3D-pelihahmon komponentit ja työvaiheet

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, insinööri (AMK)

Kevät 2022

Roope Autio

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Koulutuksen nimi — Tiivistelmä
Tekijä Roope Autio — Vuosi 2022
Työn nimi 3D-pelihahmon komponentit ja työvaiheet
Ohjaaja Antti Laakso

Opinnäytetyön tavoitteena oli tarkastella 3D-pelihahmon luomiseen liittyvät työvaiheet ja tekniikat. Työssä selvitettiin, miten näitä teknisiä ratkaisuja hyödynnetään uskottavan pelihahmon luomisessa.

Työssä käytiin läpi 3D-mallinnuksen perusperiaatteita ja muita 3D-visualisoinnin käsitteitä. Pelihahmon 3D-malli tuodaan pelimoottoriin, jotta sen voi muuttaa staattisesta 3D-mallista valmiiksi pelihahmoksi. Tätä varten tutustuttiin Unity-pelimoottorin perustoiminnallisuuksiin ja syvennyttiin Unitystä löytyviin pelihahmon toimintaan liittyviin työkaluihin.

Työn toiminnallisessa osassa tehtiin kaksi esimerkkipelihahmoa alusta alkaen. Hahmojen toteutuksessa hyödynnettiin tietopohjassa esiteltyjä tekniikoita. Hahmot luotiin niin että mahdollisimman paljon niiden toiminnallisuuksista pystytään käyttämään uudelleen toisissa pelihahmoissa.

Työn aikana nousi vahvasti esille se, miten paljon 3D-hahmon luomiseen tarvittavat työvaiheet ovat sidoksissa toisiinsa. Uskottavan pelihahmon luomista varten tarvitaan osaamista usealta eri alalta. Jokaisella pelihahmon tekemiseen osallistuvalla on hyvä olla ainakin perustason tietämys jokaisesta pelihahmon tekemiseen kuuluvasta työvaiheesta.

Avainsanat 3D-mallinnus, videopelit, animaatiohahmot
Sivut 35 sivua ja liitteitä 2 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Name of Degree Programme Abstract
Author Roope Autio Year 2022
Subject Components and stages of work of a 3D-game character
Supervisors Antti Laakso

The purpose of this thesis was to examine the techniques and stages of work that are needed in the creation of a 3D game character. This thesis will examine how these technical solutions can be used in creating a believable game character.

This thesis examines the basic principles of 3D modeling and other aspects related to 3D visualization. The model of the game character is imported into a game engine, to turn it from a static 3D model into a completed game character. To achieve this, some basic functionality of the Unity game engine was examined. As well as some specialized tools that are available in the Unity engine for creating game characters.

In the practical part of the thesis, two example game characters were created. Techniques examined in this thesis were used in their creation. The characters were created in a way that allowed for a maximum amount of their functionality to be repurposed in other characters.

Over the course of working on this thesis it became apparent just how much the various stages of work are connected to each other. Creating a believable game character requires understanding of several different areas of expertise. Each person who takes part in creating a game character should have at least a basic understanding of each stage of work that is required in the making of a game character.

Keywords 3D-modelling, videogames, animated characters
Pages 35 pages and appendices 2 pages

Sisällys

Kuvat

Liitteet

1 Johdanto

Tässä opinnäytetyössä käydään läpi 3D-pelihahmon luomiseen liittyvät työvaiheet. Videopeleissä pelihahmot asuvat pelin sisäisessä maailmassa. Pelihahmo voi olla pelaajan ohjaama pelin päähahmo, tai se voi olla taustahahmo, jonka tehtävä pelissä on tuoda uskottavuutta pelin maailmaan. 3D-pelihahmo on pelihahmo, joka on luotu 3D-grafiikoilla. 3D-hahmon toiminnallisuuteen tarvitaan 3D-malli, joka on hahmon muoto pelin maailmassa. 3D-mallilla voi olla materiaali, joka sisältää mallin pintakuviot. Jos 3D-hahmo halutaan animoida, on hahmolle tehtävä ”Riggaus” joka osoittaa miten hahmon 3D-malli taipuu ja venyy. Pelihahmo tarvitsee tietysti myös koodin, joka määrää miten hahmo käyttäytyy. Kaikki nämä työvaiheet tarvitaan, jos halutaan luoda valmis 3D-pelihahmo, joka on uskottava osa pelimaailmaa.

Hyvä pelihahmo on pelin rahallisesti arvokkain osa. Pelihahmo voi olla pelin maskotti, jolla peliä markkinoidaan. Hyvää pelihahmoa voi kaupata erillään pelistä. Hahmon kuvan voi liimata t-paitoihin ja kahvimukeihin. (Adams, 2014, s. 182)

Super Mario on hyvä esimerkki suositusta pelihahmosta. Mario on ollut suosittu pelihahmo 2D-pikseligrafiikoiden ajoista lähtien. Hahmon suosio on jatkunut vahvana senkin jälkeen, kun videopelit siirtyivät 3D-grafiikkaan. Super Marion tunnistavat myös ihmiset, jotka eivät ole koskaan pelanneet yhtäkään videopeliä. Mario on uskottava osa omaa pelimaailmaansa. Hän on visuaalisesti yhdenmukainen pelimaailman kanssa. Myös Marion animaatiot auttavat uskottavuudessa. Animaatioiden avulla hän ei ole pelkästään liikkuva 3D-malli, vaan uskottava osa pelimaailmaa. Kun peliohjaimesta painetaan hyppynappia, Marion malli nousee ilmaan ja samalla se toistaa hyppyanimaation. Hyppyanimaatio saa Marion tuntumaan siltä, että hän on ponnistanut maasta ja loikannut ilmaan, eikä vain ottanut komentoa peliohjaimesta ja nostanut pelihahmon mallia ylöspäin. Uskottavuuteen auttaa myös erilaiset efektit. Juostessaan Mario jättää jälkeensä pölypilviä, jotka ilmaisevat, että hän on osa pelimaailmaa ja hänen jalkansa nostattavat tomua maasta.

Tässä opinnäytetyössä selvitetään valmiin 3D-pelihahmon luomiseen tarvittavat työvaiheet teknisestä näkökulmasta. Tekniset ratkaisut auttavat tuomaan pelihahmon eloon, ja muuttamaan hahmon staattisesta 3D-mallista, pelihahmoksi, joka tuntuu luontevalta osalta pelin maailmaa.

Tämän opinnäytetyön toiminnallisessa osassa toteutetaan demo, jossa luotua 3D-pelihahmoa ohjataan 3D-ympäristössä. Valmis 3D-pelihahmo pyritään tekemään uskottavaksi osaksi pelimaailmaa, tekemällä sille animaatiot ja ohjelmoimalla hahmo toistamaan ne sopivaan aikaan.

3D-hahmoa tehdessä kiinnitetään huomiota siihen, miten hahmon eri ominaisuuksia voi hyödyntää uudelleen. Jos pelissä on useita eri hahmoja, niillä jokaisella on tietty määrä yhtenäisiä ominaisuuksia. Hahmon tekeminen alusta alkaen jokaiselle pelihahmolle on vaivalloista, ja se ei ole kustannustehokasta. Kiinnittämällä huomiota hahmon modulaarisuuteen, voidaan luoda hahmo, jonka ominaisuuksia ja piirteitä voi helposti vaihtaa ja muokata. Näin pystytään tekemään useita eri hahmoja samasta pohjasta.

2 Hahmon mallinnus

Tässä osassa tarkastellaan mitä tarvitaan hahmon 3D-mallin tekemiseen. Pelihahmon malli luodaan 3D-mallinnusohjelmalla. Blender on ilmainen 3D-mallinnusohjelma. Kuka tahansa voi ladata sen, ja sen käyttö on vapaasti sallittua henkilökohtaisissa ja kaupallisissa tuotannoissa. Tämän työn esimerkit on tehty Blender-ohjelmalla, mutta työvaiheet on pyritty esittämään niin että perusperiaatteet soveltuvat mihin tahansa 3D-mallinnusohjelmaan.

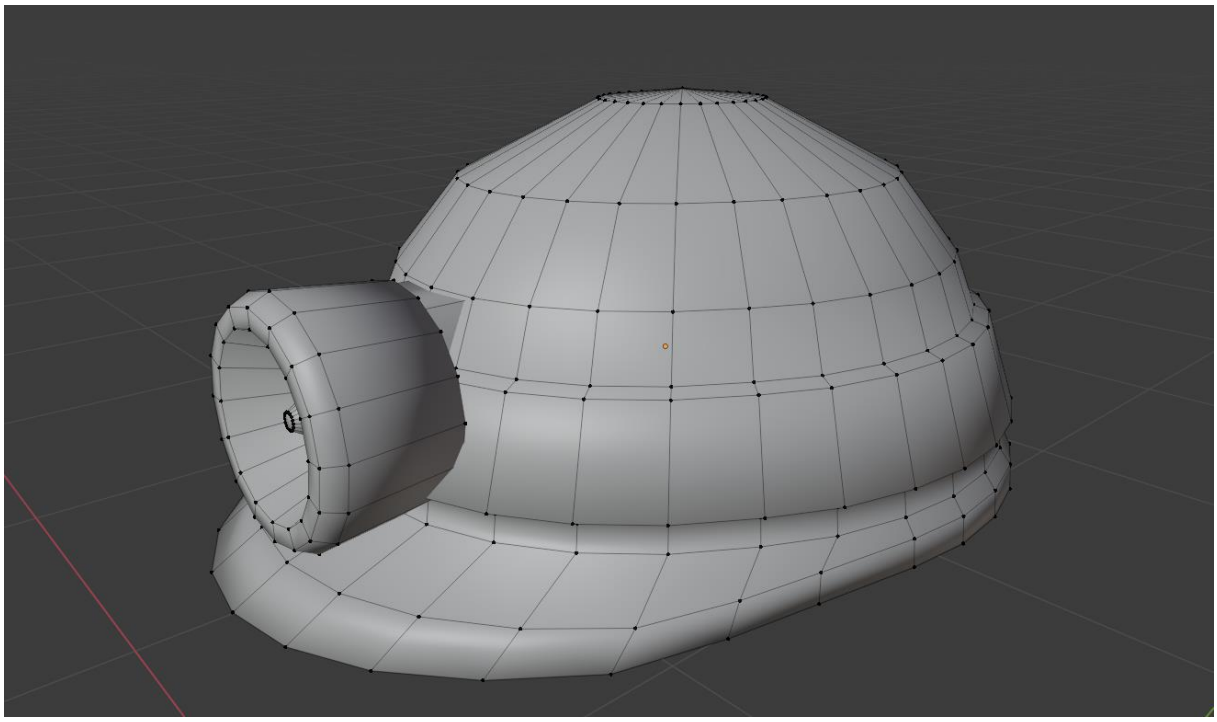
2.1 Geometria

3D-pelihahmo tarvitsee 3D-mallin. Malli on 3D-pelihahmon tärkein osa, ilman mallia ei ole 3D-pelihahmoa. Malli on hahmon muoto kolmiulotteisessa tilassa. Se on tietokoneen sisällä oleva kappale, joka edustaa todellista, tai kuvitteellista asiaa. 3D-pelihahmon malli on tietokoneen sisällä oleva patsas, joka tuodaan eloon koodin ja animaatioiden avulla.

3D-polygonimalli koostuu vertekseistä, viivoista ja polygoneista. Verteksi on piste kolmiulotteisessa tilassa. Viiva yhdistää kaksi verteksiä janalla. Polygoni on taso, joka muodostuu yhdistämällä kolme, tai useampia verteksiä viivoilla. Malli muodostuu piirtämällä kuva näiden osien yhdistelmästä. (Gee & Falco, 2010, s. 4)

Kuvassa 1 on kuva kypärän 3D-mallista. Mustat pisteet ovat verteksejä. Verteksit yhdistyvät toisiinsa viivoilla. Kun kolme tai useampia verteksejä yhdistetään viivalla, syntyy taso. Vertekseistä ja viivoista muodostunut verkko täytetään tasoilla, joista tietokone piirtää kuvan.

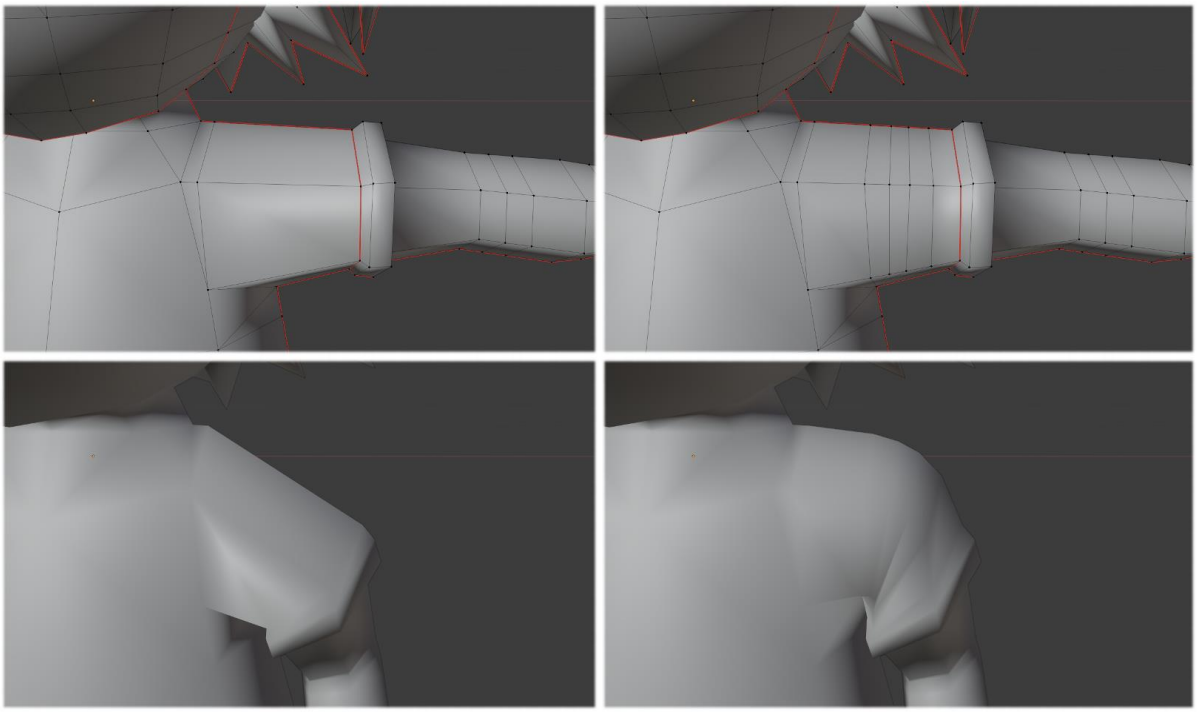
Kuva 1. Kuva jossa näkyy verteksit, viivat ja tasot, jotka muodostavat 3D-mallin



Kun tehdään 3D-mallia pelihahmolle, on otettava huomioon tiettyjä asioita. Mallin geometria on suunniteltava niin, että turhia polygoneja ei ole. Polygonien määrän vähentäminen parantaa suorituskykyä (Ward, 2011). Turhien polygonien välttäminen on hyvä käytäntö missä tahansa mallinnusprojektissa, mutta pelihahmoa tehdessä siihen on kiinnitettävä erityistä huomiota. Esimerkiksi siksi, että erilaisilla peleillä on erilaisia vaatimuksia. Puhelimella pelattavan mobiilipelin hahmot eivät tarvitse yhtä paljon polygoneja, kuin suuren budjetin konsolipelin hahmot (Maxwell, 2019).

