Jarkko Velinen

2D Side-scroller -pelin toteuttaminen Unreal Enginellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

18.5.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Jarkko Velinen 2D Side-scroller -pelin toteuttaminen Unreal Enginellä 44 sivua 17.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Tietoverkot
Ohjaaja(t)	Yliopettaja Janne Salonen Lehtori Jussi Alhorinne

Insinöörityön tarkoituksena oli tutustua Unreal Engineen ja opetella tekemään toimiva 2D Side-scroller -peli sen avulla. Projektissa käydään läpi pelikehityksen työkaluja ja pelin tekemisen eri vaiheita yksityiskohtaisesti.

Peli on kaksiulotteinen sivulta päin kuvattu tasohyppelypeli, jossa pelaajahahmo etenee väistellen satuttavia objekteja ampuen vihollisia ja yrittäen päästä tason loppuun asti kuolematta.

Prosessi aloitettiin tutkimalla vaihtoehtoja ja perehtymällä Unreal Enginen mahdollisuuksiin. Pelimoottorilla on paljon potentiaalia, mutta joissain tapauksissa on haasteellista oppia käyttämään sen kaikkia ominaisuuksia hyödyksi. Muutaman kuukauden opettelemisen aikana ehdin tutustua vain pieneen osaan Unreal Enginen kokonaisuudesta.

Tuloksena peli täytti vaatimukset toimivasta pelistä. Hahmon liikkuvuus, animaatiot ja hyppiminen toimivat oikealla tavalla. Viholliset hoitivat roolinsa tarkoituksensa mukaisesti. Tekstuurit olivat sopivia. Pelin sisäiset objektit sekä liikkuvat alustat toimivat mainiosti. Pelihahmo pystyi kuolemaan ja saavuttamaan päämääränsä. Sinikopioiden erilaiset järjestelmät hoitivat tehtävänsä.

Avainsanat	Unreal Engine, peli, pelikehitys, kaksiulotteinen, sinikopio, hahmo

Author(s) Title Number of Pages Date	Jarkko Velinen Making a 2D Side-scroller game using Unreal Engine 44 pages 17 May 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Data Networks
Instructor(s)	Janne Salonen, Principal Lecturer Jussi Alhorinne, Lecturer

The purpose of this Bachelor's thesis was to get more familiar with Unreal Engine and learn how to make a functional 2D Side-scroller game using Unreal Engine. In the process of the project, there will be information about the tools used for the development of the game and detailed step by step instructions of different stages of making the game.

The game is a two-dimensional platform game viewed from the side, where the player character goes forward while dodging hurtful objects, shooting enemies and tries to reach the end of the level without dying.

The process began by investigating different options and getting acquainted with the possibilities of Unreal Engine. The game engine has lots of potential, but in some cases it is challenging to learn how to use all its features to your advantage. Within a few months of learning, I had the time to get familiar with only a small part of the entirety of Unreal Engine.

As a result, the game fulfilled its requirements of a functioning game. The movement, animations and jumping of the character worked properly. Enemies carry out their roles according to their purpose. Textures were fitting. The objects and moving platforms inside the game worked perfectly. The player character could die and reach its destination. The different systems of blueprints fulfilled their job.

Keywords	Unreal Engine, game, two-dimensional, platform, blueprints, character, development

Sisällys

Lyhenteet

VR	Lyhenne sanasta virtuaalitodellisuus, joka on tietokonesimulaation tuottamien aistimusten avulla luotu keinotekoinen ympäristö.
2D	Kaksiulotteisuus. Kaksi tasoa, voidaan käyttää esim. X- ja Z-suuntia.
3D	Kolmiulotteisuus. Kaikki kolme tasoa. X-, Y- sekä Z-suunnat.

1 Johdanto

Olen ollut erittäin kiinnostunut peleistä pienestä pitäen, ja se on vieläkin lempiharrastukseni. Halusin hyödyntää kiinnostuksiani myös tämän projektin tekemiseen.

Tarkoituksena oli tutustua syvemmin pelien tekemiseen. 2D Side-scroller tuntui olevan sopivan haasteellinen ensimmäiseksi peliprojektiksi uudella pelimoottorilla.

Pelialalla on tarjolla mm. pelisuunnittelua, ohjelmointia, 3D-mallintamista ja äänisuunnittelua, joista kaksi ensimmäistä on lähempänä omaa kiinnostusaluetta. Minulla on kuitenkin paljon opittavaa, jos haluan päästä pelialle osallistumaan pelien tekemiseen.

Peli on toteutettu Unreal Engine -pelimoottorilla käyttäen sinikopiopohjaista projektia. Sen avulla pystytään tehdä hienoja ja hyviä pelejä.

Peliteollisuus on jatkuvasti kasvava ala maailmanlaajuisesti, ja myös Suomessa. Suomen peliteollisuuden liikevaihdoksi on arvioitu 2,4 miljardia euroa 2015. Maailmanlaajuisesti pelimyynnin arvoksi arvioitiin 92 miljardia dollaria ja sen on arvioitu ohittavan 100 miljardin dollarin raja vuonna 2017. [1.]

2 Pelikehityksen työkalut

Pelinkehitykseen soveltuvia työkaluja on tarjolla hyvinkin paljon. Monet niistä ovat saatavilla ilmaiseksi. Harrastelijoinakin on helpompi päästä niihin sisään, kun hankintakustannukset ovat alhaiset.

2.1 Pelimoottorit

Pelimoottori on videopelin ohjelmistokehys, joka on suunniteltu videopelien tekemiseen ja kehittämiseen. Pelinkehittäjät voivat niiden avulla tehdä uusia pelejä monille eri konsoleille, mobiililaitteille ja tietokoneille. Pääominaisuus on yleensä jonkunlainen kuvantamismoottori, joka tarkoittaa kuvan luomista mallista tietokoneohjelman avulla. Sitä tarvitsee 2D- tai 3D-grafiikkoja varten. Siihen sisältyy myös fysiikkamoottori, törmäyksen tunnistaminen, ääniä, skriptiä, animaatiota, tekoälyä ja muita tarpeellisia ominaisuuksia. [2.]

Suosittuja pelimoottoreja ovat Unity ja Unreal Engine. Olen käyttänyt kumpaakin. Minulla oli vaikea valinta teenkö projektini Unityllä vai Unreal Enginellä. Molemmissa pelimoottoreissa on hyvät puolensa, mutta ajattelin, että voisi olla mielenkiintoista tutustua Unreal Engineen enemmän, koska en ollut sitä käyttänyt ennen nykyistä projektia.

2.1.1 Unreal Engine

Unreal Engine 4 on Epic Gamesin kehittämä pelimoottori, jolla pystyy luomaan asioita tehokkaasti. Se tarjoaa voimakkaan todistetun suorituskyvyn, johon voi luottaa. Kaikki on mukana paketissa, eikä se tarvitse mitään ylimääräisiä ostoja tai liitännäisiä. Unreal Enginen käytäntö on hyvinkin helppo. Motto perustuu siihen, että he onnistuvat, kun käyttäjäkin onnistuu. Pelimoottoria voi käyttää ilmaiseksi ja rakentaa sen avulla mitä vain. Mutta kun peliä käytetään, niin tuotoista maksetaan 5% Epic Games -yhtiölle. [4.]

Unreal Engine on täysi pakkaus kehittämistyökaluja, jotka on tehty kelle vain reaaliaikaisen teknologian työskenteleville. Se soveltuu hyvin yritysohjelmiin, elokuvallisiin kokemuksiin ja korkealaatuisiin peleihin PC:lle, konsolille, puhelimille ja VR:lle. Unreal Engine antaa kaiken, jota tarvitsee aloittaakseen, kasvaakseen ja erottumaan joukosta. [3.]

Maailmanluokan työkalusetti ja helposti lähestyttävä työnkulku vahvistaa kehittäjiä nopeasti iteroimaan ideoita ja näkemään välittömiä tuloksia ilman, että koskee riviinkään koodia. Kaikilla Unreal Engine -yhteisössä on kuitenkin myös vapaus muokata ja laajentaa pelimoottorin ominaisuuksia. [3.]

Kuva 1. Unreal Enginen käyttöliittymä.

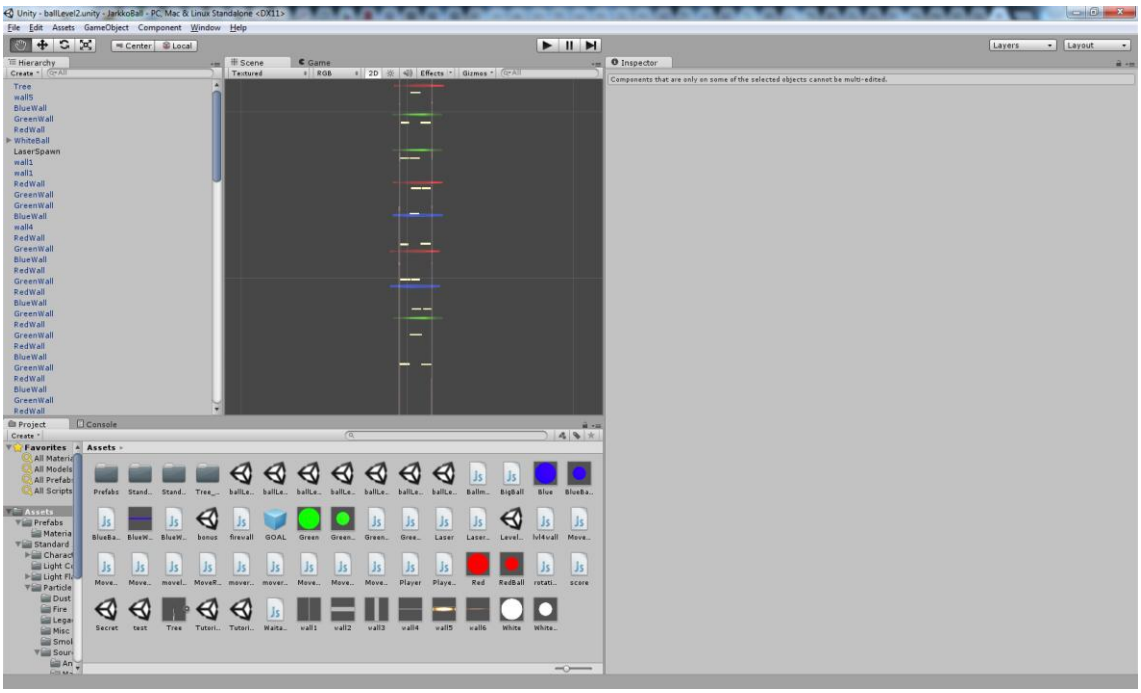
Tunnettuja Unreal Engine 4:llä tehtyjä pelejä ovat mm. ARK: Survival Evolved ja Street Fighter V. Vielä ei ole montaa erityisen suurta menestyksekästä Unreal Engine 4-julkaisuja, mutta aiemmalla versiolla on toteutettu mm. Batman: Arkham City, Bioshock Infinite, Mass Effect ja Borderlands 2. [5.]

Unreal Engine 4:ssä käytetään pelien ohjelmointiin C++-ohjelmointikieltä tai sinikopioita. Sinikopio on visuaalinen, solmupohjainen (node-based) käyttöliittymä, jolla voi käsitellä pelimoottorin toimintoja editorin sisältä. Se on erittäin joustava ja voimakas, koska suunnittelijat pystyvät käyttämään käytännössä lähes kaikkia konsepteja ja työkaluja, jotka ovat yleisesti lähinnä vain ohjelmoijille avoimia. Sitä on helpompi lähestyä, vaikka ei olisi paljoa aikaisempaa ohjelmointitaustaa. [6.]

Iso syy Unreal Enginen valintaan opinnäytetyön kehitysympäristöksi oli uteliaisuus nähdä ja kokeilla, mitä kaikkea sinikopio-järjestelmällä saa tehtyä ja ottaa selvää saanko omin voimin tehtyä sen avulla mielenkiintoisen toimivan pelin.

2.1.2 Unity

Unity on Unity Technologiesin ylläpitämä monialustainen pelimoottori. Sillä onnistuu hyvin 2D- ja 3D-pelien kehittäminen konsoleille, mobiililaitteille, PC:lle, selaimille, VR- ja TV-alustoille. Pelejä voi tehdä vaivattomasti ja helposti Unityllä ja tehdä niistä erittäin optimoituja ja kauniita. Käytettyjä ohjelmointikieliä ovat C# ja Javascript. [7.]



Kuva 2. Unityn käyttöliittymä.

Menestyksekkäitä pelejä, joita on tehty käyttäen Unityä, ovat muun muassa Hearthstone: Heroes of Warcraft, Wasteland 2 ja Temple Run Trilogy. [8.]

Unityllä tehtyjen pelien latauksia oli 5 biljoonaa 2016. Ympäri maailmaa on 770 miljoonaa pelaajaa, jotka pelaavat Unityllä tehtyjä pelejä. 1000 parhaan mobiilipelin joukosta 34 % on tehty Unityllä. [9.]

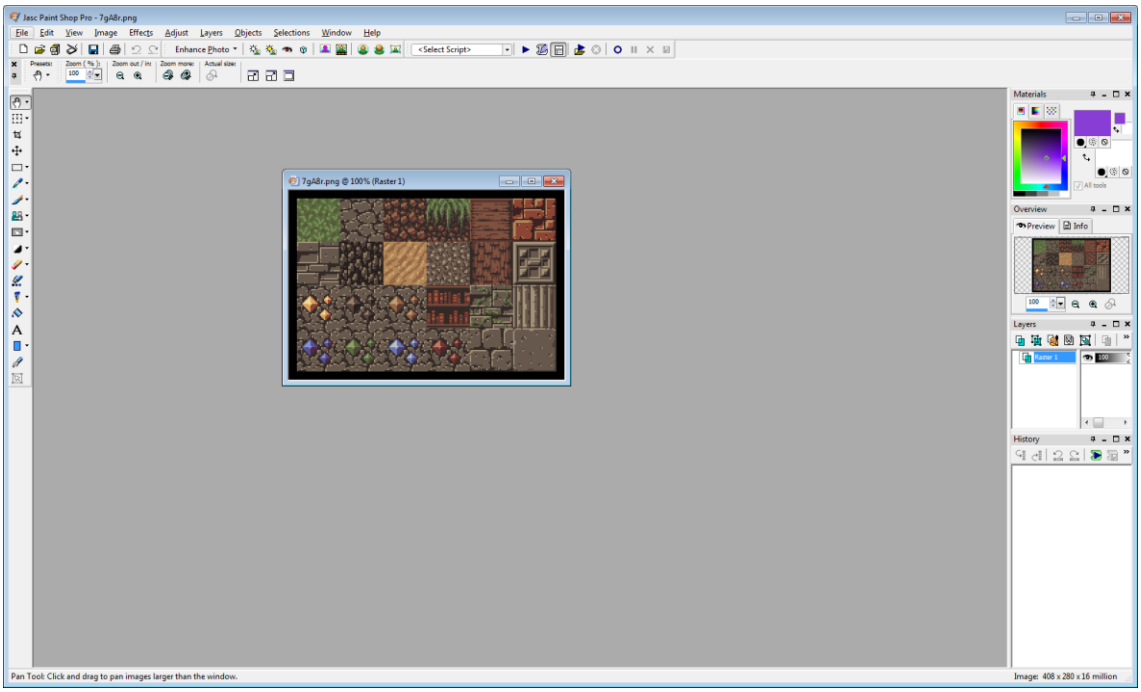
Ennen opinnäytetyön tekemistä olen tehnyt peliprojektin Unityllä, mutta päätin tällä kertaa kokeilla ottaa uuden jännittävän haasteen vastaan sen sijaan, että tyytyisin turvalliseen vanhaan tuttuun vaihtoehtoon.

