OULUN AMMATTIKORKEAKOULU

Alisa Luomanmäki

RPG-pelin prototyypin kehitys Unityllä

3D-mallit ja animaatiot

RPG-pelin prototyypin kehitys Unityllä

3D-mallit ja animaatio

Alisa Luomanmäki
Opinnäytetyö
Kevät 2023
Tradenomi, tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tradenomi, tietojenkäsittely

Tekijä: Alisa Luomanmäki
Opinnäytetyön nimi: RPG-pelin prototyypin kehitys Unityllä
Työn ohjaaja: Teppo Räisänen
Työn valmistumislukukausi ja -vuosi: Kevät 2023
Sivumäärä: 48

Opinnäytetyössä kehitetään pelimoottori Unityllä 3D-pelin prototyyppi, jonka pääpaino on 3D-malleissa ja animaatioissa. Työssä keskitytään ihmismäisen 3D-mallin tuontiin Blenderistä Unityyn ja käsitellään, mitä asioita tulee ottaa huomioon tässä prosessissa. Lisäksi tutkitaan Animatorin käyttöä monipuolisten animaatioiden luomisessa.

Vaikka RGP-prototyyppi jää kesken, valmis hahmo pystyy liikkumaan syötteiden mukaan, taistelemaan vihollisia vastaan, valitsemaan erilaisia toimintoja hahmolle ja suorittamaan muita perustoimintoja.

Asiasanat: Unity, prototyyppi, pelikehitys, animaatio, 3D-malli

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Business Information Technology

Author: Alisa Luomanmäki
Title of thesis: Development of an RPG game prototype with Unity
Supervisor(s): Teppo Räisänen
Term and year when the thesis was submitted: Spring 2023
Number of pages: 48

The aim of the thesis project is to create a 3D game prototype by utilizing the Unity game engine, with a particular focus on 3D modeling and animations. The project involves importing a human-like 3D model from Blender to Unity, and the associated challenges and considerations are discussed. The research also explores the use of the Animator to produce a range of animations.

Despite the RPG prototype being incomplete, the main character can execute fundamental actions, such as movement based on user input, engaging in combat with enemies, selecting various actions for the character, and performing other basic functionalities.

Keywords: Unity, prototype, game development, animation, 3D model

SISÄLLYS

SANASTO

Armature – Tietokone animaatiossa käytettävät tukiranka, joka koostuu kinemaattisista ketjuista toisiinsa muodostaen liikkuvia osia (Wikipedia 2022).

GameObject – GameObject on perusluokka kaikille elementeille/kokonaisuuksille Unity Scenessä. Kaikki GameObjectit sisältävät peruskomponentin Transformin, johon tallentuvat sijainnin, käännösten ja koon arvot pelissä. Nämä arvot mahdollistavat sen, että GameObject tietää sijaintinsa ja mittasuhteensa suhteessa vanhempaansa eli siihen, onko se toisen GameObjectin lapsi tai world space -koordinaatistossa.

Indie-peli – Indie-pelit ovat yleensä luovia ja innovatiivisia, sillä pienet pelifirmat voivat keskittyä kehittämään juuri sellaisia pelejä, joita he haluavat tehdä ilman ulkopuolisten rahoittajien tai julkaisijoiden asettamia rajoituksia.

NPC – Non-playable character eli ei-pelaajahahmo. NPC on tietokonepelin sisäinen hahmo, jota pelaaja ei voi ohjata, vaan se toimii pelin maailmassa autonomisesti tai pelaajan toimien reaktiona.

RPG – Role-playing game eli roolipeli. Roolipelit ovat pelityyppi, jossa pelaajat voivat hallita yhtä tai useampaa pelihahmoa, jotka seikkailevat usein fantasiamaailmassa

1 JOHDANTO

Miettiessäni opinnäytteen aihetta, se antoi minulle mahdollisuuden alkaa tekemään erästä peli ideaani. Idea tähän peliin tuli minulta ja pelisuunnittelijalta, jonka kanssa olimme suunnitelleet kirjaa. Otin erään kohtauksen tarinasta, jonka halusin muokata peliksi.

Olin opetellut tekemään kaikkea, mitä peruspelin tekemiseen tarvitaan: 3D-mallinnusta, animaatioiden käsittelyä peleissä, äänet ja niiden asettelu, elokuvamaiset kamerakulmat ja paljon muuta. Nyt tämä oppimani taidot piti vain yhdistää yhteen tässä pelissä, mutta aika ei riittäisi valmiin pelin tekemiseen. Opiskellessani Unity Learn-oppimisympäristössä tutustuin prototyyppeihin, ja kehitin myös tuolloin ensimmäisen prototyyppini. Ymmärsin, että minunkin kannattaisi tehdä opinnäytetyöni prototyyppinä, koska aikaa ja resursseja ei ollut paljon ja enkä tiennyt kannattaako peliä tehdä loppuun asti.

Opinnäytetyön tavoitteena on kehittää ihmismäinen 3D-malli Blenderissä ja toteuttaa pelin prototyyppi Unityllä. Prototyypin avulla pyritään selvittämään, miten 3D-mallin saadaan liikkumaan animaatioiden avulla ja miten prototyypin eri elementit toimivat peliympäristössä. Tavoitteena on myös arvioida prototyypin teknistä toteutettavuutta ja kehitystyön mahdollisia haasteita.

Seuraavassa luvussa käsitellään opinnäytetyön tietoperustaa, kun taas luvuissa 3 ja 4 käydään läpi projektin toteutusta. Lopuksi luvussa 5 esitellään työn johtopäätökset.

2 IDEASTA PROTOTYYPIKSI

> *You got it. You have THE game idea of the year, maybe the one of the century. It is so awesome that you know the game industry will never be the same after that. You start writing everything down, sometimes shaped as a small note on a post-it, some other times as a 552 pages document with charts, drawings and strong references. After seconds or days of labour you are now sure that what you have in your hands is unique and perfect. Great. Good news is, you might actually have something good in hands. Bad news is, ideas are worth nothing, only execution makes the difference. (Victorino 2015.)*

Monet digitaalisia pelejä pelanneet, ovat varmasti ideoineet uusia ominaisuuksia pelaamaansa peliin, pohdiskelleet miten jonkin asian voisi tehdä paremmin tai miten kehittäisivät kokonaan paremman pelin. Eräät vievät ideansa eteenpäin kirjoittelemalla niitä paperille, piirtämällä tai muokkaavat jo olemassa olevaa peliä. Parhaimmista ideoista voidaan kehittää uusi peli, mutta yksi pelikehityksen tärkeimmistä vaiheista kannattaa suorittaa ensin: pelin prototyypin kehittäminen (Starloop Studios 2020).

2.1 Mikä on prototyyppi?

Pelikehityksen prototyyppi on koeversio peli-ideasta, jonka avulla voidaan testata, saavutetaanko haluttuja tuloksia ja onko ideassa potentiaalia peliksi. Prototyyppi on siis keskeinen työkalu pelin suunnittelussa, joka auttaa kehittäjiä hahmottamaan pelin ominaisuuksia ja mahdollisia ongelmia.

Kuva 1. Aiemmin tehty prototyyppi Unityn Scene-näkymässä.

Pelien prototyyppi määritelmä voi olla epäselvä. Minimaaliset vaatimukset pelin prototyypille ovat, että ydin pelimekaniikat toimivat johdonmukaisesti, prototyyppi on viihdyttävä ja opettaa pelaajalle tarvittavan sekä vakuuttaa sijoittajat sijoittamaan peliin. Näistä tärkeimpiä ominaisuuksia ovat pelimekaniikka ja sijoittajien vakuuttaminen. Mitkä ominaisuudet voivat olla liikaa prototyypissä? Niitä voi olla grafiikat, tunteet, äänet, tarinat, hahmot, maailman rakennus … Kuitenkin se olisi liian riisuttu versio pelistä. Pelisuunnittelijoiden tuleekin sisällyttää näitä syvyyksiä esituotannossa prototyypin tekosuunnitelmiin. (Polsinelli 2018.) Pelin prototyyppi onkin kevyt versio pelistä, joka halutaan saavuttaa onnistuneesti (Eden 2020). Kuvassa 1 on valmis prototyyppi.

Peliprototyypin kehitystä pitää käyttää uusien ideoiden luomiseen ja vaihtoehtoisten ratkaisujen löytämiseen. Tekemällä prototyyppiä itselleen ja tiimilleen, samalla kerätään arvokasta tietoa tiedon etsinnästä, asioiden määrittelystä, ideoinnista ja testaamisesta. (Starloop Studios 2020). Prototyypin saavutettujen ja testattavien ominaisuuksien jälkeen se voidaan tyrmätä tai kehittää valmiiksi tuotteeksi.

2.1.1 Indie-pelit

Indie-pelikehittäjät ovat usein hyvin omistautuneita ja haluavat luoda jotain ainutlaatuista ja persoonallista. Heillä on omat intohimo projektit, joissa ei välitetä dokumenteista tai tuhansista hienoista kuvista. Ei tarvita monen sadan henkilön kehitystiimiä. Indie-pelien kehittäjät tekevät peliä itselleen. He tietävät mitä ovat tekemässä ja miten peli tulee toimimaan. (Victorino 2015.) Indie-pelien tekemisessä on usein kyse enemmän intohimosta kuin voitosta, ja siksi monet indie-pelit ovatkin yllättäviä menestystarinoita kuten ruotsalaisen Markus Perssonin tekemä indie-peli Minecraft (Colligan & Sar 2019).

2.2 Prototyypin kehitys

Prototyypin kehityksessä tärkeitä tekijöitä ovat tiimin osaaminen, koko ja käytettävissä oleva aika. Prototyyppi voidaan toteuttaa joko paperiversiona tai digitaalisena versiona. Paperiversion tekeminen on nopeaa, eikä se vaadi koodaustaitoja ja on helposti muokattavissa, mutta sen pelimekaniikkaa ei voi testata käytännössä. (Mignano 2016.) Digitaalisia peliprototyyppejä voi tehdä monin eri tavoin, mutta käydään kaksi yleisintä läpi: nopea ja kehittyvä (Unity Learn 2022).

Nopeassa kehityksessä prototyypille määritellään vaatimukset ja niiden pohjalta luodaan prototyyppi. Peliä testataan ja muokataan muuttuvien vaatimusten mukaan. Tätä kehitystä tekevät harrastajat sekä pienien indie-pelien tekijät. Toimivaksi testattu idea voidaan siirtää tuotantoon tai prototyyppi muuttuu kehittyväksi prototyypiksi viilailua varten ennen suurempaa tuotantoa. (Unity Learn 2022.)

Kehittyvässä prototyyppimallissa pyritään alusta asti luomaan valmis tuotos, jonka pelaajat saavat pelattavaksi. Projektissa ideana on pitkä kehityskaari, jossa prototyyppiä kehitetään ja hienosäädetään alkuperäisen idean pohjalta. (Unity Learn 2022.)

2.3 Prototyypin arviointi

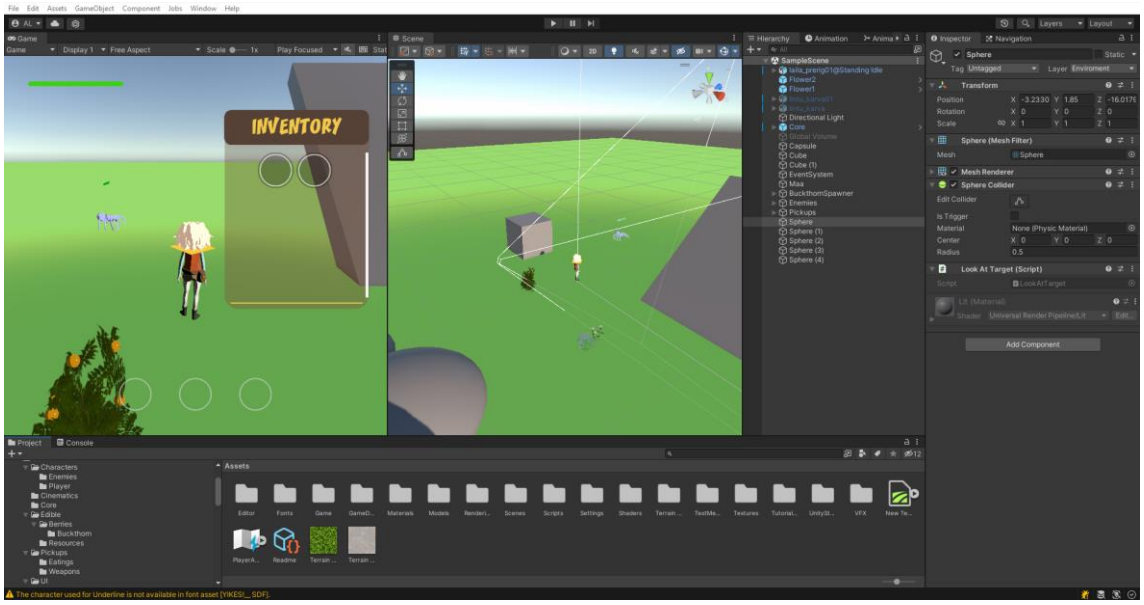
Prototyypin arvioinnin onnistumiseen vaikuttavat se, mitä halutaan saavuttaa, kuten esimerkiksi tietyn toiminnon, pelimaailma ja tarina tai pelimekaniikan toimivuus. Prototyypin testaamisessa on käytössä monia erilaisia menetelmiä, joita voidaan käyttää sen määrittelyssä osana esituotannon vaihetta.

Tärkeä tekijä, joka vaikuttaa arvioinnin onnistumiseen, on pelin kohdeyleisö. Kohdeyleisö on määriteltävä jo prototyypin varhaisessa vaiheessa, sillä valmistuneen prototyypin on tarkoitus olla suunnattu testattavaksi juuri kyseiselle yleisölle. Esimerkiksi, jos suunnitellaan realistista lentosimulaatiopeliä, on tärkeää selvittää, ketkä ovat sen potentiaalisia pelaajia, mukaan lukien sukupuoli, ikäjakauma ja muut kiinnostuksen kohteet. Vasta kun kohdeyleisö on tiedossa, prototyyppi voidaan kehittää heille ja testata heillä. (Eden 2020.)

Esituotantovaiheessa tutkitaan myös, onko markkinoilla muita samankaltaisia pelejä ja miten ne menestyvät, jotta saadaan tietoa mahdollisista kilpailijoista ja siitä, onko markkinoilla tarpeeksi kysyntää uudelle pelille. Tämä auttaa arvioimaan, onko prototyypillä potentiaalia menestyä ja houkutella sijoittajia. Tarkastelussa otetaan huomioon myös se, millaista lisäarvoa prototyyppi voisi tuoda pelimaailmaan, jotta se erottuisi kilpailijoista ja houkuttelisi pelaajia. (Eden 2020.)

