Pekka Virkkunen

Ajoneuvon toteutus Unreal Development Kit -pelimoottorille

Metropolia Ammattikorkeakoulu

Medianomi

Viestintä

Opinnäytetyö

25.5.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Pekka Virkkunen Ajoneuvon toteutus Unreal Development Kit -pelimoottorille 37 sivua + 2 liite 25.5.2013
Tutkinto	Medianomi
Koulutusohjelma	Viestintä
Suuntautumisvaihtoehto	3D-visualisointi ja animaatio
Ohjaaja(t)	Lehtori Kristian Simolin

Tässä toiminnallisessa opinnäytetyössä toteutettiin pelattava ajoneuvo Unreal Development Kit –pelimoottorille. Tavoitteena oli tutkia, mitä työvaiheita ajoneuvon luomisessa reaaliaikaiselle pelimoottorille tarvitaan ja mitä niissä tulee ottaa huomioon. Opinnäytetyön käytönnönosuuden tulos on nelirenkainen ajoneuvo, jonka katolla on ase.

Opinnäytetyössä käytetyt ohjelmat olivat 3ds Max 2011 mallinnukseen, Adobe Photoshop CS5 tekstuurien luontiin ja muokkaukseen ja Unreal Development Kit (versio 2012-05) pelattavan ajoneuvon toteuttamiseen.

Opinnäytetyössä keskitytään enemmän Unreal Development Kitissä tehtäviin työvaiheisiin, joista oli puutteellisesti löydettävissä tietoa pelimoottorin kehittäjän Epic Gamesin sivustoilla. Työvaiheisiin kuuluu tiedostojen tuonti UDK:hon, materiaalien ja fysiikka- ja animaatiotiedostojen luonti sekä kooditiedostojen muokkaus valmiista koodeista. 3ds Maxin työvaiheista käydään läpi lähinnä luiden asettelu ja skinnaus, vahinkomallin luonti morph targettia käyttäen ja mallin exporttaus actorX-plugilla. Tarkoituksena oli myös koota tästä työstä opas pelattavan ajoneuvon luonnista kiinnostuneille.

Saavutin työn alkuvaiheessa asettamani tavoitteet, vaikkakin kirjoitusvaihe vei odotettua enemmän aikaa. Lopputuotoksesta löytyy liitteenä video, jossa tulee esille ajoneuvon eri ominaisuuksia, kuten ajoneuvon aseella ampuminen ja ajoneuvon tuhoutuminen. Lisäksi liittenä on ajoneuvon UPK-tiedosto, joka sisältää kaikki UDK:ssa luodut ajoneuvon tiedotot sekä tarvittavat kooditiedostot ajoneuvolle.

Avainsanat	3D, Low poly, ajoneuvo, UDK, Unreal Development Kit

Author(s) Title Number of Pages Date	Pekka Virkkunen Vehicle Creation for Unreal Development Kit 37 pages + 2 appendices 25 May 2013
Degree	Bachelor of Arts
Degree Programme	Media
Specialisation option	3D Visualization and Animation
Instructor(s)	Kristian Simolin, Lecturer

This thesis includes creating a playable vehicle for Unreal Development Kit game engine. The goal was to study the process of creating a vehicle for real-time game engine and everything that needs to be taken into account. A product of the thesis was a four-wheeled vehicle with a weapon mounted on the roof. The software used for this thesis includes 3ds Max 2011 for modeling and rigging, Adobe Photoshop CS5 for texture creation and editing and Unreal Development Kit (version 2012-05) for combining the elements to a working real-time model.

The thesis focuses on the tasks related to Unreal Development Kit, which were inadequately documented on the website of the game engine's developer, Epic Games. These tasks include the following aspects: importing files to Unreal Development Kit, creating materials, physics and animation related files and modifying existing codes for your vehicle. In 3ds Max, rigging and skinning the vehicle are focused on as well as creating a damage model using morph targets and exporting the 3D assets using an actor-X plugin. The aim was also to create a guide for anyone interested in creating a playable vehicle.

I met the set goals for this project in the beginning, although the writing process took longer than I expected. The end result can be found in the attached video, which showcases some of the features of the vehicle, such as driving, shooting with the vehicle turret and destruction of the vehicle. I also attached the UPK file that includes all my vehicle files from Unreal Development Kit and the UnrealScript files that are needed for the vehicle.

Keywords	3D, Low poly, vehicle, UDK, Unreal Development Kit

Sisällys

1 Johdanto

Pelit ovat olleet mielenkiinnon kohteeni pienestä lähtien, ja myös yksi suurimmista syistä, miksi yleensäkin hain 3d-alalle, joten opinnäytetyön tekeminen peleihin liittyen tuntui luonnolliselta. Aiheeksi ajoneuvon luonti peliin sopi useammastakin syystä. Ensinnäkin en ollut aiemmin itse tehnyt ajoneuvoa, joten nyt oli hyvä tarttua tähän aiheeseen ja sen tuomiin haasteisiin. Toiseksi aiheesta oli heikosti löydettävissä tietoa varsinkin Unreal Development Kitin osalta verrattuna esimerkiksi hahmomallinnukseen. Lisäksi ajoneuvon luonnista pelikäyttöön ei ollut aiemmin tehty opinnäytetöitä, toisin kuin esimerkiksi pelihahmomallinnuksesta, jota itsekin aiemmin kokeilin. Yksi tavoitteistani on samalla testata, kuinka uuden asian tekeminen onnistuu vain verkkolähteitä ja omaa ongelmanratkaisukykyä hyväksi käyttäen.

Opinnäytetyöni tarkoitus on antaa lukijalle kaikki tarvittava tieto pelattavan ajoneuvon luontiin Unreal Development Kitiin käyttäen 3ds Maxia mallinnustyökaluna. Oletuksena on kuitenkin 3ds Maxin hallinta, eli en aio käydä yksityiskohtaisesti läpi mallinnusta ja Maxin työkaluja. En myöskään itse osaa UnrealScript-koodikieltä, joten sitä en osaa opettaa, eli käytössä ovat UDK:n mukana tulevat valmiit koodaukset, joita muokataan omaan ajoneuvoon sopiviksi. Tästä syystä näiden ohjeiden avulla voi luoda ajoneuvon, jossa on korkeintaan neljä pyörää, koska useamman tekeminen vaatii koodausta (Epic Games, Inc. 2010). Itse ajettavan ajoneuvon lisäksi opinnäytetyöstä löytyy tietoa teksturoinnista, toimivan ajoneuvoaseen luonnista, ajoneuvon vahingoittumiseen tarvittavista malleista sekä jonkin verran koodin muokkaamiseen liittyvistä asioista, kuten ajoneuvon ohjaamisen ja nopeuden säädöistä.

Opinnäytetyö on jaettu osiin työvaiheiden mukaan. Toisessa luvussa käydään läpi työtä edeltävät vaiheet sekä kaikki 3ds Maxissa tarvittavat toimenpiteet, jotka tarvitaan mallin viemiseen UDK:n ympäristöön ajoneuvoksi. Mallinnusvaiheesta käydään lähinnä läpi teksturointiin ja riggaukseen liittyviä seikkoja sekä exporttaustoimet. Kolmannessa luvussa käydään läpi ne vaiheet, joita tarvitaan UDK:ssa lopullisen toimivan ajoveuvon aikaansaamiseksi. Importtauksen lisäksi käsitellään materiaalien luontia, animaatioita ja koodin muokkausta ajoneuvon käyttöön.

Opinnäytetyön käytännön osuuden tuotoksena syntyy Unreal-pelimoottorissa toimiva ajettava 3d-ajoneuvo.

2 Käsitteiden määrittely

Hi-poly ja low-poly tarkoittavat mallinnuksessa eri versioita 3d-mallista, joista hi-poly-versiossa voi teoriassa olla niin paljon polygoneja kuin on vain tarpeen yksityiskohtia varten, kun taas low-poly-versio on optimoitu peliympäristöön, jolloin käytetään minimimäärän geometriaa muotojen luomiseksi. Usein hi-poly-mallia käytetään normaalikartan luomiseen, jolloin yksityiskohdat saadaan siirretty low-poly-versioon tekstuurina.

Normal map eli normaalikartta on RGB-kuva, jossa eri värikanavat vastaavat pinnan normaalien X-, Y- ja Z-koordinaatteja (Hajioannou, 2013). Normaalikartoitustekniikkaa (normal mapping) käytetään yksityiskohtien näennäiseen lisäämiseen yksinkertaiselle pinnalle valon avulla. Tekniikka on yleisesti käytössä peleissä, joissa se mahdollistaa yksityiskohtaisemmat mallit vähäisellä geometrialla säästäen muistia.

Material ID:tä käytetään 3ds Maxissa useamman materiaalin asettamiseksi yhdelle objektille. Material ID löytyy Editable Poly –muokkaimen alta, jossa valituille polygoneille voidaan asettaa oma ID-numero, jota vastaavasti voidaan käyttää halutussa materiaalissa. UDK osaa myös tunnistaa asetetut ID:t, jolloin on mahdollista käyttää useampaa materiaalia.

MIP mapping (MIP tulee latinan sanoista *multum in parvo,* mikä tarkoittaa paljon vähässä) on yksi antialisoinnin muoto, jota käytetään reaaliaikaisessa 3d:ssä. Se ehkäisee yksityiskohtaisen tekstuurin sahalaitaistumista kaukaa katsottuna muuttamalla tekstuurin kokoa pienemmäksi. Pelimoottorit luo tarvittavat MIP-mapit automaattisesti puolittamalla tekstuurin leveyden ja korkeuden ja käyttämällä pikselien värien keskiarvoja pienemmän kartan luomiseen. Esimerkiksi 128x128-kokoisesta kartasta siis luodaan 64x64-, 32x32-, 16x16-, 8x8-, 4x4-, 2x2- ja 1x1-kokoiset versiot. (Skinner, 2000.)

Morpher (muodonmuuttaja) on 3ds Maxissa muokkain, jota käytetään normaalisti animaatiossa esimerkiksi kasvoilmeiden luomiseen. Eri ilmeet luodaan morph targeteilla, 3d-mallin muokatuilla versioilla, joista tallennetaan verteksien sijainnit. Morpher-muokkaimella voidaan interpoloida eri ilmeiden välillä. Tätä tekniikkaa käytetään ajoneuvossa eri vahinkomallien luontiin.

Polygoni on kaksiulotteinen suljettu muoto, esimerkiksi kolmio tai neliö, joka muodostuu useista sivuista (edge) yhdistettynä vertekseihin. Verteksillä tarkoitetaan sivujen leikkauspisteitä. Polygonimalleissa sadat tai tuhannet polygonit muodostavat yhdessä 3d-mallin muodon. Hyvä 3d-malli koostuu neliöistä, joita on helpompi muokata. Peleissä käytetään neliöiden sijaan kolmioita, joita pyritään käyttämään mahdollisimman vähän halutun muodon aikaansaamiseksi. (Slick, 2013.)

Skinnauksella tarkoitetaan työvaihetta, jossa 3d-malli yhdistetään luihin, jolloin luita liikuttamalla voidaan animoida itse mallia. 3ds Maxissa käytetään skin-muokkainta, jossa yksittäiset verteksit painotetaan liikkumaan yhden tai useamman luun mukana.

UDK eli Unreal Development Kit on Epic Gamesin ilmainen versio Unreal Engine 3 – pelimoottorista. Ohjelmaa saa käyttää opetuksellisiin ja ei-kaupallisiin tarkoituksiin maksutta, mutta kaupallinen toiminta vaatii lisensoinnin.

UV-mappauksella tarkoitetaan työvaihetta, jossa 3d-mallin pinnalle projisoidaan 2d-tekstuuri. Monimutkaisissa malleissa onnistuneeseen UV-mappaukseen tarvitaan 3d-mallin niin sanottu unwrappaus, jossa pinta avataan ja litistetään 2d-tasoon yhdessä tai useammassa osassa.

3 Ideointi ja ajoneuvon luonti

3.1 Ennen mallinnusta

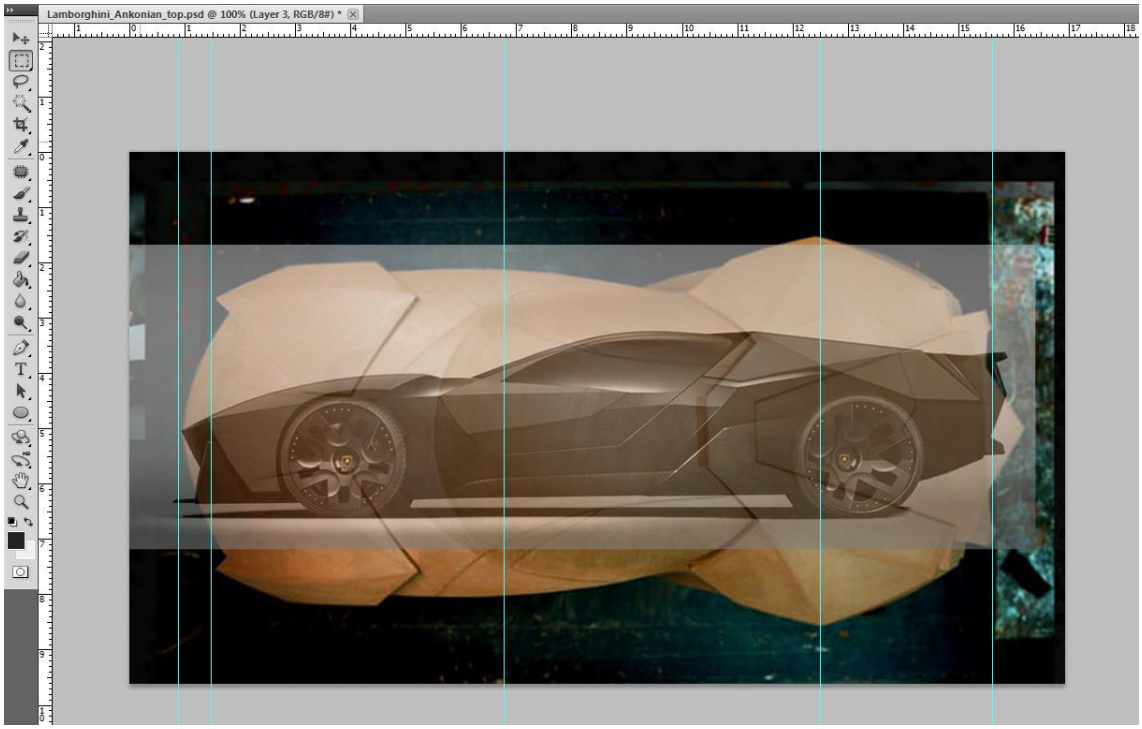
Ennen varsinaista ajoneuvon mallintamista tein pikaisen testiajoneuvon, jolla halusin saada selville, osaanko yleensäkään kaikkia tarvittavia työvaiheita. Samalla tietysti tuli joitain ongelmia, joiden ratkaisu helpotti varsinaisen työn tekemistä. Nämä ongelmat sisällytän myöhemmin niille kuuluviin vaiheisiin työssäni.

Hyvän mallinnuksen pohjalla ovat aina hyvät referenssikuvat. Jos on selvä näkemys ajoneuvosta ja visuaalista lahjakkuutta, voi hahmotella ja piirtää omat referenssikuvat, mutta nykyään on myös helppo löytää minkä tahansa tyyppistä materiaalia referenssiksi netistä. Mallintamisen kannalta on tärkeää löytää ainakin sivuprofiilikuva, etukuva, kuva takaa sekä mieluusti myös ylhäältä. Itselläni oli aluksi mielessä jokin nelipyöräinen panssaroitu ajoneuvo, josta ei kuitenkaan löytynyt sitä, mitä aluksi ajattelin. Jossakin vaiheessa päädyin sivulle, josta löytyi erilaisia konseptiajoneuvoja, joista lopulta valikoitui nuoren autosuunnittelija Slavche Tanevskyn vuonna 2008 suunnittelema Lamborghini Ankonian (kuva 1), joka vakuutti hävittäjämäisellä kulmikkuudellaan ja sporttisuudellaan.

Kuva 1. Slavche Tanevskyn suunnittelema Lamborghini Ankonian (2011).

Ankonianista ei löytynyt varsinaisia pohjapiirustuksia, jotka olisivat helpottaneet mallintamista huomattavasti, mutta onneksi sentään löytyi kuva suoraan sivulta sekä yksi kuva suoraan ylhäältä, joka oli otettu suunnitelman pohjalta tehdystä pienoismallista. Suoraan edestä tai takaa olevaa kuvaa ei löytynyt, joten näillä piti tulla toimeen. Eri kuvakulmista löytyi kuitenkin kuvia, jotka eivät käyneet suoranaisesti referenssikuviksi, mutta auttoivat yksityiskohtien ja pinnanmuotojen hahmottamisessa.

Referenssikuvat tulee ensi alkuun muokata kuvankäsittelyohjelmalla siten, että ajoneuvo on samankokoinen eri kuvissa. Lisäksi kannattaa katsoa, että jotkin avainkohdat, kuten esimerkiksi ovenkahva ja auton alku- ja loppupää ovat kohdikkain, jottei ongelmia ilmene kuvien kohdistamisessa 3ds Maxissa.