2.2 Assettien tekemiseen käytettävät apuvälineet

Pelimoottorin lisäksi peliin tarvitsee assetteja, joita ovat muun muassa tekstuurit, 3D-mallit ja äänitehosteet. Assettien tekemiseen löytyy paljon erilaisia ohjelmia, ilmaisia ja maksullisia. En kuitenkaan itse alkanut käyttää erityisen montaa ohjelmaa tähän tarkoitukseen.

2.2.1 Paint Shop Pro 9

Paint Shop Pro on hyvin toimiva kuvankäsittelyohjelma Windows-käyttöjärjestelmässä. Ohjelmalla voi tuottaa ja piirtää erilaisia kuvatehosteita ja muokata kuvia. Käytin ohjelmaa lähinnä joidenkin simppeleiden kuvien piirtämiseksi sekä kuvista ylimääräisen taustan poistamiseen. [10.]



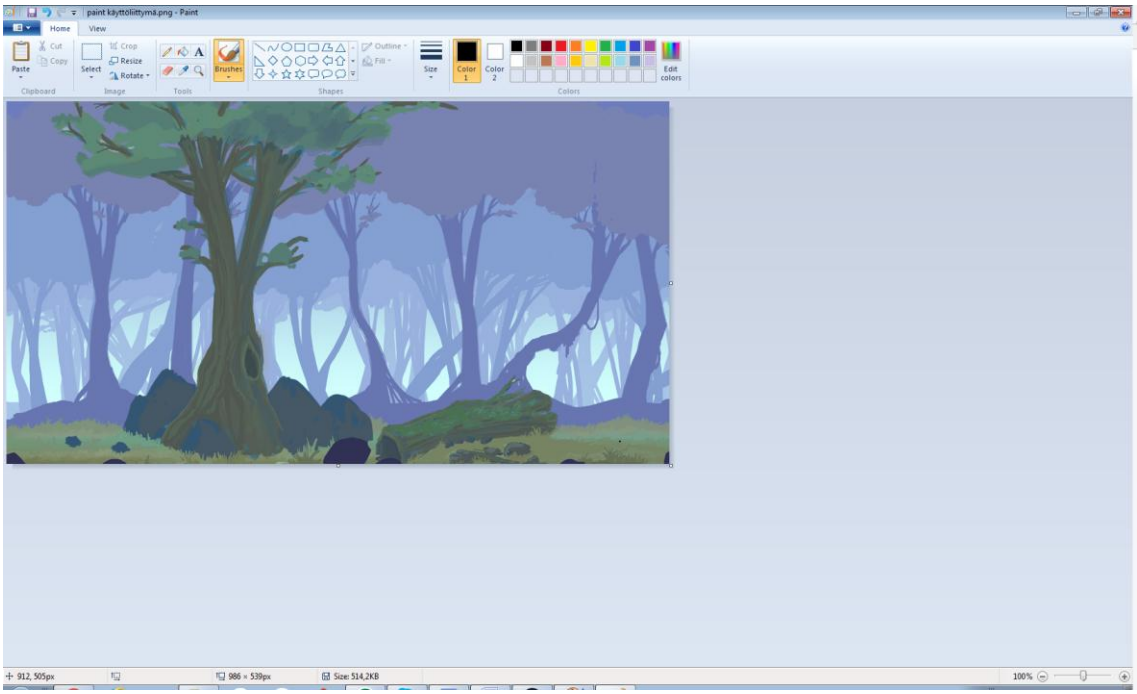
Kuva 3. Paint Shop Pro 9:n käyttöliittymä.

En ole tutustunut kovin syvällisesti kyseiseen ohjelmaan enkä myöskään käyttänyt kuin murto-osan sen potentiaalista.

2.2.2 Microsoft Paint

Paint on kuvankäsittelyohjelma, joka tulee Windowsin mukana. Se tukee BMP-, JPEG-, GIF-, PNG- ja TIFF-kuvaformaatteja. Paintin uusi versio otettiin käyttöön Windows 95:ssa, jolloin nimi pelkistettiin Paintiksi. Paintti on tullut myös Windows XP, Windows Vista ja Windows 7 mukana. [11.]

Ohjelma on melko kevyt ja helppo. Olen tehnyt sillä nopeita pieniä kuvanmuokkauksia tarvittaessa.

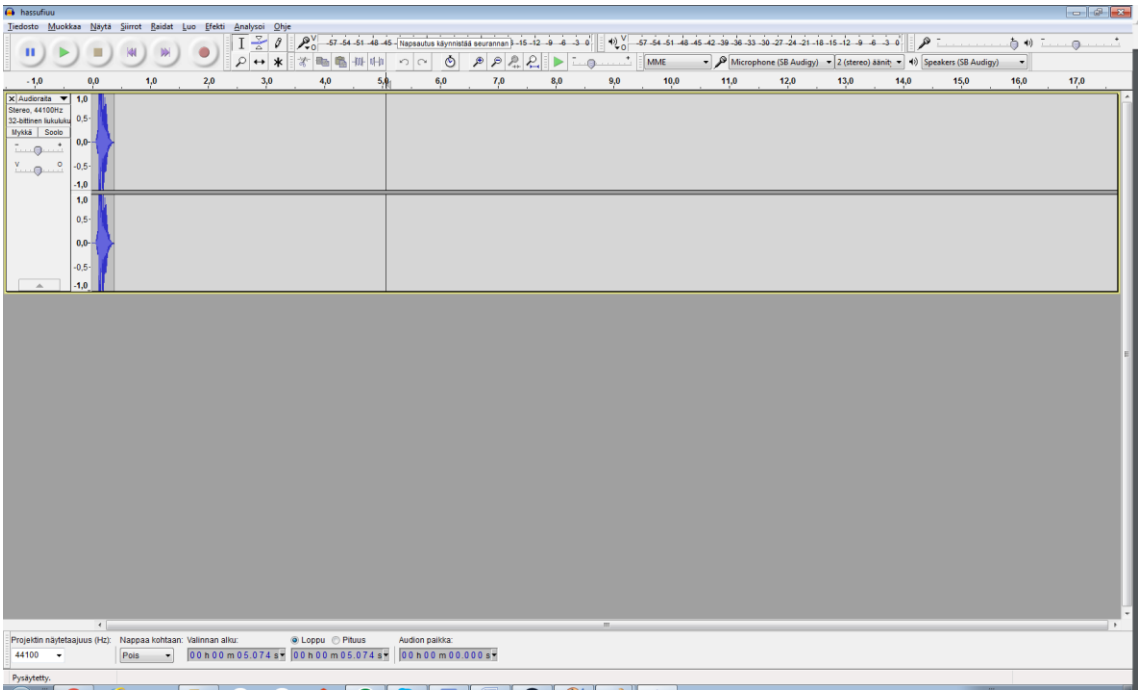


Kuva 4. Paint-käyttöliittymä.

2.2.3 Audacity

Audacity on ilmainen ääniohjelma monen kentän ääninauhoitukseen ja äänen muokkaukseen. Audacity tarjoaa myös avoimen lähdekoodin. Se toimii Windows'illa, Macilla ja Linuxilla. Ohjelman keveydestä huolimatta, sillä pystyy käsittelemään äänitehosteita erittäin monipuolisesti erilaisten efektien avulla ja lopuksi muuntaa tehoste käytetyn pelimoottorin ymmärtämään muotoon. [12.]

Nauhoitin Audacityllä omia äänitehosteita peliäni varten.



Kuva 5. Audacityn käyttöliittymä.

2.3 Ulkopuolisten assettien käyttöönotto

Assettien luominen voi usein olla hyvinkin aikaa vievää. Sen takia niitä ei kannata alkaa välttämättä tekemään ihan kaikkia täysin itse. Työskentelyprosessia voi nopeuttaa ja suoraviivaistaa hakemalla esimerkiksi tekstuurit niitä tarjoavista palveluista. Tarjoavia palveluita ovat muun muassa Textures.com ja Pixabay. Monilla pelimoottoreilla on myös omat ohjelmistoon sisäänrakennetut asset-kaupat. Niitä voi myös yleisesti hakea Googlesta.

Kun harkitsee käyttävänsä valmiita assetteja, kannattaa huomioida, miten paljon se säästää aikaa, miten laadukkaita ne ovat ja miten yleisesti niitä on käytetty. Laatu saattaa vaihdella laidasta laitaan ilmaisissa asseteissa ja voi mennä paljonkin aikaa saada ne toimintakuntoon. Muut pelintekijät ja osa pelaajista kuluttavat sen verran paljon aikaa pelien parissa, että he saattavat tunnistaa tuttuja assetteja muista peleistä, ja tämä saattaa rikkoa pelikokemusta. Kuitenkin prototyyppiä tehdessä assettien yleisyydellä ei ole suurta merkitystä.

Unreal Enginen asset-kauppa on nimeltään Marketplace. Tarjolla on iso määrä grafiikkaa, koodia, äänitehosteita ja malliprojekteja. Pelimoottorin kehittäjä on myös tuottanut osan kaupan sisällöstä, ja se on äärimmäisen korkealaatuista. Muutkin kaupan tarjoamat assetit ovat tasokkaita. Huono puoli on se, että ilmaista materiaalia Marketplacessa on melko vähän.

Unityn Asset Storessa on assetteja pullolla, mutta laatu saattaa vaihdella suuresti. Myös Unityssä on integroitu pelikehitysympäristöön, sillä assettien hankkiminen ja projektiin lisääminen on todella helppoa.

Hain itse jonkin verran tekstuureja ja ääniä. Halusin keskittyä olennaiseen ja edetä projektissani tehokkaasti, joten en kokenut tarpeelliseksi käyttää montaa tuntia omien tekstuurien luomiseen, koska olen huomannut, että aika on liiankin tärkeä resurssi eikä sitä ikinä riitä kaikkeen haluamaan ja joutuu priorisoimaan asioissa karsimalla pois, mitä pystyy.

3 Kenttäsuunnittelun teoriaa

Monissa peleissä navigointia helpotetaan käyttöliittymän avulla pelaajan pelimaailmassa. Peleissä voi olla esimerkiksi jonkunlainen tehtäväkompassi, mikä osoittaa aina paikkaa, jonne pitää seuraavaksi mennä. Joissain peleissä on käytössä myös nuoli, jota voi seurata päästäkseen perille. Autopeleissä tulee ruutuun yleensä isolla teksti, josta ilmenee, että menee väärään suuntaan. Pelimaailmasta olisi kuitenkin fiksua muokata sellainen, mihin pystyy uppoutumaan. Siihen auttaa, jos pelaajan on helppo kulkea ilman keinotekoista apua. Pelaajaa voi opastaa monin keinoin mm. liikkeellä, valolla, ympäristön muodoilla, väreillä, äänitehosteilla ja kiintopisteillä. [13.]

Valoisat paikat tuntuvat usein turvallisilta. Hiiviskelypeleissä ne taas saattavat tuntua vaarallisilta paikoilta, koska on silloin huomion keskipiste, jota ei halua. Valo helposti kiinnittää katseen. Liike on myös hyvä huomionherättäjä, peleissä saattaa olla esimerkiksi hahmo, josta näkee vilauksen ja se saa haluamaan jahdata hahmoa ja pysyä perässä. Ääni on erityisen kiinnostava kun sen lähdettä ei voi nähdä. Esimerkiksi kauhupeleissä kuuluu karmivia ääniä, jolloin tietää, että pahaa on luvassa. Liikennettä ohjastavat nuolet ovat erittäin tehokkaita, mutta se ei välttämättä tee pelitunnelmasta yhtä hienoa, jos vain seurataan nuolta.

Pelin sisällä olevat erilaiset pelimekaniikat vaihtelevat, mutta on myös paljon yleisiä pelimekaniikoita, jotka ovat monilla peleillä yhteisiä. Erilaiset pelimekaniikat kuitenkin haastavat pelaajan ajattelua ja ongelmanratkaisukykyjä. Hiiviskelypelissä etsii piilopaikkoja ja kiertää valoisia alueita ja yrittää keksiä reittejä, joista pääsee vartioiden ohi. Usein myös pitää kurkata ja seurata, missä viholliset liikkuvat ja tehdä nopeita liikkeitä hyvällä ajoituksella. Ammuskelupeleissä yrittää löytää hyviä aseita ja panoksia sekä kykenee sijoittautumaan sillä tavalla, että vihollinen ei pysty helposti yllättämään. Se voi olla iso virhe mennä sijaintiin, jossa pitäisi esimerkiksi pitää silmällä 4 eri suuntaa samaa aikaa. Se ei tule käytännössä onnistumaan kovin tehokkaasti. Halutaan olla se, joka pyrkii yllättämään vihollisen ja välttämään sen, että ollaan yllätetty sen sijaan.

Väreillä voidaan usein selventää, mitkä kohdat ovat turvallisia ja mitkä eivät. Niillä voi tehdä vaikkapa kiipeilykohdista selviä, mistä pystyy kiipeämään ja mistä ei. Peleissä voi myös välillä olla mekaniikka, jolla voi vaihtaa oman hahmon väriä ja sen avulla pyrkii vaihtamaan omaa väriä samanlaiseksi kuin vihollisella. Esimerkiksi jos vihreä pallo olisi kohta osumassa hahmoon, niin vaihdetaan oman hahmon väri vihreäksi torjuaksesi sen. Tein itse pelin, jossa voi vaihtaa oman hahmon värisi vihreäksi, siniseksi ja punaiseksi. Pelissä tuli jatkuvasti vihollisia, jotka olivat vihreitä, sinisiä tai punaisia. Piti aina nopeasti vaihdella omaa väriä sitä mukaan, kun näkee, mikä väri on osumassa hahmoon seuraavaksi. Se saattaa kuulostaa melko helpolta, mutta siinä menee yllättävänkin helposti sekaisin, jos pelin tempo on nopeanpuoleinen.