2.4 Kehitysalusta Unity

Unity on Unity Tecnologiesin luoma alustariippumaton reaaliaikainen kehitysalusta (Wikipedia 2023). Ensimmäinen Unityn kehitysalusta valmistui vuonna 2005 Tanskassa kolmen ohjelmoijan tuotoksena. Päämääränä oli luoda edullinen pelimoottori ammattimaisilla työkaluilla amatööripelikehittäjille. Inspiraationa oli helppo työnkulku, yksinkertainen elementtiliukuhihna eli asset pipeline ja vedä-pudota-rajapinta käyttäjille. (Haas 2014, 1.) Unityn pipelinella tarkoitetaan sarjaa toimintoja tai prosesseja, jotka suoritetaan tietyssä järjestyksessä tietyn ohjelmointiprojektin aikana.



Kuva 2. Unityn kehitysalusta.

Tämä päivänä Unitylla kehitetään 2D- ja 3D-pelejä yli kahdellekymmenelle eri alustalle. Valmiit pelit ovat rojaltivapaita ja niillä on yli puolitoista miljoonaa kuukausittaista käyttäjää. Alun perin pelien kehitysalustana toiminut Unity on laajentunut monelle eri toimialalle peliteollisuuden lisäksi; arkkitehti, auto- ja elokuvateollisuus. (Unity Technologies 2023.) Kuvassa 2 näkyy miltä Unityn käyttöliittymä näyttää.

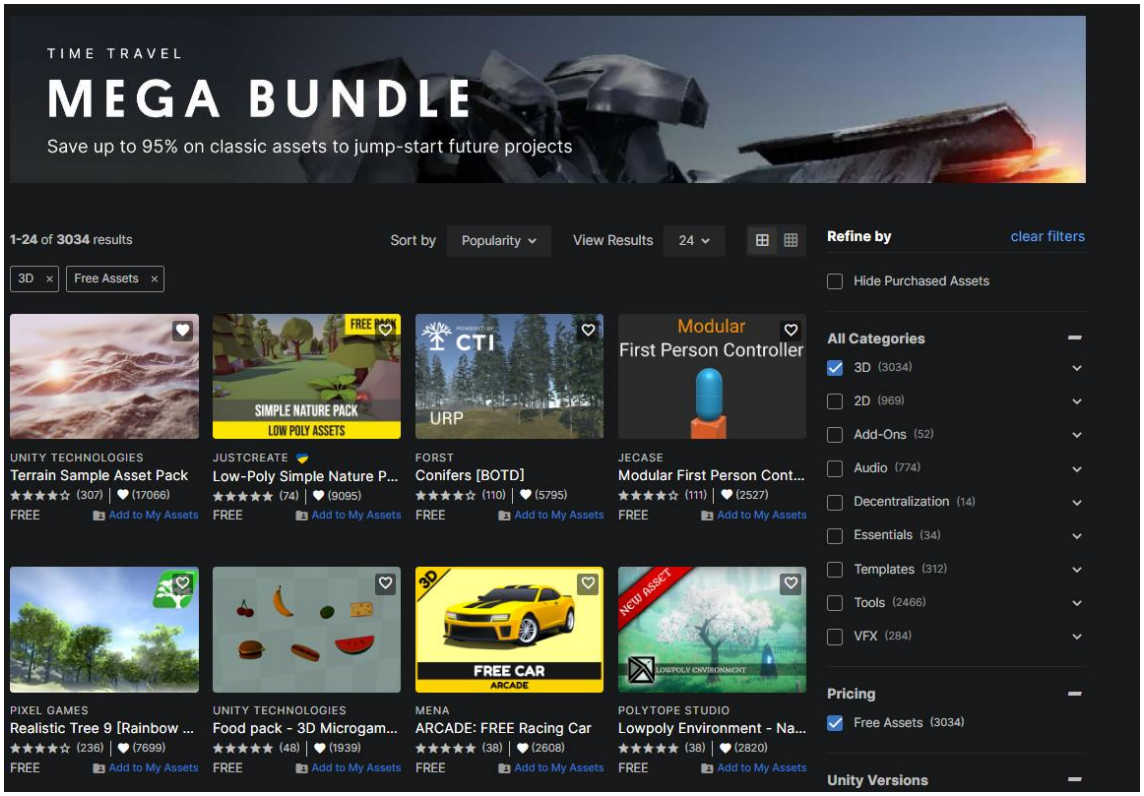
2.4.1 Pelikehitys aloittelijalle Unityllä

Aloittavalle pelikehittäjälle Unity on todella käyttäjäystävällinen. Unityn dokumentaatiossa selitetään miten pelialusta ja sen eri osat toimivat. Paras tapa onkin tutustua Unityssa oleviin mahdollisuuksiin ja opetella kehitysavustaan käyttöä on Unity Learnin kautta. Opiskelupolut tutustuttavat

aloittelijan Unityn eri mahdollisuuksiin riippuen mitä ominaisuuksia halutaan saavuttaa Unitylta. Unityn ollessa helppo käyttää ja alustavasti ilmainen pienille tiimeille, Unityllä onkin kuukaudessa n. 1.5 miljoonaa käyttäjää (Unity Technologies 2023). Tietoa ei ole onko mukana ilmaisversion käyttäjät.

Internetistä löytyy paljon vastauksia, jos jokin alkaa askarruttamaan Unityn käytössä, koska monet etsivät apua samoihin toistuviin ongelmiin. Monet indie-pelikehittäjät neuvovat YouTube-videoilla miten erilaisia pelejä voi tehdä. Jos haluaa opiskella enemmän Unitya niin kursseja ja oppaita löytyy ilmaisista maksullisiin.

Lisäksi aloittelevat pelikehittäjät voivat hyötyä myös Unityn laajasta yhteisöstä, jossa on paljon kokemusta ja tietoa jaetaan avoimesti. Unityn käyttäjät voivat jakaa omia projektejaan ja tarjota apuaan foorumeilla, Discord-ryhmissä ja muissa yhteisöissä. Yhteisössä voi myös verkostoitua muiden pelikehittäjien kanssa saaden arvokkaita kontakteja, yhteisprojekteja ja mahdollisesti tulevaisuuden työpaikan.



Kuva 3. Kuvakaappaus Unity Storesta.

Unity Store on Unity Technologiesin ylläpitämä sivusto, josta voi hankkia ilmaisia ja maksullisia assetteja. Assetit ovat erilaisia peliin tuotavia elementtejä, kuten 3D-mallit, audiot, kuvat, animaatiot, erilaiset kirjastot, lisäosat, valmiita koodeja ja muut vastaavat. Kuvassa 3 on kuva Unity Storesta ja kuvassa oikealla näkyy assettien hakuluokat.

3 PROJEKTIN PÄÄMÄÄRÄT

3.1 Projektin hallinta

Kehitystiimimme koostuu vain kahdesta henkilöstä: pelisuunnittelijasta ja pelikehittäjästä. Pelikehittäjä toteuttaa pelisuunnittelijan suunnitelmia ja varattuna aikana kaksisataa tuntia prototyypin kehittämiseen. Projektin suunnittelua ja kehitystä tehdään mahdollisuuksien mukaan niin pitkälle, kun keretään, mutta tiimin pienen koon takia osa jää pakosti kesken. Tiimin kommunikointiin käytämme WhatsAppia, Google Drivea ja Hack&Plania, ja pidämme viikoittaisia palavereita sekä tarvittaessa muita kokouksia.

Kahteensataan tuntiin kuuluvat 3D-mallien teko ja etsiminen, suunnittelu, palaverit, kehitys Unitylla ja testaaminen. Projekti pyritään toteuttamaan nollabudjetilla. Tarvittavia resursseja saadaan vanhoista projekteista, itse tekemällä ja netistä.

Pelikehityksessä käytämme Unityn kehitysalustan kehitysversiota 2021.3.8f1 ja ohjelmointikieleksi valikoitui helppokäyttöinen C#. Ohjelmointikielenä C# mahdollistaa monien muiden pelin komponenttien, kuten fysiikan, animaatioiden ja käyttöliittymän toteuttamisen Unityssä.

3.2 Pelin kuvaus

Kolmiulotteinen RPG-pelimaailma, jossa pelaaja ohjaa nuorta Laila-hahmoa kolmannessa persoonassa. Pelattava hahmo on noin 11-vuotias tyttö, joka pystyy suorittamaan erilaisia toimintoja, kuten kävelemään, juoksemaan, hyppimään, tarttumaan esineisiin, puolustautumaan, hyökkäämään, tippumaan, kiipeilemään ja kommunikoimaan NPC-hahmojen kanssa. Pelimaailma on rajoitettu saareen, jolle pelaaja ajautuu haaksirikon seurauksena. Lailan avustuksella pelaaja auttaa toista loukkaantunutta tyttöä ja kokee saaren mysteerejä. Pelin teemana on 1990-luvun Perämeri, ja saaren muoto on mallinnettu Raahen edustan Iso-Kraaselin mukaan.

Pelin tunnelma on viileä syysilta, hetki juuri ennen auringonlaskua. Pelihahmon lämpötilalla on myös vaikutusta peliin, joten jos lämpötila laskee liian alas, hahmo liikkuu hitaammin tai alkaa menettämään lämpöpisteitä. Lämpötilan pisteitä voisi nostaa vaatteilla ja nuotiolla, mutta eri alueilla saarta on eri lämpötiloja, joissa kylmyys alkaa laskemaan lämpöpisteitä nopeammin.

Pelaajalla on myös apunaan tekoälyllä varustettu lintu, jonka avulla hän voi löytää NPC-hahmoja tai muita kohteita. Pelisuunnittelijan kanssa on harkittu myös pelin aiheen soveltuvuutta kirjaksi, mutta juonen yksityiskohdat jätetään tässä vaiheessa avoimeksi.

3.3 Alkusuunnitelma

Peliä pelattaisiin tietokoneella. Prototyyppiä siis kehitetään toimimaan näppäimistöllä ja hiirellä. Eräs tavoite on saada pelattava hahmo liikkumaan pelaajan syötteiden mukaan.

Tavoitteena on myös saada tavaraluettelo ja tavarat toimimaan halutusti ja toimintakuvakkeet toimimaan Minecraftin lailla. Lisäksi saada lokki seuraamaan Lailaa ja luoda onnistunutta kommunikointia heidän välilleen. Lopuksi pitäisi tehdä raakaversio saaresta käyttäen Unityn Terrain-komponenttia. Prototyyppiin ei tarvitse tehdä tutoriaalia, koska emme ole suunnitelleet testausta, koska aika ei riitä.

Projektiin otin aiemmista projekteistani RPG-peleille tarvittavat ominaisuudet käyttöön, kuten tavaraluettelon, tavaroiden hallinnan pelissä, vihollisen tekoälyn ja muita pieniä koodin pätkiä. Aiemmat C#-koodit on tehty Diablo-tyyppiselle pelille, jossa pelitilanne on kuvattu yläviistosta. Hahmoa liikutellaan pelimaailmassa painamalla peliympäristöä hiirellä. Toimintoja valitaan painamalla toimintakuvaketta ja hahmo taistelee automaattisesti, kun vihollista on klikattu kerran hiirellä. Vanha koodi antaa pohjaa, mutta muokattavaa koodia olisi paljon.

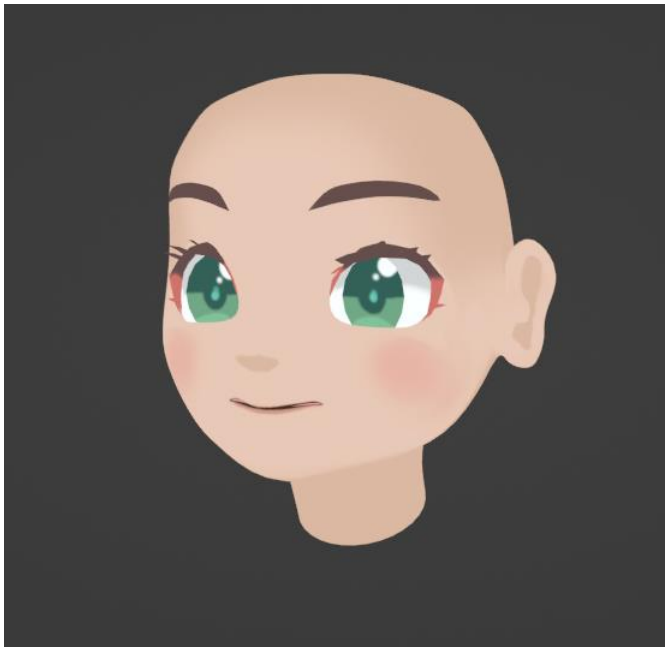
3.4 Näkökulma

Pyrin kehittämään peliä monipuolisesti eli tämä sisältää koodaamista, testaamista, UI-suunnittelua ja sen toteutusta, hahmon reagointia maailmaansa ja paljon muuta, mutta rajaan aiheesta kirjoittamisen Unityn animaatioon. Olen aina ollut kiinnostunut animaatiosta ja pidän pelien animointien liikkeestä, väreistä ja visuaalisuudesta. Unityn animaation ohjelmointi edellyttää visuaalista silmää

ja monia testauksia erilaisten animointien aktivoinnin ja lopettamisen oikeassa ajassa ja paikassa. Tämä on tärkeää pelin visuaalisen vaikutelman luomiseksi ja pelaajakokemuksen parantamiseksi.

4 PROJEKTI

4.1 Lailan 3D-malli



Kuva 4. Blenderissä Lailan ensimmäiset kasvot.

Suoritin aiemmin hahmonmallinnuskurssin, jossa valmistin 3D-mallin Lailasta. Kurssilla käytimme Blender-nimistä ilmaista ja avoimen lähdekoodin ohjelmistoa ihmishahmojen mallintamiseen. Prototyyppiä varten tein erilaisia kokeiluja riggaus-vaiheessa. Riggaus termiä käytetään, kun 3D-mallille lisätään kontrollit tyypillisesti animaatiota varten (Blender Manual 2023). Kokeilin kurssin aikana kahta erilaista Blenderin animointitapaa tähtäimenä prototyyppi; Armature ja Shape Keys. Shape Keys jätin pois, koska prototyypissä sen säätäminen kohdalleen olisi ollut ajanhukkaa eikä lisäisi pelattavuutta, ehkä tunnelmaa, koska sillä olisi saanut pelattavan hahmon naamalle erilaisia ilmeitä. Kuvassa 4 näkyy eräs versio Lailasta, joka pidin liian anime-tyylisenä.

