**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Információs Rendszerek Tanszék**

HKK Online

Szerző: Témavezető:

Bessenyei Ádám Tamás Dr. Vörös Péter

Programtervező informatikus BSc. egyetemi adjunktus

**Budapest 2024**

**Tartalomjegyzék**

Tartalomjegyzék

[**1.** **Bevezetés:** 4](#_Toc197000157)

[**2.** **Felhasználói dokumentáció** 6](#_Toc197000158)

[Multiplayer gomb 13](#_Toc197000159)

[**3.** **Fejlesztői dokumentáció:** 14](#_Toc197000160)

[**Fordítás és futtatás:** 15](#_Toc197000161)

[**Architektúra:** 15](#_Toc197000162)

[**MenuScene** 16](#_Toc197000163)

[**ExistingDecksCanvas:** 20](#_Toc197000164)

[**CardListCanvas:** 21](#_Toc197000165)

[**CharacterSelectionScreen:** 22](#_Toc197000166)

[HeroInfoPanel: 22](#_Toc197000167)

[Portraits: 23](#_Toc197000168)

[**ReadyCheckSelectionScreen:** 24](#_Toc197000169)

[HeroInfoPanel: 24](#_Toc197000170)

[Decks/Deck Icon: 24](#_Toc197000171)

[PlayAndBackButtons: 25](#_Toc197000172)

[**Egyebek:** 25](#_Toc197000173)

[CardCollection: 25](#_Toc197000174)

[DecksStorage: 25](#_Toc197000175)

[CharacterAssetByName: 26](#_Toc197000176)

[SceneReloader: 26](#_Toc197000177)

[PlayerPrefsResetter: 26](#_Toc197000178)

[ 27](#_Toc197000179)

[Main camera, EventSystem 27](#_Toc197000180)

[**BattleScene** 27](#_Toc197000181)

[**SceneReloader:** 30](#_Toc197000182)

[**Visual:** 30](#_Toc197000183)

[**TopPlayerArea és LowerPlayerArea:** 31](#_Toc197000184)

[**Logic:** 34](#_Toc197000185)

[**TurnManager:** 35](#_Toc197000186)

[**Player:** 35](#_Toc197000187)

[**PlayerTurnMaker** 36](#_Toc197000188)

[**Commands:** 36](#_Toc197000189)

[Kapcsolódó osztályok és kiegészítők: 38](#_Toc197000190)

[**Dragging:** 39](#_Toc197000191)

[**DOTween:** 39](#_Toc197000192)

[**Mirror:** 39](#_Toc197000193)

[Mirror lapvető működési elve: 40](#_Toc197000194)

[**NetWorking** 40](#_Toc197000195)

[**Tesztelés** 40](#_Toc197000196)

[**Összegzés** 42](#_Toc197000197)

[További lehetőségek: 42](#_Toc197000198)

# **Bevezetés:**

Motiváció az alkalmazás elkészítéshez

A szakdolgozat a Magyar gyűjtögetős kártyajáték a “Hatalom Kártyái Kártyajáték”-on alapúló multiplayerben játszható kártyajáték és pakli szerkesztő szimulátor. Nem minden funkció és játékelem jelenik meg a játékban. Az alkalmazás felhasználja a kártyajáték egyes kártyáit és azok funkcióit hogy egy érdekes és interaktív kártyajáték jöhessen létre. Célja hogy a játékos megismerje a kártyákat és azok fő funkcióit és akár saját paklit is készíthessen

Egy tradíciónális gyűjtögetős kártyajátékhoz legalább 2 játékos szükséges aktívan és sok erőbelefektetés esetén se biztos hogy 2 ember fizikailiag képes legyen leülni és játszani egy játszmát kényelmesen. Idő, pénz, energia, távolság stb okokből meghíúsolhat az erre való törekvés. Ismerősök és saját magam is rengetegszer beleütköztem ebbe a problémába és egyik fő célja ennek az alkalmazásnak az lenne hogy olyanok is tudják élvezni ezt a játékot akik más okokból nem lennének képesek egyépként.

A játék maga nagyon öreg már 1995-ben megjelent és azóta egy nem nagy de hűséges tábort talált magának magyarok körében s bár voltak kísérletek arra hogy igazán elterjedjen akár online akár fizikai formában máshol nem igazán sikerült.

A 2010-es években már készül egy bizonyos HKK online ami megpróbálta a már akkor is rengeteg kártyát tartalmazó játékot online felületre rárakni és bár sikernek számított nem maradt fenn sokáig. Azóta rengeteg új kártyajáték és régi kártyajátéknak az online verziója jelent meg és ezek bebizonyították hogy igen is igény van online formátumban a játékosoknak ezekre a kártyajátékokra az eddig felsorolt okok miatt is de csak azért is mert gépek segítségével igazán leegyszerűsíthetjük akár a mechanikákat is.

Az alatt az idő alatt amíg dolgoztam a projektemen számomra is meglepő módon a HKK egy bizonyos szinten felélesztette a HKK online gondolatát. Habár a jelenleg létező verzió alig ha tart lépést az azóta kijött óráisok ellen mint a Hearthstone, Magic Arena vagy Yugioh Master duel. Ezek mind képesek voltak megvalósítani az online kártyajáték elméletét és sok esetben túlszárnyalni azt.

Amíg a jelenlegi HKK online jobban hasonlít egy úgy nevezett “tabletop szimulátor”-hoz ami azt takarja hogy valójában ugyanúgy játszik az ember online formában mintha valóságban játszanánk csak ahelyett hogy fizikailag mozgatnám a kártyáimat, vizuális reprezentációkat mozgatok a számítógépen és egyéb gombok/eszküzök segítségével állítja be az ember játékosok életét, paklik tartalmát és egyebeket.

Egyszerűségen felül, amiben nagy segítség lehet az alkalmazás akár kezdő akár új játékosoknak az a tesztelés gondolata. Digitális környezetben sokkal több dolgot és azok kombinációját lehet tesztelni gyorsobban és hatákonyabban legyen az egy kompetitív pakli összeállítása valódi versenyekre, vagy csak meggyőződés arról hogy valóban megéri bizonyos kártyákat beszerezni pénzért.

Utolsó motivációs pont pedig annak a gondolatnak az álma, hogy egyszer a játék elterjed akár külföldön is és bármekkora tábort szerez magának annak bemutassa mire is képes ez a játék, amin magyarok dolgoznak már több mint 30 éve.

# **Felhasználói dokumentáció**

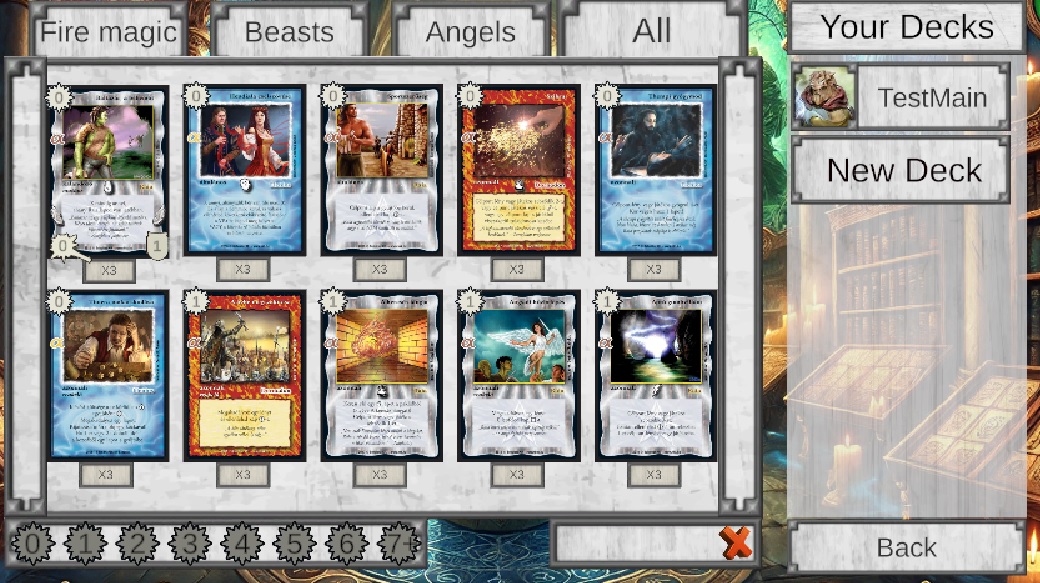
Ebben a fejezetben lesz bemutatva az alkalmazás használata, telepítése, funkciói ábrákkal kiegészítve.

A project lefrisseb verziója megtalálható az következő [github](https://github.com/Aroddn/Hkk) linken keresztül

Letöltés githubról

A program indítása után elsőnek menüoldalt láthatjuk ahol 4 gombbal tudunk interaktálni.

A Play gomba kattinva egyenlőre nem történik semmi mivel az egy majd jövőbeli fejlesztésnek a helye ami lehetővé teszi hogy egy játékos akár mesterséges intelligencia ellen is játszhasson.

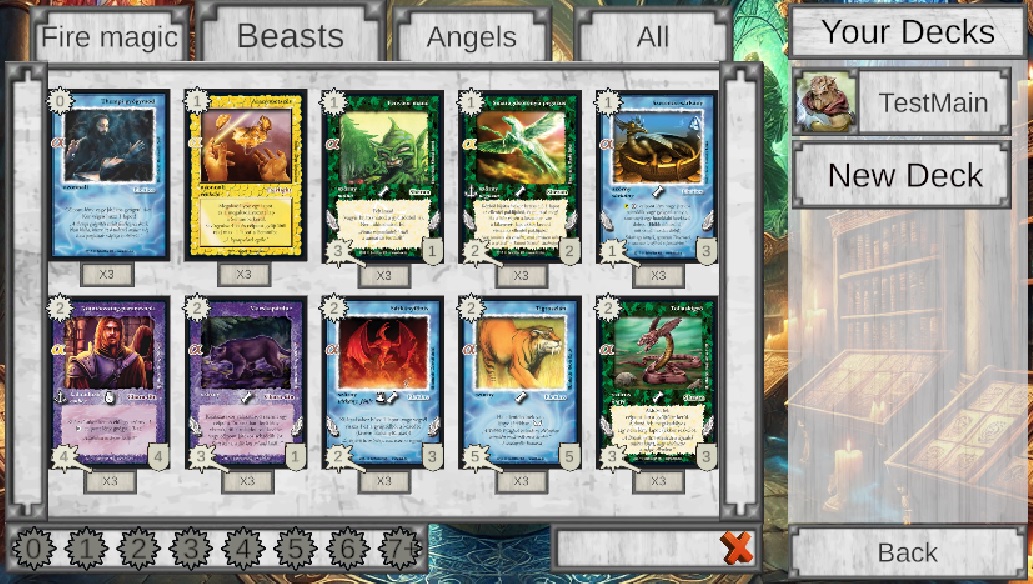
A “Collection” gombot megnyomja egy új oldalt látunk, ahol a játékos kártyái és pakliaji vannak eltárolva

