Heidi Hällström

Kaksijalkaisen rigin automatisointi

Oman autorigaustyökalun kehittäminen Mayassa

Metropolia Ammattikorkeakoulu

Medianomi

Viestinnän tutkinto-ohjelma

Opinnäytetyö

28.11.2022

Tiivistelmä

Tekijä(t):	Heidi Hällström
Otsikko:	Kaksijalkaisen rigin automatisointi
Sivumäärä:	39 sivua
Aika:	28.11.2022
Tutkinto:	Medianomi
Tutkinto-ohjelma:	Viestinnän tutkinto-ohjelma
Suuntautumisvaihtoehto:	3D-animointi ja -visualisointi
Ohjaaja(t):	Lehtori Jaro Lehtonen

Tämä opinnäytetyö käsittelee autorigaustyökalun rakentamista Python-ohjelmointikielen avulla 3D-ohjelmisto Mayassa. Työn tavoitteena on nopeuttaa rigausprosessia automatisoinnin avulla hyödyntäen Python-skriptejä. Hahmorigaus vaatii tarkkuutta, toistoa ja on aikaa vievää. Automatisoinnilla pyritään optimoimaan ajan käyttöä ja vähentämään virheitä. Työssä esitellään lyhyesti jo olemassa olevia autorigaustyökaluja, hahmorigauksen haasteita ja pohditaan oman autorigaustyökalun rakentamista vastaamaan näihin haasteisiin.

Työn teoriaosuudessa käydään läpi autorigaustyökalun lopputulokseen vaikuttavia tekijöitä, kuten hahmorigauksen tarpeita, Python-koodikielen rakennetta ja työkalun käyttöliittymän suunnittelun periaatteita. Lähteenä käytetään alan kirjallisuutta ja artikkeleita. Käytännön osuudessa esitellään oman autorigaustyökalun rakentamisen vaiheita, työkalun ominaisuuksia ja käyttöliittymää. Projektin vaiheita esiteltäessä käydään läpi koodin kirjoittamisessa tehtyjä valintoja ja sitä, mitä niillä on saatu aikaiseksi.

Projektin lopputuloksena on autorigaustyökalu, jolla pystyy rigaamaan kaksijalkaisen hahmon skinnaamista vaille valmiiksi. Autorigaustyökalun käyttöliittymä on helppokäyttöinen ja ohjaa käyttäjää rigauksen eri vaiheissa. Automatisoitu työkalu nopeuttaa kaksijalkaisen hahmon rigaamista ja ehkäisee virheiden syntymistä manuaalisesti rakennettuun rigiin verrattuna. Kaiken kaikkiaan, oman autorigaustyökalun rakentaminen nopeuttaa rigausprosessia ja mahdollistaa omien preferenssien implementoinnin rigeissä.

Avainsanat: 3D, Maya, rigaus, automatisointi, Python

Abstract

Author(s):	Heidi Hällström
Title:	The Automation of a Biped Rig
Number of Pages:	39 pages
Date:	28 November 2022
Degree:	Bachelor of Culture and Arts
Degree Programme:	Media
Specialisation option:	3D Animation and Visualization
Instructor(s):	Jaro Lehtonen, Senior Lecturer

This thesis concentrates on building an auto-rigging tool in 3D software Maya by using the programming language Python. The aim of this project is to speed up the process of rigging by means of automation, utilizing Python scripts. The character rigging process is repetitive, time-consuming and demands accuracy. Automation aims to optimize time usage and minimize human error. The objective is to briefly explore the ready-made auto-rigging tools, the challenges of rigging, and then build an own tool based on that.

I started by investigating the demands of character rigging in 3D industry, the structure of the programming language Python, and the principles of designing the user interface. I examined literature and articles about rigging, programming, and user interface design. The auto-rigging tool was built keeping in mind the results of the research. The steps of building the tool and creating the user interface are documented in the practical part of the thesis. When presenting the stages of the project, I go through the choices made in writing the code and what it has achieved.

The outcome is an auto-rigging tool that can rig a common biped character ready to be skinned. The user interface of the tool is easy to use, and it guides the user through the different stages of rigging. The automated tool speeds up the rigging of a biped character and prevents errors compared to a manually built rig. In conclusion, speeding up the rigging process by making an auto-rigging tool is a beneficial way to create a rig for one’s own needs.

Keywords: 3D, Maya, rigging, automation, Python

Sisällys

1 Johdanto 1

1.1 Työn lähtökohdat 1
1.2 Hahmorigauksen perusongelmat 2
1.3 Saatavilla olevat autorigaustyökalut 2

2 Keskeisiä käsitteitä 6

3 Mitä ongelmia lähdin ratkomaan 7

3.1 Hahmorigaaminen 7
3.2 Koodi ja sen käytännöt 13
3.3 Käyttöliittymä 17

4 Autorigaustyökalun rakentaminen ja esittely 19

4.1 Koodi 20
4.2 Ominaisuudet 20
4.3 Käyttöliittymä 27
4.4 Koodin turvaaminen 31

5 Yhteenveto 33

5.1 Prosessin ongelmakohdat ja loppupäätelmät 33
5.2 Miten parantelisin työkalua 35

Lähteet 36

1 Johdanto

Opinnäytetyöni käsittelee oman autorigaustyökalun rakentamista Python-koodikielen avulla. Aluksi esittelen aiheeni ja työni lähtökohdat. Käyn läpi hahmorigauksen perusongelmia ja esittelen muutamia valmiita autorigaustyökaluja. Toisessa luvussa käyn läpi rigauksen ja koodaksen perussanastoa. Kolmannessa luvussa kerron ongelmista, joita lähdin ratkomaan, ja neljännessä luvussa esittelen oman autorigaustyökaluni. Lopuksi yhteenvedossa käyn läpi prosessin ongelmakohtia ja kerron, miten lähtisin jatkokehittelemään työkalua. Opinnäytetyössäni oletan, että lukija on tutustunut Maya-3D-ohjelmiston käyttöön ja tuntee rigauksen alkeet.

Rigaamisella tarkoitetaan 3D-hahmolle tai objektille luotavaa animointijärjestelmää eli luurankoa, jolla mallia voidaan liikuttaa, sekä sen käyttöliittymää, jonka avulla mallia animoidaan (ScreenSkills 2021). Autorigaustyökaluilla tarkoitetaan koodeja, jotka rakentavat tällaisia animointijärjestelmiä automaattisesti.

1.1 Työn lähtökohdat

Opinnäytetyöni lähtökohtana oli perehtyä koodaamaan Python-ohjelmointikielellä ja tehdä sen avulla jokin itseäni hyödyttävä työkalu 3D-ohjelmisto Mayassa. Oma mielenkiintoni on suuntautunut hahmojen rigaukseen, ja halusin tehdä työkalun, joka helpottaisi tulevaisuudessa työtäni. Lähdin tekemään työkalua sen hetkisillä tiedoillani rigaamisesta, sillä pääasiani oli opetella koodaamaan. Työn automatisointi koodin avulla on alkanut olemaan rigaajalla vaadittu taito. Työn nopeuttamiseksi monilla tuotannoilla onkin omat autorigaustyökalunsa. (ScreenSkills 2021.) On kuitenkin hyvä tietää, miten työkaluja on kehitetty, ja tämän tiedon omaksuminen olikin yksi työni lähtökohdista.

Päädyin suunnittelemaan ja toteuttamaan kaksijalkaisen hahmon autorigaustyökalun, sillä sen perustan tekeminen toistui omissa opinnoissani jatkuvasti. Tämän tekeminen oli myös tullut minulle hyvin tutuksi, joten minun ei tarvinnut

käyttää aikaa rigaustekniikoiden opetteluun, vaan pystyin suoraan aloittamaan työkalun kirjoittamisen.

1.2 Hahmorigauksen perusongelmat

Rigaaminen on yksi pääkomponenteista 3D-hahmoanimoinnissa ja tapahtuu noin keskivaiheilla tuotantoa, ennen itse animointia. Prosessi ei kuitenkaan välttämättä ole lineaarinen, vaan tuotannon eri osat menevät osittain päällekkäin. Hahmoanimoinnissa rigit ovat tärkeimpiä animaattorin työkaluja. Rigaajan on hyvä pitää mielessä, että animaattoreiden tausta saattaa vaihdella, eikä heillä välttämättä ole taustaa teknisten toimintojen tulkitsemiseen. Näin ollen rigaajan kaksi tärkeää tavoitetta on tehdä rigistä helppo käyttää ja vaikea rikkoa. (Maraffi 2009, 2–5.)

Hahmorigausta vaikeuttaa se, että sitä on mahdollista tehdä monella eri tapaa, ja mielipiteitä parhaista tavoista on useita (O'Hailey 2019, luku 1). Rigaaminen vaatii paljon tarkkaa työtä, ja rigi itsessään on kompleksi animointijärjestelmä, jonka tulee lopulta olla helposti lähestyttävä ja mukava käyttää. Rigissä tulee ottaa huomioon muun muassa luiden sijainti ja orientaatiot, kontrolliobjektien ulkomuoto ja liikutettavuus, oikeanlaiset objektien yhteydet ja geometrian liikkuvuus luiden mukana.

3D-tuotannoissa kaksijalkaisia hahmoja voi olla hyvinkin monenlaisia erilaisilla mittasuhteilla ja ominaisuuksilla. Rigin rakentaminen toistuvasti erilaisille hahmoille on hidasta ja työlästä. Ajan säästämisen ja virheiden minimoimisen saavuttamiseksi monilla 3D-tuotannoilla on omat automatisoidut työkalunsa rigaamiseen. Vaikka on hyvä tietää, miten rigi rakennetaan, autorigaustyökalut alkavat olla alalla standardi. (ScreenSkills 2021.)

1.3 Saatavilla olevat autorigaustyökalut

Valmiita autorigaustyökaluja on saatavilla jonkin verran. Eri ohjelmistoilla on usein omat autorigaustyökalunsa, joiden avulla saadaan tehtyä yksinkertaisia kaksijalkaisia rigejä. Keskityn tässä opinnäytetyössä vain Mayan omiin tai

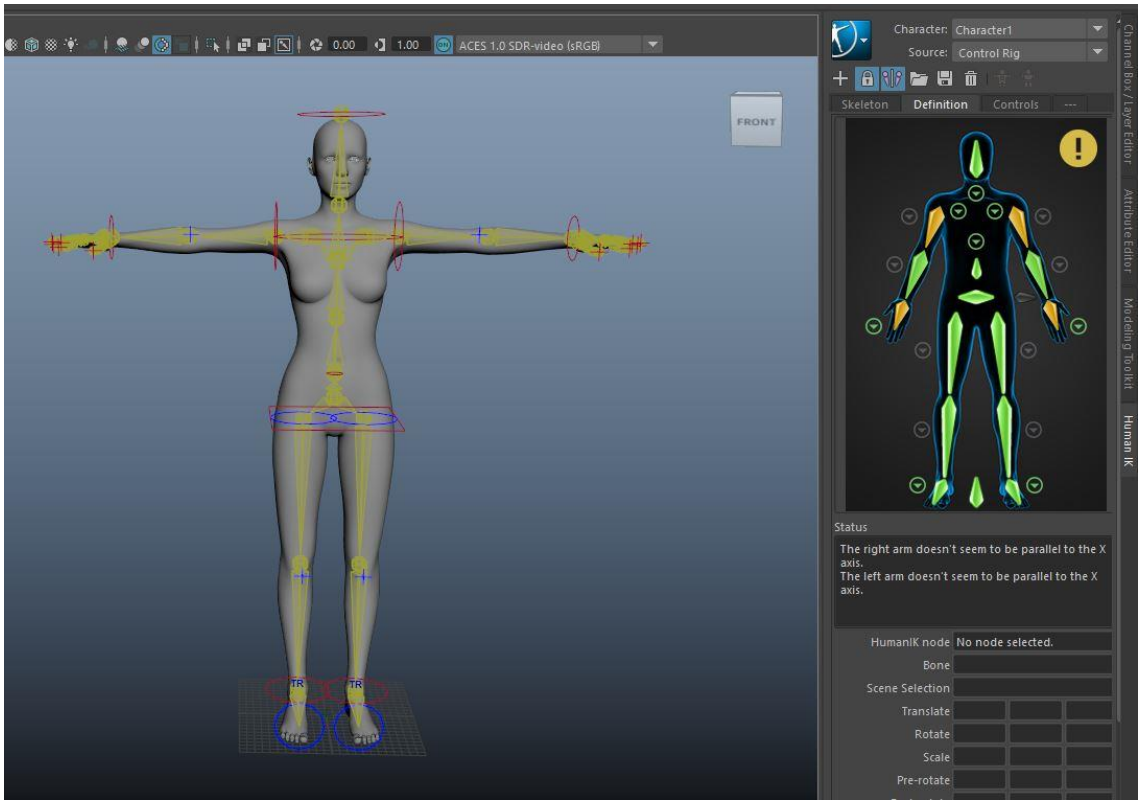
Mayan kanssa yhteensopiviin työkaluihin. Myös Mayaan on kehitetty useita plugineita ja maksullisia työkaluja, mutta käyn läpi ainoastaan yleisemmin tiedossa olevia ja käytettyjä metodeja nopeaan rigaukseen. Käyn läpi pari erilaista tapaa autorigata hahmo, jotka ovat Mayan sisältä löytyvä HumanIK ja Adoben luoman animaatiokirjaston Mixamon autorigaustyökalu.

Mayassa kätevin autorigaustyökalu löytyy Autodeskin kehittämästä HumanIK-väliohjelmistosta, jolla voidaan luoda yksinkertainen kokovartaloluuranko ja kontrollirigi. HumanIK tarjoaa uudelleenkohdistavan työkalun, jonka avulla saadaan muun muassa käytettyä samaa animaatiodataa eri kokoisten ja eri mallisten hahmojen välillä sekä yhdistettyä liikkeenkaappaus-dataa erilaisille hahmoille. (Autodesk 2021a.) HumanIK-työkalulla luuranko luodaan yhtä painiketta painamalla, jonka jälkeen luut asetellaan hahmon geometrian mukaan paikoilleen. HumanIK:lla tehtyyn rigiin kuuluu vartalon ja pään luut, käden ja jalkojen luut sekä sormien luut.

