Antti Lehesvuori

PELIPROTOTYYPIN KEHITYS UNITY-PELIMOOTTORILLA

Tietojenkäsittelyn koulutusohjelma

2018

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

PELIPROTOTYYPIN KEHITYS UNITY-PELIMOOTTORILLA

Lehesvuori, Antti
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Lokakuu 2018
Ohjaaja: Kyngäs, Jari
Sivumäärä: 44
Liitteitä: 0

Asiasanat: Peliala, Ohjelmistokehitys, Ohjelmointi.

Tämän opinnäytetyön tarkoituksena oli lähestyä pelin suunnittelua ja toteutusta pelimoottorin avulla. Pelimoottoreiden tutkinnan lähtökohtana oli omien taitojen kartoitus ja sen perusteella valitun Unity pelimoottorin käyttö.

Työn edetessä tutkimukseen otettiin mukaan myös itsenäinen ns. Indie-pelinkehitys, joka on saanut suurta suosiota viime vuosina pelimoottoriteknologian siirtyessä vapaille markkinoille, ja ilmaisten ratkaisujen yleistyessä.

Opinnäytetyön haluttuna tuloksena oli peliprototyyppi, jonka luontiprosessia kuvattiin askel askeleelta. Prototyyppi luotiin käyttäen erilaisia opetusmateriaaleja; tärkeimpiä näistä olivat Sebastian Laguen, Asbjørn Thirslundin (Brackeys) ja Unity Technologies:in tarjoamat opetusvideot.

DEVELOPING A GAME PROTOTYPE WITH UNITY GAME ENGINE

Lehesvuori, Antti
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Data Processing
October 2018
Supervisor: Kyngäs, Jari
Number of pages: 44
Appendices: 0

Keywords: Game-industry, Software development, Programming.

The purpose of this thesis was to approach the design and execution of a game as a beginner with the use of a game engine.
Game engines were examined based on research of beginner skill level and was carried through using the selected Unity game engine.

Study also included the so called indie game development industry, which has grown in the past years due to the game engine technology entering free market, and the spread of free solutions.

The desired outcome of the thesis was to create a MVP (Minimum viable product)-type game prototype, and its creation process is explained step by step.
In the creation of this project, different educational materials were used, most important of which were the tutorial videos created by Sebastian Lague, Asbjørn Thirslund (Brackeys) and Unity Technologies.

SISÄLLYS

1 JOHDANTO

Nykyään pelinkehitys on siirtynyt pelimoottoreiden kehityksen ja yleistymisen ansiosta suurten julkaisijoiden piiristä vapaille markkinoille, mikä on aiheuttanut runsasta kasvua julkaistujen pelien määrässä. Uusien ideoiden lisääntyminen pelinkehityksen piirissä mahdollistaa suuremman yleisön saavuttamisen.

Pelimoottori on alati kehittyvä konsepti, jonka tarkoitus on muuttunut pelin välttämättömien osien ajamisesta myös pelin sisällön luomiseen. Niiden kehitystä nopeuttaa runsas kilpailu pelialalla.

Työn taustalla oli kiinnostus pelinkehitykseen, ja siihen liittyviin työkaluihin. Mielenkiintoni taustalla on pitkä peliharrastus, jonka ansiosta olen oppinut runsaasti erilaisia taitoja; itselleni tärkeintä on ollut englanninkielen taidon kehitys varhaisnuoruudessa englanninkielisten pelien avulla. Pelit ovat mielestäni tärkeitä työkaluja oppimiseen ja arjen ohella rentoutumiseen. Ne mahdollistavat mahdottomien asioiden kokemisen, kun rajana on vain mielikuvitus. Lähdin tutkimaan pelinkehitykseen liittyviä haasteita, ja oppimaan enemmän itse pelinkehitys-prosessista. Tämän innoittamana aloitin ensimmäisen pelinkehitysprojektini, jossa käytin apuna erilaisia verkosta löytyviä opetusmateriaaleja.

Projektin tuloksena on pieni, muutaman perusmekaniikan roolipeliprototyyppi, jonka päälle voi lähteä rakentamaan syvällisempiä mekaniikkoja, ja testata niiden toimivuutta. Tämä projekti tulee toimimaan omassa käytössäni olevana testiympäristönä, ja mahdollisuutena oppia lisää pelinkehityksestä.

2 PELISUUNNITTELUSTA

Ennen pelin luomista on suunniteltava sen perusta, eli vastattava kysymyksiin; miksi peli luodaan, mikä on sen päämäärä ja minkälainen pelistä tulee. Peliä suunnitellessa pitää muistaa, ettei kannata lähteä luomaan liian isoa ja vaativaa kokonaisuutta. Lähtötilanteessa tulee ottaa huomioon tämänhetkisen osaamisen lisäksi omat heikkoudet ja vahvuudet, ja lähteä rakentamaan suunnitelmaa rajoitukset huomioon ottaen. Varsinkin ensimmäisen pelin luonnissa kannattaa aloittaa yksinkertaisella ja pienellä päämäärällä.

Ensimmäiset peliprojektit toimivat niin sanottuna harjoituksena, joten ei kannata odottaa heti suurta menestystä, vaan tähdätä minimalistiseen toimivaan kokonaisuuteen. (Floyd & Portnow 2015a.)

Pelisuunnittelijalta vaaditaan useita taitoja: artistista näkemystä, tarinankerrontaa ja kykyä rakentaa näistä kokonaisuuksia. Näiden kykyjen avulla voi tehdä visiostaan totta. Nykymarkkinoilla on tärkeää tehdä pelikokemuksesta viihdyttävä ja mukaansatempaava, jotta peli selviää äärimmäisen kovassa kilpailussa. (Crosby 2008.)

Pelien luonti itsenäisesti on prosessi, joka koostuu epäonnistumisista ja niiden kautta oppimisesta. Tästä syystä ei saa alussa odottaa itsestään liikoja, ja muistaa pitää pää kylmänä vastoinkäymisten edessä. Pelin luomisessa tarvitaan taidon lisäksi myös itsekriittisyyttä, joka auttaa kysymään apua muilta, enemmän osaavilta ihmisiltä. Nykyään on helppo löytää vastauksia internetistä ja sitä kautta on mahdollista kartuttaa osaamistaan ilman suurta tutkimustyötä. Suuremmat tehtävät kannattaa jakaa pienempiin osiin, jotta niiden saavuttaminen ei tuntuisi mahdottomalta. (Floyd & Portnow 2015b.)

Peliprototyypin suunnittelussa käytetään usein termiä ”Minimum viable product” (MVP) joka tarkoittaa käytännössä pienintä mahdollista kokonaisuutta, joka tuottaa hyödyllistä dataa valmistuessaan. MVP:n hyödyntäminen on tärkeää pelin perustuksia rakennettaessa, koska leikkaamalla turhia ominaisuuksia prototyypistä, sen ongelmien hahmottaminen on paljon helpompaa. Myöhemmin ominaisuuksia voi tietenkin lisätä,

mutta alussa tärkeintä on yksinkertaisimpien toimintojen sujuva toiminta. (Floyd & Portnow 2015c.)

Suunnittelusta toteutukseen siirtyminen on usein suuri askel, sillä lähestymistapoja on runsaasti, ja on vaikeaa valita juuri oikeat välineet ilman suurempaa kokemusta. Aloittelijan onneksi saatavilla on useita pelimoottoreita ja muita työkaluja, joista jokainen pyrkii yksinkertaistamaan pelin luontiprosessia.

3 PELIMOOTTORI

Pelimoottorin konsepti on yksinkertainen, sen tehtävänä on erottaa tietyt pelin toimintaan liittyvät yksityiskohdat, esimerkiksi renderöinti (grafiikan piirto peliruudulle), fysiikan mallinnus ja komentojen käsittely, itse pelin sisällön luomisesta. Pelimoottori sisältää näihin tehtäviin tarvittavat työkalut, ja näin automatisoi niiden sisältämisen peliin.

Pelimoottorit sisältävät osia, joita muuntelemalla peli saadaan toimimaan halutusti. Animointi, tarvittavien komponenttien lataus, osien välinen interaktiivisuus ja jopa tietyt tekoälyyn vaaditut toiminnot voivat kaikki olla pelimoottorin osia, joita ei tarvitse itse alkaa rakentamaan. Pelimoottoria voidaan kutsua työkalupakiksi, ja näiden osien oikeanlainen yhteistoiminta on riippuvainen pelin tekijästä. (Ward 2008.)

3.1 Pelimoottoreiden historiaa

Pelimoottori-termi sai alkunsa 1993, kun id Softwaren DOOM pelisarjan ensimmäinen osa julkistettiin, ja sen mukana mainittiin termi ”DOOM engine”. Termi kuvasi id Softwaren rakentamaa ja käyttämää teknologiaa, josta tuli uudistava tekijä tietokonepeliteollisuudessa. Pelaajat alkoivat muuntelemaan id:n aikaisemmin julkaisemia pelejä hakkeroimalla omaa sisältöään kyseisiin peleihin; tämä teki vaikutuksen id:n päähenkilöihin John Romeroon ja John Carmackiin, jonka seurauksena id Softwaren pääasiallisesta kehityskohteesta tuli pelimoottorin jalostaminen sisällön luontia mahdollistavana työkaluna. (Lowood 2014.)

Pelimoottoreiden lyhyessä historiassa on yksi tekijä ollut ajankohtainen useassa eri tilanteessa; teknologisen kehityksen suuren kasvun ongelmallisuus. Kun laitteistosta tulee entistä monimutkaisempaa ja voimakkaampaa, jäävät ohjelmistoratkaisut vääjäämättä jälkeen. Konsoleiden siirryttyä tehokkaisiin moniytimisiin prosessoreihin vuonna 2004 (Ps3, Xbox360) olivat aiemmin julkaistut pelimoottorit välittömästi vanhanaikaisia. Yksikään tuon aikakauden pelimoottoreista ei ollut arkkitehtuuriltaan sellainen, että se olisi hyödyntänyt uutta teknologiaa tehokkaasti, eikä näin ollen pystynyt ottamaan siitä kaikkea hyötyä irti. Vasta viiden vuoden ohjelmistokehityksen

jälkeen julkaistiin pelejä, jotka toivat esiin uuden laitteiston tehot ja ominaisuudet. (Normann 2016.)

3.2 Pelimoottorin valinta

Oikean pelimoottorin valinta on tärkeä päätös, joka vaikuttaa koko prosessiin. Valinta tulee tehdä projektikohtaisesti ja osanottajien taitojen perusteella. Seuraavaksi tutkin joitain tämän hetken suosituimpia moottoreita.

Unity pelimoottoria hyödyntäviä pelejä ovat muunmuassa CUPHEAD, Firewatch, Pokemon Go ja Ori and the Blind Forest.

Unity on sulava ja helppokäyttöinen pelimoottori, jonka avulla on helppo hallinnoida peliprojektia voimakkaasti alusta lähtien. Tämän lisäksi Unity on ilmainen pelimoottori, jolla voi julkaista pelejä tietyille alustoille ilman lisenssiä. Unity:llä on mahdollista kehittää 2D- ja 3D- pelejä samalla käyttöliitymällä ilman suurempia muutoksia. (O'Flanagan 2015.)

Unity:lla on markkinoiden suurin yhteensopivuus, ja sillä on mahdollista tuottaa pelejä jopa 25 eri alustalle. (Unity Technologies 2018.)

Unityn avulla luodut pelit kattoivat joulukuussa 2017 34% 1000 suosituimman mobiilipelin listasta, ja se on suosituin ”mainstream” pelimoottori mobiilipelien kehitykseen. Se on myös yleistyvien VR-pelien suosituimpia pelimoottoreita, ja sillä on luotu 90% Samsung Gear VR:n ja 53% Oculus Riftin peleistä. (Elhady 2017.)

Unreal Engine 4 pelimoottoria hyödyntäviä pelejä ovat muunmuassa Ark: Survival Evolved, Dead by Daylight, FFVII: remake, Gears of War 4.

