Tomi Viitanen

PELIHAHMON 3D-MALLINTAMINEN JA LIITTÄMINEN INTERAKTIIVISEKSI PELIMOOTTORIIN

Tietotekniikan koulutusohjelma
2018

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

PELIHAHMON 3D-MALLINTAMINEN JA LIITTÄMINEN INTERAKTIIVISEKSI PELIMOOTTORIIN

Viitanen, Tomi
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Helmikuu 2018
Sivumäärä: 34
Liitteitä: 2

Asiasanat: 3D-mallinnus, pelimoottori, Unity, pelihahmo

Tämän opinnäytetyön tarkoitus on luoda pelihahmo mobiilipeliin, osana toimeksiantajayrityksen Nopia Oy:n isompaa peliprojektia. Tässä työssä käydään läpi 3D-pelihahmon kehityksen työvaiheita hyödyntäen pelinkehitysympäristö Unitya ja 3D-mallinnusohjelma Blenderia. Lopuksi selvitetään, miten hahmo saadaan interaktiiviseksi käyttäjälle.

Pelihahmon kehitys aloitetaan suunnittelemalla konsepti, jonka jälkeen tutustaan miten 2D-kuva voidaan muuttaa kolmiulotteiseksi mallinnukseksi. Seuraavaksi hahmolle luodaan lyhyitä animaatioita ja tutustutaan, miten ne saadaan lopulliseen mobiilipeliin.

Lisäksi opinnäytetyössä selvennettään 3D-ohjelmistojen peruskäsitteitä sekä pelimoottorin etuja.

3D-MODELING A GAME CHARACTER AND MAKING IT INTERACTIVE IN GAME ENGINE

Viitanen, Tomi
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Computer Science
February 2018
Number of pages: 34
Appendices: 2

Keywords: 3D-modelling, game engine, Unity, game character

The purpose of this thesis was to create a game character for a mobile game. In this thesis we will go through different phases of developing 3D character, using Unity-game engine and 3D modeling software Blender. Lastly, we will clarify how to make the character interactive to a user.

Development of a game character is started by concepting it, after that we will sort out how to turn 2D image to a 3D model. Then the character is being animated, and we will go through how we will import those animations in the final game.

This thesis also includes explanation of some basic knowledge of game engine and 3D software products.

LYHENTEET JA KÄSITTEET

ASSET	Yksittäinen resurssi Unityssa, voi olla esimerksi skripti tai objekti.
C#	Microsoftin .NET-konseptia varten kehittämä ohjelmointikieli.
FPS	Lyhenne sanoista Frames Per Second. Näytölle sekunnissa piirrettävien kuvien määrä.
FRAME	Yksittäinen kuva.
INDIE	Yleensä pienellä budjetilla toimiva itsenäinen tekijä/tekijäryhmä. Ei toimi minkään suuren julkaisijan alaisuudessa.
PHOTOSHOP	Adoben kehittämä kuvienmuokkaamiseen tarkoitettu tietokoneohjelma.
RENDERÖINTI	Kuvan luominen tietokoneohjelman avulla.
RIGGAUS	Hahmolle määritelty luuranko, joka koostuu luista ja nivelistä sekä niiden ohjaamiseen tarkoitetuista kontrolleista, joilla hahmo voidaan animoida.
SCULPTAUS	Poligonien muotoilua vaihtoehtoisella tavalla. Kuin saven muotoilua, mutta digitaalisesti poligonien muodossa.
SHADER	Määrittelee miten 3D-malliin osuvaa valoa käsitellään.
SKRIPTI	Komentosarja. Lyhyt tietokoneohjelma.
TEKSTUURI	Pintakuvio. Yleensä kaksiulotteinen bittikarttakuva, jota käytetään geometrisen muodon pinnoittamiseen.

SISÄLLYS

1 JOHDANTO

Videopelien historia ylettyy 1940-luvun lopulle. Se on nykypäivänä nopeimmin kasvava viihdeteollisuuden ala maailmanlaajuisesti. Tekniikan kehittymisen myötä pelit ja niiden kehittäminen on kokenut täydellisen muodonmuutoksen. Samoin mobiililaitteiden kehitys sekä digitaalinen jakelu ovat vaikuttaneet alaan radikaalisti 2000-luvulla. Tämä on mahdollistanut pienten studioiden sekä itsenäisten pelinkehittäjien menestymisen välillä erittäinkin yksinkertaisilla peli-ideoilla. Harva raapustaa nykypäivänä alusta loppuun koko pelinsä sisältämän koodin itse. Nykyään valmiiden pelimoottoreiden takia siihen pystyy melkein kuka vain, joka jaksaa asiaan vähän perehtyä.

Tämä työ jakautuu kolmeen osaan. Ensimmäisessä osassa tutustutaan 3D-mallintamisen perusteisiin, ja käydään läpi kolmiulotteista avaruutta sekä mallintamiseen käytettäviä ohjelmistoja sekä työkaluja. Toisessa osassa pääpaino on pelimoottorin selvittäminen käsitteenä, sekä tutustua Unity-pelimoottoriin.

Kolmannessa osassa käsitellään eri vaiheita, joita pelihahmon valmistamiseen tarvitaan. Pelihahmon testaamiseen tehtiin opinnäytetyön ohessa X-Runner -niminen mobiiliminipeli, osaksi toimeksiantajayrityksen Nopia Oy:n peliprojektia. Pelin perusidea pohjautuu Endless Runner – tyyppisiin peleihin, jossa pelihahmo kulkee taukoamatta eteenpäin, eikä pelaaja voi jarruttaa tai pysäyttää hahmoaan. Hahmon kulkiessa jatkuvasti eteenpäin tulee vastaan erinäisiä esteitä, joihin osumista tulee välttää siirtymällä sivulle, hyppäämällä yli tai liukumalla ali. Pelihahmo animoitiin reagoimaan näihin liikkeisiin käyttäjän syötteistä.

2 3D-MALLINTAMINEN

2.1 Mitä on 3D-mallinnus?

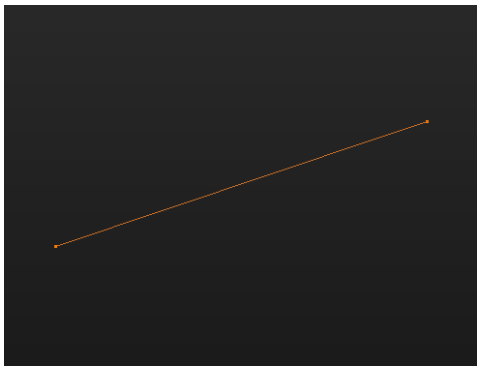
3D-mallitaminen on prosessi, jossa tietokoneella luodaan esineestä 3D-malli kolmiulotteisessa avaruudessa muokkaamalla pintoja, reunoja ja verteksejä (kärkipisteitä). 3D-malli voidaan saavuttaa siihen suunnatuilla 3D-ohjelmilla tai skannaamalla koneellisesti reaalimaailmasta esineitä tai ympäristöä. (Slick 2018.) Kolmiulotteisia mallinnuksia hyödynnetään muun muassa 3D-tulostuksessa, animaatioissa, videopeleissä, arkkitehtuurissa ja tekniikan aloilla. (Chakravorty 2017.)

2.2 3D-mallinnuksen periaatteet

Malli voi olla mitä tahansa 3D-ohjelmistossa, joka rakentuu geometriasta. Kaikkein tärkein muoto, joka toimii rakennuspalikkana kaikelle muulle, 3D -geometrisen maailman atomi, on kolmio. Kolmiopoligoni on yksinkertaisin muoto poligonista: se muodostuu kolmesta reunasta ja verteksistä. Yhdessä useat toisiinsa liitetyt kolmiot muodostavat 3D-malleja. (Zeman 2015, 23.) Seuraavassa kolme komponenttia, jotka jokainen poligoni vaatii muodostuakseen.

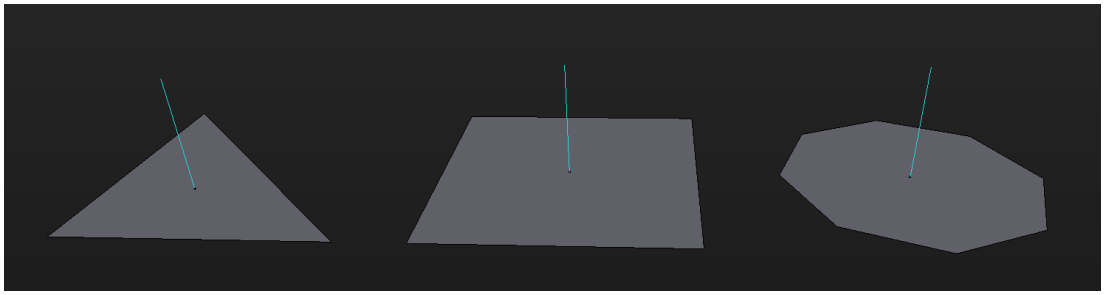
- Verteksit eli pisteet ovat tärkeimpiä aspekteja poligonille. Verteksi on piste 3D-avaruudessa, jolla on aina koordinaatit. (Zeman 2015, 24) Ilman verteksejä ei voi olla reunoja, eikä pintoja.

- Edget eli reunat ovat suoria viivoja, jotka yhdistävät kaksi verteksiä. (Kuva 1.) (Zeman 2015, 24)



Kuva 1. Reuna yhdistää kaksi verteksiä

- Face eli pinta on taso, joka muodostuu valittujen pisteiden ja reunojen väliin. Lopullisessa renderöinnissä vain pinnat näkyvät käyttäjälle, sillä se muodostuu pikseleiksi näytölle. (Zeman 2015, 24.) Pinnoilla voi olla eri määrä reunoja ja verteksejä, joiden väliin se muodostuu. Pintaa, jolla on neljä verteksiä ja reunaa, kutsutaan quad:ksi, Jos pinnalla on viisi verteksiä ja reunaa tai enemmän, kutsutaan sitä ngon:ksi (Kuva 2.)



Kuva 2. Kuvassa vasemmalta oikealle, triangle-, quad- ja ngon -pinnat sekä niiden normaalit.

Lisäksi poligonilla on aina myös normaali eli suora, joka on kohtisuorassa, tai 90-asteen kulmassa pintaan nähden. Käytännössä tämä kertoo mihin suuntaan pinta osoittaa. (Zeman 2015, 25.) Yksi pinta ei voi osoittaa, kuin yhteen suuntaan.

