3D-pelihahmo Unity-pelimoottorissa

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Mediatekniikan koulutusohjelma

Riihimäki, kevät 2015

Mika Virolainen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

TIIVISTELMÄ

RIIHIMÄKI
Mediatekniikan koulutusohjelma
Mediajärjestelmät

Tekijä Mika Virolainen **Vuosi** 2015

Työn nimi 3D-pelihahmo Unity-pelimoottorissa

TIIVISTELMÄ

3D-pelihahmon luominen ja käyttöönotto on laaja prosessi, joka vaatii useiden tekniikkojen ja ohjelmistojen osaamista. Mielestäni aiheesta puuttuu kattava selvitys, jonka avulla voidaan säästää aikaa ja ymmärtää prosessi kokonaisuutena. Opinnäytetyö keskittyy 3D-pelihahmon luomiseen ja käyttöön liittyviin työvaiheisiin, joissa tutkin mielestäni parhaat työtavat niiden selvittämiseen. Työ on hyvin käytännönläheinen, sillä siinä havainnollistetaan parhaat työkalut ja vaihtoehtoiset työtavat.

Opinnäyteyön tavoitteena oli tuoda esille eri työvaiheet, jotka vaativat erityishuomiota 3D-pelihahmon luomisessa. Työssä keskityttiin pääpiirteittäin pelihahmon luomiseen 3D-mallinnusohjelmassa, ja niiden käyttöönottoon liittyviin työvaiheisiin tarkemmin Unity-pelimoottorissa. Opinnäytetyö avaa myös 3D-peligrafiikan periaatteita ja sen vaatimuksia, jotka soveltuvat yleisesti pelinkehityksen grafiikkaan pelimoottorista riippumatta.

Opinnäytetyön käytännön osuudessa luotiin pelihahmo 3ds Max -ohjelmistossa ja otettiin käyttöön Unity-pelimoottorissa. Hahmolle asetettiin toiminnallisuus interaktiivisessa ympäristössä, mikä demonstroi saavutettua tavoitetta. Ominaisuuksia ovat esimerkiksi hahmon liikuteltavuus ja ympäristön interaktiiviset kohteet, joiden kanssa hahmo voi olla vuorovaikutuksessa. Opinnäytetyössä käytetyistä ympäristöistä huolimatta samat periaatteet pätevät myös muihin vastaaviin ympäristöihin.

Avainsanat 3D-mallinnus, pelihahmot, peligrafiikka,

Sivut 32 s. + liitteet 1 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

Riihimäki
Degree Programme in Media Technology
Multimedia production

Author Mika Virolainen **Year** 2015

Subject of Bachelor's thesis 3D game character in Unity game engine

ABSTRACT

The creation and use of a 3D game character is a broad process which requires understanding of multiple techniques and software. In my opinion the process lacks proper research documentation, which would save time and help in understanding the process as a whole. This thesis focuses on the creation of a 3D game character and the methods of bringing it to our game environment within the Unity game engine. This work is very practical, since its goal is to clarify the optimal tools and methods to achieve the final outcome.

The purpose of this thesis is to show the steps of the process that require special attention when developing a fully functional game character. This thesis mainly focuses on different important steps within the 3D modeling software and the game engine. The thesis also explains the basics of 3D graphics and what limitations it can have, which is important to understand when creating 3D game graphics.

The work process covers the creation of a game character within 3ds Max software and how to implement the fully created character in the Unity game engine. The character is set to have functionality within an interactable environment, which acts as a demonstration of the accomplished goal. Even though 3ds Max and Unity game engine were used in this thesis, these concepts work also in other software options.

Keywords 3D modelling, game characters, game graphics,

Pages 32 p. + appendices 1 p.

SISÄLLYS

TERMIT JA LYHENTEET

Asset	Assetit tarkoittavat kaikkia Unity-pelimoottorissa käytettäviä resursseja.
High poly	3D-kappale, jossa suuri määrä polygoneja.
Low poly	3D-kappale, jossa vähäinen määrä polygoneja.
Map / mapping	Materiaalikartta ja sen kartoitus sijaintiin.
Ngon	Polygoni, jossa on enemmän kuin neljä sivua.
Pelimoottori	Ympäristö, jossa on valmiit työkalut pelin tekemiseen.
Primitiivi	3D-ohjelmiston valmiita kappaleita voidaan kutsua primitiiveiksi, esimerkiksi laatikko- ja palloprimitiivi
Polygoni	3D-kappale koostuu polygoneista; polygoni koostuu vertexeistä, jotka muodostavat yhtenäisen pinnan.
Quad	Polygoni, joka on nelisivuinen.
Renderöinti	Renderöinti tarkoittaa sitä, että esimerkiksi 3D-ohjelma kokoaa 3D-kappaleen kaiken geometrisen datan, materiaali-informaation ja valon käyttäytymisen kokonaisuudeksi.
Rig	Kontrollijärjestelmä, jolla pystytään kontrolloimaan 3D-hahmoa mahdollisimman selkeästi animointia varten.
Skripti	Ohjelmakoodi, toteutettu jollain tietyllä ohjelmointikielellä.

Vertex	Vertexillä tarkoitetaan polygonin kulmapisteitä.

1 JOHDANTO

1.1 Lähtökohta

3D-pelihahmon luominen ja käyttöönotto pitää sisällään useita erilaisia vaiheita, joiden yhteensitominen voi olla ongelmallista. Tämän opinnäytetyön tarkoituksena on helpottaa hahmon luomiseen ja käyttöönottoon liittyviä haasteita. Toivon myös opinnäytetyöni avaavan ymmärrystä koko prosessin eri vaiheista, joiden avulla esimerkiksi graafikko pystyy ymmärtämään luomansa grafiikan käyttöönottoon liittyvät vaiheet ja ongelmat.

Mielestäni aiheesta puuttuu kattava, mutta kompakti ohjeistus, jolla käyttäjä pääsee alkuun grafiikan tuotannossa. Jotta opinnäytetyöni oleellisin tarkoitusperä toteutuisi, oletan lukijan ymmärtävän 3D-grafiikan tuotannon ja Unity-pelimoottorin perusteet.

1.2 Ympäristö

Tässä opinnäytetyössä käytetään esimerkkinä Unity-pelimoottoria, mutta tämä ei kuitenkaan ole ainoa varteenotettava vaihtoehto itsenäiselle pelinkehittäjälle, jolla on tiukka tai olematon budjetti. Markkinoilta löytyy useita vaihtoehtoja pelimoottorin valintaan.

Alalla ei ole mitään tiettyä pelimoottoria, jota kaikki kehittäjät käyttävät. Unityn lisäksi muita suosittuja vaihtoehtoja ovat Cry Engine, Unreal Engine ja Source Engine. Pelimoottoria valitessa tulee punnita, mikä vaihtoehdoista sopii omiin taitoihin ja projektin laajuuteen parhaiten.

Cry Engine on Crytekin kehittämä pelimoottori ja se on saatavilla 9,90 euron kuukausihintaan, jolla saa käyttöön sen kaikki ominaisuudet ja lisenssissä ei tarvitse maksaa rojalteja. Moottoriin voi hakea täyttä lisenssiä, jossa on täysi lähdekoodi sekä tuki kehittäjiltä. (Cry Engine 2015.)

Unreal Engine 4 on Epic Gamesin kehittämä pelimoottori, joka on julkaistu kaikille ilmaiseksi kaikkine ominaisuuksineen ja kohdealustatukineen. Kehittäjän tulee maksaa 5 % rojaltimaksuja Epic Gamesille vasta, kun tuote on tuottanut 3000 $ per vuosineljännes. Kehittäjän ei tarvitse maksaa rojalteja elokuvaprojekteista tai rakennus- ja konsultointiprojekteista kuten arkkitehtuuri, simulaatio ja visualisointi. (Unreal Engine 2015.)

Unity-pelimoottorista on saatavilla ilmainen versio, jossa on kaikki tärkeimmät moottorin ominaisuudet, ilman rojaltimaksuja. Unityn maksullinen versio pitää sisällään lisäominaisuuksia, jotka helpottavat peliprojekteissa työskentelyä. Lisäominaisuuksia ovat esimerkiksi tiimilisenssi, joka mahdollistaa tiimin sisäisen projektin jaon ja päivityksen vaivattomasti.

Unity on suuren suosion saavuttanut pelimoottori ja kehitysympäristö, joka pitää sisällään vahvan tuen 3D-grafiikalle. Yksi Unityn vahvuuksia on erittäin laaja alustatuki, mikä mahdollistaa pelin kehityksen usealle alus-

talle yhtäaikaisesti. Unity on kuitenkin aloittelevalle pelinkehittäjälle helpommin lähestyttävä vaihtoehto halpojen lisenssien ja helpon käyttöliittymänsä vuoksi. (Unity 2015a.)

1.3 Työn tavoite

Opinnäyteyöni tavoitteena on tuoda esille eri työvaiheet, jotka vaativat erityishuomiota 3D-pelihahmon luomisessa ja käyttöönotossa pelimoottorissa. Työssä keskitytään pääpiirteittäin pelihahmon luomiseen 3D-mallinnusohjelmassa ja käyttöönottoon liittyviin työvaiheisiin Unity-pelimoottorissa. Opinnäytetyöni avaa myös yleisesti 3D-peligraafikan periaatteita ja sen vaatimuksia.

Opinnäytetyössä luon pelihahmon 3ds Max -ohjelmistossa ja otan sen käyttöön Unity-pelimoottorissa. Luon pelihahmolle myös yksinkertaisen ympäristön ja lisään demoon yksinkertaisia pelillisiä ominaisuuksia. Ominaisuuksia ovat esimerkiksi hahmon liikuteltavuus ja ympäristön interaktiivisuus, jonka kanssa pelaaja voi vuorovaikutuksessa.

1.4 Työvaiheet

Pelihahmon luomisessa on useita vaiheita, jotka käyn läpi käytännön osuudessa. 3D-tuotannon puolella vaiheet voidaan jakaa 4 eri osa-alueeseen, kuten kuvassa 1. 3D-tuotannon puolella käsiteltäviä työvaiheita ovat mallinnus, UVW-kartat ja materiaalit, riggaus ja animointi, sekä ulosvienti ohjelmasta. Näissä työvaiheissa ei syvennytä selvittämään vaihe vaiheelta koko prosessia, vaan tuodaan esille jokaisessa vaiheessa oleelliset tekijät Unity-pelimoottoriin viemistä varten.

Kuva 1. 3D-hahmon tuotanto

Unity-pelimoottorin puolella käsitellään 3D-grafiikan käyttöönoton vaiheita ja hahmoanimaatioiden toiminnollisuuden hyödyntämistä, jonka vaiheita esitetty kuvassa 2. Opinnäytetyö pyrkii erityisesti avaamaan Unityn Mecanim-animaatiojärjestelmän hyödyllisyyttä lukijalle.

Kuva 2. 3D-grafiikka Unity-pelimoottorissa.

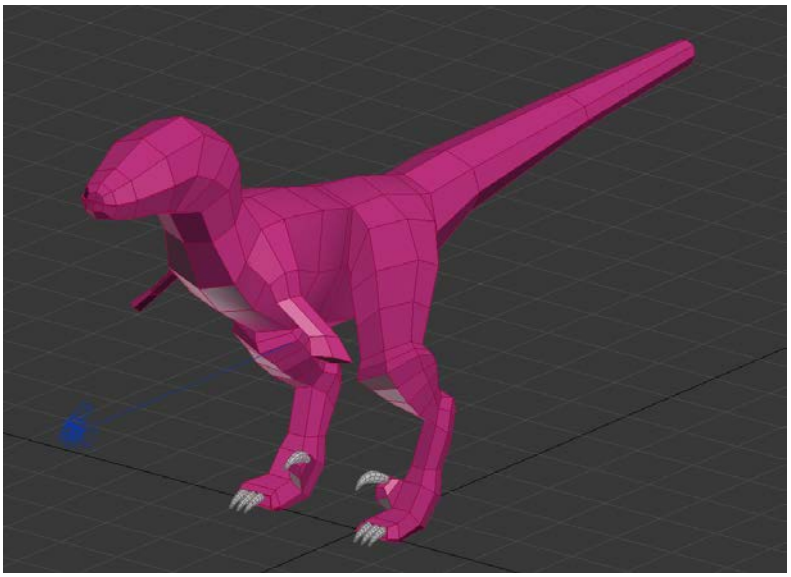
2 3D PELIGRAFIIKAN PERIAATTEET

2.1 3D-grafiikka

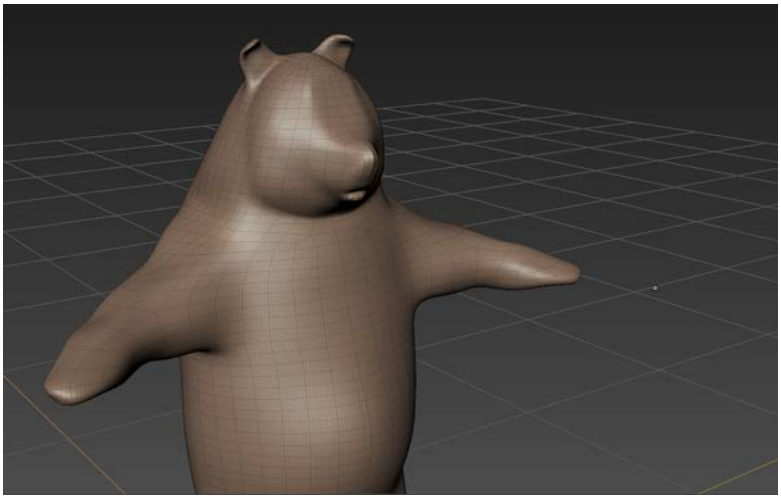
3D-grafiikassa pyritään jäljittelemään ympäristöä ja esineitä digitaalisesti; lopputulosta voidaan käyttää esimerkiksi mainos-, elokuva- ja pelituotannossa. 3D-kappale koostuu monikulmioista, eli polygoneista. Polygonien määrä kasvaa samassa suhteessa kuin yksityiskohtien mallintaminen kappaleeseen. Mitä enemmän 3D-ympäristössä on polygoneja, sitä enemmän se vaatii suoritustehoa tietokoneelta. (Totten 2012, 12.)

