VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Jussi Siltakorpi

Ideasta peliksi – Mobiilipeliohjelmoinnin oppiminen Unity3D-ympäristössä

Liiketalous

2016

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

TIIVISTELMÄ

Tekijä	Jussi Siltakorpi
Opinnäytetyön nimi	Ideasta peliksi – Mobiilipeliohjelmoinnin oppiminen Unity3D-ympäristössä
Vuosi	2016
Kieli	suomi
Sivumäärä	58
Ohjaaja	Sirkka Hellman

Tämä opinnäytetyö käsittelee mobiilipelikehityksen oppimista mobiilipelin toteuttamisen yhteydessä Unity3D-ympäristössä. Työn tarkoituksena on selvittää, miten Unity3D tukee C# ohjelmointikielen oppimista pelikehityksen aikana sekä Unity3D ohjelmiston toimivuutta pelinkehityksessä. Työn aihe valittiin, koska peliala on nopeasti kasvava kultturiteollisuudenala, joka tarjoaa työpaikkoja monille ohjelmoijille ja monille muille eri alojen ammattilaisille. Työn toisena tarkoituksena on toimia lisänä jo Vaasan ammattikorkeakoulussa opetettaville peliohjelmointia käsitteleville kursseille.

Työn teoriaosuudessa käsitellään yleisesti pelialaa Suomessa sekä esitellään Unity3D-pelimoottoria ja C#-ohjelmointikieltä peliohjelmoinnissa. Työssä käydään läpi myös Unity editorin käyttöliittymää, sekä Unity Techonologies:in tarjoamia tutoriaaleja ja Asset-kauppapaikkaa.

Työssä Unity3D v5.5-editorilla toteutettu mobiilipeli esitellään kuvin ja peliin toiminnalisuutta tuovat ohjelmointiskriptit käydään läpi. Pelin toiminallisuus kokonaisuudessaan esitetään pelattavan demon avulla.

Avainsanat Peliohjelmointi, Unity3D, C#,

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittelyn koulutusohjelma

ABSTRACT

Author	Jussi Siltakorpi
Title	From an Idea to a Playable Game – Learning Mobile Game Programming in Unity3D
Year	2016
Language	Finnish
Pages	58
Name of Supervisor	Sirkka Hellman

This thesis studied learning mobile game programming while developing a mobile game in Unity3D. This work examined how Unity3D supports learning C# programming language while developing a mobile game and how well Unity3D program works in mobile game development. The topic for this thesis was chosen because rapidly growing game industry offers many different type of careers for different professions.

The theoretical background of this thesis consists of an overview of the game industry in Finland and the Unity game engine. The thesis also examines C# programming language in Unity game developing. The graphical user interface and the related tutorials as well as Asset store, all provided by Unity Technologies, are also discussed.

The mobile game produced with Unity3D v5.5 editor is introduced with pictures and the scripts that bring functionality to the game are explained. The full functionality of the game is shown with a playable demo.

Keywords Game programming, Unity3D, C#,

SISÄLLYS

TIIVISTELMÄ
ABSTRACT

1 JOHDANTO 10
1.1 Tausta 10
1.2 Tavoitteet 11
1.3 Rakenne 11
2 PELINKEHITYS 12
2.1 Historia 12
2.2 Pelinkehitys suomessa 12
2.3 Pelialan koulutus ja työpaikat Suomessa 13
2.4 Pelimoottorit 14
3 UNITY3D 15
3.1 Lataus / Asennus 16
3.2 Projektin luonti 16
3.3 Käyttöliittymä 17
3.3.1 Scene-näkymä 18
3.3.2 Game-näkymä 19
3.3.3 Project-näkymä 19
3.3.4 Inspector-näkymä 21
3.3.5 Toolbar-näkymä 21
3.3.6 Console-näkymä 22
3.4 Ohjelmointikielet 22
3.5 Asset Store 23
4 ”MIRRORS” MOBIILIPELI 24
4.1 Idea ..... **Virhe. Kirjanmerkkiä ei ole määritetty.**
4.2 Suunnittelu 24
4.3 Päävalikko-Scenen toteutus 26
4.3.1 Päävalikko 26
4.3.2 Kentän ja pallon värin valinta 28

4.4 Kenttä-scenen toteutus 30
4.4.1 Pelialue ja pelaaja 30
4.4.2 Virtuaalinen joystick 31
4.4.3 Pelaajan sekä kameran liikkuminen 32
4.4.4 Säteen alkupiste 35
4.4.5 Säteen peilaava peliobjekti 37
4.4.6 Maalialueen portti 38
4.4.7 Pause menu sekä painike 38
4.4.8 Kentän läpäisy 39
4.5 Jatkokehitys 40
5 YHTEENVETO 41
LÄHTEET 43

LIITTEET

KÄSITTEET

GUI	Graphical User Interface – Graafinen käyttöliittymä.
HTML5	Hypertext Markup Language ohjelmointikielen viides versio.
SCRATCH	Lapsille ja nuorille tarkoitettu ohjelmointikieli joka sisältää graafisen käyttöliittymän.
C#	Oliopohjainen ohjelmointikieli.
UI	User interface – Käyttöliittymä
SKRIPTI	Komentosarja, joka liittää peliobjektiin toiminnallisuutta.
INDIEKEHTITTÄJÄ	Itsenäinen pelinkehittäjä (engl. Independent developer.)
AAA-LUOKITUS	Korkeimman kehitys- ja markkinointibudjetin peli.
ASSET	Unityn lisäosia, joita käytetään pelin tai sovelluksen rakentamiseen.
GDD	Game Desing Document
SCENE	Scene sisältää pelin peliobjektit.

KUVIO- JA TAULUKKOLUETTELO

Kaavio 1. Suomen pelitoimialan liikevaihdon kehitys miljoonina (Neogames 2016) **Virhe. Kirjanmerkkiä ei ole määritetty.**

Kaavio 2. Pelialan työntekijämäärien kehitys suomessa (Neogames, 2016) **Virhe. Kirjanmerkkiä ei ole määritetty.**

Kuva 1. Unity ohjelmiston aloitussivu. 16

Kuva 2. Projektin luomissivu. 17

Kuva 3. Unity editorin 2 by 3 Käyttöliittymä 18

Kuva 4. Scene-näkymä. 18

Kuva 5. Game-ikkunan näkymä. 19

Kuva 6. Project-näkymä Hierarchy-näkymä. 20

Kuva 7. Hierarchy näkymä jossa child-objekteja. 20

Kuva 8. Mirror (1) peliobjektin ominaisuudet. 21

Kuva 9. Unity editorin Toolbar 22

Kuva 10. Console-näkymä 22

Kuva 11. Unity Asset Store 23

Kuva 12. Peiliobjektin konseptipiirros 25

Kuva 13. Valmiin kentän konseptipiirros 25

Kuva 14. Main Menu Game-näkymässä 26

Kuva 15. Main Menun Scene-näkymässä 27

Kuva 16. Level Select painikkeen On Click eventin tiedot. 28

Kuva 17. MainMenu-objektin Inspector-näkymä. 28

Kuva 18. Inspector-näkymän pyyhkimisen mahdollistavat asetukset. 29

Kuva 19. Kenttävalikko pelissä. 30

Kuva 20. Kenttä sekä pelaajaobjekti scene-näkymässä 30

Kuva 21. Virtual joystickin osat JoyImg ja JoyBackground 31

Kuva 26. Joystickien sijainti pelin GUI:ssa 33

Kuva 23. Beamstart-objektin Inspector-näkymä 36

Kuva 24. Mirror-objektin rakenne. 37
Kuva 25. Pausemenu päällä pelissä. 39
Kuva 26. Järjestys joka on asetettu Build Settings valikossa. 40

Koodiesimerkki 1. Kameran liikettä ohjaava Update metodi. 27
Koodiesimerkki 2. Kentänvalintapainikkeiden luontikoodi. 29
Koodiesimerkki3. skriptin osa joka laskee peliobjektien liikkumista. 32
Koodiesimerkki 4. Metodit jotka palauttaa kosketuksen sijainnin. 32
Koodiesimerkki 5. Suuntatietojen keräys pelaajan liikuttamiselle ja kameran siirtämiselle. 34
Koodiesimerkki 6. Kameran liikkeen laskenta. 34
Koodiesimerkki 7. Säteen aloitus ja uuden säteen lähetys osumiskohdasta. 35
Koodiesimerkki 8. OnTriggerStay metodi. 37
Koodiesimerkki 9. Heijastusten määrää tarkkaileva metodi. 38
Koodiesimerkki 10. Kutsutut pause metodit. 39
Koodiesimerkki 11. Metodi jota kutsuttaessa vie seuraavaan sceneen. 40

LIITELUETTELO

LIITE 1. Pelidemon pc-version latauslinkki

LIITE 2. Pelin toteuttamisessa käytetyt skriptit

1 JOHDANTO

Tämän opinnäytetyön aihe valittiin, koska mobiilipeliteollisuus on kasvanut jo vuosia ja on edelleen vahvassa nousussa niin Suomessa kuin maailmallakin. Oma kiinnostukseni pelinkehitykseen motivoi valitsemaan peliohjelmointia käsittelevän aiheen. Työssä tullaan käyttämään kuvia sekä ohjelmakoodia havainnollistamaan esimerkkipelin suunnittelua, toimintoja, sekä Unity editorin käyttöliittymää.

1.1 Tausta

Suomessa mobiilipeliala on ollut kasvussa viime vuosikymmenen aikana. Peliala työllisti vuonna 2015 Suomessa noin 2700 työntekijää, (Neogames, 2016). Suomesta on noussut useita menestyspelejä, kuten Angry Birds, Clash of Clans ja Hill Climb Racing. Tunnettuja Unity3D-pelimoottoria käyttäviä pelejä ovat muun muassa Angry Birds-pelit, Hearthstone: Heroes of Warcraft ja Pokemon Go. Nämä pelit eivät ole pelikehityksen kannalta teknisesti vaativia pelejä ja jopa itsenäisillä kehittäjillä on mahdollisuus luoda näiden menestystarinoiden kaltaisia pelejä. Pelikehitys tarjoaa uramahdollisuuksia monille eri taiteen aloille kuten graafisille artisteille, musiikin tekijöille, näyttelijöille, suunnittelijoille sekä ohjelmoijille.

Tulevaisuudessa pelejä ja pelisuunnittelun opetuksia tullaan hyvin mahdollisesti käyttämään myös suurten yhteiskunnallisten haasteiden ratkaisemiseen maailman ensimmäisen pelillistämisen professori Juho Hamarin mukaan. Pelillistäminen on yksi pelikehityksen uusimmista suuntauksista, joka tarkoittaa peleistä tuttujen ratkaisumenetelmien, sekä tekniikoiden hyödyntämistä esimerkiksi virtuaalitodellisuuden simulaatioissa. Hamari tuo esiin, että virtuaalitodellisuudessa monien vaativien tehtävien harjoittaminen on pelien keinoin huomattavasti halvempaa ja turvallisempaa kuin oikeassa elämässä. (Ilkka-lehti 2016)

Pelimoottorin avulla pelinkehitystä pystyy nopeuttamaan ja helpottamaan huomattavasti verrattuna siihen, että pelin kaikki ominaisuudet kirjoitettaisiin alusta lähtien. Pelimoottorit sisältävät pelien perusmateriaalit, mutta itse sisältö, ulkonäkö sekä objektien toiminnallisuus täytyy kehittäjän itse luoda. Nämä ominaisuudet yhdessä muodostavat valmiin pelin. (Ward 2008, 1.)

1.2 Tavoitteet

Toteutuksena tehdyn mobiilipelin tavoitteena on käydä läpi mobiilipeleissä ja pelikehityksessä yleisesti esiintyviä ominaisuuksia, joiden varaan pystyy luomaan kokonaisvaltaisen pelikokemuksen. Toteutetun mobiilipelin on tarkoitus toimia esimerkkinä siitä, mitä Unity editoria ensimmäistä kertaa käyttävä voi saada aikaan sekä toimia myös mahdollisena opetusmateriaalin tukena Vaasan ammattikorkeakoulun peliohjelmointia käsittelevillä kursseilla. Tuotoksen skriptit on pyritty säilyttämään mahdollisimman helppolukuisina ja universaaleina, jotta ne olisi helpompi ymmärtää, jos ohjelmointi ei ole lukijalle tuttua.

Toteuteussa pelissä käytettävien skriptien ohjelmointikieleksi valittiin C#, koska se on suosituin ohjelmointikieli Unity3D-pelinkehitysyhteisössä. C# on myös yleisesti ottaen paremmin dokumentoitu ja suurin osa Unityn lisäosista Asset Storessa on tehty C# kanssa (Korhonen, A. 2016). C# kieltä opetetaan myös Vaasan ammattikorkeakoulun tietojenkäsittelyn koulutusohjelmaan kuuluvilla ohjelmointi kursseilla, joten kursseilla opitut taidot ja tekniikat tukevat hyvin toisiaan.

1.3 Rakenne

Luvussa kaksi esitellään lyhyesti pelinkehityksen historiaa, sekä esitellään pelialaa yleisesti Suomessa, sekä käydään läpi mitä pelimoottorit ovat. Kolmas luku käsittelee Unity Technologiesin tarjoamia pelinkehitystyökaluja. Luvussa käydään läpi uuden projektin luominen, esitellään editorin käyttöliittymässä esiintyvät näkymät, skriptauksessa käytettävät ohjelmointikielet ja lisäksi kerrotaan Unity learn-osiosta sekä Asset storesta. Luvussa neljä esitellään toteutettu peli ja analysoidaan käytettyjä skriptejä. Luvussa viisi on yhteenveto.

