

RUANDER Oktatási Kft.

# **SZAKDOLGOZAT**

Sárosi Fanni  
Szoftverfejlesztő

Budapest  
2019

RUANDER Oktatási Kft.

# CubiX játék

**Konzulens:**

Kis Balázs

**Készítette:**

Sárosi Fanni

Szoftverfejlesztő

Budapest, 2019

# Tartalomjegyzék

<b>Bevezető</b> .....	2
<b>Felhasználói tervdokumentáció</b> .....	3
Legalacsonyabb rendszerkövetelmények: .....	3
Részletes funkcionális leírás .....	3
Telepítési és használati útmutató.....	4
Hibák .....	9
<b>Fejlesztői dokumentáció</b> .....	10
Követelményspecifikáció .....	10
Unity.....	10
Visual Studio .....	11
Kezdeti dokumentáció:.....	12
<b>Diagramok</b> .....	13
Package Diagram.....	13
UseCase Diagram .....	15
Class Diagram .....	17
Sequence Diagram.....	20
State Machine Diagram .....	23
Deployment Diagram .....	26
Component Diagram .....	28
<b>Tesztelés</b> .....	31
<b>Jövőbeni tervek</b> .....	33
<b>Összegzés</b> .....	34
<b>Irodalomjegyzék</b> .....	35

## Bevezető

Amióta tudomásomra jutott, hogy szakdolgozat keretén belül egy saját programot kell írni az OKJ-s tanfolyam elvégzéséhez, azóta tudtam, hogy játékot szeretnék írni. Nagy szenvedélyem maguk a számítógépes játékok és meg akartam ragadni a lehetőséget, hogy a fejlesztésén keresztül jobban megértsem az azokban zajló folyamatokat, rendszereket, felismerjem mennyi munka is egy működő és elsősorban élvezhető program írása. Egy kicsit bonyolult volt legelőször megismerni a Unity-t és akár egy kis játék részletet is kreálni benne, de úgy érzem, ha haladó nem is, de egy középhaladó szintet elsajátítottam benne. Mindenképp egy kisebb árkád stílusú játékkal akartam kezdeni, melyet könnyű játszani és a nem éppen tapasztalt játékosok is szívesen, könnyedén tudják használni, kezelni. Bár eredetileg egy kicsit másfajta játékot terveztem, a végére összeállt a kép és megszületett a CubiX. A CubiX egy árkád játék, melyben egy kockával kell akadályokat kikerülnünk, és bár egyszerűen hangzik, ez nem mindig az. Az egyes szintek teljesítése után változik a játék struktúrája, a mozgó akadályok és az ugrás újabb játéktechnikákat csempésznek be, ezzel néhány szinten borsot törve az orrunk alatt. Ha pedig egy kicsit másra vágunk, egy úgynevezett végtelen szinten üthetjük el az időt, egyre gyorsabban létrejövő akadályokkal. A játékot egy kicsit magunkévá is tehetjük a „Costumise” menü segítségével, mellyel változtathatjuk az általunk vezérelt kocka színét, ezzel egyénire szabva azt. Egyelőre ennyit a játékról nagy vonalakban, most kezdünk neki beleásni kicsit.

## Felhasználói tervdokumentáció

### Legalacsonyabb rendszerkövetelmények:

- AMD E-450 APU
- 4 GB RAM
- AMD RADEON HD 6320 Graphics
- Windows 7
- 100 MB szabad lemezterület

### Részletes funkcionális leírás

Az egyetlen dolog, amit a felhasználónak meg kell adnia az a használni kívánt felhasználónév a játék legelején, a „Submit” gomb megnyomása után már csak a felhasználói felületet kell értelemszerűen használni. A gombokon egyértelműen rajta van, mit takarnak. A „New Game” hatására elkezdődik a játék az 1. szinttől. Ekkor a felhasználónak a billentyűzetten található „W” „A” „S” „D” betűk, vagy a nyilak használatával kell elnavigálnia az akadályok között, ha pedig ki szeretne lépni, azt az „Esc” billentyű segítségével tudja megtenni, egyéb gombnyomásra nincsen szükség, az adott szint automatikusan újraindul „GAME OVER!” esetén és a szint teljesítése után az új szint is automatikusan betöltődik.

Egyéb játékmódok pár kattintásra elérhetőek (alább részletezve).

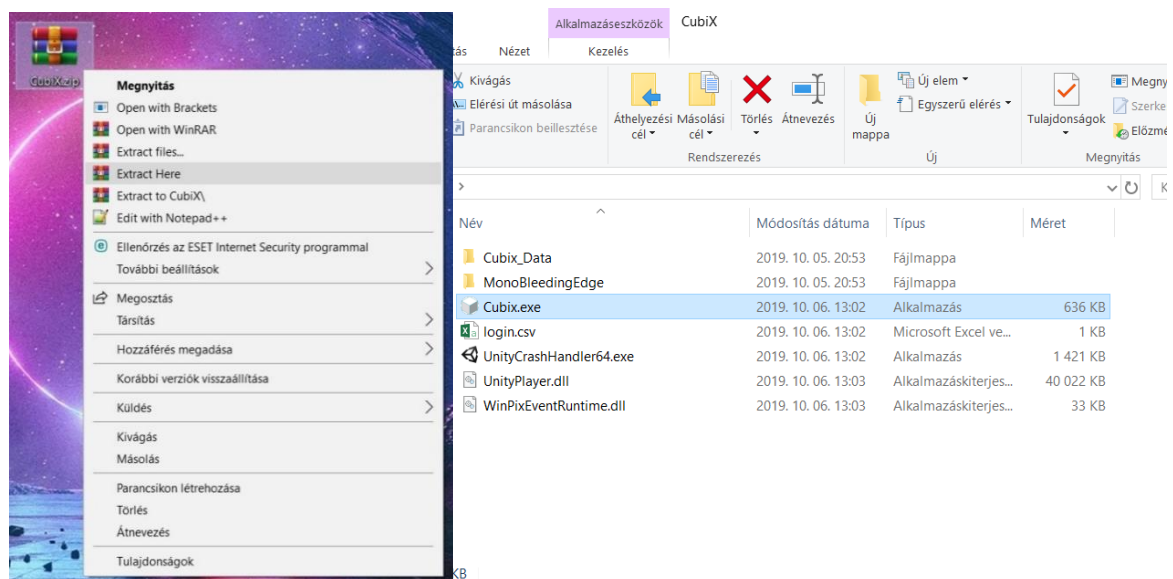
A felhasználónak lehetősége nyílik személyre szabásra is a „Costumize” menün keresztül, melyre kattintva egy ColorPicker ugrik a szemünk elé. A ColorPicker használata sem igényel mély informatikai ismereteket, egy színskálán állítható a kívánt szín, majd megadható annak árnyalata is. A haladó felhasználók könnyebben is beállíthatják a kocka színét, a színskála és árnyalat alatt megtalálhatóak az RGBA értékek is, így, ha például teljesen zöld kockát szeretnénk elég csak beírunk a 0, 255, 0 értékeket és már meg is van. A változtatások a „Back” gomb hatására lépnek érvénybe.

Egyéb dolgok, amiket a felhasználó állíthat a „Settings” menü alatt találhatjuk. A zene hangerejét a felső csúszka állításával lehet változtatni. A teljesképernyős módot a felhasználó kikapcsolhatja a pipa kivételével a „Full Screen” felirat mellett. Alatta pedig a felbontás változtatható 320x200 és 1920x1080 közötti értékre. Végül pedig a grafikai beállítás állítható: alacsony, közepes, magas fokozatba.

A legutolsó „Quit” gombbal a felhasználó bezárhatja a programot, újraindítás esetén, ha ugyanazt a felhasználónevet írja be, a mentett változtatásokkal indul a program.

### Telepítési és használati útmutató

Másolja át a CubiX nevű mappát számítógépére, majd nyissa meg és indítsa el a „Cubix.exe” fájl rendszergazdaként (jobb klikk a fájlra és „Futassa rendszergazdaként” opció).



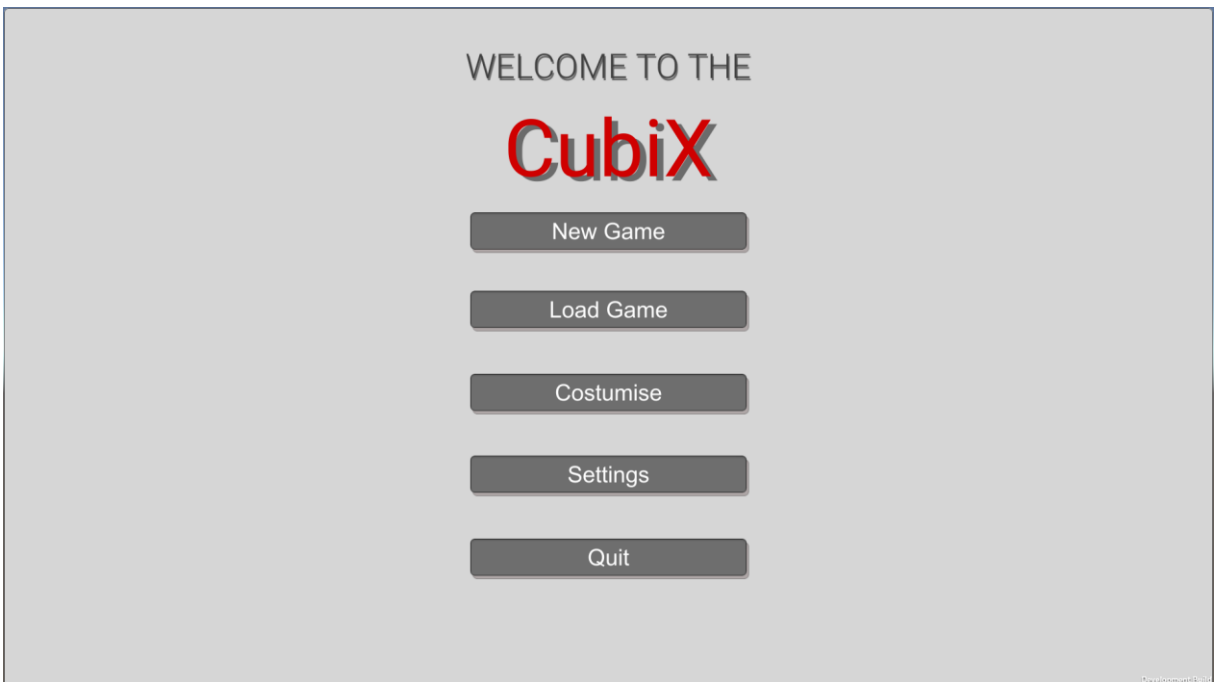
Ezután a program magától települ és elindul. A kép, amit látnia kell a következő:



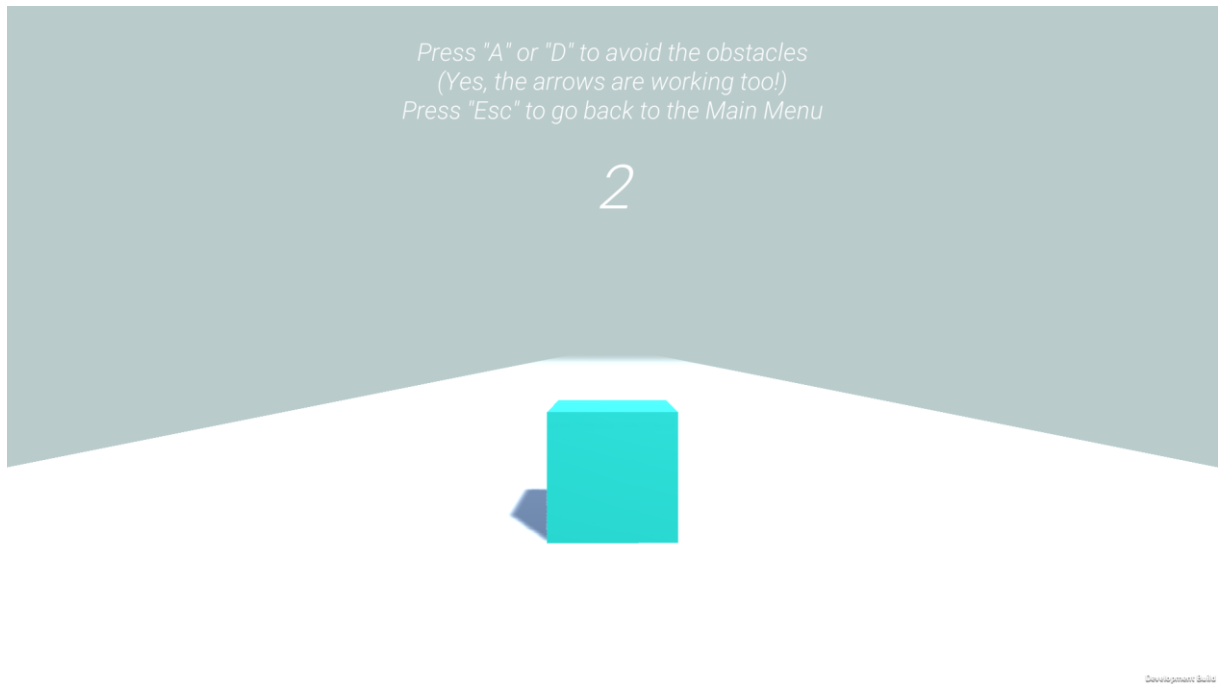
Majd megjelenik a bejelentkezés képernyő, írja be a felhasználónevet, amit használni szeretne és kattintson a „Submit” gombra.

