LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Pelinkehitys Unity 5-pelimoottorilla

Taskinen, Tuomas
Udd, Jaakko

2016 Laurea

Laurea-ammattikorkeakoulu
Leppävaara

Pelinkehitys Unity 5-pelimoottorilla

Taskinen, Tuomas
Udd, Jaakko
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Toukokuu 2016

Laurea-ammattikorkeakoulu **Tiivistelmä**
Laurea Leppävaara
Tietojenkäsittelyn koulutusohjelma

Taskinen, Tuomas & Udd, Jaakko

Pelinkehitys Unity 5-pelimoottorilla

Vuosi 2016 Sivumäärä 38

Opinnäytetyön tavoitteena oli tehdä toimiva pelisovelluksen prototyyppi PC-alustalle Unity 5-pelimoottorilla. Tutkimusmenetelmänä käytettiin proton rakentamista ja tuloksena syntyi prototyyppi pelisovelluksesta. Opinnäytetyö on tarkoitettu pelinkehitysprosessista kiinnostuneille.

Raportissa käsitellään Unity 5:n työkaluja, joita käytettiin pelisovelluksen prototyypin luomisessa. Toteutusluvussa kuvattu ohjelmointi auttaa aloittelevia peliohjelmoijia erilaisten pelitoimintojen toteutuksessa. Lisäksi raportissa on kuvattu suunniteltujen peliobjektien ja -sisältöjen tuottaminen Unity 5–alustalle.

Tietoperustan muodostaa teoria kolmesta pelinkehitysprosessin tasosta, jotka osoittautuivat hyödyllisiksi rajallisessa ajassa suoritettavalle peliprojektille. Konseptivaiheessa suunniteltiin alustavat pelitoiminnot ja -hahmot. Työstämisvaiheessa aloitettiin suunniteltujen pelitoimintojen ohjelmointi ja hahmojen luonti. Pelinkehityksen viimeisessä vaiheessa havaittiin kehitystarpeita pelimekaniikoissa. Esimerkiksi pelaajan kontrollointi vaati lisää kehittämistyötä ja sitä onnistuttiin parantamaan.

Prototyyppi sisältää yhden testikentän, jota koehenkilöt testasivat. Koehenkilöiden palautteen perusteella prototyyppiä kehitettiin. Pelisovelluksen kehittäminen jatkuu myös tämän opinnäytetyöraportin julkaisemisen jälkeen. Tavoitteena on tuottaa kokonainen pelisovellus, joka on tarkoitus julkaista pelikäyttöön.

Asiasanat: Pelinkehitys, Pelisuunnittelu, Ohjelmointi, 3D-mallinnus

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology

Taskinen, Tuomas & Udd, Jaakko

Game Development with Unity 5 Game Engine

Year 2016 Pages 38

The main goal of the thesis was to develop a functional prototype of a game application for PC platform using Unity 5 game engine. Prototype developing was used as a research method and a prototype of game application was formed as a result. This thesis is produced to those who are interested in game development.

The report contains information about the tools in Unity 5 which were exploited to create the prototype of the game application. The programming described in the chapter of the thesis gives guidelines on how to execute distinct game mechanics for novice game programmers. The game asset production pipeline for Unity 5 platform is also described in the report.

The theory of three levels of game development process was used as a knowledge base and it proved to be beneficial for the game project that was developed in a limited amount of time. The concept phase contains information about designing game mechanics and game assets. The creation of the designed game mechanics and game assets were commenced in the production phase. The need to develop the game mechanics was identified in the last phase of the game development. For instance, player control demanded more development work and it was enhanced successfully.

The prototype contained one game level that the test subjects tested. The prototype was enhanced according to the feedback from the test subjects. The development of the game application continues after the thesis is published. The objective is to produce a complete version of the game application and it is proposed to be published for gaming purposes.

Keywords: Game development, Game design, Programming, 3D-modeling

Sisällys

1 Johdanto 6
2 Opinnäytetyössä käytetyt työkalut 7
2.1 Unity 5 7
2.2 Mari ja Modo 8
3 Pelisuunnittelu 8
3.1 Säännöt pelin määrittelijänä 9
3.2 Pelimekaniikat 10
3.3 Pelin käyttöliittymäsuunnittelu 10
3.3.1 HUD:n suunnittelu 11
3.3.2 Käyttöliittymän ulkoasu ja tunnelma 11
3.4 Pelikenttien suunnittelu 12
3.4.1 Pelikenttien polut 12
3.4.2 Objektien sijoittelu pelikentille 13
3.4.3 Pelikentän visualisointi 14
3.4.4 Pelikentän suunnittelun ongelmia 14
4 Pelin prototyypin toteutus 15
4.1 Projektin aloitus Unity 5:llä 15
4.2 Pelaajan kontrollointi 16
4.2.1 PlayerHealth-skripti 18
4.2.2 Vihollisen EnemyBox-skripti 20
4.2.3 Pisteiden kerääminen 21
4.2.4 Unity 5:n navigointityökalu 22
4.3 Pelikäyttöliittymän suunnittelu ja toteutus 23
4.4 Kenttäsuunnittelu 27
4.4.1 Ensimmäinen kenttä-suunnitelma 28
4.4.2 Kenttien luonti 29
4.5 Pelisisällön tuottaminen Unity 5-pelimoottoriin 31
4.6 Pelin hienosäätövaihe 33
5 Yhteenveto 34
Lähteet 36
Kuviot 37
Taulukot 38

1 Johdanto

2000-luvulla peliteollisuus on ollut nopeimmin kasvava viihdeteollisuuden ala. Peliteollisuus on ohittanut jopa tallennetun musiikin myynnin ja saavuttaa elokuva-alaa. Pelimarkkinoiden globaaliuden takia peliteollisuus on Suomessa merkittävä tekijä kansantalouden ja vientitulojen kannalta. (Neogames Finland Association 2016.)

Opinnäytetyön aiheena on pelinkehitys Unity 5-pelimoottorilla. Opinnäytetyön tavoitteena oli tehdä toimiva pelisovelluksen prototyyppi PC-alustalle Unity 5-pelimoottorilla. Tuomas Taskinen keskittyi pelin rakentamiseen vaadittavien toimintojen tekemiseen ja ohjelmointiin Unity 5:llä. Jaakko Udd keskittyi graafisen sisällön tuottamiseen. Opinnäytetyössä otettiin huomioon pelin kannalta tärkeimmät ohjelmointiin ja graafisen sisällön tuottamiseen liittyvät toimintaprosessit. Raportissa käsitellyt pelisuunnitteluteoriat tukivat projektin kulkua.

Tämä opinnäytetyö on toiminnallinen opinnäytetyö. Toiminnallisen opinnäytetyön tuotoksena tuotettiin prototyyppi pelisovelluksesta. Pelinkehityksen kannalta tärkeimmät vaiheet on kirjattu opinnäytetyön kirjalliseen raporttiin. Tämä toimii opiskelijoiden ammatillisen tiedon, taidon ja sivistyksen näytteenä (Vilkka 2006, 70). Tutkimusmenetelmänä käytettiin proton rakentamista ja tuloksena syntyi prototyyppi pelisovelluksesta. Prototyypeillä tarkoitetaan nopeasti rakennettavia karkean tason malleja suunnitellusta tuotteesta. Niiden avulla testataan ja varmistetaan tuotteen toimivuus. Kohderyhmälle suoritetusta testauksesta saadun palautteen perusteella prototyyppiä voidaan kehittää ennen lopullisen tuotteen julkaisemista. Prototypoinnilla voidaan vähentää kehitettävän tuotteen epäonnistumisen riskejä. (Koivisto 2012.)

Raportissa käsitellään Unity 5–pelimoottorin tärkeimpiä työkaluja, joita vaadittiin pelisovelluksen prototyypin toteuttamiseksi. Unity 5-pelimoottori sopii pienille studioille, indie-pelien tuottajille tai niille yksityishenkilöille, jotka haluavat toteuttaa oman pelin. Unity 5:llä on suuri ja aktiivinen käyttäjäyhteisö, joka mahdollistaa aloittelevien sekä kokeneempien käyttäjien informaation jakamisen esimerkiksi ongelmatilanteissa. (Blackman 2011, xix.) Nämä ominaisuudet sopivat aloitteleville pelinkehittäjille, jonka vuoksi pelimoottoriksi valittiin Unity 5.

Opinnäytetyön raportti on tarkoitettu pelinkehitysprosessista kiinnostuneille. Raportin toiminnallisessa osuudessa kuvattu ohjelmointi auttaa aloittelevia peliohjelmoijia erilaisten pelitoimintojen toteutuksessa. Lisäksi raportissa on kuvattu suunniteltujen peliobjektien ja -sisältöjen tuottaminen Unity 5–alustalle.

Aineistoa opinnäytetyön tekemiseen kerättiin kirjoista ja internet-sivustoilta. Opinnäytetyön yksi osa-alue on ohjelmointi, jossa käytettiin C#-ohjelmointikieltä. Ohjelmoinnin tukena käytettiin Unity 5:n kotisivuilta saatavaa ohjelmointimanuaalia. 3D-mallintamisen ja teksturoinnin tukena käytettiin kyseisten ohjelmien manuaaleja sekä kirjallisuutta.

Pelisovelluksen kehittäminen jatkuu myös tämän opinnäytetyöraportin julkaisemisen jälkeen. Tavoitteena on tuottaa kokonainen pelisovellus, joka on tarkoitus julkaista pelikäyttöön. Mahdollisena julkaisukanavana on suunniteltu käytettävän Steam-jakelua. Steam-jakeluun pääsee Steam Greenlight-yhteisön kautta. Steam-yhteisö antaa palautetta ja Greenlight auttaa julkaisijoita kehittämään peliä. Yhteisössä pelinkehittäjät ja julkaisijat esittelevät pelejään kuvakaappauksilla ja videoilla. Jos peli saa riittävästi yhteisön tukea, se valitaan Steam-jakeluun. (Valve Corporation 2016.)

2 Opinnäytetyössä käytetyt työkalut

Päätyökaluna oli Unity 5–pelimoottori, jolla koottiin pelisovelluksen prototyyppi. Unity 5-pelimoottori sisältää työkaluja animointiin, ohjelmointiin eli skriptaamiseen ja materialisoimiseen. Pelisovelluksen tuottaminen vaatii pelihahmoja ja muita peliobjekteja, jotka tuotettiin Modo 3D-mallinnusohjelmalla.

2.1 Unity 5

Opinnäytetyön lopullinen tuotos, eli prototyyppi pelisovelluksesta tehtiin Unity 5-alustalle. Unity 5-pelimoottorilla voidaan kehittää 2D- ja 3D-pelejä useille eri alustoille. Unity 5:n ilmaisversio antaa mahdollisuuden pelien kehityksestä kiinnostuneille kokeilla, tuottaa, oppia ja myydä omia pelejä. (Blackman 2011, xix.)

Pelin toteutukseen kuuluu aina ohjelmointi eli skriptaaminen. Yksinkertaisinkin peli tarvitsee skriptaamista. Sillä ohjataan animaatioita, objektien fyysistä käyttäytymistä ja kaikkia pelin tapahtumia sekä määritetään milloin ne tapahtuvat. Skriptaaminen tehtiin Unity 5:ssä MonoDevelop -työkalulla. (Unity Technologies 2015.)

Unity 5:n animaatiojärjestelmä perustuu animaatioleikkeisiin eli klippeihin. Tietyn objektin paikasta, ilmansuunnasta tai muusta olemuksesta kootaan animaatioklippi. Klipit sisältävät informaatiota siitä, miten tietty objekti muuttaa paikkaansa, ilmansuuntaansa tai muita asetuksia. Tietyn objektin animaatioklippien toimintoja organisoidaan ja hallitaan Animator Controller-työkalulla. (Unity Technologies 2015.)