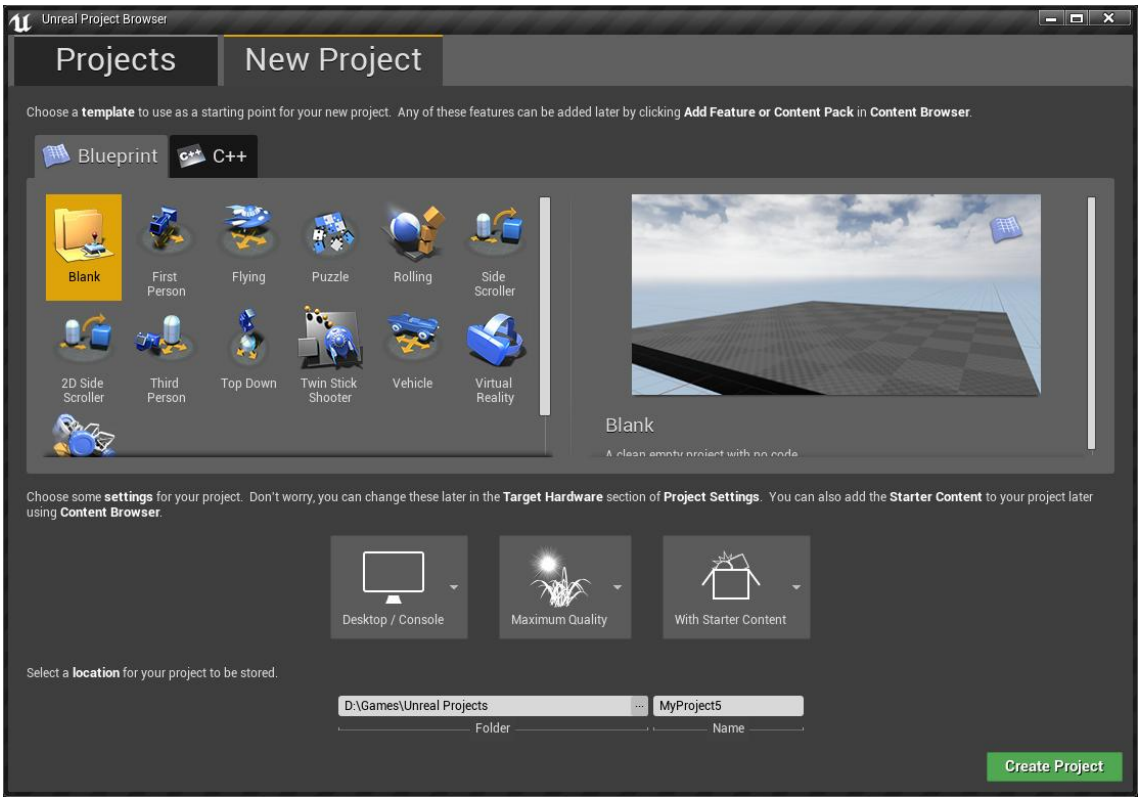
Kiintopiste on usein jotain kaukaisuudessa näkyvää. Siellä voi olla esim. rakennus tai erikoisenmuotoinen maasto. Kiintopiste voi hyvin auttaa pelaajaa suunnistamaan ja pysymään kartalla siitä, minne pitää mennä. Se on yleensä nähtävissä melkein kaikkialta ympäri kaupunkia, jolloin pelaaja onnistuu suunnistamaan sinne päin ilman, että eksyy helposti.

On myös monia pelejä, joissa hädin tuskin on lähes yhtään pelaajaa helpottavia elementtejä navigoimiseen. Minulla itsellä on henkilökohtaisesti melko huono suuntavaisto ja olen jopa onnistunut eksymään pelin sisällä moneen otteeseen. Siihen voi helposti turhaantua, kun ei enää pysy perässä tai keksi, minne pitäisi mennä seuraavaksi. Haahuillaan ympäri aluetta päättömästi ja päädytään jatkuvasti väärään paikkaan. Pahimmassa tapauksessa olen joutunut lopettamaan pelin kesken juuri sen takia, kun turhaudun siihen, että en enää tiedä, minne pitäisi mennä. Ymmärrän kuitenkin sen, että se tuo peliin lisää omanlaista vapautta, mikäli navigoimista ei aleta syöttämään ja pakottamaan liikaa, mutta siinä on aina omat riskinsä, mitä se tuo tullessaan.

4 Toteutus

4.1 Suunnittelu

Pelin suunnittelu alkoi siitä, kun aloin miettimään, mitä vaihtoehtoja minulla on. Ensin aloin pohtimaan kovasti, teenkö opinnäytetyön Unityllä vai Unreal Enginellä. Aloin pikkuhiljaa vähän tutustumaan Unreal Engineen ja asensin pelimoottorin itselleni. En kuitenkaan ollut heti sen jälkeen vielä vakuuttunut, että tekisin projektin Unreal Enginellä. Kävin game jam -tapahtumassa, jossa aloittelijat tekevät pelejä käyttäen Unityä tai Unreal Engineä ja saavat samalla apua kokeneemmilta ohjelmoijilta. Päädyin luomaan nopean yhden illan pelin siellä ja olin vaikuttunut, miten kätevästi sai niin paljon aikaiseksi lyhyessä ajassa, joten innostuin käyttämään sitä enemmänkin. Hetken päästä tein lopullisen päätöksen, että teen jonkunlaisen pelin käyttäen Unreal Engine -pelimoottoria.



Kuva 6. Unreal Enginen uusi projekti.

Seuraava vaihe oli, että aloin miettimään, minkä tyyppisen pelin tekisin. Vaihtoehtoja oli monia, enkä ollut varma, mikä olisi fiksuin vaihtoehto. Kokeilin monia malliprojekteja. Harkitsin First Person-, Puzzle-, Rolling-, Third Person-, Twin Stick Shooter- ja 2D Side-scroller -pelien tekemistä. Aloin arvioimaan, kuinka haastavia kyseiset vaihtoehdot olisivat. Tajusin myös, että minulla ei ylipäätänsä ole kovin hyviä ideoita puzzle-pelin luomista varten, joten sain sen poissuljettua nopeasti. Ajattelin myös, että 3D-pelissä voisi olla ehkä vähän liiankin aikaa vievää toteuttaa haluamallani tavalla. Tavoitteeni oli onnistua tekemään kunnollinen pelattava peli, joten päädyin tulokseen, että 2D Side-scroller -peli voisi olla sopiva realistinen haaste, jossa uskoin voivani onnistua.

2D Side-scroller -pelit ovat usein tasohyppelypelejä. Siinä on kuitenkin paljon mietittävää, minkä tyyppisen pelin siitä haluaa tehdä. On paljon esim. myös pelejä, joissa hahmo menee automaattisesti eteenpäin jatkuvasti ja pelaajan täytyy hyppiä esteiden yli. Halusin kuitenkin tehdä pelin, jossa voi liikkua omalla tahdilla ja pysähtyä myös halutessaan. En oikeastaan miettinyt lopullista suunnitelmaa heti alkuvaiheessa vaan päätin, että alan tekemään vähitellen uusia ominaisuuksia peliin ja aloitin tekemällä yleisiä tasohyppelyyn tarvittavia elementtejä ja mietin lisäominaisuuksia myöhemmin. En ollut varma, kuinka sulavasti ja nopeasti projekti menisi eteenpäin, joten en halunnut tehdä liian kunnianhimoisia tavotteita, koska alkaisi harmittaa, jos puoliakaan suunnitelmista ei tule toteutumaan.

Tiesin kuitenkin, että nauttisin nopeatempoisesta toimintapainotteisesta peliympäristöstä, joten halusin tehdä hahmon, jolla pystyn ainakin hyppimään, väistelemään sekä hyökkäämään. Tarvitsen myös liikkuvia alustoja ja vihollisia, jotta vauhdikkaalle hahmolle saa tekemistä. Tässä oli jo melko hyvät lähtökohdat aloittamaan pelin luominen.

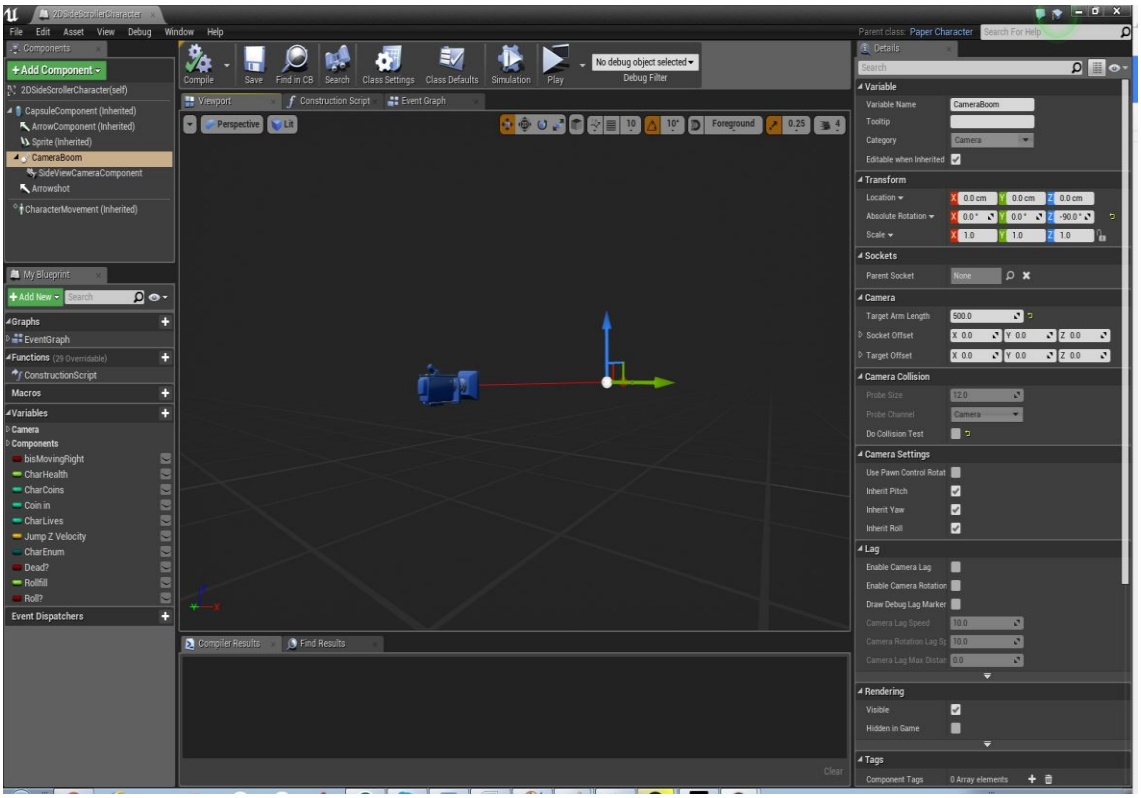
Kuva 7. Kuvakaappaus yhdestä luomani pelini tasosta.

4.2 Pelin toteutuksen vaiheita osa 1

4.2.1 Skenen asettaminen

Aluksi kun on käynnistetty Unreal Engine ja menty 2D Side-scroller -projektin sisään, niin siellä on melko suppea malli valmiina. Aloitin poistamalla ylimääräiset tekstuurit ympäriltä ja jätin lähinnä oman hahmoni. Oletuksena hahmolle on asetettu kyky liikkua, jota voi alkaa muokata haluamalla tavalla. Halusin, että hahmoni kuitenkin pystyy seisomaan jonkinlaisen objektin päällä, joten hain itselleni nopeasti netistä jonkunlaiset väliaikaiset tekstuurit laatasta testausta varten. Kuva ensin tallennetaan koneelle ja sitä kautta siirretään se Unreal Engine -projektin sisälle. Tein myös oman kansion projektin sisällä nimeltä ”Sprites” ja laitan sinne kaikenlaiset tekstuurit. Sen jälkeen kun ollaan lisätty kuva sinne, niin täytyy ekstraktata kuvasta sprite, joka tapahtuu helposti, kun menee oikealla hiirennapilla kuvasta Sprite actions -> Extract sprites ja silloin oletusasetukset ovat tähän tarkoitukseen ihan sopivat, joten painetaan vain extract. Sen jälkeen valitaan sprite ja siirretään se pelimaailmaan ja asetetaan se hahmon alapuolelle, niin hahmon pitäisi automaattisesti seistä sen päällä. Kannattaa kuitenkin olla tarkkana, missä Y-akseli on, koska pelissä merkkaavat vain X- ja Z-suunnat, sillä peli on 2D eikä 3D.

Tässä vaiheessa on hyvä alkaa katsomaan kameran asetuksia. On hyvä varmistaa, että kamera osoittaa oikeaan paikkaan, ja sen keskipiste on halutussa kohdassa ja etäisyys on sopiva. Oletuksena projektin sisällä on ”2DsideScrollerBP”-kansio, jonka sisältä avataan ”Blueprints” ja sen sisällä on ”2DsideScrollerCharacter”, joka pystytään avata.



Kuva 8. Hahmon sisäiset asetukset Unreal Enginessä.

Halusin, että kamera on enemmän zoomattuna lähemmäs. Se onnistuu menemällä SideViewCameraComponentiin ja sisällä oikealla puolella näkyy Camera Settings, jonka alapuolella on Ortho Width. Tämä asetus kuvastaa sitä, miten kaukana kameran näkökulma on. Aloin kokeilemaan eri etäisyyksiä testatakseni, miltä se näyttää. Se ei kuitenkaan tässä vaiheessa ole kovin olennaista löytää täydellistä etäisyyttä, koska se tulee myös riippumaan muista asioista, kuten hahmon koosta ja ympäristöstä. Päädyin laittamaan itselleni sen arvoksi 500, joka on noin 4 kertaa lähempänä kuin oletusarvo.

Kamera voi hyvin osoittaa esim. liian ylhäälle, jolloin kannattaa tarkistaa CameraBoom-asetuksia. Tässä tapauksessa sieltä voi muutella Socket Offsetia ja asettaa arvot 0:ksi. Se ei kuitenkaan ole aina välttämättä juuri, mitä halutaan. Hyvä tapa on myös kokeilla muuttaa arvoja kokeillakseen, mitä siinä tapahtuu.

Tässä vaiheessa olin kuitenkin sitä mieltä, että hahmoni oli liian iso peliäni varten, joten halusin tehdä siitä pienemmän. Tämä onnistuu menemällä hahmon Sprite-asetuksiin ja voidaan käyttää joko pikanäppäintä R tai painaa keskeltä ylhäältä pientä painiketta, jonka kuvauksessa lukee valitse ja skaalaa objekti. Nyt voidaan valita keskeltä niin, että kaikki suunta-akselit ovat samalla valittuna, jolloin voi helposti muokata hahmoa heiluttelemalla hiirtä eri suuntiin ja siten hahmottamaan, mikä koko tuntuu hyvältä.

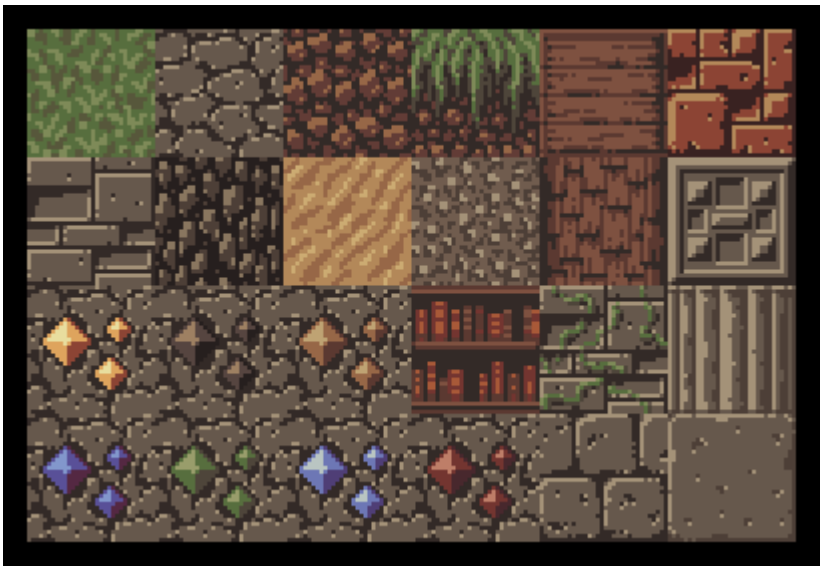
Nyt jos aloitetaan tässä vaiheessa peli, niin huomataan, että hahmo on huomattavasti aiemmin laitetun laatan yläpuolella, mikä johtuu siitä, että CapsuleComponent törmäys-asetukset pitää myös korjata täsmäämään hahmon spriten kanssa. Voidaan mennä kyseisiin asetuksiin ja valita kapseli ja yrittää asettaa se suurin piirtein täsmäämään hahmosi kokoa ja levittää se hahmon ympärille samankokoiseksi.

