OAMK
OULUN AMMATTIKORKEAKOULU

Joonas Ylläsjärvi

KERÄILYKORTTIPELI

KERÄILYKORTTIPELI

Joonas Ylläsjärvi
Opinnäytetyö
Syksy 2016
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikkan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä(t): Joonas Ylläsjärvi
Opinnäytetyön nimi: Keräilykorttipeli
Työn ohjaaja(t): Lasse Haverinen
Työn valmistumislukukausi ja -vuosi: Syksy 2016
Sivumäärä: 37

Työssä aiheena oli Unity Enginellä toteutettu keräilykorttipeli. Projektissa tavoitteena oli luoda toimiva vuoropohjainen korttipeli ajatellen myös sen mahdollisuuksia Unity Asset Storessa myytävälle assetille. Aiheeseen päädyttiin omasta mielenkiinnosta pelien tekemiseen, korttipeleihin ja niiden toimintaan.

Työssä käytettiin Unityn pelimoottoria ja C#-ohjelmointikieltä, joila pelin kaikki toiminnot luotiin itse. Korteille pyrittiin luomaan toisistaan poikkeavia toiminallisuuksia ja erikoisominaisuuksia perusominaisuuksien lisäksi. Pelin moninpeliosuus päätettiin toteuttaa rakentamalla peli samalla koneella pelattavaksi vuoropohjaiseksi peliksi.

Lopputuloksena työssä syntyi toimiva vuoropohjainen peli. Projektin aikana saatiin laajempi kuva siitä, miten korttipelit rakentuvat. Työssä oli haastetta pyrittäessä siihen, että ulkopuolisella olisi helppoa muokata peliä haluamakseen. Lisäksi huomattiin, kuinka monia ehtoja tulee ottaa huomioon ja kuinka paljon sisältöä ohjelmassa on luodessa päällepäin yksinkertaista peliä.

Asiasanat: Unity, korttipeli, pelinkehitys, ohjelmointi

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software development

Author(s): Joonas Ylläsjärvi
Title of thesis: Collectible Card Game
Supervisor(s): Lasse Haverinen
Term and year when the thesis was submitted: Autumn 2016
Number of pages: 37

The subject of this thesis was Collectible Card Game made with Unity Engine. Aim of the project was to create turn-based card game with possibilities selling at Unity Asset Store. Subject came from own interest in making games and playing card games.

The game was made with C#-programming language and where all functions were self-made. For cards there was aim to make different features in addition to the basic functions. Multiplayer part in the game is accomplished by making it local turn-based.

The result was functional game. During the project, we got wider view of how card games were built. The project had challenges at creating easily customizable game mechanics for different preferences. In addition it was found how many conditions must be taken into account and how much of the content of the program can be in simple game.

Keywords: Unity, card game, collectible card game, game development, programming

SISÄLLYS

TIIVISTELMÄ 3
ABSTRACT 4
SISÄLLYS 5
SANASTO 7
1 JOHDANTO 9
2 PELIN KOKONAISUUS 10
2.1 Digitaaliset keräilykorttipelit 10
2.2 Unity Engine 10
2.3 Asset Store 11
2.4 Muut käytetyt työkalut 12
2.5 Pelin tavoite ja ominaisuudet 12
3 TYÖN TOTEUTUS 18
3.1 Ohjelman rakenne 18
3.2 Kortin nostaminen korttipakasta 21
3.3 Kortin pelaaminen 21
3.4 Ehtolauseet 22
3.5 Uusien korttien luominen 23
3.6 Korttipakan muuttaminen 24
3.7 Toteutuksen eri vaiheet 24
3.7.1 Suunnittelu 25
3.7.2 Objektien sijainnit 25
3.7.3 Objektien luominen listasta ja korttipakan toiminta 25
3.7.4 Objektien siirtäminen käteen sekä kentälle 26
3.7.5 Kortin pelaaminen 26
3.7.6 OnMouseOver 26
3.7.7 BoxCollider 27
3.7.8 Korttien ominaisuudet ja vaikutukset 27
3.7.9 Elämäpisteet, vahinko ja kortin hinta 28
3.7.10 Kortin arpominen 30

3.7.11 Loitsukortti 31
3.7.12 Kaksi pelaajaa ja vuorot 31
4 TULOKSET JA JOHTOPÄÄTÖKSET 33
4.1 Paranneltavia ominaisuuksia 33
4.2 Lisättäviä ominaisuuksia 33
4.3 Asset Storessa myynti 34
5 YHTEENVETO 35
LÄHTEET 37

SANASTO

Asset	Voidaan sanoa paketiksi, joka voi sisältää mm. ohjelmakoodia tai grafiikkaa
BoxCollider	Laatikon muotoinen objekti, jonka törmäystä tarkastellaan muihin objekteihin nähden
Card Draw	Kortin nosto pelaajan käteen
Charge	Kortin ominaisuus antaa mahdollisuuden hyökätä välittömästi
Collectible Card Game (CCG)	Keräilykorttipeli
GameObject	Peliobjekti, pitää sisällään mm. skriptejä, transformeja, grafiikkaa
Mana	Energia tai hinta, joka pelaan täytyy maksaa, kun hän haluaa pelata kortin
Minion	Pelaajan kentälle pelaama taistelija, jolla voidaan hyökätä
Multiple target spell	Useisiin kohteisiin vaikuttava loitsu
Prefab	Peliobjektista tiedosto, johon voidaan määrittää ennältä määrättyjä arvoja
Scene	Peliin ladataan yksi näkymä kerrallaan, johon sisältyvät kaikki peliobjektit

Shader	Skriptitiedosto, joka määrittelee kuvan ominaisuuksia
Single target spell	Yhteen kohteeseen vaikuttava loitsu
Taunt	Kortin ominaisuus, jolloin kortti täytyy tuhota ennen kuin pelaajaan, tai muihin kortteihin voi hyökätä
TextMesh	Alue, johon voidaan määritellä tekstiä
Transform	Tämä kertoo peliobjektin sijainnin, rotaation ja mittakaavan

1 JOHDANTO

Erilaisten korttipelien digitaaliset versiot ja näistä erilaiset muunnokset ovat lisääntyneet viime vuosien varrella ja oman kiinnostuksen myötä halusin myös itse toteuttaa vastaavanlaisen pelin. Korttipeleissä tulee ottaa huomioon useita ehtoja sekä sitä, miten kortit ja peli itsessään toimivat. Näistä käytetty nimitys ”collectible card game” eli keräilykorttipeli sisältää nimensä mukaisesti korttien keräilyä ja näillä pelaamista, yleensä muita pelaajia vastaan.

Työn tarkoituksena oli luoda pohja paketille, josta jatkokehityksellä saadaan myyntikelpoinen. Toiminnallisuuksista peliin on toteutettu yleisimmät pelimekaniikat sekä korttien eri vaikutukset. Ohjelmointi on pyritty tekemään huomioimalla pelin helppo muokkaus. Työssä käytettiin C#-ohjelmointikieltä ja pelimoottorina Unityä, joka tarjoaa myös assettien myymiseen soveltuvan ympäristön.

Tässä raportissa tarkastellaan Unity-pelimoottorilla tehtyä demoa korttipelistä sekä siihen kuuluvia eri vaiheita ja rakennetta. Raportissa myös pohditaan ja käydään läpi, mitä ominaisuuksia tulisi ohjelman sisältää, jotta julkaisu Unity Asset Storessa olisi mahdollinen.

Työn tavoitteena oli tehdä toimiva demoversio korttipelistä. Pelissä tulisi ilmetä korttipelin pääpiirteet, joita ovat korteilla pelaaminen sekä korttien vaikutus. Pelissä keskityttiin lähinnä toiminnallisuuteen ohjelmoinnin kanssa. Kaikki toiminnot olivat siten itse toteutettavana, koska tavoitteena oli luoda peli alusta alkaen. Pelin tulee myös hoitaa se, etteivät pelaajat voi tehdä sääntöjen vastaisia toimintoja.

2 PELIN KOKONAISUUS

Luvussa kerrotaan yleisesti peliin liittyvistä asioista. Käytetyistä työkaluista kerrotaan, mitä on tarvittu projektin toteuttamiseen. Kappaleessa on myös selitettynä lyhyesti pelin rakenteesta.

