének korai szakaszait. Persze mindenki más, de a digitális világban elég könnyen leszűrhető viselkedés alapján, beszéd alapján vagy felhasználási gyorsaság alapján is, hogy ki mennyire korán találkozott már az újabbnál újabb technológiákkal és mennyire tudja felvenni a lépést azzal, amik a korábbi évekre jellemző ütemben megjelentek a médiában, közéletben, mindennapokban vagy akár a tudományban, oktatásban

## Statisztikai elemzés

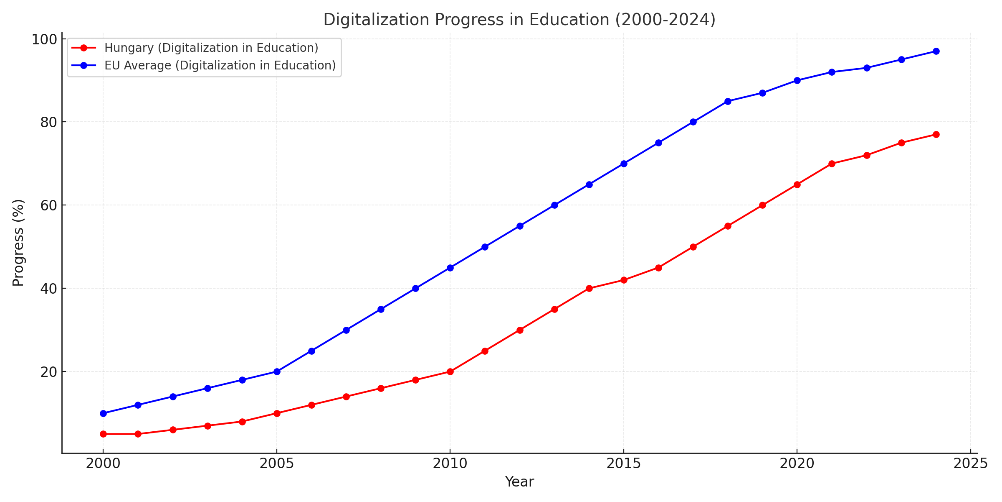
A digitalizáció és a digitális készségek fejlesztése fontos kérdés, amivel minden nap foglalkoznak a világon. A magyar kimutatásokat tekintve az emberek digitális készségei folyamatosan fejlődő lineáris vonalban vizualizálhatók, de az eltérés aránya szembetűnő lehet.



1.ábra: KSH és digitális fejlődés kimutatása [4]

A Központi Statisztikai Hivatal (KSH) és más releváns források alapján a digitális kompetenciák fejlődése Magyarországon az elmúlt két évtizedben lassú, de fokozatos növekedést mutatott. A 2023-as adatok szerint a magyar lakosság 64% - ának van legalább alapszintű digitális készsége, ami valamivel meghaladja az Európai Uniós (EU) átlagot (61%) ​.

A felzárkózás a világon mindenhol folyamatosnak mondható mivel kulcsfontosságú szerepet játszik a technológia minden ország:

* gyártásipari fejlődésében
* modernizációs folyamataiban
* hadipari és katonai felkészültségükben
* oktatásügyi szinten tartásában

2.ábra: EU és Magyarország digitális fejlődésének kimutatása a KSH adatai alapján [5]

# Fejlesztési elemek és metódusok meghatározása

Mielőtt a fejlesztési környezet, szoftver, nyelv kiválasztása megtörténne az előtt szükség van a lépések és pontok meghatározása, amik szerint időhöz kötötten megállapítható és végig lekövethető a projekt éppen folyamatban lévő szakasza az elejétől a végéig.

## Fejlesztőkörnyezet és verzió kiválasztás oka

Fejlesztőkörnyezetek terén széles a választék, amiből lehet válogatni, minden környezet egyedi „engine” -el van ellátva, ami nemcsak, hogy sajátos kinézetet ad az editor megjelenésének, de más-más teljesítményi és megjelenítési opciókat kínál.

A 3 legkedveltebb környezetek a következők:

* Unreal Engine
* Godot
* Unity

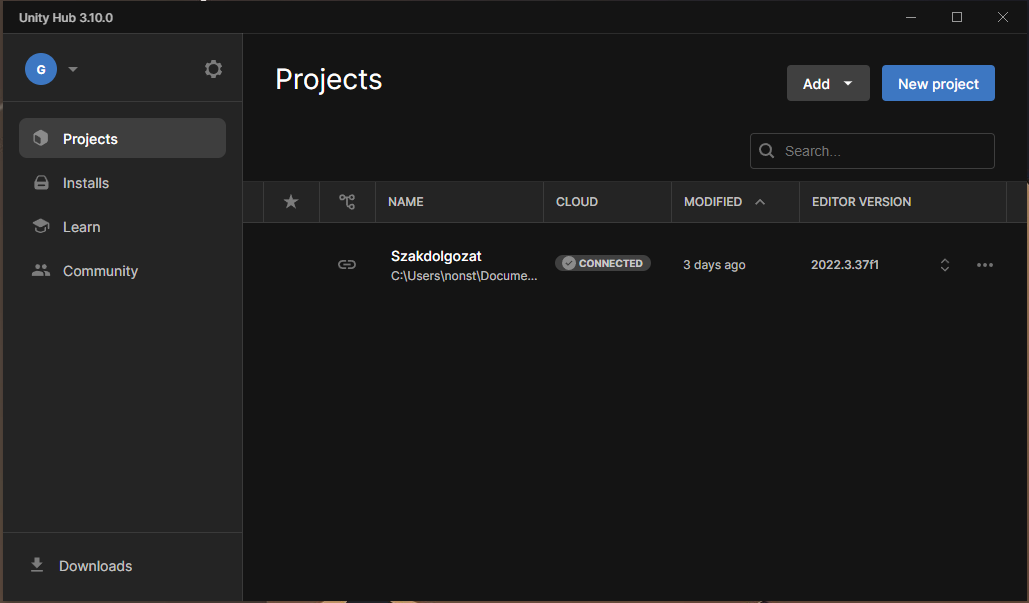
Ezek mindenki számára ingyenesen elérhetők, de bizonyos szolgáltatásokat igénybe lehet venni, amik már bizonyos árú előfizetésekért érhetők el.

A szoftverek közül a legnagyobb potenciállal és összetettséggel az Unreal Engine rendelkezik. 3D, 2D és VR játékok fejlesztésére is lehetőséget biztosít, egyedi személyre szabható editor felületet nyújt felhasználóinak. Híres a fotó realisztikus renderelésről, amely a fejlett megvilágítási és anyagkezelési rendszereknek (pl. Lumen és Nanite az UE5-ben) köszönhető. Rendelkezik beépített fizikai motorral (Chaos Physics), amely realisztikus töréseket, ütközéseket és szimulációkat tesz lehetővé. Általában nagy szabású, nyílt világú összetettebb játékok készítésére használják. Ebben fejlesztették többek között a Gears of War sorozatot vagy a Hellblade: Senua’s Sacrifice-t.

Utána rögtön a Godot látható a listában, amit főleg 2D-s vagy 3D-s játékok készítésére alkalmaznak. Egyedi jellemzői közé sorolható a GD Script, ami egy Pythonhoz hasonló, könnyen tanulható, játékfejlesztésre optimalizált szkriptelési nyelv. Bár nyelvek terén elég bőkezű a választék mivel megengedő környezetnek számít. Könnyen tanulható, nagyon jó kezdők számára, akik éppen belecsöppentek a játékfejlesztés világába. Ezek mellett a Godot lehetővé teszi, hogy csak a projektedhez szükséges funkciókat töltsd be, ezzel könnyen optimalizálható a teljesítmény. Ebben készült játék például a Kingdoms of the Dump vagy a Sea Salt.

Számomra mégis a legszimpatikusabb környezet a Unity volt, mivel nehézségi szintben valahol az előbb említett kettő közé sorolható mégis sok példa mutatja, hogy alkalmas AAA játékok elkészítésére is. Támogatja a PC, konzol, mobil (iOS, Android), web (WebGL), AR/VR és egyéb platformok fejlesztését. Fő programozási nyelve a C#, ami könnyen tanulható és számomra ismert, kedvelt nyelv. A Unity Asset Store rengeteg előre elkészített 3D modellel, scripttel, textúrával és más eszközzel segítheti a fejlesztési folyamatot. Emellett rengeteg oktatóanyag van az interneten különböző leleményes fejlesztési metódusokra, amelyekkel el lehet sajátítani a bonyolultabb játék fejlesztési képességeket. Folyamatosan fejlődik új funkciókkal és eszközökkel, például az URP (Universal Render Pipeline) és HDRP (High Definition Render Pipeline) ami lehetővé teszi a játékok foto realisztikus megjelenítését. A Profiler és egyéb optimalizációs eszközök segítségével gyorsan azonosíthatod és javíthatod a teljesítményproblémákat és még sok más hasonló jellegzetességgel rendelkezik, amik nagyon gyakran frissülnek újabb és újabb Editor verziók formájában. Ebben az Engine-ben készültek például a Cuphead, Subnautica és Ori and the Blind Forest nevű játékok.

Maga az Engine 2005-ös megjelenése óta világszinten használt, elismert és népszerű. Nem csak kizárólag játékfejlesztésre alkalmazzák, hanem például a filmiparban való szimulálásra, oktatásra, építészeti vizualizációkra vagy orvosi szimulációkra is használják. Sok Indie játékfejlesztő eszköze. Stúdiók terén az Electronic Arts (EA), a Ubisoft, a Blizzard Entertraiment vagy a Supercell is mind a mai napig fejlesztenek benne legyen szó mobil, PC, vagy VR játékokról. És biztos vagyok benne, hogy az elkövetkezendő években érkezni fognak még Unity-ben készül AAA címek.

Én a Unity 2022.3.37f1 Editor verzióját választottam mivel ez egy megbízható régóta támogatott LTS verziónak számít. Kódoláshoz a Visual Studio 2022-es verzióját használtam, illetve a fejlesztés nyomon követesét a GitHub Dekstop alkalmazásában figyeltem, mivel így a „Push-Pull” metódust használva egy online repository-ból több helyről is hozzáfértem a projekthez és naprakészen tudtam fejleszteni.

3. ábra: Unity Hub felülete

## Fejlesztés dinamikájának lépései

Ezen lépések alapjai gyakran előbb képesek megszületni, mint hogy bármi anyagi jellegű támogatásról vagy kivitelezésről szó esne. A koncepció váza ötlet szinten lehet, hogy évekkel ezelőtt is létezett. Ilyenkor fontos a teljeskörű felmérés és a fejlesztési szakaszok megállapítása.

A lépések ezekben a pontokban meghatározhatók:

* Ötlet és Koncepció kidolgozása
* Játékmenet tervezése
* Technológia és Eszközök kiválasztása
* Prototípus készítése (Alfa fázis)
* Béta tesztelés és adatbázis felállítása
* Kiadás
* Statisztikai elemzés

A lista a teljeskörű fejlesztés lefedését hivatott elméleti szempontból ellátni. Egy nagy fejlesztő cég és egyénileg dolgozó szabad általában önerőből finanszírozott játékfejlesztőnek (azaz másnéven Indie játékfejlesztőnek) is kulcsfontosságú definiálni a lépéseket, ami alapján konzisztensen meg lehet határozni bármelyik pillanatban, hogy a jelen állás szerint hol tart a projekt. Ez alapján lehet teljesítménymutatókat, célokat vagy pénzügyi döntéseket meghozni, megalapozni.

### Ötlet és koncepció kidolgozása

Az ötletelés fázisban alakul ki a koncepció és ez a játékfejlesztés első és nagyon fontos szakasza. Itt határoztam meg a projekt alapjait, és alapoztam meg a későbbi fejlesztési döntéseket. Fontos egyértelműen meghatározni, miről fog szólni a játék, mi lesz a központi élmény, és milyen érzéseket akar kiváltani a játékosokból egyszóban milyen céllal születik. Itt fektetjük le a mechanikai alapokat és a narratív irányvonalat. Ezek mellett, ami piaci szemszögből még fontos lehet az a piaci réteg megcélzása, illetve a befogadóréteg meghatározása magyarul a célközönsége.