The image shows a login screen for a game called CubiX. At the top, the word "CubiX" is displayed in a large, red, stylized font. Below it, the text "User Name:" is shown in a dark grey font. Underneath the text is a white text input field with the placeholder text "Enter text...". Below the input field is a dark grey button with the word "Submit" in white text. The entire screen has a light grey background. In the bottom right corner, there is a small, faint text that reads "Developed by: [illegible]".

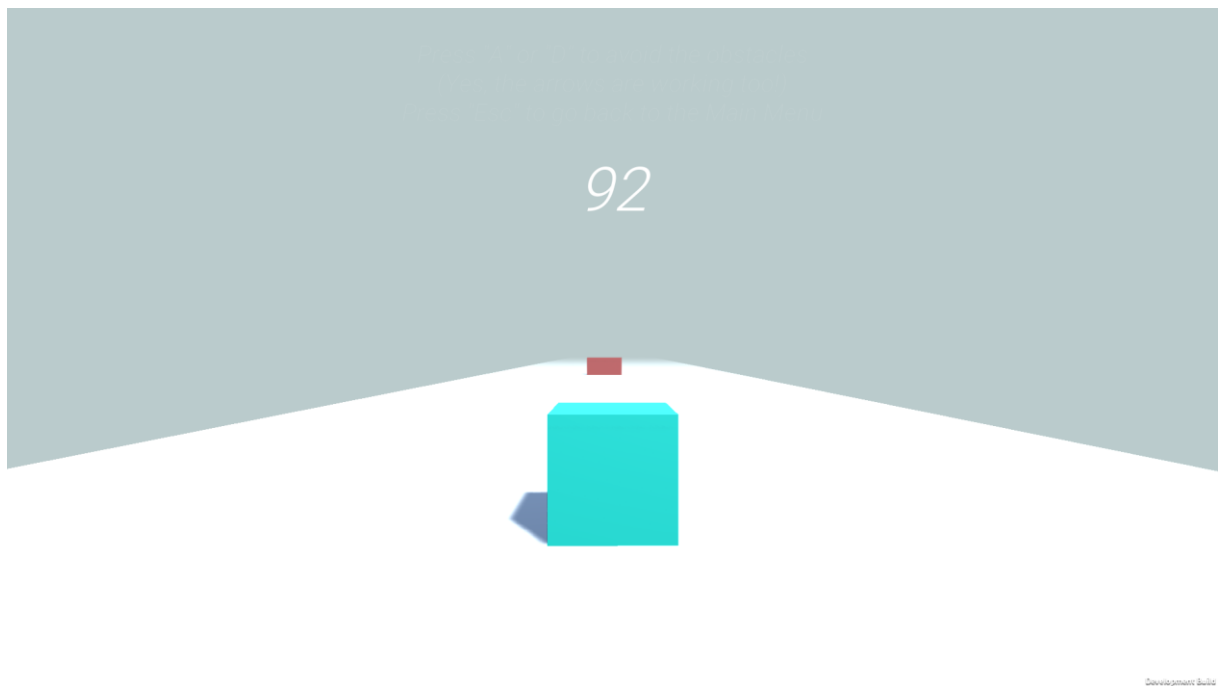
Bejelentkezés után eldöntheti mit szeretne csinálni.

The image shows the main menu of the game CubiX. At the top, the text "WELCOME TO THE" is displayed in a dark grey font. Below it, the word "CubiX" is shown in a large, red, stylized font. Underneath the title, there are five dark grey buttons with white text, arranged vertically. The buttons are labeled "New Game", "Load Game", "Costumise", "Settings", and "Quit". The entire screen has a light grey background. In the bottom right corner, there is a small, faint text that reads "Developed by: [illegible]".

A „New Game” gombra kattintva új játékot kezdhet, az 1. szinttől kezdődően. Rákattintva automatikus elindul a játék. A további utasításokat a játékban a képernyő felső részén olvashatja.

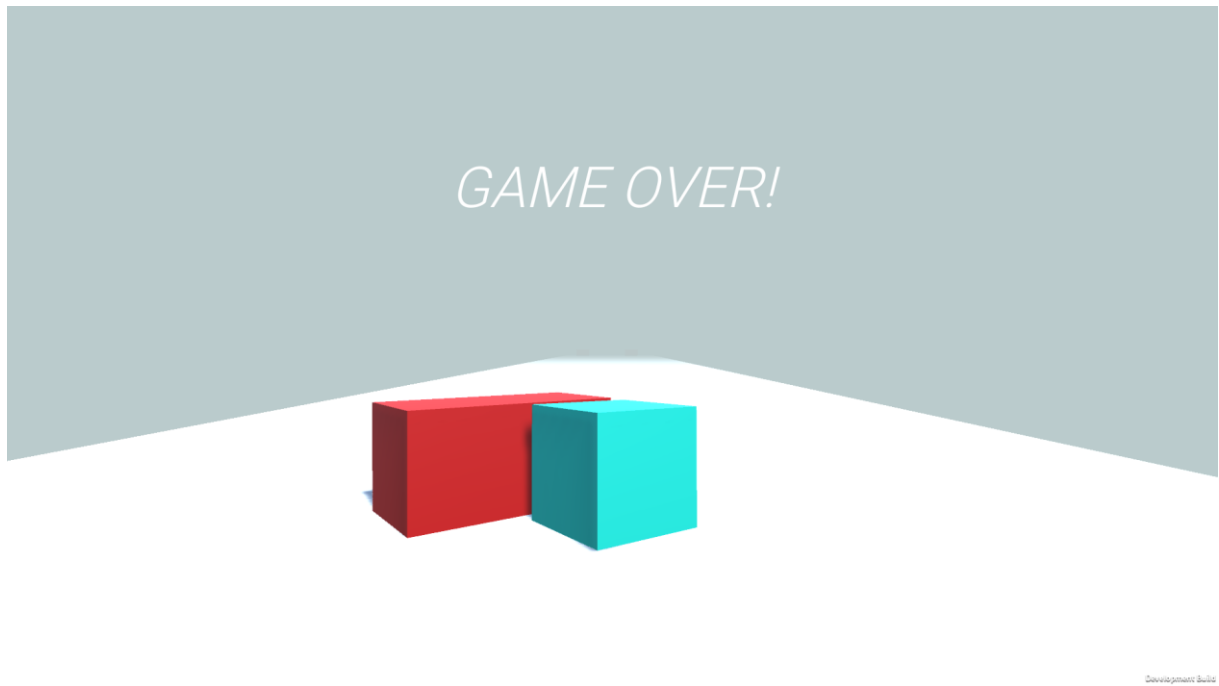


A játék célja a „W” „A” „S” „D” gombok vagy a nyilak segítségével elkerülni a szembe jövő akadályokat. (Későbbiekben ugrani is lehet a „Space” gombbal)

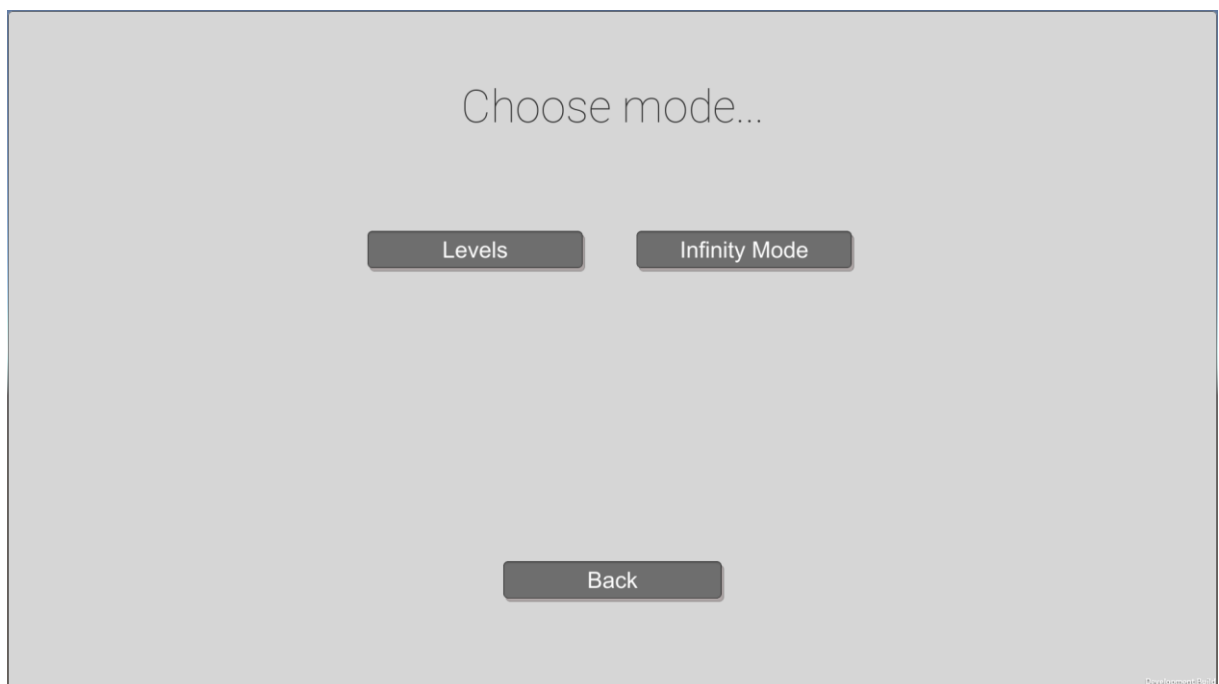


Abban az esetben, ha ez nem sikerül, vagy leesünk a pályáról, a játék kiírja „GAME OVER!” és újakezdi az adott szintet.

