UNITY 3D:N SOVELTUVUUS RAKENNUSSUUNNITTELUUN

Pelimoottorin ominaisuuksien hyödyntäminen rakennussuunnittelun apuvälineenä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Visamäki, Tradenomi

Syksy, 2018

Marika Nokso Thomsen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittely Tradenomi
Visamäki

Tekijä Thomsen Marika **Vuosi** 2018

Työn nimi Unity 3D:n soveltuvuus rakennussuunnitteluun

Työn ohjaaja Lauri Salminen

TIIVISTELMÄ

Tämän tutkimuksen tarkoitus on kartoittaa mahdollisuuksia ulkopuolisen sovelluksen käyttämiseen rakennussuunnittelun apuvälineenä. Vertex suunnitteluohjelmalla suunniteltu rakennus siirretään pelimoottoria hyödyntävälle Unity 3D sovelluskehitysalustalle ja selvitetään mitä rakennukselle on mahdollista sovelluksessa tehdä. Sitten IFC-tiedonsiirtoa hyväksikäyttäen koitetaan siirtää muokattua tietoa takaisin Vertexiin. Tässä työssä selvitetään mitä ja miten tietoa voi siirtää Vertexin ja Unityn välillä ja saako siitä lisäarvoa rakennussuunnitteluun. Työssä keskitytään myös selvittämään, mitä tietoa IFC-tiedoston avulla voi siirtää ja onko kyseinen tiedostoon mahdollista tehdä muutoksia manuaalisesti ja niin, että muutokset voi avata suunnitteluohjelmalla.

Työhön liittyneen kehitysprojektin aikana kävin läpi erilaisia tapoja hyödyntää Unityä rakennussuunnittelussa. Vertexillä suunnitellut rakennukset ja rakennuksen osat avattiin Unityllä ja sitten testasin, mitä kaikkea rakennukselle voi Unity sovelluksessa tehdä. Kokonaisen rakennuksen muokkaaminen Unityssä oli hankalaa, joten työssä testasin myös sovellusta, jossa asiakas voi suunnitella rakennusta kokoamalla sen rakennuselementeistä. IFC-tiedostolle tehdyssä vertailututkimuksessa selvitin, muutosten tekemistä manuaalisesti suoraan IFC-tiedostoon. Se osoittautui haastavaksi, sillä tiedosto on laaja ja vaikeasti luettava. Yhdestä muutoksesta saattaa muodostua kymmeniä rivejä koodia ja rakennuksessuunnittelussa käydään läpi satoja eri osia ja niihin kohdistuvia muutoksia.

Avainsanat rakennussuunnittelu, pelimoottori, Unity 3D, rakennuksen tietomalli (BIM), IFC-tiedonsiirto, Vertex BD

Sivut 28 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's Degree Programme
Business Information Technology
Visamäki, Hämeenlinna

Author Thomsen Marika **Year** 2018

Subject Unity 3D in handling building information models

Supervisors Lauri Salminen

ABSTRACT

The purpose of this research was to find the possibilities for using external software as a tool for designing buildings. A building designed in Vertex BD architect program is moved into the Unity 3D software designing program that utilizes the game engine. Tests were made to improve the building in Unity, and to change it as if the customer would like to have some changes made into his building design. Finally, the IFC-file is used in attempt to return the changed information back to Vertex. This report clarifies how game engine can be used to improve work in building industry and it also gives insight into what is an IFC-file and how the information can be moved by making changes to the file manually.

During the testing part of this report examined different options of how to use Unity 3D in building designing. Buildings and building parts designed in Vertex were moved to Unity 3D, and then tested how they work in Unity. Altering a whole building proved out to be difficult, so a test-version of an application where customer could design his own building by using building parts was also created. To test the IFC-file small comparative examination was made to see how many changes must be made manually into the file to be able to create one change in the building. It was found out that there are dozens of changes in the IFC-file so changing IFC-file manually is difficult and time consuming.

Keywords Game engine, Unity 3D, Building Information Model (BIM), Industry Foundation Classes (IFC), Vertex BD

Pages 28 pages

SISÄLLYS

1 JOHDANTO

Tämän työn aiheena on kartoittaa mahdollisuuksia Unity 3D sovelluskehitysalustan hyödyntämiseen rakennussuunnittelussa. Työn käytännön osuudessa tulen tekemään kolmivaiheisen testausprojektin, jossa pyrin siirtämään rakennuksen Vertexistä Unityyn, muokkaamaan rakennusta Unityssä ja palauttamaan muokatun rakennuksen takaisin Vertexiin. Tähän työhön kirjaan testausprojektin aikana havaitsemiani ongelmia ja hyötyjä, jota Unityn käyttäminen rakennussuunnittelussa voi aiheuttaa. Tässä työssä on tarkoitus kartoittaa mahdollisuutta, jossa asiakas pystyisi suunnittelemaan ja muokkaamaan omaa rakennustaan itsenäisesti täysin arkkitehtiohjelman ulkopuolelle rakennetussa sovelluksessa. Tämä sovellus käyttää hyödykseen Unity 3D pelimoottoria, joka mahdollistaa käyttäjän liikkumisen rakennuksessa ja mahdollisten muutoksien tekemisen rakennukseen. Tässä työssä myös testataan, pystyykö Unityssä muokatun rakennuksen myös palauttamaan onnistuneesti takaisin arkkitehtiohjelmaan. Tällöin luonnossuunnittelijan olisi mahdollista saada asiakkaalta jo valmiiksi muokattu luonnospohja suoraan suunnitteluohjelmaan, jossa hänen tehtäväkseen jäisi lähinnä rakennusteknisten seikkojen varmistaminen ja muu hienosäätö.

Testausprojektini aikana pyrin löytämään vastaukset seuraaviin kysymyksiin:

1. Pystyykö Unity 3D:tä hyödyntämään rakennussuunnittelussa?
2. Miten Vertexillä suunnittelun rakennuksen saa onnistuneesti siirrettyä Unityyn?
3. Pystyykö siirrettyä rakennusta muokkaamaan Unityssä?
4. Pystyykö asiakas suunnittelemaan oman rakennuksen Unityssä?
5. Saako muokatun rakennuksen siirrettyä takaisin Vertexiin?

2 RAKENNUSSUUNNITTELUN TYÖVÄLINEET

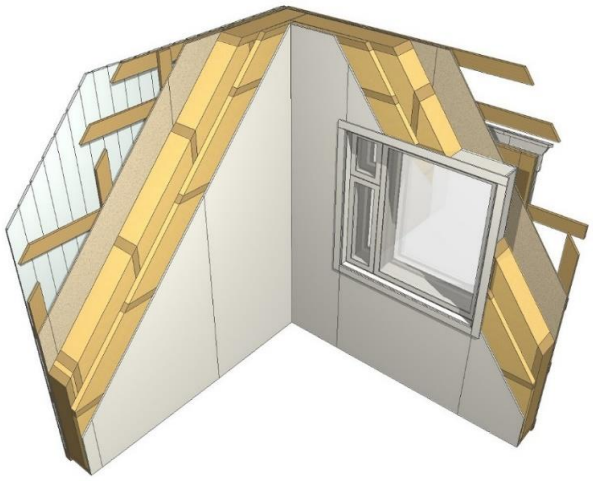
Rakennussuunnittelua tehdään nykyään lähes poikkeuksetta 3D-suunnittelun mahdollistavilla työkaluilla. Enää ei riitä, että asiakkaalle työstetään rakennuksesta pohjakuva ja läpileikkaukset. Sen sijaan rakennuksesta valmistetaan luonnoskuvat, lupakuvat ja rakennekuvat. Joka pieninkin ruuvi ja palkki pyritään mallintamaan tarkasti, jotta rakennuksesta voidaan antaa mahdollisimman tarkkoja tietoja eteenpäin niin tarjouslaskijalle, ostajalle kuin tuotantoonkin. Asiakkaalle halutaan tarjota kolmiulotteisia luonnoskuvia ja myös interaktiivisia kokemuksia suunnitellusta rakennuksesta. Yleensä rakennussuunnittelu lähtee kuitenkin liikkeelle hyvin vaatimattomista lähtökohdista. Asiakas ja myyjä luonnostelevat paperille hahmotelman siitä, mitä asiakas tarvitsee. Pohjana voi olla jokin valmis rakennusmalli, tai liikkelle lähdetään ihan tyhjältä ruutupaperilta. Luonnossuunnittelijan tehtävänä on muodostaa näistä hahmotelmista ensimmäinen luonnosmalli suunnitteluohjelmalla. Rakennuksen luonnossuunnittelu lähtee usein liikkeelle siitä, että otetaan asiakkaan hahmotelman kanssa mahdollisimman samankaltainen rakennusmalli ja aletaan siitä muokkaamaan hahmotelmaa vastaavaa kokonaisuutta suunnitteluohjelmalla. Silloin rakennuksesta muodostuu ensimmäinen virallinen tietomalli ja siitä voidaan asiakkaalle tarjota nähtäväksi kolmiulotteinen visualisoitu ja usein myös interaktiivinen malli.

2.1 Vertex BD suunnitteluohjelma

Rakennusalan johtaviin suunnittelu- ja tiedonhallintaohjelmistoihin kuuluu suomalainen Vertex BD, jonka toimittaja on Tamperelainen yritys Vertex Systems Oy. Vertex Systems aloitti jo vuonna 1977 kehittämällä rakennusten lujuuslaskentaan soveltuvia tietoteknisiä apuvälineitä. Pian he huomasivatkin, että koko rakennussuunnittelun siirtäminen piirrustuspöydältä tietokoneelle olisi se todennäköisin tulevaisuuden trendi. Väärässä he eivät olleetkaan, sillä nykyään on harvassa sellaiset rakennussuunnitelmat, jotka arkkitehti on käsin piirtänyt. Yrityksen perustajajäsen ja hallituksen puheenjohtaja Mikael Piekkala toteaakin yrityksen 40-vuotis artikkelissa, että Vertex Systems oli mukana uurtamassa uraa teollisuuden digitalisaatiolle ja nykypäivän startupeille. He saivat ensimmäiseksi asiakkaakseen Vesiluodon talotehtaan ja siitä lähtien on rakennussuunnittelun suunnitteluohjelma ollut yksi Vertexin keskeisimmistä tuotteista. (Vertex Uutiset 2018). Vertex BD tallentaa kaikki rakennuksen suunnitteluvaiheet älykkääseen tietomalliin, joka siirtää tietoa rakennuksesta kerroksittain eri tasojen välillä. Tietomalliin tallentuu virtuaalisesti kaikki todellisia rakennusosia vastaavat osat ja niiden suhteet toisiinsa. Tietomallista tieto siirtyy helposti myös ohjelmiston ulkopuolelle, esimerkiksi helpottamaan tarjouslaskentaa ja rakennusmateriaalien määrälaskentaa. (Vertex BD yleiskatsaus 2018.)