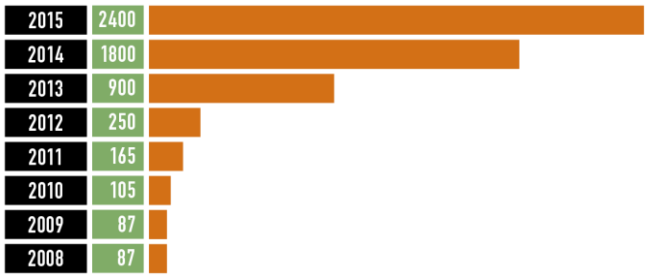
2 PELINKEHITYS

2.1 Historia

Pelien kehitys alkoi jo 1950-luvulla, mutta ensimmäiset kaupalliset pelit saapuivat kuluttajien käyttöön 1970-luvun alussa *Computer space* ja *Pong* pelien myötä arcade pelihalleihin. Ensimmäiset kotikäyttöön suunnitellut pelikonsolit olivat Magnavox Odyssey (1972) ja Atari (1972). Teknologiayritykset huomasivat nopeasti peliteollisuuden potentiaalin ja vuosina 1972–1985 yli 15 yritystä aloittivat pelikehityksen alati kasvaville markkinoille (Chikhani, 2015).

2.2 Pelinkehitys Suomessa

Peliteollisuus on ollut 2000-luvulta lähtien Suomen viihdeteollisuuden nopeimmin kasvava haara ja siten siitä on tullut merkittävä osa suomalaista kultturivientiteollisuutta. Suomen pelitoimialan ydintoimintojen liikevaihto oli Neogamesin mukaan vuonna 2015 noin 2,4 miljardia euroa. Edellisvuoteen verrattuna kasvua on tapahtunut noin 33 % ja vuosikymmenen alkuun verrattuna kasvua on tapahtunut noin 2520 %. (Neogames, 2016).



Kaavio 1. Suomen pelitoimialan liikevaihdon kehitys miljoonina (Neogames 2016)

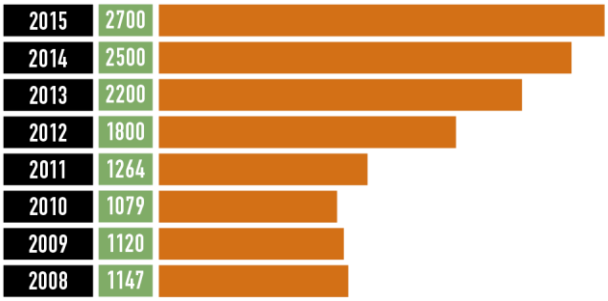
Suomi on tunnettu pioneerina mobiilipelien kehityksessä. Tähän vaikuttanee Nokian vahva läsnäolo, sekä kannettavan teknologian kehitys suomessa. Vaikuttavia tekijäitä suomalaisen pelikehityksen keskittymisestä mobiilialustoille ovat olleet mobiilialustojen helpohko kehitysympäristö, helposti tavoitettavat jakelukanavat,

kehittyneet kehitystyökalut sekä pienet lähtösijoitus summat. Neogamesin julkaiseman raportin mukaan jopa 85 % suomalaisista pelialan yrityksistä kehittää pelejä mobiilialustoille (Neogames Report 2014, 21-23. 2015).

Suomesta on noussut useita kansainvälisesti menestyneitä pelitaloja. Menestyneimpien pelikehittäjien joukkoon kuuluu Supercell, Fingersoft, Remedy Entertainment sekä Rovio Entertainment. Supercell nousi maailmanlaajuiseen suosioon mobiilipeleillään *Hay Day* ja *Clash of Clans*. Fingersoft tuli tunnetuksi yksinkertaisesta mobiilipelistään *Hill Climb Racing*. Rovio saavutti maailmanlaajuisen suosion *Angry Birds*-peleillään, josta on tehty myöhemmin myös elokuvia. Remedy taas tunnetaan *Max Payne*-pelisarjastaan ja tuottaa nykyisin AAA-luokituksen pelejä pc:lle ja konsoleille.

2.3 Pelialan koulutus ja työpaikat Suomessa

Peliteollisuuden nopean kasvun myötä osaksi maailman johtavia kulttuurisektorin teollisuudenaloja, on suomalaisen koulutusyhteisön pitänyt pystyä vastaamaan työmarkkinoiden alati kasvaviin tarpeisiin. Peliala työllisti vuonna 2015 Suomessa noin 2700 työntekijää yli 260 yrityksessä. Vuosikymmenen alusta pelialan työntekijämäärä on kasvanut noin 250 %. (Ammattinetti, Peliteollisuus 2016.)



Kaavio 2. Pelialan työntekijämäärien kehitys suomessa (Neogames, 2016)

Suomessa pelialan koulutusta on ollut mahdollista saada 2000-luvun alusta. Peliala ja sen tehtävien jatkuva kehitys tuottaa haasteita koulutusohjelmille. Vasta viime vuosina on alettu panostaa koulutusohjelmiin, jotka valmistavat pelialan eri osaamista vaativiin tehtäviin. Kasvavan teollisuudenalan vaatimuksiin suomessa on pyritty lisäämällä koulutusohjelmien aloituspaikkoja, sekä aloittamalla uusia pelialaan

liittyviä koulutusohjelmia eri koulutusasteilla. Erilaisia pelialaan liittyviä koulutusohjelmia on tarjolla toisen asteen oppilaitoksissa, ammattikorkeakouluissa sekä yliopistoissa ympäri Suomea. (Vähäkainu & Mononen & Neittaanmäki 4-6, 2014.)

2.4 Pelimoottorit

Pelimoottorilla tarkoitetaan ohjelmistokehystä jonka ympärille kehittäjät voivat rakentaa pelinsä. Pelimoottorin komponentteja ovat tavallisesti komentojen syöttäminen, grafiikkamoottori, äänimoottori, verkkoliikenne, fysiikkamoottori, GUI sekä valmiiksi kirjoitetut skriptit (Enger, 2013.) Pelimoottorien käyttö pelikehityksessä ei ole välttämätöntä, mutta niiden käyttö varsinkin itsenäisillä pelinkehittäjillä nopeuttaa pelinkehitystä sekä auttaa kehittäjää keskittymään sisällön tuottamiseen sekä pelin idean kehittämiseen. (Ward, 2008).

Markkinoilta on saatavilla monia erilaisia pelimoottoreita ja usein niistä on tarjolla ilmais- sekä kaupallinen lisenssi. Pelimoottoreita käyttävät niin suuret pelitalot kuin itsenäiset kehittäjätkin. Toiset pelimoottorit ovat graafisesti ja teknisesti vaativimpia kuin toiset. Tästä johtuen osa pelimoottoreista soveltuu hyvin itsenäiseen pelikehitykseen ja toiset taas suurten pelitalojen AAA-luokituksen peleille. (Ward, 2. 2008.)

Suosittuja pelimoottoreita indie-kehitysyhteisössä ovat Unreal Engine 4, GameMaker: Studio, Source, CryEngine sekä Unity. Kaikilla näistä on omat vahvuutensa ja heikkoutensa. Monissa eri pelikehittäjien yhteisöissä Unity on suositelluin pelimoottori ohjelmoinnin aloittelijoille.

3 UNITY3D

Unity3D kehitysympäristön työkalut ovat täysin ilmaisia kaikille kehittäjille joiden liikevaihto tai rahoitus on alle 100,000 dollaria. Unity Techonologies ei myöskään ota rojalteja pelien mahdollisista tuotoista. (Downie, 2016.) Pelikehittäjien ei tarvitse erikseen kehittää jollekin spesifille alustalle vaan Unityn editorin avulla pystyy pelin tai ohjelman kääntämään lähes kaikille tunnetuille alustoille (Unity Technologies, 2016a).

Unityn vetovoimaisimmat ominaisuudet uusia pelikehittäjiä ajatellen ovat Unity Techonologiesin tarjoamat dokumentaatiot, learn-osion tutoriaalit sekä ilmaiset verkossa olevat oppijaksot. Unityn dokumentaatiosta löytyy ilmaiseksi satoja artikkeleita ja videotutoriaaleja C# ja JavaScript(Unity Script)-ohjelmointikielille. Näistä palveluista on paljon apua niin vasta-alkajille kuin kehittyneemmillekin pelikehittäjille. Unity on tuottanut kaikki videotutoriaalit, joten ne ovat hyvin laadukkaita, selkeitä ja tasa laatuisia (Unity Technologies, 2016b). Näiden avulla uudetkin kehittäjät pystyvät helposti etenemään kohti monimutkaisempia pelimekaniikoita ja kehittämään ideaansa eteenpäin. Unityn taustalla toimii myös kattava Unity Asset Store. Asset storen tarjonnasta löytyy paljon valmiita ilmaisia sekä maksullisia lisäosia, jotka nopeuttavat pelinkehitystä todella huomattavasti.

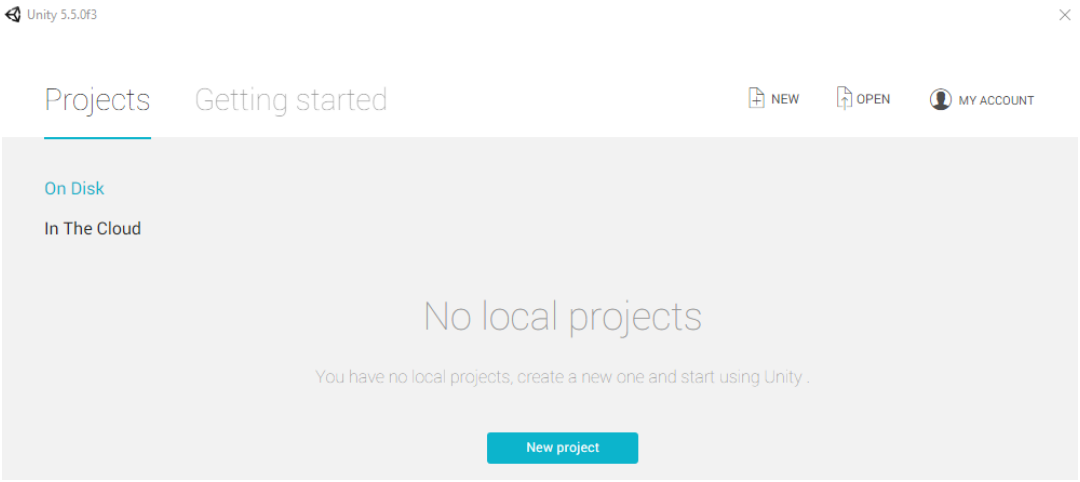
Unityyn 4.3 päivityksessä lisätty 2D-moottori mahdollistaa myös 2D-pelien kehittämisen Unityssä. 2D-pelien suosio onkin kasvanut huimasti viime vuosien aikana. 2D-pelien kehitys on yksinkertaisempaa kuin 3D-pelien ja ne vetoavat nostalgiallaan yleisöön joka tutustui pelaamiseen 90-luvulla. 2D-pelit myös toimivat erittäin hyvin mobiililaitteilla. Näistä ja monista muista syistä 2D-pelit ovat monien indiekehittäjien suosiossa ja usein kehittäjien ensimmäiset pelit ovatkin 2D-pelejä. (Totilo, 2010.)

3.1 Lataus / Asennus

Unity editorin voi ladata tietokoneelle osoitteesta https://store.unity.com. Sivulla näkyvät versiovaihtoehdot Personal, Plus, Pro ja Enterprise, joista peruskäyttöön valitaan useimmiten ilmainen Personal. Kun ohjelmisto on asennettu halutuin asetusten kanssa, kirjaudutaan ohjelmistoon sisään Unity-tunnuksilla tai luodaan ilmaiseksi uudet tunnukset.

3.2 Projektin luonti

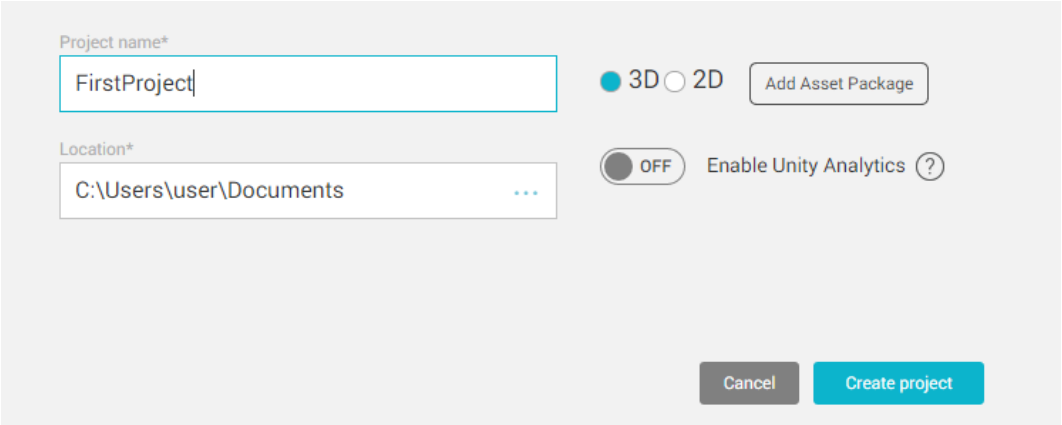
Asennuksen jälkeen ensimmäisellä käynnistyskerralla luodaan ilmaiset tunnukset Unity palveluun. Tunnukset toimivat niin itse ohjelmistossa kuin Unityn kotisivuilla ja Asset Storessa. Tunnusten luonnin jälkeen avautuu kuvassa 1. näkyvä ikkuna joka näyttää olemassa olevat projektit niin paikalliselta kovalevyltä kuin pilvessä olevalta levyltä. Pilvilevyn ansiosta on mahdollista avata omia projektejaan muuallakin kuin omalla tietokoneella.



Kuva 1. Unity-ohjelmiston aloitussivu.

