Henrik Brummer

Pelihahmo Unity-moottorille

Metropolia Ammattikorkeakoulu

Medianomi (AMK)

Viestintä

Opinnäytetyö

2.5.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Henrik Brummer Pelihahmo Unity-moottorille 30 sivua + 2 liitettä 2.5.2017
Tutkinto	Medianomi (AMK)
Koulutusohjelma	Viestintä
Suuntautumisvaihtoehto	3D-animointi ja -visualisointi
Ohjaaja(t)	Lehtori Kristian Simolin

Opinnäytetyön tarkoituksena oli käydä läpi reaaliaikaisen pelihahmon valmistus sekä tuonti Unity-moottoriin. Kävin aluksi läpi Unity-moottorin keskeisimpiä ominaisuuksia sekä käyttöliittymän. Tutkin myös työssäni pelihahmoihin liittyviä 3D-mallinnuksen käsitteitä sekä prosesseja, sekä vertailen eri lähestymistapoja. Tämän jälkeen siirryn hahmon toteutukseen, käyden läpi oleellisimmat työvaiheet pelihahmon konseptoinnista animointiin. Lopuksi toin hahmon Unity-moottoriin, jossa tein hahmolle lisää animaatioita sekä partikkeliefektin.

Tuloksena syntyi reaaliajassa pyörivä animoitu 3d-pelihahmo Unity-moottorille.

Avainsanat	3D-grafiikka, pelihahmo, Unity

Author(s) Title Number of Pages Date	Henrik Brummer Game character for the Unity engine 30 pages + 2 appendices 2 May 2017
Degree	Bachelor of Culture and Arts
Degree Programme	Media
Specialisation option	3D Animation and Visualization
Instructor(s)	Kristian Simolin, Senior Lecturer

The purpose of this thesis was to go through the creation of a real-time game character and its application into the Unity game engine. I begin by presenting Unity's most important concepts and user interface. I also go through concepts relating to the 3D-modeling of game characters, and compare different approaches. Then, I moved on to creating the character, all the way from concept art to animation. Lastly, I brought the character into the Unity engine, where I made some additional animations and particle effects.

The final result was an animated 3D-character running in real-time on the Unity engine.

Keywords	3D graphics, game character, Unity

Sisällys

1 Johdanto

1.1 Työn lähtökohdat

Tämän opinnäytetyön tarkoitus oli käydä läpi viehättävän näköisen low poly -pelihahmon luominen mobiilialustoille Unity-moottorilla. Työn ponnahduslautana toimi Metropolia Game Studion päiväkoti-ikäisille lapsille suunnattu Soludus-peliprojekti. Pyrin vastaamaan siihen, millaisia työvaiheita 3D-pelihahmon luonnissa on ja mitä tulee ottaa huomioon erityisesti kehitettäessä Unity-moottorille.

Rajasin siis työni hahmon kehityksen kokonaiskuvaan sekä Unityn puolella työskentelyyn. Käyn läpi lyhyesti esimerkiksi 3D-mallinnuksen eri työvaiheet menemättä kuitenkaan erityisen syvälle käyttämieni ohjelmistojen työkaluihin tai käytäntöihin. Koin tämän rajauksen mielekkääksi siksi, että hahmon mallinnuksen käytännöt ovat ohjelmistoista suurimmaksi osaksi riippumattomia. 3D-mallinnuksen, teksturoinnin ja rigauksen voi suorittaa useilla eri ohjelmistoilla, kuten Autodesk Maya, Autodesk 3DS Max tai Blender.

Aloitan työni esittelemällä Unity-pelimoottorin luvussa 2. Käyn hieman läpi Unityn historiaa sekä esittelen Unityn keskeisimmät toiminnot. Luvussa 3 käyn aluksi läpi muutamia 3D-hahmojen mallinnukseen liittyviä käsitteitä. Sitten esittelen työn toiminnallisessa osuudessa luodun hahmon työvaiheet konseptoinnista animointiin. Luvussa 4 valmis hahmo tuodaan Unityyn.

Työn tavoite on toimia lyhyenä oppaana hahmon luonnista Unity-moottoriin niille, joita aihe kiinnostaa. Sinänsä opinnäytetyössä esitelty tieto löytyy muistakin lähteistä, mutta pyrin luomaan tiiviin paketin jonka pohjalta lukija voi aloittaa omien hahmojensa kehityksen joutumatta selaamaan paljolti muita lähteitä. Lähestyn aihetta peligraafikon näkökulmasta, eli en käsittele pelihahmoon liittyvää ohjelmointia.

1.2 Keskeiset käsitteet

Pelimoottori - Ohjelmistokehys, jolla rakennetaan pelejä.

Reaaliaikainen grafiikka – Grafiikkaa, joka piirtyy tietokoneruudulla niin nopeasti, että sen kanssa voi interaktoida. Esimerkiksi videopeleissä uusi kuva piirtyy usein jopa kolmekymmentä kertaa sekunnissa, joka sallii pelattavuuden. 3D-elokuvissa yhden kuvan piirtämisessä saattaa kestää tunteja tai jopa päiviä.

Verteksi - Piste, joista 3D-mallit koostuvat.

Edge - Kahden verteksin välille piirtyvä viiva 3D-mallin pinnalla.

Polygoni - Useammasta kuin kahden verteksin välille piirtyvä taso 3D-mallin pinnalla.

Topologia - Tapa, jolla edget ja polygonit ”virtaavat” mallin pinnalla.

UV-kartta - 3D-mallin pintaa kuvaava kaksiulotteinen projisointi, jolle voidaan maalata 3D-mallin tekstuuri.

Tekstuuri - Kaksiulotteinen kuva, joka määrittää 3D-mallin pinnan ominaisuuksia kuten värin.

Rigaus - Prosessi, jossa 3D-malli valmistellaan animoitavaksi asettamalla sille luita sekä kontrolliobjekteja, jotka ajavat luiden liikettä.

2 Unity-pelimoottori

Unity on monille alustoille suunnattu pelimoottori, jonka on kehittänyt Unity Technologies. Pelimoottori on ohjelmisto, jolla rakennetaan pelejä, jotka sitten pyörivät älypuhelimilla, tableteilla, tietokoneilla ja pelikonsoleilla. Pelimoottorissa yhdistetään eri ohjelmistoissa luodut graafiset elementit sekä koodi, eli pelin taustalla pyörivä logiikka. Unitylla voi kehittää pelejä jopa 27 eri alustalle. Unity ei ole yhtä tehokas tai suunnattu huippugrafiikkaan kuten muut pelimoottorit kuten Unreal Engine tai CryEngine, mutta se sopii hyvin yksinkertaisempien pelien kehittämiseen useille alustoille. (Pluralsight 2014.) Unity on myös halpa, tehden siitä hyvän vaihtoehdon pienemmille pelistudioille. Henkilökohtaiseen käyttöön Unitysta saa ilmaisversion, josta kuitenkin löytyy lähes kaikki maksullisen version toiminnot. Näin Unity on myös hyvä opiskelijoiden käyttöön.

2.1 Unityn lyhyt historia

Unity alkoi kahden ohjelmoijan, Nicholas Francisin ja Joachim Anten, yhteisenä projektina, jota he kehittivät verkon kautta. David Helgason liittyi myöhemmin projektiin, ja he päättivät kolmistaan kehittää perinteisten pelien sijasta teknologiaa, jonka päälle pelin voisi rakentaa helpommin. Kolmikko päätti kehittää alustan verkkoon suunnatulle 3D-grafiikalle, ja perustivat yrityksen nimeltä Over the Edge Entertainment (OTEE). Unityn kehityksessä kesti vuosia, ja pelimoottorin valmistuessa tiimi päätti kehittää kokonaisen pelin luomallaan teknologialla. Lopputuloksena syntyi maaliskuussa 2005 Ambrosia Softwaren julkaisema peli Gooball (kuvio 1). Peliprojektin kehitys auttoi parantamaan Unitya sekä rahoittamaan jatkokehitystä.

Kuvio 1. Gooball (Ambrosia Software)

Ensimmäinen versio Unity-moottorista julkaistiin myöhemmin samana vuonna, ja tiimi alkoi heti kehittämään seuraavaa versiota. Alkujaan Unity tuki ainoastaan kehitystä Applen OS X-käyttöjärjestelmälle, mutta tuki Microsoftin Windowsille ja nettiselaimille seurasi nopeasti perässä. (Haas 2002.) Vuosien varrella Unitysta on julkaistu monia uusia versioita, jotka ovat lisänneet toimintoja sekä tuettuja alustoja. Yksi merkittävimmistä julkaisuista oli Unity 4.3, joka lisäsi työkalut 2D-pelien kehittämiselle. Unityn uusin julkaisu, Unity 5, lisäsi muun muassa realistisempia materiaaleja sekä parannuksia fysiikkamallinnukseen. (Unity3d 2017.)

2.2 Unity-käsitteistö

Unityn käyttöön liittyy valtavasti erilaisia toimintoja, jotka tulee ymmärtää pelihahmon rakentamisessa. Esittelen tässä kappaleessa termistöä, jolle on käyttöä opinnäytetyön toiminnallisessa osuudessa. Tämä ei ole läheskään kattava lista Unitysta löytyvistä ominaisuuksista ja toiminnoista.

2.2.1 Scene

Scenet ovat tiedostoja, jotka sisältävät pelin objekteista koostuvan hierarkian. Scenet ovat tapa paloitella peli osiksi, jotka lopulta yhdistetään lopulliseen peliin. Esimerkiksi pelin kentät saattavat olla erillisiä scenejä, tai pelin alkuvalikko voidaan rakentaa erillisessä scenessä. Kun Unityssa aloittaa uuden projektin, Unity luo nimeämättömän scenen, jonka hierarkiasta löytyy kamera sekä yksinkertainen valo. Kameran asetukset riippuvat siitä, onko projekti 2D vai 3D. (Unity User Manual.)