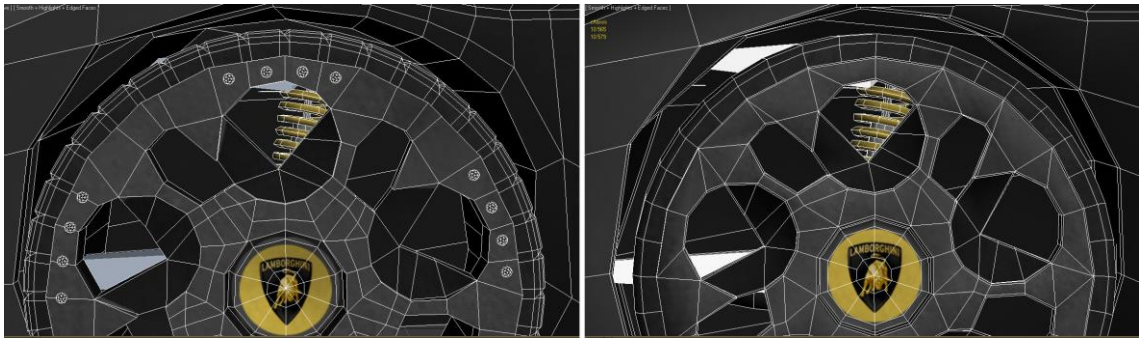


Kuva 2. Referenssikuvien kohdistaminen Photoshopissa viivaintyökalun avulla.

Pieniä ongelmia tässä tapauksessa aiheutti se, että kuvien mallit eivät olleet täysin samanlaisia johtuen siitä, että pienoismalli ei varmasti ollut täysin samoilla mitoilla tehty kuin konseptikuva. Ero ei kuitenkaan ollut liian suuri, joten kuvia pystyi käyttämään mallinnuksessa, kunhan otti poikkeavuudet huomioon. Muutenkin mallintaessa peliä varten ei ainakaan tässä tapauksessa ollut turhan tarkkaa, oliko lopullinen malli täysin suunnitelmien mukainen, vaikka niitä pyrin parhaani mukaan noudattamaan.

3.2 Huomioita mallinnuksesta ja teksturoinnista

En mene yksityiskohtiin, mitä auton mallintamiseen ja teksturointiin tulee, mutta joitain asioita pitää muistaa nimenomaan pelimallinnusta varten. Nykyään koneiden teho riittää pyörittämään polygonimäärältään, tai pelien kannalta olennaisemmin triangeli- eli kolmiomäärältään suuria malleja, mutta optimointi kannattaa aina. Pienimmät yksityiskohdat on paras tehdä normalmapeilla. Esimerkiksi UDK:n mukana tulevissa ajoneuvoissa on kuudesta seitsemääntuhanteen kolmiota. Toisaalta jos tulevana alustana on mobiili, rajoittaa se jo polygonimäärää merkittävästi. Ajoneuvoni lopulliseksi kolmiomääräksi tuli noin 20 000. Alunperin tein liikaa turhia yksityiskohtia geometrialla, jolloin kolmiomäärä ilman sisätilaakin oli lähempänä 50 000. Kuvassa 3 on esimerkki optimointityöstä renkaasta, jossa geometriaa on vähennetty itse muodon muuttumatta merkittävästi. Loput hävinneet yksityiskohdat, kuten pultit, on lisätty normaalikarttaan. Geometrian vähentäminen onnistui kohtuullisen helposti, mutta aiheutti myös runsaasti lisätyötä, kun sitä teki useampaan kertaan. Kannattaa siis alusta tarkkaan miettiä, mitkä yksityiskohdat tekee geometrialla ja mitkä tekstuureilla. Toki ajan riittäessä voi tehdä aluksi ajoneuvosta niin sanotun hi-poly-mallin, jota käyttää normaalikarttojen renderöintiin low-poly-versioon.

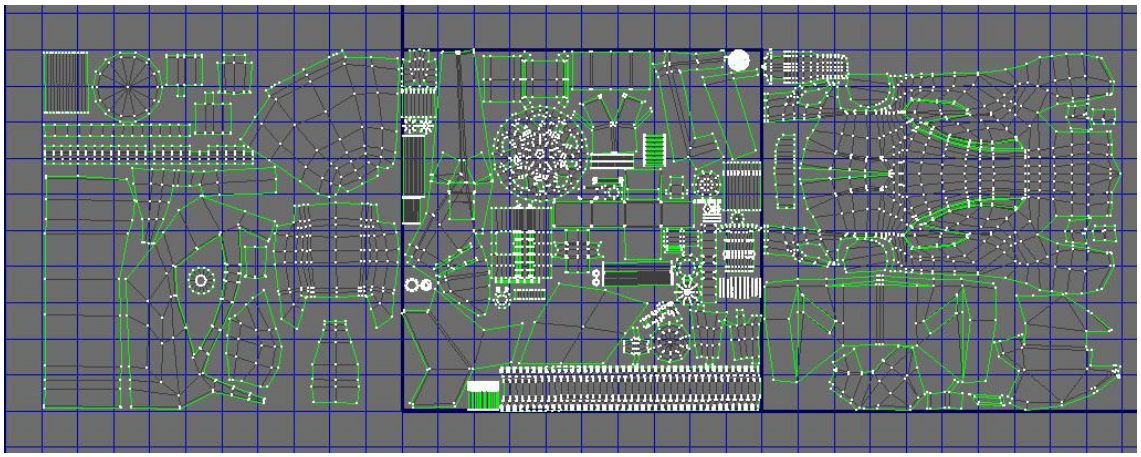


Kuva 3. Optimoidun renkaan ero alkuperäiseen. Yhdessä renkaassa kolmioiden määrä tippui 7440:stä 1640:een.

Hyvin yksinkertaisen mallin tekemiseen ei oikeastaan tarvita muuta kuin ajoneuvon runko, renkaat, jousitukset ja mahdollinen ase. Tein jälkikäteen lisäksi vielä yksinkertaisen sisätilan, joka näkyy ikkunoiden läpi. Sisätilan tarpeellisuus kannattaa miettiä ajoissa työn helpottamiseksi, vaikka sen jälkeen päin lisääminenkään ei ole mahdottoman vaikeata. Lisäksi kannattaa päättää, haluaako esimerkiksi konepellin tai takakontin aukenevaksi esimerkiksi ajoneuvon vahingoittuessa, jolloin niidenkin sisätilat tulisi mallintaa. Renkaita sijoittaessa tulee jättää tilaa riittävästi renkaan ja

ajoneuvon rungon väliin, ikään kuin jousitus olisi toisessa ääriasennossaan. Pelimoottorissa renkaat painuvat lähemmäksi, kun pelimoottori ottaa huomioon ajoneuvon painon vaikutuksen jousitukseen. (Epic Games, Inc. 2013.) Jos tilaa ei ole riittävästi, on mahdollista, että rengas menee osittain rungon läpi pelissä.

Teksturoinnissa kannattaa ottaa huomioon, että UDK tukee ajoneuvoissa korkeintaan 4096x4096-resoluutioista tekstuuria (Epic Games, Inc. 2013). Jotta tekstuurikoon automaattinen pienennys eli mipmappays toimii, on resoluution oltava kahden potenssissa, eli esimerkiksi 512x512 tai 1024x512. Useamman tekstuurin käyttö yhdessä ajoneuvossa onnistuu Material ID:itä hyväksi käyttäen. Tein omaani alunperin kaksi eri 2048x2048-kokoista tekstuuria, joista toisessa oli runko ja toisessa kaikki loput osat, jolloin lopulliseen malliin sai enemmän yksityiskohtia. Myöhemmin tein vielä sisätilalle oman UVW:n, koska olin jo tehnyt tekstuurit ja uvw:n muokkaus olisi tuottanut lisää työtä. Huono puoli useamman tekstuurin käytössä on se, että jokainen lisätekstuuri vie pelissä enemmän muistia, joten paras olisi tietysti mahduttaa kaikki tekstuurit yhdelle kartalle ja painottaa yksityiskohtia suurentamalla niiden osien UVW-mappeja, joihin tarvitsee tarkemman tekstuurin. Jos kuitenkin päätyy käyttämään useampaa karttaa, kannattaa ne sijoitella vierekkäin Unwrap UVW –muokkaimessa (Kuva 4) selkeyden vuoksi. Neliön sisään kartta on laitettava vain silloin, jos haluaa renderöidä Maxista ulos esimerkiksi normal mapin tai alpha mapin teksturointia varten. Lisäksi tietysti eri karttojen pitää olla omissa material ID:eissään, eli esimerkiksi rungon UV:t, kuvassa oikealla, asetettuna ID ykköseen, pienemmät osat ID kakkoseen ja sisäosa kolmoseen. Tämän avulla UDK:ssa voi jokaiselle asettaa erillisen materiaalin.

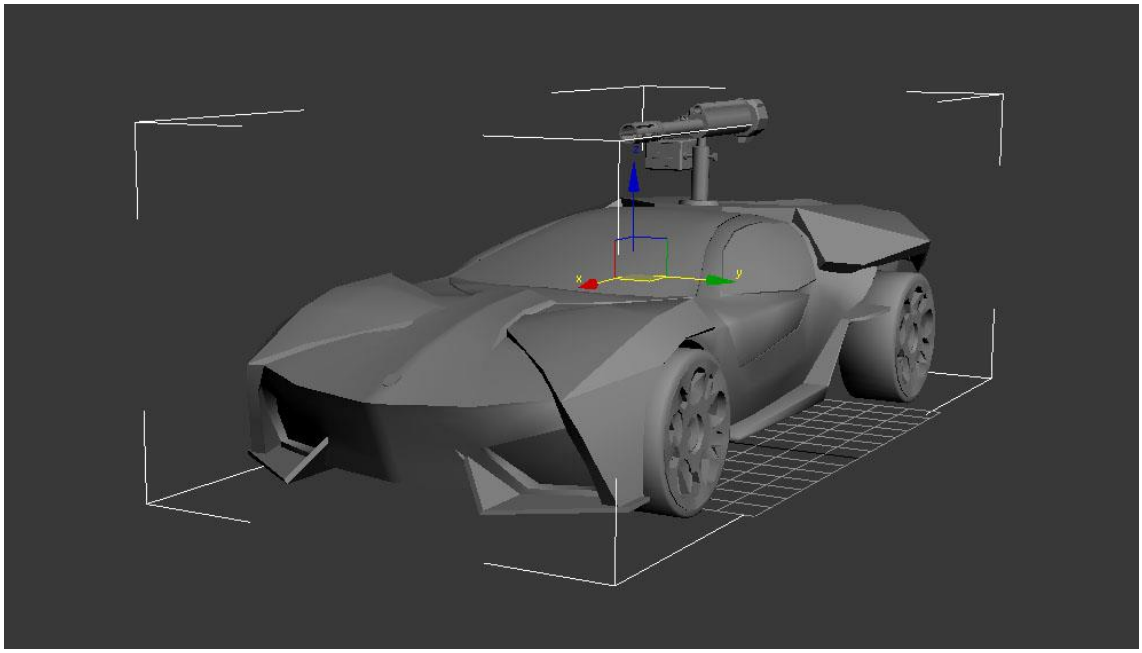


Kuva 4. Sisätilojen, osien ja ulkopuolen UVW-kartat sijoitettuna vierekkäin Unwrap UVW-muokkaimessa.

3.3 Luut ja skinnaus

UDK:ssa on mahdollista animoida malleja kokonaisina kappaleina, jolloin 3d-mallista riittää niin sanottu static mesh eli staattinen malli. Jos kuitenkin halutaan, että kappaleen eri osia voidaan animoida, esimerkiksi ajoneuvossa rengaiden pyöriminen ja kääntyminen, tulee 3d-malliin lisätä luusto liikkuvia osia varten. Tällaista luullista 3d-mallia kutsutaan UDK:ssa skeletal meshiksi eli luurankomalliksi. Ajoneuvoon tehdään siis luusto, joka on kuitenkin paljon yksinkertaisempi kuin esimerkiksi pelattavalla hahmolla.

Mallinnuksen ja uvw-mappauksien valmistuttua aletaan valmistella mallia UDK:hon viemistä varten. Tämä tarkoittaa käytännössä luiden tekoa ja skinnausta. Jos ajoneuvo on valmistettu useasta kappaleesta, kannattaa ne viimeistään tässä vaiheessa yhdistää yhdeksi kokonaiseksi kappaleeksi. Mallin tulee olla origossa siten, että renkaiden alasosa on xy-tasossa ja keula osoittaa positiviseen x-suuntaan (kuva 5). Koko riippuu tietysti käytetyistä yksiköistä. Omassani olivat käytössä geneeriset yksiköt, yksikön koon ollessa yksi senttimetri. Tällöin ajoneuvon pituudeksi tuli noin 260 yksikköä eli senttimetriä. Alunperin viedessäni ajoneuvon UDK:hon se oli liian suuri hahmon kokoon nähden, mutta yksinkertaisesti skaalaamalla Maxissa ja eksporttaamalla uudelleen tämän sai korjattua.



Kuva 5. Malli asetettuna origoon skinnausta varten.

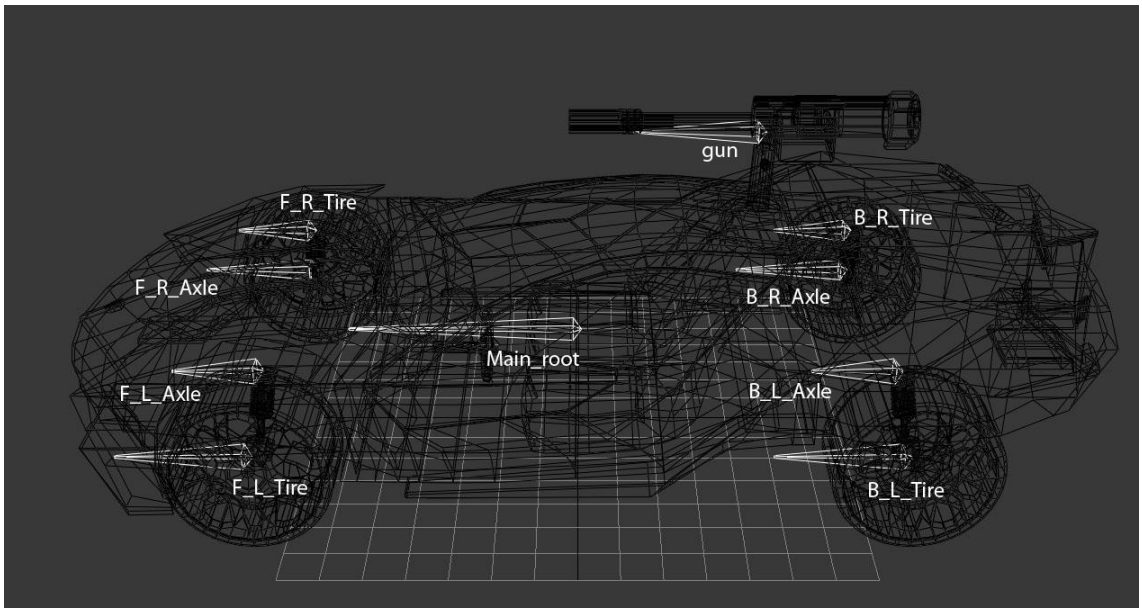
Ajoneuvoa varten tarvitaan ainakin yhdeksän eri luuta. Ajoneuvossa tulee olla pääluu, johon auton runko skinnataan. Kaikki muut luut linkitetään tämän luun lapsiksi (child). Jokaiselle renkaalle tulevat omat luunsa, kuten myös renkaat runkoon kiinnittäville akseille. Akselin luu ei saa olla renkaan lapsi, vaan sekin liitetään suoraan runkoon (Epic Games, Inc. 2013). Lisäksi erillisiä luita käytetään kaikkiin niihin osiin, joihin halutaan jotain animaatiota UDK:ssa. Esimerkiksi ajoneuvossani aseelle on luu, jotta se voi osoittaa siihen suuntaan, johon ammutaan. Lisäksi jos haluaa esimerkiksi oven avautuvan, pitää sille luoda oma luunsa. Luita käytetään myös auton vahingoittuessa osien irtoamiseen, jolloin irtoavalla osalla pitää olla oma luunsa. Jos haluaa ajoneuvoon enemmän kuin yhden istumapaikan, pitää jokaiselle tehdä oma luu. Omassa ajoneuvossani en tullut ajoissa ajatelleeksi tarvetta useammalle, joten ajoneuvosta tuli yksipaikkainen. Lisäksi halutessaan voi jo Maxissa määrittää tarkat paikat ajokameralle ja eri efekteille, kuten aseen suuliekille sijoittamalla luut näiden paikoille, mutta tämä ei ole pakollista, sillä paikat pystyy määrittämään myös UDK:ssa. (Epic Games, Inc. 2013.)