Mallin topologian on oltava sopiva. Topologia on mallin ulkoisen pinnan muoto, joka koostuu tasoista, viivoista ja vertekseistä. 3D-pelihahmossa topologia on tärkeää esimerkiksi animaatioiden kannalta. Paikallaan oleva kappale saattaa toimia, vaikka sen topologia ei ole ideaalinen. Mutta animoitavassa hahmossa mallin vääntely ja venyttely saattaa korostaa huonoa topologiaa, mikä johtaa epäluonnollisten taipumien ja muotojen syntymiseen. Pelihahmot on mallinnettava niin, että niiden animoiminen on mahdollista. Taipuville osille on tehtävä topologia, joka antaa niiden liikkua luonnollisesti, kun niitä animoidaan. Tämä tarkoittaa, että nivelten ympärillä pitäisi olla enemmän verteksejä, jotta taipumakohdassa on enemmän muokattavaa materiaalia, ja liikkeestä tulee luonnollisemman näköinen (Maxwell, 2019). Kuvassa 2 näkyy topologian vaikutus hahmoon. Ylärivillä on kuvat mallin geometriasta. Alarivillä on kuva mallista sen jälkeen, kun olkapäätä on liikutettu mallin luurangon avulla. Vaikka paikallaan olevan mallin muoto ei muutu, kun siihen lisätään verteksejä, liikutetuissa malleissa on selvä ero. Oikeanpuoleisessa mallissa on enemmän verteksejä olkapään kohdalla, joka mahdollistaa luonnollisemman näköisen taipumisen.

Kuva 2. Kuvassa näkyy topologian vaikutus liikuteltavaan malliin



Toinen asia joka, kannattaa ottaa huomioon, kun tehdään malleja pelejä varten on mallin koko. Vääränkokoinen malli on väärässä mittakaavassa, kun sen tuo pelimoottoriin. Tämän

voi välttää tarkistamalla, että 3D-mallinnusohjelman mittakaava on asetettu mittaan, joka on yhteensopiva pelimoottorin kanssa. Yhdenmukaisuuden vuoksi on hyvä varmistaa, että samaa mittakaavaa käytetään pelin jokaiselle mallille.

Kun tehdään 3D-mallia Blenderissä, tekemisessä voidaan hyödyntää ”Modifiereita”. Modifier on automaattinen toimi, joka vaikuttaa 3D-malliin eri tavoilla (Blender Documentation Team, Introduction, n.d. -a). Esimerkiksi ”Decimate” modifier poistaa hahmosta verteksejä ja vähentää sen polygonien määrää (Blender Documentation Team, n.d. -b). ”Mirror” tai peilaus, on modifier, joka luo mallista kopion, joka on geometrian peilikuva (Blender Documentation Team, n.d. -c). Peilaus-modifieriin voi muuttaa mihin suuntaan malli peilataan, vaaka vai pystysuuntaan. Yhdellä mallilla voi olla samaan aikaan useita modifiereitä. Modifier vaikuttaa malliin pysyvästi vasta sinä vaiheessa, kun se liitetään malliin. Liittämisen jälkeen modifier poistuu mallista, ja sen tekemät muutokset tulevat osaksi mallin geometriaa. Ennen liittämistä modifierin voi poistaa käytöstä missä tahansa vaiheessa ja mallin geometria pysyy ennallaan.

Hahmo mallinnetaan neutraaliin asentoon, tämä asento on nimeltään ”Bind pose”, joskus käytetään myös nimiä ”Reference pose” tai ”Rest pose” (Gregory, 2014, s. 552). Yleensä asento on sellainen, jossa hahmo seisoo kädet ojennettuna sivuille. Tätä asentoa kutsutaan myös T-asennoksi, koska se muistuttaa T-kirjainta. T-asento antaa hyvän näkymän hahmon jokaiseen osaan. Tästä asennosta on hyvä lähteä animoiman hahmoa, ja tekemään muita jatkotoimia. 3D hahmoon kuuluu paljon muutakin kuin pelkkä malli. Siihen tulee materiaali, joka määrittää pintakuvion ja luuranko, joka antaa mallin liikkua ja mahdollistaa sen animoimisen.

2.2 Materiaalit ja kartoitukset

3D-malli itsessään on vain muoto, yksivärinen patsas. Mallin ulkonäköä voi muuttaa, asettamalla sille materiaalin. Materiaali muuttaa mallin pinnan tuntumaa ja kuviointia. Materiaalin lisäämistä malliin voisi verrata yksivärisen savipatsaan maalaamiseen ja lakkaamiseen.

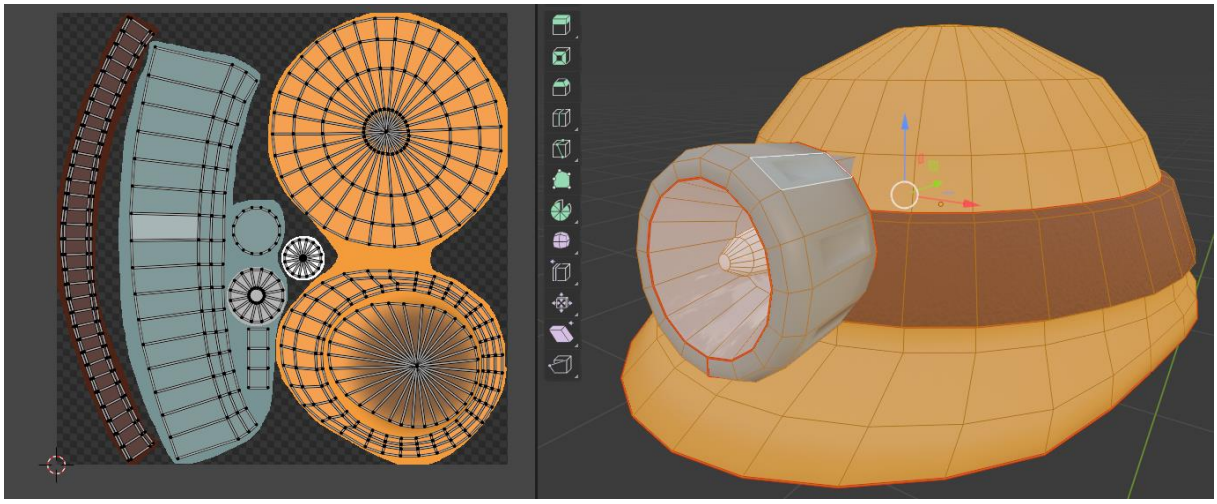
Materiaali määrittää sen, miltä mallin pinta näyttää. Se määrää mallin pinnan kuvion ja karkeuden. Materiaali päättää myös, miten valo käyttäytyy mallin pinnan kanssa. Useilla malleilla voi olla sama materiaali. Yhdelle mallille voi määrittää useita materiaaleja, mallin eri osille voi määrittää erilaiset materiaalit. (Gregory, 2014, s. 468)

Mallin pinnan kuvitusta voi säätää antamalle materiaalille tekstuurin. Tekstuuri on kuva, joka sisältää mallin pinnan kuvituksen. Mallille voi myös lisätä erilaisia ”Kartoituksia”, jotka ovat kaksiulotteisia kuvia, jotka sisältävät tietoa siitä minkälaisia ominaisuuksia mallin pinnalla pitäisi olla. Kaksiulotteisen kuvan lisääminen kolmiulotteiseen malliin vaatii UV-kartoituksen. UV-kartoituksessa mallille tehdään UV-kartta, joka sisältää koordinaatiston, joka osoittaa miten mallin osat sijoittuvat 2D-kuvaan. UV-kartoituksessa mallin vertekseille annetaan kaksiulotteiset koordinaatit. Yleensä näitä koordinaatteja edustaa kaksi arvoa, U ja V. Koordinaatisto alkaa kuvan vasemmasta alakulmasta arvolla 0, 0 ja päättyy oikeaan yläkulmaan, jonka arvo on 1, 1. Polygoni asetetaan kuvaan antamalla sen vertekseille koordinaatit. Kartoituksen jälkeen koordinaattien määräämä osa kuvasta näkyy 3D-mallin pinnassa. Mallin pinnassa olevan 2D-tekstuurin pikseleistä käytetään nimeä ”Tekseli”. Tekstuurin kuvakoolla ei ole merkitystä, koska kartoitus tehdään koordinaatistolla, joka alkaa ja päättyy aina kuvan reunoihin. Tekstuurin voi siis vaihtaa eri kokoiseen versioon ilman uuden UV-kartoituksen tekemistä. (Gregory, 2014, ss. 462-463)

Blenderissä UV-kartoituksen voi luoda tekemällä ”Unwrap” toiminto mallille. Voitaisiin sanoa, että unwrapissa mallin pinta ”kuoritaan” ja irrotetaan mallista, sitten se levitetään litteäksi kuvan päälle. Mallin pinta asetellaan niin että kuvassa olevat kuviot asettuvat haluttuun kohtaan mallissa. Tämä mallin asettelu kaksiulotteiseen kuvaan luo UV-kartan, jota käytetään tekstuurien ja muiden kartoitusten asetteluun. Kuva 3 näyttää miten mallin geometria on levitetty UV-kartaksi. Vasemmalla on UV-kartta, joka on tehty oikealla

näkyvälle kypärän mallille. Mallinnetun kypärän tasot on levitetty 2D-kuvan päälle, niin että tekstuurin kuviot asettuvat haluttuun kohtaan mallissa.

Kuva 3. Kuvassa näkyy mallin UV-kartta ja malli



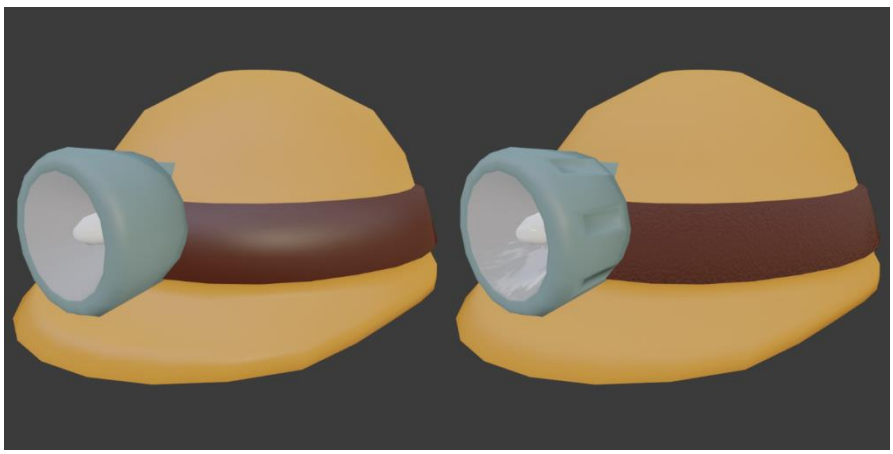
Kun mallin geometria on UV-kartoitettu, sen materiaalille voi asettaa muita kartoituksia. ”Diffuse map” tai ”Albedo map” on mallin tekstuuri, joka sisältää tiedon siitä mitä värejä mallin pinnassa on. Tekstuuri muuttaa vain mallin pinnan kuviointia, se on kuin liimaisi litteän tarran malliin. Tekstuuriin voi olla piirrettynä reikä, mutta se näkyy mallin pinnassa vain litteänä kuvana. (Gregory, 2014, s. 462)

Jos halutaan muuttaa mallin pintaa, on käytettävä muita kartoituksia. ”Normal map” on tietoa mallin pinnan muodosta, joka on tallennettu 2D-kuvaan. Kuvan jokaiselle tekselille tallennetaan normaalivektori, joka kertoo mihin suuntaan pinta tekselin kohdalla osoittaa. Normal mapin avulla materiaaliin voi piirtää syvennyksen, joka renderöidään kuin se olisi kaiverrettu malliin, vaikka mallin geometria ei sisältäisi kyseistä muotoa. Malli, johon on muotoiltu pienetkin yksityiskohdat olisi raskas pelimoottorille. Normal map tallentaa pinnan muodot kevyempään muotoon. Normal map ei kaiverra malliin reikää, se vain saa valaistuksen käyttäytymään kuin mallissa olisi reikä. Se ei siis sovellu suurten muotojen ilmaisemiseen, mutta mahdollistaa pienten yksityiskohtien lisäämisen malliin. Toinen tapa pinnan muotojen lisäämiseen on ”Bump map” joka tallentaa 2D-kuvaan tiedon siitä, miten ”korkealla” pikselin sijainti on pinnan tasosta. Korkeus tallennetaan mustavalkoiseen kuvaan, joka sisältää ainoastaan tiedon pinnan korkeudesta jokaiselle tekselille. Nykyään useimmat

pelimoottorit käyttävät Normal mappeja bump mappien sijaan. ”Specular map” tai ”Gloss map” sisältää tiedon siitä mitkä mallin osat ovat kiiltäviä ja miten paljon ne heijastavat valoa. Specular map säilyttää 2D-kuvassa tiedon siitä miten kiiltävä mallin pinta on tietyssä kohdassa. ”Environment map” on kartoitus, joka sisältää kuvan mallin halutusta ympäristöstä. Environment map on kuva, joka sisältää mallin kiiltävien osien heijastuksissa näkyvät kuviot. Samoin kuin Normal map edustaa muotoa, jota ei todellisesti ole mallin muodossa, Environment map edustaa mallin ympäristöä, joka ei todellisesti ole mallin ympäristössä. Ympäristön näyttäminen valmiista kuvasta vie vähemmän laskentatehoa, kuin heijastusten laskeminen reaaliaikaisesti mallia ympäröivistä objekteista. Toisin kuin muut kartoitukset, Environment map ei ole UV-kartoitetttu mallin pintaan. (Gregory, 2014, ss. 520-523)

Esimerkki siitä miten erilaiset kartoitukset vaikuttavat malliin näkyy kuvassa 4. Vasemmalla on malli, jossa on vain diffuse map, joka antaa mallille pintavärin. Oikealla on malli, jossa on diffuse mapin lisäksi normal map, joka tekee urat lampun runkoon, bump map, joka antaa pintakuvion hihnalle, ja gloss map, joka määrittää, että lampun sisäosa on kaikista kiiltävin osa kypärää.

