# Videopelihahmon 3D-mallinnettu tuotanto ja implementaatio Blenderillä ja Unityllä

**Jeremias Jokila**

2022 Laurea

**Laurea-ammattikorkeakoulu**

# Videopelihahmon 3D-mallinnettu tuotanto ja implementaatio Blenderillä ja Unityllä

Jeremias Jokila

Tradenomi (Tietojenkäsittely)

Opinnäytetyö

Marraskuu, 2022

**Laurea-ammattikorkeakoulu** **Tiivistelmä**

Tradenomi (Tietojenkäsittely)

Tietojenkäsittelyn koulutusohjelma (AMK)

Jeremias Jokila

**Videopelihahmon 3D-mallinnettu tuotanto ja implementaatio Blenderillä ja Unityllä**

Vuosi 2022 Sivumäärä 36

---

Opinnäytetyössä tutkittiin ja testattiin 3D-mallinnetun videopelihahmon tuotantoa Blender- ja Unity- työkaluilla. Työn tavoitteena oli demonstroida tyypillistä videopelihahmon tuotannon prosessia ja esittää sen avainaskeleet sekä niihin liittyvää teoriaa. Työllä ei ollut toimeksiantajaa ja se ohjattiin palvelemaan työn toteuttajan taitojen kehitystä.

3D-mallinnukseen työssä hyödynnettiin Blender-sovellusta ja implementaatio suoritettiin Unity-pelimoottorissa. Työn päätuotoksena oli tähtikuonokontiaisen pohjalta suunniteltu teksturoitu videopelihahmo, jolla on kävelyanimaatio. Tämä videopelihahmo implementoitiin peliprojektiin Unityn sisällä. Työn toteutuksen aineistona toimivat sekä aiheen kirjallisuus, että kokemusperäinen tieto. Opinnäytetyö toimi taidonnäytteenä prosessin käytäntöjen sekä teorian omaksumisesta, ja sen tuotokset lisättiin työn toteuttajan portfolioon hyödynnettäväksi työnhaussa.

Asiasanat: 3D-mallintaminen, animaatio, pelinkehitys, unity, blender

**Laurea University of Applied Sciences** **Abstract**

Bachelor's Degree Programme in Business Information Technology

Jeremias Jokila

**Production & Implementation of a 3D-modelled Video Game Character Using Blender and Unity**

Year 2022 Pages 36

In this thesis, the topic of research and testing was the production process of a 3D-modeled video game character using Blender and Unity as tools. The goal of the thesis was to demonstrate the process typical to video game character production and to present the associated key steps, as well as theory related to them. This thesis did not have an external client and as such was directed to serve personal skill development for the person carrying out the process.

For 3D modeling, this work utilized the Blender application whilst the implementation steps were carried out inside the Unity game engine. The main resulting product of the work was a textured video game character designed based on a star-nosed mole, with a walking animation. This video game character was implemented inside a game project in Unity. The knowledge base used in this work is based on both relevant literature and experiential knowledge. This thesis served as a demonstration of understanding for both the process practices and theory, and its output products were added to the work writer's portfolio for use in finding employment.

Keywords: 3D-modelling, animation, game development, unity, blender

Sisällys

# 1 Johdanto

Tässä opinnäytetyössä käsitellään videopelihahmon tuotantoprosessia 3D-mallintamisen näkökulmasta kattaen prosessin olennaiset vaiheet. Aihe on valittu erityisesti henkilökohtaiseen aikaisempaan kokemukseen ja kiinnostukseen pohjautuen. Olen entuudestaan tehnyt vastaavanlaisia 3D-mallinnukseen liittyviä projekteja pienemmässä mittakaavassa tutustumiskeinona 3D-mallintamisen käytänteisiin.

Tämän opinnäytetyön tavoitteena on esittää ja kattaa videopelihahmon mallinnuksen eri vaiheet kokonaisvaltaisesti ja avata sen taustalla olevia teknologisia periaatteita, sekä tämän lisäksi avata 3D-mallien implementaatioon vaikuttavat pääpiirteet pelimoottorissa.

Opinnäytetyön lopputuloksena on 3D-malli videopelihahmosta, jolla on kävelyanimaatio, ja joka esiintyy Unity-projektin kontekstissa. Hahmoa implementoidaan Unity-projektiin, mutta tämän projektin pohjalta ei muodosteta kokonaista peliä, vaan sitä käytetään esimerkkinä prosessista. Tuotokset eivät sisällä tämän prosessin ulkopuolisia elementtejä.

Henkilökohtaisesti tämän projektin tekijänä koen tämän aiheen tarjoavan erityismahdollisuuden oppia lisää aiheesta, sillä olen entuudestaan paneutunut ainoastaan perinteiseen piirrettyyn animaatioon. Tavoitteenani tämän aiheen valitsemisessa oli valita aihe, josta saan henkilökohtaisesti suurimman mahdollisimman hyödyn tulevaisuudentavoitteideni kanssa.

# 2 Keskeinen Terminologia

**Blender**

Suosittu ilmaiseksi saatavilla oleva avoimen lähdekoodin 3D-mallintamisohjelmisto

**Unity**

Suosittu ilmaiseksi saatavilla oleva pelinkehitykseen soveltuva pelimoottori.

**Pelimoottori**

Ohjelmistoympäristö, joka suorittaa pelin koodin sekä saattaa sisältää muita toiminnallisuuksia.

**Karteesinen koordinaatisto**

Koordinaatisto, jolla voidaan kuvata kappaleen tai pisteen sijainti sen arvoina tila-avaruudessa koordinaatiston akseleihin verrattavissa olevia arvoja hyödyntäen. 3-ulotteisessa koordinaatistossa (pystysuunta, vaakasuunta, syvyys) yleisimmät nimet akseleille ovat X, Y ja Z.

**Eulerin kulma**

Eulerin kulma on koordinaatistoon suhteessa oleva rotaatio, joka määritellään koordinaatiston akseleista siten, että kappaleen rotaatio lasketaan kulman asteluvussa eroavuudesta jokaisen olemassa olevien akselin ominaissuunnasta. Eulerin kulmilla voidaan määritellä jokainen käytössä olevan koordinaatiston sisällä mahdollinen rotaatio.

**Primitiivi / Primitive**

Manipuloitavissa oleva geometrinen elementti, joka toimii 3D-mallin osana. Verteksit, reunat ja tahkot ovat primitiivejä.

**Verteksi / Vertex**

Pienin geometrinen primitiivi, jota 3D-mallinnuksen kontekstissa käsitellään. Kolmiulotteisessa tilassa oleva 0-ulotteinen elementti, jolla on sijainnin koordinaatit. Toimii polygonin kulmana ja reunan päätepisteenä.

**Reuna / Edge**

Kahden verteksin välillä oleva vektori. Toimii yhden tai useamman polygonin reunana.

**Taso / Plane**

3-ulotteisen tilan kahtia jakava 2-ulotteinen pinta, jolla on ääretön pinta-ala.

**Tahko / Face**

Reunoilla tasosta rajattu osa. Polygonin täyttävä taso.

**Polygoni / Polygon**

Kolmiulotteisessa tilassa oleva kaksiulotteinen suljettu monikulmio, jolla ei ole paksuutta. Polygonit muodostavat polygonimallin pinnan. Puhekielessä teknisestä eroavuudesta huolimatta polygonia ja tahkoa saatetaan käyttää tarkoittamaan samaa asiaa, sillä kaikki polygonit sisältävät tahkon.

*Tri*: Kolmiverteksinen polygoni

*Quad*: Neliverteksinen polygoni

*N-gon*: Yli neliverteksinen polygoni

**Polygonimalli / Mesh**

Useasta polygonista muodostettu kolmiulotteinen kappale. Voi muodostaa yksinään, tai yhdessä usean polygonimallin kanssa kokonaisen 3D-mallin.

**Tekstuuri**

3D-mallin väritys. Usein tekstuuria käytetään puhekielessä terminä viittaamaan kuvatekstuuriin- tekstuurin tyyppiin, jossa kuvatiedosto projisoidaan mallin pinnalle.

**Luu / Bone**

3D-mallinnuksen kontekstissa rigin osana toimia ”vipu”, joka on sidottu polygonimalliin, ja jota voidaan käyttää 3D-mallin animoimiseen. Verrattavissa perinteisen stop motion animaatiossa hyödynnettäviin armatuureihin.

**Rig**

Yhdestä- tai useammasta luusta muodostuva ”luuranko”, jota manipuloimalla voidaan animoida tai poseerata 3D-mallin liikkuvia osia.

**3D-malli / 3D Model**

Digitaalinen kolmiuloitteinen malli, joka voi sisältää yhden- tai useamman polygonimallin, tekstuurin ja/tai rigin.

**GDD / Game Design Document**

Pelin suunnittelua ohjaava dokumentti, joka asettaa tavoitteet teeman, toiminnallisuuden, käytettävien teknologioiden, aika tavoitteiden sekä muiden suunnittelun huolenaiheiden osalta.

**UV-kartta**

3D-mallin pinta kartoitettuna 2-uloitteisesti. Hyödynnetään kuvien kartoittamisessa mallin pinnalle.

**Renderöinti / Rendering**

Prosessi, jossa tietokone tulkitsee sille syötetyn 3D-mallin tietokoneen näytöllä esitettäväksi.

**Avainpiirros / Key frame**

Perinteisestä animaatiosta lainattu termi. Hahmon liikkeiden kannalta suuren tärkeysasteen asento.

**Proseduraalinen animaatio**

Animaation määrittely koodilla ajoaikaisesti.

**Kuvataajuus**

Videolla tai animaatiossa sekunnissa toistettujen kuvien määrä, usein merkittynä englanninkielisellä lyhenteellä FPS (Frames Per Second)

**Unity - Scene**

Scene on Unityn sisällä oleva valmis 2-, tai 3-ulotteinen ympäristö, jonka sisällä Unity suorittaa pelin toiminnallisuuden.

**Unity – Blend Tree**

Järjestelmä Unityssä, joka interpoloi liikettä siihen syötettyjen animaatioiden välillä riippuen siihen syötetyn muuttujan arvosta.

# 3 Ohjelmistot

## 3.1 Blender

Tässä opinnäytetyössä 3D-mallintamiseen hyödynnetään Blender Foundation -organisaation julkaisemaa avoimen lähdekoodin Blender 3D-mallinnusohjelmaa. Blender-ohjelmisto on saavuttanut erityistä suosiota johtuen sen helposta saatavuudesta ja siihen kohdistuvasta jatkuvasta tuesta ja päivityksistä Blender Foundationin osalta. Blender onkin toiminnallisuudeltaan verrattavissa moniin alan ammattilaistason ohjelmistoihin. Blender on ladattavissa ilmaiseksi www.blender.org -verkko-osoitteesta. (Blender 2022.)

Blender sisältää toiminnallisuudeltaan laajan kirjon työkaluja moneen eri tarkoitukseen myös 3D-mallintamisen ulkopuolella. Tässä opinnäytetyössä ei syvennytä itse työkaluihin tai niiden tarkkaan funktionaalisuuteen, vaan käsitellään Blenderin 3D-mallintamiseen keskittyvillä työkaluilla aikaansaatavia tuloksia, sekä lyhyesti niiden taustalla olevaa teoriaa mallintamisen, teksturoinnin, rigauksen, ja animaatioiden luomisen osalta.