2.2.2 GameObject

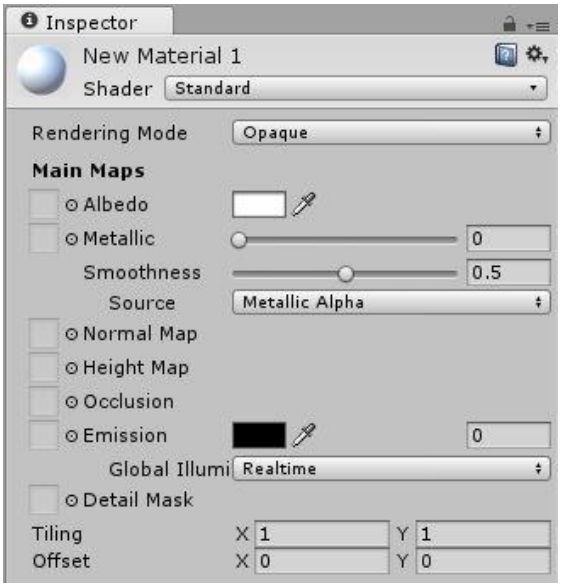
GameObject on yleinen termi kaikille objekteille, joita pelin scenessä voi olla. Unityn GameObject-valikosta voi luoda tyhjän GameObjectin, joka ei sisällä mitään muuta tietoa kuin objektin koordinaatit suhteessa origoon. Koordinaatit ovat objektin Transform-komponentti, joka löytyy kaikilta objekteilta. Tyhjään GameObjectiin voi sitten liittää muita komponentteja, kuten 3D-mallin tai kameran tai koodia. (Unity User Manual.)

2.2.3 Asset

Assetit ovat useimmiten Unityn ulkopuolella valmistettuja tiedostoja, jotka tuodaan projektiin. Esimerkiksi 3D-ohjelmistoissa valmistetut mallit ovat assetteja, kuten myös äänitiedostot ja tekstuurit. Toisaalta joitain assetteja voi myös valmistaa Unityssa, kuten animaatioita. Yleisesti ottaen, kaikki pelissä käytettävät tiedostot ovat assetteja. Yhdistämällä assetteja tyhjiin GameObjecteihin saadaan aikaan pelissä näkyvät elementit, olivat ne sitten hahmoja tai puita tai aluskasvillisuutta. (Unity User Manual.)

2.2.4 Materiaali

Materiaalit määrittävät pelissä näkyvien objektien pinnan tai partikkeleiden ulkonäön. Materiaalit koostuvat kuvatekstuureista sekä shadereistä. Shaderit ovat algoritmeja, jotka määrittävät kuinka materiaaliin asetetut kuvatekstuurit sekä valaistus yhdistetään lopulliseksi ruudulla näkyväksi pikseliksi. Unityssa tulee valmiiksi useita eri shadereitä, ja ne käyttäytyvät eri tavoin. Kuviossa 2 on nähtävillä Unityn perusmateriaali.



Kuvio 2. Standard-materiaali

Uudessa materiaalissa on oletuksena Unityn Standard Shader, jossa voi värin lisäksi säätää mm. materiaalin sileyttä ja metallisuutta. Mobiilialustoille suunnatussa, kevyessä Mobile/Diffuse shaderissa voi säätää ainoastaan materiaalin väriä. (Unity User Manual.)

2.2.5 Prefab

Kun scenessä olevaan GameObjectiin on liitetty useita komponentteja ja sitä halutaan käyttää useasti scenessä, objektista kannattaa luoda prefab-assetti. Prefab luodaan raahaamalla objekti Hierarchy-ikkunasta Project-ikkunaan. Tällöin objektin nimi näkyy sinisellä tekstillä Hierarchy-ikkunassa. Tämän jälkeen objektista voi tehdä rajattomasti klooneja, joita kaikkia voi muokata samanaikaisesti muokkaamalla Project-ikkunassa näkyvää prefab-assettia. Peliympäristöön saattaa esimerkiksi tulla paljon samankaltaisia kiviä. Kivi luodaan tekemällä GameObject, johon liitetään komponenteiksi kiven 3D-malli

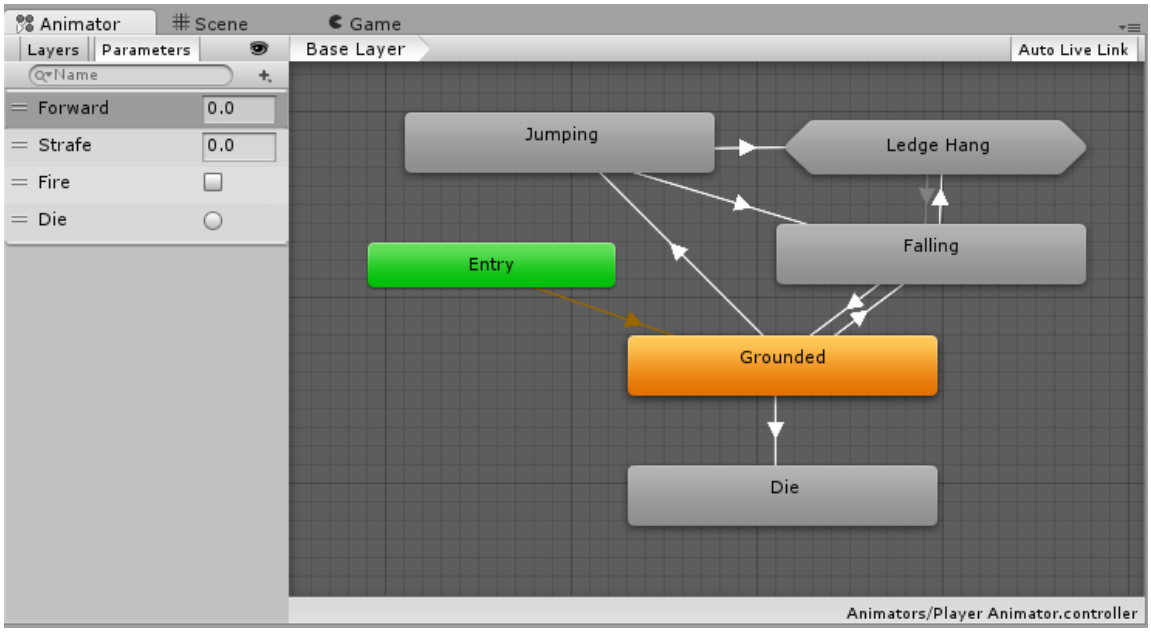
sekä materiaali, joka määrittää pinnan värin ja muut ominaisuudet, kuten kiiltävyyden. Kun kivestä luodaan prefab-assetti ja tätä monistetaan kaikkialle ympäristöön, kaikkien kivien väriä voi muokata muokkaamalla yhtä, sen sijaan että pitäisi käydä kaikki kivet yksitellen läpi. (Unity User Manual.)

2.2.6 Animation Clip

Animation Clip on assetti, joka sisältää animaatiodataa. Animation Clipejä voi luoda Unityn Animation-ikkunassa, mutta usein ne valmistetaan muissa ohjelmissa kuten Mayassa tai Blenderissä. Animation Clip usein sisältää yhden lyhyen animaation, kuten hahmon kävelyn tai hypyn. (Unity User Manual.)

2.2.7 Animator controller

Animator Controller on Unityssa luotava assetti, joka lisätään pelihahmon komponentiksi. Animator Controlleriin lisätään hahmolle kuuluvat Animation Clipit, jonka jälkeen niistä voi valmistaa kaavion (kuvio 3).



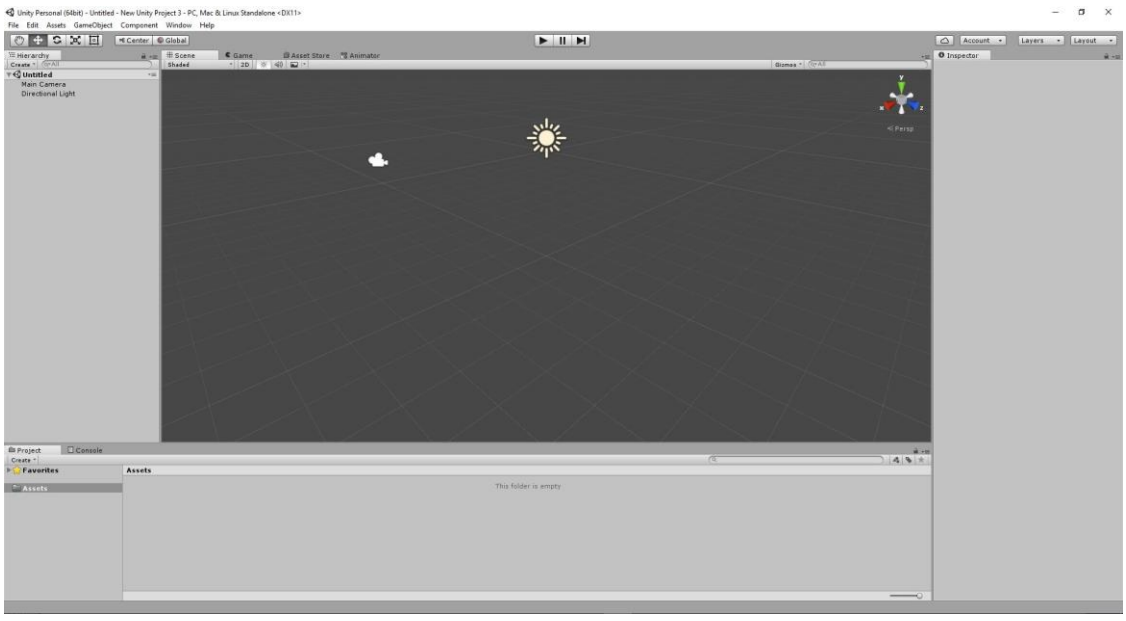
Kuvio 3. Animation controllerin asetuksia säädetään Animator-ikkunassa. (Unity user manual)

Controllerissa näkyvä kaavio määrittää, mistä animaatioista pitää pystyä siirtymään mihin animaatioihin. Esimerkiksi hahmon pitäisi pysytä siirtymään sulavasti juoksuanimaatiosta hyppyyn. Controllerissa tätä siirtymää voi viilata. Animation Clipejä voi myös asettaa eri animaatiotasoille, jotka lisätään toisiinsa. Esimerkiksi silmien räpäyttelyn kannattaa olla eri tasolla kuin kävelyn, jotta hahmo voi räpäytellä silmiään samalla kun kävelee. (Unity User Manual.)