Mallinnuksen ja visualisoinnin jälkeen 3D-grafiikka renderöidään. Renderöinti tarkoittaa sitä, kun tietokone kokoaa yhteen 3D-kappaleen kaiken geometrisen datan, materiaali-informaation ja valon käyttäytymisen. Tästä muodostuu kokonaisuus, johon on yhteenlaskettu kaikki ominaisuudet, eli loppurenderöinti. Renderöintiprosessin raskaus ja kesto riippuvat tietokoneen tehoista, sekä koottavan informaation määrästä. (Wikipedia 2015a.)

Erityisesti peligrafiikkaa tehdessä 3D-grafiikka tulee optimoida mahdollisimman kevyeksi, jotta saadaan lopputuotteen laiterajoitukset mahdollisimman matalaksi, kuitenkaan karsimatta liikaa visuaalisista yksityiskohdista. Peleissä käytetään usein matalia polygonimääriä yksittäisissä objekteissa. Kuvassa 3 nähdään low poly -dinosaurus, joka materiaalien avulla voitaisiin saada yksityiskohtaiseksi matalasta polygonimäärästä huolimatta. Pelin aikana lasketaan 3D-mallit, tekstuurit, valot ja mahdollinen interaktiivisuus pelimaailmassa reaaliajassa freimi kerrallaan. Suurin vaikuttaja 3D-grafiikan raskauteen on polygonien määrä. Kuvassa 4 näkyy esimerkkinä korkeampi polygonimäärä hahmossa, jossa korostuu hahmon geometrian pehmeys, jota on hankalampi saavuttaa low poly -hahmossa. (Totten 2012, 14.)



Kuva 3. Low poly -malli dinosauruksesta, jossa on 1 646 polygonia ja 1 049 vertexiä.



Kuva 4. High poly -malli karhusta, jossa on 180 000 polygonia.

2.2 Mallinnuksessa huomioitavaa

On tärkeää, että hahmon polygonit on rakennettu oikeaoppisesti, jotta se voidaan animoida sulavasti. Polygonikokonaisuutta kutsutaan 3D-topologiaksi. Mallintaessa tulee varmistaa, että polygonit ovat nelisivuisia, eli quadeja. 3D animaatio-ohjelmat osaavat vääntää quad-topologiaa oikein animaatioita tehdessä, kun taas kolmiot (polygonit, joissa on kolme sivua) saattavat vääristyä väännettäessä. Polygonit, joissa on enemmän kuin neljä sivua tunnetaan nimellä ngon ja niitä tulisi välttää yleisesti mallintaessa, sillä ne vääristyvät animoidessa. Erityisesti hahmon raajojen ja kehon taipumiskohdat vaativat polygoneja, jotta animaatio olisi mahdollisimman sulavaa. (Totten 2012, 15.)

Projektin kattavalla suunnittelulla saadaan määriteltyä valitut kohdealustat, jotka asettavat rajoitteet grafiikalle. Käyttötarkoitus ja kehityksen kohdealusta määrittävät, kuinka monta polygonia voidaan käyttää ilman, että lopputuotteen suorituskyky kärsii. Polygonien suositusmäärät vaihtelevat projektin kohdelaitteiston ja vaaditun laadun mukaan, mutta yleisesti PC-peleissä ideaalinen määrä olisi noin 1500 - 4000 polygonia per kappale. Mobiililaitteille ideaalinen polygonimäärä olisi noin 300 - 1500 polygonia per kappale. Korkeamman budjetin peleissä tällä hetkellä polygonimäärät hahmoissa vaihtelevat 5000 - 7000 kolmiopolygonia per hahmo. Kyseiset arvot ovat kuitenkin vain arvioita, joka vaihtelevat pelitilanteen mukaan. (Unity Manual 2015a.)

Oli projektin tavoite mikä tahansa, jo mallinnusvaiheessa on tärkeää huomioida kappaleiden oikea skaala suhteessa pelimaailman ympäristöön ja määrittää 3D-ympäristön mittasuhteet oikeiksi. Tämän avulla peliobjektit tulevat käyttäytymään valitussa pelimoottorissa jo oletuksena mahdollisimman luonnollisesti valon ja pelimoottorin fysiikoiden suhteen. Esimerkiksi oikein mitoitettu kappale on suunnitellusti helpompi sijoittaa peliympäristöön, ja se myös käyttäytyy realistisemmin peliympäristön fysiikassa, kuin väärissä mitoissa tuotu kappale. (3ds Max nda; Unity Manual 2015b.)

2.3 Materiaalit ja tekstuurit

3D-objektille voidaan määritellä ulkoasu, eli pintamateriaali. Materiaali määrittelee miten valo käyttäytyy osuessaan kyseiseen kappaleeseen, esimerkiksi minkä värin valo heijastaa kappaleesta. Materiaalissa voi olla määritetty myös 2D-kuva, eli niin sanottu tekstuuri. Tekstuuri asetetaan kappaleen pintaan haluttuun sijaintiin.

Tekstuurit ovat kuvatiedostoja, jotka liitetään kappaleen pintaan. Tekstuureilla voidaan saavuttaa illuusio hyvin tarkoista pintakuvioista, kuitenkaan lisäämättä kappaleeseen enempää polygoneja. Materiaalit ovat erityisen suuressa roolissa muotojen ollessa yksinkertaisia (low poly).

Diffuse maps

Tunnetaan myös nimellä Color Maps. Diffuse maps tekstuurit antavat 3D-kappaleen pinnalle värin ja erilaisia ominaisuuksia, kuten vaatekappaleita.

Bump Maps

Saavat täysin litteät pinnat näyttämään epätasaisilta. Yleensä mustavalkoisia kuvia, joissa tummemmat värit kuvastavat syviä uurteita ja vaaleammat pinnasta kohoavia piirteitä.

Normal maps

Bump map -tekstuurin kaltainen, mutta vieläkin tätä voimakkaampi. Normal mappingillä voidaan helposti luoda il-

luusio tarkoista yksityiskohdista riippumatta kappaleen todellisista polygonimääristä.

Alpha maps

Alpha maps -tekstuureilla voidaan vaivattomasti luoda erilaisia muotoja yksinkertaisista objekteista vain poistamalla kappaleen näkyvyys eri kohdista. Tällä tavalla voidaan esimerkiksi luoda puunlehtiä hyvin pienellä polygonimäärällä.

Reflection maps

Tällä tekstuurilla voidaan määritellä kappaleen heijastavuus tarkasti. Kappale vaikuttaa heijastavan ympäristöä, mutta todellisuudessa heijastus tulee ennalta määritetystä kuvasta.

Illumination maps

Specular maps -tekstuurien tavoin nämä kuvat määrittävät, mitkä kappaleen alueet hehkuvat ympäristön valoista riippumatta.

Specular maps

Nämä tekstuurit kontrolloivat kappaleen pinnan kiiltävyyttä.

(Totten 2012, 17-20.)

Kuva 5. Bump ja normal mapping

2.4 Valaistus

Valaistus on oleellinen osa 3D-grafiikkaa ja sen luonnollinen käyttäytyminen vaikuttaa kokonaisuuden yleisilmeeseen merkittävästi. 3D-kappaleet vaativat valoa näkyäkseen, aivan kuten oikeassa maailmassakin. Useimmat 3D-ohjelmat sisältävätkin valosysteemejä, jotka imitoivat oikean valon käyttäytymistä.

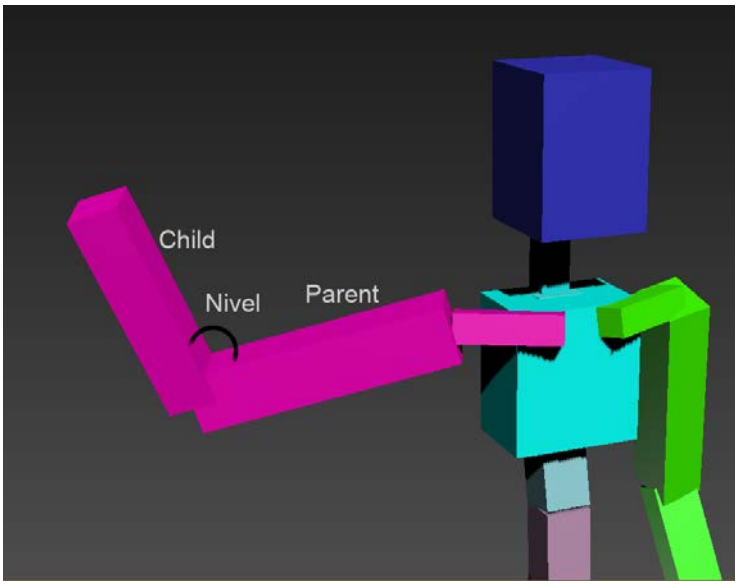
Mallinnettaessa pelimoottoriin vietävää grafiikkaa valaistus toteutetaan vasta pelimoottorin omilla valaistusjärjestelmillä. Hyvän työskentelytavan mukaista on kuitenkin varmistaa materiaalien käyttäytyminen etukäteen 3D-ohjelman omilla valoilla, jotta saadaan suuntaa-antava tulos siihen, miltä grafiikka tulee näyttämään pelimoottorissa. (Digital-Tutors Blog 2014a.)

2.5 Hahmon animaatiokontrolli ja sen liittäminen geometriaan

Pelihahmoille luodaan luuranko, jonka avulla animointi on selkeää ja helpompaa. Unity-pelimoottori vaatii kaikkien animaatioiden pohjautuvan rigeihin, joten tämä työvaihe on oleellinen Unity-pelimoottoria varten animoitaessa.

3D-hahmolle luodaan animaatiokontrolliksi, eli rigiksi näkymätön luurakenne, jonka myötä jokainen polygoni on liitettynä johonkin rakenteen luuhun. Kun liikutamme hahmon luurangon luuta, siihen liitännäiset polygonit liikkuvat sen mukana. Tämän avulla luurangon luiden liikuttelu vaikuttaa hahmomme geometriaan, jolloin animaatioista saadaan pienellä vaivalla sulavaa. Tämä mahdollistaa hahmon animoimisen ilman, että tarvitsee liikutella ja muokata jokaista polygonia yksitellen. (Digital-Tutors Blog 2014b.)

Animaatiokontrollin luut muodostavat hierarkian, jossa luut ovat toistensa parent- ja child-luita kuten kuvassa 6. Kinemaattisuus tietokoneanimaatoissa voidaan jakaa kahteen osaan: forward kinematics ja inverse kinematics. Forward kinematics perustuu rungon, tai hierarkiassa parent-luun manipulaatioon, jolloin sen child-luut liikkuvat sen mukana. Inverse kinematics perustuu hierarkian päädyn, eli esimerkiksi pelkästään child-luun liikutteluun ja parent liikkuu mukana ennalta määritetyn algoritmin mukaan; tätä kutsutaan inverse kinematiikaksi. (Autodesk 3ds Max Manual 2013.)



Kuva 6. Hahmon luurangon hiearkia.

Luurakenteen ja hahmon geometrian polygonien liittämistä toisiinsa kutsutaan skinnaukseksi. Skinnauksessa määritellään kunkin luun vaikutusvoima ja alue hahmon geometrian eri osioihin. Esimerkiksi hahmon kyynärvarren luu vaikuttaa täysin hahmon käsivarteen, mutta kyynärpään alueella vain hieman. Tällä tavalla hahmon liikuttelu saa pehmeän ja luonnollisemman näköisen liikkeen. (3ds Max tutorials nda.)