### 3.2 Unity

Tässä opinnäytetyössä tuotetun 3D-mallin implementaatioon hyödynnetään Unity Technologies -yrityksen Unity-pelimoottoria. Unity on yksi suosituimmista ilmaiseksi saatavilla olevista yleiskäytössä olevista pelimoottoreista ja onkin saavuttanut suurta suosiota sekä yritys-, että yksityiskäytössä. Unityssä on käytettävissä suuri kirjo valmiita toiminnallisuuksia sekä kattavat ohjelmakirjastot C#-kielelle edesauttaen tehokasta pelinkehitystä Unityssä. Unity on ladattavissa ilmaiseksi www.unity.com -verkko-osoitteesta. (Unity 2022.)

Blenderin 3D-mallintamisprosessin jälkeen tässä opinnäytetyössä kerrotaankin siis 3D-mallien tuomisesta Unity-pelimoottoriin ja niiden implementaatiosta Unity-projektin kontekstissa, sekä jonkin verran Unityn valmiista toiminnallisuuksista, jotka ovat tässä kontekstissa relevantteja. Tässä opinnäytetyössä ei kuitenkaan syvennytä yleismaailmallisesti Unityn sisällä pelin ohjelmointiin.

## 4 3D-Mallinnuksen perusteet

### 4.1 3D-Mallinnuksen teoria

Digitaalisessa 3D-mallintamisessa mallintamisen perusoperaatiot suoritetaan manipuloimalla 3D-mallin geometristä tietoa kolmiulotteisen karteesisen koordinaatiston- vaakasuuntaisella-, pystysuuntaisella- ja syvyysakseleilla- yleisimmin merkittynä X, Y, ja Z (Puhakka 2008, 29–31). 3D-mallin elementtejä voidaan siirtää näillä akseleilla mielivaltaisesti yksinkertaisilla siirtomuunnoksilla lisäämällä tai vähentämällä kappaleen- tai kappaleen elementin koordinaateista haluttu muutosvektori (Puhakka 2008, 91). Kappaletta voidaan myös kiertää intuitiivisesti Eulerin kulmia hyödynten. Eulerin kulmien mukainen kiero määritetään kulman eroavuutena suhteessa jokaiseen koordinaatiston akseleista. Täten voidaan tarkasti kuvata mikä tahansa haluttu orientaation kappaleelle lisäämällä näiden kulmien kierrot yhteen orientaation muutoksena. (Puhakka 2008, 106.)

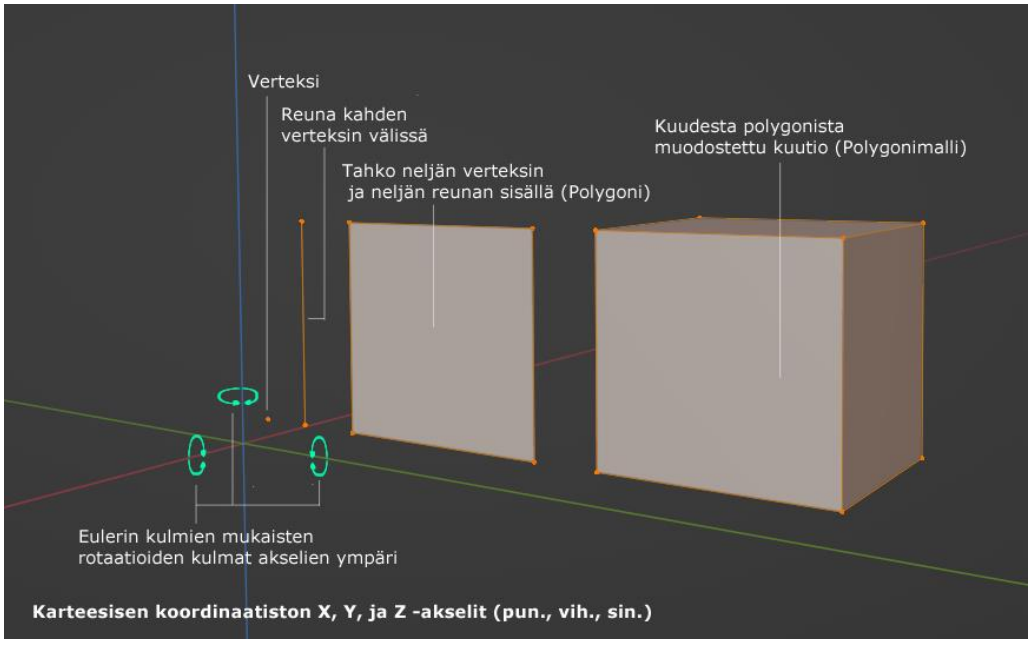
Manipuloitavista geometrisistä peruselementeistä- primitiiveistä yksinkertaisin on verteksi. Verteksi on 3D-mallintamisessa vakiintunut termi, joka vastaa pistettä 3-uloitteisessa avaruudessa. Verteksi on 0-ulotteinen primitiivi- tarkoittaen sitä, että sillä ei ole leveyttä, korkeutta, tai pituutta, mutta tästä huolimatta sillä on sijaintiaan edustava koordinaatti. Verteksin pääfunktio osana 3D-mallin rakennetta onkin siis toimia koordinaattina 3D-mallin korkeamman tason geometristen primitiivien osille.

Mallintamisessa voidaan siis hyödyntää verteksejä muodostamaan korkeamman tason primitiivejä. Verteksistä seuraavaksi korkeamman tasoinen primitiivi on reuna. Reuna on kahden verteksin välille muodostettu yksiulotteinen vektori- tarkoittaen sitä, että vaikka reunalla ei ole

muita ulottuvuuksia, sillä on mitattava pituus. Tämä reunavektori määritellään kahden verteksin erotuksena siten, että vektorin pituus vastaa matkaa ensimmäisen ja jälkimmäisen verteksin välillä. (Puhakka 2008, 31–33.)

Seuraavaksi primitiivien hierarkkisessa järjestyksessä on taso. Taso määritetään kolmiulotteisen avaruuden halkaisevana 2-ulotteisena pintana. Tasolla on siis mitattava korkeus ja leveys, muttei paksuutta. On kuitenkin huomioonotettavaa, että yksinään tason korkeus ja leveys ovat äärettömät, kattaen äärettömän kokoisen pinta-alan. (Puhakka 2008, 41.) On kuitenkin mahdollista rajata tason pinta-alaa useasta reunasta muodostetulla suljetulla murtoviivalla rajallisen kokoiseksi alueeksi muodostaen rajallisen monikulmion. (Puhakka 2008, 40; Puhakka 2008 46–47). Tässä kontekstissa 3D-mallintamisessa yleistyneet termit ovat reunoilla rajatulle tasolle tahko ja monikulmiolle polygoni. Polygonilla on sisältämänsä tahkon tasoon suhteessa kohtisuora normaalivektori, joka määrittää sen, kumpi puoli polygonista on polygonin etupuoli (Puhakka 2008, 41–42; Puhakka 2008). Polygonien takapuoli tyypillisesti jätetään renderöimättä tehden polygonien takapuolesta läpinäkyvän, ellei toisin määritelty.

Kun useasta polygonista on muodostettu 3-ulotteisen kappale, siitä käytetty termi on polygonimalli.




Kuvio 1 Primitiivit ja Eulerin kulmat Blenderissä.

Polygonilla voi olla kuinka monta kulmaa tahansa, mutta yleisimmin 3D-mallintaessa hyödynnetään nelikulmion- (quad) ja kolmion (tri) muotoisia polygoneja. Tämä käytäntö johtuu siitä, että ko. muodot ovat usein paras tapa muodostaa monimutkaisempia kappaleita sekä siitä, että moni eri mallinnusohjelmista löytyvä 3D-mallintamisen työkalu toimii parhaiten näiden muotojen kanssa. (Turbosquid 2022).

Tämän lisäksi 3D-mallia renderöidessä algoritmit tyypillisesti yksinkertaistavat kuvaa generoidessa polygonit kolmioiksi halkaisemalla mallin polygonit, kunnes tämä tavoite on saavutettu. Neliöiden ja kolmioiden renderöity loppu-ulkonäkö on täten helpompi ennustaa ja hallita kuin suurempikulmaisten polygonien, sillä jo olemassa olevia kolmioita ei tarvitse jakaa ja neliöt muodostavat yhdellä halkaisulla kaksi siistiä kolmiota ilman suurempia muodon vääristymiä.

## 4.2 3D-Mallinnus osana pelikehitystä

Kolmannen ulottuvuuden tuominen videopeleihin fundamentaalisesti laajentaa mahdollisten pelikokemusten kirjon mittakaavaa huomattavasti verrattuna vastaavanlaisiin kokemuksiin, jotka ovat toteutettavissa puhtaasti kaksiulotteisia pelimekanismeja- ja käytäntöjä hyödyntäen. 1993 Doom loi tien edelläkävijänä 3D-grafiikan hyödyntämiseen videopelien kontekstissa ja 1990 edetessä eri tietokoneiden valmistajat alkoivat panostaa graafiseen suorituskykyyn entistä enemmän kiihtyen voimakkaasti ajan kuluessa (Puhakka 2008, 27). 2000-luvun paikkeilla 3D-pelejä julkaistiin jo laajasti sekä tietokoneelle, sekä usean eri valmistajan konsoleille.

Nykymaailmassa 3D-mallinnukseen liittyvän osaamisen saralla kehitys kiihtyykin edelleen jatkuvasti. Erityisesti internetin 2000-luvun jälkeinen räjähdysmäinen kasvu on mahdollistanut tietotaidon tehokkaan välityksen ihmisiltä toisille. 3D-mallintamiseen sekä pelikehitykseen liittyvä verkkopohjainen sisältö onkin nykypäivänä hyvin laaja-alaista ja kattavasti dokumentoitua läpi internetin, luoden hyvät mahdollisuudet jo harrastelijatason 3D-mallintajille kehittää osaamistaan.

Kasvaneen yleisen osaamistason ja teknologioiden kehityksen myötä, 3D-Mallinnusta hyödynnetään siis vuosi vuodelta yhä enemmän myös videopelien saralla. Nykypäivänä erinäiset AAA-videopelistudiot työllistävätkin laaja-alaisesti erinäisten osaamisalueiden digitaalisen ympäristön 3D-artisteja. Saatavilla olevat resurssit mahdollistavat jopa 3D-mallinnuksen hyödyntämisen pelinkehitysprosessissa soolopelinkehittäjänä.

# 5 Pelihahmon suunnittelu

## 5.1 Pelihahmo pelin kontekstissa

Pelihahmo täyttää pelin kontekstissa jonkinlaisen olennaisen funktion, joka on pelikokemuksen ytimessä. Hahmo siis toimii eräänlaisena työkaluna tai esteenä pelin tavoitteisiin suhteutettuna.

Pelaajahahmo esimerkkinä toimii pelaajan edustajan pelin kontekstissa, sekä työkaluna pelin järjestelmien kanssa suoritettavien interaktioiden toteuttamiseen vaikuttaen pelin ydinkokemusta muodostavaan kokonaisuuteen voimakkaasti. Muut pelihahmot voivat taasen mahdollistaa pelille kyvyn aktivoida pelaajaa suorittamaan interaktioita pelin kanssa joko aktiivisesti haastaen- tai edesauttaen pelaajaa pelin tavoitteissa.

