Pelikehitys Unreal Engine 4 -pelimoottorilla

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeen ammattikorkeakoulu, Tietojenkäsittelyn koulutusohjelma

Hämeenlinna, syksy 2018

Miikka Koskinen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Visamäki, Hämeenlinna

Tekijä Miikka Koskinen **Vuosi** 2018

Työn nimi Pelikehitys Unreal Engine 4 -pelimoottorilla

Työn ohjaaja/t Tommi Lahti, Lauri Salminen

TIIVISTELMÄ

Tämän opinnäytetyön tavoitteena oli kehittää yksinkertainen, mutta toimiva peli PC-alustalle käyttäen Unreal Engine 4-pelimoottoria ja sen työkaluja, mutta työn tarkoituksena oli myös selvittää, kuinka helppoa aloittelijan on käyttää Unreal Engine 4 -pelimoottoria pelikehitystyökaluna ja millaisia tuloksia tällainen henkilö voi saada aikaan. Tätä työtä aloittaessa opinnäytetyön tekijä ei ollut käyttänyt Unreal Engine 4 -pelimoottoria ollenkaan, mutta hänellä oli paljon aikaisempaa perusosaamista eri ohjelmointikielistä.

Tässä opinnäytetyössä on teoria että käytännöllinen osuus, mutta työ painottuu selvästi käytännölliselle osuudelle enemmän. Teoriaosuudessa käydään läpi Unreal Engine 4:än perus asioita ja muutama sen perustyökaluista, kun taas käytännöllisessä osuudessa keskitytään valitsemaan ja tekemään oikeita ja toimivia ratkaisuja tämän opinnäytetyön aikana tehtävään peliin.

Opinnäytetyötä tehdessä suurena apuna toimivat lukuisat eri Youtubesta löytyvät tutoriaalit, sekä Epic Gamesin tarjoamat tutoriaalit ja laaja dokumentointi Unreal Engine 4:än käytöstä ja sen työkaluista.

Opinnäytetyön tekijä sai työtä varten tehdyn pelin valmiiksi ja oppi paljon uutta Unreal Engine 4 -pelimoottorin käytöstä tätä opinnäytetyötä tehdessä. Kaikkiin tutkimuskysymyksiin saatiin myös vastaukset tämän opinnäytetyön aikana.

Avainsanat pelikehitys, Unreal Engine, PC

Sivut 21 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Visamäki, Hämeenlinna

Author Miikka Koskinen **Year** 2018

Subject Game Development using Unreal Engine 4 -game engine

Supervisors Tommi Lahti, Lauri Salminen

ABSTRACT

The goal of this thesis was to develop and create a simple, yet complete and functioning game to PC-platform using Unreal Engine 4 -game engine, but also to find out how easy or difficult it is for a completely new user to learn and use Unreal Engine 4 -game engine as a game development tool and what are the possibilities for the user with this software. At the start of this thesis the author had never used Unreal Engine 4 -game engine for any purpose, but he had a lot of basic understanding and experience of using different programming languages.

This thesis includes a theoretical as well as a practical part. This thesis is a lot more focused on the practical part than on the theoretical part. In the theoretical part this thesis is focused on the general and basic information about Unreal Engine 4 as well as few basic tools of Unreal Engine 4 and the practical part in the practical part the thesis is focused on choosing and implementing correct and functional solutions to the game that was developed during this thesis.

Numerous different Youtube tutorials and numerous different tutorials made by Epic Games and their comprehensive documentations, about using Unreal Engine 4 -game engine as a game development tool were used as aid in the making of this thesis.

The author managed to complete the game that was developed for and during this thesis and author learned a great deal about Unreal Engine 4 and the usage of Unreal Engine 4 -game engine as a game development tool. Every research question also was answered during this thesis.

Keywords game development, Unreal Engine, PC

Pages 21 pages

SISÄLLYS

SANASTO 4

1 JOHDANTO 1

2 UNREAL ENGINE 2

2.1 Mikä Unreal Engine 4 on? 2
2.1.1 Kehittäjäpalkkiot 2
2.1.2 Alustat 2
2.1.3 Kehitystyö 2

3 UNREAL EDITOR 3

3.1 Blueprint Visual Scripting-järjestelmä 3
3.2 Unreal Enginen käytön esivaatimukset 4
3.3 Actorit 5
3.3.1 Actorien luonti 6
3.3.2 Komponentit 6
3.4 Blueprintit 7
3.4.1 Blueprint-luokat 7

4 PC-PELIN KEHITYS UNREAL ENGINE 4 -PELIMOOTTORILLA 8

4.1 Projektin tavoitteet 8
4.2 Suunnittelu 8
4.3 Projektin aloitus 8
4.4 Valmiiksi määritellyt pohjat 9
4.4.1 2D Side Scroller 9
4.4.2 Side Scroller 10
4.5 Hahmon luominen ja muokkaaminen 10
4.5.1 Animaatioiden luominen 11
4.5.2 Hyppyanimaation käyttäminen 11
4.6 Kentän luominen ja muokkaaminen 13
4.7 Valikkojen luominen 15
4.7.1 Päävalikko 15
4.7.2 Kenttävalikko 17
4.7.3 Voittoruutu 18
4.7.4 Häviämisruutu 19
4.8 Pelin päättyminen 19

5 YHTEENVETO 21

6 LÄHTEET 22

SANASTO

C++	Ohjelmointikieli, jolla Unreal Engine 4 on tehty.
Collision	Videopeleissä tällä viitataan siihen, onko jokin pelin osa kiinteä eli voiko esimerkiksi pelaajan hahmo kävellä jonkun objektin läpi.
Epic Games	Amerikkalainen videopelinkehittäjä, sekä Unreal Engineä kehittävä yritys.
Flipbook	Unreal Enginessä oleva animaatioihin tarkoitettu leikekirja, johon voi laittaa useamman Spriten animaation luomista varten.
HUD	Head-up Display, eli käyttöliittymän osa, mikä antaa visuaalista tietoa pelin ja tai pelaajan tilasta.
Launcher	Erillinen sovellus, jota käytetään ohjelman tai tietyn valmistajan ohjelmien asennukseen ja aukaisemiseen.
OS	Operating System, Käyttöjärjestelmä.
Renderöinti	On sitä, kun laitteen grafiikkaprosessori luo tiedostosta visuaalisen näkymän.
Sprite	Kaksiulotteinen bitmap-grafiikka, joka on suunniteltu olemaan osana isompaa kokonaisuutta, joko stattisena tai animoituna objektina.
Tile	Niin sanottu ”laatta” grafiikka, joka käytetään joissakin peleissä kenttien rakennukseen, sekä yleensä suhteellisena pituus mittana muihin grafiikoihin.
UDK	Unreal Development Kit, Unrealin Enginen kehitystyökalu.
Unreal Engine	Epic Gamesin kehittämä pelimoottori.
VR	Virtual Reality, Virtuaali todellisuus.

1 Johdanto

Tässä opinnäytetyössä keskitytään ja tutustutaan Unreal Enginen kehitystyökaluun ja pelikehitykseen Unreal Engine 4 -pelimoottorilla. Ensimmäisenä käydään läpi mikä Unreal Engine 4 oikein on ja mitä sillä tehdään. Unreal Enginen kehitystyökalua kutsutaan myös UDK:si (Unreal Development Kit).

Käyttämällä valmista pelimoottoria säästää paljon aikaa pelinkehityksessä, koska kaikkia pelin osia ei tarvitse tehdä alusta asti. Pelimoottorit yleisesti tarjoavat valmiiksi asetettuja moduuleja, kirjastoja, effektejä ja työkaluja. Yleisesti ottaen pelimoottorien kehitystyökalujen käyttö on verrattain helppoa opetella ja niistä on yleisesti ottaen saatavilla laajat dokumentaatiot, tutoriaalit ja ohjeistukset myös vanhemmille/edistyneemmille pelinkehittäjille. (InstabugBlog 2018)

Opinnäytetyön tekijällä on aikaisempaa kokemusta pelikehityksestä, mutta ei opinnäytetyössä käytettävällä pelimoottorilla/ohjelmalla. Opinnäytetyön tekijän motivaationa toimii suuri oma kiinnostus pelinkehitykseen, sekä halu opetella käyttämään Unreal Engine 4 -pelimoottoria kehitystyökaluna tehokkaasti.

Työn lähtökohtana on muiden ohjelmointikielien ja ohjelmistokehitystyökalujen käytön perustaidot.

Tämän opinnäytetyön tutkimuskysymyksiä ovat seuravat: Miten luodaan PC-peli Unreal Engine 4-pelimoottoria käyttäen? Miten luodaan hyvä ja käytännöllinen valikko Unreal Engine 4:ssä? Miten Unreal Engine 4:ssä valikoissa siirtyminen toteutetaan? Miten Unreal Engine 4:ssä hahmon liikkuminen toteutetaan? Miten Unreal Engine 4:ssä animaatioiden käsittely toteutetaan?

