TAMPEREEN
AMMATTIKORKEAKOULU

3D-PELIKEHITYS ALOITTELIJAN NÄKÖKUL-MASTA

Antti Häyrinen

Opinnäytetyö
Syyskuu 2017
Tietojenkäsittely

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely

HÄYRINEN, ANTTI
3d-pelikehitys aloittelijan näkökulmasta

Opinnäytetyö 54 sivua, joista liitteitä 3 sivua
Syyskuu 2017

Tämän opinnäytetyön tarkoituksena oli tutustua 3d-pelikehityksen ja -mallintamisen perusteisiin aloittelijan näkökulmasta. Tavoitteeksi asetettiin yksinkertaisen luontotyyppisen 3d-pelimaailman luominen, joka sisältäisi jotain itse mallinnettuja 3d-objekteja. Tätä varten toteutettiin suomalaisille peliyrityksille kyselytutkimus, jonka avulla kartoitettiin muun muassa niiden käyttämiä pelimoottoreita ja 3d-mallinnusohjelmistoja.

Opinnäytetyössä mallinnettiin yksinkertainen talo ja tulisija käyttäen Maya LT -mallinnusohjelmistoa. Tämän jälkeen mallinnettuihin objekteihin lisättiin tekstuurit, jotka tehtiin suurimmaksi osaksi itse otetuista valokuvista PixPlant-ohjelman avulla. Teksturointivaiheessa tutustuttiin myös Maya LT:n Hypershade-materiaalieditoriin ja UV-editoriin, jonka avulla tekstuurit kohdistettiin 3d-objektien pinnalle.

3d-objektit siirrettiin Unity-pelimoottoriin ja sen avulla toteutettiin yksinkertainen 3d-pelimaailma, jossa tutustuttiin keskeisiin 3d-pelikehityksessä käytettyihin tekniikoihin ja Unityn tarjoamiin työkaluihin. Näitä olivat muun muassa maaston muotoilu ja teksturointi Terrain editorin avulla, yksinkertaisten skriptien tekeminen, puun tekeminen puueditorilla ja tulen ja savun toteuttaminen hiukkassysteemin avulla. Lopuksi tutustuttiin vielä Occlusion culling -tekniikkaan, jonka avulla 3d-pelimaailman suorituskykyä voi parantaa.

Opinnäytetyön tavoite saatiin täytettyä, mutta samalla huomattiin, että 3d-pelikehitys vaatii alati kehittyvistä pelimoottoreista, mallinnusohjelmistoista ja muista työkaluista huolimatta jonkin verran syventymistä.

Asiasanat: unity, maya lt, pelimoottori, 3d-mallinnus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems

HÄYRINEN, ANTTI:
3D Game Development from a Beginner's Point of View

Bachelor's thesis 54 pages, appendices 3 pages
September 2017

The purpose of this thesis was to explore the basics of 3D game development and 3D modeling from a beginner's point of view. To examine what 3D modeling software and game engines are generally used in the area of game development, a survey was conducted in Finnish game companies.

A house and a fireplace were modeled with Maya LT 3D modeling software and they were placed to a 3D terrain that was made with Unity. To texture the objects some photographs were taken and they were converted to textures using PixPlant 3 software. Texturing was done using the Hypershade material editor and the UV editor of Maya LT.

After the creation of a 3D terrain in Unity, it was used to explore some of the techniques and tools commonly used in 3D game engines. Those included sculpting and texturing a terrain with terrain editor, making simple scripts, using and generating a tree with the tree editor of Unity and making fire and smoke effects with the particle system. After finishing the landscape editing, two 3D graphics optimization techniques, namely occlusion culling and level of detail (LOD), were examined briefly.

As a result, it was noticed that even though modern game developing tools and 3D modeling software are very powerful and user friendly, some knowledge about 3D fundamentals is still needed when developing 3D games.

Key words: unity, maya lt, game engine, 3d modeling

SISÄLLYS

LYHENTEET JA TERMIT

FPS	Frames per second, näytölle sekunnissa piirrettävien kuvien lukumäärä
Polygoni	3d-objektin pinnan muodostava monikulmio
Verteksi	Polygonin kärkipiste
Skripti	Pieni tietokoneohjelma, ”koodinpätkä”
Komponentti	Unityn objektin yksittäinen ominaisuus
Asset	Unityn yksittäinen resurssi, esimerkiksi objekti tai skripti
Scene	Unityn yksittäinen pelimaailma, ”kenttä”

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on tutustua 3d-pelikehittämisen alkeisiin ja tähän liittyen kartoittaa suomalaisissa peliyrityksissä käytettäviä kehitysohjelmistoja. Opinnäytetyössä käsitellään 3d-mallintamisen alkeet sekä sijoitetaan mallintamalla tehdyt objektit pelimoottorin avulla luotuun luontotyyppiseen 3d-pelimaailmaan. Ennen varsinaiseen käytännön työhön siirtymistä toteutettiin kysely suomalaisille peliyrityksille, jossa kartoitettiin tarkemmin heidän käyttämiään 3d-mallinnusohjelmistoja, pelimoottoreita sekä pelien kehitys- ja kohdealustoja. Näin pyritään saamaan käsitys siitä, mitä ohjelmistoja alalla yleisesti käytetään ja mihin näin ollen kannattaa tutustua syvällisemmin. Opinnäytetyön painotus on hyvin pitkälti visuaalisessa puolessa, eikä esimerkiksi ääniä tai pelattavuutta käsitellä sen tarkemmin.

Opinnäytetyöllä ei ole ulkopuolista toimeksiantajaa, vaan sen aihe syntyi puhtaasti opinnäytetyön tekijän omasta kiinnostuksesta pelialaa ja ennen kaikkea kolmiulotteista pelikehitystä kohtaan. Nykyään pelikehityssovellukset ovat jo sangen monipuolisia ja aloittelijaystävällisiä, eikä esimerkiksi ohjelmointitaito ole enää niin olennaista kuin vielä joitakin vuosia sitten.

Opinnäytetyössä lähtökohtana on aloittelijan tasolla oleva pelikehittäjä, joten lähestymistapa pyritään pitämään yksinkertaisena ja selkeänä. Opinnäytetyö tehdään siitä näkökulmasta, minkälaisia asioita aloittelijan tulisi ottaa huomioon ja mitä haasteita aloitteleva pelikehittäjä saattaa kohdata matkan varrella toteuttaessaan ensimmäistä yksinkertaista 3d-peliympäristöään.

2 KYSELYTUTKIMUS PELIMOOTTOREISTA JA 3D-MALLINNUSOHJELMISTOISTA

2.1 Kyselytutkimuksen toteutus

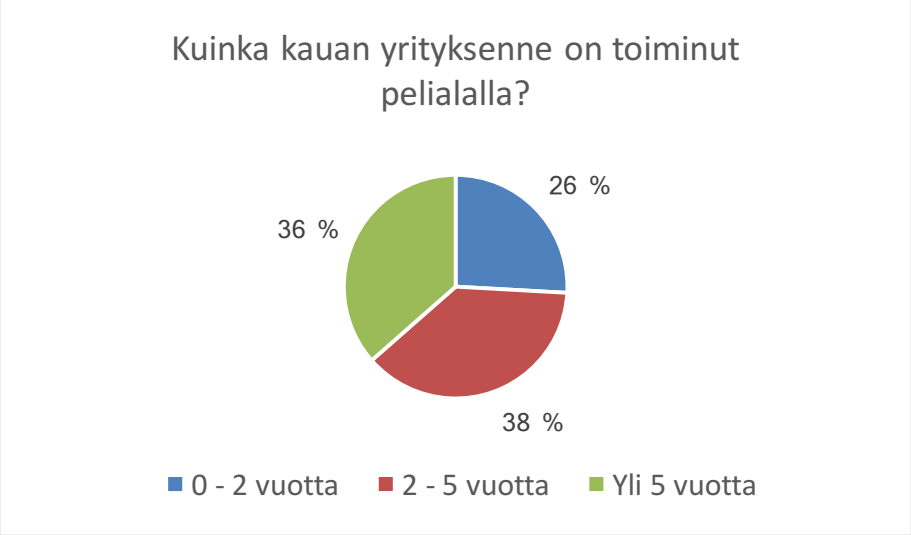
Opinnäytetyön perustana oli suomalaisille peliyrityksille toteutettu kyselytutkimus, jonka tarkoituksena oli kartoittaa peliyritysten käyttämiä pelimoottoreita sekä 3d-mallinnusohjelmistoja. Tutkimuksessa kysyttiin lisäksi pelikehityksen kohdealustoja, pelikehityksessä käytettyjä käyttöjärjestelmiä sekä sitä, kehittääkö yritys 2d-pelejä, 3d-pelejä vai molempia. Näiden lisäksi kysyttiin vielä yrityksen kokoa ja sitä, kuinka kauan se on toiminut alalla. Tutkimuksen kyselylomakkeessa oli myös vapaa kommenttikenttä. Tarkoituksena oli pitää kysely yksinkertaisena ja selkeänä, jotta yritysten kynnys vastata siihen olisi mahdollisimman matala. Kyselylomake on liitteessä 1.

Kysely toteutettiin Google Forms -palveluun luodun web-pohjaisen kyselylomakkeen avulla, ja tutkimus tehtiin 13.3. – 31.3.2017 lähettämällä linkki kyselylomakkeeseen 187:ään eri yritykseen. Määräaikaan mennessä vastaus saatiin 84:stä yrityksestä. Yritysten yhteystietojen hankkimiseen käytettiin suomalaisen pelialan kattojärjestö Neogamesin verkkosivuja, joilla on lista suomalaisista pelialan toimijoista. Listan perusteella vierailtiin yritysten kotisivuilla ja pyrittiin löytämään sopiva sähköpostiosoite, johon lähetettiin linkki kyselylomakkeeseen.

Kyselyn tulosten analysointiin käytettiin IBM:n kehittämää tilastolliseen analyysiin tarkoitettua SPSS Statistics -ohjelmaa. Analyysissä tutkittiin lähinnä vastausten jakautumia sekä tehtiin ristiintaulukointia. Ristiintaulukointi on tekniikka, jonka avulla voidaan tutkia muuttujien välisiä riippuvuuksia sekä niiden jakautumista (Kvantitatiivisten menetelmien tietovaranto 2013). Ristiintaulukoinnin avulla voidaan esimerkiksi tutkia sitä, miten yrityksen ikä vaikuttaa yrityksen henkilöstömäärään. Mitenkään erityisen syvälliseen analysointiin tulosten suhteen ei kuitenkaan ryhdytty, koska se ei kuulunut tämän opinnäytetyön aihepiiriin.

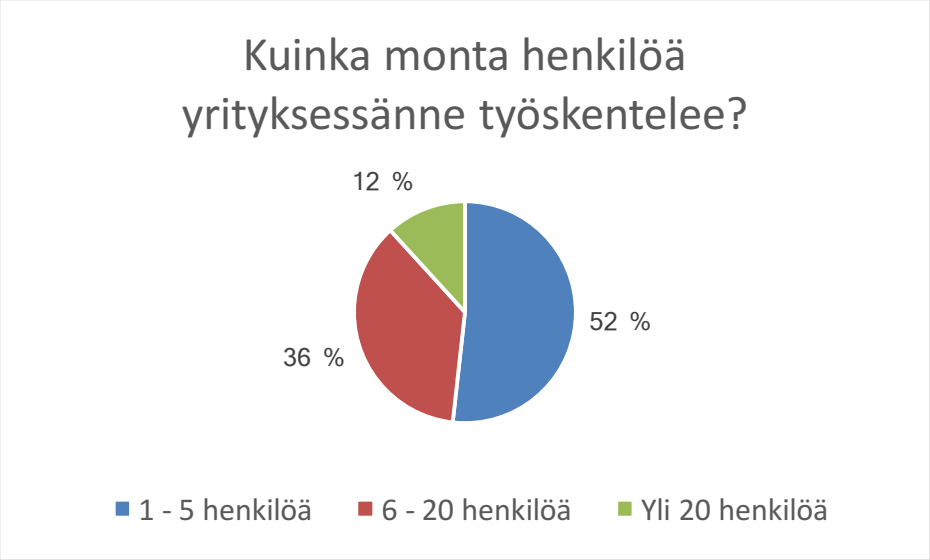
2.2 Kyselyn tulokset

Yrityksistä suurin osa eli 38 % (32 kappaletta) oli toiminut alalla 2-5 vuotta. Niukasti toiselle sijalle tuli yli 5 vuotta alalla toimineet yritykset, joita oli 36 % vastanneista (31 kappaletta). 0-2 vuotta alalla toimineita yrityksiä oli vastanneista 26 % (22 kappaletta) (kuva 1).



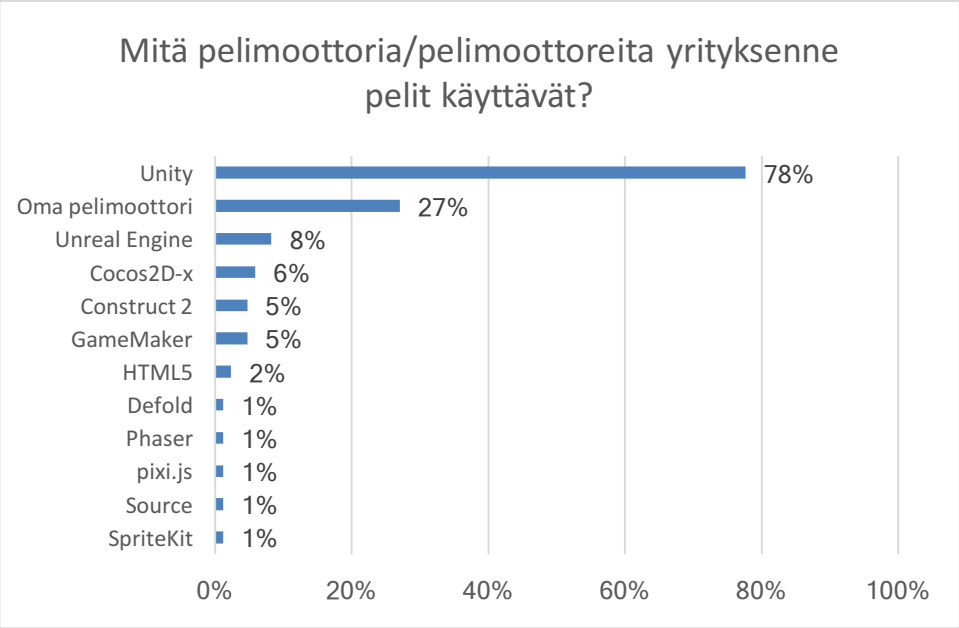
KUVA 1. Kuinka kauan yritys on toiminut pelialalla

Vastanneista yrityksistä suurin osa on 1-5 henkilön yrityksiä. Näitä oli kaikista yrityksistä yli puolet eli 52 % (44 kappaletta). Toiseksi eniten eli 36 % (31 kappaletta) on 6-20 henkilön yrityksiä ja vähiten eli 12 % (10 kappaletta) on suuria eli yli 20 henkilön yrityksiä (kuva 2).



KUVA 2. Kuinka monta henkilöä yrityksessä työskentelee

Unity on ylivoimaisesti suosituin yksittäinen pelimoottori, jota suomalaiset peliyritykset käyttävät. Melko paljon käytetään myös omaa pelimoottoria, mutta näiden kahden jälkeen muut pelimoottorit ovat melko pienessä asemassa. Suurin osa yrityksistä myös käyttää useampaa pelimoottoria, ja tämän takia käytettyjen pelimoottorien kokonaismäärä on suurempi kuin yritysten (kuva 3).

