Opinnäytetyö (AMK)

Tietotekniikka / Peliteknologia

NTIETS13P

2016

Aukusti Manninen

UNITY 5 EDITORIN LAAJENTAMINEN

– työkalu 3D-mallikokoelman muokkaamiseen

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka / Peliteknologia

2016 | Sivumäärä

Yliopettaja Mika Luimula

Aukusti Manninen

UNITY 5 EDITORIN LAAJENTAMINEN

- työkalu 3D-mallikokoelman muokkaamiseen

Työn tarkoituksena oli tutkia ja kehittää 3D-mallikokoelman muokkaustyökalu Unity 5 -pelimoottoriin. Työkalun oli tarkoitus toimia editori-tilassa, mikä tarkoitti sitä, että pelimoottori ei ollut aktiivisena. Tämän vuoksi kehitystyö poikkesi normaalista Unityn parissa työskentelystä. Laajennuksen avulla käyttäjä kykeni muokkaamaan mallikokoelmaa, joka koostui numeroiden ja desibeli-lyhenteen 3D-malleista. Laajennuksen sai käyttöön lisäämällä itsetehty Unityn primitiiviobjekti Scene-ikkunaan. Tämän jälkeen muokkaus tapahtui primitiiviobjektissa olevan komponentin parametrien avulla.

Teoriaosuudessa tutkittiin eri pelimoottoreita ja niitä tarkasteltiin myös muokattavuuden näkökulmasta. Tutkimuksen perusteella päädyttiin Unity 5 -pelimoottoriin sen tehokkaan editoriohjelmointirajapinnan vuoksi. Vertailun jälkeen teoriaosuudessa käytiin läpi Unity 5 -pelimoottorin ominaisuuksia ja pelimoottorin laajentamista yleisellä tasolla. Laajentamiseen liittyvät huomioitavat asiat käytiin myös läpi. Teoriaosuuden loppupuolessa perehdyttiin opinnäytetyön etenemiseen sekä vastaan tulleisiin ongelmiin ja niiden ratkaisuihin. Teoriaosuus päättyy lopputyön kriittiseen arviointiin, jossa myös käsitellään puutteita sekä jatkoehdotuksia.

Opinnäytetyön asiakkaana toimi Uplause-yritys Helsingistä ja sen aikana valmistui laajennus, joka täytti asiakkaan toiveet. Työ toteutettiin Unity 5 -pelimoottorilla ja ohjelmointikielenä käytettiin C#-ohjelmointikieltä. Asiakas toimitti 3D-mallikokoelmat, joita muutettiin laajennukseen sopivaksi.

ASIASANAT:

Pelimoottori, Unity, Editorin laajentaminen

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology / Game Technology

2016 | Total number of pages

Principal Lecturer Mika Luimula, Adj. Prof.

Aukusti Manninen

EXTENDING THE UNITY 5

- tool for modifying 3D-model collection

The goal of this thesis was to study and develop a parametric customization tool extension for a 3D-model collection to the Unity 5. The extension was required to work in the editor mode. This meant that the game engine itself was not active or in play mode. Because of this, the working methods differ from the normal Unity game development. With this extension, a user could modify model collection which includes the numbers and an abbreviation of decibel. The extension was made available when a user added a custom made primitive object within the Unity. After the creation of the primitive object, a customization of the models is done with the components parameters.

In the theory part of the thesis starts with a research of the different game engines. The results of the research were used to decide on which game engine to use in the development. After this, the theory part includes a research about extending the Unity editor. End part of the theory contains a reflection about the development of the parametric customization tool for 3D-model collection. Also there were conclusions about the development process with suggestions for the continuation of the development.

The customer of this thesis was company called Uplause from Helsinki. The product of the practical part of the thesis was an extension which covered almost every aspect that the customer wanted. The work was done with the Unity 5 game engine and C# programming language. The customer gave the 3D-model collection which were customized to work with the extension properly.

KEYWORDS:

Game engine, Unity, Extending the Unity

SISÄLTÖ

KUVAT

KÄYTETTY SANASTO

IDE	Integrated Development Environment. Kehitysympäristö, joka on tehty ohjelmointia varten.
OpenGL	Ohjelmointirajapinta grafiikan tuottamiseen
Vulkan	Khronos Groupin julkaisema ohjelmointirajapinta, joka tulee olemaan OpenGL:n seuraaja
UV-kartta	UV-kartta heijastetaan 3D-mallin päälle. Se on 2D-tekstuuri. U- ja V-kirjaimia käytetään X- ja Y-kirjainten sijaan, koska 3D-ympäristö on jo varannut X-, Y- ja Z-kirjaimet.
RGB	Tietokoneen käyttämät digitaaliset väriarvot Red, Green ja Blue.
Ohjelmointirajapinta	Tietokoneohjelmien välille luotu toiminallisuus, joka mahdollistaa niiden välisen kommunikoinnin ja tiedonvaihdon.
Verteksi	3D-mallissa oleva solmukohta tai piste, johon yhdistyvät mallin reunat. Pisteet ja reunat muodostavat 3D-mallin rautalankaversion.
Primitiiviobjekti	Yksinkertainen 3D-muoto Unity-pelimoottorissa. Näihin kuuluvat esimerkiksi kuutio, sylinteri, taso ja kuula.

1 JOHDANTO

Pelinkehitykseen on tarjolla useita eri kehitystyökaluja, jotka auttavat pelinkehittäjää työskentelyssä. Kehittyneimpiä työkaluja kutsutaan pelimoottoreiksi. Kehitystyökaluina ne tarjoavat pelinkehittäjälle esimerkiksi fysiikka-moottorin, animaatiotyökaluja, sekä käyttöliittymän. Ohjelmointi tapahtuu perinteisellä tavalla, mutta apuna käytetään pelimoottorin kirjastoja sekä pelimoottorin mukana tulevaa käyttöliittymää.

Opinnäytetyön tavoitteena on laajentaa Unity 5 -pelimoottoria lisäämällä siihen oma työkalu sekä primitiiviobjekti, joka sisältää työkalun komponentin. Asiakkaana opinnäytetyössä toimi Uplause niminen yritys Helsingistä. Yritys tuottaa desibelimittauksen avulla ohjattavia visualisointeja ja pelejä stadioneiden isoille näytöille, joita yleisö kontrolloi äänen avulla.

Työkalu toimii Unityn editoritilassa, ja sen tarkoitus on mahdollistaa 3D-mallien muokkaaminen ilman pelimoottorin aktivointia. 3D-mallit koostuvat numeroista ja desibelin lyhenteestä. Laajennus toteutetaan käyttämällä Unityn editoriohjelmointirajapintaa apuna käyttäen. Mallikokoelmaa muokataan tyhjään peliobjektiin liitetyn komponentin avulla. Laajennuksen avulla kyetään muokkaamaan 3D-mallikokoelman merkkien kaltevuutta, lihavointia, pääteviivaa sekä syvyyttä. Asiakkaan toiveiden mukaisesti siihen liitettiin myös muokkaustyökalu, jolla kyetään muuttamaan merkin jokaista sivun paksuutta erikseen. Ulkoasua voi muuttaa sijoittamalla materiaaleja komponentissa oleviin materiaalipaikkoihin sekä muuttaa väriä värinvalitsimella. Komponentti sisältää kolmelle materiaalille paikat ja ne ovat merkin täyttömateriaali, reunusmateriaali sekä sivumateriaali. Laajennuksessa on automaattinen paketointi kun käyttäjä on saanut merkkien muokkauksen valmiiksi. Paketin talletuskansion saa itse valita ja sen voi ottaa käyttöön muissa Unity-projekteissa. Työn ohessa suoritetaan tutkimusta eri tavoista toteuttaa laajennus sekä samalla testataan laajennuksen toimivuutta pelimoottorin ollessa aktiivisena. Tutkimukseen sisältyy myös kirjainten typografisten ominaisuuksien ymmärtäminen ja niiden soveltaminen editorityökaluun. Työn tilaaja antaa 3D-kirjaimet, mutta työskentelyssä vaaditaan 3D-mallinnustaitoja ja -tietoja.

Kirjallisessa osiossa vertaillaan eri pelimoottoreita ja niiden muokkaavuutta. Vertailun pohjalta perustellaan pelimoottorin valintaa. Tämä jälkeen tarkempaan tarkasteluun otetaan Unity 5 -pelimoottori ja tutkitaan sen soveltuvuutta omien työkalujen tekemiseen.

Tutkimuksessa käydään läpi editorin laajentamiseen liittyvät alkutoimet sekä eroavaisuudet Unity 5 -pelimoottorin tavanomaisen ohjelmoinnin ja editoriohjelmoinnin välillä. Samalla tarkastellaan huomioitavia asioita, jotka vaikuttavat suorituskykyyn sekä muihin ongelmiin. Opinnäytetyön teoriaosuus käsittelee uutta aihe aluetta, josta löytyy vai vähän tieteellistä tutkimusta. Opinnäytetyön kirjallisen osion lopussa verrataan muita valmiita 3D-tekstityökaluja toisiinsa ja tutkitaan niiden ominaisuuksia. Samalla tarkastellaan lähemmin opinnäytetyön etenemistä suunnittelusta toteutuksen kautta valmiiseen tuotokseen.

Työn tarkoituksena on luoda tehokas ja yksinkertainen työkalu, joka mahdollistaa valmiiden 3D-mallikokoelman muokkaamisen typografian tavoin. Työkalun on tarkoitus nopeuttaa tai vähentää käsintehtyjen 3D-mallikokoelmien työtaakkaa. Ohjelmoinnin ohessa toteutetaan tutkimusta laajennuksen tehokkuudesta ja hyödyistä ohjelmiston kehitystyössä. Samalla testataan laajennuksen käyttämistä suoraan ohjelmassa, jotta havaittaisiin sen käyttömahdollisuudet efektien luomisessa.

2 PELIMOOTTORIT KEHITYSTYÖKALUINA

Pelinkehittäjien avuksi löytyy lukematon määrä erilaisia pelimoottoreita. Pelimoottori on kehitystyökalu, joka tehostaa pelien tekemistä. Ne sisältävät valmiita kirjastoja tai jopa käyttöliittymiä pelikehittäjien avuksi. Pelimoottoreita on monenlaisia, ja niitä on kehitetty erilaisiin tarkoituksiin. Esimerkiksi Litbcod-pelimoottori, joka on enemmänkin funktiokirjasto kuin pelimoottori, kohdistuu ”rogue-like”-tyylisten ascii-symbolipelien tekemiseen. Kun taas modernit pelimoottorit, kuten Unity 5 tai Unreal Engine, hallitsevat laajasti monia pelien osa-alueita, kuten fysiikkaa, grafiikkaa ja animointia. Niihin on myös luotu käyttöliittymät helpottamaan käyttöä.

2.1 Pelimoottorien vertailu

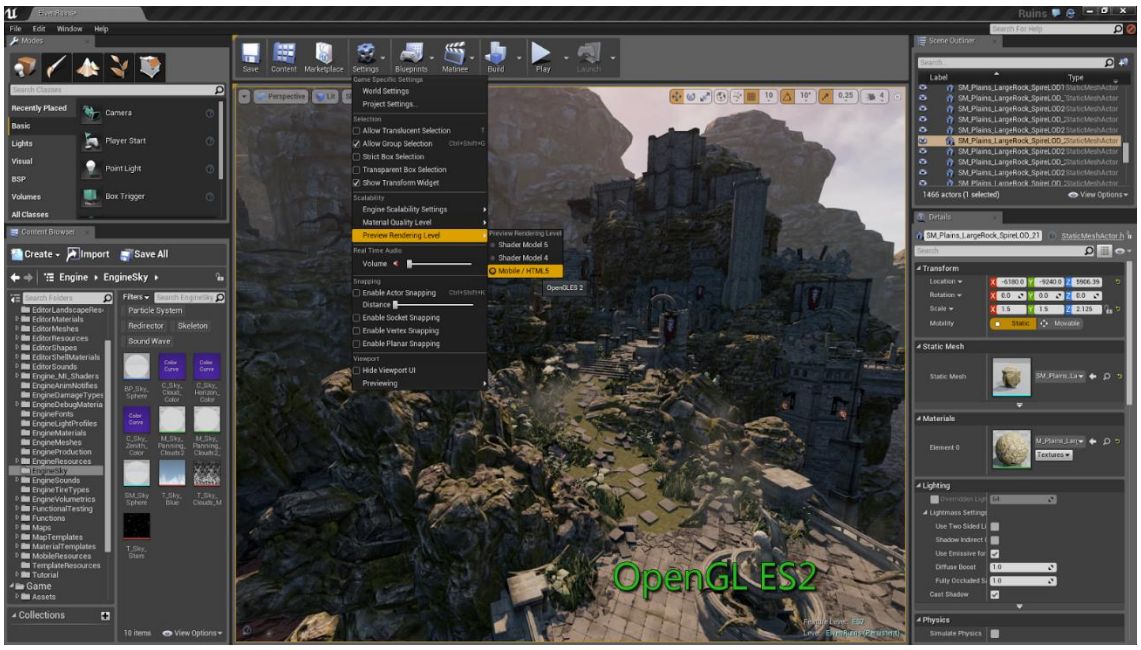
Tarkastelun kohteeksi valittiin neljä parhainta pelimoottoria, joihin kuuluvat Unreal Engine, Unity 5, CryEngine ja Source 2 (Framebench 2015). Pelimoottoreita on hyvä verrata keskenään ennen projektin aloitusta, jotta saa käyttöönsä parhaan mahdollisen työkalun juuri kyseistä projektia varten. Toiset pelimoottorit sopivat paremmin 3D-pelien tekemiseen, kun taas toisilla on helpompi kehittää 2D-pelejä. Pelimoottoreissa vaihtelevat monet asiat, kuten muokattavuus, visuaalinen lopputulos, fysiikkamallinnus ja tehokkuus. Toinen asia, mikä täytyy ottaa huomioon pelimoottoria valittaessa, on ohjelmointikieli. Käytetyimpiä ohjelmointikieliä ovat C++, C# tai JavaScript. Python-ohjelmointikieli löytyy Blender 3D-mallinnusohjelman pelimoottorista sekä PyGame-pelinkehityskirjastosta.