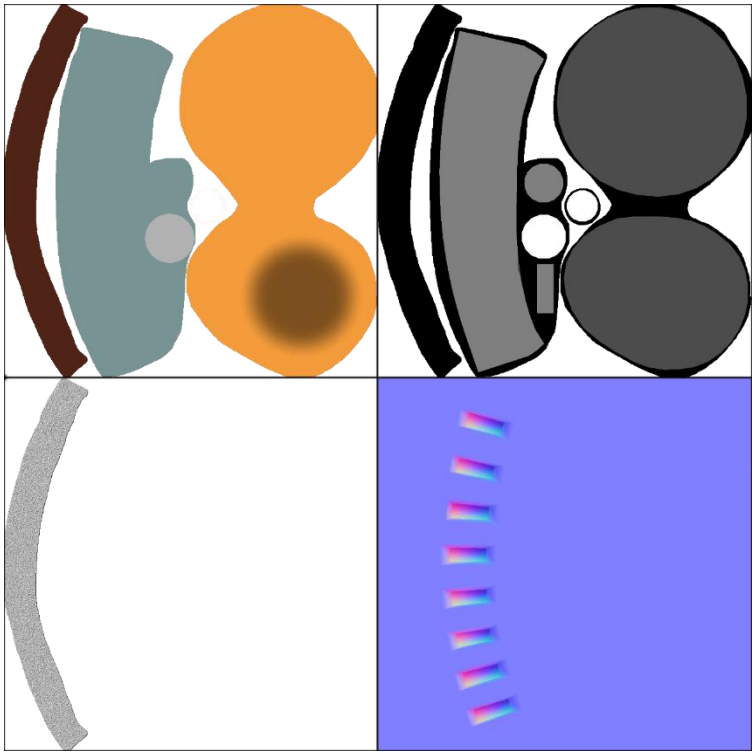
Kuva 4. Vertailu erilaisten kartoitusten vaikutuksista malliin.



Kuvaan 5 on koottu mallin kartoitukset. Vasemmassa yläkulmassa on diffuse map. Oikeassa yläkulmassa on Gloss map. Gloss map on mustavalkoinen, musta osa tarkoittaa karkeaa pintaa, joka ei heijasta valoa ja valkoinen tarkoittaa sileää pintaa, joka heijastaa paljon valoa.

Kartoitus tekee ruskeasta hihnasta karkean ja saa lampun sisäpinnan kiiltämään. Lampun heijastuksessa näkyvä kuva ei synny mallin ympäristöstä, se tulee environment mapista. Vasemmassa alakulmassa on bump map. Se tekee kypärän hihnasta kuhmuraisen. Oikeassa alakulmassa on normal map, joka tekee urat lampun runkoon. Kuten aiemmissa kuvissa näkyy, lampun geometriassa ei ole uria. Urat lasketaan normal mapista ja piirretään mallin pintaan. Yhdistelemällä erilaisia kartoituksia saadaan malli, jossa on useita eri pintakuvioita. Malliin voisi myös luoda erilaisia pintakuvioita antamalla mallin eri osille omat materiaalit, joissa on eri pintakuviot. On kuitenkin suositeltavaa käyttää mahdollisimman vähän eri materiaaleja, koska useamman materiaalin renderöiminen on raskaampaa tietokoneelle.

Kuva 5. Mallin kartoitukset.



2.3 Rigging ja skinning

Kun pelihahmolla on malli ja materiaali, se on visuaalisesti valmis. On kuitenkin tehtävä vielä töitä, jotta mallista saa valmiin pelihahmon. Tietokone ei tiedä mitkä mallin osat liikkuvat, tai miten niiden kuuluu liikkua. Tätä varten mallille on tehtävä armature, joka on mallin luuranko. Luuranko koostuu luista, jotka on liitetty toisiinsa. Tämä luiden sarja määrittää missä mallin nivelet ovat. Luurangon jokaiselle luulle määritetään painoarvo, joka päättää

miten paljon se vaikuttaa mallin eri osiin. Prosessi, jossa luiden painoarvot määritellään, on nimeltään ”Skinning”. Koko prosessista, jossa mallille määritellään luuranko ja luiden painoarvot, käytetään nimeä ”Rigging” (Amin, 2014).

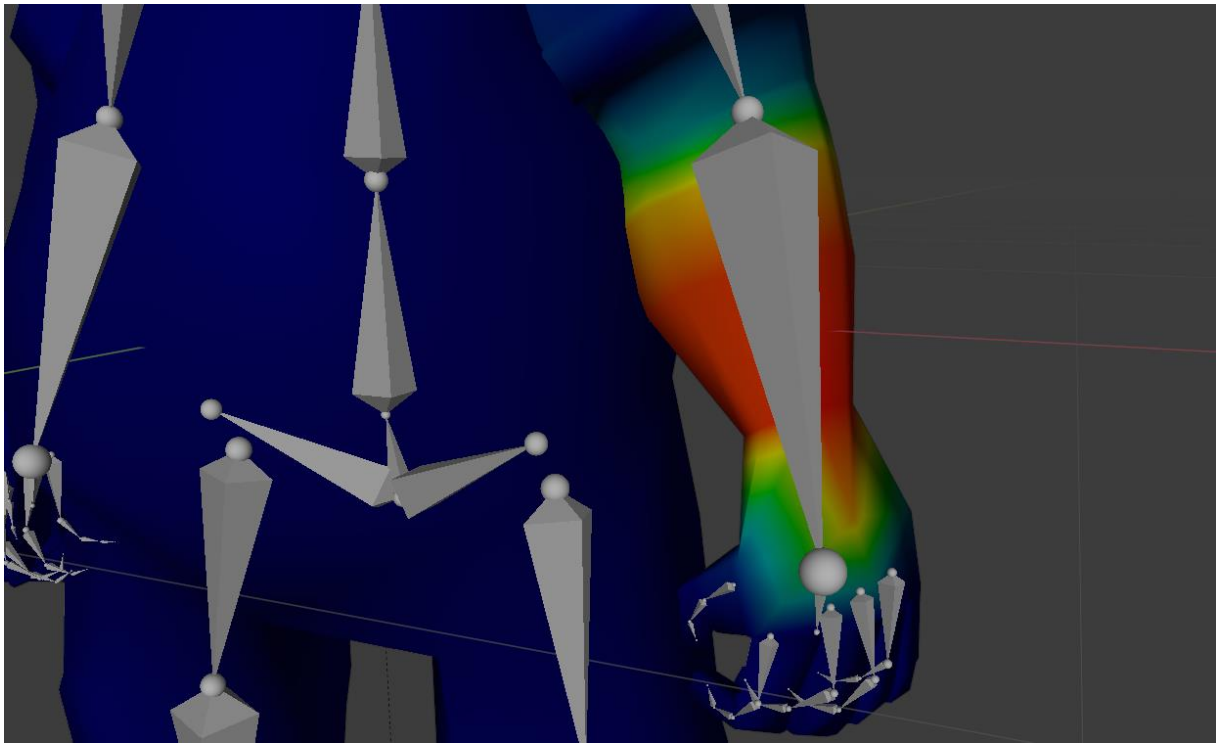
3D-pelihahmoja voi animoida usealla eri tavalla. 3D-grafiikan alkuaikoina suosittu menetelmä oli ”Rigid hierarchical animation”. Tässä menetelmässä hahmo koostuu erillisistä kiinteistä osista, jotka ovat kiinnitetty toisiinsa nivelten kohdalla. Osat pysyvät aina saman muotoisina ja kääntyvät ainoastaan nivelten kohdilla. Tämä tapa sopii hahmoille, jotka koostuvat kovista osista, kuten vaikka koneille, mutta se saattaa näyttää kömpelöltä, kun sitä käytetään ”pehmeiden” hahmojen animoimiseen. Toinen tapa on ”Per-Vertex animation” jossa mallin verteksejä voi liikutellaan yksi kerrallaan. Verteksien siirtäminen yksi kerrallaan mahdollistaa hyvin tarkan liikkeen, mutta se ei ole tehokas tapa tehdä animaatiota. Tästä syystä se ei sovi hahmojen liikkeiden animoimiseen. Per-Vertex animaatiota käytetään kasvojen ilmeiden animoinnissa, jossa tarkat liikkeet ovat tärkeitä. Kolmas tapa on ”Skinned animation” tai luurankoanimaatio. Luurankoanimaatiossa 3D-mallille tehdään luuranko, joka kiinnitetään malliin. Luurankoon kiinnitetty 3D-malli seuraa luurangon liikkeitä, kuten ihmisen nahka seuraa omaa luurankoaan. Malli venyy ja muuttaa muotoaan luurangon liikkuessa. Luurankoanimaatio on nykyään yleisimmin käytössä oleva 3D-animaation tyyli peleissä ja elokuvissa. (Gregory, 2014, ss. 544-547)

Riggausta varten tarvitaan luuranko. Luuranko kostuu ”luista” jotka ovat yhdistetty toisiinsa, luut yhdistetään tiettyyn hierarkiaan (Gregory, 2014, ss. 548-549). Esimerkiksi, ihmishahmon luurangossa selkärankaan on yhdistetty olkapää ja olkapäähän on yhdistetty käsi. Tässä esimerkissä selkäranka on hierarkian ensimmäinen luu, eli hierarkian ”Root”. Kun selkärankaa liikutetaan, olkapää seuraa sen liikettä, ja käsi taas seuraa olkapään liikettä. Pelimoottorille tärkeintä on luiden välissä olevat nivelet, pelimoottori ei kiinnitä huomiota luihin, pelkästään niiden niveliin (Gregory, 2014, s. 549). Yhdistelemällä luita toisiinsa, voidaan luoda minkä tahansa muotoinen luuranko. On myös mahdollista tuoda ohjelmaan valmis luuranko, johon malli kiinnitetään. Joissain ohjelmissa on myös omasta takaa kirjasto, joka sisältää valmiita luurankoja kaikista yleisimmille hahmoille, kuten vaikka ihmisille.

Luuranko yhdistetään malliin. Jotta luurangon liikuttelulla on vaikutus malliin, luurangolle on tehtävä selväksi, mitkä sen luista hallitsevat mitäkin hahmon osia. Luurangon luille annetaan jokin osa mallista hallittavaksi ja painoarvo, joka kertoo miten suuri vaikutus luulla, on hallittavaan kohtaan. Prosessi, jossa painoarvot määritellään, on nimeltään ”Skinning” koska 3D-malli määritellään luurangon ”nahkaksi” (Gregory, 2014, s. 570).

Kuvassa 6 näkyy luun vaikutus hahmoon. Malliin on värjätty ranneluun vaikutus hahmon malliin. Punainen väri tarkoittaa, että painoarvo on maksimissaan ja luu ohjaa täysin tätä osaa. Sininen tarkoittaa, että luu ei vaikuta osaan. Sävyt, jotka ovat näiden värien välissä tarkoittavat, että luun vaikutus osaan on täyden ja mitättömän vaikutuksen välissä. Kuvassa näkyy, että luun vaikutus on maksimissaan luun keskellä ja se hiipuu, kun siirrytään lähemmäs luun päätä, kunnes vaikutus on täysin mitätön ja seuraava luu ottaa hallinnan. Esimerkkikuva, jossa näkyy punainen ja sininen väri on otettu Blender-ohjelmasta. Muissa mallinnusohjelmissa värit voivat olla erilaisia, esimerkiksi 3D-mallinnusohjelma Maya käyttää mustaa, valkoista ja harmaan sävyjä painoarvojen ilmaisuun.

Kuva 6. Kuvankaappaus weight paint näkymästä.



Luiden painoarvot voi määrittää muutamalla eri tavalla. Arvot voi määrittää tarkasti käsin. Mallin jokaiselle luulle voi antaa tarkan painoarvon, yksi verteksi kerrallaan. Tämä voi käydä työlääksi hyvin nopeasti. Toinen vaihtoehto, olisi antaa painoarvot automaattisesti. Joissakin ohjelmissa on ominaisuus, jossa luuranko arvaa parhaat mahdolliset painoarvot, kun se liitetään malliin. Tämä on tietysti helppoa, mutta automattinen painoarvojen jako ei välttämättä aina tuota sopivia tuloksia. Painoarvot voi myös ”Maalata” mallin pintaan, tätä kutsutaan nimellä ”Weight painting”. Weight paintingissa valitaan luu, jolle asetetaan painoarvot ja sitten kerrotaan painoarvo, joka määrittää miten paljon luu vaikuttaa maalattaviin vertekseihin. Painoarvo ”Maalataan” valittuihin vertekseihin, siirtämällä hiiren osoitin niiden päälle 3D-ohjelman näkymässä. Kun hiirestä klikataan, valittu painoarvo asetetaan osoittimen alla oleviin vertekseihin.

3 Hahmo pelimoottorissa

3D-hahmosta tulee pelihahmo vasta sitten kun se on pelissä. Pelin tekemiseen käytetään Pelimoottoria. Pelimoottori sisältää työkalut, jotka mahdollistavat erilaisten pelien rakentamisen. Unity on pelimoottori, joka sisältää työkalut pelin rakentamiseen, ja mahdollistaa sen julkaisun useilla eri alustoilla. Unityn käyttö on ilmaista tietyin rajoituksin, jopa kaupallisissa tarkoituksissa. Tässä opinnäytetyössä käytetään Unity-moottoria ja tarkastellaan pelihahmon luomista Unitystä löytyvien työkalujen avulla.