Unreal Engine 4 (UE4) on tämän hetken voimakkain saatavilla oleva pelimoottori, mutta se on tarkoitettu selvästi ammattilaiskäyttöön vaikeamman käyttöliittymänsä perusteella. UE4 on ollut käytössä nykyajan suurimpien ja vaikuttavimpien pelien

luonnissa. Pienemmille pelinkehittäjille UE4 ei kuitenkaan sovellu hyvin. (O'Flanagan 2015.)

UE4 omaa suppean yhteensopivuuden, koska se keskittyy tärkeimpiin alustoihin. (Epic Games 2018).

UE4 on saanut Guinness World Recordsin palkinnon "Menestynein pelimoottori" (Elhady 2017).

Game Maker Studio 2 pelimoottori on käytössä muunmuassa Undertale, Hotline Miami, Hyper Light Drifter ja Turmoil peleissä.

Game Maker Studio (GMS) on tunnettu pelialaan perehdyttävänä työkaluna, GMS on yksinkertainen 2d-pelimoottori, joka on ollut suuressa suosiossa harrastelijoiden keskuudessa. Sillä on luotu useita suosittuja pelejä, joista ei yksinkertaisen työkalun käyttö näy ulospäin. Tämä pelimoottori toimii suureksi osaksi ennalta määrätyillä tapahtumilla, eikä tarvitse hyvää ohjelmointitaitoa. (O'Flanagan 2015.)

O'Flanagan painottaa GMS:än helppokäyttöisyyttä ja aloittelijaystävällisyyttä vertauksessaan.

GMS:än tavoite on murtaa seinät peliteollisuuteen, tarjoten jokaiselle halukkaalle mahdollisuuden pelien luontiin ilman erikoistaitoja. (YoYo Games Ltd 2018).

3.3 Tarkemmin Unity- pelimoottorista

Tässä työssä keskityn Unity-pelimoottorin käyttöön, joten perehdyn tarkemmin sen toimintoihin ja ominaisuuksiin.

Unity pitää itseään tehokkaana työkaluna, oli lähtökohtana pelin taiteellinen puoli, puhdas logiikan koodaus, tai mitä tahansa siltä väliltä.

Taiteelliseen työhön Unity tarjoaa reaaliaikaisen ”Cinemachine” työkalun, jolla on mahdollista luoda näyttävää filmaattista sisältöä ilman teknologian osaamista. Se toimii suurilta osin yksinkertaisella käyttöliittymällä, mutta sen käyttäytymiseen on mahdollista myös vaikuttaa tarkasti näin halutessaan. Unityn taiteilijoille tarjoamina työkaluina mainitaan internet sivustolla myös jälkikäsittelyyn erikoistuva ”Post-Processing Stack”, joka mahdollistaa tarkan kontrollin lopputuotteen laatuun. (Unity Technologies 2018b.)

Pelin logiikan suunnittelu ja toteutus on yksinkertaista helppokäyttöisellä ”Drag&Drop”-periaatteella toimivalla editorilla, jolloin tiettyjä toiminnallisuuksia on mahdollista asettaa objektille muutamalla hiiren vedolla, jos haluttu koodi on jo olemassa.

Unity tukee natiivisesti kahta ohjelmointikieltä: C#:a ja UnityScriptiä. UnityScript on Unitylle suunniteltu kieli, jonka pohja on JavaScript- kielessä. Unity tarjoaa MonoDevelop- ja Visual Studio community-skriptieditorit asennuspakkauksensa yhteydessä. (Unity Technologies 2018c.)

Yksi suurimmista eteen tulevista esteistä pelinkehityksen yhteydessä on hyvän tekoälyn luominen, mutta Unityn sisäänrakennetut koodipohjat auttavat tässäkin tapauksessa. Unityn kanssa käytettävät koodieditorit saavat automaattisesti Unityn ennalta asetetut luokkakirjastot käyttöönsä; tämä mahdollistaa muun muassa tekoälyyn tarvittavien luokkien kokoamisen erittäin helposti.

Unity sisältää monia pelin luomista helpottavia komponentteja, joista tärkeimpiä mainittavia ovat fysiikan mallinnukseen käytettävät moottorit, tekoälyyn tukeutuva navigointijärjestelmä, ja kehittyneet lopputuotteen optimointiin erikoistuvat työkalut. (Unity Technologies 2018d.)

Koska Unity toimii käsi kädessä Blender 3D-mallinnusohjelman kanssa, on mahdollista tuottaa yhteensopivia 3D-malleja ja animaatioita tehokkaasti.

4 INDIE-PELINKEHITYS

Päädyin tutkimaan indie-pelinkehitystä, koska tämä opinnäytetyö on tehty käytännössä indie- kehittäjän näkökulmasta, vaikka julkaistavaa peliä ei lopputuloksena ole.

4.1 Mitä on Indie-pelinkehitys?

Itsenäinen pelinkehitys, eli suurista julkaisijoista irrallinen, pienellä tiimillä ja budjetilla toimiva pelinkehitysprosessi, tunnetaan yleisesti nimellä Indie-pelinkehitys. Se on ollut suosiossa viime vuosina. Varsinkin pelimoottoriteknologian levitessä suurilta pelistudioilta kenen tahansa käytettäväksi mahdollistui Indie-kehityksen suuri kasvu.

Indie-kehityksen yksi suuri hyöty on irrallisuus välikäsistä, jolloin lopputuotteeseen ei tarvitse tehdä rahoittajien vaatimia muutoksia, ja peli valmistuu juuri oman näkemyksen mukaisesti. (Gril 2018.)

Indie- termi on moniosainen, ja sen merkityksestä ollaan usein montaa eri mieltä. Sillä voidaan tarkoittaa vain itsenäistä rahoitusta, mutta myös kaikkien kehitykseen liittyvien osien itsenäistä tuottamista. Indie- peli on ennen kaikkea innovatiivinen ja uusi kokemus pelaajalle, sen takana on yleensä intohimoinen tekijä, eikä ole vain rahantuottoväline suurelle pelifirmalle. (Gril 2008.)

Indie-pelien yleisö vaihtelee radikaalisti pelin mukaan. Gril mainitsee neljä tutkimuksissaan yleisintä eteen tullutta ryhmää. (Gril 2008.)

1. Kulttipelaaja
   -Aktiivinen
   -Pelialan julkaisuista hyvin tietävä
   -Etsii erilaisia, massojen kaihtamia pelejä

2. Kasuaalinen pelaaja
   -Pelaa pelejä internetissä, ja muuten helposti tavoitettavissa olevilla alustoilla.
   -Joka ikäluokasta löytyy ihmisiä, jotka pelaavat enimmäkseen ajanviettotarkoituksessa

3. Entinen hardcore-pelaaja
   -Runsaasti pelejä elämässään pelannut
   -Mahdollisesti ammattilaistasolla
   -Jostain syystä päätynyt vähempään pelaamiseen (perhesyyt jne.)
   -Tietoinen pelien kehityksestä ja markkinoista

4. Pelialan ammattilainen tai akateeminen pelaaja:
   -Pienin osa yleisöstä kuuluu tähän kategoriaan
   -Pelin tulee olla todella vaikuttava, jotta alan ammattilaiset kiinnostuvat siitä.

4.2 ”Indiepocalypse”

Indie-kehittäjien huulilla on viime aikoina ollut ”Indiepocalypseksi” kutsuttu ilmiö. Indiepocalypse tarkoittaa käytännössä menestyvän pelin luomisen vaikeutta indie -pelien saralla, nyt kun pelejä on tarjolla koko ajan enemmän. Ryan Clark kuitenkin on sitä mieltä, että indiepocalypse on myytti. (Clark 2015.)

Indiepocalypsen todellisuudesta on ollut eri mieltä myös Adam Coster, joka tutkimuksissaan hyödynsi SteamSpy- ohjelman tarjoamaa dataa, joka sisältää kaikkien Steam pelien käyttäjätietoja. Costerin tutkimukset osoittivat korrelaatiota julkaistujen pelien määrän lisääntymisen ja useamman pelin huonon menestymisen välillä, mutta lopulta päätyi tulokseen, että huonosti menestyneiden pelien laatu oli pääasiallinen syy myynnin puutteeseen. (Coster 2015.)

”Indiepocalypse” keskittyy viiteen pääideaan:

4.2.1 Liikaa kilpailua

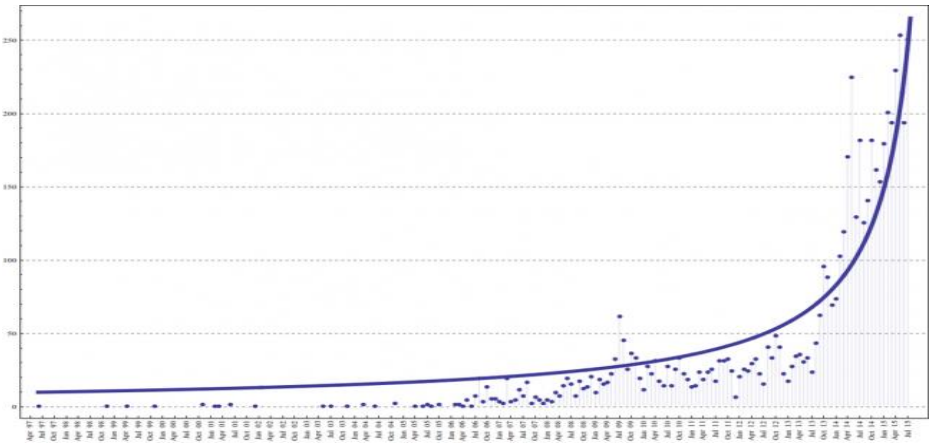
Steam on yksi suurimmista digitaalisista pelimarkkinoista, jonka hiljattainen ”Steam Greenlight” ohjelma on antanut pienille julkaisijoille mahdollisuuden päästä suuren yleisön markkinoille, antamalla yleisölle valtaa valita mitä pelejä Steamissa on myytävänä. Tämä on aiheuttanut huolta pienten kehittäjien keskuudessa, sillä se on tuonut runsaasti ns. piilossa olleita pelejä kilpailuun mukaan. (Clark 2015.)

Clark painottaa, että kilpailu on aina ollut yhtä laajaa, Greenlight on vain tuonut kilpailun kehitysprosessin eri vaiheeseen. Aikaisemmin kilpailu on ollut Steamiin pääsemisestä, kun nykyään se on siirtynyt Steamin sisäiseksi. (Clark 2015.)

Costerin mukaan pelin kuin pelin menestyksen avain on sen laatu ja markkinointi; kumoten väitteet saturaation suuresta vaikutuksesta indie- myynnin heikentymiseen. (Coster 2015.)

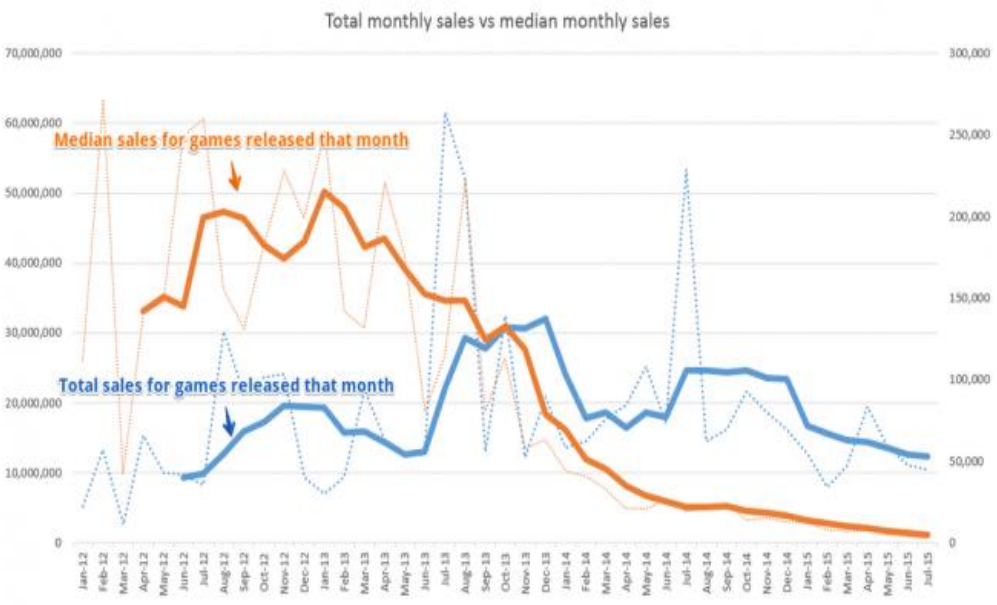
4.2.2 Taulukot kertovat

Kuva 1 kertoo pelien julkaisumäärän runsaasta noususta Steamissa. Vaikka tämä voi aluksi näyttää pelottavalta, pitää muistaa, että suuri osa nousukaaren peleistä on aikaisemmin julkaistu muualla, ja jälkeenpäin tuotu Steamiin sen helppouden takia. Tämä trendi ei myöskään voi jatkua loputtomiin, sillä eteen ei tule tilannetta, jossa jokainen ihminen planeetalla julkaisee pelejä Steamissa. (Clark 2015.)