2 Unreal Engine

2.1 Mikä Unreal Engine 4 on?

Unreal Engine 4 on pelimoottori, jolle on tehty kokonainen pelikehitystyökalu, jonka pelinkehittäjät ovat kehittäneet pelinkehittäjille. Unreal Engine 4 -pelimoottori on itsessään ohjelmoitu C++-ohjelmointikielellä ja sen lähdekoodi on täysin vapaa kaikille käyttäjille. Unreal Enginen kehittäjä ja omistaja on Epic Games. Ensimmäinen valmis versio Unreal Enginestä julkaistiin toukokuussa vuonna 1998, mutta Unreal Enginen pelikehitystyökalusta tuli ilmainen vasta Unreal Engine 3: sen aikana. Unreal Engineä alun perin käytettiin ensimmäisen persoonan ammunta pelissä nimeltä Unreal, jonka yhtenä kehittäjänä toimi sen aikainen Epic MegaGames. (Epic Games 2018h; Gamasutra 2008)

2.1.1 Kehittäjäpalkkiot

Kuten kaikissa muissakin valmiissa pelimoottoreissa ja myös Unreal Enginessä kehittäjä kerää kehittäjäpalkkioita onnistuneista projekteista. Unreal Enginessä kehittäjä palkkio on 5% ohjelman bruttotuloista $3,000 jälkeen per vuosineljännes. (Epic Games 2018g)

2.1.2 Alustat

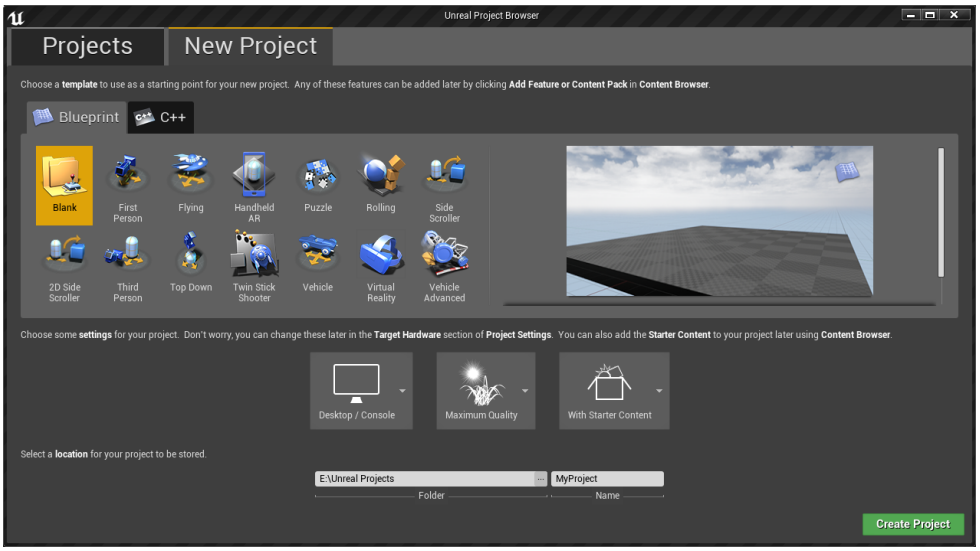
Unreal Engine 4 -pelimoottori tukee tällä hetkellä ohjelmistokehitystä seuraaville alustoilla: Windows PC, Playstation 4, Xbox One, Mac OS X, iOS, Android,VR(mukaan lukien, mutta rajoittumatta: SteamVR/HTC Vive, Oculus Rift, Playstation VR, Google VR/Daydream, OSVR ja Samsung Gear VR), Linux, SteamOS ja HTML5. Kehitystyökalu itsessään tukee vain Windowsia, OS X:ää ja Linuxia. (Epic Games 2018g)

2.1.3 Kehitystyö

Epic Games teki Unreal Engine 4 -pelimoottorin lähdekoodista täysin vapaan, koska Epic Games haluaa, että kaikki voivat osallistua pelimoottorin kehitystyöhön. Pelimoottorin lähdekoodin saa ladattua Epic Gamesin GitHubista. Kuka tahansa voi jättää lisäys pyynnön Epic Gamesille, jonka jälkeen Epic Games tarkistaa koodin ja päättää lisääkö sen osaksi pelimoottoria. Epic Games myös lisää kaikkien niiden kehittäjien nimet pelimoottorin virallisten kehittäjien nimien joukkoon, joiden koodia käytetään osana pelimoottoria. (Epic Games 2018g)

3 Unreal Editor

Pelikehitys alkaa tarvittavien ohjelmistojen latauksesta ja asennuksesta. Unreal Engine 4 -pelimoottori ja sen kehitystyökalun saa ladattua Epic Gamesin launcherista**,** jonka saa ladattua suoraan Epic Gamesin verkkosivuilta. Unreal Engine 4 -pelimoottorin uusin julkaistu versio tätä työtä aloittaessa oli 4.19.2. Epic Gamesin launcherista saa myös ladattua pelimoottorin vanhempia versiota versioon 4.0.2 asti. Pelimoottorin versio 4.19.2 on kuitenkin viimeisin vakaa versio pelimoottorista ja on aina suositeltavaa käyttää viimeisintä vakaata versiota ohjelmista, mutta kannattaa myös pitää mielessä, että kehitystyökalun version vaihtaminen keskellä projektia ei ole suositeltavaa, jos se ei tuo mitään lisäarvoa projektille esim. uusia hyödyllisiä ja käytettäviä ominaisuuksia.

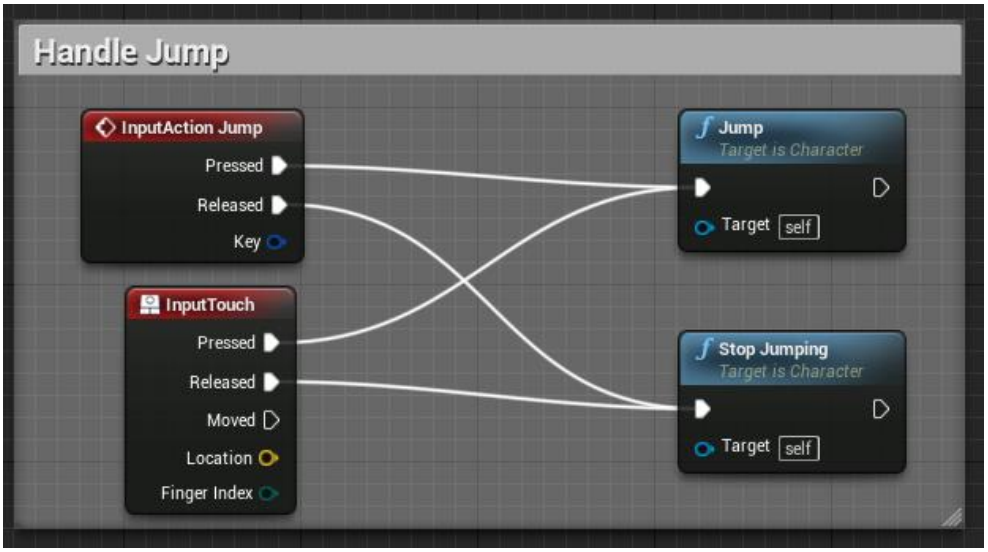


Kuva 1 Unreal Editorin projektin luonnin perusnäkymä

3.1 Blueprint Visual Scripting-järjestelmä

Unreal Editor ei ole myöskään täysin koodiin perustuva kehitystyökalu Unreal Editor tarjoaa natiivina kehitystyökalun C++-koodikielellä ja Unreal Enginen omalla Blueprint Visual Scripting-järjestelmällä. Epic Games:in mukaan Blueprint Visual Scripting-järjestelmä on eräänlainen pelin/pelitoimintojen skriptaus järjestelmä, joka pohjautuu konseptiin solmupohjaisesta rajapinnasta, jota käytetään peli elementtien luomiseen Unreal Editorissa. Kuten monissa muissakin laajasti käytettävissä skriptikielissä, sitä käytetään olio-luokkien- tai olioiden määrittelemiseen pelimoottorissa. Unreal Editoria käyttäessä saattaa huomata, että Blueprint-järjestelmän määrittelemät oliot, joihin viitataan, ovat kutsumanimeltään Blueprinttejä.

Blueprint-järjestelmä on erittäin joustava ja kattava työkalu, mikä mahdollistaa suunnittelijoiden työskentelyn virtuaalisesti täysimittaisissa konsepteissa ja työkaluissa, mikä on aikaisemmin ollut yleensä vain saatavilla ohjelmoijille. Tämän lisäksi Blueprinteille spesifioitu merkkaus on saatavilla Unreal Enginen C++-täytäntöönpanossa, joka antaa ohjelmoijien luoda uusia perustason järjestelmiä, joita suunnittelijat voivat laajentaa. (Epic Games 2018f)



Kuva 2 hyppyä käsittelevä Blueprint-skripti

3.2 Unreal Enginen käytön esivaatimukset

Esivaatimuksina Unreal Enginen käytölle on ilmaisen Epic Games käyttäjän omistaminen ja järjestelmävaatimuksien täyttäminen. Epic Games käyttäjäksi voi rekisteröityä Epic Gamesin omilla verkkosivuilla ja sieltä myös näkee Unreal Enginen järjestelmävaatimukset. Vertailun vuoksi Unreal Engine 4:n järjestelmävaatimukset ja opinnäytetyön tekijän kokoon pano listattiin Taulukkoihin 1 ja Taulukkoon 2. (Epic Games 2018a)

Taulukko 1. Unreal Engine 4 -pelimoottorin suositellut järjestelmävaatimukset

Käyttöjärjestelmä	Windows 7/8 64-bittinen tai MacOS 10. 13 High Sierra tai Ubuntu 15.04
Prosessori	Neli-ytiminen Intel tai AMD, 2.5GHz tai nopeampi.
Muisti	Windows: 8GB MacOS: 8GB Linux: 16GB
Grafiikkakortti	Windows: DirectX 11 tukeva grafiikkakortti MacOS: Metal 1.2 tukeva grafiikkakortti Linux: Nvidia GeForce 470GTX tai uudempi