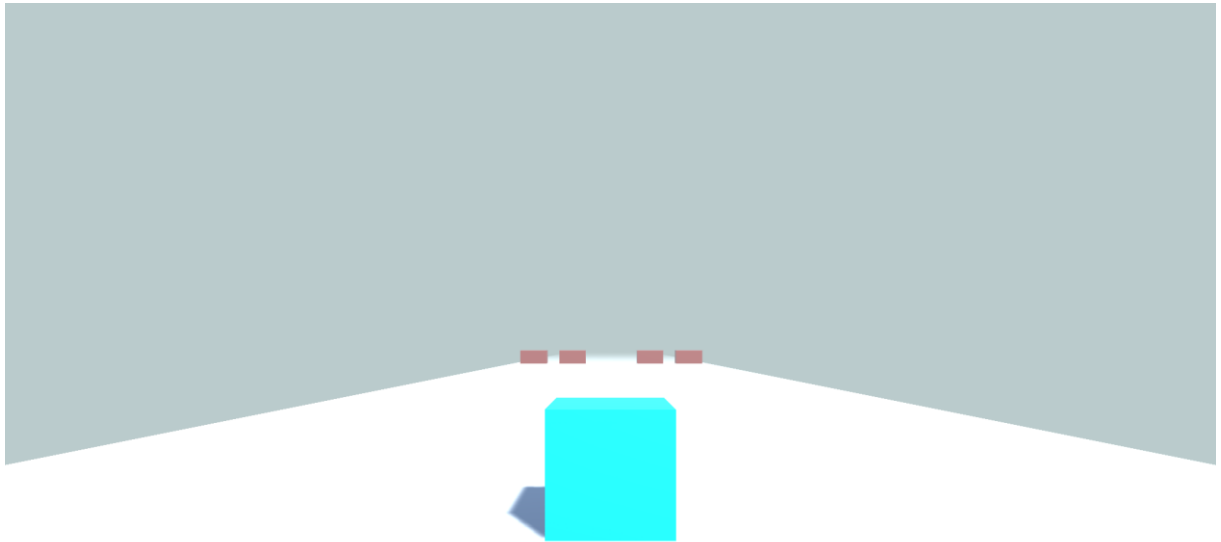




A „Load Game” gombot választva kiválaszthatjuk, hogy a rendes játék melyik szintjével akarunk játszani („Levels” – menüpont), vagy az „Infinity Mode” -t, mely egy végtelen játék, ami az idő előrehaladtával egyre gyorsabb és gyorsabb lesz.

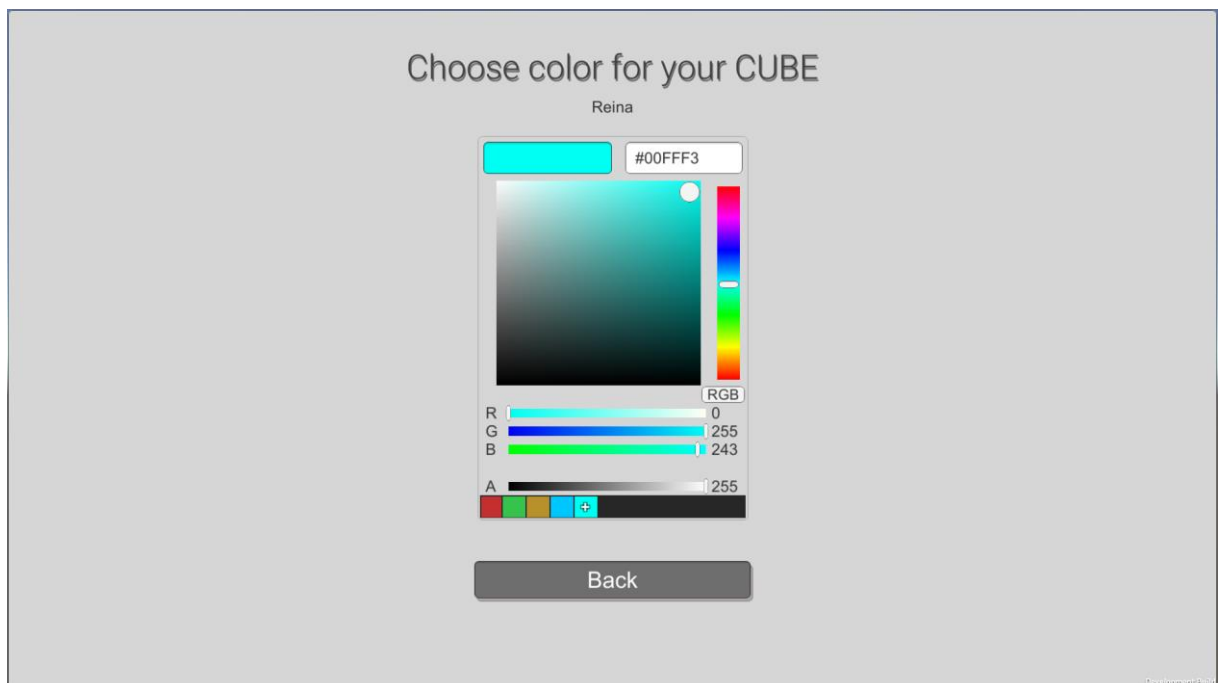


### Infinity Mode:



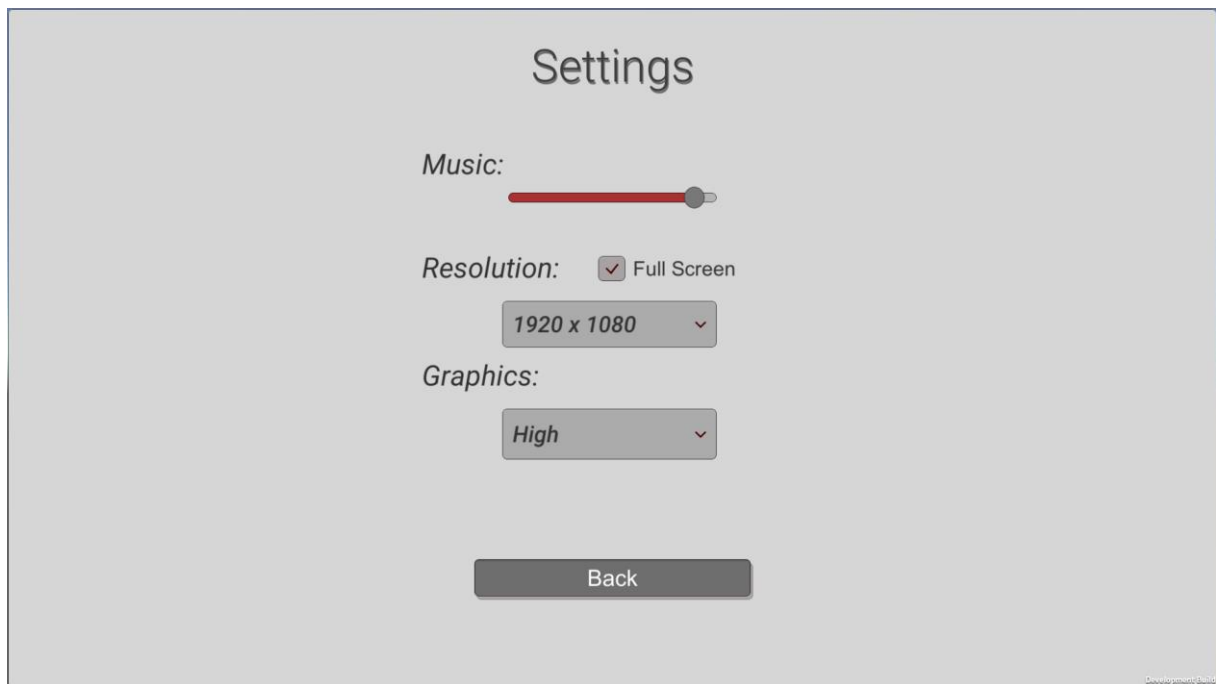
Development Build

A „Costumize” gombra kattintva elérhető az a felület, mellyel a játékos által irányított kocka színét lehet változtatni. Felül lehet megtekinteni, hogy pontosan melyik felhasználó van bejelentkezve a játékba. A változtatások a „Back” gomb hatására lépnek érvénybe.



Development Build

Végül, de nem utolsó sorban, a „Settings” menü, melyben a program egyes tulajdonságait lehet állítani: a zene hangerejét, a képernyő felbontását, teljesképernyős vagy ablakos nézetet szeretnénk és a grafika beállítását.



A „Quit” gombbal pedig bezárhatjuk a játékot.

### Hibák

Hiba	Megoldás
– A resolution dropdown duplikálva jeleníti meg az adatokat	Ez egy ismert bug Unity-ben, igazán hatékony megoldás nem igazán van még rá, dolgoznak rajta a fejlesztők

## Fejlesztői dokumentáció

### Követelményspecifikáció

Az eddigi tanultakat és azok kiegészítéseit felhasználva, egy olyan program megírása volt a cél, amely egy, a mi érdeklődési körünkbe tartozik és szívesen foglalkozunk vele. Szinte mindegy volt milyen platformra, milyen nyelven, milyen eszközökkel, csak legyen egy működő program, amelyet teljes egészében mi fejlesztettünk. Fontos volt, hogy Visual Studio-ban fejlesszünk és ha lehetséges C# vagy valamilyen C-hez kapcsolódó nyelvben.

A szoftvernek, magának egy szórakoztató játéknak, felhasználóbarátnak és könnyen kezelhetőnek, felhasználó felületnek pedig egyértelműnek kell lennie.

Az optimalizáltság is fontos tényezője volt a programnak. Bár az én játékom kicsi, nem annyira megterhelő a számítógépnek, de nekem is gondolnom kellett az optimalizáltságra. Egy példának hoznám az Infinity Mode-t, amiben egy kritikus pont volt, hogy a már kikerült akadályok törlésre kerüljenek, mivel ha a játékos elég jó, idővel egyre több és több memóriát evett volna a program és bár ahhoz, hogy ténylegesen probléma legyen, rengeteg időnek kellett volna elteltie, így is ez egy lehetséges eset volt.

### Rendszerspecifikáció a felhasználói dokumentációnál.

### Unity

A játék készítéséhez az egyik legjobb C nyelv alapú programot használtam a Unity-t. A Unity egy cross-platform játékmotor, melyet a Unity Technologies hozta létre és továbbra is fejleszt. Legelőször 2005-ben jelent meg egy Apple konferencián. 2018-ban a játékmotor ki lett terjesztve több mint 25 különböző platformra. A Unity használható mind 3D, mind 2D játék megvalósítására, virtuális és kibővített valóság kreálására is. Érdekesség, hogy a programot nem csak a videójáték fejlesztők körében használják, hanem átvette a film, autóipar, építészet és a gépipar is. (Wikipedia)

Rengeteg új verzió jött létre a kezdetek óta, a legutolsó stabil verzió a 2019.2.8., melyet 2019. októberében hoztak nyilvánosságra. A játék megírása közben én a Unity 2019.1.5f1 (64-bit) verziót használtam.

A Unity játékmotor scripting API-t ajánl elsődlegesen C#-ban. A scripting API jelentése alkalmazás programozó interfész (Application Programming Interface) és ezek jelentik a kapcsolódási pontokat a játékmotorral, mely engedi, hogy vezéreljük azt. (MichaelHouse, 2014)

Egy másik fontos dolog, melyről írni szeretnék a Unity Asset Store. A ColorPicker (színválasztó) asset-et, amit használok a programomban is innen töltöttem le, mint külső osztálykönyvtárat. Ez egy kollektív oldal, ahova különböző felhasználók, különböző előre megírt és megalkotott programokat, programrészeket töltenek fel és ezeket megosztják a nagyvilággal. Természetesen vannak ingyenes és fizetős asset-ek is, ez főképp a szerző belefektetett munkájának függvénye. A Unity ráadásul magába a programba is beimplementálta, egy külön fülön böngészhetjük az Asset Store-t és találhatjuk meg a számunka legmegfelelőbb elemet.

### Visual Studio

Mint már említettem, a Unityhez tartozó scripting API-ket C# nyelven írhatjuk meg. Ehhez a Microsoft Visual Studio 2017 Community Edition változatát használtam, mivel a 2019-es verzió még nem kompatibilis a Unity-vel. A Microsoft Visual Studio egy integrált fejlesztői környezet (IDE – Integrated Development Environment), melyet a Microsoft fejleszt. Használják számítógépes programok, weblapok, web alkalmazások, web szolgáltatások és mobil alkalmazások fejlesztésére is.