HumanIK:n vaihtoehtoinen käyttöliittymä on Quick Rig -työkalu, jonka lopputulos on yhdenmukainen HumanIK:n kanssa. Quick Rig -työkalua voi käyttää kahdella tapaa: joko käyttäen toimintoa, joka luo automaattisesti nopean rigin, tai käyttäen step-by-step-toimintoa, jolla käydään läpi yksityiskohtaisemmin hahmorigin luominen. (Autodesk 2021c.) Automaattinen yhden painalluksen toiminto antaa valmiiksi skinnatun rigin, mutta luiden paikat saattavat olla väärissä kohdissa, mikä johtaa polygoniverkon huonoon taipumiseen. Step-by-step-toiminnolla saa hieman paremman lopputuloksen, sillä sen avulla hahmon luontiin pystyy vaikuttamaan enemmän. Ensin asetetaan opasluut oikeille paikoilleen hahmon mittasuhteiden mukaan. Työkalu tarjoaa mukavasti vaihtoehtoja opasluiden käytön helpottamiseksi. Oikean ja vasemman puolen luita voi vapaasti siirtää, ja ne saa helposti peilattua toiselle puolelle symmetriseksi. Työkalu tarjoaa myös painikkeet kaikkien opasluiden valitsemiselle, piilottamiselle ja värien vaihtamiselle. Kun painikkeet ovat paikallaan, voidaan siirtyä rigin rakentamiseen. Työkalussa on painikkeet luurangon ja kontrollirigin luomiselle ja tämän jälkeen skinnaukselle. Kummankin metodin lopullisen rigin käytettävyys on sa-

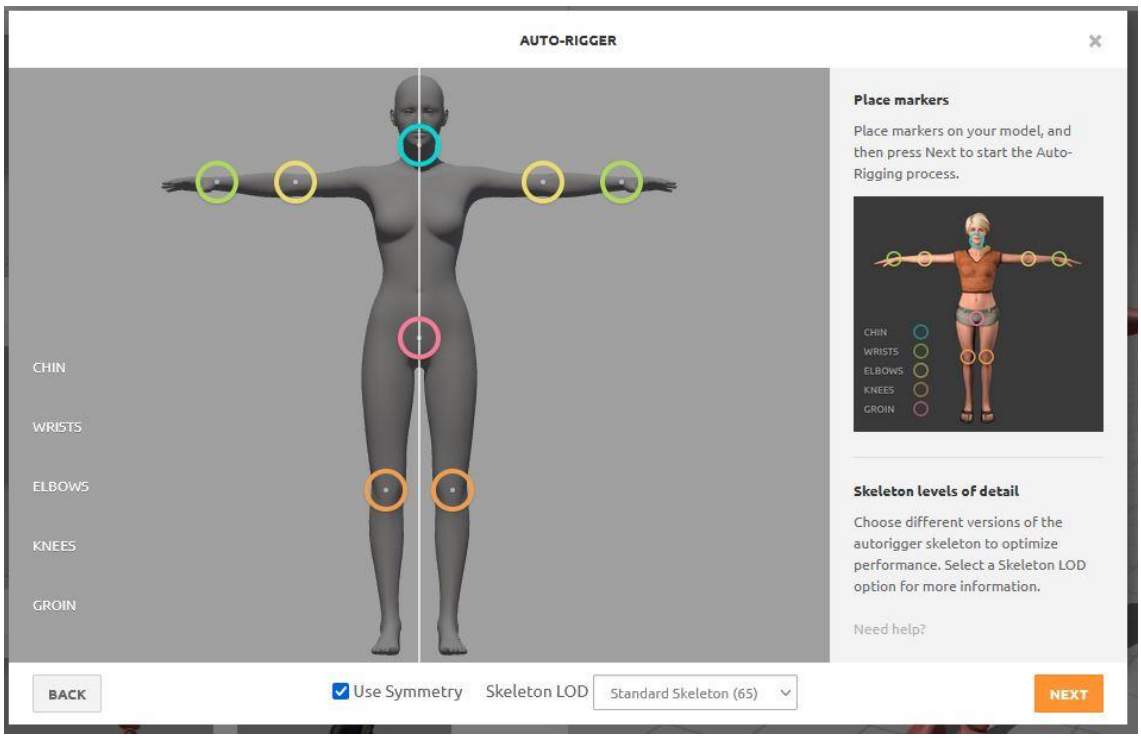
manlainen. Quick Rigin hyviä puolia on sen nopeus ja helppokäyttöisyys. Isoimpia ongelmia lopullisessa rigissä on sormien puute, jotka ovat tärkeä osa hahmoanimaatiota. Rigissä ei myöskään ole erikseen kontrolliobjektia päälle ja kaulalle eikä lantion alueen heilutusta mahdollistavaa kontrolliobjektia, joka on kätevä muun muassa juoksu-, hyppy- ja kävelyanimaatioiden teossa. Jalkojen liikuttelu toimii suhteellisen hyvin, mutta niistä puuttuu usein käytetyt jalkojen attribuutit, joilla hallitaan jalan askellusta.

Koin luuketjujen siirtelyn hieman vaivalloiseksi HumanIK:lla, ja pidin Quick Rigin vaihtoehtoisella käyttöliittymällä tehtävästä luiden sijoittelusta enemmän. HumanIK:n luurankona voi käyttää myös Quick Rigilla tehtyä luurankoa, mutta tähän tulee lisätä sormien luut itse, ja tällöin autorigaustyökalun nopeus ja vaivattomuus kärsii.



Kuva 1. HumanIK:sta saatava rigi.

Toinen nopea metodi rigin luomiseen on käyttää Adoben Mixamo-animaatiokirjastoa. Hahmo muunnetaan fbx-tiedostoksi ja ladataan Mixamoon. Mixamon rigaustyökalussa asetetaan merkit leuan, ranteen, kyynärpään, polvien ja haaravälin kohdille (ks. kuva 2). Tämän jälkeen luodaan rigi, jota voidaan käyttää mallina Mixamon valmiissa animaatioissa. Hahmo ladataan Mixamosta halutun animaation kera, jonka jälkeen se tuodaan takaisin Mayaan. Mayassa näkyy luuranko, joka on luotu Mixamossa, mutta minkäänlaista kontrollirigiä Mixamon autorigistä ei saa. Mixamon tulokset eivät ole erityisen haluttuja juuri kontrollirigin puutteen takia, mutta Mixamon työkalu sopii erinomaisesti animaatioiden käyttämiseen sellaisenaan tai hahmon asettamiseen haluttuun asentoon.



Kuva 2. Mixamon hahmon autorigaustyökalu (Adobe 2022).

Kaikissa metodeissa rigi jää vajavaiseksi. HumanIK antaa vain kohtalaisia tuloksia, joiden käyttö sopii animaatioihin, joissa voidaan joko käyttää suoraan Mixamon animaatioita tai jossa hienovaraiselle animaatiolle ei ole tarvetta. Liikkeenkaappauksesta saatavan datan hyödyntämiseen hahmolla HumanIK:n käyttö on erittäin hyödyllinen työkalu.

2 Keskeisiä käsitteitä

Rigi on animointijärjestelmä, joka koostuu mallille tehdystä luurangosta ja sen käyttöjärjestelmästä, jolla mallia ohjataan.

Skinning tarkoittaa prosessia, jossa määritellään polygoniverkko liikkumaan rigin mukana osoittamalla, kuinka paljon tietty verteksi liikkuu luun tai luiden mukana (O'Hailey 2018, 129).

Joint, eli rigin luut, joilla luustoa ja siten koko mallia taivutellaan (Sievanen 2020).

Kontrolliobjekti on apuobjekti, jonka avulla luita liikutetaan. Animaatiota ei ole hyvä tallentaa suoraan luihin, vaan sen sijaan käytetään kontrollereita, jotka mahdollistavat rigin käytön.

Lokaattori on pieni kuvake, joka merkitsee pistettä avaruudessa.

IK, Inverse Kinematics, tarkoittaa luiden liikkeen laskemista epähierarkkisessa järjestyksessä, tarkoittaen esimerkiksi sitä, että ranne liikuttaa kyynärpäätä ja olkapäätä (O'Hailey 2018, 79).

Gimbal-lock tapahtuu, kun luun rotaatio kiertyy tavalla, jolla se estää toisen rotaatioradan kiertymisen (O'Hailey 2018, 225).

Skripti on tiettyyn erikoistarkoitukseen tehty ohjelma. Se on usein yhdistelmä valmiita työkaluja ja kirjastoja. Skriptejä käytetään usein automatisoimiseen. (Saikkonen 2011.)

3 Mitä ongelmia lähdin ratkomaan

Tässä luvussa käyn läpi autorigaustyökalun luomisen tärkeitä kohtia, joiden avulla pääsen liikkeelle työkalun rakentamisessa. Tulen keskittymään ensinnäkin hahmorigaukseen ja sen haasteisiin, joista keskeisimpiä ovat luurangon rakentaminen ja kontrollirigin luominen. Tämän jälkeen perehdyn Python-koodikielen ominaisuuksiin, pohdin koodikielen valintaan liittyviä kysymyksiä ja käyn läpi käytäntöjä siistin koodin kirjoittamiseen. Lopuksi esittelen käyttöliittymän rakentamisen periaatteita.

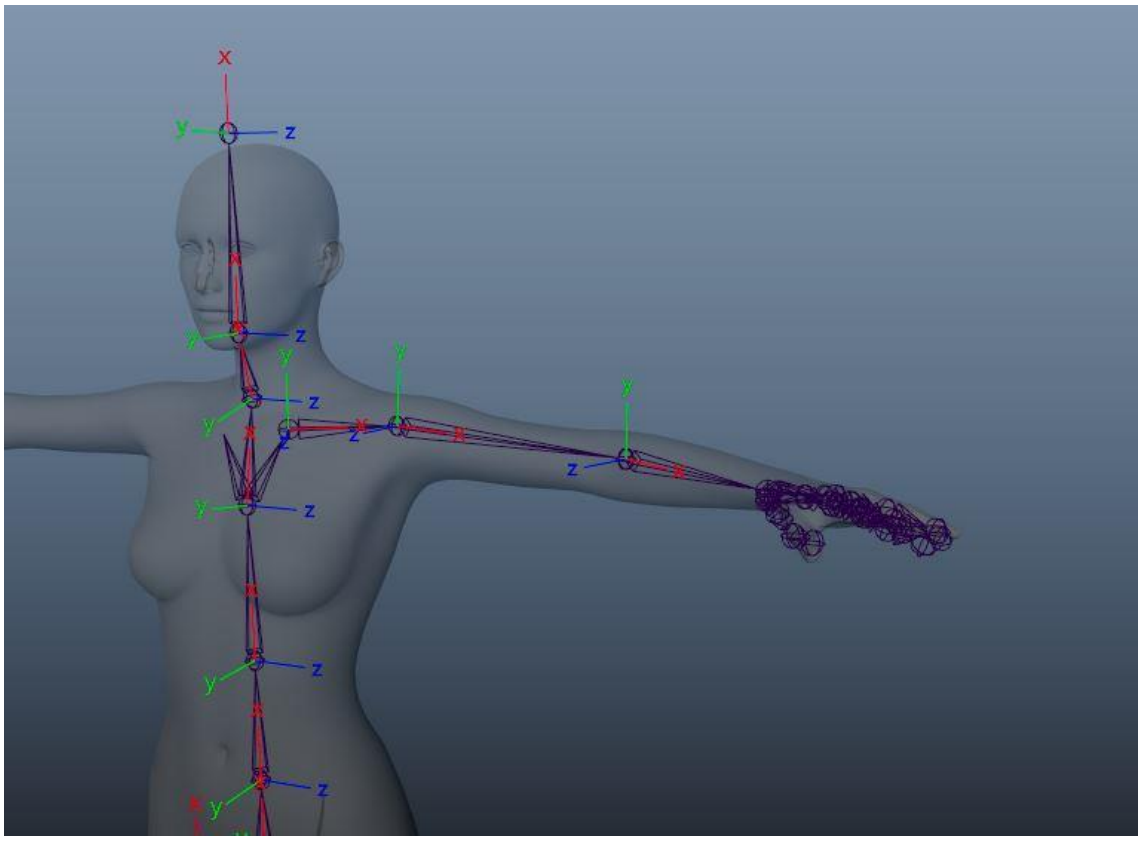
3.1 Hahmorigaaminen

Hahmorigauksen perustana on luoda luuranko ja käyttöliittymä hahmolle, jotka yhdessä muodostavat hahmon rigin. Rigin tarkoituksena on olla mahdollisimman helppokäyttöinen ja intuitiivinen. Sen tulee olla animaattorin tarpeita varten rakennettu, ja siinä tulee ottaa huomioon sen tuotannon vaatimukset, jota varten rigi rakennetaan. (Maraffi 2009.)

Kaksijalkaisten, usein humanoidien hahmojen, rigaaminen on suurin osa hahmorigaajan työtä, ja hyvä työkalupakki prosessin apuna on korvaamaton. Rigit on hyvä pitää modulaarisina, jotta ne voi mukauttaa erilaisiin hahmoihin mahdollisimman vähillä muutoksilla. (Assaf 2015.) Monien kaksijalkaisten hahmojen pohjana käytetään samanlaista luurankoa, jonka mittasuhteet vaihtelevat hahmon mukaan. Rigin rakentaminen on tarkkaa työtä, jossa tulee noudattaa tarkasti tiettyjä sääntöjä. Esittelen seuraavaksi prosesseja, jotka toistuvat hahmorigauksessa jatkuvasti hahmosta riippumatta.

