SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

PELINKEHITYKSEN ELIN-KAARI UNREAL ENGINELLÄ

TEKIJÄ/T: Sami Kimpimäki
Panu Puurunen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Sami Kimpimäki, Panu Puurunen	
Työn nimi Pelinkehityksen elinkaari Unreal Enginellä	
Päiväys 22.05.2018	Sivumäärä/Liitteet 38
Ohjaaja(t) Mikko Pääkkönen, TKI-asiantuntija	
Toimeksiantaja/Yhteistyökumppani(t)	

Tiivistelmä

Opinnäytetyö toteutettiin tekijöiden käyttöön tulevaisuuden suunnitelmia varten, tarkoituksena oli aloittaa pelinkehitysprosessi ja samalla tutustua, sekä opetella useat pelinkehitykseen liittyvät vaiheet. Ideana oli luoda 2d mobiilipeli, jossa pelaaja läpäisee kentän kerrallaan, päihittäen viholliset naputtelemalla näissä näkyvän satunnaisen numerosarjan. Tavoitteena oli saada aikaan pelattava kokonaisuus yhdellä kentällä, alusta loppuun. Tämän lisäksi peliin kuuluisi pelattava hahmo ja vähintään kolme vihollistyyppiä per kenttä, jotta vihollisten välillä olisi tarpeeksi variaatiota. Lopuksi pelille suunniteltiin pientä demojulkaisua työn lopun yhteydessä, laittamalla se Google Play Storeen ladattavaksi.

Työ aloitettiin suunnittelemalla ensin pelin graafinen tyyli, sekä perusidea paremmin. Käytetyt tekniikat pitkälti oli valittu jo ennen kuin projekti edes alkoi, aikasemman kokemuksen vuoksi, pääasiallisesti Unreal Engine ja Blender. Kuitenkin muutama tekniikka tai työtapa tuli käyttöön varsinaisen työnteon aikana.

Työn tavoitteet eivät toteutuneet kokonaan opinnäytetyön aikana. Päähahmon vaatimukset saatiin toteutettua kokonaan, mutta vihollistyypeistä osa jäi työn aikana kesken. Samalla myös kentän osiin ei saatu tarpeeksi variaatiota työn aikana, joten se päätettiin jättää kasaamatta Unreal Enginen puolella, kunnes näitä osia on luotu tarpeeksi. Edellisiin pointteihin vedoten, myöskään Play Storen demojulkaisua ei toteutettu työn lopussa.

Avainsanat
Unreal Engine 4, Blender, C++, Mobiilipeli, 2D

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Sami Kimpimäki, Panu Puurunen			
Title of Thesis Game Development & Production with Unreal Engine			
Date	22 May 2018	Pages/Appendices	38
Supervisor(s) Mr. Mikko Pääkkönen, RDI Specialist			
Client Organisation /Partners			

Abstract

The purpose of this thesis was to start the process of developing a game while simultaneously learning the ins and outs of this process. The reason behind this was to kickstart some of the future plans of the thesis authors. The idea was to create a 2d mobile game, in which the player defeats enemies by inputting a serial number displayed above these enemies, while clearing the level they are on. The main goal was to achieve a working game that would be playable from start to finish, at least inside the scope of one actual game level. In addition, the game would include the player character and preferably at least three different enemy types to achieve sufficient variation in enemies. Finally, the game would be partly published by making a short demo available for download on Google Play Store.

First, the work on the game began by first deciding what kind of graphical style to use, and at the same time further refining the initial idea or vision of the game. The software used for the project was primarily already decided before work even began. This is due to the authors' prior experience regarding the programs chosen, I.e Unreal Engine 4 and Blender.

Some of the the goals set for this thesis were not completely reached. The work on the game's main playable character was finished, but some of the planned enemies were still in the development stage. Similarly, some of the planned modular pieces for the levels were still unfinished. Because of this, the levels were deemed to not be sufficiently varied at this time and were not assembled in Unreal Engine. Lastly, the demo for the Google Play Store was not released because of this.

Keywords
Unreal Engine 4, Blender, C++, Mobile game, 2D

SISÄLTÖ

TERMIT JA LYHENTEET

Unreal Engine 4 = Epic Games vuonna 2014 julkaisema pelimoottori

Sprite = pelimoottorin sisäinen 2d-objekti

Blueprint = Unreal Enginen visuaalinen ohjelmointi systeemi

C++ = perinteikäs olio-ohjelmointikieli

Google Play = Android-käyttöjärjestelmän digitaalisen sisällön levitys sovellus

Blender = Ilmainen 3d-mallinnusohjelmisto

Mesh = Kokoelma 3d-mallin vertexejä, edgejä ja faceja, josta itse malli muodostuu

Topology = 3d-mallin meshin pinnan asetelma

Asset = mikä tahansa peliin luotu 3d-objekti, piirros tai kappale.

1 JOHDANTO

Opinnäytetyön tarkoituksena oli toteuttaa mobiilipeli ja tutustua pelinkehittämisen vaiheisiin ja julkaisuun. Peli toteutettiin Unreal Engine 4:lla ja sen assetit luotiin Blenderillä. Opinnäytetyössä pyrittiin luomaan tuotos, jota pystyisi jatkokehittämään julkaistavaksi tuotteeksi, tai minkä avulla työssä saatuja tietoja ja taitoja voisi hyväksi käyttää alalla.

Opinnäytetyössä käydään yksityiskohtaisesti läpi pelinkehittämiseen vaaditut vaiheet ensin teoriaa tarkastellen, jonka jälkeen katsotaan varsinaisen työn vaiheet moottorin puolella, sekä 3d-assettien luomisen vaiheet Blenderissä. Lopuksi työssä käydään läpi pelin julkaisemisen vaiheet, vaikka itse peliä ei välttämättä julkaistaisikaan työn yhteydessä.

Pelinkehitys on monivaiheinen prosessi, joka jakaa paljon samoja vaiheita ohjelmistokehityksen kanssa. Eri henkilöillä tai yrityksillä voi olla hiukan eroavat tavat ja vaiheet pelin koosta ja tavoitteesta riippuen. Tässä dokumentissa käsitellyt vaiheet sopivat lähes kaikkeen pelinkehitykseen.

2 PELINKEHITYKSEN VAIHEET

Yleisesti pelin voi jakaa kolmeen osaan, joihin pelinkehitysvaiheet osittain perustuvat. Nämä osat ovat moottori, assetit ja säännöt. Moottori nimensä mukaisesti on peliä ajavana alustana toimiva kokoelma työkaluja, tekniikoita ja datakirjastoja. ”Asset” tai assetit, löyhästi tarkoittaa kaikkea peliin tulevaa tai kuuluvaa dataa, tähän siis esimerkiksi sisältyy kaikki pelin 3d-objektit, musiikki, sekä konseptit ja muu taide. Kolmas, näitä kahta yhdistävä osa, on pelin säännöt. Sääntöihin sisältyy toisinsanoen pelin varsinainen pelitapa, mekaniikat, ja esimerkiksi voiton tai häviön perusteet. Näihin perustuen itse pelinkehitysvaiheet voi jakaa osiin, suuresti riippuen kehitysajan käytöstä ja tärkeydestä. Vaiheet eivät ole mitenkään varsinaisesti määriteltyjä kenenkään toimesta, mutta monet kehittäjät silti seuraavat samankaltaisia työnvaiheita. Näitä vaiheita voidaan yleisesti tunnistaa kahdeksan, jotka suoritetaan yleensä järjestyksessä. (Thorn 2013, 17-19.)

Ensimmäinen näistä vaihesta on melkein kaikissa tapauksissa pelin perusidean kehittely ja muovaaminen kunnolliseksi. Tähän siis pitkälti sisältyy mekaniikkojen suunnittelu ja yleisesti pelin sääntöjen kehitys, miten peliä pelataan, miten peli voitetaan tai hävitään, mihin genreen peli kuuluu ja mitä tämän tai muiden genrejen ominaisuuksia peliin otetaan mukaan. (Thorn 2013, 19-21.)

Toinen vaihe on prototyypin luominen näistä säännöistä ja näin teorian pyörittäminen, siitä miten ne soveltuvat peliksi. Säännöt kuitenkin voivat vielä vaihdella myöhemmin, riippuen itse pelin oikeasta testauksesta. (Thorn 2013, 21.)

Kolmannessa vaiheessa näitä aikaisemmin luotuja sääntöjä ja mekaniikoita suunnittellaan vielä eteenpäin samalla parannellen jo kehiteltyjä. Aikaisemman prototyyppivaiheen testaukset oletettavasti auttavat hienosäätämään näitä lisää. (Thorn 2013, 21-22.)

Neljäntenä vaiheena pidetään usein peliin tarvittavan moottorin luomista. Jos moottori tehdään tällä tavalla, niin se yritetään räätälöidä mahdollisimman tarkasti pelin tarpeisiin pohjautuen. Moottori toimii perustana kaikelle mitä pelissä voi tai ei voi tehdä, sekä pelin asseteille. (Thorn 2013, 22.)

Viidennessä vaiheessa luodaan pelin assetit. Nämä ovat siis mitä tahansa pelissä näkyvää tai kuuluvaa videota, ääniä, musiikkia tai 3d-objekteja. Tässä vaiheessa siis tehdään varsinainen graafinen sisältö peliin. Assettien luontivaihe on myös yleisesti pisin kaikista vaiheista mitä pelinkehitykseen sisältyy, sillä peleihin sisältyy useasti useita kymmeniä tai satoja assetteja ja kaikkien näiden tekeminen vie oman aikansa. Tämä vaihe myös yleisesti suoritetaan moottorin teon ja käsikirjoituksen tekemisen ohella. (Thorn 2013, 22.)

Kuudentena vaiheena voidaan pitää pelikohtaisten scriptien, eli koodin kirjoittamista. Tässä vaiheessa kirjoitetaan peliin sisältyvää koodia, joka ei kuulu pelkästään moottoriin vaan on pelikohtaista, pitkälti siis mekaniikat ja säännöt jotka suunniteltiin aikasemmissa suunnitteluvaiheissa. Scriptaus vaiheen jälkeen pelin olisi tarkoitus olla ”valmis”, tai toisinsanoen pelikunnossa. Seuraava vaihe perustuukin tämän valmiustason selvittämiseen. (Thorn 2013, 23.)

Seitsemäs vaihe kehityksessä on varsinainen testaus ja debugaus vaihe. Toisinsanoen palkatut tai vapaaehtoiset testaajat pelitestaavat, missä kunnossa kyseinen peli on ja dokumentoivat löydetyt ongelmakohdat ja bugit, jotka pyritään korjaamaan ennen julkaisemista. (Thorn 2013, 23.)

Kahdeksas eli viimeinen vaihe on pelin julkaisu ja markkinointi. Vaiheen tarkoitus on yksinkertaisesti markkinoida peliä ja yrittää saada mahdolliset pelaajat huomaamaan ja täten hankkimaan sen. (Thorn 2013, 24.)

Näistä vaiheista työssä ei käyty kaikkia läpi, esimerkiksi moottorin luontiin ei menty ollenkaan, sillä työssä käytettiin jo valmiiksi olemassa olevaa pelimoottoria. Kuitenkin suurin osa vaiheista pitää paikkansa tämänkin työn työvaiheita kuvaillessa.

3 SUUNNITTELU

3.1 Pelimekaniikat

Päämekaniikaksi pelille kehittyi ominaisuus, jossa vihollisten ja eri kentässä olevien peliobjektien yläpuolella on numerokoodi. Kirjoittamalla tämän numerokoodin pelaaja tuhoaa vihollisia ja vuorovaikuttaa muihin peliobjekteihin, kuten ovien avaamiseen tai hissin käyttöön. Pelaajan edetessä vaikeampiin kenttiin numerokooditkin vaikeutuvat ja numerot voivat muuttua symboleihin. Peli on suunniteltu mobiilialustoille, joten numerokoodit syötetään näytöllä olevilla painikkeilla.

Jotta pelissä riittäisi pelattavaa pitemmälle ajalle, haluttiin pelaajan suorittavan pelikentät mahdollisimman nopeasti. Joka pelikentän parhaat suoritusajat pääsisivät julkiselle tulostaululle. Käyttäjät sitten kilpailisivat parhaimmasta suoritusajasta. Tämä myöskin lisää pelattavuutta, kun käyttäjät yrittävät parantaa aikojansa samoissa kentissä ja vähentää tarvetta valtavalle määrälle kenttiä.

Pelissä, jossa nopea liikkuminen on tärkeää, on pelihahmon sulavaan ja mielekkääseen ohjattavuuteen kiinnitettävä huomiota. Tämän takia perinteisen liikkumisen ja hyppäämisen lisäksi pelihahmolle haluttiin kaksoishyppy ja hyppy seinän kautta, jotta pelaajalla olisi helpompi navigoida vertikaalisissa kentissä.

3.2 Kenttäsuunnittelu

Pelin kentistä haluttiin enemmän vertikaalisia kuin horisontaalisia tuomaan haastetta peliin. Tämä suunniteltiin pitäen mielessä pelimekaniikkoja kuten kaksoishyppyä ja hyppyä seinän kautta.

Kenttien suunniteltiin käyttävän eri teemoja. Tällaisia teemoja voi olla esimerkiksi urbaani pilvenpiirtäjä, jonka päätettiin olevan ensimmäinen teema, jonka pohjalta alettiin tekemään assetteja. Pelin kenttien ollessa simulaatioita pelihahmolle näissä teemoissa voi olla hyvinkin luovia, kuten joku voi olla unimaailmaan taikka avaruuteen sijoittuva. Jokaisen teeman alle on ajateltu tulevan noin 10 eri kenttää. Kenttien assetit tulevat olemaan modulaarisia paloja, joista on helppo kasata ja muuttaa kenttiä.

4 TYÖKALUT JA TEKNIIKKA

4.1 Unreal Engine 4

Unreal Engine 4 on Epic Gamesin vuonna 2014 julkaisema pelimoottori. Moottori soveltuu niin isoille pelistudioille kuin pienillekin indie-kehittäjille ja sillä voi kehittää 2d ja 3d pelejä genrestä riippumatta. Tuettaviin julkaisualustoihin kuuluvat muun muassa iOS, Android, Windows, Playstation 4 ja Xbox One. Unreal Engine 4 (UE4) on ilmainen käyttää, mutta kaupallisen tuotteen bruttotuoton ylittäessä 3000 dollaria per vuosineljännes on maksettava 5% rojalti. (Epic Games.)

UE4:lla tehtyihin peleihin lukeutuvat muun muassa Fortnite (Epic Games), Tekken 7 (Bandai Namco) ja Gears of War 4 (The Coalition). Menestyneitä nimikkeitä löytyy myös mobiilialustoilta, joista esimerkkinä Netmarble Gamesin Lineage 2: Revolution joka noin kahden kuukauden aikana onnistunut ylittämään viiden miljoonan käyttäjän rajan. (MMORPG 2018.)