2.1.1 Unreal Engine

Epic’s Unreal Engine 4 -pelimoottori on ilmainen, mutta siinä on 5 % rojaltimaksu saaduista tuotoista (Unreal Engine 2016a). Kuvassa 1 on pelimoottorin käyttöliittymä. Unreal Engine 4 tukee monia eri alustoja, ja siinä käytetään ”Blueprints Visual Scripting”-järjestelmää, joka on visuaalista palikkaohjelmointia. Mahdollista on myös käyttää C++-ohjelmointikieltä kun haluaa toteuttaa jotain mitä ei valmiilla palikoilla pysty toteuttamaan. Pelimoottori on markkinoiden kehittynein grafiikan osalta. Sillä saa näyttävimmän näköistä

jälkeä vaikka on osittain kömpelö käyttää. Monet toiminnollisuudet ovat sijoittuneet hankalasti useiden vaiheiden taakse, ja tämä hidastaa työskentelyä. Käyttöliittymä on kuitenkin kehittynyt merkittävästi edelliseen versioon verrattuna.

Epic Games:llä ei ole runsaasti palveluita, mutta se on perustanut 5 miljoonan dollarin rahaston uusille innovatiivisille Unreal Engine 4 -pelimoottorilla tehdyille projekteille. Unreal Engine 4 -pelimoottorin käyttöliittymällä pääsee yrityksen verkkokauppaan, jossa myydään erilaisia käyttäjien ja Epic Gamesin tekemiä lisäosia pelimoottoriin. Kauppa on toistaiseksi vielä uusi ja sieltä ei löydy runsaasti lisäosia, mutta niiden laatu on kuitenkin tasokasta. Lisäosan tekijä saa myyntituloista 70 % ja Epic Games 30 %.



Kuva 1. Unreal Enigne 4 -pelimoottorin käyttöliittymä (Unreal Engine 2016b).

Unreal Engine on hyvä työkalu näyttävän 3D-pelin tekemiseen, mutta jos suunnittelee 2D-pelin tekemistä, niin tämä pelimoottori ei ole siihen paras mahdollinen. Visuaalista ohjelmointia on yksinkertaista käyttää, mutta se estää ymmärtämästä miksi asiat tapahtuvat niin kuin ne tapahtuvat. Jos toiminnollisuutta haluaa selvittää, niin siihen tarvitaan C++-ohjelmointikielen ymmärtämistä, jota Unreal Enginen tukee ja katsomaan mitä ohjelmointipalikat sisältävät. (Create 3D Games 2016.)

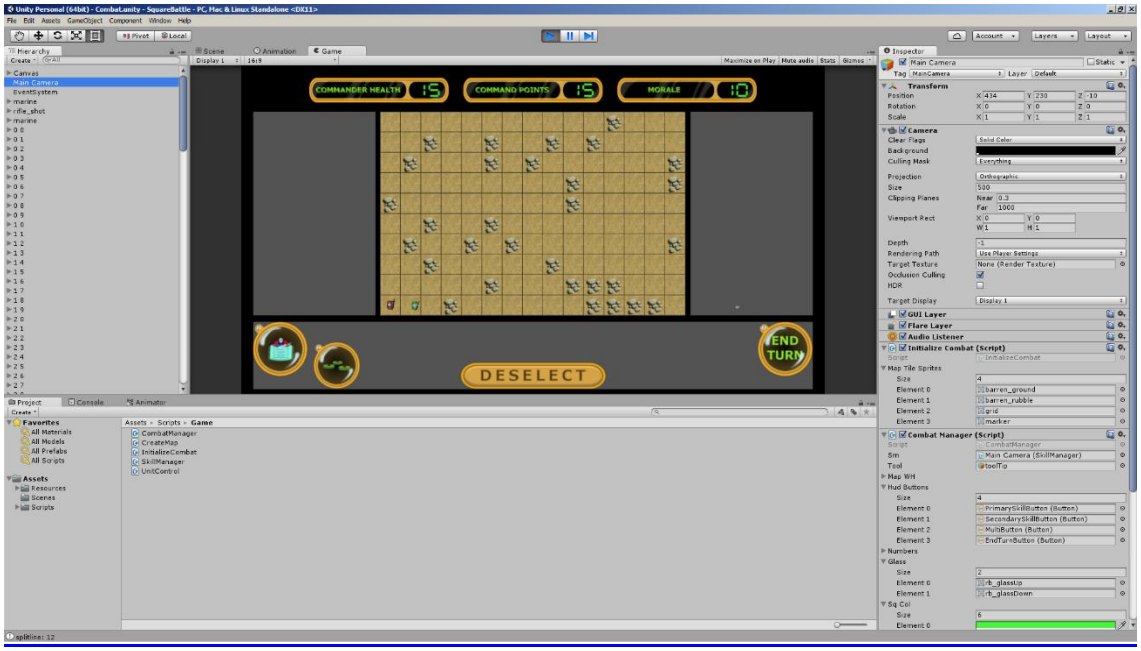
2.1.2 Unity 5

Unity 5 -pelimoottori on käytetyin pelimoottori maailmassa (Kuva 2). Sitä käyttää 47 % pelinkehittäjistä, jotka käyttävät pelimoottoreita (Wilcox 2014). Unity on ilmainen, jos käyttää henkilökohtaista versiota, mutta ammattilaisversio maksaa 75 dollaria kuukaudessa. Unity 5 ei sisällä rojaltimaksuja. Henkilökohtaisella versiolla saa tienata 100 000 dollariin asti, jonka jälkeen täytyy hankkia ammattilaisversio. (Unity 2016a.)

Unity 5 -pelimoottorilla on aktiivinen käyttäjäkunta, joka jakaa neuvoja ja ohjeistuksia internetissä. Pelimoottorin dokumentoinnissa olisi parantamisen varaa varsinkin editoripuolella, jossa dokumentointi on erittäin niukkaa (Tardes 2015, 1). Tässäkin tapauksessa käyttäjät auttavat toisiaan. Unity 5 tukee pelimoottoreista eniten erilaisia alustoja ja julkaisun tekeminen on kohtuullisen yksinkertaista Unityn puolella.

Unity 5:ssä on graafinen taso noussut runsaasti uusimman version myötä, mutta se ei silti yllä Unreal Enginen tasolle. Tämän puutteen Unity korvaa helppokäyttöisellä ja nopeasti opittavalla käyttöliittymällä. Sitä on yksinkertaista muokata hyvän editoriohjelmointirajapinnan avulla ja siksi lisäosien määrä on runsas Unity Asset Storessa, joka tarjoaa kauppapaikan käyttäjien tekemille lisäosille (Paskova 2014). Kaupassa on runsaasti erilaisia lisäosia, jotka esimerkiksi laajentavat Unity 5 -pelimoottoria tai tarjoavat valmiita 3D-malleja. Unity ottaa myyntituloista 30 % ja lisäosan tekijä saa 70 %. Kauppa sisältää yli 15 000 lisäosaa, joista monet ovat edullisia, ellei jopa ilmaisia. (Create 3D Games 2016.)

Unity 5 tukee JavaScript ja C#-ohjelmointikieliä. JavaScriptiä on muokattu ja sitä kutsutaan UnityScript-ohjelmointikieleksi. Aikaisemmissa versioissa Unity tuki myös Boo-ohjelmointikieltä, mutta se lakkautettiin vähäisten käyttäjien vuoksi. Suosituin ohjelmointikieli on C# ja ohjeistukset käsittelevät asioita sen avulla. Myös internetistä löytyy eniten ohjeita C#-ohjelmointikielelle. (Aleksandr 2014.)



Kuva 2. Unity 5 editorinäkymä 2D-pelin kehityksestä.

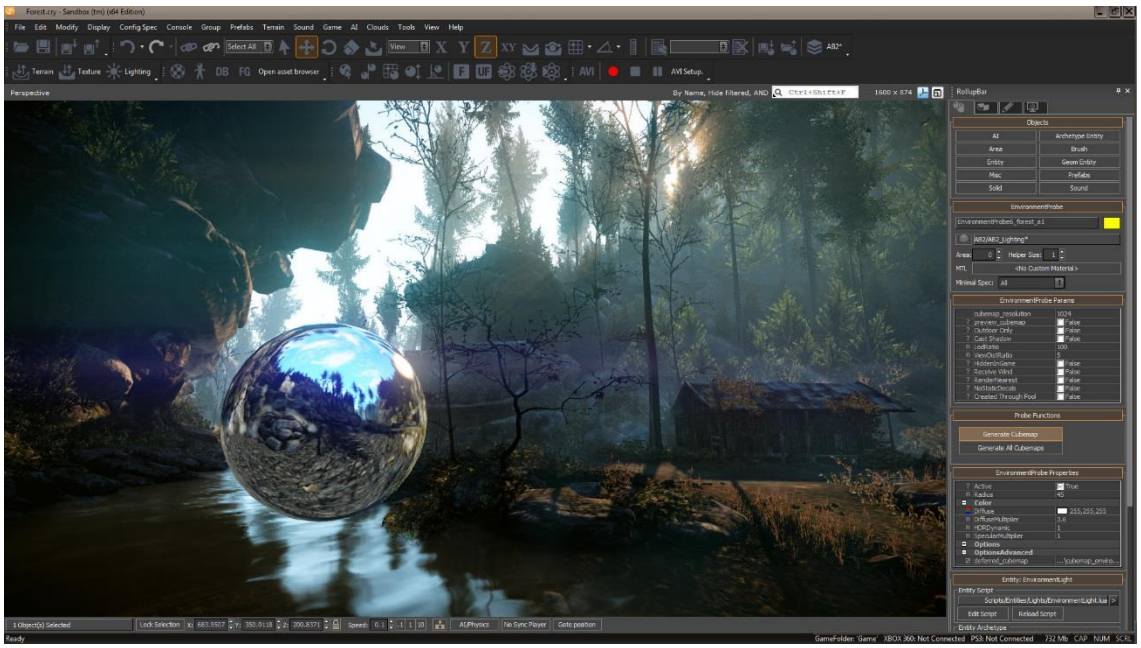
Unity 5 -pelimoottori on tehokas, ja sillä on nopeaa luoda uusia projekteja. Yksinkertainen käyttöliittymä ja lukuisat keskustelufoorumit nopeuttavat kehitystä. Lähes jokaiseen ongelmaan löytyy apua. Poikkeuksena tästä on editori puolen ohjelmointi, josta on haastavampaa löytää neuvoja. Unity sopii myös mainiosti 2D-pelien kehittämiseen ja se on helpottunut Unity 4 -version mukana tulleiden ominaisuuksien sekä työkalujen myötä.

2.1.3 CryEngine

CryTekin tekemä CryEngine 3 -pelimoottori, joka on ilmainen jos niin haluaa (Kuva 3). CryEngine sivustolla vain kehotetaan tukemaan pelimoottorin kehitystä omien varojen mukaan. Summasta on mahdollista osoittaa osa rahastoon, jolla tuetaan indie-pelinkehittäjiä jotka käyttävät CryEngineä. CryTek on ilmoittanut, että mitään ylimääräisiä rojaltimaksuja ei ole eikä niitä ole tulossa (CryEngine 2016).

Pelimoottori tukee monia alustoja ja siinä on ”Hot Update”-ominaisuus, joka automaattisesti muuttaa graafisten elementtien skaalaa alustalle sopivaksi. CryEnginellä saa visuaalisesti näyttävää jälkeä aikaiseksi ja siitä kertoo pelimoottorilla julkaistut pelit kuten

FarCry-pelisarja. Editorista löytyy työkalu graafisten- sekä pelielementtien muokkaamista varten. Työkalu toimii myös reaaliajassa, joten muutoksia voi tehdä pelin pyöriessä. Tämä auttaa kehitystyössä, kun on mahdollista nähdä muutokset saman tien. CryEngine sisältää myös tekoälyteknologiaa, kuten polunetsintä-algoritmi, joka toimii muuttuvassa ympäristössä. Animaatiopuolella löytyy automaattinen ”lipsync”-toiminto, joka käyttää äänitiedostoa animoidessaan hahmon suun liikkeitä (Giant Bomb 2014). CryEngine käyttää C++, Lua ja C#-ohjelmointikieliä.



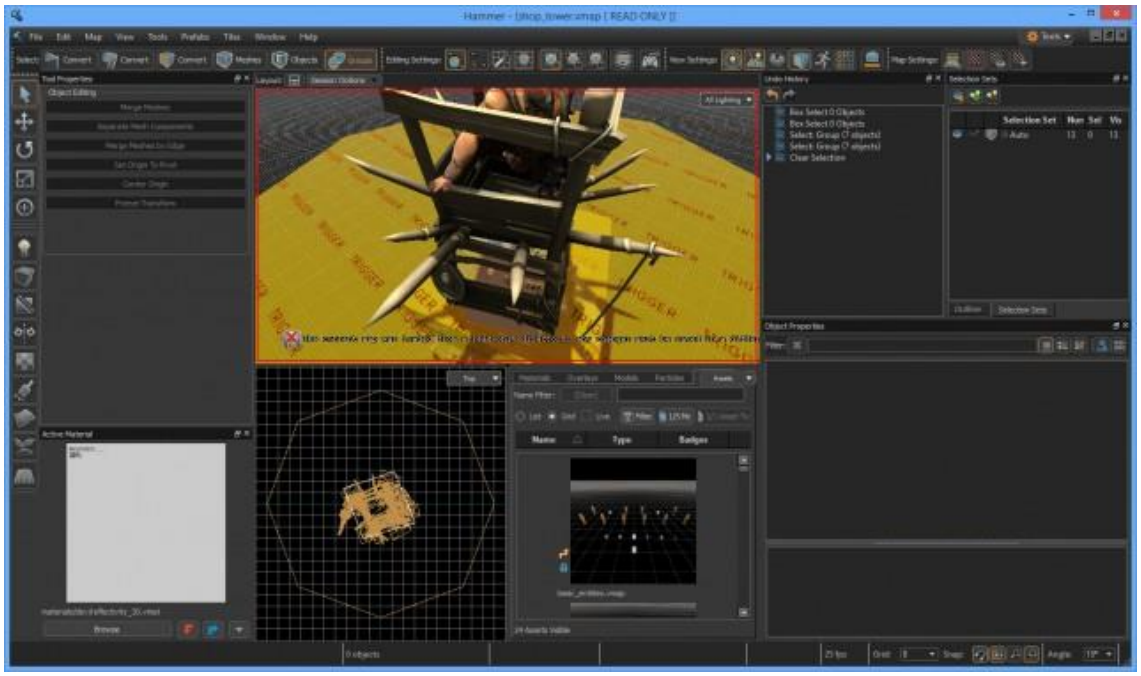
Kuva 3. CryEngine editori (ModDB 2011).