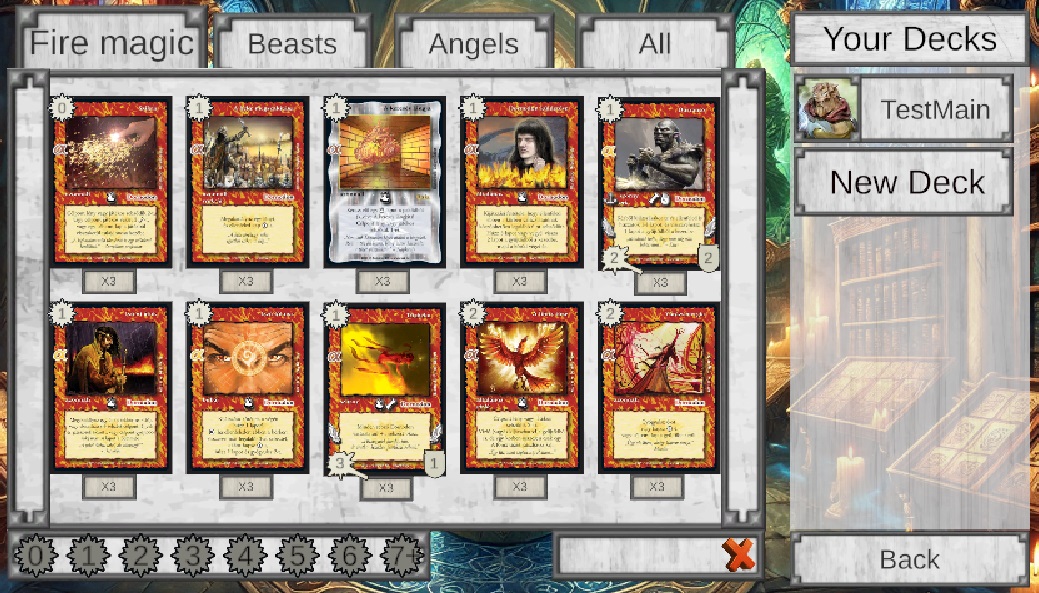
Az “All” fül az alapvető beállítás és ezen belül végigmehetünk az összes meglévő kártyán.

Lapozni úgy lehet az oldalakban, ha a rányomunk az oldalsó gombokra. A jobb gomb a következő oldalra léptett a bal pedig vissza. A kártyák a “Mana költségük” alapján vannak alapértelmezetten sorba rendezve növekvően.

A többi szűrő page az angels, beast és fire magicra

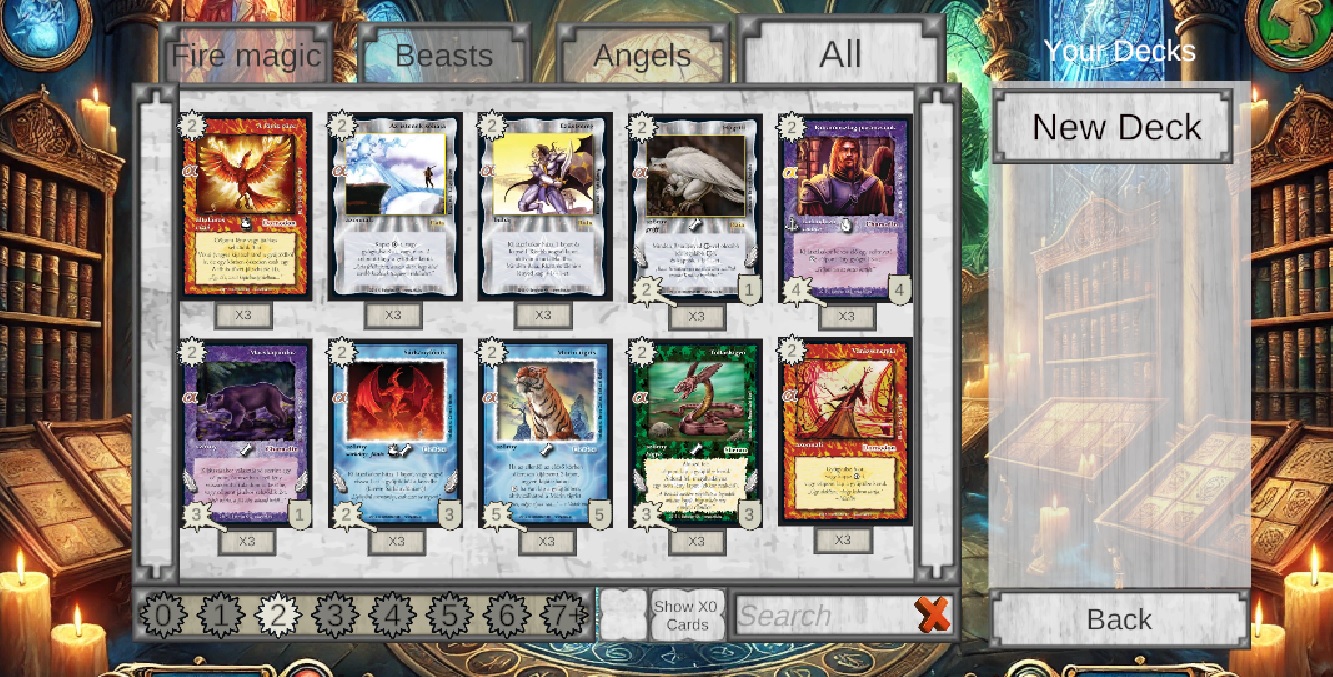


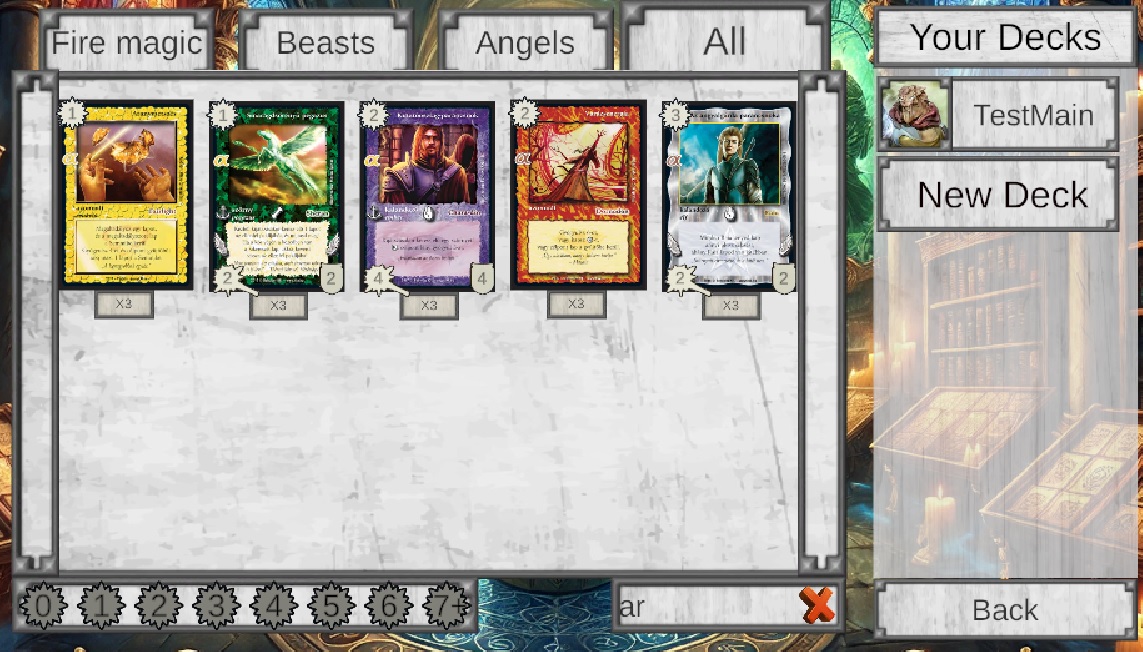




Ezek a szűrések jelenleg csak azt a célt szolgálják hogy a játékos megnézhesse a 3 “alfa paklit” ami valójában kezdő pakliknak minősülnek és elérhetőek a valódi játékban is

További keresési funkció a bal also sarokban levő számok. Ha rákatittunk egy számra akkor az azzal megegyező költségű lapra tudunk szűrni. Ha pedig ki akarjuk kapcsolni ez a szürést akkor az éppen aktív szűrő gombra kell megint rányomnunk



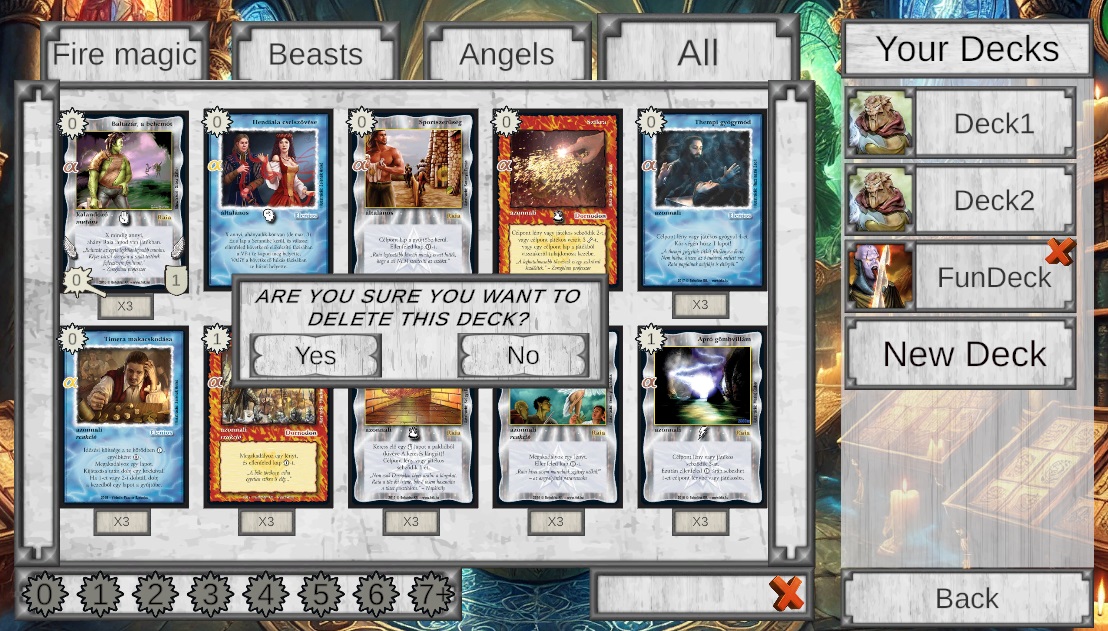
Ezen felül a jobb alsó sarokban lévő “Search”-tel szűrhetünk lapok nevére és ez a funkció működik együtt a költségre való szűréssel is. Egyszerűen csak beírjuk valamilyen szót vagy betűt a keresőbe és Enter megnyomásá után csak azokat a lapokat látjuk amire rákerestünk.