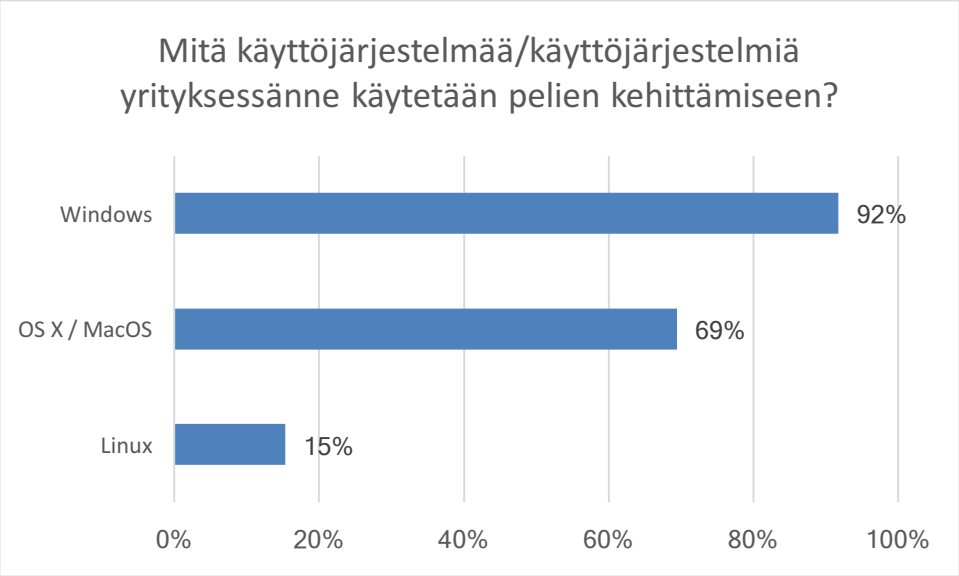


KUVA 3. Yritysten käyttämät pelimoottorit

Windows on suosituin käyttöjärjestelmä peliyritysten keskuudessa 92 %:n (78 kappaletta) osuudellaan. Applen käyttöjärjestelmää (OS X / MacOS) käyttää 69 % (59 kappaletta)

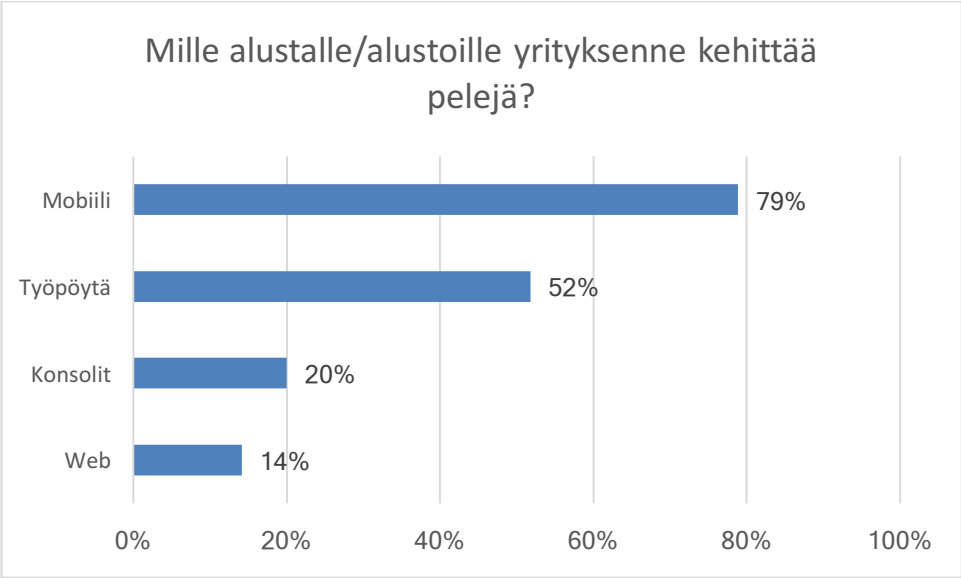
peliyrityksistä ja Linuxia 15 % (13 kappaletta) (kuva 4). Eräs kyselyyn vastanneista yrityksistä kommentoi asiaa seuraavasti:

”Kaikki työntekijät saavat välitä Windowsin ja Macin väliltä vapaasti. Samoin mallinnussoftien kanssa, ja mitä ikinä studiota koodin kirjoittamiseen haluavatkin käyttää. Pakotettuja on Unityn lisäksi firman oma serveriympäristö, Github ja Jira.”



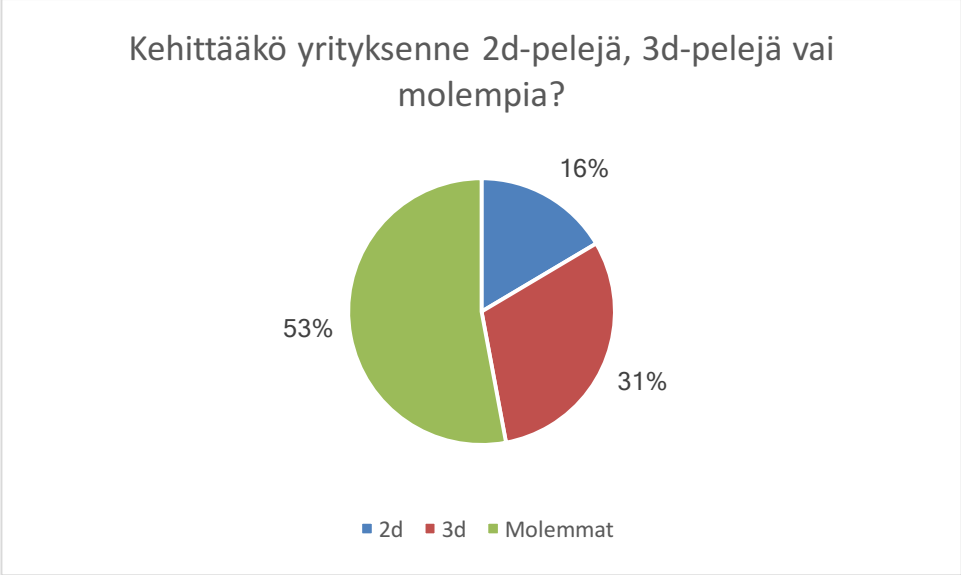
KUVA 4. Peliyritysten käyttämät käyttöjärjestelmät

Suosituin pelikehityksen kohdealusta on mobiililaitteet 79 %:n (67 kappaletta) osuudella, ja hieman yli puolet eli 52 % (44 yritystä) kehittää pelejään työpöytäympäristöön. Konsoleiden osuus on 20 % (17 yritystä) ja Web-pelien 14 % (12 yritystä) (kuva 5). Näiden lisäksi yksi yritys mainitsi kehittävänsä sovelluksia HTC Vive -virtuaalitodellisuusjärjestelmälle.



KUVA 5. Pelikehityksen kohdealusta.

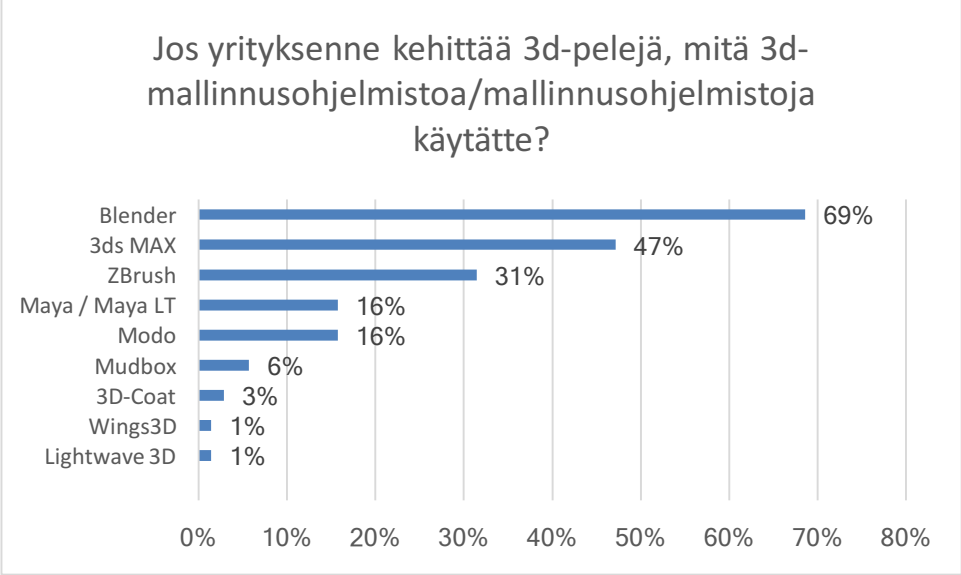
Suurin osa kyselyyn osallistuneista yrityksistä eli 53 % (45 yritystä) kehittää sekä kaksi- että kolmiulotteisia pelejä. Pelkästään kolmiulotteisia pelejä kehittää 31 % (26 yritystä) ja pelkästään kaksiulotteisia pelejä 16 % (13 yritystä) yrityksistä (kuva 6).



KUVA 6. Kehittääkö yritys kaksiulotteisia vai kolmiulotteisia pelejä

Suosituin mallinnusohjelmisto oli Blender 69 %:n (48 yritystä) osuudella. Tämän jälkeen toiseksi suosituin oli 3ds MAX 47 %:n (33 yritystä) osuudella ja kolmanneksi suosituin ZBrush 31 %:n (22 yritystä) osuudella (kuva 7). Kyselyyn osallistuneista 84:sta peliyrityksestä yhteensä 71 ilmoitti kehittävänsä kolmiulotteisia pelejä. Näistä 71:sta ainoastaan yksi yritys jätti mainitsematta käytetyn 3d-mallinnusohjelmiston, joten vastaus

saatiin siis 70:sta yrityksestä. 3d-mallinnusohjelmistojen kohdalla yleisesti oli havaittavissa se, että hyvin monet yritykset käyttivät useampaa kuin yhtä vaihtoehtoa. Tämä voi johtua varmasti samasta syystä kuin käyttöjärjestelmienkin kohdalla, eli työntekijällä on vapaus valita itselleen mieleinen kehitysympäristö. Toisaalta syynä voi olla myös se, että jokaisella 3d-ohjelmistolla on omat vahvuutensa, jolloin mallinnuksen eri työvaiheissa käytetään eri ohjelmistoja.



KUVA 7. Peliyritysten käyttämät 3d-mallinnusohjelmistot

2.3 Muita huomioita kyselytutkimuksesta

84:stä yrityksestä 65 ilmoitti käyttävänsä Unityä, ja näistä 65:stä yrityksestä pelkästään Unityä ilmoitti käyttävänsä 41 kappaletta. Näin ollen melkein puolessa vastanneista peliyrityksistä Unity oli ainoa käytössä oleva pelimoottori. Unityn suosio oli suurta oikeastaan kaiken kokoisissa yrityksissä, ja ainoastaan kaikkein pienimmissä eli 1-5 hengen yrityksissä Unityn suhteellinen suosio oli hieman pienempää kuin suuremmissa yrityksissä.

Toiseksi käytetyin pelimoottori oli oma pelimoottori, jota ilmoitti käyttävänsä yhteensä 23 yritystä. Pelkästään omaa pelimoottoria käytti 11 yritystä. Oman pelimoottorin käyttäminen näyttäisi yleistyvän sitä mukaa, mitä vanhempi yritys on kyseessä. Myös yrityksen koko näyttäisi vaikuttavan hieman, sillä yli 5 hengen yrityksissä oman pelimoottorin käyttäminen oli jonkin verran yleisempää kuin 1-5 hengen yrityksissä.

Kolmanneksi suosituin pelimoottori oli Unreal Engine, jota käytti 7 yritystä. Näistä yrityksistä 2 käytti ainoastaan Unreal Engineä. Molemmat pelkästään Unreal Engineä käyttävät yritykset kehittivät ainoastaan 3d-pelejä.

Näiden kolmen edellä mainitun pelimoottorin lisäksi muita pelimoottoreita käytettiin lähinnä melko satunnaisesti. Sellaisia yrityksiä, jotka eivät käyttäneet ollenkaan kolmea suosituinta pelimoottoria, oli ainoastaan 5 kappaletta. Nämä kaikki yritykset olivat kooltaan 1-5 hengen yrityksiä, ja yhtä yritystä lukuun ottamatta kaikki ilmoittivat tekevänsä vain 2d-pelejä. Yksikään näistä yrityksistä ei myöskään ollut yli 5 vuotta vanha.

3d-mallinnusohjelmistojen suhteen hajontaa oli enemmän kuin pelimoottoreissa, eikä mitään selkeää yhdistävää tekijää 3d-mallinnusohjelman valinnan suhteen voitu kyselyn perusteella löytää. Yrityksen iällä ei näytä olevan juurikaan vaikutusta 3d-mallinnusohjelman valintaan, vaan mallinnusohjelmistojen jakauma on hyvin samanlainen jokaisessa ikäryhmässä. Myöskään yrityksen koko ei näyttäisi juurikaan vaikuttavan käytettävään 3d-mallinnusohjelmistoon, ainoastaan Maya / Maya LT oli 1-5 hengen yrityksissä suhteellisesti vähemmän käytetty kuin suuremmissa firmoissa ja vastaavasti taas Blender ei ollut kaikkein suurimmissa yli 20 henkilön firmoissa niin suosittu kuin pienemmissä yrityksissä. Tässä kohtaa on kuitenkin hyvä huomata, että varsinkin suurimpien yli 20 henkilön yritysten kohdalla otos on niin pieni, että tulos ei liene tilastollisesti merkitsevä.

2.4 Kyselytutkimuksen epävarmuustekijät

Ensimmäisenä kyselyn epävarmuustekijöitä mietittäessä mieleen tulevat ongelmat, joita ilmeni heti kyselylomakkeen teko- ja testausvaiheessa. Google Forms tallentaa lomakkeelta saadut vastaukset Excel-tyyppiseen taulukkoon. Kuitenkaan lomakkeen kysymyksiin tehdyt muutokset eivät automaattisesti poista vastaustaulukon aikaisemmissa vaiheissa syntyneitä vanhempia vastauksia, vaan tämä pitää tehdä manuaalisesti. Kuitenkin Google Formsin oman yhteenvetosivun vastausmäärälaskuri jättää vanhat rivit huomioimatta, ja tämä aiheutti aluksi hämmennystä. Tästä seurasi se, että taulukon alkuun jäi muutama vanha testirivi, jotka oli kuitenkin jälkikäteen helppo jäljittää aikaleiman perusteella. Lisäksi tuloksista poistettiin yksi epämääräinen vastaus. Kaikesta tästä seurasi lievä epävarmuus koko Google Forms -järjestelmän luotettavuutta kohtaan.

Toinen ongelma tämän tyyppisessä kyselyssä on se, että lomake ei millään tavalla voi kontrolloida sitä, vastaako sama yritys tai henkilö useampaan kertaan. Vastauksista löytyi kaksi täysin identtistä vastausparia, mutta todennäköisesti tässä kohtaa kyse on sattumasta, sillä kysely on kuitenkin hyvin suppea sisältäen vain seitsemän kysymystä.

Joidenkin kysymysten kohdalla voi olla myös tulkinnanvaraisuutta. On mahdollista, että vastaaja on esimerkiksi mieltänyt PSP:n tai muun vastaavan käsikonsolin mobiililaitteeksi, vaikka kyselyn laatija on tarkoittanut mobiililaitteella pelkästään puhelimia ja tabletteja.

3 3D-MALLIEN TEKEMINEN

3.1 Mitä 3d-mallinnus on?

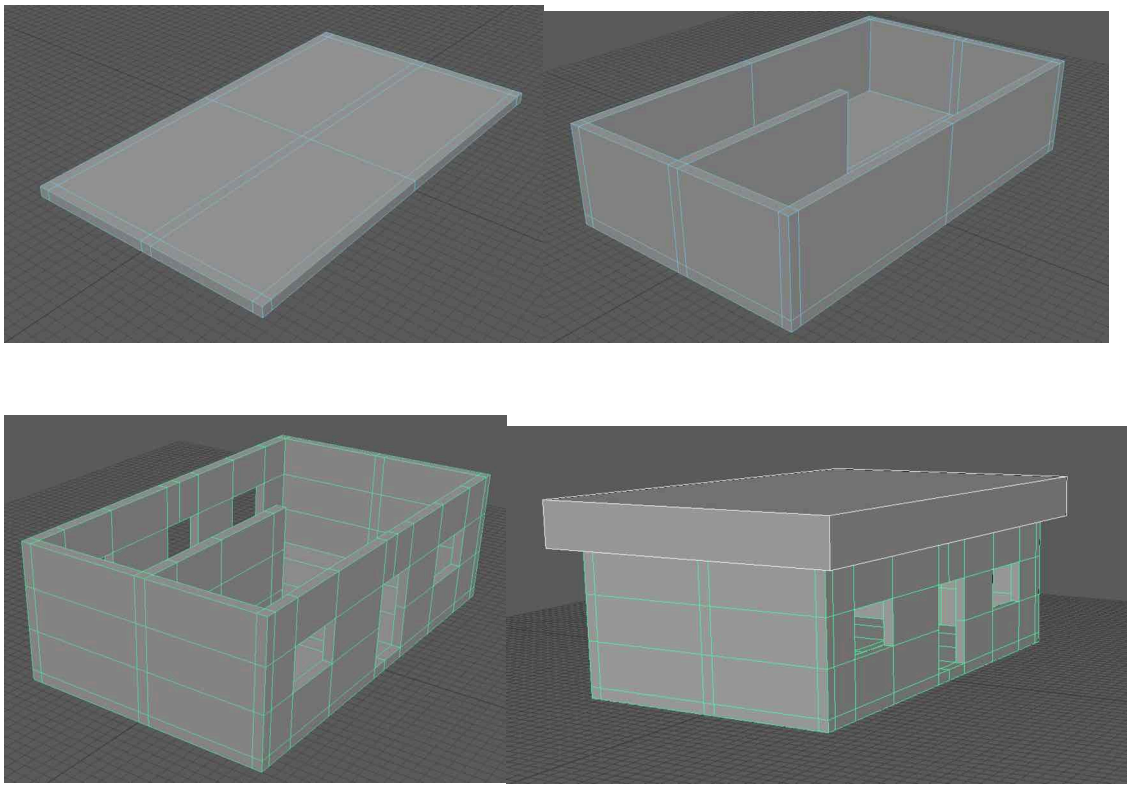
3d-mallinnuksella tarkoitetaan prosessia, jonka tarkoituksena on luoda tietokoneen avulla 3d-malli tasosta tai objektista 3d-avaruuteen monikulmioiden (polygonien), sivujen ja kärkipisteiden (verteksien) avulla. 3d-mallinnus voi tapahtua manuaalisesti 3d-mallinnusohjelmistoa apuna käyttäen tai esimerkiksi skannaamalla koneellisesti jokin oikea esine tai ympäristö (Slick 2016). 3d-mallinnusohjelmiston avulla tapahtuvassa mallinnuksessa 3d-malleja voi luoda useammalla eri tavalla, mutta yleisin tapa on käyttää perustana jotain 3d-peruselementtiä, kuten esimerkiksi tasoa, kuutiota, palloa tai sylinteriä (Zeman 2015, 30), jonka geometriaa muokataan haluttuun suuntaan erilaisten mallinnusohjelmiston työkalujen avulla.