Kolmiopoligoni on siis kaikkein yksinkertaisin poligonimuoto. Nelikulmaiset sekä ngon:t pystytään aina jakamaan kolmioihin. Pelimoottorit suosivat kolmiopoligoneja, kun taas mallinnuksessa tulisi aina käyttää nelikulmaisia poligoneja. (Pluralsight www-sivut 2014) Reunoja ja pintoja voidaan muokata niitä liikuttamalla, kiertämällä

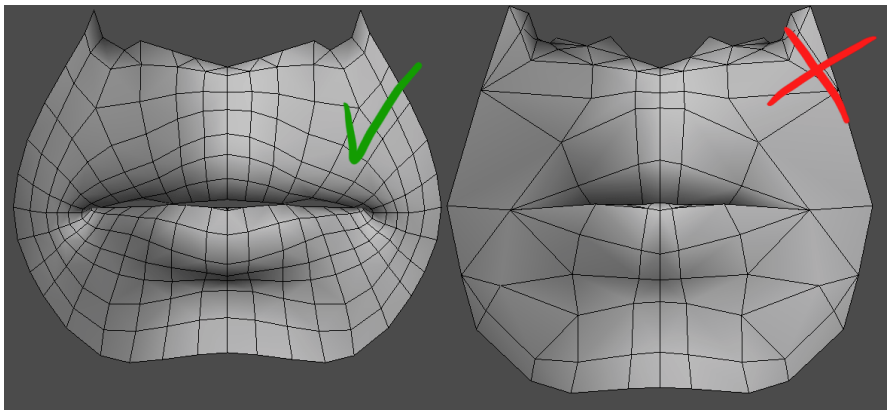
tai muuttamalla kokoa. Verteksit ovat sen sijaan itsenäisiä pisteitä, joita voidaan liikuttaa 3D-avaruudessa, vaikuttamatta muihin vertekseihin. Yhdessä nämä verteksit ja reunat muodostavat verkon jota kutsutaan topologiaksi. (Zeman 2015, 25.)

2.2.1 Topologia

Topologia viittaa 3D-objektin geometriseen pintaan, tarkemmin sanottuna sitä, miten se muodostuu. Tietokoneella generoidulla 3D-topologialla on tavoitteena saada tarvittava määrä yksityiskohtia mallinnukseen, kuitenkin mahdollisimman vähäisellä määrällä pintoja säästääkseen tietokoneen laskentatehoa. (Slick 2017.)

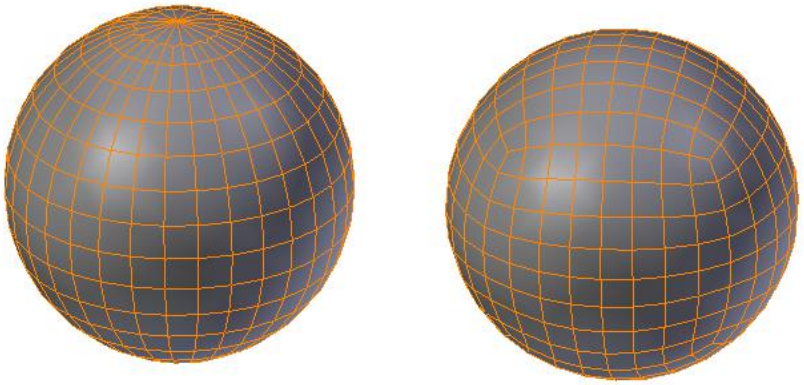
Poligonien määrä on erityisen tärkeä aspekti videopeleissä, jossa grafiikka piirtyy ruudulle reaaliajassa, niin että sen kanssa voi olla vuorovaikutuksessa. Videopeleissä uusi kuva voi renderöityä usein jopa kolmekymmentä kertaa sekunnissa, joka sallii pelattavuuden.
Mitään tarkkaa rajaa poligonien määrälle ei kuitenkaan voida määritellä, sillä riippuu myös pitkälti tekstuurien resoluutiosta ja pelin optimoinnista kuinka paljon laskentatehoa peli vaatii. Isojen pelitalojen ja budjettien tietokonepeleissä saattaa nykyään olla lähes miljoona poligonia näkyvissä kerrallaan, mobiilipeleissä määrä on huomattavasti pienempi. 3D-elokuvissa sen sijaan saattaa nykypäivänä olla jopa 300 miljoonaa poligonia yhdessä kohtauksessa. Toisin kun peleissä, videoalalla grafiikan ei tarvitse päivittyä reaaliajassa, vaan yhden kuvan renderöinti saattaa kestää tunteja tai jopa päiviä.

3D-mallintajat pyrkivät puhtaaseen topologiaan, mutta erityisesti jos mallia tullaan animoimaan, on hyvä topologia elinehto. Tällä tarkoitetaan poligonien levittämistä tasaisesti, ja reunasilmukoiden sijoittelua suhteessa tiheämmin tarvittaviin paikkoihin. (Slick 2017.) Esimerkiksi kasvoissa usein sijoitellaan silmukat tiheästi suun ja silmien ympärille, jossa tapahtuu paljon liikehdintää. (Kuva 3.)



Kuva 3. Esimerkki hyvästä ja huonosta topologiasta. (MDU 115: Foundations of 3D Graphics www-sivut)

Huomioitavaa on myös, että saman näköisillä esineillä voi olla useampia erilaisia topologioita. Tätä havainnollistaa Kuva 4.



Kuva 4. Kahdella pallolla eri topologia.

2.2.2 XYZ-akselit

3D-avaruus koostuu X-, Y- ja Z-akseleista. Nämä kolme ulottuvuutta vastaavat kaikkien objektien sijainnista, jotka ovat kolmiulotteisessa tilassa. 3D-avaruudessa on yksi piste, josta kaikki muu mitataan. Tämä piste on koko avaruuden keskipiste, ja sijaitsee paikassa jossa kaikki kolme akselia leikkaavat toisensa. Matematiikassa pistettä kutsutaan origoksi, mutta 3D-grafiikassa se tunnetaan yleensä world origin -nimellä. (Zeman 2015, 2.)

Jos kolmiulotteista tilaa tarkastellaan suoraan edestäpäin, horisontaalinen akseli tunnetaan aina X-akselina. Kun puhutaan 3D-grafiikasta, on olemassa kahdenlaisia koordinaattimenetelmiä. Y-ylös ja Z-ylös. Y-ylös on standardi animoinnissa, ja Z-ylös on yleensä standardi arkkitehtuurissa ja tekniikan aloilla. (Zeman 2015, 2.) Näin ollen molemmat Z- ja Y-akselit suorittavat pysty- ja syvyysakselin virkaa 3D-ohjelmistosta riippuen.

2.2.3 Objektien ja poligonien muokkaaminen

Jokaisella objektilla on oma pivot point eli kääntöpisteeksi kutsuttu piste, josta sen geometrian muuntautuminen mitataan ja lasketaan. Sen sijaintia voidaan vaihtaa mihin tahansa koordinaatteihin. (Zeman 2015, 18.) On olemassa kolme tavanomaista tapaa, jolla voidaan muokata objektin muotoa. Translate, rotate ja scale. (Zeman 2015, 8.)

- Translate on yksinkertaisin muuntautumismuoto objektille. Se mahdollistaa objektin koordinaattien muuttamisen kaikilla akseleilla. Toisin sanoen se liikuttaa objektia 3D-avaruudessa. (Zeman 2015, 9) Translate on ainoa muuntautumismuoto, jossa pivot point liikkuu objektin mukana.

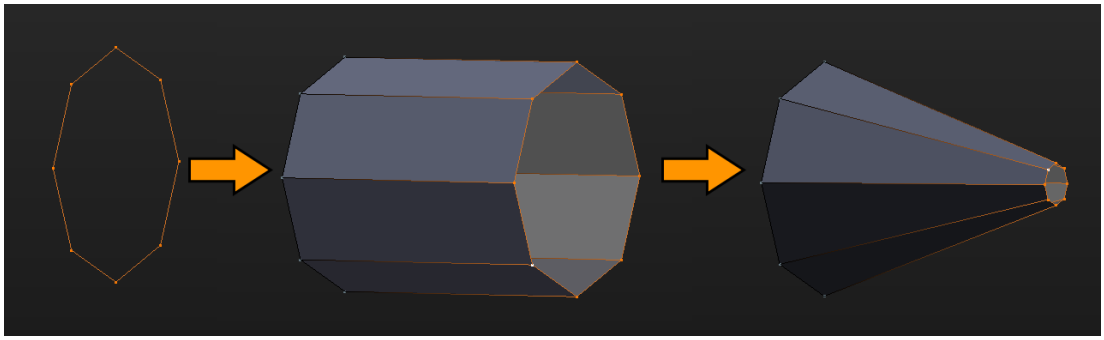
- Rotation on myös objektin liikuttamista, mutta pivot pointin pysyessä paikallaan, kiertää objekti sen ympäri. Tämä voidaan toteuttaa myös kaikilla akseleilla. Jos pallo kuvitellaan narun päässä ja kiepautetaan palloa, se kiertää täydellisen ympyrän pisteen ympäri josta pidät narua kiinni. Rotation toimii vastaavalla tavalla. (Zeman 2015, 9.)

- Scale on muutos, joka muuttaa objektin komponenttien välimatkaa toisiinsa. Toisin sanoen tämä tarkoittaa, että sillä voidaan muuttaa objektin kokoa kaikilla akseleilla. (Zeman 2015, 11.)

Useimmat 3D-ohjelmistot sisältävät Object Moden, jossa edellä mainitut muuntaumiskeinot muuttavat koko objektin muotoa ja tilaa. Tämän lisäksi ohjelmistot sisältävät

Edit Moden, joka sisältää lukuisia työkaluja ja on tarkoitettu yksittäisten verteksien, reunojen tai pintojen muokkaamiseen.

Translate, rotation ja scale löytyvät myös näistä työkaluista, mutta näiden lisäksi erittäin yleinen työkalu on extrude, joka luo objektiin lisää geometriaa venyttämällä valittuja komponentteja. Kuvassa 5 havainnollistetaan yksittäisten komponenttien muokkaamista edit modessa. Vasemmalla puolella on kahdeksan verteksinen ja reunainen ympyrä, johon on seuraavassa vaiheessa luotu lisää geometriaa käyttämällä extrude-työkalua. Viimeisessä vaiheessa on äskeisessä luodut uudet verteksit pienennetty käyttäen Scale-työkalua.



Kuva 5. Extrude-, ja scale-työkalun havainnollistamista.