2.1 Digitaaliset keräilykorttipelit

Keräilykorttipeleistä hyvinä esimerkkeinä voi pitää mm. pelejä Hearthstone, Magic: The Gathering Game ja The Elder Scrolls: Legends. Nämä pelit ovat toisistaan poikkeavia, kun niitä tarkastellaan lähemmin. Pääasiassa idea peleissä on kuitenkin sama: kerätään hyviä kortteja, joilla pelataan ja voitetaan muita pelaajia. Varsinaisesti peli alkaa, kun tarvittavat kortit ovat kerättyinä ja niistä rakennetaan eri korttien yhdistelmiä.

Peleissä korteilla on erilaisia vaikutuksia ja omaisuuksia. Digitaalisissa korttipeleissä pelaaminen tapahtuu nopeammin verrattuna fyysisiin versioihin, koska suurimman osan laskemisesta hoitaa pelin ohjelma. Myös sääntöjen vastaisia toimintoja ei pääse tapahtumaan, sillä ohjelma tarkistaa ne.

2.2 Unity Engine

Unity on Unity Technologiesin kehittämä ja ylläpitämä pelimoottori. Rekisteröityneitä käyttäjiä sillä on 5,5 miljoonaa ympäri maailman (Unity Technologies 2016a, viitattu 22.11.2016). Unity mahdollistaa 2D- ja 3D-pelien kehittämisen mm. tietokoneelle (Windows, Mac, Linux), konsoleille, mobiililaitteille sekä selaimelle. Ohjelmointi tapahtuu C#-, Javascript- sekä Boo-ohjelmointikielillä. Pelejä, joita Unityllä on toteutettu, ovat mm. Hearthstone, Firewatch, Ori and the Blind Forest, Angry Birds 2 ja Kerbal Space Program. (Unity Technologies 2016b, viitattu 17.11.2016.)

Pelimoottori on ilmainen käyttää, kunnes pelillä ansaitaan yli 100 000 dollaria. Tämän jälkeen käyttäjän tulee ostaa Unity Plus, jolla rajaa voidaan nostaa 200 000:een saakka. Mahdollisuutena on myös ostaa Unity Pro, jossa ei ole

ansaintarajaa. Maksulliset versiot tuovat ohjelmaan erinäisiä lisäominaisuuksia. (Unity Technologies 2016c, viitattu 22.11.2016.)

2.3 Asset Store

Unityn sivuilta löytyy sen kehittämä Asset Store, jossa myydään yksittäisten ihmisten tai ryhmien tekemiä assetteja Unityyn. Ostajina Asset Storessa ovat niin suuret yritykset kuin omia projektejaan tekevät yksityiset henkilöt. Unity Asset Storeen voi kuka tahansa lisätä omatekoisen assetin.

Hinnat asseteissa vaihtelevat nollasta eurosta useisiin satoihin euroihin. Onkin siis tärkeää hinnoitella tuote sen mukaan, kenelle tuote on tarkoitettu. Hinnoittelussa kannattaa myös huomioida paketin laajuus ja sen vaikutus vaikuttaa hintaan. Monet asseteista ovat vielä kehityksen alla, ja niihin saadaan viikoittain lisäsisältöä ja parannuksia. (Unity Technologies 2016d, viitattu 15.11.2016.)

Asset Storessa myymiseen tarvitsee luoda Unityyn käyttäjätunnukset ja tämän jälkeen sinne tehdään Asset Store -julkaisijatunnukset. Demosta tai muusta sisällöstä tehdään paketti, jota voidaan hallita Asset Store -työkaluilla, jolla voidaan esimerkiksi valita, mitä halutaan lähettää jaettavaksi. Tämän jälkeen paketti lähetetään julkaisijatunnuksien kautta tarkastettavaksi Asset Storelle. (Unity Technologies 2016e, viitattu 15.11.2016.) Assettien myynnissä Unitylle jää 30 prosenttia myynnistä ja kehittäjälle loput 70 prosenttia sekä maksettavaksi mahdolliset verot ja pankkikulut (Unity Technologies 2016f, viitattu 15.11.2016). Asseteja julkaistessa on suositeltavaa tarkistaa tarkemmat käyttäjäehdot Unityn omilta sivuilta.

Ohjelmassa on käytetty Unityn kehittämiä ilmaisia assetteja, jotka voidaan pelinäkymästä nähdä. Assetteihin kuuluvat graafiset maisemat, jotka luovat peliin näyttävyyttä. Assetit eivät vaikuta ohjelman toimintaan ja ohjelma toimii siten myös ilman Unityn graafisia assetteja.

2.4 Muut käytetyt työkalut

Työssä on Unityn lisäksi käytetty seuraavia ohjelmia: Lucidchart, GIMP ja Microsoft Visual Studio. Ohjelmat eivät kuitenkaan ole luodun ohjelman kannalta oleellisia ohjelmia. Näiden ohjelmien tilalle voidaan valita muu ohjelma, joka sisältää vastaavanlaiset perusominaisuudet eli grafiikan luonnin sekä skriptien kirjoittamisen.

Lucidchart on verkkopohjainen diagrammien luontityökalu, jonka on kehittänyt Lucid Software. Ohjelmasta on saatavilla ilmaisversio, ja maksullisella versiolla saadaan esimerkiksi monimutkaisempia diagrammeja ja tallennustilaa. Tätä ohjelmaa käyttämällä on luotu raportissa nähtävät UML-kaaviot. (Lucid Software Inc. 2016, viitattu 22.11.2016.)

GIMP on The GIMP Teamin kehittämä ilmainen kuvankäsittelyohjelma. Ohjelma mahdollisti tarpeelliset kuvan muokkausominaisuudet. Ohjelmaa käytettiin pelissä näkyvien korttien luomiseen. Myös pelaajien sankareiden grafiikat on toteutettu kyseisellä ohjelmalla. (The GIMP Team 2016, viitattu 22.11.2016.)

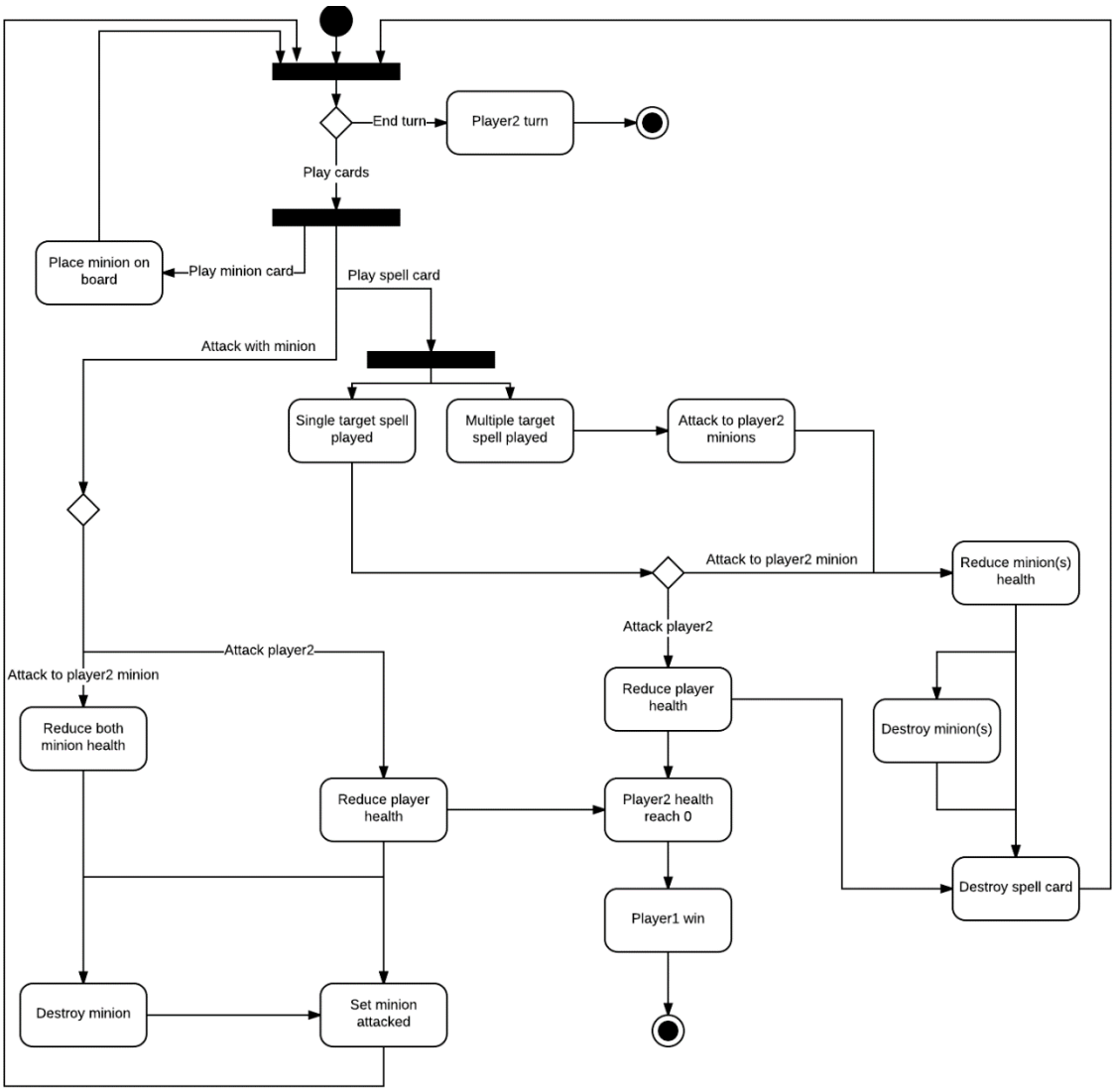
Microsoftin kehittämä Visual Studio on ohjelmankehitysympäristö, joka tukee useita eri ohjelmointikieliä. Ohjelma tarjoaa Unityn ympäristöön hyvän syntaksin, eli se osaa kertoa oikean kieliopin ohjelmointikielen rakenteeseen. Tästä on käytetty ilmaisversiota, joka on tarkoitettu Unityn kanssa käytettäväksi. (Microsoft 2016, viitattu 22.11.2016.)

