Mobiilipelin kehittäminen Androidille Unreal Engine 4 -peli-moottorilla

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeen ammattikorkeakoulu, Tietojenkäsittelyn koulutusohjelma

Hämeenlinna, kevät 2018

Roope Kuikka

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Visamäki, Hämeenlinna

Tekijä Roope Kuikka **Vuosi** 2018

Työn nimi Mobiilipelin kehittäminen Androidille Unreal Engine 4 -pelimoottorilla

Työn ohjaaja Tommi Saksa

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli kehittää mobiilipeli Android-käyttöjärjestelmälle käyttäen Unreal Engine 4 -pelimoottoria. Tutkimusongelmat, joihin opinnäytetyössä keskityttiin, olivat pelin kontrollointi mobiililaitteen kosketusnäytöllä, toimivien ja selkeiden valikoiden luominen ja pelin skaalautuvuus erikokoisille näytöille.

Tutkimusongelmiin haettiin vastauksia perehtymällä Unreal Engine 4 -pelimoottorin dokumentointiin, mobiilipelien kehittämisen teoriaan sekä kehittämällä yksinkertainen mobiilipeli, joka painottuu tutkimusongelmien osa-alueisiin. Opinnäytetyö on jaettu teoria- ja käytännön osuuteen. Teoriaosuudessa kerrotaan mobiilipelien kehittämisestä, C++-ohjelmointikielestä yleisesti ja perustietoja Unreal Engine 4 -pelimoottorista. Käytännön osuudessa käydään läpi peliin luotuja ominaisuuksia käyttäen apuna luotuja ohjelmakoodeja ja ruutukaappauksia Unreal Engine -pelimoottorin sisältä.

Työtä aloittaessa Unreal Engine 4 ja sen käyttämä C++-ohjelmointikieli olivat työn tekijälle tuntemattomia. Opinnäytetyön tavoitteisiin päästiin ja peliin saatiin luotua suunnitellun mukaiset kosketusnäyttökontrollit, valikot, ja peli saatiin skaalautumaan halutulla tavalla. Opinnäytetyötä tehdessä Unreal Engine todettiin melko vaikeaksi ja hitaaksi kehitysympäristöksi sisäistää, ja osa suunnitelluista ominaisuuksista jäi puuttumaan.

Avainsanat pelinkehitys, mobiilipeli, Unreal Engine, Android

Sivut 27 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Visamäki, Hämeenlinna

Author Roope Kuikka **Year** 2018

Subject Developing a Mobile Game for Android Using Unreal Engine 4 –Game Engine

Supervisor Tommi Saksa

ABSTRACT

The aim of this thesis was to develop a mobile game for Android using Unreal Engine 4 -game engine. The main research problems in this thesis were controlling mobile game with touchscreen, creating working and clear menus and fitting the game for varied sizes and aspect ratios of screens.

The methods used to solve the research problems were studying documentation of Unreal Engine 4, theory of mobile game development and developing a simple mobile game, focusing on areas of research problems. The thesis is divided into theoretical and practical parts. The theoretical part consists of mobile game development, general info about C++-programming language and the basics of Unreal Engine 4. The practical part discusses features added to the game and gives information about how said features were created, showing programmed C++-code and screenshots from inside the game engine.

In the starting point of the project, the author of the thesis had no experience of Unreal Engine 4 or C++-programming language it uses. The goals of the thesis were met and the features of touch controlling, menus and scaling to varied sizes and aspect ratios were implemented in the game as planned. While developing the game, Unreal Engine 4 was found to be quite difficult and slow to learn, which resulted in some of the planned features not getting implemented.

Keywords game development, mobile game, Unreal Engine, Android

Pages 27 pages

SISÄLLYS

SANASTO

Pelimoottori	Pelimoottorit ovat ohjelmistoja joilla voi luoda pelejä. Pelimoottorit sisältävät usein valmiiksi esim. fysiikat, renderöinnin, äänet ja tekoälyn.
Unreal Engine	Pelimoottori, jota tulen käyttämään mobiilipelin kehittämisessä.
Android	Ensisijaisesti kosketusnäyttöisille älypuhelimille ja tablettitietokoneille suunniteltu Linux-pohjainen käyttöjärjestelmä.
Skripti	Ohjelmointikielellä kirjoitettu koodi, jolla voidaan määrittää pelin toimintaa, esimerkiksi objektien liikkeitä.
C++	Ohjelmointikieli, jolla Unreal Enginessä kirjoitetaan skriptit.
Kasuaalipelit	Kasuaalipelit (Casual games) ovat pelejä, joita on helppo oppia pelaamaan. Kuka tahansa voi pelata niitä, vaikka vain muutaman minuutin ajan.
Resoluutio	Kertoo, kuinka monta kuvapistettä (pikseliä) näytöllä on.
Sprite	Kaksiulotteisia pikseligrafiikka kuvia, joista voi muodostaa 2D animaatioita.

1 JOHDANTO

Älypuhelimien ja muiden mobiililaitteiden ohella myös mobiilipelaaminen on yleistynyt ja kehittynyt paljon viime vuosien aikana. Tässä opinnäytetyössä kerron aluksi yleisesti mobiilipelien kaupallistumisesta, erilaisista genreistä ja kehitysympäristöistä eli pelimoottoreista, joita on kehitetty helpottamaan pelien luomista. Opinnäytetyön alkuosassa kerron tarkemmin muutamasta mobiilipeleille yleisestä ongelmasta tai huomioon otettavasta asiasta, joihin tulen keskittymään mobiilipelin kehittämisessä. Opinnäytetyössä kerrotaan myös hieman C++-ohjelmointikielestä, jota tulen käyttämään pelin ohjelmoimisessa.

Opinnäytetyön tavoitteena oli luoda mobiilipeli Android-käyttöjärjestelmälle käyttäen Unreal Engine -pelimoottorin versiota 4.18. Peli on yksinkertainen ja helposti lähestyttävä 2D-peli. Valitsin Unreal Enginen pelinkehitysalustaksi, koska sen oppiminen kiinnostaa minua ja minulla ei ole siitä ennestään minkäänlaista kokemusta. Pelin ohjelmointi tapahtui Unreal Engine -pelimoottorissa käyttäen C++-ohjelmointikieltä, josta minulla ei myöskään ollut aiempaa kokemusta. Pelin kehityksessä keskityin erityisesti pelin kontrollointiin kosketusnäyttöä käyttäen, valikoiden käyttäjäystävällisyyteen ja selkeyteen ja pelin skaalautuvuuteen eri kokoisille ja resoluutioisille näytöille.

Opinnäytetyössä selvitettävät tutkimuskysymykset ovat:

Kuinka luoda Androidille mobiilipeli, joka käyttää kosketusnäyttöä pelin kontrollointiin Unreal Engine 4:llä?

Kuinka luoda käyttäjäystävällinen ja toimiva valikko mobiilipelille Unreal Engine 4:llä?

Kuinka luoda eri resoluutioille ja ruudun kuvasuhteille skaalautuva mobiilipeli Unreal Engine 4:llä?

2 MOBIILIPELIEN KEHITTÄMINEN

Mobiilipelit ovat mobiililaitteilla, erityisesti älypuhelimilla ja tablettitietokoneilla pelattavia pelejä. Mobiilipelit ovat viime vuosien aikana yleistyneet todella paljon, ja älypuhelimien kehityttyä ne kykenevät pyörittämään yhä suurempia ja raskaampiakin pelejä. Mobiilipelejä löytyy laidasta laitaan, monenlaisille kohderyhmille. Vuonna 2016 mobiilipelien tuotot olivat ensimmäistä kertaa suuremmat kuin konsoli- ja PC-peleillä. (Chan 2017.)

2.1 Mobiilipelien kaupallisuus

Mobiilipeleillä on useita erilaisia mahdollisia ansaintamalleja, joista pelinkehittäjä voi valita yhden omalle pelilleen sopivimman tai yhdistellä useaa eri ansaintamallia. Oikean mallin valitseminen voi olla yksi pelinkehittäjän vaikeimmista päätöksistä. Monet mobiiliapplikaatiot ja -pelit ovatkin epäonnistuneet vain huonon ansaintamallin valinnan takia. (Malhotra 2016.)

Erilaisia ansaintamalleja ovat maksulliset pelit, kokeiluversiot, pelin sisäiset ostot ja pelin sisäinen mainostaminen. Maksulliset pelit ovat nimensä mukaisesti pelejä, joista joutuu maksamaan ennen kuin niitä pääsee käyttämään ollenkaan. Alusta alkaen maksulliset pelit ovat nykyään melko harvinaisia, koska suuri osa potentiaalisista pelaajista luultavasti jättää pelin kokonaan kokeilematta, jos sitä ei voi testata ennen ostopäätöksen tekemistä. Hieman yleisempää on se, että pelistä on erillinen ilmainen kokeiluversio, josta puuttuu ominaisuuksia tai se sisältää mainoksia ja täyden version pelistä saa päivittämällä maksettuun versioon. Tätä kahden erillisen version niin kutsuttua freemium-mallia käytti muun muassa huippusuosioon noussut Angry Birds -mobiilipeli. (Munir 2014.)

Tämän hetken ehdottomasti suosituin ja eniten tuottavin bisnesmalli on pelin sisäiset ostot, jota käytetään lähes poikkeuksetta kaikissa eniten tuottoa tehneissä mobiilipeleissä (Think Gaming 2018). Tällaisissa peleissä pelaajilla on mahdollisuus tehdä pieniä ostoja edistääkseen omaa peliä jollakin tavalla tai avaamaan uusia ominaisuuksia peliin, ja addiktoituneimmat pelaajat saattavatkin käyttää satoja tai jopa tuhansia euroja huomaamattaan. Pelin sisäisiä ostoja sisältäviin peleihin lisätään usein myös pelin sisäistä mainontaa, joissain peleissä se saattaa olla ainoa ansaintakeino. Tällöin pelin tuotto perustuu mainostuloihin, jotka muodostuvat siitä, että käyttäjät näkevät esimerkiksi mainosbannereita tai lyhyitä mainosvideoita. (Malhotra 2016.)

2.2 Mobiilipelien genret ja kohderyhmät

Mobiilipelit eivät ole kaikki samanlaisia, eivätkä ne kaikki ole suunniteltu samalle kohdeyleisölle. Pelejä voidaan jakaa kymmeniin erilaisiin genreihin

ja monet pelit sopivatkin usein useampaan kuin yhteen genreen. Kolme yleisintä ja menestyneintä mobiilipeli genreä ovat puzzle-, eli pulmapelit, toiminta-/strategiapelit ja kasuaalipelit. Pulmapelit ovat pelejä, joissa pelaajan tulee ratkaista vaikeudeltaan erilaisia tasoja edetäkseen pelissä. Hyvä esimerkki pulmapeleistä on Kingin kehittämä Candy Crush Saga. King oli yksi ensimmäisistä firmoista, jotka toivat mobiilipelit todella suureen suosioon. Toiminta- ja strategiapelit eivät ole yhtä suosittuja mobiililla, kuin ne ovat pelikonsoleilla. Siitä huolimatta ne ovat saaneet kaapattua suuren osan markkinoista itselleen. Hyvä esimerkki strategia pelistä on suomalaisen pelifirman Supercellin kehittämä Clash of Clans, jossa pelaajan on tarkoitus rakentaa ja puolustaa omaa kylää, hyökäten samalla muiden pelaajien kyliin. Kasuaalipelit on genre, johon suurin osa mobiilipeleistä enemmän tai vähemmän kuuluu. Kasuaalipelit vaativat vain vähän taitoa pelata ja niiden pääasiallinen tarkoitus on hetkellinen hauskanpito ilman suurempaa sitoutumista peliin. Kasuaalipeleihin kuuluvat esimerkiksi myös pulmapeleiksi lasketut Candy Crush Saga ja Angry Birds. (HDDmag 2017.)