2.3 Unityn käyttöliittymä

2.3.1 Unityn yleisnäkymä

Avatessa Unity näkyviin tulee ensin projektivalikko, josta voi avata aikaisemmin luodun Unity-projektin tai luoda uuden projektin. Jos luo uuden projektin, Unity pyytää määrittämään projektin nimen sekä sen, onko projekti 2D vai 3D. Kun tämä on tehty, esiin tulee Unityn tyhjä yleisnäkymä. Kuvio 4 näyttää Unityn yleisnäkymän uudessa projektissa.



Kuvio 4. Unityn yleisnäkymä

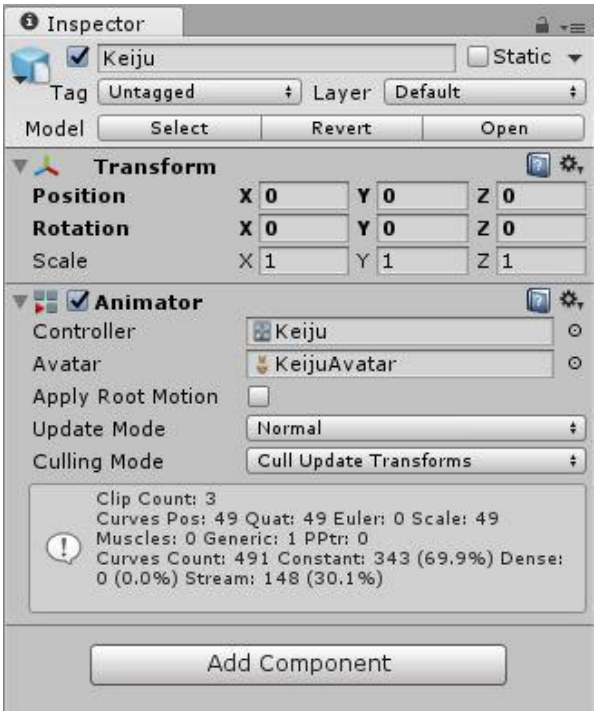
Yleisnäkymä sisältää Toolbarin, Inspector-ikkunan, Project-ikkunan, Hierarchy-ikkunan sekä Game-näkymän ja Scene-näkymän. (Unity User Manual.)

2.3.2 Toolbar

Toolbarista löytyy Unityn yleisimmät työkalut ja toiminnot. Eri työkaluja käytetään eri näkymissä. Esimerkiksi Toolbarista löytyvät objektien liikutteluun, pyörittämiseen ja skaalaamiseen tarvittavat työkalut on tarkoitettu käytettäviksi Scene-näkymässä. Play- ja Pause napit käynnistävät pelin, joka näkyy sitten Game-näkymässä määritetyn aktiivisen kameran kautta. Toolbarista löytyy myös linkki käyttäjän Unity-tiliin sekä Layers-valikko, josta voi valita, mitkä objektit kamera näyttää. (Unity User Manual.)

2.3.3 Inspector-ikkuna

Inspector-ikkunassa voi tarkastella valitun GameObjectin ominaisuuksia ja komponentteja (kuvio 5).



Kuvio 5. Inspector-ikkunassa näkyy valitun GameObjectin komponentit.

Ikkunassa näkyy esimerkiksi GameObjectin sijainti 3D-koordinaatistossa. Kun projektiin tuodaan uusia assetteja, Inspector-ikkuna näyttää assetin tuontiasetukset. Tuontiasetukset voi nähdä uudelleen valitsemalla assetti Project-ikkunasta. Toolbarista löytyvä Layers-valikko näkyy myös Inspector-ikkunassa, sekä muut projektin asetuksiin liittyvät

valikot. Kuvassa näkyy Keiju-hahmon prefab valittuna Inspector-ikkunassa. Koordinaattien lisäksi ikkunassa näkyy prefabiin liitetty Animator-komponentti. (Unity User Manual.)

2.3.4 Project-ikkuna

Project-ikkunasta näkyy kaikki projektiin kuuluvat assetit sekä assettien kansiorakenne. Uudessa projektissa löytyy ainoastaan kansio nimeltä ’Assets’ ja loput täytyy luoda itse, ottaen projektin tarpeet huomioon. Esimerkiksi 3D-malleille, tekstuureille, materiaaleille, äänille ja koodille kannattaa luoda omat kansionsa. Eri asseteista kootaan valmiit prefabit, joille voi myös tehdä oman kansion. (Unity User Manual.)

2.3.5 Hierarchy-ikkuna

Hierarchy-ikkuna näyttää Scenessä olevat GameObjectit. Jotkut objektit ovat suoraan Project-ikkunasta otettuja assetteja, mutta useimmat ovat todennäköisesti useista asseteista ja komponenteista koostuvia prefab-objekteja. Hierarchy-ikkunassa objekteja voi asettaa toisten objektien ”lapsiksi”, jolloin lapsiobjekti seuraa objektia, joka on sen ”vanhempi”. Tällöin lapsiobjektit koordinaatit määrittyvät vanhemman mukaan, sen sijaan että ne määrittyisivät maailman origon mukaan. (Unity User Manual.)

2.3.6 Scene-näkymä

Scene-näkymässä GameObjectit asetellaan kolmiulotteiseen pelitilaan. Pelin objekteja voi liikutella, pyörittää sekä skaalata. Scene-näkymään asetellaan myös pelin valot ja kamerat, joiden kautta peli nähdään. (Unity User Manual.)

2.3.7 Game-näkymä

Game-näkymässä näkyy lopullinen peli määritetystä aktiivisesta kamerasta. Game-näkymän ja Scene-näkymän välillä voi vaihdella painamalla nappia, joka löytyy heti näkymien yläpuolelta. Painalla Play-nappia Game-näkymä tulee automaattisesti esille, ja peli käynnistyy. (Unity User Manual.)

3 Hahmon toteutus

3.1 Soludus-hanke

Soludus on Metropolia Game Studion kehittämä EU-rahoitettu peliprojekti, jonka tarkoitus on opettaa kestävän kehityksen periaatteita päiväkoti-ikäisille lapsille. Peli on kehitetty tabletti-alustoille käyttäen Unity-moottoria. Kuviossa 6 näkyy kuvakaappaus Soluduksen kentästä, jossa keiju opastaa karhua sähkön säästämisessä.

Kuvio 6. Kuvakaappaus Soludus-pelistä

Peli pohjautuu AR-teknologiaan, jossa peli lataa jokaisen kentän kun tabletin kamera tunnistaa kentän oman, paperille tulostetun symbolin ympäristössä. Kentissä Hippa-keiju opastaa pelaajaa auttamaan Otso-karhua suorittamaan kestävään kehitykseen liittyviä tehtäviä, kuten roskien lajittelua sekä sähkön säästämistä. Hippa-keiju on se hahmo, jota tämä työ käsittelee. Keiju esiintyy useimmiten edestä kuvattuna peliruudun alalaidassa tai kentän vieressä.

3.2 Pelihahmon workflow

Pelihahmon toteuttaminen on työläs prosessi, johon kuuluu monta eri työvaihetta. Prosessi riippuu kuitenkin suuresti siitä, mille alustoille ja mihin tarkkuuteen tähdätään. Tarkkuudella tarkoitan polygonimäärän lisäksi kuvatekstuurien resoluutioita ja materiaalien monimutkaisuutta. Teknologian kehittyessä uusimpiin peleihin on pystytty valmistamaan aina vain yksityiskohtaisempia ja realistisempia hahmomalleja. Pelialan ikonisin hahmo on ehkä Nintendon pomppiva putkimies, Mario. Nintendo 64-konsolilla Marion pelimalli koostui 752 polygonista Super Mario 64 -pelissä. Vuonna 2013 Wii U -pelikonsolille julkaistussa Super Mario 3D World –pelissä Marion mallissa oli 9128 polygonia. (A+Start) Mario on kuitenkin suhteellisen yksinkertainen hahmo, joka ei pyri realismiin. Realismiin pyrkivissä uusimman konsolisukupolven peleissä polygonien määrä on vielä suurempi. Playstation 4:lle julkaistussa The Order: 1886 –pelissä hahmot koostuvat yli 100 000 polygonista (DualShockers 2014).

Tällaisten pelimallien toteuttamiseen on vakiintunut ns. ’High to low’ –workflow. Tässä työskentelytavassa tehdään ensin kymmenistä miljoonista polygoneista koostuva malli, johon veistetään ohjelmien kuten Zbrush avulla jokainen yksityiskohta vaatteiden saumoista hahmon ihohuokosiin. Tätä kutsutaan high poly -malliksi. Tämän jälkeen valmistetaan muutamista tuhansista polygoneista koostuva malli, joka myötäilee high polyn muotoja. Tätä kutsutaan low poly -malliksi, joka on tarpeeksi kevyt pyöriäkseen pelimoottorissa. (Wikipedia.) Polygonien tarkka lukumäärä riippuu täysin hahmosta ja pelistä, johon hahmo tulee. Määrä riippuu myös siitä, kuinka läheltä mallia voi tarkastella pelissä. Esimerkiksi autopeleissä polygonibudjetti käytetään lähinnä autoihin, ja tien laidalla seisova yleisö tehdään mahdollisimman kevyesti.

Soludus on kuitenkin tableteille tarkoitettu peli, joten sen tulee olla kevyt. En siis kokenut, että high poly -mallin tekeminen olisi tarpeellista. Tämä tietysti riippuu myös pelin tyylisuunnasta, joka on Soludus-projektissa yksinkertainen ja värikäs. Keijun tuotanto seurasi siis perinteisiä 3D-peligrafiikan käytäntöjä. Hahmo alkaa konseptipiirroksista, joiden perusteella rakennetaan 3D-malli. 3D-malli sitten valmistellaan teksturoitavaksi. Tätä työvaihetta kutsutaan UV-mappaukseksi. UV-mapin jälkeen mallille rakennetaan liikuteltava luuranko. Tätä työvaihetta kutsutaan rigaukseksi. Kun luuranko on valmis, hahmo voidaan animoida. Animoinnin jälkeen malli sekä animaatiot siirretään erikseen Unityyn. Mallin voi teksturoida heti UV-mappauksen jälkeen, mutta itselläni teksturointi oli jatkuva prosessi, ja palasin maalaamaan hahmon tekstuuria useasti muiden työvaiheiden

ohessa. Seuraavissa kappaleissa käyn tarkemmin läpi nämä hahmomallinnuksen työvaiheet, käyttäen Soluduksen keijuhahmoa käytännön esimerkkinä.

