Afshin Safari

Työkalu glTF-3D-mallien tuontiin ja visualisointiin pelimoottorissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

23.11.2021

Tiivistelmä

Tekijä:	Afshin Safari
Otsikko:	Työkalu glTF-3D-mallien tuontiin ja visualisointiin pelimoottorissa
Sivumäärä:	41 sivua
Aika:	23.11.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaajat:	Lehtori Antti Laiho Teknologiajohtaja Juhani Korpinen

---

Opinnäytetyön tavoitteena oli tutkia ja laajentaa Unreal Engine 4 -pelimoottoria toteuttamalla siihen lisäosa. Lisäosa tehtiin helsinkiläiselle yritykselle, joka tarjoaa asiakkailleen rakennusarkkitehtuuriin liittyviä työkaluja. Lisäosan oli tarkoitus automatisoida yrityksen glTF-3D-mallien tuontiprosessi Unreal Engine 4 -pelimoottoriin.

Opinnäytetyön tutkimusosuudessa tutustuttiin Unreal Engine 4 -pelimoottorin ominaisuuksiin ja työkaluihin, Unrealin C++-, Python- ja Blueprint-ohjelmointirajapintoihin, Unreal Editorin laajentamiseen ja Unreal-lisäosien kehittämiseen. Tutustumisen jälkeen aloitettiin projektiosuus suunnittelemalla lisäosan eri komponentit ja sovittiin lisäosan vaatimusmäärittelystä. Lisäosan vaatimuksena oli yrityksen glTF-3D-mallien lataaminen, niiden tuominen ja visualisoiminen pelimoottorissa. Nämä vaatimukset asetettiin myös arvioitavaksi osaksi projektin onnistumisen määrittämiseen.

Suunnitteluvaiheen jälkeen siirryttiin toteutusvaiheeseen, jossa toteutettiin lisäosan komponentit Unreal Engine 4 -pelimoottoriin. Lisäosa toteutettiin C++-, Python- ja Blueprint-ohjelmointikielillä.

Työn tuloksena saatiin asetettujen vaatimusten mukaan toimiva lisäosa, jota käytettiin tuomaan ja visualisoimaan yrityksen glTF-3D-malleja Unreal Engine 4 -pelimoottorissa.

Avainsanat: Unreal Engine 4, Unreal Editorin laajentaminen, glTF, 3D

Abstract

Author:	Afshin Safari
Title:	Tool for importing and visualizing glTF 3D models in game engine
Number of Pages:	41 pages
Date:	23 November 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Game Applications
Supervisors:	Antti Laiho, Senior Lecturer Juhani Korpinen, CTO

The goal of this thesis was to study and extend the Unreal Engine 4 game engine by implementing a plugin. The plugin was created for a company based in Helsinki, which is offering its customers tools related to building architecture. The purpose of the plugin was to automate the company's glTF 3D models import process into the Unreal Engine 4 game engine.

The research part of this thesis was focused primarily on getting familiar with Unreal Engine 4 game engine's features and tools, Unreal's C++, Python and Blueprint APIs, extending Unreal Editor and developing Unreal plugins. After the introduction, the project part was started by designing the various components of the plugin and agreeing on specific requirements of the plugin. Requirements for the plugin were to download the company's glTF-3D models, import, and visualize them in the game engine. These requirements were also set as an evaluable part to determine the success of the project.

After the research, the project part was started by designing the various components of the plugin. After the design phase, we moved onto the implementation phase, where we implemented the plugin to the Unreal Engine 4 game engine. The plugin was implemented with C++, Python and Blueprint programming languages.

As a result of this work, all set requirements were met and the plugin was used to import and visualize the company's glTF 3D models in the Unreal Engine 4 game engine.

Keywords: Unreal Engine 4, Extending Unreal Editor, glTF, 3D

Sisällys

Lyhenteet

JSON: JavaScript Object Notation. Avoimen standardin tiedostomuoto tiedonvälitykseen.

REST: Representational State Transfer. HTTP-protokollaan perustuva arkkitehtuurimalli, joka on tarkoitettu REST-ohjelmointirajapintojen toteuttamiseen.

HTTP: Hyper Text Transfer Protocol. Protokolla, jota selaimet käyttävät tiedonsiirtoon.

URL: Uniform Resource Locator. Sitä käytetään tunnistamaan tiedoston sijainti internetissä.

API: Application Programming Interface. Sovellusohjelmointirajapinta, jonka avulla sovellukset voivat keskustella toistensa kanssa.

glTF: Graphics Language Transmission Format. Khornos-ryhmän kehittämä avoin tiedonsiirtoformaatti.

Blueprint: Unreal Engine 4 -pelimoottorin käyttämä visuaalisen ohjelmoinnin järjestelmä.

Plugin: Lisäosa, jonka avulla lisätään tai muokataan ohjelman ominaisuuksia.

3D Kolmiulotteisuus. Jotain, jolla on leveys, korkeus ja syvyys. X-, Y- ja Z-suunnat.

1 Johdanto

Lisäosaksi sanotaan ohjelmistoa, joka sisältää tiettyjä toimintoja. Lisäosien avulla lisätään uusia toimintoja olemassa oleviin ohjelmiin ja laajennetaan niiden ominaisuuksia. Lisäosilla parannetaan ohjelmien käyttökokemusta muokkaamatta niiden lähdekoodia.

Opinnäytetyön tavoitteena on laajentaa Unreal Engine 4 -pelimoottoria toteuttamalla siihen lisäosa, jonka avulla voitaisiin automatisoida Tridify Oy:n glTF-3D-mallien tuontiprosessi Unreal Engine 4 -pelimoottoriin.

Tridify on suomalainen ohjelmistoyritys, joka tarjoaa asiakkailleen rakennusarkkitehtuuriin liittyviä työkaluja, jotka helpottavat asiakkaiden töiden tuottavuutta ja luovuutta. Esimerkiksi Tridify BIM Viewer on yrityksen kehittämä työkalu, joka mahdollistaa 3D-BIM-mallien tarkastelemisen selainympäristössä ilman erikseen asennettavia sovelluksia.

Opinnäytetyön alkuosassa käydään läpi lisäosan kehittämisessä käytetyt työkalut ja teknologiat ja tutustutaan Unreal Engine 4 -pelimoottoriin, Unreal Editorin laajentamiseen ja Unreal-lisäosien kehittämiseen. Tutustumisen jälkeen laajennetaan Unreal Engine 4 -pelimoottoria toteuttamalla siihen lisäosa.

2 Lisäosan kehityksessä käytetyt teknologiat ja työkalut

Unreal Engine 4 -pelimoottoriin tehdyn lisäosan kehityksessä käytettiin monia eri teknologioita ja työkaluja. Tässä luvussa käydään lyhyesti, mutta kuitenkin tarpeeksi tarkasti läpi käytetyt teknologiat ja työkalut.

2.1 Unreal Engine 4 -pelimoottori

Unreal Engine 4 on suosittu ja laajalti käytetty ilmainen pelimoottori, jonka Epic Games on kehittänyt. Se julkaistiin vuonna 2014, ja se on viimein neljännen sukupolven versio, joka yhdistää fysiikkaperusteiset renderöintimateriaalit (Physically based rendering, PBR), säteenseurantatuen (Ray Tracing Texel eXtreme, RTX), valaistuksen, heijastukset sekä erilaisia kehitystyökaluja, joilla voidaan kehittää pelejä, sovelluksia ja elokuvia. [1.] Unreal Engine 4:llä tehtyjä suosittuja pelejä ovat mm. Fortnite, Gears 5, ARK: Survival Evolved, Conan Exiles ja Hellblade: Senua’s Sacrifice. [2.]

Unreal Engine 4:llä on mahdollista kehittää sovelluksia useille eri alustoille, kuten PC, nykyiset pelikonsolit (PlayStation, Nintendo Switch, Xbox One), AR/VR-laitteet (Oculus Rift, Google Daydream, PlayStation VR) ja iOS/Android-mobiililaitteet. Tämä on osa syytä siihen, että pelikehittäjät käyttävät sitä niin laajalti AAA-pelien tekemiseen. Pelikehittäjien lisäksi myös tuotesuunnittelijat ja arkkitehtuurit käyttävät Unrealia sen tarjoaman fotorealistisen grafiikan vuoksi. [1.]

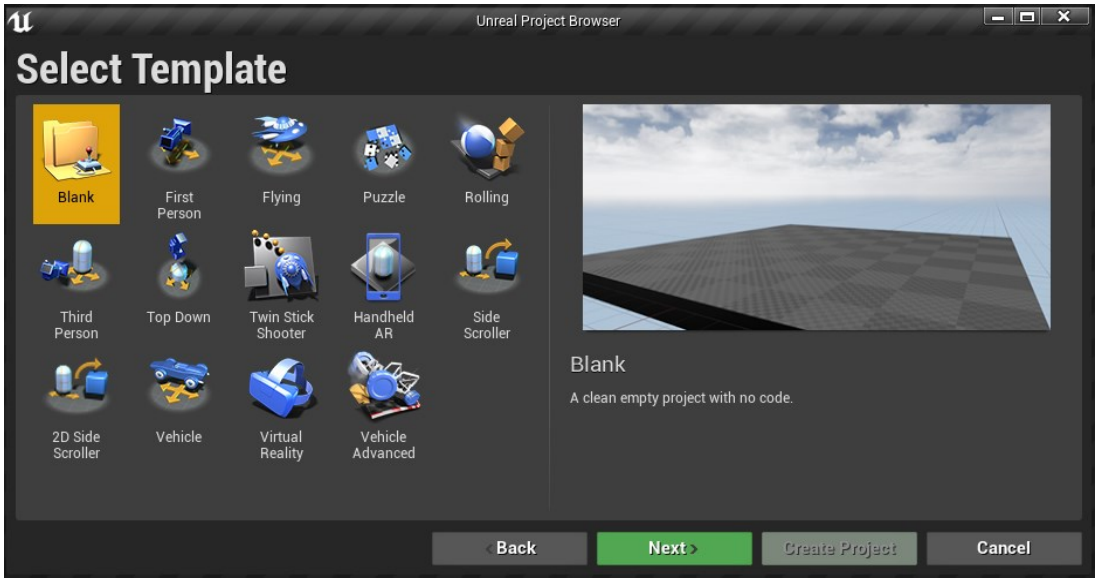
Unreal Engine 4:ssä on open source- eli avoin lähdekoodi, ja tämä tekee siitä todella joustavan, koska kuka tahansa voi tarkastella ja muokata kaikki sen ominaisuudet. [1.] Unreal Engine 4:llä sovellusten kehitys tapahtuu ylipäätänsä C++-ohjelmointikielellä, mutta C++-ohjelmointirajapinnan lisäksi pelimoottori tarjoaa Blueprint-nimisen visuaalisen ”node-based” eli solmupohjaisen ohjelmointijärjestelmän, jolla voidaan kehittää pelejä tai nopeita prototyyppejä kirjoittamatta yhtään C++-riviä koodia. [1.] Unrealissa C++:n ja Blueprintin lisäksi voidaan käyttää Python-ohjelmointikieltä, mutta Pythonia käytetään enemmän aikaa vievien editoritoimintojen automatisointiin.

Unreal Engine 4:llä on myös online-alijärjestelmä, joka tarjoaa peleille mahdollisuuden integroida erilaisia toimintoja Steam-alustalle tai Xbox Network -palveluun, kuten pelin sisäisten ostosten tekeminen, saavutukset ja tulostaulu. [3.]

Tehosta ja monimutkaisuudesta huolimatta Unreal Engine 4 tarjoaa modernin käyttöliittymän ja erinomaiset työkalut, joilla voidaan luoda visualisointeja, simulaatioita ja hämmästyttäviä pelejä.

Unreal-projektin luominen

Jotta voidaan luoda uusi Unreal-projekti, täytyy asentaa Epic Games -käynnistysohjelma koneelle. Epic Games -käynnistysohjelman kautta voidaan asentaa koneelle haluttu Unreal Engine 4 -pelimoottorin versio. Tätä projektia aloittaessa Unreal Engine 4:n uusin versio oli 4.22. Kuvassa 1 näkyy Unreal Editorin projektiselaimen perusnäkymä, jonka avulla luodaan uusi projekti.



Kuva 1. Unreal Editorin projektiselaimen perusnäkymä.

Unreal Engine tarjoaa useita eri valmiita projektipohjia. Projektia luotaessa valitaan, onko valittu projekti toteutettu käyttäen C++-ohjelmointikieltä vai Blueprintejä. Valinnasta riippumatta voidaan tarvittaessa kuitenkin käyttää Blueprintejä ja C++-ohjelmointikieltä sekaisin keskenään.

2.2 Unreal Engine 4:n työkalut ja editorit

Unreal Engine 4 koostuu monista eri työkaluista ja editoreista, kuten taso- (Level), materiaali-, fysiikka-, Blueprint- ja UMG-editorit, joissa voidaan hallinnoida pelialueen tasoa, luoda materiaaleja ja käyttöliittymiä sekä lisätä peliin toiminnallisuuksia. [4.] Tässä projektissa käytettiin pääasiassa Blueprint- ja UMG-editoreja lisäosan käyttöliittymän tekemiseen.