Tämä kontrastoituu voimakkaasti ns. passiivisten interaktioiden kanssa, joita pelaaja voi suorittaa suhteessa pelin ympäristöön. Pelihahmoja voidaan myös tarvittaessa hyödyntää lavasteina luomaan eloisuutta pelin ympäristöön ja vahvistamaan pelin tunnelmaa, vaikka hahmot eivät aktiivisesti osallistuisikaan pelin toimintaan.

Pelihahmon onnistunut suunnittelu suhteutettuna käyttökontekstiinsa onkin siis olennainen osa pelihahmon tuotantoa.

## 5.2 Konseptointivaihe / GDD

Pelin suunnitteluprosessi on jokseenkin verrattavissa palvelumuotoilusta tuttuun tuotesuunnittelun prosessiin. Ennen pelin tuotantovaihetta pelille luodaan pohjaksi kattava GDD hyödyntäen monia samoja periaatteita. GDD, eli Game Design Document on pelikehityksessä käytettävä termi suunnitelmalle, jonka tavoitteena on kattaa lopputuotteena muodostuvan pelin elementit. GDD:n yleinen sisälto voi kuitenkin kontekstin perusteella vaihdella suuresti sen sisällä määriteltyjen linjausten suhteen siinä, miten ja kuinka tarkasti eri linjaukset pelille ovat määritelty. GDD:n suunnittelu- ja ylläpitoprosessiin onkin useita eri ratkaisuja ja kuten yleisessä ohjelmistokehityksessä. (French 2022.)

Pelihahmo nimensä mukaisesti esiintyy vastaavanlaisen pelin kontekstissa, jolloin senkin suunnitteluprosessi tulisi suorittaa GDD:n asettamia linjauksia, vaatimuksia ja tavoitteita tulkiten ja tukien. Täten GDD:tä tehdessä on tärkeää asettaa tavoitelinjaukset myös hahmojen funktionaalisille-, teknisille-, sekä visuaalisille ominaisuuksille.

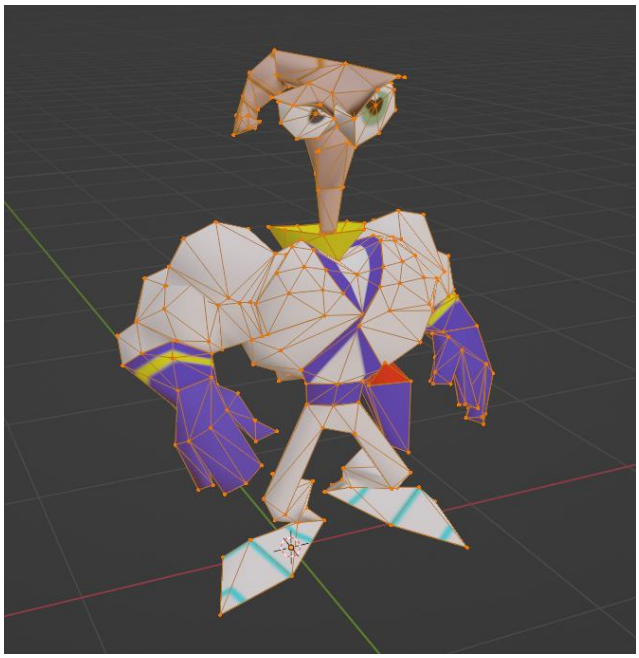
Tässä kontekstissa prosessin lopputuotteena kokonaista peliä ei kuitenkaan muodostu, joten koen, että kokonaista GDD:tä ei tässä kontekstissa ole mielekästä tehdä. Täten voidaan tiivistää GDD:n elementit projektin kontekstissa siten, että keskitytään erityisesti 3D-

mallinnuksen tavoitteille ja yleiselle visuaaliselle ilmeelle relevantteihin kysymyksiin hyödyntäen jo olemassa olevia pelejä inspiraationa.

### 5.2.1 Hahmon pelin genre ja hahmon funktio sen kontekstissa

Pelin genre, jossa hahmo esiintyy vaikuttaa suunnitteluprosessiin olennaisesti. Pelin genre saattaa asettaa vaatimuksia tai linjauksia, jotka on otettava huomioon 3D-mallinnuksen alkuvaiheesta lähtien, jotta hahmo täyttää sen käyttökontekstin funktionaaliset ja visuaaliset tarpeet.

Tasohyppelyssä voidaan esimerkiksi olettaa, että pelaajahahmolla tulisi olla funktionaalisesti kykeneväisyys hypyn suorittamiseen, mikäli tarkoituksena ei ole toteuttaa pelin genren kumoaminen pelimekaanikkojen kautta. Tällaisessa kontekstissa kastemato saattaisi olla huono vaihtoehto pelaajahahmolle, sillä kastemato ei kykene luonnollisesti hyppäämään. Tämä voidaan kuitenkin kiertää rationalisoimalla tämä kyvykkyys jollakin hahmolle suunnitellulla ominaisuudella, joka tarkoituksena on kiertää tämä funktionaalinen ongelma. Kastemato, jolla on ulkoisesti ihmismäiseen muotoon muototutuva puku voisi täten olla tapa, jolla voidaan rationalisoida hahmon kyky hyppäämiseen pelimaailmallisena funktiona.



Kuvio  Earthworm Jim pelistä Earthworm Jim 3D (1999) avattuna Blender-sovelluksessa (mukaillen The Models Resource 2022).

Toisena funktionaalisena esimerkkinä pelissä, jossa pelaaja näkee pelaajahahmon hahmon silmien näkökuvakulmasta, ei välttämättä ole tarkoituksenmukaista mallintaa kokonaista hahmoa. Tässä kontekstissa kuvakulman rajauksen takia on usein mielekkäämpää mallintaa pelkästään pelaajahahmon kädet, jos voidaan olettaa, että pelaajahahmolla ei tulla näkemään

toisesta kuvakulmasta- rikkoen illuusion. Tämä onkin se metodi, jota useimmat ensimmäisen persoonan kuvakulmaan kamerakulman rajaavat pelit hyödyntävät.

Syventymättä enempää kaikkien mahdollisten genrejen asettamiin mahdollisiin vaatimuksiin, rajaan tässä opinnäytetyössä toteutettavan projektin viitekehyksen genren suhteen jo aikaisemmin esimerkkinä hyödynnettyyn 3D-tasohyppelyyn, sillä uskon sen antavan hyvät mahdollisuudet demonstroida prosessia.

### 5.2.2 Visuaalinen ilme

3D-mallinnuksessa, kuten muissakin luovissa prosesseissa on mahdollista saada aikaiseksi voimakkaasti eroavia lopputuotteita. Täten on usein kannattavaa asettaa selkeä tavoite halutulle visuaaliselle ilmeelle jo suunnittelun alkuvaiheessa. Tällä voidaan suunnata tuotantoprosessia mielekkäästi selkeää tavoitetta kohden nopeuttaen prosessia huomattavasti suhteessa ns. ”tavoitteen löytämiseen matkan aikana”. Pienikokoisella tuotantotiimillä- tai sooloprojektilla tämä on erityisen tärkeää, sillä ihmisresurssien määrän takia korostuu erityisesti tarve tehdä merkityksellistä edistystä kohti tavoitetta suhteessa tehtyyn työmäärään.

Tästä esimerkkinä pelinkehityksen Indie-puolella suosioon on noussut vanhemman aikakauden pelikonsolien teknisten rajoitusten keinotekoinen jäljittely visuaalisen ilmeen suuntauksessa. Tämä johtuu nostalgian lisäksi siitä, että vanhempien konsolien visuaalinen taso on nykypäivän harrastelijalle helppo saavuttaa ilman suurikokoista tiimiä tai ammattilaistason erikoiskoulusta uusimmista teknologioista.

Toisin sanoen, tarkoituksenmukainen keinotekoisten rajoitteiden asettaminen 3D-mallien yksityiskohtaisuuden tasolle on tehokas tapa saavuttaa sekä tuotantotehokas tuotantoprosessi sekä visuaalisesti miellyttävä lopputulos. Tämän takia tässä opinnäytetyössä toteutettavan projektin visuaalinen ilme tulee jäljittelemään 2000-luvun vaihteen konsolien 3D-mallien teknologisia rajoituksia visuaalisessa ilmeessään.

Ottaen huomioon aikaisemmin asetettu suuntauslinjausten viitekehys genrelle ja visuaaliselle ilmeelle, onkin siis mielekästä tarkastella ilmeeltään tavoitetta vastaavia aikakaudella menestyneitä pelejä inspiraation lähteinä. 3D-tasohyppelyjen saralla menestyneitä pelisarjoja aikakaudella olivat mm. Crash Bandicoot ja Spyro the Dragon. Vaikka näiden sarjojen käytännön toteutus eroaakin toisistaan huomattavasti funktionaalisuuden suhteen, ovat ko. pelit hyviä esimerkkejä siitä, miltä aikakaudella toteutetut 3D-mallinnetut pelihahmot näyttivät.

Kuvio 1 Crash Bandicoot pelistä Crash Bandicoot 2: Cortex Strikes Back (oik.) (1997) ja Spyro the Dragon (vas.) pelistä Spyro 2: Gateway to Glimmer (1999) Blender-sovelluksessa (mukaillen The Models Resource 2022).

| Kuvion 3 3D-mallien ominaisuudet | Crash Bandicoot | Spyro the Dragon |
|---|---|---|
| Verteksejä | 344 | 230 |
| Reunoja | 909 | 631 |
| Tahkoja | 568 | 411 |

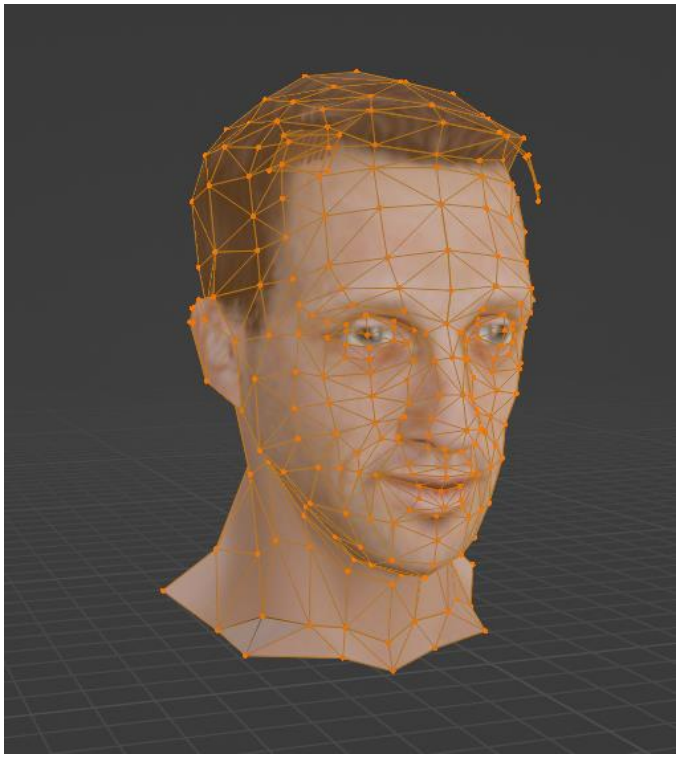
Taulukko 1 Kuvion 2 pelaajahahmojen 3D-mallien tekniset ominaisuudet (tiedot Blenderistä)

Tarkastelemalla näitä esimerkkejä voidaan muodostaa karkeita arvioita siitä, mille tasolle tavoitelopputuloksena olevan 3D-mallin tekniset ominaisuudet tulisi mallintaa rikkomatta asetettuja teknisiä rajoitteita. On huomioonotettavaa, että näiden mallien hyödyntämien verteksien lukumäärä on huomattavan pieni verrattuna nykyaikaisiin peleihin, sillä aikakauden konsolien suorituskyky rajoitti voimakkaasti peleissä hyödynnettyjen 3D-mallien monimukaisuuden tasoa.