4.1.1 Armature

3D-ihmishahmossa asetetaan luut, jotka ovat näkymättömiä tukia hierarkkisessa järjestyksessä, jossa ylimpänä hierarkiassa on yleensä lantio, joka jakaantuu ja laajenee luu luulta raajoihin ja

päähän. Luiden välistä yhteyttä sanotaan niveliksi. Nämä luut muodostavat hahmon ”tukiluurangon” eli armaturen. Jokainen luu on kiinnitetty johonkin kohtaan luuta ympäröivän verteksiverkon verteksiin. Luuta liikutettaessa myös verteksiverkko liikkuu mukana, riippuen siitä millainen paino luulla on verteksiverkon verteksiin. Luun paino verteksiin on annettu arvojanalla nollasta yhteen. Nolla arvolla verteksi ei liiku yhtään. Arvolla yksi verteksi seuraa luun liikettä täydellisesti. Unityyn tuotavassa 3D-mallissa, verteksillä voi olla myös yhteys enintään neljään eri luihin (Unity Technologies 2022). Verteksiverkon ja luiden tasapainon säätämistä painoilla sanotaan skinnaukseksi, ja se tehdään yleensä jollain saatavilla olevista 3D-mallinnusohjelmista, koska nämä luut ovat 3D-animaation riggausvaiheen tuotos. Hyvällä valmiilla 3D-hahmolla onkin verteksiverkko, armature, animaatiot ja materiaalit, jotka yleensä tuodaan valmiina pakettina Unityyn. Valmiita hahmon elementtejä voi myös tuoda peliin, mutta se pitää ottaa huomioon kehitystyössä, jottei homma mene sekavaksi. Riskinä on, ettei työ onnistu välttämättä halutulla tavalla sekä ajanhukkaamiseen uudelleen sovituksessa Unityssa, joten minulla kävi useasti.

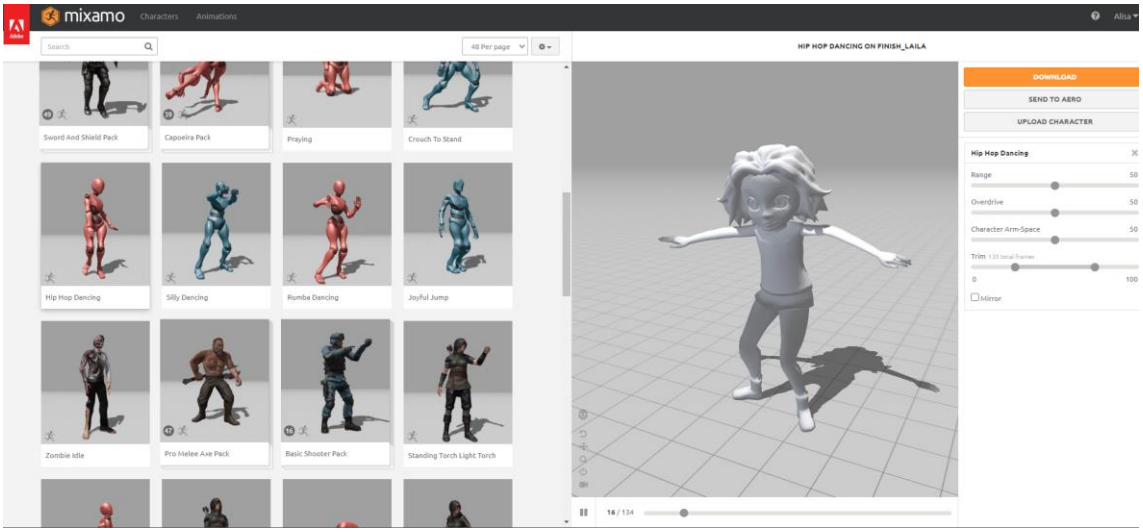
Kuva 5. Lailan 3D-malli Blenderissä, johon on asetettu Blenderin oma armature.

Erilaisten internetkeskusteluiden selaamisen ja kokeilujen jälkeen Unityn ja Blenderin välillä mietin ajan käyttöä ja miten saisin yksinkertaisesti hyvän riggauksen ja animaatiot hahmoille ajan säästä-

miseksi prototyyppiä varten. Kuvassa 5 näkyy Blenderissä tehty riggaus. Päätin siis tehdä automaatioriggauksen Mixamolla, koska sen tarjoamassa selainpalvelussa riggauksen lisäksi on myös animaatioita, joita sekoittamalla Unityn Animatorilla saa erilaisia toimintoja tehtyä hahmolle eikä animaatiota tarvitsisi etsiä muualta.

4.1.2 Mixamo

Mixamo on ilmainen selainpalvelu kaikille, jotka ovat rekisteröityneet Adoben käyttäjiksi. Palvelun (Mixamo 2023) tarjoaa 3D-hahmoja ja liikekaapattuja animaatiota, jotka ovat kaikki rojaltivapaita erilaisiin projekteihin. Animaatioita on tuhansia erilaisia, jotka on jaettu kahdeksaan osioon; taistelu, seikkailu, urheilu, tanssi, fantasia, supersankari ja skinnaustesti.



Kuva 6. Lailan 3D-malli Mixamossa.

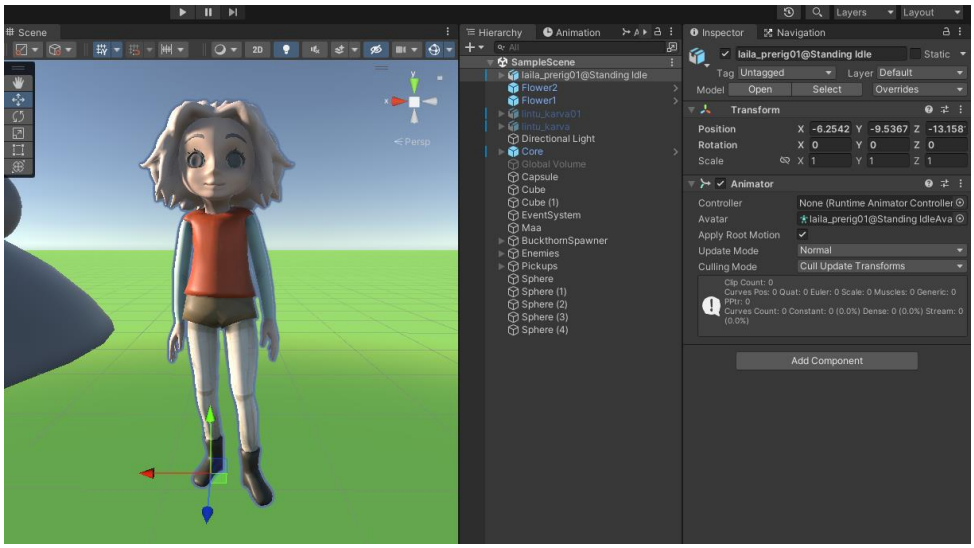
Palveluun ladattiin t-asennossa oleva 3D-hahmo. Prosessin aikana piti määritellä 3D-hahmo oikein päin ja se asettaa tietyt väripallot oikeille paikoilleen osoittamaan hahmon leuan, ranteet, kyynärpäät, polvet ja haaran. Hahmon Mixamo-skinnauksen voisi tehdä myös Blenderissä lisäosan avulla. Mixamon animaatioita voi myös ladata Blenderiin, missä niitä voi muokata tai yhdistää kokonaisuuksiksi. Jos verteksiverkon verteksit eivät liiku oikein animaation kanssa, esimerkiksi iso maha liikkuu jalkojen mukana kuin kevyt kangas, voidaan myös muokata Blenderissä hahmossa olevan verteksiverkon painoa, jotta animaatio liikkuu sulavasti ja luontevasti kuten kuvassa 6.

4.1.3 Animaatiot ja mallin sovitus

Lailan 3D-malli on perushahmo, joten sille löytyi paljon sopivia animaatioita eikä se tarvitse Blenderissä uutta skinnausta. Se on kuitenkin piirroshahmotyyppinen lapsi, jolla on tavallista isompi pää, lyhyt vartalo ja pitkät raajat. Hahmon mallintamisessa piti testata koko ajan, että hahmo näyttää luonnolliselta ja mittasuhteet eivät ole silmää häiritseviä, koska Mixamon perusanimaatiot ovat tehty aikuisille hahmoille. Hahmon ollessa vielä todella kulmikas Blenderissä, latasin sen vähän välillä Mixamoon ja laitoin liikkumaan hankalia animaatioita. Muokkasin hahmoa, kunnes sen mittasuhteet näyttivät hyvältä Mixamon animaatiossa. Mixamon animaatiota pystyy kustomoimaan eri tavoin riippuen animaatiosta. Kuten usein kävi, että hahmon käsi meni läpi jostain kohtaa kehoa, niin animaatiosta voi löytyä erilaisia muokkaimia, että virheestä pääsee eroon kuten esimerkiksi käsien liikkeen laajuuden säätö. Jos hyvin häiritsevästä virheestä ei pääse eroon, kannattaa valita jokin toinen animaatio.

Ensimmäiseksi latasin palvelusta skinnatun hahmoni ja idle-animaationi eli hahmo vain oleskelee, josta on hyvä alkaa rakentamaan hahmon liikkumista ja muiden animaatioiden sekoittamista. Aina, kun tarvitsisin animaation uudelle toiminnolle, etsin halutunlaisen, muokkasin sen ja latasin tarvitsemani animaation palvelusta. Hahmon materiaalit toin erikseen Blenderistä Unityyn.

4.2 3D-Mallien tuonti Unityyn



Kuva 7. Lailan 3D-malli Unityn Hierachyssa.

Unityyn tuotavat 3D-mallit vedin kehitysalustan Hierarchyyn, jolloin malli näkyy pelinäkymässä kuten kuvassa 7. Hierarchyssä malli muuttuu GameObjectiksi, GameObject nimesin lopulliseen muotoonsa, jotta se erottuu hyvin muista. Jotta 3D-malli toimii ja näkyy oikein pelinäkymässä, sille pitää tehdä tarvittavia muutoksia. Lailan 3D-mallin sijoitin Player-GameObjectin lapseksi, jolloin sen käsittely helpottui.

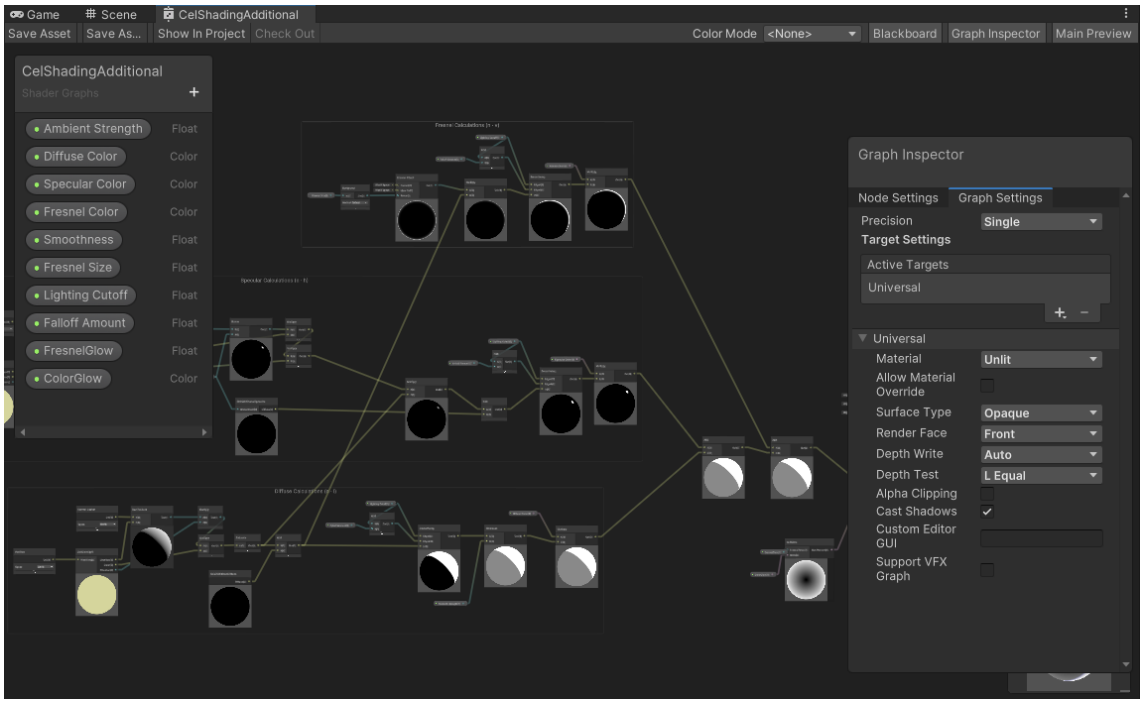
4.2.1 Sijoitus GameObjectissa

3D-malli voi olla miten sattuu vanhemmassa GameObjectin. Se piti kääntää oikein päin Transform-komponentin arvoilla. Mallin etupuolen tuli suunnata/kääntää sinisen nuolen ja ylöspäin vihreän nuolen mukaan. Tärkeää on verrata myös, miten iso 3D-malli on peliympäristössä. Kokoa muunsin Transformin Scalessa tai Project-ikkunan kansion kautta mallia sen Inspector kohdasta Scale Factor. Piti muistaa, että Transform muuttaa vain tietyn GameObjectin tai sen Prefabsien kokoa. 3D-mallin asetuksien oma Scale Factor muuttaa kaikkia pelissä olevia mallin kopiota. Sijoitin 3D-mallin olemaan jalkojensa tasolla vanhempansa GameObjectin origoon.

4.2.2 Shader

Sarjakuvamaiset 3D-mallit ovat yleisesti käytössä videopeleissä, animaatioissa, mainoksissa ja markkinointimateriaaleissa. Niitä käytetään usein pelien tai animaatioiden hahmojen, esineiden ja ympäristöjen luomisessa. Blenderissä Laila näytti upealta, mutta Unityssa hahmoon vaikuttavat Shaderit.

Shader tarkoittaa Unityssa osaa, joka toimii grafiikkapipelinessa. Shaderit suorittavat laskelmia, miten värit näkyvät ruudulla. Halusin sarjakuvamaisen Shaderin, ja pidin The Legend of Zelda: Breath of the Wild-pelin visuaalisuudesta, joten halusin sellaisen pinnan Lailan 3D-mallille.



