jamk.fi

Nivelletyn kaivoskoneen animointi Unreal Engine 4 -pelimoottorilla

Mikko Juutilainen

Opinnäytetyö
Toukokuu 2015

Ohjelmointitekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t)	Julkaisun laji	Päivämäärä
Juutilainen, Mikko	Opinnäytetyö	5.5.2015
	Sivumäärä 67	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
Nivelletyn kaivoskoneen animointi Unreal Engine 4 -pelimoottorilla

Koulutusohjelma
Ohjelmointitekniikan koulutusohjelma

Työn ohjaaja(t)
Pietikäinen, Kalevi

Toimeksiantaja(t)
Cybercube Oy

Tiivistelmä

Opinnäytetyössä toteutettiin Cybercube Oy:lle Unreal Engine 4-pohjaisiin kaivossimulointeihin helposti integroitava nivelletyn kaivoskoneen malli. Animoitujen kaivoskoneiden tulisi kyetä seuraamaan ennalta määrättyä reittiä sekä suorittamaan konetyypille ominaisia toimintoja. Tavoitteena oli luoda toimintamallit kahdelle eri kaivoskonetyypille sekä testiympäristö niiden toiminnan testaamista varten. Työssä yritettiin hyödyntää mahdollisimman paljon Unreal Enginen valmiiksi tarjoamia komponentteja.

Kaivoskoneiden 3D-mallit valmisteltiin 3DS Max–mallinnusohjelmalla rakentamalla niihin Unreal Enginen ymmärtämä luurankohierarkia. Tämän hierarkian pohjalta luotiin Unreal Editorissa koneiden fysiikka- ja animointimallit. Kaivoskoneiden yleinen kantaluokka toteutettiin C++ -kielellä ja se periytettiin Unreal Enginen WheeledVehicle-luokasta. Tästä kantaluokasta periytettiin edelleen nivellettyjen koneiden kantaluokka sekä tyyppikohtaiset C++ - ja Blueprint-kantaluokat. Suurin osa koneiden toimintalogiikasta rakennettiin ActorComponent-luokasta perittyihin komponentteihin Unreal Enginen sisältämällä Blueprints Visual Scripting -järjestelmällä. Koneiden animointi toteutettiin konetyyppikohtaisesti Animation Blueprinteillä.

Työn tuloksena syntyivät toimivat kauhakuormaaja- ja kippiautotyyppisten nivellettyjen kaivoskoneiden mallit. Ne seurasivat onnistuneesti annettua splini-muotoista reittiä sekä suorittivat konetyyppikohtaisia animaatioita. Kaivoskoneiden simulointimallia kehitetään ja tarkennetaan edelleen jatkossa.

Avainsanat (asiasanat)
Unreal Engine, 3D, Blueprint, C++, Luokka, Luuranko, Hierarkia, Mallinnus, Simulointi, Animointi, 3DS Max, Visuaalinen, Skriptaus

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
Juutilainen, Mikko	Bachelor´s Thesis	5.5.2015
	Pages 67	Language Finnish
		Permission for web publication (X)

Title

Animating an articulated mining vehicle in Unreal Engine 4

Degree Programme

Software Engineering

Tutor(s)

Pietikäinen, Kalevi

Assigned by

Cybercube Oy

Abstract

The thesis focuses on creating a model of an articulated mining vehicle for Cybercube Oy. The model should be easily integrated to the Unreal Engine 4 based mining simulations. The animated mining vehicles should be able to follow a pre-defined path and perform actions typical for the vehicle. The scope of the thesis was to create operating models for two different types of vehicles, and to create a testing environment to test how they operate. The attempt was to maximize the use of components provided by the Unreal Engine.

The 3D models of the mining vehicles were modified in the 3DS Max modeling software to build them a skeletal hierarchy supported by the Unreal Engine. Based on this hierarchy, the assets for physics and animations were created in the Unreal Editor. The general base class of the mining vehicle was implemented in C++ and it was inherited from the Unreal Engine's AWheeledVehicle class. The base class for articulated vehicles and vehicle type-specific C++ and Blueprint base classes were further inherited from this base class. Most of the operational logic of the vehicles was built into separate components based on the ActorComponent class using the Unreal Engine's Blueprints Visual Scripting system. The animation of the vehicles was implemented with the type-specific Animation Blueprints.

The thesis resulted in working models of an articulated loader and a dumper truck. They successfully followed the given spline route and performed type-specific animations. The simulation models of the mining vehicles will be further improved in the future.

Keywords

Unreal Engine, 3D, Blueprint, C++, Class, Skeletal, Hierarchy, Modeling, Simulation, Animation, 3DS Max, Visual, Scripting

Miscellaneous

Sisältö

Kuviot

SANASTO

AUTODESK MAYA

Alias Systems Corporationin alun perin kehittämä, nykyisin Autodeskin omistama suosittu 3D-mallinnusohjelma.

ACTOR

Tässä opinnäytetyössä paljon esiintyvällä termillä actor tarkoitetaan yleisesti Unreal Engine 4:n Actor-luokasta periytyvää objektia. Actor-luokkaan viitattaessa sanan perässä esiintyy aina termi luokka.

BLENDER

Blender on ilmainen avoimen lähdekoodin 3D-mallinnusohjelma. Mallinnusominaisuuksien ohessa Blenderissä on myös integroitu pelimoottori.

BVS

Tässä opinnäytetyössä BVS on lyhenne usein esiintyvästä Blueprint Visual Scripting-termistä. Blueprint Visual Scripting on Blueprint-luokissa käytettävä visuaalinen skriptikieli, jolla toteutettiin suurin osa työn kohteena olevien kaivoskoneiden toimintalogiikasta.

IDE (Integrated Development Environment)

Integroitu kehitysympäristö on yhdestä tai useammasta osasta koostuva paketti, jonka tarkoituksena on auttaa ohjelmoijia suunnittelemaan ja toteuttamaan ohjelmistoja. Yksinkertaisimmillaan IDE sisältää vain tekstieditorin ja kääntäjän, joka tukee yhtä tai useampaa ohjelmointikieltä. Nykyaikaiset integroidut kehitysympäristöt ovat parhaimmillaan todellisia monitoimityökaluja. Ne sisältävät muun muassa tuen usealle eri ohjelmointikielelle, monipuoliset koodin analysointityökalut, ennakoivan tekstinsyötön, syntaksikorostuksen, työkalut erilaisten käyttöliittymien suunnitteluun ja toteutukseen sekä monipuoliset virheenkorjaustyökalut.

KEYFRAME-ANIMAATIO

Keyframe-animaatio on animaatiotekniikka, jossa avainkehys (engl. keyframe) esittää halutun animaation alku- ja lopputilan. Animaatio syntyy, kun näiden tilojen välillä tehdään tasainen siirtymä. Keyframe-animaatio on edelleen suosittu animaation muoto, mutta se soveltuu huonosti dynaamiseen animaatioon, jossa esiintyy paljon muuttuvia tekijöitä.

LUU

3D-grafiikassa luut (engl. bone) ovat virtuaalisia kappaleita, jotka siirtävät niihin sidottuja verteksejä. Luut mahdollistavat mallin osien, kuten raajojen, helpon liikuttelun ilman tarvetta käsitellä yksittäisiä verteksejä. Luiden liitoskohtia nimitetään niveliksi (engl. joint).

LUURANKO

Luuranko on virtuaalinen luusto (engl. skeleton), joka yhdistää halutulla tavalla asemoidut luut hierarkkiseen järjestykseen. Iso osa nykyaikaisten tietokonepelien animoinnista perustuu virtuaalisten luurankojen liikutteluun.

MESH

Mesh on monitahokas 3D-grafiikassa, jonka muodon määrittää kokoelma verteksejä, reunaviivoja ja polygoneja. Arkikielessä näistä monitahokkaista puhuttaessa tarkoitetaan yleensä 3D-mallia. Kun malliin lisätään luuranko animointia varten, siitä käytetään yleisesti termiä Skeletal Mesh.

PELIMOOTTORI

Pelimoottori on tietokonepelien tekemiseen tarkoitettu sovelluskehys, joka tyypillisesti sisältää pelinkehityksen kannalta oleelliset toiminnot kuten 2D- tai 3D-renderöijän, fysiikkamoottorin (ja törmäysmallinnuksen), äänien soittamisen, skriptauksen, animoinnin, tekoälyn sekä pelimaailman ja verkkoliikenteen hallinnan. Pelimoottori sisältää usein uudelleen käytettävien ohjelmakomponenttien lisäksi myös graafisen ohjelmointiympäristön. Tunnettuja ja yleisesti käytössä olevia pelimoottoreita ovat muun muassa CryEngine, Unity ja Unreal Engine, joista viimeiseksi mainittu on käytössä tässä opinnäytetyössä.

POLYGONI

Monikulmio eli polygoni tarkoittaa 3D-grafiikassa kolmesta tai useammasta verteksistä muodostuvaa kappaletta, jonka valitusta pisteestä reunaviivaa (engl. edge) seuraamalla päästään lopulta takaisin aloituspisteeseen.

SPLINI

Tietokonegrafiikassa splini (engl. spline) tarkoittaa matemaattisella yhtälöllä muodostettua pehmeää kurvia, joka kulkee kahden tai useamman kontrollipisteen kautta. Kaksi yleisintä splinin tyyppiä ovat Bézier-käyrät (Bézier curve) ja B-splinit (B-spline).

REFLEKTIO

Tietojenkäsittelytieteessä reflektio (engl. reflection) tarkoittaa tietokoneohjelman kykyä tutkia ja muokata itseään ajonaikaisesti. Näin ollen kaikkien ominaisuuksien, kuten luokkien, muuttujien tai funktiokutsujen ei tarvitse olla tiedossa ohjelman kääntämisen aikana, vaan ohjelmaan voi muun muassa lisätä niitä suorituksen aikana dynaamisesti tarvittaessa.

TEKSTUROINTI

Tekstuuri (engl. texture) tarkoittaa tietokonegrafiikassa bittikarttakuvaa, jonka voi kartoittaa polygonin pinnalle (texture mapping). Näin 3D-malliin saadaan lisää yksityiskohtaisuutta ja väriä. Tätä toimenpidettä kutsutaan teksturoinniksi.

VERTEKSI

Verteksi (engl. vertex) tarkoittaa 3D-grafiikassa yksittäistä polygonin reunapistettä, joka sijaitsee polygonin kahden reunan leikkauspisteessä.

1 OPINNÄYTETYÖN LÄHTÖKOHDAT

Opinnäytetyön tarkoituksena oli toteuttaa nivelletyn kaivoskoneen animointimalli kehitteillä olevaan, Unreal Engine 4-pelimoottorin päälle kehitettävään kaivossimulointiohjelmistoon. Tarkoituksena ei ollut rakentaa fysikaalisesti oikeaa mallia, vaan pohja, jonka päälle voidaan jatkossa lisätä yksityiskohtia ja uutta toiminnollisuutta. Näin malli tarkentuu asteittain lähemmäksi simuloinnissa käytettävää lopullista versiota. Animointimalli kehitettiin lähtökohtaisesti hyödyntämällä mahdollisimman paljon Unreal Enginen valmiiksi tarjoamia komponentteja. Kaivoskoneiden tuli kyetä seuraamaan mallin avulla valmiiksi laskettua reittiä ja koneiden tuli taittua keskinivelen kohdalta niiden kääntyessä kurveissa.

Tavoitteena oli rakentaa mallit kahdelle erityyppiselle, nivelletylle kaivoskoneelle. Molempien mallien tuli pystyä suorittamaan konetyypille ominaisia toimintoja, kuten nostovarren nosto tai kippaaminen. Malleista oli tavoitteena tehdä sellaiset, että uusien kaivoskonetyyppien ja kaivoskoneiden lisääminen järjestelmään on helppoa tulevaisuudessa olemassa olevien mallien pohjalta. Mallien testaamista varten tarvittiin myös testiympäristö, jossa koneiden liikkeitä ja toimintoja voitiin testata.

Työn teoriaosuudessa avataan tarkemmin malleilta vaadittuja ominaisuuksia sekä esitellään lyhyesti kaivossimuloinnin teoriaa, jotta lukija saa paremman käsityksen mallien käyttötarkoituksesta. Teoriaosuudessa esitellään myös luurankopohjaista animointia sekä Unreal Engine 4-pelimoottoria. Toteutusosassa käsitellään kaivoskoneiden 3D-mallinnusta sekä ajoneuvo-tyyppisen nivelletyn kaivoskoneen toteutusta Unreal Engine 4:llä.

Opinnäytetyön tilaajana toimi Cybercube Oy, joka tarjoaa 3D-grafiikkasovelluksia sekä ohjelmistoratkaisuja muun muassa prosessinhallintaan, simulointiin ja logistiikan optimointiin. Yritys toimii usealla eri toimialalla mukaan lukien kaivosteollisuus, satamatoiminta, liikenne / logistiikka ja tehdasteollisuus.

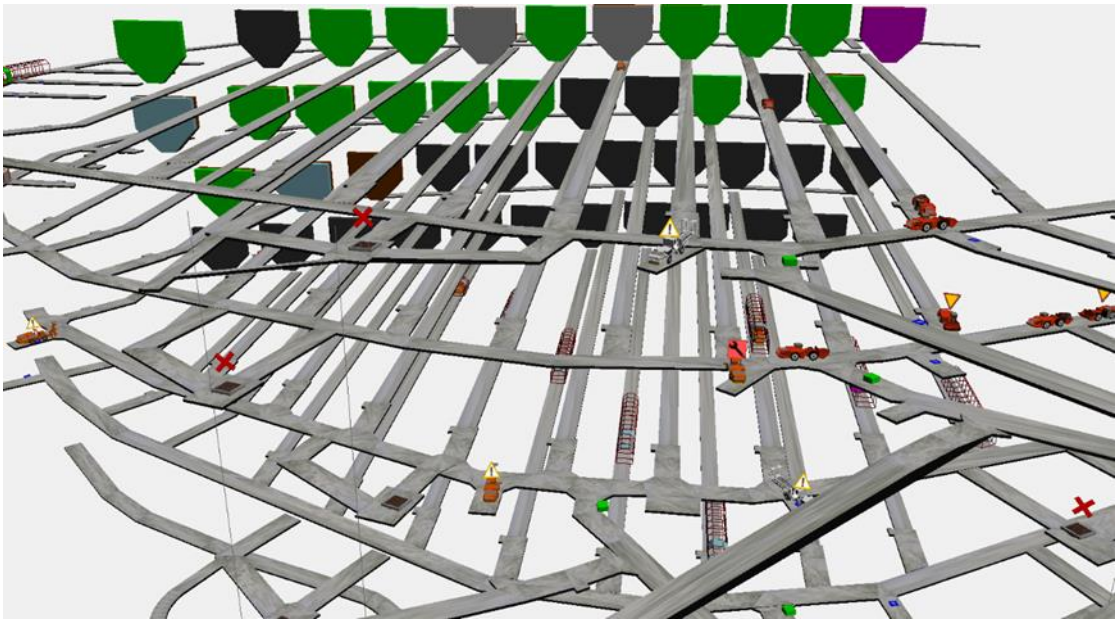
2 KAIVOSSIMULOINTI

Tietokonesimuloinnin tarkoituksena on jäljitellä reaalimaailman ilmiöitä ja tapahtumia tietokoneohjelman avulla. Simulointiohjelmalle annetaan tunnettua dataa, ja ohjelma tuottaa sen perusteella uutta dataa, jonka avulla voidaan tehdä erilaisia ennusteita tai päätelmiä. Melkein kaikkea, mitä voidaan laskea, voidaan myös simuloida. Simulointeja käytetään muun muassa logistiikan optimointiin, tieteelliseen mallinnukseen, koulutusvälineenä ja markkinoinnin apuvälineenä. Monet laitetoimittajat haluavat osoittaa simulaation avulla, kuinka hyvin heidän tuotteensa ratkaisee asiakkaan ongelmia. On paljon havainnollisempaa näyttää visuaalisesti toiminnan sujuvuutta, kuin esimerkiksi esittää numeerisia taulukoita tai erilaisia kuvaajia. Tämän tyyppinen data onkin yleensä vain tietokonesimuloinnin oheistuote. (Tietokonesimulointi 2014.)

Kaivossimuloinneilla pyritään simuloimaan erilaisia kaivostoiminnan prosesseja, kuten louhintaa. Simuloinnin laajuus voi vaihdella yksittäisen työkoneen simuloimisesta koko kaivoksen täysimittaiseen simulointiin. Tyypillisesti kaivossimulointia käytetään erilaisten suunnitelmien testaamiseen, konekannan optimointiin, järjestelmän pullonkaulojen löytämiseen sekä kaivostoiminnan visualisointiin. Kuten tietokonesimuloinnit yleensä eivät kaivossimuloinnitkaan vastaa täysin reaalimaailmaa. Tietokoneiden laskentatehon kasvaessa simulointien tarkkuus kuitenkin lisääntyy jatkuvasti, ja kaivoskoneetkin saavat tämän myötä lisää laskettavia parametreja teholukujen laskemista varten. (Mt.)

Opinnäytetyö on osa kaivossimulointiohjelmistojen tuotekehitysprosessia. Unreal Engine 4 -pelimoottorilla on suuri vaikutus etenkin kaivossimulointien ulkoasuun tulevaisuudessa. Uudistettu järjestelmä myös luo automaattisesti olemassa olevan pistedatan pohjalta kaivoksen käytäväverkon sekä muodostaa niistä kaivoskoneiden ajoreitit splini-kurveilla. Tästä johtuen opinnäytetyön kaivoskoneiden reitinseurantakin perustuu splini-kurveihin.

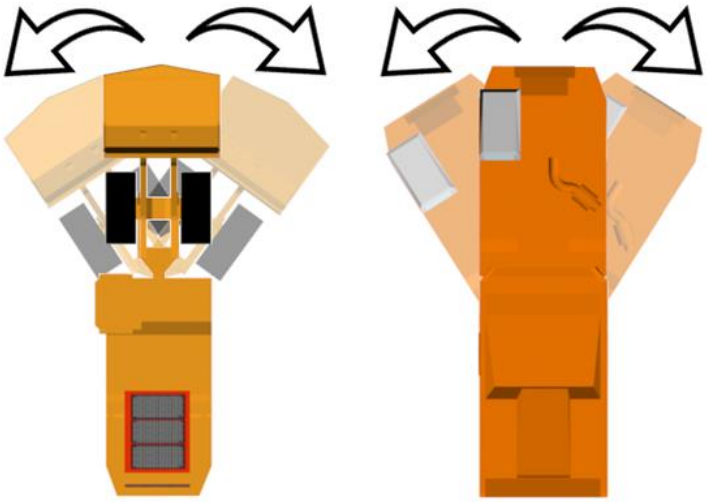
Kuviossa 1 on kuvakaappaus työn tilaajan nykyisestä kaivossimulointiohjelmistosta. Kuvaruudulta saa välitöntä visuaalista palautetta eri komponenttien tiloista, ja hiiren kursorin yliviennillä niistä saa vielä tarkempia tietoja.



Kuvio 1. Cybercube Mining Simulation

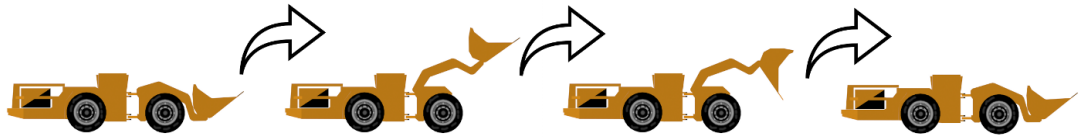
3 NIVELLETYT KAIVOSKONEET

Opinnäytetyössä nivelletyllä kaivoskoneella tarkoitetaan nelipyöräistä ajoneuvoa, jonka runko on jaettu nivelellä kahteen osaan. Ajoneuvo kääntyy taittumalla nivelen kohdalta, eikä siinä ole lainkaan kääntyviä renkaita. Työssä kaivoskoneiden täytyy pystyä seuraamaan splinikurveista muodostuvaa reittiä siten, että ne taittuvat kääntyessään luonnollisen näköisesti. Kuviossa 2 esitetään havaintokuvat sekä kauhakuormaajan että kippiauton taittumisesta keskinivelen kohdalta.



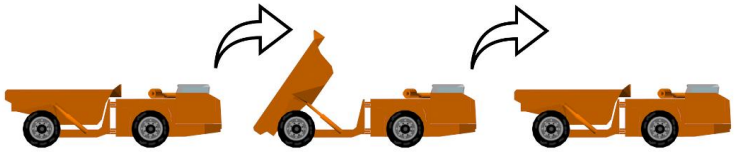
Kuvio 2. Kaivoskoneiden taittuminen nivelen avulla

Jokaisella kaivoskonetyypillä on omat toimintonsa, joiden pohjalta muodostetaan toiminta-animaatiot. Kauhakuormaajan tapauksessa vaaditut toiminta-animaatiot ovat nostovarren nosto, kauhan kippaus ja nostovarren sekä kauhan palautus alkuasentoon. Kuviossa 3 on havainnollistettu kauhakuormaajan toiminta-animaatioiden kierto.



Kuvio 3. Kauhakuormaaja-tyyppisen kaivoskoneen toiminta-animaatiot

Kippiauto-tyyppiseltä kaivoskoneelta vaaditut toiminta-animaatiot ovat kippilavan nosto ja sen palautus alkuasentoon. Kuviossa 4 on havainnollistettu kippiauton toiminta-animaatioiden kierto.



Kuvio 4. Kippiauto-tyyppisen kaivoskoneen toiminta-animaatiot

4 SKELETAL ANIMATION

Skeletal animation on paljon käytetty tietokoneanimoinnin tekniikka, jossa animoinnin kohde on jaettu kahteen osaan: näkyvä osa (skin tai mesh) sekä yleensä sen sisällä oleva hierarkkinen joukko toisiinsa yhdistettyjä luita (skeleton tai rig). Luita liikuttamalla toteutetaan näkyvän osan animointi. Tekniikkaa käytetään usein luontokappaleiden, kuten ihmisten, luonnollisen oloisen liikkeen mallinnukseen, mutta sitä voidaan käyttää myös muiden kappaleiden animointiin – muun muassa nivelletyn kaivoskoneen. (Skeletal animation 2014.)

Skeletal animation -tekniikassa polygonimallin virtuaalisen rangan rakennusprosessiin viitataan englanninkielisellä termillä ”Rigging”. Rangan jokainen luu yhdistetään useimmiten polygonimallin näkyvän osan haluttuun verteksiryhmään. Esimerkiksi kauhakuormaajan nostovarren verteksit yhdistetään sen nostoliikkeen animointiin

tarkoitettuun luuhun. Tähän luun ja verteksiryhmän yhdistävään prosessiin viitataan englanninkielisellä termillä ”Skinning”. (Mt.)

Polygonimallin pisteet yhdistetään luihin painoarvoilla (weight). Painoarvo kertoo skaalan, jolla piste seuraa tiettyä luuta. Useimmiten painoarvo on reaaliluku väliltä 0-1. Painoarvon ollessa 0 luun transformaatio ei vaikuta pisteeseen lainkaan, kun taas painoarvon ollessa 1, piste seuraa täysin luun transformaatiota. Piste voi saada painoarvoa useasta luusta, mutta tällöinkin painoarvojen summa on korkeintaan 1. Skaalojen suhde kertoo, missä suhteessa piste mitäkin luuta seuraa.