Jos tähtää näyttävän 3D-pelin toteuttamiseen, niin CryEngine 3 -pelimoottori on varteenotettava vaihtoehto. Muokattavuudesta ei tosin löytynyt tietoa, joka taas on kyseisen opinnäytetyön tekemisessä tarpeellinen ominaisuus.

2.1.4 Source 2

Source 2 on Valven pelimoottori, jota ei ole vielä julkaistu (Kuva 4). Tästä pelimoottorista löytyy vain vähän tietoa. Valve on ilmoittanut, että pelimoottorista tulee ilmainen eikä se sisällä mitään rojalti maksuja. Source 2 on 64-bittinen kun aikaisempi versio oli 32-bittinen. Pelimoottori tulee tukemaan monia eri alustoja ja se tukee directx10:iä ja

directx11:sta, sekä OpenGL rajapintoja (Anthony 2014). Valve on myös ilmoittanut, että Source 2 -pelimoottori tulee tukemaan uuden sukupolven Vulkan-rajapintaa, joka on OpenGL:n seuraaja (Chapple 2015). Source 2 -pelimoottori tulee todennäköisesti toimimaan C++-ohjelmointikielellä.



Kuva 4. Source 2 -pelimoottori (Anthony 2014).

2.2 Pelimoottorin valinta

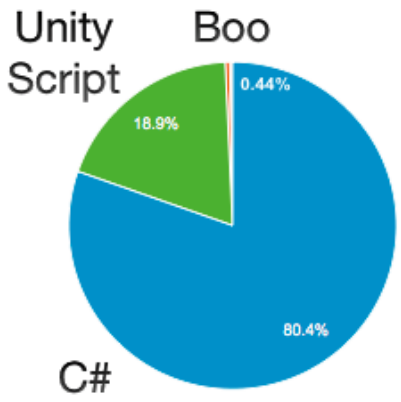
Pelimoottoreita vertailtaessa vaihtoehdoiksi jäi vain Unity ja Unreal Engine (Taulukko 1). Pelimoottorin valintaan voi vaikuttaa monikin eri asia, mutta tässä opinnäytetyössä ratkaisuun vaikutti pelimoottorin muokattavuus. Tämä ja käytettävän ohjelmointikielen huomioon ottaen valinta oli varsin helppo. Unity 5 on helposti muokattavissa ja siinä käytetään C#-ohjelmointikieltä. Muokkaamismahdollisuudet ovat lähes rajattomat ja tämän vuoksi täytyy karsia visuaalisuudesta, jonka Unreal Engine olisi tarjonnut. Unreal Enginessä käytetään visuaalista palikkaohjelmointia ja C++-ohjelmointikieltä. Muokatessa olisi täytynyt hallita C++:aa hyvin, ja tämän vuoksi Unreal Engine jäi toiseksi. CryEngine- ja Source-pelimoottoreiden muokattavuudesta ei löytynyt tietoa, joten nämä pelimoottorit eivät soveltuneet käyttöön.

Taulukko 1. Neljä eri pelimoottoria vertailussa.

Pelimoottori	Hinta	Ohjelmointikielet	Muokattavuus
Unreal Engine 4	ilmainen	C++, Visual scripting language	kohtalainen
Unity 5	ilmainen	C#, UnityScript	hyvä
CryEngine 3	ilmainen	C++, Lua, C#	tuntematon
Source 2	ilmainen	C++	tuntematon

2.3 Unity 5 -pelimoottori

Unity 5 -pelimoottori on hyvä työkalu pelien, visuaalisille installaatioiden sekä demojen tekemiseen. Tämä kehitystyökalu on joustava, ja se on muokattavissa. Ohjelmoinnissa voidaan käyttää C#- ja UnityScript-ohjelmointikieliä. UnityScript on muokattu JavaScript-ohjelmointikielestä, jotta se sopisi paremmin Unity 5:n ohjelmointikieleksi. Aikaisemmissa Unityn versioissa oli mukana Boo-ohjelmointikieli, mutta vähäisten käyttäjien vuoksi sen tuki lakkautettiin (Aleksandr 2014). Kuvassa 5 näkyy eri ohjelmointikielten käyttäjien prosentuaaliset osuudet.

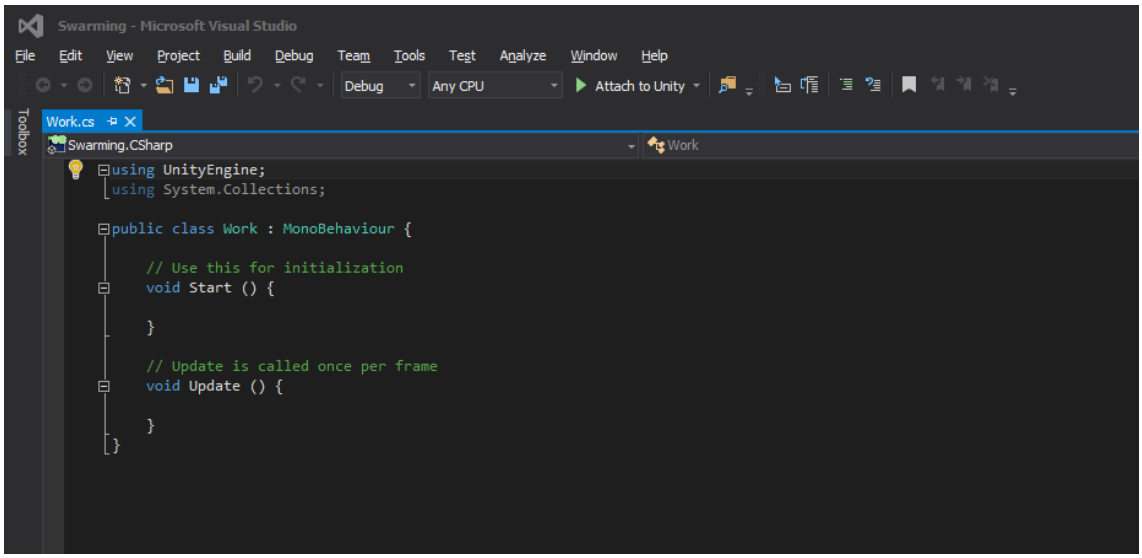


Kuva 5. Unity 5 -pelimoottorissa käytetyt ohjelmointikielet (Aleksandr 2014).

Unity 5 -pelimoottorin mukana tulee IDE nimeltään Monodevelop, joka on tarkoitettu ohjelmointiin. Tämän voi halutessaan vaihtaa esimerkiksi Microsoft Visual Studio -kehitystyökaluun. Visual Studion saa yhdistettyä Unity 5 -pelimoottoriin, joka mahdollistaa virheentarkastustoiminnan käytön. Tämä helpottaa huomattavasti virheiden etsimistä projektista. Virheiden paikannus ei tarkoita sitä, että koodista etsitään

vain syntaksivirheitä vaan sillä paikannetaan myös logiikka-virheitä (Thorn 2015, 53). Yleensä koodi liitetään editoripuolella johonkin Scene-ikkunassa olevaan peliobjektiin. Tästä on tietysti poikkeuksia, kuten esimerkiksi staattiset luokat.

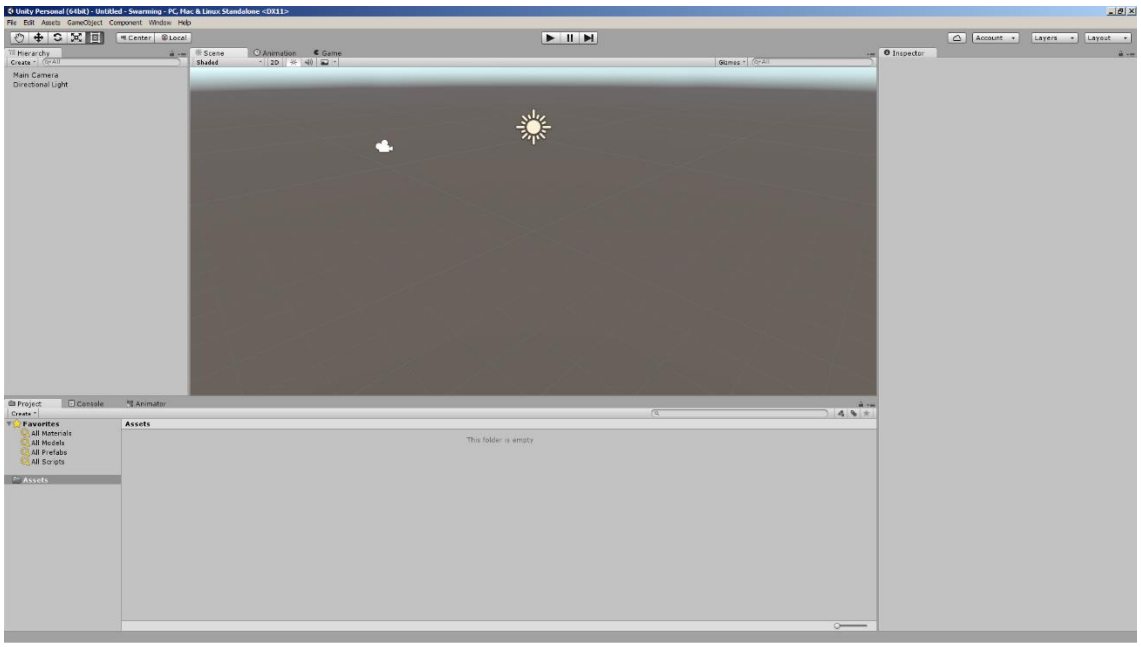
Komentosarjatiedosto sisältää linkit Unityn kirjastoihin ja luokan, joka periytyy MonoBehavior-luokasta (Kuva 6). Tämä mahdollistaa Unityn komentojen käytön tehokkaasti. Luokkaan luodaan myös Start- ja Update-metodit, jotka viittaavat pelinkäynnistykseen ja päivitykseen. Huomioitavaa on, että Update-metodia kutsutaan ajon aikana 60 kertaa sekunnissa.



Kuva 6. Unity 5 -pelimoottorin komentosarjatiedoston sisältö Visual Studiossa.

Unity 5:n editoria on aluksi vaikea hallita, mutta jo lyhyen käytön jälkeen sen toiminta alkaa hahmottua. Editorista muodostuu helppokäyttöinen ja kätevä työkalu pelinkehittäjälle, kun perustoiminnot ovat selvillä. Editorinäkymä avautuu aina ensin, kun aloitetaan käyttämään Unity 5 -pelimoottoria. Keskellä on Scene-ikkuna, johon lisätään elementtejä, jotka tulevat peliin. Vasemmalla puolella on Scene-ikkunan hierarkia, johon lisätyt peliobjektit tulevat näkyviin. Alalaidassa on projektinäkymä, jossa näkyy kaikki projektikansiossa olevat tiedostot. Välilehdissä on myös konsoli virheilmoituksia varten sekä tässä tapauksessa animaattori animaatioiden ohjausta varten. Oikeassa laidassa on Inspector-ikkuna, johon tulee näkyviin valitun peliobjektin tiedot sekä siihen liitetyt komponentit. Editorin asettelua voi muuttaa mielensä mukaan ja kuvassa 7 oleva näkymä on vain yksi lukemattomista mahdollisuuksista. Alueita voi irrottaa eri ikkunoiksi

ja on suositeltavaa, että pelinäkymä irrotetaan omaksi ikkunaksi kun peliä kehitetään sekä testataan. Tosin tässä tapauksessa on myös suositeltavaa, että tietokoneessa on kaksi näyttöä. Muuten pelinäkymä-ikkunan irrotus on turhaa.



Kuva 7. Unity 5 -editorinäkymä.

Unity 5 -pelimoottorin sisältä löytyy animaattori ja animointityökalut. Näiden avulla pystytään tekemään 2D- ja 3D-animaatioita kohtuullisen vaivattomasti. Yksinkertainen 2D-animointi tapahtuu pudottamalla kuvat animointi-aikajanalle, jonka jälkeen täytyy vain hienosäätää ajoitukset. 3D-mallinnusohjelmista voi tuoda animaatiot 3D-mallin mukana helpottaen animointia.

Animointia on helpotettu luomalla Mecanim-systeemi Unity 5 -pelimoottoriin. Mecanim on animaattori-komponentti, jonka avulla voidaan määrittää peliobjektille erilaisia tiloja animointien kera. Tämä tarkoittaa sitä, että on esimerkiksi kävely- ja juoksu-tilat, joiden välissä on siirtymät. Siirtymien avulla voidaan vaihtaa animaatioita. Animaattori sisältää myös animaatioiden sekoituksen, jonka avulla kyetään yhdistämään vaikkapa juoksu- ja heitto-animaatiot yhteen.