Uutta projektia aloittaessa ohjelma kysyy, aloitetaanko projekti 3D:nä vai 2D:nä ja antaa mahdollisuuden lisätä haluttuja lisäosapaketteja. Projekti nimetään sekä valitaan haluttu tallennuskohde. Tässä kohtaa pystyy myös vaikuttamaan siihen, mitä

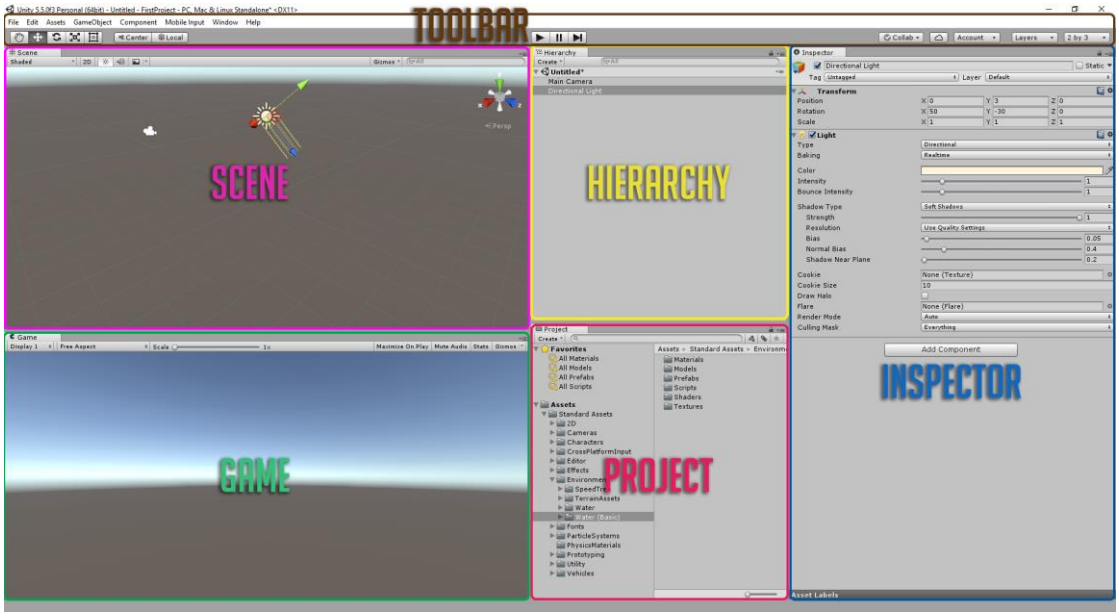
asset paketteja projektiin lisätään. Assetteja pystyy lisäämään projektiin myös projektin luomisen jälkeen. Kuvassa 2. näkyy projektin luomissivun asetukset.



Kuva 2. Projektin luomissivu.

3.3 Käyttöliittymä

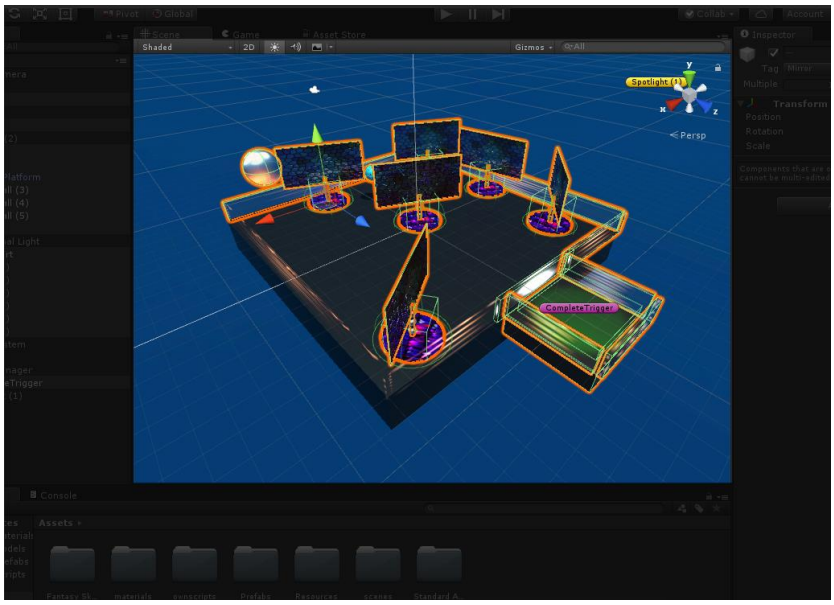
Käyttöliittymä koostuu tärkeimmistä näkymistä Unity-pelikehityksessä. Niitä ovat työkalurivi (Toolbar), kenttä- (Scene), peli- (Game), tarkastaja- (Inspector) ja projekti- (Project) sekä hierarkia-näkymä (Hierarchy). Käyttöliittymää voi tarpeidensa mukaan muokata ja omia käyttöliittymämalleja voi myös tallentaa. Valmiiksi muokattuja käyttöliittymämalleja voi valita Layout-pudotusvalikosta. Oletuksena käyttöliittymästä on piilotettu ääneen, animaatioon ja valaistukseen liittyvät näkymät. Kuvassa 3. on esiteltynä editorin 2 by 3 näkymä, joka soveltuu erinomaisesti kahden näytön työskentelyyn, jossa editori on toisella näytöllä ja skriptieditori toisella.



Kuva 3. Unity editorin 2 by 3 Käyttöliittymä

3.3.1 Scene-näkymä

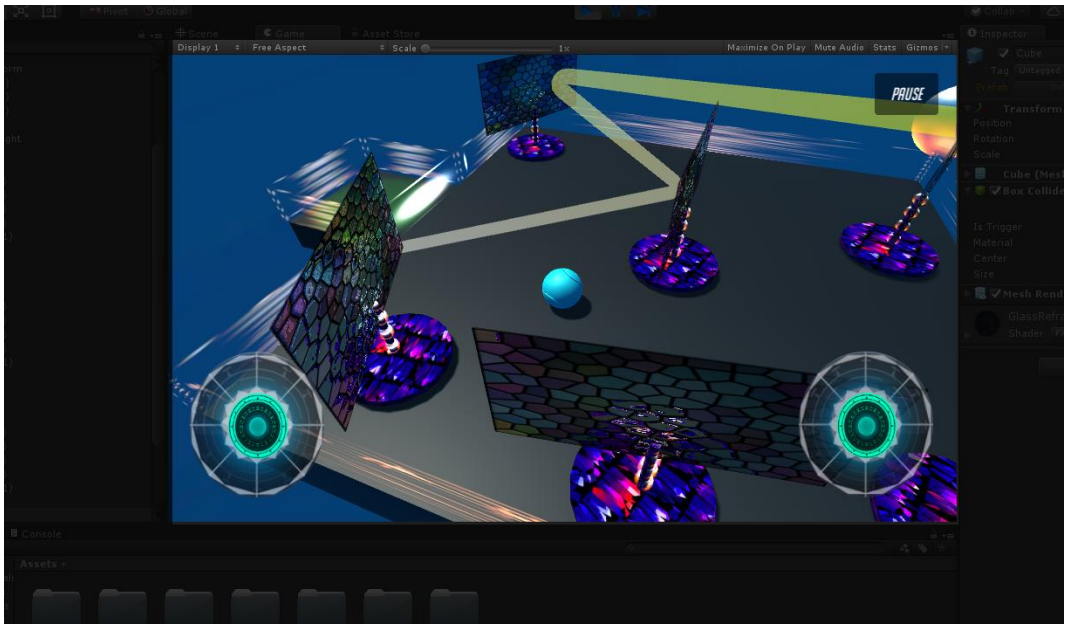
Scene-näkymä on pelin rakennusalusta. Näkymässä on mahdollista valita, siirtää ja muokata peliobjekteja. Peliobjekteja ovat esimerkiksi kenttärakenne, pelihahmot, kamerat ja valot. Scene-näkymässä toimiminen ja objektien manipuloiminen on ensimmäisiä asioita, joita täytyy opetella Unity-editoria käytettäessä. Kuvassa 4. esimerkki Scene-näkymästä. (Unity Technologies, 2016c.)



Kuva 4. Scene-näkymä.

3.3.2 Game-näkymä

Game-näkymän kuva on renderöity peliin asetetusta kamerasta tai kameroista. Näkymä kuvastaa, miltä scene-näkymässä luodun pelin lopullinen pelattava versio näyttää. Näkymää voi ohjata Toolbarista löytyvillä painikkeilla. Näkymässä on tarkoitus testata pelin ominaisuuksien toimivuutta käytännössä. Kuvassa 5. on esimerkki pelinäkymästä. (Unity Technologies, 2016d.)



Kuva 5. Game-ikkunan näkymä.

3.3.3 Project-näkymä

Tästä näkymästä löytyy projektissa käytettävissä olevat assetit. Assettejä projektiin voi tuottaa itse, ladata ilmaiseksi tai ostaa Unity Asset Storesta. Näkymästä pystyy lisäämään haluttuja assetteja suoraan Scene-näkymään uusiksi peliobjekteiksi tai lisäämään olemassa olevien peliobjektien komponenteiksi. Kuvassa 6. on esimerkki Project-näkymän asseteista. (Unity Technologies, 2016e.)

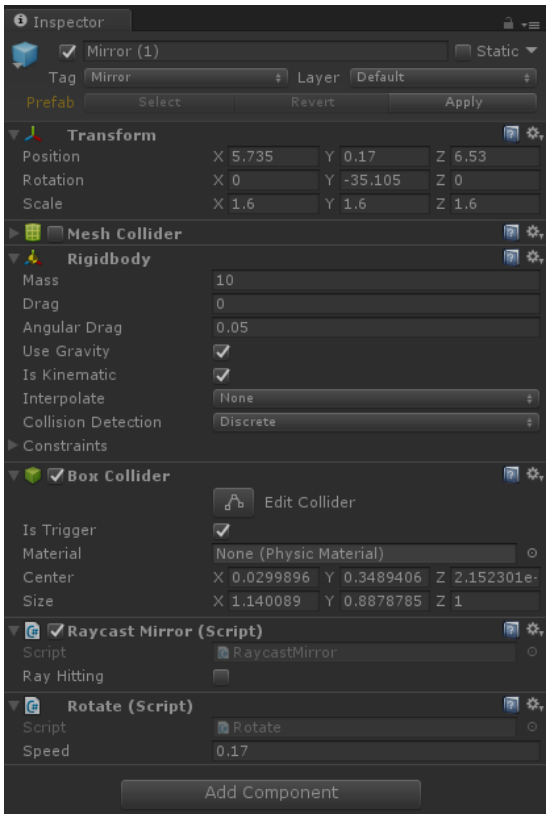
Kuva 6. Project-näkymä Hierarchy-näkymä.

Tässä näkymässä listataan jokainen peliobjekti muokattavana olevasta scenestä. Kun objekteja lisätään tai poistetaan scene-näkymässä, ilmestyvät ja poistuvat ne myös hierarchy-näkymässä. Objektit listataan hierarchy-näkymään niiden luomisjärjestyksen mukaan, mutta niitä voi myös uudelleen järjestää tai muokata objektien ”sukulaisuus” suhteita toisiinsa. Unityssä on mahdollista tehdä objekteista Parent-, Child- tai Sibling-objekteja. Child-objektit perivät Parent-objektin liike ominaisuudet. Kuvassa 7. on esimerkki hierarchy-näkymästä, jossa on käytetty sukulaisuussuhteita. (Unity Technologies, 2016f)

Kuva 7. Hierarchy näkymä jossa child-objekteja.

3.3.4 Inspector-näkymä

Inspector-näkymää käytetään näyttämään ja muokkaamaan peliobjektien sekä assettien komponentteja kuten fysiikkaa, skriptejä tai muita ominaisuuksia. Kun Hierarchy-näkymästä valitaan objekti, näkyvät sen ominaisuudet sekä kaikki objektin komponentit ja materiaalit sekä mahdollistaa niiden muokkaamisen. Kuvassa 8. näkyy Mirror-peliobjektin ominaisuudet. (Unity Technologies, 2016h)



Kuva 8. Mirror (1) peliobjektin ominaisuudet.

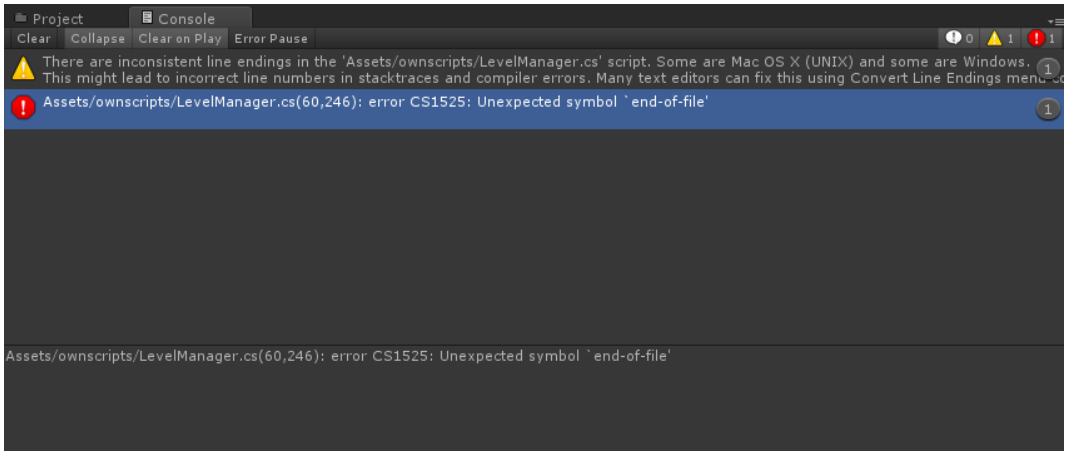
3.3.5 Toolbar-näkymä

Toolbar sisältää työkaluja jotka kukin vaikuttavat editorin eri ominaisuuksiin. Työkaluilla pystyy muokkaamaan esimerkiksi scene-näkymässä olevien objektien kokoa, sijaintia sekä suuntaa. Työkalujen avulla voi kontrolloida Game-näkymän etenemistä, vaihtamaan editorin käyttöliittymää ja hallitsemaan mitä objekteja näytetään Scene-näkymässä. Kuvassa 9. näkyy Toolbarin sijainti editorin perus asettelussa. (Unity Technologies, 2016f.)