Fontos meghatározni a játék műfaját is mivel a története szempontjából, illetve grafikai és világépítési szempontból fontos tud lenni, hogy hova van a játékos belehelyezve. A nagyobb játékkiadók gyakran csinálnak piackutatást a koncepció felállításakor pont azzal a céllal, hogy megtudják a tökéletes réteg, műfaj, stílus, hogy van elképzelve az adott generációban, korszakban.

### Játékmenet tervezése

A játékmenet tervezése a projekt egyik kulcsfontosságú szakasza, mivel itt dől el, hogy a játék milyen interakciókat kínál a játékosok számára. Ebben a fázisban határozod meg, hogyan fog működni a játék mechanikája, milyen szabályok irányítják, és milyen kihívások várnak a játékosokra.

A számomra felmerült kérdések a következők voltak:

* Mi lesz a játékos célja, mit kell teljesítenie?
* Mit kap jutalmul a teljesítésért?
* Mi motiválja a játék folytatásában?
* Milyen eszközökkel és mechanikákkal éri el ezt a célt?
* Hogyan biztosítod, hogy a játék kihívásai érdekesek és kiegyensúlyozottak legyenek?

Számos hasonló kérdést tettem fel magamnak annak érdekében, hogy értelmet adja a játéknak és folyamatosan le tudja kötni a játékos figyelmét

A játékos célja a minél jobb eredmény elérése lesz, egyfajta versenyszellem, amit a többi játékos eredményei láttán érezhet. A motiváció sokakban megvan, hogy mások felé kerekedjenek és ezt ezen játékok során megtehetik mivel az top eredmények láthatóak. Ez egyben meg tudja adni a játékosnak a motivációt is mivel mások felé tud kerekedni a saját generációján belül. A játékok teljesítéséért a játék megerősítéseket küldd a teljesítésről, illetve az azzal elért helyezésről. A mechanikák, amikkel ezt el tudom érni pont a tesztekben merülnek ki. Logika, reakcióidő, gyorsaság, türelem, memória. Ezek mind olyan tesztelhető készségek, amikkel mindenki rendelkezik csak eltérő arányban.

A játékmenet tervezésénél figyelembe kell venni az interaktivitást és a játékosok viselkedését is. Fontos, hogy a játék különböző rétegeket kínáljon: például legyen benne gyorsan megtanulható, de mélyebb stratégiákat kínáló rendszer.

Általánosságban ebben a szakaszban készülnek a dokumentációk és az első vázlatok a játék rendszereiről, mint például:

* Fő mechanikák és játékrendszerek (pl. harc, felfedezés, építés stb.)
* Szintek és pályák szerkezete
* Interfész és felhasználói élmény tervezése

A játékmenet prototípusok segítségével tesztelhető, hogy láthatóvá váljon, mennyire szórakoztató és működőképes a tervezett rendszer. Az eredmények alapján a tervek finomhangolása elengedhetetlen a későbbi fejlesztési szakaszok sikeréhez.

A cél itt az, hogy egyértelmű, részletes, és átlátható tervet készíts, amely a fejlesztőcsapat minden tagja számára érthető, és amely alapot biztosít a játék megvalósításához.

Az én esetemben a játék mechanikáinak kitalálása a koncepcióban és a tesztelhetőségben merült ki, mivel olyan játékokat kellett kitalálnom, amik reprezentálni tudják az adott játékos generáción belüli eredményeit. A játékoknak érthetőnek kell lenniük idősek és gyerekek számára is, figyelemfelkeltőnek kell lennie, hogy meg tudja fogni az adott játékos fantáziáját és logikailag konzisztensnek kell lennie, amivel elkerülhetők a félreértések és logikai malőrök.

### Technológia és eszközök kiválasztása

A megfelelő technológia és eszközök kiválasztása döntően befolyásolja a játékfejlesztés menetét és eredményét. Ez a lépés alapozza meg a munkafolyamatokat, illetve határozza meg, hogy a csapat vagy az egyén milyen gyorsan, hatékonyan és kreatívan tud dolgozni.

Ebben a szakaszban határoztam meg, hogy a projekt igényeihez melyik fejlesztői motor, nyelv, platform és eszközök illenek leginkább. Kiemelt szerepet kap a játék célplatformja (pl. PC, mobil vagy konzol) és a rendelkezésre álló költségvetés, illetve azt befogadó közönség. Például egy komplex, 3D-s akciójátékhoz más technológia szükséges, mint egy egyszerűbb mobiljátékhoz.

A döntés során nemcsak technikai tényezőket kell figyelembe venni: az eszközöknek és a technológiának támogatnia kell az együttműködést és a verziókezelést is, hiszen a fejlesztés későbbi szakaszaiban az időt és erőforrást takaríthat meg.

Egy jó technológiai választás nemcsak a jelenlegi projekt igényeit kell, hogy kiszolgálja, hanem a jövőbeli frissítések és bővítések lehetőségét is figyelembe kell vennie. Egy rossz döntés ugyanis korlátokat szabhat, amelyek lassítják a fejlesztést, vagy akár gátolják a játék koncepciójának teljes megvalósulását. Ezért a választási folyamatot mindig alapos kutatásnak kell megelőznie.

A cél egy olyan technológiai háttér kialakítása, amely összhangban van a csapat képességeivel, a projekt céljaival és a végső játékélménnyel. Ez a döntés alapvetően határozza meg, hogy a játékfejlesztés milyen irányban halad tovább.

A Unity -ben való fejlesztés miatt egyértelműen a C# nyelvet kellett választanom mivel a környezet ezt a nyelvet támogatja. Számomra nem újdonság a nyelv mivel korábban foglalkoztam már „Dekstop alkalmazás” fejlesztésével és játék fejlesztésével is, viszont rengeteget fejlődött az évek alatt ezért érdemes volt elsajátítanom az újdonságokat a kódolás logikájának felépítése végett.

A környezet könnyen összekapcsolható a „Visual Studio” -val, ami pár alapvető import beállítással könnyed hozzáférést biztosít azokhoz az eszközökhöz, amikre például a 3 dimenziós térben való koordináláshoz vagy a játékos interakcióinak lekezeléséhez nekem szükségem volt. Így biztosítva volt minden ahhoz, hogy megkezdjem a fejlesztést.

### Prototípus készítése (Alfa fázis)

A prototípus vagy másnéven „Alpha version” egy félkész állapota a játéknak. Ilyenkor pont ott tart általában a fejlesztés, hogy a fejlesztőcsapatnak szüksége van külső tesztelőkre a belső tesztelő embereken kívül. Illetve ez a fázis a promótálás kezdete, amikor a nagyvilág számára megjeleníthető tartalom készülhet a meglévő kinézetről, működő funkciókról és egy kicsit meg tudja mozgatni azt a réteget, akik az adott zsánerre nyitottak.

A tesztelésekhez privát hozzáféréseket szoktak osztogatni a kontent készítő videósoknak vagy stream-er embereknek, akik nemcsak, hogy az esetleges „bug” -ok megkeresésében is sok segítséget tudnak nyújtani, de a közönségük már biztosíthat egy nagyobb vagy kisebb számú befogadóréteget a játék felhasználóbázisának.

Mechanikailag és fejlesztési szempontból, a prototípus készítésekor a hangsúly a működésen van, nem a grafikai részleteken. Az egyszerűbb, „helykitöltő” elemek (például alapvető modellek, textúrák vagy animációk) használata lehetővé teszi, hogy a csapat gyorsabban dolgozzon, miközben a játékmenet alapjaira összpontosít. [6] [7]

Egy prototípusnak a következőket kell tartalmaznia:

* A játék legfontosabb mechanikáinak működő modelljét. Például, ha a játék tartalmaz harcot, ugrást vagy rejtvényeket, ezeknek már működniük kell.
* Alapvető felhasználói felületet, amely segíti a játék tesztelését, még akkor is, ha ez csak ideiglenes dizájn elemekkel van megoldva.
* Ideiglenes vizuális elemeket („placeholder” grafikák), amelyek segítik a funkciók vizualizálását, de nem reprezentálják a végleges művészeti irányvonalat.

### Béta tesztelés és adatbázis felállítása

A béta tesztelés a játékfejlesztés egyik legfontosabb része, ahol a cél a stabilitás, hibajavítás és a játékmenet finomítása. Ebben a szakaszban a játék már majdnem kész, de szükség van a játékosok visszajelzéseire, hogy a lehető legjobb élményt nyújtsa. Ezzel együtt történik az adatbázis létrehozása is, ami a játék technikai hátterét adja, főleg, ha online vagy többjátékos funkciók is vannak.

A bétatesztelésnek két folyamata van:

* Zárt bétatesztelés:

Csak kiválasztott tesztelők (pl. csapat tagjai, meghívott szakértők vagy dedikált tesztelők) játszhatnak a játékkal. Ez lehetőséget ad arra, hogy kontrollált környezetben gyűjts adatokat.

* Nyílt bétatesztelés:

A játék szélesebb közönség számára elérhetővé válik, így nagyobb volumenű visszajelzést kapsz a játékélményről és a technikai stabilitásról.

Az adatbázis felépítéséhez a koncepció által már előre tudható, hogy mekkora komplexitású struktúrára lesz szükség, mivel minél nyíltabb online terekben a játék annál fontosabb a jogi és személyi adatok biztonságos eltárolása és megőrzése. Ebben az esetben a tesztelési jelleg miatt a játékosoktól csak alapvető adatokat kérünk be ami a statisztikához vagy a beazonosításhoz szükséges lehet. Ilyen adatok például a játékos neve, születési dátuma és email címe. Mint egy űrlap kitöltésekor.

Ha az adatbázis készen áll a tesztelési folyamatnak több külön célja kerül előtérbe, ami a fejlesztési szempontokat és az élvezeti szempontokat finomítja a játékosok számára.

Előtérbe kerül a hibajavítás, játékmenet optimalizálása, teljesítmény optimalizálása és a játék felhasználói vélemények általi javítása. Mivel ezek a visszajelzések segítenek megérteni, hogyan reagálnak a játékosok a játék mechanikájára, történetére és általános élményére.

Azt mondják, ha a felhasználó elégedett a cég is elégedett lesz. Sok esetben sajnos van példa arra, hogy egy cég visszaél a játékosok ideével és energiájával. Illetve vannak esetek amikor egy játék fejlesztése megakad a béta fázisban és az éveken keresztül csúsztatott megjelenési dátumig nem jut el a játék. Ilyen volt például a Star Citizen, a játékot 2014-ben tervezte kiadni a Cloud Imperium Games kiadó, a koncepciója egy masszív űrszimulátor játék, ami akkoriban nagy mértékben hívta fel magára a figyelmet, a végtelen univerzum és a valósághű fizika ígéretével. A közösségi finanszírozás révén rekordösszeget gyűjtött (600+ millió dollár). Ám a játék évek óta az „alpha” szakaszban van, és bár folyamatosan frissül, a végleges megjelenés időpontja bizonytalan. Emiatt a támogatók nagy része csalódott a kiadóban a be nem teljesített ígéreteik miatt.

### Kiadás

A játék kiadása a fejlesztési folyamat csúcspontja, ahol a projekt végre a közönség elé kerül. Ez a szakasz magában foglalja a technikai, marketing és logisztikai feladatok összehangolt végrehajtását, hogy a játék sikeresen elérje a célközönségét.

* Végleges verzió elkészítése:

A béta tesztelésből származó hibák javítása, optimalizációk és az utolsó finomhangolások befejezése után a játék eléri a „gold master” állapotot, vagyis a kiadásra kész végleges verziót.

* Platformkövetelmények teljesítése:

Ha a játékot különböző platformokra szánják (pl. Steam, konzolok, stb…), minden platform specifikus követelményét teljesíteni kell. Ezek magukban foglalhatják a minőségellenőrzési vizsgálatokat (pl. PlayStation Certification, Apple App Review).

* Kiadói dokumentáció:

Minden hivatalos anyagot, például licencszerződéseket, adatvédelmi szabályzatokat és felhasználói kézikönyveket elő kell készíteni. A kiadás a játékfejlesztés gyümölcse, ahol a munka során megálmodott ötletek végre életre kelnek a játékosok számára. A megfelelő előkészítéssel és utókövetéssel a kiadás nemcsak a projekt lezárását, hanem hosszú távú sikerek kezdetét is jelentheti.

### Statisztikai elemzés

A statisztikai elemzés döntő része a játékfejlesztés utolsó szakaszának. Segít a fejlesztőknek és kiadóknak megérteni a játék teljesítményét, valamint a játékosok viselkedését és kedvelését. Ez az információ elengedhetetlen a további fejlesztésekhez, frissítésekhez, hibajavításokhoz, és a játék sikeres életciklusának biztosításához.