2.5 Pelin tavoite ja ominaisuudet

Peli etenee vuoropohjaisesti lokaalilla moninpelillä. Pelaaja voi pelata kortteja kädestään haluamansa määrän, jota kuitenkin rajoittaa niiden hinta. Jokaisellla kortilla on hinta, jota kuvataan termillä ”mana”. Kentällä olevilla minion-korteilla eli pelaajien ohjaamilla taistelijakorteilla voi pelaaja hyökätä vastustajan minioneihin tai itse sankariin eli pelaajaan. Kun pelaaja on tehnyt haluamansa korttien siirrot ja pelaamiset, päättää hän tämän jälkeen vuoronsa ”End Turn”-painikkeesta. Jokaisen vuoron alussa pelaajan manat täyttyvät, hän saa yhden

manan lisää sekä nostaa korttipakastansa yhden kortin. Pelin voittaa pelaaja, joka saa toisen pelaajan elämäpisteet loppumaan.

Pelin eteminen ja toiminta voidaan havainnollistaa myös UML-kaaviosta (kuva 1), jossa jokainen vaihe on merkittynä tapahtumasarjaksi. Kuvaajasta myös ilmenee, että pelissä ei ole aikarajaa. Kaavioon on merkittynä pelaajan tekemät valinnat timantilla ja loput on automatisoitua toimintaa.

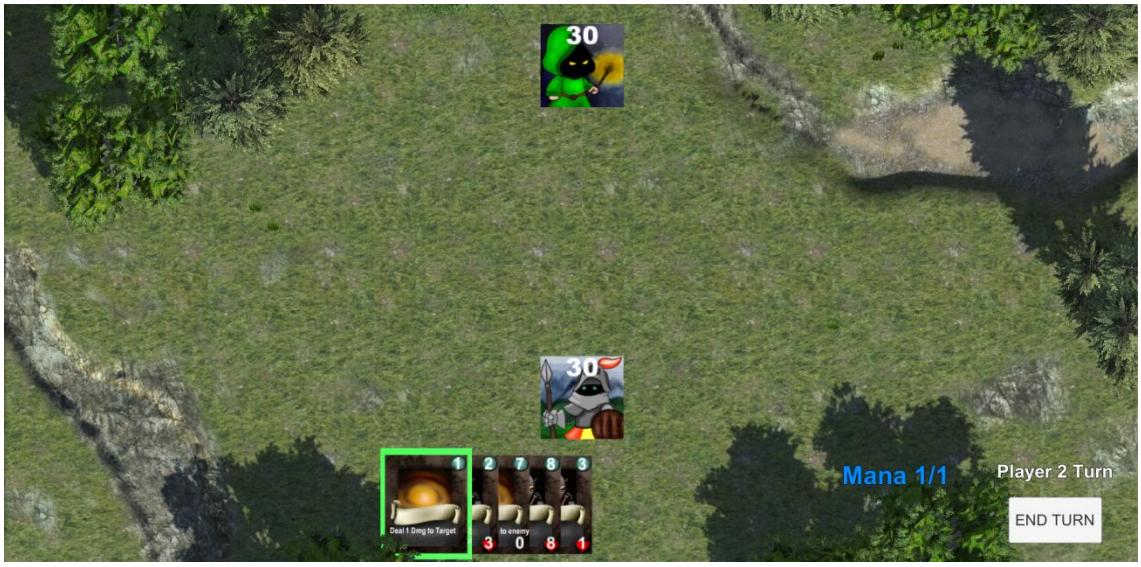


KUVA 1. Pelin tapahtumasarja UML-kaaviolla ilmaistuna Pelaaja 1:n vuorolla.

Tällä hetkellä peliin on asetettu ennalta säännöt, mutta näitä voi assetin käyttäjä muuttaa halutessaan. Tällä hetkellä pelissä olevia sääntöjä sekä rajoituksia:

- Pelaajalla on maksimissaan 10 manaa.
- Manat alkavat yhdestä ja kasvavat kierroksittain yhdellä.
- Pelaajalla voi olla kädessään korkeintaan 10 korttia.
- Pelaajalla voi olla kentällä seitsemän minionia.
- Pelaajan korttipakka koostuu 30 kortista.
- Peliä voi pelata vain samalla koneella vuoropohjaisesti.
- Pelin häviää, jos elämäpisteet loppuvat.

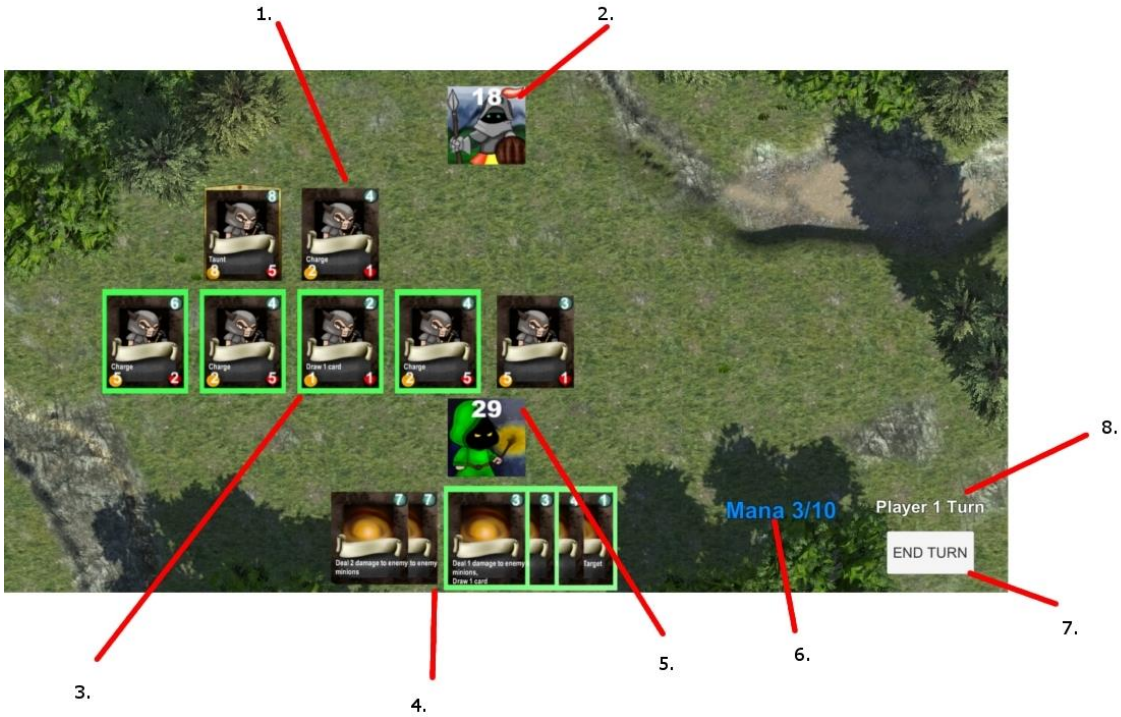
Pelitilanne pelin aloitusvaiheessa (kuva 2) ja pelin loppuvaiheessa (kuva 3) havainnollistavat pelin kulkua. Kuvista voidaan nähdä Unityn tarjoama graafinen tausta ns. korttipöydälle.