Kuva 1. Steamissa kuukaudessa julkaistut pelit (Clark 2015.)

Kuvan 2 oranssi viiva kertoo myytyjen kopioiden määrän julkaisukuukautena. Julkaistujen pelien määrän kasvu selittää trendin laskun, sillä kun tarkastelun kohteena olevaan joukkoon lisätään aikaisemmin julkaistuja pelejä, ja runsaasti samanlaisia pelejä, on trendin lasku odotettavissa. (Clark 2015.)



Kuva 2. Kuukauden kokonaismyynti vs. mediaanimyynti. (Clark 2015.)

Hyvälaatuisten indie- pelien määrä suhteessa julkaistujen pelien kokonaismäärään on pysynyt vakaana, kuten on myös indie- pelien kokonaisomistus; laskevan trendin syynä on Costerin mukaan runsas määrä pelejä, joita ihmiset eivät halua ostaa. (Coster 2015.)

4.2.3 Mobiilialustan saturaatio

Mobiilialustat ovat täynnä ilmaisia, ja melkein identtisiä kopioita toisistaan olevia pelejä avoimuutensa ja helppokäyttöisyytensä takia; tästä syystä monet pelkäävät tietokonemarkkinoiden avautumisen johtavan myös tietokoneen pelimarkkinoiden ylisaturaatioon. Clark ei näe mobiilialustojen avoimuutta syynä niiden saturaatioon. (Clark 2015.)

Mobiilialustojen näennäiset ongelmat ovat sen matala hintataso, huonolaatuiset pelit ja pelien menestyksen määrittäminen latauskertojen mukaan. Clarkin mielestä syynä näihin ongelmiin on suurimmaksi osaksi yleisö. Mobiilipelien kuluttajia ovat enimmäkseen ihmiset jotka omistavat älypuhelimen ja haluavat tekemistä ajankulutukseen. Tällöin laadulla ei ole niin suurta merkitystä kuin tietokonemarkkinoilla, jonka yleisö pelaa pelejä harrastuksena. (Clark 2015.)

4.2.4 Hiljattaiset Indie-pettymykset

Korkean profiilin Indie-pelien huono menestys julkaisussa on huolettanut useita julkaisijoita ja mediaa, luoden ”Indiepocalypselle” pohjaa, mutta Clarkin mielestä syynä tähän on aivan muut tekijät. Tilastollisesti kyseessä on ollut hyvin pieni ilmiö, ottaen huomioon nopean julkaisutahdin ja monen pienemmän julkaisun runsaan menestyksen. Korkean profiilin indie-pelit ovat myös lisääntyneet viime vuosina, joten on luonnollista, että suurimmat nimet eivät menesty odotettavasti. (Clark 2015.)

4.2.5 Korkealaatuisten Indie-pelien (III) budjettien räjähtävä kasvu

Viimeisen kahdenkymmenen vuoden aikana korkealaatuisten (AAA) pelien vaadittava budjetti on noussut fenomenaalisesti, ja tämä antaa aihetta uskoa, että myös Indie-pelit tulevat kokemaan samanlaista nousua. Vaaditun alkupääoman nousu tulee vaikeuttamaan aloittelevien, pienen budjetin pelintekijöiden kilpailua suuremmilla markkinoilla.

Edellä mainittu budjettien kasvu johtuu suureksi osaksi teknologian kehityksen aiheuttamasta kustannusnoususta, joka pääasiassa vaikuttaa graafista etulyöntiasemaa hakevien pelien kustannuksiin. Suurin osa indie-peleistä lähestyy kilpailua innovaatioillaan, eivätkä kiinnitä niin suurta huomiota teknologiseen paremmuuteen, joten vaikka kustannuksen tulevat lisääntymään, ei se ole niin huomattavaa, kuin suurten julkaisijoiden kustannusten nousu. (Clark 2015.)

Lopuksi Clark kertoo indie-kehittäjän työn olevan rankkaa ja riskialtista, mutta vaikka kilpailu alalla onkin kovaa ja raskasta, on se myös palkitsevaa ja antaa hyvän mahdollisuuden toteuttaa itseään. Clark pitää indiepocalypseä ylimääräisenä pelotteena, jonka ei tulisi antaa vaikuttaa päätökseen ryhtyä pelien kehittäjäksi.

5 HAHMON LUONTI JA ANIMAATIO BLENDER-3D MALLINNUSOHJELMALLA

Työssä käytettiin pohjana Sebastian Laguen, Asbjørn Thirslundin (Brackeys) ja Unityn tarjoamia opetusvideoita ja niiden avulla rakennettuja koodikokonaisuuksia.

Kehittämässäni peliprototyypissä on ihmishahmoinen pelaaja- ja vastustajahahmo, joiden mallinnus ja animointi tapahtui Blender 3D- mallinnusohjelmalla. Blender on ilmainen, aloittelijaystävällinen ja helppokäyttöinen 3D-mallinnusohjelma, johon on tarjolla runsaasti opetusmateriaalia Blenderin nettisivuilla ja muualla internetissä. Blenderiä käytetään suureksi osaksi pikanäppäimillä, joka on mielestäni sen suurin oppimiskynnys aloittelijoille. Kuitenkin pienen harjoittelun jälkeen on käyttö sulavaa ja helppoa.

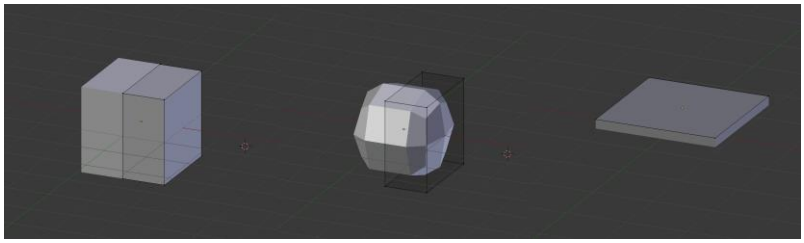
5.1 Työssä käytetyt Blenderin työkalut

Seuraavaksi käyn lyhyesti läpi useimmin työssä käyttämäni työkalut.
Muuntimet ovat aluksi simulaatiomoodissa joka näyttää visuaalisen representaation sen tuloksista, kuitenkaan muuttamatta objektia ennen sen vahvistamista. Muuntimien vahvistus tehdään yleensä työn ollessa valmis, jotta ei tapahdu odottamattomia virheitä.

Mirror-Peilimuunnin mahdollistaa symmetrisen mallinnuksen kopioimalla muodon suhteessa valittuun akseliin; tämän työkalun oikean toiminnan edellytyksenä on työstää vain puolikasta objektia, eli on hyvä poistaa valitun akselin toisella puolella olevat verteksit. (Kuva 3.)

Subdivision Surface eli ”SubSurf”- työkalu simuloi mallin pintojen jakamista pienempiin, näin antaen mallille pehmeämmän pinnan. Työkalu toimii parhaiten lähes valmiin mallin kanssa, sillä jos työstettävä muoto on liian yksinkertainen, on tuloksena runsaasti pienempi objekti. (Kuva 3.)

Solidify-työkalu paksuntaa pintoja, tehden esimerkiksi vaatteista realistisemman näköisiä. Kuitenkaan tätä ei tule käyttää runsaasti liikkuvien mallien kanssa, sillä runsaat muutokset aiheuttavat pintojen leikkausta toisten pintojen kanssa. (Kuva 3.)



Kuva 3. Mirror-, SubSurf- ja Solidify-muuntimet järjestyksessä.

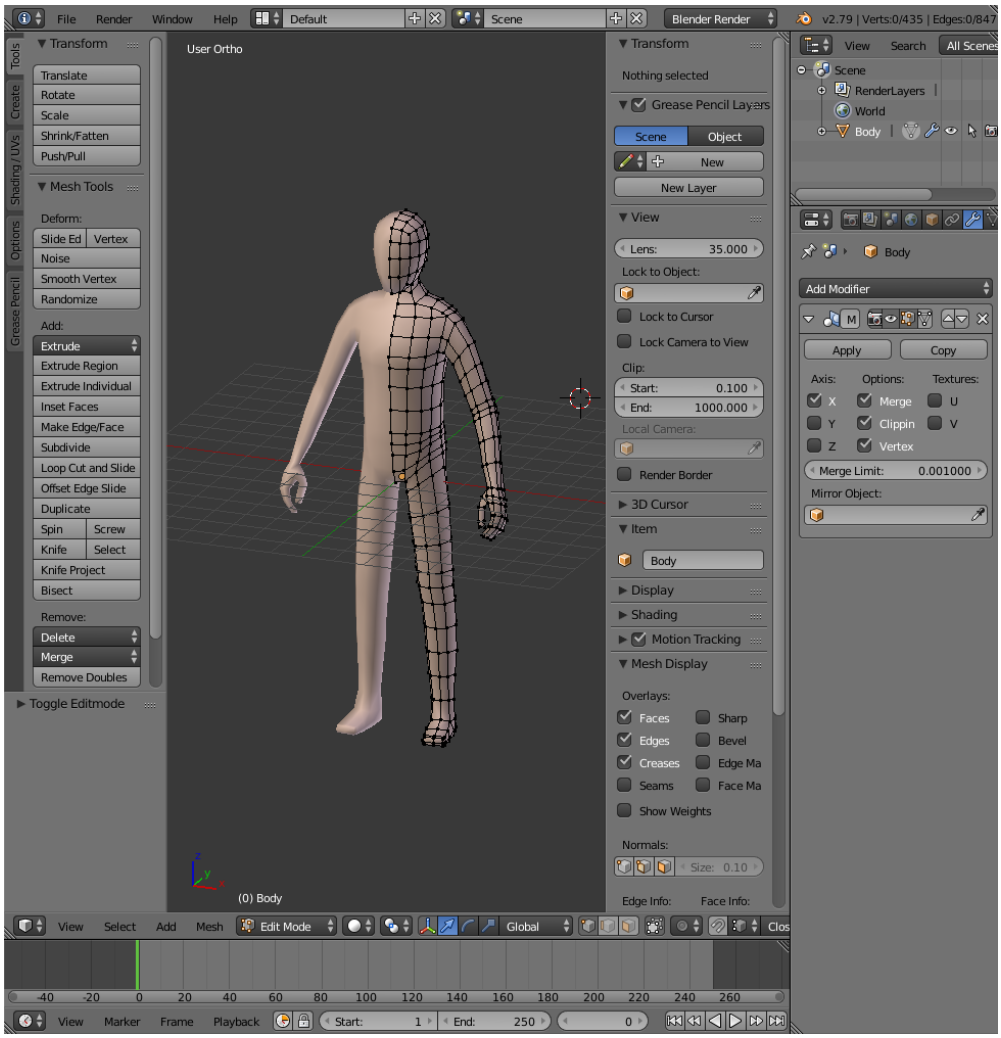
5.2 Ihmishahmon mallinnus

Mallinnus tapahtuu pääasiassa yksinkertaisten geometristen objektien muuntelulla, ja niihin kuuluvien suorien leikkauspisteiden siirtelyllä, kunnes haluttu muoto ollaan saatu aikaiseksi. Aluksi mallinsin yksinkertaisen ihmismuodon peilimuuntimen avulla, joka varmistaa symmetrisen muodon saavutuksen. (Kuva 4.)