Ebben a projekt környezetben ez a legfontosabb rész, mivel a tesztelői adatbázis kielemzése után kerülnek elő a válaszok a feltett hipotézisek és kérdések megválaszolására. Az eredmény lehet várt/váratlan, de akármi is legyen kutatási teret nyithat azoknak, akik tovább foglalkoznának ezzel a kérdéssel.

# Saját projekt felépítése

## Szint tervezési szempontok

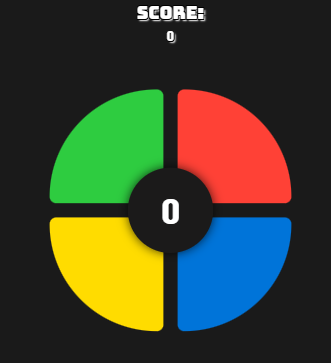
A projekt sikeres kivitelezéséhez fontos volt meghatározni azokat a tervezési elveket és szempontokat, amelyek biztosítják a játék átláthatóságát, optimalizálhatóságát és a felhasználói élményt.

Első sorban egy olyan intuitív játékmenetet kellett meghatározni, ami a megadott fejlesztési célnak megfelel tehát különböző játékok által megállapíthatók generációs különbségek a felhasználók digitális képességeiben. Tehát a szinteket úgy határoztam meg hogy mindegyik egy-egy olyan képességet teszteljen a játékosoknál, ami attól függően jobb vagy rosszabb lehet minél többet találkozott a játékos ilyen jellegű virtuális játékokkal, minél gyakrabban old meg bonyolultabb, összetettebb gondolkozást igénylő feladatokat, illetve minél több olyan eszközt használ a mindennapjaiban, ami befolyásolhatja a megoldás kialakulásának idejét vagy éppen a megoldás minőségét.

A szinteket így 4 részre tagoltam és az által álltam neki a fejlesztésnek.

### Első szint (Simon say’s)

Az első szint a játékos logikai képességeit hivatott felmérni így több opció közül én a „Simon mondja” nevű közismert játék egyik változatát szerettem volna megvalósítani.

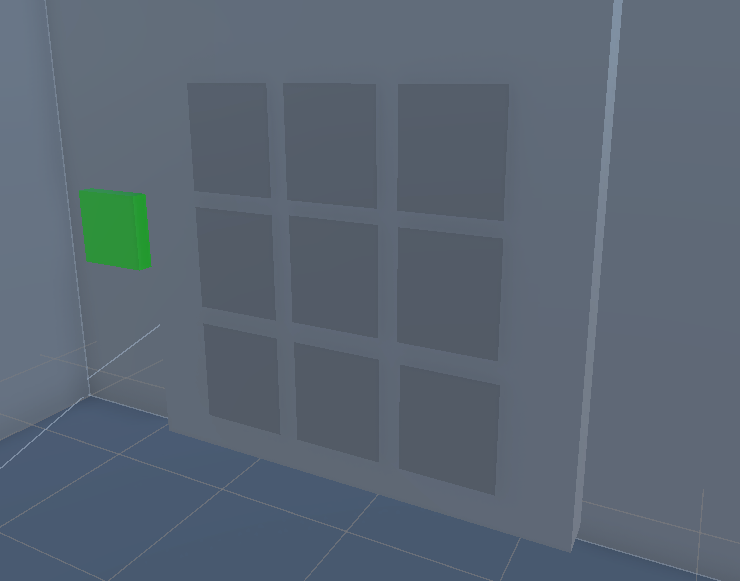
A játék egy klasszikus formája a memóriajátékoknak, lényege, hogy a gép által generált színeket a játékos ugyan abban a sorrendben vissza tudja adni és minden helyesen eltalált kombináció után egy színnel bővül a sorozat, így szükség van folyamatos figyelemre és pontosságra. A cél, hogy a játékos minél tovább képes legyen pontosan visszaidézni a szekvenciát és el tudjon érni minél magasabb pontszámot.

4. ábra: Simon Say’s illusztráció egy online memóriafejlesztő felületről [8]

A játék nem csak szórakoztatás szempontjából alkalmas a feladatára, de a játszása során rengeteg kognitív készség fejlődik. Alapvetően az agyunk rövidtávú memória központja az, amit igénybe vesz, de a figyelemkontrollt is erősíti mivel a folyamatosan gyorsuló és bővülő sorozatot pontosan kell visszaadni. Ezen kívül nagy a szerepe a koncentrációs készségnek is, hiszen egy apró figyelmetlenség és a folyamat lezárul.

Az ilyen jellegű kihívások pozitív hatással vannak az agyunk működésére és tökéletes első kihívás a kevesebb digitális kompetenciával rendelkező személyek számára is. Segíthet a mentális frissesség megtartásában. Emellett a rendszeres memória- és figyelemgyakorlatok segíthetnek a tanulási képességek fejlesztésében, az információk gyorsabb feldolgozásában és a problémamegoldó készség javításában. A játék tehát nem csupán szórakoztató kihívás, hanem hasznos eszköz is lehet a többi szint felvezető játékának. A különböző generációba született személyek számára egyaránt ismerős a játék neve így nem okozhat gondot a tesztelők számára. Illetve a játékmenet könnyen átlátható így gyorsan elsajátítható és tanulható.[9]

#### Vizuális kialakítás

Az alapvető játék mintájára én is létrehoztam egy 3D-s játékpanelt, amibe beleépítettem a játék mechanikáit. A játék vizuálisan letisztult és minimalista, ami segít a játékos figyelmét a lényeges elemekre irányítani. A falon található négyzetrácsos panel egy interaktív felület, amely a játék fő mechanikáját támogatja. A mellette lévő zöld gomb vizuálisan jól elkülönül mivel a játék indítására szolgál. Az egyszerű geometriai formák és a kontrasztos színek egyértelművé teszik a játékos számára, hogy mely elemekkel lehet kapcsolatba lépni, ezáltal intuitív felhasználói élményt biztosítanak.

5. ábra: Simon Say’s vizuális kialakítása

Ez a verzió a klasszikus Simon mondja játék egy továbbfejlesztett és nehezített változata, mivel nem csupán a hagyományos négy színnel operál, hanem egy kilencgombos rácsot használ, ami jelentősen növeli a kihívás szintjét. A megnövelt lehetőségek miatt a játékosnak több vizuális információt kell feldolgoznia és hosszabb szekvenciákat kell megjegyeznie.

A panelen kívül a felhasználói interfészen is látható olyan elem, ami a játékhoz köthető, ez egy pontok visszajelzésére szolgáló panel, ami folyamatosan frissül amint a játékos a „következő szintre lépett” tehát egyel bővül az eddigi színek kombinációs listája.

#### Játék logika kialakítása

A játék kialakításában voltak logikai és programozási kihívások mivel ez egy kibővített verziója az alapvető 4 gombos változatnak. A mechanikája alapvetően 4 kulcsfontosságú elemből épült fel:

* Szekvenciális színgenerálás
* Felhasználói bemenetek lekezelése
* Ellenőrzési fázis
* Játékállapotok kezelése és pont mentése

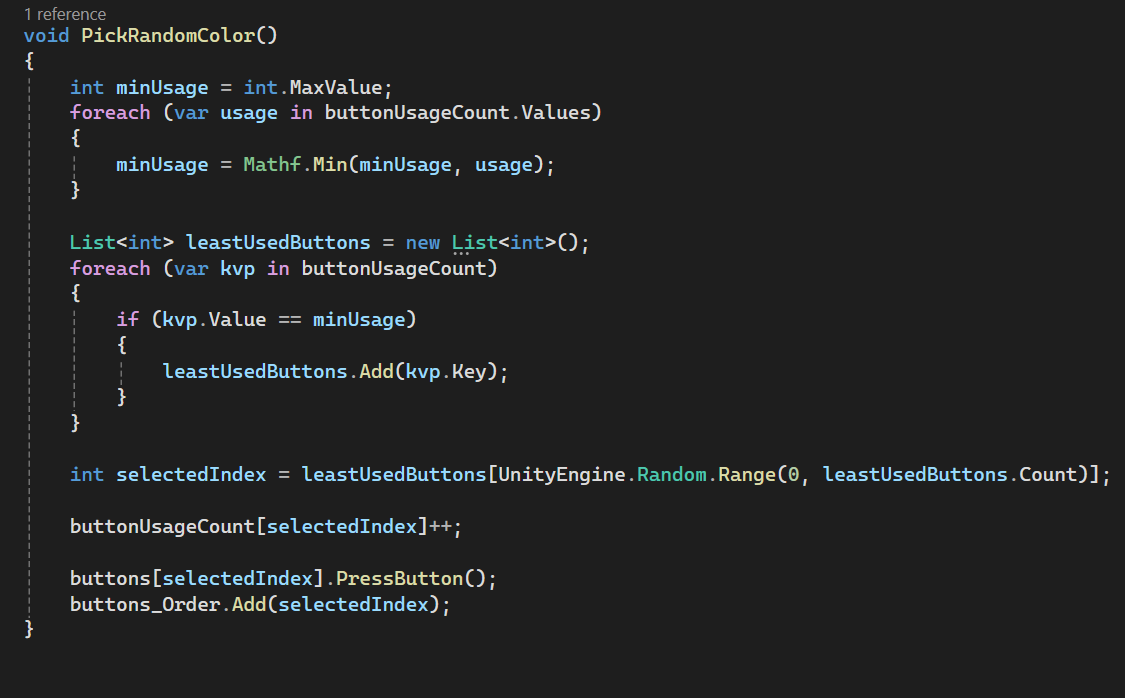
Ezek az elemek 4 különböző osztály segítségével kerültek implementálásra, így elkülöníthető minden segédfolyamat, illetve logika, ami a játék játszhatóságát alapvetően nyújtja. A kódok elkülönítése a fejlesztés szempontjából fontos volt mivel így konzisztensebb, letisztultabb és átláthatóbb lett, ami továbbfejleszthető és könnyen hibakezelhető.

#### Szekvenciális színgenerálás

A színek generálása az egyik fontos metódusa a játéknak mivel ez által bővül az a színlista, amit a játékosnak pontos sorrendben vissza kell adnia. Az általam megvalósított verzió máshogy közelíti meg ennek a logikának a felépítését.

A gombok sorrendjét egy listába tárolom és folyamatosan bővítem a gép által random sorsolt gombbal. A kódom használati gyakoriság által véletlenszerűen választja a soron következő gombot, azért építettem bele ezt a metódust mert a tesztek során többször előfordultak olyan sorozatok, amik ismétléses alapon ugyan azokat a gombokat használták fel, ez egyszerűsítette a játékot és lehetővé tette a könnyű pontszerzést. A komplexitást ezáltal a módszer által növelni tudtam, mivel így minden gomb kihasználásra kerül és jobb koncentrációs készséget igényel a gombok szétszórtsága és mennyisége miatt.

Ehhez a már elmentett sorrendben lévő gombokat egy úgynevezett „Dictionary” -be is menti, ami kulcs-érték párokat tárol így tudja meghatározni, hogy adott gomb hányszor került eddig használatra.

A függvény, ami a random kiválasztást kezeli a „PickRandomColor ()” néven van meghatározva.