Jos useammalla hahmolla on samanlainen luurakenne, voidaan animaatioita jakaa niiden välillä. Tämä mahdollistaa animaatioiden kierrätyksen hahmosta toiseen ja helpottaa työprosessia. Animaatioita voidaan hieman muokata, jotta jokaisella hahmolla on hieman omantyylinen tapa liikkua, kuitenkaan luomatta samanlaista animaatiota jokaiselle hahmolle alusta loppuun. (Unity Manual 2015c.)

2.6 Animointi

Animaatiot voidaan toteuttaa valitussa 3D-mallinnusohjelmassa, kunhan kyseisestä ohjelmasta pystytään viemään 3D-kappaleet Unityn tukemassa muodossa ulos. Tuetut tiedostomuodot näkyvät taulukossa 1. Tyypillisessä pelianimaatiossa viedään rigattu hahmo ja animaatiot pelimoottoriin, jossa ne voidaan aktivoida skriptillä. Skriptit määrittelevät milloin hahmon animaatiota käytetään.

Taulukko 1. Unityn tukemat tiedostomuodot. (Unity Manual 2015j.)

Tuetut export-tiedostot	Tuetut 3D-ohjelmistoformaatit
.fbx	Max (.max)
.dae	Maya
.3ds	Blender
.dxf	Cinema4D

.obj	Modo
	Lightwave
	Cheetah3D

Animointi tapahtuu avaintamalla hahmon luurangon, eli Rigin, muutoksia. Kun luurangon luut liikkuvat, liikkuvat hahmon polygonit sen vaikutusalueiden asettamien rajojen mukaan. Jos käytetään 3ds Maxin Character Animation Toolkit -järjestelmää, animaatiot tallentuvat eri CAT-järjestelmän tasoille, joita voidaan sekoittaa keskenään tarvittaessa. Yleinen hahmoanimaatio on idle-animaatio, joka lähtee pyörimään pelissä automaattisesti yleensä hahmon ollessaan ”käyttämättömänä” paikallaan tietyn ajan. Animaatio voi olla esimerkiksi hahmon paikallaan seisomista, mutta kevyesti heiluen hengityksensä tahdissa. (Autodesk Help 2015; Wikipedia 2015b.)

Animoija voi tehdä kaikki tarvittavat animaatiot yhteen Unityyn vietävään tiedostoon (esimerkiksi .fbx), tai tehdä jokaisesta tarvitusta animaatiosta oma tiedostonsa. Mikäli kaikki animaatiot päätetään pitää samassa tiedostossa, voidaan ne Unityssä vaivattomasti pilkkoa erillisiksi animaatioiksi. Yhden tiedoston viemistapa auttaa myös tiedostojen hallinnassa, mutta molemmissa tavoissa on puolensa. Unity vaatii animaatioiden olevan tehty luurankoon rigiä hyödyntäen. (Digital-tutors 2014.)

3 UNITY-PELIMOOTTORI KEHITYSYMPÄRISTÖNÄ

Unity on suuren suosion saavuttanut pelimoottori ja kehitysympäristö, joka pitää sisällään vahvan tuen 3D-grafiikalle. Yksi Unityn vahvuuksia on erittäin laaja alustatuki, mikä mahdollistaa pelin kehityksen usealle alustalle yhtäaikaisesti. Taulukossa 2 nähdään Unity-pelimoottorin laaja alustatuki. (Unity 2015a.)

Taulukko 2. Unityn alustatuki.

Työpöytäalustat	Mobiilialustat	Konsolialustat
Mac	iOS	Xbox 360
Windows	Windows Phone 8	Xbox One
Linux	Android	Wii U
Web	BlackBerry 10	PlayStation 3
	PlayStation Mobile	PlayStation 4
	Tizen	PlayStation Vita

Yksi Unityn parhaita ominaisuuksia on sen monipuolisuus. Verrattuna esimerkiksi Unreal Engineen, joka perustuu ensimmäisen persoonan pelien tekemiseen, Unityssä on valmiudet hyvin erilaisten pelien vaivattomaan tekemiseen, ilman itse Unity-pelimoottorin muokkaamista. Unityssä

voidaan käyttää kolmea eri yleisesti tunnettua ohjelmointikieltä pelinkehityksessä: Javascriptiä, C#- ja Boo-kieltä. Tämä tekee pelin kehittämisestä Unityllä helposti lähestyttävää, sillä se tukee yleisesti käytettyjä ohjelmointikieliä. (Unity Blogs 2014.)

3.1 Vienti Unityyn

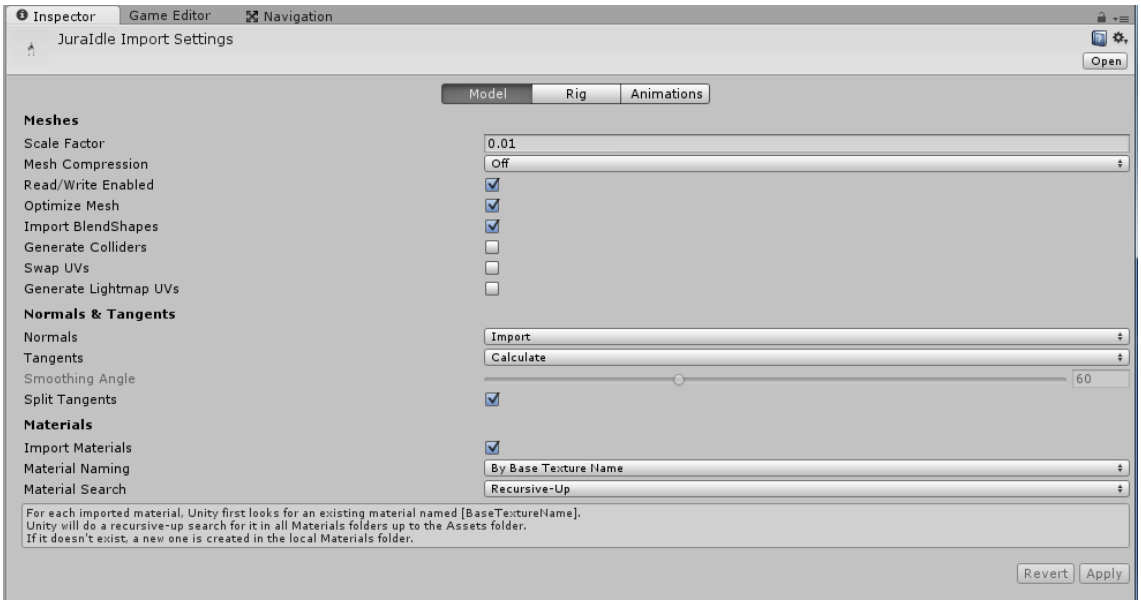
3D-malleja voidaan viedä Unityyn usealla eri tavalla. Unity tukee yleisimpiä 3D-export -tiedostoformaatteja, kuten .fbx- ja .obj-tiedostoja. Tämän lisäksi Unity osaa lukea joidenkin 3D-ohjelmistojen natiiveja tiedostotyyppejä suoraan, kuten Blender, 3ds Max, Maya, Cinema 4D, Cheetah 3D ja Modo. Natiivien tiedostotyyppien lukemiseksi työtietokoneella tulee olla kyseinen ohjelmisto asennettuna. (Unity Manual 2015d.)

Tässä opinnäytetyössä käytettiin .fbx-formaattia, mutta myös .max-tiedostot olisivat olleet sopivia. Fbx-tiedostoissa on mahdollisuus määrittää kattavien asetusten avulla, mitä viedään ulos ja tiedostokoot ovat .max-tiedostoon verrattuna huomattavasti optimaalisempia.

3.2 3D-grafiikan tuominen Unity-pelimoottoriin.

Unity-pelimoottori perustuu asset-työskentelyyn. Assetit tarkoittavat kaikkia resursseja, joita projektissa on käytetty; kaikkea grafiikasta koodin pätkiin. Assetit liitetään peliobjekteihin ominaisuuksina. Esimerkiksi voimme luoda tyhjän peliobjektin pelimaailmaamme, johon liitämme 3D-mallimme assetin grafiikaksi ja joka näkyy peligrafiikkana. UVW-kartta kulkeutuu kappaleen mukana 3D-ohjelmasta Unityyn, ja luotu tekstuuri asetetaan uudemman kerran pelimoottorissa kappaleeseen materiaalikomponenttina. (Unity Manual 2015e.)

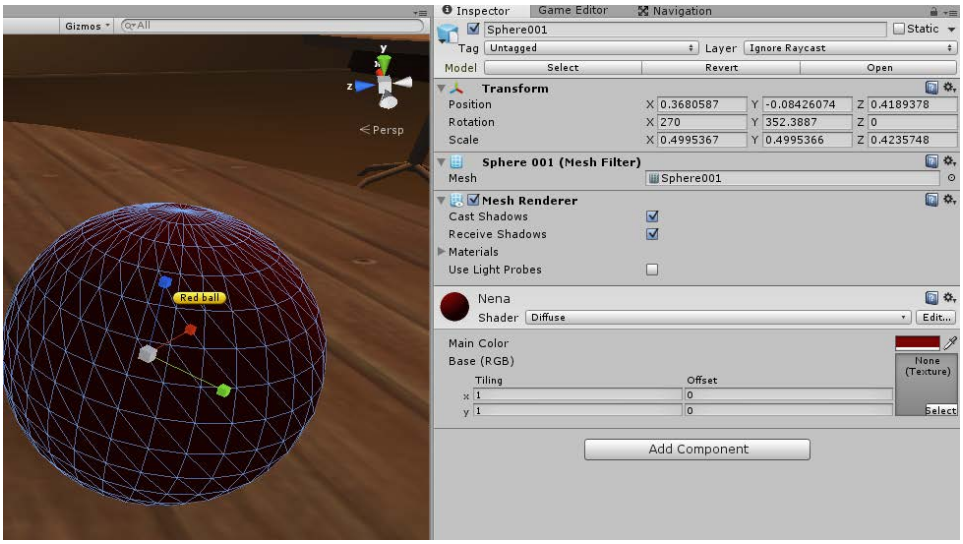
Kun Unityyn tuodaan uusi graafinen asset, voidaan tarkastella sen tuontiasetuksia projekti-ikkunassa. Unityn tuontiasetukset, jotka näkyvät kuvassa 7, on jaettu kolmeen pääkategoriaan: model, eli kappaleen geometrinen malli, rig, eli luuranko animaatiolle ja animation, eli kappaleen animaatioasetukset.



Kuva 7. 3D-grafiikan tuontiasetukset

3.3 3D-graafiikan käyttö Unityssä

Unityn toiminnallisuus perustuu vahvasti graafisten elementtien ympärille. Toiminnallisuus liitetään graafisiin elementteihin komponentteina. Peliobjektissa voi olla esimerkiksi transform- (sijainti), materiaali- tai skriptikomponentteja, kuten kuvan 8 pallossa on. Transform-komponentti on jokaiselle peliobjektille pakollinen, sillä se määrittää peliobjektin sijainnin pelimaailmassa. Komponentit ovat ikään kuin kappaleen sisältämiä ominaisuuksia ja muuttujia. (Unity Manual 2015f.)



Kuva 8. Peliobjektin materiaalikomponentti, jossa on punainen diffuusi väri.

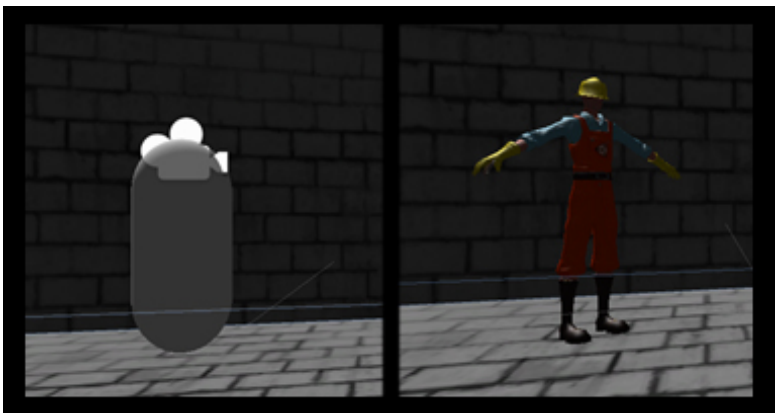
3.4 Collider-komponentti

Collider-komponentti peliobjektissa mahdollistaa tämän reagoimisen muiden collider-komponentin sisältävien objektien kanssa. Collider-komponentit ovat Unityyn sisäänrakennettu ominaisuus, jolla voidaan havaita kappaleiden osumia toisiinsa. Esimerkiksi collider-komponentti pelimaailman rakenteissa ja pelihahmo-objektissa estää pelihahmoa kulkemasta tämän lävitse tai putoamasta painovoiman vaikuttaessa pelimaailman ulkopuolelle. Collidereilla voidaan myös laukaista tapahtumia, kutsua skriptejä tai Mecanim-siirtymiä. Esimerkiksi jos pallo osuu pelaajaan, pelaaja menettää pisteen. (Tuts+ 2011.)

