2D-pelin toteutus Unity-pelimoottorilla

Nelli Kaikkonen

2022 Laurea

Laurea-ammattikorkeakoulu

2D-pelin toteutus Unity-pelimoottorilla

Nelli Kaikkonen

Tietojenkäsittely

Opinnäytetyö

Lokakuu, 2022

Laurea-ammattikorkeakoulu **Tiivistelmä**

Tietojenkäsittely

Tradenomi (AMK)

Nelli Kaikkonen

2D-pelin toteutus Unity-pelimoottorilla

Vuosi 2022 Sivumäärä 31

Opinnäytetyön tavoitteena oli kehittää yksinkertainen 2D-peli Unity-pelimoottorin avulla. Opinnäytteessä tutustuttiin muun muassa pelinkehitykseen sekä Unity-pelimoottoriin ja sen toimintaan. Työllä ei ole toimeksiantajaa, joten sen hyöty ja tavoitteet olivat enemmin henkilökohtaisia. Opinnäytetyö on suunnattu pelinkehityksestä kiinnostuneille.

Peli toteutettiin C#-ohjelmointikielellä. Peliprojektin koodieditorina hyödynnettiin Visual Studio Code -ohjelmaa. Pelin sprite-grafiikat toteutettiin Pixilart-ilmaistyökalulla.

Opinnäytetyö koostuu kahdesta osuudesta: teoria- ja toteutusosioista. Teoriaosuudessa käsitellään pelinkehitystä yleisesti sekä verrataan 2D- ja 3D-pelinkehityksen ominaisuuksia. Toteutusosiossa käydään läpi peliprojektia vaihe vaiheelta. Pelin kehitysmenetelmänä hyödynnettiin muun muassa Scrum-metodologiaa.

Peliprojekti oli luonteeltaan ensimmäinen peliprojekti, eikä sen taustalla ollut aiempaa kokemusta pelinkehityksen parissa. Pelin ohjelmointia kuitenkin helpotti aiempi ohjelmointikokemus. Projektin haastavista tilanteista riippumatta peliprojektin lopputulokseksi saatiin toiminnallinen 2D-peli. Projektin perusteella voidaan sanoa, että Unity-pelimoottori soveltuu ensimmäisen peliprojektin kehitystyökaluksi.

Asiasanat: pelinkehitys, peliohjelmointi, Unity

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor's Degree

Nelli Kaikkonen

Development of a 2D Game with Unity Game Engine

Year 2022 Pages 31

The goal of this Bachelor's thesis was to develop a simple 2D game with Unity. The thesis looked into 2D game development as well as Unity and its functions. The thesis does not have a commissioner, so its benefits and goals were more so of the personal variety. The thesis is aimed at people who are interested in game development.

The game was developed using the code language C#. The game project utilized Visual Studio Code as its code editor. The game's sprite graphics were created using a free tool called Pixilart.

The thesis consists of two sections: theory and execution. The theory focuses on game development as a whole as well as comparisons between 2D and 3D game development. In the execution segment the game project will be presented phase by phase. As a part of the game's development, Scrum methodology was utilized, among others.

By its nature the game project was a first game project, and thus did not have previous game development experience backing it. Programming the game however was aided by previous coding experience. Despite the project's challenges, a functional 2D game was successfully developed as its outcome. Based on this game project, the conclusion that Unity game engine is well suited to be used as a development tool for a first game project can be made.

Keywords: game development, game programming, Unity

Sisällys

1 Johdanto

Kiinnostus pelinkehitykseen on lähtöisin omasta pitkäaikaisesta kiinnostuksestani videopeleihin, joka muuntui ajan myötä kiinnostukseksi pelien sisäisiä rakenteita ja toimintoja kohtaan. Kiinnostus kasvoi tutkintoni edetessä, sillä opin muun muassa ohjelmointikieliä ja niiden kaikenkirjavia käyttötapoja. Pelien taustalla tapahtuva kehitys alkoi tuntua konkreettisemmalta ja ymmärrettävämmältä, jolloin syntyikin ajatus oman ensimmäisen pienimuotoisen peliprojektin aloittamisesta.

Opinnäytetyössä käsitellään pelinkehitystä oman peliprojektin kautta. Opinnäytetyön tavoitteena oli tutustua pelinkehitysprosessiin sekä Unity-pelimoottoriin kehittämällä toiminnallinen 2D-peli. Pelin kehityksessä hyödynnettiin Unity-pelimoottoria sekä C#-ohjelmointikieltä pelin toiminnallisuuksien rakentamisessa. Opinnäytetyö käy läpi pelin kehitysprosessia ja sen vaiheita. Lisäksi opinnäytetyössä käsitellään 2D-pelinkehitystä sekä Unity-pelimoottoria pelinkehitysympäristönä ja sen ominaisuuksia.

