KYMENLAAKSON AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma / Ohjelmistotekniikka

Elina Salo

3D MALLINNUS PELEIHIN JA MALLIEN KÄYTTÖ UNITY3D:SSÄ

Opinnäytetyö 2015

TIIVISTELMÄ

KYMENLAAKSON AMMATTIKORKEAKOULU
Ohjelmistotekniikka

SALO, ELINA	3D mallinnus peleihin ja mallien käyttö Unity3D:ssä
Opinnäytetyö	32 sivua + 10 liitesivua
Työn ohjaaja	Opettaja Paula Posio
Toimeksiantaja	Kymenlaakson ammattikorkeakoulu
Huhtikuu 2015	
Avainsanat	3d, peli, ohjelmointi, unity3d, c#

Nykyään suuri osa peleistä käyttää 3D-grafiikkaa peleissään 2D-grafiikan sijaan. Tämän takia pelialalla olisi hyvä osata tehdä 3D-malleja, sekä tietää kuinka niitä malleja käytetään pelien tekemisessä.

Opinnäytetyössä esitellään teoriaa 3D-mallinnuksesta peleihin, erityisesti minkälaisia asioita täytyy pitää mielessä peleihin mallinnettaessa. Käytännön mallinnusosiossa käytetään Autodesk 3ds Max -ohjelmaa, ja tehdään muutama yksinkertainen 3D-malli. Yhteen malliin tehtiin myös luuranko, jonka avulla mallia voidaan liikuttaa. Lisäksi toteutettiin yksinkertaiset animaatiot.

Mallien teon jälkeen tehtiin Unity3D-ohjelmalla yksinkertainen peli, jossa malleja käytetään. Pelin skripteissä käytettiin C#-ohjelmointikieltä.

Lopputuloksena on yksinkertainen yhden kentän peli, jossa pelaaja ohjaa animoitua 3D-hahmoa sokkelossa ja välttelee vihollisia.

Opinnäytetyön tekeminen oli varsin opettava kokemus. Siinä oppi hyvin, millaisia asioita pitää peleihin mallinnettaessa ottaa huomioon sekä kuinka malleja käytetään peleissä.

ABSTRACT

KYMENLAAKSON AMMATTIKORKEAKOULU
University of Applied Sciences
Information Technology

SALO, ELINA	3D modeling for Games and Using Models in Unity3D
Bachelor's Thesis	32 pages + 10 pages of appendices
Supervisor	Paula Posio, Principal Lecturer
Commissioned by	Kymenlaakso University of Applied Sciences
April 2015	
Keywords	3d, game, programming, unity3d, c#

Nowadays numerous games use 3D graphics instead of using 2D graphics. This is why game development benefits from knowledge of how to create 3D models and how to use them

This thesis explains the theory of 3D modeling for games, for example what is good to keep in mind when modeling for games. Autodesk 3ds Max program was used for the modeling practice, and a few simple models were created with it. One of the models is rigged so it can be moved, and simple animations were created for it.

After making the models, a simple game was created using Unity3D, and the 3D models were used in it. The scripts were created with C# programming language.

The final result of the thesis is a simple game with one level, where the player controls an animated 3D model in a maze and avoids enemies.

SISÄLLYS

Liite 2. PlayerMovement.cs

Liite 3. CharacterRotate.cs

Liite 4. EnemyPatrol.cs

Liite 5. GameManager.cs

Liite 6. ArrowRotate.cs

Liite 7. Destoy.cs & Menu.cs

LYHENTEET JA TERMIT

3D	Kolmiulotteinen. Kolmiulotteisilla objekteilla on pituus, leveys sekä syvyys.
Polygoni	Monikulmio, 3D-mallit koostuvat kolmi- ja nelikulmaisista polygoneista.
Vertexi	Piste, mikä on polygonimallissa polygonien kärjissä.
UV map	Mallipohja tekstuurille, joka saadaan ikään kuin avaamalla ja tasoittamalla 3D-mallin pinnat kaksiulotteiseen muotoon.
Key frame	3D-mallin animoinnissa asetettu avainkohta, johon määritetään missä asennossa mallin halutaan kyseisellä hetkellä olevan.
Skripti	Kooditiedosto, jossa voidaan määrittää halutut toiminnat peliobjektille.
Prefab	Unityssa luotu valmis malliversio peliobjektista, se säilyttää objektille määritetyt asetukset ja komponentit.

JOHDANTO

Olen aina ollut enemmän graafinen ihminen kuin ohjelmoija, ja kiinnostunut pelien graafisesta puolesta. Tämän takia pelien 3D-mallien tekoa koskeva aihe tuntui hyvältä opinnäytetyöaiheelta. Tutustuin hieman siihen, minkälaisia asioita pitää ylipäätänsä ottaa huomioon peleihin mallinnettaessa, ja sen jälkeen oli ideana tehdä muutama oma malli ja peli, joissa käyttää kyseisiä malleja.

Työkaluksi valitsin mallien tekoon 3ds Max -ohjelman, sillä se on käytössä monissa pelifirmoissa, joten sen tutkiminen vaikutti kannattavalta. Lisäksi siitä sai ilmaisen opiskelijalisenssin Autodeskin sivuilta. Aikaisempaa kokemusta mallintamisesta ja kyseisestä ohjelmasta oli vähän. Olin tätä ennen tehnyt yhden 3D-mallin, joka oli toteutettu samalla ohjelmalla.

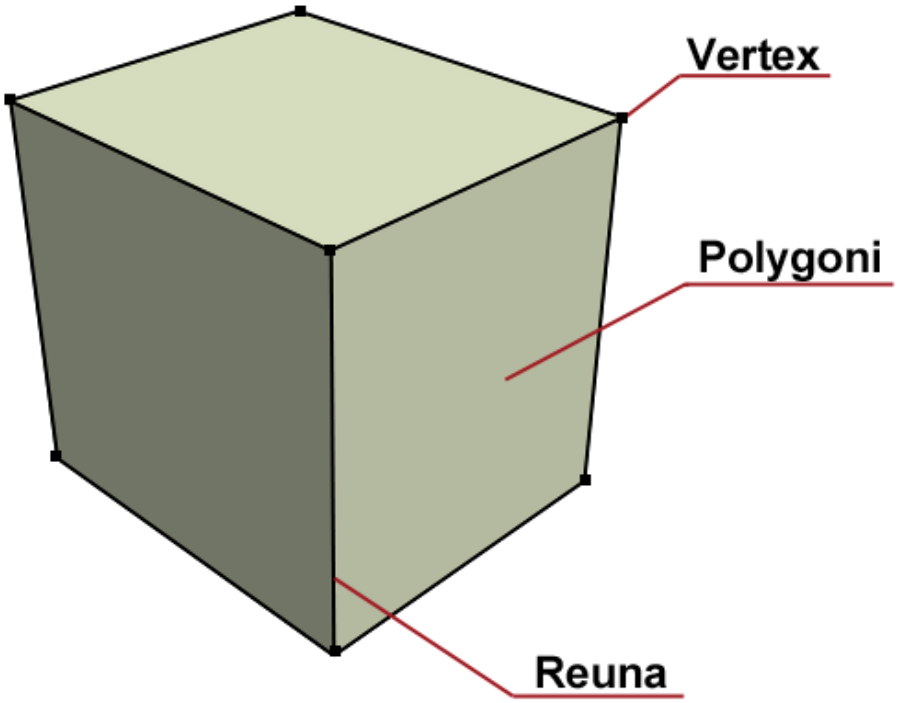
Ohjelmointiosioon päätin käyttää Unity3D:tä, sillä myös se on käytössä monissa pelifirmoissa ja se soveltuu hyvin 3D pelien tekoon. Pitkän aikaa kyseisessä ohjelmassa ei edes ollut omia työkaluja 2D tekemiseen, sillä se keskittyi 3D:hen. Unitystä löytyy myös paljon tietoa ja ohjeita, mikä oli hyvä, sillä kyseinen ohjelma ei ollut järin tuttu minulle ja viime käyttökerrasta oli kulunut kauan aikaa. Ohjelmointikielenä käytin C# kieltä, sillä se oli minulle ennestään tuttu. Muita Unityssä toimivia kieliä en ollut juurikaan käyttänyt.

Opinnäytetyön tavoitteena oli tutustua pelien mallintamiseen, tehdä muutama 3D-malli ja käyttää niitä yksinkertaisessa toimivassa pelissä.

1. 3D-MALLINNUS

1.1 3D-mallinnuksen perusteet

Peleissä 3D-mallit koostuvat pääosin kolmesta eri osasta: itse polygonimalli, mallin tekstuuri sekä mallin luuranko. 3D-mallien tekemiseen on muutamia erilaisia mallinnuskeinoja, mutta polygonimallintaminen on näistä yleisin.



Kuva 1: Polygonimallin osat.

Polygonimallit koostuvat yksinkertaisista nimensä mukaisista polygoneista eli monikulmiosta. 3D-malleissa nämä polygonit ovat joko nelikulmioita tai kolmioita. Polygonimallissa on määriteltynä vertexejä eli pisteitä ja niiden välille reunoja, jotka on aseteltu halutuille kohdille koordinaatistossa. Polygonit asettuvat siten, että niiden kulmat ovat tietyissä pisteissä ja kahden polygonin välillä on reuna. Tällä tavoin muodostuu 3D-polygonimalli (kuva 1). Polygonien lukumäärä ja se, kuinka tiheästi niitä on, määrittää 3D-mallin tarkkuuden. Vähäinen määrä polygoneja saa mallin näyttämään varsin kulmikkaalta, mutta suuri määrä polygoneja on raskaampi käyttää. (1.)

Tekstuurit ovat toinen osa peleissä näkyvistä malleista. Ilman tekstuureita kaikki mallit olisivat vain yksivärisiä muotoja. Tekstuurissa 3D-mallin polygonit on ikäänkuin avattu ja tasoitettu 2D-pinnaksi, hieman niin kuin eläin olisi nyljetty ja siitä olisi saatu talja. Tähän tekstuuripohjaan voidaan laittaa haluttuja värejä ja yksityiskohtia, jotka tulevat näkymään mallissa, kun tekstuuri lisätään siihen. (2, 13-14.)

Jos 3D-malli on tarkoitus animoida, malliin kuuluu vielä luuranko. Tämä luuranko määrittelee, mistä kohdista hahmo voi taipua, aivan kuin ihmisenkin luuranko tekee.

Kun hahmon luuranko on luotu, tulee määrittää, mikä luu vaikuttaa mihinkin kohtaan mallista ja kuinka paljon.