Kuva 4. Ihmishahmo alkutekijöissään.

Blender sisältää runsaasti automaattisia muuntimia, jotka auttavat lopullisen hahmon saavuttamisessa esimerkiksi pehmentämällä valmista muotoa, ja paksuntamalla litteitä pintoja tehden niistä inhimillisemmän näköisiä. Aluksi hahmoa tulee muunnella kuitenkin manuaalisesti leikkaustyökalun avulla, ja lisäämällä muutosta vaativille

alueille silmukoita, sillä automaattiset työkalut eivät toimi halutusti näin alkukantaisen hahmon kanssa. Myös materiaalin lisäys auttaa hahmottamaan miten haluttu muoto voidaan saavuttaa. Hahmon kädet, pää ja jalat muotoiltiin erillisistä kuutioista, jotka liitettiin aluksi mallinnettuun ruumiiseen. Tässä vaiheessa on hyvä ottaa huomioon animointivaiheessa liikkuvat nivelet, ja lisätä lisäsilmukoita paremman animaation saavuttamiseksi. (Kuva 5.)

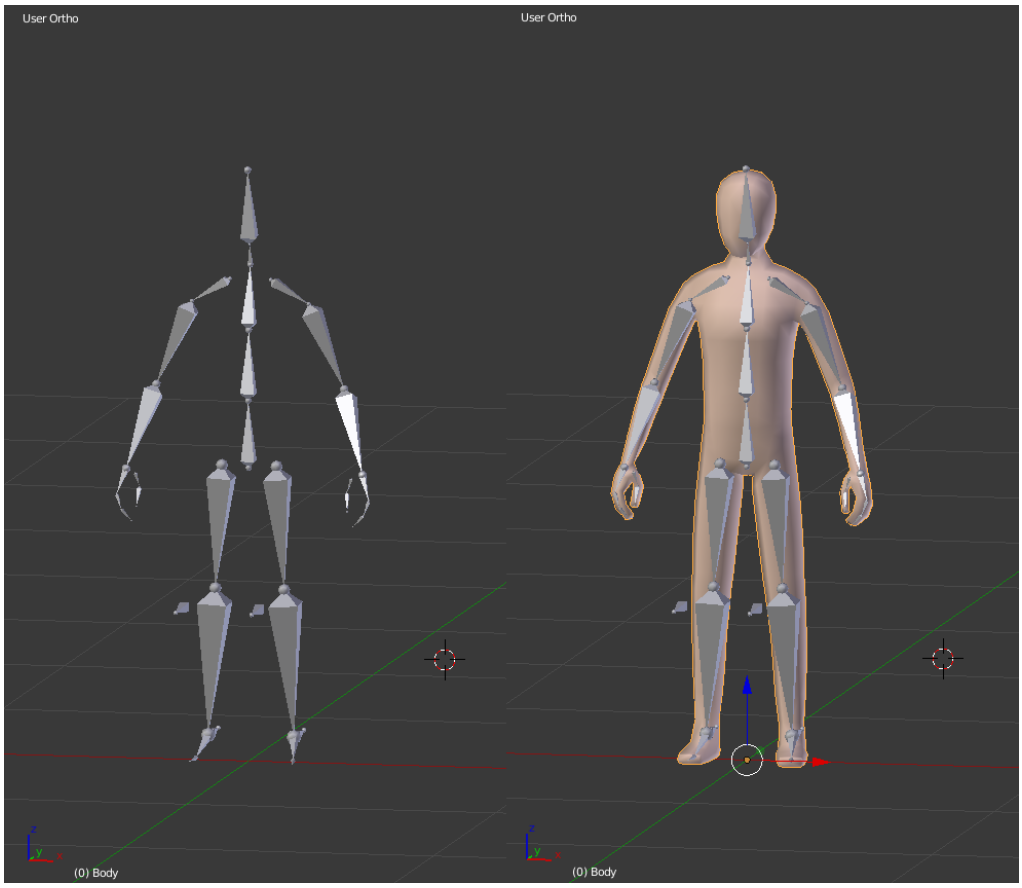


Kuva 5. Ihmishahmo ottaa muotoaan.

Ennen animointia on hyvä luoda suuntaa antavat vaatteet hahmolle, jotta animaatiotilanteessa nähdään mahdolliset päällekkäisyydet jotka johtuvat mallin venymisestä tai kutistumisesta sen osien liikkuessa. Näin voidaan korjata jo animointivaiheessa viat, joista voi seurata runsaasti päänvaivaa.

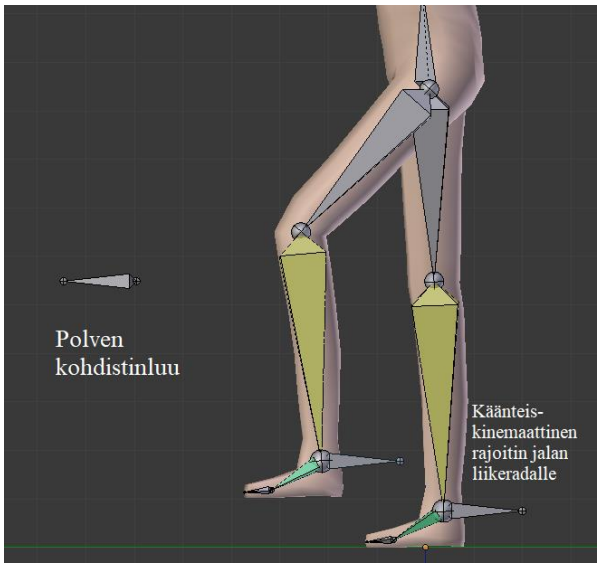
5.3 Ihmishahmon animointi

Minkä tahansa hahmon animointi alkaa luomalla hyvä animaatioluista koostuva luuranko, jolla mallin eri osia liikutellaan. Ihmishahmon ollessa kyseessä luiden sijainti on lähellä oikeaa luurankoa. Luiden sijaintien tulee olla äärimmäisen tarkkoja, jotta mallin muoto pysyy oikeanlaisena. (Kuva 6.)



Kuva 6. Animaatioluuranko yksin ja mallinnetun hahmon kanssa.

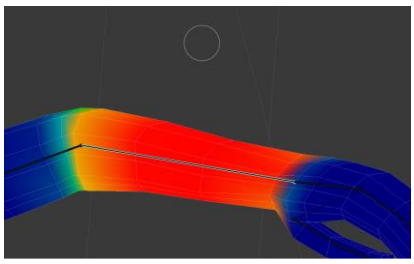
Mallia liikuttavien luiden lisäksi animointia helpottaa ns. ohjausluiden käyttö, esimerkiksi polvien suuntaamisessa käytetyt kohdistinluut pitävät huolen, että polvet eivät taivu taaksepäin jalkaa liikutettaessa. Myös koko jalan luonnollista liikerataa voidaan emuloida ohjausluiden ohessa kinemaattisten rajoittimien avulla; tämä mahdollistaa myös jalan liikuttamisen yhden kantapääluun avulla (Kuva 7.)



Kuva 7. Jalan ohjausluut ja kinemaattinen rajoitin.

5.4 Luiden vaikutusalueiden määrittely

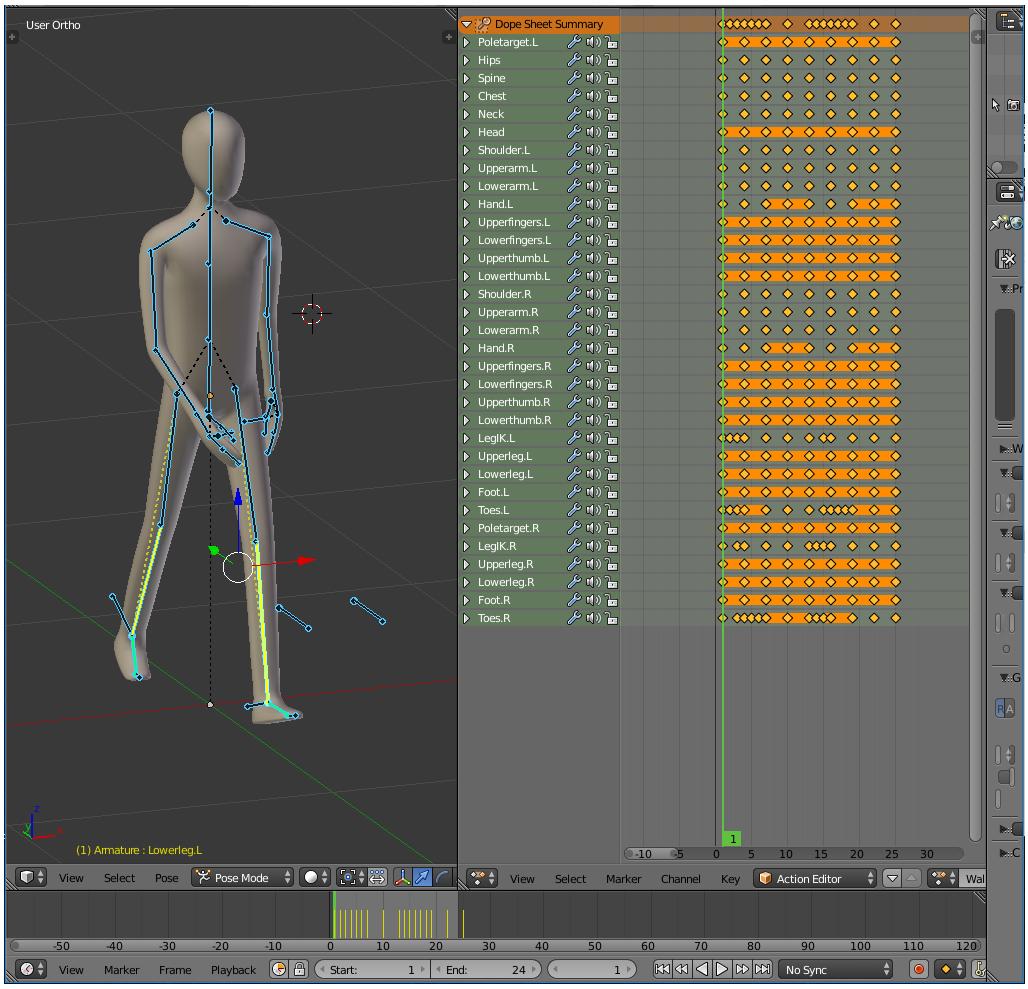
Luiden painoarvot tarkoittavat sitä, missä suhteessa mallin osat liikkuvat luita liikuteltaessa, ja suuri osa animaation pehmeydestä riippuu oikeanlaisesta vaikutusvallan ja -alueen suhteesta. Yksi Blenderin työkaluista mahdollistaa automaattisten vaikutusalueiden määrittelyn, ja kuten muidenkin automaattisten työkalujen käytön kanssa, on tärkeää hienosäätää tuloksia jälkeenpäin; tämä tapahtuu painomaalaukseksi kutsutulla prosessilla, jolla voidaan itse nostaa ja laskea luun vaikutusta yhteen leikkauspisteeseen kerrallaan. Kuvassa vaikutuksen vahvuus skaalautuu punaisesta siniseen, eli vahvasta heikkoon. (Kuva 8.)



Kuva 8. Painomaalaus käytännössä.

5.5 Animointiprosessi

Animointiprosessi alkaa määrittämällä luiden sijainnit avainkehyksiin, tämän jälkeen luodaan tietyin aikavälein haluttuja asentoja, joiden välillä luut liikkuvat sulavasti asennosta toiseen. Kävelyanimaatiosta saadaan sulava luomalla ensin puolet animaatiosta, jonka jälkeen se kopioidaan ja liitetään animaation perään käänteisesti. (Kuva 9) Luotujen avainkehysten välissä olevia automaattisia siirtymiä voidaan muokata lisäämällä luukohtaisia avainkehyksiä tarvittaviin kohtiin. Blenderissä on mahdollista animaation ajaminen sitä muokatessa, jolloin on helppo nähdä korjausta tai hienosäätöä vaativat avainkehykset. Luiden piirtotyyppiä voidaan vaihtaa yksinkertaisempaan, jolloin nähdään animaatio tarkemmin. (Kuva 9.)