Ohjelmointi UE4:llä tapahtuu joko C++:lla tai Blueprintilla. C++:n ollessa suorituskyvyltään tehokkaampi, mutta ohjelmointia osaamattomalle opettelukynnys on suhteellisen korkea ja tästä syystä Blueprint visuaalinen ohjelmointi soveltuu paremmin esimerkiksi artisteille ja kenttäsuunnittelijoille, jotka eivät halua käyttää aikaa perusteelliseen ohjelmoinnin opetteluun. Blueprintit soveltuvat hyvin pelin prototyypin tekemiseen tai käytettäväksi silloin kun Blueprint-koodia ei tarvitse ajaa joka ruudunpäivityksellä, mutta on myös mahdollista kehittää kokonainen toimivat peli niillä. Heikkoutena Blueprinteissa verrattuna C++:aan on huomattavasti heikompi suorituskyky ja käyttäjän ollessa riippuvainen olemassa olevista solmuista (node). Solmuja voi tehdä itsekin, mutta tämä vaatii C++:n käyttöä.

Vaikka UE4:llä voi tehdä 2d-pelejä, ovat sen työkalut heikommat verrattuna suosittuun kilpailijaansa Unity 3d-pelimoottoriin. Tästä riippumatta UE4 valikoitui käytettäväksi moottoriksi opinnäytetyön tekijöiden jo ennestään kattavan UE4 kokemuksen myötä, jota on kertynyt useista pelijameista (game jam). Ennestään oleva kokemus mahdollisti nopeamman pelin kehityksen.

4.2 C++ Unreal Enginessä

UE4:n C++:sta löytyy hieman eroavaisuuksia standardi C++:n kanssa. Epic Games on rakentanut oman versionsa kyseisestä kielestä standardi C++:n päälle. Epic Games on lisännyt joukon omia makrojaan helpottamaan ohjelmointia, toimimaan merkkaimina automaatiselle roskienkeräykselle (garbage collection) ja toimimaan linkkinä C++:n ja Blueprinttien välillä.

Esimerkkinä näistä makroista on UPROPERTY() ja UFUNCTION(). Näillä makroilla voidaan antaa muuttujalle tai funktiolle tageja, jotka UE4 käsittelee käännön yhteydessä. Näillä tageilla voidaan esimerkiksi antaa Blueprint-luokalla luku- ja kirjoitusoikeus C++-luokan muuttujalle, tai määrittää muuttujan näkyvyys editorissa.

Makrojen lisäksi UE4:sta löytyy omia muuttuja- ja säiliötyyppejä. Uudet muuttujat tunnistaa F-etuliitestä (FString) ja säiliöt taas T-etuliitteestä (TArray). F-tyyppiset muuttujat kääntyvät vaivatta perusmuuttujiksi kuten float tai string sekä toisinpäin. (Epic Games.) Kuva 1 kuvaa UE4:n C++-koodin rakennetta.

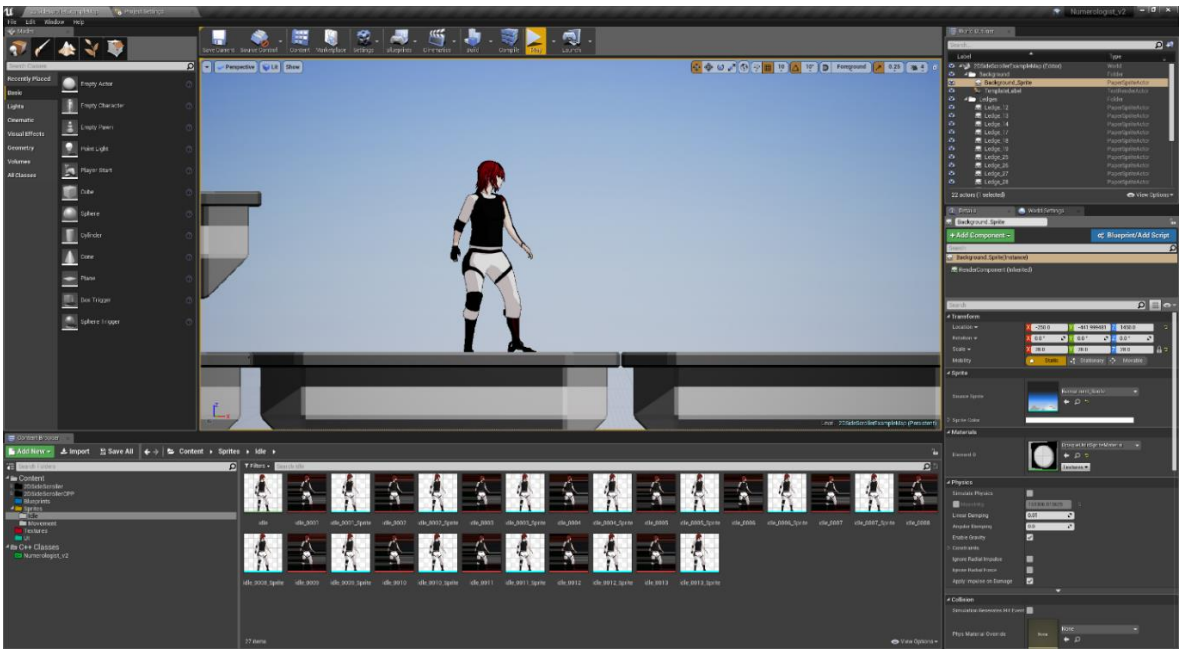
```
25  public:
26
27      APaperEnemyParentClass();
28
29      virtual void Tick(float DeltaTime) override;
30
31      virtual void CheckCodeMatch(const FString& InputCode) const override;
32
33      FString GetCode() const;
34
35      UPROPERTY(EditAnywhere)
36          bool GenerateOwnCode;
37
38      UPROPERTY(EditAnywhere, BlueprintReadWrite)
39          FString CodeString;
40
41
42  };
43
```

KUVA 1. esimerkki Unreal Enginen c++ -koodista

4.3 Unreal Editor

Unreal Editor pitää sisällään suurimman osan työkaluista pelinkehittämiseen. Editori pitää sisällään työkalut kuten materiaalieditorin, partikkeli-editori Cascaden ja animaatioeditori Personan, jonka sisällä voi hallita 3d-mallia, sen animaatioita ja tehdä animaatio Blueprintin. Myös tuki kolmannen osapuolen laajennuksille löytyy editorista.

Editorin avatessa keskeltä löytyy viewport, joka kuvastaa pelimaailmaa. Sen alta löytyy content browser, josta löytyvät kaikki koodiluokat ja assetit. Content browserista löytyy mahdollisuus hakea ja suodattaa tiedostoja ja kansioita, sekä mahdollisuus järjestellä näitä. Vasemmalta puolelta editoria löytyy työkaluja maaston luontiin ja pelimaailman hahmottelemiseen. Oikealta puolella on world outliner, jossa listattuna kaikki pelimaailman assetit sekä details- ja world setting-paneelit, kuten kuvasta 2 näkee.



KUVA 2. Yleisnäkymä editorista

4.4 Visual Studio

Ohjelmointi UE4:lla on suunniteltu tapahtuvan Microsoftin Visual Studiolla. Visual Studio on yksi suosituimmista kehitysympäristöistä (IDE) (Krill 2017). Se tarjoaa tuen useille eri ohjelmointikielille, teknologioille ja julkaisualustoille. UE4 luo valmiiksi Visual Studio-projektin jokaisella projektille.

UE4:ssa käytettävä C++ syntaksi ei ole suoraan yhtenevä C++ standardin kanssa. C++:n eroavaisuuksia käsitellään tarkemmin seuraavassa osiossa. Tämä aiheuttaa tilanteita jolloin Visual Studion Intellisense ei toimi tai luulee osan syntaksista olevan virheellinen.

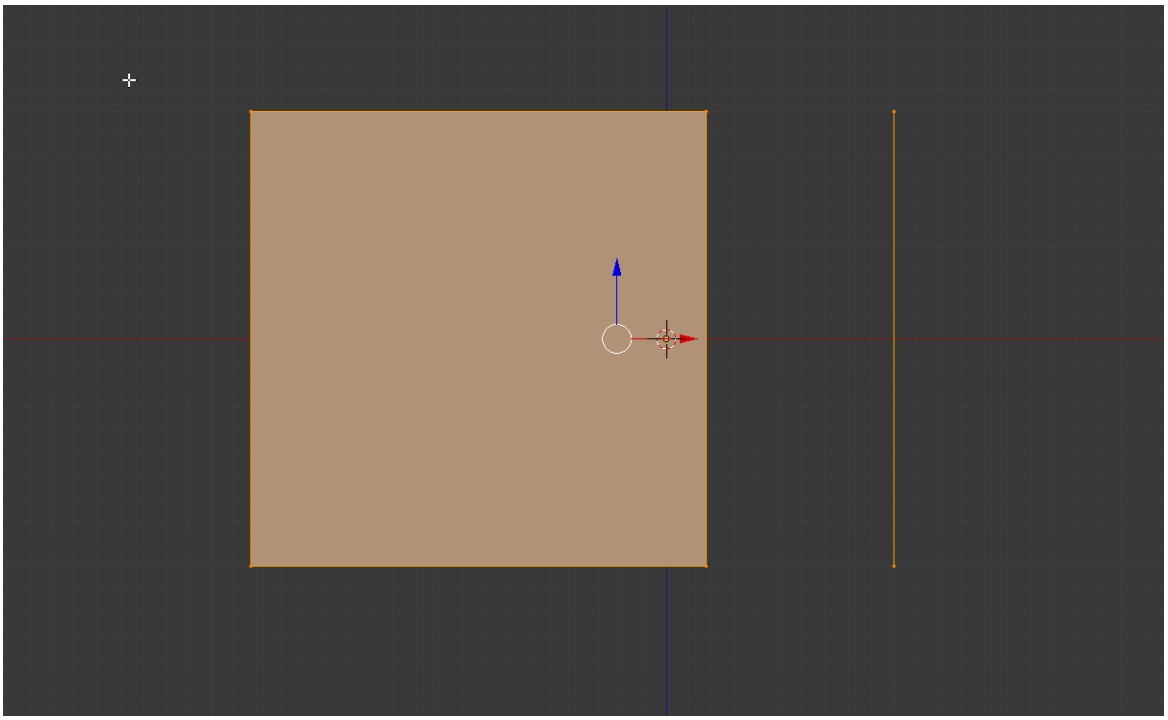
4.5 Yleistä 3d-mallinnuksesta

4.5.1 Vertex, face ja edge

Kyseiset nimitykset ovat mallinnuksessa käytettävien rakennuspalikoiden tai ”polygonien” (mikä tahansa kaksiulotteinen ja kaksi tai useampi sivuinen kuvio) osia. Vertex tarkoittaa 3d-näkymän pisteavaruudessa sijaitsevaa yhtä pistettä. Edge tarkoittaa pisteitä yhdistävää suoraa tai viivaa, joka samalla toimii polygonin reunana. Face on tämä pisteiden ja viivojen yhdistelmästä syntyvä näkyvä sivu, kuvassa 3 esimerkki näistä. Faceille / polygoneille on myös omat nimityksensä siitä riippuen, kuinka monireunaisia ne ovat. Quads, eli nelisivuiset polyognit ovat näistä yleisimpiä millä mallinnus toteutetaan. Tämän lisäksi on olemassa tris eli kolmisivuiset, sekä ngon eli suuremmat kuin nelisivuiset. (Thorn 2013, 140.)

Riippuen mallinnuksen tarkoituksesta, kaikkia näitä erisivuisia polygoneja voi käyttää erilaisiin tarkoituksiin. Pelien tapauksessa tosin, yleisin tapa tehdä malli, on käyttää quadeja, eli nelisivuisia polygoneja. Näin yleensä saadaan aikaan sopivin lopputulos, sekä helpoiten muokattavissa oleva malli,

joka myös oletettavasti tuottaa vähiten ongelmia pelimoottoriin tuotaessa. Pelimoottorit kuitenkin yleensä muuntavat kaikki 3d-objektien polygonit kolmisivuisiksi.



KUVA 3. Esimerkki face vasemmalla, edge oikealla, pisteet vertexejä

4.5.2 Mallinnustekniikat

Yleisesti ottaen kaikkiin 3d-mallinnusohjelmistoihin pätevät samat mallinnustekniikat, jotkin tosin ovat enemmän erikoistuneita tiettyyn tekniikkaan, esim zbrush (sculptaus). Ohjelmistot omaavat kuitenkin myös pienempiä eroja toisiinsa verrattuna, kuten erilainen UI (käyttöliittymä), näppäinyhdistelmät, ohjelman sisällä käytettävä termistö ja muita.

Perinteisesti yleisin tapa, sekä ehkä helpoin, on tehdä malli käyttäen niin sanottua Polygonimallinnusta (polygonal modeling). Tällä metodilla käyttäjä manuaalisesti luo, liikuttaa ja muokkaa mallin polygoneja aiemmin mainituilla osilla (face, edge, vertex).

Toinen tapa tehdä malli, on käyttää kurvi- tai käyrämallinnusta (curve modeling). Tässä tekniikassa mallinnusohjelmistolla luodaan yksittäisiä käyriä haluamaansa muotoon, joihin käyttäjä voi proseduraalisesti luoda polygonipinnan. Näitä käyriä on useampia, kuten NURBS käyriä (nonuniform rational B-spline), Bézier käyriä, splinikäyriä, sekä muita (Farin, Hoschek ja Kim 2002).

Näiden lisäksi kolmas tapa tehdä 3d-malli, on sculptaus (sculpting) (Ward 2008, 101). Sculptaaminen on hyvinkin verrattavissa veistämiseen tai saven muovaamiseen, mutta digitaalisesti. Itse sculptaamiseenkin sisältyy eri tekniikoita, millä sen voi toteuttaa. Sculptauksessa yleisesti myös käytetään hyvinkin paljon korkeamman polygonimäärän omaavaa mallia. Ehkäpä käytetyin tapa sculptaukseen on ”siirtymä” (displacement), jossa mallin polygonit vain liikkuvat muokkauksen mukaisesti. Tämä voikin aiheuttaa venymistä alueilla, jossa polygoneja ei enään ole tarpeeksi muokkauksen jälkeen.

Toinen samankaltainen tähän verrattuna on volumetrinen muokkaus. Kolmas, hyvinkin yleistyvä tapa on dyntopo eli dynaaminen topologia. Sculptaus tehdään täysin samalla tavalla kuin aikasemmilla tekniikoilla, mutta nimensä mukaisesti ohjelmisto luo dynaamisesti lisää polygoneja malliin käyttäjän muokatessa sitä. Malliin haluttu tarkkuus ja luotujen polygonien määrä riippuu pitkälti dyntopon asetuksista.

