Teemu Hänninen

RAKENNUSMALLIN PELILLISTÄMINEN

Opinnäytetyö
Tietojenkäsittely

2017

Kaakkois-Suomen
ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Teemu Hänninen	Tradenomi (AMK)	Joulukuu 2017
Opinnäytetyön nimi Rakennusmallin pelillistäminen		63 sivua 0 liitesivua
Toimeksiantaja Kaakkois-Suomen ammattikorkeakoulu		
Ohjaaja Jukka Selin		

Tiivistelmä

Opinnäytetyössä tutkitaan, kuinka pelimoottoreita ja rakennusten 3D-malleja voidaan hyödyntää, jotta saadaan parempi käsitys digitaalisesta ympäristöstä. Aiheesta esitellään projekti, jota pohjustetaan siihen liittyvällä teorialla.

Opinnäytetyön viitekehys koostuu kolmesta osasta. Ensimmäisenä kerrotaan, mitä tarkoittaa pelillistäminen ja kuinka peliteknologiaa voidaan hyödyntää AEC-alalla 3D-mallien kanssa. Seuraavaksi käydään läpi pelimoottoreiden tärkeimpiä ominaisuuksia, joiden avulla rakennusmalli voidaan pelillistää. Kolmannessa osassa kerrotaan 3D-mallintamisen menetelmistä ja niistä seikoista, joita täytyy huomioida mallin tuotantovaiheessa, jos malli halutaan tuoda pelimoottoriin.

Opinnäytetyön viimeisessä osassa esitellään, kuinka rakennuksesta tehty digitaalinen 3D-malli voidaan pelillistää Unity 3D -pelimoottorin avulla. Pelillistämisellä on useita tavoitteita. Pelaajan pitää pystyä vapaasti liikkumaan ympäristössä ja jollakin tavalla pystyä olemaan vuorovaikutuksessa ympäristön kanssa. Vuorovaikutus on toteutettu niin, että pelaaja voi näppäinkomennoilla avata talon ovia. Lisäksi tavoitteena on selvittää, millä keinoilla rakennuksen 3D-malliin voidaan lisätä realismia.

Realismiin vakuttavista tekijöistä tärkeimmäksi osoittautuu realistinen valaistus. Lisäksi projekti osoittaa, että ilmaisella SketchUpilla ja Unity 3D -pelimoottorilla pystytään mallintamaan ja pelillistämään rakennuksia luonnosteluun ja esittelyyn riittävällä tarkkuudella, mutta rakennussuunnitteluun ne eivät ole riittävän tarkkoja.

Asiasanat

3D-malli, 3D-mallintaminen, AEC, pelillistäminen, pelimoottori, unity, sketchup

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Teemu Hänninen	Bachelor of Business Administration	December 2017

Thesis title

Gamification of a building model

63 pages
0 pages of appendices

Commissioned by

South-Eastern Finland University of Applied Sciences

Supervisor

Jukka Selin

Abstract

The thesis investigated how game engines and 3D models of buildings could be utilized to gain a better understanding of the digital environment. This thesis presented a project that based on the related theory.

The framework for the thesis consisted of three parts. In the first part gamification as a term was explained and furthermore, the potential benefits of game technology combined with 3D models were explored. In the second part, the main features of game engines were explained. The features explained were limited to the most important features needed in order to create a realtime digital environment from a 3D building model. In the third part, different methods of 3D modeling were described and the things that need to be taken into account when importing a 3D model into a game engine were also examined.

The final part of the thesis demonstrated how the 3D model of a building could be turned into a realtime environment with Unity 3D. This project had several goals. The player must be able to move freely around the environment and in some way be able to interact with the environment. The interaction was implemented by giving the player the option to open doors with predefined keyboard commands. Additionally, the goal of the project was to find out ways to increase realism in the 3D model.

As a conclusion, it became evident that realistic lighting was the most important factor to take into account when creating realistic digital environments. The project also demonstrated that the free version of SketchUp and Unity 3D engine were sufficient tools for the presentation and design of buildings. However, they did not offer sufficient accuracy for architectural design that could be used as a blueprint for the entire construction process of a building.

Keywords

3D model, 3D modeling, AEC, gamification, game engine, unity, sketchup

SISÄLLYS

1 JOHDANTO 5

2 PELILLISTÄMINEN 6

2.1 Pelillistäminen terminä 6

2.2 Peliteknologian hyötykäyttö 7

2.3 Pelimoottorit 11

2.4 Unity 3D 13

3 3D-MALLINTAMINEN 19

3.1 Mikä on 3D-malli 19

3.2 3D-mallintamisen menetelmiä 22

3.3 SketchUp 24

3.4 Tiedostomuodot 27

4 DEMO AIHEESTA 30

5 PÄÄTÄNTÖ 58

LÄHTEET 60

1 JOHDANTO

Kun muinaiset egyptiläiset arkkitehdit suunnittelivat Gizan pyramideja, heillä ei ollut käytössään tietokoneita, tai näin voidaan ainakin olettaa melko suurella todennäköisyydellä. Pyramidien suunnittelu ja visualisointi siis tehtiin analogisilla menetelmillä. Tuhansia vuosia myöhemmin, kun Lontoon korkeimpiin rakennuksiin kuuluva Leadenhall Building valmistui, oli arkkitehdeillä käytössään digitaaliset työvälineet, ja tietokonesimulaatioissa rakennus rakennettiin yhteensä 37 kertaa (Construction in the City s.a.). Voidaan vain arvailla, kuinka paljon aikaa ja resursseja olisi kulunut, jos Gizan pyramidit olisi pitänyt rakentaa 37 kertaa ennen kuin lopputulokseen oltiin tyytyväisiä.

Noin puoli vuosisataa on kulunut siitä, kun Ivan Sutherland kehitti SKETCHPAD-ohjelman, jonka avulla käyttäjä pystyi ensimmäisen kerran tuottamaan grafiikkaa tietokoneen näytölle. Puolessa vuosisadassa tietokonegrafiikka on edennyt huimasti, ja tietokoneiden laskentatehon myötä tietokoneavusteisesta suunnittelusta on saatu korvaamaton apuväline rakennusten suunnittelussa. Viimeisten 20 vuoden aikana varsinkin videopelit ovat antaneet peliohjelmoijille syyn kehittää uusia, tehokkaita menetelmiä esittää 3D-grafiikkaa tietokoneilla. Pelimäisiä elementtejä on alettu hyödyntää erilaisissa hyötypeleiksi kutsutuissa sovelluksissa, ja 3D-pelimoottoreiden kehittymisen myötä peliteknologian sovellusalueet laajenevat eri teollisuuden aloille. Tavanomaisten suunnitteluohjelmien kuvakulmat on tarkoitettu suunnitteluun ja mallintamiseen, joten ne eivät tarjoa sellaista intuitiivista tapaa kokea digitaalisia ympäristöjä mihin reaaliaikaiset 3D-pelimoottorit pystyvät.

Tässä opinnäytetyössä kerrotaan aluksi, mitä pelillistäminen terminä tarkoittaa ja tutkitaan lähemmin, mitä tarkoittaa nimenomaan rakennusmallien pelillistäminen 3D-pelimoottorien avulla. Pelimoottoreista kerrotaan keskeisimmät ominaisuudet, joiden avulla rakennusmalli voidaan pelillistää. Näitä ominaisuuksia tarkastellaan Unity-pelimoottorin näkökulmasta. Lisäksi selvitetään, mitä tarkoittaa 3D-malli ja millaisilla menetelmillä niitä voidaan tuottaa. Lähteinä on käytetty alaan liittyvää kirjallisuutta ja tieteellisiä artikkeleita. Pelimoottoreihin ja 3D-mallintamiseen liittyvää tietoa on hankittu myös näitä aiheita käsittelevistä oppaista. Opinnäytetyön viimeisenä osana demonstroidaan käytännön

projektilla, kuinka mallinnusohjelmassa tehty 3D-malli muutetaan reaaliaikaiseksi peliympäristöksi Unity-pelimoottorin avulla. Projektin tarkoitus on osoittaa, että rakennusmallin tarkastelu reaaliaikaisesti on käyttäjälle mieluisa tapa kokea digitaalisia ympäristöjä, ja sen avulla tilan hahmottaminen on helpompaa ja tuntuu luontevammalta kuin suunnitteluohjelmissa. Lisäksi tarkoitus on näyttää, että nykyaikaisilla ilmaisilla ohjelmistoilla voidaan suhteellisen helposti tehdä toimivia rakennusesittelyjä, eikä siihen vaadita suurta määrää ohjelmointia. Projektissa myös tutkitaan, millä keinoilla peliympäristöön saadaan realismia.

2 PELILLISTÄMINEN

Pelillistäminen on termi, joka on yleistynyt viime vuosien aikana. Tästä huolimatta pelillistäminen ei ole uusi ilmiö, koska ovathan pelit ja leikit aina olleet jollain tavalla osa kulttuuriamme. Elämän pelimäisiä ominaisuuksia on tutkittu jo 1960-luvulta lähtien, joten ei ole yllättävää, että peleistä tuttuja ominaisuuksia on alettu tuoda osaksi liiketoimintaa ja koulutusta. (Zichermann & Cunningham 2011.)

2.1 Pelillistäminen terminä

Zichermannin ja Cunninghamin mukaan (2011) pelillistäminen on pelimäisen ajattelun ja pelimekaniikan hyödyntämistä pelien ulkopuolisessa kontekstissa, ja siihen liittyy olennaisesti tavoitteet ja käyttäjän palkitseminen. Palkitseminen auttaa käyttäjää pääsemään tavoitteeseensa ja tekee prosessista viihdyttävämpää.

Pelillistäminen kuitenkin ymmärretään käsitteenä helposti väärin, koska se yhdistetään tavanomaisiin viihdepeleihin, vaikka pelillistetyillä hyötysovelluksilla ei ole paljoa yhteistä pelien kanssa. Pelillistämisen tarkoitus ei ole tehdä yritystoiminnasta ja hyötysovelluksista pelkästään viihdyttäviä ja pelin kaltaisia, vaan tehostaa liiketoimintaa peleistä tuttujen ominaisuuksien avulla. Pelillistämisellä tarkoitetaankin käytännössä pelimäisten ominaisuuksien hyödyntämistä ongelmien ratkaisemisessa ja käyttäjien sitouttamisessa. (Haonperä 2013.)

Pelillistämisen keinot eivät rajoitu pelkästään liiketoiminnassa käytettäviin sovelluksiin ja toimintatapoihin, vaan niitä voidaan hyödyntää myös koulutuksessa. Esimerkiksi suomalainen VR Track on kehittänyt työntekijöidensä työturvallisuustaitoja niin sanotun pakopelin avulla. Pakopelien ideana on, että ryhmän täytyy ratkaista ongelmia yhdessä ja näin selviytyä pois jostakin suljetusta tilasta. Pelin avulla kehitetään työntekijöiden ongelmanratkaisukykyä ja vaikutetaan ihmisten asenteisiin työturvallisuutta kohtaan. (Savela 2017.)

2.2 Peliteknologian hyötykäyttö

Rakennus- ja kiinteistöalalla peliteknologian keinot voivat tarjota mielenkiintoisia vaihtoehtoja esittää todellisia ympäristöjä digitaalisesti sekä asiakkaille että suunnittelijoille. Vuonna 1994 Warren Berger kuvaili Real Estate -lehdessä (1994, 15) tulevaisuudennäkymän kiinteistönvälittäjän ja asiakkaan välisestä vuorovaikutuksesta. Bergerin visiossa asiakas astelee kiinteistönvälittäjän puheille ja ilmoittaa olevansa kiinnostunut jotakin asuntokohteesta. Sen sijaan, että hypätään autoon ja ajetaan konkreettisesti katsomaan, miltä asuntokohde näyttää, kiinteistönvälittäjä pyytääkin asiakasta istumaan tietokoneen ääreen. Kiinteistönvälittäjä avaa tietokoneen näytölle kartan ja pyytää asiakasta osoittamaan kartalta haluamansa kohteen. Asiakas tekee työtä käskettyä, ja kiinteistönvälittäjä klikkailee tietokoneen näytölle näkyviin virtuaalisen version talokohteesta. Tämän jälkeen hiiri annetaan asiakkaan käsiin, joka saa vapaasti käydä tutkimassa tulevaa kotiaan virtuaalisesti tietokoneen näytöllä.

Hypätään ajassa 23 vuotta eteenpäin nykyhetkeen, vuoteen 2017. Vuonna 2017 edellä kuvatun kaltainen skenaario ei ole enää pelkkä tulevaisuudenvisio. Jatkuvasti kehittyvä teknologia on mahdollistanut yhä kehittyynempää tietotekniikkaa entistä pienemmissä laitteissa, mistä hyvä esimerkki ovat älypuhelimet (Fisher ym.). Tekniikan kehittyessä myös ohjelmistot kehittyvät ja tuovat uusia ulottuvuuksia siihen, miten tietoa esitetään. Rakennusalalla tämä tarkoittaa kirjaimellisesti uutta ulottuvuutta: rakennuksia ei enää suunnitella pelkästään kaksiulotteisina piirustuksina arkkitehdin pöydällä, vaan tietokoneen avulla luodut kolmiulotteiset mallit eli 3D-mallit ovat jo pitkään olleet laajalti käytössä rakennusalalla. Kiinteistöalalla tiedetään, että valmis kiinteistö täytyy myös esitellä houkuttelevasti potentiaalisille ostajille. Asiakkaan mielenkiinto vangitaan näyttävillä 3D-malleilla, joissa kohteen parhaat puolet tuodaan

esille (IndiaCADworks 2013). Varsinkin uudiskohteiden markkinointi saa uutta potkua 3D-malleista, koska ne pystyvät kuvia paremmin havainnollistamaan kolmiulotteisia tiloja (Unrealer 2017).

Kuluneen vuosikymmenen aikana AEC-alaa on muuttanut rakennuksen tietomallintaminen, eli BIM (Building Information Modeling) (Zhao 2017). Kun tavanomainen 3D-malli yksinkertaisimmillaan tarkoittaa pelkästään talon geometrian esittävää mallia, niin BIM on paljon muutakin, ja 3D-malli on vain yksi sen osa. Olioihin perustuvassa parametrisessa tietomallintamisessa rakennuksesta tehdään digitaalisesti tarkka virtuaalinen malli, joka sisältää kaiken tiedon mitä tarvitaan rakennuksen elinkaaren aikana suunnittelun aloituksesta rakennuksen purkamiseen. Kun yhtä parametria muutetaan, päivittyvät muut parametrit automaattisesti. Rakennuksen tietomallintamisessa 3D-mallia ei luoda kaksiulotteisten kuvien pohjalta, vaan se luodaan suoraan käyttäen rakennuksen mittatietoja (Eastman 2008).

Tietomallintamisen keskeisin ero tavanomaiseen 3D-mallintamiseen on sen sisältämän tiedon semanttisuus, eli tieto ei ole pelkästään visuaalista. Tietomalli ei ole staattinen esitys jostakin kohteesta, vaan simulaatio, joka ”elää” tiedon muutoksien mukana (Lappalainen 2016). Tietomallintamisen hyödyt ovat ilmeiset. Kun talon kaikki elementit on tarkasti tietomallinnettu, pystytään esimerkiksi seinien sisällä kulkevien sähköjohtojen ja putkien törmäystarkastelua automatisoimaan, mikä vähentää suunnitteluvirheitä ja nopeuttaa rakennusprosessia (Eastman 2008). Bryden ym. (2012) mukaan tietomallintamisen hyödyntäminen rakennusprojekteissa vaikuttaa positiivisesti kustannuksiin, suunnitteluun, ajankäyttöön ja osallistujien kommunikointiin.

Vaikka 3D-malleja voikin tarkastella eri kulmista aivan kuin oikean elämän kohteita, niin on kuitenkin muistettava, että lopulta nekin ovat yhtä staattisia kuin perinteiset kaksiulotteiset piirustukset. Ne eivät tarjoa käyttäjälle todellista elämää muistuttavaa vuorovaikutusta ympäristön kanssa. Peliteknologiaa hyödyntämällä mahdollistetaan käyttäjän ja ympäristön välinen reaaliaikainen vuorovaikutus. Suunnittelijan tai asiakkaan ei tarvitse tyytyä pelkästään katselijan rooliin, kun 3D-malli pelillistetään nykyaikaisten pelimoottorien avulla. (Pluralsight 2014.)

Kun Berger vuonna 1994 kirjoitti näkemyksensä virtuaalisesta asuntoesittelystä, olivat 3D-pelimoottorit vielä lapsenkengissään. Nykyaikaiset pelimoottorit pystyvät kuitenkin luomaan dynaamisia ja monimutkaisia kolmiulotteisia ympäristöjä, ja visuaalisen realistisuuden lisäksi myös fysiikan lakeja pystytään mallintamaan reaalielämää muistuttavalla tavalla. Pelimoottoreista esimerkiksi Unity 3D ja CryEngine ovat tunnettuja realistisesta fysiikkamallinnuksestaan. Pelimoottorien kehitys onkin mahdollistanut niiden käyttämisen myös muihin tarkoituksiin kuin peleihin, kuten arkkitehtoniseen suunnitteluun ja armeijan simulaatioihin (Kosmadoudi 2012). Armeijat ovatkin hyödyntäneet pelillistämistä osana koulutusta jo satoja vuosia, ja varsinkin Yhdysvaltain armeija on ollut edelläkävijä videopelien hyödyntämisessä (Zichermann ym. 2011). Pelimoottorien ja rakennusmallien yhteinen potentiaali tiedetään, mistä kertoo se, että esimerkiksi vuonna 2014 Yhdysvaltalainen tietokoneavusteisen suunnittelun ohjelmistoihin erikoistunut yritys Autodesk osti ruotsalaisen Bitsquid-ohjelmistoyrityksen, jonka käsialaa on samaa nimeä kantava pelimoottori. Myöhemmin Autodesk lanseerasi pelimoottorin uudestaan omalla tuotenimellään Stingray (Pluralsight 2014).

Kuva 1. Nykyaikaisilla pelimoottoreilla toteutetaan näyttäviä pelejä, kuten Witcher 3 (2015)

Pelillistämällä käyttäjä siirretään ympäristön keskelle ja simuloidaan reaalielämän kokemusta. Mallinnusohjelmien työkaluilla ympäristön tarkkailu ei vastaa ihmisen intuitiivista tapaa liikkua ja kokea tila ja aika, koska tilaa tarkkaillaan ulkopuolisesta näkökulmasta. Ensimmäisen persoonan näkökulmasta subjektiivinen kokemus korostuu ja käyttäjä tuntee vahvemmin olevansa läsnä ympäristössä. ”Omin silmin” koettuna havainnointi antaa paremman käsityksen tilan mittasuhteista ja objektien suhteista toisiinsa. (Pelosi 2017; Lappalainen 2016.)

Johnsin ja Lowen (2005) tutkimuksessa selvitettiin pelimoottorin käyttämistä arkkitehtien työvälineenä. Tutkimuksessa Wellingtonin Victoria-yliopiston oppilaat käyttivät Unreal-pelimoottoria keskeisenä työvälineenä projektissa, jonka tarkoituksena oli suunnitella muistomerkki World Trade Centren kunniaksi New Yorkin Staten Islandille. Tutkimuksessa selvisi, että pelimoottorin avulla opiskelijat pystyivät paremmin hahmottamaan mittasuhteita ja saivat täydellisemmän käsityksen ajasta ja avaruudesta. Lisäksi opiskelijoiden visuaalinen ja auditiivinen kokemus korostui. Opiskelijoiden mainitsemia haittapuolia olivat pelimoottoriohjelmiston jyrkkä oppimiskäyrä ja joskus haastava työnkulku.

Leuvenin yliopiston tutkimuksessa vuonna 2011 (Boeykens) tutkittiin pelimoottorien hyödyntämistä historiallisten rakennuskohteiden digitaalisissa rekonstruktioissa. Perinteisesti historiallisten kohteiden rekonstruktioissa on keskitytty kaksiulotteisiin CAD-piirustuksiin. Vaikka nykyään on kuitenkin siirrytty enemmän 3D-mallien käyttöön, niin näissäkin malleissa on käytetty tavanomaisia menetelmiä, minkä tuloksena mallit ovat staattisia. Ne eivät sisällä interaktiivisuutta eikä niitä voi tutkia reaaliaikaisesti. Pelimoottorien avulla "pelaaja" voi käydä tutkimassa historiallisen rakennuksen digitaalista mallia reaaliaikaisesti ja saada käsityksen siitä, kuinka kohde on rakennettu. Esineille voidaan lisätä hyperlinkkejä, joita pelaaja voi klikata ja näin saada lisää tietoa kohteesta. Pelimoottorien etuna on niiden visuaalinen näyttävyys ja se, että ne mahdollistavat joustavan vuorovaikutuksen pelaajan ja ympäristön välillä. Tarvittaessa kuvakulmaa voi vaihtaa pelaajan kuvakulmasta suunnittelijalle sopivaan kuvakulmaan. Esimerkiksi kamera voisi siirtyä pelaajan taakse niin että pelihahmo näkyy kameran edessä. Tätä kuvakulmaa suunnittelija voi hyödyntää saadakseen paremman käsityksen ympäristöstä. Suunnittelussa voidaan

ottaa huomioon myös liikuntarajoitteet, koska esimerkiksi pyörätuolilla liikkumista pystytään mallintamaan riittävällä tarkkuudella.

Pelillistämisen voi tarvittaessa ulottaa yksittäisistä rakennuksista kokonaisiin kaupunkimalleihin. Pelimoottoreiden avulla onkin jo simuloitu kokonaisia suomalaisia kaupunkeja muun muassa Oulussa ja Espoossa (Lappalainen 2016). Kaupunkimallintamisessa yksityiskohtien korvaaminen tekstuureilla ei olisi suuri ongelma, jos tarkoitus olisi käyttää malleja esimerkiksi kulkuyhteyksien suunnitteluun, koska tällöin rakennusten pienimmät yksityiskohdat olisivat toissijaisia. Toinen käyttötarkoitus voisi olla kaupunkien suurten rakennusten varjostuksen suunnittelu. Tällaiseen suunnitteluun tarvittavaa vuorokaudenajan mukaan realistisesti vaihtuvaa valaistusta pystytään simuloimaan esimerkiksi Unity-pelimoottorilla. Pelillistettyjä kaupunkimalleja pystyisi myös käyttämään koulutustarkoituksiin esimerkiksi pelastuslaitosten simulaatioissa (Lappalainen 2016).

