Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Általános Informatikai Intézeti Tanszék

A képen szimbólum, Betűtípus, embléma, szöveg látható

Automatikusan generált leírás

**Android alapú rendszer szoftver illetve hardver tulajdonságainak**

**megjelenítése Flutter segítségével**

Szakdolgozat

**KÉSZÍTETTE**:

Bakos Kristóf István

IRF1JB

BSc mérnökinformatikus hallgató

**Témavezető**:

Dr. Mileff Péter

Miskolc, 2024

A képen vázlat, szimbólum, embléma, clipart látható

Automatikusan generált leírás**Miskolci Egyetem Gépészmérnöki és Informatikai Kar**

**Általános Informatikai**

**Intézeti Tanszék**

3515 Miskolc-Egyetemváros

Szak**: Mérnökinformatikus** Szakdolgozat azonosító:

Szakirány: Informatikai rendszermérnök

**Intézmény azonosító: FI 87515**

**SZAKDOLGOZAT FELADAT**

**Bakos Kristóf Isvtán**

BSc mérnökinformatikus hallgató részére

A tervezés tárgyköre: **Informatikai rendszermérnök**

A feladat címe: **Flutter alkalmazás fejlesztése**

**A feladat részletezése:**

1. A szakdolgozat célja egy Flutter alkalmazás készítése, amely bármely Android alapú rendszer szoftver illetve hardver tualjdonságait részletesen ismerteti egy felhasználóbarát felületen.
2. Funkciók megvalósítása:

* Az adatok megfelelő és pontos lekérdezésének biztosítása
* Grafikus felület megtervezése
* Animációk, felhasználóbarát környezet megvalósítása
* Optimalizálás, személyre szabhatóság biztosítása

Tervezésvezető: Smid László

Tanszék, beosztás:

Általános Informatikai Intézeti Tanszék mesteroktató

Konzulens(ek): Cég, beosztás:

A szakdolgozat kiadásának időpontja: 2023.09.25.

A szakdolgozat beadásának határideje: 2023.11.17.

Miskolc, 2023.11.17.

**Mileff Péter**

mesteroktató

1. A szakmai gyakorlat helye:

2. A szakmai gyakorlat vezetőjének neve:

3. A szakdolgozat módosítása: szükséges (a módosítást külön lap tartalmazza)

nem szükséges (a megfelelő rész aláhúzandó)

Miskolc,

tervezésvezető aláírása

4. A tervezést ellenőriztem: (1)

(2)

(3)

(4)

dátum, tervezésvezető aláírása

5. A szakdolgozat beadható

nem adható be

Miskolc,

konzulens aláírása tervezésvezető aláírása

6. A szakdolgozat ….. szövegoldalt,

….. db rajzot,

….. db CD mellékletet

….. egyéb mellékletet tartalmaz.

7. A szakdolgozat bírálatra: bocsátható

nem bocsátható

A bíráló neve, címe:

Miskolc,

tanszékvezető aláírása

8. Osztályzat: a bíráló javaslata:

a tanszék javaslata:

a Záróvizsga Bizottság döntése:

Miskolc,

a Záróvizsga Bizottság elnökének aláírása

**EREDETISÉGI NYILATKOZAT**

Alulírott Bakos Kristóf István; Neptun-kód: IRF1JB a Miskolci Egyetem gépészmérnöki és Informatikai Karának végzős mérnökinformatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy a Flutter alkalmazás fejlesztése című szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom

felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;

- tartalmi idézet hivatkozás megjelölése nélkül;

- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc,.............év ………………..hó ………..nap

…….……………………………….

Tartalom

[Bevezető 5](#_Toc166568925)

[-A játékipar kezdete és a kiemelkedő játékok 5](#_Toc166568926)

[-mostani telefonos és játékok piaci helyzete 7](#_Toc166568927)

[-Napjaink fontosabb technológiai 8](#_Toc166568928)

[Ray tracing 8](#_Toc166568929)

[Procedurális generálás 9](#_Toc166568930)

[Mesterséges Intelligencia használata a játékokban 10](#_Toc166568931)

[Cloude Gaming napjainkban 11](#_Toc166568932)

[-VR 12](#_Toc166568933)

[Napjainkban használatos fejlesztői környezetek 12](#_Toc166568934)

[Unity 12](#_Toc166568935)

[Unreal Engine 5 13](#_Toc166568936)

[Roblox 14](#_Toc166568937)

[Játék fejlesztés Android és Google háttérrel 14](#_Toc166568938)

[-HTML5-ben játékfejlesztés-> IOS->OpenGLES 15](#_Toc166568939)

[Meghatározott terveim a sasját játékkal szemben 15](#_Toc166568940)

[Elindulás, kezdetek bemutatása 15](#_Toc166568941)

[A projekt jellenlegi állapota(kódbemutatása) 15](#_Toc166568942)

[További fejlesztési lehetőségek (1 oldal) 15](#_Toc166568943)

[Angol Magyar összefoglaló 16](#_Toc166568944)

[Referenciák (10 darab minimum) 16](#_Toc166568945)

# Bevezető

-saját nézőpontomból

Tartalmi rész

# -A játékipar kezdete és a kiemelkedő játékok

Amikor a modern számitógépes rendszereken történő játékokról kezdünk el beszélgetni elsőnek fontos megemlíteni az ezen a szakmai részen történő első lépéseket, amelyek még 1962-ben kezdődött a Spacewar nevezetű játékkal, amit a Massachusett-i egyetemen alkottak meg. Ez volt e videójátékok kezdetének a legelső lépése. A játék egyik legnagyobb technológiai lépése az volt, hogy a játék DPD-re (Programmed Data Processor-1) készült, amely azokban az időkben a (leggizdább vúút 😂) legkorszerűbb technológiának felelt meg. Amelyen az utána érkező processzorok mind alapultak.