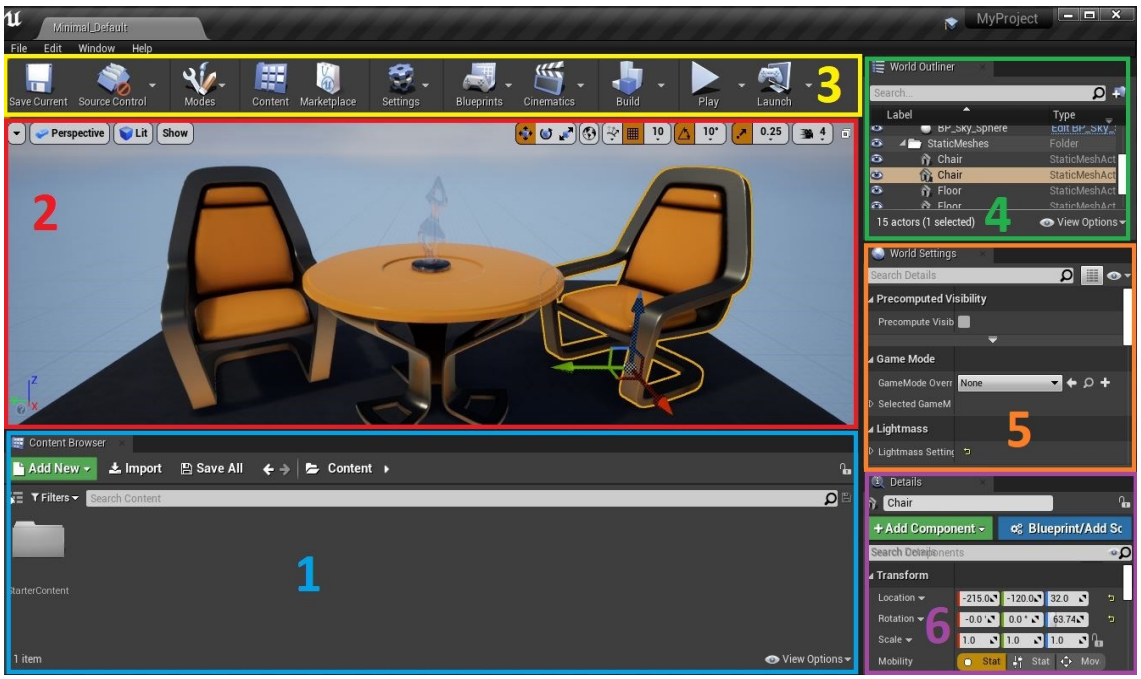
Level Editor

Unreal Engine 4 tarjoaa visuaalisen käyttöliittymän nimeltä Level-editori. Se on Unreal Enginen käynnistymiseditori, joka koostuu eri näkymistä (taulukko 1). [5.]

Taulukko 1. Level-editorin näkymät [5].

#	Näkymän nimi	Kuvaus
1	Content browser	Sisältöselain, jonka avulla voidaan lisätä ja muokata projektin tiedostoja.
2	Viewport	Näyttöalue, josta näkyy auki oleva kenttä.
3	Toolbar	Päätyökalupalkki, josta löytyvät editorin yleisimmin käytetyt toimintopainikkeet.
4	World Outliner	Näyttöalue, josta näkyvät kentässä olevat objektit.
5	World Settings	Näyttöalue, josta voidaan muokata kenttään liittyvät asetukset.
6	Details	Näyttöalue, josta voidaan nähdä ja muokata nykyistä valintaa koskevat tiedot.

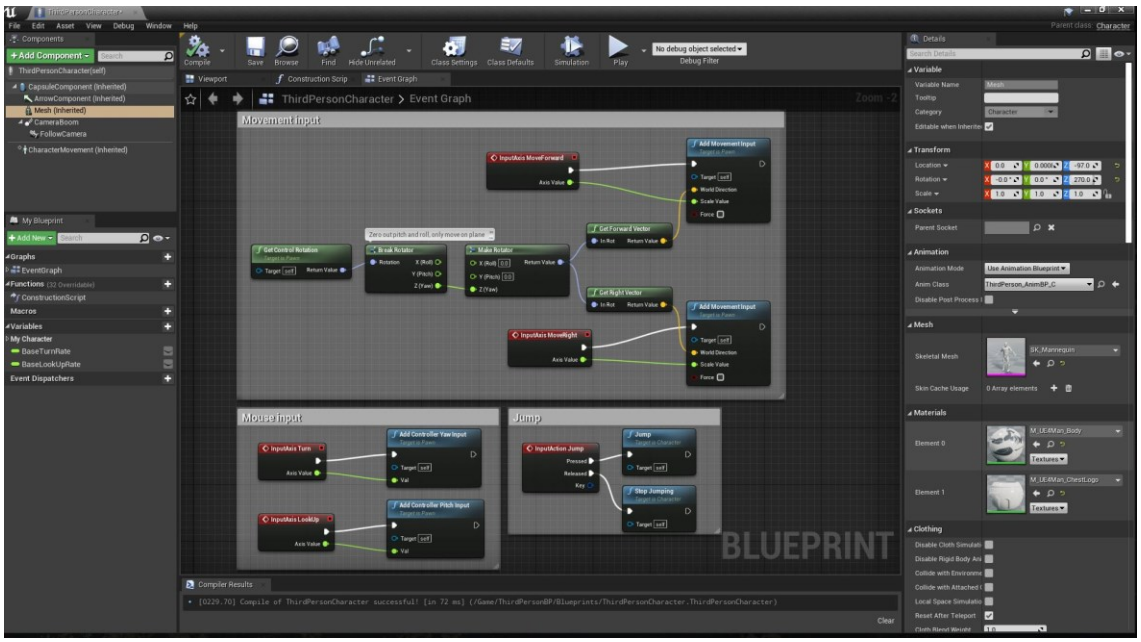
Level-editorin avulla päästään muihin tarvittaviin työkaluihin ja editoreihin. Kuva 2 esittää Level-editorin eri näkymät numeroituna.



Kuva 2. Level-editori.

Blueprint Editor

Blueprint-editori on yksinkertaisesti solmupohjainen kaavioeditori (kuva 3), jonka avulla luodaan ja muokataan Blueprintilla tehtyjä pelitoiminnallisuuksia. Blueprint-editori koostuu useista eri paneeleista, kuten Components- ja My Blueprint -paneelit. Components-paneelista voidaan lisätä erilaisia komponentteja, kuten esimerkiksi CharacterMovement-komponentti, joka on tarkoitettu pelihahmon liikkumiseen. My Blueprint -paneelista voidaan lisätä ja muokata Blueprintissä toteutettavia funktioita ja muuttujia. [6.]



Kuva 3. Blueprint-editorin tapahtumagraafi.

Blueprint-editorissa löytyy myös tapahtumagraafi (Event Graph), joka on Blueprint-editorin koodialue. Tapahtumagraafin kautta lisätään, muokataan ja poistetaan solmuja. Solmuja lisätään tapahtumagraafista (kuva 4). [7.] Blueprint-editori myös tarjoaa erilaisia virheenkorjaus- ja analysointityökaluja, joita voidaan käyttää Blueprintien optimoinnissa ja virheenkorjauksissa. [8.]

Kuva 4. Lista kaikista käytettävissä olevista solmuista.

UMG Editor

UMG-editori eli Unreal Motion Graphics UI Designer on visuaalinen editori, jonka avulla voidaan helposti luoda käyttöliittymiä. UMG-editori tarjoaa kahta eri pienoisohjelmatyyppiä (Widget), Editor Utility Widget ja Widget Blueprint (taulukko 2). [9.] Jokainen Widget-tyyppi on tarkoitettu tiettyyn käyttöliittymään tekemiseen. Esimerkiksi lisäosan käyttöliittymän tekemiseen käytettiin Editor Utility Widget -tyyppiä, koska sillä voidaan laajentaa ja muokata Unreal Editoria.

Taulukko 2. UMG-editorin Widget-tyypit [9].

Widegtin nimi	**Kuvaus**
Widget Blueprint	Käytetään pelin sisäisten valikoiden ja käyttöliittymien tekemiseen ja muokkaamiseen.
Editor Utility Widget	Käytetään Unreal Editorin laajentamiseen ja muokkaamiseen.

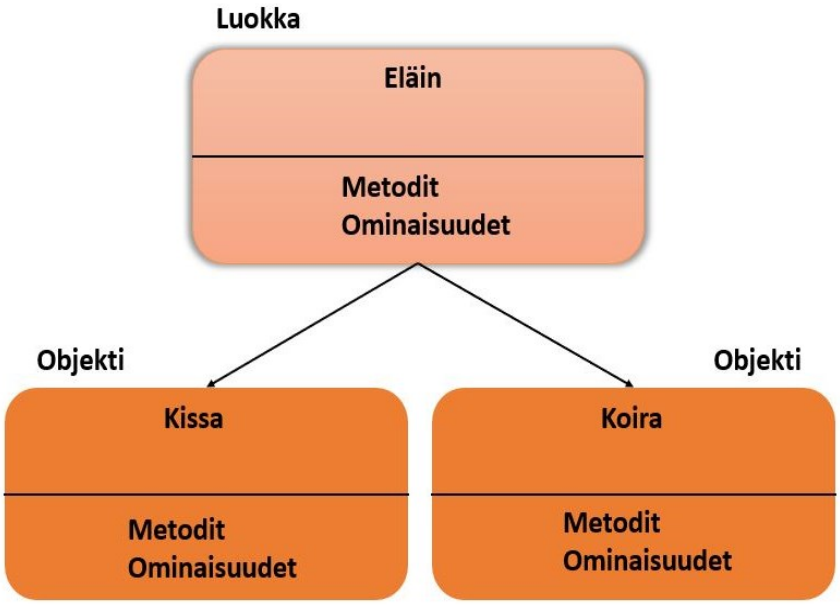
UMG-editorissa löytyy kaksi välilehteä, Designer ja Graph (kuva 5). Designer-välilehdellä suunnitellaan käyttöliittymän visuaalista ulkoasua ja lisätään käyttöliittymän elementtejä, kuten napit ja tekstit, jotka löytyvät Paletti-paneelista. Graph-välilehdellä taas ohjelmoidaan käyttöliittymään lisättyjen elementtien toiminnot. [9.]

Kuva 5. UMG-editori.

2.3 Unreal C++ -ohjelmointirajapinta

C++ on tehokas keskitason ohjelmointikieli, jota voidaan käyttää mm. käyttöjärjestelmien, sovelluksien ja pelien tekemiseen. C++-ohjelmointikieli perustuu C-kieleen, johon on lisätty uusia ominaisuuksia. C++ tukee erilaisia ohjelmointitappoja, kuten funktionaalista ja olio-ohjelmointia. Tämä tekee siitä tehokkaan. C++ on suosittu ohjelmointikieli suorituskykykriittisissä sovelluksissa, joissa nopeus ja muistinhallinta on tärkeää. [10.] Tämän takia yleensä pelimoottoreita tai simulointiohjelmistoja tehdään C++-ohjelmointikielellä, koska tällaisissa ohjelmistoissa muistinhallinta ja suorituskyky on todella tärkeä.

C++ on olio-orientoitunut ohjelmointikieli. Olio-ohjelmointi on ohjelmointiparadigma, jonka tarkoitus on tehdä koodista uudelleenkäytettävämpi, tehokkaampi ja modulaarisempi. Olio-ohjelmointi perustuu luokkiin ja objekteihin. Luodaan luokkia ja luoduista luokista tehdään ilmentymiä (instance) eli olioita (objects) (kuva 6). Luokka sisältää jäsenmuuttujia (member variables) ja jäsenfunktioita (member functions), jotka määrittelevät olioiden tietosisällöt ja toiminnot. [11.]



Kuva 6. Eläin-luokka, josta luodaan kaksi instanssia eli objektia.

Olio-ohjelmointi sisältää erilaisia periaatteita, kuten kapselointi- ja perintöperiaate. Kapselointiperiaatteella kapseloidaan eli piilotetaan käyttäjältä luokan yksityiskohdat luokan sisälle käyttämällä ”private”- ja ”protected”-näkyvyysmääritteitä. Perintöperiaatteella taas tarkoitetaan sitä, että uudet luokat perivät kantaluokkien tiedot ja ominaisuudet. Luokkaa, joka peri kantaluokan ominaisuudet, kutsutaan aliluokaksi. [11.]

Unreal Engine 4 on kokonaan tehty C++-ohjelmointikielellä. Unreal Enginessä natiivi lähestymistapa sovellusten kehittämiseen on C++:lla. Sitä käytetään pelin sisäisen logiikan tekemiseen, työnkulkujen yksinkertaistamiseen ja kehitysputken parantamiseen. Koska Unreal Enginen lähdekoodi on avoin, käyttämällä C++:aa voidaan myös tehdä muutoksia Unreal Editoriin ja laajentaa Editoria lisäosilla.

Unreal Enginessä C++:lla ohjelmointi eroaa hieman tavallisesta C++-ohjelmoinnista. Unrealilla on oma C++-ohjelmointirajapinta ja se tarjoaa useita eri ”makroja” ja kantaluokkia. Unrealissa uusia luokkia luodaan ja peritään yleensä

jostain Unrealin olemassa olevista kantaluokista. Kantaluokkatyypit ovat UObject, UStruct, AActor ja AActorComponent (taulukko 3). [12.]

Taulukko 3. Unrealin kantaluokkatyypit [12].

Luokan nimi	**Kuvaus**
UObject	Kaikkien Unrealissa olevien objektien perusluokka, joka tarjoaa useita moottorin tärkeimpiä toimintoja, kuten automaattinen roskienkeräys ja ominaisuuksien serialisointi.
UStruct	Tavallisen tietotyypin perusluokka, jota käytetään järjestämään erilaisia ominaisuuksia.
AActor	Kaikkien niiden peliolioiden perusluokka, joita voidaan sijoittaa pelimaailmaan.
AActorComponent	Perusluokka, joka lisätään AActoriin, kuten kameran perspektiivit ja fysikaaliset vuorovaikutukset.