Tekniikan etuna on se, että animaatio voidaan muodostaa yksinkertaisemmin liikuttamalla luita sen sijaan, että liike määriteltäisiin liikuttamalla erikseen polygonimallin pisteitä. Tästä on myös se etu, että yhteneväisen luurankohierarkian sisältäviä malleja voidaan liikutella samoilla animaatiomäärittеillä, vaikka niiden näkyvät osat poikkeaisivatkin toisistaan. Tekniikan tarkoituksena ei kuitenkaan koskaan ole jäljitellä oikeaa anatomiaa tai fysiikan prosesseja, vaan ainoastaan kontrolloida mallin muodonmuutosta luiden transformaatiota kontrolloivien muuttujien vaikutuksesta. (Skeletal animation 2014.)

Jokaisella luulla on oma kolmiulotteinen transformaationsa, johon sisältyy luun paikka, skaala ja orientaatio. Vaihtoehtoisesti luulla voi olla myös isäntä-luu (parent bone) ja joukko lapsi-luita (child bones). Luut muodostavatkin hierarkian, jossa lapsisolmun (node) lopullinen transformaatio muodostuu sen oman transformaation ja isäntä-transformaation yhteisvaikutuksesta. Tämän seurauksena muun muassa kauhakuormaajan nostovarren luun liike saa kauhan liikkumaan sen mukana. (Mt.)

Nivelten asentojen laskenta määritellään usein kahden eri kinemaattisen yhtälöryhmän avulla. Näihin viitataan englanninkielisillä termeillä ”Forward Kinematics” ja ”Inverse Kinematics”. Ensimmäisessä nivelketjua liikutellaan kantaluusta lähtien hierarkkisesti, kunnes päästään haluttuun asentoon. Jälkimmäinen kääntää nimensä mukaisesti laskennan käänteiseksi. Siinä loppuasento on määritelty etukäteen, ja Inverse Kinematics laskee nivelien liikkeet, joilla tähän asentoon päästään. Skeletal animationilla viitataan forward kinematics-osaan, ja sitä käytetään myös opinnäytetyön kaivoskoneiden animoinnissa. Niiden kanssa haluttuun asentoon päästään aina muutaman nivelen hierarkkisella ketjutuksella. (Skeletal animation 2014.)

5 UNREAL ENGINE 4

5.1 Yleistä

Unreal Engine on Epic Gamesin vuodesta 1998 asti kehittämä, laajasti peliteollisuudessa käytössä oleva pelimoottori (Unreal Engine 2015). Unreal Engine 4 on kirjoitushetkellä pelimoottorin uusin julkaisu. Siitä on opinnäytetyössä käytössä versio 4.7.3, johon Unreal Engine 4:ää koskeva teksti perustuu. Unreal Engine 4:n käyttö on ollut maaliskuusta 2015 asti maksutonta, ja Epic Gamesin ansaintamalli perustuu sillä tehtyjen tuotteiden menestykseen (5% $3000USD vuosineljänneksellä ylittävästä osuudesta/tuote). Unreal Engine 4:n mukana tulee sen täysi C++ -lähdekoodi, joka mahdollistaa käytännössä myös omien lisäysten ja korjausten tekemisen pelimoottoriin. (Sweeney 2015.)

Unreal Engine 4 -kehitystyökalut ovat saatavilla sekä Microsoftin Windows PC- että Applen Mac OS X-alustoille. Tuetut C++ -ohjelmointiympäristöt ovat Windowsilla Microsoftin Visual Studio 2013 ja OS X:llä Applen Xcode. Unreal Engine sisältää tuen lähes kaikille suosituimmille kohdealustoille, joita ovat muun muassa Windows PC, Mac OS X, iOS, Android, Xbox One, PS4, Linux, HTML5 ja SteamOS. Se sisältää myös tuen tuloaan tekeville virtuaalitodellisuuslaitteille (FREQUENTLY ASKED QUESTIONS (FAQ) n.d.).

Unreal Engine 4:ssä on DirectX 11-pohjainen grafiikan renderöintijärjestelmä, joka tukee muun muassa laskennallista varjostusta (deferred shading), globaalia valaistusta (global illumination), läpikuultavaa valaistusta (lit translucency) ja vektorikenttiä hyödyntävän hiukkassimuloinnin (particle simulation) näytönohjaimen grafiikkasuorittimella (GPU). Pelimoottorissa on paljon valmiiksi toteutettuja grafiikkaefektejä kuten Ambient Occlusion, Ambient Cubemaps, Bloom, Lens Flare, Tone Mapping, Vignette ja Depth of Field. (Rendering Overview n.d.)

Unreal Engine 4 käyttää Nvidian näytönohjainten laskentakapasiteettia hyödyntävää PhysX 3.3-fysiikkamoottoria fysiikan simuloinnin laskentaan. Sen avulla on toteutettu fyysisten objektien törmäysmallinnus ja vuorovaikutus. Pelimoottorista löytyy myös tuki Nvidian APEX PhysX Lab-työkalulla luoduille hajotettaville malleille (APEX Destructibles) ja vaatteiden simuloinnille (APEX Cloth) (APEX n.d). (Physics Simulation n.d.)

5.2 Unreal Editor

5.2.1 Yleisnäkymä

Kuviossa 5 näkyy Unreal Editorin yleisnäkymä. Keskellä on hallitseva näkymä 3D-maailmaan, johon voi luoda uusia objekteja ja jossa niitä voi valita, siirtää ja muokata niiden ominaisuuksia. Yläreunassa on työkalurivi, jossa ovat napit usein käytetyille toiminnoille, kuten tallennus, valaistuksen koostaminen, C++ -koodin kääntäminen ja simulaation käynnistys. Editorin vasemmassa reunassa näkyy tilapaneeli (mode pane), josta voi vaihtaa editorin tilaa. Alareunassa on tärkeä Content Browser, josta pääsee käsiksi sekä editorin ulkopuolelta tulevaan sisältöön että erityiseen sisältöön, kuten materiaalit tai Blueprint-luokat. Oikeassa reunassa on ylhäällä World Outliner, jossa näkyvät kaikki maailmaan luodut objektit. Objekteja voi myös muun muassa organisoida kansioihin ja etsiä nimellä. World Outliner -paneelin alapuolella on Details-paneeli, jonka kautta voi muokata valittuna olevan objektin julkisia ominaisuuksia.

Kuvio 5. Testiympäristö kaivoskoneiden testaamista varten

Kaivoskoneiden testiympäristö rakennetaan editorin yleisnäkymässä. Content Browserista vedetään tunnelireitin ja pelkistetyn reitin Blueprint-luokista instanssit 3D-näkymään muokattavaksi. Myös kaivoskoneiden Blueprint-luokista Toro0011_BP ja Toro50_BP vedetään instanssit maailmaan ja niille määritellään edellä mainitut reitit seurattavaksi.

5.2.2 Assetit

Unreal Enginessä maailman sisällön yksittäistä osaa tai resurssia kutsutaan useimmiten Assetiksi. Osa ohjelman tarvitsemasta perussisällöstä, kuten valaistus, luodaan Unreal Editorissa. Suurin osa sisällöstä, kuten 3D-mallit, äänet ja tekstuurit, luodaan kuitenkin erillisillä työkaluilla ja tuodaan editoriin importoimalla, jolloin niistäkin tulee Assetteja. Assetteja voi kopioida projektista toiseen Migrate-työkalun avulla.

Staattinen malli (Static Mesh)

Staattinen malli on polygoneista koostuva geometrinen kappale, joka voidaan siirtää näytönohjaimen välimuistiin ja piirtää tehokkaasti näytönohjaimen toimesta. Koska staattiset mallit ovat välimuistissa, niiden verteksejä ei voida animoida. Staattisia malleja voidaan ainoastaan siirtää, pyöritellä ja skaalata. (Static Meshes n.d.)

Unreal Enginessä staattiset mallit ovat maailman geometrian luonnin perusyksiköitä. Ne luodaan ulkoisella mallinnusohjelmalla, kuten 3DSMax. Valmiit 3D-mallit importoidaan Unreal Editoriin Content Browserin kautta paketoitavaksi. Staattiset mallit esiintyvät maailmassa useimmiten StaticMeshActor-komponentin muodossa, joka muodostuu automaattisesti, kun staattisen mallin assetti viedään maailmaan. Staattisia malleja voidaan käyttää myös staattisissa liikkujissa, kuten ovet tai hissit, sekä kiinteissä fysiikkaobjekteissa. Staattisia malleja käytettiin opinnäytetyön tunnelien muodostuksessa sekä testiympäristössä. (Mt.)

Luurankomalli (Skeletal Mesh)

Luurankomallit koostuvat kahdesta osasta, jotka ovat mallin pinnan määrittävät polygonit sekä hierarkkinen joukko toisiinsa yhdistettyjä luita, joita voidaan käyttää polygonien animointiin. Luurankomalleja käytetään Unreal Engine 4:ssä useimmiten esittämään hahmoja tai muita animoituja objekteja. Myös opinnäytetyön kaivoskoneiden mallit ovat luurankomalleja. Polygonimallit, luiden muodostaminen ja verteksien sitominen niihin tehdään ulkoisella mallinnusohjelmalla, kuten 3DSMax. Valmiit luurankomallit importoidaan staattisten mallien tapaan Content Browserin kautta Unreal Editoriin paketoitavaksi. (Skeletal Meshes n.d.)

Materiaali (Material)

Materiaali on Unreal Editorissa luotava assetti, jolla pinnoitetaan malleja ja määritellään sitä kautta paljon niiden visuaalista ulkoasua. Korkealla tasolla ajateltuna materiaaleja voidaan ajatella maaleina, joilla objektit maalataan. Tosiasiassa objektin pinta yritetään kirjaimellisesti saada näyttämään siltä, että se on tehty kyseisestä materiaalista. Materiaalille voidaan määrittää muun muassa väri, tai kuinka kiiltävä tai läpinäkyvä se on. Teknisemmin ajateltuna materiaalia käytetään sen laskemiseen, kuinka valo käyttäytyy osuessaan sillä maalattuun pintaan. Unreal Engine 4:ssä käytetään fyysiseen olemukseen perustuvaa varjostusmallia. Tämä tarkoittaa, että materiaalille voidaan määrittää paremmin todelliseen maailmaan pohjautuvia ominaisuuksia, kuten ”metallinen”, ”peilimäinen” tai ”rosoinen”. Unreal Editorissa on materiaalien muokkaamista varten oma Material Editor. (Materials n.d.)

Tekstuurit (Texture)

Tekstuurit ovat materiaaleissa käytettyjä kuvia, jotka kartoitetaan niille pinnoille, joihin materiaali on asetettu. Tekstuureita voidaan joko käyttää sellaisenaan pinnan värin määrittelyyn tai sen pikseliarvoja voidaan käyttää materiaaleissa maskina tai osana muita laskuja. Tekstuurit luodaan useimmiten ulkoisesti kuvankäsittelyohjelmalla kuten Gimp ja importoidaan Unreal Editoriin. (Textures n.d.)

Yksittäinen materiaali voi käyttää useita tekstuureita, joista jokaista käytetään eri tarkoitukseen. Materiaalilla voi esimerkiksi olla tekstuurit perusvärille, heijastukselle ja normaalikartalle. Lisäksi tekstuurin alpha-kanavaan voi olla varastoituna muun muassa emittointiin ja rosoisuuteen liittyvä kartoitus. (Mt.)

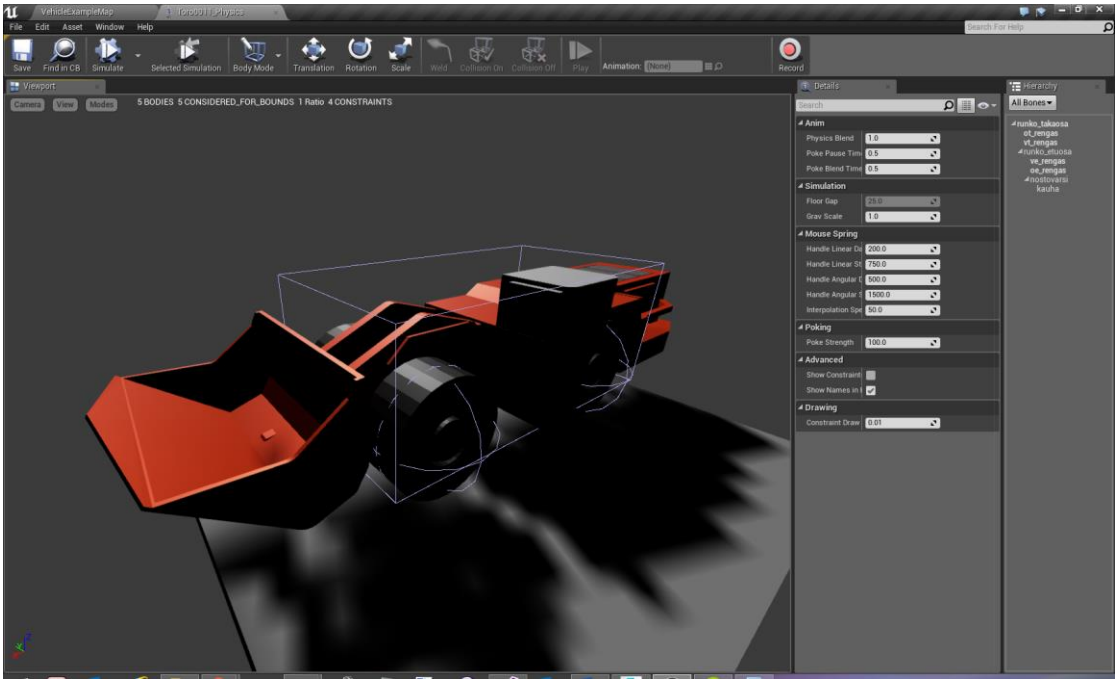
Luuranko (Skeleton Asset)

Unreal Engine 4:ssä luurangot (Skeleton) ovat luurankomalleista (Skeletal Mesh) irrotettuja assetteja. Jokaisella luurankomallilla täytyy olla kytkös luuranko-assettiin. Kaikki animaatio-assetit taas ovat kytköksissä näihin luuranko-asseteihin luurankomallin sijaan. Luuranko-assetti ei ole sama asia kuin luurankomallin luuhierarkia. Unreal Enginessä luuranko-assetti on luu- ja hierarkiatiedon sisältävä lista muun muassa tietystä kaivoskonetyypistä. Tämän listan avulla tehdään kytkökset animointiin. Luu-

ranko-assetin roolia animaatiojärjestelmässä voidaan ajatella luurankomallin ja animaation yhdyskappaleena. Usea luurankomalli voi jakaa saman luuranko-assetin, ja kaikki samaa luuranko-assettia käyttävät luurankomallit voivat jakaa animaatioita. Tästä johtuen saman rakenteen jakavat kaivoskonetyypit voivat hyödyntää samoja animaatioita. (Skeleton Assets n.d.)

5.2.3 Physics Asset Tool (PhAT)

Physics Asset Tool (PhAT) on Unreal Editoriin integroitu editori, jolla voi luoda ja muokata luurankomallien fysiikka-assetteja. Kuviossa 6 on Physics Asset Tool -työkalun yleisnäkymä. Vasemmalla näkyy hallitsevana simulointinäkymä, jossa näkyvät luurankomallin lisäksi luodut fysiikkakappaleet. Kun yläreunassa olevasta työkalurivistä painaa simuloinnin päälle, näkee simulointinäkymästä, kuinka kappaleet käyttäytyvät fysikaalisesti. Simulointinäkymän oikealla puolella on Details-paneeli, johon tulee joko fysiikkasimuloinnin tai valitun luun ominaisuudet editoitavaksi. Oikeassa reunassa näkyy omassa paneelissaan mallin luurankohierarkia, josta kullekin luulle voi lisätä fyysisen kappaleen (body).



Kuvio 6. PhAT-työkalun näkymä

5.3 Tärkeimmät Unrealin C++ -kantaluokat

Kaikki opinnäytetyössä käytetyt luokat periytyvät jostakin Unrealin kantaluokasta. Tässä osiossa käydään hierarkiajärjestyksessä lyhyesti läpi niistä tärkeimmät siltä osin, kuin ne vaikuttavat kaivoskoneisiin, niiden animointiin ja käyttäytymiseen. Unreal Engine 4 -dokumentaation mukaisesti luokkien nimistä on jätetty selostusosassa pois isot alkukirjaimet A ja U. A esiintyy luokissa, joista luodut objektit voidaan ”synnyttää” maailmaan ja ovat näin ollen actoreita. U taas esiintyy kaikissa pelattavuuteen liittyvissä luokissa, joista luodut objektit kuuluvat aina jollekin actorille, mutta jotka eivät voi esiintyä itsenäisesti maailmassa. Yliluokat on merkitty luokkien otsikoissa kaksoispisteellä.

UObject

Object on kantaluokka kaikille Unreal-objekteille. Siitä perittyjä luokkia voidaan merkitä UCLASS-makrolla, jolloin Unrealin objektien hallintasysteemi tulee niistä tietoiseksi. Hallintasysteemi ei sovellu käytettäväksi joka tilanteeseen, mutta se tarjoaa luokille monia hyödyllisiä palveluja mukaan lukien roskienkeräys, viittausten päivitys, reflektointi, serialisointi ja verkon replikointi. (Objects n.d.)

AActor : UObject

Actor on kantaluokka objekteille, joita voidaan sijoittaa tai ”synnyttää” pelimaailmaan. Actorit ovat geneerisiä objekteja, jotka sisältävät 3D-transformoinnit, kuten translaatio, rotaatio ja skaalaus. Niitä voidaan ajatella tavallaan datasäiliöinä, jotka pitävät sisällään ActorComponent-tyyppisiä komponentteja. Näiden komponenttien avulla määritellään, kuinka actor liikkuu tai miltä se näyttää. Yksi Actor-luokan päätoiminto on tiedon replikointi verkon yli verkkosovelluksissa, kuten moninpeleissä. (AActor n.d.)

Actor-luokalla on muutama oleellinen virtuaalinen funktio, joita uudelleenmäärittelemällä voidaan actoreille luoda kustomoitua käyttäytymistä niin C++ - kuin Blueprint-koodista käsin. Simuloinnin käynnistyksen yhteydessä kutsutaan jokaisen actorin BeginPlay()-funktiota, ja sinne voidaan sijoittaa muun muassa objektin alustukseen liittyvää koodia. Jokaisella syklillä taas kutsutaan actorin Tick()-päivitysfunktiota, jonka DeltaSeconds-parametri kertoo edellisestä syklistä kuluneen ajan. Tämän parametrin

avulla voidaan muun muassa skaalata tapahtumien nopeutta, jolloin ne kestävät yhtä kauan riippumatta esimerkiksi ajoalustan suorituskyvystä. (Actors n.d.)

Kaikkia actoreita ei kuitenkaan tarvitse jatkuvasti päivittää, joten Actor-luokan PrimaryActorTick.bCanEverTick-muuttujalla voidaan kontrolloida, mitä objekteja päivitetään tai jätetään päivittämättä. Ohjesääntönä voidaan pitää, että mikäli objektin tilan ei tarvitse koskaan muuttua, ei sitä myöskään tarvitse päivittää. Tästä voi esimerkkinä mainita staattisesti paikallaan olevan kivilohkareen, jolle alustuksen yhteydessä määritellään ulkonäköön vaikuttavia ominaisuuksia, mutta jonka tila ei luomisen jälkeen koskaan muutu. Turhat päivitykset syövät koko järjestelmän suorituskykyä, joten päivittäminen kannattaa tehdä vain sitä tarvitseville objekteille. Tästä johtuen myös actorien päivittäminen on oletuksena pois päältä. (AActor n.d.)

APawn : Actor

Pawn on kantaluokka kaikille actoreille, joita ohjataan joko pelaajan tai tekoälyn toimesta. Kaivossimuloinnin tapauksessa tekoäly on aina vastuussa actorien ohjauksesta, ja ohjauskäskyt tulevat useimmiten kaivoskoneiden toimintaa ohjaavilta managereilta. Pawn-luokka määrittelee kaivoskoneiden ulkoasun lisäksi niiden fyysistä vuorovaikutusta maailmassa. (Pawn n.d.)

AWheeledVehicle : APawn

WheeledVehicle on perusta renkailla liikkuville actoreille. Se käyttää ajoneuvon simulointiin oletuksena nelirenkaisille ajoneuvoille soveltuvaa WheeledVehicleMovementComponent4W-komponenttia, mutta skaalautuu tarvittaessa myös tästä suuremmalle rengasmäärälle. Sekä WheeledVehicle-luokka että sen simulointikomponentti täyttävät säätövaraltaan lähtökohtaisesti kaivoskoneiden tarpeet, joten ne toimivat sellaisenaan kaivoskoneiden pohjana. WheeledVehicle-luokka sisältää UPROPERTY-huomautuksella myös ajoneuvon luurankomallille tarkoitetun Mesh-komponentin, johon editorissa syötetään kaivoskoneiden luurankomallit. (AWheeledVehicle n.d.)

5.4 Blueprint-luokat

5.4.1 Yleistä

Kaivossimuloinnin ydintoiminnot ja -luokat on toteutettu Unreal Engine 4:n tapaan C++ -koodilla. Blueprint-luokilla voidaan laajentaa näitä toimintoja visuaalisesti Unreal Editorissa koodin kirjoittamisen sijaan. Blueprint-luokalle täytyy aina määrittää UObject-luokasta periytyvä yliluokka (parent class). Myös UObject käy Blueprint-luokalle yliluokaksi. Yliluokan määritys mahdollistaa sen ominaisuuksien perimisen Blueprint-luokan käyttöön. Blueprint-luokat voivat laajentaa sekä C++ -luokkia että toisia Blueprint-luokkia. (Blueprint Class n.d.)

Konstruktori

Jokaisella Blueprint-luokalla on ConstructionScript()-funktio, joka suoritetaan kertaalleen, kun objekti luodaan. Tässä funktiossa voidaan suorittaa muun muassa objektin alustukseen liittyviä toimenpiteitä. ConstructionScript koostuu solmu-kaaviosta, joka ajetaan jokaiselle Blueprint-luokan instanssille uudestaan aina luokan kääntämisen yhteydessä. Se ajetaan myös aina, kun editorissa olevaa objektia siirretään tai sen arvoja muutetaan (Blueprint Fundamentals). Tätä ominaisuutta hyödynnetään kaivostunnelien dynaamisessa luonnissa TunnelBP-luokassa (ks. 6.2 Reittiluokat).

Muuttujat

Kuten C++ -luokkiin, myös Blueprint-luokkiin voi lisätä muuttujia ja funktioita. Muuttujat voivat sisältää arvon, tai referenssin maailmassa olevaan objektiin tai actoriin. Blueprint-luokka pääsee suoraan käsiksi kaikkiin sisäisiin muuttujiinsa, ja ne voidaan asettaa näkyväksi myös ulkopuolelta. Tällöin niiden arvoja voi muuttaa myös maailmaan asetettujen instanssien kautta. Blueprint-luokan muuttujat voivat olla referenssejä tai perustietotyyppiä kuten boolean, integer tai float. Jokaisesta muuttujatyypistä voi luoda myös taulukon (Array). (Blueprints User Guide: Variables n.d.)