Tavallisimmilla CAD- ja BIM-mallinnusohjelmilla toteutettu geometria saattaa kuitenkin johtaa monimutkaisiin ja raskaisiin malleihin. Kaksiulotteiset tekstuurit ovat kevyempiä käsitellä kuin kolmiulotteiset objektit, ja pelimoottoreissa geometrian vaatimaa laskentatehoa onkin kierretty käyttämällä tekstuureita, joiden avulla luodaan vain illuusio monimutkaisesta kolmiulotteisesta geometriasta. Kääntöpuolena on se, että tekstuurit eivät tarjoa rakennusalan tarpeisiin riittävän tarkkaa tietoa rakennuksen geometriasta. Rakennuksen mallia pystyttäisiin tällöin hyödyntämään vain luonnosteluun ja hahmottamiseen. Reaaliaikaisen grafiikan piirtämisessä siis korostuu geometrian, tekstuurien ja valaistuksen optimointi, mikä merkitsee usein sitä, että rakennussuunnittelun kannalta tärkeitä yksityiskohtia joudutaan karsimaan. (Pelosi 2017.)

2.3 Pelimoottorit

Tämän opinnäytetyön ei ole tarkoitus olla opas peliohjelmointiin. 3D-mallin pelillistäminen onnistuu hyvin vähäisellä ohjelmoinnilla, kun työkaluna käytetään valmista pelimoottoriohjelmistoa, ja siksi tässä selvitetään lyhyesti, mikä on pelimoottori. Pelimoottoriohjelmistoja on valmiiksi saatavilla sekä ilmaisina että maksullisina versioina. Pelimoottorin voi myös ohjelmoida kokonaan itse, mutta tämä on hyvin vaativa prosessi. (Thorn 2011.)

Pelimoottori ei ole aivan yksiselitteinen termi, eikä pelintekijöidenkään keskuudessa vallitse yksimielisyyttä siitä, kuinka pelimoottori pitäisi tarkalleen määritellä. Thorn lähtee kirjassaan (2011) siitä ajatuksesta, että peleissä on monia ominaisuuksia ja toimintoja, jotka toistuvat samankaltaisina kaikissa peleissä. Esimerkiksi grafiikka piirretään käytännössä kaikissa 2D- ja 3D-peleissä samoja periaatteita noudattaen. Myös pelimaailman sisäiset fysiikat perustuvat usein samoihin fysiikan lakeihin. Näitä toistuvia ominaisuuksia ei ole järkevää ohjelmoida joka kerta alusta asti uudestaan, vaan ne voidaan koostaa yhdeksi ohjelmistoksi. Tällaista ohjelmistoa kutsutaan pelimoottoriksi. Pelimoottori on siis ohjelmisto, joka helpottaa ja nopeuttaa pelinkehitystä. (Thorn 2011.)

Bakerin määrittelyn mukaan (2016) pelimoottori on pelin kehityksessä käytettävä ohjelmistokehys, joka koostuu viidestä eri osasta:

- pelin grafiikan piirtäminen
- tekoäly
- äänten käsittely
- fysiikkamoottori
- pelimekaniikka.

Pelimoottoreissa voi olla fysiikan simuloimista varten oma fysiikkamoottori, tai niihin on voitu integroida jokin erillinen fysiikkamoottori. Fysiikkamoottorit eivät ole kokonaisia pelimoottoreita, vaan pelkästään pelin fysiikan toteuttamiseen tarkoitettuja ohjelmistoja. Yksi ensimmäisiä fysiikkamoottoreiden käyttökohteita oli tykkien ammuksien lentoratojen laskeminen (Physics engine 2017). Tunnettuja fysiikkamoottoreita ovat esimerkiksi Havok ja nVidia PhysX (Laine 2017).

Rakennusmallin pelillistämisessä pelimoottorin tärkein ominaisuus on grafiikan piirtäminen. Pelimoottori kääntää 3D-mallin geometriatiedot näytöllä näkyväksi kuvaksi. (Lappalainen 2016.)

Käytännössä markkinoilla on kaksi varteenotettavaa ilmaista pelimoottoria, jotka ovat Unity 3D ja Unreal Engine. Molemmilla pelimoottoreilla on omat vahvuutensa, ja valinta kannattaakin tehdä sen perusteella, millaisen pelin ha-

luaa tehdä. Unreal Engine on hieman järeämpi, ja sillä voidaan tehdä graafisesti markkinoiden näyttävimpiä pelejä. Unreal Enginen aloituskynnys on kuitenkin Unity 3D:tä korkeampi (ValueCoders 2017). Seuraavaksi kerrotaan joitakin Unity 3D:n olennaisimmista ominaisuuksista, jotka ovat tarpeen rakennusmallin pelillistämisessä.

2.4 Unity 3D

Yksi tämän hetken tunnetuimmista ja suosituimmista pelimoottoreista on Unity Technologiesin kehittämä Unity 3D (käytetään myös nimeä Unity), joka soveltuu sekä 3D- että 2D-pelien kehittämiseen. Unityn verkkosivujen mukaan (Company Facts 2017) se on maailman suosituin ilmainen pelimoottori, ja sen suosio jatkaa kasvuaan. Esimerkiksi vuonna 2016 tuhannen suosituimman mobiilipelin joukosta 34 % oli toteutettu Unityllä. Unityn käyttäjinä on joitakin maailman suurimmista yrityksistä, kuten Coca-Cola, Disney ja Microsoft.

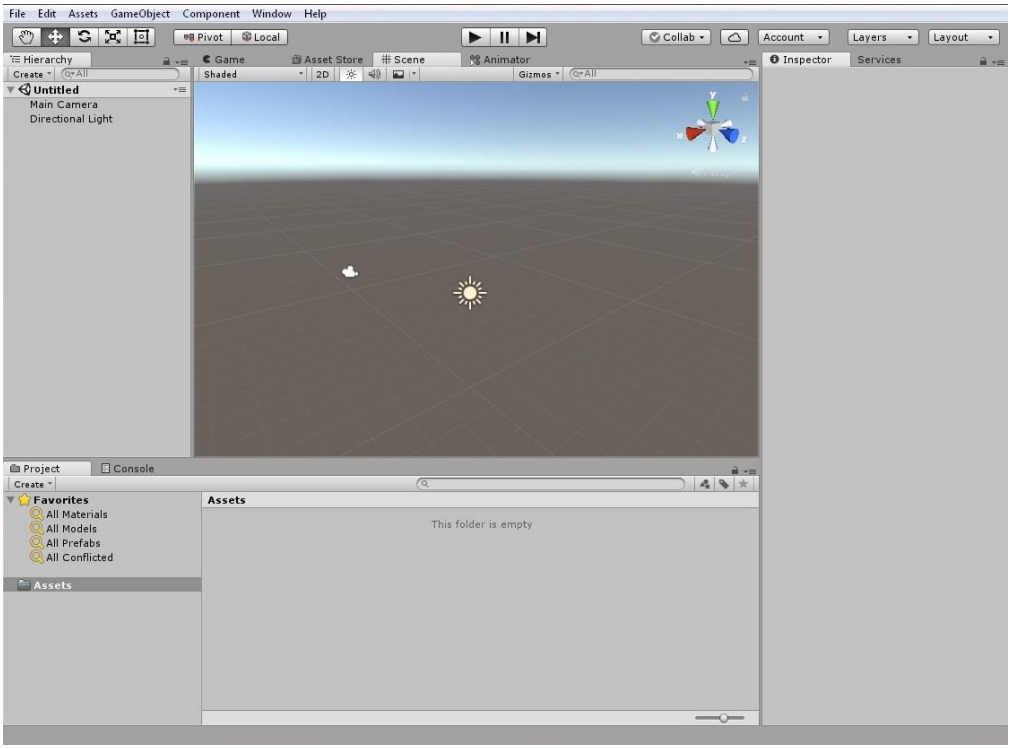
Vaikka Unitystä puhutaankin pelkästään pelimoottorina, niin siihen kuuluu pelkän moottorin lisäksi myös editori, joka tarjoaa työkalut moottorin hyödyntämiseen pelin kehityksessä (Selin 2017). Unityn vahvuuksiin kuuluukin juuri sen editorin käyttöliittymän selkeys (Baker 2016), mikä tekee siitä helposti lähestyttävän aloittelijallekin, eikä pelin tekeminen Unityllä vaadi suuria määriä ohjelmointia. Toisaalta Unity on myös ammattilaisille riittävän tehokas pelimoottori, ja sillä on toteutettu useita kaupallisesti menestyneitä pelejä, kuten Lara Croft: GO ja Deus Ex: The Fall (Sinicki 2016). Unityn vahvuuksia on myös se, että sillä voi kehittää pelejä useille eri alustoille. Unityn versio 2017.1 tukee kaikkia suosituimpia alustoja, joita ovat esimerkiksi

- Windows, Windows Phone
- Linux/Steam OS
- Mac
- Android
- iOS
- Tizen
- Xbox One
- Playstation 4
- Oculus Rift, Steam VR, Playstation VR
- WebGL.

Unitystä on saatavilla kolme eri versiota: Personal, Plus ja Pro. Unity Personal on ilmainen versio, jonka kuka tahansa voi ladata ja asentaa Unityn verkkosivuilta. Plus maksaa 35 dollaria kuukaudessa ja sisältää joitakin lisättyjä ominaisuuksia, kuten mainonnan helpompi lisääminen peleihin. Unity Pro on monipuolisin versio, ja sen käyttöoikeus maksaa 125 dollaria kuukaudessa. Sellaisten yritysten tai yksityishenkilöiden, jotka Unityn kaupallisella käytöllä saavuttavat yli 200 000 dollarin vuositulot, on käyttöehtojen mukaan ostettava Pro-versio. (Company Facts 2017.)

Unityn ominaisuuksia: assetit, peliobjektit ja scenet

Useimmissa ohjelmointiympäristöissä, kuten NetBeans ja Microsoft Visual Studio, kehitetyt ohjelmat ovat projekteja. Näin on myös Unityssä, eli jokainen peli on oma projektinsa.

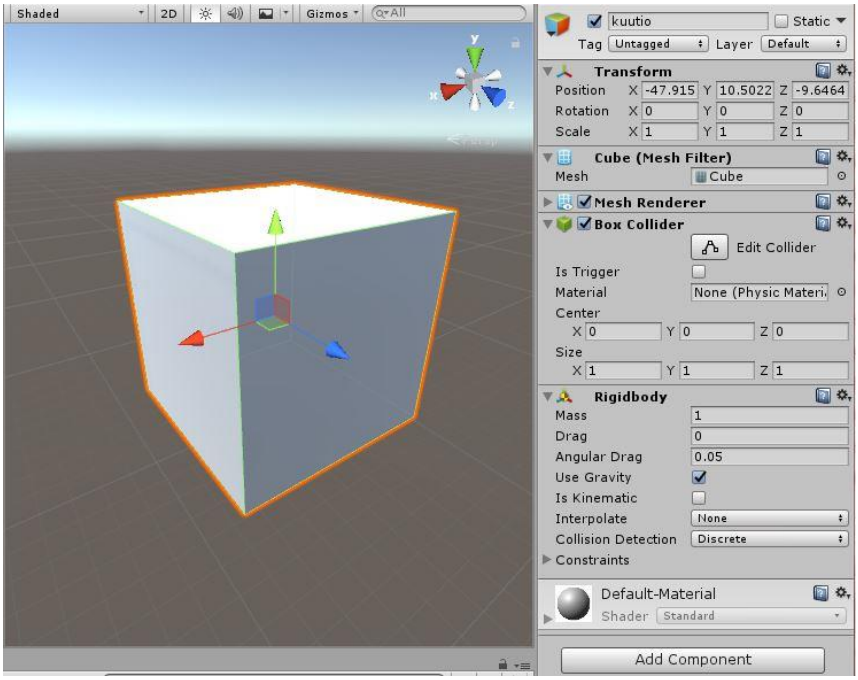


Kuva 2. Unityn perusnäkymä, jossa aktiivisena on scene-ikkuna

Unityssä kaikki pelin sisältö muodostuu asseteista ja peliobjekteista. Assetit ovat kokoelma kaikista pelissä tarvittavista elementeistä, joita voivat olla esimerkiksi äänet, kuvat, koodit ja 3D-mallit. Siis kaikki palaset, mitä pelissä käytetään, ovat assetteja. Unity sisältää valmiiksi laajan kirjaston assetteja, minkä lisäksi niitä voidaan tuoda ulkoisista tiedostoista tai asentaa Unityn sisäisestä

kauppapaikasta, Asset Storesta. Kauppa sisältää sekä ilmaisia että maksullisia assetteja.

Objektit ovat ohjelmoitavia peliolioita, joille määritellään toimintoja ja ominaisuuksia, eli ne ovat kuin tavallisia ohjelmointiolioita. Objekteja voidaan tehdä suoraan Unityn valikosta "GameObject" tai muodostaa asseteista. Pelkkä peliobjekti ei itsessään vielä sisällä mitään toimintaa, vaan toiminnallisuus lisätään komponenteilla, joita voidaan liittää peliobjekteihin. Esimerkiksi objektin paikka pelimaailmassa määritellään ”transform”-komponentissa, joka sisältää objektin paikkatiedot kolmiulotteisessa koordinaatistossa. Myös esimerkiksi objektien fyysiset ominaisuudet määritellään komponenteilla. Peli voisi sisältää vaikkapa peliobjektin "kuutio", joka on kuutionmuotoinen kolmiulotteinen kappale. Tähän objektiin voidaan lisätä komponentti "Rigidbody", joka lisää objektille fyysiset ominaisuudet, kuten massan, joten objekti tottelee nyt fysiikan lakeja. Kun objekti esimerkiksi asetetaan epätasapainoon jonkin korokkeen päälle, niin pelin käynnistyessä se putoaa korokkeelta alas fysiikan lakien mukaisesti. Ilman Rigidbody-komponenttia objekti jäisi paikoilleen, koska sillä ei olisi fyysisiä ominaisuuksia.



Kuva 3. Objekti nimeltä ”kuutio”, jonka komponentiksi on lisätty Rigidbody

Lisäksi objekteista voidaan koostaa valmiita objektipaketteja, joita voi helposti monistaa ja käyttää uudestaan. Näitä paketteja kutsutaan nimellä prefab.

Objekteja käsitellään ja lisätään peliin näkymässä, jota kutsutaan sceneksi. Käytännössä koko pelimaailma rakennetaan raahaamalla ja pudottamalla elementtejä scene-näkymään, ja juuri tämä graafinen käyttöliittymä tekee Unityn käyttämisestä helppoa (Selin 2017). Scenet jakavat pelin eri osiin, ja peli voi sisältää useita eri scenejä. Esimerkiksi pelillistetty sovellus, jonka tarkoitus on esitellä jotakin rakennusta, voisi sisältää kaksi sceneä. Yhdessä scenessä pelaaja liikkuu talon sisällä ja tutkii ympäristöä, ja toinen scene voisi olla esimerkiksi taukovalikko, joka aktivoidaan jollakin näppäinkomennolla. Scenet ovat myös assetteja ja tallentuvat asset-kansioon.

Unityn ominaisuuksia: törmäystunnistus

Törmäystunnistus on erittäin keskeinen termi peliohjelmoinnissa. Törmäyksen tunnistamisen avulla pelimaailman kappaleille kerrotaan, törmäävätkö ne toisiinsa vai voivatko ne kulkea toistensa läpi. Rakennusmallin pelillistämisessä törmäyksen tunnistusta tarvitaan, koska pelaajan täytyy pystyä liikkumaan rakennuksesta luodun 3D-mallin sisällä ja sen ympäristössä reaalielämää muistuttavalla tavalla. Ilman toimivaa törmäystunnistusta pelimaailmassa liikkuminen ei olisi reaalielämän mukaista, koska pelaaja voisi esimerkiksi kävellä seinien läpi.

Törmäystunnistus kuuluu fysiikkamoottorin tehtäviin, ja Unityn versiossa 5 ja sitä uudemmissa versioissa fysiikkamoottorina toimii PhysX 3.3 (Physics in Unity 5.0). Unity on hyvä sovellus rakennusmallin pelillistämiseen, koska fysiikkamoottorin ominaisuuksien hyödyntämistä varten on valmiiksi tarjolla helposti käytettävät työkalut. Tunnistimia voi luoda itse yksittäisille objekteille liittämällä uusi komponentti haluttuun objektiin. Unityn voi myös antaa automaattisesti luoda tunnistimet 3D-mallille, joka tuodaan Unityyn ulkoisesta sovelluksesta. Erilaisia törmäystunnistimia ovat esimerkiksi

- box collider
- sphere collider
- capsule collider
- mesh collider.

Box, sphere ja capsule collider ovat niin sanottuja primitiivi-collidereita. Ne luovat kappaleen ympärille yksinkertaisen törmäyksentunnistimen, joka on nimensä mukaisesti laatikon, pallon tai kapselin muotoinen. Nämä tunnistimet sopivat geometrisesti yksinkertaisille kappaleille. Esimerkiksi kuution muotoisen peliobjektin ympärille luodaan box colliderilla kuution muotoinen törmäystunnistin. (Colliders 2017.)

Monimutkaisempiin objekteihin soveltuu mesh collider, joka luo kappaleen ympärille sen polygoniverkon mallisen törmäyksentunnistimen. Mesh collider on tarkkuudeltaan parempi kuin primitiivi-colliderit. Haittapuolena on se, että mesh collider vaatii enemmän laskentatehoa, ja mitä tarkempi 3D-malli, sitä monimutkaisempi mesh collider vaaditaan. Usein törmäystunnistimissa tehdäänkin kompromisseja ja pyritään löytämään keskitie suorituskyvyn ja törmäystunnistimien tarkkuuden välillä. (Selin 2017.)

Lisäksi on olemassa myös compound colliderit, jotka ovat useiden törmäystunnistimien yhdistelmiä. Esimerkiksi pelimaailmassa sijaitsevan lehtipuun ympärille voidaan tehdä törmäystunnistin yhdistämällä pallon ja laatikon muotoinen tunnistin. Pallo rajataan latvan ympärille niin, että oksat ja lehdet jäävät tunnistimen sisään. Puun rungon ympärille tehdään samalla tavalla rajaamalla box collider. Tällainen tunnistin ei täysin mukaile kohteen muotoja, mutta sillä pystytään simuloimaan törmäyksiä tarkemmin kuin pelkällä primitiivi-colliderilla.

Unityn ominaisuuksia: objektien animointi

Unityssä peliobjekteille voi luoda animaatioita. Käytännössä animointi tapahtuu manipuloimalla objektin sijaintia ja asentoa pelimaailmassa.

Animator Controller sisältää viittaukset kaikkiin sen sisällä käytettäviin animaatioihin ja siirtyy näiden animaatioiden välillä käyttäen niin sanottua tilakonetta. Animaatioita voi raahata ja pudottaa Animator Controlleriin graafisen käyttöliittymän kautta. (Animator Controller 2017.)

Animointi on oiva tapa elävöittää ympäristöä, ja animaatioiden avulla voidaan vaikkapa jäljitellä puuttuvaa fysiikkamallinnusta. Esimerkiksi Unity ei simuloi

nesteiden fysiikkaa, mutta veden päällä kelluvat esineet voidaan saada näyttämään siltä, että ne liikkuvat veden laineiden mukana, kun niille luodaan sopivat animaatiot. Esimerkiksi vene, joka liikkuu laineiden mukana, voidaan animoida huojumaan niin, että sen liike mukailee aaltojen liikettä.

Kuva 4. Unityn veden päälle sijoitettu vene

Animaatioilla voidaan myös tehdä talon elementeille toimintoja. Esimerkiksi ovet ja ikkunat voidaan avata tekemällä niille animaatiot. Ovi voisi animaation aloituskohdassa olla 0 asteen kulmassa ja loppukohdassa 90 asteen kulmassa. Animointityökalulla määritellään tämän siirtymän kesto esimerkiksi niin, että oven kääntyminen nollasta 90 asteen kulmaan kestäisi 5 sekuntia.

Unityn ominaisuuksia: ohjelmointikielet

Unityllä koodia voi kirjoittaa natiivisti kahdella eri kielellä, jotka ovat UnityScript ja C#. UnityScript on Unityä varten kehitetty ohjelmointikieli, joka perustuu JavaScriptiin. C# on pelialalla yleisesti käytetty ohjelmointikieli, joka muistuttaa Javaa ja C++:aa. Unityssä koodeja kutsutaan skripteiksi (script), ja niitä voi liittää komponentteina haluttuihin objekteihin aivan kuten muitakin komponentteja. Skriptejä voi kirjoittaa Unityn omalla MonoDevelop-editorilla tai jollakin erillisellä koodieditorilla. Koodissa voidaan kutsua peliobjekteja normaalin olio-ohjelmoinnin tapaan.

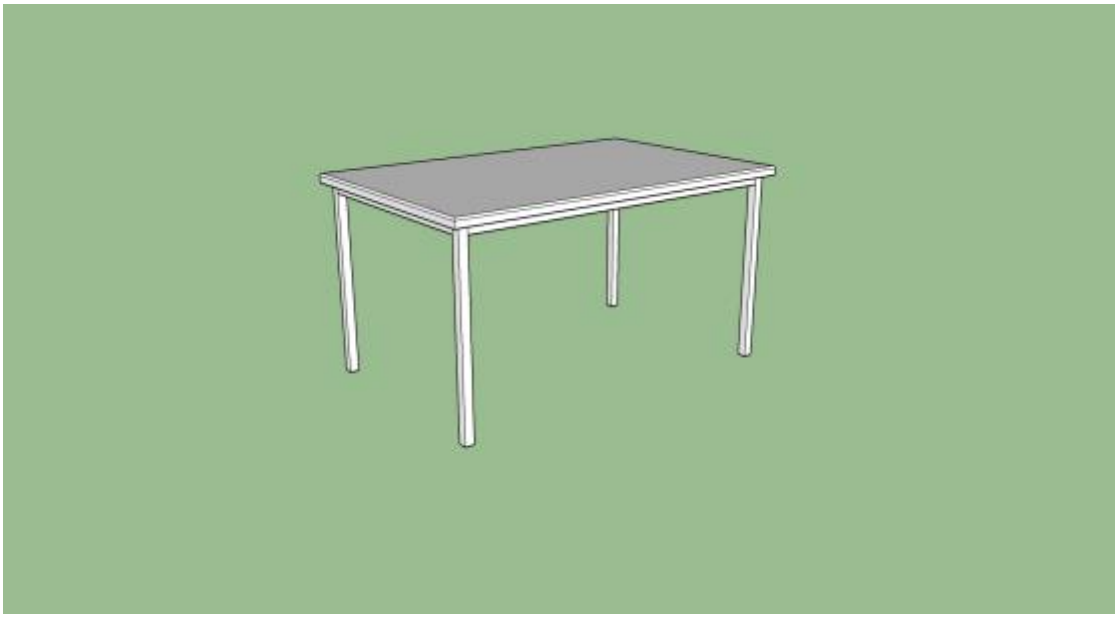
3 3D-MALLINTAMINEN

3D-mallintamisesta on hyvä tietää ainakin perusteet, vaikka pelillistettävä malli olisikin jo valmiiksi saatavilla jonkun toisen osapuolen tekemänä. Mallien käsittely pelimoottorissa on helpompaa, kun tietää millaisista osista ne koostuvat ja kuinka ne on tehty.