6. ábra: „PickRandomColor ()” függvény színek random kiválasztásához

Maga a függvény egy egyszerű folyamat elvén működik:

1. Létrehozunk egy változót az elején, amelyben a legkisebb előfordult számot fogjuk tárolni (int minUsage)
2. Végig megyünk a szótár elemein egy ciklussal és egy matematikai függvénnyel meghatározzuk és átadjuk a legkevesebbszer használt gomb előfordulási számát
3. Létrehozunk egy listát (a leastUsedButtons tartalmazza az összes olyan gomb indexét, amely a legkevesebbszer szerepelt a játék során) és Ha egy gomb előfordulási száma (kvp.Value) megegyezik az előzőleg kiszámított minUsage értékkel, akkor hozzáadjuk a listához
4. Ha több olyan gomb van, amely egyformán kevésszer szerepelt, akkor véletlenszerűen kiválasztunk közülük egyet és a selectedIndex egy olyan gomb indexét fogja tartalmazni, amit hozzáadathatunk a sorozathoz
5. a buttonUsageCount szótárban megnöveljük a kiválasztott gomb előfordulásainak számát
6. A gép megnyomja az adott gombot, hogy ez vizuálisan is jelezve legyen a játékosnak majd hozzáadjuk ahhoz a listához, ami a játék jelenlegi sorozatát tárolja

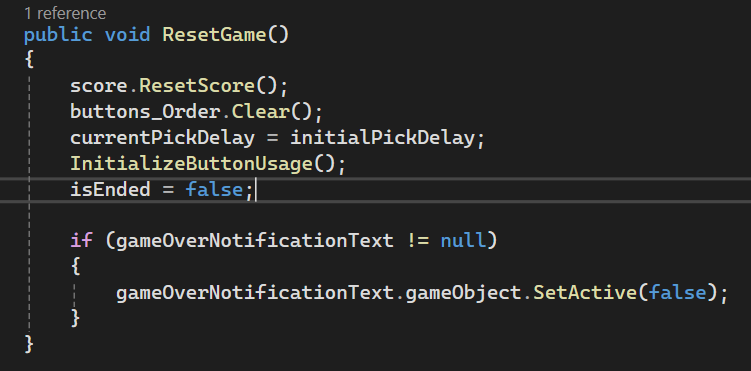
Ez a módszer jobb és kiegyensúlyozottabb, mint egy teljesen véletlenszerű választás. Így sosem lesz egyhangú a játék.

#### Felhasználói bemenetek lekezelése

A játékban 2 fajta fő felhasználói bemenet van:

1. A játék indítására használt gomb

Itt a kód egy gomb komponens formájában van inicializálva, ami megnyomásra elindítja a folyamatot. A gomb megnyomásakor meghívódik egy „ResetGame ()” nevű metódus, ami alaphelyzetbe állítja az értékeket például a:

* Játékos eredményét
* Törli a gombok sorrendjének listáját
* Visszaállítja a játék alapvető sebességét
* Alaphelyzetbe állítja a gombok előfordulásának szótárát
* Elrejti a játék végén visszajelzésül szolgáló értesítést
* A játék végét jelző változót visszaállítja „hamis” értékre

7. ábra: „ResetGame ()” metódus

1. Simon gombok megnyomása

Itt a játékos a megadott „SimonSaysButton” nevű listában lévő (összesen 9) gombot képes megnyomni. Amikor rányom egy gombra meghívódik egy függvény, ami ellenőriz 3 megszabott feltételt:

* 1. A játék éppen nem mutatja a sorozatot
  2. Van legalább egy gomb a sorrendben
  3. A játék nem ért véget

Ha ezek a feltételek teljesülnek akkor a játékos meg képes nyomni a gombot majd jön a vizuális visszajelzés a játékos számára a gomb felvillanásával és egy hang lejátszásával. Ezután a játék ellenőrzés fázisa következik.

#### Ellenőrzési fázis

Amint megtörténik a játék általi gomb kiválasztása és a játékos erre reagálva megnyomott egy gombot, ami szerinte a soron következő, elkezdődik az ellenőrzése annak, hogy az adott gomb jó volt-e.

Itt 2 lehetséges dolog történhet:

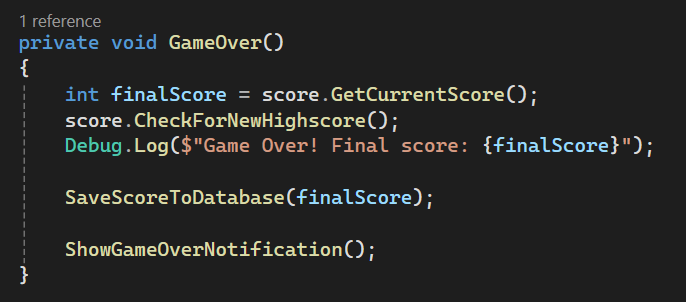
1. A játékos a helyes gombot nyomta meg

* a „pickNumber’ változó növekszik ezáltal érzékeli a játék a tovább haladást
* növekszik a játékos pontszáma
* elindul a következő színválasztási folyamat

1. A játékos a helytelen gombot nyomta meg
   * a játék végét jelző változó igaz értékre vált („isEnded”)
   * meghívódik a „Game Over ()” metódus

#### Játékállapotok kezelése és pont mentése

Amint meghívódott a „Game Over ()” függvény a játék ciklikussága megszakad és a következő lépések hajtódnak végre:

* Elmenti a játékon belül a végső pontszámot
* Ellenőrzi, hogy született-e új rekord a pontszámmal
* Megjeleníti az értesítést, ami a játék végét jelzi a játékosnak
* Elmenti a pontszámokat az adatbázisba a játékos azonosítójával

8. ábra: Simon Say’s „Game Over ()” függvény

### Második és harmadik szint (Pontossági reakciós teszt)

A második szint mechanikailag egy célpontokra lövő játékot prezentál a játékosnak. A célpontok bizonyos időközökkel periodikusan jelennek meg, és ezeket kell minél pontosabban kilőni.

Ez a játék sok szempontból kiderítheti az egyének digitális képességeit mivel komplexebb irányításra van szükség hozzá. Aki nem találkozott még videojátékokkal, amiben megtalálható a lövés metódus, mint olyan, annak a feladat, hogy a célpontokra minél gyorsabban és pontosabban rá tudja húzni az egeret majd kattintani tudjon megerőltető és fárasztó lehet. Ennek a gyakorlatnak az ismétlésével viszont nem csak az egérrel való kordinációs képesség javulhat a 3 dimenziós térben, de később ez a tudás felhasználható sok más népszerűbb játékban is és meghozhatja a játékos kedvét az ilyen jellegű kikapcsolódáshoz is.

Pszichológiai szemszögből vizsgálva, akár közelebb hozhatja az idősebb generációt (akik nem rendelkeznek nagy digitális tudással) a fiatalabbakhoz, akik ilyen jellegű játékokkal nap mint nap találkoznak, játszanak és lekötik az idejüket. A közös tanulási folyamat szociális értékeit nem szabad alábecsülni, mivel az, ha a fiatalabb tud tapasztalatot átadni az idősebbeknek közös tanulási metódusok által, könnyen örömtelivé tehet pillanatokat és hozzásegítheti az idősebbeket a könnyedebb digitális eszközhasználathoz. A javuló teljesítmény így nem csak a játékokban mért eredményekben ütközhet ki, hanem a mindennapokban is előidézheti a magabiztosabb eszközhasználatot ezzel rengeteg önbizalmat növelve az egyéneknél.

Sok tanulmány szerint az egyének érzékelési képességei nagy arányban tudnak fejlődni ilyen jellegű játékok játszásával mivel gyors reakcióidőt és döntéshozatalt igényelnek és az ingereknek hála ezek a „skill” -ek gyorsan tudnak fejlődni. Ezen kívül az úgynevezett „multi tasking” tehát szétosztott figyelmet igénylő feladatok elvégzésében is képesek fejlődni.[10]

Sok szülő és ezzel foglalkozó szakember szerint viszont az akciójátékok túl gyakori használata vagy játszása okozhat személyiségben feltűnő változásokat, amik negatív irányba terelik a játékost. Az erről szóló cikkek azt kutatják, hogy milyen hatással vannak a „lövöldözős” játékok a stresszközpontunkra ezáltal a türelmi képességünkre és a nyugalmunkra. A játékok megjelenésekor ezért fontos kérdés eldönteni az adott játék irányultságát és az ennek megfelelő tartalmi besorolást adni neki. Erre már van bevett gyakorlata a játékiparnak, amit „PEGI” -nek hívnak ez a „Pan European Game Information” rövidítése és egy olyan besorolási rendszer, ami arra szolgál, hogy ezáltal egy megfelelő egységes módon legyenek a fogyasztók tájékoztatva, hogy pontosan mit is tartalmaz a videójáték.[11][12]

A következő besorolási szintek vannak:

* **3+** – Minden korosztálynak megfelelő, erőszakmentes.
* **7+** – Enyhe félelmetes jelenetek lehetnek benne.
* **12+** – Enyhébb erőszak, trágár nyelv, szexuális utalás.
* **16+** – Erőszak, durva nyelv, droghasználat valósághű ábrázolással.
* **18+** – Intenzív erőszak, explicit szexuális tartalom, súlyos nyelvezet, súlyosabb témák.

Ezen kívül, ha a játék megkapta a besorolási szintet különböző jelzők kerülhetnek még a dobozára, amik az alábbiak lehetnek:

* Erőszak/Agresszió
* Durva nyelvezet
* Félelem keltő elemek
* Drog- vagy alkohol fogyasztás
* Szexuális tartalom
* Szerencsejátékok
* Diszkrimináció
* Online interakciók

9. ábra: PEGI tartalom jelzések [13][14]

Az eddig olvasott irodalmak és tartalmak által én a saját projektemet nézve úgy döntöttem, hogy 2 értéket fogok mérni a szinteken. A pontosságot találati pontok koordinátái szerint és a reakcióidőt, ami megjelenéstől találatig számított másodperc és milliszekundum formában kerül eltárolásra.

A tesztelések során kiderült, hogy a játékosok igényessége nagyon eltérő tud lenni, akadt olyan tesztelő, aki nagyon ügyelt a pontosságra és akadt olyan, akit csak a reakció idejének javítása érdekelt.

#### Vizuális kialakítás

A játék környezete szerint a szobának egy máshogyan kialakított verziójában játszódik. A játékos belépve 2 elemre lehet figyelmes, egy pisztolyra, ami egy oszlopon van elhelyezve és egy zöld gombra, ami az indítást vezérli.

Közelebb érve felugrik egy moduláris ablak, amiben megtalálható a szöveg, ami segít a játékosnak megérteni, hogy mi is lesz a feladata ebben a kihívásban, illetve irányt mutat a játék elkezdéséig.

Első lépésként felhívja a figyelmét a pisztoly felvételére, ami nélkül nem tudja elkezdeni a játékot. Ha mégis megpróbálja egy felugró szöveg emlékezteti a következő kiírással „Kérem vegye fel a fegyvert”. Ezen kívül jól érthető módon leírja, hogy pontosan hány célpont fog megjelenni, hol fognak ezek a célpontok megjelenni és mi az, amire figyelmet kell fordítani a játék során.

Ez az ablak 1 alkalommal ugrik fel szóval a játékosnak figyelmet kell fordítani a szövegre, hogy értse a játékot.

Ha a felvétel megtörtént és a játékos megnyomta az indító gombot, a játék visszaszámol és a játékos előtt elkezdenek megjelenni a céltáblák, amik már kilőhetők. Majd a játék végeztével egy szöveg tovább haladásra utasítja a játékost.

10. Ábra: 2. szint megjelenés illusztrálása

A célpont objektumok 2 fő elemből állnak össze:

1. A 3 dimenziós alap, ami a színt és a kinézetet biztosítja
2. Egy láthatatlan 2 dimenziós kör alakú tárcsa, ami az eredmények mentéséért felel

Azért volt szükség ezek elkülönítésére mivel a Unity 3 és 2 dimenziós vektorokat is tud kezelni, viszont azokat az objektum közepétől képes mérni. Az én esetemben az objektum, amin mérni akartam az 3 dimenziós volt, de csak 2 dimenziósan akartam az adatokat menteni róla (X és Y tengely mentén). Így egy tárcsa modellel oldottam meg úgynevezett vizuális „renderer” elem nélkül (tehát vizuális megjelenítés nélkül), így a találati pontot 2 tengely mentén tudtam mérni és menteni.

