Izrada kartaške videoigre u programskom alatu Unity

Kerovec, Ivan

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:638043

Rights / Prava: Attribution-NonCommercial 3.0 Unported/Imenovanje-Nekomercijalno 3.0

Download date / Datum preuzimanja: **2024-07-10**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Ivan Kerovec

IZRADA KARTAŠKE VIDEOIGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Kerovec

Matični broj: 17060/07–R

Studij: Informacijski i poslovni sustavi

IZRADA KARTAŠKE VIDEOIGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor :

Doc. dr. sc. Mladen Konecki

Varaždin, rujan 2023.

Ivan Kerovec

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovome radu prikazuje se postupak razvoja kartaške videoigre koristeći programski alat za izradu videoigara, Unity. Na početku rada obrađuje se alat Unity kao razvojno okruženje specijalizirano za izradu videoigara, te njegove najbitnije komponente koje su korištene u razvoju praktičnog dijela. Nadalje, obraditi će se tema kartaških igara, usredotočujući se na povijest i utjecaj takozvanih "kolekcionarnih kartaških igara" (eng. *Collectible Card Game* - kraće CCG), te njihov utjecaj na suvremene kartaške videoigre. Na kraju rada obrađuje se proces izrade kartaške videoigre koja služi kao praktični dio ovog završnog rada.

Ključne riječi: Unity; Videoigre; Karte; Razvoj videoigre; Mehanike;

Sadržaj

1. Uvod

Glavna tema ovog završnog rada je izrada kartaške videoigre koristeći alate za razvoj softvera, prvenstveno Unity razvojni okvir koji je specijaliziran za izradu raznih vrsta videoigara, Visual Studio za pisanje izvornog koda, te pomoćni alati poput Figme i Aseprite, koji su uglavnom korišteni za izradu grafičkih elemenata videoigre.

Žanr kolekcionarskih kartaških igara oduvijek je bio popularan oblik zabave. Mnogima je samo skupljanje karata veliki dio apela, bilo to zbog želje za izradom raznih špilova od kojih svaki ima posebnu strategiju za pobjedu, ili jednostavno zato jer vole skupljati rijetke karte. Konstruiranje špila također je važan dio kartaške igre, jer najčešća razlika između pobjede i gubitka je kvaliteta samog špila komplementirana s iskustvom i vještinom igrača.

Moja glavna motivacija za biranje ove teme je moje zanimanje za kartaške igre općenito. Žanr kartaških igara je vrlo fleksibilan, pogotovo u digitalnom obliku. Kao najveće inspiracije u obliku videoigara istaknuo bi videoigru *"Hearthstone"* od kompanije *"Blizzard Entertainment"*, koja mi je dugo vrijeme bila jedna od najdražih videoigara, te videoigra *"Slay the Spire"* od kompanije *"Mega Crit Games"*, u koju sam uložio gotovo više od 600 sati.

Razvoj kartaških videoigara, te posebice njihovih pravila, pruži dobru priliku za korištenje objektno orijentiranog i komponentnog oblika programiranja. Time je i programski alat Unity također vrlo dobra opcija, uzimajući u obzir da se za skripte koristi C# programski jezik, te okvir je uglavnom temeljen na komponente i njihove međusobne veze. Vjerujem da će mi ovaj projekt dati vrijedno iskustvo u izgradnji međusobno ovisnih objektno orijentiranih sustava, te dublje proširiti moje znanje sa C# programskim jezikom.

2. Metode i tehnike rada

Pri izradi praktičnog dijela ovog završnog rada temeljito se koristi programski alat Unity, zajedno s dodatnim alatima za obradu skripti i grafike. Za Obradu skripti i programskog koda korišten je Visual Studio 2022 Community Edition. Za izradu grafike i vizualnih materijala korišteni su online alat Figma, uglavnom za elemente korisničkog sučelja, te alat Aseprite, koji je korišten za izradu karata i njihovih vizualnih elemenata. Fontovi koji se koriste u praktičnom dijelu preuzeti su preko stranice Google fonts. Kao pomoć pri izradi videoigre uglavnom je korištena službena dokumentacija Unity alata.

3. Programski alat Unity

Unity je razvojni pogon razvijen od tvrtke Unity Technologies, namijenjen prvenstveno za izradu 2D i 3D videoigara, ali se također koristi i za razvoj interaktivnih simulacija ili animacija izvan industrije videoigara, poput inženjerstva ili arhitekture. Alat je napisan u programskom jeziku C++, a za pisanje programskih skripti koristi se programski jezik C#[1]. Unity se smatra kao iznimno fleksibilni alat u području razvoja videoigara, te je kao takav jedan od najpopularnijih opcija za početnike i hobiste, ali i iskusne programere. Sučelje alata smatra se intuitivno te lagano za naučiti i koristiti, s tim da korisniku pruži znatne mogućnosti u cijelokupnom procesu razvoja videoigara, uključujući gotovo sve što je korisniku potrebno da razvije videoigru. Osim jednostavnosti korištenja, jedan od većih faktora za uspjeh alata je izrazita podrška zajednice korisnika. Iako je službena dokumentacija opširna, postoji veliki broj članaka i videa od naprednijih korisnika koji redovite koriste alat u kojima dijele svoje znanje i tehnike rada. Time je baza znanja dovoljno velika da pokrije većinu potreba početničkih i naprednih korisnika. Cijenovno, alat je besplatan za koristiti za osobne i edukativne svrhe, te ukoliko korisnik ne zaradi više od 100.000 $ u zadnjih 12 mjeseci. Za takve korisnike postoje prilagođeni planovi ovisno o veličini kompanije.[2]

3.1. Kratka povijest

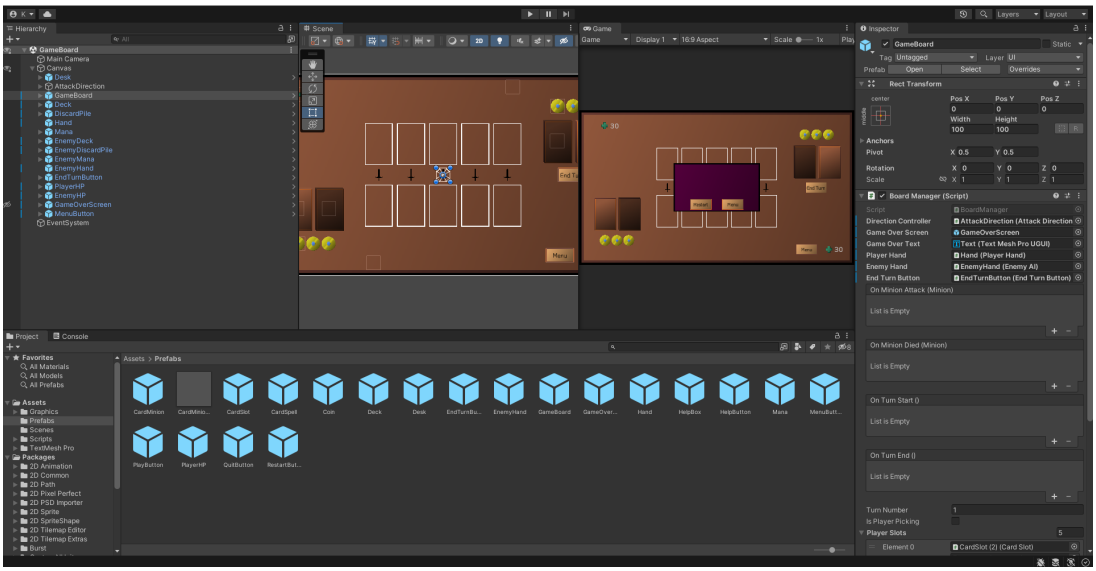
Unity, kao razvojni alat za razvoj videoigara, službeno je nastao 2005. godine na svjetskoj konferenciji programera (eng. *Worldwide Developers Conference*, kraće WWDC) održanu od strane kompanije Apple, te je predstavljen kao razvojni pogon za videoigre za platformu MAC OS X[1]. U nadolazećim godinama Unity je uveo podršku za druge platforme, poput iOS 2008. godine, te Android, osobna računala i konzole 2010. godine.[3] Danas je alat iznimno popularan izbor za manje razvojne timove ili samostalne korisnike, najviše u području mobilnih videoigara. Prema [4], 53% od 1000 najpopularnijih mobilnih videoigara na Apple aplikacijskom tržištu (eng. *Apple App Store*) i Google Play aplikacijskom tržištu napravljene su pomoću Unity alata. Neke od videoigara koje smatram da su vrijedne spomenuti, a napravljene su koristeći Unity, su: Beat Saber, Hollow Knight, Rimworld, Genshin Impact, Hearthstone te mnogo drugih.[5]

3.2. Značajke alata

U sljedećem odjeljku opisuju se glavne značajke i komponente razvojnog alata Unity koje se koriste pri razvoju, s većim naglaskom na komponente koje su korištene za razvoj videoigre iz praktičnog dijela.

3.2.1. Korisničko sučelje alata Unity

Prvi element s kojim će se svaki korisnik pri stvaranju novog projekta sresti je korisničko sučelje alata. Na prvi pogled, korisničko sučelje Unity-a može izgledati kompliciranо, pogotovo ako korisnik nije prethodno upoznat s razvojem videoigara. Na sljedećoj slici prikazuje se uobičajeni raspored elemenata koji predstavljaju Unity-jevo korisničko sučelje:

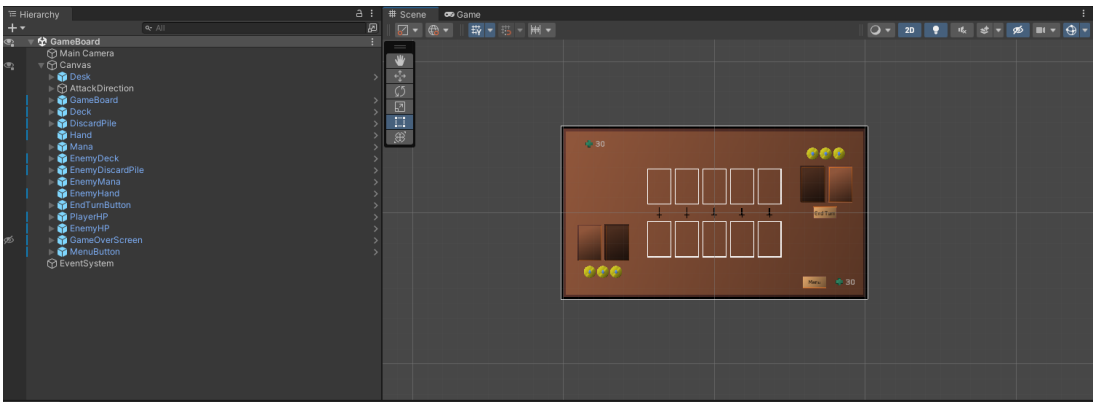


Slika 1: Korisničko sučelje alata Unity (Izvor: vlastita izrada)

Na gornjem dijelu sučelja nalazi se alatna traka koja, osim što omogućuje pregled korisničkog Unity profila i Unity servisa, također sadrži kontrole za pokretanje i zaustavljanje simulacije igre, te postavke za raspored sučelja. S gornje lijeve strane, a ispod alatne trake, nalazi se hijerarhijski prozor, koji prikazuje hijerarhiju objekta igre (eng. *GameObject*) u trenutno otvorenoj sceni. Objekti igre vezani su za trenutno otvorenu scenu, pa tako su i hijerarhijski prozor i prikaz scene međusobno povezani. Desno od hijerarhijskog prozora nalazi se prikaz scene, koji služi za prikaz, navigaciju i uređivanje objekta igre na trenutnoj sceni. Moguće je scenu prikazati u 2D ili 3D obliku. Osim toga, prikaz scene također sadrži alatnu traku za lakše navigiranje i uređivanje scene. Desno od prikaza scene nalazi se prikaz igre, koji prikazuje, to jest simulira izgled igre preko kamera koje su dodane u scenu. Također omogućuje testiranje i pokretanje trenutno otvorene scene. Desno od prikaza igre nalazi se prozor za pregled i uređivanje postavka (eng. *Inspector window*) trenutno obilježenog objekta igre. Raspored elemenata u inspektoru mijenja se ovisno o komponentama koje trenutno odabrani objekt igre sadrži. Na dnu sučelja nalazi se projektni prozor, koji prikazuje sve materijale koje su dodane i spremne za koristiti u projektu. Pod materijale misli se na slikovne datoteke, zvučne datoteke, skripte, teksture te svi ostali elementi koji se koriste u izradi videoigre. Osim pregleda materijala, postoji i pregled konzole koja prikazuje poruke, upozorenja te greške koje mogu nastati tijekom simulacije igre. Ispod projektnog prozora nalazi se statusna traka koja prikazuje notifikacije o stanju procesa alata, te postavke vezane za procese alata.[6]

3.2.2. Scene

Scene su glavno "platno" za sve elemente koji će se pojavljivati u videoigri. Kada korisnik napravi novu scenu, u njoj će se nalaziti kamera, koja služi kao prozor za prikaz (eng. *viewport*) scene, te ako je scena postavljena za 3D, također će se dodati usmjereno svijetlo. Ovisno o opsegu videoigre koja se izrađuje, projekt može sadržavati više scena po potrebi. Jednostavnija igra može se u potpunosti napraviti u jednoj sceni, dok kompleksnije igre mogu sadržavati više scena za svaku razinu, izbornike, itd[7]. Praktični dio ovog rada sadrži 3 scene. Glavna scena naziva se "GameBoard", to jest igraća ploča, i u njoj su postavljeni svi objekti bitni za igranje videoigre. Scena koja se prva otvori pri pokretanju videoigre zove se "Menu", te ona služi kao glavni izbornik. S glavnog izbornika igrač ulazi u "GameBoard" scenu kada pokreće igru, ili može ući u treću scenu pod nazivom "Help", koja ukratko objašnjava pravila i način igranja igre.

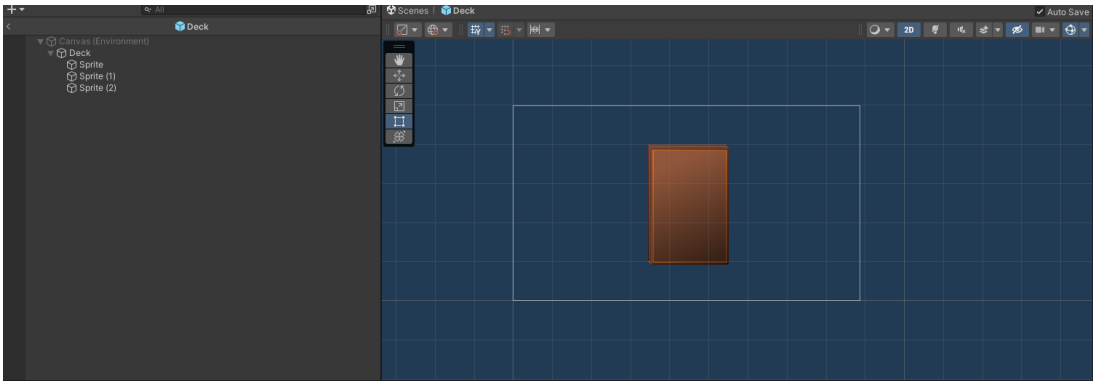


Slika 2: Glavna scena videoigre, zajedno s prikazom hijerarhije objekta igre u sceni (Izvor: vlastita izrada)

3.2.3. Objekti igre (eng. *GameObjects*)

Objekti igre su najbitniji dio Unity-a. Svi entiteti na sceni su objekti igre, bili oni likovi, zvukovi, čestice, korisničko sučelje ili nešto slično. Objekti igre sami od sebe nemaju posebno značenje, već postoje kako bi se nadogradili s komponentama. Komponente definiraju svrhu i izgled objekta igre ovisno o različitim kombinacijama istih na istom objektu. Objekti igre koji predstavljaju likove ili stvari na sceni vjerojatno će imati komponente za prikazivanje modela ili tekstura, dok će objekt igre zaslužan za nevidljivu logiku na sceni vjerojatno imati samo skriptu kao komponentu. Svaki objekt igre, neovisno o svrhi, ima transformacijsku komponentu, koja određuje njegovu poziciju i orijentaciju na sceni. Tu komponentu nije moguće ukinuti.[8]

Objekt igre moguće je pretvoriti u predložak, koji se zatim može ponovno upotrebljivati i referencirati preko skripti u kodu. Predložak objekta igre sprema se u projektni prozor (eng. *Assets*), te se može uređivati neovisno o trenutno otvorenim scenama. Time se ostvaruje efikasnost kod rada s velikim brojem istih objekata igre, jer kada se spreme promjene na glavnom predlošku, sve instance tog predloška se sinkroniziraju s predloškom.