Pelimoottorissa on myös fysiikkamoottori, joka on helppokäyttöinen ja muokattavissa. Fysiikan voi myös halutessaan kytkeä pois päältä. Fysiikan saa käyttöön liittämällä ”Rigidbody”-komponentti ja ”Collider”-komponentti haluttuun peliobjektiin. Tämän

jälkeen pelin käynnistyessä objekti alkaa pudota jos alapuolella ei ole mitään. ”Collider”-komponentti rekisteröi törmäykset muiden ”Collider”-komponenttien kanssa, jolloin Unityn fysiikkamoottori voi toimia sen mukaan.

3 TYÖKALUN OHJELMOINTI UNITY 5 -PELIMOOTTORIIN

Unity 5 -pelimoottori on helposti laajennettavissa, ja se on suunniteltu niin, että kuka tahansa voi halutessaan muokata pelimoottoria. Unity 5 -pelimoottorissa on erinomainen ohjelmointirajapinta editorin muokkaamista varten, mutta dokumentaatio ei ole parhaimmasta päästä (Tadres 2015, 1). Ohjelmointirajapinta toimii kommunikaatiovälineenä, jonka avulla ohjelmoijan tekemät komentosarjat keskustelevat Unity-pelimoottorin kanssa. Editoriohjelmointirajapinta koostuu useista komennoista, joita käyttämällä ohjelmoija voi tehdä muutoksia Unity-pelimoottorin editoriin sekä ohjata sitä haluamallaan tavalla. Keskustelupalstoilta kuitenkin löytyy paljon apua muilta käyttäjiltä, jotka ovat selvittäneet ohjelmointirajapinnan eri ominaisuuksia. Pelimoottorin käyttäjät ovat luoneet tuhansia laajennuksia, joista suurin osa löytyy Unity asset store -sivustolta (Paskova 2014). Laajennuksista osa on ilmaisia ja toiset taas maksullisia riippuen työmäärästä ja laadusta. Yleensä työkaluja tehdään silloin, kun tarvitaan jotain toiminnollisuutta jota ei pelimoottorista vielä löydy, tai kun pelinkehitysprosessi tarvitsee avustustoimintoja.

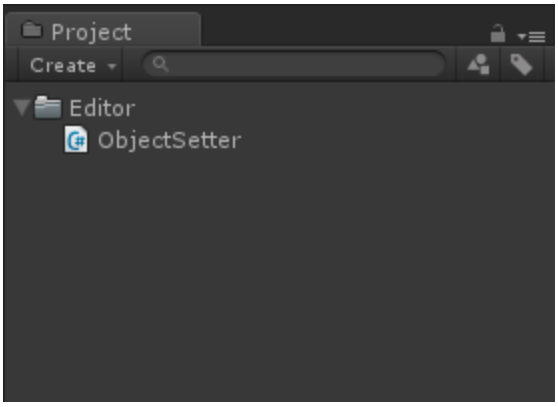
3.1 Suunnittelu ja eri lähestymistavat

Työkalun tekemiseen tartutaan yleensä silloin, kun tarvitaan jotain, mitä pelimoottorista ei vielä löydy. Erilaisia lähestymistapoja on useita. Jokainen ohjelmoija luo oman lähestymistavan, mutta yleisesti ottaen työkalu kannattaisi suunnitella sillä tavalla että sitä voidaan käyttää useissa eri projekteissa. Tämä tarkoittaa sitä, että yleensä ei kannata kuluttaa aikaa, jos työkalu toimii vain tietyissä tilanteessa. Poikkeuksena on tietysti se, että työkalu on välttämättömyys pelinkehitykselle tai se nopeuttaa kehitystyötä enemmän kuin sen ohjelmointiin kuluu aikaa. Esimerkiksi testaukseen liittyvät työkalut voivat olla tarpeellisia. Näissä tapauksissa voidaan tehdä työkalu, joka visualisoi muuten näkymättömiä toimintoja pelissä. Työkalun avulla voidaan myös poistaa ohjelmoinnin tarvetta erilaisissa tilanteissa. Esimerkiksi voisi ottaa viholliseditori-käyttöliittymän. Sen avulla voidaan vaikuttaa vihollisen ominaisuuksiin suoraan editoritilassa. Käyttöliittymän avulla vältytään siltä, että tarvitsisi muuttaa koodia (Paju 2014, 27). Tämä nopeuttaisi pelinkehitystä huomattavasti.

Suunnitellessa työkalua kannattaa selvittää kuinka siitä voidaan tehdä yleishyödyllinen, sillä jos työkalusta saadaan tarpeeksi hyvä niin sen voi laittaa Unityn kauppaan myyntiin. Kannattaa myös varmistaa, että Unity Asset Storesta ei löydy samanlaista laajennusta. Mahdollisuuksien mukaan on hyvä tutkia muita samankaltaisia laajennuksia.

3.2 Työkalun ohjelmointi

Editorin laajentaminen tapahtuu luomalla komentosarjatiedostoja. Poikkeuksena on se, että kun tavallisesti komentosarjatiedosto sijoitetaan peliobjektiin, niin editorikomentosarjatiedostot sijoitetaan Editor-kansion sisään. Kuvassa 8 on Unityn editorin projekti-ikkuna ja Editor-kansio. Unity hakee automaattisesti komentosarjatiedostot Editor-kansiosta, jolloin käyttäjä voi halutessaan käyttää niitä pelinkehityksessä.



Kuva 8. Projekti-ikkunassa Editor-kansio (Elwin 2014).

Toinen huomioitava asia editorikomentosarjoja tehdessä on käytettävän kirjaston vaihtaminen. Normaalisti uudessa komentosarjatiedostossa on UnityEngine-kirjasto, mutta editorikomentosarjatiedostoa tehdessä se täytyy vaihtaa UnityEditor-kirjastoksi. Komentosarjatiedostossa oleva luokka ei voi enää tässä vaiheessa periytyä MonoBehaviour-luokasta. Perittävää luokkaa valittaessa täytyy tietää mitä haluaa muokata. Editor-luokasta periytymistä käytetään silloin kun halutaan lisätä komponentti, joka näkyy Inspector-ikkunassa. EditorWindow-luokan avulla saadaan komponentille aikaiseksi oma ikkuna. Editorikomentosarjassa ei käytetä perinteisiä Update- ja Start-metodeita, vaan siellä käytetään esimerkiksi OnEnable- sekä

OnInspectorGUI-metodeja. Niitä kutsutaan viesti-metodeiksi, joita Unity kutsuu tietyissä tilanteissa. OnEnable- ja OnInspectorGUI-metodien lisäksi on myös OnDisable- ja OnDestroy-metodit. OnEnable-metodia kutsutaan aina kun valitaan objekti, joka sisältää komponentin ja OnDisable-metodia kutsutaan silloin kun valinta poistuu. OnEnable-metodiin kannattaa sijoittaa kaikki alustuskomennot ja niiden lisäksi komentosarjatiedosto kohdistetaan target-muuttujan avulla toiseen komentosarjatiedostoon, joka on tavallinen UnityEngine-kirjastoa käyttävä komentosarjatiedosto (Tardes 2015, 56-57). Jos editoritilassa halutaan päivittää komponentteja, niin silloin täytyy kutsua erikseen update-komentoa kuten kuvassa 9 viimeisellä rivillä. Editorikomentosarja on yleensä yhteydessä tavalliseen komentosarjatiedostoon, joka toteuttaa komponentin muutokset Scene-ikkunassa olevaan peliobjektiin.

```
void OnEnable()
{

    _target = (ModifyTool)target;

    _target.GetFontsToList();

    Debug.Log("plugin created");

    EditorApplication.update += Update;
}
```

Kuva 9. Editorikometosarjan OnEnable-metodi.

OnInspectorGUI-metodin sisälle sijoitetaan komentosarjat, jotka luovat komponentin elementit Inspector-ikkunaan. Elementteinä voi esimerkiksi olla liukusäädin, tekstikenttä tai värinvalitsin. Erilaisia elementtejä ja komentoja on monia, joilla pystytään muokkaamaan komponentin ulkoasua.

Primitiiviobjektit löytyvät Gameobject-valikon alta ja niihin lukeutuvat erilaiset 3D-muodot kuten kuutio, pallo tai tyhjä peliobjekti. Niiden tarkoitus on olla tilapäisiä objekteja, kun mekaniikkaa ohjelmoidaan tai silloin kun tehdään prototyyppiä. Tällä tavoin pelinkehitys ei hidastu, kun ei tarvitse odottaa uusien graafisten elementtien valmistumista. Editoriohjelmoinnilla on myös mahdollista lisätä omia primitiiviobjekteja editoriin. Tämä tapahtuu luomalla komentosarjatiedosto Editor-kansioon. Tiedosto käyttää myös UnityEditor-kirjastoa kuten komponentin komentosarjatiedostossa, mutta sen ei tarvitse

periytyä Editor-luokasta vaan MonoBehaviour-luokasta. Komentosarjatiedostoon tehdään metodit, jotka luovat primitiiviobjektin Scene-ikkunaan (Kuva 10). Metodeita tarvitaan ainakin kaksi, sillä on huomioitava se mahdollisuus, että käyttäjä luo primitiiviobjektin toisen peliobjektin lapseksi. Tässä tapauksessa metodin täytyy tarkastaa onko peliobjekti valittuna, jos peliobjekti on valittuna, niin silloin primitiiviobjekti luodaan tämän lapseksi.

```
using UnityEngine;
using UnityEditor;
using System.Collections;

public class TextTool : MonoBehaviour {

    //instantiates new gameobject to root
    [MenuItem("GameObject/3D Object/3D modifiable text")]
    public static void Make3DText()
    { CreateTextTool.Create(); }


    //instantiates new gameobject as child of selected gameobject
    [MenuItem("GameObject/3D Object/3D modifiable text")]
    public static void Make3DText_child()
    {
        GameObject go = CreateTextTool.Create();
        if(Selection.activeGameObject)
        {
            go.gameObject.transform.parent = Selection.activeGameObject.transform;
        }
    }

}
```

Kuva 10. Näkymä komentosarjatiedostosta, jossa luodaan primitiiviobjekti.

3.3 Laajennuksen tekemiseen liittyvät huomiot

Työkalun ohjelmointi on erilaista kuin muiden komentosarjojen ohjelmointi. Huomioitavia seikkoja on useita. Näistä ensimmäinen on päivitykseen liittyvät asiat, sillä editoritilassa komentosarjatiedostojen Update-metodia ei ilman komentoa ajeta. Tähän on ratkaisuna komento, jonka avulla Update-metodi ajetaan normaaliin tapaan (Ohjelmakoodi 1). Rivin lopussa oleva Update-sana viittaa ohjelmoijan luomaan Update-funktioon. EditorApplication.update-funktiokutsu takaa sen, että Update-funktiota kutsutaan toistuvasti pelimoottorin olessa aktiivisena.

Ohjelmakoodi 1. Update-komento

```
EditorApplication.update += Update;
```

Toinen huomioitava asia liittyy komponentin tai peliobjektin aktivointiin. Nämä täytyy tarkastaa koodissa ja niitä varten täytyy ohjelmoida metodit. Unityssä komponentin voi kytkeä pois päältä tai peliobjektin voi kokonaan sammuttaa, jolloin komponentin komentosarjatiedoston täytyy huomioida tämä tilanne. Ohjelmoijan täytyy ratkaista mitä näissä tapauksissa tapahtuu. Yleensä kun komponentti kytketään pois päältä, niin komponentin tekemät muutokset täytyisi palauttaa alkuasetuksille. Update-metodin sisällä täytyy toistamiseen tarkastaa komponentin tila (Ohjelmakoodi 2). Tätä varten on EditorUtility.GetObjectEnabled-metodi, joka palauttaa ykkösen kun komponentti on aktivoitu ja nollan kun komponentti on pois päältä.

Ohjelmakoodi 2. Komponentin tilan tarkistus.

```
void Update()
    {
        if (isDone)
        {
            EditorApplication.update -= Update;
        }
        else
        {
            Status();
        }
    }

void Status()
    {
        i = EditorUtility.GetObjectEnabled(_target);
        _target.i = i;
        if (!(i == old_i))
        {
            old_i = i;
            _target.IsEnabled(i);

        }
    }
```

Työkalun ohjelmoimisessa on myös huomioitava tilanne, jossa käyttäjä vaihtaa peliobjektia. Tässä tilanteessa kun keskitys katoaa komponentista, niin komentosarja kutsuu OnDestroy-metodia, jota kutsutaan myös silloin kun peliobjekti tuhotaan. Jossain tilanteissa tämä tapahtuma voi aiheuttaa hankaluuksia ja jos asiasta ei tiedä niin se voi aiheuttaa hämmennystä.

Komponentin komentosarjatiedosto käydään läpi aina kun komponentin sisältävä peliobjekti valitaan. Tämä täytyy huomioida komponenttia suunniteltaessa, jottei tule vahin-

gossa alustettua muuttujia yhä uudestaan ja uudestaan. Tämä johtuu siitä, että komponentin aktivointiin liittyvät koodirivit täytyy sijoittaa OnEnable-metodin sisälle (Ohjelmakoodi 3). OnEnable-metodia kutsutaan aina kun komponentti aktivoituu.

Ohjelmakoodi 3. OnEnable-metodi.

```
using UnityEngine;
using UnityEditor;
using System.Collections;

[CustomEditor(typeof(ModifyTool))]
public class Editor_ModifyTool : Editor {

    ModifyTool _target;

    void OnEnable()
    {

        _target = (ModifyTool)target;
        Debug.Log("plugin created");

        EditorApplication.update += Update;
    }

}
```

Editorissa valmiina oleva Undo-toiminto täytyy myös huomioida, jos haluaa että käyttäjät voivat halutessaan peruuttaa toimiaan. Tätä varten on Undo.RecordObject-metodi, jota käyttämällä saadaan muutosten historia käyttöön jolloin käyttäjä voi peruuttaa toimintojaan (Unity 2016b). Toinen tapa on käyttää SerializedObject-luokkaa, joka hoitaa automaattisesti peruutukset (Unity 2016c). Molempia tapoja käytetään UnityEditor-kirjaston ja Editor-luokan kanssa. Laajennus ilman peruutusmahdollisuutta on vajavainen. Tätä mahdollisuutta kuuluisi aina käyttää, koska käyttäjät olettavat sen olevan mahdollista.