Kuva 9. Animointi-ikkuna.

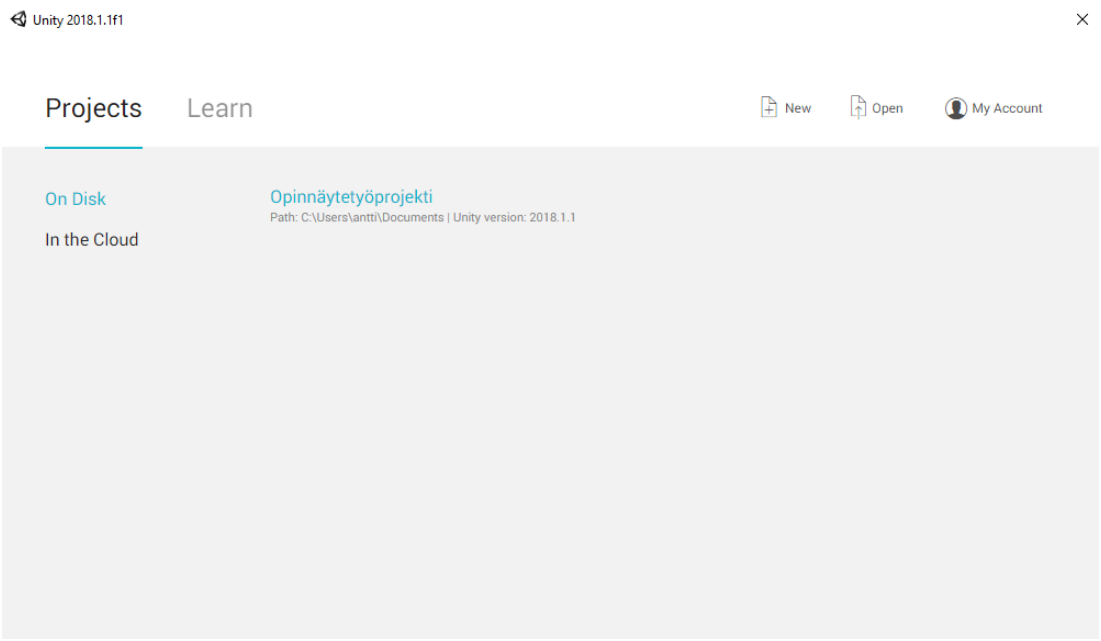
5.6 Lisävarusteiden liittäminen animaatioihin

Pelin aiheesta riippuen, tulee hahmolle mallintaa eri tarvikkeita. Esimerkiksi keskiaikaisen roolipelin ollessa kyseessä, minun tuli mallintaa useita aseita, kuten miekka, kirves ja kilpi. Näiden liittäminen jo valmiiseen animaatioon on yksinkertaista, ja tapahtuu lisäämällä yksi luu käteen, jossa asetta pidetään. Kun luu kopioidaan halutun sijainnin omaavasta luusta, on sen vanhempi sama, kuin sillä josta se kopioitiin, jolloin sen sijainti on riippuvainen vanhemman sijainnista, eikä liikeratoja tarvitse muokata erikseen. Kyseessä oleva esine liitetään juuri luotuun luuhun, ja varmistetaan sen sulava liikkuminen käymällä avainkehykset läpi. Aseiden mallintamisen lisäksi on tärkeää luoda myös muutama eri hyökkäysanimaatio, jotta taistelu ei visuaalisesti toista itseään.

Kuva 10. Valmis pelaajahahmo.

6 PELIPROJEKTIN LUONTI UNITY-PELIMOOTTORILLA

Peliprojektia aloitettaessa luotiin yksinkertainen pelialue muutamilla korkeuseroilla, jotta Unityn polunetsintätyökalun toimintaa voitiin esitellä.
Unity- ohjelman aloitusikkunassa tulee luoda uusi, tai valita jo tallennettu projekti, joka tallentaa kopion kaikista käytetyistä tiedostoista projektin omaan kansioon. Aloitusikkunassa on myös linkkejä opetusmateriaaleihin ja tilikohtaiseen pilvivarastoon. (Kuva 11). Skriptauskielenä oli käytössä C#, ja koodieditorina Microsoft Visual Studio.

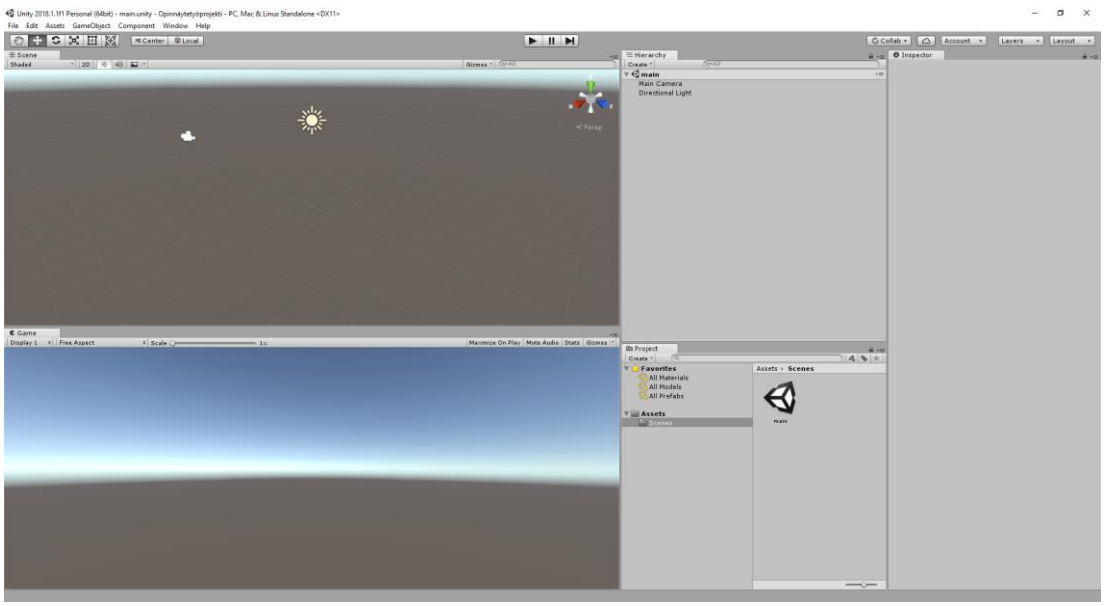


Kuva 11. Unityn aloitusikkuna

Projektia luotaessa siihen on mahdollista liittää joko itse luotu asset-paketti, tai ladata sopiva paketti Unityn tarjoamasta asset-storesta. Unityn asset-storesta löytyy maksullisia ja maksuttomia paketteja, jotka voivat sisältää useita projektin kannalta hyödyllisiä tiedostoja, esimerkiksi hahmo- tai muita malleja, ääniefektejä ja skriptejä. Tässä projektissa ei kuitenkaan käytetty valmiita paketteja, vaan luotiin omat tiedostot eri opetusmateriaalien avulla.

6.1 Unityn käyttöliittymä

Unityn käyttöliittymää on mahdollista järjestellä täysin niin kuin haluaa, jokainen osa on liikuteltavissa ja niiden kokoa on mahdollista muuttaa. Alla olevassa kuvassa on vasemmalla peli-ikkuna, joka näyttää kameran näkymän kameraobjektin perspektiivistä, ja scene-ikkuna, joka mahdollistaa objektien tutkimisen muunneltavasta näkymästä. Keskellä on hierarkia- ja projekti- ikkunat, jotka pitävät sisällään vastaavasti aktiiviset objektit ja projektikansiossa olevat objektit. Oikealla, inspector, eli tarkasteluikkunassa näkyvät valitun objektin yksityiskohdat, ja niihin liitetyt skriptit (Skriptillä tarkoitan koodikokonaisuuden osaa, joka luo haluttavan toiminnon). (Kuva 12).

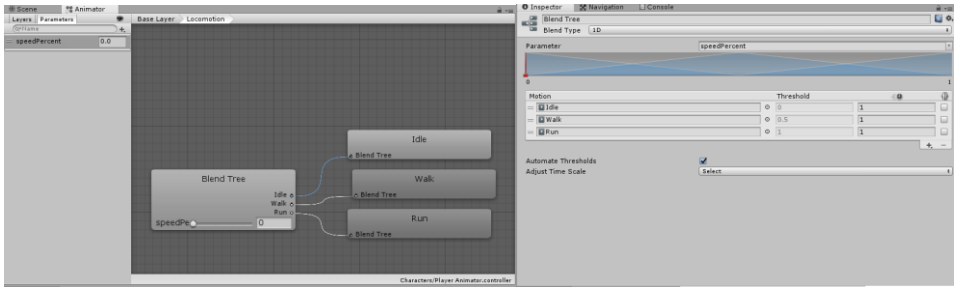


Kuva 12. Unityn käyttöliittymä

6.2 Pelaajahahmon integrointi projektiin

Pelaajahahmolle tulee luoda pohja, johon voidaan liittää aikaisemmin mallinnettu player- tiedosto; tämä tuodaan Unityyn kopioimalla se projektin kansioon, jonka jälkeen sen mukana tuotujen animaatioiden oikea toiminta voidaan varmistaa. Pohjaan liitetään myös projektin aikana luodut tarvittavat skriptit ja muut halutun toiminnan mahdollistavat osat. Toimiakseen halutusti, Blenderissä luodut animaatiot tarvitsevat erillisen animaatio-ohjaimen, joka mahdollistaa animaation sitomisen hahmon

liikkumisnopeuteen. Sulava animaatio kävely- ja juoksuanimaatioiden välillä tapahtuu hahmon nopeuden kiihtyessä. (Kuva 13).

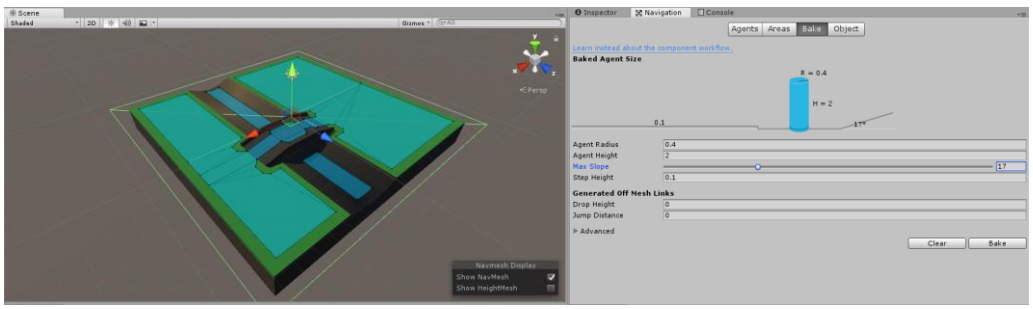


Kuva 13. Locomotion- animaatiopuu kontrollerissa.

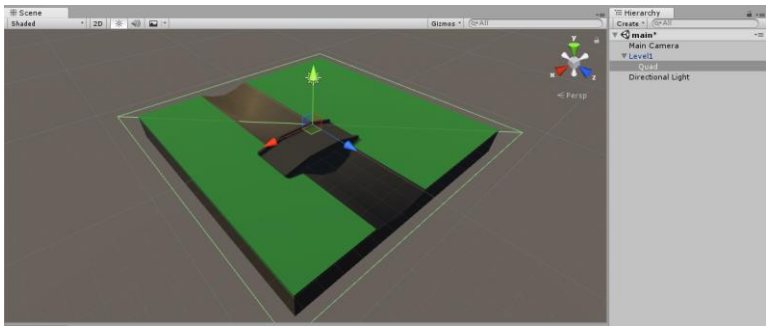
Animaatiokontrollerin lisäksi tarvitaan myös skripti, joka keskustelee animoitavan hahmon, ja animaatiokontrollerin välillä, ja kontrollerin kanssa yhteistoiminnassa toistaa animaation.