Lähdetään tarkastelemaan rigiä luista, joiden asettamisesta hahmon rigaaminen aloitetaan. Ensisijaisesti on tärkeää määrittää luun orientaatio. Tämä on ylätason rotaatio, jolla luun koordinaatiston akselit säädetään osoittamaan haluttuihin suuntiin. Näin luun alkuasento saadaan oikeaksi, niin että sen rotaatioarvot pysyvät nollassa. Useimmiten hahmorigauksessa pyritään rakentamaan luut niin,

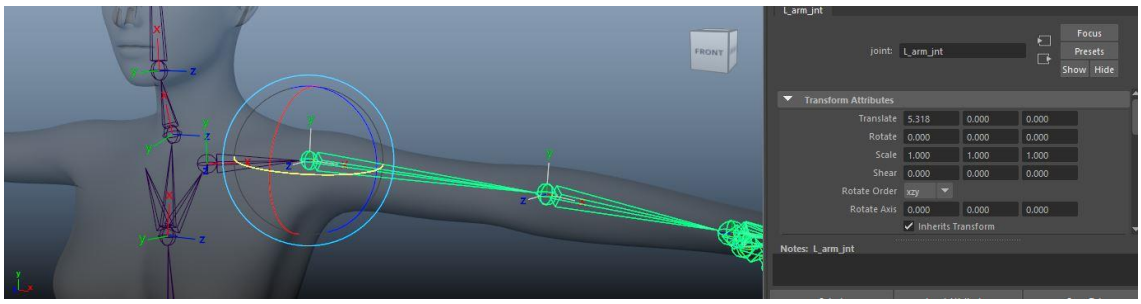
että niiden x-akseli osoittaa seuraavaan luuhun ja luun y-akseli ylöspäin. Selkeyden vuoksi z-akselin on hyvä osoittaa samaan suuntaan muiden luiden z-akseleiden kanssa. Poikkeuksena tähän ovat sormet, joissa peukalon luontainen taipumissuunta on horisontaalinen, kun muut sormet taipuvat vertikaalisesti. Rigin päätarkoitus on luoda animaattorin työstä helpompaa, joten tätä nopeuttaakseen sormien luut voidaan rakentaa taipumaan luonnollisella tavalla. Parhaimmassa tapauksessa sormien liikettä voi säätää samasta kontrollerista, jolla sormet saadaan puristumaan nyrkkiin ja vastavuoroisesti levittymään auki. Tällöin peukalon luun y-akselin tulee osoittaa sivulle ja z-akselin ylöspäin.



Kuva 3. Luiden orientaatiot.

Luiden liikkeessä tulee selvittää kiertojärjestys, jotta voidaan pyrkiä estämään Gimbal-lock eli jonkin kiertoradan lukittautuminen (O’Hailey 2018, 225). Luun kiertojärjestyksellä tarkoitetaan järjestystä, jossa luu pyörii akselinsa ympäri käyttäen joko x, y tai z-akselia. Mayassa luun kiertojärjestyksen oletusasetus on

xyz, jossa luun järjestys lasketaan päinvastaisesti eli z-akselista x-akseliin. Toisin sanoen, z-akseli vaikuttaa molempiin y- ja x-akseleihin ja y-akseli vaikuttaa x-akseliin. X-akseli pyörii vain itsensä ympäri eikä vaikuta kahteen muuhun. Esimerkiksi olkapään ja olkavarren luille tyypillisin kiertojärjestys on xzy, kyynärpään luulle zxy ja ranteen luulle yzx. (Assaf 2015, luku 6.) Olkapään luulla tämä tarkoittaa sitä, että luun kiertojärjestys alkaa eniten käytetystä rotaatiosta eli horisontaalisesti kääntyvästä y-akselin rotaatiosta. Tätä rotaatiota käytetään esimerkiksi kävely- ja juoksuanimaatioissa, joissa käden liike on sivuttaista. Sen jälkeen lasketaan z-akselin rotaatio, eli ylös- ja alaspäin kääntyvä akseli. Lopuksi lasketaan vähiten käytetty akseli, joka on x-akseli.



Kuva 4. Olkavarren rotaatiojärjestys xzy.

Luiden asennossa tulee ottaa huomioon IK-ketjun rakentaminen. IK-ketju alkaa ensimmäisestä luusta ja loppuu viimeiseen. Näiden välillä voi olla monia luita, jotka taipuvat IK-ketjun mukaan (Autodesk 2021b). IK eli Inverse Kinematics tarkoittaa luiden liikkeen laskemista epähierarkkisessa järjestyksessä. Tämä tarkoittaa esimerkiksi sitä, että ranne liikuttaa kyynärpäätä ja olkapäätä. (O'Hailey 2018, 79.) IK-ketju tulee tyypillisesti jaloille, jotta hahmoa voidaan liikuttaa ilman, että jalat liikkuvat vartalon mukana. IK-ketjulla tehdyt jalat mahdollistavat muun muassa kävelyanimaation tekemisen. IK-ketjun asettamista varten ketjuun kuuluvien luiden tulee muodostaa kulma, jonka mukaan IK-ketju taipuu. IK-ketjun taipumiskulman määrittämiseen on muitakin tapoja, mutta pääasia on saada ketju taipumaan halutussa kulmassa.

Rigin luurankoa rakentaessa on otettava huomioon oikeanlainen hierarkia, jotta luut seuraavat toisiaan oikeanlaisessa järjestyksessä. Usein luurangon ensimmäinen luu on lantioluu, johon yhdistyvät selkäranka ja jalat. Jalan luut rakentuvat reisiluusta, sääriluista, jalkapöydästä ja varpaista. Selkäranka koostuu yhdestä tai useammasta selkärankaluusta, rintaluusta, kaulaluusta ja pääluusta. Rintaluuhun yhdistyvät solisluu, olkavarren luu, käsivarren luu ja kämmenen luu. Kämmeneen yhdistyvät sormien luut.

Luurangon valmistuttua siirrytään kontrollirigin eli hahmon kontrollijärjestelmän luomiseen. Animaattorin ei tule koskea rigin luurankoon, vaan kontrollirigi on animaattorin työväline. Kontrollirigi koostuu pääosin kontrolliobjekteista, jotka ohjaavat luita. Kontrolliobjektit ohjaavat luita erilaisten constraintejen tai muiden kytkösten avulla. Constrainit ovat eri tavoin määriteltyjä yhteyksiä ohjaavan objektin ja ohjatun objektin välillä. Nämä yhteydet vaikuttavat objektien välillä niiden liikkeeseen, skaalautuvuuteen ja kiertymiseen. (O'Hailey 2018, 231.) Kontrollirigin rakentamisessa oikein asetetut constraintit ja kontrolliobjektien hierarkia vaikuttavat merkittävästi rigin käytettävyyteen.

Rigin käytettävyyden parantamiseksi muun muassa jaloille voi luoda attribuutteja, joilla jalan liikuttaminen helpottuu. Attribuuttien luominen tapahtuu connection editorin kautta, yhdistämällä haluttu toiminto tai objekti kontrolliobjektiin tehtyyn attribuuttiin.

Kaiken tämän toistaminen jatkuvasti vie aikaa ja koko prosessi on altis virheille. Pienet virheet prosessin alkuvaiheessa voivat johtaa prosessin tekemiseen alusta alkaen. Siksi pidetäänkin hyvänä toimintatapana skriptata eli rakentaa koodin avulla rigi, sillä niin virheet voi korjata nopeasti, ja voidaan välttyä rigin uudelleenrakentamiselta manuaalisesti. (ScreenSkills 2021.)

On hyvä pohtia, mitä tarpeita autorigaustyökalulla tulee olla vastatakseen tähän prosessiin. Työkalun tulee helpottaa edellä mainittuja toimintoja samalla minimoiden virheiden mahdollisuutta. Autorigaustyökalussa on tärkeää miettiä, ke-

nelle työkalua tehdään. Rigausta aloittelevan tai siihen perehtymättömän käyttäjän tarpeisiin vastatakseen autorigaustyökalun tulisi olla mahdolisimman yksinkertainen ja suorittaa edellä mainitut toiminnot ilman suurempia säätelymahdollisuuksia. Kokeneemman rigaajan käsissä tai monimutkaisempien rigien rakentamisessa autorigaustyökalun toimintojen olisi hyvä olla mahdollisimman paljon muokattavissa. Tärkeintä on kuitenkin tuottaa mahdollisimman mutkattomasti hyvä pohjarigi, jota voi käyttää sellaisenaan tai jonka päälle kokeneempi rigaaja pääse rakentamaan haastavampia toimintoja.

Vastatakseen modulaarisuuden tarpeisiin autorigaustyökalun tulisi olla käyttäjälle moderoitavissa. Muun muassa käsien ja selkärangan luiden lukumäärä voisi olla sellainen, johon käyttäjä pystyisi vaikuttamaan.

Otetaan tarkempaan tarkasteluun selkärangan rigaaminen. Ihmisellä selkäranka jaetaan kolmeen osaan, jotka ovat kaularanka, rintaranka ja lanneranka. Kaularanka koostuu seitsemästä nikamasta, rintaranka eli yläselkä kahdestatoista ja lanneranka eli alaselkä viidestä. (Selkäkanava 2022.) Näin monelle luulle ei kuitenkaan ole tarvetta, vaikka hahmo haluttaisiinkin pitää realistisena. Nikamien lukumäärää tärkeämpää on ymmärtää, miten selkäranka liikkuu. Alaselkä on se alue selkärangasta, josta suurin osa selän liikealueesta tapahtuu. Alaselkä sekä liikkuu että kiertyy. Keskiselkä, rintakehän takana, ei itsestään kierry, sillä kylkiluut ovat kiinni nikamista. Realistista rigiä varten luiden sijaintiin on kiinnitettävä erityistä huomiota, jotta ne saadaan jäljittelemään ihmisselkää. (O’Hailey 2018, 119.) Tämän vuoksi on tärkeää antaa autorigaustyökalun käyttäjälle mahdollisuus muokata luiden sijaintia. Enemmän sarjakuvamaista hahmoa varten on hyvä mahdollistaa rigissä myös yläselän kierto. Kummin vain, on kuitenkin tärkeää pyrkiä pitämään luiden lukumäärä mahdollisimman pienenä, ja realistisempaankin selän rigiin riittää usein kolmesta neljään luuta. (O’Hailey 2018, 119.) Tavallisimmin humanoideissa rigeissä on yksi tai kaksi selkäluuta riippuen hahmon mittasuhteista ja käyttötarkoituksesta. Näin ollen hahmon selkäluiden määrä on sellainen muuttuja, joka olisi hyvä antaa käyttäjän itse määritellä.

Käden rigauksessa on otettava huomioon käden kokonaisvaltainen liike ja siihen vaikuttavat tekijät. Käsivarsi koostuu kolmesta luusta, jotka ovat olkavarren luu *humerus* ja kyynärvarren luut *radius* ja *ulna* (Mayo Clinic 2022). Humerus kiinnittyy yläpäästään pallonivelellä lapaluuhun, mikä tekee käsivarresta vartalon liikkuvimman osan. Humeruksen alaosa kiinnittyy kyynärvarteen kyynärnivelellä. Kyynärvarren mutkikas luu- ja niveljärjestelmä mahdollistaa sen luiden, radiuksen ja ulnan, kiertymisen toistensa ympäri. (Fontini Publishing house 2021.)

Kaikki tämä kuulostaa rigauksen kannalta monimutkaiselta, mutta usein perusrigeissä näkee käsivarressa olevan kahdesta neljään luuta. Kahdella luulla pärjää, ja useammat luut mahdollistavat luonnollisemman käden kierron ranteessa ja olkavarressa. (Sarkamari 2020.) Sen lisäksi käsivarsi kiinnittyy solisluuhun, joka puolestaan kiinnittyy rintakehän luuhun. Autorigaustyökalun kannalta mahdollisuus luiden lisäämiseen olka- ja kyynärvarren sekä kyynärvarren ja ranteen välille saattaisi olla hyvä ratkaisu.

Yksi tärkeimpiä ominaisuuksia rigillä on sen helppokäyttöisyys, joka perustuu itse rigin käyttöliittymän hyvään suunnitteluun. Kontrolliobjektien tulisi olla loogisia ja intuitiivisia muodoiltaan, jotta animaattori ymmärtää, mitä vartalon osaa mikäkin kontrolliobjekti liikuttaa. Hyvä kontrollirigi kontrolliobjektien määrä on tasapainossa. Hyvin toimivassa rigissä saattaa olla satoja toimintoja, jolloin on hyvä priorisoida se, mitkä toiminnot tarvitsevat kontrolliobjektin ja mitkä toiminnot voi suorittaa muun muassa attribuuttien avulla. (Maraffi 2009, 5–8.)

Toinen kontrollirigiin liittyvistä tärkeistä ominaisuuksista on se, että rigin tulisi olla vaikeasti rikottava. Tämä voidaan saavuttaa tarjoamalla intuitiivisia kontrolliobjekteja, jotka toimivat hyvin rigin ja tuotannon tarpeiden kanssa samalla rajoittaen kontrolliobjektien käyttöä niin, ettei tarpeet ylity. (Maraffi 2009, 5–8.) Kontrolliobjektien liikuttamista voidaan rajoittaa niin, ettei animaattori pysty viemään liikettä tämän rajoituksen yli. Alalla on muodostunut käytännöksi erottaa toisistaan eri väreillä rigin oikea ja vasen puoli sekä keskilinja. Väritystapoja on erilaisia ja kontrolliobjektien värien valinta voisi olla hyvä ominaisuus autorigaustyökalussa.

Hahmorigauksessa on hyvä ottaa huomioon liikkeenkaappaus eli motion capture. Sillä tarkoitetaan tekniikkaa, jolla näyttelijän liikkeet tallennetaan ja tästä saatavaa dataa käytetään digitaalisesti luodun hahmon liikkeenä (Ahlroth 2007). Autorigaustyökalulla on hyvä tuottaa rigi, joka on yhteensopiva mocap-datan kanssa.