4 LAAJENNUS 3D-TEKSTIN TUOTTAMISEKSI JA MUOKKAAMISEKSI UNITY 5 -PELIMOOTTORIIN

Laajennus 3D-tekstin tuottamiseksi ja muokkaamiseksi Unity 5 -pelimoottoriin oli opinnäytetyön käytännönosuuden aiheena. Tämä kuitenkin muuttui asiakkaan kanssa keskustelun jälkeen, jolloin sovittiin että laajennuksen tarvitsee toimia vain numeroilla sekä desibelilyhenteen merkeillä. Laajennuksen täytyi pystyä muokkaamaan malleja mahdollisimman monipuolisella tavalla, jotta useita mallikokoelmia ei tarvittaisi. 3D-mallien muokkaus toteutettaisiin muuttamalla mallin verteksien sijaintia sekä käyttämällä Unityn animaattorin ”blendtree”-ominaisuutta.

Animaattorin ”blendtree”-ominaisuus on tehty animaatioiden sekoittamiseen. Opinnäytetyön kohdalla sitä käytettiin kolmen animaation sekoittamiseen. Animaatiot koostuivat kahden kuvan pituisista animaatioista, joissa ei ollut liikettä. Ne olivat ainoastaan eri muotoja ja ”blendtree”-ominaisuudella niitä pystyi yhdistelemään.

Opinnäytetyön nimeksi tuli parametrinen 3D-mallikokoelman muokkaustyökalu, joka on suunniteltu asiakkaan sovellusta silmällä pitäen. Laajennus rajattiin käsittelemään vain numeroita, jotta pysyttiin projektin aikataulussa. Työskentely alkoi muiden aiheeseen liittyvien laajennuksien tarkastelulla ja prototyyppien tekemisellä sekä Unityn editoriohjelmoinnin opettelulla. Alkuvaiheiden jälkeen alkoi laajennuksen toteuttaminen.

4.1 Muut 3D-tekstin muokkaus laajennukset

Ennen opinnäytetyön suunnittelun aloittamista tutustuttiin kahteen maksulliseen Unity 5 -pelimoottorin laajennukseen, jotka käsittelivät 3D-tekstin muokkausta ja generointia. Unity Asset Storesta löytyy monenlaisia laajennuksia tekstinmuokkausta varten, mutta Flying 3DText- ja Realtime 3D Text -laajennukset olivat lähimpänä opinnäytetyötä. Kuitenkaan kumpikaan näistä laajennuksista ei ollut sellainen, jota asiakkaan toiveiden pohjalta suunniteltiin. Tutkimuksesta oli kuitenkin apua ratkaisujen tekemisessä ja se selvensi myös sen, että samankaltaista laajennusta ei vielä Unity Asset Storesta löydy.

4.1.1 FlyingText3D

FlyingText3D-laajennus, jonka mainoskuva on kuvassa 11, generoi tietokoneeseen asennetusta ja käyttäjän valitsemasta TrueType-fonteista 3D-tekstin (Unity Asset Store 2016a).

Kuva 11. FlyingText3D (Unity Asset Store 2016a).

Generoitujen mallien topologia ei ole tarpeeksi laadukas ja tutkimusten pohjalta tämän kaltaista systeemiä ei voi hyväksikäyttää opinnäytetyössä. Laajennuksessa on tekstin asettelua varten parametreja, kuten kirjainten ja rivien välin etäisyyden määritys. Tekstin sijainnin määrittävät parametrit olivat mielestäni turhia, sillä peliobjektissa on jo Transform-komponentti, joka määrittää peliobjektin sijainnin. Jokaiselle 3D-kirjaimelle voi halutessaan asettaa Rigidbody-komponentin ja fysiikkamateriaalin. Näiden avulla kirjaimet voidaan laittaa toimimaan fyysisessä ympäristössä realistisesti. Laajennuksessa on myös mahdollista asettaa materiaalit kirjaimille ja niiden reunoille sekä vaihtaa väriä. FlyingText3D-laajennusta tutkittaessa havaittiin se, että laajennus ei varsinaisesti antanut vastauksia opinnäytetyön suunnitteluun. Värin- ja materiaalinvaihto ominaisuudet olivat kuitenkin hyödyllisiä, sillä ne sisältyvät opinnäytetyöhön.

4.1.2 Realtime 3D Text

Realtime 3D Text -laajennus ei varsinaisesti ole vain 3D-tekstin muokkain vaan sen tarkoitus on auttaa kehittäjiä luomaan 3D-valikkoja (Kuva 12).

Kuva 12. Realtime 3D Text (Unity Asset Store 2016b).

Tässä laajennuksessa on mukana 3D-tekstin muokkain, joka on hyvin alkeellinen muokkausominaisuuksiltaan. Valikko-työkalun lisäksi laajennus kykenee muuttamaan minkä tahansa asennetun fontti-kirjaston 3D-kirjaimiksi. Automaattinen 3D-mallien luominen fontti-kirjastosta on tehokas, mutta mallien topologia ei ole tarpeeksi laadukasta. 3D-kirjaimien syvyyttä, kokoa ja tarkkuutta voi muuttaa, mutta siinä on kaikki mitä voi kirjaimille tehdä. Laajennus ei oikeastaan tarjoa mitään kunnon välineitä 3D-mallien muokkaamiseen ja tämän vuoksi sillä ei ollut juurikaan tutkimusarvoa. 3D-kirjainten generoiminen fontti-kokoelmasta kuitenkin herätti pohdintaa, mutta lopulta päädyttiin siihen tulokseen että generoitujen kirjainten topologia oli liian heikko.

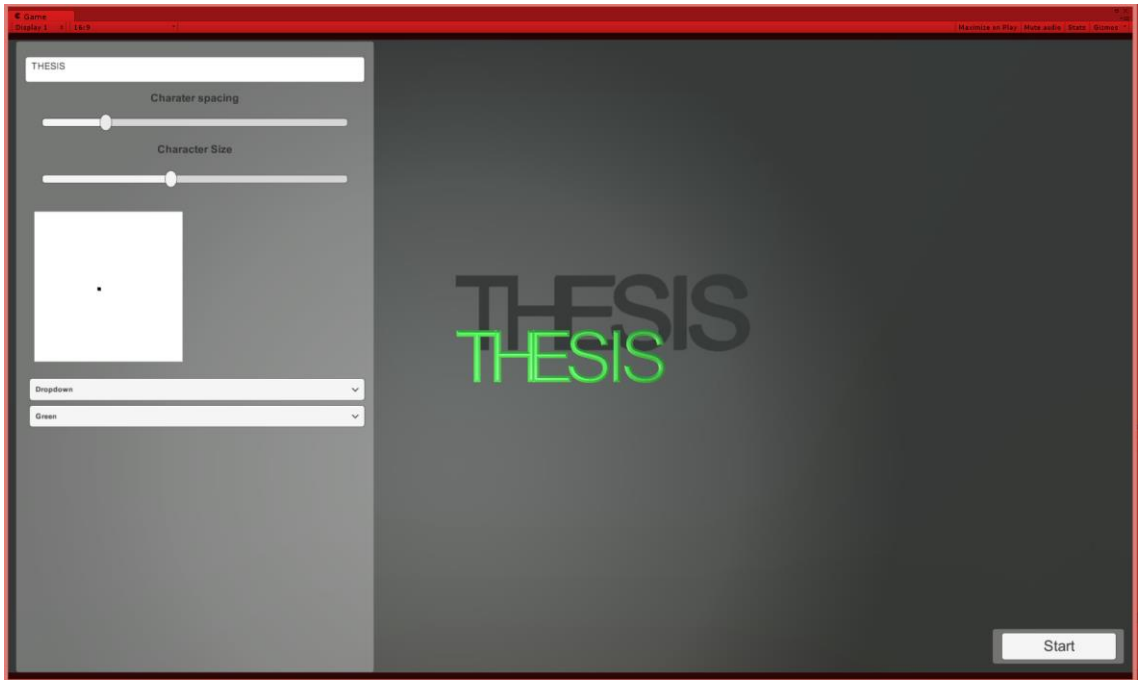
4.2 Laajennuksen suunnittelu

Laajennuksen suunnittelu alkoi asiakkaan toiveiden pohjalta. Aluksi täytyi tutustua editoriohjelmointiin, joka poikkeaa paljon Unityn peliohjelmoinnista. Tutustumisen tukena

suunniteltiin ja toteutettiin kaksi prototyyppiä, joista jälkimmäistä käytettiin pohjana laajennusta tehdessä. Suunnitelman tekeminen oli osittain progressiivista, joka laajeni prototyyppejä tehdessä. Parhaat ratkaisut täytyi löytää tekemällä kokeita ja tutkimalla asiaa, sillä kokemusta asiasta ei ollut. Suurimpia haasteita oli 3D-mallinnusohjelmasta tuotujen mallien muokkaamista verteksejä manipuloimalla ja tähän tutkimukseen kului useita viikkoja. Tutkimusten jälkeen havaittiin, että lihavoitu- ja pääteviiva-muotojen toteuttaminen verteksejä liikuttamalla olisi haasteellista sekä aikaa vievää.

4.2.1 Ensimmäinen prototyyppi

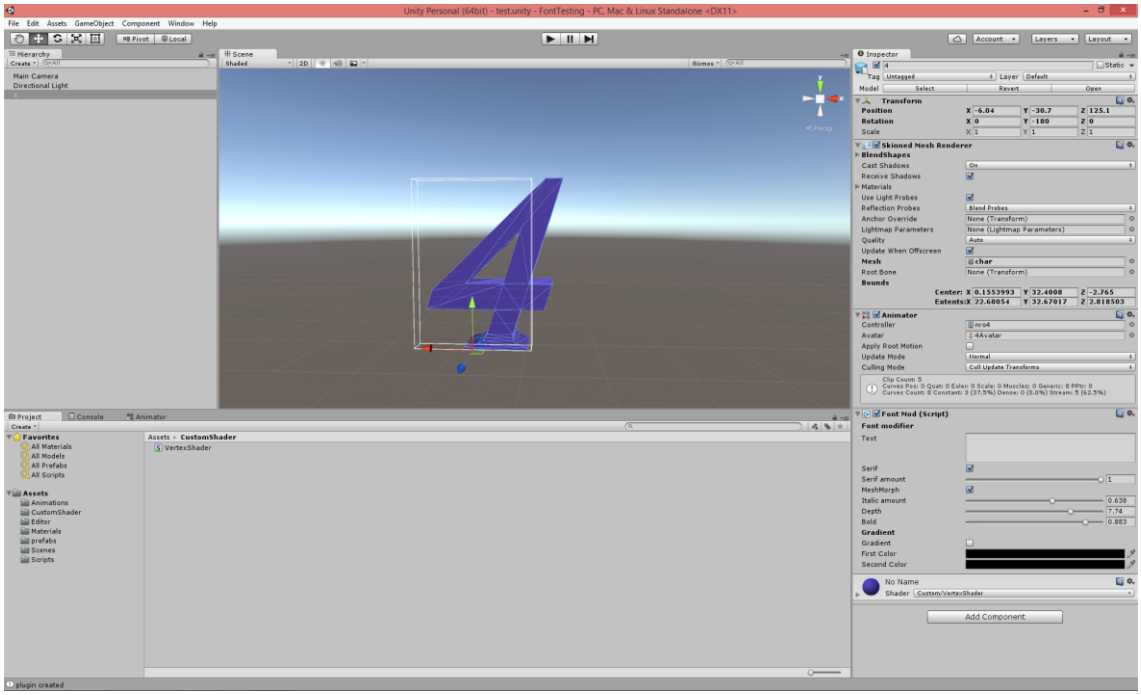
Ensimmäinen prototyyppi ei varsinaisesti ollut Unity 5 -pelimoottorin laajennus vaan tavallinen Unity-ohjelma, joka täytyi käynnistää toimiakseen (Kuva 13). Tässä vaiheessa asiakkaalta oli tullut vain vähän informaatiota projektista. Prototyypissä tutkittiin erilaisia mahdollisuuksia luoda 3D-kirjaimia käyttäjän syötteestä sekä kirjaimiin kohdistuvia tehoste efektejä ja animaatioita. Prototyyppiin ohjelmoitiin yksinkertainen käyttöliittymä, jossa käyttäjä kykenee muokkaamaan kirjainten väliä, kokoa ja lampun sijaintia sekä valita materiaalin 3D-malleille. Lisäksi yksinkertaisia animaatioita pystyi valitsemaan ”dropdown”-valikosta. Tästä prototyypistä jäi käyttäjän syötteen kääntäminen 3D-malleiksi. Syöte käytiin läpi kirjain kerrallaan ja se muutettiin ascii-arvoksi. Tämän jälkeen arvoa käytettiin 3D-mallien taulukon indeksinä, josta saatiin luotua oikea 3D-malli pelinäkymään.



Kuva 13. Ensimmäinen prototyyppi.