Colliderit voivat olla erilaisia muotoja ja ne näkyvät Unity-moottorinäkymässä vihreinä viivoina kappaleen ympärillä. Muotoja voivat olla esimerkiksi pallo-, kapseli- ja laatikko-collider. Vaihtoehtoisesti käytettävissä on myös mesh collider, joka luo osumapinnan muodon täysin identtiseksi pelikappaleen kanssa. Tämä on hyvin yksityiskohtainen collider, joka voi olla koneelle erittäin raskas laskea, riippuen kappaleen yksityiskohtaisuudesta. (Unity 2015b.)

3.5 Hahmon kontrolli

Hahmon liikkumisen kontrollointi tarkoittaa käytännössä näppäinyhdistelmien liittämistä sijaintikomponentin eri akseleiden muutokseen. Unityn mukana tulee valmiita hahmokontrolli-skriptejä. Vaihtoehtoina ovat ensimmäisen persoonan tai kolmannen persoonan kontrollerit, jotka näkyvät kuvassa 9.



Kuva 9. Unityn valmiit hahmokontrollerit

Ensimmäisen persoonan kontrollerissa hahmon grafiikkana toimii vain kapseliprimitiivi, jossa kamera asetettu ylös hahmon silmien korkeudelle. Tämän kontrollerin tarkoituksena on kuvastaa näkymää hahmon silmistä. Kolmannen persoonan kontrollerissa sen sijaan kamera asettuu sopivan

etäisyyden päähän hahmon taakse, jossa luonnollisesti hahmon grafiikka on kameran näkymässä. (Unity Manual 2015g.)

Kontrolleri mahdollistaa loppukäyttäjän eri toimintojen toteuttamisen. Toiminnot voivat olla esimerkiksi esineen poimiminen, joka kontrollerin avulla laukaisee poimimisanimaation ja esineen häviämisen peliympäristöstä.

3.6 Adventure Creator -plugin

ICEBOX Studiosin Adventure Creator on Unity-pelimoottoriin ostettava laajennus, joka auttaa valmiin pohjarakenteen luomisessa erilaisiin seikkailupeliformaatteihin. Se mahdollistaa työajan optimoinnin, sillä suurin osa pelillisten ominaisuuksien, kontrollien ja hahmon linkitysten rakenteista on jo olemassa. (Adventure Creator 2015.)

Käytin omassa projektissani Adventure Creator -laajennusta, sillä halusin keskittää aikani peligrafiikkaan, mutta silti saada toimivan testikentän aikaiseksi. Vaihtoehtoisesti olisin voinut muokata Unity-pelimoottoria enemmän seikkailupeleihin soveltuvaksi, mutta tämä olisi vienyt suunnattomasti aikaa ja resursseja.

Unity-pelimoottorista löytyy valmiita hahmokontrollereita, mutta ne ovat hyvin yksinkertaisia pohjia, joiden päälle tulisi kehittää lisää ominaisuuksia, jotta niiden toiminta olisi monimuotoista.

4 MECANIM-ANIMAATIOJÄRJESTELMÄ

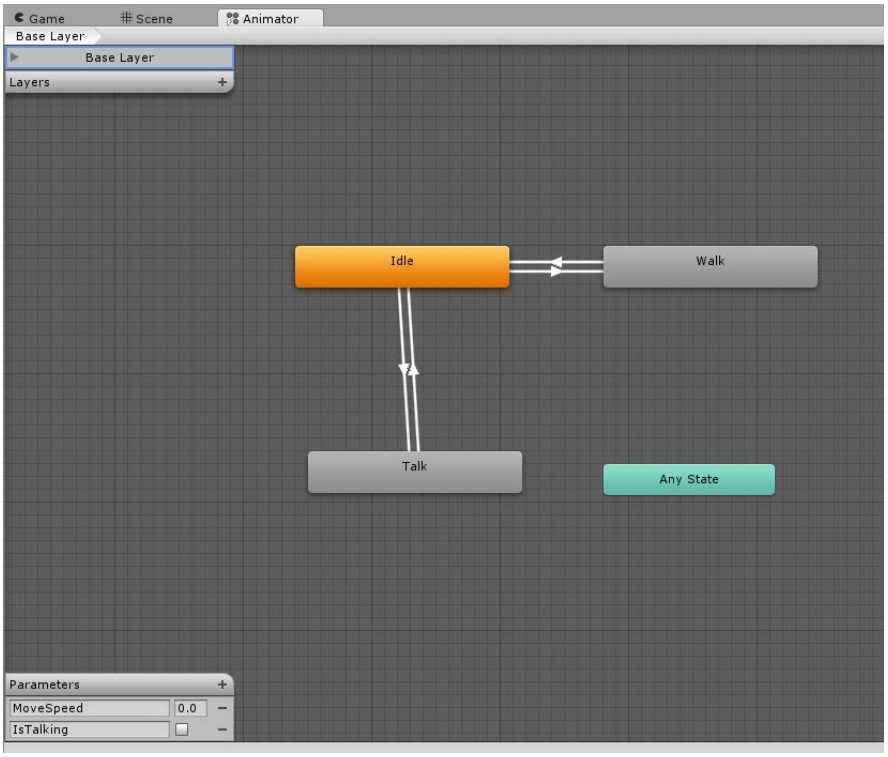
4.1 Mecanimin periaatteet

Mecanim on Unityn 4 versiossa saapunut sisäänrakennettu joustava ja voimakas animaatiojärjestelmä. Sillä voidaan joko luoda animaatioita tai yhdistää muualta tuotuja animaatioita toimivaksi kokonaisuudeksi. Tuodut animaatiot voidaan asetella node-pohjaiselle käyttöliittymälle ja linkittää toisiinsa tietyin ehtolausekkein. (Unity blogs 2012.)

Ehtolauseke, tai state eli tila, voi olla esimerkiksi hahmon haavoittuminen, jolloin hahmon liikkumisanimaatioksi voi automaattisesti vaihtua hidas ja huojuva kävely. Tämä tekee animaatioiden ja niiden siirtymien hallinnasta erityisen tehokasta ja selkeää. (Unity blogs 2012.)

Node-pohjainen käyttöliittymä näyttää kunkin animaation omana kappaleenaan, josta voidaan luoda yhteyksiä toisiin animaatioihin, kuten kuvassa 10. Tämänkaltainen havainnollistava käyttöliittymä helpottaa erityisesti monimutkaisemmin animoidun hahmon hallintaa. Mecanim-

animaatiojärjestelmä on erittäin joustava animaatioiden suhteuttamisessa toisiinsa. Järjestelmällä pystytään myös korvaamaan tietty node täysin toisella animaatiolla ja muutos tapahtuu saman tien. (Unity blogs 2012.)



Kuva 10. Animator (Mecanim)-ikkunan Node-pohjainen käyttöliittymä.

4.2 Avatar ja Animator controller

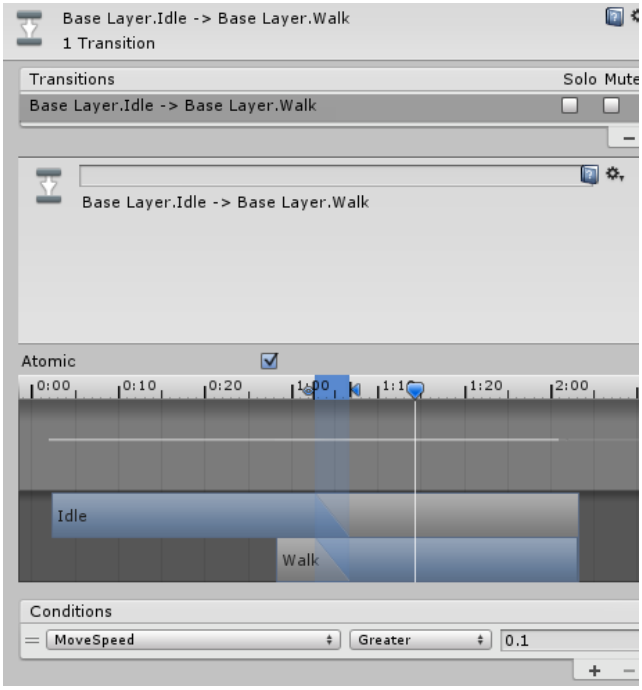
Mecanim-järjestelmä soveltuu erityisesti humanoid-hahmojen animaatioiden hallintaan. Koska useimmat pelihahmot ovat humanoid-pohjaisia, Unityssä on kehitetty niitä varten kattavat työkalut. Humanoid-hahmoilla on hyvin vertailtavissa olevat luurankorakenteet, kuten raajat, pää ja keho, joiden pohjalta niiden animaatioita voidaan soveltaa animaatiokontrolliluurangosta toiseen. Oleellinen osa animaatioiden kohdentamisessa luurankojen välillä on määrittää yksinkertaisen humanoid-hahmon luurakenne oman hahmon luurakenteeseen. Tätä liittämistä kutsutaan nimikkeellä Avatar, joka määrittää luurangon, joihin tietyt animaatiot perustuvat. Avatar luodaan oletuksena importatun tiedoston pohjalta. (Unity Manual 2015h.)

Animator-kontrolleri vaaditaan animaatioiden liittämiseen luurankoon, ja se mahdollistaa animaatioiden vaihtelun saumattomasti pelitilanteen mukaan. Kaikki peliobjektit, joissa on avatar vaativat myös Animator Controller -komponentin, joka toimii linkkinä hahmon ja sen käyttäytymisen välillä. (Unity Manual 2015h.)

4.3 Mecanim State Machines

Animaatioiden linkitys tarkoittaa niiden yhdistämistä toisiinsa, jolloin niiden välille syntyy siirtymä. Siirtymään asetetaan tietyt ehdot kyseisen siirtymän aktivoitumiselle. Esimerkiksi Move Speed -parametrin ollessa enemmän kuin 0.1, aktivoituu siirtymä idle-animaatiosta kävely-animaatioon. (Unity Manual 2015i.)

Siirtymien nopeutta ja tapaa pystytään kontrolloimaan Mecanimin Transition-välilehdessä. Käyttäjä pystyy määrittämään, missä vaiheessa ja kuinka hitaasti animaatiot sekoittuvat keskenään, kuten kuvassa 11 näkyy. Parhaiden asetusten saamiseksi, transition-välilehden alapuolella näkyy preview-animaatio, jonka avulla pystytään näkemään säädöt saman tien käytännössä. Tämä mahdollistaa tehokkaan työskentelytavan ja animaation laatuun pystytään vaikuttamaan myös animoinnin jälkeen helposti tällä tavalla. (Unity Manual 2015i.)



Kuva 11. Siirtymien hallintaa animaatioiden välillä.

4.4 Legacy-järjestelmä ennen Mecanim-järjestelmää

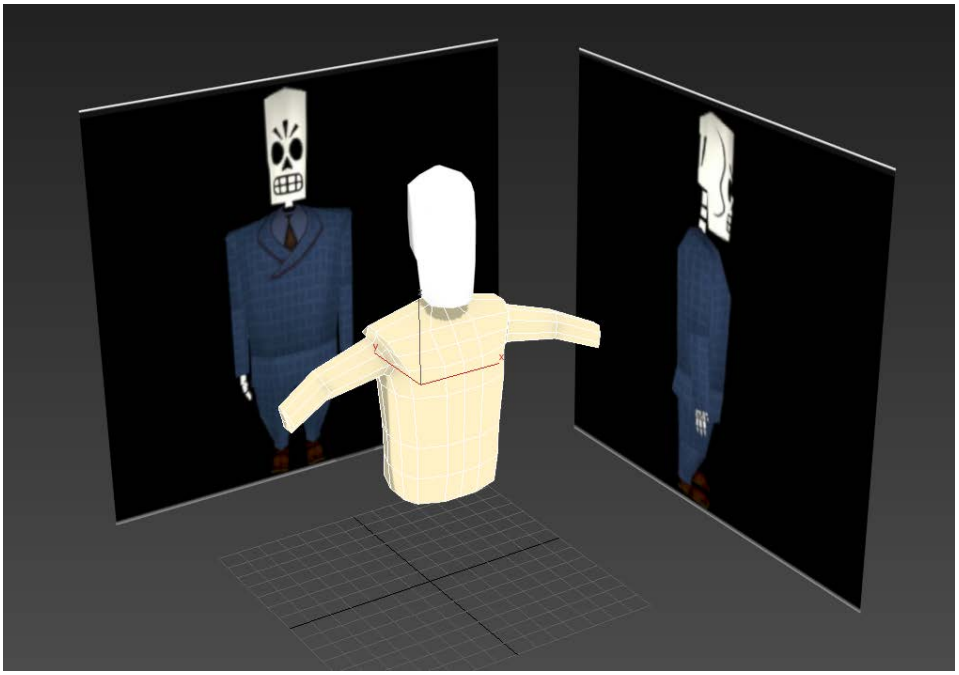
Ennen Mecanimin saapumista Unityssä oli käytössä yksinkertainen Legacy-järjestelmä. Legacy pystyi myös sekoittamaan animaatioita keskenään, mutta tämä vaati paljon työtä ja koodia onnistuakseen. Mecanim perustuu enemmän visuaaliseen työskentelyyn, jossa työvaiheita on helpotettu valmiilla säätimillä. Taaksepäin yhteensopivuuden takia Legacy-järjestelmä on vieläkin käytettävissä Unityssä. (Unity Forum 2013.)