Luita ei voi sijoittaa miten sattuu, vaan niiden sijainnilla ja suunnalla on merkitystä. Pääluu pitää laittaa siten, että se on samansuuntaisia ajoneuvon kanssa, eli positiivisen x- ja z-akselin mukaan. Pääluu määrittää myös ajoneuvon massakeskipisteen, eli se kannattaa sijoittaa suurin piirtein keskelle ajoneuvoa pituussuunnassa ja vähän alas korkeussuunnassa, jolloin auto on tasapainossa. Testiautossani (kuva 6), joka oli melko lyhyt, oli ongelmana auton keuliminen heti, kun pelin aloitti. Tämä johtui pääluun sijainnista, joka teki ajokista takapainoisen. Luun siirtäminen eteenpäin korjasi ongelman. Renkaiden luut sijoitetaan siten, että yksi akseli osoittaa pyörimissuuntaan ja toinen kääntymissuunnan mukaisesti. Sillä ei ole merkitystä, käyttääkö x-, y-, vai z-akselia, koska tämä määritellään myöhemmin UDK:ssa, kunhan akselit osoittavat positiiviseen suuntaan. Itselläni oli aluksi ongelma, jossa jousitusosat pyörähtivät UDK:ssa ympäri. Tämä johtui siitä, että y-akseli osoitti alaspäin, koska olin luonut luun väärässä ikkunassa ja jättänyt tarkistamatta luiden orientaatiot. Eli näennäisesti kaikki oli kunnossa, mutta kun luun valitsi, näki rotaation olevan väärä. Tällaisten ongelmien välttämiseksi kannattaa tarkistaa luut ainakin ennen skinnausta.

Kuva 6. Testiauton keuliminen, kun pääluu oli sijoitettu väärin, jolloin massakeskipiste oli liian takana.

Rengasluut tulee sijoittaa siten, että niiden pivotti on keskellä rengasta, jolloin renkaan pyöriessä liike on tasaista. Kääntyvissä renkaissa, siis yleensä eturenkaissa, y-suuntaisella asettelulla määrätään, minkä pisteen suhteen renkaat kääntyvät ajoneuvoa kääntäessä. Akselien luut sijoitetaan siten, että niiden pivotti on siinä kohdassa, jossa akseli tai jousitus kiinnittyy auton runkoon. Jos ajoneuvossa on ase, jonka on tarkoitus pyöriä tähtäyssuunnan mukaan, sijoitetaan sen pivotti siihen kohtaan, josta ase liikkuu.

Luut kannattaa nimetä loogisesti, jotta pelkän nimen nähdessä tietää, mistä luusta on kyse. Paras tapa on sellainen, jolla nimet on helppo muistaa ulkoa. Ajoneuvossani käytin samaa käytäntöä kuin UDK:n ajoneuvoissa, eli pääluu on Main_root ja akselit ja renkaat on nimetty siten, että esimerkiksi oikea eturengas on F_R_Tire, jossa F ja R ovat englannista etukirjain sanoille ”front” ja ”right”, jolloin nimen nähdessä tietää heti, mikä luu on kyseessä. Kuvasta 7 näkee kaikkien luiden sijainnit ja nimet.



Kuva 7. Ajoneuvon luut nimineen

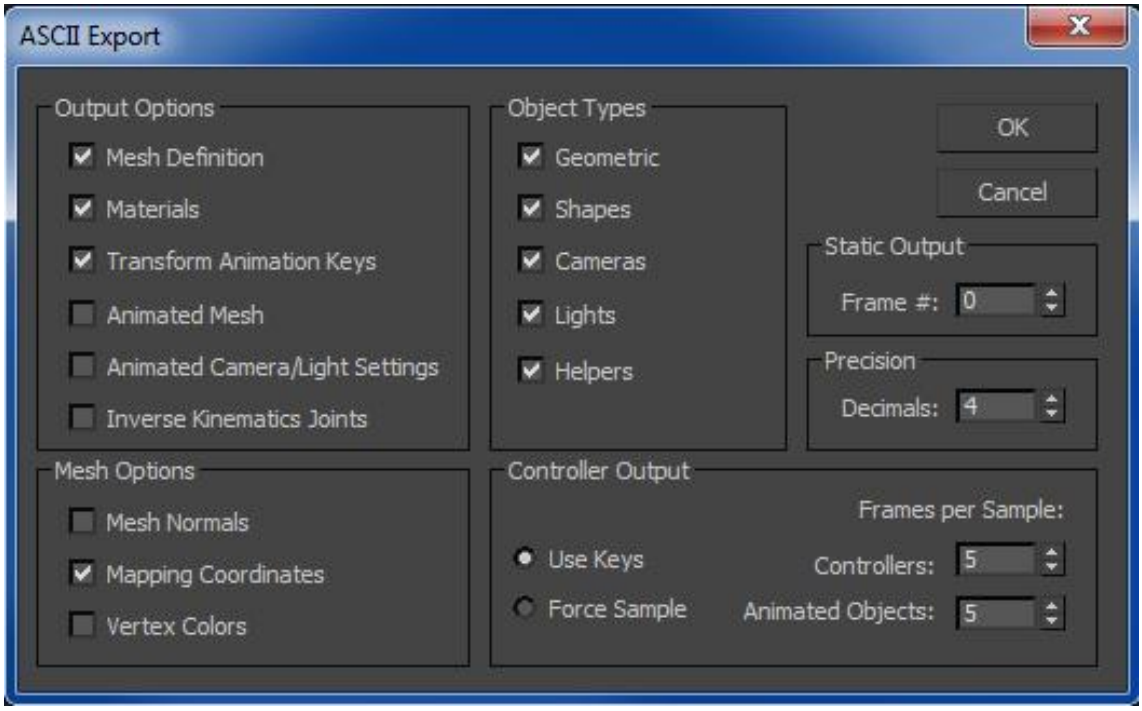
Ajoneuvon skinnaus on yksinkertaista osien skinnautuessa kokonaan yhteen luuhun. Kaikki osat pitää olla skinnattuna johonkin luuhun, jotta exporttaus onnistuu. Kannattaa lopuksi tarkistaa jokainen luu yksi kerrallaan, ettei esimerkiksi rengasta pyörittäessä liiku mikään muu osa ajoneuvoa.

3.4 Ajoneuvon vahinkomallin osat

Jos ajoneuvoon haluaa vahinkomallinnuksen, pitää se tehdä erillisellä morph targetilla. Tämä onnistuu kopioimalla ajoneuvomallin ja poistamalla luut. Yhteen morph targettiin tehdään yhden osan vahingoittuminen eli esimerkiksi konepellin rypistyminen törmäyksessä. Tässä pitää kuitenkin muistaa, että vahingoittuminen UDK:ssa toimii siten, että yhtä luuta kohti on yksi mahdollisuus vahinkomallille. Omassa mallissani käytin vain yhtä vahinkomallia, jonka lisäsin myöhemmin, jotta senkin toimivuus tulisi kokeiltua. Morph targetit pitää lopuksi lisätä luulliseen ajoneuvoon Morpher-muokkaimella.

Lisäksi ajoneuvosta voi tehdä erillisiä osia ajoneuvon tuhoutumiseen, jossa sen räjähtäessä esimerkiksi renkaat lentävät ilmaan. Kopioin erikseen yhdestä renkaasta, yhdestä etu- ja taka-akselista, aseesta ja auton rungosta mallit, jotka vietiin erikseen UDK:hon. Nämä luuttomat osat, toisin kuin itse ajoneuvomalli, exportattiin yksi kerrallaan käyttämällä Maxin omaa ”export selected” toimintaa. Tallennusmuodoksi valitaan ”ASCII Scene Export (*.ASE)”, jolloin näkyviin tulee kuvan 8 ikkuna. Varmista,

että asetukset ovat samat kuin kuvassa. Ensimmäisellä kokeilukerrallani ”Mapping Coordinates” kohdassa ei ollut ruksia, jolloin tekstuurit eivät toimineet UDK:ssa.



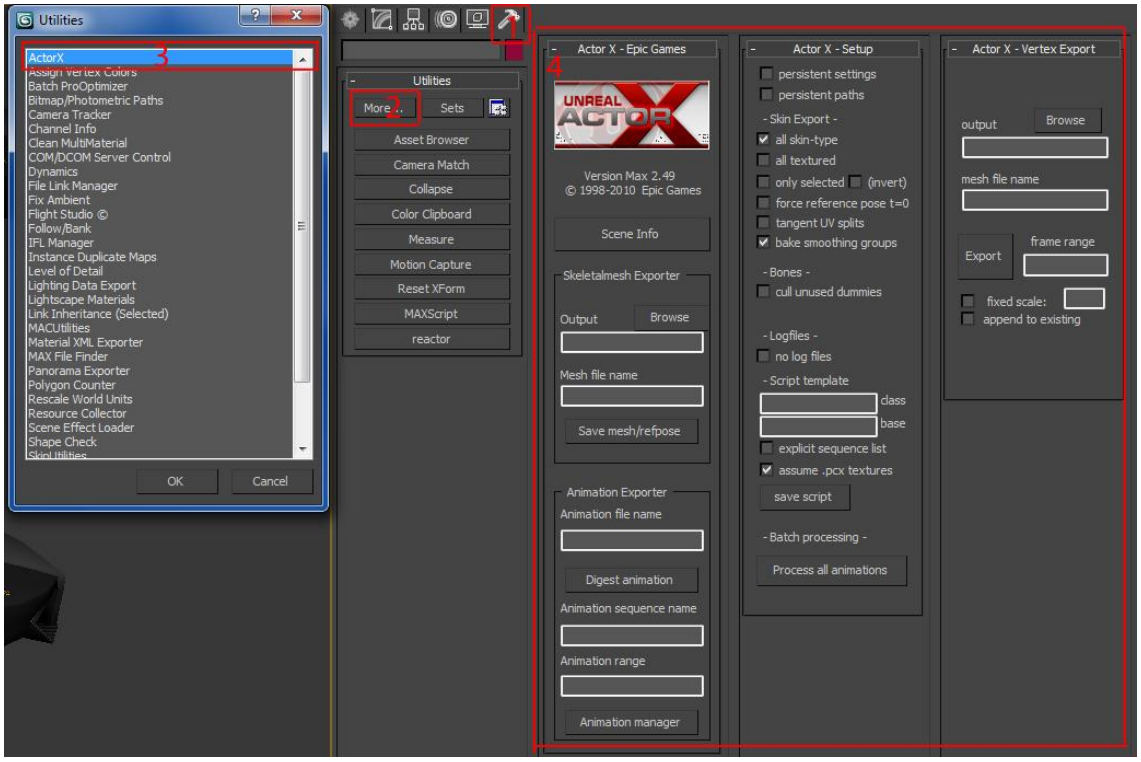
Kuva 8. Maxissa avautuva ASCII-exporttausikkuna.

3.5 Exporttaus

Itse ajoneuvon exporttaamiseen tarvitaan erillistä Unreal Actor X -pluginia. Plugin löytyy UDK:n kansiosta ”Binaries/ActorX”, jossa on vielä erikseen kansiot Maxin ja Mayan eri versioille. Oikea versio plug-inista pitää kopioida Maxin plugins-kansioon, jolloin se toimii seuraavan käynnistyksen jälkeen. Uusimpiin versioihin ei välttämättä löydy tukea, kuten tällä hetkellä, kun 3ds Max 2012:een ei ole saatavilla Actor X:ää, jolloin ainoaksi vaihtoehdoksi jää vanhemman version käyttäminen. Tässä tapauksessa, jos malli on luotu uudemmalla versiolla, se pitää tallentaa nimellä, jolloin voi valita tallennusmuodoksi vanhemman max-tiedoston. Maxissa toimii vanhat tiedostot uusissa versioissa, mutta uudet ei toimi sellaisinaan vanhoissa.

Oikein asennettuna ActorX löytyy maxin Utilities-välilehdestä, jonka ikoni on vasara (kuva 9). Itse plugin aukeaa Maxin sivuikkunaan. ActorX sisältää erilliset kohdat skeletalmeshin, eli esimerkiksi hahmon tai ajoneuvon, ja animaation ulos viemiseen. Jos tiedosto sisältää muita malleja ajoneuvon lisäksi, pitää ruksittaa ”only selected”

kohta ennen exporttausta. Ennen tallennusta valitaan ajoneuvomalli ilman luita. Skeletalmesh exporter kohtaan määritetään tallennuskansio ja nimi, ja tallennusnappia painamalla pitäisi tulla ilmoitus ”Unsmooth groups processing: [numero] vertices added”. Tässä vaiheessa tulee myös virheilmoitus mahdollisista puuttuvista skinnauksista, jolloin ne pitää korjata. Ajoneuvon lisäksi samalla tavalla tallennetaan morph targetit, jotka tehtiin vahinkomallinnusta varten. Tässä pitää muistaa, että ei tallenna erillistä muokattua mallia, vaan itse ajoneuvomalliin morpher-muokkaimella lisätty malli, joka muokkaimesta säädetään sataan prosenttiin. Tämän jälkeen malli tallennetaan, kuten edellinenkin.



Kuva 9. ActorX-plugin avaaminen Maxissa (kohdat 1-3) ja aukeava ikkuna (kohta 4)

Mahdollisten animaatioiden, kuten oven avautumisen kohdalla käytetään alempaa exportteria. Animaatiot tehdään maxissa luilla, kuten normaalistikin ottaen kuitenkin huomioon, että luuhierarkiassa luut pitää olla liitettynä ajoneuvon pääluun lapsiksi. Lisäksi jos animaatioita on useampia, ne pitää sijoittaa aikajanalle peräkkäin, eli ei päällekkäin, jolloin jokainen yksittäinen animaatio voidaan tehdä erikseen. Animaatioiden viemiseksi nimetään ylempään laatikkoon tulevan animaatiotiedoston nimi, esimerkiksi omassa tapauksessani ”Ankonian_anim”, seuraavaan laatikkoon yksittäisen animaation nimi, esimerkiksi oven avaukseen ”open_door” ja lopuksi

kyseisen animaation väli aikajanalla, eli esimerkiksi 0-20, mikä tarkoittaa, että animaatio alkaa framesta 0 ja päättyy frameen 20. ”Digest animation”-nappia painamalla animaatio lisätään animaatio manageriin, joka avautuu alimmalla napilla. Yksittäiset animaatiot pitää vielä lisätä erikseen managerissa ulosvietäviin keskellä olevia nuolia käyttämällä, minkä jälkeen paketti on valmis tallennettavaksi. Tiedosto tallentuu aiemmin määritettyyn kansioon, johon ajoneuvo tallennettiin. Jos animaatioita exporttaa ennen ajoneuvon tallennusta, kannattaa käyttää tallennusta nimellä, sillä en ainakaan itse löytänyt, mikä on PSA-tiedoston oletustallennuskansio.

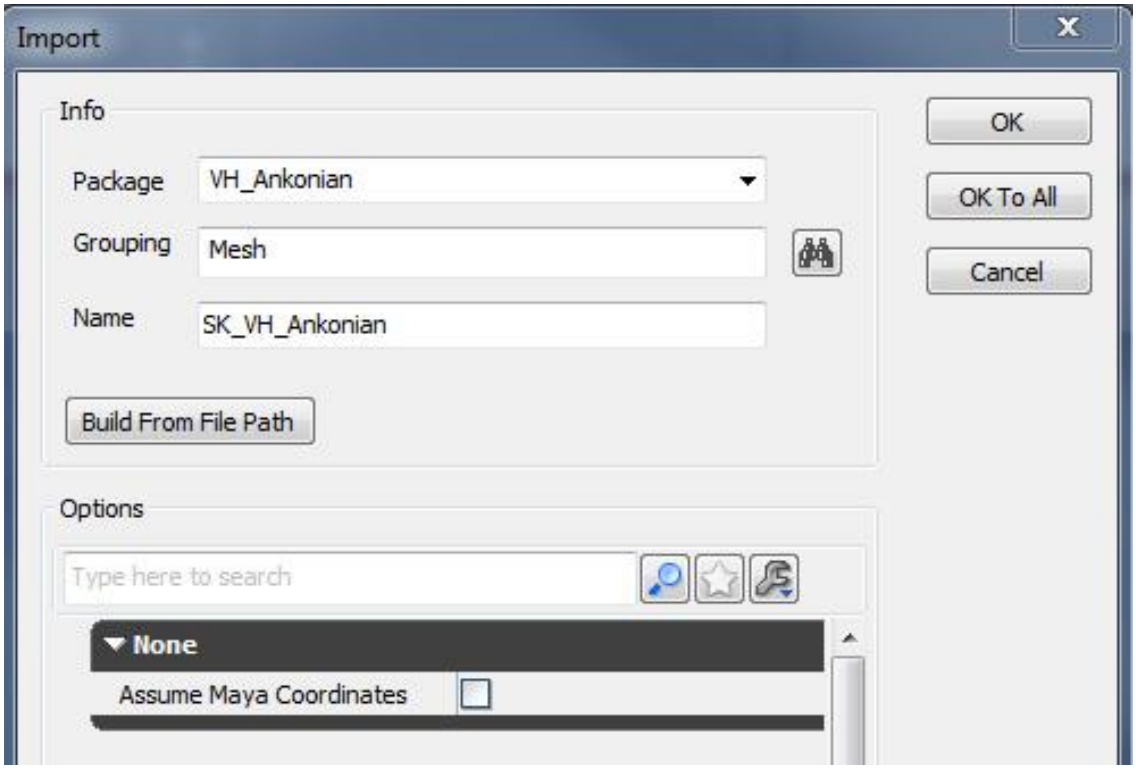
4 UDK:ssa mallista ajettavaksi ajoneuvoksi

4.1 Tiedostojen tuonti UDK:hon

Unreal Development Kitissä, tai UDK:ssa, on omalaatuinen käyttöjärjestelmä, joka näyttää päälle päin melko yksinkertaiselta ja helppokäyttöiseltä, mutta valikkojen ja painikkeiden takaa löytyy paljon uusia valikkoja ja erilaisia säätömahdollisuuksia. Oletan jonkinlaista perustietämystä UDK:sta, joten en aio joka ikistä toimintaa tarkalleen kuvata. UDK:n valmiita ajoneuvoja kannattaa käyttää hyödyksi oman ajoneuvon luonnissa. Esimerkiksi normaalin nelipyöräisen ajoneuvon pohjana on hyvä käyttää VH_Scorpionia, josta löytyy kaikki tarvittavat tiedostot tykillisen nelipyörän valmistamiseen. Scorpionin tiedostoja kannattaa käyttää esimerkkeinä, joita tutkimalla oppii ymmärtämään mitä kaikkea ajoneuvoon kuuluu ja millaisia asetuksia tarvitaan. Yleensä oletusarvoista muutetut asetukset näkyvät tummennettuina, joten ne on helppo havaita. Lisäksi jotkut tiedostot voidaan jopa kopioida, ja muuttaa tarvittaessa omaan sopiviksi. Näin tehdään myös tarvittavien koodien kohdalla, mikä helpottaa työtä huomattavasti.