A Visual Studio-nak tartalmaz egy kód szerkesztés támogatást is (IntelliSense), továbbá egy integrált debugger-t és egyéb beépített eszközöket, mint például: code profiler, form designer-t GUI-k építéséhez stb.

A C# mellett 35 különböző programozási nyelvet támogat, többek között C, C++, JavaScript, XML, HTML, de plug-in használatával a Python is elérhetővé válik.

A legutolsó verzió a Microsoft Visual Studio 2019. (Wikipedia)

## Kezdeti dokumentáció:

Minden projektnek 5 fázisa van:

1. Konceptió és kivitelezés: Ebben a szakaszban átgondolod a projektjavaslatot vagy az üzleti esetet. Itt van az ideje részletezni hogyan építed fel a projektet, úgy, hogy az megfeleljen a érdekelt felek igényeinek.
2. Meghatározás és tervezés: A projektfeladatokat a hatókörnek megfelelően határozzuk meg. A feladatok rangsorolása azzal kezdődik, hogy mindegyiket felsorolják a projekt fontossági sorrendje szerint. Költségvetési becsléseket és ütemtervet készítenek, így a csapat minden tagja tisztában van a munkához rendelkezésre álló erőforrásokkal és idővel.
3. Indítás: Megkezdődik a feladatok és az erőforrások kiosztása. A csapat tagjai értesülnek a felelősségről. A munka valójában itt kezdődik, de a terv mindig a közelben van referenciaként, hogy megbizonyosodjunk arról, hogy a dolgok a pályán vannak-e. A tervek megváltoztatása és módosítása egy projekt során a folyamat része, ezért nagyszerű, ha a terv a közelben van.
4. Teljesítmény és ellenőrzés: Ez a terv felügyeleti része. A csapat állapotának frissítéseit kiértékeljük annak biztosítása érdekében, hogy a projekt előrehaladása összhangban legyen a korai előrejelzésekkel. Az erőforrások újraelosztása - ha szükséges - a projekt nyomon követése érdekében történik, hivatkozva a tervre. A valós idejű dashboard használata lehetővé teszi a projekt előrehaladásának könnyű nyomon követését és a pontos kép megőrzését.
5. Bezárás: Már majdnem a célba érkezel, de még nem vagy ott! A munka befejezéséhez az ügyfelek jóváhagyására van szükség, és ez magában foglalja a projektben részt vevő összes érdekelt fél bejegyzését is. A bezárás általában tisztítási feladatokról szól. (ProjectManager)

Minden projekt esetében fontos a kezdeti dokumentáció, melyben az ötleteket és a felépítést részletezzük. Az én esetemben ez Word dokumentummal kezdődött, melyben leírtam, mit, hogyan szeretnék felépíteni a játékkal kapcsolatban, milyen elemek legyenek és milyen struktúrában legyen felépítve maga a program. A dokumentációt a program végeztével újra elővettem, hogy megbizonyosodjak, minden előre kigondolt és kivitelezhető tervet megvalósítottam.

## Diagramok

### Package Diagram

A package diagram egy fajta szerkezeti ábra, amely bemutatja a modellelemek elrendezését és felépítését közép-nagy méretű projekteknel. Megmutathatja mind az alrendszerek, mind a modulok közötti szerkezetet és függőséget, bemutatva a rendszer különböző nézeteit.

A csomag diagramokat általában arra használják, hogy magas szintű rendszer elemeket építsenek fel. A packageket (csomagokat) egy nagy rendszer megszervezésére használjk, amely diagramokat, dokumentumokat és egyéb kulcsfontosságú szolgáltatásokat tartalmaz.

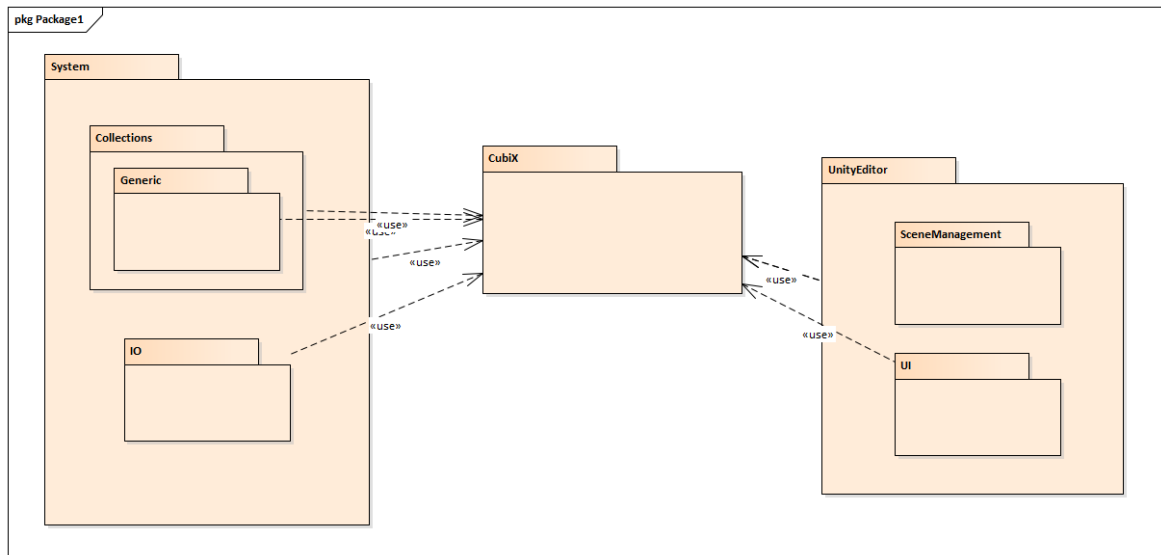
A package diagramot használhatják egy komplexebb osztálydiagram leegyszerűsítésére is, mert csoportosítani tudja az osztályokat egy csomaggá.

Jellemzői:

- A csomagok egy téglalapként jelennek meg egy kis füllel a tetején.
- A csomag neve a fülön vagy a téglalap belsejében található.
- A pontozott nyilak a függőségeket jelöli.
- Egy csomag függ egy másiktól, ha az egyiket megváltoztatjuk, annak hatására a másiknak is meg kell változnia. (VisualParadigm)

Az alábbi Package Diagramon látható, hogy a programom milyen namespace-ket használ. A Visual Studio részéről látható a System.Collections (ez kezeli a különböző listákat, tömböket és szótárakat) és a System.Collections.Generic (azokat az interfészeket és osztályokat tartalmazza, amelyek meghatározzák az általános gyűjteményeket) továbbá a System.IO, mely segítségével lehet írni és olvasni fájlokat vagy adatforrásokat. A login.csv fájl miatt használja ezt a CubiX.

Másik félről lehet látni UnityEditor-t, mely tartalmazza a SceneManager-et (ahogy a neve is adja ez felelős az egyes menük és egyéb jelenetek váltásáért) és a UI-t, mely a vizuális elemek programozhatóságát teszi lehetővé.





## UseCase Diagram

UseCase diagram az elsődleges formája a rendszer/szoftver követelményeknek egy új szoftveres program fejlesztése esetén. Főképp az elvárt viselkedést és nem a pontos metódust írják le. A UseCase diagram lehet írott és vizualizált bemutatás is. A kulcsszerepe a UseCase diagramnak, hogy segít egy szoftvert létrehozni a végfelhasználó szemszögéből. Ez egy hatékony módszer kommunikálni a rendszer viselkedését a felhasználó szempontjából, azáltal, hogy meghatározza az összes külsőleg látható rendszer viselkedést.

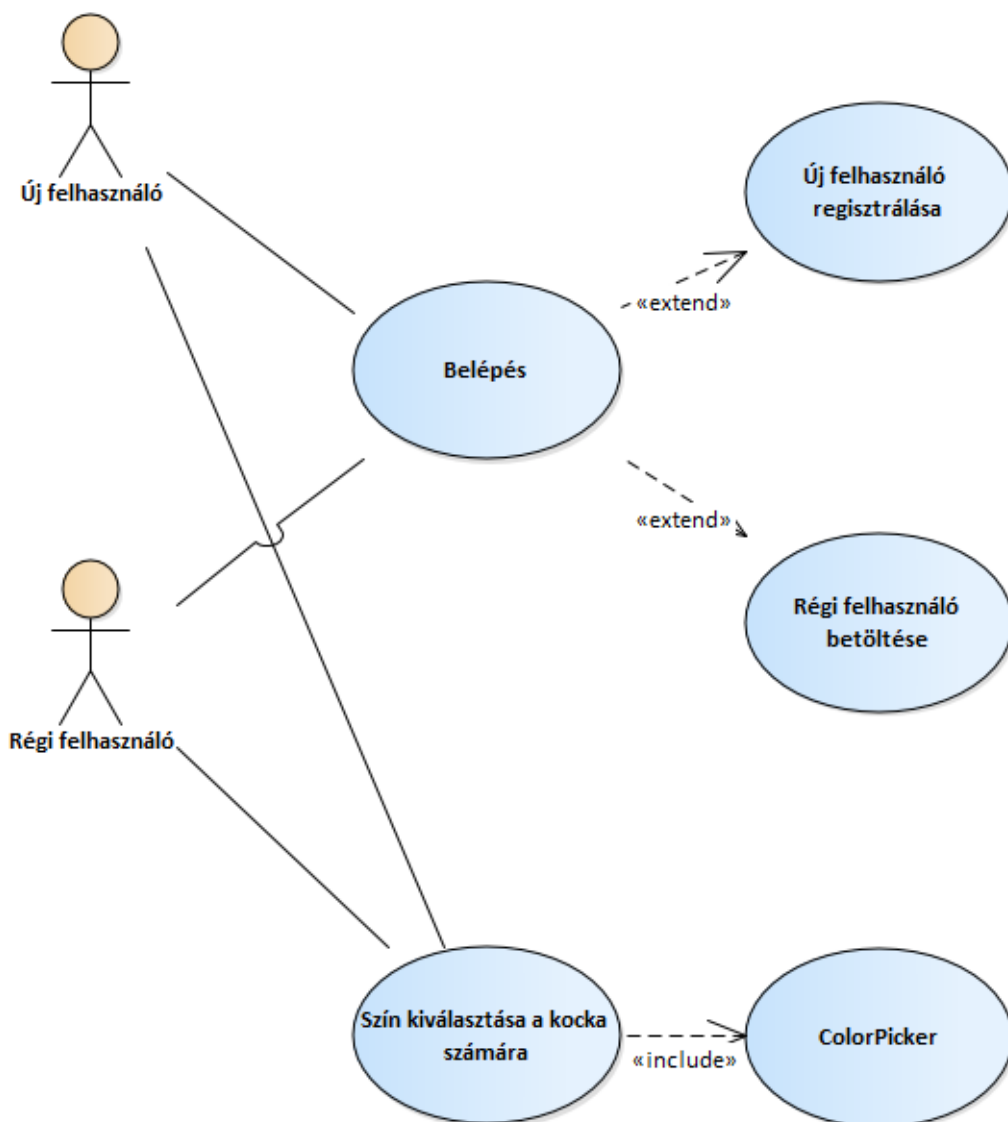
A Use Case diagram általában egyszerű, nem mutat részleteket:

- Összegez bizonyos kapcsolatok a use case-k, az actor és a rendszerek között
- Nem mutatja a sorrendet, mely lépések hatására éri el a célját a use case

A UseCase diagramokat általában a fejlesztés korai szakaszában hoznak létre és a fejlesztők gyakran használják a következő okok miatt:

- Meghatározza a rendszer kontextusát
- Megmutatja a rendszer követelményeit
- Érvényesíti a rendszer felépítését
- Vezeti az implementációkat és teszteseteket generál
- Az elemzők és a domain szakértők által kidolgozott (VisualParadigm)

Az én UseCase Diagramomon egy új és egy régi felhasználó perspektíváját mutatom be. Az új felhasználó belépéskor megad egy felhasználónevet, melyet a rendszer végig futtat a CSV állományon és mivel olyan név nem szerepel a listában hozzáadja ahhoz és kreál egy alapértelmezett szín értéket mellé. A játék során ezt az új felhasználó a régivel egyetemben változtathatja, mely szintén kommunikál a CSV fájlal és felülírja az alapértelmezett, vagy régi felhasználó esetén a már meglévő szint. A régi felhasználó esetében a belépés során betöltés történik, azaz a rendszer megtalálta a megadott felhasználónevet a fájlban, és annak adatai betöltve engedi tovább a játékost.