2.2 **Rakennuksen tietomalli**

Rakennuksen tietomalli (Building information model, BIM) on moniulotteinen käsite. Se on rakennuksen digitaalinen ilmentymä, joka sisältää kolmiulotteisena kaiken tiedon rakennuksesta sen pienintä mahdollista yksityiskohtaa myöden. Kuvassa 1 on mallinnettuna rakennuksen kaksi seinää ja ikkuna. Kuvasta voi huomata, että seinien rakenneosat näkyvät kerroksittain ja jokainen käytetty lauta on erikseen mallinnettu. Myös ikkuna vastaa todellisuutta ja se on myös visualisoitu siten, että lasin läpi voi nähdä. Tietomalli on tietokanta projektiin liittyvästä informaatiosta ja sitä voi hyödyntää niin rakennussuunnitteluun tarkoitetut ohjelmistot kuin myös muut ulkopuoliset ohjelmistot. Rakennuksen tietomalli pyrkii mallintamaan ja yhdistelemään toisiinsa kaikki rakennuksen osat tarjoten niistä mahdollisimman realistista ja monipuolista tietoa. Rakennuksen tietomallia voi rakennussuunnittelun lisäksi käyttää esimerkiksi projektihallinnassa, rakennuksen sisustuksen suunnittelussa, sekä rakennuksen huoltosuunnitelmia laadittaessa.
(Miettinen & Paavola 2014)



Kuva 1. Rakennuksen seinän mallinnus (Vertex BD yleiskatsaus 2018).

Nykyaikaiset rakennussuunnittelua tukevat ohjelmistot ovat automatisoineet tietomallin luomisen siten, että tietomalli rakentuu samalla kun suunnittelija luonnostelee rakennusta. Rakennus muodostuu erilaisista elementeistä jotka on etukäteen tarkasti määritelty ja joita suunnitteija voi nopeasti lisätä, muokata ja poistaa käyttäessään ohjelmistoa. Tämä vähentää suunnittelun aikana tapahtuvia inhimillisiä virheitä, sillä suunnittelijan ei tarvitse puuttua vakioelementtien yksityiskohtiin ollenkaan. Rakennuksen tietomalliin voi tuoda tietoa tai sitä voi siirtää eteenpäin hyödyntämällä IFC-ominaisuusjoukkoja. (Vertex BD yleiskatsaus 2018.)

2.3 Rakennuksen tietomallin hyödyntäminen pelisuunnittelussa

Voiko BIM-malleja hyödyntää peleissä ja muissa 3D-ympäristöissä? Vastaus on kyllä. Monissa peleissä käytetään rakennuksia, jotka jäljittelevät todellisuutta niin tarkasti kuin mahdollista. Tällainen rakennus on helpointa luoda käyttäen BIM-mallinnusta ja siirtämällä malli eteenpäin pelimoottoriin. (Kuva 2.) Sen lisäksi että BIM-malleja voidaan hyödyntää pelikenttien luonnissa, tietomalli voi tuoda peliin tietoa rakennuksen eri osien materiaaleista tai muista ominaisuuksista, joita voidaan hyödyntää pelissä. BIM tietomalli on semanttinen tietokanta, jonka avulla voidaan luoda monimutkainenkin rakennuskompleksi, jonka sisällä ja ulkona pelihahmo voi kulkea. BIM-mallien vienti 3D-pelimoottoriin ei kuitenkaan aina ole yksinkertaista. Monesti pelinkehittäjän tarvitsee käyttää erillistä mallinnustyökalua, voidakseen siirtää suunnittelemansa rakennukset CAD-sovelluksesta pelialustalle. (Yan, Culp & Graf 2010)

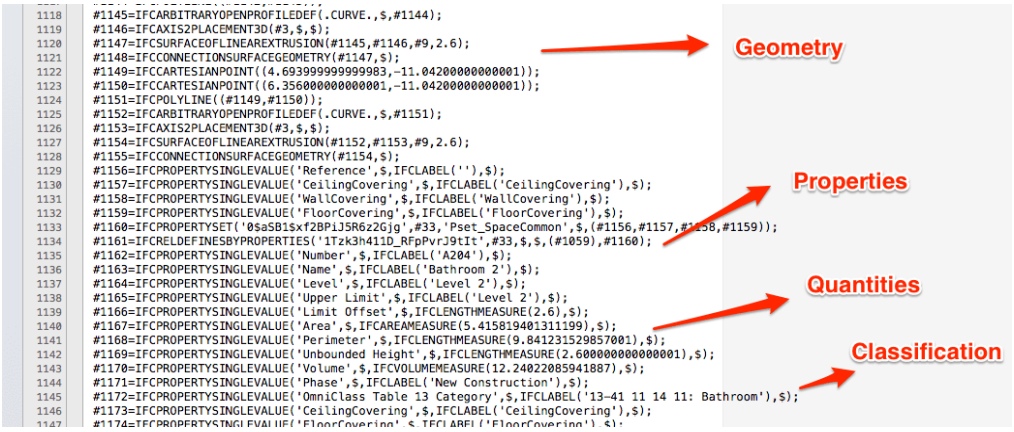
Kuva 2. BIM-malli Unityssä (Hougaard 2014)

Nykyään kehitteillä on kuitenkin joustavampia ja interaktiivisempia tapoja viedä BIM-dataa suunnitteluohjelmista pelialustoille siten, että mallit ovat päivitettävissä ja parhaimmassa tapauksessa suoraan muokattavissa. Käyttäjän on paljon helpompaa hahmottaa tilaa ja ympäristöä, jos hän voi kulkea rakennuksessa ja sen ulkopuolella sen sijaan, että rakennusta tarkastellaan pyörittelemällä sen 3D-mallia. Unity on joustava alusta BIM-tietomallien käsittelyyn, koska saatavilla on monia eri ratkaisuja ongelmiin, joita todella yksityiskohtaiset ja isot BIM-mallit voivat aiheuttaa. NVYVE studion johtava suunnittelija Adam Simonar toteaa: ”Nykyään asiakkaan on mahdollista astua sisään tulevaan kotiinsa jo vuosia ennen sen valmistumista ja nähdä kuinka aurinko paistaa sisään ikkunoista aamulla” Zahner on yritys, joka on käyttänyt Unityä kehittääkseen teknologiaa, jossa asiakas itse voi suunnitella julkisivuja ja nähdä muutosten vaikutukset rakennuksen hinta-arvioon. (Hougaard 2014)

2.4 IFC-ominaisuusjoukot

Industry Foundation Classes (IFC) kerää yhteen kaiken rakennuksen tietomallin yhteydessä tarvittavan tiedon, ja esittää sen yhtenä tiedostona, jonka voi avata lähes missä tahansa rakennussuunnittelu ohjelmassa. IFC-tiedonsiirto on vastaus siihen, miten rakennuksen suunnitteluun tarvittavaa tietoa voi helpoiten siirtää esimerkiksi rakennussuunnittelijalta sähkösuunnittelijalle, joka käyttää eri ohjelmistoa suunnitelmiensa tekoon. IFC-standardi kehitettiin 1994 parantamaan rakennusten tietomallien käytettävyyttä ja yhteentoimivuutta rakennusalan eri ohjelmistojen välillä. IFC-standardia ISO 16739:2013 valvoo nykyään buildingSMART, joka on rakennusalan standardeja maailmanlaajuisesti kehittävä yhteisö. IFC-tiedostoja tuottaa ja lukee pääasiassa rakennusten tietomalleja käsittelevät ohjelmistot. (Laakso & Nyman 2016)

IFC-tiedosto on kirjoitettu EXPRESS tietomallinnuskielellä ja se on avoin dataformaatti, jonka määrittelee ISO 10303:21 standardi. Se sisältää tekstimuodossa kaiken sen tiedon, mitä rakennuksen tietomalli tarvitsee avautuakseen eri ohjelmistoissa. IFC-tiedosto on luettavissa tekstitiedostona (Kuva 3.), mutta se on vaikealukuista, sillä se on tarkoitettu lähinnä suunnitteluohjelmien luettavaksi. IFC-tiedosto etenee askelmina, jolloin seuraava askel voi aina viitata aikaisempiin askeliin. IFC-tiedostoon tallennetaan kaikki rakennuksen tiedot mahdollisimman yksityiskohtaisesti. Se ei ole vain geometristä dataa koordinatteina, vaan tiedostoon voi lisäksi tallentaa tietoa, ominaisuuksia ja määriä rakennuksen elinkaaren kaikista eri vaiheista. Kun rakennuksen tietomalleja siirrellään eri ohjelmistojen välillä, osa tietomallin tiedoista katoaa, siksi natiivi tietomalli on aina korkealaatuisempi kuin IFC-tiedostosta avattu tiedosto. (What is IFC and what do you need to know about it? 2016)



Kuva 3. IFC-tiedoston tekstimuoto (What is IFC and what do you need to know about it? 2016)

3 SOVELLUSKEHITYS UNITY PELIMOOTTORILLA

Nykyään lähes kuka tahansa voi suunnitella pelejä ja sovelluksia erilaisille alustoille kuten esimerkiksi kännykkään ihan harrastepohjalta, sillä siihen tarvittavat välineet on kehitetty monipuolisiksi ja helppokäyttöisiksi, eikä niiden hankkimiseen tarvitse sijoittaa suuria summia rahaa. Erilaisia kännykällä pelattavia pelejä syntyykin jatkuvasti ja menestyneen pelin kehittämisen unelma on harrastajankin saavutettavissa. Alun perin grafiikkamoottorit kehitettiin isojen ja tehoja vaativien pelien taustalle, mutta nykyään pelimoottoria voi hyödyntää myös muunlaisiinkin sovelluksiin, kuten esimerkiksi virtuaalitodellisuutta ilmentäviin sovelluksiin tai havainnollistamaan esimerkiksi kokonainen kaupunki 3D-mallina.

3.1 Pelimoottori

Pelimoottori (game engine) on rajapinta (framework), joka on kehitetty helpottamaan pelisuunnittelijoiden työtä heidän kehittäessään 2D- ja 3D-pelejä, simulaatioita, virtuaalitodellisuutta ja vastaavia tuotteita tietokoneille tai mobiililaitteille. Se tarjoaa ohjelmoijalle mm. grafiikkaan liittyvää laskentaa. Pelin kehittäjä voi siis tuoda objekteja peliin ja pelimoottori osaa laskea objektille vallitsevien valaistusolosuhteiden mukaisen ulkoasun. Se myös antaa työkalut käsitellä objektien interaktiivisuutta ja liikkumista peliympäristössä. Lisäksi pelimoottorin avulla voi tuoda suunniteltavaan peliin erilaisia ääniasetuksia, erikoistehosteita, animaatioita yms. (Game engines – how do they work? 2018). Pelimoottorin avulla pelin kehittäjä voi keskittyä olennaiseen eli pelin toiminnallisuuteen ja tarinan läpiviemiseen ja pelimoottori tekee taustalla vaadittavan työn, joka muuten veisi pelin kehittäjältä hurjasti työaikaa.