On myös hyvä olla tarkkana, että Y-akselin arvo on 0, koska mikäli esimerkiksi hahmo olisi -100 sijainnissa ja laatta 0 sijainnissa, niin saattaisi näyttää, että ollaan samassa paikassa, mutta todellisuudessa ollaan eri syvyystasolla.

4.2.2 Tekstuurien lisääminen

Peliin tarvitsee tekstuureja, jotta saadaan lisää maisemia, kävelyalustoja, hahmoja ja mitä ikinä tarpeen tuleekaan. Yksi vaihtoehto on, että tehdään täysin omia tekstuureja esimerkiksi Paint Shop Pro 9 -ohjelman avulla. En kuitenkaan itse kokenut, että haluaisin käyttää tähän liian paljoa aikaa, koska se oli korkeampi prioriteetti suoraviivaisemmin kehittää peliä eteenpäin.

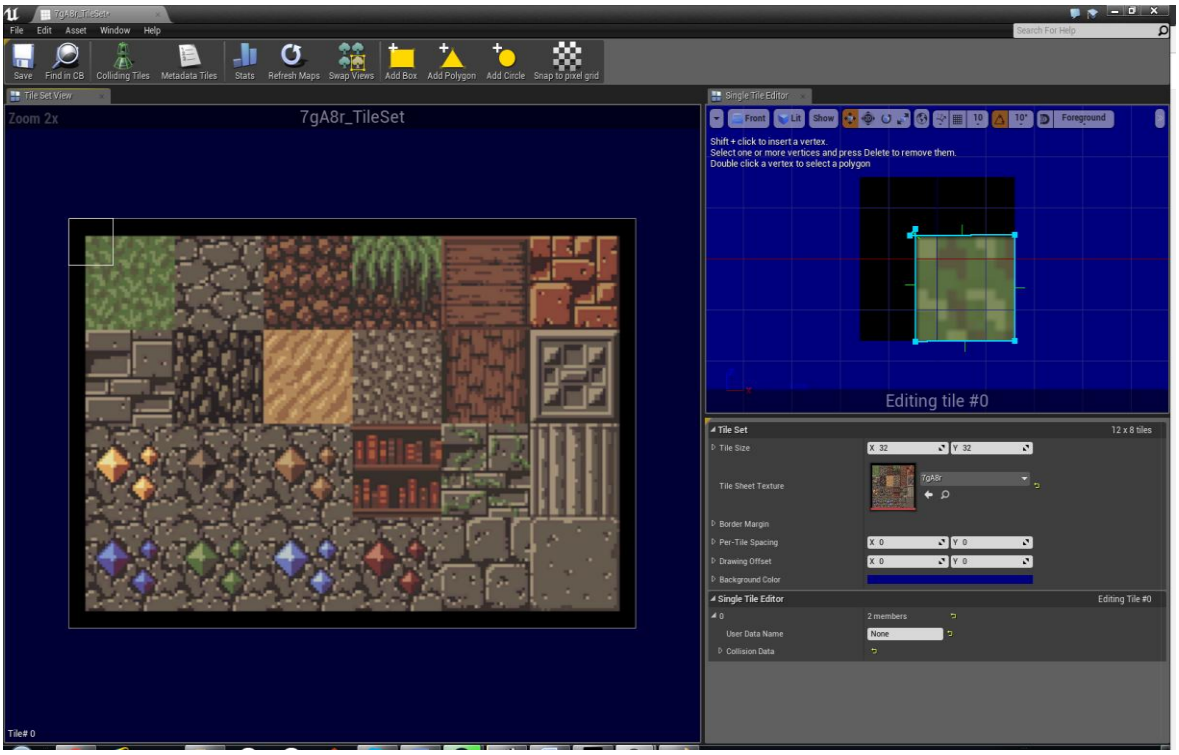
Päädyin tulokseen, että haen ilmaisia tekstuureja netistä ja lisään niitä peliini. Löysin paljonkin sellaisia grafiikoita, joihin olin tyytyväinen. Hain yksittäisiä kuvia ja sen lisäksi myös valmiita spritesheetejä.



Kuva 9. Yksi käyttämäni spritesheet.

4.2.3 Tiilisarjan lisääminen

Aiemmin lisäsin projektiini spritesheetin hyvällä syyllä, koska sille tulee heti tarvetta. Painetaan lisäämää spritesheetiä projektin sisällä oikealla hiirennapilla ja mennään Sprite actions -> Create Tile Set. Nyt pääsee alkuun. Se ei kuitenkaan ole ihan niin helppo, että olisi vielä täysin valmis. Tällä hetkellä ne ovat pelkästään tekstuureja, mutta ideana on saada ne reagoimaan törmäyksiin ja tekemään mahdolliseksi esimerkiksi seistä niiden päällä. Se on melko helppoa lisätä tiilille törmäyskohdat, mutta siinä on kuitenkin hiukan tekemistä.



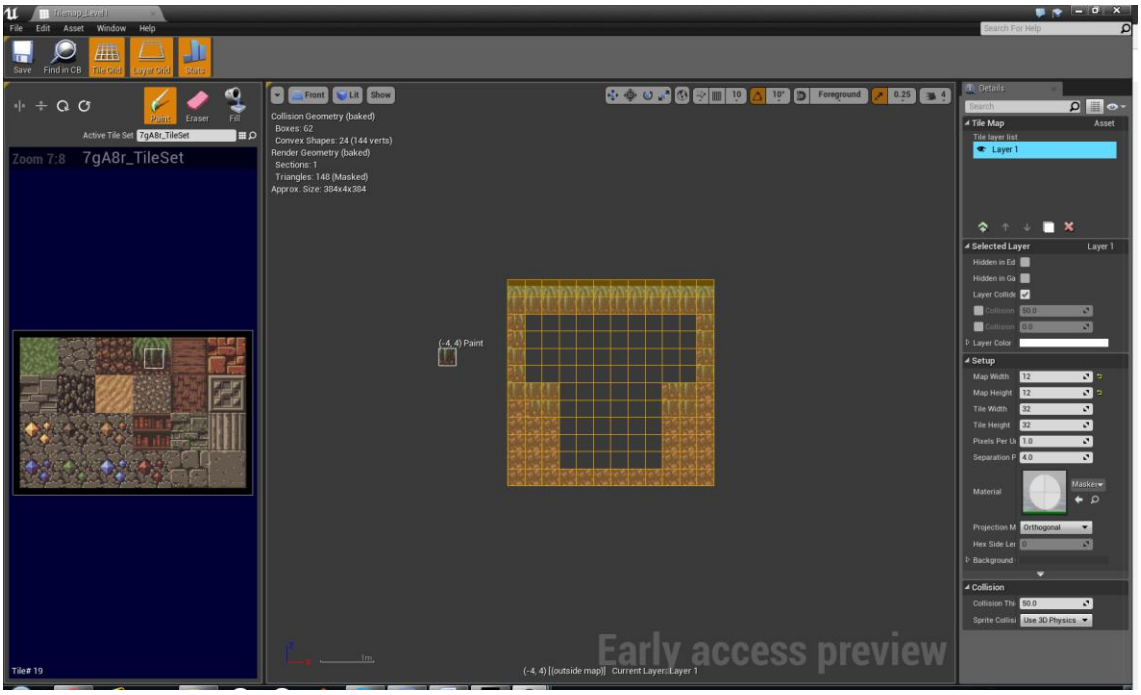
Kuva 10. Tiilisetti Unreal Engine -näkymässä.

Kuvassa 10 on tiilisetin näkymä Unreal Enginessä. Tässä tiilisetissä on 12x8 määrä eri tiiltä, eli yhteensä 96, joka nähdään myös oikealla Tile Set -kohdassa. Tämä tarkoittaa sitä, että mikäli halutaan kaikkien tiilien olevan törmäyskelpoisia, niin niille täytyy lisätä se yksitellen.

Aluksi valitaan spritesheet-kuvasta kohta, jonka seurauksena valittua aluetta ympäröi pieni valkoinen neliö. Tämä siis vastaa yhtä tiiltä. Nähdään oikealla ylhäällä, miltä valittu tiili näyttää. Oletuksena siinä tosin ei näy vaaleansinistä neliörajausta alueen ympärillä. Tämän voi lisätä ylhäältä painamalla Add Box -painiketta. Tässä täytyy kuitenkin olla tarkkana, koska kun painetaan kyseistä painiketta, niin se täyttää koko neliön alueen, eli myös mustan osion. Sillä oletuksella, että halutaan mustan alueen olevan ”tyhjää”, niin täytyy klikata ohutta rajausneliötä ja siirtää sitä lähelle ilmestyvien nuolien avulla täsmäämään haluttua törmäysaluetta. Keskimmäiset tiilet kuitenkin onnistuvat helpommin ja ne täsmäävät visuaalista törmäysaluetta oletuksena pelkästään sillä, että painetaan Add Box -painiketta. Mikäli on vaikean muotoisia tiiliä, niin pystytään rajata aluetta tarkemmin pitämällä shift-näppäintä pohjassa valitsemisen aikana, sen avulla saadaan rajausneliölle nystyrät, joita voidaan taivutella vapaammin.

4.2.4 Tiilikartan luominen

Nyt on aika luoda tiilikartta. Tiilikartan luomiseen käytetään aiemmin luomaa tiilisettiä. Alkuun pääsee helposti menemällä projektiin ja etsimään sieltä tehty tiilisetti, jota painetaan oikealla hiirennapilla ja valitaan create tile map.



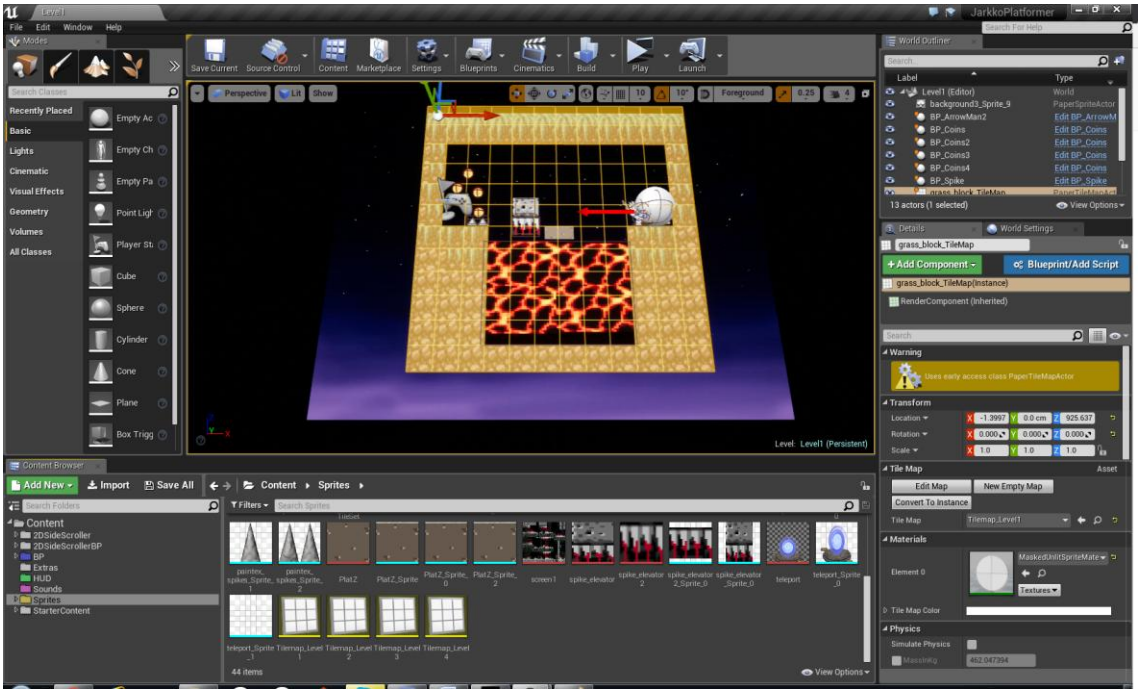
Kuva 11. Tiilikartta Unreal Engine -projektin sisällä.

Kuvassa 11 on visuaalinen näkymä tiilikartasta ja sen asetuksista. Tiilikartta ei ole oletuksena näin iso, mutta sitä pystytään muokata helposti. Alussa on oletuksena 4x4 tyhjää ruutua, eli yhteensä 16. Pystytään oikealta puolelta asetuksia suurentaa tai pienentää alueen määrää. Kyseisessä kuvassa on käytössä 12x12 kartan kokona, eli se koostuu yhteensä 144 ruudusta.

Tiilikartassa on valittuna tiilisetti, jonka sain luotua projektiin aiemmin. Pystytään halutessa vaihtaa aktiivista tiilisettiä toiseen, mikäli se on tarpeellista. Tässä tapauksessa minulle kuitenkin riittää, että käytän vain yhtä tiilisettiä tiilikartan tekemiseen.

Vasemmalla nähdään tiilisetti ja sen sisällä voidaan valita haluttu ruutu sen sisältä, jota pieni valkoinen neliö kuvastaa. Kun ollaan valittu kohta tiilisetistä, niin ilmestyy tiilikartan lähelle pieni valkoinen neliö, jonka sisällä on kuva valitusta kohdasta tiilisetissä. Pystytään helposti painaa tiilikartassa tyhjää ruutua, niin se piirtää tyhjän ruudun päälle valitun ruudun. Pystytään nopeasti vaihdella valittua ruutua ja käyttää sitä kuin se olisi väritystyökalu. Mikäli tehdään vahingossa virheitä, niin pystytään aina valita Eraser-työkalu ja poistaa sillä halutut ruudut muuttamalla ne tyhjäksi.

Tiilikartan tekeminen ei oikeastaan ole pakollista pelin tekemisen kannalta, mutta se antaa mukavamman helpomman tavan lisätä peliin uusia tasoja. Kun ollaan tyytyväisiä tehtyyn tiilikarttaan, niin pystytään halutessa poistumaan tiilikarttanäkymästä menemällä projektin päänäkymään ja valitsemaan sieltä tehdyn tiilikartan ja voidaan raahata ja pudottaa se halutulle alueelle.



Kuva 12. Tiilikartta valittuna projektini päänäkymässä.