Slika 3: Prikaz uređivanja predloška (Izvor: vlastita izrada)

3.2.4. Skriptabilni objekti (eng. *Scriptable Objects*)

Skriptabilni objekti su jedan od načina za spremanje podataka, posebice nepromjenjive podatke koji se više puta instanciraju u projektu. Skriptabilni objekt može se referencirati u predlošku koji se instancira, i time svaka njegova instanca ima pristup istim podacima. Samo jedna kopija referenciranog skriptabilnog objekta postoji u memoriji, što smanjuje nepotrebno kopiranje istih podataka te smanjuje korištenje memorije u videoigri. Podaci u skriptabilnom objektu mogu se mijenjati kod uređivanja, ali i tijekom vremena izvođenja videoigre.[9]

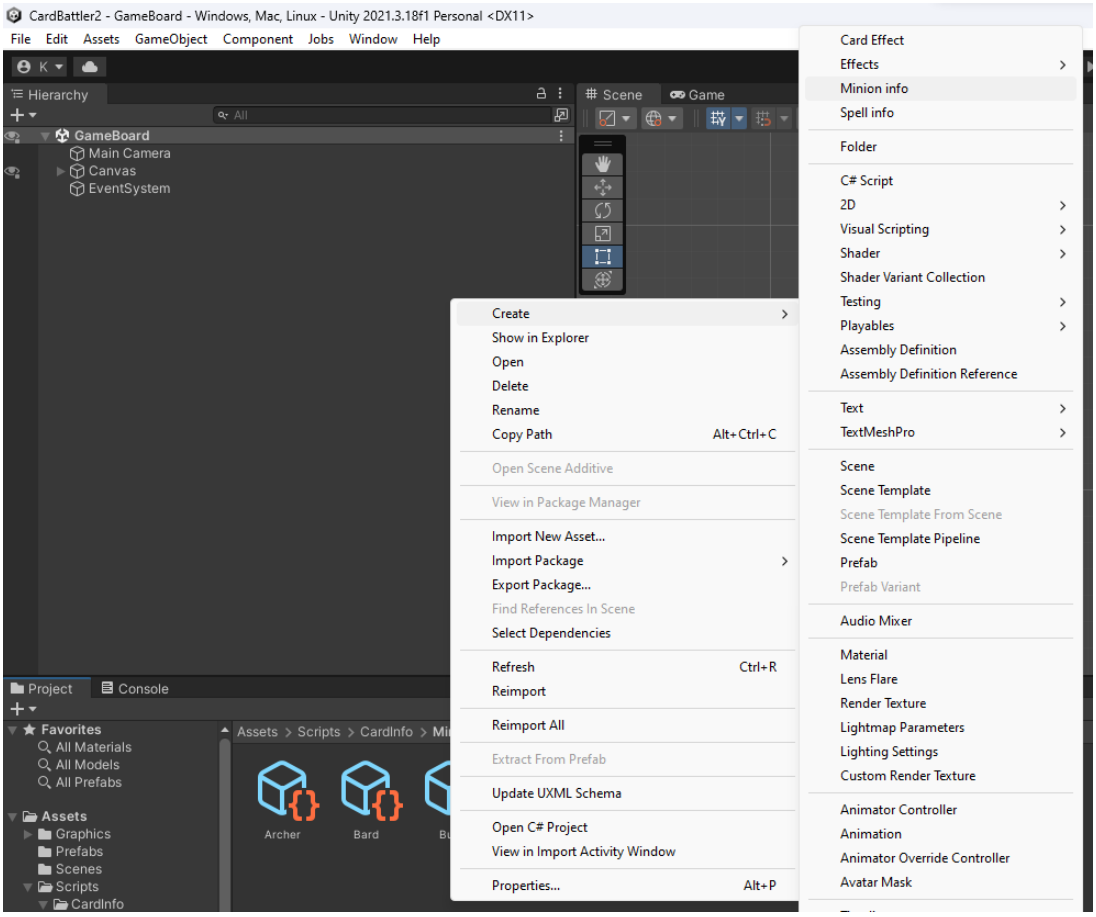
Iako su skriptabilni objekti zapravo skripte koje naslijeđuju ugrađenu Unity klasu "ScriptableObject", oni se ne mogu direktno dodati kao komponente na objekt igre, već se moraju referencirati preko skripte. Skriptabilni objekti postoje u projektnim datotekama, te ih je moguće instancirati ako se klasi skriptabilnog objekta doda atribut "CreateAssetMenu". Sljedeći isječak koda prikazuje takvu klasu skriptabilnog objekta, a koja se koristi u praktičnom dijelu ovog rada:

```
[CreateAssetMenu(menuName = "Minion info", fileName = "MinionInfo")]
public class MinionInfo : CardInfo
{

    [Header("Stats")]
    public int Attack;
    public int Health;
    public KeywordType Keyword;

    [Header("Effects")]
    public List<CardEffect> Effects;
}
```

Klasa "MinionInfo" naslijeđuje abstraktnu klasu "CardInfo", koja naslijeđuje ugrađenu klasu "ScriptableObject" i time predstavlja skriptabilni objekt, što znači da je klasu "MinionInfo" moguće instancirati više puta u projektnim datotekama. Instanciranje skriptabilnog objekta moguće je preko izbornika kada se pritisne desna tipka miša u prostoru projektnih datoteka[9], te je isti prikazan na sljedećoj slici:



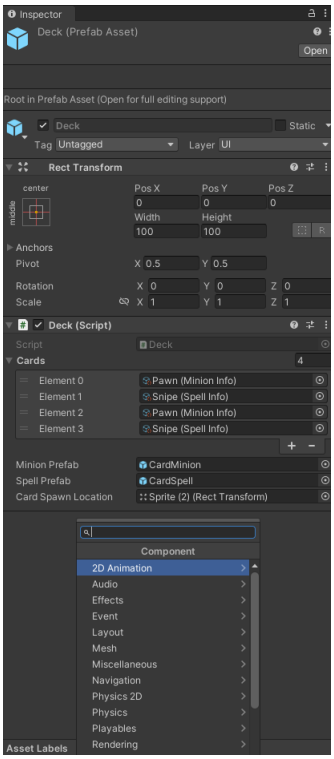
Slika 4: Izbornik za instanciranje skriptabilnih objekata(Izvor: vlastita izrada)

U praktičnom dijelu završnog rada skriptabilni objekti koriste se za spremanje osnovnih podataka o kartama, te spremanje njihovih posebnih efekata. Kolekcionarske kartaške igre vrlo često imaju veliku kolekciju karata, i u tom slučaju dobro je imati način da se karte mogu brzo i efikasno dodavati. Skriptabilni objekti su jedno rješenje za navedeni problem jer ukidaju potrebu da se koriste predlošci za karte, već je samo potrebno instancirati novi skriptabilni objekt za tip karte i popuniti potrebne podatke, poput imena, opisa i sliku. Efekti za karte također koriste skriptabilne objekte, ali na nešto različiti način. Više o implementaciji efekata za karte biti će prikazano u opisu praktičnog dijela.

3.2.5. Komponente

Komponente su funkcionalni dio svakog objekta igre. Svaka komponenta sadrži atribute i funkcionalnosti koje definiraju njihovu glavnu svrhu. Kao što je prethodno spomenuto, svaki objekt igre uvijek ima minimalno jednu komponentu, a to je transformacijska komponenta[10]. Komponente je moguće dodati preko inspektora objekta igre koristeći ugrađenu listu komponenta, ili je moguće napraviti svoju komponentu u obliku skripte.

Na prethodnoj slici prikazan je izbornik koji se pojavi kada korisnik pritisne gumb "Add component" u inspektoru, kada ima odabran objekt igre ili predložak. Također, na slici se vidio skripta pod nazivom "Deck" kao komponenta, zajedno s njezinim korisničko definiranim atri-



Slika 5: Dodavanje komponenata na objekt igre (Izvor: vlastita izrada)

butima. Unity smatra skripte koje korisnik izradi kao ugrađene komponente, te se kao takvi pojavljuju u inspektoru objekta igre. Uređivanje i podešavanje komponenti moguće je preko inspektora u načinu uređivanja, ili je moguće preko skripti na način da se komponente referenciraju u kodu i podešavaju prema određenoj logici[11].

Ovisno o kontekstu videoigre, najčešće nije potrebno vlastite komponente, osim skripti za upravljanje istima, jer su velika većina elementarnih i najčešće korištenih funkcionalnosti pokrivene s ugrađenim komponentama.

Kod izrade praktičnog dijela ovog rada, najviše sam koristio komponente vezane za korisničko sučelje, zajedno s vlastitim skriptama koje su upravljale njima te odrađivale logiku videoigre. Najbitnija komponenta u ovom radu je **canvas** komponenta, koja je detaljnije obrazložena u nastavku.

3.2.6. Canvas komponenta

Canvas komponenta je glavna komponenta za svu logiku i vizuale koji su vezani za korisničko sučelje videoigre. Objekt igre koji sadrži canvas komponentu na sceni smatra se "platno" scene, te svi elementi korisničkog sučelja dodaju se kao djeca tog objekta igre. U slučaju da korisnik doda element korisničkog sučelja na scenu bez postojećeg canvas objekta igre, Unity će ga automatski dodati, te postaviti novododani element kao dijete[12].

Objekti korisničkog sučelja razlikuju se od ostailh objekata igre u određenim stvarima. Oni, umjesto obvezne transformacijske komponente, sadrže komponentu pod nazivom "Rect Transform", koja uglavnom služi istu svrhu kao i transformacijska komponenta, ali za 2D ele-

mente korisničkog sučelja.[13] Osim toga, elementi korisničkog sučelja imaju različiti redoslijed iscrtavanja (eng. *Draw order*) od ostalih objekata. Redoslijed iscrtavanja elemenata korisničkog sučelja je hijerarhijski određeno, što znači da element koji je zadnje dijete canvas objekta biti će iscrtano na vrhu svih ostalih elementa korisničkog sučelja. Redoslijed elemenata u hijerarhiji može se programski mijenjati koristeći ugrađene metode transformacijske komponente korisničkog sučelja. Te metode su:[12]

- **SetAsFirstSibling()** - postavlja objekt kao prvo dijete svog roditelja.
- **SetAsLastSibling()** - postavlja objekt kao zadnje dijete svog roditelja.
- **SetSiblingIndex(n)** - postavlja objekt kao *n*-to dijete svog roditelja, gdje *n* predstavlja prirodni pozitivni broj proslijeđen kao parametar metodi.

4. Kolekcionarske kartaške igre

S obzirom na to da je žanr videoigre praktičnog dijela rada najsličniji žanru **kolekcionarskih kartaških igara** (eng. *Collectible Card Game* - kraće CCG), ovo poglavlje posvetit će se kolekcionarskim kartaškim igrama kao žanr igara. Obradit će se njihova povijest, objasniti njihove tipične skupove pravila, te nabrojat će se neke od popularnijih i utjecajnih igara koje su definirale žanr kakav je danas.

Kolekcionarske kartaške igre mogu se opisati kao vrsta zabavne aktivnosti u kojoj igrači skupljaju karte s raznim vrstama učinaka, sastavljaju špil koristeći veći broj karata koje imaju zajedničku sinergiju i/ili prate određenu strategiju, te koriste napravljeni špil u borbi protiv drugog igrača i njegovog špila[14].

Prema [14], kolekcionarske kartaške igre mogu se podijeliti na tri razine. Prva je kolekcijska razina. U kolekcijskoj razini, igrač traži i sakuplja one karte koje želi imati u kolekciji, bilo to kupnjom ili razmjenom od drugih igrača. Druga je konstrukcijska razina, u kojoj igrač, ovisno o njegovoj kolekciji, konstruira špil od ograničenog broja karata ovisno o pravilima igre, te koristi taj špil tijekom igranja. Zadnja razina je igrača razina, gdje igrač koristi jedan od svojih napravljenih špilova u igri protiv drugog igrača.

4.1. Povijest

Žanr kolekcionarskih kartaških igara prvi put se pojavio u obliku bejzbol karata prije gotovo više od 100 godina, s najranijim bilježenim pojavama u 1904. godini.[14] Osim bejzbola, razna duhanska poduzeća također su uveli karte kao oblik međusobnog natjecanja i izgradnje lojalnosti prema brendu[15][16].

Slika 6: Primjer karte koja se mogla pronaći u duhanskim proizvodima (Izvor: [16])

Iako je koncept skupljanja karata već dugo postojao, koncept s kojim se kolekcionarske kartaške igre danas najviše povezuju pojavio se tek 1993. godine, kada je izašao prvi set karata za igru *Magic: The Gathering*, koja je čvrsto definirala današnji oblik i pravila da-

našnjih kolekcionarskih kartaških igara[14]. Popularnost igre naglo je rasla s vremenom, toliko da od 1996. godine već su postojali turniri s velikim ulogom nagrada za pobjednike. Prema procjeni izdavača igre, popularnost u brojevima dosegnula je oko 6 milijuna globalnih igrača od 2009. godine[15]. Danas je *Magic: The Gathering* još uvijek jedna od najpopularnijih kartaških igara općenito, izdavajući karte u fizičkom, ali i virtualnom obliku, od kojih se neke prodaju u vrijednosti od 2 milijuna američkih dolara[17].

Slika 7: Prodaja jedinstvene *Magic: The Gathering* karte za vrijednost od 2 milijuna američkih dolara (Izvor: [17])

4.2. Virtualne kolekcionarske kartaške igre

Kolekcionarske kartaške igre pojavile su se u virtualnom obliku već od 1997. godine, kada su izašle prve dvije digitalne kartaške igre: *Chron X* i *Sanctum*. Također, od 2002. godine igra *Magic: The Gathering* dobila je digitalni oblik[14]. Digitalne kolekcionarske kartaške igre imaju mogućnost da poprave neke od problema koji se pojavljivaju u fizičkom obliku. Jedna od najvećih prednosti digitalnog oblika je mogućnost igranja protiv igrača neovisno o njihovoj fizičkoj lokaciji[15]. Osim toga, pruže kvalitetne promjene u obliku digitalne kolekcije karata, mogućnost igranja protiv umjetne inteligencije, te digitano rangiranje i ljestvice, iako neki igrači više preferiraju igranje u fizičkom obliku zbog socijalnih ili strategijskih aspekata.

4.2.1. Hearthstone

Jedna od utjecajnijih digitalnih kolekcionarskih kartaških igara je videoigra *Hearthstone* od kompanije *Blizzard Entertainment*. Prvo izdanje videoigre pojavilo se 2014. godine, te je igra nastala kao eksperiment od ljubitelja kolekcionarskih kartaških igara u kompaniji. Videoigra je postigla veliku razinu uspjeha, bilježeći preko 100 milijuna globalnih igrača u 2018. godini[18].

Tematika igre osnovana je na drugu popularnu videoigru iste kompanije, *World of Warcraft*, što je pomoglo kod popularnosti videoigre na način da je igračima omogućila da skupljaju

svoje najdraže likove iz spomenute igre u obliku karata. Osim toga, mehanike i pravila videoigre napravljene su na način da se brzo mogu naučiti i razumjeti, čak i kao početnik u žanru. Uzimajući u obzir da je videoigra besplatna i karte se mogu nabaviti bez da igrač mora uložiti novce, *Hearthstone* je stvorio novi standard za digitalne kartaške igre, te je dugo vrijeme održavao takvu kvalitetu.

Slika 8: Primjer jedne runde videoigre *Hearthstone* (Izvor: [19])

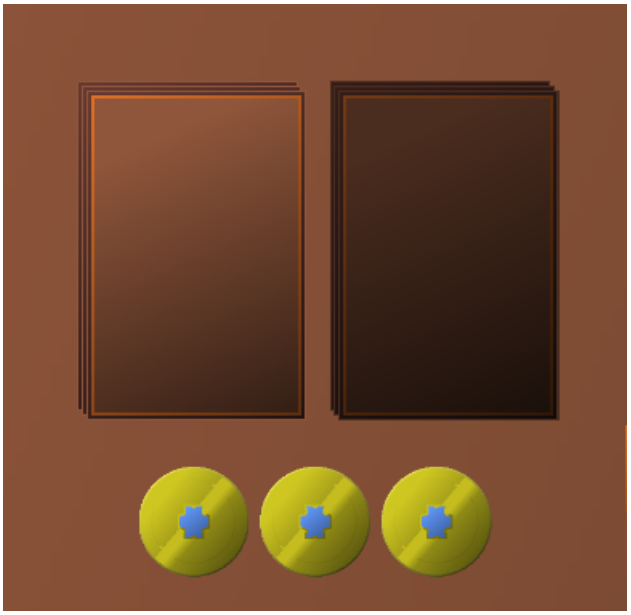
5. Razvoj kolekcionarske kartaške videoigre u Unity alatu

Ovo poglavlje posvetit će se izradi prototipa kolekcionarske kartaške videoigre u programskom alatu Unity. Detaljno će se obraditi tijek rada u Unity-u, što uključuje izradu igrače ploče, izradu predložaka za karte, integraciju vizualnih elemenata, izradu "AI" protivnika, te pisanje programskog koda za cjelovitu logiku videoigre.

5.1. Koncept i pravila videoigre

5.1.1. Potezi i resursi

Koncept i pravila videoigre dosta su jednostavna, ali omogućuju igraču da strateški razmišlja i odigra karte na način da maksimizira njihovu moguću vrijednost. Igrači odrađuju poteze naizmjenično, bez mogućnosti igranja karata tijekom protivnikovog poteza. Svaki potez, igrač izvlači prvu kartu na vrhu svog špila, te dobije na raspolaganje tri novčića koji predstavljaju resurs za igranje karata. Svaka karta, neovisno o vrsti, ima cijenu izraženu u novčićima. Kako bi igrač odigrao kartu, potreban mu je broj novčića jednak cijeni karte koju želi odigrati. Nakon što odigra kartu, broj novčića na raspolaganju umanji se za cijenu odigrane karte. Cijene karte variraju od jednog do tri novčića, pa igrač mora pažljivo odlučiti koje će karte odigrati u svom potezu kako bi najbolje iskoristio resurse na raspolaganju. Novčići se svaki potez osvježe na tri novčića, i više od tri nije moguće imati, što znači da nema poante u štednji resursa za nadolazeće poteze.