Tarkastelemalla suurempaa varhaisen 2000-luvun pelien viitekehystä voidaan huomata, että teknisten rajoitusten takia hahmojen visuaalinen ulkomuoto oli usein suunniteltu

hyödyntämään rajoitettu verteksien määrä mahdollisimman tehokkaasti. Aikakauden tekniset rajoitukset estivät aidon fotorealismin, jolloin ihmishahmojen mallintaminen vaati tarkkaa verteksi-budjetin priorisointia kuvamaan tärkeimmät muodot. Tästä huolimatta ihmishahmojen ulkomuoto jäi kuitenkin usein kauas uskottavuudesta.

Aito realismi ei ollut yksinkertaisesti saavutettavissa aikakauden teknisten rajoitusten puitteissa ja menettelevä lopputulos usein vaati suurta määrä verteksejä erityisesti aikakauden mittapuulla uskottavien ihmiskasvojen mallintamiseen. Tämän lisäksi jo yhden 3D-mallin verteksien viitekehystä suurempi lukumäärä vei konsolilta huomattavasti enemmän suorituskykyä vaatien korkeampaa optimisoinnin tasoa muilta pelissä hyödynnettäviltä elementeiltä, jotta peli ei kärsisi suorituskykyongelmista. Tämä usein johti siihen, että pelihahmojen määrässä tai ympäristön yksityiskohtaisuuden tasossa jouduttiin kompensoimaan monimutkaisempien pelihahmojen viemiä resursseja.



Kuvio 2 Pelkkä Tony Hawkin pää käyttää yhteensä 424 verteksiä Tony Hawk’s Pro Skater 4 (2003) -pelissä (mukaillen The Models Resource 2022, tiedot Blenderistä).

Aikakauden pelien suositussa suunnittelufilosofiassa tämä usein tarkoitti sitä, että pelihahmoissa käytettiin sarjakuvamaista muotokieltä ja hahmokonsepteja, jotka pystyivät hyödyntämään rajoitetun määrän verteksejä tehokkaasti kärsimättä huomattavasti ulkomuodossaan. Hyödyntämällä dramaattisia ja voimakkaita muotoja sekä värejä helposti tunnistettavien ja visuaalisesti kiinnostavien hahmojen tuotanto usein onnistuikin huomattavasti matalammalla verteksien määrällä kuin tavoitellessa realismia.

## 5.3 Hahmon konsepti

Seuraten esittäminäni esimerkkejä lähdin suunnittelemaan konseptia hahmolle perustuen eläimeen tai olentoon, joka olisi helposti tunnistettava ja joka hypoteettisesti olisi voinut esiintyä pelissä 2000-luvun vaihteessa.

Tutkittuani olemassa olevia eläimiä välttäen niitä vaihtoehtoja, jotka olivat jo voimakkaasti edustettuja, päädyin tähtikuonokontiaiseen sen ikonisen kuonon takia. Erityisesti kontiaisen ikoninen kuono vaikutti toimivalta tavalta luoda selkeä visuaalinen kiintopiste suunnitellulle hahmolle sekä saada se erottumaan jo olemassa olevista hahmoista.

Tein tältä pohjalta nopean konseptipiirrustuksen, joka ilmaisisi idean konkreettisemmin. Tässä vaiheessa tulisi esittää selkeästi hahmon tärkeimmät yksityiskohdat selkeästi, jotta hahmon mallintamisesta vastaava kykenee jäljentämään halutun ulkomuodon joutumatta arvaamaan yksityiskohtia. Tätä varten yleisimmin konseptipiirrustuksia tehdään useita kappaleita eri kuvakulmista esittäen lisätarkennuksia eri hahmon muodostavista elementeistä. Tässä tapauksessa kyseessä on kuitenkin sangen yksinkertainen hahmo, jonka tulen itse mallintamaan, jolloin pelkkä karkea piirros yhdestä kuvakulmasta riittää hyvin kuvaamaan hahmon olemusta.

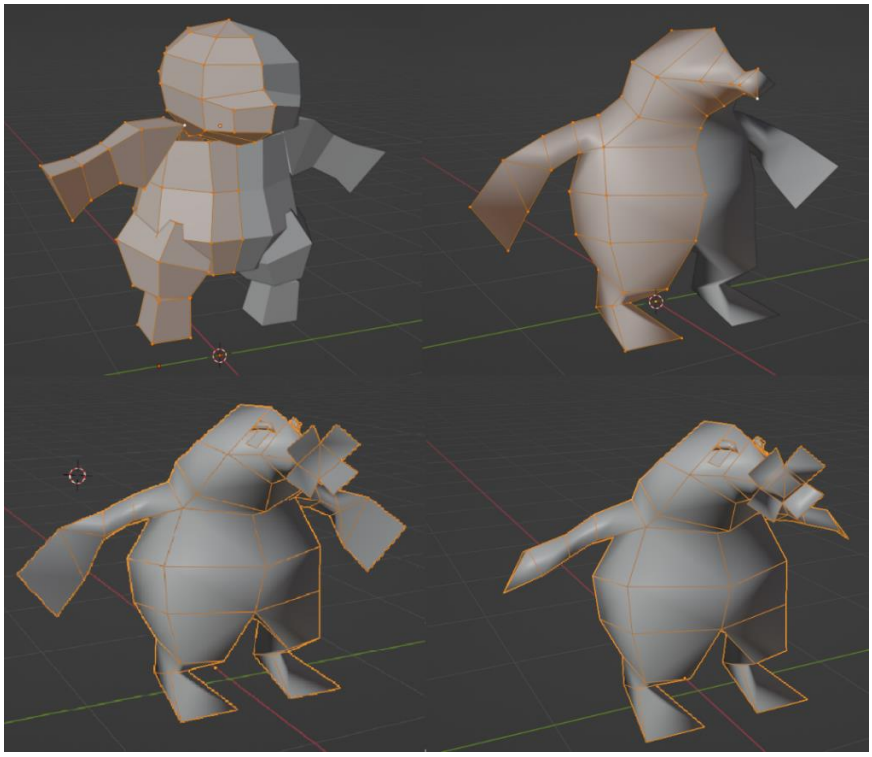
Kuvio 3 Yksinkertainen konseptipiirros tähtikuonokontiaiseen pohjautuvalle hahmolle.

# 6 Hahmon polygonimallin mallinnus

Mallintaessa on usein hyödyllistä muodostaa ensiksi karkea malli yksinkertaisista muodoista- ja lähteä jalostamaan sitä kohti lopullista tavoitetta, jotta hahmon mittasuhteen ovat konkreettisesti visualisoitu ennen yksityiskohtia. 3D-Mallintaessa voidaan parsia yhteen useita eri geometrisiä kappaleita myöhemmin prosessissa sekä jalostaa näitä muotoja mielivaltaisesti, joten tässä vaiheessa ei ole vielä tarvetta erityiselle tarkkuudelle. Tämän pelihahmon kaltaisessa kontekstissa, missä symmetrisyys on olennaista, voidaan myös asettaa ohjelmistossa päälle toiminnallisuus, joka automaattisesti peilaa mallin halutulla akselilla.

Manipuloimalla, yhdistämällä, ja luomalla verteksejä voidaan manipuloida ja jalostaa mallinnettavaa kappaletta halutulla tavalla eteenpäin. 3D-mallintamisen ohjelmistot tarjoavatkin tähän tarkoitukseen laajan kirjon eri työkaluja- liian laajan ruvetakseni listaamaan niitä tässä. Olennaisesti lähes kaikkien työkalujen funktiot voidaan kuitenkin yksinkertaistaa edellä mainittujen operaatioiden suorittamiseen usealle verteksille samanaikaisesti tiettyjen parametrien sisällä. Esimerkiksi loop selection -työkalulla voidaan valita kerralla kaikki tietyllä kappaleen kiertävällä janalla olevat verteksit ja esim. scale -funktiolla voidaan kasvattaa tai pienentää valittujen verteksien etäisyyksiä toisistaan.

Hyödyntämällä tarjolla olevia työkaluja voidaankin jalostaa työn alla olevan 3D-mallin muotokieltä haluttuun lopputulokseen verrattain helposti, mikäli valitsemamme ohjelman käyttöliittymä on hallussa.



Kuvio 4 Tähtikuonokontiais-hahmon mallintamisen prosessikuvat Blenderissä.

Prosessikuvista kuviossa 5 voidaan huomata, että hahmo on aseteltu seisomaan suorassa, käsien leväten kohti sivuja. Tämä asento on olennainen hahmon tulevaa animointia varten. Kun luodaan 3D-mallille animaatioita, mitä kauempana uusi asento on siitä, missä asennossa 3D-malli on mallinnettu, sitä enemmän 3D-malli vääristyy asennon muutoksen seurauksena. Tämä on erityisen huomattavaa 3D-malleilla, joilla on matala verteksien lukumäärä, sillä vääristymä jakautuu huomattavasti voimakkaammin yksittäisille vertekseille. Yleisimmin animaatiota vaativat hahmot mallinnetaan seisoen jalat suorana kädet joko osoittaen suoraan sivulle, tai pienessä kulmassa alaspäin kämmenet kohti maata. Näitä asentoja kutsutaan yleisenä käytäntönä vastaavasti T-, ja A-asennoiksi käsien asennon mukaan. Hahmon asettelu vastaavanlaisesti minimoi hahmon asennon muutoksesta johtuvat vääristymät siten, että mikään realistinen hahmon asento ei ole huomattavasti toistaan kauempana oletusasennosta suhteessa toisiinsa.

| Ominaisuudet | Tähtikuonokontiais-hahmo |
|---|---|
| Verteksejä | 126 |
| Reunoja | 242 |
| Tahkoja | 124 |

Taulukko 2 Tähtikuonokontiais-hahmon lopulliset tekniset ominaisuudet.

Tarkastellessa polygonimallin teknisten ominaisuuksien lopullisia tilastoja voidaan todeta, että mallin hyödyntämä verteksien- ja täten reunojen ja tahkojen määrä pysyi huomattavasti alle tavoiterajan helposti pysyen 2000-luvun vaihteen teknisten rajoitusten sisällä. Halutessani voisin lisätä mallille lisää verteksejä, mikäli haluaisin vielä tässä vaiheessa jalostaa mallia entisestään. Koen tämän kuitenkin tarpeettomaksi, sillä malli on jo tässä vaiheessa esteettisesti tyydyttävä.

# 7 Teksturointi

3D-mallintamisessa teksturoinnilla viitataan erinäisten kuvien, värien, ja pintojen piirtämiseen 3D-mallin pinnalle. Teksturointia siis hyödynnetäänkin 3D-mallien elävöittämiseen nostamalla ne pois yksivärisyydestä (Puhakka 2022, 206).