2.2 Mari ja Modo

Modo on 3D-mallintamiseen, animaation luontiin ja renderöintiin tarkoitettu ohjelma, joka toimii Windows-, Mac- ja Linux-käyttöjärjestelmissä. Renderöinnillä tarkoitetaan prosessia, jolla muunnetaan digitaalinen tieto näytölle sopivaan esitysmuotoon. 3D-mallintamisella tarkoitetaan objektien luontia kolmiulotteisiksi käyttäen hyväksi verteksejä eli keskipisteitä ja niiden X-, Y- ja Z-arvoja kolmiulotteisessa avaruudessa. Esimerkiksi kuutio koostuu kahdeksasta verteksistä, joilla kaikilla on omat X-, Y- ja Z-ulottuvuutensa. Tietyt verteksit yhdistyvät toisiinsa luoden särmän tai reunan eli edgen. Kun kolme tai suurempi määrä verteksejä yhdistyy, ne muodostavat polygonin eli sivun. Kuutiolla on kuusi polygonia. (Modo Online Help 2014.)

Modo-ohjelmalla 3D-mallinnettu objekti tuotiin Mari 3.0–teksturointiohjelmaan. Mari-teksturointiohjelmalla luodaan tekstuurit eli värit 3D-objekteihin piirtämällä UV-kankaaseen yksilöity maalaus. Tekstuureilla tarkoitetaan erilaisia värikarttoja, jotka liitetään 3D-objektiin. (Mari User Guide 2015.)

3 Pelisuunnittelu

Pelisuunnittelussa on kolme tasoa. Pelisuunnittelun konseptitasolla tarkoitetaan pelisuunnittelun vaihetta, jossa projektiryhmä suunnittelee pelin pääidean, kohdeyleisön ja pelaajan roolin pelissä. Nämä asiat dokumentoidaan ja ne tulisi pitää samana koko suunnitteluprosessin ajan. Konseptivaiheessa voi nopeasti toteuttaa suunnitelman olennaisimmista pelimekaniikoista, jotta pelin pelattavuutta pystytään jo tässä vaiheessa arvioimaan. Pelisuunnittelun pääpiirteiden havainnollistamiseen ei tule käyttää liikaa aikaa, koska pelin idea saattaa muuttua usein suunnitteluprosessin aikana. (Adams & Dormans 2012, 13.)

Adams ja Dormans kertovat Game Mechanics- kirjassaan (2012, 15) pelisuunnittelija Kyle Gablerin käytännöllisen neuvon, jonka tarkoitus on auttaa toteuttamaan pelejä lyhyellä aikavälillä. Yksinkertainen, mutta relevantti neuvo on ”make the toy first” eli tee ensin lelu. Gabler tarkoitti tällä neuvolla, että on parempi varmistaa pelimekaniikan toimivuus, kuin käyttää aikaa sisällön tuottamiseen.

Peli-idean suunnittelun jälkeen siirrytään työstämisvaiheeseen. Tässä vaiheessa luodaan pelimekaniikoita, pelikenttiä ja luonnostellaan pelin tarina sekä luodaan peliobjekteja. Työstämisvaiheessa on tärkeää projektiryhmän työskentely pienissä sykleissä, jolloin valmistuu pelattavaa ja prototyyppistä materiaalia peliin. Sitä testataan ja kehitetään jatkuvasti toimivammaksi ennen kuin pelisuunnittelu voi edetä seuraavalle tasolle. Tässä vaiheessa projektiryhmä voi joutua uudelleensuunnittelemaan monia pelitoimintoja. Työstämisvaiheessa on

hyvä ottaa kohdeyleisöä kokeilemaan pelin toimivuutta. Pelikehitysryhmä ei saa itse testaamalla hyvää kuvaa siitä, kuinka pelaajat oikeasti pelaavat ja lähestyvät peliä. (Adams & Dormans 2012, 13.)

Viimeisenä pelikehityksessä on hienosäätövaihe. Tässä vaiheessa pelikehitystiimi on tyytyväinen ja tehnyt peliin riittävästi toimintoja. Toimintoja ei enää lisätä peliin vaan niiden hienosäätö alkaa. Tämän vaiheen uhkakuvana on, että keksitään uusia ideoita, joita ei aikaisemmissa vaiheissa keksitty. Uusien ideoiden toteuttamisella voi olla tuhoavia vaikutuksia peliin ja se lisää huomattavasti testauksen ja hienosäätövaiheen työmäärää. Tässä vaiheessa on oleellisinta ottaa pelistä pois asiat, jotka eivät tuo lisäarvoa pelille. Tämän lisäksi on tärkeää keskittyä niihin elementteihin, jotka toimivat pelissä ja kehittää niitä paremmiksi. Hienosäätövaiheen työmäärää ei tule aliarvioida, koska se voi viedä yli puolet koko pelikehitysprosessin ajasta. (Adams & Dormans 2012, 13–14.)

3.1 Säännöt pelin määrittelijänä

Säännöt määrittelevät sen, mitä pelaaja voi pelissä tehdä ja kuinka pelissä toimitaan. Pelien tulisi olla ennalta-arvaamattomia, ettei pelaaja tiedä kaikkea heti sen alussa. Yleensä ennalta-arvattava peli ei ole yhtä viihdyttävä kuin ennalta-arvaamaton peli.(Adams & Dormans 2012, 1.)

Pelaajat haluavat, että heidän taitonsa ja strategiset päätöksensä vaikuttavat pelin kulkuun. Jos pelaajan taidoilla tai strategisilla päätöksillä ei ole väliä, pelaaja turhautuu nopeasti. Tämä ei kuulu hyvän pelin tunnusmerkkeihin. Monimutkaiset säännöt voivat tehdä pelistä ennalta-arvaamattoman. Monimutkaiset pelisysteemit sisältävät monia interaktiivisia osia. Yksittäisen peliosan säännöt voivat olla helppo ymmärtää, mutta pelin kaikki osat yhdistettyinä voivat tehdä siitä yllättävän ja ennalta-arvaamattoman. Shakki on klassinen esimerkki tästä vaikutelmasta. Shakin kuudentoista liikkuvan pelinappulan liikuttamisen säännöt on helppo hallita, mutta niistä yksinkertaisista säännöistä koostuu yhdessä hyvä ja monimutkainen peli. (Adams & Dormans 2012, 2-3.)

Suurin osa peleistä sisältää kolme ennalta-arvaamatonta elementtiä, jotka ovat pelisuunnittelussa avaintekijöitä. Nämä elementit ovat sattuma, pelaajan valinnat ja monitahoiset säännöt. Osa pelaajista pitää siitä, että peli sisältää monia ennalta-arvaamattomia elementtejä ja toiset pitävät pelin monimutkaisuutta ja pelistrategiaa pelin avaintekijöinä. Aiemmin mainituista kolmesta elementistä sattuma on helpoin toteuttaa, mutta se ei aina ole paras tekijä tehdä pelistä ennalta-arvaamatonta. Monimutkaiset pelisysteemit ja niiden lukuisat eri sattumat ovat liian vaikeita suunnitella toimiviksi pelin kannalta. (Adams & Dormans 2012, 2-3.)

3.2 Pelimekaniikat

Yleisiä peleissä esiintyviä mekaniikoita ovat muun muassa fysiikka-, talous- ja etenemismekaniikka. Fysiikkaa ovat pelin elementtien positio ja suunta, liikkuminen sekä interaktiivisuus peliobjektien törmätessä. Peleissä fysiikkaa voi esimerkiksi olla pelaajan liike paikasta toiseen, hyppiminen tai jollain kulkuneuvolla ajaminen. (Adams & Dormans 2012, 6.)

Pelin talousmekaniikkaan luokitellaan esimerkiksi pelissä helposti tunnistettavat ja kerättävät asiat. Näitä asioita voivat olla raha, energia tai ammukset. Pelin talous ei välttämättä sisällä pelkästään aineellisia tavaroita, vaan se voi myös sisältää pelaajan energiaan tai taianomaisiin asioihin liittyviä kerättäviä elementtejä. (Adams & Dormans 2012, 6.)

Pelin etenemismekaniikassa ja pelikenttien suunnittelussa määritellään, miten pelaaja etenee pelimaailmassa. Yleisesti pelihahmon täytyy päästä aloituspaikasta määrättyyn paikkaan voittaakseen päävihollisen tai pelastaakseen jonkun pelikentän lopussa. Tällaisessa pelissä pelaajan eteneminen on kontrolloitu useiden mekaniikoiden kautta, jotka estävät pelaajan kulkua pelikentän loppuun. Erilaiset vivut ja katkaisimet ovat tyypillisiä etenemismekaniikoita, joita pelaajan tulee löytää edetäkseen pelissä. (Adams & Dormans 2012, 6.)

3.3 Pelin käyttöliittymäsuunnittelu

Peleissä pelaaja tekee valintoja ja aiheuttaa pelitapahtumia. Käyttöliittymä on pelaajan ja pelin yhteys. Hyvin suunniteltu pelikäyttöliittymä tekee pelikokemuksesta miellyttävän. Pelien käyttöliittymät sisältävät interaktiivisia painikkeita, valikkoja ja monia muita käyttöliittymäkomponentteja. Loistavat käyttöliittymät eivät synny vahingossa, vaan ne vaativat paljon suunnittelua. Huonosti suunniteltu pelikäyttöliittymä voi vaikuttaa negatiivisesti pelikokemukseen. Käyttöliittymissä visuaalisuus on tärkeää, mutta vielä tärkeämpää on sen toimivuus. Pelin pelattavuus kärsii, jos pelaaja joutuu etsimään informaatiota pelin aikana tai jos pelaaja ei ymmärrä, kuinka navigoida valikosta toiseen. (Fox 2004, xvi.) Pelikäyttöliittymän tavoitteena on tehdä menusta yksinkertainen ja helppokäyttöinen. Pelikäyttöliittymän käytettävyyteen vaikuttaa muun muassa valikkojen lukumäärä ja pelin aloituksen helppous. (Fox 2004, 24.)

Paras tapa aloittaa pelikäyttöliittymän suunnittelu on tehdä vuokaavio. Pelikäyttöliittymä käsittää kaikki alkuvalikosta pelin aikana ilmestyviin valikkoihin. Hyvin suunniteltu vuokaavio antaa ohjelmoijalle mahdollisuuden aloittaa pelin ohjelmointi ilman menun lopullisia visuaalisia komponentteja. Vuokaavion luonnin tarkoitus on tehdä käyttöliittymästä helppokäyttöinen ja organisoida pelikehitysprosessin kulkua. (Fox 2004, 13.)

Vuokaavion luontiprosessi aloitetaan kirjaamalla pelivalikossa tarvittavat tiedot. Kaikkia pelivalikon tietoja ei kannata laittaa yhdelle sivulle. Vuokaavio koostuu taulukoista, jossa yksi taulukko kuvaa näytöllä näkyviä valikkovaihtoehtoja. Ensimmäiseksi luodaan vuokaavio kuvaamaan päävalikkoon tulevia valikkovaihtoehtoja, kuten esimerkiksi uusi peli, asetukset ja lataa. Tämän jälkeen luodaan valikkovaihtoehdoille omat taulukot ja lisätään niihin tulevat tiedot. Päävalikon ja alavalikoiden taulukot yhdistetään viivoilla kuvaamaan, mistä valikosta pääsee mihinkin. (Fox 2004, 14.)

Videopeleissä on yleensä alkioita, joihin pääsee vain, kun pelissä tietynlaiset tehtävät on suoritettu. Monissa peleissä on myös alkioita, jotka eivät ole saatavilla pelin alussa. Esimerkiksi pelikentät voivat olla lukossa ennen kuin edellinen pelikenttä on suoritettu loppuun. Pelikäyttöliittymän suunnitteluvaiheessa on hyvä tunnistaa tämän kaltaiset alkiot. Pelin alussa halutaan näyttää, kuinka monta pelikenttää on mahdollista saavuttaa. Tämä tulee vaikuttamaan siihen, miten pelikäyttöliittymän pelivalikko suunnitellaan. (Fox 2004, 14–15.)