Slika 9: Novčići kao resurs za igranje karata, te glavni i odbacni špil (Izvor: vlastita izrada)

5.1.2. Karte

Postoje dvije vrste karti koje igrač može koristiti. Prva su sljedbenici. Oni predstavljaju razna bića i stvorenja koja služe kao glavni napadači i branitelji. Sljedbeničke karte, osim cijene, imaju vrijednost napada i vrijednost zdravlja. Kada sljedbenik napada drugog sljedbenika ili protivnika direktno, uzima se u obzir njegova vrijednost napada, koja se nanosi kao šteta na primatelja napada. Kada je sljedbenik primatelj štete, uzima se u obzir njegova vrijednost zdravlja, koja se umanji za vrijednost napada napadača. Ako vrijednost zdravlja sljedbenika postane manja ili jednaka nuli, sljedbenik se smatra uništenim, te se miješa u odbacni špil, odnosno groblje (eng. Graveyard). Sljedbenici imaju ključne riječi, koje definiraju na koji način djeluju tijekom borbe. Sljedbenici mogu imati jednu od sljedećih tri mogućih ključnih riječi:

- Zaštitnik (eng. *Guardian*) - sljedbenik s ovom ključnom riječi štiti sljedbenike koji se nalaze na lijevoj i desnoj strani od njega, te prima štetu koju bi oni primili umjesto njih.
- Nadjačanje (eng. *Overpower*) - sljedbenik s ovom ključnom riječi, kada nanese štetu na protivničkog sljedbenika u vrijednosti većoj od njegove vrijednosti zdravlja, prijenosi višak štete na životne bodove igrača, ili, ako je napadnuti sljedbenik zaštićen preko sljedbenika s ključnom riječi zaštitnik, ostatak štete prijenosi se sljedbeniku zaštitniku.
- Trnje (eng. *Thorns*) - sljedbenik s ovom ključnom riječi, kada je napadnut, također nanosi štetu na svog napadača u vrijednosti svoje vrijednosti napada.

Ključne riječi proširuju moguću strategiju u potezu na način da potaknu igrača da pozicionira svoje sljedbenike na način da mogu maksimalno iskoristiti svoje ključne riječi. Na primjer, sljedbenici zaštitnici mogu zaštiti vrijedne sljedbenike od neprijateljskih napada, dok sljedbenici s ključnom riječi "nadjačanje" mogu probiti kroz zaštitnike, ili nanositi štetu na protivnikove životne bodove indirektno preko njegovih sljedbenika. Sljedbenici također mogu imati posebne efekte, koji se okinu kada se njihov određeni uvjet ispuni. Uvjeti za okidanje efekta variraju i mogu biti gotovo svaka radnja u potezu, na primjer, kada se sljedbenik odigra, kada sljedbenik primi štetu, na početku ili na kraju poteza i sl. Efekti dodaju dodatni element maksimiziranja potencijalne vrijednosti karte, ovisno koliko igrač ispunjuje njihove uvjete za okidanje. Na sljedećoj slici nalazi se primjer sljedbenika sa svim navedenim elementima.

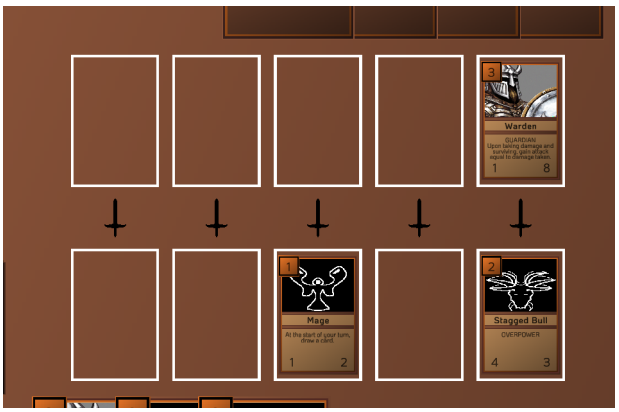
Slika 10: Sljedbenik s napadnom vrijednosti od 1, zdravstvenom vrijednosti od 8, ključnom riječi "zaštitnik", te posebnim efektom. (Izvor: vlastita izrada)

Druga vrsta mogućih karata su čarolije (eng. *spells*). One nemaju tijelo, to jest napadnu i zdravstvenu vrijednost, i ne mogu se postaviti direktno na igraču ploču, ali uvijek imaju poseban efekt koji se okine kada se ona odigra. Efekti koje pružaju čarolije su često utjecajnije od efekata sljedbenika, kao kompenzacija za nedostatak tijela. Čarolije omogućuju igraču da direktno utječe na stanje igrače ploče, bilo to kroz pojačanje svojih sljedbenika, ili nanošenje štete na protivnikove sljedbenike.

Slika 11: Primjer karte čarolije (Izvor: vlastita izrada)

5.1.3. Igrača ploča i tijek igre

Igrača ploča sastoji se od ukupno deset polja za postavljanje sljedbenika, to jest pet za svakog igrača. Igrač, kada je njegov potez i ima dovoljno resursa, može postavljati sljedbenike na svoju stranu igrače ploče, ili preko karta čarolije utjecati na trenutno stanje ploče. Svaki sljedbenik može zauzeti jedno polje na ploči, te kada je uništen, njegovo polje se oslobađa. Kada bilo koji od igrača završi svoj potez, kreće napadačka faza. Igrač koji je završio svoj potez ulazi u napadačku fazu kao napadač, dok suprotni igrač ulazi u napadačku fazu kao branitelj. Sljedbenici napadača redom napadaju protivničku stranu igrače ploče, počevši s lijeva na desno. Ako se nasuprot sljedbenika napadača nalazi protivnički sljedbenik, napadač nanosi štetu protivničkom sljedbeniku. Ako je nasuprotno polje prazno, napadač će direktno napasti igrača i oduzeti mu životne bodove. Kada su životni bodovi jednog igrača manji ili jednaki nula, igra se završava te se suprotni igrač smatra kao pobjednik. Kako balans igre ne bi bio naklonjen igraču koji prvi uzima potez, napdačke faze počinju tek nakon drugog poteza.



Slika 12: Igrača ploča, s postavljenim sljedbenicima (Izvor: vlastita izrada)

5.2. Inspiracija

Kao glavne inspiracije za mehanike i pravile igre ističu se videoigra *Slay the Spire* kompanije *Mega Crit Games*, te videoigra *Inscryption* koju je razvio Daniel Mullins. U videoigri *Slay the Spire*, igrač koristi karte kako bi direktno napadao neprijatelja, koristeći ograničeni broj resursa svaki potez. Početna vrijednost resursa u igri je tri, te većina karata imaju cijene između jedan i tri resursa, što daje vrijednost svakoj karti neovisno o trenutnom broju poteza. U videoigri *Inscryption* igrač postavlja karte koje predstavljaju razne divlje životinje na igraču ploču, koja se sastoji od osam polja, to jest po četiri za svakog igrača. Karte obično napadaju samo karte koje se nalaze nasuprot sebe, što potiče štrateško pozicioniranje karata.

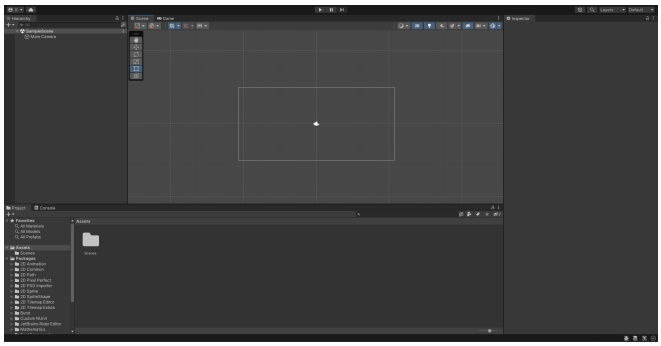
Slika 13: Igrača ploča videoigre *Inscryption* (Izvor: [20])

5.3. Razvoj videoigre - početni koraci

5.3.1. Izrada 2D projekta

Kao početak potrebno je napraviti novi 2D Unity projekt. Videoigra će bit u potpunosti 2D, iako neke videoigre, poput primjerice *Hearthstone* koriste 3D objekte kako bi videoigra izgledala više realistična. Nakon kreiranja praznog 2D projekta, Unity će pripremiti novu praznu scenu, zajedno s objektom kamere.

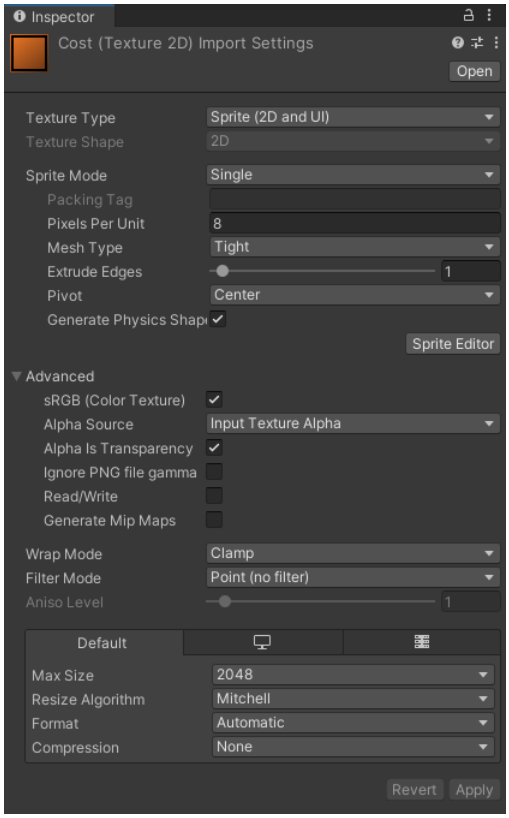
Plan je da se cijela videoigra odvija preko jednog canvas objekta, jer canvas i njegove komponente nude adekvatna rješenja za neke od mehanike koje je potrebno implementirati. Time je prvi objekt koji se dodaje na scenu **Canvas** objekt. Taj objekt će biti zadužan za iscrtavanje svih objekta koji će se pojavljivati u videoigri, i svaki objekt koji se dodaje na scenu dodaje se kao dijete canvas objekta.



Slika 14: Prazna scena nakon kreiranja novog 2D projekta u Unity alatu (Izvor: vlastita izrada)

5.3.2. Uvoz vizualnih elemenata

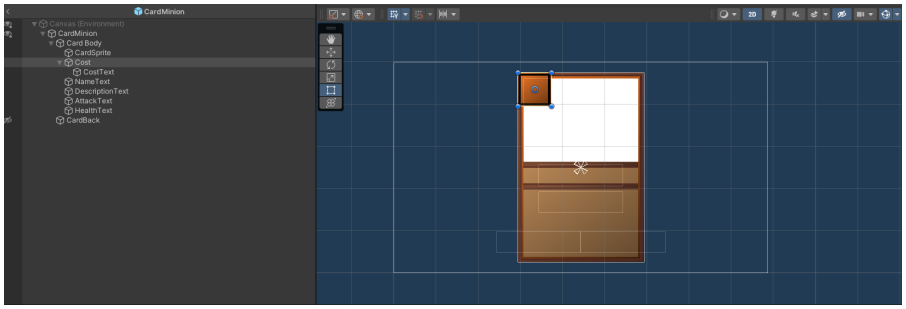
Vizualni materijali, poput slika za karte, igraču ploču, polja i sl. potrebno je dodati u projektne datoteke. Najjednostavniji način dodavanja materijala je mišom ih povući u prostor projektnih materijala. Time su materijali dodani u projekt i moguće ih je koristiti u izradi videoigre. Budući da su vizualni materijali korišteni u ovome projektu vrlo niske rezolucije, točnije 32x32, potrebno je prilagoditi postavke uvođenih materijala kako bi se pravilno prikazivali. To se radi na način da se novododani vizualni elementi obilježe i preko inspektora podese njihove postavke. Potrebno je postaviti postavku **Filter mode** na Point (no filter), te postavku **Compression** na None. Također, vrijednost postavke **Pixels Per Unit** potrebno je postaviti sukladno s rezolucijom slike. U ovom slučaju, većina elemenata imaju tu vrijednost postavljenu kao jedan od višekratnika broja 8. Time će se slike prikazivati u većem obliku, te će biti izoštrene.



Slika 15: Postavke većine slika u projektu (Izvor: vlastita izrada)

5.4. Izrada predložaka za karte

Karte su glavni element gotovo bilo koje kartaške igre, te su kao takve najbitniji element u izradi kartaške videoigre. Svaka karta, neovisno o vrsti, sadrži cijenu koja je potrebna da se odigra, ime karte, te kratki opis. Ti elementi ostati će jednaki u obe vrste karti. Za manju složenost koda, postoje dva predloška, to jest predložak za karte od sljedbenika, te predložak za karte čarolije. Za prikazivanje tekstualnog sadržaja, koristi se jedan od Unity ugrađenih alata, **TextMeshPro**, koji omogućuje veliku slobodu u prilagođavanju izgleda teksta. Na sljedećoj slici prikazuje se gotovi raspored elemenata za predložak karte sljedbenika.



Slika 16: Predložak karte sljedbenika (Izvor: vlastita izrada)

Jedina razlika kod rasporeda objekata karta čarolije je da su izostavljeni objekti za prikaz vrijednosti napada i vrijednosti zdravlja, jer karte čarolije nemaju tijela. Slika koja predstavlja portret karte je na slici prikazana kao bijela pozadina, no ona će programski biti promijenjena ovisno o podacima karte.

5.4.1. Abstraktna klasa "Card"

Karte sljedbenika i karte čarolije, iako su konceptualno različite, sadrže dovoljno jednakih atributa i metoda da ih je moguće ujediniti kao abstraktnu klasu, te uvesti specifične funkcionalnosti vrsta karta pojedinačno preko polimorfizma. U nastavku je prikazano zaglavlje abstraktne klase "Card", koja služi kao bazna klasa karte i sadrži sve atribute i metode koje karta mora imati, neovisno o vrsti.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;
using UnityEngine.EventSystems;
```

Ovo su biblioteke koje će biti korištene u većini skripti u ovome projektu. Najbitnija biblioteka je **UnityEngine**, koja sadrži cjelovito sučelje Unity aplikacije (eng. *Application Interface* - kraće API), te preko nje možemo pristupiti svim Unity metodama i komponentama.

```
public abstract class Card : MonoBehaviour, IPointerEnterHandler,
    IPointerExitHandler, IPointerClickHandler
{
    public string Name;
```

```
public string Description;
public int ManaCost;
public bool IsTravelling = false;

protected int lastIndex;
protected const float FLIP_SPEED = 10f;
protected const float CARD_TRAVEL_SPEED = 10f;
protected const float MINION_SIZE = 0.6f;
```

Klasa "Card" je postavljena kao abstraktna klasa, jer preko nje želimo referencirati ostale vrste karti koje ju nasljeđuju. Klasa "Card" nasljeđuje glavnu Unity klasu za objekte igre, **MonoBehaviour**, koja pretvara skriptu koja ju nasljeđuje u komponentu koja se može dodati na objekt igre. Time također imamo pristup važnim ugrađenim metodama koje su bitne za funkcionalnost igre. One koje su najbitnije za razvoj ovog projekta navedene su u nastavku:

- **void Start()** - metoda se poziva kada se skripta prvo učita, prije svih metoda Update.
- **void Update()** - metoda se poziva svaki okvir (eng. *frame*) videoigre.

Osim klase "MonoBehaviour", klasa "Card" također implementira sučelja (eng. *interface*) **IPointerEnterHandler**, **IPointerExitHandler**, te **IPointerClickHandler**. Navedena sučelja dodaju metode za prepoznavanje kada miš uđe u prostor karte, kada izađe iz prostora karte, te kada korisnik pritisne na kartu. Navedena sučelja dostupna su preko biblioteke "UnityEngine.EventSystems".

Nakon deklaracije klase, navedeni su atributi koje svaka vrsta karte nasljeđuje. Atributi s modifikatorom pristupa "public" su potencijalno bitni za referenciranje izvan svoje klase, dok su atributi s modifikatorom pristupa "protected" namijenjeni samo za baznu klasu, te klase koje nasljeđuju baznu klasu. Svi navedeni atributi, osim prva tri, služe za kalkulacije kod kretanja karte na sučelju.

```
[SerializeField]
protected Image Texture;
[SerializeField]
protected GameObject CardBack;
[SerializeField]
protected TextMeshProUGUI NameText;
[SerializeField]
protected TextMeshProUGUI CostText;
[SerializeField]
protected TextMeshProUGUI DescriptionText;
```

Navedeni atributi služe za referenciranje objekata u predlošku, poput teksta i portreta karte.

```
public bool IsFlipped;
public bool IsInHand = false;
public bool IsInspected = false;
public bool IsSelected = false;
```

```
public Hand hand = null;
public Vector3 PositionInHand;
```

U ovom dijelu ističe se atribut "hand" kojem je tip klase "Hand". Taj atribut referencira ruku u kojoj će se karta nalaziti. Klasa "Hand" također je abstraktna klasa, te će detaljnije biti objašnjena kasnije.