5 PELIHAHMON LUOMINEN

5.1 Grafiikan luonti 3ds Maxissa

Tavoitteenani tässä vaiheessa oli lähteä luomaan yksinkertaista hahmoa, jonka halusin animoida tekemään pelidemolle tarpeelliseksi näkemäni toiminnot. Tämä työvaihe vaati ymmärrystä 3D-grafiikasta ja tuntemusta 3ds Max -ohjelmistosta, mutta samat työtavat pätevät myös muihinkin 3D-mallinnusohjelmistoihin.

Pelihahmon mallinnus aloitettiin järjestämällä referenssikuvat hahmoa varten paikoilleen kuten kuvassa 12. Kuvat hahmostani olivat edestä ja sivulta, minkä pohjalta mallinnus oli suunnitelmallisempaa ja tarkempaa. Referenssikuvat ovat oleellinen apuväline mallinnettaessa jo olemassa olevaa konseptia. Niiden avulla pystytään mahdollisimman tarkasti noudattamaan suunniteltua konseptia. Kuvien linjaamisessa ja mitoittamisessa käytin Adobe Photoshop -ohjelmiston guide-apuviivoja. Loin aluksi kaksi tyhjää plane-primitiiviä, joilla oli sama kuvasuhde kuin referenssikuvissa. Asetin referenssikuvat 90 asteen kulmaan keskenään, jotta etunäkymästä näkyi kuva hahmosta edestä ja vasemmasta näkymästä näkyi hahmon kuva oikealta sivulta.



Kuva 12. Referenssikuvat työalueen ympärillä.

Hahmon yksinkertaisuuden vuoksi lähdin liikkeelle pelkästä box-primitiivistä, jossa jokaisessa vaiheessa pidin polygonimäärän mahdollisimman vähäisenä. Yksityiskohtia ja polygoneja lisättiin tarvittaessa eri

hahmon osa-alueisiin. Halusin olla ajan tasalla polygonien määrästä jokaisessa mallinnuksen vaiheessa, joten asetin polygonien lukumäärän näkyviin työskentelyn ajaksi.

Mallintaessa tuli pitää mielessä hahmon tarkoitus, joten se mallinnettiin T-asentoon, jossa hahmon kädet ovat sivuilla ja jalat hieman erillään toisistaan. Mallintamalla hahmo T-asentoon pystyin näkemään raajat selkeämmin ja animoimaan hahmon raajojen liikkeet luonnollisemmin. Unityssä on myös animaatioiden hiomiseen tarkoitettuja työkaluja, jotka toimivat oikein vain hahmon ollessa T-asennossa. T-asento on lähtökohtaisesti hyvä periaate hahmoa mallinnettaessa, sillä sen avulla vältytään ongelmilta luurankovaiheessa ja pidetään työ selkeänä.

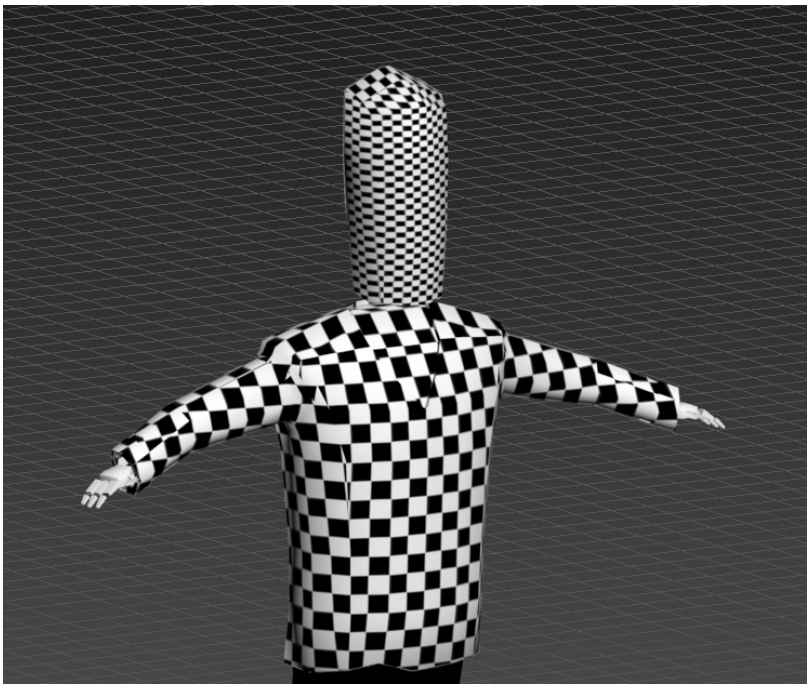
5.2 UVW-kartoitus

Hahmoni malli oli valmis, mutta siitä puuttui tekstuurit. Jos olisin asettanut materiaalit tekstuureineen tässä vaiheessa suoraan hahmoon, ne eivät olisi menneet paikoilleen hahmon pinnassa. Seuraava tavoitteeni oli määrittää UVW-kartoitus 3D-geometrialle.

UVW-kartoituksella tarkoitetaan 2D-kuvan asettamista 3D-malliin. UVW-kartoituksella pystytään määrittelemään tarkasti mihin tekstuuri asettuu 3D-kappaleessa. Käytännössä, 3D-kappaleen geometria litistetään 2D-tasoksi, jonka päälle asetetaan tekstuurikuva tarkasti haluttuihin kappaleen koordinaatteihin.

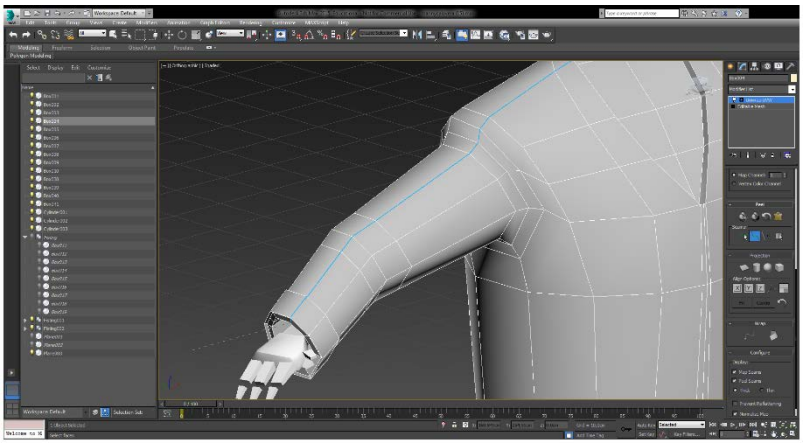
UVW-kartoitus tehdään mahdollisimman yksinkertaisesti ja suunnitellaan, missä kohden takkia saumat menevät oikein. Valmiit UVW-kartat renderöidään ulos 3ds Maxista ja ne tuodaan Adobe Photoshopiin. UVW-kartan kuvapohjien avulla pystymme luomaan hahmon tekstuurit tarkasti oikeille kohdilleen. Valmiit tekstuurit asetetaan hahmon materiaaliksi.

Asetin hahmoni tekstuuriksi shakkikuvion, jotta näin miten nykyinen UVW-map toimii. Kuten kuvasta 13 voi huomata, shakkikuviot eivät asetu tasaisesti ja ne ovat erikokoisia eri puolilla hahmoa. Halusin itse määrittää UVW-kartoituksen, jotta pääsin parhaaseen mahdolliseen lopputulokseen. 3ds Max luo aina oletusarvoisen UVW-kartan 3D-objekteille, mikä mielestäni vain häiritsee UVW-työskentelyä, joten poistin sen ja aloitin puhtaalta pöydältä. UVW-poistaminen vaati kappaleen muuttamista editable meshiksi ja sitten utilities-välilehdeltä UVW remove -toiminnon valitsemista.



Kuva 13. Epätasainen UVW-kartoitus.

Aloitin UVW-kartoituksen valitsemalla unwrap UVW -modifier ja päätin luoda itse saumat hahmoon. Hahmon pinnan UVW tuli pilkkoa osioihin, jotta sen levittäminen 2D-tasoksi menisi tasaisesti ja hallittavasti pienempään tilaan. Unwrap UVW -modifierin peel-osiosta löytyy työkalu seams, jonka avulla hahmon jakaminen osioihin on vaivatonta. Työkalulla voidaan valita vertexit, joista sauma alkaa ja päättyy, jolloin työkalu lisää sauman vertexien välisille edgeille. Saumat sijoitettiin hahmon takkiin sellaisiin sijainteihin, joissa kuvittelisin niiden oikeassa takissa sijaitsevan, kuten nähdään kuvassa 14.



Kuva 14. Saumojen määrittäminen hahmon takin reunoihin.

Quick Peel -työkalulla sain valitun UVW-map alueen tasoitettua ja levitettyä tasaisesti 2D UVW -näkymään. Kun olin saanut UVW-mapin tehtyä,

renderöin sen ulos 3D-ohjelmasta. UVW-karttakuva voitiin nyt viedä valitsemaani kuvankäsittelyohjelmaan, jossa pystyin sijoittamaan kartan mukaan oikeisiin paikkoihin oikeat tekstuurin elementit (kuva 15.). Kun tallensin tekstuurin ja asetin sen 3D-ohjelmassa hahmoni materiaaliksi, asettuivat takin tekstuurit oikein hahmon päälle.

Kuva 15. UVW-mapping ja tekstuurit

UVW-mappingin jälkeen pääsin rakentamaan hahmolle animaatiokontrolli-järjestelmää, mikä helpottaa animointiprosessia vaikuttavasti. Lähdin pohtimaan, millä tavalla rakennan hahmolle animaatiokontrollin, eli rigin.

5.3 Rigging CAT-järjestelmässä

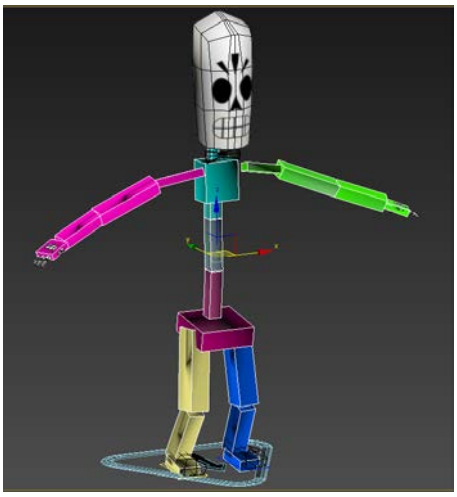
Hahmo oli tässä vaiheessa mallinnettu ja UVW-kartoitus asetettu halutulla tavalla kohdilleen. Seuraavaksi tavoitteenani oli alkaa valmistelemaan hahmon animointia. Animaation tekemiseen tarvitsi kuitenkin tehdä ensiksi rigi, sillä ilman sitä animaation tekeminen olisi tarpeettoman työlästä.

Riggauksella tarkoitetaan animaatiokontrollerin luomista, jonka avulla 3D-malli voidaan animoida vaivattomammin. Kontrollerille määritetään vaikutusalueita mallista, joihin sen liikuttelu vaikuttaa. Tämä tarkoittaa sitä, ettei 3D-kappaleen yksittäisiä vertexejä tarvitse siirrellä erikseen, vaan voidaan määrittää erillinen kontrolli, jonka liikuttelun myötä halutut osat mallista liikkuvat. Hahmon animaatiokontrolleria kutsutaan usein luurangoksi. Unity-pelimoottori vaatii animaatioiden pohjautuvan luurankorakenteeseen ja tämä on selkein tapa animoida hahmo.

3ds Max-ohjelmistossa voidaan joko käyttää biped-nimistä luurankosysteemiä, mikä mahdollistaa jo tehtyjen animaatioiden yhdistämisen luurankoon ja niiden miksaamisen keskenään. Toinen vaihtoehto on Character Animation Toolkit -luurankosysteemi, jonka työskentelytavat koen kaik-

kein tutuimmaksi. Character Animation Toolkit (CAT) -luurangon animointi muistuttaa paljon Adobe-ohjelmistojen tasotyöskentelytapaa. CAT-järjestelmässä on olemassa animaatiotasoja, joihin voidaan animoida eri animaatioita jopa yksittäisille luille ja sekoittaa animaatiotasoja tarkasti keskenään. Tämä mahdollistaa sulavia vaihdoksia animaatioiden välille ja pitää animaatiot eläväisinä.