#### Játék logika kialakítása

A játék több fő mechanikából áll, amik összehangoltan tudják kiadni a teljes játékélményt. A kulcsmechanikák a következők:

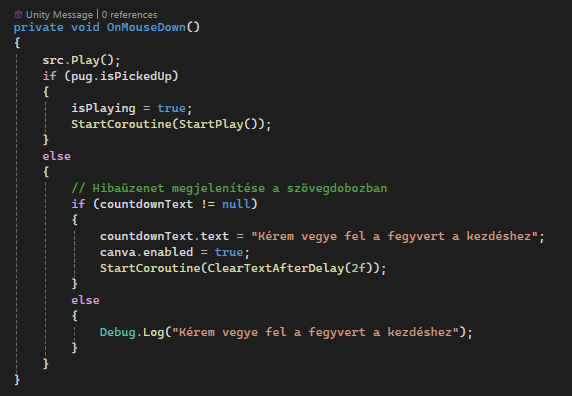
* Indító mechanizmus (a gomb segítségével)
* Lövési mechanizmus (bal kattintással)
* Célpontok megjelenítése (ciklikus generálással)
* Találati pont meghatározása és reakcióidő mérése
* Adattárolás találatkor (lövési szekciókkal)

Mivel a 2. és 3. szint is lövési metódusok által működik ezért ezek hasonló módon vannak scriptelve. A 2 játékot lövési szekciókkal külön választottam „Target” adatok és „Shooting” adatokra. A „Target” játék szimbolizálja a 2. szintet és a „Shooting” a 3. szintet. Minden játékosnál a játékos adatfelvitele után keletkezik az adatbázisban egy úgynevezett „Shooting Session” és a lövések után az adatok a megadott „Session ID” -val tehát szekció azonosítóval kerülnek mentésre. Így minden játékoshoz specifikusan köthető egy-egy generált adat.

Az indító mechanizmus egy gomb segítségével történik, ami egy script-el van ellátva. Mivel a 2 szint különböző módon kezdi el a ciklus generálást ezért az indító gomb kódjai is el vannak egymástól különítve és csak bizonyos funkciókban egyeznek meg.

Közös funkcióik:

1. Nem engedi a játékosnak elkezdeni a játékot amíg a fegyvert fel nem vette

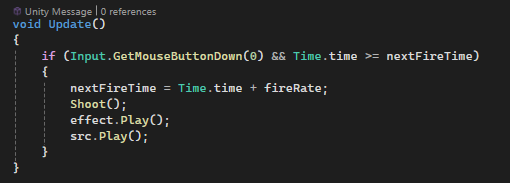
A játékos amint megpróbálja elkezdeni a játékot a nélkül, hogy felvette volna a fegyvert, egy megadott szöveg megjelentésével utasítja arra, hogy vegye fel a kezdéshez a fegyvert.

11. Ábra: indító gomb kód illusztráció

Ezért részben az „OnMouseDown ()” metódus felel, majd, ha megtörtént a felvétel, kezdődik a játék előtti vissza számlálás

1. Visszaszámlál a megnyomást követően a játék kezdése előtt

A kódnak a fejlesztő felületen referenciaként meg van adva egy üres szövegdoboz a megadott méretekkel, ide generálja a 3-ról indított visszaszámlálást majd miután végzett a számlálással eltünteti a szövegdobozt és indítja a játék célpontgenerálásának metódusát.

A lövés mechanizmusa a pisztoly modelljéhez rendelt „Gun.cs” nevű script-el működik. Ebben a változók definiálásán kívül 2 fontosabb metódus vezérli a lövést. Az „Update ()” metódus felel a játékos által végrehajtott lövési művelet kezeléséért. Ez a metódus minden egyes képkocka során automatikusan lefut a Unity motor működése közben, így válik teljesen alkalmassá arra, hogy valós időben figyelje a játékos bemeneti műveleteit, például a kattintásokat. Egyszerre figyeli a játékos műveleteit, szabályozza a tűzgyorsaságot, és elindítja a lövéshez kapcsolódó vizuális-hanghatásokat, ezek mellett a konkrét találati logikát.

12.ábra: „Update ()” metódus illusztráció

A lövés kulcsmechanikáját a „Shoot ()” metódus adja. A „Shoot ()” metódus a játék lövésmechanikájának központi eleme, amely minden lövés során felel a találat érzékeléséért, a találati pontok kiszámításáért, az adatok rögzítéséért és a célpont sebzéséért.

A metódus működése egy raycast (sugárkövetés) használatával kezdődik, amely a játékos nézőpontjából (a hozzá rendelt kamerából) előrefelé indul el, és meghatározott távolságon belül képes észlelni, hogy a játékos mire célzott. Amennyiben a sugár eltalál egy objektumot, a rendszer megpróbálja lekérni annak „Target” komponensét, amely azt jelzi, hogy az adott objektum sebezhető. Ha a célzott objektum közvetlenül nem rendelkezik ilyen komponenssel, a szülő objektum vizsgálatával próbálja megtalálni azt. Ez lehetővé teszi, hogy az összetettebb hierarchiában elhelyezkedő célpontokat is kezelje.

Miután az objektum eltalálásra kerül, a rendszer kiszámítja a találati pont helyzetét az objektum lokális koordinátarendszeréhez viszonyítva. Ez azt jelenti, hogy a találat az objektum középpontjához képest értelmezhető, így pontos információt ad arról, hogy hol érte a lövedék a célt. Az így kapott koordináták két tizedesjegyre kerekített double értékként kerülnek eltárolásra.

A „Shoot ()” metódus hozzáértően kezeli azt, hogy a játék éppen melyik szinten van. Ha az második szinten történik a lövés, akkor az Target adatbázis példányba kerülnek mentésre az új találati pontok, de csak abban az esetben, ha az adott pozíció még nem szerepel a korábban eltalált pontok listájában. Ez segít elkerülni a duplikált adatokat. A harmadik szinten, ahol a játék más típusú lövöldözési logikát alkalmaz, a „Shooting” adatbázis példány felel az adatok gyűjtéséért, és minden egyes találatot rögzít, akár többször is, valamint a Gun osztály saját „hitpoints” (tehát találati) listájába is elmentésre kerülnek ezek az értékek.

13. Ábra: „Shoot ()” metódus illusztráció

Amennyiben az eltalált objektum rendelkezik „Target” komponenssel, a metódus meghívja annak „TakeDamage ()” függvényét, amely a célpont életpontját csökkenti a „Gun” által definiált „damage” értéknek megfelelően. Ez biztosítja, hogy a lövés nemcsak vizuálisan jelenik meg, hanem hatással is van a játék világára.

Összességében a lövés teljes folyamatát átfogja:

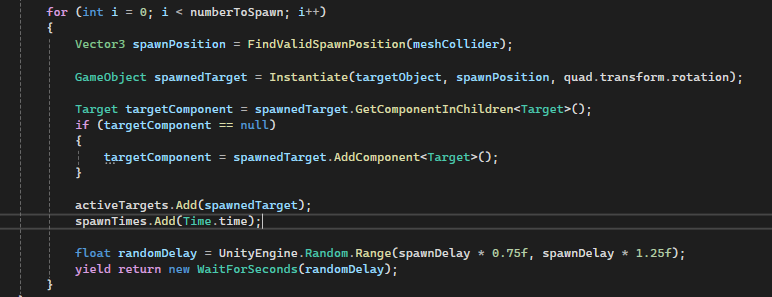
* Érzékeli a célzást és a találatot
* Kezeli az adatgyűjtést
* Alkalmazza a sebzést

A célpontok egy egyszerű „Target” nevű script-el vannak inicializálva, amiben 3 függvény van definiálva:

* „TakeDamage ()”, ami a sebzés fogadására szolgál
* „Die ()”, ami a target objektum találat utáni törlésére szolgál
* „HittedObject ()” ami a találatot szimbolizáló „bool” (logikai) változó visszaadására szolgál, úgynevezett „Getter” (lekérő) függvény

A target objektumok generálásáért 2 script felelős, a második szint esetében az ObjectSpawner a harmadik szint esetében pedig az ObjectSpawner\_1place. Az alapvető különbség ahogy fentebb is említettem, hogy a 2. szinten különböző helyekre jellennek meg a megadott számú célpontok míg a 3. szint esetében ez egy helyre való lehelyezéssel történik meg. Az osztályok fő célja, hogy a játék során meghatározott számú célpontot generáljanak egy meghatározott területen, és figyelemmel kísérjék azok elpusztítását. Emellett felelősek a találatok időzítéséért, az eredmények naplózásáért, valamint a játék végének kezeléséért is.

A második szint esetében az ObjectSpawner inicializálja a szükséges listákat és beállításokat. Első lépésként meghatározza, hogy hány célpontot kell generálni, majd a kijelölt területen belül megpróbál olyan helyeket találni, amelyek biztosítják, hogy a célpontok ne kerüljenek egymás mellé. Ennek ellenőrzésére egy algoritmust alkalmaz, amely figyelembe veszi a már meglévő célpontok pozícióját, és csak akkor helyez el egy újat, ha a generálás pozícióját nem találja túl közelinek a másikhoz. Ezáltal a játékosnak mindig egy átlátható és egyenletesen eloszló célmezőt biztosít.

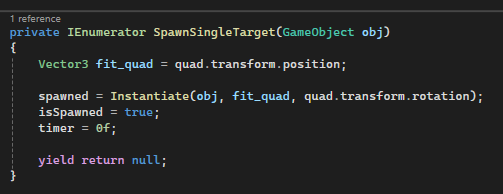
A játék során az osztály folyamatosan figyelemmel kíséri, hogy mely célpontok kerültek eltalálásra. Amint egy célpont megsemmisül, a rendszer azonnal rögzíti az eseményt, eltávolítja az aktív célpontok listájából, és eltárolja a találat időpontját. Ez az időmérés kulcsfontosságú, hiszen a játék során pontos statisztikák készülnek arról, hogy a játékos milyen gyorsan és pontosan reagál. Az adatok tárolása nem csupán a játékmenet során történik, hanem a rendszer azokat egy adatbázisba is elmenti, amely később visszakereshető és elemezhető.

14. ábra: ObjectSpawner célpont generálás implementálási részlet

Amikor a játékos az összes célpontot eltalálja, a játék véget ért. Ekkor megjeleníti a megfelelő értesítést, amely jelzi, hogy a szint teljesítve lett. Az értesítés egy meghatározott idő után automatikusan eltűnik, hogy a felhasználói élmény zökkenőmentes maradjon.

A harmadik szint egy speciális változata az alap ObjectSpawner osztálynak, amely a célpontokat nem véletlenszerű helyekre, hanem egy meghatározott pontban hozza létre. Ez a játékmód egy adott helyre koncentrálja a célpontokat, mint például egy lőtéren vagy egy célorientált gyakorlópályán, ahol a játékosnak ugyanarra a pontra kell fókuszálnia. Így ez a reakció idő mérésére tökéletesen alkalmas.

A működési elve a következő, inicializáláskor beállít egy adott koordinátát, amelyet az összes generált célpont használni fog. Mivel a célpontok egy helyen jelennek meg, az osztály nem használ térbeli ütközésvizsgálatot vagy egyéb elhelyezési logikát. Ennek eredményeként az osztály gyorsabb és kevesebb számítási erőforrást igényel, miközben biztosítja, hogy a célpontok mindig pontosan ugyanott jelenjenek meg.

Emellett az osztály felelős a célpontok követéséért és eltávolításáért is, az ObjectSpawner osztályhoz hasonlóan. Amint egy célpont megsemmisül, az osztály azonnal újat generál ugyanazon a helyen.

15. Ábra: ObjectSpawner célpont generálás implementálási részlet

A reakcióidő mérés miatt, rendelkezik beépített időmérési funkcióval, amely rögzíti, hogy mennyi idő telik el a célpont megjelenése és eltalálása között. Ezeket az adatokat a rendszer elmenti, így a játékos visszanézheti a teljesítményét, és elemezheti, hogy milyen gyorsan reagál a megjelenő célpontokra. Ezek mellett lehetőséget ad az adatok összevetésére más játékosokkal.

Amint a megfelelő számú objektum megjelent és kilövésre került, a kód értesítést küld a rendszernek, amely jelezheti a kör végét vagy a következő nehézségi szint kezdetét. Ez biztosítja, hogy a játék pontosan szabályozható maradhasson, és a játékos mindig egyértelmű visszajelzést kapjon a teljesítményéről.

### Labirintus (türelmi játék)

A navigációs képesség 3 dimenziósan kialakított térben önmagában is megpróbáltató feladat lehet annak, aki a mindennapjaiban nem találkozik ilyen jellegű játékokkal. A hagyományos 2 dimenziós játékokkal ellentétben sokkal nagyobb figyelmet igényel és fárasztóbb feladat lehet megtanulni. A kamera mozgatása elengedhetetlen, kézzel való komplex koordinációt igényel, ami nem csak az idősebbeknek tartogat kihívást. Ez a fajta térbeli gondolkodás, kihívást jelent, ugyanakkor rendkívül hasznos edzést nyújt az agy számára. Ezek a típusú játékok intenzíven fejlesztik a térbeli tájékozódás képességét, ami alapvető szerepet játszik az életünkben.