Opinnäytetyön teoriapohjaa käsitellään erityisesti käytännön pelinkehityksen näkökulmasta. Opinnäytetyössä ei perehdytä pelinkehitykseen syvemmin, vaan katsaus keskittyy peliprojektin kannalta olennaiseen. Peliprojektin käsitteleminen opinnäytetyössä tulee myös päättymään ensimmäiseen toiminnalliseen versioon kehitettävästä pelistä. Opinnäytetyö kehittää pelille jatkokehitysideoita, mutta opinnäytetyössä ei käsitellä peliprojektin kehitystä tästä eteenpäin.

2 Pelinkehitys

Pelinkehitys on prosessi, jossa peli viedään ajatuksesta toiminnalliseksi toteutukseksi. Pelinkehitykseen kuuluu kaikki pelin ideoinnista ja suunnittelusta sen toteutukseen ja julkaisuun. Pelillä itsessään tarkoitetaan sitä, että pelaaja on vuorovaikutuksessa olemassa olevan sisällön kanssa. Tätä vuorovaikutusta voidaan määrittää erinäisten sääntöjen ja päämäärien avulla. Erityisesti videopeleissä, eli laitteilla pelattavissa peleissä, vuorovaikutteisuus tapahtuu muun muassa audiovisuaalisuuden avulla. Pelille olennaista on myös se, että pelaajan on mahdollista vaikuttaa pelin kulkuun. (freeCodeCamp 2019; Chandler & Chandler 2011, 1-2.)

Pelinkehitykseen kuuluu monenlaisia rooleja, kuten ohjelmoija, artisti tai testaaja. Olennainen rooli monissa peliprojekteissa on myös julkaisijoilla. Julkaisijan tehtävänä on huolehtia pelin markkinoinnista ja jakelusta. Pelinkehitystä tapahtuu itsenäisten tekijöiden

sekä suurempien pelistudioiden toimesta. Erityisesti itsenäisen pelinkehityksen suosio kasvoi pelimoottorien kehityksen myötä, sillä ne mahdollistivat pelien kehittämisen ilman kattavaa tietoa esimerkiksi peliohjelmoinnista. (freeCodeCamp 2019; Chandler & Chandler 2011, 13.)

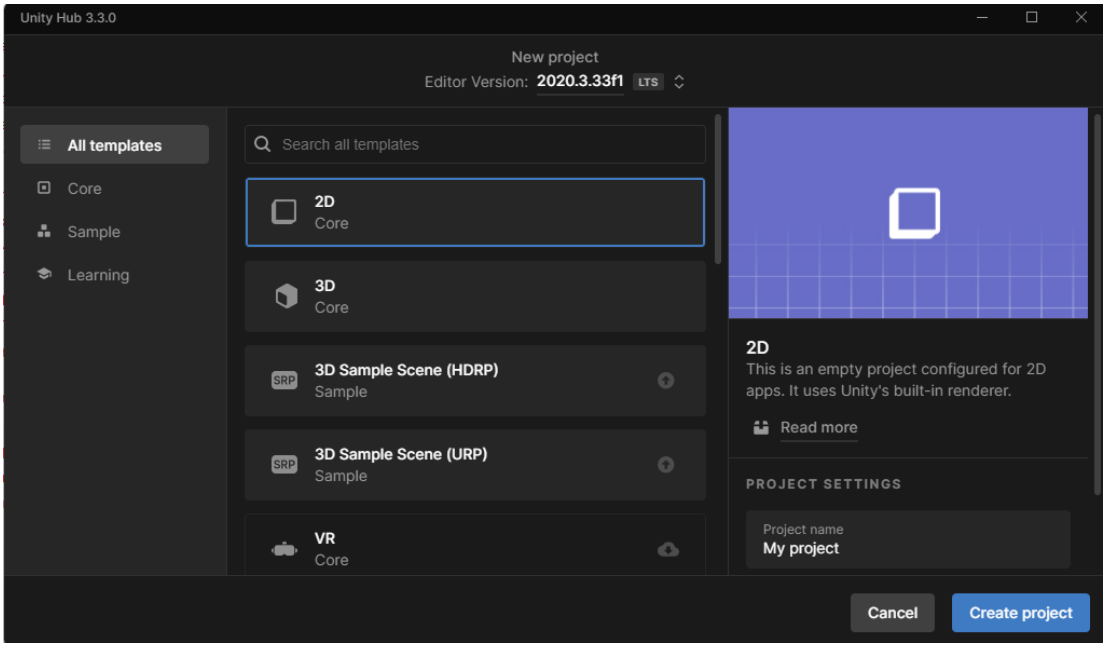
Bondin (2018, 92) mukaan tärkeä osa hyvää pelinkehitystä on iteratiivisen suunnittelun prosessi. Iteratiiviseen suunnitteluun kuuluu neljä vaihetta, jotka ovat analyysi, suunnittelu, toteutus ja testaus. Analyysivaiheessa on tarkoitus selvittää päämäärä ja saatavilla olevat resurssit projektille. Analyysivaiheessa on myös hyvä saada vastaus siihen, kenelle peli on tarkoitettu sekä tutustua samanlaisiin peleihin ja niiden kehitysratkaisuihin. Suunnitteluvaihe kattaa kaiken ideoinnista varsinaiseen suunnitelmaan. Toteutusvaiheessa on tarkoitus luoda toiminnallinen peli suunnitelman pohjalta. Lopulta testausvaiheessa peli tuodaan ihmisten pelattavaksi, ja kerätään palautetta pelin toiminnasta. Iteratiivisen suunnittelun mukaan, testausvaiheesta siirrytään takaisin analyysivaiheeseen, jossa käydään läpi testaajien palautetta ja siirrytään uudestaan prosessin seuraaviin vaiheisiin. (Bond 2018, 92, 98.)