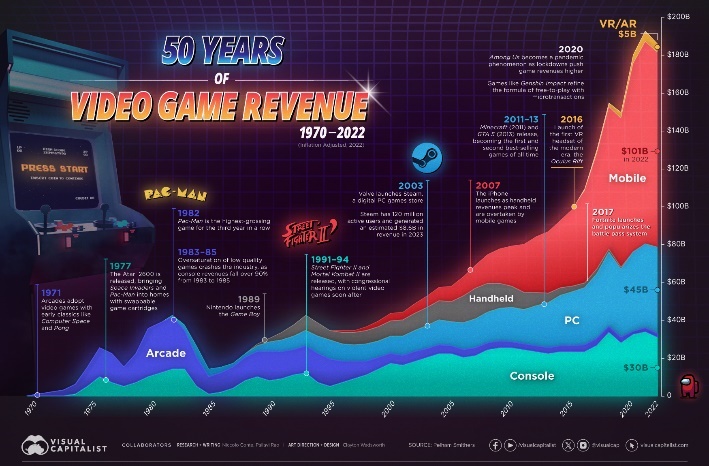
Az 1970-es években kezdőtek az általam ismert legelső digitális környezetben lévő és a nagy közönség által használható játékkal, ami nem volt más, mint a Pong. Ezen játék létrejötte elsőnek nem is egyszerű programozásnak köszönhető, hanem még az akkori idő villamos mérnökeinek hiszen maga a játék mindössze egy nyomtatott áramkörön futott, amit még a villamos mérnökök az áramköri jelekkel rajzoltak fel és valósították meg. Ezen játékok elsőnek nem voltak az otthonokban elérhetők, hanem csak az úgy nevezett játéktermekben, ahol hatalmas gépeken lehettet egy játékok játszani külön kiosztott irányítókkal, amik vagy sima gombokkal, illetve joystick-okkal voltak a legtöbb esetben felszerelve. A játék alap mente annyiból ált, hogy a képernyő jobb, illetve bal oldalán elhelyezkedő 2 fehér négyzet volt, amelyek között egy fehér kocka mozgott labda gyanánt. A játék célja az volt, hogy ezen labdát úgy üsse el az ellenfélnek, hogy az képtelen legyen azt visszaütni. A játékosok ez által szervez pontokat amig az egyik játékos el nem érte a 11 pontot. A játéknak 2 féle üzemmódja volt használható egyik a számitógép ellen, illetve a másik amikor 2 játékos játszott egyszerre.

A következő mérföldkőnek nevezhető játék, amit ki szeretnék emelni az nem más, mint az 1978-ban kiadott Space Invaders játéktermes játék volt, ami a maga idejében és a maga fajtájának megteremtője volt az akkori időkben. A játék lényege az volt, hogy képernyő tetejéről érkező űrlényeket egy kicsi a képernyő alján elhelyezkedő horizontálisan mozgó lézer ágyúval lelőjük és minél több pontot érjünk el ez által. Ez volt az első játék abban az időben, amelyben a nem ember irányította karakterek képesek voltak a játékosra lőni. A játék egyik érdekessége hogy nem csak műfajt teremtett hanem a modern videojátékokban való egyre nehezedő szintrendszert is bevezette az által hogy a játék programozása során nem számoltak a fejlesztők azzal hogy a játék előrehaladtával az ellenfelekből egyre kevesebb lesz és ezt követően a rendszer egyre gyorsabban tudja kirajzolni az ellenfeleket ami által egyre kevesebb idő telik el 2 egymást követő frame között ami által a játék maga gyorsabban halad ez által drasztikusan felgyorsul a játék 1-2 ellenfél maradta esetén.

A következő nagyobb lépés ebben az iparban a 3D-s játok lennének, amelyek közül is legfőképpen szeretném kiemelni az 1993-ban kiadott sikercímet a Doom-ot. Amely igaz nem a legelső volt a 3d-s műfajban viszont a műfajt kezdő Wolfenstein 3D-hez képest, ami viszont csak a felszint kapargatta 3D-s műfajnak addig a Doom-ban történtek meg az első olyan fejlesztési lépések, amelyek felismerhetőek a modern játékfejlesztésben. Ezen dolog alatt értem természetesen a játékban való elemek dinamikus kitöltését és ezáltal való gyorsabb feldolgozást, illetve még hasonló okos megoldásokat melyek mind azért voltak szükségesek és mai napig szükségesek, hogy az adott játékok megfelelő sebességgel tudjanak működni a játékosok játékélményéért. Mi sem bizonyíthatná az jobban, hogy ezen játék magas szinten van optimalizálva minthogy a mai napig készítenek ebből a játékból olyan változatokat melyek képesek akár a leg abszurdabb eszközön is futni csak hogy itt említsek párat okos termosztátok elektromos zongorák vagy akár még egy digitális lázmérőn is.

-ide még jöhet 1-2 játék valami már teljesen kifejlett 3D-s rendszerrel és valami modernkori valami rate tracinggel (pl god of war)

# -mostani telefonos és játékok piaci helyzete

Természetesen a játék fejlesztéssel és mindennel együtt kéz a kézben jár a gazdaság is és ezen kérdésekből való statisztikai elemzések. Aminek egy rövi elemzését szeretném bemutatni a következő részben. Első, amit fontos megemlíteni, hogy napjainkban mekkora piaci része van ezeknek a számitőgépes játékoknak. A jobb oldalon ezen értékeke leolvashatóak évről évre. Ami által láthatjuk, hogy ezen piac évről évre stabil növekedést produkál főleg az elmúlt 10 év során, amiben szinte megduplázódott maga a videójátékos piac értéke. Ezen növekedés a mobilos piacnak tudható be legfőképpen, mint az láthatjuk hiszen 2022-ben csak a telefonos játék piac kicsit több mit 101 milliárd amerikai dollárnak megfelelő piaci részt jelentett. Továbbá az ábrán az elmúlt évek az ezen a piacon megjelenő legnagyobb technológia vagy akár játékmeneti mérföldövek olvashatóak le. Ilyen például a 2003-ban megjelenő Steam amelyet Valve nevű cég adott ki azért hogy a játék megjelenését sikeresen letudják vezeti akár online környezetben így megszabadulva CD-s változatú játékoktól. Ugyan úgy észrevehető az ábrán egy 1982 környéki nagy ugrás, amin az abban az időben történő játéktermes játékoknak tudhatók be úgy, mint a Pac-Man, Space Invaders illetve Asteroids. Egy kicsit vissza térve a modernebb időkbe 2019 környékén észrevehetünk egy hatalmas ugrást az egész piacon, ami kétségkívül betudható az abban az évben történő Covid járványnak és illetve ezt követő korlátozásoknak, amelyek rengeteg embert kényszeritettek az otthon való kikapcsolódásra, aminek az egyik legjobb módja a videojátékok. Egyik fontosabb dolog, amit kiszeretnék emelni nem más, mint a 2016-ban megjelenő első nagy közönségnek szánt Virtual Reality röviden VR headset amelyek napjainkra több mint 5 milliárdos részt foglal le az egész piacról ami számom még mindig egész kevésnek tűnik a további társaihoz képest. Viszont egyben megérthető is hiszen maga a technológia nagyon friss és a használatához olyan ezközökre van szükség, ami a mai egyszerű háztartásokban még nem feltétlenül találhatóak meg. Viszont az elkövetkező időkben remélem tovább nő ez a piaci rész számolva a legujabb technológiai újításokkal mint például a Disney által kifejlesztett Holo Tile ami nem feltétlenül eheze a technológiához készült viszont a használata rengeteg új opcióhoz és megoldáshoz vezetne.