Viimeisen kahden vuosikymmenen aikana tietokonegrafiikasta on muodostunut hyödyllinen työkalu arkkitehtuurin, suunnittelun ja rakentamisen alalla (AEC-ala). Ensimmäiset 3D-mallinnusohjelmat syntyivät 1970-luvun alussa, ja niillä esitetty grafiikka oli vielä hyvin alkeellista. Rakennuksia alettiin mallintaa kolmiulotteisesti 1970- ja 1980-lukujen taitteessa, kun ensimmäiset CAD-ohjelmistot sekä Michiganin ja Carnegie-Mellonin yliopistojen omat ohjelmistot alkoivat sisältää mallintamisen perusominaisuuksia. Ensimmäiset mallinnusohjelmistot olivat kuitenkin epäkäytännöllisiä, koska sen aikaisissa tietokoneissa ei riittänyt laskentateho. Lisäksi ohjelmistot olivat erittäin kalliita: yksi käyttölisenssi saattoi maksaa jopa 35 000 dollaria (Eastman 2008). Laitteistojen sekä ohjelmistojen kehittymisen myötä mallinnusohjelmistojen hinta on laskenut ja saatavuus parantunut (Chopine 2011). Samalla 3D-mallintamisen tarve kasvaa, koska sen sovellusalueet laajenevat. Kasvavia alueita ovat esimerkiksi videopelit ja animaatio. Peliteollisuus on jo ohittanut elokuvateollisuuden liikevaihdon määrässä mitattuna, ja videopelien jatkuva kehitys vie samalla eteenpäin myös 3D-teknologiaa (Puhakka 2008).

3.1 Mikä on 3D-malli

3D-malli tarkoittaa digitaalista, matemaattisesti luotua esitystä jostakin kolmiulotteisesta kohteesta. Kohde voi olla olla jokin eloton kappale tai elävä olento, ja se voi olla todellinen tai kuviteltu. 3D-mallia voisi verrata esimerkiksi rakennuksen pohjapiirustukseen, joka puolestaan on kaksiulotteinen esitys kohteesta. Piirustuksella on kaksi ulottuvuutta: pituus ja leveys. 3D-mallilla on pituuden ja leveyden lisäksi myös syvyys. Termi 3D johdetaan englanninkielisistä sanoista "three dimensional", eli kolmiulotteinen. 3D-mallit ovat tietokonegrafiikan olennaisimpia rakennuspalikoita. Esimerkiksi useimmissa videopeleissä pelin hahmot ja objektit ovat 3D-malleja. Myös animointi tehdään 3D-mallien avulla. (Slick 2016a.)

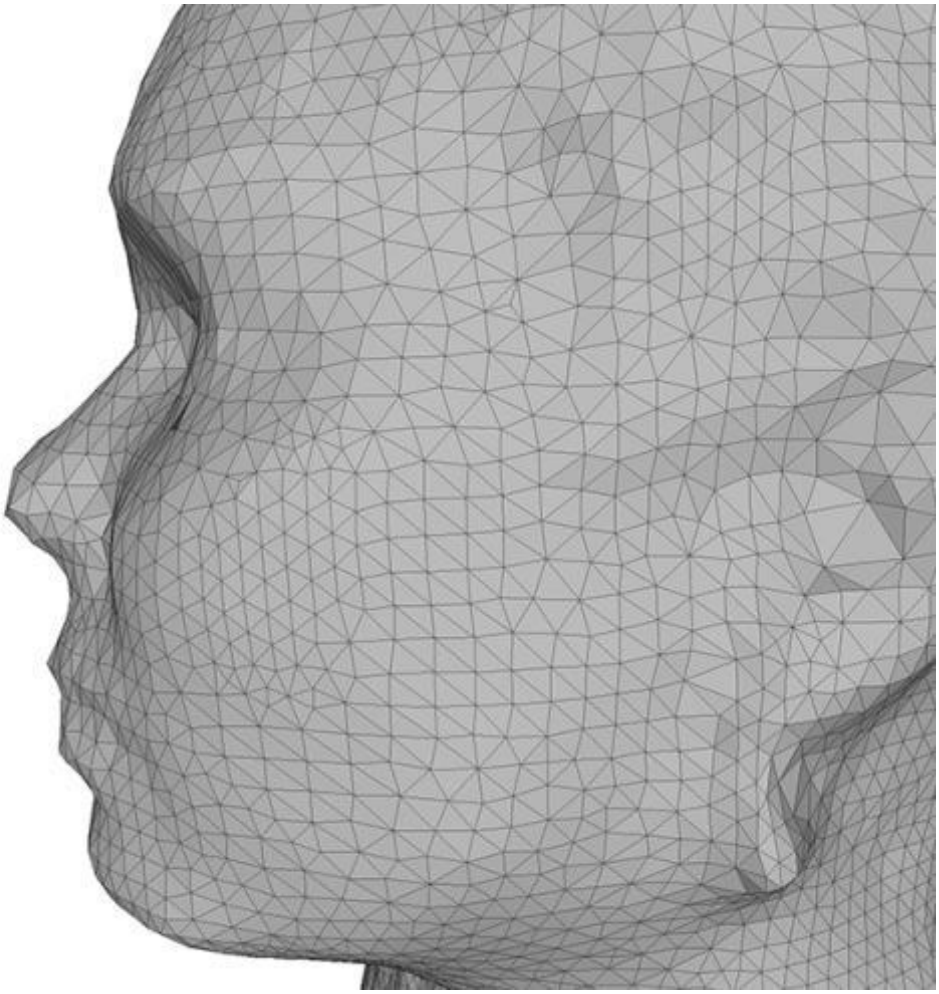


Kuva 5. Trimble SketchUpilla luotu yksinkertainen pöytää esittävä 3D-malli.

3D-malleja on useita eri tyyppisiä. Yleisin 3D-malli on polygonimalli, joka koostuu polygoneista (Slick 2016a). Se on vanhin tapa esittää geometrisia kappaleita tietokoneilla (Prunier 2009). Polygonissa eli monikulmiossa on usein kolme tai neljä kulmaa, ja se koostuu vertekseistä, reunoista ja pinnoista.

- Verteksi on kolmiulotteisessa tilassa sijaitseva piste.
- Reuna tarkoittaa kahden verteksin yhdistävää viivaa.
- Kolme tai useampi samalla tasolla toisiinsa liittyvää reunaa muodostavat pinnan.

Polygonilla on kaksi ulottuvuutta: leveys ja pituus. Ne ovat siis kaksiulotteisia ”rakennuspalikoita”, jotka muodostavat 3D-mallin pinnan. Polygonien muodostamaa pintaa kutsutaan polygoniverkoksi (mesh). Polygoniverkkoa voidaan verrata fyysiseen verkkoon, joka painetaan tiiviisti jonkin kappaleen päälle niin, että se mukailee kappaleen muotoja (Stevenson 2015).



Kuva 6. Polygoniverkko koostuu polygoneista (Stevenson 2015)

Vaikka polygoneissa usein onkin vain kolme tai neljä kulmaa, niin teoriassa polygonissa voi olla kuinka montaa kulmaa tahansa. Tämä ei kuitenkaan ole käytännöllistä, koska tietokoneelta vaaditaan sitä enemmän laskentatehoa mitä enemmän polygoneissa on kulmia. Kolmion muotoiset polygonit ovat yksinkertaisimpia ja niiden reaaliaikainen piirtäminen vaatii tietokoneelta vähiten tehoa. Tästä syystä ne ovat suosittuja varsinkin pelimoottoreissa toteutettavassa 3D-grafiikassa, koska peleissä grafiikka täytyy piirtää reaaliaikaisesti. Monet 3D-mallintajat suosivat myös nelikulmaisia polygoneja, koska ne on tarvittaessa helppo muuttaa kolmioiksi. (Chopine 2011.)

Käytännössä 3D-mallin tiedostokoko on suhteessa polygonien määrään. Polygonien määrä ei kuitenkaan itsessään määritä 3D-mallin laatua, vaan myös polygonien tiheys, eli resoluutio, on mallin laatuun vaikuttava tekijä (Slick 2016a).

3.2 3D-mallintamisen menetelmiä

3D-mallin tuottamista ja muokkaamista kutsutaan 3D-mallintamiseksi. Mallintaminen tehdään sitä tarkoitusta varten kehitetyillä tietokoneohjelmilla. 3D-malleja voidaan luoda useilla erilaisilla menetelmillä, ja mallinnusohjelmia on tarjolla eri mallinnusmenetelmiä varten. Tällaisia ohjelmistoja ovat esimerkiksi Trimble Sketchup, 3DS Max, Maya ja Blender. (Slick 2016d.)

Polygonimallinnus eli mesh-mallinnus on kevyt ja kaikkein yleisin mallinnusmenetelmä. Polygonimallinnuksella luodut mallit ovat kevyitä käsitellä. Polygonimallinnusta monimutkaisempia ja raskaampia menetelmiä ovat parametroituihin polynomikäyriin perustuvat tekniikat, kuten Spline- ja NURBS-mallinnus (non-uniform rational basis spline) (Puhakka 2008). Nämä menetelmät perustuvat Bézier-käyriin, joiden muotoja muokataan ohjausvertekseistä eli ohjauspisteistä, ja käyrien väliin muodostetaan kaarevia pintoja. Näillä menetelmillä saadaan aikaiseksi luonnollisempia ja sulavampia malleja kuin polygonimallinnuksella, koska ne ovat matemaattisesti tarkempia. Bézierkäyrät kehitettiin alunperin teollisuudenalojen tarpeisiin (Puhakka 2008). NURBS- ja Spline-mallinnusta käytetäänkin varsinkin teollisessa mallintamisessa (Slick 2016b).

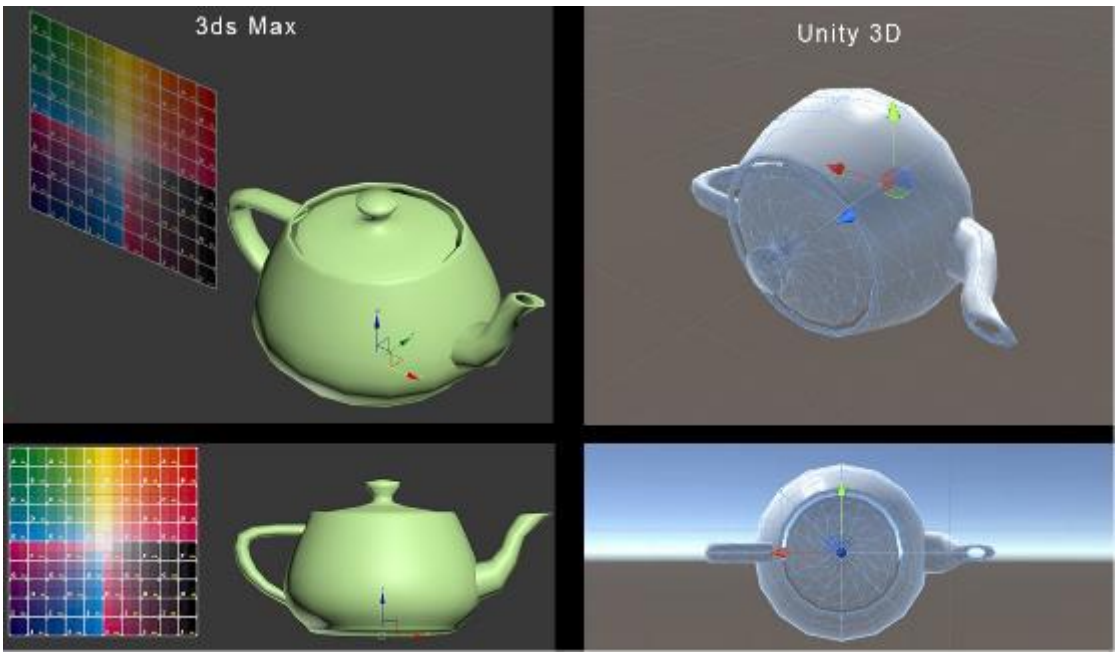
3D-malleja voidaan myös tuottaa kuvaamalla kohde eri kulmista. Tällä menetelmällä saadaan muodostettua kohteesta stereokuva, jonka avulla määritetään sen syvyysrakenne. Kohteita voidaan myös mitata kolmiulotteisesti, mikä perustuu lasermittaukseen. Näiden menetelmien tuloksena saadaan pistepilvi, joka kuvaa kohteen pintaa. Tämä pistepilvi muutetaan sen jälkeen polygoniverkoksi. Laajempia maastoalueita voidaan mitata ilmasta käsin esimerkiksi lentokoneella tai satelliiteilla. Lisäksi on olemassa fraktaalitekniikoita, jotka soveltuvat maanpintojen realistiseen mallintamiseen. (Puhakka 2008.)

Mallinnusmenetelmä ja -ohjelmisto on järkevää valita sen mukaan, mihin 3D-mallia aiotaan jatkossa käyttää. Hannus (2017) muistuttaa, että huonosti toteutettu 3D-malli saattaa aiheuttaa ongelmia silloin kun se viedään pelimoottoriin. Ongelmia aiheuttavia tekijöitä voivat olla esimerkiksi toisiaan leikkaavat pinnat. 3D-mallin pelillistäminen pitää siis huomioida jo mallin tuotantovaiheessa. Pahimmillaan esimerkiksi asuntoesittelyä varten luotu rakennusmalli

on pelimoottoriin vietynä hyödytön, jos sille ei pystytä luomaan törmäyksentunnistimia. Tällainen ongelma saattaa aiheutua esimerkiksi vääränlaisesta tiedostomuodosta (Selin 2017). Tiedostomuotoja käsitellään lisää myöhemmin.

Pelimoottorit eivät renderöi NURBS-malleja, vaan käyttävät polygonimalleja. Suurimmat grafiikan piirtämiseen tarkoitetut ohjelmointirajapinnat, Direct3D ja OpenGL, on suunniteltu polygonien piirtämiseen (Puhakka 2008). Tästä syystä ei siis ole tarkoituksenmukaista mallintaa NURBS-menetelmillä, jos malleja halutaan käyttää pelimoottorissa.

Eri 3D-mallinnusohjelmat ja pelimoottorit käyttävät myös erilaisia koordinaatistoja, mistä saattaa aiheutua ongelmia. Esimerkiksi Unityssä on käytössä vasemman käden koordinaatisto, jossa y-akseli osoittaa ylöspäin, kun taas Autodeskin 3ds Max -mallinnusohjelmassa z-akseli osoittaa ylöspäin. Unreal Engine -pelimoottorissa on käytössä vasemman käden koordinaatisto, jossa z-akseli osoittaa ylöspäin. Näistä erityyppisistä koordinaatiston akseleista johtuen 3D-mallit saattavat olla väärässä asennossa, kun ne siirretään alkuperäisestä mallinnusohjelmasta vaikkapa pelimoottoriin. (Portelli 2015.)



Kuva 7. 3ds Maxin ja Unityn koordinaatistot eroavat toisistaaan (Portelli 2015)

Sellaisten objektien akselit, joille halutaan luoda animaatioita, kannattaa suunnata oikein jo 3D-mallinnusohjelmassa. Esimerkiksi ovet saattavatkin aueta

vääristä kohdista pelimoottorissa, jos niiden akselit ovat väärissä kohdissa. Pelimoottorin puolella akseleita ei enää voi muuttaa.

3D-malleja voidaan valaista reaaliaikaisesti, tai mallin pinnoille voidaan laskea etukäteen staattinen valaistus. Reaaliaikaisella valaistuksella mallista saadaan dynaaminen, koska varjot muodostuvat valonlähteen mukaan, vaikka valaistava kohde tai valonlähde olisi liikkeessä. Ennalta laskettu valaistus tunnetaan myös nimellä ”baked lighting”, eli valaistus ”leivotaan” kohteen pinnoille laskemalla jokaiselle polygonille valon perusteella kirkkausarvo, joka tallennetaan valotekstuureihin. Pintojen tekstuurien väri kerrotaan sen jälkeen valotekstuurin sisältämällä kirkkausarvolla. Valotekstuurit ovat staattisia, eli ne eivät muutu dynaamisesti valonlähteen mukana. Leivottu valaistus vaatii reaaliaikaista valaistusta vähemmän laskentatehoa. (Puhakka 2008.)

Nykyaikaiseen 3D-grafiikkaan liittyvät myös erilaiset tehosteet, joilla malleista saadaan realistisempia ja näyttävämpiä. Joissakin mallintamisohjelmissa on omat renderöintimoottorit. Peleissä efektit kuitenkin yleensä ovat pelimoottorin tehtäviä, eli 3D-malli tuodaan pelimoottoriin ja efektit tehdään reaaliaikaisesti pelin aikana. Tällaisia tehosteita ovat esimerkiksi kuhmutus (bump mapping) ja ohjelmoitavat sävyttimet (shader). Varsinkin sävyttimet ovat vieneet reaaliaikaista tietokonegrafiikkaa eteenpäin. Sävyttimistä käytetään myös termiä varjostin, mutta se ei ole yhtä kuvaava termi kuin sävytin. Sävyttimet ovat pieniä ohjelmia, joilla voidaan tehokkaasti laskea pinnoille tehosteita. Verteksisävytin suorittaa geometristen arvojen laskemisen, ja pikselisävytin laskee väriarvot yksittäisille pikseleille. Esimerkiksi verteksisävyttimien avulla kappaleen pinnalle on mahdollista luoda veden pintaa muistuttava aaltoileva efekti, kun sävyttimet muokkaavat reaaliaikaisesti verteksien sijainteja. Kuhmutus on tekniikka, jonka avulla tasaiselle pinnalle muodostetaan näennäisesti epätaisaisuutta, jotta siitä saadaan realistisemman näköinen. Todellisuudessa pinnan geometria ei kuitenkaan muutu, vaan sen pinnalle muodostetaan epätasaisen pinnan valoja ja varjoja jäljitteleviä tekstuureja. (Puhakka 2008.)

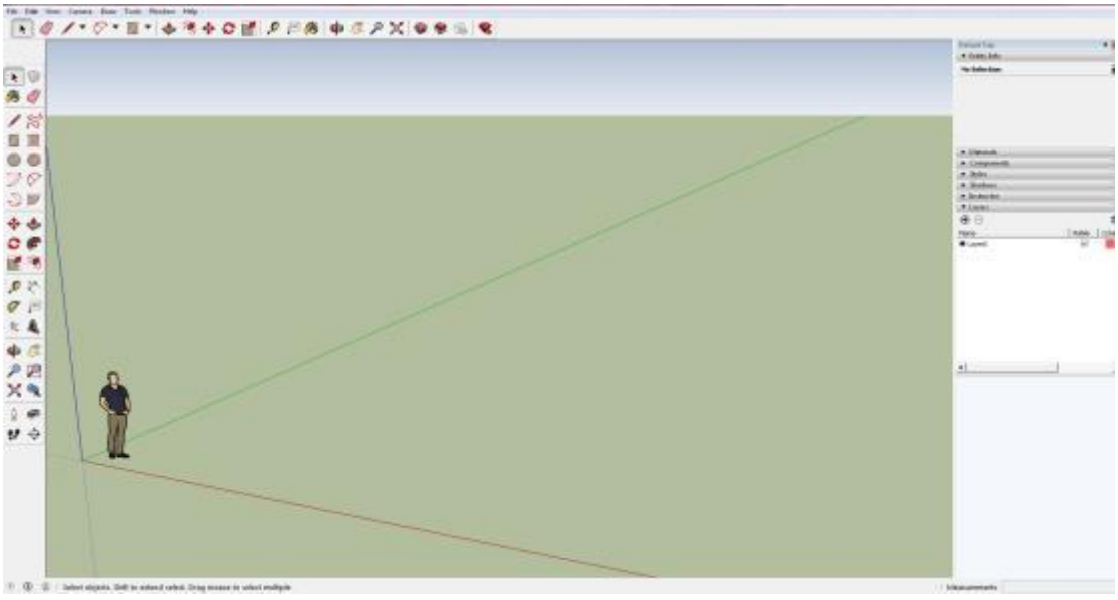
3.3 SketchUp

SketchUp on Trimblen omistama polygonimallinnusohjelma, jonka ensimmäinen versio julkaistiin vuonna 2000. SketchUpin alkuperäinen kehittäjä oli

@Last Software, jonka Google osti vuonna 2006. Nykyään SketchUpin omistava Trimble osti ohjelman Googlelta vuonna 2012 (SketchUp School 2017).