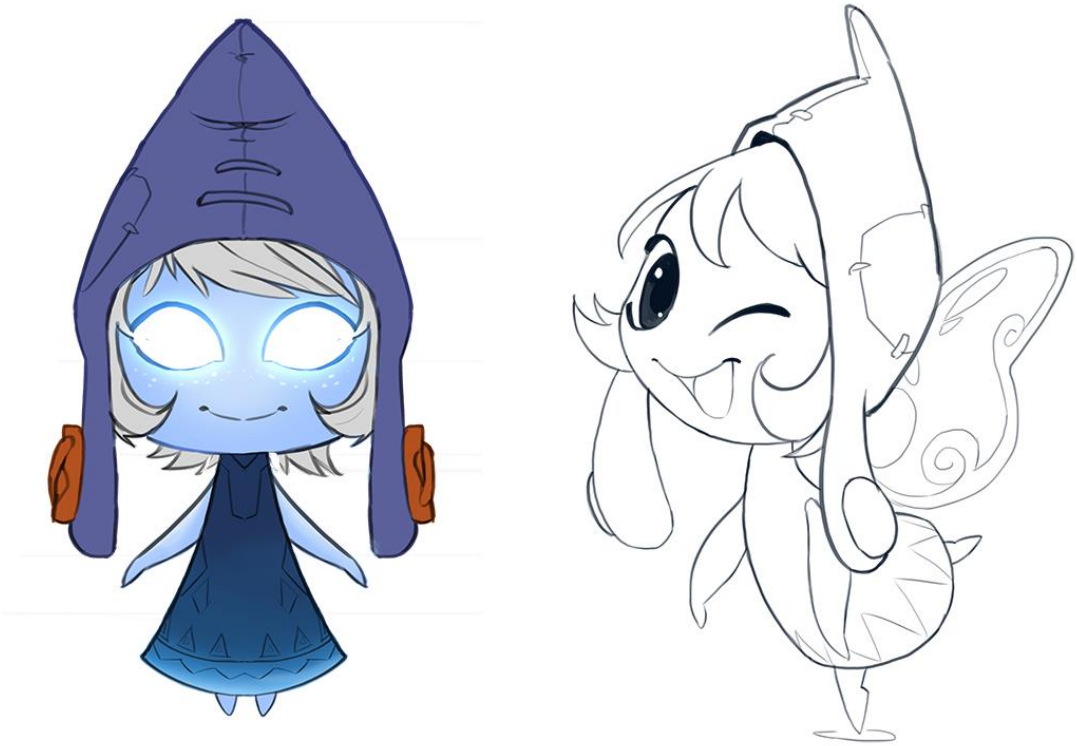
3.3 Konseptointi

Konseptitaiteella tarkoitetaan valmistettavan pelin tai elokuvan visuaalista suunnittelua ennen varsinaista tuotantoa. Konseptitaiteilijan työnä on kääntää pelisuunnittelijoiden ja elokuvaohjaajien ideat visuaalisiksi. Parhaimmissa tapauksissa tuotettu konseptitaide johtaa täysin uusiin ideoihin. (Creative Bloq 2012.)

Tullessani mukaan Soludus-projektiin, keijuhahmo oli jo jokseenkin vakiintunut. Eri hahmovaihtoehdoista oltiin päädytty nimenomaan keijuun, ja hahmosta oli valmiina toisen 3D-artistin tekemä tilapäinen malli. Tehtävänäni oli siis pikemminkin iteroida keijuhahmoa, sen sijaan että aloittaisin puhtaalta pöydältä. Tilapäistä mallia käytettiin jo pelissä, joten hahmon mittasuhteita ei kannattanut muuttaa valtavasti. Päätin myös säilyttää joitain tilapäisen mallin aspekteja, kuten hiippalakin ja siivet. Tältä pohjalta lähdin piirtämään hahmolle uutta visuaalista ilmettä Adobe Photoshopissa (kuvio 7). Digitaalisessa piirtämisessä etuna on se, että valmiita piirustuksista voi nopeasti tehdä kopioita, joita sitten on helppo muutella. Näin voi nopeasti kokeilla monia erilaisia ideoita hahmon ulkonäölle.

Kuvio 7. Ensimmäiset konseptit keijusta.

Alustavissa sketseissäni kokeilin keijulle erilaisia hiustyylejä sekä silmiä. Kokeilin myös erilaisia värejä, vaikka väritystä ei tarvitse lyödä lukkoon tässä vaiheessa. 3D-mallissa pinnan väritys lisätään vasta myöhemmin, ja sitä on aina helppo muuttaa halutessa. Sketseissäni pyrinkin siis pikemminkin saamaan aikaan silmää miellyttävät mittasuhteet ja yleisilmeen. Suuri pää ja suuret silmät tekevät hahmosta huomiota herättävän ja lapsenomaisen, jonka ajattelin sopivan hyvin päiväkoti-ikäisille lapsille suunnattuun peliin. Kuviossa 8 näkyy, miten hahmon ulkonäkö kehittyi tehdessäni lisää luonnoksia.



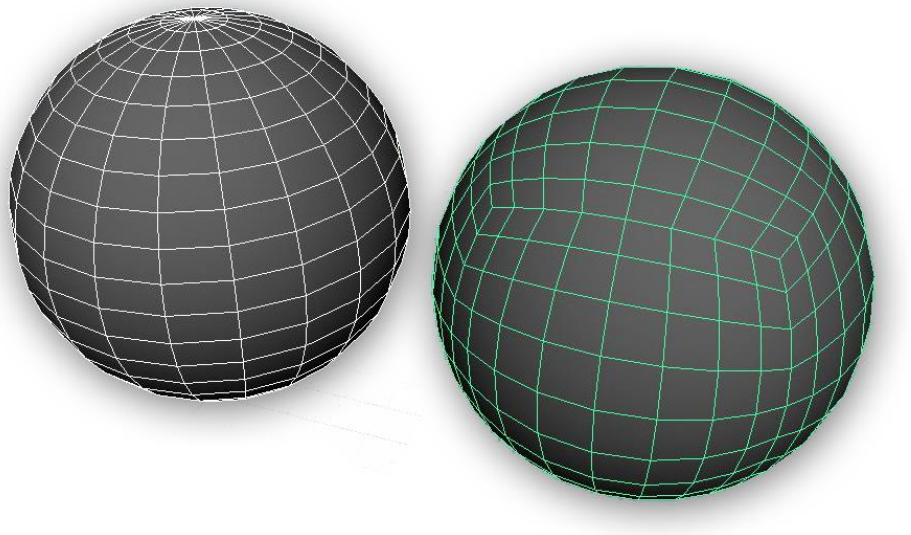
Kuvio 8. Keijun hahmosuunnittelun iterointia.

Jatkoin hahmon piirtämistä, kunnes olin tarpeeksi tyytyväinen mittasuhteisiin että koin voivani aloittaa hahmon 3D-mallinnuksen piirustusten pohjalta. Piirsin hahmosta myös toiminnallisemman kuvan, jolla toin esille keijun leikkisää persoonaa. Keijulla ei ollut mallinnukseen käytetyissä konsepteissa pupilleja, mutta toin ne myöhemmin takaisin. Keijun visuaalinen ilme kehittyi muutenkin useasti tekoprosessin aikana. Lisäsin esimerkiksi hiippalakkiin kulkusen, sekä mekon kuviointi muuttui. Mekko oli pitkään vihreä, kunnes päätin taas tehdä siitä sinisen.

3.4 Mallinnus

Peleissä esiintyvät 3D-mallit koostuvat pisteistä, joita kutsutaan vertekseiksi. Yhdestä verteksistä toiseen kulkevaa viivaa kutsutaan edgeksi. Kun kolme verteksiä yhdistetään toisiinsa edgeillä, niiden väliin muodostuu pinta, jota kutsutaan yksinkertaisesti kolmioksi. Pintoja, jotka koostuvat yli kolmesta verteksistä, kutsutaan polygoneiksi. Neljästä verteksistä koostuvaa pintaa kutsutaan quadiksi. (Wikipedia.) Quadin voi aina jakaa kahteen kolmioon. Pelimoottoreissa polygonit jaetaan aina kolmioiksi, joten malliin ei suositella käytettävän muita kuin kolmioita ja nelikulmaisia polygoneja. Lopputuloksena syntyy lähinnä nelikulmaisista polygoneista koostuva verkko, jossa saattaa olla jonkin verran kolmioita halutuissa paikoissa.

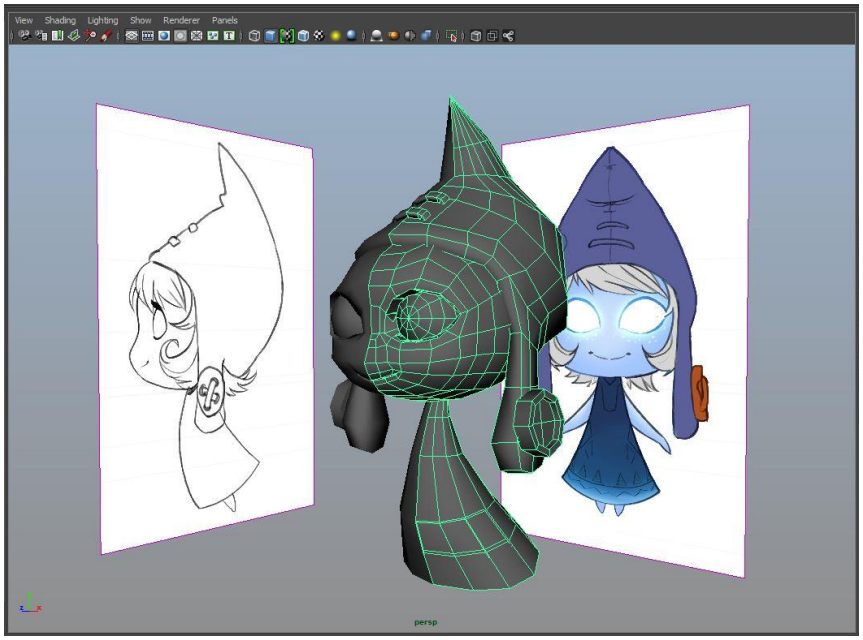
Tapaa, jolla verkko on aseteltu mallin pinnalle, kutsutaan mallin topologiaksi. Polygoneista koostuvia rinkuloita kutsutaan loopeiksi. Animoitavissa pelimalleissa suositaan ”siistiä” topologiaa, joka tarkoittaa esimerkiksi sitä, että pinnalta löytyy looppeja halutuista paikoista, kuten silmien ja suun ympäriltä. Tämä sallii sen, että nämä alueet ovat helpompia animoida. Topologiassa tulee myös ottaa huomion verkon suhteellinen tiheys mallin eri osissa. Hahmoilla etenkin kasvoihin kiinnittyy eniten huomiota, joten kasvoissa kannattaa olla tiheämpi polygoniverkko kuin esimerkiksi jalkapohjissa. Tiheämpi verkko sallii suuremman määrän yksityiskohtia sekä pyöreyttä. Kuviossa 9 havainnollistetaan topologian merkitystä 3D-malleissa.