4.6 Blender

Blender on ilmainen avoimen lähdekoodin omaava, pääasiallisesti 3d-mallinnukseen käytettävä ohjelmisto. Se tukee kuitenkin mallinnuksen lisäksi muita siihen vahvasti liittyviä toimintoja, kuten animointia, simulointia, renderöintiä ja monia muita.

Blenderiä ylläpitää Blender Foundation ja suuri osa sen lisäosista tai ominaisuuksista kehitetään käyttäjien toimesta. Blender Foundation silloin tällöin myös kehittää omia animaatioitansa, esitelläkseen blenderin toimintakykyä.

Blender valittiin käytettäväksi ohjelmistoksi tekijöiden ennestään omaavasta kokemuksesta, sekä ennenpitkäisen tuotteen kaupallistamisen vuoksi. Pelialan yleisintä 3d-ohjelmistoa, mayaa, ei valittu käytettäväksi juuri tämän syyn takia, sillä maya on kylläkin ilmainen opiskelijoille, mutta tällä lisenssillä ei voi tuottaa kaupallista sisältöä.

4.6.1 Scene

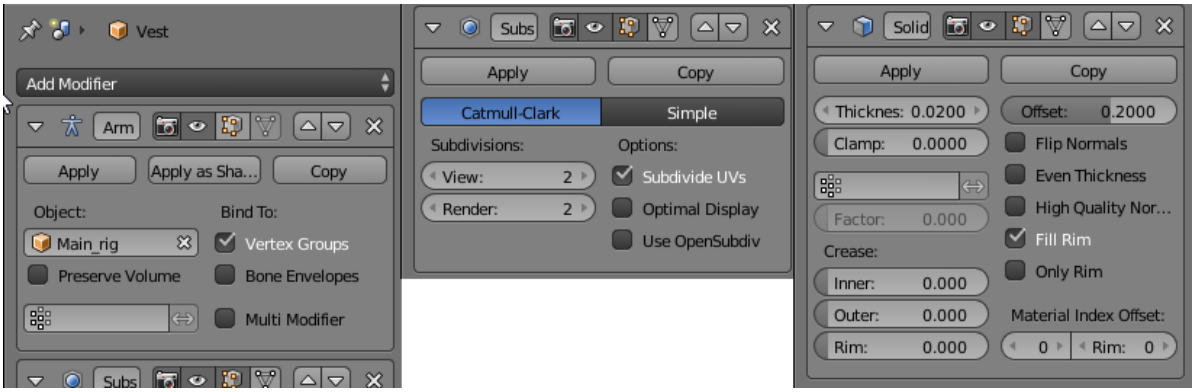
Scene toimii blenderin pääasiallisena näkymänä tai nimensä mukaisesti ”näyttämönä” yhdelle tai useamalle objektille, mitä käyttäjä haluaa renderöidä. Scenestä voi juurikin hallita monia asetukia tähän liittyen kuten asettaa näkyvän taustan, painovoiman hallinta, Värien hallintaa ja monia muita.

Blenderissä on myös käytössä useampia kerroksia (layer), joista voi vapaasti valita mitä kerroksia, sekä scenejä valitaan mukaan renderöintiin. Nämä kerrokset voivat sisäisää täysin omat objektinsa.

4.6.2 Muuntimet ja lisäosat

Blenderissä on useita muuntimia tai näiden ominaisuuksia joita muokata. Näistä hyvinkin monet ovat erittäin hyödyllisiä varsinkin mallinnuksen kannalta. Työ, jonka muuten voisi joutua tekemään käsin, helpottuu huomattavasti, jos tarkoitukseen löytyy sopiva muunnin. Muutamia usein käytettyjä esimerkkejä: Mirror (peilikuva) kopioi tai peilaa tehdyn objektin valitun koordinaatin mukaisesti, subdivision (jakaminen) jakaa objektin facet / polygonit kahtia, täten tuplaten niiden määrän. Kuvassa 4 muutama esimerkki.

Add-ons eli toisinsanoen lisäosat, ovat hyvin suuri osa blenderin modulaarisuutta ja monipuolisuutta. Osa lisäosista on pakattu suoraan blenderin mukaan ja käyttäjä voi halutessaan aktivoida mitä tarvitsee tai haluaa ohjelman asetuksista. Näitä lisäosia tai skriptejä voi ladata ja asentaa myös manuaalisesti.



KUVA 4. Esimerkki ominaisuuksista. Armature, subdivide ja solidify

4.6.3 Node editor

Solmueditoria käytetään pääasiallisesti renderöitävän objektin tai scenen materiaalien ja tekstuurien, sekä shadereiden muokkaukseen. Solmueditorissa olevat ominaisuudet ja nodet (solmut) riippuvat valitusta rendereristä.

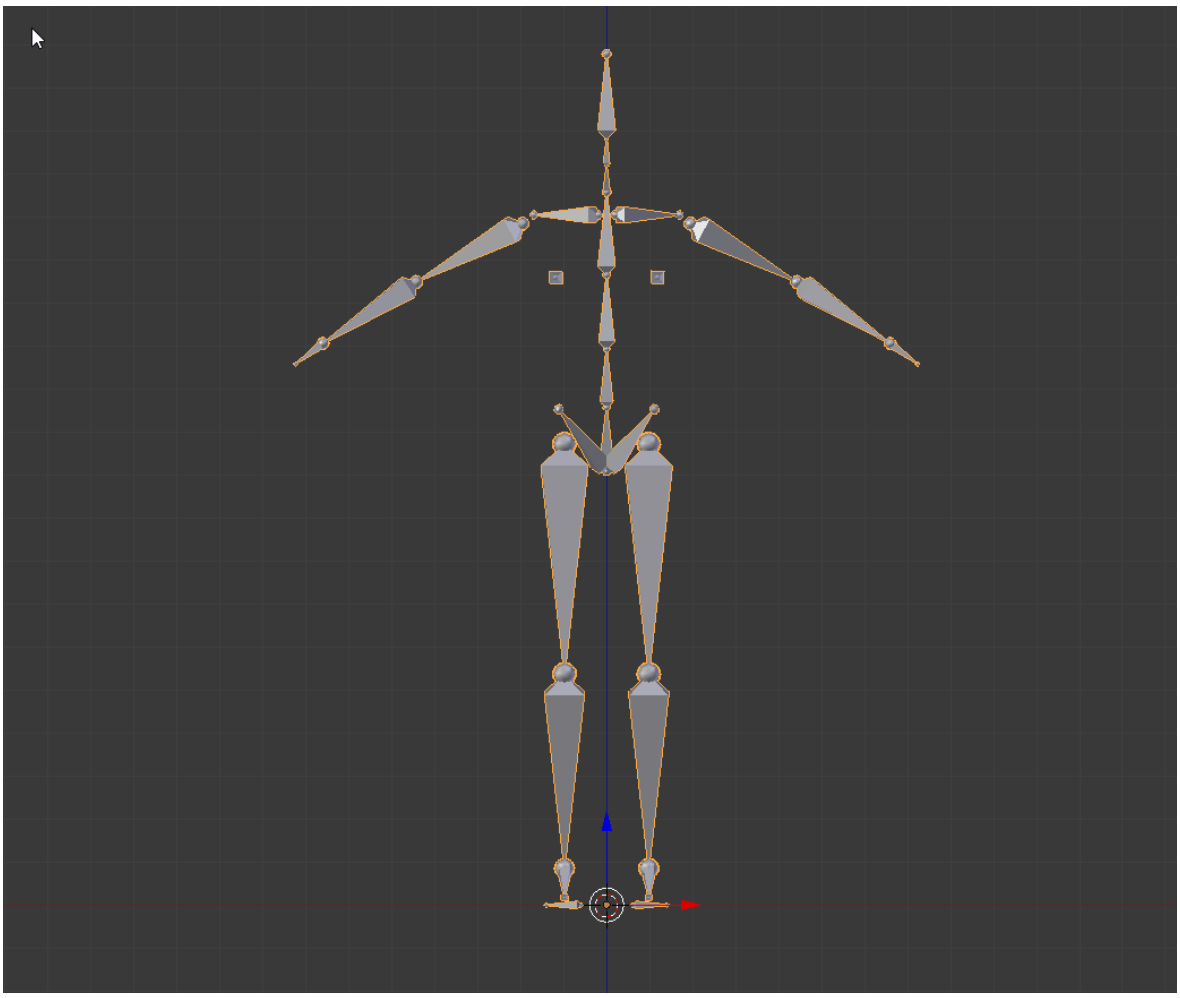
Editorin ”solmut”, nodet, esittävät eri datalähteitä, muuntimia ja tulosteita. Nämä yhdistetään toisiinsa, luoden dataa prosessoivan ketjun, jonka tulos syötetään tulostesolmulle. (Blender Foundation.)

4.6.4 Armature, rigify ja skinning

Armature tai yleisesti rig on animoitavalle mallille tehty luuranko, joka toimii hahmon animoinnin perustana. Rigin osat ja niille määritetyt arvot määräävät miten hahmo muotoutuu ja liikkuu.

Rigify on yksi blenderin lisäosista, jossa tulee muutamia lisäasetuksia itse rigien muokkaamiseen, sekä valmiiksi luotuja rigejä. Nämä ovat käyttövalmiita semmoisenaan tai muokattavissa kuten Haluaa, ks. kuva 5.

Skinning, blenderissä weight paint, tarkoittaa objektin meshin (tekninen nimitys geometriselle mallille) ”painottamista” mallille luotuun rigiin. Blenderissä yhdellä rigin osalla, luulla, voi olla painoarvo väliltä 0-1, tämä arvo kertoo kuinka paljon kyseessä oleva luu muovaa tai liikuttaa sille määrättyä aluetta mallista.

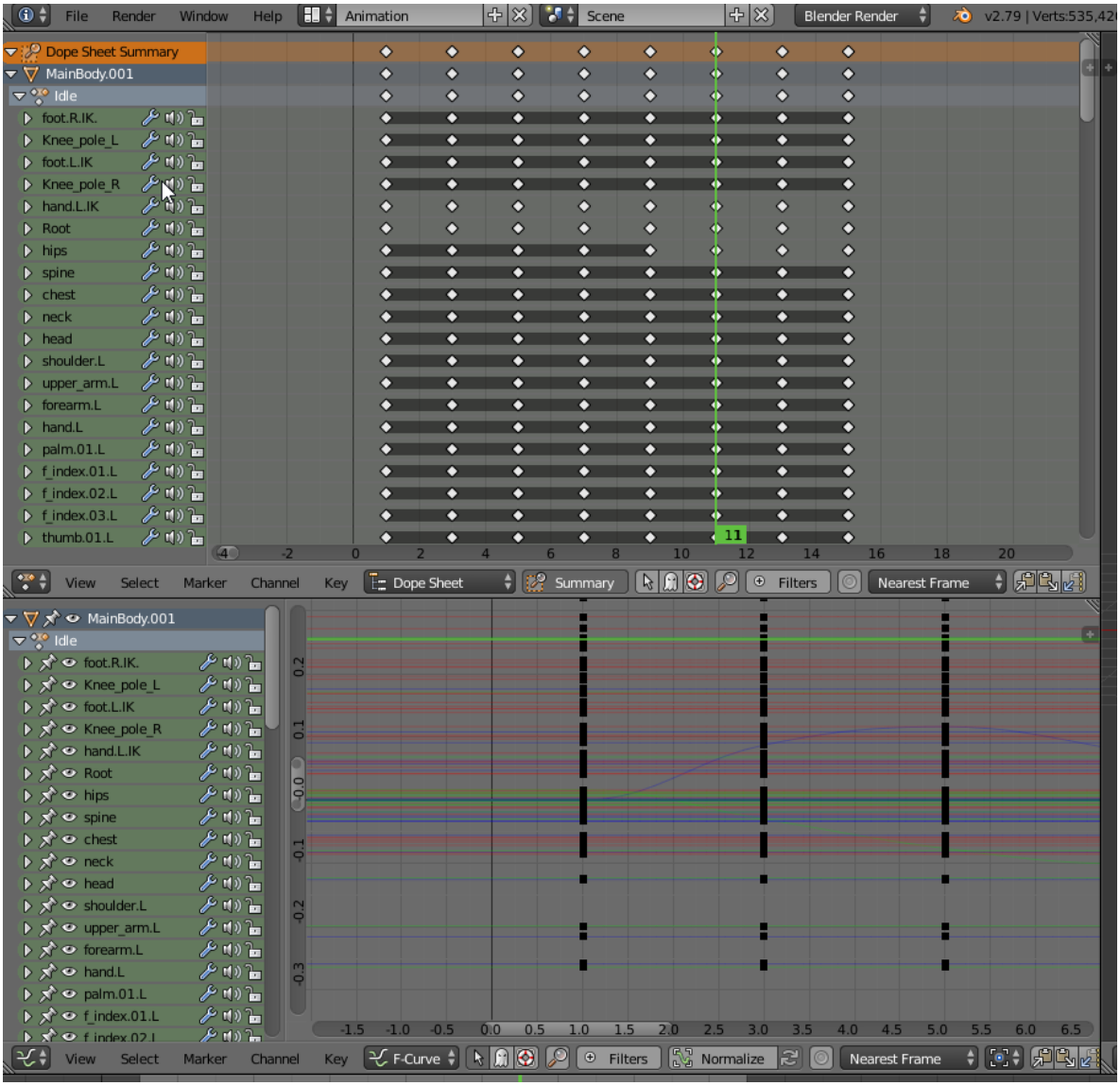


KUVA 5. Rigifyn basic human meta-rig

4.6.5 Animointi

Yksinkertaisimmillaan animointi tapahtuu liikuttamalla rigin luita haluamaansa paikkaan Pisteavaruudessa, joko syöttämällä lukuja yksitellen eri akseleille (x,y,z) tai siirtämällä vapaasti itse luusta. Tähän tosin on muitakin tapoja, kuten liikeratojen piirtäminen tai mocap (motion capture), jossa näyttelijän tai objektin liikettä seurataan yleensä kameroiden avulla.

Kun haluttu asento on saavutettu, tallennetaan asennon pisteavaruuden lokaatio ja rotaatio arvot yhdeksi frameksi (kuva). Tämä muodostaa jokaiselle rigin luulle oman avaimen joka sisältää kyseiset arvot, ks kuva 6. Prosessi toistetaan niin monta kertaa, kuin tekijä näkee tarpeelliseksi animaation luomiseksi. Blenderissä, kuten monessa muussakin animoinnin mahdollistavassa ohjelmistossa, on tyhjien framien interpolaatio, toisinsanoen ohjelma tulkitsee key framien välisten tyhjien framien rigin luiden position ja rotaation laskemalla niille liikeradan, perustuen näihin aikasempiin key frameihin.