1. ábra Videójáték piaci értékelése

A diagramon ugyan úgy jól kiszúrhatóak a kézi konzolok (Handheld) amelyeknek a fénykor a fénykora mostanra eltűnt egyűt a hozzá tartozó eszközökkel mint a PSP vagy a Game Boy és jókora űrt hagyott, amelyet, mint láthatjuk a számitógépes játékpiac bekebelezett. Viszont ez a szűk réteg még mai és virágzik olyan eszközökkel, mint a Nintendo által készített Switch ami igaz nem illik bele ebbe a kategóriába tökéletesen hiszen maga az eszköz használható egyszerű console módban is. Viszont ez nem magyarázza azt, hogy miért nem számoltak a Steam Deck-el ami egy ugyan csak modern időkben készített kézi konzol amelyet a már említett Valve készített azzal az indokkal hogy játékokat lehessen élvezni utazás vagy egyéb teendőink közben ha esetleg nincsne a közelünkben áramforrás vagy esetleg nem tudunk magunkkal hordani egy asztali számitógépet. Lehet ezért nem jelenik meg ez az eszköz, mint kézi konzol az ábrán hiszen a rendszer tulajdonképpen egy hordozható játékeszköz, ami viszont Windwos operációs rendszer fut, ami így számítógépnek sorolható be. Természetesen ezen a piacon nem csak ez a két szereplő versenyzik rengetek hasonló eszköz lehet még találni különböző kínai gyártók által viszont azok a márkák nem annyira ismertek hogy említést tegyek róluk az esetleg megemlíthető vetéjtárs esetleg az Asus legújabb kézi számítógépe a Rog Ally lehet.

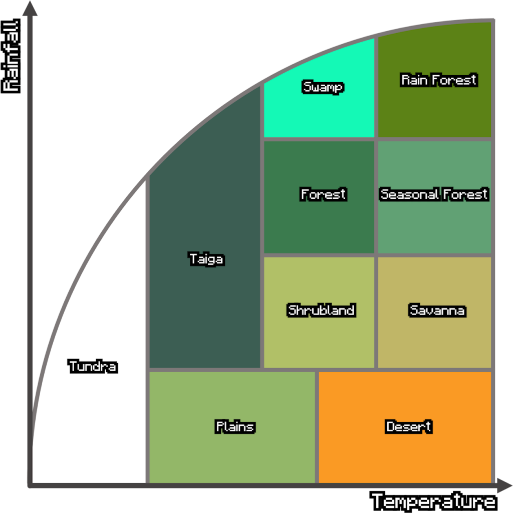
-lehetsléges irási alap: modern időkben meglőtt lootbox menyiség Live service games,XBOX Gamepass

# -Napjaink fontosabb technológiai

## Ray tracing

A modernkori játékfejlesztésben rengeteg mérnöki megoldást találhatunk annak érdekében, hogy a játékok fejlesztése egyre gyorsabban történhessen vagy ugyan olyan sebességgel tudjanak ezen játékok készülni. Mind ezt úgy, hogy ezzel egy időben a kész termék magasabb minőséget tudjanak képviselni. Ezek a technológiák közé tartozik a ray tracing technológia, amelynek az úttörése 2020 környékén indult meg. Az egész technológia alapja arra épül, hogy a játékokban megjelenő fényt visszatükröző elemek, mint a tükrök, illetve a pocsolyák olyan grafikával tudjanak megjelenni, ami játék közben valós időben változnak. Ezen technológia működése felettébb érdekes hiszen a való életben ahhoz, hogy mi úgy lássuk a vissza tükröződő elemeke ahogy az valójában történik. Ahhoz arra van szükség, hogy a napból érkező összes látható spektrumú fénysugár vissza patanyon az objektumokról egészen a tükröződő felületig, ahonnan ez sugarak a szemünkbe érkezzenek, amit természetesen egy a mai számítógépekkel kiszámítani nem lehetetlen viszont olyan mértékűen megnőne a szükséges számítási kapacitás, hogy a játékban egy kiszámított képkocka teljes kiszámítási ideje akár perceket is igénybe vehet. Ami természetesen nem a leg optimálisabb ahhoz képest, hogy a modernkori játékoknál egyik minimum követelmény az élvezhetőséghez az, hogy elérjék a 30 képkocka/másodpercet vagy angolul Frame Per Secundum-ot röviden (FPS). Nos ahhoz, hogy ezt a kívánt minimumot elérjük egy elég trükkös megoldáshoz kell fordulnunk méghozzá ahhoz, hogy valójában nem az összes objektumról vissza pattanó fénysugarakat szimuláljuk le, hanem ezen sugarak kezdőpontjának inkább, felfogjuk a játékban elhelyezett képzeletbeli kamerát. Az innen indítható minden irányba elindítunk egy képzeletbeli fénysugarat erre a tükröződő felszínre és minden olyan elem tükörképét jelenítjük meg ami ezen tükröződő felszínről a képzeletbeli fénysugaraink elérnek. Na most eddig ez elméletben egészen egyszerű és kivitelezhető is és mindenekelőtt nagy mértékben lecsökkenti számítási szükségletet. Viszont ez nem elég a kellő FPS szám eléréséhez. Ahhoz, hogy ezen technológia futásközben is megvalósítható legyen az már nem a játék fejlesztőkön mullt, hanem inkább a technológián, hogy elérje azt a szintet amire szükség volt. Ezen technológia mai napjainkban már elérhető és megtalálható a mindennapjainkba. A legtöbb videókártya amely RTX jelzéssel van ellátva már képesek ezen technikát alkalmazni.