KUVA 2. Pelistä otettu pelinäkymä pelin aloitusvaiheesta.

KUVA 3. Pelistä otettu pelinäkymä useiden vuorojen jälkeen.

Pelinäkymässä (kuva 4) voidaan nähdä eriteltynä siihen kuuluvat elementit. Kukin elementti pitää sisällään ohjelmoituja toimintoja. Elementeille selitykset on nähtävissä taulukosta 1.



KUVA 4. Pelinäkymässä esiintyvien elementtien merkinnät merkittynä kuvaan.

TAULUKKO 1. Pelinäkymän elementtien selitykset

Elementin nro.	Selitys
1	vastapuolen kentällä olevat minionit
2	vastapuolen sankari ja sankarin elämäpisteet
3	kyseisen vuoron pelaajan kentällä olevat minionit
4	kyseisen vuoron pelaajan kädessä olevat kortit
5	kyseisen vuoron pelaajan sankari ja sankarin elämäpisteet
6	tekstikenttä joka ilmoittaa jäljellä olevan manan
7	painike, josta vuoro päätetään
8	tekstikenttä, joka ilmoittaa pelaavan pelaajan vuoron

Korttien eri ominaisuudet

Pelissä korteilla on tietynlaisia ominaisuuksia, joiden toimintaa ja vaikutusta oli luontevampaa kuvata englanninkielisin termein. Taulukossa 2 on listattuna pelissä käytettyjen toimintojen ja ominaisuuksien nimet ja niille vastaava selvennös. Selvennökset eivät ole suoria suomennoksia kyseiselle ominaisuudelle.

TAULUKKO 2. Korttien eri ominaisuuksien selvennökset

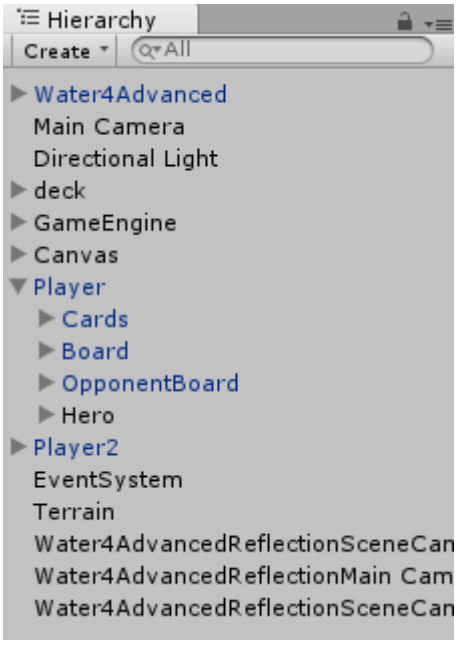
Ominaisuus	Selvennös
charge	pelaajan minioni voi hyökätä heti kun tämä on pelattu kentälle
single target damage spell	loitsu tekee vahinkoa yhteen minioniin tai pelaajaan
multitarget damage spell	loitsu tekee vahinkoa kaikkiin vastapuolen minioneihin
card draw	nostaa määrätyn verran kortteja
taunt	minion joka on tapettava ennen kuin pelaaja voi hyökätä minioneilla vastapuolen pelaajaan
minion health & damage	minioneiden elämä ja hyökkäyspisteet
card cost	kortin vaatima mana hinta

3 TYÖN TOTEUTUS

Tässä luvussa käydään syvemmin läpi ohjelman rakennetta ja toteutusta vaiheittain. Luvuissa myös mainitaan vaihtoehtoisia toteutustapoja sekä kerrotaan, miksi päädyttiin kyseiseen vaihtoehtoon.

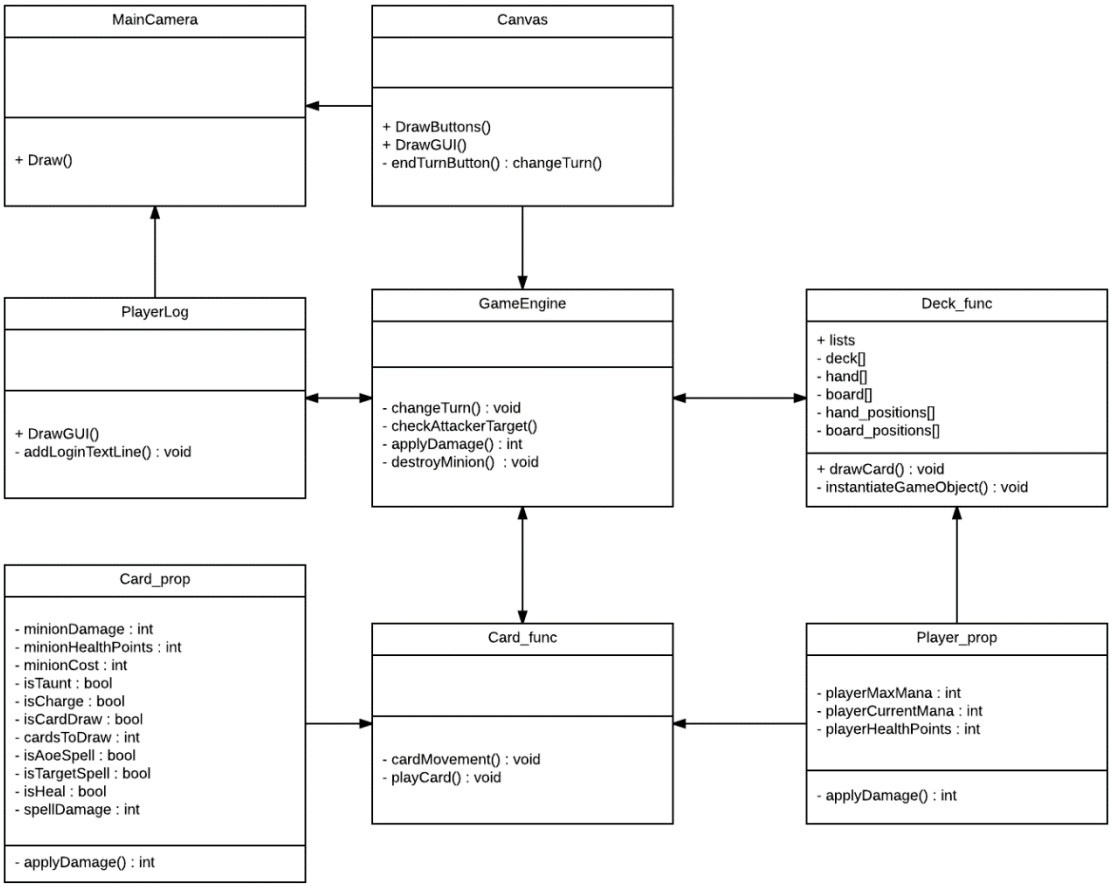
3.1 Ohjelman rakenne

Pelissä oleva ”scene” (kuva 5) on otettu Unityn editorinäkymästä, josta sen rakenne voidaan havaita. Kuvassa on myös Unityn automaattisesti lisäämiä komponentteja sekä muutama Unityn tarjoama ilmainen komponentti, jotka vaikuttavat siihen, miltä peli näyttää.



KUVA 5. Scenen rakenteen ilmaiseva kuva, joka on otettu Unityn editorin näkymästä.

Tarkemman käsityksen ohjelman rakenteesta nähdään UML-kaaviossa (kuva 6), jossa on lueteltuna eri toiminnot ja muuttujat sekä Unityn omille että itse tehdyille komponenteille. Kaaviossa on myös merkittynä yhteydet eri komponenttien välillä. Komponenttien keskeisimmät toiminnat on kerrottuna taulukossa 3, jossa on myös mukana Unityn omia komponentteja.