2.1 2D-pelinkehitys vs. 3D-pelinkehitys

3D-pelit ovat myöhempi lisäys pelinkehityksen maailmaan. 2D-pelit olivat 1990-luvulle saakka videopelien oletusformaatti. 3D-teknologian kehitys muutti tämän asetelman, ja nykyään 3D-pelit ovat hyvin yleisiä ja 3D-teknologia erittäin kehittynyttä. 2D-pelit ovat kuitenkin myös yhä hyödynnetty peliformaatti. 3D-pelinkehitykseen verrattuna 2D-pelinkehitys vaatii vähemmän resursseja ja on yleisesti vähemmän monimutkaista, sillä se ei vaadi niin paljon kehittämistä esimerkiksi pelihahmon ja kameran ohjauksen osalta. (Starloop 2022.)

Unity-pelimoottorin tarjoama tuki ja työkalut eroavat 2D- ja 3D-pelinkehityksessä. Unity tarjoaa peliprojektia luodessa mahdollisuuden valita 2D ja 3D:n välillä. Oheisessa kuviossa 1 esitetään uuden Unity-projektin luomista, jossa vaihtoehtona ovat muun muassa 2D- ja 3D-mallipohjat. 2D-projektin valitseminen tuo kehitysympäristöön valmiiksi 2D-pelinkehitykselle avuliaita ominaisuuksia. Esimerkiksi kameran oletusasetukset ovat säädetty 2D-pelinkehityksen mukaiseksi. Kamera on asetettu ortograafiseen projektioon, mikä tarkoittaa sitä, että pelin sisältö näytetään kaksiulotteisena. Tätä asetusta voi kuitenkin muuttaa, ja kameraa voi vaihdella vapaasti kaksi- ja kolmiulotteisen välillä. (Godbold & Jackson 2016, 11.)

Unity luokittelee myös 2D-peliympäristöön tuodut kuvat automaattisesti sprite-grafiikoiksi, eli 2D-kuviksi, kun taas 3D-ympäristössä nämä luokiteltaisiin 3D-kappaleisiin lisättäviksi tekstuureiksi. 2D-pelinkehitystä varten Unityn versiossa 4.3 on lisätty oma fysiikkajärjestelmä 2D-peleille. Unityn lisäämä tuki 2D-pelinkehitykselle tekee kyseisestä pelimoottorista hyvin tehokkaan työvälineen 2D-pelien kehitykseen. (Godbold & Jackson 2016, 13, 18.)