## Class Diagram

A Class Diagram, avagy osztály diagram megmutatja a statikus felépítést egy rendszerben. Tartalmazza az osztályokat és az osztályok közötti kapcsolatokat.

Az osztály hasonló szerepű objektumok csoportja a rendszerben, amelyek két részből állnak:

- Szerkezeti elemek (attribútumok), melyek definiálják mit tudnak az osztály elemei
  - Az osztály egy objektumának állapotát képviseli
  - Az osztály statikus vagy szerkezeti leírása.
- Viselkedési elemek (műveletek), melyek definiálják mit tudhatnak az osztály elemei
  - Definiálja, ahogy objektumok egymásra hatnak
  - Az osztály dinamikus vagy viselkedési leírása. (VisualParadigm)

Az alábbi osztálydiagramban a leszármazást ábrázoltam. Mint látható, majdnem az összes osztály a MonoBehaviour Unity osztály leszármazottja, amelyek pedig nem, azoknak pedig valamilyen őse a MonoBehaviour. A diagram bal szélén megfigyelhető a ColorPicker mint külső osztálykönyvtár felépítése, ennek is nagy részének őse a MonoBehaviour class, de az sokkal kevesebb osztályt érint. Ezáltal a ColorPicker még általánosabbá válik és ténylegesen beépíthető szinte bármelyik programba.



A főbb elemek kibontva a következők:

#### GameManager:

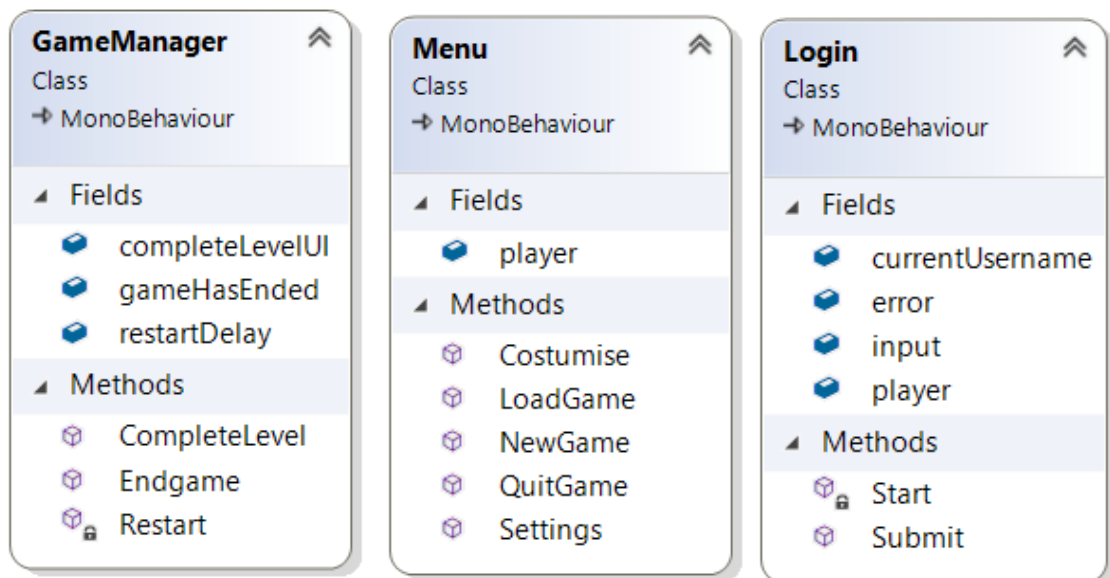
Amint látható, főképp a szint menedzseléséért felel. Üzenetet kap arról, ha egy szintet újra kell indítania, vagy épp a következőt betöltenie.

#### Menu:

Ez felel a menü kezeléséért. A gombokra kattintva hívhatók meg a függvények, amelyet tartalmaz. Minden egyes függvény egy új jelenetet jelenít meg, tehát SceneManagement-ben is szerepet játszik.

#### Login:

A csv fájl olvasása itt történik meg. Az error változó a hiba kiírására szolgál, abban az esetben ha a felhasználó nem ad meg semmit felhasználónévnek



## Sequence Diagram

A Sequence Diagramok olyan interakciós diagramok, amelyek részletezik a műveletek végrehajtását. Két objektum közötti interakciót mutatják, időfókuszban vannak és vizuálisan mutatják a kölcsönhatások sorrendjét a diagram függőleges tengelyének felhasználásával, hogy ábrázolják milyen üzenetek mikor lettek elküldve.

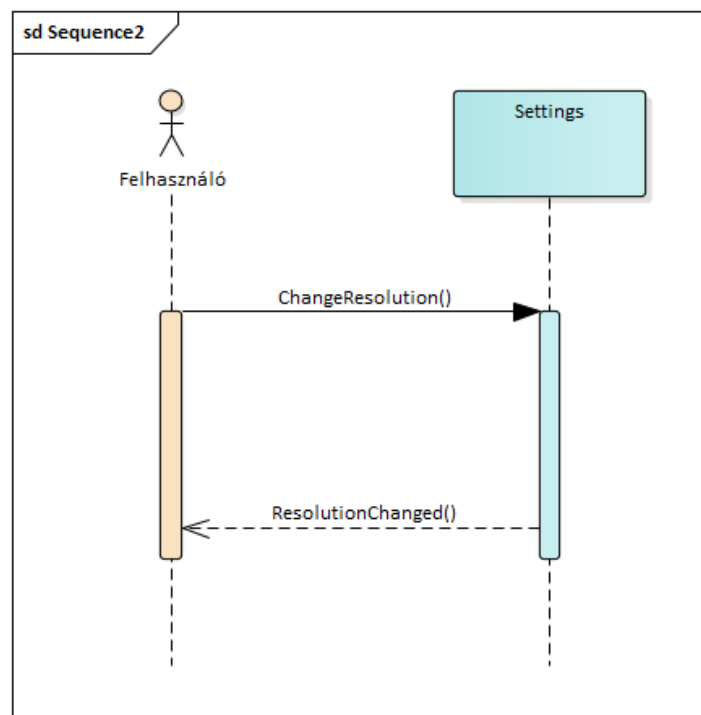
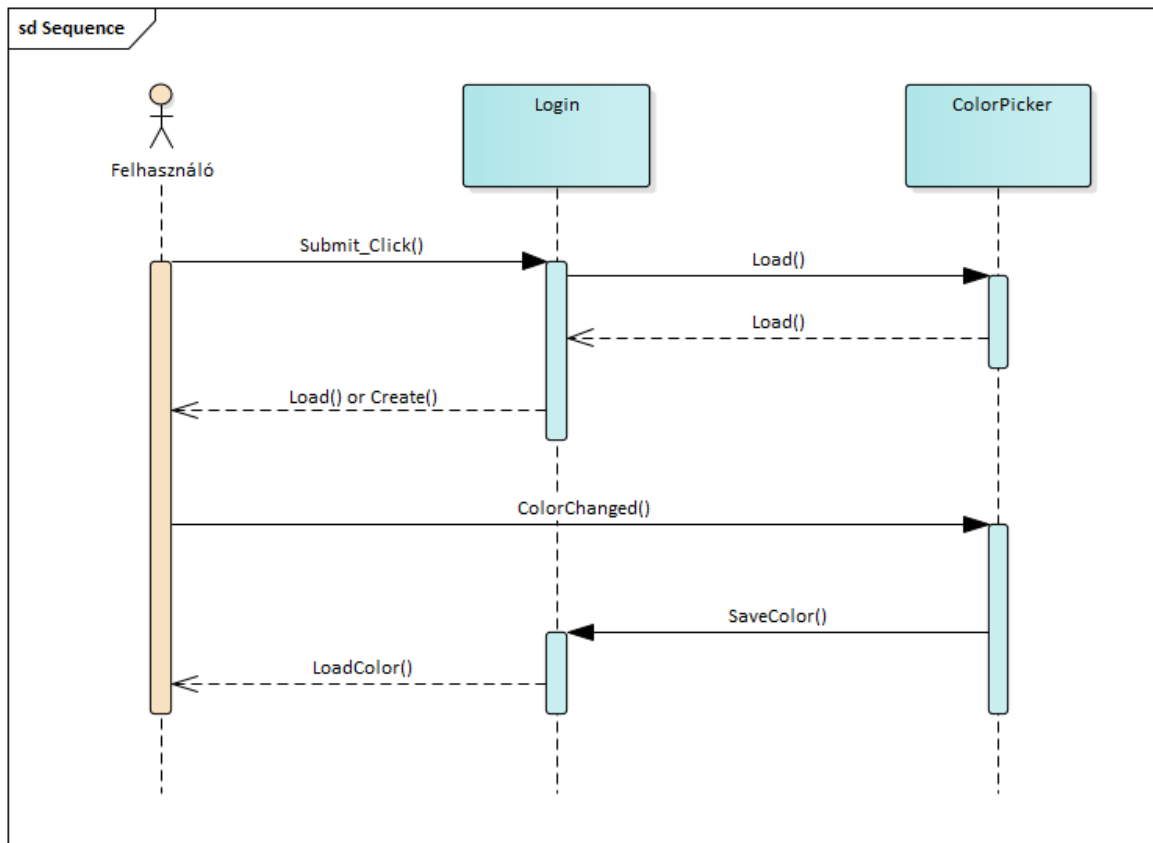
Meghatározza:

- Az interakciókat, amelyek egy olyan együttműködésben zajlanak, mely megvalósítja a felhasználási esetet vagy egy műveletet.
- A magas szintű interakciókat, amelyek a program felhasználója és a program között, a rendszer és más rendszerek között, vagy alrendszerek között történik.