Mobiilipelien suuri kasvu ympäri maailmaa on johtanut siihen, että yhä useammat ihmiset pääsevät pelaamaan, eikä pelaaminen rajoitu vain konsoli ja PC pelaajille. Pelaajien usein ajatellaan olevan joko nuoria, usein miespuolisia pelaajia, jotka pelaavat monimutkaisia, paljon taitoa vaativia pelejä, tai sitten vanhempia naisia, jotka pelaavat kasuaalipelejä silloin tällöin. Suurin osa sijoittuu kuitenkin johonkin näiden kahden ääripään väliltä. Vaikka mobiilipelien kohderyhmien laajuus on suuri mahdollisuus tehdä monenlaisia pelejä, on se samalla myös haaste pelinkehittäjille innostaa ja lähestyä niin suurta yleisöä. (Bevans 2017.)

2.3 **Mobiilipelien kehitysympäristöt**

Mobiilipeleistä on nykyään paljon tarjontaa ja pelien kilpailu on todella kovaa, joten onnistuakseen luomaan jotain suurta on mobiilipelin oltava todella hyvin suunniteltu ja toteutettu. Pelinkehittämisprosessia nopeuttamaan on luotu monia eri kehitysympäristöjä eli pelimoottoreita, jotka helpottavat esimerkiksi grafiikoiden renderöintiä, fysiikoiden ohjelmointia ja äänien lisäämistä peliin. Pelimoottoreita löytyy useita kymmeniä erilaisia, moniin eri käyttötarkoituksiin. Pelimoottoreiden hinnat vaihtelevat ilmaisista useisiin satoihin euroon kuukaudessa. (ThinkMobiles 2017.)

Osassa kehitysympäristöistä ei tarvitse ollenkaan ohjelmointia, vaan ne ovat kokonaan visuaalisia ”raahaa ja pudota” -tyyppisiä työkaluja. Tämän kaltaisilla työkaluilla pystyy hyvin oppimaan perusteita pelinkehityksestä ja lähes kuka tahansa kykenee niitä kokeilemaan, mutta niillä on lähes mahdotonta onnistua luomaan mitään uniikkia menestyspeliä. Esimerkkejä tämän tyyppisistä pelimoottoreista ovat Scirra Ltd:n Construct 2 ja Clickteamin Fusion, joista on molemmista olemassa ilmainen kokeiluversio. (ThinkMobiles 2017.)

Aloitteleville pelinkehittäjille on useita sopivia kehitysympäristöjä, jotka sisältävät ohjelmointia, mutta ovat muuten mahdollisimman helppokäyttöisiä. Tällaisia pelimoottoreita ovat esimerkiksi Corona SDK ja Game Maker. Nämä sopivat hyvin pienempien indie-peliprojektien kehittämiseen, mutta suurien firmojen käytettäväksi ne eivät skaalaudu kovin hyvin. (ThinkMobiles 2017.)

On olemassa myös monia raskaampia ja monimutkaisempia pelimoottoreita, joilla voi kehittää monien muiden mahdollisten alustojen lisäksi myös mobiilipelejä, ja ne soveltuvat suurienkin firmojen käytettäväksi. Tällaisia raskaampia kehitysympäristöjä ovat esimerkiksi Unity, MonoGame ja Epic Gamesin Unreal Engine, jota tullaan käyttämään tässä opinnäytetyössä. Vaikka nämä ovatkin hieman monimutkaisempia, raskaampia ja vaativat enemmän osaamista ohjelmoinnista, ei se tarkoita, että ne eivät olisi aloittelijaystävällisiä, sillä suurimmalle osalle näistä kehitysympäristöistä löytyy todella paljon dokumentointia, oppaita ja niillä on suuri käyttäjäkunta. Näiden suurempien kehitystyökalujen käytön hinta määräytyy usein ohjelmistoa käyttävän firman koon tai tulojen mukaan. (ThinkMobiles 2017.)

2.4 **Mobiilipelien kontrollointi**

Pelikokemuksen kannalta tärkein tekijä grafiikoiden lisäksi on pelin kontrollointi. Pelaajat eivät tule viihtymään pelin parissa, jos pelin kontrolleja ei ole helppo oppia, intuitiiviset tai eivät vastaa pelaajan liikkeisiin. Nykyaikaisissa mobiililaitteissa pelien kontrollointimahdollisuuksia ovat kosketusnäyttö ja erilaiset sormien liikkeet näytöllä, näppäimet, älypuhelimen sisäänrakennetut laitteet sekä ulkoiset ohjaimet. (Scolastici & Nolte 2013, 162.)

Vanhemmissa puhelimissa pelejä on perinteisesti kontrolloitu käyttämällä puhelimen fyysistä näppäimistöä, ja juurikaan muunlaisia mahdollisuuksia ei pelien kontrollointiin ollut. Puhelimien kehityttyä tukemaan kosketusnäyttöjä ja muita sisäänrakennettuja sensoreita, harvassa uudessa puhelimessa on enää ollenkaan näppäimiä puhelimen etupuolella, joten pelitkään eivät niitä enää usein tue. Luonnollinen kehitys näppäinkontrolleista oli siirtyä tukemaan kosketusnäyttöä, jota suurin osa nykyaikaisista mobiilipeleistä käyttää. Kosketusnäytön avulla pelejä voi ohjata eri tavoilla, käyttämällä erilaisia sormien liikkeitä näytöllä, kuten esimerkiksi napautusta, pyyhkäisyä, pitkään painamista, tuplanapautusta tai kahden sormen multitouch-liikkeillä, kuten näytön nipistämisellä, kääntämisellä, kahden sormen napautuksella ja skrollaamisella. (Scolastici & Nolte 2013, 163–170.)

Kosketusnäyttöjen lisäksi peleissä voidaan käyttää myös älypuhelimen sisäänrakennettuja sensoreita ja muita laitteita. Todella monet pelit käyttävät hyväkseen älypuhelimien gyroskooppia, jolla pystyy havaitsemaan puhelimen liikkeitä, kuten sen kääntelyä tai ravistelua. Osa peleistä käyttää puhelimen GPS-signaaleja pelaajan paikantamiseen luodakseen kokemuksen lisätystä todellisuudesta eli AR:stä (Augmented Reality), jossa pelaaja

kokee itse olevansa pelissä mukana. Myös puhelimista löytyviä kameroita voidaan käyttää AR-kokemuksen luomiseksi, jolloin pelin objektit näkyvät olevan oikeaan maailmaan sekoittuneena. Hyvä esimerkki AR-pelistä on vuonna 2016 julkaistu Nianticin kehittämä Pokémon GO, joka käyttää hyväkseen sekä kameraa että GPS-signaaleja AR-kokemuksen luomiseksi. Ei niin suosittu mutta yksi mahdollinen pelin ohjaustapa on puhelimen mikrofonin käyttäminen esimerkiksi puheentunnistamiseen tai puhaltamisen voimakkuuden havaitsemiseen, jota käytetään esimerkiksi applikaatioissa, jotka esittävät jonkin puhallinsoittimen soittamista. (Scolastici & Nolte 2013, 172–178.)

Älypuhelimille on myös kehitetty yhteensopivia ulkoisia ohjaimia, jotka yhdistyvät puhelimeen bluetooth-yhteydellä. On myös kehitetty laitteita, joilla saa perinteisien konsolien peliohjaimia yhdistettyä mobiililaitteisiin. Tällaisiin ohjaimiin yhteensopivuuden ohjelmoiminen saattaa kuitenkin vaatia paljon lisätyötä, joten pelinkehittäjän tulee miettiä, onko se tarpeellista. Pelkän ohjainkontrolloimisen tukeminen ei kuulosta hyvältä idealta, koska suurimmalla osalla mobiilipelaajista ei ohjaimia löydy, joten se sulkisi suuren osan potentiaalisista pelaajista heti pois. (Scolastici & Nolte 2013, 178–179.)

2.5 **Skaalautuvuus eri resoluutioilla ja kuvasuhteilla**

Android-järjestelmää käyttäviä mobiililaitteita on olemassa todella suuri määrä ja samoin on myös eri kokoisia ja tarkkuuksisia näyttöjä. On tärkeää, että peli tukee mahdollisimman montaa erilaista näyttöä, jotta pelkkä näytön vaatimus ei rajaa suurta määrää potentiaalisista pelaajista pois. Resoluutiot vaihtelevat 480 x 854 pikselistä aina 1600 x 2560 pikseliin asti. Kuvasuhteet ovat useimmiten 16:9, 16:10 tai 4:3. (Material Design n.d.)

Eri laitteiden näyttöjen erot saattavat aiheuttaa vaikeuksia mobiiliapplikaatioiden ja -pelien kehittäjille, mutta kehitysympäristöjen ja pelimoottorien kehittäjät ovat onneksi tiedostaneet tämän, ja pelimoottoreihin on usein valmiiksi kehitetty työkalut sovelluksen skaalautuvuuden säätöön. Esimerkiksi Unreal Engine 4 -pelimoottorissa on käyttöliittymätyökalu nimeltä Widget Blueprint, jolla valikoiden skaalautuvuutta voi säätää (Epic Games 2017b). Silti pelinkehittäjän täytyy peliä suunnitellessaan ja kohdealustoja miettiessään pitää mielessään mahdolliset erikokoiset näytöt pelin käyttäjäystävällisyyden kannalta.

3 C++-OHJELMOINTIKIELI

Tietokoneet ovat kykeneviä suorittamaan mitä monimutkaisempia ohjelmia ja tehtäviä. Jotta tietokone suoriutuisi oikein tehtävästään, täytyy sen ihmisten luomia ohjelmia apunaan käyttäen tietää tarkalleen mitä tehdä. Tietokoneiden ymmärtämä kieli ei ole mitään ihmisten kieltä muistuttavaa, joten ihmiset ovat kehittäneet laajan valikoiman erilaisia ohjelmointikieliä. Ohjelmointikielten avulla pystymme muuttamaan ihmisille helpommin opittavaa kieltä muotoon, jota tietokone ymmärtää. Projektin tarpeista riippuen, on monia eri tekijöitä mitkä vaikuttavat oikean ohjelmointikielen valintaan. (Cplusplus 2017.)