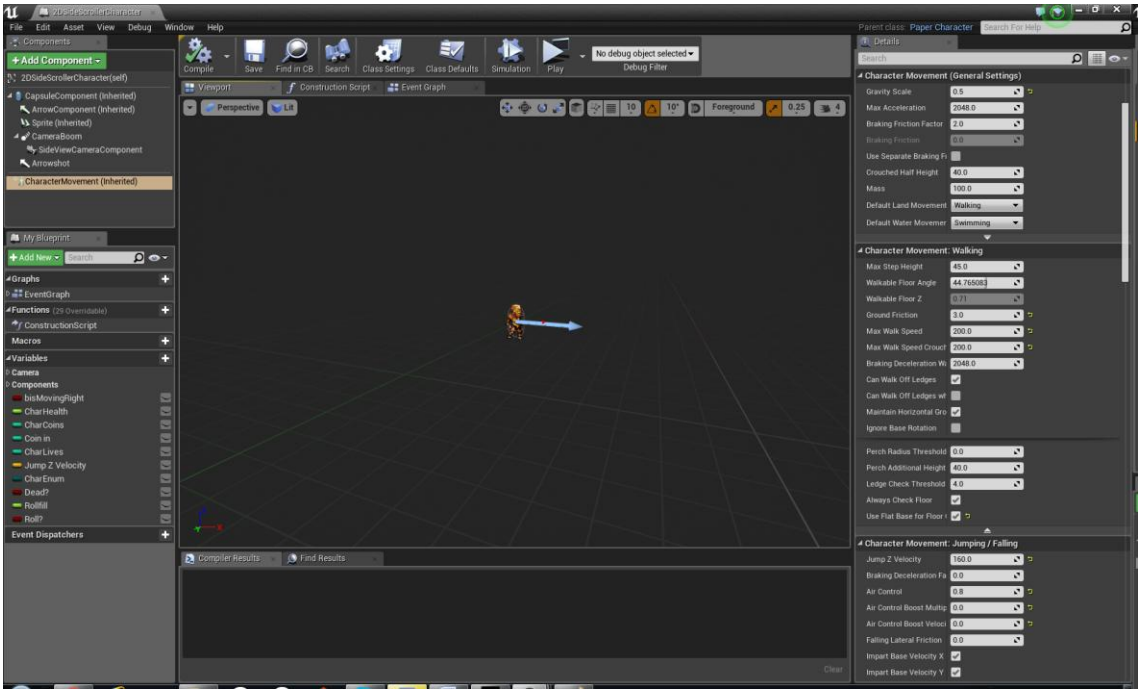
Kuvassa 12 olen lisännyt tiilikartan, ja maasto on näin käytössä pelin sisällä.

4.2.5 Hahmon ominaisuuksien muuttaminen

Oletuksena projektin sisällä on ”2DsideScrollerBP”-kansio, jonka sisältä voidaan avata ”Blueprints”-kansio ja sen sisällä on ”2DsideScrollerCharacter”, joka pystytään avata, eli samalla tavalla kuin aiemminkin kamera-asetuksia muuttaessa. Viimeksi täältä muutettiin hahmon kamera-asetuksia ja kokoa, mutta tällä kertaa voidaan katsoa myös hahmon liikkumisasetuksia.

ArrowComponent on yksi asetus, joka saattaa olla hyödyllistä katsoa läpi, jos haluaa lisätä hahmolle jonkinlaisen ampumiskyvyn. Nuolta voi siirtää, jolloin hahmon oletusampumissuunta muuttuu.

Nyt kuitenkin katsotaan vähän hahmon liikkumisasetuksia (CharacterMovement).



Kuva 13. Hahmon liikkumisasetukset.

Oikealla pystytään muuttaa kaikenlaisia asetuksia. Kokonaisuudessa niitä on hyvinkin paljon ja sitä pystyy myös vähän halutessaan kokeilla, mitä ne tekevät. Omassa projektissani kuitenkin haluan muuttaa vain muutamaa arvoa.

Tällä hetkellä oletuksena hahmo on äärimmäisen nopea, koska hahmon kokoa muutettiin aiemmin, joten suhteutettuna se vaikuttaa paljon nopeammalta kokoonsa nähden. Kamera-asetuksista zoomattiin noin 4 kertaa lähemmäksi pelinäkymää, joten hahmo myös luonnollisesti kulkee ruudun päästä päähän paljon nopeammin. Voidaan etsiä kohta, jossa lukee ”MaxWalkSpeed” ja etsiä sopiva arvo, joka tuntuu hyvältä. Se määrittää sinun kävelemisnopeuden. Projektissani se on hyvin olennainen asia, koska lähes kaikki liike lasketaan kävelemiseksi, enkä ole mahdollistanut esimerkiksi uimista tai lentämistä. Tämän arvo on oletuksena 600 ja itse vaihdoin sen 200:ksi, eli 3 kertaa hitaammaksi. Se tuntui hyvältä ja sopivalta mielestäni.

Mikäli halutaan hahmolla olevan hyppyominaisuus, niin voi olla, että halutaan muuttaa ”Jump Z velocity” -arvoa, joka määrittää sen, miten voimakas hyppy on, mitä voimakkaampi, niin sitä enemmän ponnistetaan korkeammalle. Tämä arvo on kuitenkin myös suhteutettu omaan kokoon, joten vaikka kahdella hahmolla olisi sama arvo hyppyponnistuksessa, niin ne eivät välttämättä ponnahda yhtä korkealla, mikäli niiden massat eroavat toisistaan.

Saatetaan myös haluta muuttaa ”Gravity Scale” -asetusta, joka muuttaa painovoimaa. Oma syyni muuttaa tätä oli se, että halusin hahmoni olevan ilmassa kauemmin ja putoavan hitaammin hypätessä. Oletusarvona ollaan ilmassa vain pienen hetken, joka ei mielestäni sopinut pelini tyyliin yhtä hyvin kuin hieman heikompi painovoima. Muutin tämän asetuksen arvon 0,5:een 1,0:sta. Tämä tarkoittaa sitä, että painovoiman voimakkuus on puolillaan.

Hahmon hyppyasetuksissa näet myös arvoja kuin ”Air Control”, joka määrittää sen, kuinka hyvin pystytään kontrolloida hahmoa ilmalennon aikana. Arvot menevät 0:sta 1:een siten, että 0 tarkoittaa ei lainkaan kontrollia ja 1 täysi kontrolli. Itse käytän hieman siitä välistä, mutta kuitenkin enemmän kallistuen täyteen kontrolliin arvolla 0,8.

4.2.6 Liikkuvan alustan luominen sinikopioita käyttäen

Nyt pääsimme jännittävään osioon, sillä nyt alamme tutustumaan enemmän sinikopioihin, jotka ovat erittäin tärkeä osa Unreal Engineä. Tässä vaiheessa voi olla ihan hyvä tehdä uusi kansio ja nimetä se vaikka ”BP”:ksi. Tähän kansioon voidaan laittaa uudet projektissa tarvittavat komponentit, joihin käytetään sinikopioita.

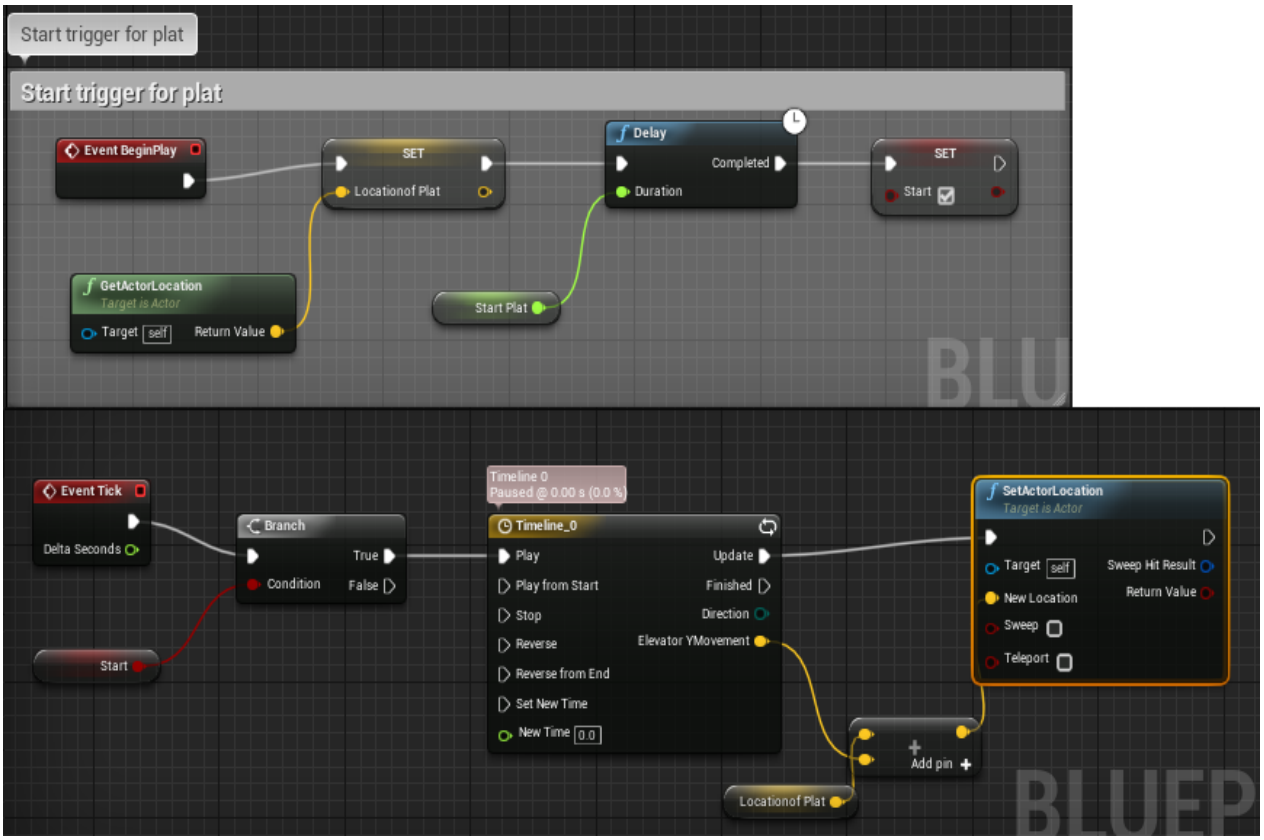
Päänäkymässä voidaan avata luoma kansio ja luoda sinne uusi sinikopio luokka, jonka vanhemman luokaksi valitaan toimija. Blueprint class -> Pick a parent class -> Actor.

”Actor” eli toimija on objekti, joka voidaan sijoittaa tai synnyttää pelimaailman sisällä. Tämä on oman projektini kannalta tärkein ja yleisin vaihtoehto. Kun ollaan luotu toimija ja menty sen sisäisiin asetuksiin, niin on aika alkaa miettiä, mitä tarkkaan ottaen halutaan sen tekevän.

Tavoitteena on tehdä liikkuva alusta, joten luonnollisesti halutaan, että sen päälle voi astua ja sen voi nähdä. Oletuksena se ei tee vielä mitään, eikä näkyisi pelaajalle visuaalisesti, vaikka se lisättäisiin pelimaailmaan. Vasemmalla on kohta, jossa näkyy ”Add Component”, painetaan sitä ja kirjoitetaan ”paper sprite” -hakukenttään. Tämän jälkeen voit valita sen ja lisätä sille spriten, joka pystytään tehdä valitsemalla oikealta Source Sprite ja valitsemalla sieltä halutut. Spritenä voi olla lähes mikä vain tekstuuri, se ei juurikaan vaikuta sen toimintoihin ollenkaan, ainoastaan miltä halutaan sen näyttävän.

Kuva 14. Esimerkkikuvia projektini liikkuvista alustoista.

Unreal Enginessä on objekteja valitessa 3 erilaista näkymää, jotka ovat Viewport, Construction Script ja Event Graph. Tähän asti on tullut käytettyä vain Viewport-osaa. Nyt tutustumme Event Graph -osioon, jonka sisällä luodaan sinikopiot. Niillä määritetään logiikka, jolla halutaan asioiden toimivan. Ne ovat vaihtoehtoinen tapa koodille tehdä asioita.

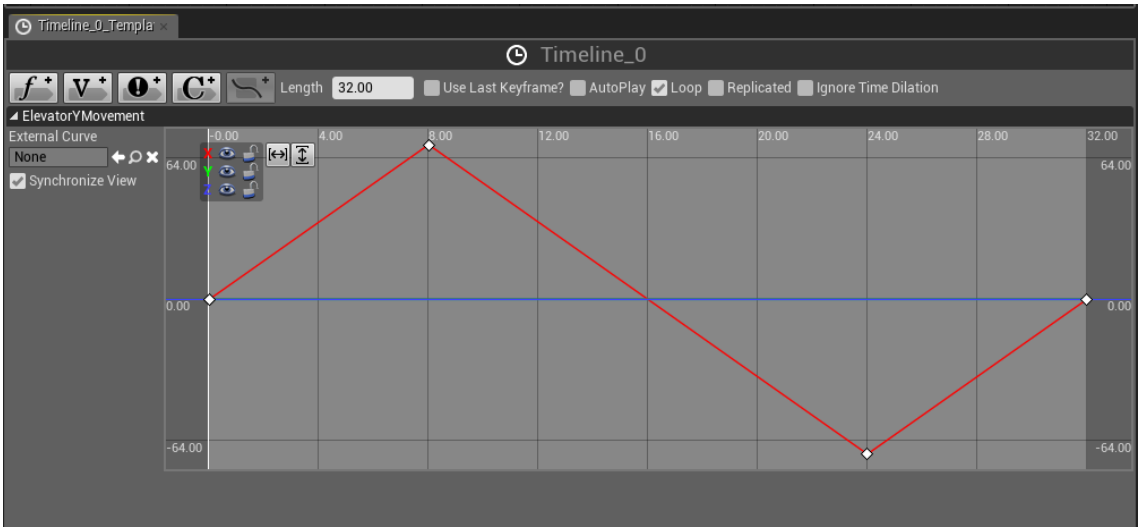


Kuva 15. Liikkuvan alustan sinikopiot.

Kuvassa 15 on lopputulos siitä, miltä sinikopiot näyttävät, kun ne on saatu toimimaan. Tämä on yksi monista tavoista tehdä asia, eikä yhtä ainoaa oikeaa tapaa ole tämän tai muidenkaan sinikopioiden tekemisessä.

Aloitetaan ylemmästä osiosta, jonka ympärille olen kommentoinut ”Start trigger for plat”. Aluksi laitetaan ”Event BeginPlay”, jossa määritetään, mitä halutaan tapahtuvan, kun tämä toimija aktivoituu. Ensin halutaan tehdä uusi vektorimuuttuja, jonka tarkoituksena on asettaa sijainti täsmäämään alustan sijaintia. Nimeksi voidaan laittaa periaatteessa mitä vain, mutta voi olla fiksua antaa sen nimeksi jotain, mikä kuvastaa hyvin sen tarkoitusta, esimerkiksi ”LocationOfPlat”. Halutaan seuraavaksi yhdistää muuttujan palautusarvoksi ”GetActorLocation”, joka hakee muuttujalle alustan sijainnin. Sen jälkeen voidaan halutessa määritellä viiveen, kuinka kauan kestää, että alusta aktivoituu lisäämällä viiveen. Lopuksi voidaan tehdä uuden totuusarvomuuttujan, joka asetetaan todeksi viiveen jälkeen. Sen nimeksi voi antaa esim. ”Start”.

Alemmassa osiossa aloitetaan lisäämällä ”Event Tick”, jonka tarkoitus on kutsua itseä yhtä monta kertaa sekunnissa, kuin pelin kuvataajuus, eli käytännössä päivittää ja tarkistaa tilanteen jatkuvasti monia kertoja sekunnissa uusien muutosten varalta ja toimii heti niiden tapahtuessa. Seuraavaksi lisätään ”Branch”, jonka ehdoksi asetetaan aiemmin lisätty totuusarvomuuttuja ”Start”. Käytännössä tämä siis tarkistaa koko ajan, onko ”Start”-muuttujan arvo totta vai ei. Mikäli se ei ole totta, niin ei tehdä mitään ja tarkastetaan se uudelleen loputtomia kertoja. Mikäli se on totta, niin lisätään aikajana.



Kuva 16. Aikajana sinikopion sisällä.