A következő és talán legfontosabb funkció maga a pakinak a készítése amit a “New deck” gombbal tudunk elérni. Legelőször is kiválaszhatjuk mi legyen a paklihoz használt “Karakterünk” ami majd meg fog jelleni amikor elkezdünk játszani.

Majd ha ez kész akkor átvisz minket egy másik de kollekcióhohz hasonló oldalra ahol a kártyák rákattuntásával berakhatjuk őket a deckbe.

 Minden kártyából 3 áll rendelkezésünkre és 1 kártyába egy pakliba maximum 3 ugyanolyan. Elmenthetünk félkész paklikat is azonban csak olyan paklival használhatunk majd játékra amiben leglalább 40 kártya van. Ha a pakliból ki akarunk venni egy kártyát akkor a paklin belül csak rá kell kattintanunk és indikálva van mennyi van még a kártyából a pakliban és a kollekciónkban. Természetesen ezen felül nevet is adhatunk a saját paklinknak mielőtt elmentjük a “Done” gombbal de később is szerkeszhetjük hiszen mentés után már meg fog jelenni a kollekciónkban, viszont ki is törölhetjuk ha rányomunk a piros X gombra a pakli jobb felső sarkában



## Multiplayer gomb

Egy játszma indulásakor véletlenszerűen dől el melyik játékos kezd. Mindkét játékos 4 lappal kezdi a játékot viszont az akinél a kezdés volt az első körében nem húz lapot.

A játékmenetben a kör elején az éppen körön lévő játékos húz 1 lapot és kap annyi Manát ahanyadik köre van. A kézben lévő lapok úgy játszhatjuk ki hogy rákattintuk és elkezdjük őket húzni. Természetesen ez csak akkor lehetséges ha van elég Manák hogy ki is játszuk azt a lapot. Azt hogy a lap kijátszható-e onnan is láthatjuk hogy van-e zöld körvonala.

TODO kép higlighról

Ha egy lapot behúzunk a kijátszási zónába ami egy kicsit különbözik varázslatok és lények között, és ott elengedjük akkor lények esetén az asztalra kerülnek a kijátszó játékos oldalán a varázslatok pedig a gyűjtőbe kerülnek miután kifejtették hatásukat. Ha esetleg valamelyik varázslanál célpontot kell választani egyszerűen csak lenyomva tarve a a bal egérgombot ráhúzzuk arra amit célpont szánunk s amikor a célkereszt rajta vana célponton elengedjük

TODO célzás kép

Miután kijátszottunk egy lapot az akció joga átmegy az ellenfélhez és ő is dönthet hogy kijátszik lapot vagy passzol. Miután a két játékos egymás után passzol akkor a kör véget ér és új kör kezdődik

TODO aktív lények támadás

A játék célján belül elég az ellenfél életpontját 0-ra csökkentenünk hogy megnyerjük a játékot ha ez megtörtének megkapjuk a “You won” kiírást és lehetőség van új játék kezdésére

A “Quit” opció egyszerűen csak bezárja a programot viszont a program bezárása után is eltárolja a játékos áltál készített esetleges paklikat

# **Fejlesztői dokumentáció:**

Ebben a fejezetben az alkalmazás működése kódszinten kerül bemutatásra, diagramokkal kiegészítve. Az alkalmazás a Unity játékmotor segítségével készült, amely C# programozási nyelvet használ. Ennek megfelelően a dokumentáció a C# nyelv sajátosságaira fog támaszkodni, amelyek közül a legfontosabbakat itt soroljuk fel:

**Fontosabb C# és Unity sajátosságok:**

1. Osztály alapú objektumorientált programozás  
   A Unity-ben minden komponens és script osztályként van definiálva, gyakran a MonoBehaviour osztályból származtatva. De mivel a projektünk használ Networking
2. Életciklus metódusok  
   A Unity speciális függvényei, mint például:
   * Start() – az objektum indulásakor fut le.
   * Update() – minden frame-ben meghívódik.
3. Komponens alapú architektúra  
   Az objektumok viselkedését komponensek (script-ek, physics, renderer stb.) hozzáadásával lehet alakítani.
4. Unity Editor használata  
   A szerkesztő lehetővé teszi a jelenetek, objektumok, prefabek és assetek vizuális szerkesztését, valamint az Inspectorban változók könnyű szerkesztését.
5. SerializeField és public változók  
   A public vagy [SerializeField] attribútummal ellátott mezők szerkeszthetők az Inspectorban, ami segíti a tervezők és fejlesztők közötti munkamegosztást.
6. Event-driven programozás  
   Unity támogatja események használatát (UnityEvent, delegate, event) különböző rendszerek, például UI kezelés során.
7. Prefab rendszer  
   Újrahasználható objektumok sablonjai, amelyeket jelenetekben többször is fel lehet használni.
8. Coroutine-ok  
   A Unity lehetőséget nyújt időalapú folyamatok egyszerű kezelésére coroutine-ok segítségével (StartCoroutine, IEnumerator).
9. Physics használata  
   Beépített fizikai motor támogatja az ütközést, mozgást – BoxCollider, Raycasting
10. Scene-kezelés  
    Több jelenet is kezelhető a SceneManager osztály segítségével, lehetőség van például jelenetváltásra futás közben.

A Unity editor egy elég egyedi fejlesztői környezet ahol az objektumok elhelyezése és a C# scriptek megírása is egyaránt nagyon fontos és ennek a a kettőnek a kapcsolatából születik meg a végleges projekt

## **Fordítás és futtatás:**

A project a hivatalos Unity Hub editor-al készült és Editor version 2022.3.16f1-ön lett legnagyobb részben fejlesztve. A forráskódot az editor segítségével lehet megnyitni és innen is lehet kiexportálni saját programmá.

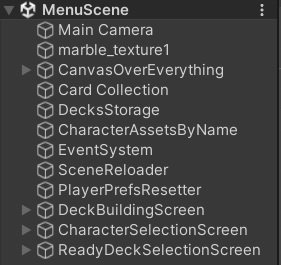
## **Architektúra:**

Ebben a fejezetben lesz az alkalmazás működése kódszinten elmagyarázva, diagramokkal kiegészítve. Az alkalmazás a Unity keretrendszerben íródott, mely C#-ot használ. Így a dokumentáció ennek a nyelvnek a sajátosságait fogja használni, mint az objektum orientált programozás és az eventek. Ezen felül a Unityben a legfontosabb elemek az objektumjaink és a hozzájuk tartozó scriptek.

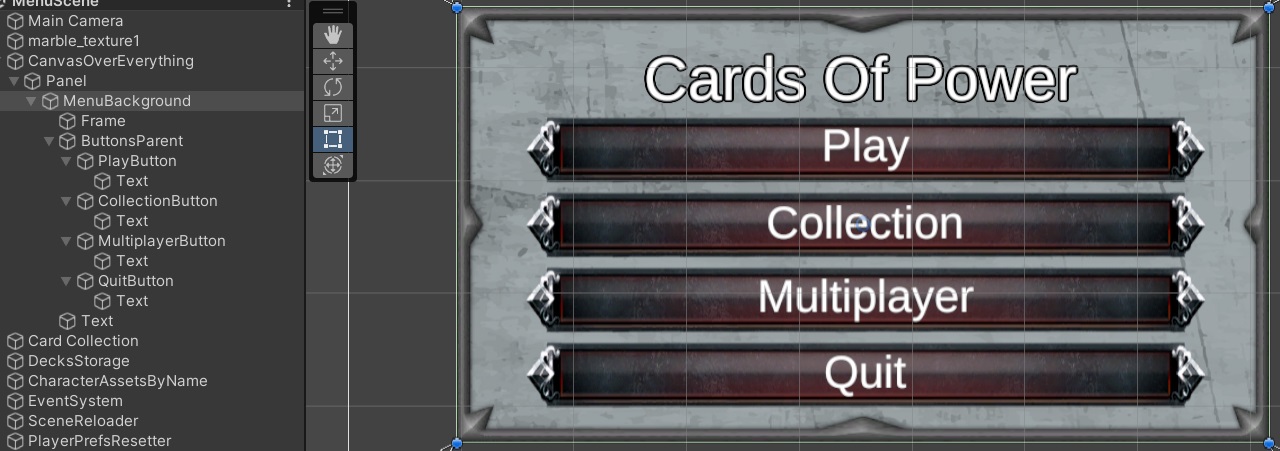
A Unity belül “Scene”-ek be helyezzük az objektumainkat melyeket egy “Main camera”-val figyelünk meg.