1.2 3D-mallinnus peleihin

Peleihin mallintaminen eroaa hieman esimerkiksi elokuviin mallintamisesta. Periaatteet ovat samat, mutta mallien muodostuksessa tulee ottaa huomioon hieman erilaisia asioita. Peleissä se laite, jolla peliä pelataan renderöi eli luo kuvaa mallien datasta reaaliajassa, kun taas elokuvissa jokainen kuva mallinnetaan valmiiksi etukäteen. Tämän takia peleissä mallit eivät saa olla liian raskaita ja yksityiskohtaisia, vaan niiden tulee olla sopivia alustaan nähden, kun taas 3D-elokuvissa animaattorit voivat hioa jokaisen kohdan näyttämään niin hyvältä kuin on tarvetta, jos aikaa riittää. Erittäin laadukkaiden 3D-elokuvien jokaisen kuvan mallintamisessa voi mennä päiviä, mutta peleissä tämä ei ole mahdollista. Sen takia pelejä mallinnettaessa on erilaisia keinoja, joilla saa paremman näköisiä malleja ja sujuvampaa peliä vähemmällä polygonimäärällä. (3.)

Yksi keino on mallintaa vain se, mitä on näkyvissä. Jos peli on kuvattu ensimmäisestä persoonasta, ei koko hahmoa tarvitse olla mallinnettuna ruudulla, vaan voidaan tyytyä mallintamaan vain ne osat, jotka mallista näkyy, kuten hahmon kädet. Toinen tapa vähentää ruudulla olevia polygoneja on se, että kauempana olevista asioista esitetään ruudulla mallit, joissa on vähemmän polygoneja. Tämä toimii hyvin, sillä pelaaja eivät kuitenkaan erottaisi kaikkia yksityiskohtia kaukana olevasta asiasta, joten sitä voidaan hyvin hieman yksinkertaistaa ja vähentää polygoneja. (4.)

Laadukkaat tekstuurit ovat myös hyvä tapa saada malli näyttämään hyvältä, vaikka siinä olisi vähemmän polygoneja. Joitakin yksityiskohtia ei välttämättä tarvitse mallintaa, vaan ne voidaan tuoda esiin tekstuurissa. Tekstuureissa hyödynnetään usein myös toistuvuutta. Esimerkiksi rakennuksen seinässä voidaan käyttää pienempää tekstuuria, joka toistuu saumattomasti yhä uudelleen ja uudelleen, täyttäen koko halutun alueen.

Pelien 3D-mallien tekeminen ja animoiminen vievät paljon aikaa, minkä takia pelejä tehtäessä suositellaan elementtien uudelleenkäyttöä. Esimerkiksi samaa animaatiota voidaan käyttää useammalla hahmolla, jos hahmojen rakenne on samankaltainen. Ihmishahmoilla hahmon luurangon rakenne on sama, minkä takia yhden hahmon

animaatioita voidaan hyödyntää toisessakin hahmossa. Tämä on varsin hyödyllistä ja säästää aikaa. (5, 223.)

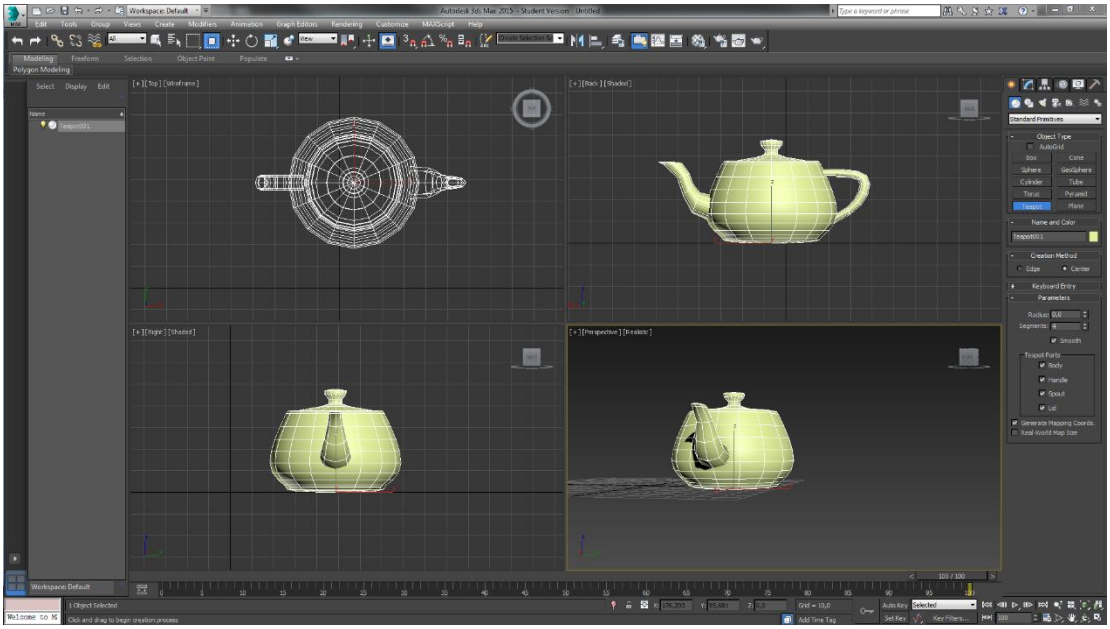
3D elementtejä voidaan myös käyttää uudelleen, esimerkiksi pelin kaupunkia tehtäessä ei jokaista rakennusta tarvitse mallintaa omanlaisekseen, vaan voidaan käyttää uudelleen samoja rakennuksia ja tekstuurien avulla ne saadaan näyttämään hieman erilaisilta. (6.)

2. KÄYTETTYJEN TYÖKALUJEN KUVAUKSET

2.1 3ds Max

Autodesk 3ds Max on jo yli 20 vuotta vanha Windows käyttöjärjestelmillä toimiva 3D- mallinnus ja renderöintiohjelmistosarja. Alkuperäiseltä nimeltään se oli 3D Studio Max. Kyseisellä ohjelmalla voidaan muun muassa mallintaa, animoida ja renderöidä 3D-malleja, ja kyseinen ohjelma on usein ammattimaisessa käytössä. Useiden pelien hahmot ja muut elementit on mallinnettu 3ds Maxilla ja joissakin elokuvissa on myös käytetty kyseistä ohjelmaa 3D efekteissä. (7.)

Autodesk 3ds Max -ohjelman avatessa tulee näkyviin työtila, joka on jaettu neljään eri osaan. Näissä ruuduissa näkyy mallinnusalue eri näkökulmista, ja niitä voidaan muokata omien tarpeiden mukaan näyttämään malli tarvituista suunnista (kuva 2).



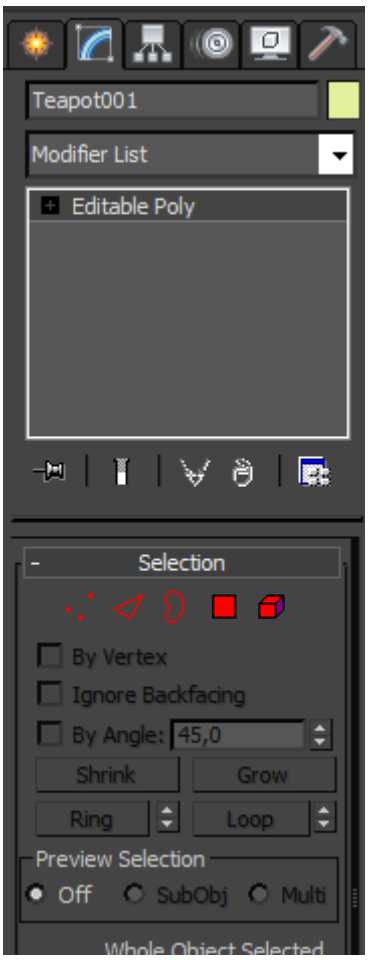
Kuva 2. 3ds Maxin perusnäkymä.

Ruuduissa voidaan myös vaihtaa sitä, millä tavalla se näyttää mallit. Aloitettaessa oletuksena kaikissa muissa ruuduissa paitsi yhdessä on valintana wireframe, eli suomeksi käännettynä rautalanka valinta. Rautalankamallissa näkyy ainoastaan tasojen ääriviivat. Mallinnettaessa kannattaa vaihtaa sivuilla oleviin näkymiin shaded eli varjostettu valinta, sillä siinä näkyy mallin tekstuuri tai materiaali.

Ohjelman yläpalkista löytyy Create eli luontivalikko, ja sen alta löytyy alavalikko Standard Primitives, josta löytyy mallintamisessa käytettävät perusmuodot. Tästä valikosta voi luoda tarvittavat muodot mitä aiotaan käyttää pohjana mallintamisessa.

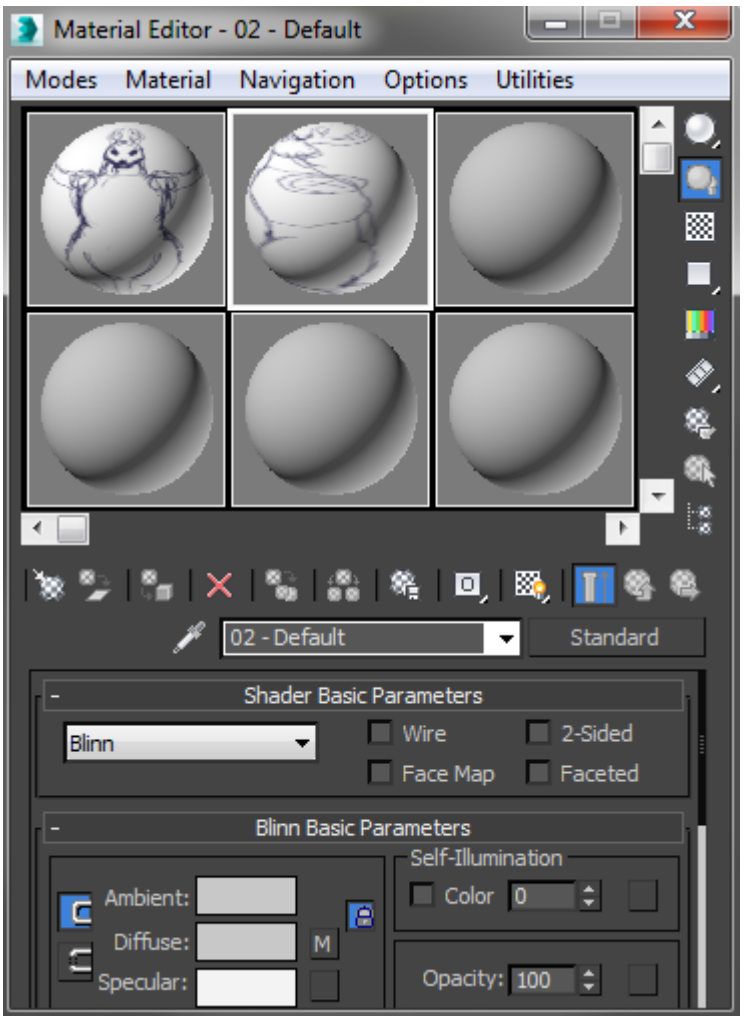
Näpäyttämällä hiiren oikeanpuoleisella painikkeella luotua mallia saadaan esiin valikko, mistä voidaan muun muassa määritellä kuinka mallia käsitellään. Move eli liikutusvalinnassa mallia voidaan liikuttaa näkyvissä olevista nuolista x, y ja z koordinaateissa. Rotate eli kääntämisvalinnassa mallia voidaan kääntää näkyvissä olevista ympyröistä. Ja Scale eli skaalausvalinnassa mallia voidaan suurentaa ja pienentää. Valikosta löytyy myös valinnat, joilla malli voidaan muuttaa muokattavaan muotoon. Tämä täytyy tehdä malleille, ennen kuin niitä voi muokata.