## Procedurális generálás

A mostani idők játékait végig nézve nem tudnánk találni akár egy olyan műfajt, amiben nem találhatóak procedurálisan generált elemek. De mégis „Mi lehet ez a procedurális generállás?” lehetne feltenni a kérdés és a válasz egyszerre lehetne meglepően egyszerű mégis mélységleteiben bonyolult. A procedurális generálás létrejöttének egyik fő indoka az volt, hogy ahogy a gépek fejlődtek és egyre nagyobb és szebb játéktereket lehetett létrehozni a játékosok számára úgy ezzel együtt egyre nagyobb szükséggé vált, hogy a játékokhoz felhasználható minél nagyobb mennyiségű felhasználható grafikai elemre álljon rendelkezésre. Ezen elemek mindegyikének a megalkotása, illetve modellezése igencsak hosszadalmas és ezzel együtt költséges feladat bármelyik videójáték gyártó cégnek. Ezért kezdték el a fejlesztők kísérletezni azzal az ötlettel, hogy ezen grafikai elemek létrehozását meggyorsítsák számitógépek segítséggel. Kezdetben a procedurális generálás még az 1980-as években jelent meg. Az első rougeliek játékokban, ahol a különböző felszedhető tárgyak, kincsek vagy akár a szobák elhelyezkedésének generálására használták. Ezek a régi rendszerek még vagy ASCII kódos megoldásra alapoztak vagy valamilyen központi rácsrendszerhez illeszkedő megoldás jöhetett még szóba. Ezzel együtt természetesen léteztek olyan megoldások, amik pseudo random megoldásokkal dolgoztak mármint a szükséges elemek legenerálása egy központi kezdő szám- vagy betűsort képeztek, amelyek generálása természetesen véletlen szerű volt viszont a belőlük továbbá generált pálya elemek konzisztensen függvények alapján generálódtak. Ennek természetes következménye az, hogy a lehetségesen legenerált pályák száma véges. További opciók között említésre méltó az úgy mondott „noise map” módszer, ami technika lényeke annyi, hogy a generálásnál valójában egy 2D-s koordináta rendszert használnak, ahol az egyes koordináták értékéhez a számitógép hozzárendel egy 1 és -1 közötti számot majd ezen koordináta rendszer alapján egy vagy akár több szabály bevezetését követően a módosítások megtörténnek a játék környezetben. Ezzel is további véletlen szerűséget biztosítva. Erre a technológiára az elmúlt időben csak egy példát ismerek, mégpedig a Minecraft című játékban, amiben a nyíltvilágú és procedurálisan generált környezetet szerették volna felosztani különböző egységekre. Ezen problémára született a (képet hivatkozás ide) látható megoldás, ami 2 noise map kombinálásával készül méghozzá úgy, hogy a 2 noise map volt hivatalos beosztva arra, hogy a különböző tájegységek esősséget illetve hőmérsékletét reprezentálja és ezen kombinációból született mega feljebb látható kép. Az ábra görbülete pedig magyarázható azzal, hogy az ezen noise map-ek valójában csak 1 és 0 közötti értékeket reprezentálnak és az igy kapott értékeket egy koordináta rendszere felírjuk egy félkörívet kapunk. Napjainkban rengetegen próbálkoznak a mesterséges intelligencia bevonásával ebbe a területbe, amelyről (referncia ide) olvashatunk. Ezen összefoglaló megemlít a procedurális generálásnak több módját, ami a mesterséges intelligenciával felhasználva történne. Az egyik ilyen módszer nem lenne más, mint a „Search Based Procedural Content Generarion” (SBPCG) ezen procedurális generállás egyik főbb ismérve nem más mit az, hogy a generálást követően a végeredményeket felhasználjuk egy evolúciós algoritmusban, ahol egy megfelelő fitnesz érték meghatározása során a eldönthetjük hogy az adott végeredmények megfelelnek e a kritériumainknak vagy folytatódjon a keresés. Ezen technika bizonyítottan működő képes tud lenni 2D-s rács alapú rendszereknél a pálya generáció során. (valahova ide ékne egy referenicioa in links) Egy másik kutatása az evolúciós algoritmusnak ezen a területén az egy autóversenyes játékban történik, ahol magát a útvonalat a gép generálja majd ezen egy szimulált versenyző halad végig és a pálya kiértékelése 2 fitnesz függvény alapján történik az egyik függvény azt értékeli hogyha minél jobbnak ha a pályában minél nagyobb az útvonal diverzitása míg a szimulált versenyzőnek a fitnesz függvénye azt értékeli hogy a pályán való végig haladás során mekkora volt a sebesség ingadozás. Majd ezen értékeke összegzéséből születik meg a pálya vég pontszáma. Majd ezen folyamatot addig ismételik amig el nem érnek egy kívánt pontszámot. Ezen megoldás igen érdekes részemről mert a pálya élvezhetőségét nem csak egy szempontból vizsgálják, hanem többől is, ami esélyt ad arra, hogy pontosabban lehessen értékelni és ez által ajánlani mint kipróbálható opciók a játékosoknak. Ami még felkeltette ezen nyilatkozatban az érdeklődésemet az nem más, mint az előbb említett SBPCG használata a labirintus szerű szintek használatához. A hivatalos tanulmányban ezen módszerhez megemlítik, hogy a módszer használatához egyik fő szükséglet a labirintusok reprezentációja amiről 4 különböző módot említ amiből 2 direkt reprezentáció és 2 indirekt reprezentációja lenne ezen labirintusoknak. (Befejező mondat) A procedurális generálási technológiák még napjainkban is fejlődnek. A folyamatos szükséglet miatt ezen technolóiában a fejlődés szinte állandó. Köszönhetően annak rengeteg embernek, akik sok különböző szakterületről dolgoznak együtt azért, hogy egyre realisztikusabb világokat lehessen generálni.