Modellezi:

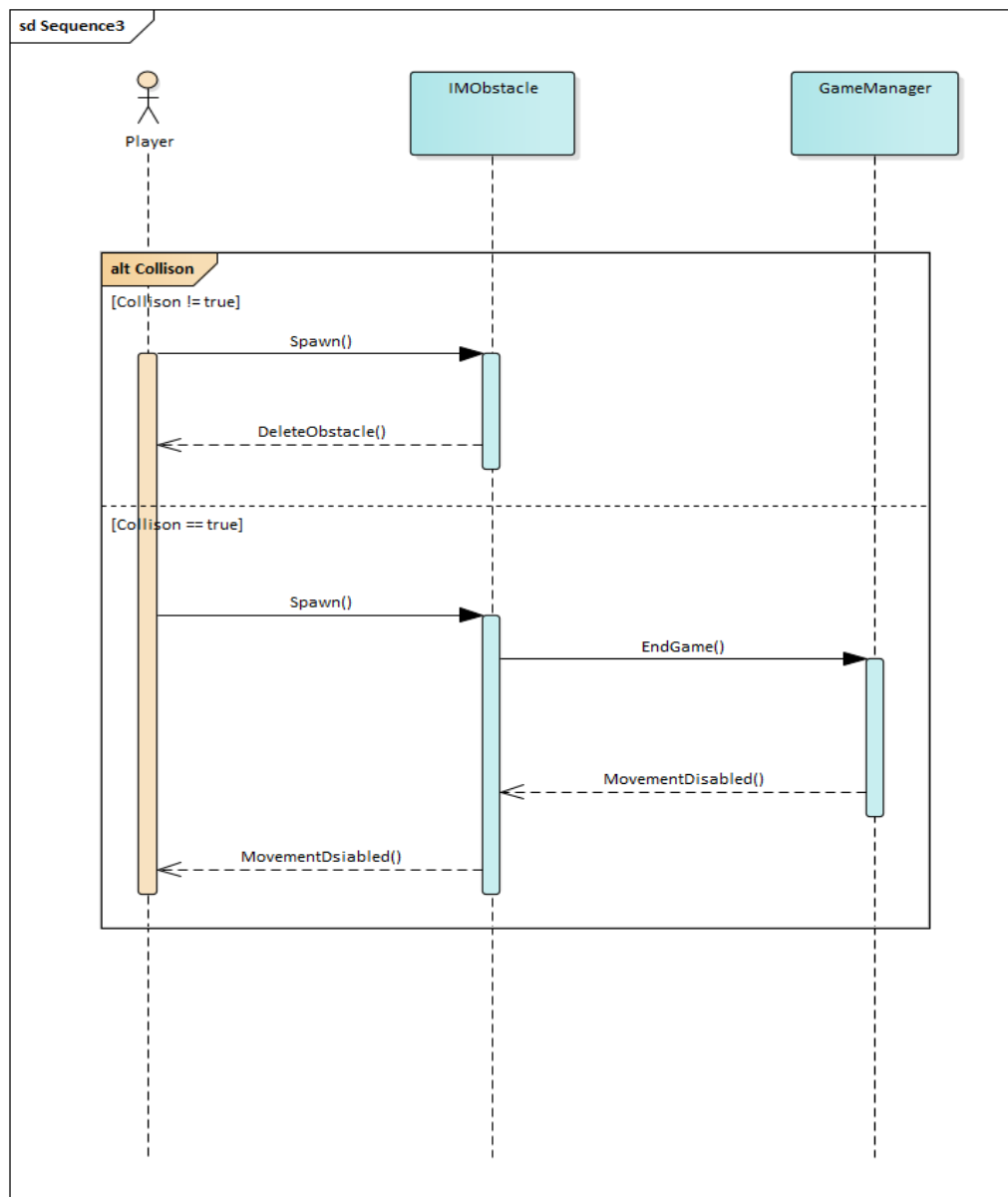
- A magas szintű interakciókat a rendszer aktív objektumai között.
- Az interakciókat az objektumpéldányok között együttműködés által, amely megvalósítja a felhasználási esetet
- Az interakciókat objektumok között együttműködés által, mely megvalósít egy műveletet
- Vagy a modell általános interakcióit (bemutatva az összes lehetséges utat az interakción keresztül), vagy az interakció konkrét eseteit (csak egy utat mutatva az interakción keresztül) (VisualParadigm)

Az első szekvencia diagram a Login (,mely a login.csv-t használja) és a ColorPicker kapcsolatát mutatja be. A felhasználó a Login képernyőn a felhasználónév beírása után rákattint a Submit gombra, mely üzenetet küld a login.csv-nek, amely a ColorPicker-től kapott színt betölti és visszaadja az irányítást a felhasználónak ezután. Majd az idő elteltével, ha a felhasználó változtatni szeretné a kocka színét akkor azzal üzenetet küld a ColorPicker-nek a változtatásra, majd az elmenti a login-csv-be, ahonnan pedig visszaadja a vezérlést úgy, hogy a megadott színt tölti be a felhasználónak.



A második szekvencia diagramom a felbontás változtatását taglalja. A felhasználó a változtatással üzenetet küld, majd a Settings továbbítja magának a programnak, az végrehajtja a változtatásokat, majd pedig visszaadja a vezérlést a felhasználónak.

A harmadik szekvencia diagram az alábbiakban tekinthető meg és egy kicsit több magyarázatot is igényel. Az Infinity Mode-n belül értelmezett ütközést írja le. Van egy Player (a kocka és az azt irányító felhasználó), egy IObstacle és egy GameManager. Azzal, hogy a játékos elkezd játszani a játékmóddal elkezd megjeleníteni a játék az akadályokat. Ekkor jön szóba az „alt”. Amennyiben a játékos sikeresen kikerülte az akadályt, az esetben egy kis idő elteltével ki is törli azokat, ez addig folytatódik ameddig valóban ütközés nem következik be, de minden esetben új és új akadályokkal való kommunikációt igényel, kétszer nem fut le ugyanazzal az Obstacle-el. Viszont, ha a játékos ütközést szenvedett el, akkor az Obstacle az EndGame() metódus lefuttatását kéri a GameManager-től, melynek hatására a vezérlés úgy adódik vissza, hogy a játékos mozgását a GameManager letiltja.





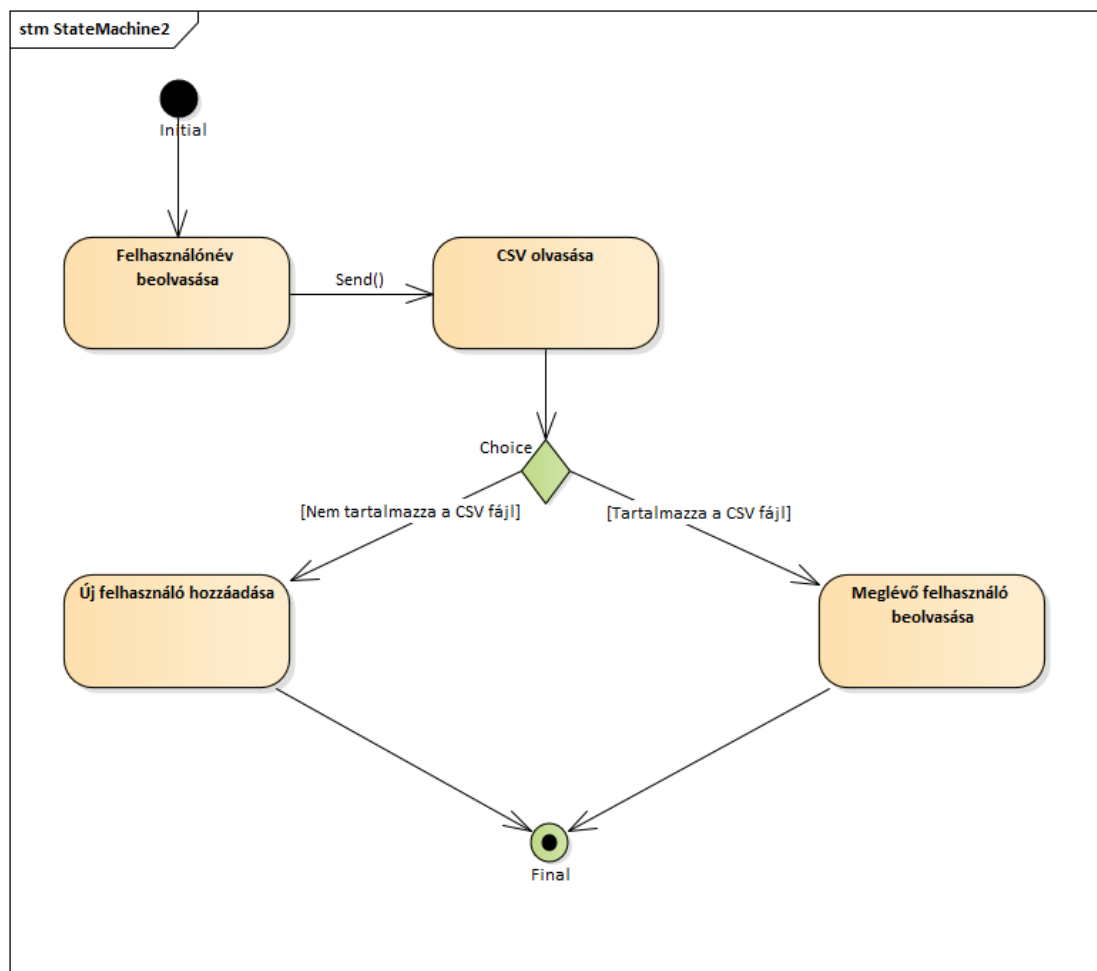
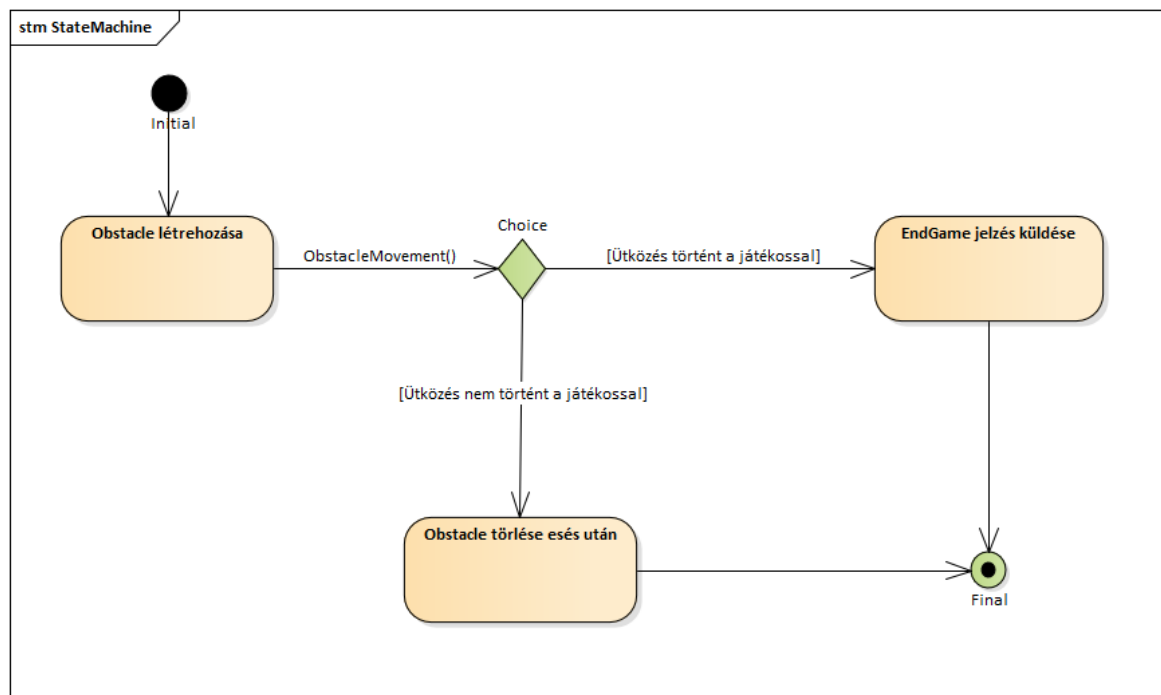
## State Machine Diagram

Egy egység viselkedése nem közvetlen eredménye az ő bemenetének, az előző állapotától is függ. Egy entitás történelmének modellezésére a legjobb módszer egy State Machine diagram. Ez megmutatja az entitás különböző állapotait és azt is hogy egy-egy esemény bekövetkezésekor, hogy válaszol az entitás, miközben egyik állapotból a másikba mozog. Egy rendszer dinamikus környezetének lemodellezésére használják.