6.3 Pelaajahahmon ja kameran liikuttaminen

Peliprojektin luonti tulee aloittaa yksinkertaisimpien mekaniikoiden luomisesta, jotka tässä tapauksessa ovat liikkumiseen liittyvät skriptit ja kameran asettaminen hahmoa seuraavaksi objektiksi. Hahmon liikuttamiseen vaaditaan polunetsintätyökalulla alustettu pelialue (Kuva 14.) jossa tulee olla ”RayCasting” (kamerasta luotu viiva, joka tunnistaa hiiren klikkauksen osumakohdan) mekaniikkaa tukeva pinta; tämä voidaan varmistaa asettamalla pelialueen pinnalle näkymätön polygoniverkko. (Kuva 15). Liikkumisen sallimiseksi alustetulla alueella, mutta ei esimerkiksi koristeeksi tarkoitettujen objektien päällä, tulee alueet asettaa walkable- tilaan navigaatioasetuksissa.



Kuva 14. Sinisellä alustettu alue ja asetukset.



Kuva 15. Pelialueen RayCastin tunnistava polygoniverkko.

6.3.1 Pelaajahahmon liikuttaminen

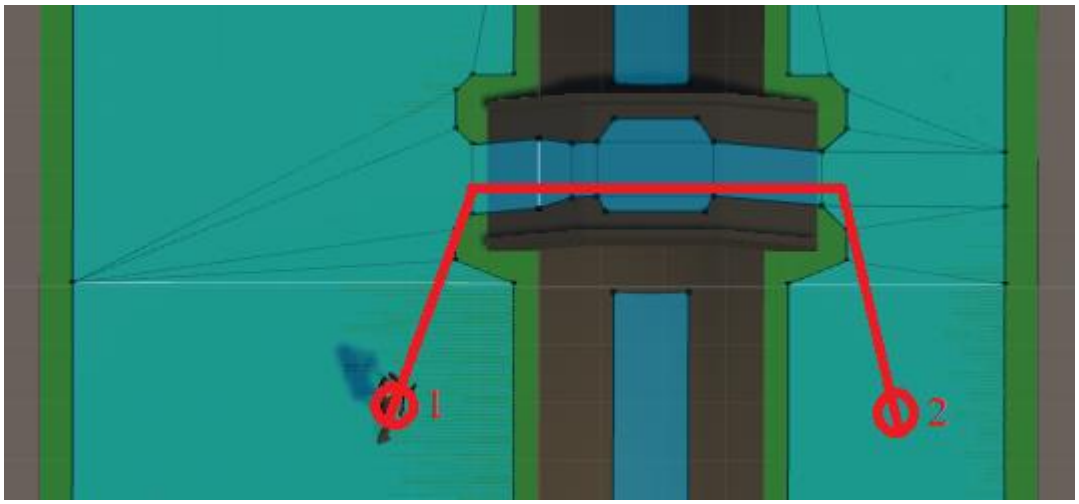
Pelaajahahmon liikuttaminen tapahtuu skripteillä, ja jotta niiden yhteistoiminta polunetsintätyökalun kanssa toimii halutusti, tarvitsee pelaajahahmo NavMeshAgent, navigaatioagentti- komponentin; tähän viitataan koodissa nimellä agent. Skripteihin tehdään muutoksia prosessin aikana aina, kun tarvitaan viittauksia projektin muihin osiin, joten niiden helppolukuisuus on äärimmäisen tärkeätä. Hahmon liikuttamisen pohjana toimii PlayerController- skripti, joka tunnistaa napinpainallukset ja RayCast-objektin sijainnin, joiden avulla se kutsuu muita skriptejä, joissa määritetään haluttu toiminta tapahtuvaksi. Koodin kirjoittamisen aikana on hyvä hyödyntää Debug.Log toimintoa, joka kertoo konsolissa virheistä ja toiminnallisuuden puutteista.

(Luodaan RayCast objekti: ”ray” hiiren sijaintiin, ja tallennetaan sen osumakohta: hit; tämän onnistuessa kerrotaan konsolissa osumakohta, ja käsketään PlayerMotorin liikuttaa pelaaja sijaintiin. Lopuksi poistetaan myös mahdollinen keskitetty objekti) (Koodi 1.)

```
if (Input.GetMouseButtonDown(0))
        {
            Ray ray = cam.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;

            if (Physics.Raycast(ray, out hit, 100, movementMask))
            {
                Debug.Log("We hit " + hit.collider.name + " " + hit.point);
                motor.MoveToPoint(hit.point);
                RemoveFocus();
            }
        }
```

Koodi 1. (Lague & Thirslund 2018.)

PlayerController- skriptin lisäksi pelaajan liikuttamiseen tarvitaan PlayerMotor-skripti, joka sisältää pääasiassa PlayerControllerin kutsumia toimintoja ja kontrolloi aikaisemmin mainittua navigaatioagentti- komponenttia; sen toiminnallisuus perustuu Unityyn sisältyvän Unity.AI- kirjaston käyttöön. Skriptin alustuksessa haetaan navigaatioagentti komennolla GetComponent<NavMeshAgent>(); tällä toiminnolla koodissa voidaan kutsua skriptin sisältävän objektin eri osia. Navigaatioagentti mahdollistaa lyhimmän polun etsimisen päämäärään. (Kuva 16.)



Kuva 16. Lyhin mahdollinen matka pisteestä 1 pisteeseen 2.

(Metodit, joilla liikutetaan pelaajahahmoa, asetetaan hahmo seuraamaan klikattua objektia, lopetetaan seuraus ja asetetaan pelaaja katsomaan objektia.) (Koodi 2.)

```
public void MoveToPoint (Vector3 point)
    {
            agent.SetDestination(point);
    }
public void FollowTarget(Interactable newTarget)
    {
            agent.stoppingDistance = newTarget.radius * 0.8f;
            agent.updateRotation = false;
            target = newTarget.interactionTransform;
    }
public void StopFollowingTarget()
    {
            agent.stoppingDistance = 0f;
            agent.updateRotation = true;
            target = null;
    }
void FaceTarget()
    {
            Vector3 direction = (target.position - transform.position).nor
            malized;
```

```
            Quaternion lookRotation = Quaternion.LookRotation(new Vector3(di-
            rection.x, 0f, direction.z));

            transform.rotation = Quaternion.Slerp(transform.rotation, lookRo-
            tation, Time.deltaTime * 5f);
    }
```

Koodi 2. (Lague & Thirslund 2018.)

6.3.2 Kameran liikuttaminen

Kameran liikuttamiseksi luodaan CameraController- skripti, joka sisältää kameran toimintoihin, kuten zoomaamiseen ja kääntämiseen liittyvät parametrit, ja tässä työssä sen kiinnityskohteen, eli pelaajan, sijainnin saavan target- muuttujan. Skriptissä asetettavat näkyvyysmääreet kertovat mitä kaikkia parametrejä voidaan muokata suoraan Unityn editorissa (public), ja mitä taas täytyy muokata suoraan koodissa (private). Unity määrittää oletusarvoiset kontrollit projektin alussa, joten näitä voidaan kutsua tietyillä nimillä koodissa; kameran kääntämiseen käytetään näppäimistön A ja D kirjaimia, joita kutsutaan Input.GetAxis(”Horizontal”); komennolla.

(Update- metodissa käydään läpi komennot, jotka tapahtuvat reaaliaikaisesti, kuten kameran zoomaaminen hiiren rullalla, ja kääntäminen A ja D näppäimillä. LateUpdate- metodia kutsutaan jokaisen Updaten jälkeen, ja sen käyttö on pääasiassa automaattisille toiminnoille, kuten kameran sijainnin määrittämiseen sen seuratessa pelaajaa.) (Koodi 3.)

```
void Update()
    {
        currentZoom -= Input.GetAxis("Mouse ScrollWheel") * zoomSpeed;
        currentZoom = Mathf.Clamp(currentZoom, minZoom, maxZoom);

        currentYaw -= Input.GetAxis("Horizontal") * yawSpeed * Time.deltaTime;
    }
void LateUpdate()
    {
        transform.position = target.position - offset * currentZoom;
        transform.LookAt(target.position + Vector3.up * pitch);
        transform.RotateAround(target.position, Vector3.up, currentYaw);
    }
```

Koodi 3. (Lague & Thirslund 2018.)

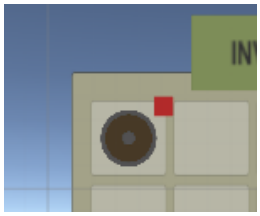
6.4 Vuorovaikutus

Pelissä tulee olla mahdollista poimia ja käyttää erilaisia tavaroita ja tarvikkeita; kaikki nämä luodaan erillisestä Interactable- kantaluokasta, joka määrittää niiden poimimiseen, käyttämiseen ja varastoimiseen liittyvät toiminnot. Kantaluokasta periytyvät luokat sisältävät siinä määritellyt osat, joten niitä ei tarvitse koodata useammin kuin kerran. Voidaan olettaa, että tästä kantaluokasta periytyvät kaikki pelissä vuorovaikutuksen kohteena olevat asiat, kuten poimittavat tavarat, painettavat nappulat ja vastustajat; joten kantaluokasta pitää löytyä näiden toiminnalle yhteiset piirteet: hahmon fokusointi, käyttösuunta, objektin liikkuessa seuraus ja itse virtuaalinen vuorovaikutuskomento. Myös muihin skripteihin tulee lisätä toimintoja, joihin viitataan kantaluokassa, kuten PlayerMotor- skriptiin toiminto, jolla pelaajahahmo kääntyy ja kävelee kohti valittua objektia.

Tavaroiden ja tarvikkeiden hyödyntämiseksi tarvitsemme Interactable- kantaluokasta periytyvän ItemPickup- luokan, johon luodaan tavaroiden nostamisen ja varastoimisen mahdollistava toiminto; Tämän lisäksi tarvitaan myös ”Inventory”- luokka, joka pitää muistissa kannetut tavarat ja mahdollistaa niiden käyttämisen.

ItemPickup- luokassa tarkennetaan Interactable- kantaluokassa määriteltyjä toimintoja poimittavan esineen tarkoituksiin sopivaksi. Poimittava esine tulee lisätä GameManager- objektin sisältämään inventaarioon, jonka jälkeen se poistetaan pelialueelta. Esineen tiedot luodaan ScriptableObject- komponentilla Item, joka on irrallisen datan tallennukseen käytettävä pohjapiirros: siihen tallennetaan tavaran nimi, UI-ikoni, statistiikat ja esineen ollessa päälle-puettava tarvike, asetetaan sille graafinen komponentti; jonka lisäksi mahdollistetaan tähän pohjapiirrokseen perustuvien tavaroiden luonti suoraan Unityn käyttöliittymässä. Inventaarion toiminta perustuu yksinkertaiseen listaan, johon lisätään tai josta poistetaan nostettuja esineitä, ja ne tallennetaan aikaisemmin mainitun GameManagerin sisältämään instanssiin Inventory- skriptistä; tähän tallentuvissa objekteissa on luodun Item- pohjapiirroksen sisältämä data myös tallennettuna. Inventory- skripti tunnistaa siinä tapahtuvat muutokset, ja näin graafinen käyttöliittymä tietää, milloin päivittää sen sisältämät ikonit.