Taulukko 2. Työssä käytettävä kokoonpano

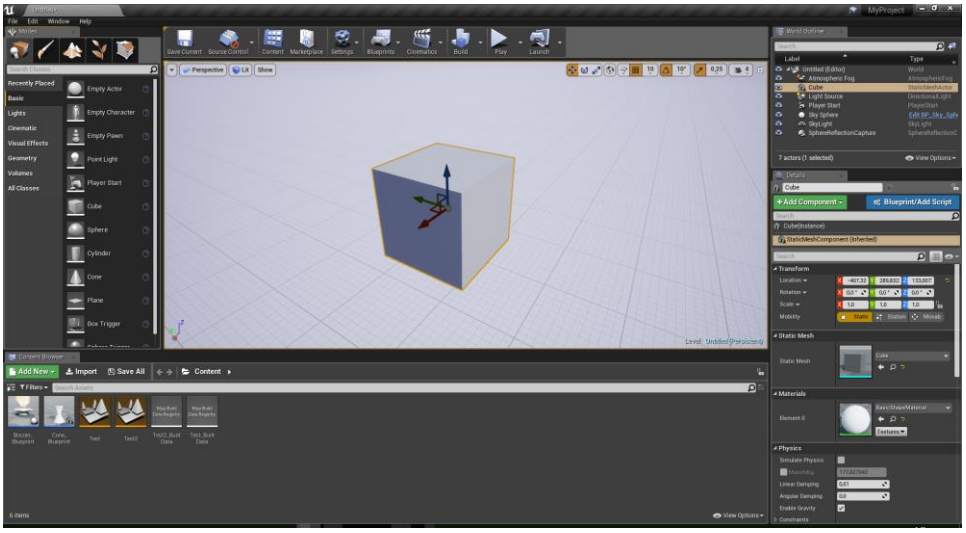
Käyttöjärjestelmä	Windows 10 Pro 64-bit
Prosessori	Intel Core i7 4770K 3.50GHz
Muisti	16GB
Grafiikkakortti	MSI Nvidia GeForce GTX 970

3.3 Actorit

Actorit ovat olioita, joita voi sijoittaa vapaasti pelikentälle. Actors on geneerinen luokka, joka tukee muutoksia, kuten muuntoa (translation), kääntöä(rotation) ja skaalaamista (scale). Actoreita voi synnyttää (Spawn) ja tuhota koodin puolelta käyttäen joko C++:saa tai Blueprinttejä. C++:saa käyttäessä AActor-luokka on kaikkien Actorien pohja luokka. (Epic Games 2018c; Epic Games 2018e)

Unreal Engine sisältää monia erilaisia Actoreita jotka sisältävät, mutta eivät rajoitu seuraaviin: StaticMeshActor, CameraActor ja PlayerStartActor. (Epic Games 2018e)

StaticMeshActor on normaalisti jokin muoto, mikä näkyy visuaalisesti pelikentällä. Unreal Engine sisältää, muutamia perusmuotoja, joista yksi on näkyvillä kuvassa 3.



Kuva 3 StaticMeshActor: Cube

Kuvan 3 vasemmassa reunassa on valikko, josta voi vetää valmiita Actoreita pelikentälle. Kuvassa 6 käytän Cube-nimistä Actoria. Kuvassa 3 myös näkyy Actorin sisällä kolme eriväristä nuolta. Nämä kolme nuolta kuvastavat X, Y ja Z akselia, joita voi säätää oikeassa reunassa sijaitsevasta ”Transform” valikosta.

3.3.1 Actorien luonti

Uusien AActor luokan instanssien luomista kutsutaan ”Spawnaamiseksi”. Tämä voidaan esimerkiksi tehdä käyttämällä geneeristä funktiota nimeltä SpawnActor(). (Epic Games 2018e)

3.3.2 Komponentit

Actorit voi kuvitella kokoelmana objekteja, joita kutsutaan komponenteiksi. Erilaisilla komponenteilla voidaan säädellä esimerkiksi, miten actori liikkuu tai miten actori renderöidään. Toinen actorien päätehtävä on ominaisuuksien ja funktioiden kutsujen replikointi verkkoon ohjelman ollessa päällä. Komponentit liitetään ne sisältävään actoriin, kun actori luodaan. Jokaisella actorilla on RootComponent-niminen ominaisuus, joka määrää, että mikä komponentti toimii actorin pohjana. Actoreilla itsellään ei ole ominaisuuksia, jonka takia niillä ei ole määritettyä paikkaa, rotaatiota tai skaalausta. Sen sijaan ne ovat riippuvaisia niiden sisältämien komponenttien tai tarkennettuna niiden RootComponenttina toimivan komponentin arvoihin. (Epic Games 2018e)

Käytännössä kehittäjä voi luoda yhden actorin, jolle lisää komponentteina lisää ominaisuuksia esimerkiksi, jos pelaajalla olisi vaikkapa soihtu kädessä, niin kehittäjä voisi lisätä soihdun tulipäähän valonlähteen, joka näin ollessaan loisi valaistuksen soihdun ympärille.

3.4 Blueprintit

Blueprintit ovat Unreal Enginelle ominaisia visuaalisesti ohjelmoituja skriptejä. Skripteillä voidaan mm. ohjelmoida peliin erilaisia toimintoja tai tapahtumia. Suurin osa Unreal Enginen valmiista pohjista sisältävät pelaajalle valmiiksi luodut kontrollit hahmon liikuttamista varten, jotka on tehty Blueprinteillä. Muita esimerkkejä Blueprinttien käytöstä ovat esimerkiksi Unreal Engine 4 -pelimoottorin dokumentaatioissa annettu esimerkki nappulasta, joka avaa oven, kun sen päälle astuu pelaajan hahmolla. Muita hyviä käyttökohteita Blueprinteille ovat esimerkiksi HUD, objektien spawnaaminen tai tuhoaminen ja kameran kulman vaihtaminen, kun vaikkapa painaa, jotain nappulaa tai osuu johonkin objektiin. (Epic Games 2018f)

3.4.1 Blueprint-luokat

Blueprint-luokat ovat ideaalisia ohjelman interaktiivisien osien luomiseen, kuten esimerkiksi ovet, nappulat, katkaisijat, keräiltävät- tai hajoavat tausta esineet. Blueprintillä voit esimerkiksi tehdä skriptit, joista alkaa jokin tapahtuma ketju esimerkiksi, kun jotain tiettyä nappulaa painaa, niin ovi avautuu tai jokin valo syttyy päälle. (Epic Games 2018f)

4 PC-pelin kehitys Unreal Engine 4 -pelimoottorilla

4.1 Projektin tavoitteet

Tämän opinnäytetyön käytännöllisen osuuden tavoitteena on luoda yksinkertainen ja valmis PC-peli käyttäen Unreal Engine 4 -pelimoottoria ja sen sisältämiä työkaluja. Pelin tekemisen tarkoituksena on oppia Unreal Engine 4 -pelimoottorin käyttämistä pelikehityksessä paremmin. Ennen opinnäytetyön aloittamista opinnäytetyön tekijä oli käynyt läpi tutoriaaleja Unreal Engine 4 -pelimoottorista oppiakseen paremmin sen käyttöä pelikehitystyökaluna. Tämän opinnäytetyön toisena tavoitteena on myös kertoa lukijalle pelinkehittämisestä ja sen eri vaiheista käyttäen Unreal Engine 4 -pelimoottoria. Työn tarkoituksena ei ole kuitenkaan toimia ohjeena lukijalle. Grafiikat tätä projektia varten luotiin käyttämälle Paint.net vapaanlähdekoodin grafiikka ohjelmaa.

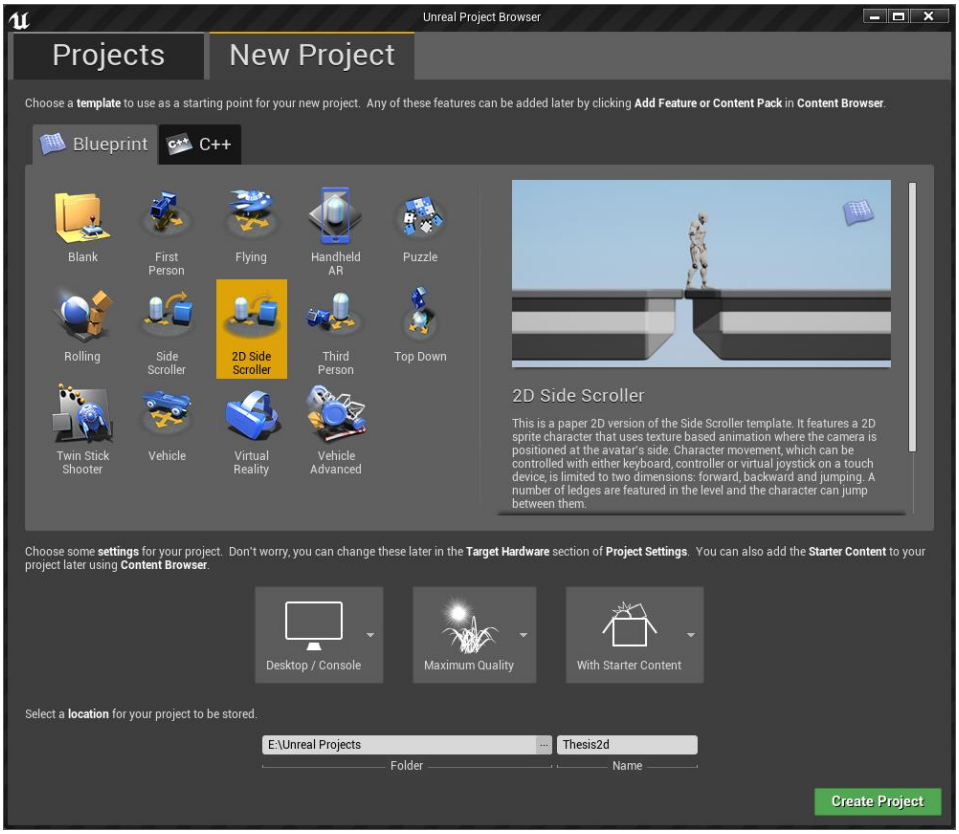
4.2 Suunnittelu

Opinnäytetyön tekijä ei ole ikinä tehnyt mitään valmista peliä Unreal Engine 4 -pelimoottorilla, joten opinnäytetyön tekijä päätti, että opinnäytetyötä varten olisi hyvä valita jokin yksinkertainen ja idealtaan helppo peli. Valmiita projekti pohjia katsoessa opinnäytetyön tekijä päätti, että valmiista pohjasta olisi hyvä aloittaa, koska niihin sisältyy hyviä esimerkkejä toimivista Blueprint ratkaisuista esimerkiksi hahmon liikkumiseen ja animaatioihin. Hyvin yksinkertainen kaksiulotteinen tasohyppelypeli olisi verrattain helppo tehdä, koska tasohyppely peleissä ei tarvitse olla mitään monimutkaisia tai erikoisia mekaniikkoja hahmon liikkumisen lisäksi ja tähän tarkoitukseen tehdyssä valmiissa pohjassa on jo tehty osittain toimiva Blueprint ratkaisu.