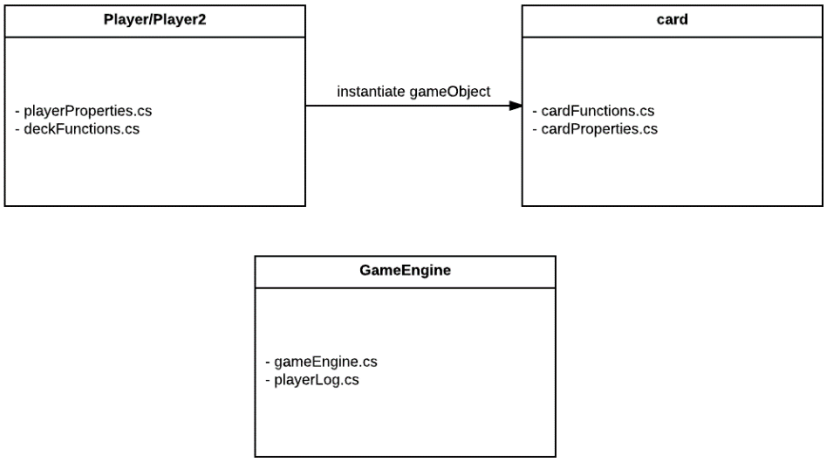


KUVA 6. Ohjelmakoodin rakenne esitettynä UML-kaaviolla.

TAULUKKO 3. Kuvan 6 komponenttien ominaisuuksien selitys

Komponentti	Komponentin ominaisuus
GameEngine	pitää sisällään scriptejä, jotka huolehtivat pelin toiminnallisuudesta
PlayerLog	ikkuna, jossa nähdään tapahtuneet eventit
Deck_func	kortin nostaminen ja listat
Player_prop	pelaajan tiedot, manat, elämäpisteet
Canvas	Unityn oma käyttöliittymä, johon sisältyvät painikkeet sekä tekstit
MainCamera	pelin kamera, luodaan automaattisesti joka sceneen
Card_func	kortin liikuttaminen sekä kortin pelaaminen
Card_prop	kortin ominaisuudet

Itse toteutetut peliobjektit voidaan lajitella neljään eri parent- eli emo-peliobjektiin (kuva 7). Nämä sisältävät muita peliobjekteja ja komponentteja (kuva 6). Näistä pelissä on valmiina muut, paitsi kortin peliobjekti, joka luodaan pelin aikana.



KUVA 7. Peliin luotujen peliobjektien rakenne kaaviolla ilmaistuna.

3.2 Kortin nostaminen korttipakasta

Kortin nostaminen korttipakasta tapahtuu arpomalla korttipakan koon mukaan numero. Määritellyn korttipakan mukaan luodaan uusi kortti pelaajan käteen. Kortit ovat peliobjekteja, joille on luotuna ”Prefab”. Prefabit ovat peliobjekteja, joille on määriteltynä arvot ennalta.

Kortit vähenevät korttipakasta, joten samaa korttia ei voida nostaa uudelleen. Korttipakkaan voidaan kuitenkin laittaa useampia kappaleita samasta kortista, mutta kortit käsitellään itsenäisinä. Korttien loputtua korttipakasta ei pelaaja saa enää uusia kortteja. Tämän sijaan pelaaja menettää elämäpisteitä ns. tyhjän kortin nostamisesta.

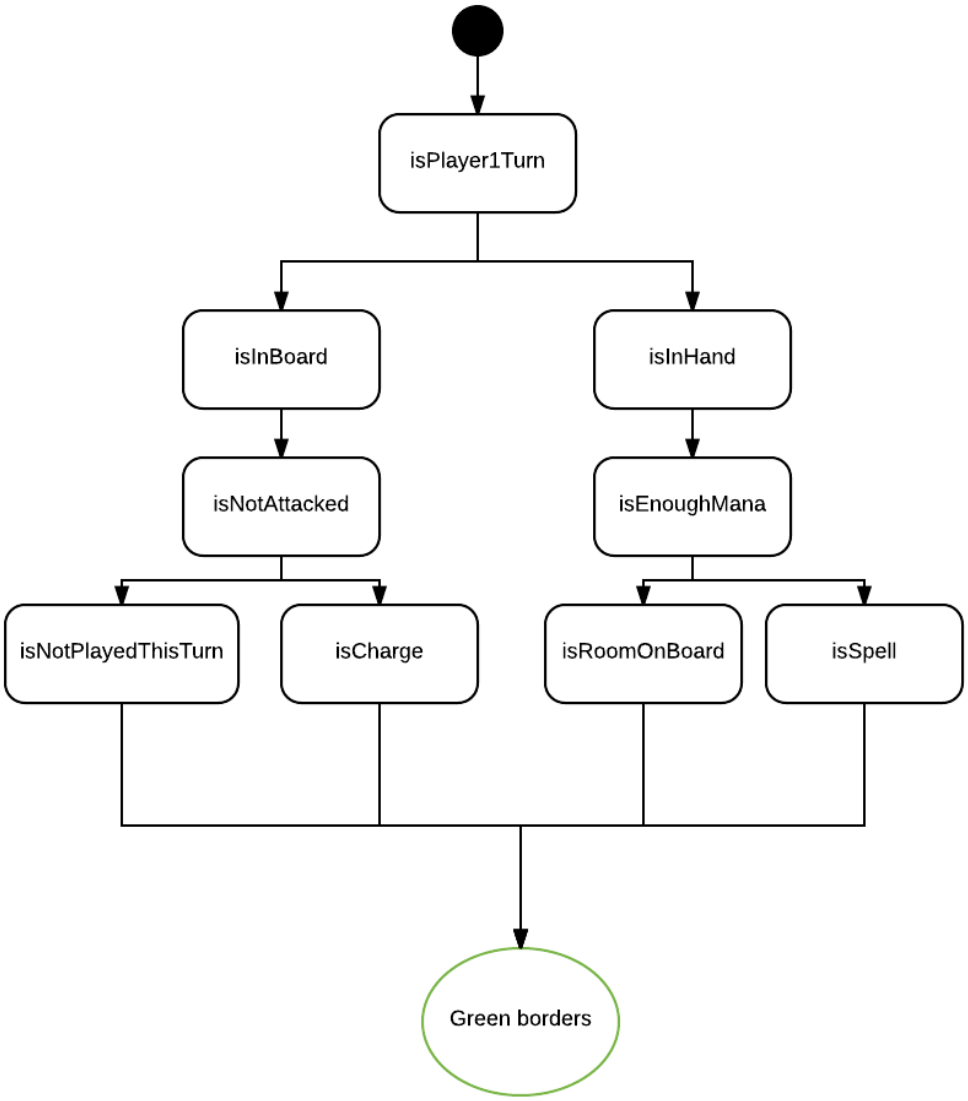
3.3 Kortin pelaaminen

Kortti pelataan kädestä sillä, että se raahataan hiirellä kentälle. Kortin pelaamisen voi keskeyttää, kun raahattu kortti vapautetaan käden kohdalla. Jos kyseessä on minion-kortti, siirtyy se kentälle, jos siellä on tilaa. Hyökkääminen minionilla, kun se on pelattuna kentälle, tapahtuu raahaamalla kortti halutun kohteen päälle ja vapauttamalla hiirenpainike. Jos kyseessä on loitsukortti, niin

kortti aktivoituu kun se raahataan kentälle. Kohteeseen tarkoitettu loitsukortti aktivoituu, kun se raahataan kohteen päälle.

3.4 Ehtolauseet

Ennen kuin kortilla voidaan hyökätä, käydään läpi useita ehtolauseita (kuva 8), jotta ei tehdä sääntöjen vastaisia liikkeitä. Korttien ympärille tulee vihreät laidat, joka kuvastaa sitä, voidaanko se pelata kentälle tai voidaanko sillä voidaan hyökätä. Jos jokin asetetuista ehdoista ei täyty, ei sille tule vihreitä laitoja.



KUVA 8. Ehdot siitä, voidaanko kortti pelata. Tätä kuvastaa vihreät laidat kortissa.

Ehtolauseet voidaan jakaa kahteen eri ryhmään sen mukaan, onko kortti kädessä tai onko se pöydällä. Pelaajat voivat liikuttaa vain omia korttejaan.

- Korttien ollessa pöydällä tarkistetaan,
    - onko kortti pöydällä
    - onko kortilla jo hyökätty
    - onko kortilla ”charge” eli voidaanko sillä hyökätä välittömästi
    - onko kortti pelattu tällä vuorolla.
- Korttien ollessa kädessä tarkistetaan,
    - onko kortti kädessä
    - onko pelaajalla tarpeeksi manaa pelata kortti
    - onko kortti kertakäyttöinen loitsu
    - onko kentällä tilaa kortin pelaamiseen.

3.5 Uusien korttien luominen

Uusien korttien luominen onnistuu Unityn editorissa. Korttien luontiprosessissa kopioidaan jokin olemassa olevista kortin prefab-tiedostoista. Korttiin voi tuoda haluamansa kuvan sekä tehdä muutokset kortista tuleviin vaikutuksiin ja ominaisuuksiin. Kortin ominaisuuksien muuttaminen editorinäkymässä (kuva 9) onnistuu yksinkertaisella ja nopealla tavalla. Prefabiin voidaan suoraan kirjoittaa halutut tiedot ja luoda kortista täsmälleen tietojen mukainen.