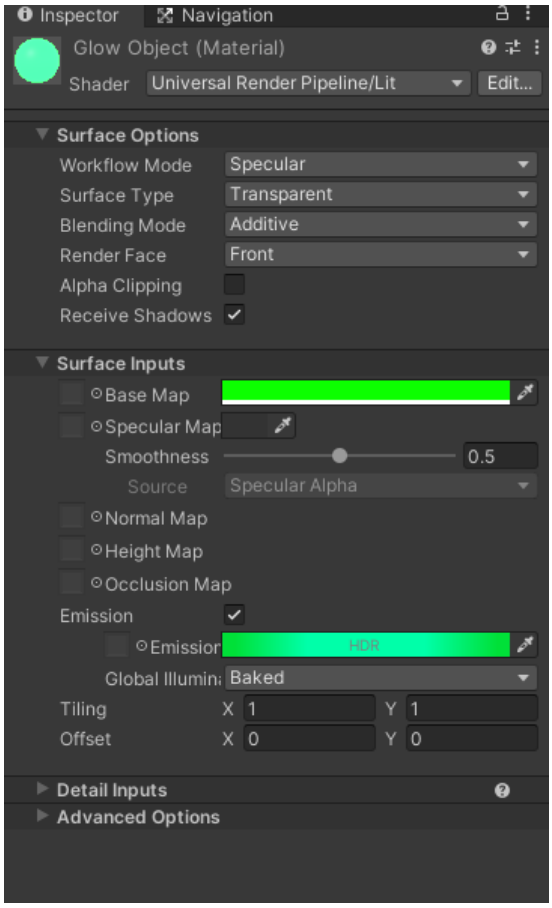
Kuva 8. Zelda-Shaderin asetukset. Kuvassa on Grap Shader Editorilla luotu Shader, joka on solmupohjainen.

YouTubesta löysin ohjeet Shaderiä varten, jota kutsun Zelda-Shaderiksi, joka luo Cel Shading efektin. Kuvassa 8 näkee tämän Shaderin monimutkaisuuden. Tämä Shader laskee, miten valo osuu 3D-mallin tahkojen pintaan, tahko on neljän verteksin välinen alue, ja siten se luo valon ja varjo alueet luoden samanlaisen visuaalisen ulkonäön kuin Zeldassa. Kuvassa 9 näkee Zelda-Shaderin vaikutuksen. Shadereilla voitaisiin saada peliin erilaisia vaikutteita pintoihin kuten muovi, metallinen, lasinen, mutainen, virtaava vesi tai hitaasti liikkuva hohtava tulinen laava.

Kuva 9. Vasemmanpuoleisella 3D-mallilla on Universal Render Pipelinen oletus Shader eli Lit Shader. Oikeanpuoleisella mallilla on Zelda-Shader.

4.2.3 Materiaalit

Käytössäni oleva uudempi Unityn versio, jota käytin kehitystyössä, lisäsi haasteita. Ottaessani Unity Storesta vanhempia malleja käyttöön projektiin, mallit hohtivat vaaleanpunaisina pelissä, koska niissä oli vanhat materiaalit, joiden tietojen käsittely ei toiminut uudessa versiossa. Tällöin minun täytyi asettaa uudet materiaalit, Shaderit ja tarvittaessa myös tekstuurit (Unity Technologies 2023b).



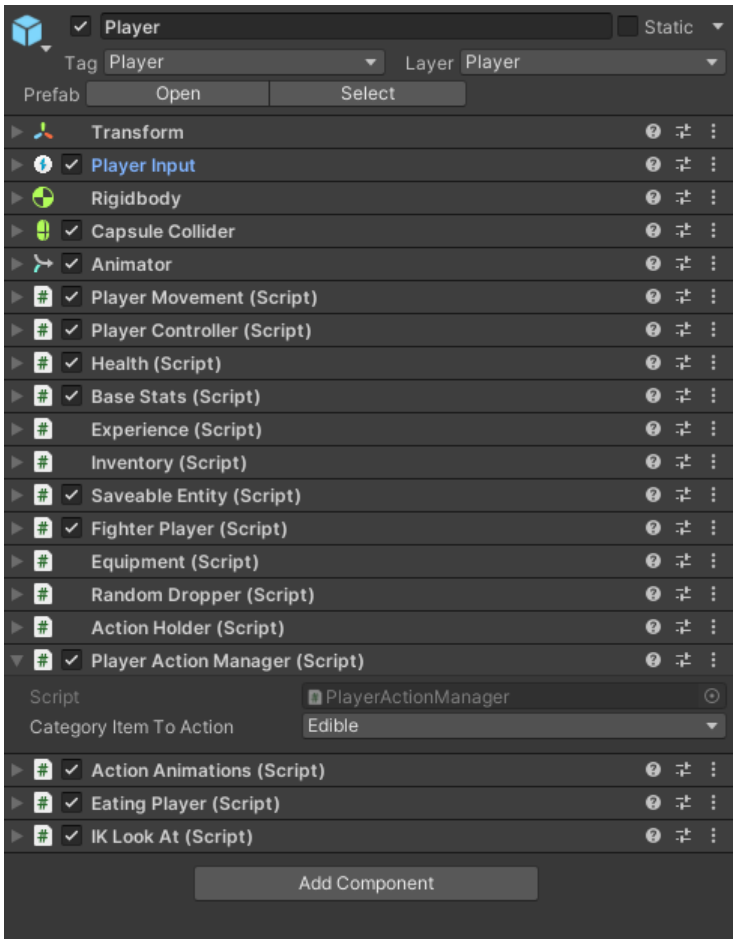
Kuva 10. Unityn Materiaalin arvojen asetukset.

Unityn materiaalit ovat graafisia komponentteja, joita käytetään 3D-mallien ulkonäön määrittämiseen. Riippuen siitä, minkälaista Shaderia materiaalilla käytetään, voidaan säätämällä arvoja tehdä 3D-mallista eri värisen, läpinäkyvän, kiiltävän, heijastavan, liikkuvan ja erilaiset efektit. Myös erilaisilla tekstuurimalleilla voi luoda kuvioita ja kuvia, jolloin saadaan 3D-mallille realistinen pinta. Kuvassa 10 on Inspectorissa tehty kuultava materiaali.

4.2.4 Prefab

Saadessani GameObjectin valmiiksi eli sen 3D-malli on sijoitettu oikein vanhempaansa nähden, Shaderit ja materiaalit on määritelty ja tarvittavat komponentit lisätty, tallensin sen valittuun kansioon Prefabsina. Prefab on valmis elementti, jota voi kopioida peliin. Jos Prefab-elementtiä muokkaa Prefab—tilassa niin se vaikuttaa kaikkiin samoihin Prefabsiin. Pelin Sceneen tuoduissa Prefabsin ei tarvitse olla identtisiä. Niihin voi tehdä pelitilassa muokkauksia, jolloin siitä saa variaatioita peliin, kunhan ylikirjoittaa alkuperäisen Prefabsin tai tallentaa uutena.

4.3 C# -ohjelmointikieli



Kuva 11. Player-GameObjectin lopulliset komponentit. Kaikki risuaidalla merkityt komponentit ovat itse koodattuja tai muokattuja.

Player-GameObjectiin sijoitin kaikki pelaajaa tai pelihahmoa koskevat komponentit, jotka näkyvät kuvassa 11. Ohjelmointikielenä on käytetty C#. Se on oliopohjainen koodikieli eli se koostuu muuttujista, funktioista ja luokista. Nämä komponentit vaihtavat tietoa keskenään pelitilassa. Miten pelihahmo liikkuu, hahmon terveyspisteet, taistelu, kerätyt tavarat tallentuvat ja ym.

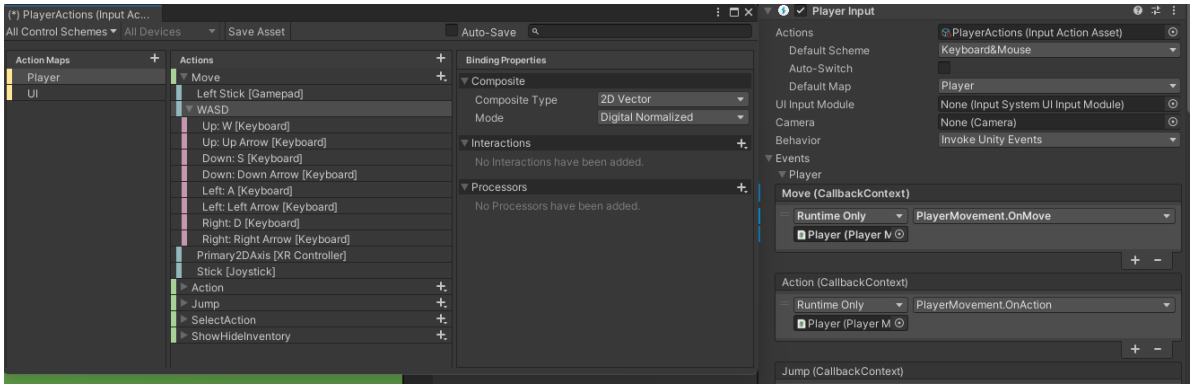
Itse koodatut komponentit Unityssä perivät MonoBehaviourin, joka on Unityn oma C#-perusluokka. Luokassa on määritelty funktiota valmiiseen käyttöön kuten Awake, Start, Update, OnTriggerEnter ja paljon muita. Valmiit funktiot lisätään komponentteihin, kun niitä tarvitaan. Tämän ansiosta pelikehitys on helppoa ja nopeaa. (Çamönü 2020.)

Aukaisen hieman Update-funktiota. Komponentissa se lukee sisällään olevaa C#-koodia tai kutsuu komponentin muita funktiota. Pelimoottori kutsuu pelissä olevia Update funktiota kuusikymmentä

kertaa sekunnissa eli sen tehtävä on päivittää esimerkiksi GameObjectin paikkaa ruudulla näkyvää aikaa (Unity Technologies 2021).

4.4 Liikkuminen pelihahmolla

4.4.1 PlayerInput ja PlayerMovement



Kuva 12. PlayerActions ja Player-GameObjectin PlayerInput-komponentti.

PlayerInput-komponentti lukee määritellyistä ulkoisista syötteistä tulevia arvoja. Aiemmissa Unityn versioissa pelaajan syöte tarkastettiin Update-funktiossa, joka framessä, mutta uusi InputAction automaattisesti käsittelee aktivoinnin ja käytöstä poiston automaattisesti, lisäksi se myös käsittelee tallentaen kutsut toimintoihin (Unity Technologies 2020). InputActionsissa määrittelin kaikki ne komennot, joita tarvitsin hahmon liikuttamiseen ja valintoihin UI:ssa. Toiminnot, jotka olin määritellyt käyttöön ovat Move, Action, Jump, SelectAction, ShowHideInventory ja Sprint. Valitsin toiminnoksi Invoke Unity Events, jonka kautta syötteet tulevat koodiin. PlayerMovement-komponentin tehtävänä on ottaa PlayerInputin Eventit vastaan. Kuvassa 12 on PlayerAction Input ja Player Input.

```
public void OnMove (InputAction.CallbackContext context)
{
    moveDirection = context.ReadValue<Vector2> ();
}
```

Kuva 13. OnMove-funktio lukee PlayerInputista tulevat WASD-näppäimistösyötteen ja palauttaa sen, jonka voi tallentaa Vector2-arvona. Vector2 on luokka, jolla on kaksi ominaisuutta; x ja y.

Pelaajan painaessa näppäimistön W-, A-, S- tai D-painiketta arvot muuttuvat ja syöte tulee PlayerInputista siirtyen Eventtiin ja sen jälkeen PlayerMovement-komponentin OnMove-funktioon,

jonka on kuvassa 13. OnMove-funktio tallentaa arvon PlayerMovement-komponentin moveDirection-muuttujaan.

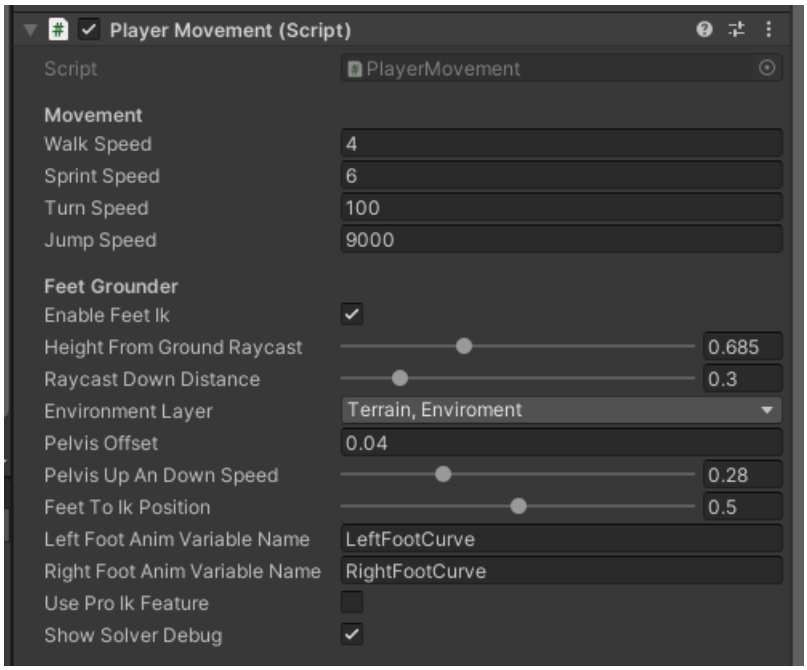
```
void Move (Vector2 direction) {
    float turnAmount = direction.x;
    float fDirection = direction.y;
    if (direction.sqrMagnitude > 1f) direction.Normalize ();

    if(isSprinting)
    {
        desiredSpeed = direction.magnitude * sprintSpeed * Mathf.Sign (fDirection);
    }
    else
    {
        desiredSpeed = direction.magnitude * walkSpeed * Mathf.Sign (fDirection);
    }
    float acceleration = IsMoveInput ? groundAccel : groundDecel;

    forwadSpeed = Mathf.MoveTowards (forwadSpeed, desiredSpeed, acceleration * Time.deltaTime);

    anim.SetFloat ("ForwardSpeed", forwadSpeed);

    transform.Rotate (0, turnAmount * turnSpeed * Time.deltaTime, 0);
}
```

Kuva 14. Move-funktio PlayerMovement-komponentissa.

Kutsumalla Move-funktiota peliä päivittävässä Update-funktiossa saadaan koko ajan pelaajan liikkumissyötettä seuraava funktio. Puretaan direction -parametri, joka on tallennettu moveDirection, x ja y muuttujiin. Niiden arvot voivat olla 0, -1 ja 1. Nolla tarkoittaa, että syötettä ei ole eikä hahmo liiku. Y-arvo positiivinen yksi tulee W-näppäimestä ja sitä painamalla hahmon tulee liikkua eteenpäin. S-näppäimen syötteen Y-arvosta tulee negatiivinen yksi ja silloin hahmon tulee liikkua taaksepäin. A- ja D-näppäimet antavat x-arvon positiivisena tai negatiivisena, jolloin määritellään kääntymissuunta vasemmalle tai oikealle.