Tämä osio aloitetaan esittelemällä, miten pelin sisäiset objektit toimivat Unityssä. Tarkastellaan minkälaisia ominaisuuksia, yksittäisillä objekteilla on, ja miten niitä voi liittää toisiinsa pelihahmon luomista varten. Sitten esitellään animaatiot. Mistä animaatio koostuu ja miten päätetään milloin hahmo toistaa animaatioita. Lopuksi katsotaan Unityn tarjoamia työkaluja proseduaalisten animaatioiden tekemiseen, niitä hetkiä varten, jossa hahmon on tehtävä jotain, jolle ei ole valmista animaatioita.

3.1 Peliobjektit

Peliobjekti on pelin sisällä oleva objekti. Unity-moottorilla tehdyssä pelissä, Jokainen asia on peliobjekti. Pelaajahahmo on peliobjekti, valot ja ruohikko ovat peliobjekteja. Kaikki asiat,

jotka näkyvät pelissä ovat peliobjekteja. Vaikka kaikki pelin sisäiset asiat ovat peliobjekteja, niiden toiminnallisuudet voivat erota suuresti toisistaan. (Unity technologies, 2022 -f)

Peliobjekti saa toiminnallisuuden sen komponenteista ja lapsiobjekteista. Peliobjektiin voidaan lisätä toiminnallisuutta lisäämällä sille komponentteja (Unity technologies, 2022 -h). Tyhjä peliobjekti sisältää yhden komponentin, peliobjektin sijainnin. Sijainti-komponentti sisältää peliobjektin sijainnin, koon, ja kierron. Sijainti-komponentti on jokaisessa peliobjektissa, sitä ei voi poistaa (Unity technologies, 2022 -f).

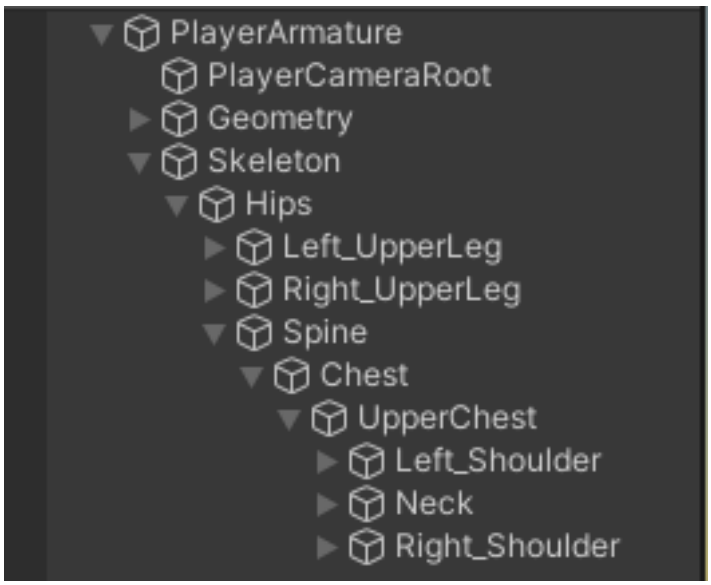
Sijainti-komponentti määrää objektin koon ja sijainnin, muut komponentit määräävät peliobjektille muita ominaisuuksia. Esimerkiksi ”Mesh renderer” komponentti piirtää peliobjektin sijaintiin 3D-mallin. ”Rigidbody” komponentti antaa objektin pudota fysiikkasimulaation avulla. ”Script” komponentti mahdollistaa peliobjektin komponenttien ohjaamisen koodin avulla. Komponentteja on lukuisia ja niitä kaikkia ei voida käsitellä erikseen. Tärkein asia tämän työn kannalta, on tietää, että peliobjekti, jossa ei ole komponentteja, on vain tyhjä piste pelimaailmassa.

Peliobjektille voi asettaa lapsiobjekteja eli siis ”Child” objekteja. Peliobjekti, jolle on asetettu child-objekti, on parent-objekti. Child-objekti seuraa parent-objektin sijaintiin tehtyjä muutoksia. Child-objektille voi asettaa oman child-objektin. Objekteja voi asettaa ketjuun niin monta ”sukupolvea” kuin haluaa, ketjun vanhin parent-objekti on nimeltään root-objekti. Child-objektit toimivat aivan kuten mikä tahansa muukin peliobjekti, niille voi lisätä komponentteja, ja niiden sijaintia voi muokata. (Unity technologies, 2022 -n)

Kuva 7 näyttää parent-objektin, jolle on asetettu monta child-objektia. Pelissä olevat objektit näkyvät listassa, joka löytyy Unity-editorista. Listassa näkyy sisennettynä root-objektin child-objektit. ”PlayerArmature” on ketjun ensimmäinen parent-objekti, se on siis ketjun root-objekti. Kaikki ketjun sisällä olevat objektit seuraavat ”PlayerArmature” objektin sijaintia. Osalla root-objektin child-objekteista on omat child objektinsa. Objektin nimen vieressä oleva kolmio ilmaisee, että objektilla on child-objekteja. Child-objektit saa piilotettua painamalla hiirellä kolmiosta. Root-objektin child-objektien piilottaminen piilottaa samalla

kaikkien sukupolvien child-objektit. Child-objektien ketjulla ei ole pituusrajaa, se voi venyä hyvinkin pitkäksi. Ketjun piilottaminen selventää näkymää ja helpottaa työskentelyä.

Kuva 7. Kuvankaappaus Unity-editorista, jossa näkyy peliobjektien hierarkia.



Kun pelihahmo luodaan peliin, se todennäköisesti koostuu root-objektista, ja useista child-objekteista, joihin kaikkiin on lisätty komponentteja. 3D-pelihahmo tarvitsee ”mesh renderer” komponentin 3D-mallin piirtämistä varten. Se tarvitsee ”Script” komponentin hahmon koodia varten. Jos malli halutaan animoida, tarvitaan animator-komponentti. Animator liikuttaa hahmon luurankoa, joka on todennäköisesti hahmon mallin child-objekti. Luurangon jokainen luu on erillinen, luurangon child-objekti. Pelihahmo koostuu siis monista eri komponenteista, ja objekteista. Kuva 7 näyttää Unityn valmiin hahmopohjan rakennetta. PlayerArmature-objekti sisältää script komponentin, joka säilyttää hahmon toimintaan liittyvän koodin. Geometry-objekti sisältää mesh renderer komponentin, joka piirtää hahmon 3D-mallin. Skeleton-objekti on hahmon luuranko, jonka child objekteista löytyy hahmon luut.

Pelihahmo voi esiintyä useaan kertaan pelissä. Jos se on pelaajahahmo, se todennäköisesti esiintyy hyvin usein. Jos pelihahmo on eläin, joka esiintyy laumoissa, sama hahmo voi esiintyä useita kertoja samassa kohtauksessa. Pelihahmon voi monistaa kopioimalla siitä useita kappaleita haluttuun paikkaan. On kuitenkin olemassa parempi tapa. Hahmon monistaminen sujuu paremmin käyttämällä prefab-objektia.

Prefab on kopio peliobjektista, joka on otettu talteen myöhempää käyttöä varten. Mistä tahansa peliobjektista voi tehdä prefab-objektin. Prefab-objekti säilyttää peliobjektin, sekä sen kaikki komponentit ja child-objektit, pakkauksessa, joka voidaan myöhemmin sijoittaa peliin. Prefabiin tehdyt muutokset vaikuttavat sen kaikkiin esiintymiin. (Unity technologies, 2022 -j)

Esimerkiksi, jos pelissä on lauma lampaita. Lauman saa luotua tekemällä yhden lampaan, joka on kopioitu useaan kertaan niitylle. Jos myöhemmin päätetään, että lampaiden villan on oltava vaaleanpunaista, on väri vaihdettava erikseen jokaiselle kopiolle. Jos lauma on tehty tallentamalla yksi lammas prefab-objektiksi, joka on sitten lisätty useaan kertaan niitylle, riittää kun alkuperäinen lammas muutetaan, ja kaikki niityn lampaat muuttuvat prefabin mukaiseksi. Pelihahmo pystytään ottamaan talteen prefab-objektiin, josta se sitten kutsutaan tarpeen tullen. Mitä jos halutaan että osalla lampaista on valkoinen villa ja osalla vaaleanpunainen? Tallennetaanko kaksi prefab-objektia, joilla on eriväriset villat? Ei tarvitse, tähänkin on parempi ratkaisu, voidaan käyttää Scriptable object -luokkaa.

Scriptable object on säilytystapa, jolla voidaan tallentaa paljon dataa kevyeen muotoon (Unity technologies, 2022 -m). Lampaita varten voidaan luoda lammasluokka, jolle annetaan parametreiksi lampaan prefab ja lampaan väri. Lammas-luokasta voidaan luoda kaksi versiota, valkoinen, jossa ”väri”-parametrin arvo on ”valkoinen” ja vaaleanpunainen, jossa se on ”vaaleanpunainen”. Lammasluokalle voi asettaa muitakin parametrejä, kuten vaikka lampaan kävelynopeus, lempiruoka ja villan määrä. Kun uusi lammas luodaan, se käyttää aina samaa prefab-objektia. Tarvitaan koodi, joka lisää lampaan peliin. Ensin koodi lisää lampaan prefab-objektin peliin, sitten se lukee lammasluokasta lampaan värin ja muut ominaisuudet, ja määrittää ne sopiviksi. Näin samasta prefabista saa erilaisia lampaita.

3.2 Third Person Character Controller

Pelihahmo vaatii toimiakseen yhdistelmän erilaisia objekteja ja komponentteja, sekä myös koodia, joka ohjaa hahmon toimintaa. Pelihahmon visuaalisen osan lisäksi on paljon tehtävää ”Pellin alla”. Onneksi tätä työtä voi helpottaa, käyttämällä valmista pohjaa.

Unity technologies on luonut ja julkaissut ilmaisen ”Third Person Character Controller” paketin, joka sisältää hahmopohjan, jonka päälle voi rakentaa oman pelihahmon. Hahmopohja sisältää objektien hierarkian, johon kuuluu hahmon ”Ohjain” jossa on koodi hahmon ohjaamista ja kameran liikuttamista varten, sekä esimerkkipelihahmon, jolla on 3D-malli ja luuranko, joille on tehty valmiit animaatiot ja riggaus. Hahmopohja on suunniteltu helposti muokattavaksi, joten sen ominaisuuksia ja osia voi poistaa tai lisätä helposti. Hahmon mukana tulevaa esimerkkikoodia voi muokata, koodin kautta hahmon nopeutta, hyppykorkeutta ja muita ominaisuuksia voi muuttaa helposti. Hahmon animaattorille voi lisätä uusia animaatioita. Koko hahmon malli voidaan korvata toisella ihmismallisella hahmolla, ja hahmon animaatiot toimivat edelleen samalla tavalla ilman ylimääräistä työtä.

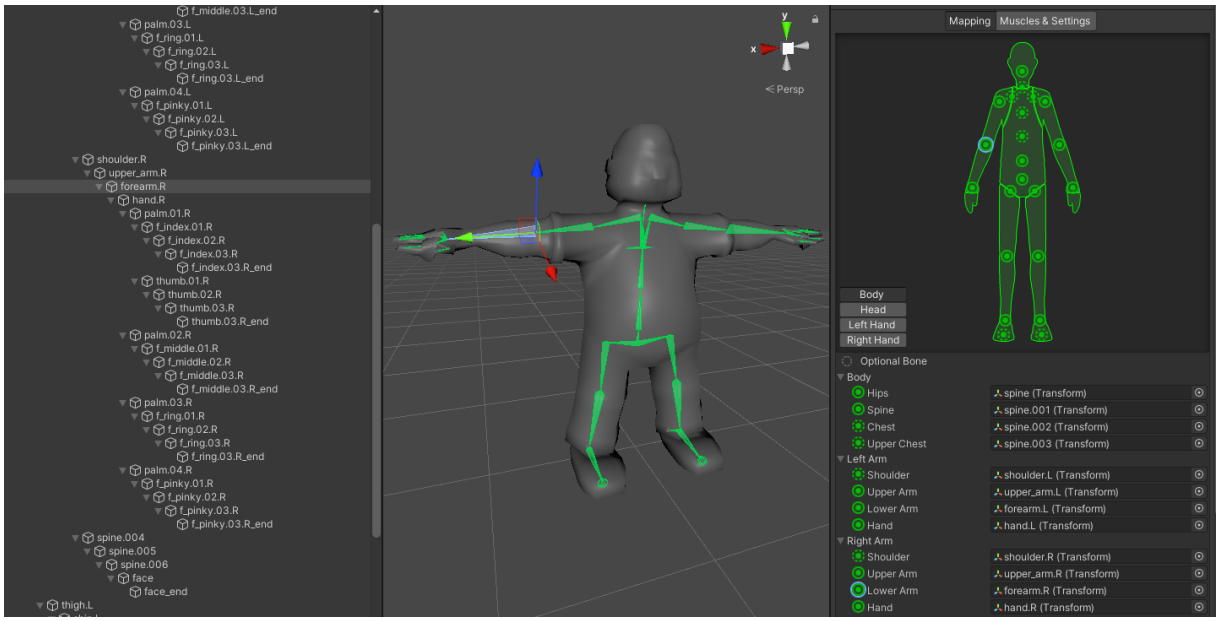
Hahmopohja käyttää humanoid avataria, joka mahdollistaa animaatioiden vaihtamisen toisiin ihmismallisiin animaatioihin. Unityssä on kaksi eri avataria. ”Humanoid” ihmismäisille malleille, eli siis malleille, joilla on kaksi kättä ja jalkaa, ja ”Generic” joka on kaikille muille malleille. Humanoid avataria varten mallin luurangosta valitaan, mitkä luut edustavat tiettyjä ruumiinosia. Kun avatar on säädetty hahmolle, Unity tunnistaa hahmon ruumiinosat, riippumatta siitä, miten ne on nimetty hahmon luurangossa. Humanoid avataria käyttävät hahmot pystyvät käyttämään ihmisen muotoisille malleille luotuja animaatioita, vaikka mallien luurangot eivät täsmäisi toisiaan. (Unity technologies, 2022 -g)

Jos halutaan kopioida animaatio yhdestä mallista toiseen, se ei onnistu ilman avataria, elleivät mallien luurangot ole täysin identtisiä. Animaatiota ei voida toistaa, koska animaation tarvitsema luu ei välttämättä löydy noudattamalla toisen mallin luurangon rakennetta. Humanoid avatarissa riittää, että tietää mistä ruumiinosa löytyy, ei tarvitse tietää, mikä luu ohjaa sitä. (Unity technologies, 2022 -k)

Kuva 8 näyttää avatar työkalun. Vasemmalla on lista mallin luista. Listassa näkyy, että mallin luu, ”Forearm_R” on valittuna. Keskellä on kuva mallista ja luurangosta. Forearm_R on valittuna. Oikealla on Lista humanoid avatarin käyttämistä luista. Mallin luu, ”Forearm_R” on merkitty avatarin ”Lower arm” luuksi. Kun mallille annetaan humanoid avatarille tehty animaatio, se ei liikuta mallin ”Forearm_R” luuta, vaan avatarin ”Lower arm” osaa. Ihmismäiset mallit pystyvät jakamaan animaatioita humanoid avatarin avulla. Vaikka mallien

luurangot olisivat eri muotoisia, humanoid avatarissa ruumiinosat ovat aina kiinni toisissaan samalla tavalla. Kun humanoid avatar on viritetty hahmolle, tiedetään aina missä hahmon ruumiinosat ovat.