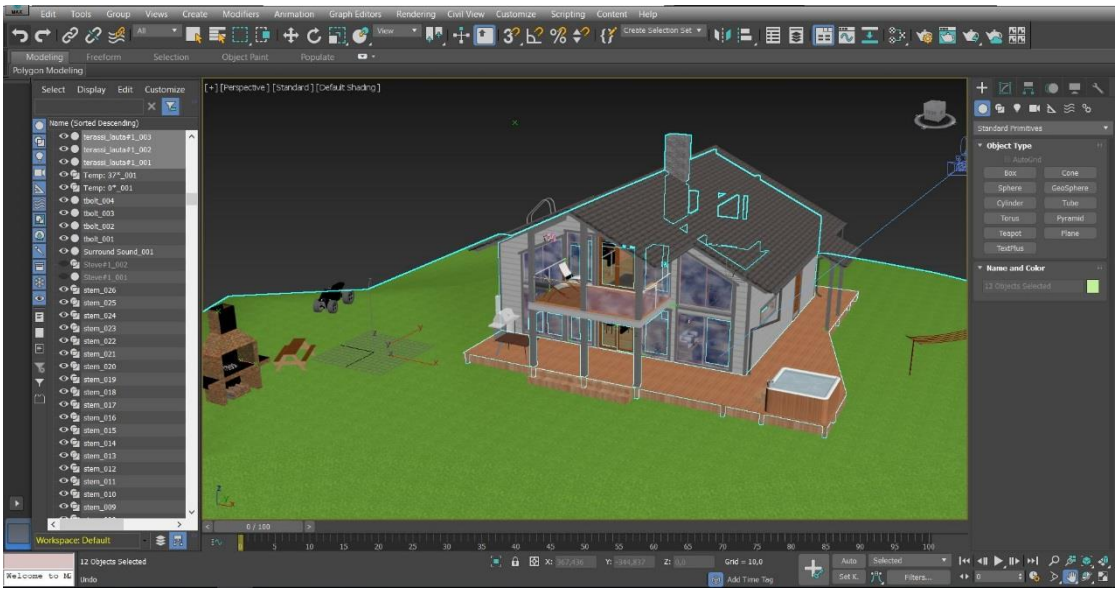
SketchUpilla mallintamisen aloittaminen on helppoa ja nopeaa käyttäjäystävällisen käyttöliittymän ansiosta, ja se on helposti lähestyttävä ohjema 3D-mallintamisesta kiinnostuneille. Yksinkertaisesta käyttöliittymästä huolimatta SketchUpilla voi luoda laadukkaita 3D-malleja, ja se on yksi yleisimmistä rakennusmallintamisessa käytetyistä ohjelmista (Importing Objects 2017). SketchUp sopii moniin muihinkin tarkoituksiin kuten tuotesuunnitteluun tai vaikkapa 3D-tulostamiseen (AN-Cadsolutions 2015).

SketchUpista on saatavilla kaksi eri versiota. SketchUp Make on ilmainen versio, joka sopii varsinkin aloittelijoille. SketchUp Pro on ohjelman maksullinen versio, joka sisältää enemmän toimintoja, ja tukee useampia eri tiedostomuotoja. Sketchup-mallit tallennetaan natiivisti skp-päätteisinä tiedostoina, ja lisäksi niitä voi viedä myös muihin tiedostomuotoihin, kuten COLLADA (.dae) ja Google Earth (.kmz). Maksullisella versiolla tiedostoja voi lisäksi viedä muotoihin 3DS, DWG, DXF, FBX, OBJ, VRML, XSI, PDF, EPS, EPIX ja tuoda muodoista DWG ja DXF (AN-Cadsolutions). Molemmat versiot sisältävät kuitenkin kaikki perusominaisuudet joita 3D-mallintamisessa tarvitaan (SketchUp School 2017). Alla olevassa kuvassa näkyy SketchUpin perusnäkymä. Kuvasta voidaan nähdä, että SketchUp muistuttaa ulkoasultaan hieman Microsoftin Paint-piirto-ohjelmaa.



Kuva 8. SketchUpin perustyökalupalkit

Vertaillaan SketchUpin perusnäkymää 3ds Max -ohjelman näkymään. Tässä vertailussa havaitaan, kuinka paljon monimutkaisemmalta 3ds Max näyttää. Sen ulkoasu saattaa jopa pelästyttää uuden käyttäjän.



Kuva 9. 3ds Maxin käyttöliittymä eroaa SketchUpista

SketchUpissa piirretään kaksiulotteisia geometrisia muotoja, kuten suorakulmioita, joiden pintoja manipuloimalla niistä muodostetaan kolmiulotteisia kappaleita. SketchUp tarjoaa geometristen muotojen piirtämiseen valmiit työkalut, jotka löytyvät työkalupalkista. Esimerkiksi suorakulmion ja ympyrän piirtämiseen on valmiit työkalut.

SketchUpilla kaarevat pinnat eivät todellisuudessa ole mateemaattisesti kaaria. Tämä johtuu siitä, että Sketchup on polygonimallinnusohjelma, ja kuten aiemmin on todettu, niin polygonimallien pinnat koostuvat polygoneista, jotka ovat kulmikkaita. Esimerkiksi ympyrät ovat Sketchupissa oletuksena 24-kulmaisia monikulmioita. Toiminta perustuu siihen, että ihmissilmälle esimerkiksi 24-kulmainen monikulmio näyttää ympyrältä. Kulmien määrää voi muuttaa ja näin muodostaa kaarelle sulavamman pinnan.

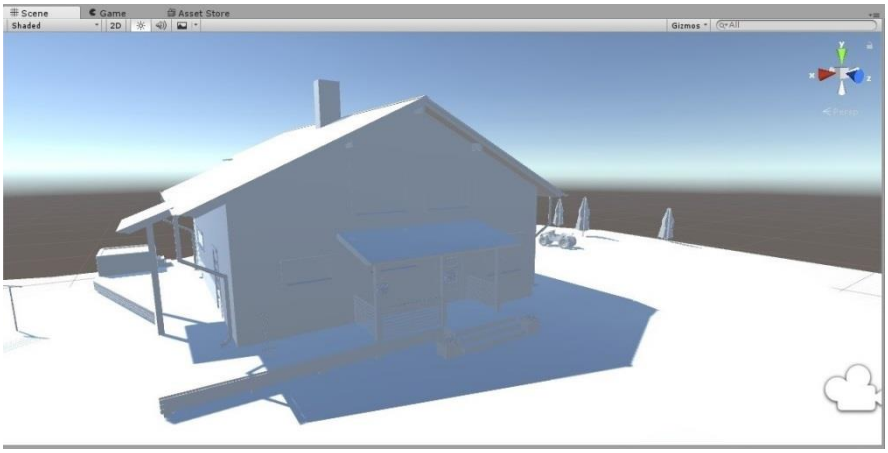
Elementtejä voidaan koota ryhmiksi ja niistä voidaan tehdä komponentteja. Komponentteja kannattaa käyttää esimerkiksi talomallinnuksessa, kun halutaan monistaa jokin talon elementti samanlaisena eri kohtiin taloa. Tällaisia komponentteja voivat olla esimerkiksi ikkunan- ja ovenkarmit. Komponenttitiedot säilyvät, kun malli siirretään Unityyn. Unity tunnistaa siis eri komponentit,

joita talomalli sisältää. Tästä on hyötyä, kun halutaan luoda toimintoja talon eri osille.

3.4 Tiedostomuodot

Teoriassa 3D-mallin tiedot voidaan määritellä suoraan ohjelman lähdekoodissa, mikä on kuitenkin hyvin kömpelö menetelmä, ja sitä tulisi välttää. Parempi vaihtoehto on tallentaa 3D-mallin tiedot erilliseen tiedostoon, jota eri ohjelmat osaavat lukea. Näin mallia voidaan siirrellä eri ohjelmien välillä. Yksinkertaisimmat tiedostomuodot, kuten OBJ, sisältävät vain mallin geometriatiedot. Kehittyneemmät tiedostomuodot, kuten FBX, sisältävät monipuolisempaa dataa, ja siksi FBX on nykyään laajalti käytetty tiedostomuoto. (Prunier 2009.)

3D-mallin tiedostomuoto vaikuttaa siihen, kuinka malli toimii pelimoottorissa. Esimerkiksi Sketchupista collada-muotoon viedyn natiivin skp-tiedoston tekstuurit katoavat, kun se tuodaan Unityyn. Tästä syystä ei ole mitään järkeä muuttaa tai tuoda SketchUpin 3D-malleja missään muussa muodossa kuin alkuperäisessä skp-muodossa, jos niitä on tarkoitus käyttää Unityssä.



Kuva 10. Collada-formaatissa tekstuurit eivät siirry Unityyn

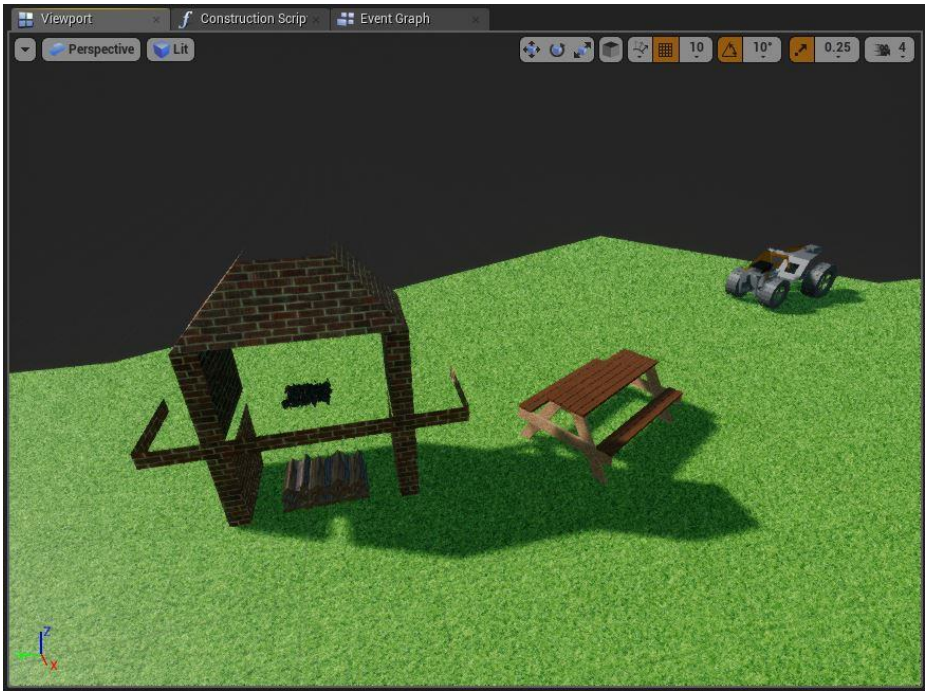
Toisaalta jos tarkoituksena on käyttää jotakin toista pelimoottoria, esimerkiksi Unreal Engineä, niin tässä tapauksessa FBX-tiedostomuoto on ainoa vaihtoehto. Unreal Engine ei nimittäin tue skp-tiedostoja. FBX-formaatin etuna onkin juuri sen yhteensopivuus pelimoottoreiden kanssa (Chakravorty 2017).

Tiedostojen muuttaminen kierrättämällä niitä ensin jonkin toisen ohjelman kautta ei ole täysin ongelmatonta. Avasin SketchUpilla tehdyn 3D-mallin ensin

3ds Maxissa, ja vein (export) tiedoston uudestaan FBX-muodossa. Kun mallin toi Unityyn, niin törmäystunnistimien automaattinen generointi ei onnistunut, minkä lisäksi osa tekstuureista puuttui. Unityn piti laskea mallin pinnoille uudet normaalit, minkä jälkeen törmäystunnistimien toiminta oli silti vain välttävää. Pelimaailmassa liikkuminen ei ollut mahdollista. Myös Unreal-moottorissa toistuvat samat ongelmat, minkä lisäksi alkuperäisen mallin ikkunoiden läpinäkyvät tekstuurit menettävät läpinäkyvyytensä, kuten voidaan nähdä kuvassa 11.

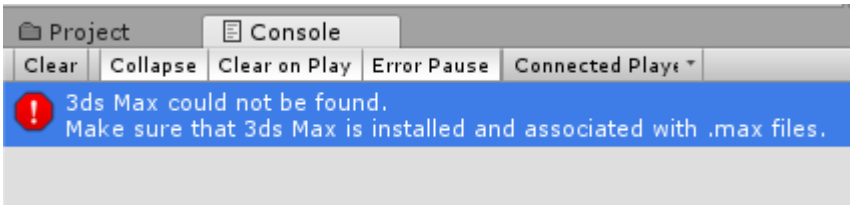
Kuva 11. Unreal-pelimoottorissa ikkunoiden läpinäkyvät tekstuurit menettävät läpinäkyvyytensä

Mallista myös puuttui kokonaan useita pintoja. Tällaisia virheitä oli havaittavissa ainakin pihan objekteissa ja parvekkeen kaiteessa. Tämän lisäksi talon sisätilojen pintoja puuttui runsaasti.



Kuva 12. Pihan objektit eivät siirry pelimoottoriin ilman virheitä

3ds Maxin oman max-tiedostoformaatin ongelmana on se, että sitä ei voi käyttää pelimoottoreissa, jos samalle laitteistolle ei ole asennettu myös 3ds Max –ohjelmaa. Tämä johtuu siitä, että max-formaatti on Autodeskin omistama ja suunnittelema tiedostomuoto (proprietary file), eli sitä pystyy muokkaamaan vain samalla ohjelmalla millä se on luotu (Chakravorty 2017). Tästä syystä vapaat formaatit ovat joustavampia vaihtoehtoja.



Kuva 13. Unityssä ei voi käyttää max-tiedostoja, jos 3ds Max ei ole asennettuna samalle tietokoneelle

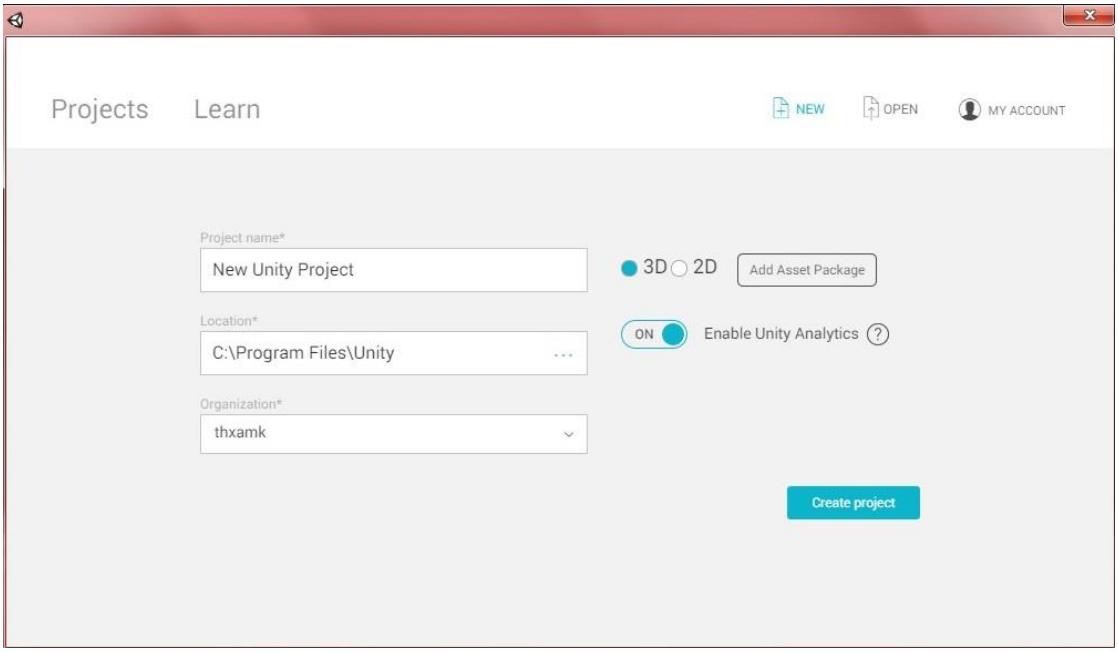
Jos mallien tiedostoformaatteja siis halutaan muuttaa, kannattaa se tehdä samalla ohjelmalla millä mallit on alun perinkin luotu käyttämällä niiden omia vientiominaisuuksia. Paras vaihtoehto olisi kuitenkin välttää kokonaan formaatin muuttamista. Tämä vaatisi sen, että tuotantoprosessin alussa määritellään mitä ohjelmia ja formaatteja käytetään. Lisäksi mallin tuotantovaiheessa tulisi varmistaa, että mallintamismenetemät ovat virheettömiä. Nämä ovat samoja havaintoja, kuin Pelosin (2017) tutkimuksessa.

4 DEMO AIHEESTA

Tätä opinnäytetyötä varten tein aiheeseen liittyvän demon. Tavoitteena oli pelillistää rakennusmalli sellaiseksi, että pelaaja voi liikkua vapaasti talon sisätiloissa ja ulkopuolella, ja näin saada käsityksen siitä, miltä ympäristö tosielämässä näyttää.

Tämän demon rakennusmallina on käytetty Mikkelin vuoden 2017 asuntomessujen kohdetta numero 13, Saimaan Aava (Asuntomessut Mikkelissä 2017). Rakennus on kaksikerroksinen hirsitalo, joka sijaitsee rinteen harjalla. Rakennuksen 3D-mallin olen tehnyt aiemmin keväällä 2017 opintoihin liittyvänä harjoitustyönä. SketchUp oli tähän tarkoitukseen loistava ohjelma kolmesta syystä. Ensimmäinen syy on se, että en ollut koskaan aiemmin tehnyt 3D-malleja, joten ohjelman täytyy olla aloittelijaystävällinen. Toinen syy on se, että SketchUp sisältää maailman suurimman kirjaston ilmaisia 3D-malleja, joita saa vapaasti hyödyntää omissa projekteissa (Find a 3D model of anything 2017). Tämä on hyödyllistä esimerkiksi siksi, että rakennuksen huonekaluja ei tarvitse itse mallintaa, vaan sisustamiseen voi käyttää valmiita 3D-malleja. Kolmas ja kenties tärkein syy on se, että ohjelman pitää olla yhteensopiva Unity-pelimoottorin kanssa. Rakennuksesta oli tarkoitus luoda ulkoisesti kohdetta muistuttava 3D-malli saatavilla olevien mittatietojen mukaisesti. Rakennuksen pohjapiirustuksessa on annettu seinien pituudet, sekä ovien ja ikkunoiden leveydet ja korkeudet, joten näiltä osin rakennus oli mahdollista mallintaa kiitettävällä tarkkuudella. Kaikista talon elementeistä ei kuitenkaan ollut saatavilla tarkkoja mittoja. Esimerkiksi katon korkeutta ei ole mainittu piirustuksissa. Mikäli mittatiedot olivat puutteelliset, käytin apuna yleisiä rakennusmääräysten mittoja.

Ensimmäisenä täytyy luoda Unityyn uusi projekti. Unityn käynnistämisen yhteydessä ensimmäisenä avautuu automaattisesti näkymä uuden projektin aloittamista varten. Jos Unity on jo käynnissä, voidaan klikata vasemmasta yläkulmasta File > New project. Projektin tyypiksi voi valita 3D tai 2D. Valitaan 3D, koska tässä ei olla tekemässä 2D-peliä. Annetaan projektille eli pelille haluttu nimi ja valitaan tallennuskansio. Halutessa voidaan myös valita asset-paketteja, jotka tuodaan heti mukaan projektiin. Kun kaikki asetukset ovat kohdillaan, valitaan Create project, minkä jälkeen siirrytään uuden projektin editoriin.

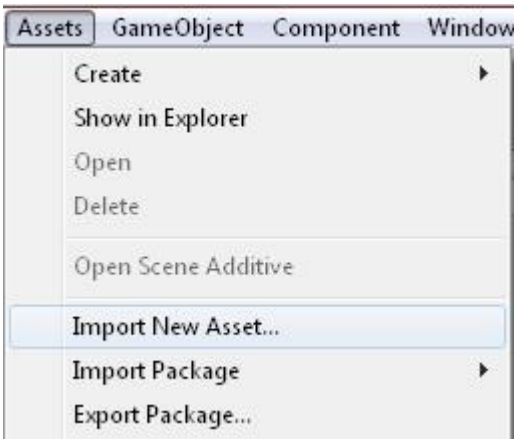


Kuva 14. Unityn uusi projekti luodaan ensimmäisenä

Uuteen projektiin kannattaa ensimmäisenä tuoda kaikki tarvittavat assetit. Toki pelin kehittämisen aikana saattaa ilmaantua tarve tuoda uusia assetteja, ja niitä voidaan tuoda projektiin sitä mukaa kun niitä tarvitaan.

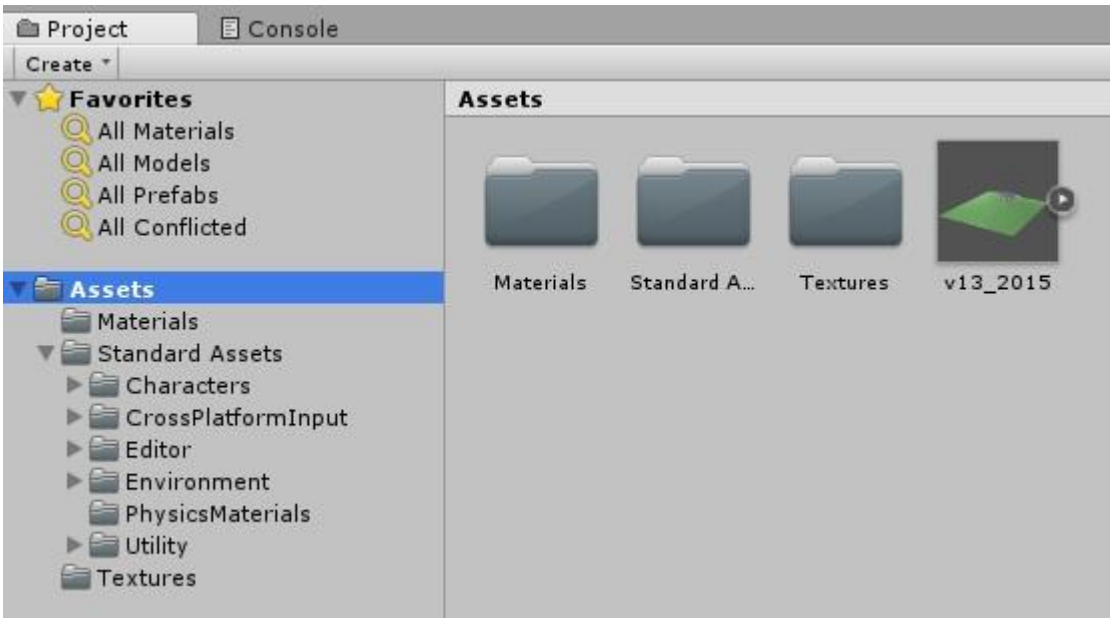
Tarvittavia assetteja ovat rakennuksen 3D-malli, pelihahmon liikkumiseen tarvittavat assetit ja ympäristöön liittyvät assetit. Asseteille voi tehdä omia alikansioita, tai ne voi tuoda suoraan Assets-pääkansioon. Kansiolla ei ole muuta merkitystä kuin se, että assetteja on helpompi hallita, kun ne löytyvät järkevästi nimetyistä kansioista.