CAT-systeemistä löytyy valmiita luurankomalleja erilaisia 3D-hahmoja varten, joista animoija voi valita sopivan hahmoaan varten. Joissakin tapauksissa voi olla viisainta luoda luuranko itse, jos on tarve tarkemmalle kontrollille hieman erilaisen hahmon geometriassa. Ensin määritellään hahmolle lantioluu, joka on myös samalla koko luurangon keskus. Kukin luu määritellään sopivaan kohtaan huomioiden luiden suunnitellut vaikutusalueet (kuva 16). Esimerkiksi lantioluun halusin sijaitsevan hahmon vyötärön seudulla ja otin huomioon, mistä kohdista ajattelin hahmon luonnollisesti taipuvan. Lantion jälkeen määritin jalat ja selkärangan, joiden nivelmäärät asetin tarpeellisuuden mukaan. Lopulta hahmon luuranko oli kokonaisuudessaan valmis ja testasin sitä asettamalla CAT-systeemin oletus kävelyanimaation. Tässä vaiheessa luuranko oli kuitenkin vielä irrallaan 3D-hahmomme geometriasta. Sen liittämiseksi tarvitsemme skin-modifieria, joka määrittää kunkin luun vaikutusalueet geometriassa.

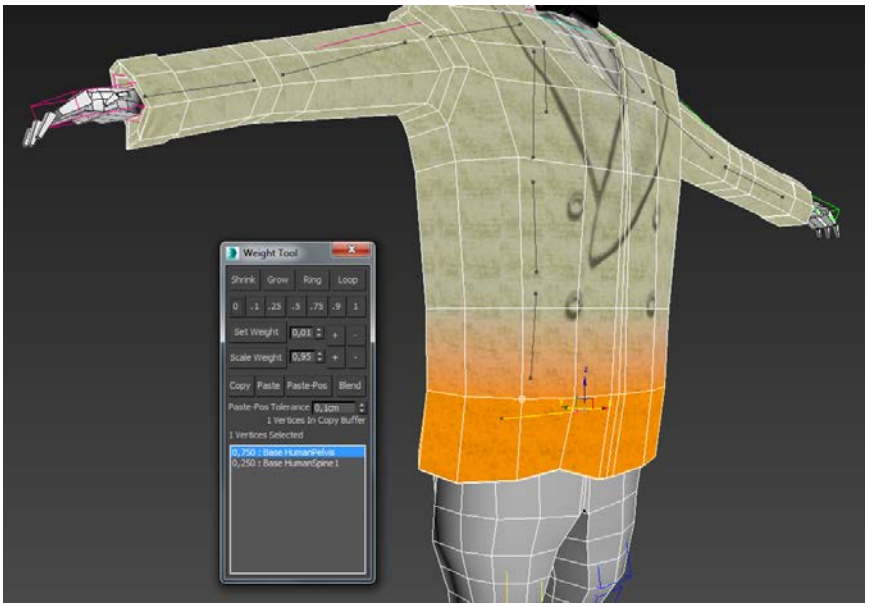


Kuva 16. Hahmon CAT-Rig

5.4 Skinning-vaihe

Liitetään luuranko hahmon geometriaan skin modifierilla. Tätä vaihetta kutsutaan skinning-vaiheeksi, jossa määritellään kunkin luun vaikutusalue kuhunkin hahmon vertexiin. Lähtökohtaisesti skinning-vaihe voi olla haastava ensikertalaiselle, joten suosittelen perehtymään skin modifier -työkaluun rauhassa. Hyvä perehtyminen eri työkaluihin säästää aikaa, kun löytää parhaat työskentelytavat heti alusta lähtien.

Skin-modifierin asetuksista pystytään määrittämään vertex-moodi päälle, minkä jälkeen pystyin määrittämään kunkin vertexin painoarvon suhteessa kuhunkin luuhun. Tämä mahdollistaa tarkan kontrollin luiden vaikutusalueista, mutta voi osoittautua työlääksi, jos hahmon geometrian polygonimäärä on korkea. Hahmoni oli tässä tapauksessa mallinnettu mahdollisimman pienellä polygonimäärällä, joten päätin työskennellä vertex-tasolla ja hyödynsin modifierin omaa weight toolia vertexien painoarvojen määrittelyyn. Weight toolilla pystyn valitsemaan valmiista ennalta asetetuista arvoista, tai määrittämään näppärästi oman arvoni tietylle vertexille (kuva 17). Työkalussa pystyn myös näkemään, mitkä muut luut vaikuttavat kyseiseen vertexiin ja laskemaan niiden arvoja suhteessa toisiinsa.



Kuva 17. Rigin luiden vaikutusalueiden painottaminen.

High poly -hahmoissa työskentely kannattaa aloittaa envelope-työkalulla. Envelopet ovat luiden vaikutusalueita merkitseviä kapselimuotoja. Käytännössä luun envelopen alueen sisälle jäävät vertexit ovat luun vaikutusalueen sisällä. Envelopen vaikutusalueen vahvuuksia ja muita säätöjä voidaan säätää luukohtaisesti modifierin asetuksista.

Skinning-vaiheessa kannattaa aina välillä tarkastella, miten geometria taipuu luun eri asennoissa. Hyvä keino on animoida hahmo erilaisiin ääriasentoihin ja niiden pohjalta aikajanalla siirtyen korjailla luiden vaikutusalueita oikeiksi.

5.5 Hahmon animointi CAT-järjestelmässä

Tavoitteenani tässä vaiheessa oli saada animoitua hahmolle tarpeellisimmat idle- ja kävelyanimaatiot. CAT-järjestelmän animaatiot perustuvat tasoihin, joista kuhunkin voidaan tallentaa animaatiota ja

sekoittaa niitä tarvittaessa keskenään. Animaation luominen tasoille tapahtuu täysin samalla tavalla kuin 3ds Maxissa animointi muutenkin; voidaan käyttää esimerkiksi auto key -ominaisuutta, varmistetaan halutun CAT-tason olevan valittuna ja käännellään luita eri kohdissa aikajanalla.

Käytin kävely-animaatioon CAT-järjestelmän omaa CAT Motion -ominaisuutta, jossa hahmon luurangon pohjalta tehdään oletuskävelyanimaatio. Tätä kävelyanimaatiota voidaan eri liukusäätimillä säädellä kohti toivottua lopputulosta. Lisäsin myös omia tasoja, joihin muokkasin toivottuja kädenliikkeitä, pään heilumista ja muokkasin jalkojen etäisyyttä toisiinsa kävellessä.

Halusin pitää työskentelyn mahdollisimman järjestelmällisenä, joten päätin animoida kaikki animaatiot samalle aikajanalle. Tämä tarkoittaa sitä, että animoin kävelyanimaatiota aikajanalle kohtiin 1-50 ja freimit 51-100 sisälsi idle-animaation, jossa hahmo vain seisoi paikallaan hieman hengitellen ja päätään liikkuttaen. Tällä tavalla sain kaikki animaatiot siirtymään Unityyn export-vaiheessa yhtenä kokonaisuutena ja pilkoin animaatiot erilleen vasta Unityn Import-asetuksissa.

Tärkeää tämänkaltaisessa työskentelyssä on huomioida, että animaatiot toistuvat, eli looppaavat, saumattomasti. Tämä tarkoittaa sitä, että hahmon kävellessä pysähtymättä eteenpäin sama animaatio voi pyöriä alusta loppuun toistuvasti ilman, että näkyy selkeää nykäisyä kunkin kierron välillä.

5.6 Export

Tässä vaiheessa hahmoni oli valmis animaatioineen, joten tavoitteenani oli saattaa se Unity-pelimoottoriin käytettäväksi. 3ds Max -ohjelmistosta hahmo pystytään viemään ulos .fbx-muodossa, joka on yksi Unityn tukemista pakkausmuodoista. Valitsin käytettäväksi kyseisen tiedostomuodon, koska olen tottunut käyttämään sitä, ja sen asetukset olivat minulle entuudestaan tuttuja. En myöskään ollut törmännyt ongelmiin kyseistä tiedostomuotoa käyttäessäni. Vaihtoehtoisesti olisin voinut käyttää 3ds Max -ohjelmiston natiivia tiedostomuotoa, jota Unity osaa lukea, mikäli samalla työkoneella on myös 3ds Max asennettuna.

Fbx Export -asetuksista on tärkeää tarkistaa, että animaatiot on valittu vietäväksi mallin mukana, ja että Bake Animation -valinta on päällä. Bake Animation tarkoittaa animaatioiden tiedon sitomista kokonaisuuteen. Määritän animaation alku- ja loppufreimit, sekä asetan step-osioon arvoksi 1, joka määrittää, että jokainen animaation freimi otetaan huomioon Bake-prosessissa. Tällä tavalla en hävitä mitään tietoa animaatiosta ja saan sen sellaisenaan Unityyn.

6 PELIHAHMO UNITY-PELIMOOTTORISSA

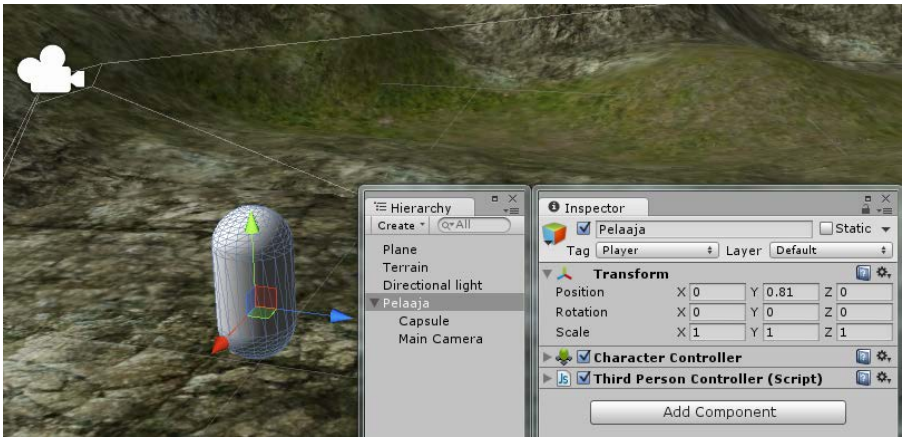
6.1 Hahmon ohjaus Unityn omilla skripteillä

Tavoitteenani tässä vaiheessa oli lähteä luomaan pelihahmoa pelimoottoriin. Halusin ensin luoda pelihahmon pelilogiikan mahdollisimman valmiiksi ennen kuin toin luomani hahmon grafiikan ja animaatiot moottoriin. Tämä työvaihe edellyttää Unityn perustuntemusta, mutta opinnäytetyöni havainnollistaa työvaiheet mahdollisimman kattavasti.

Aluksi loin tyhjän peliobjektin Unity-projektiin. Tämä toimi niin kutsuttuna säiliönä, jonka sisälle keräsin kaiken pelihahmolle oleellisen kuten grafiikan animaatioille ja toiminnallisuudelle tärkeät kontrolliskriptit. Peliobjekti kannattaa nimetä tunnistettavaksi ja tässä tapauksessa annoin sille nimen Pelaaja.

Loin uuden 3D-kapselin, jonka siirsin hierarkiassa Pelaaja-objektin sisälle. Kapselin tarkoitus oli toimia väliaikaisena grafiikkana hahmolle ennen kuin toin sen geometrian Unityyn. Poistin kapselin komponenteista Collider-komponentin, sillä halusin kapselin toimivan vain grafiikkana, eikä niinkään interaktiivisen kappaleena pelimaailmassa. Sijoitin myös Main Camera -kameran Pelaaja-objektin sisälle hierarkiassa ja liikutin sen haluamaani sijaintiin hahmon yläpuolelle tai taakse (kuva 18s).

Lisäsin Pelaaja-objektiin Character Controller -komponentin, joka määrittää hahmon törmäykset ympäristön kanssa. Seuraavaksi lisäsin Unityn valmiin kolmannen persoonan kontrolliskriptin, minkä jälkeen hahmo oli ohjattavissa pelissä WASD-ohjauksella. Third Person Controller (Script) -komponentista pystyin säätämään hahmon liikkumista eri säädöille miellyttäväksi.