Funktiot

Blueprint-funktiot toteutetaan omina solmu-kaavioinaan, joihin muodostetaan toimintalogiikkaa BVS-solmuilla. Funktioita voidaan suorittaa tai kutsua muista Blueprintin kaavioista käsin, missä ne näkyvät yhtenä solmuna muiden joukossa. Funktioilla

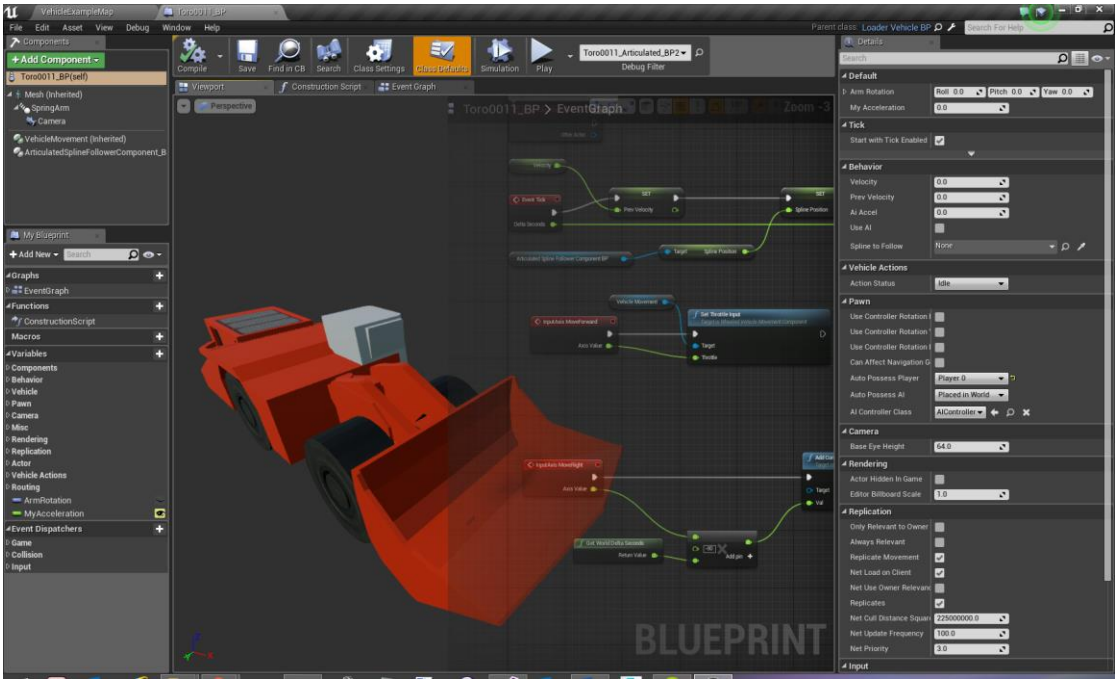
on aluksi yksittäinen, funktion nimeä kantava sisääntulo-solmu, josta lähtee ulos yksittäinen suoritusnasta (exec, ks. 5.3.1 Blueprint Visual Scripting). Kun funktiota kutsutaan toisesta kaaviosta, tämä nasta aktivoituu, jolloin koko siihen yhdistetty solmuverkko suoritetaan. Blueprint-funktioille voi antaa C++ -kielen tavoin näkyvyyden, jonka avulla määritellään, mitkä objektit voivat kutsua sitä. Funktioille voidaan myös asettaa paikallisia muuttujia sekä yhden tai useamman parametrin ja paluuarvon. (Functions (Blueprints) n.d.)

Komponentit

Komponentit ovat oleellinen osa actorien ja siten myös kaivoskoneiden rakennetta. Ne käytännössä määrittävät, miltä actor näyttää ja miten se toimii. Actorin komponentteja voidaan lisätä, poistaa ja vaihtaa lennossa. Tämä mahdollistaa dynaamisen käyttäytymisen ja toiminnan actoreille. Kaivoskoneille lisättäväksi tarkoitettu SplineFollowerComponent-komponentti esimerkiksi ohjaa kaivoskoneiden liikkumista splini-reittiä pitkin, ja SkeletalMesh-komponentin avulla kaivoskoneille voidaan lisätä luurankomallit ja Animation Blueprintit. (Components n.d.)

Komponentit periytyvät Unrealin ActorComponent-luokasta ja Actoreiden tapaan myös niitä voidaan päivittää jokaisella syklillä niiden TickComponent()-funktion avulla. Näin ollen niillä voidaan suorittaa erilaista jatkuvaa toimintaa. SplineFollowerComponent esimerkiksi käyttää TickComponent()-funktiota päivittääkseen isäntänsä kulkua splini-reittiä pitkin, ja SkeletalMesh-komponentti puolestaan kutsuu sitä animaation ja luukontrollerien päivittämiseen. (Mt.)

Kuviossa 7 näkyy yleinen Actor-tyyppisen Blueprint-luokan editori-näkymä. Kuvaa on muokattu sen verran, että samassa kuvassa näkyy puolittain kahden eri välilehden sisältö. Keskellä vasemmassa puolikkaassa näkyy Viewport-välilehden näkymä, josta nähdään, miltä luokan instanssi parhaillaan näyttää. Oikeassa puolikkaassa näkyy Event Graph-välilehden näkymä, jossa on BVS-solmuista rakentuva toimintalogiikka. Kuvion vasemmassa reunassa ylhäällä on luokan komponenttilista ja sen alapuolella muuttujien ja funktioiden hallintaikkuna. Oikeassa reunassa on ikkuna, jossa voidaan määrittää valittuna olevan kohteen ominaisuuksia.



Kuvio 7. Actor-tyyppisen Blueprint-luokan muokattu editori-näkymä

5.4.2 Blueprint Visual Scripting

Unreal Engine 4 sisältää visuaalisen skriptikielen, jolla Blueprint-luokkiin voidaan lisätä toimintalogiikkaa. Tähän Blueprint Visual Scripting-systeemiin viitataan tässä työssä useimmiten lyhenteellä BVS. Se on rakenteeltaan ja ulkoasultaan hyvin lähellä aiemmassa Unreal Engine-versiossa UDK:ssa esiintyvää Kismet-työkalua. Kismet on tarkoitettu lähinnä pelikentän tapahtumien hallintaan, ja Unreal Enginessä 4:ssä sitä vastaa pelikenttään sidottu Level Blueprint. BSV vie visualisen skriptauksen Unreal Engine 4:ssä kuitenkin pidemmälle ja se on vastuussa kaikkien Blueprint-luokkien sisäisen logiikan hallinnasta. BSV onkin syrjäyttänyt C++ -kielen kanssa kokonaan vielä UDK:ssa mukana olevan UnrealScript-skriptikielen. (Blueprint Overview n.d.)

BVS rakentuu niin sanotusta solmuverkosta (node network), jossa jokainen solmu vastaa ajettavan toiminnan osaa, kuten funktiokutsua tai sijoitusoperaatiota. Solmuun voi tulla tai siitä voi lähteä yksi tai useampi nasta (pin). Nastoihin yhdistetään muun muassa funktioiden argumentteja. Erityinen nasta-tyyppi on suoritus-nasta (exec), jonka avulla määritellään solmuverkon suoritusjärjestys. Tapahtuman tai funktion suoritus lähtee aina liikkeelle niin kutsutusta lähtösolmusta, jolla on ainoas-

taan yksi lähtevä suoritusnasta. Tästä nastasta lähdetään liikkeelle yhdistäen se seuraavana ajettavana olevan solmun tulevaan suoritusnastaan. Tätä jatketaan, kunnes solmuverkko ratkaisee määrätyn ongelman.

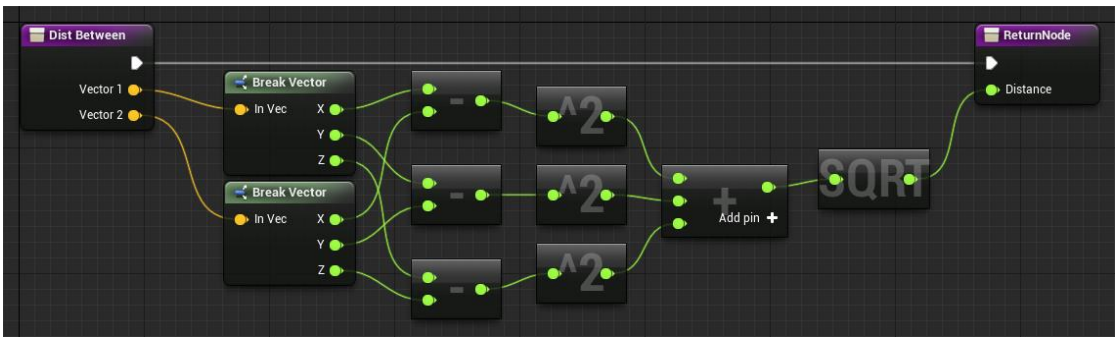
Alla on esimerkki yksinkertaisesta C++ -funktiosta, joka laskee kahden kolmiulotteisen pisteen välisen etäisyyden:

```
float DistBetween(FVector Vector1, FVector Vector2)
{
    float dx = FMath::Square(Vector2.X - Vector1.X);
    float dy = FMath::Square(Vector2.Y - Vector1.Y);
    float dz = FMath::Square(Vector2.Z - Vector1.Z);

    float dist = FMath::Sqrt(dx + dy + dz);

    return dist;
}
```

Kuviossa 8 on esimerkki vastaavanlaisen funktion toteutuksesta BVS:llä. Kuviosta nähdään, että muuttujien käsittely ei vaadi BSV:ssä suoritusnastojen (exec) käyttöä. Tästä johtuen esimerkin sisääntulosolmun lähtevä suoritusnasta on yhdistetty suoraan paluuarvon palauttavan solmun tulevaan suoritusnastaan.



Kuvio 8. Esimerkki Blueprint-funktiosta

BVS soveltuu monipuolisuutensa ansiosta hyvinkin monimutkaisten ongelmien ratkaisemiseen. Se sisältää muun muassa useista ohjelmointikielistä tuttuja ominaisuuksia, kuten eri tyyppien vertailuoperaatiot, toistorakenteet ja ajonaikainen tyyppitarkistus. Se sisältää myös monia logiikan rakentamisen kannalta hyödyllisiä ominaisuuksia, kuten DoOnce, DoN ja Gate. Blueprint-editorista voi valita virheenkorjausta (debugging) varten luokasta muodostetun instanssin ja sen muuttujien arvoja voi seurata Event Graph-näkymässä ajonaikaisesti. BVS-solmuille voi asettaa myös pysäytyspisteitä (breakpoint), joihin koodin suoritus pysähtyy. Tällöin voidaan tarkastella rauhassa Blueprint-luokan tilaa pysäytyshetkellä.

Aivan kuten useimmissa muissakin ohjelmointikielissä, täytyy Blueprint-luokat kääntää (compile) ennen kuin niitä voidaan käyttää ohjelmassa. Kääntämisprosessi luo Blueprint-luokasta muistiin uuden UClass-luokan, jossa on UFunction()-funktio, johon jokainen Blueprintin solmuverkko muunnetaan tavukoodiksi. Jokaista Blueprintissä esiintyvää muuttujaa kohden UClass-luokkaan luodaan UProperty-muuttuja. Kääntäjään luoman tavukoodin ajaminen tapahtuu alustakohtaisessa virtuaalikoneessa, mikä mahdollistaa saman Blueprint-koodin ajamisen alustasta riippumatta (Blueprint FAQ and Tips n.d). (Blueprint Compiler Overview n.d.)

Visuaalisen ohjelmoinnin käytöstä on C++ -koodiin nähden sekä useita etuja että haittoja. Etuina nopean kehitystyön lisäksi, sitä on helppo oppia ja lukea, eikä se vaadi pohjalla olevaa ohjelmointikielen tai -syntaksin tuntemista. Lisäksi kirjoitus- ja ohjelmointivirheet vähenevät, eivätkä poikkeustilanteet kaada koko järjestelmää. Visuaalisen solmurakenteen takana on myös paljon piilotettua toimintaa, josta käyttäjän ei tarvitse huolehtia.

Haittapuolena visuaalisessa ohjelmoinnissa on sen rajoittuneisuus C++ -ohjelmointiin verrattuna. BVS ei esimerkiksi sovellu XML-parserin tai tietokantalukijan ohjelmointiin, vaan se toimii pääasiassa valmiiksi määrätyillä komponenteilla. Komponentin sisällä olevan virheen korjaaminen on useimmiten hankalaa, jos se on edes mahdollista. Lisäksi visuaalinen koodi on hitaampaa suorittaa kuin C++ -koodi. Lähteiden mukaan BVS-koodin suorittaminen on keskimääräisesti noin 10 - 15 kertaa C++ -koodia hitaampaa, mistä johtuen se soveltuu huonosti suorituskyvyn kannalta kriittisten ohjelmaosien toteutukseen. Kaivoskoneiden liikkeiden syklin aikana vaatima prosessointi on kuitenkin sen verran vähäistä, että BVS soveltuu niiden toteutukseen mainiosti.

5.5 Animation Blueprint

Animation Blueprint on luurankomallien animointiin erikoistunut Blueprint-tyyppi. Sen kaavioiden avulla voidaan sulauttaa yhteen (blend) animaatioita, kontrolloida suoraan luurangon luita ja muodostaa luurankomallille uusi asento joka päivityssyklillä. Jokaisella Unrealin Pawn-luokasta perityllä objektilla on SkeletalMeshCom-

ponent-komponentti, joka viittaa animoitavaan luurankomalliin sekä instanssi luurankomallia ohjaavasta Animation Blueprintistä. Animation Blueprint pääsee tapahtuma- ja animaatiokaavioiden kautta käsiksi Pawn-objektin ominaisuuksiin, laskien niiden avulla luurankomallin asennon sulauttamalla yhteen animaatioketjujen ja luukontrollerien aikaan saamia muodonmuutoksia. (Animation Blueprints n.d.)

Event Graph

Jokaisella Animation Blueprintillä on yksi tapahtumakaavio. Se sisältää erityisesti animointia varten tarkoitettuja tapahtumia, jotka laukaisevat niihin liittyvien solmuverkkojen suorittamisen. Tärkeimmät animaatioon vaikuttavat tapahtumat ovat Animation Blueprint-instanssin luonnin yhteydessä kutsuttu alustustapahtuma BlueprintInitializeAnimation() sekä jokaisella päivityssyklillä kutsuttava BlueprintUpdateAnimation(). Tapahtumakaaviota käytetään useimmiten animaatioon dynaamisesti vaikuttavien tekijöiden laskemiseen ja päivittämiseen, joita voidaan sitten hyödyntää animaatiokaaviossa asentojen muodostuksessa. (EventGraph n.d.)

Anim Graph

Animation Blueprintillä on animaatiokaavio, johon voidaan lisätä animaatio-solmuja (node) animaatioketjujen samplaamista, animaation sulautusta (blend) ja luiden transformaatiota luukontrollien (Skeletal Controls) avulla ohjaamista varten. Animaatiokaaviota käytetään luurankomallin hetkellisen asennon laskemiseen, ja se asetetaan lopuksi mallille joka syklillä. Animaatiokaaviossa voidaan käyttää tapahtumakaaviossa tai koodissa laskettuja arvoja. (AnimGraph n.d.) Kaivoskoneiden tapauksessa käytetään tyyppikohtaisten AnimationBlueprint-luokkien C++ -kantaluokassa esiteltyjä luukontrollereita luiden rotaation määritykseen. Niiden arvoja lasketaan koneiden tyyppikohtaisissa Blueprint-luokissa.

Skeletal Controls

Animation Blueprintin SkeltalControls-solmut mahdollistavat luurangon luiden suoran hallinnan animaatiokaaviossa. Tämä mahdollistaa nivellettyjen kaivoskoneiden nivelten dynaamisen asemoinnin tarpeen mukaan. Kaivossimuloinnista käsin voidaan esimerkiksi ohjata kauhakuormaajan kauha ala-asentoon kuorman lastaamista varten

samalla, kun kone taipuu keskinivelensä kohdalta reitinseurantakomponentin mukaisesti ajaessaan kohti kuormauspistettä.

5.6 Blueprint- ja C++ -luokkien yhteiskäyttö

5.6.1 Yleistä

Yksi Unreal Engine 4:n suurimmista vahvuuksista sovelluskehittäjän näkökulmasta on se, että C++ -luokkia voidaan laajentaa Blueprinteillä. Tämä mahdollistaa molempien luokkatyyppien parhaiden ominaisuuksien yhdistämisen kehitystyössä. C++ -luokka lisätään projektiin helpoiten Unreal Editorin kautta, jolloin siihen tehdään automaattisesti Blueprint-luokkien kanssa reflektointiin tarvittavat lisäykset. Samalla C++ -luokalle voidaan määrittää Unreal Enginen objekteille tarkoitetusta UObject-kantaluokasta periytyvä yliluokka. C++ -luokkiin lisätään Unreal Enginen reflektointia ohjaavia makroja, joilla voidaan määrittää kuinka C++ -luokka näyttäytyy Blueprint-luokille. Makroihin voi edelleen lisätä tarkenteita ja metadataa, joilla voi määrittää yksityiskohtaisesti C++- ja Blueprint-luokkien välistä suhdetta.

Jotta Blueprint-luokka voi saada tietoa C++ -luokasta, täytyy ennen C++ -luokan esittelyä kutsua UCLASS()-makroa. Tämä kertoo kääntämisen yhteydessä Unreal Header Tool-työkalulle (UHT), että kyseinen luokka reflektoidaan. Luokan rungon alkuun lisätään tämän lisäksi GENERATED_BODY()-makro, joka lisää UHT:n automaattisesti generoiman luokan runkoon reflektoinnissa tarvittavia funktioita ja määrittelee muuttujatyyppien peitenimet (typedef). (Noland 2014.) UCLASS()-makrolle voidaan lisätä tarkenteita (specifier) ja metadataa, joiden avulla voidaan määritellä kuinka C++ -luokka toimii Blueprint-luokkien kanssa. Nämä määritteet periytyvät luokkaperinnän mukana. Esimerkiksi Blueprintable-tarkenne mahdollistaa C++ -luokan käytön Blueprint-luokan kantaluokkana. (C++ and Blueprints n.d.)

5.6.2 Muuttujien reflektointi

C++ -luokan muuttujat määritellään Unreal Enginelle reflektoitavaksi lisäämällä UPROPERTY()-makro ennen niiden esittelyä. Muuttujat toimivat edelleen C++ -koodissa kuten ennenkin, mutta samalla ne voivat tulla näkyviin ja muokattavaksi Unreal Edi-

toriin UPROPERTY()-makrolle annettujen tarkenteiden ja metadatan mukaisesti. (Unreal Architecture). Tarkenteilla ja metadatalla voidaan kontrolloida hyvin yksityiskohtaisesti, kuinka muuttujat näkyvät ja toimivat Unreal Editorin puolella. Tarkenteilla voidaan määrittää muun muassa muuttujien näkyvyyttä, kirjoitus- ja lukuoikeuksia ja tallennusta. Metadatalla taas voidaan antaa muuttujille ominaisuuksia, kuten työkaluvinkki (tooltip), ryhmä (category), raja-arvot ja editorissa näkyvä nimi (DisplayName).

Alla on esimerkki reflektoitavan float-tyyppisen C++ -muuttujan esittelystä.

```
UPROPERTY(BlueprintReadWrite, VisibleAnywhere, EditAnywhere, Category =
"Example", meta = (Tooltip = "This is sample float", ClampMin = "-1.0", UIMin =
"-1.0", ClampMax = "1.0", UIMax = "1.0"))
float SampleFloat;
```

SampleFloat-muuttujaan voi esimerkin reflektointimääritteiden mukaisesti kirjoittaa sekä siitä voi lukea arvon Blueprint-luokasta käsin (BlueprintReadWrite). Se näkyy Blueprint-näkymän ominaisuuksissa (VisibleAnywhere) ja sitä voi muokata siellä (EditAnywhere) ja se menee ”Example”-ryhmän alle (Category). SampleFloat antaa tooltip-tekstin ”This is sample float”, kun hiiren kursori viedään Blueprint-näkymässä sen päälle. Käyttöliittymässä muuttujan kohdalla näkyy liukusäädin, jolla voi säätää float-arvoa väliltä -1.0 - 1.0 (UIMin, UIMax). Editori ei anna alittaa tai ylittää näitä arvoja edes käsin asettamalla (CampMin, ClampMax).

5.6.3 Funktioiden reflektointi

Muuttujien tapaan myös funktioilla on oma UFUNCTION()-makro, joka lisätään Unreal Enginen reflektointia varten ennen C++ -funktioiden määrittelyä. Reflektoidut funktiot toimivat C++ -koodissa aivan kuten mitkä tahansa C++ -funktiot. Niillä voi edelleen olla C++ -toteutus, niitä voi kutsua muualta C++ -koodista käsin ja niiden toteutuksessa voi olla kutsuja muihin C++ -funktioihin riippumatta siitä, ovatko ne reflektoituja vai eivät. Reflektoituja funktioita voi kuitenkin kutsua tai uudelleenmääritellä BVS-systeemistä käsin. Myös UFUNCTION()-makrolle voi lisätä tarkenteita ja metadataa, joilla ohjataan funktioiden käyttömahdollisuuksia Blueprint-luokissa. (Functions (C++) n.d.) Blueprint-funktioiden tapaan myös reflektoiduille C++ -funktoille voi-

daan määrittää paluuarvoja laittamalla argumenttlistaan referenssit haluttuihin tietotyyppeihin. Nämä referenssit näkyvät funktion BVS-solmussa omina lähtönastoinaan ja ne voidaan yhdistää muuhun BVS-logiikkaan sekä C++ -koodiin.

Alla on esimerkki reflektoitavan C++ -funktion esittelystä.

```
UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category = "Example", meta =
(FriendlyName = "DoSample", HidePin = "CppOnlyInput", DefaultToSelf = "TheVehi-
cle"))
void SampleFunction(AMiningVehicleBase *TheVehicle, float CppOnlyInput = 2.0f);
virtual void SampleFunction_Implementation();
```

SampleFunction()-funktion voi reflektiomääritteiden mukaisesti uudelleen määritellä Blueprint-luokassa samalla, kun C++ -luokassa on funktion natiivi toteutusosa SamleFunction_Implementation() (BlueprintNativeEvent). Lisäksi funktiota voi kutsua Blueprint-koodista (BlueprintCallable). SampleFunction() menee ”Example”-ryhmän alle (Category) ja näkyy Blueprint-koodissa nimellä ”DoSample” (FriendlyName). Blueprint-koodissa funktiosta on näkyvissä ainoastaan TheVehicle-argumentti, joka on oletuksena luokan oma instanssi. CppOnlyInput-argumentti on ainoastaan C++ -koodista käytettävissä, sillä se on piilotettu Blueprint-koodista (HidePin).

C++ -koodin integrointi on tehty Unreal Editorissa todella helpoksi. Sen voi kääntää suoraan Unreal Editorista käsin siten, että muutokset päivittyvät suoraan Blueprint-luokkiin ilman tarvetta sulkea editoria välissä. Blueprint- ja C++ -luokkien yhteiskäytössä on kuitenkin muutamia tekijöitä, jotka on hyvä ottaa heti alusta lähtien huomioon. Blueprint-luokat voivat periytyä vain yhdestä C++ - tai Blueprint-luokasta, joten ne eivät tue C++ -kielen tavoin moniperintää. Tämänkin jälkeen C++ -luokista voidaan kuitenkin tuoda uutta sisältöä Blueprint-luokkiin muun muassa koostamalla niistä komponentteja. Tämä Blueprint-luokkien rajoitus ei luonnollisesti koske millään tavalla C++ -luokkien normaalia moniperintää; se on edelleen sallittua. Toinen, kenties itsestään selvä rajoite on, että C++ -luokat eivät voi periytyä Blueprint-luokista.

6 ARKKITEHTUURI

6.1 Lähtökohdat

Unreal Engine 4:n rakenne ja tulevaisuuden käyttötarkoitukset antoivat selkeät raamit nivellettyjen kaivoskoneiden arkkitehtuurille. Vaikka reitinmuodostus ja -seuranta sekä koneiden animointi oli tarkoitus toteuttaa pääasiassa Blueprint-luokkiin, oli ehdottoman tärkeää, että toimintaa voidaan ohjata myös C++ -koodista käsin. Tämä siitä syystä, että simulointiohjelmistot toimivat pääasiassa C++ -”puolella”, eivätkä ne pääse helposti käsiksi siihen osaan koneiden dataa, joka on määritelty puhtaasti Blueprint-luokissa. Reflektoinnin toteuttamalla tämäkin olisi ollut mahdollista, mutta se olisi ollut tarpeettoman monimutkainen lähestymistapa siihen nähden, että Unreal Enginessä oli jo valmiiksi toteutettu reflektointi C++ -luokista Blueprint-luokkiin.