3.2 Koodi ja sen käytännöt

Jatkuvasti saman rigin rakentaminen ei ole ajallisesti kannattavaa, joten on hyvä pyrkiä automatisoimaan toimintoja. Toimintojen automatisoiminen tapahtuu skriptaamisen avulla. Paras tapa Mayassa tehdä näin, on käyttää Mayan sisäänrakennettuja koodikieliä, erityisesti Pythonia (Mechtley 2011).

Mayalla on myös oma koodikielensä MEL, joka koostuu sanoista Maya Embedded Language. MEL-skriptit, eli komentosarjat, määrittelevät ja luovat pohjimmiltaan Mayan graafisen käyttöliittymän. Mayan graafinen käyttöliittymä suorittaa MEL-käskyjä ja Maya-komentoja. MEL on suhteellisen helppo luoda, muokata ja suorittaa, mutta sitä käytetään vain Mayassa, ja sillä on useita teknisiä rajoituksia. MEL ei nimittäin tue olio-ohjelmointia, ja se voi kommunikoida Mayan kanssa vain komentomoottorissa määritettyjen rajapintojen joukon kautta tai kutsumalla Pythonia. (Mechtley 2011, luku 5.)

Python on alun perin Mayan ulkopuolella rakennettu koodikieli, joka tarjoaa tehokkaita ominaisuuksia ja jolla on laaja käyttäjäkunta. Koska MEL-koodikieli on hyödyllinen ainoastaan Mayassa, Pythonin käyttö ulottuu laajemmalle eri ohjelmistoihin. (Mechtley 2011, luku 5.) Perusteluita Pythonin käyttämiseen MEL:n sijaan on monia. Python on myös edistyneempi koodikieli, joten sen avulla saa tehtyä enemmän asioita kuin MEL-koodikielellä. Pythonilla pystyy kommunikoimaan eri koodikielten välillä, ja se voidaan suorittaa missä tahansa käyttöjärjestelmässä, mikä poistaa tarpeen kääntää työkaluja uudelleen eri käyttöjärjestelmille tai ohjelmistoversioille. (Mechtley 2011, luku 5.) Lisäksi Python on alan standardikieli, sen käyttö soveltuu Mayan lisäksi moniin eri 3D-alan ohjelmistoihin kuten Blenderiin, 3dsMaxiin, Houdiniin ja moniin muihin. (Richter 2017.)

MEL:n sanastoa on kuitenkin tärkeä ymmärtää, sillä monet Mayan komennot haetaan Python-koodikielen sisältä MEL-komennoista. Pääsy Mayan komentoihin saadaan tuomalla skriptiin maya.cmds-kirjasto. (Autodesk 2020.)

Pythonin perusteisiin kuuluvat muun muassa muuttujat, listat, kysyminen, if-ehtolauseet, while-silmukka, for-silmukka ja listat, funktiot ja luokat. Muuttuja on yksi tärkeimpiä ohjelmoinnin käsitteitä. Muuttujaan tallennetaan jokin ohjelmassa tarvittava tieto tai arvo, jota voi ohjelman edetessä vaihtaa. Pythonissa muuttujille on erilaisia tietotyyppejä, kuten merkkijonot (string), kokonaisluvut (integar), desimaaliluvut (floating-point) ja tosi/epätosi (boolean). Listat puolestaan ovat tapa ryhmitellä tietoja. Ne luodaan antamalla listalle nimi ja listan sisältö määritellään laittamalla = -merkin jälkeen listan jäsenet hakasulkuihin pilkulla eroteltuina. (Peda.net 2022.)

Kysyminen Pythonissa tapahtuu käyttämällä komentoa input, kun ohjelman halutaan kysyvän käyttäjältä tietoa. If-ehtolausetta käytetään, kun halutaan ohjata toimintaa ehtojen avulla. Ehtolauseen if-osiossa määritellään mitä tapahtuu, jos annettu lause toteutuu. Lauseessa on myös else-toiminto, jossa määritellään mitä tapahtuu silloin, kun lauseen ensimmäinen osa ei toteudu. (Peda.net 2022.) Tämän lisäksi on myös vaihtoehto käyttää elif- eli else if -lausetta, jonka merkitys on seuraava: jos edelliset ehdot eivät olleet totta, kokeile tätä ehtoa (W3 Schools 2022). Elif-osia voi olla useita, mutta else-osia vain yksi (Peda.net 2022).

Pythonilla on kaksi primitiivistä silmukkakomentoa, jotka ovat while-silmukka ja for-silmukka. While-silmukalla suoritetaan joukko lauseita niin kauan, kun ehto on tosi. Break-lauseella while-silmukan pystyy pysäyttämään, vaikka sen ehto olisi vielä tosi. Continue-lauseella pystytään pysäyttämään while-silmukan nykyinen iteraatio ja jatkamaan seuraavalla ehdolla. Else-lauseella voimme suorittaa seuraavan ehdon tai koodin osan, kun while-silmukan ehto ei ole enää tosi. For-silmukalla voidaan puolestaan suorittaa joukko lauseita, mikä tapahtuu kerran jokaiselle listan, tuplen tai joukon kohteelle. (W3 Schools 2022.) For-silmukassa luodaan muuttuja, joka on usein esimerkeissä käytetty i. Sille luodaan

myös lista, jossa on niin monta jäsentä kuin silmukkaa halutaan toistaa. For-silmukkaan annetaan jonkin käsky, jonka silmukka suorittaa kerran jokaiselle listan jäsenelle. While- ja for-silmukat eroavat toisistaan siten, että while-silmukassa toistojen määrää ei tarvitse tietää etukäteen. (Peda.net 2022.)

Pythonissa funktioita voi myös kirjoittaa itse. Funktioiden tarkoitus on yhdistää monimutkaisia käskysarjoja kokonaisuudeksi, joita pystyy kutsumaan ohjelmassa myöhemmin. Funktio määritellään kirjoittamalla def ja sen perään funktion nimi. Parametrit kirjoitetaan funktion nimen perään sulkuihin, jonka jälkeen lisätään kaksoispiste. Tämän jälkeen funktion käskyt kirjoitetaan funktion määrittelyn alle sisennettynä. (Peda.net 2022.)

Python on olio-ohjelmointikieli, mikä tarkoittaa, että siinä voi tehdä luokkia, joihin voidaan tallentaa tietoja (Kivelä 2022). Melkein kaikki Pythonissa on olioita ominaisuuksineen ja menetelmineen (W3 Schools 2022). Oliot puolestaan ovat tietorakenteita, jotka sisältävät funktioita. Luodakseen olion täytyy määritellä luokka, joka kuvaa olion ominaisuudet. Funktioita luokan sisällä, jotka kuuluvat oliolle, kutsutaan metodeiksi. (Aalto-yliopisto 2022.) Luokkien merkityksen ymmärtämiseksi on hyvä tutustua __init__() -funktioon. Kaikilla luokilla on __init__()-funktio, joka suoritetaan aina kun luokka käynnistetään. Self-parametri, jota käytetään luokan yhteydessä, on viittaus luokan esiintymään sillä hetkellä. Sitä käytetään, jotta päästään käsiksi muuttujiin, jotka kuuluvat luokkaan. Self-parametria ei tarvitse nimetä, mutta sen on oltava minkä tahansa funktion ensimmäinen parametri. Inhertance-toiminnon avulla voidaan määrittää luokka, joka perii kaikki menetelmät ja ominaisuudet toisesta luokasta. (W3 Schools 2022.)

Ei ole olemassa sitovaa sääntöä, miten koodia pitäisi kirjoittaa. Huono koodi voi kuitenkin johtaa ongelmiin myöhemmin. Puhtaasti kirjoitettua koodia on helppo lukea, siitä on helppo korjata virheitä, sitä on helppo ylläpitää ja kehittää eteenpäin, ja se on erittäin suorituskykyinen. (Omeyer 2022.) Jotkin käytännöt edesauttavat näitä ominaisuuksia, ja niiden ymmärtäminen on tärkeää, jotta koodia olisi helppo jatkokehittää myöhemmin. Rigaamisessa on useita erilaisia tapoja

ja monia erilaisia ominaisuuksia, joten myös autorigaustyökalun jatkokehittäminen on olennainen asia.

Koodin kirjoittamisessa tulee ottaa huomioon hyvät nimeämiskäytännöt. Nimeämisessä kannattaa noudattaa johdonmukaisuutta. Sen ylläpitäminen on tärkeää sekaannusten välttämiseksi, varsinkin jos muut kehittäjät työskentelevät koodin parissa. Kun nimetään muuttujia, funktioita tai luokkia, kannattaa käyttää merkityksellisiä nimiä, jotka paljastavat niiden tarkoituksen (ks. kuva 5). Tällöin kannattaa suosia pitkiä ja kuvailevia nimiä lyhyiden ja moniselitteisten sijaan. Esimerkiksi muuttujien nimeämisessä koodin lukijan pitäisi pystyä käsittämään sen nimen perusteella, mitä muuttuja tallentaa sisäänsä. (Omeyer 2022.)

```
# Not recommended
c = [“UK”, “USA”, “UAE”]

for x in c:
print(x)

# Recommended
cities = [“UK”, “USA”, “UAE”]
    for city in cities:
        print(city)
```

Kuva 5. Miten kirjoittaa tarpeeksi kuvaavia nimiä (Omeyer 2022).

Kuten muuttujienkin kohdalla, funktioiden nimeämiskäytännöt on hyvä pitää johdonmukaisina ja selkeinä. Funktiot on myös hyvä pitää lyhyinä ja ytimekkäinä. Funktion tulisi tehdä yksi toiminto hyvin. Hyvä sääntö on, että jos funktion nimeen olisi tulossa ”ja”, se kannattaa hajauttaa kahdeksi eri funktioksi. Luokkien kanssa työskennellessä kannattaa välttää lisäämästä ylimääräistä kontekstia eli tarpeettomia muuttujia muuttujien nimiin (ks. kuva 6). (Omeyer 2022.)

```
# Not recommended
class Person:
    def __init__(self, person_username, person_email, person_phone, pers
        self.person_username = person_username
        self.person_email = person_email
        self.person_phone = person_phone
        self.person_address = person_address

# Recommended
class Person:
    def __init__(self, username, email, phone, address):

        self.username = username
        self.email = email
        self.phone = phone
        self.address = address
```

Kuva 6. Esimerkki siitä, kuinka luokkien kanssa työskennellessä tulisi jättää turhat muuttujien nimeämiset tekemättä (Omeyer 2022).

3.3 Käyttöliittymä

Käyttöliittymä on tärkeä osa tuotetta. Kun se on tehty hyvin, käyttäjä ei kiinnitä siihen huomiota. Toisaalta taas huonosti tehtyä käyttöliittymää on vaikea käyttää. Noudatettaessa käyttöliittymäsuunnittelun pääperiaatteita voidaan välttää epäonnistumisia. Käyttöliittymäsuunnittelun neljä pääperiaatetta ovat käyttäjän asettaminen käyttöliittymän hallintaan, tuotteen kanssa tapahtuvan vuorovaikutuksen tekeminen mukavaksi, kognitiivisen kuormituksen vähentäminen ja käyttöliittymän johdonmukaisuus. (Babich 2019).

Käyttäjän asettaminen käyttöliittymän hallintaan tarkoittaa sitä, että käyttäjä tuntee olonsa itsevarmaksi käyttäessään työkalua. On hyvä antaa käyttäjälle mahdollisuus mennä taaksepäin tai perua tekemisiään työkalussa, jotta käyttäjä voi tutkia käyttöliittymää huoletta. On tärkeää käyttää valikkoja, otsikoita ja navigointipolkuja, jotta käyttöliittymän navigointi pysyy selkeänä ja käyttäjä ei tunne olevansa eksyksissä missään vaiheessa. Käyttöliittymän tulisi olla läpinäkyvä, ja

käyttäjän tulisi tietää, mitä milloinkin tapahtuu. Jokaisen käyttäjän tekemän toiminnon jälkeen käyttöliittymän tulisi osoittaa, että toiminto on suoritettu onnistuneesti. On hyvä näyttää esimerkiksi lataamiseen liittyvät tapahtumat pitääkseen käyttäjän ajan tasalla. Käyttöliittymän suunnittelussa on hyvä pitää mielessä, että kaiken tyyppisten käyttäjien, aloittelijasta ammattilaiseen, tulee pystyä hyötymään järjestelmästä yhtäläisesti. (Babich 2019).

Tuotteen kanssa tapahtuvan vuorovaikutuksen tekeminen mukavaksi viittaa käyttöliittymän siistiin ulkoasuun. On hyvä poistaa kaikki toiminnot tai tiedot, jotka eivät ole hyödyllisiä tai hämmentävät käyttäjää. Käyttöliittymässä on hyvä ottaa huomioon sen käyttäjät käyttämällä kieltä ja termejä, joita kaikki helposti ymmärtävät ja jotka ovat helposti luettavissa. Työkalu on hyvä optimoida niin, että siitä poistaa päällekkäiset tai toistuvat toiminnot. Tärkeiden elementtien tulee olla helposti saavutettavissa ja havaittavissa. On tärkeää tarjota ratkaisuja ja ohjeita virheisiin, joita käyttäjä saattaa kohdata käyttäessään käyttöliittymää. (Babich 2019.) Virheiden mahdollisuutta on hyvä pyrkiä minimoimaan mahdollisuuksien mukaan.

Kognitiivisella kuormituksella tässä kontekstissa tarkoitetaan henkistä painetta, jota käyttäjä tuntee käyttäessään tuotetta. Hyvän tuotteen tavoite on vähentää kognitiivista kuormitusta. Tämä voidaan saavuttaa jakamalla isommat tiedot paloiksi ja vähentämällä ja optimoimalla tehtävän suorittamiseen tarvittavien vaiheiden määrää. Tiedot ovat hyvä näyttää siten, ettei käyttäjien tarvitse muistaa mitään, vaan he voivat helposti tunnistaa asioita aiemmista kokemuksistaan. Tiedot voidaan myös järjestää oikein tuotteen tehokkuuden parantamiseksi, jotta käyttäjät voivat tavoittaa tärkeät tiedot nopeasti. (Babich 2019).