3.3.1 HUD:n suunnittelu

HUD on englanninkielinen lyhennys sanoista heads-up display, jolla tarkoitetaan käyttöliittymää, joka näytetään pelin aikana. HUD voi sisältää esimerkiksi pelaajan terveysmittarin, pisteet tai peliajan. Pelin aikana näytettävän käyttöliittymän suunnittelussa tärkeintä on sen sisältämien elementtien organisointi. Suunnitteluvaiheessa on myös hyvä tietää kaikki elementit, jotka tulee olla esillä pelin aikana tai sen pysähtyessä. HUD tulee muuttumaan pelinkehitysprosessin aikana, koska peliä testatessa voi ilmetä asioita ja tietoja, joita pelaaja tarvitsee pelatessaan peliä. (Fox 2004, 24–25.)

3.3.2 Käyttöliittymän ulkoasu ja tunnelma

Paras tapa aloittaa käyttöliittymän ulkoasun suunnittelu on luoda malli käyttöliittymästä. Tavoitteena ei ole tehdä lopullisia visuaalisia objekteja malliin, vaan määritellä ja visualisoida käyttöliittymän tunnetta ja ulkoasua. Väriteema on tärkeä osa pelikäyttöliittymää. Pelikäyttöliittymää katsoessa loppukäyttäjän tulisi nähdä yhdellä silmäyksellä lähes koko pelin värimaailma. Värimaailma tulisi pitää johdonmukaisena, jotta pelissä säilyy yhteneväinen ilme. (Fox 2004, 27–30.)

Värimaailmaa suunniteltaessa luodaan värikartta, jolloin tulee varmistaa, millaista tunnetta pelin halutaan herättävän kohdeyleisölle. Pelin värimaailman tulisi sopia pelin aiheeseen. Pelikäyttöliittymä sisältää usein kuvia, mikä voi vaikuttaa väriteeman valintaan. Esimerkiksi kuva punaisesta autosta voi vaikuttaa käyttöliittymän väriteeman yhdeksi värivalinnaksi punaisen. Parempi tapa on tehdä värivalinta ensin ja sen jälkeen sovittaa kuva pelin väriteemaan. (Fox 2004, 27–30.)

3.4 Pelikenttien suunnittelu

Pelikenttien suunnittelussa on tärkeää, että kentät sopivat pelimaailmaan. Kentissä tulisi olla pelaajalle yllätyksiä, haasteita ja vaaroja. Rakennettaessa pelikenttiä tulisi ottaa huomioon logiikka, jolla kentät rakennetaan. Tällöin pelaaja pystyy suorittamaan kentän eksymättä tai joutumatta kiertelemään etsiessään polkua uuteen alueeseen. Loogisuus on tarpeellista kentän pääväylän luomisessa, mutta salareiteissä ja piilossa olevissa alueissa loogisuutta ei tarvita. Suunnittelijoiden täytyy tietää kenttien rakentamisen ja kehittämisen perusteet, koska kenttien suunnittelu voi pilata tai parantaa pelikokemusta. (Moore 2011, 293–294.)

Kenttien suunnittelussa yksi lähestymistapa on kuvitella kenttää tarinankerrontana. Kentällä on alku, missä pelaaja ilmestyy peliin. Kentässä on keskikohta, jonka tapahtumaketjuna pelaaja tutkii pelikenttää ja loppu, missä pelaaja lähtee kentältä. Kentän läpi kulkeva polku voi olla joko lineaarinen tai haarautuva. Sen eri reitit voivat johtaa eri alueille ja saattavat olla osittain päällekkäisiä. Lineaarinen tai suoraviivainen pelikenttä on helpompi suunnitella, koska siinä ei ole päällekkäisiä reittejä. (Moore 2011, 300.)

Kirjassa Basics of Game Design (2011, 300–301) Moore kertoo esimerkin pelikentästä. Pelaaja aloittaa kentän luolaston sisäänkäynniltä. Sisäänkäynnin ympäristö on yksi tutkittavista alueista, mutta ainoa reitti eteenpäin kentässä on mennä sisään luolastoon tai luolaan. Jos kentän suunnittelija haluaa polun olevan lineaarinen, luolan täytyisi olla kuin pitkä tunneli kääntyillen ylös ja alas, sekä mahdollisesti sisältää pieniä sivukäytäviä, joita pelaaja voi tutkia. Luola saattaa liittyä lopuksi ison rakennuksen kellaritiloihin ja pelaaja voi kerros kerrokselta edetä talosta ulos seuraavaan kenttään. Kentän polku on alusta loppuun lineaarinen, vaikka luolassa on sivukäytäviä ja rakennuksessa monta kerrosta. Jos suunnittelija kehittää luolasta luolaston, pelikentästä saadaan haarautuva. Lisäksi rakennus voi sisältää monia uloskäyntejä, jotka vievät pelaajan toisiin rakennuksiin, mitkä olisivat yhdistetty toisiinsa hisseillä ja portaikoilla. (Moore 2011, 300–301.)

Suunnittelijan täytyy ottaa huomioon myös pelialusta, jotta peli ei tulisi raskaaksi. Mikäli alusta on tehokas komponenteiltaan, suunnittelija voi tehdä kenttään kohtia, joissa pelaaja voi katsella yleisnäkymää koko kentästä. Mikäli grafiikkaa on liiaksi alustan tehokkuuteen verrattuna, peli hidastuu liian paljon. (Moore 2011, 301.)

3.4.1 Pelikenttien polut

Pelikenttä voi sisältää useita polkuja, jotka avautuvat eri aikoina pelin edetessä. Esimerkiksi kenttä voi sisältää lukitun oven, jonka pelaaja avaa avaimella, mikä löytyy toiselta puolelta kenttää. Oven avauduttua uusi polku avautuu samaan pelikenttään. Tällä kertaa oven aukaisu

voi laukaista ansan, joka aktivoi uusia vihollisia alueelle. Viholliset estävät pelaajan etenemistä. Tällaisen lähestymistavan hyvä puoli on se, että suunnittelija voi käyttää samoja graafisia peliympäristöjä ja objekteja hyödykseen luodessaan uusia alueita samassa pelikentässä. (Moore 2011, 301.)

Suunnittelijalla on mahdollisuus antaa pelaajalle vihjeitä siitä, miten pelikentässä edetään. Pelaaja voi esimerkiksi nähdä uuden aseen tai palkinnon paikasta, jonne hän ei kuitenkaan sillä hetkellä pääse. Tätä paikkaa voidaan korostaa esimerkiksi ylimääräisellä valaistuksella, jotta pelaaja kiinnittää huomionsa paikkaan tai palkintoon. (Moore 2011, 301.)

Rakennettaessa usean polun sisältämää pelikenttää eli leveliä, suunnittelijan on uudistettava polkuja. Samaan aikaan aktiivisina olevat polut voivat olla vaikeusasteeltaan samankaltaisia. Jos polut aukeavat pelaajalle eri aikaan, myöhemmin aktivoituvien polkujen tulisi olla vaikeampia aikaisempiin polkuihin verrattuna, jotta pelaaja säilyttää mielenkiinnon. Suunnittelijan täytyy myös muistaa pitää laskentatehot alhaalla, esimerkiksi vihollisten liiallinen lisääminen kenttään johtaisi jossain vaiheessa pelin hidastumiseen laskentatehon ylikuormituksen takia. Kentistä saa monipuolisempia vaihtamalla ympäristön ulkonäköä tai lisäämällä pulmanratkaisua taistelukohtauksen tilalle. (Moore 2011, 301–302.)

3.4.2 Objektien sijoittelu pelikentille

Objektien sijoittelu kentille on yhtä tärkeää, kuin itse kentän suunnittelu. Objekteja ei kannata sijoittaa sellaiseen paikkaan, missä pelaaja ei näe niitä tai pelaaja ei pääse käsiksi niihin. Aarrearkun kaltaiset objektit kannattaa sijoittaa paikkaan, jonne pelaaja ei aluksi pääse. Pelin edetessä pelaaja voi päästä käsiksi arkkuun ja peli palkitsee pelaajan ponnistelut. (Moore 2011, 302.)

Objektien sijoittelu pelikentällä voi toimia ohjeistuksena pelaajalle. Objekteja voidaan sijoittaa kentällä merkeiksi, joiden mukaan pelaaja suunnistaa. Esimerkiksi pelaaja näkee kaukana kiinnostavan tavaran ja lähestyy sitä. Tavaran poimittuaan pelaaja näkee vielä kauempana toisen tavaran ja lähtee sen perään. Tätä voi jatkua niin kauan, kunnes pelaaja on saavuttanut pisteen, jonne kentän suunnittelija halusi pelaajan menevän. (Moore 2011, 302.)

Objektit voivat olla uudestaan syntyviä sekä itsestään syntyviä. Tämä on hyödyllistä tilanteissa, joissa pelaajalle kertyy pisteitä tai rahaa kolikoista tai vihollisten tuhoamisesta. Hyviä esimerkkejä hyödyllisistä tavaroista pelissä ovat pelaajan terveydentilaa parantavat pullot, ammuslippaat, rahakirstut sekä turvapaikat, joihin pelaaja voi mennä turvaan tai tallentamaan pelin. (Moore 2011, 302–303.)

3.4.3 Pelikentän visualisointi

Kentän luonnissa käytetään apuna suunnitteluvaiheen dokumentaatiota. Dokumentaatiossa pitäisi esiintyä ajatukset ja ideat, kuten esimerkiksi kentän kasvillisuus, luonto tai rakennukset, puistot ja muut asiat. Huomionarvoiset viholliset sekä tärkeät tavarat täytyy myös mainita suunnitteludokumentaatiossa. (Moore 2011, 306–307.)

Visuaalisia kollaaseja on hyvä olla dokumentaatiossa, jotta luontivaiheessa ymmärretään miltä pelikentät näyttävät. Mooren mukaan (2011, 307) on helppoa käyttää internetistä löydettyjä kuvia ja piirroksia dokumentaatiossa, mutta scifityyppinen maailma voi olla haastavampaa koota valmiista kuvista. Moore kuitenkin muistuttaa, ettei tällaisen dokumentoinnin tulisi päätyä julkiseksi, koska kuvat ja piirrokset saattavat rikkoa tekijänoikeuslakia jos lupaa ei ole haettu alkuperäisiltä taiteilijoilta. (Moore 2011, 307.)

Luontivaiheessa ympäristöt saattavat muuttua erilaisiksi kuin, mitä oli suunniteltu. Tämä johtuu usein siitä, että luonnista vastaavat eri henkilöt kuin suunnittelusta. Kaikki haluavat kuitenkin jotain alkuperäistä ja omalaatuista, mitä kukaan ei ole aikaisemmin tehnyt. Luonnista vastaavat henkilöt olisi kannustettava keksimään muutamia eri ideoita, joista voitaisiin keskustella suunnittelusta vastaavien kanssa. Aina ei kuitenkaan haluta luoda täysin uutta, vaan halutaan saada peliin identtisen näköistä taidetta, kuten esimerkiksi sotakoneisto joltain tietyltä aikakaudelta. (Moore 2011, 307–308.)

3.4.4 Pelikentän suunnittelun ongelmia

Pelikenttien suunnittelussa voi ilmestyä lukuisia ongelmia, joita kannattaa välttää. Pelaajan liikkuminen pelikentällä ei saa olla liian rajattua. Pelaaja pyrkii tutkimaan kenttää etsiessään mahdollisia reittejä päästäkseen seuraavaan kenttään, joten vain yhteen rajattu reitti olisi pelaajalle pettymys. On olemassa tilanteita, joissa kentällä täytyy olla rajoitteita liikkumiselle. Tällaisia tilanteita ovat esimerkiksi alueet, joissa esiintyy objekteja liian paljon, jonka seurauksena pelisovelluksen toiminta hidastuu. Tämän kaltaiset tilanteet vältetään estämällä pelaajan pääsy kyseisiin alueisiin. Toisaalta pitäisi varoa tekemästä pelikenttää liian laajaksi. Jos kenttä kuvaa esimerkiksi aavikkoa, missä ei ole maamerkkejä suunnistuksen avuksi, pelaaja saattaa eksyä pitkäksi aikaa. Tällainen tilanne johtaa pelaajan turhautumiseen, mikä ei kuulu hyvin pelin tunnusmerkkeihin. (Moore 2011, 308.)