KUVA 6. Esimerkki animaation keyframeista

4.6.6 Teksturointi ja renderöinti

Teksturointi tapahtuu pitkälti materiaaleja muokkaamalla, pois lukien varsinainen kuvateksturointi, mutta tämäkin vaatii perusmateriaalin toimiakseen. Teksturoinnilla tarkoitetaan siis mallin pintamateriaalien maalaamista tai sijoittamista mallin pinnalle.

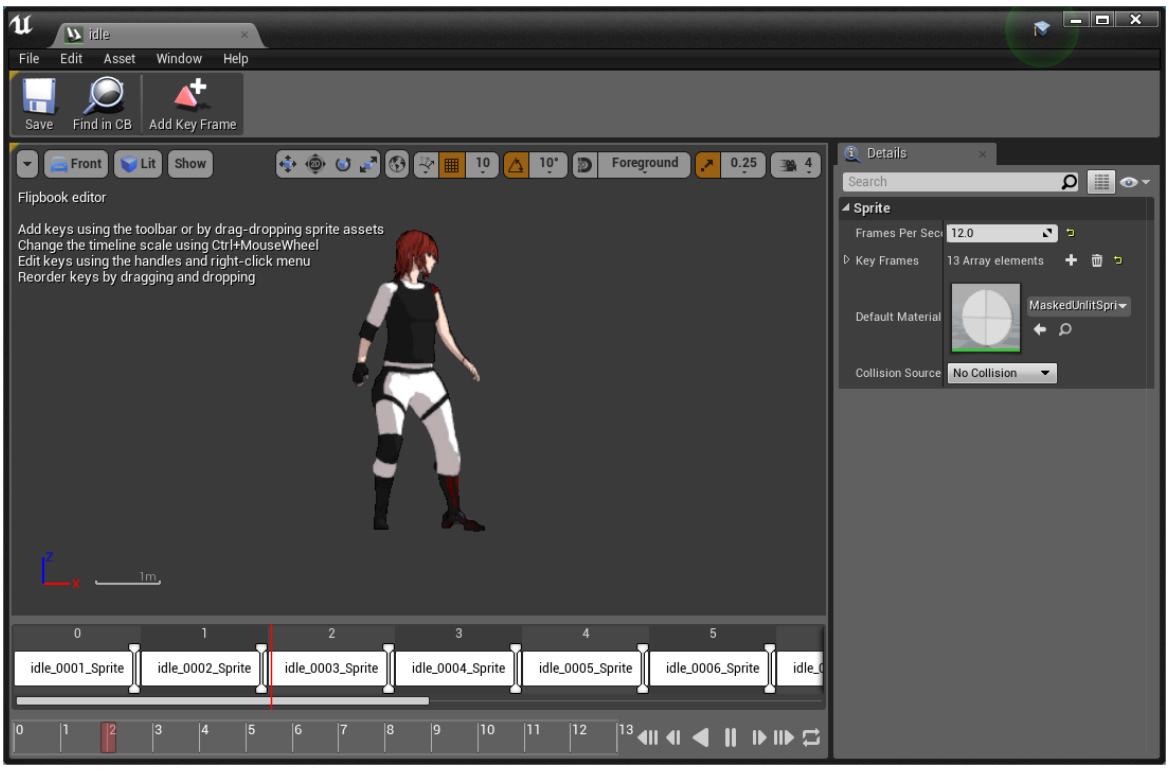
Blenderiin kuuluu kaksi rendereriä: blender render ja cycles render. Cycles on näistä uudempi ja tehokkaampi, blender renderin ollessa yksinkertaisempi. Rendereillä on myös eroja asetuksiin, ominaisuuksiin ja laatuun nähden.

5 TOTEUTUS

5.1 Spritet ja Flipbook

Blenderissä renderöidyt kuvat tuotiin UE4:een PNG-tiedostona. Monet muutkin kuvatiedostoformaatit käyvät kuten JPG ja TGA mutta PNG valittiin sen alpha-kanavatuen takia. Alpha-kanava on kuvatiedostoissa oleva kuvan läpinäkyvyyttä ilmaiseva kanava. UE4:n sisällä tuodut kuvatiedostot muutetaan UE4:n omaan sprite muotoon. Spriteille voi halutessaan määrittää törmäysalueen (collision area), tämä tehtiin staattisille objekteille joiden muoto ei pelin aikana muutu.

Jotta spriteistä saadaan kasattua animaatio, on sitä varten luotava flipbook. Flipbook pitää sisällään aikajanan johon animaation spritet laitetaan sekä muutamia asetuksia animaatiolle. Käyttäjä voi joko luoda tyhjän flipbookin ja siihen raahata ja pudottaa haluamansa spritet aikajanalle. Helpompi tapa, varsinkin jos spritejä on paljon, on valita kaikki haluamansa spritet content browserista ja painaa hiiren oikeaa nappia. Sieltä löytyy valinta ”create flipbook”. Tämä luo uuden flipbookin ja heittää spritet aikajanalle valmiiksi, kuten kuvassa 7. Jos spritet on järkevästi numeroitu, menee ne aikajanalle oikeassa järjestyksessä. Spritejen järjestystä voi vapaasti muuttaa aikajanalla.



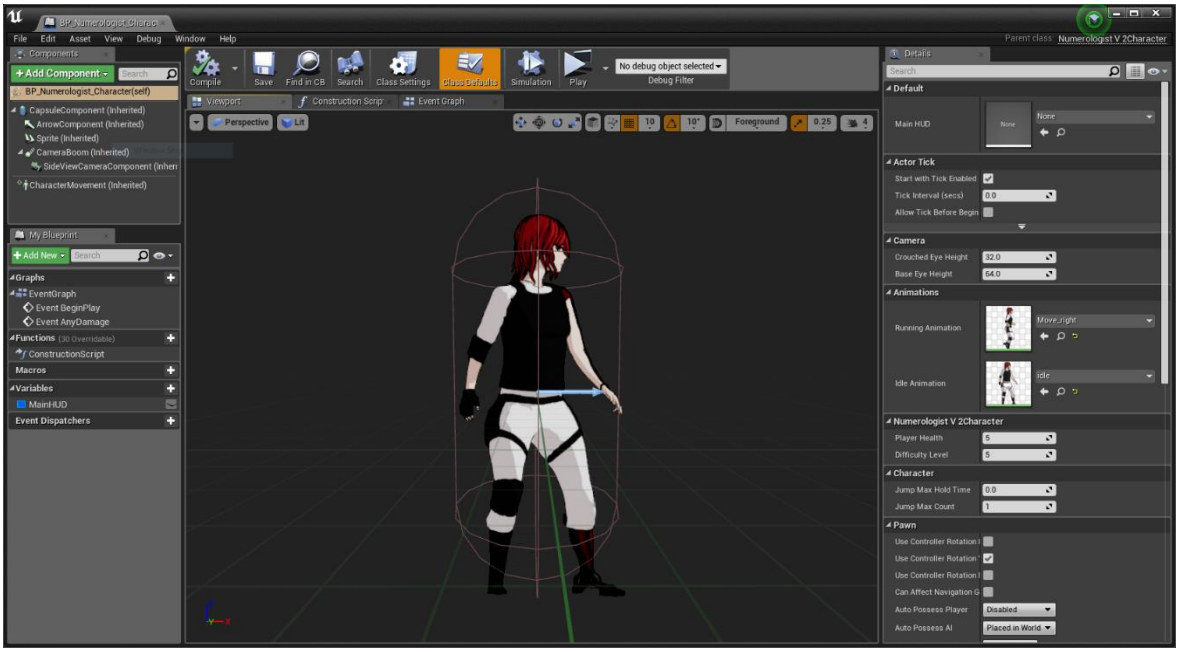
KUVA 7. Näkymä flipbookista

5.2 Mekaniikkojen ohjelmointi

Actor on kaikista pelkistetyin ohjelmointiluokka jonka voi sijoittaa pelimaailmaan. Kaikki muut pelimaailmaan sijoitettavat ohjelmointiluokkia käyttävät objektit ovat actor-luokan aliluokkia. Tilanteessa jossa jonkin logiikkaa sisällä pitävän objektin luokkaa ei tiedetä, voidaan sitä käsitellä actor-luokkana.

Ohjelmointi aloitettiin pelihahmo-luokasta. Pelihahmo-luokka tehtiin luomalle UE4:sta löytyvä Paper2dCharacter-luokka. Kyseinen luokka pitää sisällään perusohjaimet liikkumiselle ja flipbook-animaatioiden vaihdon juoksun ja oletusasennon välillä. Tämän päälle rakennettiin sitten omat toiminnot. Pelihahmolle lisättiin int-tyyppiset muuttujat CurrentHealth ja MaxHealth vastaamaan hahmon elämistä. Hahmon pelimaailmaan luonnin yhteydessä CurrentHealth:n arvo laitetaan MaxHealth:n arvoksi. Vihollisten tehdessä pelihahmoon vahinkoa vähennetään se CurrentHeath:sta, jonka jälkeen tarkistetaan, onko kyseinen arvo vähemmän tai yhtä suuri kuin nolla. Jos näin on, lähtee asiasta tieto pelimoodille.

Hyppy-mekaniikoille tehtiin oma funktionsa. Jos pelihahmo on maassa, sen sallitaan hyppäävän, kun taas hahmon ollessa ilmassa hahmo voi hypätä uudestaan mutta vain kerran. Silloin boolean muuttujan CanDoubleJump arvo muutetaan false, joka pelaajan laskeuduttua palautetaan true-arvoon. Ennen kuin hahmon toista hyppyä ilmassa tarkistetaan, onko pelihahmon edessä seinää. Jos näin on, tehdään hyppy seinän kautta, eli kiihdytetään pelihahmo yläviistoon seinästä poispäin. Sen jälkeen pelaaja voi käyttää vielä hyppyä ilmassa. C++-luokan ollessa valmis, luotiin siitä perivä Blueprint-luokka, jonka näkee alla olevasta kuvassa 8. Blueprintissä on helppo tehdä pieniä hienosäätöjä, eikä tarvitse odottaa koodin kääntymistä.



KUVA 8. Pelihahmon Blueprint

Koodin kirjoittamista varten HUD:iin (head-up display) luotiin painikkeet numeroille yhdestä neljään vastaamaan koodissa käytettyjä numeroita. Painikkeiden painaminen kutsuu pelihahmo-luokasta löytyvää funktiota CodesInput() ja lähettää tälle parametrina mitä numeroa pelaaja oli painanut. Kun pelaaja on antanut kolmesta viiteen merkkiä pitkän koodin, kentän vaikeustasosta riippuen, hakee funktio kaikki CodeInterface-rajapintaa käyttävät actorit pelimaailmasta ja kutsuu näiden rajapinnan funktiota CheckCodeMatch(FString Code). Tämän jälkeen kirjoitettu koodi nollataan.

CodeInterface-rajapinta haluttiin luoda yhtenäistämään luokkia joihin pelaaja voi vaikuttaa koodeilla. Tämä myös yksinkertaisti prosessia, jolla koodien vastaavuus tarkistetaan yksinkertaisesti vain kutsumalla CheckCodeMatch() funktiota ja antaa sille parametrina kirjoitettu koodi. Rajapinta tarjoaa myös funktiot GenerateCode() sekä GetCode(). GenerateCode() pitää huolen uniikkien numerokoodien luonnista sitä tarvitseville luokille ja GetCode() palauttaa luokan numerokoodin. Katso kuvasta 9 rajapinnan rakenne.

```
1   #pragma once
2
3   #include "CoreMinimal.h"
4   #include "CodeInterface.generated.h"
5
6
7   UINTERFACE(MinimalAPI)
8   class UCodeInterface : public UInterface
9   {
10      GENERATED_BODY()
11  };
12
13
14  class NUMEROLOGIST_V2_API ICodeInterface
15  {
16      GENERATED_BODY()
17
18
19  public:
20
21      virtual void CheckCodeMatch(const FString& InputCode) const;
22
23      FString GetCode() const;
24  };
25
```

KUVA 9. Esimerkki CodeInterface -rajapinnasta

Viholliset ja muut actorit joihin pelaaja voi vaikuttaa koodeilla perivät edellä mainitun CodeInterface-luokan. Nämä kuitenkin ylikirjoittavat CheckCodeMatch-funktion sopimaan omaa rooliaan koodin ollessa oikein. Vihollisroboteilla koodin ollessa oikein ne tuhoutuvat mutta tykeillä, ovilla ja hisseillä koodin kirjoittaminen aiheuttaa vaihdon kahden eri tilan välillä. Näitä tiloja voi olla oven avautuminen ja sulkeutuminen tai tykillä sen ollessa aktiivinen tai lepotilassa jolloin se ei voi ampua. Vihollisten kohdalle käytössä on yksi kantaluokka, josta aliluokat perivät suurimman osan käytöksestään. Vihollistyyppikohtaiset tekoälykäyttäytymiset ohjelmoidaan näihin aliluokkiin erikseen.

5.3 Pelimoodi

Pelimoodi on ydinosa peliä. Se muun muassa seuraa pelin alkamista ja loppumista sekä määrittää voittotavoitteet. Tärkeänä perusasetuksena pelimoodissa voi määrittää mikä on oletushahmo-luokka (default pawn class) joka pelaajalle luodaan pelimaailmaan sekä mitä hahmokontrolleria (character controller) se käyttää.

Projektissa pelimoodin haluttiin myös hoitavan pelin alkaminen ja loppuminen voittoineen tai häviöineen sekä pelatun ajan mittaaminen ja päivittäminen HUD:iin. Peli alkaa lähtölaskentaan jonka aikana pelihahmon liikkuminen on estetty. Kun lähtölaskenta on suoritettu, pelihahmon liikkuminen sallitaan ja peliaikaa aletaan mitata. Jos pelaaja avaa taukovalikon menee pelimoodille tieto, jolloin ajan mittaaminen keskeytetään ja valikon sulkeuduttua jatketaan mittaamista. Peliaika pelimoodissa on tallennettu float tyyppiseen muuttujaan ja se muotoillaan merkkijonoksi ennen lähettämistä HUD:iin.

Kun pelihahmo läpäisee kentän, menee tästä tieto pelimoodiin joka lopettaa ajan mittaamisen ja estää pelihahmon liikkumisen. Ruudulle ilmestyy pop-up, josta pelaaja näkee peliaikansa ja tulostaulukon sekä painikkeet pelata kenttä uudestaan tai palata päävalikkoon.

Pelaaja häviää pelin, jos hänen elämänsä laskevat nollaan. Tämän käydessä pelihahmo lähettää tiedon pelimoodiin joka lopettaa ajan mittaamisen ja estää pelihahmon liikkumisen samalla tavalla kuin kentän läpäistäessä. Tällä kertaa kuitenkaan ilmestyvä pop-up ei pidä sisällään peliaikaa eikä tulostaulukkoa.