Kuva 9. Unity editorin Toolbar

3.3.6 Console-näkymä

Console-näkymässä näkyy skripteissä tai editorissa tapahtuvia virheitä, varoituksia ja muita tarpeellisia tietoja debuggaukseen. Debug.log, Debug.LogWarking ja Debug.LogError-funktioiden avulla pystyy lähettämään console-näkymään informaatiota koodin tai pelin tapahtumista. Kuvassa 10. näkyy erroreita Console-näkymässä. (Unity Technologies, 2016i)



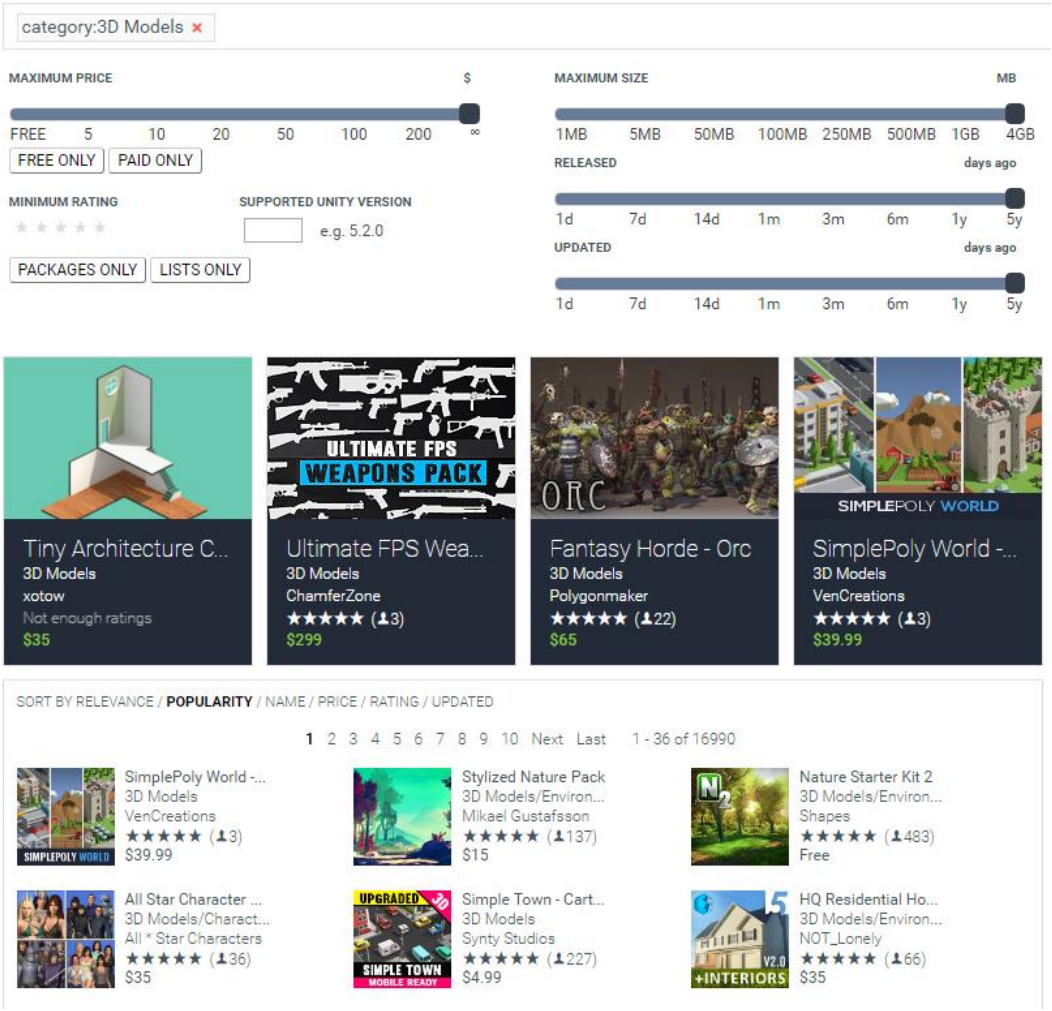
Kuva 10. Console-näkymä

3.4 Ohjelmointikielet

Ohjelmointikielistä Unity tukee C#, JavaScriptiä sekä vähemmän käytettyä Pythoniin pohjautuvaa Boo Scriptiä. C#, JavaScript ja Python ovat globaalisti hyvin suosittuja ohjelmointi-kieliä. Ne ovat kaikki kasvattaneet markkinaosuuksiaan ohjelmointikielien joukossa viime vuosina. (Cass, 2016.) Näitä ohjelmointikieliä käytetään yleisesti ohjelmoinnin opettamiseen, sillä ne ovat helppolukuisia, moderneja sekä ne ovat hyvin kysyttyjä nykypäivän teknologiayrityksissä (Bouwkamp, 2016).

3.5 Asset Store

Unity Technologiesin ylläpitämä Unity pelikehittäjille suunnattu verkkokauppa josta on saatavilla yhteisön tekemiä lisäosia Unityyn. Kauppaan tuottavat sisältöä monet pelikehityksen eri osa-alueiden ammattilaiset kuten graafikot, musiikin tekijät ja ohjelmoijat. Asset Storesta löytyy tuhansia erilaisiin projekteihin sopivia Assetpaketteja, jotka sisällöntuottajat ovat itse hinnoitelleet. Storesta löytyy esimerkiksi 3D-malleja, animaatioita, audiota, kokonaisten projektien templateja, partikkeliefektejä, skriptejä, shadereita sekä tekstuureja ja materiaaleja. Valmiit assetit ovat tehokas työkalu nopeuttamaan pelikehitystä.



Kuva 11. Unity Asset Store

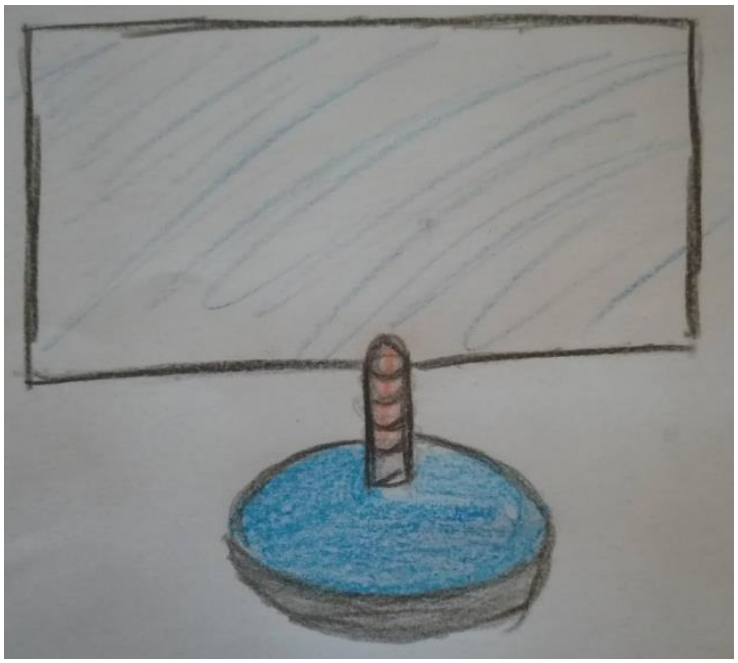
4 ”MIRRORS” MOBIILIPELI

”Mirrors” on yksinkertainen kolmiulotteinen pulmapeli. Pelissä liikutaan vierivällä pallolla ja tarkoituksena on heijastaa sädettä peileihin, kunnes kaikki peilit saavat osuman säteestä ja portti seuraavaan kenttään avautuu. Säde heijastuu peileistä riippuen kulmasta missä säde osuu peiliin. Peilejä käännellään liikkumalla pallolla peilin alapuolelle. Kun kaikki peilit saavat osuman samanaikaisesti, säteestä avautuu portti alueelle, josta kenttä läpäistään ja siirrytään seuraavaan kenttään.

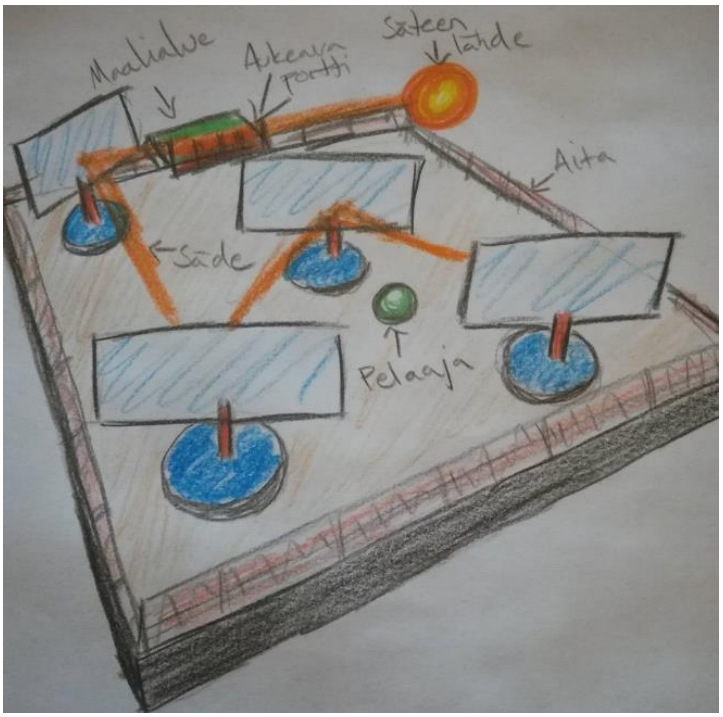
4.1 Suunnittelu

Videopelin suunnittelu tapahtuu samaan tyyliin kuin minkä tahansa visuaalisen teoksen kuten elokuvan. Suunnittelu ja sen dokumentointi ovat tärkeitä vaiheita pelikehityksessä sillä ne toimivat koko projektin pohjana. Pelin genre ja graafinen tyyli tulisi päättää heti projektin alussa. Hahmojen ja peliobjektien visuaalinen suunnittelu on tärkeä osa kehitystä, koska pelaaja on niiden kanssa jatkuvasti vuorovaikutuksessa. Usein kentistä, valaistuksesta, hahmoista ja erilaisista peliobjekteista tehdään konseptipiirroksia. Usein kaikista suunnittelun dokumenteista koostetaan Game Desing Document (GDD), jota käytetään peliprojektin hallinnassa. (Crosby 2011, 2.)

Pelin suunnittelu aloitettiin kokonaan säteen heijastamiseen peilien kautta pohjatuvan idean varaan. Tärkeimmät objektit olivat siis säde, peili, pelaaja sekä alue missä peilaaminen tapahtuu. Idea oli hyvin selkeä, joten aivan aluksi tein konseptipiirroksia paperille. Konseptipiirrosten avulla on helppo havainnollistaa, minkälaista lopputulosta editorilla halutaan tehdä. Kuvassa 12. on peiliobjektin konseptipiirros. Kuvassa 13. on valmiin kentän konseptipiirros.



Kuva 12. Peiliobjektin konseptipiirros



Kuva 13. Valmiin kentän konseptipiirros

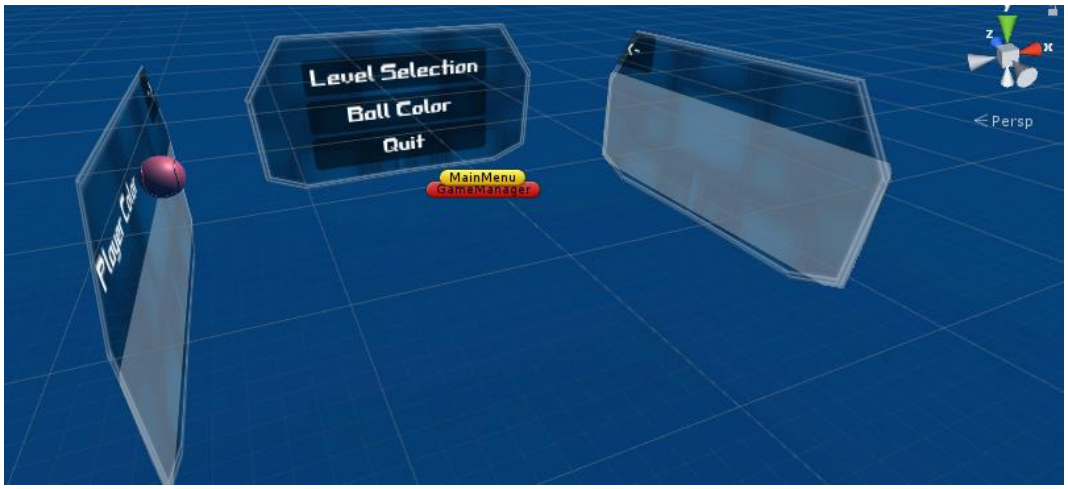
Peli-idean ollessa yksinkertainen pulmapeli tyyppinen ratkaisu jää suunnittelussa tärkeimmäksi osaksi kenttien suunnittelu (Davies, M. 2015). Opinnäytetyössä toteutetun pelin tarkoituksena ei ole tarjota tuntien pelielämystä, niin suunnittelin yksinkertaisen kentän jolloin pystyin keskittymään pääasiassa eri toimintojen ohjelmointiin. Kun peli on toiminnallisesti valmis, on helppo jatkaa pelin lisäkehitystä ja lisätä kenttien monimuotoisuutta sekä audiovisuaalisuutta.