Ymmärrettävästi teksturointi onkin olennainen mallintamisprosessin vaihe, sillä väri usein vaikuttaa voimakkaasti 3D-mallinnetun kappaleen- eli tässä tapauksessa hahmon yleiseen visuaaliseen vaikutelmaan.

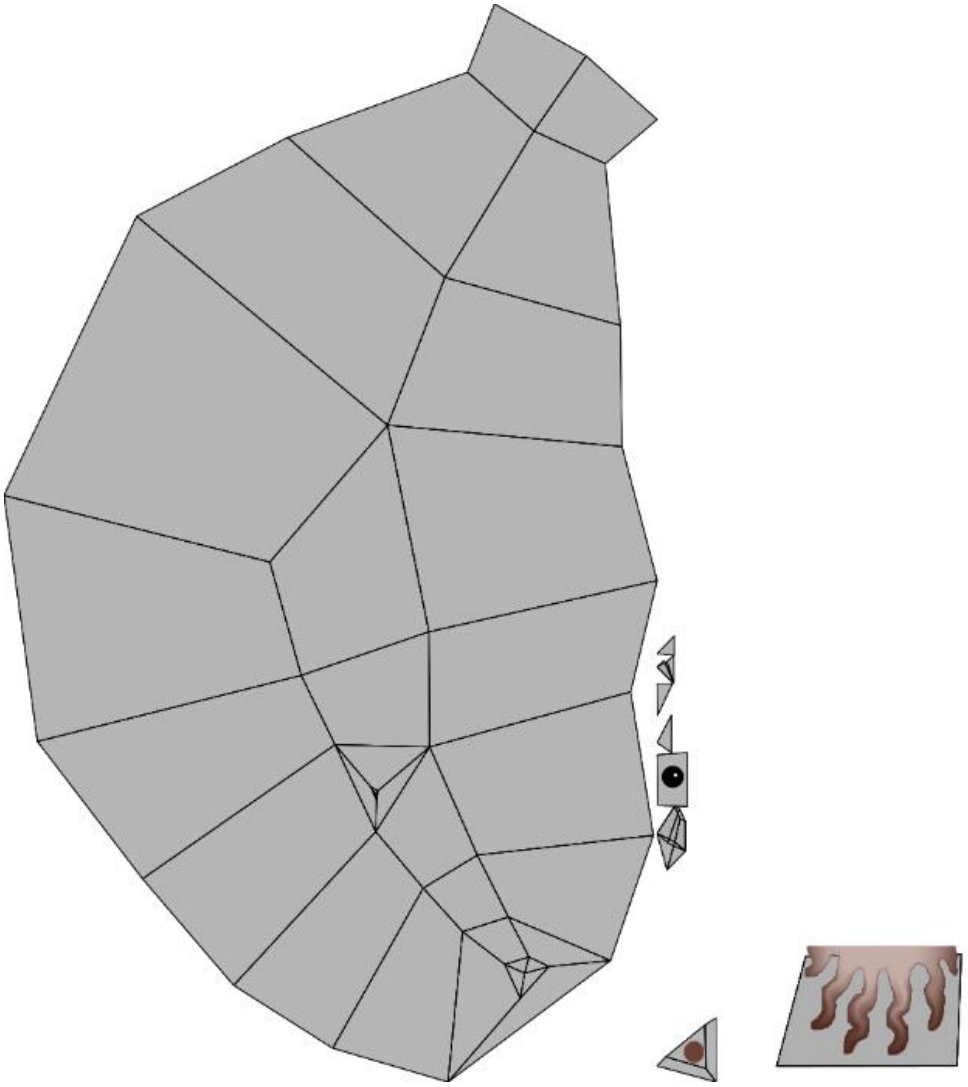
Välillä mallintaessa saattaa myös olla mielekkäämpää luoda osa mallin toivotuista yksityiskohdista 2-ulotteisina kuvina ja projisoida ne tekstuureina mallin pintaan, mikäli ko. yksityiskohtien mallintaminen vaatisi liian monta verteksiä ollakseen kustannustehokasta. Voidaankin siis hyödyntää teksturointia sekä vahvistamaan ja muokkaamaan mallin tunnelmaa, että lisäämään mallin yksityiskohtaisuutta kustannustehokkaalla tavalla suhteessa lisägeometriaan.

## 7.1 Kuvatekstuurit

Termi tekstuuri tulee suoraan sen yleisesti käytössä olevasta merkityksestä pintarakenteena. Tekstuureilla onkin alun perin pyritty luomaan vaikutelma hienorakenteisesta pinnasta muutoin litteillä 3D-mallin pinnoilla (Puhakka 2022, 206).

Teksturoinnin kontekstissa tekstuurilla usein viitataan nimenomaan kuvatekstuureihin, jotka projisoidaan kappaleen pinnalle kartoittamalla tekstuurina hyödynnettävien kuvat vastaamaan sijainteja 3D-mallin pinnalla olevilla tahkoilla. Tämä suoritetaan litistämällä mallin pinta origamimaisesti litteäksi 2-ulotteiseksi tasorepresentaatioksi- UV-kartaksi, jotta 2-uloitteisten kuvien kartoittaminen onnistuisi pinnalle hyödyntäen UV-kartan tekstuurikoordinaatteja. (Puhakka 2022, 206-209.) 3-ulotteisilta kappaleilta tämä vaatii saumojen määrittämistä 3D-mallille kappaleen reunoja mukaillen- saumoilla tarkoittaen kohtia, joista mallin reunat voidaan ”avata” muodostaen reiän. Tämä on tarpeellista, sillä umpinainen kappale ei litisty 2-ulotteiselle pinnalle. Saumoja voidaan myös tarpeen mukaan lisätä litistymisestä aiheutuneiden väärentymien lieventämiseksi.

Mikäli halutaan tekstuurin olevan joltain elementiltään symmetrinen usean eri mallin kohdan välillä, voidaan asettaa UV-kartalla mallin osia päällekkäin, jolloin molempiin samaan kohtaan kartoitettuihin mallin osiin tulee tekstuurikoordinaatti ja täten sama kuvatekstuuri. Tätä tekstuurien peilaamista voidaan hyödyntää asettamalla sauma keskelle hahmon keskilinjaa ja asettamalla puoliskojen kartoitukset toistensa päälle UV-kartalla. Mikäli sille on tarve, voidaankin siis hyödyntää kuvatekstuurin yhtä osaa mielivaltaisella määrällä eri mallin osa-alueita.



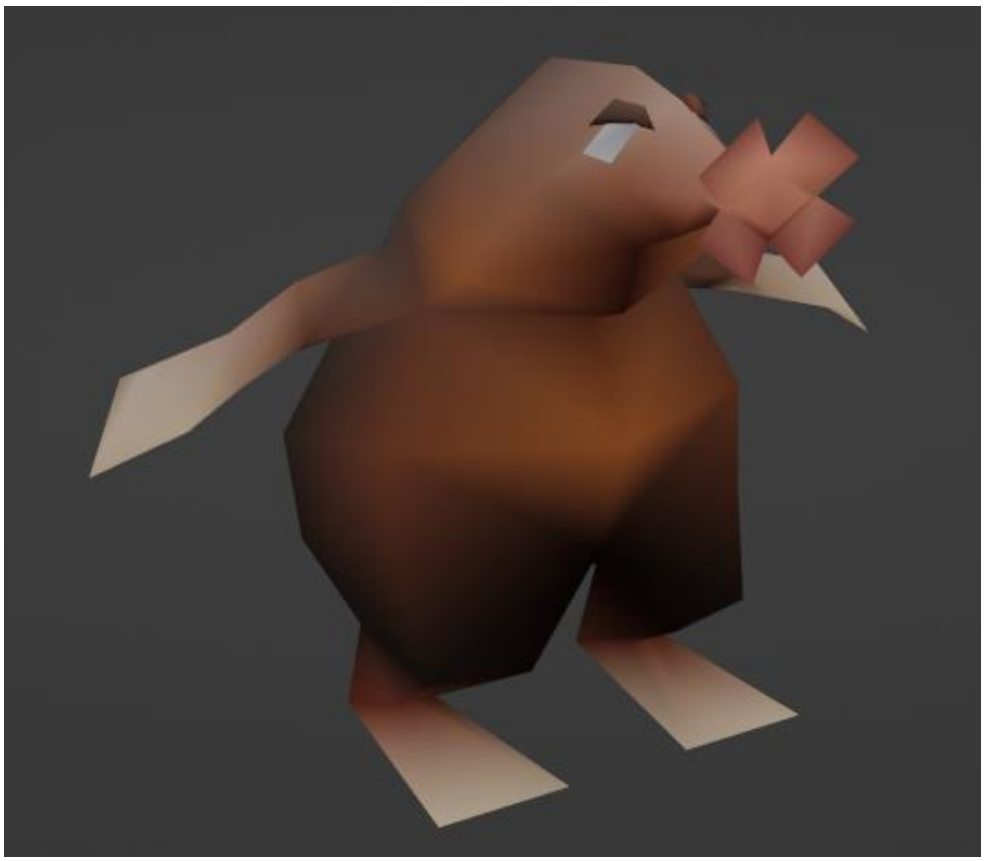
Kuvio 5 Tähtikuonokontiais-hahmon hyödyntämät kuvatekstuurit UV-kartalle aseteltuina.

## 7.2 Verteksivärit

Verteksiväritys on tapa luoda yksinkertaisia värejä- ja värimuutoksia mallin pintaan ilman kuvatiedostojen hyödyntämistä. Tämä suoritetaan antamalla mallin vertekseille sijainnin lisäksi myös haluttu väriarvo, joka ohjaa tahkon pinnan värimuutosta matemaattisesti. Mitä lähempänä tahkon pinta on verteksiä, sitä enemmän pinnan väri siirtyy kohti verteksin väriarvoa

liukuvärinä. Koska tätä muutosta ajaa matemaattiset funktiot- tarkoittaa tämä sitä, että kappaleen koosta tai näytön resoluutiosta riippumatta värien muutos skaalautuu aina ilman vääristymiä. Tätä väritystapaa kutsutaan myös nimellä Gouraud-sävytys. (Puhakka 2008, 204–205.)

Verteksiväritys on kuvatiedostojen tekstuureina hyödyntämiseen verrattuna hyvin kustannustehokasta, sillä tekstuurin projisointi kappaleen pintaan vie huomattavasti enemmän matematiikkaa suhteessa verteksiväreihin. Koska verteksivärityksessä väriarvoa ohjaavat verteksit, kappaleen geometria kuitenkin saattaa rajoittaa tällä metodilla aikaansaatujen lopputulosten yksityiskohtaisuutta erityisesti matalilla verteksien lukumäärillä.



Kuvio 6 Tähtikuonokontiais-hahmo pelkkiä verteksivärejä hyödyntäen.

## 7.3 Hahmon teksturointi

Teksturoinnissa yksi 3D-mallin on mahdollista hyödyntää sekä kuvatekstuureja, että verteksivärejä. Usein 3D-malli saattaakin hyödyntää sekä tekstuureja, että verteksiväritystä eri mallin osa-alueilla tarpeiden mukaan. Tapoja yhdistää nämä kaksi tekniikkaa onkin useita. (Puhakka 2008, 214–215.)