A State Machine diagramot általában az objektum állapotfüggő viselkedésének leírására használják. Egy objektum különbözően reagál ugyanarra az eseményre, annak függvényében, hogy milyen állapotban van jelenleg. A State Machine diagrammokat általában objektumokra alkalmazzák, de nem lehet alkalmazni bármelyik elemre, amelynek viselkedése befolyásol más entitasokat, mint például az actorok, use case.k, metódusok, alprogram rendszerek stb. és általában ezeket interakciós diagrammokkal együtt használják (pl.: Sequence Diagram) (VisualParadigm)

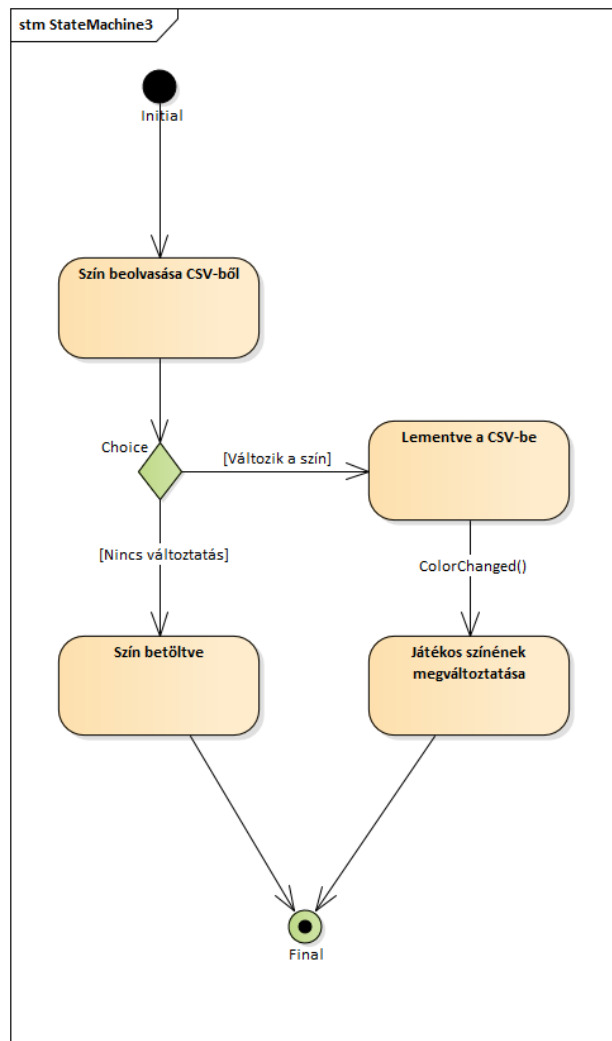
Az állapot egy objektum attribútumértékeinek és hivatkozásainak absztrakciója. Az értékkészleteket állapotokba csoportosítjuk olyan tulajdonságok szerint, amelyek befolyásolják az objektum bruttó viselkedését. (Rumbaugh)

Az első State Machine diagram az akadály játékkal való ütközés állapotváltozásait mutatja be. A belépés után az Obstacle létrehozott állapotban van és megkezdni a mozgását. Ekkor elérkezünk egy Choice-hoz, mely az ütközés meglétét vizsgálja. Amennyiben ez megvalósult az Obstacle EndGame jelzést küld és ezzel kilépünk az Obstacle életútjából. Ha nem történt ütközés, akkor miután az Obstacle leesett a pályáról töröljük azt és szintén kilépünk, mivel, így megszűnik az objektum létezni.



A következő diagram a CSV fájl életútját elemzi. Belépünk és elvégezzük a felhasználónév beolvasását, elküldjük a CSV fájlunk, aki elkezd megkeresni a megadott nevet. Szintén két eset közül választhatunk. Nem találtuk meg, ez esetben a fájl új felhasználónevet ad hozzá a meglévők után, ha viszont megtaláltuk beolvassuk azt. Mindkét eset végén elérkezünk a kilépő ponthoz.

Az utolsó State Machine Diagram pedig a következő:



A belépés hatására beolvassuk a színt, mely a Choice után két eset történhet. Ha változik a szín lementjük a CSV-be felülírva az előzőt, ezután a megváltoztatott színt betöltjük a kocka színének, viszont, ha nincs változtatás csak simán betöltjük a már meglévő színt a fájlból.

## Deployment Diagram

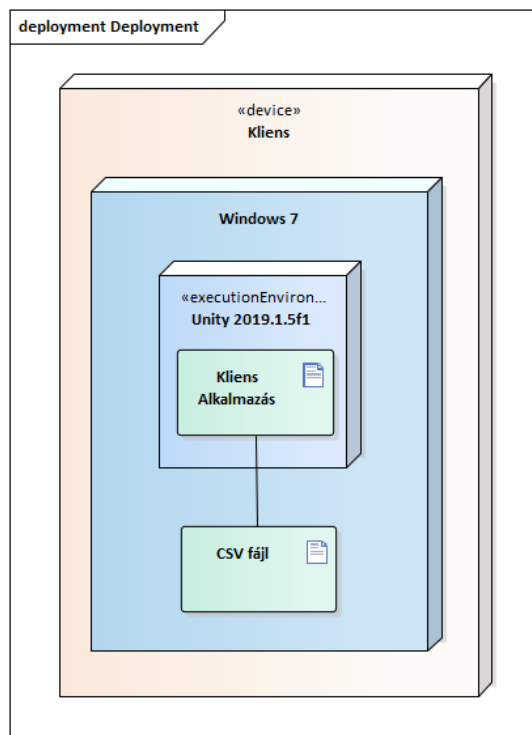
A Deployment Diagram egy olyan diagram, mely megmutatja a futási idő alatti feldolgozó csomópontok és az azokon élő komponensek konfigurációját. A telepítési diagram egy fajta szerkezeti diagram, melyet a objektum-orientált rendszer fizikai szempontok szerinti lemodellezésére használják. Gyakran modellezik a rendszer statikus telepítési nézetét (a hardver topológiája).

Célja:

- Megmutatják a run-time rendszer felépítését
- Meghatározzák a hardvert, amelyet a rendszer implementációjára használnak és összekötik a hardver különböző elemeit
- Modellezik a fizikai hardver elemeket és a közöttük lévő kommunikációs utat
- A rendszer felépítésének tervezésére használják
- Hasznosak a szoftveralkatrészek vagy csomópontok telepítésének dokumentálására is

A Deployment diagramok fontosak a beágyazott, kliens/szerver és elosztott rendszerek vizualizációjában, meghatározásában és dokumentálásában és a futtatható rendszerek menedzselésében.

A telepítési diagram egy speciális osztálydiagram, amely a rendszer csomópontjaira fókuszál. Grafikailag a telepítési diagram csúcsok és ívek gyűjteménye. (VisualParadigm)



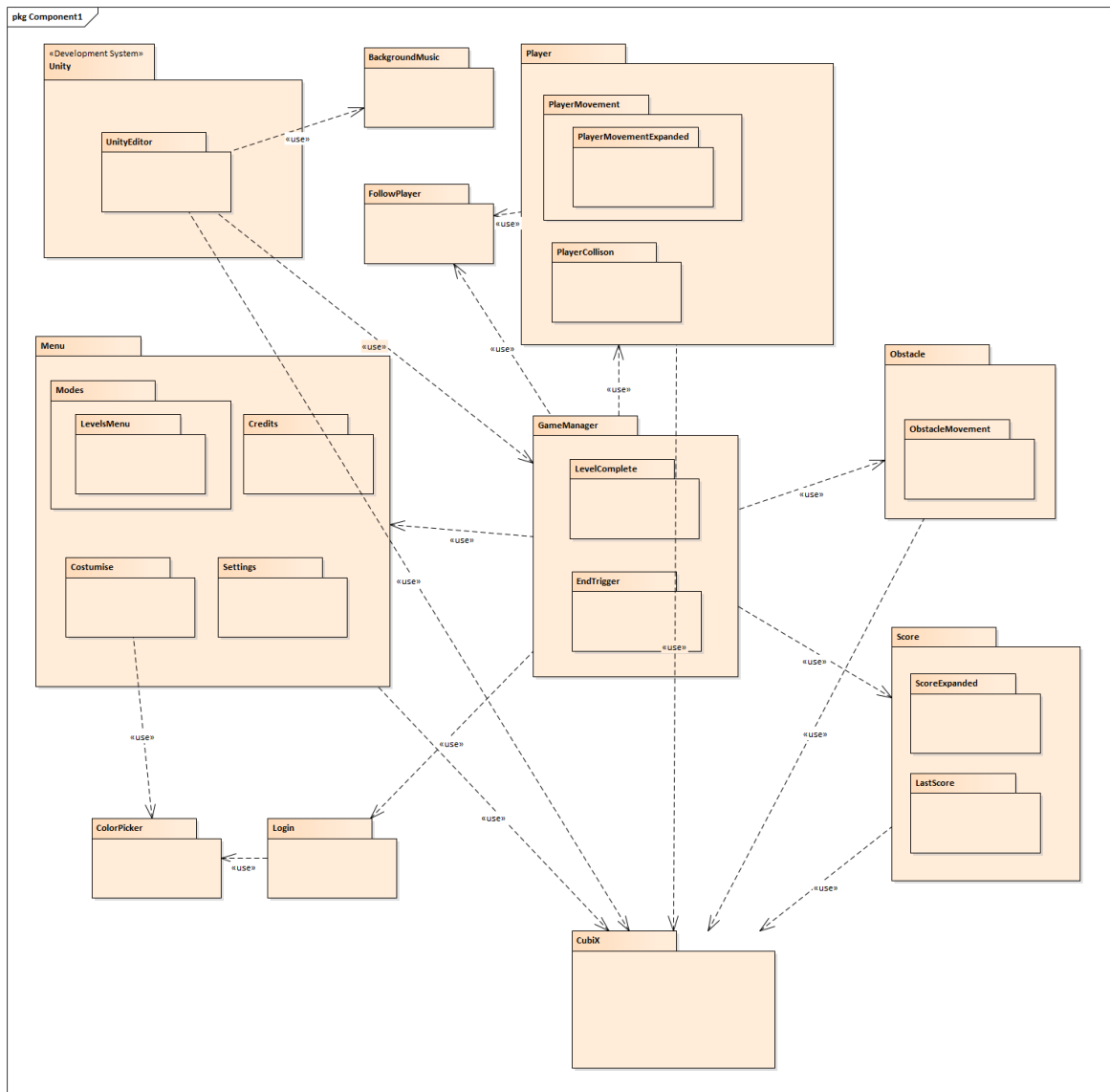
A fenti telepítési diagram magyarázata a következő: A programnak két fájlra van szüksége, az egyik maga a Kliens a másik a CSV fájl, amiből olvas. Mindkettő ajánlatos legalább Windows 7-es verzióján futtatni. A klienst a Unity 2019.1.5f1 verzióján készítettem, és a Unity gyakori változtatása miatt nem érdemes korábbi verzióval futtatni. A CSV fájl olvasása bármilyen CSV olvasóval lehetséges, én a Visual Studio olvasóját használtam és ezáltal kötöttem az asszociációs kapcsolatot maga a Kliens és a CSV fájl között.