Toisena ongelmana on, että pelikentästä tehdään sisällöltään sellainen, joka ei tue pelattavuutta, pelimekaniikkaa tai pelimoottoria. Esimerkki tällaisesta ongelmasta on tilanne, jossa pelaaja selviää hypystä rotkon yli näkymättömän jalansijan avulla. Tällaiset ratkaisut tuovat vaikeutta pelattavuuteen, mutta saattavat viedä peli-ilon pelaajalta. (Moore 2011, 308.)

Pelikentän visualisoinnissa ongelmat voivat johtua aikakausien tai teemojen sekoittamisesta. Esimerkkinä keskiaikainen kirkko, jossa esiintyy egyptiläisiä hieroglyfejä. Myös liiallinen dramatisointi valaistuksen kanssa voi tuottaa vaikeuksia. Kauhupelissä tarvitsee hämäryyttä, mutta jos pelikenttä on liian hämärä, pelaaja ei näe pelata ja pelikokemus kärsii.(Moore 2011, 309.)

4 Pelin prototyypin toteutus

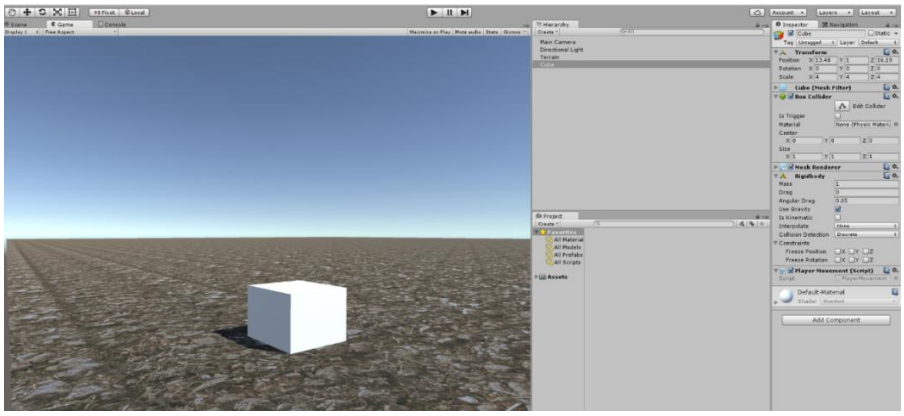
Tässä kappaleessa kuvataan pelisovelluksen prototyypin kehittäminen. Pelisuunnittelussa on otettu huomioon pelin käyttöliittymä- ja kenttäsuunnittelu. Adams ja Dormans esittelevät Game Mechanics- kirjassaan (2012, 13) kolme pelinkehitysprosessin tasoa. Näitä tasoja hyödynnettiin pelin prototyypin suunnittelussa ja toteutuksessa.

Pelinkehitysprosessi aloitettiin konseptivaiheella, jossa suunniteltiin peli-idea ja pelin alustavat pelihahmot sekä niiden toiminnot. Pelin päähahmo on laiva, jota pelaaja ohjaa. Pelimaailma koostuu erilaisista kentistä, joissa pelaajan tehtävä on ohjata laiva turvallisesti aloituspaikasta maaliin kentän läpäisemiseksi. Pelissä täytyy suojata laivaa erilaisilta vaaratilanteilta ja vihollisilta, jottei pelaajan laiva tuhoutuisi. Pelaaja voi väistellä vihollisia tai ampua niitä. Pelin taloudelliseen pelimekaniikkaan kuuluu kolikoiden kerääminen. Kolikoiden kerääminen kerryttää pelaajalle pisteitä.

Pelinkehityksen työstämisvaihe aloitettiin heti konseptivaiheen jälkeen. Tässä vaiheessa aloitettiin suunniteltujen pelihahmojen luonti. Samalla pystyttiin aloittamaan pelitoimintojen skriptaaminen. Kun peliin oli tuotettu riittävästi toimintoja ja sisältöä, siirryttiin pelinkehityksen viimeiseen vaiheeseen eli hienosäätöön.

4.1 Projektin aloitus Unity 5:llä

Pelin toteuttaminen opinnäytetyönä on lyhyen aikavälin projekti. Game Mechanics -kirjassa pelisuunnittelija Kyle Gablerin (2012, 15) neuvo ”make the toy first” on hyvä neuvo lyhyessä aikavälissä suoritettaville peliprojekteille. Tämä neuvo soveltuu myös aloitteleville pelinkehittäjille. Peli-idean suunnittelun jälkeen aloitettiin elementtien toteutus, jotka olivat mahdollista toteuttaa rajoitetussa ajassa. Kuviossa 1 on esitetty Unity 5:n käyttöliittymä peliprojektin alussa.



Kuvio 1: Projektin aloitus Unity 5:llä

Prototyypin toteuttaminen aloitettiin suunniteltujen pelihahmojen luomisella. Samalla aloitettiin työstämään pelin toimintoja, kuten pelaajan liikettä. Pelaajaa varten luotiin tyhjä peliobjekti, johon koottiin pelaajan toimintoja vaativia komponentteja. Varsinaisen pelaajan tilalla käytettiin kuutiota ennen lopullisen 3D-objektin valmistumista.

Kuviossa 1. Hierarchy-ikkunassa näkyy peliobjektit, jotka avoinna oleva peliprojekti sisältää. Project-ikkunassa ovat kaikki peliprojektin sisältämät peliobjektit ja komponentit. Inspector-ikkuna näyttää Hierarchy- tai Project-ikkunasta valitun objektin eri asetukset ja sen komponentit, kuten siihen liitetyt skriptit. Peliprojektin toteutusvaiheen ohjelmointi eli skriptaus aloitettiin pelin kannalta oleellisimmalla asialla, pelaajan kontrolloimisella.

4.2 Pelaajan kontrollointi

Pelaajan on tarkoitus kontrolloida peliobjektia rotaatiolla Y-akselin suuntaisesti eli kääntymällä sivuille. Pelaaja liikkuu vakionopeudella itsestään eteenpäin. Pelaajan pelihahmo on laiva, jonka keulassa olevaa asetta liikutellaan hiirellä. Painamalla vasemmanpuoleista hiiriosoitinta pelaaja ampuu. Pelaajan tulee pystyä häviämään peli eli kuolemaan. Kuolemista varten pelaajalle rakennettiin terveysmittari eli PlayerHealth-skripti, jolla mitataan pelaajan terveydentilaa. Pelaajan terveydentila voi heiketä pelaajan törmätessä pelikentän varrella oleviin esteisiin tai vihollisen hyökätessä. Pelaajan pelihahmoa varten luotiin animaatioita, joita hallitaan Unity 5:n Animator Controller-komponentilla. Myöhemmissä pelikentissä käytetään aikarajoitusta, mikä tarkoittaa, että pelaajan tulee suorittaa pelikenttä loppuun vaaditussa ajassa. Kolikoita keräämällä pelaaja voi saada lisäaikaa pelikentän suorittamiseen.

Pelaajan kontrolloimista varten luotiin PlayerControl–skripti. Pelaajan liikkumista varten luotiin move-metodi (Kuvio 2), johon on määritelty liike eteenpäin. Kyseisessä metodissa yleiseksi muuttujaksi määriteltiin forwardspeed-muuttuja. Yleiseksi muuttujaksi määriteltyjä

muuttujia voi muokata Unity 5:n Inspector-näkymässä, jotta muuttujan arvo on helposti hienosäädettävissä peliin sopivaksi. Kääntymiselle määriteltiin turn-metodi, joka suoritetaan, jos pelaaja painaa näppäimistöltä Horizontal-syötteen vastaavia näppäimiä. Horizontal-syötteen vastaavilla näppäimillä tarkoitetaan näppäimistöllä A- ja D-näppäimiä tai vasemman- tai oikeanpuoleista nuolinäppäintä. Move- ja turn-metodit suoritetaan Update-funktiossa, joka suoritetaan jokaisella kuvakehyksellä.

```
public float forwardSpeed;
public float turnSpeed;

void Update (){
    turn();
    move();
}

private void move (){
    transform.position += transform.forward * Time.deltaTime * forwardSpeed;
}

private void turn(){

    if(Input.GetAxis("Horizontal") < 0){
        transform.Rotate(-Vector3.up * turnSpeed * Time.deltaTime);
    }
   else if (Input.GetAxis("Horizontal") > 0){
        transform.Rotate(Vector3.up * turnSpeed * Time.deltaTime);
    }
}
```

Kuvio 2: Pelaajan alustava liikkuminen PlayerControl-skriptissä

Pelaaja liikuttaa laivan keulassa olevaa asetta hiirellä. Aseen halutaan liikkuvan vain, jos pelaaja liikuttaa hiirtä oikealle tai vasemmalle. Asetta ei haluta kohdistaa pelaajan laiva-peliobjektia kohti. Tämän takia aseen kääntymiseen määriteltiin maksimikulma, jonka se voi kääntyä. Pelaaja-objektin lapsiobjektiksi lisättiin 3D-mallinnettu ase. Ase-objektin liikuttamista varten luotiin Shoot-skripti, joka liitettiin aseeseen.

Aseen liikuttamista varten luotiin mousefollow-metodi (Kuvio 3) Shoot-skriptiin. Metodissa rotationY-muuttuja määritetään hiiren X-akselin suuntaiselle liikuttamiselle. Tällä tarkoitetaan hiiren liikuttamista vasemmalle tai oikealle. Aseen kääntymisen herkkyyden säätämistä varten määriteltiin sensitivity-muuttuja. RotationY-muuttujaan määriteltiin maksimi ja minimi arvo käyttämällä Mathf.Clamp-funktiota. Funktioon on määritelty rotationY-muuttujan minimiarvoksi -60 ja maksimi arvoksi 60.

Jokaisella peliobjektilla on Transform-komponentti. Sitä käytetään pelaajan positioon, rotaatioon ja skaalaukseen(Unity Technologies 2015). Transform-komponentti sisältää yleisen loca-

lEulerAngles-muuttujan, jonka avulla mousefollow-metodissa aseen kääntyminen on määritelty suhteessa pääpeliobjektiin. Pelaajan peliobjekti pystyy halutessaan kääntymään 360-astetta, jonka vuoksi aseen kääntyminen täytyi tehdä suhteelliseksi pääpeliobjektiin nähden.

```
void mousefollow (){

    rotationY += Input.GetAxis("Mouse X") * sensitivity;
    rotationY = Mathf.Clamp(rotationY, -60, 60);
    transform.localEulerAngles = new Vector3(0, rotationY,0);
}
```

Kuvio 3: Mousefollow-metodi

Pelaaja ampuu painamalla hiiren osoitinta. Ampumista varten ase-peliobjektiin luotiin uusi tyhjä SpawnPoint-niminen peliobjekti. SpawnPoint-peliobjekti asetettiin aseen piippuun ja siihen kohtaan syntyy luoti pelaajan painaessa Fire1-vastaisen syötteen painiketta eli hiiren vasemmanpuoleista painiketta. Ampumista varten luotiin bullettime-muuttuja, jonka arvoa käytetään määrittämään kuinka usein pelaaja pystyy ampumaan.

```
void Update(){

    if (Input.GetButton("Fire1") && Time.time > bullettime){

        bullettime = Time.time + timeBetweenBullets;
        GameObject clone = (GameObject)Instantiate(shot, shotSpawn.position, shotSpawn.rotation);
        Destroy(clone, 2);
    }
}
```

Kuvio 4: Ampuminen

Shoot-skriptin shot-muuttuja määriteltiin yleiseksi peliobjektiksi, joka näkyy Inspector-näkymässä. Shot-peliobjektia varten luotiin Prefab-peliobjekti, jota käytetään luotina. Prefab-peliobjektista syntyy klooni pelaajan ampuessa. Shoot-skriptiin määriteltiin kloonin tuhoutuminen. Luotia varten tehty Prefab-peliobjekti sisältää bullet-skriptin, johon määriteltiin luodin nopeus ja liikkuminen eteenpäin. Esteitä tai vihollisia varten tehtyyn EnemyBox-skriptiin määriteltiin, mitä tapahtuu luodin osuessa niihin.