Ajoneuvon tuominen UDK:hon alkaa Content Browserista, jonka vasemmassa alareunassa on import-painike. Sisääntuonti on hyvä aloittaa itse ajoneuvon skeletal meshistä, joka tallennettiin PSK-tiedostona actorX:ää käyttäen. Tiedoston valittua aukeaa ikkuna (kuva 10), johon nimetään skeletal meshin nimen lisäksi pakkauksen nimi (package) ja joukon nimi (grouping). Näissä kannattaa käyttää samaa nimeämiskäytäntöä kuin UDK:n omissa ajoneuvoissa, eli pakkauksen nimeksi VH_ajoneuvonnimi, omassa tapauksessani VH_Ankonian, ja joukon nimeksi Mesh. Ajoneuvon nimessä, kuten kaikissa muissakin nimissä UDK:ssa käytetään lyhenteitä, jotka kuvaavat tiedostoa. Esimerkiksi ajoneuvossa on nimen edessä ”SK_VH_”, jossa

SK on lyhenne sanasta skeletal (luurankoinen) ja VH sanasta vehicle (ajoneuvo), jolloin pelkästä nimestä voi jo päätellä, mikä tiedosto on kyseessä. Käytin samaa nimeämistä kaikkialla, mutta toki voi käyttää omanlaista järjestelmää, jos näkee sellaisen paremmaksi. Myöhemmässä vaiheessa, kun koodeja muokataan oman ajoneuvon mukaisiksi, työ helpottuu, kun tarvitsee vain muuttaa nimet oikeiksi. Lisäksi tapa on selkeä, joten miksi muuttaa toimivaa. Jos mallinnus on tehty Mayassa, ruksitetaan lisäksi alhaalla oleva laatikko, jotta koordinaatit ovat oikein päin.

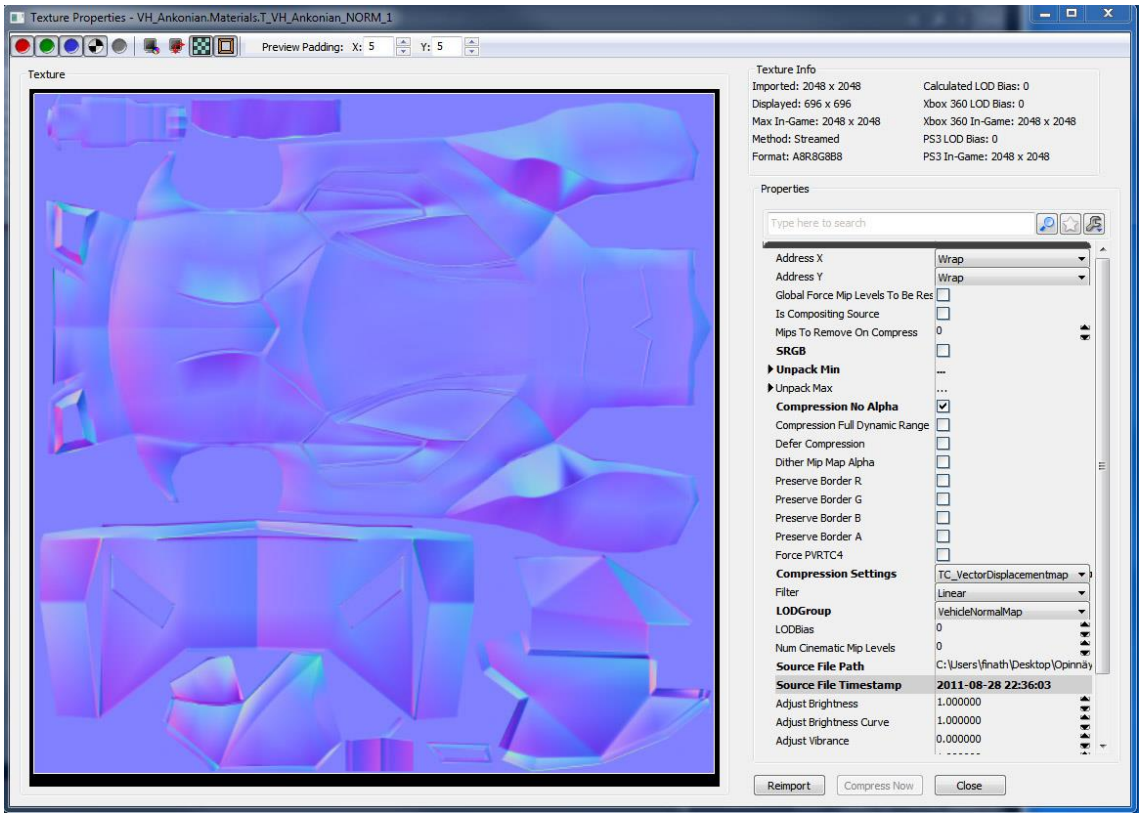


Kuva 10. Ikkuna, joka avautuu tuodessa mallia UDK:hon

OK:ta painettua malli latautuu Content Browseriin. Jos jossain vaiheessa pitää tuoda malli uudelleen UDK:hon muutoksien takia, onnistuu se helposti painamalla hiiren oikeaa painiketta kuvakkeen kohdalla ja valitsemalla ”reimport skeletelmesh”, olettaen, että uusi malli on tallennettu entisen päälle. Itse pakkaus on kuitenkin tallentamatta, joten se valitaan vasemmalla olevasta listasta, ja hiiren oikealla valitaan tallennus. Ajoneuvo kannattaa tallentaa ”Vehicles”-kansioon, joka löytyy uusimmissa versioissa polusta ”UDKGame/Content/UT3/Vehicles”. Tallentuva UPK-tiedosto on se tiedosto, johon kaikki ajoneuvon tiedostot tästä lähtien tallennetaan. Suosittelen tallentamaan jokaisen importauksen jälkeen, koska UDK:lla on tapana kaatua ilman suurempaa syytä, ja saman asian toistaminen ei ole kokemuksen mukaan erityisin mukavaa.

Räjähdystä varten luodut irtonaiset osat tuodaan samaan mesh-kansioon ajoneuvon kanssa yksi kerrallaan. Nämä osat tallennettiin erillisinä ASE-tiedostoina, joten ne ilmestyvät static mesheinä Content Browseriin. Tuodessa saattaa tulla ilmoitus, jossa ilmoitetaan verteksien määrän suhteen ylittävän oletetun prosentin, mutta tästä viestistä ei tarvitse välittää. Selaimessa valitetaan static meshien törmäysmallin (collision model) puuttumisesta. Tämä luodaan menemällä editoriin, joka avautuu kaksoisklikkauksella. Ylhäällä on ”Collision”-valikko, josta löytyy eri vaihtoehtoja. Ensimmäinen on yksinkertaisin: kuutio, joka muotoutuu mallin mittojen mukaan. Törmäysmallin saa näkyviin painamalla yläpalkin painiketta, jossa muotoa ympäröi punainen ääriviiva. Näille malleille riittänee yksinkertaisin malli, mutta muitakin muotoja voi testata, jos haluaa esimerkiksi renkaalle paremmin 3d-mallia myötäilevän törmäysmallin.

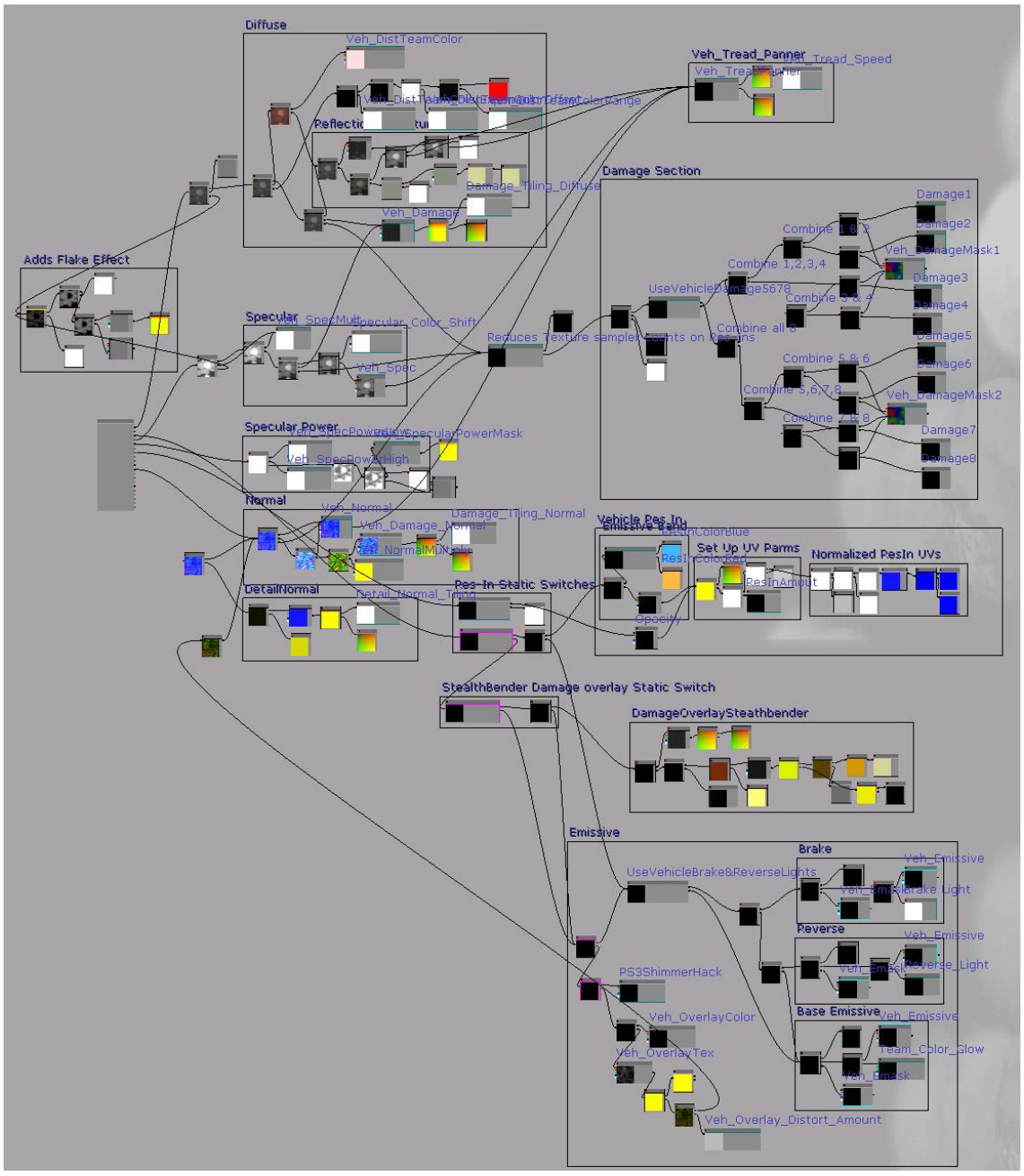
Tekstuurit tuodaan ohjelmaan samalla tavalla. UDK tukee .bmp, .pcx, .tga, .float ja .psd formaatteja. Tekstuurien koko pitää olla kahden potensseissa, esimerkiksi 1024x512, koska UDK tekee automaattisesti tekstuurien resoluution pienentämisen puolittamalla koon. Pienennystä tarvitaan tekstuurin yksityiskohtien vähentämiseksi, kun malli on kauempana kamerasta (level of detail), jolloin se vie vähemmän muistia. Lisäksi pienentämällä kokoa vähennetään sahalaitaisuutta, jota esiintyy suuriresoluutioisessa tekstuurissa, kun sitä tarkastellaan kauempaa. (Epic Games, Inc. 2013.) Oletussäätöjä import-ikkunassa ei tarvitse muuttaa, mutta joitain säätöjä tarvitsee tehdä erikseen kaksoisklikkaamalla materiaaleja, jolloin aukeaa kuvan 11 ikkuna. Normaalikartoissa laitetaan SRGB pois päältä ja ruksitetaan ”compression no alpha”, koska normaalikartta ei käytä alphakanavaa. Lisäksi ”compression settings”- kohtaan valitaan TC_Normalmap ja LODGroupiin VehicleNormalMap. Tämän avulla vaikutetaan tekstuurin pakkauksen laatuun ja tekstuurin kokoon. Ajoneuvossa on mahdollisuus käyttää suurempaa tekstuuria (2048x2048) kuin normaalisti (1024x1024). Muissa tekstuureissa laitetaan ”compression no alpha”-kohtaan ruksi sen mukaan, onko alpha kanava käytössä ja säädetään LODGroupiksi Vehicle. Tarvittaessa tekstuurin kokoa voi pienentää LODBias-kohdasta, jossa esimerkiksi lisäämällä LODBiasia yhdellä koko puolittuu.



Kuva 11. Tekstuurin säätöikkuna, jossa määritetään muun muassa pakkauslaatu.

4.2 Materiaalien luonti

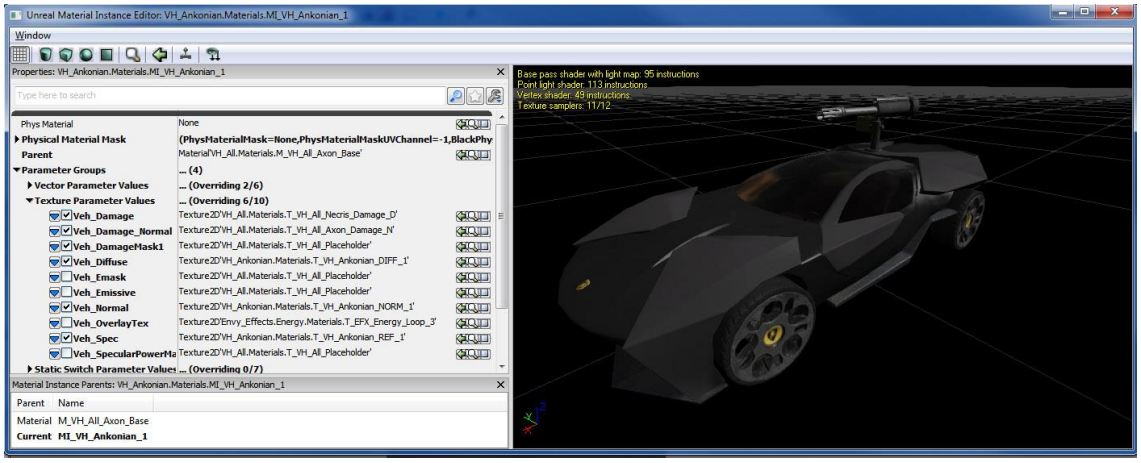
Tekstuureista pitää koostaa materiaalit, aivan kuten 3ds Maxissakin. Materiaalit voi tehdä alusta itse, jolloin on paremmat mahdollisuudet vaikuttaa materiaalin ominaisuuksiin ja ulkonäköön. Tällöin joidenkin ominaisuuksien, kuten ajoneuvon vahingoittumisessa muuttuvan tekstuurin aikaansaaminen käy haastavaksi. Materiaalin voi myös tehdä samalla tavalla kuin muissa ajoneuvoissa, eli käyttää valmista ajoneuvojen materiaalia pohjana, jolloin materiaalit vain sijoitetaan niille kuuluviin kohtiin. Tässä tapauksessa kuitenkin säätövaraa jää vähän, eikä esimerkiksi läpinäkyvän tai heijastavan materiaalin tekeminen ole mahdollista. Itse päädyin ratkaisuun, jossa lopulta yhdistin oman materiaalin edut valmiin materiaalin etuihin kopioimalla valmiin materiaalin ”create a copy” –toiminnolla ja lisäsin materiaalin sisään haluamani noodit. Tällöin sain käyttöön sekä heijastusefektit, että vahingoittumisen. Tämä vaati lähinnä taitoa ”lukea” materiaalia, että osasi sijoittaa tekstuurit oikeille paikoille varsin sekavassa materiaalissa (kuva 12). En käy läpi uuden materiaalin luontia, koska sitä varten on olemassa erilliset kattavat tutoriaalit. Sen sijaan kerron, miten tehdään samanlaiset materiaalit kuin UDK:n valmiissa ajoneuvoissa.