Aikajanan avulla pystytään määrittelemään miten ison matkan liikkuva alusta liikkuu mihinkin suuntaan annetussa ajassa. Aluksi tarvitsee miettiä, miten haluaa alustan liikkuvan. Tärkeä lähtökohta on liikkumissuunta, haluaako liikkuvan alustan menevän sivuttain oikealle ja vasemmalle vai hissin tavoin ylös ja alas. Olen tehnyt peliini molempia ja erilailla toimivia liikkuvia alustoja. Se on todella helppoa sen jälkeen kun ollaan saatu ensimmäisen toimimaan ja samalla periaatteella pystytään tehdä toinenkin ainoastaan tekemällä pieniä muutoksia aikajanaan.

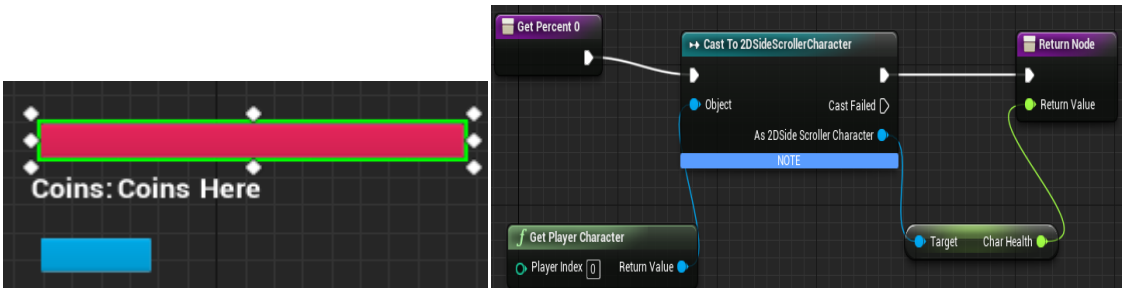
Kyseisessä kuvan esimerkissä on sivuttain liikkuva alusta, eli se liikkuu ainoastaan X-akselilla. Voidaan varmuuden vuoksi pistää Y- ja Z-akselit lukkoon siltä varalta, ettei mitään vahinkoja satu. Aikajanalle asetetaan päämäärä pisteitä eri alueille, jotka määrittävät, missä alusta on milläkin hetkellä.

Ideana oli, että alusta on alussa ruudun keskellä, menee sieltä aluksi ruudun oikeaan päähän, sen jälkeen vasempaan päähän ja sieltä takaisin keskellä sieltä, mistä lähti. Eli aluksi ensimmäinen piste alkaa nollasta, jatkaa sieltä 70 pituusyksikköä oikealle 8 sekunnin aikana, josta menee vasemmalle 140 pituusyksikköä 16 sekunnin aikana ja sieltä taas 70 pituusyksikköä oikealle 8 sekunnin aikana ja on takaisin lähtöpisteessä. Eli hissi liikkuu yhteensä 140 pituusyksikköä molempiin suuntiin ja siten on lopuksi sieltä, missä lähti, ja matkan nopeus on tasaista liikettä, joka tapahtuu kokonaisuudessa 32 sekunnin aikana, 16 sekuntia kumpaankin suuntaan. Sen lisäksi aikajanalle laitetaan ikuinen toistaminen päälle, eli se toistaa samaa reittiä niin kauan, kun peli pyörii.

4.2.7 Heijastusnäytön ja elämäpisteiden lisääminen

Heijastusnäytön saa lisättyä menemällä oikealla hiirenpainikkeella User Interface -> Widget Blueprint. Tänne voi lisätä hahmolle haluttuja ominaisuuksia, joista mielestäni tärkein on elämäpisteet, joiden avulla näkee, kuinka hyvässä kunnossa ollaan ja kauan pysytään vielä elossa. Olen lisännyt peliini myös esim. mittarin, joka kertoo, kuinka monta kolikkoa olen kerännyt, mutta se ei oikeastaan ole kovin olennainen pelissäni, koska en ehtinyt luoda niille mitään syvällistä tarkoitusta. Lisäsin sinne myös mittarin, joka näyttää, onko hahmon kierähdysliike valmiina käytettäväksi. Kierähdyksestä kerron lisää myöhemmässä vaiheessa.

Heijastusnäyttöön on hyvä lisätä esimerkiksi edistymispalkkeja ja tekstikenttiä.



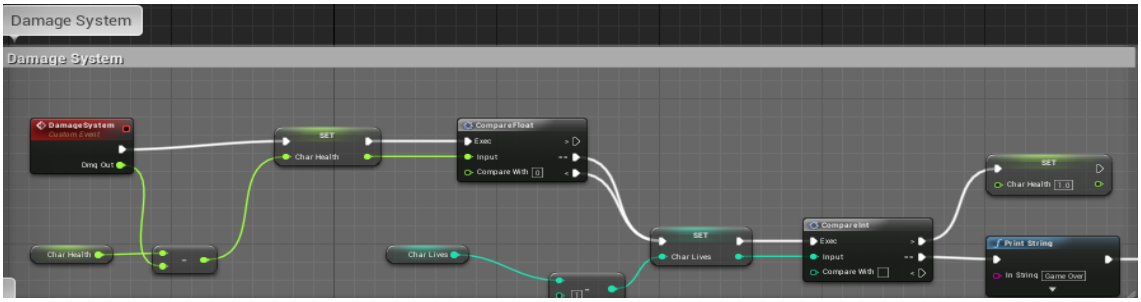
Kuva 17. Heijastusnäytön lisätyt edistymispalkit ja tekstikenttä.

Kuva 18. Elämäpisteiden edistymispalkin sinikopiot.

Heijastusnäytön edistymispalkit pitää synkronoida toimimaan hahmon kanssa ja esimerkiksi elämäpalkille täytyy antaa sinikopioiden avulla tieto, että paljon hahmolla on elämäpisteitä jäljellä, jotta edistymispalkki osaa näyttää sen oikein. Hahmon omiin sinikopioihin tarvitsee myös lisätä heijastusnäyttö päivittymään, jotta se näkyy ja toimii oikein pelin sisällä. Esim. kun törmätään objektiin, kutsutaan vahinkosysteemiä ja määritetään, kuinka paljon se tekee vahinkoa hahmoon.

4.2.8 Vahingon tuottaminen hahmolle

On tärkeää, että hahmollesi pystyy tehdä vahinkoa pelin sisällä, mikäli halutaan, että hahmosi pystyy myös kuolla. Tarpeellista on saada luotua hahmosi sisälle mukautettu vahinkojärjestelmä, jota voidaan kutsua aina sen tarpeen. Vahinkosysteemiä kutsutaan jokaisen objektin ja vihollisen kohdalla, jonka halutaan voivan vahingoittaa hahmoa. Esimerkiksi jos on laavaa, voidaan lisätä laavan sinikopioihin, että kun törmäys tapahtuu, siitä siirretään tieto hahmoon ja kutsutaan vahinkosysteemiä ja otetaan haluttu määrä vahinkoa eli samalla poistetaan hahmon elämäpisteistä sen mukaisesti.



Kuva 19. Hahmon sisäinen vahinkosysteemi tapahtuma.

Käytännössä se, miten systeemi toimii, on, että aina kun tapatumaa kutsutaan, se vähentää annetun määrän hahmon elämäpisteistä ja tarkistaa, onko elämänpisteiden arvo 0 tai pienempi. Mikäli arvo on 0 tai pienempi, hahmolta vähennetään yksi elämä ja sen jälkeen, jos elämiä on vielä jäljellä, niin hahmon elämäpisteet palautetaan takaisin täysiin. Tässä siis käytännössä puhutaan kuolemisesta ja uudelleensyntymisestä. Jos elämiä ei ole enää jäljellä, niin printataan ”Game Over”.

Alussa yritin kovasti saada systeemin toimimaan, jossa hahmolla on tietty määrä elämiä, jotka vähenevät vähitellen ja lopuksi hävitään peli, kun ne loppuvat kesken. Päädyin kuitenkin lopuksi eri ratkaisuun, enkä oikeastaan käyttänyt paljoa hyödyksi hahmon elämiä.

4.2.9 Äänitehosteiden lisääminen peliin

Äänet eivät ole välttämättömiä, mutta ne tuovat peliin aina lisätunnelmaa ja voi olla helpompaa eläytyä peliin, kun tapahtumien aikana kuullaan ääniä. Aluksi on tarpeellista miettiä, halutaanko hakea netistä ilmaisia ääniä vai luoda itse äänitehosteet peliisi. Hain muutamia äänitiedostoja netistä ja kokeilin, miten ne toimivat ja sopivat pelin sisällä. Äänitiedoston formaatti on hyvä olla esimerkiksi .wav. Kaikki formaatit eivät ole yhteensopivia peliprojektin kanssa. Halusin kuitenkin luoda myös omia ääniä peliini ja käytin siihen Audacity-ohjelmaa, jonka avulla sain melko vaivattomasti nauhoitettua lyhyitä äänipätkiä, joita pystyy käyttämään myös pelissäni. Nauhoitin itse pari hassua ääntä ja annoin myös tyttöystäväni tehdä pari hauskaa äänähdystä, joista kaikkia käytin pelissäni. Aluksi ääni nauhoitetaan Audacityn avulla. Sen jälkeen tiedoston formaatti muutetaan .wav-tiedostoksi ja lopuksi äänitiedosto siirretään projektin sisälle. Kun ollaan lisätty äänitiedosto peliin, voidaan painaa oikealla hiirennapilla sitä ja valita ”Create cue”. Pystyt kuunnella ääntä kun laitetaan hiiri äänimerkin kohdalle. Voidaan myös halutessa mennä projektin sisällä lisätyn äänitiedoston muokkausasetuksiin. Pidin itse äänissä pitkälti oletusasetuksia, koska en kokenut tarpeelliseksi muunnella mitään.

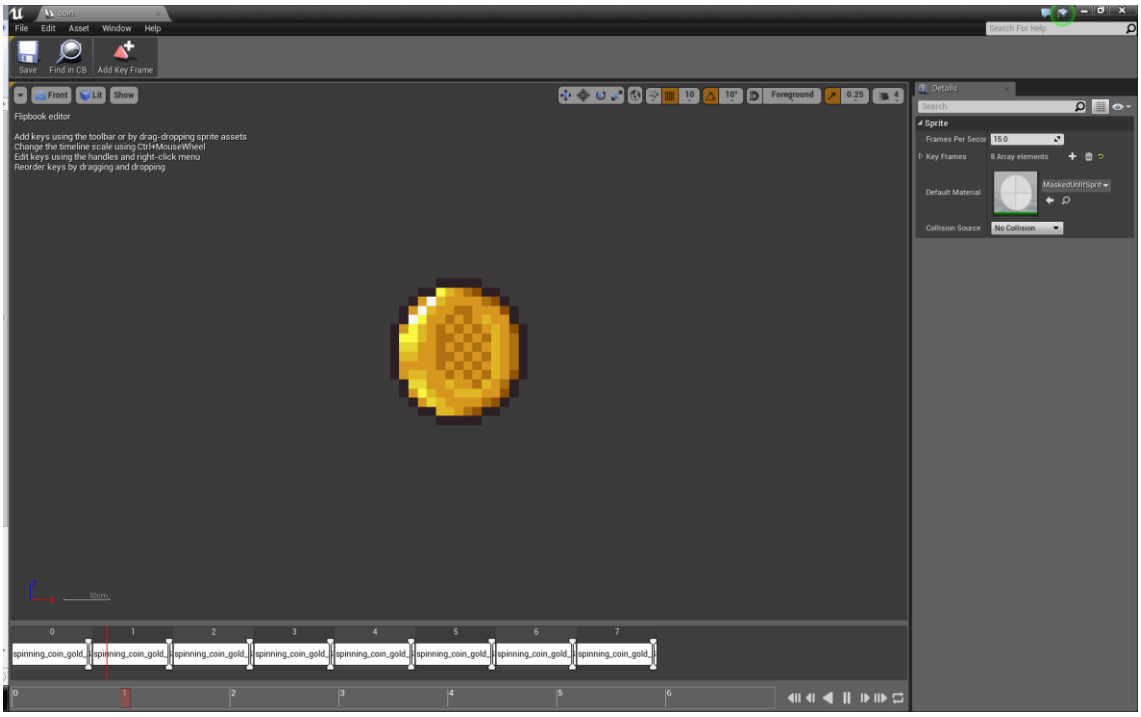
Mikäli haluat käyttää luomaasi äänitehostetta pelin sisällä, sinun pitää lisätä haluamasi tapahtuvan sinikopioihin soita ääni toiminto, josta valitset tekemäsi äänitehosteen. Esimerkiksi, mikäli halutaan kuulla ääni aina kun hahmo hyppää, voidaan mennä hahmon sinikopioihin, etsiä sieltä kohta, jossa käsketään hahmon hypätä ja lisätä heti sen perään käsky soittaa haluttu ääni. Samalla logiikalla pystytään lisätä ääniä minkä tapahtuman hyvänsä perään.

4.3 Pelin toteutuksen vaiheita osa 2

4.3.1 Sprite-animaation luominen

Saatetaan haluta myös jonkunlaisia animaatioita pelissä, tarvitaan ensin sprite-arkki. Mikäli halutaan päästä helpolla, niin voidaan etsiä netistä valmiita sprite-arkkeja, joissa on tausta valmiiksi poistettu. Voidaan myös halutessa luoda oma sprite-arkki, mutta se on jokseenkin työlästä ja tarkkaa puuhaa. Tarvitaan monta kuvaa samasta objektista eri asennoissa, joista pystytään rakentaa animaatio.

Kun on sprite-arkki halutusta objektista, niin lisätään se projektiin ja painetaan oikealla hiirennapilla Sprite Actions -> Extract Sprites. Nyt tarvitaan erotella kuvat halutulla tavalla, automaattiset asetukset yleensä osaavat erotella ne suoraan, mikäli sprite-arkki on selkeästi tehty ilman sekoittavaa taustaa. Tämän jälkeen sinulla pitäisi olla monta eri kuvaa samasta objektista, mikäli kaikki meni nappiin. Voidaan valita kaikki halutut kuvat, joita halutaan käyttää animaatiossa ja painetaan Create Flipbook.

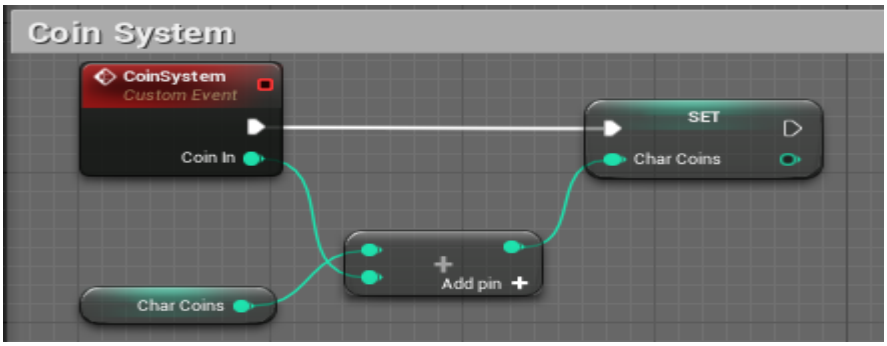


Kuva 20. Kolikon animaatioasetukset.