Kun on valittuna malli, joka on muokattavassa muodossa, näkyy oikealla palkissa Modify välilehden alla mallin muokkaustyökaluja. Sieltä malliin voidaan lisätä ns. Modifier-ominaisuuksia, kuten symmetriaominaisuus. Hieman alempaa löytyy kohta, jossa voidaan määritellä halutaanko käsitellä mallin vertexejä, reunoja, vai tasoja. Näiden alta löytyy vielä valintaan liittyviä lisäominaisuuksia ja työkaluja, joita voi käyttää mallinnettaessa (kuva 3).



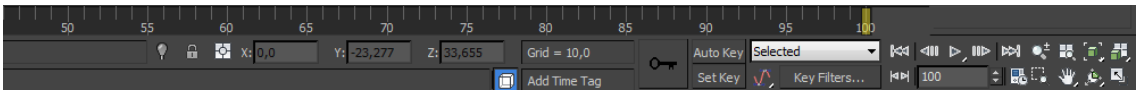
Kuva 3. Modifier välilehti.

Materiaalivalikosta pystyy muokkaamaan mallien eri materiaaleja, ja luomaan uusia materiaaleja, jos on tarvetta (kuva 4). Kyseisen valikon saa auki päävalikoista tai painamalla M-näppäintä näppäimistöstä.



Kuva 4. Materiaalivalikko.

Animaatiovalikon alta löytyy Bone-työkalu, jolla luodaan mallille animoinnissa tarvittavat luut. Koko näkymän alareunasta löytyy aikajana, jonka avulla mallille tehdään animaatiot (kuva 5). Kyseiselle janalle asetetaan mallille avainasentoja ja malli liikkuu automaattisesti avainasentojen välillä.



Kuva 5. Animaation aikajana.

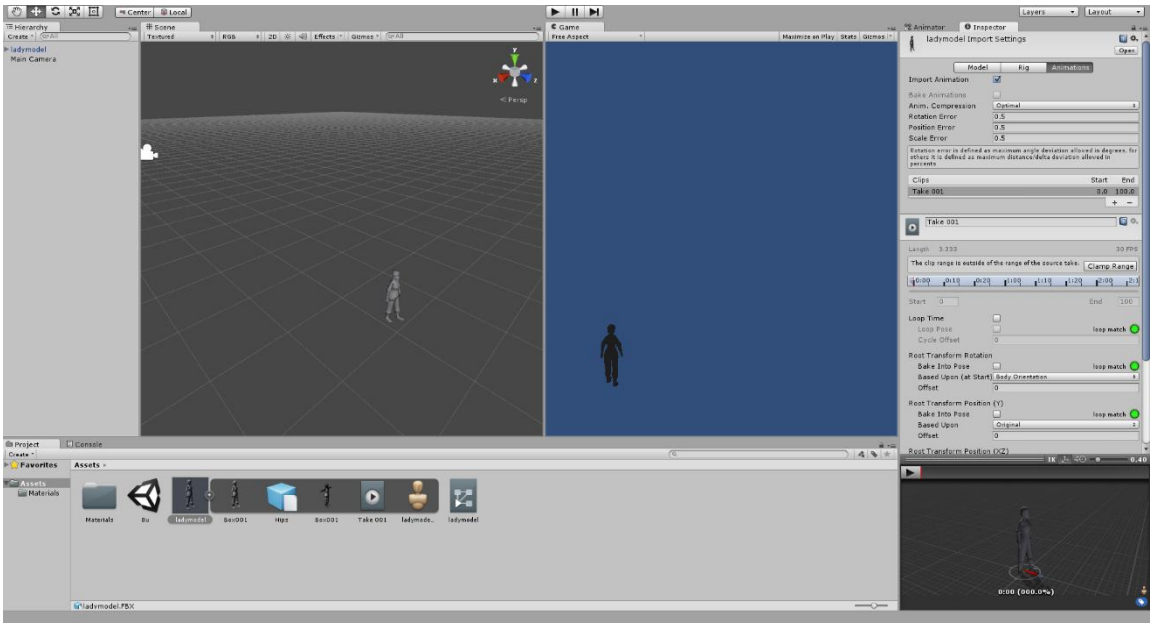
2.2 Unity3D

Unity3D on komponenttipohjainen pelimoottori, jolla voidaan tehdä pelejä useille eri alustoille, ja siitä on olemassa niin ilmaisversio kuin maksullinenkin versio. Ilmaisversiossa on hieman vähemmän ominaisuuksia kuin maksullisessa versiossa, mutta ilmaisversiotakin voidaan käyttää peleissä, joita aiotaan myydä ilman, että

tarvitsisi maksaa Unitylle lisenssimaksuja. Tämän takia Unity on varsin suosittu varsinkin aloittelevien pelintekijöiden parissa. (8.)

Unityn editorin käyttöliittymä on varsin selkeä ja helppokäyttöinen. Siinä voidaan vetää peliobjekteja pelialustalle, ja niille voidaan helposti lisätä editorissa komponentteja, jotka antavat objektille tiedot, miten se toimii. Objektiin voi esimerkiksi lisätä fysiikkaominaisuuden tai skriptikomponentteja. Skriptit ovat kooditiedostoja, joilla voidaan määrittää, miten objektin halutaan käyttäytyvän. Useiden eri komponenttien ansiosta Unityllä olisi mahdollista tehdä yksinkertaisia pelejä erittäin vähäisellä skriptausmäärällä. Unityn skripteissä käytetään C#-, Javascript- tai Boo-ohjelmointikieltä. Unitystä ja siihen skriptaamisesta löytyy paljon materiaalia ja ohjeita. (9.)

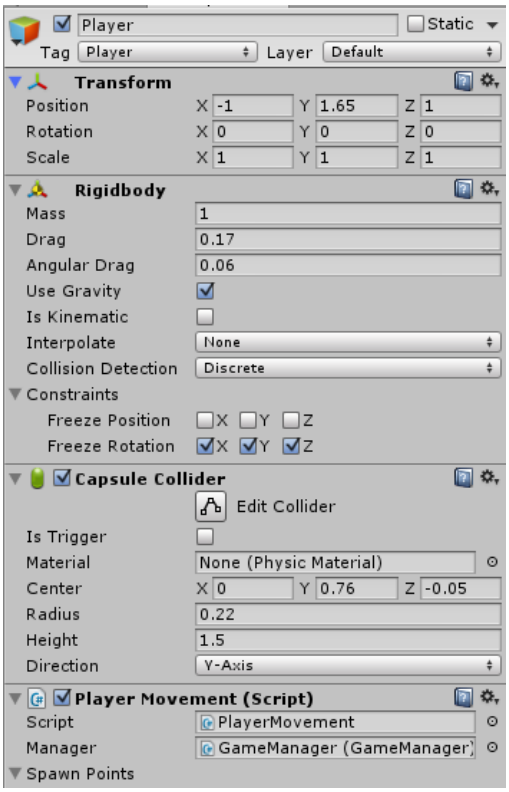
Aloitettaessa uusi Unity-projekti, avautuu näkymä, jossa isoimpana näkyvät Scene-ruutu, missä tapahtuu pelin muokkaus. Sen vierellä on Game-ruutu, missä näkyy pelin kameran näkökulmasta, eli siitä näkee miltä peli näyttää. Näkymän reunoilta löytyy pelin hierarkianäyttö, projektin tiedostot -näyttö, sekä valitun objektin tiedot näyttävä Inspector (kuva 6).



Kuva 6. Unityn perusnäkymä.

Scene ruudussa tapahtuu pelikentän luominen, ja sinne asetellaan peliin kuuluvia objekteja. Unityssä voi GameObject -valikosta luoda erilaisia peliobjekteja, kuten yksinkertaisia muotoja ja valonlähteitä. Peliobjekteja voidaan käyttää pelissä, ja näille voidaan asettaa Inspectorissa skriptejä ja muita ominaisuuksia (kuva 7). Näistä

objekteista voi sen jälkeen luoda Prefabeja eli ikään kuin valmiita malliversioita. Prefab säilyttää objektin sisältämät komponentit ja skriptit, ja niin voidaan luoda useita samanlaisia objekteja helposi. Yksi esimerkki komponenteista on Collider-komponentit, jotka tekevät törmäystarkastelua ja katsovat osuvatko ne toiseen collideriin.



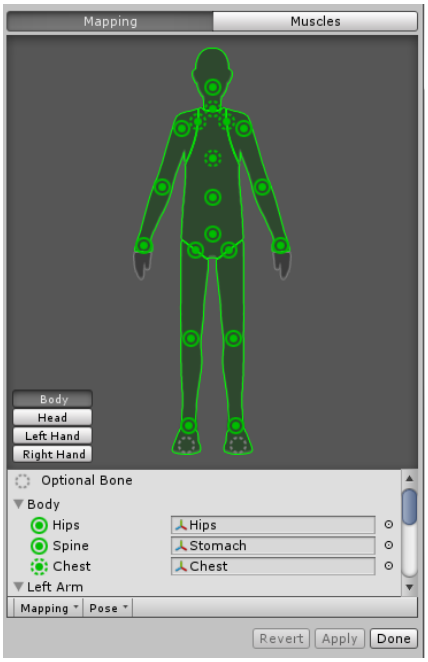
Kuva 7. Inspector-näkymä kun peliobjekti on valittuna.

Luotujen skriptien ja peliobjektien toimivuutta voidaan testata Game-ruudussa, jonka yläpuolella olevista napeista painamalla voidaan peli käynnistää. Tällöin Game-ruudussa alkaa luotu peli pyörimään, ja siitä voi tarkistaa ja testata toimivatko skriptit oikein. Jos pelin pyöriessä siinä ilmenee joitakin virheitä, virheilmoitukset tulevat näkyville Console-kohtaan. Sinne tulee myös ilmoitus, jos jossakin skriptissä on sellainen vika, mikä estää pelin käynnistämisen.