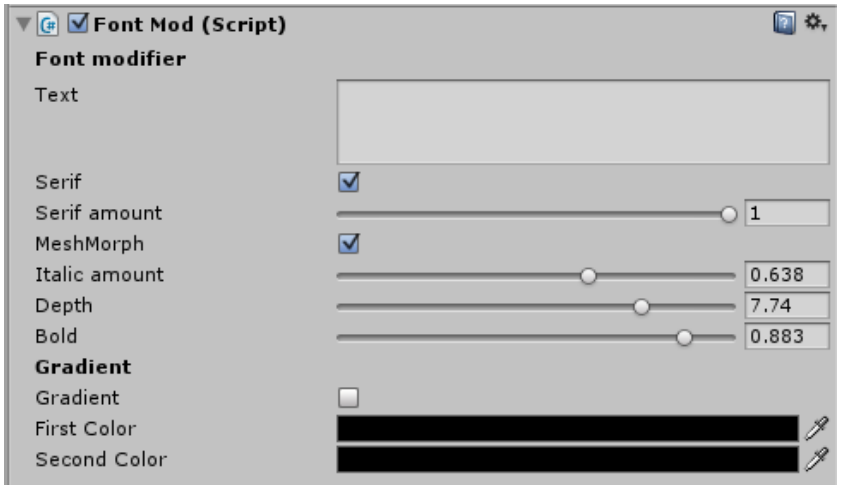
4.2.2 Toinen prototyyppi

Toinen prototyyppi oli laajennus (Kuva 14). Prototyypin tarkoituksena oli opetella Unityn editoriohjelmointia. Sitä varten mallinnettiin yksi 3D-numero, jolla oli kolme muotoa. Normaalimuodon lisäksi mallista tehtiin pääteviiva- ja lihavoitu-muodot. Tämän jälkeen käytettiin 3D-mallinnusohjelman ”morpher”-työkalua mallin muodon muuttamiseen ja sen avulla saatiin kuusi kuvainen animaatio. Animaation avulla Unity 5 -pelimoottorissa koottiin animaattori, jossa käytettiin ”blendtree”-ominaisuutta.



Kuva 14. Toinen prototyyppi.

Tähän prototyyppiin ohjelmoitiin komponentti, joka liitetään peliobjektiin. Peliobjektissa pitää olla 3D-malli ja animaattori, jotta komponentti toimii oikein. Komponentti sisältää mallin muuttamiseen tarvittavia käyttöliittymä-elementtejä (Kuva 15). Prototyypillä pystyy muuttamaan mallin syvyyttä, lihavointia, kaltevuutta sekä pääteviivan kokoa.



Kuva 15. Prototyypin komponentti inspector-ikkunassa.

Tämän prototyypin pohjalta alkoi varsinaisen laajennuksen ohjelmoiminen. Suurin osa koodista pystyttiin uusiokäyttämään, mutta sitä täytyi muokata paremmaksi. Eniten vanha koodi muuttui, kun mallien muutokset täytyi tapahtua kaikkiin komponentin omaavan peliobjektin alla oleviin 3D-mallieihin.

4.2.3 Parametrisen 3D-mallikokoelman muokkaustyökalun suunnittelu

Suunnitelman perustana oli asiakkaan tavoite, joka oli laajennus Unity 5 -pelimoottoriin. Laajennus oli 3D-mallilkokoelman muokkaustyökalu, jota ohjattiin parametrien avulla. Mallikokoelma koostui numeroiden ja desibelilyhenteen 3D-malleista. Asiakas halusi mahdollisimman monipuolisen muokkaimen, jotta 3D-mallikokoelmia ei tarvittaisi montaa. Laajennuksen täytyi kyetä muuttamaan mallien muotoa lihavoimalla, taivuttamalla, paksuntamalla tai lisäämällä niihin pääteviivat. Malleihin täytyi myös pystyä sijoittamaan tekstuurit ja valitsemaan värit. 3D-mallit suunniteltiin siten, että jokaisessa mallissa on kolme materiaalia. Ne kohdistuivat mallin sivuun, keskukseen ja ääriviivaan. Pääteviiva- ja lihavoitu-muotojen aikaan saamiseksi päätyttiin käyttämään Unityn animaattoria, ja sen ”blendtree”-ominaisuutta. Tämä ominaisuus mahdollistaa 3D-mallin animaatioiden sekoituksen. Kahden kuvan pituisia animaatioita sekoitettiin keskenään, jonka avulla saatiin haluttu lopputulos aikaiseksi. Ongelmakohtana oli animaattori-komponentin toimiminen editoritilassa, jonka tutkimiseen kului aikaa. Animaattorin parametrien muutoksia ei aluksi saatu päivittymään. Tämä ratkaistiin sijoittamalla animaattorin päivitys kutsu komponentin Update-toimintoon, jolloin animaattorin päivitystä kutsuttiin toistuvasti. Sillä oli vain vähäinen vaikutus suorituskykyyn.

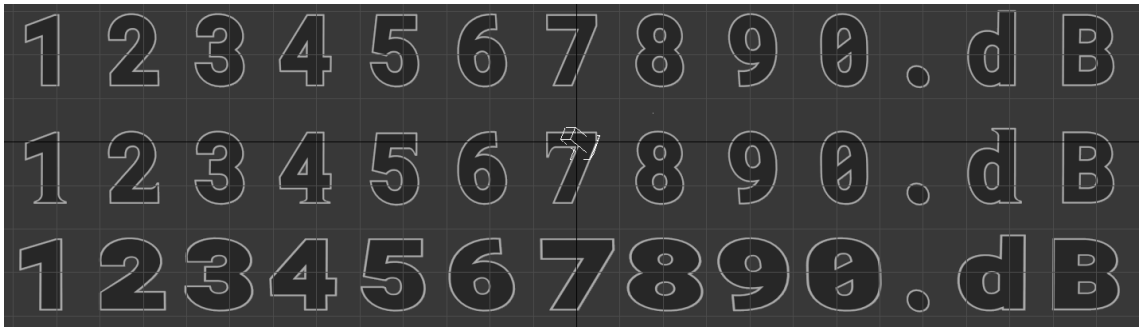
Tavoitteena oli, että muokatut mallit pystyttiin ottamaan käyttöön asiakkaan omassa applikaatiossa. Tätä varten suunniteltiin laajennus sillä tavalla, että kun käyttäjä on tyytyväinen muutoksiin, niin koko mallikokoelma käydään läpi ja jokaiseen malliin tehdään komponentin parametrien mukaiset muutokset. Muutosten jälkeen malleista tehdään ”prefab”-objekti, joka tallennetaan projektikansioon. Lisäksi projektikansioon tallennetaan malleissa käytetyt materiaalit ja mallien muodot.

4.3 Laajennuksen toteutus

Prototyyppien ja tutkimusten jälkeen alkoi laajennuksen toteutus. Se alkoi 3D-mallikokoelman muokkaamisella, jonka jälkeen ohjelmoitiin laajennus. 3D-mallikokoelma koostui numeroista ja desibelilyhenteestä. Laajennuksessa on neljä tiedostoa, jotka ovat Primitive.cs, ModifierComponent.cs, CreateTextTool.cs ja ModifyTool.cs. Kaksi ensimmäistä tiedostoa ovat editorikomentosarjatiedostoja ja jälkimmäiset kaksi ovat tavallisia Unityn komentosarjatiedostoja.

4.3.1 Mallikokoelman muokkaus ja valmistelu laajennusta varten

3D-mallikokoelman muokkaaminen tapahtui 3Ds Max -mallinnusohjelmalla. Jokaisesta numerosta tarvittiin perusmuodon lisäksi pääteviiva- ja lihavoitu-muodon (Kuva 16). Tämä osoittautui odotettua hankalammaksi, sillä asiakkaan antaman mallikokoelman tiedostomuoto oli fbx ja se oli tuotu Modo-mallinnusohjelmasta. Mallien akselit olivat kääntyneet väärään suuntaan ja mallien keskipiste oli muualla kuin mallin keskellä. Näiden asioiden takia jouduttiin tekemään muutoksia kolmasti ja siihen tuhlaantui aikaa runsaasti. 3Ds Max -ohjelmasta opittiin paljon uusia asioita ongelmia ratkottaessa. Selkein oppi oli se, että kahta eri mallinnusohjelmaa ei kannata sekoittaa keskenään.



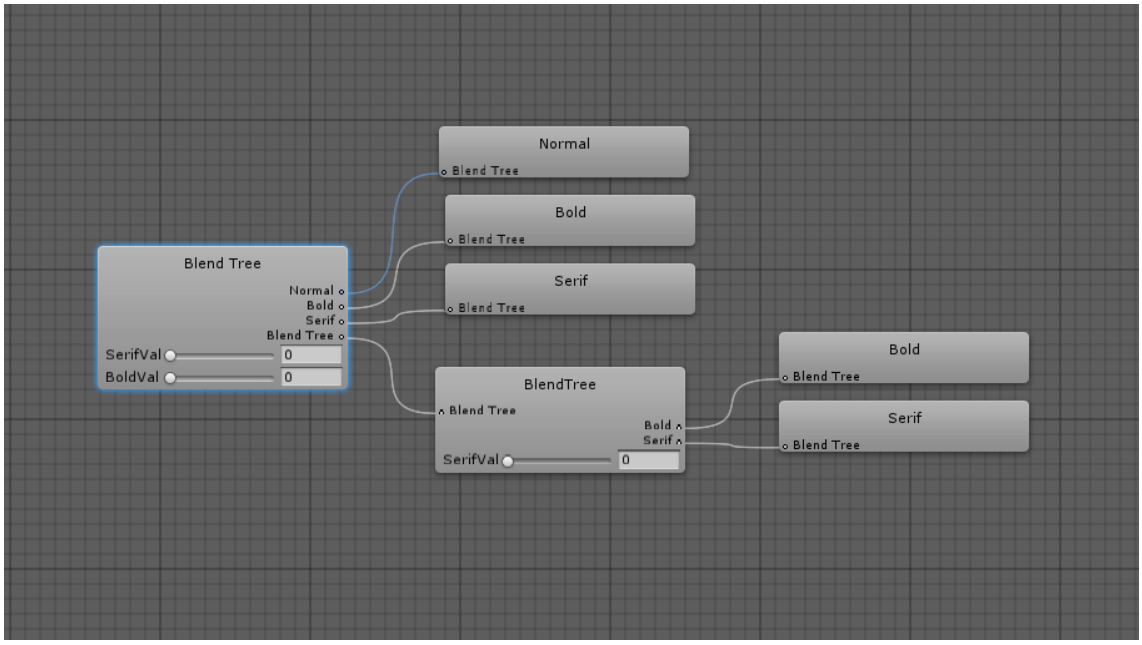
Kuva 16. 3D-mallinnusohjelmassa olevat mallit.

3D-mallit tarvitsivat myös verteksien värit. Niitä käytetään etsittäessä oikeat verteksit Unityssä. Värit ovat RGB-muodossa, joita pystytään vertaamaan koodissa ja näin löytämään oikean väriset verteksit. Oikeat verteksit löydettyä, niitä pystytään manipuloimaan halutulla tavalla. Tässä tietyissä tapauksessa niiden avulla muokataan reunusviivan pak-

suutta. Työskentelyn edetessä huomattiin reunusviivan muokkaamisen olevan hankalampaa kuin alun perin oli ajateltu. Ominaisuus jäi kehityksestä pois ja samalla verteksien värit jäivät hyödyntämättä.

Jokaiselle 3D-mallille tuli myös animaatio, joka oli 6 kuvaa pitkä. Animaation toteutin 3Ds Maxin ”morpher”-muuttujaa käyttäen. ”morpher”-muuttujan avulla on mahdollista muuttaa parametrien avulla 3D-malli toiseksi, kunhan siinä on saman verran verteksejä kuin alkuperäisessä mallissa. 3D-mallin animaatiossa, jokainen muoto oli 2 kuvaa pitkä ja tämä animaatio jaettiin kolmeen osaan Unityssä. Animaatiot sijoitettiin animaattoreihin, joissa oli ”blentree”-ominaisuudet.

Toisen prototyypin pohjalta aloitettiin ohjelmoimaan laajennusta. Siinä prototyypissä oli mahdollista muokata vain yhtä mallia eikä sitä voinut käyttäjä vaihtaa. Prototyypissä oli kuitenkin jo mallin valmiina muokkausominaisuudet, joiden koodia muuttamalla saatiin monet ominaisuuksista toimimaan useammalla mallilla. Funktiot muutettiin toimimaan silmukoissa, jotka käyvät käyttäjän syötteestä luodut listat läpi. Jokaisesta mallista tallennetaan keskipiste, verteksit ja alkuperäinen malli. Myös jokaisen mallin animaattorikomponentit tallentuvat listaan. Näin tehtiin, koska pääteviiva- ja lihavoitu-muodot käyttävät animaattorin ”blendtree”-ominaisuutta hyväksi (Kuva 17).



Kuva 17. Unityn animaattorinäkymä.

Animaattorien parametreja kontrolloidaan komponentin parametrien avulla. Tässä kohtaa pitää aina tarkistaa se, että jos kyseisen parametrin arvo muuttuu, niin mallia täytyy muuttaa animaattorin avulla. Animaattorit päivitetään automaattisesti Update-funktion avulla, jolloin arvojen muutokset tulevat näkyviin.

4.3.2 Laajennuksen alustus ja ohjelmointi

Unity 5 -pelimoottorissa luotiin editorikomentosarjatiedosto ModifierComponent.cs ja sen kohteeksi tavallinen komentosarjatiedosto ModifyTool.cs. Tavallinen komentosarjatiedosto toteuttaa kaikki editorikomentosarjatiedoston havaitsemat käyttäjän toiminnot komponentissa. Primitive.cs ja CreateTextTool.cs tiedostojen avulla luotiin GameObject-ylävalikkoon lisäys, jonka avulla käyttäjän on mahdollista lisätä komponentin sisältävä kustomoitu primitiiviobjekti Scene-ikkunaan (Ohjelmakoodi 4). Oman primitiiviobjeiktin valikkoon lisäämisen ohjelmointiin tarvitaan vain muutama komentorivi Unity 5 -pelimoottorin hyvän editoriohjelmointirajapinnan vuoksi.