4.2.1 PlayerHealth-skripti

Pelaajan tulee pystyä häviämään, eli kuolemaan, pelissä. Pelaajan ei haluta kuolevan ensimmäisestä vihollisen hyökkäyksestä. Sitä varten luotiin PlayerHealth-skripti. Ensimmäiseksi skriptissä määriteltiin tarvittavat muuttujat terveydentilan määrittämiseksi (Kuvio 5). StartingHealth-muuttuja määrittää pelaajan terveydentilan pelin alussa. CurrentHealth-muuttuja määrittää pelaajan nykyistä terveydentilaa. Pelin aikana pelaajan terveydentilan kuvaamista

varten käytetään Unity 5:n käyttöliittymän Slider-komponenttia. Pelaajan terveydentilan heikentymistä varten luotiin PlayerLoseHp-niminen animaatio ja pelaajan kuolemista varten PlayerDies-animaatio. Animaatiot luotiin Unity 5:n Animation-ikkunassa. Pelaajan Animator Controller-komponenttiin määriteltiin LoseHP- ja Die -parametrit animaatioiden suorittamiseksi skriptin kautta.

```
public class PlayerHealth : MonoBehaviour
{
    public int startingHealth = 100;
    public int currentHealth;
    public Slider slider;
    BoxCollider box;
```

Kuvio 5: PlayerHealth-skriptin terveydentilan määrittelevät muuttujat

PlayerHealth-skriptin ensimmäisenä funktiona käytetään Awake-funktiota (Kuvio 6). Tämä funktio suoritetaan ennen Start- ja Update-funktioita (Unity Technologies 2015). Update-funktioon määriteltiin aikaisemmin muun muassa pelaajan liike. Awake-funktiossa haetaan kaikkien tarvittavien komponenttien viittaukset, jotta niitä pystytään käyttämään skriptissä. CurrentHealth-muuttujan määrä alustetaan vastaamaan startingHealth-muuttujan arvoa pelin alussa. Pelaajan peliobjekti haetaan FindGameObjectWithTag-funktiolla, joka etsii Player-merkityn peliobjektin. GetComponent-funktiolla haetaan pelaajan komponentti PlayerControl-skripti.

```
Animator animator;
PlayerHealth playerhealth;
PlayerControl playercontrol;
GameObject player;
BoxCollider box;

void Awake(){

    animator = GetComponent<Animator>();
    box = GetComponent<BoxCollider>();

    player = GameObject.FindGameObjectWithTag("Player");
    playercontrol = player.GetComponent<PlayerControl>();

    currentHealth = startingHealth;
}
```

Kuvio 6: PlayerHealth-skriptin Awake-funktio

PlayerHealth-skriptiin tehtiin playerLoseHealth-metodi pelaajan altistuessa vihollisen hyökkäykselle ja pelaajan kuolemista varten playerDies-metodi (Kuvio 7). Metodien edessä on

määritelmä public, mikä tarkoittaa, että kyseisiä yleisiä metodeja voi kutsua muiden peliobjektien skripteistä. Pelaaja menettää pelihahmon kontrolloinnin kuollessaan. Tämän takia playerDies-metodia suoritettaessa PlayerControl-skripti poistetaan käytöstä, jotta pelaaja ei enää pysty kontrolloimaan pelaajaa. Metodi laukaisee Animation Controller-komponenttiin määritetyn Die-parametrin, joka aktivoi pelaajan Playerdies-animaation. Pelaajan terveydentilan heiketessä LoseHP-parametri aktivoi PlayerLoseHp-nimisen animaation. PlayerHealth-skriptiin määriteltyjä metodeja ja currentHealth-muuttujaa kutsutaan vihollisen EnemyBox-skriptistä.

```
public void playerLoseHealth(){

    slider.value = currentHealth;
    anim.SetTrigger("LoseHP");

}
public void playerDies(){

    box.enabled = false;
    playercontrol.enabled = false;
    anim.SetTrigger("Dies");
}
```

Kuvio 7: PlayerLoseHealth- ja playerDies-metodit

Testauksen yhteydessä pelaajan kuollessa Playerdies-animaatio toistettiin useaan kertaan. Tämä ei ollut haluttu lopputulos, koska kuolema-animaatio halutaan suorittaa vain kerran. Kyseistä ongelmaa varten playerDies-metodi suoritettaessa deaktivoi pelaajan Box Collider-komponentin, jotta pelaaja ei pysty enää menettämään nykyisestä terveydentilastaan tai kuolemaan uudelleen.

4.2.2 Vihollisen EnemyBox-skripti

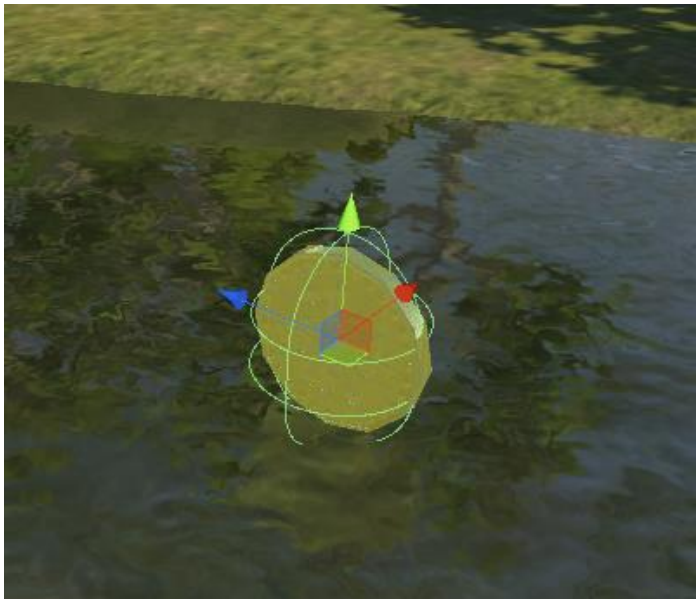
Pelaajan PlayerHealth-skriptin luomisen jälkeen luotiin viholliselle EnemyBox-skripti (Kuvio 8). Skriptistä kutsutaan PlayerHealth-skriptiin luotuja metodeja pelaajan altistuessa vihollisen hyökkäykselle tai pelaajan kuollessa. EnemyBox-skriptin Awake-funktiossa haetaan pelaajan peliobjekti ja viitataan sen komponenttiin PlayerHealth-skriptiin, jotta voidaan käyttää siihen luotuja metodeja. EnemyBox-skriptin pääajatuksena on, että jos vihollisen Collider-komponentti osuu Player-merkittyyn peliobjektiin eli pelaajaan, pelaaja menettää määritellyn damages-muuttujan arvon verran nykyisestä terveydentasosta eli currentHealth-muuttujasta. Pelaajan osuessa viholliseen, vihollisen peliobjekti tuhotaan ja tuhoutumisen yhteydessä suoritetaan efekti. Efekti suoritetaan käyttämällä Instantiate-funktiota.

```
    void OnTriggerEnter(Collider other) {
        if (other.gameObject.CompareTag("Player") && playerHealth.currentHealth > 0) {

            playerHealth.currentHealth -= damages;
            playerHealth.playerLoseHealth();

            Destroy(gameObject);
            GameObject boxiclone = (GameObject)Instantiate(explosion, transform.position, transform.rotation);
            Destroy(boxiclone, 2);
        }
        else if (playerHealth.currentHealth <= 0){
            playerHealth.playerDies();
        }
       else if (other.gameObject.CompareTag("bullet"))
        {
            Destroy(gameObject);

            GameObject bulletexplosion = (GameObject)Instantiate(explosion, transform.position, transform.rotation);

            Destroy(bulletexplosion, 1);


        }
    }
}
```

Kuvio 8: EnemyBox-skripti

4.2.3 Pisteiden kerääminen

Pelaaja voi kerätä pelin aikana kolikoita, joista kertyy pisteitä. Pisteitä varten luotiin uusi tekstiobjekti, jotta pelaaja voi seurata pisteiden kertymistä. Kolikko-peliobjektia varten luotiin uusi tyhjä peliobjekti, jonka sisälle tuotiin alustavasti 3D-mallinnettu kolikko. Kolikkoon lisättiin pelaajalle näkymätön Capsule Collider-komponentti (Kuvio 9), minkä avulla kolikko tunnistaa fyysisen yhteentörmäyksen. Skriptin avulla määritettiin mitä tapahtuu, kun pelaaja osuu kolikon Capsule Collider-komponenttiin.



Kuvio 9: Kolikko ja Capsule Collider-komponentti

OnTriggerEnter(Collider other)-funktiota kutsutaan, kun toisen peliobjektin Collider osuu peliobjektiin (Unity Technologies 2015). Peliobjekteja on mahdollista merkitä CompareTag()-funktiolla. PlayerControl-skriptiin on määritelty, mitä tapahtuu pelaajan Collider-komponentin osuessa Pick Up-merkityn peliobjektin Collider-komponenttin (Kuvio 10). Pelaajan osuessa kolikon Collider-komponenttiin, kolikko deaktivoituu ja countScore-metodi suoritetaan. CountScore-metodissa pisteitä varten tehty teksti-objektin teksti määritetään Score-sanaksi ja count-muuttujan määrää kasvatetaan kymmenellä.

```
private void Start(){
    countScore();
    rb = GetComponent<Rigidbody>();
    count = 0;

}

void OnTriggerEnter(Collider other){
    if (other.gameObject.CompareTag("Pick Up"))
    {
        other.gameObject.SetActive(false);
        count = count + 10;
        countScore();

        GameObject coinclone = (GameObject)Instantiate(coinplus, transform.position, coinplus.transform.rotation);
        Destroy(coinclone, 1);
    }
}
    public void countScore(){
    countText.text = "Score: " + (int)count;
}
```

Kuvio 10: Kolikon kerääminen

4.2.4 Unity 5:n navigointityökalu

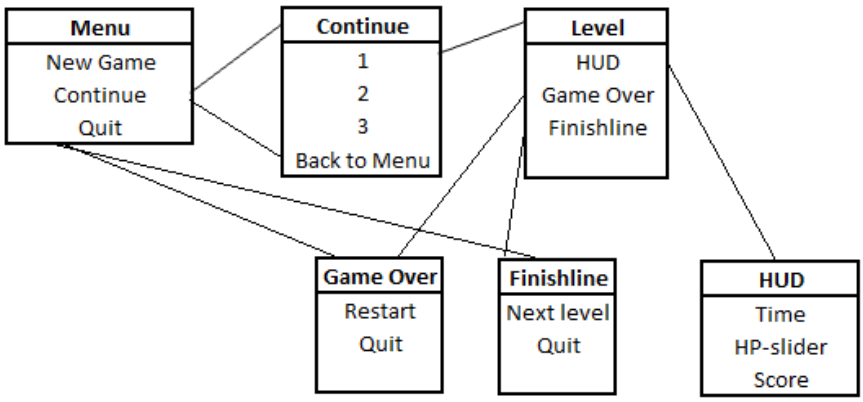
Unity 5 sisältää NavMesh-navigointityökalun, jonka avulla luodaan alueita, joissa peliobjektit voivat liikkua. Työkalu antaa pelihahmoille kyvyn ymmärtää, että niiden tulee kävellä portaita päästäkseen toiseen kerrokseen tai hypätä päästäkseen ojan yli. Navmesh Agent on peliobjekteihin lisättävä komponentti, jonka avulla pelihahmot liikkuvat kohteisiinsa ja välttelevät yhteentörmäyksiä toisiinsa. (Unity Technologies 2015.)