2.3 Ohjelmisto

3D-mallinnukseen on tarjolla useita eri ohjelmistoja, ja riippuu aivan käyttäjän mieltymyksestä ja käyttötarkoituksesta, minkä ohjelmiston valitsee. Pixologicin kehittämä ZBrush on erityisesti orgaanisiin hahmoihin suuntautuneiden taitelijoiden suosima, sen erinomaisten 3D-sculptaustyökalujen myötä. Autodeskin valmistama AutoCAD on laajalti käytetty tekniikan aloilla. Samaiselta yritykseltä ovat myös 3Ds Max ja Maya, jotka ovat suosittuja videoefekteissä, sekä peli- ja animointialalla. Blender sen sijaan tarjoaa avoimeen lähdekoodiin perustuvan 3D-grafiikan yleispaketin ilmaiseksi. (Creger 2015.) Se on erityisesti indiekäyttäjien suosiossa. Tässä työssä tullaan myös

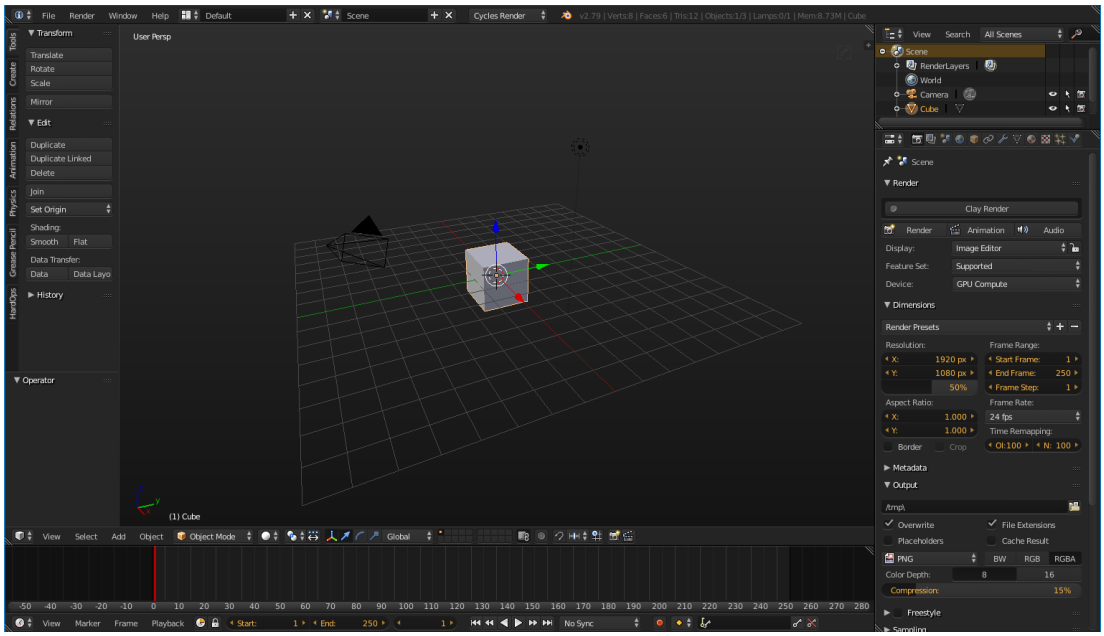
käyttämään Blenderiä, koska se on suunniteltu toimimaan yhdessä Unity-pelimoottorin kanssa, sekä lisäksi se on itselleni tutuin 3D-ympäristö entuudestaan.

2.3.1 Blender-ohjelmisto

Blender on erinomainen alusta aloittaville 3D-taiteilijoille. Se sisältää muun muassa mallintamisen, sculptauksen, riggauksen, animoinnin, simuloinnin, renderöintimoottorin, kuvankäsittelyn, videoeditoinnin sekä oman pelimoottorin pelien luomiseen. Siitä huolimatta, että Blender on tarkoitettu enemmän viihdealan käyttöön, kuin tekniikan aloille, se käyttää hieman poikkeuksellisesti Z-ylös koordinaattimenetelmää. Ohjelma perustuu GNU GPL (General Public License) -lisenssiin, ja näin ollen se on täysin ilmainen kaikille, myös kaupalliseen käyttöön. Se on saatavilla Linuxille, macOS:lle ja Windowsille. (Blender Foundation www-sivut 2018)

Blenderin perusti aikoinaan hollantilainen Ton Roosendaal, joka kehitti ensimmäisen version Blenderistä NeoGeo-animaatiostudionsa käyttöön vuonna 1988. Ohjelmiston kehitystä jatkettiin, ja vuonna 2002 hän perusti voittoa tavoittelemattoman Blender Foundation – säätiön, jonka tarkoituksena on tukea ohjelmiston kehittämistä lahjoitusten ja sponsorien avulla. (Blender Foundation www-sivut 2018)

Blender saa aloittelevilta käyttäjiltä paljon negatiivista palautetta sen käyttöliittymästään, ja usein se koetaan erittäin hitaaksi käyttää. Toisin kuin useimmat ohjelmat, Blender ei auta käyttäjää pääsemään alkuun juuri mitenkään, kun sen ensimmäisen kerran avaa. (Kuva 6.) Tämä johtuu siitä, että se nojaa vahvasti pikanäppäimiin ja niiden yhdistelmiin, jotka on opeteltava. Toisaalta kun oppimisprosessi etenee, siitä tulee erittäin nopea, ja tehokas 3D-työkalu käyttäjälle.



Kuva 6. Blenderin käyttöliittymän aloitusnäkymä (versio 2.79a).

Sen edullisuuden vuoksi, Blender on kerännyt suuren yhteisön internetissä, missä moni harrastaja on luonut opetusvideoita ja ilmaisia sekä maksullisia lisäosia ohjelmaan. Usein parhaimmat jopa päätyvät Blenderiin sisäänrakennetuksi ohjelman päivittyessä uuteen versioon. Näin ollen ohjelma on kehittynyt huimaa vauhtia viimeisten vuosien aikana, ja siitä on tullut todellinen kilpailija maksullisille 3D-ohjelmille. Se on nykyään eniten google-hakuja kerännyt 3D-ohjelmisto.

Itse pidän Blenderin parhaimpina puolina sen nopeaa työnkulkua, ja kun pikanäppäimet muistaa ulkomuistista, sillä mallintaminen on erittäin jouhevaa. 3D-mallinnukseen tarkoitetut ohjelmat ovat yleensä melko raskaita prosessorille, mutta Blender on huomattavasti kevyempi, kuin mikään muu ohjelmisto jota olen kokeillut. Toisin kuin muut vastaavat ohjelmat, se ei vaadi asennusta tietokoneelle toimiakseen. Näin ollen sitä voidaan käyttää jopa suoraan muistitikulta.

2.4 Tiedostomuodot

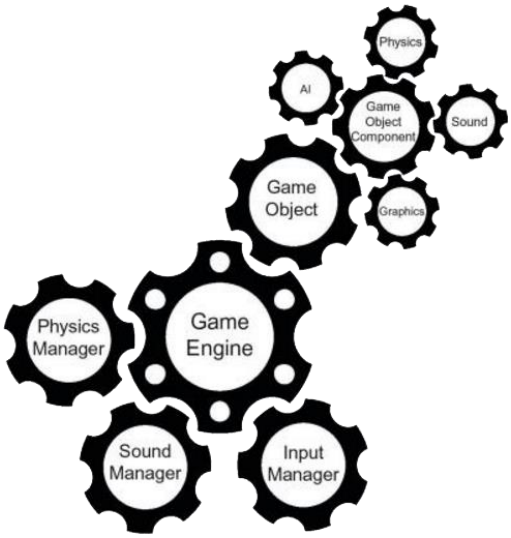
Ongelma 3D-tiedostomuodoissa on niiden runsas määrä. AutoCAD luo dwg-tiedostoja, Blender luo blend-tiedostoja, Maya luo ma- tai mb-tiedostoja jne. Siksi on luotu

neutraaleja tiedostoformaatteja, jotka mahdollistavat niiden kuljettamisen eri 3D-ohjelmistojen välillä. Neutraalien tiedostomuotojen päätarkoitus on sisältää tietoa 3D-mallista. Yleisimpiä formaatteja ovat STL, OBJ, FBX ja COLLADA. Eri formaatit ovat laajalti käytettyjä 3D-tulostuksessa, videopeleissä, elokuvissa ja arkkitehtuurissa. Jokaisella alalla on omat suosikkinsa tiedostoformaattien suhteen. Esimerkiksi 3D-tulostuksessa paljon käytetty STL-tiedostomuoto sisältää vain objektin geometrian, kun pelialalla suosittu FBX voi sisältää muun muassa geometrian, materiaalin ja animaation. (Chakravorty 2017.)

3 PELIMOOTTORI

3.1 Pelimoottori käsitteenä

Pelimoottori on ohjelmistokehys, joka on suunniteltu videopelien luontiin ja kehittämiseen. Ymmärtääkseen kuinka pelimoottori toimii, verrataan sitä auton tuotantoon. Aivan kuten auton moottorikin, pelimoottori on pääratas, joka pyörittää kaikkea sen ympärillä (Kuva 7). Se toimii kuin autonvalmistustehdas, joka kokoaa auton jo valmiiksi tehdyistä osista. Pelimoottori vastaavasti kokoaa muun muassa skriptit, äänet, grafiikat ja muodostaa pelin. (Crosby 2011; Ward 2008; De Byl 2012, 17.)



Kuva 7. Pelimoottorin osat (De Byl 2012, 18).

Mietitään esimerkkinä pelin fysiikkaa. Jos pelimaailmaan luodaan pallo-objekti, tämä saadaan reagoimaan painovoiman kanssa yhdellä napin painalluksella. Ilman pelimoottoria vastaava operaatio tarvitsisi usean rivin koodin kirjoittamista. Valmis pelimoottori nopeuttaa huomattavasti pelisuunnittelijoita koodaamaan pelin valmiiksi, koska niihin on ohjelmoitu valmiiksi monimutkaiset matemaattiset algoritmit sekä graafinen käyttöliittymä. (Ward 2008.)

Toisin sanoen pelimoottori tekee kaiken raskaan työn pelinkehittäjän sijasta. Tämä mahdollistaa itse pelinkehittäjän keskittymään pelin yksityiskohtiin ja luomaan pelistä ainutlaatuisemman. Tästä syystä pelimoottori on erinomainen vaihtoehto aloitteleville ja itsenäisille pelinkehittäjille. (Ward 2008; Unity Technologies www-sivut.)

3.2 Mikä on Unity?

Unity on ammattitason pelimoottori, jota käytetään videopelien luontiin useille pelialustoille (Hocking 2015, 4). Kyseistä pelimoottoria on tarjolla eri lisenssityypeillä. Pelkistetty Unity, lisänimeltään Free on ilmaiseksi ladattavissa oleva monialustainen pelinkehitysympäristö, josta on karsittu joitakin ominaisuuksia maksulliseen Pro-versioon verrattuna. Free-versiolla tehtyjä pelejä voi jakaa vapaasti. Tosin kehittäjän tienatessa yli 100 000:n dollarin vuosiansiot tulee käyttäjän ostaa Unityn Pro-version lisenssi. Lisenssi on hankittava jokaisen työryhmän kehittäjälle erikseen. Maksullisen version etuja on muun muassa kääntäjä useammalle alustalle. (Unity Technologies www-sivut 2018.)

Ohjelmisto tarjoaa helpon lähestymisen ohjelmointiin tukemalla kolmea eri ohjelmointikieltä. Microsoft-yhtiön kehittämä C# on selvästi käyttäjien eniten suosima kieli. Muita ovat JavaScriptiä mukaileva UnityScript sekä hieman harvinaisempi, Pythonia mukaileva Boo. (Unity Technologies www-sivut 2018.)

Unity on todellisuudessa kahden komponentin ohjelmistopaketti, vaikka se yleensä käsitetään vain pelimoottorina. Paketin toinen komponentti on editori. (Thorn 2013, 9.) Unityn mukana tulee myös ohjelmointiympäristö MonoDevelop.

3.2.1 Unityn historia

Unity on vuonna 2005 julkaistu pelinkehitysympäristö. Ensimmäinen versio näki päivänvalonsa Applen isännöimässä konferenssissa, ja se olikin alun perin suunniteltu vain Applen OS X -käyttöjärjestelmälle. Sen on kehittänyt alun perin kolmen tanskalaisen henkilön yritys Unity Technologies sen jälkeen, kun heidän ensimmäinen pelinsä GooBall ei saavuttanut haluttua menestystä. Unity Technologies kuitenkin ymmärsi pelimoottorin merkityksen, ja sen sijaan, että he olisivat aloittaneet uuden peliprojektin, he keskittyivät pelimoottorin kehitykseen. Pelinkehitysympäristö on sen jälkeen kääntynyt yli kahdellekymmenelle alustalle (Kuva 8). (Takahashi 2014; Unity Technologies www-sivut 2018.)