Unrealissa voidaan myös luoda luokkia, jotka eivät peri näistä kantaluokista, mutta silloin luodut luokat eivät hyödy Unreal Engineen sisään rakennetuista ominaisuuksista. Esimerkkejä Unreal Engineen sisään rakennetuista ominaisuuksista ovat Unrealin automaattinen roskienkeräys ja muistin hallinta. [12.]

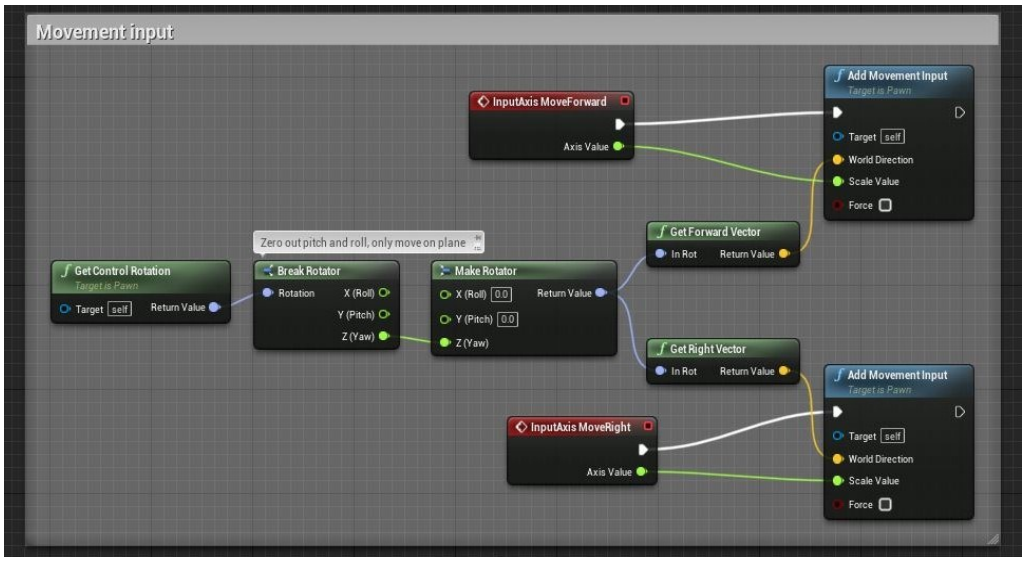
Unrealissa C++-luokkia luodaan lisäämällä niihin Unreal-makroja. Esimerkiksi kun tavalliseen luotuun C++-luokkaan lisätään UCLASS-makro, se muuttaa sen tavallisen C++-luokan Unreal Engine UCLASS -luokaksi. UCLASS-luokka on periaatteessa yksinkertainen C++-luokka, johon on lisätty UCLASS-makro ja joitakin header-tiedostoja, jotka mahdollistavat tavallisen luokan integroinnin Unreal Engineen. Käyttämällä Unreal-makroja ja kantaluokkia varmistetaan, että luotujen luokkien olioiden muistit on hallittu oikein ja ne ovat suunnittelijoiden ja taiteilijoiden käytettävissä. Tällä tavalla suunnittelijat ja taiteilijat voivat hyödyntää ohjelmoijien luomia olioita ja lisätä niihin uusia toimintoja Blueprintillä. [12.] Unrealin makroista kerrotaan tarkemmin luvussa 2.6 Blueprintin ja C++:n välinen kommunikaatio.

Unrealin C++-ohjelmointirajapinnan hyvä puoli Blueprintiin nähden on sen suorituskyky. C++ on huomattavasti nopeampi kuin Blueprint, ja C++:lla voidaan muokata ja laajentaa Unreal Editoria. C++:lla suurten projektien hallinta on myös helpompaa. [13.]

2.4 Blueprint- visuaalinen ohjelmointikieli

Blueprint on Unreal Engine 4:n visuaalinen ohjelmointikieli, joka tarjoaa intuitiivisen, solmupohjaisen (node-based) käyttöjärjestelmän. Blueprintillä voidaan toteuttaa pelitoiminnallisuuksia, peliprototyyppejä ja jopa kokonaisia pelejä ilman yhtäkään riviä koodia. [14.]

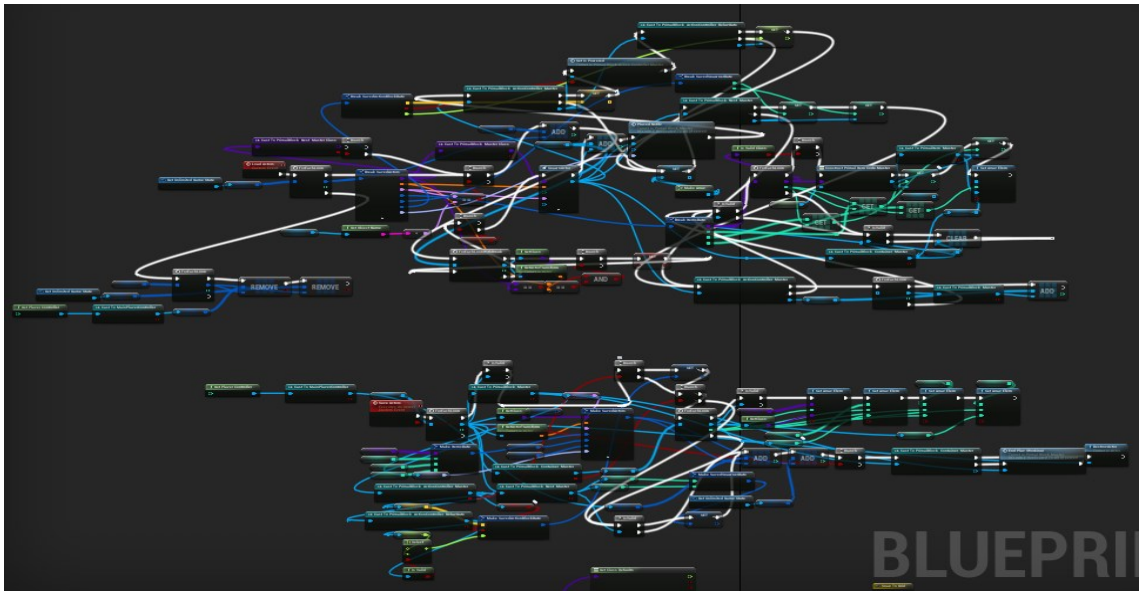
Blueprintillä tehdään toiminnallisuuksia yhdistelemällä valmiiksi tehtyjä solmuja (kuva 7). Solmut koostuvat eri toiminnoista ja tapahtumista, joita on rakennettu Unreal Engineen sisään. Blueprintiin voidaan luoda myös omia solmuja. Omia solmuja luodaan Unreal Engineen sisään rakennetuista solmuista tai C++-ohjelmointirajapinnan kautta luoduista toiminnoista. Blueprint käyttää pitkälti C++-ohjelmointirajapinnan ominaisuuksia, joten voidaan yhdistää Blueprint ja C++-ohjelmointirajapinta keskenään. Tämä antaa mahdollisuuden suunnittelijoille ja taitelijoille käyttää ja laajentaa ohjelmoiden C++:lla luotuja toimintoja Blueprintillä. [14.]



Kuva 7. Blueprintillä luotua koodia, joka säätelee pelaajan liikkumista.

Blueprintin käytön tärkeimpiä etuja C++-ohjelmointirajapintaan nähden on se, että Blueprintillä on nopeampi toteuttaa pelitoiminnallisuuksia. Blueprintillä hyödynnetään ennalta Unreal Engineen sisään rakennettuja solmuja, joiden yhdistäminen vie usein vähemmän aikaa kuin koodin kirjoittaminen ja sen kääntäminen. Blueprint on myös paljon yksinkertaisempaa kuin C++, ja sitä on paljon helpompi oppia. Tämän takia sitä yleensä suositellaan uusille pelikehittäjille, joilla ei ole aiempaa ohjelmointitaitoa ja jotka ovat vasta aloittamassa Unreal Enginellä pelien tekemisen.

Vaikka Blueprintillä on nopeampi toteuttaa toiminnallisuuksia kuin C++-ohjelmointirajapinnalla, Blueprintillä toteutetuista toiminnallisuuksista voi helposti syntyä iso sotku (kuva 8), ellei olla huolellisia toiminnallisuuksien toteutuksissa. Blueprintillä monimutkaisten toiminnallisuuksien toteuttaminen usein luo valtavan määrään solmuja, ja tämä vaikeuttaa Blueprintin sisällön lukemista ja ymmärtämistä. [15.]



Kuva 8. Sotkuinen Blueprint [16].

Onneksi Blueprint-editori tarjoaa eri menetelmiä ja työkaluja, joilla voidaan vähentää Blueprintin sisällön sotkua ja pitää sisältö puhtaampana ja luettavampana. Esimerkiksi funktio- ja makrotyökalut on tehty juuri tätä varten. Jos toteutus on toistuva, kannattaa luoda siitä funktio tai makro. Tällä tavalla voidaan

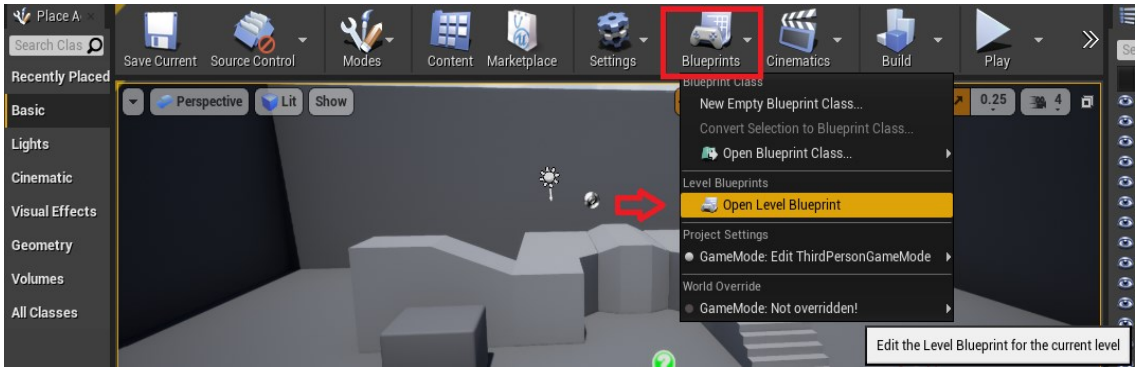
parantaa koodin laatua ja välttää koodin toistumista. Koodin kopiointi vähentää aina luettavuutta ja lisää työtä. Muita menetelmiä, joita voidaan käyttää Blueprintin sisällön luettavuuden ja laadun parantamiseksi, ovat esimerkiksi funktioiden, makrojen ja muuttujien kategorointi tai solmujen kommentointi. [17.]

Blueprint-tyypit

Blueprint-tyypit koostuvat neljästä päätyypistä, Level Blueprint, Blueprint Interface, Blueprint Class ja Data-Only Blueprint. Blueprint-tyypeistä käytetyimmät ovat Level Blueprint ja Blueprint Class. [18.]

Level Blueprint

Jokaisella kentällä on yksi Level Blueprint (kuva 9), joka vastaa koko kentän toimintalogiikasta. Level Blueprintin kautta käsitellään kenttään liittyviä tapahtumia, jotka tapahtuvat vain kerran ja ovat todella kenttäkohtaisia, kuten esimerkiksi välianimaatiot (cutscene). Level Blueprint on ainoa Blueprint-tyyppi, jota ei voida itse luoda. Kullekin kentälle Unreal Engine itse luo yhden Level Blueprintin, ja sitä voidaan muokata vain Level Blueprint-editorista. [18.]



Kuva 9. Valikko, josta saadaan Level Blueprint -editori auki.

Blueprint Interface

Blueprint Interface on samanlainen kuin yleisen ohjelmoinnin rajapinta. Rajapinta kuvaa kaikki ominaisuudet, joita luokan täytyy toteuttaa. Blueprint

Interface lisätään Blueprint-luokkaan, jolloin kyseinen Bluperint-luokka takaa Blueprint-rajapintaan määritetyt ominaisuudet. [18.]

Data-Only Blueprint

Data-Only Blueprint on periaatteessa Blueprint-luokka, joka sisältää vain kanta-luokkansa ominaisuudet ilman solmukaaviota tai toiminnallista logiikkaa. Data-Only Blueprintin avulla voidaan helposti säätää perityt muuttujat ja komponentit ilman niiden etsimistä isosta solmukaaviosta. [18.]

Blueprint Class

Blueprint Class on käytetyin Blueprint-tyyppi, ja sen avulla voidaan lisätä toimintoja olemassa olevien peliluokkien päälle. Blueprint-luokka luodaan aina jostain Unrealin pääluokista. Taulukossa 4 ovat yleisimmät pääluokat, joista peritään uutta Blueprint-luokkaa luotaessa. [18.]

Taulukko 4. Unrealin yleisimmät luokat [18].

Luokan nimi	**Kuvaus**
Actor	Mikä tahansa esine, joka on maailmassa. Tukee kääntämistä, kiertämistä ja skaalausta.
Pawn	Actor, jota voidaan hallita ja joka saa käyttäjän syötteen ohjaimelta.
Character	Pawn, joka sisältää kyvyn kävellä, juosta ja hypätä.
Character Controller	Actor, joka on vastuussa Pawnin hallitsemisesta.
Game Mode Base	Määrittelee pelinsäännöt, tulokset ja minkä tahansa pelityypin.
Actor Component	Komponentti, joka lisätään mihin tahansa Actoriin. Actor-komponentilla ei ole fyysistä sijaintia.
Scene Component	Komponentti, jolla on Scene-muunnos ja joka voidaan liittää muihin Scene-komponentteihin.