C++-ohjelmointikielen historia ulottuu vuoteen 1979 asti. Se sai alkunsa nimellä C with Classes, eli C Luokilla, jonka oli nimensä mukaisesti tarkoitus olla olio-ohjelmointia (object-oriented) tukeva versio C-ohjelmointikielestä. Kaikkien C-ohjelmointikielen ominaisuuksien lisäksi C with Classes tuki myös luokkia, inline-funktioita, funktioiden oletusarvoja ja vahvaa muuttujien tyyppien tarkistusta. Vuonna 1983 ohjelmointi kielen nimi muutettiin C with Classesista muotoon C++ ja vuonna 1998 se ensimmäistä kertaa standardisoitiin viralliseksi ohjelmointikieleksi, jolloin sen versio kulki nimellä C++98. C++-ohjelmointikieltä on kehitetty sen alkuajoista aina tähän päivään asti ja sen viimeisin versio, C++17 julkaistiin vuoden 2017 joulukuussa. (Cplusplus 2017.)

C++-ohjelmointikieli on todella suuri ja monimutkainen oppia, joten sitä ei pidetä kovinkaan aloittelijaystävällisenä uusille ohjelmoijille ja sitä suositellaankin oppimaan hyvässä opetuksessa. Ohjelmointikielen kaksi suurinta vahvuutta ovat sen nopeus ja skaalautuvuus, joten useimmat paljon resursseja vaativat ohjelmistot, kuten esimerkiksi käyttöjärjestelmien osat tai vaativaa grafiikkaa käyttävät 3D pelit on ohjelmoitu C++-ohjelmointikielellä. (Codementor 2016.) Edellä mainittujen käyttötarkoituksien lisäksi sillä on tehty monia paljon käytettyjä graafisia sovelluksia, kuten Adoben Photoshop ja Illustrator. C++-ohjelmointikielellä on tehty myös internetselain Mozilla Firefox ja yksi eniten käytetyistä tietokantaohjelmistoista, MySQL. (Invensis 2015.)

4 UNREAL ENGINE 4

Unreal Engine on Epic Gamesin kehittämä pelimoottori, jota käytettiin ensimmäistä kertaa Unreal-tietokonepelissä, vuonna 1998 julkaistussa ensimmäisen persoonan ammuntapelissä. Sen neljäs versio, Unreal Engine 4 julkaistiin vuonna 2014.

4.1 Yleisesti Unreal Engine 4 -pelimoottorista

Unreal Engine 4 -pelimoottori on kokoelma kehitystyökaluja, joilla voi kehittää applikaatioita ja pelejä PC:lle, Macille, Linuxille, nykyaikaisille pelikonsoleille, internet selaimille, VR- ja AR-laitteille sekä Android- ja iOS-mobiililaitteille. Pelimoottorin käyttämisestä on Unreal Enginen neljännen version yhteydessä tehty täysin ilmaista, jotta kuka tahansa voi sitä käyttää ja oppia kehitystiimin koosta ja varakkuudesta riippumatta. Pelimoottorin myymisen sijaan Epic Games tienaa siten, että kun Unreal Enginellä tehty peli julkaistaan ja se tuottaa vähintään 3000 $ vuosineljänneksellä, tulee pelin tuotoista 5 % maksaa rojalteina Epic Gamesille. (Epic Games 2017a.)

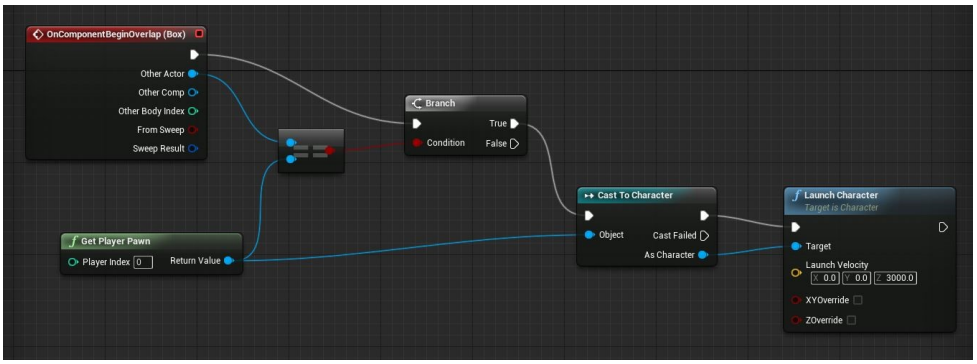
Unreal Editorin ulkoasu on pitkälti käyttäjän muokattavissa. Kaikki Unreal Editorin ominaisuudet ei kuitenkaan ole samaan aikaan näkyvissä, koska se veisi liikaa tilaa näytöllä. Unreal Editorin työkalut aukeavat sitä mukaan, kun tietyn tyyppisiä asioita aletaan muokkaamaan. Kuvassa 1 näkyy oletusnäkymä Level Editorista, eli editorista, jolla voi muokata pelin tasojen ulkoasua ja toimivuutta. Unreal Engine –pelimoottori on kokonaan avointa lähdekoodia, joten kaikki sen ominaisuudet ovat käyttäjien tarkasteltavissa ja muokattavissa. (Epic Games 2017b.)

Kuva 1. Level Editor (Epic Games 2017c).

4.2 Skriptit Unreal Engine 4 -pelimoottorissa

Skripteillä voidaan määrittää kaikki pelin toiminta, kuten peliobjektien liikkeet tai pisteiden lasku. Unreal Engine 4 -pelimoottorissa skriptit kirjoitetaan C++-ohjelmointikielellä. Ennen Unreal Enginen käytön aloittamista suositellaan, että on jonkin verran kokemusta C++-ohjelmointikielestä, mutta myös muiden ohjelmointikielien, kuten C#:n tai Javan osaamisesta on hyötyä, koska niissä on paljon samankaltaisuuksia. C++-ohjelmointikieltä pidetään yleisesti vaikeana oppia, mutta Unreal Enginen kanssa sen oppimisen pitäisi olla helpompaa. Skriptien kirjoittamiseen ja muokkaamiseen Windows tietokoneella suositellaan Visual Studio -ohjelmankehitysympäristöä, jonka kanssa Unreal Engine on tehty toimimaan hyvin yhteistyössä. (Epic Games 2017b.)

Pelielementtien ohjelmoimiseen Unreal Enginessä on kaksi eri mahdollisuutta. C++-skriptien lisäksi peliä voi kehittää visuaalisten skriptien avulla, joita kutsutaan Blueprinteiksi (Kuva 2). Niitä käyttämällä ihmiset, joilla ei ole yhtään kokemusta ohjelmoimisesta voivat luoda kokonaisia pelejä pelkällä visuaalisella Blueprint skriptauksella. Usein projekteissa kumpaakaan näistä tavoista ei käytetä yksistään, vaan niitä yhdistellään esimerkiksi siten, että peliohjelmoija kirjoittaa monimutkaisempia C++-skriptejä, joita pelisuunnittelija sitten yhdistelee Blueprinttien avulla muodostaen mielenkiintoisia pelimekaniikoita. (Epic Games 2017b.)



Kuva 2. Blueprinteillä skriptaamista (Epic Games 2017d).

4.3 Mobiilipelit

Unreal Engine on aikaisemmin ollut erityisen tunnettu sen 3D-grafiikoista PC- ja konsolipelien puolella. Viime vuosien aikana ja erityisesti Unreal Enginen neljännen version julkaisun myötä, se on alkanut tukemaan enemmän mobiilipelien kehittämistä ja siinä on 3D-grafiikoiden tuen lisäksi hyviä ominaisuuksia myös 2D-pelien kehittämiseen. (Pluralsight 2014.)

Mobiilipelien kehittäminen Unreal Engine -pelimoottorilla ei eroa toisille alustoille suunnattujen pelien kehittämisestä juuri ollenkaan. Joitain hyvin pieniä rajoitteita mobiilipelien kehityksessä on, kuten esimerkiksi tietynlai-

sia materiaaleja tai tekstuureja ei pystytä käyttämään. Suurin ero mikä mobiilipelien kehittäjän tulee ottaa huomioon, on mobiililaitteiden pienempi tehokkuus verrattuna nykyaikaisiin tietokoneisiin ja pelikonsoleihin. Lisäksi mobiilipelin suunnittelussa tulee kehittäjän ottaa huomioon esimerkiksi pelin kontrollointi mobiililaitteella ja mobiilipelien erilainen kohderyhmä.

5 MOBIILIPELIN TOTEUTUS

5.1 Tavoitteet

Tavoitteena on saada tehtyä valmis mobiilipeli Android-käyttöjärjestelmälle. Peli tulee olemaan todella yksinkertainen 2D-grafiikkaa käyttävä kasuaalipeli. Peliä kehittäessä tullaan oppimaan lisää Unreal Engine -pelimoottorin käyttämisestä. Ennen pelin kehittämisen aloittamista opinnäytetyön tekijä on käynyt läpi tutoriaaleja oppiakseen perusteita Unreal Enginen käytöstä ja skriptien kirjoittamisesta C++-ohjelmointikielellä. Opinnäytetyön tarkoituksena on kertoa lukijalle pelin kehittämisprosessista melko yksityiskohtaisesti, mutta tarkoitus ei ole kuitenkaan kirjoittaa opasta. Grafiikat peliä varten tehdään Photoshop nimisellä kuvanmuokkausohjelmalla.

Peliä kehittäessä tullaa painottamaan erityisesti pelin kontrollointia, valikoiden käyttäjäystävällisyyttä ja pelin skaalautumista erikokoisille ja kuvasuhteisille näytöille. Tarkoituksen on vastata seuraaviin tutkimuskysymyksiin:

Kuinka luoda Androidille mobiilipeli, joka käyttää kosketusnäyttöä pelin kontrollointiin Unreal Engine 4:llä?

Kuinka luoda käyttäjäystävällinen ja toimiva valikko mobiilipelille Unreal Engine 4:llä?

Kuinka luoda eri resoluutioille ja ruudun kuvasuhteille skaalautuva mobiilipeli Unreal Engine 4:llä?

5.2 Mobiilipelin suunnittelu

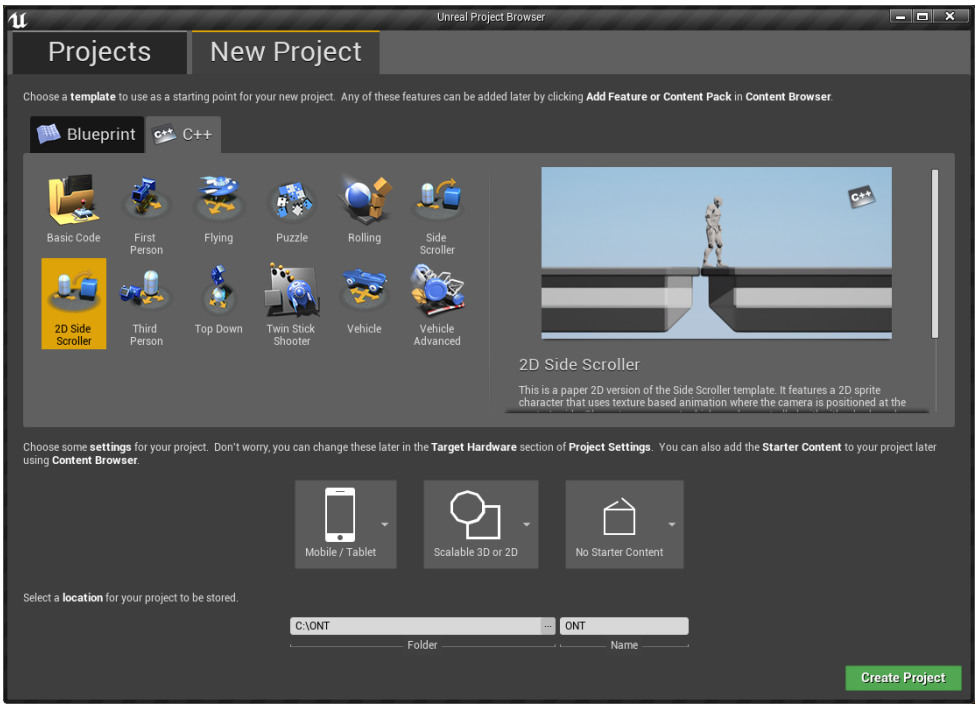
Peliä suunniteltaessa päädyttiin tekemään kasuaalipelin Android-laitteille, koska kasuaalipelit ovat useimmiten pelattavuudeltaan todella yksinkertaisia ja pelien kehittämien on hyvä aloittaa jostain yksinkertaisesta. Pelin luomisessa tullaan käyttämään Unreal Engine -pelimoottorin Paper 2D -työkalua, jolla voi luoda 2D-pelejä tai pelejä joissa käytetään 2D- ja 3D-grafiikoiden yhdistelmää.