2. ábra Minecrat noise map komináció

## Mesterséges Intelligencia használata a játékokban

Mivel előzőleg már bemutattam a mesterséges intelligencia használatát adott problémák megoldására szeretném további lehetséges területeket megemlíteni, amikben nagyobb előre lépésekre lehet számitani ezen technológiák felhasználásával. Az egyik ilyen lehetséges terülte nem lenne más, mint a játékokban való játékos interakciók lennének a világgal mármint ide érhetők a nem játék által irányított karakterek, amelyekhez gyakran szoktak szövegkönyveket írni a fejlesztők azon érdekből, hogy az általuk megalkotott játék úgynevezett „univerzuma” részletesebb legyen. Nos ezek akár több száz oldalas szövegkönyvek is lehetnek, amiket természetesen most már automatikus rendszerek segítik a feltöltésüket és adott karakterekre illeszteni ezeket. Viszont ezt akár különböző mesterséges intelligencia modellel is lehetne helyettesíteni akár amelyek megkapnák az eredeti beállitásokat valamilyen prompt rendszer alapján, ami lehetne: „te egy vadnyugaton vagy kocsmáros, akihez beérkezik egy vásárló” miután ilyen vezényszavakkal fel lettek készítve ezen modellek utána a többiről már gondoskodhatna a mesterséges intelligencia úgy mint a válasz generállását és egyben ezen válasz felolvasásáról is gondoskodhatna adott előre felvett minták alapján. De továbbá megvalósíthatónak tartom akár azt is, hogy maga a játékos álljon szóba ezen nem játékos karakterekkel és ezzel is növelve a játékok immerzióját és élvezhetőségi faktorát. Egy másik fontos és elég könnyen megvalósítható ötletem lenne a mesterséges ellenfelek irányitására. Mégpedig egy olyan, hogy az ellenfeleket a tesztelés során fejlesztett AI modellel lehetne irányitani, aminek a feladata nem inkább az lenne, hogy magát az ellenfelet megpróbálja legyőzni, hanem inkább egy minél hosszabb és élvezetes meccseket próbáljon létrehozni. Mondjuk valahogy úgy történne ez rendszer, hogy a játékosról kezdetben semmilyen adatot nem használna és ahogy ezen játékos a játékban egyre nagyobb és jobb szinten tudna játszana a mesterséges intelligencia ugyan ilyen ütemmel fejlődne és állítaná magát az ellenfelél tapasztalati szintjéhez folyamatosan egészen a játék végéig. Amit még lehetséges opciónak tartok ezen a téren az nem más, mint a mesterséges intelligencia használata a játékok és egyáltalán a programok tesztelésére is. Hiszen ezen tesztelési feladatok meglehetősen könnyen helyettesíthetők legalább is az általánosabb dolgok. Viszont igencsak nagy mennyiségű munkát igényelnek. Ha csak arra gondolunk, hogy hogyan lenne ezen célra fejlesztette mesterséges intelligenciához szükséges mennyiségű adatunk akkor csak elég belegondolni, hogy egy játék önmagában akár több száz úgynevezett bugot tartalmazhat, amit fel lehetne használni ehhez a projekthez. Ha ezen projektekhez sikeresen tudnánk fejleszteni egy GPT-hez hasonló AI modellt, amit a játék programozása közben képes lenne szimulálni az általunk irt kód működő módját és illetve annak a sarkalatos pontjait megtalálni, ami alapján lehetőségünk ezen bugok létrejöttének a megakadályozását. Vagy valami hasonló módon lehetőségünk lehetne egy olyan modellt kifejleszteni, ami a számitógépeken való optimalizálásában lenne képes segíteni. Hiszen napjainkban is az egyik legnagyobb probléma maguknak a fejlesztőknek, hogy a játékok, amiket kiadnak azok valójában nem egy fajta gépen futnak, mint a konzolok esetében, hanem több különböző processzor és videókártya kombináción és még továbbá eltérő memória típusokon és mennyiségeken.

## Cloude Gaming napjainkban

Egy másik érdekes technológia, amit szeretnék megemlíteni és elég szorosan kötődik a mai időkben megfigyelhető szoftver fejlesztői trendeknek, ami nem lenne más, mint a felhősités. Ezen kategóriában szeretném sorolni azon technológiát, amit szeretnék bemutatni, ami nem lenne más, mint a Cloude Gameing. Ezen technikai ötlet még úgymond elég újkori ötlet feltéve, hogy az egész a 2000 évek elejeén született meg egy G-cluster nevű projekttel. Amit 2000-ben mutattak be az az évi E3-on. Majd ezt követően 2003-ban meg is jelent a nagy közönség számára. Ezt követően az ami nagyobb lépésnek számítót azt az volt hogy 2010-ben komoly előre lepéseket tettek az adattömörítés terén amik kihatással voltak a grafikai tömörítésekre is és ezzel párhuzamosan a mobiltelefonos piacon is fejlődésen mentek végbe ami csak tovább növelte a felhő alapú játszásban rejlő potenciált. Ezek után megjelentek olyan nagyobb szolgáltatások, mint az OnLive vagy a Gaikai. Amikből a Gaikai-t a Sony Computer Entertainment felvásárolta 2012-ben. Majd a OnLive-ot pedig miután nem tudott profitot termelni 2014-ig így a technológiai fejlesztésekhez tartozó jogokat felvásárolta ismételten a Sony Computer Entertainment aki pedig 2015-ben megszüntette az OnLive-ot igy a Sony-hoz került a felhő alapú fejlesztések légy nagyobb része. Ezek voltak a nagyobb események cloude gaming kezdetén. Ha az időben előreugrunk egy kicsit akor a következő nagyobb eredmény nem mási, mint a Google által 2019-ben bejelentett Stadia nevű szolgáltatás, amit még abban az évben el is indított. Majd ezt követően az Amazon is bemutatta a saját hasonló technikára építő megoldását a Luna nevű coulde gaming szolgáltatást. Majd ezt követően az Nvidia jelentette be 2021-ben a saját platformját a Geforce Now-t azért, hogy össze tudja fogni a többi nagy platformokat egy helyen a felhasználók érdekében. Ezen törekvések nagyszerűen haladtak viszont idővel kibuktak a hibák, amiket sajnos nem lehettet könnyen vagy legalábbis nem horribilis összegekből megoldani úgy, mint a játékokat élvezhetetlenné tévő késés, ami egyesesetekben akár 1- 2 másodperc is lehetett. Igy a Google véget vettet ennek a projektnek még 2023 január 18. -án. Mai napjainkban rengeteg hasonló project áll rendelkezésre a már előbb említetek felett, mint a Valve Steam Cloude Play név alatt futó szolgáltatása. Viszont, ha már e technológia történelmén igy végig futottam akkor hadd említsem meg ezen technológia lehetséges és vagy ha már létező előnyeit is. Egyik fontos szempont ezen szolgáltatásoknál, hogy úgy kínálnak magas minőségű játékélményt hogy valójában nem szükséges hozzá olyan számitógépet építenünk vagy vennünk amik nagyon magas összegekbe kerülnek hanem egyszerűen befőzetünk egy 1 havi összeget ami általhozzáférünk egy felhőben lévő számitógéphez ami képes futtatni a játékot. Nincs is magas feltételekhez köve ezen lehetőség csupán csak annyihoz, hogy legyen egy elég nagy sávszélességű internet elérésünk egy számitógép, ami képes ezen keresztül kommunikálni a szerverrel és az alapvető irányitásokhoz szükséges eszközök ide értendő az egér és billentyűzet vagy akár egy kontroller is. Mivel a modern háztartásokban a nagy sávszélességű internet már eléggé elterjedt és általánosan egy háztartásra jut egy számitógép, ami megfelel ezen elvárásoknak így ezen lehetőség lehet a legoptimálisabb a legtöbb háztartásban. Viszont milyen további lehetőségek rejlenek ebben e módszerben? Először is mivel szerverekről beszélünk, amiknek egyik nagy előnye a párhuzamositás így lehetővé válna valószínűleg, hogy az adott játék enginek több gépet is használhassanak egyszerre ez által még több részletesebb játékszimulációt lehetne létrehozni. Vagy esetleg egy másik előny, amiben a felhő alapú rendszer segíthet az nem más, mint voxel-ek általi számitógépes grafika használata, ami a modern környezetben egyedüli hátulütője az, hogy nagyon memória igényes futás közben viszont a szerverek rendelkeznek az ehhez szükséges memória a mérettel és ezen felül is viszont ezen elemekből 1 eredeti formát kéne megtartani ahhoz hogy a további kisebb grafikájú elemeket kilehessen számitani ami igazán csak hosszútávon bizonyulna megérőnek.