Ohjelmakoodi 4. Primitiiviobjektin lisäys GameObject-valikkoon.

```
public class TextTool : MonoBehaviour {

    //instantiates new gameobject to root
    [MenuItem("GameObject/3D Object/3D modifiable text")]
    public static void Make3DText()
    { CreateTextTool.Create(); }


    //instantiates new gameobject as child of selected gameobject
    [MenuItem("GameObject/3D Object/3D modifiable text")]
    public static void Make3DText_child()
    {
        GameObject go = CreateTextTool.Create();
        if(Selection.activeGameObject)
        {
           go.gameObject.transform.parent = Selection.activeGameObject.transform;
        }
    }

}
```

ModifierComponent-tiedoston ohjelmointi, joka luo Inspector-ikkunaan komponentin käyttöliittymän, aloitettiin OnEnable-funktiolla. Sen sisään luotiin editoriohjelmointirajapinnan target-muuttujan avulla yhteys ModifyTool-tiedostoon. ModifyTool-tiedosto toteuttaa parametrien muutokset malleihin, jotka tulevat

ModifierComponent-tiedostosta. Target-muuttujan alustuksen lisäksi sinne lisättiin komentosarjat, jotka aloittavat päivityksen suorituksen sekä tarkkailevat peruutustoimintoa. Laajennuksen vuoksi OnEnable-funktioon lisättiin myös funktiokutsu, joka lataa 3D-mallit Resource-kansiosta listaan. Se alustetaan joka kerta uudestaan, koska muuten lista täyttyisi samoista 3D-malleista. Käyttäjän vanhat syötteet talletetaan oldInput-muuttujaan ja sitä käytetään pää-asiassa peruutustoiminnon yhteydessä sekä tarkastettaessa tekstikentän muutoksia.

3D-numeroiden ja komentosarjatiedostojen alustuksen valmistuttua ohjelmoitiin komponenttiin tekstikenttä, johon käyttäjä pystyy syöttämään haluamansa numerot. Tekstikenttä havaitsee enter-näppäimen painalluksen sekä jos kohdistus katoaa tekstikentästä. Molemmissa tapauksissa aloitetaan syötteen tarkastus.

Ohjelmakoodi 5. OnEnable-funktio kokonaisuudessaan.

```
void OnEnable()
    {
        //Links this script to modifytool script
        Debug.Log("Enabled modifierComponent");
        _target = (ModifyTool)target;
        //Get and set the oldinput value. Mainly for undo events.
        oldInput = _target._oldInput.ToString();

        _target.GetFontsToList();

        if (int.Parse(oldInput) > 0 && _target.instantiatedFonts.Count == 0)
        {
            //if undo has happened and there is oldinput the instantiate fonts.
            _target.InstantiateModels();

        }
        EditorApplication.update += Update;
        Undo.undoRedoPerformed += UndoCheck;

    }
```

Syöte tarkastetaan koodissa ennen kuin malleja tuodaan Scene-ikkunaan, koska se hyväksyy ainoastaan numeroita. Tämä toteutus mahdollisti ascii-arvojen käytön, joiden avulla numerot tuotiin Scene-ikkunaan käyttäjän nähtäväksi. Ascii-arvon avulla saadaan indeksinumero peliobjektilistaa varten (Ohjelmakoodi 6).

Ohjelmakoodi 6. Metodi mallien tuontia varten.

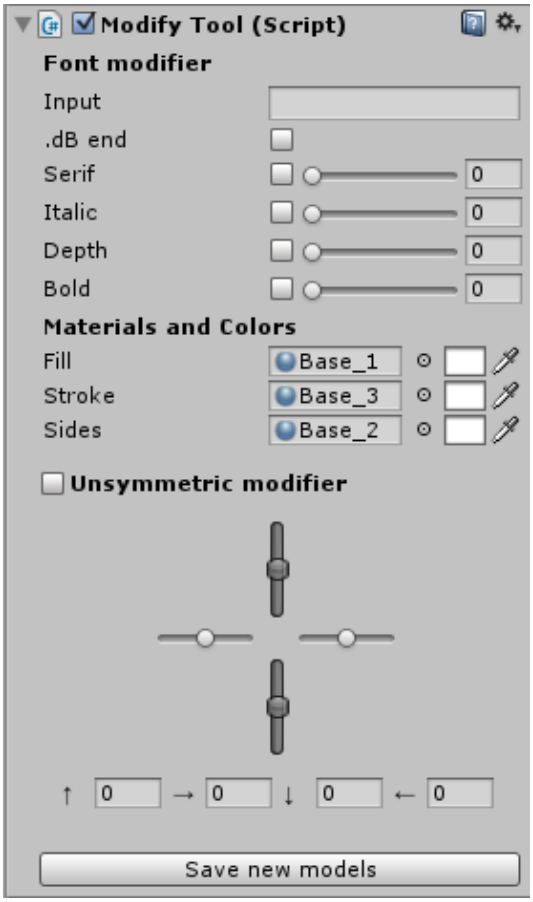
```
    void InstantiateModels()
    {
        char[] textToChars = text.ToCharArray();
        instantiatedFonts = new List<GameObject>();
        float offset = 0;

        for (int x = textToChars.Length - 1; x >= 0; x--)
        {
            Debug.Log("Trying to instantiate: " + fontSet[(int)textToChars[x] -
48].name);
            GameObject t = UnityEditor.PrefabUtility.InstantiatePre-
fab(fontSet[(int)textToChars[x] - 48]) as GameObject;
            t.transform.parent = this.gameObject.transform;
            t.transform.localPosition = new Vector3(-offset * 15f, 0f, 0f);
            instantiatedFonts.Add(t);
            offset += 1f;
        }
        if (dBEnd) Instantiate_dB();

        SetAllMeshes(instantiatedFonts);
        VertexMoveForeach();
    }
```

Numero 0 on ascii-arvossa 48, joten vähentämällä 48 saadaan ensimmäiseksi indeksiksi 0. Desibelilyhenne ratkaistiin tekemällä komponenttiin valintaruutu, jonka avulla käyttäjä saa lyhenteen näkyviin Scene-ikkunaan. Lyhenne tulee aina numeroiden perään. Tämä ratkaistiin luomalla desibelilyhenteen merkeille oma lista. Se liitetään aina numerolistan loppuun tai poistetaan sieltä. Tällä tavoin käyttäjän syöttämät numerot pysyvät aina Scene-ikkunassa vaikka desibelilyhenne luodaan tai poistetaan.

Toisesta prototyypistä otetut koodit muokattiin toimimaan mallilistojen kanssa. Funktioihin lisättiin listan iterointi, jolloin kaikki Scene-ikkunaan luodut mallit käydään läpi ja niihin tehdään käyttäjän tekemät muutokset. Koodin muokkaamiseen kului paljon aikaa ja joissain tapauksissa listoista muodostui moniulotteisia kuten esimerkiksi 3D-mallin verteksi-listassa.

Komponentin käyttöliittymän ohjelmointi oli yksinkertaista, mutta käyttöliittymässä tapahtuvien muutosten havaitseminen ja niiden mukaan toimiminen oli hankalampaa. Kompo-

nenttiin ohjelmoitiin serif-, bold-, italic- ja depth-muuttujille käyttöliittymä (Kuva 18). Jokaisen muuttujan saa pois päältä valintaruudun avulla ja sen perässä on liukusäädin, jonka avulla käyttäjä voi määritellä muuttujan suuruuden. Desibelilyhenteelle ohjelmoitiin myös oma valintaruutu, jolla desibelilyhenteen saa päälle, tai pois päältä. Muuttujien lisäksi komponenttiin ohjelmoitiin kolmelle eri materiaalille lokerot ja värinvalitsimet. Niiden avulla käyttäjä voi helposti muokata 3D-mallien ulkoasua pudottamalla materiaali objekti lokeroon. Helpotuksen vuoksi materiaalilokeron jälkeen sijoitettiin värinvalitsin, jolla kyetään helposti vaihtamaan kyseisen materiaalin väriä. Värinvalitsin on myös sitä varten, jos käyttäjä ei halua laittaa omaa materiaalia vaan vaihtaa ainoastaan väriä. Komponentin viimeisin ominaisuus on 3D-mallin koon muuttaminen x- ja y-akseleilla. Ominaisuuden käyttöliittymä koostui neljästä liukusäätimestä ja neljästä desimaalilukukentästä. Muokkaamisessa käytettiin 3D-mallin keskipistettä apuna ja siihen verteksien sijaintia vertaamalla pystyttiin liikuttamaan vain valittuja verteksejä. Desimaalilukukentät antavat käyttäjälle mahdollisuuden kirjoittaa muuttujalle arvon. Ne myös helpottavat arvojen nollaamisen, koska muutoin ei muuttujan arvoa näy missään. Esimerkiksi ensimmäinen pystyliukusäädin vaikuttaa vain 3D-mallin keskipisteen yläpuolella oleviin vertekseihin (Kuva 18).



Kuva 18. Laajennuksen komponentti Inspector-ikkunassa.

UV-karttojen muuttaminen 3D-mallin muuttumisen mukaan onnistui osittain. Se onnistui silloin kun mallia muutettiin verteksejä siirtämälle, mutta ei silloin kun käytettiin animaattoria. Animaattoria käytettäessä mallin muoto tallentui eri tavalla ja siitä ei saatu luettua oikeita verteksjä, jotta olisi kyetty muokkaamaan UV-karttaa oikein. Italic- ja Unsymmetric-muuttujien arvojen muuttuessa UV-kartta päivittyi, mutta Bold- ja Serif-muuttujien arvojen muuttuessa UV-kartta ei päivittynyt, ja siksi tekstuurien vääristyminen onnistuttiin estämään vain osittain.

4.3.3 Undo-toiminnon lisääminen laajennukseen

Komponenttiin täytyi myös erikseen ohjelmoida Undo-toiminto, jos käyttäjä haluaa perua muutoksen. Toimintoa varten täytyi kirjoittaa komento, jotta kaikki komponentin muutokset tallentuvat muistiin (Ohjelmakoodi 6).

Ohjelmakoodi 6. Undo-toiminto editorikomentosarjatiedostossa.

```
void OnEnable()
{

    …
    Undo.undoRedoPerformed += UndoCheck;
}

public override void OnInspectorGUI()
{
    Undo.RecordObject(_target, "Revert changes");

    …
}
```

Tämä ei kuitenkaan riittänyt sillä input-kentän kautta luodaan peliobjekteja, jotka ovat 3D-numeroita. 3D-mallien poistaminen käyttäjän peruessa toimintonsa täytyi ensin ohjelmoida komponentti havaitsemaan tilanne, jonka jälkeen poistaminen piti käsitellä erikseen. Sen jälkeen täytyi luoda uudet 3D-numerot vanhan input-kentän syötteen mukaan (Ohjelmakoodi 7). Undo-toiminto täytyi vielä tarkastaa input-kentän syötteen muuttumisen varalta ja sitä varten ohjelmoitiin UndoCheck-funktio. Lopuksi se liitettiin Undo.undoRedoPerformed-tapahtumaan, joka sijaitsi ModifierComponent.cs komentosarjatiedostossa. Undo-toiminnon ymmärtämiseen ja ohjelmoimiseen kului yllättävästi aikaa, sillä Unityn dokumentaatio oli tältä osin puutteellista. Peliobjektien tuhoaminen editoritilassa vaatii normaalista poikkeavan funktiokutsun ja dokumentaatiossa varoitettiin sen käyttämisestä, koska sillä voi vahingossa poistaa pysyvästi peliobjektin. Tavallisen Destroy-funktiokutsun sijasta käytetään DestroyImmediate-funktiokutsua.

Ohjelmakoodi 7. Undo-funktio, jota kutsutaan peruutustilanteessa.

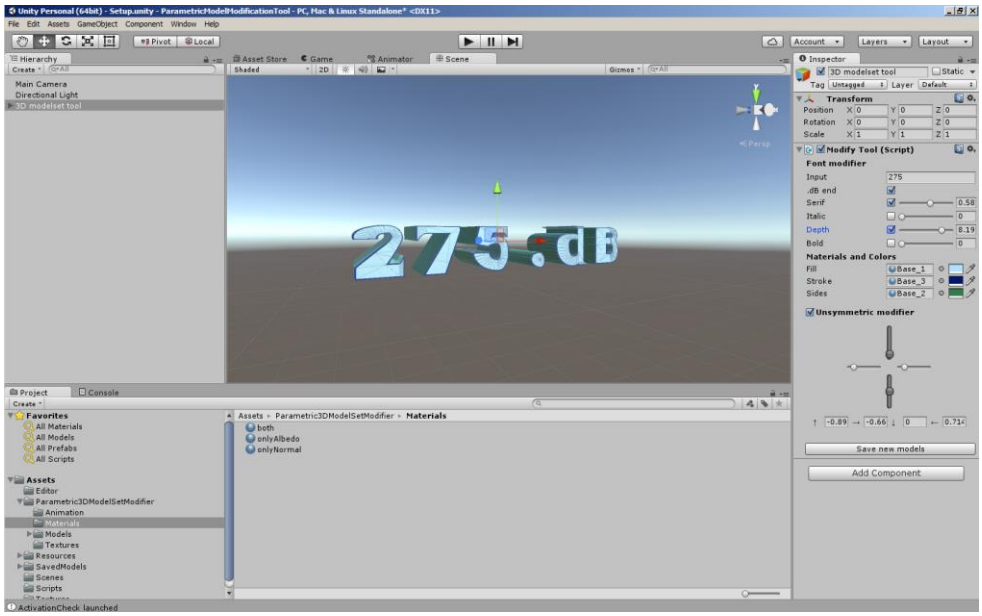
```
public void Undo()
    {
        undoUsed = true;

        if (transform != null)
        {
            for (int j = transform.childCount - 1; j >= 0; --j)
            {
                DestroyImmediate(transform.GetChild(j).gameObject);
            }
            transform.DetachChildren();
        }
        DestroyAllGameObjects();
        InputOperations();
        ForceUpdateMaterials();
    }
```

4.3.4 Mallikokoelman tallennus ja paketointi

Viimeisenä laajennuksen ominaisuutena ohjelmoitiin tallennus-ominaisuus. Tallentaminen tapahtuu ”Save new models”-nappulan avulla. Se aloittaa toiminnon, joka ensin luo SavedModel-kansion projektin Assets-kansion alle. SavedModel-kansion alikansioksi luodaan vielä Materials-kansio talletettuja materiaaleja varten ja Mesh-kansio talletettuja 3D-malleja varten. Kun kansiot on luotu niin ohjelma jatkaa materiaalien tallentamisella. Materiaalit käydään läpi ja ne joita ei projektikansiosta löydy luodaan tallennusta varten. Kaikki luodut materiaalit sijoitetaan Materials kansioon. Tämän jälkeen käydään koko 3D-mallikokoelma läpi ja ohjelma tekee jokaiseen malliin muutokset, jonka jälkeen muoto talletetaan Mesh-kansioon. 3D-mallien peliobjekteista tehdään ”prefab”-objekteja ja ne talletetaan suoraan SavedModels-kansioon. ”Prefab”-objekteja voidaan käyttää muissa Unity-projekteissa. Tallennusprosessin loputtua SavedModel-kansiosta tehdään paketti, joka voidaan ladata toiseen Unity-projektiin. Objektien paketointi etsii kaikki malleihin liittyvät osat ja liittää ne pakettiin mukaan. Tämä tehtiin helpottamaan luotujen 3D-mallikokoelmien käyttöä ja siirtoa. Paketteihin tuli mukaan myös animaattorit, jotka eivät ole tarpeellisia. Yksi ratkaisu niiden välttämiseksi olisi ollut se, että ”prefab”-objektit olisi rakennettu komentosarjojen avulla alusta, ja animaattori-komponentti olisi jätetty tässä vaiheessa pois.