3.2 Unity

Tanskalaiset ohjelmoijat Nicholas Francis, Joachim Ante ja David Helgason aloittivat Unityn kehittämisen vuonna 2002 alkuperäisenä tarkoituksenaan kehittää omiin pelimoottoreihinsa sopiva objektien varjostukseen käytettävä lisäosa eli Shader. Pian miehet päätyivät yhdistämään jo kehittämänsä pelimoottorit ajatuksenaan kehittää peli ja sitten lisensoida pelin käyttämä tekniikka. Pelin sijasta he kuitenkin loivat pelien tekemiseen soveltuvan työkalun Unityn, jonka perimmäinen idea on antaa työkalut pelien tekemiseen kaikille asiasta kiinnostuneille mahdollisimman pienellä vaivalla. Unityn ensimmäinen versio julkaistiin vuonna 2005 ja vuonna 2009 siitä julkaistiin myös Windows yhteensopiva versio. (United they stand 2009)

Unityn ilmaisversion voi ladata ja sitä voi käyttää ilman kuluja. Kaiken mitä Unityllä luo saa omaan omistukseensa. Unity näyttää ilmaisversiolla tuotetuiden pelien alussa oman logonsa. Tämä on mahdollista niin yksityisille kuin yrityksillekin, kunhan yrityksen vuotuinen tulos jää alle 100 000 dollarin. Mikäli yritys tienaa enemmän kuin 100 000 dollaria, on heidän siirryttävä Unity Pron käyttäjiksi, jolloin lisenssi maksaa 125 dollaria kuukaudessa. (Unity Personal. 2018)

Unityssä yhdistyy kolme tärkeää pelinkehitystä helpottavaa ominaisuutta:

- Pelimoottori, joka auttaa pelimaailman luomisessa ja visualisoinnissa
- Sovellus, jolla ohjelmoija voi helposti yhdistää pelin graafiset ja toiminnalliset ominaisuudet
- Editori ohjelmakoodin kirjoittamiseen ja helppoon yhdistämiseen oikeiden peliobjektien kanssa.

(Buyuksalih, Bayburt, Buyuksalih, Baskaraca, Karim & Rahman 2017)

Unity on helppokäyttöinen alusta, jolla myös harrastelija voi aloittaa pelin kehittämisen suhteellisen vaivattomasti. Unity tarjoaa kattavat ohjeet ja käy läpi perustoimintoja videomateriaalin avulla, jossa vaiheet käydään läpi askel askeleelta. Unityllä saa kuitenkin aikaan näyttävän lopputuloksen, sillä pelimoottori tukee 3DNVIDIA PhysX grafiikkamoottoria, jolla saadaan aikaan erittäin yksityiskohtaista ja erittäin realistista grafiikkaa. (The world's leading content-creation engine. 2018)

Unityn suurin etu on sen skaalatuvuus eri alustoille. Unityllä voi luoda sovelluksia Windows, Mac, Linux ja Android-alustoille ja Unityllä tehtyjä sovelluksia voi julkista myös iPhone- ja Windows-puhelimissa. Unityllä valmistetun sovelluksen voi siis helposti saada julkaistua suurelle joukolle käyttäjiä. (Buyuksalih ym. 2017)

3.2.1 3D-mallien käyttö

Unity tarjoaa käyttäjälle perusobjekteja kuten kartioita, sylintereitä ja palloja, joita voi käyttää apuna tasojen luomisessa tai esimerkiksi väliaikaisina objekteina ennen varsinaisten peliobjektien tuomista ohjelmaan. Unityssä on myös valmiiksi rakennettu pelihahmo (Ethan), jonka käyttäjä voi ladata ohjelmaan ilman lisäkustannuksia latamaalla paketin Unityn tarjoamasta Standard Assets hakemistosta. Ethan liikkuu tasolla perusnäppäimiä käyttämällä ilman mitään lisäohjelmointia. Näiden lisäksi Unityssä on käyttäjille avoin Asset Store, jossa sekä Unityn kehittäjät että käyttäjät voivat julkaista tekemiään grafiikoita, objekteja, animaatioita, ohjelmapätkiä tai Unityyn kytkettäviä lisäosia. Käyttäjät voivat julkaista sisältöä joko ilmaiseksi tai pientä korvausta vastaan. (Ralph. 2017)

Unityn ulkopuolella tuotettuja objekteja on mahdollista siirtää Unityyn, sillä Unity tukee monia yleisiä 3D-objektiformaatteja. Objektien siirtäminen Unityyn on helpoimmillaan todella yksinkertaista. Objektipaketti, joka sisältää 3d-objektin lisäksi myös tarvittavat tekstuurit, kopioidaan Unity-projektin Assets-kansioon, josta Unity tunnistaa objektit automaattisesti. Unity tunnistaa fbx-, dae-, 3ds-, dfc-, obj- ja skp-tiedostoja. On kuitenkin huomioitava, että mikäli alkuperäiseen tiedostoon tekee muutoksia, täytyy se ladata uudelleen Unityyn, jotta muutokset päivittyisivät. Unityyn on mahdollista tuoda myös päivittyviä objektipaketteja mm. Blenderistä, LightWavesta tai 3D Studio Maxista. Tällöin pitää kuitenkin huomioida, että kyseinen ohjelmisto pitää löytyä joka koneelta, jolla Unity-projektia käsitellään. Lisäksi projektiin voi siirtyä myös turhaa tavaraa ja isot tiedostot voivat hidastaa Unity-projektin latausta. (Model file formats. 2018) 3ds-tiedostomuoto toimii suoraan objekteja Vertexistä siirrettäessä. Vertex pakkaa tiedostot ja tekstuurit yhteen 3ds-pakettiin, jonka voi kopioida projektin assets-kansioon.

3.2.2 Ohjelmointi Unityssä

Kaikki Unityssä on ohjelmoitavissa tekemään mitä ikinä tarvitaan. Jokainen näppäimistön painike tai hiiren liike voidaan valjastaa pelin tekijän tarkoituksiin. Unityssä kaiken sitoo toisiinsa kontrollerit, joita ohjelmoidaan käyttäen C# tai UnityScript ohjelmointikieliä. C# on varsin yleinen ohjelmointikieli, joka on myös suosittu Unityn käyttäjien keskuudessa. UnityScript on muokattu JavaScript ohjelmointikielestä ja kehitetty vartavasten Unitylle. Lisäksi myös DLL-yhteensopivat .NET-ohjelmointikielet ovat käytettävissä Unityssä. Unity on tehnyt ohjelmoinnin ja objektien käytön erittäin yksinkertaiseksi yhdistämällä Visual Studion suoraan Unityssä luotaviin scripteihin. Visual Studion voi halutessaan vaihtaa mihin tahansa muuhun editoriin omien mieltymysten mukaan. Jokainen Unityssä käytettävä scripti rakennetaan samalla kaavalla (MonoBehaviour-luokka), jossa tapahtumat tapahtuvat joko kerran scriptin ajon alussa (Start-funktio) tai kerran jokaisen framen aikana ajon ollessa käynnissä (Update-funktio). (Creating and Using Scripts. 2018)

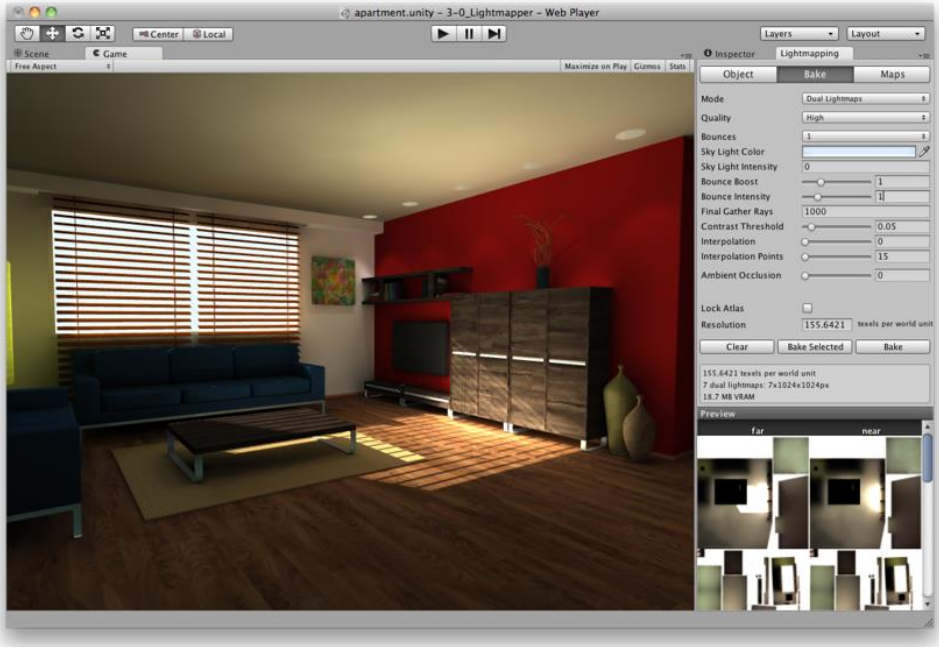
3.2.3 Maaston luonti

Unity 3D:ssä on erittäin kehittynyt maastomoottori (Terrain Engine), jolla voi luoda pikkutarkkoja ja laajoja pelimaailmoita. Unityn pelimaiseman voi tehdä elävämmäksi lisäämällä tuuliefektejä sekä ääntä. Unityssä on paljon asetuksia, joilla voi vaikuttaa maaston valaistukseen, elävyyteen ja äänimaailmaan. Maaston objektit, kuten puut ja ruohonkorret liikkuvat tuulen mukaisesti ja maaston eri kohtiin voi asettaa erilaisia ääniefektejä kuten linnun laulua. Maaston renderöinti on optimoitu mahdollisimman tehokkaaksi, jotta peli toimisi joustavasti vaikka pelimaailmassa on paljon ladattavia yksityiskohtia. Unity tukee maaston korkeuseroja kuvaaviin harmaasävyisiin korkeuskarttoihin (Heightmap), joissa maaston

korkeuserot ilmoitetaan vaaleilla ja tummilla sävyillä siten, että vaalein on korkein kohta ja tummin on syvin kohta kartassa. Korkeuskartta tulee tuoda Unityyn RAW-tiedostona. (Height Tools. 2018)

3.2.4 Valon hallinta

Unityssä valojen ja varjojen renderöinti on viety huippuun ja pelimaailman valaistusta voikin säätää monessa eri paikassa ja monella eri tavalla. Periaatteena on, että pelimoottorille kerrotaan valonlähde, valon väri ja valon suunta ja sen mukaan Unity laskee objektille valaistuksen ja varjot. Yhteen objektiin voi vaikuttaa monta eri valonlähdettä ja valaistusasetukset lasketaan joka kehyksessä (frame). Unity pelimoottori voi laskea monimutkaisiakin valaistusvaikutuksia erilaisille objekteille ja niiden materiaaleille. Tällöin on käytössä valaistuskartta ja sen mukaisen valaistuksen laskentaprosessia kutsutaan valaistuskartan leivonnaksi (Lightmap baking) (Kuva 4). Valoa voi Unityssä säätää globaalisti, objektikohtaisesti, ja kamerakohtaisesti. On siis oltava tarkkana, että kaikki säädöt tukevat toisiaan, että saadaan aikaiseksi mahdollisimman realistinen valaistus. (Lighting overview. 2018)



Kuva 4. Valaistuksen laskentaa ja sen näyttävyyttä rakennuksen sisätiloissa. (Lightmapping Quickstart. 2017)

4 KEHITYSPROJEKTIN SUUNNITELMA

Tässä kehitysprojektissa käytetään kahta eri tutkimusmenetelmää: soveltavaa ja komparatiivista tutkimusta. Kehitysprojektissa tulee olemaan kolme eri vaihetta, joissa kussakin keskitytään yhteen keskeiseen ongelmaan. Kehitysprojektin arvoitu kesto on noin 20 päivää.