Sok már elkészült tanulmány és kutatás bizonyította, hogy hogyan segíti a 3D-s videojátékok használata a térbeli memória és navigációs képességek működésének fejlődését. Napjaink 3D-s játékaiban elengedhetetlen a vizuális térben való koordináció és a mentális térképezés, ezért aki gyakran játszik ilyen videójátékokkal az előbb hozzászokik egy másik játék irányításához is.

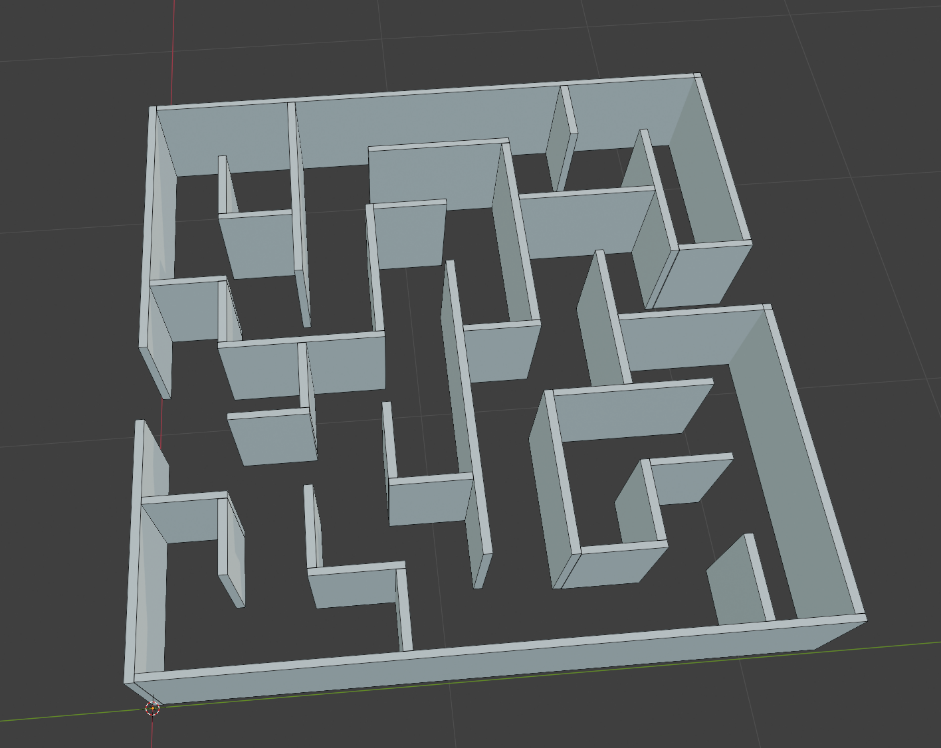
Ez a labirintus esetében az első cél az átjutás majd a feladat teljesítése. A játék során a játékos miután eljutott az útvesztő végére (amit külön színű fal jelez) besétálva elindít egy időzítőt, illetve beindítja a feladat nehézségéért felelő mechanizmusokat. 5 gombot jelenít meg elhelyezve a labirintuson belül, amiből 4 „rossz” és 1 „jó”. A rossz gombok megnyomásakor a gomb eltűnik és az időzítő megy tovább. Jó gomb megnyomásakor az időzítő megáll és a játék szöveges visszajelzést küld a játék végét jelezvén.[15][16]

#### Vizuális kialakítás

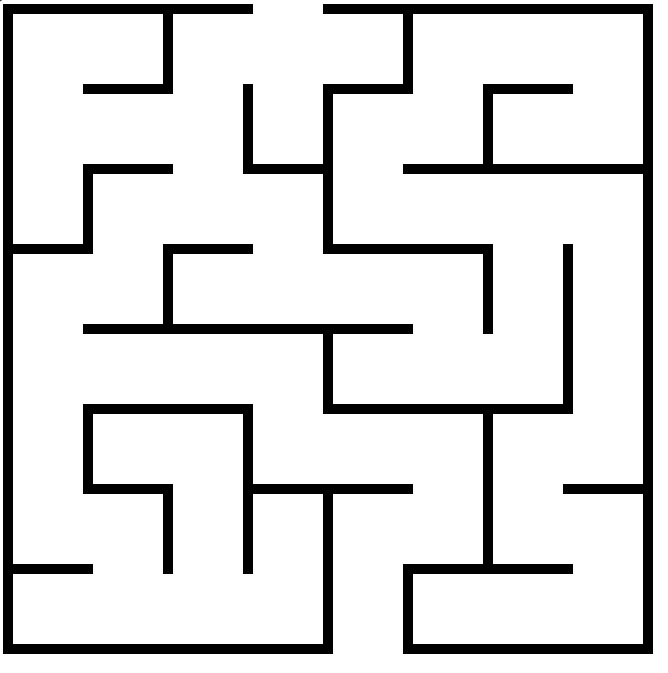
A játék 2 fő vizuális elemből áll, a labirintusból és a szobából, ami minden eddigi játéknak része volt. A labirintus generálására több lehetőség közül én a statikus verziót választottam és saját kezűleg készítettem el a Blender nevű 3 dimenziós tervező szoftverben.

A Blender egy ingyenes nyílt forráskódú szoftver, amit virtuális tárgyak, szobák, karakterek modellezéséré és animálására használnak. De ezeken felül még videókészítésre, textúrázásra, játékszoftverben használt vizuális effektek készítésére is tökéletesen megfelel. A program letisztult kezelőfelülettel rendelkezik, ami a laikus szem számára összetettnek tűnhet viszont kellő gyakorlással és tanulással csupán saját idő és energiaráfordítással megtanulhatóak a funkciói. Ezek mellett nagyon támogató közösség van a háttérben, ami lehetővé teszi bárkinek, hogy igénybe vegyék az online felületeken lévő ingyenes oktatóanyagokat vagy kurzusokat, amik egy tanulási folyamat során bemutatják akár a teljes modellkészítési folyamatot vagy egy tanulási görbén végig vezetve megtanítják az ilyen jellegű munkához szükséges alapokat. A Blender legjobban a „kreatív műhely” kifejezéssel írható le, mivel teret ad az ötletek kibontakoztatására, legyen szó egyszerű modellezésről vagy komplex animációs projektekről.

Versenytársai között szerepel az Autodesk Maya, a 3ds Max, a Cinema 4D, vagy éppen a ZBrush. Ezek a szoftverek régóta jelen vannak a professzionális filmiparban, animációs stúdiókban és játékfejlesztő cégeknél. Ezek a szoftverek összetettebb működéssel bírnak, elsajátításuk nehezebb ennek megfelelően az árazásuk is magasabb. Ezeknek megvannak a maguk előnyei. Például a ZBrush jobban specializálódott a részletes, szobrászati munkákra, a Maya pedig a filmes animációknál számít ipari szabványnak. A Cinema 4D-t gyakran használják mozgóképes grafikai projektekhez, például reklámfilmek készítéséhez.

Mikor kiválasztottam a szoftvert nekiálltam a tervezési feladatnak és kidolgoztam egy olyan labirintus modellt, ami nem okoz túl nagy kihívást, de tud fejtörést okozni a tapasztaltabb játékosoknak is.

16. Ábra: Labirintus modell illusztráció

A modellt egy „.svg” (Scalable Vector Graphics) kiterjesztésű vektorgrafikus fájlból készítettem, ami alapvetően csak a vonalakat tartalmazta. Ezt a kiterjesztésű fájlt weboldalak esetében szoktál előszeretettel alkalmazni mivel az .svg fájlok különlegessége, hogy vektoros alapúak, nem pixelekből, hanem matematikai alakzatokból állnak. Ezáltal akármekkora méretre nagyíthatók, minőségromlás nélkül.

17. Ábra: Labirintus alapjául szolgáló vektorgrafika illusztráció

A Blender-be való importálás után úgynevezett matematikai (Bezier) görbék jönnek létre, amiket extrudálással (tehát térbeli kinyomással) 3 dimenzióssá lehet alakítani. Így ezek után minimális szépítgetéssel és a megfelelő helyeken való kibővítések tökéletesen reprezentálni tudta az elképzeléseimet.

#### Játék logika kialakítása

A játék teljesíthetőségének logikája szintén 2 részre bontható:

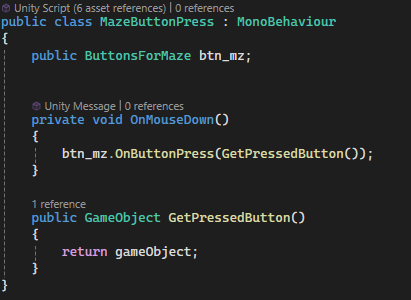
1. A játékosnak el kell érnie a labirintus végére, hogy elindítsa az időmérő mechanizmust és megjelenjenek a gombok
2. A gombok megjelenése után a játékosnak meg kell keresnie az 5 közül a „jó” gombot valahol elrejtve a többi között a labirintusban, ami leállítja az időzítőt és lezárja a játékot

A labirintus elkészítése mellett a gombok elkészítése gyorsan megtörtént, minden gomb jól elkülöníthető színt kapott, amivel láthatóvá válik a labirintus falain. Nincs különbség a „jó” és „rossz” gombok kinézete között így ránézésre nem lehet megállapítani éppen melyiket találta meg az ember.

A gombokat mindig megadott helyre generálja a játék és 3 script segítségével funkcionálja őket:

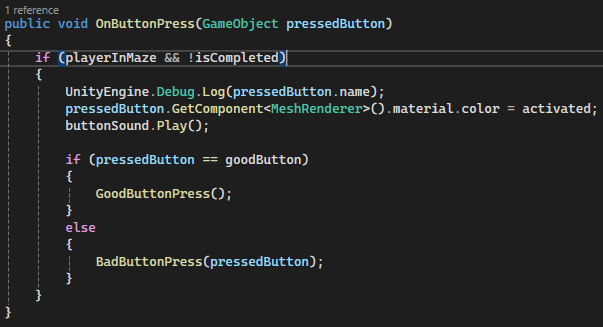
* MazeTrigger: ez a script felel a játék elindításáért, amint a játékos eljutott a labirintus végére, elindítja a játékot és az időzítőt

18. Ábra: MazeTrigger osztály illusztráció

* MazeButtonPress: egy egyszerű 2 funkciós osztály, ami a gombok megnyomását kezeli, van benne egy „getter” (lekérő) metódus, ami a megnyomott gombot adja vissza és egy gomblenyomást aktiváló „OnMouseDown ()” függvény

19. Ábra: MazeButtonPress osztály illusztráció

* ButtonsForMaze: ez egy összetettebb kód, ami a 3 osztály összehangolásáért felel, ellenőrző metódusokból és visszajelző funkciókból áll

A ButtonsForMaze tartalmazza az „OnButtonPress ()” funkciót, amit a láthatunk a gombnyomás funkciójában meghívni. Gyakorlatilag ez kezeli a játék lezárását. Ha a gomb lenyomására a lekérés a „jó” gombot adja vissza a játék lezárul és visszajelzést küld a játékosnak, hogy megtalálta a megfelelőt, kiírja a végső idejét, gratulál, átváltja a gomb színét zöld színűre és tovább irányítja. Ha viszont a „rossz” gombot nyomja meg a gomb egyszerűen eltűnik és az időzítő megy tovább. Gomblenyomás előtt még végrehajt egy ellenőrzést, hogy a játék éppen futási állapotban van-e, ez a korai fejlesztési szakaszok alatt szükséges volt a mechanikák megfelelő működésének ellenőrzése végett.

20. Ábra: „OnButtonPress ()” függvény illusztráció

A fejlesztés során úgy készítettem el a játék gombjait, hogy azok mennyisége bővíthető legyen. A változók inicializálásánál a kód a rossz gombokat egy tömb formájában definiálja, ezzel lehetővé téve, hogy ha nagyobb labirintusba kerülne a játék, több gombot el lehessen helyezni, ami nyilván nehezítené a mostani formulát. Mivel a játék lezárásáért szolgáló „jó” gombból csak 1 létezik ezért az sima „GameObject” -ként van definiálva.