4.2 Päävalikko-Scenen toteutus

4.2.1 Päävalikko

Pelin päävalikko-rakenne luotiin luomalla tyhjä peliobjekti, jonka nimi on UI Container. Tämä peliobjekti toimii valikko ikkunoiden isäntäobjektina, jolloin sen lapsiobjektit perivät sen Transform-ominaisuudet. UI Containerin lapsiobjekteiksi luotiin kolme samankokoista UI Panel-objektia, jotka toimivat näkyvinä valikko ikkunoina. Kuvassa 14. näkyy MainMenu-peliobjekti pelissä ja kuvassa 15. Menu-näkymä editorissa.

Kuva 14. Main Menu Game-näkymässä



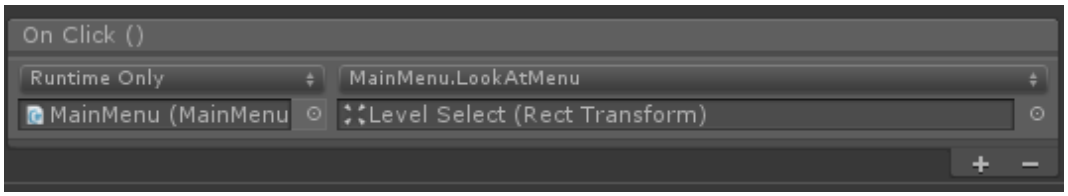
Kuva 15. Main Menun Scene-näkymässä

Valikon UI-button peliobjekteihin luodaan toiminnallisuutta MainMenu-skriptissä. MainMenu-skriptiin on tarkoitus saada mahdollisimman paljon menussa käytetyistä toiminnoista, jotta scenen kehittäminen pysyy siistinä ja selkeänä.

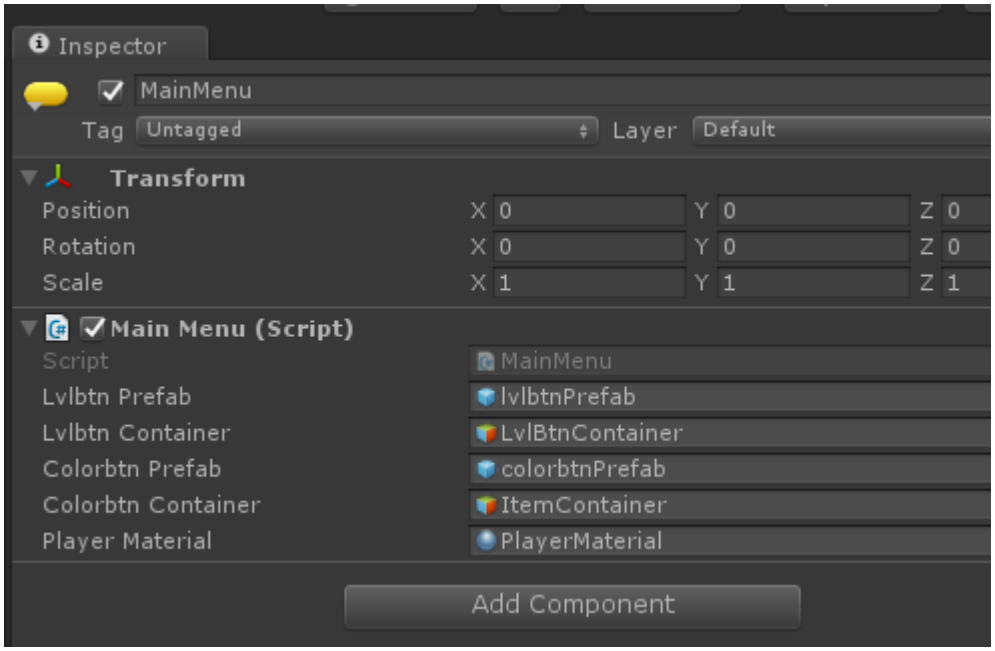
Quaternion rakenteen Slerp-funktio luo kameralle pallon muotoisen liikeradan jolloin kameran kääntymiseen saadaan kolmiulotteisuutta ja se liikkuu sulavasti (koodiesimerkki 1). MainMenu-skriptissä luodaan metodit joita käytettiin button-objektien On Click()-eventissä painikkeen Inspector-ikkunassa (Kuva 16). MainMenu-skripti lisätään MainMenu-peliobjektiin komponentiksi ja julkiset attribuutit lisätään Inspector-näkymässä (kuva 17). Painikkeen klikkaus kääntää kameran kohti valittua ikkunaa luoden valikkoon kolmiulotteista tunnelmaa. Ball Color ja Level Select-paneeleihin luotiin myös päävalikkoon palauttavat painikkeet.

```
private void Update()
{
    if(camDesiredLookAt != null)
    {
        camTransform.rotation = Quaternion.Slerp(camTransform.rotation, camDesiredLookAt.rotation,
                                                                        3 * Time.deltaTime);
    }
}
```

Koodiesimerkki 1. Kameran liikettä ohjaava Update metodi (MainMenu.cs)



Kuva 16. Level Select painikkeen On Click eventin tiedot.



Kuva 17. MainMenu-objektin Inspector-näkymä.

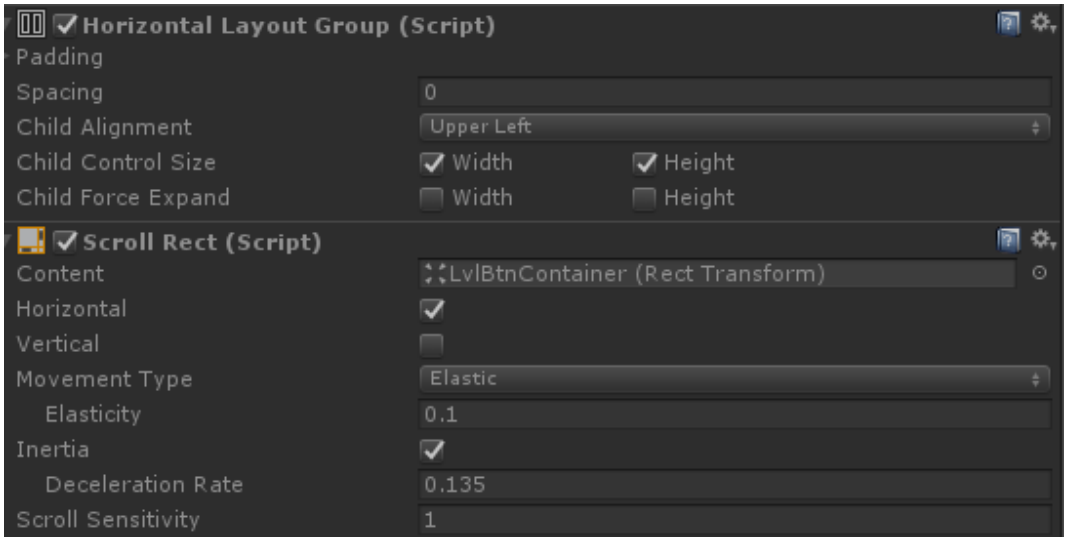
4.2.2 Kentän ja pallon värin valinta

Pelin käynnistyessä skripti hakee GetComponent-komennon avulla projektin kansioista Assets → Resources → Levels kaikki Sprite-luokan kuvat ja luo Array-taulukon ja niitä vastaavat painikkeet Level Select-paneelin sisällä olevan containerin sisään joka määrittää painikkeiden koon ja sijainnin (Koodiesimerkki 2). Tämä on hyvä ratkaisu kenttävalikon tekoon, koska uusia kenttiä tehdessä niitä ei tarvitse erikseen lisätä painike listaan, vaan skripti luo ne automaattisesti. MainMenu-skriptissä myös määritellään painikkeille On Click-tapahtuma, jolloin painiketta klikatessa siirrytään painikkeen nimeä vastaavaan sceneen eli tässä tapauksessa pelattavaan kenttään. Painikkeen Inspector-näkymässä on laitettu komponenttien asetuk-

set niin, että kenttä painikkeiden sijaintia voi liikuttaa pyyhkäysyllä (Kuva 18). Samaa toimintaa käyttävä skripti ja komponenttien asetukset on asetettu myös pallon värin valinnalle Options-ikkunassa. Kuvassa 19. on Level Select-ikkuna pelissä.

```
Sprite[] thumbnails = Resources.LoadAll<Sprite>("Levels");
foreach(Sprite thumbnail in thumbnails)
{
    GameObject container = Instantiate(lvlbtnPrefab) as GameObject;
    container.GetComponent<Image>().sprite = thumbnail;
    container.transform.SetParent(lvlbtnContainer.transform, false);

    string sceneName = thumbnail.name;

    container.GetComponent<Button> ().onClick.AddListener(() => LoadLevel(sceneName));
}
```

Koodiesimerkki 2. Kentänvalintapainikkeiden luontikoodi (MainMenu.cs)



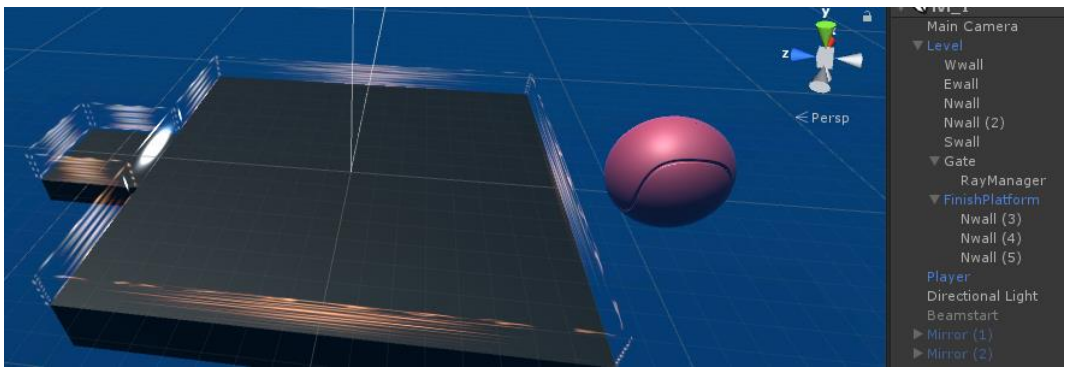
Kuva 18. Inspector-näkymän pyyhkimisen mahdollistavat asetukset.

Kuva 19. Kenttävalikko pelissä.

4.3 Kenttä-scenen toteutus

4.3.1 Pelialue ja pelaaja

Ensimmäisenä tyhjään kenttä-sceneen luotiin alue, missä pelaajalla on mahdollisuus liikkua ja mihin peilit sijoitetaan. Alusta sekä sitä ympäröivät seinät luotiin muokkaamalla useita kolmiulotteisia cube-peliobjekteja. Pelaaja luotiin Unityn valmiista assetista ”RollerBall”, joka on käytännössä hieman muokattu sphere-peliobjekti. Kuvassa 20. esitellään Level, Gate, FinishPlatform ja Player-peliobjektit sekä niiden lapsiobjektit.

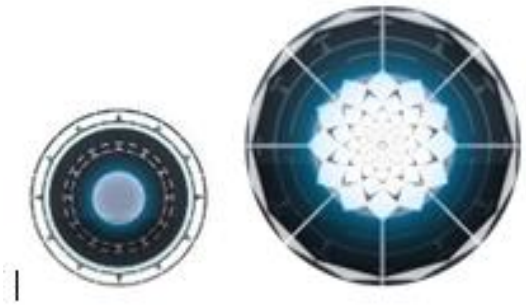


Kuva 20. Kenttä sekä pelaajaobjekti scene-näkymässä

4.3.2 Virtuaalinen joystick

Peli on suunniteltu mobiililaitteella pelattavaksi, joten tarvitaan ohjausjärjestelmä, joka toimii kunnollisesti pienellä kosketusnäytöllä. Virtuaalisen joystickin avulla mobiililaitteeseen saadaan lisättyä konsolilaitteiden ohjaimista tutut ”tatit” joilla ohjaus on riittävän tarkkaa ja nopeaa.

Virtual joystickin luomista varten luodaan tyhjä peliobjekti nimeltä ”GUI”, joka toimii isäntänä kaikille sceneen lisättäville UI-objekteille. Tulevia UI-objekteja ovat esimerkiksi joystickit, pysäytysvalikko sekä voittoruutu. GUI-peliobjektin avulla luodaan siis alusta, johon suoraan näytöllä näkyvät peliobjektit tullaan asettamaan. Virtual joystic rakennetaan kahdesta UI-image peliobjektista ja niiden toimintaa ohjaavasta skripteistä. Aluksi GUI-objektin lapseksi luodaan uusi UI-canvas peliobjekti, jonka lapseksi lisätään tässä tapauksessa UI-image peliobjekti nimeltä ”JoyBackground” joka toimii joystickin liikkumisalueena. JoyBackgroundin lapseksi lisätään UI-image peliobjekti ”JoyImg”, joka toimii joystickin liikkuvana osana (kuva 21). JoyBackground-objekti asetetaan GUI:n oikeaan alakulmaan käyttämällä anchor presets-valikkoa objektin Inspector-näkymässä. JoyImg-objekti asetetaan samaan paikkaan keskelle JoyBackground-objektia.