Vaikka verteksivärien suosio teksturoinnissa on ajan saatossa laskenut konsolien suorituskyvyn noustessa, 2000-luvun vaihteessa suuri osa pelihahmoista hyödynsikin sekä verteksiväritystä, että kuvatekstuureja teksturoinnissa. Tämän tähtikuonokontiais-hahmon teksturointi vastaa pitkälti samaa kaavaa. Tähtikuonokontiais-hahmon päävärit ovat asetettu verteksivärejä hyödyntäen ja hahmon silmien sekä kuonon yksityiskohdat hyödyntävät tekstuureja antaen ko. aikakautta uskollisesti jäljittelevän vaikutelman.

Kuvio 7 Teksturoitu tähtikuonokontiais-hahmon 3D-malli Blenderissä.

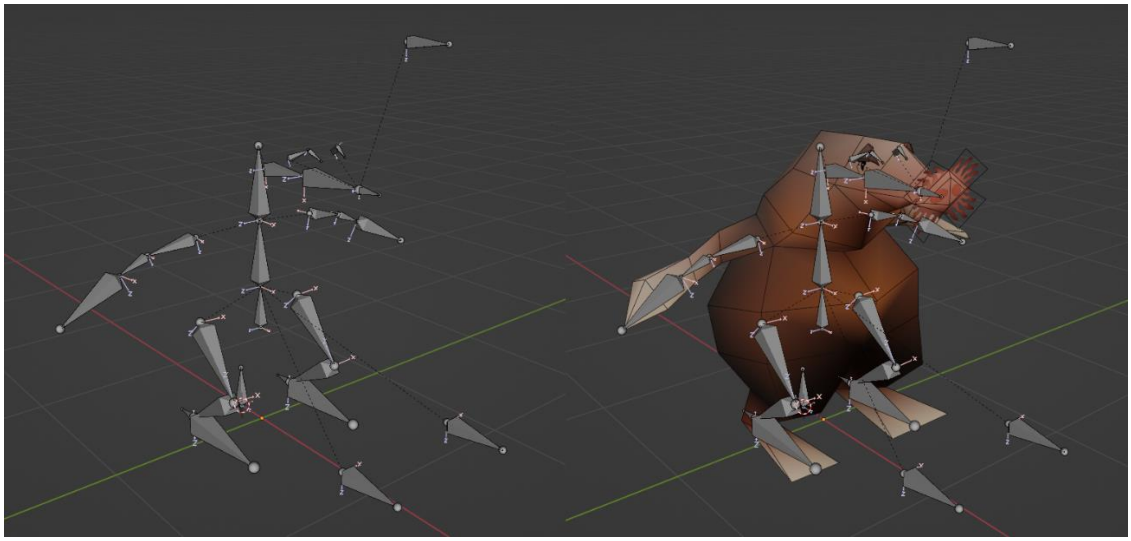
# 8 Animaatio

## 8.1 Rigaus

3D-mallintamisessa rigaus on välivaihe, jossa 3D-malli valmistellaan animaatiota varten. Rigauksessa 3D-mallille luodaan rig: eräänlainen luurankomalli, joka ohjaa 3D-mallin liikkeitä samankaltaisesti kuin perinteisessä stop motion -animaatioissa hyödynnettävä armatuuri. Rig koostuu useasta eri luusta, jotka asetetaan hierarkkisesti ohjaamaan tiettyjä mallin osia niitä liikuttaessa interpoloiden luiden asentojen välillä. (Puhakka 2008, 431)

3D-mallin rigauksen voi suorittaa monella eri tapaa mallintajan mieltymysten mukaan, mutta ydintasolla luiden ominaisfunktiot voidaan yksinkertaistaa samankaltaisiksi kuin mallintamisen työkalujen: luut liikuttavat ja muokkaavat tiettyjen verteksien sijaintia, orientaatiota, ja suhteita muihin vertekseihin mallintajan asettamien parametrien sisällä. Luut kuitenkin poikkeavat näistä muokkaustyökaluista siten, että ne kykenevät palauttamaan mallin näiden muutosten jälkeen takaisin alkuasentoon sekä muistavat oman orientaationsa ja sijaintinsa suhteessa siihen. Luiden vastaavan kaltaiset muutokset voidaan interpoloida tapahtumaan ajallisesti halutulla aikavälillä samanaikaisesti muiden luiden kanssa, mahdollistaen monimutkaisten animaatioiden luomisen.

Riippuen rigin rakenteesta, voidaan myös hyödyntää käänteistä kinematiikkaa ohjaamaan osia rigin rakenteesta. Käänteisellä kinematiikalla voidaan asettaa yhteen kytkettyjä luita seuraamaan määrittellyn sarjan päässä olevan luun liikkeitä ilman, että joudutaan erikseen määrittelemään asento jokaiselle sarjan luulle. Tämän tähtikuonokontiais-hahmon kontekstissa hahmon jalat seuraavat käänteistä kinematiikkaa hahmon polvien taipuen automaattisesti suhteessa jalan asentoon. (Puhakka 2008, 433.) Suunta, johon polvi pyrkii kompressiotilanteessa taipumaan, on määritetty pole target -luilla, jotka ovat polvien edessä kaukana muusta luurakenteesta. Tämän lisäksi pään yläpuolella leijuva luu ohjaa silmien aukiolon asentoa (ks. Kuvio 9).



Kuvio 8 Tähtikuonokontiais-hahmon rig sekä ilman polygonimallia, että polygonimallin kanssa Blenderissä.

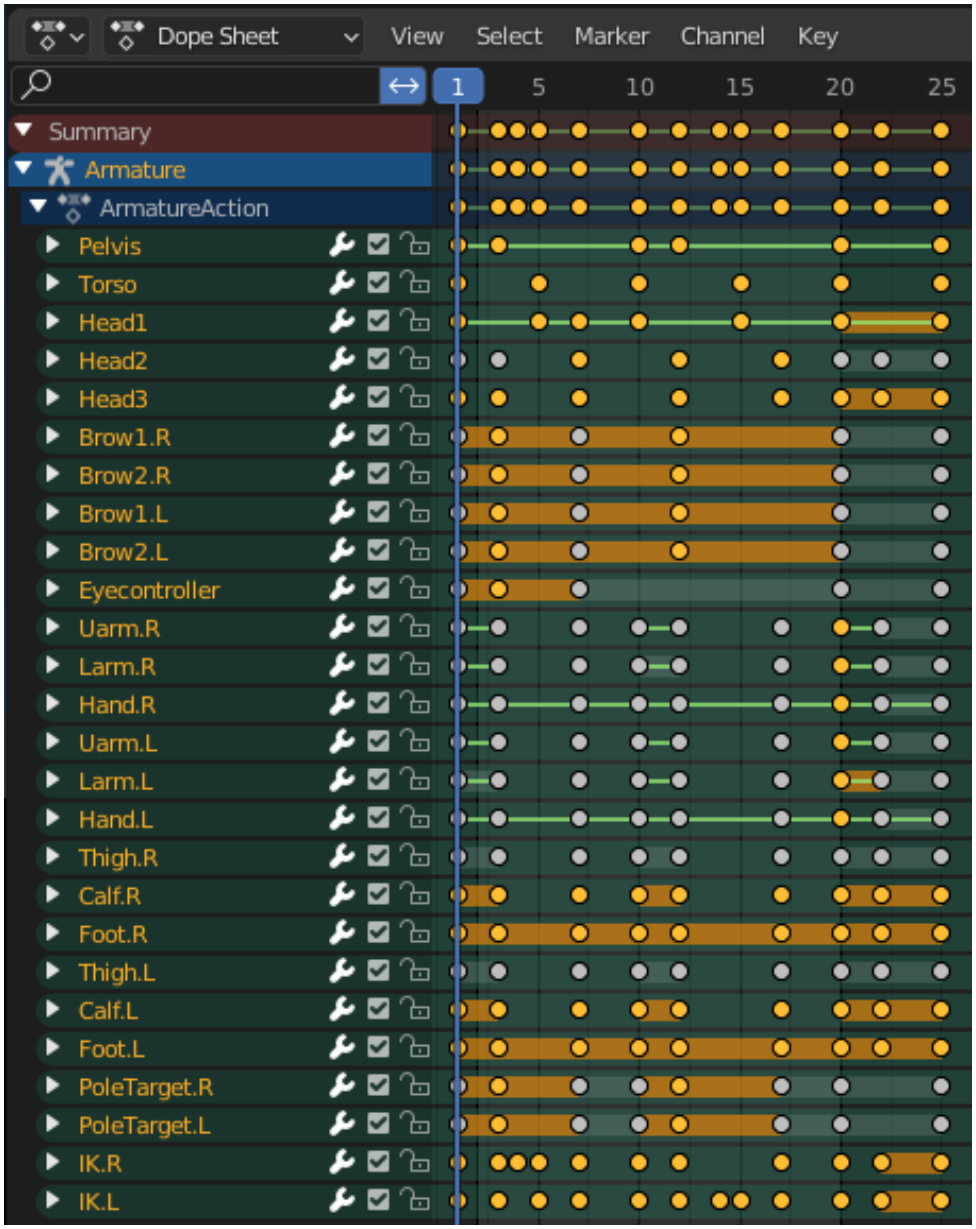
## 8.2 Animaatioiden toteutus

Animaatioita 3D-mallille luodessa luodaan rigiä hyödyntäen sarja asentoja luille, jotka toistetaan halutulla aikavälillä. Tämä vastaa tapaa, jolla perinteinen animaatio toimii. Työskentely tapahtuukin ns. avainpiirroksia- eräänlaisia tärkeimpiä liikeratoja kuvastavia asentoja hyödyntäen. Toisin kuin perinteisessä animaatiossa, prosessissa on kuitenkin mahdollisuus interpoloida asentojen välillä automaattisesti hyödyntäen tietokoneen prosessorin suorittamia matemaattisia laskelmia.

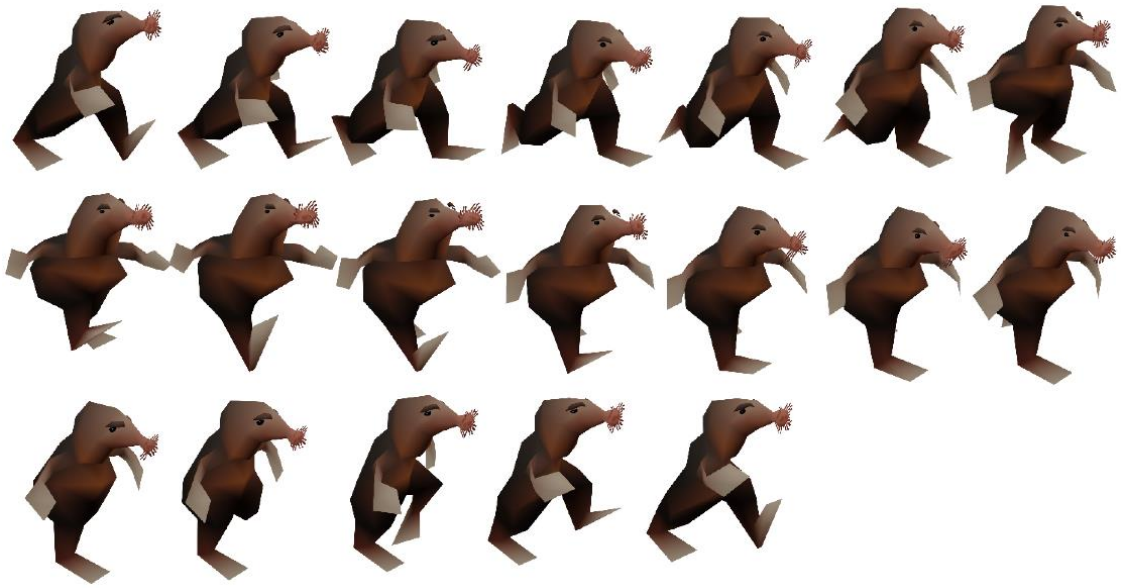
Täten on mahdollista säätää eri luiden nopeutta, kiihtyvyyttä, inertiaa liikkeessä, sekä sitä, kuinka tarkasti luut seuraavat luotuja asentoja. Näitä ominaisuuksia tarvittaessa voidaan hienosäätää jopa yksittäisluukohtaisesti. Rigin tarjoamien luiden rajoitteissa voidaankin siis luoda 3D-mallille lähes minkälaisia animaatioita halutaankaan manipuloimalla luiden asentoa ja niiden ajoitustaulukkoa. Mikäli haluttaisiin niin tehdä, on myös mahdollista interpoloida useita eri animaatioita keskenään, joten taivas on animaatioprosessin luovuuden rajana.