## Adatbázis tervezés

A játékosok előrehaladásának, beállításainak, statisztikáinak és más egyéni adatainak tárolása kulcsfontosságú része egy modern videojátéknak. Az alábbiakban módszerek szoktak előfordulni a tervezési folyamatban:

* Lokális fájl alapú mentés
* Lokális adatbázis alapú mentés
* Online szerver alapú mentés
* PlayerPrefs / Registry alapú mentés

Ezek egyaránt alkalmasak az adatok mentésére, viszont más-más technológiát és megvalósítást igényelnek platformtól függően. A projekt esetében az első kettő (Lokális fájl és adatbázis) opció volt alkalmas, mivel mind a kettő könnyen kezelhető, SQL (strukturált lekérdezőnyelv) alapú és átlátható struktúrát biztosít az azonosító alapú adatmentéshez.

### Lokális fájl alapú adatmentés

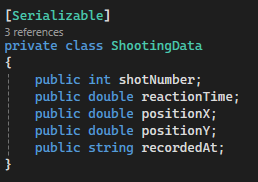
Játékfejlesztéskor az egyik legegyszerűbb és legkézenfekvőbb módszer a játékos adatok mentésére a lokális fájlalapú mentés. Ez gyakorlatilag azt jelenti, hogy a játék a játékos adatait például pontszámokat vagy statisztikákat közvetlenül az adott eszközön tárolja, ahol a játék fut. Nincs szükség internetkapcsolatra vagy külső szerverre.

A leggyakrabban használt fájlformátumok közé tartozik a JSON, az XML és a sima szövegfájlok (TXT). Számomra a JSON formátum volt a legszimpatikusabb, mivel egyszerre olvasható és jól strukturált későbbi adatfeldolgozáshoz.

A lokális fájlalapú mentés előnye az egyszerűségében rejlik mivel, gyorsan megvalósítható, nincs szükség külső infrastruktúrára, és a mentés/feltöltés műveletek pillanatok alatt elvégezhetők. Ezek mellett azonban fontos figyelembe venni a korlátait is, mivel az adatok csak azon az eszközön érhetők el, ahol mentés történt, és mivel a fájlok fizikailag hozzáférhetők a felhasználó számára, nem védettek a manipulációval vagy törléssel szemben.

A játékban az ilyen fajta adatmentést csak biztonsági mentés funkciójára használom. A játékok végén nem csak az adatbázisba mentődnek le a végleges adatok, hanem egy „PlayerDataBackup” nevű vészhelyzeti mappába 2 formában:

* egy „AllPlayerData.json” nevű fájlba mentésre kerül minden játékos adata
* minden játékosnak keletkezik egy egyedi nevű JSON fájl, ami tartalmazza a saját adatait és eredményeit

Kódban ez egy script alapján van kezelve, ami strukturálja és JSON formává alakítja az alapvetően elmenteni kívánt adatokat. A kódon belül minden adat osztály formájában létre van hozva és egy megadott elérési út mentén mentésre kerül.

21. Ábra: példa a definiált osztályra

### Lokális adatbázis alapú adatmentés

A játékfejlesztés során a biztonsági mentésen kívül szükség van egy átlátható és hatékonyan kezelhető adattárolási módszerre is. Erre a célra kiváló a lokális adatbázisba történő mentés, esetemben az SQLite használata, a könnyű kezelhetőség és professzionális adatkezelés miatt.

Az SQLite egy beágyazott, fájlalapú adatbázis-kezelő rendszer, amely nem igényel külön szervertelepítést vagy hálózati kapcsolatot mégis egy teljes értékű SQL adatbázist biztosít. A fejlesztés során ez lehetővé tette, hogy a játékos adatait például a statisztikákat, elért eredményeket, strukturált, relációs formában tároljuk.

Ezt a fajta adatbázist könnyen kezelhetjük az SQLite DB Browser nevű grafikus felülettel, amely lehetővé teszi az adatok vizuális szerkesztését, lekérdezését és ellenőrzését. Ez rendkívül fontos és hasznos a fejlesztési és tesztelési szakaszban, mivel így minden átlátható és jól szerkeszthető formában elérhető.

Ez az adatbázis jól kezeli a nagymennyiségű, összetett adatokat és képes kapcsolatok, lekérdezések, szűrések és rendezések kezelésére. Ezek mellett biztonságosabb, mint a fájlalapú mentés, mivel az adatok nem közvetlenül olvashatóak szöveges formában, így nehezebb őket manipulálni vagy meghamisítani.

Főbb előnyei, ami miatt ezt a módszert választottam:

* Strukturált adattárolás relációs rendszerben
* Egyszerű integráció játékmotorokkal, például Unity-vel
* Nagy mennyiségű adat gyors kezelése és lekérdezése
* SQLite DB Browser révén könnyen szerkeszthető, vizsgálható
* Nincs szükség külön szerverre mivel minden egy fájlban tárolódik
* Stabil és megbízható, még komplex adatszerkezetek esetén is

A statisztika készítése során ezzel a technológiával könnyen ki tudtam nyerni az adatokat és kezelni tudtam őket.

## Szintek közötti váltás

Miután elkészültek a szintek szükség volt egy logikára, ami a játék linearitását képviseli, tehát nem egymástól elkülönülő szintekként jelennek meg a pályák, hanem dinamikusan követik egymást és egy fluid élményt biztosítanak az éppen játékban lévő játékosnak.

Az egyszerű lineáris videójátékok olyan játékélményt kínálnak, ahol a játékos előre meghatározott útvonalon haladhat végig, szintenként strukturálva. A lineáris felépítés egyik legfontosabb tervezési aspektusa a szintek közötti fluid, zökkenőmentes átjárás biztosítása. Ez nem csak technikai megoldás, hanem kulcsfontosságú a játékos élményének fenntartása szempontjából: egy jól áramló játékmenet segíti a beleélést, csökkenti a frusztrációt, és fenntartja a játékos motivációját.

Ez nyújt lehetőséget a fejlesztőknek az iránymutatásra is. A szintek logikus és gördülékeny összekötése megmutathatja, merre érdemes haladnia, mit kell elérnie, és milyen kihívások várnak rá a következő szakaszban. Ezáltal a játékos sosem érzi elveszettnek magát, és mindig van világos célja.

Ebben az esetben a különböző jelenetváltásokkal nem kellett foglalkoznom mivel a 4 szintet egy jeleneten belül helyeztem el. Ezáltal viszont szükség volt egy folyamatra, ami által ellenőrizni tudom a következő kérdéseket:

1. Elindította-e a játékos a szinten lévő játékot?
2. Teljesítette-e már a feladatot, amit a szint képviselt?
3. Ha teljesítette mentődtek-e az adatok az adatbázisba?
4. Ha elmentődött minden tovább tud-e jutni a játékos a következő szintre?
5. Egyértelmű-e a tovább haladás jelzése?

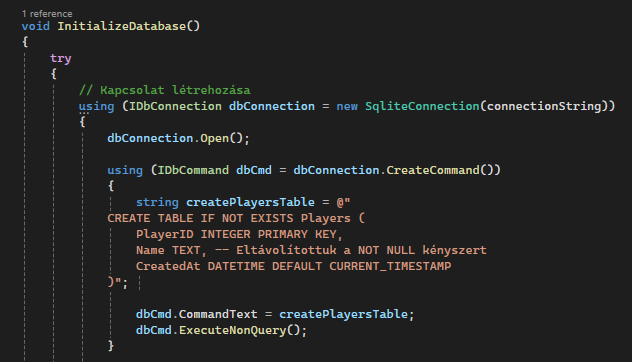
A kérdések megválaszolásakor minden adott lesz arra, hogy a játékos tovább tudjon jutni és egyértelmű legyen a tovább haladás.

### Szint indítás és teljesítés ellenőrzése

Ahogy azt fentebb a szintek logikai tervezésekor bemutattam minden szint előtt egy moduláris ablak gondoskodik a játékos iránymutatásáról. Adott szintlogikák elindításához pedig az első három esetben indítógombok biztosítják a játék kezdetét míg a labirintus esetében az első alkalommal való labirintus végébe történő belépés. A kódokban definiáltam logikai változókat, amik igaz vagy hamis értékeket vehetnek fel, általában egyértelmű névvel ellátva például „isPlaying”. A játék indítása előtt ezek a változók hamis értéken vannak definiálva majd a játék indításakor (tehát például a „start” gomb megnyomásakor) igaz értékre váltanak.

Ez a változó által folyamatosan meghatározható, hogy a játékos mikor van játékban és mikor teljesítette azt. Kódolásban ezáltal szabályozhatóvá válok az, hogy amikor a játékos játékban van ne tudjon új játékot indítani, illetve, hogy ha befejezte azt küldjön visszajelzést a tovább haladásról a játékosnak.

### Eredménymentés a játék végével

Az eredmények mentését a játék során egy üres objektumra húzott script vezérli, amiben a teljes mentés logikája definiálva van. A kód elején végbe megy egy ellenőrző metódus, ami az adatbázis lokális mappáját ellenőrzi, hogy létezik-e az adatbázis fájlja. Ha nem akkor létrehozza az inicializáláskor ezek mellett a metódus során a táblákat külön létrehozza minden játéktípusra.

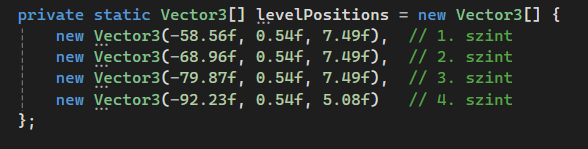
22. Ábra: Inicializálás metódusából illusztrációs részlet

Miután a játékokat teljesítette a játékos, végbemegy egy mentés az adatbázisba az éppen játékban lévő azonosítóval.

### Szintek közötti tovább haladás

Mivel a szintek egy „scene” -en belül vannak elhelyezve az adott játék végével meg kellett oldanom az átjárási lehetőséget.

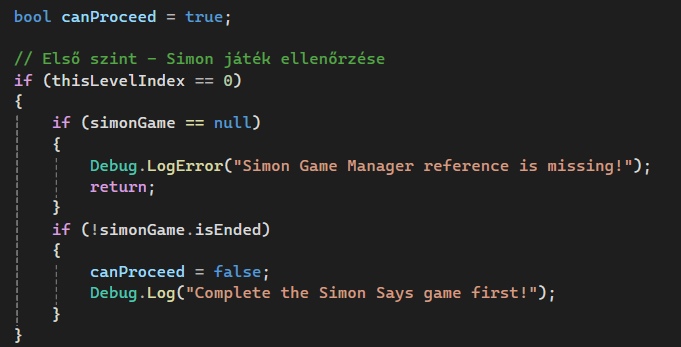
A fenti ellenőrzések és adatmentések után a játékos előtt felugrik egy szöveg, ami a tovább haladásra utasítja, majd megadott másodperc határon belül eltűnik. A szöveg pontos tartalma az, hogy „Eredményeit mentettük! Tovább haladhat a mögötte lévő ajtónál.”. Majd, ha ezután megfordul a játékos láthatja az ajtót a szoba másik végén. Erre az elemre való utalást a súgó gomb megnyomásánál is találhat.

Minden szinten elhelyezett ajtóhoz, társul egy Collider, ami egy script segítségével tovább haladást biztosít, ha az előzetes feltételek teljesültek (tehát a játékos teljesítette a szinten lévő játékot). Ezeket a fejlesztés során „NextGameCollider” -nek neveztem el, ami utal a feladatára is. A hozzárendelt script-ben el vannak tárolva az ajtó előtti pontos 3 dimenziós pozíciók vektorosan, olyan formában, hogy minden szinthez egyedi azonosító van rendelve így a játék valós időben látja, hogy a játékos jelenleg melyik szinten tartózkodik és melyik szintre szeretne átmenni.