Ebben a projekben kettő “Scene” található, az egyik a Battlescene melyben maga a kártyajátéknak a játékmenete történek, a másik pedig a “Menuscene” amin keresztül elérhetjük a játékot és a pakliszerkesztési menüpontot

## **MenuScene**

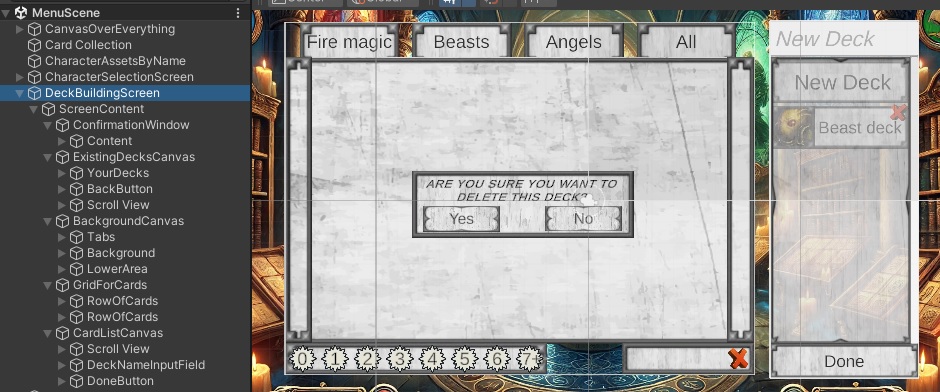


A “Menuscene” tarmazza a 4 fő oldalt amit látunk. Az első a “CanvasOverEverything”-ben található ami tarmazza a fő menüoldat és annak minde gombját és vizuális objektumait

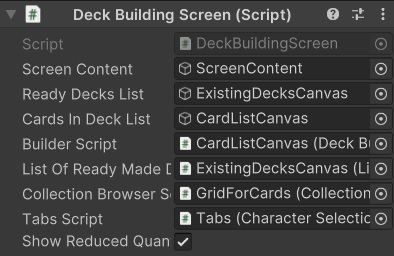


A MenuBackground és Frame lényegében csak a vizuális reprezentációt segítik elő ami fontos azok a “PlayButton”, “CollectionButton”, “MultiplayerButton” és “QuitButton”. Eznek a gomboknak mind be van állítva hogy egy kicsit másabb képet mutassanak amikor a cursor rajta van-e a gombon vagy meg van-e nyomva. S természetesen hogy ne csak dísznek legyenek ott mindegyikhez hozzá vannak úgynevezett “C# script”ek adva amik akkor aktiváklódnak ha rányomunk ezekre a gombokra. Nagyon egyszerűen működnek mindegyiknek az a feladata hogy egy másik screen-t láthassunk, a quit gomb kivételével mert az csak bezárja a programot

A gomb amire a “Collection” szó van írva elvisz minket a “Deckbuilding screen-re, vagyis pontosabban elrejti a menu oldalunk és aktiválja a “Deckbuilding screen”-t ami alapvetően ki lenne kapcsolja amikor elindul a program.



A “DeckBuildingScreen” talán a legbonyolultabb objektum a “Menuscene”-en belül mert nagyon sok kis eleme van és ezek között ugrálunk igazából.



A fő objektumhoz hozzá van csatolva egy “Decl Building Screen Script” ami lehetővé teszi hogy egyszerűen váltogathassunk oldalok között amik mind része a hierarchiának és létherhozza:

public static DeckBuildingScreen Instance;

Nevű publikus és statikus adattagot, ez azért lesz releváns később mert más script-ek használják a DeckBuildingScreenben eltárol egyes adatokat és ezért hivatkozásként léteznie kell ennek. Persze nem véletlenül publikus és staikus hiszen ahoz hogy könnyen elérhetőnek kell Lennie publikusan és a staikus kulcsszó nem engedi hogy ebből az adattagból több legyen mint egy miközbe fut a program így elkerüljük hogy az adatok szét legyenek szórva.

A 2 fő oldal amik között váltogatunk itt az a “ExistingDeckCanvas”, “CardListCanvas”. A másik 2 a BackGroundCanvas és GridForCard ugyanis szinte mindig ugyanaz persze néha ezek is változnak.

Minden esetre kezdjük a BackGroundCanvassal.

Todo kép ide a bgcanvsaról loweareaval

Ez az amit valamilyen formában itt mindig látni fogunk. Valójávan ez, összerakva a GridForCards objektummal segíti elő hogy ki tudjuk listázni a kártyáinkat.

A “Tabs” elemhez hozzá van kötve a “Character Selection Script” ami segít abban hogy szűrjünk a kártyákon azzal hogy rányomunk az egyes kis fülekre, illetve meghatározza melyik szűrési fül lesz az aktív amikor megnyílik az oldal. Ezen kívül taralmazza a lapozó gombokat is az ablak két oldalán amik a “PreviousButton” és a “NextButton”. Ezek lehetőve teszik hogy lapozzunk a lapok között hiszen egyszerre csak 10 lapot jelenítunk meg

A “GridForCards”-hoz a “Collection Browser Script” van hozzákötve.



Ugynevezett “Prefab”-eket használnuk itt amik igazából előre elkészített objektum minták. Ebben az esetben a “SpellCardMenu” és a “CreatureCardMenu”-ról vannak ilyenjeink amik meghatározzák hogy az egy kártya hogy jelenjen meg a kollekciónkban. Így később csak meg kell mondanunk hogy melyik “CardAsset” a kártya aminek az információit meg kell jeleníteni. Persze a “GridForCards” felelős azért is hogy a kártyáink hogyan helyezkedjenek el, ebben az esetben két sort adtam egy oldalnak és minden sorban 5 darab jelenik meg. A többi script és objektum ami hozzá van kötve még ehhez a scriptnek referenciák szempontjából vannak ott hogy egyszerűen manipulálhassuk mely kártyákat kell kirajzolnunk.



Itt egy példa hogy mit láthatunk ha először megnyitjuk a kollekciónk. Minden kártya egyenlő távolságra van elhelyve egymástól és a Prefaboknak köszönhetően látjuk mennyi áll rendelkésünkre egy adott kártyából.

Lowerarea:

Szorosan összefüg a “GridForCards” objektummal hiszen a “Collection Browser”-hez hozzákötött utolsó 3 script a referenciák hogy az itt kiválasztott szűrések továbbítódjanak és megváltozzon a Grid.

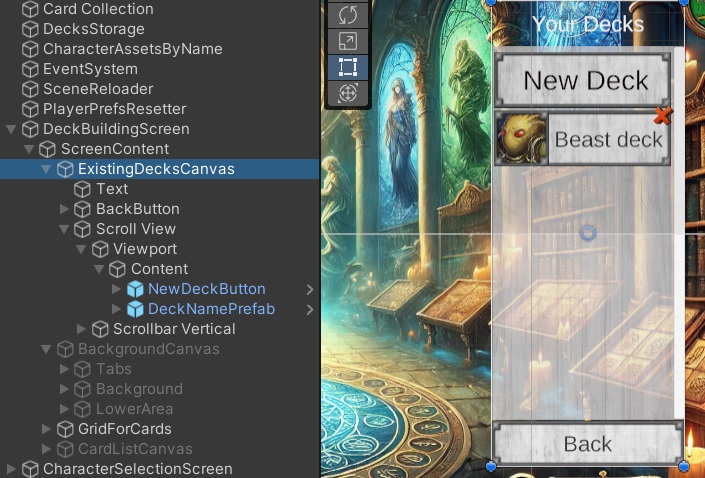


A “LowerArea”-nak is különlegessége hogy folymatosan látjuk amikor lapokat nézünk és állandó szűrési lehetőségeket biztosít. A “ManaFilter” lehetővé teszi hogy csak bizonyos költséggel azonosakat lássunk és a hozzá kötött script ugyanezzel a névvel felelős azért hogy egyértelműen lássuk melyik van éppen aktiválvi. Amikor megnyitjuk a kollekciót akkor mindegyik ikon még fehér ezzel jelezve hogy 0-tól végtelenig minden költségű kártyát látunk azonban ha az egyikre rákattintunk akkor az az ikon beszürkül ezzel jelezve hogy aktív a szűrés.

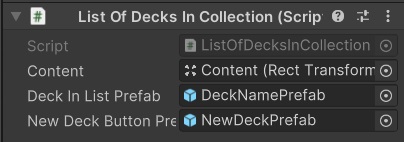
A “KeywordInputField” egy olyan “Input Field” objektum amivel konkrét kártyára kereshetünk rá mivel jelenlegi állapotában szűr a beírt string-re az összes kártyanév között. Egy továbbfejlesztési opció bizonyos kulcsszavak hozzáadása a kártyákhoz mint például a speciális képességük és akkor azokra is tudunk szűrni. Így ha például csak azokat a kártyákat akarjuk amiknek a szövegében a célpont szó szerepel akkor csak azokat lássuk

Végül a “Toggle” arra hivatott hogy tudjunk váltani kártyák között amik megvannak és amiket még nem szereztünk meg viszont még ebben az implementálásban minden kártya a rendelkezésünkre áll ami hozzá lett adva az alkalmazáshoz így jelenleg nincs funkciója.

## **ExistingDecksCanvas:**

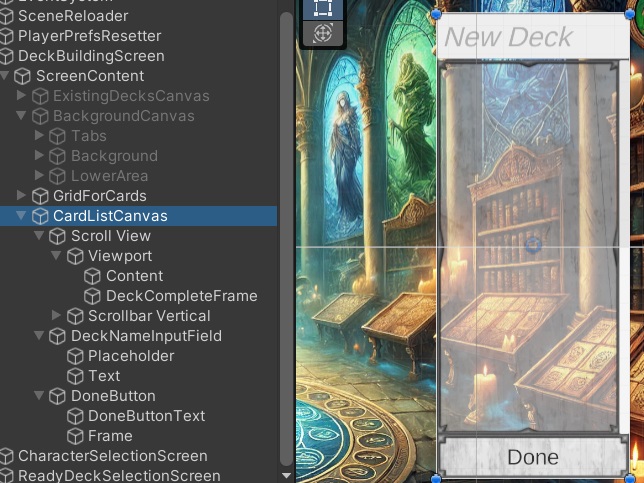


Ez a kompónens felelős azért hogy láthassuk a már elkészült paklikat és hogy lehetőségünk legyen újjakat csinálni. Hozzá van kötve a “ListOfDecksInCollection” nevű script hogy betölthessük ezeket a paklikat és szintén itt is Prefabek segítségével tudunk könnyen olyan objektumot létrehozni amit ebbe a Scroll Viewba beleillik és csak bizonyos információkat kell átadnunk kódból

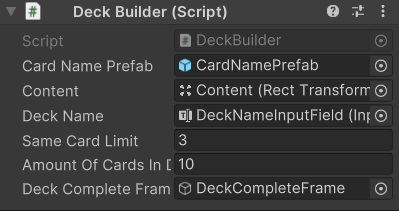


A “New Deck” gomb addig elérhető amíg nincs 9 elmentett paklik az adatbázisban és addig mindig megjelenik. Végül pedig persze szerepel egy “Back” gomb is ami vissza visz minket a fő menuoldalra.