## -VR

# Napjainkban használatos fejlesztői környezetek

Unity  
A játékfejlesztés kezdetekben még nem igényelt mást csak jártasságot a programozási nyelveben és azt, hogy képes legyél ezt felhasználva grafikusan megjeleníteni az adott játékot. Viszont az idő előrehaladtával és az ipar fejlődésével együtt ezen módszerek nem lehettek hosszútávon fenntartható. Ezért születtek meg a napjainkban is használt játék fejlesztési környezetek és a különböző játékmotorok, amik használatával a játékfejlesztés nagy mértékben megkönnyebbült és gyorsabb is lett ezáltal. Természetesen ezen egységek létrejöttével megszületett általánosítások nem kínáltak személyre szabható megoldásokat, ami néhány esetben nem a tökéletes megoldást nyújtja, ami estelegesen futás közbeni vagy akár optimalizálási hibákat eredményezhet. Ezen problémák fennállása mellett is jobb választásnak bizonyulnak ezen általánositott megoldásokkal szolgáló rendszerek egyedül a fejlesztési idő lecsökkentése miatt. Amik napjainkban talán a legismertebb ilyen környezetek nem lennének mások, mint a Unity vagy az UnrealEngine5. Amelyek népszerűségüket 2 teljesen eltérő indokból szerezték. A Unity ugyan is egy főleg egyszerűbb fejlesztői környezet, ami által a szakmába belekezdő embereknek nagy könnyítést jelent nem is beszélve arról, hogy ezen játékmotor rengeteg különböző platformot támogat, ami a játékok terjesztését is megkönnyíti. Nem is beszélve arról, hogy a különféle platform opciók által több olyan ember választja, aki egy specifikus platformon működő játékot szeretnének viszont ezzel együtt szeretnék elsajátítani a szakmai alapokat is. Egyúttal az sem elhanyagolható tény, hogy a játékmotor képes mind háromdimenziós, illetve kettődimenziós projekteket kezelni vagy akár szimulációkat is, szinte nevezhetnénk a játékmotorok svájcibicskájának. Hiszen a játékfejlesztésben szinte minden főbb ágat le tud fedni egymagában viszont nem olyan jól, mint egy specifikus rendszer. Amit még fontos megemlíteni, hogy a unity C#-ot használ a belsős programok futtatásához, amit megint csak az egyik legkőnyibben megérthető és tanulható programozási nyelvek között tartanak számon. Mindezen indokok miatt olyan kiemelkedő napjainkban ezen fejlesztői platform.