3.2 Esimerkkimallien tekeminen Maya LT:llä

Kyselytutkimuksen perusteella kävi selväksi, että suomalaisissa peliyrityksissä on käytössä sangen laaja kirjo erilaisia 3d-mallinnusohjelmistoja. Suosituin ohjelmisto oli Blender, jonka suosiota varmasti selittää se, että se on ilmainen avoimen lähdekoodin ohjelmisto. Tässä opinnäytetyössä päädyttiin kuitenkin käyttämään Maya LT -mallinnusohjelmistoa, koska se oli kyselytutkimuksessa mukana olleista mallinnusohjelmistoista ainoa, josta opinnäytetyön tekijällä oli edes hieman aikaisempaa kokemusta. Maya LT on ominaisuuksiltaan karsittu versio Maya-mallinnusohjelmistosta, ja se on suunnattu indie-pelinkehittäjille. Maya LT on maksullinen ohjelmisto, mutta siitä on kuitenkin saatavilla ilmainen 30:n päivän kokeiluversio.

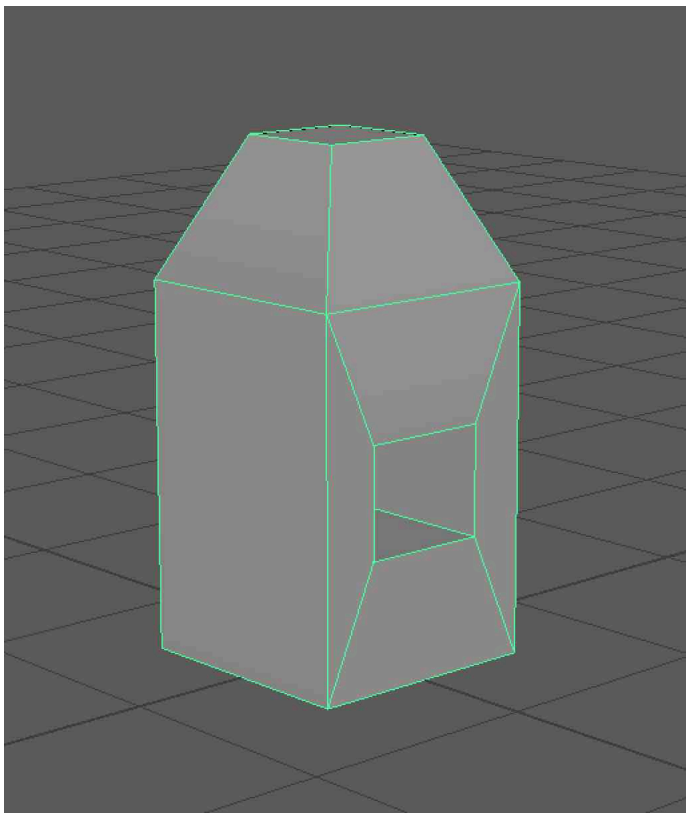
Maya LT:n avulla mallinnettiin yksinkertainen talo ja talon sisälle sijoitettava yksinkertainen tulisija. Talon mallinnuksessa lähtökohtana toimi kuutio, josta muotoiltiin talon sokkeli. Sokkeliin lisättiin seinien paikat jakamalla sitä pienempiin osiin Multi-Cut-työkalulla, jonka jälkeen seinät pursotettiin Extrude-toiminnon avulla ylös sokkelista. Tämän jälkeen seiniin lisättiin ikkunoiden paikat niin ikään Multi-Cut-työkalun avulla, ja ikkunaaukot tehtiin poistamalla pinnat niiden kohdalta sekä seinän ulko- että sisäpuolelta. Ikkunapaikkojen poiston jälkeen täytyi vielä lisätä pinnat muodostuneiden aukkojen reunoille

käyttäen Bridge-työkalua. Lopuksi talon päälle tehtiin katto lisäämällä uusi kuutio ja venyttämällä se sopivan kokoiseksi. Talon mallintaminen vaihe vaiheelta on esitelty kuvassa 8.



KUVA 8. Talon mallinnuksen vaiheet

Tulisija mallinnettiin hyvin pitkälti samoja työkaluja ja periaatteita noudattaen kuin talokin. Tulipesä lisättiin Extrude-työkalun avulla luomalla ensin suorakaide tulisijan yhdelle sivulle ja tämän jälkeen pursottamalla sitä sisäänpäin. Tulisijan yläpuolelle tehtiin niin ikään Extrude-toiminnon avulla pienempi suorakaide, jonka jälkeen suorakaidetta vedettiin ulospäin, jotta tulisijan yläpuolelle saatiin muodostumaan kartio. Tulisija on kuvassa 9.



KUVA 9. Tulisija etuviistosta

Yleisesti ottaen mallinnuksessa tulisi huomioida, että mallinnettavaan objektiin ei jää muita monikulmioita kuin kolmioita tai nelikulmioita (Thorn 2017, Meshes – work only with good topology). Muutenkin polygonimäärä tulisi pitää aina mahdollisimman pienenä. (Thorn 2017, Meshes – minimize polygon count)

3.3 Esimerkkimallien teksturointi ja pinnoitus

Jotta 3d-objektit olisivat jotain muutakin kuin pelkkiä yksivärisiä möykkyjä, niiden pintaa täytyy käsitellä erilaisten tekniikoiden avulla. 3d-objektien pinnan muodostamiseen liittyy oleellisesti materiaalit, shaderit (varjostimet) ja tekstuurit. Pelimoottorista tai mallinnusohjelmasta riippuen termit saattavat tarkoittaa hieman eri asiaa, mutta pääperiaate on sama.

Kun 3d-objektille halutaan luoda pinnoitus, luodaan tätä varten materiaali. Materiaali koostuu shaderista eli varjostimesta, joka määrittelee sen, millaisia ominaisuuksia materiaalilla on. Esimerkiksi peilaavan materiaalin shaderilla on yleensä eri ominaisuudet kuin

peilaamattoman materiaalin shaderilla. Teoreettisesti tarkasteltuna shaderit ovat matemaattisia laskutoimituksia, jotka muodostavat 3d-objektin pinnan kuvainformaation sen perusteella, miten valo käyttäytyy osuessaan 3d-objektin pintaan (Zeman 2015, 116-117). Käytettävästä shaderista riippuu muun muassa se, millaisia tekstuureita materiaalissa voidaan käyttää.

Tekstuurit ovat 2d-kuvia, joita käytetään materiaalien yhteydessä. Yksinkertaisimmillaan tekstuuri on pelkkä 3d-objektin pintaan lisätty kuviointi. Toisaalta tekstuuriksi kelpaa myöskin elokuva tai proseduraalinen tekstuuri. Proseduraalisella tekstuurilla tarkoitetaan kuvaa, joka luodaan matemaattisten kaavojen avulla, ja jota voidaan muokata, editoida ja animoida reaaliaikaisesti (Zeman 2015, 156).

3.3.1 Talon teksturointi

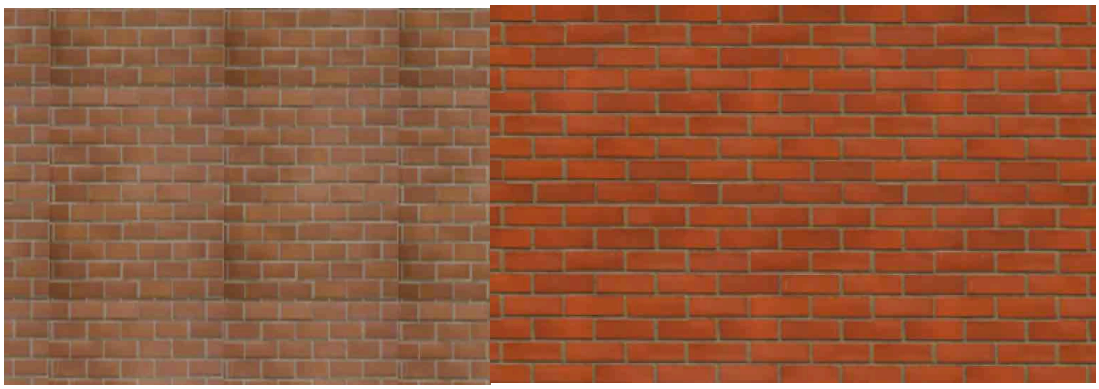
Ensimmäiseksi täytyi päättää, minkälaiset pintamateriaalit taloon halutaan. Tässä päädyttiin seuraavaan ratkaisuun:

- Ulkoseinä tiiliverhoilua
- Katon ulkoreuna pystypaneelia ja pintamateriaaliksi jotain bitumia muistuttavaa
- Sisällä lattiaan puupinnoitus ja seiniin tapetti

Internetistä löytyy runsaasti erilaisia ilmaisia ja vapaasti käytettävissä olevia kuvia sisältäviä kuvapankkeja. Tässä opinnäytetyössä päädyttiin kuitenkin siihen, että tekstuurit laadittaisiin ensisijaisesti itse otetuista valokuvista ja valmiisiin kuvapankkeihin turvauduttaisiin ainoastaan siinä tapauksessa, että sopivaa valokuvaa ei saada itse otettua. Yhden kameran kanssa tehdyn iltapäiväkävelyn seurauksena suurin osa pintamateriaaleihin vaadittavista valokuvista olikin kasassa. Ainoastaan talon lattiamateriaali ja tulisijan pinnoitus jäivät puuttumaan, joten näiden kohdalla turvauduttiin internetistä löytyviin kuvapankkeihin.

Jotta valokuvista tulisi hyviä tekstuureita, ne pitää muuttaa saumattomiksi. Tämä tarkoittaa sitä, että kuvan reunat muokataan sellaisiksi, että saumakohta ei erotu siinä vaiheessa, kun samaa kuvaa on useampi vierekkäin. Tekstuurit voi muuttaa saumattomiksi manuaalisesti esimerkiksi Photoshopin tai muun kuvankäsittelyohjelmiston avulla. Photoshopilla tämä tehtäisiin niin, että ensiksi Offset-työkalun avulla kuvan asemointia muutetaan siten,

että kuvan reunat siirtyvät keskelle. Tämän jälkeen näkyvät saumat häivytetään Clone-työkalun avulla (Cardoso 2015). Tässä opinnäytetyössä päädyttiin kuitenkin käyttämään PixPlant 3 -ohjelmaa, joka tekee saumattomia tekstuureita automaattisesti. Tämän lisäksi PixPlant myös korjaa automaattisesti valaistuksen tasaisuutta sekä värimaailmaa. Kuvassa 10 on käytetty lähtökohtana samaa valokuvaa. Vasemmanpuoleisessa kuvassa tekstuurin reunojen toistuvuus on selkeästi havaittavissa, kun taas oikeanpuoleisessa PixPlantin avulla käsitellyssä kuvassa toistuvuuden havaitseminen on huomattavasti vaikeampaa. Myös värisävy on käsitellyssä kuvassa eri. Tekstuurit tehtiin 512 * 512 pikselin kokoisiksi ja samalla niistä tehtiin myös normaalikartat, joista enemmän seuraavassa kappaleessa. Tekstuureiden suhteen olisi suotavaa, että niiden koko pikseleinä olisi jokin kahden potenssi, eli esimerkiksi 16 * 16, 32 * 32, 64 * 64 ja niin edelleen (Ahearn 2016).



KUVA 10. Käsittelemätön ja PixPlantilla käsitelty tekstuuri

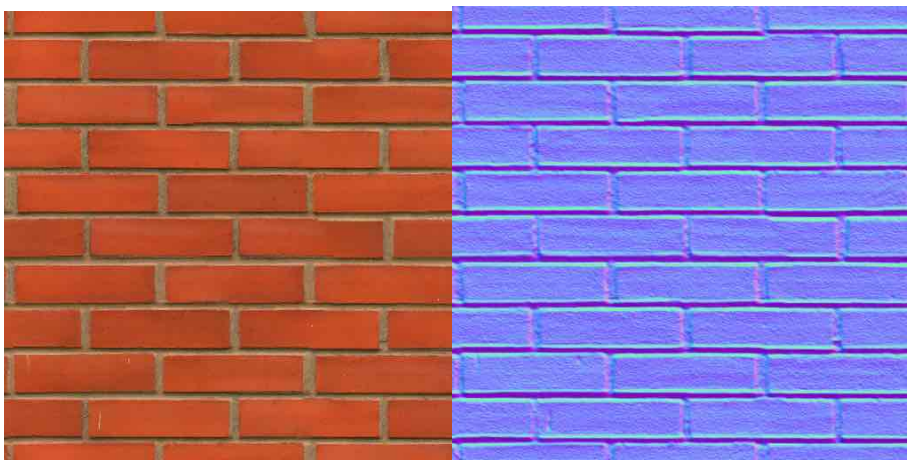
Tekstuureita varten valokuvia otettaessa kannattaa huomioida muutama asia. Kuvat kannattaa ottaa niin suurella pikselitarkkuudella kuin vain mahdollista, jotta yksityiskohdat erottuisivat hyvin. Kameran tulisi olla mahdollisimman suorassa, ja varsinkin suoria linjoja sisältävät kohteet, kuten esimerkiksi tiiliseinä, tulisi kuvata niin kohtisuoraan kuin vain mahdollista, jotta linjat eivät ole eri levyisiä kuvan eri reunoilla. Kameran automaattiasetusta tulisi välttää, jotta esimerkiksi syväterävyys ei jää vahingossa liian pieneksi. Myös objektiivin laatuun kannattaa kiinnittää huomiota, jotta kuvaan ei tulisi turhaa säröä, kuten esimerkiksi kalansilmäefektiä, jossa kuvan reunat pyrkivät kaareutumaan sisäänpäin. Kalansilmäefektiä pystyy välttämään myös kuvaamalla kohteet hieman kauempaa, mutta suuremmalla zoomilla (Ahearn 2016). Kuvassa 11 kamera ei ole ollut ihan kohtisuorassa kuvattavaa kohdetta vasten, jolloin lautojen raot eivät ole täysin pystysuoria. Tällöin myös kuvasta muodostetussa tekstuurissa näkyy lautojen saumoissa virheitä.

KUVA 11. Vino valokuva ja siitä tehty tekstuuri

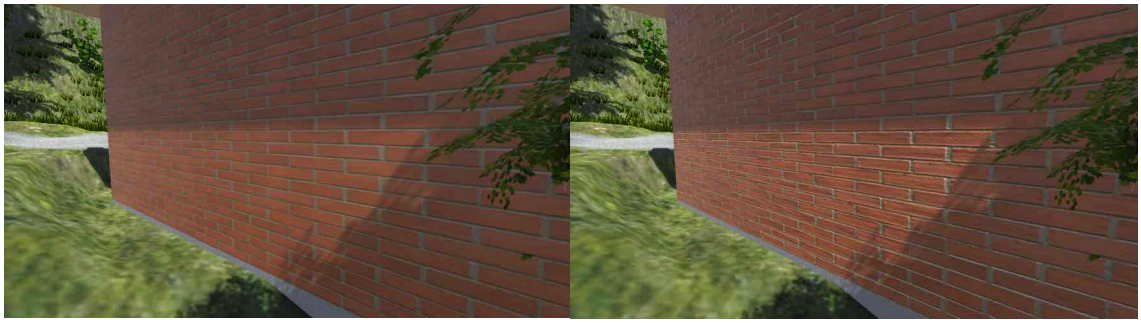
3.3.2 Normaalikartta

Teksturointiin liittyy läheisesti normaalikartta eli normal map. Normaalikartta on tekstuuri, jonka avulla 3d-objektin pintaan voidaan luoda syvyysvaikutelmaa ja korkeuskuviointia (Zeman 2015, 150). Tämän lisäksi normaalikartan avulla voidaan lisätä pieniä yksityiskohtia, kuten naarmuja ja uurteita. Tässä opinnäytetyössä normaalikartat tehtiin PixPlantilla tekstuurien luomisen yhteydessä, mutta tarvittaessa normaalikarttoja voi tehdä myös esimerkiksi Unityllä, Photoshopilla tai useammallakin internetistä löytyvällä verkkosivulla.

Normaalikartan avulla tehty syvyysvaikutelma perustuu pinnan heijastavuuden manipulointiin. Normaalikartta on ikään kuin kartta vektoreita, jossa jokainen pikseli sisältää tiedon siitä, mihin suuntaan pinta kyseisessä kohdassa osoittaa (Unity technologies 2017, Normal map). Näin ollen pinta saadaan vaikuttamaan suoraan edestä katsottaessa hyvinkin eloisalta ja realistiselta, mutta lähempää ja varsinkin pinnan suuntaisesti tarkasteltaessa vaikutelma häviää. Koska normaalikartta ei lisää objektin polygonimäärää, se on suorituskyvyn kannalta tehokas tekniikka (Zeman 2016, 150). Kuvassa 11 on esitelty tiiliseinätekstuuri ja siitä muodostettu normaalikartta, ja kuvassa 12 on esitelty tiiliseinä oikeassa ympäristössään pelimoottorissa ilman normaalikarttaa sekä normaalikartan kanssa.