## **CardListCanvas:**

****

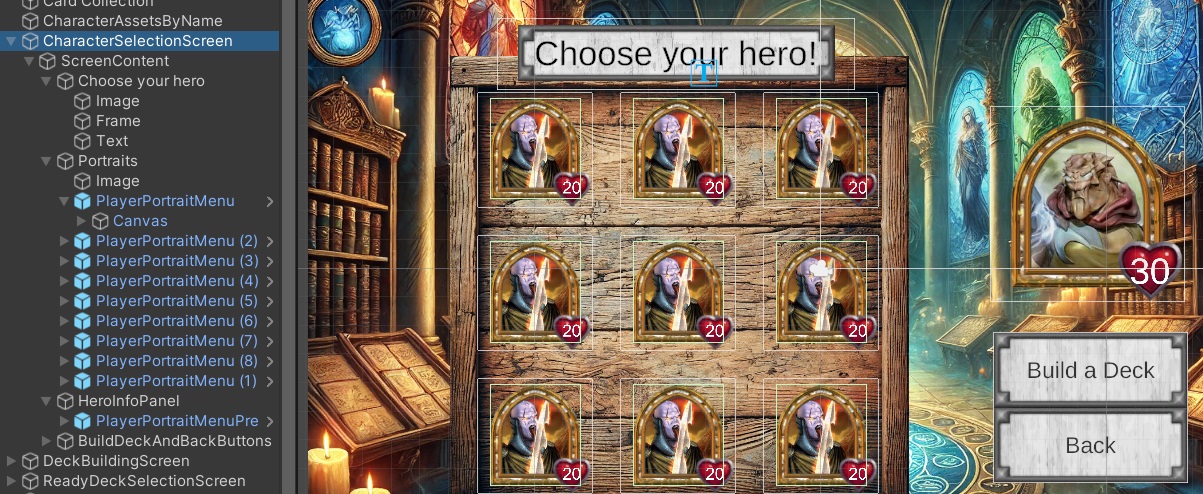
Nagyon hasonló kinézetileg a “ExistingDecksCanvas”-hoz viszont teljesen más a funkciója. Ugyanigy ez fog megjelenni helyette ha a “New Deck” opciót választjuk. Itt készíthetjük el a paklinkat azáltal hogy belerakunk és kiveszünk lapokat



Ehhez az objektumhoz a “Deck Builder” script van hozzácsatolva hogy könnyen állíthassuk

## **CharacterSelectionScreen:**

A “CharacterSelectionScreen” akkor jelenik meg miután a játkos kiválasztja hogy egy új paklit akar csinálni és itt választhat 9 különböző hősből ami majd reprezentálni fogja a karaterét miután a játékel kezdődik.

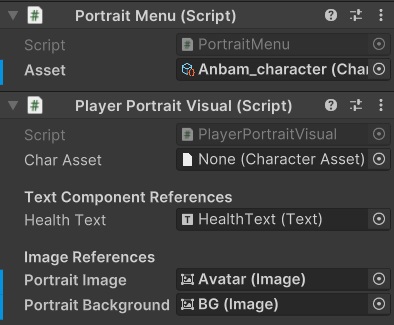


Az Editorban egyformában néznek ki a válaszható karakterek viszont a “PlayerPorraitMenu” prefabek mind el vannak látva egy különböző “Asset”-el amiben kinyerek a megfelelő informéciókat mind példul a képeket.

### HeroInfoPanel:



### Portraits:



**Egyebek a CharacterSelectionScreen-en belül:**

Megemlítendő még a „Choose your hero” és „BuildDeckAndBackButtons” objuktumok is de nincs fontos hozzájuk kötődő kód. A „Choose your hero” egyszerűen csak egy tábla ami egy jobb vizuális környezet érdekében lett létrehozva szintúgy mint az „Image” a „Portraits”-on belül hogy egyértelmű legyen mit lehet ezen az oldalon csinálni és mik az opciók.

„BuildDeckAndBackButtons” pedig ahogy a neve is mutatja két gombot tartalmaz, az egyik továbbvisz a pakli szerkesztéshez, a másik pedig visszaléptet minket. A kódok amiket meghívnak pedig más helyeken vannak elhelyezve mint „CharacterSelectionScreen”, „DeckBuildingScreen” vagy „HeroInfoPanel

## **ReadyCheckSelectionScreen:**

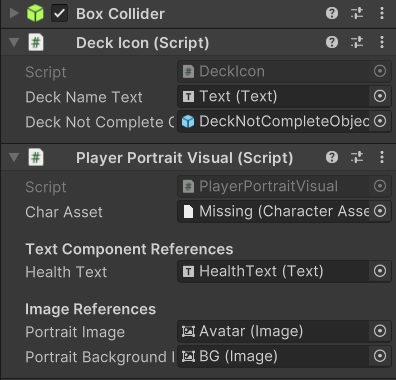


### HeroInfoPanel:

Ugyanaz az objektum a „CharacterSelectionScreen” reprezentálja hogy melyik opció én éppen szelektálva,

### Decks/Deck Icon:

Hasonló mint a „CharacterSelectionScreen”-nek a „Portraits” objektuma viszont itt a már létező paklik vannak feltűntetve és vizuálisan a „Deck Icon” mutatja a paklink nevét, hogy melyik hőst választottuk és hogy a pakli megfelelően van-e összerakva a játékhoz. Ha nem megfelelő, például túl kevés lap van benne, egy nagy X ikon lesz a karakter felett és nem is tudjuk kijelölni. Ahogy a „PlayerPortraitVisual” így ennek is van egy „Box Collider”-je ami az objektumhoz tartozik. Ez azért fontos mert ez mutatja meg az ikonnak a „hitbox”-át ami azt jelenti hogy a hozzá illő kód akkor aktiválódik hogy ezen a területen belül kattintunk. Ha kijelöltünk egy paklit és utánna a „Play” gombbal mehetünk tovább ami azonnal továbbadja a játékosnak a szükséges pakli információkat.



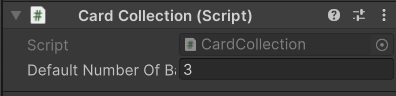
### PlayAndBackButtons:

Itt is hasonlóan mint a „CharacterSelectionScreen”-nél a két gombbal segítségével navigálhatunk tovább vagy vissza és a gomb megnyomása előidéznek képernyő váltásokat. A „PlayButton” átrank mintket a „GameScene”-re amíg a „BackButton” visszarak minket a főoldalra a „MenuScene”-en belül

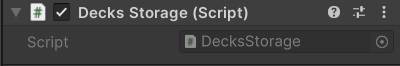
## **Egyebek:**

Ezek azok a C# unity kódok amik a “MonoBehaviour” osztályból vannak leszármaztatva ami azt jelenti hogy használhatóak ahogy metódusok bennük mint az Awake(), Start() és Resources.LoadAll<T>() amik rendkvül fontosak egy ilyen projektben. Ha a “MonoBehaviour” nincs egy objektumhoz akkor az ilyen metódusok nem fognak működni. Ezek az objektumok olyan logikákat tartalmaznak amik betöltenek vagy tárolnak valamit. A betöltés itt lehet valódi asset is projektból vagy csak segítenek hogy tudjunk könnyen nézeteket változtatni.

### CardCollection:

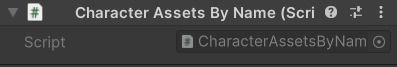


### DecksStorage:



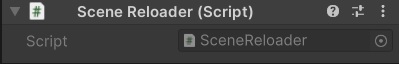
Tárolja a játékos összes meglévő pakliját és azoknak az összes fontos attribútumját mint a paklinév, melyik hőst választotta a játékos a paklihoz és természetesen a magákat a kártyákat is. Ezt úgy éri el hogy tartalmazza a DeckInfo nevű osztályt is amiben valójában tárolódnak ezek az attribútumok és ezeket tölti be a PlayerPrefs osztályból ami a Unitynak egy beépített osztálya ami képes egyszerű játékadatok tárolására és egyfajta perzisztenciaként meghívható a program két különböző elindítása után. Ez az amivel elérjük hogy a paklik ne vesszenek el amikor bezárjuk a játékot

### CharacterAssetByName:



A program elindatása után betölti közvetlenül az Awake() metódussal betölti az összes meglévő hőst a mappákból aztán azokat tárolja is. Természetesen ez azért fontos hogy utánna a public CharacterAsset GetCharacterByName(string name) a hős neve alapján megtalálhassuk és felhasználhassuk az adott karakter tulajdonságait.

### SceneReloader:



Ez a fő jelentkezelő a “MenuScene”-en belül és mivel ez is “MonoBehaviour” ezért könnyen megívható “UI” eseményekből mint például gombok és hozzáférhet olyan fontos osztályokhoz és metódusokhoz mint a SceneManager vagy a Application.Quit();

public void ReloadScene() pedig szükséges ahhoz hogyha egy új játékot akarunk elkezdeni egy lefutott után mert ennek a logikája biztosítja hogy minden előző játékből general kártya ID-je törlődjön és újak jöhessenek létre

### PlayerPrefsResetter:



Egy nagyon hasznos kis objektum, inkább csak a kezelhetőség kedvéért létezik mivel tartalmazza a bool DeleteAllFromPlayerPrefs amit az editoron kereszül még indítás előtt be tudunk állítani hogy igaz vagy hamis legyen. Amikor a program elindul ennek az objektumnak a Awake() parancsa meghívódik, amikor létrejön a példánya és még a Start() parancs előtt vagy kitörli a játékos előző preferenciáit(vagyis a pakliait). Így láthatjuk milyen amikor elsőnek elindul a program egy új eszközön anélkül hogy manuálisan kitörölnénk mindent

### Main camera, EventSystem

Ezek olyan objektumok amiket a Unity Editor maga general és nélkülök nem működni a program semmilyen szinten. A “Main Camera” felelős azért a nézetért amit a játékos éppen lát, ha nincs kamera nincs mit látni ezért is része minden alap unity projektnek ez a kamera objektum. Szinte egész végig egyhelyben marad és mi csak manipuláljuk éppen mit mutatunk neki. Az “EventSystem” pedig automatikusan generálódik amikor egy “UI” elemet adunk hozzá a projekthez, például egy gombot. E nélkül az aktív “UI” elemek nem is tudnak reagálni semmilyen interakcióra

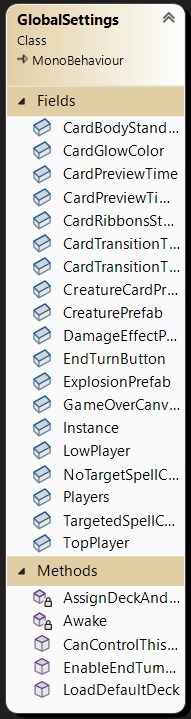
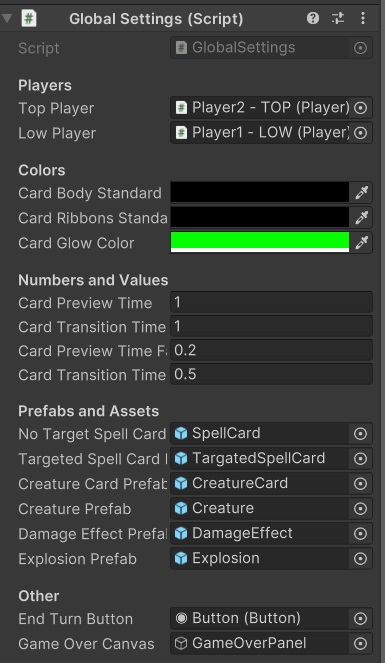
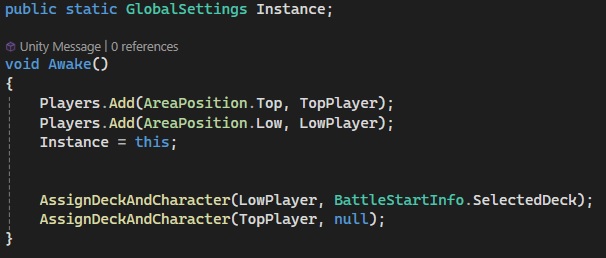
## **BattleScene**

****

A “BattleScene” a másik fő része az alkalmazásnak ami tartalmazza minden játékmenethez szükséges vizuális és kodó megoldást.