## Component Diagram

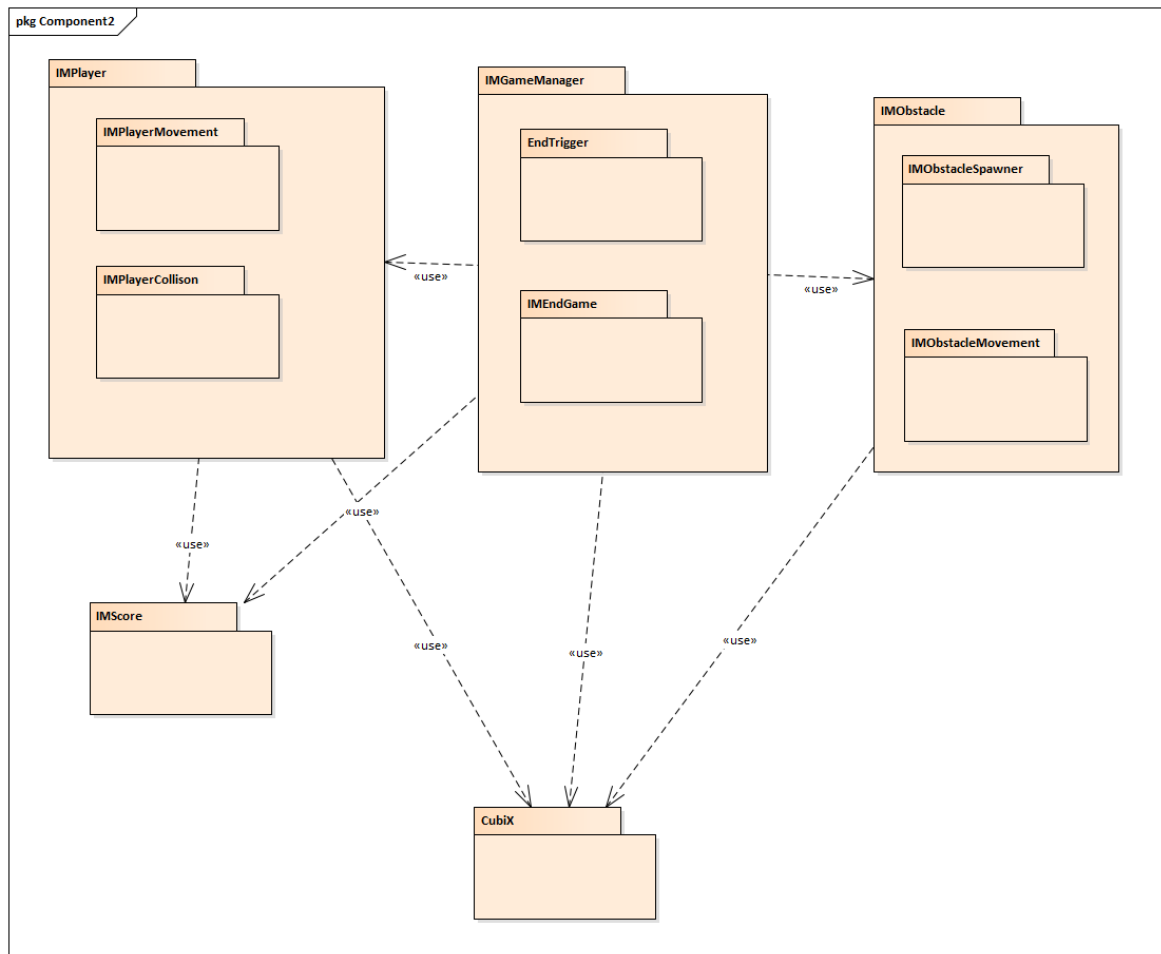
A komponensdiagramokat az objektum-orientált rendszerek fizikai szempontjainak modellezéséhez használják, amelyeket alkotóelem-alapú rendszerek megjelenítéséhez, meghatározásához és dokumentálásához, valamint futtatható rendszerek előzetes és fordított tervezéssel történő felépítéséhez használnak. A komponensdiagramok alapvetően osztálydiagramok, amelyek a rendszer olyan összetevőire összpontosítanak, amelyek gyakran a rendszer statikus megvalósítási nézetének modellezésére szolgálnak.

Egy komponensdiagram a fejlesztés alatt álló tényleges rendszert különféle magas szintű funkcionalitásokra bontja. Mindegyik elem a teljes rendszeren belül egy egyértelmű célért felel, és más alapvető elemekkel csak a minimális szinten lép interakcióba. (VisualParadigm)

Az alábbi Component Diagramon látható az, hogy a CubiX milyen főbb csomagokból épül fel és ezeknek milyen a kapcsolata. Mint látható a program a Unity-n belül a UnityEditor-t, mint csomagot használja. A UnityEditor kapcsolatban áll a BackgroundMusic csomaggal, a GameManager-rel és magával a játékkal természetesen. Ha jobban megnézzük, látható, hogy a program maga 4 főbb csomagra osztható, ezek a Player (játékos), Obstacle (akadály), Menu és a GameManager, mely a játék menedzselését teszi lehetővé, ezért szinte az összes package-t használja valamilyen módon. A Player és az Obstacle egymáshoz hasonlóan a Movement-ért felel, plusz a Player az ütközések kezelését is végzi. A Menu és a benne található csomagok építik fel a felhasználói felület igen nagy részét. A Customize kapcsolatban van a ColorPicker külső osztálykönyvtárral, ami pedig a login.csv fájljal, ami magából a Login csomagból építkezik. A Score maga az elért eredmény képi megjelenítéséért felel. Ahogyan viszont látható, a CubiX program használja az összes fenti csomagot együtt véve.



A másik kiegészítő Component Diagram az az Infinity Mode package diagramja. A lényege megegyezik az előző diagraméval. A főbb csomagok az IMPlayer, IMObstacle és IMGameManager, kiegészítő csomag az IMScore. Természetesen a CubiX használja a fentiek közül mindegyiket.





## Tesztelés

A tesztelés egy fontos folyamat egy számítógépes program életében. A tesztek azok, amelyek során előkerülnek olyan hibák (bug-ok), amelyekre fejlesztés közben nem gondoltunk, vagy csak tüzetesebb felhasználói szintű használat után derül ki. Ezek megtalálása és kijavítása kulcsfontosságú, mert növelik a játékos élményt. Egy szimpla fejlesztőnek szüksége van egy másik pár szemre, hogy az esetleges üzemzavarokat és nem várt viselkedéseket detektálja. A tesztelőknél tudnia kell a helyes működését a programnak, szóval találnak nemkívánt hibákat, üzemzavarokat.

Amint kész lettem a játékkal, nagy boldogsággal küldtem el a barátaimnak, akik önként jelentkeztek letesztelni. A következőkben az ő tapasztalataikról szeretnék írni pár szót. Ehhez egy szimpla dokumentumot küldtem el, melyet nekik kellett kitölteni. A dokumentum ebből áll:

Pozitívum	Negatívum	Hiba részletes leírása

A pozitív visszajelzések között mindenkinél szerepelt, hogy szórakoztató a játék, amire nagyon büszke vagyok, mivel ez volt a tényleges célom. Sokan mondták, hogy az infinity mode kikapcsolódást okoz, szívesen játszanák akár vizsgaidőszakban stressz levezetéseként. Örültek a kocka színváltoztatásának is, de ebből jött is egy hiba, amelyet a mai napig nem sikerült megoldani. A szín megváltoztatása után, ha egy szintet töltünk be, a kocka színe nem változik meg, míg a Unity Editor-ban ez a változtatás végrehajtódott. A grafikai megvalósítás is egy pozitív pont volt majdnem mindenkinél. Többek között megemlítették még a változatos szinteket és játékmódokat.

A negatívként említhető, hogy egyesek túl nehéznek találták a játékot. A kocka mozgása túl gyors volt számukra, melyet csökkentettem is, így téve még élvezetesebbé a játékot. Másoknak a zene volt túl sok egy idő után, ezt egy egyszerű hozzáadott zene csúszka oldotta meg a „Setting” menüben.

Az infinity mode során a játékos csak az ütközés, avagy „GAME OVER!” után látta az elért pontszámát, melyek egy kicsit elvettek a versenyszellemű játékosok élményéből. Ennek megoldására egy a játék során eltelt időt mutató TextBox-ot tettem be, mellyel mindenki láthatja, hogy éppen mennyi ideje csinálja. Egy ember pedig rámutatott pár helyesírási hibára is. Mindezekhez hozzátartozik még, hogy a játék elég nyers még és további optimalizációra szorul.

A hibák megoldása összegezve:

Hiba	Megoldása
– Túl gyors	– A kocka sebességének csökkentése
– Idő után idegesítő zene	– Zene csúszka a „Settings” menüben
– Nincs elért pontszám Infinity Mode alatt	– Az elért pontszám mutatásának megoldása
– Helyesírási hibák	– Kijavítva

A hiba kezelés a programban csak egy helyen kulcsfontosságú, mégpedig a Login képernyőnél, nem adható üres felhasználónév, ez esetben, ha mégis megpróbáljuk a program figyelmeztet bennünket és nem enged addig tovább. A játék többi részében semmilyen súlyos hibát nem okozhat a felhasználó.

The screenshot shows the login interface of a game called 'CubiX'. The title 'CubiX' is displayed in a large, red, stylized font. Below it, the text 'User Name:' is followed by a text input field containing the placeholder 'Enter text...'. A 'Submit' button is located below the input field. At the bottom of the screen, a red warning message reads: 'Warning! The user name can not be empty!'. The entire interface is set against a light gray background.

## Jövőbeni tervek

A játék jövőbeni fejlesztése a terveim között van, hiszen ahogyan Balázs is mondaná, nincs kész program, csak időben abbahagyott.

Többek között, több játékmódot szeretnék megvalósítani. The Wall játékmód, melyet Kalandra Fal című közkezdvelt műsor ihletett, a kockát különböző alakzatokon keresztül kell átjuttatni.

Nehézségi szintek bevezetése. Könnyű, közepes és nehéz nehézségi szint beállítása, amelyek a kocka sebességének és az akadályok sebességének gyorsításával nehezedik.

A grafika és a menü tovább csiszolását, egy modernebb kinézet elérése.

Több személyre szabható tartalom, mint például a háttér, a talaj és az akadályok színnek megváltoztatása.

Több zene, változtatás szintváltásként.

## Összegzés

A játékot az elején megterveztem és úgy érzem, hogy a terveim nagy részét valóra is váltottam. A legnehezebb részek mindenképpen a ColorPicker és a Login képernyő scriptjének megírása volt. Kifejezettem büszke vagyok a pályák kinézetére és nehézségére, és az Infinty Mode játékelményére.

Összegzésképpen mindenképp el szeretném még mondani, mennyire szórakoztató és kihívásokkal teli volt ez a feladat. Teljesen az alapoktól kellett elsajátítanom a Unity használatát és továbbfejlesztenem a megszerzett tudást, amelyet az OKJ tanfolyam során szereztem. Az egész egy élmény volt számomra és még ha nehézségekbe is ütköztem Kis Balázs mindig ott volt és segített, iránymutatást adott, ha kellett. Nélküle nem sikerült volna.

Köszönöm a RUANDER Oktatási Kft.-nek is és minden dolgozójának, akik ezalatt az egy év alatt összes létező módon támogattak és a lehető leggyorsabban jártak el minden kérdés megválaszolása esetén.

Továbbá szeretném még megköszönni barátaimnak, akik a húzós határidő ellenére is lelkesedéssel játszották a játékomat és segítettek véleményükkel és útmutatásukkal.

A játék mindenféleképpen további fejlesztéseken fog átesni a jövőben, hiszen potenciál még mindig van benne és tökéletesíteni szeretném, akár csak személyes használatra is.

## Irodalomjegyzék

**MichaelHouse** Game Development Stack Exchange [Online]. - 2014. 11 10. - 2019. 10 12. - <https://gamedev.stackexchange.com/questions/87001/what-is-scripting-and-what-is-a-scripting-api>.

**ProjectManager** How to create a project plan - Project Manager [Online]. - 2019. 10 16. - <https://www.projectmanager.com/academy/how-to-make-a-project-plan>.

**Rumbaugh James** State Machine Diagram - Visual Paradigm [Online]. - 2019. 10 16. - <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/>.

**VisualParadigm** Class Diagram - Visual Paradigm [Online]. - 2019. 10 14. - <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/>.

**VisualParadigm** Component Diagram - Visual Paradigm [Online]. - 2019. 10 17. - <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/>.

**VisualParadigm** Deployment Diagram - Visual Paradigm [Online]. - 2019. 10 14. - <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/>.

**VisualParadigm** Package Diagram - Visual Paradigm [Online]. - 2019. 10 13. - <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/>.

**VisualParadigm** Sequence Diagram - Visual Paradigm [Online]. - 2019. 10 14. - <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>.

**VisualParadigm** State Machine Diagram - Visual Paradigm [Online]. - 2019. 10 14. - <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/>.

**VisualParadigm** Use Case Diagram - Visual Paradigm [Online]. - 2019. 10 14. - <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>.

**Wikipedia** Microsoft Visual Studio - Wikipedia [Online]. - 2019. 10 13. - [https://en.wikipedia.org/wiki/Microsoft\\_Visual\\_Studio](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio).

**Wikipedia** Unity Game Engine - Wikipedia (angol) [Online]. - 2019. 10 10. - [https://en.wikipedia.org/wiki/Unity\\_\(game\\_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)).