Kuva 12. Ajoneuvon pohjamateriaalin monimutkaisuus editorissa katsottuna.

Content Browserissa on vasemmassa listauksessa vehicles-kansion alla yksittäisten ajoneuvokansioiden lisäksi ylimpänä VH_All kansio. Sieltä löytyy Material-kansiosta pohjamateriaali M_VH_All_Axon_Base. Tätä ei tässä tapauksessa kopioida, vaan siitä tehdään instanssi klikkaamalla kuvaketta hiiren oikealla ja valitsemalla ”Create New Material Instance Constant”, joka tallennetaan oman ajoneuvon Materials kansioon. Näitä luodaan niin monta kuin on material ID:eitäkin. Mikäli aikomuksena on tehdä ajoneuvoon Unrealissa käytetettävät joukkuevärit, eli sininen ja punainen, täytyy näitä instansseja tehdä kaksi, antaen nimeksi toiselle esimerkiksi MI_VH_Ankonian_Blue ja

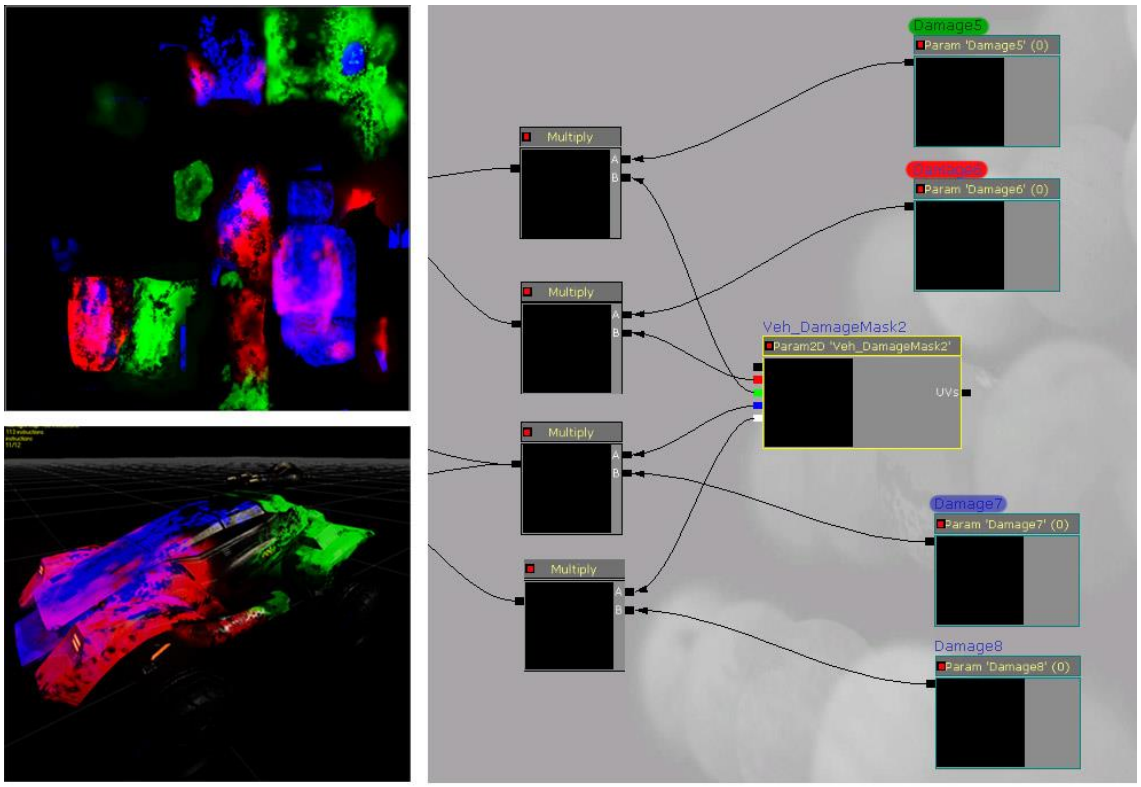
toiselle MI_VH_Ankonian_Red. Itse en tehnyt erivärisiä versioita, koska olin päätynyt tekemään ajoneuvoon useampia materiaaleja, jolloin scriptistä johtuen ei niiden teko ollut mahdollista, tai ainakaan onnistunut helposti. Tämä taas aiheutti ongelmia, kun pelasi Unrealin pelimuotoja editorin ulkopuolella, joissa tämä on vaadittua. Siinä tapauksessa vain yksi materiaaleistani toimi, jolloin tekstuuri oli tietysti suurimmalta osin väärä.



Kuva 13. Material Instance Editor, jossa säädetään tekstuurit käyttöön ajoneuvolle.

Instanssin tallennettua avautuu Material Instance Editor –ikkuna (kuva 13), jossa määritetään eri tekstuurit materiaalia varten. Jos haluaa pallon sijaan nähdä oman ajoneuvon ikkunassa, pitää sen skeletal mesh valita Content Browserissa ja painaa vihreää nuolta materiaali-ikkunassa. Texture Parameter Values -kohdasta löytyy paikat muun muassa diffuusi-, spekulaari- ja normaalikartoille. Materiaalien viereisiin laatikoihin pitää lisätä ruksi, jotta ne tulevat käyttöön. Lisäksi on mahdollista lisätä valaisevat materiaalit sijoittamalla Veh_Emissiveen valon värikartta ja ylempään Veh_Emaskiin maski, joka määrittää, mitkä kohdat ovat valaisevia. Vahinkokartat, jotka määrittävät miltä ajoneuvon pinta näyttää sen vahingoittuessa, ovat valmiiksi täytettynä oletuskartoilla, mutta jos haluaa realistisemman vahingon, on nekin luotava itse. Tyydyin kuitenkin valmiisiin tekstuureihin, mutta joka tapauksessa on luotava vahinkomaski, jolla määritetään mihin vahinko tulee. Tätä varten tehdään tekstuuri, jossa värit on maalattu suoraan RGB-värikanaviin, jolloin jokainen kanava vastaa vaurioaluetta. Esimerkiksi punainen voi vastata ajoneuvon etuosaa, vihreä sivuja ja keskiosaa, ja sininen takaosaa, jolloin esimerkiksi ampuessa edestä vahinko tulee vain etupuolelle. Tämä onnistuu tietysti helpoiten maalaamalla jossain ohjelmassa, jossa voi suoraan maalata 3d-malliin värit, ja kokoamalla eri värit yhdeksi tekstuuriksi

Photoshopissa. Kuvassa 14 on esimerkki Scorpionin vahinkomaskista avuksi maskin ymmärrystä varten. Omassa ajoneuvossa tajusin myöhemmin, että en pysty hyödyntämään maskia kunnolla, koska käytössäni oli rungolle vain yksi luu. Koodissa maskit liitetään eri luihin ja morph targetteihin, jolloin niiden maskissa valittujen vaikutusalueiden tekstuuria voidaan muuttaa erikseen.



Kuva 14. Kuvassa ylhäällä oikealla Scorpionin vahinkomaskitekstuuri, alapuolella tekstuuri ajoneuvossa, oikealla maskin käyttö materiaalissa ja alhaalla osa koodia, jossa maskin värikanavat liitetään eri osiin. Vahinkomaski tulee materiaalissa keskellä olevaan noodiin, josta sen eri värikanavat on liitetty Multiply-noodilla nimettyihin noodeihin, esimerkiksi vihreä värikanava Damage5:een. Damage5 taas on koodissa liitetty takapuskurien luihin. Täten ampuessa takapuskuria Damage5:den arvo nousee ja vihreän värikanavan alueella vahinkotekstuuri tulee näkyviin.

Materiaalista löytyy eri säätömahdollisuuksia, jotka saa päälle ruksittamalla laatikon niiden vasemmalla puolella. Scalar Parameter Values -valikosta löytyy esimerkiksi Detail_Normal_Tiling säätö, jolla voidaan vaikuttaa pinnan epätasaisuutta muokkaavan normaalikartan kokoon. Suurempi arvo pienentää kartan kokoa.

Lisäksi tarvitaan ajoneuvon vahingoittumista varten kaksi materiaalia: MaterialInstanceConstant, joka pohjautuu valmiista M_VH_Burnout materiaalista sekä MaterialInstanceTimeVarying, joka perustuu edelliseen materiaaliin. Näitä tarvitaan siis kumpaakin yksi jokaista perusmateriaali kohti, tai ainakin niitä materiaaleja, joissa tahdotaan vahingon näkyvän. M_VH_Burnout löytyy VH_All-kansion alakansioista Material. Hiiren oikealla klikkaamalla sen päällä voidaan valita ”Create New Material Instance (Constant)” ja tiedosto tallennetaan jälleen oman ajoneuvon materiaalikansioon järkevästi nimeten. Itse materiaaliin täytyy lisätä omat tekstuurit. Texture Parameters Values –kohdan alta löytyy Veh_Diffuse, johon laitetaan oman ajoneuvon väritekstuuri. Lisäksi pitää ruksittaa päälle tekstin vasemmalla puolella olevasta laatikosta. Jos vahinkoa varten on tehtynä erillinen maski, jossa eri värikanaviin on määritetty vahinkoalueet, lisätään se Veh_DamageMask1 ja Veh_DamageMask2 kohtiin. Valmiista materiaalista tehdään TimeVarying –kopio klikkaamalla hiiren oikealla ja valitsemalla ”Create New Material Instance (Time Varying)”. Tähän materiaaliin ei tarvita muutoksia.

Vahinkotekstuurit voidaan lisätä nyt StaticMesheihin. Materiaalien paikat löytyy Mesh Editorissa LODInfon alta, jossa Elements-kohta avattua löytyy materiaali määrästä riippuen yksi tai useampi kohta materiaaleille. Materiaalina tässä käytetään MaterialInstanceConstant-versiota.

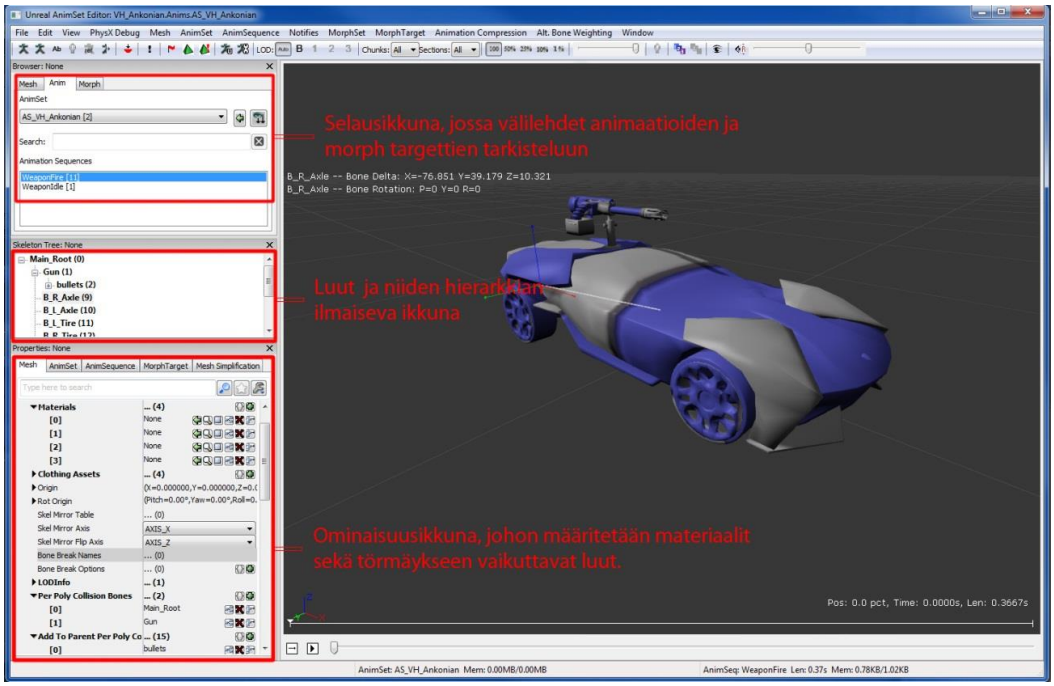
4.3 AnimSet Editor

Kaksoisklikkaamalla ajoneuvon kuvaketta päästään AnimSet Editor –ikkunaan. Tässä ikkunassa lisätään muun muassa mahdolliset animaatiot ja morph targetit ajoneuvoon. Uusi animaatiokokoelma luodaan valitsemalla yläpalkista file ja sieltä new animset. Se kannattaa tallentaa omaan ryhmitelmään, esimerkiksi Anims nimellä. MorphTarget-ryhmä luodaan samalla tavalla valiten New MorphTargetSet. Tämän jälkeen siihen voidaan tuoda animaatiota valitsemalla ”import PSA” ja morph targetteja valitsemalla import MorphTarget. Tällöin animaatiot tulevat näkyviin Anim-välilehteen, jolloin ne voidaan valita ja esikatsella niitä painamalla soittonappia ikkunan alapalkista. Morph targetit tulevat Morph-välilehteen, jossa liuttamalla janaa voidaan säädellä muodonmuutoksen määrää. Omalla kohdallani kumpaakin oli käytössä vain yksi testimielessä tehtynä, mutta useamman tuonti onnistuu samaa toistaen. Morph targettien kohdalla täytyy muistaa, että jos skeletal meshiin tehdään muutoksia ja

tuodaan uudelleen UDK:hon, vanhat morphaukset lakkaavat toimimasta oikein, ja ne täytyy tehdä uudelleen uutta mallia käyttäen.

AnimSet-ikkunassa (kuva 15) lisätään myös materiaalit ajoneuvoon, sekä määritetään luut, joiden mukaan törmäys tapahtuu (Per Poly Collision Bones). Vasemman puolen ikkunoista alin on ominaisuusikkuna, jossa nämä säädetään. Ylimpänä on Materials, josta nuolta painamalla avautuu numeroita nollasta ylöspäin. Nämä vastaavat mallinnettaessa luotuja Material ID -ryhmiä. Esimerkiksi minulla paikkoja oli neljä, joista ensimmäinen oli rungon materiaaleja, toinen pienempiä yksityiskohtia, kolmas sisätiloja ja viimeinen ikkunoita varten. Muistin käytön kannalta ideaalitilanteessa materiaaleja tulisi vain yksi, mutta osittaen suunnittelemattomuuden, eli jälkikäteisten lisäysten, ja toisaalta tietämättömyyden takia niitä lopulta oli tapauksessani neljä.

Alempana ovat kohdat luita varten. ”Per Poly Collision Bonesiin” lisätään ajoneuvon pääluu, eli omassani Main_Root –luu ja mahdollisen aseen luu. Kaikki loput lisätään alempaan ”Add To Parent Per Collision Bone” -ryhmään. Järjestyksellä ei ole väliä, mutta nimet pitää itse kirjoittaa, jolloin avuksi käy ylempi ikkuna, jossa kaikki luut ovat listattuna.



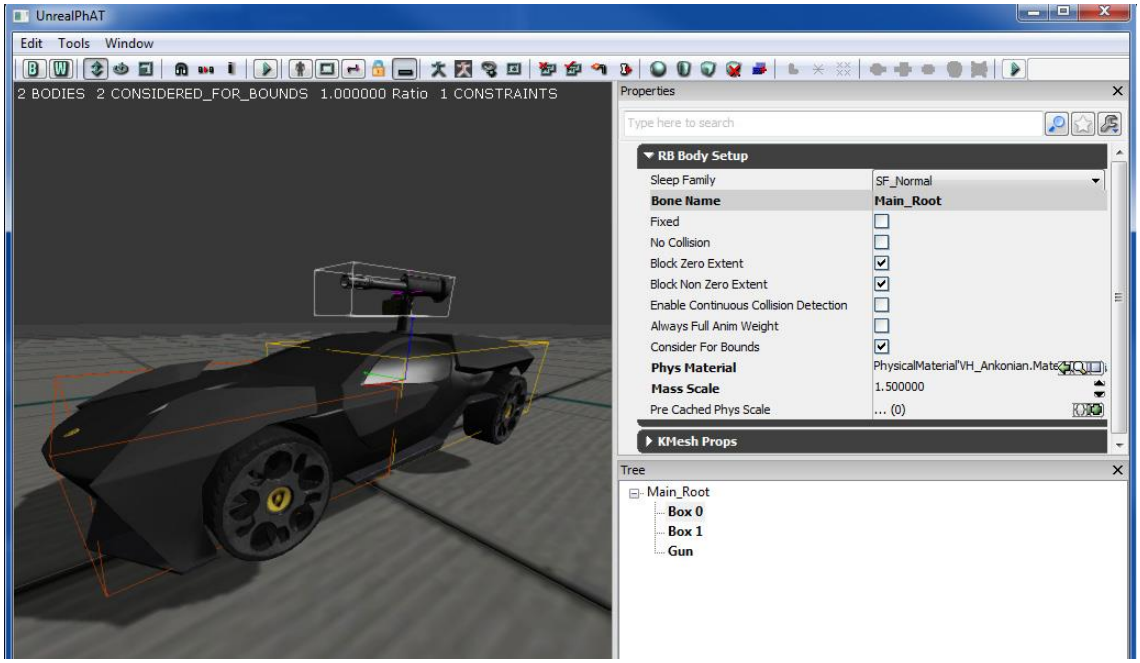
Kuva 15. AnimSet-ikkuna, jossa määritetään useita ajoneuvoon liittyviä ominaisuuksia.