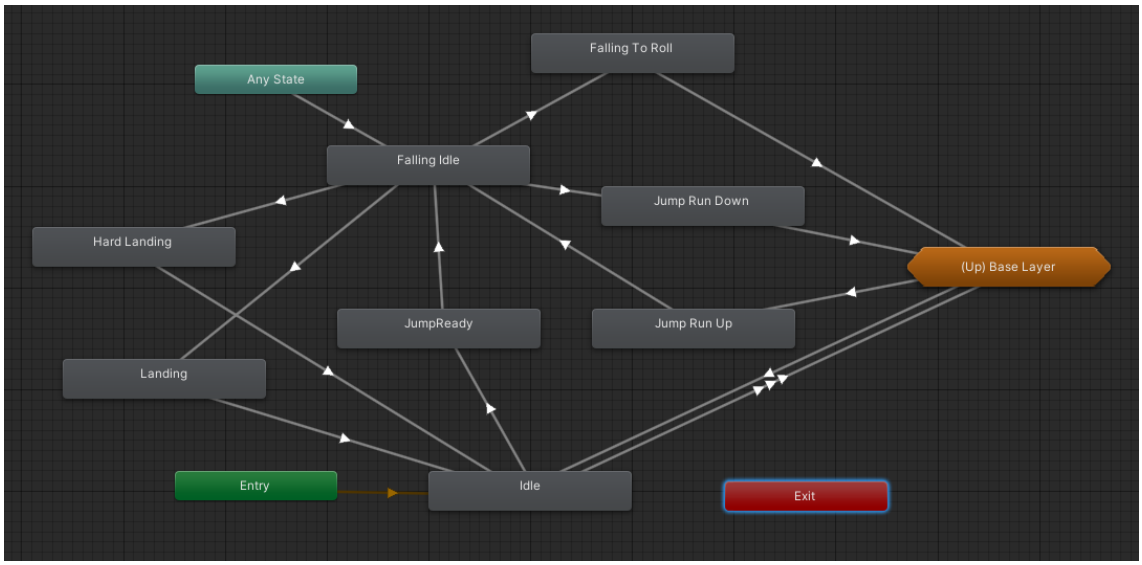
Pelitilassa PlayerMovement-komponentti etsii Awake-funktiossa samassa GameObjectissa olevan Animatorin-komponentin ja tallentaa sen muuttujaan myöhempää käyttöä varten. Pelaajan antaessa WASD-näppäimillä syötteen, OnMove-funktio tallentaa syötteen arvot ja Update-funktion kutsuu, koko ajan Move-funktiota. Se purkaa tallennetun syötteen arvot, jossa parin tarkistimen jälkeen kuten, onko hahmo kosketuksessa maahan ja antaako pelaaja juoksu syötettä, valitaan oikeat laskelmat, että tiedetään hahmon liikkumisnopeus pelissä. Nopeus asetetaan Animator-komponentin ForwardSpeed-parametrille. Kuvassa 14 on koodin rakenne. Animator-komponentti valitsee oi-

kean animaation hahmon nopeuden ja muiden Animatorin-parametrien mukaan. Hahmon kääntymisen syöte annetaan suoraan GameObjectille. Kääntymistä ja liikkumisnopeutta voidaan säätää komponentin muuttujilla.



Kuva 15. PlayerMovement-komponentin lopullinen muoto.

Sitten vain testailin hahmon liikkumista ja säädin muuttujien arvoja tai muutin koodia. Suurin aika meni säätämisessä. Kaltevien pintojen luominen testejä varten auttaa varmistamaan, että hahmo liikkuu sujuvasti erilaisilla maastoilla. Vaikka säätäminen voi olla aikaa vievää, se on välttämätöntä pelin toimimisen kannalta parhaalla mahdollisella tavalla. Säätöarvot ovat kuvattuna kuvassa 15.



Kuva 16. Lailan hyppyjen, tippumisen ja maahan laskeutumisen rakenne.

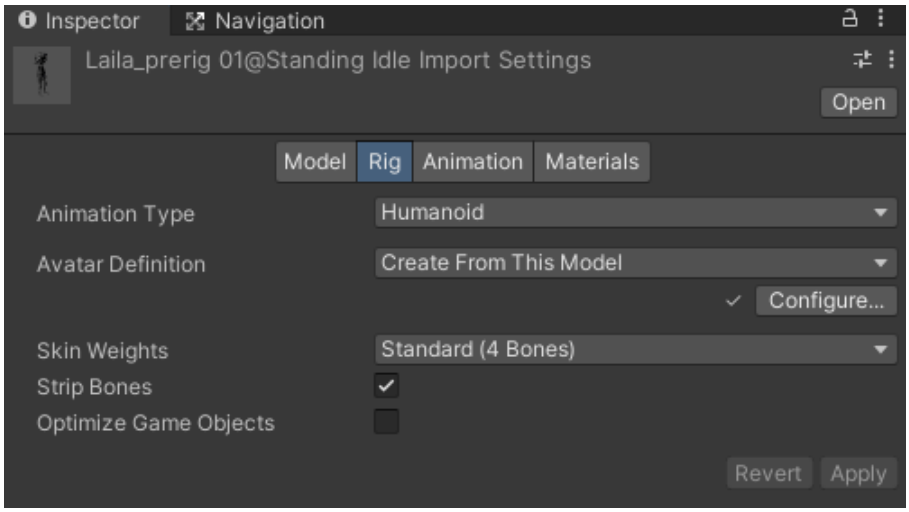
Loin myös hahmolle hyppimistoiminnon. Sen lisääminen voi esimerkiksi lisätä pelin monipuolisuutta ja haastavuutta. Kuvassa 16 on Animator Controllerissa kuvattuna miten erilaiset nuolet osoittavat eri animaatioihin, jotka sisältävät ehtoja, joita tulee toteuta, että kyseinen animaatio aktivoituu.

4.5 Lailan animaatiot

Lisäsin Lailalle Animator-komponentin. Komponentti on vastuussa animaatioiden toistamisesta ja säätämisestä. Se tarjoaa myös useita muita ominaisuuksia, kuten animaatioiden kerrostamisen ja sekoittamisen, liikkeiden interpoloinnin, animaatioiden ajoittamisen ja nopeuden hallinnan. Animator-komponentti tarvitsee Animator Controllerin ja hahmon avatarin toimiakseen.

Valitsin myös Animator-komponenttiin Apply Root Motion eli Animator-komponentin animaatiot ja liikkumisen olevan synkronismissa keskenään eli hahmo liikkuu ruudulla saman matkan animaation kanssa. Tämä nopeuttaa töitä, ettei tarvitse alkaa testailemaan ja etsimään ihanteellisia nopeuksia animaatiolle ja liikkumiselle.

4.5.1 Hahmon Rig Avatar ja Animator Controller



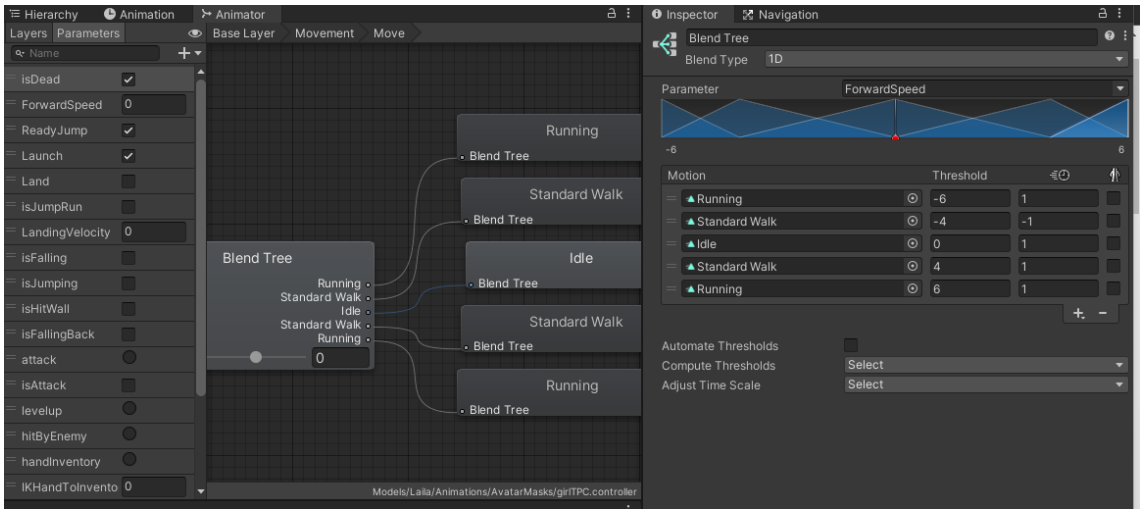
Kuva 17. Inspectorissa Lailan 3D-mallin asetukset kohdasta Rig.

Kuvassa 17 on kuvattu määritellyt säädöt Lailan 3D-mallille. Loin avatarin tästä 3D-mallista, joka toimii pohjana seuraaville tuleville animaatiolle. Hahmo on myös humanoidi, joten se avaa käyttöön myös Avatar Maskin, josta kerron toiminta-animaatiot osiossa.

Animator Controllerissa järjestellään ja säilytetään hahmon tai esineen animaatiota. Lailan 3D-malli on ihmismäinen hahmo, joten sillä voi luoda kerroksellisia animaatiota eli yhdistää eri animaatiota yhteen, painottaa niitä eli mikä animaatio dominoi ja luoda monimutkaisia animaatiojärjestelmiä, kuten Blend Tree.

Animator Controlleria ohjataan parametreilla eli esimerkiksi annetaan true-arvo parametrille isJumping ja siten hyppyanimaatio aktivoituu. Tämä animaatio on niin kauan voimassa, kunnes isJumping kumotaan false-arvolla. Parametriehtoja voi olla useita, jotta jokin tietty animaatio aktivoituu tai ei aktivoidu, kuten esimerkiksi että hahmo ei voisi hypätä, jos se on tippumassa. Lailan liikkumiskomennot tulevat PlayerMovement-komponentista, mutta joidenkin toimintojen aktivointi hahmolla tapahtuu muissa komponenteissa, koska halusin pitää hahmon eri toiminnot erillään selkeyden vuoksi.

4.5.2 Hahmon liikkuminen ja Blend Tree



Kuva 18. Vasemmalla kuvassa Animatorissa liikkumisen Blend Tree. Oikealla Blend Tree on avattu muokattavaksi Inspectoriin.

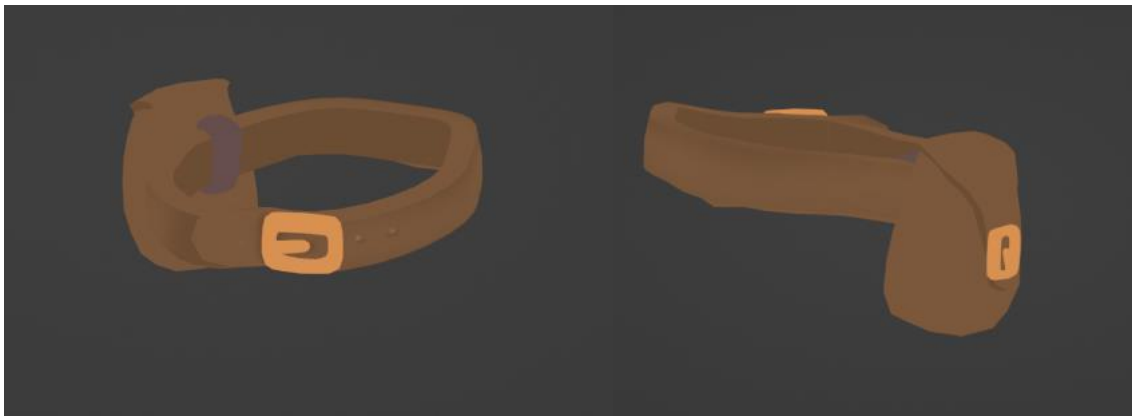
PlayerMovementin antaessa arvon parametri ForwardSpeedille ja muiden parametrien ollessa suotuisia päästään Blend Treehin. Blend Tree sekoittaa kolmea eri animaatiota, idle, kävely ja juoksu, riippuen millaisen arvon se saa parametrista. Nolla arvolla pysytään Idle-animaatiossa. Arvon kasvaessa hahmo siirtyy kävelyanimaatioon. Arvolla neljä kävelyanimaatio on optimaalinen. Jos arvo vielä nousee, muuttuu kävely sulasti juoksuksi, jonka maksimiarvo on kuusi. Samat positiiviset arvot voidaan kääntää negatiivisiksi, jolloin saadaan taaksepäin liikkuva hahmo. Myös animaation voi muokata liikkumaan taaksepäin muuttamalla nopeuden negatiiviseksi. Kuten tässä tein juoksulle ja kävelylle, muuten olisi pitänyt etsiä taaksepäin liikkuvat animaatiot. Kuvassa 18 on Blend Tree Lailan liikkumisesta.

4.6 Liikkumisesta mielenkiintoista

Lailan hahmo liikkui ruudulla hyvin ja animaatiot toimivat oikein, mutta hahmon liikkuminen on tylsää katsottavaa. Erilaiset visuaaliset efektit, kuten hahmon jättämä jälki tai pölyn tai savun muodostuminen liikkeen yhteydessä, voivat tehdä hahmon liikkumisesta vaikuttavamman näköistä. Lailan hahmon liikkumisen kehittäminen voi lisätä pelin viihdearvoa ja antaa pelaajalle uusia haasteita. Liike on tärkeä osa pelejä, sillä se kuvaa hahmon tapahtumia peliympäristössä, kun peli saa syötteen pelaajalta.

Vaikka käveleminen ja juokseminen ovat yleisiä liikkumistapoja peleissä, ne voivat olla hidastempoisia ja tylsiä. Siksi on tärkeää pitää pelaajan mielenkiintoa yllä muokkaamalla liikettä mielenkiintoisemmaksi. Erilaiset hahmon liikkumisnopeudet voivat luoda pelaajalle tunteen siitä, että he tekevät enemmän pelissä kuin todellisuudessa tekevät. Liikkeellä voi myös vaikuttaa pelin tyyliin ja tunnelmaan sekä tarinan kerrontaan hidastamalla tai tarjoamalla pelaajalle hengähdystauon. Lisäksi hahmon animaation liikkeillä on merkitystä pelaajille, sillä he joutuvat katsomaan hahmoa koko ajan. (Razbuten 2022.)