5.4 HUD ja käyttöliittymä

Käyttöliittymän voi jakaa kahteen kategoriaan. Pelaajan pelatessa tarjoaa pelaajalle reaaliaikaista informaatiota pelistä, kuten kuvasta 10 ilmenee. Toisena kategoriana on valikot kuten päävalikko tai taukovalikko.

Tässä pelissä HUD:n ei tarvitse tarjota paljoa informaatiota pelaajalle. Sitäkin tärkeämpänä sen on oltava mahdollisimman huomaamaton, eikä peittää ruutua. Ne tiedot mitä HUD kuitenkin tarjoaa, ovat montako elämää pelaajalla on, sekä näyttää ajan, joka pelaajalle on mennyt kentän suorittamiseen. HUD:sta löytyy myös painike, joka avaa taukovalikon.

Valikot haluttiin pitää mahdollisimman yksinkertaisina ja selkeinä. Päävalikosta asetuksiin, josta pystyy säätämään musiikin ja ääniefektien äänenvoimakkuutta sekä mahdollisesti muuttamaan graaffista kuvanlaatua. Kuvanlaadun valintatarvetetta ei ole vielä päätetty ja asiaa tullaan arvioimaan beta-testauksesta saadun palautteen avulla. Päävalikosta pääse myös listaan pelin kentistä. Pelaajan valitessa kentän ruudulle aukeaa pop-up jossa play-painike sekä tulostaulukko. Taukovalikossa on painikkeet asetuksiin, päävalikkoon ja resume, joka jatkaa peliä.

Käyttöliittymät UE4:ssa toteutetaan käyttäen UMG:tä (unreal motion graphics). UMG:tä voi käyttää joko C++-koodissa tai Blueprinteissä. Työssä päädyttiin käyttämään Blueprinttejä UMG:n kanssa niiden sisältävän logiikan ollessa hyvin yksinkertaista eikä näin vaikuttavan merkittävästi suorituskykyyn. Toinen syy Blueprinttien valinnalle käyttöliittymien luontiin oli välttää turha odottelu Visual Studion kääntäessä koodia.

Kuva 10. Yksi versio HUD:n asettelusta

5.5 Google Play

Google Play-palveluiden integraatio löytyy valmiina UE4:sta. Käytettävinä palveluina löytyvät saavutukset (achievements) ja tulostaulu (leaderboard). Näitä ominaisuuksia varten tarvitaan 25$ hintainen Android kehittäjätili.

UE4:ssa projektin asetuksista pitää ottaa käyttöön Google Play tuki ja syöttää Android kehittäjäkonsolista löytyvät sovellus id ja lisenssiavain. Tämän jälkeen lisätään listaan konsolissa luodut saavutukset ja tulostaulut niiden nimen ja id:n perusteella. Palvelujen käyttö on yksin kertaista ja tarvittavat Blueprint solmut niiden käyttöä varten on valmiina.

5.6 Mallinnus

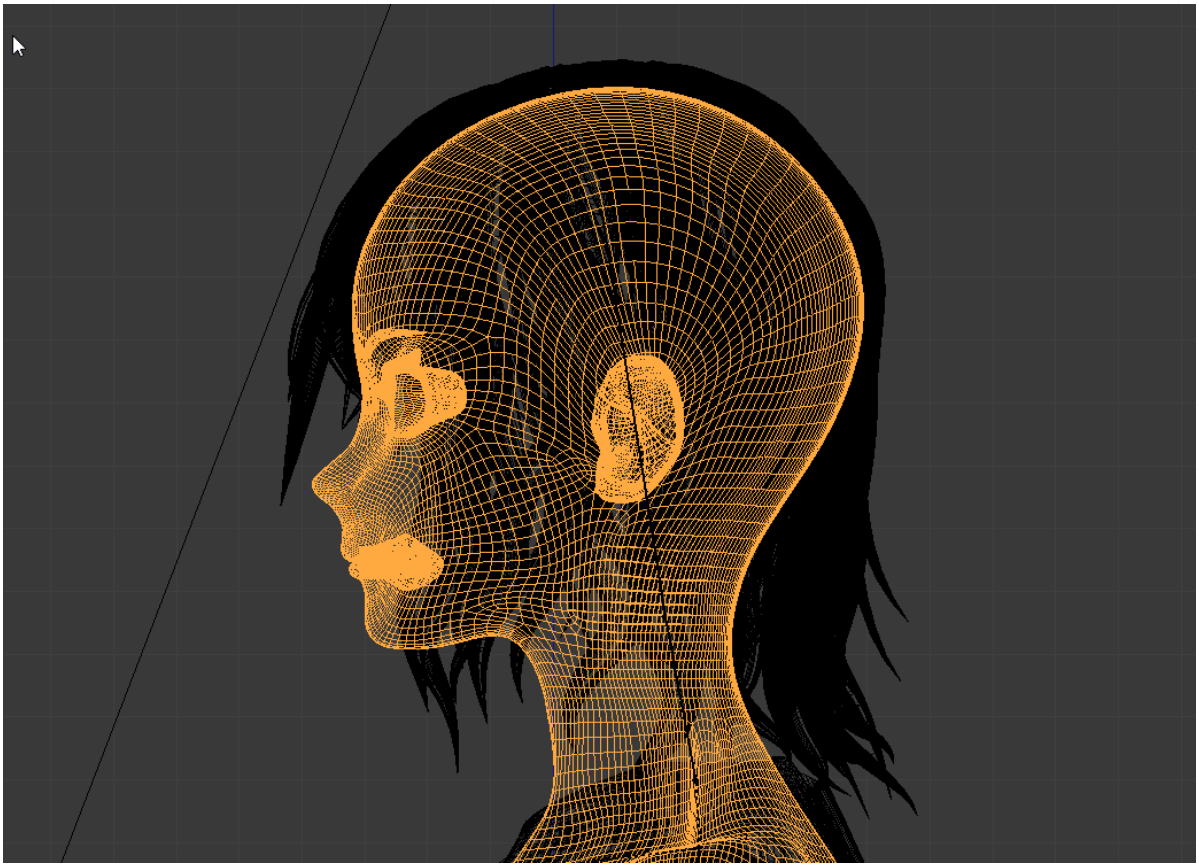
Yleisesti hyvä tapa aloittaa mallinnus, on tarkastella ja tutkia lähdemateriaalia, joka jollain tavalla liittyy mallinnettavaan asiaan. Tällöin saa hyvän kuvan siitä, mitä tekniikoita tai tyyliä mallintamiseen mahdollisesti voisi käyttää. Koska työssä valittiin käytettäväksi hyvin piirrosmainen tyyli, ja kaikki hahmot ja esineet renderöidään 2d kuvina, myös valitut tekniikat mallinnukseen ja muihin pääosiin on valittu sen mukaisesti

Varsinainen työ alkaa tämän vaiheen jälkeen. Opinnäytetyössä valittiin käytettäväksi mallinnustekniikaksi polygonimallinnus, sen helppouden ja yksinkertaisuuden vuoksi. Työssä ei myöskään tarvita mahdollisimman realistista tai tarkkaa jälkeä malleihin, johon sculptaus soveltuisi paremmin. Ennen kuin itse mallia aletaan työstämään, täytyy lisätä pari modifieria. Nämä ovat mirror (peilaus) x-akselilla, sekä subdivision, mieluiten vähintään asteikolla 1, eli tuplana. Tätä määrää voi myöhemmin suurentaa tarvittaessa, ja todennäköisesti pelkästään tuplaaminen ei riitä hyvän geometrian saavuttamiseksi.

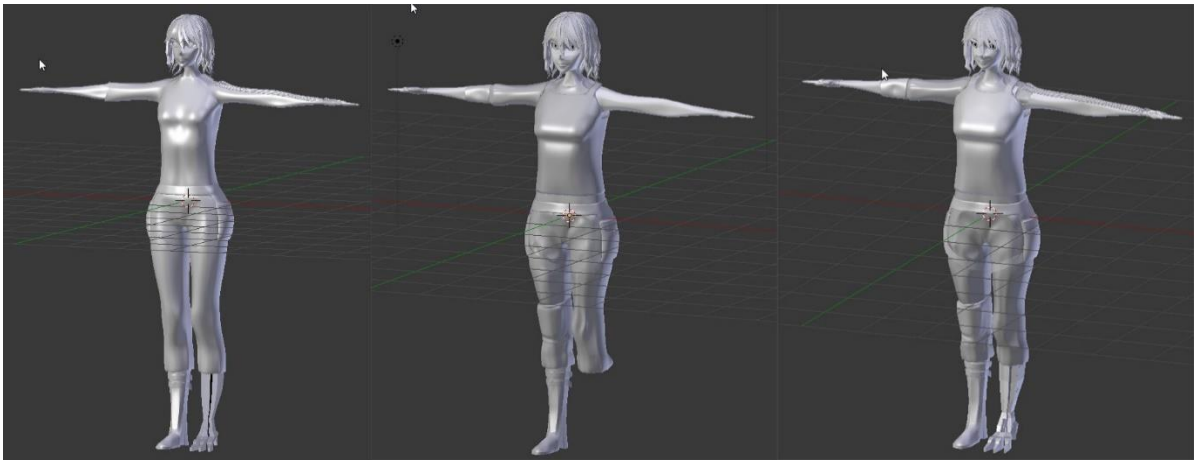
Geometriasta puheenollen, mallille mitä työstetään, olisi hyvä saada mahdollisimman sulava ja luonnollinen meshin geometria tai topology. Tällä tarkoitetaan siis sitä, että mallin edge flow olisi mahdollisimman luonteva ja looginen mallin muotoihin nähden. Esimerkiksi ihmishahmon kanssa on tärkeää kiinnittää huomiota siihen, miten ihmiskehon lihakset muotoutuvat ja liikkuvat, ja perustaa suunniteltu meshin topology tämän mukaisesti. Tämän lisäksi, on hyvä ottaa huomioon myös, kuinka suurella tarkkuudella (eli kuinka paljon polygoneja tietyssä paikassa on) tietyt paikat kehosta tehdään, näihin kuuluvat yleensä paikat, joissa tapahtuu paljon venymistä tai omaavat suuren liikeradan. Aikaisimmin mainittuihin paikkoihin kuuluvat muun muassa kasvot (ks. kuva 11), polvi- ja kyynärtaipeet, olkapäät, sekä muut paljon taipuvat osat, esimerkiksi sormet ja ranne. Hyvän edge flowin ja oikeiden tarkkuuksien saavuttaminen takaa mallin sulavan deformoinnin, yhdessä oikein tehdyn skinningin kanssa. Tämä edge flowin saavuttaminen pätee myös muihin orgaanisiin malleihin, sekä muutamilla muokkauksilla, myös epäorgaanisiin. (Figgins.). (Thorn 2013, 148.)

5.6.1 Hahmot ja aktiiviset objektit

Työssä valittiin tehtäväksi pelin pelaajan hahmo ensimmäisenä, joten tämän keho mallinnetaan ennen muita, kuvasta 12 näkee hahmon eri vaiheita. Vaikkakin kyseessä on piirroshahmo, on kuitenkin viisainta seurata ihmiskehon mallista, miten mallintaa kaikki oikein. Tämä tapahtuu helposti etsimällä referenssikuvan, jonka pystyy asettamaan 3d-näkymän taustalle. Kuvan seuraaminen tapahtuu helposti muuttamalla näkymässä oleva objekti wireframe (ristikko, kuten kuvassa 11) näkymälle, tällöin objektissa näkyy pelkästään sen faceja yhdistävät reunat.