Inventaario tarvitsee graafisen käyttöliittymän sen käytön mahdollistamiseksi; tämän luonti tapahtuu peli-ikkunan päälle maalattavan canvas- objektin käytöllä. Canvas on tarkoitettu käyttöliittymän elementtien luomiseen ja se on sidonnainen pelin ollessa käynnissä vain ruudun kokoon, eikä vastaa pelin zoomaukseen tai kameran kääntelyyn mitenkään. Aluksi canvakseen lisätään pienempiä paneeleita tavaroiden varastoimista varten inventaarion koon verran, jonka jälkeen määritellään mitä näille voi tehdä. Tässä työssä inventaarion sisällä tulee olla mahdollista käyttää ja poistaa tavaroita; tämä toiminnallisuus saadaan aikaiseksi lisäämällä kaksi painikekomponenttia inventaario-objektiin; ItemButton, joka käyttää kyseessä olevan tavaran, ja RemoveButton, joka poistaa tavaran. (Huom. Tämä ei ole GameManagerin sisältämä inventory- skripti, vaan erillinen peliobjekti). Inventaarion koko on rajoitettu, joten voidaan luoda ennalta määrätty määrä paneeleita, jotka täyttyvät sitä mukaa kuin tavaroita nostetaan; kutsumme näitä InvSlot- nimellä. Paras tapa näiden luomiseen on luoda ”Prefab” (Prefabricated object) yhdestä valmiista InvSlot-objektista, ja luoda siitä haluttu määrä kopioita, jotka saavat kaikki alkuperäiseen objektiin tehdyt muutokset automaattisesti. Inventaariolle luodaan myös erillinen paneeli nimeä varten. (Kuva 17.)



Kuva 17. Inventaario-ikkuna. Tavarapainike, ja poistopainike

Varusteiden luominen tapahtuu myös ScriptableObject- toiminnallisuuden avulla, paras tapa tähän on periyttää se aikaisemmin luodusta Item- ScriptableObjectista. Equipment- ScriptableObjectiin tulee lisätä varusteiden ominaisuudet ja varusteen käyttösijainti, esimerkiksi haarniskalle armorMod- ja damageMod- muuntimet ja equipSlot- sijainti; näin saadaan varmistettua sille haluttu paikka ja toiminta. ArmorMod- ja damageMod- muuntimien toiminta tullaan määrittämään myöhemmin statistiikkojen määrittämisen yhteydessä. Varusteiden toiminnan ohjaus tapahtuu GameManageriin lisättävällä Equipment- Manager- skriptillä, joka sisältää varustesijaintien määrän suuruisen taulukon, johon lisätään tai siitä poistetaan varusteita niitä käyttämällä.Varusteiden käyttämiseen tarvitaan kaksi metodia

EquipmentManageriin, Equip, joka lisää varusteen sille alustettuun paikkaan taulukossa (kypärä = 0, haarniska = 1, kilpi = 2 ja ase = 3) ja poistaa sen inventaariosta, ja UnEquip, joka poistaa kyseisen varusteen taulukosta, ja lisää sen inventaarioon. Varusteiden graafisten osien määrittely tapahtuu SkinnedMeshRenderer-komponentin avulla, jonka alustuksessa määritetään haluttu objekti, ja sen parent-objektin deformaatioluut, jotka tässä tapauksessa ovat pelaajamallin body- objektin sisältämät luut; näin saadaan lisättäville grafiikoille haluttu animaatio pelaajaan verrattuna. (Kuva 18.)

Kuva 18. SkinnedMeshRenderer lisää grafiikat hahmolle varusteiden käytön yhteydessä.

Jotta pelin alkaessa voidaan asettaa pelaajahahmolle tietyt varusteet oletuksena, tulee määrittää Item- ScriptableObjectissa luotu IsDefaultItem- muuttuja todeksi halutuille tavaroille, jotka tässä tapauksessa ovat hahmolle luodut vaatteet ja hiukset. Koodiin luodaan toiminto, joka käy läpi asetetut oletusvarusteet, ja asettaa ne hahmon päälle. Maassa oleville nostettaville tavaroille, tulee lisätä MeshFilter- ja Meshrenderer-komponentit, ja toiminto, joka muuttaa SkinnedMeshRenderer- tyyppiset grafiikat halutunlaisiksi komponenteiksi; tämä toiminto lisätään kontekstivalikkoon [ContextMenu("Convert to regular mesh")] komennolla, ja sen avulla saadaan luotua Unityn editoriin pikanäppäin (Convert to regular mesh), jolla grafiikat voidaan muuttaa pelialueelle renderöitäviksi.

(Haetaan SkinnedMeshRenderer objektista, lisätään siihen MeshRenderer ja MeshFilter, jonka jälkeen poistetaan SkinnedMeshRenderer.) (Koodi 4.)

```
[ContextMenu("Convert to regular mesh")]
void Convert()
    {
            SkinnedMeshRenderer skinnedMeshRenderer = GetComponent<SkinnedMe-
            shRenderer>();
            MeshRenderer meshRenderer = gameObject.AddComponent<MeshRen-
            derer>();
            MeshFilter meshFilter = gameObject.AddComponent<MeshFilter>();

            meshFilter.sharedMesh = skinnedMeshRenderer.sharedMesh;
            meshRenderer.sharedMaterials = skinnedMeshRenderer.sharedMateri-
            als;

            DestroyImmediate(skinnedMeshRenderer);
            DestroyImmediate(this);
    }
```

Koodi 4. (Lague & Thirslund 2018.)

6.5 Statistiikat

Taistelumekaniikan toiminta perustuu statistiikkojen, kuten terveyspisteiden, haarniskan vahvuuden ja aseiden voiman muokkaamiseen. Tässä prototyypissä terveyspisteiden menetys korreloi suoraan varusteiden voimakkuuksien kanssa, mutta usein on tapana käyttää monimutkaisia kaavoja sen tarkkaan laskemiseen; tämä johtuu usein syvällisestä taistelumekaniikasta verrattuna tämän projektin yksinkertaiseen toimintaan.

Statistiikkaskriptejä tarvitaan useita: CharacterStats, joka toimii kantaluokkana pelaajan ja vastustajan statistiikkojen logiikan luomiselle; PlayerStats, joka laskee hahmon terveyspisteet ja haarniska- ja vahinkomuuntimet, jotka saadaan päällä olevista varusteista ja Stat, joka alustaa ja mahdollistaa luodun statistiikan hakemisen muissa luokissa. Stat- skripti on sarjallistettu, joten sen osat saavat näkyvyysmääreen private, mutta niitä voidaan kuitenkin muokata Unityn editorissa. Stat- skriptillä voidaan laskea jokainen statistiikka erikseen, kun sitä kutsutaan muista skripteistä.

(Sarjallistetaan skripti, alustetaan muuttuja ja luodaan lista aktiivisista muuntimista, jonka jälkeen metodissa lasketaan muuntimien vaikutus lopulliseen statistiikkasummaan ja palautetaan lopullinen summa.) (Koodi 5.)

```
[SerializeField]
    private int baseValue;

    private List<int> modifiers = new List<int>();

    public int Getvalue ()
    {
        int finalValue = baseValue;
        modifiers.ForEach(x => finalValue += x);
        return finalValue;
    }
```

Koodi 5. (Lague & Thirslund 2018.)

CharacterStats- skriptissä alustetaan hahmoon vaikuttavat statistiikat, luodaan metodi terveyspisteiden vähentämiselle ja terveyspisteiden loppuessa luodaan virtuaalinen kuolematapahtuma, sillä pelaaja- ja vastustajahahmon kuollessa tulee tapahtua eri asioita; pelaajan kuollessa tulee mahdollistaa pelin uudelleenaloitus, ja vastustajan kuollessa tulee mahdollisesti tiputtaa tavaroita jne.

(Haetaan ja alustetaan armor- statistiikan sisältämä summa ja käytetään Mathf.Clamp-komentoa varmistamaan, että otettavan vahingon ollessa negatiivista, ei se lisää terveyspisteitä; jonka jälkeen vähennetään terveyspisteitä ja niiden ollessa 0 tai vähemmän, kutsutaan kuolemistapahtumaa.) (Koodi 6.)

```
public void TakeDamage (int damage)
    {
        damage -= armor.Getvalue();
        damage = Mathf.Clamp(damage, 0, int.MaxValue);

        currentHealth -= damage;

        if (OnHealthChanged != null)
        {
            OnHealthChanged(maxHealth, currentHealth);
        }

        if (currentHealth <= 0)
        {
            Die();
        }
    }
```

Koodi 6. (Lague & Thirslund 2018.)

6.6 Vastustaja

Vastustajahahmon luominen on lähes identtinen pelaajahahmon luomisprosessin kanssa, mutta eroavaisuudet löytyvät niihin liitetyistä skripteistä. Vastustaja on tekoälyn kontrolloima, ja sen toiminta perustuu pelaajahahmon seuraamiseen silloin, kun pelaaja on tietyn matkan päässä vastustajasta, ja ollessaan tarpeeksi lähellä, tulee sen hyökätä pelaajan kimppuun. Vastustajalle asetettu EnemyController- skripti sisältää metodit pelaajan sijainnin tunnistamiseen verrattuna vastustajan sijaintiin (Tähän tarvitaan myös pelaajan sijainnin tallentava PlayerManager- skripti), seuraamiseen pelaajan ollessa tarpeeksi lähellä, pelaajan suuntaan katsomiseen ja lopulta pelaajan kimppuun hyökkäämiseen. Jotta taistelu toimii molempiin suuntiin, luodaan vastustajalle myös Interactable- luokasta periytyvä Enemy- luokka, joka mahdollistaa sen fokusoimisen, ja sitä vastaan taistelun. Vastustajan kuolintapahtuma luodaan CharacterStats- luokasta periytyvässä EnemyStats- luokassa, jossa ylikirjoitetaan virtuaalinen Die- metodi, ja vastustajan kuollessa tuhotaan peliobjekti.

(Ylikirjoitetaan kuolinmetodi, jossa tuhotaan kuollut vastustajaobjekti) (Koodi 7.)

```
public override void Die()
    {
        base.Die();

        Destroy(gameObject);
    }
```

Koodi 7. (Lague & Thirslund 2018.)

6.7 Taistelu

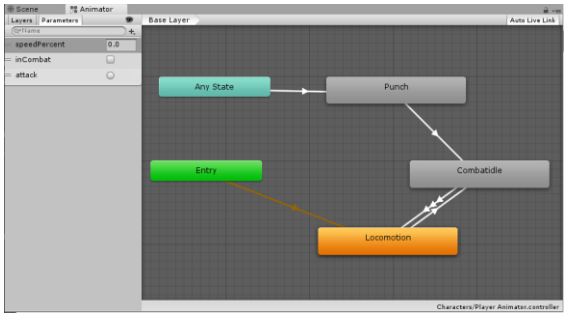
Tässä projektissa luotu taistelumekaniikka toimii ajoitettujen hyökkäyskutsujen avulla, jotka muuttavat pelin ollessa käynnissä aikaisemmin asetettuja CharacterStats-luokasta periytyviä statistiikkoja. Pelaajahahmon hyökkäykset toimivat napinpainalluksilla, ja vastustajan hyökkäykset tapahtuvat tietyin aikavälein. Hyökkäykset kutsuvat TakeDamage- metodia, joka aiheuttaa terveyspisteiden vähenemisen armor- ja damage- statistiikkojen mukaan. Hyökkäysten aikaväliä hallinnoidaan attackCooldown- ja attackSpeed- muuttujilla.

(Tarkistetaan onko hyökkäysten välille asetettu odotusaika mennyt, jonka jälkeen tarkistetaan, ollaanko parhaillaan hyökkäämässä. Jos molemmat ovat halutussa tilassa, kutsutaan hyökkäystapahtumaa, ja asetetaan hahmon inCombat- tila todeksi; tämän lisäksi asetetaan muistiin aika, jolloin hyökkäys tapahtui) (Koodi 8.)

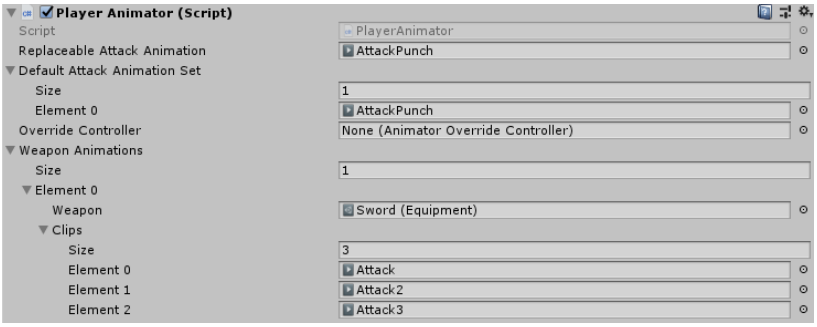
```
public void Attack(CharacterStats targetStats)
    {
        if (attackCooldown <= 0f)
        {

            opponentStats = targetStats;
            if (OnAttack != null)
            {
                OnAttack();
            }

            attackCooldown = 1f / attackSpeed;
            InCombat = true;
            lastAttackTime = Time.time;

        }
    }
```

Koodi 8. (Lague & Thirslund 2018.)