KUVA 9. Kortin ominaisuudet sisältävään komponenttiin määritellään halutut tiedot.

3.6 Korttipakan muuttaminen

Demossa ei vielä tällä hetkellä ole pelin sisäistä korttipakan muuttamista, joten korttipakan muuttaminen tapahtuu Unityn editorissa. Unityn editor-näkymästä valitaan haluttu pelaaja ja avataan korttipakka näkyviin. Tämän jälkeen viedään korteista tehdyt prefab-tiedostot korttipakkaan (kuva 10).

KUVA 10. Korttipakan lista, johon on vietynä prefabeja.

3.7 Toteutuksen eri vaiheet

Työn toteutus oli jaettuna eri vaiheisiin ja niistä kerrotaan tarkemmin seuraavissa luvuissa. Luvuissa myös mainitaan vaihtoehtoisia toteutustapoja sekä kerrotaan, miksi päädyttiin kyseiseen vaihtoehtoon.

3.7.1 Suunnittelu

Projektissa oli esimerkkinä muita vastaavanlaisia keräilykorttipelejä, jotka ovat mainittuna luvussa 2.1, joiden pohjalta voitiin listata halutut ominaisuudet ja toiminnot. Näistä karsittiin toteutettavaksi tärkeimmät sekä ne, jotka käytössä olevalla aikataululla olisi mahdollista toteuttaa. Pelissä nähtävästä näkymästä (kuva 3) laadittiin luonnos paperille, josta päästiin listaamaan kaikki toteutettavat ominaisuudet. Tämän jälkeen ohjelman rakenne ja eri toimintojen suhteet suunniteltiin UML-kaavioita hyväksikäyttäen.

3.7.2 Objektien sijainnit

Objektien sijainnit määrättiin listaan, jossa jokaisella solulla oli vastaava peliobjekti. Peliobjektit sisälsivät transform-komponentin, jolla määräytyi sijainti. Tällä keinolla kaikkien objektien sijaintia oli helpompi muuttaa, koska hiirellä voitiin siirtää objekteja eikä skriptissä tarvinnut määritellä uudelleen sijaintia jokaiselle peliobjektille.

3.7.3 Objektien luominen listasta ja korttipakan toiminta

Jotta korttipakka toimisi niin kuin kuuluukin, eli kortit vähenevät korttipakasta ja jokaisen kortin määrä on määritelty, oli luonnollista tehdä korttipakka omaksi listaksi. Listaan on määritelty korttia vastaava peliobjekti, joka luodaan ”Instantiate”-metodilla (kuva 11). Luodulle peliobjektikloonille asetetaan luontivaiheessa arvoja, jotta tiedetään, kumman pelaajan kortti on.

```
a = i - 1;
clone = Instantiate(draw, hand_positions[a].position, hand_positions[a].rotation) as GameObject;
hand[a] = clone;
clone.gameObject.GetComponent<card_func>().hand_id = a;
if (Player1)
{
    clone.transform.parent = cards1.transform;
    clone.gameObject.GetComponent<card_func>().Player1 = true;
}
if (!Player1)
{
    clone.transform.parent = cards2.transform;
    clone.gameObject.GetComponent<card_func>().Player1 = false;
}
```

KUVA 11. Instantiate-metodi, jolla luodaan pelikentälle peliobjekti.

3.7.4 Objektien siirtäminen käteen sekä kentälle

Kun tutkittiin, miten kortteja saadaan liikuteltua hiirellä, päädyttiin ohjelmassa käyttämään ”OnMouseDrag”-metodia. Tällä tarkastellaan hiiren painallusta ja liikettä. Kun kortti luodaan pelaajan käteen korttipakasta, pysyy se samana peliobjektina, vaikka se siirrettäisiin kentälle. Siltä muuttuvat arvot, jotka määrittelevät onko kortti pelattu kädestä.

3.7.5 Kortin pelaaminen

Korttien pelaaminen päätettiin toteuttaa siten, että kortti vedetään tarpeeksi kauas kädestä, jolloin se aktivoituu. Vastaavasti yhden kohteen loitsu on periaatteessa minion-kortti charge-ominaisuudella. Loitsukortille valitaan minion-kortin tapaan kohde. Loitsukorttia rajaa myös se, että sen sijannin tulee olla tarpeeksi kaukana lähtöpisteestä, jotta se pelataan. Tällä saadaan aikaiseksi se, että korttien pelaamisen voi myös peruuttaa.

3.7.6 OnMouseOver

Tavoitteena oli saada lähetettyä tieto toiselle objektille, kun hiirellä liikutellaan korttia. Tässä päädyttiin käyttämään ”OnMouseOver”-metodia. Tällä keinolla käydään ehtolauseita läpi, jossa boolean-arvojen vaihtumista tarkastellaan.

Objektin liikkuessa ja palautuessa lähtöpisteeseen muuttuu objektin boolean-arvo. Hiiren ollessa objektin päällä, käytetään myös ”OnMouseOver”-metodia.

Tässä boolean-arvo muuttuu se mukaan, onko hiiri objektin päällä. Tietyn ajan päästä, kun pelattava kortti on vietynä toisen objektin päälle ja hiiri on vapautettuna, tarkastellaan kentällä olevat objektit läpi. Ne objektit, joilla boolean-arvot ovat ehtojen mukaiset, lähetetään kysesisiltä peliobjekteilta viestit toisilleen.

3.7.7 BoxCollider

Toisessa toteutustavassa käytössä oli pelinsisäinen ”BoxCollider”. Näiden tilalta voidaan käyttää myös muun muotoisia ”Collider”-ominaisuuksia, jos käytössä olisi erimuotoisia kortteja. Nämä ominaisuudet tarkastelevat pelissä tapahtuvaa törmäystä. Jokaisella pelikortilla tässä toteutustavassa on kortin kokoinen ”BoxCollider”. Hiiren kohdalla vastaavasti on pieni ”Collider”, jonka törmäystä tarkastellaan. Tämä toteutustapa ei kuitenkaan tuo ohjelmaan parempaa toimivuutta, vaan se tuo ylimääräistä laskentaa törmäyksille.

3.7.8 Korttien ominaisuudet ja vaikutukset

Tarkoituksena oli toteuttaa yksinkertainen minioni-kortti sekä lisäksi jokin loitsukortti. Minion-kortille toteutettiin perusominaisuuksien lisäksi erikoisominaisuuksia. Näihin ominaisuuksiin kuuluvat ”taunt” ja ”charge”.

Kun pelataan minion-kortti, jolle on määriteltynä ominaisuus ”taunt”, muutetaan tällöin pelaajan kokonaislukumuuttujaa. Kun pelaajan hallussa oleva ”taunt”-ominaisuuden sisältävä minion kuolee, muutetaan tällöin pelaajan kokonaislukumuuttujaa.

Toisen pelaajan hyökätessä minionilla toiseen pelaajaan tai minioniin tarkistetaan tällöin, onko pelaajan taunt-arvo yhtä suuri kuin nolla. Arvon salliessa hyökkäys tapahtuu normaalisti, mutta arvosta voidaan myös katsoa, estetäänkö hyökkäys. Jos hyökättävään kohteeseen olevalla minion-kortilla on taunt-ominaisuus, tapahtuu hyökkäys tällöin normaalisti.

Minion-korttin ominaisuudella ”charge”, on oma muuttuja. Muuttujan arvoa tarkastellaan, kun kortti on pelattu kentälle. Muuttujan avulla ohitetaan minionilla

hyökkäämistä koskeva rajoitus ja täten sillä voidaan hyökätä välittömästi. Minionille olisi tämän avulla mahdollista lisätä hyökkäysten määrää vuorossa.