Käyttöliittymän johdonmukaisuus tarkoittaa käyttöliittymän elementtien ja toimintojen tekemistä yhdenmukaiseksi koko tuotteessa. Tällä tavalla työkalun käyttöliittymästä saa selkeän. Käyttöliittymän tulisi olla tehty kauttaaltaan samanlaisella visuaalisella tyylillä, ja siinä tulisi käyttää yhdenmukaista värisuunnittelua ja typografiaa. Samantyyppisten elementtien tulisi näyttää ja käyttäytyä samalla

tavalla. Johdonmukaisuus on hyvä säilyttää suunnittelussa ja käyttäjien odotuksissa. Parhaimmillaan käyttäjä osaa odottaa mitä tulee saamaan tuotteesta. (Babich 2019.)

Vaikka on hyvä muistaa nämä neljä pääperiaatetta, tärkeintä on kuitenkin tehdä käyttöliittymästä sellainen, että sitä oikeasti pystyy käyttämään. Parhaiten autorigaustyökalun käyttöliittymän käytettävyyttä pystyisi testaamaan antamalla sen testattavaksi animaattoreille ja rigaajille.

4 Autorigaustyökalun rakentaminen ja esittely

Lähtiessäni rakentamaan omaa autorigaustyökaluani halusin sen ratkovan ensisijaisesti ajankäyttöön liittyviä ongelmia. Huomasin jatkuvasti rakentavani samanlaista pohjaa hahmoille. Vaikka perusrigi on melko yksinkertaista rakentaa, tekemisen prosessi vaatii tarkkuuta, ja se on altis virheille. Olemassa olevat autorigaustyökalut eivät täyttäneet tarpeitani rigauksen suhteen, ja mielsin niistä saadun rigin jatkokehittämisen itselleni vieraaksi. Halusin luoda pohjan, jonka päälle minun on helppo rakentaa, mutta jota voi myös käyttää sellaisenaan.

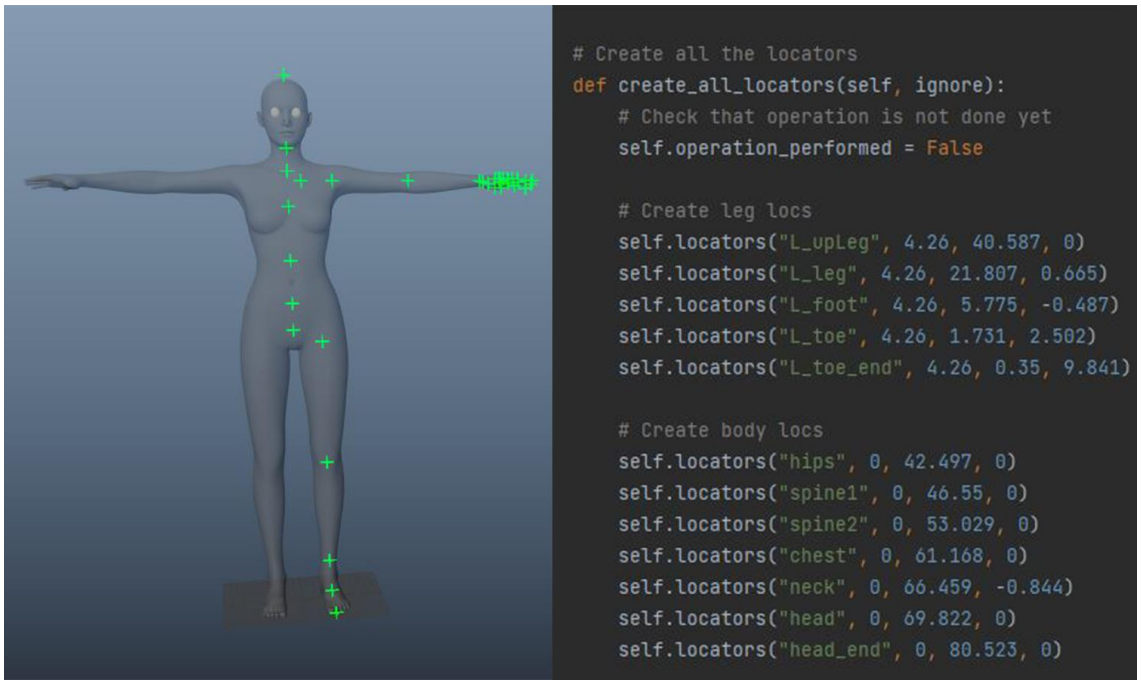
Aloitin työkalunkirjoittamisen ongelmanratkaisulla. Pohdin, mitä ominaisuuksia haluan ja miten ratkaista nämä koodin avulla. Päädyin luomaan ensin lokaattorit, jotka on nimetty luurangon luiden mukaan. Lokaattorit liikutetaan hahmon geometrian mukaan paikoilleen, jonka jälkeen nappia painamalla saadaan luotua luuranko. Tässä työvaiheessa luurangon ominaisuuksia voi vielä muuttaa tarvittaessa. Tämän jälkeen määritellään halutut värit kontrollirigin kontrolliobjekteille. Kontrolliobjektien värien määrittely on jaettu kolmeen osioon, jotka ovat keskikohta, vasen puoli ja oikea puoli. Värien valinnan jälkeen luodaan kontrollirigi, joka on valmis skinnattavaksi. Käytin rigin luonnin pohjalla mallihahmoa, joka oli mallinnettu T-asentoon.

4.1 Koodi

Suurimpia haasteita oli itse työkalun rakentaminen. Lähtötasoni koodaamisessa oli olematon. Päätin lähteä opettelemaan Pyhton-koodikieltä sen monikäyttöisyyden vuoksi. Alustavasti opettelin koodaamaan lyhyempiä skriptejä, joiden avulla koodaaminen Mayassa tuli tutuksi. Hyödynsin PyCharm-ohjelmistoa, jolla on hyvä yhteensopivuus Mayan kanssa. Tutustuin myös MEL-koodiin, mutta Pythonin ollessa laajemmin ohjelmoinnissa käytetty koodikieli, koin sen opettelun hyödyllisemmäksi. Käytin Pythonista versiota 3, jota Maya tukee.

4.2 Ominaisuudet

Aloitin koodin kirjoittamisen luomalla lokaattorit, jotka asettelin skriptiin mallihahmon mukaan. Lokaattorit tulevat määrittämään luiden sijainnit. Loin yksittäisen lokaattorin käyttämällä locators-komentoa, ja määrittelin lokaattorille nimen, sekä sen alustavat koordinaatit. Loin lokaattorit hahmon vartalolle päästä lantioon, sekä vasemman puolen käden ja jalan lokaattorit. Yksinkertaistaakseni koodaamista päätin, että autorigaustyökalulla voi luoda vain yhden rigin kerrallaan sceneen eli Mayan työtilaan. Varmistaakseni, että lokaattoreita luodaan vain yhdet kappaleet loin toiminnon, jolla tarkistetaan, onko operaatio tehty vai ei. Asetin operaation epätodeksi, ja jos operaatio suoritetaan, lokaattorit listataan. Luodaan ryhmä, ja lokaattorit asetetaan tähän ryhmään. Nyt lokaattorien sijaintia voi muuttaa vapaasti.



```
# Create all the locators
def create_all_locators(self, ignore):
    # Check that operation is not done yet
    self.operation_performed = False

    # Create leg locs
    self.locators("L_upLeg", 4.26, 40.587, 0)
    self.locators("L_leg", 4.26, 21.807, 0.665)
    self.locators("L_foot", 4.26, 5.775, -0.487)
    self.locators("L_toe", 4.26, 1.731, 2.502)
    self.locators("L_toe_end", 4.26, 0.35, 9.841)

    # Create body locs
    self.locators("hips", 0, 42.497, 0)
    self.locators("spine1", 0, 46.55, 0)
    self.locators("spine2", 0, 53.029, 0)
    self.locators("chest", 0, 61.168, 0)
    self.locators("neck", 0, 66.459, -0.844)
    self.locators("head", 0, 69.822, 0)
    self.locators("head_end", 0, 80.523, 0)
```

Kuva 7. Lokaattorit Mayassa ja koodissa.

Tehdäkseni työkalun käytöstä helpompaa päätin lokaattoreiden luonnin yhteydessä käyttää Mayan x-ray toimintoa, jolla geometriasta saa läpinäkyvän (Autodesk 2017). Näin lokaattorit on mahdollista asetella hahmolle kätevämmin. Koodissa tein toimminon nimeltä mesh_check, joka listaa scenen geometrian. Jos geometriaa löytyy scenestä, se valitaan ja muutetaan x-ray toiminnolla läpinäkyväksi.

```
# Check if there is mesh in the scene
def mesh_check(self, xray):
    mesh_list = cmds.ls(type="mesh")
    for mesh in mesh_list:
        if cmds.objExists(mesh):
            # Select mesh
            cmds.select(mesh_list)
            # Turn on xray to see joints
            cmds.displaySurface(xRay=xray)
```

Kuva 8. Mesh_check toiminto koodissa.

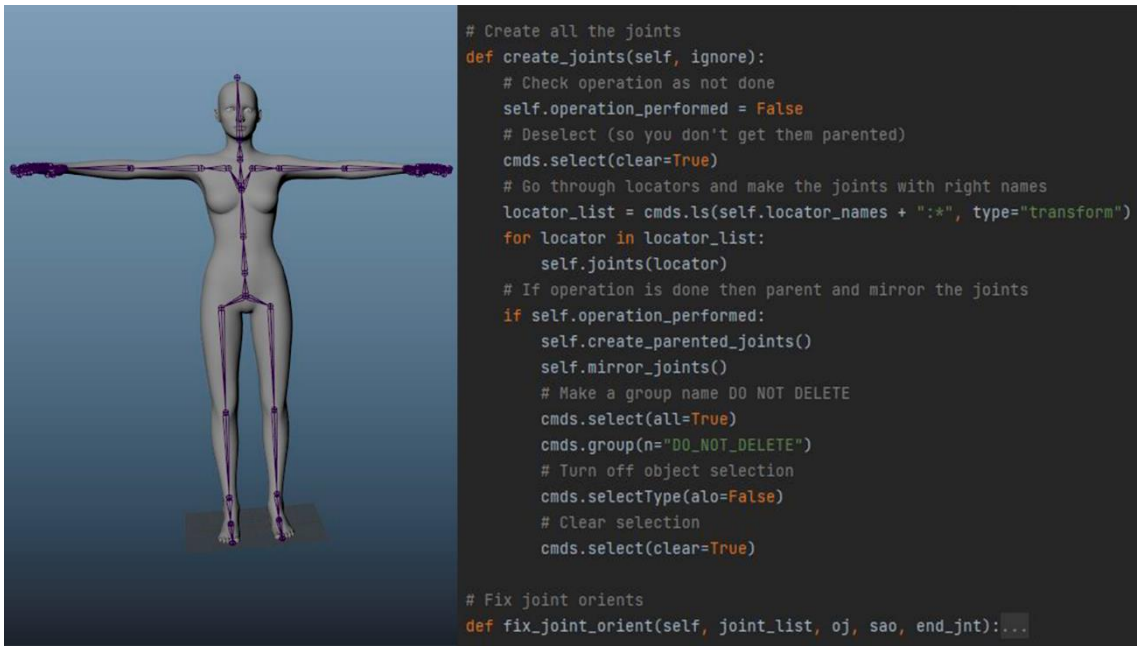
Luiden luomisessa aloitin tekemällä yksittäisen luun positioon 0, 0, 0. Määritin luun ottamaan nimen lokaattorin mukaan. Erotin lokaattorin päätteen ja vaihdoin tilalle luun nimen. Määritin luiden syntymisen tapahtumaan vain silloin, kun niitä

ei vielä löydy scenestä. Lisäsin luun luiden listaan. Lopuksi määritin luun positioksi lokaattroin uuden position, ja merkitsin operaation todeksi eli tapahtuneeksi.

```
# Make a joint
def joints(self, name):
    # Clear selection to prevent parenting
    cmds.select(clear=True)
    # Make the joint names based on locators but make sure not to have locator suffix
    sections = name.split(":")
    joint_name = sections[-1]
    sections = joint_name.split(self.LOCATOR_SUFFIX)
    joint_name = "".join(sections[:-1])
    # Add the jnt suffix
    joint_name = joint_name + "_jnt"
    # Make sure you can only create joints when they do not exist in the scene
    if not cmds.objExists(joint_name):
        # Get locators new position
        locpos = cmds.pointPosition(name)
        # Create a joint in pos 0
        joint = cmds.joint(p=(0, 0, 0), n=joint_name)
        # Add to the joint list
        self.joint_list.append(joint)
        # Select the joint
        cmds.select()
        # Move the joint to locator's position
        cmds.move(locpos[0], locpos[1], locpos[2])
        # check operation done
        self.operation_performed = True
```

Kuva 9. Yhden luun määrittely koodissa.