Pelissä on tarkoitus väistellä ylhäältä putoavia ja sivuilta tulevia objekteja ja pisteitä saa sitä mukaan, mitä kauemmin selviää. Pelihahmoa voi ohjata vasemmalle koskettamalla mobiililaitteen näytön vasenta puolta ja oikealle koskettamalla näytön oikeaa puolta. Sivulta tulevia objekteja pystyy väistämään pyyhkäisemällä näyttöä ylöspäin, jolloin pelihahmo hyppää.

Grafiikat ja animaatiot peliä varten opinnäytetyön tekijä tekee itse ja ne tulevat myös olemaan todella yksinkertaisia

5.3 Projektin aloitus

Opinnäytetyötä varten luotiin uusi projekti Unreal Engineen, käyttäen kuvan 3 mukaisia asetuksia. Projektia luotaessa on mahdollista valita malli sen perusteella, minkälaisen pelin aikoo luoda tai vaihtoehtoisesti projektin voi aloittaa kokonaan tyhjästä. Täytyy myös valita, onko valittu malli toteutettu käyttäen Blueprinttejä vai C++-skriptejä. Näitä molempia voi kuitenkin myöhemmin käyttää sekaisin keskenään, valinnasta riippumatta. Opinnäytetyön projektiin valittiin pohjaksi C++-skripteillä toteutettu ”2D Side Scroller” -peli, koska siitä saa hyödyllisiä esimerkkejä erityisesti pelihahmon liikuttamista varten, vaikka mallin mukana tulevia grafiikoita ei aiotaan käyttää.



Kuva 3. Uuden projektin asetukset.

Projektin aloittamisvaiheessa pitää valita onko peli tehty ensisijaisesti mobiililaitteille vai tietokoneille/pelikonsoleille ja onko grafiikan tarkoitus olla helpommin skaalautuvaa 2D tai 3D grafiikkaa vai onko tarkoitus, että grafiikka on mahdollisimman hyvälaatuista. Asetuksia voi kuitenkin muokata myöhemmin. Lisäksi alussa voi valita lisätäänkö projektiin valmiiksi laaja valikoima resursseja, kuten esimerkiksi grafiikkaa, partikkeliefektejä, materiaaleja ja ääniä. Opinnäytetyön projektissa ei näitä tule luultavasti tarvitsemaan, joten niitä ei oteta turhaan projektin kokoa kasvattamaan.

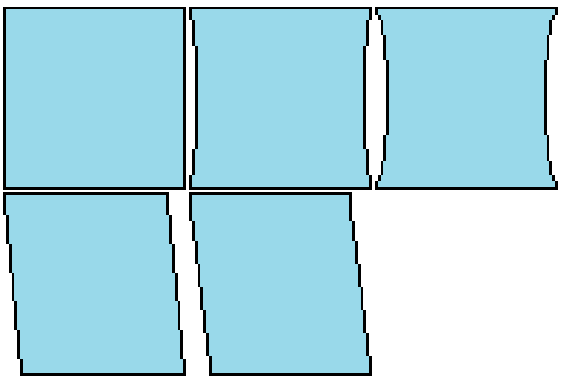
5.4 Pelihahmo

Pelaajan ohjaama hahmo on yksinkertainen vaaleansinisen värinen neliö. Pelihahmo ja kaikki muukin peliä varten tehty grafiikka on niin kutsuttua

sprite-grafiikkaa, eli kaksiulotteista pikseligrafiikkaa. Pelihahmoa pystyy liikuttamaan vasemmalle ja oikealle, minkä lisäksi se pystyy myös hyppäämään. Valitun ”2D Side Scroller” -mallin mukana tuli hahmo, jonka liikkeisiin on käytetty Unreal Engine -pelimoottorista valmiiksi löytyvää Character Movement -komponenttia. Tämä sama komponentti todettiin hyvin toimivaksi myös opinnäytetyön pelihahmon kanssa, joten pelihahmon liikkeiden ohjelmoimista uudelleen ei koettu tarpeelliseksi. Character Movement -komponentin arvoja pystyy säätämään editorin sisältä (Kuva 4) ja niitä säätämällä saadaan hahmon liikkeet halutun kaltaiseksi.

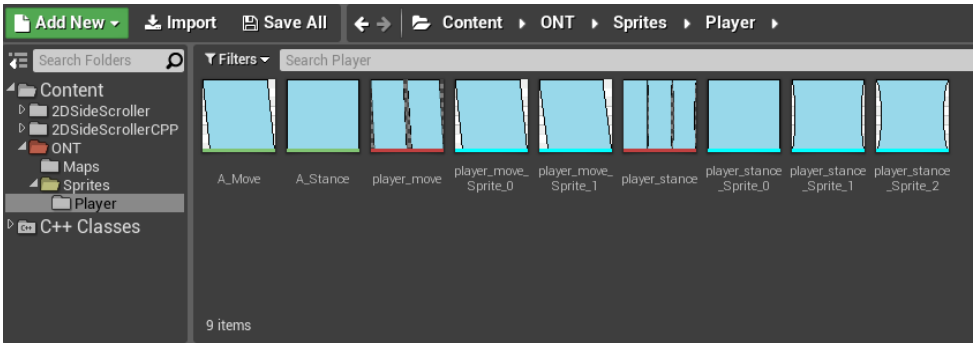
Kuva 4. Pelihahmon liikkeiden säätimet.

Pelihahmon spritet luotiin Photoshopissa ja kaikki hahmon spritet ovat samassa kuvatiedostossa (Kuva 5), josta Unreal Engine osaa automaattisesti jakaa sen yksittäisiksi spriteiksi.



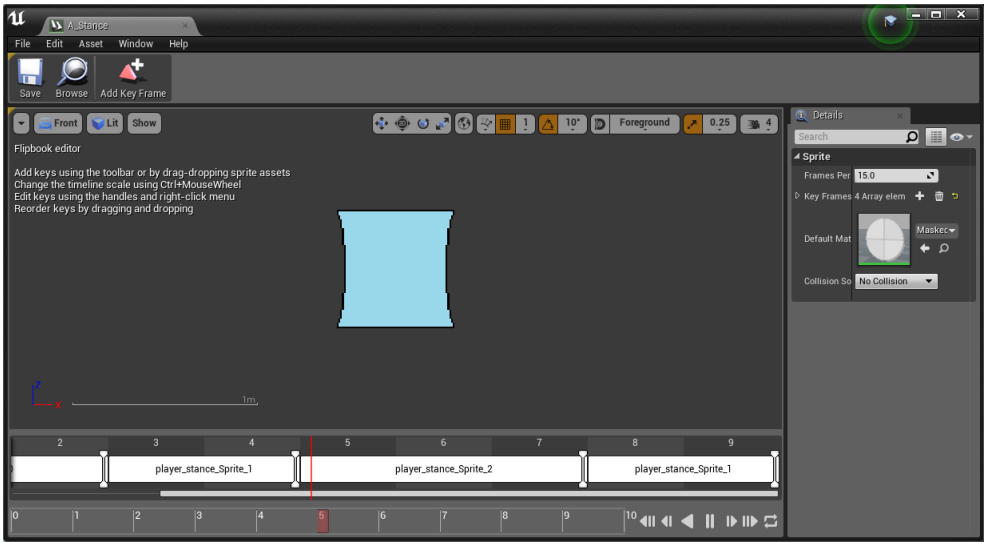
Kuva 5. Pelihahmon sprite-grafiikka.

Kuvassa 6 näkyy, kuinka player_move ja player_stance -tiedostot saatiin jaettua automaattisesti erillisiksi numeroiduiksi spriteiksi. Unreal Enginessä kaksiulotteisten pelien animaatioita kutsutaan Flipbookeiksi. Flipbookeja on todella yksinkertainen luoda valmiista spriteistä ja niitä on myös melko yksinkertainen muokata. Kuvassa näkyvät A_Move ja A_Stance ovat Flipbook-animaatioita.



Kuva 6. Pelihahmon animaatiot ja erotellut spritet.

Flipbookin saa luotua valitsemalla kaikki animaatioon halutut spritet ja hiiren oikean painikkeen takaa saa valittua ”Create Flipbook” ja näitä luotuja Flipbookeja voi muokata Flipbook-editorissa (Kuva 7). Flipbook-editorin tarkoitus on lähinnä pystyä muokkaamaan spritejen järjestystä ja animaation nopeutta. Animaation nopeutta pystyy säätämään muokkaamalla joko kuvien määrää sekunnissa (frames per second) tai säätämällä kuinka monen kuvan ajan tietty sprite näkyy.



Kuva 7. A_Stance Flipbook-editorissa.

Peli valitsee oikean animaation näytettäväksi käyttämällä seuraavaa funktiota:

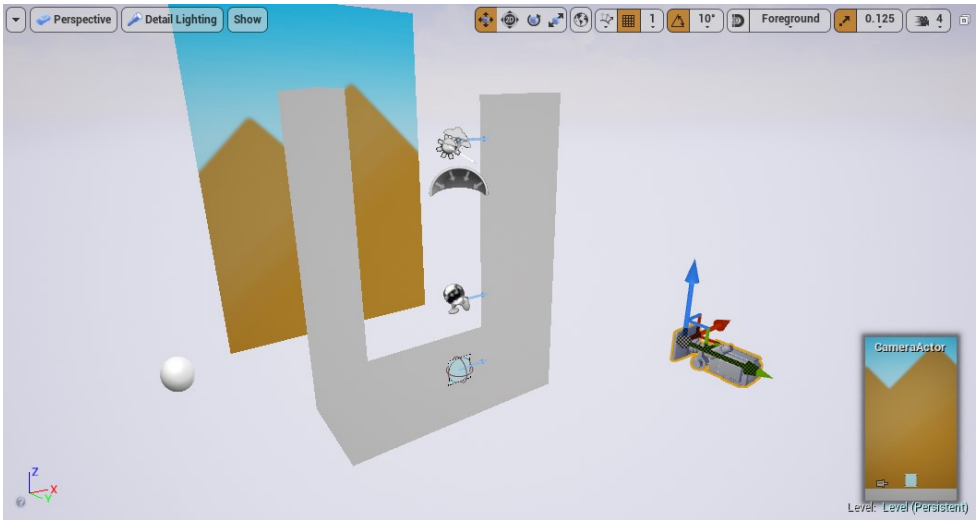
```
void AONTCharacter::UpdateAnimation()
{
    const FVector PlayerVelocity = GetVelocity();
    const float PlayerSpeedSqr=PlayerVelocity.SizeSquared();

    UPaperFlipbook* DesiredAnimation=(PlayerSpeedSqr>0.0f)?
    MovingAnimation : StandingAnimation;

    if(GetSprite()->GetFlipbook() != DesiredAnimation){
        GetSprite()->SetFlipbook(DesiredAnimation);
    }
}
```

UpdateAnimation-funktiota kutsutaan pelin jokaisella kuvalla. Funktio tutkii, onko pelihahmo paikallaan vai liikkeessä ja valitsee Flipbookin MovingAnimation tai StandingAnimation sen mukaisesti. MovingAnimation- ja StandingAnimation-olioihin pystyy valitsemaan halutut animaatiot Unreal Editorista.