4.6.1 Staattinen vaate



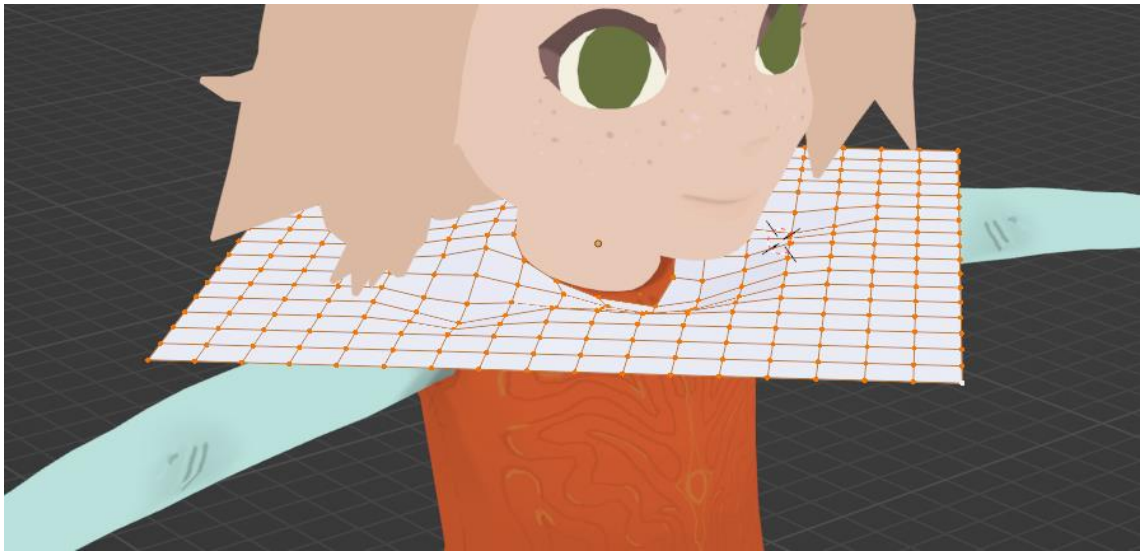
Kuva 19. Vyölaukku Blenderissä.

Epäsymmetrian luonti tuli minulle ensimmäisenä mieleen, kun olin testannut pelihahmon liikkumista. Miettimisen jälkeen ideaksi tuli ottaa toispuoleinen vyölaukku. Etsin netistä nahkalaukkujen kuvia inspiraation lähteeksi. Blenderissä aloin luoda Lailan mallille vyölaukkua, että mittasuhteet pysyisivät oikean kokoisina. Aluksi halusin, että vyölaukun pussin läpän saisi auki ja kiinni, kun Lailan käsi käy siellä, mutta jätin sen pois ajan vähyyden takia. Valmis vyölaukku on kuvassa 19 Käytin vain paria väriä tekstuureissa, koska Zelda-Shader varjostaa vyölaukun kauniisti. Valmiin vyölaukun sijoitin hahmon armaturen lantion luun kohdalle.

4.6.2 Dynaamiset vaatteet

Unityssa hyödynsin Cloth-komponenttia, jolla voidaan simuloida kankaan liikkeitä. Cloth-komponentti vaatii, että valitulla GameObjecktilla on Skinned Mesh Renderer-komponentti. Skinned Mesh Renderer eroaa tavallisesta Mesh Render-komponentista, että se renderöi Bone-animaatioita eli

rig-animaatiota. Skinned Mesh Renderer tekee vertekseistä nivelkohtia, joiden mukaan ”kangas” taipuu.



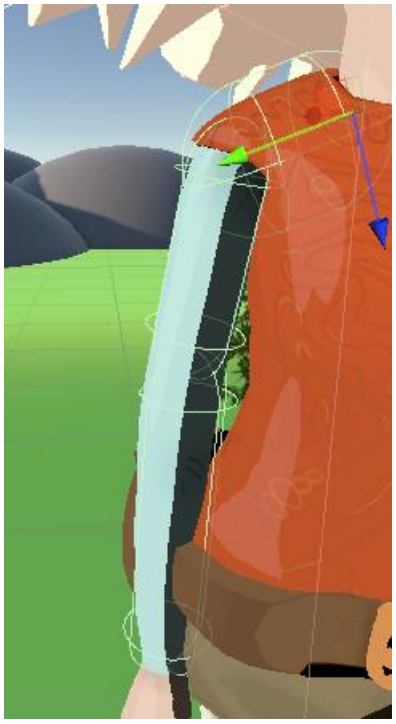
Kuva 20. Ponchon luominen Lailan 3D-malli pohjana Blenderissä.

Poncho-vaate oli alun perin ideana suojata Laila-hahmoa kylmillä alueilla, mutta se soveltuu myös rikkomaan liikkumisanimaatiota tuomalla epäsäännöllistä liikettä. Mutta millainen 3D-mallin pitäisi olla, että Skinned Mesh Renderer toimisi? Netistä löytyvillä ohjeilla ei löytynyt helposti vastauksia, joten tein erilaisia versioita ponchosta Blenderissä, jotka toin Unityyn ja testailin, miten Cloth-komponentti toimii eri 3D-vaatemalleille. Paras malli oli yksinkertainen neliö, johon oli tehty pääaukko, ja tasainen jako verteksiverkossa. Muokkasin vaatteen muotoa niin, että kaula-aukon kohdalle saatiin kiinteä kohta, joka ei liiku samalla tavalla kuin muu kangas. Se muodon näkee hyvin kuvassa 20.

Unityssä sijoitin 3D-ponchon Laila-hahmon armaturen olkapäille ennen päätä. Tämä mahdollistaa vaatteen liikkumisen pelitilassa yhdessä hahmon kanssa.

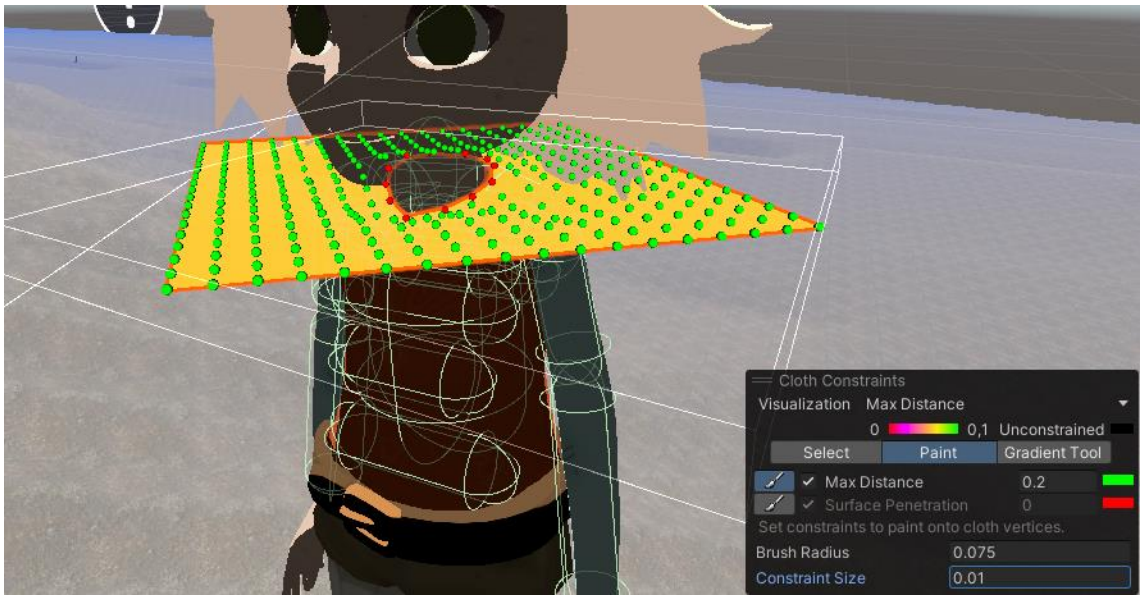
Cloth-komponentti vaatii kapseli – tai pallotörmäyttimen (Collider), joita vasten ponchon verteksit laskeutuvat. Jollei niitä aseteta, kangas menee läpi Lailan kehosta. Törmäyttimiä piti sijoittaa armatureen sen verran, että se kattaa alueen missä poncho roikkuisi pelitilassa. Poncho on sijoitettu armaturen kohtaan, jossa on haarautuva luuranko eli ranka, pää ja kädet. Animaatioissa on tällä alueella myös paljon liikettä, joissa poncho liikkuu mukana.

Laila tarvitsisi siis paljon törmäyttimiä. Hyvänä sääntönä on, että yhtä GameObjectia kohden on yksi törmäytin, koska useampi sekoittaa toisensa. Testailemalla, uudelleen sijoittamalla ja luomalla uusia GameObjeckteja ja törmäyttimiä, poncho tarvitsi lopulta viisitoista kapselitörmäyttimiä, että vaate ei mennyt mistään läpi ja vaatteen laskeutuminen Lailan päälle näytti luonnolliselta.



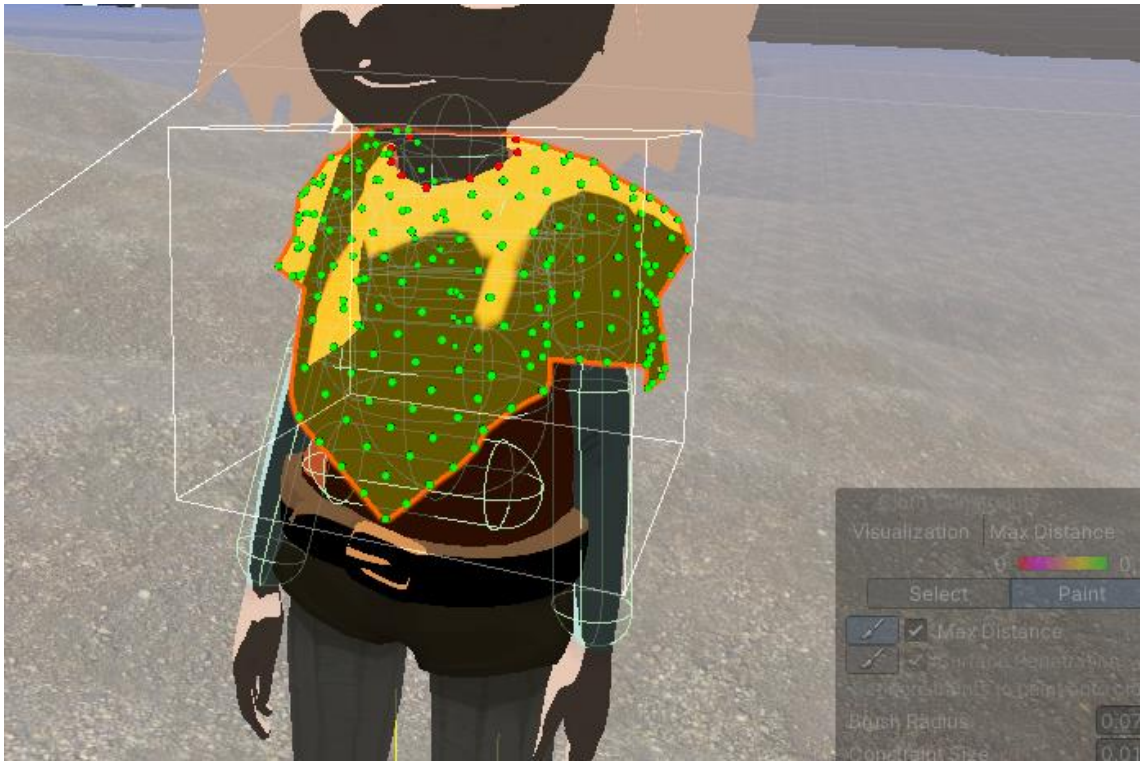
Kuva 21. Lailan oikea olkapää, jossa näkyy vihreitä kapselitörmäyttimiä.

Törmäyttimien piti olla suurempia kuin hahmon verteksiverkko. Kuvassa 21 näkee oikean käden törmäyttimien koon. Pelitilassa poncho laskeutuu törmäyttimen päälle ja siten vaate jää irti Lailan-mallista eikä mene hahmosta läpi. Jos törmäytin menisi aivan verteksiverkon pintaa myöten niin hahmon animaatiossa poncho meni hahmosta läpi useasti. Törmäyttimet liikkuvat hahmon armaturen mukana, mutta se ei suurene tai pienene, jos animaatiossa verteksiverkko muuttuu. Tämä testaaminen ja opiskelu vei kyllä oman aikansa. Alun perin ponchokin oli suurempi, mutta pienensin sitä koska siten minun ei tarvitsisi laittaa niin paljon törmäyttimiä.



Kuva 22. Ponchon verteksit on määritelty vihreällä aktiivisiksi ja punaisella epäaktiivisiksi.

Cloth-komponentti tarvitsi verteksien rajoitteet eli riggaus painot nollasta yhteen. Verteksit, jotka liikkuvat hahmon liikkeiden mukana eli nivelet on merkattu vihreällä, jotka näkyvät kuvassa 22. Pelin muokkaustilassa poncho oli jäykkä, mutta pelitilassa se laskeutui törmäyttimien päälle.



Kuva 23. Pelitilassa poncho laskeutuu törmäyttimien päälle.

Yleensä materiaalit kuten Lailan 3D-mallissa näkyy vain tahkon yhdellä puolella, koska mallin sisälle ei tarvitse nähdä. Muokkasin poncholle oman Zelda-Shaderin, jossa materiaali näkyy 3D-malleissa tahkojen etu- ja takapuolelle. Vaatteen liikkuessa se ei ollut enää läpinäkyvä toiselta puolelta. Kuvassa 23 on valmis poncho.

Materiaalissa on vain yksi väri. Silti poncho näytti monipuoliselta, koska siinä on erottuva graafinen varjo. Jos peliin haluaisi vaihtelevuutta ja erilaisia ponchoja, voisi vaihtaa materiaalin värin tai luoda tekstuureilla eri kuviota ponchoon, jolloin pitäisi tehdä uusi Shaderi. Näin saisi monta vaihtoehtoa luotua ponchoa helposti. Niille voisi sitten lisätä eri ominaisuuksia tuomaan pelihahmolle, vaikka lämpötilan kestävyyttä tai nostattamaan terveyttä.