Lisäksi AnimSet Editorissa lisätään pistokkeita (socket), joita tarvitaan muun muassa efektejä varten ja kuskin sijainnin määrittämiseksi. Yläpalkissa on Socket Managerin kuvake, jossa nuoli osoittaa punaista puoliympyrää. Siitä avautuu ikkuna, jossa lisätään pistokkeita painamalla ”New Socket” –nappia ja valitsemalla luu, johon pistoke alun perin sijoitetaan. Valitulla luulla on merkitystä, sillä se toimii pistokkeen vanhempana (parent), eli esimerkiksi aseen efektien pistokkeet tulee olla aseen luun lapsia (child), jolloin ne liikkuvat aseen mukana. Pistokkeiden nimiksi voi laittaa niille parhaiten sopivan, kunhan ne muistaa myöhemmin kirjata koodiin samalla nimellä. Myöhempää työtä tietysti hieman helpottaa, jos katsoo UDK:n valmiiden ajoneuvojen pistokkeiden nimet ja käyttää samoja, jolloin ne toimivat ilman muutoksia koodissa. Esimerkiksi pelaajan paikan määrittävän pistokkeen nimi on PlayerSocket ja sen määrittämiseksi voi käyttää Main_Root luuta, jolloin omassa tapauksessani sitä tarvitsi liikuttaa vain jonkin verran sivulle Y-akselilla, jotta se olisi keskellä kuskin paikkaa. Myöhemmin ajoneuvon ollessa pelissä voi hieno säätää kuskin tarkalleen oikealle paikalle. Tarvittavien pistokkeiden määrä riippuu ajoneuvosta. Ajoneuvossani niitä tuli kaikkiaan yhdeksän kappaletta. PlayerSocketin lisäksi tarvitsin VH_Death-pistokkeen, jolla määritetään kohta, josta ajoneuvon tuhoutuessa luodaan ajoneuvon rungon tuhoutunut static mesh ja irtoavat osat. Aseeseen tarvitaan yksi pistoke, TurretFireSocket, joka määrittää suuliekin ja ammuksen lähtöpaikan. Lisäksi lisätään GunViewSocket, joka vaikuttaa kameran sijaintiin ja liikkeeseen tähdättäessä. DamageSmoke01 ja ExhaustPort -pistokkeet määrittävät savuefektien paikat. Booster01 ja Booster02 ovat ajoneuvon buustiefektin alkukohdat.

4.4 Fysiikka

Ajoneuvo tarvitsee kolme eri fysiikkaan liittyvää tiedostoa, jotka määrittävät muun muassa massaa ja törmäyspinnan muotoa. Näitä varten tarvitaan yksi Physics Asset tiedosto, sekä kaksi Physical Material tiedostoa, joista toinen on ajamista varten. Physical Materialit voidaan kopioida ”Create a copy” -toiminnolla VH_Scorpionilta, jotka löytyvät sen Materials kansiosta. Nämä materiaalit toimivat sellaisenaan hyvin saman kokoisissa ajoneuvoissa. Tarvittaessa Densityä, eli tiheyttä, muuttamalla voidaan kasvattaa tai vähentää ajoneuvon massaa. Myöhemmin, kun ajoneuvo on valmiina ajettavaksi, voi myös testailla muiden ominaisuuksien säätämäistä, mutta tässä vaiheessa ne on hyvä jättää rauhaan.

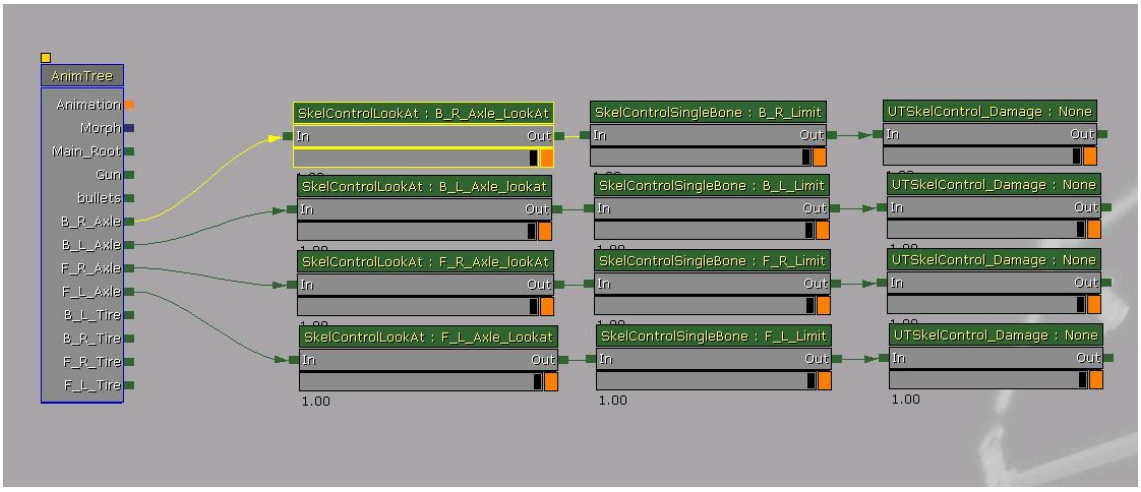
Physics Asset luodaan valitsemalla ajoneuvon skeltal meshin hiiren oikealla painikkeella avautuvasta valikosta ”Create New Physics Asset”. Avautuvaan pienempään ikkunaan ei tarvitse muita muutoksia kuin ”Create Joints” pois päältä, koska nivelille ei ole tarvetta. Seuraavaksi avautuu UnrealPhAT-ikkuna, jossa näkyy jokaista luuta kohden yksi auton osia ympäröivä laatikko. Yläpalkissa olevasta soittonapista saa fysiikkasimulaation käyntiin, jolloin oletettavasti osat lentävät ympäriinsä ja tippuvat maahan. Tärkein kappale ajoneuvolle on rungon pääluun laatikko. Physics Asset:ia käytetään pelihahmon osumisessa ajoneuvoon, joten esimerkiksi avautuvaan oveen kannattaa myös tehdä oma laatikko. Muut voi valita yksittäin puunäkymästä ja poistaa valitsemalla ”Delete All Bodies Below”. Yläpalkista löytyvät työkalut laatikoiden muokkaamiseen ja liikuttamiseen. Tarkoitus on myötäillä ajoneuvon muotoa, ei kuitenkaan liian tarkkaan, esimerkiksi omassa ajoneuvossani riittää kaksi laatikkoa rungolle ja yksi aseelle (kuvio 16). Renkaista ei tarvitse välittää, koska niiden simulaatio hoidetaan fysiikkakoodissa. Jos ajoneuvon muoto vaatii, voi tarvittaessa luoda lisää laatikoita tai muita primitiivejä yläpalkin valikosta. (Epic Games, Inc. 2013.) Ajoneuvon rungon laatikon ollessa valittuna voidaan ylemmästä oikeanpuoleisesta ikkunasta ”RB Body Setup” valikosta säätää massan mittasuhdetta (Mass Scale). Scorpionissa se oli säädettynä 1,5:een, joten siitä on hyvä ainakin lähteä liikkeelle, ja tarvittaessa myöhemmin säätää. Lisäksi sen yläpuolelle on kohta, johon valitaan fyysinen materiaali, tässä tapauksessa se materiaaleista, joka ei ole ajoa varten.



Kuva 16. Runkoa ja asetta ympäröivät laatikot, jotka toimivat yhteentörmäyksen rajoina.

4.5 Animaatiot

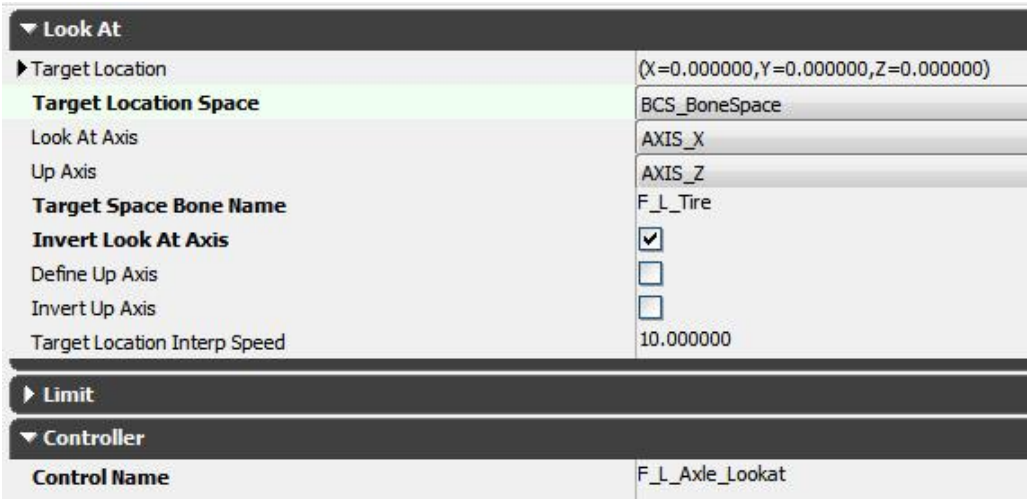
Animaatoita varten pitää luoda animaatiopuu, AnimTree, jota käytetään UE3:ssa ajoneuvo renkaiden liikuttamiseen simulaation mukaan. Lisäksi AnimTreessä soitetaan mahdolliset itse luodut animaatioit. AnimTreen luominen onnistuu Content Browserissa hiiren oikeaa painamalla ja valitsemalla ”New AnimTree”. Tallensin sen samaan kansioon kuin AnimSetin, eli Anims alakansioon ja nimesin AT_VH_Ankonian. Kaksoisklikkaamalla luotua kuvaketta avautuu AnimTree editor, jossa on oletuksena yksi noodi, AnimTree, jossa kaksi output-laatikkoa, ”Animation” ja ”Morph”. Tarkoituksena on luoda jokaiselle luulle oma output. Ensin täytyy kumminkin yhdistää AnimTree ajoneuvon skeletal meshiin. Valitsemalla AnimTree noodin avautuu alapalkkiin valikko, jossa ylimpänä on AnimTree. Sen alta löytyy ”Preview Mesh List”, jonka sisällä on ”Preview Skel Mesh”. Tähän lisätään oman ajoneuvon skeletal mesh Content Browserista. Lisäksi sen alla on ”Preview Morph Sets”, johon lisätään mahdollinen Morph Target Set. Jos haluaa nähdä animaatioita preview-ikkunassa pitää lisätä myös AnimSet-tiedosto ”Preview Anim Set List”-kohdan alle. Tämän jälkeen luiden outputtien lisääminen onnistuu klikkaamalla AnimTree noodia hiiren oikealla ja valitsemalla ”Add SkelControl Chain” ja valitsemalla listalta luun nimen.



Kuva 17. AnimTree, jossa akselin luihin on liitetty noodeja.

Kun kaikki tarvittavat luut on lisätty listaan, voidaan siirtyä tarvittavien noodien luontiin. Jokaista akselia varten tarvitaan SkelControlLookAt-noodi, johon liitetään SkelControlSingleBone-noodi, johon taas liitetään UTSkelControlDamage-noodi. Noodeja lisätään klikkaamalla editorissa tyhjää kohtaa hiiren oikealla ja valitsemalla ”New Skeletal Control”. Noodit kannattaa tehdä ensin valmiiksi yhdelle akselille, jonka

jälkeen ne voidaan kopioida muihin ja vaihtaa tarvittavat nimet vastaamaan muita akseleita. Kaikissa noodeissa pitää muuttaa asetuksia. Valitsemalla noodin alapalkkiin tulevat taas valikot (kuva 18). SkelControlLookAt -noodissa Target Location Space –kohtaan muutetaan BCS_BoneSpace. Invert Look At Axis ruksitetaan päälle ja sen yläpuolelle laitetaan akselia vastaavan renkaan luun nimi, esimerkiksi akselin ollessa oikea taka-akseli, laitetaan siihen oikean takarenkaan luun nimi, omassa tapauksessani B_R_Tire. Lisäksi annetaan itse noodille nimi Control Name kohtaan, esimerkiksi B_R_Axle_LookAt. SkelControlSingleBone –noodiin muutetaan Adjustments valikossa Apply Rotation, sekä Rotation valikossa Add Rotation päälle. Luun rotaatioavaruudeksi muutetaan jälleen BCS_BoneSpace ja kontrollerin nimeksi annetaan esimerkiksi B_R_Limit. UTSkelControl_Damagessa määritetään ajoneuvon tuhoutumisessa luotavat osat. UDK:hon erikseen static meshinä tuodut osat voivat lentää ympäriinsä ajoneuvon räjähtäessä. Jokaista irtoavaa osaa kohden pitää olla oma luu, ja tässä noodissa voidaan luulle lisätä staticmesh ”Death Static Mesh” kohtaan. Lisäksi ”On Death Active” pitää ruksittaa päälle. Kun nämä kaikki noodit on yhtä akseliluuta varten valmiina voidaan ne kopioida muille Crtl ja W käyttäen, ja muuttaa luiden, noodien ja StaticMeshin nimet.



Kuva 18. SkelControlLookAt-noodin asetukset. Muutetut asetukset näkyvät tummennettuina.

Renkaita varten tarvitaan UTSkelControl_Damagen lisäksi SkelControlWheel noodi. Sen asetuksissa tarvitsee muuttaa pyörimisakseli eli Wheel Roll Axis Axis_Y:ksi. Lisäksi kontrollerille voi antaa taas järkevän nimen, esimerkiksi oikealle takarenkaalle B_R_Tire_Cont. Jälleen kummatkin noodit voi tehdä valmiiksi yhdelle, ja kopioida muille renkaille nimet muuttaen. Lisäksi Wheel Displacementillä voidaan tarvittaessa

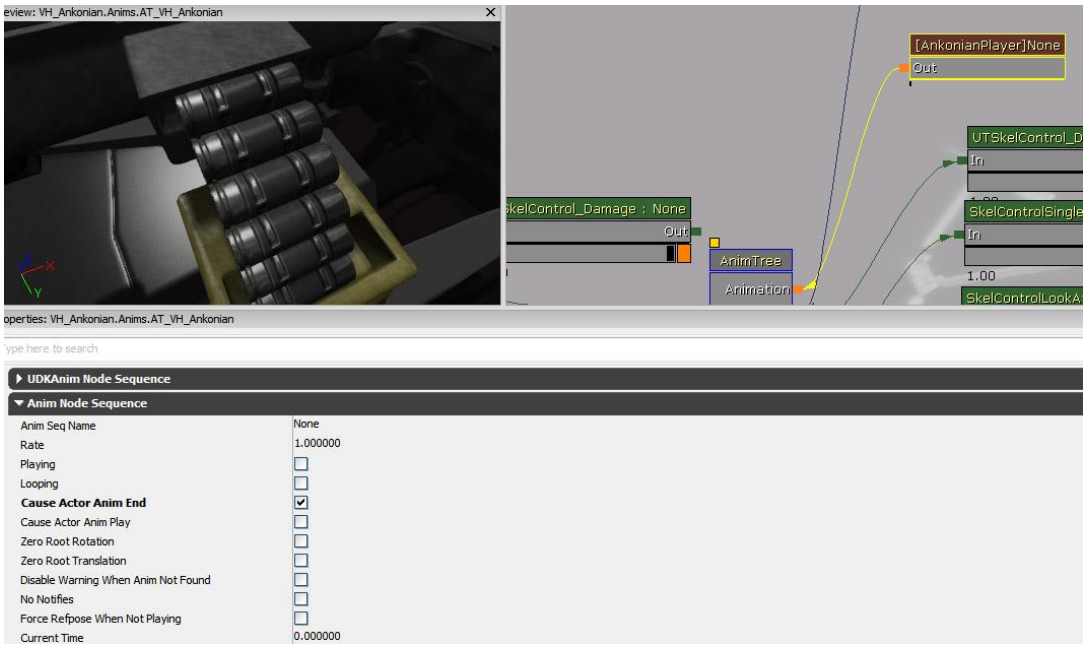
muuttaa renkaan sijaintia korkeusuunnassa, jos pelissä huomataan, että rengas osuu runkoon tai on muuten väärässä kohdassa. Myös Wheel Max Render Displacementiä voi muuttaa, jolla estetään renkaan painumista rungon sisään. Nämä kannattaa tietysti tehdä siinä vaihessa, kun ajoneuvo on ajettavana pelissä, jolloin mahdolliset ongelmat tulevat esille.

Mikäli ajoneuvossa on liikuteltava ase, sitä varten tarvitaan vahinkonoodin lisäksi UTSkelControl_TurretConstraint. Sen avulla määritetään missä suunnissa ase liikkuu, ja maksimikulmat liikkeelle. Nämä säädöt ovat tietenkin asekohtaisia. Omassa tapauksessani halusin, että asetta voi pyörittää sivuttain ja pystysuunnassa, muttei pyörittämään, jolloin ruksitin Constraint Pitch ja Constraint Roll -kohdat. Maksimi- ja minimikulmiksi tuli testailujen jälkeen 65 ja -15 Pitch kohtaan, jolloin ase ei katolla leikannut ajoneuvon läpi ylös tai alas tähdätessä.