Sillä tämän tähtikuonokontiais-hahmon idea oli toimia 3D-tasohyppelyn kontekstissa, eräs olennaisimmista animaatioista, joka mallilla tulisi olla on toistuva kävelyanimaatio. Täten tämä on se animaatio, jonka loin hahmolle demonstroimaan tätä käytännössä.

Proseduraalinen animaatio on tässä projektissa hyödyntämätön, mutta silti mainitsemisen arvoinen teknologia 3D-mallien animaatioiden toteuttamiseen erityisesti videopelien kontekstissa. Proseduraalisessa animaatiossa ei luoda hahmolle asentoja, vaan määrittellään luiden animaatio hyödyntäen koodia pelin ajoaikaisesti.




Kuvio 9 Tähtikuonokontiais-hahmon kävelyanimaation ajoitustaulukko Blenderissä.



Kuvio 10 Tähtikuonokontiais-hahmon kävelyanimaatio kuvasarjana.

Tarkastelemalla kuvioita 10 ja 11, nähdään animaation ajoitustaulukko, sekä animaatio kuvasarjaksi renderöitynä. Ajoitustaulukkoa voidaan lukea siten, että vasemmalla nimetyn luun rivillä jokainen piste on asetettu avainasento, joiden välillä animaatio interpoloidaan. Animaation yksi kierros kestää kokonaisuudessaan noin 2/3 sekuntia kuvataajuudella 30 kuvaa sekunnissa. Voidaan myös huomata, että ajoitustaulukon ylälaidan numerot vastaavat tätä kuvataajuutta. Huomioonotettavaa on kuitenkin se, että animaatio ei ole pysyvästi sidonnainen tähän nopeuteen, vaan sen nopeus voidaan määrittää uudestaan mielivaltaisesti ilman, että joudutaan luomaan uusia avainpiirroksia.

Animaatio käytössä olevalla kuvataajuudella sisältää 25 kuvaa, joista hyödynnetään 19 kappaletta. Johtuen animaatiossa käytetyistä interpoloinnin metodeista, animaation ylimääräisiä käyttämättömiä kuvia hyödynnetään ohjaamaan animaatiota siten, että se jatkuu sulavasti animaation alussa olevaan asentoon uudestaan. Animaation saavuttaessa kuvan 19, se ohittaa loput taulukosta ja hyppää animaation alkuun, jotta animaatio toistuu tarpeen mukaan loputtomasti.

## 9 Hahmon implementaatio Unityssä

Kun hahmon mallintamisen prosessi on valmis, voidaan siirtyä sen implementoimiseen Unityssä. Blenderistä saa ulos mallin kaikkine olennaisine ominaisuuksineen Unityn hyväksymään muotoon tuomalla sen ulos Blenderistä .fbx-tiedostona. Tässä tapauksessa tämä tarkoittaa sitä, että tiedostoon pakataan polygonimallin lisäksi myös siihen kytketyn rigin, käyttämän teksturoinnin, sekä animaatioiden tiedot. Unity kykenee siten hyödyntämään näitä

ominaisuuksia ilman niiden erillistä tuontia ja määrittelyä Unityn sisällä. Tuomalla tämän .fbx-tiedoston Unity projektiin, voidaankin siis hyödyntää 3D-mallia sellaisena kun se oli Blenderissä ilman suuria toimenpiteitä.

Tarvittaessa voidaan myös vaihtaa 3D-mallin .fbx-tiedostoon pakattuja ominaisuuksia, mikäli jokin Unityn sisään tuoduista mallin ominaisuuksista ei toimi toivotulla tavalla. Tämän projektin kontekstissa tähtikuonokontiais-hahmon teksturoinnin materiaali ei toiminut toivotulla tavalla, joten loin uuden ulkoisen materiaalin toivotuilla ominaisuuksilla natiivisti Unityssä ja vaihdoin sen hahmolle korjaten ongelman.

## 9.1 Funktionaalinen implementaatio

Suurin osa hahmon implementaation funktionaalisuudesta on ymmärrettävästi koodaukseen pohjautuvaa, eikä täten eroa suuresti ohjelmoinnista muussa ohjelmistokehityksen kontekstissa. Unity hyödyntää ohjelmointikielenään Microsoftin kehittämää C#-kieltä. Ohjelmoinnin ja koodauksen yleisiin teoreettisiin käsitteisiin syventymättä, hahmon funktionaalinen implementaatio toteutui luomalla C#-skriptin, joka tukee hahmon ja sitä seuraavan kameran liikkumista kolmannen persoonan perspektiivistä. Tällä hahmolla on pelkkä kävelyanimaatio- ja täten muita funktionaalisuuksia ei ole tässä kontekstissa luotu.

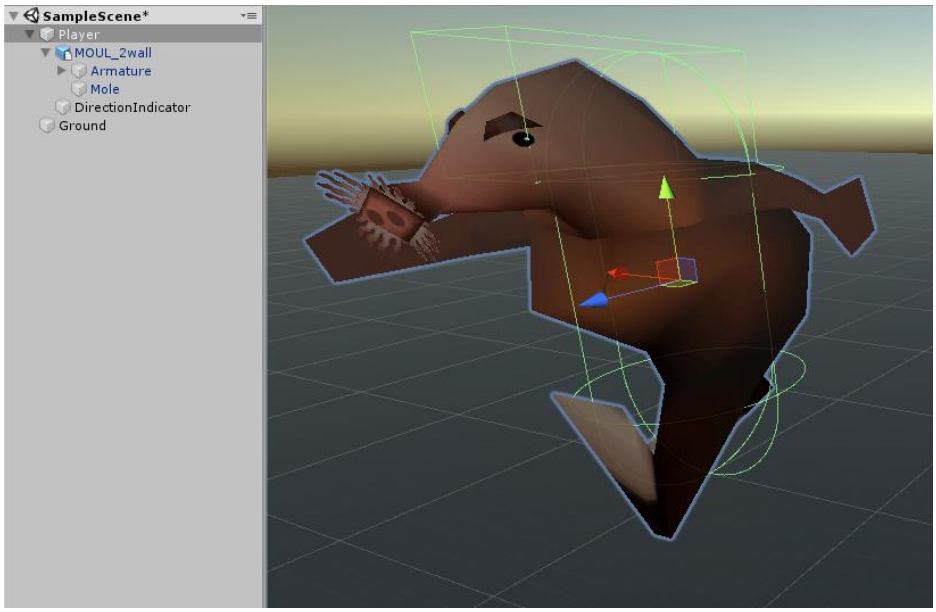
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CharacterController : MonoBehaviour {

    public Transform PlayerCam, character, centerPoint;

    private float mouseX, mouseY;
    public float mouseSensitivity = 10f;
    public float mouseYPosition = 1f;

    private float moveFB, moveLR;
    public float moveSpeed = 2f;

    private float zoom;
    public float zoomSpeed = 3;

    public float zoomMin = -2f;
    public float zoomMax = -10f;

    public float rotationSpeed = 1000f;

    // Use this for initialization
    void Start () {

        zoom = -3;

    }

    // Update is called once per frame
    void Update() {

        zoom += Input.GetAxis("Mouse ScrollWheel") * zoomSpeed;

        if (zoom > zoomMin)
            zoom = zoomMin;

        if (zoom < zoomMax)
            zoom = zoomMax;

        PlayerCam.transform.localPosition = new Vector3(0, 0, zoom);

        if (Input.GetMouseButton (1))  {
            mouseX += Input.GetAxis("Mouse X");
            mouseY += Input.GetAxis("Mouse Y");
        }

        mouseY = Mathf.Clamp(mouseY, -10f, 60f);
        PlayerCam.LookAt(centerPoint);
        centerPoint.localRotation = Quaternion.Euler(mouseY, mouseX, 0);

        moveFB = Input.GetAxis("Vertical") * moveSpeed;
        moveLR = Input.GetAxis("Horizontal") * moveSpeed;

        Vector3 movement = new Vector3(moveLR, 0, moveFB);
        movement = character.rotation * movement;

        character.GetComponent<CharacterController>().Move(movement * Time.deltaTime);
        centerPoint.position = new Vector3(character.position.x, character.position.y + mouseYPosition, character.position.z);

        if (Input.GetAxis("Vertical") > 0 | Input.GetAxis ("Vertical") < 0) {

            Quaternion turnAngle = Quaternion.Euler(0, centerPoint.eulerAngles.y, 0);
            character.rotation = Quaternion.Slerp(character.rotation, turnAngle, Time.deltaTime * rotationSpeed * 15);