Luodakseni kaikki luut käyn läpi kaikki lokaattorit ja suoritan yksittäisen luun luomisen jokaisen kohdalla. Kun luut ovat luotu parentoin luut oikeisiin luihin muodostaen vartalon, vasemman jalan, vasemman käden ja sen sormet. Lopuksi parentoin sormien luut käden luuhun, olkapään luun rintaluuhun ja jalkojen luut lantion luuhun. Jokaisen ketjun parentoinnin jälkeen korjaan luiden orientaatiot vastamaan haluttuja orientaatioita. Orientaatioiden korjaamisen jälkeen useimmissa luissa x-akseli on määritelty osoittamaan seuraavaan luuhun, y-akseli ylöspäin ja z-akseli sivulle. Lopuksi peilaan käsien ja jalkojen luut oikealle puolelle. Näin saan luotua rigin luurangon oikeilla luiden nimillä ja orientaatioilla. Luiden luonnin yhteydessä kytken x-ray toiminnon pois, ja scenestä löytyvä geometria piirtyy normaalisti.



```
# Create all the joints
def create_joints(self, ignore):
    # Check operation as not done
    self.operation_performed = False
    # Deselect (so you don't get them parented)
    cmds.select(clear=True)
    # Go through locators and make the joints with right names
    locator_list = cmds.ls(self.locator_names + ":*", type="transform")
    for locator in locator_list:
        self.joints(locator)
    # If operation is done then parent and mirror the joints
    if self.operation_performed:
        self.create_parented_joints()
        self.mirror_joints()
        # Make a group name DO NOT DELETE
        cmds.select(all=True)
        cmds.group(n="DO_NOT_DELETE")
        # Turn off object selection
        cmds.selectType(alo=False)
        # Clear selection
        cmds.select(clear=True)

# Fix joint orients
def fix_joint_orient(self, joint_list, oj, sao, end_jnt):...
```

Kuva 10. Luut Mayassa ja mm. niiden position määrittely koodissa.

Luuston luonnin jälkeen luita pystyy vielä muokkaamaan tarvittaessa. Kontrollirigin luominen perustuu luiden positioon, ja tämä tulisi luoda vasta kun luut ovat varmasti halutuilla paikoillaan. Aloitan kontrollirigin rakentamisen määrittämällä listan kontrollien nimistä. Välttääkseni päällekkäisyyksiä varmistan, että kontrollirigi luodaan vain silloin kun sitä ei ole luotu sceneen aiemmin. Tarkistan myös että luut löytyvät scenestä, sillä ilman luita kontrollirigiä ei luoda. Aloitin luomalla root- ja master-kontrolliobjektit, jotka ovat ympyrän muotoisia NURBS-käyrä objekteja. NURBS-käyrät ovat avaruuskäyriä, joiden muoto määritellään matemaattista NURBS-mallia käyttäen (Joensuu 2016).

Seuraavaksi siirryn jalan kontrolliobjektien luomiseen, mutta sitä ennen tarvitsen jalan IK-ketjut. Määritän IK-ketjut vasemmalle ja oikealle jalalle. Luon IK-ketjun upLeg-luusta Leg-luuhun, asetan sen solveriksi Rotae Plain IK solverin. Määritän toisen IK-ketjun foot-luusta toe-luuhun ja kolmannen toe-luusta toe_end-luuhun. Näille asetan solveriksi Single Chain IK solverin ja nimeän ne. Tämän jälkeen muutan jalkojen luiden pivotit attribuuttien tekoa varten. Määritän ensin pivotin muuttamisen yleisesti, jonka jälkeen määritän jokaisen jalan luun uuden pivotin sijainnin. Jalan pivotin muuttaminen tapahtuu ryhmittämällä valittu objekti

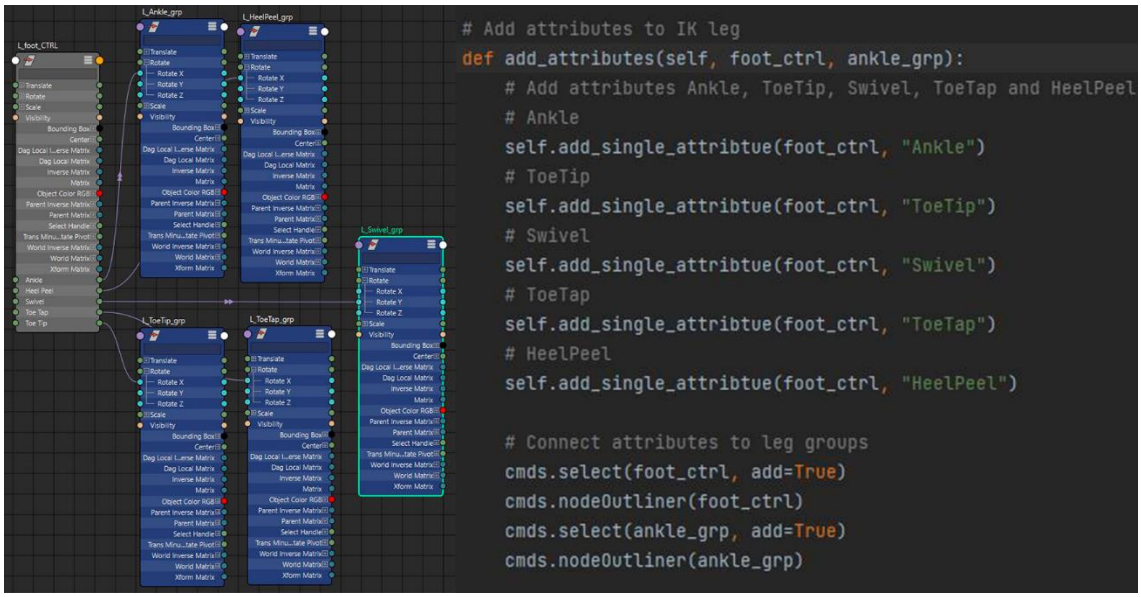
ja antamalla ryhmälle nimi. Ryhmän pivot muutetaan move-kommennolla vastaamaan tietyn luun sijaintia.

```
# Change single pivot
def change_single_pivot(self, items, grp, jnt_pos):
    cmds.group(items, n=grp)
    cmds.move(jnt_pos[0], jnt_pos[1], jnt_pos[2], grp + ".scalePivot", grp + ".rotatePivot", rpr=True)
    cmds.snapMode(p=True)

# Change pivot points in legs
def change_pivots(self, ball_ik, toe_ik, ankle_ik, toetap_grp, heelpeel_grp, swivel_grp, toetip_grp, ankle_grp,
                  toe_jnt, toe_end_jnt, foot_jnt):
    # Get toe_jnt position
    toe_jnt_pos = cmds.pointPosition(toe_jnt + ".rotatePivot")
    # Define things to group
    items = ball_ik, toe_ik,
    # Change pivot to Toe_jnt
    self.change_single_pivot(items, toetap_grp, toe_jnt_pos)

    # Change pivot to Toe_jnt
    self.change_single_pivot(ankle_ik, heelpeel_grp, toe_jnt_pos)

    # Define things to group
    items = toetap_grp, heelpeel_grp,
    # Change pivot to Toe_jnt
    self.change_single_pivot(items, swivel_grp, toe_jnt_pos)

    # Get toe_end_jnt position
    toe_end_jnt_pos = cmds.pointPosition(toe_end_jnt + ".rotatePivot")
    # Change pivot to Toe_end_jnt
    self.change_single_pivot(swivel_grp, toetip_grp, toe_end_jnt_pos)

    # Get foot_jnt position
    foot_jnt_pos = cmds.pointPosition(foot_jnt + ".rotatePivot")
    # Change pivot to foot_jnt
    self.change_single_pivot(toetip_grp, ankle_grp, foot_jnt_pos)

    # Turn off snapMode
    cmds.snapMode(p=False)
```

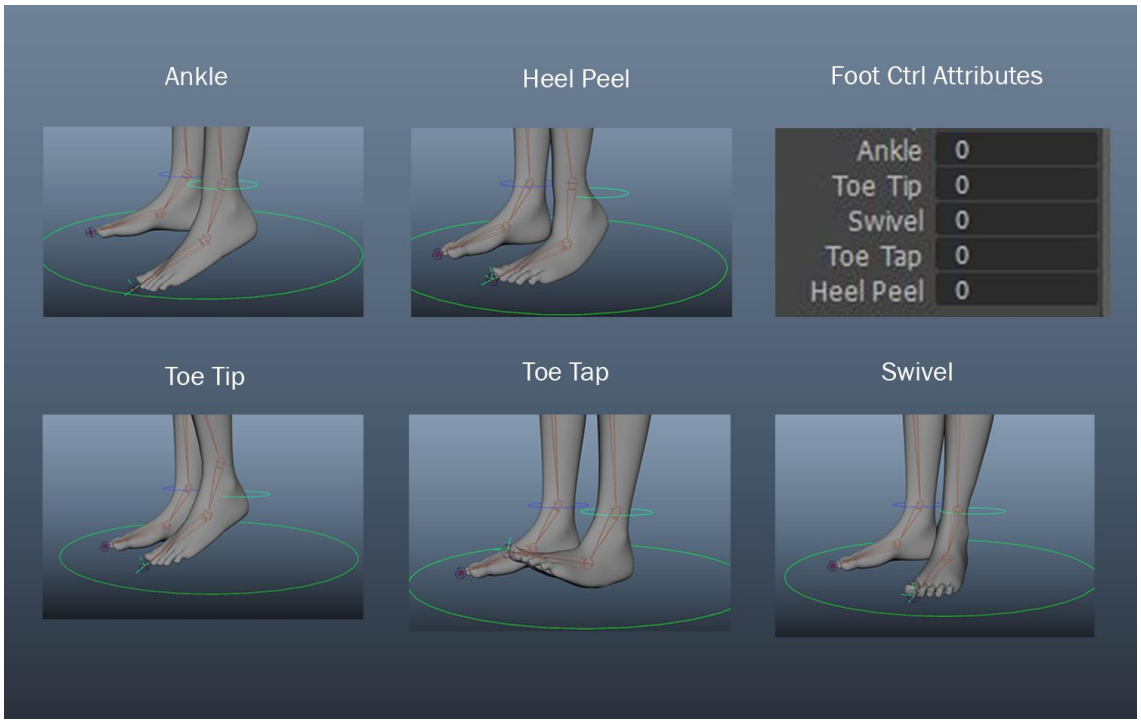
Kuva 11. Jalan pivotin muuttaminen koodissa.

Tämän jälkeen siirryn luomaan jalan kontrolliobjektit. Teen kontrolliobjekteille ryhmän, jonka laitan samaan positioon foot-luun kanssa. Luon polville oman kontrolliobjektinsa, jonka nimeän knee_ctrl ja ryhmitettyäni sen siirrän ryhmän samaan sijaintiin leg-luun kanssa. Liikutan polven kontrollin ryhmää hieman eteenpäin, jotta polven kontrollia on helpompi liikuttaa. Tämän jälkeen luon Pole Vector constraintin, jolla saadaan IK-ketju seuraamaan polven kontrolliobjetkia. Lopuksi parentoin jalan ja polven kontrolliobjektien ryhmät master-kontrolliobjektiin.

Seuraavaksi siirryn luomaan attribuutit jaloille. Attribuuteilla saadaan aikaseksi jalan liikuttaminen sujuvasti. Määritän ensin sen, kuinka tehdään yksittäinen lisättävä attribuutti jalan kontrolliobjektille. Tämän jälkeen määritän kaikki tarvittavat attribuutit, jotka nimeän seuraavasti: Ankle, ToeTip, Swivel, ToeTap ja HeelPeel. Yhdistän jokaisen attribuutin oikeaan rotaation suuntaan. Täten attribuutin muuttaminen konrolliobjektista liikuttaa haluttua kohtaa jalassa.

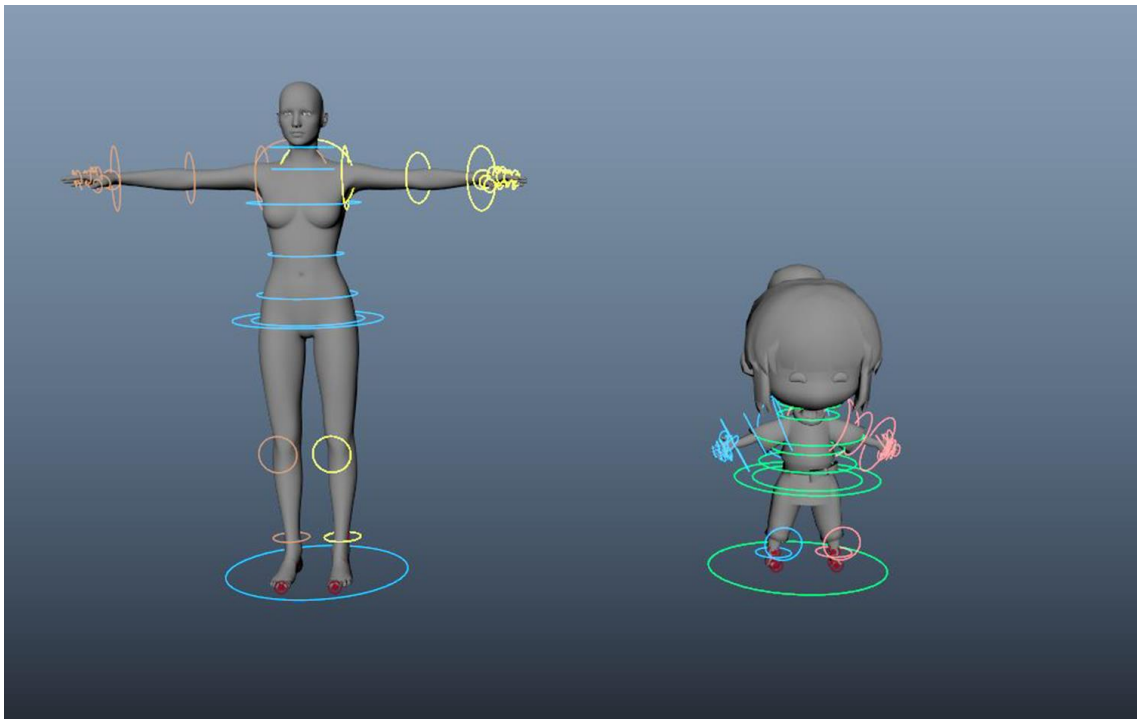


Kuva 12. Attribuuttien yhteydet Mayassa ja niiden luominen koodissa.



Kuva 13. Attribuuttien esittely Mayassa.