5.5 Pelitason luominen

Pelin mallissa tuli mukana esimerkki taso (map), joka näyttää kuvan 3 mukaiselta esimerkiltä sisältäen esimerkiksi ihmismäisen pelihahmon ja 2D-spritejä joiden päällä pelihahmolla voi liikkua. Esimerkkitason muokkaamisen sijaan luotiin uusi tyhjä taso, jota alettiin muokkaamaan halutun kaltaiseksi.

Pelin näkymä määritellään käyttämällä kameraobjekteja kuvaamaan peliä. Valitun ”2D Side Scroller” -mallin hahmossa oli valmiina kamera, joka seuraa hahmon liikkeitä, mutta koska opinnäytetyön peliin ei haluttu liikkuvaa kameraa, se otettiin pois käytöstä ja sen tilalle peliin lisättiin uusi kamera.



Kuva 8. Pelin näkymä kolmiulotteisessa maailmassa.

Vaikka peli näyttääkin kaksiulotteiselta, on se oikeasti 3D maailmassa, mutta oikeanlaisilla 2D-spriteillä ja kameran asetuksilla se saadaan näyttämään kaksiulotteiselta. Peliä pystyy kuitenkin tarkastelemaan ja muokkaamaan sekä 3D, että 2D näkymässä. Kuvassa 8 näkyy pelialue 3D maailmassa ja kun kamera on valittuna, kuten kuvassa, näkyy oikeassa alareunassa kameran näkymä. Kuvassa näkyy kameran ja pelihahmon lisäksi myös harmaat palkit, joilla pelialue on rajattu, jotta pelihahmo ei putoa pelialueelta ulos. Lisäksi taustalla näkyy vaalea sprite, joka muodostaa peliin taustakuvan. Kameran asetuksista voi valita perspektiivisen projektion, jolloin peli näyttää kolmiulotteiselta tai ortografisen projektion, mikä saa kolmiulotteisen maailman näyttämään kaksiulotteiselta. Pelialue näkyy ka-

merassa oikein, kun kameran kuvasuhteen arvo on 0.5625, joka tulee kuvasuhteesta 9:16, mikä on perinteinen HD-kuvasuhde 16:9 käännettynä pystyasentoon.

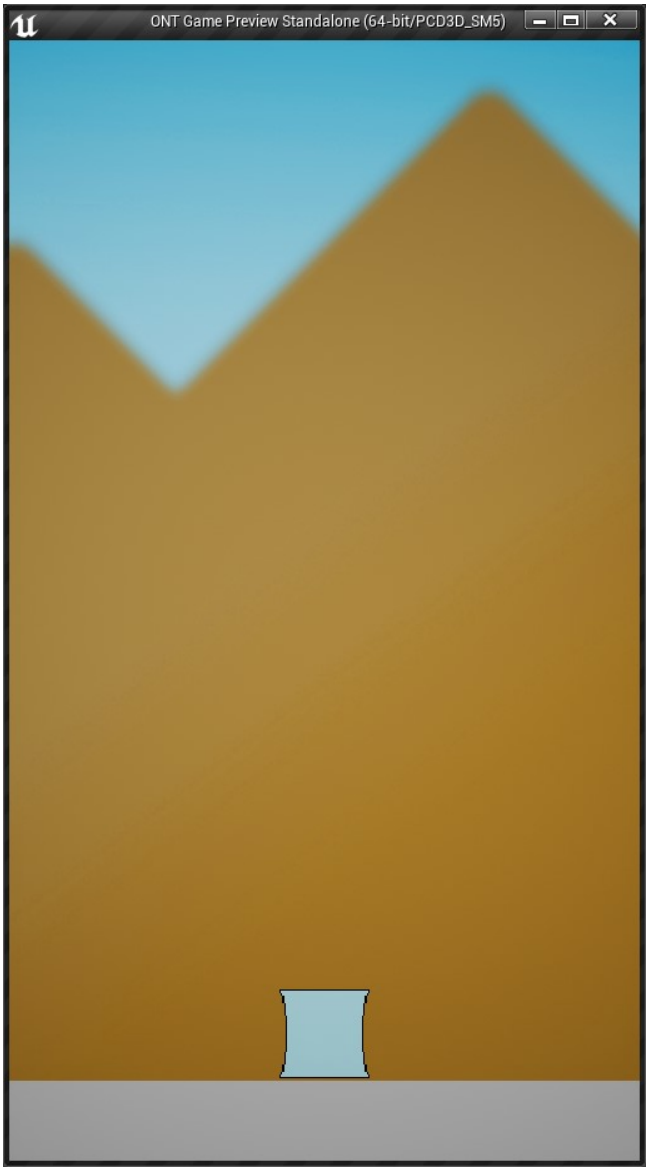
Pelin alkaessa peli ei osaa automaattisesti valita, mitä kameraa pelissä tulisi käyttää, vaikka kameroita olisikin vain yksi. Kameran kontrollointia varten tehtiin uusi Camera.cpp -skripti, jossa kamera otetaan käyttöön:

```
void ACamera::BeginPlay()
{
    Super::BeginPlay();

    APlayerController* OurPlayerController =
    UGameplayStatics::GetPlayerController(this, 0);

    OurPlayerController->SetViewTarget(Camera);
}
```

Camera-skriptin BeginPlay-funktio suoritetaan heti pelin alkaessa. Siinä käytetään APlayerController-luokan SetViewTarget-funktiota, jolla on annettu parametrina Camera. Kamera on julkinen olio ja se on asetettu muokattavaksi missä tahansa, joten käytettävä kamera voidaan valita helposti Unreal Editorin sisällä. Käytettävä kamera voidaan helposti valita Unreal Editorin sisällä, koska Camera-olio on asetettu julkiseksi ja muokattavaksi missä tahansa:

```
public:
    UPROPERTY(EditAnywhere)
    AActor* Camera;
```

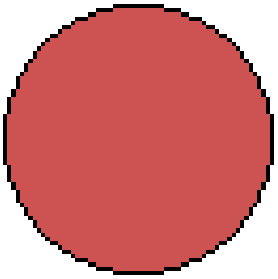
Pelin alkaessa haluttu kamera otetaan käyttöön ja peli alkaa näyttämään kaksiulotteiselta ja harmaat palkit sivuilta on rajattu pois näkökentästä (Kuva 9).



Kuva 9. Näkymä pelin käynnistyessä.

5.6 **Viholliset**

Pelin viholliset ovat yksinkertaisia punaisia ympyröitä (Kuva 10). Ne muodostuvat vain yhdestä kuvasta, eli vihollisille ei ole tehty animointia. Vihollisia varten luotiin kaksi uutta skriptiä. Ne ovat EnemySpawner.cpp -skripti, jolla toteutetaan vihollisten ilmestyminen tasoon halutulla tavalla, sekä Enemy.cpp -skripti, jota muokkaamalla voidaan säätää itse vihollisten toimintaa. Jotta vihollisia pystyy lisäämään tasoon helposti, luotiin Enemy-skriptistä kolme erillistä blueprinttiä, joille kullekin on asetettu hieman erilaiset liikkumista ohjaavat asetukset. Yksi Enemy-blueprinteistä on ylhäältä alaspäin liikkuva vihollinen ja kaksi muuta blueprinttiä ovat oikealle ja vasemmalle liikkuvat viholliset.



Kuva 10. Vihollisen sprite.

5.6.1 Ylhäältä putoavat viholliset

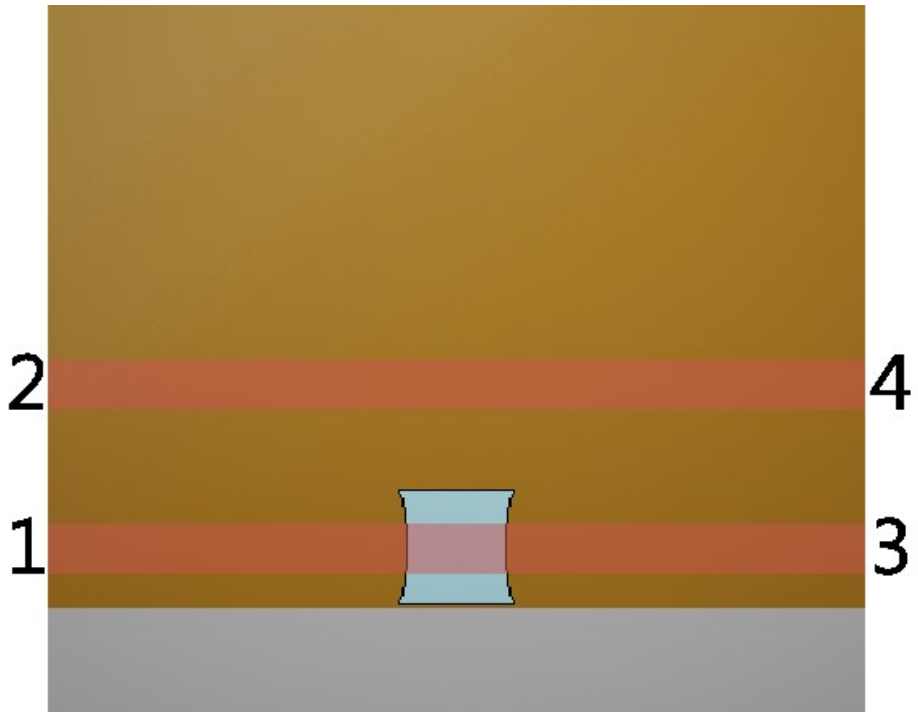
Ylhäältä putoavien vihollisten luomista ohjataan seuraavalla koodilla, joka löytyy EnemySpawner-skriptin Tick-funktiosta, eli funktiosta jota kutsutaan pelin jokaisella kuvalla.

```
if (currentTimeVert >= 2.f) {

   float xCoord= generateX(DeltaTime);

   locationVert.X = xCoord;

   AEnemy* newEnemy = GetWorld()->SpawnActor<AEnemy>
   (spawningEnemyVert, locationVert, rotationVert, spawn
   Params);

   currentTimeVert = 0.f;
}
```

Koodi tutkii muuttujaa currentTimeVert, joka on float-tyyppinen muuttuja ja se kasvaa yhdellä joka sekunti. Kun currentTimeVert on yhtä suuri tai suurempi kuin 2, skripti luo uuden ylhäältä putoavan vihollisen ja nollaa currentTimeVert-muuttujan. Tämä saa uuden ylhäältä putoavan vihollisen ilmestymään aina kahden sekunnin välein. Vihollisen luova SpawnActor-funktio tarvitsee saa parametreina olion, joka halutaan luoda, sijainnin, suunnan ja FActorSpawnParameters-luokan olion. Luotava spawningEnemyVert-olio on asetettu globaaliksi muuttujaksi, eli siihen voidaan Unreal Editorin sisältä valita haluttu Enemy-blueprint. Vihollisen sijainti ja suunta on alustettu luokan rakentajassa sijainnin X-koordinaattia lukuun ottamatta. Vihollisen X-koordinaattia varten ohjelmoitiin oma generateX-funktio, jossa X-koordinaatille arvotaan satunnainen float-arvo pelialueen rajojen sisäpuolelta. FActorSpawnParameters-luokan olio on myös alustettu luokan rakentajassa. Se pitää sisällään tiedon muun muassa luotavan objektin nimestä ja objektin omistajasta. Kaikki spawnParams-olion parametrit on asetettu oletusarvoihin.