Kuva 21. Virtual joystickin osat JoyImg ja JoyBackground

Paikalleen asettelun jälkeen tehdään joystickin liikkumista ohjaava VirtualJoystic-skripti, joka sijoitetaan JoyBackground-objektin komponentiksi. Kyseisessä skriptissä määritellään tarvittavat peliobjektit, niiden liikkumien suhteessa toisiinsa

(Koodiesimerkki 3), sekä toimenpiteet kun kosketus alkaa, siirtyy sekä loppuu. VirtualJoystic-skripti myös laskee JoyImg-peliobjektin sijainnin suhteessa sille asetettuun nollapisteeseen (koodiesimerkki 4).

```
public virtual void OnDrag(PointerEventData ped)
{
    Vector2 pos;
    if (RectTransformUtility.ScreenPointToLocalPointInRectangle(bgImg.rectTransform, ped.position,
                                                                 ped.pressEventCamera, out pos))

    {
        pos.x = (pos.x / bgImg.rectTransform.sizeDelta.x);
        pos.y = (pos.y / bgImg.rectTransform.sizeDelta.y);

        inputVector = new Vector3(pos.x * 2 + 1, 0, pos.y * 2 - 1);
        inputVector = (inputVector.magnitude > 1.0f) ? inputVector.normalized : inputVector;

        joystickImg.rectTransform.anchoredPosition = new Vector3(inputVector.x * (bgImg.rectTransform.sizeDelta.x / 3)
                                                                 , inputVector.z * (bgImg.rectTransform.sizeDelta.y / 3));

       // Debug.Log(inputVector);
    }
```

Koodiesimerkki 3. skriptin osa joka laskee peliobjektien liikkumista.(VirtualJoystic.cs)

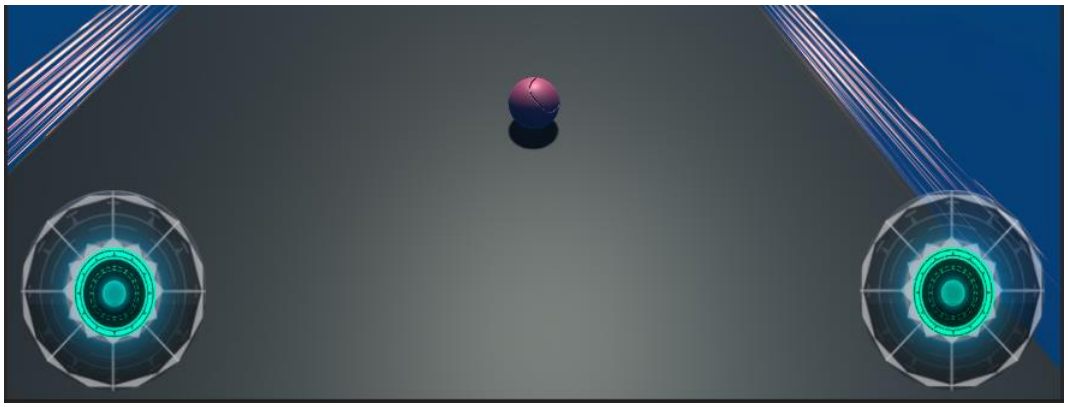
```
public float Horizontal()
{
    if (inputVector.x != 0)
        return inputVector.x;
    else
        return Input.GetAxis("Horizontal");

}

public float Vertical()
{
    if (inputVector.z != 0)
        return inputVector.z;
    else
        return Input.GetAxis("Vertical");
}
```

Koodiesimerkki 4. Metodit jotka palauttaa kosketuksen sijainnin.(VirtualJoystic.cs)

4.3.3 Pelaajan sekä kameran liikkuminen

Pelin kamera on toteutettu niin sanotulla vapaalla kolmannen persoonan kameralla. Kamera siis seuraa automaattisesti pelaajan liikkumista, mutta pelaajalla on mahdollisuus vaikuttaa kameran kuvakulmaan. Pelaaja myös liikkuu aina joystickin osoittamaan suuntaan riippumatta kameran sijainnista.

Pelaajan ja kameran kuvakulman liikuttaminen suoritetaan BallMotor- ja BallCamera-skripteillä, jotka lisätään molemmat Player-peliobjektin komponenteiksi ja joihin pelaaja vaikuttaa kahdella virtual joystick-peliobjektilla. Vasemmalla puolella olevalla joystickillä ohjataan kameran kuvakulmaa ja oikealla joystickillä ohjataan pallon liikettä suhteessa kameran kuvakulmaan (kuva 26).



Kuva 22. Joystickien sijainti pelin GUI:ssa

BallMotor-skripti noutaa VirtualJoystic-skriptistä julkisen VirtualJoystic-luokan avulla joystickien input datan ja niitä tietoja käytetään asettamaan voimaa Player-objektin fysiikkaan tai liikuttamaan kameran kuvakulmaa. VirtualJoystic-luokan ollessa julkinen, voidaan siihen Inspector-näkymän kautta lisätä molemmat tehdyt VirtualJoystic-peliobjektit. BallMotor-skriptissä luodaan Vector3-luokalle RotateWithView arvo ja aliohjelma joka tarkastaa kameran suunnan ja palauttaa Vector3-arvon nollaan kameran suunnan mukaisesti. Tämän avulla Vector3-luokan PoolInputin VirtualJoystic-luokalta saamalta arvot ovat normalisoituneet ja tällöin liike suoritetaan oikein, rippumatta kameran sijainnista (koodiesimerkki 5).

```
private Vector3 PoolInput()
{
     Vector3 dir = Vector3.zero;


    dir.x = JoyStick.Horizontal();
    dir.z = JoyStick.Vertical();

    if (dir.magnitude > 1)
    dir.Normalize();

return dir;
}

private Vector3 RotateWithView()
{
    if (CamTransform != null)
    {
        Vector3 dir = CamTransform.TransformDirection(MoveVector);
        dir.Set(dir.x, 0, dir.z);
        return dir.normalized * MoveVector.magnitude;
    }

    else
    {
        CamTransform = GetComponent<BallCamera>().CamTransform;
        return MoveVector;

    }

}
```

Koodiesimerkki 5. Suuntatietojen keräys pelaajan liikuttamiselle ja kameran siirtämiselle (BallMotor.cs).

Myös BallCamera-skripti käyttää hyväkseen VirtualJoystic-luokalta saatuja sijaintitietoja. Saatujen sijaintitietojen mukaan ”MainCamera” tagillä varustetua kameraa pystyy ohjaamaan VirtualJoystick-peliobjektin avulla (koodiesimerkki 6).

```
void Update ()
{
    currentX += JoyStick.Horizontal() * sensitivyX;
    currentY += JoyStick.Vertical() * sensitivyY;

    currentY = ClampAngle(currentY, Y_ANGLE_MIN, Y_ANGLE_MAX);
}

private void LateUpdate()
{
    Vector3 dir = new Vector3(0, 0, - distance);
    Quaternion rotation = Quaternion.Euler(currentY, currentX, 0);
    CamTransform.position = thisTransform.position + rotation * dir;
    CamTransform.LookAt(thisTransform.position);
}
```

Koodiesimerkki 6. Kameran liikkeen laskenta (BallCamera.cs).

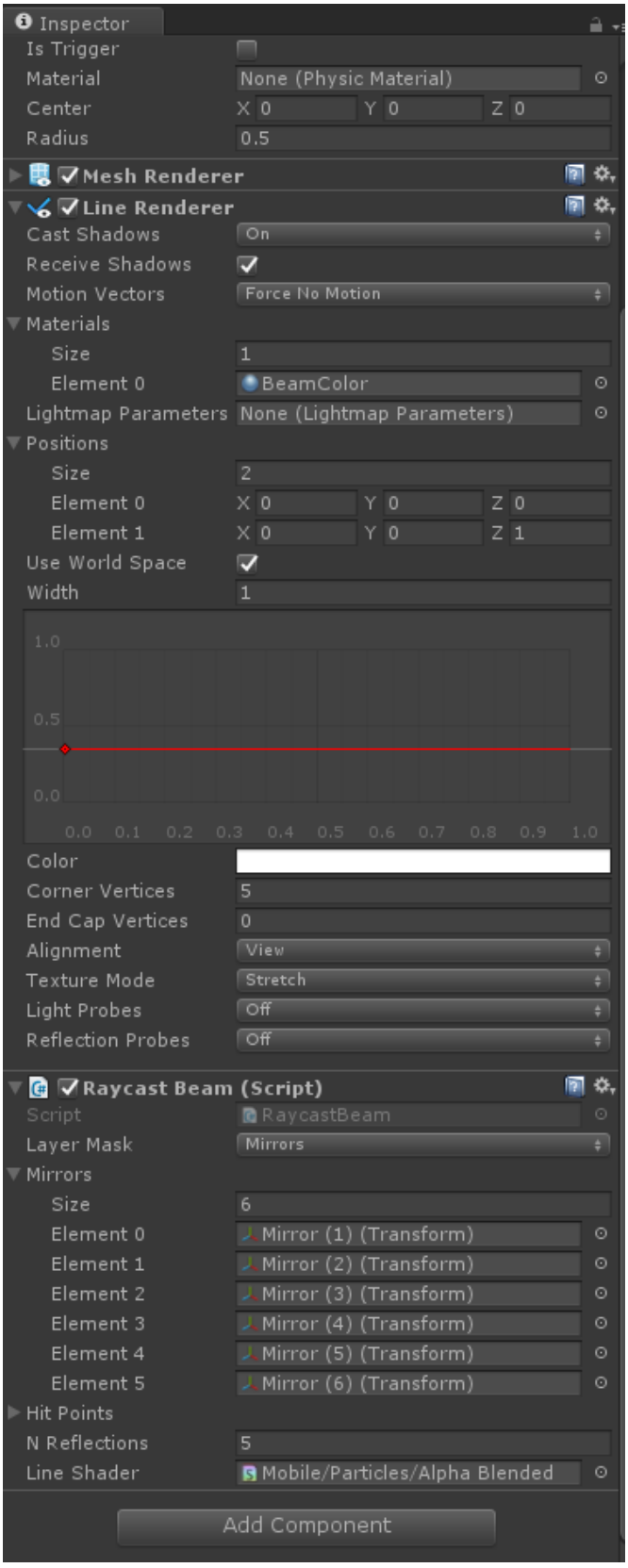
4.3.4 Säteen alkupiste

Tämän skriptin toimintaan saaminen halutulla tavalla yhdessä RayManager.cs skriptin kanssa oli työn ylivoimaisesti vaativin ja tärkein osuus, koska koko pelin idea rakentuu näillä skripteillä toteutettuihin toimintoihin. Beamstart-peliobjekti toimii alkupisteenä Unity-pelimoottorista löytyvälle Physics.Raycast-säteelle joka on itsessään näkymätön säde, joka palauttaa bool True arvon osuessaan toiseen peliobjektiin. Line renderer-komponentin avulla säteen kulkureitille pystytään piirtämään näkyvä säde. RaycastBeam-skripti on lisättynä Beamstart-peliobjektin komponentiksi. RaycastBeam-skripti aloittaa Raycast säteen lähettämisen, laskee osumiskohdan sekä aloittaa uuden säteen lähettämisen saapuneesta osumiskohdasta peilin osoittamaan suuntaan (koodiesimerkki 7). Säteen lähettämisen ja heijastamisen lisäksi RaycastBeam-skripti yhdessä RayManager-skriptin kanssa laskee, moneenko peiliin säde osuu samanaikaisesti. Kuvassa 23 esitellään Beamstart-objektin inspector ikkunassa näkyviä asetuksia.

```
for (int i = 0; i <= nReflections; i++)
{
    //If no ray reflect
    if (i == 0)
    {
        //Check ray hits
        if (Physics.Raycast(ray.origin, ray.direction, out hit, 20, layerMask))
        {
            // Calculate reflection direction
            inDirection = Vector3.Reflect(ray.direction, hit.normal);

            // Construct the new ray based on the direction and hit point of the previous one
            ray = new Ray(hit.point, inDirection);
```

```
            // Ray hits the object
            RayManager.rayManager.RayReflected(hit.transform);
            hitPoints.Add(hit.point);
        }
    }
    else // Ray has reflected
    {
        //Check ray hits
        if (Physics.Raycast(ray.origin, ray.direction, out hit, 20, layerMask))
        {
            inDirection = Vector3.Reflect(inDirection, hit.normal);

            ray = new Ray(hit.point, inDirection);
```

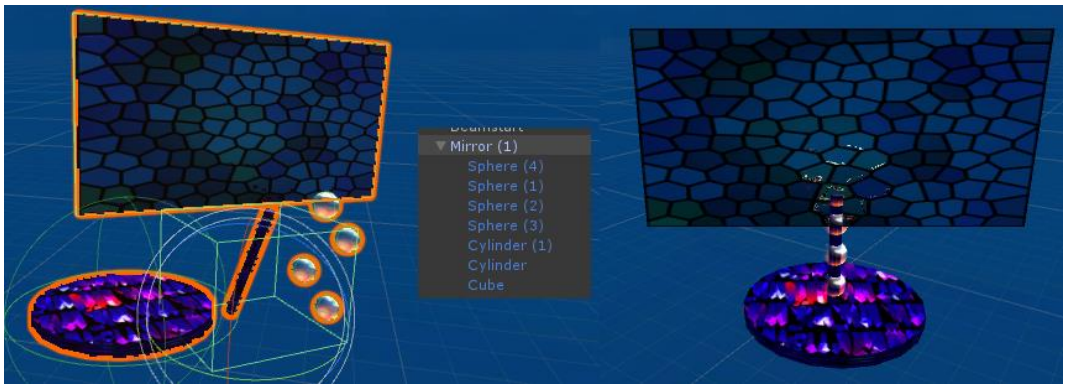
Koodiesimerkki 7. Säteen aloitus ja uuden säteen lähetys osumiskohdasta (RaycastBeam.cs).



Kuva 23. Beamstart-objektin Inspector-näkymä

4.3.5 Säteen peilaava peliobjekti

Sädettä eteenpäin peilaava Mirror-peliobjekti luotiin muokkaamalla useita geometrisista objekteista. Tämä tapa ei ole käytännöllinen suurissa projekteissa, joissa esiintyy paljon peliobjekteja, koska ne nostavat pelin vaatimaa suorituskykyä. Tämä ratkaisu on kuitenkin toimiva tämän tyylisessä projektissa ja oikean 3D-mallin tekemiseen ei ollut perusteita. Kuvassa 24 esitetään Mirror-peliobjektin rakenne.