3.7.9 Elämäpisteet, vahinko ja kortin hinta

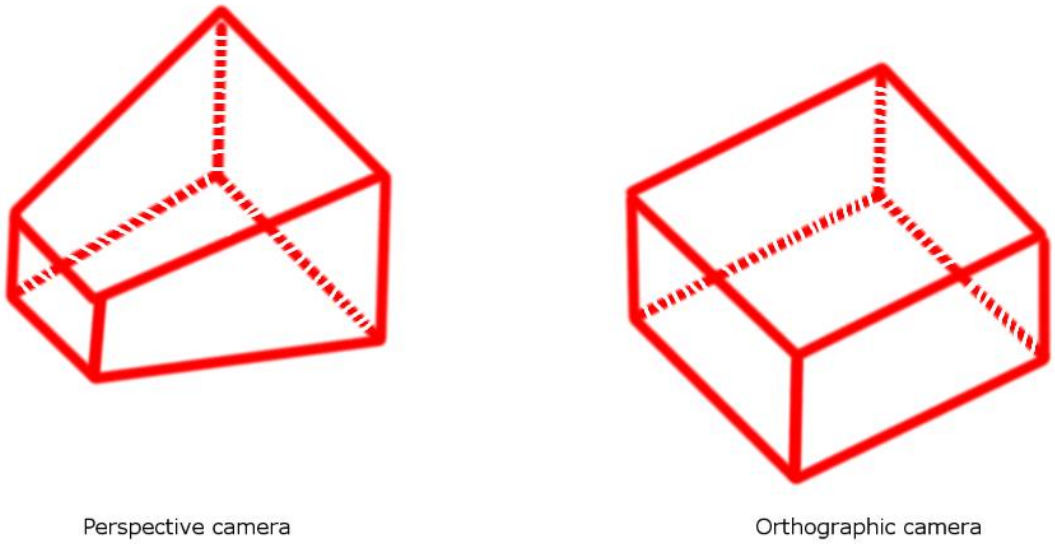
Minion-kortille määriteltiin skriptiin kokonaislukumuuttujat elämäpisteille sekä vahingolle. Vahingolla tarkoitetaan, kuinka paljon elämäpisteitä kulutetaan kohteelta johon hyökätään. Minion-kortin lisäksi loitsukortille annettiin muuttuja, joka ilmaisee kortin vaatiman ”manan” eli energian tämän pelaamiseen. Manalla kuvataan kortin pelaamisesta maksettava hinta. Kyseiset arvot näkyvät reaaliaikaisesti kortissa (kuva 12). Tätä varten peliobjektiin lisättiin ”TextMesh”, koska kortissa olevia arvoja muutetaan. Tähän ”TextMesh”-komponentiin voidaan ohjelman kautta päivittää haluttu teksti tai numero.

KUVA 12. Minion-korttissa olevat elementin merkittynä kuvaan.

TAULUKKO 4. Minion-kortiin merkittyjen elementtien selitykset

Elementin nro.	Selitys elementille
1	vihreät laidat, jotka kuvastavat, voidaanko minionilla hyökätä tai voidaanko kortti pelata kädestä
2	tekstikenttä, joka ilmoittaa kortin muut ominaisuudet
3	minionin hyökkäysvoima
4	kortin pelaamiseen vaadittu mana
5	minionin elämäpisteet

Korteissa olevat tekstit näkyivät toisten korttien lävitse, jolloin ohjelmaan lisättiin ”shader”-tiedosto. Fonttitiedosto muutettiin toiseen muotoon, jotta tekstit jäivät piiloon toisten korttien peittäessä ne (Unity Technologies 2016g, viitattu 11.09.2016). Kortteja siirrellessä ilmeni ongelmaksi, että tekstit välillä katosivat. Ongelma korjaantui, kun valittiin ”Main Camera”-objekti (kuva 5) ja tämän asetuksista muutettiin ”Perspective camera” valintaan ”Orthographic camera”. Asetukset vaikuttavat siihen, miten kameraa katsotaan (kuva 13).



KUVA 13. Kameroiden eri näkymien erot havainnollistettuna.

3.7.10 Kortin arpominen

Molemmille pelaajille on määrättynä oma korttipakka. Tämä koostuu listasta, johon mahtuu ennalta määrätysti 30 solua, jos kyseessä on 30 kortin korttipakka. Jokaisessa listalla olevassa solussa on korttia vastaava peliobjekti. Kortin nostaminen tapahtuu sillä, että arvotaan numero 0–29 ja luodaan solussa määritelty peliobjekti. Numerot arvotaan kyseiseltä väliltä, koska listan ensimmäinen arvo on solussa 0. Jos arvoa vastaavassa solussa on arvona ”null” eli tyhjä, arvotaan uusi numero. Kun peliobjekti on luotu, muutetaan kyseisen solun arvo tyhjäksi.

Tästä voidaan päätellä, että kyseisellä toiminnalla tapahtuu paljon turhaa laskemista ja uusien numeroiden arpomista. Kun korttipakassa olevat kortit vähenevät ja arvottua lukua vastaa tyhjä, joudutaan arpomaan uusi luku. Kortin nostamista ei kuitenkaan tapahdu kovin usein, eikä tämä vaikuta pelin toimintaan, koska nykyisillä prosessoreilla ei tämän laskemiseen kulu paljon aikaa.

Ohjelmaan tarvitaan muutos, jos kortin nostamista tai vastaavanlaista toimintaa tapahtuu useammin ja halutaan saada ohjelma kevyemmäksi. Tämä onnistuu sillä, että arvoa ei muuteta tyhjäksi, vaan listan viimeinen arvo siirrettäisiin kyseiselle paikalle. Tämän jälkeen arvottavien lukujen määrää pienennettäisiin yhdellä.

3.7.11 Loitsukortti

Loitsukortin pelaaminen on toiminnaltaan (kuva 14) melkein samanlainen kuin minion-kortin vahingon tekeminen. Sen pelaamista ei kuitenkaan rajoita kentällä olevien minion-korttien määrä eikä se, onko minion-korteilla ”taunt”-ominaisuutta. Loitsukortin poikkeaa minion-kortista myös sillä, että sen aiheuttaman vahingon määrä on nähtävissä ominaisuus kentässä. Loitsukortilla ei myöskään ole elämäpisteitä.

```
void SpellHandler(int id)
{
    if (spellPlayed)
    {
        int dmg;
        if (Player1Turn)
        {
            dmg = play1.GetComponent<deck_func>().hand[id].GetComponent<card_prop>().spellDmg;
            if (play1.GetComponent<deck_func>().hand[id].GetComponent<card_prop>().isAoeDmg)
            {
                for (int i = 0; i <= 6; i++)
                {
                    if (play2.GetComponent<deck_func>().board[i] != null)
                    {
                        play2.GetComponent<deck_func>().board[i].GetComponent<card_prop>().SendMessage("ApplyDamage", dmg);
                        if (destroy)
                            play2.GetComponent<deck_func>().board[i] = null;
                        destroy = false;
                    }
                }
            }
            play1.GetComponent<deck_func>().hand[id].GetComponent<card_func>().attacking = false;
            play1.GetComponent<deck_func>().hand[id].GetComponent<card_func>().attacked = true;
            play1.GetComponent<player_prop>().currentMana = play1.GetComponent<player_prop>().currentMana -
                play1.GetComponent<deck_func>().hand[id].GetComponent<card_prop>().cost;
            play1.GetComponent<deck_func>().hand[id].GetComponent<card_prop>().PlaySpell();
            play2.GetComponent<deck_func>().hand[id] = null;
        }
```

KUVA 14. Loitsukortin metodi, jolla sen vaikutukset tulevat voimaan.

3.7.12 Kaksi pelaajaa ja vuorot

Jotta pelistä saisi pelatattavan, tuli peliin lisätä myös toinen pelaaja. Toista pelaajaa ohjataan samalla tietokoneella vuoropohjaisesti. Komponentteihin tuli lisätä totuusarvoja, joilla voidaan antaa pelaajille identiteetit. Nämä huolehtivat

mm. siitä, että kortit vaikuttavat oikeaan kohteeseen. Samanlaiset kortit ovat identtisiä keskenään, mutta ne ovat merkittynä tietyn pelaajan omistukseen. Kortit eivät myöskään katoa mihinkään, kun vuoroa vaihdetaan, vaan niiden sijaintia muutetaan sen mukaan, kenen pelaajan vuoro on.

4 TULOKSET JA JOHTOPÄÄTÖKSET

Luvussa kerrataan, mitkä asiat oltaisiin voitu tehdä paremmin ja millaisia ongelmia projektin edetessä ilmeni. Pelissä löytyvien ongelmien etsimisessä hyvänä apuna oli ”PlayerLog”-skripti, jolla merkittävimmät pelissä tapahtuneet eventit kirjattiin tekstilaatikkoon. Ohjelman muissa skripteissä oli määriteltynä, mitä ja milloin tulostus tekstilaatikkoon tapahtuu. Esimerkiksi kun jokin objekti luodaan, tällöin saadaan näkyviin objektin nimi tekstikenttään. Tätä seuraamalla oli mahdollista paikantaa, missä kohtaa ohjelmassa tapahtui virhe. Vastaavanlaisen ominaisuuden saadaan myös ”Debug.Log”-metodilla, joka tulostaa halutun tekstin Unityn editorin konsoliin.