Pelissä ohjattavan laivan vesialue määriteltiin NavMesh-työkalulla. NavMesh-työkalulla pystyy määrittelemään eriarvoisia alueita. Eriarvoiset alueet kuvataan eri väreillä. Jokaiselle liikkuvalle viholliselle määriteltiin oma alueensa: vihreä, violetti ja harmaa (Kuvio 11). Pelaajan pääasiallinen reitti on kuvattu sinisellä. Pelaajan mennessä viholliselle määritellyn alueen sisälle, vihollinen aktivoituu ja lähtee liikkumaan pelaajaa kohti sekä yrittää ampua sitä. Vihollisille määriteltiin eriarvoiset alueet, jotta pelaajan mennessä viholliselle määritellylle alueelle, se ei aktivoisi niitä kaikkia samaan aikaan.

Kuvio 11: Unity 5:n NavMesh-työkalu

4.3 Pelikäyttöliittymän suunnittelu ja toteutus

Pelikäyttöliittymä käsittää kaiken aina alkuvalikoista pelin aikana ilmestyviin valikkoihin. Kirjassa Game Interface Design (Fox 2004, 13) suositellaan aloittamaan pelikäyttöliittymän suunnittelu tekemällä vuokaavio pelin aikana ilmestyvistä valikoista. Opinnäytetyön rajallisen aikataulun takia tässä pelikehitysprosessissa toteutettiin yksinkertainen menurakenne (Kuvio 12). Menu sisältää kaksi sivua, pää- ja kenttäsivun. Pääsivulta Continue-painiketta painaessa siirrytään Continue-kohtaan. Pelaaja pystyy avaamaan lukittuja pelikenttiä suorittamalla aina edellisen pelikentän loppuun.



Kuvio 12: Pelikäyttöliittymän vuokaavio

Kuviossa 12. Menu-kohta kuvaa pelin alussa ilmestyvää valikkoa. Pelaaja pystyy halutessaan siirtymään Continue-kohtaan, josta voi seurata pelin etenemistä tai jatkaa pelaamista tietystä pelikentästä. Jos pelaaja häviää pelin, hänen on pystyttävä aloittamaan kenttä uudelleen tai lopettamaan peli. Tullessaan maaliin pelaajalla täytyy olla mahdollisuus siirtyä seuraavaan pelikenttään tai lopettamaan peli.

HUD-sanalla tarkoitetaan pelin aikana näytettäviä käyttöliittymän tietoja, joita pelaaja tarvitsee (Fox 2004, 24–25). Tässä pelissä tärkeimpiä tietoja ovat peliaika, pisteet ja pelaajan terveydentila (Kuvio 13). HUD:in suunnittelussa tärkeintä oli tietojen sijoittelu ruudulle. Peliaika näytetään ruudun yläreunassa, keskellä ja numeroina. Pisteet sijoitettiin oikeaan yläreunaan. Pelaajan terveydentilaa varten käytetään Unity 5:n käyttöliittymätyökaluihin kuuluvaa Slider-komponenttia, joka sijoitettiin vasempaan alareunaan.

Kuvio 13: Alustava HUD

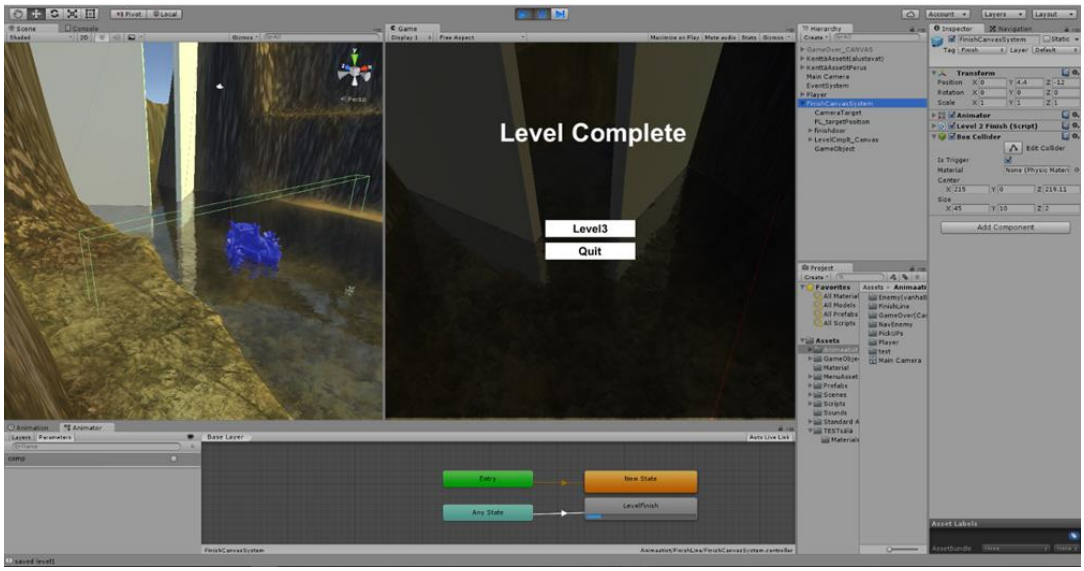
Menua suunniteltaessa otettiin huomioon tarvittavien painikkeiden määrä ja niiden sijoittelu käyttöliittymässä. Pelimenussa on alustavasti kolme painikevaihtoehtoa New Game-, Continue- ja Quit-painikkeet. Pelaajan valitessa Continue-painikkeen, menu-näkymä vaihtuu Continue-näkymään (Kuvio 14). Continue-näkymä tulee sisältämään kaikki pelikentät, jotka pelissä on mahdollista saavuttaa. Continue-näkymän suunnittelussa oli tärkeää huomioida, kuinka monta pelikenttää peliin mahdollisesti toteutetaan. Näkymä tehtiin alustavasti kahdeksalle pelikentälle. Tässä pelikehitysvaiheessa ei tiedetä tarkkaa kenttien lukumäärää, mutta Continue-näkymä suunniteltiin helposti muokattavaksi.

Kuvio 14: Alustava Continue-näkymä

Ennen varsinaisten pelikenttien luontia suunniteltiin pelikenttien loppunäkymä, joka näkyy kentän suorittamisen jälkeen (Kuvio 15). Pelaajan tulee kentän läpäisemiseksi löytää ovi, jonka jälkeen kenttä on suoritettu. Kentän suorittamista varten luotiin tyhjä FinishCanvasSystem-peliobjekti, johon lisättiin peliobjekteja ja komponentteja, joita suoritetaan pelaajan tullessa maaliin.

Pelaajan saavuttua maaliin suoritetaan Animation-näkymässä luotu LevelFinish-animaatio. LevelFinish-animaation suorittamiseksi oven lähelle lisättiin pelaajalle näkymätön Collider-komponentti. LevelFinish-animaation hallitsemista varten käytetään Animator Controller-komponenttia, johon määriteltiin comp-parametri. FinishCanvasSystem-objektiin lisättiin skripti, johon määriteltiin, että pelaajan osuessa Collider-komponenttiin comp-parametri suoritetaan. Comp-parametri suorittaa LevelFinish-animaation.

Pelikentän läpäisemisen jälkeen pelaajan tulee pystyä siirtymään seuraavaan pelikenttään tai lopettamaan peli. Pelaajan kuollessa näytetään Game Over-käyttöliittymä. Tämä toteutettiin samoilla toiminnoilla kuin kentän suorittamisen jälkeen näytettävä käyttöliittymä.



Kuvio 15: Pelikentän suorittaminen

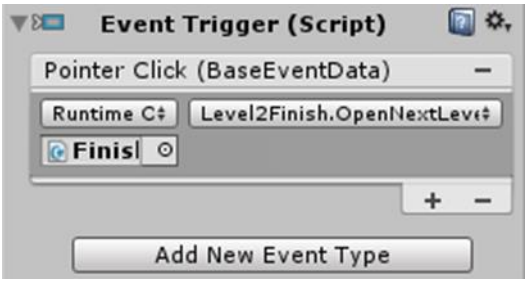
Unity 5:ssä on käyttöliittymätyökalu, jonka avulla voidaan luoda yksilöllisiä käyttöliittymiä nopeasti. Canvas on alue, jonka sisälle kaikki käyttöliittymän elementit tulee tehdä. (Unity Technologies 2015.) FinishCanvasSystem-peliobjekti sisältää LevelCompleteCanvas-nimisen Canvas-komponentin, johon lisättiin Level Complete-tekstikenttä ja Level3 – ja Quit-painikkeet.

Painikkeita varten FinishCanvasSystem-peliobjektin skriptiin lisättiin kaksi metodia. Näitä metodeja käytetään muissakin pelikentissä. OpenNextLevel-metodi hakee aktiivisen pelikentän numeron ja lisää siihen arvon yksi. Metodia käytetään Level3-painikkeessa (Kuvio 16) ja sama metodi soveltuu muihinkin skeneihin. Quit-painikkeelle määriteltiin exitToMenu-metodi, joka suoritettaessa lataa Menu-nimisen skenen.

```
public void openNextLevel()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
public void exitToMenu()
{
    SceneManager.LoadScene("Menu");
}
```

Kuvio 16: OpenNextLevel- ja exitToMenu-metodit

Level3- ja Quit-painikkeisiin lisättiin Event Trigger-komponentti (Kuvio 17). FinishCanvasSystem-peliobjekti liitettiin kyseiseen komponenttiin, jotta painikkeita varten tehdyt metodit voidaan suorittaa. Tämän jälkeen painiketta painettaessa, siihen määritetty metodi suoritetaan. Event Trigger-komponenttia käytettiin myös Menu ja Continue-näkymien painikkeiden luomisessa.



Kuvio 17: Event Trigger-komponentti

4.4 Kenttäsuunnittelu

Kentän prototyyppi suunniteltiin luomalla kentälle alustava tarinankerronta. Pelissä on oma teemansa, jota noudatettiin kenttäsuunnittelussa. Kentässä on joki, jota pitkin pelaaja liikkuu aluksellaan. Pelaajan on päästävä maaliin, joka on kuvattu pelissä porttina. Joen varrella suunniteltiin vihollisia ja esteitä pelaajalle. Joen ympärillä suunniteltiin maasto ja kasvillisuus. Kenttä pyrittiin tekemään lineaariseksi, jottei pelaaja tylsistyisi eikä peli tulisi liian raskaaksi tietokoneelle.

Tarinankerronta antaa hyvän alkuasetelman kentän suunnitteluun. Sen jälkeen tehtiin listaus kentälle tulevista objekteista. Tätä listaa hyödyntämällä tehtiin alustava piirros kentästä lintuperspektiiviä hyödyntäen. Piirrokseen merkittiin paikat ja objektit kentässä. Tässä vaiheessa sijoittelu tehtiin siten, ettei kenttiin tule alueita, jotka ylikuormittavat tietokoneen toimintaa huomattavasti.

Kentän suunnittelun jälkeen aloitettiin sen kehittämisvaihe. Kun suurin osa objekteista olivat valmiita kenttää varten, aloitettiin kentän rakentaminen Unity 5-pelimoottorilla. Ensimmäiseksi luotiin alusta eli Unity 5:n Terrain-objekti, jota muokkaamalla saatiin kenttään tasoeroja. Sen jälkeen tuotiin vesistö kenttään ja sijoiteltiin objekteja kenttään suunnittelupiirroksen mukaisesti.

Joet ja vesistö ovat suuressa roolissa pelikentässä. Kentän ympäristö sisältää paljon puita, kallioita ja kasvillisuutta. Realistista näkymää ei tehty lyhyen aikataulun takia, mutta graafisesti kaunista maisemaa tavoiteltiin peliin. Oikeita jokimaisemia tutkittiin ja otettiin talteen kuvia kolmiulotteisten objektien ja kenttien luonnin tueksi (Kuvio 18).