Tasohyppelypeleissä pohjimmainen idea on aina päästä hahmolla kentän alusta kentän loppuun asti kuolematta matkalla. Yleisesti ottaen tasohyppely peleissä kartalle sijoitetaan liikkuvia ja/tai paikallaan pysyviä vihollisia ja esteitä sen laisiin kohtiin, että pelaaja joutuu miettimään hetken, kuinka hän pääsee etenemään kentällä kuolematta. Tyypillisiä esteitä tasohyppely peleissä pelaajalle ovat erilaiset yksinkertaiset viholliset, jotka yleensä ottaen ampuvat projektiileja tai yrittävät juosta pelaajaa kohti, sekä erilaiset ansat, joihin koskettaessa pelaaja menettää yhden osumapisteen tai pelaaja häviää pelin.

4.3 Projektin aloitus

Unreal Engine 4:ssä projekti aloitetaan valitsemalla projektille ensin oikeat asetukset. Tätä työtä varten tehtävää projektia varten päätettiin, että tehdään 2D tasohyppely peli.



Kuva 4 Projektin aloitusasetukset

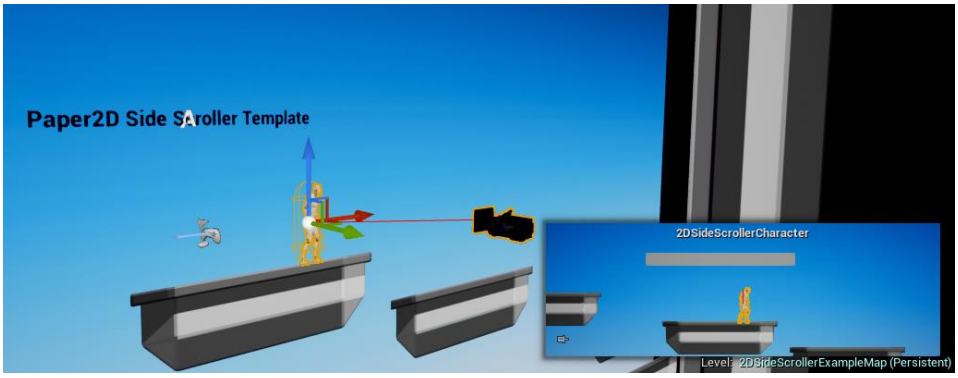
Koska tässä työssä peli tehdään PC:lle, niin alustaksi täytyy valita ”Desktop/Console” ja grafiikoiden puolesta voi huoletta valita ”Maximum Quality” eli maksimi laatu, koska tietokoneissa grafiikkakorteilla on oletusarvoisesti tarpeeksi laskenta tehoa tällaisen pelin pyörittämiseen.

4.4 Valmiiksi määritellyt pohjat

Unreal Engine 4:ssä on tätä työtä tehdessä ollut 14 valmista pohjakenttää Blueprinttejä käyttäessä ja C++-toteutuksessa 12 valmista pohjakenttää. Nämä pohjakentät sisältävät tyypillisesti muutaman valmiiksi tehdyn perus komponentin ja ympäristön. Blueprint- ja C++-toteutuksien välisellä valinnalla ei ole mitään suurempaa merkitystä, kuin se, että esimerkki toiminnot, kuten hahmon liikuttaminen on tehty sillä toteutuksella minkä tässä kohtaa valitsee. Jo heti projektin luomisen jälkeen pystyy käyttämään C++-koodia ja blueprinttejä sekaisin.

4.4.1 2D Side Scroller

Tätä työtä varten valittiin blueprint-pohja nimeltä ”2D Side Scroller”. Pohja itsessään sisältää liikuteltavan esimerkkihahmon, jolle on tehty muutama animaatio ja ohjaus valmiiksi, yksi GameMode ja yksi esimerkkikenttä, joka sisältää muutaman tason, joidenka päälle voi hypätä.



Kuva 5 Kuva 2D Side Scroller pohjasta, oikeassa alakulmassa näytetään Character-actoriin kiinnitetyn kameran kuvakulma.

4.4.2 Side Scroller

Koska tätä työtä varten päätettiin tehdä tasohyppely peli, niin toinen varteen otettava pohja olisi ollut ”Side Scroller” niminen pohja mikä on käytännössä sama toimivuudeltaan, kuin 2D Side Scroller, mutta grafiikkoina käyttäisi kolmiulotteisia muotoja. Tätä työtä varten valittiin kuitenkin 2D pohja, koska siihen on verrattain helppoa tuottaa omia ok-tasoisia grafiikoita ilmaisilla kuvankäsittelyohjelmilla.

4.5 Hahmon luominen ja muokkaaminen

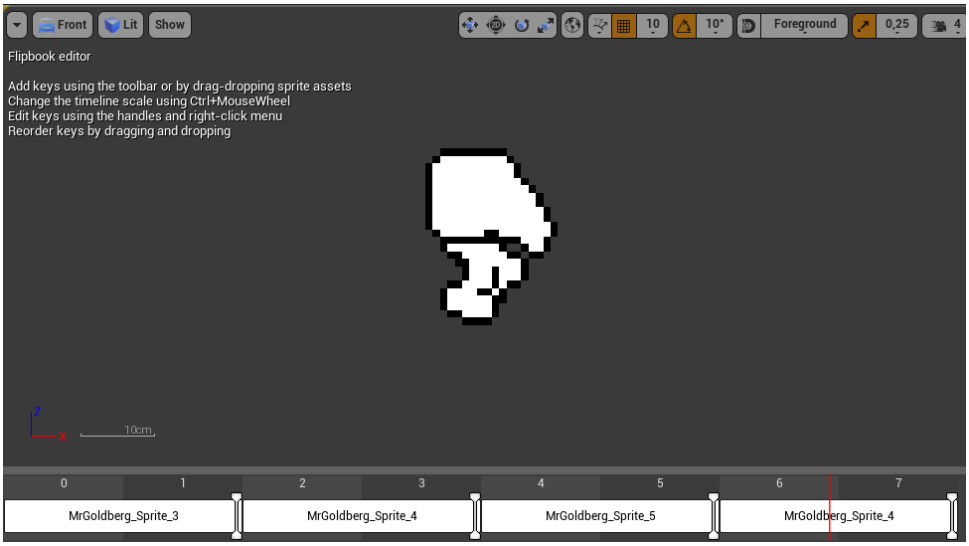
2D Side Scroller peleissä hahmot ja niiden animaatiot luodaan lähes aina Sprite grafiikoita käyttämällä. Unreal Enginessä käytännössä hahmolle annetaan Spriten kokoa vastaava collisioni ja sitten eritellään eri objekteihin törmätessä toiminnon esimerkiksi viholliseen törmätessä toiminto voi olla kuolema tai osumapisteiden väheneminen yhdellä. Unreal Engine 4:ssä voi käyttää Sprite Sheet:tejä, jotka ovat käytännössä kuvia mitkä sisältävät monta eri Sprite:ä yleensä yhteen Sprite Sheettiin laitetaan yhden objektin kaikki animaatiot mukaan. Unreal Engine 4:ssä Sprite Sheetistä on todella helppo erotella Spritet erilleen Sprite Sheetistä ja ne täytyykin erotella, jotta animaatiot saadaan eroteltua toisistaan luomalla Flipbookkeja.

Kuva 6 Tässä työssä käytettävä pelaajan Sprite Sheet, joka sisältää seisomis-, hyppy- ja kävely animaatioiden Spritet.

Kuva 7 Content browserissa näkyvä Sprite Sheet ja työkalulla toisistaan erotetut Spritet.

4.5.1 Animaatioiden luominen

Unreal Enginessä animaatiota saadaan tehtyä luomalla Spriteistä ”Flipbook” ja blueprinttien avulla hahmo saadaan käyttämään oikeita animaatioita oikeaan aikaan. Flipbookkeja luodessa niitä täytyy melkein aina kuitenkin muokata, että kuinka monta kuvaa pitkiä animaation eri osat ovat.