Tuodaan projektiin ensimmäisenä itse rakennusmalli. Vaikka Unity tukeekin suoraan Sketchupin natiivia .skp-tiedostomuotoa, niin tästä huolimatta SketchUpin eri versioiden välillä esiintyy ongelmia, koska todellisuudessa Unityn versio 2017.1 ei vieläkään täysin tue SketchUpin versiota 2017. Tästä syystä malli pitää tallentaa ensin SketchUp 2015 -muodossa, mikä onneksi onnistuu helposti SketchUpin kautta. Kun malli on tallennettu oikeasssa muodossa, se voidaan tuoda Unityyn. Valitaan ylävalikosta Assets > Import New Asset, tai klikataan hiiren oikealla napilla alareunan assets-ikkunassa, mikä avaa saman valikon. Tämän jälkeen valitaan tiedosto, joka assetteihin halutaan tuoda.



Kuva 15. Assetit tuodaan valikosta valitsemalla ”Import New Asset…”

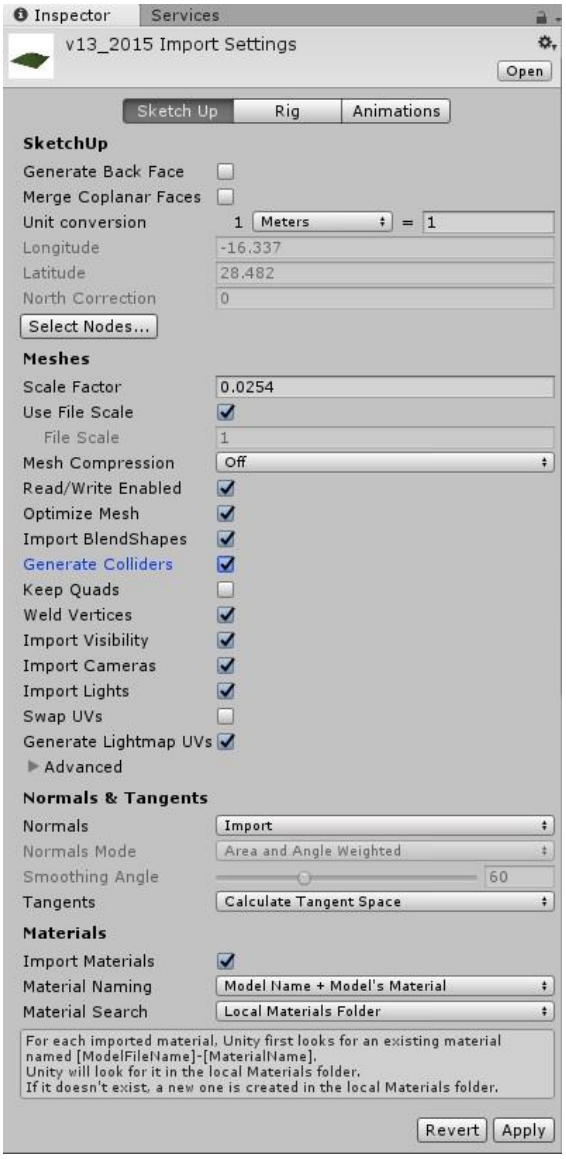
Assetit ilmestyvät alareunan ”Assets”-näkymään kun ne on onnistuneesti tuotu projektiin. Nyt voimme siis ihastella Assets-kansioon ilmestynyttä rakennusmallia, joka löytyy kansiosta omalla tiedostonimellään ”v13_2015”. Muokkasin mallista SketchUpissa useita versioita, joten se on nimetty sillä logiikalla, että ensin ilmoitetaan versionumero eli 13, ja loppuosa kertoo SketchUpin version eli 2015. Tämä helpottaa tiedostojen hallintaa, koska käyttäjä näkee heti tiedoston nimestä, mikä versio on kyseessä.



Kuva 16. Tiedosto v13_2015 on onnistuneesti tuotu Unityn assetteihin

Kun haluttu tiedosto on tuotu Unityn assetteihin, kannattaa myös tuontiasetukset viilata kohdilleen, ennen kuin mallia lähdetään siirtämään sceneen. Tässä tapauksessa talolle täytyy luoda ainakin törmäystunnistimet. Tämä onnistuu helpoiten, kun valitaan ensin talon asset, ja sen jälkeen oikean reunan Inspec-

tor-ikkunasta laitetaan rasti kohtaan ”Generate Colliders”. Nyt Unity tekee mallille automaattisesti törmäystunnistimet, jotka ovat tyyppiä ”Mesh collider”. Samalla laitetaan rasti myös kohtaan ”Generate Lightmap UVs”. Tämä täytyy tehdä siksi, että talolle halutaan myöhemmin laskea valotekstuurit. Muita muutoksia asetuksiin ei tarvitse tehdä, joten vahvistetaan valinnat klikkaamalla ”Apply”.



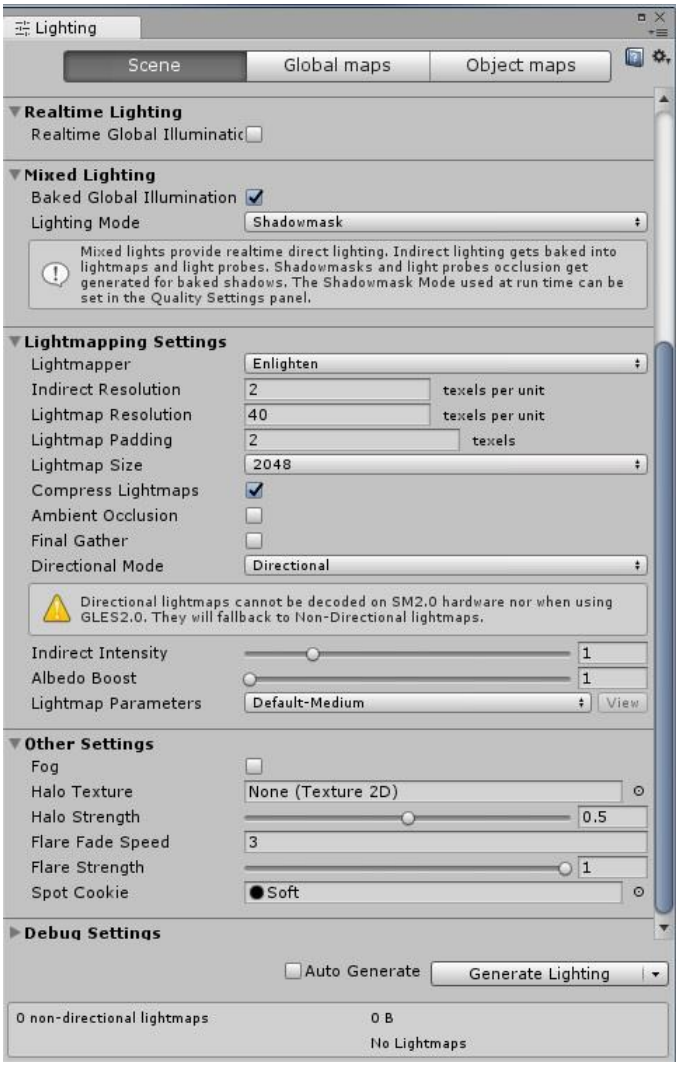
Kuva 17. Luodaan 3D-mallille törmäystunnistimet valitsemalla ”Generate Colliders”

Tuodaan projektiin seuraavaksi kaikki muut tarvittavat assetit paketteina, joista ensimmäisenä vaikkapa ”Characters”-paketti. Tämä paketti sisältää muun muassa pelimaailmassa liikkumiseen tarvittavat assetit. Paketit tuodaan projektiin samasta valikosta kuin yksittäiset assetit, mistä valitaan tällä kertaa ”Import Package”, ja sen jälkeen ”Characters”. Tämän jälkeen avautuu valikko, josta

voi vielä yksitellen valita mitä paketin sisältöä projektiin halutaan tuoda. Valitaan kaikki ja klikataan Import. Kaikki muutkin asset-paketit tuodaan samalla tavalla. Jos on tiedossa, että kaikkia paketin assetteja ei tulla tarvitsemaan projektissa, niin kannattaa valita vain tarvittavat assetit, koska paketit kasvattavat nopeasti projektin kokoa.

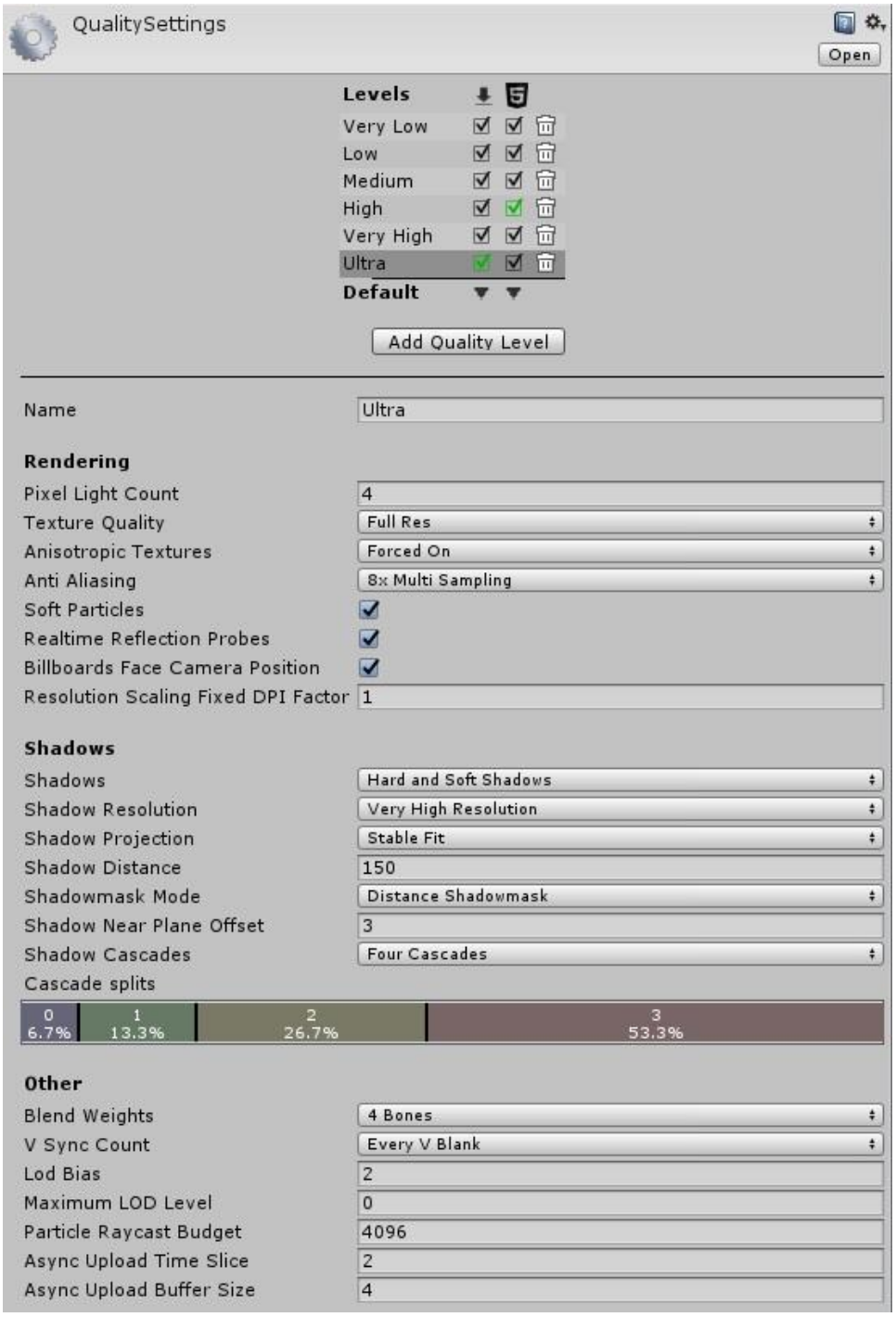
Kuva 18. Asset-paketeista voi valita kaikki assetit tai tarvittaessa valita yksittäisiä assetteja

Tässä vaiheessa kannattaa vielä tehdä joitakin muutoksia asetuksiin. Valotekstuurit halutaan laskea vasta sitten, kun kaikki pelin objektit on aseteltu paikoilleen sceneen. Oletuksena Unity kuitenkin haluaa aloittaa laskemisen automaattisesti aina kun sceneen tehdään muutoksia. Tämän asetuksen voi vaihtaa valikosta Window > Lighting. Välilehdeltä ”Scene” löytyy alareunasta valinta Auto Generate, jonka kohdalta otetaan rasti pois, joten valaistuksen laskeminen aloitetaan vasta sitten kun käyttäjä klikkaa kohdasta ”Generate Lighting”.



Kuva 19. Scenen valaistukseen liittyvät asetukset kannattaa säätää kohdilleen

Lisäksi projektin yleisiä grafiikka-asetuksia voi muuttaa, kun valitsee valikosta Edit > Project Settings > Quality Settings. Reunanpehmennys eli antialiasointi on oletuksena kaksinkertainen. Reunanpehmennys vähentää objektien reunojen sahalaitoja, mikä tekee niistä terävämpiä, ja asetuksista voidaan ottaa käyttöön kahdeksankertainen reunanpehmennys. Myös varjojen laatua voi muuttaa, kun kohtaan ”Shadow Resolution” valitaan korkein mahdollinen resoluutio. Näillä asetuksilla grafiikasta saadaan mahdollisimman näyttävää, vaikka ne vaativat enemmän laskentatehoa.



Kuva 20. Reunanpehmennystä ja varjojen tekstuurien laatua voi nostaa asetuksista

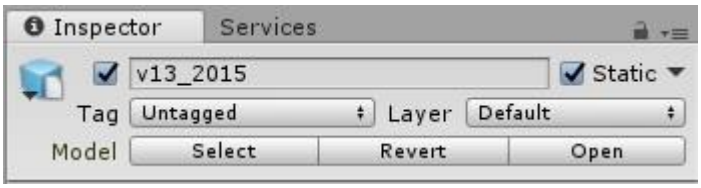
Kun kaikki assetit on tuotu projektiin ja asetukset viritetty kohdilleen, niin voidaan alkaa rakentaa sceneä. Ensimmäisenä sceneen on järkevintä siirtää talo, koska se muodostaa suurimman osan pelialueesta.

Tämä rakennusmalli ei itse asiassa ole optimaalisella tavalla mallinnettu pelillistämisen näkökulmasta, koska kaikki elementit rakennuksesta huonekaluihin ja maanpintaan ovat yhtä ja samaa 3D-mallia. Järkevämpää olisi tuoda objektit erillisinä 3D-malleina, koska se pienentää yksittäisen mallin tiedostokokoa,

ja lisäksi malleja on helpompi hallinnoida Unityssä. Myös törmäystunnistimet on tarkempi luoda yksinkertaisemmalle mallille. Tämä muodostaa mielenkiintoisen tutkimuskysymyksen projektia varten. Pystytäänkö tekemään toimiva sovellus, vaikka ei olisi käytetty optimaalisia menetelmiä?

Valitaan talo-asset ja raahataan se sceneen. Jos pudottaminen ja raahaaminen ei onnistu, kannattaa varmistaa, että aktiivisena on scene-ikkuna. Game-ikkunaan ei nimittäin voi suoraan lisätä elementtejä, vaan sen tarkoitus on toimia pelin esikatseluna. Game-ikkunasta voi siis kätevästi tarkastella, miltä peli näyttää pelaajan eikä pelintekijän näkökulmasta.

Kun talo on lisätty sceneen, siitä tehdään staattinen objekti, koska valotekstuurit voidaan laskea etukäteen vain sellaisille objekteille, joiden sijainti tai asento ei muutu pelin aikana. Määrittely onnistuu inspector-ikkunasta, kun malin nimen vieressä lisätään raksi ruutuun ”Static”. Unity kysyy tässä vaiheessa, halutaanko kaikista aliobjekteistakin (child objects) tehdä staattisia, mihin vastataan kyllä. Todellisuudessa kaikki talon elementit eivät tule olemaan staattisia, koska ovelle halutaan luoda animaatiot. On kuitenkin helpompi ensin muuttaa kaikki elementit staattisiksi, ja sen jälkeen käydä yksitellen ottamassa ruksi pois muutamasta kohdasta. Eli seuraavaksi valitaan sellaiset ovet, jotka halutaan muuttaa takaisin dynaamisiksi objekteiksi, ja otetaan niiden kohdalla Static-ruksi pois.



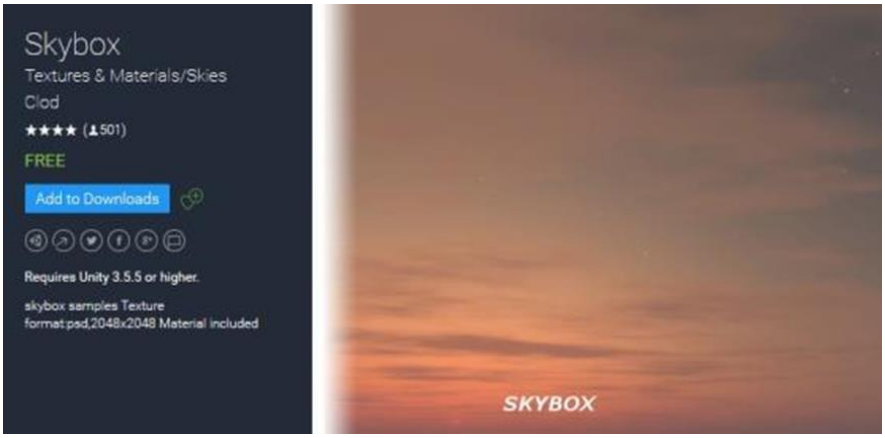
Kuva 21. Objekti määritellään staattiseksi insepctor-ikkunassa

Unity tekee automaattisesti uuteen projektiin yhden Directional Light –objektin, joka on valonlähde, ja sen tyyppinä (mode) on oletuksena Realtime, eli se valaisee scenen reaaliaikaisesti. Tyypiksi pitää vaihtaa Baked, jos halutaan, että Directional Lightia käytetään etukäteen laskettujen valotekstuurien valonlähteenä. Tyypin voi vaihtaa objektin ominaisuuksista Inspector-ikkunassa. Samalla pystyy muuttamaan valon muitakin ominaisuuksia, kuten voimakkuutta ja väriä. Myös valon langettamien varjojen tyyppiä voi muuttaa. Vaihtoehdot ovat Hard Shadows ja Soft Shadows. Varjot voi myös kokonaan poistaa. Soft

Shadows luo pehmeitä varjoja ja on paras vaihtoehto, kun halutaan luoda mahdollisimman realistiset varjot. vaikka ne vaativatkin enemmän laskentatehoa kuin Hard Shadows. Valon intensiteettiä on nostettu 1,8:aan, kun oletuksena intensiteetti on 1. Valon värillä ja kulmalla voidaan lisätä realismia. Eli silloin kun aurinko paistaa korkealta, on se yleensä kirkkaimmillaan ja varjot ovat lyhyitä. Jos aurinko paistaa matalalta, varjot ovat pidempiä, ja valo saa oranssin sävyjä. Tässä demossa valossa on käytetty oranssin sävyä, mikä luo vaikutelman iltahämärästä auringonlaskun aikaan.

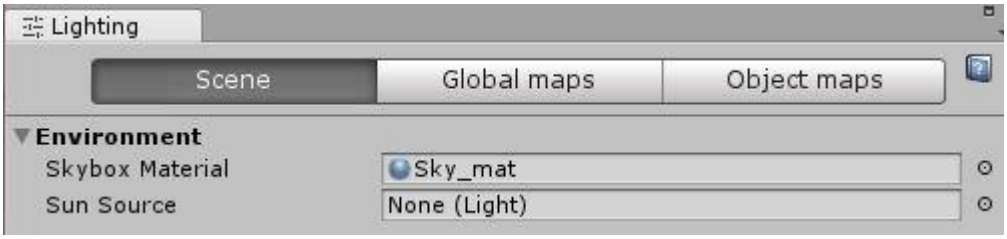
Kuva 22. Directional Light –objektin asetuksia

Realismin tuntua lisää, kun pelin taustalla näkyy jokin maisema tai vähintään taivas. Peleissä yleinen ratkaisu on skybox, joka on pelimaailman ympärille käärittyä tekstuuri. Skybox renderöidään koko scenen ympärille, mikä antaa vaikutelman, että horisontissa näkyy jokin maisema. Skybox voidaan tehdä itse mistä tahansa kuvista tai siihen voi käyttää Unityn valmiita tekstuureja. Asset Storesta löytyy myös skyboxeja sekä ilmaisina että maksullisina. Tässä projektissa on käytetty Asset Storesta ladattua ilmaista WorldSkies-skyboxia.