        }

    }
}
```

Kuvio 11 Hahmon liike- ja kamerafunktionaalisuutta ohjaava CharacterController-skripti Unityssä.

Skriptin luomisen jälkeen voidaan asettaa haluttu Unity-projektin sisäinen elementti kutsumaan sitä Unityssä, mikäli ko. elementti on tällä hetkellä projektin käytössä. Jotta Unityssä voidaan ottaa käyttöön elementtejä, hyödynnetään Unityn Scene-toiminnallisuutta. Scene on eräänlainen 3-, tai 2-ulotteinen renderöitävä oletustila, johon voidaan raahata eri projektiin tuotuja valmiselementtejä- tai johon voidaan luoda niitä hyödyntäen skriptejä. Unityllä onkin laaja C#-ohjelmakirjasto erinäisiä Unityn sisällä olennaisiin toiminnallisuuksiin vaikuttavia funktioita, joita voidaan kutsua Unityn C#-skripteissä.

Jotta hahmolla olisi jotain, jonka päällä seistä- asetin ensiksi Scenelle tason, joka toimii maana. Tämän jälkeen jo edellä mainittu kameran ja liikkeen ohjainskripti asetettiin scenelle luodun näkymättömän pillerin muotoisen Player -nimisen capsule-kappaleen kutsuttavaksi. Itse hahmomalli asetettiin tämän jälkeen capsulen lapseksi scenen hierarkiassa ja täten capsulen liike periytyy scenen sisällä näkyvälle hahmolle.

Hahmon ohjaimen tällä tavoin implementoimisen pääetu on se, että capsulen kaltaiselle yksinkertaiselle kappaleelle on huomattavasti helpompi laskea erinäisiä fysiikkaan, liikkeeseen, rotaatioon sekä muihin olennaisiin toiminnallisuuksiin liittyviä laskelmia ilman suuria poikkeustapauksia siinä vaiheessa, kun niihin pohjautuvia funktionaalisuuksia pelinkehitysprosessissa aletaan hahmolle implementoimaan. Tämän lisäksi koska itse funktionaalisuuden skriptit ovat pelaajahahmon 3D-mallin sijaan capsulen alaisuudessa, voidaan tarvittaessa nopeasti vaihtaa sen lapsena olevaa 3D-mallia, mikäli halutaan implementoida sama skripti toisen 3D-mallin kontekstissa.




Kuvio 12 Scenen hierarkia Unityssä.

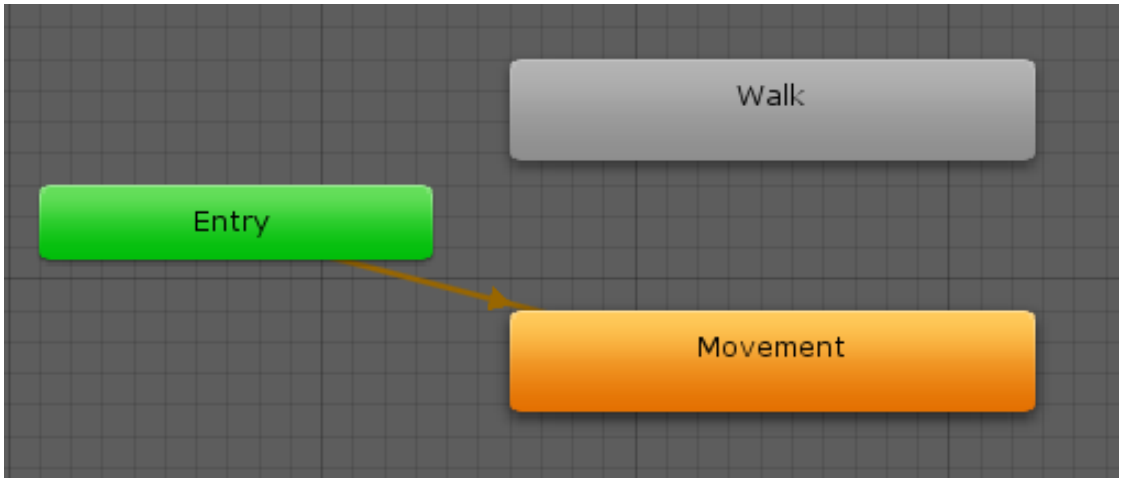
## 9.2 Animation controller ja Blend tree

Puhtaasti täysin koodipohjaisesta implementaation askeleista poiketen, voidaan myös hyödyntää joitakin Unityn sisäisiä valmistyökaluja implementaatiossa. Pelihahmon kontekstissa eräs näistä ominaisin lienee Unityn animation controller-funktionaalisuus, jolla voidaan ohjata mallin omistamia animaatioita. Tämä funktionaalisuus ohjaa animaatioiden toistoa interpoloimalla mallin animaatioita toistensa kanssa haluttujen parametrien mukaisesti muodostaen toiminnallisuuteen pohjautuvia animaatiokomplekseja.

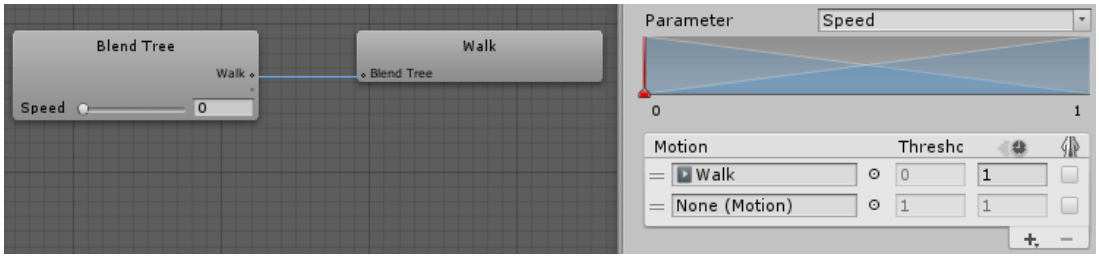
Tämä toteutetaan Unityn sisällä luomalla uuden animation controllerin sisälle blend tree -järjestelmiä. Näiden sisään vastaavasti voidaan kytkeä useampi erillinen animaatio, joiden välillä hahmoa interpoloidaan numeromuuttujan arvon mukaisesti. Tämän muuttujan arvoa voidaan taas säätää ja kutsua Unityn C#-skriptien sisällä eri funktiolla- sekä animation controllerin muilla elementeillä. Todellisessa implementaatiossa liikkumiseen littyvä animaatioiden blend tree saattaisi sisältää kolme eri animaatiota eri kävelyn nopeuksille interpoloiden niiden välillä määritellyn muuttujan ”Speed” -arvon mukaan. Tällöin kolmesta eri animaatiosta voitaisiin muuttujaa säätämällä luoda animaatioiden välimuotoja mille tahansa halutulle kävelytahdille.

Mikäli on tarve, useampia blend tree -järjestelmiä voidaan myös kytkeä toisiinsa muodostamaan monimutkaisempia animaatio-ohjaimia, joissa esim. hypätessä animaatio siirtyisi eri hyppysuuntien animaatioista koostuvan blend treen alaisuuteen ja takaisin kävelyn blend treen alaisuuteen hahmon kohdattua maan uudestaan. Tarvittaessa on myös mahdollista kytkeä yksittäisiä animaatioita animation controlleriin ja interpoloida niitä blend tree -kompleksien kanssa olematta niiden sisällä.

Tässä tapauksessa hahmolla on kuitenkin vain yksi toteutettu animaatio, jolloin animaatioiden implementaatio jää lähinnä tämän animaation kytkemiseen päälle- ja pois päältä, sekä sen nopeuden säätämiseen. Tämä on funktionaalisuus, joka voidaan tehdä tarvittaessa myös ilman animation controlleria suoraan kutsumalla animaation toistonopeutta skriptin sisällä, mutta luomalla blend tree voidaan varautua mahdollisuuteen, jossa animaatioiden määrä kasvaa. Tällä hetkellä implementaatio on toteutettu siten, että ”Speed” -muuttujan arvoksi tulee hahmon liikeskriptissä määritellyn liikesuuntavektorin normalisoitu pituus 0 ja yhden väliltä. Täten uusien animaatoiden lisääminen vaatisi vain niiden asettamista blend treen sisään.




Kuvio 13 Esimerkki-animation controller, jonka sisällä ”Movement” -blend tree ja ”Walk" -animaatio. Unityn kutsuessa animation controlleria se ajaa oletuksena ”Entry” liitäntään kytketyn animaation tai blend treen.




Kuvio 14 "Speed" -muuttuja ohjaa blend treetä; oikealla risteävät viivat kuvastavat blend treen eri tilojen välistä interpolaatiota eri muuttujan arvoilla.

## 10 Loppumietelmät ja johtopäätökset

Koen, että lopputuotos vastaa hyvin prosessin alussa asetettuja tavoitteita ja edustaa hyvää esimerkkiä siitä, miten harrastelijatason 3D-mallinnuksen prosessi pelikehityksen kontekstissa näyttää. Implementaatio oli prosessissa jokseenkin yksinkertainen, mutta toi esille hahmon kannalta relevanteimmat osapuolet, joten mielestäni se vastasi tämän projektin tavoitteita. Kokonaisuudessaan toteutus oli mielestäni onnistunut.

Teknisiltä ominaisuuksiltaan 3D-mallinnusprosessin myötä tuotettu hahmo mielestäni täyttää 2000-luvun viitekehyksen kriteerit ja olisi voinut helposti esiintyä ko. aikakauden pelissä, mikäli joku olisi vastaavanlaisen idean silloin saanut.

Koen, että henkilökohtainen kiinnostus kyseisen visuaalisen ilmeen hyödyntämiseen jossakin tähän opinnäytetyöhön suhteessa ulkoisessa projektissa on suuri tämän opinnäytetyön toteutuksen jälkeen ja kykenen toteuttamaan vastaavanlaisen prosessin huomattavasti nopeammin uudestaan, kuin ennen tämän opinnäytetyön toteuttamista. Koenkin siis tämän prosessin parissa työskentelyn myötä 3D-mallintamisen työskentelymetodieni optimoituneen ja rutinoituneen huomattavasti suhteessa lähtöpisteeseen.

Suurin haaste tälle opinnäytetyölle oli teoriapohja, sillä itse käytännön mallintamis- ja implementaatioprosessissa näiden hyödyntämismahdollisuudet ovat ymmärrettävästi rajoitettuja. 3D-mallintamisen teoreettisessa puolessa pystyin kuitenkin hyödyntämään niitä tukemaan informaatiota jossain määrin mielekkäällä tavalla. Aiheeseen liittyvä painettu kirjallisuus on kuitenkin jokseenkin sellaista, että aiheesta ei ole lähivuosina julkaistu paljoa helposti saatavilla olevaa painettua kirjallisuutta suomeksi. Onneksi kuitenkin mallintamisen taustalla oleva teoria on pysynyt kutakuinkin samana, vaikka terminologia on jokseenkin internetin myötä kehittynyt. Tämän opinnäytetyön kanssa mielekkäästi risteäviä sähköisiä akateemisia lähteitä on olemassa nihkeästi, mutta kuten tässä opinnäytetyössä onkin mainittu- internetissä aiheesta tarjolla olevan tiedon kirjo on massiivinen nykypäivänä ja koen, että tämän projektin ollessa suhteellisen käytännönläheinen- ei tästä aiheutunut ongelmia toteutuksen laadulle.

Kaiken kaikkiaan koen, että prosessi toimi kokonaisvaltaisesti mielekkäänä harjoitteena 3D-mallintamisesta ja sen hyödyntämisestä. Suosittelisin kaikkia aiheesta kiinnostuneita lämpimästi kokeilemaan vastaavanlaista projektia. Kaiken kaikkiaan uskon saaneeni tästä opinnäytetyöstä irti tulevaisuudensuunnitelmieni kannalta relevanttia tietotaitoa mielekkäällä tavalla.

Lähteet

Painetut

Puhakka, A. 2008. 3D-Grafiikka. Helsinki: Talentum.

Sähköiset

Tris, Quads, and N-Gons. Turbosquid. Viitattu 12.11.2022. https://resources.turbosquid.com/training/modeling/tris-quads-n-gons/

Blender homepage. Blender. Viitattu 15.11.2022. https://www.blender.org/

Unity homepage. Unity. Viitattu 15.11.2022. https://unity.com/

French, J.L. 2022. How to write a game design document with examples. Gamedevbeginner. Viitattu 15.11.2022. https://gamedevbeginner.com/how-to-write-a-game-design-document-with-examples/

Earthworm Jim. The Models Resource. Viitattu 15.11.2022. https://www.models-resource.com/nintendo_64/earthwormjim3d/model/5656/

Spyro. The Models Resouce. Viitattu 15.11.2022. https://www.models-resource.com/playstation/spyro2riptosrage/model/26411/

Crash Bandicoot. The Models Resource. Viitattu 15.11.2022. https://www.models-resource.com/playstation/crashbandicoot2cortexstrikesback/model/23690/

Tony Hawk. The Models Resource. Viitattu 15.11.2022. https://www.models-resource.com/pc_computer/tonyhawksproskater4/model/14631/

Kuviot



Taulukot