Unreal Engine 5  
Viszont ezen platformok listája itt nem áll meg. Hiszen nem olvashatunk úgy erről a témáról, hogy nem említenénk meg a UnrealEngine 5-öt. Ugyanis napjainkban ezen játékmotor, ami a szakma lejobbjai között emlegetnek a magával hordozott korlátokkal együtt is. Ugyan is azon platformok, amiken ezen a motor képes futni egészen szükősnek mondható az előbb említett Unity mellet viszont ez meg is érthető, ha megtudjuk, hogy ez nevezhető napjaink csúcs technikájának, ami indokolja a csúcskategóriás eszközöket. Ezen állitást pedig mi nem igazolhatná jobban mint a készítők által bemutatott 5 nagyszabásu rendszerük amik nem mások mint a Lumen,Niagara,Chaos,Nanite és a Quixel amik közül a Quixel lehetne egy kakukk tojás hiszen ezen nem egy beépített rendszer hanem egy külső adatbázis aminek annyi célja hogy a legnagyobb fotogrammetriai könyvtára legyen a világnak és ez által minél könnyeben tudjanak a fejlesztők nagy felbontásúelemeket felhasználni. A többi négy már beépített rendszernek felel meg amik közül a 2 legizgalmasabb az nem lenne más, mint a Lumen, illetve a Nanite ami közvetlen a játékok grafikájára van kihatással. Ugyanis a Lumen úgy van jellemezve, mint egy globális és dinamikus megvilágitási rendszer, ami képes azonnali változásokat lekövetni és ezen változások szerint változtatni az árnyékokon és fényforrásokon ezáltal megszüntetve a szükségességét a fejlesztők által készített és használt lightmapping-nek. A másik rendszer, ami nem lenne más, mint a Nanite pedig egy új renderelési megoldás. Hiszen at előző megoldás tartalmazott egy nagy hibát ugyan is ezen módszer úgy működik, hogy a játékban mikor háromdimenziós objektumokat rajzolgatunk ki a képernyőre akkor ezen elemeknek van egy „mesh-je” ami alapján betöltődnek az objektumot jellemző poligonok viszont ezen poligonok mennyisége állandó mesh-en ként vagyis ezen nagy részletességű elemeket minden esetben ugyan olyan menyiségű számítási igénnyel rendelkeznek. Ami ingen csak nagy probléma feltételezve, hogy ezen polygonok láthatósága digitális kijelzőböl kifolyólag csak egy adott méretig láthatóak, amik nem lennének mások, mint a pixelek méretei. Igy viszont belátható, hogy ez egy nem túl optimális megoldás ezért a fejlesztők egy olyan trükkhöz folyamodtak, hogy ezekből a mesh-ekből valójában csinálnak többet, amik sorra kevesebb poligont tartalmaznak majd ezen mesh-ek közül kezdenek el a játék futása közben cserélgetni attól függően, hogy milyen szükséglet áll fent. Szemben az új rendszerben, ami úgy működik, hogy a játékmotor azon poligonokat rajzolja ki, amik nem kisebbek, mint a kijelzőn látható pixelek ezzel megspórolva rengeteg számítási kapacitást és memória helyet is hiszen betöltve a memóriában csak egy modell van nem pedig mindegyik vagy legalább is egy része azoknak a alacsonyabb felbontású minták amit az előbb említettem. Ami még egy fontos aspektusa a Nanite működésének az nem más minthogy ezen beépített rendszer támogatja a legtöbb háromdimenziós modellt kezelő fájlformátumot. Ideértve a CAD programok által használt modelleket is ezzel is segítve a fejlesztőket.  
Azon fejlesztői környezetek, amiket még érdemes megemlíteni az nem más lenne, mint az AppGameKit és az Amazon Lumberyard lenne, amik nem azon részeit célozzák meg ennek az iparnak, mint az előbb említett 2 fejlesztési környezet, hanem 2 megint csak teljesen eltérő irányzattal rendelkeznek. Mint például a Lamberyard főleg arra összpontosít, hogy a játékot készítő embereknek egyszerűbb legyen a már más amazon platformon futtatni vagy használni a játékokat ideértve természetesen a CloudeGaminget vagy ami még napjainkban egész nagy sikere van az a streaming platformokon való játék ami a streamer és a nézők bevonásával történő játék lenne. De ezen felül tartalmaz C++ támogatást is, ami futásidőben ez egyik legjobb opció viszont igen magas tanulási íve van igy ezen eszköz használata csak is az igazán tapasztaltaknak ajánlott. Vagy akár az AppGameKit-et is említhetem, ami egyik főbb szempontja az volt hogy a fejlesztésre elegendő legyen egy applikációs ablakot nyitva tartani úgy hogy ne legyen szükség egy extra képernyőre aminek köszönhetően akár a telefonunkon utazás közben is képesek vagyunk fejleszteni. Mind erre képes ez a fejlesztési platform azok mellet hogy az itt elkészített projektek futtathatóak szinte bármilyen platformon ami annak köszönhető hogy a fejlesztői felület egy egyedi saját fejlesztésű kódolási nyelvet kapott.

Roblox  
Lezárás képpen szeretném megemlíteni azon fejlesztési környezettet amit nem sorolnák a szokásos értelemben vett fejlesztési környezetek közé hiszen ezen elemek valójában már egy kész termékeke amik mondhatni már játékok viszont fel vannak szerelve olyan elemekkel amik segítségével lehetősége nyílik a játékosoknak az önmaguk által meghozott szabályok alapján játszani. Az ilyen platformokon valójában egy olyan eszköz lesz elkészítve, ami egy tojáshéjnak felelne meg és azon tartalmáról a játékosok döntenének. Vagyis ezeken a platformokona a játék grafikai elemeit nem lehet változtatni viszont az ezen elemek által létrehozott játéklogikán igen. Ami azt eredményezi hogy a felhasználók rengeteg játékot tudnak létrehozni és megosztani egymás között ami által nagy játékos közösségekre tehetnek szert a fejlesztők. Amik ilyen fajta platformok közül a legnépszerűbb az nem más mint a Roblox című platform.

## Játék fejlesztés Android és Google háttérrel

Az egyik főbb szempont a fejlesztés megkezdése előtt nem más, mint hogy az adott környezetben, amiben fejleszteni szeretnénk megvizsgáljuk a lehetőségeinket. Ezért én is így tettem a fejlesztésem kezdetén, amikor is elhatároztam, hogy Androidos platformon szeretnék fejleszteni. Az első ötletem tehát nem volt más, mint hogy végig nézek a már adott platformhoz tartozó lehetőségeken, ami között meg is találtam a Google által fejlesztett könyvárat, ami az Android Game Development Kit nevet (AGDK) viseli. Ezen rendszert egyszerűen hozzá lehet csatolni egy Visual Studio-s környezethez, ami után már egy létező projektünket fojttathatjuk az AGDK által nyújtott lehetőségek által. Ami rengeteg lehetőséget ajánl a fejlesztés különböző szakaszaiban. Mint például C és C++ programozási nyelv támogatása, ami alapján lehetőség nyílik az érdeklődőknek, hogy játékmotorokat fejlesszenek vagy legalább is kiegészítsék a már létező saját projektjüket. De ezek felet ez a könyvár ajánl sok más egyéb opciót, mint például a játék framerate-jének a beállításához, ami által állandósíthatjuk a játék által igényelt számítási szükségletet és ez által a játékhoz való bemeneti jelek késedelmét is lecsökkenthetjük. Ezzel is növelve a játékélményt a játékosokban. Vagy amit miatt ezen eszköz hasznosnak nevezhetjük az nem más lenne, mint a futásidő közbeni memória, illetve CPU profilozás aminek köszönhetően elég egyszerűen meglehet találni olyan hibákat amiket programozás közben nem lehet elkapni egy debuggerrel hiszen a hiba csak is futásközben történne meg mint például egy egyszerű textura betöltés a program futása közben ami a betöltés után nem lesz összeszedve a garbige collectro által viszont íj módon a memória telítődik ami nagy mértékű lassuláshoz illetve memory leak-hez vezethet. Ami még részemről egy hatalmas segitségnek tartanék ebben a könyvtárnak az nem lenne más mint a GameTextInput hiszen a saját projektem fejlesztése során rengetegszer jutottam arra hogy valamilyen szöveges formátumot kéne kiírni a képernyőre a játékosoknak viszont ezen probléma megoldására az én esetemben egy megoldás lett volna ami nem más mint hogy készítek egy spritesheetet a programban használni kívánt karakterekhez amiket ezt követően egyesével hozzá kellet volna rendelni a String karakterekhez majd azt követően mindegyik ilyen karakternek kellet volna készíteni egy külön objektumot amiket ezek után lehetett volna használni mint egy egybe tartozó mondat. Vagy egy másik megoldás lehetne az is hogy ugyan úgy létrehizozm a spritesheetet a karaktereknek viszont a kiírásuk nem objektumonként történik hanem a futás idő közben generálódik egy textura ezekből amit csak egy objektumra lehetett volna rakni így megkönnyítve a dolgot viszont igy az objektum kirajzoláskor a méreteit változtatni kellene minden egyes ilyen textura változásnál hiszen ezen objektumok ha csak 5 karakternek megfelelő hosszall rendelkeznek akkor egy hosszabb vagy rövideb szó vagy mondat torzítva szerepelt volna. Viszont a már említett GameTextInput osztály erre megoldást szolgáltat ezzel is le egyszerűsítve a feladatott a felhasználóknak és ezzel is rengetegidőt megspórolva.