Assets-valikosta voidaan tuoda peliin Unityn ulkopuolisia resursseja, kuten 3D-malleja tai tekstuureita. Kun projektin tiedostot näyttävässä kohdassa valitsee tuodun 3D-mallin, näkyy Inspector-kohdassa tietoja ja asetuksia mallista, ja sen kautta pystyy muuttamaan hahmon animaatiotyyppiä ja säätelemään mallin mahdollisia animaatioklippejä.

Jos kyseinen malli on ihmismäinen malli, voidaan sen animaatiotyyppi valita ihmismäiseksi, jolloin pääsee muokkaamaan, mikä mallin luu vastaa mitäkin ihmisen

ruumiinosaa (kuva 8). Tämä on tarpeellista, jos käyttää muualta hankittuja valmiita ihmisanimaatioita.



Kuva 8. Ihmismallin luiden määrittely.

Unityn Animator-kohdassa näkyy pelissä olevan valitun mallin mahdolliset animaatiotiedostot, ja sen kautta voidaan eri animaatioklippien välille laittaa ehtoja, joissa määritellään mikä aiheuttaa animaation vaihtumisen. Ehtoina voivat toimia esimerkiksi kulunut aika tai voidaan luoda muuttujia ja määrittää, että animaatio vaihtuu muuttujan arvon ollessa tietynlainen.

3. TYÖN TOTEUTUS

3.1 Suunnitelmat

Tarkoituksena oli luoda kolme 3D-mallia, yksi pelihahmolle, yksi vihollisille ja yksi maalina toimiva malli, ja sen jälkeen luoda peli, missä käytetään kyseisiä malleja. Suunitelmana oli tehdä yksinkertainen peli, jossa tarkoituksena on sokkelossa välttää vihollista ja kerätä mahdollisimman paljon pisteitä ennen kuin aika loppuu.

Mallinnettaessa haluttiin mallintaa päähahmoksi ihminen, jotta sille päästäisiin toteuttamaan yksinkertaiset ihmisanimaatiot. Vihollishahmoon haluttiin mallintaa jotakin peuran pääkalloa muistuttavaa. Lopulta päädyttiin pystyssä kulkevaan hahmoon, jonka pää muistuttaa hieman peuran pääkalloa. Tähän tuli inspiraatiota The Wolf Among Us -pelin Jersey Devil -hahmosta (kuva 9). Maalina toimivaksi malliksi

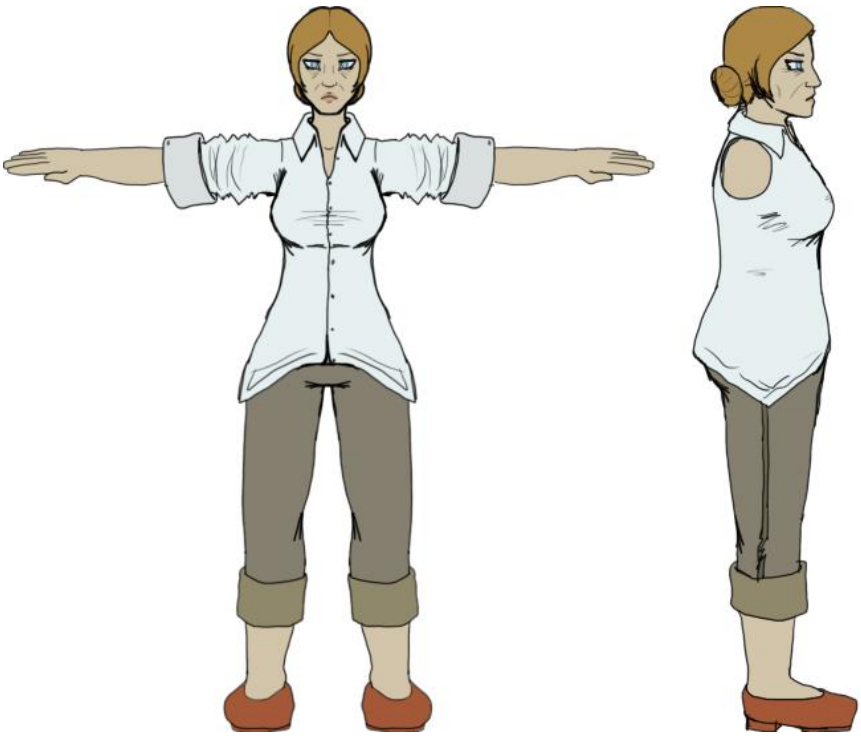
päätettiinn tehdä kissa. Pelin ideaksi muodostui se, että pelaajahahmon tarkoistus on pelastaa mahdollisimman monta kissaa sokkelossa, ennen kuin aika loppuu. Viholliseen osuessaan hahmo ei kuole, mutta pelaaja joutuu takaisin aloituspisteeseen ja menettää 10 sekuntia ajastaan.

Kuva 9. Jersey Devil hahmo The Wolf Among Us pelistä.

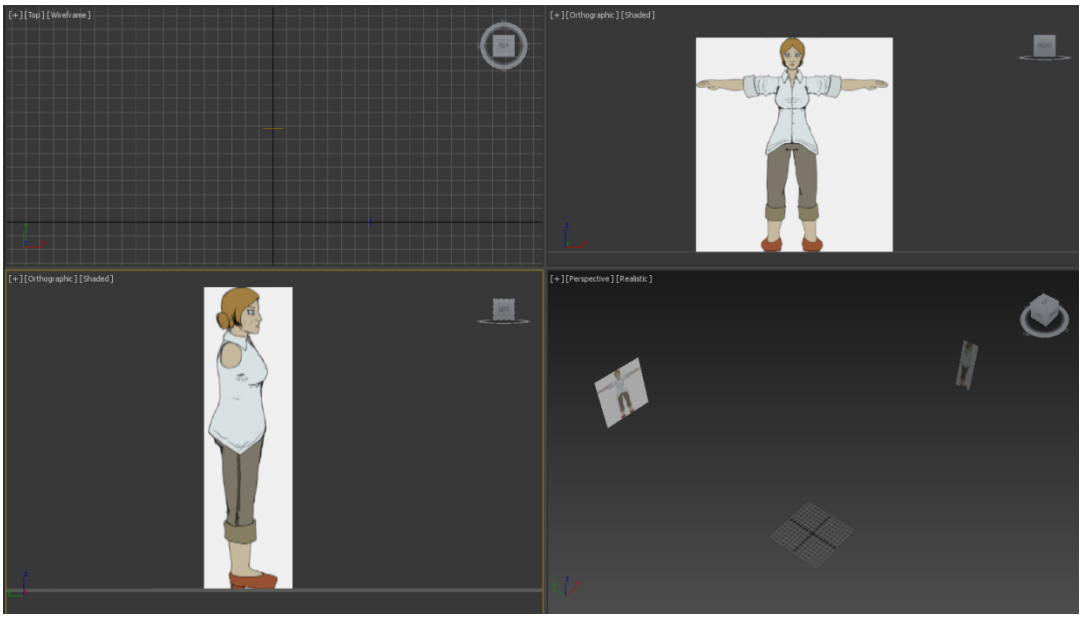
Käytettävissä olevan ajan vuoksi päätettiin, että osa hahmoista saa jäädä keskeneräisemmiksi. Mallinnus on mukaansatempaavaa mutta hidasta, jolloin mallien viimeistelyyn uppoaa helposti liikaa aikaa. Päädyttiin ratkaisuun, jossa pelaajahahmo viimeistellään pisimmälle tekstuureineen ja luineen, kissalle ei laiteta luita eikä animaatioita ja vihollinen saa jäädä yksinkertaisimmaksi luonnokseksi.

3.2 3D-mallin toteutus

3D-mallien tekeminen aloitetaan siitä, että halutusta mallinnettavasta asiasta tehdään luonnokset sekä edestä että sivulta (kuva 10). Kun nämä luonnokset on tehty, avataan 3ds Max ohjelma, jossa luodaan kaksi plane eli tasoelementtiä, jotka sijoitetaan mallinnusalueella siten, että toinen taso näkyy keskellä edestä katsottuna, ja toinen näkyy keskellä sivulta katsottuna. Tämän jälkeen näihin laitetaan tekstuureiksi hahmosta tehdyt luonnokset, jotka toimivat mallinnettaessa mallina (kuva 11). Tällä järjestelyllä on ideana se, että hahmoa muotoiltaessa voidaan yrittää seurata hahmon takana näkyvää mallia.



Kuva 10. Luonnos pelaajan hahmosta.

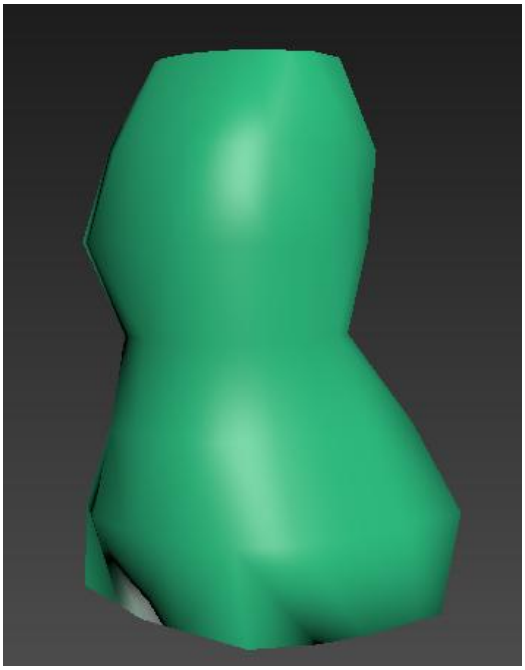


Kuva 11. Luonnos aseteltuna 3ds Maxiin.

Itse mallin tekeminen aloitetaan luomalla luonnoksiin verrattuna sopivan kokoinen nelikulmio, joka tulee olemaan hahmon kehon pohja. Ennenkuin nelikulmiota aletaan tarkemmin muokkaamaan, poistetaan kuutiosta puolet, ja sen jälkeen lisätään kulmioon symmetriaominaisuus. Symmetriaominaisuus aiheuttaa sen, että malli muodostuu kahdesta puolikkaasta, jotka ovat toistensa peilikuvat. Jos jotakin kulmion pistettä muokataan, toisella puolella vastaavalle pisteelle käy samoin. Symmetriaominaisuus on varsin hyvä, jos mallinnetaan asioita, jotka ovat suhteellisen

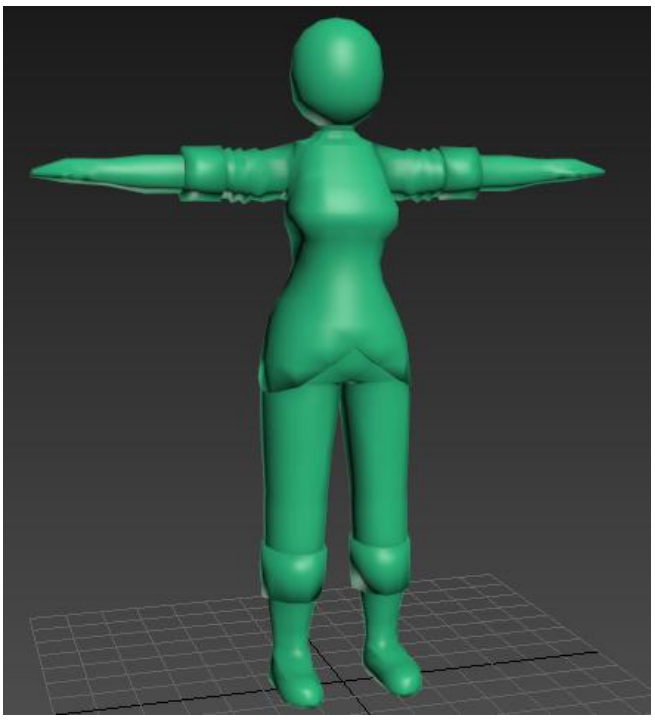
symmetrisiä, kuten ihminen. Symmetriassa on kyllä omat ongelmansa, joita täytyy välttää. Esimerkiksi kulmion keskimmäisiä pisteitä siirrettäessä pitää varoa, ettei aiheuta malliin reikää.