Kuvio 1: Unityn tarjoamia mallipohjia uusille projekteille

2.2 Käsitteistö

Collider	Collider määrittää peliobjektin alueen, jolla se voi törmätä toisten peliobjektien Collidereihin (Unity Documentation 2022).
Idle-animaatio	Animaatio, joka toistuu, kun pelihahmo ei liiku mihinkään suuntaan, eli kun se ei saa syötteitä pelaajalta.
Kohtaus	Kohtaukset ovat ympäristöjä, jotka sisältävät pelin sisältöjä. Yksi kohtaus voi sisältää pelin kokonaisuudessaan tai peli voi olla jaettu useampaan kohtaukseen, riippuen pelin laajuudesta. (Unity Documentation 2022.)
Komponentti	Komponentit ovat peliobjektiin lisättyjä toiminnallisia ominaisuuksia, jotka auttavat määrittämään peliobjektien toimintaa (Unity Documentation 2022).
MDA-viitekehys	Pelisuunnittelun viitekehys, joka määrittää pelin mekaniikan, dynamiikan ja estetiikan. MDA:n mukaan pelisuunnittelijan tulisi keskittyä ensin pelin estetiikkaan. (Bond 2018, 20-21.)

Parallax-tausta	Tausta, joka on jaettu tasoihin (layer), jotka ovat asetettu eri etäisyyksille ja liikkuvat eri nopeuksilla suhteessa kameraan.
Pelimoottori	Pelimoottori on ohjelma, jonka avulla voidaan kehittää pelejä. Pelimoottori tarjoaa toiminnallisuuksia muun muassa grafiikkarenderöintiä, fysiikkamallinnusta ja tekoälyä varten. (Full Scale 2021.)
Peliobjekti	Peliobjektiksi lasketaan kaikki peliin lisätyt kappaleet, kuten pelihahmot tai tausta (Unity Documentation 2022).
Skripti	Kooditiedosto, joka sisältää toimintaohjeita pelille. Skriptien avulla määritetään muun muassa yhteys pelaajan syötteiden ja pelin välille. (Unity Documentation 2022.)
Sprite-grafiikka	Sprite-grafiikat ovat kaksiulotteisia kuvia (Unity Documentation 2022).

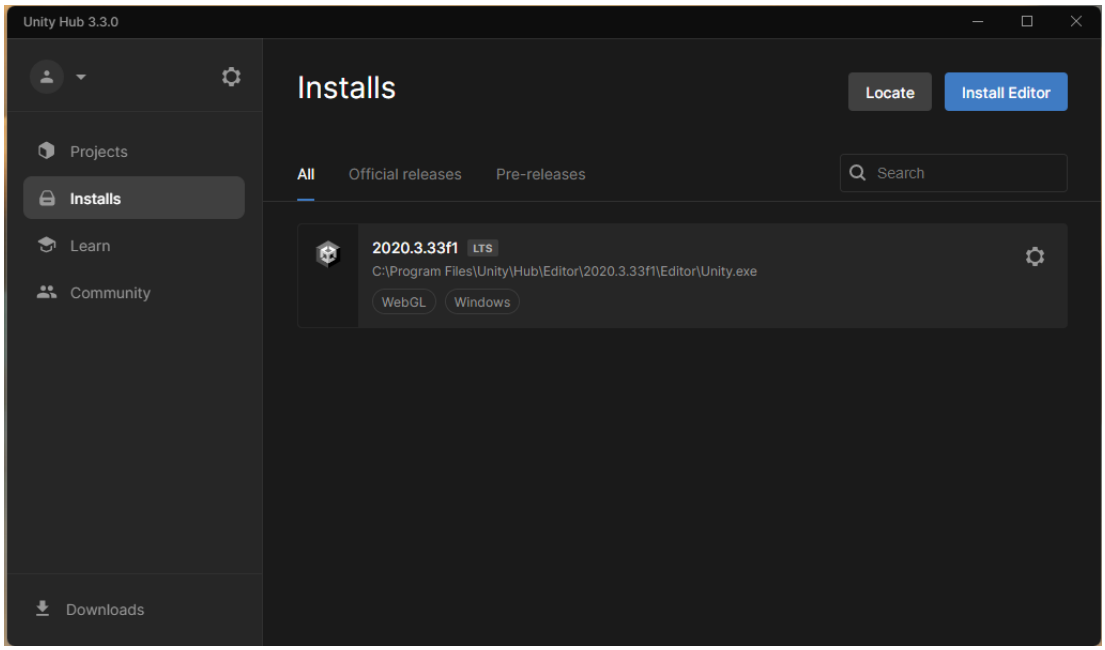
3 Unity

Unity on 2000-luvun alussa kehitetty pelimoottori. Unity tarjoaa kehitysympäristön sekä kehitystyökaluja pelinkehityksen tueksi. Unityn kehittivät tanskalaiset Nicholas Francis ja David Helgason sekä saksalainen Joachim Ante. Unityn tarkoitus oli aluksi olla yksinkertainen työkalu 3D-pelinkehitykseen, ja se oli alunperin kehitetty Mac-alustoille. Unity julkaistiin kuitenkin myöhemmin myös Windows-alustoille, ja sen tarjonta alustojen osalta on laajentunut jälkeenpäin mobiililaitteisiin ja erinäisiin pelikonsoleihin. (Unity Developers 2020.)