Soveltava tutkimus on sellaista tutkivaa toimintaa, joka pyrkii luomaan uutta tietoa ja menetelmiä ennalta asetettujen ongelmien ratkaisemiseksi. (Tutkimus- ja kehittämistoiminta 2018). Tässä soveltavaan tutkimukseen pohjaavassa kehitysprojektissa pyritään selvittämään, voiko tavallinen käyttäjä kännykkäsovelluksella suunnitella sellaisen kokonaisuuden, joka voidaan siirtää arkkitehdin vaativamman tason ohjelmistoon. Kehitysprojektin aikana käydään läpi tutkimukselle asetetut kysymykset: Pystyykö Unityä hyödyntämään rakennussuunnittelussa? Pystyykö arkkitehdin luoman rakennuksen siirtämään Unityyn? Miten asiakas pystyisi tekemään rakennukseen muokkauksia? Pystyykö asiakas suunnittelemaan oman rakennuksen? Ja saako tällaisen muokatun rakennuksen palautettua takaisin arkkitehdin työpöydälle?

Komparatiivinen tutkimusmenetelmä eli vertailututkimus on keskeinen analyysimenetelmä. Siinä testataan rutiininomaisesti erilaisia hypoteeseja ja vertailun tuloksena saadaan vahvistusta teorialle ja löydetään mahdollisia uusia ulottuvuuksia tutkimukseen, joiden pohjalta voidaan tehdä uusia hypoteeseja. Komparatiivisessa vertailtutkimuksessa käydään systemaattisesti läpi tutkimuskohteita ja etsitään niistä yhtäläisyyksiä ja eroavaisuuksia riippuen tutkimuksen kohteesta ja asetetuista tutkimusongelmista. Vertailun kohteeksi valitaan yleensä vain muutama lähdemateriaali, koska on tärkeämpää tehdä huolellinen vertailu pienelle määrälle materiaalia kuin vetää suuria linjoja, koska liian suuren lähdemateriaalin läpikäyminen veisi liikaa aikaa ja resursseja. (Collier 1993.) Komparatiivista tutkimusta käytetään kehitysprojektin kolmannessa vaiheessa selvittämään, voiko IFC-tiedostoa muunnella manuaalisesti ja sen seurauksena palauttamaan muunnettua tietoa takaisin Vertexiin.

4.1 Ensimmäinen vaihe: Rakennuksen siirto Vertexistä Unityyn

Tässä vaiheessa käyn läpi menetelmän, jonka avulla Vertexillä suunniteltu rakennus tai rakennuksen osa voidaan siirtää Unityyn. Testaan myös objektin käytettävyyttä kulkemalla rakennuksessa ja sen ulkopuolella Unityn standaripelihahmolla. Tämän testausvaiheen arvioitu kesto on noin viisi päivää ja sen tuloksena kirjaan ylös tekemiäni huomioita ja mahdollisen vastauksen siihen, pystyykö objekteja siirtämään Vertexistä Unityyn ja kuinka helppoa se on?

4.2 Toinen vaihe: Rakennuksen muokkaus Unityssä

Tässä vaiheessa pyrin testaamaan, voiko Unity-sovellusta käyttäen tehdä muokkauksia rakennukseen. Lähestyn ongelmaa myös toisesta näkökulmasta ja kokeilen valmistaa sovelluksen, jossa rakennuksen voi suunnitella käyttämällä Vertexistä tuotuja rakennuksen vakioelementtejä. Näitä ovat esimerkiksi seinäelementti, ikkunaelementti ja ovielementti. Tämän testausvaiheen arvioitu kesto on noin kymmenen päivää ja testauksen tuloksena kirjaan ylös tekemiäni huomioita sekä mahdolliset ratkaisut siihen, pystyykö Vertexistä tuotua rakennusta muokkaamaan Unityllä ja pystyykö Unity-sovelluksella suunnittelemaan pienelementtirakennusta.

4.3 Kolmas vaihe: Muokatun rakennuksen siirto Vertexiin

Tässä vaiheessa pyrin saamaan vastauksen siihen, pystyykö Vertexillä avaamaan suunnitteluohjelman ulkopuolella muokatun rakennuksen. Päämääränä on se, että Unityllä tehdyssä sovelluksessa tehdyt muutokset voisi palauttaa Vertexiin. Tässä hyödynnetään IFC-tiedostoa, jonka avulla Vertexiin on mahdollista siirtää tietoa suunnitteluohjelman ulkopuolelta. IFC-tiedoston kirjoittaminen alusta asti niin, että Vertex ymmärtäisi tiedoston sisällön, ei ole helppoa. Jopa pienen muutoksen kirjoittaminen IFC-tiedostoon on monimutkaista, sillä useat ominaisuudet ovat konekielisiä ja sama informaatio saattaa esiintyä tiedostossa monessa eri kohdassa. Tässä tutkimuksessani tulen käyttämään komparatiivista tutkimusmenetelmää IFC-tiedostojen vertailuun.

Tässä tutkimuksessani valmistan ensin kaksi erillistä IFC-tiedostoa Vertex arkkitehtiohjelmaa apuna käyttäen. Ensimmäisessä tiedostossa on rakennus ennen haluttua muutosta ja toisessa tiedostossa on rakennus muutoksen jälkeen. Vertailen näitä kahta IFC-tiedostoa komparatiivista tutkimusmenetelmää hyödyntäen, ja kirjaan ylös havaitsemani eroavaisuudet tiedostojen välillä, sekä arvioni siitä, pystyykö IFC-tiedostoa muuttamaan manuaalisesti ja palauttamaan Vertex-ohjelmaan halutun muutoksen aikaansaamiseksi.

Tämä vaihe kestää arviolta viisi päivää ja testauken tuloksena kirjaan tekemiäni huomioita ja mahdolliset ratkaisut siihen, pystyykö Unityllä muokattua tai rakennuselementeistä suunniteltua rakennusta palauttamaan Vertexiin.

5 SOVELLUSTESTAUSPROJEKTI

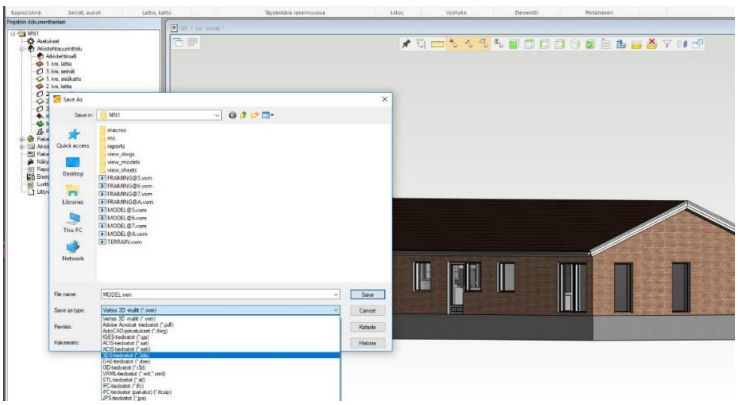
Sovellustestausprojekti kesti noin neljä viikkoa ja sen aikana testasin Vertexillä suunnitellun rakennuksen siirtoa Unityyn. Siirretyn rakennuksen muokkaamista ja rakennuksen suunnittelua käyttäen Vertexistä tuotuja rakennuselementtejä. Lisäksi kehitin sovellusta, jolla asiakas voi suunnitella rakennuksen käyttäen valmiita vakioelementtejä. Seuraavissa kappaleissa käyn sovellustestausvaiheet läpi kohta kohdalta. Viimeisessä vaiheessa testasin muutosten tekemistä IFC-tiedostoon ja sen avaamista Vertexillä.

5.1 Rakennuksen siirto Vertexistä Unityyn

Projektin ensimmäisessä vaiheessa selvitin, onko Vertexistä mahdollista siirtää rakennus Unity 3D:n objektiksi ja mitä kaikkea tulee ottaa huomioon, että lopputuloksesta tulee käytettävä. Rakennuksen siirtoa varten valmistin ensin Unityssä sopivan maaston. Latasin valmiin maastopaketin Asset Storesta ja siistin siitä ylimääräisiä objekteja pois, jotta sain helpolla näyttävän palan puistoa, johon voin rakennuksia tuoda. En katsonu tarpeelliseksi valmistaa maastoa alusta alkaen, sillä se ei ole tämän työn keskeinen osa. Maaston luominen alusta on kuitenkin hyvä idea, mikäli haluaa näyttää asiakkaalle tulevan rakennusprojektin oikeanlaisessa ympäristössään. Unityllä voi helposti toteuttaa juuri oikeanlaiset maaston korkeuserot ja puuston, kuin tarve vaatii.

5.1.1 Rakennuksen valmistelu Vertexissä

Vertexillä suunnittelin pari erilaista rakennusta ja muutaman elementin testausta varten. Kun rakennukset olivat julkaisukunnossa, tallensin ne Vertexissä 3ds-formaattiin, jota Unity osaa suoraan lukea. Tämä tapahtuu Vertexissä kohdassa File → Vie → Tallenna toiseen formaattiin. Ja valitsemalla tallennusmuodoksi 3ds-tiedostot. (Kuva 5.) Tämä luo Vertex-projektista kansion, jonka sisällä on 3d-objekti ja siihen käytetyt pintamateriaalit.



Kuva 5. 3ds-tiedoston tallentaminen Vertexissä

5.1.2 Rakennuksen tuonti Unityyn

Unityyn 3d-objektin sai siirrettyä tallentamalla Vertexissä luodun kansion Unity-projektin alla olevaan Assets-kansioon. Unity tunnistaa 3d-objektin ja elementit automaattisesti. Sitten 3d-objektin voi siirtää luotuun maailmaan. Objektin koko on aluksi 1:1:1, joten mikäli maasto on hyvin pienessä mittakaavassa, voi rakennuksen mittasuhteita joutua muokkaamaan alaspäin, jotta talo sopisi haluttuun ympäristöön. Tämä onnistuu helposti tuotavan objektin asetuksissa. On hyvä myös huomioida, että mikäli rakennus on maastoon nähden liian suuri, joutuu karttaa zoomaamaan reilusti ulospäin, jotta rakennuksen voi nähdä.