Kuvio 9. Kahdella pallolla voi olla sama muoto, mutta eri topologia.

Hahmon 3D-mallin voi toteuttaa monella eri ohjelmistolla. Yleisimpiä ovat esimerkiksi Autodeskin Maya ja 3DS Max, sekä ilmainen Blender. Itse olen tottunut eniten Mayaan, ja se minulla oli käytössä studiolla. Keiju on siis toteutettu Mayalla, mutta 3D-mallinnuksen työtavat ja periaatteet ovat laajalti riippumattomia käytetystä ohjelmistosta.
Aloitin mallinnuksen sijoittamalla tekemäni konseptipiirrokset keijusta 3D-tilaan. Olin piirtänyt hahmon suoraan edestä sekä sivulta, ja varmistanut, että kuvissa hahmolla on täysin samat mittasuhteet. Näin pystyin vertailemaan rakentamaani 3D-mallia tekemiini piirustuksiin jatkuvasti mallinnuksen aikana.

Aloitin mallinnuksen keijun päästä. Sain haluamani topologian aloittamalla laatikosta, jonka sitten pyöristin jakamalla pinnan verkkoa tiheämmäksi. Sitten jouduin käsin luomaan polygoniloopit silmien ja suun ympärille. Koin tämän tärkeäksi, koska halusin saada hahmon räpyttelemään silmiään ja avaamaan suutaan myöhemmin Unityn puolella. Muut keijun muodot sain helposti aikaan aloittamalla Mayan tarjoamista perusmuodoista, kuten kartioista ja sylintereistä. Silmien tuli olla täydellisiä palloja, etteivät ne muljahtaisi oudosti silmäkuopassa animoitaessa. Mallinnuksen aikana jaoin hahmon keskeltä kahtia ja poistin toisen puoliskon, jotta voisi keskittyä yhteen puoliskoon, jonka peilasin sitten toiselle puolelle (kuvio 10).



Kuvio 10. Keijun mallinnuksen alkuvaiheet.

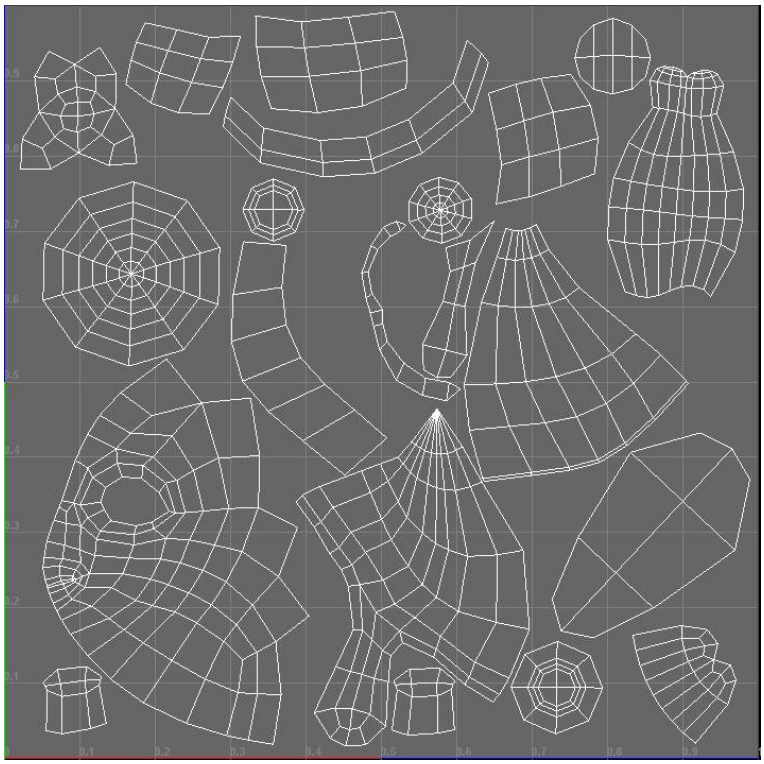
Siivet, hiukset ja ripset syntyivät yksittäisistä polygoneista, joille en tarvinnut lainkaan paksuutta. Hiusten ja siipien yksityiskohdat syntyivät alpha-tekstuureilla, joista lisää seuraavassa luvussa.

3.5 Teksturointi

3.5.1 UV-mappaus

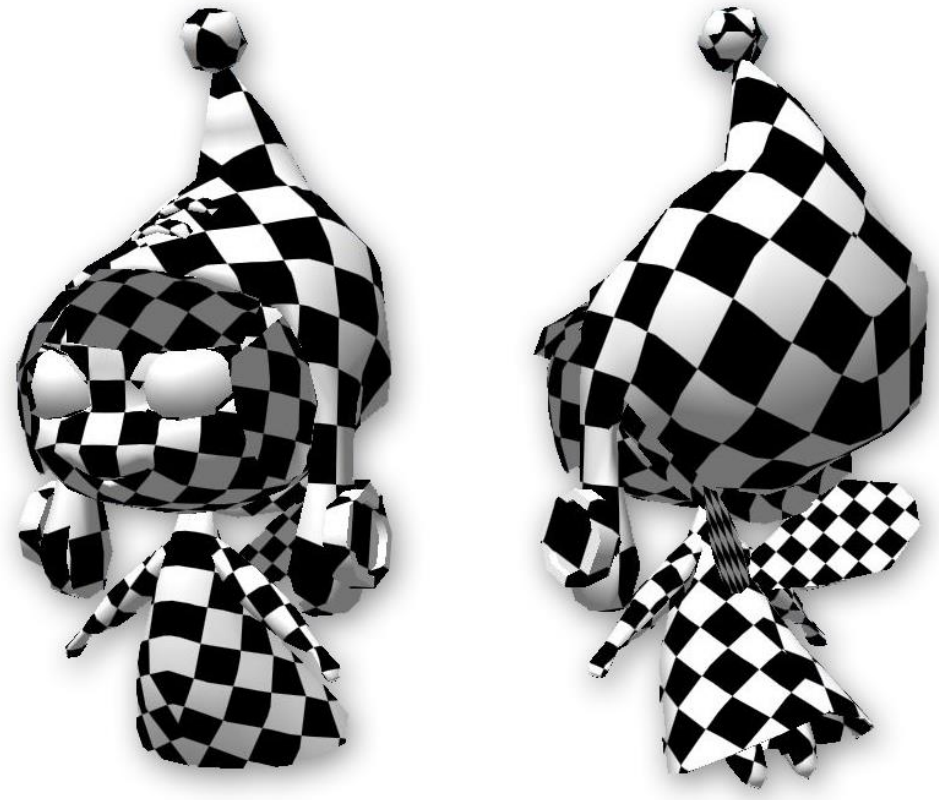
Ennen kuin 3D-mallin pintaan voi laittaa värejä ja materiaaleja, mallille tulee määrittää ns. UV-mappi. Kirjaimet U ja V ovat yksinkertaisesti kaksiulotteisen koordinaatiston koordinaatteja, kun kirjaimet X, Y ja Z ovat jo käytössä. Kun 3D-malli UV-mapataan, tämä tarkoittaa sitä, että mallin kolmiulotteinen pinta projisoidaan kaksiulotteiseen koordinaatistoon. Tämän jälkeen pinnan voi värittää kuvatekstuureilla. Monimutkaisten 3D-mallien projisointi ei kuitenkaan voi koskaan onnistua täydellisesti. Helppo vertauskuva on maapallosta tehdyt karttaprojektiot. Koska pallon pintaa ei voi pakottaa litteäksi ilman että pinta venyy tai repeää, maapallosta tehdyt kartat ovat usein harhaanjohtavia. Esimerkiksi mantereiden suhteelliset koot eivät vastaa täysin todellisuutta, koska kartta on paikoin venynyt. Sama rajoite pätee 3D-malleihin. Jotta pinta ei venyisi, 3D-malliin tulee päättää saumakohdat, joista pinta ”repeää”. Hahmo tavallaan nyljetään, ja lopullinen UV-mappi on kuin eläimen talja.

Aloitan UV-mappauksen jälleen poistamalla hahmon toisen puoliskon. Tämän voi tehdä silloin, kun hahmon kuvatekstuuri saa olla symmetrinen. En mielestäni tarvinnut hahmoon erityisesti epäsymmetriaa hiuksia lukuun ottamatta, joten pystyin työstämään vain yhtä puoliskoa. Kun toinen puolisko peilataan takaisin, sillä on samat UV-koordinaatit kuin työstetyllä puoliskolla. Keijun valmis UV-kartta on nähtävillä kuviossa 11.



Kuvio 11. Keijun UV-kartta.

Sen jälkeen teen yksinkertaisen projisoinnin jokaiselle hahmon eri osalle. Pää, käsivarsi, silmä, hattu, jalka ym. saavat kaikki oman UV-alueensa. Nämä ovat kuitenkin hyvin epätäydellisiä, koska niissä ei ole ollenkaan saumoja eikä niitä ole vielä levitetty tasaisiksi. Esimerkiksi käsivarsi on yhdestä päästä umpinainen sylinteri, joten sitä ei voi levittää tasaiseksi sellaisenaan. Pyrin laittamaan saumat sellaisiin paikkoihin, joista pelaaja ei huomaa niitä, eli yleensä hahmon selkäpuolelle tai alle. Tehdäkseni sauman valitsen haluamani polygoniverkon edget ja leikkaan käden UV-kartan siitä kohtaa Mayan työkaluilla. Tämän jälkeen Maya osaa levittää UV-alueen mahdollisimman litteäksi pinnaksi, jossa on mahdollisimman vähän venymistä. Muista 3D-ohjelmista löytyy vastaavat toiminnot. UV-mapin toimivuuden voi tarkastaa laittamalla mallille jonkin toistuvan kuvatekstuurin. Mayassa on valmiina mustavalkoinen shakkiruudukko, jonka laitan mallin pinnalle (kuvio 12).