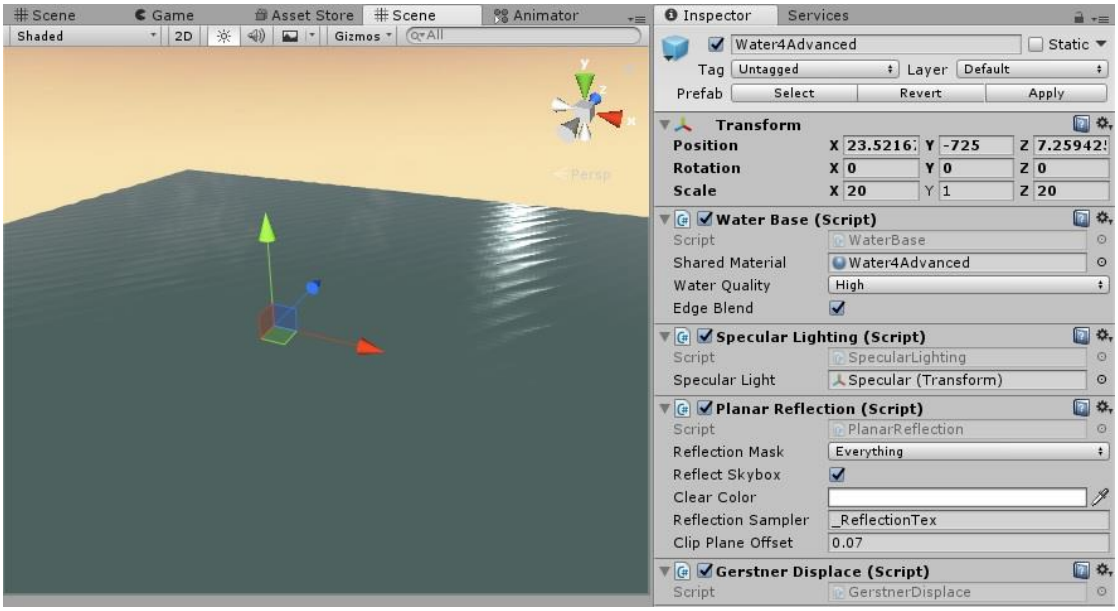
Kuva 23. Unityn kauppapaikasta voi asentaa ilmaisen skybox-tekstuurin

Skyboxin tekstuuri asetetaan scenen valoasetuksista, mihin päästään valitsemalla ylävalikosta Window > Lighting > Settings. Valitaan välilehti ”Scene” ja asetetaan ”Skybox Material”-kohtaan materiaaliksi Sky_mat.



Kuva 24. Skyboxin materiaalin voi vaihtaa asetuksista

Lisätään ympäristöön vielä järvimaisema niin talo ja sen tontti eivät näytä kelluvan jossakin tyhjässä avaruudessa. Järvi- tai merimaiseman luominen ei vaadi muuta kuin oikean assetin, ja tähän tarkoitukseen löytyykin jo sopiva asset Environment-paketista. Veden mallintamiseen on valmiina useita eri prefabeja, joista valitaan tähän tarkoitukseen sopiva ”Water4Advanced” ja raahataan se sceneen. Tämä luo sceneen suorakulmion muotoisen, vedenpintaa esittävän objektin. Objektia kannattaa suurentaa niin, että sen reunat eivät näy pelaajan silmiin asti, mikä luo illuusion horisonttiin asti yltävästä merenpinnasta. Vedenpinta on myös hyvä esimerkki sävyttimien käytöstä reaaliaikaisessa grafiikassa.



Kuva 25. Järvimaisema on helppo luoda, kun käytetään valmista prefabia

Valaistuksen asetteluun on paljon vaihtoehtoja. Suurin osa lampuista on tehty niin, että niissä on käytössä sekä spotlight että point light, ja niiden tyypiksi on asetettu baked. Poikkeuksina alakerran makuuhuoneen lamppu, eteisen kattovalo ja etuoven yläpuolella sijaitseva pihalamppu. Näissä valoissa tyypiksi on asetettu realtime, koska makuuhuoneen ovi ja etuovi ovat dynaamisia objekteja, joten valo ja varjot käyttäytyvät realistisemmin ovien pinnoilla, kun ne valaistaan reaaliaikaisesti. Toki myös parvekkeen ja kuistin ovet ovat dynaamisia objekteja, mutta näissä on tehty kompromissi, eli ne valaistaan vain staattisesti.

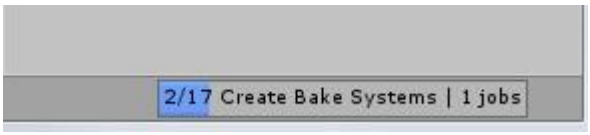
Point light ja spotlight on aseteltu samaan kohtaan niissä lampuissa, joissa on käytössä molemmat valot. Näiden kahden valon yhdistelmällä saadaan lamput näyttämään aidommilta. Point lightin tehtävä on simuloida lampun pistemäistä valoa, ja spotlightin tehtävä on suunnata valokeila ja luoda varjoja. Tästä syystä point lightin varjot ovat poissa käytöstä. Spotlight puolestaan luo varjoja, joiden tyyppinä on Soft Shadows. Spotlighteissa voidaan määritellä valokeilan kulma, eli sillä määritellään, kuinka laajan alueen keila valaisee. Kulmat ja valokeilan etäisyys (range) on määritelty ympäristön mukaan niin, että valaistus näyttää aidolta. Esimerkiksi yläkerran aulan kattovalossa on 175 asteen kulma ja etäisyys 6.

Kuva 26. Yläkerran aulan kattolampussa on käytössä spotlight.

Pihalampuissa point lightin etäisyys on pieni. Esimerkiksi etuoven puolella olevissa pihalampuissa etäisyys on vain 0,5. Valokeilan muodostaa spotlight.

Kuva 27. Pihalampuissa on käytössä point light, jonka etäisyys on lyhyt

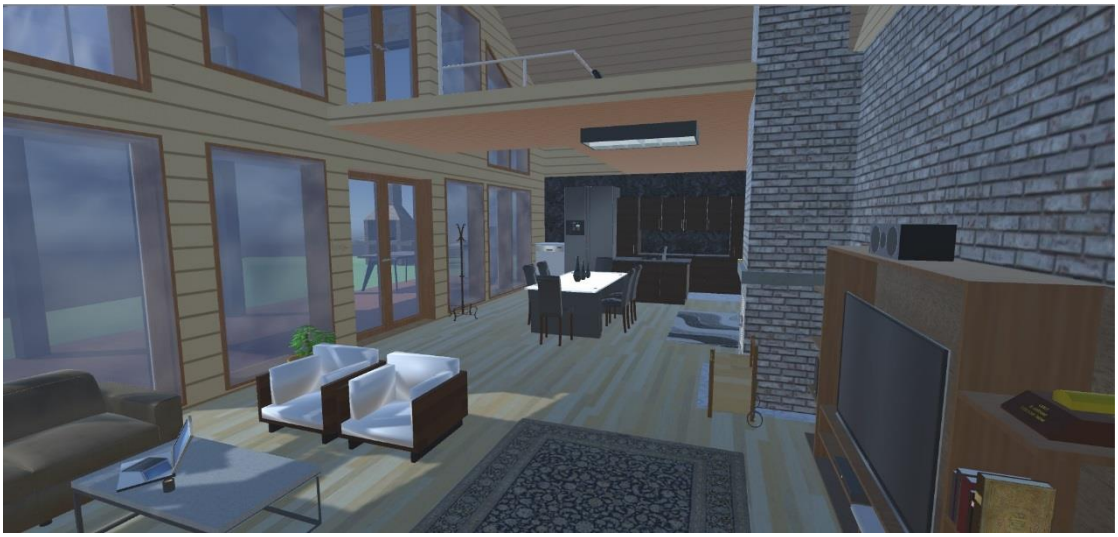
Valaistus voidaan laskea, kun kaikki valot on sijoitettu paikoilleen. Käydään siis klikkaamassa asetuksista Generate Lighting, minkä jälkeen Unity ilmoittaa laskennan olevan käynnissä oikeaan alakulmaan ilmestyvällä palkilla.



Kuva 28. Unityn oikeaan alakulmaan ilmestyy palkki, joka kertoo, että valaistusta lasketaan

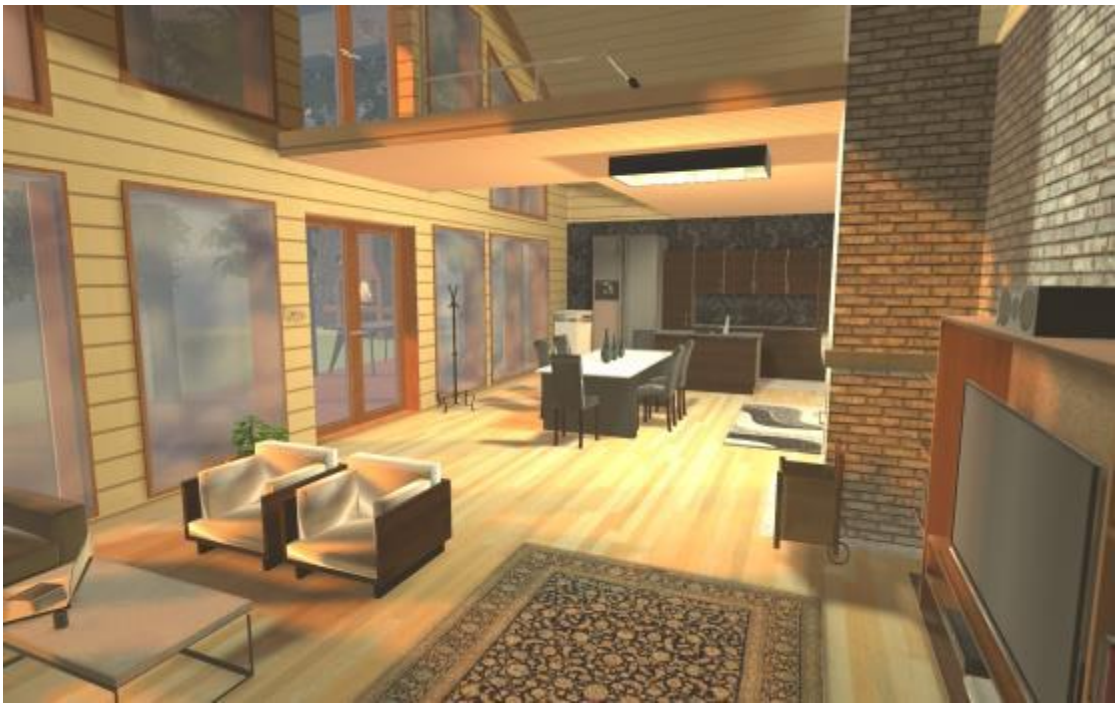
Laskennassa on yhteensä 27 vaihetta, ja valaistuksen laskemiseen kului noin 25 minuuttia. Laskenta tapahtuu taustalla, joten editoria voi käyttää samaan aikaan kun laskenta tapahtuu.

Vertaillaan miltä ympäristö näyttää, kun käytössä on pelkkä directional light oletusasetuksilla. Nyt huomataan, kuinka suuri merkitys valaistuksella on realismin kannalta. Ympäristö huokuu värimaailmaltaan kylmyyttä, eikä se näytä elävältä.



Kuva 29. Ympäristö Unityn oletusvalaistuksella

Huomataan myös, että valaistus vaikuttaa realismin lisäksi esteettisyyteen. Valaistu scene on sekä realistisempi että tyylikkäämpi. Huolellisesti suunniteltu valaistus tekee ympäristöstä houkuttelevamman näköisen markkinoinnin näkökulmasta.



Kuva 30. Sisätilojen valaistuksessa on käytetty valmiiksi laskettuja valotekstuureja

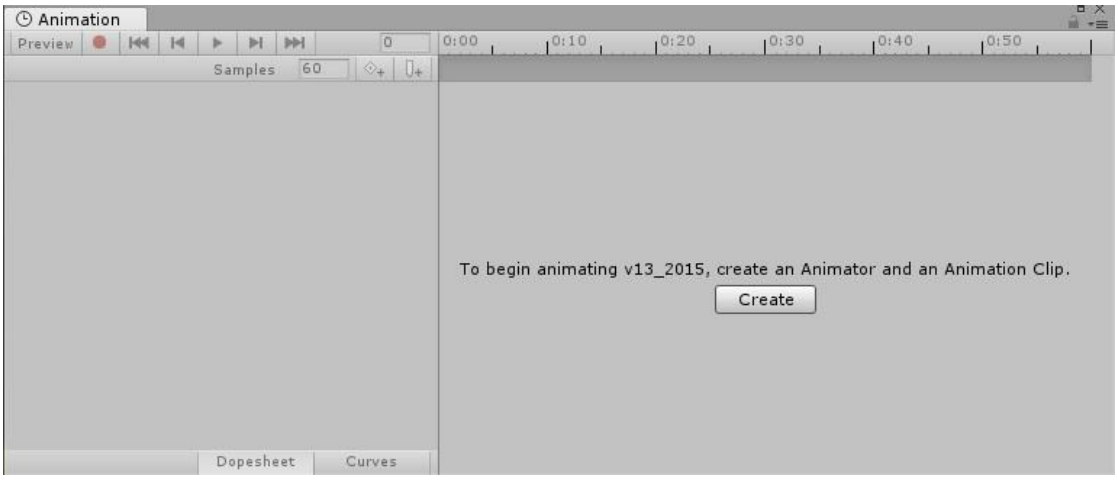
Alakerran makuuhuoneessa on käytössä reaaliaikainen valaistus. Makuuhuoneen kattolampun kohdalle on sijoitettu point light, joka loistaa valoa jokaiseen suuntaan. Tämän lisäksi samaan kohtaan on aseteltu spotlight, joka loistaa valokeilan tiettyyn suuntaan. Valot on suunnattu kattolampusta lattiaan päin, ja ainoastaan spotlight muodostaa varjoja.

Kuva 31. Alakerran makuuhuoneessa on käytetty reaaliaikaista valaistusta

Yläkerran makuuhuoneessa on käytössä etukäteen laskettu valaistus, koska huoneessa ei ole liikkuvia objekteja. Valonlähteiksi on määritelty point light ja spotlight, samalla tavalla kuin alakerran makuuhuoneessa. Tosin kirkkautta on säädetty hieman korkeammaksi.

Kuva 32. Yläkerran makuuhuoneessa on käytetty etukäteen laskettua valaistusta

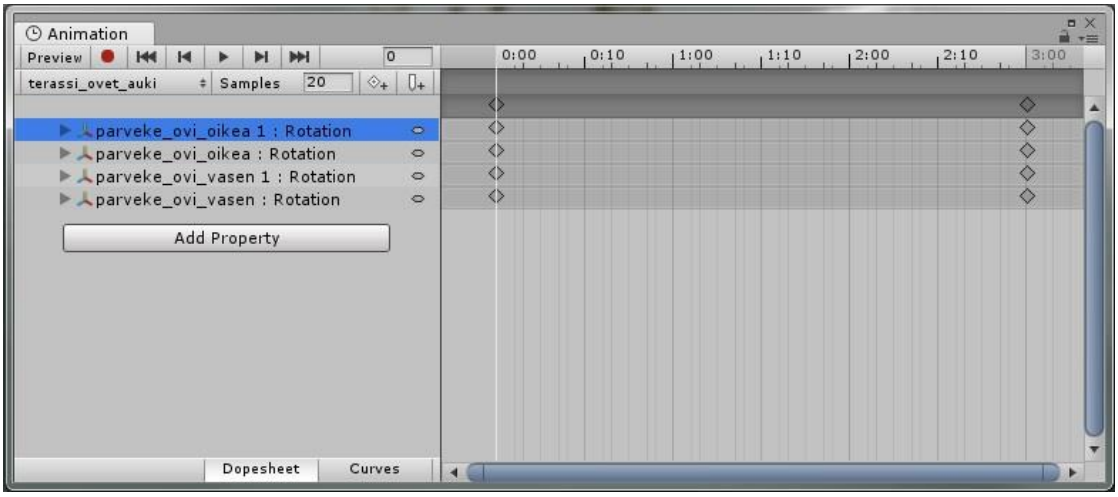
Unityn animaatioilla pystyy tekemään esimerkiksi oville toiminnallisuutta niin, että ne aukeavat. Animaatiota varten peliobjektille pitää määritellä animaatio ja animaatio-ohjain (animator controller). Valitaan ensin koko talomallin sisältävä objekti nimeltä ”v13_2015” ja klikataan ylävalikosta Window > Animation, mikä avaa animointia varten uuden ikkunan. Tehdään uusi animaatio valitsemalla Create ja annetaan sille kuvaava nimi, esimerkiksi ”etuovi_auki”.



Kuva 33. Animaatio luodaan Animation-ikkunassa

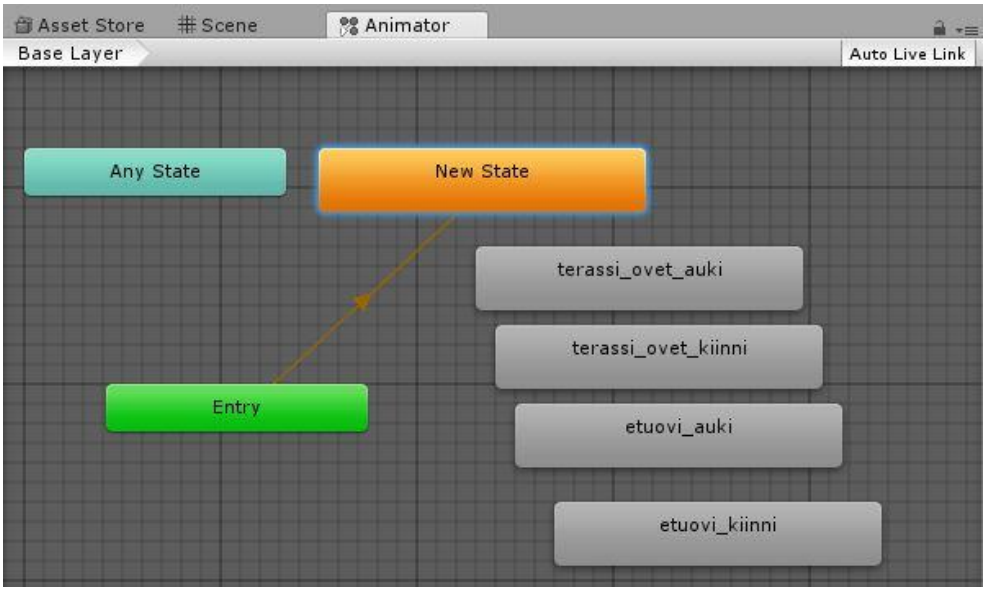
Valitaan scenestä se ovi, joka halutaan animoida, ja käynnistetään animaation nauhoittaminen klikkaamalla punaista ympyrää Animation-ikkunan yläpalkissa. Määritellään aikajanalle kaksi avainkehystä (keyframe) merkitsemään animaation aloitus- ja lopetuspisteitä. Jos nauhoitus on käynnissä, niin nämä pisteet luodaan automaattisesti aina kun objektin sijaintia tai asentoa muutetaan.

Oven avautuminen tapahtuu niin, että ovi on eri kulmassa animaation alussa ja lopussa. Ensimmäisen keyframen kohdalle siis ovi jätetään alkuperäiseen asentoon. Klikataan aikajanalla siitä kohdasta, johon lopetuspiste halutaan luoda, ja muutetaan oven kulmaa niin, että se on avautunut 90 asteen kulmaan alkuperäisestä kulmasta. Lopetetaan nauhoittaminen, minkä jälkeen animaatio ilmestyy assetteihin. Haluamme myöhemmin aktivoida tämän animaation näppäinkomennolla, joten käydään vielä animaation ominaisuuksista ottamassa raksi pois kohdasta ”loop”, jotta animaatio ei pyöri automaattisesti jatkuvalla syötöllä.



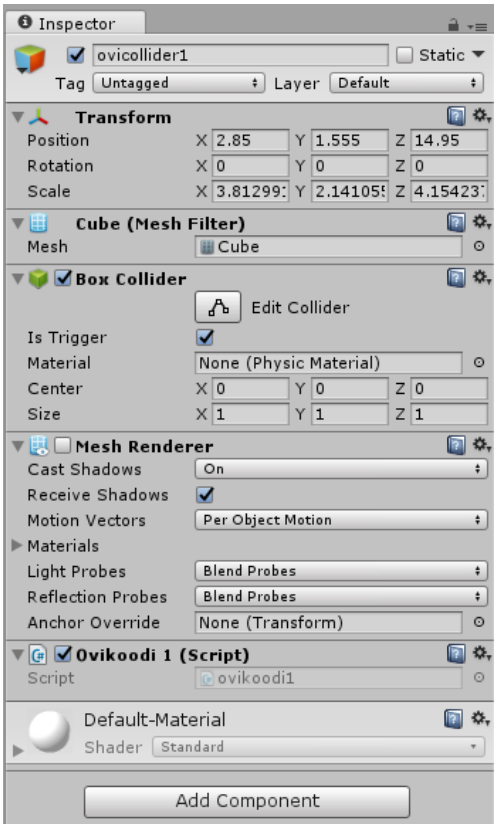
Kuva 34. Animointi tehdään määrittelemällä aikajanalle keyframeja

Animaation luonnin yhteydessä Unity luo automaattisesti myös Animator Controllerin, joka lisätään komponentiksi objektiin ”v13_2015”. Animaatioita kutsutaan myöhemmin koodissa tämän objektin kautta. Kaikki ovien animaatiot lisätään tähän samaan controlleriin. Controllerissa pitää olla oletustilana tyhjä ”New State”.



Kuva 35. Animator-tilakone sisältää kaikkien ovien animaatiot

Törmäystunnistimia voidaan käyttää ”liipaisimina”, jotka käynnistävät jonkin toiminnon, kun pelaaja liikkuu niiden määrittelemälle alueelle. Törmäystunnistimen asetuksista laitetaan tällöin rasti kohtaan ”Is Trigger”. Tällöin tunnistimeen ei voi törmätä, vaan se toimii ainoastaan liipaisimena. Tätä ominaisuutta voidaan käyttää oven avaamiseen niin, että ovi aukeaa kun pelaaja kävelee sen lähelle.



Kuva 36. Törmäystunnistimen asetuksista löytyy ominaisuus ”Is Trigger”

Törmäystunnistinta ei luoda suoraan itse ovelle, vaan ensin tehdään tyhjä peliobjekti, joka voi olla esimerkiksi kuutio, jonka osaksi törmäystunnistin liitetään. Lisäksi objektista tehdään näkymätön, koska sen ainoa tehtävä on toimia säiliönä törmäystunnistimelle, joten pelaajan ei tarvitse nähdä sitä. Objektia ei piirretä, kun sen ominaisuuksista otetaan ruksi pois kohdasta ”Mesh Renderer”. Kuutio sijoitetaan sitten scenessä siihen kohtaan mistä halutaan tehdä jonkin toiminnon käynnistävä alue. Ovi saadaan aukeamaan, kun pelaaja kävelee tähän alueelle. Kuutioon pitää liittää komponentiksi myös skripti, joka kertoo, mitä tapahtuu pelaajan astuessa alueelle. Tässä objektiin on liitetty ”ovikoodi1”-niminen skripti, joka on ohjelmoitu C#-ohjelmointikielellä. Koodissa määritellään, että oven avaava animaatio käynnistetään, kun pelaaja astuu törmäystunnistimen sisälle. Toinen animaatio, joka sulkee oven, käynnistetään silloin kun pelaaja poistuu samalta alueelta. Animaatioille on annettu niiden toimintaa kuvaavat nimet ”terassiovi1_auki” ja ”terassiovi1_kiinni”.