Kuvio 18: Vesistöreferenssi

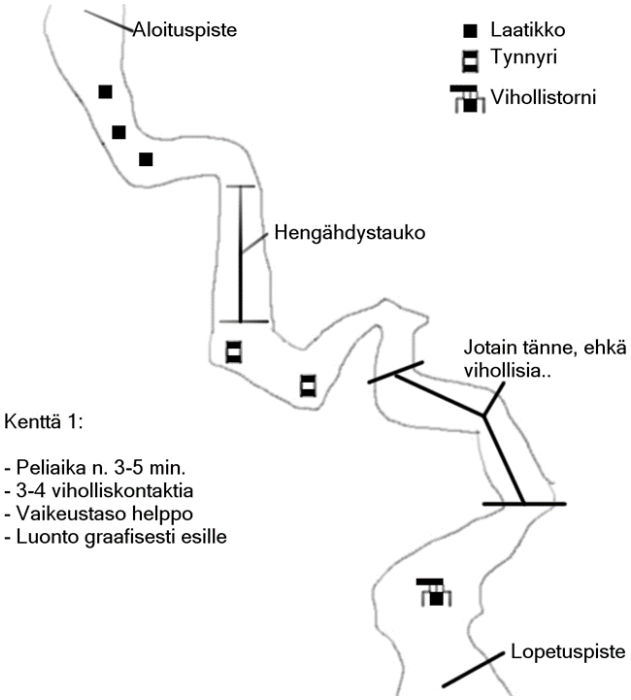
4.4.1 Ensimmäinen kenttä-suunnitelma

Ensimmäisen kentän pitää olla yksinkertainen, jotta pelaaja ymmärtäisi ja oppisi pelimekaniikan nopeasti. Pelaaja aloittaa kentän alkukohdasta ja lähtee liikkeelle. Ensimmäiseksi pelaajaa opastetaan liikuttamaan alusta, joten kentän alussa täytyy olla pieniä ja helppoja esteitä pelaajalle väisteltäväksi. Kun pelaaja on selvinnyt esteistä, kentässä on suora, jonka aikana pelaaja sisäistää oppimaansa. Tämän jälkeen pelaajalle esitellään vihollisista räjähtävät tynnyrit. Tynnyri-objekteja sijoitetaan yksi kerrallaan pelaajan reitille ja pelaajalle opastetaan, että tynnyreitä voidaan väistää tai ampua. Kentän lopuksi pelaajalle esitetään vihollistorni, joka ampuu pelaajaa. Pelaajan on tuhottava vihollistorni ampumalla tämä. Tämän jälkeen pelaaja opastetaan porteille, jonka jälkeen kenttä on suoritettu. Kun ensimmäisen kentän tarina oli tehty, tehtiin sen pohjalta taulukko (Taulukko 1). Taulukko kertoo, mitä objekteja kenttään tulee.

Pelaaja	**Luonto**	**Viholliset**	**Rakennukset**
Alus	Puu	Tynnyri	Laatikko
	Kivi	Torni	Kivilaituri
	Ruoho		Lopetusovi
	Pensas		

Taulukko 1: Ensimmäisen kentän 3D-objektit

Ensimmäisestä kentästä luodusta taulukosta pystyttiin rakentamaan lintuperspektiivistä kuva kentästä (Kuvio 19). Ensimmäiseksi kenttään piirrettiin joki, joka merkitsee samalla kentän lineaarisen polun. Polkuun merkittiin kentän aloituspiste sekä lopetuspiste. Tämän jälkeen aloitettiin sijoittelemaan kentän tarinan ja taulukon mukaisia objekteja kuvaan. Kun joen kuvaus oli valmis, aloitettiin suunnittelemaan pelikentän ulkonäköä.

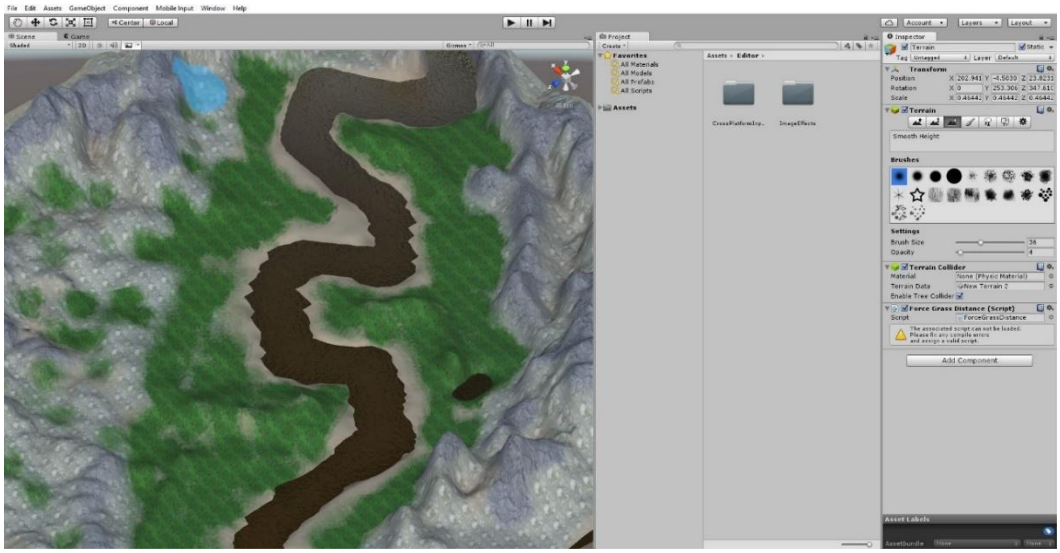


Kuvio 19: Ensimmäisen kentän suunnittelukuva

4.4.2 Kenttien luonti

Unity 5-ohjelmassa luotiin ensiksi Terrain-objekti, joka toimii pohjana varsinaiselle kentälle. Objektin ominaisuuksia muokattiin ja ensimmäiseksi määriteltiin kentän kooksi 500x500 yksikköä. Tämän jälkeen kentän tasoeroja muokattiin. Näin saatiin kenttään vuoristoa, jonka yhtenä tarkoituksena oli peittää kentän reunoja.

Kun kentän alustavat muodot oli saatu tehtyä, aloitettiin maalaamaan Terrain-objektiin tekstuureilla ulkonäköä (Kuvio 20.). Tekstuurit tehtiin toisella ohjelmalla ja ladattiin Unity 5-pelimoottoriin. Terrainin teksturoinnin jälkeen kenttään lisättiin tärkeimpiä ominaisuuksia yksi kerrallaan. Yksi tärkeimmistä sisällöistä pelissä on vesi, jota varten oli muokattu polku, missä pelaaja liikkuu. Vesiobjektin sijainti nostettiin haluttuun korkeuteen, jotta joenuomat täyttyisivät vedestä.



Kuvio 20: Kentän maalausprosessi

Tämän jälkeen kenttään lisättiin kasvillisuutta. Ruohon tekstuurit lisättiin Terrain-objektin ominaisuuksiin ja tämän jälkeen ruoho maalattiin pensseleillä kentän pinnalle. Puu-objekti lisättiin myös Terrain-objektin ominaisuuksiin ja maalattiin samalla tavalla kenttään kuin ruohokin. Kenttään lisättiin yksitellen muita luonto-objekteja, kuten kiviä.

Seuraavaksi kenttään lisättiin objektit, jotka ovat pelissä vuorovaikutuksessa pelaajaan. Näitä asioita ovat pelaajan ohjastama laiva, laatikot, öljytynnyrit, vihollistornit ja portti. Tämän jälkeen aloitettiin kentän luomisessa hienosäätövaihe, jossa parannettiin objektien lopullista sijaintia kentällä ja muokattiin pelimoottorin renderöinnin ominaisuuksia (Kuvio 21). Tässä vaiheessa yhdistettiin pelisisältö ja pelitoiminnot. Pelikentän valmistuttua sitä testattiin, jotta nähtäisiin sen toimivuus. Tärkeimmäksi kriteeriksi oli valittu pelattavuus ja viihdyttävyys. Jos kenttä ei läpäissyt näitä kriteerejä, sitä kehitettiin.

Kuvio 21: Kentän luontiprosessi

4.5 Pelisisällön tuottaminen Unity 5-pelimoottoriin

Yhtenä pelisisältönä on vihollistorni, joka pyrkii tuhoamaan pelaajan laivan ampumalla. Torni tuhoutuu, jos pelaaja on ampunut sitä tarpeeksi monta kertaa. Tornin mekaniikka rakennettiin siten, että torni pystyy kääntymään pysty- ja vaakasuuntaan. Torni-objekti luotiin kahteen erilliseen komponenttiin, kääntyilevään ja ampuvaan torniin sekä sen jalustaan.

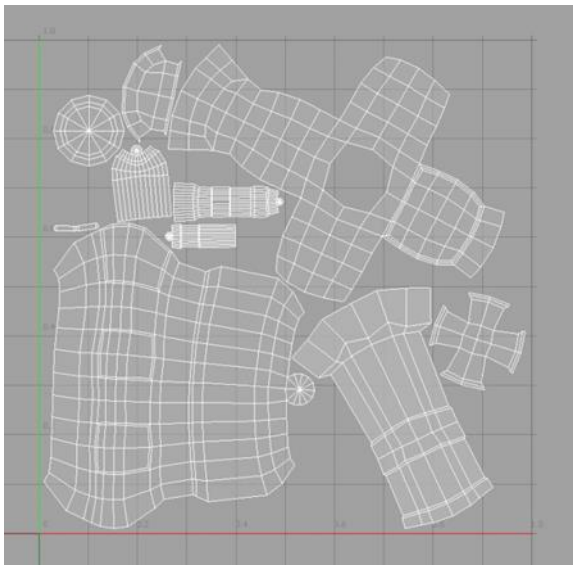
Torni-objektin tuottaminen alkoi suunnitteluvaiheesta. Tämän jälkeen luotiin Modo-ohjelmalla kolmiulotteinen objekti. Sille tehtiin UV-kartta, jonka tehtävänä on levittää kolmiulotteinen objekti kaksiulotteiseksi teksturoimista varten. Teksturoiminen tapahtui Mari-ohjelmalla ja prosessissa tuotettiin halutut tekstuurikartat objektille. Lopuksi valmis objekti ja siihen liittyvät tiedostot vietiin Unity 5-ohjelmaan, jossa luotiin lopullinen torni-asset eli peliobjekti. Tuottamisprosessia käytettiin muidenkin peliobjektien luomisessa.

Torni-objektin luominen aloitettiin Modo 3D-mallinnusohjelmassa kuution luomisella. Kuutiolle tehtiin SDS subdivide-toiminto, koska torni suunniteltiin kääntyvän x- ja y-akselillaan. Toiminto jakaa jokaisen kuution polygonin kahdella ja pyöristää valittua polygoni-joukkoa sekä kuution kulmia. Kuution lisäksi alkuvaiheen mallinnuksessa käytettiin perinteisiä muotoja, kuten sylinteriä ja torusta. Objektikokonaisuuden polygoneja ja verteksejä muokattiin, kunnes tornin siluetti alkoi hahmottua.

Yksinkertaiseen torni-objektiin lisättiin kohta kohdalta yksityiskohtia, kuten muttereita ja metallilevyjen saumoja, kunnes highpoly-objekti oli valmis. Highpoly-objekti on yksikohtainen versio peliin tulevasta objektista ja sisältää suuren määrän polygoneja. Highpoly-objektista tehtiin kopio ja sen polygonien määrää pienennettiin. Sitä pienennettiin peliin sopivaksi ja tornin sopivaksi polygoni-määräksi määriteltiin 1956 polygonia (Kuvio 22). Objektin muodon todettiin olevan tarpeeksi tarkka jäljitelläkseen highpoly-objektia.

Kuvio 22: Modo-ohjelmalla mallinnettu torni-objekti

Seuraavaksi peliin tulevalle objektille luotiin UV-kartta(Kuvio 23). UV-kartan tarkoitus on levittää objektin kolmiulotteinen koordinaatisto kaksiulotteiseen muotoon. UV-kartta mahdollistaa tekstuurien luomisen objektille. UV-kartan valmistuttua tehtiin normal map baking-toiminto, jossa peliin tulevalle objektille leivottiin normal map-tekstuuri highpoly-objektin tiedoista. Tornin tekstuuritiedostona käytettiin aluksi tga-tiedostomuotoa ja pikselien määränä on 4096 kertaa 4096.