Ongelma Cloth-komponentissa on, että se vie tietokoneelta laskentatehoja. Mitä enemmän vertekseja on vaatteessa, sitä enemmän joudutaan simuloimaan vaatteen liikkeitä. Eurogamerin artikkelissa (Purchese 2009) kerrotaan yhden animaattorin tehneen Batman: Arkham Asylum peliin kaksi vuotta animaatiota Batmanin viitalle, jotta viitan liikkeet olisivat realistisia. Lopulta animaatioita kertyi yli seitsemänsataa erilaisiin tilanteisiin.

4.7 Toiminta-animaatiot

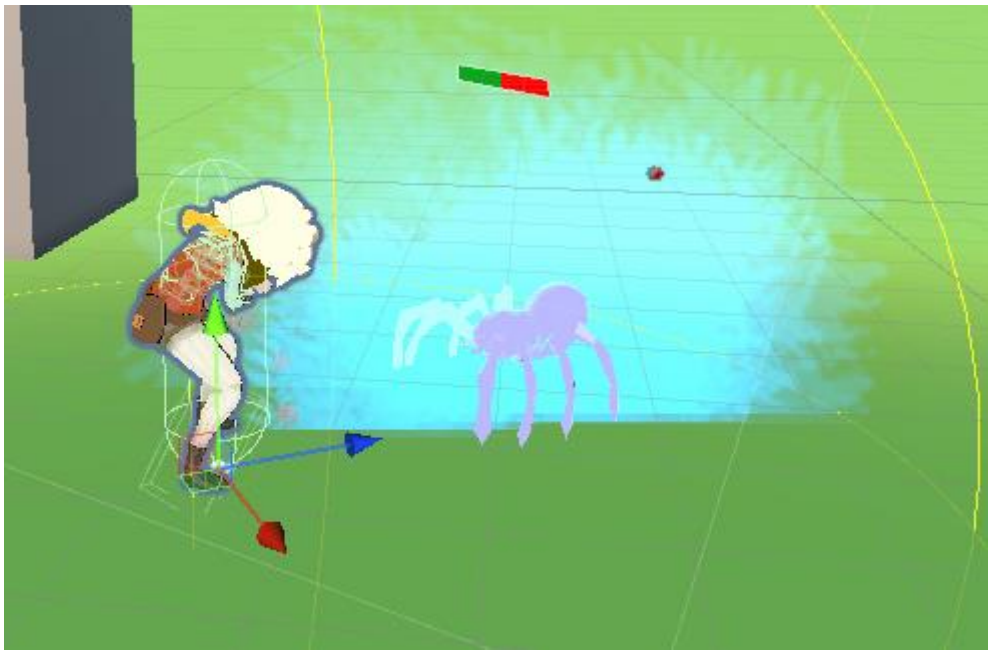
Kehittäessäni Lailaa tein hänelle terveyspisteet eli hän voi menettää terveyttään ja menehtyä eli tarvittiin iskunotto- ja kuolemisanimaatiot. Jonkun tai jonkin pitäisi vahingoittaa Lailaa, jotta voisin testata miten animaatiot taistelutilanteissa. Pelisuunnittelija oli puhunut yhdestä hirviöstä, joka asustaa saarella, mutta minulla ei ollut aikaa alkaa suunnittelemaan, tekemään 3D-mallia ja animaatioita sille.

4.7.1 Vihollinen

Kuva 24. Vasemmalla perushämähäkki. Oikealla muokattu versio.

Unity Storesta löysin hämähäkin, jolla oli perusanimaatiot: idle, liikkuminen, iskunotto, hyökkäys ja kuolema. Koska pelimaailmassa vallitsee pahanenteinen kylmyys, päätin lisätä hämähäkkiin valkoisen perus-Zelda-Shaderin, joka loi lumisen tunnelman olentoon. Kuitenkin halusin korostaa kylmyyden vaikutusta vielä enemmän, joten päätin lisätä erikoisefektin Particle Systemin avulla, joka on erikoistehosteiden työkalu, jolla voi luoda erilaisia partikkeleita peliin kuten tulta, sumua, räjähdyksiä, sadetta tai muita visuaalisia efektejä. Loin hohtavan huurteisen kylmyyden efektin hämähäkin ympärille. Hämähäkin liikkuessa ympäristössä aiemmin ilmaantuneet partikkelit eivät seuraa hämähäkkiä vaan muodostavat kylmän hohtavan polun. Ajan myötä partikkelit katoavat ja samoin myös kuljettu polku. Kuvassa 24 on alkuperäinen ja muokattu versio hämähäkistä.

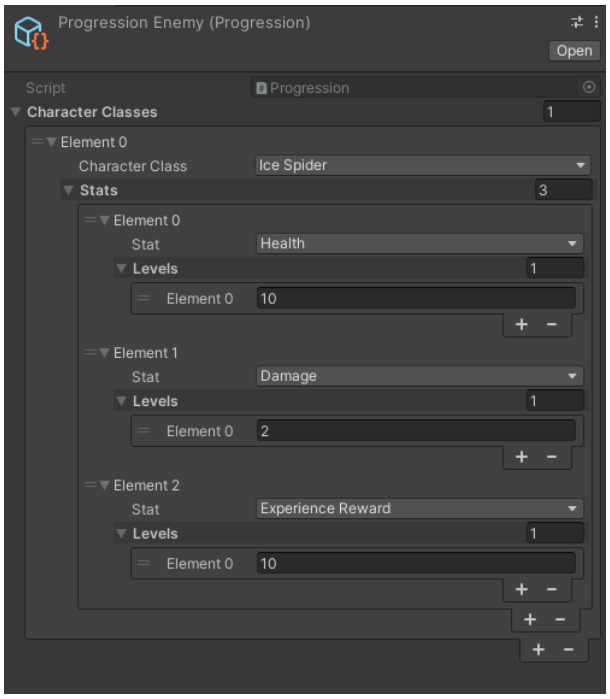
Pelitilassa jäähämähäkki hyödyntää Unityn navigointijärjestelmää, NavMeshiä, joka kattaa ennakkoon määritellyn pelialueen, jossa tekoälyä käyttävät GameObjectit voivat kulkea. NavMeshAgent-komponentti mahdollistaa hahmon liikkumisen NavMesh-verkossa, ja Mover-komponentti asettaa kohteen, johon jäähämähäkki pyrkii liikkumaan. Mover-komponentti myös määrittelee suurimman sallitun liikkumisnopeuden ja ohjaa liikkumisanimointia.



Kuva 25. Laila saa osuman jäähämähäkiltä.

Fighter Animal -komponentti puolestaan määritteli taistelun säännöt, kuten hyökkäysten välisen ajan, kohteen etäisyyden, jolloin jäähämähäkki siirtyy sen luokse sekä etäisyyden, jolloin hyökkäys ja hyökkäysanimaatio aktivoituvat. Jäähämähäkin terveyspisteet näkyvät palkissa sen yläpuolella, ja menettäessä terveyspisteitä, iskunottamisanimaatio aktivoituu. Kuvassa 25 hämähäkki on lopettanut hyökkäyksensä.

Hämähäkin hyökkäysanimaatio aktivoi FighterAnimal-komponentin Hit-funktion. Se tarkistaa onko vihollista, mitkä ovat vahinkopisteet kohteelle ja onko kohde tarpeeksi lähellä, että kohteen Health-komponentti voi ottaa vahinkopisteet vastaan.



Kuva 26. Progression-skriptattava olio, johon laittaa dataa pelin hirviöistä.

Jäähämähäkin Health-komponentti ottaa terveys-arvolukeman BaseStats-komponentista. BaseStatsiin on tallennettu skriptattava olio, joka on nimetty Progressioniksi, jonka näkee kuvassa 26. Hahmo- tai viholliskehitystietoja sisältävässä tiedostossa on hämähäkin perustiedot, kuten että kyseessä on tason yksi vihollinen, jonka terveyspisteet ovat kymmenen, vahinkoa se tuottaa kaksipistettä ja kymmenen kehityspistettä, jotka Laila-hahmo saa tappaessaan jäähämähäkin.

Jos kohde karkasi, jäähämähäkki tarkkailee sitä ennen kuin palaa takaisin aloituspaikalleen. Tekoälyä voi muokata AIAnimalControl-komponenttia säätämällä; kuinka kauan hämähäkki jahtaa kohdetta, kuinka kauan kadonnutta vihollista etsitään ja tarvittaessa muita hämähäkkejä voidaan kutsua avuksi, jos niitä on lähistöllä.

Taistelun jatkuessa ja jäähämähäkin menettäessä kaikki terveyspisteet, kuolema-animaatio aktivoituu. Particle Systems ei tuota enää jäähuurre-partikkeleita, jäähämähäkki muuttuu näkymättömäksi ja DestroyDeadEnemy-komponentti poistaa "tapetun" GameObjectin pelistä, kun sen viimeiset partikkelit ovat poistuneet pelistä.

Tämä jäähämähäkin Prefabsin toimii pohjana kaikille eläinvihollisille. Eli nyt jos olisi vain uusi eläin 3D-malli ja tarvittavat animaatiot niin uuden vihollisvariaation tekeminen olisi nopeaa ja helppoa pienellä säätämisellä.

4.7.2 Laila ja vihollinen

Lailalle onnistui helposti asettaa osumanottamisanimaatio jäähämähäkiltä, ja terveyspisteiden loppuessa kuolema-animaatio aktivoitui myös oikein säätämisen ja testaamisen jälkeen. Laila tarvitsisi seuraavaksi puolustautumiskeinon.

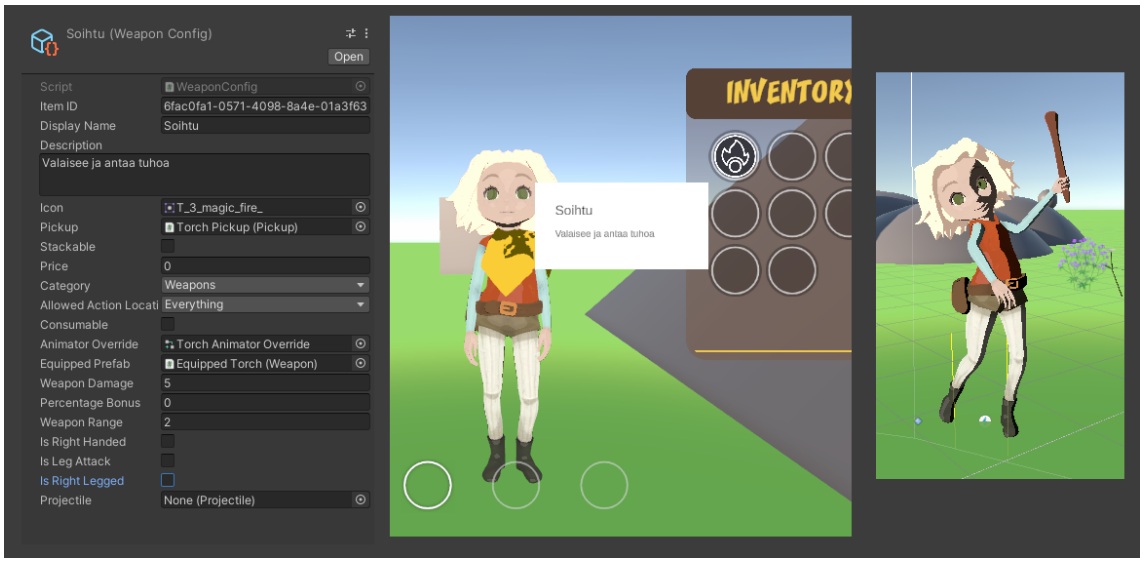
Kuva 27. Potkuanimaatio, jossa näkyy laatikkotörmäytin.

En kokenut Lailan olevan luonnostaan väkivaltainen, joten päätin antaa hänelle perushyökkäykseksi potkimisen. Jäähämähäkin hyökkäys tarkistaa onko vihollinen iskunotto alueen sisällä, mutta Lailalla tarkistus menee oikeassa jalassa olevan törmäyttimen triggerin aktivoinnin kautta. Kuvassa 27 näkee potkun iskualueen. Jos Lailalla on hyökkäys menossa, ja osuessa toiseen törmäyttimeen, joka on merkitty ”Enemy”-tagilla, käydään miltei samanlainen tarkastus kuin jäähämähäkillä. Erotuksena tarkastetaan, onko asetta käytössä ja mikä aseen vahingon tuotto on. Törmäytin myös aktivoidaan ja epäaktivoidaan, riippuen missä vaiheessa hyökkäys on menossa.

Lailalla on koko ajan käytössä toiminto, joka määräytyy pelitilan toimintakuvakerivillä olevasta aktiivisesta toimintakuvakkeesta. Toimintakuvaketta on kolme, joihin voi asettaa mieleisensä toiminnon kuten aseen taikka marjoja syötäväksi. Jos valitussa toimintakuvakkeessa ei ole mitään toi-

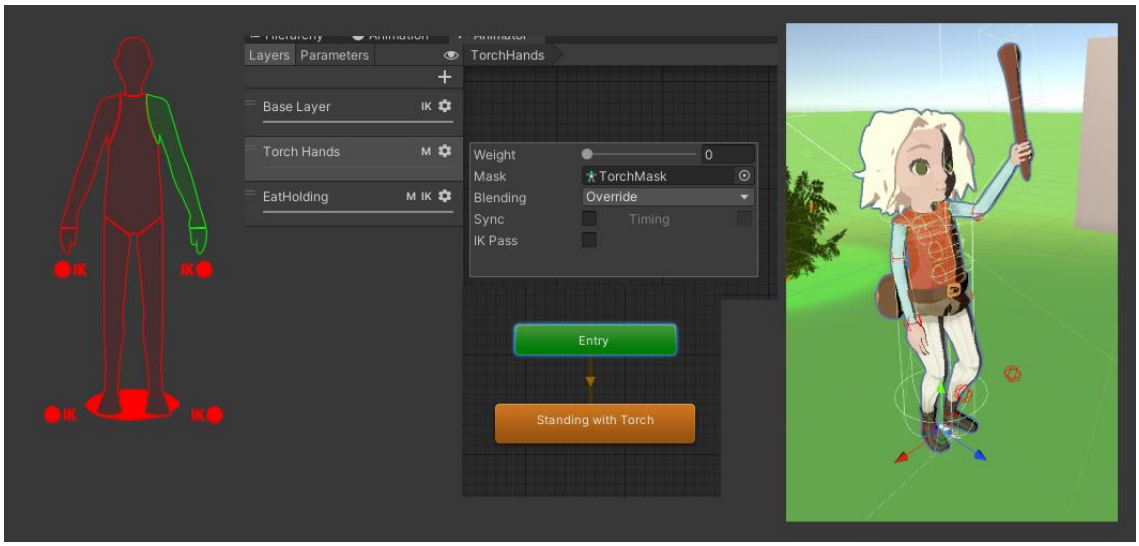
mintaa asetettuna, potkuhyökkäys toimii perustoimintona. Valitun toiminnan näkee eri värisenä toimintakuvakerivillä. Toiminta aktivoituu painamalla hiiren vasenta korvaa ja toiminta vaihdetaan pyörittämällä hiiren rullaa.