Hahmon ollessa taistelussa, asetetaan sille taisteluasento jonka pohjalta hyökkäysanimaatiot luotiin, näin varmistetaan niiden välillä siirtymisen sulava animointi. Hyökkäysanimaatioiden lisäämisen yhteydessä luodaan uusi animaatiopuu, joka saa oletusarvona yhden hyökkäysanimaation (Kuva 19); aseen vaihtuessa luodaan eri hyökkäyksiä sisältävä taulukko, ja toiminta joka valitsee sen sisältämistä animaatioista satunnaisesti yhden, ja ylikirjoittaa sillä animaatiopuussa asetetun hyökkäyksen; tämä mahdollistaa yhden animaatiokontrollerin käytön tilanteessa, jossa on mahdollista olla useita animaatioyhdistelmiä. (Kuva 20.)

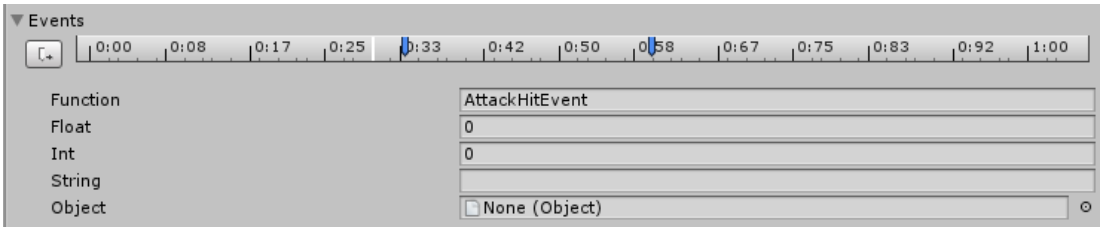


Kuva 19. Valmis animaatiokontrolleri.



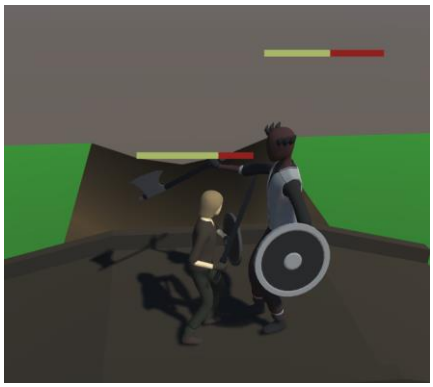
Kuva 20. PlayerAnimator- skriptin asetukset Unityn editorissa.

Hyökkäysten vaikutusten sitominen animaatioihin on käytössä ensisijaisesti, jotta on mahdollista päättää, millä hetkellä animaatiossa tehdään vahinkoa vastustajaan, mutta se myös mahdollistaa niiden toimintojen muuttamisen Unityn editorissa ilman erikseen luotavaa koodia. Hahmon hyökätessä toistetaan yksi kolmesta satunnaisesta animaatiosta, joista kaksi vahingoittavat vastustajaa kerran ja yksi kahdesti, näin saadaan taisteluun hieman satunnaisuutta. Tämä toiminnallisuus voidaan luoda animaatioiden lisäämisen yhteydessä asettamalla valitulle animaatiolle kaksi AttackHitEvent- tapahtumaa yhden sijaan. (Kuva 21.)



Kuva 21. AttackHitEvent- tapahtumat hyökkäysanimaatiossa (sinisellä).

Taisteluanimaatioiden lisäksi tarvitaan myös visuaalinen komponentti, joka kertoo terveyspisteiden määrän taistelun aikana; tämän luomiseen käytetään käyttöliittymän canvas- objektia, ja sen toiminta määritetään HealthUI- skriptissä seuraavanlaiseksi: Hahmon menettäessä terveyspisteitä muuttuu terveysmittari näkyväksi, ja menetetyn prosentin verran punaiseksi. (Kuva 22.)



Kuva 22. Hyökkäysanimaatiot ja terveyspistemittari.

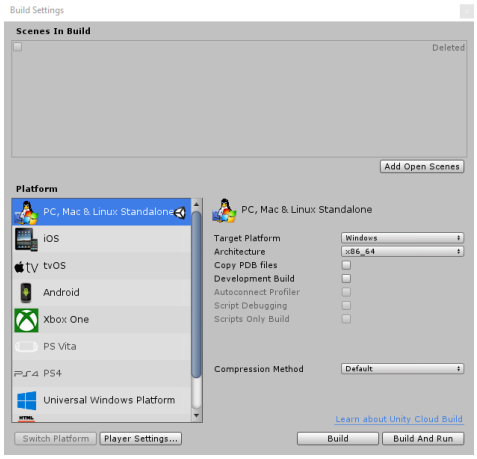
(Terveyspisteiden vaihtuessa tarkistetaan, onko mittari näkyvissä, jos ei, asetetaan se näkyväksi ja täytetään healthSlider- komponenttia menetetyn pistemäärän verran. Hahmon kuollessa tuhotaan komponentti.) (Koodi 9.)

```
void OnHealthChanged(int maxHealth, int currentHealth)
    {
        if (ui != null)
        {
            ui.gameObject.SetActive(true);
            lastMadeVisibleTime = Time.time;

            float healthPercent = (float)currentHealth / maxHealth;
            healthSlider.fillAmount = healthPercent;

            if (currentHealth <= 0)
            {
                Destroy(ui.gameObject);
            }
        }
    }
```

Koodi 9. (Lague & Thirslund 2018.)

6.8 Projektin valmistuessa

Projektin ollessa halutussa vaiheessa, voidaan se kääntää Unityn sisällä usealle eri alustalle. Näin yksinkertaisen pelin ollessa kyseessä, on tämä prosessi nopea ja kevyt, mutta kesto ja raskaus nousevat projektin koon kasvaessa. (Kuva 23)



Kuva 23. Kääntäjäasetukset Windows- alustalle luotavassa koontiversiossa.

6.9 Valmiin prototyypin kuvaus

Valmis peliprototyyppi sisältää seuraavat mekaniikat:

- Täysin animoidut pelaaja- ja vastustajahahmot.
- Polunetsinnällä varustetut liikkumiseen liittyvät skriptit.
- Pelaajaan kohdistettu kamera.
- Inventaario, joka mahdollistaa tavaroiden ja varusteiden poimimisen, poistamisen ja käyttämisen.
- Statistiikkoja muokkaavat varusteet, kuten haarniskat ja aseet.
- Statistiikkojen perusteella toimiva taistelusysteemi.
- Satunnaisesti toimiva hyökkäysanimaation valinta, joka asettaa 1/3 mahdollisuuden tuplavahinkoon.
- Graafinen käyttöliittymä, joka sisältää inventaarion ja terveyspistemittarin.

7 LOPUKSI

Pelinkehitysprosessin aloitus voi tuntua aloittelijalle miltei ylitsepääsemättömältä esteeltä, mutta kun otetaan huomioon verkosta löytyvä massiivinen määrä opetusmateriaalia ja dokumentointia, on kenen vain helppo päästä alkuun. Itse koin prosessin haastavana, mutta hyvin opettavaisena ja palkitsevana.

Opinnäytetyötä aloittaessani ei minulla ollut juurikaan pelinkehitykseen liittyviä taitoja, joten luomani prototyyppi on suppea, eikä sisällä mitään suuria ideoita, mutta ensiaskeleenani pelinkehitykseen sen tärkeys on huomattava. Nyt opinnäytetyön ollessa valmis, olen innokas jatkamaan pelien kehitystä, ja haluan löytää hyviä ja mielenkiintoisia tapoja luoda uusia oppimismahdollisuuksia peleistä kiinnostuneille.

Pelimoottorin avulla voidaan nykyään kehittää näyttäviä kokonaisuuksia suurella skaalalla. Laajasta pelimammutista pieneen harrastelijamaiseen kokonaisuuteen. Pelimoottorit ovat lähes täysin syrjäyttäneet alkuperäisen pelinkehitystavan rakentaa kaikki itse, ja tulevaisuudessa tulevat kehittymään mahdollisesti jopa pelimarkkinoiden ulkopuolelle muuhunkin ohjelmistokehitykseen.

Suuremmilla pelistudioilla on omat suljetut ympäristönsä pelinkehitykseen, ja ne ovat vaalittuja salaisuuksia. Kuitenkin markkinoilla olevilla avoimilla pelimoottoreilla on mahdollista luoda loputtomia maailmoja ja tarinoita pelaajien nautittavaksi.

Lopulta pelinkehityksen tärkein askel kuitenkin on sisällön hahmottaminen ja luominen, joka saadaan toteutettua pelimoottorin avulla vaivattomasti.

LÄHTEET

Clark, R. 2015. The 5 Myths of the Indiepocalypse. Viitattu 16.03.2018 https://www.gamasutra.com/blogs/RyanClark/20150908/253087/The_5_Myths_of_the_Indiepocalypse.php

Coster, A. 2015. Indiepocalypse? More like INDIESCHMOCALYPSE! Viitattu 11.08.2018 https://www.bscotch.net/post/indiepocalypse-more-like-indieschmocalypse

Crosby, T. 2008. How Making a Video Game Works. Viitattu 22.01.2018 https://electronics.howstuffworks.com/making-a-video-game.htm

Elhady, H. 2017. Top Game Engines in 2018. Viitattu 27.02.2018 https://blog.instabug.com/2017/12/game-engines/

Epic Games. 2018. What is Unreal Engine 4. Viitattu 23.01.2018 https://www.unrealengine.com/en-US/what-is-unreal-engine-4

Floyd, D. & Portnow, J. 2015a. Making Your First Game: Basics – How To Start Your Game Development - Extra Credits. Viitattu 22.01.2018 https://www.youtube.com/watch?v=z06QR-tz1_o

Floyd, D. & Portnow, J. 2015b. Making Your First Game: Practical Rules - Setting (and Keeping) Goals - Extra Credits. Viitattu 22.01.2018 https://www.youtube.com/watch?v=dHMNeNapL1E

Floyd, D. & Portnow, J. 2015c. Making Your First Game: Minimum Viable Product - Scope Small, Start Right - Extra Credits. Viitattu 22.01.2018 https://www.youtube.com/watch?v=UvCri1tqIxQ

Gril J. 2008. The State of Indie Gaming. Viitattu 21.03.2018 https://www.gamasutra.com/view/feature/132041/the_state_of_indie_gaming.php

Lague, S. & Thirslund, A. 2018. How to make an RPG in Unity. Viitattu 4.10.2018 https://www.youtube.com/playlist?list=PLPV2KyIb3jR4KLGCCAciWQ5qHudKtYeP7&disable_polymer=true

Lowood, H. 2014. Game Engines and Game History. Viitattu 28.01.2018 http://www.kinephanos.ca/2014/game-engines-and-game-history/

Normann K. 2016. Game Engines: Past, Present and Future. Viitattu 21.03.2018 https://www.gamasutra.com/blogs/KevinNormann/20160412/270186/Game_Engines_Past_Present_and_Future.php

O' Flanagan, J. 2015. Game Engine Analysis and Comparison. Viitattu 23.01.2018 https://www.gamesparks.com/blog/game-engine-analysis-and-comparison/

Unity3D. 2018. Features. Viitattu 23.01.2018 https://unity3d.com/unity

Unity3D. 2018. Creating and Using Scripts. Viitattu 21.03.2018. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

Unity3D. 2018. Art and Design. Viitattu 27.02.2018 https://unity3d.com/unity/features/editor/art-and-design

Ward, J. 2008. What is a Game Engine? Viitattu 22.01.2018 https://www.gamecareerguide.com/features/529/what_is_a_game_.php

YoYo Games Ltd. 2018. Easy To get Started. Viitattu 23.01.2018 https://www.yoyogames.com/gamemaker