Attribuuttien luomisen ja yhdistämisen jälkeen siirryn luomaan vartalon kontrolliobjektit. Kontrolliobjektien luominen noudattaa samaa kaavaa kaikissa rigin osissa. Ensin luodaan NURBS-käyrä maailman positioon 0,0,0, joka nimetään ja ryhmitetään. Ryhmän sijainti muutetaan halutun luun kanssa samaksi, jonka jälkeen luodaan parent constraint kontrolliobjektin ja luun välille. Näin luu saadaan seuraamaan kontrolliobjektia. Toistan saman käsille ja sormille. Lopuksi parentoin olkapään ryhmän rinnan kontrolliobjektiin ja sormien ryhmät käden kontrolliobjektiin.



Kuva 14. Kontrollirigi kahdella eri hahmolla.

4.3 Käyttöliittymä

Valmiiden autorigaustyökalujen käyttö vaatii opettelua ja tutoriaalien katsomista ja tahdoin oman työkaluni olevan intuitiivinen ja helppokäyttöinen. Tarkoituksena oli tehdä työkalu, jonka avaamalla tietäisi mitä tehdä. Työkalun intuitiivisen käytön saavuttamiseksi, sen suunnittelu vaati myös käyttöjärjestelmän suunnitteluun perehtymistä. Käyttöliittymäsuunnittelu on erittäin laaja aihe ja käsittelen tässä luvussa sitä vain sen osalta, mitä itse tein autorigaustyökaluuni.

Tein käyttöliittymästä yksinkertaisen. Helpottaakseni käyttöä jokaisessa painikkeessa lukee, mitä ne tekevät ja niiden yläpuolella ohjeistetaan miten toimia.

```
# ---------------------------------------------
# UI
# Define spacing for UI
self.spacing = 2
# Define separator
cmds.separator(p=self.column_layout, h=self.spacing, style="none")
# Add text for placing the locators
cmds.text(l="Place locators and move them to wanted locations.")
# Make button to create locators
cmds.button(l="Place locators", p=self.column_layout, c=self.create_all_locators)
# Make button to hide locators
cmds.button(l="Toggle locators visibility", p=self.column_layout, c=self.toggle_locators)
# Separator
cmds.separator(p=self.column_layout, height=10, style="in")
# Add text for creating joints
cmds.text(l="Press to create joints based on locator positions.")
# Make button to create joints
cmds.button(l="Create joints", p=self.column_layout, c=self.create_joints)
cmds.separator(p=self.column_layout, height=10, style="in")
# Add text to choose the control colors
cmds.text(l="Choose control colors:")
# Add slider to choose colors for controls
self.controller_color_input1 = cmds.colorIndexSliderGrp(p=self.column_layout, min=4, max=24,
                                                        l="Body: ")
cmds.separator(p=self.column_layout, h=self.spacing, style="none")
self.controller_color_input2 = cmds.colorIndexSliderGrp(p=self.column_layout, min=4, max=24,
                                                        l="Left side: ")
cmds.separator(p=self.column_layout, h=self.spacing, style="none")
self.controller_color_input3 = cmds.colorIndexSliderGrp(p=self.column_layout, min=4, max=24,
                                                        l="Right side: ")
cmds.separator(p=self.column_layout, h=self.spacing, style="none")
# Make button to create controls
cmds.button(l="Create controls", p=self.column_layout, c=self.create_controls)
```

Kuva 15. UI koodissa.

Tein painikkeen lokaattoreiden asettamiselle. Tehdäkseni työkalun käytöstä entistä miellyttävämpää, ja päätin antaa käyttäjälle mahdollisuuden piilottaa lokaattorit halutessaan. Lokaattoreita ei periaatteessa tarvitse enää luiden luomisen jälkeen. Luodakseni hallinnan tunnetta käyttäjäkokemukseen päätin, että käyttäjä voi itse säätää lokaattorit joko piilotetuksi tai näkyväksi.

Koodissa toteutin tämän luomalla funktion, jossa tarkastetaan löytyykö scenestä lokaattoreiden ryhmää, joka aiemmin on määritelty saamaan nimen lokaattoreiden ryhmän luonnin yhteydessä. Jos tällainen ryhmä löytyy, sen näkyvyyttä voi muuttaa.

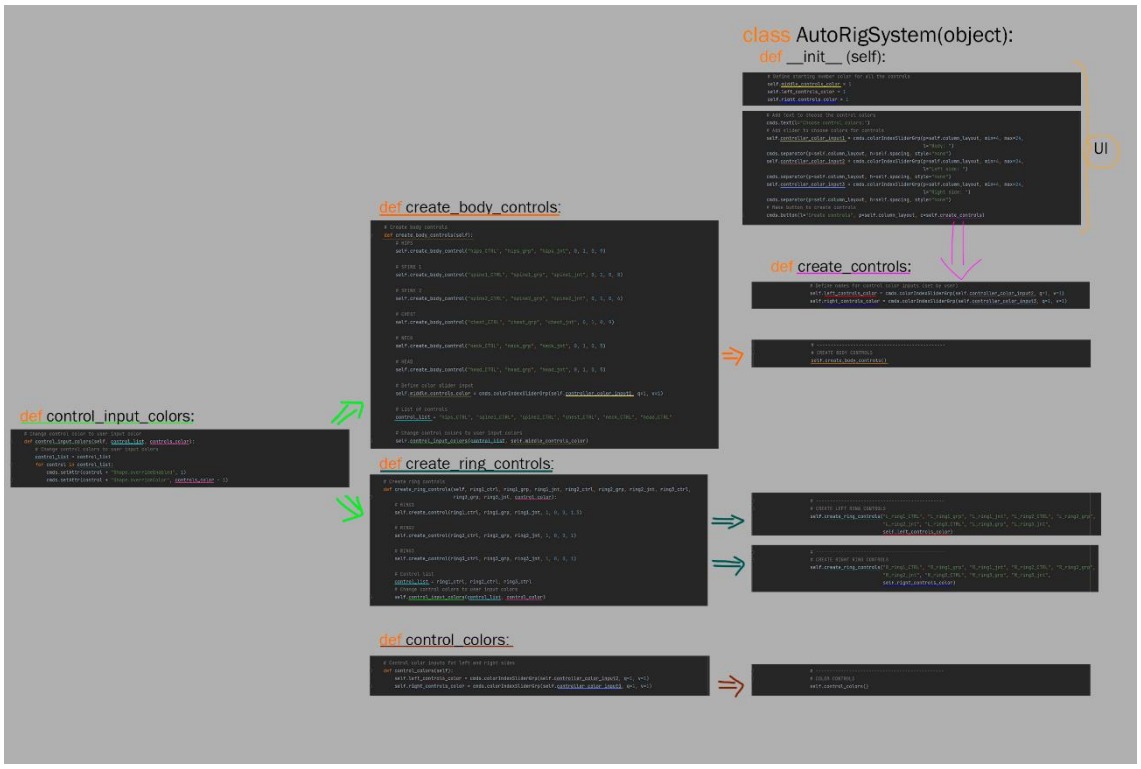
```
    # Toggle locators visibility
    def toggle_locators(self, ignore):
        # Check if group exist, if it does, set visibility
        loc_grp = self.locator_group
        if cmds.objExists(loc_grp):
            cmds.setAttr(self.locator_group + '.visibility', not cmds.getAttr(self.locator_group + '.visibility'))
```

Kuva 16. Näkyvyyden muuttaminen koodissa.

Painikkeesta ”Create joints” luodaan luut. Kontrolliobjektien luomiseen käytettävän painikkeen taakse sisällytin kaiken muun rigiin liittyvän, kuten IK-ketjut, attribuuttien tekemisen, itse kontrolliobjektien luomisen ja constraintien tekemisen. Ennen tätä painiketta ohjasin käyttäjää valitsemaan kontrolliobjektien värit vartalolle, oikealle ja vasemmalle puolelle tekstin avulla. Kontrolliobjektien värien valinnaksi valitsin liukusäätimet, joista pystyy valita Mayan Index väreistä itselleen mieluisimmat. Määrittelen koodissa keskellä olevien vartalon kontrolliobjektien ja oikean ja vasemman puolen kontrolliobjektien värit alustavasti numeroon 1, mikä on harmaa. Koodin käyttöliittymässä annan käyttäjän syöttää halutun värin, joka vastaa tiettyä Index numeroa koodissa. Tämän toiminnon rakentaminen käyttöliittymään oli yksi haastavimmista vaiheista.

Annoin käyttäjän syöttää halutut värit kontrolliobjekteille. Laitoin työkalun käyttöliittymään painikkeen ”Create controls”, josta kontrolliobjektit luodaan. Funktiossa create_controls määritellään ensin nimet vasemman ja oikean puolen kontrolliobjektien väreille käyttäjän syötteen mukaan. Näitä käytetään myöhemmin samassa funktiossa, kun määritellään vasemman ja oikean puolen kontrolliobjekteja, joille halutaan määrittää tarkka väri. Keskivartalon kontrolliobjektien väri määritellään omassa funktiossaan, joka kutsutaan create_controls-funktiossa. Create_body_controls-funktiossa luodaan ensin kontrolliobjektit kutsumalla aiemmin luotua funktiota create_body_control, jossa luodaan yksittäinen vartalon kontrolliobjekti, ja määrittämällä kontrolliobjekteille tarkemmat nimet ja muodot. Tämän jälkeen asetetaan käyttäjän antama vartalon kontrolliobjektien väri. Ensin määrittämällä käyttäjän syöttö, ja sitten listaamalla kontrolliobjektit listaan nimeltä control_list. Tämän jälkeen kutsutaan funktiota control_input_co-

lors, jossa määritellään koodi käymään läpi kontrolliobjektien lista, ja muuttamaan niiden väri halutuksi väriksi. Create_body_controls-funktiossa haluttu väri määritellään olevan käyttäjän syöttämä väri.



Kuva 17. Käyttäjän valitsemien kontrolliobjektien värien määrittäminen koodissa.

Erotin lokaattoreihin, luihin ja kontrolliobjekteihin liittyvät toiminnot viivalla, joka tehdään separator-komennolla. Tässä komennossa tulee määritellä sarakkeen asettelu, joka tehdään rowColumnLayout-toiminnolla, jossa määritellään muun muassa sarakkeen määrä, leveys ja tila sarakkeiden välillä pikseleissä. Sen li-

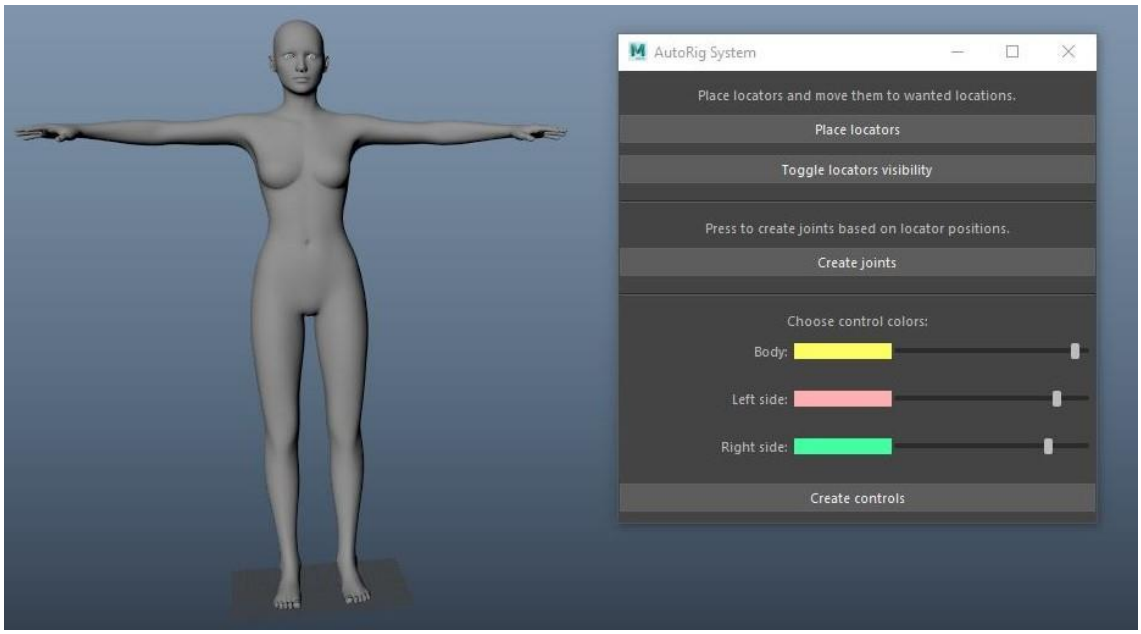
säksi separator-komennossa määritellään viivan korkeus, jonka määritin aikaisemmin koodissa luvuksi 2, ja nimesin muuttujan *self.spacing*. Lopuksi valitsin viivan tyyliksi ”none”.

```
# Define spacing for UI
self.spacing = 2
# Define separator
cmds.separator(p=self.column_layout, h=self.spacing, style="none")
```

Kuva 18. Erottajan komento koodissa.

```
# Define column layout
self.column_layout = cmds.rowColumnLayout(nc=1, w=200, rs=(1, 10))
```

Kuva 19. Sarakkeen asettelu, johon erottaja perustuu.



Kuva 20. Autorigaustyökalu avattuna Mayassa.

4.4 Koodin turvaaminen

Lopuksi halusin varmistaa, ettei työkaluun tule virhettä, johon koodi ei osaisi vastata ja jonka takia työkalu sulkeutuisi. Päätin ohjata käyttäjää selkeiden tekstien avulla. Kun työkalussa painetaan ”Place locators” -painiketta lokaattorien luomiseksi. Se antaa tekstin omassa ikkunassaan, joka kertoo että käyttäjä voi

liikuttaa lokaattoreja, muttei poistaa niitä. Käyttäjän tulee hyväksyä viesti luetuksi, jotta päästään eroon tästä ikkunasta. Loin tämän käyttämällä confirmDialog-komentoa.

```
        # If operation is done make the locator group
        if self.operation_performed:
            locators = cmds.ls(type="locator", l=True)
            cmds.group(locators, n=self.locator_group)
            # Inform user to not delete locators
            cmds.confirmDialog(title='Confirm', message="You can move locators. Do not delete them.", button="OK",
                               defaultButton="OK", cancelButton="OK", dismissString="OK")
            # Clear selection
            cmds.select(clear=True)
            # Clear selection type
            cmds.selectType(alo=False)
            # Make sure that only locators can be selected
            cmds.selectType(lc=True)
            self.mesh_check(True)
            # Clear selection
            cmds.select(clear=True)
```

Kuva 21. Ohjeiden luominen koodissa ja komento sille, että vain lokaattorit ovat valittavissa scenessä.