Toiminnan vaihtuessa eri komponentit käyvät läpi Animatorin parametrien ja Layerien eli kerrosten nollaamista tai aktivointia. Jos uudessa toiminnassa on esine, käydään läpi sen sijoitus armaturessa ja sijoitetaan se oikeaa kohtaan. Edellinen esine poistetaan pelistä.



Kuva 28. Vasemmalla soihdun tiedot. Keskellä näkyy pelitilassa soihtu tavaraluettelossa, ja kun hiiri on kuvakkeen päällä, näkyvät tiedot esineestä. Viimeisessä kuvassa Laila on lyömässä soihdulla.

Peliin on lisätty esineitä ja aseita, joita Laila voi kantaa tavaraluettelossaan. Niitä voi pudottaa pelimaailmassa, jossa ne asettuvat NavMeshille eivätkä esimerkiksi katoa seinien sisään. Eräs näistä on soihtu, joka toimii myös aseena. Kuvassa 28 Laila käyttää soihtua hyökkäysanimaatiossa. Aseet ja esineet ovat skriptattavia olioita, joissa kerrotaan eri esineiden ominaisuuksista, onko esine oikea- vai vasenkätinen, Animator Override, joka ylikirjoittaa perushyökkäyksen animaation aseelle ominaisella hyökkäysanimaatiolla, iskuvoiman tuho ja muita yksityiskohtia.



Kuva 29. Vasemmalla Humanoid-maski, jossa valittuna on vasen käsi. Sen vieressä Animatorin kerroksen asetukset ja alla näkyy kerroksen ainut animaatio. Oikeanpuoleisessa kuvassa painotusarvo on yksi ja ylikirjoitus on ottanut vallan vasemmasta kädestä.

Animatorissa on Layers-ominaisuus, jota kutsun kerrokseksi. Peruskerros toimii pohjana muille kerroksille, johon kannattaa laittaa perusanimaatiot kuten juoksut ja hypyt. Seuraavat kerrokset ylikirjoittavat alemmat kerrokset, jos niiden painotusarvoa muutetaan. Kerroksien painotusarvo on nollasta yhteen.

Jos halutaan esimerkiksi nostaa vasen käsi ylös jostain animaatiosta, mutta pitää perusanimaatiot voimassa, luodaan sille oma kerros. Kerrokselle asetetaan Avatar Mask, kuvassa 29, jossa määritelty mihin hahmon kehon osaan halutaan animaation vaikuttavan, yllä olevassa kuvassa vihreä käsi on valittu. Kerroksen animaatioksi on asetettu vain yksi animaatio, jossa ihmishahmo pitää soihtua. Jos painotusarvoksi laitetaan yksi suoraan, soihtukäsi myös nousee heti pystyyn. Jos painotusarvoa alkaisi pikkuhiljaa kasvattamaan niin käsi alkaisi samassa tahdissa nousemaan ylös. Painotusarvoa laskiessa alkaa myös käsi laskea ja silloin peruskerroksen sen hetkinen animaatio kokonaan vallassa. Kuvassa 30 Laila käyttää soihtua aseena.

Kuva 30. Jäähämähäkki tekee kuolemaa, Laila on menettänyt energiaa ja soihtuhyökkäys-toiminto on kolmannessa toimintakuvakkeessa ja hän saavuttaa uuden hahmotason erikoisefektin kera.

4.8 Loppuvaihe

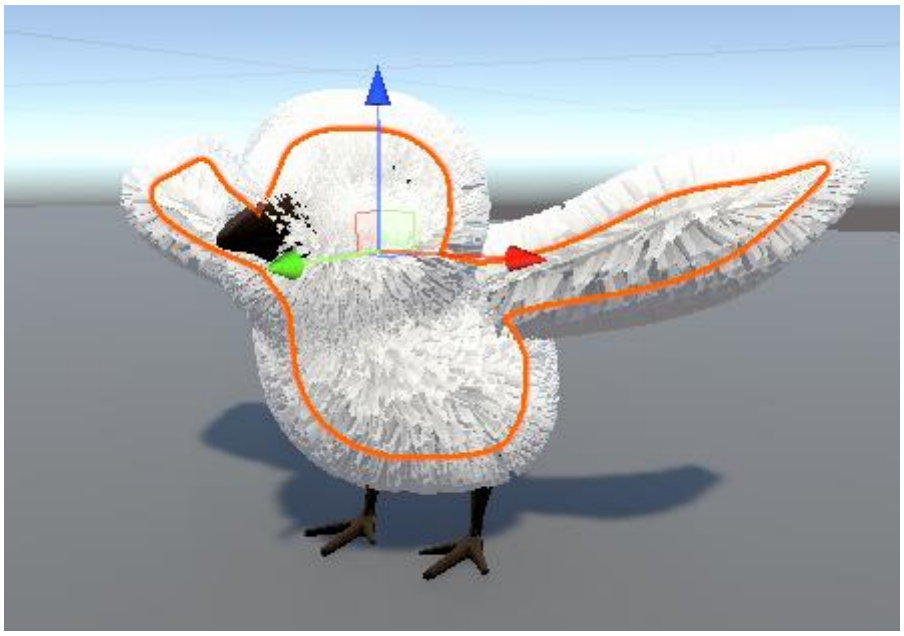
Alun perin tein Lailan hahmon juoksemaan tasaisella alustalla, jossa oli hieman epätasaisuuksia eli neliömäisiä pintoja, joka olivat hieman kaltevassa kulmassa. Kun käytin Terrain-työkalua maaston tekemiseen, huomasin suuria muutoksia hahmon liikkumisessa. Tällä hetkellä hahmo liikkuu animaation tahdissa, eli animaation liike ja kuljettu matka vastaavat toisiaan. Epätasaiset kaltevuudet sekoittavat maatarkastuksen, joka laskee etäisyyttä maahan ja asettaa hahmon tippumisanimaatioon, jos maa on liian kaukana. Kun maatarkastus huomaa hahmon taas olevan kosketuksessa maahan, pyöritetään maahan laskeutumisanimaatio, ja heti sama kuvio uudestaan. Nyt hahmo juoksi reippaasti yli isoissa pinnanmuodon kaltevissa kohdissa, ja maatarkastus sekosi taas.

Otin käyttöön pinnanmittauksen molemmille jaloille, jotta hahmo reagoisi paremmin epätasaisiin pintoihin. Liikkuminen jäi nyt hyvin töksähtäväksi epätasaisilla pinnoilla. Projektin aikataulu loppui kesken. Tämähän oli prototyyppi.

4.9 Miten jatkaa eteenpäin

Seuraavaksi pitäisi luoda NPC-hahmot peliin, saada heiltä tehtäviä ja suorittaa niitä. Tätä varten pitäisi miettiä, miten keskustelua NPC-hahmon kanssa käydään pelaajalle ymmärrettävästi. Eli UI-suunnittelua ja sen toteuttamista.

Olin miettinyt kovasti, miten loisin vuorokauden muutoksen peliin ja erilaiset kylmät alueet tai korkeuserot vaikuttaisivat kylmyyteen. Keräilin aiemmin jo tietoa aiheesta ja tiesin, miten lähtisin kehittämään niiden toteutusta, mutta en päässyt niin pitkälle projektissa.



Kuva 31. Keskeneräinen lokki, joka on tuotu Blederistä Unityyn Shader-testaukseen.

Peliä pitäisi alkaa suunnittelemaan myös, miten ja mitä pelidataa kerrotaan pelaajalle: tekemättömät ja tehdyt tehtävät, missä päin kasvaa jokin kerättävä kukka, miten hahmon eri taitojen kehitys tapahtuu ja miten se näytetään ja paljon muuta. Lokin kehitys jäi kokonaan pois, koska siitä olisi voinut oman opinnäytetyöaiheen. Kuvassa 31 on alkuvaiheen lokin malli, jonka kehitys jäi kesken. Siltä puuttuu ”räpylät” ja silmät, koska päätin keskittyä muuhun pelikehityksessä.

Eräs asia, mitä emme suunnitelleet ollenkaan olivat äänet. Musiikista kerkesimme jutella, mutta luonnon ja hahmon äänet eivät olleet kehityslistalla.

5 JOHTOPÄÄTÖS

Pelisuunnittelijan kanssa pidimme palavereita tasaisesti kerran viikossa. Kommunikoimme aina kun tarvetta tuli kuten nopeita mielipiteitä, että kehitys saattoi jatkua. Pelisuunnittelijalla ei ollut riittävästi aikaa suunnitella peliä, mutta prototyyppiä pystyi silti kehittämään alueilla, joissa ei tarvittu pelisuunnittelijan ideoita, koska yhteinen näkemys niistä oli olemassa

Projektin ollessa melko avonainen eli ei ollut suunnitelmia, joita toteuttaa. Joten kehitin ja testailin peliä. Projekti turhautti, koska mitään ei ollut tapahtunut. Jonkin ajan päästä ymmärsin, että projektin suunnitelmat olivat pysähtyneet ja eikä ollut etappeja eli en saanut projektista onnistumisen iloa, vaikka prototyyppi oli hyvällä mallilla. Sitten tein itselleni tavoitteita, ja mielekkyys tehdä prototyyppiä tuli takaisin.

En osaa sanoa mikä meni parhaiten projektissa. Itse kuitenkin nautin eniten onnistumisen ilosta, kun sain toimintakuvakkeet toimimaan oikein. Tiesin, miten halusin niiden toimivan, joten tiedon etsiminen, kokeilu ja testaaminen toivat jännitystä, olenko oikeilla jäljillä. Lopputulokseen olin tyytyväinen. Pidin siitä myös, että tein Lailan 3D-mallin alusta asti ja sain sen toimimaan Unityssä. Tekisin siihen jo muita muutoksia kuten kasvot, jotta saisin niihin ilmeitä.

Ongelmatilanteissa auttoi Unityn dokumenttien lukeminen ja googlettamien. Suurin osa aikaa oli testaamista eli muokkasin koodia ja testasin, miten se vaikutti peliin. Idea oli päässäni, mutta en tiennyt, miten saisin sen toimimaan oikein. Peliin lisäsin myös erilaisia omia virityksiä, joita en mahtanut sisällyttää tähän kirjalliseen osioon; tavaraluettelon ja tavaroiden koodien kehitykset, hyppimiset, törmäykset seiniin, lokkiapulaisen 3D-malli ja sen Shader-testaukset. Animaatiosta jäi pois selitys, miten Laila syö marjoja, niistä saa energiaa ja ottaa uuden marjan vyölaukusta.

Olen kuitenkin tykästynyt peliin ja harmittaa, että se jää pahasti kesken. Monta kertaa tuli itselleni myönnettyä, että peliä tehdessä vaaditaan paljon erilaisia taitoja ja parasta on ryhmätyö, jossa jokainen saa tuoda parhaat puolet esille.

Kirjoittamisessa piti rajata todella paljon aihetta. Halusin kertoa aiheesta niin, että sillä olisi opettavainen näkökulma. Toivottavasti joku innokas saisi vastauksia kysymyksiinsä.

LÄHTEET

Blender Manual, 2023. Rigging. Hakupäivä 9.1.2023. https://docs.blender.org/manual/en/3.3/animation/introduction.html#rigging.

Çamönü, İ. 2020. What is MonoBehaviour? Hakupäivä 28.2.2023. https://www.codinblack.com/what-is-monobehaviour/.

Colligan, A. & Sar, R. 2019. The History of Minecraft. Hakupäivä 19.1.2023. https://www.thescienceacademystemmagnet.org/2019/12/20/the-history-of-minecraft/.

Eden, M. 2020. Testing A Game Prototype Step-By-Step. Hakupäivä 13.1.2023. https://meliorgames.com/game-development/testing-a-game-prototype-step-by-step/.

Haas, J. K. 2014. A History of the Unity Game Engine. Worcester: Worcester Polytechnic Institute.

Mignano, M. 2016. Use Paper Prototyping to design your game. Hakupäivä 22.3.2023. https://www.gamedeveloper.com/design/use-paper-prototyping-to-design-your-games.

Mixamo, 2023. Empowering creativity with animated 3D characters. Hakupäivä 9.1.2023. https://www.mixamo.com/#/.

Polsinelli, P. 2018. What is a game prototype? Hakupäivä 13.1.2023. https://ppolsinelli.medium.com/what-is-a-game-prototype-506472ec67ad.

Purchese, R 2009. Eidos expects Batman To Score in the 90s. Hakupäivä 26.1.2023. https://www.eurogamer.net/eidos-expects-batman-to-score-in-the-90s.

Razbuten 2022. How Games Make ”Boring” Movement Seen Interesting. Video. Hakupäivä 19.1.2023. https://www.youtube.com/watch?v=Vb-lv-YjnfI.

Starloop Studios 2020. Rapid Game Prototyping: Why is it Important in Game Development? Hakupäivä 19.1.2023. https://starloopstudios.com/rapid-game-prototyping-why-is-it-important-in-game-development/.

Unity Learn 2022. Get started with prototyping. Hakupäivä 29.12.2022. https://learn.unity.com/tutorial/get-started-with-prototyping?pathwayId=61a65568edbc2a00206076dd&missionId=61a617feedbc2a00253d67ee#.

Unity Technologies 2022a. Mesh.boneWight. Hakupäivä 9.1.2023. https://docs.unity3d.com/ScriptReference/Mesh-boneWeights.html.

Unity Technologies 2020. GameObject components for input. Hakupäivä 10.1.2023. https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/Components.html.

Unity Technologies 2021. Important Classes. Hakupäivä 28.2.2023. https://docs.unity3d.com/Manual/TimeFrameManagement.html.

Unity Technologies 2023a. Unity. Hakupäivä 16.1.2023. https://unity.com/.

Unity Technologies 2023b. Materials. Hakupäivä 26.1.2023. https://docs.unity3d.com/Manual/Materials.html.

Victorino, L. 2015. What you should know about game prototypes. Hakupäivä 12.1.2023, https://lvictorino.com/blog/what-is-a-prototype.html.

Wikipedia 2022. Armature (computer animation). Hakupäivä 29.12.2022. https://en.wikipedia.org/wiki/Armature_(computer_animation).

Wikipedia 2023. Unity (pelimoottori). Hakupäivä 16.1.2023. https://fi.wikipedia.org/wiki/Unity_(pelimoottori).