A legtöbb vizuális objektumnak a viselkedéset egy hozzá csatólt C# script befolyásolja az alapján hogy a játékos éppen mimlyen akciót hajt végre.

A “Main Camera” az a látószög amit a játékos lát és ezen belül van az összes vizuális kompónens. GlobalSettings:

A “GlobalSettings” egy univerzális főkapcsoló ahol a legfőbb információkat és adattagokat tartarmazza. Ide adjuk meg referenciának többek között a képernyő felső és alsó részén lévő játékusankhoz kötödő fő objektumokat. Továbbra a “Number and Values opciók határozzák milyen gyorsaságúak legyenek az animáció például lapkijtászásnál vagy lap húzásnál. Mivel magához az objektumhoz a kódnak csak egy másolatás kötjük ezért azok az értékek lesznek ténylegsen használva a programban amiket a Unity Editoron belül állítunk be. Az akarjuk hogy ebből a sciptből csak 1 legyen az egész programban ezért itt is egy statikus és publikus “Instance”-t hozunk létre ami az Awake() függvény miatt azonnal létrehozódik amikor a Globalsettings valahol meghívódik és lehetővé teszi hogy Singleton osztály legyen

A “Prefab and Asset” mezők pedig tarmazzák azokat az előre elkészített objektumokat amiknek a másolatait használni fogjuk több kód futtatásában is. Ezek főleg a vizuális reprezentáció a lény és varázslat kártyáknak amik vagy a játékos kezében vannak vagy lények esetében az asztalon is lehetnek. A “DamageEffect” és “Explosion” tagok is szintén Prefabek és ezek mint megtalálhatóak a projek “Assest/Prefabs” mappájában

## **SceneReloader:**

Ez az objektum és a hozzá kapcsolt script ugyanezzel a névvel felelős azért hogy esetleges más “Scene”-eket dinamikusan betölthessünk és ha elégtük a játék vége állapotot és újra akarunk játszani akkor reset-el minden eddigi információt hogy betölthessünk újakat

## **Visual:**

Az egyik legfontosabb része a “Scene”-nek hiszen itt található minden látható elem ami a “Scene”-en belül van beleértve az asztalt,paklit, gyűjtőt, semmit, karakterportrék kör vége gomb stb. és rengeteg szabad szemmel nem látható objektum amiknek a jelentősége hogy tudjuk a kártyákat és portrékat hova kell mozgatni a képernyőn belül:



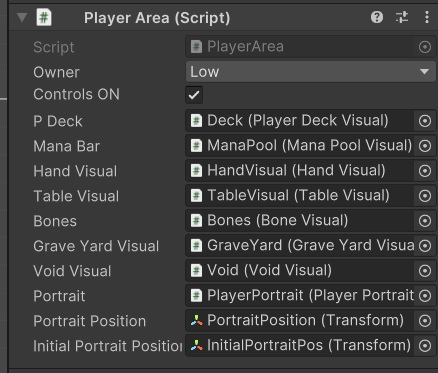
A “boards” objektum szimplán az éppen manuálisan beállított hátteret tartalmazza amin a játékot játszuk

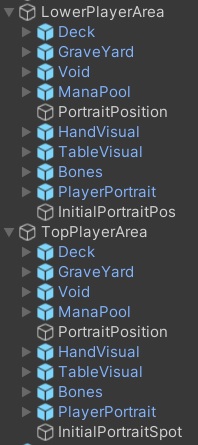
Olyan elemek mint a “PlayPreviewSpot” és “DrawPreviewSpot” szimplán pozícióknak szolgálnak arra hogyha egy lapot kijátszunk annak a nagyított képe hol jelenjen meg és ha húzunk egy lapot annak a nagytott képet hol lássuk. Ezen kívül az “OtherDraw” olyan kártyának a pozíciója lenne amit nem a pakliból kapunk hanem valahonnan máshonnan

A “DamageEffectTest” és “HealEffectTest” tesztelés szempjából relevánsak a sebzést és gyógyulást mutate vizuális kompónensek demonstrálásához

A “CanvasOverEverything” pedig tartalmazza a játék vége oldalt amihez vágunk ha egy játéknak vége. Nevéből is ítélve a célja hogy mindenképpen lássuk ha meghívjuk.

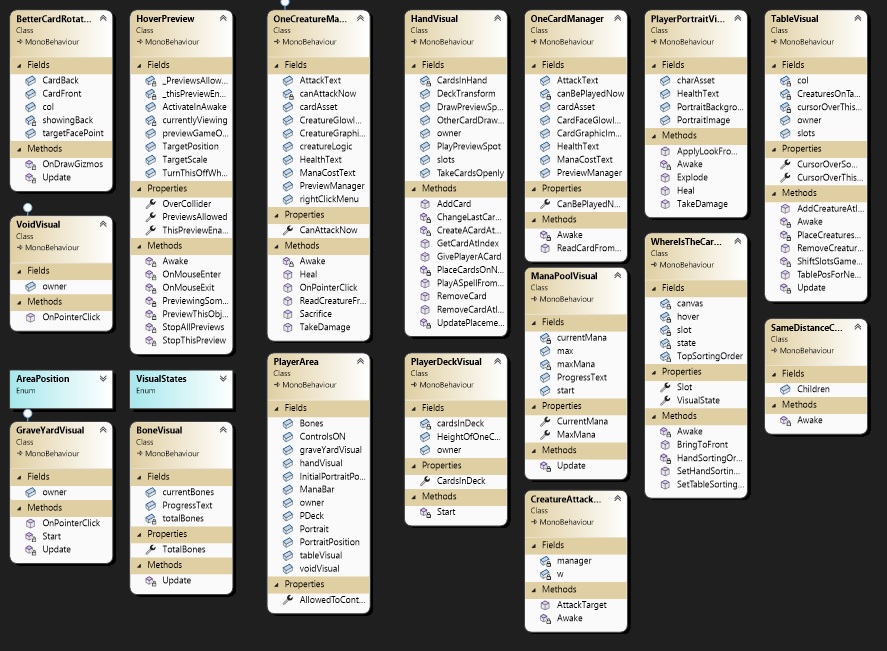
## **TopPlayerArea és LowerPlayerArea:**

A tartalmuk szinte megegyző ennek a két objektumnak hiszen ezek reprezentálják az egyes játékosaknak azokat a tagokat amik az ő “oldalukhoz” tartoznak.



A Unity editoron belül így vannak láthatóak. a “LowerPlayerArea” és a “TopPlayerArea”-hoz is szintén a “Player Area” fő script van csatlakoztatva amiben beállíthatjuk hogy a felső vagy alsó játékosnak feleljen-e meg és így kódon belül is tudjuk manupulálni az adattagjait szimplán azzal hogy “PlayerArea” az AreaPosition.Low vagy Top-ról van-e szó hiszen az Owner adattag egy “AreaPositon” enum.

Az összes többi elem melynek nagyészeri Prefabek valamelyik vizuális kompónenst tartalmazzák és irányítják



Itt jól látható hogy szinte minden objektumhoz van valamilyen háttéri kód ami segít őket változtatni.

A “PlayerArea” scriptnek ezeknek a adjuk meg referenciaként és innen tudjuk változtatni őket. Például a “ManaPoolVisual” és “BonesVisual” scriptek felelősek a mana és a csontok növekedésélért és csökkenéséért. Mindekét játékos rendelkezik “ManaPool” és “Bones” objektumokkal amik mutatják mennyi az aktuálos és a Mana esetében mennyi a maximális ami ebből lehet egy játékosnak. Ezekhez vannak hozzakapcsolva az előbb megnevezett scriptek és a referenciájuk pedig a “PlayerArea”-ba vannak behúzva. Így ha például a felső “PlayerArea”-ának változtatjuk a “ManaPoolVisual”-jában lévő adattagját a megfelelő függvénnyel akkor a felső játékosnak a manajá fog változni.

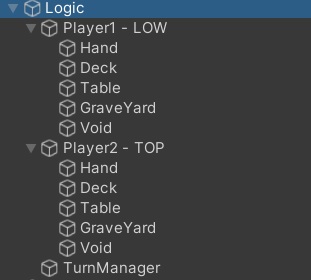
Hasonló logikával működik a “Graveyard” és a “Void” is amik tárolják hogy mely lapok vannak a felső és alsó játékos gyűjtőjében és semmijében. Ezen kívül a “PlayerDeckVisual” segítségével láthatjuk hogy körülbelül mennyi lap van még egy játékos paklijában illetve fontos pozicióknak számítanak hogy húzás során honnan induljon a lap felhúzása.

A “PlayerPortraitVisual” mutatja hogy melyik hőssel játszik és a játékos és mennyi az aktuális életereje. Ha ez bármikor 0-ra esik az aktiválja a “Die” függvényt annak a játékosnak és a játék végetér.