Symmetriaominaisuuden lisäyksen jälkeen kulmiota aletaan muokata. Aivan aluksi kulmio muokataan karkeasti seuraamaan luonnoksen muotoja (kuva 12), ja pikkuhiljaa sitä hienosäädetään, esimerkiksi kulmia pehmennetään ja niin edelleen. Kädet ja jalat hahmolle tehdään esimerkiksi extrude- eli ulostyöntöominaisuudella, jolla valituista tasoista saadaan työntymään ulos halutun pitkiä osia. Tämän jälkeen ulos työntyneet käsien ja jalkojen perustat muovataan halutunlaisiksi.



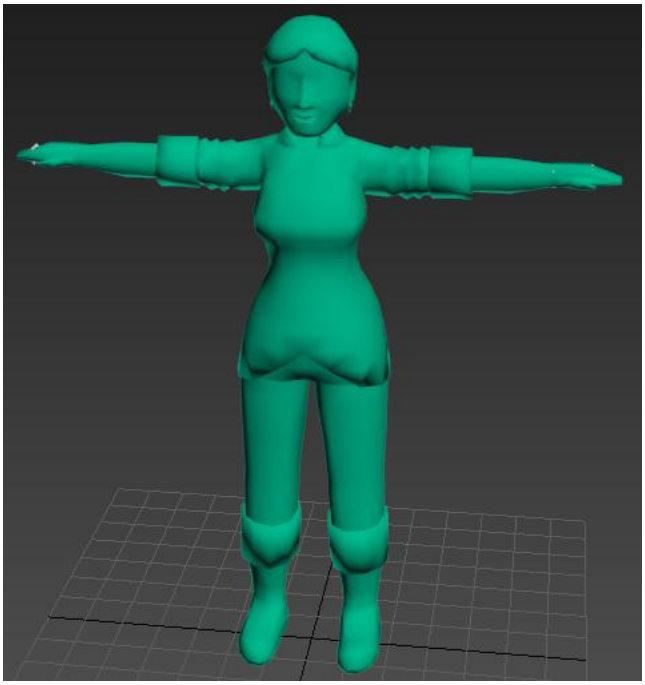
Kuva 12. Alkuvaiheessa oleva malli.

Hahmon päätä varten täytyy mallin symmetriaominaisuus ottaa pois hetkeksi, jos hahmon pää tehdään erillisestä pallosta, joka kiinnitetään hahmolle muokattuun kaulaosaan. Luodaan uusi halutunlainen valmispallo, joka siirretään suunnilleen haluttuun kohtaan mallin yläpuolelle. Tämän jälkeen malli ja pallo yhdistetään olemaan osa samaa elementtiä siten, että jos toista liikuttaa niin toinenkin liikkuu. Sitten sekä mallin kaulan kohdalta sekä pallon alapuolelta poistetaan tasot niistä kohdista, mistä niiden pitäisi olla kiinni toisissaan. Tämän jälkeen valitaan syntyneiden aukkojen reunat ja yhdistystyökalulla yhdistetään kaula ja pallo. Yhdistystyökalu luo tasot kyseisten yhdistettävien elementtien välille, luoden hahmolle kaulaa (kuva 13).



Kuva 13. Malli jonka pää on juuri kiinnitetty.

Pään yhdistämisen jälkeen hahmoon lisätään uudestaan symmetria, mikä tapahtui samalla tavalla kuin aikaisemmin. Sen jälkeen taas jatketaan hahmon muovaamista halutunlaiseksi (kuva 14).

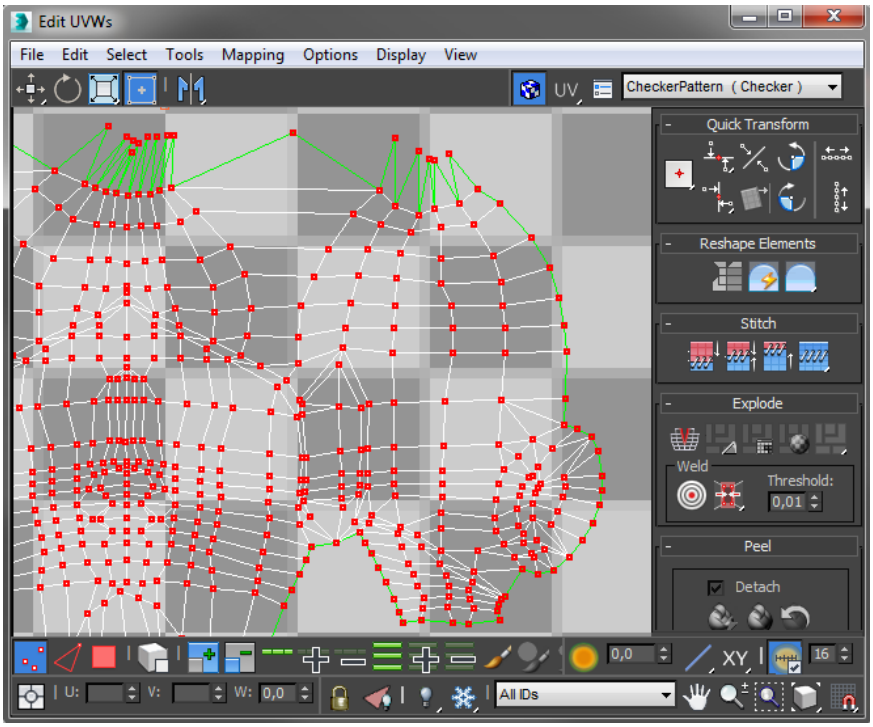


Kuva 14. Valmiiksi muotoiltu malli.

3.2.1 UV map ja tekstuuri

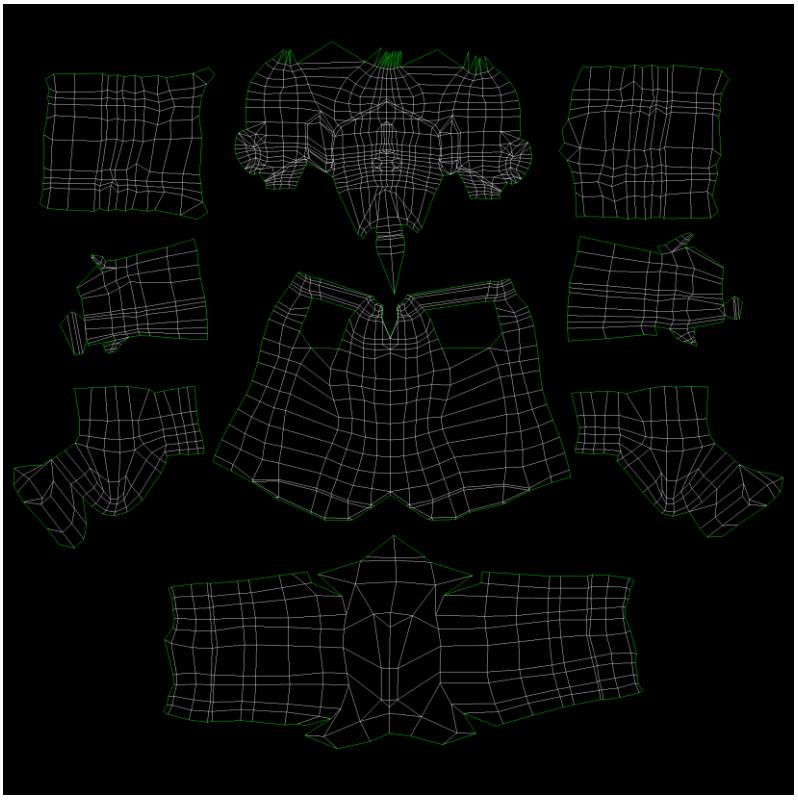
Kun malli on saatu halutunlaiseksi, seuraavaksi on vuorossa hahmon UV mappaus, mikä tarkoittaa sitä, että hahmosta tehdään niinsanottu kartta, jossa hahmon tasot on ikäänkuin tasoitettu kaksiulotteiseksi pinnaksi. Tämän tarkoituksena on saada mallille määriteltyä se, miten mallin tekstuuri tulee menemään. Ennen kuin UV mappausta voi aloittaa, täytyy hahmon symmetria finalisoida hahmoon kiinni. Mallin täytyy myös olla valmis ennen tätä, sillä jos mallia muutetaan UV mappauksen jälkeen, niin se voi sotkea kyseisen UV mappauksen, ja mappaus täytyisi tehdä uudestaan.

UV mappauksessa avautuu uusi ikkuna nimeltään UV editori (kuva 15), josta näkee, miltä kyseinen UV map näyttää sillä hetkellä. UV mappia luotaessa valitaan mallista tasoja, ja valitut tason siirretään napinpainalluksella näkymaan UV editorissa. Editorissa on eri työkaluja, joilla voidaan esimerkiksi yhdistää tasoja, sillä koko 3D-mallia ei pysty yhdellä kerralla siirtämään UV mapiksi, vaan se täytyy tehdä pala kerrallaan. UV mapissa jää aina jotkin reunat avoimiksi, sillä periaatteessa se on ikäänkuin eläimen talja. Nämä avoimet reunat kannattaa yrittää sijoittaa siten, että ne ovat paikassa missä ne eivät ole kauhean näkyvällä paikalla, sillä niissä kohdissa voi helposti sattua virheitä teksturoitaessa ja hahmoon voi jäädä näkymään saumaa niille kohdille.



Kuva 15. UV editori.