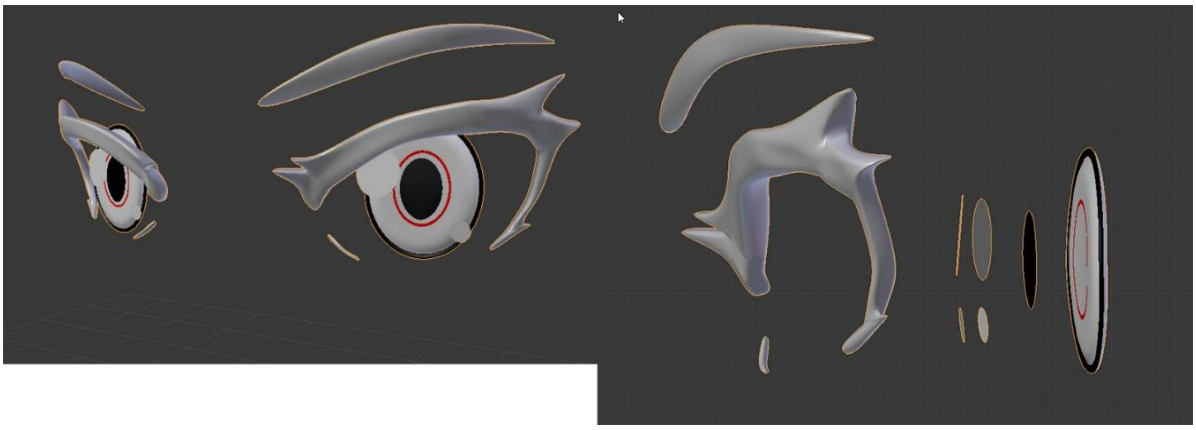
KUVA 11. Wireframe mallin päästä



KUVA 12. Eri mallinnusvaiheita päähahmosta

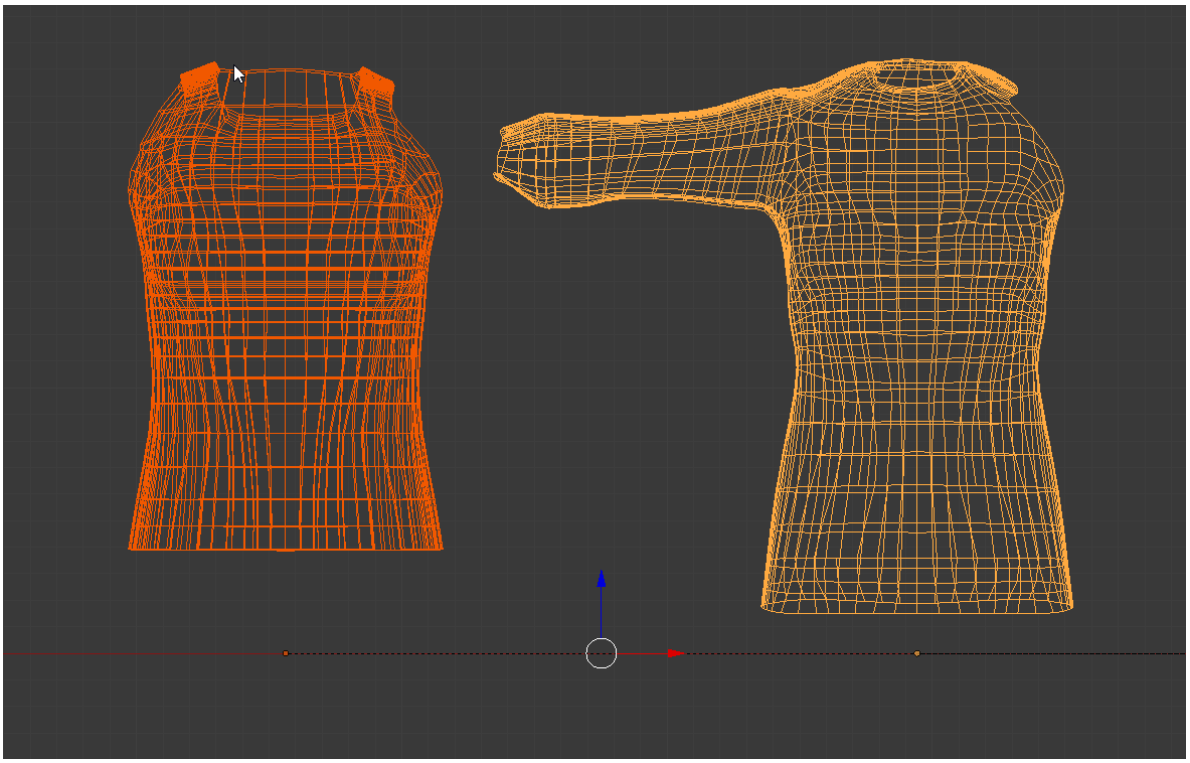
Kehon jälkeen hahmolle tehtiin kehosta vielä puuttuvat osat, eli hiukset ja silmät. Näitä molempia on useampi tapa tehdä, riippuen mihin mallia käytetään. Tässä tapauksessa hiukset tehtiin Bézier-käyriä käyttämällä. (Farin ym. 2002) Näihin voi määrittää bevel ja taper objektit, joiden mukaan käyrän muoto, paksuus ja leveys muokkautuu, kuvassa 13 näitä käyriä, sekä bevel ja taper osia. Tämän Bézier-käyrän voi sitten automaattisesti muuntaa käytettäväksi meshiksi. Silmät perinteisesti tehdään yksinkertaisesti joko yhdellä tai kahdella pallon muotoisella meshillä, mutta piirroshahmojen tapauksessa on parempi käyttää eri tapaa. Hahmolle luotiin silmäkuoppa, joka värjätään halutulla värillä, oletettavasti valkoisella. Silmäkuopan edustalle tehtiin hiukan reunoilta kaareva, ellipsin muotoinen kiekko, joka tässä tapauksessa on itse silmä. Tällä tavalla tehty silmä on hyvinkin piirrosmaisen näköinen, ks kuva 14.

KUVA 13. Bézier -käyrien muotoja



KUVA 14. Silmän osat

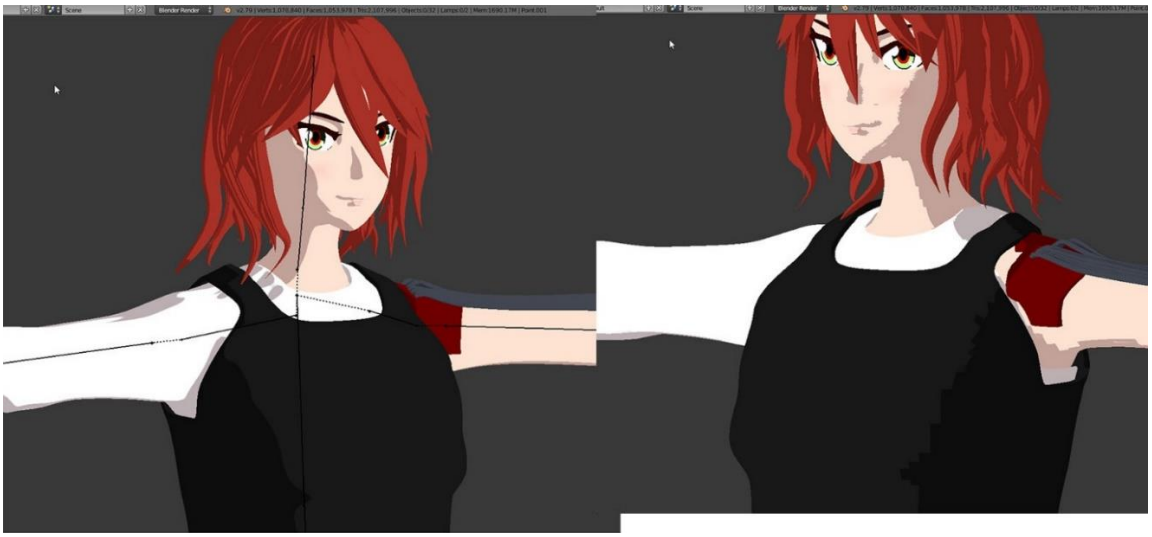
Seuraavaksi vuorossa ovat kaikki vaatteet tai varusteet, mitä hahmolle halutaan. Ehkäpä helpoin tapa, jos sculptausta ei käytetä, on tehdä duplikaatti juuri tehdyn hahmon meshistä, halutusta kohdasta ja muokata sitä tarpeen vaatiessa, kuten kuvassa 15. Suurin osa hahmon varusteista tehtiin tällä tavalla (paita, housut, käsineet, liivi, suojukset, kenkä), muut varusteet mallinnettiin erikseen ja sovitettiin hahmolle sopiviksi. Kaikki nämä varusteet tehdään samalla mallintamisperiaatteella kuin itse hahmo.



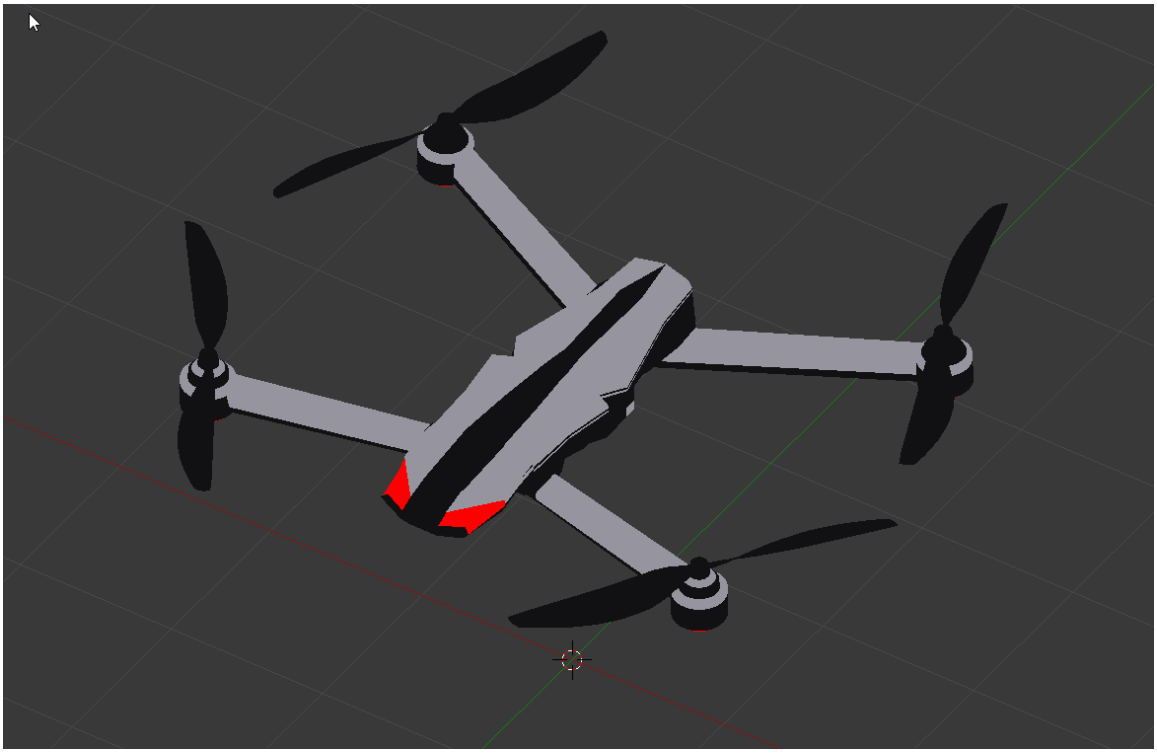
KUVA 15. Mallin base meshin kopiointia toiseksi varusteeksi

Muiden hahmojen tai aktiivisten eli liikkuvien objektien, jotka ovat orgaanisia, työnkulku on pitkälti sama mitä päähahmolla. Mallinnuksen eroavaisuuden tulevat vastaan näiden ollessa epäorgaanisia, kuten tämän työn vihollisten suhteen, jotka ovat pitkälti robotteja tai muita mekaanisia laitteita. Näi-

hin sovelletaan muuten samaa työnkulkua kuin orgaanisiin, paitsi eri shading (ks. kuva 16), tarvittaessa eri lisätyt ominaisuudet, sekä erilainen meshin tekotapa, jolla malliin yritetään saada hyvin tasaiset ja terävät pinnat, kuvassa 17 yksi näistä vihollisista. Lisätyistä ominaisuuksista erilaisia voi olla esimerkiksi bevel ja solidify. Bevel ”tasoittaa” tai terävöittää mallin jyrkemmät reunat, riippuen asetuksista mitä tähän laittaa, lisäämällä reunoihin geometriaa. Solidify nimensä mukaisesti tekee objektista kiinteän luomalla siihen lisää geometriaa / polygoneja. Tämä ei tarkoita, että objekti olisi oikeasti volyymiltaan kiinteä, vaan että se on mallillisesti kiinteä tai suljettu, mutta ontto sisältä.



KUVA 16. Vasemmalla smooth ja oikealla flat shading



KUVA 17. Yksi pelin vihollisista

5.6.2 Staattiset objektit ja kenttä

Staattisten objektien mallinnus ei perusteiltaan eroa hahmojen mallinnuksesta paljoakaan. Näiden mallintamisessa ei tosin tarvitse yleensä ottaa huomioon, miten meshin edge flow deformoituisi parhaiten, sillä nimensä mukaisesti niillä ei tule olemaan animaatioita. Näin tekijä voi keskittyä optimoimaan itse polygonien määrän mahdollisimman matalaksi. Kuitenkin niin, että malli pysyy tarpeeksi hyvännäköisenä.

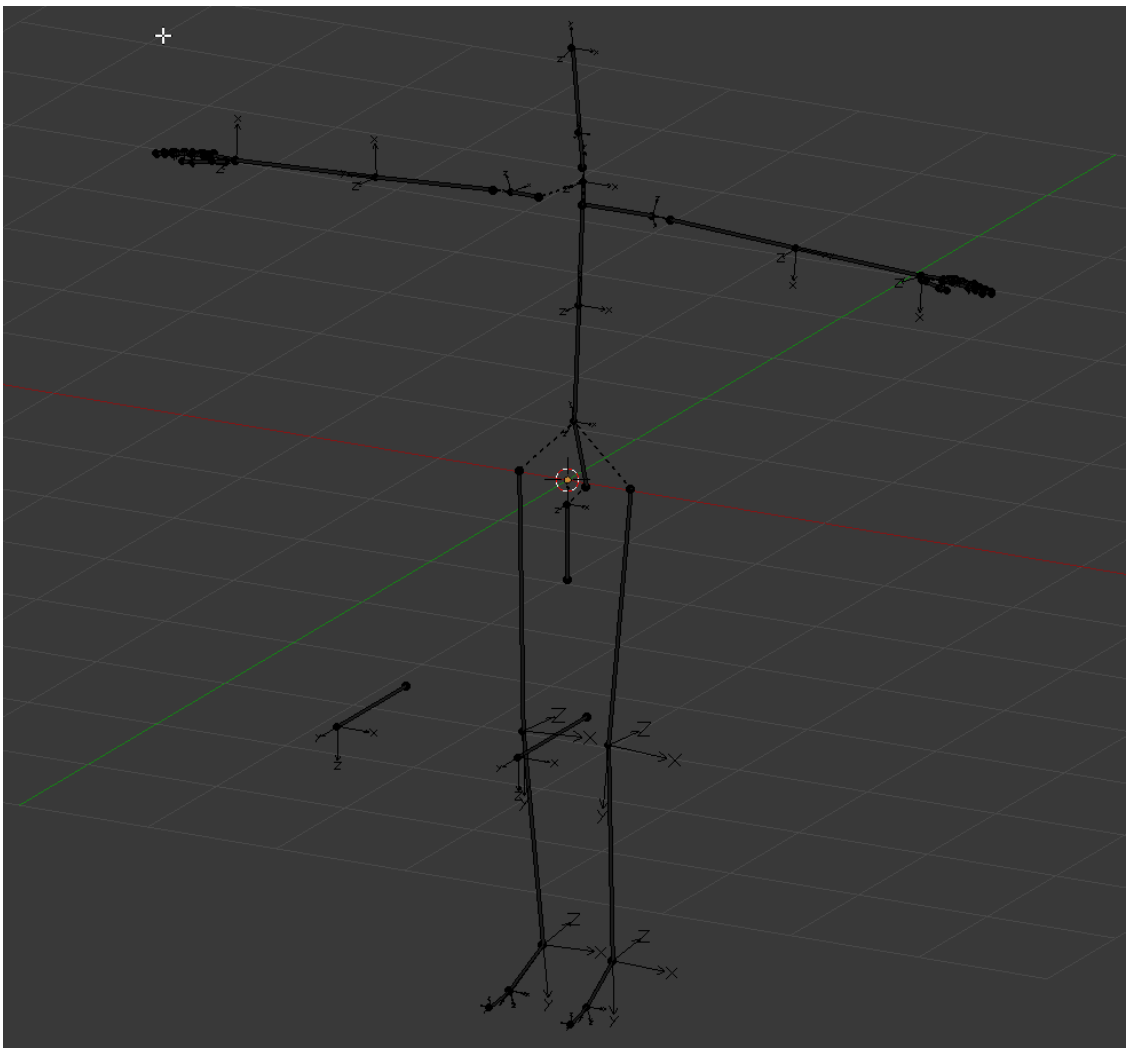
Suurin osa näistä staattisista objekteista on kentässä olevia ”proppeja”, esimerkiksi tuoleja, pöytiä, valoja tai kasveja. Kuten hahmot, nämä pidetään erillisenä varsinaisista kentän ”rakennusosista”.

Pelin kentät rakennetaan blenderin puolella modulaarisiksi kentän palasiksi, jotka Unreal Enginessä yhdistetään kattavaksi kokoonpanoksi. Mallinnuksen prosessi on samanlainen, kuin objektien kanssa. Kentän osia olisi tarkoitus luoda useampia, jotta kenttiin saadaan aikaan tarpeeksi variaatiota, ja näihin voi vaihdella proppeja tarpeen vaatiessa.