Kuva 8. Humanoid rigin asettelu.



Humanoid avatarin käyttö tarkoittaa sitä, että hahmopohjan esimerkkimalli voidaan helposti vaihtaa mihin tahansa toiseen ihmispohjaiseen malliin. Jos näin tehdään, ei ole tarvetta muokata hahmon animation controlleria. Vaikka projekti ei käyttäisikään valmista hahmopohjaa, humanoid avatar on tärkeä työkalu. Sen avulla voidaan käyttää uudelleen kaikki samaa avataria käyttävät animaatiot.

3.3 Animator ja blend tree

Animator komponentti on elintärkeä uskottavan pelihahmon tekemiseen. Animator komponentti, yhdessä animation controllerin kanssa mahdollistaa pelihahmon animaatioiden toistamisen sopivalle hetkellä.

Unity tallentaa pelihahmon animaatiot ”Animation clip” muotoon. Animation clip on leike, joka sisältää jonkin hahmon liikkeistä. Animation clipit voi tuoda ulkoisesta ohjelmasta, tai ne voidaan tehdä Unityn omilla animaatiotyökaluilla. Animation clip sisältää tietyn määrän

ruutuja, joiden aikana liike tapahtuu. Tässä asiayhteydessä ruutu tarkoittaa ennalta sovittua aikamäärää, joka kertoo, miten pitkään animaatio pysyy yhdessä asennossa. Jos animaatio toistuu nopeudella 1 fps, eli 1 ruutu sekunnissa, yksi ruutu kestää sekunnin. Jos nopeus on 30 fps, yksi ruutu kestää sekunnin kolmanneskymmenesosan. Toisin sanoen asento vaihtuu 30 kertaa yhden sekunnin aikana. Keyframe on ruutu, johon on tallennettu liikkeelle tärkeä asento. Keyframet ovat tyypillisesti animaattorin käsin tekemiä. Ruudut jotka, täyttävät tilan kahden keyframen välillä ovat Inbetween ruutuja. 3D-animaatiossa tietokone laskee inbetween ruutujen asennot. Animation clip koostuu keyframeista jotka sisältävät liikkeen tärkeimmät asennot ja inbetween ruuduista, jotka täydentävät väliin jäävät asennot.

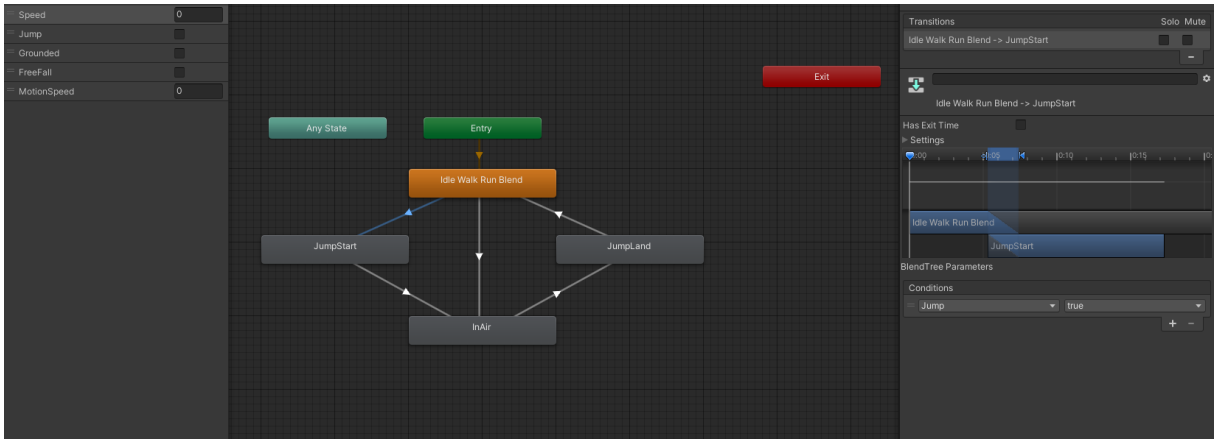
Animator komponentti tarvitsee Animation controllerin toimiakseen. Animator on peliobjektiin lisättävä komponentti, joka mahdollistaa animaatioiden lisäämisen peliobjektiin, se linkitetään peliobjektin Animation controlleriin (Unity technologies, Animator, 2022 -c). Animation controller sisältää pelihahmon animaatiot, ja listan ehdoista, jotka määräävät milloin animaatiot toistetaan (Unity technologies, 2022 -d). Animation clipit21 asetetaan animation controlleriin ja ne kytketään toisiinsa erilaisilla siirtymillä.

Siirtymiin asetetaan ehtoja, joita voi ohjata pelihahmon koodista. Ehtoja on useita eri tyyppejä. Bool on tosi tai epätosi arvo, siirtymä voidaan asettaa toteutumaan silloin kun arvo on tosi. Trigger on kytkin, kytkin antaa yhden signaalin silloin kun kytkin laukaistaan. Siirtymän voi asettaa toteutumaan, kun kytkimeltä saadaan signaali. Integer ehto on kokonaisluku. Integer ehto toteutuu, kun luku täsmää ehtoon määrättyä lukua. yhdessä siirtymässä voi olla useita ehtoja. Jos siirtymän on määrätty useita ehtoja, jokaisen ehdon on täytyttävä, jotta siirtymä toteutuu. (Unity technologies, 2022 -a)

Kuva 9 näyttää animation controllerin. Vasemmalla on lista ehdoista, jotka määräävät animaatioiden siirtymiä. Keskellä olevassa ikkunassa näkyy animaatioleikkeet ja siirtymät, jotka yhdistävät niitä. Oikealla on näkyvissä valitun siirtymän asetukset. Kuvassa näkyy, että siirtymä, joka vie hyppyanimaatioon on valittuna. Siirtymän asetuksissa näkyy, että tämä siirtymä tapahtuu silloin kun ”Jump” ehto on tosi. Oikealla oleva ikkuna, jossa on kaksi sinistä palkkia päällekkäin, on siirtymän aikajana. Siniset palkit esittävät animaatioleikkeitä, joiden välillä siirtymä tapahtuu. Kohta, jossa näkyy sininen yliviivaus tarkoittaa, että nämä leikkeet

sekoitetaan yhteen. Animaatiot sekoitetaan ottamalla asento ensimmäisestä animaatiosta ja asento toisesta animaatiosta. Asentojen välipiste lasketaan, ja hahmo asetetaan laskettuun asentoon. Siirtymän alkaessa asento ottaa enemmän vaikutusta ensimmäisestä asennosta. Toisen asennon vaikutusta lisätään vähitellen, kunnes hahmo on siirtynyt kokonaan toiseen asentoon ja siirtymä on valmis. Sekoittaminen tekee siirtymästä luonnollisemman. Ilman sekoitusta, hahmo hyppäisi välittömästi asennosta toiseen.

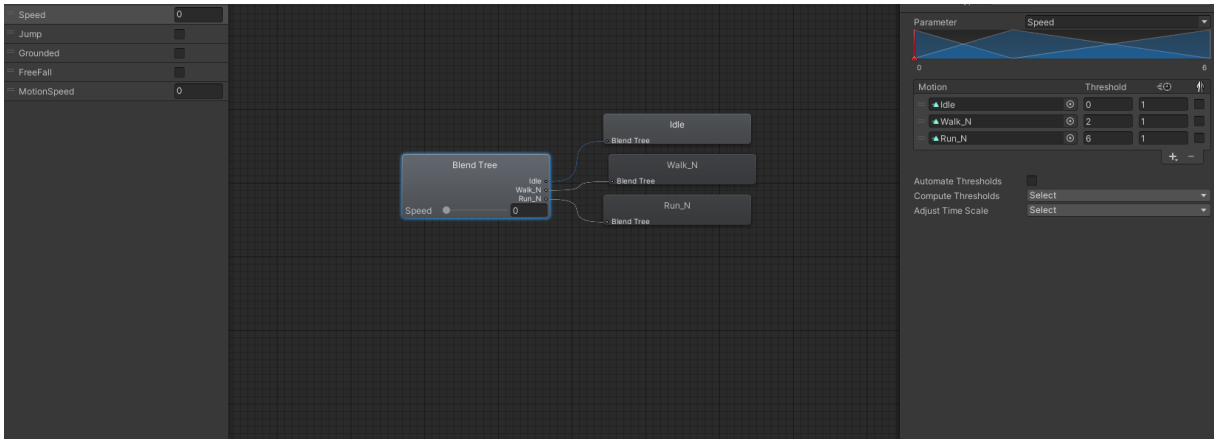
Kuva 9. Kuvankaappaus animation controllerin näkymästä Unity editorissa.



Blend tree sisältää useita animaatioita joita ”blendataan” tai sekoitetaan toisiinsa (Unity technologies, 2022 -e). Animaatioita sekoitetaan samalla tavalla kuin siirtymissä, mutta blend tree ei käytä animation controllerin siirtymiä animaatioiden välillä siirtymiseen. Kuvassa 10 on hahmon blend tree, jota käytetään kun hahmo on maan pinnalla. Vasemmalla on lista ehdoista, jotka määräävät animaatioiden siirtymiä. Blend tree ei käytä omaa listaa. Tämä on sama lista, joka on käytössä hahmon animation controllerissa. Keskellä olevassa ikkunassa näkyy blend tree, johon on kytketty kolme animaatiota. ”Idle” jossa hahmo seisoo paikallaan. ”Walk_N” jossa hahmo kävelee hitaasti eteenpäin. Ja ”Run_N” jossa hahmo juoksee eteenpäin. Blend tree siirtyy näiden animaatioiden välillä käyttäen ”Speed” arvoa, joka näkyy keskellä olevan ikkunan blend tree laatikossa. Tämä blend tree on yksiulotteinen, eli se käyttää vain yhtä arvoa. Oikealla näkyy, miten animaatiot on aseteltu blend treehen. Animaatiot toistetaan kun ”Speed” eli hahmon nopeus saavuttaa tietyn arvon. Kun nopeus on 0, toistetaan ”Idle” animaatio, jossa hahmo seisoo paikoillaan. Kun nopeus on 2, toisteaan hahmon kävelyanimaatio. Kun nopeus on 6, hahmon juoksuanimaatio toistetaan.

Jos nopeus on arvojen 2 ja 6 välillä, toistetaan animaatio, joka on sekoitus kävelyanimaatiosta ja juoksuanimaatiosta. Jos nopeus on 4, sekoitus ottaa enemmän vaikutusta kävelyanimaatiosta. Jos nopeus on 5, juoksuanimaation vaikutus sekoitukseen on suurempi. Blend tree sekoittaa animaatioita sitä mukaan, kun hahmon nopeus muuttuu. Tämä muuttaa hahmon animaatioita pelaajan komentojen mukaan ja saa hahmon tuntumaan uskottavammalta.

Kuva 10. Yksiuloitteinen blend tree ja sen sisältämät kolme animation clippiä



Näitä työkaluja voi hyödyntää, jos tehdään esimerkiksi hahmon hyppyanimaatio. Jos halutaan että pelihahmo toistaa hyppyanimaation hypätessään, hyppyanimaatio lisätään animation controlleriin ja hyppyanimaatioon johtavaan siirtymään laitetaan ehto, jota voi ohjata koodista. Hahmon koodiin laitetaan rivi, joka määrää Animation controllerin täyttämään hyppyehdon, silloin kun hahmo hyppää. Kuvan 9 animation controllerissa ehto on bool arvo nimeltään ”Jump”. Ehdon avulla hahmo toistaisi hyppyanimaation hypätessään, mutta toiminnallisuutta voisi vielä parantaa. Jos tarkoituksena on tehdä uskottava pelihahmo, hyppyanimaation on toteuduttava tavalla, joka vastaa pelaajan odotuksia.

Hyppyyn valmistaudutaan ponnistamalla. Sitten hypätään ja noustaan ilmaan. Ilmassa aletaan jossain vaiheessa laskeutumaan alaspäin. Lopuksi laskeudutaan maahan, isku maahan aiheuttaa vastareaktion hyppääjään. Jos hyppyanimaatio on asetettu toimimaan aiemmin mainitulla tavalla, hyppyanimaation toisto aloitetaan aina kun hahmo hyppää. Tämä tapa toistaa vain yhden animaation. Yhteen animaation voi sisällyttää kaikki ylempänä kuvatut vaiheet, mutta yksi animaatio, joka sisältää kaikki vaiheet, ei toimi joka tilanteessa.

Jos putoaminen alkaa tietyssä vaiheessa hyppyanimaatiota, se näyttää sopivalta vain silloin, jos jokainen hyppy on aina yhtä korkea. Jos myöhemmin päätetään, että pelihahmon on hypättävä korkeammalle, animaatio on ajoitettava uudelleen. Vaihtoehtona on asettaa putoamisanimaatiolle ehto. Hahmon nopeus pystysuunnassa voidaan laittaa numeeriseksi ehdoksi Animation controlleriin, ja putoamisanimaation ehdoksi voidaan asettaa, että se toistuu vain silloin kun putoamisehto ilmaisee, että hahmo putoaa. Toinen vaihtoehto on hyödyntää Animation controllerin blend tree ominaisuutta.