23. Ábra: illusztráció a szintek pozíciójának tárolásáról

Az érintkezési pillanatának állapotától függően 2 dolog történhet:

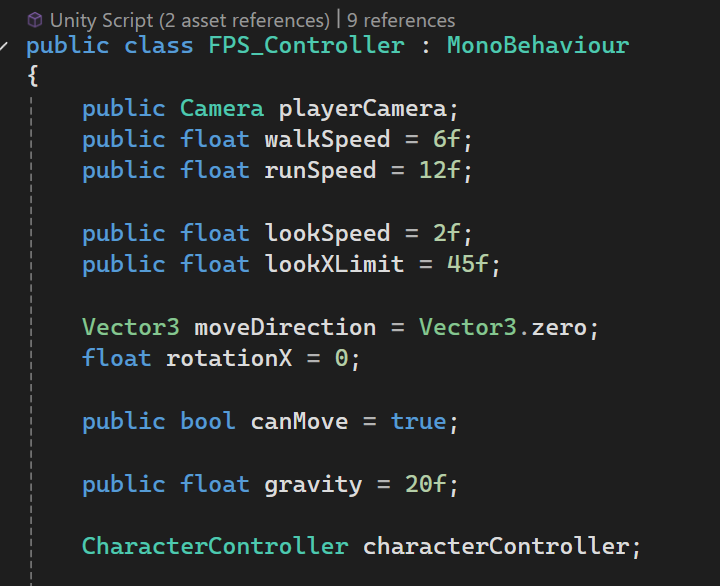
1. Ha minden feltétel teljesült, ellenőrzi éppen melyik szinten tartózkodik a játékos és az utána következő szint szobájába teleportálja át
2. Ha a feltételek nem teljesültek, visszalöki a játékost és jelzi, hogy még nem teljesítette a szinten lévő feladatot

Ezt egy úgynevezett „OnTriggerEnter ()” nevű függvény vezérli, ami állapotok átállításával tudja engedni vagy megtiltani az átjárást a játékosnak.

24. Ábra: illusztráció az „OnTriggerEnter ()” függvényből

## Alapvető játékos irányítás

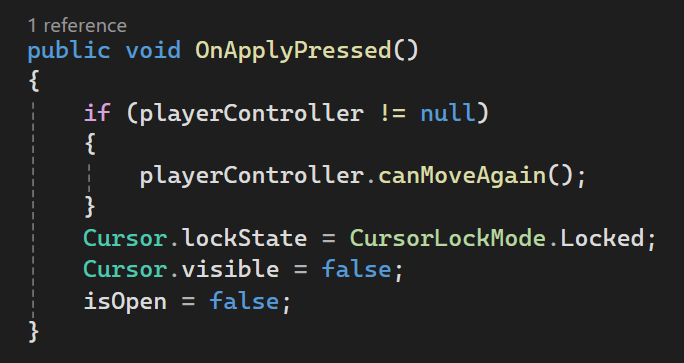
Kezdetben egy mozgási funkciót szerettem volna adni a játékosnak, így kiválasztottam egy nekem szimpatikus 3D elemet, adtam hozzá megfelelő fizikai beállítást és „Collider” - eket, hogy érintkezni tudjon a környezeti elemeivel majd egy script segítségével megadtam neki az alapvető irányító billentyűket, amik mozgásnál az én esetemben a „WASD” és a nyilak voltak, interaktáláskor a „bal kattintás” - al érzékelt a program és a „LEFT SHIFT” -el egy futás opciót is adtam a játékosnak.

A script írásakor figyelembe kellett vennem a fizikai hatásokat a játékos objektumra, illetve az ütközést a különböző objektumokkal. A változók meghatározásakor ez is kulcsfontosságú volt.

25.ábra: FPS\_Controller script részlet

A kamera objektum alapvetően „playerCamera” néven kerül inicializálásra, ez a jelentben lévő egyetlen olyan objektum, ami kamera szerepet tölt be, illetve köze van a karakter közvetlen irányításához. A játékos karakteren ezen kívül még úgynevezett kapszula elem van, ami a 3D-s környezetben való fizika, illetve érzékelés miatt kulcsfontosságú. Az objektum önmagában nem lenne elég mivel a környezete nem látja, ezért tartalmaznia kell egy „Mesh Renderer” -t, ami a megjelenítéséért felel, illetve egy „Capsule Collider” - t, ami pedig a fizikai hatásokat teszi észlelhetővé (például az ütközést).

Az alap értékek meghatározásakor fontos volt még a képen látható „walkSpeed” és „runSpeed”, ami a mozgás sebességét hivatott meghatározni. Ahogy a „lookSpeed” és a „lookXLimit” pedig a körbetekintésnek a paramétereit határozzák meg, pontosabban a sebességét és az arányát.

A további kódok szempontjából fontos volt még a „canMove” változó, ami engedélyezni, illetve letiltani tudja a játékos mozgását. Ennek az átállítása csak akkor fontos amikor olyan környezeti hatás történik a játékossal, mint például egy oktató ablak felugrása. Az ablak felugrása után szükséges idő a megértéshez és e közben a mozgás engedélyezése elterelhetné a figyelmet a szöveg elolvasásáról. Így ezekben a pillanatokban letiltásra kerül a mozgás addig amíg az ablak aktív (tehát a játékos olvassa azt).

26.ábra: részlet TriggerZoneForWindow script részlet

Ebben a script részletben pont ennek az alkalmazására lehet látni egy példát. A kód önmagában egy üres objektumhoz van kapcsolva, amin kizárólag egy „Box Collider” található, aminek testreszabott szélességi és hosszúsági paraméterei vannak.

Feladatát tekintve arra való, hogy ha a játékos ütközik ezzel a láthatatlan dobozzal, feldob a program egy úgynevezett moduláris ablakot, ami információkat tartalmaz. Ez egy grafikai megjelenítő segítségével a háttérben inaktív egészen addig a pillanatig amíg nem találkozik a játékos a zónával.

A kód ezen része egy függvény, ami az akkor hívódik meg amikor a játékos rányom a „Rendben” gombra, ami bezárja a moduláris ablakot. Elsőnek ellenőrzi, hogy létezik-e a játékost irányító script az adott pályán, ez hibakezelési szempontból fontos. Amikor érzékelte a játékost, mint elemet és a hozzá tartozó „script” - et, azonnal meghívja a „CanMoveAgain ()” függvényt, ami annyit tesz, hogy a logikai változó értékét visszaállítja „true” tehát igaz értékre. Majd alaphelyzetbe állítja a kurzort, hogy a játék további részében ne zavarja a játékmenetet.

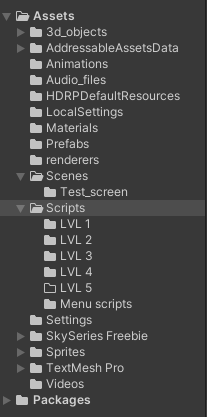
Annak érdekében, hogy az irányítás mindenki számára könnyen érthető és átlátható legyen, elhelyeztem egy úgynevezett „súgó” gomb funkciót, ami a játék bal alsó sarkában van vizuálisan jelezve. A játékos a „T” (mint tutorial) gomb megnyomásával elő is hozhatja az erre a célra összerakott moduláris ablakot, ami tartalmazza az irányítással kapcsolatos információkat és a tovább haladási lehetőséget képes formában jelezve.

27. Ábra: illusztráció az irányítást segítő Súgó funkcióról

## Mappastruktúra összeállítása

A mappák elrendezése a játék fejlesztésének fontos aspektusa. A különböző elemek későbbi átláthatósága miatt olyan rendszert kell felépíteni, ami később is konzisztens tud maradni, még ha 1000-es nagyságú is a projekt importált vagy létrehozott objektumainak nagysága. A struktúra meghatározásakor mindegyik vizuális vagy logikai elemnek külön rendszert kell biztosítani, hogy a növekedés során követhető maradjon és ha változtatásra van szükség ne vegyen el felesleges időt a keresgélés. Ezek mellett, ha csoportmunkáról beszélünk szintén fontos a felépítés mivel a programozók, fejlesztők, modellezők vagy 3D modellezők aszinkron munkája miatt, ha valaki eltér a rendszerezési normától könnyen előfordulhat keveredés és ez a sprint hetek esetében idő által behatárolt munkánál frusztráló lehet.

A számomra fontos alapvető mappák a következőképpen kell, hogy elemeket tároljanak:

* 3D elemek mappája (modellek struktúrája)
* Script-ek mappája (kódok struktúrája)
* Rendering mappa (fejlesztőkörnyezet vizuális megjelenítő objektumai)
* „Scene” -ek mappája (szintek struktúrája)
* Sprie-ok mappája (vizuális elemek struktúrája)
* Material-ok mappája (modellekhez tartozó vizuális anyagok struktúrája)
* Audio fájlok mappája (hanganyagok struktúrája)
* Package-ek mappája (külsőleg importált objektumok)

28.ábra: Projekt mappastruktúrája

Ezek által tökéletesen el lehet tájékozódni, ha új elem beillesztésére esne sor vagy egy már felhasznált elemen kellene finomítani. A scripteket szintenként külön mappába tettem mivel így a logikák nem keverednek és egy-egy játékelemhez könnyebben megtalálom a hozzá tartozó script fájlt.

# Statisztikai eredmények

Irodalomjegyzék



|  |  |
| --- | --- |
| [1] | Bene Ágnes, Móré Marianna, Zombory Júlia "A digitalizáció néhány elemének időseket érintő hatásai - karantén előtti helyzetkép", 2020 |
| [2] | M. Esther del Moral Pérez, Alba P. Guzmán Duque and L. Carlota Fernández García "Game-Based Learning: Increasing the Logical-Mathematical, Naturalistic, and Linguistic Learning Levels of Primary School Students", 2018 |
| [3] | Vinton Cerf archív profilja [Online]  Avaible: https://shorturl.at/c5OMh |
| [4] | Illéssy Miklós és Huszár Ákos " Technológiai fejlődés és munkaerőpiac: hogyan hat az automatizáció a munkahelyekre Magyarországon?", 2022 |
| [5] | Központi Statisztikai Hivatal (KSH) 1.23 Digitális ismeretek statisztika [Online] Avaible: https://www.ksh.hu/ffi/1-23.html |
| [6] | GameAce Game Studio, "Game Prototyping Essentials: Everything You Need to Know", 2024 [Online]  Avaible: https://game-ace.com/blog/game-prototyping/ |
| [7] | Yuriy Denisyuk, Pingle global game development partner, " Why is prototyping important for the game development process?", 2023 [Online]  Avaible: https://pinglestudio.com/blog/full-cycle-development/game-prototyping |
| [8] | Fact Monster, "Simon Says," 2024. [Online].  Available: <https://www.factmonster.com/games/simonsays> |
| [9] | E. K. Michael and D. A. Balota, "Simon says: Reliability and the role of working memory and attentional control in the Simon task," Journal of Experimental Psychology: Learning, Memory, and Cognition, vol. 38, no. 6, pp. 1637–1651, 2012. [Online].  Available: https://doi.org/10.1037/a0029143 |
| [10] | C. Wang, X. Liu, J. Liu, and B. Wang, "The link between parent–child relationship and learning engagement among adolescents: The chain mediating roles of learning motivation and academic self-efficacy," Frontiers in Education, vol. 7, 2022.[Online].  Available: https://www.frontiersin.org/articles/10.3389/feduc.2022.854549/full |
| [11] | L. E. Jarrott and C. Smith, "Implementing reverse mentoring to address social isolation among older adults," Journal of Gerontological Social Work, vol. 61, no. 5,pp.513–525,2018.[Online].  Available: https://www.tandfonline.com/doi/abs/10.1080/01634372.2018.1448030 |
| [12] | L. Konzack, "Pan European game information (PEGI) system," in Encyclopedia of Video Games: The Culture, Technology, and Art of Gaming, vol. 2, pp. 474–476, 2012. |
| [13] | L. Dogruel and S. Joeckel, "Video game rating systems in the US and Europe: Comparing their outcomes," International Communication Gazette, vol. 75, no. 7, pp.672–692,2013.[Online].  Available: https://doi.org/10.1177/1748048513482540 |
| [14] | G. L. West, B. L. Drisdelle, K. Konishi, J. Jackson, P. Jolicoeur, and V. D. Bohbot, "Habitual action video game playing is associated with caudate nucleus-dependent navigational strategies," Proceedings of the Royal Society B: Biological Sciences, vol. 284, no. 1856, p. 20162762, 2017. [Online].  Available: https://doi.org/10.1098/rspb.2016.2762 |
| [15] | G. D. Clemenson and C. E. Stark, "Virtual environmental enrichment through video games improves hippocampal-associated memory," The Journal of Neuroscience, vol. 35, no. 49, pp. 16116–16125, 2015. [Online].  Available: https://doi.org/10.1523/JNEUROSCI.2580-15.2015 |

Mellékletek

Ábrajegyzék

**Nincs ábrajegyzék-bejegyzés.**

Táblázatjegyzék

[1. táblázat (forrás megjelölésével) [2] 1](#_Toc97890942)

1. A megfelelő rész aláhúzandó. Amennyiben a válasza NEM, akkor a következő bekezdést és táblázatot törölje a dolgozatból. [↑](#footnote-ref-1)