Kuva 24. Mirror-objektin rakenne.

Kaikkiin muihin kentässä esiintyviin peliobjekteihin on asetettu Ignore Raycast Layer paitsi Mirror-peliobjektiin. Tämän avulla pystytään varmistamaan, että peilit ovat ainoita peliobjekteja jotka pystyvät heijastamaan sädettä. Mirror-peliobjektin komponentiksi lisätty Rotate-skriptin sisältämä OnTriggerStay-metodi aktivoi laukaisimen pelaajan koskettaessa peilin jalustaa. Rotate-skripti pyörittää peiliä hitaasti oman y-akselin ympäri transform.rotate-funktiota käyttäen niin kauan kuin pelaaja pysyy peilin trigger colliderin sisällä (koodiesimerkki 8).

```
public class Rotate : MonoBehaviour {

    public float speed;


    void OnTriggerStay ()
    {


        transform.Rotate(new Vector3(0, 1, 0) * speed);
    }
}
```

Koodiesimerkki 8. OnTriggerStay-metodi (Rotate.cs)

4.3.6 Maalialueen portti

RayManager-skripti säätelee maalialueelle ennenaikaisen kulun estävän Gate-peliobjektin aktivointia. RayManager-skripti tarkkailee, moneenko Mirror-peliobjektiin säde osuu yhtäaikaisesti. Säteen osuessa alle vaaditun määrän on seinä aktivoituna. Kun vaadittujen yhtäaikaisten osumien määrä on riittävän suuri, Gate-peliobjekti poistetaan. Jos kuitenkin pelaaja tekee virheen ja säde osumien määrä laskee, aktivoidaan Gate-peliobjekti uudelleen (koodiesimerkki 9).

```
public void RayReflected(Transform sender)
{
    // Make sure we don't get duplicate reflections from the same mirror
    if(!reflectionSenders.Contains(sender))
    {
        //Debug.Log("Reflection received from " + sender.gameObject.name);
        reflectionSenders.Add(sender);

        if (currentReflectedRayCount < requiredRayCount)
            currentReflectedRayCount += 1;
    }

    if (currentReflectedRayCount >= requiredRayCount)
    {
        if (OnReflectAll != null)
            OnReflectAll.Invoke();
    }
}
```

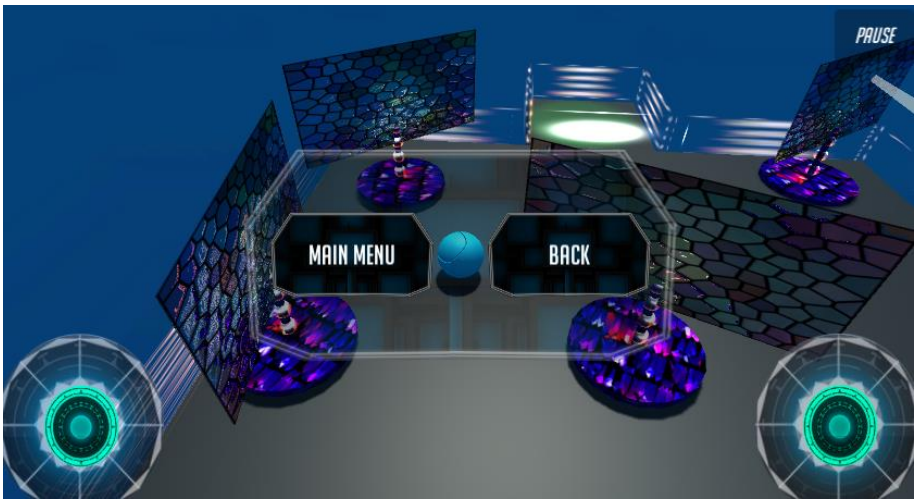
Koodiesimerkki 9. Heijastusten määrää tarkkaileva metodi (RayManager.cs)

4.3.7 Pause menu sekä painike.

GUI:n oikeaan ylälaitaan asetettu Pause-painike kutsuu OnClick-eventillä LevelManager-skriptistä TogglePause- sekä TogglePauseMenu-metodeja ja asettaa Time.timescale funktion arvoksi 0, joka pysäyttää pelin sisäisen ajan ja tuo esiin PauseMenu-objektin GUI:hin(koodiesimerkki 10). Tällöin pelaaja ei liiku ja pystyy ainoastaan käyttämään PauseMenu-objektissa olevia main menu sekä back-painikkeita. Main menu-painike vie pelaajan takaisin päävalikkoon ja pelin aika alkaa kulkea taas normaalisti, back-painikkeesta PauseMenu-objekti poistuu näkyvistä ja peli jatkuu normaalisti. Kuvassa 25 näkyy PauseMenu-objekti aktivoituna pelin sisällä.

```
public void TogglePauseMenu()
{

    pauseMenu.SetActive(!pauseMenu.activeSelf);

}
public void TogglePause()
  {
  paused = !paused;

  if (paused)
  {
      Time.timeScale = 0;
  }
  else if (!paused)
  {
      Time.timeScale = 1;
  }
```

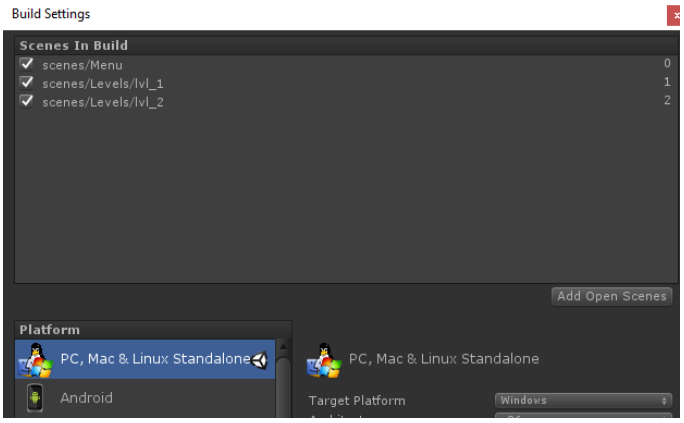
Koodiesimerkki 10. Kutsutut pause-metodit (MainMenu.cs)



Kuva 25. Pausemenu päällä pelissä.

4.3.8 Kentän läpäisy

Kentän läpäisy tapahtuu liikuttamalla pelaaja vihreällä valolla merkatulle alueelle. Alueella on tyhjä peliobjekti, johon on lisätty Box Collider ja siinä on Is Trigger-komponentti. Pelaajan osuttua laukaisimen alueelle suoritetaan LevelComplete-skripti, joka avaa ruudulle winWindow-objektin. Avautuvan winWindow-objektin toiminta on määritelty LevelManageri-skriptissä. Main menu-painikkeesta palataan päävalikkoon ja Next level-painikkeesta siirrytään projektin buildissa seuraavaksi asetettuun sceneen, tässä tapauksessa sceneen lvl_2. Kuvassa 26 näkyy projektin build-asetukset.



Kuva 26. Järjestys joka on asetettu Build Settings valikossa.

```
public void NextLevel()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```

Koodiesimerkki 11. Metodi jota kutsuttaessa vie seuraavaan sceneen. (LevelManager.cs)

4.4 Jatkokehitys

Pelin ollessa tällä hetkellä alusta asti täysin toimiva jää siihen silti erittäin paljon jatkokehitysmahdollisuuksia. Työssä en keskittynyt pelin audiovisuaaliseen puoleen juuri lainkaan, joten objektien materiaaleissa ja tekstuureissa on vielä paljon parantamisen varaa. Peliin ei ole lisätty äänimaailmaa ollenkaan, mikä on yksi tärkeistä osuuksista pelitunnelmaa luodessa.

Pelissä on tällä hetkellä vain kolme pelattavaa kenttää, mutta koska muu toiminnallisuus on tehty, on uusien kenttien rakentaminen nopeaa ja vaivatonta. Peliin voi tulevaisuudessa tehdä myös ominaisuuksia joilla pelaaja saadaan palaamaan pelin pariin aina uudestaan. Näitä ominaisuuksia ovat esimerkiksi pelihahmon kustomointi kauppapaikassa, kenttien läpipeluu aikaa vastaan sekä palkitseminen nopeasta ajasta.

5 YHTEENVETO

Työn päätavoitteena oli tuottaa alkuperäistä ideaa vastaava mobiilipeli, jossa käydään läpi peleissä esiintyviä toimintoja ja tekniikoita. Tavoite saavutettiin ja mobiilipelin kehitystaso vastaa mielestäni hyvin moniin mobiilipelikehityksessä esiintyvien haasteiden vaatimuksiin. Toisena tavoitteena oli oppia pelikehityksen aikana C#-ohjelmointikielen käyttöä ja voidaan todeta, että tuotettujen ohjelmakoodien esiintyminen editorissa ja interaktiivisina tapahtumina auttavat ymmärtämään koodissa tapahtuvia toimintoja.

Toteutetun mobiilipelin ollessa ensimmäinen minun itsenäisesti toteuttamani peliprojekti, on turvallista todeta, että Unity3D-pelimoottori ja editori yhdessä kattavan dokumentaation ja aktiivisen kehitysyhteisön avustuksella sopivat erinomaisesti pelinkehityksestä kiinnostuneille aloittelijoille.

Opinnäytetyö oli aiheena haastava ja mielenkiintoinen. Pelikehityksessä tehtävä ohjelmointi on jatkuvaa ongelmanratkaisua ja usein työtä tehdessä kului paljon aikaa pelkästään miettimään, millaisilla logiikoilla peliin haluttuja toimintoja voisi olla mahdollista toteuttaa.

Jatkossa aion jatkaa työssä toteutetun pelin kehittämistä myös opinnäytetyöprosessin jälkeen. Tavoitteena on saada peli vastaamaan itseni asettamia vaatimustasoja ja julkaista peli ilmaiseksi Google Play Storessa. Toiveena olisi myös jatkaa pelikehityksen parissa työelämässä.

LÄHTEET

Ammattinetti.fi, 2016. Peliteollisuus Viitattu 13.12.2016
http://www.ammattinetti.fi/ammattialat/detail/1a2d07830a65344600a33079becd6e88

Bouwkamp, K. 2016. The 9 Most In-Demand Programming languages of 2016
Viitattu 13.12.2016
http://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2016/

Cass, S. 2016 The 2016. Top Programming Languages Viitattu 13.12.2016
http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages

Chikhani R., 2015. The History Of Gaming: An Evolving Community
Viitattu 12.12.2016
https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/

Crosby, T. 2011. How Making a Video Game Works? Viitattu 09.01.2017
http://electronics.howstuffworks.com/making-a-video-game1.htm

Davies, Marsh. 2015. A Good Puzzle Game Is Hard To Build. Viitattu 09.01.2017
https://www.rockpapershotgun.com/2015/01/22/how-to-make-a-puzzle-game/

Downie, C. 2016. Evolution of our products and pricing Unity3D.com
Viitattu 13.12.2016
https://blogs.unity3d.com/2016/06/16/evolution-of-our-products-and-pricing/

Enger, M. 2013. Game Engines: How do they work? Viitattu 12.12.2016
http://www.giantbomb.com/profile/michaelenger/blog/game-engines-how-do-they-work/101529/

Ilkka 24.12.2016. s.44 Pelien konsteista apua suuriin ongelmiin. Viitattu 10.1.2017

Korhonen, A. 2016 Toimitusjohtaja. Viversio Oy. Sähköposti haastattelu 09.01.2017 Email anton.korhonen@viversio.com

Neogames.fi. 2016. Tietoa toimialasta Viitattu 12.12.2016
http://www.neogames.fi/tietoa-toimialasta/

Neogames The Game Industry Of Finland Report 2014, 2015. Viitattu 13.12.2016

http://www.neogames.fi/wp-content/uploads/2015/02/Neogames_report2015_full.pdf

Unity Technologies 2016a. Build once, deploy anywhere Unity3D.com Viitattu 13.12.2016
https://unity3d.com/unity/multiplatform

Unity Technologies 2016b. Unity3D.com Viitattu 13.12.2016
https://unity3d.com/learn

Unity Technologies 2016c The Scene view. Viitattu 13.12.2016
https://docs.unity3d.com/Manual/UsingTheSceneView.html

Unity Technologies 2016d. The Game view. Viitattu 13.12.2016
https://docs.unity3d.com/Manual/GameView.html

Unity Technologies 2016e. The Project Window. Viitattu 13.12.2016
https://docs.unity3d.com/Manual/ProjectView.html

Unity Technologies 2016f. The Hierarchy window. Viitattu 13.12.2016
https://docs.unity3d.com/Manual/Hierarchy.html

Unity Technologies 2016g. The Inspector window. Viitattu 13.12.2016
https://docs.unity3d.com/Manual/UsingTheInspector.html

Unity Technologies 2016h. The Toolbar. Viitattu 13.12.2016
https://docs.unity3d.com/Manual/Toolbar.html

Unity Technologies 2016i. Console Window. Viitattu 10.01.2017
https://docs.unity3d.com/Manual/Console.html

Totilo, S. 2010. The Revenge Of 2D Viitattu 13.12.2016
http://kotaku.com/5577352/the-revenge-of-2d

Vähäkainu, P. & Mononen, L. & Neittaanmäki, P. Jyväskylän Yliopisto. 2014.
Viitattu 13.12.2016
https://www.jyu.fi/it/tutkimus/suomen_pelialan_koulutus

Ward, J. 2008. What is Game Engine? Viitattu 12.12.2016
http://www.gamecareerguide.com/features/529/what_is_a_game_.php

Wu, B. 2013. Top 10 Reasons to Choose Unity 3D for App and Game development Viitattu 12.12.2016
http://www.pepwuper.com/top-10-reasons-to-choose-unity-3d-for-app-and-game-development/

LIITE 1. Pelidemon pc-version latauslinkki

https://mega.nz/#!pcVV2IzB!b6P6AMmt7JRwRuUXfN1Dvmel7RpeIqh-N7iUG0HV2YI

LIITE 2. Pelin toteuttamisessa käytetyt skriptit

1. BallMotor.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BallMotor : MonoBehaviour {

    public float moveSpeed = 5.0f;
    public float drag = 0.5f;
    public float terminalRotationSpeed = 25.0f;
    public Vector3 MoveVector { set; get; }
    public VirtualJoystick JoyStick;

    private Rigidbody thisRigidBody;
    private Transform CamTransform;

        void Start ()
    {
        thisRigidBody = gameObject.AddComponent<Rigidbody>();
        thisRigidBody.maxAngularVelocity = terminalRotationSpeed;
        thisRigidBody.drag = drag;

        }


    void Update()
    {
        MoveVector = PoolInput();

        MoveVector = RotateWithView();

        Move();
    }

    private void Move()
       {
           thisRigidBody.AddForce((MoveVector * moveSpeed));
       }

    private Vector3 PoolInput()
    {
         Vector3 dir = Vector3.zero;



        dir.x = JoyStick.Horizontal();
        dir.z = JoyStick.Vertical();

        if (dir.magnitude > 1)
        dir.Normalize();

    return dir;
    }
```