Hyppyanimaation blend treessä voisi olla kaksi animaatiota, ylöspäin menevä hyppy ja alaspäin menevä putoaminen. Blend tree voidaan asettaa sekoittamaan animaatiot yhteen hahmon pystysuuntaisen nopeuden perusteella. Jos hyppyanimaatio toteutettaisiin näin, niin hypätessä Animation controller siirtyy hyppyanimaation blend treehen. Blend tree saa arvoksi hahmon pystysuuntaisen nopeuden. Kun hahmo liikkuu ylöspäin, toistetaan hyppyanimaatiota. Kun hahmo lähestyy hypyn lakipistettä, nopeus yläsuuntaan hidastuu ja blend tree sekoittaa vähitellen putoamisanimaatiota hyppyanimaatioon. Kun hahmo putoaa ja nopeus yläsuuntaan on negatiivinen blend tree siirtyy kokonaan putoamisanimaatioon.

3.4 Animation rigging

Jotta pelihahmo pystyy olemaan uskottava osa pelimaailmaa, sen on pystyttävä olemaan uskottavassa kanssakäymisessä pelimaailman kanssa. Tämä tarkoittaa, että hahmon animaatioiden on käyttäydyttävä odotetusti joka tilanteessa. Ei kuitenkaan ole mahdollista tehdä hahmolle animaatioita jokaiseen mahdolliseen tilanteeseen. Tähän ongelmaan auttaa animation rigging.

Animation rigging mahdollistaa proseduaalisten animaatioiden lisäämisen pelihahmoon (Unity technologies, 2022 -b). Proseduaaliset animaatiot ovat animaatioita, joita ei ole tehty valmiiksi etukäteen. Tämä tarkoittaa siis hahmon liikettä, joka ei löydy animation controllerista. Esimerkiksi, jos halutaan että pelihahmo katsoo tärkeitä kohteita kääntämällä päänsä niitä kohti, ei ole järkevää tehdä erillistä animaatiota jokaiselle mahdolliselle hahmon sijainnin, ja pään asennon yhdistelmälle. Animation rigging mahdollistaa pään kääntämisen proseduaalisesti, jolloin pelimoottori ottaa hallinnan hahmon päästä ja kääntää sen

osoittamaan kohteeseen, riippumatta siitä millaista animaatiota hahmon muut osat toistavat.

Animation rigging toimii Unityn animaatiojärjestelmän päällä, joten se vaatii toimiakseen animator komponentin. Toinen tarvittava osa on Rig builder, johon lisätään halutut rigit. Kun hahmossa on Animator- ja rig builder komponentit, voidaan lisätä rigit. Rig sisältää constrainit, jotka määräävät rigin objektien liikkeitä. (Unity technologies, 2022 -l)

Constrain eli rajoite, rajoittaa luun liikettä ja ottaa siitä hallinnan. Rajoitettu luu ei liiku animaation mukaan, vaan se seuraa rajoitukselle säädettyjä ehtoja. Rajoitteita on useita erilaisia, mutta niillä on joitain yhtenäisyyksiä. Rajoite vaatii rajoitettavan objektin, jota kutsutaan nimellä ”Constrained Object”. Tämä on se objekti johon rajoite vaikuttaa. Tarvitaan myös ”Source Object”, joka on objekti, joka vaikuttaa rajoitettuun objektiin. (Unity technologies, 2022 -i)

Esimerkiksi. Jos halutaan että hahmo katsoo maassa olevaa kolikkoa kääntämällä kasvonsa sitä kohti, voimme saavuttaa tämän animation rigging avulla. Hahmo tarvitsee Animator- ja rig builder komponentit. Sitten on tehtävä rig, joka kääntää hahmon päätä. Päärig vaatii rajoitteen, tähän tarkoitukseen sopii ”Multi-Aim Constraint” rajoite. Multi-Aim Constraint kääntää rajoitettua objektia kohteeseen päin (Unity technologies, 2022 -i). Tässä tapauksessa rajoitteen Constrained Object on hahmon pään luu ja Source Object on maassa oleva kolikko. Kun rajoite on säädetty oikein, rig, jossa rajoite on, on lisättävä hahmon rig builderiin. Näillä säädöilla hahmo kääntää päänsä kolikkoa kohti.

Hahmo kääntää nyt päätään, mutta Animation rigging antaa mahdollisuuden paremmalle lopputulokselle. Liikkeen saa vaikuttamaan luonnollisemmalta, jos hahmo ei käännä ainoastaan päätään, vaan hahmon muu vartalo seuraa perässä. Hahmon selkärankaan voi laittaa toisen Multi-Aim Constraint rajoitteen. Rig builder ottaa vastaan useita rigejä, joten selkärangan rigi voidaan lisätä samaan rig builderiin pään kanssa. Rig builder säilyttää rigit eri tasoilla, eli järjestys, jossa rigit lisätään, on tärkeä. Jos pää on ennen selkärankaa, pää käännetään osoittamaan kolikkoa ja sitten selkäranka käännetään seuraavaksi. Pää ei nyt osoita kolikkoon, koska pään asento ei muutu selkärangan uuden sijainnin mukaan, ja pää on

edelleen asennoitu, kuin se olisi selkärangan alkuperäisessä sijainnissa. Rigit on sijoiteltava niin, että objektit, joihin on liitettynä muita rajoitettavia objekteja, tulevat ensin.

Jos selkärankaa rajoitetaan, sen ei kuuluisi liikkua yhtä paljon kuin pään. Tähän auttaa painoarvon säätö. Rigillä on painoarvo, joka määrää miten paljon se vaikuttaa rajoitettuun objektiin. Laskemalla painoarvoa vaikutuksesta tulee pienempi. Jos pään painoarvo on 100 % ja selän 50 % Pää osoittaa suoraan kohti kohdetta ja selkä mukailee pään liikettä hieman.

Animaatioista, jotka luodaan perustuen yhden loppupisteen sijaintiin, käytetään nimeä ”Inverse kinematics” tai ”IK” (Gregory, 2014, s. 596). Kun työskennellään yleisessä 3D-mallinuksessa, käytetään tätä nimeä. Animation rigging on Unitylle uniikki työkalusarja, joka mahdollistaa Unityn tekevän IK animaatioita pelihahmoon (Unity technologies, 2022 -b).

4 Hahmon toiminta

Tässä osassa laitetaan kaikki esitetyt tekniikat yhteen ja käydään läpi uuden 3D-pelihahmon luominen alusta loppuun. Ennen kuin hahmolle tehdään mitään teknistä työtä, on tiedettävä, minkälainen hahmo ollaan tekemässä. On päätettävä hahmon ulkonäkö ja toiminnallisuus. Jos pelihahmo on pelaajan ohjaama päähahmo, suunnitteluvaihe on erityisen tärkeä. Erilaiset pelit vaativat erilaisia pelaajahahmoja. Pelaajahahmon suunnittelu on suoraan sidoksissa siihen, minkälaista peliä ollaan tekemässä. 3D-tasohyppelypeli ja kolmannen persoonan ammuntapeli voivat vaativat hyvinkin erilaisia hahmoja. Hahmon suunnittelu on harvoin vain yhden henkilön tehtävänä. Pelkästään hahmon ulkonäköön vaikuttaa konseptitaiteen piirtäjät, animaattorit ja 3D-mallintajat. Tämän lisäksi hahmon ulkonäköön saattaa vaikuttaa tuottajien ja muiden raharuhtinaiden mielipiteet. Kun suunnitellaan hahmon toimintaa vaikuttajien määrä kasvaa entisestään. Pelisuunnittelija suunnittelee, miten hahmo käyttäytyy pelimaailmassa ja minkälaisia toimia se pystyy tekemään. Animaattorin on luotava hahmolle animaatiot, jotka vastaavat suunnittelijan näkemystä hahmon toimista. Ohjelmoijan tehtävä on saada pelihahmo toimimaan pelissä suunnittelijan ohjeiden mukaan. Ja myös saamaan hahmo toistamaan animaatiot sopivaan aikaan.

Suunnittelussa auttaa, jos pelille on kirjoitettuna Game design document. Game design document on dokumentti, johon on kirjoitettu yksityiskohtainen kuvaus pelin järjestelmistä ja toiminnasta (Adams, 2014). Dokumenttia päivitetään jatkuvasti pelin kehityksen aikana ja sen on tarkoitus toimia oppaana kehitystiimille sekä pitää suunnitelmat selvässä järjestyksessä.

Game design documentin osa, joka käsittelee pelihahmoa voisi olla seuraavanlainen ”Peli on 3D-tasohyppelypeli. Pelaajahahmon tehtävänä on päihittää hirviöt ja kerätä aarteita. Pelaaja voi valita pelihahmonsa kahdesta eri vaihtoehdosta. Pelihahmot ovat identtisiä ominaisuuksiensa puolesta. Ainoastaan hahmon ulkonäkö on erilainen. Pelaajahahmo kerää aarteita haavilla ja päihittää vihollisia lyömällä niitä lankulla. Pelaaja voi vaihtaa käyttämäänsä työkalua napin painalluksella. Kun lankku on valittuna, hyökkäysnapin painaminen toistaa heilutusanimaation ja hahmon edessä olevat viholliset vahingoittuvat, jos ne ovat tarpeeksi lähellä pelaajaa. Kun haavi on valittuna, hyökkäysnapin painaminen toistaa heilutusanimaation ja hahmon edessä oleva aarre kerätään, jos se on tarpeeksi lähellä pelaajaa. Pelihahmo pystyy hyppäämään. Kun hyppynappia painetaan, toistetaan hyppyanimaatio ja pelihahmo nouse ilmaan. Pelihahmon suuntaa voi ohjata hypyn aikana, mutta hahmo ei pysty hyppäämään ilmassa. Kun hahmo laskeutuu, toistetaan laskeutumisanimaatio, mutta vain silloin, jos hahmo ei liiku. Jos hahmo liikkuu laskeutuessaan, laskeutumisanimaatiota ei toisteta, vaan mennään suoraan liikkumisanimaatioon. Hahmo liikkuu peliohjaimen analogisella sauvalla. Hahmon nopeus riippuu siitä, miten paljon sauvaa liikutetaan. Hitaassa liikkeessä hahmo toistaa kävelyanimaatiota, ja nopeassa liikkeessä hahmo toistaa juoksuanimaatiota. Jos hahmo törmää viholliseen tai johonkin muuhun vaaraan, hän menettää pisteitä elämämittaristaan ja toistaa animaation, joka ilmaisee vahingon ottamista. Jos elämämittari menee nollaan, hahmo joutuu takaisin pelin alkuun. Hahmo voi pidentää elämämittariaan poimimalla kypärän. Kypärä poimitaan liikuttamalla hahmo sen päälle, jolloin se katoaa maasta ja ilmestyy hahmon päähän. Jos hahmo ottaa liikaa vahinkoa, kypärä putoaa hahmon päästä.”

Tällaisesta dokumentista 3D-mallintaja saa tietää, että tarvitaan useita 3D-malleja. Malli molemmille pelattaville hahmoille ja heidän työkaluilleen, sekä malli poimittavalle kypärälle. Dokumentin pitäisi myös paljastaa onko hahmoilla joitain erityisominaisuuksia, jotka on

otettava huomioon. Esimerkiksi jos pelaajahahmolla on kyky kasvattaa ylimääräinen käsi, 3D-mallintajan on hyvä tietää siitä.

Tätä projektia varten tehtiin kaksi pelaajahahmoa. Molemmat hahmot ovat ihmisiä, joten ne toimivat humanoid rigin kanssa. Hahmot poikkeavat toisistaan koon ja muodon perusteella, jotta pystyttäisiin tarkastelemaan miten erimuotoisten hahmojen animaatioita voi käyttää toistensa kanssa. Hahmot mallinnettiin Blender-ohjelmassa, seuraten tässä opinnäytetyössä esitettyjä periaatteita. Malleille tehtiin geometria, UV-kartoitus ja pintamateriaali. Mallit tehtiin Box modelling tekniikalla, jossa hahmot muotoiltiin karkeista geometrisista muodoista. Hahmot ovat symmetrisiä, joten mallinnuksessa hyödynnettiin peilaus-modifieria, joka luo mallista peilikuvan valitussa suunnassa. Peilauksen avulla tarvitsee tehdä vain mallin toinen puoli, puutuva puoli peilautuu mallin toiselle puolelle ja kopioi kaikki muutokset toiselta puolelta.

Hahmot rigattiin käyttämällä valmista ihmisen mallista luurankoa. Hahmot eivät noudattaneet normaalin ihmisen mittasuhteita, joten luurankoa oli muokattava sopimaan malliin. Blenderissä on ominaisuus, jossa kopioitujen luiden nimet voi asettaa käänteisiksi. Esimerkiksi oikea käsi voi olla nimeltään ”Arm.r”, mutta kun sen kopioi, nimen voi kääntää niin että luu tunnistetaan vasemmaksi kädeksi, ja nimeksi tulee ”Arm.l”. Näin tarvitsee tehdä luurangosta vain toinen puoli, joka kopioidaan, käännetään ja liitetään yhteen luomaan kokonainen luuranko, jonka ohjelma tunnistaa sisältävän oikean ja vasemman puolen. Peilaus-modifierin avulla on asetettava painoarvot vain mallin toiselle puolelle, ohjelma peilaa painoarvot mallin toiselle puolelle, jolloin weight painting on tehtävä vain puolelle mallista. Tämä tietysti toimi vain, jos malli on symmetrinen. Painoarvot peilautuvat vain, jos mallin molemmat puolet ovat toistensa peilikuvia.

Riggauksen jälkeen malli olisi valmis pelimoottoriin viemistä varten, ja animaatioiden tekoon. Koska pelihahmon toiminta vaatii niin monen eri osa-alueen tehtäviä, olisi järkevää järjestää tapaaminen, jossa pelisuunnittelija, ohjelmoija ja animaattori keskustelevat siitä mitä hahmo pystyy tekemään. Näin ohjelmoija tietää minkälaisia toiminnallisuuksia hahmolle tehdään, ja animaattori tietää mille hahmon toimille on tehtävä animaatiot.