Kuva 8. Unityn tukemat alustat (Unity Technologies www-sivut 2018).

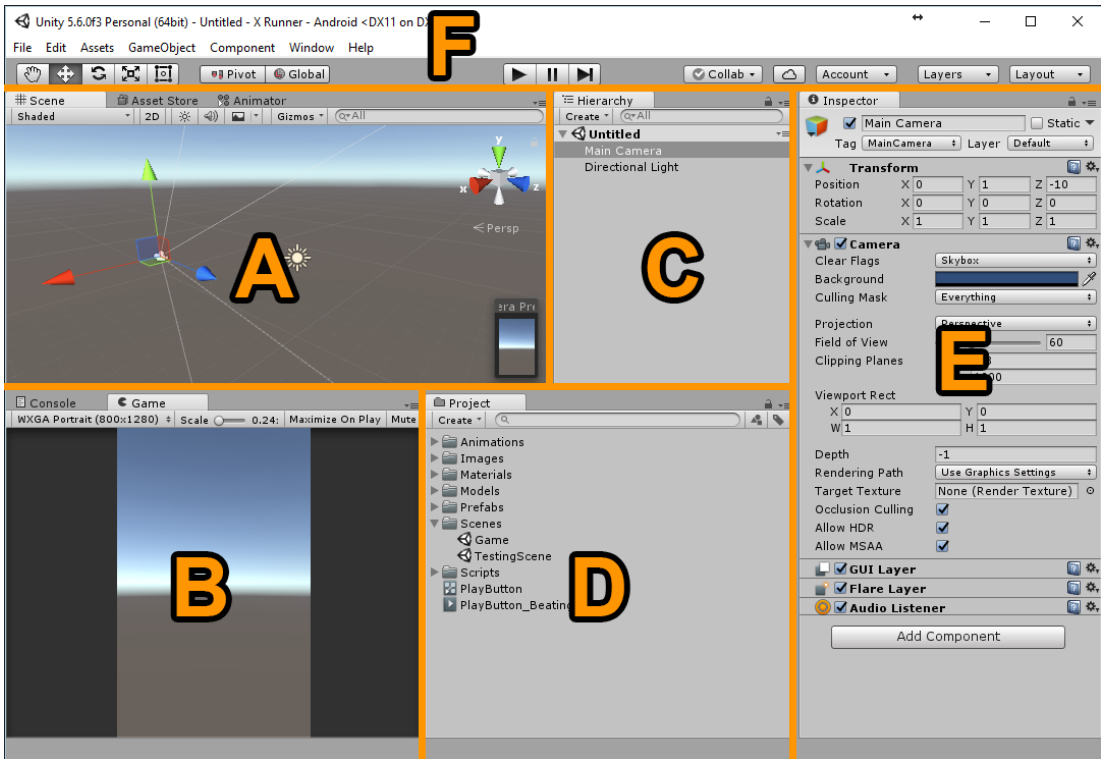
3.2.2 Unity-pelimoottori

Unity-pelimoottori on etenkin indiepelinkehittäjien suosiossa. Valttikorttina Unitylla on tähän erityisesti pelimoottorin monialustaisuus. Monialustaisuudella tarkoitetaan sitä, että kehittäjä voi yhden Unity-projektin pohjalta kääntää projektin mille tahansa tunnetulle pelaamiseen käytetylle alustalle. Lisäksi indiekehittäjiä houkuttelee sen edullisuus, sekä sen tuki tunnettuja ja yleisesti käytettyjä tiedostomuotoja kohtaan. Kehitysalustaan voi ladata Asset Store -sivustolta ilmaisia ja maksullisia lisäosia, malleja, tekstuureja, skriptejä, animaatioita ja ääniä. Tämän lisäksi ohjelmisto on muun muassa suunniteltu toimimaan yhdessä 3D-mallinnusohjelmisto Blenderin kanssa. Unityn kehitysympäristö itsessään ei ole grafiikoiden, äänien, musiikkien luontiin. (Thorn 2013, 11; McKleinfeld 2012; Menard & Wagstaff 2015, 3.)

3.2.3 Käyttöliittymä

Aloittaessa uuden projektin Unitylla, editori on ensimmäinen näkymä, johon pelinkehittäjä törmää. Se on suunniteltu visuaaliseksi ja helppokäyttöiseksi. Editori on ohjelmiston konkreettisin osa, jossa pelinkehittäjä viettää suurimman osan ajastaan. Sitä käytetään kehitysprosessin ajan vain kehittäjien toimesta pelin muokkaamiseen. Editointialusta kattaa kaikki tarvittavat työkalut pelin muokkaamiseen. Itse editori ei näy lopullisessa pelissä mitenkään. (Thorn 2013, 9-10.)

Unityn graafista käyttöliittymää voidaan muokata käyttäjän halujen mukaan, vetämällä eri paneeleita eri kohtiin. Käydään läpi pelinkehityksen kannalta kuusi tärkeää paneelia. (Kuva 9.) Scene- (kohta A), Game- (kohta B), Hierarchy- (kohta C), Project- (kohta D), Inspector-paneeli (kohta E) sekä työkalut (kohta F). Näitä käytetään kaikkien peliobjektien hallinnointiin. Peliobjekteilla tarkoitetaan muun muassa esineitä, valaistuksia, ääniä ja pelihahmoja.

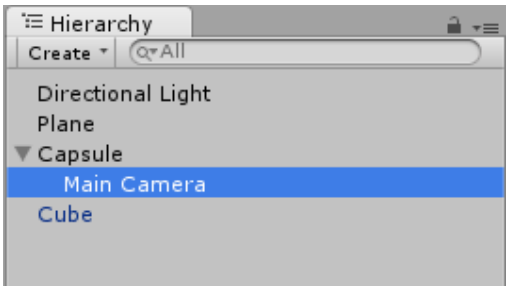


Kuva 9. Unityn (versio 5.6.0f3) käyttöliittymä osioituna.

Työkalurivistöltä löytyy tartunta-, siirto-, kierto-, skaalauspainikkeet objekteille. Tärkeimpänä kuitenkin pelin testaamisen kannalta paneelista löytyy Play-nappi, jota klikkaamalla ohjelmisto avaa Game-ikkunan pelitilaan, jossa pelin testaaminen tapahtuu. (Lee 2015, 105.)

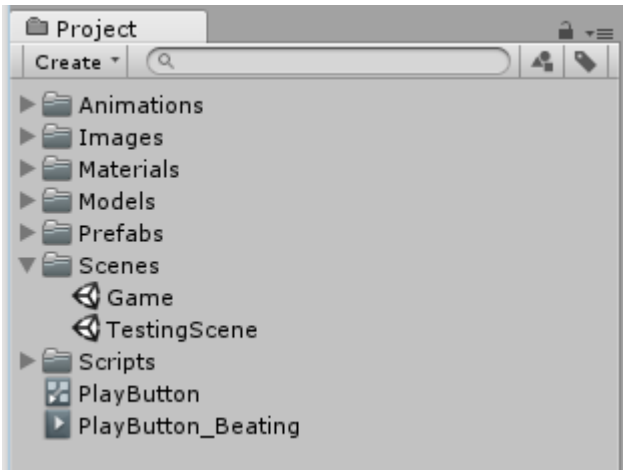
Peli rakentuu kentistä, jotka esiintyvät Scene-paneelissa. Näkymässä voi esiintyä vain yksi kenttä kerrallaan. Scene- sekä Game-paneelissa on kaikki pelin sisältämät objektit näkyvissä, mutta vain ensiksi mainitussa kameraa voi liikutella hiirellä ja näppäimistöllä vapaasti kaikkiin suuntiin, jälkimmäisessä sen sijaan näkyy, miltä peli todellisuudessa tulee näyttämään kentän pääkameran perspektiivistä.

Hierarchy-paneelissa on kaikki scenen objektit listan muodossa. Objektia klikkaamalla muuttuu kyseinen objekti aktiiviseksi. Näin päästään käsiksi tietyn objektin asetuksiin ja komponentteihin. Objektit ilmaistaan hierarkisessa järjestyksessä, ja niille voidaan määrittää children- eli lapsiobjekteja. (Lee 2014, 105.) Tämä tarkoittaa käytännössä sitä, että lapset seuraavat, kun hierarkiassa korkeammalla oleva parent-objekti liikkuu. Kuvassa 10 pelikamera on siirretty pelihahmona toimivan kapselin lapseksi, näin kamera seuraa pelihahmoa, kun se liikkuu tai rotatoi.



Kuva 10. Kamera pelihahmona toimivan kapselin lapsena.

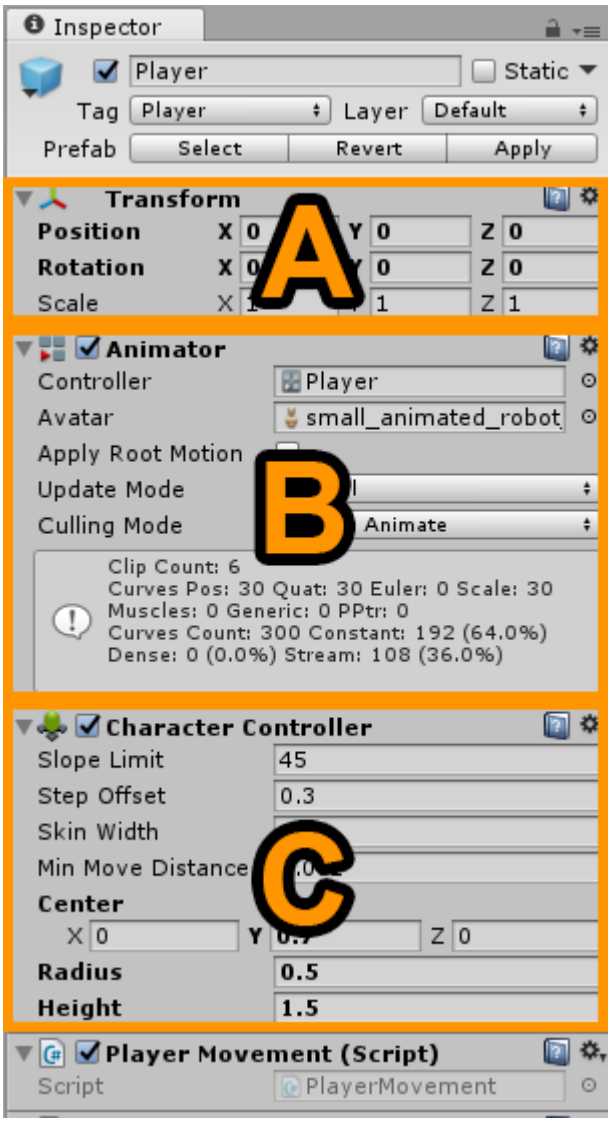
Project-paneeli esittää kaikki projektin sisältämät kansiot ja tiedostot. Tämän paneelin päätarkoitus on lisätä helposti pelimaailmaan valmiiksi määriteltyjä objekteja. (Lee 2015, 105.) (Kuva 11.)