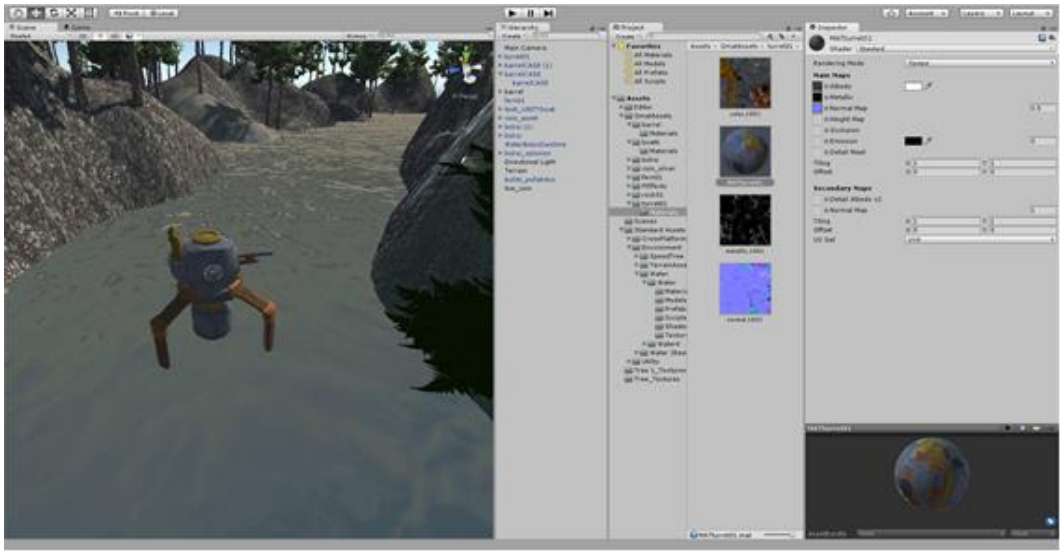
Kuvio 23: UV-kartta tornista

Torni-objekti lähetettiin Mari-ohjelmaan teksturoitavaksi fbx-tiedostona. Ohjelmassa luotiin uusi projekti ja fbx-tiedosto avattiin. Unity 5-ohjelmaa varten tehtiin kolme tekstuurikanavaa eli karttaa: color-, normal- ja metallicmap. Ensimmäinen tekstuurikartta pitää sisällään objektin materiaalin pinnan väriarvot, metallic-tekstuuri materiaalin metallipitoisuuksien arvot

ja normal-tekstuuri materiaalin pintojen syvyys- ja suunta-arvot. Jälkimmäiseen tekstuurikanavaan avattiin Modo-ohjelmassa leivottu normal map-tekstuuri ja korjattiin baking-prosessissa ilmaantuneita virheitä sekä lisättiin uusia yksityiskohtia.

Mari-ohjelma ei käytä suoraan samaa shader-tekniikkaa eli ohjelman tapaa lukea materiaaleja, kuin Unity 5-ohjelma, mutta tarjoaa erilaisia vaihtoehtoja. Tornin teksturoimiseen käytettiin Unreal-shaderia ja tekstuurikartat olivat resoluutioltaan aluksi 4096x4096. Valmiit tekstuurikartat pienennettiin sopiviksi Unity 5-ohjelmaa varten, jotta pelimoottori ei kävisi liian raskaaksi. Tekstuurikartat lähetettiin tga-tiedostoina samaan kansioon, jossa sijaitsee myös fbx-tiedosto torni-objektista.

Unity 5-ohjelmaan tuotiin valmis 3D-mallinnettu torni-objekti. Sen jälkeen lisättiin samaan kansioon halutut tekstuurit. Fbx-tiedoston mukana tulevat tiedostot, joita ei tarvittu, poistettiin. Jos materiaali on luotu objektille Modo-ohjelmassa, se tulee fbx-tiedoston mukana Unity 5-ohjelmaan. Materiaali voidaan myös luoda Unity 5-ohjelmassa. Tornin materiaalissa käytetään Unity 5:n standard shader-asetuksia ja siihen lisättiin valmiit tekstuurit. Virheiden tarkistuksen jälkeen prototyyppi torni-objetista oli valmis ohjelmoijalle (Kuvio 24).



Kuvio 24: Torni Unity 5-pelimoottorissa

4.6 Pelin hienosäätövaihe

Pelin hienosäätövaihe aloitettiin, kun peliin oli luotu tarpeeksi sisältöä ja toimintoja. Hienosäätövaiheen tarkoitus on parannella pelin nykyisiä toimintoja. Lisäksi kaikki turhat elementit, jotka eivät tuoneet pelattavuudelle lisäarvoa, poistettiin pelistä.

Pelin hienosäätövaiheessa havaittiin kehitystarpeita muun muassa pelaajan kontrolloimisessa. Pelaajan kontrolloimista kehitettiin lisäämällä pelaajalle mahdollisuus kiihdyttää peliobjektin vauhtia. Tämä paransi pelattavuutta. Lisäksi skripteihin luotuja yleisiä muuttujia säädettiin sopiviksi.

Osaa peliin toteutetuista toiminnoista tullaan käyttämään muissakin pelikentissä. Aloittelevina pelinkehittäjinä kyseistä asiaa ei osattu huomioida skriptauksen yhteydessä, jonka takia osaa skripteistä jouduttiin muokkaamaan muihinkin pelikenttiin soveltuviksi. Tästä esimerkkinä painike, joka avasi aiemmin vain tietyn pelikentän. Painikkeeseen liitettyä skriptiä muokattiin jokaiseen pelikenttään sopivaksi.

5 Yhteenveto

Opinnäytetyön tavoitteena oli tehdä toimiva prototyyppi pelisovelluksesta PC-alustalle Unity 5-pelimoottorilla. Pelisuunnittelu osoittautui tärkeäksi pelinkehitysprosessin vaiheeksi. Pelisuunnitteluteoriat antoivat hyvän pohjan tämän opinnäytetyön pelinkehitysprosessin suunnittelulle. Hyvä suunnittelu mahdollisti ohjelmoinnin aloittamisen samaan aikaan kuin peliin aloitettiin tuottamaan sisältöä. Lisäksi se helpotti ongelmatilanteiden ratkaisua.

Tässä opinnäytetyössä käytetyt pelinkehitysteoriat osoittautuivat käytettäviksi. Esimerkiksi pelinkehitysprosessin kolme tasoa olivat hyödyllisiä rajallisessa ajassa suoritettavalle opinnäytetyölle. Pelin ideoinnin jälkeen aloitettiin työstämään peliin vaadittavia toimintoja ja objekteja. Pelin toimintoja ja pelimekaniikkaa laajennettiin määriteltyyn aikarajaan asti, jonka jälkeen aloitettiin pelin hienosäätö. Hienosäätövaiheessa pelitoimintoja ja -sisältöjä hienosäädettiin peliin sopiviksi sekä poistettiin ne turhat elementit pelistä, jotka eivät tuoneet lisäarvoa pelille.

Raportti antaa suuntaa pelinkehitysprosessin suuresta työmäärästä. Opinnäytetyön tekijöinä suosittelemme aloittelevia pelinkehittäjiä aloittamaan pelinkehityksen mahdollisimman yksinkertaisesta peli-ideasta. Yksinkertaisimmassakin peli-ideassa hyvä pelisuunnittelu tukee prosessin kulkua.

Tietojenkäsittely vaatii jatkuvaa uuden oppimista ja jatkuvaa itsensä kehittämistä. Esimerkiksi uusien järjestelmien ja ohjelmien käytön opettelu ovat olennainen osa tietojenkäsittelyn alaa. Opinnäytetyöprojekti oli jatkuvaa uuden oppimista ja sen takia ongelmatilanteita syntyi jatkuvasti. Merkittäviä ongelmatilanteita ei kuitenkaan syntynyt hyvän suunnittelun takia.

Pelisovelluksen prototyyppi sisältää yhden testikentän, jota koehenkilöt testasivat. Ensimmäinen testaus suoritettiin pelinkehitysprosessin keskivaiheilla, jotta pystyttiin kehittämään pelin

toimivuutta. Testausta suorittaneet koehenkilöt kuuluivat pelin kohdeyleisöön, jonka takia heidän arvionsa pelin toimivuudesta ja ulkoasusta olivat tärkeitä. Prototyypin valmistuttua viimeisessä testauksessa selvitettiin, kuinka peli oli kehittynyt. Palautteen perusteella todettiin, että pelisovelluksen prototyyppi on kehityskelpoinen. Pelisovelluksen kehittäminen jatkuu myös tämän opinnäytetyöraportin julkaisemisen jälkeen. Tavoitteena on tuottaa kokonainen pelisovellus, joka on tarkoitus julkaista pelikäyttöön. Mahdollisena kanavana suunnitellaan käytettävän Steam-jakelua.

Lähteet

Adams, E. & Dormans J. 2012. Game Mechanics, Advanced Game Design. Berkeley: New Riders Games.

Blackman, S. 2011. Beginning 3D game development with Unity : The world's most widely used multi platform game engine. New York : Apress.

Fox, B. 2004. Game Interface Design. Boston: Course Technology / Cengage Learning.

Koivisto, M. 2012. Näin kehität WAU-palveluja?. Viitattu 4.5.2016: https://ssl.ttlry.fi:8621/sites/hetky.ttlry.mearra.com/files/u62/DIAGONAL_PaMu_Tiivistelma.pdf

Mari User Guide 2015. The Foundry Group LLC. Viitattu 1.3.2016: https://s3.amazonaws.com/thefoundry/products/mari/releases/3.0v2/Mari_3.0v2_UserGuide.pdf

Modo Online Help 2014. The Foundry Visionmongers Limited. Viitattu 1.3.2016: http://modo.docs.thefoundry.co.uk/modo/801/help/index.html

Neogames Finland Association 2016. Tietoa toimialasta. Viitattu 1.3.2016: http://www.neogames.fi/tietoa-toimialasta/

Unity Technologies 2015. Unity Documentation. Viitattu 1.3.2016: http://docs.unity3D.com/Manual/AnimationOverview.html

Valve Corporation 2016. Steam Greenlight. Viitattu 8.5.2016: http://steamcommunity.com/workshop/about/?appid=765&section=faq

Vilkka, H. 2006. Tutki ja havainnoi. Viitattu 1.3.2016: http://hanna.vilkka.fi/wp-content/uploads/2014/02/Tutki-ja-havainnoi.pdf

Kuviot

Kuvio 1: Projektin aloitus Unity 5:llä 16
Kuvio 2: Pelaajan alustava liikkuminen PlayerControl-skriptissä 17
Kuvio 3: Mousefollow-metodi 18
Kuvio 4: Ampuminen 18
Kuvio 5: PlayerHealth-skriptin terveydentilan määrittelevät muuttujat 19
Kuvio 6: PlayerHealth-skriptin Awake-funktio 19
Kuvio 7: PlayerLoseHealth- ja playerDies-metodit 20
Kuvio 8: EnemyBox-skripti 21
Kuvio 9: Kolikko ja Capsule Collider-komponentti 21
Kuvio 10: Kolikon kerääminen 22
Kuvio 11: Unity 5:n NavMesh-työkalu 23
Kuvio 12: Pelikäyttöliittymän vuokaavio 23
Kuvio 13: Alustava HUD 24
Kuvio 14: Alustava Continue-näkymä 25
Kuvio 15: Pelikentän suorittaminen 26
Kuvio 16: OpenNextLevel- ja ExitToMenu-metodit 26
Kuvio 17: Event Trigger-komponentti 27
Kuvio 18: Vesistöreferenssi 28
Kuvio 19: Ensimmäisen kentän suunnittelukuva 29
Kuvio 20: Kentän maalausprosessi 30
Kuvio 21: Kentän luontiprosessi 30
Kuvio 22: Modo-ohjelmalla mallinnettu torni-objekti 32
Kuvio 23: UV-kartta tornista 32
Kuvio 24: Torni Unity 5-pelimoottorissa 33

Taulukot

Taulukko 1: Ensimmäisen kentän 3D-objektit 28