```
public abstract void LoadInfo();
```

Metoda "LoadInfo()" je abstraktna, što znači da klase koje naslijeđuju baznu klasu moraju implementirati funkcionalnost metode. Ona služi za popunjavanje potrebnih informacija o karti, ovisno o vrsti, te ažuriranje teksta i slike predloška karte. Pošto sljedbenici i čarolije imaju nešto različite podatke, obje će klase imati svoju implementaciju metode.

```
public IEnumerator Flip()
{
    Vector3 currentScale = GetComponent<RectTransform>().localScale;
    if (!IsFlipped)
    {
        while (GetComponent<RectTransform>().localScale.x > 0.1f)
        {
            GetComponent<RectTransform>().localScale = new Vector3(Mathf.Lerp(
                GetComponent<RectTransform>().localScale.x, 0, FLIP_SPEED * Time
                .deltaTime), currentScale.y, currentScale.z);
            yield return null;
        }
        CardBack.SetActive(true);
        while (GetComponent<RectTransform>().localScale.x < currentScale.x - 0.1
            f)
        {
            GetComponent<RectTransform>().localScale = new Vector3(Mathf.Lerp(
                GetComponent<RectTransform>().localScale.x, currentScale.x,
                FLIP_SPEED * Time.deltaTime), currentScale.y, currentScale.z);
            yield return null;
        }

        IsFlipped = true;
    }
    else
    {
        while (GetComponent<RectTransform>().localScale.x > 0.1f)
        {
            GetComponent<RectTransform>().localScale = new Vector3(Mathf.Lerp(
                GetComponent<RectTransform>().localScale.x, 0, FLIP_SPEED * Time
                .deltaTime), currentScale.y, currentScale.z);
            yield return null;
        }
        CardBack.SetActive(false);
        while (GetComponent<RectTransform>().localScale.x < currentScale.x - 0.1
            f)
        {
            GetComponent<RectTransform>().localScale = new Vector3(Mathf.Lerp(
```

```
                GetComponent<RectTransform>().localScale.x, currentScale.x,
                FLIP_SPEED * Time.deltaTime), currentScale.y, currentScale.z);
            yield return null;
        }
        IsFlipped = false;
    }

    GetComponent<RectTransform>().localScale = currentScale;
    yield break;
}
```

Metoda "Flip()" ima povratnu vrijednost "IEnumerator", što znaći da je to zapravo **korutina** (eng. *Coroutine*). Korutine su metode koje mogu obustaviti svoj rad sve dok se određena instrukcija ne završi[21]. Korutine se uglavnom koriste kada se neka radnja odvija preko više okvira videoigre. U ovom slučaju, metoda "Flip()" simulira okretanje karte na suprotnu stranu, na način da se postepeno smanjuje x vrijednost skalarnog vektora karte sve dok nije jednaka 0. Nakon toga se promjeni aktivno stanje poleđine karte, što će ili prekriti ili otkriti kartu. Na kraju se x vrijednost skalarnog vektora karte vraća na vrijednost prije pozivanja metode.

```
public void SetHandIndex()
{
    lastIndex = GetComponent<RectTransform>().transform.GetSiblingIndex();
}

public int GetHandIndex()
{
    return lastIndex;
}
```

Metode "SetHandIndex()" i "GetHandIndex()" su pomoćne metode za određivanje pozicije karte u ruci. Canvas element iscrtava objekte ovisno o njihovoj poziciji u hijerarhiji, pa je potrebno kartu postaviti kao zadnje dijete ruke kako bi se ono vidjelo preko ostalih. "SetHandIndex()" metoda sprema zadnju poziciju karte u hijerarhiji, dok ju "GetHandIndex()" vraća.

```
public void OnPointerEnter(PointerEventData eventData)
{
    if (IsInHand && hand is PlayerHand)
    {
        PlayerHand player = (PlayerHand) hand;
        player.InspectCard(this);
    }
}

public void OnPointerExit(PointerEventData eventData)
{
    if (IsInspected && hand is PlayerHand)
    {
        PlayerHand player = (PlayerHand) hand;
        player.CancelInspect(this);
    }
}
```

```
    public void OnPointerClick(PointerEventData eventData)
    {
        if (IsInHand && IsInspected && hand is PlayerHand)
        {
            PlayerHand player = (PlayerHand) hand;
            player.SelectCard(this);
        }
    }
}
```

Metoda "OnPointerEnter" pozove se kada pokazivač prođe preko karte, te poziva metodu "InspectCard" klase "PlayerHand", koja služi za isticanje karte u ruci igrača. "OnPointerExit" se pozove kada pokazivač izađe iz karte, te poziva metodu "CancelInspect" koja prekida isticanje trenutne karte. "OnPointerClick" se pozove kada igrač pritisne na kartu, i time se poziva metoda "SelectCard" koja označuje trenutnu kartu kao obilježenu od igrača.

5.4.2. Abstraktna klasa "CardInfo"

Korištenje predložaka za kreiranje karata ima prednosti, poput dodatne mogućnosti prilagodbe svake karte, ali u slučaj da je potrebno mijenjati funkcionalnost ili izgled svih karti, potrebno bi bilo svaki predložak mijenjati posebno. U tu svrhu se za kreiranje karti koriste skriptabilni objekti. Klasa "CardInfo" je bazna klasa za spremanje podataka koje sve karte, neovisno o vrsti, moraju imati.

```
public abstract class CardInfo : ScriptableObject
{
    [Header("Display name")]
    public string Name;

    [Header("Description")]
    [TextArea(1, 6)]
    public string Description;

    [Header("Graphics")]
    public Sprite Texture;

    [Header("Cost")]
    public int Cost;

    [Header("Effects")]
    public List<CardEffect> Effects;
}
```

Klasu "CardInfo" naslijedit će klase "MinionInfo" i "SpellInfo", zbog razlike podataka među vrstama karti. Ona sprema ime, opis, teksturu portreta, cijenu karte, te efekte karte. Ti podaci, zajedno sa specifičnim podacima ovisno o vrsti klase, učitat će se pomoću abstraktne metode "LoadInfo()", koju svaka vrsta karte implementira u svojoj klasi.

5.4.3. Klasa "MinionInfo"

Klasa "MinionInfo" nadopunjuje podatke klase "CardInfo" s podacima koji su posebni za karte sljedbenike.

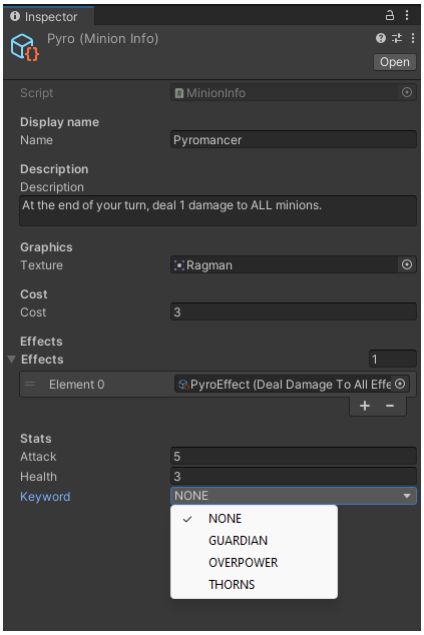
```
public enum KeywordType
{
    NONE,
    GUARDIAN,
    OVERPOWER,
    THORNS
}


[CreateAssetMenu(menuName = "Minion info", fileName = "MinionInfo")]
public class MinionInfo : CardInfo
{

    [Header("Stats")]
    public int Attack;
    public int Health;
    public KeywordType Keyword;
}
```

Enumeracija "KeywordType" predstavlja moguće ključne riječi koje sljedbenik može imati. Atribut "CreateAssetMenu" omogućuje da se stvori instanca skriptabilnog objekta "MinionInfo". Pošto bazna klasa "CardInfo" nasljeđuje klasu "ScriptableObject", ona se smatra kao takav te klase koje ju nasljeđuju također se broje kao skriptabilni objekti.

Podaci koji su posebni za sljedbenike su njihova napadna vrijednost (atribut "Attack"), zdravstvena vrijednost (atribut "Health"), te njihova ključna riječ.



Slika 17: Primjer izrade nove instance sljedbenika (Izvor: vlastita izrada)

5.4.4. Klasa "SpellInfo"

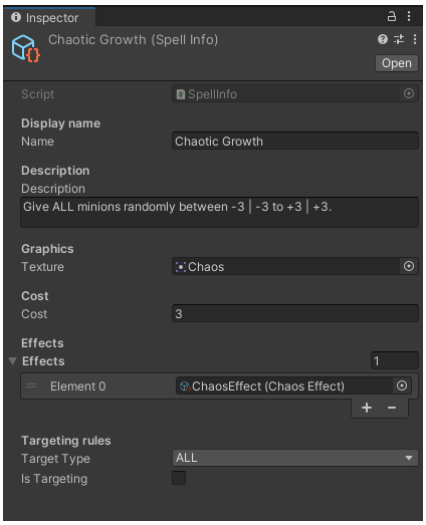
Klasa "SpellInfo", kao i klasa "MinionInfo", nasljeđuje klasu "CardInfo", te nadopunjuje njezine podatke s podacima posebnim za karte čarolije.

```
public enum TargetType
{
    FRIENDLY,
    ENEMY,
    ALL
}

[CreateAssetMenu(menuName = "Spell info", fileName = "SpellInfo")]
public class SpellInfo : CardInfo
{
    [Header("Targeting rules")]
    public TargetType TargetType;
    public bool IsTargeting;

}
```

Enumeracija "TargetType" označava da li je čarolija namijenjena za igračeve sljedbenike, protivnikove sljedbenike, ili sve sljedbenike. Atribut "IsTargeting" označava da li čarolija utječe na jednog sljedbenika, ili na više.



Slika 18: Primjer izrade nove instance čarolije (Izvor: vlastita izrada)

5.4.5. Klasa "Minion"

Klasa "Minion" je glavna klasa za sve funkcionalnosti i mehanike koje su posebne za karte sljedbenike. Skripta klase "Minion" je postavljena kao komponenta na predlošku karte sljedbenika. Sljedbenici su namijenjeni da budu postavljeni na igraču ploču i da djeluju kao "tijela" koja se međusobno bore, za razliku od čarolija koji nestanu kada se odigraju.

```
public class Minion : Card
{
```

```
    [SerializeField]
    private Image CardBody;

    public int CurrentAttack;
    public int CurrentHealth;
    public KeywordType Keyword;
    public List<CardEffect> Effects;
    private bool IsDead = false;

    public MinionInfo Info;

    public UnityEvent OnDeath;
    public UnityEvent<int> OnDamageTaken;
    public UnityEvent<Minion> OnAttack;

    public TextMeshProUGUI AttackText;
    public TextMeshProUGUI HealthText;

    public CardSlot PositionOnBoard = null;
```

Glavni opisni atributi klase "Minion" su "CurrentAttack", "CurrentHealth", "Keyword", te lista "Effects. Ti podaci se učitaju iz skriptabilnog objekta tipa "MinionInfo" kada se predložak karte prvo instancira. Atributi koji su tipa "UnityEvent" predstavljaju događaje koji se pozovu kada se njihov uvjet ispuni. Primjerice, događaj "OnDeath" će pozvati sve metode koje imaju okidač spojen na taj događaj. Metode efekata sljedbenika pretplaćaju se na događaje kada se sljedbenik postavi na igraču ploču.

Atribut tipa "CardSlot" predstavlja referencu na polje na kojem je sljedbenik postavljen. Ako je sljedbenik u ruci jednog od igrača, atribut poprima vrijednost null.

```
public IEnumerator MoveToSlot()
    {
        IsTravelling = true;

        while (IsTravelling && this != null)
        {
            float x = Mathf.Lerp(GetComponent<RectTransform>().position.x,
                PositionOnBoard.GetComponent<RectTransform>().position.x,
                CARD_TRAVEL_SPEED * Time.deltaTime);
            float y = Mathf.Lerp(GetComponent<RectTransform>().position.y,
                PositionOnBoard.GetComponent<RectTransform>().position.y,
                CARD_TRAVEL_SPEED * Time.deltaTime);
            Vector3 movement = new Vector3(x, y);
            GetComponent<RectTransform>().position = movement;
            float diffX = Mathf.Abs(PositionOnBoard.GetComponent<RectTransform>().
                position.x - GetComponent<RectTransform>().position.x);
            float diffY = Mathf.Abs(PositionOnBoard.GetComponent<RectTransform>().
                position.y - GetComponent<RectTransform>().position.y);
            GetComponent<RectTransform>().localScale = new Vector3(Mathf.Lerp(
                GetComponent<RectTransform>().localScale.x, MINION_SIZE,
                CARD_TRAVEL_SPEED * Time.deltaTime), Mathf.Lerp(GetComponent<
                RectTransform>().localScale.y, MINION_SIZE, CARD_TRAVEL_SPEED * Time
```

```
            .deltaTime));
        if (diffX <= 0.01f && diffY <= 0.01f)
        {
            IsTravelling = false;
        }
        yield return null;
    }
    yield break;
}
```

Metoda "MoveToSlot()" je korutina koja je zaslužna za pomicanje karte sljedbenika na polje koje je igrač odredio. Brzina kretanja karte određena je konstantom "CARD_TRAVEL_SPEED" pomnoženom s vrijednosti "Time.deltaTime", koja predstavlja zadnje vrijeme između dva okvira igre. Sljedbenici se također smanjuju u veličini kada putuju do polja, što je određeno konstantom "MINION_SIZE".

```
public void ToggleRaycastTarget(bool value)
{
    CardBody.raycastTarget = value;
    Texture.raycastTarget = value;
    CardBack.GetComponent<Image>().raycastTarget = value;
    CostText.raycastTarget = value;
    AttackText.raycastTarget = value;
    HealthText.raycastTarget = value;
    NameText.raycastTarget = value;
    DescriptionText.raycastTarget = value;
}

public override void LoadInfo()
{
    Name = Info.Name;
    Description = Info.Description;
    ManaCost = Info.Cost;
    CurrentAttack = Info.Attack;
    CurrentHealth = Info.Health;
    Keyword = Info.Keyword;

    Texture.sprite = Info.Texture;
    AttackText.text = CurrentAttack.ToString();
    HealthText.text = CurrentHealth.ToString();
    NameText.text = Name;
    CostText.text = ManaCost.ToString();
    DescriptionText.text = Description;
    Effects = new List<CardEffect>();

    foreach (CardEffect effect in Info.Effects)
    {
        CardEffect cardEffect = (CardEffect) ScriptableObject.CreateInstance(
            effect.GetType());
        cardEffect.SetCastingCard(this);
        cardEffect.Condition = effect.Condition;
        Effects.Add(cardEffect);
    }
```

```
}

public void UpdateInfo()
{
    AttackText.text = CurrentAttack.ToString();
    HealthText.text = CurrentHealth.ToString();
}
```

Metoda "ToggleRaycastTarget(bool value)", ovisno o parametru, isključuje ili uključuje mogućnost da pokazivač miša prepoznaje tijelo karte kao interaktivni element. Metoda se koristi u svrhu biranja mete kod čarolija, koje je ovisno o polju na kojem se sljedbenik nalazi, a ne postavljenim sljedbenicima. Isključenjem tijela karte kao interaktivnog elementa omogućuje da se polje ispod sljedbenika može pritisnuti.

Metoda "LoadInfo()" služi za učitavanje i ažuriranje svih atributa karte iz skriptabilnog objekta (atributa "Info" tipa "MinionInfo") koji opisuje sljedbenika. Efekti karte također su skriptabilni objekti, i oni se instanciraju tijekom izvođenja igre jer je potrebno spremiti referencu karte kojoj pripada efekt u same podatke efekta. Metoda "UpdateInfo()" ažurira tekstualne vrijednosti koje su vidljive na karti. Metoda se poziva kada se vrijednosti napada ili zdravlja sljedbenika promijene kako bi se uskladile tekstualne i aktualne vrijednosti.

```
public void TakeDamage(int amount)
{
    CurrentHealth -= amount;
    UpdateInfo();
    if (CurrentHealth <= 0 && !IsDead)
    {
            StartCoroutine(Die());
    }
    else OnDamageTaken.Invoke(amount);
}

public void ChangeStats(int attack, int health)
{
    CurrentAttack += attack;
    CurrentHealth += health;

    if (CurrentAttack < 0) CurrentAttack = 0;
    UpdateInfo();
    if (CurrentHealth <= 0 && !IsDead) StartCoroutine(Die());
}

public IEnumerator Die()
{
    IsDead = true;
    BoardManager.instance.OnMinionDied.Invoke(this);
    OnDeath.Invoke();

    BoardManager.instance.UnsubscribeMinionEvents(this);
    PositionOnBoard.PlacedMinion = null;
    hand.DiscardPile.ShuffleCardInDeck(Info);
    StartCoroutine(Flip());
```

```
        while (!IsFlipped) yield return null;
        yield return new WaitForSeconds(0.3f);
        Destroy(gameObject);
        yield break;
    }
}
```

Metoda "TakeDamage(int amount)" je metoda za nanošenje štete na sljedbenika. Zdravstvena vrijednost (Health) sljedbenika se umanji za parametar, te ako je nakon toga zdravstvena vrijednost manja ili jednaka 0, pozove se metoda "Die()" koja poziva sve događaje uvjetovane sa uništenjem sljedbenika, zatim odjavljuje sve ostale događaje vezane za tog sljedbenika, i na kraju se dodaje u groblje igrača i uništava s polja.

Metoda "ChangeStats(int attack, int health)" mijenja vrijednosti napada i zdravlja sljedbenika ovisno o parametrima. Ako je vrijednost napada sljedbenika manja od nule nakon promjene, postavlja se na 0. Ova metoda se koristi kod efekata karti koje mijenjaju statistike sljedbenika, bilo to na pozitivni ili negativni način.

5.4.6. Klasa "Spell"

Klasa "Spell" objedinjuje atribute i metode koji su posebni za karte čarolije.