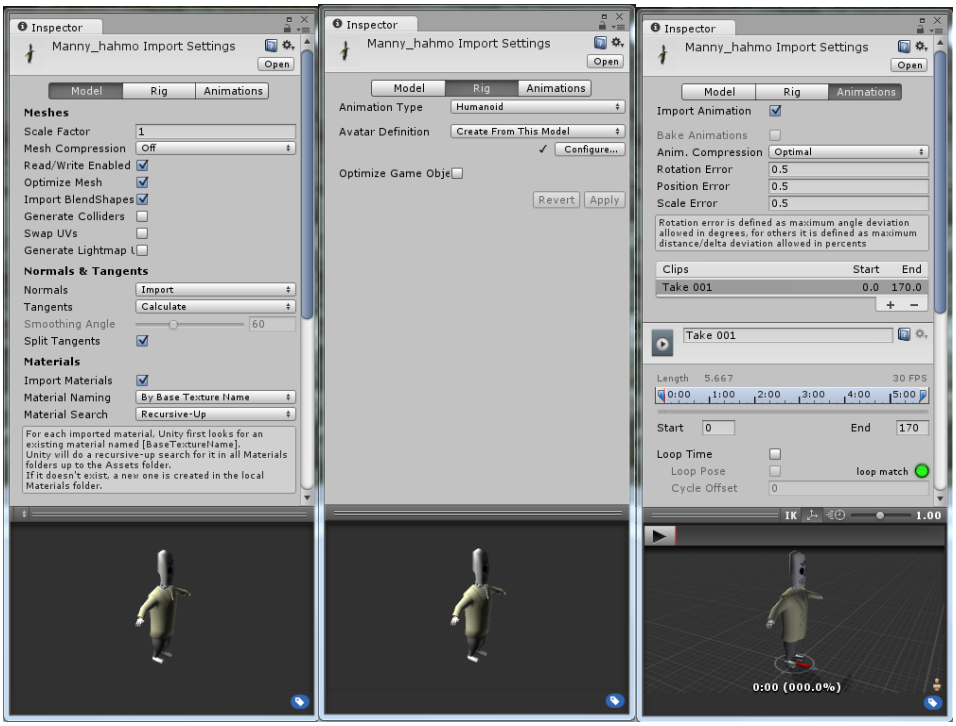
Kuva 18. Kamera sijoitettu Pelaaja-peliobjektin taakse.

6.2 Oman hahmon tuominen Unityyn

Seuraavaksi halusin tuoda Unityyn oman hahmoni Import New Asset -toiminnolla. Valitsin tuotavaksi hahmon .fbx-muotoisen tiedoston ja valitsin import, minkä jälkeen Inspector-valikkoon aukesi tuomani assetin import-asetukset. Kyseistä asetuksista pystyin määrittämään, mitä Unity tuo .fbx-tiedoston tiedoista mukanaan projektiin, kuten materiaalit ja animaatiot.

Rig-välilehdeltä määrittelin animaatiotyypin hahmon luurangolle sopivaksi. Animaatiotyyppivaihtoehtoja ovat Generic-animaatiotyyppi, joka soveltuu parhaiten luurangoille, joilla on jokin muu kuin ihmisen luurankorakenne. Humanoid-vaihtoehto taas mahdollistaa ihmistyyppisten hahmojen luurankojen ja niiden animaatioiden yhteiskäytön keskenään. Humanoid-tyyppisissä luurangoissa on pää, keho ja raajat. Valitessani Humanoid -animaatiotyypin, pystyin hyödyntämään täysin eri luurankojen pohjalle tehtyjä animaatioita omassa hahmossani.

Animations-välilehdeltä pystyin määrittelemään, tuonko animaatiot mallin mukana ja tarvittaessa pilkkomaan animaatioaikajanan erillisiin animaatioklippeihin. Esimerkissä olin animoinut kaikki haluamani animaatiot peräkkäin, joten oli tarpeellista pilkkoa ne yksittäisiksi animaatioiksi Import-asetuksissa (kuva 19). Valitsin ensimmäisen, Take 001:ksi nimetyn animaation, ja rajasin sen aloitus- ja lopetusfreimit kävelyanimaation pituuden mukaan. Nimesin Take 001 -animaatioklipin paremmin nimellä ”kävely” ja painoin alhaalla olevaa + painiketta lisätäkseni uuden klipin käsittelyyn. Kaiken kaikkiaan erottelin 3 eri animaatiota toisistaan: kävely, idle- ja puheanimaatiot.



Kuva 19. Hahmon import-asetukset

Sijoitin hahmoni grafiikan Pelaaja-objektin alle hierarkiassa ja kohdilleen pelimaailmassa, aivan samalla tavalla kuin tein kapselille ja kameralle. Poistin tässä vaiheessa turhan kapselin hierarkiasta. Testatessani peliä näin hahmon pelimaailmassa, mutta tämän kävelyanimaatio ei kytkeytynyt päälle liikkuessaan. Tämä johtui siitä, etten ollut vielä linkittänyt animaatioita hahmoon ja toiminnallisuuteen.

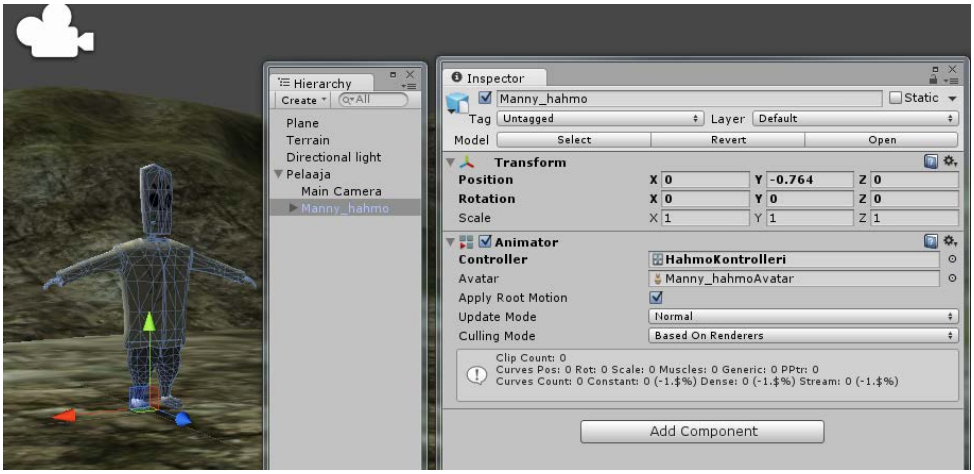
6.2.1 Uuden animaation tuominen projektiin jälkikäteen

Mikäli olisin vielä tarvinnut lisää animaatioita hahmolleni, olisin voinut lisätä uuden animaation hahmoomme jälkikäteen. Olisin voinut animoida uuden animaation mallinnusohjelmassa aiemmin käytettyyn luurankoon, ja tuoda .fbx-tiedoston täysin samalla tavalla Unityyn kuin aiemminkin. Tässä tapauksessa en kuitenkaan olisi halunnut tuoda muuta kuin animaatiot, joten jätin materiaalit tuomatta ja käänsin huomioni rig-välilehteen.

En halunnut luoda rig-välilehdeltä uutta Avataria, sillä loin sen jo tuodessani hahmon ensimmäistä kertaa Unity-pelimoottoriin. Sen sijaan päätin käyttää aiemmin luomaani Avataria, jotta animaatiot kohdistuvat oikeaan hahmon versioon.

6.3 Animaatioiden liittäminen hahmoon Mecanim-järjestelmässä.

Seuraavaksi sidoin animaatiot hahmoon Unityn Mecanim-järjestelemässä. Loin Animator Controllerin projekti-ikkunassa, nimesin sen esimerkillisesti ”Hahmokontrolleri” -nimellä ja siirsin sen hahmon Animator-komponentin sisällä olevaan Controller-osioon (kuva 20). Tämän avulla Unity osaa yhdistää kyseisen kontrollerin kuuluvan hierarkiassa olevaan hahmoon.

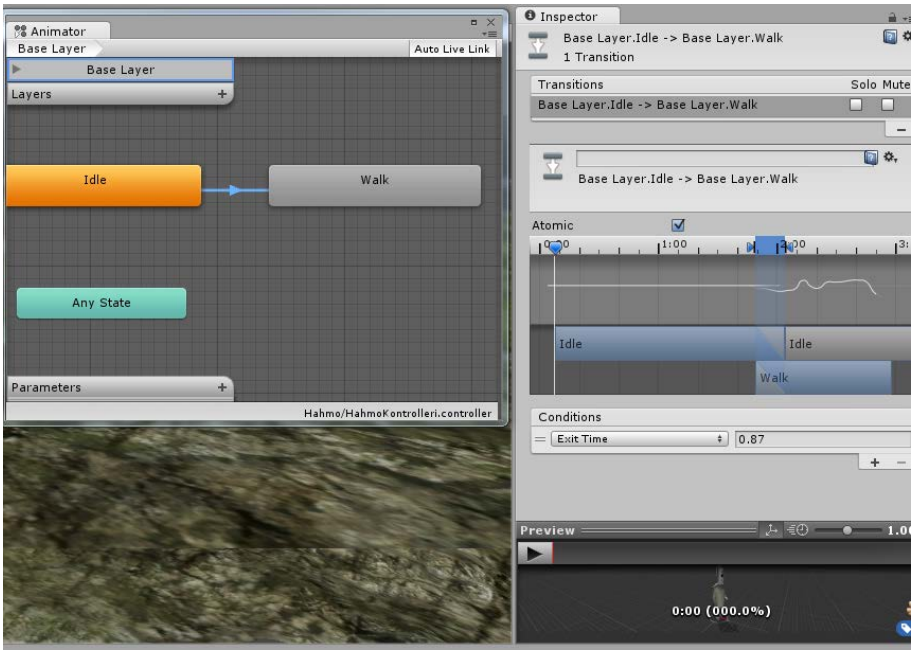


Kuva 20. Hahmon Animator Controller.

Kontrollerin määrittämisen jälkeen Animator-ikkunaan avautuu Mecanim-järjestelmän nodepohjainen -käyttöliittymä. Lisäsin animaatioklipit Animator-ikkunaan ja aloitin niiden yhdistämisen parametreihin, jotka tarkistavat skriptiehtojen täyttymisen. Animaatiot löytyivät projekti-ikkunassa olevan hahmon alta, jonka sain auki painamalla nuoli-ikonia. Ensimmäinen animator-ikkunaan vetämäni animaatioklippi muuttuu väriltään oranssiksi, mikä merkitsee sen olevan oletusanimaatio, joka käynnistyy kun aloitan pelin.

6.4 Siirtyminen idle-animaatiosta walk-animaatioon

Kun raahasin walk-animaation idle-animaation vierelle Animator-ikkunaan, pystyin huomaamaan sen luovan uuden laatikon nimeltään walk, joka oli väriltään harmaa. Harmaa väri osoittaa, että kyseinen animaatio ei ole oletusanimaatio, eli se ei lähde pyörimään ellei siihen siirrytä ehtojen täyttyessä. Siirtymisen luomiseksi tulee klikata idle-animaatiota hiiren oikealla painikkeella, jolloin kursorin päähän tulee nuoli, jolla pystytään määrittelemään mihin animaation siirtymän halutaan johtavan. Valitsin kohteeksi walk-animaation ja animaatio-nodejen väliin ilmestyi nuoli. Nuolta klikkaamalla saadaan auki siirtymäasetukset inspector-ikkunaan (kuva 21).



Kuva 21. Animaatio tilat ja niiden siirtymät

Siirtymä käytännössä vaihtaa animaatiosta toiseen tietyn ehdon täyttyessä, esimerkiksi pelaaja-objektin liikkeen alkaessa. Siirtymien asetuksista pystyin määrittämään miten nopeasti ja millä tavalla siirtymä toteutuu. Siirtymiä luodessa on tärkeää varmistaa, että animaatiot johtavat toisesta tilasta aina takaisin alkuperäiseen. Loin siis uuden siirtymän walk-animaatiosta takaisin idle-animaatioon.

Halusin seuraavaksi asettaa ehdon, jonka täyttyessä siirtyminen idle-animaatiosta walk-animaation alkaa ja ehdon, jonka täyttyessä animaatio palaa takaisin. Pystyin määrittelemään uuden parametrin Animator-ikkunan alalaidasta ja valitsemalla haluamani parametrityypin vaihtoehdoista. Tässä tapauksessa käytin Float-tyyppistä parametriä, jolle annoin nimeksi speed. Tavoitteenani oli, että jos hahmon speed on korkeampi kuin arvo 0, alkaa siirtyminen kävely-animaatioon ja jos hahmon speed on vähemmän kuin arvo 0.1, siirrytään takaisin idle-animaatioon.

Asetin luomani speed-parametrin molempien siirtymien conditions-kohtaan ja halutut arvot ehdoiksi. Osuus näiden siirtymien osalta oli Animator-ikkunassa nyt valmis, mutta animaation ehto ei vielä voinut aktivoitua, sillä en ollut määritellyt sitä liitoksiin mihinkään variaabeliin hahmon kontrolliskriptissä. Käytännössä speed-parametrin arvot eivät muuttuneet, koska arvolle ei ole asetettu lähdettä.