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

public class ovikoodi1 : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    void OnTriggerEnter(Collider other)
    {
        GameObject.Find("v13_2015").GetComponent<Animator>().Play("terassiovi1_auki");
    }

    void OnTriggerExit(Collider other)
    {
        GameObject.Find("v13_2015").GetComponent<Animator>().Play("terassiovi1_kiinni");
    }
}
```

Kuva 37. Animaation käynnistäminen törmäystunnistimien avulla

Animaatio voidaan myös käynnistää näppäinkomennolla, mikä pitää määritellä koodissa. Esimerkiksi ovi voidaan avata painamalla E-näppäin pohjaan. Alla olevassa koodiesimerkissä talon etuovi avautuu, kun E-näppäin painetaan pohjaan (GetKeyDown). Ovi sulkeutuu, kun E-näppäin päästetään vapaaksi

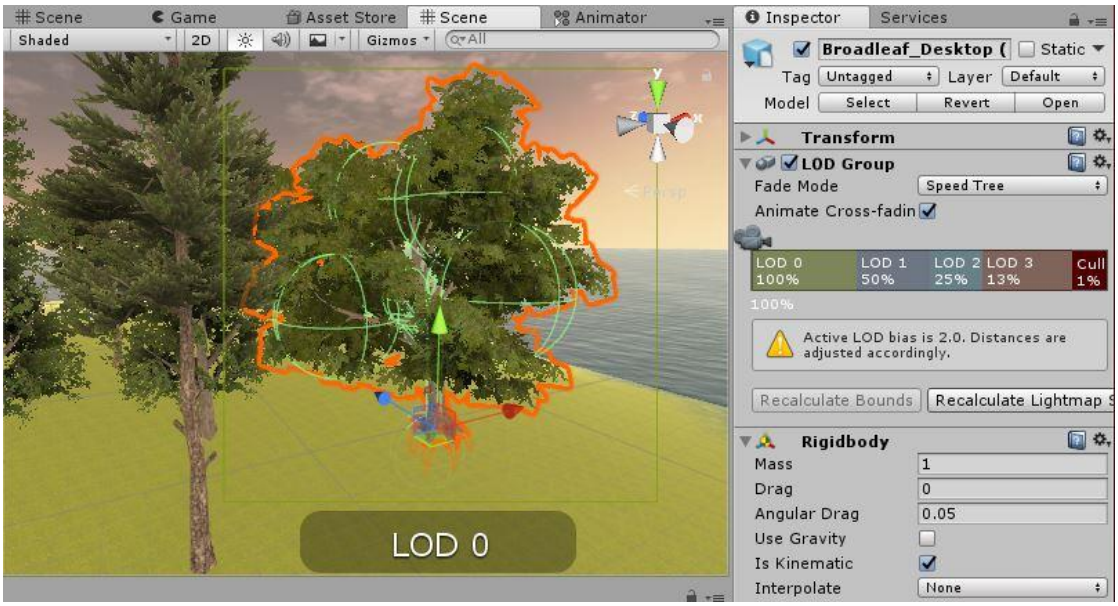
(GetKeyUp). Koodissa siis määritellään, mikä animaatio käynnistetään, kun tiettyä näppäintä painetaan. Koodissa määritellään ensin olio nimeltä ”talo”, minkä jälkeen kutsutaan Animator Controlleria, joka löytyy talo-oliosta. Animator sisältää kaikki ovien animaatiot, joita voidaan kutsua niiden nimillä. Tästä syystä animaatiot kannattaakin nimetä järkevästi, koska silloin niitä on helpompi kutsua koodissa.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class ovikoodi : MonoBehaviour {
6     private GameObject talo = null;
7
8     void Start () {
9         this.talo = GameObject.Find("v13_2015");
10    }
11
12    void Update () {
13        if (Input.GetKeyDown(KeyCode.P))
14        {
15            this.talo.GetComponent<Animator>().Play("terassi_ovet_auki");
16        }
17
18        if (Input.GetKeyUp(KeyCode.P))
19        {
20            this.talo.GetComponent<Animator>().Play("terassi_ovet_kiinni");
21        }
22        if (Input.GetKeyDown(KeyCode.E))
23        {
24            this.talo.GetComponent<Animator>().Play("etuovi_auki");
25        }
26
27        if (Input.GetKeyUp(KeyCode.E))
28        {
29            this.talo.GetComponent<Animator>().Play("etuovi_kiinni");
30        }
31    }
32 }
```

Kuva 38. Animaation käynnistäminen näppäinkomennolla

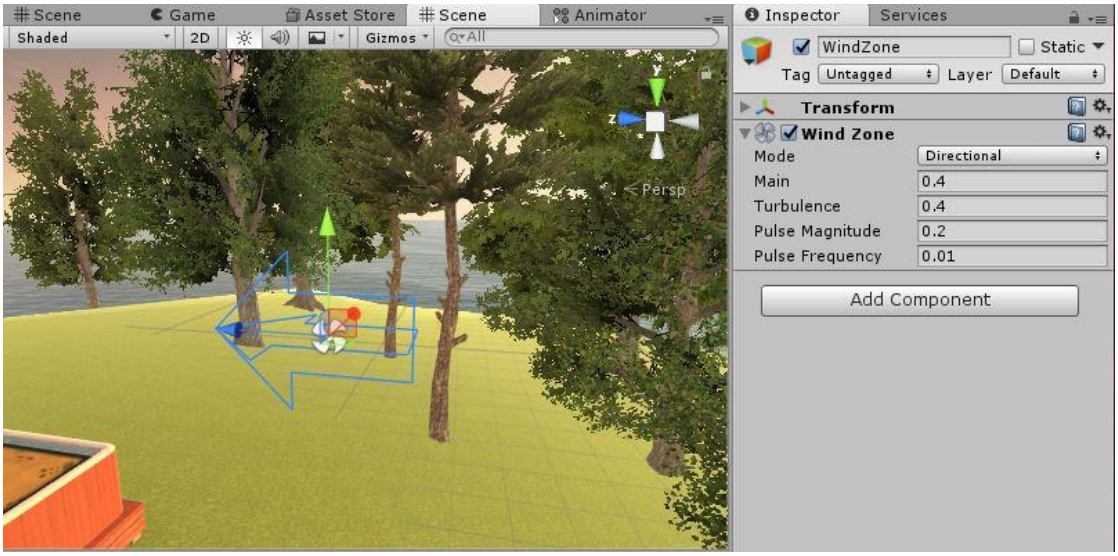
Unityn omilla työkaluilla on mahdollista luoda ympäristöön elementtejä, jotka tuovat maisemaan realistisuutta erilaisten maaston pinnan muotojen ja tekstuurien avulla. Maastoon voi lisätä kiviä ja puita sekä muuta kasvustoa. Puita voi tehdä yksityiskohtaisesti itse Unityn omalla editorilla. Nopeampaa on kuitenkin käyttää valmiita ”Speedtree”-prefabeja, jotka löytyvät asseteista environment-paketista. Vaihtoehtoina on lehti- ja kuusipuita, sekä palmuja. Suomalaiseen maisemaan sopivat parhaiten ensin mainitut. Puiden mittoja voi säätää erilaisiksi, mikä on realistista, koska todellisessakaan ympäristössä

puut eivät ole toistensa kopioita. Lisätään sceneen sopiva määrä speedtree-prefabeja ja muokataan niiden mittoja niin, että jokainen puu on hieman eri kokoinen.



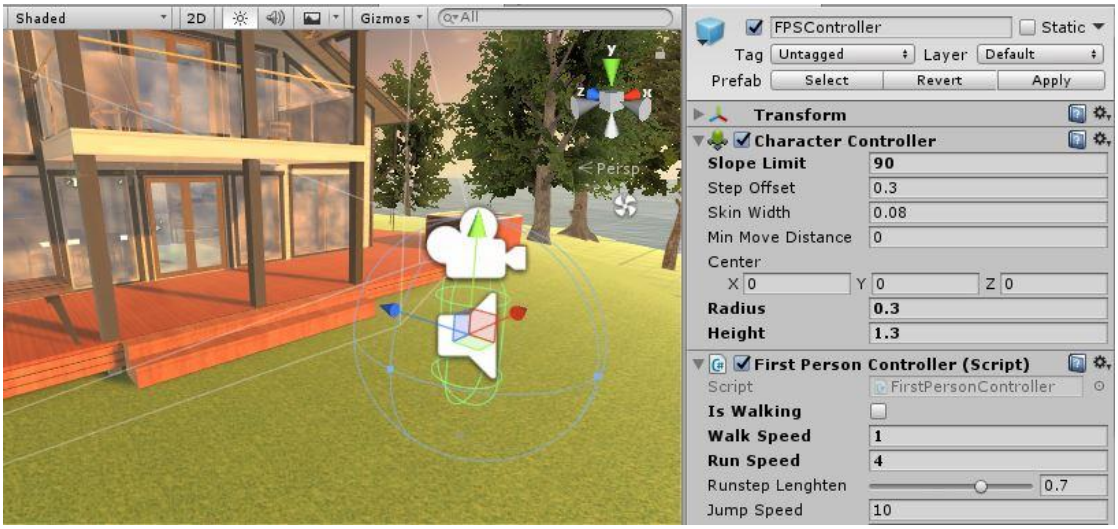
Kuva 39. Maisema näyttää realistisemmalta, kun siihen sijoitetaan puita

Sceneen voi myös lisätä ”windzone”-objektin, joka simuloi tuulen vaikutusta ympäristöön ja saa puut huojumaan tuulen mukana. Objektille voidaan määritellä muun muassa tuulen voimakkuus sen mukaan, minkälaista tuulta halutaan simuloida. Tähän talomaisemaan sopii rauhallinen tuuli, joka heiluttaa kevyesti puiden latvoja.



Kuva 40. Windzone-objekti saa puut huojumaan tuulessa

Pelimaailmassa liikkumista ei tarvitse ohjelmoida kokonaan itse, vaan tähän tarkoitukseen löytyy sopiva prefab. Pelihahmon liikuttamiseen tarvittavat prefabit löytyvät Characters-kansiosta, joka on jo aiemmin tuotu projektiin. Etsitään kansiosta FPSController-prefab ja pudotetaan se suoraan sceneen haluttuun kohtaan. Kun peli käynnistetään, se alkaa siitä kohdasta mihin FPSController on sijoitettu. Vaihtoehtona on myös RigidBodyFPSController, joka sisältää RigidBody-komponentin, eli se lisää pelihahmolle massan ja muita fyysisiä ominaisuuksia. Parempi vaihtoehto on kuitenkin FPSController ilman rigidbody-komponenttia, koska tällöin pelaajan liikkuminen on sulavampaa.



Kuva 41. FPS-controller sijoitetaan sceneen siihen kohti, mistä peli halutaan aloittaa

FPSControllerin ominaisuuksia voi vielä viilata. Liikkumisen nopeutta on laskettu, koska oletuksena liikkuminen on niin nopeaa, että se ei sovi rakennuksen tutkimiseen. Sekä kävelyn että juoksemisen nopeutta voi säätää objektin ominaisuuksista kohdista Walk Speed ja Run Speed. Kävelyn nopeudeksi on asetettu 1, ja juoksun nopeudeksi on asetettu 4. Myös pelaajan kokoa voi muuttaa määrittelemällä pelihahmon säteen ja korkeuden Unityn mittayksiköissä. Testaamisen tuloksena päädyin asettamaan säteeksi 0,3 ja korkeudeksi 1,3, koska näillä arvoilla pelaajan ja ympäristön mittasuhteet tuntuivat luonnollisimmilta ja saivat liikkumisen tuntumaan realistiselta. Kohdasta Head Bob kannattaa vielä määrittää kaikki arvot nollaan, mikä estää näkökentän heilumisen liikkuessa. Tämän ominaisuuden tarkoitus on simuloida pään heilumista liikkumisen aikana, eikä se sovi tähän tarkoitukseen. Viimeisenä asetuksena säädetään vielä Step Interval arvoon 1, mikä määrittää askelista kuuluvan äänen tiheyden. Arvolla 1 äänet sopivat parhaiten yhteen liikkumisen

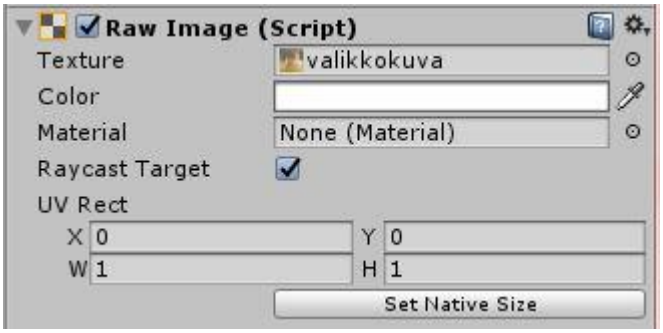
nopeuden kanssa. Lisäksi skin width kannattaa olla noin 10% hahmon säteestä, slope limit 90 astetta ja step offset 0,3.

Viimeisenä hienosäätönä lisätään taloon ja ympäristöön elonmerkkejä partikkeliefekteillä. Partikkeliefekteillä voidaan mallintaa esimerkiksi savua ja takassa palavaa tulta. Näihin tarkoituksiin sopivat Unityn Particle Systems -asseteista löytyvät prefabit FireMobile ja Smoke, jotka on sijoitettu takkaan, savupiippuun ja pihalla olevaan grilliin. Efektien ominaisuuksista kohtaan Particle System Destroyer on asetettu min ja max duration arvoihin 1000000. Tällä varmistetaan se, että efekti pyörii koko pelin pyörimisen ajan.

Kuva 42. FireMobile-partikkelifekti simuloi liekkejä

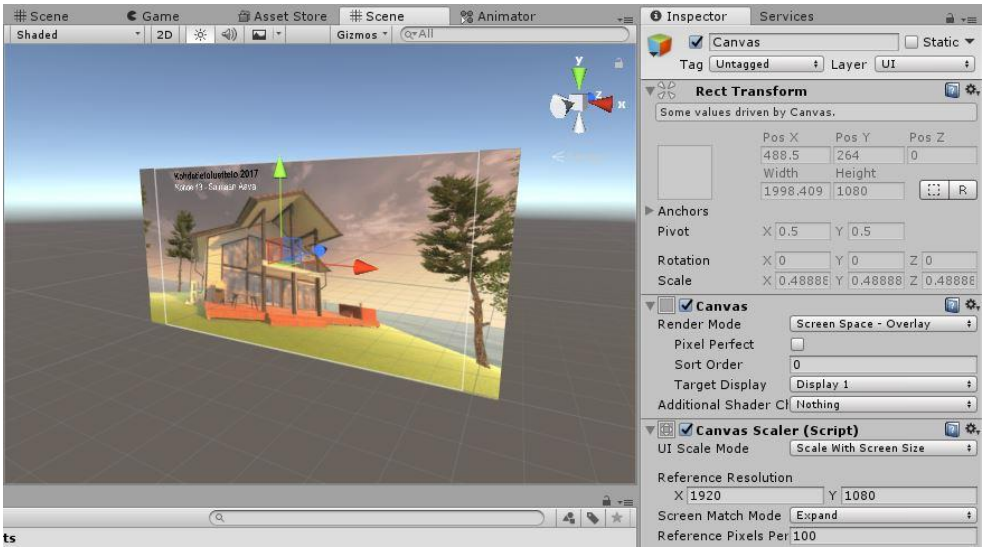
Peleissä on usein joku taukovalikko (pause screen, pause menu), mikä keskeyttää pelin suorittamisen, joten lisätään sellainen tähänkin talomallin esittelyyn. Taukovalikko on helppo toteuttaa, kun siihen käytetään Canvas-objektia, joka löytyy Unityn UI-objektien alta. UI-objektit (User Interface) ovat käyttöliittymän toteutusta varten tarkoitettuja objekteja. Canvas toimii pohjana koko käyttöliittymälle, eli kaikkien muiden käyttöliittymän elementtien tulee olla sijoitettuna canvas-objektin alle. Valikkoa varten täytyy ensin luoda uusi scene, koska taukovalikko ja rakennuksen esittely ovat eri sceneissä. Todellisuudessa siis vaihdetaan näkymää kahden eri scenen välillä, kun hypätään pelistä taukovalikkoon, ja tämä siirtymä toteutetaan pienellä koodilla.

Valitaan aluksi ylävalikosta File > New Scene, mikä luo uuden scenen, ja sen jälkeen tallennetaan uusi scene nimellä ”scene2”. Sceneen tehdään ensimmäisenä uusi Canvas, joka toimii pohjana kaikille muille käyttöliittymän elementeille. Valitaan ylävalikosta GameObjects > UI > Canvas, mikä luo uuden Canvas-objektin. Canvas kuvastaa näytöllä näkyvää kuvaa, joten siihen on helppo asetella elementtejä sen perusteella kuinka niiden halutaan näkyvän pelaajalle. Lisätään canvasin alle taukovalikon taustakuvaa varten RawImage-objekti. Taustakuvaa varten otetaan jokin edustava kuva rakennuksesta, mihin tarkoitukseen riittää vaikkapa Windowsin oma kuvakaappaustyökalu. Tallennetaan kuva jpg-muodossa nimellä ”valikkokuva” ja tuodaan se Unityn assetteihin. Valitaan RawImage objekti, jonka inspector-ikkunassa kohtaan ”Texture” lisätään tekstuuriksi tallentamamme kuvatiedosto.



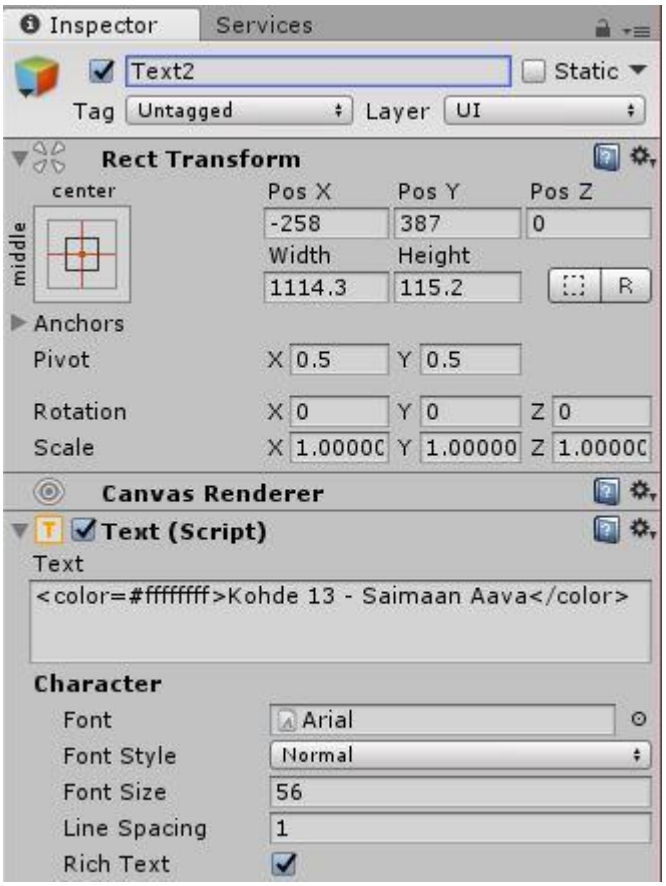
Kuva 43. Canvasin taustakuva

Kun taustakuva on paikoillaan, lisätään canvasiin myös kaksi text-objektia, jotka sisältävät valikossa näkyvän tekstin. Objektit asetellaan canvasin vasempaan yläkulmaan allekkain.



Kuva 44. Taukovalikon elementtejä

Tekstiä voi muotoilla samaan tapaan kuin HTML-tekstiä, eli ulkoasu määritellään tageilla, joiden sisään tekstiä kirjoitetaan. Tekstiä ja muotoiluja kirjoitetaan suoraan text-objektin inspector-ikkunassa olevaan tekstikenttään, mikä tekee tekstin tuottamisesta helppoa. Kirjoitetaan siis ensin ylempään text-objektiin taukovalikon otsikko, joka on ”Kohdetietoluettelo 2017”, ja määritellään tekstin väriksi musta ja korostuskeinoksi lihavointi. Alempaan text-objektiin kirjoitetaan kohteen tarkemmat tiedot, eli ”Kohde 13 – Saimaan Aava”. Tekstin väriksi asetetaan valkoinen, mikä määritellään tageilla. Kirjoitetaan siis tekstin ympärille tagi <color> ja määritellään sen sisällä valkoisen värin HEX-arvo, joka on #ffffff.