Osittain helpottaakseni käyttöä, ja osittain estämästä käyttäjää rikkomasta koodia, säädin sitä, mitä käyttäjä voi milloinkin scenestä valita. Lokaattoreiden luonnin jälkeen scenestä voi valita vain lokaattorit.

Luiden luonnin jälkeen scenestä mikään objekti ei ole valittavissa. Tässä kohtaa ei ole tarvetta koskea scenessä mihinkään. Halusin säästää itselleni tämän työvaiheen yhdeksi työkalun osaksi tulevaisuuden jatkokehitystä varten. Luut ryhmitetään ja ryhmästä käytetään nimeä DO_NOT_DELETE, eli käyttäjää käsketään olemaan poistamatta mitään tämän ryhmän sisältä. Kontrollirigin luotua valittavissa on vain kontrolliobjektit.

5 Yhteenveto

5.1 Prosessin ongelmakohdat ja loppupäätelmät

Autorigaustyökalun rakentaminen oli kaiken kaikkiaan palkitseva ja opettava projekti. Vaikean alun jälkeen innostus koodaamista kohtaan kasvoi ja työkalun rakentaminen oli hyvin koukuttavaa.

Projektin alussa koodaamisosaamisen taso oli itselläni matala. Minulta meni pitkään ymmärtää erilaiset silmukat ja kirjaston käyttäminen Maya-komentojen osalta. Koodaaminen on ennen kaikkea ongelman ratkomista ja ongelman purkamista mahdollisimman pieniin osiin. Tärkeintä oli löytää oikeanlainen logiikka ja olla ajattelematta liian monimutkaisesti. Useimpiin ongelmiin ratkaisu oli yksinkertainen. Pythonin opettelussa yksi isoimpia haasteitani oli käskyjen käytön ymmärtäminen. Käskyt vaativat Mayan työkalujen ja toimintojen syvällistä tuntemusta. Koodaaminen auttoi ymmärtämään käyttämieni käskyjen merkitystä paremmin.

Työkalun rakentamiseen haasteita loi myös jatkuva oppiminen ja rajallinen aika. Projektin alussa en osannut koodata ja lopussa koodaamisen taitoni olivat huomattavasti kehittyneet. Aloitin työkalun rakentamisen vajavaisilla taidoilla ja rakensin työkaluni pohjan sen mukaan. Myöhemmin kun opin lisää, oli työlästä lähteä muuttamaan koko koodia sulavammaksi. Minulla oli myös rajallinen aika projektin tekemiseen, joten jouduin tekemään joitain kompromisseja koodin suhteen. Muun muassa olisin halunnut, että käyttäjä pystyy halutessaan nimeämään luut ja kontrolliobjektit. Koodin kirjoittaminen niin olisi ollut järkevämpää ja lopulta se olisi helpottanut myös omaa työmäärääni.

Myös joidenkin Maya komentojen muuttaminen Python-koodiksi oli työlästä. Osa Pythonin ja MEL:n komennoista oli hyvin dokumentoituja, mutta osaa oli aluksi vaikeaa ymmärtää. Tähän vaikutti myös oma osaamiseni taso. Projektin lopussa olisin varmasti onnistunut ratkaisemaan joitain ongelmia, joihin en enää pystynyt käyttämään enempää aikaa. Tällaisista ongelmista suurin oli NURBS-

käyrien muodon muuttaminen. En päässyt käsiksi muotoon verteksi-pisteiden tasolla, jolloin päädyin jättämään kontrolliobjektien muodon alkuperäiseksi ympyräksi.

Yllättävä haaste oli tehdä työkalusta ja koodista sellainen, ettei se menisi rikki käytettäessä. Projektia aloittaessani ajatukseni olivat ainoastaan siinä, kuinka saisin kaikki ominaisuudet toimimaan. Päätin jättää koodin suojaamisen viimeiseksi tehtäväksi, jos aikaa sille jäisi. Lopulta se tuntui kuitenkin niin tärkeältä ominaisuudelta, että päätin työstää sen kuntoon. Sain myös tärkeimmät ominaisuudet toimimaan nopeammin mitä olin kuvitellut.

Projektin alussa olin varma siitä, että joutuisin katsomaan useita tutoriaaleja koodin kirjoittamista varten. Aloittaessani kirjoittamisen päätin kuitenkin tehdä koodin niin pitkälle itsenäisesti kuin vaan pystyisin. Lopulta päädyin kirjoittamaan koodin katsomatta yhtään tutoriaalia. Olin hakenut esimerkkejä autorigaustyökaluista ja siitä miltä ne näyttävät, mutta en ollut katsonut miten ne oli tehty. Jälkikäteen ajateltuna, olisi ehkä ollut hyvä katsoa pari esimerkkiä siitä, miten koodi on muotoiltu muissa projekteissa. Itselleni tämä oli kuitenkin kaikista opettavin tapa, ja olen tyytyväinen siten tehtyyn lopputulokseen.

Suurimmaksi edukseni opinnäytetyötä kirjoittaessani osoittautui koodin selkeä ja tarkka kommentointi. Projektin tekemisen ja opinnäytetyön kirjoittamisen välillä oli noin puoli vuotta aikaa, joten pelkäsin unohtaneeni joitain osia koodista. Yllättävää oli kuitenkin se, kuinka nopeasti tekemäni asiat palautuivat mieleeni, ja kuinka suuresti aikaisemmin tehty tarkka kommentointi auttoi tässä.

Kaiken kaikkiaan koin oman autorigaustyökalun rakentamisen hyödylliseksi sekä koodin opettelun osalta, että nopeamman rigausprosessin aikaan saamiseksi. Pidin myös siitä, että omaa työkalua tehtäessä pystyin vaikuttamaan siihen, mitä ominaisuuksia rigillä on. Koin hyödylliseksi omien mieltymysten ja tärkeäksi koettujen ominaisuuksien implementoinnin autorigaustyökaluun.

5.2 Miten parantelisin työkalua

Autorigaustyökalustani löytyisi paljon paranneltavaa ja minulla on paljon ideoita lisättäviin ominaisuuksiin. Ensisijaisesti parantelisin koodia ja antaisin käyttäjälle mahdollisuuden nimetä luut ja kontrolliobjektit halutessaan. Lokaattoreiden asettamisen helpottamiseksi antaisin käyttäjälle mahdollisuuden valita A -ja T-asennon välillä, jonka mukaan lokaattorit ilmaantuisivat sceneen. Nopeuttaakseni rigaajan työtä entisestään sisällyttäisin mahdollisuuden tuoda rigattava hahmo autorigaustyökalun kautta Mayaan. Lisäisin käsiin IK-ketjut ja antaisin käyttäjälle mahdollisuuden vaihtaa FK-kädet IK-käsiksi ja takaisin. Lisäisin vartalon, käsien ja jalkojen luihin mahdollisuuden litistämiseen ja venyttämiseen, lisätäkseni rigin käytön monipuolisuutta. Käsien luonnollisemman kierron saavuttamiseksi lisäisin mahdollisuuden lisätä ylimääräisiä luita. Myös selkärankaan voisi valita useamman selkärangan luun. Tällaisenaan autorigaustyökalusta uupuu kasvojen rigi, jonka haluaisin lisätä tulevaisuudessa työkaluun.

Lähteet

Aalto-yliopisto 2022. Python oppimateriaali. [verkkokurssi]. MyCourses Aalto-yliopisto. Saatavuus https://mycourses.aalto.fi/mod/book/view.php?id=442395&chapterid=2485 (luettu 15.11.2022).

Adobe 2022. Mixamo. [verkkosivu] Adobe. Saatavuus https://www.mixamo.com/ (luettu 1.11.2022).

Ahlroth, Jussi 6.7.2007. Mitä on liikkeenkaappaus? [verkkodokumentti]. Helsingin Sanomat. Saatavuus https://www.hs.fi/kulttuuri/art-2000004495007.html?share=70262dcd5ba8a0dd7190611657bd8e7f (luettu 20.11.2022).

Autodesk 11.6.2017. Change the display mode for objects. [verkkodokumentti]. Autodesk. Saatavuus https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/Maya/files/GUID-E8487A83-8786-4840-B13F-0392812A59B9-htm.html (luettu 13.11.2022).

Autodesk 7.12.2020. Python in Maya. [verkkodokumentti]. Autodesk. Saatavuus https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-Scripting/files/GUID-C0F27A50-3DD6-454C-A4D1-9E3C44B3C990-htm.html (luettu 19.11.2022).

Autodesk 2021a. HumanIK. [verkkodokumentti]. Autodesk. Saatavuus https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-CharacterAnimation/files/GUID-EDBDA3DB-4715-40EF-9ADF-412F78BFF98E-htm.html (luettu 3.11.2022).

Autodesk 2021b. IK Handles overview. [verkkodokumentti]. Autodesk. Saatavuus https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-CharacterAnimation/files/GUID-8AFE1FEB-C69C-4841-9787-3C196FDA9A1F-htm.html (luettu 29.8.2022).

Autodesk 2021c. Quick Rig tool. [verkkodokumentti]. Autodesk. Saatavuus https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-CharacterAnimation/files/GUID-DC29C982-D04F-4C20-9DBA-4BBB33E027EF-htm.html (luettu 3.11.2022).

Assaf, Eyal 2015. Rigging for Games: A Primer for Technical Artists Using Maya and Python. Iso-Britannia: Routledge.

Babich, Nick 7.10.2019. The 4 Golden Rules of UI Design. [verkkodokumentti]. Adobe. Saatavuus https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/ (luettu 9.11.2022).

Fontini Publishing house 29.9.2021. Käsivarret ja kädet. [verkkodokumentti]. If. Saatavuus https://www.if.fi/vakuutukset-paasta-varpaisiin/opi-tuntemaan-kehosi/kasivarsi-kasi (luettu 16.11.2022).

Joensuu Janne 2016. 3D-alan sanasto. [pdf]. Opinnäytetyö. Tietojenkäsittelyn koulutusohjelma. Kajaani: Kajaanin Ammattikorkeakoulu. Saatavuus https://www.theseus.fi/bitstream/handle/10024/113462/Joensuu_Janne_3D-alan_sanasto.pdf?sequence=1 (luettu 17.11.2022).

Kivelä, Erno 2022. Luokka/Olio. [verkkodokumentti] Matikki. Saatavuus https://www.matikki.fi/guidebook-python/python/guide-basics/luokka.html (luettu 20.11.2022).

Maraffi, Chris 2009. MEL Scripting a Character Rig in Maya. Kalifornia: New Riders.

Mayo Clinic 2022. Arm Bones. [verkkodokumentti]. Mayo Clinic. Saatavuus https://www.mayoclinic.org/diseases-conditions/broken-arm/multimedia/arm-bones/img-20007018 (luettu 29.8.2022).

Mechtley Adam, Trowbridge Ryan 2011. Maya Python for Games and Film: A Complete Reference for the Maya Python API. Yhdysvallat: CRC Press.

O'Hailey Tina 2018. Rig it Right! Maya Animation Rigging Concepts, 2nd edition. Iso-Britannia: Routledge.

Omeyer Alex 5.4.2022. 10 Must-know patterns for writing clean code with Python. [verkkodokumentti]. Dev. Saatavuus https://dev.to/alexomeyer/10-must-know-patterns-for-writing-clean-code-with-python-56bf (luettu 18.11.2022).

Peda.net 2022. Johdatus Pythoniin ja kilpikonnagrafiikkaan. [verkkodokumentti]. Peda.net. Saatavuus https://peda.net/jyvaskyla/ict/palvelut/ohjelmointi-robotiikka/pop/l7fl (luettu 22.11.2022).

Richter, Alexander 2.9.2017. Scripting for Artists in Maya. [verkkovideo]. Youtube. Saatavuus https://www.youtube.com/watch?v=82ZYwdL_o-k (katsottu 2.11.2022).

Saikkonen, Riku 23.9.2011. Ohjelmmoinnin peruskurssien laaja oppimäärä. [verkkodokumentti] Aalto. Saatavauus https://wiki.aalto.fi/download/attachments/58943799/eluento.pdf?version=1&modificationDate=1316783571000&api=v2 (luettu 22.11.2022).

ScreenSkills 15.6.2021. An Introduction to 3D animation rigging. [verkkovideo]. Youtube. Saatavuus https://www.youtube.com/watch?v=1i65a1Z-jHQ (katsottu 29.8.2022).

Selkäkanava 2022. Selän rakenne ja toiminta. [verkkodokumentti]. Selkäliitto. Saatavuus https://selkakanava.fi/selkakipu/selan-rakenne-ja-toiminta (luettu 16.11.2022).

Sievanen, Elina 2020. Johdanto modulaariseen rigaukseen. [pdf]. Opinnäytetyö. Viestinnän koulutusohjelma. Helsinki: Metropolia. Saatavuus https://www.theseus.fi/bitstream/handle/10024/340927/sievanen_elina.pdf?sequence=2&isAllowed=y (luettu 30.8.2022).

W3 Schools 2022. Python tutorial. [verkkodokumentti] W3 Schools. Saatavuus https://www.w3schools.com/python/default.asp (luettu 22.11.2022).

Kuvalähteet

Kuva 1. Drazen 2020. Saatavuus https://sketchfab.com/drazenk

Kuva 14. Jade L 2017. Saatavuus https://sketchfab.com/3levan