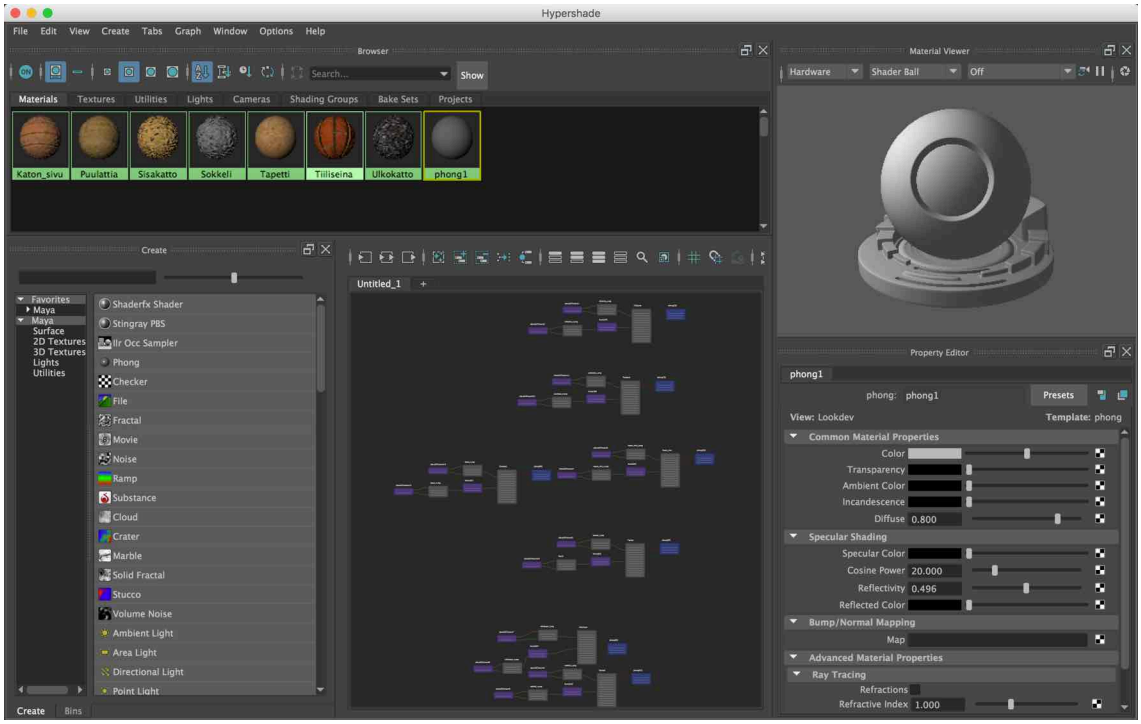
KUVA 11. Tiiliseinätekstuuri (vas.) ja siitä tehty normaalikartta (oik.)



KUVA 12. Vasemmalla tiiliseinä ilman normaalikarttaa ja oikealla normaalikartan kanssa Unityn peliympäristössä.

3.3.3 Materiaalin luominen Maya LT:n Hypershadella

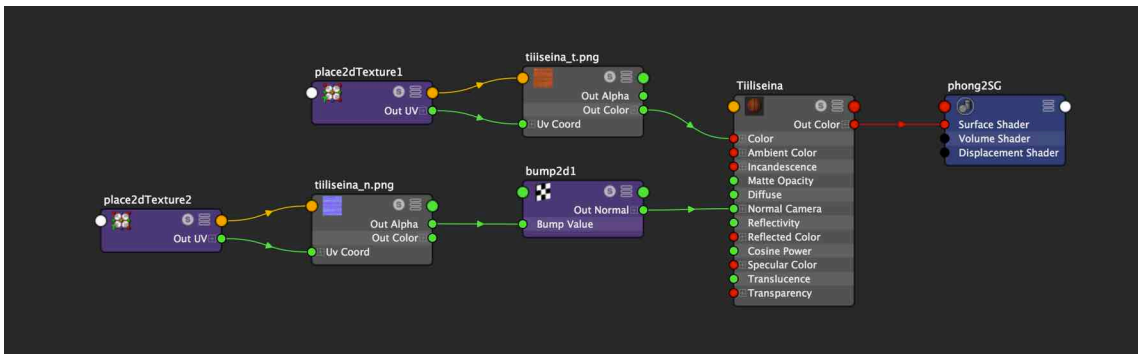
Kun tarvittavat tekstuurit ja normaalikartat oli luotu, seuraava vaihe oli siirtyä takaisin Maya LT:n pariin ja aloittaa talon pintamateriaalien tekeminen. Maya LT:ssä materiaalit luodaan käyttämällä Hypershade-materiaalieditoria (kuva 13). Hypershade on aloittelijan kannalta hieman monimutkaisen oloinen kokonaisuus, mutta perusasiat sen kanssa saa kuitenkin haltuun kohtalaisen nopeasti.



KUVA 13. Maya LT:n Hypershade-materiaalieditori

Hypershaden toiminta pohjautuu solmuihin (node) ja näiden välisiin kytkentöihin. Jokainen yksittäinen ominaisuus on oma solmunsa, ja solmuissa on erilaisia sisään- ja ulostuloja. Solmuja voivat olla esimerkiksi tiedostot, tekstuurit, shaderit ja muut materiaalin osa-alueet, ja Hypershaden keskellä oleva työnäyttö toimii solmujen ja niiden yhteyksien editorina. Kun vasemman reunan selaimesta valitaan esimerkiksi jokin materiaali tai shader, tämä ilmestyy työnäyttöön yhdeksi tai useammaksi solmuksi kyseisen ominaisuuden rakenteesta riippuen. Kun työnäytöstä on valittuna jokin solmu, sen parametreja pääsee säätämään oikeassa reunassa olevan Property Editorin kautta.

Materiaalin tekeminen aloitettiin valitsemalla materiaalille sopiva shader, ja tässä tapauksessa kaikkiin materiaaleihin päädyttiin valitsemaan Phong. Tämän jälkeen materiaaliin lisättiin tekstuuri ja normaalikartta, mutta muuten asetukset jätettiin oletusarvoihinsa. Shader tarjoaisi tämän lisäksi säätömahdollisuuksia esimerkiksi materiaalin heijastavuuden ja läpinäkyvyyden suhteen, mutta oletusarvot toimivat kaikille talon ja tulisijan materiaaleille hyvin. Oheisessa kuvassa 14 on esitelty tiiliseinämateriaalin rakenne Hypershade-editorissa. Kuvasta nähdään, että kyseinen materiaali koostuu tekstuurista, normaalikartasta ja nämä yhdistävästä shaderista.

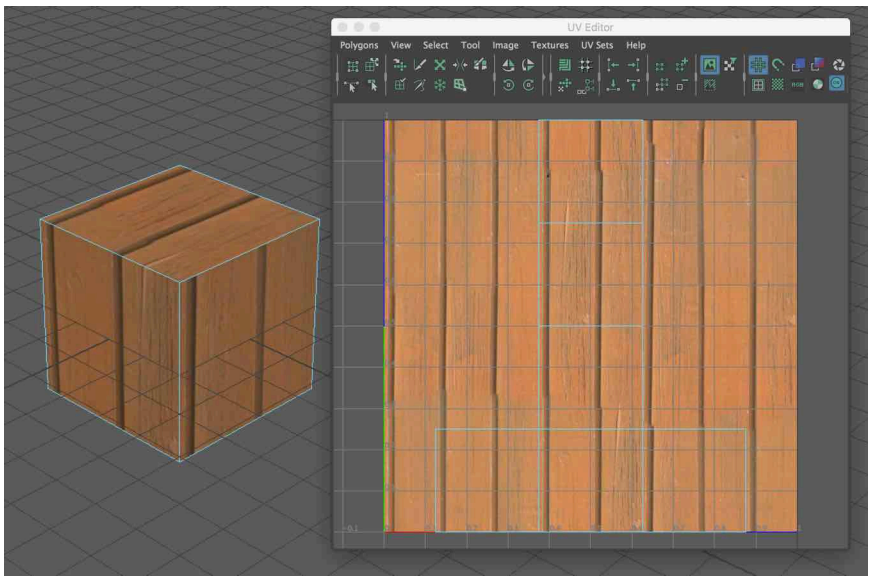


KUVA 14. Tiiliseinämateriaalin rakenne Hypershade-editorissa

3.3.4 UV-mappaus

Kun materiaalit oli saatu tehtyä, ne täytyi sijoittaa talon pinnalle haluttuihin kohtiin. Tämä tehtiin UV-mappauksen avulla. UV-mappauksella tarkoitetaan kaksiulotteisen kuvan asettelua kolmiulotteiselle pinnalle (Ahearn 2016). Termi UV kuvaa kahta UV-avaruuden akselia, eli horisontaalista U-akselia ja vertikaalista V-akselia (Zeman 2015, 163). UV-mappaus täytyy tehdä kaikille 3d-pinnalle sijoitettaville tekstuureille. Yksinkertaisille objekteille UV-mappauksen tekeminen on yleensä helppoa, mutta monimutkaisemmille pinnoille, kuten esimerkiksi ihmis- tai eläinmallille se voi olla todella työlästä ja vaikeaa (Zeman 2015, 164).

Jos käytettävässä tekstuurissa on selkeä kuviointi, kuten esimerkiksi talon tiiliseinän tapauksessa, tekstuurin skaalaaminen ja sijoittaminen 3d-objektin pinnalle on melko helppoa. Jos sen sijaan tekstuuri on epämääräisempi (esimerkiksi betonipinta, sora ym.), sen skaalauksesta ja muodosta voi olla yllättävän vaikeaa saada silmämääräisesti selvää. Tässä kohtaa hyvä apukeino on tehdä UV-mappaus käyttäen jotain selkeää visuaalista tekstuuria ja sijoittaa varsinainen lopullinen tekstuuri pinnalle vasta mappauksen jälkeen. Eräs hyvä tekstuuri tähän käyttöön on ruutukuviointi (Zeman 2015, 164 – 165). Tekstuurien UV-mappaus tehdään UV-editorilla. UV-editorin avulla 3d-objekti hajotetaan kaksiulotteisiin tasoihin, jotka sijoitetaan ja skaalataan halutulla tavalla tekstuurin päälle. Kuvassa 14 olevassa esimerkissä tasot näkyvät sinisinä ruutuina tekstuurin päällä.

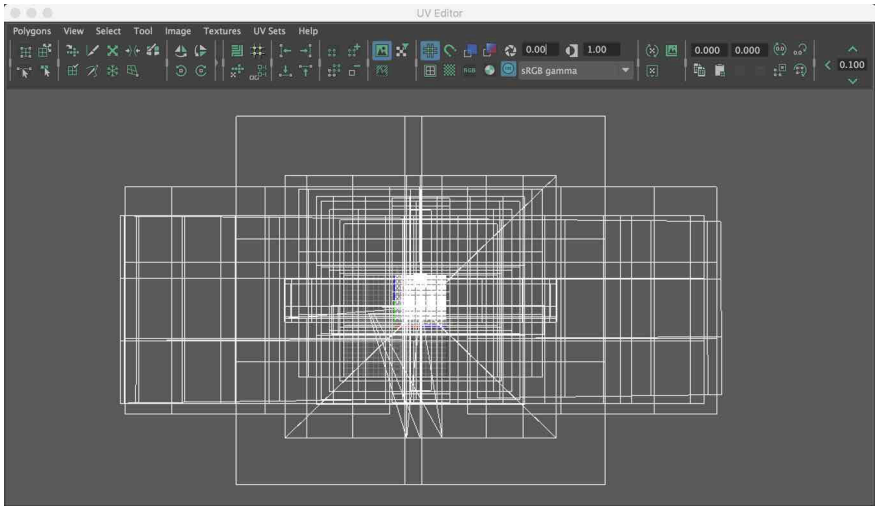


KUVA 14. UV-editori ja esimerkkikuutio

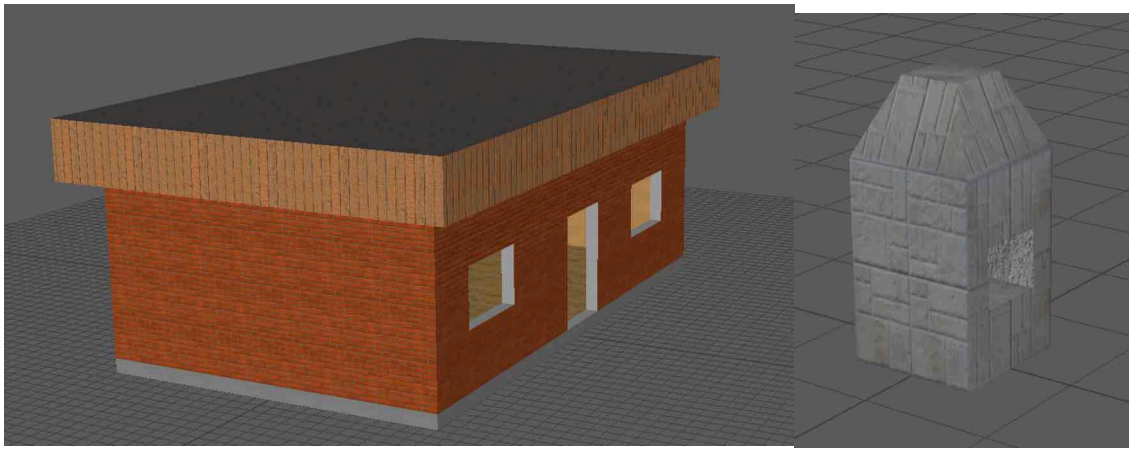
Opinnäytetyön 3d-objektit olivat siinä mielessä yksinkertaisia kohteita, että ne sisälsivät vain suoria pintoja. Silti UV-mappaus osoittautui haasteelliseksi, ja loppujen lopuksi työ saatiin tehtyä enemmänkin yrityksen ja erehdyksen kuin syvän ymmärtämyksen kautta. Talon kohdalla UV-mappauksessa toimivaksi prosessiksi osoittautui seuraava toimintamalli:

1. Valitaan kaikki saman suuntaiset samalla tekstuurilla päällystettävät pinnat.
2. Sijoitetaan niihin haluttu materiaali.
3. Jos materiaalit eivät mappaudu oikein, eli ovat esimerkiksi poikittain tai väärässä kulmassa, korjataan asia kokeilemalla mapata tekstuurit jonkin toisen akselin suuntaisesti UV-editorin Planar-toiminnon avulla. Tämän jälkeen lopputulos on yleensä oikea.
4. Lopuksi skaalataan tekstuuri silmämääräisesti oikean kokoiseksi.

Tällä tavalla lopputulos saatiin riittävällä tarkkuudella silmää miellyttäväksi. Ongelmaksi muodostui lähinnä se, että UV-editorin näkymä meni niin hirveäksi sotkuksi, että asioiden korjaaminen sen avulla olisi ollut liki mahdotonta (kuva 15). Talon teksturoinnin jälkeen täytyi vielä teksturoida tulisija, mutta tämä oli huomattavan helppoa taloon verrattuna. Kuvassa 16 on esitettynä talo ja tulisija valmiiksi teksturoituina.



KUVA 15. UV-editorin näkymä talon teksturoinnin jälkeen



KUVA 16. Valmiiksi teksturoitu talo ja tulisija

Jälkikäteenkään asiaa tutkittaessa ei varsinaisesti löytynyt mitään selkeää vastausta, miten kyseisen kaltaisen montaa eri toistuvaa materiaalia sisältävän tapauksen kanssa olisi järkevintä toimia. Epäselväksi jäi, olisiko talo esimerkiksi kannattanut jakaa useampiin objekteihin ja olisiko tekstuurit pitänyt skaalata jo kuvankäsittelyvaiheessa sopivampaan kokoon. Näin ollen UV-mappauksesta jäi se tuntuma, että asiaan pitää tulevaisuudessa ehdottomasti perehtyä vielä tarkemmin.