2.5 C++:n ja Blueprintin välinen viestintä

Unreal Engine mahdollistaa C++:n ja Blueprintien keskenään yhdistämisen. Ohjelmoijat käyttävät C++-ohjelmointirajapintaa toteuttamaan pelitoiminnallisuuksia, joita suunnittelijat ja taiteilijat voivat hyödyntää ja laajentaa Blueprintilla.

C++-ohjelmointirajapinnan ja Blueprintien välinen viestintä tapahtuu Unrealin heijastusjärjestelmällä (Reflection System). Unrealin heijastusjärjestelmä sisältää erilaisia makroja (taulukko 5), joita ohjelmoijat lisäävät C++-koodiin. Heijastusjärjestelmän makrojen avulla Blueprint-luokat voivat saada tietoa C++-luokassa toteutetuista toiminnallisuuksista. Heijastusjärjestelmän makrot ovat UCLASS, USTRUCT, GENERATED_BODY, UPROPERT ja UFUNCTION. [20.]

Taulukko 5. Heijastusjärjestelmän makrot [20].

Makron nimi	**Kuvaus**
UCLASS()	Kertoo Unrealille, että kyseinen luokka heijastetaan.
USTRUCT()	Kertoo Unrealille, että kyseinen struktuuri heijastetaan.
GENERATED_BODY()	Lisää kaikki heijastuksessa tarvittavat peruskoodit kyseiseen luokkaan ja struktuuriin.
UPROPERTY()	Mahdollistaa luokassa ja struktuurissa määriteltyjen muuttujien käytön Blueprintissä.
UFUNCTION()	Mahdollistaa luokassa ja struktuurissa määriteltyjen funktioiden käytön Blueprintissä.

Heijastusjärjestelmän makroille on mahdollistaa antaa myös argumenttina lisämääritteitä. Lisämääritteillä määritetään, miten halutaan makron käyttäytyvän Blueprintissä. Lisämääritteisiin kuuluu makron kategoria ja Blueprintin muokattavuustieto. Esimerkiksi BlueprintCallable-lisäämäärite kertoo Unrealille, että kyseinen makro on Blueprintissä kutsuttavissa. [20.]

Jotta voidaan käyttää heijastusjärjestelmän makroja, täytyy lisätä kyseisen luokan header-tiedostoon generated.h-niminen tiedosto. Tällä tavalla kerrotaan Unrealille, että kyseinen luokka heijastetaan. [20.] Ilman generated.h-tiedoston lisäämistä ei voida käyttää heijastusjärjestelmän makroja. Esimerkkikoodissa 1 on MyPawn-niminen luokka, jonka funktiot ja muuttujat heijastetaan käyttämällä Unrealin heijastusjärjestelmän makroja.

```
#include "CoreMinimal.h"
#include "GameFramework/Pawn.h"
#include "MyPawn.generated.h"

UCLASS()
Class MYPROJECT_API AMyPawn : public APawn
{

     GENERATED_BODY()
     public:
          // Sets default values for this pawn's properties
          AMyPawn();
     protected:
          // Called when the game starts or when spawned
          virtual void BeginPlay() override;
     public:
          // Called every frame
          virtual void Tick(float DeltaTime) override;

          // Called to bind functionality to input
     virtual void SetupPlayerInputComponent(class UInputComponent*
     PlayerIputComponent) override;

     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, category = "MyPawn")
     UStaticMeshComponent* PawnMesh;

     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, category = "MyPawn")
     USpringArmComponent* CameraSpringArm;

     UPROPERTY(VisibleAnywhere, BlueprintReadOnly, category = "MyPawn")
     UCameraComponent* Camera;

     UFUNCTION(BlueprintImplementableEvent, category = "MyPawn")
     void Jump();
}
```

Esimerkkikoodi 1. MyPawn-niminen luokka, jonka ominaisuudet heijastetaan käyttämällä Unrealin heijastusjärjestelmän makroja, kuten UFUNCTION() ja UPROPERTY().

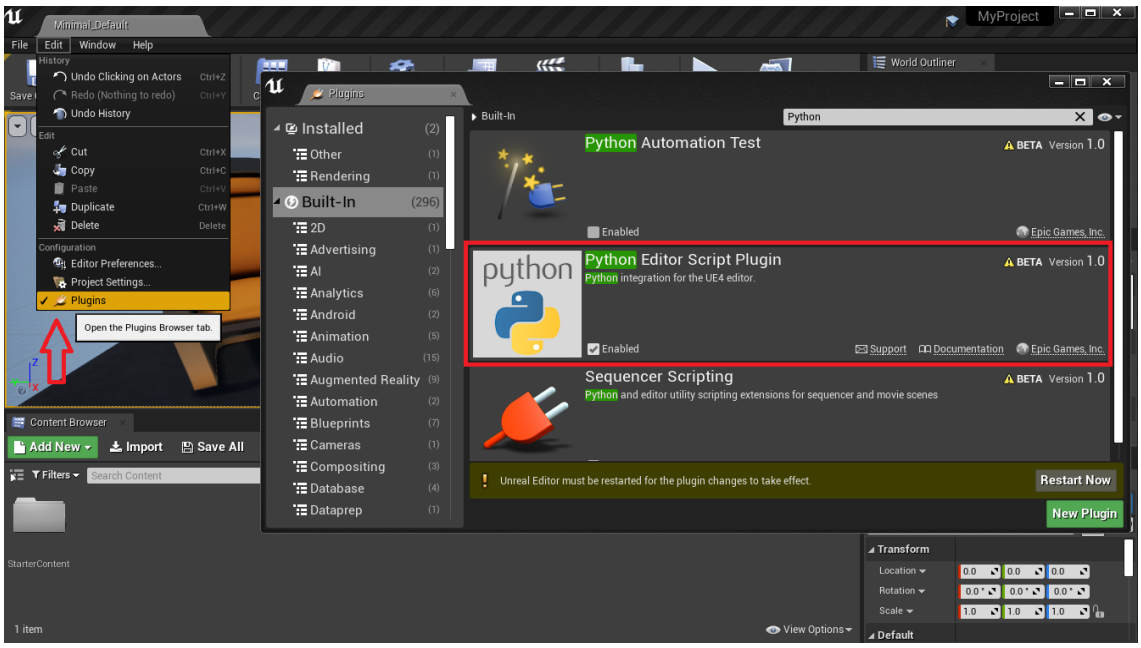
2.6 Unreal Python -ohjelmointirajapinta

Python on dynaaminen korkean tason olio-orientoitu ohjelmointikieli. Se on tehokas ja tulkittu kieli, eli sillä kirjoitettua koodia ei tarvitse kääntää konekieliseksi ohjelmaksi. Pythonia on helpompi oppia kuin muita ohjelmointikieliä, ja se on tarkka syntaksin kanssa. Tämän takia Pythonia yleensä suositellaankin aloitteleville ohjelmoijille ensimmäiseksi ohjelmointikieliksi, koska se pakottaa aloittelijan oppimaan kirjoittamaan puhdasta ja hyvää koodia. Pythonia voidaan integroida muihin teknologioihin, koska se on modulaarinen. Python tarjoaa paljon erilaisia kirjastoja (libraries) ja kehyksiä (frameworks), joita voidaan käyttää nopeuttamaan sovellusten kehitystä. [21.]

Pythonin tehosta ja helppokäyttöisyydestä huolimatta sillä on myös huonoja puolia. Koska se on tulkittu ohjelmointikieli, se on hidasta verrattuna C++-ohjelmointikieleen, joka on käännetty kieli. Pythonin toinen huono puoli on se, että se on dynaaminen ohjelmointikieli, eli muuttujilla ei ole määrättyjä tyyppejä, mikä voi johtaa helposti erilaisiin virheisiin, joita täytyy sitten testata ja korjata erikseen. [21.]

C++- ja Blueprint-ohjelmointirajapintojen lisäksi Unreal Engine 4 tukee Pythonia. Unreal Python -ohjelmointirajapinta verrattuna C++- ja Blueprint-ohjelmointirajapintoihin on uudempi. Unreal Engine 4 alkoi tukea Pythonia vasta version 4.19 jälkeen, ja siitä lähtien sitä on kehitetty ja päivitetty jatkuvasti.

Unreal Python -ohjelmointirajapintaa ei ole asennettu Unreal Engineen oletuksena. Ennen kuin voidaan alkaa käyttää Pythonia Unreal Enginen kanssa, se on asennettava erikseen. Sen asentaminen on melko suoraviivaista, ja se onnistuu helposti Unreal Editorin Plugins-työkalun kautta. Kuvassa 10 näkyy, miten asennetaan Unreal Python -ohjelmointirajapinta Unreal Engineen. [22.]



Kuva 10. Python-ohjelmointirajapinnan asentaminen Unreal Engineen.

Unreal Python -ohjelmointirajapinta tarjoaa laajan valikoiman luokkia ja toimintoja, joita voidaan käyttää Unreal Editorin muokkaamiseen [23]. Jotta voidaan käyttää näitä eri luokkia ja toimintoja, täytyy Python-skriptiin alkuun lisätä unreal-niminen moduuli, joka mahdollistaa kaikkien Unrealin luokkien ja toimintojen käytön. Python-, C++- ja Blueprint-ohjelmointirajapinnat voidaan yhdistää keskenään. Tämä mahdollistaa Pythonilla tehtyjen toimintojen käytön sekä C++-koodissa että Blueprintissä. [22.]

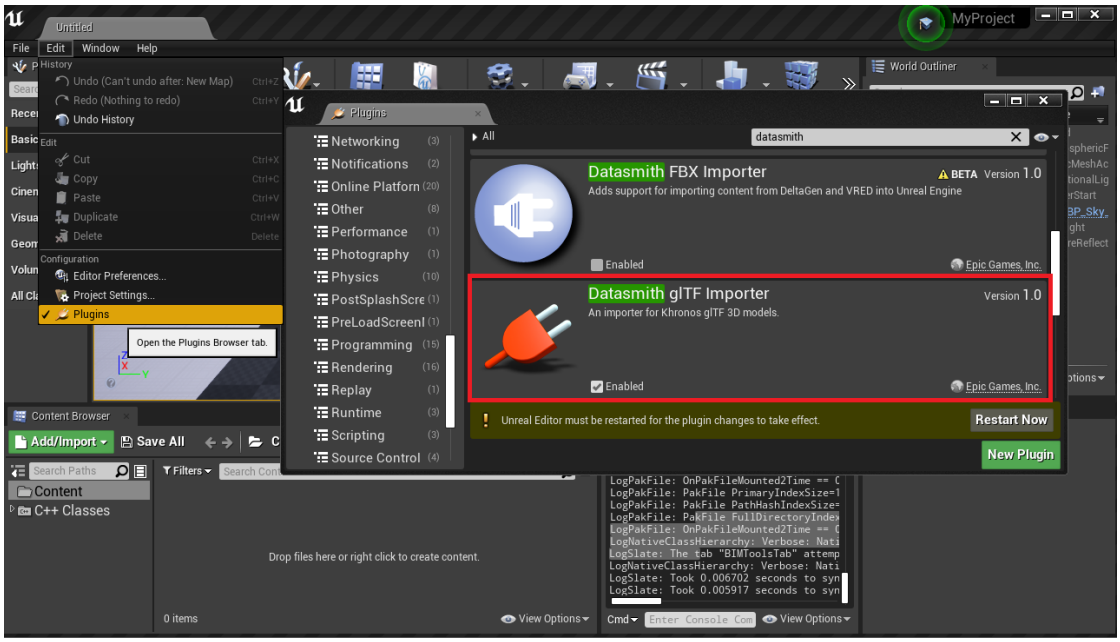
Unreal Python -ohjelmointirajapinnan hyvä puoli Unreal C++- ja Blueprintin-ohjelmointirajapintoihin nähden on se, että Pythonilla voidaan helpommin ja nopeammin automatisoida Unreal Editorin erilaisia manuaalisia ja toistuvia tehtäviä, kuten esimerkiksi 3D-mallien tuonti tai suuri määrä tiedostojen uudelleenjärjestämisiä ja -nimeämisiä. Tehokkuudesta ja nopeudesta huolimatta Python-ohjelmointirajapinnan heikko puoli on se, että sitä ei voida vielä käyttää pelien tai sovellusten tekemiseen, koska se on tällä hetkellä ainoastaan Editorissa käytettävissä. [22.]

2.7 Datasmith-työkalu

Datasmith on Epic Gamesin kehittämä ilmainen työkalu, joka yksinkertaistaa 3D-mallien tai kohtausten (scenes) tuonnin 3DS Max-, Blender-, AutoCAD- tai Revit-3D-mallinnusohjelmistoista Unreal Engine -pelimoottoriin. Datasmith-työkalu automatisoi useita eri vaiheita 3D-mallien tuonnissa Unreal Engine -pelimoottoriin. Osa näistä vaiheista ovat esimerkiksi materiaalien muuttaminen Unreal Engine -materiaaleiksi, mallien valokarttojen luominen ja projektihierarkian säilyttäminen. [24.]

Jotta voidaan alkaa käyttää Datasmith-työkalua, se täytyy ensin asentaa Unreal Engine -pelimoottoriin. Datasmith-työkalu asennetaan Unreal Editorin Plugins-työkalun avulla. Datasmith tukee useita eri tiedonsiirtoformaatteja, kuten OBJ (3D Object File), FBX (Filmbox), COLLADA (DAE) ja glTF (Graphics Language Transmission Format), ja jokaiselle tiedostoformaatille se tarjoaa oman lisäosan.