5.1.3 Valaistuksen säätö Unityssä

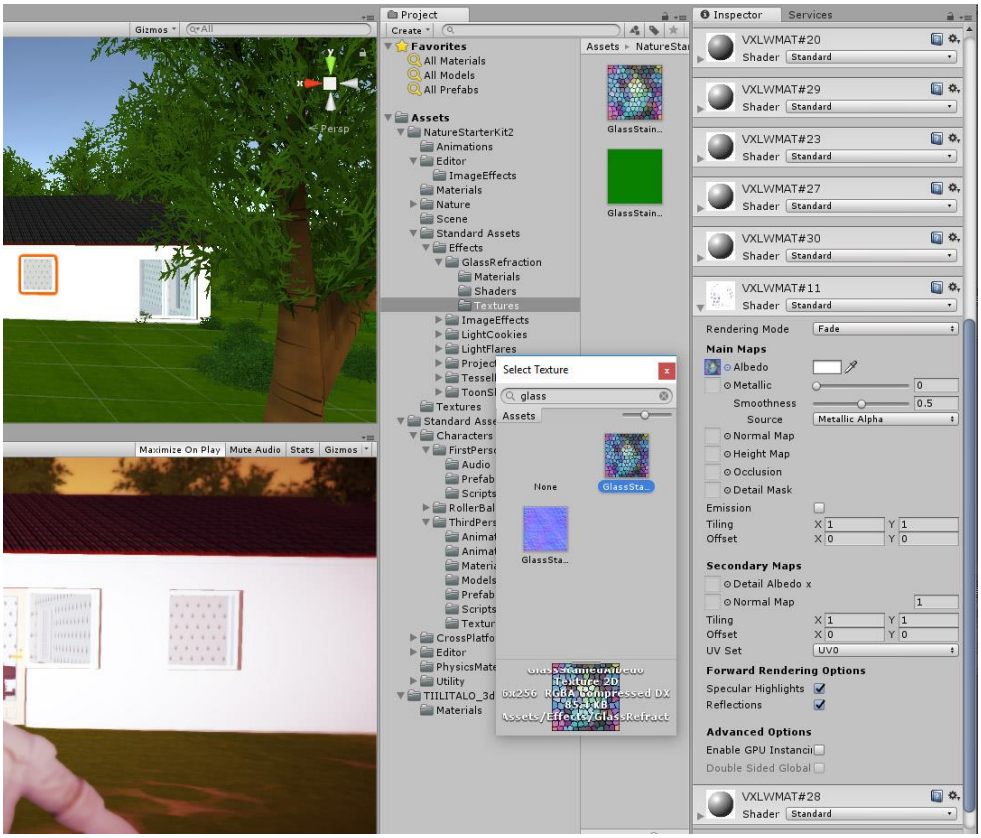
Objektiin kohdistuvaa valaistusta voi säätää monella eri tavalla ja monesta eri paikasta. Kameralla (Main camera), yleisellä valaistuksella (Directional lighting ja koko maailmalla on kaikilla omat asetuksensa. Mikäli objekti on maastossa kummallisen värinen esim. vaaleansininen, se voi johtua siitä, että yleisen valon väri on sininen. Tämän voi korjata valikosta Windows → Lighting ja valitsemalla scenen väriksi esimerkiksi valkoisen. (Kuva 6.)

Kuva 6. Yleisvalon värin säädön vaikutus objektiin

5.1.4 Ikkunoiden materiaalin muuttaminen lasiksi

Vaikka rakennuksen ikkunoiden materiaali Vertexissä on lasia ja siten läpinäkyvää, ei se Unityyn tuotaessa ole säilyttänyt automaattisesti läpinäkyvyyttään. Unityssä on kuitenkin tarjolla runsas joukko erilaisia perusmateriaaleja, joiden joukosta löytyy myös läpinäkyvää lasia. Rakennuksen osat ovat erikseen valittavissa, joten ikkunalasin materiaalin voi Unityssä vaihtaa haluamakseen. Vertex on luonut jokaisesta ikkunaelementin materiaalista geneerisesti nimetyn materiaalin (esim. VXLWMAT#11 on esimerkkitapauksen ikkunalasimateriaali), joten täytyy olla tarkkana, että valitsee oikean materiaalin listasta. Materiaalin saa muutettua lasiksi, valitsemalla seuraavat asetukset: Rendering Mode = Fade ja Albedon tekstuuriksi valitaan Standard Assets hakemistosta

GlassStainedAlbedo. (Kuva 7.) Riittää, että muutoksen tekee yhteen ikkunalasielementtiin ja kaikkien rakennuksen samankaltaisten ikkunoiden lasi muuttuu läpinäkyväksi. Rakennuksen ikkunalasien muokkaaminen läpinäkyväksi tuo huomattavasti lisää todellisuudentuntua rakennukseen. (Kuvat 7 ja 8.)



Kuva 7. Objektin materiaalin muuttaminen lasiksi

5.1.5 Pelihahmon käyttäminen Unityssä

Unity tarjoaa myös täysin toimivan geneerisen pelihahmon, jolla voi testata pelaajan liikkumista rakennuksen ulko- ja sisäpuolella. Pelihahmon saa ladattua kenttään Standard Assets hakemistosta ja se on täysin ohjattavissa ilman mitään sen kummempia säätöjä. Pelihahmon koko kannattaa skaalata sopivaksi peilaten rakennuksen kokoa. Mikäli objekti on tuontivaiheessa rakennettu siten, että sen materiaalit ovat läpäisemättömiä (Generate Colliders), ei pelihahmo pysty kulkemaan rakennuksen läpi. Rakennuksen ovista on kuitenkin mahdollista tehdä läpäiseviä, poistamalla deaktivoimalla Mesh Collider ovielementin valikosta. Sen jälkeen hahmo voi kulkea ovien läpi myös talon sisälle. (Kuva 8.)

Kuva 8. Pelihahmo katselee ulos keittiön ikkunasta.

5.2 **Rakennuksen muokkaaminen Unityssä**

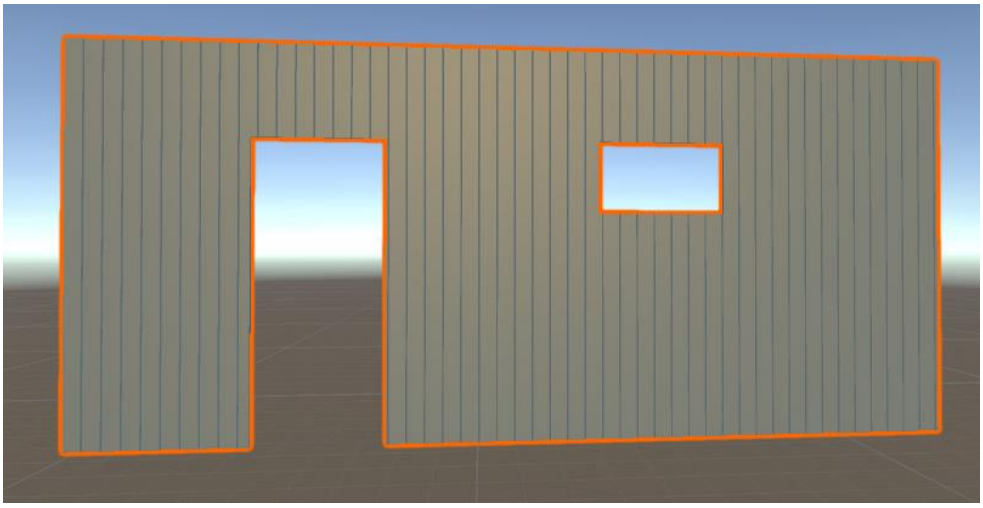
Tässä projektin vaiheessa on tarkoitukseni selvittää onko rakennukseen mahdollista tehdä muutoksia Unityssä. Muutoksia ovat esimerkiksi ikkunan tai oven siirtäminen tai rakennuksen venyttäminen pidemmäksi. Aion myös testata voiko rakennuksen rakentaa ns. palikoista eli elementti kerrallaan, jolloin käyttäjällä on enemmän mahdollisuuksia vaikuttaa rakennuksen ulkonäköön.

5.2.1 Ikkunan siirto ja rakennuksen koon muuttaminen

Kun rakennus on siirretty Vertexistä Unityyn, on jokaisesta tallin osasta luotu oma elementtinsä Unityyn. Esimerkiksi testauksessa käyttämäni perustalli muodostuu 127 osasta. Yksi ikkunakin koostuu ikkunaruudusta ja neljästä sitä ympäröivästä pielilaudasta. Ikkunan paikkaa pystyy siirtämään ja sen mittasuhteita pystyy muuttamaan Unityn xyz-koordinaatistoa hyödyntämällä. Ongelmaksi muodostuu kuitenkin rakennuksen seinä, jossa muutoksen tulisi näkyä. Rakennuksen siirtovaiheessa rakennuksen seinään on tehty aukot oletettujen ikkunoiden ja ovien mukaisesti (Kuva 9.). Nämä aukot eivät muuta paikkaa Unityssä, vaikka ikkunaa siirtäisi tai sen kokoa muuttaisi. Siitä johtuen, ikkunan siirto ei tuota haluttua lopputulosta.

Autotallin seinäpaloja voi tuoda Unityyn siten, että kaikki mahdolliset ikkunoiden ja ovien siirtomahdollisuudet on otettu huomioon ja ikkunan siirron yhteydessä Unity vaihtaa myös seinäpalan oikeaksi. Tämä on kuitenkin rajoittava tekijä monessakin mielessä, sillä asiakas saattaa haluta lisätä ja poistaa ovia ja ikkunoita ja muuttaa niiden kokoja mielin määrin.

Jos ikkuna voi olla seinällä neljässä eri kohdassa ja se voi olla neljää eri kokoa ja lisäksi ovi voi olla seinällä neljässä eri paikassa on vaihtoehtoisten seinien määrä 3^4 = 81 kappaletta. Lisäksi seinä muodostuu monesta eri kerroksesta, jotka on kaikki leikattu samaan malliin. Ei siis riitä, että muutoksen tekee ulkoseinän aukkoihin, vaan lisäksi muutos on tehtävä eristekerrokseen, tuulensuojalevyyn ja sisäseinään. Vaihtoehtoisten seinien määrä nousee näin 324 kappaleeseen. Mikäli nämä kaikki vaihtoehdot tuodaan Unityyn, kuormittaa se ohjelman muistia ja tekee ohjelman käyttämisestä hitaampaa.



Kuva 9. Autotallin seinän uloin kerros siten, kun se on Unityyn siirtynyt.

Rakennuksen venytys vaikuttaa edellä mainittuihin valmiiksi leikattuihin seinälevyihin siten, että aukot venyvät samassa suhteessa seinän kanssa. Ovia ja ikkunoita on siis mahdotonta säilyttää oikean kokoisina, jos rakennusta aletaan venyttämään Unitystä löytyvillä keinoilla. Kokonaisten rakennusten käsittely ja muokkaus Unityssä ei siksi nykyisessä muodossaan palvele rakennussuunnittelua.