5.7 Rigging & skinning

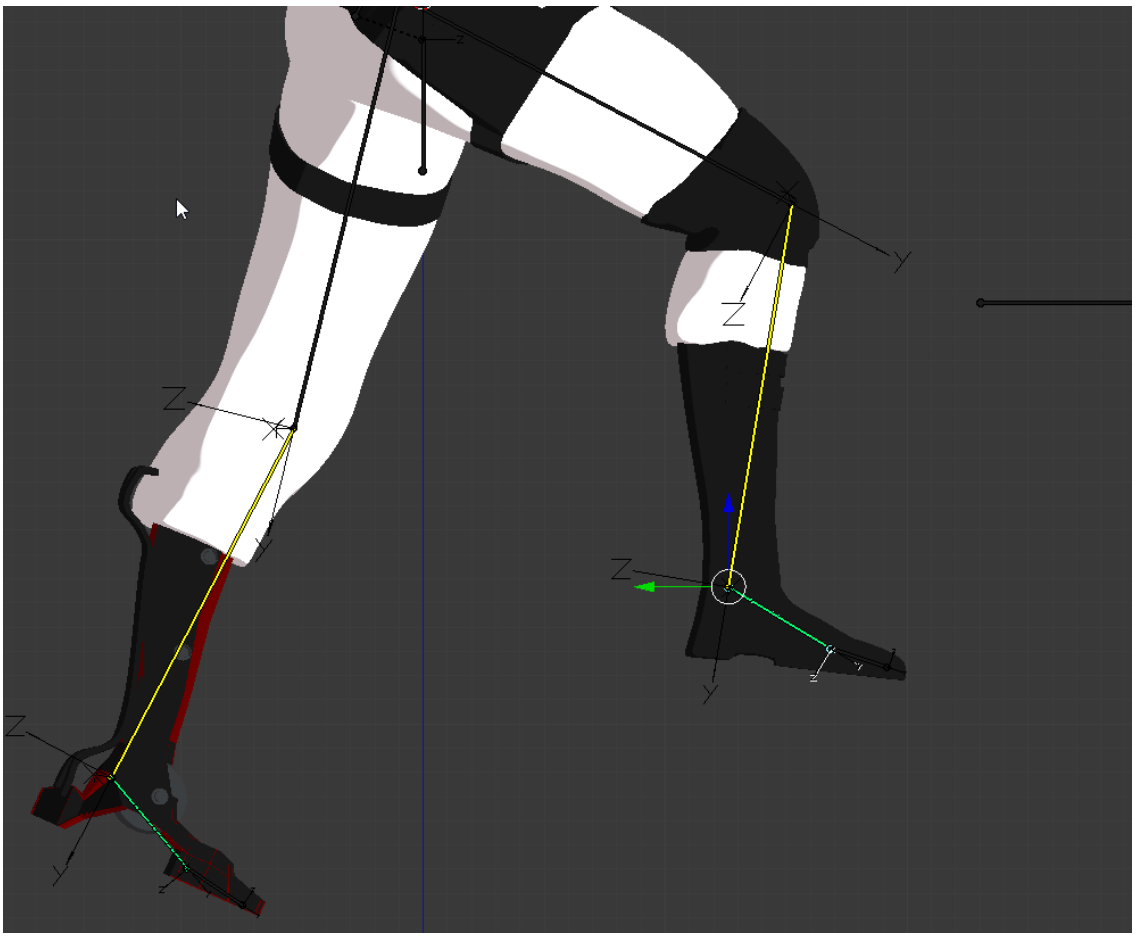
Rigging-nimi pohjautuu aikasemmin mainittuun rig (armature) objektiin, ja on prosessi, jossa tämä kyseinen rig luodaan. Rigillä pyritään matkimaan mallin kuviteltua luurankoa, joka kuvastaisi miten mallin eri osat liikkuvat. Yksinkertaisimmillaan koko rig voi olla pelkästään yksi ”luu”, joka liikuttaa joko koko mallia tai tiettyä osaa siitä. Rigillä tulee aina olla juuri, eli luu, josta koko rigiä on mahdollista liikuttaa animoinnin aikana. Luonnollisesti mitä enemmän mallille tarvitaan liikerataa tai riippuen, kuinka monimutkainen kyseinen malli on, sitä enemmän luita kyseinen rig tarvitsee. Kuvassa 18 esimerkki pelin ihmishahmojen rigistä.

Tässä työssä tehtiin muutama erilainen rig, josta yhteen käytettiin blenderiin saatavilla olevaa rigify -lisäosaa. Tätä rigifyssa valmiiksi olevaa ”metarig” rigiä käytettiin muutamilla muokkauksilla päähahmon riginä. Kyseistä lisäosaa käytettiin, sillä perusteella, että se säästää jonkin verran aikaa, kun hahmon rigiä ei tarvitse tehdä kokonaan alusta asti. Muiden mallien rig tehtiin blenderin perus armature objektilla. Armaturehan aloittaa vain yhdestä luusta, jota hallitaan hyvin samalla tavalla, kuin mallin meshiä polygonaalisessa mallinnuksessa. Eli luusta voidaan valita jompikumpi tämän päistä, josta extrudataan (luo toisen luun, joka on kiinnitetty alkuperäiseen, sama periaate kuin polygonien vertex ja niiden välinen edge) uusi luu, prosessi toistetaan tarpeen mukaan. Näiltä luilta voi myös muokata kokoa, ominaisuuksia, rotaatiota tai positiota. Myös muut mallin meshiin käytetyt ominaisuudet toimivat, kuten peilaus, joskin hiukan eri tavalla. Luut usein myös nimetään, ihmiskehossa vastaavan sopivimman luun nimityksellä, jos luita on useampi, on ne hyvä numeroida. Myös symmetristen luiden tapauksessa, on hyvä erotella näiden nimitys L tai R merkinnällä, vastaten vasenta ja oikeaa puolta. Kun kaikki tarvittavat luut on luotu, ne asetellaan mahdollisimman tarkasti paikoilleen mallin meshin sisään. Myös ennen skinningin aloittamista on tärkeää tarkistaa luiden rotaatio, mallin kaikkien taittuvien osien hallitseva suunta olisi hyvä olla sama kaikilla luilla.



KUVA 18. Päähahmon rig / armature

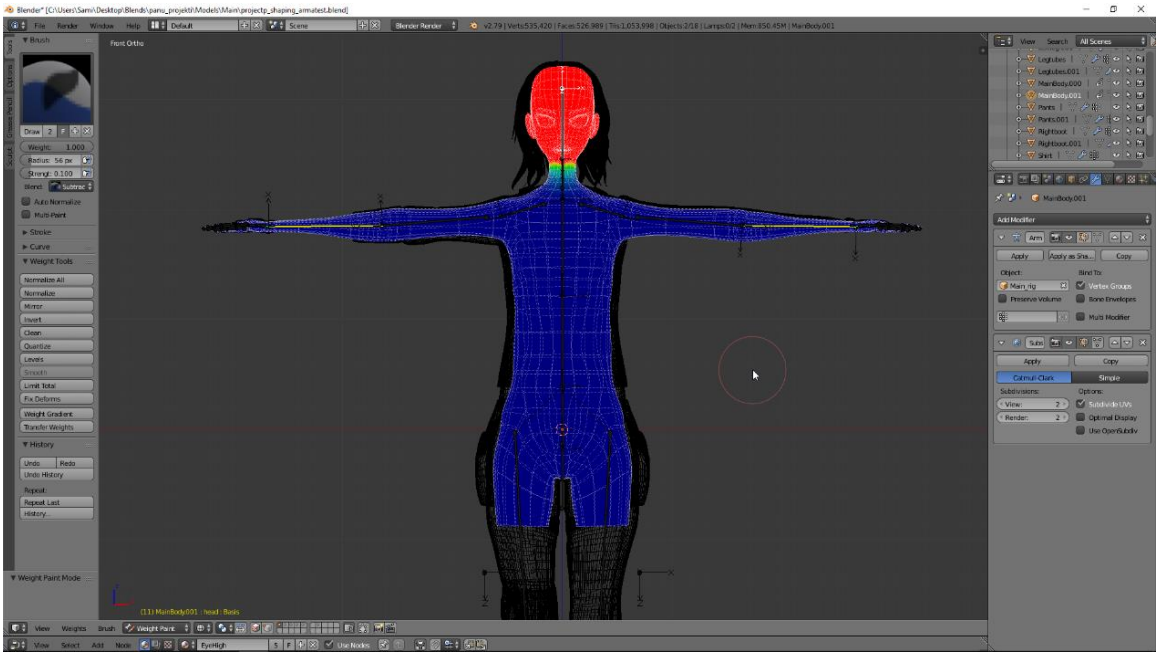
Näiden asioiden lisäksi, yksi animoinnin työtaakkaa hyvinkin helpottava, ja osittain realismiakin luova ominaisuus, on IK-ketju. IK eli Inverse Kinematics on algoritmi, joka laskee sille merkatuiden luiden vaaditun position, yhden ”ankkuri” luun mukaan siten, että tämä ankkurina toimiva luu on sille määritetyssä kohdassa (Bermudez 2017), ks kuva 19. Tämä on siis päinvastainen algoritmi normaaliin luiden liikutukseen, jossa ne yksitellen asetetaan paikoilleen. Työssä IK-ketjua käytettiin jaloille ja käsille. Blenderin IK-ominaisuudessa on myös valinta käyttää kyseiselle ketjulle ”pole” luuta. Tämä toimii pitkälti suuntimena esimerkiksi polville ja kyynärpäille, työn tapauksessa, joihin IK-ketju pyrkii suuntaamaan rotaationsa. Yleisesti IK-algoritmia käytetään myös sormille tai muille taipuville osille, joissa on useampi nivel.



KUVA 19. Jalkojen luut seuraavat IK -ketjun mukana, Vihreällä merkattu liikuttaa ketjua

Skinning suoritettiin aluksi blenderissä olevalla automaattisella painotuksella, joka luukohtaisesti määrittää mallin meshin vertexien painotusarvot luille. Nämä painotusarvot kontrolloivat, kuinka paljon kyseisen luun liikuttaminen vaikuttaa sille määrättyyn meshin osaan, toisinsanoen kuinka paljon se deformoi meshiä. Painotusarvot näkyvät painotustyökalunäkymässä eri väreinä, sininen on 0, vihreä 0,5 ja punainen 1, ks. kuva 20. Blenderin automaattinen painotus ei tosin ole kovin hyvä painottamaan ihmishahmon osat, varsinkin suuresti taipuvat kehonosat, kuten sormet, ranteet, olkapäät ja polvet. Nämä täytyy yleensä tehdä itse melkein kokonaan, ja muutakin ”siivousta” automaattisesta painotuksesta täytyy tehdä. Tämä toiminto kuitenkin säästää jonkin verran aikaa, varsinkin jos täysin oikeanlainen deformointi ei ole tarpeen. Hyvän skinningin tavoitteena on siis saada hahmo deformoitumaan mahdollisimman luonnollisella tavalla, ilman merkittäviä virheitä (Figgins).

Blenderissä on kaksi päätapaa tehdä skinning prosessi, automaattisen lisäksi. Tässä tapauksessa armaturen luut yhdistetään malliin ilman automaattista painotusta, mutta samalla luoden luita vastaavat tyhjät vertex groupit (ryhmät). Näihin ryhmiin tallennetaan painotusinfoa joko ”maalaamalla” blenderin weight paint työkaluilla, yksittäinen luu valittuna, tai manuaalisesti valitsemalla vertexit, asettaen näille painoarvon, joka tallentuu valittuun vertex grouppiin.

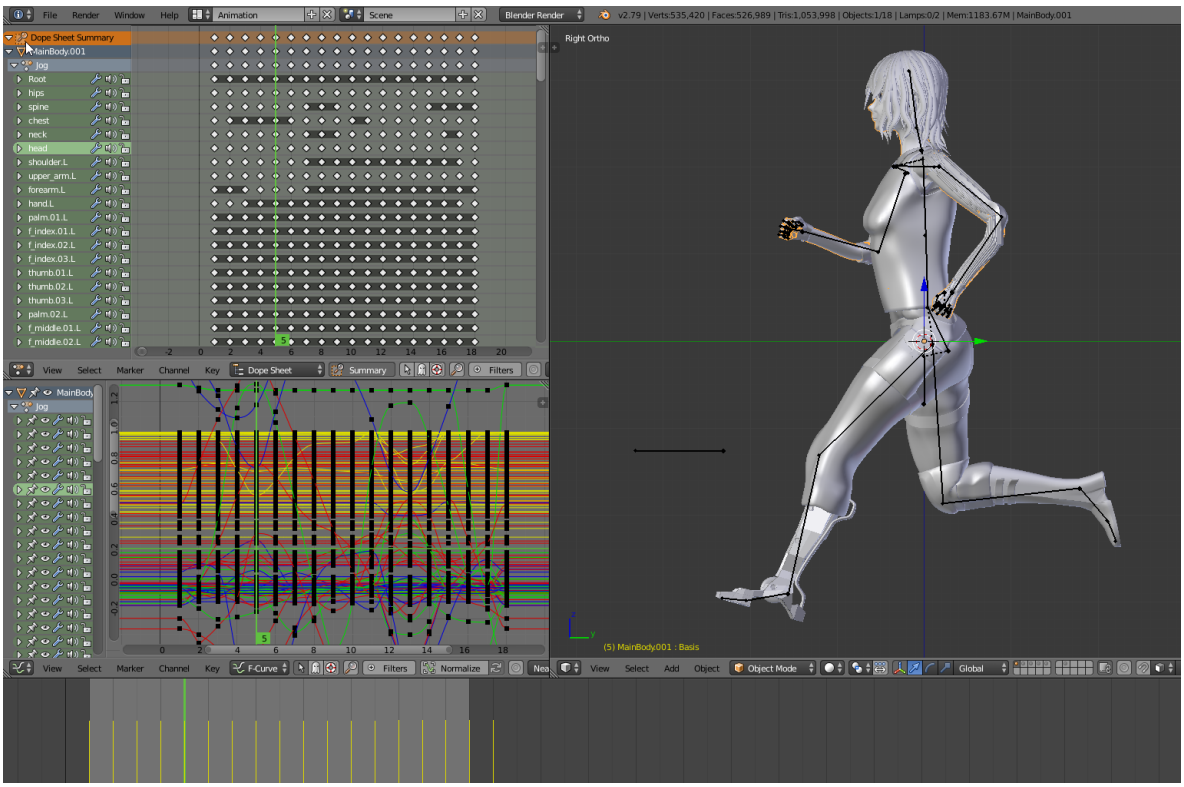


KUVA 20. Blenderin weight paint -työkalut, pään luu valittuna

5.8 Animointi

Animoinnin perusta on pitkälti sama, kuin millä tahansa liikuvalla kuvalla tai videolla, eli katsojalle näytetään tietty määrä kuvia sarjassa, tietyn ajan sisällä. Tämä tunnetusti luo vaikutelman liikkuvasta kuvasta.