Kuva 11. Project-paneelissa on projektin kaikki kansiot esillä.

Inspector-paneeli (Kuva 12.) näyttää aina aktiivisena olevan objektin komponentit ja niiden arvot. (Lee 2015, 105). Transform-komponentti (kohta A) kertoo peliobjektin koordinaatit 3D-pelimaailmassa, ja se on ainoa pakollinen komponentti jokaiselle objektille. Muut tarkasteltavat ja seuraavassa luvussa keskeiset komponentit ovat Animator (kohta B), jolla voidaan sijoittaa animaatioita peliobjektiin. Animator-komponentti vaatii viittauksen Animator Controlleriin, joka on Unity-pelinkehitysympäristön sisältämä assetti. Siihen lisätään hahmolle kuuluvat animaatioleikkeet, jonka jälkeen niistä voidaan muodostaa tarvittava kaavio niiden toimiakseen halutulla tavalla. Character Controller (kohta C) on komponentti, joka mahdollistaa pelihahmon liikuttamisen käyttäjän antamilla syötteillä, ja määrittää objektin törmäysrajat. Muita komponentteja voi olla esimerkiksi skripti.

Kun Unityyn luodaan C#-skripti, sisältää se aina valmiiksi Start- ja Update-funktiot. Kaikki Start-funktiossa olevat komennot ajetaan läpi heti kun skripti suoritetaan. Update-funktio sen sijaan ajetaan läpi jokaisella framella, eli jos peli pyörii 30 FPS:llä, ajetaan funktio 30 kertaa sekunnissa. (Unity Technologies www-sivut 2018.)



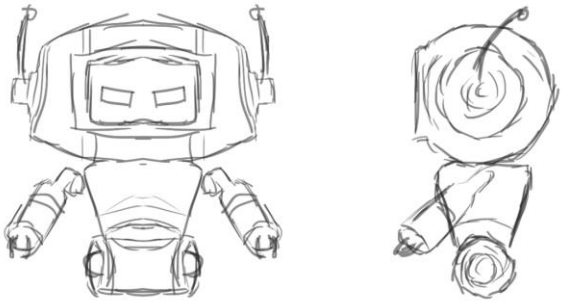
Kuva 12. Inspector-paneeli osioituna.

Kuvassa 12 on vasemmassa yläkulmassa teksti ”Tag”. Tagit ovat viitesanoja, joita voidaan liittää yhteen tai useampaan objektiin. Esimerkiksi pelaajan voi määrittää ”Player”-tagiksi, viholliset ”Enemy”-tagiksi ja pelissä kerättävät kolikot ”Collectable” -tagiksi. Tagit helpottavat objektien tunnistamista skripteissä. Näin vältytään peliobjektien manuaalisesta lisäämisestä skripteihin. Unityn C#-skripteissä voidaan käyttää GameObject.FindWithTag(”Player”) komentoa löytääkseen kaikki objektit, jotka sisältävät tässä tapauksessa ”Player” tagin.

4 PELIHAHMON LUONTI MOBIILIPELIIN

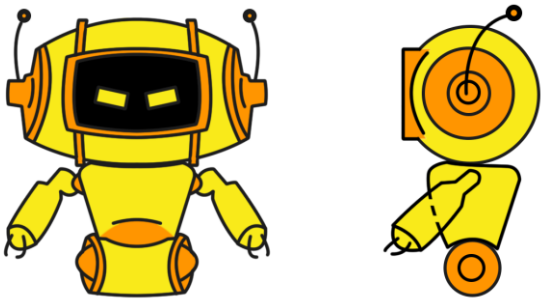
4.1 Konseptointi

Konseptoinnin tavoitteena on tuottaa visuaalinen esitys suunnitelmasta, ideasta tai tunnelmasta ennen kuin tuotteesta tehdään lopullinen. (Pickthall 2012.) Tässä työssä mallinnettava mobiilipelin hahmo on robotti. Koska kyseinen konsepti muutetaan 3D-malliksi, on se hyvä piirtää edestä sekä sivusta oikeassa mittasuhteessa toisiinsa nähden. Aloitan konseptoinnin tekemällä raakoja hahmotelmia, kunnes olen konseptiin tyytyväinen. (Kuva 13)



Kuva 13. Lopullinen hahmotelma.

Seuraavaksi piirrän hahmotelman puhtaaksi Adobe Photoshopilla, selkeyttääkseni hahmon yleisilmettä. Tässä vaiheessa on myös hyvä miettiä hahmon värimaailmaa. Kuvassa 14 lopullinen konsepti, joka siirretään seuraavaksi Blenderiin ja mallinnetaan kolmiulotteiseksi konseptikuvan päälle.



Kuva 14. Puhtaaksi piirretty konsepti.

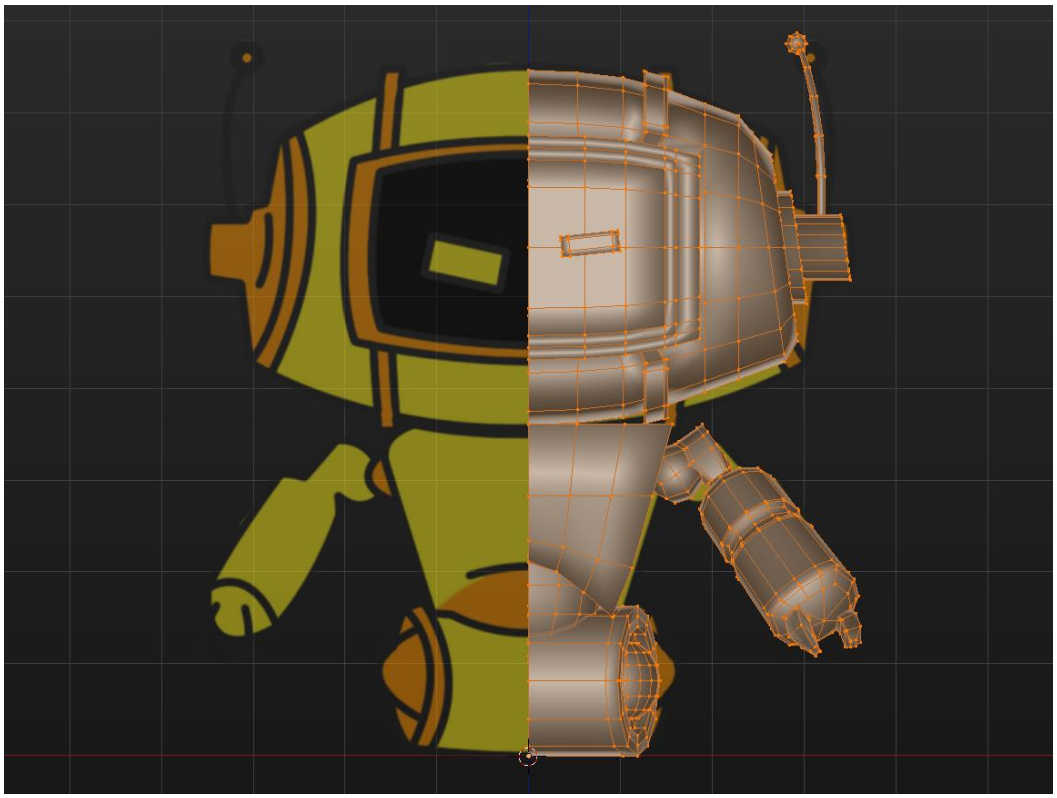
4.2 Mallintaminen

Blenderissä on Background Images -ominaisuus, joka mahdollistaa 2D-kuvien lisäämisen kolmiulotteiseen avaruuteen. Kun kuvat on lisätty ohjelmaan, saa jokaisen kuvan valita näkymään haluamastaan kamerakulmasta. Tässä tapauksessa valitsen etuprofiilin näkymään etukamerasta, joka on linjassa y-akselin (syvyysakselin) kanssa. Toisen kuvan laitan näkymään oikealta puolelta, joka on linjassa x-akselin kanssa.

3D-ohjelmat sisältävät valmiita perusmuotoja kuten esimerkiksi, laatikon, pallon, sylinterin, kartion ja neliön muotoisen tason. Kaikista näistä muodoista päästään objektin geometriaa muuttamalla haluttuun lopputulokseen, mutta on helpompaa aloittaa muodosta, joka on lähimpänä tavoiteltua lopputulosta. Koska robotti on symmetrinen x-akselilla, käytän apunani Blenderin mirror-työkalua. Tämä tarkoittaa sitä, että voin mallintaa vain puolet hahmosta ja ohjelma peilaa toisen puoliskon.

Aloitan mallintamisen robotin päästä, johon valitsin sylinterin. Saavutin haluamani topologian käyttämällä pääasiassa extrude, scale ja rotate -työkaluja. Sitten poistin sylinterin etuosasta verteksejä, jättäen päässä olevalle näytön kohdalle aukon. Lisäsin aukkoon laatikon, jonka kokoa muutin eri akselilla siihen sopivaksi käyttäen scale-työkalua. Näytön kehykset toteutettiin myös extrude ja scale -työkaluja käyttäen. Lopuksi lisäsin laatikon myös silmäksi ja kiersin sitä rotate-työkalulla antamaan hahmolle ilmettä. Kädet ja jalat luotiin myös käyttäen perusmuodoista sylinteriä lähtökohtana. Keskiruumiin mallinnuksen aloitin puolestaan laatikosta.

Kuvassa 15 on näkymä Blenderistä, jossa aikaisemmin tehty konseptikuva näkyy vasemmalla puolella. Oikealla puolella on konseptin päälle mallinnettu lopullinen versio hahmon topologiasta ennen mirror-työkalun käyttämistä. Lopullisen hahmon nelikulmaisten poligonien määräksi tuli 2014. Pelimoottorin muuttaessa poligonit kolmioiksi, tulee lopullinen lukema olemaan 4196.



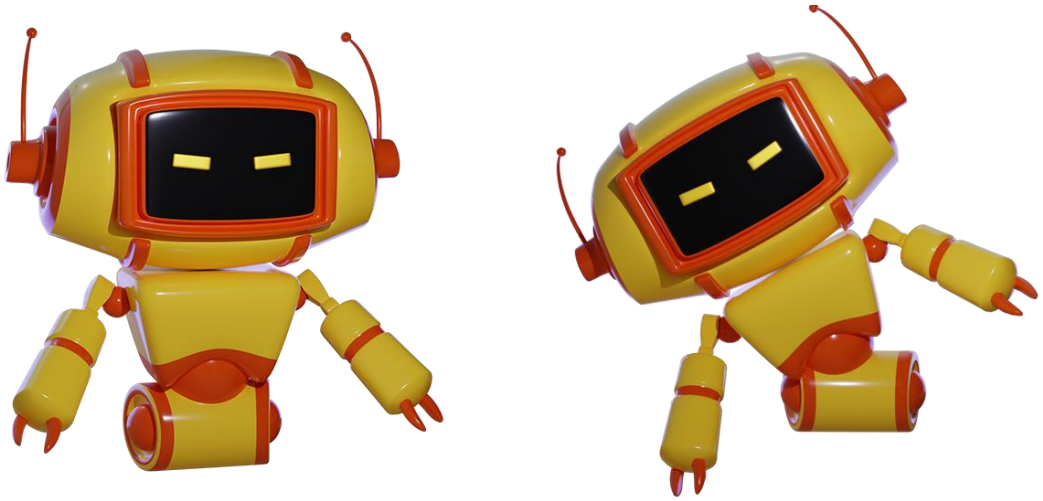
Kuva 15. Robotin topologia.