Unityn asema pelimoottoreiden keskuudessa on vahva. Vuonna 2019 yli puolet suosituimmista mobiili-, PC- sekä konsolipeleistä olivat kehitetty Unity-pelimoottorilla. Unity on myös ilmoittanut, että Unity-pelimoottoria käyttää kuukausittain noin 1,5 miljoonaa aktiivista kehittäjää. (Hollister 2020.)

3.1 Asennus

Unityn asennus tapahtuu Unityn verkkosivujen kautta. Ennen Unity-editorin lataamista, Unity suosittelee Unity Hub -nimisen ohjelman lataamista. Unity Hub mahdollistaa peliprojektien organisoinnin ja luonnin sen kautta. Lisäksi Unity Hub sisältää erinäisiä pelinkehityksen tutoriaaleja, ja sen avulla voi nähdä kaikki jo asennetut Unity-editorit sekä asentaa uusia versioita. Unity Hubin asennuksen jälkeen voi ladata haluamansa Unity-editorin, joko Unityn verkkosivujen kautta tai Unity Hubin kautta. Oheisessa kuviossa 1 esitetään näyttökaappaus Unity-editorin asennuksista Unity Hubissa. (Unity; Unity3D.)

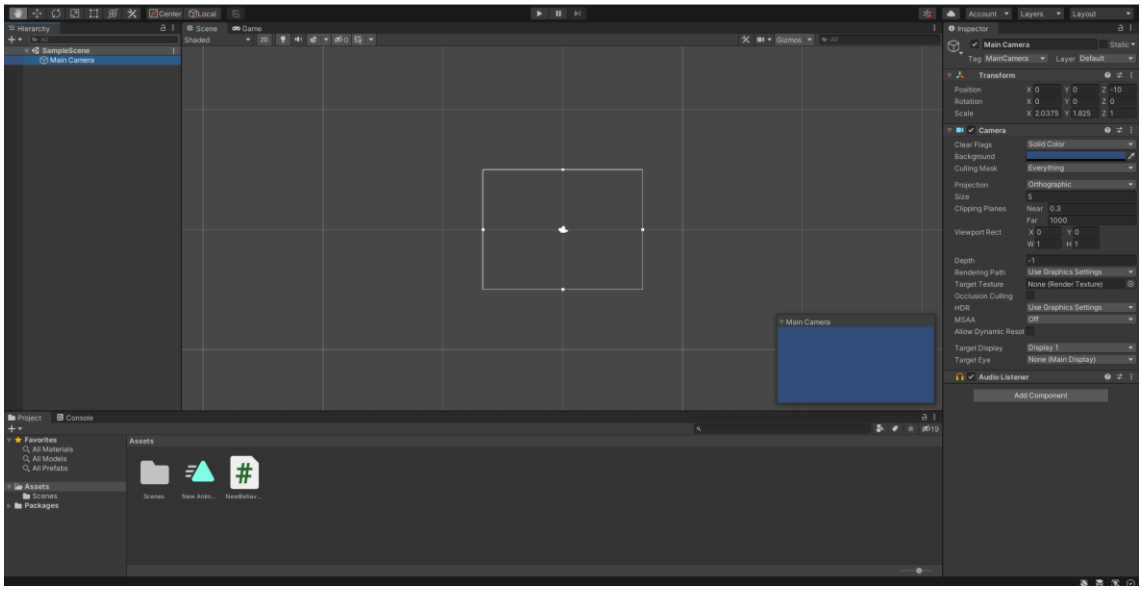


Kuvio 2: Asennukset Unity Hubissa

Unitylla on olemassa erilaisia lisenssejä eri tarpeisiin. Unity Personal on harrastelijoille tai pienille organisaatioille tarkoitettu ilmainen lisenssi, joka sisältää perus Unity-alustan. Unity Pro ja Plus ovat ammattilaisille ja yrityksille tarkoitettuja maksullisia lisenssejä, jotka sisältävät enemmän ominaisuuksia, kuten työkaluja pelien analytiikan seuraamiseen. Unity Enterprise on myös maksullinen lisenssi, joka sisältää Unity Pro:n sekä lisäksi työkaluja laajempiin käyttötarpeisiin. (Unity Store.)