Kun hahmon kaikki tasot on saatu siirrettyä UV mappiin ja yhdistettyä miten haluaa, täytyy ne enään asetella siten että kaikki osat ovat näkyvissä ruutukuvioisella taustalla. Tämän on sen takia, että kun UV mapista tallennetaan tekstuurimalli, vain ruudutettu alue tulee malliin. Kun tekstuurien tekoon tarvittava malli on tehty (kuva 16), mallille tehdään tekstuuri. Tähän tarvitaan jokin kuvankäsittelyohjelma, jossa avataan kyseinen tekstuurimalli ja sitten vain laitetaan haluttuja värejä haluttuihin kohtiin. Tekstuuria tehtäessä on hyvä pitää 3ds Max auki, sillä jos tekstuuri tallennetaan aina välillä, ja 3D-mallille on asetettu tekstuuriksi kyseinen tekstuuri, voidaan heti tarkistaa miltä tekstuuri näyttää. Tämä on erittäin hyvä menetelmä varsinkin yksityiskohtia kuten silmiä tehtäessä, sillä voi olla vaikeaa saada silmät oikeille paikoille. Tässä vaiheessa tekstuuria vain muokataan niin kauan, kunnes tulos on halutunlainen (kuva 17).



Kuva 16. Valmis UV map.

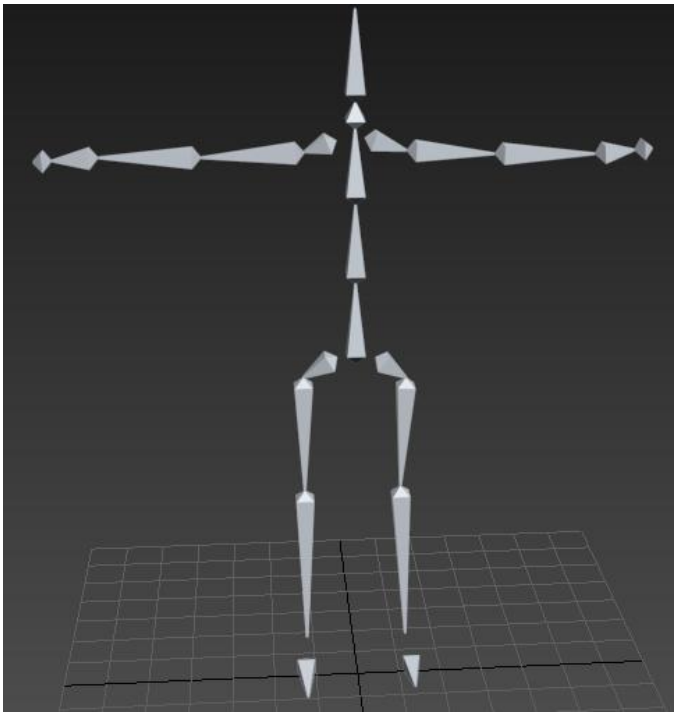
Kuva 17. Pelihahmon valmis tekstuuri.

3.2.2 Hahmon luuranko ja riggaaminen

Kun hahmon tekstuuri on valmis, siirrytään seuraavaksi tekemään hahmolle luurankoa. Käyttämällä Alt+X-näppäinkomentoa valittu malli muutetaan läpinäkyväksi, mikä helpottaa luurangon tekoa. Sitten avataan 3ds Maxin ”luu”-työkalu millä luodaan malleille luita. Luita yksinkertaisesti vedetään haluttuihin kohtiin, mutta niitä on hyvä vetää tietyssä järjestyksessä, sillä ihmismäisellä hahmolla luissa tulee olla tietynlainen hierarkia. Lantion alueella oleva luu on se mistä aloitetaan. Luiden hierarkiaa voi muuttaa editorissa, mistä on apua, jos tekee jotakin väärin.

Kun kaikki luut on tehty ja ne ovat oikeassa järjestyksessä (kuva 18), on seuraavaksi vuorossa hahmon riggaaminen. Riggaamisessa ikääkuin kiinnitetään hahmo luihin, ja määritellään kuinka paljon mikäkin luu vaikuttaa mihinkin pisteeseen hahmoa. Kun malli ja luut yhdistetään ja aloitetaan riggaaminen, syntyy valmiiksi jo määritelmiä missä luu vaikuttaa sitä lähimpänä sitä lähimpänä oleviin pisteisiin hahmossa. Valmiit määritelmät ovat usein epätäydellisiä, joten mallintaja joutuu käymään läpi ja säätämään mikä luu vaikuttaa mihinkin. Usein voi tapahtua pieniä virheitä, kuten sellainen, että rintakehän luu vaikuttaa leuassa olevaan pisteeseen tai vastaavaa. Nämä

pitää korjata. Tässä vaiheessa täytyy vain liikutella hahmon eri luita ja katsoa, miten akoko hahmo liikkuu, ja hienosäätää sekä korjat, kunnes asiat näyttävät sopivilta.



Kuva 18. Hahmon luuranko.

3.2.3 Animointi

Kun hahmon riggaus on saatu valmiiksi, siirrytään animoimaan hahmoa. Pelihahmolle tehtiin kaksi animaatiota, yksinkertainen kävelyanimaatio sekä paikallaanolo-animaation. Animointi tapahtuu asettamalla aikajanalle avainkohtia, missä määritellään minkälaisessa asennossa minkäkin kohdan hahmosta on tarkoitus olla, ja niin malli animoituu automaattisesti liikkumaan avainkohdasta toiseen.

Avainkohtia tehdään painamalla ensimmäiseksi Set Key-painiketta, ja tämän jälkeen mallin luita liikutetaan haluttuun asentoon. Kun haluttu asento on saavutettu, valitaan koko malli ja sen kaikki luut ja sen jälkeen painetaan painiketta, missä on avaimen kuva. Tällöin aikajanalle tulee näkymään merkki siitä että siihen on asetettu avainkohta. Tämän jälkeen siirretään liukusäädintä aikajanalla siihen kohtaan mihin halutaan seuraava avainkohta. Halutun kohdan löydyttyä muutetaan mallin asento halutuksi, valitaan malli ja luut ja asetetaan avainkohta. Tätä toistetaan kunnes kaikki halutut avainkohdat on luotu (kuva 19). Kun kaikki avainkohdat on saatu asetettua, painetaan taas Set Key-painiketta, jotta animointiasetus menee pois päältä, ja

animaatio on valmis. Valmista animaatiota voi tarkastella vetämällä alareunan liukusäädintä.

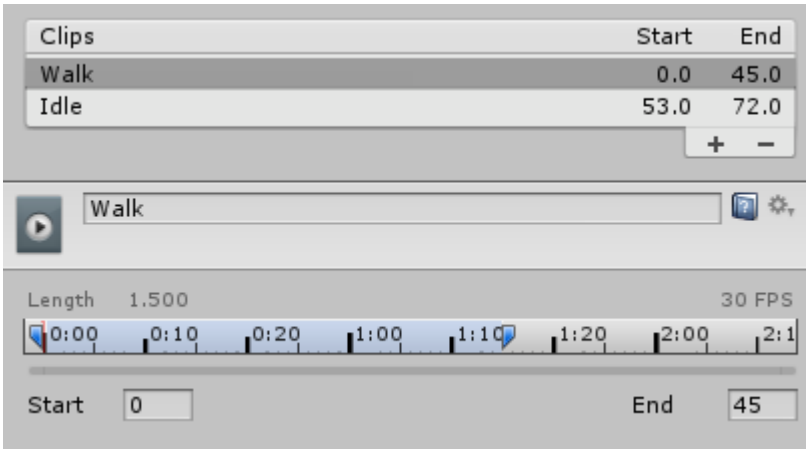
Kuva 19. Animaatio jana missä on määritetty animaatiota.

3.3 Pelin toteutus Unityssä

3.3.1 Hahmojen ja animaatioiden tuonti Unityyn

Kun hahmot ovat valmiita, ne tallennetaan Export-toiminnolla .FBX tiedostomuotoon, ja tämän jälkeen Unityssä kyseiset tiedostot tuodaan projektiin mukaan. Mallien tekstuurit täytyy myös tuoda erikseen mukaan projektiin.

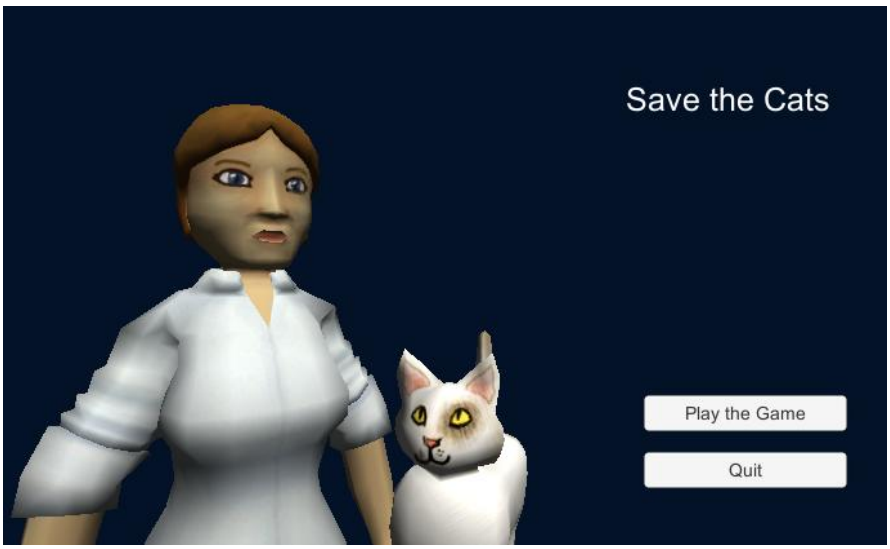
Jos pelihahmon kaikki animaatiot tehtiin yhteen tiedostoon, pitää Unityssä määritellä hahmon animaatiot uudestaan. Tämä tapahtuu valitsemalla tuotu malli tiedostojen katselualueella, jolloin nähdään mallin tuontiasetukset. Näissä asetuksissa on animaatio välilehti, josta löytyy animaatioon liittyvät osiot. Kyseisessä osiossa näkyy kohta Clips, josta löytyvät kyseisen mallin animaatioklipit. Aluksi siinä on klippi, missä on kaikki tehdyt animaatiot yhdessä. Valittaessa klippi, avautuu näkymä, jossa voidaan rajata, missä kohtaa kyseinen animaatio on (kuva 20). Omassa mallissani oli kaksi animaatiota, kävelyanimaatio sekä paikallaanoloanimaatio.



Kuva 20. Animaatioiden muokkaus.