Kun 3D-mallin geometria on valmis, voidaan pinnoille määritellä materiaalit. Jokaiselle poligonille voidaan halutessa määrittää oma materiaali. Vaihtoehtoinen ja enemmän mahdollisuuksia tarjoava tekniikka on UV-kartoitus. UV-kartoituksella saadaan piirrettyä 2D-kartta hahmon poligoneista. Joka voidaan kiedota 3D-mallin ympärille. U- ja V-kirjaimet ovat nimiä 2D-akseleille, XYZ-kirjaimien ollessa jo varattuja 3D objekteille. (Solanki 2015.) Tällä tekniikalla yhdessä poligonissa voi olla niin monta eri materiaalia kuin kaksiulotteisen tekstuurin resoluutio mahdollistaa. Tässä työssä käytetään kuitenkin Blenderin omia shadereita, jotka asetetaan konseptikuvan mukaan halutuille polygoneille.

4.3 Animointi

Animaatiot eivät sinänsä vaikuta itse peliin, mutta luovat käyttäjälle visuaalisesti miellyttävämmän pelikokemuksen. Monimutkaisemmat hahmot tulee rigata, jotta ne voidaan animoida. Tässä työssä käytettävä robotti kuitenkin animoidaan vaan muuttamalla objektien rotaatiota.

Ensiksi keskiruumis tulee asettaa parent-objektiksi käsille ja päälle, näin ollen keskiruumiin kallistuessa myös kädet ja pää seuraavat. Jalat pidetään itsenäisenä objektina. Kuvassa 16 on renderöity kuva robotista, jossa oikean puoleisesta kuvasta voidaan huomata miten kädet ja pää seuraavat, kun keskiruumista käännetään.



Kuva 16. Blenderissä renderöity kuva havainnollistamaan parent-objektin käyttäytymistä.

Robotti animoidaan hyppäämään, liukumaan, kuolemaan, menemään eteenpäin ”kävelyanimaatiolla”, sekä silloin kun mitään näistä ei tapahdu, tarvitsee sille animoida idle-animaatio. Käytännössä tällä tavoitellaan sitä, että hahmo ei ole täysin paikallaan ennen kuin itse peli alkaa. Idle-animaatiota tullaan hyödyntämään pelin alkuvalikossa.

Koska robotin jalat, pää, keskiruumis ja molemmat kädet ovat omia objekteja ja jokaisella objektilla on oma pivot point, voidaan jokainen näistä animoida erikseen. Animointi tapahtuu ”keyframejen” asettelulla Blenderin aikajanalla. Keyframe on keino tallentaa tietoa objektista, kuten esimerkiksi rotaatio tai koko. Kun asetetaan kaksi eri arvoja sisältävää keyframea eri kohtiin aikajanaa, animaatio syntyy keyframejen väliin. Esimerkkinä käytän hahmon ”kävelyanimaatiota”.

Robotin jalkojen virkaa toimittaa sylinteri. Sylinterin pivot point asetetaan tässä tapauksessa keskelle sylinterin geometriaa, jotta se pyörii renkaan lailla eteenpäin. Sylinterin rotaatio lukitaan nollaan asteeseen frameen 15. Frameen 29 asetetaan rotaatio

X-akselilla 359:n asteeseen. Näin ollen sylinteri pyörii täyden kierroksen 15:sta framen aikana. FPS:n ollessa 24, kestää yksi pyörähdys hieman yli puolisekuntia. Yksi pyörähdys on tässä tilanteessa tarpeeksi, koska animaatio voidaan laittaa pelimoottorissa alkamaan alusta, kun viimeinen frame animaatiosta on toistettu. Tätä kutsutaan animaation looppaamiseksi.

4.4 Unity-projektin aloittaminen

Projektia aloittaessa määritetään se 3D-projektiksi sekä annetaan sille nimi ja polku, johon se tallennetaan. Tyhjä Unity-scene sisältää aloittaessa vain pelin kannalta tärkeimmät objektit, kameran ja valonlähteen. Ilman kumpaakaan näitä objekteja, pelaaja ei pysty näkemään pelissä mitään. Tyhjä scene on hyvä tallentaa heti ennen pelin kehittämistä, jotta se voidaan nimetä.

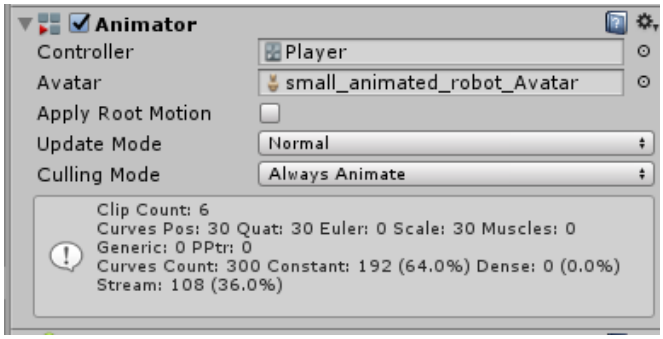
4.4.1 Hahmon tuonti

Blender ja Unity ovat suunniteltu toimimaan yhdessä, joten Unity osaa lukea Blenderin luomasta blend-tiedostosta objektien geometrian, materiaalit ja animaatiot. Pelihahmoa ei näin ollen tarvitse muuttaa mihinkään 3D-objektien siirtämiseen tarkoitettuun neutraaliin tiedostomuotoon. Blenderissä luodusta scenestä tulee kuitenkin poistaa kaikki ylimääräinen, joita voivat olla esimerkiksi kamera ja valot.

Mallinnetun pelihahmon blend-tiedosto voidaan lisätä suoraan projektin Assets-kansioon, mistä Unity löytää automaattisesti sen. Nyt pelihahmo on valmis vedettäväksi project-paneelista scene-paneeliin. Jolloin se saadaan näkyviin Unityn scene-ikkunassa. (Kuva 17.)

Kuva 17. Robotti Unityn Scenessä.

Pelihahmon tagiksi on hyvä heti muuttaa Player, jota voidaan hyödyntää myöhemmin skripteissä, esimerkiksi sen törmätessä esteeseen. Hahmolla ei ole tässä vaiheessa muita komponentteja kuin Transform. Hahmolle lisätään komponentti Animator (Kuva 18.), ja hallitakseen pelihahmon animaatioita tulee projektiin luoda Animator Controller project-paneelista, joka nimetään ”Player”:ksi ja liitetään pelihahmon Animator-komponentin controller-kenttään.

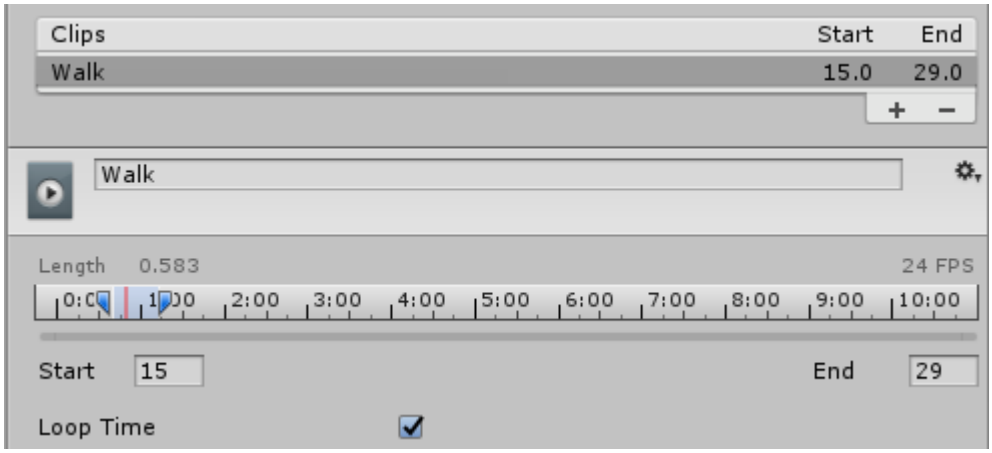


Kuva 18. Animator komponentti.

4.4.2 Animointien tuonti

Aikaisemmin projektiin tuodun blend-tiedoston sisältämät animaatiot täytyy leikata pelimoottorissa. Tämä voidaan suorittaa valitsemalla project-paneelista kyseinen tiedosto, jolloin inspector-paneeliin avautuu Blenderissä luotu animaatio. Ennen leikkaamista kaikki luodut animaatiot toistuvat peräkkäin toinen toisensa jälkeen.

Esimerkkinä aikaisemmin käytetty ”kävelyanimaatio” erotellaan luomalla siitä oma leike. Leike nimetään ”Walk”:ksi ja asetetaan se alkamaan framesta 15 ja loppumaan frameen 29, kuten se aikaisemmin oli animoitu. Välttääkseen sylinterin pyörähtämistä vain kerran, tulee rastittaa myös Loop Time. (Kuva 19.)



Kuva 19. Animaatioiden leikkaus tapahtuu inspector-paneelissa.

4.4.3 Hahmon liikuttaminen syötteillä

Jotta käyttäjä voi omilla syötteillään hallita pelihahmoa, tulee hahmolle antaa komponentti CharacterController, joka vastaa pelihahmon liikuttamisesta ja törmäyksistä. Tämän lisäksi hahmolle luodaan C#-skripti PlayerMovement (LIITE1).

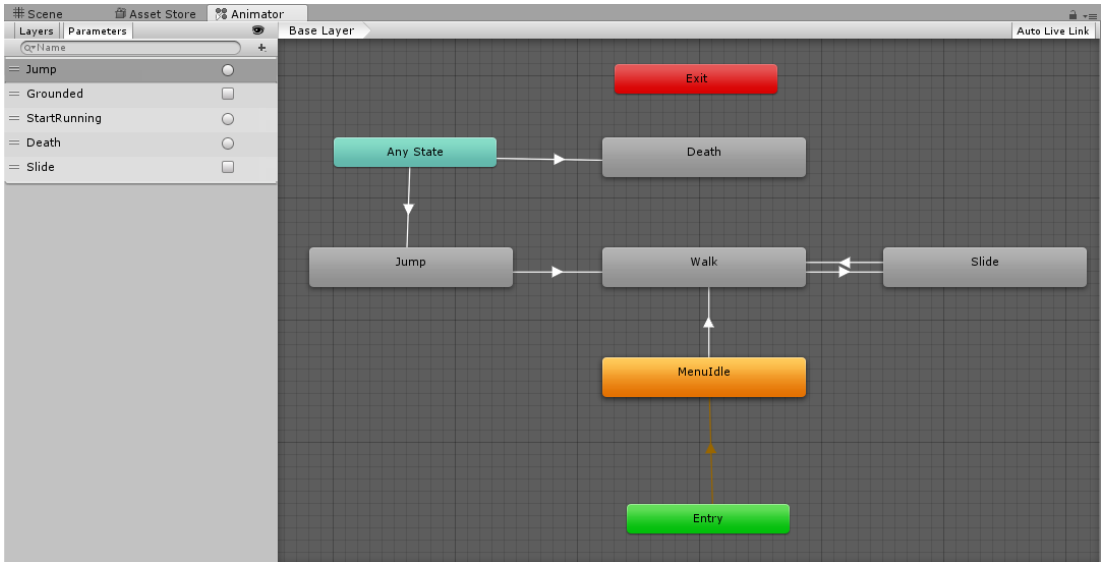
Hahmolle luodussa PlayerMovement-skriptissä luodaan muuttuja nimeltään controller, tyyppiä CharacterController. Start-funktiossa tämä muuttuja sijoitetaan komponenttiin CharacterController GetComponent -komennon avulla. Näin ollen ”controller”-muuttujalla voidaan hallita kyseistä komponenttia.