3.2 Käyttöliittymä

Seuraavaksi käyn läpi Unityn käyttöliittymää ja sen ominaisuuksia oman kokemukseni pohjalta. Alla sijaitsevassa kuviossa 3 esitetään Unityn kehitysympäristö siten, miten se näyttäytyy juuri luodussa peliprojektissa. Peliympäristöön on luotu vain muutama mallitiedosto, mutta muuten peliympäristö on koskematon.



Kuvio 3: Unityn käyttöliittymä

Kun Unity Hubissa luodaan uusi projekti ja se klikataan auki, avautuu yllä oleva näkymä. Tässä näkymässä tapahtuu kaikki pelin varsinaiseen rakennukseen liittyvä työ. Unityn oletusnäkymässä pelin tiedostot löytyvät alareunasta Assets-valikosta, peliobjektit löytyvät vasemmasta reunasta Hierarchy-valikosta ja peliobjektien konfigurointi ja komponentit löytyvät oikeasta reunasta Inspector-valikosta. Itse kehitettävän pelin näkee kohtausnäkymästä, joka sijaitsee ylläolevan kuvan keskellä. Unityn kehitysympäristö on kuitenkin hyvin joustava, ja työkaluja ja ikkunoita voi järjestää haluamallaan tavalla.

Käyttöliittymän vasemmasta reunasta löytyvä, Hierarchy-niminen valikko sisältää projektin peliobjektit. Automaattisesti luotuna valikosta löytyy Main Camera -niminen peliobjekti, joka sisältää pelin kameran. Kamera määrittää sen, mitä pelaajalle näkyy peliä pelattaessa. Kaikki peliin lisättävät elementit, kuten pelihahmo ja viholliset, tulevat näkymään tässä valikossa. Nimensä mukaisesti, peliobjekteille voi määrittää hierarkian asettamalla niitä esimerkiksi toisen peliobjektin lapseksi. Konkreettisesti tämä tapahtuu siirtämällä peliobjekti toisen peliobjektin sisälle.

Projektiin tuodut sekä luodut sisällöt, kuten sprite-grafiikat, skriptit ja kohtaukset, löytyvät Assets-valikon alta. Tiedosto-näkymään voi viedä omia tiedostoja, kuten kuvia. Tästä näkymästä voi lisätä esimerkiksi sprite-grafiikoita suoraan peliin tai skriptejä peliobjekteihin vetämällä. Assets-valikossa on mahdollista organisoida peliprojektin tiedostorakennetta, luomalla kansioita ja järjestelemällä tiedostoja niille sopiviin kansioihin.

Peliobjektien komponentteja voi tarkastella Inspector-valikosta. Inspector-valikko sisältää myös muun muassa työkalut peliobjektien sijainnin ja koon tarkempaan konfiguroimiseen. Kokoa ja sijaintia voi muokata myös kohtausnäkymässä, mutta Inspector-välilehdessä nämä

voi määritellä desimaalin tarkkuudella halutessa. Kaikki informaatio yksittäisistä peliobjekteista löytyy jokaisen peliobjektin yksilöllisestä Inspector-näkymästä.

Varsinaisen peliympäristön rakentaminen tapahtuu kohtausnäkymässä. Lisätyt peliobjektit ilmestyvät kohtausnäkymään, jossa niistä voi ryhtyä rakentamaan peliympäristöä. Painamalla kohtausnäkymän yllä olevaa Play-nappia voi testata, miten peli toimii. Tämä tapahtuu Game-välilehdellä, joka löytyy kohtausnäkymän kanssa samasta ikkunasta. Pelinäkymässä peliä voi tarkastella ja pelata kuten valmista tuotosta. Tässä näkymässä sitä ei kuitenkaan voi muokata pysyvästi, vaan pysyvien muokkausten tekemiseksi täytyy palata kohtausnäkymään.

4 Pelin toteutus

Peli toteutettiin täysin Unity-pelimoottorilla ja hyödyntäen Unityn kehitystyökaluja. Pelin ohjelmointikielenä toimi C#. Koodieditorina käytettiin Visual Studio Code -ohjelmaa.