4 UNITY JA PELIMAAILMAN LUOMINEN

4.1 Unity

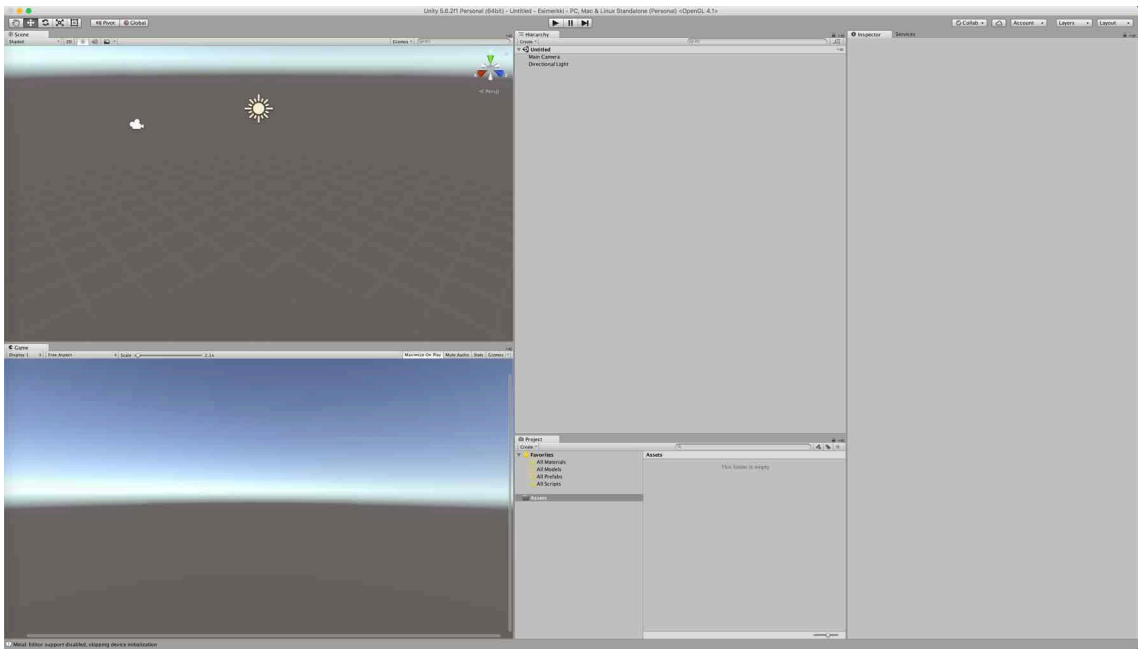
Opinnäytetyössä käytettäväksi pelimoottoriksi valikoitui Unity, koska se osoittautui kyselytutkimuksen perusteella ylivoimaisesti suosituimmaksi pelimoottoriksi suomalaisten peliyritysten keskuudessa. Unityllä luotiin yksinkertainen 3d-maailma, jonka avulla tutustuttiin 3d-pelikehityksen ja -tekniikoiden alkeisiin. Unity on Unity Technologiesin kehittämä pelimoottori, jonka avulla pystyy julkaisemaan yli 25:lle eri kohdealustalle. Siitä on saatavilla useampaa erilaista lisenssiä, joista tässä opinnäytetyössä käytettiin ilmaista Personal-lisenssiä, joka maksuttomuudestaan huolimatta tarjoaa käyttöön pelimoottorin kaikki ydinominaisuudet. Käytetty versio pelimoottorista oli 5.62.

4.1.1 Unityn perusnäkymä

Unityn perusnäkymä on käyttäjän muokattavissa, mutta sen oleelliset osat ovat Scene-näkymä, pelinäkymä, hierarkiaikkuna, projekti-ikkuna ja Inspector-ikkuna (kuva 17):

- Scene-näkymä on varsinainen editointi-ikkuna, jossa 3d-maailman ja peliobjektien muokkaaminen tapahtuu.
- Pelinäkymässä maisema (scene) näkyy siinä muodossa, miten se kuvautuu pelimaailman kameran kautta.
- Hierarkiaikkuna sisältää listan kaikista pelimaailman objekteista.
- Projekti-ikkuna sisältää kaikki projektiin kuuluvat assetit.
- Inspector-ikkunassa voidaan säätää valitun peliobjektin ominaisuuksia ja komponentteja.

Tämän lisäksi ylärivin työkalupalkissa on näkymän liikutteluun tarkoitettuja työkaluja sekä muun muassa Play-painike, jonka avulla toteutettavaa peliä pystyy testaamaan suoraan pelinäkymä-ikkunassa.



KUVA 17. Unityn perusnäkymä

4.1.2 Assetit

Assetit ovat mitä tahansa peliobjekteja, skriptejä tai muita resursseja, joita voidaan käyttää Unityssä. Assetteja tulee jonkin verran Unityn mukanakin, mutta niitä voi hankkia lisää Asset Storen kautta, johon pääsee valitsemalla valikosta Window -> Asset Store. Tämän opinnäytetyön 3d-maailmassa hyödynnettiin runsaasti valmiita assetteja.

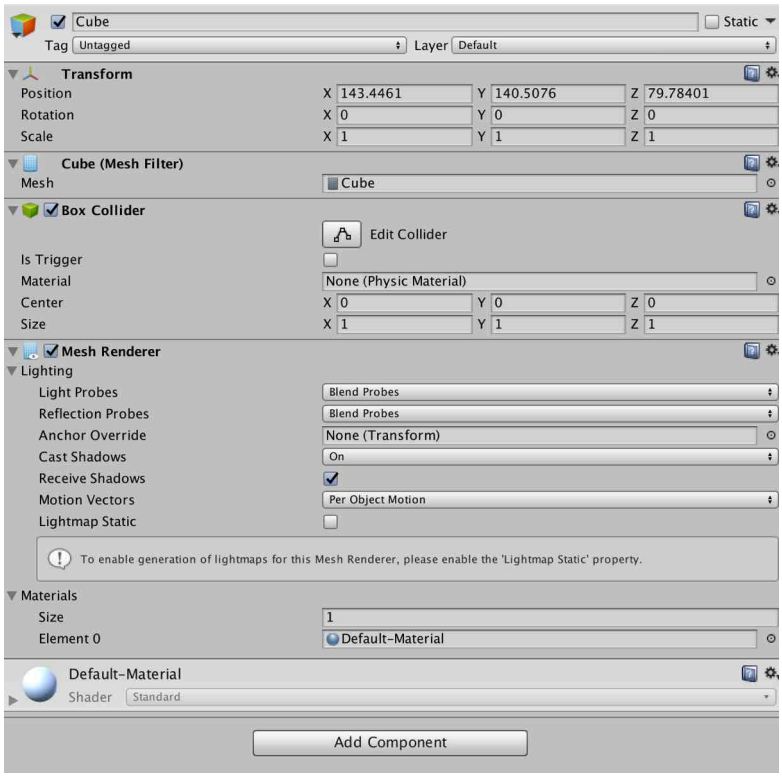
4.1.3 Objektit ja komponentit

Jokainen scene sisältää oletusarvoisesti kaksi objektia, jotka ovat Main Camera ja Directional Light. Main Camera on nimensä mukaisesti scenen pääkamera, jonka kautta näkymä renderöidään pelinäkymä-ikkunaan. Directional Light -objekti matkii auringonvaloa ja tuottaa scenen valaistuksen.

Unityn avulla pystyy lisäämään ja muotoilemaan 3d-perusobjekteja kuten kuutioita, palloja, sylintereitä ja tasoja, mutta varsinaista 3d-mallintamista sen avulla ei voi tehdä. Objekti lisätään sceneen valitsemalla valikosta GameObject -> 3D Object -> haluttu objekti. Objektit koostuvat komponenteista, ja lisätty 3d-objekti koostuu oletuksena seuraavista komponenteista (kuva 18):

1. Transform: Tämä määrittelee objektin sijainnin, asennon ja koon 3d-avaruudessa XYZ-koordinaatteina.
2. Mesh: Tämä on varsinainen 3d-objektin polygonimalli.
3. Collider: Collider määrittelee objektin fyysiset törmäysrajat. Toisin sanoen collider on elementti, joka mahdollistaa objektin törmäämisen toiseen colliderilla varustettuun objektiin. Colliderin avulla objektista voidaan myös tehdä Trigger, josta enemmän jäljempänä.
4. Renderer: Renderer määrittelee sen, miten objekti piirretään eli renderöidään ruudulle. Jos Rendererin ottaa pois päältä, objekti muuttuu näkymättömäksi.

Näiden lisäksi objektiin voi lisätä komponentteja Add Component -painikkeen avulla.



KUVA 18. Kuutio ja sen komponentit Inspector-ikkunassa

Colliderien suhteen olisi suositeltavaa, että ne tehtäisiin objektin ympärille käyttäen 3d-perusmuotoja, kuten esimerkiksi kuutioita, palloja ja sylintereitä. Colliderit voidaan tehdä myös tarkasti 3d-objektin ulkopintoja mukailevaksi Mesh Collidereiksi, mutta näitä tulisi välttää, koska ne vaikuttavat suorituskykyyn negatiivisesti (Unity technologies 2017, Colliders).

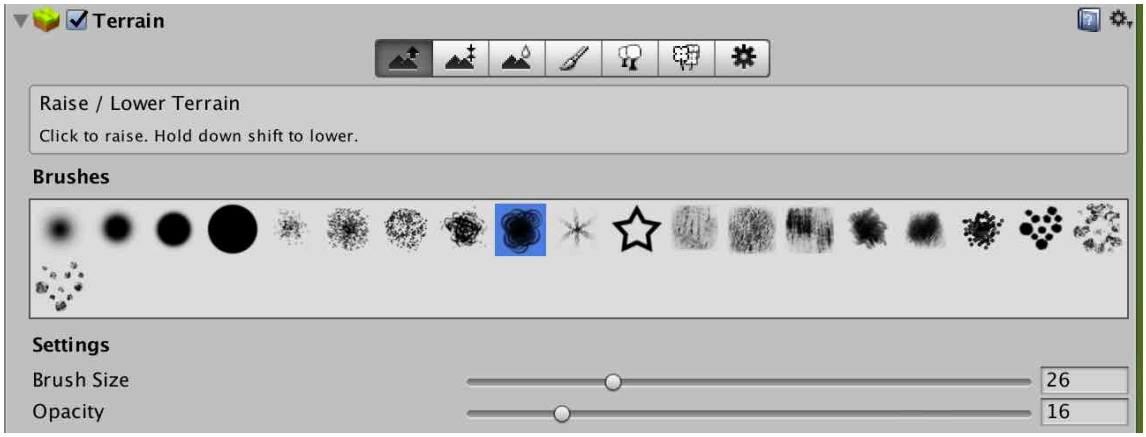
4.2 Maaston muotoilu

Luontotyyppisen pelimaailman luomiseen käytetään Terrain Editoria, ja uusi maasto-objekti luodaan valitsemalla GameObject -> 3d Object -> Terrain. Terrain-objektin aseuksissa on runsaasti erilaisia säätömahdollisuuksia koosta aina valaistukseen ja tuulen voimakkuuteen asti, mutta niihin ei tässä vaiheessa puututa sen tarkemmin.

Maastoa muotoillaan ja siihen lisätään objekteja ja yksityiskohtia erilaisten sivellintyökalujen avulla maalamalla hieman samaan tyyliin kuin esimerkiksi kuvankäsittelyohjelmissa. Siveltimiä on eri muotoisia ja niiden kokoa ja peittävyyttä pystyy säätämään (kuva 19). Erilaisia siveltimiä voi myös tehdä itse kuvankäsittelyohjelmalla (Gerasimov 2015).

Maaston muotoiluun on tarjolla kolme eri toimintoa:

- Maaston muodon ja korkeuden vapaa maalaaminen.
- Maaston muodon maalaaminen tiettyyn kohdekorkeuteen. Kohdekorkeuden voi määrittää joko suoraan numeroarvona tai valita hiiren avulla maasto-objektista osoittamalla.
- Muodon tasoittaminen ja pehmentäminen. Tämä työkalu pyrkii tasoittamaan ja pehmentämään maaston muotoja ympäristöä mukaileviksi.

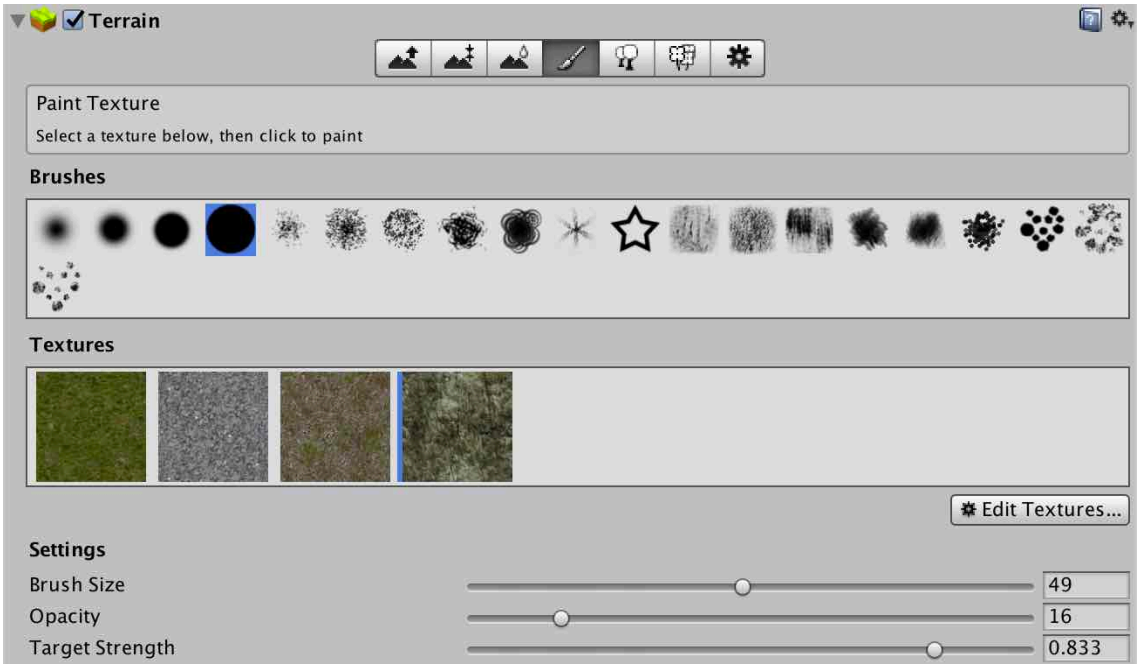


KUVA 19. Terrain editorin siveltimet ja työkalut

Ensimmäinen maasto-objektiin lisätty tekstuuri tulee automaattisesti koko objektin perustekstuuriksi, ja tähän valittiin Unityn omasta asset-kirjastosta ruohotekstuuri. Tämä helpotti maaston muotojen hahmottamista huomattavasti verrattuna pelkkään yksiväriseen pintaan. Huomioitavaa on, että jos projektin luomisvaiheessa ei ole valinnut Unityn assetteja mukaan projektiin, ne täytyy tuoda erikseen (Assets -> Import Package -> …).

Tässä vaiheessa olisi myös hyvä kiinnittää huomiota toteutettavan maailman mittakaavaan. Unityssä 1 yksikkö = 1 metri (Thorn 2017). Käytännössä tätä on muotoiluvaiheessa pelkällä silmällä vaikea hahmottaa, jos maastossa ei ole vielä mitään objekteja, joita voisi käyttää apuna. Eräs keino on lisätä maailmaan jokin 3d-perusobjekti, esimerkiksi 1*1*1-kokoinen kuutio, johon muotoiltavan maailman kokoa voi suhteuttaa.

Esimerkkimaailmaan muotoiltiin pienten vuorien ympäröimä laakso, jonka oli tarkoituksena toimia 3d-maailman ympäristönä, johon myöhemmässä vaiheessa sijoitettaisiin muut objektit. Lisäksi laaksoon muotoiltiin monttu, josta tehtäisiin lampi. Lopuksi maaston pintaan lisättiin erilaisia tekstuureita elävöittämään maastoa. Esimerkiksi vuoret pinnoitettiin kivipintaisella tekstuurilla, ja soratekstuurilla tehtiin laaksoon johtava polku. Tekstuurit maalattiin maaston pintaan Paint Texture -ominaisuuden avulla (kuva 20). Maasto-objektin valmis pinta on kuvassa 21.

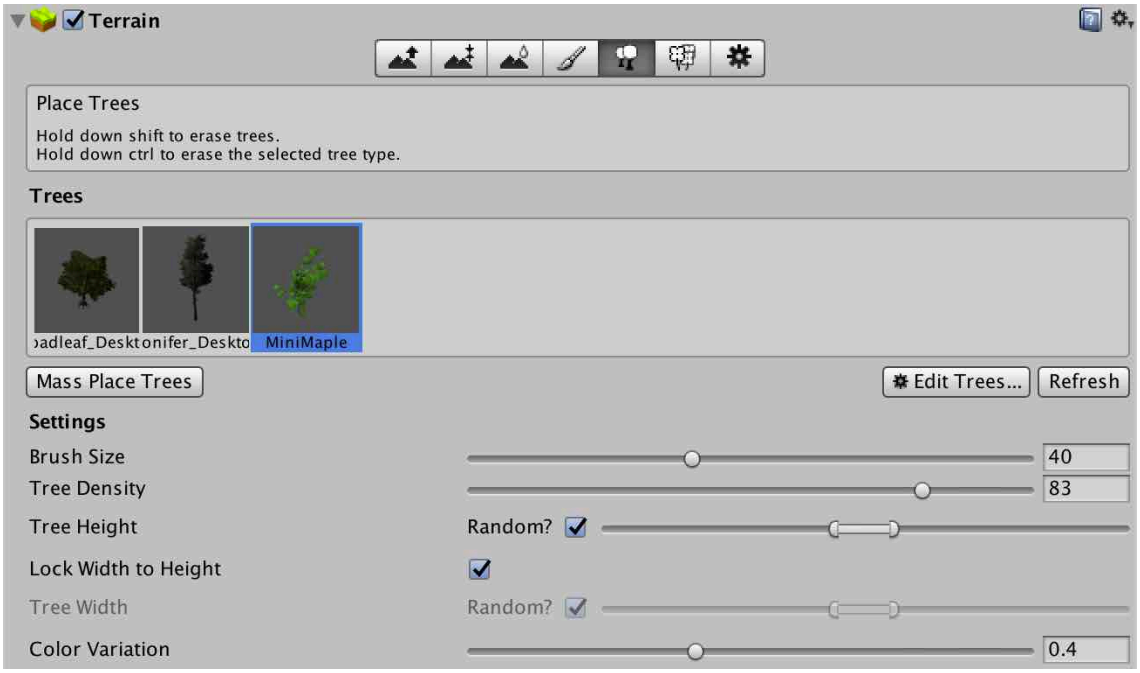


KUVA 20. Paint texture –toiminto ja maastossa käytetyt tekstuurit

KUVA 21. Muotoiltu pelimaailman maasto-objekti

4.3 Puut ja kasvit

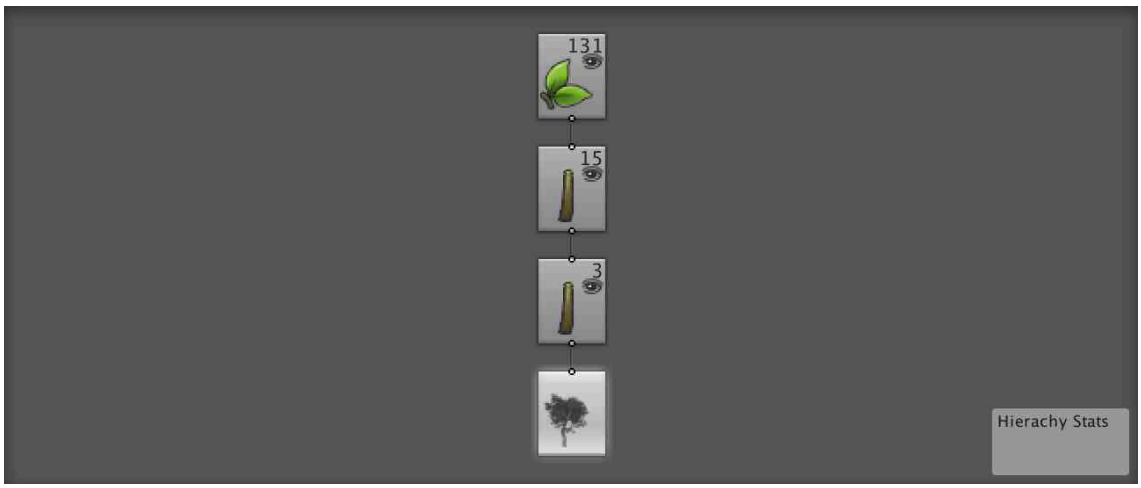
Kun maasto oli muotoiltu, siihen lisättiin puita ja ruohikkoa. Puut maalattiin maaston käyttäen Terrain Editorin Place Trees -työkalua (kuva 22). Place Trees -työkalun asetuksista voi säätää esimerkiksi käytettävän siveltimen kokoa sekä sijoitettavien puiden tiheyttä ja pituusvaihtelua. Jos halutaan sijoittaa puita tasaisesti koko maasto-objektin alueelle, voidaan käyttää Mass Place Trees -ominaisuutta. Puina käytettiin kahta Unityn omaa peruspuumallia, ja näiden lisäksi luotiin yksi oma puu käyttäen apuna Unityn puueditoria.