4.4.4 Animointien muuttaminen interaktiiviseksi

Animator Controller (Kuva 20.) hallitsee kaikkia aikaisemmin Blenderissä luotuja animaatioita, ja miten ne käyttäytyvät pelissä. Kaavio alkaa Entry-tilasta ja siirtyy heti

pelin avautuessa MenuIdle-animaatioon, joka on ensimmäinen animaatio, jonka pelaaja näkee. Oranssi väri MenuIdle-laatikossa kuvastaa, että se on asetettu oletustilaksi kaikista animaatioista. Nuolen suunta osoittaa, että siitä ei voida siirtyä minnekään muualle kuin Walk-animaatioon, joka tapahtuu heti kun käyttäjä napauttaa ruutua.

Kuvan 20 vasemmassa reunassa on nimettyjä parametreja. Ympyrä parametrin oikealla puolella indikoi sen olevan Trigger-parametri, joilla voidaan käynnistää prosesseja. Neliö puolestaan kuvastaa Boolean-parametria, jolla voi olla vain arvot tosi tai epätosi. StartRunning-parametri on liitetty MenuIdle ja Walk animaatioiden väliseen siirtymään.



Kuva 20. Animator-paneelissa Animator Controller-komponentin kaavio.

PlayerMovement-skripti sisältää boolean-muuttujan isRunning, jolle on asetettu oletusarvoksi epätosi. Unity tarkastaa jokaisella päivittämällään framella PlayerMovement-skriptistä Update-funktion onko pelaaja napauttanut näyttöä ja samalla ”triggeroinut” pelihahmon kävelyanimaatioon. Kuvassa 21 Update-funktio ei pääse etenemään, ennen kuin isRunning on muuttunut todeksi.

```
private void Update()
{
    if (!isRunning)
        return;
```

Kuva 21. Update-funktio ei etene ennen kuin isRunning on tosi.

Samoin kuin CharacterController-tyyppinen muuttuja ”controller” sijoitettiin aikaisemmin komponenttiin, tulee myös skriptiin luotu Animator-tyyppinen ”anim”-muuttuja sijoittaa Animator-komponenttiin.

Pelaajan napauttaessa näyttöä, suorittaa peli StartRunning-funktion (Kuva 22.), joka on määrätty suoritettavaksi pelinhallinnasta vastaavassa GameManager-skriptissä. StartRunning-funktio, muuttaa isRunning arvoksi tosi ja ajaa sen jälkeen Kuvassa 22 luodun Trigger-parametrin ”StartRunning”. Kun koodi ajaa triggerin, siirtyy Animator Controller ”MenuIdle”-tilasta ”Walk”-tilaan. Samaan aikaan Update-funktio pääsee etenemään aloitusvalikosta peliin.

```
public void StartRunning()
{
    isRunning = true;
    anim.SetTrigger ("StartRunning");
}
```

Kuva 22. StartRunning-funktio.

Kuvan 20 turkoosi Any State-laatikko on tapauksia varten, jossa kesken minkä tahansa animaation voidaan suorittaa toinen animaatio, joka on yhteydessä siihen. Tässä tapauksessa ohjattava pelihahmo voi hypätä tai kuolla kesken kaikkien animaatioiden. Kuvassa 20 oikealla oleva Slide liukumisanimaatio voidaan suorittaa vain kävelyanimaation aikana.

4.4.5 Mobiilikontrollit

Jotta pelihahmo saadaan interaktiiviseksi käyttäjän kanssa, tulee sen rekisteröidä käyttäjän antamat syötteet. Pelihahmolle luodaan komponentiksi C#-skripti MobileInputs (LIITE 2), johon luodaan julkinen staattinen instanssi. Näin siihen päästään käsiksi pelihahmon kontrolleista vastaavasta PlayerMovement-skriptissä.

PlayerMovement-skriptiin luodaan funktio MoveLane (Kuva 23.), joka määrittää pelaajan liikkumisen kolmella eri kaistalla. Funktio sisältää boolean-parametrin nimeltä goingRight

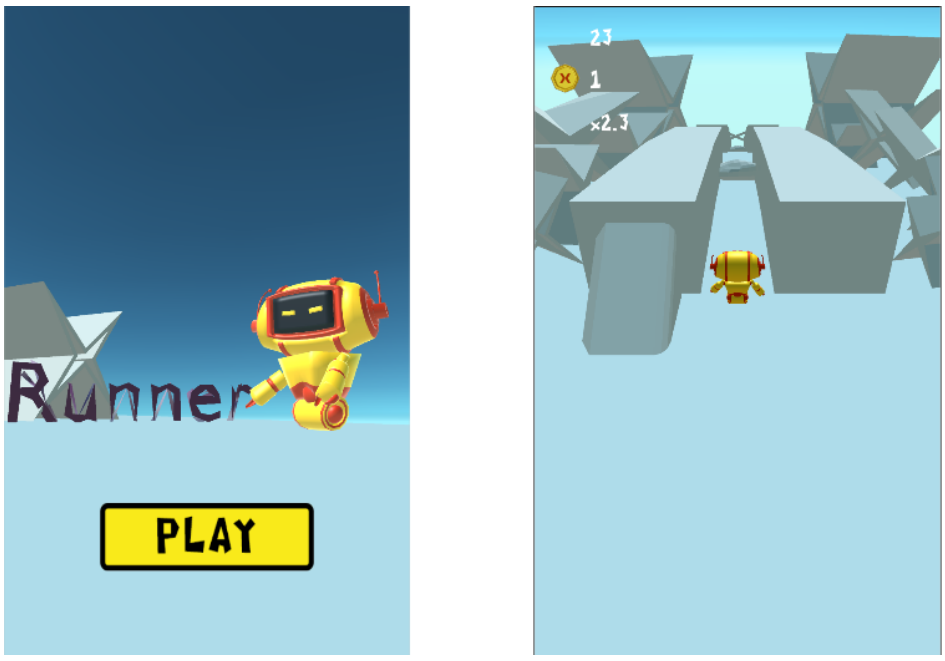
```
private void MoveLane(bool goingRight)
{
    if (!goingRight) {
        desiredLane--;
        if (desiredLane == -1)
            desiredLane = 0;
    }
    else
    {
        desiredLane++;
        if (desiredLane == 3)
            desiredLane = 2;
    }
}
```

Kuva 23. MoveLane-funktio.

PlayerMovement-skriptin Update-funktiossa voidaan tarkastella käyttäjän syöte. Kun käyttäjä pyyhkäisee oikealle, muuttuu parametrin arvoksi tosi, tällöin MoveLane suorittaa jälkimmäisen ehdon ja hahmo siirtyy oikealle. (Kuva 24.)

```
if (MobileInput.Instance.SwipeLeft)
    MoveLane (false);
if (MobileInput.Instance.SwipeRight)
    MoveLane (true);
```

Kuva 24. Käyttäjän syötteisiin reagoivat ehtolauseet.



Kuva 25. Ruudunkaappaus hahmosta pelin alkuvalikossa ja itse pelissä.

5 YHTEENVETO

Opinnäytetyön tarkoituksena oli selvittää, mitä vaiheita pelihahmon valmistaminen alusta loppuun asti pitää sisällään. Samalla perehtyen 3D-mallintamisen ja pelinmoottorin perusteisiin. Voidaan todeta, että pelihahmon voi valmistaa usealla eri tavalla. Riippuukin pitkälti sen käyttötarkoituksesta, millaista lähestymistapaa siihen tulee käyttää, ja mille alustalle lopullinen hahmo on tarkoitus sisällyttää.

Olen tyytyväinen työssä luodun robotin visuaaliseen ilmeeseen, sekä sen sulavasta reagoinnista animaatioiden kera käyttäjän syötteisiin. Sen sijaan työn aihepiirin rajauksessa ei onnistuttu täysin. Tämä johti siihen, että jouduin useasti puntaroimaan mitä asioita tulisi sisällyttää opinnäytetyöhön. Koen, että varsinkin robotin mallintamista koskeva osio jäi turhan suppeaksi.

Projektia tehdessä opin paljon uutta. Haastavinta, mutta myös samalla parasta ja opettavinta antia itselleni oli UnityEnginen C#-skriptien kirjoittaminen, joihin en ole aikaisemmin ehtinyt tutustumaan. 3D-mallintamista ja hieman animointia olen tehnyt jo pari vuotta, mutta oli erittäin antoisaa oppia hyödyntämään niitä pelimaailmassa. Projektia tehtäessä oli myös ilo havaita, kuinka bisnesajattelumallit ovat ohjelmistonkehittäjillä muuttumassa. Nykypäivänä on mahdollista käyttää erittäin monipuolisia ohjelmistoja ilmaiseksi, ja jopa mahdollisesti tienata rahaa niiden avulla. Siinä missä maksullisen lisenssin omaavat ohjelmat tarjoavat yleensä tukipalvelua ongelmatilanteisiin, nämä ilmaiseksi käytettävissä olevat ohjelmistot ovat luoneet huiman kommuunin internetin keskustelupalstoilla joissa käyttäjät jakavat tietoa toisilleen ilmaiseksi.

Loppuun voidaan todeta, että harva varmaan käsittää miten pitkälle pelimoottoreiden ansiosta nykyään pääsee pelkillä ohjelmoinnin perustaidoilla. Unity ja Blender toimivat saumattomasti yhteen ja yhdessä ne ovat hauska ja helppo tie päästä käsiksi pelinkehityksen maailmaan.

LÄHTEET

Blender Foundation www-sivut 2018. Viitattu 25.1.2018. https://www.blender.org/, https://www.blender.org/foundation/history/

Chakravorty, D. 2017. 8 Most Common 3D File Formats Simply Explained. Viitattu 4.2.2018. https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/

Creger, R. 2015. 8 awesome options for 3D modeling software. Viitattu 20.2.2018. https://99designs.com/blog/tips/3d-modeling-software-guide/

Crosby, T. 2011. How Making a Video Game Works. Viitattu 14.11.2017. http://electronics.howstuffworks.com/making-a-video-game.htm

De Byl, P. 2012. Holistic Game Development with Unity. Focal Press.