```
public class Spell : Card
{
    public SpellInfo Info;

    public TargetType TargetType;
    public bool IsTargeting;

    public List<CardEffect> Effects;
```

Od atributa, jedino su "TargetType" i "IsTargeting" jedinstveni kartama čarolijama. Atribut "Info" tipa "SpellInfo" je skriptabilni objekt koji drži podatke o vrsti čarolije, koji se kasnije učitaju preko metode "LoadInfo()".

```
public override void LoadInfo()
    {
        Name = Info.Name;
        Description = Info.Description;
        ManaCost = Info.Cost;

        Texture.sprite = Info.Texture;
        NameText.text = Name;
        CostText.text = ManaCost.ToString();
        DescriptionText.text = Description;

        TargetType = Info.TargetType;
        IsTargeting = Info.IsTargeting;
        Effects = new List<CardEffect>();

        foreach (CardEffect effect in Info.Effects)
```

```
        {
            CardEffect cardEffect = (CardEffect)ScriptableObject.CreateInstance(
                effect.GetType().ToString());
            if (cardEffect != null) cardEffect.SetCastingCard(this);
            cardEffect.Condition = effect.Condition;
            Effects.Add(cardEffect);
        }
    }
```

Metoda "LoadInfo()" gotovo je identična kao i u klasi "Minion", osim što ažurira podatke posebne za karte čarolije.

```
public IEnumerator MoveToDiscardPile()
    {
        IsTravelling = true;

        while (IsTravelling && this != null)
        {
            float x = Mathf.Lerp(GetComponent<RectTransform>().position.x, hand.
                DiscardPile.GetComponent<RectTransform>().position.x,
                CARD_TRAVEL_SPEED * Time.deltaTime);
            float y = Mathf.Lerp(GetComponent<RectTransform>().position.y, hand.
                DiscardPile.GetComponent<RectTransform>().position.y,
                CARD_TRAVEL_SPEED * Time.deltaTime);
            Vector3 movement = new Vector3(x, y);
            GetComponent<RectTransform>().position = movement;
            float diffX = Mathf.Abs(hand.DiscardPile.GetComponent<RectTransform>().
                position.x - GetComponent<RectTransform>().position.x);
            float diffY = Mathf.Abs(hand.DiscardPile.GetComponent<RectTransform>().
                position.y - GetComponent<RectTransform>().position.y);
            GetComponent<RectTransform>().localScale = new Vector3(Mathf.Lerp(
                GetComponent<RectTransform>().localScale.x, MINION_SIZE,
                CARD_TRAVEL_SPEED * Time.deltaTime), Mathf.Lerp(GetComponent<
                RectTransform>().localScale.y, MINION_SIZE, CARD_TRAVEL_SPEED * Time
                .deltaTime));
            if (diffX <= 0.01f && diffY <= 0.01f)
            {
                IsTravelling = false;
            }
            yield return null;
        }
        yield break;
    }
}
```

Korutina "MoveToDiscardPile()" vrlo je slična metodi "MoveToSlot()" u klasi "Minion", ali umjesto da se karta kreće prema polju, kreće se prema groblju. Razlog tome je da protivnik može vidjeti koja je karta odigrana.

5.4.7. Klasa "CardEffect"

Efekti su, kao i sljedbenici i čarolije, spremljeni kao skriptabilni objekti, to jest instance skriptabilnog objekta tipa klase "CardEffect". Svaki efekt zapravo ima svoju klasu koja nasljeđuje klasu "CardEffect", te implementira jednu ili više njezinih metoda.

```
public enum EffectCondition
{
    OnDeath,
    OnPlay,
    OnTurnStart,
    OnTurnEnd,
    OnAttack,
    OnMinionDied,
    OnMinionAttack,
    OnTakingDamage
}
```

Enumeracija "EffectCondition" predstavlja sve moguće događaje koji mogu uvjetovati pozivanje efekta. Svaki efekt može imati samo jedan uvjet.

```
[CreateAssetMenu(menuName = "Card Effect", fileName = "CardEffect")]
public class CardEffect : ScriptableObject
{
    [Header("Trigger condition")]
    public  EffectCondition Condition;

    protected Card CastingCard;

    public virtual void Apply()
    {
        return;
    }
    public virtual void Apply(Minion target)
    {
        return;
    }

    public virtual void Apply(Minion target, int amount)
    {
        return;
    }

    public virtual void Apply(List<Minion> targets)
    {
        return;
    }

    public virtual void Apply(int amount)
    {
        return;
    }
```

```
    public void SetCastingCard(Card card)
    {
        castingCard = card;
    }
}
```

Atribut "CastingCard" tipa "Card" sprema referencu karte kojoj efekt pripada preko metode "SetCastingCard(Card card)". Moguće je da postoje više karti s istim efektom u jednom trenu, te kako bi se spriječilo prebrisavanje reference karte, svaki efekt se posebno instancira po karti i sprema se njegova referenca u listu "Effects", koju svaka karta sadrži.

Virtualna metoda "Apply" i njezina preopterećenja služe za implementaciju funkcionalnosti samog efekta. Klase pojedinačnih efekata implementiraju jednu od preopterećenih metoda ovisno o uvjetovanom događaju. Kada se efekti sljedbenika pretplaćivaju na događaje, pretplatit će se ona metoda "Apply" koja odgovara parametrima događaja. Primjerice, događaj "OnDamageTakenint " koji predstavlja primanje štete kao sljedbenik, kao parametar prima vrijednost tipa integer, što znači da će se na taj događaj pretplatiti svi efekti koji implementiraju metodu "Apply(int amount)".

Za karte čarolije, uvjet nije bitan, jer čarolije smatraju sve svoje efekte kao da imaju uvjet "Onplay", što znači da se pozovu kada se karta odigra.

U nastavku se nalazi primjer klase efekta "ShrapnelEffect", koji se poziva kada se sljedbenik kojem pripada efekt uništi. Efekt svim protivničkim sljedbenicima uzrokuje štetu u vrijednosti 1.

```
[CreateAssetMenu(menuName = "Effects/ShrapnelEffect")]
public class ShrapnelEffect : CardEffect
{
    public override void Apply()
    {
        if (castingCard.hand is PlayerHand)
        {
            foreach (var slot in BoardManager.instance.GetPlayerSlots(false))
            {
                if (slot.IsOccupied()) slot.PlacedMinion.TakeDamage(1);
            }
        }

        else
        {
            foreach (var slot in BoardManager.instance.GetPlayerSlots(true))
            {
                if (slot.IsOccupied()) slot.PlacedMinion.TakeDamage(1);
            }
        }
    }
}
```

5.5. Igrač i mehanike igrača

Sam objekt igrača nema vizualni oblik, već je predstavljen kao pozicija na zaslonu gdje se nalaze igračeve karte. Objekt igrača na sebi ima skriptu koja nasljeđuje baznu klasu "Hand", ovisno o kojem se igraču radi. U ovom projektu, protivnik je programski upravljan, i time ima zasebnu klasu kako bi se realizirale te funkcionalnosti.

Mehanike koje su usko povezane za igrača su oba špila, životni bodovi igrača, te novčići za igranje karata. Njihova programska logika će također biti pokrivena u ovom poglavlju.

5.5.1. Abstraktna klasa "Hand"

Klasa "Hand" ujedinjuje sve atribute i metode koje su bitne za osnovnu funkcionalnost ljudskog i programskog igrača.

```
public abstract class Hand : MonoBehaviour
{
    protected const int MAX_HAND_SIZE = 6;
    protected const float CARD_HOVER_DISTANCE = 10f;
    protected const float CARD_LERP_SPEED = 10f;
    protected const float CARD_GAP = 15f;

    public ManaCounter ManaCounter;
    public Deck Deck;
    public Deck DiscardPile;
    public PlayerHP PlayerHP;

    protected Card SelectedCard;
    protected int EmptyDeckDamage = 1;

    [SerializeField] protected List<GameObject> CardsInHand = new List<GameObject>()
        ;
    public abstract void DrawCard(Deck PlayerDeck);

    public abstract IEnumerator PlayCard(CardSlot slot);
}
```

Konstante služe za kalkuliranje pozicije karta u ruci, primjerice detaljni prikaz karte, razmak između karti u ruci te brzina kretanja karte iz špila u ruku. Atributi s modifikatorom pristupa "public" koji slijede nakon konstanta su reference za bitne mehanike igrača. To su atributi "ManaCounter" koji predstavlja brojač resursa za igranje karata, "Deck" koji predstavlja glavni špil iz kojeg igrač izvlači karte, "DiscardPile" koji predstavlja groblje u koje se miješaju karte koje su odigrane ili sljedbenici koji su uništeni, te "PlayerHP" koji predstavlja brojač za igračeve životne bodove.

Atribut "SelectedCard" predstavlja kartu koju je igrač obilježio da odigra. Programski upravljan igrač ne mora razmišljati o kartama koje će odigrati za razliku od ljudskog igrača, te u njegovoj klasi uglavnom ne koristi atribut "SelectedCard". Atribut "EmptyDeckDamage" predstavlja štetu koju igrač primi kada pokuša izvući kartu iz špila koji je prazan. Mehanika je

implementirana jer je moguće da oba igrača ostanu bez načina da nanose štetu drugom igraču, što bi značilo da nije moguće završiti igru u tom slučaju.

Lista "CardsInHand" tipa "GameObject" pohranjuje reference od instanci predložaka karti koje igrač drži u ruci. Instance predložaka karti instanciraju se kada se izvlače iz špila, a uništavaju se kada se odigraju ili kada se njihov sljedbenik uništi. Skripta instance dohvaća se preko pozivanja metode "GetComponentCard ()" na referenci objekta.

Metoda "DrawCard(Deck PlayerDeck)" služi za izvlačenje karte iz špila koji se proslijedi u parametru. Postupak izvlačenja karte razlikuje se između ljudskog i programskog igrača, čime je metoda abstraktna i namijenjena da bude implementirana u klasama koje nasljeđuju baznu klasu.

Metoda "PlayCard(CardSlot slot)" je zaslužna za odigravanje odabrane karte na polje proslijeđeno preko parametra. Metoda je abstraktna zbog istog razloga kao i metoda "DrawCard".

5.5.2. Klasa "PlayerHand"

Klasa "PlayerHand" je glavna klasa za ljudskog igrača. Ona sadrži atribute i metode koji su vezani za upravljanje kartama u ruci. Ljudski igrač mora imati mogućnost detaljnijeg pregleda opisa karte, otkazivanja odabrane karte, te listanje kartama u ruci. Programski upravljan igrač ne razmišlja na isti način kao i ljudski igrač, te za njega nije potrebno implementirati metode za pristupačnost.

```
public class PlayerHand : Hand
{
    public static PlayerHand instance;

    public override void DrawCard(Deck deck)
    {
        GameObject card;
        if (CardsInHand.Count < MAX_HAND_SIZE)
        {
            card = deck.Draw(0);

            if (card != null)
            {
                card.GetComponent<Card>().IsInHand = true;
                card.GetComponent<Card>().hand = this;
                StartCoroutine(card.GetComponent<Card>().Flip());
                card.GetComponent<RectTransform>().SetParent(gameObject.transform,
                    true);
                CardsInHand.Add(card);
            }
            else
            {
                PlayerHP.TakeDamage(emptyDeckDamage++);
            }
        }

    }
```

```
    private void Start()
    {
        instance = this;
    }
```

Statični atribut "instance" služi kao globalna referenca za instancu ruke ljudskog igrača, koja se postavlja prije početka igre preko metode "Start()". Time je moguće pristupiti toj instanci u bilo kojoj klasi koristeći "Playerhand.instance". Razlog tome je zbog pogodnosti, pošto će uvijek postojati samo jedna instanca ruke ljudskog igrača.

Metoda "DrawCard(Deck deck)" implementirana je iz bazne klase "Hand". Ako je broj karti u igračevoj ruci veća od dozvoljenog (konstanta "MAX_HAND_SIZE koja je jednaka 6), ne događa se ništa. Ako igraču ruka nije puna, pokušava se dohvatiti instanca karte koja se instancira iz špila pozivom metode "deck.Draw(0)". Ako u tom špilu nema više karata, metoda "deck.Draw(0) vraća vrijednost null, te igrač prima štetu jednaku vrijednosti atributa "EmptyDeckDamage", koja se zatim inkrementira za 1. Ako "deck.Draw(0)" vrati referencu instance karte, ona se pohrani u listu karta u ruci. Karte su okrenute leđima prema gore kada su prvo instancirane. Pozivom metode "card.Flip()" okreće se karta na vidljivu stranu.

```
public void InspectCard(Card card)
    {
        if (card.IsInspected) return;
        card.IsInspected = true;
        card.GetComponent<RectTransform>().position = card.PositionInHand + new
            Vector3(0, CARD_HOVER_DISTANCE, 0f);
        card.SetHandIndex();
        card.GetComponent<RectTransform>().transform.SetAsLastSibling();
    }

    public void CancelInspect(Card card)
    {
        if (card.IsSelected) return;
            card.IsInspected = false;
            card.GetComponent<RectTransform>().transform.SetSiblingIndex(card.
                GetHandIndex());
    }

    public void SelectCard(Card card)
    {
        if (ManaCounter.GetCurrentMana() >= card.ManaCost && SelectedCard == null &&
             BoardManager.instance.IsPlayerTurn())
        {
            card.IsSelected = true;
            SelectedCard = card;
            BoardManager.instance.SetPickingSlots(card);
        }
    }

    public void CancelSelect()
    {
        if (SelectedCard != null)
```

```
        {
            SelectedCard.IsSelected = false;
            CancelInspect(SelectedCard);
            SelectedCard = null;
            BoardManager.instance.CancelPickingSlots();
        }
    }
```

Metoda "InspectCard(Card card)" podiže proslijeđenu kartu iznad drugih, te ju postavlja kao zadnje dijete u hijerarhiji karta u ruci, što znaći da će biti prva iscrtana na ekranu. Ona služi kako bi igrač imao bolji pregled opisa pojedine karte. Metoda "CancelInspect()" vraća odabranu kartu na originalnu poziciju u ruci i hijerarhiji.

Metoda "SelectCard(Card card)" služi za odabiranje karte koju igrač namjerava odigrati. Ako igrač nema dovoljno resursa da odigra kartu ili on nije na potezu, poziv metode se rano poništava. U protivnom odabrana karta se prosljeđuje upravljaču igrače ploče kako bi se označila polja koja mogu biti odabrana u korist odigravanja karte. Metoda "CancelSelect()" poništava igračev odabir karte i vraća igraču ploču u prvobitno stanje.

```
public override IEnumerator PlayCard(CardSlot slot, Card card)
    {
        card.IsSelected = false;
        card.IsInHand = false;
        card.IsInspected = false;
        ManaCounter.ReduceMana(card.ManaCost);
        CardsInHand.Remove(card.gameObject);

        if (card is Minion)
        {
            Minion playedMinion = (Minion) card;
            SelectedCard = null;
            StartCoroutine(BoardManager.instance.PlaceMinion(playedMinion, slot));
        }
        else
        {
            Spell playedSpell = (Spell) card;
            SelectedCard = null;
            foreach (var effect in playedSpell.Effects)
            {
                if (playedSpell.IsTargeting) effect.Apply(slot.PlacedMinion);
                else effect.Apply();
            }
            BoardManager.instance.CancelPickingSlots();
            DiscardPile.ShuffleCardInDeck(playedSpell.Info);
            StartCoroutine(playedSpell.MoveToDiscardPile());
            while (playedSpell.IsTravelling) yield return null;
            Destroy(playedSpell.gameObject);
        }
        yield break;
    }

    private void Update()
```

```
    {
        if (Input.GetMouseButtonDown(1))
        {
            CancelSelect();
        }

        if (CardsInHand.Count != 0)
        {
            int n = CardsInHand.Count;
            for (int i = 0; i < n; i++)
            {
                if (CardsInHand[i].GetComponent<Card>().IsInHand && !CardsInHand[i].
                    GetComponent<Card>().IsInspected)
                {
                    Vector3 movement = new Vector3(Mathf.Lerp(CardsInHand[i].
                        GetComponent<RectTransform>().position.x, GetComponent<
                        RectTransform>().position.x + i * CARD_GAP, CARD_LERP_SPEED
                        * Time.deltaTime), Mathf.Lerp(CardsInHand[i].GetComponent<
                        RectTransform>().position.y, GetComponent<RectTransform>().
                        position.y, CARD_LERP_SPEED * Time.deltaTime));
                    CardsInHand[i].GetComponent<Card>().PositionInHand = movement;
                    CardsInHand[i].GetComponent<RectTransform>().position = movement
                        ;
                }
            }
        }
    }

}
```

Korutina "PlayCard(CardSlot slot)" služi za odigravanje odabrane karte. Prvo se resursi igrača smanjuju za iznos cijene karte, te se karta uklanja iz liste karta u ruci. Ako je odabrana karta sljedbenik, to jest klase tipa "Minion", ona se postavlja na polje proslijeđeno parametrom "slot". U protivnom, karta je čarolija, to jest klase tipa "Spell", što znači da se iterira kroz listu efekata karte i pozivaju se njihove metode, te se na kraju instanca karte uništi.

U metodi "Update()", koja se poziva svaki okvir, kalkulira se pozicija svake karte u ruci. Metoda konstantno iterira kroz listu karti u ruci i namiješta njihovu poziciju ovisno o broju karti u ruci. Razmak i brzina micanja karata određeni su preko konstanta "CARD_GAP" i "CARD_LERP_SPEED" respektivno. Osim toga, metoda "Update()" također sluša kada igrač pritisne desnu tipku miša, te pozove metodu "CancelSelect()" koja poništava trenutno odabranu kartu.

5.5.3. Klasa "EnemyAI"

Klasa "EnemyAI" dosta je slična implementaciji klase "PlayerHand", ali bez dodatnih metoda za pristupačnost, poput detaljnijeg pregleda karti ili poništavanja odabira karte. Umjesto toga sadrži metodu koja je zaslužna za odigravanje jednog cijelog poteza kada je programski igrač na redu.