4.1 Paranneltavia ominaisuuksia

Kortilla hyökkäämisestä, kortin palautumisesta käteen ja kortin pelaamisesta tulisi tehdä sujuvampaa. Tällöin ei tarvitse odotella niin kauan ja pelaaja voi tehdä seuraavan toiminnon. Seuraavien komentojen estäminen ennen edellisen komennon valmistumista johtuu ohjelman rakenteesta sekä toteutustavasta. Toteutustapa myös takaa toimintojen toimivuuden. Ohjelman rakennetta muuttamalla, voidaan korjata odotusaikaan ja vuoronvaihtoon liittyvät ongelmat. Minionilla tai loitsukortilla hyökätessä on sujuvampaa, jos kortti jää paikoilleen ja hiiren kohdalle tulee indikaattori, joka ilmaisee, mihin hyökkäys tehdään. Tällä vältytään siltä, ettei kortti peitä kohdetta.

4.2 Lisättäviä ominaisuuksia

Demoon kertyi myös lisättäviä ominaisuuksia, jotka lisäajalla olisi voinut toteuttaa. Pelaajan olisi tärkeää nähdä, paljonko hänellä itsellään sekä vastapuolella on kortteja korttipakassa jäljellä. Pelaajan tulisi myös nähdä vastustajan kädessä olevien korttien määrän. Korttien tuhouduttua on selkeämpää, jos tuhoutumisesta on viivästetty animaatio. Tällöin pelaaja ehtii nähdä, mitä pelissä tapahtui.

4.3 Asset Storessa myynti

Ajatuksena oli, että peli olisi laitettu myyntiin Asset Storeen kesällä ja samalla olisi nähty, miten tuote menisi kaupaksi. Palautteista olisi kerätty mahdollisia parannusehdotuksia. Unityssä tulee melko usein päivityksiä, eivätkä assetit välttämättä enää toimi uudemmassa versiossa. Assettien kehittäjät joutuvat itse päivittämään assetit uudempaan Unityn versioon toimivaksi (Unity Technologies 2016f, viitattu 22.11.2016). Joskus Unityn päivittäminen ei vaikuta assettien toimintaan, eikä tällöin assetteja tarvitse päivittää.

Varattu aikataulu ei riittänyt Unityssä tapahtuneiden muutosten johdosta tarvittaviin korjauksiin. Unityn käyttöehtojen mukaan vanhempaan versioon tehtyä assettia ei Asset Storessa voi myydä. Tästä voidaan oppia assetteja myydessä, että assetteja tulee varautua päivittämään aina ajoittain.

Assettien myyminen on osaksi haastavampaa ja työläämpää kuin esimerkiksi pelien. Näihin ei tarvitse tehdä päivityksiä tai korjauksia kuin silloin, kun löydetään ohjelmasta jokin virhe tai tarvittava parannus. Toimiva peli voidaan jättää myytäväksi ja keskittyä seuraaviin projekteihin.

Demopeli on mahdollista laittaa myyntiin Unityn Asset Storeen, kun päivitysten myötä tarvittavat muutokset on tehty. Jotta demolle olisi myös ostajia, on sen hyvä sisältää lisäksi seuraavat ominaisuudet:

- verkon kautta pelattava moninpeli
- mahdollisuus muokata pelimekaniikkaa
- tuki mobiililaitteille.

Asseteissa tärkeimpänä ominaisuutena on niiden muokattavuus. Toteutetun ohjelman rakenne on mietitty lähinnä nykyiselle mekaniikalle. Tämän takia ohjelmaan tulisi tehdä pelimekaniikan muutos helpommaksi.

5 YHTEENVETO

Työn aiheena oli keräilykorttipeli, jossa havaitaan eri korttien toiminta ja vaikutukset. Pelissä on perusmekaniikoiden lisäksi toteutettuna toisistaan poikkeavia erikoistoimintoja. Ohjelman toteutusvaiheessa on pyritty huomioimaan pelin mahdollinen muokkaaminen.

Opinnäytetyön lopputulos oli hyvä, huomioon ottaen asetetut tavoitteet. Työssä päästiin kokemaan ohjelman rakentaminen vaiheittain. Asioiden ja ominaisuuksien priorisointi tuli tärkeäksi, kun tavoitteena oli päästä haluttuun lopputulokseen. Tällöin tiettyjä ominaisuuksia tuli jättää pois, kun pystyi arvioimaan niiden tärkeyden sekä laajuuden.

Ohjelman määrittelyvaiheessa oli haastavaa suunnitella ohjelman rakenne, kun tiedossa ei ollut tarkkoja määritelmiä ohjelman muokattavuudesta. Tässä olisi ollut tärkeää, että kaikki asiat olisivat olleet listattuna, joita mahdollinen assetin ostaja voi muokata. Ohjelman rakennetta pohtiessa syntyi haasteeksi, miten jokin toiminto voidaan integroida johonkin muuhun toimintoon. Usein toiminto tehtiin vain kyseistä ohjelmaa ja sen pelimekaniikkaa varten.

Peliä tehdessä kävi selville, kuinka laaja yksinkertainen korttipeli voi olla. Korttipelissä on paljon huomioon otettavia ehtoja, jotka rajaavat, mitä pelaaja voi tehdä. Tätä voidaan soveltaa myös muiden ohjelmien rakenteita suunnitellessa.

Opittuihin asioihin voidaan luokitella UML-kaavioiden käyttö. Kaavioiden käyttö vei aikaa, koska UML-kaavioita ei ole käytetty edellisissä projekteissa juuri ollenkaan. Mielestäni UML-kaaviot selventävät työn vaiheita ja prosessien toimintaa. Näitä käyttämällä ohjelman rakenteen pystyi havaitsemaan paremmin, mikä perustelee sen, miksi näitä tulisi käyttää.

Työstä jäi puuttumaan myyntivaihe. Aikataulun sekä ilmenneiden ongelmien takia myyntivaihe jätettiin pois. Opinnäytetyöprojektissa ei ollut muita kehittäjiä

mukana, mikä toi projektiin omat hyvät puolensa sekä haasteet. Hyvinä puolina voidaan pitää, että ohjelmoin projektissa kaiken alusta loppuun saakka. Itse ohjelmoidessa rakenne ja toiminta olivat tuttuja ja helppoja muokata tarpeen vaatiessa.

Haasteina voidaan pitää sitä, ettei ohjelman rakennetta tarkastellessa ollut toista mielipidettä ohjelman helposta muokattavuudesta. Lisäksi demossa oli useita osa-alueita, eikä yksittäisiin ominaisuuksiin voinut syventyä tarkemmin varatulla aikataululla. Yhtenä osa-alueena voidaan pitää assettien myyntiä. Tässä tapauksessa olisi ollut helpompaa, jos kyseiseen tehtävään olisi varattu asiaan perehtynyt henkilö, joka voi määritellä tuotteelle halutut ominaisuudet, hinnan sekä myyntiehdot. Tässä näkyi selvästi, kuinka projektia helpottaa, kun voi keskittyä tiettyyn asiaan.

LÄHTEET

Lucid Software Inc. 2016. Diagram maker. Viitattu 22.11.2016, https://www.lucidchart.com/.

Microsoft 2016. Integrated development environment. Viitattu 22.11.2016, https://www.visualstudio.com/.

The GIMP Team 2016. Image manipulation program. Viitattu 22.11.2016, https://www.gimp.org/.

Unity Technologies 2016a. Unity Company facts. Viitattu 22.11.2016, https://unity3d.com/public-relations/.

Unity Technologies 2016b. Games made with Unity. Viitattu 17.11.2016, https://madewith.unity.com/.

Unity Technologies 2016c. Unity payment plans. Viitattu 22.11.2016, https://store.unity.com/.

Unity Technologies 2016d. Unity Asset Store. Viitattu 15.11.2016, https://www.assetstore.unity3d.com/en/.

Unity Technologies 2016e. Selling Assets. Viitattu 15.11.2016, https://unity3d.com/asset-store/sell-assets.

Unity Technologies 2016f. Asset Store Provider Agreement. Viitattu 15.11.2016, https://unity3d.com/legal/as_provider.

Unity Technologies 2016g. 3DText with shader file. Viitattu 11.09.2016, http://wiki.unity3d.com/index.php?title=3DText.