Kuvio 12. Ruudukolla voi tarkistaa UV-kartan toimivuuden.

UV-mappi toimii, kun pinnalla näkyy tasasivuisia neliöitä, joissa ei esiinny paljolti venymistä.

3.5.2 Tekstuurin maalaus

Pelihahmolle tulevat tekstuurit riippuvat suuresti pelin tyylistä. Realistisilla pelimalleilla on pelkän värin lisäksi aimo määrä muita kuvatekstuureja, joilla säädellään materiaalin ominaisuuksia. Sarjakuvaimaisen tyylitellyissä peleissä taas on usein pelkkä käsinmaalattu kuvatekstuuri, johon on tavallaan jo maalattu pinnan varjostus. Päätin, että tämä työskentelytapa sopisi paremmin Soludus-projektiin.

En kuitenkaan maalannut tekstuuria täysin käsin, vaan aloitin viemällä UV-mapatun malliin ohjelmaan nimeltä xNormal. xNormal on lähinnä tarkoitettu siirtämään high poly -mallien yksityiskohdat low poly -mallien tekstuureihin, mutta sillä on myös muita käyttötarkoituksia. Yksi näistä on Ambient Occlusion(AO)-tekstuurien tuottaminen. AO-teks-

tuuri piirtää mallin pintaan yleisen, pehmeän varjostuksen. Tämä tarkoittaa sitä, että mallissa oleviin rakoihin ja muihin ahtaisiin paikkoihin piirtyy pehmeä varjo, kun taas avoimet alueet pysyvät valkoisina. Käytän tätä pohjanani varsinaisen tekstuurin maalaamiseen Photoshopissa.

Photoshopissa asetan AO-tekstuurin omaksi tasokseen, jonka säädän Multiply-sekoitustilaan. Tällöin tekstuurin tummat alueet tummentavat alla olevia tasoja, kun taas valkoisilla alueilla ei ole mitään vaikutusta. Sitten voin alkaa asettelemaan jokaiselle UV-alueelle haluamani värin. Perusvärien jälkeen alkaa varsinainen maalausprosessi, jossa lähinnä lisäilen erilaisia yksityiskohtia sekä lisään tummien ja vaaleiden alueiden välistä kontrastia. Kuvittelen, että keiju olisi valaistustilanteessa, jossa valo tulisi pehmeästi ylhäältä ja edestä. Maalaan siis lisää varjoa leuan alle ja selkään. Koska hahmoon ei tule Unityssa valaistusta, tekstuurissa itsessään tulee olla pehmeä siirtyminen valosta varjoon. Muulloin hahmo näyttäisi litteältä. Maalaan myös silmiin valmiit heijastukset.
Hiusten ja siipien alueelta jätän joitain kohtia läpinäkyviksi. Kun tekstuuri tallennetaan png-muotoon, nämä läpinäkyvyydet tallentuvat kuvan alpha-kanavaan, jolloin niistä tulee myös läpinäkyviä Unityssa. Keijun lopullinen tekstuuri näkyy kuviossa 13.

Kuvio 13. Lopullinen käsinmaalattu tekstuuri.

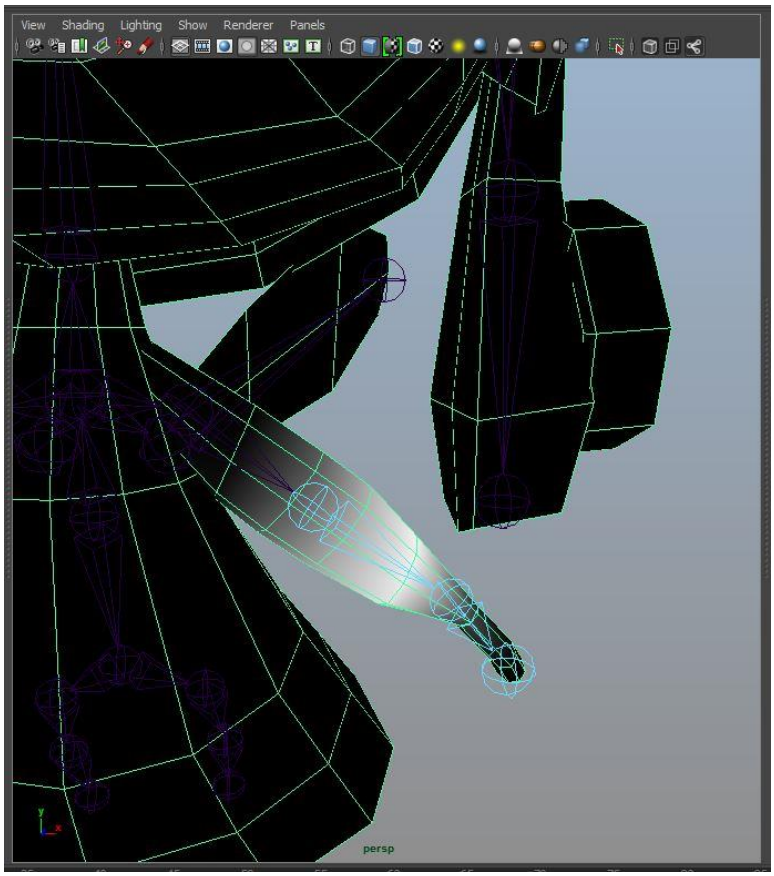
Tekstuurin koko tulisi olla neljällä jaollinen. Yleisimpiä kokoja pelitekstuureille ovat 512x512, 1024x1024 ja 2048x2048 pikseliä. Mitä isompi tekstuuri, sitä enemmän se raskauttaa peliä. Maalasin keijun tekstuurin 2048x2048-kokoon, sillä ison tekstuurin voi aina kutistaa myöhemmin, jos se todetaan liian raskaaksi pelille. Pienen tekstuurin suurentaminen pehmentäisi tekstuuria.

3.6 Rigaus

Rigauksella tarkoitetaan prosessia, jolla 3D-malli valmistellaan animoitavaksi. Mallille rakennetaan luuranko, jonka luut liikuttavat mallin eri osia (kuvio 15). Mallista tehdään siis eräänlainen nukke, jota voi asetella eri asentoihin. Tallentamalla asennot saadaan aikaan animaatiota.

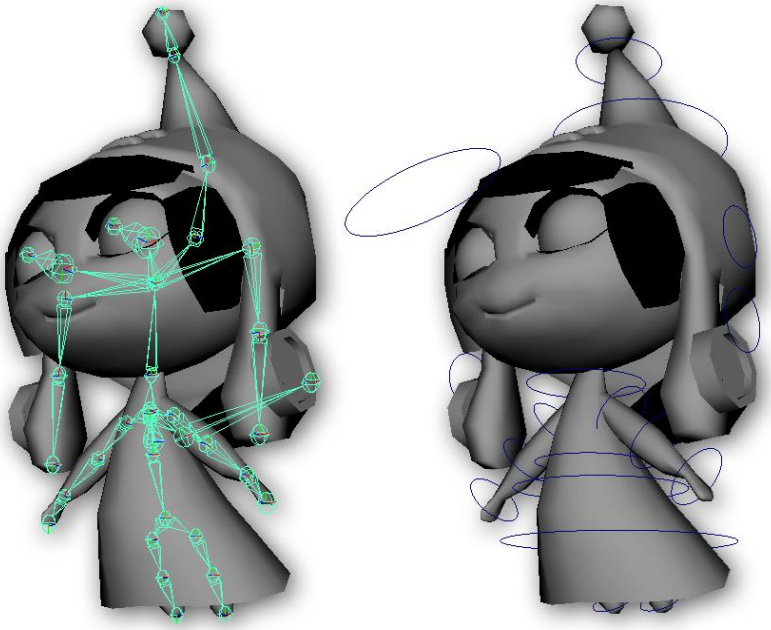
Keijun rigin rakentaminen alkoi luiden eli jointien asettelulla. Jointit muodostavat hierarkian, jossa alemmat jointit liikkuvat ylempien mukana. Ranteen luu siis liikkuu kyynärpään luun mukana, ja kyynärpään luu liikkuu olkapään mukana, ja niin edelleen. Luiden asettelussa on tärkeää, että ne taittuvat halutulla tavalla. Aloitin yksinkertaisista ketjuista, kuten käsivarsista ja selkärangasta. Tehtyäni yhden käsivarren jointit, pystyin peilaamaan ne toiselle puolelle. Tämän jälkeen siirsin käsien ketjut selkärangan ketjun hierarkiaan, selän jointien alapuolelle. Tällöin luustosta tuli yhtenäinen. Luurangon hierarkian ylintä jointia kutsutaan rootiksi, eli juureksi. Toistamalla tätä prosessia sain luurangon nopeasti kasattua. Silmien jointien kanssa täytyi olla tarkka. Jotta silmä pyörisi kuopassaan halutulla tavalla, jointin tulee olla täysin keskellä silmän palloa. Käytin tähän apuna silmän verteksien koordinaatteja, sen sijaan, että olisi sijoittanut jointit silmämääräisesti. Asetin myös useita luita keijun hattuun, jotta voisin saada sen myöhemmin heilumaan pään liikkeiden johdosta.

Kun luuranko oli valmis, se piti liittää 3D-malliin. Tätä kutsutaan skinnaukseksi. Skinnauksen ajatuksena on määritellä, mihin kohtiin 3D-mallia mikäkin luu vaikuttaa, ja millä voimakkuudella. Mayan automaattinen skinnaustoiminto kykenee arvioimaan suurin piirtein jokaisen luun todennäköisen vaikutusalueen, mutta tällä ei ikinä saa aikaan täydellistä lopputulosta. Automaattisen skinnauksen jälkeen vaikutusalueita tulee säätää käsin. Kuviossa 14 näkyy, kuinka skinnaus määrittelee luiden vaikutusalueet.



Kuvio 14. Keijun kyynärpään luun vaikutusalue näkyy valkoisella.