Pelin kehitys ajoittui noin kolmen kuukauden ajalle. Kehitys tapahtui kolmessa vaiheessa, Iteratiivisen suunnittelun mukaisesti. Nämä kolme vaihetta olivat suunnitteluvaihe, toteutusvaihe sekä testausvaihe. Analyysivaihetta ei sisällytetty tähän työhön samalla tavalla, sillä se on tapahtunut jo opinnäytetyön aihetta määritellessä. Ensimmäisenä oli suunnitteluvaihe, jossa oli tarkoitus rakentaa pelin ideaa. Seuraava vaihe oli toteutusvaihe, jossa suunnitelma toteutettiin Unity-ympäristössä. Viimeisenä vaiheena oli testausvaihe, jossa testattiin ensimmäistä versiota valmiista pelistä. Testauksen perusteella kehitettiin jatkokehitysideoita. Tässä opinnäytetyössä ei käsitelty iteratiivisen prosessin seuraavaa suunnittelu- tai toteutusvaihetta, vaan opinnäytetyön lopputuloksena oli ensimmäinen toiminnallinen versio pelistä, sekä jatkokehitysideat seuraavalle Iteratiivisen suunnittelun kierrokselle.

4.1 Suunnitteluvaihe

Suunnitteluvaihe alkaa ideoinnista, jonka pohjalta on tarkoitus rakentaa suunnitelma toteutusvaiheelle. Pelisuunnittelun tarkoituksena on määrittää muun muassa pelin päämäärä, säännöt sekä tilat, eli pelialueet, jossa pelaaja voi liikkua. Suunnittelussa tulee myös huomioida pelin juoni, pelihahmo ja -maailma sekä esteettiset puolet, kuten pelin tunnelma. Suunnitteluvaiheessa luodaan pohjaa tulevalle toteutusvaiheelle. (Bond 2018, 40, 49-50, 92.)

Bond (2018, 20) kuvaa yhtä pelisuunnittelun viitekehyksistä nimeltä MDA. MDA antaa omat määrityksensä pelin mekaniikalle (Mechanics), dynamiikalle (Dynamics) sekä estetiikalle (Aesthetics). Estetiikalla tarkoitetaan tässä yhteydessä pelin aiheuttamia tunnereaktioita, dynamiikalla yhteyttä pelaajan syötteiden ja pelin välillä ja mekaniikalla pelin varsinaisia rakennuspalikoita, kuten pelihahmoa ja pelimaailman osia. MDA myös käsittelee näkökulmia,

joista pelin suunnittelija ja pelaaja tarkastelevat peliä. MDA-viitekehys nimittäin määrittelee, että pelin suunnittelijan tulisi ensin keskittyä pelin estetiikkaan, ja vasta sitten pelin dynamiikkaan ja mekaniikkaan. Pelaajan näkökulmasta tämä järjestys on usein päinvastainen, sillä pelikokemus alkaa useimmiten mekaniikasta ja päättyy estetiikkaan. Tämän opinnäytetyön peliprojektissa suunnittelu suoritettiin MDA-viitekehyksen mukaisesti. (Bond 2018, 20-21.)

4.1.1 Idea

Pelin perusajatuksena oli luoda yksinkertainen 2D-peli. Pelin teossa kiinnosti erityisesti pelimaailman luominen, joten halusin luoda sellaisen pelin, joka sallisi tämän. Pelimaailman ideoinnissa toimi pohjana ajatus pelin estetiikan suunnittelusta, eli siitä, miten pelimaailma voisi aiheuttaa tunnereaktioita pelaajassa. Päädyin sivustavieritettävään peliin (side-scroller), sillä se tuki luovuutta ollen samaanaikaisesti suhteellisen yksinkertainen formaatti ensimmäiselle peliprojektille.

Pelin tehtävänä on saattaa pelihahmo kentän läpi ilman, että pelihahmo osuu matkan varrelta löytyviin vihollisiin. Pelin idea lähti pelihahmosta, ja muut elementit rakentuivat sen ympärille yksi kerrallaan. Pelaajalla täytyi olla jokin päämäärä, joten päätin, että pelihahmon täytyy kulkea A:sta B:hen. Öinen metsämaisema selittäisi pelihahmon tarvetta päästä kotiin. Pelihahmolle täytyi asettaa esteitä, joten matkalle ilmestyivät vihollishahmot, joiden yli pelihahmon on hypättävä. Lopulta pelin lopusta löytyy määränpää; otuksen koti. Peli päättyisi pelihahmon päästessä kotiinsa.