5.6.2 Sivuilta tulevat viholliset

Sivuilta tulevien vihollisten koodi muistuttaa paljon ylhäältä tulevien vihollisten koodia. Se sijaitsee myös EnemySpawner-skriptin Tick-funktiossa ja siinä käytetään samaa tapaa luoda uusi vihollinen aina neljän sekunnin välein. Erona putoaviin vihollisiin, sivulta tuleville vihollisille ei arvota suoraan koordinaattia, vaan niille arvotaan kokonaisluku yhden ja neljän väliltä. Arvottu luku määrittää mihin kohtaan luotu vihollinen ilmestyy ja mihin suuntaan se lähtee kulkemaan. Vihollisilla on kaksi mahdollista eri korkeutta johon ne voivat ilmestyä. Puolet vihollisista ilmestyvät pelihahmon korkeudelle, jolloin pelaajan on pakko hypätä niiden yli. Vaihtelun ja lisähaasteen kannalta toinen puoli vihollisista kulkee hieman pelihahmon yli, jolloin niiden yli voi hypätä, mutta se ei ole pakollista.



Kuva 11. Sivuilta tulevat viholliset

Kuvassa 11 näkyy mihin kohtaan pelitasoa viholliset ilmestyvät minkäkin numeron saatuaan. Mikäli viholliselle arvotaan luku 1 tai 2, vihollinen ilmestyy pelihahmon vasemmalle puolelle ja vihollinen luodaan oikealle päin liikkuvasta Enemy-blueprintistä, jolloin vihollinen kulkee oikealle kuvaan 11 merkattua punaista viivaa pitkin. Mikäli viholliselle arvotaan luku 3 tai 4, luodaan vihollinen muuten samalla tavalla, kuin lukujen 1 ja 2 tapauksissa, mutta siinä käytetään vasemmalle päin liikkuvaa Enemy-blueprinttiä. Edellä kuvattu toiminto on ohjelmoitu seuraavanlaisella ohjelmakoodilla:

```
if (currentTimeHori >= 4.f) {

    int rand = FMath::RandRange(1, 4);

    if (rand == 1) {
```

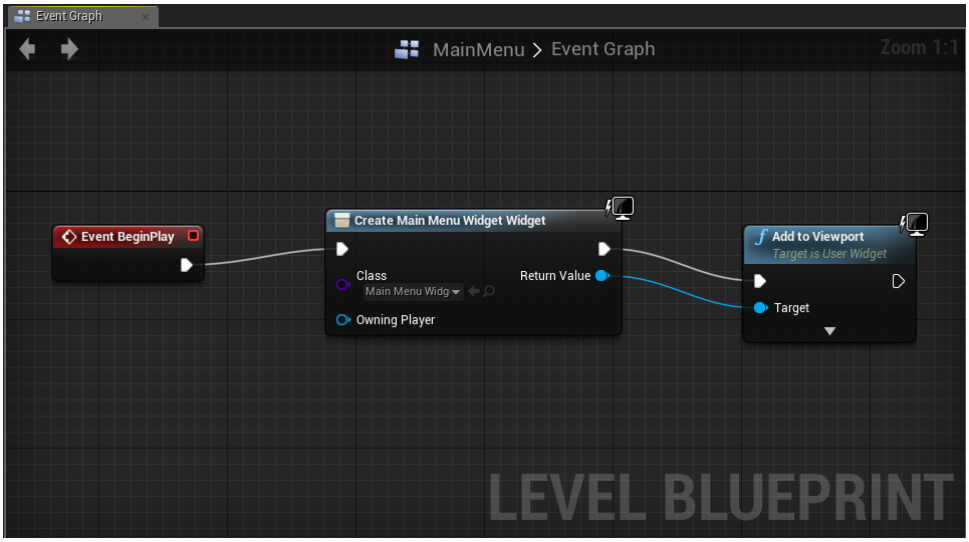
```
            locationHori.X = -330.f;
            locationHori.Z = 70.f;

            AEnemy* newEnemy = GetWorld()->SpawnActor<AEnemy>
            (spawningEnemyRight, locationHori, rotationHori,
            spawnParams);
        } else if (rand == 2) {

            locationHori.X = -330.f;
            locationHori.Z = 150.f;

            AEnemy* newEnemy = GetWorld()->SpawnActor<AEnemy>
            (spawningEnemyRight, locationHori, rotationHori,
            spawnParams);
        } else if (rand == 3) {
            locationHori.X = 330.f;
            locationHori.Z = 70.f;

            AEnemy* newEnemy = GetWorld()->SpawnActor<AEnemy>
            (spawningEnemyLeft, locationHori, rotationHori,
            spawnParams);
        } else if (rand == 4) {

            locationHori.X = 330.f;
            locationHori.Z = 150.f;

            AEnemy* newEnemy = GetWorld()->SpawnActor<AEnemy>
            (spawningEnemyLeft, locationHori, rotationHori,
            spawnParams);
        }
    currentTimeHori = 0.f;

    }
```

5.7 **Valikoiden luominen**

5.7.1 Alkuvalikko

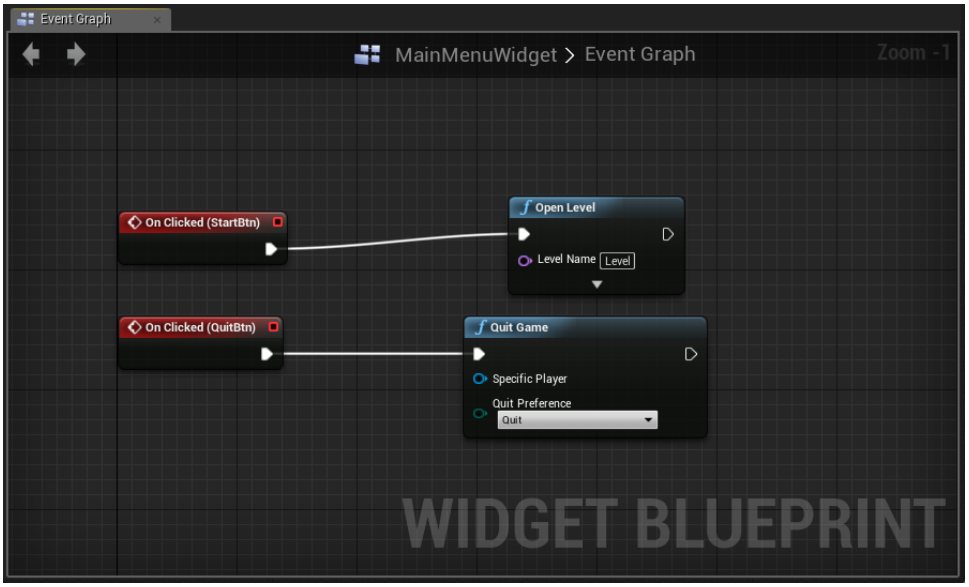
Alkuvalikko on ensimmäinen asia, jonka pelaaja näkee pelin avaamisen jälkeen, mikä tekee myös valikoiden käyttäjäystävällisyydestä tärkeää. Valikossa olevat grafiikat eivät saa ajaa pelaajia pois ja painikkeiden tulee olla selkeitä, jotta pelaaja saa selkeän käsityksen, siitä mitä mikäkin painike tekee. Valikko ei saa myöskään olla liian ahdas ja painikkeita ei saa olla liikaa.

Valikot Unreal Engine -pelimoottorilla saa luotua helposti käyttämällä siihen sisäänrakennettua Widget Bluprint-työkalua. Siinä nimensä mukaisesti käytetään C++-koodin sijaan Blueprinttejä valikoiden ja painikkeiden toiminnallisuuksien ohjelmoimiseen. Pelin alkuvalikkoa varten luotiin MainMenuWidget niminen Widget Blueprint, ja sitä varten luotiin tyhjä MainMenu niminen taso, josta Widget Blueprinttiä kutsutaan. Projektin asetuksista pystytään helposti vaihtamaan oletuksena avattava taso MainMenuksi, mikä on tärkeää, jotta peli tietää mikä taso tulee avata pelin käynnistettyä.



Kuva 12. MainMenu-tason Blueprint

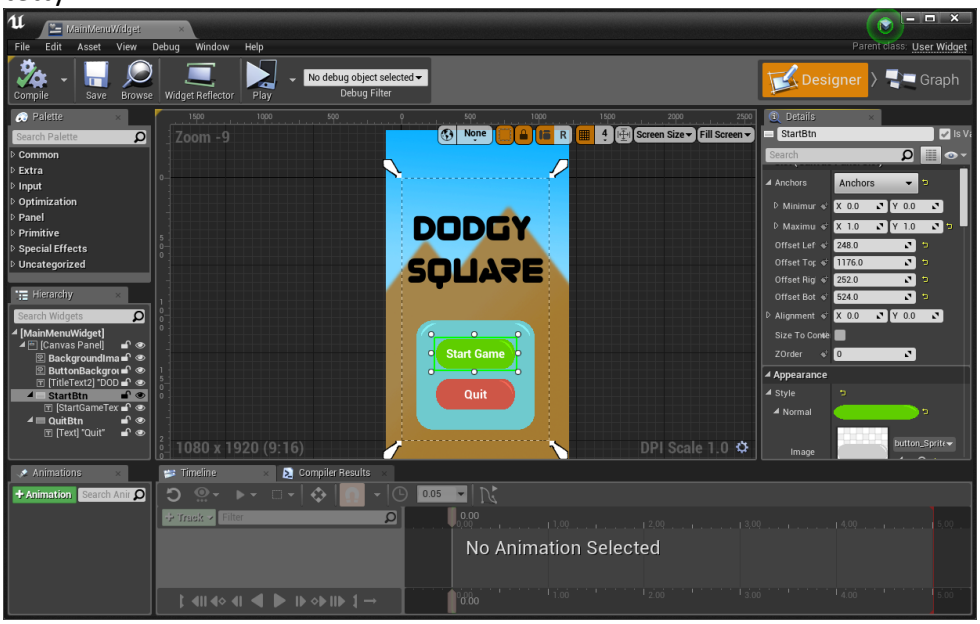
Jos tyhjä MainMenu-taso käynnistettäisiin sellaisenaan ei se tietäisi avata Widget Blueprint-valikkoa. Tämän takia tasolle kerrotaan Blueprint-skriptillä, että tason käynnistyessä sen tulee luoda MainMenuWidget ja asettaa se näkyväksi (Kuva 12).