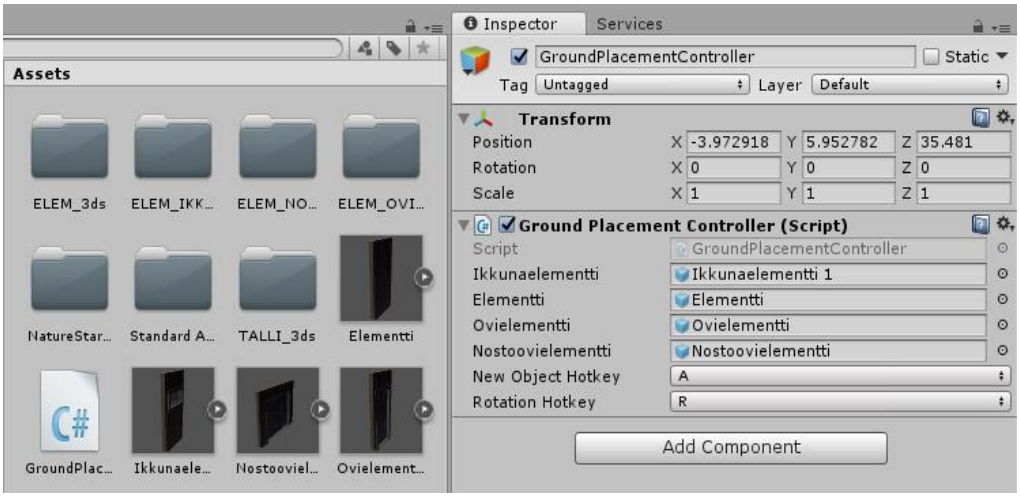
5.2.2 Rakennuksen suunnittelu käyttäen elemettejä

Rakennuksen suunnittelu elementeistä kokoamalla antaa ohjelman käyttäjälle enemmän vapauksia. Käyttäjää ei ole sidottu tekemään muutoksia jo valmiiseen rakennukseen, vaan hän voi rakentaa elementeistä lähes minkä tahansa kokonaisuuden. Tämä suunnittelutapa on lähempänä asiakkaan ruutupaperille piirtämiä luonnoksia, koska rajoja ei ole laitettu. Kolikon kääntöpuolena on se, ettei asiakkaan suunnittelema kokonaisuus ole välttämättä totetutuskelpoinen, sillä asiakkaalla ei ole tarvittavaa teknistä tietoa rakennuksen toimivuudesta. Tällaisesta suunnitteluohjelmasta voidaan kuitenkin antaa luonnossuunnittelijalle visuaalinen kuva siitä, mitä asiakas haluaa. Luonnossuunnittelijan tehtäväksi jää luonnoksen muokkaaminen toteutuskelpoiseksi ratkaisuksi.

Elementit suunniteltiin Vertexissä vastaamaan mahdollisimman tarkasti autotallin peruselementtejä. Tätä testiä varten loin peruselementin, ikkunaelementin, nosto-ovielementin ja varaston oven sisältävän elementin. Käytin elementeissä pystylaudoitusta ja punaista väriä, mutta varsinaista suunnittelohjelmaa varten, elementit tulisi tietenkin luoda myös muita julkisivuvaihtoehtoja mukaillen. Myös Unityssä on mahdollista vaihtaa elementin pinnan materiaali, joten rakennuksen voi ensin suunnitella yhtä materiaalia käyttäen ja sitten lopussa eri julkisivumateriaaleja testaten.

Kun elementit siirretään Unityyn, tulee ottaa huomioon elementtien oikea skaalaus haluttuun ympäristöön. Elementtien tuomisen yhteydessä oikean koon voi määrittää kätevästi elementin Import Settings valikossa kohdassa Scale Factor. Samassa valikossa voi myös säätää tuotavan objektin valaistusta, pintoja ja tekstuureja. Jotta elementtejen paikkaa ja suuntaa voisi helpoiten hallita, kannattaa jokaista elementtiä varten luoda tyhjä elementti ns. ohjauselementiksi. Tyhjän elementin sisään laitetaan Vertexistä tuotu elementti ja tyhjä elementti toimii ankkurina, jossa määritellään elementin suunta, paikka ja törmäysasetukset (colliders). Ohjauselementeistä tehdään oletuselementtejä (prefab) projektin Assets-kansioon. (Kuva 10.)

Seuraavaksi elementtejen käyttöä varten luodaan kontrolleri C#-kieltä hyödyntäen. Kontrolleri ohjaa käyttäjää tuomaan elementin sovelluksen maisemanäkymään, valitsemaan hiiren rullalla oikean elementin, kääntämään elementtiä 90-asteen kulmassa ja asettamaan elementin haluttuun paikkaan maisemassa. Tässä testissä kontrollerin nimi on Ground placement controller ja siihen pystyy Unityn puolella suoraan liittämään halutut oletuselementit. Lisäksi uuden objektin tuomiseen parhaiten soveltuva näppäin voidaan määritellä kontrollerin asetuksissa. (Kuva 10.)



Kuva 10. Kontrolleriin määritetyt oletuselementit

Ground Placement Controller

Luodaan aluksi tarvittavat oliot ja määritellään mitä tapahtuu kontrollerin käynnistyksen yhteydessä ja jokaisen päivityksen yhteydessä. Päivityksiä tapahtuu jatkuvasti ja nopealla syklillä kontrollerin ollessa käynnissä. Se vaikuttaa mm. siihen, että elementti näyttää liikkuvan luontevasti hiiren liikkeiden mukana. Tämän kontrolleri scriptin pohjana on käytetty Jason Weimannin Unity3d Collegeen 2017 lataamaa skriptiä ja siihen on vaikutteita otettu lisäksi Peysbubbyn 2016 Unity-forumille jakamasta objektien vaihto skriptistä.

```
public class GroundPlacementController : MonoBehaviour {

    [SerializeField]
    private GameObject ikkunaelementti;
    [SerializeField]
    private GameObject elementti;
    [SerializeField]
    private GameObject ovielementti;
    [SerializeField]
    private GameObject nostoovielementti;
    [SerializeField]
    private KeyCode newObjectHotkey = KeyCode.A;
    [SerializeField]
    private KeyCode rotationHotkey = KeyCode.R;

    private GameObject currentPlaceableObject;
    private bool elementIsSwitching;
    private int currentElement;

    private void Start()
    {
        elementIsSwitching = true;
    }

    private void Update ()
    {
        HandleNewObjectHotKey();

        if (currentPlaceableObject != null)
        {
            MoveCurrentPlaceableObjectToMouse();
            RotateFromMouseWheel();
            RotateObject();
            ReleaseIfClicked();
        }
    }
```

Elementin tuonti näppäintä painamalla

Uusi elementti tuodaan ruudulle, kun sovelluksen käyttäjä painaa haluttua näppäintä. (Tässä testitapauksessa näppäin A). Mikäli käyttäjällä on jo ohjauksessa oleva elementti, se korvataan oletuselementillä.

```
private void HandleNewObjectHotKey()
{
 if (Input.GetKeyDown(newObjectHotkey))
 {
```

```
  if (currentPlaceableObject == null)
  {
    currentPlaceableObject = Instantiate(elementti);
  }
  else
  {
    Destroy(currentPlaceableObject);
  }
 }
}
```

Elementti seuraa hiiren liikkeitä

Pääkamera (Main camera) lähettää säteen siihen kohtaan, missä hiiri on kyseisellä hetkellä ja valittuna oleva objekti viedään kyseiseen pisteeseen. Tämä saa aikaan vaikutelman, että objekti seuraa hiiren liikkeitä.

```
private void MoveCurrentPlaceableObjectToMouse()
{

 Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
 RaycastHit hitInfo;
  if (Physics.Raycast(ray, out hitInfo))
  {
   currentPlaceableObject.transform.position = hitInfo.point;
  }
 }
```

Elementin vaihtaminen hiiren rullalla

Kun hiiren rullaa liikutetaan ylös tai alas, vaihtaa se uuden elementin käyttöön tapauslistalta. ElementIsSwitching määritellään todeksi ja kun elementti on vaihdettu palautuu se taas epätodeksi.

```
private void RotateFromMouseWheel()
{

 float w = Input.GetAxis("Mouse ScrollWheel");
 if (w > 0f)
 {
  currentElement = currentElement + 1;
  elementIsSwitching = true;
 }
 else if (w < 0f)
 {
  currentElement = currentElement - 1;
  elementIsSwitching = true;
 }
```

Jokaista elementtiä varten luodaan switch-case, jossa tuhotaan vanha objekti ruudulta ja tuodaan tilalle haluttu objekti. ElementIsSwitching on määritellään epätodeksi muutoksen jälkeen.

```
 if (elementIsSwitching == true)
 {
  switch (currentElement)
  {
  case 0:
```

```
  Destroy(currentPlaceableObject);
  currentPlaceableObject = Instantiate(ikkunaelementti, new
   Vector3(currentPlaceableObject.transform.position.x,
   currentPlaceableObject.transform.position.y,
   currentPlaceableObject.transform.position.z), Quaternion.identity)
   as GameObject;
  elementIsSwitching = false;
  break;
  }
 }
```

Varmistetaan vielä, että currentElement pysyy haluttujen tapausten sisällä.

```
if (currentElement > 3)
{
 currentElement = 0;
}
if (currentElement < 0)
 {
 currentElement = 3;
 }
}
```

Elementin kääntäminen näppäintä painamalla

Kun käyttäjä painaa elementin kääntämiseen määritettyä näppäintä (Tässä tapauksessa R), kääntyy kyseessä oleva objekti 90 astetta y-akselin suuntaisesti.

```
 private void RotateObject()
 {
  if (Input.GetKeyDown(rotationHotkey))
  {
   currentPlaceableObject.transform.Rotate(0,90,0);
  }
 }
```

Elementin paikalle asettaminen hiirtä klikkaamalla

Kun käyttäjä klikkaa haluttuun kohtaan, elementti jää siihen.

```
private void ReleaseIfClicked()
{
 if (Input.GetMouseButtonDown(0))
 {
  currentPlaceableObject = null;
 }
}
```

Ground Placement Controller toimii nyt niin, että elementtejä pystyy käyttämään halutun tallirakennuksen seinien suunnitteluun. (Kuva 11.)

Kuva 11. Autotallin suunnittelua elementtejä asettelemalla.

Tarkemman lopputuloksen aikaansaamiseksi olisi hyvä, jos palat hakeutuisivat ankkuroituihin pisteisiin kartalla. Eli aina tasan siihen kohtaan, mihin edellinen elementti loppuu. Näin saisimme aikaiseksi mahdollisimman mittatarkan lopputuloksen ja voisimme lähettää luonnoksen mukana luonnossuunnittelijalle rakennuksen tarkkoja mittoja luonnossuunnitelua helpottamaan. Tätä ei kuitenkaan tässä kehitysprojektissa otettu huomioon. Lisäksi olisi erittäin hyvä, että käyttäjä voisi pyöritellä karttaa suunnitellessaan tallia, jolloin hän näkisi tallin jokaiselta mahdolliselta kantilta. Ei kuitenkaan ole suotavaa, että kameran voisi viedä maan alle. Tätäkään ei tässä kehitysprojektissa käyty läpi, mutta uskon sen olevan suhteellisen helposti toteutettavissa.