Ajoneuvon rungolle ei ole välttämätöntä lisätä noodeja, paitsi UTSkelControl_Damage tuhoutumista varten. Tämä siis liitetään ajoneuvon pääluuhun, omassa tapauksessani Main_Rootiin. Lisäksi, jos on ajoneuvon vahingoittumista varten tehtynä morph targetteja, ne saadaan toimimaan lisäämällä MorphNodeWeight –noodi johon liitetään MorphNodePose. Poselle pitää Morph Name kohtaan antaa sama nimi kuin sille annettiin AnimSet editorissa eli omassa tapauksessani S_VH_Ankonian_body_dam. Jos morphaus toimii, sen pystyy näkemään esikatseluikkunassa liuttamalla MorphNodeWeightin alla olevaa aikasäädintä.

Animaatiot tarvitsevat omat noodinsa toimiakseen. Olin tehnyt yhden animaation testausta varten, joka oli ampumisanimaatio. Erillaiset animaatiot tarvitsevat erityyppisiä noodeja, lähtökohtaisesti kuitenkin ainakin yhden noodin animaation soittoa varten. Animaation saaminen toimivaksi aiheutti hankaluuksia, kun alunperin käytin Scorpionin AnimTreetä lähtökohtana animaation lisäämiseen. Siellä oli käytetty ajoneuvon edessä olevalle saksimaiselle aseelle soitinta, joka tekee saksien avaamisanimaation. Kuvittelin, että oma ampumisanimaationi pitäisi tehdä samalla tavalla, mutten koskaan saanut sitä toimimaan. Myöhemmin apu löytyi Youtuben tutoriaalivideosta, jossa neuvottiin käyttämään Cicadan AnimTreetä esimerkkinä (Youtube, 2013). Tällöin tarvittiin vain yksi noodi, UTAnimNodeSequence, joka toimi soittimena animaatiolle. Tätä noodia käytetään AnimSetistä löytyvien animaatioiden soittamiseen. Siihen voi ”Anim Seq Name” –kohtaan nimetä yksittäisen animaation, tai jättää nimen pois, jolloin koodissa määritetään soitettava animaatio. Tässä

tapauksessa nimi jätetään pois. Tärkeä on valita nimenomaan UT-alkuinen noodi, eikä UDK-alkuinen, joka ei toiminut syystä tai toisesta. Ainoat muutokset noodiin on nimeäminen ”Node Name”-kohtaan, jotta se voidaan kutsua koodissa. Soitin löytyy hiiren oikealla klikatessa ”New Animation Sequence”-kohdasta. Kuvassa 19 näkyy soitin liitettynä puun Animation-porttiin.



Kuva 19. Ampumisanimaation soitin. Noodi on UTAnimNodeSequence, jonka nimi kutsutaan koodissa.

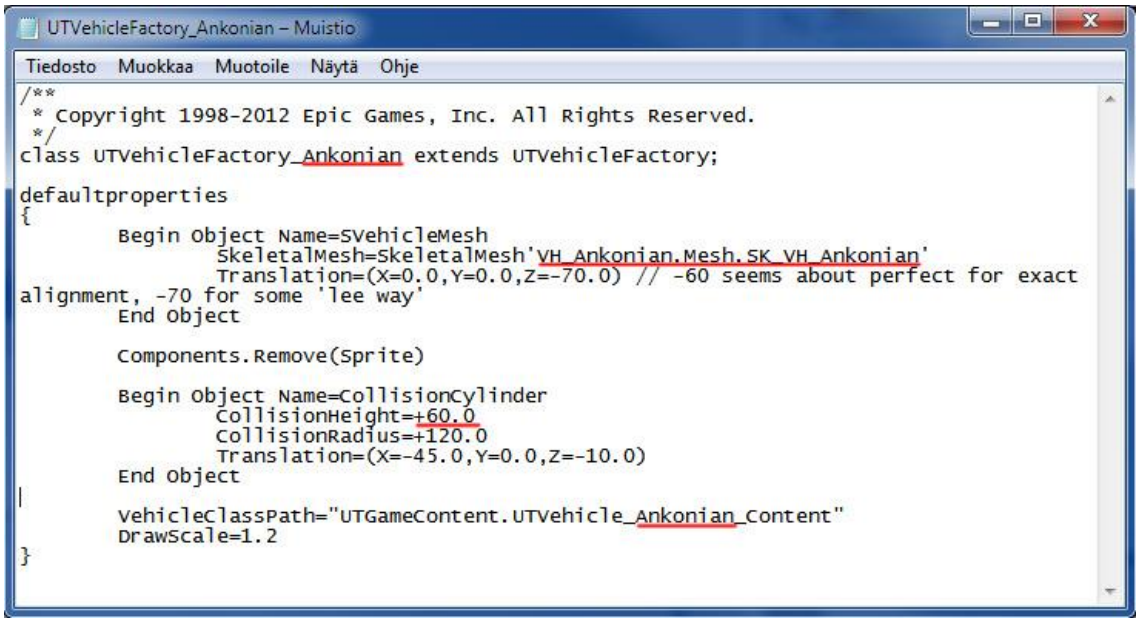
4.6 UnrealScript-koodit ajoneuvolle

Ajoneuvolle tarvitaan eri kooditiedostoja määrittämään ominaisuuksia ja yleensäkin toimivuutta. Näiden tekeminen onnistuu tarvitsematta varsinaista koodaustaitoa kopioimalla UDK:n valmiiden ajoneuvojen tiedostoja ja muokkaamalla niitä omiin tarpeisiin. Tietyille asioille voi tarvita koodia, joita ei valmiiksi ole toisten ajoneuvojen koodeissa. Epic Gamesin verkkosivuilta löytyy ajoneuvolle tekninen ohjeistus, josta löytyy eri ominaisuuksille koodipätkät. Näiden käyttö vaatii ymmärrystä siitä, mihin ne sijoitetaan ja mitä arvoja niille annetaan. En käy enempää niitä läpi, mutta tämä on hyvä tietää ja kannattaa ainakin käydä läpi, mitä ominaisuuksia on mahdollista lisätä ja säätää. Oma ajoneuvoni oli tehty hyvin pitkälti Scorpionin kaltaiseksi, joten kopioin Scorpionin kooditiedostoja. Tarvitsin kaikkiaan viisi kooditiedostoa; UTVehicleFactory-tiedoston, joka tarvitaan lisätäkseen ajoneuvon peliin, UTVehicle-tiedosto, jossa on ajoneuvon ominaisuudet ja toimintaan liittyvä koodi, UTVehicle_Content-tiedosto, johon

määritetään ajoneuvoon liitettävät eri tiedostot, UTWeap-tiedosto asetta varten sekä UTVehicleWheel-tiedosto renkaiden ominaisuuksia varten. Tarvittavat tiedostot sijaitsevat UDK:n Development kansion alla, josta Src-kansion takaa löytyy UTGame ja UTGameContent –kansiot.

UDK kannattaa olla tässä vaiheessa päällä, koska sieltä pitää tarkastaa tiedostojen nimiä. Tarvittaessa tiedostosta saa koko nimen polkuineen siinä muodossa kuin ne tarvitaan koodiin valitsemalla tiedoston, klikkaamalla sitä hiiren oikealla ja valitsemalla ”Copy Full Name to Clipboard”. Kannattaa myös muistaa, että kooditiedostoja muuttaessa muutokset astuvat voimaan vasta, kun UDK:n käynnistää uudelleen ja vastaa myöntävästi käynnistyksen yhteydessä ilmestyvään kysymykseen skriptien uudelleenrakentamisesta. Jos kaikki koodit eivät ole valmiita, saattaa se aiheuttaa virheen, joka estää ohjelman käynnistämisen. Tällöin voi yrittää käynnistää ohjelman ilman uudelleen rakentamista.

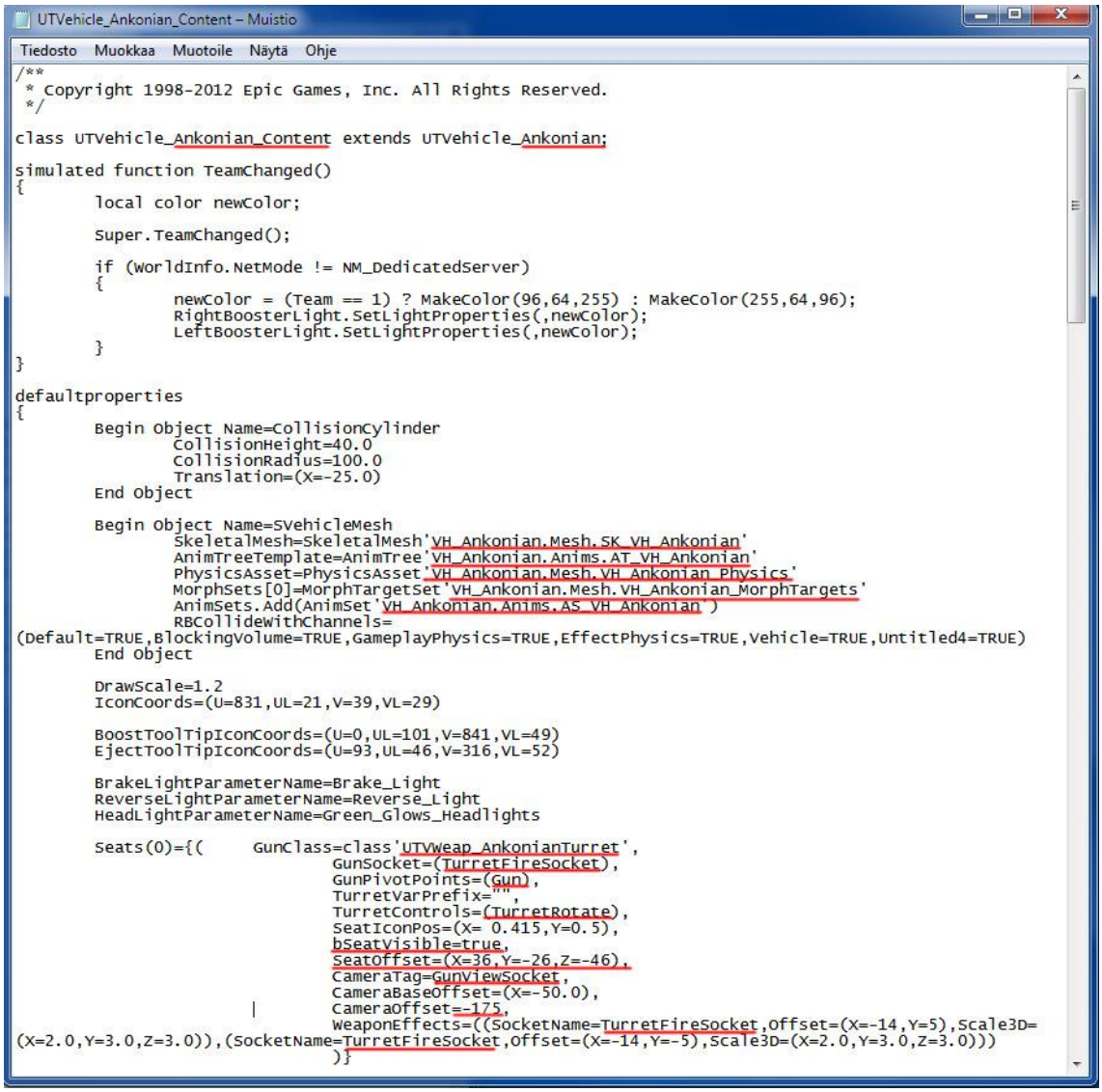
Aloitin UTVehicleFactory-tiedostosta, joka löytyy UTGameContentin Classes-kansiosta. Kopioin siis UTVehicleFactory_Scorpion-tiedoston ja nimesin muuttamalla Scorpionin Ankonianiksi. Itse koodiin tarvitaan muutama muutos. Kohdat, joissa lukee Scorpion vaihdetaan oman ajoneuvon mukaisiksi, siis tässä tapauksessa Ankonian. SkeletalMesh-kohtaan määritetään ajoneuvo polkuineen Vehicles-kansion sisällä. Kansiot erotetaan pisteellä, jolloin esimerkiksi omaksi polukseni tuli VH_Ankonian.Mesh.SK_VH_Ankonian (kuva 20). Lisäksi alempana olevaan kohtaan määritetään törmäyssylinterin (collision cylinder) koko, mikä riippuu ajoneuvon mitoista. Tämän säätäminen kannattaa tehdä siinä vaiheessa, kun pääsee testaamaan ajoneuvoa pelattavana. Törmäyssylinteri vaikuttaa pelaajan ja ajoneuvon törmäykseen, muttei esimerkiksi luodin. Omassa ajonneuvossa laskin hieman korkeutta, jotta ajoneuvon päälle pystyi hyppäämään helpommin. Myöhemmin kylläkin tajusin, että ajoneuvossani ei sylinteri ole käytössä, koska luut on määritetty törmääviksi.



Kuva 20. UTVehicleFactory-tiedoston koodi. Muutokset alleviivattu punaisella.

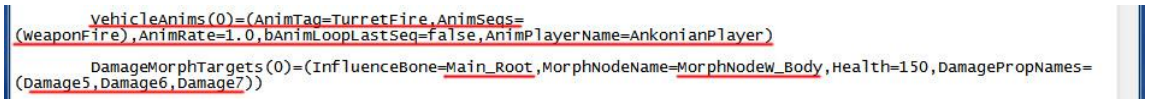
Seuraavaksi kopioin samassa kansiossa sijaitsevan UTVehicle_Scorpion_Content tiedoston (kuva 21), johon vaihdoin Ankonianin nimeen Scorpionin tilalle. Tähän tiedostoon tulee melko paljon muutoksia, jotka ovat kuitenkin nopeasti tehtävissä, jos alusta asti on käyttänyt samaa nimeämiskäytäntöä kuin valmiissa ajoneuvoissa. Suurin osa muutoksista on polkujen vaihtamista omiin, jolloin ne voi tarvittaessa kopioida UDK:ssa itse tiedostoista, kuten aiemmin mainitsin. Nimissä kannattaa olla tarkkana, koska kirjoitusvirhe voi aiheuttaa ajoneuvon tai jonkin sen ominaisuuden toimimattomuuden. Koodin yläosassa on defaultproperties-kohdan alla paikat SkeletalMeshin, AnimTreen, PhysicsAssetin, MorphSetin ja AnimSetin poluille. Hieman alempana on kohta, joka alkaa Seats(0)-rivillä. Tämän viereen määritetään aseen tietoja. Luokkaan laitetaan aseen kooditiedoston nimi, joka luodaan myöhemmin, omassa tapauksessani UTWeap_AnkonianTurret. Aseen pistoke (socket), joka määritettiin aiemmin laitetaan sen alle. Jos käytit samoja nimiä kuin Scorpionissa, eli TurretFireSocket, tähän ei tarvita muutosta. GunPivotPoints kohtaan tulee sen luun nimi, joka liikuttaa asetta, omassa tapauksessani siis ”Gun”. TurretControls kohtaan määritetään AnimTreessä aseen luuhun liitetyn UTSkelControl_TurretConstrained-ohjaimen nimi, joka pysyy myös samana, jos olet käyttänyt samaa nimeämiskäytäntöä. Jos haluaa pelaajan näkyväksi ajaessa, tulee lisätä rivit ”bSeatVisible=true” ja SeatOffset=(X=0,Y=0,Z=0), johon myöhemmin voidaan muuttaa arvoja kuskin oikean paikan löytämiseksi. Lisäksi CameraTagissa ja WeaponEffects kohdassa olevat nimet tarvittaessa muutetaan vastaavien pistokkeiden nimiin. Mikäli haluaa useamman

henkilön ajoneuvon, pitää kaikille paikoille tehdä oma koodiosa, esimerkiksi Seats(1) seuraavalle paikalle ja tarvittavat tiedot sen alle samalla tavalla kuin ensimmäisessä.



Kuva 21. UTVehicle_Ankonian_content-tiedostossa tuli paljon muutoksia. Lisäsin myös pari riviä GunClass-koodiin, jotka tekivät kuskista näkyvän ja paikan oikeaksi ajaessa.