Hocking, J. 2015. Unity in Action: Multiplatform Game Development in C# with Unity 5 1st Edition. Manning Publications.

Lee, Z. 2015. Building a Game with Unity and Blender. Packt Publishing.

McKleinfeld, D. 2012. Indie Game Developers Rally Behind Cheap-to-Use Unity Engine at Unite 2012. Viitattu 20.12.2017. http://www.digitaltrends.com/computing/is-the-unity-engine-ready-for-the-speedway/

Menard, M. & Wagstaff, B. 2015. Game Development with Unity, 2nd Edition, Cengage Learning.

MDU 115: Foundations of 3D Graphics. Viitattu 10.1.2018. http://emycul.blogspot.fi/2015/06/human-character-topology.html

Pickthall, J. 2012. Just what is concept art? Viitattu 14.1.2018. https://www.creativebloq.com/career/what-concept-art-11121155

Pluralsight www-sivut 2018. Viitattu 15.2.2018. https://www.pluralsight.com/blog/film-games/ngons-triangles-bad

Slick, J. 2017. The Definition of Topology and Its Purpose in 3D Animation. Viitattu 10.1.2018. https://www.lifewire.com/topology-in-3d-animation-2181

Slick, J. 2018. What is 3D Modeling? Viitattu 13.2.2018. https://www.lifewire.com/what-is-3d-modeling-2164

Solanki, N. 2015. What is UV Mapping? Viitattu 12.12.2017. http://www.theappguruz.com/blog/uv-mapping

Takahashi, D. 2014. John Riccitiello sets out to identify the engine of growth for Unity Technologies (interview). Viitattu 12.1.2018. http://venturebeat.com/2014/10/23/john-riccitiello-sets-out-to-identify-the-engine-of-growth-for-unity-technologies-interview/

Thorn, A. 2013. Unity 4 Fundamentals: Get Started at Making Games with Unity. Focal Press.

Unity Technologies www-sivut 2018. Viitattu 26.2.2018. http://unity3d.com/unity, https://www.assetstore.unity3d.com/en/, https://docs.unity3d.com/Manual/Tags.html

Ward, J. 2008. What is a Game Engine? Viitattu 29.11.2017. http://www.gameca-reerguide.com/features/529/what_is_a_game_.php

Zeman, N. 2015. Essential Skills for 3D Modeling, Rendering, and Animation. CRC Press.

```
// C#-skripti PlayerMovement.cs

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

public class PlayerMovement : MonoBehaviour {

    private const float LANE_DISTANCE = 2.5f;
    private const float TURN_SPEED = 0.05f;

    private bool isRunning = false;

    // Animaatio
    private Animator anim;

    // Liikkumis muuttujat
    private CharacterController controller;
    private float jumpForce = 6.0f;
    private float gravity = 12.0f;
    private float verticalVelovity;

    // Pelihahmo aloittaa keskikaistalta, 0 = Vasen, 1 = Keski, 2 = Oikea
    private int desiredLane = 1;

    // Nopeuden muuttujat
    private float originalSpeed = 7.0f;
    private float speed;
    private float speedIncreaseLastTick;
    private float speedIncreaseTime = 2.5f;
    private float speedIncreaseAmount = 0.25f;

    private void Start()
    {
        speed = originalSpeed;
        controller = GetComponent <CharacterController> ();
        anim = GetComponent<Animator>();
    }

    // Update-funktio suoritetaan joka framella, kun peli käynnistyy
    private void Update()
    {
        if (!isRunning)        // Ehtolause, jolla varmistetaan ettei peli ala en-
nen kuin pelaaja koskettaa näyttöä
            return;

        if(Time.time - speedIncreaseLastTick > speedIncreaseTime)
        {
            speedIncreaseLastTick = Time.time;
            speed += speedIncreaseAmount;
            GameManager.Instance.UpdateModifier (speed - originalSpeed);
        }

        if (MobileInput.Instance.SwipeLeft)
            MoveLane (false);
        if (MobileInput.Instance.SwipeRight)
            MoveLane (true);


        // Laskee millä kaistalla tullaan olemaan
        Vector3 targetPosition = transform.position.z * Vector3.forward;
        if (desiredLane == 0)
            targetPosition += Vector3.left * LANE_DISTANCE;
```

```
        else if (desiredLane == 2)
            targetPosition += Vector3.right * LANE_DISTANCE;

        // Lasketaan liikkumis delta
        Vector3 moveVector = Vector3.zero;
        moveVector.x = (targetPosition - transform.position).normalized.x * speed;

        bool isGrounded = IsGrounded ();
        anim.SetBool("Grounded", isGrounded);

        // Laskee Y-akselin
        if (isGrounded)         // Jos hahmo on maassa, painovoima ei vaikuta
        {
            verticalVelovity = -0.1f;

            // Pelihahmo hyppää mikäli käyttäjä pyyhkäisee ylöspäin
            if (MobileInput.Instance.SwipeUp) {
                anim.SetTrigger ("Jump");
                verticalVelovity = jumpForce;
            }
            else if (MobileInput.Instance.SwipeDown)
            {
                // Liukuminen
                StartSliding();
                Invoke ("StopSliding", 1.0f);

            }
        } else
        {
            verticalVelovity -= (gravity * Time.deltaTime);

            // Jos pelihahmo on ilmassa, ja käyttäjä pyyhkäisee alas, tip-
puu hahmo nopeammin maahan
            if (MobileInput.Instance.SwipeDown)
            {
                verticalVelovity = -jumpForce;
            }
        }

        moveVector.y = verticalVelovity;
        moveVector.z = speed;

        // Pelaajan liikutus
        controller.Move(moveVector * Time.deltaTime);

        // Kääntää pelaajaa menosuuntaan vaihtaessa kaistaa
        Vector3 dir = controller.velocity;
        if (dir != Vector3.zero)
        {
            dir.y = 0;
            transform.forward = Vector3.Lerp (transform.forward, dir, TURN_SPEED);
        }

    }
    // Metodi suorittaakseen liukumisanimaation ja muuttaakseen pelaajan korkeutta
välttääkseen osumia
    private void StartSliding()
    {
        anim.SetBool ("Slide", true);
        controller.height /= 2;
        controller.center = new Vector3(controller.center.x, controller.cen-
ter.y / 2, controller.center.z);
    }
    // Metodi lopettaakseen liukumisanimaation ja muuttaakseen pelaajan korkeuden
takaisin normaaliksi
    private void StopSliding()
```

```
    {
        anim.SetBool ("Slide", false);
        controller.height *= 2;
        controller.center = new Vector3(controller.center.x, controller.cen-
ter.y * 2, controller.center.z);
    }

    private void MoveLane(bool goingRight)
    {
        // Jos mennään vasemmalle, miinustetaan desiredlanen ar-
voa ja varmistetaan ettei voi mennä alle 0:n
        if (!goingRight) {
            desiredLane--;
            if (desiredLane == -1)
                desiredLane = 0;
        }
        // Jos mennään oikealle lisätään desiredlanen ar-
voa, ja varmistetaan ettei voi mennä yli 2:
        else
        {
            desiredLane++;
            if (desiredLane == 3)
                desiredLane = 2;
        }
    }

    private bool IsGrounded()
    {
        Ray groundRay = new Ray (
                            new Vector3 (
                                controller.bounds.center.x,
                                (controller.bounds.center.y - controller.bounds.ex-
tents.y) + 0.2f,
                                controller.bounds.center.z),
                            Vector3.down);
        Debug.DrawRay (groundRay.origin, groundRay.direction, Color.cyan, 1.0f);

        return (Physics.Raycast (groundRay, 0.2f + 0.1f));
    }

    public void StartRunning()
    {
        isRunning = true;
        anim.SetTrigger ("StartRunning");
    }

    private void Crash()
    {
        anim.SetTrigger ("Death");
        isRunning = false;
        GameManager.Instance.OnDeath();
    }

    private void OnControllerColliderHit(ControllerColliderHit hit)
    {
        switch (hit.gameObject.tag)
        {
        case "Obstacle":
            Crash ();
            break;
        }
    }

}
```

```
// C#-skripti MobileInputs.cs


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

public class MobileInput : MonoBehaviour {

    // deadzone on alue, jonka ulkopuolelle pyyhkäisyn tulee mennä, että se
rekisteröidään syötteeksi (tämä välttääkseen vahinkosyötteitä)
    private const float DEADZONE = 100.0f;

    public static MobileInput Instance { set; get; }

    private bool tap, swipeLeft, swipeRight, swipeUp, swipeDown;
    private Vector2 swipeDelta, startTouch;

    public bool Tap { get { return tap; } }
    public Vector2 SwipeDelta { get { return swipeDelta; } }
    public bool SwipeLeft { get { return swipeLeft; } }
    public bool SwipeRight { get { return swipeRight; } }
    public bool SwipeUp { get { return swipeUp; } }
    public bool SwipeDown { get { return swipeDown; } }

    private void Awake()
    {
        Instance = this;
    }

    private void Update()
    {
        // Alustetaan boolianit
        tap = swipeLeft = swipeRight = swipeDown = swipeUp = false;

        // Tarkastetaan syötteet
        // Standalone = hiiren syötteet pelin testaamiseen
        #region Standalone Inputs
        if(Input.GetMouseButtonDown(0))
        {
            tap = true;
            startTouch = Input.mousePosition;
        }
        else if(Input.GetMouseButtonUp(0))
        {
            startTouch = swipeDelta = Vector2.zero;
        }
        #endregion

        // Mobiilisyötteet
        #region Mobile Inputs

        // Ehtolause tarkastamaan onko näytöllä käyttäjän kosketus
        if(Input.touches.Length != 0)
        {
            // Mikä on kosketuksen tila
            if(Input.touches[0].phase == TouchPhase.Began)
            {
                tap = true;
                startTouch = Input.mousePosition;
            }
            // Muutoin jos kosketus on lopetettu tai peruttu
```

```
            else if(Input.touches[0].phase == TouchPhase.Ended || In-
put.touches[0].phase == TouchPhase.Canceled)
            {
                startTouch = swipeDelta = Vector2.zero;
            }
        }

        #endregion

        // Lasketaan etäisyys
        swipeDelta = Vector2.zero;
        if (startTouch != Vector2.zero)
        {
            // Tarkastetaan mobiilisyöte
            if (Input.touches.Length != 0) {
                swipeDelta = Input.touches [0].position - startTouch;
            }

            // Tarkastetaan standalone
            else if (Input.GetMouseButton (0))
            {
                swipeDelta = (Vector2)Input.mousePosition - startTouch;
            }
        }

        // Tarkastetaan jos kosketus on "deadzonen" ulkopuolella
        if (swipeDelta.magnitude > DEADZONE)
        {
            // Tämä on rekisteröity syöte
            float x = swipeDelta.x;
            float y = swipeDelta.y;

            if (Mathf.Abs (x) > Mathf.Abs (y)) {
                // Vasen ja oikea
                if (x < 0)
                    swipeLeft = true;
                else
                    swipeRight = true;
            } else
            {
                // Ylös ja alas
                if (y < 0)
                    swipeDown = true;
                else
                    swipeUp = true;
            }

            startTouch = swipeDelta = Vector2.zero;
        }
    }

}
```