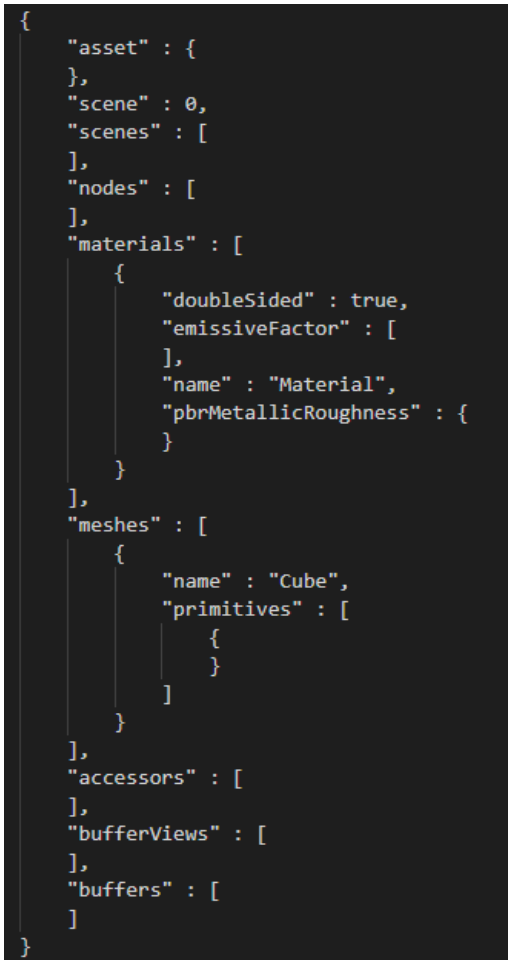
Insinöörityöprojektissa käytettiin Datasmith glTF Importer -lisäosaa glTF-3D-mallien tuomiseen Unreal Engine -pelimoottoriin. Kuva 11 havainnollistaa Datasmith glTF Importer -lisäosan asentamista.



Kuva 11. Datasmith glTF Importer -lisäosa.

glTF-tiedonsiirtoformaatti

glTF eli (Graphics Language Transmission Format) on Khornos-ryhmän kehittämä avoin tiedonsiirtoformaatti. Khornos-ryhmä kehitti glTF-formaattia vähentämään 3D-tiedostojen kokoa ja nopeuttamaan niiden siirtoa. glTF tallentaa 3D-mallien tiedot, kuten geometrian ja materiaalit JSON-formaattina, ja tämä pienentää tiedostojen kokoa ja nopeuttaa niiden siirtoa ajoaikana. Kuvassa 12 näkyvät Blender-mallinnusohjelmistolla luodun Cube-nimisen 3D-mallin glTF-tiedoston tiedot JSON-formaatissa. [25.]



Kuva 12. Cube-nimisen 3D-mallin tiedot JSON-formaatissa.

Koska glTF-tiedosto on JSON-formaatissa, se mahdollistaa myös tiedoston virheenjäljityksen ja muokkaamisen tekstitiedostona [25].

3 Unreal Editorin laajentaminen

Tietokoneohjelman lisäosa on ohjelmisto, joka sisältää tiettyjä toimintoja, jotka ovat enimmäkseen kolmannen osapuolen luomia. Lisäosilla lisätään uusia toimintoja olemassa oleviin ohjelmiin ja laajennetaan niiden ominaisuuksia. Lisäosia luodaan parantamaan muiden ohjelmien käyttökokemusta muokkaamatta niiden lähdekoodia. [26.]

Unreal Engine tarjoaa useita eri työkaluja ja Engineen sisään rakennettuja lisäosia, joita voidaan käyttää sovelluksia kehittäessä. Epic Games Marketplace -sivustolta löytyy myös muiden tekemiä maksullisia ja ilmaisia lisäosia, joita voidaan ladata ja asentaa Unreal Engineen. Unreal-lisäosien tekeminen ei ole helppoa ilman aiempaa C++-ohjelmointi kokemusta ja Unreal Enginen kehitysympäristön ymmärrystä. Lisäosien tekemiseen ei löydy paljon apua internetistä. Lisäosan tekemiseen liittyviä dokumentaatioita tai opetusvideoita löytyy todella vähän. Tuntuu siltä, että Unreal Engine -pelimoottorin kehittäjät olettavat, että kehittäjät itse jotenkin oppivat kehittämään lisäosia ja laajentamaan editoria, koska Enginen lähdekoodi on tarkasteltavissa. Tästä syystä uusille kehittäjille lisäosien kehittäminen tuntuu usein hyvin hankalalta.

3.1 Unreal-lisäosan rakenne

Unreal-lisäosa sisältää yhden tai useamman C++-moduulin. Moduulit koostuvat useista eri luokista. Moduulien avulla lisätään Unreal Engineen erilaisia toiminnallisuuksia. Lisäosan ja moduulin välinen ero on se, että lisäosaa voidaan käyttää useissa eri projekteissa. Periaatteessa lisäosien avulla jaetaan moduuleja ja niiden toimintoja eri projektien kesken.

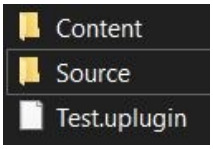
Moduuleja on kolme eri tyyppiä, ja lisäosa voi sisältää kaikki nämä tyypit. Moduuli-tyypit ovat Editori, Runtime ja Developer. Nimiensä mukaan Editori-moduuli toimii editoritilassa, Runtime ajoaikana ja Developer sekä ajoaikana että editoritilassa, mutta vain kehityksen aikana. Jokaisella luodulla moduulilla on cpp-, h- ja build.cs-tiedosto. build.cs-tiedostoon määritellään moduuliriippuvuuksia, kirjastoja ja polkuja. [27; 28.]

Moduulien lisäksi lisäosa voi sisältää sisältöä, kuten lisäosan käyttöliittymä ja muita tiedostoja. Lisäosan sisältöä otetaan käyttöön asettamalla lisäosan uplugin-tiedostossa olevan CanContainContent-asetus tilaan true. Lisäosan uplugin-tiedostosta löytyy lisäosan kuvaus (kuva 13). Siihen määritellään lisäosaan liittyvät tiedot, kuten lisäosan versio, nimi, moduulit yms. [27.]

```
{
  "FileVersion": 3,
  "Version": 1,
  "VersionName": "1.0",
  "FriendlyName": "TestPlugin",
  "Description": "",
  "Category": "Other",
  "CreatedBy": "Afshin Safari",
  "CreatedByURL": "",
  "DocsURL": "",
  "MarketplaceURL": "",
  "SupportURL": "",
  "CanContainContent": true,
  "IsBetaVersion": true,
  "IsExperimentalVersion": false,
  "Installed": false,
  "Modules": [
    {
      "Name": "TestPluginModule",
      "Type": "Editor",
      "LoadingPhase": "Default"
    }
  ]
}
```

Kuva 13. TestPlugin-nimisen lisäosan uplugin-tiedosto.

Kuvassa 14 näkyy TestPlugin-nimisen lisäosan kansiorakenne. Lisäosan moduulin lähdekoodit sijaitsevat Source-kansiossa ja lisäosan sisällöt Content-kansiossa. Lisäosan kansiorakenne sijaitsee joko Unreal Enginen Plugins (/Engine/Plugins) -kansiossa tai luodun projektin Plugins ([Projektin kansio]/Plugins) -kansiossa riippuen siitä, onko lisäosa Engine- vai Projekti-tason lisäosa.

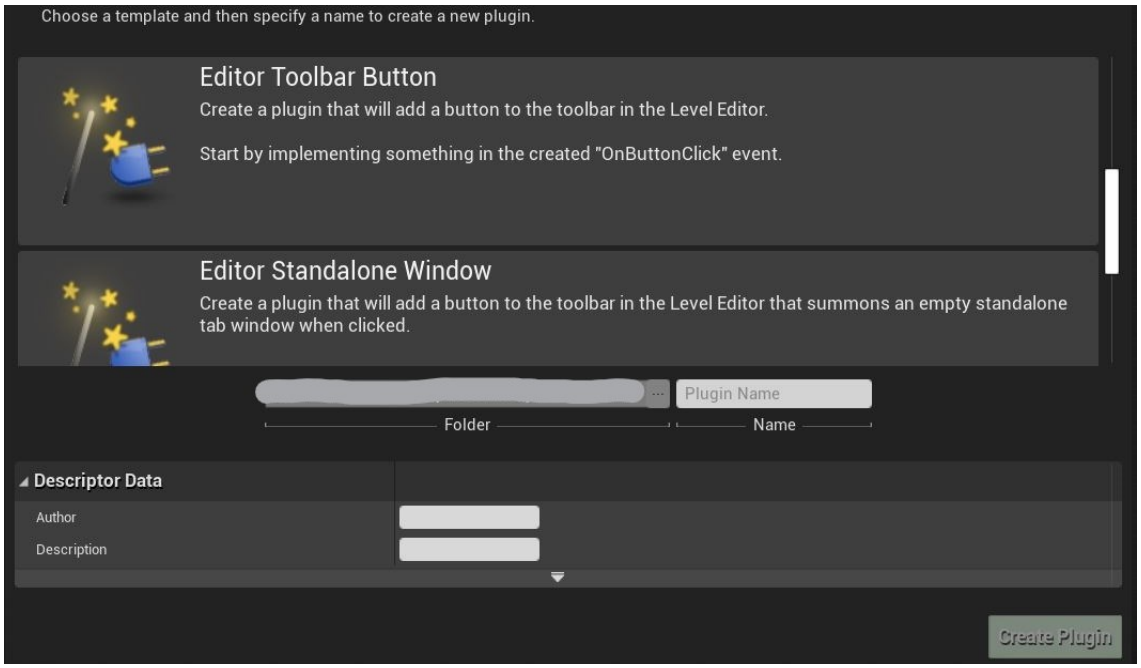


Kuva 14. TestPlugin-nimisen lisäosan kansiorakenne.

3.2 Unreal-lisäosien luominen

Unreal-lisäosat voidaan luoda joko Engine- tai Project-tasolla. Engine-tasolla oleva lisäosa on Engineen sisään rakennettu, ja sen lähdekoodi on valmiiksi käännetty ja sitä asennetaan tiettyyn Unreal Enginen versioon Epic Games käynnistysohjelman kautta. Tällä tavalla lisäosaa voidaan käyttää useissa eri projekteissa. Projektitason lisäosa on toisaalta käytössä vain tietyssä projektissa ja se täytyy manuaalisesti lisätä projektin Plugins-kansioon ja kääntää sen lähdekoodi. [27.]

Unreal-lisäosia voidaan luoda manuaalisesti tai Unrealin Plugins-työkalun avulla (kuva 15), jolloin Unreal luo automaattisesti luodun projektin Plugins-kansion sisälle lisäosan kaikki tarvittavat kansiot ja tiedostot.



Kuva 15. Unreal Editorin Plugins-työkalu, jonka avulla voidaan luoda lisäosia.

Unreal-lisäosia on erityyppisiä. Esimerkiksi tässä projektissa lisäosan tyypiksi valittiin Editor Toolbar Button -tyyppi. Nimensä mukaisesti Editor Toolbar Button -lisäosa laajentaa Unreal Editoria lisäämällä painikkeen Unreal Editorin päätyöpalkkiin.

4 Lisäosan suunnittelu ja toteutus

Tässä luvussa käydään läpi opinnäytetyössä Unreal Engine 4 -pelimoottoriin toteutetun lisäosan suunnittelua ja toteutusta.

4.1 Yleiskuva

Tridifyssa pyritään jatkuvasti tuottamaan rakennusarkkitehtuuriin liittyviä aputyökaluja, joilla voitaisiin helpottaa asiakkaiden töiden tuottavuutta ja luovuutta. Yrityksessä päätettiin laajentaa Unreal Engine 4 -pelimoottoria toteuttamalla siihen lisäosa, jonka avulla voitaisiin automatisoida Tiridfy Oy:n glTF-3D-mallien tuontiprosessi Unreal Engine 4 -pelimoottoriin.

Lisäosan nimeksi valittiin BIMTools, ja tavoitteena oli saada lisäosa siihen vaiheeseen, että pystyttäisiin lataamaan glTF-tiedostoformaatissa olevia 3D-malleja Tridify Oy:n palvelimesta rajapinnan välityksellä, tuomaan ladatut 3D-mallit pelimoottoriin ja visualisoimaan niitä pelimoottorissa.

4.2 Lisäosan suunnittelu

Lisäosan tarve syntyi, koska haluttiin automatisoida Tridify Oy:n glTF-3D-mallien tuontiprosessi Unreal Engine 4 -pelimoottoriin. Toteutettavasta toiminnallisuudesta päätettiin tehdä lisäosa, koska tavoitteena oli saada toiminnallisuus toimimaan useissa eri projekteissa. Ennen käytännön työn aloittamista sovittiin lisäosan vaatimusmäärittelystä. Nämä vaatimukset asetettiin myös arvioitaviksi osaksi projektin onnistumisen määrittämiseen.

Lisäosan vaatimukset:

- Lisäosa tulee ladata glTF-3D-mallit Tridify Oy:n palvelimesta rajapinnan välityksellä annetun mallin ”ShareKey” mukaan.
- Lisäosa tarvitsee käyttöliittymän, jonka avulla käyttäjä syöttää mallin ”ShareKey” ja lataa glTF-3D-mallit.
- Lisäosan tulee tuoda ladatut glTF-3D-mallit pelimoottoriin ja visualisoida niitä pelimoottorissa.