KUVA 22. Place Trees -toiminto ja pelimaailmassa käytetyt puut

Oman puun tekeminen aloitetaan valitsemalla valikosta GameObject -> 3D Object -> Tree. Puueditorin avulla määritellään ensiksi puun rakenne, eli haarojen ja niiden alihaarojen lukumäärä sekä se, missä haaroissa lehtiryhmät sijaitsevat (kuva 23). Tämän jälkeen jokaisen haaran ja lehtiryhmän parametreja säätämällä luodaan puusta halutun mallinen. Säädettäviä parametreja on todella paljon mukaan lukien puun muoto, pituus, tiheys, käppyräisyys ja esimerkiksi aurinkoa kohti taipuminen. Periaatteessa puueditorilla tehdyn puun perusrakenne perustuu satunnaisgenerointiin siemenen (Seed) perusteella, mutta lisäksi puun jokaista haaraa ja oksaa voi säätää myös käsin. Lopuksi jokaiseen haaraan ja lehtiryhmään valitaan haluttu materiaali.

Puun tekeminen vaatii yritystä ja erehdystä, ja parhaaseen lopputulokseen pääsee yleensä kokeilemalla. Esimerkkipelimaailmaan tehtiin vaahteran kaltainen puu, jonka runko oli kolmihaarainen ja jossa oksia oli yhteensä 15 ja lehtiä 131 (kuva 24). Puueditori osoittautui hieman bugiseksi tapaukseksi, ja muutaman kerran se hukkasi esimerkiksi materiaalit lopullisesti, jolloin puun tekeminen jouduttiin aloittamaan alusta.



KUVA 23. Puueditorin hierarkianäkymä.

KUVA 24. Esimerkki puueditorilla luodusta ”vaahterasta” sijoitettuna valmiiseen pelimaailmaan.

Puiden lisäämisen jälkeen lisättiin ruohikko. Ruohikon sijoittamiseen käytettiin Terrain Editorin Paint Details -ominaisuutta, jonka avulla voi maalata maisemaan yksityiskohtia samaan tyyliin kuin puitakin. Yksityiskohtia voivat olla ruohikon lisäksi esimerkiksi kivet, kukat ja sienet. Paint Details tukee sekä kaksi- että kolmiulotteisia objekteja (Gerasimov 2015). Ruohikkoa lisättäessä tutustuttiin myös billboardeihin ja billboarding-ilmiöön (kuva 25). Billboard on kaksiulotteinen eli litteä objekti, tarkemmin ottaen suorakulmio johon on sijoitettu tekstuuri (Ganovelli ym. 2015, Billboarding). Billboarding taas

tarkoittaa billboardien ominaisuutta, jossa billboardit kääntyvät aina osoittamaan kameraa kohti riippumatta siitä, mistä suunnasta niitä katsotaan. Detaljieditori antaa mahdollisuuden valita, onko billboarding käytössä vai ei. Jos billboarding otetaan pois päältä, billboardit osoittavat aina kiinteästi johonkin tiettyyn suuntaan eivätkä käänny kameraa kohti. Tämä kuitenkin näytti kummalliselta ainakin tässä opinnäytetyössä käytettyjen ruohikkotekstuurien kanssa.

KUVA 25. Samat ruohotupsut kuvattuna eri kulmista. Molemmissa kuvissa ruohikko on samassa kulmassa kameraan nähden, mutta eri kulmassa pelimaailmaan nähden.

4.4 Kameran liikuttelu

Jotta kameraa voisi liikuttaa 3d-esimerkkimaailmassa, toteutettiin yksinkertainen SpectatorController-skripti (liite 2). Unity tukee skriptaamista sekä C#- että UnityScript-kielillä, joista tässä opinnäytetyössä käytettiin ensimmäistä. Kameraa varten luotiin oma Spectator-peliobjekti, johon lisättiin kamera ja kameran liikuttelua hoitava skripti. Spectator-peliobjektina toimi läpinäkyvä pallo (GameObject -> 3D Object -> Sphere), johon lisättiin Rigidbody-komponentti, jonka avulla saatiin Unityn fysiikkamoottori käyttöön. Tämä mahdollistaa Spectator-objektin liikuttelun AddForce-funktion avulla.

SpectatorController-skripti koostui kahdesta funktiosta, Start ja FixedUpdate. Start-funktio suoritetaan kertaalleen peliä käynnistettäessä, ja siihen sijoitettiin alustustoimenpiteet, eli tässä tapauksessa kameran alkusijainnin koordinaattien lukeminen. FixedUpdate suoritetaan tasaisin väliajoin pelin ollessa käynnissä, joten siihen sijoitettiin kaikki liikkumiseen liittyvä toiminta. FixedUpdate-funktiossa luetaan ensiksi näppäimistön ja hiiren liik-

keet Input.GetAxis-funktion avulla. Tämän jälkeen kameran asentoa muutetaan transform.Rotate-funktion ja sijaintia Rigidbody.AddForce-funktion avulla. Kameran asennon muuttamisen jälkeen sen z-akseli nollataan, jotta kamera kuva pysyy vaakatasossa pelimaailman koordinaatistoon nähden. Kameran liikuttelussa tulee huomioida se, että kameran halutaan liikkuvan suhteessa omaan koordinaatistoonsa. Näin ollen liikesuunta pitää muuttaa globaaliin koordinaatistoon sopivaksi Transform.TransformDirection-funktion avulla.

4.5 Omien 3d-objektien lisääminen pelimaailmaan

Tässä vaiheessa pelimaailmaan tuotiin aikaisemmin Maya LT:llä mallinnetut objektit. Objektien siirtoon käytettiin Maya LT:n Send to Unity -toimintoa, joka tallentaa 3d-mallit Unityn ymmärtämässä FBX-muodossa. Mallit kannattaa tallentaa suoraan Unity-projektin asset-kansioon, jolloin niiden pitäisi ilmestyä automaattisesti Unityn projekti-ikkunan näkymään. Send to Unity -toiminnon yhteydessä käytettiin ”Autodesk Media & Entertainment”-preset-asetusta, joka toimi muuten hyvin, mutta pari asiaa vaati hieman lisäsäätöä, joista ensimmäinen oli 3d-mallien mittakaava. Oletusarvoisesti Maya LT käyttää yksikköinä senttimetrejä, kun taas Unityssä yksikköinä on metrit. Tämä aiheuttaa sen, että Maya LT:llä mallinnetut objektit ovat Unityssä mittakaavaltaan sadasosa alkuperäisestä. Helpoimmin tämän saa korjattua muuttamalla Unityn Import-asetuksista mittakaavaksi (Scale Factor) 100 oletusarvona olevan 1:n sijaan. Import-asetuksia voi säätää Inspector-ikkunassa silloin, kun objekti on valittuna projekti-ikkunassa. Toinen huomio oli se, että normaalikartat eivät automaattisesti tulleet osaksi talon materiaaleja, vaan ne täytyi lisätä materiaaleihin käsin. Tämä tapahtui valitsemalla projekti-ikkunasta haluttu materiaali ja tämän jälkeen lisäämällä oikea normaalikartta Inspector-ikkunassa olevaan Normal Map -kohtaan.

Myös Unityllä on mahdollista luoda tekstuureista normaalikarttoja. Tämä tapahtuu siten, että valitaan projekti-ikkunasta haluttu tekstuuri, muutetaan sen tyypiksi Inspector-ikkunassa Normal Map ja painetaan Apply-painiketta. Unityssä normaalikartan säätömahdollisuudet ovat kuitenkin hyvin suppeat rajoittuen pelkkään Bumpiness-parametriin.

Kun talo oli sijoitettu paikalleen, sijoitettiin tulisija sen sisälle (kuva 26). Tulisijaan lisättiin sylinterin mallinen piippu (GameObject -> 3D Object -> Cylinder) sekä liekki ja savuefekti käyttäen apuna hiukkassysteemejä, joista enemmän jäljempänä. Tämän jälkeen edellä mainituille objekteille lisättiin vielä colliderit, jotta kamera ei menisi niiden läpi. Talon tapauksessa päädyttiin käyttämään Mesh Collideria, vaikka tämä ei suorituskyvyn kannalta olekaan suositeltava vaihtoehto.

KUVA 26. Talo ulkoa ja sisältä valmiissa pelimaailmassa. Tulisijaan on lisätty liekki, piippu ja savuefekti.

4.6 Muut objektit

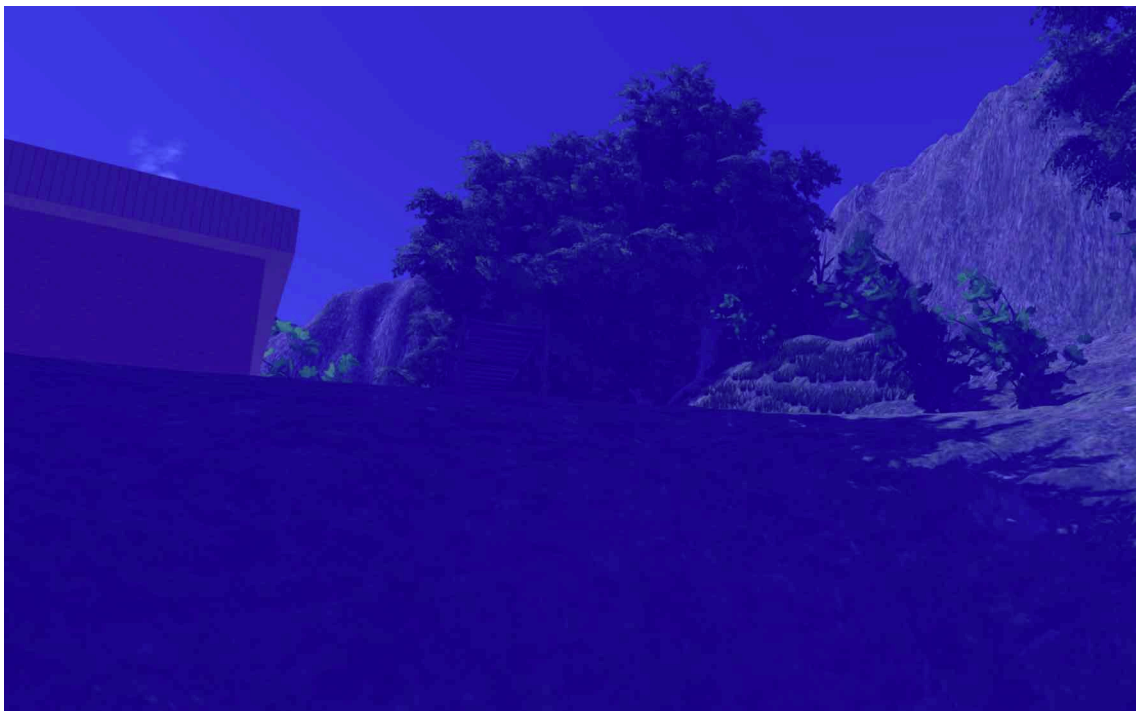
3d-maailmaan sijoitettiin vielä täytteeksi muutamia objekteja, jotta maisema ei näyttäisi liian tyhjältä. Näitä olivat muun muassa keskiaikainen kartano, kaivo, penkki, muutamat kivet ja lampeen menevä laituri. Kartano asetettiin alueen reunalle, ja sen ympärille tehtiin kiviaita. Lisäksi aidan sisäpuoliselle piha-alueelle tehtiin kivetys, ja kaivo sijoitettiin kartanon viereen. Lampeen menevän laiturin läheisyyteen sijoitettiin penkki ja kivet, ja lisäksi penkin eteen tehtiin nuotio. Kaikki toteutettiin käyttäen valmiita assetteja, ja tämän jälkeen 3d-maailman perusrakenne alkoi olla valmis.

4.7 Vesi

Veden pinnan tekeminen alusta alkaen on melko monimutkainen prosessi, joten tässä kohtaa päädyttiin käyttämään Unityn omaa valmista WaterProDaytime-assettia (kuva 27). Sukellettaessa vedenpinnan alapuolelle kameran näkymä muuttuu sinisävyiseksi,

jolla matkitaan yleistä peleistä tuttua efektiä (kuva 28). Kameran kuvan värin muuttaminen toteutettiin lisäämällä kameraan Color Correction Curves -efekti, jonka asetuksista voimistettiin sinistä värisävyä. Efekti laitetaan päälle aina pinnan alapuolelle mentäessä, ja sitä ohjataan omalla WaterArea-skriptillä (liite 3), joka on liitetty Spectator-objektin komponentiksi. Lampeen sijoitettiin näkymätön kuutio, joka toimii niin sanottuna triggerinä eli ”kytkimenä”, jonka avulla voidaan aktivoida erilaisia asioita. Aina kun kamera menee kuution alueelle, skripti laittaa kameraefektin päälle, ja vastaavasti kuution alueelta poistuttaessa skripti sammuttaa efektin. Kuutio on tehty näkymättömäksi ottamalla sen ominaisuuksista Mesh Renderer -komponentti pois päältä. Vastaavasti sen colliderin ominaisuuksista on laitettu ”Is Trigger”-kohta aktiiviseksi, jolloin se toimii ainoastaan triggerinä eivätkä muut objektit voi törmätä siihen.

KUVA 27. Veden pinta



KUVA 28. Maisemaa vedenpinnan alta katsottaessa

4.8 Tuli

Tulen ja savun tekemiseen käytettiin Unityn hiukkassysteemiä. Hiukkaset ovat pieniä 2d-kuvia tai 3d-objekteja, joita tuotetaan suuri määrä ja saadaan tämän avulla aikaiseksi erilaisia efektejä, kuten esimerkiksi savua, tulta tai sadetta. Kaikilla hiukkassysteemillä voidaan ajatella olevan kolme keskeistä ominaisuutta: emitteri, hiukkanen ja elinikä (Thorn 2015, Particle systems). Hiukkaset syntyvät emitterissä. Emitterin ominaisuuksista voidaan säätää esimerkiksi lähtevien hiukkasten lukumäärää ja hiukkasparven muotoa. Hiukkasella on elinikä, ja siihen liittyvillä asetuksilla voidaan määritellä muun muassa miten hiukkasen väri, läpinäkyvyys, muoto tai koko muuttuu eliniän aikana tai miten hiukkasen nopeus käyttäytyy tänä ajanjaksona. Kun emitteri lähettää hiukkasia suuren parven, saadaan aikaiseksi erilaisia efektejä. Hiukkassysteemi tarjoaa suuren määrän erilaisia säätömahdollisuuksia, ja tulen ja savun tekeminen olikin melko pitkälti yritystä ja erehdystä. Kokeilemalla ja internetistä löytyvien ohjeiden avustamana saatiin aikaiseksi kuitenkin ihan kelvollinen lopputulos.