Tästä johtuen C++ -kantaluokkiin lisättiin Unrealin reflektio-määritteillä kaikki ne muuttujat ja funktiot, joille haluttiin sallia pääsy niin C++ -kuin Blueprint-luokistakin käsin. Lisäksi jokainen C++ -luokka periytettiin käyttötarkoitukseen sopivasta Unreal Enginen luokasta, jolloin saatiin ratkaisevan paljon valmiiksi toteutettuja palveluja luokan käyttöön.

Arkkitehtuurin suunnittelussa piti ottaa huomioon muutamia Blueprint- ja C++ -luokkien yhteistoiminnan rajoitteita. Ensinnäkin C++ -luokka ei voi periytyä Unrealin Blueprint-luokasta. Toinen merkittävä rajoite oli Blueprint-luokista puuttuva tuki moniperinnälle.

6.2 Ajoneuvojen luokat

6.2.1 Ajoneuvojen C++ -luokat

Koska kaikki kaivoksissa toimivat ajoneuvot eivät ole nivellettyjä, nähtiin tarve yleiselle ajoneuvon kantaluokalle, joka ei ota ajoneuvon nivellykseen tai muuhun rakenteeseen lainkaan kantaa. Tälle kantaluokalle annettiin nimeksi AMiningVehicleBase ja sen kantaluokaksi tuli Unreal Enginen AWheeledVehicle. Näin jokainen kaivoksessa

operoiva ajoneuvo sai käyttöönsä AWheeledVehile-luokan tarjoamat ajoneuvon simulointipalvelut.

Kaikki simuloinnissa käytetyt ajoneuvot tulisivat oletusarvoisesti tyypistä riippumatta liikkumaan splini-käyrää seuraten. Koska reitin muodostus ja seuranta oli tarkoitus toteuttaa Blueprint-luokkiin, lisättiin MiningVehicleBase-luokkaan Unrealin reflektiomäritteillä reititykseen tarvittavat muuttujat ja funktiot. Alla on näiden elementtien header-tiedostossa oleva esittelyosa.

```
/** A spline component for a vehicle to follow */
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Routing)
ARouteSpline *SplineToFollow;

/** Vehicle's current distance along a spline from the spline's start point */
UPROPERTY(BlueprintReadWrite, Category = Routing)
float SplinePosition;

/** The function to call from the spline follower component when the vehicle
has reached the end of the spline. BlueprintNativeEvent makes this function
available for being overridden by a Blueprint */
UFUNCTION(BlueprintNativeEvent, BlueprintCallable, Category = Routing)
void OnRouteEnd();
/** A native implementation part of the FollowRoute function lies here */
virtual void OnRouteEnd_Implementation();
```

SplineToFollow -reittikomponentti on Actor-luokasta periytyvä datasäiliö, joka sisältää myös varsinaisen SplineComponentin jota reitityksessä luetaan. SplinePosition on liukulukumuuttuja, joka ilmaisee matkana kuinka kauaksi kone on edennyt spliniä pitkin sen alkupisteestä lähtien. Tätä muuttujaa päivitetään reitinseuranta-komponentista käsin. OnRouteEnd() on natiivi Blueprint-tapahtuma (event) jota reitinseuranta-komponentti kutsuu, kun sen omistaja-ajoneuvo on saapunut splinin päätepisteeseen. Näin ajoneuvon toimintaa voidaan ohjata eteenpäin niin C++ - kuin Blueprint-luokastakin käsin.

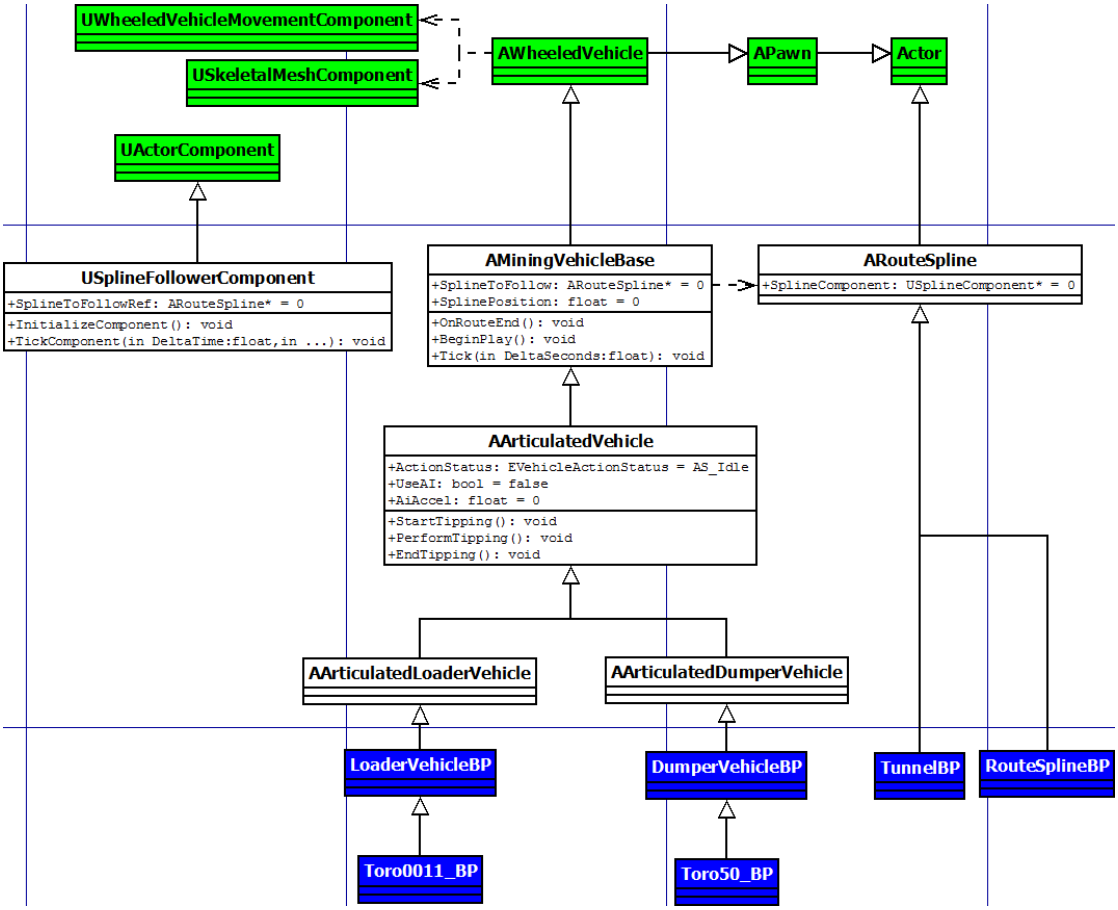
Seuraavalle tasolle luokkahierarkiassa lisättiin MiningVehicleBase-luokasta peritty nivellettyjen kaivoskoneiden kantaluokka AArticulatedVehicleBase, joka laajentaa MiningVehicleBase-luokkaa toimintamääritteillä (EVehicleActionStatus-enumeraatio) ja niiden aktivointitapahtumilla (events). Tapahtumat on eritelty noston aloitusta (StartTippint), kippaamista (PerformTipping) ja lähtöasentoon palaamista (Endtipping) varten. Otsikkotiedoston tapahtumien esittelyyn lisättiin Unrealin UFUNCTION-huomautukset, joten Unreal Enginen reflektointisysteemi poimi ne käytettäväksi myös Blueprint-luokista käsin.

Viimeiselle C++ -perintätasolle luotiin konetyyppejä varten varatut, toistaiseksi tyhjät AArticulatedLoaderVehicle- ja AArticulatedDumperVehicle-luokat. Nämä luokat toimivat kantaluokkina konetyyppikohtaisille Blueprint-luokille, joihin toteutettiin BVS:llä konetyyppikohtaista toimintaa. Niihin lisätään mitä todennäköisimmin tulevaisuudessa kaivossimuloinneissa tarvittavia kaivoskoneiden tyyppikohtaisia ominaisuuksia.

6.2.2 Ajoneuvojen Blueprint-luokat

Ajoneuvojen luokkahierarkiassa ensimmäiset Blueprint-luokat olivat kaivoskoneiden tyyppikohtaiset DumperVehicleBP- ja LoaderVehicleBP, jotka periytyivät vastaavista AArticulatedDumperVehicle- ja AArticulatedLoaderVehicle- C++ -luokista. Näihin Blueprint-luokkiin määriteltiin kaivoskoneiden tyyppikohtaisia ominaisuuksia, kuten rengas-asetukset sekä toteutettiin animaatioiden hallinta BVS:llä. DumperVehicleBP toimi kantaluokkana työn kippiauton Toro50_BP-luokalle, ja LoaderVehicleBP vastaavasti kauhakuormaajan Toro0011_BP-luokalle.

Kuviossa 9 on esitetty kaivoskoneiden kannalta tärkeimpien luokkien rakenteet ja niiden väliset suhteet. Unreal Enginen C++ -luokat on esitetty kuviossa vihreällä, omat C++ -luokat valkoisella ja Blueprint-luokat sinisellä värillä.



Kuvio 9. Kaivoskoneiden luokkahierarkian tärkeimmät komponentit

6.3 Reittiluokat

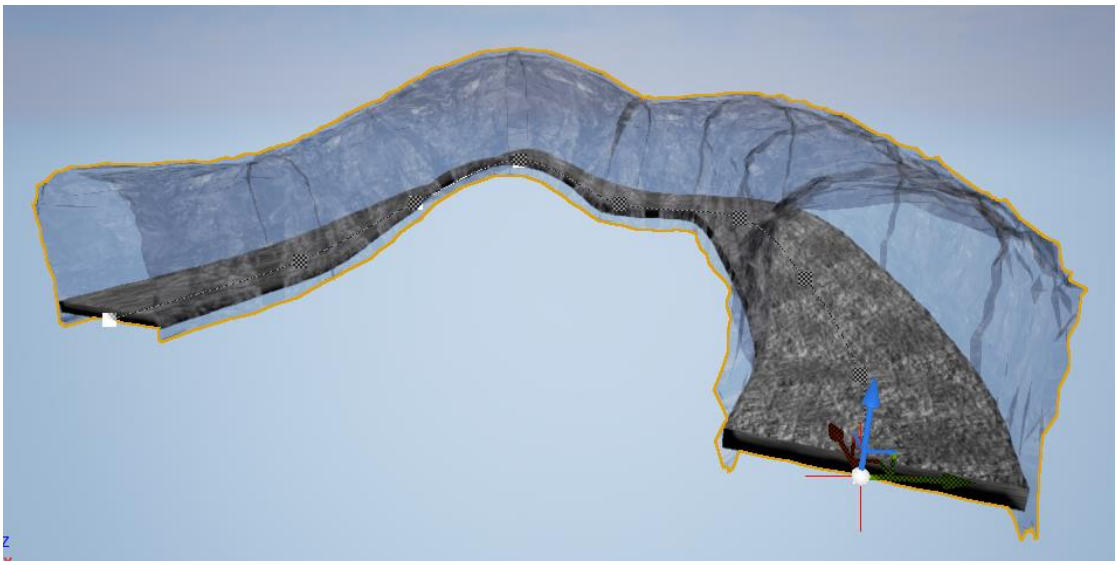
Reittiluokat ovat niitä ohjelman osia, joilla määritellään ajoneuvojen kulkemat reitit. Niitä varten luotiin RouteSpline-niminen C++ -kantaluokka, joka periytyy Unrealin Actor-luokasta. Se on toistaiseksi hyvin yksinkertainen luokka, joka sisältää osoittimen SplineComponent-tyyppiseen komponenttiin. Tästä osoittimesta reitinseurantakomponentti lukee reittidataa (spliniä). RouteSpline-luokan sisältöä voidaan laajentaa tulevaisuudessa muun muassa erilaisilla säännöillä, kuten ajosuunta tai nopeusrajoitus.

RouteSpline-luokasta perittiin kaksi Blueprint-luokkaa, RouteSplineBP ja TunnelBP. Niiden komponenttilistoihin lisättiin SplineComponent-tyyppinen splini-komponentti, joka määrittää reitin muodon. Tämä komponentti yhdistettiin reitinseurantaa varten Blueprint-luokkien Construction Scripteissä kantaluokan SplineComponent-osoittimeen. SplineComponent ei voi esiintyä maailmassa ilman Actor-tyyppistä omistajaa,

mistä johtuu myös RouteSpline-luokan Actor-perintä. Kun omistaja viedään maailmaan, menee splini-komponentti sen mukana. Tämä mahdollistaa lopulta myös sen muokkaamisen. Reittiluokkien hierarkia esiintyy kuviossa 9.

RouteSplineBP-luokka edustaa splini-reittiä yksinkertaisimmillaan. Se käsittää vain reitin muodon, ja sitä voi käyttää reitin luomiseen muun muassa avoimissa paikoissa. TunnelBP sen sijaan vie reitinmuodostuksen hiukan pidemmälle. Reitin muodon lisäksi se luo ja päivittää dynaamisesti tunnelia muokatun reitin mukaisesti. Tunnelin luonti on toteutettu TunnelBP-luokan Construction Scriptissä. Se luo jokaisen reittisplinin pisteen väliin kaksi Unreal Enginen SplineMesh-tyyppistä komponenttia, joista toinen edustaa tunnelin lattiaa ja toinen holvia. Komponentit saavat automaattisesti muotonsa, kun niille syöttää päätepisteiden sijainnin ja tangentin. Tämän lisäksi niille täytyy asettaa staattiset lattia- ja holvimallit. Lopputuloksena on dynaaminen tunneli.

Kuviossa 10 näkyy esimerkki TunnelBP-luokan dynaamisesta tunnelista, joka on muodostunut automattisesti reitinvedon ohessa. Tunnelin holvi on tarkoitettu läpinäkyväksi, jotta sen sisällä operoivat kaivoskoneet näkyvät myös ulkopuolelta katsottuna. Jos kuviota katsoo tarkasti, voi nähdä tunnelin lattian myötä kulkevan splini-käyrän ja sen kontrollipisteet.

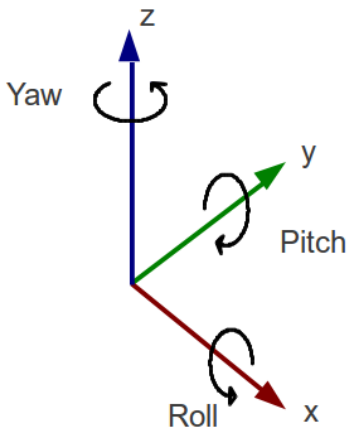


Kuvio 10. TunnelBP-luokan dynaamisesti muodostama tunneli

6.4 Animointiluokat

Nivellettyjen kaivoskoneiden animointiin tarvittiin dynaaminen järjestelmä, jotta koneiden nivelien asentoja voitiin kontrolloida hallitusti niin C++ -kuin Blueprint-luokistakin käsin. Unrealissa dynaaminen animointi voidaan toteuttaa Animation Blueprinteillä, jotka periytyvät lähtökohtaisesti UAnimInstance-luokasta. Koska nivelletyillä kaivoskoneilla oli yhteinen taivutukseen tarkoitettu nivel sekä konetyypeillä yhteiset toiminta-animointiin tarkoitetut nivelet, luotiin nivelletyille koneille yhteinen kantaluokka UArticulatedAnimInstance sekä siitä perimällä omat luokat kauhakuormaaja- ja kippiauto-tyyppisiä toiminta-animointeja varten. Näihin luokkiin lisättiin UPROPERTY-huomautuksilla luukontrollerit jokaisen nivelen asemoimista varten.

Kaikki opinnäytetyön luukontrollerit ovat Unrealin FRotator-tyyppisiä rotaatio-informaation sisältäviä struktuureita, jotka määrittävät liikkeen asteina kunkin kolmiulotteisen koordinaatiston akselin (x, y ja z) ympäri. Liikkeet tunnetaan termeillä kääntyminen (engl. yaw), nyökkääminen (engl. pitch) ja kallistuminen (engl. roll). Kuviossa 11 on havainnollistettu nämä liikkeet. Unrealissa positiivinen x-akseli osoittaa eteenpäin ja z-akseli ylöspäin kuvion mukaisesti. Näin ollen nivellettyjen kaivoskoneiden nivelien liikkeet rajoittuvat kääntymiseen (keskinivel) ja nyökkäämiseen (nostovarsi, kauha ja kippilava).



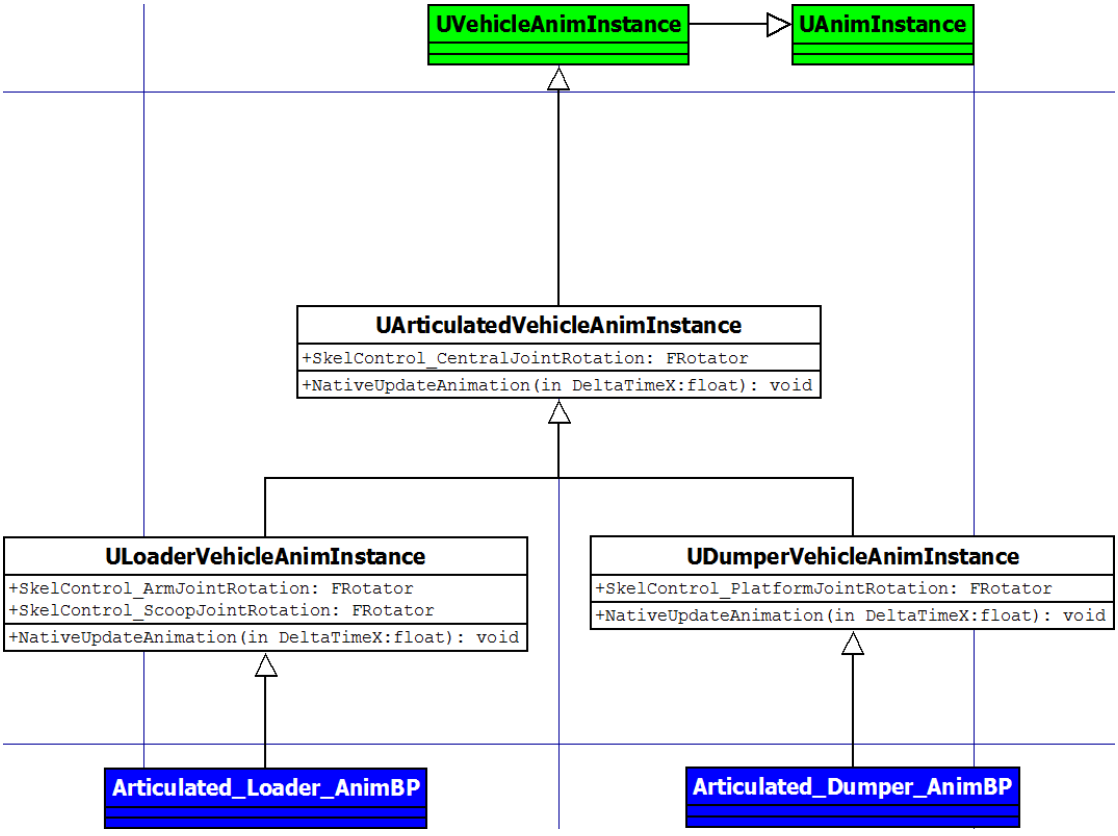
Kuvio 11. Kolmiulotteisen koordinaatiston akselien ympäri tapahtuva liike

UArticulatedAnimInstance on oma kantaluokka kaikille nivellettyjen kaivoskoneiden Animation Blueprinteille. Se periytyy Unrealin UVehicleAnimInstance-luokasta, joka tarjoaa ajoneuvon renkaiden animoinnin AWheeledVehicle-tyyppisille objekteille.

UArticulatedAnimInstance-luokkaan lisättiin nivellettyjen kaivoskoneiden keskinivelen taivutukseen tarkoitettu SkelControl_CentralJoinRotator-luukontrolleri. Luokassa tehtiin myös uudelleenmäärittely UAnimationInstance-kantaluokan NativeUpdateAnimation()-metodista, jolloin animaation kontrollointi voidaan siirtää tarvittaessa kokonaan C++ -koodiin.

UArticulatedAnimInstancesta perittiin vielä tyyppikohtaiset animointiluokat. Kauhakuormaajille tarkoitettu ULoaderVehicleAnimInstance lisäsi kantaluokkaansa luukontrollerit nostovarren sekä kauhan kääntöjä varten. Kippiautojen animointi sai oman DumperVehicleAnimInstance-luokkansa, jossa esiteltiin luukontrolleri kippilavan kääntymistä varten.

Kuviossa 12 on esitetty kaivoskoneiden animoinnin kannalta keskeisimpien luokkien rakenteet ja niiden suhteet. UVehicleAnimInstance on Unreal Enginen C++ -kantaluokka, joka esiintyy kuvassa vihreänä. Siitä perityt omat C++ -luokat näkyvät valkoisella, ja Blueprint-luokat sinisellä värillä.



Kuvio 12. Kaivoskoneiden animointi-instanssien tärkeimmät komponentit

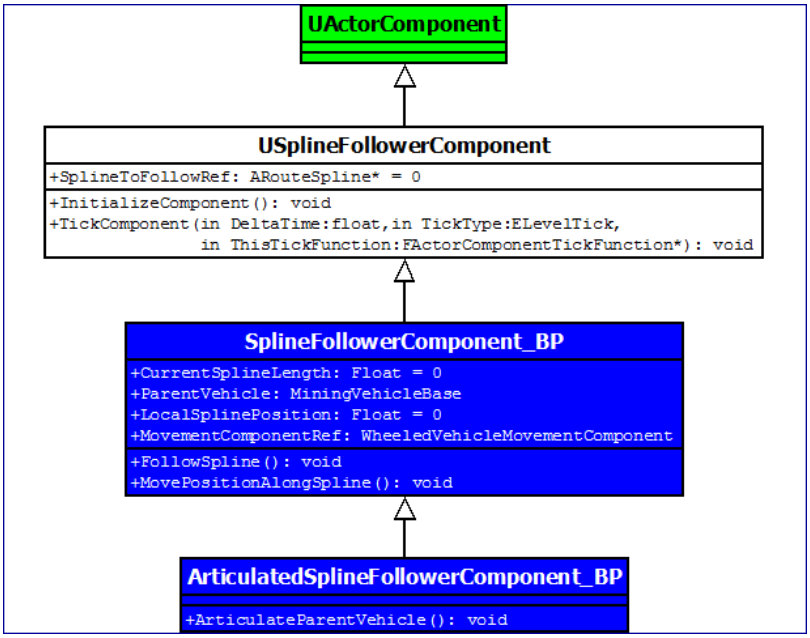
6.5 Reitinseurantakomponentit

Ajoneuvoille tarvittiin järjestelmäosa, joka olisi vastuussa ajoneuvojen kulkemisesta niille annettuja splini-reittejä pitkin. Koska logiikka haluttiin ainakin aluksi toteuttaa BVS:llä, ohjasi kaivoskoneiden luokkahierarkian rakenne reitinseurantalogiikan lopulta omaan komponenttiinsa. Unreal Enginestä löytyi ActorComponent-luokka, jonka pääasiallinen tarkoitus oli toimia kantaluokkana komponenteille, jotka laajentavat Actor-tyyppisien luokkien ominaisuuksia ja käyttäytymistä (ActorComponent n.d.). Tämä mahdollisti sen, että ajoneuvo saataisiin seuraamaan reittiä yksinkertaisesti lisäämällä Unreal Editorissa reitinseurantakomponentti niiden ajoneuvoluokkien komponenttilistalle, joille haluttiin lisätä kyky seurata reittejä. Tätä varten luokan esittelyn yhteydessä olevaan UCLASS-makroon täytyi lisätä avainsana ”meta=(BlueprintSpawnableComponent)”.