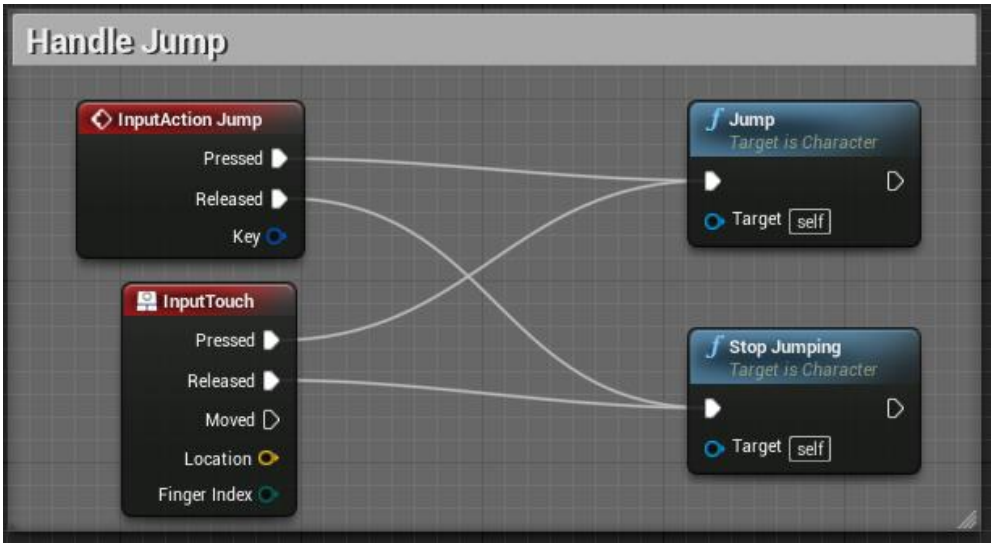
Kuva 8 Flipbook editorista, jossa on näkyvillä valmis kävelyanimaatio.

Animaatiota muunneltiin, kunnes se näytti sulavalta ja kuvan 8 tapauksessa sulava kävelyanimaatio saatiin aikaiseksi lisäämällä loppuun toinen jalanvaihto animaation osa, sekä venyttämällä animaation jokainen osio kahden kuvan pituiseksi. Oletuksena Unreal Engine 4, laittaa kaikki osat yhden kuvan pituisena.

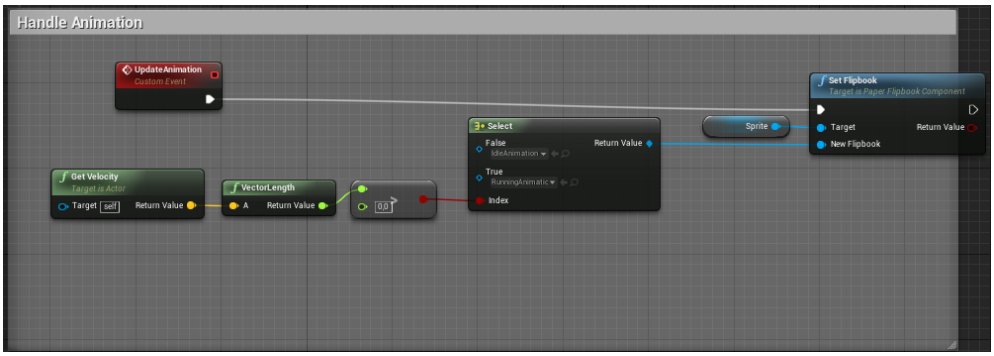
4.5.2 Hyppyanimaation käyttäminen

Projekti pohjassa oli valmiiksi tehty toimiva ratkaisu kävelyanimaation pyörittämiseen. Kuitenkin hypätessä hahmo vain pyörittää tätä kävelyanimaatiota ja tarkoitus olisi, että se pyörittäisi hyppyanimaatiota. Aihetta tutkiessa huomattiin, että animaatioiden pyörittämiseen blueprinteillä löytyy muutama erilainen tapa. Ensimmäinen tapa on luoda uusi Boolean muuttuja blueprint editorissa ja asettaa sille Set-funktio, joka muuttaa arvon True:ksi eli todeksi hypyn alkuun ja Set-funktion, joka palauttaa arvon

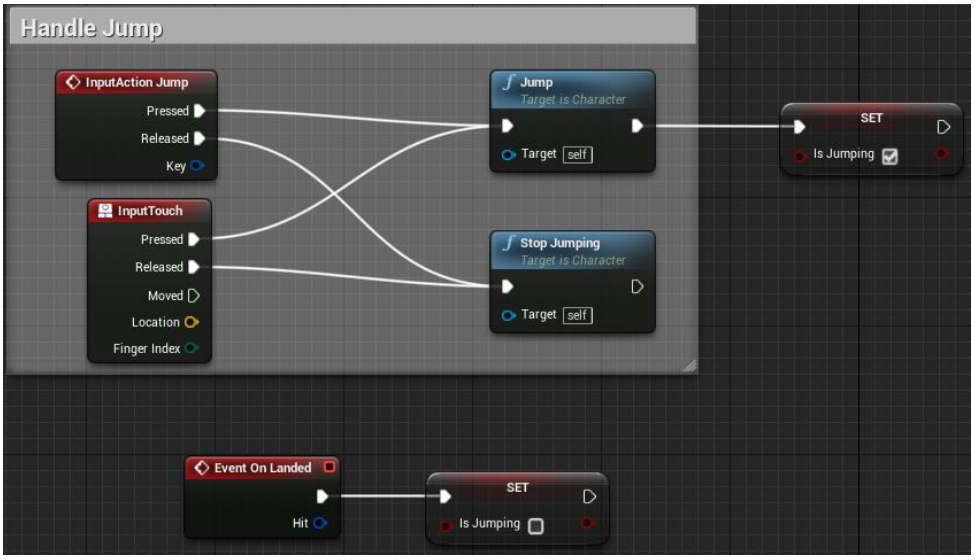
False:ksi eli epätodeksi, kun hahmo osuu maahan, sekä riippuen tämän muuttujan arvosta hahmo näyttäisi animaation sen mukaan. True set-funktion paikka oli verrattain helppo päätellä, koska esimerkki blueprinteissä on tehty erikseen osio hyppäämiselle. False set-funktio taas kuului laittaa erillisen ”Event on Landed” funktion jälkeen, kun funktio laukaistaan. ”Event on Landed” funktio laukaistaan aina, kun blueprintin kohde eli tässä tapauksessa pelaajan hahmo osuu maahan tai tasoon, jonka päälle voi mennä. Animaation vaihtamisen pystyi esimerkki blueprinttiä käyttäen helposti päättelemään, sillä esimerkki blueprintissä on käytössä ”Update Animation” funktio. ”Update Animation” funktion jälkeen piti lisätä haara johon laitettiin ehdoksi aiemmin luodon muuttujan tilat eli True ja False, jolloin aina, kun ehto täyttyy, niin hahmo näyttää animaation sen mukaan. Vertailun vuoksi kuvissa 9 ja 10 alkuperäisen esimerkki blueprintin osat ja kuvissa 11 ja 12 tämän tavan mukaiset blueprintit.



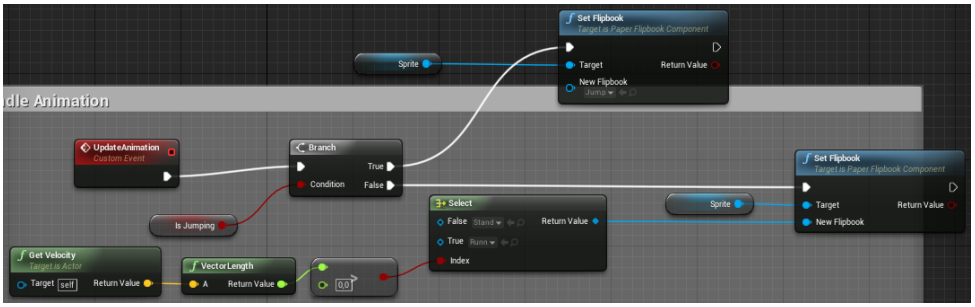
Kuva 9 Esimerkki Blueprintin hypyn käsittelevä skripti



Kuva 10 Esimerkki Blueprintin Animaation käsittelevä skripti



Kuva 11 Hyppyanimaatiota varten muokattu hypynkäsittelijäskripti



Kuva 12 Hyppyanimaatiota varten muokattu animaationkäsittelijäskripti

Tässä tavassa on kuitenkin yksi heikkous tai huonopuoli. Tässä tavassa jokaiselle animaatiolle täytyy erikseen luoda oma muuttuja ja asettaa sille set Funktiot, sekä luoda uusi haara, jokaista uutta animaatiota kohden.

Toinen tapa toteuttaa tämä sama hyppyanimaation toiminnallisuus blueprintissä on ensin luoda Enumeration-muuttuja, joka muistuttaa hieman perinteisissä koodikielissä käytettävää String-listaa. Erona perinteisten koodikielien String-listaan on kuitenkin se, että Enumeration-muuttujassa voi olla vain yksin sen sisältämistä arvoista aktiivisena, tämä siis tarkoittaa sitä, että käytännössä se on muuttuja, jolla voi olla yksi aikaisemmin määritetyistä arvoista aktiivisena ja tätä aktiivista arvoa voi vaihtaa blueprintin puolella set-Funktiolla. Listaan voi lisätä pelkästään merkkijonoja, mutta näitä merkkijonoja voidaan käyttää blueprinteissä.