Savupiipusta tuleva savu on hyvin yksinkertainen hiukkassysteemi. Malliltaan se on hieman leviävä kartio, jossa savuhiukkaset etenevät vakionopeudella ja muuttuvat vähitellen

läpinäkyviksi. Nuotio koostuu kolmesta eri hiukkassysteemistä, joista yksi on liekit, toinen savu ja kolmas liekeistä lentävät kipinät. Liekkien hiukkassysteemi on muodoltaan leviävä kartio, jossa hiukkasten nopeus kasvaa ylöspäin mentäessä. Hiukkaset muuttuvat läpinäkyviksi ennen katoamistaan samaan tapaan kuin savupiipusta tulevassa savussakin. Liekeissä on lisäksi kipinöitä. Kipinät kulkevat vakionopeudella, mutta toisin kuin savun ja liekkien tapauksessa, niitä ei häivytetä muuttamalla läpinäkyviksi, vaan ne hävitetään pienentämällä niiden kokoa. Nuotion savuefekti toimii samoin kuin savupiipun tapauksessakin, ainoastaan mittakaava on suurempi. Sekä savun että nuotion tapauksessa hiukkasina käytettiin Unitystä löytyviä vakioassetteja. Valmiit hiukkassysteemeillä toteutetut efektit on esitelty kuvassa 29.

KUVA 29. Savu- ja tuliefekti

4.9 Viimeistely, kameraefektit ja taivas

Lopuksi pelimaailma viimeisteltiin lisäämällä kameraan muutama efekti ja valitsemalla sopiva Skybox taivaaksi. Kameraefektit ovat skriptejä, jotka lisätään kameraobjektiin objektiin komponenteiksi Inspector-ikkunan kautta (Add Component -> Image Effects -> …). Kamerassa kokeiltiin erilaisia Unityn tutoriaaleissa suositeltuja efektejä, ja niistä

päädyttiin käyttämään seuraavia: Antialiasing, Screen Space Ambient Obscurance ja Global Fog. Antialiasointi eli reunanpehmennys nimensä mukaisesti pehmentää objektien reunoja vähentäen pikselöitymisestä johtuvaa sahalaitaisuutta. Screen Space Ambient Obscurance tummentaa reikien ja saumojen kohtia sekä lähellä toisiaan olevia pintoja. Esimerkiksi talon sisänurkat muuttuivat aidomman näköisiksi tämän avulla. Global Fog lisää yleisen sumuefektin, joka muuttaa kauempana kamerasta olevat kohteet utuisemmiksi.

Skybox on elementti, jonka avulla toteutetaan taivas ja horisontti. Asset Storesta löytyi miellyttävä pilvitekstuurein varustettu Skybox (kuva 30), joten tähän ei uhrattu sen enempää omaa aikaa. Skybox asetettiin paikalleen valaistusasetuksista (Window -> Lighting -> Settings), jossa kohtaan ”Skybox Material” valittiin haluttu materiaali.

KUVA 30. Taivas ja horisontti

4.10 Renderöinnin optimointia Occlusion cullingin avulla

Renderöinti on yksi keskeisimmistä tietokonegrafiikkaan liittyvistä käsitteistä varsinkin 3d-grafiikasta puhuttaessa, ja yksinkertaistetusti sillä tarkoitetaan yksittäisen ruudulle piirrettävän kuvan muodostamista. 3d-grafiikka on pohjimmiltaan pelkkiä numeroita ja geometriaa, joka ei sellaisenaan sisällä mitään ihmissilmälle ymmärrettävää kuvatietoa.

Jotta numerot ja geometria saadaan ihmissilmän ymmärtämään muotoon, ne täytyy renderöidä kaksiulotteiseksi kuvaksi käyttäen erilaisia algoritmeja ja matemaattisia kaavoja. 3d-grafiikan renderöinti onkin hyvin monimutkainen operaatio (Zeman 2015, 111).

3d-näkymää renderöitäessä tapahtuu monesti ylipiirtoa (Dickinson 2015, Fill rate). Tämä tarkoittaa sitä, että jotkin jo renderöidyt kohdat jäävät jonkin toisen, yleensä lähempänä kameraa olevan objektin peittoon. Tämä heikentää turhaan suorituskykyä. Ylipiirtoa voidaan vähentää käyttämällä occlusion culling -tekniikkaa. Occlusion cullingin idea yksinkertaistetusti on se, että pelimaailma jaetaan pienempiin soluihin, ja jokaisesta solusta kuvataan näkymä virtuaalikameran avulla. Ne solut, jotka jäävät kustakin solusta katsottaessa kokonaan pimentoon, jätetään renderöimättä. Tämä operaatio toistetaan jokaiselle solulle koko pelimaailman alueella (Dickinson 2015, Occlusion culling). Lisäksi käyttäjä voi itse manuaalisesti määritellä vastaavia alueita eli occlusion areoita. Nämä mahdollistavat occlusion cullingin myös liikkuville objekteille (Unity technologies 2017, Occlusion culling).

Unityssä kaikkien objektien, jotka halutaan ottaa occlusion cullingiin mukaan, täytyy olla tyypiltään Occluder Static ja Occludee Static. Nämä ominaisuudet laitetaan päälle objektin ollessa valittuna hierarkiaikkunassa Inspector-ikkunan Static-valintaruudun viereisestä valikosta. Occluder Static tarkoittaa, että objekti on mahdollisesti peittävä. Yleensä kaikki staattiset objektit ovat Occluder Static -tyyppisiä, paitsi hyvin pienet objektit, joiden ei haluta peittävän mitään kokonaan missään tilanteessa tai läpinäkyvät objektit, kuten esimerkiksi vedenpinta ja lasi. Occludee Static tarkoittaa, että objekti on peittyvä, eli jos se jää kokonaisuudessaan peittoon, sitä ei renderöidä (Unity technologies 2017, Occlusion culling).

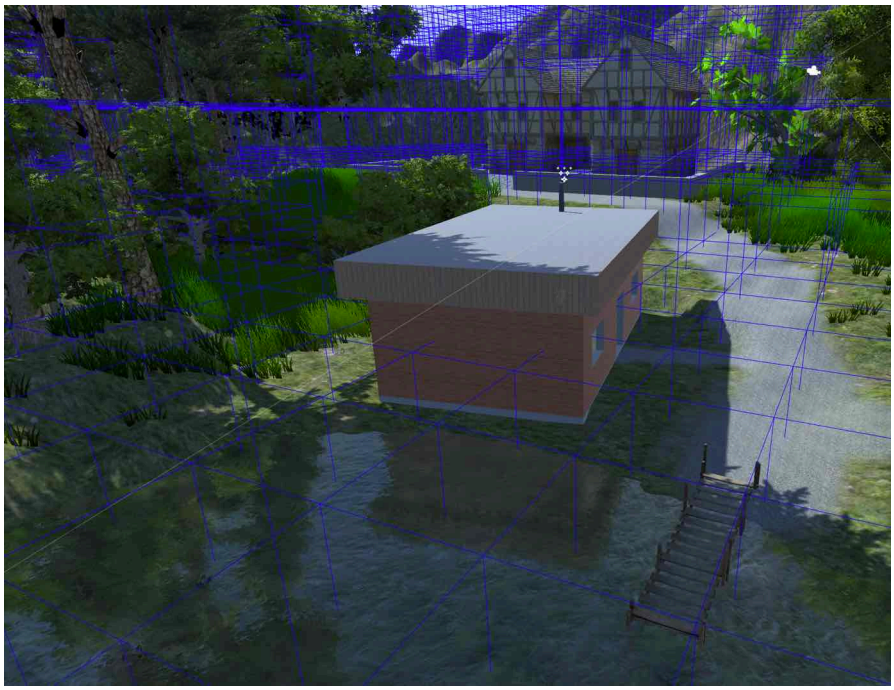
Occlusion culling toimii ennalta lasketun peittävyysdatan (occlusion data) perusteella. Occlusion cullingin asetukset löytyvät valitsemalla valikosta Window -> Occlusion Culling ja tämän jälkeen avautuvasta ikkunasta Bake-välilehti. Unityssä käyttäjä pystyy säätämään kolmea eri parametria, joiden perusteella occlusion data lasketaan:

1. Smallest Occluder: Tämä parametri määrittelee pienimmän objektin koon, joka tulkitaan mahdollisesti peittäväksi elementiksi ja joka näin ollen otetaan mukaan occlusion datan laskentaan. Unityn vakioarvo on 5.

2. Smallest Hole: Tässä kohdassa määritellään pienin mahdollinen reikä, jonka läpi katsottaessa näkymä renderöidään. Käytännössä tämän arvon tulee olla korkeintaan yhtä suuri kuin pienin pelimaailmasta löytyvä reikä.
3. Backface Threshold: Tämän avulla voidaan pyrkiä välttämään polygonien takasivujen mukaan ottamista occlusion cullingiin ja pyrkiä pienentämään occlusion datan kokoa. Maksimiarvo on 100 ja miniarvo on 5. Oletusarvo on 100, jolloin kaikki takasivut otetaan laskentaan mukaan.

Lopuksi valitaan ikkunan alareunasta ”Bake”, joka käynnistää occlusion datan laskennan.

Esimerkkipelimaailmassa kokeiltiin muutamia arvoja ja päädyttiin seuraavaan: Smallest Occluder = 0.75, Smallest Hole = 0.25 ja Backface Threshold = 5. Näillä arvoilla kaikki näytti edelleen silmämääräisesti toimivalta, ja myös occlusion datan laskenta-aika ja koko pysyi järkevissä rajoissa. Kuvassa 31 on esiteltynä osa pelimaailmaa occlusion culling -soluihin jaettuna.



KUVA 31. Occlusion culling -soluja.

Kuvassa 32 on esimerkki Occlusion Cullingin vaikutuksesta. Molemmissa kuvissa kamera on samassa paikassa ja kameran näkymä ruudulla on sama. Vasemmanpuoleiseen näkymään on kuitenkin renderöitynä 6.2 miljoonaa kolmiota ja 6.6 miljonaa verteksiä, kun taas oikeanpuoleisessa kuvassa on vain 2 miljoonaa kolmiota ja 2.3 miljoonaa verteksiä. Tästä seurauksena vasemmanpuoleisessa kuvassa ruudun päivitysnopeus on 11.2

FPS, kun taas oikeanpuoleisessa kuvassa päivitysnopeus on huomattavasti parempi 19.2 FPS.

KUVA 32. Esimerkki Occlusion cullingin vaikutuksesta

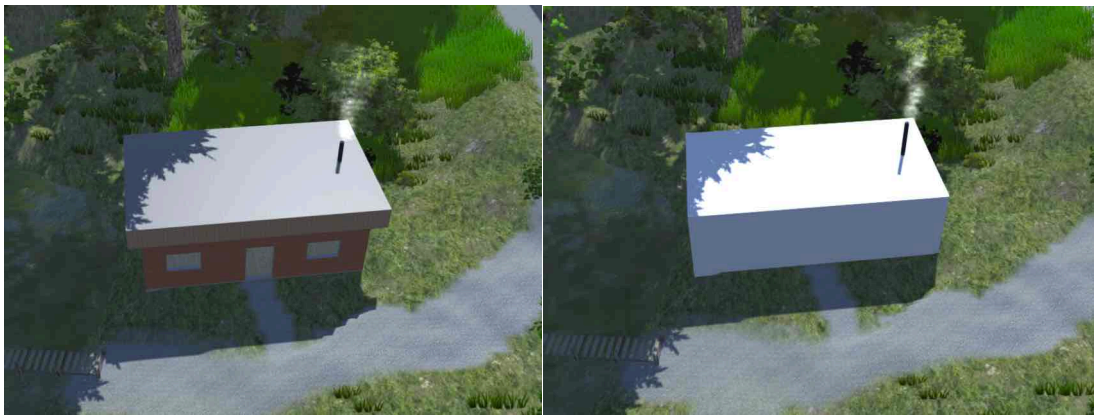
Occlusion dataa laskettaessa kokeiltiin eri vaihtoehtoja, ja mitä pienemmällä Smallest Occluder -arvolla laskennan tekee, sitä kauemmin se kestää ja sitä suurikokoisempi occlusion datasta tulee. Esimerkkinä mainittakoon, että esimerkkikokoonpanossa muutettaessa Smallest Occluder -parametri arvosta 5 arvoon 1, occlusion datan koko muuttui noin kahdestasadasta kilotavusta noin neljään megatavuun. Myös laskenta-aika moninkertaistui, ollen kuitenkin edelleen muutaman minuutin luokkaa. Kokeiltaessa Smallest Occluder -arvoa 0.1, järjestelmän muisti loppui kesken eikä occlusion dataa saatu laskettua ollenkaan.

4.11 LOD (Level of Detail)

LOD on lyhenne sanoista Level of Detail, ja sen avulla pyritään parantamaan 3d-grafiikan suorituskykyä. LOD:n ajatus on, että kauempana kamerasta olevista kohteista karsitaan yksityiskohtia suorituskyvyn parantamiseksi. Käytännössä tämä toteutetaan siten, että esimerkiksi 3d-objekteista tehdään useampi versio eri määrällä yksityiskohtia, ja kameraa kauemmas kohteesta siirrettäessä käytetään yksinkertaisempaa ja vähemmän yksityiskohtia sisältävää versiota.

Unityssä LOD on toteutettu käyttämällä LOD-ryhmiä, joiden avulla määritellään eri piirtoetäisyyksien yksityiskohtamäärät. LOD-ryhmät saa käyttöön lisäämällä objektille LOD Group -komponentti (Add Component -> Rendering -> LOD Group). Esimerkkipelimaa-

ilmassa LOD-ryhmiä ei lähdetty oikeasti toteuttamaan, mutta kokeilumielessä taloon lisättiin yksi LOD-ryhmä, joka muuttaa talon pelkäksi kuutioksi kameraa kauemmas vietäessä (kuva 33).



KUVA 33. Talon muuttuminen kuutioksi siirryttäessä LOD-ryhmästä toiseen.

4.12 Valaistus

Unityssä on kolme päävalaistustyyppiä ja -järjestelmää (Thorn 2017, Level lighting – preparation):

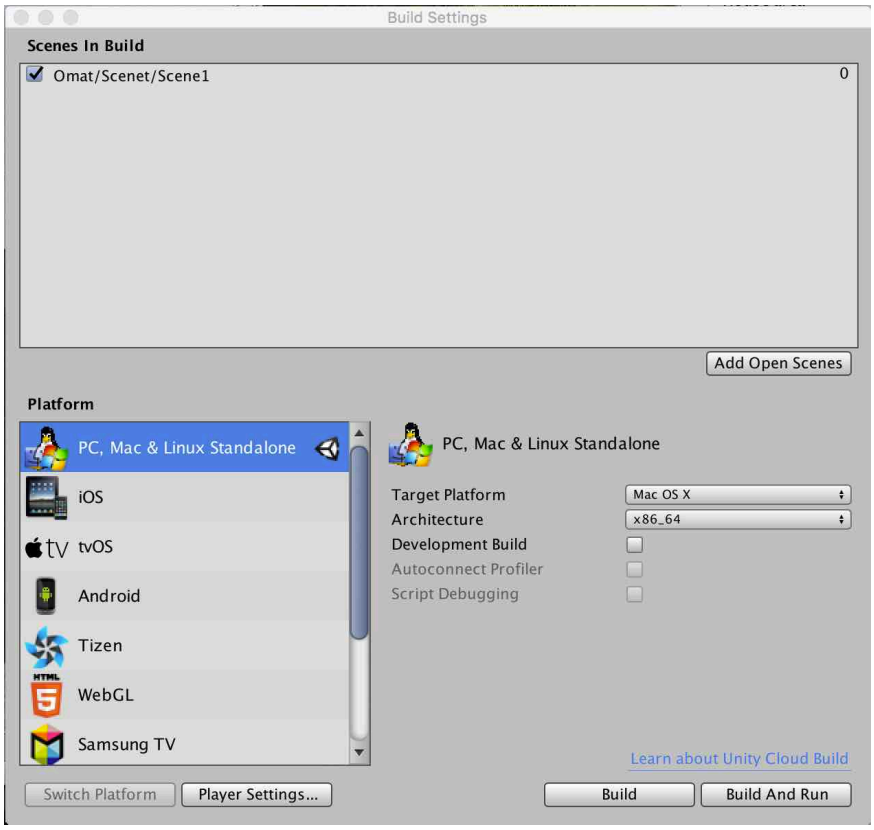
- Baked lighting
- Real-time lighting
- Precomputed global illumination

Baked Lighting perustuu esilaskettuun (Baked) valaistukseen, joka on tallennettu tekstuuriin, jota kutsutaan nimellä light map. Baked Lightingin paras puoli on hyvä suorituskyky. Huono puoli on se, että se toimii vain staattisten eli paikallaan pysyvien valojen ja objektien, kuten seinien ja muiden kiinteiden esineiden kanssa. Real-Time Lighting tarkoittaa nimensä mukaisesti reaaliaikaisesti laskettavaa valaistusta, ja näin ollen se toimii hyvin myös dynaamisten eli liikkuvien objektien ja valonlähteiden kanssa. Haittapuolena on, että reaaliaikainen valaistus vaatii runsaasti suorituskykyä. Precomputed Global Illumination on ikään kuin kahden edellä mainitun valaistuksen yhdistelmä, ja se perustuu osittain esilaskettuun valaistukseen. Precomputed Global Illumination mahdollistaa valonlähteiden reaaliaikaisen siirtelyn ja manipuloinnin, mutta edelleenkin se toimii vain staattisten objektien kanssa (Thorn 2017, Level lighting – preparation).