```
public override void DrawCard(Deck deck)
    {
        GameObject card;
        if (CardsInHand.Count < MAX_HAND_SIZE)
        {
            card = deck.Draw(0);

            if (card != null)
            {
                card.GetComponent<Card>().IsInHand = true;
                card.GetComponent<Card>().hand = this;
                card.GetComponent<RectTransform>().SetParent(gameObject.transform,
                    true);
                CardsInHand.Add(card);
            }
            else
            {
                PlayerHP.TakeDamage(emptyDeckDamage++);
            }
        }
    }

    public override IEnumerator PlayCard(CardSlot slot, Card card)
    {
        ManaCounter.ReduceMana(card.ManaCost);
        card.IsInHand = false;
        CardsInHand.Remove(card.gameObject);
        if (card is Minion minion)
        {
            StartCoroutine(BoardManager.instance.PlaceMinion(minion, slot));
        }
        else
        {
            Spell spell = (Spell) card;
            foreach (var effect in spell.Effects)
            {
                if (spell.IsTargeting) effect.Apply(slot.PlacedMinion);
                else effect.Apply();
            }
            StartCoroutine(spell.MoveToDiscardPile());
            while (spell.IsTravelling) yield return null;
            yield return new WaitForSeconds(0.2f);
            StartCoroutine(spell.Flip());
            while (!spell.IsFlipped) yield return null;
            yield return new WaitForSeconds(1f);
            Destroy(spell.gameObject);
        }
        yield break;
    }

    private void Update()
    {
        if (CardsInHand.Count != 0)
```

```
    {
        int n = CardsInHand.Count;
        for (int i = 0; i < n; i++)
        {
            if (CardsInHand[i].GetComponent<Card>().IsInHand && !CardsInHand[i].
                GetComponent<Card>().IsInspected)
            {
                Vector3 movement = new Vector3(Mathf.Lerp(CardsInHand[i].
                    GetComponent<RectTransform>().position.x, GetComponent<
                    RectTransform>().position.x + i * CARD_GAP * -1,
                    CARD_LERP_SPEED * Time.deltaTime), Mathf.Lerp(CardsInHand[i
                    ].GetComponent<RectTransform>().position.y, GetComponent<
                    RectTransform>().position.y, CARD_LERP_SPEED * Time.
                    deltaTime));
                CardsInHand[i].GetComponent<Card>().PositionInHand = movement;
                CardsInHand[i].GetComponent<RectTransform>().position = movement
                    ;
            }
        }
    }
}
```

Metode "DrawCard(Deck deck)", "PlayCard(CardSlot slot, Card card)" i "Update()" vrlo su slično implementirane kao i u klasi "PlayerHand", samo s pojedinim manjim promjenama. U klasi "DrawCard(Deck deck)" izvučena karta se ne preokreće kako ljudski igrač ne bi vidio što je programski igrač dobio. U Metodi "PlayCard(CardSlot slot, Card card)", karte se preokreću kada su odigrane kako bi se pokazale ljudskom igraču. U metodi "Update()" ne sluša se korisnikov unos, te karte se redaju u suprotnom smjeru od karata igrača.

```
public IEnumerator PlayTurn()
{
    bool hasPlayableMinion = false;
    bool hasPlayableSpell = false;
    bool isPlayingMinion = true;
    bool hasMana = true;
    int lowestCost = ManaCounter.GetCurrentMana();

    List<CardSlot> slots = BoardManager.instance.GetPlayerSlots(false);
    while (hasMana && CardsInHand.Count > 0)
    {
        yield return new WaitForSeconds(0.6f);
        int index;
        List<CardSlot> occupiedSlots = new List<CardSlot>();
        List<CardSlot> emptySlots = new List<CardSlot>();
        foreach (CardSlot slot in slots)
        {
            if (slot.IsOccupied())
            {
                occupiedSlots.Add(slot);
            }
            else emptySlots.Add(slot);
        }
```

```
if (occupiedSlots.Count == 5)
{
    isPlayingMinion = false;
}

if (isPlayingMinion)
{
    index = Random.Range(0, emptySlots.Count);
    hasPlayableMinion = false;
    List<GameObject> cards = new List<GameObject>(CardsInHand);
    foreach (var card in cards)
    {
        Card script = card.GetComponent<Card>();
        if (script.ManaCost < lowestCost) lowestCost = script.ManaCost;
        if (script is Minion minion && script.ManaCost <= ManaCounter.
            GetCurrentMana())
        {
            StartCoroutine(PlayCard(emptySlots[index], minion));

            hasPlayableMinion = true;
            break;
        }
    }

    isPlayingMinion = hasPlayableMinion && occupiedSlots.Count != 5;
}

{
    List<CardSlot> enemySlots = BoardManager.instance.GetPlayerSlots(
        true);
    List<CardSlot> enemyOccupiedSlots = new List<CardSlot>();
    hasPlayableSpell = false;
    foreach (var slot in enemySlots)
    {
        if (slot.IsOccupied()) enemyOccupiedSlots.Add(slot);
    }
    List<GameObject> cards = new List<GameObject>(CardsInHand);
    foreach (var card in cards)
    {
        Card script = card.GetComponent<Card>();
        if (script.ManaCost < lowestCost) lowestCost = script.ManaCost;
        if (script is Spell spell && script.ManaCost <= ManaCounter.
            GetCurrentMana())
        {

            if (spell.TargetType == TargetType.FRIENDLY && occupiedSlots
                .Count > 0)
            {
                index = Random.Range(0, occupiedSlots.Count);
                StartCoroutine(PlayCard(occupiedSlots[index], spell));
                hasPlayableSpell = true;
            }
            else if (spell.TargetType == TargetType.ENEMY &&
```

```
                        enemyOccupiedSlots.Count > 0)
                    {
                        hasPlayableSpell = true;
                        StartCoroutine(PlayCard(enemyOccupiedSlots[Random.Range
                            (0, enemyOccupiedSlots.Count)], spell));
                    }
                    else if (enemyOccupiedSlots.Count > 0 && occupiedSlots.Count
                         > 0)
                    {
                        hasPlayableSpell = true;
                        if (enemySlots.Count > 0) StartCoroutine(PlayCard(
                            enemyOccupiedSlots[Random.Range(0,
                            enemyOccupiedSlots.Count)], spell));
                        else StartCoroutine(PlayCard(occupiedSlots[Random.Range
                            (0, occupiedSlots.Count)], spell));
                    }
                }
            }
        }

        if (lowestCost > ManaCounter.GetCurrentMana()) hasMana = false;
        if (!hasPlayableMinion && !hasPlayableSpell) break;
    }
    yield return new WaitForSeconds(1.5f);
    BoardManager.instance.ChangeTurn(false);
    yield break;
  }
}
```

Metoda "PlayTurn()" odgovorna je za odigravanje jednog cijelog poteza programskog igrača. Metoda prvo provjerava da li programski igrač ima dovoljno mjesta za odigrati sljedbenika. Ako nema, preskače se odabir karte sljedbenika, i kreće se izravno na odabir karte čarolije. Ako programski igrač ima dovoljno slobodnih polja, iterira se kroz njegove karte u ruci i provjerava da li igrač ima dovoljno resursa da odigra jednu od karti sljedbenika. Ako se takva karta ne nađe, ili ako igrač nema karte sljedbenika u ruci, preskače se na igranje karta čarolija.

Kod biranja karte čarolije, metoda prvo provjerava ako igrač ima kartu čarolije u ruci, te ako ima, da li ima dovoljno resursa da ju odigra. Ako se takva karta nađe, provjerava se da li postoje prigodne mete za odigravanje te čarolije. Primjerice, ako ljudski igrač nema sljedbenika na polju, programski igrač će preskočiti igranje čarolije koja utječe samo na sljedbenike ljudskog igrača.

Cijela metoda se ponavlja u petlji sve dok se ostvari barem jedan od sljedećih uvjeta:

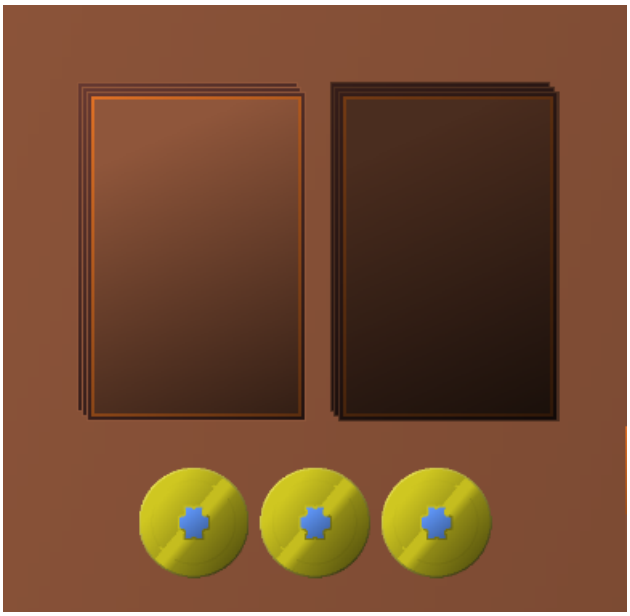
- Programski igrač više nema dovoljno resursa da odigra bilo koju kartu
- Programski igrač nema više karata u ruci
- Programski igrač ne može odigrati kartu zbog nedostatka primjerenih polja

Nakon petlje programski igrač daje potez ljudskom igraču. Metoda je zapravo napravljena kao korutina. Razlog tome je dodavanje kratkih pauzi između akcija programskog igrača

kako bi se održao primjeren tempo.

5.5.4. Predložak špila i klasa "Deck"

Svaki igrač ima u korist dva špila. Prvi je glavni špil, iz kojeg igrač obično izvlači. Drugi je odbacni špil ili groblje, iz kojeg igrač može izvlačiti jedino preko određenih efekata. Predložak špila sadrži vizualni element koji predstavlja špil, te skriptu tipa klase "Deck".



Slika 19: Glavni i odbacni špil iznad resursa igrača (Izvor: vlastiti rad)

Klasa "Deck" zaslužna je za funkcionalnosti oba špila (glavni i groblje). Glavne funkcionalnosti su instanciranje i izvlačenje karti, miješanje karte u špil, te miješanje cijelog špila.

```
public class Deck : MonoBehaviour
{
    [SerializeField]
    public List<CardInfo> Cards = new List<CardInfo>();
    [SerializeField]
    private GameObject MinionPrefab;
    [SerializeField]
    private GameObject SpellPrefab;
    [SerializeField]
    private Transform CardSpawnLocation;
```

Lista "Cards" sadrži reference skriptabilnih objekata podataka svih karti u špilu. Kada se karta izvlači, provjerava se da li je tip izvlačene karte klasa "MinionInfo" ili "SpellInfo", te ovisno o rezultatu instancira se odgovarajući predložak ("MinionPrefab" kao predložak sljedbenika, te "SpellPrefab" kao predložak čarolije). Atribut "CardSpawnLocation" označuje poziciju gdje se karta instancira, te je postavljena na vrh objekta špila.

```
public GameObject Draw(int index)
    {
        if (cards.Count > 0)
```

```
        {
            GameObject spawnedCard;
            CardInfo card = cards[index];
            cards.Remove(card);
            if (card is MinionInfo info)
            {
                spawnedCard = Instantiate(MinionPrefab, CardSpawnLocation.transform,
                    false);
                spawnedCard.GetComponent<Minion>().Info = info;
                spawnedCard.GetComponent<Minion>().LoadInfo();
            }
            else
            {
                spawnedCard = Instantiate(SpellPrefab, CardSpawnLocation.transform,
                    false);
                spawnedCard.GetComponent<Spell>().Info = (SpellInfo) card;
                spawnedCard.GetComponent<Spell>().LoadInfo();
            }
            return spawnedCard;
        }
        return null;
    }

    public void ShuffleCardInDeck(CardInfo card)
    {
        int index = Random.Range(0, cards.Count);
        cards.Insert(index, card);
    }

    private void Start()
    {
        if (cards.Count > 1) Shuffle();
    }

    public void Shuffle()
    {
        int count = cards.Count;
        if (count <= 1) return;
        for (int i = count - 1 ; i > 1; i--)
        {
            int rand = Random.Range(0, i+1);

            (cards[i], cards[rand]) = (cards[rand], cards[i]);
        }
    }
}
```

Metoda "Draw(int index)" uzima podatke o karti iz liste "Cards" ovisno o proslijeđenom indeksu, izbacuje navedeni skriptabilni objekt iz liste, te instancira odgovarajući predložak za vrstu karte. Metoda kao povratnu vrijednost vraća referencu na instancirani objekt karte, ili null, ako je špil prazan.

Metoda "ShuffleCardInDeck(Cardinfo card)" miješa skriptabilni objekt podataka karte

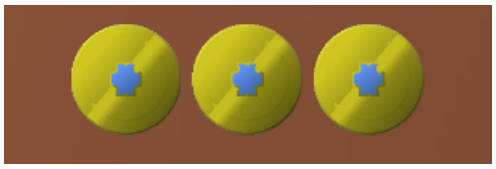
na nasumični indeks u špilu.

Metoda "Shuffle()" nasumično mijenja indekse karata u špilu, ako je broj karata u špilu veći od 1.

5.5.5. Novčići i klasa "ManaCounter"

Novčići predstavljaju glavni resurs koji je potreban za igranje karata. Svaka karta ima svoju cijenu, te ako igrač nema dovoljan broj novčića, ne može odigrati tu kartu. Novčići se vrate na maksimalnu vrijednost (3) na početku igračevog poteza. Predložak za novčiće sadrži glavni objekt "ManaCounter", te tri instance predloška novčića, koji su zapravo samo slike novčića.

Klasa "ManaCounter" sadrži glavne funkcionalnosti potrebne za punjenje i trošenje resursa. To su smanjivanje broja novčića, dodavanje novčića, te vraćanje trenutnog broja novčića.



Slika 20: Vizualni izgled novčića (Izvor: vlastita izrada)

```
public class ManaCounter : MonoBehaviour
{
    private const int MAX_MANA = 3;

    private int CurrentMana = MAX_MANA;

    [SerializeField]
    private List<GameObject> ManaCoins = new List<GameObject>();

    public bool ReduceMana(int amount)
    {
        if (amount > CurrentMana || amount <= 0) return false;
        else
        {
            int prevMana = CurrentMana;
            CurrentMana -= amount;
            for (int i = prevMana - 1; i >= CurrentMana; i--)
            {
                ManaCoins[i].SetActive(false);
            }
            return true;
        }
    }



    public bool RestoreMana(int amount)
    {
        if (amount > 3 || amount <= 0) return false;
        else
```

```
        {
            int prevMana = CurrentMana;
            CurrentMana = Mathf.Min(CurrentMana + amount, MAX_MANA);
            for (int i = prevMana; i < CurrentMana; i++)
            {
                ManaCoins[i].SetActive(true);
            }
            return true;
        }
    }

    public int GetCurrentMana()
    {
        return CurrentMana;
    }
}
```

Konstanta "MAX_MANA" predstavlja maksimalni broj novčića koji igrač može imati. Atribut "CurrentMana" predstavlja broj novčića koji igrač ima na raspolaganju u jednom trenutku igre.

Lista "ManaCoins" tipa "GameObject" sadrži reference objekata pojedinih novčića u svrhu skrivanja i prikazivanja istih ovisno o resursu koji pojedini igrač ima na raspolaganju.

Metode "ReduceMana(int amount)" i "RestoreMana(int amount)" smanjuju i povećavaju broj novčića jednog igrača, respektivno. Igrač ne može imati manje od 0, ili više od 3 novčića. U metodi se izračunava broj novčića koji se trebaju prikazati ili sakriti nakon promjene broja resursa.

Metoda "GetCurrentMana()" vraća broj novčića koji su igraču na raspolaganju u tom trenutku.

5.5.6. Životni bodovi igrača i klasa "PlayerHP"

Igrač može imati najviše 30 životnih bodova u jednom trenutku igre, i može ih izgubiti tijekom igre na više načina. Najčešće se gube preko napada protivnikovih sljedbenika ili pokušaja izvlačenja karte iz praznog špila.

Slika 21: Prikaz životnih bodova igrača (Izvor: vlastita izrada)