Lisäosan vaatimusten määrittelyn jälkeen aloitettiin tutustuminen Unreal Engine 4 -pelimoottoriin, Unreal Editorin laajentamiseen ja Unreal-lisäosien kehittämiseen. Tutustumisen jälkeen aloitettiin asetettujen lisäosan vaatimusten pohjalta lisäosan toteuttaminen. Koska aiempaa kokemusta Unreal-lisäosien toteutuksesta ei ollut, jouduttiin usein tutkimaan ja kokeilemaan asioita parhaiden ratkaisujen löytämiseksi.

4.3 Lisäosan käyttöliittymän suunnittelu

Asetettujen vaatimusten pohjalta lisäosalle suunniteltiin käyttöliittymä. Käyttöliittymän suunnittelussa tutustuttiin Unreal Editorin käyttöliittymien luontitapoihin. Unreal Editorin käyttöliittymiä on mahdollista luoda kahdella eri tavalla, Slate-ohjelmistokehyksellä tai UMG:tä käyttämällä. Suunnittelussa tutustuttiin näihin molempiin ja verrattiin niiden ominaisuuksia keskenään. Vertailussa huomattiin, että UMG ei ominaisuuksiltaan eroa Slate-ohjelmistokehyksestä. UMG:llä luodaan käyttöliittymiä graafisesti, mikä tekee käyttöliittymän luomisesta helpompaa. Slatella taas käyttöliittymiä luodaan kirjoittamalla koodia. UMG:ltä löytyy myös paljon enemmän opetusmateriaaleja kuin Slatelta ja UMG:n dokumentaatiota päivitetään jatkuvasti. Vertailun jälkeen päädyttiin käyttämään UMG:tä sen yksinkertaisuuden ja helppokäyttöisyyden vuoksi.

Lisäosan käyttöliittymä koostu kuudesta elementistä: tekstin syöttökentästä, Materials & Textures- ja Generate Lightmap UVs -valintaruuduista, Import Uniform Scale spinbox -valikosta, Lightmap Resolution -valintalistasta sekä Import-painikkeesta. Tekstin syöttökenttään syötetään ladattavan mallin ShareKey ja Materials & Textures- ja Generate Lightmap UVs-valintaruuduilla määritetään, halutaanko tuoda mallin materiaalit ja tekstuurit pelimoottoriin ja luoda mallille valokartat. Import Uniform Scale spinbox -valikolla säädetään mallin koko, Lightmap Resolution -valintalistalla valitaan mallille luodun valokartan resoluutio ja Import-painiketta painamalla ladataan 3D-mallit ja tuodaan ne pelimoottoriin.

Lisäosan käyttöliittymän suunniteltiin niin, että se sisältäisi samoja parametrejä, joita löytyy Datasmith glTF-tuojasta mallien tuontia varten. Tällä tavalla käyttäjä

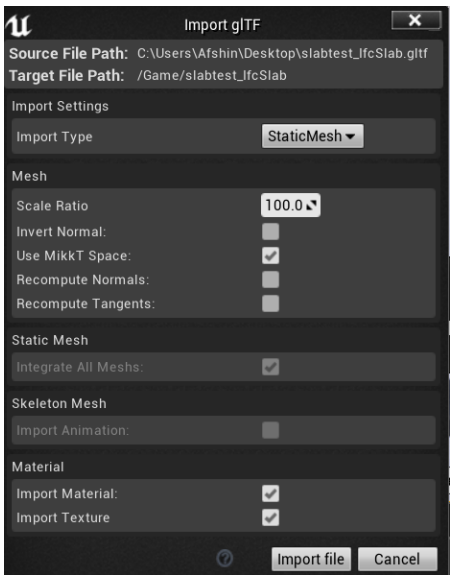
pystyisi määrittämään Datasmith glTF-tuojan parametrejä suoraan lisäosan käyttöliittymästä.

4.4 glTF-tuojan valitseminen

3D-mallien tuontia ja visualisointia varten tarvittiin glTF-tuoja. Ajan säästämiseksi päätettiin käyttää olemassa olevia Unreal Enginen glTF-tuojia mallien tuontia ja visualisointia varten. Tutustuttiin kolmeen eri Unreal Enginen glTF-tuojaan. Vertailtiin niiden ominaisuuksia, hyviä ja huonoja puolia keskenään. Sen jälkeen valittiin kolmen glTF-tuojan välillä paras, ja sen avulla tuotiin mallit pelimoottoriin ja visualisoitiin ne pelimoottorissa.

glTF-tuoja (Code4Game)

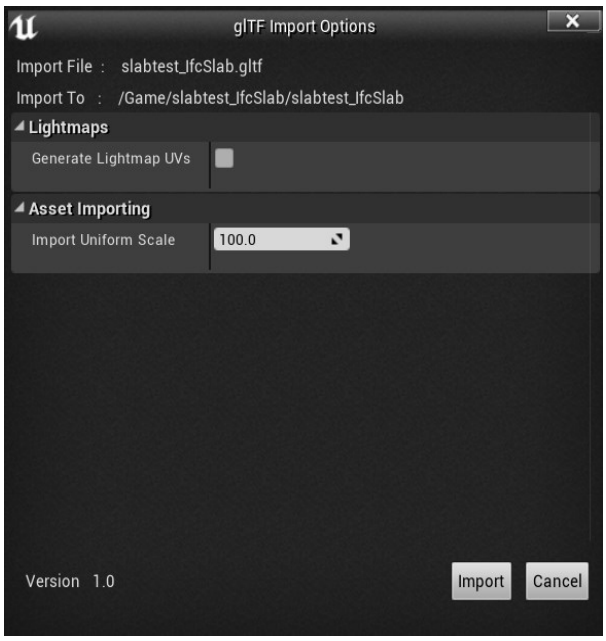
Ensimmäinen glTF-tuoja, jota kokeiltiin, oli Code4Gamen glTF-tuoja. Työkalussa on mallin tuontia varten useita eri parametrejä (kuva 16), kuten mallin materiaalien tuonti ja mallin koon säätäminen. Mallin pelimoottorille tuonnin jälkeen mallin nimi pysyy oikeana, mutta se ei osaa luoda tuodulle mallille valokartta. Työkalu täytyy myös manuaalisesti asentaa projektiin ja kääntää lähdekoodi, koska sitä ei ole rakennettu Unreal Engine -pelimoottoriin sisään. Työkalusta ei myöskään löydy dokumentaatiota.



Kuva 16. Code4Gamen glTF-tuoja.

glTF-tuoja (Khronos Group)

Khoronos-ryhmän glTF-tuojassa verrattuna Code4Gamen glTF-tuojaan on vähemmän parametreja mallin tuontia varten (kuva 17). Khronosin glTF-tuojan suurin ero verrattuna Code4Gamen glTF-tuojaan on siinä, että se on rakennettu suoraan Unreal Enginen sisään eli sen voi helposti asentaa projektiin Unreal Editorin Plugins-työkalun avulla. Se osaa luoda tuodulle mallille valokartat, mutta se luo mallin materiaalit väärin. Mallin nimi ei myöskään pysy oikeana, ja malli kääntyy 90 astetta tuonnin jälkeen. Tästäkään glTF-tuojasta ei valitettavasti löydy dokumentaatiota.

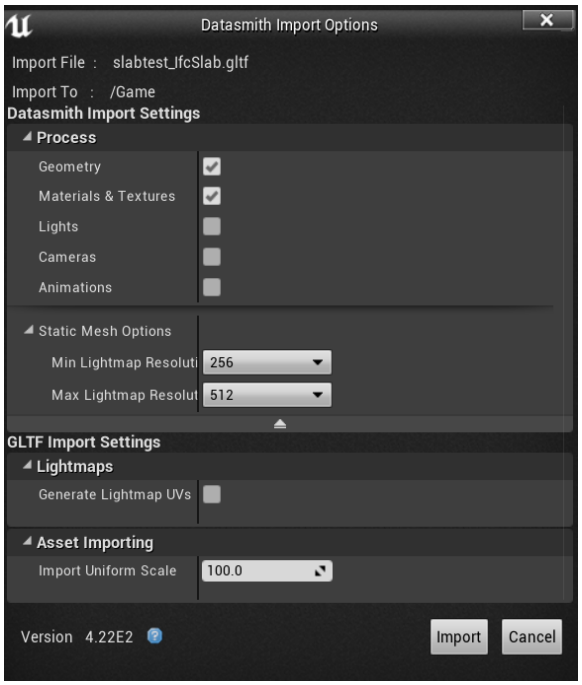


Kuva 17. Khoronos-ryhmän glTF-tuoja.

Datasmith-glTF-tuoja (Epic Games)

Viimeinen glTF-tuoja, jota kokeiltiin, oli Datasmith-glTF-tuoja. Se toimii melko samalla tavalla kuin Code4Gamen glTF-tuoja. Työkalussa on myös mallin tuontia varten lähes samoja parametrejä (kuva 18) kuin Code4Gamen glTF-tuojassa. Datasmith-glTF-tuojan suurin ero verrattuna kahteen edelliseen on sen juostavuus. Työkalusta löytyy dokumentaatiota, ja työkalun tuontiprosessi on muokattavissa Blueprintillä ja Pythonilla. Koska se on Epic Gamesin itse kehittämä työkalu, sitä myös päivitetään jatkuvasti. Työkalun ainoa heikko puoli on

se, että sen C++-API:a ei ole vielä julkaistu. Työkalun tuontiprosessia ei voi muokata vielä käyttäen C++-ohjelmointikieltä. Tämä vaikuttaa työnkulkuun, jos ei ole aiempaa kokemusta Blueprint- ja Python-ohjelmointikielistä.



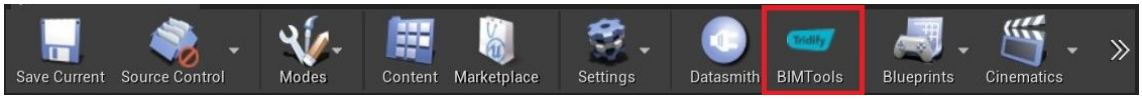
Kuva 18. Datasmith-glTF-tuoja.

Pitkän harkinnan ja vertailun jälkeen valittiin Datasmith-glTF-tuoja mallin tuontia ja visualisointia varten. Kaikilla kolmella glTF-tuojalla oli hyvät ja huonot puolet, mutta Datasmith-glTF-tuoja oli paras vaihtoehto kaikista kolmesta. Se on Epic Gamesin itse kehittämä työkalu, siitä löytyy dokumentaatiota ja se mahdollisti 3D-mallien tuontiprosessin muokkaamisen.

4.5 Lisäosan alustus ja ohjelmointi

Taustatutkimusten ja suunnitelmien pohjalta aloitettiin lisäosan toteutus. Lisäosan toteutus alkoi luomalla Unreal Enginen projektiselaimen kautta uusi C++-projekti, jonka mallipohjaksi valittiin Blank. Luodun projektin kansion sisälle luotiin Plugins-niminen kansio, johon Unreal Editorin Plugins-työkalun kautta luotiin Editor Toolbar Button -tyypinen lisäosa (kuva 19). Tätä lisäosaa käytettiin pohjana BIMTools-lisäosan toteuttamiseen. Unreal generoi automaattisesti Plugins-

kansion sisälle lisäosan kaikki tarvittavat kansiot ja tiedostot, kuten lisäosan uplugin-tiedosto, Source- ja Resource-kansiot.



Kuva 19. Lisäosan nappi, joka lisättiin Unreal Editorin päätyöpalkkiin.

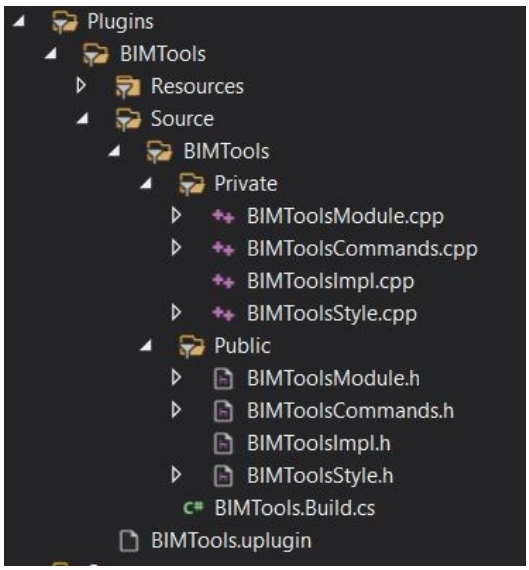
Lisäosassa käytetään Singleton-suunnittelumallia, koska lisäosan moduulista tarvitaan vain yksi ilmentymä. Tätä varten lisäosaan luotiin BIMToolsImpl- ja BIMToolsModule-luokat. BIMToolsImpl-luokka sisältää lisäosan moduulin toimintojen toteutuksen ja BIMToolsModule-luokka luo BIMToolsImpl-luokasta ilmentymän, jota voidaan käyttää muissa lisäosan moduulin sisälle luoduissa luokissa. BIMToolsImpl-luokasta luodaan ilmentymä BIMToolsModule-luokan StartupModule-metodissa (esimerkkikoodi 2).

```
FBIMToolsModule* FBIMToolsModule::Singleton = nullptr;
TSharedPtr<BIMToolsImpl>     FBIMToolsModule::BIMToolsModuleImplPtr =
nullptr;

void FBIMToolsModule::StartupModule()
{
     Singleton = this;
     BIMToolsModuleImplPtr = MakeShareable(new BIMToolsImpl());
     FBIMToolsModule::Get()->StartupModule();
}
```

Esimerkkikoodi 2. BIMToolsModule-luokan StartupModule-metodi, jossa luodaan BIMToolsImpl-luokasta ilmentymä. StartupModule-metodia kutsutaan, kun Unreal Editor lataa moduulin.