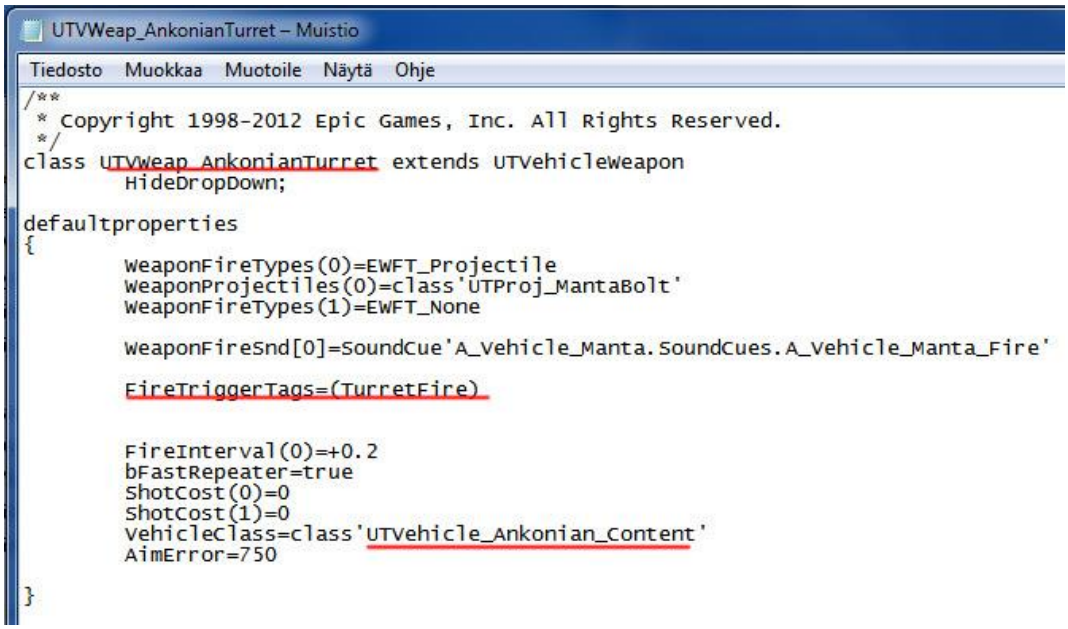
Animaatiota varten lisätään oma koodi, jossa määritetään animaatiomerkki (tag), jolla animaatio soitetaan, animaation nimi sekä animaation soittimen nimi (kuva 22). Jos animaatioita on useampia, lisätään jokaiselle oma rivi muuttaen numeroa sulkujen sisällä.



Kuva 22. Animaatiolle lisätty koodi Content-tiedostoon ja morph target osan muokkaus.

Morph targetille on Scorpionin koodissa oletuksena useampi rivi, jossa määritetään kaikkien luut, Morph Target Setistä löytyvät nimet sekä maskin värikanavia vastaavat nimet. Omassa tapauksessani tarvitsin vain yhden, joten poistin loput ja muokkasin nimet oikeiksi. Koodin lopussa on materiaalilistauksia, joihin muutetaan polut oikeiksi. SpawnMaterialLists kohtaan laitetaan kaikki materiaalit, hakasulkeiden sisälle materiaali-ID-numero alkaen nollasta. TeamMaterials kohtaan laitetaan mahdolliset joukkuekohtaiset materiaalit, eli toiseen sininen ja toiseen punainen materiaali. Lisäksi alempana määritetään kummatkin fysikaalimateriaalit ja tuhoutumismateriaali (BurnOut).

Samassa kansiossa luodaan vielä aseelle oma tiedosto. Tässä tapauksessa kopioin Mantan aseen tiedoston UTWeap_MantaGun, koska sen ampumisefekti on lähimpänä omalle aseelleni sopivaa. Tässä tiedostossa ei nimen lisäksi tarvitse muuttaa kuin yläosassa oleva nimi UTWeap_AnkonianTurretiksi ja alhaalla VehicleClass kohtaan oman ajoneuvon Content-tiedoston nimi (kuva 23). Lisäksi animaatiota varten lisäsin FireTriggerTagin nimen, joka siis kertoo koodille milloin ammutaan.



Kuva 23. Aseen kooditiedosto. Tässä tapauksessa käytin pohjana Mantan asetta, joka oli lähimpänä omaani ammuksen osalta.

UTGame kansion Classes-alakansiossa on luotava UTVehicle ja UTVehicleWheel – tiedostot. Jälleen UTVehicleScorpionWheel tiedosto kopioidaan ja nimetään uudelleen. Koodiin ei tarvita vielä muutoksia, mutta myöhemmin renkaan koko kannattaa

testaamalla säätää oikeaksi ja muita arvoja muuttamalla voi vaikuttaa ajoneuvon ajettavuuteen. UTVehicle_Scorpion on viimeinen kopioitava tiedosto. Tähänkin tiedostoon tarvitaan lähinnä muutos yläosan class kohdan nimeen ja koodin alaosassa olevaan osioon, jossa renkaiden luokkien nimet pitää vaihtaa vastaamaan oman ajoneuvon Wheel-tiedostoa. Tämä siis, jos nimet ovat luilla ja AnimTreen kontrollereilla samoja. Lisäksi ylempänä on defaultpropertiesin alla säädettäviä ominaisuuksia, kuten ajoneuvon energiamäärä (health) ja nopeus.

Kun kaikki koodit on tehtynä. Voidaan kokeilla ajoneuvon toimivuutta. Ensin täytyy tietenkin käynnistää UDK, ja päivittää skriptit. Tässä vaiheessa voi tulla virhesanomia, jotka ovat väriltään keltaisia tai punaisia niiden vakavuudesta riippuen. Konsoli kertoo tiedoston ja kohdan, joka virheen aiheuttaa. Useimmin ainakin omalla kohdallani kyse oli kirjoitusvirheestä, tai polun väärin muistamisesta, jolloin se pitää käydä korjaamassa ja kokeilla uudestaan. Jos virheitä ei tule, alhaalle tulee vihreällä viesti ”Success – 0 errors, 0 warnings.” UDK ei saa koodeja päivittäessä olla auki, vaikka toisen UDK:n käynnistäminen on mahdollista, sillä se aiheuttaa aina virhesanomia uudelleenrakennuksessa.

UDK:ssa ajoneuvo lisätään Content Browserin viereisestä välilehdestä ”Actor Classes”. Sieltä löytyy kohta Vehicles, jonka alla pitäisi olla oman ajoneuvon tehdas (factory), eli esimerkiksi UTVehicleFactory_Ankonian. Ajoneuvon lisäys peliin onnistuu yksinkertaisesti raahaamalla se pääikkunaan. Ajoneuvo saattaa näyttää menevän maan läpi editorissa, mutta peliin mennessä ajoneuvo on kuitenkin maan pinnalla. Jos kaikki on mennyt hyvin, pitäisi ajoneuvo nyt olla pelattavissa. Tästä alkaakin mahdollisten vikojen korjaus ja säätövaihe. Jos haluaa aseet uudemmissa UDK:n versioissa käyttöön, pitää View-valikon alta valita World Properties, jossa Game Type kohdasta voidaan vaihtaa pelimuodoksi UTDeathmatch. Ajoneuvo ei vahingoitu ammuksista, ellei ensin käy sen sisällä. Tämän jälkeen pitäisi pystyä testata ajoneuvon vahingoittumista ja tuhoutumista.

Ongelmien ratkaisussa voi auttaa pelkkä looginen päättelykyky, jos esimerkiksi renkaat pyörii vääränsuuntaisesti, on vika mahdollisesti AnimTreen SkelControlWheel-noodissa. Jotkut ongelmat voivat johtua kirjoitusvirheistä koodissa, jolloin ongelmaan liittyvät koodit kannattaa tarkistaa. Tietyt ongelmat voivat olla lähtöisin jo Maxista ennen exportausta, kuten aiemmin kertomani massakeskipisteen takapainotteisuuden tapauksessa. Usein melkein minkä tahansa ongelman ollessa kysessä löytyy ratkaisu

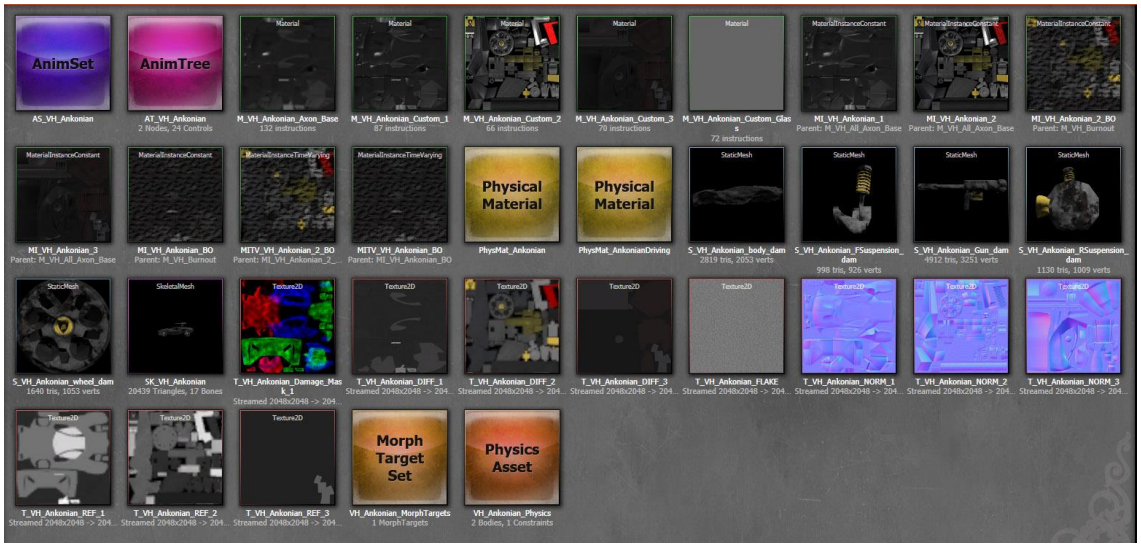
Epic Gamesin foorumeilta, koska jollakin muulla on ollut jo sama ongelma, johon on löydetty ratkaisu. Oman ajoneuvoni sain lopulta melko kivuttomasti toimimaan ensimmäisellä yrityksellä. Oikeastaan animaation toimimattomuuden lisäksi oli vain ongelmana renkaiden leikkaaminen osittain rungon sisään. Tämän korjaus onnistuu koodissa säätämällä jousituksen jäykkyyttä suuremmaksi. Jos haluaa testata asetuksia helpommin kuin muuttamalla koodia ja lataamalla UDK:n aina uudestaan, tarvitaan konsolikomentoa pelissä. Tämä onnistuu pelatessa painamalla Tab-näppäintä ollessa ajoneuvon sisällä ja kirjoittamalla ”editactor class=vehicle” (Epic Games, Inc. 2013). Tällöin aukeaa ajoneuvon ominaisuusikkuna, josta löytyy kaikki koodissa olevat säädöt. Näitä muokkaamalla tapahtuu muutokset pelissä reaaliajassa. Ainoa ongelma on valikkojen ja ominaisuuksien määrä, mikä vaikeuttaa halutun säädön löytämistä. Yläpalkissa on kuitenkin etsintäpalkki, johon kirjoittamalla näkyviin jäävät vain avainsanan sisältävät säädöt. Pitää myös muistaa, että tässä valikossa tehdyt muutokset eivät jää voimaan, vaan lopullinen muokkaus pitää tehdä siihen kooditiedostoon, jossa ominaisuus on.

5 Pohdintaa lopuksi

Aloittaessani opinnäytetyötä en osannut odottaa kohtaamiani vaikeuksia nimenomaan kirjallisessa osuudessa, jonka kirjoittaminen vei turhan pitkään. Käytännön osuus oli mukava suorittaa ja tuli tehtyä ajallaan, mutta inspiraatio kirjoittaa löytyi oikeastaan vasta sitten, kun se oli pakko löytää. Tästä syystä jouduin myöhemmin tuomaan ajoneuvon uudelleen UDK:hon ja käymään työvaiheet uudelleen läpi palauttaakseni mieleeni, mitä kaikkea työ oikeastaan pitikään sisällänsä. Toisaalta oli ihan hyvä, että välissä tuli käytyä harjoittelussa pelifirmassa, jolloin tuli kokemuksen kautta parempi ymmärrys peleihin liittyvistä rajoituksista esimerkiksi tekstuurien määrän suhteen.

Työ alkoi siis käytännön vaiheella, jossa mallinsin Lamborghini Ankonian – konseptiauton ja tein sille tarvittavat luut liikkumista varten UDK:n pelimoottorissa. Haastavin työvaihe tuli exporttauksen jälkeen, kun piti saada ajoneuvosta täysin toimiva pelattavana. Valmiissa ajoneuvossa olikin päälle kolmekymmentä tekstuuria, materiaalit, 3d-mallia ja muuta ajoneuvoon liittyvää tiedostoa (kuva 24). Erilaisia ongelmia aiheutti muun muassa ohjeiden puutteellisuus, kun virallisilta sivuilta löytyvässä tutoriaalissa ei listattu kaikkia työvaiheita, mutta yrityksen ja erehdyksen kautta löytyi usein ratkaisu, sekä tietysti selaamalla Epic Gamesin foorumeja, joista löytyi kattava ketju ajoneuvoihin liittyen UDK:ssa. Yleensäkin nykyään on kohtuullisen

helppo löytää ratkaisuja Googlen avulla ongelmaan kuin ongelmaan, ja viimeistään itse kysymällä foorumeilta ratkaisu yleensä löytyy. Pyrin kuitenkin omalla kohdalla pitämään myös omaa ongelmanratkaisukykyä yllä yrittämällä ratkaista ongelmat ensin itsenäisesti päättelemällä ja testaamalla. Tällöin usein samalla oppii ohjelman toimivuudesta ja ominaisuuksista jotain uutta ja parhaimmillaan nimenomaan ymmärtämään, mikä auttaa myös tulevissa ongelmissa.



Kuva 24. Kaikki lopullisen ajoneuvon tiedostot UDK:ssa. Lisäksi koodipuolelle tuli viisi tiedostoa.

Lopputuloksena oli ajettava ajoneuvo, jossa on toimiva tykki katolla, kuten alkuperäisessä suunnitelmassa olikin. Ajoneuvo on kuvassa 25 samassa kuvakulmassa kuin opinnäytetyön olevassa konseptikuvassa, josta näkee lopullisen mallin onnistumisen verrattuna alkuperäiseen. Sain lopulta kaiken toimimaan suunnitelmien mukaan. Suurin ongelma oli animaation toiminnassa, jonka sain vasta viime hetkillä ratkaistua Youtube-videon avulla. Animaation toimimattomuus oli turhauttavaa, koska en ymmärtänyt, mistä se johtui, eikä aiemmin useat yritykset tuottaneet toivottua tulosta. Ongelma oli siinä mielessä kuitenkin hyvä, että se pakotti tutustumaan ajoneuvon koodeihin tarkemmin ja opin uusia asioita silloinkin, kun en saanut animaatiota toimimaan.

Kaiken kaikkiaan projekti oli onnistunut. Sain syvennettyä osaamistani reaaliaikamallien luomisessa esimerkiksi teksturoinnin osalta. Mallista ei missään nimessä tullut täydellinen. Jos tekisin ajoneuvon uudelleen, kiinnittäisin enemmän huomiota optimointiin, varsinkin tekstuurien ja materiaalien määriin, jotka kieltämättä

osaamattomuuttani nousivat liian suuriksi. Luultavasti myös tekisin enemmän osia vahingoittumista varten, esimerkiksi aukeavat ja irtoavat ovet ja konepellin. Koodipuolesta opin jonkin verran ymmärtämään ihan vain tarkastelemalla koodin rakennetta, mutta siihen olisi mukava syventyä syvällisemmin ja oppia ainakin perusteet kunnolla, jolloin voisi mahdollisesti saada itse jotain aikaiseksi koodaamalla. Tämä tietysti avaisi uusia mahdollisuuksia, kun tekemiset eivät rajoittuisi valmiiseen koodiin.

Toivottavasti työstäni on hyötyä oppaana niille, jotka haluavat oman ajoneuvonsa pelattavaksi. Jatkotutkimusta olisi mielenkiintoista nähdä varsinkin koodipuolesta, mikä avaisi paljon uusia mahdollisuuksia erityyppisille ajoneuvoille.

Kuva 25. Lopullinen Ankonian pelissä.

Lähteet

Hajioannou, Yanni 2013. What Is a Normal Map? [verkkodokumentti]
http://gamedev.tutsplus.com/articles/glossary/quick-tip-what-is-a-normal-map/
(luettu 23.5.2013)

Epic Games, Inc. 2013, Importing Texture Tutorial. [verkkodokumentti]
http://udn.epicgames.com/Three/ImportingTextureTutorial.html
(luettu 23.5.2013)

Epic Games, Inc. 2013,Setting Up Vehicles. [verkkodokumentti]
http://udn.epicgames.com/Three/SettingUpVehicles.html
(luettu 23.5.2013)

Epic Games, Inc. 2013 ,Texture Support And Settings. [verkkodokumentti]
http://udn.epicgames.com/Three/TextureSupportAndSettings.html
(luettu 23.5.2013)

Sanders, Adrien-Luc 2013, What is a 3D Polygon? [verkkodokumentti]
http://animation.about.com/od/glossaryofterms/g/What-Is-A-3d-Polygon.htm
(luettu 23.5.2013)

Slick, Justin 2013, Anatomy Of A 3D Model. [verkkodokumentti]
http://3d.about.com/od/3d-101-The-Basics/a/Anatomy-Of-A-3d-Model.htm
(luettu 23.5.2013)

Skinner, Johnathan 2000, Mip-Mapping in Direct3D. [verkkodokumentti]
http://www.gamedev.net/page/resources/_/technical/directx-and-xna/mip-mapping-in-direct3d-r1233
(luettu 23.5.2013)

Youtube 2013. UDK Vehicle Wheeled FireAnim. [verkkodokumentti]
https://www.youtube.com/watch?v=-HfzaWzVsAY
(luettu 23.5.2013)

Kuvalähteet

Kuva 1: Wlogger 2011. Lamborghini Ankonian.
http://wlogger.com/lamborghini-ankonian/
(luettu 10.5.2011)

Liitteet

Video-tiedosto, jossa esitellään ajoneuvoa pelissä.

Ajoneuvon UDK-tiedostot pakattuna