```
public class PlayerHP : MonoBehaviour
{
    [SerializeField]
    private bool IsPlayer;
```

```
    private const int MAX_HP = 30;

    private int CurrentHealth;

    [SerializeField]
    private TextMeshProUGUI HealthText;

    private void Start()
    {
        currentHealth = MAX_HP;
        UpdateText();
    }

    private void UpdateText()
    {
        healthText.text = currentHealth.ToString();
    }

    public void TakeDamage(int amount)
    {
        currentHealth -= amount;

        UpdateText();
        if (currentHealth <= 0)
        {
            BoardManager.instance.EndGame(IsPlayer);
        }
    }
}
```

Atribut "IsPlayer" postavlja se na vrijednost "true" ako instanca brojača životnih bodova pripada ljudskom igraču, a u protivnom slučaju je "false". Konstanta "MAX_HP" služi za ograničavanje maksimalnog broja života igrača, dok atribut "CurrentHealth" prati trenutnu vrijednost. Atribut "HealthText" služi za ažuriranje vizualnog brojača koji je vidljiv igraču.

Preko metode "Start()" ažurira se vrijednost trenutnih životnih bodova oba igrača na maksimalni, te se oba brojača vizualno ažuriraju koristeći metodu "UpdateText()". Metoda "TakeDamage(int amount) smanjuje životne bodove igrača za proslijeđeni iznos. Ako time igrač padne na 0 ili manje životnih bodova, poziva se metoda iz klase igrače ploče koja završava igru, te proglašava pobjednika ovisno o proslijeđenoj vrijednosti "IsPlayer".

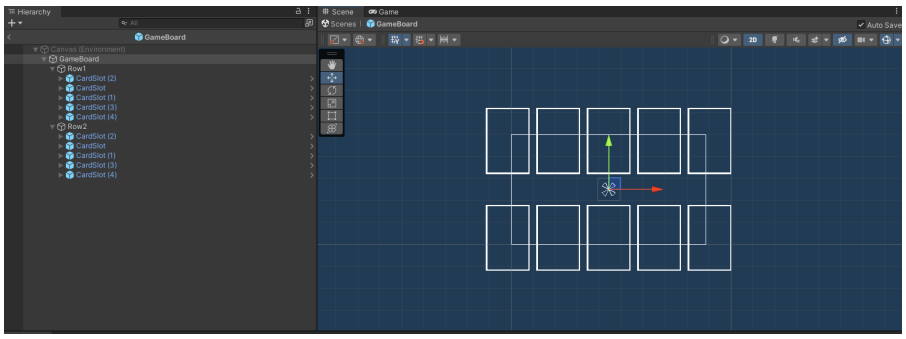
5.6. Igrača ploča i tijek igre

U ovoj sekciji obrađuje se kontrolni sustav tijeka i stanja igrače ploče na koju igrači utječu na način da postavljaju, uništavaju, i općenito utječu na sljedbenike preko čarolija ili ostalih efekata. Osim stanja igrače ploče, sustav igrače ploče također prati i poziva događaje koji ovise o tijeku igre, primjerice na kraju poteza jednog od igrača, ili kada je bilo koji sljedbenik uništen. Zadnja mehanika koja je vezana za igraču ploču je praćenje poteza i inicijaliziranje borbene faze. Na kraju svakog poteza, osim prvog, igrač koji je završio potez napada sljedbenike protivnika.

Za igraču ploču također su vezani polje za sljedbenike, pokazivač smjera napada te tipka za završavanje poteza, koji će također biti objašnjeni u ovoj sekciji.

5.6.1. Predložak igrače ploče

Predložak igrače ploče sastoji se od ukupno deset instanci predloška polja za sljedbenike, te su grupirani po pet ovisno o igraču kojem pripadaju. Glavni objekt igrače ploče sadrži skriptu klase "BoardManager" koja je zaslužna za glavne mehanike igrače ploče i tijeka igre. Na sljedećoj slici prikazan je predložak igrače ploče zajedno s njegovom hijerarhijom.



Slika 22: Predložak igrače ploče (Izvor: vlastita izrada)

5.6.2. Predložak polja za sljedbenike i klasa "CardSlot"

Predložak polja za sljedbenike dosta je jednostavan. On sadrži glavni objekt predloška koji nosi skriptu "CardSlot", te vizualni element polja. U nastavku slijedi klasa "CardSlot", koja sadrži glavne funkcionalnosti.

```
public class CardSlot : MonoBehaviour, IPointerClickHandler
{
    [SerializeField] private Sprite AvailableSlot;
    [SerializeField] private Sprite PickingSlot;

    [SerializeField] private Image CurrentSprite;

    public bool IsPickable;

    public bool IsPlayerSlot = false;
```

```
    public int ID;

    public Minion PlacedMinion = null;

    public bool IsOccupied()
    {
        return PlacedMinion != null;
    }

    public void SetPickingSlot()
    {
        CurrentSprite.sprite = PickingSlot;
        IsPickable = true;
    }

    public void SetDefaultSprite()
    {
        CurrentSprite.sprite = AvailableSlot;
        IsPickable = false;
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        if (BoardManager.instance.IsPlayerPicking && IsPickable)
        {
            StartCoroutine(PlayerHand.instance.PlayCard(this, PlayerHand.instance.
                SelectedCard));
        }
    }
}
```

Klasa "CardSlot" implementira sučelje "IPointerClickHandler", jer preko metode "OnPointerClick(PointerEventData eventData)" sluša kada igrač pritisne na polje kada ima odabranu kartu za odigrati. Atributi "AvailableSlot" i "PickingSlot" spremaju izgled polja kada je u neutralnom stanju, te kada ga je moguće odabrati za bacanje odabrane karte. Atribut "CurrentSprite" je referenca na trenutni izgled polja. Atribut "IsPickable" poprima vrijednost "true" kada stanje polja odgovara igračevoj trenutno odabranoj karti, a u protivnom poprima vrijednost "false". Atribut "IsPlayerSlot" postavlja se na "true" na svim poljima koji pripadaju ljudskom igraču, dok atribut "ID" služi za lakšu identifikaciju polja u metodama. Atribut "PlacedMinion" sprema referencu na sljedbenika koji je postavljen na tom polju.

Metoda "IsOccupied()" vraća vrijednost "true" ako se sljedbenik nalazi na tom polju, a u protivnom vraća vrijednost "false". Metoda "SetPickingSlot()" omogućuje da se polje odabra za odigranje karte, te mijenja izgled polja u odgovarajuću sliku. Metoda "SetDefaultSprite" vraća polje u obično stanje i također mijenja izgled polja na prvobitnu sliku.

5.6.3. Pokazivač smjera napada

Pokazivač smjera napada je vizualni element na sredini igrače ploče koji prikazuje u kojem će smjeru sljedbenici napadati, to jest koji igrač će započeti borbu kao napadač. Predložak je vrlo jednostavan, te sadrži pet slika mačeva koji prikazuju u odgovarajućem smjeru, te se programski okreću ovisno o potezu.

Slika 23: Pokazivač smjera napada (Izvor: vlastita izrada)

```
public class AttackDirectionController : MonoBehaviour
{
    private const float ROTATION_SPEED = 16f;

    [SerializeField]
    private List<Image> sprites;




    public void SwitchPointingDirection(bool playerTurn)
    {
        foreach (var sprite in sprites)
        {
            StartCoroutine(RotateSprite(sprite, playerTurn));
        }
    }

    public IEnumerator RotateSprite(Image sprite, bool playerTurn)
    {
        Quaternion rotation = Quaternion.identity;
        if (playerTurn)
        {
            while (sprite.GetComponent<RectTransform>().localRotation.eulerAngles.z
                < 176f)
            {
                rotation.eulerAngles = new Vector3(0, 0, Mathf.Lerp(sprite.
                    GetComponent<RectTransform>().localRotation.eulerAngles.z, 180f,
                     ROTATION_SPEED * Time.deltaTime));
                sprite.GetComponent<RectTransform>().localRotation = rotation;
                yield return null;
            }
            rotation.eulerAngles = new Vector3(0, 0, 180f);
            sprite.GetComponent<RectTransform>().localRotation = rotation;
        }
        else
        {
            while (sprite.GetComponent<RectTransform>().localRotation.eulerAngles.z
                < 356f)
```

```
            {
                rotation.eulerAngles = new Vector3(0, 0, Mathf.Lerp(sprite.
                    GetComponent<RectTransform>().localRotation.eulerAngles.z, 360f,
                     ROTATION_SPEED * Time.deltaTime));
                sprite.GetComponent<RectTransform>().localRotation = rotation;
                yield return null;
            }
            rotation.eulerAngles = new Vector3(0, 0, 0);
            sprite.GetComponent<RectTransform>().localRotation = rotation;
        }

        yield break;
    }
}
```

Konstanta "ROTATION_SPEED" određuje brzinu okretanja mačeva. Lista "sprites" tipa "Image" sprema reference od slika mačeva. Metoda "SwitchPointingDirection(bool playerTurn)" iterira kroz listu slika mačeva i rotira ih za 180$\deg usmjerusuprotnomodkazaljkenasatu, itoprekokorutine"Rot$

5.6.4. Klasa "BoardManager"

Klasa "BoardManager" upravlja cijelim tijekom igre, od početka do kraja, i brine se za sve vezano uz borbu između sljedbenika, tijek poteza, te prijavljivanje i odjavljivanje okidača efekata na događaje.

```
public class BoardManager : MonoBehaviour
{
    public static BoardManager instance;

    [SerializeField]
    private AttackDirectionController directionController;
    [SerializeField]
    private GameObject gameOverScreen;
    [SerializeField]
    private TextMeshProUGUI gameOverText;

    public PlayerHand PlayerHand;
    public EnemyAI EnemyHand;

    [SerializeField]
    private EndTurnButton endTurnButton;

    public UnityEvent<Minion> OnMinionAttack;
    public UnityEvent<Minion> OnMinionDied;
    public UnityEvent OnTurnStart;
    public UnityEvent OnTurnEnd;

    public int TurnNumber = 1;
    public bool IsPlayerPicking = false;

    public bool IsPlayerTurn()
    {
```

```
        return TurnNumber % 2 == 0;
    }

    [SerializeField] private List<CardSlot> playerSlots = new List<CardSlot>();
    [SerializeField] private List<CardSlot> enemySlots = new List<CardSlot>();
```

Statični atribut "instance" sadrži referencu na skriptu objekta igrače ploče koja je dostupna u bilo kojoj klasi. Postoji samo jedna instanca igrače ploče u igri, te funkcionalnosti klase "BoardManager" potrebne su u mnogim drugim klasama. Atribut "directionController" predstavlja referencu na prethodno spomenuti pokazivač smjera napada. Atributi "gameOverScreen" i "gameOverText" su reference na završni zaslon igre, koji se prikazuje kada igra završi. Atributi "PlayerHand" i "EnemyHand" sadrže reference za ruke ljudskog i programskog igrača. Atribut "endTurnButton" sprema referencu od tipke za prekidanje poteza od strane ljudskog igrača.

Atributi tipa "UnityEvent" su događaji vezani za tijek igre, te se na njih pretplaćivaju okidači efekata koji ovise o tome. Atribut "TurnNumber" prati broj trenutnog poteza, dok atribut "IsPlayerPicking" označava kada igrač bira polje za odabranu kartu. Metoda "IsPlayerTurn()" vraća pozitivnu vrijednost ako je trenutno potez ljudskog igrača, a u suprotnom vraća "false". Budući da programski igrač uvijek dobije prvi potez, svi parni potezi će uvijek biti od ljudskog igrača. Liste "playerSlots" i "enemySlots" sadrže reference za polja ljudskog i programskog igrača.

```
void Start()
    {
        instance = this;
        gameOverScreen.SetActive(false);

        for (int i = 0; i < playerSlots.Count; i++)
        {
            playerSlots[i].IsPlayerSlot = true;
            playerSlots[i].ID = i;
            enemySlots[i].ID = i;
        }
        directionController.SwitchPointingDirection(IsPlayerTurn());
        endTurnButton.ChangeButtonSprite();

        StartCoroutine(StartGame());
    }

    public IEnumerator StartGame()
    {
        PlayerHand.Deck.Shuffle();
        EnemyHand.Deck.Shuffle();

        for (int i = 0; i < 4; i++)
        {
            PlayerHand.DrawCard(PlayerHand.Deck);
            EnemyHand.DrawCard(EnemyHand.Deck);
            yield return new WaitForSeconds(0.3f);
        }
        EnemyHand.DrawCard(EnemyHand.Deck);
```

```
        yield return new WaitForSeconds(0.3f);

        StartCoroutine(EnemyHand.PlayTurn());
        yield break;

        public void ChangeTurn(bool isPlayer)
    {
        if (IsPlayerPicking && isPlayer != IsPlayerTurn()) return;
        endTurnButton.ChangeButtonSprite();
        directionController.SwitchPointingDirection(IsPlayerTurn());
        if (TurnNumber != 1) StartAttackPhase(IsPlayerTurn());
        OnTurnEnd.Invoke();
        TurnNumber++;
        OnTurnStart.Invoke();
        if (IsPlayerTurn())
        {
            PlayerHand.DrawCard(PlayerHand.Deck);
            PlayerHand.ManaCounter.RestoreMana(3);
        }
        else
        {
            EnemyHand.DrawCard(EnemyHand.Deck);
            EnemyHand.ManaCounter.RestoreMana(3);
            StartCoroutine(EnemyHand.PlayTurn());
        }
    }
    }
```

U metodi "Start()" prvo se isključuje vidljivost završnog zaslona, budući da je igra tek započela. Iteriranjem kroz svih polja igrača postavljaju se oni koji pripadaju ljudskom igraču, te se enumeriraju za lakšu usporedbu kasnije. Budući da programski igrač odrađuje prvi potez, postavlja se smjer pokazivača napada prema ljudskom igraču i onemogućuje se igraču mogućnost da pritisne tipku za završetak poteza. Na kraju se poziva korutina "StartGame()", koja priprema početak igre.

Korutina "StartGame()" započinje tijek igre, počevši s miješanjem špilova oba igrača. Svaki igrač izvlači po 4 karte, te budući da programski igrač započinje svoj potez, dobije dodatnu kartu. Na kraju metode poziva se korutina za odigravanje poteza programskog igrača.

Metoda "ChangeTurn(bool isPlayer)" služi za završavanje poteza trenutnog igrača, te započinjanje poteza suprotnog igrača. Igrač ne može završiti svoj potez ako on nije trenutno na potezu, ili ako ima odabranu kartu. Tipka za završavanje poteza mijenja oblik, te pokazivač smjera napada s okreće, ovisno koji igrač dobiva sljedeći potez. Ako trenutno nije prvi potez, započinje se borbena faza sljedbenika u smjeru igrača koji tek dobiva potez. Nakon borbene faze okidaju se događaji za kraj i početak poteza, jer je jedan igrač završio svoj potez, dok je drugi tek započeo svoj. Igraču koji dobiva sljedeći potez ispunjuju se novčići, te izvlači jednu kartu iz svog špila. Ako se radi o programskom igraču, također se poziva korutina za odigravanje njegovog poteza.

```
public void SetPickingSlots(Card card)
    {
```

```
        IsPlayerPicking = true;
        if (card is Minion)
        {
            foreach (var slot in playerSlots)
            {
                if (!slot.IsOccupied()) slot.SetPickingSlot();
            }
        }
        else
        {
            Spell spell = card as Spell;
            switch (spell.TargetType)
            {
                case TargetType.FRIENDLY:
                    foreach (var slot in playerSlots)
                    {
                        if (slot.IsOccupied()) slot.SetPickingSlot();
                    }
                    break;
                case TargetType.ENEMY:
                    foreach (var slot in enemySlots)
                    {
                        if (slot.IsOccupied()) slot.SetPickingSlot();
                    }
                    break;
                case TargetType.ALL:
                    for (int i = 0; i < playerSlots.Count; i++)
                    {
                        if (playerSlots[i].IsOccupied()) playerSlots[i].
                            SetPickingSlot();
                        if (enemySlots[i].IsOccupied()) enemySlots[i].SetPickingSlot
                            ();
                    }
                    break;
            }
        }
    }


    public void CancelPickingSlots()
    {
        IsPlayerPicking = false;
        for (int i = 0; i < playerSlots.Count; i++)
        {
            playerSlots[i].SetDefaultSprite();
            enemySlots[i].SetDefaultSprite();
        }
    }
```

Metoda "SetPickingSlots(Card card)" iterira kroz sva polja na ploči i omogućuje igraču da izabere one koji odgovaraju karti koja je proslijeđena u parametru "card", te postavlja vrijednost atributa "IsPlayerPicking" na "true", što znači da je ljudski igrač u tijeku biranja polja za

odigravanje karte. Druga metoda, "CancelPickingSlots()", vraća stanje svih polja u prvobitno stanje, te postavlja vrijednost "IsPlayerPicking" na "false".