Esimerkkipelimaailmassa ei juurikaan paneuduttu valaistukseen, koska siihen ei ollut varsinaisesti tarvetta. Pelimaailma on ulkotila, jossa valaistuksen hoitaa auringonvaloa simuloiva Directional Light -peliobjekti. Valaistusasetuksista kuitenkin kokeiltiin eri asetuksia reaaliaikaisen ja esilasketun valaistuksen väliltä, mutta ne eivät eronneet keskenään sen enempää suorituskyvyn kuin ulkonäönkään perusteella, joten todelliset käytännön erot jäivät tältä osin epäselviksi. Nyrkkisääntönä voitaneen kuitenkin pitää sitä, että mitä vähemmän kohdealustassa on suorituskykyä, sitä enemmän tulisi suosia esilaskettua valaistusta. (Unity technologies 2017, Lighting overview)

4.13 Itsenäinen sovellus

Lopuksi pelimaailmasta tehtiin kokeilumielessä itsenäinen sovellus. Tämä tapahtui valitsemalla Unityn valikosta ”Build Settings”, jonka jälkeen aukeavasta ikkunasta voidaan valita muun muassa kohdealusta ja scenet, jotka halutaan ottaa mukaan (kuva 34). Lopuksi valitaan joko ”Build” tai ”Build And Run” riippuen siitä, halutaanko sovellus käynnistää saman tien.



KUVA 34. Unityn Build Settings -ikkuna

Itsenäisen sovelluksen kanssa törmättiin yllättäviin ongelmiin. Jostain syystä Skybox ei suostunut toimimaan, vaan taivas oli tasaisen harmaa. Vielä vakavampi ongelma oli se, että vedenpintana käytetty Unityn oma WaterProDaytime-asset kaatoi säännöllisin väliajoin koko tietokoneen aiheuttaen kernel panicin. Virheraporttia tutkittaessa ongelma näytti liittyvän näytönohjaimeen. Kyseinen asset tuntui myös aiheuttavan yleisiä grafiikkavirheitä koko pelimaailman alueelle. Loppujen lopuksi kaatuilu saatiin poistettua muuttamalla vedenpinnan asetuksista tyyppi taittavasta (Refractive) heijastavaksi (Reflective), mutta grafiikkavirheet tuntuivat silti säilyvän.

Skyboxin kanssa ongelmaksi osoittautui Global Fog -kameraefekti, jonka tuottama sumu peitti taivaanrannan näkyvistä. Global Fog sisältää asetuksen ”Exclude Far Pixels”, jonka avulla Unityn omassa pelinäkymässä kameran kuva renderöityy oikein, mutta jostain syystä itsenäisessä sovelluksessa kyseinen asetus ei kuitenkaan toimi. Tämä todennäköisesti liittyy jollain tapaa renderöintiliukuhihnaan tai -järjestykseen, mutta koska se ei kuulunut tämän opinnäytetyön aihepiiriin, asian annettiin tässä vaiheessa olla. Vaihtoehtoinen tapa on käyttää Global Fogin sijaan Unityn valaistusasetuksista löytyvää Fog-asetusta, ja tämä toimi myös tässä tapauksessa riittävän hyvin. Tämän jälkeen pelimaailma olikin valmis (kuva 35).

KUVA 35. Valmista pelimaailmaa.

4.14 Muita huomioita pelimaailman toteutuksesta

Laitteistona opinnäytetyössä toimi MacBook Pro (mid 2015), jossa oli käyttöjärjestelmänä MacOS 10.12.6, 2.2:n gigahertsin Intel Core i7 -suoritin, 16 gigatavua muistia ja Intel Iris Pro -näytönohjain. Melko varhaisessa vaiheessa tuli huomattua, että 3d-pelikehityksessä suorituskykyä tarvitaan paljon, ja kyseinen kokoonpano oli varsinkin näytönohjaimen osalta turhan tehoton. Myös muistia olisi saanut olla enemmän, sillä Unity jouduttiin silloin tällöin sulkemaan pakolla muistin loppumisen takia. Maya LT osoittautui hieman epävakaaksi ohjelmistoksi, ja se kaatui useamman kerran opinnäytetyön 3d-malleja tehtäessä.

Ongelmia aiheutti myös Terrain-objektin kopionti. Kyseisestä objektista halutiin yhdessä vaiheessa tehdä kopio toiseen sceneen, jotta siihen voitaisiin vapaasti kokeilla erinäköisiä muutoksia. Muutokset kuitenkin kopioituivat aina myös alkuperäiseen Terrain-objektiin. Loppujen lopuksi ainoaksi tavaksi tehdä Terrain-objektista täysin itsenäinen kopio oli mennä tekemään kopiointioperaatio käyttöjärjestelmän tiedostohallinnan kautta. Mikään Unityn sisällä tehty kopiointi- tai kahdennusoperaatio ei toiminut, vaan kopioitu Terrain sisälsi aina viittaukset alkuperäiseen Terrain-elementtiin.

Unityn occlusion cullingin laskenta tuottaa pahimmillaan miljoonia yksittäisiä pieniä tiedostoja. Tämä aiheutti ongelman tiedostojen pilvipalveluna käytetyn iCloud Driven kanssa, jonka synkronointi ei pystynyt käsittelemään näin suurta tiedostomäärää ja meni totaalisesti jumiin. Asiasta selvittiin pakottamalla iCloud Drive -synkronointi pois päältä ja menemällä web-sovelluksen kautta poistamaan ongelmia aiheuttanut kansio pilvestä. Poistamisen jälkeen jouduttiin kuitenkin odottamaan useampi vuorokausi, ennen kuin kansio sisältöineen oli kokonaan hävinnyt.

Hieman samantyylinen vahinko sattui, kun epähuomiossa tietokoneen kiintolevyn ollessa melkein täynnä annettiin käyttöjärjestelmälle lupa optimoida tilankäyttöä. Tästä seurasi se, että kaikki tietokoneella olleet kuvat poistettiin paikalliselta kiintolevyltä ja siirrettiin pelkästään iCloudiin. Näin ollen myös kaikki pelimaailmassa käytetyt tekstuurit häipyivät pilveen, eikä pelimaailma enää korjaantunut pelkästään siirtämällä tekstuurit takaisin paikalliselle kiintolevylle, vaan jokainen Unity-projektissa ollut tekstuureita sisältävä objekti piti poistaa ja tuoda uudelleen.

5 POHDINTA

Opinnäytetyön tarkoituksena oli tutustua 3d-mallintamisen ja -pelikehityksen perusteisiin aloittelijan näkökulmasta. Tätä silmällä pitäen asetettiin tavoitteeksi luoda yksinkertainen 3d-pelimaailma, jossa olisi mukana myös jotain itse mallinnettuja objekteja. Jotta opinnäytetyöhön saataisiin mukaan enemmän tutkivaa näkökulmaa ja perustaa pelimoottori- ja mallinnusohjelmistovalinnalle, päädyttiin toteuttamaan myös suomalaisille peliyrityksille suunnattu aiheeseen liittyvä kyselytutkimus. Lopputulosta tarkasteltaessa voidaan todeta, että asetetut tavoitteet täyttyivät. Silti opinnäytetyöstä jäi tekijälleen hieman ristiriitainen vaikutelma.

Opinnäytetyön parasta antia oli ehdottomasti peliyrityksille toteutettu kyselytutkimus. Siihen saatiin runsaasti vastauksia, ja vastaavia tutkimuksia ei muutenkaan ole tehty paljoa. Sen sijaan itse 3d-esimerkkipelimaailman toteutukseen liittyvässä aihepiirin rajauksessa ei onnistuttu aivan täysin, ja varsinkin Unityä koskeva osio jäi turhan pintapuoliseksi ja sekavaksi. Tämä johtui pitkälti siitä, että opinnäytetyö tehtiin omasta aihevalinnasta, josta tekijällä ei ollut juuri mitään aikaisempaa kokemusta. Opinnäytetyön edetessä joutui koko ajan miettimään, mitä asioita otetaan mukaan ja mitä ei, ja aihepiiri osoittautui aloittelijalle liian laajaksi kokonaisuudeksi. Tästä syystä opinnäytetyön kanssa tuli myös kiire, ja monet tärkeät asiat jäivät käsittelemättä, ja toisaalta mukana on myös ehkä vähemmän oleellista optimointiin liittyvää asiaa. Jälkiviisas on toki aina helppoa olla, mutta 3d-perusteiden opiskeluun olisi ehkä kannattanut valita jokin yksinkertaisempi ja vähemmän yksityiskohtia sisältävä kokonaisuus. Tällainen olisi voinut olla esimerkiksi jokin yksinkertainen sisätila, joka olisi sisältänyt vain muutaman objektin. Tällöin kokonaisuus olisi todennäköisesti pysynyt selkeämpänä ja sen avulla olisi ollut helpompi ymmärtää esimerkiksi valaistuksen vaikutusta. Myöskin mallinnettavat objektit olisi voitu valita yksinkertaisemmiksi, koska nyt varsinkin talo oli teksturoinnin suhteen yllättävän sekava tapaus.

Toisaalta opinnäytetyötä tehtäessä oli ilo havaita, kuinka helposti ja pienellä vaivalla 3d-pelikehityksessä pääsee alkuun. Tarjolla on runsaasti eri ohjelmisto- ja pelimoottorivaihtoehtoja, joista monet ovat ilmaisia tai ainakin sangen edullisia. Unity osoittautui erinomaiseksi ja helposti lähestyttäväksi työkaluksi, ja tästä syystä sen suosiota on helppo ymmärtää. Lisäksi lukuisat internetin aiheeseen liittyvät keskustelupalstat ja verkkosivut

tarjoavat runsaasti tietoa, ja yleensä hetken etsiskelyn jälkeen ongelmaan kuin ongelmaan löytyi ratkaisu.

Jatkoa ajatellen ja oman ammattitaidon kehittämistä silmällä pitäen voisi olla järkevää, että seuraavaksi projektiksi ottaisi jotain yksinkertaisempaa mutta samalla sellaista, jonka pystyisi viimeistelemään paremmin. Unityssä skriptaukseen ja ohjelmointipuoleen pitää tutustua tarkemmin, kuten myös pelattavuuden ja äänimaailman tekemiseen. Lisäksi 3d-mallinnusohjelmistoista Blender täytyy ehdottomasti ottaa opiskelun kohteeksi, koska se tuntuu olevan erittäin suosittu. Jos taas miettii opinnäytetyön aihepiiriä enemmän tutkimuksen kannalta, eräs mielenkiintoinen jatkotutkimuskohde voisi olla erilaiset pelien serveri- ja back end -ratkaisut, kuten eräs kyselyyn vastanneista peliyrityksistä kommentoikin:
”Nykyään alkaa olla yhtä relevanttia kysyä firmojen käyttämistä backend-ratkaisuista. Modernit pelit pyörivät yhtä paljon clientissä ja pilvessä ja kumpaankin on oma teknologiapuunsa joka ei välttämättä liity mitenkään graffaengineen.”

LÄHTEET

Tampereen yliopisto, Yhteiskuntatieteellinen tietoarkisto. 2013. Kvantitatiivisten menetelmien tietovaranto. Päivitetty 14.5.2013. Luettu 10.6.2017
http://www.fsd.uta.fi/menetelmaopetus/ristiintaulukointi/ristiintaulukointi.html

Slick, J. 2016. What Is 3D Modeling?. Päivitetty 20.10.2016. Luettu 5.7.2017
https://www.lifewire.com/what-is-3d-modeling-2164

Zeman, N. 2015. Essential Skills for 3D Modeling, Rendering, and Animation. CRC Press.
http://proquest.safaribooksonline.com.elib.tamk.fi/book/animation-and-3d/9781482224122

Thorn, A. 2017. Mastering Unity 5.x. Packt Publishing.
http://proquest.safaribooksonline.com.elib.tamk.fi/book/programming/game-programming/9781785880742/

Cardoso, C. 2015. Lumion 3D Best Practices. Packt Publishing.
http://proquest.safaribooksonline.com.elib.tamk.fi/book/3d-printing/9781783550852

Ahearn, L. 2016. 3D Game Textures, 4th Edition. CRC Press.
http://proquest.safaribooksonline.com.elib.tamk.fi/book/programming/game-programming/9781351859769

Unity technologies. 2017. Unity User Manual.
https://docs.unity3d.com/Manual/index.html

Gerasimov, V. 2015. Building Levels in Unity. Packt Publishing.
http://proquest.safaribooksonline.com.elib.tamk.fi/book/programming/game-programming/9781785282843

Ganovelli, F. & Corsini, M. & Pattanaik, S. & Di Benedetto, M. 2015. Introduction to Computer Graphics. Chapman and Hall/CRC
http://proquest.safaribooksonline.com.elib.tamk.fi/book/programming/graphics/9781439852798

Dickinson, C. 2015. Unity 5 Game Optimization. Packt Publishing.
http://proquest.safaribooksonline.com.elib.tamk.fi/book/programming/game-programming/9781785884580

Thorn, A. 2015. Unity Animation Essentials. Packt Publishing.
http://proquest.safaribooksonline.com.elib.tamk.fi/book/programming/game-programming/9781782174813

LIITTEET

Liite 1. Kyselyssä käytetty lomake

Kysely pelimoottoreista ja 3d-mallinnusohjelmista opinnäytetyötä varten

Kyselyn tarkoituksena on kartoittaa opinnäytetyötä varten suomalaisten pelintekijöiden käyttämiä pelimoottoreita ja 3d-mallinnusohjelmistoja. Kysely on lyhyt (7 kysymystä) ja sen tekemiseen kuluu aikaa muutama minuutti. Vastausaikaa on maaliskuun loppuun asti.

Kyselyn tekijä:
Antti Häyrinen
Tietojenkäsittely, Tampereen ammattikorkeakoulu

*Pakollinen

Kuinka kauan yrityksenne on toiminut pelialalla? *

- ○ 0 - 2 vuotta
- ○ 2 - 5 vuotta
- ○ Yli 5 vuotta

Kuinka monta henkilöä yrityksessänne työskentelee? *

- ○ 1 - 5 henkilöä
- ○ 6 - 20 henkilöä
- ○ Yli 20 henkilöä

Mitä pelimoottoria/pelimoottoreita yrityksenne pelit käyttävät? *

- ☐ Unity
- ☐ Unreal Engine
- ☐ Blender Game Engine
- ☐ Construct 2
- ☐ GameMaker
- ☐ CryEngine
- ☐ Amazon Lumberyard
- ☐ Oma pelimoottori
- ☐ Muu: _____

Mille alustalle/alustoille yrityksenne kehittää pelejä? *

- ☐ Mobiili (Android, iOS, Windows Phone, ...)
- ☐ Työpöytä (Windows, OS X / MacOS, Linux, ...)
- ☐ Konsolit (PS, XBox, Nintendo, ...)
- ☐ Web
- ☐ Muu: _____

Mitä käyttöjärjestelmää/käyttöjärjestelmiä yrityksessänne käytetään pelien kehittämiseen? *

- ☐ Windows
- ☐ OS X / MacOS
- ☐ Linux
- ☐ Muu: _____

Kehittääkö yrityksenne 2d-pelejä, 3d-pelejä vai molempia? *

- ○ 2d
- ○ 3d
- ○ Molemmat

Jos yrityksenne kehittää 3d-pelejä, mitä 3d-mallinnusohjelmistoa/mallinnusohjelmistoja käytätte?

- ☐ Blender
- ☐ Maya / Maya LT
- ☐ 3ds MAX
- ☐ Cinema 4D
- ☐ Modo
- ☐ Mudbox
- ☐ ZBrush
- ☐ Muu: _____

Muuta kommentoitavaa:

Oma vastauksesi

Liite 2. Kameran liikuttamiseen käytetty SpectatorController-skripti

```
using UnityEngine;
using System.Collections;

public class SpectatorController : MonoBehaviour {

    public float speed;

    public float horizontalSpeed;
    public float verticalSpeed;

    private Rigidbody rb;
    private Vector3 objectRotation;
    private Transform objectTransform;

    void Start() {
        rb = GetComponent<Rigidbody> ();
        objectTransform = GetComponent<Transform> ();

    }

    void FixedUpdate() {

        float translationZ = Input.GetAxis("Vertical");
        float translationX = Input.GetAxis("Horizontal");

        float h = horizontalSpeed * Input.GetAxis("Mouse X");
        float v = verticalSpeed * Input.GetAxis("Mouse Y");

        Vector3 globalTransform = objectTransform.TransformDirection (translationX, 0.0f, translationZ);
        rb.AddForce (globalTransform * speed);

        transform.Rotate(v, h, 0);
        transform.eulerAngles = new Vector3(transform.eulerAngles.x, transform.eulerAngles.y, 0.0F);
    }
}
```

Liite 3. WaterArea-skripti

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

public class WaterArea : MonoBehaviour {

    public GameObject Kamera;
    private ColorCorrectionCurves CCC;

    void Start () {
        CCC = Kamera.GetComponent<ColorCorrectionCurves>();
    }

    void OnTriggerEnter(Collider other) {
        if (other.gameObject.CompareTag ("WaterAreaTriggerZone")) {
            CCC.enabled = true;
        }
    }

    void OnTriggerExit(Collider other) {
        if (other.gameObject.CompareTag ("WaterAreaTriggerZone")) {
            CCC.enabled = false;
        }
    }
}
```