Tästä lähtökohdasta luotiin ActorComponentista peritty SplineFollowerComponent-luokka, joka toimii C++ -kantaluokkana splini-reittejä lukeville Blueprint-luokille. Tähän luokkaan lisättiin UPROPERTY-huomautuksella ARouteSpline-tyyppinen SplineToFollowRef-muuttuja, jonka tarkoitus on viitata siihen reittiin, jota ajoneuvo parhaillaan seuraa.

Itse reitin seuraaminen toteutettiin SplineFollowerComponent-luokasta perittyyn SplineFollowerComponent_BP-nimiseen Blueprint-luokkaan. Tähän luokkaan lisättiin Unreal Editorissa muutamia paikallisia referenssimuuttujia helpottamaan logiikan rakentamista. Näistä tärkeimmät olivat komponentin omistajaan viittaava ParentVehicle (MinigVehicleBase) sekä omistajan liikkeeseen viittaava MovementComponentRef (WheeledVehicleMovementComponent), josta saatiin luettua muun muassa ajoneuvon nopeus. Lisäksi SplineFollowerComponent_BP-luokkaan lisättiin paikallinen float-tyyppinen matkamittari LocalSplinePosition ja parhaillaan seurattavan reitin pituuden tallentava float-tyyppinen CurrentSplineLength.

Nivellettyjä kaivoskoneita varten luotiin vielä SplineFollowerComponent_BP-luokasta peritty ArticulatedSplineFollowerComponent_BP-luokka, joka laajensi kantaluokkansa toimintaa lisäämällä reitinseurantaan ajoneuvojen taittumisella kurvien mukaisesti. Kuviossa 13 on esitetty oleellisilta osin reitinseurantakomponenttien luokkahierarkia.

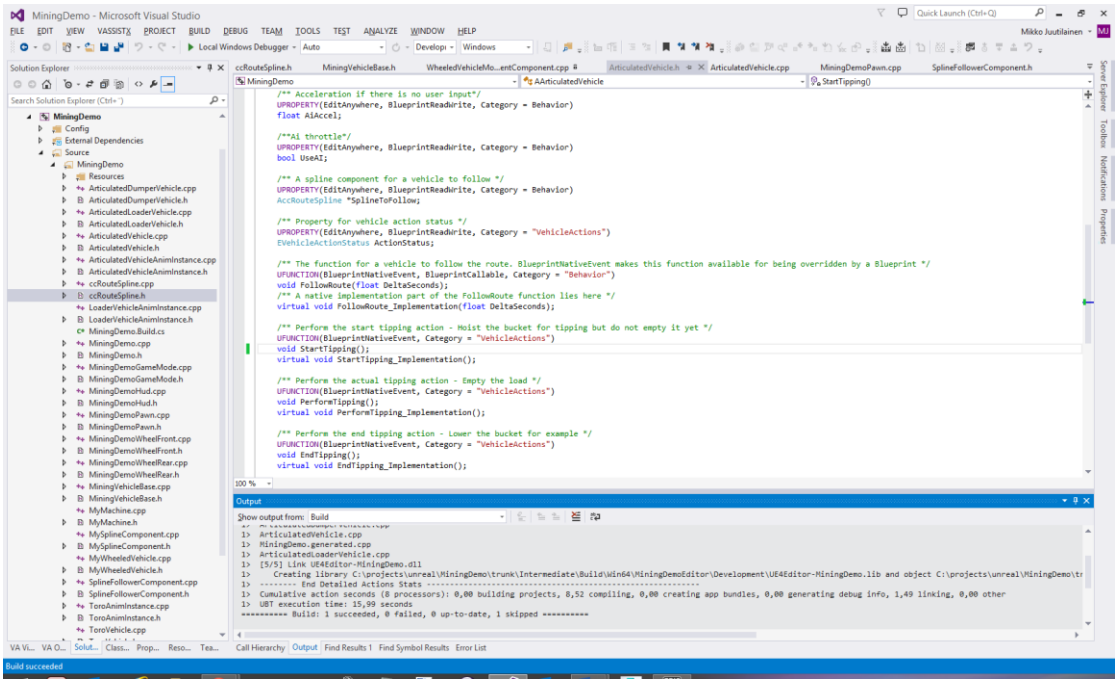


Kuvio 13. Reitinseurantakomponenttien luokkarakenne

6.6 Microsoft Visual Studio 2013

Unreal Engine 4 -kehitysympäristö oli opinnäytetyön kirjoitushetkellä saatavilla sekä Microsoftin Windowsille että Applen OS X:lle. Kummallekin järjestelmälle oli yksi Unreal Engine 4:n tukema IDE (Integrated Development Environment) eli integroitu kehitysympäristö. Windowsilla se oli Microsoft Visual Studio 2013 ja OS X:llä Xcode. Näistä Xcode oli lähtökohtaisesti ilmainen, ja Visual Studiosta oli myös saatavilla ilmainen Express Edition, joka on rajoituksistaan huolimatta täysin toimiva kehitysympäristö Unreal Engine 4:n kanssa työskentelyyn. Koska työn kohteena olivat Windows-pohjaiset kaivossimuloinnit, oli Visual Studio käytännössä ainoa vaihtoehto työn kehitysympäristöksi. Kirjoitushetkellä ainoa Unreal Engine 4:n tukema versio Visual Studiosta oli 2013.

Kuviossa 14 nähdään työssä käytetty yleisnäkymä Visual Studio 2013 -kehitysympäristöstä. Vasemmassa reunassa on projektin luokkanäkymä. Keskellä hallitsevassa osassa on tekstieditori koodin kirjoittamista varten. Tekstieditorin alla on tulostusikkuna kääntäjän ja ajettujen ohjelmien ilmoituksia varten.



Kuvio 14. Microsoft Visual Studio 2013 -kehitysympäristö

7 TYÖN TOTEUTUS

7.1 Kaivoskoneiden 3D-mallinnus

7.1.1 Lähtökohdat

Kaivoskoneiden animointimallia varten täytyi kaivoskoneista olla olemassa myös polygonimallit, joita animoida ja liikutella. Polygonimallien tekeminen on paljon aikaa vievä prosessi, joten pohjalle otettiin työn tilaajan kaivossimulaatioissa valmiiksi käytössä olevat Toro0011-kauhakuormaajakoneen sekä Toro50-kippiauton mallit. Näitä malleja tuli työstää siten, että ne täyttäisivät Unreal Enginen ja animoinnin asettamat vaatimukset.

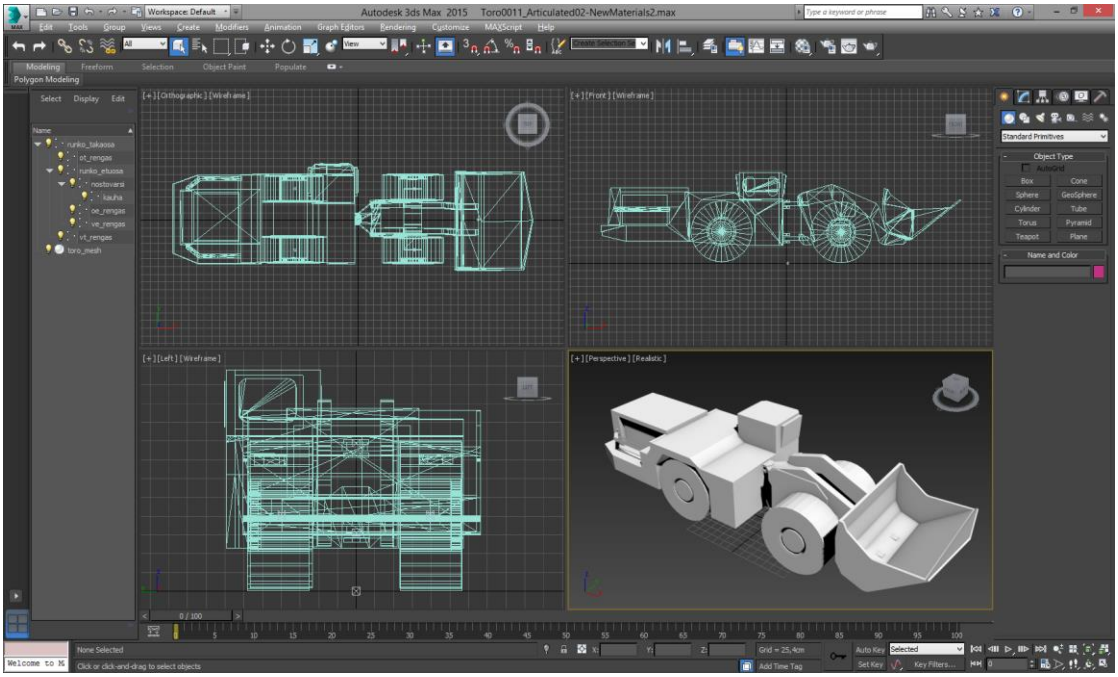
Ensimmäinen kriteeri oli, että mallien täytyi olla Autodesk FBX-formaatissa. Unreal Editor sisältää monipuolisen FBX-importterin, joka tuo lähdetiedostosta geometrian lisäksi editoriin myös mallin käyttämät materiaalit ja tekstuurit. Toinen kriteeri oli mallien luurankorakenne, joka vaati samalla sen, että mallin polygonit ovat yhdessä kappaleessa. FBX-importteri tunnistaa myös lähdetiedoston sisältämän luurankorakenteen ja luo tuonnin yhteydessä sitä vastaavat luuranko- ja fysiikka-assetit. Jotta mallit voisivat hyödyntää Unreal Enginen WheeledVehicle-ajoneuvoluokan tarjoamia ominaisuuksia, täytyi niillä olla luuranko, josta löytyy yksi kantaluu (engl. root bone) sekä oma luu jokaiselle renkaalle. Luurankomalli valittiin myös siksi, että uusien kaivoskoneiden lisääminen tulisi olemaan tulevaisuudessa suoraviivaista – vain yksi polygonikappale, josta löytyy yhdenmukainen luurankorakenne, ja se on suoraan yhteensopiva jo olemassa olevien animaatioiden kanssa (ks. 4 Skeletal Animation). Varjopuolena tällaisessa Skinned Meshissä on se, että jos polygonikappaleeseen tehdään muutoksia, vaatii se usein muutoksia myös pisteiden ja luurankojen sidoksiin. Tästä johtuen mallin polygonikappale kannattaa pyrkiä saamaan lopulliseen muotoonsa ennen Skinning-vaihetta (ks. 7.1.4 Skinning).

7.1.2 3DS Max 2015

Kaivoskoneiden 3D-mallien käsittelyyn valittiin Autodesk 3DS Max 2015 -mallinnusohjelma. Työn olisi voinut tehdä monella muullakin työkalulla, kuten Autodesk Maya tai Blender, mutta valintaan vaikutti muutama tekijä. Ensinnäkin, 3DS Maxin käyttö

oli työn tekijälle ennestään tuttua niin opintojen kuin työnkin puolesta. Monipuolisuudestaan tunnetusta mallinnusohjelmasta löytyi myös kaikki ne ominaisuudet, joita mallin työstämiseen tarvittiin. Yhtenä valintaan vaikuttavana tekijänä oli myös fakta, että FBX-tiedostoformaatti on nykyisin saman Autodesk-yrityksen omistama (FBX 2015).

Tästä johtuen 3DS Maxin tuottamat FBX-viennit ovat suurella todennäköisyydellä laadukkaita. Viimeisenä valintaan vaikuttavana tekijänä mallinnustyön pohjaksi otetut kaivoskoneiden mallit olivat valmiiksi 3DS Maxin omassa tiedostoformaatissa. Kuviossa 15 nähdään työhön valitun 3DS Max 2015 -mallinnusohjelman käyttöliittymän yleisnäkymä.



Kuvio 15. 3DS Max 2015 -mallinnusohjelma

7.1.3 Pohjamallin käsittely

Pohjamallien polygonikappaleet olivat lähtötilanteessa Editable Mesh -tyyppisiä, eikä malleissa ollut ollenkaan luurankoa. Niissä oli keyframe-animointi, joka ei sisältänyt lainkaan koneiden taivutusta. Tästä johtuen mallien hierarkioista puuttui myös etu- ja takaosaa erottava nivellys, ja rungot muodostuivat yhdestä kokonaisesta polygonikappaleesta. Mallien muukin geometria oli jaettu animointia varten loogisesti useaan staattiseen polygonikappaleeseen. Esimerkiksi kauhakuormaajakone-mallin

nostovarsi ja kauha olivat erillisiä kappaleita. Animointi oli toteutettu nivelten kohdalle asetettujen apupisteiden ja kappaleiden välisen hierarkian avulla, jolloin kappaleiden liike saatiin aikaiseksi vastaavia apupisteitä kääntämällä.

Koska vaatimuksena oli yhtenäinen polygonirakenne, liitettiin kaivoskonemallien erilliset polygonikappaleet yhteen valitsemalla yksi polygonikappaleista ja käyttämällä Editable Meshin Attach List -toimintoa. Avautuvasta ”Attach List” -ikkunasta valittiin kaikki geometria-tyyppiset kappaleet ja painettiin Attach-nappia. Samalla poistettiin turhaksi jääneet apupisteet, jotka oli aiemmin liitetty erillisiin kappaleisiin.

Pohjamallit olivat jo lähtökohtaisesti väritettyjä, mutta väritystä paranneltiin hieman lisäämällä renkaisiin teksturointi. Malleja myös siirrettiin siirtotyökalulla siten, että koneiden renkaiden alareunat olivat korkeussuunnassa 0-tasolla, ja keskinivelet olivat ylhäältä katsoen origossa. Tämä asemointi helpottaisi kaivoskoneiden rungon kääntöanimointia tulevaisuudessa.

7.1.4 Luurangon luominen

Unreal Editorin FBX-tuonnissa on rajoitettu mallin luurangon kantaluiden (engl. root bone) määrä yhteen, ja muut mallin luut tulee tästä johtuen asettaa hierarkiassa sen alle (FBX Import Errors n.d.). Kantaluu edustaisi perinteisessä ajoneuvomallissa koko runkoa. Nivelletyssä ajoneuvossa runko kuitenkin taittuu keskeltä, joten rungon etu- ja takapään osat tarvitsevat omat luunsa. Kaivoskonemallien luut luotiin 3DS Maxin Bones-systeemillä, joka löytyy käyttöliittymän komentopaneelista Create → Systems-välilehden alta.

Kaivoskonemallien luurangon kantaluuksi luotiin rungon takaosaa edustava luu ja sille annettiin nimeksi ”runko_takaosa”. Rungon etuosaa edustava luu olisi voitu valita yhtä hyvin kantaluuksi, mutta edellä mainittujen rajoitusten vuoksi etu- ja takaosien luut eivät voineet sijaita loogisesti samalla hierarkian tasolla. Kantaluu täytyi asemoida mallinnusympäristön origoon, sillä Unreal Enginessä luurankomallin (Skeletal Mesh) napapiste määritellään aina kantaluun sijaintiin. Sen oletetaan olevan origossa, jotta Unreal Enginen ajoneuvoluokkien tarjoamat simulointi- ja animointipalvelut toimisivat oikein. (FBX Skeletal Mesh Pipeline n.d.)

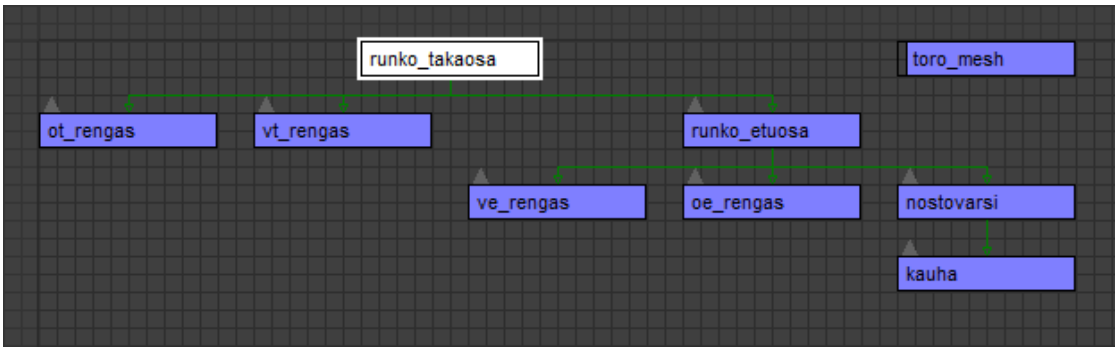
Rungon etuosalle luotiin oma luu, jolle annettiin nimeksi ”runko_etuosa”. Se asetettiin kantaluun tavoin mallinnusympäristön origoon. Tämä helpottaisi koneen etu- ja takaosien erillistä hallintaa, sillä koneen keskinivel oli asemoitu myös origoon. Rungon etuosan luu asetettiin luurankohierarkiassa rungon takaosaa edustavan kantaluun alle Schematic View -näkymässä.

Koneen jokaista rengasta kohden luotiin oma luunsa, ja kukin niistä asemoitiin vastaavan renkaan keskipisteeseen. Luille annettiin nimet ”oe_rengas” (oikea eturengas), ”ve_rengas” (vasen eturengas), ”ot_rengas” (oikea takarengas) ja ”vt_rengas” (vasen takarengas). Takarenkaiden luut asetettiin hierarkiassa suoraan kantaluun alle, sillä niiden oli tarkoitus seurata koneen takaosan liikettä. Eturenkaiden luut asetettiin samasta syystä hierarkiassa rungon etuosaa edustavan luun alle. Normaalin einivelletyn ajoneuvon tapauksessa kaikki neljä rengasta olisi yhdistetty saman kantaluun alle.

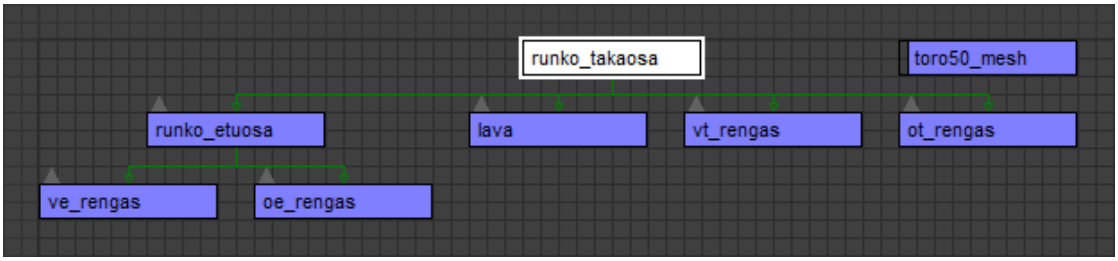
Edellä mainittu luuhierarkia määritti pohjarangan kaikille nivelletyille kaivoskoneille. Kullekin konetyypille piti kuitenkin lisätä vielä luut tyyppikohtaista käyttäytymistä sekä toiminta-animaatiota varten. Kauhakuormaajakoneen tapauksessa luotiin luut nostovartta sekä kauhaa varten. Nostovarren luulle annettiin nimi ”nostovarsi” ja se asemoitiin nostovarren juureen. Kauhan luulle nimeksi annettiin ”kauha” ja se asemoitiin nostovarren toiseen ääripäähän kippaamista varten. Kauhan luu asetettiin hierarkiassa nostovarren alle, kun taas nostovarren luu asetettiin rungon etuosaa edustavan luun alle. Kippiauton lavaa varten luotiin perusrangan lisäksi ainoastaan yksi luu joka sai nimen ”kippilava”. Se asetettiin hierarkiassa takaosan eli kantaluun alle, ja asemoitiin lavan kippauspisteeseen.

Ennen seuraavan Skinning-työvaiheen aloittamista oli tärkeä varmistaa, että kaikkien luiden napapisteet (pivot point) olivat linjattuna maailman koordinaatiston mukaisesti (X eteen, Y vasemmalle ja Z ylös). Muussa tapauksessa sekä fysiikka että animointi menivät Unreal Editorin puolella sekaisin. Linjaus tehtiin valitsemalla kaikki mallin luut ja valitsemalla komentopaneelin Hierarchy-välilehden alta ensin Affect Pivot Only ja sen jälkeen Align to World.

Kuviot 16 ja 17 esittävät visuaalisesti kauhakuormaajan ja kippiauton valmiit luurankorakenteet. Kuvioissa valkoisella taustalla näkyvät rungon takaosan luut ovat luurankohierarkioiden kantaluita (root bone). Kantaluihin yhdistyy suoraan seuraavalla hierarkiatasolla koneiden takarenkaiden luut sekä rungon etuosien luut. Kippiauton tapauksessa myös lava yhdistyy kantaluuhun. Rungon etuosan luihin taas yhdistyvät koneiden eturenkaiden luut sekä kauhakuormaajan tapauksessa nostovarren luu. Nostovarren luuhun yhdistyy vielä kauhan luu.



Kuvio 16. Kohdistettu näkymä 3DS Maxin Schematic Viewiin - Kauhakuormaaja-tyyppisen nivelletyn kaivoskoneen luurankohierarkiasta



Kuvio 17. Kodistettu näkymä 3DS Maxin Schematic Viewiin – Kippiauto-tyyppisen nivelletyn kaivoskoneen luurankohierarkia

7.1.5 Skinning

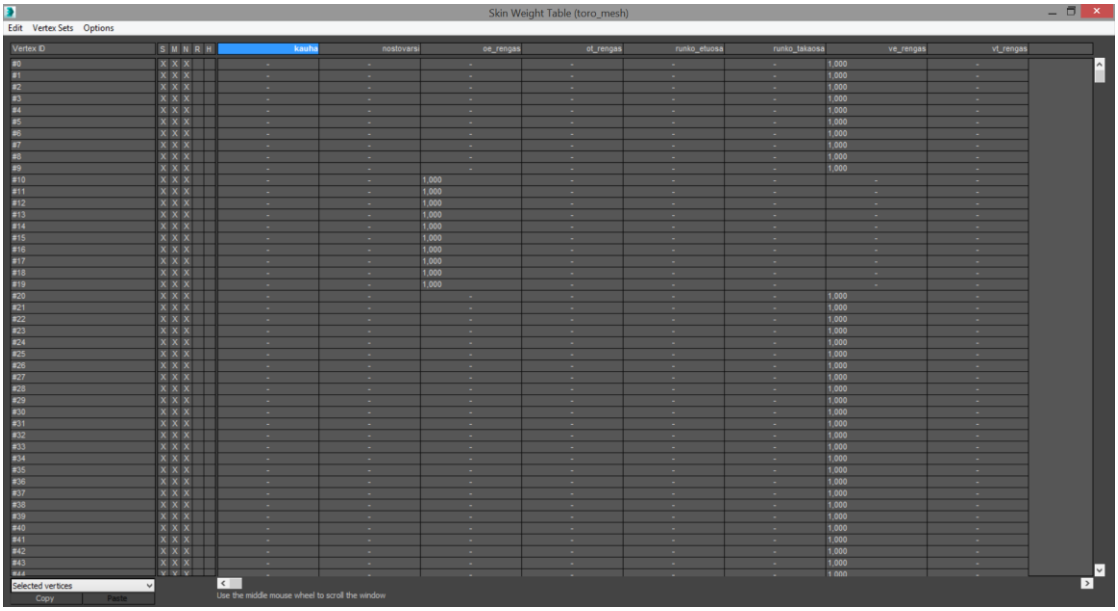
Kun kaikki koneen animointia varten tarvittavat luut oli asemoitu paikoilleen ja asetettu oikeaan paikkaan hierarkiassa, oli seuraavaksi vuorossa mallin pisteiden yhdistäminen luurankoon. Tähän käytettiin Skinning-tekniikkaa (ks. 4 Skeletal Animation).