```
public IEnumerator PlaceMinion(Minion minion, CardSlot slot)
    {
        slot.PlacedMinion = minion;
        minion.transform.SetParent(slot.transform, true);
        slot.PlacedMinion.PositionOnBoard = slot;
        minion.ToggleRaycastTarget(false);
        SubscribeMinionEvents(minion);
        StartCoroutine(slot.PlacedMinion.MoveToSlot());
        if (!slot.IsPlayerSlot)
        {
            while (slot.PlacedMinion != null && slot.PlacedMinion.IsTravelling)
                yield return null;
            if (slot.PlacedMinion != null) StartCoroutine(slot.PlacedMinion.Flip());
        }
        if (IsPlayerPicking) CancelPickingSlots();
    }

    public void SubscribeMinionEvents(Minion minion)
    {
        foreach (var effect in minion.Effects)
        {
            Debug.Log("SUBSCRIBING:␣" + effect + "␣|␣CONDITION:␣" + effect.Condition
                );
            switch (effect.Condition)
            {
                case EffectCondition.OnPlay:
                    effect.Apply();
                    break;
                case EffectCondition.OnDeath:
                    minion.OnDeath.AddListener(effect.Apply);
                    break;
                case EffectCondition.OnTurnStart:
                    OnTurnStart.AddListener(effect.Apply);
                    break;
                case EffectCondition.OnTurnEnd:
                    OnTurnEnd.AddListener(effect.Apply);
                    break;
                case EffectCondition.OnAttack:
                    minion.OnAttack.AddListener(effect.Apply);
                        break;
                case EffectCondition.OnMinionAttack:
                    OnMinionAttack.AddListener(effect.Apply);
                    break;
                case EffectCondition.OnMinionDied:
                    OnMinionDied.AddListener(effect.Apply);
                    break;
                case EffectCondition.OnTakingDamage:
                    minion.OnDamageTaken.AddListener(effect.Apply);
                    break;
```

```
            }
        }
    }

    public void UnsubscribeMinionEvents(Minion minion)
    {
        Debug.Log("Unsubscribing " + minion.Name);
        foreach (var effect in minion.Effects)
        {
            switch (effect.Condition)
            {
                case EffectCondition.OnDeath:
                    minion.OnDeath.RemoveListener(effect.Apply);
                    break;
                case EffectCondition.OnTurnStart:
                    OnTurnStart.RemoveListener(effect.Apply);
                    break;
                case EffectCondition.OnTurnEnd:
                    OnTurnEnd.RemoveListener(effect.Apply);
                    break;
                case EffectCondition.OnAttack:
                    minion.OnAttack.RemoveListener(effect.Apply);
                    break;
                case EffectCondition.OnMinionAttack:
                    OnMinionAttack.RemoveListener(effect.Apply);
                    break;
                case EffectCondition.OnMinionDied:
                    OnMinionDied.RemoveListener(effect.Apply);
                    break;
                case EffectCondition.OnTakingDamage:
                    minion.OnDamageTaken.RemoveListener(effect.Apply);
                    break;
            }
        }
    }
```

Korutina "PlaceMinion" služi za postavljanje proslijeđene karte sljedbenika na proslijeđeno polje. Koriste se metode iz klase "Minion" da se pozicionira karta na odgovarajuće polje. Kada se sljedbenik postavlja na polje, potrebno je spojiti okidače njegovih efekata na njihove odgovarajuće događaje. U tu svrhu poziva se metoda "SubscribeMinionEvents(Minion minion)", koja iterira kroz sve efekte proslijeđenog sljedbenika i registrira one kojima uvjet odgovara događaju. Metoda "UnsubscribeMinionEvents(Minion minion)" je gotovo identična, jedino što odjavljuje sve okidače efekata karte od događaja, jer nakon što je sljedbenik uništen, ne bi smio okidati svoje efektove, osim ako se radi o efektu s uvjetom uništenja tog sljedbenika.

```
public void StartAttackPhase(bool isAfterPlayerTurn)
    {
        for (int i = 0; i < playerSlots.Count; i++)
        {
            if (isAfterPlayerTurn) Combat(playerSlots[i], enemySlots[i]);
            else Combat(enemySlots[i], playerSlots[i]);
        }
```

```
}


public void Combat(CardSlot initiatorSlot, CardSlot defenderSlot)
{
    if (initiatorSlot.PlacedMinion == null) return;
    OnMinionAttack.Invoke(initiatorSlot.PlacedMinion);
    initiatorSlot.PlacedMinion.OnAttack.Invoke(defenderSlot.PlacedMinion);
    int initiatorAttack = initiatorSlot.PlacedMinion.CurrentAttack;
    if (defenderSlot.PlacedMinion != null)
    {
        int overkillDamage = 0;
        bool hasThorns = defenderSlot.PlacedMinion.Keyword == KeywordType.THORNS
            ;
        int defenderAttack = defenderSlot.PlacedMinion.CurrentAttack;
        bool isDefended = false;
        if (defenderSlot.PlacedMinion.Keyword != KeywordType.GUARDIAN)
        {
            foreach (var slot in GetAdjacentSlots(defenderSlot))
            {
                if (slot.PlacedMinion != null && slot.PlacedMinion.Keyword ==
                    KeywordType.GUARDIAN)
                {
                    overkillDamage = slot.PlacedMinion.CurrentHealth -
                        initiatorAttack;
                    slot.PlacedMinion.TakeDamage(initiatorAttack);
                    isDefended = true;
                    break;
                }
            }
        }

        if (!isDefended)
        {
            overkillDamage = defenderSlot.PlacedMinion.CurrentHealth -
                initiatorAttack;
            defenderSlot.PlacedMinion.TakeDamage(initiatorAttack);
        }

        if (initiatorSlot.PlacedMinion != null && initiatorSlot.PlacedMinion.
            Keyword == KeywordType.OVERPOWER && overkillDamage < 0)
        {
            overkillDamage = Mathf.Abs(overkillDamage);

            if (isDefended)
            {
                defenderSlot.PlacedMinion.TakeDamage(overkillDamage);
            }
            else
            {
                if (defenderSlot.IsPlayerSlot) PlayerHand.PlayerHP.TakeDamage(
                    overkillDamage);
```

```
                else EnemyHand.PlayerHP.TakeDamage(overkillDamage);
            }
        }

        if (initiatorSlot.PlacedMinion != null && hasThorns)
        {
            initiatorSlot.PlacedMinion.TakeDamage(defenderAttack);
        }
    }
    else
    {
        if (initiatorSlot.IsPlayerSlot) EnemyHand.PlayerHP.TakeDamage(
            initiatorAttack);
        else PlayerHand.PlayerHP.TakeDamage(initiatorAttack);
    }
}
```

Metoda "StartAttackPhase(bool isAfterPlayerTurn)" pokreće fazu napada sljedbenika, ovisno o igraču koji je upravo završio potez. Metoda iterira kroz polja igrača s lijeva na desno i poziva metodu "Combat(CardSlot initiatorSlot, CardSlot defenderSlot)", gdje parametar ""initiatorSlot" predstavlja polje igrača napadača, dok parametar "defenderSlot" predstavlja polje igrača koji se brani.

Cijeli postupak između dva suprotna sljedbenika, to jest polja, odvija se preko metode "Combat(CardSlot initiatorSlot, CardSlot defenderSlot)". Na početku se provjerava ako je postavljen sljedbenik na polju napadača. Ako nije, trenutni par polja se preskače. Kada se nađe polje napadača koje sadrži sljedbenika, provjerava se ako na suprotnom polju postoji sljedbenik. Ako ne postoji, igrač branitelj prima štetu jednaku vrijednosti napada sljedbenika. U protivnom, prvo se provjerava da li sljedbenik na suprotnom polju ima ključnu riječ "Thorns", kako bi se odredilo hoće li sljedbenik napadač kasnije primiti štetu od sljedbenika branitelja. Sljedbenik branitelj nije uvijek onaj koji prima štetu od sljedbenika napadača. Prije dijeljenja štete, provjeravaju se lijevo i desno polje pokraj sljedbenika branitelja. Ako jedno od tih polja sadrži sljedbenika s ključnom riječi "Guardian", on će umjesto njega primiti štetu. U slučaju da se s obje strane sljedbenika nalaze sljedbenici s ključnom riječi "Guardian", uzima se u obzir lijevi sljedbenik. Također, ako sljedbenik branitelj ima ključnu riječ "Guardian", ne može biti zaštićen od drugog sljedbenika. Nakon što odgovarajući sljedbenik igrača branitelja primi štetu, provjerava se da li sljedbenik napadač ima ključnu riječ "Overpower". Ako ima, te ako je sljedbenik koji prima štetu snižen ispod 0 životnih bodova, šteta koja predstavlja višak nanosi se na originalnog sljedbenika koji je trebao primiti štetu u slučaju da da je taj bio zaštićen, ili izravno na igrača branitelja. Na kraju se nanosi šteta na sljedbenika napadača, u slučaju da sljedbenik branitelj ima ključnu riječ "Thorns".

```
public CardSlot GetCardSlot(bool isPlayerSlot, int index)
    {
        if (isPlayerSlot) return playerSlots[index];
        else return enemySlots[index];
    }

    public List<CardSlot> GetPlayerSlots(bool isPlayerSlots)
```

```
{
    if (isPlayerSlots) return playerSlots;
    else return enemySlots;
}

public List<CardSlot> GetAdjacentSlots(CardSlot slot)
{
    List<CardSlot> slots = new List<CardSlot>();
    if (slot.IsPlayerSlot)
    {
        for (int i = 0; i < playerSlots.Count; i++)
        {
            if (playerSlots[i].ID == slot.ID)
            {
                if (i-1 >= 0) slots.Add(playerSlots[i-1]);
                if (i + 1 < playerSlots.Count) slots.Add(playerSlots[i + 1]);
                break;
            }
        }
    }
    else
    {
        for (int i = 0; i < enemySlots.Count; i++)
        {
            if (enemySlots[i].ID == slot.ID)
            {
                if (i - 1 >= 0) slots.Add(enemySlots[i - 1]);
                if (i + 1 < enemySlots.Count) slots.Add(enemySlots[i + 1]);
                break;
            }
        }

    }
    return slots;
}

public CardSlot GetOpposingSlot(CardSlot slot)
{
    if (slot.IsPlayerSlot)
    {
        for (int i = 0; i < playerSlots.Count; i++)
        {
            if (playerSlots[i].ID == slot.ID) return enemySlots[i];
        }
        return null;
    }
    else
    {
        for (int i = 0; i < enemySlots.Count; i++)
        {
            if (enemySlots[i].ID == slot.ID) return playerSlots[i];
        }
        return null;
```

```
        }
    }
```

Navedeno su pomoćne metode koje vraćaju reference na polja ovisno o potrebi. Metoda "GetCardSlot(bool isPlayerSlot, int index)" vraća polje proslijeđenog igrača koje sadrži proslijeđeni identifikator. Metoda "GetPlayerSlots(bool isPlayerSlots)" vraća sva polja koja pripadaju odgovarajućem igraču. Metoda "GetAdjacentSlots(CardSlot slot)" vraća listu polja koja se nalaze na lijevoj i desnoj strani proslijeđenog polja. Veličina liste je 1 ili 2 elementa, ovisno o poziciji polja. Metoda "GetOpposingSlot(CardSlot slot)" vraća polje koje se nalazi suprotno od proslijeđenog polja. To polje uvijek će pripadati protivničkom igraču.

```
    public void EndGame(bool hasPlayerLost)
    {
        Destroy(PlayerHand.gameObject);
        Destroy(EnemyHand.gameObject);
        Destroy(directionController.gameObject);
        gameOverScreen.SetActive(true);
        if (hasPlayerLost)
        {
            gameOverText.text = "Defeat!";
        }
        else gameOverText.text = "Victory!";

        Destroy(gameObject);
    }
}
```

Metoda "EndGame(bool hasPlayerLost)" zaslužna je za završavanje igre, kao i završavanje ovog poglavlja. Ona uništava instance ruka oba igrača kako ne bi odrađivali poteze nakon završetka igre, te prikazuje završni zaslon na kojem će pisati "Victory!" ako je ljudski igrač pobjedio, odnosno "Defeat!" ako je izgubio. Igru je moguće ponovno pokrenuti na završnom zaslonu preko tipke "Restart".

6. Zaključak

Tema ovog rada bila je razvoj prototipa kartaške videoigre u programskom alatu Unity. Kroz cijeli postupak razvoja upoznao sam se s načinima kako se sve može koristiti objektno orijentirano programiranje u izradi kartaške videoigre. Žanr kolekcionarskih kartaških igara bogat je prilikama za kreativnost kod izrade karti, pravila igre, posebnih efekata i sl. Osim objektno orijentiranog programiranja, postupkom razvoja videoigre naučio sam mnogo o izradi sustava koji međusobno komuniciraju. Projekt videoigre sadrži preko 17 skripti koje u nekom obliku predstavljaju zasebni sustav. Dodatna prednost kod toga je dodatno iskustvo rada u alatu Visual Studio, kojeg sam koristio kao integrirano razvojno okruženje (eng. *integrated development environment*, kraće IDE), te malog unaprijeđivanja vještine crtanja i vizualnog dizajna koristeći alat Aseprite i Figma.

Rezultat rada je prototip kartaške videoigre s 14 različitih karti, 2 vrste karata, razni posebni efekti, te programski upravljan protivnik. Prototip je razvijen na način da se lagano može proširiti s dodatnim funkcionalnostima, kartama, vrstama karti i sl. Iako nije savršen i postoje manji problemi, mogu reći da je bio izvrsni programski izazov.

Popis literature

[1] Wikipedia contributors, *Unity (game engine) — Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Unity_(game_engine)&oldid=1171307860, (Online; pristupljeno 27.8.2023), 2023.

[2] Unity, *Pricing*, https://unity.com/pricing, (Online, pristupljeno 27.8.2023).

[3] H. Koranne, *History of Unity3D*, https://www.linkedin.com/pulse/history-unity3d-harsh-koranne, 2021 (Online, pristupljeno 27.8.2023).

[4] D. Takahashi, *Unity files for IPO, reveals $163 million loss for 2019 and 1.5 million monthly users*, https://venturebeat.com/business/unity-files-for-ipo-reveals-163-million-loss-for-2019-and-1-5-million-monthly-users/, 2020, (Online, pristupljeno 27.8.2023).

[5] J. Drake, *19 Great Games That Use The Unity Game Engine*, https://www.thegamer.com/unity-game-engine-great-games/, 2023, (Online, pristupljeno 27.8.2023).

[6] Unity technologies, *Unity's interface*, https://docs.unity3d.com/Manual/UsingTheEditor.html, 2022 (pristupljeno 27.8.2023).

[7] Unity technologies, *Scenes*, https://docs.unity3d.com/Manual/CreatingScenes.html, 2022 (Online, pristupljeno 28.8.2023).

[8] Unity technologies, *GameObjects*, https://docs.unity3d.com/Manual/GameObjects.html, 2022 (Online, pristupljeno 28.8.2023).

[9] Unity technologies, *ScriptableObject*, https://docs.unity3d.com/Manual/class-ScriptableObject.html, (Online, Pristupljeno 30.8.2023).

[10] Unity technologies, *Introduction to components*, https://docs.unity3d.com/Manual/Components.html, 2022 (Online, pristupljeno 29.8.2023).

[11] Unity technologies, *Use components*, https://docs.unity3d.com/Manual/UsingComponents.html, 2022 (Online, pristupljeno 29.8.2023).

[12] Unity technologies, *Canvas*, https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html, (Online, Pristupljeno 29.8.2023).

[13] Unity technologies, *Rect Transform*, https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-RectTransform.html, (Online, Pristupljeno 29.8.2023).

[14] S. J. Johansson, „What Makes Online Collectible Card Games Fun to Play?," siječanj 2009.

[15] S. Adinolf i S. Turkay, „Collection, creation and community: a discussion on collectible card games," lipanj 2011., str. 3–11.

[16] N. Huffman, *Preview of the W. Duke, Sons Co. Digital Collection*, https://blogs.library.duke.edu/bitstreams/2014/10/24/collection-preview-w-duke-sons-co-cigarette-cards/, 2014 (Online, pristupljeno 2.9.2023).

[17] J. Edwards, *He found a rare Magic: The Gathering card. Post Malone paid $2M for it.* https://www.washingtonpost.com/nation/2023/08/17/post-malone-magic-card-one-ring/, 2023, (Online, pristupljeno 2.9.2023).

[18] A. Chalk, *Blizzard celebrates 100 million Hearthstone players with free card packs for everyone*, https://www.pcgamer.com/blizzard-celebrates-100-million-hearthstone-players-with-free-card-packs-for-everyone/, 2018 (Online, pristupljeno 3.9.2023).

[19] P. Kollar, *Hearthstone: Heroes of Warcraft review: Heart of the Cards*, https://www.polygon.com/2014/4/25/5654196/hearthstone-heroes-of-warcraft-review, 2014 (Online, pristupljeno 3.9.2023).

[20] M. Phill, *Inscryption review – a deck-building captivating and atmospheric masterpiece*, https://gamingtrend.com/feature/reviews/inscryption-review-a-deck-building-captivating-and-atmospheric-masterpiece/, 2023 (Online, pristupljeno 4.9.2023).

[21] Unity technologies, *Coroutine*, https://docs.unity3d.com/ScriptReference/Coroutine.html, (Online, Pristupljeno 4.9.2023).

Popis slika