Ohjelmistoissa, joilla animointia voi tehdä, prosessi tosin on erilainen, vaikkakin perusperiaate onkin sama. Kuvien sijasta tallennetaan rigin jokaisen yksittäisen luun, koordinaatistossa lokaatio ja rotaatio (x,y,z,-x,-y,-z) arvot niin sanotuiksi Key (avain) arvoiksi. Tällä tavalla tallennetaan yksi asento kerrallaan, joka vastaa yhtä framea (kuvaa). Kuten kuvassa 21, yksi frame valittuna. Prosessi toistetaan, kunnes saadaan aikaan kokonainen animaatio, joka on tarkoitukseen tarpeeksi sulava. Malliahan liikutetaan aikasemmin mainituista rigin luista, näille voi myös määrätä erilaisen muodon, Custom bone shape, joka nimensä mukaisesti asettaa luulle toisen objektin muodon. Custom bone shape voi helpottaa animointia helposti nähtävillä muodoilla tai esimerkiksi osottaen mihin suuntaan luun pääsuunta osoittaa. Työssä käytettiin vain blenderin valmiiksi tarjoamia muotoja (stick), sekä axis ja x-ray valintoja, jotka vastaavasti tuovat koordinaatiston suunnat näkyviin luusta ja antaa koko rigin näkyä mallin läpi.



KUVA 21. Yksi juoksuanimaation keyframe

5.9 Teksturointi ja renderöinti

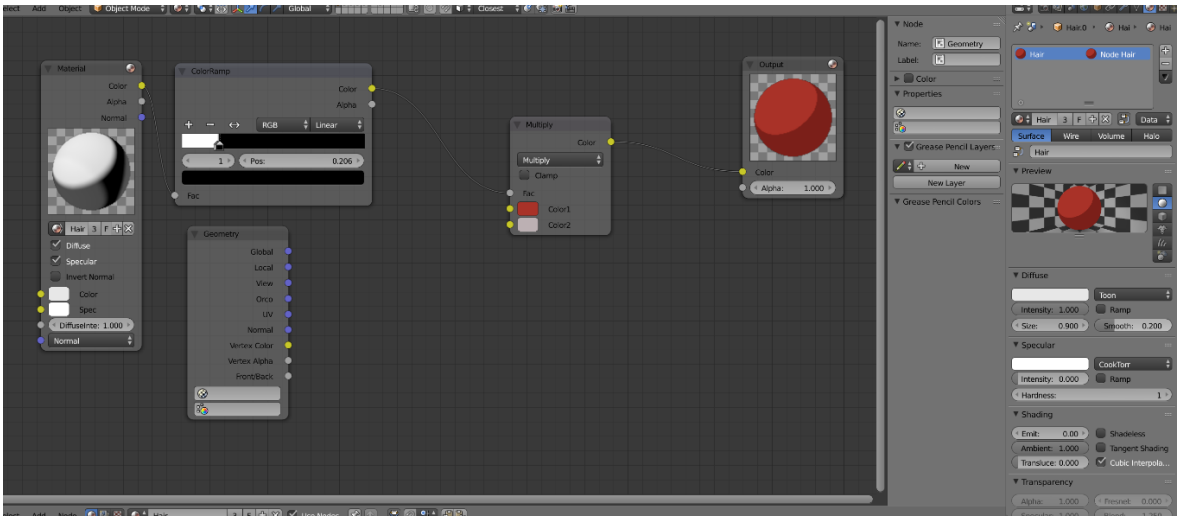
Materiaaleihin tässä työssä käytettiin pelkästään yksinkertaisia, yksittäisiä värejä. Näihin sovelletaan eri efektejä node-editorilla, jotta aikaan saatu väritys olisi piirrosmainen. Varsinkin varjostus oli tärkeää saada oikeanlaiseksi, sekä toimimaan blenderin valo- ja varjosimulaatioiden kanssa.

Työssä käytettiin kahta eri tapaa värjätä hahmon mesh. Yksinkertaisempi tapa on tehdä materiaali, asettaa tälle tietty väri, ja valita meshistä alueet, joihin kyseinen materiaali lisätään. Oletuksena materiaali menee koko objektin pinnalle. Toinen tapa mitä työssä käytettiin itse hahmon ihon värjäämiseen, on blenderin vertex paint -työkaluja. Näillä työkaluilla on mahdollista värjätä yksittäisiä vertexejä mallista tietyllä värillä, iho värjättiin tällä tavalla, koska hyvän tuloksen saavuttamiseksi, siihen pitää paikoitellen sekoittaa eri värisävyjä. Tämä ei joko ole mahdollista pelkästään materiaalin väriä vaihtamalla, tai on huomattavasti vaikeampaa.

Näille materiaaleille luotiin node-editorin avulla shaderit, jotka renderöidessä muuntavat materiaalin väriä riippuen valon määrästä, täten luoden varjostuksen. Tämä efekti on hyvin samankaltainen kuin cel-shadingissä.

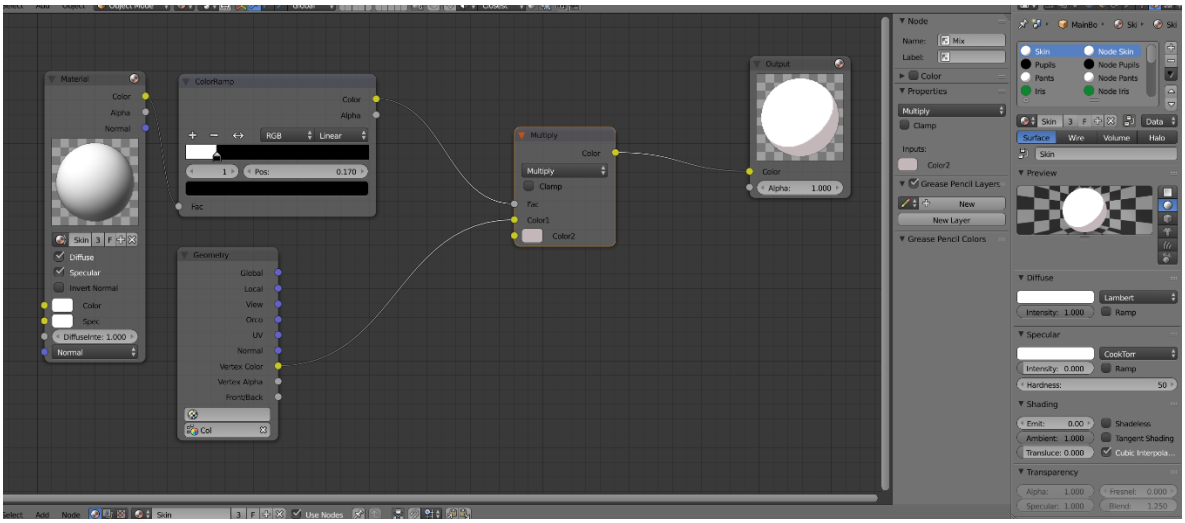
Työssä valittiin renderöintiin käytettäväksi blenderin sisäistä blender renderiä tämän helppokäyttöisyyden ja tarkoitukseen sopivuuden vuoksi (Blender Foundation). Piirroshahmojen renderöinti cycles renderillä vaatii monien ominaisuuksien disabloimista, sekä renderereiden nodet ovat osittain erilaiset. Työhön käytetty node setup ei sovellu suoraan käytettäväksi cyclesilla. Käytetyssä node setupissa otetaan ensin materiaalista väriarvo, joka syötetään ColorRamp nimiselle nodelle. Tämä node toimii varjostuksen värjääjänä. Seuraavaksi yhdistetään tästä nodesta väriarvo Multiply noden

fac (factor) input arvoon, tässä Multiply nodessa syötetään myös materiaalin pääväri. Lopuksi Multiply nodesta syötetään jälleen väriarvo Output nodelle, joka on varsinainen aikaansaatu väri, kuten kuvassa 22.



KUVA 22. Node setup

Vertex paint -työkaluilla värjätty iho eroaa hiukan aikaisemmin läpikäydystä setupista. Sen sijaan, että Multiply nodessa päätettäisiin pääväri, syötetään tämän värikenttään Geometry noden vertexin väriarvo. Aikaansaatu väri tässä tapauksessa ottaa vastaan vertexeihin värjätyt arvot. Kuten kuvassa 23, erona juurikin yhdistetty geometry node.



KUVA 23. Ihon node setup

Lopuksi hahmosta renderöidään animoinnin kaikki framet eli kuvat, joita käytetään moottorin puolella spritejen tekemiseen. Blenderin renderöinnin asetuksista mainittakoon ”freestyle”, kyseinen asetus piirtää renderöinnin yhteydessä mallin siluetille ja kaikille reunakohdille piirrettymäiset viivat, kuten kuvissa 24, 25 ja 26. Näitä viivojen piirroskohtia voi myös itse merkata tai poistaa mallin meshin mukaisesti. Kaikki kuvat renderöidään PNG-muodossa, sillä se tukee alpha -kanavaa, toisinsanoen läpinäkyvyyttä. Taustan täytyy olla läpinäkyvä, jotta spritet muodostuvat oikein.

KUVA 24. Node setupin vaikutus varjostukseen

KUVA 25. Vertaus freestylesta. Vasemmalla ilman, oikealla kanssa

KUVA 26. Lopputulos päähahmosta

6 POHDINTA

Opinnäytetyössä aloitetun pelin kehittäminen tulee jatkumaan. Seuraavaksi tarvitaan lisää kenttiä ja käyttöliittymän ja muiden toimintojen hienosäätöä ennen beta-testausta. Beta-testauksessa haetaan käyttäjäkokemusta (user experience) ennen pelin julkaisua. Sitten voikin pelin jälkituen ohella alkaa suunnitella seuraavaa projektia. Opinnäytetyön aikana pelinkehitystaidot kasvoivat huomattavasti niin itse opinnäytetyönä tehdyn pelin, kuin myös kuluneen vuoden aikana osallistuttujen game jamien ansiosta. Näiden ja tulevien taitojen avustamana tekijät toivovat tulevaisuudessa pystyvänsä lunastamaan paikkansa peliteollisuudessa tavalla tai toisella.

Ongelmiakin työn tekemisessä tuli vastaan. UE4:n C++ API kehittyy ja muuttuu joka versiolla, joten kaikki ohjeet eivät enää pidä paikkaansa. Rajapinnan ohjelmointi tuotti erityisesti ongelmia tästä syystä. C++-koodin kääntäminen voi viedä useammankin minuutin vanhemmilla prosessoreilla. Suositeltavaa jättää hienosäätö Blueprintteihin, jollei halua odotella jokaisen pienen muutoksen jälkeen koodin kääntymistä. Hidas työskentely, sekä eri Game Jameihin osallistuminen venytti aikataulua huomattavasti pidemmälle, kuin oli alun perin suunniteltu. Jos opinnäytetyön tekisi uudestaan, olisi hyvä pitää paremmin kiinni aikataulusta.

Peli ei edennyt julkaistavaksi opinnäytetyön aikana, mutta käydään hiukan läpi, mitä julkaiseminen olisi vaatinut. Kuten Google Play-palvelujen kohdalla mainittiin, vaatii julkaiseminen Android kehittäjätilin. Ennen julkaisuakin voi Google Play:n laittaa pelin alpha- ja beta-versiot helpottamaan versioiden jakamista testaajille. Julkaisun lähestyessä kannattaa suunnata lukemaan Google Play:n ohjeistusta julkaisusta ja ylläpidosta. Laajasta ohjeistuksesta löytyy tarvittavat ohjeet muun muassa julkaisuun, ohjelman päivittämiseen ja lokalisointiin. (Google.)

LÄHTEET JA TUOTETUT AINEISTOT

MMORPG 2018-01-08. [Verkkoaineisto]. [Viitattu 2018-03-23.] Saatavissa: https://www.mmorpg.com/

BERMUDEZ, Luis 2017. Unity3DAnimation. [Verkkoaineisto]. [Viitattu 2018-05-13.] Saatavissa: https://medium.com/unity3danimation/overview-of-inverse-kinematics-9769a43ba956

EPIC GAMES. Unreal Engine 4 Documentation. [Verkkoaineisto]. [Viitattu 2018-04-22.] Saatavissa: https://docs.unrealengine.com/en-us/

FIGGINS, Kiel. 3dFiggins.com, Painting Weights. [Verkkoaineisto]. [Viitattu 2018-04-25.] Saatavissa: http://www.3dfiggins.com/writeups/paintingWeights/#weightmultiple

FARIN, Gerald; HOSCHEK, Josef ja MYUNG-SOO, Kim 2002. Handbook of Computer Aided Geometric Design. Amsterdam.

EPIC GAMES. Unreal Engine. [Verkkoaineisto]. [Viitattu 2018-03-23.] Saatavissa: http://www.unrealengine.com

GOOGLE. Play Console Help. [Verkkoaineisto]. [Viitattu 2018-05-13.] Saatavissa: https://support.google.com/googleplay/android-developer/

KRILL, Paul 2017-08-16. Infoworld. [Verkkoaineisto]. [Viitattu 2018-04-22.] Saatavissa: https://www.infoworld.com/article/3217008/development-tools/the-most-popular-ides-visual-studio-and-eclipse.html

PETTIT, Nick 2015. Teamtreehouse, 3d modeling. Treehouse Island, Inc. [Verkkoaineisto]. [Viitattu 2018-05-08.] Saatavissa: http://blog.teamtreehouse.com/asset-workflow-game-art-3d-modeling

PETTIT, Nick 2015. Teamtreehouse, Texture mapping. .Treehouse Island, Inc. [Verkkoaineisto]. [Viitattu 2018-05-08.] Saatavissa: http://blog.teamtreehouse.com/asset-workflow-game-art-texture-mapping

THORN, Alan 2013. Game Development Principles.

WARD, Antony 2008. Game Character Development. Boston, MA, USA.

EPIC GAMES. Unreal Engine. [Verkkoaineisto]. [Viitattu 2018-05-09.] Saatavissa: https://answers.unrealengine.com/index.htm

BLENDER FOUNDATION. Blender Docs, Node Editor. [Verkkoaineisto]. [Viitattu 2018-05-09.] Saatavissa: https://docs.blender.org/manual/en/dev/editors/node_editor/introduction.html

BLENDER FOUNDATION. Blender Docs, Renderer, Materials. [Verkkoaineisto]. [Viitattu 2018-05-09.] Saatavissa: https://docs.blender.org/manual/en/dev/render/blender_render/materials/introduction.html#using-materials