Tässä kehitysprojektissa ei myöskään yritetty asettaa suunnitellulle rakennukselle kattoa tai perustuksia. Eniten ongelmia ennustan tulevan sopivan katon valitsemisvaiheessa. Koska käyttäjä on saanut asetella vapaasti seinäpaloja kartalle, voi katto olla lähes minkä mallinen tahansa. Vertexissä on kehittynyt katon laskentaohjelma, joka seiniä osoittamalla ja kattokaltevuuden valitsemalla laskee tarvittavan katon ja rakentaa sen automaattisesti seinien päälle. Unityssä ei ole kuitenkaan tällaista ominaisuutta, joten Unityyn täytyisi tuoda yleisimpiä kattomalleja omina objekteinaan ja sitten toivoa, että asiakkaan suunnittelemaan kokonaisuuteen löytyy katto saatavilla olevista vaihtoehdoista. Autotallit ovat kuitenkin yleensä suorakaiteen muotoisia, eivätkä ne normaalisti kasva liian suuriin mittoihin. Olettaisin siis, että tuomalla noin parikymmentä perusmuotoista kattoa Unityyn, katamme noin kaksi kolmasosaa asiakkaiden suunnittelemista talliluonnoksista.

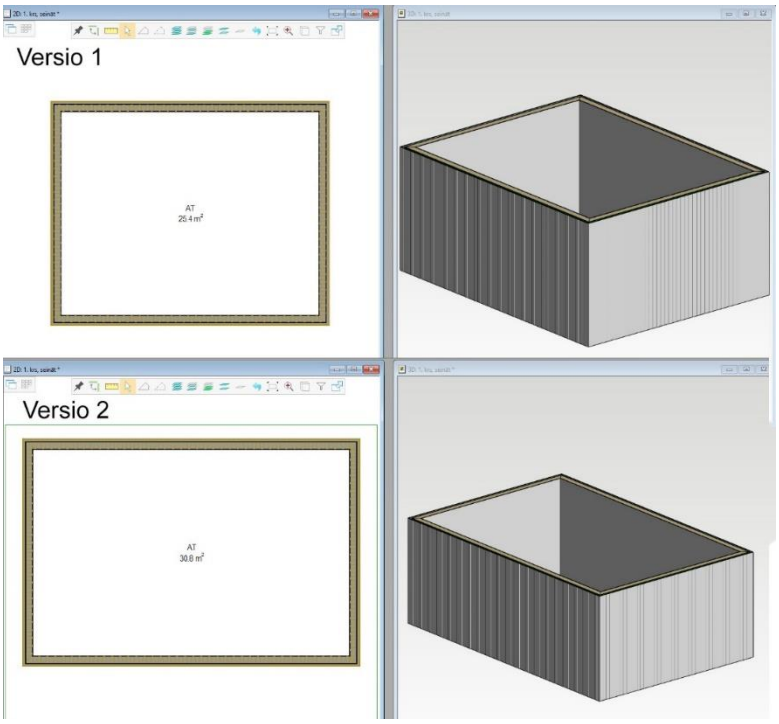
5.3 **Muokatun rakennuksen siirto Vertexiin**

Koska Vertexistä suoraan siirrettävään rakennukseen oli vaikeaa tehdä muutoksia Unityssä, lähestyn tätä kolmatta vaihetta hieman eri suunnasta. IFC-tiedoston avulla on mahdollista tuoda informaatiota Vertexiin, joten testaan pystyykö IFC-tiedostoa manuaalisesti muokkaamaan Vertexin ulkopuolella siten, että muutoksen saa näkyviin Vertexissä. Teen tämän vaiheen vertailututkimuksena.

5.3.1 IFC-tiedostojen vertailu

Vertex BD suunnitteluohjelmaa hyödyntämällä luon kaksi täysin identtistä rakennusta, joissa on vain yksi eroavaisuus. Tämä eroavaisuus on rakennuksen venyttäminen. Näiden kahden rakennuksen ominaisuudet tallennan kahdeksi erilliseksi IFC-tiedostoksi, joita vertailen VisualStudion vertailutyökalun avulla.

Kuvassa 12 on kaksi eri versiota pienelementtirakennuksesta. Näiden kahden rakennuksen ainoa eroavaisuus on, että versio 2 on 1200 mm versio 1:stä pidempi. Seinäelementtinä on käytetty ulkorakennuksen perusrakenteista seinää, jossa sisälevyn ja julkisivun vuorauksen välissä on eristettä ja tuulensuojalevyt. Katto, ikkunat ja ovet yms on jätetty pois, koska ne pidentäisivät IFC-tiedostoa ja tekisivät muutoksien kirjaamisesta paljon pidemmän prosessin. Kummastakin rakennuksen versiosta luotiin IFC-tiedosto vertailua varten. Vertailun tulokset on esitetty Taulukossa 1.



Kuva 12. 2D- ja 3D-seinäelementit Vertex suunnitteluohjelmassa

Taulukko 1. Vertailun tulokset, kun seinän pituutta on muutettu

Rivejä yhteensä	659
Rivejä, joissa muutoksia	72
Rivejä, joissa muutos on konekielinen	33
Rivejä, joissa muutos on annettu koordinaateissa	38

Muutokset koostuivat pääosin koordinaatteihin tehdyistä muutoksista ja epäselvistä konekielisistä nimikkeistä, jotka liittyivät usein rakennuksen eri materiaaleihin. Ei ole täyttä varmuutta siitä, vaikuttaako konekielinen osuus mihinkään, jos esimerkiksi rakennuksen materiaaleja haluaisi muuttaa. Tälläinen nimikkeeseen kohdistuva epäselvä muutos on esimerkiksi seuraavan kaltainen:

```
Versio 1
#210=IFCBUILDINGELEMENTPART('3E5HqQyMjDEv$kpGI1863Q',#5,'UTV_21x120_
PYSTY','WALLS/SIDING',$,#209,#208,$);

Versio 2
#210=IFCBUILDINGELEMENTPART('2D$oYXhOH7cBcE8dt1L_I1',#5,'UTV_21x120_
PYSTY','WALLS/SIDING',$,#209,#208,$);
```

Seinän pituutta ilmaisevat muutokset on ilmoitettu koordinaatteina Seuraavassa muodossa:

Versio 1 #212=IFCCARTESIANPOINT((5852.,0.));

Versio 2 #212=IFCCARTESIANPOINT((6052.,0.));

Koordinaateista on selkeästi havaittavissa 1200 mm muutos, joka rakennukselle on tehty. Tällainen rakennuksen koordinaatistoon kohdistuva muutos oli tehty jokaiselle materiaalille erikseen ja jokaista seinää kohden, ja kaikissa IFC-tiedoston koordinaatteja käsittelevissä riveissä oli havaittavissa sama 1200 mm muutos. Muutoksia oli tämän rakennuksen venyttämisen seurauksena syntynyt yhteensä 38 kappaletta.

Mikäli IFC-tiedoston koodiin muokattaisiin manuaalisesti uudet koordinaatin jokaiselle 38:lle riville, pystyisi seinää venyttämään myös manuaalisesti, ja se aukeaisi oikean näköisenä arkkitehtiohjelmassa. Täytyy kuitenkin ottaa huomioon, että jos seiniä venytellään moneen suuntaan ja monina eri osioina, on hyvin vaikeaa selvittää, mitä koordinaattia tulisi milloinkin muuttaa, koska koordinattejen selosteet ovat IFC-tiedostossa hankalasti esitetty viitauksina aikaisempiin riveihin.

6 JOHTOPÄÄTÖKSET

Unity vastaanottaa laajasti erilaisia objekteja ohjelman ulkopuolisista sovelluksista ja myös Vertexistä löytyy tiedostoformaatti, joka aukeaa Unityssä lähes mitään muutoksia tekemättä. Vertexissä mallinnetun rakennuksen voi viedä 3ds-formaattiin, joka on yhteensopiva Unityn kanssa. Kaikki erilliset rakennuksen osat eristemateriaaleista lähtien pysyvät omina aliobjekteinaan myös Unityyn siirron jälkeen. Tämä mahdollistaa sen, että jokaista rakennuksen osaa voi muokata erikseen Unityssä. Esimerkiksi pintamateriaaleja pystyy siis vaihtamaan vielä Unityyn siirron jälkeenkin. Yksi rakennusosien käsittelyä hankaloittava tekijä on se, että rakennusosien nimet ovat siirron jälkeen geneerisiä koodeja, joten aikaa menee jonkin verran siihen, että saa selville mikä materiaali sopii mihinkin rakennuksen osaan. Unityssä on kattava kokoelma erilaisia materiaaleja, jotka on valmiiksi käsitelty esimerkiksi läpäisemään valoa. Rakennuksen ikkunat voi esimerkiksi vaihtaa Unityssä sellaisiin, joista näkyy läpi ja jotka läpäisevät valoa.

Vertexissä on itsessään kehittynyt visualisointiominaisuus, joten jo Vertexillä pystyy saamaan aikaan näyttäviä markkinointimateriaaleja suunnitellusta rakennuksesta. Unityllä voi rakennuksen markkinoinnin viedä kuitenkin vielä astetta realistisempaan suuntaan. Rakennuksen voi upottaa maisemaan, joka on suunniteltu vastaamaan rakennuksen tulevaa ympäristöä ja Unityssä voi asettaa pelihahmon kulkemaan niin rakennuksen sisällä kuin ulkopuolellakin. Näin asiakas voi nähdä tulevan rakennuksensa juuri sellaisena, kuin se ehkä joskus omalla tontilla seisoo. Erityisesti rakennuksen valaistukseen ja ympäristöstä tulevan valaistuksen, kuten auringonkierron säätämiseen saa Unityssä kulumaan aikaa, sillä asetuksia on paljon erilaisia. Maisemointia voi Unityssä myös elävöittää lisäämällä tuulen vaikutuksen ympäristöön sekä esimerkiksi luonnosta kuuluvaa linnunlaulua projektiin. Tällaiset ominaisuudet ovat hyödyllisiä varsinkin suurissa rakennusprojekteissa, kuten esimerkiksi julkisia rakennuksia markkinoitaessa. Pienen rakennuksen kuten esimerkiksi autotallin sijoittaminen luonnolliseen ympäristöönsä, ei kuitenkaan välttämättä ole siihen käytetyn työaikapanoksen arvoista.