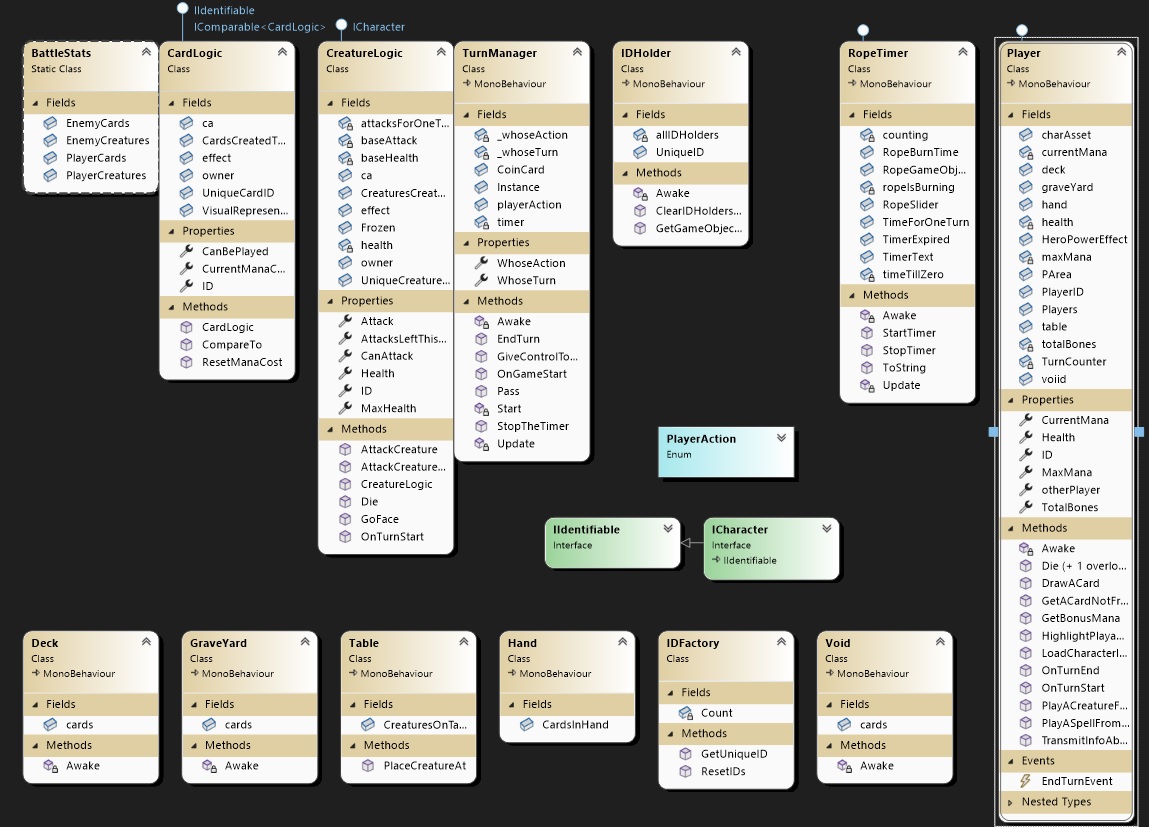
Végül pedig a “HandVisual” és “TableVisual” felelős a kézben és asztalon lévő kártyákért. A “HandVisual”-nak a slotjaiba kerülnek be a kártyák laphúzás után és amikor egy lap kijátszódik eltűnik a slotból. Plusz az “SameDistanceChildren” segítségével mindig egyenlő eloszlásban lesznek a lapok a kézben attól függetlenül hogy 2 vagy 10 lap van-e benne. A “TableVisual” pedig az a terület ahova a kijátszok lények kerülnek. Ez is használja a “SameDistanceChildren” scriptet hogy a lények megtartság az egyenlő távolságot. Tartozik hozzá egy úgynevezett “ColliderBox” ami meghatározza mely területen belül kell elengednünk a lény lapot hogy sikeres legyen a kijátszása. Ezen felül a “TableVisual” felelős egy lény eltávolításáról is az asztalról ha meghalna.

A scripteken kívül két pontot is megadunk névileg a “PortraitPosition” és “InitialPosition”-t. Az elsőt hogy azért hogy hol fog a karakter portrénk elhelyezkedni véglegesen egy játékosnak, a másodikat pedig hogy honnan induljon amikor a játék elején lévő lejátszódik az animáció ami elhelyezi a játékosok hőseit.

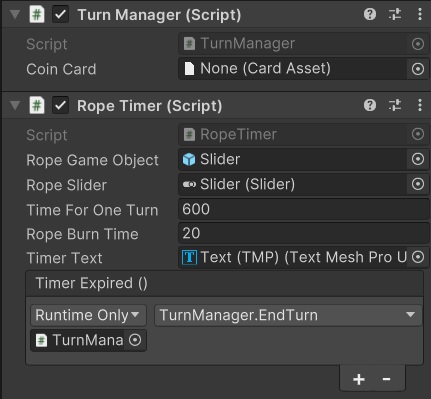
## **Logic:**



A másik nagy része a “BattleScenek” ami a logikai kompónensekkel foglalkozik és és adatbázis szinten ide vannak ténlyegesen eltárolva mi van a játékos kezében, asztalán, gyűjtőjében, semmijében és paklijában



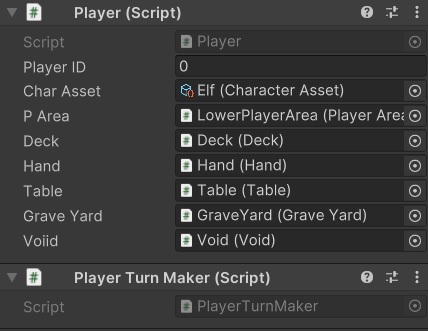
## **TurnManager:**



Szintén a logikához tartozik a “TurnManager” vagyis a köröket szabályozó algoritmus. Ennek segítségével tudjuk megmondani milyen hosszú legyen egy kör mivel a “RopeTimer” script is hozzá van csatlakoztatva. Illetve itt állítjuk be magában a “TurnManager scriptben melyik játékos melyik játékos kezd, kinek az akciója van, mennyi kártyát húzúnk a játék elején stb”. Singletonkként viselkedik mivel csak egyre van szükségünk a projekben és itt döntjük el kinek a köre vagy akciója van és így váltogathatjuk hogy melyik játékosnak mutatjuk mely lapokat tudja kijátszani.

## **Player:**

A “Player” egy robusztus osztály ami felelős szinte minden játékoshoz kötődő akcióhoz és vezérli a kártyák kijátszását. A játék elején 2 ilyen játékos osztályt hozunk létre és kapcsoljuk össze a Top és LowPlayer objektumokkal a projekben. Saját ID-t kapnak és bármilyen változás történek a játékon vagy valamin ami a játékoshoz kapcsolódik azt ezen az osztályon keresztül történik



A “Char Asset” határozza meg melyik karakter fogja reprezentálni a játékost.

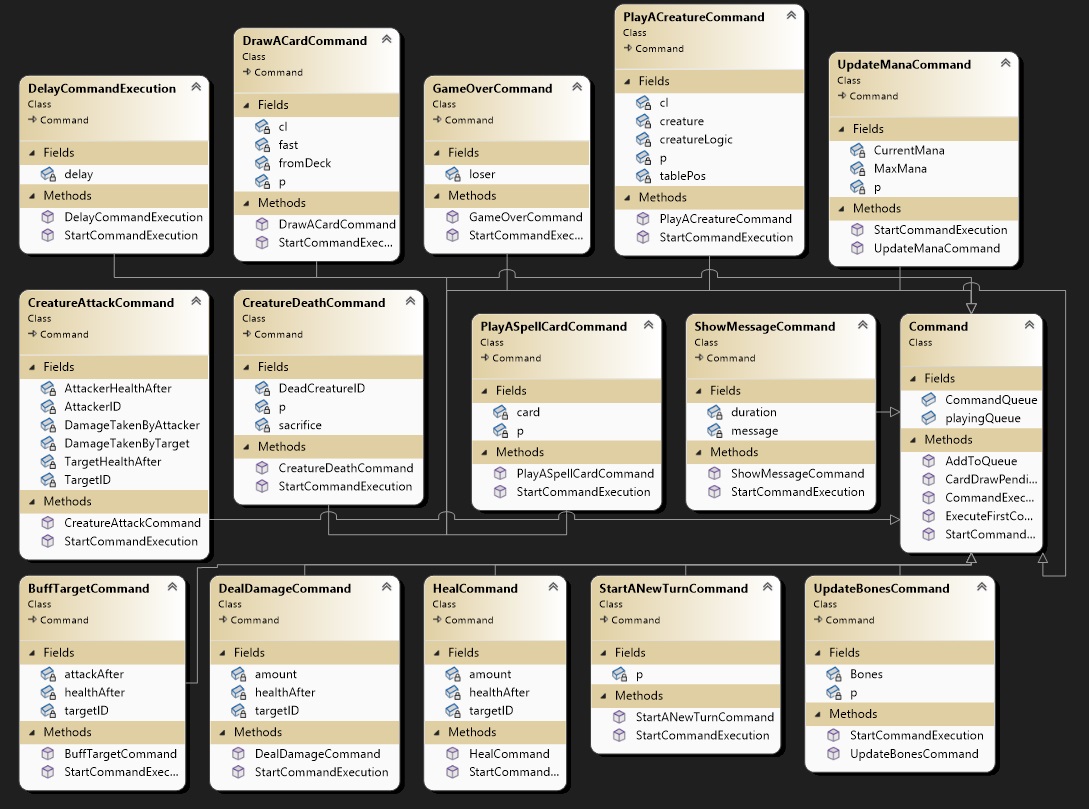
“Deck”-től “Voiid”-ig az adattagok egyszerűbb scriptek és nagyrészt tárolóknak funkcionálnak hogy tudjuk hogy a játékosok kártyái hova kerülnek

## **PlayerTurnMaker**

Ennek a feladata mindössze annyi hogy a kör elején kiírja “ShowMessageCommand” segítségével hogy hogy a saját vagy az ellenfél köre van.

Scriptek visual nélkül:

## **Commands:**

A “Scripts” mappán belül találhatók a “Command” scriptek nagy része. Ezek már olyan kódokat tartalmaznak amik nem feltétlenül egy objecktumra vannak rátéve hanem a háttérből vezérelnek eseményeket és azoknak a szinkronizálását. A célja, hogy biztosítsa, hogy a játékos által kiadott parancsok (pl. kártya húzása, akciók végrehajtása) ne ütközzenek egymással.

A diagramból jól látható hogy mindegyik “Command” az alap osztályból származtatható le. A Command osztály a parancsok kezeléséért felelős alap osztály. A parancsok végrehajtásához használt sor (Queue) és annak a logikája van benne implementálva.

* **CommandQueue statikus változó:** A parancsokat sorban tartja, hogy biztosítsa a megfelelő végrehajtást.
* **playingQueue statikus változó:** Meghatározza, hogy a sor éppen végrehajtás alatt van-e, hogy a további parancsokat csak akkor hajtsa végre, amikor az előzőek befejeződtek.

#### StartCommandExecution():

#### Ez egy virtuális metódus, amelyet a konkrét parancsok (pl. DrawACardCommand) implementálnak. Az adott parancs végrehajtásának megkezdésére szolgál. Mivel alapértelmezetten nem csinál semmit, minden egyes parancsnak felül kell írnia ezt a metódust, hogy meghatározza, mit kell csinálnia, amikor a parancsot végrehajtják.

#### CardDrawPending():

Ez a statikus metódus azt ellenőrzi, hogy van-e a parancs sorban egy kártya húzása (DrawACardCommand), amely még nem lett végrehajtva. Ez segít abban, hogy a játék ne próbáljon végrehajtani olyan parancsokat, amelyek kártya húzását előzik meg, ezzel biztosítva a helyes sorrendet.