## -HTML5-ben játékfejlesztés-> IOS->OpenGLES

# Meghatározott terveim a sasját játékkal szemben

Megtvalóstiás főbb lépesei részletes leirása kód részletek stb(ez less a fő lényeg kb 15-20 oldal kéne)

## Elindulás, kezdetek bemutatása

(openGELS és a komplex tervezés kezdeti alapok )

## -A projekt jellenlegi állapota(kódbemutatása)

## További fejlesztési lehetőségek (1 oldal)

Nos mivel ezen program nem felel meg egy kész állapotban lévő programnak személyem szerint bár ennyi idő alatt nem is tudtam teljesen kiismerni minden egyes fortélyt, illetve technikát ezen szakterületen. Ugyanúgy ahogy természetesen nem futottam bele minden hibába problémába vagy esetleges tervezési akadályba még úgy is, hogy elég hosszú ideje dolgozok ezen a projekten. De ez csak azt jelenti, hogy tovább szeretnék foglalkozni ezzel az applikációval, amelyhez jó pár ötletem van még a továbbiakra. Ilyen ötletem lenne az, hogy a játékhoz több ellenfelet szeretnék készíteni az EneymCharacter osztály felhasználásával. Ezek az ellenfelek szeretném, ha mozgás stílusukban eltérőek lennének például lehetnének olyan ellenfelek, amelyek mozgásában nem szólnának bele a környező falak viszont ilyen fajta könnyítéssel valamivel könnyeb lenne őket megsebezni vagy esetleg nem közvetlen támadási mintája lenne. Továbbá szeretném, ha nem csak az egyedüli interakciók csak a játékos és e gép által irányított karakterek között lennének meg hanem szeretném, hogy adott ellenfelek különbözően reagáljanak adott ellenfél típusokra, mint például megtámadják egymást vagy egymással együtt harcolva plusz effektusokat tudjanak előidézni ez által is bonyolítva a játékot és egyszerre élvezhetőbbé téve azt. Ezek által maga a játékosok egyre többször buknának el ahogy előre haladnak és bukásonként egyre nagyobb szinteket vagy titkokat oldhatnának fel, amik később egyszerűbbé tennék ezen szintek elérését vagy a rajtuk való szórakozást. Ugyan ilyen módon szeretném hozzá adni a játékhoz valamilyen csapattársi rendszert viszont arra még nem találtam ki jó megoldást, hogy hogy lehetne kreatívan ezt beépítene, hogy ne csak egyszerűen az ellenfelek életének megduplázásával történjenek a nehezítés. Viszont egy lehetséges opciónak tartom ezen probléma megoldására, hogy maga a labirintus mérete nőne meg és nem ugyan azon a helyen küzdenének maguk a játékosok, hanem 2 különböző ponton, ahonnan megkéne találniuk egymást és közösen kijutni a labirintusból. Viszont ezen ötlet több problémát is felvet, mint például, ha 2 játékos közösen harcol akkor az aktív ellenfelek valójában kit célozzanak, mint közvetett ellenfelet kettejük közül. Amit még tervezek extra tartalomként beépítene a játékomba az az, hogy maga a játékos karakter nem csak egyféle képen tudjon harcolni az ellenfelekkel, hanem különböző fegyverekkel, amik között a váltás gyorsan történhetne és ez által is egy magasabb szintre lehetne elérni. Nem is beszélve arról a játékon belül a játékosok által elért gyakorlottság még egy extra szórakoztató lehetőség lenne. Ami még megfordult a fejembe ezen játék fejlesztése során, hogy maga a játékban lehetne létrehozni különböző csapdákat, amelyek a játékos által lehelyezhető eszközök lennének és esteleg adott ellenfél típusok ellen használhatóak lennének ezzel is adva a játékosoknak egy olyan lehetőséget, ami által a tapasztaltiakból tanulnak és ezek alapján nagyobb gyakorlottságra tennének szert és ezt követően nagyobb sikerélményük lehet a játék folyamán. Amely módszer további problémák vetne fel mondjuk milyen fajtájukra legyenek a csapdák már, mint lehetséges opcióknak sorolom a csapdákat melyek főbb célja csak annyi lenne, hogy az ellenfeleket megsebezze, de lehetséges lenne olyan csapdát is csinálni, ami elfogná valamilyen ellenfél típust és míg egy másik típust pedig sebezne. Aztán ezen elfogott ellenfeleket meg mondjuk fel lehetne használni saját eszközök fejlesztésére vagy akár muníciónak adott fegyverekbe.

//Valahova rakj egy képet jógyerek úgy jó lesz

-Bluetooth kapcsolattal való közös játék

-Ellenfelek reakciója további ellenfelekre

-több féle harci stilus amit váltható a játék folyamán

Angol Magyar összefoglaló  
(1-1,5 oldaél) a project és az én viszonyom

# Referenciák (10 darab minimum)