Aiemman kuvan näkymässä on lisätyt kuvat, jotka automaattisesti pyörii animaationa. Voidaan muuttaa asetuksista, kuinka nopea kuvataajuus halutaan. Kuvissa pystyy myös haluttaessa muuttaa tiettyjä kuvia näkymään pidemmän aikaa kuin toiset, mutta tässä tapauksessa on tasaisesti pyörivä kolikko, joten se ei ole tarpeen. Animaation pystyy lisätä esimerkiksi uuteen toimijaan.

4.3.2 Kolikkokokoelman luominen

Tarvitaan järjestelmä kolikoiden keräämistä varten. Aiemmin tuli tehtyä vahinkojärjestelmä, ja tämä on hyvinkin samankaltainen prosessi. Hahmon sisälle pitää tehdä kolikkojärjestelmä, joka pitää kirjaa, kuinka monta kolikkoa hahmolla on ja osaa lisätä niitä hahmolle sitä mukaa, kun kerää uusia kolikoita.



Kuva 21. Kolikkojärjestelmä.

Tarvitaan toimija, jolle lisätään tehty kolikkoanimaatio. Toimijalle pitää lisätä törmäystunnistus ja sinikopioihin yhdistys hahmoon, josta kutsutaan kolikkojärjestelmää ja lisätään hahmon tietoihin kolikkomäärä+1 kolikon kerätessä. Sinikopioihin lisätään lopuksi käsky tuhota toimija törmäyksen ja kolikonlisäyksen jälkeen, jotta kolikko katoaa eikä samaa kolikkoa tule kerättyä monta kertaa.

Kolikkokokoelma ei itsessään kuitenkaan tee vielä mitään muuta kuin lisää hahmon kolikkomäärää. Siihen voi halutessa lisätä ominaisuuksia, mitä tapahtuu aina tietyn kolikkomäärän kerätessä. Voidaan esimerkiksi antaa hahmolle lisäelämän, lisätä hahmon elämäpisteitä tai parantaa hahmon kykyjä. Minulla oli suunnitelmia kolikoiden kanssa, mutta ajanpuutteen takia en ehtinyt antaa niille suurta tarkoitusta.

4.3.3 Hahmon spritejen lisääminen

Oletushahmon tilalle voi halutessa lisätä omat spritet. En kuitenkaan ole taitava graafikko, joten jouduin turvautumaan hakemaan netistä sopivan näköisen hahmon, jonka lisäsin peliini.

Kuva 22. Lisäämäni pelihahmon animaatioihin tarvittavat kuvat.

Samalla periaatteella kuin aiemmassa kolikkoesimerkissä, projektiin lisätään sprite-arkki, joka puretaan moneksi erilliseksi kuvaksi. Kuitenkin tässä tapauksessa halusin paljon erilaisia animaatioita, joten valitsin aina haluttuja kuvia, joiden avulla sain aikaiseksi pysähdys-, hyppy-, juoksu-, kuolema-, ampumis- ja kierähdysanimaation.

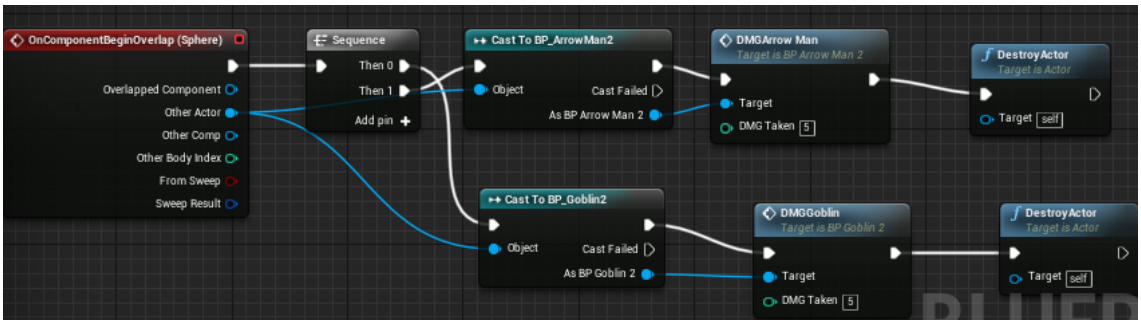
Animaatioita varten voi olla hyvä käyttää ”Enumeration”-järjestelmää. Tämän sisälle kirjoitetaan erikseen uudet nimet kaikille animaatioille, joita halutaan käyttää hahmolla pelissä. Eli lisään aiemmin mainitut animaatiot niminä.

Tarvitaan hahmolle uusi järjestelmä, jonka tarkoituksena on auttaa animaation päivittämisessä. Sinne lisätään ”Enumeration”-muuttuja, joka pitää kirjaa, mikä animaatio on valittu. Tarvitaan ”Switch on Enum_Character_Animations” -toiminto, josta lisätään jokaisen animaation nimi erikseen omaan kohtaan, joille lisätään haluttu animaatio toimimaan ja tekemään sitä käytännössä. Tämä on tarpeellista synkronoida hahmon kanssa, jotta pystytään seuraavaksi lisätä animaatiot haluttuihin kohtiin. Esimerkiksi hahmon hyppäyksen sinikopioihin lisätään ”set char enum” hypyksi samalla hetkellä, kun hahmolle annetaan hyppykäsky. On hyvä varmistaa, että animaatio aina palautuu pysähdysanimaatioon tai juoksuanimaatioon, joten voidaan tehdä esimerkiksi niin, että lisätään sinikopioihin toiminto, joka ottaa hahmon nopeuden, ja jos hahmon nopeus on 0, niin animaatio palautuu pysähdysanimaatioon. Mikäli nopeus on jotain muuta kuin 0, niin animaatio palautuu juoksuanimaatioon.

4.3.4 Ammuntaprojektiilien lisääminen

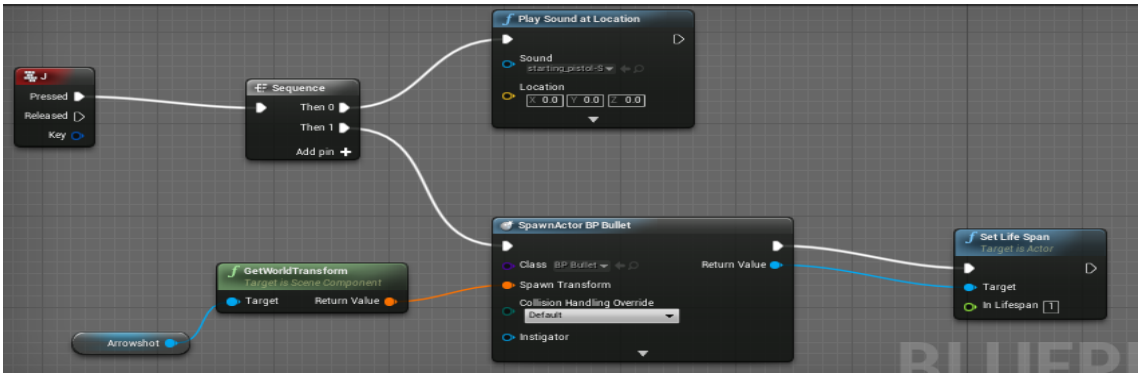
Tarvitaan jälleen uusi toimija, jonka tarkoitus on olla ammuntaprojektiili. Lisätään uusi komponentti, joka määrittää projektiilin liikkeen. Komponentille voi halutessaan lisätä projektiilin nopeutta. Tarvitaan myös komponentti, joka auttaa törmäyksen tunnistuksessa sekä ”Paper Flipbook” komponentin, jolle lisätään sprite, joka näyttää projektiilin liikeanimaation.

Hahmoni ammuksiin olen lisännyt projektiilin sinikopioihin toiminnon, joka tuhoaa panoksen törmäyksen tapahtuessa. Olen myös erikseen lisännyt sinikopioihin, mitä tapahtuu, kun panos osuu viholliseen.



Kuva 23. Ammuksen sinikopiot viholliseen osuttaessa.

Käytännössä törmäyksen tapahtuessa tieto siirretään viholliseen, jonka sisälle on tehty vahinkosysteemi, jonka avulla niiden elämäpisteitä vähennetään haluttu määrä, joka on pelissäni esimerkiksi 5. Lopuksi panos tuhoaa itsensä, koska on täyttänyt tarkoituksensa. Tämä tarkoittaa siis sitä, että panos katoaa pelistä heti, kun se on osunut viholliseen ja tekee samalla tietyn määrän vahinkoa.



Kuva 24. Hahmon sinikopioiden ammuntakäsky.

Hahmolle täytyy saada myös keino ampumiseen. Aiemmassa kuvassa käytännössä määritellään, kun pelaaja painaa tiettyä näppäintä, tässä tapauksessa ”J”, niin soitetaan ampumisääni ja synnytetään ammuntaprojektiili. Ammuntaprojektiilille tarvitsee antaa tieto, mihin kohtaan syntyä. Tarkoitus on saada se täsmäämään hahmon katsomissuuntaa, eli nuolikomponentin mukaisesti. Lopuksi halusin lisätä panokselle eliniän, joka tässä tapauksessa on 1 sekunti. Halusin lisätä eliniän rajoittaakseni ammuksen kulkemaetäisyyttä. Ei ole tarkoitus, että panos menee monen kuvaruudun läpi ja osuu vihollisia, joita ei olla vielä edes nähty.

4.3.5 Satuttavan objektin lisääminen

Käyttämällä aiempaa tietoa ja systeemejä hyväksi, satuttavan objektin lisääminen on melko helppoa. Voidaan tehdä tai hakea esimerkiksi piikin näköisiä objekteja, jotka lisätään projektiin. Tehdään toimija, jolle lisätään kyseinen sprite ja lisätään laatikon törmäysalue täsmäämään sopivasti spriteä. Toimijan sinikopioihin halutaan törmäyksen tapahtuessa liittää tieto hahmoon ja kutsua vahinkosysteemiä, jotta objekti tuottaa vauriota hahmolle. Voidaan vapaasti lisätä tämä toimija pelikentällesi.

4.3.6 Vihollisten tekeminen

Vihollisten tekemisessä vaadittu työmäärä riippuu hyvin paljon siitä, kuinka monimutkaisesti tahdotaan vihollisten toimivan. Lisäsin peliini 2 vihollista, joista ensimmäinen kävelee ja seuraa pelaajaa niin kauan, kunnes pelaaja menee tarpeeksi kauas tai tuhoaa vihollisen. Voidaan kutsua tätä vaikka peikoksi. Toinen vihollinen on jousiampuja, joka alkaa aina ampumaan nuolia pelaajaa kohti suoraviivaisesti vaakatasoon pelaajan nähtyään ja tekee sitä, kunnes pelaaja menee tarpeeksi kauas tai tuhoaa vihollisen, samoin kuin peikon kanssa.

Vihollista varten tarvitset uuden toimijan, jolle voidaan lisätä haluttu sprite. Peikolle ja jousiampujalle löytyi netistä hyvä sprite-arkki, joissa oli monia kuvia samoista vihollisista, joiden avulla pystyy helpommin luomaan animaatioita haluttaessa.

Kuva 25. Jousiampuja vihollisen sprite-arkki.

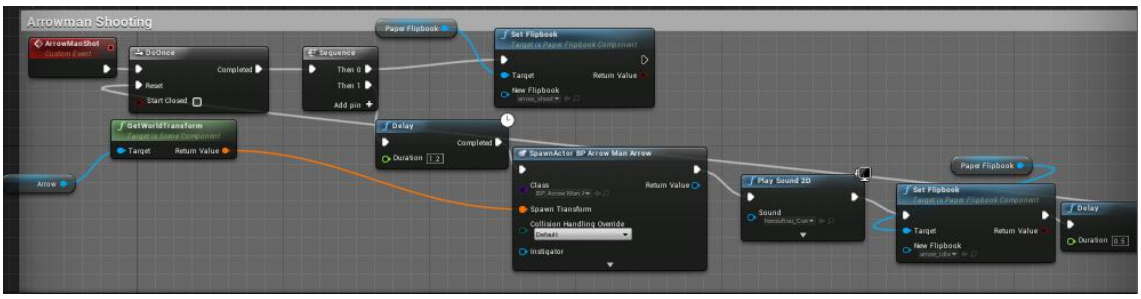
Viholliselle kannattaa lisätä kapsuli, jolle asetetaan oikeanlainen törmäysalue täsmäämään visuaalisesti spriteä. Mikäli halutaan, että vihollisen pystyy tappamaan, sille on hyvä lisätä sinikopioihin vihollisen oma vahinkojärjestelmä samankaltaisesti kuin aiemmin pelaajahahmon kanssa. Aina kun vahinkojärjestelmää kutsutaan, vihollisen elämäpisteistä vähennetään tietty määrä ja tarkistetaan, paljonko vähennyksen jälkeen vihollisella on elämäpisteitä jäljellä. Mikäli elämäpisteitä on vähemmän tai yhtäsuuri kuin 0, niin tuhotaan toimija, eli vihollinen poistetaan pelimaailmasta.

Jotta pelaajahahmo pystyy vahingoittamaan vihollisia, ammuttaviin projektiileihin täytyy lisätä sinikopioihin tieto, mitä tehdään törmäyksen tapahtuessa. Mikäli luoti törmää viholliseen, siitä siirretään tieto eteenpäin vihollisen sinikopioihin, joista kutsutaan vihollisen omaa vahinkojärjestelmää ja lopuksi tieto, että luoti voidaan tuhota pelimaailmasta.

Halusin myös lisätä vihollisen sinikopioihin, että aina kun hahmo törmää viholliseen, niin hahmoon lisätään tieto, kutsutaan hahmon vahinkojärjestelmää ja tehdään vahinkoa niin paljon, että hahmo kuolee.

Halusin tehdä ampumisanimaation jousiampuja viholliselle, jossa käytin aiempaa sprite-arkkia hyödykseni. Animaatio tehdään samalla periaatteella kuin aiemminkin ja yritetään saada se näyttämään hyvältä pelaajan silmään.

Jotta jousiampuja pystyy ampumaan nuolia, niin kyseisen vihollisen sinikopioihin tarvitsee tehdä muutamia lisäyksiä. Tehdään uusi jousiampujan ampumisjärjestelmä, jossa määritellään, mitä tapahtuu, kun ampumisjärjestelmää kutsutaan. Ideana on, että lopputuloksena pienin väliajoin jousiampuja näyttää animaation ja ampuu nuolen.



Kuva 26. Jousiampujan ampumisjärjestelmän sinikopiot.

Käytännössä aluksi annetaan käsky, että tehdään tämä vain kerran, asetetaan jousiampuja tekemään ampumisanimaatiota. Sen jälkeen laitetaan pieni viive, jotta animaatio saadaan täsmäämään näyttämään siltä, että nuoli ammutaan vasta animaation lopussa. Viiveen jälkeen synnytetään nuoli jousiampujan katsomissuuntaan ja soitetaan ääniefekti. Lopuksi jousiampujan animaatio laitetaan takaisin joutilaaksi ja lisätään pieni viive, jonka jälkeen nollataan ensimmäinen käsky ja aloitetaan juttu alusta, eli jousiampuja jatkaa ampumista.

Jousiampujalle halutaan myös, että hän alkaa ampumaan vasta, kun pelaajahahmo on tarpeeksi lähellä ja lopettaa ampumisen, mikäli hahmo menee tarpeeksi kauas jousiampujasta. Jousiampujan on hyvä osata myös kääntyä siihen suuntaan päin, missä pelaajahahmo on sen sijaan, että katsoisi aina samaan suuntaan siitä huolimatta, missä hahmo on.