Kuvassa 20 näkyy lisäosan tiedostorakenne. Lisäosan Source-kansiosta löytyy BIMTools-moduulin lähdekooditiedostot ja build.cs-tiedosto. Resources-kansiosta taas löytyvät lisäosan kuvakkeet.



Kuva 20. BIMTools-lisäosan tiedostorakenne.

4.6 glTF-tiedostojen lataaminen ja tallentaminen

Unreal Engine tarjoaa HTTP API:n, jonka avulla voidaan suorittaa HTTP/REST-pyyntöjä. Lisäosassa glTF-tiedostojen lataamiseen käytetään HTTP-protokollan GET-metodia. REST-pyyntö lähetetään rajapintaan, minkä jälkeen rajapinta käsittelee pyynnön ja palauttaa JSON-muotoisen vastauksen. Rajapinnasta palautettua vastausta ei kuitenkaan voida käsitellä sellaisena, kuin se on, vaan se täytyy lukea ja deserialisoida eli kuvata olioksi, ja tällä tavalla vastauksen käsittely ohjelmallisesti helpottuu merkittävästi.

Jotta voidaan käyttää Unrealin HTTP API:a pyyntöjen lähettämiseen ja käsitellä rajapinnalta palautettu JSON-muotoinen vastaus, täytyy build.cs-tiedostossa olevaan PublicDependencyModuleNames-osioon lisätä Http-, Json- ja JsonUtilities-moduulit. Samalla luokkaan, josta pyyntö lähetetään ja palautettu vastaus käsitellään, täytyy lisätä HttpModule.h- ja JsonSerializer.h-tiedostot. Rajapinnasta palautettu JSON-muotoinen vastaus voidaan lukea ja deserialisoida käyttämällä TJsonReader- ja FJsonObject-luokkia.

Lisäosassa pyynnön lähettämistä ja palautetun vastauksen käsittelyä varten luotiin kaksi metodia, SendRequest ja OnResponseRecived. SendRequest-

metodilla lähetetään pyyntö rajapintaan, ja OnResponseRecived-metodilla taas käsitellään rajapinnasta palautettu vastaus (esimerkkikoodi 3).

```
void BIMToolsImpl::SendRequest(FString Url)
{
     TSharedRef<IHttpRequest> Request = FHttpModule::Get().Cre-
     ateRequest();
     Request->SetURL(Url);
     Request->SetVerb("GET");
     Request->OnProcessRequestComplete().BindRaw(this, &BIMTool-
     sImpl::OnResponseReceived);
     Request->ProcessRequest();
}
```

Esimerkkikoodi 3. SendRequest-metodi, joka ottaa parametrina mallin URL. SendRequest-metodissa luodaan Request-objekti FHttpModuulista. Määritetään otsikkotiedot ja pyynnön metodi, joka on tässä tapauksessa HTTP-protokollan GET-metodi. ProcessRequest-komennolla lähetetään pyyntö. Onnistuneen pyynnön jälkeen suoritetaan OnProcessRequestComplete-metodi. OnProcessRequestComplete-metodi ottaa argumenttina Callback-metodin, tässä tapauksessa OnResponseRecived-metodin, jonka avulla käsitellään rajapinnasta palautettu vastaus.

glTF-tiedostojen lataamisen jälkeen ne piti tallentaa koneelle. Tarkoitus oli tiedostojen tallentamisen yhteydessä luoda projektin kansion sisälle kansio nimeltä Downloads, johon tallennetaan kaikki glTF-tiedostot. Tiedostojen ja kansioiden käsittelyä varten Unreal Engine tarjoaa oman tiedostojärjestelmän nimeltä UFS (Unreal File System). Käyttämällä Unrealin tiedostojärjestelmää voidaan luoda ja poistaa kansioita, tallentaa tiedostoja koneelle sekä päästä käsiksi projektin kansioihin.

Ohjelmassa jokaisen uuden 3D-mallin glTF-tiedostojen lataamisen alussa luodaan Downloads-kansio, johon tallennetaan ladatut glTF-tiedostot ja samalla tarkistetaan, onko Downloads-kansio jo olemassa. Jos kansio on olemassa, poistetaan se ja tilalle luodaan uusi. Tällä tavalla uuden 3D-mallin glTF-tiedostojen lataamisen yhteydessä koneelta poistetaan vanhan 3D-mallin ladatut glTF-tiedostot. Kansion luominen ja poistaminen tapahtuu CreateDownloadsDirectory- (esimerkkikoodi 4) ja DeleteDownloadsDirectory (esimerkkikoodi 5) -metodeilla.

```
void BIMToolsImpl::CreateDownloadsDirectory()
{
     IPlatformFile& PlatformFileManager = FPlatformFi-
     leManager::Get().GetPlatformFile();

     if (PlatformFileManager.CreateDirectory(*DownloadsPath))
     {
          UE_LOG(LogTemp, Error, TEXT("Directory was created"));

     } else {
          UE_LOG(LogTemp, Error, TEXT("Failed to create directory"));
     }
}
```

Esimerkkikoodi 4. CreateDownloadsDirectory-metodi, jonka avulla luodaan Downloads-kansio ja tallennetaan tiedostot.

```
void BIMToolsImpl::DeleteDownloadsDirectory(*DownloadsPath)
{
     IPlatformFile& PlatformFileManager = FPlatformFi-
     leManager::Get().GetPlatformFile();

     if (PlatformFileManager.DirectoryExists(*DownloadsPath))
     {
          FFileManagerGeneric::Get().DeleteDirectory(*DownloadsPath,
          true, true);
     }
}
```

Esimerkkikoodi 5. DeleteDownloadsDirectory-metodi, joka poistaa Dowloads-kansion. Kansion olemassaolo tarkistetaan DirectoryExists-metodilla, ja jos kansio on olemassa, se poistetaan DeleteDirectory-metodilla.

4.7 glTF-tiedostojen tuonti ja visualisointi

3D-mallin glTF-tiedostojen lataamisen ja tallentamisen jälkeen seuraava vaihe oli tuoda ladatut mallit pelimoottoriin ja visualisoida niitä pelimoottorissa. Tätä varten käytettiin Datasmith-glTF-tuojaa. Mallien tuomista varten jouduttiin muokkaamaan Datasmith-glTF-tuojan tuontiprosessia. Työkalun tuontiprosessia piti muokata siten, että se kävisi Downloads-kansiossa olevat kaikki ladatut glTF-tiedostot läpi ja toisi ne pelimoottoriin. BIMTools-lisäosan kehittämisvaiheessa Datasmith-glTF-tuojan tuontiprosessi oli muokattavissa vain Pythonilla, sillä työkalun C++ API:a ei ollut vielä julkaistu. Tästä syystä jouduttiin käyttämään Unrealin Python API:a C++-API:n kanssa rinnakkain. Tämä hankaloitti kehitystyötä, koska aiempaa kokemusta Python-ohjelmointikielestä ei ollut ja lisäosa oli kehitetty kokonaan C++-ohjelmointikielellä.

Jotta voidaan ajaa Python-koodia Unreal Editorissa, täytyy erikseen Plugins-työkalun kautta asentaa Python Editor Script -lisäosa Unreal Editoriin. Tämä ei kuitenkaan ollut tavoitteena, koska se tarkoitti sitä, että käyttäjä joutuisi itse Plugin-työkalun kautta asentamaan Python Editor Script -lisäosan Unreal Editoriin. Tämä ongelma ratkaistiin lisäämällä Python Editor Script -lisäosa BIMTools-lisäosan uplugin-tiedostossa olevaan Plugins-osioon. Samalla tavalla myös Datasmith-glTF-tuoja lisättiin uplugin-tiedoston Plugins-osioon (esimerkkikoodi 6). Tällä tavalla varmistettiin, että BIMTools-lisäosan asennuksen yhteydessä asentuvat myös Python Editor Script ja Datasmith-glTF-tuoja Unreal Editoriin.

```
"Plugins": [
     {
          "Name": "PythonScriptPlugin",
          "Enabled": true
     },
     {
          "Name": "DatasmithImporter",
          "Enabled": true
     },
     {
          "Name": "DatasmithGLTFImporter",
          "Enabled": true
     },
   ]
```

Esimerkkikoodi 6. BIMTools.uplugin-tiedoston Plugins-osio, johon lisättiin Python Script Plugin ja Datasmith-glTF-tuoja.

Datasmith-glTF-tuojan tuontiprosessin muokkaamista aloitettiin luomalla uusi Python-skriptitiedosto nimeltä DatasmithGLTFImport.py, johon toteutettiin ”Downloads”-kansion läpikäynti ja mallien tuontilogiikat. Tiedostoon luotiin kaksi funktiota, count_gltf_files_in_directory ja process_directory. count_gltf_files_in_directory-funktio (esimerkkikoodi 7) palauttaa “Downloads”-kansion sisällä olevien glTF-tiedostojen kokonaismäärän.

```
def count_gltf_files_in_directory(directory):
     gltf_file_count = 0
     item_list = os.listdir(directory)
     for item in item_list:
         item_full_path = os.path.join(directory, item)
         if os.path.isdir(item_full_path):
              gltf_file_count =+ count_gltf_files_in_direc-
              tory(item_full_path)
         else:
             ext = os.path.splitext(item)[1]
             if ext == '.gltf' :
                gltf_file_count =+ 1
     return gltf_file_count
```

Esimerkkikoodi 7. count_gltf_files_in_directory-funktio käy Downloads-kansion sisällä olevat glTF-tiedostot läpi ja palauttaa niiden kokonaismäärän, jota käytetään latauspalkin tekemiseen.

Process_directory-funktiolla taas käydään Downloads-kansiossa olevat glTF-tiedostot läpi, määritetään Datasmith-glTF-tuojan parametrit mallien tuontia varten ja tuodaan ne pelimoottoriin.

Raskaiden tiedostojen, tässä tapauksessa glTF-tiedostojen, tuominen Unreal Editoriin on hidas tehtävä, ja ne aiheuttavat Editorin jäätymisen. Tätä varten BOMTools-lisäosaan toteutettiin latauspalkki (esimerkkikoodi 8). Latauspalkkien tekemiseen käytetään Unrealin FScopedSlowTask-luokkaa, joka helpottaa Editorin latauspalkkien tekoa. FScopedSlowTask-luokka tarjoaa useita eri metodeja, kuten enter_progress_frame, should_cancel ja make_dialog, joita voidaan käyttää latauspalkkien tekemiseen ja hallitsemiseen.

```
# get the number of gltf files in the Downloads directory
number_gtlf_files = count_gltf_files_in_directory(downloads_directory)

with unreal.ScopedSlowTask(number_gtlf_files, "Batch Import") as
slow_task:
     slow_task.make_dialog(True)
```

Esimerkkikoodi 8. Latauspalkin luonti käyttämällä Unrealin Python-API:a. ScopedSlowTask-funktio ottaa argumenttina ladattujen glTF-tiedostojen kokonaismäärän.

glTF-tiedostojen lataamisen jälkeen täytyi ajaa DatasmithGltfImport.py-Python-skripti jotta pystyttiin tuomaan mallit pelimoottoriin. Tarkoitus oli ajaa Python-skriptin käyttäen C++-API:a. Python-skriptejä voidaan ajaa C++:lla lisäämällä build.cs-tiedoston PrivateDependecyModuleNames-osioon PythonScriptPlugin-

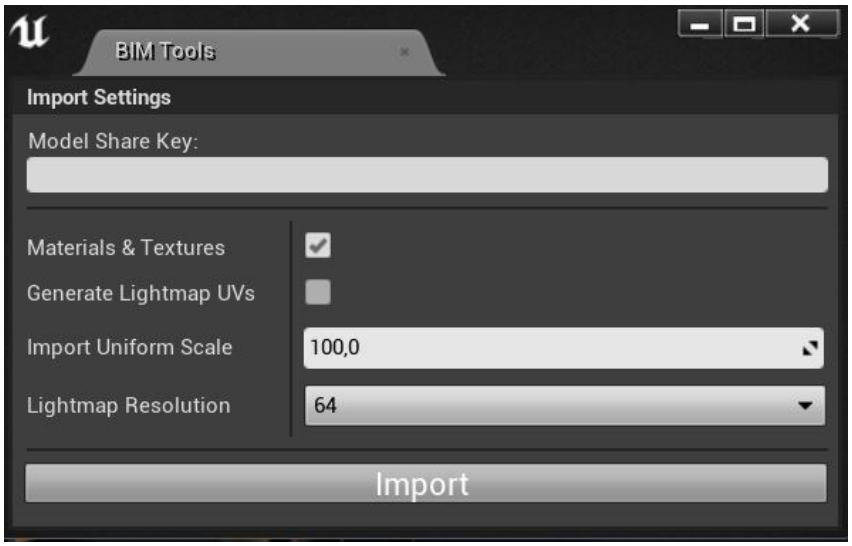
ja Python-moduulit. Lisäksi tiedostoon, jossa Python-skriptin ajetaan, täytyy lisätä PythonScriptPlugin.h-tiedosto. Python-skriptin ajamista varten lisäosaan luotiin ExecutePythonScript-metodi, joka ottaa argumenttina Datasmith-glTF-tuojan parametrejä mallien tuontia varten. ExecutePythonScript-metodissa ajetaan DatasmithGLTFImport.py-skripti ExecPythonCommand- (esimerkkikoodi 9) komennolla.

```
FPythonScriptPlugin::Get()->ExecPythonCommand(*PythonScript);
```

Esimerkkikoodi 9. Python-skriptin ajaminen C++:lla käyttäen PythonScriptPlugin-luokan ExecutePythonCommand-metodia.