4.1 Hahmon animaatiot

Hahmon animaatiot tuovat hahmon eloon ja antavat tärkeää tietoa pelaajalle. Oletetaan että pelissä on este, joka saa hahmon kompastumaan, ja olemaan etenemättä, kunnes hahmo nousee taas jaloilleen. Miten pelaajalle tehdään selväksi, että hahmo on kaatunut ja ei voi edetä ennen kuin nousee taas ylös? Ruudulle voisi laittaa tekstin, jossa lukee ”Hahmo on kaatunut, odota kunnes hän nousee.”. Toinen vaihtoehto olisi hyödyntää animaatioita. Hahmolle voi tehdä kaatumisanimaation, joka toistetaan, kun hahmo osuu esteeseen. Hahmoa ei voida liikuttaa, ennen kuin animaatio, jossa se nousee jaloilleen, on toistunut. Tämä auttaa hahmon uskottavuudessa. Koska hahmo reagoi esteeseen odotetusti kaatumalla maahan. Se myös antaa pelaaja tietää milloin hän voi jatkaa, katsomalla milloin hahmo nousee taas jaloilleen. Teksti, jossa lukee kauanko hahmolle kestää nousta, voi irrottaa pelaajan pelikokemuksesta, mutta nousuajan ilmaisu animaatiolla tuntuu luontevalta osalta peliä. Jos peli on nopeatempoinen nettimoninpeli, pelaaja saattaa haluta tarkan numeerisen arvon nousuajalle, jotta voi suunnitella etukäteen seuraavan liikkeensä. Animaatiot voivat ilmaista monia hahmon toimia, kuten vaikka esineiden poimimista, hyppäämistä, liikkumisnopeutta ja paljon muuta. Animaatiot myös auttavat hahmon uskottavuudessa saamalla hahmon näyttämään liikkeillään pelaajan komentojen ja ympäristön vaikutuksen hahmoon, joka saa hahmon tuntumaan luontevalta osalta pelimaailmaa.

Game design documentista saadaan selville mitä animaatioita hahmo tarvitsee. Koska käytössä on humanoid rig, yhdelle hahmolle tehdyt animaatiot sopivat molemmille. Tämä tarkoittaa, että animaatiot on tehtävä vain kerran, ja ne voi käyttää toiselle hahmolle. Tai mille tahansa muulle hahmolle, joka käyttää humanoid rigiä. Tämä myös tarkoittaa että, hahmoille voisi ladata animaatiot valmiista nettikirjastosta. Jos tehdään näin, on huomioitava animaatioiden sopivuus. Esimerkkihahmot ovat mittasuhteiltaan liioiteltuja ja grafiikaltaan piirrostyylisiä. Realistiselle ihmishahmolle tehdyt animaatiot saattaisivat näyttää kummalliselta näissä hahmoissa. Tätä työtä varten hahmojen animaatiot tehtiin alusta alkaen. Hahmot animoitiin Blender-ohjelmassa. Ohjelma sisältää työkalut animaatioiden tekemistä varten. Kaksi esimerkkihahmoa käyttävät samoja animaatioita

kaikissa muissa toimissa, paitsi liikkumisessa. Hahmoilla on erilaiset juoksuanimaatiot, joka auttaa ilmaisemaan hahmojen persoonallisuutta.

Mallit tuotiin .fbx tiedostona Unityyn. Animaatiot kulkevat .fbx tiedoston mukana. Animaatiot voidaan tuoda samalla aikajanalla, joka sisältää kaikki animaatiot yhdessä pötkössä. Tämän ”Pötkön” voi sitten erotella tuontivaiheessa animation clipeiksi. Erottelu tapahtuu antamalla tietty aikaväli aikajanalta, joka tallennetaan oman animation clippinään. Toinen vaihtoehto on tallentaa Blenderissä jokainen animaatio omaan ”Action” aikajanaansa. Kun .fbx tiedoston tuo Unityyn, Unity erottelee actionit automaattisesti. Erotellut actionit voidaan tallentaa animation clippeinä. Animaatioita ei ole pakko tehdä erillisessä ohjelmassa. Kappale, jossa on Animator komponentti pystyy myös tallentamaan animaatioita, jotka tehdään Unity-editorissa. On animaattorin päätettävissä, onko ulkoisen ohjelman käyttäminen, ja animaatioiden tuominen tehokkaampaa, kuin animaatioiden tekeminen Unity-editorissa.

Kun malli tuodaan Unityyn, sen avatarin tyyppi päätetään. Näillä kahdella hahmolla on eri muotoiset luurangot, mutta niiden on silti jaettava animaatioita keskenään. Tämä vuoksi hahmoille luodaan humanoid tyyppiset avatarit. Hahmojen luurangoista poimitaan luut, jotka vastaavat ihmisen ruumiinosia. Koska mallit tehtiin valmiiseen luurankoon, joka noudattaa läheisesti ihmisen luurankoa, Unity osaa poimia sopivat luut automaattisesti. On tietysti hyvä tapa tarkistaa automaattisesti määritellyt luut, ja tehdä korjauksia tarvittaessa.

Animoitavalle hahmolle lisätään animator komponentti. Ja sitten sille tehdään animation controller, johon valmiit animation clipit laitetaan. Animation clip sisältää toistettavan animaation, animator laittaa animation clipin vaikuttamaan hahmoon, ja animation controller kertoo milloin animation clip toistetaan. Tätä työtä varten pystytään hyödyntämään valmista third person controllerin animation controlleria. Se kuitenkin vaatii pieniä muokkauksia sopiakseen tähän projektiin. Animation controlleriin on lisättävä animaatio työkalun heiluttamista varten. Heilutusanimaatio lisätään animation controlleriin, ja kytketään muihin animaatioihin siirtymillä. Heilutusanimaatio tarvitsee siirtymän animaatioon ja pois animaatiosta. Animaation siirtymistä varten tehdään uusi ehto. Animation controlleriin lisätään uusi trigger, joka laitetaan ehdoksi heilutusanimaation

aloitukselle. Kun animaatio halutaan toistaa, koodi, joka ohjaa työkalun käyttöä lähettää animation controllerille viestin, jossa se käskee laukaisemaan heilutustriggerin. Tämä aloittaa heilutusanimaation.

4.2 Hahmon modulaarisuus

Hahmo toistaa tiettyjä toimia useaan kertaan. Hahmo heiluttaa lankkua ja haavia samasta napista ja kaksi pelaajahahmoa ovat toiminnaltaan identtisiä. Koska hahmoilla on niin monta toimea, jotka muistuttavat toisiaan, ei ole järkevää tehdä jokaista toimea alusta alkaen. Tähän auttaa modulaarisuus. Kun hahmo tehdään modulaariseksi, ominaisuuksia voi lisätä, poistaa ja vaihtaa. Toiminnon sisältävän moduulin voi kopioida toista käyttötarkoitusta varten helposti, jolloin työtaakka kevenee.

Kaksi pelaajahahmoa ovat identtisiä toimintansa puolesta. Ainoa ero niiden välillä on hahmojen ulkonäkö ja animaatiot. Tässä olisi kaksi moduulia. Hahmon toiminta, joka sisältää hahmon koodin, ja toiminnallisuudet. Ja hahmon ulkonäkö, joka sisältää hahmon 3D-mallin, luurangon ja animation controllerin. Paras tapa olisi ottaa hahmon toiminnallisuus ja vaihtaa siihen hahmon ulkonäkö tarpeen mukaan. Hahmon ulkonäköön kuuluu hahmon malli ja luuranko. Hahmoilla on omat animation controllerit, jotka ovat sidoksissa hahmon luurankoon. Hahmon toiminnallisuus on siis moduuli, johon liitetään hahmon ulkonäkö, eli malli, luuranko, animation controller ja avatar.

Myös hahmon työkalut voidaan toteuttaa moduuleina. Kypärä ja kädessä olevat työkalut eivät ole osa hahmoa, ne ovat moduuleja, jotka liitetään hahmoon tarvittaessa. Hahmon ulkomuoto muuttuu antamaan tietoa pelaajalle. Esimerkiksi kun pelaaja näkee, että hahmolla on kypärä pääsään, pelaaja tietää, että hahmolla on nyt ylimääräisiä elämäpisteitä. Kun kypärä on erillinen moduuli, sen voi poistaa ja lisätä hahmolle tilanteen mukaan.

Helpoin tapa toteuttaa kaksi pelihahmoa, olisi tehdä kaksi kopiota samasta pelihahmosta, ja sitten vaihtaa toiseen niistä hahmon malli, luuranko, animation controller ja avatar. Kopiot voidaan sitten tallentaa kahdeksi eri prefab-objektiksi, joista kutsutaan haluttu hahmo, kun hahmo valitaan. Toinen vaihtoehto olisi käyttää yhtä prefab-objektia. johon on tallennettu

molemmat hahmot. Mutta niin että vain yhden hahmon objektit ovat kerrallaan aktiivisina. Hahmojen toisistaan eroavat osat, eli mallit ja animation controller, voidaan sitten aktivoida script-komponentista löytyvän koodin avulla. Tieto siitä kumpi hahmo halutaan aktivoida, voidaan vaikka tallentaa scriptable objektiin. Luodaan luokka, joka sisältää parametrit prefabille ja halutulle hahmolle. Prefab kutsutan paikalle ja sitten luetaan scriptable objektista kumpi hahmo aktivoidaan. Scriptable objektia käytettäessä ei tarvitse käyttää molemmille hahmoille samaa prefabia. Prefab-parametriin kelpaa mikä tahansa objekti. Scriptable objektiin voisi myös tallentaa hahmon muita ominaisuuksia. Liikkumisnopeudelle, hyppykorkeudelle ja mille tahansa ominaisuudelle voidaan tehdä omat parametrit, joista luetaan hahmoon laitettavat arvot. Molemmat esimerkkihahmot ovat ominaisuuksiltaan samanlaisia, joten tällaiselle järjestelylle ei ole tarvetta.

Hahmon kypärä toimii samalla tavalla molemmilla hahmoilla. Se ilmestyy päähän, kun se poimitaan ja putoaa päästä, kun hahmo ottaa tarpeeksi vahinkoa. Päähän ilmestymistä varten pelin on tiedettävä missä hahmon pää sijaitsee. Hahmon luurangossa on oma luu päätä varten, mutta luurangon muoto ei välttämättä täsmää täysin mallin muotoa. Tätä varten voidaan hyödyntää child-objekteja. Pääluun child-objektiksi voidaan laittaa peliobjekti, joka kertoo kypärän halutun paikan. Tämä ”paikkaobjekti” sijoitetaan tarkalleen siihen paikkaan, johon kypärän halutaan ilmestyvän. Koska paikkaobjekti on pääluun child-objekti, se seuraa aina pään liikettä. Jos kypärä asetetaan paikkaobjektin child-objektiksi, sekin seuraa pään liikettä. Kypärän saa ilmestymään tarkalleen haluttuun paikkaan, tallentamalla koodiin viittaus paikkaobjektin sijainnista ja sitten luomalla kypärä haluttuun sijaintiin. Kypärän olisi myös pudottava hahmon päästä silloin kun hahmo ottaa liikaa vahinkoa. Kypärän saa putoamaan lisäämällä sille Rigidbody-komponentin, joka altistaa peliobjektin pelimoottorin simuloidun painovoiman vaikutukselle. Vaikka kypärä putoaakin painovoiman vaikutuksesta, se on edelleen pääluun child-objekti, joten se seuraa pääluun liikkeitä. Tämä johtaa siihen, että kypärä näyttää ”leijuvan” hahmon perässä. Jos hahmon päässä olevan kypärän halutaan putoavan maahan uskottavasti, tarvitaan koodi, joka muuttaa sen parent-objektin niin että kypärä ei ole enää sidoksissa pääluuhun ja samalla aktivoi objektin Rigidbody-komponentin, joka saa sen putoamaan hahmon päästä. Esimerkkihahmo on toteutettu hieman eri tavalla, kypärän toiminnassa on ”huijattu” vähän. Kypärä, joka putoaa hahmon päästä, ei ole sama kypärä joka hahmolla on päässään.

Esimerkkihahmolla on pääluun child-objektiksi asetettu kypärän malli. Tämä malli ei kuitenkaan lähde minnekään, kun se ”putoaa” hahmon päästä. Kun kypärä ”putoaa”, sen Mesh renderer-komponentti poistetaan käytöstä, joka tekee kypärästä näkymättömän. Hahmolla on siis koko ajan päässään näkymätön kypärä. Kun hahmo poimii uuden kypärän, hahmon päässä jo valmiiksi oleva kypärä tulee näkyväksi. Hahmon päästä putoava kypärä on erillinen prefab-objekti, joka kutsutaan kypärän sijaintiin samalla hetkellä kun ”oikeasta” kypärästä tulee näkymätön. Tällä prefabilla on valmiiksi rigidbody komponentti ja koska sen ainoa tehtävä on pudota, komponentti on valmiiksi aktiivinen. Näkymätön kypärä on tässä tilanteessa helpompi käyttää, koska se tarvitsee vain kytkeä päälle tai pois. Tämä pätee vain, jos hahmo voi poimia ainoastaan yhdenlaisia päähineitä. Jos päähineitä on useita, voi olla helpompaa kutsua uusi malli jokaiselle päähineelle, sen sijaan että hahmo pitää päässään näkymätöntä versiota jokaisesta vaihtoehdosta ja säilyttää niistä jokaiselle oman pudotettavan prefab vastineen. Esimerkkihahmon kypärään liittyvät toiminnallisuudet ovat kaikki yhden root-objektin alla. Koko kypärän toiminnan voi siis helposti tallentaa prefab-objektiksi ja lisätä sen kumpaankin pelaajahahmoon. Tai mille tahansa hahmolle tai esineelle, jonka haluttaan käyttävä kypärää. Pääluu antaa ainoastaan sijainnin tässä tapauksessa, joten humanoid rigiä ei tarvita. Kypärän voi siis monistaa mille tahansa peliobjektille. Samoja periaatteita voidaan hyödyntää hahmon työkaluihin. Haavi ja lankku voidaan asettaa käsiluun child-objektiksi. Työkalut vaihdetaan vuorotellen näkyviksi ja näkymättömiksi, riippuen siitä kumpi niistä on merkitty aktiiviseksi koodissa.

4.3 Valmis hahmo

Kuvassa 11 on ruutukaappauksia, joissa valmiit pelihahmot näkyvät pelimaailmassa. Vasemmalla yläkulmassa on ensimmäinen hahmo toistamassa ”idle” animaatiotaan. Vasemmalla alakulmassa on sama hahmo toistamassa juoksuanimaatiotaan. Hahmon kypärä puuttuu tästä kuvasta ja hahmon työkalu on vaihtunut. Kypärä ilmestyy ja poistuu hahmon päästä, pelitilanteen mukaan. Työkalun voi vaihtaa napin painalluksella. Hahmon jaloissa näkyy pölypilvi. Pilvi on tehoste, joka on lisätty tuomaan uskottavuutta pelihahmoon. Sen tarkoitus on luoda tuntumaa siitä, että pelihahmo aiheuttaa toimillaan vaikutuksen pelimaailmaan. Tässä tapauksessa vaikutus on, että hänen jalkansa nostattavat tomua maasta. Oikealla yläkulmassa on toinen hahmo toistamassa työkalun heilutusanimaatiota.