3.3.2 Päävalikko

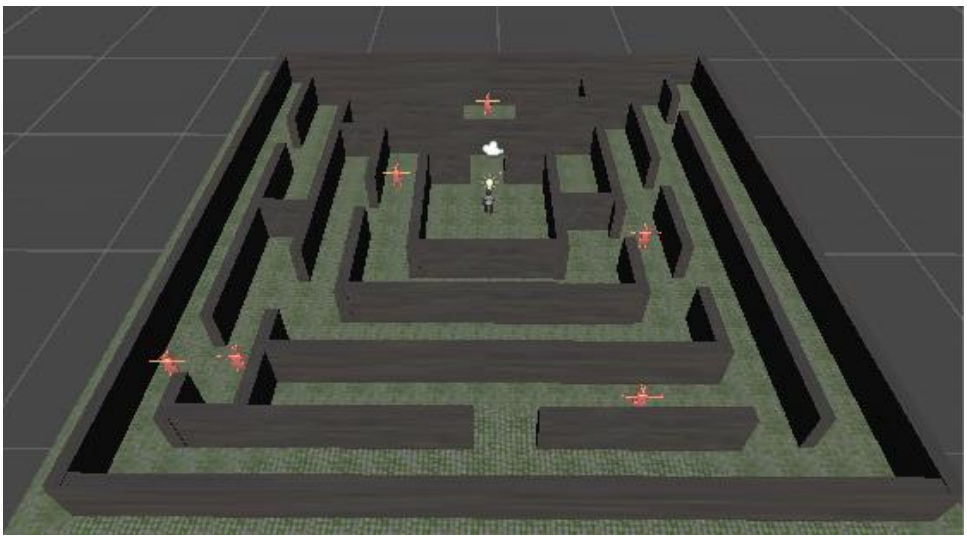
Esimerkkipeliä varten luotiin erittäin yksinkertainen päävalikko käyttäen Unityn UI-työkaluja. Kyseisillä työkaluilla luotiin kaksi painiketta, joista toinen käynnistää pelin ja toinen sulkee pelin, sekä teksti, missä lukee pelin nimi. Päävalikon taustalle laitettiin pelaajan ja kissan hahmot seisomaan, jottei valikkoruutu olisi aivan tyhjä (kuva 21).



Kuva 21. Lopullinen päävalikko.

3.3.3 Kentän luonti

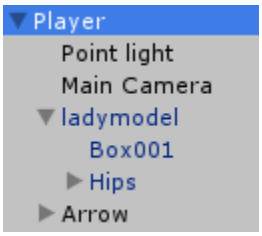
Pelikenttä luotiin käyttäen Unityn laatikkoelementtejä, joille annettiin tekstuurit. Aluksi luotiin yksi iso laatikko pohjaksi, ja seinät tehtiin useista pienemmistä laatikoista (kuva 22). Jotta seinien teko olisi helpompaa, tehtiin seinästä prefab, eli valmiselementti, jossa on valmiiksi tekstuuri ja oikeat mittasuhteet. Prefabeista luotuja seiniä voitiin asetella halutuille kohdille.



Kuva 22. Suorakulmioista luotu kenttä.

3.3.4 Pelaajan hahmo

Pelattavan hahmon malli asetettiin Player nimisen tyhjän peliobjektin lapseksi, ja saman objektin lapseksi asetettiin pelin kamera, valonlähde ja kompassi, jotta ne seuraavat pelaajaa (kuva 23).



Kuva 23. Player peliobjektin hierarkia.

Player objektille lisättiin Capsule Collider- ja Rigidbody-komponentit, sekä PlayerMovement C#-skripti, joka hoitaa pelaajan liikuttamisen sekä muita pelaajaan liittyviä asioita.

FixedUpdate-funktiossa käsitellään hahmon liikuttaminen, se on toteutettu yksinkertaisesti käyttäen AddForce ominaisuutta, joka ikään kuin työntää hahmoa ohjattuun suuntaan. Tämä aiheuttaa sen, että ohjaus on tavallaan hieman liukas, mikä oli haluttu ominaisuus, että peli olisi hankalampi.

OnCollisionEnter-funtioon tullaan, kun hahmo osuu johonkin, ja kyseisessä funktiossa katsotaan, onko osuttu kohde vihollinen. Jos se on vihollinen, niin ohjelma kutsuu muita funktioita jotka huolehtivat osumat viholliseen.

Die-funktiota kutsutaan OnCollisionEnter-funktiosta, kun viholliseen on osuttu. Die-funtiossa pelaaja siirretään takaisin syntymispaikkaan ja luodaan syntymispaikalle partikkeliefekti tallennetusta prefabista. Partikkeliefektille on laitettu Destroy skripti, jossa efekti tuhotaan hetken päästä, jotta se ei jää turhaan olemaan olemassa, kun sitä ei enää tarvita.

OnTriggerEnter-funktiossa käsitellään, mitä tapahtuu, kun pelaaja kerää kissan pelissä. Siellä luodaan uusi kissa satunnaisesti johonkin määritellyistä syntymispisteistä, poistetaan kerätty kissa, sekä lopuksi kutsutaan GameManager-scriptin funktiota, jossa lisätään pistemäärää.

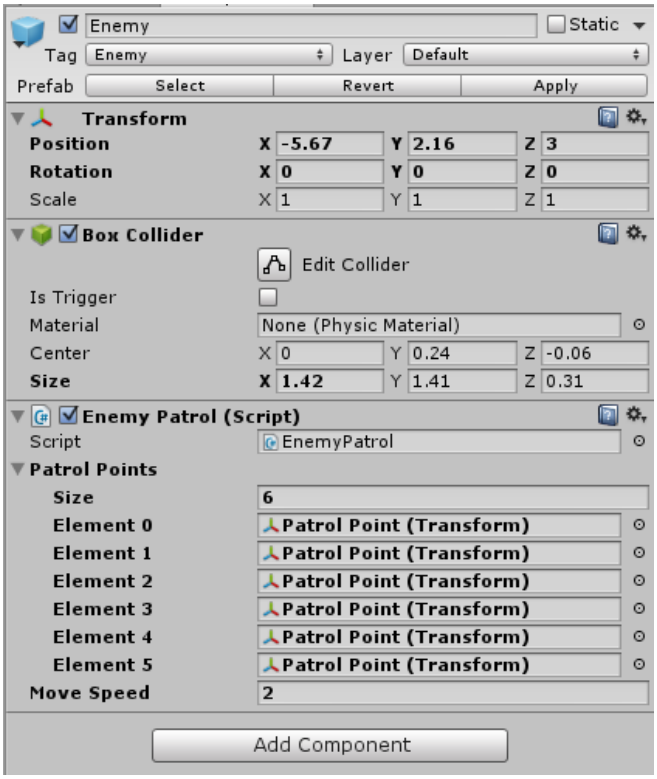
Pelaajaa varten tehtiin myös toinen skripti, CharacterRotate. Kyseisessä skriptissä käännetään hahmoa riippuen siitä, mihin suuntaan hahmoa liikutetaan. Alunperin tätä yritettiin tehdä samassa skriptissä kuin liikkumista, mutta tämä aiheutti ongelmia kameran ollessa kiinni hahmossa. Haluttiin, että kameran kuvakulma pysyi samana. Kun kaikki skriptit olivat kiinni hahmossa, jonka lapsena myös kamera oli, kamera kääntyi hahmon mukana. Tämän takia luotiin kaksi eri skriptiä sekä tyhjä Player peliobjekti.

CharacterRotate skriptissä määritetään myös hahmon animaation status. Koska mallilla on vain kaksi animaatiota, kävely ja paikallaanolo, on animaatioden vaihtumisen ehtona boolean-tyyppinen muuttuja. Skriptissä kyseinen muuttuja laitetaan todeksi, kun ohjauspainikkeita painetaan, ja muissa tilanteissa se on epätosi. Unityn animaationhallinnassa hahmon animaatioiden välille on luotu ehdot, mitkä tarkkailevat missä tilassa kyseinen muuttuja on, ja sen ollessa tosi animaationhallinta toistaa kävelyanimaatiota.

3.3.5 Vihollinen

Vihollisen mallia tallennettaessa ja tuodessa Unityyn oli käynyt jonkinlainen virhe, joka johti siihen, että hahmo on perustilassa väärin kääntynyt. Tämä aiheutti hieman ongelmia, kun vihollishahmoa yritettiin saada kääntymään. Hahmo päätyi makaamaan

maassa, mikä ei ollut haluttua. Onneksi tämä ongelma saatiin ratkaistua luomalla tyhjä peliobjekti Enemy, jonka lapseksi asetettiin vihollisen malli, joka oli käännetty haluttuun asentoon. Tämän jälkeen kyseiselle peliobjektille asetettiin Box Collider-komponentti, Enemy tagi, ja vihollisen skripti (kuva 24), sen sijaan, että ne olisi asetettu suoraan malliin. Tämän jälkeen siitä luotiin prefab.



Kuva 24. Enemy peliobjektin komponentit.

EnemyPatrol skriptissä hoidetaan vihollisen liikkuminen. Kyseisessä skriptissä on muuttujana taulukko, jonka koko ja sisältö määritetään Unityn puolella. Tähän taulukkoon tulee tyhjiä peliobjekteja, jotka on aseteltu pelikentällä kohtiin, joiden kautta vihollinen kulkee. Toisessa muuttujassa määritetään, mikä on tämänhetkisen kohdepisteen numero. Update-funktiossa katsotaan, onko tämänhetkinen sijainti sama kuin kohdepisteen sijainti, ja jos näin on, niin tämänhetkinen kohdepiste vaihdetaan seuraavaan ja lähdetään kulkemaan sitä kohti, paitsi jos kyseessä oli listan viimeinen kohdepiste. Viimeisen kohdepisteen kohdalla vaihdetaan kohdepiste taas ensimmäiseksi kohdepisteeksi, ja hahmo kiertää samaa haluttua rataa yhä uudelleen ja uudelleen. Hahmo myös käännetään katsomaan aina seuraavaa kohdepistettä.

3.3.6 Kerättävä kissa

Kerättävien kissojen 3D-mallin kanssa oli sama ongelma kuin vihollisen mallin kanssa, joten sille juoduttiin myös tekemään samanlainen tyhjä peliobjekti jolle laitettiin skriptit ja muut, ja oikeinpäin käännetty malli asetettiin sen lapseksi. Kissaan lisättiin myös spottivalaisu, jotta se näkyisi paremmin.

Kerättävillä kissoilla ei ollut omaa skriptiä, sillä sen keräämisestä tapahtuvat asiat käsiteltiin muualla, mutta sille asetettiin Cat tagi sekä Box Collider-komponentti. Tämän jälkeen siitä luotiin prefab.

3.3.7 GameManager

GameManager on peliin luotu tyhjä peliobjekti, mihin liitettiin GameManager niminen skripti. Kyseisessä skriptissä on toteutettu laskuri ja pisteenlasku sekä niiden näyttäminen ruudulla. Lisäksi skriptissä on toteutettu pelin loppuessa päävalikkoon palaaminen.

Update-funktiossa tapahtuu laskurin laskeminen, sekä siellä tarkastellaan, onko laskuri mennyt nollaan. Jos aika on loppunut, peli lataa päävalikon.