Kaivoskoneiden mallit olivat jaettavissa selkeisiin loogisesti erillisiin kappaleisiin, joista kullekin oli määritelty oma luunsa. Tästä johtuen kaikki mallin pisteet voitiin täysipainoisesti yhdistää siihen luuhun, joka edusti kappaletta, johon piste kuului. Näin ollen esimerkiksi kaikki oikean eturenkaan pisteet voitiin yhdistää eturengasta

edustavaan luuhun painoarvolla 1. Samoin voitiin jatkaa kaikkien mallin pisteiden kanssa, kunnes ne oli yhdistetty omiin luihinsa.

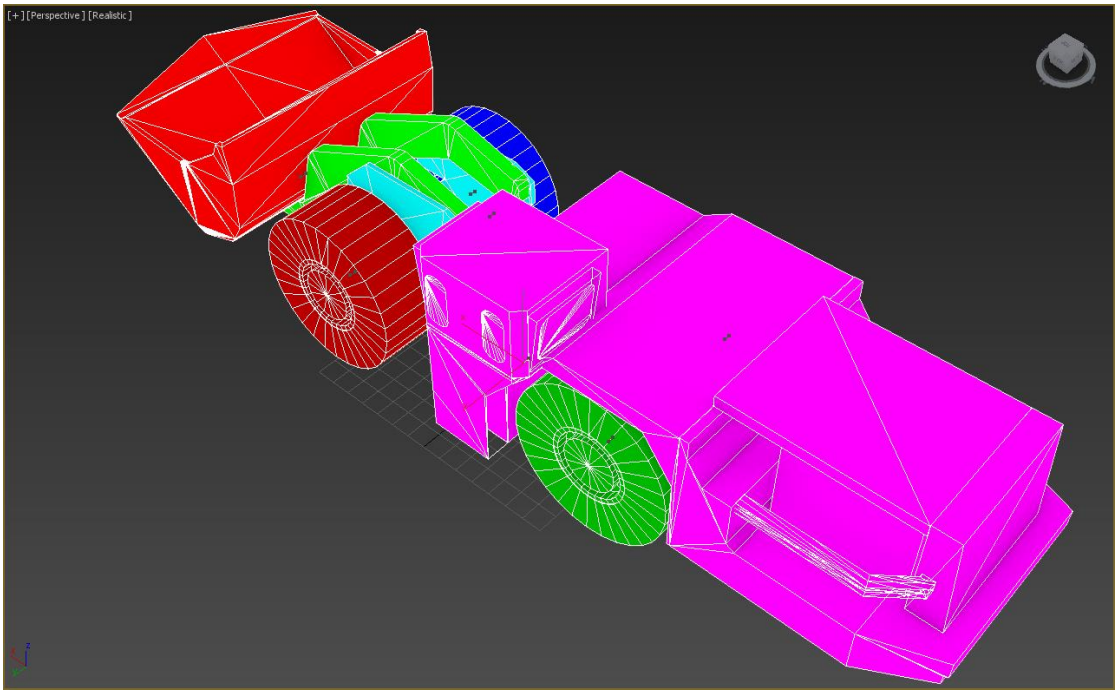
Skinning-toimenpidettä varten valittiin kaivoskoneen polygonimalli ja lisättiin sen määritteisiin komentopaneelin Modify-välilehden määrite-listasta kohta Skin. Tämä määrite valittiin polygonimallin määritelistalta ja siihen lisättiin kaikki kaivoskoneen luurankoa varten luodut luut painamalla ”Bones: Add” -nappia ja valitsemalla avautuvasta ”Select Bones” -ikkunasta kaikki luut. Seuraavaksi luiden täytyi saada lihaa ympärilleen. Tätä varten Parameters –ryhmästä painettiin päälle ”Edit Envelopes” ja valintaryhmästä asetettiin päälle ”Vertices”. Nyt polygonimallista saattoi valita pisteitä painoarvojen asettamista varten. Tätä toimenpidettä varten avattiin ”Weight Properties”-ryhmän alta ”Skin Weight Table”-ikkuna.

Skin Weight Table on ikkuna, jossa näkyy taulukkomuodossa vaihtoehtoisesti joko valittujen tai kaikkien pisteiden painoarvot Skin Modifieriin lisättyjen luiden suhteen. Kuten aiemmin mainittiin, kunkin pisteen tuli olla yhdistettynä täysipainoisesti siihen luuhun joka edusti pisteen kanssa samaa kappaletta. Alkutilanne ei ollut lähellekään tätä, sillä Skin Modifier yritti heti aluksi yhdistää automaattisesti mallin pisteet lähellä oleviin luihin. Tämä enemmänkin sekoitti tilannetta, joten ikkunan vasemmasta alareunasta valittiin valintaryhmäksi All Vertices. Tämän jälkeen kaikki taulukon alkiot valittiin Select All –valinnalla ja kaikki taulukon painoarvot nollattiin. Kun painoarvot oli resetoitu, valittiin valintaryhmäksi Selected Vertices. Tämän jälkeen taulukon otsikkorivistä aktivoitiin luu, ja kaivoskoneen polygonikappaleesta valittiin näkymää kääntelemällä niitä pisteitä jotka haluttiin yhdistää aktivoituun luuhun. Pisteiden painoarvoksi annettiin 1. Tätä toimenpidettä jatkettiin, kunnes jokainen kaivoskoneen piste oli yhdistettynä omaan luuhunsa. Kuviossa 18 nähdään Skin Weight Table-ikkuna.



Kuvio 18. Skin Modifierin Skin Weight Table

Kuviossa 19 näkyy esimerkki kauhakuormaajan Skinning-vaiheen lopputuloksesta, jota kutsutaan yleisesti muun muassa termeillä Skinned Mesh, Skeletal Mesh tai Rigged Mesh. Mallin jokaisen luun pisteet näkyvät kuvassa omalla värillään. Etuosan luun pisteet näkyvät kuvassa syaanilla, nostovarren vaaleanvihreällä ja kauhan vaaleammalla punaisen sävyllä. Jokaisen mallin renkaan luiden pisteet näkyvät lisäksi kuvassa omalla värillään. Valmis malli vietiin 3DS Maxista ulos Export-toiminnolla Unreal Editorin ymmärtämässä Autodesk FBX-formaatissa.

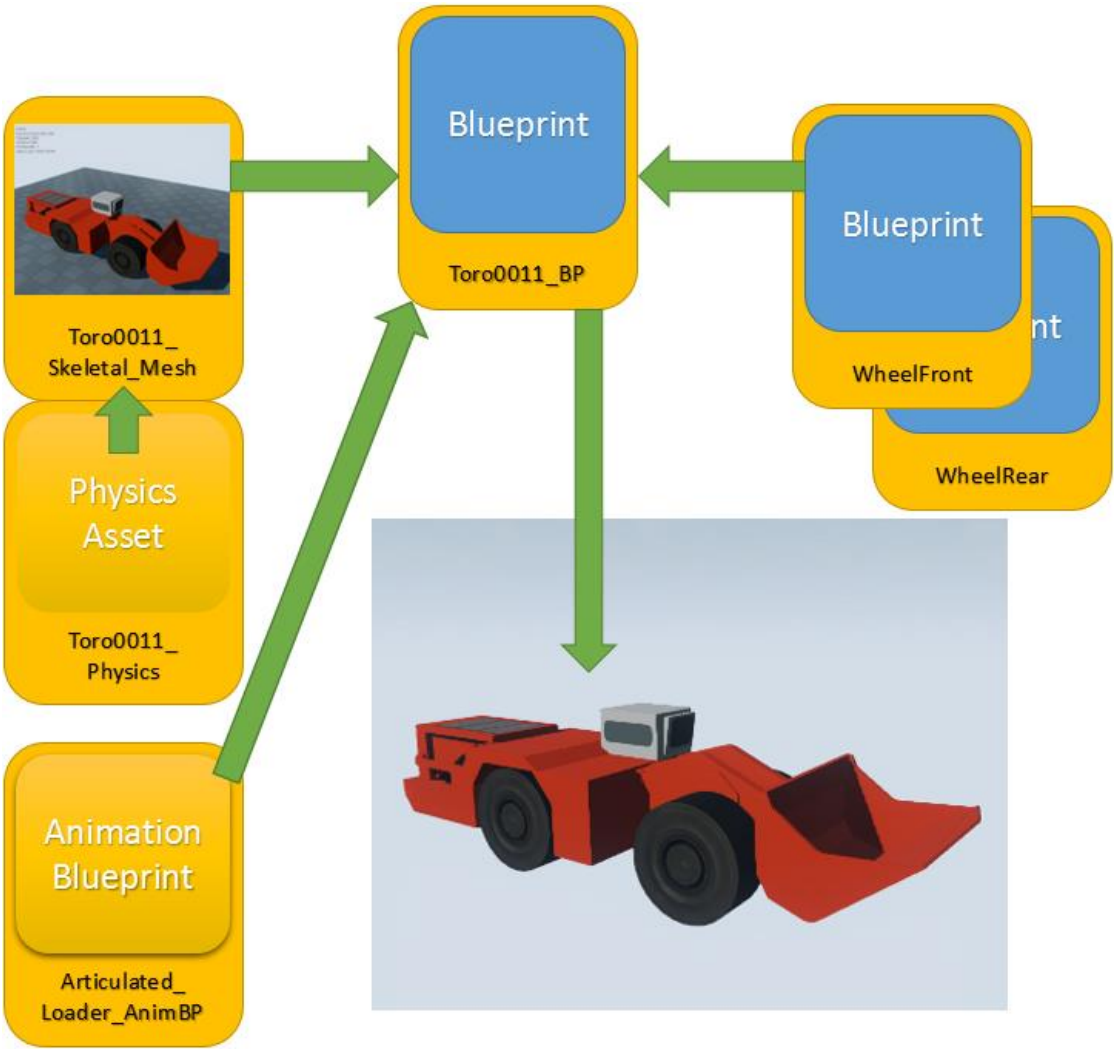


Kuvio 19. Kauhakuormaajakonemallin pisteet yhdistettynä luihin

7.2 Kaivoskoneiden rakentaminen

7.2.1 Kaivoskoneiden rakenne

Kaivoskoneet rakentuvat Unreal Enginen WheeledVehicle-tyyppisen ajoneuvon mukaisesti useasta eri komponentista, jotka muodostavat yhdessä toimivan kokonaisuuden. Kuviossa 20 on esitetty kaivoskoneiden tarvitsemat komponentit. Keskiössä on kaivoskoneen lopullinen Blueprint-luokka, joka koostuu Animation Blueprintistä, koneen luurankomallista ja fysiikka-assetista sekä renkaat määrittelevistä Blueprint-luokista. Kaivoskoneen liikkuminen määritellään lopullisessa Blueprint-luokassa sijaitsevien komponenttien avulla.



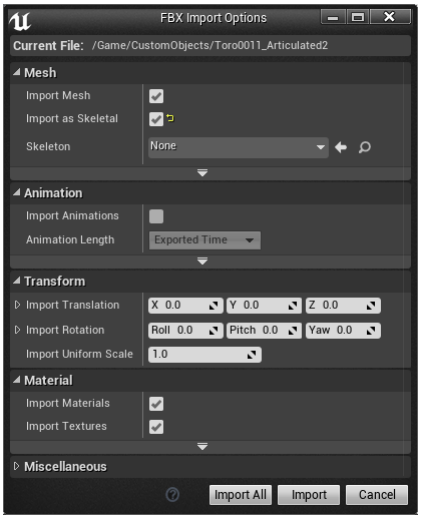
Kuvio 20. Kaivoskoneen koostumus Unreal Enginessä

7.2.2 Mallin valmistelu Unreal Editorissa

Ennen kuin kaivoskoneiden luurankomalleille voitiin toteuttaa liikkumista tai animointia, täytyi niille tehdä alustavia toimenpiteitä. Toimenpiteillä ne tuli saattaa yhteensopiviksi sekä Unreal Engine 4:n ajoneuvo- että animointikomponenttien kanssa.

Mallien tuonti

Mallinnusvaiheen jälkeen kaivoskoneen malli tuotiin Unreal Editoriin FBX Import-toiminnolla. Tuontityökalu tunnisti mallissa olevan luurankorakenteen ja ehdotti automaattisesti mallin importoimista sen kanssa (Import as Skeletal). Kuviossa 21 nähdään Unreal Editorin FBX-tuontityökalun näkymä kaivoskoneiden tuontivaiheessa.



Kuvio 21. Unreal Editorin FBX Import-näkymä

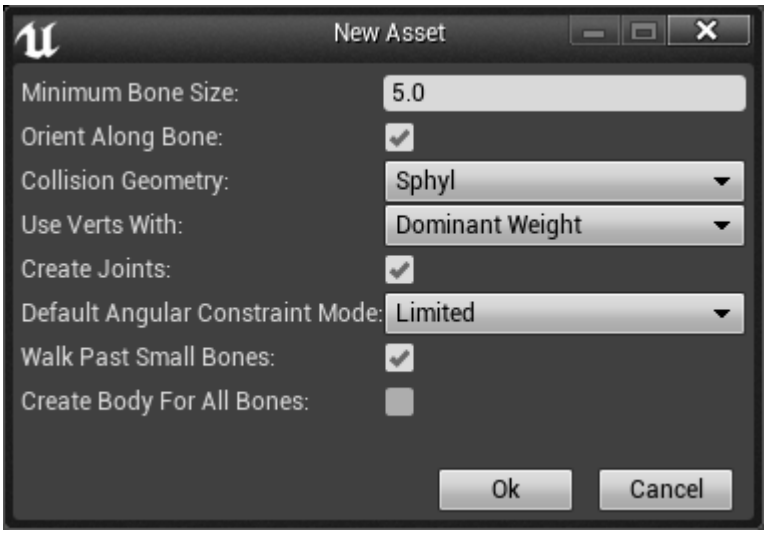
Tuontityökalu loi FBX-mallin pohjalta Skeletal Meshin, fysiikka-assetin, skeleton-assetin sekä materiaali- ja tekstuuri-assetit.

Fysiikan säätö PhAT-työkalulla

Kaivoskoneille luotiin FBX-importtauksen yhteydessä automaattisesti fysiikka-assetit, mutta ne eivät soveltuneet sellaisenaan käytettäväksi, sillä Physics Asset Tool (PhAT) yrittää parhaansa mukaan luoda jokaiselle nivelelle sellaiset kappaleet (body), että kaikki niveleen skinnatut verteksit saadaan käärittyä sen sisälle (Vehicle Art Setup n.d.). Tästä johtuen muun muassa kaivoskoneiden korin fysiikkakappaleet olivat suuria palloja, jotka jo itsessään estäisivät koneiden renkaiden kosketuksen maahan. Lisäksi koneiden toiminta-animaatioita varten lisättyjen nivelien (nostovarsi, kauha ja

kippilava) fysiikkakappaleet olivat ainakin toistaiseksi turhia, sillä niiden fysiikkaa ei ollut tarkoitus mallintaa ollenkaan.

Selkeyden vuoksi kaivoskoneiden kaikilta niveliltä poistettiin aluksi fysiikkakappaleet, ja niille lisättiin sen jälkeen tarpeen mukaan hierarkkisessa järjestyksessä uudet, paremmin käyttötarkoitukseen soveltuvat kappaleet. Ensimmäinen fysiikkakappale luotiin kaivoskoneiden juurena toimivalle takaosaa edustavalle nivelelle. Tämä kappale sai edustaa koko koneen runkoa, joten se määriteltiin laatikon malliseksi ja asetettiin kutakuinkin koneen keskelle. Koneiden kaikki renkaat saivat siten myös oman pallon malliset fysiikkakappaleensa. Pallot saivat renkaiden kanssa yhteneväiset säteet ja keskipisteet, joten ne tulivat täsmällisesti renkaiden geometrian ympärille. Kuviosta 22 nähdään uuden fysiikkakappaleen luontiin tarkoitettu dialogi.



Kuvio 22. Uuden fysiikkakappaleen (body) luonti PhAT-työkalussa

Mallien testausprosessi

Ennen varsinaisten kaivoskoneluokkien käyttöönottoa, kaivoskoneiden mallit kävivät läpi lyhyen testausprosessin. Testausprosessin tarkoituksena oli varmistaa, että mallien luuhierarkiat ja PhATissa säädetyt fysiikat olivat yhteensopivia Unrealin geneeristen ajoneuvokomponenttien kanssa. Mallien kanssa voitiin jatkaa, mikäli ne toimivat ongelmitta näiden komponenttien kanssa, sillä kaivoskoneiden kantaluokissa hyödynnettiin samoja komponentteja. Testausprosessissa ei huomioitu vielä koneiden taittumiseen tai toiminta-animaatioihin liittyviä nivellyksiä.

Testausta varten luotiin Unreal Editorissa AWheeledVehicle-luokasta peritty Blueprint-luokka sekä VehicleAnimInstance-luokasta peritty Animation Blueprint. Kaivoskoneiden mallit asetettiin testiluokan Skeletal Mesh-komponenttiin, ja VehicleMovement-komponentin rengas-asetuksiin syötettiin mallien renkaita vastaavien luiden nimet. Animation Blueprint-komponentin asentoketjuun lisättiin ainoastaan MeshSpacerefPose- ja WheelHandler-solmut, joista asento vietiin suoraan FinalAnimationPose-solmulle. Vastaavien prosessien tarkemmat kuvaukset löytyvät tämän opinnäytetyön osioista Animation Blueprint ja Kaivoskonetyyppien Blueprint-kantaluokat.

Onnistuneen testausprosessin lopputuloksena oli ajoneuvo, jolla pystyi pienen hienosäädön jälkeen ajelemaan vapaasti testiympäristössä.

7.2.3 Reitinseuranta-komponenttien toiminta

Tässä osassa perehdytään reitinseurantakomponenttien (SplineFollowerComponent) sisältämien toimintalogiikoiden BVS-toteutuksiin. Komponenttien rakenne ja käyttötarkoitus selitetään arkkitehtuuri-osassa.

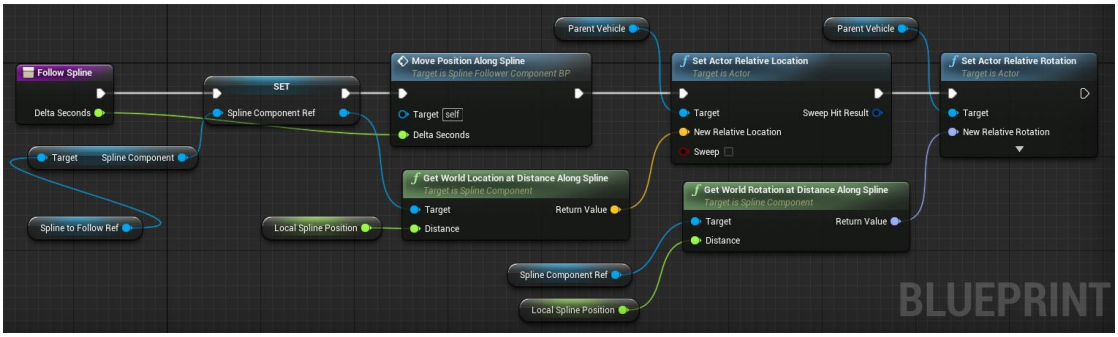
Kun SplineFollower-komponentti lisätään kaivoskoneen Blueprint-luokan komponenttilistaan, alkaa se hakemaan koneen ARouteSpline-tyyppistä SplineToFollow-muuttujaa. Heti kun muuttujalla on validi arvo, alkaa komponentti liikuttamaan isäntäänsä pitkin koneelle määrättyä splini-käyrää. Mikäli lisätty komponentti on ArticulatedSplineFollowerComponent_BP-tyyppinen, se taivuttaa reitinseurannan lisäksi isäntäänsä kurvien mukaisesti.

SplineFollowerComponent_BP-komponenttiin toteutettiin FollowSpline()-funktio, jota kutsutaan joka syklillä, kun isännän reittisplinillä on validi arvo. Funktio kutsuu heti suorituksen aluksi MovePositionAlongSpline()-funktiota, joka lukee isäntä-ajoneuvon liikekomponentista ajonopeuden ja kertoo sen sykliin kuluneella ajalla. Näin saadaan syklin aikana kuljettu matka, joka lisätään LocalSplinePosition-muuttujaan.

Isäntä-ajoneuvon SplinePosition-muuttujaa pidetään jatkuvasti ajan tasalla synkronoimalla se LocalSplinePositioni-muuttujan arvoon. Mikäli LocalSplinePosition ylittää

kuljetun splinin pituuden, lähetetään isäntä-ajoneuvolle OnRouteEnd()-tapahtuma-kutsu. Isäntä-ajoneuvo on näin ollen vastuussa siitä, mitä tapahtuu, kun reitti on kuljettu loppuun saakka.

Kun MovePositionAlongSpline()-funktio on saatu päätökseen, palataan FollowSpline()-funktioon. Funktion suoritusta jatketaan asettamalla isäntä-ajoneuvolle Unrealin apufunktioilla reittispliniä pitkin LocalSplinePosition-muuttujan verran kuljetun matkan kohdalta sijainti ja rotaatio. Kuviossa 23 on esitetty FollowSpline()-funktio kokonaisuudessaan.



Kuvio 23. SplineFollowerComponent_BP-luokan FollowSpline()-funktio

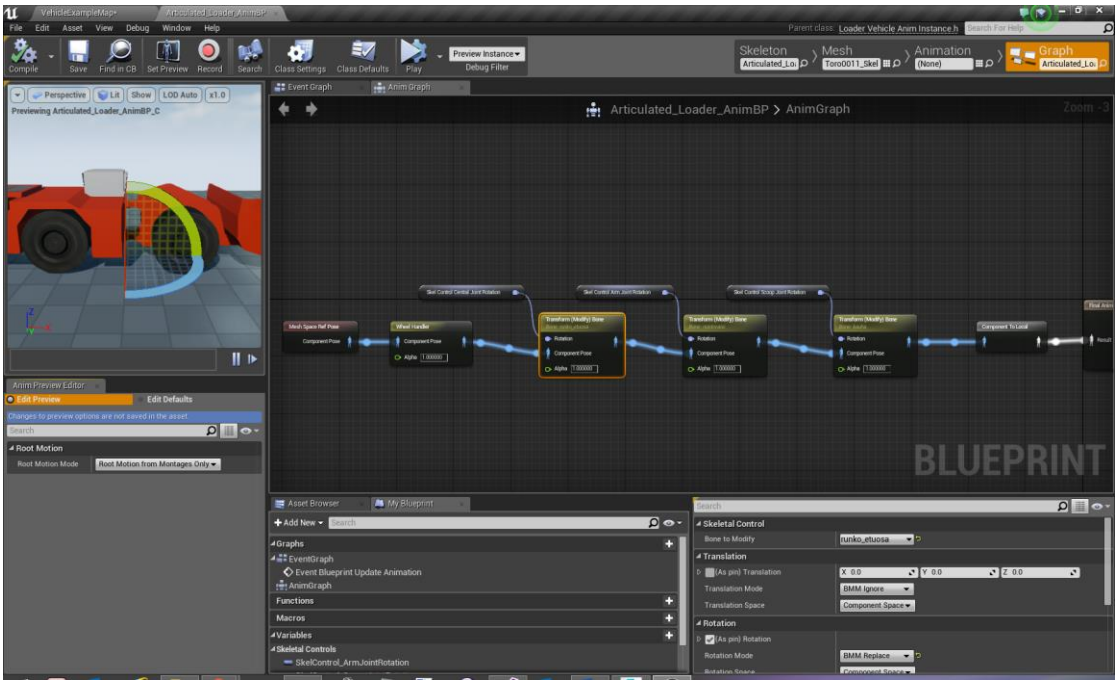
Nivelletyille kaivoskoneille luotiin oma ArticulatedSplineFollowComponent_BP -reitinseurantakomponenttinsa, joka periytettiin SplineFollowerComponent_BP-komponentista. Tähän nivellettyyn reitinseurantakomponenttiin toteutettiin isäntä-ajoneuvon taittuminen keskinivelen kohdalta reitin kurvien mukaisesti. Komponentti päivittää jokaisella syklillä kantaluokan reitinseurantaa ja siirtyy tämän jälkeen ArticulateParentVehicle()-funktioon. Tässä funktiossa lasketaan splini-käyrän rotaatiot kahdesta koneen akselivälin etäisyydellä olevasta näytepisteestä suhteessa koneen sijaintiin käyrällä. Näiden rotaatioiden erotuksesta saadaan kulma, joka asetetaan isäntä-ajoneuvon animaatio-instanssin (Animation Blueprint) keskinivellystä edustavaan luukontrolleriin.