Kuva 13. Widget Blueprintin toiminnalisuus

Myös itse Widget Blueprintin toiminnallisuudet ohjelmoidaan käyttäen visuaalista Blueprint-skriptausta. Alkuvalikossa on vain kaksi erilaista toiminnallisuutta, joista molemmat ovat painikkeiden OnClick-tapahtumia, eli ne tapahtuvat painikkeita näpäyttäessä (Kuva 13). Jotta alkuvalikko pysyisi mahdollisimman yksinkertaisena siihen on lisätty Widget Blueprint -työkalulla StartGame- ja QuitGame-painikkeiden lisäksi vain taustakuva ja pelin nimi. Widget Blueprint -editorilla on helppo lisätä uusia objekteja, kuten esimerkiksi painikkeita, tekstiä tai kuvia raahaamalla niitä editorin Palette-ikkunasta Designer-ikkunaan. Klikkaamalla lisättyjä objekteja, niitä pystyy

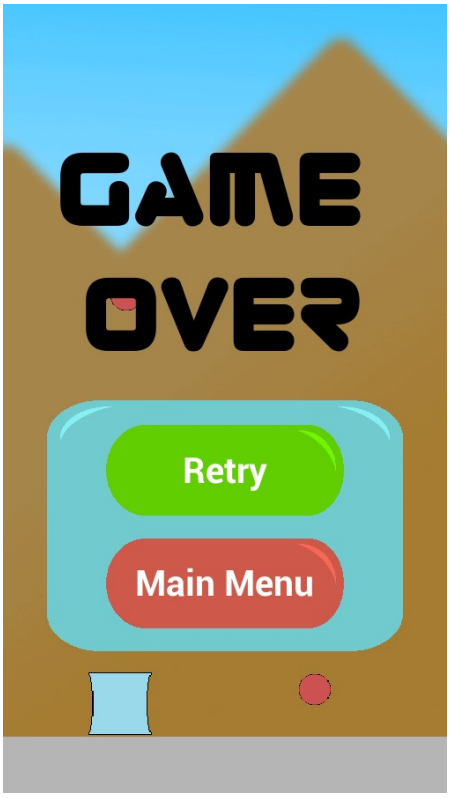
helposti muokkaamaan halutunlaiseksi säätämällä oikealla olevaa Details-ikkunaa. Kuvassa 14 näkyy edellä mainitut ikkunat, sekä keskellä olevassa Designer-ikkunassa näkyy peliä varten suunniteltu alkuvalikko, joka on rakennettu Widget Blueprint -editorin objekteista ja opinnäytetyön tekijän tekemistä grafiikoista. Widget Blueprint -työkalulla pystyy myös rakentamaan valikkoon animaatioita, mutta niitä ei opinnäytetyön pelissä ole käytetty.



Kuva 14. Widget Blueprint -työkalu

5.7.2 Game Over -näkymä

Alkuvalikon lisäksi peliin on luotu toinen Widget Blueprint, joka avautuu pelaajan osuttaessa viholliseen, eli kun pelaaja häviää pelin. Game Over -näkymälle ei ole luotu omaa tasoa, vaan näkymä avautuu pelikuvan päälle, siten että peli näkyy vielä liikkuvan valikon takana (Kuva 15).



Kuva 15. Game Over -näkymä

Game Over -näkymä muistuttaa todella paljon pelin alkuvalikkoa, sillä se on rakennettu saman näköisistä osita, jotta peli näyttää yhdenmukaiselta kokonaisuudelta. Pelin nimen sijasta näkymässä lukee isolla tekstillä Game Over (peli ohi). Painamalla Retry-painiketta peli alkaa alusta ja Game Over -näkymä poistuu ruudulta. Main Menu -painikkeella pääsee takaisin pelin alkuvalikkoon, josta peli voidaan aloittaa tai sammuttaa. Näkymän painikkeiden toiminnallisuudet on ohjelmoitu Blueprinteillä samalla tavalla kuin alkuvalikossa, mutta näkymän avautuminen on ohjelmoitu pelihahmoa kontrolloivassa C++-skriptissä.

```
FStringClassReference Widget
(TEXT("/Game/ONT/Menu/GameOverWidget.GameOverWidget_C"));

if (UClass* GameOverWidget = Widget.TryLoadClass<UUser
Widget>())
{
    if (gameOver == false) {
        UUserWidget* GameOverScreen = CreateWidget<UUser
        Widget>(GetWorld(), GameOverWidget);
        GameOverScreen->AddToViewport();

        gameOver = true;
    }
}
```

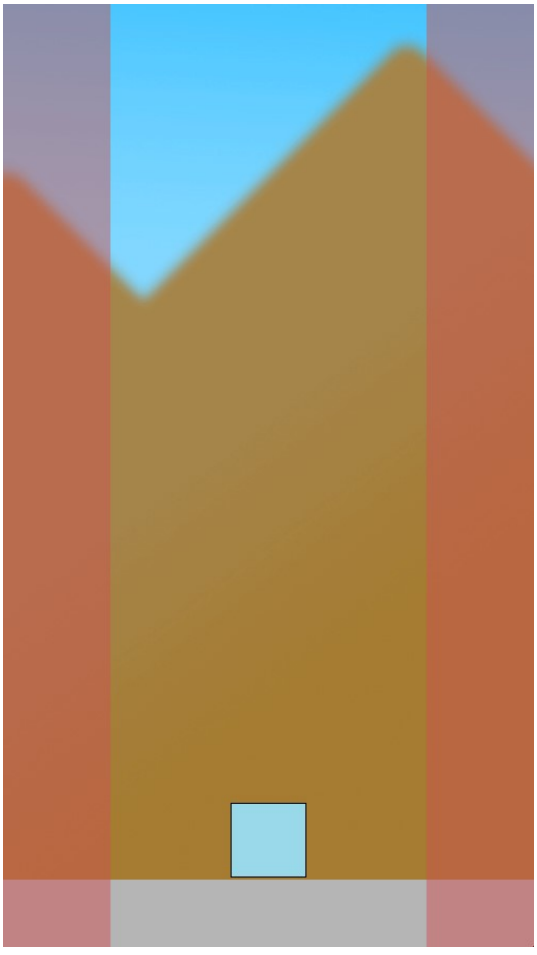
Skripti tapahtuu pelihahmon vihollisiin osumista tarkkailevan funktion sisällä. Kun pelihahmo osuu viholliseen, pelin tiedostoista haetaan luotu GameOverWidget-tiedosto ja ensimmäinen if-ehto tarkastaa löytyikö haluttu

tiedosto. Koska peli jatkaa liikkumista valikon takana ja Game Over -näkymän ei haluta aukeavan uudestaan aina pelihahmon osuttaessa viholliseen, skriptissä käytetään ehtona boolean-tyyppistä gameOver-muuttujaa. Muuttuja asetetaan tason alkaessa epätodeksi ja kun Game Over -näkymä ensimmäisen kerran lisätään ruutuun, boolean-muuttuja asetetaan todeksi. Mikäli molemmat ehdot menevät läpi, peli luo uuden Widgetin haetusta tiedostosta ja asettaa sen näkyväksi.

5.8 **Pelin kontrollointi mobiililaitteilla**

Pelien pelattavuuden ja käyttäjäystävällisyyden kannalta on erittäin tärkeää, että pelin kontrollointi on sulavaa, selkeää ja loogista. Jos pelin ohjaaminen ei miellytä pelaajaa, se tuskin tulee pelin pariin enää palaamaan.

Opinnäytetyön peliä kontrolloidaan mobiililaitteella käyttäen kosketusnäyttöä. Kuvassa 16 näkyviä punaisella merkittyjä alueita koskettamalla pelihahmoa voidaan liikuttaa sivuttaissuunnassa. Koskettamalla pelialueen vasenta reunaa pelihahmo liikkuu vasemmalle ja pelialueen oikeaa reunaa koskettamalla pelihahmo liikkuu oikealle.



Kuva 16. Alueet sivuttaiselle liikkumiselle

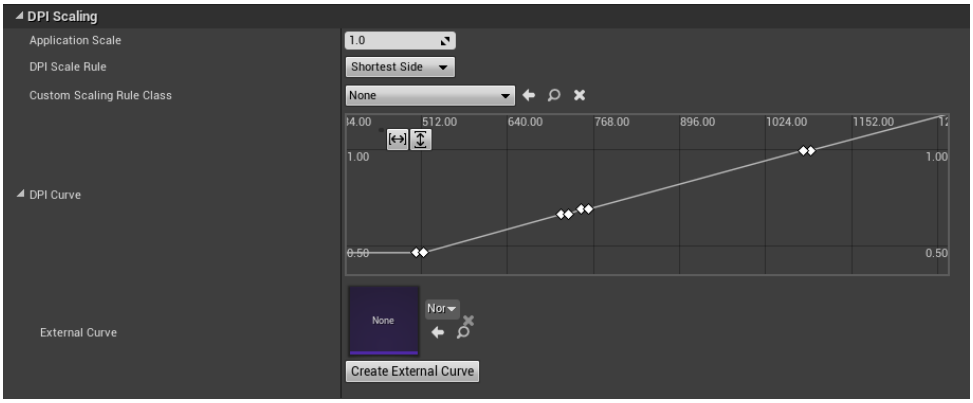
Sivuttaissuuntaisen liikkumisen lisäksi pelihahmo pystyy hyppäämään. Hyppääminen tapahtuu pyyhkäisemällä näyttöä ylöspäin, siten että pyyhkäisy alkaa pelialueen keskeltä, kuvassa 16 näkyvien punaisten alueiden välistä. Seuraavassa koodinpätkässä on tärkeimmät osat hypyn aikaansaavan pyyhkäisyn koodista:

```
void AONTCharacter::TouchStarted(const ETouchIndex::Type
FingerIndex, const FVector Location)
{
   if (Location.X > 200.0f && Location.X < 880.0f) {
      swipeStart = Location.Y;
   }
}

void AONTCharacter::TouchStopped(const ETouchIndex::Type
FingerIndex, const FVector Location)
{
   swipeEnd = Location.Y;
   swipeDist = swipeStart - swipeEnd;

   if (swipeStart != 0.0f && swipeDist >= 100.0f) {
      Jump();
   }
}
```

Hypyn koodissa tarkastetaan alkaako näytön kosketus pelialueen keskiosasta ja mikäli alkaa niin kosketuksen alun Y-koordinaatti asetetaan float-tyyppiseen swipeStart-muuttujaan. Kosketuksen päätyttyä kosketuksen lopun Y-koordinaatti asetetaan swipeEnd nimiseen muuttujaan ja näiden kahden muuttujan välinen etäisyys lasketaan ja asetetaan omaan swipeDist-muuttujaan. Mikäli muuttujien välinen etäisyys on yli 100, pelihahmo hyppää.