4.6 Kentän luominen ja muokkaaminen

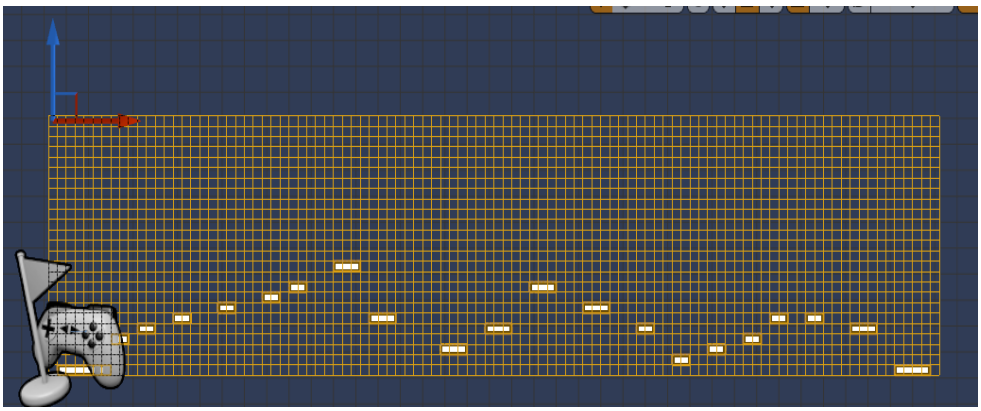
Side Scroller peleissä pelikentän luominen onnistuu muutamalla eri tavalla, yksinkertaisin tapa on tehdä Sprite-grafiikoilla erilaisia muotoja ja erottamalla Spritet toisistaan samalla tavalla, kuin hahmon animaatio

Spritet. Näille Spriteille voi myös erikseen säätää Collision-alueen, sekä renderöitymis-alueen.

pääsääntöisesti pelikentän luominen koostuu Tile eli laattagrafiikoilla rakennetuista muodoista pelikentälle. Varsinkin Side Scroller peleissä ns. kentän pohja koostuu yleensä jatkuvasta yhden värisestä Tilestä.

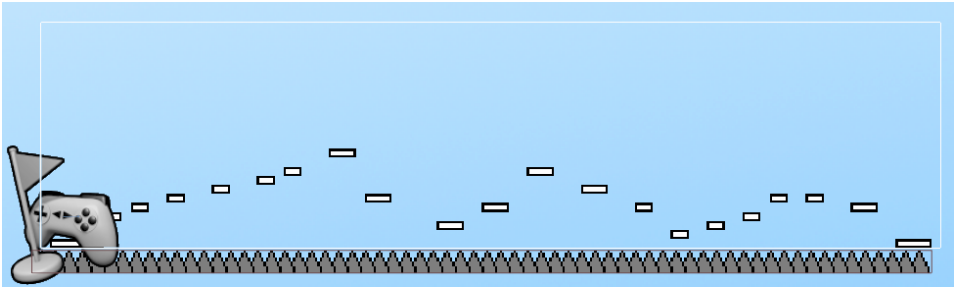
Toinen tapa tehdä kentän grafiikoita on luoda Tilemap, erilaisista Tilen osista, joista voi sitten yhdistelemällä luoda itse erilaisia muotoja kentälle. Ensimmäinen tapa on siis enemmänkin valmiiden muotojen laittamista peliin, kun taas tämä tapa on pienien muotojen osien laittamista peliin.

Tätä opinnäytetyötä tehdessä päätettiin, että jälkimmäinen tapa on parempi tapa, koska Tilemapissa, osat menevät pikselin tarkasti oikealle paikalle, eikä Tilemapin käyttö rajoita kentän luomista millään tavalla. Koska opinnäytetyön yhtenä tavoitteena oli tehdä yksinkertainen ja toimiva peli, niin pelissä käytettäviä grafiikoita päätettiin yksinkertaistaa huomattavasti. Kuvassa numero 13 on näkyvillä miltä Tilemappia käyttäen tehty kenttä näyttää. Kentän vasemmassa ja oikeassa reunassa on myös näkyvillä yksi ”tyhjä” ruutu, koska ne on täytetty läpinäkyvillä tileillä, jotta pelaaja ei pääsisi hyppäämään kentän ulkopuolelle.



Kuva 13 Tilemapin avulla tehty kenttä

Tämän jälkeen luotiin myös muutama muu kenttä, koska monivaiheisen valikon toiminta blueprinteissä vaikutti hyödylliseltä selvittää tätä työtä varten. Lisäksi jokaiseen kenttään laitettiin tausta grafiikat, koska ilman niitä jokaisessa kentässä taustalla olisi vain tyhjä harmaa alue ja koska alkuperäisenä ideana oli tehdä yksinkertainen tasohyppelypeli, niin jokaisen kentän alareunaan lisättiin ylöspäin osoittavat piikit, joihin osuessa peli loppuu. Näiden muutosten jälkeen ensimmäinen kenttä oli kuvan 14 mukainen.



Kuva 14 ensimmäinen kenttä muutosten jälkeen

Tässä kohtaa kaikki kentät olivat visuaalisesti tarpeeksi miellyttävän näköisiä, sekä tarpeeksi erilaisia keskenään, että niihin ei tarvitse enää suuria visuaalisia muutoksia tehdä tämän työn aikana.

Unreal Engine 4:ssä kentälle voi lisätä toiminnallisuuksia muuttamalla Level Bluerpinttiä. Työhön luotiin blueprintti, joka piilottaa kursorin peli kentän auetessa, koska kursori on määritetty näkyväksi jokaisen valikon auetessa. Nämä molemmat voidaan tehdä helposti käyttämällä ”Set Show Mouse Cursor”-funktiota.

4.7 Valikkojen luominen

Valikkojen luominen Unreal Engine 4:ssä on erittäin helppoa. Valikot ovat olennainen osa jokaisen pelin käyttöliittymää, koska muuten useimmissa peleissä olisi hyvin vaikeata saada selkoa pelin etenemisestä. Valikkoja voi myös käyttää kahdella eri tavalla ensimmäinen tapa on ns. pysäyttää muut pelitoiminnot siksi ajaksi, kun valikkoa käytetään ja toinen tapa on sulkea muut toiminnot kokonaan, kun valikko aktivoituu ja avata uudet toiminnot valikon jälkeen. Työtä tehdessä päätettiin käyttää molempia tapoja, koska vain toista tapaa ei voinut käyttää jokaista valikkoa varten.

Valikkojen visuaalinen näkymä ja toimivuus määritetään Widget Blueprintissä. Widget Blueprint on nimenomaan käyttöliittymän osia varten tehty Blueprint. Valikkoja luodessa on myös helppo pitää ns. samankaltainen tyyli, koska visuaalisen osuuden voi kopioida suoraan toisiin valikkoihin, niin, että valikkojen osien sijainnit säilyvät samana. Tätä käytettiin hyödyksi valikkoja tehdessä erittäin paljon.

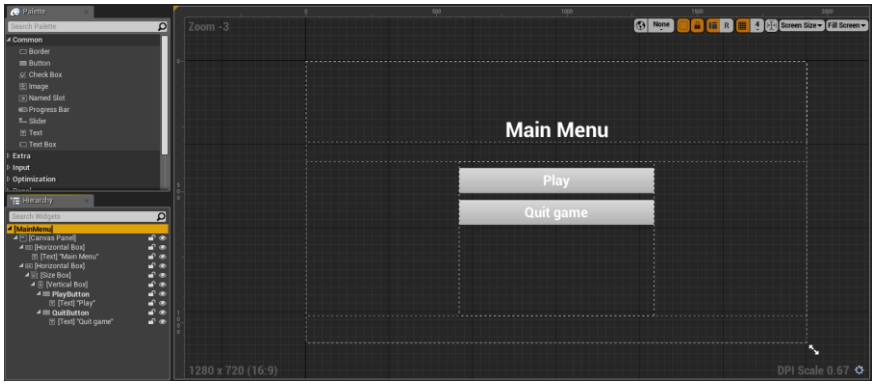
Tässä työssä tehdään ainakin kolme eri kenttää eli tätä työtä varten pitää luoda vähintään päävalikko, kenttävalikko ja kaksi erilaista pelin päättymisvalikkoa, joista toinen on kentän läpipääsyä ja toinen kentässä epäonnistumista varten.

4.7.1 Päävalikko

Päävalikko oli ensimmäinen valikko, joka tässä työssä luotiin ja sitä tehdessä tuli tutuksi valikkojen luomisen periaatteiden perusteet. Päävalikko

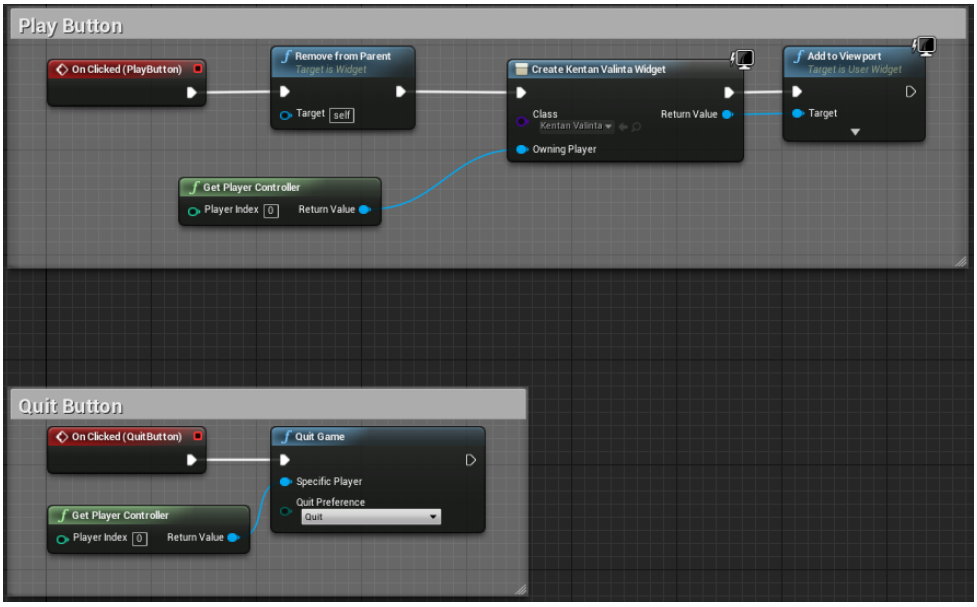
luottiin käyttämällä toista edellä mainittua tapaa, koska muuten ensimmäisen kentän grafiikat olisivat näkyneet taustalla.