7.2.4 Animation Blueprintit

Kauhakuormaajalla ja kippiautolla oli toisistaan poikkeava luurankorakenne. Tästä johtuen kumpikin ajoneuvotyyppi tarvitsi luurankonsa animointia varten oman Animation Blueprint -käsittelijänsä. Dynaamisen luonteensa vuoksi animaatioita ei voinut määrittää etukäteen, vaan lopputulos oli alati vaihtelevien nivelten asentojen

summa. Nivellettyjen kaivoskoneiden tapauksessa ajettavan reitin kurvit määrittelivät jatkuvasti keskinivelen asentoa. Toiminta-animaatioiden vaiheet tulevat todennäköisesti myös vaihtelemaan tilanteen mukaan. Tästä johtuen jokaiselle luulle luotiin animaatioinstanssien kantaluokkiin konetyyppikohtaisesti luukontrollerit, jotka määrittävät suoraan oman luunsa kääntymisen. Näin luukontrollereilla saatiin luotua mallin luurankohierarkiaa seuraava kääntymisketju. Dynaamisten luukontrollerien käytössä oli myös se etu, että luiden asentoja voidaan tulevaisuudessa ohjata tarvittaessa suoraan simulointiohjelmistosta käsin.

Kuviossa 24 esitetään Animation Blueprintin animointikaavio-näkymä (Anim Graph). Vasemmassa reunassa on animaation esikatselunäkymä. Hallitsevana elementtinä nähdään itse animointikaavio, johon luodaan animaation ketjutus. Alhaalla vasemmalla näkyvät muun muassa Blueprintiin kuuluvat muuttujat ja tapahtumat (events). Alas oikealle tulevat valitun solmun (node) tai komponentin ominaisuudet (properties) muokattavaksi.



Kuvio 24. Animation Blueprintin animointikaavio-näkymä

Animation Blueprintit luotiin Unreal Editorin käyttöliittymässä Toro0011-kauhakuormaajan ja Toro60-kippiauton importoinnin ohessa automaattisesti luotujen luurankoassettien pohjalta. Ensimmäinen Animation Blueprinteille tehtävä toimenpide oli niiden kantaluokan vaihtaminen konetyypin mukaiseksi. Kantaluokan mukana saatiin

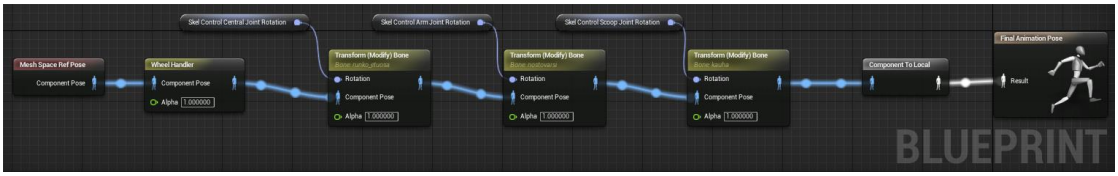
kaikki konetyypin mukaiset luukontrollerit animointikaavioon käyttöön. Koska konetyyppien animaatio-instanssien kantaluokat periytyvät Unrealin VehicleAnimInstance-luokasta, saatiin samalla käyttöön myös renkaiden animoinnin kannalta tärkeä WheelHandler-solmu (node).

Kantaluokan vaihdon jälkeen aloitettiin asentoketjun rakentaminen. Ketju lähti liikkeelle Mesh Space Ref Node-solmusta, joka toi animoinnin pohjaksi koneen lähtöasennon komponentti-tilassa (component-space). Komponentti-tilassa oletetaan luiden transformaatioiden olevan suhteessa pohjana olevaan malliin. Lähtöasento olisi ollut mahdollista tuoda ketjuun myös paikallis-tilassa (local-space) Local Space Ref Pose-solmulla, jolloin luiden transformaatioiden oletetaan olevan suhteessa itseensä. Tämä ei kuitenkaan ollut järkevä vaihtoehto, sillä kaikki luiden kontrollerit (Skeletal Controls) operoivat komponentti-tilassa, ja tilanvaihdon vaativa muunnos olisi ollut aikaa hukkaava operaatio, vaikka se olisikin ollut mahdollista (Convert Spaces Nodes n.d.). Lähtöasennosta jatkettiin WheelHandler-solmuun, joka käytännössä hoiti osaltaan renkaiden pyörimisliikkeeseen ja jousitukseen tarvittavan animoinnin. WheelHandler-solmusta ketjua jatkettiin konetyyppikohtaisten luiden kontrollereille.

Jokaista kaivoskoneen niveltä kohden lisättiin luiden hierarkkista järjestystä seuraten oma kääntymistä kontrolloiva Transform (Modify) Bone-solmunsa. Ensimmäinen näistä määriteltiin kummankin konetyypin tapauksessa kääntämään keskiniveltä edustavaa luuta. Tämä saatiin aikaiseksi valitsemalla solmun ominaisuuksista kohdeluuksi ”runko_etuosa” ja rotaatio-moodiksi (Rotation Mode) olemassa olevaan rotaatioon lisäävä (BMM Additive) ja rotaatio-tilaksi (Rotation Space) komponentti-tila (Component Space). Näin solmu sai tiedon kuinka kohdeluuta täytyi liikuttaa suhteessa muihin elementteihin. Seuraavat Transform (Modify) Bone-solmut tehtiin järjestyksessä nostovartta ja kauhaa (kauhakuormaaja) sekä kippilavaa (kippiauto) varten. Näiden ominaisuuksiin syötettiin aiemman esimerkin mukaisesti vastaavien kohdeluiden nimet, mutta rotaatio-tilaksi asetettiin Parent Bone Space. Näin luut ”perivät” hierarkiassa ylempänä olevan luun rotaation ja lisäsivät siihen omansa.

Kun kaikkien luiden kontrolleri-solmut oli lisätty, yhdistettiin viimeinen niistä jokaisessa Animation Blueprintissä esiintyvään Final Animation Pose-solmuun. Tämä solmu edustaa kaikkien komponenttien läpikäynnin jälkeistä asentoa. Väliin muodos-

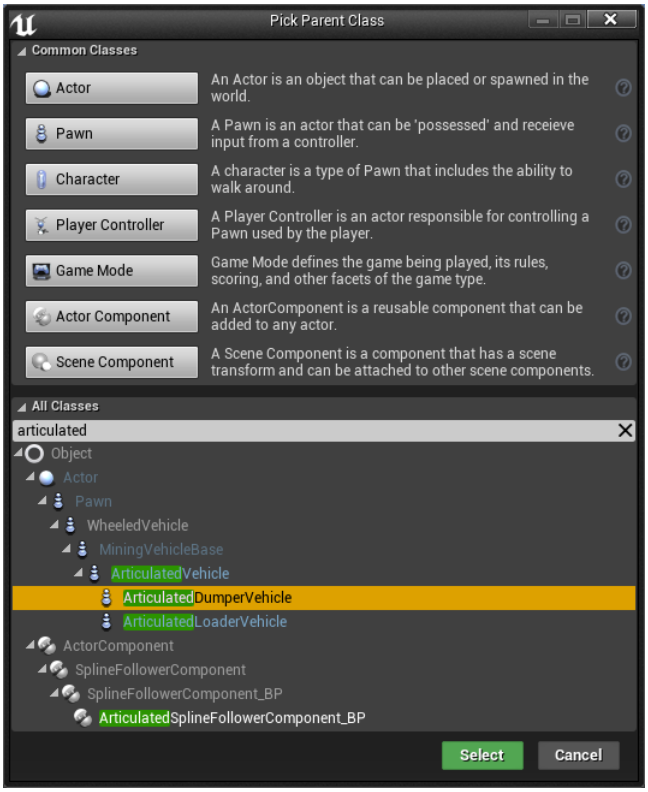
tui automaattisesti Component To Local-solmu, joka muunsi komponentti-tilassa ketjutetun asennon paikallistilaan. Kuviossa 25 on esitetty kauhakuormaaja-tyyppisen nivelletyn kaivoskoneen asennon ketjutus. Ylimpänä näkyvät luukontrolleri-solmuihin kytketyt, kantaluokista tulevat, rotaatiokontrollerit. Solmuista on piilotettu selkeyden takia luiden translaatioon ja skaalaukseen liittyvät liittimet, sillä niitä ei käytetä kaivoskoneiden animoinnissa lainkaan.



Kuvio 25. Nivelletyn kauhakuormaajan asentoketju Animation Blueprintissä

7.2.5 Kaivoskonetyyppien Blueprint-kantaluokat

Oletusarvoisesti kukin kaivoskonetyyppi sisältää erilaisen luurangon ja toisistaan poikkeavaa toiminnollisuutta. Tästä johtuen kauhakuormaaja- ja kippiauto-konetyypeille luotiin omat Blueprint-kantaluokkansa. Kuviossa 26 on esitetty uuden Blueprint-luokan luominen Unreal Editorissa. Näkymän yläosassa on joukko yleisimmin kantaluokkina käytettyjä komponentteja. Alaosan listassa näkyvät kaikki ne luokat, joista uusi Blueprint-luokka voi periytyä. Käytännössä listassa esiintyvät kaikki projektin sisältämät, UObjectista periytyvät luokat, joiden luokan reflektio-määritteistä (UCLASS) löytyy termi ”Blueprintable”. Kantaluokkaa voi hakea hakusanan avulla, kuten kuviossakin on tehty.



Kuvio 26. Uuden Blueprint-luokan luonti

Kauhakuormaaja-tyyppisen kaivoskoneen Blueprint-kantaluokalle annettiin nimeksi ”LoaderVehicleBP” ja kippiauto-tyyppisen kaivoskoneen vastaava luokka sai nimen ”DumperVehicleBP”.

Kun uudet Blueprint-luokat avattiin editoitavaksi, niiden komponenttilistat sisälsivät Unrealin AWheeledVehicle-kantaluokasta peräisin olevat VehicleMovement- ja Mesh-komponentit sekä AMiningVehicleBase-kantaluokan otsikkotiedostoissa UPROPERTY-reflektiomäärittеillä esitellyt muuttujat. Komponenttilistaan lisättiin ArticulatedSplineFollowerComponent_BP, joka sai kaivoskoneet seuraamaan annettua splinireittiä sekä taipumaan kurveissa keskinivelen kohdalta sitä seuratessaan.

Mesh-komponentin ominaisuuksista määriteltiin koneiden animointi. Koska konetyypeille oli tehty omat Animation Blueprintit, valittiin Animation Modeksi ”Use Animation Blueprint”. Seuraavaksi määriteltiin luokat, joista koneille haluttiin luoda Animation Blueprint-instanssit. Kauhakuormaajalle AnimBlueprint-luokaksi asetettiin Articulated_Loader_AnimBP ja kippiautolle Articulated_Dumper_AnimBP. Näiden animaatioluokkien instanssien kautta päästiin vaikuttamaan mallien nivelten asentoihin ja samalla toteuttamaan toiminta-animaatioita BVS:llä.

Wheel Setups

Renkaat ovat käytännössä jokaisessa konetyypissä aina samassa järjestyksessä ja samalla tavalla, joten rengas-asetukset tehtiin lähtökohtaisesti konetyyppikohtaisiin Blueprint-luokkiin. Näin jokaiselle koneelle ei tarvinnut tehdä uudelleen samaa työvaihetta. Perityillä luokilla oli kuitenkin tarvittaessa mahdollisuus korvata kantaluokkien rengas-asetukset, joten renkaiden konekohtainen muokattavuus pystyttiin kuitenkin säilyttämään.

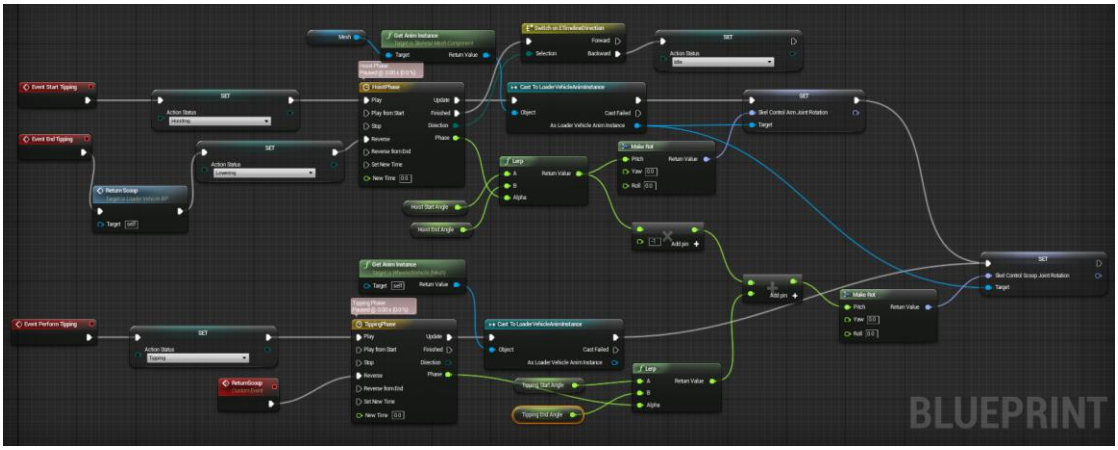
Jos ohjelman suoritus käynnistettiin puutteellisilla tai väärillä rengas-asetuksilla, saattoivat koneiden renkaat räjähdellä irti rungosta tai ne pyörähtelivät holtittomasti jokaisen akselinsa ympäri. Tämä johtui siitä, että renkaiden luut olivat puhtaasti koneiden fysiikka-assettien armoilla niin kauan, kun VehicleMovement-komponentti otti ne hallintaansa. Tätä varten komponentin rengas-asetuksista täytyi määrittää jokaiselle renkaalle mallissa vastaavan luun nimi. Luut piti asettaa oikeassa järjestyksessä (etu-vasen, etu-oikea, taka-vasen, taka-oikea), jotta komponentti osasi yhdistää luut oikeiden renkaiden paikkoihin. Samalla voitiin tehdä muita renkaisiin vaikuttavia asetuksia, kuten valita renkaiden käyttämät luokat ja asettaa renkaiden sijainnille poikkeamia. Näiden asetusten muutoksille ei kuitenkaan ollut kaivoskoneiden kohdalla tarvetta, vaan niissä käytettiin oletusarvoja.

Toiminta-animaatioiden toteutus BVS:llä

Kummankin kaivoskonetyypin Blueprint-kantaluokkaan toteutettiin BVS:llä omantyyppisiään toiminta-animaatioita kontrolloiva toimintalogiikka. Animaatioiden kontrollointi saatiin aikaan päivittämällä animaatioluokissa niveliä edustavien luukontrollereiden rotaatioarvoja. Kauhakuormaajakoneen tapauksessa LoaderVehicleBP-luokassa kontrolloitiin nostovarren sekä kauhan liikkeitä, kun taas kippiauton DumperVehicleBP-luokassa otettiin haltuun kippilavan nosto ja lasku. Näiden komponenttien liikelaajuuden kontrollointia varten luotiin kulmien raja-arvot määrittävät muuttujat.

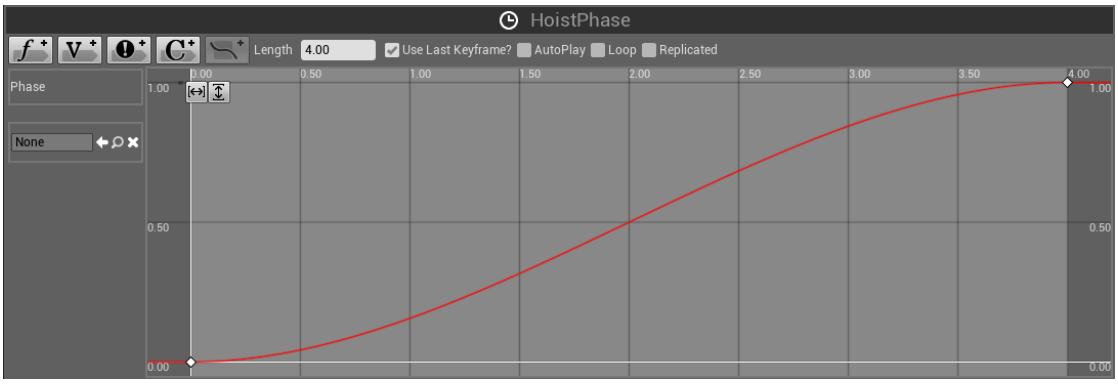
Eri toimintavaiheiden animaatioiden käynnistystä varten luokissa toteutettiin kaivoskoneiden AArticulatedVehicle-kantaluokassa esitellyt, toimintavaiheita vastaavat, natiivit tapahtumat (events). Kauhakuormaaja-tyyppisen kaivoskoneen luokkaan toteutettiin kaikki kolme tapahtumaa, jotka ovat StartTipping() (nostovarsi ylös), Per-

formTipping() (kauhan kaato) ja EndTipping() (paluu lähtöasentoon). Kippiauto-tyyppisessä kaivoskoneessa toteutettiin ainoastaan PerformTipping()- ja EndTipping()-tapahtumat, sillä kuorman kaato tapahtuu samalla kun lava nostetaan ylös.



Kuvio 27. LoaderVehicleBP-luokan tapahtumakaavio (Event Graph)

Kuviossa 27 nähdään LoaderVehicleBP-luokan tapahtumakaavio kokonaisuudessaan BVS:llä toteutettuna. Kun halutun toiminnan tapahtumaa kutsutaan, niin ensimmäisenä koneelle asetetaan aloitettua tapahtumaa vastaava toimintastatus. Tämän jälkeen suoritus siirtyy Timeline-solmuun, joka interpoloi annetussa ajassa liikkeen vaiheen arvosta 0 arvoon 1. Interpolointi suoritetaan kolmannen asteen polynomiyhtälöillä, jolloin liikkeestä saadaan luonnollisempaa vaiheistukseen syntyvän kiihtyvyyden ja hidastuvuuden ansiosta. Kuviossa 28 on esimerkki kauhakuormaaja-tyyppisen kaivoskoneen nostovarren liikevaiheen interpoloinnista Timeline-solmussa. Esimerkissä nostovarsi liikkuu ääriasennosta toiseen neljän sekunnin kuluessa.



Kuvio 28. Kauhakuormaaja-tyyppisen kaivoskoneen nostovarren liikevaiheen interpolointi Timeline-solmussa

Timeline-solmusta viedään vaihe lineaariseen interpolointi-solmuun, joka interpoloi vaiheen mukaisesti komponentin, kuten nostovarsi, kulmaa raja-arvojen väliltä. Tästä

kulmasta saadaan nyökkääminen (engl. pitch) Make Rot -solmuun, joka luo Rotator-komponentin. Kun se sijoitetaan mallin animaatio-instanssin luukontrollerille, asettuu luu Animation Blueprintissä kulman mukaiseen asentoon. Tapahtuman suoritus loppuu kun Timeline-solmu saapuu määrätyn ajan kuluttua päätökseensä. Mikäli suoritettava tapahtuma on toiminnon palautusanimaatio (EndTipping), niin kone saa sen lopussa lepostatuksen (idle).

Kauhakuormaaja-tyyppisessä kaivoskoneessa täytyi toteuttaa nostovarren ja kauhan liike siten, että ne toimisivat saumattomasti niin yhdessä kuin erikseenkin. Nostovarren nyökkäys vaikutti myös kauhan nyökkäykseen. Tästä johtuen kauhan nyökkäys piti tasapainottaa nostovarren käänteisellä nyökkäyksellä, jotta kauha pysyi noston aikana suorassa. Kun tähän tasapainotukseen lisättiin kauhan kaatonyökkäys, saatiin tilanteessa kuin tilanteessa toimiva nostovarren ja kauhan yhtäaikainen liike. Kippiautossa sen sijaan riitti yhden lavaa edustavan luun nyökkäys kaato-toiminnan aikaansaamiseksi. Koneiden toimintojen palautusanimaatiot saatiin toteutettua mutkattomasti suorittamalla Timeline-solmut käänteisesti lopusta alkuun.

7.2.6 Lopullisten Blueprint-luokkien luominen

Kun kaivoskoneiden kantaluokat olivat kunnossa niin C++ -puolella kuin Unreal Editorissakin (Blueprint-luokat), oli jäljellä enää lopullisten Blueprint-luokkien toteutus. Lopullisella tarkoitetaan tässä yhteydessä niitä luokkia, joista luodaan halutun kaivoskoneen instansseja maailmaan, ja joista ei peritä enää uusia luokkia.

Lopullisten Blueprint-luokkien kantaluokiksi valittiin DumperVehicleBP (kippiauto) ja LoaderVehicleBP (kauhakuormaaja). Uudet luokat nimettiin selkeyden vuoksi oikeiden koneiden mallimerkintöjen mukaisesti. Näin ollen kauhakuormaajan luokka sai nimekseen ”Toro0011_BP” ja kippiauton luokka ”Toro50_BP”.

Kaivoskoneluokkien komponenttilistalta valittiin Mesh (Skeletal Mesh Component), jolloin editorin Details-välilehdelle saatiin komponentin ominaisuudet muokattavaksi. Komponentin Skeletal Meshiksi valittiin kaivoskoneiden 3DS Maxista importoidut mallit (Toro0011_Skeletal_Mesh-kauhakuormaajamalli ja Toro50_Skeletal_Mesh-kippiautomalli). Näin kaivoskoneille saatiin määriteltyä ulkoasu. Koneiden animointi ja

toiminta olivat valmiiksi määriteltynä niiden Blueprint-kantaluokissa, joten niihin ei tarvinnut enää lopullisissa luokissa puuttua.

Vehicle Setup & Mechanical Setup

Koneiden VehicleMovement-komponentteihin tehtiin lopuksi vielä suuntaa antavia asetuksia, jotta koneet saatiin fyysiseltä käytökseltään lähemmäksi reaalimaailman vastineitansa. Koneiden massat asetettiin Vehicle Setupissa yli 55 tonniin. Lisäksi kaivoskoneiden moottoreille asetettiin Mechanical Setup-asetuksissa oikeanlaiset väntökäyrät ja kierroslukurajoitukset. Differentiaaliasetuksista asetettiin vielä neliveto päälle. Näiden asetusten jälkeen kaivoskoneiden liikkeet muuttuivat paljon jähmeämmäksi, mikä oli tässä tapauksessa toivottu ominaisuus.