Rakennuksen muokkaaminen yhtenä kokonaisuutena osoittautui Unityssä haasteelliseksi. Rakennuksen osat ovat leikattu oikeisiin muotoihinsa Vertexissä, eikä niiden venyttäminen jälkikäteen Unityssä suju ongelmitta. Esimerkiksi seinän venytys venyttää ovien ja ikkunoiden mittoja siten, etteivät ne enää vastaa todellisuutta. Ikkunan siirtäminen toiseen paikkaan ei myöskään siirrä seinärakenteeseen leikattua reikää ikkunan mukana. Asiakas ei siis ainakaan kovin helposti pysty tekemään haluamiaan muutoksia Vertexissä suunniteltuun ja sitten Unityyn siirrettyyn projektiin. Luonnossuunnittelija haluaisi mielellään saada asiakkaalta palautetta tarvittavista muutoksista helposti, joten Unity-sovelluksen kömpelöt

muokkausominaisuudet tekevät eivät ainakaan nykymuodossaan vastaa rakennussuunnittelun tarpeita.

Unityllä voidaan kuitenkin toteuttaa sovellus, jolla asiakas pystyy suunnittelemaan toivomansa rakennuksen, käyttäen rakennuksen vakioelementtejä hyödykseen. Varsinkin seinien suunnittelu onnistuu sovellukselta helposti ja luonnossuunnittelijalle saadaan aikaiseksi suunnitellusta tallista ainakin pohjakuva ja julkisivukuvat sekä suunnitellun rakennuksen mitat. Katon asettaminen vapaasti suunniteltuun rakennukseen ei kuitenkaan ole yhtä yksinkertaista, sillä asiakkaan suunnittelema rakennus voi olla sen muotoinen, ettei siihen sovi yksikään Unityyn valmiiksi siirretty kattomalli. Katon venyttäminen taas rikkoisi halutun kattokaltevuuden. Tällainen suunnitteluohjelma on kuitenkin lähinnä sitä, mikä oli työni tavoite. Eli asiakas pystyy itse luonnostelemaan haluamansa rakennuksen ja siitä saadaan ainakin perusiformaatiota toimitettua luonnossuunnittelijalle.

IFC-tiedosto on monimutkainen kokonaisuus koodia, josta arkkitehdin suunnitteluohjelma ammentaa tietoa rakennuksesta. IFC-tiedostoon tallennetaan kaikki relevantti tieto rakennuksen eri osista ja niiden suhteista toisiinsa sekä rakennuksen osien tarkat paikat koordinaatteina. IFC-tiedosto luodaan yleensä suunnitteluohjelmassa ja sen pääasiallinen tarkoitus on siirtää rakennuksen tietoja suunnittelun eri työvaiheiden ja niihin liittyvien suunnitteluohjelmien välillä. Koodi on vaikeaselkoista, mutta koodia voi lukea ja muokata myös manuaalisesti.

Tämän tutkimukseni aikana selvisi, että IFC-tiedoston koodi on hyvin hankalasti manuaalisesti käsiteltävissä. IFC-tiedostot luo yleensä suunnitteluohjelma, ja niiden tarkoituksena on siirtää tietoa eri ohjelmien välillä. IFC-tiedosto on manuaalisesti luettavissa, mutta se on täynnä viittauksia ja erilaisia konekielisiä tunnisteita. Tutkimukseni tuloksena pystyin kuitenkin venyttämään rakennusta lisäämällä manuaalisesti uudet koordinaatit IFC-tiedostoon. Tekemäni muutos oli kuitenkin hyvin yksinkertainen ja mikäli muutoksia tehtäisiin paljon ja eri suunnissa yhtä aikaa, olisi IFC-tiedoston muokkaaminen manuaalisesti todella aikaa vievää ellei jopa mahdotonta. Mikäli rakennuksessa olisi ollut mukana kaikki sen osat kuten esimerkiksi katto, ikkunat ja ovet, olisi IFC-tiedoston koko kasvanut sekä myös viittaukset eri rakennusosien sijainteihin moninkertaistunut, joka olisi luonnollisesti vaikeuttanut rakennuksen venytysyritystä.

Sellaisen ohjelman luominen, jolla asiakas voisi kännykällään suunnitella itselleen pienen rakennuksen ja sen tuloksena lähettämään arkkitehdille IFC-tiedoston olisi kyllä kätevää, mutta sellaisen toteuttaminen IFC-tiedostoja hyödyntämällä vaatii paljon erityisosaamista ja resursseja. Jotta rakennuksen voisi ulkopuolisesti muokatun IFC-tiedoston avulla avata suunnitteluohjelmassa, vaatii se ohjelman tekijältä paljon perehtymistä IFC-tiedoston muodostumiseen ja erilaisten muutosten tekemisen

vaikutukseen IFC-tiedostossa. Kaikesta perehtymisestäkään huolimatta, ei IFC-tiedoston luominen ulkopuolisesta ohjelmasta ole helppoa, sillä suunnitteluohjelmien nimikkeet ovat usein salattuja ja siksi ulkopuoliselle taholle käsittämättömiä.

Unityä pystyy jossakin määrin hyödyntämään rakennussuunnittelussa. Tarvitaan kuitenkin vielä paljon kehitystä rakennusosien muokattavuuden lähtökohdista, että Unityä voisi täysimittaisesti soveltaa rakennussuunnitteluun. Seiniä pitäisi pystyä muokkaamaan kulmapiste kerrallaan, jotta esileikatut aukot voisi muuttaa siirtyvän elementin osan mukana. Suunnitellun rakennuksen markkinointiin Unity on mitä parhain apuväline, jolla saadaan taatusti realistisin kuva tulevasta rakennuksesta asiakkaan käyttöön. Rakennuksen muokkaus Unityssä on kuitenkin vielä hyvin kömpelöä. Mikäli sovelluksessa pyritään kokonaisten rakennusten sijasta hyödyntämään rakennksen vakioelementtejä, voidaan aikaansaada rakennussuunnittelua helpottava luonnostelusovellus, jolla asiakas voi havainnollistaa omia ajatuksiaan tavoitteena olevasta rakennuksesta.

Tiedonsiirto takaisin Vertexiin on mahdollista IFC-tiedoston avulla, mutta käytännössä se vaatii paljon perehtymistä ja aikaa, jotta luonnossuunnittelija voisi Vertexillä avata asiakkaan Unityssä suunnitteleman rakennuksen. Asiakas voi siis suunnitella oman rakennuksensa Unity 3D:llä, mutta sen tuominen Vertexiin luonnossuunnittelijan työpöydälle ei ole vielä mahdollista. Luonnossuunnitelijalle voidaan kuitenkin lähettää muuta tietoa ja materiaalia asiakkaan suunnittelemasta rakennuksesta, jotta hänen työnsä helpottuisi.

LÄHTEET

Buyuksalih, I., Bayburt, S., Buyuksalih, G., Baskaraca, A. P., Karim, H. & Rahman, A. A. 2017. 3D Modelling and Visualization based on the Unity Game Engine – Advantages and Challenges. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences IV-4/W4. Viitattu 18.6.2018 https://doi.org/10.5194/isprs-annals-IV-4-W4-161-2017

Collier, D. 1993. The Comparative Method. Political Science: The State of the Discipline II. 105-107. Viitattu 4.4.2018. http://polisci.berkeley.edu/sites/default/files/people/u3827/APSA-TheComparativeMethod.pdf

Creating and Using Scripts. 2018. Unity Technologies. Viitattu 16.7.2018. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

Game engines – how do they work? 2018. Unity Technologies. Viitattu 18.6.2018. https://unity3d.com/what-is-a-game-engine

Height Tools. 2018. Unity Technologies. Viitattu 2.7.2018. https://docs.unity3d.com/Manual/terrain-Height.html

Hougaard, K. 2014. Unity and Architecture. Unity Blog Viitattu 14.7.2018. https://blogs.unity3d.com/2014/03/06/unity-and-architecture/

Laakso, M & Nyman, L. 2016. Exploring the Relationship between Research and BIM Standardization: A Systematic Mapping of Early Studies on the IFC Standard (1997-2007). Buildings (6), 1-23. Viitattu 12.3.2018. http://dx.doi.org/10.3390/buildings6010007

Lighting overview. 2018. Unity Technologies. Viitattu 10.7.2018. https://docs.unity3d.com/Manual/LightingInUnity.html

Lightmapping Quickstart. 2017. Unity Technologies. Viitattu 10.7.2018. https://docs.unity3d.com/2017.1/Documentation/Manual/Lightmapping.html

Miettinen, R & Paavola, S. 2014. Beyond the BIM utopia: Approaches to the development and implementation of building information modelling. Automation in Construction (43), 84-91. Viitattu 10.3.2018. http://dx.doi.org/10.1016/j.autcon.2014.03.009

Model file formats. 2018. Unity Technologies. Viitattu 2.7.2018. https://docs.unity3d.com/Manual/3D-formats.html

Ohjelmistot teollisuuden tarpeisiin. 40 vuoden kokemuksella. Vertex Uutiset 1/2018. Viitattu 3.7.2018. http://mag.vertex.fi/Vertex++Uutiset+12018/13/40+vuoden+kokemuksella

Peysbubby. 2016. 1st Answer to mouse wheel scroll weapon switch. Viitattu 27.7.2018. https://answers.unity.com/questions/1148249/mouse-wheel-scroll-weapon-switch.html

Ralph. 2017. What is the Unity Asset Store and how do I purchase Assets? Unity Technologies. Viitattu 25.6.2018. https://support.unity3d.com/hc/en-us/articles/210142503-What-is-the-Unity-Asset-Store-and-how-do-I-purchase-Assets-

The world's leading content-creation engine. 2018. Unity Technologies. Viitattu 20.6.2018. https://unity3d.com/unity

Tutkimus- ja kehittämistoiminta - Käsitteet. 2018. Tilastokeskus. Viitattu 20.4.2018. https://www.stat.fi/meta/kas/t_ktoiminta.html

United they stand. 2009. MCV The Business of Video Games. Future Publishing Limited Quay House. Viitattu 20.6.2018. https://www.mcvuk.com/development/united-they-stand

Unity Personal. 2018. Unity Technologies. Viitattu 20.6.2018. https://store.unity.com/products/unity-personal

Vertex BD yleiskatsaus. 2018. Vertex Systems Oy. Viitattu 10.3.2018. https://kb.vertex.fi/bd2018fi/vertex-bd-yleiskatsaus

Weimann, J. 2017. How to: Create a Unity3D building placement system for RTS or City builders – Let your player place a 3D object in the world. Viitattu 26.7.2018. https://unity3d.college/2017/08/02/how-to-create-a-unity3d-building-placement-system-for-rts-or-city-builders-let-your-player-place-a-3d-object-in-the-world/

What is IFC and what do you need to know about it? 2016. Areo blog. Viitattu 20.3.2018. http://blog.areo.io/what-is-ifc/

Yan, W., Culp, C. & Graf, R. 2010. Integrating BIM and gaming for real-time interactive architectural visualization. Automation in Construction 20 (2011) 446-458. Viitattu 20.6.2018 https://doi.org/10.1016/j.autcon.2010.11.013