Kun skinnaus oli valmis, rigille piti vielä tehdä kontrolliobjektit (kuvio 15). Nämä ovat yleensä rinkuloita tai muita yksinkertaisia muotoja, jotka säädetään ajamaan jointien rotaatioita eri tavoin. Ajatuksena on se, että animaattorin ei tarvitse tarttua jointeihin animoidakseen hahmoa. Kontrolleille voi myös asettaa muita ominaisuuksia. Laitoin keijulle esimerkiksi rinnan kontrolliobjektille numeerisen attribuutin, jolla pystyin ajamaan molempien siipien rotaatioita samanaikaisesti. Siivet liikkuvat usein symmetrisesti, joten tämä nopeuttaa animointia.



Kuvio 15. Keijun luuranko sekä kontrolliobjektit.

Silmien räpyttelyä varten tein keijun kasvoista kopion, jolle muokkasin silmät suljetuiksi. Tätä kutsutaan blend shapeksi. Kun hahmon malli viedään lopulta Unityyn, blend shapet säilyvät mukana, ja niitä kykenee animoimaan Unityssa.

3.7 Animointi

Keijua animoidessa yritin pitää mielessä animaation 12 periaatetta, tai ainakin ne, jotka pätivät keijun animointiin. Animaation 12 periaatetta ovat lähtöisin vanhojen Disney-mestareiden kirjasta Illusion of Life, ja kuuluvat seuraavanlaisesti:

- Squash and stretch
- Anticipation
- Staging
- Straight ahead and pose to pose
- Follow through and overlapping action
- Slow in and slow out
- Arcs
- Secondary action
- Timing

- Exaggeration
- Solid drawing
- Appeal

(Animation Toolworks.) Näistä keijun animointiin oleellisimmat olivat Anticipation, Follow through and overlapping action ja Arcs. Rigin rajoitteista johtuen hahmoa ei voinut venyttää tai litistää, joten esimerkiksi Squash and stretch ei ollut tehtävissä.

Anticipation tarkoittaa sitä, että ennen kuin hahmon tekee jonkin suuren liikkeen, sen tulee ensin tehdä pienempi, valmistautuva liike. Tällöin katsoja osaa hieman ennakoida tulevaa suurempaa liikettä, jolloin hahmon liike ei vaikuta äkkipikaiselta ja satunnaiselta. Follow through tarkoittaa sitä, että hahmon liikkeen pysähtyessä jokainen hahmon osa ei pysähdy samanaikaisesti. Esimerkiksi hiukset, vaatteiden helmat ja jäniksen korvat jatkavat hieman liikettä sen jälkeen, kun hahmon massakeskipiste on jo pysähtynyt. Overlappia voi taas havaita samoissa hahmon osissa, jos hahmon liike yllättäen vaihtaa suuntaa. Arkeilla tarkoitetaan sitä, että hahmon liikkeiden tulisi kulkea siisteissä kaarissa. Tämä tekee liikkeestä luonnollisen oloisen. (Animation Toolworks.)

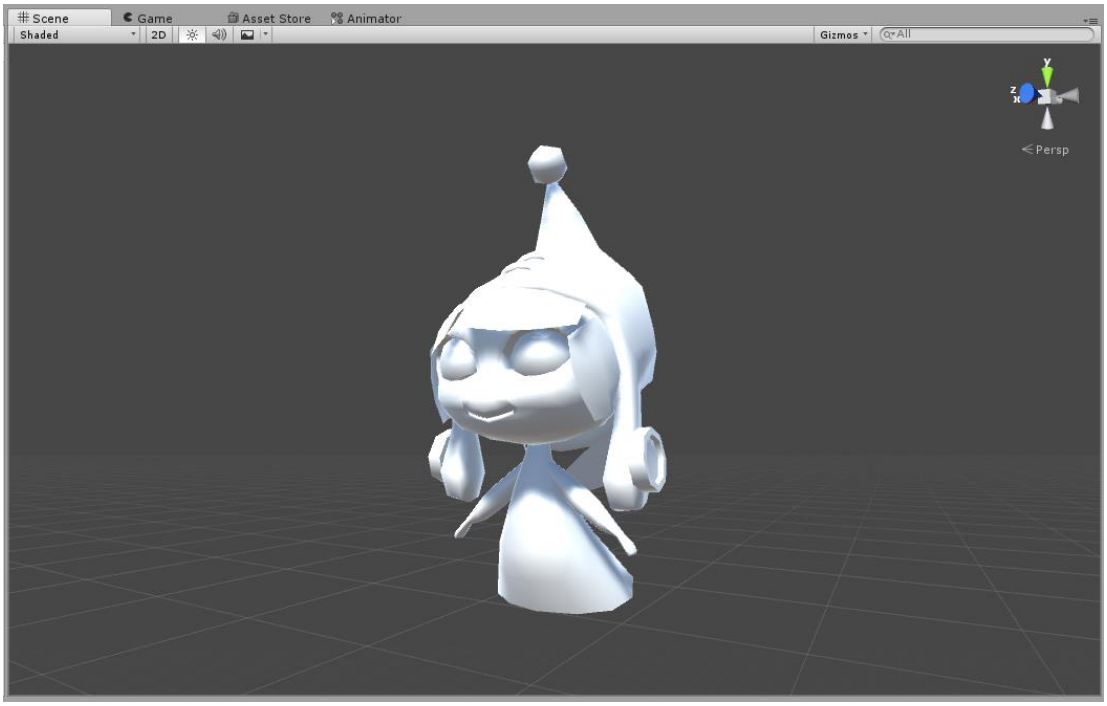
Keijulle tuli ensin idle-animaatio, jossa keiju vain leijuu paikallaan. Aloitin animoinnin liikuttamalla keijua ensin hiukan ylös ja alas. Tämän jälkeen laitoin keijun muut osat seuraaman tätä liikettä. Halusin, että keijun pää heiluisi hennosti liikkeen jäljessä, pitäen Overlap-periaatteen mielessä. Hiippalakki taas seurasi pään liikettä hieman jäljessä, ja hiippalakin kärjessä oleva kulkunen tuli viimeisenä. Keijulla tuli myös juhlimisanimaatio, jossa keiju hypähtää ja pyörii kerran ympäri. Tässä animaatiossa tuli ottaa huomioon liikkeen kaaret sekä Follow through, kun hiippalakin läpät ja kärki jatkoivat liikettä keijun itse jo pysähdyttyä.

4 Pelihahmo Unityssa

4.1 Hahmon tuonti Unityyn

Jotta hahmo voidaan tuoda Unityyn, se pitää ensin tallentaa ulos Mayasta oikeassa muodossa. Unity tukee useita eri 3D-formaatteja, kuten esimerkiksi .fbx, .dae, .3ds ja .obj-formaatteja. Suoritin tämän vaiheen seuraamalla Youtubesta löytyvää tutoriaalia. Tallen-

sin ensin Mayasta ulos keijun mallin ja rigin luurangon yhtenä .fbx-tiedostona, jonka nimesin yksinkertaisesti Keiju.fbx. Rigin kontrolliobjekteja ei tarvinnut, ja blend shapet tulevat mallin mukana. Seuraavaksi tallensin jokaisen animaation erikseen ulos Mayasta, myöskin .fbx-formaatissa. Animaation tallentamiseksi animaatiodata piti ensin siirtää kontrolliobjekteista suoraan luihin käyttämällä Mayan Bake Simulation-toimintoa. Sitten tallensin animaatiot ulos muodossa Keiju@AnimaationNimi.fbx. Kun malli ja animaatiot on nimetty näin, Unity tunnistaa heti, mille mallille animaatio kuuluu. (Toke Jepsen 2013.) Assettien tuominen Unityyn toimii joko valikosta Assets > Import New Asset, tai vaihtoehtoisesti mallin ja animaatiot voi yksinkertaisesti raahata hiirellä Project-ikkunaan. Tuon tässä vaiheessa myös maalamani kuvatekstuurin Unityyn. Kun assetti on Project-ikkunassa valittuna, sen tuontiasetuksia voi tarkastella Inspector-ikkunasta. Kuviossa 16 näkyy keijun 3D-malli Unityssa.

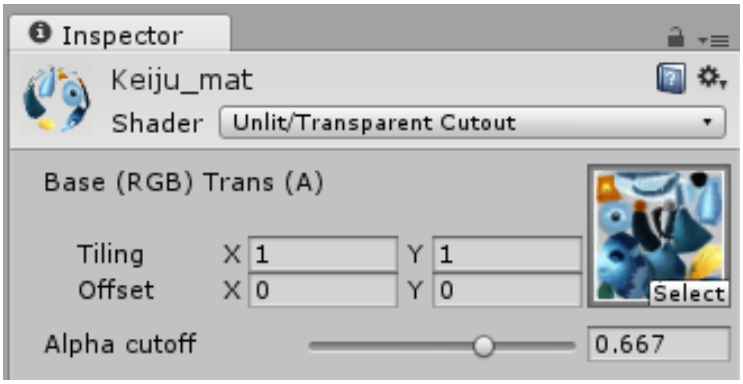


Kuvio 16. Keiju Scene-näkymässä.

Kun keijun malli on Project-ikkunassa, voin raahata sen Scene-näkymään. Tällöin Hierarchy-ikkunaan syntyy GameObject, joka sisältää keijun mallin sekä luurangon. Nimeän GameObjectin Keijuksi, ja siirrän sen Scenen origoon sijoittamalla Transform-komponentin koordinaateiksi 0,0,0. Keijulle tulee kuitenkin aluksi valkoinen Standard-materiaali, joka tulee vaihtaa.

4.2 Materiaalin luonti

Materiaalin voi luoda valitsemalla valikosta Assets > Create > Material. Tämä luo Project-ikkunaan uuden materiaalin, jossa on alkuun Unityn standard-shader. Saadakseni kuitenkin haluamani käsinmaalatun ulkomuodon hahmolle, valitsen Inspector-ikkunassa materiaalille shaderin nimeltä Unlit / Transparent Cutout (kuvio 17). Unlit tarkoittaa sitä, että materiaalin pinnan väri ei reagoi mitenkään Scenessä oleviin valoihin, vaan sitä ajaa pelkästään maalaamani kuvatekstuuri. Transparent Cutout taas tarkoittaa sitä, että kuvatekstuurini läpinäkyvät osat ovat myös läpinäkyviä 3D-mallissa. Shaderin Alpha cutoff-asetuksella voi säätää läpinäkyvyyttä.