8 POHDINTA

8.1 Lähtökohdat

Työn lähtökohtana oli toteuttaa Unreal Engine 4-pelimoottorilla toimiva lähtökohta kaivossimuloinnissa käytettäville nivelletyille kaivoskoneille. Koneiden piti pystyä seuraamaan animoidusti splinikurveista muodostuvaa reittiä ja taipua keskeltä kääntyessään reitin mutkissa. Tavoitteena oli tehdä kaivoskoneille tyyppikohtaiset toiminta-animaatiot, joita voidaan hyödyntää kaikkien samantyyppisten koneiden animointiin.

Unreal Enginen ohjelmistokehys oli kohtalaisen uusi tuttavuus työn tekijälle, joten kaivoskoneiden reitinseuranta ja animaatioiden ohjaus päädyttiin toteuttamaan Blueprint-luokkien Blueprint Visual Scripting-systeemillä. Tämä oli loistava tapa tutustua samalla Unrealin kirjastoihin sekä ohjelmistokehyksen toimintamalliin kehitystyön ohella.

8.2 Työn tulokset

Työn lopputuloksena saatiin tavoitteen mukaisesti kaksi Unreal Engine 4-ympäristössä toimivaa niveltyvää kaivoskonetta. Koneet seurasivat testiympäristössä splinimuodossa olevaa reittiä ja taipuivat liikkuessaan reitin kurveissa melko luonnollisen näköisesti. Lisäksi koneet suorittivat onnistuneesti komennosta tyyppikohtaisia toiminta-animaatiota, kuten kippaaminen.

Kaivoskoneiden arkkitehtuuri sisälsi todella paljon Unreal Engine 4:n valmiita komponentteja ja ne hitsautuivat saumattomasti omiin C++ -luokkiin niin perinnän kuin koostamisenkin kautta. Selkeä luokkahierarkia ja rakenne mahdollistavat yksityiskohtien sujuvan lisäämisen olemassa oleviin komponentteihin sekä uusien komponenttien lisäämisen järjestelmään mutkattomasti. Uuden kauhakuormaajan tai kippiauton lisääminen vaatii lähinnä uuden mallikohtaisen Blueprint-luokan luonnin ja 3D-mallin importoinnin Unreal Editoriin. Tämän jälkeen toimiva kone saadaan aikaiseksi asettamalla tyyppikohtaiset kantaluokat ja komponentit kohdalleen.

Uuden konetyypin lisääminen vaatii jatkossakin hieman enemmän työtä, sillä uusille luurankorakenteille täytyy muodostaa uudet toiminta-animaatiot. Tämäkään prosessi ei ole lopulta erityisen työläs, varsinkin jos siihen on muodostunut jonkinlainen rutiini.

8.3 Luokkahierarkia ja rakenne

Luokkahierarkia ja rakenne kokonaisuudessaan elivät yllättävän paljon työn aikana. Alun perin suunnitelmana oli tehdä jokaisesta C++ -luokkatasosta perimällä vastaavat Blueprint-luokat, joihin toimintalogiikkaa olisi toteutettu BVS:llä. Näin ollen MiningVehicleBase-luokasta olisi peritty MiningVehicleBaseBP-luokka ja ArticulatedMiningVehicle-luokasta vielä ArticulatedMiningVehicleBP-luokka. Näistä ensimmäiseen piti tulla reitinseurantalogiikka, kun taas jälkimmäinen olisi hoitanut koneiden taittumisen.

Ongelmaksi tällaisessa rakenteessa muodostui kuitenkin se tosiasia, että Blueprint-luokista puuttui C++ -luokista tuttu moniperintä. Tämä tarkoitti käytännössä sitä, että kun jollakin luokkatasolla hypättiin perinnässä C++ -luokasta Blueprint-puolelle, ei muiden C++ -luokkien ominaisuuksia voinut enää lisätä Blueprint-luokkiin kuin koostamalla. Perintäketju katkesi myös C++ -puolella ensimmäiseen Blueprint-luokkaan, sillä C++ -luokat eivät voineet enää periytyä Blueprint-luokista.

Perintähierarkiassa liian aikainen siirtyminen Blueprint-luokkiin olisi siirtänyt koko luokkahierarkian painotuksen pääasiassa Blueprint-puolelle. Joissakin tapauksissa tällainen rakenne voisi olla hyväksyttävä tai jopa toivottu ominaisuus, mutta kaivoskoneille haluttiin säilyttää kaivossimulointia silmällä pitäen kontrolli C++ -koodista käsin. Tämä kontrolli olisi menetetty tai se olisi ainakin vaikeutunut, mikäli perinnässä olisi siirrytty Blueprint-luokkiin liian varhaisessa vaiheessa. Tästä johtuen päädyttiin nykyisen kaltaiseen rakenteeseen, jossa Blueprint-luokkiin siirrytään perinnässä vasta kaivoskoneiden tyyppikohtaisista C++ -luokista. Jos tulevaisuuden tarpeet suljettaisiin kokonaan pois, olisi koko työn voinut teoriassa toteuttaa ilman riviäkään C++ -koodia.

Uudenlainen perintämalli johti siihen, että kaivoskoneiden reitinseuranta ja keskinivelten taivutus jäivät ikään kuin leijumaan ilmaan. Koska niiden toteutusta ei kan-

nattanut enää tehdä moninkertaistumisen takia konetyyppikohtaisiin Blueprint-kantaluokkiin, täytyi löytää vaihtoehtoinen tapa toteuttaa kaikkien kaivoskoneiden kanssa toimiva reitinseuranta sekä nivellettyjen kaivoskoneiden kanssa toimiva taivutus.

Aikaisempi kokemus ohjelmistokehityksestä johti onneksi pian oikeille raiteille; silloin kun ei voi moniperiä, täytyy koostaa. Tarvittiin komponentti, jonka voisi asettaa jokaiselle ajoneuvolle, ja tämä komponentti saisi aikaan ajoneuvojen liikkumisen reittiä pitkin. Komponenttia voisi laajentaa perinnällä siten, että peritty komponentti lisäisi reitin seuraamiseen koneiden taittumisen. Unreal Enginestä löytyi onneksi juuri tähän tarkoitukseen sopiva ActorComponent-kantaluokka. Sen tarkoitus oli alun perinkin lisätä uudelleen käytettävää käyttäytymistä Actor-tyyppisille objekteille, millaisia Unrealin ajoneuvotkin pohjimmiltaan olivat. Tämä johti SplineFollowerComponent-kantaluokan (C++), ja siitä perittyjen, toimintalogiikan sisältävien Blueprint-luokkien suunnitteluun ja toteutukseen.

Arkkitehtuuria ja rakennetta koskevaa tietoutta oli hyvin niukasti tarjolla Unreal Engine 4:n dokumentaatiossa, keskustelupalstoilla tai muillakaan verkkosivuilla. Tietoutta löytyi lähinnä yksittäisistä luokista ja komponenteista, välillä heikosti niistäkin. Lähes kaikista komponenteista löytyi kyllä lyhyt esittely, mutta dokumenteista ei löytynyt muutamaa poikkeusta lukuun ottamatta lainkaan tietoa siitä mihin, miten ja miksi niitä käytetään. Yleisesti ottaen kaikki löytynyt tieto tuntui pirstaleiselta ja usein sitä sai etsiä pala palalta monesta eri lähteestä. Tästä johtuen tuntui, että varsinkin rakenteellisen puolen kanssa oltiin pitkälti omillaan.

Työn lopullisesta rakenteesta ei varmastikaan tullut täydellinen. Unreal Engine 4:n parempi dokumentaatio sekä pitempiaikainen kokemus ja parempi tuntemus Unreal Engine 4:stä ja sen arkkitehtuurista olisivat todennäköisesti johtaneet erilaiseen lopputulokseen. Mielestäni lopputulos rakenteen suhteen oli kuitenkin varsin toimiva, ja se täytti hyvin tehtävänsä kaivoskoneiden toteutuksessa.

8.4 Työvaiheet

Itse työn vaiheistus oli erittäin selkeää, sillä työvaiheet etenivät aina alusta lähtien loogisesti tarpeesta toiseen. Aluksi tarvittiin malli, jota käyttää koneiden animoinnin ja toimintalogiikan toteutuksessa. Niinpä ensimmäinen vaihe oli saada välittömästi toinen kaivoskoneiden malleista 3DS Maxilla Unreal Engine 4:n tukemaan muotoon. Kauhakuormaajakone sai kunnian toimia pioneerina suuressa osassa kehitystyötä, ja kippiauton malli otettiin projektiin mukaan vasta siinä vaiheessa, kun muu toimintalogiikka alkoi olla jo valmista.

Seuraava tarve oli saada kaivoskoneiden mallit istumaan rakenteen ja fysiikan puolesta Unreal Enginen 4:n ajoneuvokomponentteihin. Tätä varten malleille tehtiin valmistelevia toimenpiteitä, ja mallit kävivät toimivuuden testaamiseksi läpi lyhyen testausprosessin Unreal Enginen geneerisistä ajoneuvokomponenteista luodun yksinkertaisen ajoneuvon avulla.

Tässä vaiheessa ongelmat koskivat lähinnä Unreal Enginen ja 3DS Maxin toisistaan poikkeavaa käsitystä mallien mittayksiköistä. Tuntui todella haastavalta saada vanhojen pohjamallien mittasuhteet Unreal Editorissa edes oikeaan suuntaan, vaikka mallinnusohjelman yksikköasetukset oli asetettu vastaamaan Unrealissa käytössä olevaa senttimetriyksikköä. Mallin geometria saattoi olla Unrealiin tuodessa oikean kokoinen, mutta viimeistään sen fysiikka saattoi mennä PhAT-työkalussa täysin sekaisin.

Lopulta mallien pohjalle päädyttiin ottamaan Unrealin esimerkkiprojektista tuotu ja todistetusti toimiva auton FBX-malli. Kaivoskoneiden mallit siirrettiin 3DS Maxissa sen ”päälle”, ja mittasuhteet osuivat sen myötä lopulta kohdalleen myös fysiikan suhteen PhAT-työkalussa. Perimmäinen syy fysiikan sekoamiseen jäi toistaiseksi epäselväksi, mutta kippiauton mallin kanssa onnistuttiin myöhemmin jo ensiyrittämällä.

Kun kaivoskoneet saatiin Unrealin ajoneuvokomponenttien kanssa yhteensopiviksi, tarvittiin reitit, joita voitaisiin asettaa kaivoskoneille seurattavaksi. Tulevaisuuden tarpeista tiedettiin sen verran, että Unreal Enginen SplineComponent oli se komponentti, joka tulisi määrittämään reittien muodon. Tätä varten toteutettiin yksinkertainen SplineComponentin sisältävä reittien kantaluokka. Tästä perittiin aluksi yksinker-

tainen RouteSplineBP-luokka ja myöhemmin vielä hieman pidemmälle menevä TunnelBP-luokka, jonka jokainen instanssi loi splini-reittinsä lisäksi automaattisesti vastaavan pätkän kaivostunnelia.

Reittien toteutuksen jälkeen tarvittiin vielä toimintalogiikka, joka saisi kaivoskoneet seuraamaan olemassa olevia reittejä. Logiikka rakennettiin alun perin kaivoskoneiden yhteiseen Blueprint-kantaluokkaan, mutta se siirrettiin myöhemmin rakennemuutosten vuoksi erilliseen SplineFollowerComponent-komponenttiin. Tämä osoittautui lopulta hyväksi ratkaisuksi. Unreal Enginen apufunktioiden (BVS-solmujen) avulla logiikan rakentaminen onnistui mutkattomasti, eikä toteutuksen kanssa ilmennyt ongelmia.

Mikäli suorituskyvyn kanssa ilmenisi myöhemmin optimointitarpeita, voidaan BVS-logiikka siirtää pienellä vaivalla myöhemmin C++ -koodiin. Vaikka kaivossimulointien reittikomponenttien pohjana ei lopulta käytettäisi työssä toteutettuja splini-reittiluokkia, ovat reitinseurantakomponentit adaptoitavissa pienillä muutoksilla mihin tahansa reittiluokkaan, jonka reittimuoto määritellään Unrealin SplineComponentilla.

Lopuksi tarvittiin vielä komponentit kaivoskoneiden toiminta-animaatioiden suorittamiseen. Koska jokaisella konetyypin edustajalla oli lähtökohtaisesti sama luurankohierarkia ja samat toiminnot, luotiin jokaiselle konetyypille oma Animation Blueprint. Tämän tehtävä oli muodostaa nivelien kääntöketjulla koneen lopullinen asento. Nivelien kääntöjä kontrolloitiin luukontrollereiden avulla konetyyppikohtaisista Blueprint-luokista käsin.

Ennen työn aloittamista oli hiukan epäselvää, mikä olisi oikea tapa lähteä toteuttamaan kaivoskoneiden animaatioita. Vaihtoehtona luurankomallille olisi ollut komponenttirakenne, jossa jokainen koneen osa olisi ollut oma staattinen mallinsa. Osien hierarkia olisi määritelty Unreal Editorissa, ja toiminta-animaatiot olisi toteutettu kääntämällä näitä malleja luiden sijaan. Vaikka tämäkin lähestymistapa olisi varmasti ajanut asiansa, olisi se vaatinut jokaisen kaivoskoneen kohdalla ylimääräisiä työvaiheita. Tämä taas olisi tehnyt prosessista virhealttiimpaa, sillä jokaisessa erillisessä työvaiheessa on mahdollisuus tehdä jotakin väärin. Etuna komponenttimallissa olisi ollut mahdollisuus vaihtaa helposti yksittäinen koneen osa toiseen. Lisäksi luuranko-

mallin skinning-tekniikan vaatima verteksien käsittely aiheuttaa jonkin verran kustannuksia suorituskykyyn komponenttimalliin verrattuna. Mahdollisuus yksittäisten osien vaihtamiseen oli kuitenkin kaivossimulointeja silmällä pitäen epäoleellinen ominaisuus. Selkeyden ja helppokäyttöisyyden nimessä animointi päätettiin lopulta toteuttaa luurankomallin avulla.

Itse animaatioiden toteutuksessa haasteena oli lähinnä nivelketjun rakentaminen luurankomallin mukaisesti siten, että nivelet saatiin kääntymään oikealla tavalla itseensä ja isäntäänsä nähden. Joillekin nivelille piti hakea yrityksen ja erehdyksen kautta oikeat kääntymissuunnat, -moodit ja -tilat. Väärässä moodissa tai tilassa kääntyvä nivel saattoi aiheuttaa muun muassa kauhakuormaajan nostovarren tai kauhan kallistumisen sivuille koneen taipuessa kurveihin ajettaessa. Onneksi oikean konfiguraation hakeminen oli nopea prosessi, sillä muutoksia pääsi testaamaan välittömästi.

8.5 Loppuanalyysi

Ajattelin työn alussa, että toteutuksessa olisi tarvinnut kirjoittaa huomattavasti enemmän C++ -koodia. Lopulta C++ -luokista tuli kuitenkin lähinnä rajapintoja Blueprint-luokkiin, joissa varsinainen toimintalogiikka rakentui yllättävän luontevasti Blueprint Visual Scripting-systeemin avulla. C++ -kantaluokkien ja rajapintojen olemassaolo oli kuitenkin ehdottoman tärkeää, sillä ilman niitä kaivoskoneiden toiminta-animaatioita ei voisi tulevaisuudessa helposti hallita C++ -koodista käsin.

Unreal Engine 4-sovelluskehys nopeutti kehitystyötä huomattavasti tarjoamalla valmiita komponentteja käytännössä jokaiseen työssä käytettyyn osa-alueeseen. Tavoite käyttää mahdollisimman paljon Unreal Engine 4:n komponentteja kehitystyössä toteutui lopulta ikään kuin itsestään. Myös viiveet muutosten tekemisen ja testaamisen välillä olivat lähes olemattomia perinteiseen C++ -koodin kääntämiseen verrattuna. Blueprint-koodiin tehdyt muutokset olivat käytännössä lähes välittömästi nähtävissä testiympäristössä, mikä nopeutti työtä entisestään.

Yksi ilkeäkin piirre Unreal Enginen parissa työskentelyssä oli, puutteellisen dokumentaation lisäksi. Ainakin työssä käytetyssä versiossa 4.7.3, ja sitä aiemmissakin versioissa, Unreal Editor kaatuili mielestäni luvattoman paljon. Kaatuilu ei useimmiten ol-

lut yhdistettävissä tiettyihin toimintoihin, vaan tuntui useimmiten satunnaiselta. Pahimmillaan kaatuminen sotki C++ -ja Blueprint-luokkien reflektointia, jos editori sattui kaatumaan niiden ollessa epärynkronisessa tilassa. Tällöin joutui yleensä käsityönä tutkimaan ja siivoamaan korruptoitunutta Blueprint-luokkaa, kunnes se saatiin taas yhteisymmärrykseen C++ -kantaluokan kanssa. Onneksi nämä tilanteet olivat kuitenkin harvassa, ja Unreal Enginen kanssa työskennellessä oppi myös lähteissä esille tulleen tärkeän rutiinin; tallenna ajoissa, tallenna usein!

Itse työn aihe oli todella mielenkiintoinen ja mukava toteuttaa. Sain käyttää toteutuksessa paljon sekä opinnoista että työstä aiemmin saamaani kokemusta. Oli myös kiva tiedostaa, että työstä on työnantajalleni oikeaa hyötyä ja saan toivottavasti olla mukana jatkokehittämässä kaivoskoneiden malleja ja toimintaa. Ainakin koneiden ajon aikaisessa taivuttamisessa on vielä parannettavaa ja siitä saisi todennäköisesti jo melko pienilläkin muutoksilla aidomman näköistä. Myös koneiden fysikaalista olemusta voisi tarkentaa lisäämällä törmäysvastetta erikseen koneiden molemmille osille. Tärkeä jatkokehityskohde olisi tuki uusille ajoneuvotyypeille, joissa on enemmän kuin 4 rengasta tai telaketju. Muun muassa Toro60-mallinen kippiauto on kuusirenkainen. Yksi ominaisuus, mitä voi myös aina kehittää, on sekä konemallien että kaivosympäristön ulkoasu.

Sain Unreal Enginen myötä ensimmäisen pintaraapaisun metaohjelmointiin siihen kehitetyn reflektoinnin kautta. Sen verran jäi kuitenkin vielä hampaan koloon, että Unreal Engine 4 tarjoaa vielä paljon lisää opittavaa. Muun muassa reitinseurantakomponentin referenssin ajoneuvon reitti-spliniin voisi yrittää häivyttää Blueprint Interfacen avulla, jolloin ajoneuvo toteuttaisi ISplineFollower-rajapinnan joka määrittelee reitti-splinin palauttavan GetRouteSpline()-metodin.

Kaiken kaikkiaan koko työstä ja sen toteutuksesta jäi todella hyvä maku suuhun. Sain paljon arvokasta tietoa Unreal Engine 4:n kanssa työskentelystä, ja erityisesti rakenteellisten rajoitusten kanssa tuskailu ja sen kautta vahvuuksien löytäminen myötävaikuttaa varmasti paljon muuhunkin ohjelmistokehitystyöhön tulevaisuudessa.

LÄHTEET

AActor. n.d. Unreal 4 Engine Documentation. Viitattu 21.4.2015. https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/GameFramework/AActor/index.html

Actors. n.d. Unreal 4 Engine Documentation. Viitattu 21.4.2015. https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Actors/index.html

Animation Blueprints. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Animation/AnimBlueprints/index.html

AnimGraph. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Animation/AnimBlueprints/AnimGraph/index.html

APEX. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Physics/Apex/index.html

AWheeledVehicle. n.d. Unreal Engine 4 Documentation. Viitattu 22.4.2015. https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/GameFramework/AWheeledVehicle/index.html

Blueprint Class. n.d. Unreal Engine 4 Documentation. Viitattu 22.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/ClassBlueprint/index.html

Blueprint Compiler Overview. n.d. Unreal Engine 4 Documentation. Viitattu 24.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/TechnicalGuide/Compiler/index.html

Blueprint FAQ and Tips. n.d. Unreal Engine Community Wiki. Viitattu 24.4.2015. https://wiki.unrealengine.com/Blueprint_FAQ_and_Tips

Blueprint Fundamentals. n.d. Unreal Engine Community Wiki. Viitattu 22.4.2015. https://wiki.unrealengine.com/Blueprint_Fundamentals

Blueprint Overview. n.d. Unreal Engine 4 Documentation. Viitattu 24.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/Overview/index.html

Blueprints User Guide: Variables. n.d. Unreal Engine 4 Documentation. Viitattu 22.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Variables/index.html

Blueprints Visual Scripting. n.d. Unreal 4 Engine Documentation. Viitattu 4.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html

Convert Spaces Nodes. n.d. Unreal Engine 4 Documentation. Viitattu 15.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Animation/NodeReference/SpaceConversion/index.html

EventGraph. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Animation/AnimBlueprints/EventGraph/index.html

FBX. 2015. Wikipedia – vapaa tietosanakirja. Viitattu 8.4.2015. http://en.wikipedia.org/wiki/FBX

FBX Import Errors. n.d. Unreal Engine 4 Documentation. Viitattu 4.4.2015. https://docs.unrealengine.com/latest/INT/Shared/Editor/FbxErrors/index.html

FBX Skeletal Mesh Pipeline. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Content/FBX/SkeletalMeshes/index.html

FREQUENTLY ASKED QUESTIONS (FAQ). n.d. Viitattu 25.4.2015. https://www.unrealengine.com/faq

Functions (Blueprints). n.d. Unreal Engine 4 Documentation. Viitattu 22.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Functions/index.html

Functions (C++). n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Reference/Functions/index.html

Materials. n.d. Unreal Engine 4 Documentation. Viitattu 26.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/index.html

Noland, M. 2014. Unreal Property System (Reflection). Viitattu 24.4.2015. https://www.unrealengine.com/blog/unreal-property-system-reflection

Objects. n.d. Unreal 4 Engine Documentation. Viitattu 21.4.2015. https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Objects/index.html

Pawn. n.d. Unreal 4 Engine Documentation. Viitattu 21.4.2015. https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Pawn/

Physics Simulaiton. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Physics/index.html

Rendering Overview. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Rendering/Overview/index.html

Skeletal animation. 2014. Wikipedia – vapaa tietosanakirja. Viitattu 20.4.2015. http://en.wikipedia.org/wiki/Skeletal_animation

Skeleton Assets. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Animation/Skeleton/index.html

Skeletal Controls. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Animation/NodeReference/SkeletalControls/index.html

Skeletal Meshes. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Content/Types/SkeletalMeshes/index.html

Static Meshes. n.d. Unreal Engine 4 Documentation. Viitattu 26.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Content/Types/StaticMeshes/index.html

Sweeney, T. 2015. IF YOU LOVE SOMETHING, SET IT FREE. Viitattu 25.4.2015. https://www.unrealengine.com/blog/ue4-is-free

Textures. n.d. Unreal Engine 4 Documentation. Viitattu 25.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Content/Types/Textures/index.html

Tietokonesimulointi. 2014. Wikipedia – vapaa tietosanakirja. Viitattu 20.4.2015. http://fi.wikipedia.org/wiki/Tietokonesimulointi

UActorComponent. n.d. Unreal Engine 4 Documentation. Viitattu 20.4.2015. https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/Components/UActorComponent/index.html

Unreal Architecture. n.d. Unreal Engine 4 Documentation. Viitattu 24.4.2015. https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Reference/index.html

Unreal Engine. 2015. Wikipedia – vapaa tietosanakirja. Viitattu 3.4.2015. http://en.wikipedia.org/wiki/Unreal_Engine

Vehicle Art Setup. n.d. Unreal Engine 4 Documentation. Viitattu 16.4.2015. https://docs.unrealengine.com/latest/INT/Engine/Physics/Vehicles/VehcileContentCreation/index.html