#### CommandExecutionComplete():

A metódus akkor hívódik meg, amikor az aktuális parancs végrehajtása befejeződött. Ez megvizsgálja, hogy van-e még várakozó parancs, és ha van, elindítja azt. Ezzel a metódussal zárul le minden egyes parancs végrehajtása, és biztosítja, hogy ne induljanak el új parancsok, amíg az előzőek nem fejeződtek be. Emellett a játéknak jelzi, hogy vissza kell térnie a játékmenethez (pl. lehetőséget ad a játékosnak a következő akcióra).

#### AddToQueue():

Ez a metódus hozzáadja az aktuális parancsot a sorhoz, majd, ha nincs már folyamatban lévő parancs végrehajtása, elindítja az első parancs végrehajtását. A parancsok hozzáadása a sorhoz és az első parancs végrehajtása biztosítja a megfelelő sorrendet és a játékmenet folyamatosságát.

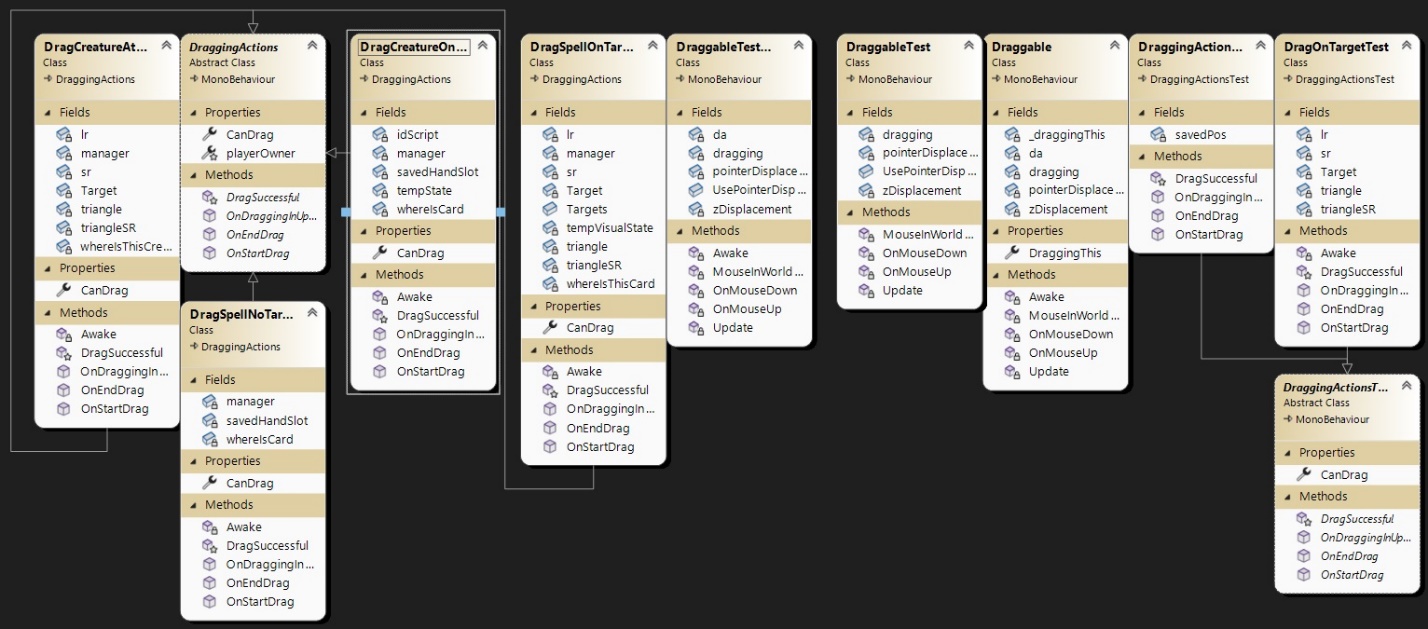
#### ExecuteFirstCommandFromQueue():

Ez a statikus metódus a sorból eltávolítja az első parancsot, és elindítja annak végrehajtását. Ez felelős azért, hogy mindig csak egyetlen parancs fusson egyszerre, és hogy a sorban várakozó parancsok fokozatosan végrehajtásra kerüljenek.

### ****Kapcsolódó osztályok és kiegészítők:****

DrawACardCommand**:**Az ilyen típusú parancsok az alosztályok, amelyek a Command osztályt öröklik és implementálják a StartCommandExecution() metódust, amely végrehajtja a konkrét logikát (pl. kártya húzása).

## **Dragging:**

Ezek az osztályok felelősek a hogy a kártyákat a kezünkből tudjuk mozgatni az asztalra vagy hogy célozzunk lények támadásaival vagy egyéb kártyák képességivel

## **DOTween:**

A **DOTween** egy nagy teljesítményű, könnyen használható tweening motor Unity-hez. A tweening olyan folyamatot jelent, amely során egy objektum tulajdonságait animáljuk az idő múlásával, például pozíciót, forgatást, méretet, színt, vagy akár egyedi tulajdonságokat. A DOTween leegyszerűsíti a sima és természetes animációk létrehozását Unity-ben, egy könnyen kezelhető API-t kínálva a fejlesztők számára.

Pozíció animálás: Egy objektum sima mozgatása egyik pontról a másikra.

Forgatás: Egy objektum forgatásának vagy méretének finom animálása.

## **Mirror:**

A **Mirror** egy nyílt forráskódú, Unityhez készült hálózati könyvtár, amely lehetővé teszi multiplayer játékok fejlesztését. A Unity korábbi beépített hálózati rendszere (UNet) megszűnésével a fejlesztői közösség igényeinek kielégítésére jött létre, és mára a legnépszerűbb harmadik féltől származó megoldás a Unity hálózati fejlesztéseihez. A Mirror célja egy **e**gyszerű, mégis nagy teljesítményű és megbízható hálózati API biztosítása, amely különösen jól alkalmazható gyors fejlesztési ciklusú projektekhez, például indie játékokhoz vagy prototípusokhoz.

Mirror lapvető működési elve:

A Mirror a **kliens-szerver architektúrán** alapszik. Egy játékos (általában a host) futtatja a szervert, míg a többi játékos ehhez kapcsolódik kliensként. Ez a struktúra jól illeszkedik a kártyajátékokhoz, ahol központi logika szükséges a játékszabályok érvényesítéséhez és az adatok szinkronizálásához.

A rendszer **NetworkBehaviour** osztályokon alapul, amelyek lehetővé teszik, hogy az objektumok hálózati eseményekre reagáljanak. A legfontosabb elemek:

* **NetworkManager** – kezeli a kapcsolódást, kliens- és szerverindítást.
* **NetworkIdentity** – minden hálózaton szinkronizált objektum egyedi azonosítója.
* **Commands** – kliensről szerverre küldött függvényhívások.
* **ClientRpc** – szerverről kliensre küldött függvényhívások.
* **SyncVar** – automatikusan szinkronizált változók a szerver és a kliensek között.

A szakdolgozat keretében készített kártyajáték egy **online multiplayer** játék, ahol a játékosok egy közös játéktérben, valós időben vesznek részt a mérkőzésekben. A Mirror ebben a kontextusban elsősorban az alábbi funkciókat látta el.

## **NetWorking**

## **Tesztelés**

A tesztelés célja a kártyajáték alapfunkcióinak megbízható működésének ellenőrzése volt többjátékos környezetben, különböző hálózati feltételek és felhasználói interakciók mellett.

**Manuális tesztelés** – fejlesztői tesztelés különböző helyzetekben.

**Black-box tesztelés** – a felhasználói oldalról működés vizsgálata.

**White-box tesztelés** –kódrészlet megfigyelése, hogy megfelelően működik-e.

|  |  |  |
| --- | --- | --- |
| Két játékos csatlakozása | A host elindítja a játékot, a kliens csatlakozik. | Sikeres |
| Kártya kijátszása |  |  |
| Körváltás működése |  |  |
| Üres pakli kezelése |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## **Összegzés**

Összegezve, a kártyajáték célja egy többjátékos, hálózati környezetben működő, Unity-alapú alkalmazás elkészítése volt, amely bemutatja egy valós kártyajáték digitális adaptációjának alapjait. A játék jelenlegi állapotában egy stabil, működőképes prototípus, amely sikeresen demonstrálja az online többjátékos működés, a kártyakezelés, valamint a körökre osztott játékmenet alapvető elemeit. Fontos azonban kiemelni, hogy ez a verzió **csupán a játék koncepcionális alapjait valósítja meg**, és még nem tartalmazza az eredeti, teljes játékrendszer minden részletét. Több kártyatípus, játékszabály, valamint a mélyebb stratégiai elemek még nem kerültek implementálásra.

### További lehetőségek:

A jelenlegi állapot ellenére a projekt jelentős bővítési lehetőségeket rejt magában, különösen az alábbi területeken:

* **Játékszabályok és játékmenet fejlesztése**: A későbbiekben lehetőség van a játék eredeti szabályrendszerének pontosabb implementálására, beleértve a különböző játékszakaszokat, speciális szabályokat és győzelmi feltételeket.
* **További és különleges kártyák bevezetése**: A jövőben a paklik bővítése, egyedi és ritka kártyák hozzáadása, illetve ezek képességeinek kezelése színesebbé és összetettebbé teheti a játékot.
* **Fejlettebb hálózati infrastruktúra**: A jelenlegi peer-to-peer (host-client) megközelítés mellett lehetőség nyílik dedikált szerverek használatára, amellyel stabilabb és nagyobb távolságokat áthidaló kapcsolatok biztosíthatók. Ez különösen akkor válik fontossá, ha a játék több mint 2-4 fő részvételét célozza meg.
* **Felhasználói élmény fejlesztése**: Grafikai visszajelzések, hanghatások és animációk bevezetése a jövőben hozzájárulhat a játékélmény gazdagításához.
* **Platformfüggetlenség**: A Mirror és a Unity lehetőséget ad arra, hogy a játék más platformokon is elérhetővé váljon, például Android vagy WebGL formájában.

A kártyajáték fejlesztése során sikerült egy olyan technikai alapot létrehozni, amelyre a későbbiekben egy komplexebb, a valós kártyajátékhoz hűbb digitális élmény építhető.

**Irodalmi jegyzék:**

1. Unity Documentation: <https://mirror-networking.gitbook.io/docs>
2. DOTween Documentation: <https://dotween.demigiant.com/documentation.php>
3. Unity Mirror Documentation: <https://mirror-networking.gitbook.io/docs>