4.8 Lisäosan käyttöliittymän toteutus

BIMTools-lisäosan käyttöliittymä toteutettiin kokonaan UMG:llä. Luotiin Unreal Editorissa uusi Editor Utility Widget, johon lisättiin käyttöliittymän elementit ja ohjelmoitiin niiden toiminnot. Lisäosan käyttöliittymä koostuu tekstin syöttökentästä, Datasmith-glTF-tuojan parametreista sekä Import-painikkeesta. Tekstin syöttökenttään syötetään mallin ShareKey, parametreillä määritetään glTF-tiedostojen tuonnin asetukset ja Import-painiketta painamalla tuodaan ladatut glTF-tiedostot pelimoottoriin. Kuvassa 21 näkyy BIMTools-lisäosan käyttöliittymä.



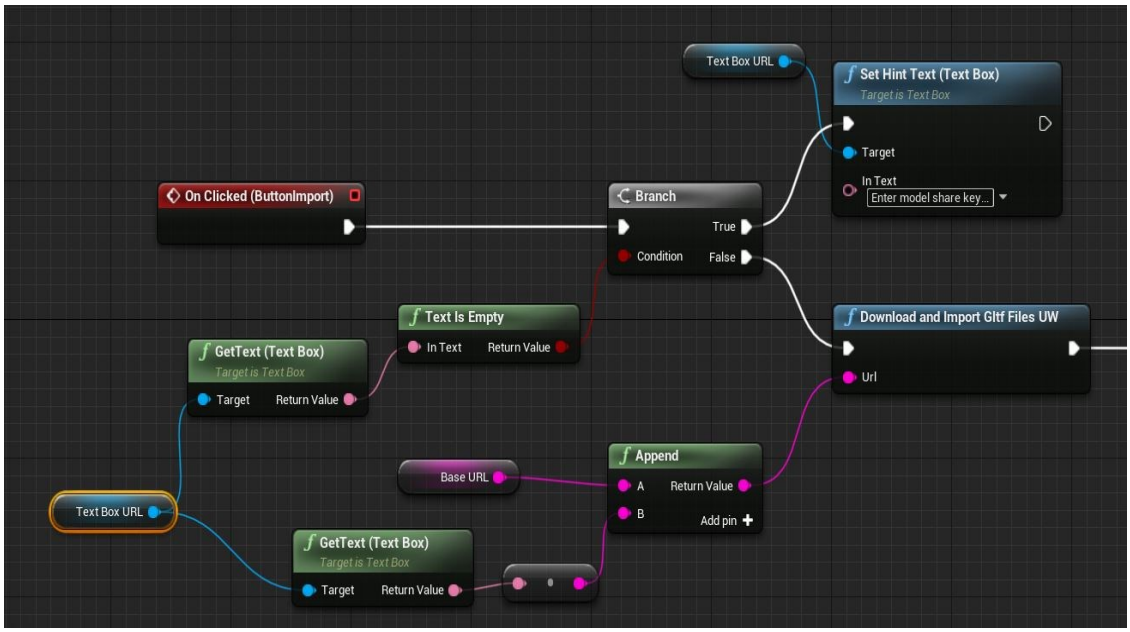
Kuva 21. BIMTools-lisäosan käyttöliittymä.

Käyttöliittymän elementtien toimintojen ohjelmointia varten lisäosaan luotiin uusi C++-luokka nimeltä BIMToolsUWidget, joka perittiin Unrealin EditorUtilityWidget-luokasta. Tämän luokan avulla heijastetaan lisäosaan toteutetut metodit käyttämällä Unrealin heijastusjärjestelmän makroja. Tällä tavalla saadaan lisäosaan toteutettuja C++-metodeja kutsuttua Blueprintissä. Esimerkkikoodissa 10 on BIMToolsUWidget-luokaan luotu DownloadAndImportFiles_UW-metodi, joka heijastetaan Unrealin heijastusjärjestelmällä.

```
UFUNCTION(BlueprintCallable, category = "BIMTools")
static void DownloadAndImportGltfFiles_UW(FString Url);
```

Esimerkkikoodi 10. DownloadAndImportGltfFiles_UW-metodi, joka heijastetaan käyttämällä Unrealin heijastusjärjestelmän UFUNTION()-makroa.

Lisäosan käyttöliittymän elementtien toiminnot ohjelmoitiin kokonaan Blueprintissä. Kuvassa 22 näkyvät käyttöliittymän tekstin syöttökentän ja Import-painikkeen toiminnat Blueprintissä. Tekstin syöttökentästä tarkastetaan, onko se tyhjä vai ei, ja jos se on tyhjä, pyydetään käyttäjää syöttämään mallin ShareKey, minkä jälkeen käyttäjä voi ladata mallit Import-painiketta painamalla. Import-painiketta painamalla kutsutaan DownloadAndImportGltfFiles_UW-metodi, jonka tehtävä on ladata glTF-tiedostot ja tuoda ne pelimoottoriin.



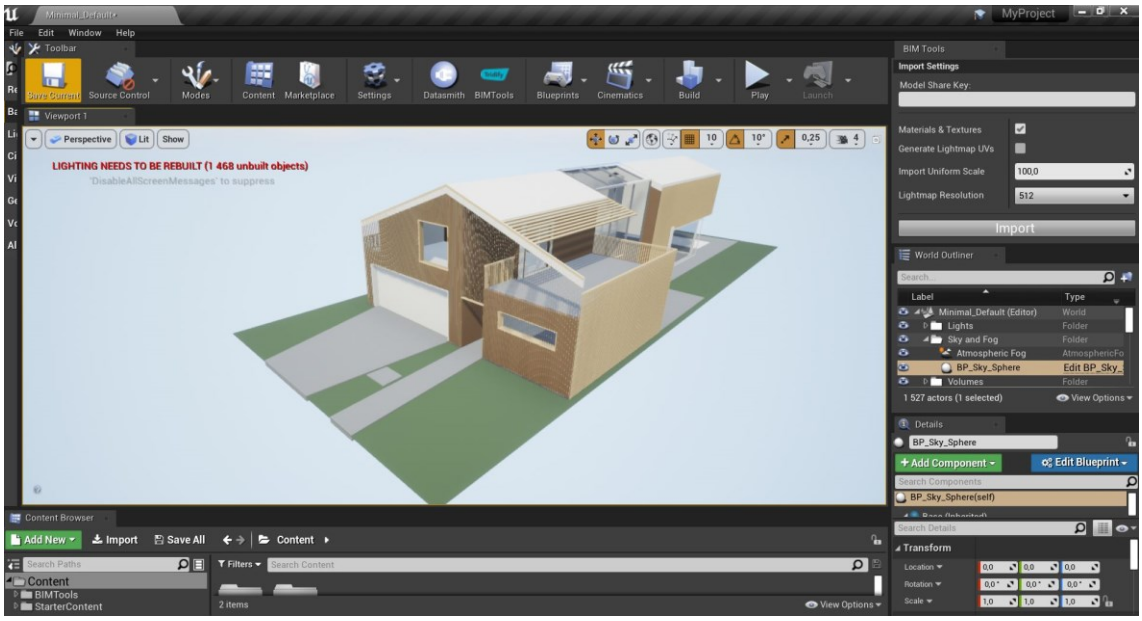
Kuva 22. Tekstin syöttökentän ja Import-painikkeen toiminnat Blueprintissä.

Lisäosan käyttöliittymä saadaan näkyviin Unreal Editorin päätyöpalkista BIMTools-lisäosan painiketta painamalla. Unreal Editorin päätyöpalkista lisäosan painiketta painettaessa kutsutaan lisäosaan toteutettua PluginButtonClicked-metodia, jonka avulla saadaan lisäosan käyttöliittymä näkyviin. Käyttöliittymän voi myös helposti vetää ja kiinnittää Unreal Editorin muihin ikkunoihin tai tehdä siitä oman ikkunansa.

5 Yhteenveto

Opinnäytetyön tekeminen aloitettiin tutkimalla lisäosan kehittämiseen käytettyjä työkaluja ja teknologioita. Tutustuttiin Unreal Engine 4 -pelimoottoriin, Unreal Editorin laajentamiseen ja Unreal-lisäosien kehittämiseen.

Tutkimusosuuden jälkeen aloitettiin projektiosuus suunnittelemalla lisäosan eri komponentit. Lisäosalle asetettiin minimivaatimukset, suunniteltiin lisäosan käyttöliittymä ja tutustuttiin erilaisiin Unreal Engine 4:n glTF-tuojiin. Suunnitteluvaiheen jälkeen siirryttiin toteutusvaiheeseen, jossa toteutettiin lisäosan komponentit Unreal Engine 4 -pelimoottoriin asetettujen vaatimusten mukaisesti (kuva 23).



Kuva 23. Esimerkki 3D-mallista, joka tuotiin pelimoottoriin ja visualisoitiin pelimoottorissa BIMTools-lisäosaa käyttäen.

Unreal Editorin laajentamiseen ja Unreal-lisäosien kehittämiseen internetistä löytyi todella vähän tukea. Tästä syystä lisäosan toteutuksessa tuen saaminen ongelmiin tuntui välillä hankalalta. Usein jouduttiin tutkimaan ja kokeilemaan asioita parhaiden ratkaisujen löytämiseksi. Tästä huolimatta opinnäytetyön tuloksena saatiin asetettujen vaatimusten mukaan toimiva lisäosa.

Lähteet

1 Dealessandri, Marie. 2020. What is the best game engine: is Unreal Engine right for you? Verkkoaineisto. <https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unreal-engine-4-the-right-game-engine-for-you>. Luettu 5.3.2021.

2 Drake, Jeff. 2020. 20 Great Games That Use The Unreal 4 Game Engine. Verkkoaineisto. <https://www.thegamer.com/great-games-use-unreal-4-game-engine/>. Luettu 6.3.2021.

3 Online Subsystem. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Online/>. Luettu 6.3.2021.

4 Tools And Editors. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/Basics/ToolsAndEditors/>. Luettu 10.3.2021.

5 Level Editor. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LevelEditor/>. Luettu 11.3.2021.

6 Blueprint Editor Reference. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Editor/>. Luettu 12.3.2021.

7 Graph Editor Tab. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Editor/UIComponents/GraphEditor/>. Luettu 14.3.2021.

8 Blueprint Debugging. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Debugging/>. Luettu 17.3.2021.

9 UMG UI Designer User Guide. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/UMG/UserGuide/>. Luettu 20.3.2021.

10 Buttice, Claudio. 2020. C plus plus Programming Language (C++). Verkkoaineisto. <https://www.techopedia.com/definition/26184/c-programming-language>. Luettu 25.3.2021.

11 Gillis, Alexander S. & Lewis, Sarah. 2020. Object-oriented programming (OOP). Verkkoaineisto. <https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP>. Luettu 26.3.2021.

12 Introduction to C++ Programming in UE4. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/en-US/ProgrammingAndScripting/ProgrammingWithCPP/IntroductionToCPP/index.html>. Luettu 1.4.2021.

13 C++ vs. Blueprints: pros and cons, which should be used, and when? 2020. Verkkoaineisto. ID Tech. <https://www.idtech.com/blog/c-vs-blueprints-differences>. Luettu 2.4.2021.

14 Introduction to Blueprints. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/>. Luettu 5.4.2021.

15 Guidelines for Programming for Blueprints. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/TechnicalGuide/Guidelines/>. Luettu 7.4.2021.

16 UE4 Blueprints From Hell. 2017. Verkkoaineisto. Tumblr. <https://blueprintsfromhell.tumblr.com/post/162871687086/submission-from-discord>. Luettu 8.4.2021.

17 Blueprint Best practices. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/BestPractices/>. Luettu 9.4.2021.

18 Types of Blueprints. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Types/>. Luettu 12.4.2021.

19 Blueprint Class. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Types/ClassBlueprint/>. Luettu 14.4.2021.

20 Unreal Property System (Reflection). Verkkoaineisto. Epic Games. <https://www.unrealengine.com/en-US/blog/unreal-property-system-reflection>. Luettu 16.4.2021.

21 What is Python? A Comprehensive Guide. 2020. Verkkoaineisto. Netguru. <https://www.netguru.com/what-is-python>. Luettu 25.4.2021.

22 Scripting the Editor using Python. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/en-US/ProductionPipelines/ScriptingAndAutomation/Python/index.html>. Luettu 27.4.2021.

23 Unreal Python API Documentation. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/PythonAPI/>. Luettu 29.4.2021.

24 Datasmith Overview. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/Overview/>. Luettu 5.5.2021.

25 Simkin, Aliaksei. 2019. glTF – Behind the scene of 3D Magic. Verkkoaineisto. <https://stayrelevant.globant.com/en/gltf-behind-the-scene-of-3d-magic/>. Luettu 8.5.2021.

26 Plugin. Verkkoaineisto. Seobility Wiki. <https://www.seobility.net/en/wiki/Plugin>. Luettu 10.5.2021.

27 Plugins. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/Plugins/>. Luettu 12.5.2021.

28 UnrealBuildTool. Verkkoaineisto. Epic Games. <https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/BuildTools/UnrealBuildTool/>. Luettu 14.5.2021.