Kuva 45. Tekstin muotoilua text-objektissa

Taukovalikko on tietenkin hyödytön, jos pelaajalla ei ole siihen pääsyä. Tehdään siis pieni skripti, missä määritellään näppäinkomennot joiden avulla pelaaja voi vaihtaa näkymää pelin ja taukovalikon välillä. Scene1:ssä skripti lisätään komponenttina FPS-controlleriin, ja scene2:ssa tehdään tyhjä objekti nimeltä ”koodiolio” mihin skripti liitetään. Molempiin objekteihin pitää siis lisätä uusi komponentti ”New Script”, mikä tehdään aivan samalla tavalla kuin mui-

denkin komponenttien lisääminen. Eli FPS-controllerin inspector-ikkunasta klikataan ”Add Component”, annetaan skriptille nimi ”poistu” ja valitaan ohjelmointikieleksi C#. Scene2:ssa sama tehdään koodiolio-objektin kohdalla, ja skriptille annetaan nimeksi ”poistu2”. Käyttämällä SceneManager-luokkaa voidaan koodissa suorittaa sceneihin liittyviä toimintoja. LoadScene saa parametrikseen sen scenen nimen, mikä halutaan ladata. Scene1:ssä koodi toimii niin, että painamalla L-näppäintä siirrytään scene2:een, ja vastaavasti scene2:ssa samalla näppäinkomennolla siirrytään scene1:een. Lisätään samaan skriptiin vielä mahdollisuus poistua koko pelistä komennolla ”Application.Quit()”, ja määritellään, että tähän tarkoitukseen käytetään perinteisesti esc-näppäintä.

poistu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class poistu : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        if (Input.GetKeyUp(KeyCode.Escape))
        {
            Application.Quit();
        }

        if (Input.GetKeyUp(KeyCode.L))
        {
            SceneManager.LoadScene("scene2");
        }

    }
}
```

poistu2.cs

No selection

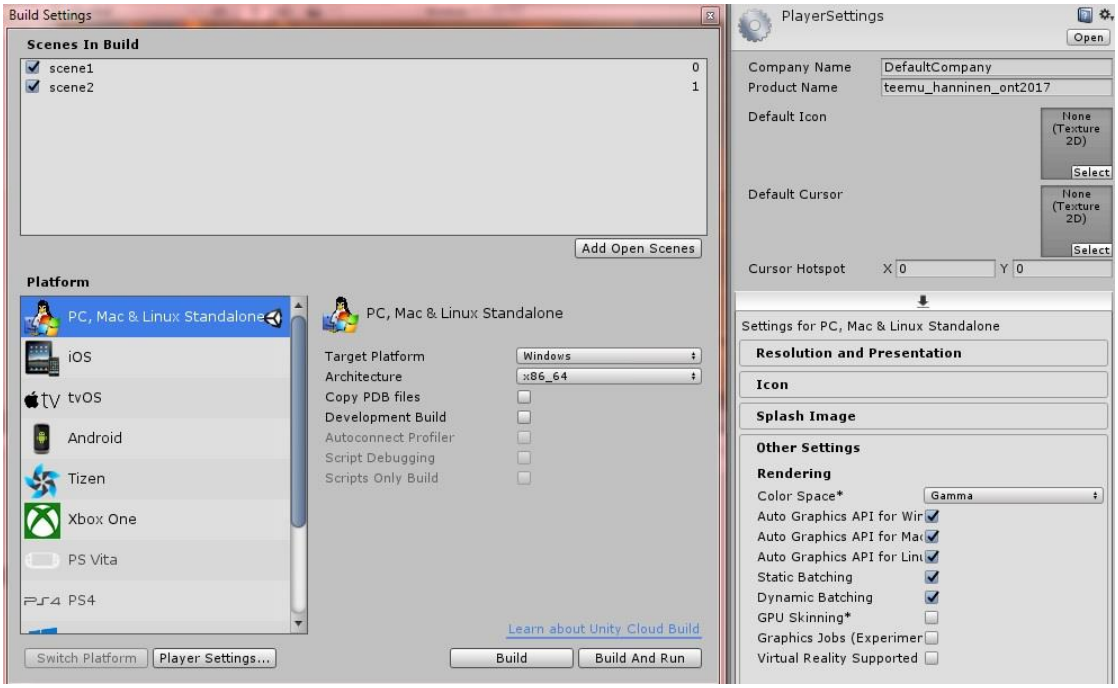
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class poistu2 : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        if (Input.GetKeyUp(KeyCode.Escape))
        {
            Application.Quit();
        }

        if (Input.GetKeyUp(KeyCode.L))
        {
            SceneManager.LoadScene("scene1");
        }

    }
}
```

Kuva 46. Skriptissä määritellään, millä näppäinkomennoilla siirrytään eri sceneen

Valmiissa taukovalikossa talo on edustettuna esittelyyn soveltuvasta kuvakulmasta. Tällaista toimintoa voisi käyttää esimerkiksi asuntomessuilla. Messujen vierailijat saavat halutessaan kierrellä pelin maailmassa tutkimassa taloa ja sen ympäristöä. Taukovalikko voidaan jättää näkyviin silloin kun halutaan näytölle pyörimään asiakkaita houkutteleva mainoskuva.

Kuva 47. Taukovalikossa on tekstiä ja taustakuva

Peli pitää tietenkin vielä kääntää valmiiksi ohjelmaksi, jos se halutaan julkaista. Tässä esimerkissä peli käännetään Windows-, Mac- ja Linux-yhteensopivaksi. Kääntäminen onnistuu valitsemalla ylävalikosta File > Build settings, minkä jälkeen kohdassa ”Scenes In Build” on mahdollista valita scenet, jotka halutaan ottaa mukaan valmiiseen peliin. Valitaan molemmat scenet, eli scene1 ja scene2. Lisäksi valitaan se alusta, jolle peli halutaan julkaista. Kannattaa myös tarkistaa PlayerSettings ja varmistaa, että Virtual Reality ei ole tuettuna, jos peliä ei ole kehitetty VR-tukea ajatellen, koska tämä saattaa aiheuttaa ongelmia pelaajan liikkumisessa. Kun kaikki asetukset ovat kohdillaan, klikataan joko Build tai Build And Run. Nyt Unity kääntää pelin julkaistavaksi paketiksi ja luo exe-tiedoston projektin nimellä. Pelin voi käynnistää tästä exe-tiedostosta, ja jos valittiin Build And Run, niin peli käynnistyy heti kun kääntäminen on valmis.



Kuva 48. Projektin kääntäminen ja julkaisu valmiiksi peliksi

Lopuksi voidaan vertailla, kuinka rakennusmallin ulkoasu on muuttunut matkan varrella, kun se on tuotu alkuperäisestä mallinnusohjelmasta pelimoottoriin. SketchUpissa valmis malli on valaistuksen osalta yksinkertaisempi eikä näytä realistiselta (kuva 49). Suurimman eron huomaa maanpinnan tekstuureissa ja varjojen laadussa. Lisäksi skybox puuttuu, joten maisema ei luo realistista vaikutelmaa.

Kuva 49. Alkuperäinen malli SketchUpissa

Unityyn tuotuna varjot muuttuvat pehmeämmiksi ja maanpinta näyttää paremmalta. Skybox saa taivaan näyttämään realistisemmalta kuin Sketchupissa.

Talo näyttää kuitenkin vielä hyvin staattiselta, koska siinä ei näy elon merkkejä. Maisema näyttää myös varsin autiolta. (Kuva 50.)

Kuva 50. Malli on tuotu Unityyn

Valmiissa versiossa skybox ja valaistus toimivat hyvin yhdessä, koska valo näyttää saapuvan taivaalla sijaitsevasta valonlähteestä ja luo realistisia varjoja (kuva 51). Partikkeliefektien avulla talomalliin on saatu myös elävä tunnelma lisäämällä savupiipusta nousevaa savua. Puut ja vedenpinta tuovat maisemaan realismia.

Kuva 51. Näkymä valmiissa pelissä

Kuvista voi kuitenkin havainnoida vain rakennuksen visuaaliset muutokset. Kuvista ei voi päätellä, millaisen vaikutelman ympäristöstä saa, kun siinä pääsee liikkumaan vapaasti.

5 PÄÄTÄNTÖ

Tärkein realismiin vaikuttava tekijä näyttäisi olevan valaistus, joka muuttaa radikaalisti 3D-mallin visuaalisuutta. Tarvitaan kuitenkin laskentatehon kehittymistä ennen kuin reaaliaikaisesti voidaan valaista korkeatasoisia rakennusmalleja. Toisaalta mallien monimutkaista geometriaa on mahdollista korvata tekstuureilla, mikä alentaa pelimoottorissa vaadittua laskentatehoa ja tuo laajemmalle yleisölle mahdollisuuden pyörittää rakennusesittelyjä omalla koneella. Lisäksi tekstuurien käyttäminen geometrian esittämiseen nopeuttaa mallinnusprosessia. Esimerkiksi talon ulkoasussa seinien hirsirakennetta jäljitellään tekstuureilla, mikä on nopeampi tapa kuin mallintaa jokainen hirsi yksitellen.

Ilmaiset ohjelmat riittävät luonnosteluun ja niiden avulla saa hyvän kuvan tilasta ja mittasuhteista. SketchUpin ilmaisesta versiosta puuttuu kuitenkin paljon sellaisia ominaisuuksia, joiden avulla voitaisiin toteuttaa rakennuksen tarkkoja tietomalleja. Nämä ominaisuudet löytyvät SketchUpin maksullisesta versiosta. Maksullisen version etuna on myös tuettujen tiedostomuotojen suurempi määrä ja se, että tiedostoja voi tallentaa eri muodoissa. Näin vältytään siltä, että tiedostoja pitäisi kierrättää jonkin toisen ohjelman kautta, jos tiedostomuotoa halutaan muuttaa. Unityn ilmainen versio sen sijaan on riittävä, eikä maksullinen versio sisällä sellaisia tarpeellisia ominaisuuksia joita ilmaisesta versiosta ei löytyisi. Varsinkin näyttävän valaistuksen rakentamiseen löytyy Unityn ominaisuuksista paljon valinnanvaraa. Muita ohjelmia ei käytetty riittävästi, että niihin pystyttäisiin tekemään laajaa vertailua.

Pelimoottorien helppokäyttöisyyden vuoksi varsinaisten pelimäisten elementtien tekeminen vie huomattavasti vähemmän aikaa kuin 3D-mallin tuottaminen. Ohjelmoinnilla pelillistämiseen saadaan lisättyä rikkaampia ominaisuuksia, kuten animaatioiden käynnistäminen käyttäjän syötteen perusteella. Nämä ominaisuudet voi kuitenkin tarvittaessa jättää pois, joten rakennusmallin pelillistäminen onnistuu kirjoittamatta riviäkään koodia. Yksinkertaisimmillaan

ei tarvita muuta kuin toimiva 3D-malli, joka tuodaan pelimoottoriin, ja sille luodaan automaattisesti törmäystunnistimet. Tämän jälkeen tarvitsee vain lisätä Unityn valmis FPSController-prefab, minkä jälkeen rakennusesittely on valmis. Valaistuksen määrittelykään ei vaadi koodin kirjoittamista. Ongelmia kuitenkin aiheutuu siitä, että ei ole yhtenäisiä tiedostomuotoja ja ohjelmistoja. Eri vaihtoehdot sopivat eri tarkoituksiin. Huonosti toteutetut mallit aiheuttavat virheitä sekä tekstuureissa että geometriassa, jos 3D-mallin tuotantovaiheessa mallintaminen on toteutettu virheellisesti. Tästä syystä myös 3D-warehousen omatekoisia malleja kannattaa käyttää varauksella, koska kolmannen osapuolen tekeleet on usein huonosti mallinnettu: ne sisältävät huonosti nimettyjä elementtejä jotka aiheuttavat ongelmia ainakin Unrealissa, ja geometriassa on usein tehty perustavanlaatuisia virheitä jättämällä malliin toisiaan leikkaavia pintoja. Esimerkiksi talon yläkerrassa sijaitseva sohva sisältää virheellisiä pintoja. Sohva on kolmannen osapuolen mallintama ja se on saatavilla 3D-warehousesta. Realismin kannalta on myös järkevämpää luoda rakennuksen ympäröivä maasto Unityn omilla työkaluilla, ja luoda mallinnusohjelmalla pelkkä rakennus. SketchUpin omilla työkaluilla tehty maasto ei ole yhtä realistinen tai näyttävä kuin mihin Unityn vastaavilla työkaluilla pystytään.

Jatkokehitystä ajatellen mukaan voisi tuoda virtuaalitodellisuuden (VR = Virtual Reality), minkä avulla subjektiivinen kokemus korostuisi entisestään. Tällaisen ominaisuuden toteuttaminen onnistuisi helposti Unityn työkaluilla. Kenties tulevaisuudessa rakennusteollisuuden koulutuksessa voidaan hyödyntää virtuaalitodellisuutta. Esimerkiksi rakentaja pystyisi virtuaalitodellisuudessa harjoittelemaan omasta näkökulmasta talon rakentamista, mikä olisi kouluttamisessa tehokas apuväline. Virtuaaliteknologia (VR-teknologia) on kuitenkin edelleen kallista, ja esimerkiksi HTC Vive -virtuaalilasit maksavat tällä hetkellä Suomessa noin 800 euroa. VR-teknologia myös vaatii enemmän laskentatehoa kuin tavanomaisesti tietokoneen näytöllä esitetty grafiikka, joten tarvittava laitteistokin on kalliimpaa.

LÄHTEET

AN-Cadsolutions. 2015. SketchUp Pro 3D-mallinnusohjelma. WWW-dokumentti. Saatavissa: https://www.an-cadsolutions.fi/ohjelmistot/sketchup-pro-3d-mallinnusohjelma/ [viitattu 16.10.2017].

Animator Controller. 2017. Unity Technologies. WWW-dokumentti. Saatavissa: https://docs.unity3d.com/Manual/class-AnimatorController.html [viitattu 1.12.2017].

Asuntomessut Mikkelissä. 2017. Kohdetietoluettelo. PDF-dokumentti. Päivitetty 1.4.2017. Saatavissa: http://asuntomessut.fi/wpcontent/uploads/2017/02/SE3741Kohdetietoluettelo_Mikkeli_Asuntomessut2017.pdf/ [viitattu 27.6.2017].

Baker, M. 2016. How Do Game Engines Work? Interesting Engineering. WWW-dokumentti. Muokattu 2.11.2016. Saatavissa: http://interestingengineering.com/how-game-engines-work/ [viitattu 29.9.2017].

Berger, W. 1994. Future Quest: Go Where No Real Estate Company Has Gone Before. *Real Estate Today*. Tammikuu 1994 (15-18).

Boeykens, S. 2011. Using 3D Design software, BIM and game engines for architectural historical reconstruction. Leuvenin yliopisto. PDF-dokumentti. Muokattu tammikuu 2011. Saatavissa: https://www.researchgate.net/ [viitattu 16.10.2017].

Bryde, D., Broquetas, M., Volm, J.M. 2012. The project benefits of Building Information Modelling (BIM). PDF-dokumentti. Muokattu 22.11.2012. Saatavissa: http://www.sciencedirect.com/ [viitattu 16.10.2017].

Chakravorty, D. 2017. 8 Most Common 3D File Formats Simply Explained. All3DP. WWW-dokumentti. Päivitetty 19.7.2017. Saatavissa: https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/ [viitattu 13.10.2017].

Chopine, A. 2011. 3D Art Essentials: The Fundamentals of 3D Modeling, Texturing, and Animation. Burlington, MA: Focal Press. Taylor&Francis Group.

Colliders. 2017. Unity Technologies. WWW-dokumentti. Saatavissa: https://docs.unity3d.com/Manual/CollidersOverview.html/ [viitattu 18.8.2017].

Company Facts. 2017. Unity Technologies. WWW-dokumentti. Saatavissa: https://unity3d.com/public-relations/ [viitattu 12.10.2017].

Construction in the City s.a. The Leadenhall Building Development Company. PDF-dokumentti. Saatavissa: https://www.theleadenhallbuilding.com/wp-content/uploads/2014/08/Exhibition_Totem_4.pdf/ [viitattu 17.10.2017].

Find a 3D model of anything. 2017. Trimble. WWW-dokumentti. Saatavissa: http://www.sketchup.com/#find-3d-models-anchor/ [viitattu 27.6.2017].

Fisher, J. Lerg, M. Louziotis, D. Virtual Technology: The Future Is Now For the Commercial Real Estate Industry. Real Estate Issues. WWW-dokumentti. Saatavissa: https://www.cre.org/real-estate-issues/virtual-technology-future-now-commercial-real-estate-industry/ [viitattu 12.10.2017].

Get good fast. 2017. Trimble. WWW-dokumentti. Saatavissa: http://www.sketchup.com/#get-good-fast/ [viitattu 27.6.2017].

Hannus, E. 2017. Tki-asiantuntija. Sähköpostikeskustelu. 6.11.2017. Kaakkois-Suomen ammattikorkeakoulu.

Haonperä, J. 2013. Mitä on pelillistäminen? Blogi. Muokattu 10.7.2013. Saatavissa: http://www.cloudriven.fi/blogi/mita-on-pelillistaminen/ [viitattu 27.6.2017].

Importing Objects from SketchUp. 2017. Unity Technologies. WWW-dokumentti. Saatavissa: https://docs.unity3d.com/Manual/HOWTO-ImportObjectSketchUp.html/ [viitattu 16.10.2017].

IndiaCADworks. 2013. Real Estate Agency's Architectural Modeling Formula. Blogi. Muokattu 5.8.2013. Saatavissa: https://www.indiacadworks.com/blog/real-estate-agencys-architectural-modeling-formula/ [viitattu 10.9.2017].

Johns, R. & Lowe, R. 2005. Unreal Editor as a Virtual Design Instrument in Landscape Architecture Studio. PDF-dokumentti. Saatavissa: http://citeseerx.ist.psu.edu/ [viitattu 18.10.2017].

Kosmadoudi, Z. 2012. Engineering design using game-enhanced CAD: The potential to augment the user experience with game elements. PDF-dokumentti. Muokattu 4.8.2012. Saatavissa: http://www.sciencedirect.com/ [viitattu 12.9.2017].

Laine, P. 2017. Microsoft rekisteröi DirectPhysics-tavaramerkin. Io-tech. WWW-dokumentti. Muokattu 30.4.2017. Saatavissa: https://www.io-tech.fi/uutinen/microsoft-rekisteroi-directphysics-tavaramerkin/ [viitattu 15.8.2017].

Lappalainen, P. 2016. Kaupunkimallinnuksen ohjekirja. Buildingsmart Finland. WWW-dokumentti. Muokattu 16.8.2016. Saatavissa: https://buildingsmart.fi/kaupunki/kaupunkimallinnuksen-ohjekirja/ [viitattu 20.10.2017].

Pelosi, A.W. 2017. Obstacles of utilizing real-time 3D visualization in architectural representations and documentation. PDF-dokumentti. Saatavissa: https://www.researchgate.net/ [viitattu 20.10.2017].

Physics Engine. 2017. Computer Hope. WWW-dokumentti. Muokattu 26.4.2017. Saatavissa: https://www.computerhope.com/jargon/p/physics-engine.htm/ [viitattu 29.7.2017].

Physics in Unity 5.0. 2017. Unity Technologies. WWW-dokumentti. Saatavissa: https://docs.unity3d.com/Manual/UpgradeGuide5-Physics.html/ [viitattu 18.8.2017].

Pluralsight. 2014. Gamification: The Future Interface of Architectural Design. Blogi. Muokattu 13.8.2014. Saatavissa: https://www.pluralsight.com/blog/film-games/siggraph-2014-news-autodesk-looks-gamifying-design/ [viitattu 16.10.2017].

Portelli, G. 2015. World Coordinate Systems in 3ds Max, Unity and Unreal Engine. WWW-dokumentti. Päivitetty 2.8.2015. Saatavissa: http://www.aclockworkberry.com/world-coordinate-systems-in-3ds-max-unity-and-unreal-engine/ [viitattu 20.10.2017].

Prunier, J. 2009. Introduction to Polygon Meshes. Scratchapixel. WWW-dokumentti. Saatavissa: http://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-polygon-mesh/ [viitattu 15.8.2017].

Puhakka, A. 2008. 3D-grafiikka. 1. painos. Helsinki: Talentum Media.

Savela, S. 2017. Mahdoton läpäistä yksin – VR Track kouluttaa työntekijöitä pakopelin avulla. Yle. WWW-dokumentti. Muokattu 22.6.2017. Saatavissa: https://yle.fi/uutiset/3-9685939/ [viitattu 25.6.2017].

Selin, J. 2017. Yliopettaja. Henkilökohtainen tiedonanto. 30.10.2017. Kaakkois-Suomen ammattikorkeakoulu.

Sinicki, A. 2016. An introduction to Unity3D for easy Android game development. Android Authority. WWW-dokumentti. Muokattu 8.1.2016. Saatavissa: http://www.androidauthority.com/an-introduction-to-unity3d-666066/ [viitattu 1.10.2017].

SketchUp School. 2017. Getting Started with SketchUp. WWW-dokumentti. Saatavissa: https://www.sketchupschool.com/sketchup/ [viitattu 3.11.2017].

Slick, J. 2016a. 3D Model Components - Vertices, Edges, Polygons & More. Verkkoartikkeli. Päivitetty 21.10.2016. Saatavissa: https://www.lifewire.com/3d-model-components-1952/ [viitattu 25.7.2017].

Slick, J. 2016b. 7 Common Modeling Techniques for Film and Games. Lifewire. WWW-dokumentti. Päivitetty 20.10.2016. Saatavissa: https://www.lifewire.com/common-modeling-techniques-for-film-1953/ [viitattu 12.10.2017].

Slick, J. 2016c. Polygonal 3D Modeling - Common Box and Edge Modeling Workflows. Lifewire. WWW-dokumentti. Päivitetty 9.10.2016. Saatavissa: https://www.lifewire.com/polygonal-3d-modeling-2139/ [viitattu 12.10.2017].

Slick, J. 2016d. List of 3D Software - Full 3D Suites. WWW-dokumentti. Päivitetty 20.10.2016. Saatavissa https://www.lifewire.com/full-3d-suites-2006/ [viitattu 3.11.2017].

Stevenson, K. 2015. What is a 3D Model, Anyway? Fabbaloo. WWW-dokumentti. Päivitetty 8.4.2015. Saatavissa: http://www.fabbaloo.com/blog/2015/4/5/what-is-a-3d-model-anyway/ [viitattu 12.10.2017].

Thorn, A. 2011. Game Engine Design and Implementation. Sudbury: Jones & Bartlett Learning.

Unrealer Blog. 2017. 3D:n lyhyt oppimäärä rakennus- ja kiinteistöalalle. Blogi. Saatavissa: https://blog.unrealer.com/2016/04/19/3d-lyhyt-oppimaara-rakennus-ja-kiinteistoalalle/ [viitattu 21.10.2017].

ValueCoders. 2017. Unreal Engine vs Unity 3D Games Development: What to Choose? WWW-dokumentti. Päivitetty 17.9.2017. Saatavissa: https://www.valuecoders.com/blog/technology-and-apps/unreal-engine-vs-unity-3d-games-development/ [viitattu 17.10.2017].

Zhao, X. 2017. A scientometric review of global BIM research: Analysis and visualization. PDF-dokumentti. Saatavissa: http://www.sciencedirect.com/ [viitattu 16.10.2017].

Zichermann, G. & Cunningham, C. 2011. Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps. 1. Painos. Kalifornia: O'reilly media.