6.4.1 Parametriin viittaaminen skriptissä

Ensiksi avasin hahmon Third Person Controller -skriptin valitsemassani editorissa. Tavoitteenani oli määritellä skriptissä, mitä Animator kontrolleria käytetään ja mitkä jo olemassa olevat skriptin variaabelit vaikuttavat luomaani speed-parametriin. Aluksi loin variaabelin nimeltään anim ja määrittelin sen olevan Animator-tyyppistä dataa. Seuraavaksi tavoitteenani oli määritellä mistä anim-variaabeli saa datansa, eli halusin sen löytävän Animator-komponentin, joka sijaitseen hierarkiassa Pelaaja-objektin alla (kuva 22).

```
private var anim : Animator;


function Start () {

anim = GetComponentInChildren(Animator);

}
```

Kuva 22. Animatoriin viittaus ja komponentin haku koodilla.

Kun anim oli linkitetty skriptissä Animator komponenttiin, pääsin käsiksi sen arvoihin skriptissä. Komento anim.SetFloat(”speed”, controller.velocity.magnitude); muuttaa parametrin arvon riippuvaiseksi kontrollerin velocity-arvosta. Lause sijoitettiin Update-funktioon if (IsGrounded) ehdon sisälle (kuva 23).

```
if (IsGrounded())
{

    transform.rotation = Quaternion.LookRotation(moveDirection);
    anim.SetFloat("speed", controller.velocity.magnitude);


}
```

Kuva 23. Hahmon liikkeen vauhdin arvon liittäminen speed parametriin

Testattaessa peliä siirtymät lähtivät toimimaan hahmoa liikuteltaessa. Animator-ikkunasta pystyin hienosäätämään siirtymiä sulavammiksi ja erilaisiksi. Tämä mahdollisti hahmon ylimääräisen liikkeen hiomisen pois ja muuten sulavan animaation vaihdoksen eri tilanteissa.

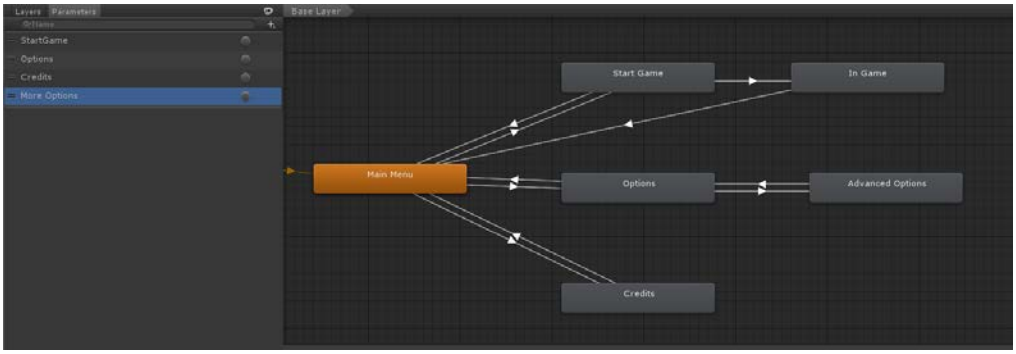
Vastaavaa toimintatapaa pystytään soveltamaan kaikkiin hahmon animaatioihin. Skriptissä voidaan esimerkiksi aktivoida triggeri, kun pelaaja osuu esineeseen, joka puolestaan aktivoi parametrin Animator-ikkunassa, jolloin hahmo toteuttaa halutun animaation.

7 UNITY 5 JA UUDET OMINAISUUDET

Unity 5 -version mukana Mecanim-järjestelmään tuodaan lisäominaisuuksia ja sen käyttöä on helpotettu. Yksi huomattavimpia uudistuksia on satunnaisnode -ominaisuus, jossa Unity valitsee useista animaatioista satunnaisesti yhden suoritettavaksi. Tämä toimii loistavasti esimerkiksi hahmon idle-animaatioiden vaihtelevuuteen, mikä vaikuttaa pelin immersioon. Hahmolla voi olla esimerkiksi erilaisia animaatioita paikallaan seisoskeluun, jotka vaihtuvat satunnaisesti. Hahmo voi yhdessä animaatiossa hengitellä ja hieman katsella ympärilleen, kun taas toisessa sytyttää tupakan ja polttaa sitä. (Unity Blogs 2015.)

Uutena ominaisuutena on IK chain kontrolloinnin syveneminen. Kehittäjä pystyy vaihtamaan eri IK-asetukset eri animaatiotasoja varten ja halutessaan muokkaamaan yksittäisiä hahmon osia koodin avulla luoden uusia animaatioita. Unityn kehittäjät julkaisevat myös API:n, joten käyttäjät pystyvät muokkaamaan ja luomaan omia työkaluja työskentelyn helpottamiseksi Mecanimissa. (Unity Blogs 2015.)

Uudessa versiossa käyttäjä pystyy myös luomaan node-tasoja ilman, että niissä on animaatioita. Tämä tarkoittaa parempaa animaatioiden hallintaa ja kehittäjä pystyy yhdistämään kokonaisuuksia yhteen, esimerkiksi kaikki pelin päävalikon elementit voivat olla Main Menu -staten alla (kuva 24). . (Unity Blogs 2015.)



Kuva 24. Node-tasojen ryhmittely.

8 YHTEENVETO

3D-hahmon käyttöönotto on monivaiheinen prosessi ja 3D-hahmoja voi olla hyvin erilaisia. Korkeamman polygonimäärän omaavat 3D-hahmot saattavat tarvita skulptausohjelmistoa yksityiskohtien hallitsemiseen, kuten esimerkiksi Autodeskin Mudboxia. Skulptausohjelmisto mahdollistaa yksityiskohtien kaivertamisen ja materiaalien tarkan maalaamisen 3D-hahmon geometrian pintaan. Kuitenkin työtavat ja vaiheet ovat hyvin samankaltaisia prosessin kokonaisuutta katsottaessa. Tästä huolimatta tekijän tulee ymmärtää 3D-grafiikan tuotannon periaatteet ja tuntea käytettävät työkalut.

Hahmon UVW-kartoitus on tärkeä vaihe ja se on paras toteuttaa hahmon mallinnuksen jälkeen. Unityn kaltainen pelimoottori ei edes sisällä omaa UVW-kartoitustyökalua joten tässä tapauksessa UVW-koordinaatit voidaan määrittää vain 3D-ohjelmistossa. Myös animaatiot tulee tehdä huolellisesti 3D-ohjelmistossa, jotta säästytään ongelmilta pelimoottorissa ja animaatioiden jatkuvalta korjailulta.

Unityn puolella on ehdottoman tärkeä muistaa hyödyntää Mecanim-järjestelmää. Se mahdollistaa animaatioiden paremman hallinnan, sillä erilaisia animaatioita saattaa kertyä suuri määrä ja sen johdosta niiden hallinta voi mennä sotkuiseksi. Mecanimin node-pohjainen käyttöliittymä havainnollistaa animaatiot innovatiivisella tavalla ja niiden välisistä interaktiivisuuksista pystytään pysymään perillä havainnollistavien transition-viivojen avulla. Mecanim mahdollistaa myös hyvän kontrollin animaatioihin Unityssä, sillä siinä pystytään varmistamaan niiden siirtymien sulavuuden animaatioista toiseen ja näkemään parametrit, joihin ne ovat liitännäisiä. Tämän avulla pystytään myös näkemään mahdolliset ongelmat, jotka ovat ilmenneet jo animointivaiheessa. Mecanimin avulla pystytään myös hieman lieventämään ongelmia säätämällä kunkin animaation siirtymisnopeutta, vahvuutta ja ajoitusta suhteessa toiseen animaatioon.

Ohjelmoinnin perustuntemus on tärkeää animaatioiden linkittämiseen Mecanimin parametreihin, tämä on kuitenkin hyvin dokumentoitu työvaihe, joten syvällistä tuntemusta ohjelmointikieleen ei tarvita.

Lähteet

3ds Max tutorials nda. Skin-modifier. Viitattu 7.5.2015 http://www.3dmax-tutorials.com/Skin_Modifier.html

3ds Max nda. Unit Setup. Viitattu 15.3.2015. http://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/3DSMax/files/GUID-69E92759-6CD9-4663-B993-635D081853D2-htm.html

Adventure Creator. 2015. About AC. Viitattu 14.4.2015. http://www.adventurecreator.org/about/features

Autodesk 3ds Max Help. 2015. Introduction to Keyframe Animations. Viitattu 5.3.2015. http://help.autodesk.com/view/3DSMAX/2015/ENU/?guid=GUID-2CFAFA4C-B8A6-489F-9811-B09B0CA74B7B

Autodesk 3ds Max Manual. 2013. CAT's IK System. Viitattu 17.4.2015. http://docs.autodesk.com/3DSMAX/15/ENU/3ds-Max-Help/index.html?url=files/GUID-BB87B15F-7A2C-4C6F-AADF-3A5F2962549E.htm,topicNumber=d30e275397

Cry Engine. 2015. Get Cry Engine. Viitattu 28.3.2015. http://cryengine.com/get-cryengine

Digital-Tutors Blog. 2014b. Bringing the hero to life for the indie game development pipeline. Viitattu 30.3.2015. http://blog.digitaltutors.com/bringing-hero-life-indie-game-development-pipeline/

Digital-Tutors Blog. 2014a. Importance of lighting. Viitattu 17.4.2015. http://blog.digitaltutors.com/understanding-the-importance-of-lighting-for-games/

Digital-Tutors. 2014. Importing Character Animations. Viitattu 18.4.2015. http://www.digitaltutors.com/tutorial/1567-Quick-Start-to-Unity-Volume-2#play-40741

Totten, V. 2012. Game Character Creation with Blender and Unity. Sybex.

Tuts+. 2011. Colliders & Unityscript. Viitattu 13.4.2015. http://code.tutsplus.com/tutorials/getting-started-with-unity-colliders-unityscript--active-8367

Unity Blogs. 2014. Unity Scripting Languages. Viitattu 18.4.2015.

http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/

Unity Blogs. 2012. More Mecanim. Viitattu 30.3.2015. http://blogs.unity3d.com/2012/06/20/more-mecanim/

Unity Blogs. 2015. New Mecanim features. Viitattu 7.5.2015. http://blogs.unity3d.com/2015/02/11/having-fun-with-the-new-mecanim-features/

Unity Forum. 2013. Main differences between Mecanim and Legacy. Viitattu 10.4.2015. http://forum.unity3d.com/threads/main-differences-between-mecanim-and-legacy.207641/

Unity Manual. 2015a. Modeling Characters for Optimal Performance. Viitattu 15.3.2015. http://docs.unity3d.com/Manual/ModelingOptimizedCharacters.html

Unity Manual. 2015b. Preparing your own character. Viitattu 20.4.2015. http://docs.unity3d.com/Manual/Preparingacharacterfromscratch.html

Unity Manual. 2015c. Retargeting of Humanoid animations. Viitattu 20.4.2015. http://docs.unity3d.com/Manual/Retargeting.html

Unity Manual. 2015j. 3D formats. Viitattu 18.4.2015. http://docs.unity3d.com/Manual/3D-formats.html

Unity Manual. 2015d. Import Object. Viitattu 15.4.2015. http://docs.unity3d.com/Manual/HOWTO-importObject.html

Unity Manual. 2015e. Asset Workflow. Viitattu 15.4.2015. http://docs.unity3d.com/Manual/AssetWorkflow.html

Unity Manual. 2015f. Using Components. Viitattu 17.4.2015. http://docs.unity3d.com/Manual/UsingComponents.html

Unity Manual. 2015g. Character Controller. Viitattu 15.4.2015. http://docs.unity3d.com/Manual/class-CharacterController.html

Unity Manual. 2015h. Avatar Creation and Setup. Viitattu 15.4.2015. http://docs.unity3d.com/Manual/AvatarCreationandSetup.html

Unity Manual. 2015i. Animation State Machines. Viitattu 15.4.2015. http://docs.unity3d.com/Manual/AnimationStateMachines.html

Unity. 2015a. Public-relations. Viitattu 16.4.2015. http://unity3d.com/public-relations

Unity. 2015b. Tutorials - Physics – Colliders. Viitattu 18.4.2015.
https://unity3d.com/learn/tutorials/modules/beginner/physics/colliders

Unreal Engine. 2015. Viitattu 29.3.2015.
https://www.unrealengine.com/what-is-unreal-engine-4

Wikipedia. 2015b. Idle Animations. Viitattu 15.3.2015
http://en.wikipedia.org/wiki/Idle_animations

Wikipedia. 2015a. Rendering (computer graphics). Viitattu 20.3.2015.
http://en.wikipedia.org/wiki/Rendering_%28computer_graphics%29

Liite 1

KUVAT LOPULLISESTA DEMOSTA