Päävalikkoa tehdessä valikosta tehtiin silmälle miellyttävän näköinen ja siihen laitettiin myös kaksi nappulaa, joista toinen nappula siirtää pelaajan kenttävalikkoon ja toinen sulkee pelin. Kuvassa 15 on näkyvillä miltä lopullinen valikko näytti muokkausnäkymässä. Kuvan vasemmassa reunassa ”Hierarchy” välilehdelle on myös näkyvillä mitä osia valikon luomisessa käytettiin.



Kuva 15 Päävalikon lopullinen ulkonäkö muokkaustilassa

Sen jälkeen, kun visuaalinen näkymä oli valmis, piti päävalikolle lisätä toimivuus käyttämällä blueprinttejä. Valikoissa blueprintit ovat aina verrattain yksinkertaisia, koska valikoissa yleensä ottaen vain siirrytään toiseen valikkoon tai näkymään pelissä. Kuvassa 16 on näkyvillä Päävalikon blueprintti, josta pystyy näkemään, että siinä lisättiin molemmille nappuloille ”On Clicked”-funktio, joka Play Buttonin tapauksessa ottaa napin painalluksen jälkeen sen itsensä sisältämän Widgetin pois näkymästä ”Remove from Parent”-funktiolla, jonka jälkeen se luo uuden Widgetin käyttämällä ”Create Widget”-funktiota, joka tämän nappulan tapauksessa on Kenttävalikon Widget ja lisää sen näkymään eli ”Viewport”:iin. ”Get Player Controller”-funktio taas pitää huolen, että pelaaja voi painaa nappuloita ensimmäisen valikon poistamisen jälkeen. Quit Buttonilla on taas todella yksinkertainen blueprintti, joka ajaa vain valmiin ”Quit Game”-funktion, joka sulkee pelin.

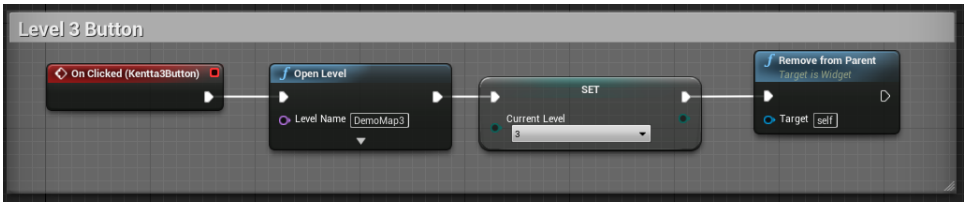


Kuva 16 Päävalikon Widget Blueprint

4.7.2 Kenttävalikko

Kenttävalikko oli huomattavasti nopeampi tehdä, koska visuaalisen ulkonäön pohjaksi pystyi kopioimaan Päävalikon visuaalisen näkymän, johon lisättiin muutama uusi nappula ja tietenkin muutettiin muutama teksti. Koska kenttiä oli kolme kappaletta, niin jokaista eri kenttää varten luotiin yksi nappula ja lopuksi valikkoon lisättiin vielä takaisin meno nappula, josta pystyisi palaamaan päävalikkoon.

Kenttään siirtyminen blueprintin puolella on todella helppo toteuttaa valmiilla ”Open-level”-funktiolla ja tämän jälkeen set-funktio asettaa valitun kentän numeron Enumeration-muuttujan valituksi arvoksi ja tämän jälkeen kutsutaan ”Remove from Parent”-funktiota poistamaan valikko näkymästä aivan, kuten Päävalikossakin. Kuvassa 17 on näkyvillä kolmoskentän valitsemis nappulan valmis blueprintti.

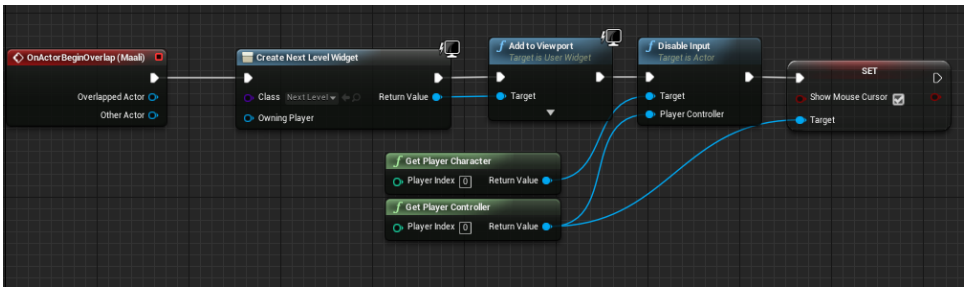


Kuva 17 kolmoskentän valitsemis nappulan valmis blueprint.

Takaisin nappulan blueprintin pystyi lähes täysin kopioimaan Päävalikon Play buttonilta, koska sen toiminta tapa on täysin sama. Ainoana erona siihen on se, että ”Create Widget”-funktion kohteena tämän nappulan kohdalla on Päävalikko kenttävalikon sijaan.

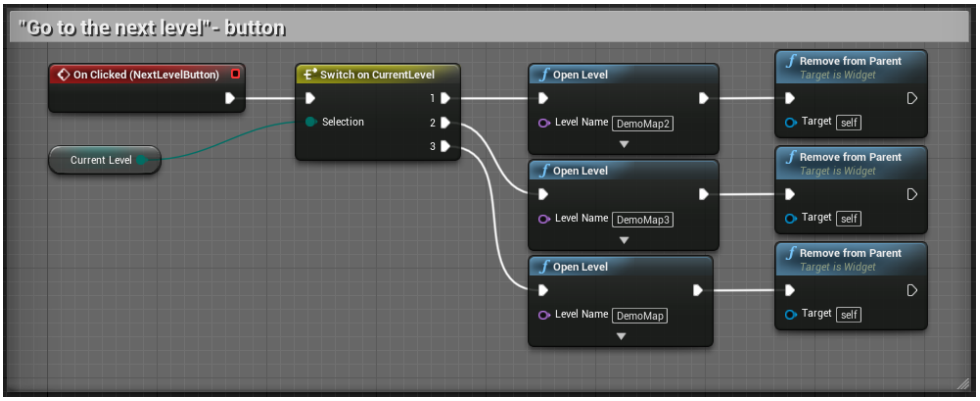
4.7.3 Voittoruutu

Kun pelaaja pääsee kentän loppuun asti kuolematta, niin pelaajalle pitäisi aueta niin sanottu voittoruutu pelinäkymään. Työssä luotiin tämän lainen voittoruutu, jossa on kaksi nappulaa, joista toinen sulkee pelin ja täysin samanlainen, kuin ”Quit Game”-nappula, jota käytettiin aikaisemmin tehdyssä valikossa. Toiminnallisuus suoritettiin luomalla uusi actori, jolle lisättiin yksi komponentti, joka oli ”Collision Box”-komponentti, jolla pystyy määrittämään, että missä kohtaa osuma rekisteröidään. Kuvassa 18 on näkyvillä ”Maali”-actorin blueprint, jonka pystyi kopioimaan täysin, myöhemmin käytettävään ”Piikit”-actorin blueprinttiin vaihtamalla ”OnActorBeginOverlap”-funktion kohteen, sekä sen jälkeen tulevan ”Create Widget”-funktion kohteen. ”Create Widget”-funktion kohde tässä tapauksessa oli ”Next Level”-Widget Blueprint, joka toimi Voittoruudun valikkona.



Kuva 18 ”Maali”-Actorin valmis Blueprint.

Muista valikoista poiketen voittoruudun ”Next Level”-valikossa on ”Go to the Next Level”-nappula, joka nykyisestä kentästä riippuen menee seuraavaan kenttään, sitä painaessa. Kuvassa 19 on näkyvillä tämän valikon blueprint toteutus, jossa näkyy, että nappulan painamisen jälkeen Enumeration muuttujan ”CurrentLevel” arvosta riippuen kytkimen jälkeen avataan uusi taso, jonka jälkeen valikko poistetaan näkymästä.



Kuva 19 ”Go to the next level”-nappulan blueprint toteutus.

4.7.4 Häviämisruutu

Työssä luotiin häviämisruutu, voittoruudun lisäksi, joka käytännössä tulisi ruutuun pelin päätyttyä, ilman, että pelaaja pääsi onnistuneesti maaliin asti. Häviämisruudussa on vain kaksi nappulaa, jotka ovat ”Play Again”-nappula, sekä ”Quit Game”-nappula joidenka toiminnallisuuden voi loogisesti päätellä jo nimistä eli ”Play Again”-nappula aloittaa kentän alusta sulkemalla nykyisen kentän ja avaamalla sen uudestaan ja ”Quit Game”-nappulan, jonka toiminnallisuus kopioitiin aiemmin tehdyistä valikoista.