Animaatio on sama haaville ja lankulle. Molemmat hahmot käyttävät täysin samaa heilutusanimaatiota. Oikeassa alakulmassa on kuva hahmosta toistamassa juoksuanimaatiotaan. Vertaamalla hahmojen asentoja, näkyy, että juoksuanimaatiot ovat erilaiset. Ensimmäinen hahmo nojaa taaksepäin juostessaan ja toinen hahmo on kallistunut eteenpäin. Hahmot jakavat kaikki muut animaatiot, paitsi juoksuanimaation. Koska hahmot käyttävät eri animation controllereita, minkä tahansa hahmon animaation voi vaihtaa, jos se koetaan tarpeelliseksi. Liite 1 Hahmojen esittely, sisältää lisää kuvia hahmoista.

Kuva 11. Ruutukaappauksia pelihahmoista pelimaailmassa

Molemmat hahmot ovat ihmisiä, mutta ne pyrittiin suunnittelemaan eri muotoisiksi, jotta päästäisiin kokeilemaan, miten kaksi erimuotoista hahmoa jakavat samat animaatiot. Tämän työn tavoitteiksi on määritelty, että lopputulos on uskottava pelihahmo, jonka ominaisuuksia voi käyttää mahdollisimman paljon uudelleen. Hahmot jakavat paljon toiminnallisuutta. Molemmat hahmot käyttävät samaa koodia liikkumiseen ja jakavat monia animaatioita. Tämän lisäksi kypärän ja työkalujen toiminnallisuus on täysin sama molemmissa hahmoissa.

Hahmon uskottavuuteen auttaa heti se, että hahmot eivät ole photorealistisia ihmisiä. Yleisö hyväksyy helpommin liioitellut ja abstraktit liikkeet piirrosmaiselta hahmolta. Mutta

realistisella ihmisellä ne näyttäisivät kummalliselta. Piirroshahmo ei näytä realistiselta, mutta se on omassa ympäristössään uskottava. On väiteltävissä, miten onnistuneesti hahmot sopivat visuaalisesti ympäristöönsä. Mutta se on enemmänkin taiteellinen ongelma. Tämän työn tarkoituksena oli selvittää tekniset ratkaisut, joilla voidaan toteuttaa taitelijan näkemys toimivana pelihahmona.

Jotta hahmot voivat uskottavasti elää pelimaailmassa, niiden on reagoitava odotetusti pelimaailmaan. Tähän auttaa hahmon animaatiot. Tekemällä hahmolle paljon animaatioita, hahmolla on enemmän mahdollisuuksia tehdä liikkeitä, jotka vastaavat odotettua käytöstä pelimaailmassa. Animaatioihin liittyy myös hahmon ohjelmointi ja animation controller. Esimerkiksi kun hahmo hyppää ilmaan, koodi ja lähettää animation controllerille viestin, joka ilmoittaa, että hahmo on hypännyt, jolloin animation controller toistaa animation clipin jossa hahmo hyppää. Kaikki nämä vaiheet yhdessä saavat toimen, jossa peli nostaa hahmon mallin ylöspäin, vaikuttamaan siltä, että hahmo on ponnistanut maasta ja hypännyt ilmaan itse. Hahmo on toiminut odotusten mukaisesti, ja pelaaja uskoo, että hahmo on osa pelimaailmaa.

5 Yhteenveto

Toimivan 3D-pelihahmon tekemiseen tarvitaan paljon eri työvaiheita ja osaamista useasta eri osa-alueesta. On oltava lahjakas taiteilija, jotta osaa suunnitella näyttävän hahmon. On osattava 3D-mallinnus hyvin, jotta hahmon malli vastaa suunnitelmaa. Taitava ohjelmoija tarvitaan, jotta hahmon tekniset ominaisuudet toimivat sujuvasti. Jokaisesta työvaiheesta voisi yksinään kirjoittaa oman opinnäytetyönsä. Tämä työ tarjoaa laajan katsauksen 3D-pelihahmon tekemiseen vaadittaviin tekniikoihin, ja siihen miten ne liittyvät toisiinsa. Pelihahmon tekeminen on laaja prosessi ja sen työvaiheiden ymmärtäminen on tärkeää, koska useat vaiheet ovat sidoksissa toisiinsa. Jos yksi henkilö on vastuussa pelihahmon tekemisestä, hänen on osattava ainakin perusteet jokaisesta vaiheesta. Isoissa pelistudioissa pelihahmon tekemiseen osallistuu usein monta eri ihmistä, jotka ovat yleensä oman alansa asiantuntijoita. Koko prosessin ymmärtäminen ja käsitys siitä miten työvaiheet liittyvät toisiinsa on tärkeää myös silloin, kun hahmoa on tekemässä useampi henkilö. Ei riitä, että 3D-mallintaja osaa tehdä näyttävän mallin. On tiedettävä, miten mallista tehdään sellainen,

että sen riggauksen ja teksturoinnin voi helposti hoitaa. 3D-mallintaja, joka ymmärtää näiden työvaiheiden perusteet, osaa tehdä mallin, joka on hyvä pohja jatkotoimille. Kun pelihahmon jokainen vaihe tehdään muut työvaiheet mielessä pitäen, kaikkien hahmon tekemiseen osallistuvien henkilöiden työstä tulee sujuvampaa. Uskottava pelihahmo tarvitsee animaatiot, jotka vastaavat pelihahmon toimia pelimaailmassa. Animaatioiden on myös toistuttava odotetusti pelaajan komentojen ja pelimaailman vaikutuksesta, jotta hahmon toimet ovat uskottavia. Tätä varten on ymmärrettävä ohjelmoinnista, animaatiosta ja animation controllerin toiminnasta. Pelihahmon tekemiseen tarvittavien työvaiheiden ymmärrys auttaa myös modulaarisuuden mahdollistamisessa. Ymmärrys siitä, miten pelihahmon osat voidaan jakaa yksittäisiin komponentteihin, on tärkeää, kun tarkastellaan mitä hahmon osia voi käyttää uudelleen muissa tarkoituksissa. Mallintajan on tiedettävä mitä eri osia kuuluu pelihahmoon ja miten niitä voi hyödyntää muissa hahmoissa. Riggaus helpottuu huomattavasti, jos samaa luurankoa voi käyttää uudelleen eri hahmoille. Animaattorin on hyvä ajatella, miten tehdään animaatioita, joita voi käyttää uudelleen monelle eri hahmolle. Jos ymmärretään miten pelin rakennuspalikat toimivat pelimoottorissa, on helpompaa suunnitella niille useita käyttötarkoituksia. Selkeä käsitys kaikista pelihahmon tekemiseen kuuluvista työvaiheista antaa paremman kuvan siitä mistä eri osista hahmo koostuu, ja auttaa paremmin ymmärtämään miten näitä osia voi hyödyntää uudelleen. Vaikka yksikään hahmon tekemiseen osallistuva henkilö ei olisi asiantuntija jokaisessa osa-alueessa, on jokaisen hahmon tekemiseen osallistuvan hyvä tietää ainakin perusperiaatteet jokaisesta työvaiheesta. Koska useat hahmon luomisen työvaiheet ovat vahvasti sidoksissa toisiinsa ja muiden työvaiheiden ymmärtäminen helpottaa kokonaista työtä ja tuottaa paremman lopputuloksen.

Lähteet

Adams, E. (2014). *Fundamentals of game design*. New Riders.

Amin, J. (3.2.2014). *Introduction to rigging in Maya - Part 1 - Introduction*. Haettu 10.5.2022 osoitteesta https://3dtotal.com/tutorials/t/maya-rigging-introduction-to-rigging-jahirul-amin-animation-character-vehicle

Blender Documentation Team. (n.d.-a). *Introduction*. https://docs.blender.org/manual/en/latest/modeling/modifiers/introduction.html

Blender Documentation Team. (n.d.-b). *Decimate Modifier.* https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/decimate.html

Blender Documentation Team. (n.d.-c). *Mirror Modifier.* https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/mirror.html

Gee, Z.;& Falco, P. (2010). *3D in photoshop*. Elsevier.

Gregory, J. (2014). *Game engine architecture*. CRC Press.

Maxwell, W. (13.12.2019). *How Many Polygons Should a Game Model Have*. Haettu 7.5.2022 osoitteesta https://cgobsession.com/how-many-polygons-should-a-game-model-have/

Unity technologies. (21.3.2022 -a). *Animation Parameters*. https://docs.unity3d.com/Manual/AnimationParameters.html

Unity technologies. (22.3.2022 -b). *Animation Rigging*. https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.2/manual/index.html

Unity technologies. (21. 3.2022 -c). *Animator*. https://docs.unity3d.com/Manual/class-Animator.html

Unity technologies. (21.3.2022 -d). *Animator Controller*. https://docs.unity3d.com/Manual/class-AnimatorController.html

Unity technologies. (21.3.2022 -e). *Blend Trees*. https://docs.unity3d.com/Manual/class-BlendTree.html

Unity technologies. (21.3.2022 -f). *GameObjects*. Haettu 25.3.2022 osoitteesta https://docs.unity3d.com/Manual/GameObjects.html

Unity technologies. (24.3.2022 -g). *Importing a model with humanoid animations*. https://docs.unity3d.com/Manual/ConfiguringtheAvatar.html

Unity technologies. (21.3.2022 -h). *Introduction to components*. https://docs.unity3d.com/Manual/Components.html

Unity technologies. (24.3.2022 -i). *Multi-Aim Constraint*. https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.2/manual/constraints/MultiAimConstraint.html

Unity technologies. (21.3.2022 -j). *Prefabs*. https://docs.unity3d.com/Manual/Prefabs.html

Unity technologies. (24.3.2022 -k). *Retargeting of Humanoid animations*. https://docs.unity3d.com/Manual/Retargeting.html

Unity technologies. (24.3.2022 -l). *Rigging Workflow*. https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.2/manual/RiggingWorkflow.html

Unity technologies. (21.3.2022 -m). *ScriptableObject*. https://docs.unity3d.com/Manual/class-ScriptableObject.html

Unity technologies. (21.3.2022 -n). *Transforms*. https://docs.unity3d.com/Manual/class-Transform.html

Ward, A. (21.9.2011). *How to create character models for games: 18 top tips*. Haettu 7.5.2022 osoitteesta https://www.creativeblog.com/how-create-character-models-games-18-top-tips-9113050

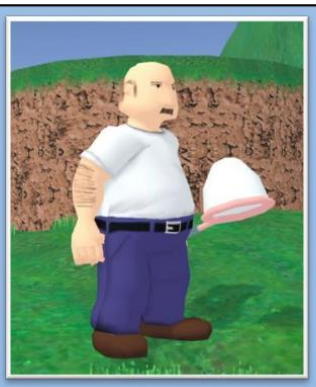
Liite 1: Hahmojen esittely

Kuvat pelihahmoista pelimaailmassa

Kuvasarja esittää heilutusanimaatiota. Ensimmäisessä kuvassa animaatio ei ole käynnistynyt. Tästä asennosta siirrytään heilautusanimaatioon, kun animation controller saa aloitusviestin. Toisessa kuvassa on animaation alkupäästä asento, jossa valmistaudutaan heilautukseen. On mietittävä tarkkaan, miten kauan tässä asennossa pysytään. Liian nopea liike ei näytä uskottavalta, mutta jos animaatioon kuluu liikaa aikaa, pelattavuus saattaa kärsiä. Kolmannessa kuvassa hahmo on heilauttanut haavia ja päätynyt kuvan asentoon. Haavi jättää jälkeensä tehosteen. Tässä vaiheessa animaatiota, tapahtuisi pelissä se toimi, joka seuraa haavin heilautuksesta.

Lankkukin aiheuttaa tehosteen

Lankun ja haavin välillä voi vaihtaa painamalla nappia. Vaihto aiheuttaa pienen animaation, jossa valittu työkalu ilmestyy käteen hieman normaalia isompana ja sitten kutistuu normaaliin kokoon. Animaatio on hienovarainen, mutta se silti antaa näkyvän palautteen vaihdosta



Kuvasarja näyttää hahmon hyppäyksen. Ensimmäisessä kuvassa hahmo seisoo paikallaa. Hyppyanimaatio ei ole vielä käynnistynyt. Toisessa kuvassa on hyppyanimaation alku. Kun hahmon koodi aloittaa hahmon ohjaimen hyppäyksen se lähettää samalla animation controllerille viestin, joka määrää aloittamaan hyppyanimaation. Kun hahmo alkaa putoamaan kolmannessa kuvassa, toistetaan putoamisanimaatio. Putoamisanimaatio ei toistu vain hypätessä. Siihen siirrytään myös silloin, kun hahmon koodi havaitsee, että hahmon jalat eivät kosketa maata. Viimeisessä kuvassa hahmo laskeutuu. Kun hahmo laskeutuu, toistetaan laskeutumisanimaatio, jossa hahmo törmää maahan. Laskeuduttaessa luodaan partikkeliefekti, jonka tarkoitus on näyttää siltä, että hahmon törmäys maahan nostattaa tomua ilmaan. Jos pelaaja liikuttaa hahmoa laskeutumisen aikana, siirrytään suoraan juoksuanimaatioon. Partikkeliefekti toistetaan molemmissa tapauksissa.




Kypärän voi poimia pelimaailmasta. Poimittava kypärä leijuu pelialueella. Partikkeliefekti antaa kypärälle hehkun. Kun pelihahmo koskettaa kypärää, se katoaa pelimaailmasta ja hahmon päähän ilmestyy kypärä. Kypärän ilmestymistä tehostaa partikkeliefekti


Kun kypärä pudotetaan, toistetaan sama tehoste kuin kypärää poimiessakin. Tällä kertaa kypärä katoaa hahmon päästä ja ilmestyy pelimaailmaan erillisenä objektina. Pudotettu kypärä laskeutuu jonnekin lähettyville ja katoaa vähän ajan päästä.


Molemmat hahmot voivat poimia kypärän