Voidaan tehdä uusi muuttuja, jolla määritetään jousiampujan ja pelaajahahmon välinen etäisyys. Ensin haetaan jousiampujan X-akselin sijaintiarvo ja vähennetään siitä pelaajahahmon X-akselin sijaintiarvo. Tämä on etäisyysmuuttujan arvo.

Etäisyysmuuttujan selvitettyä voidaan tehdä tarkistus, onko etäisyysmuuttuja pienempi tai yhtä suuri kuin 250 pituusyksikköä. Mikäli tämä on totta, niin tehdään uusi tarkistus siitä, onko etäisyysmuuttuja suurempi tai yhtä suuri kuin -250 pituusyksikköä. Mikäli molemmat pitävät paikkansa, kutsutaan jousiampujan ampumisjärjestelmää. Syy, miksi tähän tarvitsi 2 erilaista tarkistusta 1 tarkistuksen sijaan, oli se, että mikäli oltaisiin tarkistettu ainoastaan, onko etäisyysmuuttuja pienempi tai yhtä suuri kuin 250, niin jousiampuja olisi lopettanut ampumisen oikealla tavalla yhteen suuntaan. Mutta mikäli mentäisiin toiseen suuntaan, niin jousiampuja ampuisi loputtomasti perään, vaikka mentäisiin kuinka kauaksi. Eli käytännössä tarkistetaan ollaanko ison matkan päässä vasemmalla tai oikealla. Käytetty pituusyksikkö olisi voinut olla mikä vain haluttu määrä, mutta 250 ja -250 vastasi lähes yhden kuvaruudun pituutta pelimaailmassa, jonka arvelin olevan sopiva määrä. Etäisyysmuuttujan arvoa tarkistetaan monta kertaa sekunnissa koko ajan.

Kääntymisen tekemistä varten tarvitsemme tiedon jousiampujan sijainnista ja pelaajahahmon sijainnista, joita vertaillaan keskenään. Tarkistuksen jälkeen jousiampujalle annetaan käsky kääntymään toiseen suuntaan, mikäli ei osoita pelaajahahmon suuntaan.

Jousiampujan en halunnut ollenkaan liikkuvan, mutta peikolle halusin antaa myös kyvyn kävellä pelimaailmassa ja jahdata pelaajahahmoa tämän ollessa tarpeeksi lähellä. Tämän tekemiseksi voidaan samankaltaisesti kuin aiemmin, käyttää etäisyysmuuttujaa hyödyksi ja tarkistaa sen avulla pelaajahahmon ja peikon välinen etäisyys. Mikäli pelaajahahmo on tarpeeksi lähellä peikkoa, niin lisätään pieni viive, esim. 0,015 s, jonka jälkeen peikon X-akselin sijaintia lisätään tai vähennetään siitä riippuen, onko pelaaja vasemmalla vai oikealla puolella peikkoa. Eli käytännössä peikko liikkuu halutun määrän tietyin väliajoin. Itse halusin, että peikko näyttää liikkuvan melko tasaisesti ja sen takia halusin peikon liikkuvan 0,015 s väliajoin. Mikäli peikko liikkuisi sekunnin välein, niin se näyttäisi todella tökkivältä liikkeeltä.

4.3.7 Kuperkeikka ja pelin tasot

Halusin tehdä hahmolleni kuperkeikan, jonka tarkoitus oli saada nopeuspyrähdys senhetkiseen liikkumasuuntaan. Olin tehnyt aiemmin animaation pyörähdykselle, jota käytin kuperkeikan tekemiseen. Ensin aloitetaan tapahtuma siitä, kun pelaaja painaa tiettyä näppäintä, esimerkiksi ”K”. Silloin muutetaan pelaajahahmon animaatio kuperkeikaksi ja pelaajahahmon kävelemisnopeus muutetaan väliaikaisesti esimerkiksi 2 kertaa isommaksi kuin normaali kävelynopeus. Pienen viiveen jälkeen hahmon kävelynopeus palautetaan takaisin normaaliarvoon.

Halusin tehdä myös rajoituksen kuperkeikan käytölle, jotta sitä ei voisi rämpyttää koko ajan. Tein totuusarvomuuttujan, jonka avulla tarkistin, onko kuperkeikka uudelleen käytettävissä vai ei. Lisäsin kävelynopeuden normaaliarvoksi palautumisen jälkeen pienen viiveen, jonka jälkeen kuperkeikka on taas valmiiksi tehtäväksi. Lisäsin myös heijastusnäyttööni tarkistuksen, josta näkyi, onko kuperkeikka käytettävissä. Käytännössä heijastusnäyttö saa pelaajan tiedot ja tarkistaa totuusarvomuuttujan perusteella tilanteen. Palkki näkyy sinisenä, mikäli kuperkeikka on käytettävissä, ja harmaana, mikäli se ei ole vielä käytettävissä.

Aiemmin olin tehnyt hahmolleni vahinkojärjestelmän, jonka avulla pelaajahahmo pystyi ottamaan vahinkoa. Tarvitsen kuitenkin, että pelaaja pystyy kuolla elämäpisteiden mennessä nollille. Tein niin, että pelaajan kuoleman tapahtuessa pienen viiveen jälkeen käytetään toimintoa, joka hakee nykyisen tason nimen, jonka jälkeen lisätään avaa taso toiminto, joka avaa nykyisen tason. Tämä tarkoittaa siis sitä, että pelaajan kuoltua aloitetaan koko taso täysin alkupisteestä ja nollataan tilanne täysin. Näin voidaan yrittää aina alusta uudelleen kuoltua. Mikäli päästään seuraavaan tasoon, niin silloin tason nimi muuttuu ja synnytään uudelleen samassa tasossa, missä kuoltiin. Eli ei jouduta aloittamaan peliä kokonaan täysin alusta vain kyseisen tason alusta. Tason pituudet riippuvat täysin pelin luojan preferensseistä. Tasoni ovat keskipituisia, mennään noin muutaman kuvaruudun eteenpäin, niin ollaan päästy seuraavaan tasoon.

Tein myös tason loppuun portaalin, joka vie seuraavaan tasoon. Käytännössä kun törmätään portaaliin, niin esimerkiksi tason 1 lopussa avataan taso 2. Tähän käytetään toimijaa, jolle lisätään sprite ja törmäysalue samaan tyyliin kuin aiemmilla toimijoilla.

4.4 Vastaan tulevia ongelmia

Pieniä ongelmia tuli jatkuvasti, koska koko prosessi on täynnä ongelmanratkaisua. Kuitenkin mainitsen muutamia isoja ongelmia, joita on tullut projektin varrella.

1. Tuplahyppäys pelaajahahmolla. Kun yritin saada tuplahypyn toimimaan kunnolla, se ei ikinä toiminut juuri sillä tavalla, kuin olisin halunnut. Välillä hyppy menee väärään suuntaan ja välillä hyppyyn ei saa lähes yhtään korkeutta. Hypyn toimivuus riippui paljon siitä, olenko putoamassa vai nousemassa, mikä ei ollut tarkoitus. En ollut tajunnut käyttää ”Zoverride”-toimintoa, jonka avulla sain ratkaistua ongelman. Tämän avulla sain korvattua hahmon nopeuden sen sijaan, että nopeutta lisätään hahmon nykyiseen nopeuteen.

2. Objekti ei täsmännyt visuaalisesti sitä, miten pelaaja astuu sen päälle. Näytti, että pelaaja pystyy astumaan korkean objektin päälle äkkinäisesti, vaikka sitä ei pitäisi tapahtua. En oikeastaan ikinä keksinyt tähän mitään kovin fiksua ratkaisua, päätin vain tehdä pelitasostani hieman erilaisen, jonka yhteydessä ongelma katosi.

3. Pelaajahahmon ammuskudit eivät tunnistaneet törmäyksiä oikealla tavalla. Minulla oli ongelmia saada luodit osumaan seiniin ja vihollisiin, mutta ei kuitenkaan toisiinsa. Yleensä se meni niin, että joko ammukset osuu kaikkeen tai ei lähes mihinkään. Oli vaikea saada eroteltua, että tiettyihin asioihin voi osua, mutta tiettyihin ei. En halunnut, että luotini osuvat toisiinsa, koska silloin se näyttää siltä, että kudit katoavat tyhjyyteen, eikä mene viholliseen asti edes lähietäisyydeltä. Sain kuitenkin sen toimimaan, kun kokeilin erilaisia törmäysasetuksia ja muuttelin hieman projektin asetuksia.

4. Viholliset menevät lattioista ja seinistä läpi. Minulla oli paljon ongelmia saada vihollisten fysiikat toimimaan oikealla tavalla, ne yllättäen putoavat lattiasta läpi tai kävelevät suoraan seinän läpi, vaikka tämä ei ollut ollenkaan tarkoitus. Sain tämän ratkaistua sillä tavalla, että lukitsin vihollisten Y-akselin sijainnin, jotta siihen ei tulisi mitään muutoksia vihollisten liikkuessa, sillä se sekoitti ne täysin. Pelini on 2D, joten en käytännössä tarvitse Y-akselia ollenkaan, käytössä on X- ja Z-akselit.

5. Pelaajan elämien ja tarkastuspisteiden toimivuus. Tämä oli reilusti isoin ongelma, johon törmäsin. Halusin alun perin hahmolleni elämäjärjestelmän, esimerkiksi 3 elämää, joiden jälkeen peli olisi ohi. Elämäjärjestelmä itsessään ei ollut ongelma; sain sen tehtyä. Ongelma oli siinä, että pelaajan sijainti, vihollisten sijainnit ja muiden objektien sijainnit pitäisi saada manuaalisesti nollattua tason alkutilaan. Käytännössä pitää siis saada pelaajalle tallennuspiste, jonne hänet siirretään kuoltua, ja mikäli löydetään seuraavan tarkastuspisteen, niin synnytään uudelleen sinne. Sain jopa pelaajan toimimaankin tällä tavalla, mutta en ollut ottanut huomioon, kuinka vaikeata olisi manuaalisesti synnyttää jokainen vihollinen uudelleen ja siirtää ne lähtöpaikoillensa. Tajuttuani kuinka aikaa vievää tämä olisi minulta, niin päädyin tekemään vaihtoehtoisen ratkaisun ja tein erilliset tasot ja aina kuoltua koko taso nollataan ja aloitetaan alusta.

6. Taustan lisääminen pelaajahahmon kameran eteen. Yritin lisätä pelaajahahmoa seuraavan kameran eteen taustan, jotta aina kun pelaaja liikkuu, niin siellä näkyy hieno tausta samalla jatkuvasti, joka on synkronoitu pelaajahahmon ympärille. Aluksi tämä tuntui olevan melko vaivaton ja helppo toteuttaa, mutta yllättäen minulle tapahtui jotain outoa. Yritin avata pelini, ja ruutuni oli täysin musta, yritin peruuttaa aiempia tekojani, mutta se oli liian myöhäistä. En enää tiennyt tai keksinyt, missä ongelma oikein oli. Tarkastin moneen otteeseen kaikkia mahdollisia arvoja ja yritin poistaa taustankin kokonaan, mutta en saanut enää normaalia näkymää toimimaan.

   Jouduin turvautumaan varmuuskopioihin pelistäni ja otin vanhempia pelitiedostoja käyttöön uusien tilalle. Tässä tuli kuitenkin myös sellainen ongelma, että olin käyttänyt sinikopioissani tiettyä hahmoa enkä keksinyt, miten voisin suoraan muuttaa kaikki maininnat toiseen hahmoon. Jouduin manuaalisesti muuttamaan kaikki kohdat, joissa hahmo mainitaan ja siirtämään tiedot toiseen hahmoon. Tämä ei välttämättä ollut paras tai fiksuin ratkaisu, mutta se oli sillä hetkellä ainut ratkaisu, jonka onnistuin keksimään.

5 Yhteenveto

Tämän insinöörityön tavoitteena oli toteuttaa toimiva 2D Side-scroller -peli Unreal Engineä käyttäen. Loppujen lopuksi onnistuin tavoitteessani.

Käytiin läpi yksityiskohtaisesti pelin tekemisen eri vaiheita ja kerrottiin käytetyistä pelikehityksen työkaluista ja pelin suunnittelusta. Olisin voinut kertoa vielä tarkemminkin yksityiskohdista, mutta halusin pitää opinnäytetyön sivumäärät sopivissa määrissä, joten optimoin ja tiivistelin asioita hieman.

Pelinkehityksessä tuli kuitenkin mutkia matkan varrella ja projektissa meni pidempi aika kuin olin arvioinut. En saanut toteutusta täsmälleen sen mukaiseksi kuin olisin itse toivonut alun perin, mutta sain kuitenkin sen riittävän toimivaksi, jotta sitä voisi kutsua toimivaksi peliksi.

Olisin halunnut kehittää peliä vielä pidemmälle ja tehdä siihen lisää erilaisia pelimekaniikkoja, mutta ajan käyden vähiin jouduin tyytymään tekemään kompromisseja. Voisin esimerkiksi lisätä kolikoille syvemmän tarkoituksen pelin sisällä ja lisätä monia erilaisia pelitasoja, sekä tehdä tarkastuspisteistä ja elämäjärjestelmästä toimivampia.

Olen kuitenkin lopputulokseen tyytyväinen ja opinnäytetyön tekemisen myötä sain kehitettyä Unreal Engine 4:n tuntemusta ja laajennettua omaa ymmärrystä pelin tekemisen vaiheista ja eri osa-alueista. Tulevaisuudessa pystyn suunnittelemaan aikataulun kanssa paremmin, koska olen saanut paremman käsityksen paljon eri tehtävät vaativat aikaa.

Lähteet

1 Neogames. <https://www.neogames.fi/tietoa-toimialasta/>. Luettu 31.3.2017.

2 Wikipedia. Verkkodokumentti. Game Engine. <https://en.wikipedia.org/wiki/Game_engine/>. Luettu 1.5.2017.

3 Epic Games. <https://www.unrealengine.com/unreal-engine-4>. Luettu 5.5.2017.

4 Epic Games. <https://www.unrealengine.com/what-is-unreal-engine-4>. Luettu 5.5.2017.

5 Maximum PC Staff. The 10 Best Unreal Engine 3 Games. <http://www.pcgamer.com/10-best-unreal-engine-3-games/>. Luettu 5.5.2017.

6 Epic Games 2015 Blueprints Visual Scripting. <https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html>. Luettu 5.5.2017.

7 Unity Technologies 2016a. Unity – Game engine, tools and multiplatform. <https://unity3d.com/unity>. Luettu 6.5.2017.

8 Gur Dotan, 20.1.2015. Top 10 Unity Games Ever Made <http://blog.soom.la/2015/01/top-10-unity-games-ever-made.html>. Luettu 6.5.2017.

9 Unity Technologies 2016c. Fast Facts. <https://unity3d.com/public-relations>. Luettu 6.5.2017.

10 Wikipedia. Verkkodokumentti. Paint Shop Pro. <https://fi.wikipedia.org/wiki/Paint_Shop_Pro>. Luettu 6.5.2017.

11 Wikipedia. Verkkodokumentti. Paint. <https://fi.wikipedia.org/wiki/Microsoft_Paint>. Luettu 6.5.2017.

12 Audacity Team. <http://www.audacityteam.org/>. Luettu 6.5.2017.

13 Brown, M. 2015. Game Maker's Toolkit. <https://www.youtube.com/watch?v=k70_jvVOcG0>. Luettu 6.5.2017.