4.3.5 Laajennuksen viimeistely

Lopuksi komentosarjatiedostojen komentorivit siistittiin ja turhat kommentoinnit poistettiin. Unityn konsolille meneviä lokeja vähennettiin ja vain tärkeimmät informaatiot jätettiin käyttöön. Laajennuksen optimointi vaati muutamia rakenne muutoksia komentosarjatiedostoissa, jotta pystyttiin poistamaan turhat listojen iteroinnit. Ainoastaan animaattorien päivitys jäi Update-funktioon. Muut muutosten tarkastukset toteutettiin käyttöliittymän BeginCheck- ja EndCheck-funktioiden avulla. Tämä varmisti sen, että aina kun muutos tapahtuu käyttöliittymässä, niin silloin tehdään asiaan kuuluva funktiokutsu. Viimeisenä vaiheena oli testaaminen mahdollisimman monella tavalla, jotta laajennus on tarpeeksi vakaa (Kuva 19). Testaamista tapahtui paljon myös kehitystyön ohessa. Tämä johtui siitä, että oli paljon uutta asiaa, ja siksi jouduttiin useasti testaamaan ominaisuuksien toimivuus. Unityn Debug.Log-funktio tuli tässä vaiheessa erittäin tarpeelliseksi, sillä sen avulla pystyttiin seuraamaan komentosarjojen etenemistä. Testaus-vaiheen jälkeen laajennus paketoitiin oikein, jotta se on helppo ottaa käyttöön Unity-projekteissa.



Kuva 19. Parametrinen 3D-mallikokoelman muokkaustyökalu.

5 LOPUKSI

Opinnäytetyön tavoitteena oli tutkia Unity 5 -pelimoottorin editorin laajentamista ja kehittää laajennus asiakkaalle. Asiakkaana toimi Uplause-yritys Helsingistä, joka tuottaa visualisointeja ja pelejä stadioneiden isoille näytöille. Pelit ja visualisoinnit perustuvat desibelimittaukseen, jonka avulla niitä ohjataan. Yleisö pelaa pelejä yhdessä käyttämällä ääntä.

Laajennuksen tarkoitus oli toimia mahdollisimman monipuolisena 3D-mallikokoelman muokkaustyökaluna. Mallikokoelma koostui numeroista sekä desibelin lyhenteestä. Asiakkaan oli tarkoitus ottaa laajennus käyttöön oman sovelluksensa ohelle ja tarjota sitä omille asiakkailleen. Laajentamista koskevan tutkimuksen ohessa havaittiin editoriohjelmoinnin eroavaisuuksia sekä useita tärkeitä seikkoja, jotka on hyvä huomioida laajennusta kehitettäessä. Opinnäytetyö käsitteli uutta aihe aluetta, josta löytyi vai vähän tieteellistä tutkimusta. Tämän takia opinnäytetyöllä on paljon uutuusarvoa.

Opinnäytetyö toteutettiin Unity 5 -pelimoottorilla ja samalla käytettiin 3Ds Max -mallinnusohjelmaa, Visual Studio -kehitysympäristöä sekä Adobe Photoshop CC -kuvankäsittelyohjelmaa. Työskentelyn alkuvaiheessa toteutus oli progressiivista, sillä oli paljon uutta opittavaa asiaa. Loppuvaiheessa opinnäytetyötä optimoitiin ja koodia järjestettiin loogisemmaksi.

Lopputulokseksi tuli toimiva laajennus. Sillä kyettiin muokkaamaan 3D-mallikokoelman numeroiden paksuutta, syvyyttä, kaltevuutta ja pääteviivoja. Numeroille pystyttiin asettamaan materiaaleja ja vaihtamaan värejä helposti. Viimeisin ominaisuus oli numeroiden sivujen kavennus ja laajennus. Neljällä liukusäätimellä pystyi vaikuttamaan numeron muotoon. Laajennuksella muokatut mallit pystyttiin tallentamaan ”Save New Models”-napin avulla, joka myös paketoi muodot, materiaalit sekä objektit. Laajennusta tehtäessä havaittiin editori-ohjelmoinnin olevan osittain erilaista ja siinä on paljon huomioitavia seikkoja. Laajennuksen muokkausominaisuuksien monipuolisuus kuitenkin kärsi kokemattomuuden vuoksi. Editoriohjelmoinnissa tuli vastaan useita asioita, jotka toimivat erilailla kuin normaalit komennot ja samalla täytyi ottaa huomioon asioita joista ei normaalisti tarvitse välittää. Toisaalta uusia asioita tuli opittua runsaasti sillä aihe oli tuntematon entuudestaan ja uuden oppiminen tuki työskentelymotivaatiota. Unity 5 -pelimoottorin ymmärrys syveni entisestään ja pelimoottorin useat toiminallisuudet

tulivat selkeämmäksi. Opinnäytetyön myötä tuli myös selväksi, että on erittäin hyödyllistä tietyissä tilanteissa luoda omia laajennuksia nopeuttaakseen pelikehitystä.

Laajennuksesta jäi pois muutamia ominaisuuksia, jotka olisivat olleet hyödyllisiä. Näistä tärkein oli uuden mallikokoelman automaattinen käsittely, silloin kun käyttäjä lisää oman mallikokoelman. Automaattisella käsittelyllä tarkoitetaan sitä, että jokaisesta mallista otetaan automaattisesti animaatiot ja ne leikataan kolmeen osaan. Tämän jälkeen luodaan mallille animaattori, johon rakennetaan ”blendtree”-ominaisuus animaatioita varten. Viimeisenä automaation vaiheena mallista tehtäisiin Unityn ”prefab”-objekti, joka talletettaisiin projektikansioon laajennuksen käytettäväksi. Tämä lisäys auttaisi uusien mallikokoelmien tuomista laajennukseen ja se poistaisi siitä manuaalisen työn kokonaan. Laajennus ehdittiin optimoida, mutta erilaisten erikoistehosteiden tekeminen jäi pois. Molemmat ominaisuuksista jäivät pois, koska aihe oli uusi ja se aiheutti ajan loppumisen kesken. Nykyisellä tiedolla laajennuksesta saisi huomattavasti tehokkaamman, mutta silti täytyisi oppia uusia asioita runsaasti. Efekti-ominaisuus olisi myös tarpeellinen ja sen avulla saisi esimerkiksi numerot murtumaan tai salamoimaan, mutta se vaatisi runsaasti aikaa ja peräti uuden opinnäytetyön.

Materiaalien ja värien vaihtaminen onnistui hyvin, mutta UV-kartan muutosten tekeminen ei onnistunut kuin osittain. ”Blendtree”-ominaisuuden käyttö esti UV-kartan päivityksen, koska animaatioiden verteksien sijainteihin ei päästy käsiksi. Tutkimus osoitti, että verteksien sijainteihin pääsisi käsiksi vain näytönohjaimen muistin kautta ja tämä vaatisi jo alemman tason ohjelmointitaitoja. UV-karttojen päivitys onnistui kuitenkin niissä tilanteissa, joissa mallia muutettiin verteksejä liikuttamalla. Näissä tapauksissa verteksien sijainnit olivat tiedossa. Numeroiden reunaviivan paksuuden muokkaaminen liukusäätimellä jäi puuttumaan kokonaan ajan puutteen vuoksi. Toiminallisuus olisi vaatinut useita päiviä työskentelyä ja testausta, jotta reunuksen verteksien sijaintia olisi pystytty siirtämään oikein. Verteksejä olisi täytynyt siirtää poispäin ulkoreunasta ja tämän toteuttaminen olisi kestänyt pitkään, sillä valmiita ratkaisuja ei ollut. Alustava ajatus reunaviivan paksuuden muokkaamiseksi osoittautui toimimattomaksi, koska kaikkia tilanteita ei otettu huomioon.

Laajennusta varten voisi tutkia lisää ominaisuutta, jolla tietokoneessa olevista fontti-kirjastoista saadaan luotua 3D-mallit automaattisesti. Tämän kaltaisia laajennuksia on tehty useita, mutta luotujen mallien topologia ei ole riittävän tasokas. Jatkotutkimuksen tarkoitus olisi selvittää, kuinka saataisiin tehokkaasti luotua topologisesti laadukkaita 3D-malleja fontti-kirjastoista. Tähän tutkimukseen saattaisi

kulua runsaasti aikaa, sillä siinä tarvitsi olla alkeellisen tason tekoälyä mukana, jotta mallit saataisiin tasokkaiksi. Laajennukseen ehdittiin toteuttaa osittain toimivat UV-karttojen muokkautumisominaisuus. Tekstuurien venyminen saatiin poistettua melkein kokonaan. Ainoastaan silloin kun 3D-malli muutti muotoa ”blendtree”-ominaisuuden avulla, niin tekstuurit alkoivat venyä. Tämä johtui siitä, että ”blendtree”-ominaisuus käytti mallin muokkaamiseen animaatiota, jolloin verteksien sijainnit eivät päivittyneet. Tämän ongelman vuoksi olisi suositeltavaa kehittää kaikki muutosominaisuudet toimimaan verteksejä liikuttamalla, jotta niiden sijainteihin päästäisiin aina käsiksi.

LÄHTEET

Aleksandr 2014. Unity. Documentation, Unity scripting languages and you. Viitattu: 3.2.2016. http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/

Anthony, S. 2014. Extreme Tech. Valve quietly releases Source 2 engine, Source 2 version of Dota 2, and new Hammer map editor. Viitattu: 5.4.2016. http://www.extremetech.com/extreme/187723-valve-quietly-releases-source-2-engine-source-2-version-of-dota-2-and-new-hammer-map-editor

Chapple, C. 2015. Develop. Valve reveals Source 2 game engine. Viitattu: 5.4.2016. http://www.develop-online.net/news/valve-reveals-source-2-game-engine/0203875

Create 3D Games 2016. Unity 5 vs Unreal Engine 4. Viitattu: 5.4.2016. https://create3dgames.wordpress.com/2015/09/07/unity-5-vs-unreal-engine-4/

CryEngine 2016. CryEngine. Viitattu: 5.4.206. https://www.cryengine.com/get-cryengine

Elwin 2014. Sassybot. Tutorial: Extending the Unity3D editor. Viitattu: 4.2.2016. http://sassybot.com/blog/tutorial-extending-the-unity3d-editor/

Framebench 2015. Top 5 Game Engines For Developers. Viitattu: 2.2.2016. http://blog.framebench.com/top-5-game-engines-developers/

Giant Bomb 2014. CryEngine 3. Viitattu: 5.4.2016. http://www.giantbomb.com/cryengine-3/3015-2917/

ModDB 2011. CryEngine 3. Viitattu: 5.4.2016. http://www.moddb.com/engines/cryengine-3/downloads/cryengine-3-free-sdk

Paju, P. 2014. Pelieditorin luonti Unity 3D:ssä. Opinnäytetyö. Mediatekniikan koulutusohjelma. Lahden ammattikorkeakoulu. Saatavilla myös https://www.theseus.fi/handle/10024/72737

Paskova, K. 2014. Unity. Extending the Unity editor. Viitattu: 3.2.2016. http://blogs.unity3d.com/2014/04/17/extend-the-editor-to-infinity-and-beyond/

Tardes, A. 2015. Extending Unity with Editor Scripting. Birmingham, UK: PACKT publishing

Thorn, A. 2015. Mastering Unity Scripting. Birmingham, UK: PACKT publishing

Unity 2016a. Unity Pro and Unity Personal software license agreement 5.x. Viitattu: 5.4.2016. https://unity3d.com/legal/eula

Unity 2016b. Unity Documentation. Viitattu: 21.4.2016. http://docs.unity3d.com/ScriptReference/Undo.RecordObject.html

Unity 2016c. Unity Documentation. Viitattu: 21.4.2016. http://docs.unity3d.com/ScriptReference/SerializedObject.html

Unity Asset Store 2016a. Unity Asset Store, FlyingText3D. Viitattu: 25.4.2016. https://www.assetstore.unity3d.com/en/#!/content/3627

Unity Asset Store 2016b. Unity Asset Store, Realtime 3D Text. Viitattu: 25.4.2016. https://www.assetstore.unity3d.com/en/#!/content/3931

Unreal Engine 2016a. What is Unreal Engine 4. Viitattu: 5.4.2016. https://www.unrealengine.com/what-is-unreal-engine-4

Unreal Engine 2016b. Unreal Engine 4.6 released. Viitattu: 5.4.2016. https://www.unrealengine.com/blog/unreal-engine-46-release

Wilcox M. 2014. Developer Economics. Top Game Development Tools: Pros and Cons. Viitattu 30.5.2016. http://www.developereconomics.com/top-game-development-tools-pros-cons/