5.9 **Pelin skaalautuvuus**

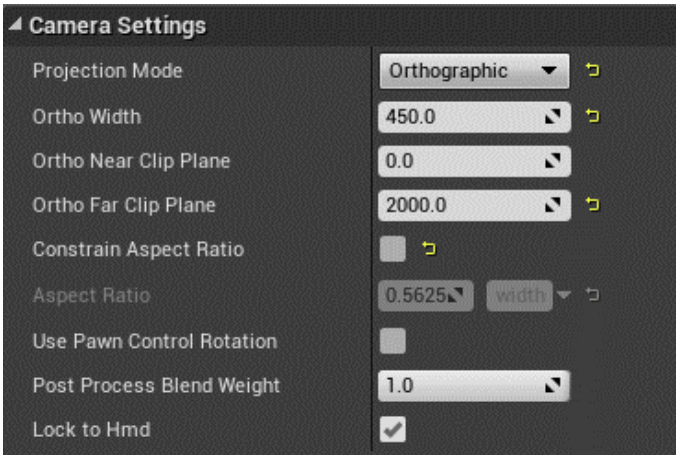
Opinnäytetyön peli on kehitetty Android-laitteille, joten on tärkeää, että peliä pystyy pelaamaan monen kokoisilla ja kuvasuhteisilla näytöillä ilman, että pelikokemus tai pelin graafinen ulkoasu kärsii. Useimmiten Unreal Engine -pelimoottorilla tehtyjen pelien valikot ovat luotu käyttäen Widget Blueprint -työkalua, kuten myös tässä projektissa. Tällä tavalla luotujen valikoiden skaalautuvuutta pystyy säätämään helposti suoraan Widget Blueprint -työkalun asetuksista (Kuva 17).



Kuva 17. Alkuvalikon skaalautuvuuden asetukset

Muokkaamalla DPI Scaling -asetusta valikko saadaan skaalautumaan oikein vain tietyllä kuvasuhteella. Kuvan tapauksessa on käytetty kuvasuhdetta 9:16. Valikko näyttää todennäköisesti hyvältä, vaikka kuvasuhde hieman eroaisikin säädetystä kuvasuhteesta. Mikäli esimerkiksi 3:4-kuvasuhteinen valikko ei näyttäisi hyvältä, asian voisi korjata luomalla uuden Widget Blueprintin samoilla komponenteilla kuin aikaisemman, mutta käyttäen DPI scaling -asetuksessa kuvasuhdetta 3:4. Oikean Widget Blueprintin pystyisi valitsemaan esimerkiksi antamalla pelaajan valita oikea kuvasuhde pelin valikosta tai usein käyttäjäystävällisempi tapa olisi luoda C++- tai Blueprint-skripti, jolla selvitetään laitteen kuvasuhde ja valitaan sopiva Widget Blueprint sen mukaan. Projektin peliä varten luotu Widget Blueprint skaalautui testien perusteella hyvin kuvasuhteille 3:4, 9:18 ja usealle kuvasuhteelle näiden välistä, joten tarvetta useamman Widget Blueprintin luomiseen ei ollut.

Itse pelin skaalautuvuus on riippuvainen peliin lisättyjen kameroiden asetuksista. Ortografiselle, eli kaksiulotteiselle kameralle täytyy asettaa kuvan leveys nähtäväksi halutun pelialueen mukaan. Kuvan korkeus skaalautuu oletuksena automaattisesti, siten että pelikuva peittää laitteen ruudun kokonaan. Tämä aiheuttaa sen, että laitteen kuvasuhteesta riippuen pystysuunnassa pelialuetta näkyy toisilla laitteilla enemmän kuin toisilla. Kameran käyttämän kuvasuhteen pystyy lukitsemaan tiettyyn arvoon. Tällöin kameran pystysuunta ei skaalaudu näytölle automaattisesti ja mikäli asetettu kuvasuhde eroaa laitteen kuvasuhteesta, jää näytön reunoille mustat palkit. Perspektiivisellä, kolmiulotteisella kameralla on vastaavanlaiset asetukset. Näytettävän pelialueen leveyden sijaan siinä säädetään asetusta Field Of View, jolla säädetään, kuinka monta astetta kameran näkökenttä on leveyssuunnassa. Jos automaattisesti skaalautuva kamera ei toimi pelin kanssa hyvin, eikä peliä ole tarkoitus pelata vain yhdellä kuvasuhteella, voi ongelman ratkaista lisäämällä tasoon useamman kameran erilaisia kuvasuhteita varten. Oikean kameran käyttöönoton pystyy toteuttamaan selvittämällä käytettävän laitteen kuvasuhteen C++- tai Blueprint-skriptissä tai antamalla käyttäjän valita pelin asetuksista haluttu resoluutio tai kuvasuhde.



Kuva 18. Pelissä käytetyn kameran asetukset

Kuvassa 18 näkyy opinnäytetyön pelissä käytetyn ortografisen kameran asetukset. Kuvan leveydeksi asetettiin 450, mikä on pelialueen rajojen välinen alue. Kameran kuvasuhdetta ei ole lukittu, vaan se skaalautuu automaattisesti laitteen näytölle pystysuunnassa. Käytettävän laitteen kuvasuhde vaikutti hieman pelissä näkyvän maan ylhäällä näkyvän alueen määrään. Kameraa testattiin usealla mahdollisella mobiililaitteen kuvasuhteella 3:4 ja 9:18 välillä ja pelialuetta näkyi jokaisella riittävästi, joten tarvetta useamman kameran luomiseen eri kuvasuhteita varten ei ollut.

6 YHTEENVETO

Ensimmäinen opinnäytetyön tutkimuskysymyksistä oli, kuinka luoda Androidille mobiilipeli, joka käyttää kosketusnäyttöä pelin kontrollointiin Unreal Engine 4:lla. Tutkimuskysymykseen saatiin vastattua hyvin. Peliä varten luodut kontrollit ovat pelaajalle intuitiiviset, loogiset ja ne on helppo oppia, mikä on erityisesti kasuaalipeleille todella tärkeää. Toinen opinnäytetyön tutkimuskysymyksistä oli, kuinka luoda käyttäjäystävällinen ja toimiva valikko mobiilipelille. Opinnäytetyössä saatiin luotua pelille yksinkertaiset ja yhdenmukaiset alku- ja loppuvalikot, mikä osaltaan vastaa tutkimuskysymykseen. Pelin toteutuksessa tutustuttiin Unreal Enginen valikoiden luomista varten tehtyyn Widget Blueprint -työkaluun. Kolmanteen tutkimuskysymykseen, eli pelin skaalautuvuuteen eri kuvasuhteisille ja resoluutioisille näytöille saatiin vastattua tutustumalla Widget Blueprin -valikoiden skaalautumiseen sekä pelin kameroiden asetuksien vaikutuksesta pelin skaalautuvuuteen. Pelin valikko ja itse pelikuva saatiin opinnäytetyössä skaalautumaan halutulla tavalla. Toiminnallisen osuuden lisäksi opinnäytetyön teoriaosuuden aiheissa painotetaan tutkimusongelmien osa-alueita.

Opinnäytetyötä tehdessä työn tekijä oppi perusteita Unreal Engine -pelimoottorin käytöstä ja C++ -ohjelmointikielellä ohjelmoimisesta, mutta molemmista on vielä paljon opittavaa jäljellä. Unreal Engine -pelimoottori todettiin opinnäytetyön aikana hieman hitaaksi oppia ja sisäistää, mikä hidasti pelin ominaisuuksien valmistumista ja muutama pelin suunnitelluista ominaisuuksista jäi pelistä puuttumaan. Tutkimuskysymyksiä koskevat ominaisuudet saatiin kuitenkin valmiiksi. Pieni kokemus Unity-pelimoottorista oli opinnäytetyössä hyödyksi, sillä Unreal Enginessä ja Unityssä on jonkin verran samankaltaisuuksia. Pelissä on paljon varaa jatkokehittämiselle. Työn tekijän on tarkoitus saada suunnitellut ominaisuudet valmiiksi harrastemielessä ja mahdollisesti suunnitella peliin kokonaan uusia ominaisuuksia.

LÄHTEET

Bevans, A. (2017). Who plays mobile games. GamesIndustry. Haettu 25.1.2018 osoitteesta http://www.gamesindustry.biz/articles/2017-06-14-who-plays-mobile-games

Chan, S. (2017). Mobile game revenue finally surpasses PC and consoles. VentureBeat. Haettu 18.1.2018 osoitteesta https://venturebeat.com/2017/07/13/mobile-game-revenue-finally-surpasses-pc-and-consoles/

Codementor (2016). Why Learn C++?. Haettu 24.1.2018 osoitteesta http://www.bestprogramminglanguagefor.me/why-learn-c-plus-plus

Cplusplus (2017). Information. Haettu 24.1.2018 osoitteesta http://www.cplusplus.com/info/

Epic Games (2017a). Unreal Engine Features. Haettu 17.1.2018 osoitteesta https://www.unrealengine.com/en-US/features

Epic Games (2017b). Unreal Engine 4 Documentation. Haettu 17.1.2018 osoitteesta https://docs.unrealengine.com/latest/INT/

Epic Games (2017c). Level Editor. Haettu 27.1.2018 osoitteesta https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/index.html

Epic Games (2017d). Blueprints. Haettu 29.1.2018 osoitteesta https://docs.unrealengine.com/latest/INT/Engine/Blueprints/QuickStart/6/index.html

HDDmag (2017). Puzzle, Casual, Action: The Top 3 Mobile Game Genres. Haettu 25.1.2018 osoitteesta https://hddmag.com/puzzle-casual-action-the-top-3-mobile-game-genres/

Invensis (2015). Applications of C / C++ in the Real World. Haettu 29.1.2018 osoitteesta https://www.invensis.net/blog/it/applications-of-c-c-plus-plus-in-the-real-world/

Malhotra, M. (2016). Types of Business Models For Mobile Application Development. Blogijulkaisu 9.6.2016. Haettu 18.1.2018 osoitteesta https://www.valuecoders.com/blog/outsourcing-and-off-shoring/types-of-business-models-for-mobile-application-development/

Material Design (n.d.). Device Metrics. Haettu 22.1.2018 osoitteesta https://material.io/devices/

Munir, A. (2014). App Monetization: 6 Bankable Business Models That Help Mobile Apps Make Money. Blogijulkaisu 10.9.2014. Haettu 18.1.2018 osoitteesta http://info.localytics.com/blog/app-monetization-6-bankable-business-models-that-help-mobile-apps-make-money

Pluralsight (2014). Unreal Engine 4 vs. Unity: Which Game Engine Is Best for You?. Haettu 25.1.2018 osoitteesta https://www.pluralsight.com/blog/film-games/unreal-engine-4-vs-unity-game-engine-best

Scolastici, C. & Nolte, D. (2013). *Mobile Game Design Essentials*. Birmingham: Packt Publishing.

Think Gaming (2018). Top Grossing Games. Haettu 18.1.2018 osoitteesta https://thinkgaming.com/app-sales-data/

ThinkMobiles (2017). Mobile game development in 2017: best tools and advice. Haettu 16.1.2018 osoitteesta https://thinkmobiles.com/blog/mobile-game-development-tools/