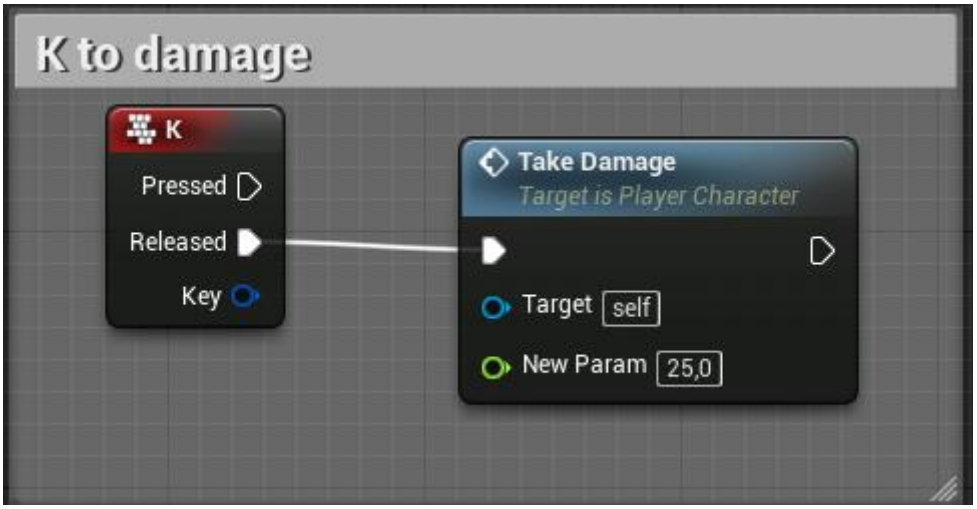
4.8 Pelin päättyminen

Yleisesti ottaen tasohyppelypeleissä peli päättyy siihen, että peli päättyy heti, kun pelaaja ottaa osuman tai vaihtoehtoisesti siihen, että pelaajan osumapisteet loppuvat esimerkiksi kolmen osuman jälkeen. Tässä työssä päätettiin, että kokeilunvuoksi voisi kokeilla molempia tapoja, edes jollain asteella.

Ensimmäisenä mainitulla tavalla yksi toteutustapa on luoda actori, johon koskettaessa peli päättyy. Koska jokaisen kentän alareunassa on piikit, niin loogisesti näihin koskettaessa pelaaja joko ottaa vahinkoa tai peli päättyy. Tässä vaiheessa päätettiin, että piikkeihin koskettaessa peli päättyy välittömästi. Tässä, siis luotiin actori, jolle lisättiin visuaaliseksi ulkomuodoksi ”PaperSprite”-komponentti, johon lisättiin kohteeksi Sprite, jota oli käytetty piikeissä ja joita oli jo käytetty aikaisemmin pelikentillä vain ulkonäön vuoksi. Tämän jälkeen vielä lisättiin ”Collision Box”-komponentti actorille osuman rekisteröimistä varten. Toiminnallisuus piikeille saatiin helposti, sillä piikeissä idea oli sama, kuin Maali-actorilla eli: pelaajan osuessa actoriin aukeaa valikko. Tässä kohtaa siis kopioitiin ”Maali”-actorilla käytetty blueprintti kokonaan, mutta vaihdettiin ”Create Widget”-funktion kohde häviämisruudun ”Game Over”-valikoksi, joka luotiin aiemmin.

Tässä kohtaa huomattiin, että toinen tapa ei edes oikein sopisi tähän työhön, mutta kuitenkin, vain esimerkin vuoksi tässä työssä tehdään yksinkertainen toteutus siitä. Ensimmäisenä luotiin pelaaja hahmolle uusi Float-muuttuja nimeltä ”Health” eli terveys. Oletus arvoksi muuttujalle laitettiin 100,0. Tämän jälkeen luotiin uusi funktio nimeltä ”Take Damage”, johon lisättiin yksi Float-parametrin kuvastamaan otettua vahinkoa. Sen jälkeen lisättiin ”Float – Float”-funktio, joka tuo kahden float-muuttujan arvojen erotuksen funktion ajon jälkeen. Tämän jälkeen lisättiin Set-funktion ”Health”-muuttujalle, joka asettaa sen arvon tuoduksi arvoksi ”Float – Float”-funktiosta, ja tämän jälkeen lisättiin haara toimimaan blueprintin If-lauseena, sekä ”float <= float”-funktion, joka tarkistaa

onko vietävä arvo pienempi tai yhtä suuri, kuin funktiolle asetettu arvo, joka oli tietenkin tässä tapauksessa 0,0. Tämä luku laitettiin tämän tarkistuksen tuloksen ehdoksi haaraan ja haaran jälkeen lisättiin, jos ehto on tosi puolelle, niin peli päättyy ja ”Game Over”-valikko aukeaa. Tämän valikon aukeamisen osuuden pystyi kopioimaan täysin jo aiemmin käytetyn ensimmäisen pelin päättymisen esimerkin yhteydessä. Tämän jälkeen pelaaja hahmon blueprinttiin lisättiin osuus, jossa ”K”-näppäintä painaessa näppäimistöllä ajetaan juuri tehty valmis ”Take Damage”-funktio, joka on näkyvillä kuvassa 20.



Kuva 20 ”Take Damage”-funktio, joka ajetaan ”K”-näppäintä painaessa näppämistöllä blueprint toteutus.

5 Yhteenveto

Ensimmäinen tämän opinnäytetyön tutkimuskysymyksistä oli, miten luodaan PC-peli Unreal Engine 4 -pelimoottoria käyttäen. Tähän tutkimuskysymykseen saatiin vastattua hyvin ja laajasti, koska opinnäytetyön aikana tehtävä peli saatiin alusta loppuun valmiiksi ja tulos on nähtävillä tässä dokumentissa. Seuraava tämän työn tutkimuskysymys oli, miten luodaan hyvä ja käytännöllinen valikko Unreal Engine 4:ssä. Tähän tutkimuskysymykseen saatiin myös vastaus. Opinnäytetyössä on esillä, kuinka peliin luodut valikot ja niiden toiminnallisuudet toteutettiin alusta loppuun. Työssä käytiin myös läpi Unreal Engine 4:n sisältämää käyttöliittymien luomista varten olevaa Widget Blueprint-työkalua. Kolmanneksi tutkittiin, miten valikoissa siirtyminen toteutetaan Unreal Engine 4:ssä. Tähänkin kysymykseen löytyy selvä vastaus opinnäytetyön käytännölliseen osuuden alta ja tähän kysymykseen saatiin myös vastattua hyvin ja laajasti, tähän kysymykseen vastaus löytyi valikkojen nappuloiden blueprint toteutuksista. Seuraavana tämän opinnäytetyön tutkimuskysymyksenä oli, että miten Unreal Engine 4:ssä hahmon liikkuminen toteutetaan? Tähän kysymykseen saatiin myös hyvä vastaus, koska hahmon liikkuminen on myös yksi toiminnallisuus, niin tähän kysymykseen löytyi vastaus pelaajan hahmon blueprint toteutuksesta ja viimeisenä tämän työn tutkimuskysymyksenä oli, että miten Unreal Engine 4:ssä animaatioiden käsittely toteutetaan? Tämän tutkimuskysymyksen vastaus oli kaksi osainen. Ensin pelaajan hahmolle piti luoda Flipbook, joka sisälsi kaikki hahmon omat Sprite-grafiikat ja sen jälkeen pelaaja hahmon blueprint toteutuksen kautta valittua animaatiota pystyi säätelemään.

Tätä opinnäytetyötä tehdessä opinnäytetyön tekijä oppi paljon Unreal Engine 4 -pelimoottorin käytöstä pelikehitystyökaluna. Kuitenkin, koska Unreal Engine 4 -pelimoottori on niin laaja, että sitä olisi mahdotonta kokonaan käydä läpi tämän kokoisessa työssä. Tämä siis tarkoittaa sitä, että Unreal Engine 4:n käytöstä pelikehitystyökaluna on opinnäytetyön tekijällä vielä paljon uutta opittavaa. Koska opinnäytetyön tekijällä on suuri kiinnostus pelikehitykseen, niin opinnäytetyön tekijän mielestä Unreal Engine 4 on aivan mahtava pelikehitystyökalu, varsinkin sellaisille henkilöille, jotka ymmärtävät ohjelmoinnin perusideoita, mutta ovat vasta-alkajia esim. pelikehityksen osa-alueella. Perus toiminnallisuudet opinnäytetyön aikana tehdyssä pelissä saatiin hyvin toteutettua ja pelissä olisi ehkä muutama toiminnallisuus mitä olisi voinut muuttaa, mutta muuten pelin toteutus on melko lähellä haluttua lopputulosta.

6 Lähteet

Epic Games 2018a. Hardware & Software Specifications. Viitattu 24.4.2018 osoitteesta https://docs.unrealengine.com/en-us/GettingStarted/RecommendedSpecifications

Epic Games 2018b. Installing Unreal Engine. Viitattu 24.4.2018 osoitteesta https://docs.unrealengine.com/en-us/GettingStarted/Installation

Epic Games 2018c. Unreal Engine 4 Terminology. Viitattu 24.4.2018 osoitteesta https://docs.unrealengine.com/en-us/GettingStarted/Terminology

Epic Games 2018d. Tools and Editors. Viitattu 24.4.2018 osoitteesta https://docs.unrealengine.com/en-us/GettingStarted/SubEditors

Epic Games 2018e. Actors. Viitattu 04.05.2018 osoitteesta https://docs.unrealengine.com/en-us/Programming/UnrealArchitecture/Actors

Epic Games 2018f. Introduction to Blueprints. Viitattu 04.05.2018 osoitteesta https://docs.unrealengine.com/en-US/Engine/Blueprints/GettingStarted

Epic Games 2018g. Frequently asked questions (FAQ). Viitattu 08.05.2018 osoitteesta https://www.unrealengine.com/en-US/faq

Epic Games 2018h. Unreal Engine Features. Viitattu 08.05.2018 osoitteesta https://www.unrealengine.com/en-US/features

InstabugBlog 2018. Top Game Engines In 2018. Viitattu 15.05.2018 osoitteesta https://blog.instabug.com/2017/12/game-engines/

Gamasutra 2008. From the past to the future: Tim Sweeney Talks. Viitattu 26.05.2018 osoitteesta https://www.gamasutra.com/view/feature/132426/from_the_past_to_the_future_tim_.php?print=1