(JATKUU)

LIITE 2. (jatkoa)

```
 private Vector3 RotateWithView()
    {
        if (CamTransform != null)
        {
            Vector3 dir = CamTransform.TransformDirection(MoveVector);
            dir.Set(dir.x, 0, dir.z);
            return dir.normalized * MoveVector.magnitude;
        }

        else
        {
            CamTransform = GetComponent<BallCamera>().CamTransform;
            return MoveVector;

        }

    }

}
```

2. BallCamera.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BallCamera : MonoBehaviour {

    private const float Y_ANGLE_MIN = 0.0f;
    private const float Y_ANGLE_MAX = 180.0f;

    private Transform thisTransform;
    private Camera cam;
    private float distance =10.0f;
    private float currentX = 0.0f;
    private float currentY = 0.0f;
    private float sensitivyX = 1.0f;
    private float sensitivyY = 1.0f;

    public VirtualJoystick JoyStick;
    public Transform CamTransform { set; get; }


        void Start ()
    {
    CamTransform = new GameObject("Camera Container").transform;
    cam = CamTransform.gameObject.AddComponent<Camera>();
    cam.tag = "MainCamera";

    thisTransform = transform;
```

```
        }
```

(JATKUU)

LIITE 2. (jatkoa)

```
// Update is called once per frame
        void Update ()
    {
        currentX += JoyStick.Horizontal() * sensitivyX;
        currentY += JoyStick.Vertical() * sensitivyY;

        currentY = ClampAngle(currentY, Y_ANGLE_MIN, Y_ANGLE_MAX);
    }

    private void LateUpdate()
    {
        Vector3 dir = new Vector3(0, 0, - distance);
        Quaternion rotation = Quaternion.Euler(currentY, currentX, 0);
        CamTransform.position = thisTransform.position + rotation * dir;
        CamTransform.LookAt(thisTransform.position);
    }

    private float ClampAngle(float angle, float min, float max)
    {
        do
        {
            if (angle < -360)
                angle += 360;
            if (angle > 360)
                angle -= 360;
        } while (angle < -360 || angle > 360);

        return Mathf.Clamp(angle, min, max);
    }
}
```

3. GameManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameManager : MonoBehaviour {

    public static GameManager instance;
    public static GameManager Instance { get; set; }

    public int playerColor = 0;


        private void Awake()
    {
        DontDestroyOnLoad(gameObject);
        instance = this;



    }
```

LIITE 2. (jatkoa)

4. LevelManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelManager : MonoBehaviour
{
    public static LevelManager instance;
    public static LevelManager Instance;

    public GameObject pauseMenu;
    public GameObject winWindow;

    private void Start()
    {
        pauseMenu.SetActive(false);
        winWindow.SetActive(false);

    }

    public void TogglePauseMenu()
    {
        pauseMenu.SetActive(!pauseMenu.activeSelf);
    }
        public void ToMenu()

        {
            SceneManager.LoadScene("Menu");
        }

     public void ToggleWinWindow()
        {
        winWindow.SetActive(!winWindow.activeSelf);
        }

    public void NextLevel()
        {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex +1);
        }
    }
        public void TogglePause()
        {
        paused = !paused;

        if (paused)
        {
            Time.timeScale = 0;
        }
        else if (!paused)
        {
            Time.timeScale = 1;
        }
}
```

LIITE 2. (jatkoa)

5. LevelComplete.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class LevelComplete : MonoBehaviour {

    public GameObject winWindow;

    void Start()
    {
        winWindow.SetActive(false);
    }

        private void OnTriggerEnter()
    {
        winWindow.SetActive(!winWindow.activeSelf);
    }
}
```

6. MainMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour {

    public GameObject lvlbtnPrefab;
    public GameObject lvlbtnContainer;
    public GameObject colorbtnPrefab;
    public GameObject colorbtnContainer;

    public Material playerMaterial;

    private Transform camTransform;
    private Transform camDesiredLookAt;

    private void Start()
    {
        Time.timeScale = 1;
        //starts to use the main camera at its position
        camTransform = Camera.main.transform;

        //Searches level thumbnail images and makes them buttons
        Sprite[] thumbnails = Resources.LoadAll<Sprite>("Levels");
        foreach(Sprite thumbnail in thumbnails)
        {
            GameObject container = Instantiate(lvlbtnPrefab) as GameObject;
            container.GetComponent<Image>().sprite = thumbnail;
```

(JATKUU)

LIITE 2. (jatkoa)

```
   container.transform.SetParent(lvlbtnContainer.transform, false);


            string sceneName = thumbnail.name;

            container.GetComponent<Button> ().onClick.AddListener(() => LoadLevel(scene-
Name));
        }
        //Searches player color thumbnail images and makes them buttons
        int textureIndex = 0;
        Sprite[] textures = Resources.LoadAll<Sprite>("Player");
        foreach (Sprite texture in textures)
        {
            GameObject container = Instantiate(colorbtnPrefab) as GameObject;
            container.GetComponent<Image>().sprite = texture;
            container.transform.SetParent(colorbtnContainer.transform, false);

            int index = textureIndex;
            container.GetComponent<Button>().onClick.AddListener(() => ChangePlay-
erColor(index));
            textureIndex++ ;
        }


    }

    private void Update()
    {
        if(camDesiredLookAt != null)
        {
            camTransform.rotation = Quaternion.Slerp(camTransform.rotation, camDesiredLoo-
kAt.rotation,
                                                                                    3 *
Time.deltaTime);
        }
    }

    private void LoadLevel(string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }

    public void LookAtMenu(Transform menuTransform)
    {
        camDesiredLookAt = menuTransform;
    }

    public void QuitGame()
    {
        Application.Quit();
    }

    public void ChangePlayerColor(int index)
    {
        float x = (index % 4) * 0.25f;
        float y = ((int)index / 4) * 0.25f;
        if (y == 0.0f)
            y = 0.75f;
        else if (y == 0.25f)

            y = 0.5f;
        else if (y == 0.50f)
            y = 0.25f;
        else if (y == 0.75f)
```

```
            y = 0f;
```

(JATKUU)

LIITE 2. (jatkoa)

```
            playerMaterial.SetTextureOffset("_MainTex", new Vector2(x,y));
        PlayerPrefs.SetInt("playerMaterial", index);
    }
}
```

7. RaycastBeam.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RaycastBeam : MonoBehaviour
{
    public LayerMask layerMask;
    private LineRenderer lineRenderer;
    private Ray ray;
    private RaycastHit hit;
    private Vector3 inDirection;
    public List<Transform> mirrors;
    public List<Vector3> hitPoints;
    public int nReflections = 2;
    public Shader lineShader;


    void Awake()
    {
        lineRenderer = this.GetComponent<LineRenderer>();
        hitPoints = new List<Vector3>();
    }

    void Start()
    {
        lineRenderer.material = new Material(lineShader);
        lineRenderer.startColor = new Color(1f, 1f, 0.03f, 0.5f);
        lineRenderer.endColor = new Color(1f, 1f, 1f, 0.5f);
        lineRenderer.startWidth = 0.9f;
        lineRenderer.endWidth = 0.2f;
    }

    void Update()
    {
        nReflections = Mathf.Clamp(nReflections, 1, nReflections);
        ray = new Ray(transform.position, transform.forward);

        // Clear all the linerenderer positions
        hitPoints.Clear();

        // Add the initial ray to the linerenderer
        hitPoints.Add(transform.position);



        RayManager.rayManager.RemoveReflections();

        for (int i = 0; i <= nReflections; i++)
        {
            //If no ray reflect
            if (i == 0)
            {
                //Check ray hits
                if (Physics.Raycast(ray.origin, ray.direction, out hit, 20, layerMask))
                {
                    // Calculate reflection direction
                    inDirection = Vector3.Reflect(ray.direction, hit.normal);
```

(JATKUU)

LIITE 2. (jatkoa)

```
                    // Construct the new ray based on the direction and hit point of the
previous one
                    ray = new Ray(hit.point, inDirection);



                    // Ray hits the object
                    RayManager.rayManager.RayReflected(hit.transform);
                    hitPoints.Add(hit.point);
                }
            }
            else // Ray has reflected
            {
                //Check ray hits
                if (Physics.Raycast(ray.origin, ray.direction, out hit, 20, layerMask))
                {
                    inDirection = Vector3.Reflect(inDirection, hit.normal);

                    ray = new Ray(hit.point, inDirection);



                    // Ray hits the object
                    RayManager.rayManager.RayReflected(hit.transform);
                    hitPoints.Add(hit.point);
                }
            }
        }

        // Set linerenderer position count as hitpoint count
        lineRenderer.numPositions = hitPoints.Count;

        for (int r = 0; r < hitPoints.Count; r++)
        {
            lineRenderer.SetPosition(r, hitPoints[r]);
        }
    }
}
```

8. RaycastMirror

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RaycastMirror : MonoBehaviour {

    public bool rayHitting = false;
        // Use this for initialization
        void Start () {

        }

        // Update is called once per frame
        void Update () {

        }
}
```

LIITE 2. (jatkoa)

9. RayManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class RayManager : MonoBehaviour {

    public static RayManager rayManager;
    public int requiredRayCount = 6;
    private int currentReflectedRayCount = 0;
    public UnityEvent OnReflectAll;
    public UnityEvent OnReflectionCancelled;
    public List<Transform> reflectionSenders;

    void Awake()
    {
        if (rayManager == null)
            rayManager = this;

        reflectionSenders = new List<Transform>();
    }

    public void RayReflected(Transform sender)

 {
        // Make sure we don't get duplicate reflections from the same mirror
        if(!reflectionSenders.Contains(sender))
        {
            //Debug.Log("Reflection received from " + sender.gameObject.name);
            reflectionSenders.Add(sender);

            if (currentReflectedRayCount < requiredRayCount)
                currentReflectedRayCount += 1;
        }

        if (currentReflectedRayCount >= requiredRayCount)
        {
            if (OnReflectAll != null)
                OnReflectAll.Invoke();
        }
    }

    public void RemoveReflections()
    {
        reflectionSenders.Clear();
        currentReflectedRayCount = 0;
        OnReflectionCancelled.Invoke();
    }
}
```

LIITE 2. (jatkoa)

10.Rotate.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rotate : MonoBehaviour {

    public float speed;


        void OnTriggerStay ()
    {


        transform.Rotate(new Vector3(0, 1, 0) * speed);
        }
}
```

11.VirtualJoystick.cs

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

public class VirtualJoystick : MonoBehaviour, IDragHandler, IPointerUpHandler, IPointer-
DownHandler
{
    private Image bgImg;
    private Image joystickImg;
    private Vector3 inputVector;

    private void Start()
    {

     bgImg = GetComponent<Image>();
        joystickImg = transform.GetChild(0).GetComponent<Image>();
    }

    public virtual void OnDrag(PointerEventData ped)
    {
        Vector2 pos;
        if (RectTransformUtility.ScreenPointToLocalPointInRectangle(bgImg.rectTransform,
ped.position,
                                                                    ped.pressEventCam-
era, out pos))

        {
            pos.x = (pos.x / bgImg.rectTransform.sizeDelta.x);
            pos.y = (pos.y / bgImg.rectTransform.sizeDelta.y);

            inputVector = new Vector3(pos.x * 2 + 1, 0, pos.y * 2 - 1);
            inputVector = (inputVector.magnitude > 1.0f) ? inputVector.normalized : input-
Vector;

            joystickImg.rectTransform.anchoredPosition = new Vector3(inputVector.x *
(bgImg.rectTransform.sizeDelta.x / 3)
```

```
                                                                    , inputVector.z *
(bgImg.rectTransform.sizeDelta.y / 3));
```

(JATKUU)

```
            // Debug.Log(inputVector);
        }
    }
    public virtual void OnPointerDown(PointerEventData ped)
    {
        OnDrag(ped);
    }

    public virtual void OnPointerUp(PointerEventData ped)
    {
        inputVector = Vector3.zero;
        joystickImg.rectTransform.anchoredPosition = Vector3.zero;

    }

    public float Horizontal()
    {
        if (inputVector.x != 0)
            return inputVector.x;
        else
            return Input.GetAxis("Horizontal");

    }

    public float Vertical()
    {
        if (inputVector.z != 0)
            return inputVector.z;
        else
            return Input.GetAxis("Vertical");
    }

}
```