Kuvio 17. Keijun materiaali.

Kun materiaali on valmis, sen voi laittaa mallille Scene-näkymässä. Ensin malli sijoitetaan Sceneen, ja sitten materiaalin voi raahata sen päälle. Kuviossa 18 näkyy keiju, jolla on asettamani materiaali sekä tekstuuri.

Kuvio 18. Keiju tekstuureineen.

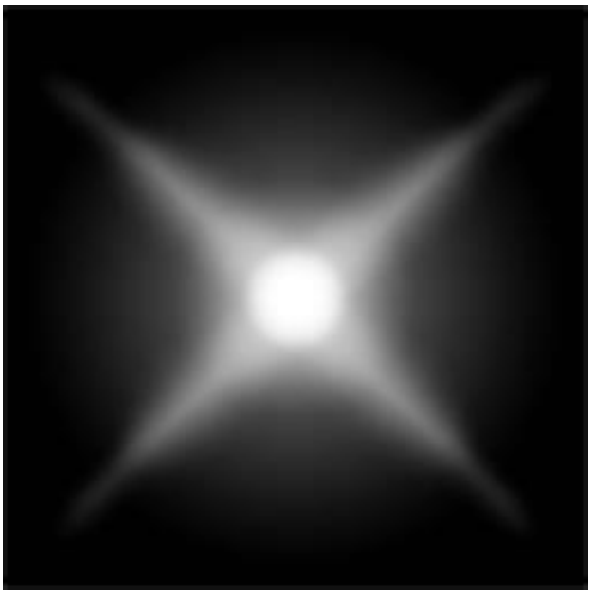
4.3 Animaatioiden liittäminen hahmoon

Animaatioiden liittämiseksi täytyy ensin luoda Animator Controller-assetti, jonka ominaisuuksia voi sitten tarkastella Animator-ikkunasta. Animator-ikkunaan tuodaan hahmon animaatiot. Valitsemalla keijun Hierarchy-ikkunassa, saan Inspector-ikkunaan näkymän, jossa voin linkittää mallin Controlleriin. Tässä vaiheessa luon myös muutaman uuden animaation keijulle Unityn puolella. Tehdäkseni tämän, valitsen valikosta Assets > Create > Animation. Raahaan uuden, tyhjän animaation myös keijun Animator Controlleriin, ja annan sille nimeksi Blink. Blink-animaatiota voi tarkastella Animation-ikkunasta. Painamalla nappia Add New Property, saa esiin kaikki hahmon komponentit joita voi animoida kuten esimerkiksi Transform-komponentin sekä keijun luurangon. Haluan kuitenkin Blink-animaatiolle hakea keijun malliin tallentuneen silmien blend shapen, jonka tein aikaisemmin Mayassa. Blend shapea voi sitten animoida syöttämällä sille arvoja. Arvossa 0 keijun silmät ovat täysin auki ja arvossa 100 ne ovat kiinni. Teen keijulle muutaman silmän räpäytyksen, sekä laitan Inspector-ikkunassa animaatiolle ruksin kohtaan

Loop Time. Tämä varmistaa sen, että animaatio toistuu loputtomiin. Luon Animator-ikkunassa uuden animaatiotason, jonne siirrän Blink-animaation. Tason asetuksista säädän weightin arvoon 1 sekä Blending-asetuksen tilaan Additive. Tällöin silmien räpyttely yhdistyy sulavasti muihin animaatioihin. Teen samalla tekniikalla silmille pienen liikkumisanimaation.

4.4 Partikkelien luonti

Loppusilauksena luon keijulle jonkinlaista tähtipölyä muistuttavan efektin. Partikkelit ovat Unityn tapa simuloida esimerkiksi räjähdyksiä, nesteitä tai kaasuja. Aloitan efektin luomalla tyhjän GameObjectin, jolle asetan Particle System –komponentin. Particle Systemissa voi määritellä partikkelien ominaisuudet, kuten elinajan, määrän, koon ja nopeuden. Partikkelit vaativat myös materiaalin. Luon siis uuden materiaalin, jonka shaderiksi asetan Particles/Additive. Photoshopin puolella maalaan tähden, jonka asetan materiaalin tekstuuriksi (kuvio 19).



Kuvio 19. Partikkelin tekstuuri.

Lopuksi asetan partikkelit sisältävän GameObjectin keijun hatun viimeisen luun lapseksi, ja asetan sen Transform-komponentin koordinaatit nollaan. Tällöin GameObject siirtyy hatun kulkusen sijaintiin, josta haluan partikkelien syntyvän. Partikkelin asetuksista tulee varmistaa, että Simulation Space -asetus on säädetty Localin sijasta Worldiin. Jos tätä ei muuta, jokainen partikkeli seuraa kulkusen heilumista luonnottomalla tavalla hahmon

liikkuessa. Halua, että partikkelit vain syntyvät kulkusen sijainnissa, mutta eivät sen jälkeen seuraa sitä. World-asetus tekee tämän.

4.5 Viimeistely

Tässä vaiheessa Scene-näkymässä on siis GameObject, johon linkittyy useampi assetti. Hahmo on kokonaisuus, joka koostuu 3D-mallista, luurangosta, materiaalista ja sen tekstuurista, animaatioista sekä partikkeliefektistä. Kun tämä GameObject raahataan Hierarchy-ikkunasta Project-ikkunaan, saadaan keijusta prefab-assetti. Projektia tehdessä en itse koskenut varsinaiseen peliin, vaan kokosin assetit omalla koneellani ja annoin ne ohjelmoijalle, joka sitten sijoitti assetit peliin. Tämä toimii helposti Export Package -toiminnolla. Valitsemalla keijun prefabin Project-ikkunasta ja painamalla valikosta Assets > Export Package, Unity tekee keijun prefabista ja kaikista sen asseteista siistin paketin, jonka ohjelmoija voi tuoda helposti peliin.

5 Yhteenveto

Pelihahmon voi valmistaa monella eri tavalla, ja oikea lähestymistapa riippuu täysin projektin luonteesta sekä hahmon roolista pelissä. Tässä opinnäytetyössä esitelty hahmo ei ole pelattava hahmo, eli siis pelaaja ei kykene suoraan kontrolloimaan sitä. Jos keiju olisi pelattava, sen kehityksessä tulisi ottaa eri asioita huomioon. Tulisi esimerkiksi ottaa enemmän huomioon, miltä hahmo näyttää kaikista mahdollisista kuvakulmista, eikä vain edestä. Keijulle tulisi myös taatusti huomattavasti enemmän animaatioita. Jos peli olisi taas kehitetty tehokkaille PC- ja konsolialustoille, keiju olisi huomattavasti raskaampi polygonimäärältään, sekä sen kehityksessä tulisi mahdollisesti käyttää moderneja veistoohjelmia kuten zBrush.

Tässä opinnäytetyössä esitelty lähestymistapa on siis melko suppea katsaus reaaliaikaisten pelihahmojen valmistamiseen. Kaikista mahdollisista eri tavoista ja tekniikoista voi taatusti kirjoittaa useamman kirjan verran informaatiota.
Projektia tehdessä opin kuitenkin paljon uutta, ja kohtasin haasteita. En ollut esimerkiksi rigannut hahmoa pitkään aikaan, ja tämä aiheutti vaikeuksia. Lopullinen rigi on toimiva, mutta ei täysin ideaali hahmon animointia varten. Sain kuitenkin pahimmat puutteet korjattua rigatessani toista hahmoa samaan peliin.

Mahdollisia suuntia kehitykselle on siis monta. Esimerkiksi hiippalakin läpät olisi voinut animoida fysiikkamallinnuksella, sen sijaan että animoin ne käsin. Tällä voisi tosin olla myös haittapuolensa. Myös kasvoihin olisi voinut laittaa enemmän luita blend shapejen sijaan. Otan nämä asiat huomioon mallintaessani tulevia hahmoja.

Lähteet

Pluralsight 2014. Unreal Engine 4 vs. Unity: Which Game Engine Is Best for You? <https://www.pluralsight.com/blog/film-games/unreal-engine-4-vs-unity-game-engine-best> (luettu 28.4.2017).

Haas, John 2002. A History of the of the Unity Game Engine. Worcester Polytechnic Institute. (luettu 28.4.2017).

Unity User Manual n.d. <https://docs.unity3d.com/Manual/> (luettu 28.4.2017).

Unity3D 2017. Unity 5.0 <https://unity3d.com/unity/whats-new/unity-5.0> (luettu 28.4.2017).

DualShockers 2014. The Order: 1886 Characters have over 100,000 Polygons, More than Ryse's Marius, but Less Blend Shapes <http://www.dualshockers.com/2014/02/20/the-order-1886-characters-have-over-100k-polygons-more-than-ryses-marius-but-less-blend-shapes/> (luettu 28.4.2017).

A+Start 2017. Super Mario – Low Poly (Evolution of Characters in Games) – Episode 1 <https://www.youtube.com/watch?v=A2gXyEyy_2U> (luettu 28.4.2017).

Creative Bloq 2012. Just what is concept art? <http://www.creativeblog.com/career/what-concept-art-11121155> (luettu 28.4.2017).

Wikipedia n.d. Low poly <https://en.wikipedia.org/wiki/Low_poly> (luettu 28.4.2017).

Wikipedia n.d. Polygonal modelling <https://en.wikipedia.org/wiki/Polygonal_modeling> (luettu 28.4.2017).

Animation Toolworks n.d. 12 principles. <http:/www.animationtoolworks.com/library/article9.html> (luettu 28.4.2017).

Toke Jepsen 2013. Maya animation to Unity.
<https://www.youtube.com/watch?v=0mnL-QKvpvY> (luettu 28.4.2017).

Kuvalähteet

Kuvio 2. Gooball.
<http://www.ambrosiasw.com/games/gooball/> (luettu 28.4.2017).

Kuvio 3. Animator controller.
<https://docs.unity3d.com/Manual/class-AnimatorController.html> (luettu 28.4.2017).

Keijun idle-animaatio
<https://vimeo.com/215506705>

Keijun juhlimisanimaatio
<https://vimeo.com/215528085>

Keijun tanssianimaatio
<https://vimeo.com/215528253>