CatCollected-funktiota kutsutaan PlayerMovement-funktiosta, kun pelaaja saa kerättyä kissan, ja siinä tehdään loput kissan keräämiseen liittyvät asiat. Kun funktioon tullaan niin se kasvattaa pistetilannetta yhdellä, sekä lisää aikaan 30 sekuntia.

EnemyHit-funktiota kutsutaan myös PlayerMovement-funktiosta, kun viholliseen on osuttu. Tässä funktiossa vähennetään 10 sekuntia ajasta, paitsi jos aikaa on vähemmän kuin 10 sekuntia, sillon aika vähenee yhteen sekuntiin.

OnGUI-funktiossa toteutetaan ajan ja pistetilanteen näkyminen ruudulla, molemmat ovat label tyyppisiä UI elementtejä.

4. YHTEENVETO

Opinnäytetyötä tehtäessä oppi paljon uusia asioita peleihin mallintamisesta. Aiemmin en ollut juurikaan miettinyt mallinnusta. Se oli selvää, että tehokkaammalla alustalla

mallit voivat olla yksityiskohtaisempia. Oli varsin mielenkiintoista tutkia peleihin mallintamisen ja elokuviin mallintamisen eroja sekä erilaisia oikoteitä, mitä pelientekijät käyttävät saadakseen pelit näyttämään hyviltä.

3ds Max on mielenkiintoinen myös ammattilaisten käyttämä ohjelma, joten siihen oli mukava tutustua. Toisaalta se herätti myös paljon kysymyksiä, sillä loppujen lopuksi käytin suurimmaksi osaksi vain sen perustyökaluja. Ohjelmaa käytettäessä melkein kaikissa valikoissa näkyi monia asioita, mitä minun ei tarvinnut käyttää, mutta ne herättivät kysymyksiä siitä, että millaisissa tilanteissa niitä oikein käytetään. Ehkä tutustun ohjelmaan vielä lisää myöhemmin.

Se jäi ärsyttämään, että tein vihollisen ja kissan 3D-mallien unityyn tuomisessa jonkun virheen, joka johti mallien ympärikääntymiseen. En ymmärrä mistä se johtui, sillä pelaajan hahmo toimi oikein ja tein ja toin ne omasta mielestäni samalla tavalla.

Unityä oli mukava käyttää, se on käyttäjäystävällinen ja ohjeita löytyy paljon. En ollut aiemmin käyttänyt Unityä paitsi jo kauan sitten olleella koulukurssilla. Alkukankeudesta pääsi kuitenkin ohi suhteellisen nopeasti.

Lopputuloksena saavutin sen, mitä oli tavoitteena, sain luotua 3D- malleja, joista yhdellä oli animaatio, sekä sain luotua Unityllä pelin, joka toimi. Olen ihan tyytyväinen siihen sen ollessa ensimmäinen kokonainen Unity projektini.

Itse opinnäytetyön kirjoittaminen oli hankalampaa. En ole hyvä kirjoittamaan asiatekstiä, joten varsinkin työn kirjoitus oli haaste. Myös oma persoona oli esteenä, koska en ole hyvä omatoimisessa työskentelyssä. Opinnäyteyö aiheutti paljon ahdistusta, mikä myös hidasti etenemistahtiani. Tämä johti siihen että tekeminen venyi varsin pitkäksi.

LÄHTEET

1. Slick, J. Anatomy of a 3D Model. Saatavissa: http://3d.about.com/od/3d-101-The-Basics/a/Anatomy-Of-A-3d-Model.htm [viitattu: 10.4.2015]
2. Pardew, L. & Whittington, D. 2005. Beginning Game Art in 3Ds MAX 8. Thomson Course Technology.
3. Masters, M. 2014. What's the Difference? A Comparison of Modeling for Games and Modeling for Movies. Saatavissa: http://blog.digitaltutors.com/whats-the-difference-a-comparison-of-modeling-for-games-and-modeling-for-movies/ [viitattu: 10.4.2015]
4. Silverman, D. 2013. 3D Primer for Game Developers: An Overview of 3D Modeling in Games. Saatavissa: http://gamedevelopment.tutsplus.com/articles/3d-primer-for-game-developers-an-overview-of-3d-modeling-in-games--gamedev-5704 [viitattu 10.4.2015]
5. MacGillivray, C. & Head, A. 2005. 3D for the Web: Interactive 3D Animation Using 3ds Max, Flash and Director. Taylor & Francis.
6. Graft, K. 2010. GDC: Infamous' Open World Trickery. Saatavissa http://www.gamasutra.com/php-bin/news_index.php?story=118581 [viitattu 10.4.2015]
7. History of Autodesk 3ds Max. 2010. Saatavissa: http://area.autodesk.com/maxturns20/history [viitattu 10.4.2015]
8. FAQ: Licensing & Activation. Saatavissa: http://unity3d.com/unity/faq [viitattu 10.4.2015]
9. Brodkin, J. 2013. How Unity3D Became a Game-Developement Beast. Saatavissa: http://news.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/ [viitattu 10.14.2015]

Liite 1. Renderöidyt 3D-mallit

Liite 2/1. PlayerMovement.cs

```
using UnityEngine;
using System.Collections;
public class PlayerMovement : MonoBehaviour {
		public GameManager manager;
		public Transform[] spawnPoints;
		public GameObject cat;
		public float moveSpeed;
		public GameObject deathEffect;
		public GameObject catEffect;
		private float maxSpeed = 4f;
		private Vector3 startPoint;
		private Vector3 input;
		// Use this for initialization
		void Start () {
				startPoint = transform.position;
				manager = manager.GetComponent<GameManager>();
				int spawnPointIndex = Random.Range (0, spawnPoints.Length);
				Instantiate(cat, spawnPoints[spawnPointIndex].position, Quaternion.identity);
		}
		// Update is called once per frame
		void FixedUpdate () {
				//Liikutetaan hahmoa
				input = new Vector3(Input.GetAxisRaw ("Horizontal"),
						0, Input.GetAxisRaw ("Vertical"));
				if(rigidbody.velocity.magnitude < maxSpeed)
				{
					rigidbody.AddForce(input * moveSpeed);
				}
		}
```

Liite 2/2.

```
        void OnCollisionEnter(Collision other)
        {
                if (other.transform.tag == "Enemy")
                {
                        Die ();
                        manager.EnemyHit();
                }
        }
        void OnTriggerEnter(Collider other)
        {
                if (other.transform.tag == "Cat")
                {
                        Instantiate(catEffect, transform.position, Quaternion.identity);
                        int spawnPointIndex = Random.Range (0, spawnPoints.Length);
                        Instantiate(cat, spawnPoints[spawnPointIndex].position,
                        Quaternion.identity);
                        Destroy(other.gameObject, 0);
                        manager.CatCollected();
                }
        }
        void Die()
        {
                transform.position = startPoint;
                Instantiate(deathEffect, transform.position, Quaternion.identity);
        }
}
```

Liite 3/1. CharacterRotate.cs

```
using UnityEngine;
using System.Collections;
public class CharacterRotate : MonoBehaviour {
		Animator anim;
		// Use this for initialization
		void Start () {
			anim = GetComponent<Animator> ();
		}
		// Update is called once per frame
		void Update () {
			//Katsoo mihin suuntaan hahmo katsoo sekä määrittää liikutetaanko hahmoa animaatiocontrolleria varten
			if (Input.GetKey ("left") || Input.GetKey ("a"))
			{
				transform.eulerAngles = new Vector3 (0,-90,0);
				anim.SetBool("moveStatus", true);
			}
			else if (Input.GetKey ("right") || Input.GetKey ("d"))
			{
				transform.eulerAngles = new Vector3 (0,90,0);
				anim.SetBool("moveStatus", true);
			}
			else if (Input.GetKey ("up") || Input.GetKey ("w"))
			{
				transform.eulerAngles = new Vector3 (0,0,0);
				anim.SetBool("moveStatus", true);
			}
```

```
                else if (Input.GetKey ("down") || Input.GetKey ("s"))
                {
                        transform.eulerAngles = new Vector3 (0,180,0);
                        anim.SetBool("moveStatus", true);
                }
                else
                {
                        anim.SetBool("moveStatus", false);
                }
        }
}
```

Liite 4. EnemyPatrol.cs

```
using UnityEngine;
using System.Collections;
public class EnemyPatrol : MonoBehaviour {
		public Transform[] patrolPoints;
		public float moveSpeed;
		private int currentPoint;
		// Use this for initialization
		void Start () {
				transform.position = patrolPoints[0].position;
				currentPoint = 0;
		}
		// Update is called once per frame
		void Update () {
		if (transform.position == patrolPoints[currentPoint].position)
				{
						currentPoint++;
				}
		if (currentPoint >= patrolPoints.Length)
				{
						currentPoint = 0;
				}
				transform.position = Vector3.MoveTowards(transform.position,
				 patrolPoints[currentPoint].position, moveSpeed * Time.deltaTime);
				transform.LookAt(patrolPoints[currentPoint],transform.up);
		}
}
```

Liite 5/1. GameManager.cs

```
using UnityEngine;
using System.Collections;
public class GameManager : MonoBehaviour {
	public int score;
	public Rect timerRect;
	public Rect countRect;
	public GUISkin skin;
	public float startTime;
	private string currentTime;
	private string countText;
	// Use this for initialization
	void Start () {
	}
	// Update is called once per frame
	void Update ()
	{
		startTime -= Time.deltaTime;
		currentTime = string.Format("{0:0.0}",startTime);
		if (startTime <= 0)
		{
			startTime = 0;
			Application.LoadLevel("menu");
		}
	}
	public void CatCollected()
	{
		score++;
		startTime += 30;
	}
```

```
	public void EnemyHit()
	{
		if (startTime > 10)
		{
			startTime -= 10;
		}
		else
		{
			startTime = 1;
		}
	}
	void OnGUI()
	{
		GUI.skin = skin;
		GUI.Label (timerRect, currentTime);
		GUI.Label (countRect, "Cats: " + score);
	}
}
```

Liite 6. ArrowRotate.cs

```
using UnityEngine;
using System.Collections;
public class ArrowRotate : MonoBehaviour {
		public Transform target;
		// Use this for initialization
		void Start () {
		}
		// Update is called once per frame
		void Update () {
				target = GameObject.Find("Cat").transform;
				transform.LookAt(target.transform.position, transform.up);
		}
}
```

Liite7. Destroy.cs & Menu.cs

Destoy.cs:

```
using UnityEngine;
using System.Collections;
public class Destroy : MonoBehaviour {
		public float lifetime = 0;
		// Use this for initialization
		void Start () {
				Destroy (gameObject,lifetime);
		}
}
```

Menu.cs:

```
using UnityEngine;
using System.Collections;
public class Menu : MonoBehaviour {
public void LoadLevel()
		{
				Application.LoadLevel ("level");
		}
public void QuitGame()
		{
				Application.Quit();
		}
}
```