4.1.2 Peliobjektit

Ideassa esiintulleet elementit täytyi vielä muuttaa peliobjekteiksi. MDA-viitekehyksen periaatteiden mukaisesti, peliobjektien kehityksessä on tarkoitus suunnitella pelin mekaniikka ja dynamiikka, eli pelin konkreettiset rakennuspalikat sekä yhteydet pelaajan syötteiden ja pelin välille. Idean toteuttamiseksi tarvittiin peliobjektit muun muassa pelihahmolle, vihollisille, taustalle sekä pelihahmon kodille. Kaikkien peliobjektien osalle täytyi myös suunnitella, mitä toiminnallisuuksia ne tarvitsivat ja miten ne toimivat yhteydessä toisiinsa.

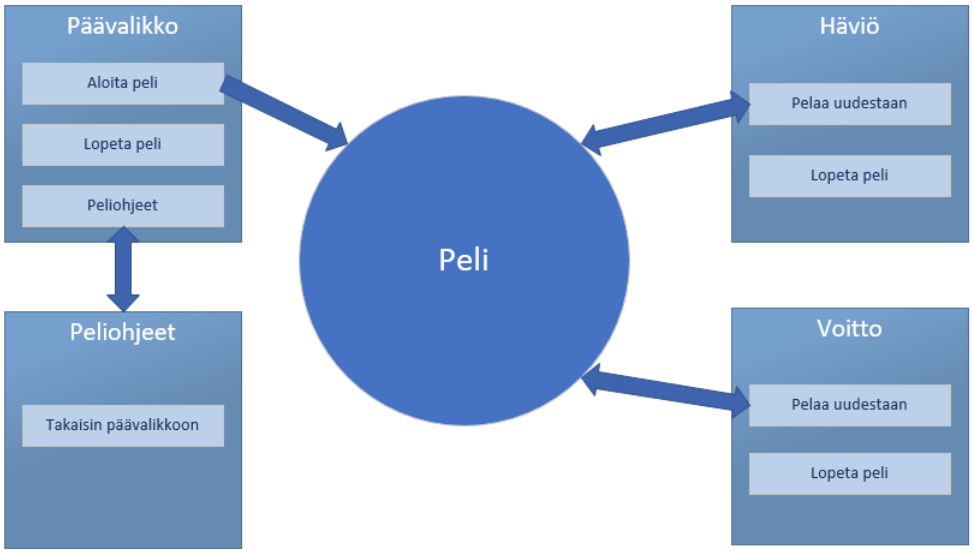
Pelihahmo tarvitsi skriptin, jotta se voi liikkua vasemmalle ja oikealle. Lisäksi pelihahmon täytyi pystyä hypätä. Pelihahmolle oli myös tarkoitus luoda eloa animaatioiden avulla. Esimerkiksi pelihahmolle tuli tehdä kävely-animaatio, joka käynnistyy aina, kun pelaaja liikuttaa hahmoa oikealle tai vasemmalle.

Vihollishahmoille täytyi tehdä toiminto, joka päättää pelin, jos pelihahmo osuu viholliseen. Tällöin pelaajalle näkyisi ruutu, jossa ilmoitetaan pelin päätöksestä ja annetaan mahdollisuus yrittää uudestaan. Myös niille oli tarkoitus saada liikettä animaatioilla. Mökille, eli

pelihahmon kodille, tuli myös toiminto, joka päättää pelin, kun pelihahmo tulee tarpeeksi lähelle. Tämä on pelin tarkoitettu loppu, joten silloin viesti pelaajalle on onnitteleva.

4.1.3 Valikot

Pelin rakenteelle on tärkeää suunnitella pelin valikot. Peli tarvitsi päävalikon, josta pelin voi käynnistää tai lopettaa. Päävalikosta olisi myös hyvä päästä peliohjeisiin. Peliohjeille tuli olla oma valikkonäkymänsä, josta pääsee takaisin päävalikkoon. Lisäksi pelin päätyttyä tarvittaisiin valikko, josta on mahdollista aloittaa peli uudestaan tai lopettaa peli. Päätösvalikoita tarvittaisiin kaksi. Toinen valikkoruutu olisi tilanteelle, jossa peli on hävitty, ja toinen tilanteelle, jossa pelaaja on päässyt pelistä läpi onnistuneesti. Oheisessa kuviossa 4 selvennetään valikkorakennetta.



Kuvio 4: Pelin valikkorakenteen suunnitelma

4.2 Toteutusvaihe

Suunnitteluvaiheen jälkeen alkoi toteutusvaihe, jossa suunnitelmavaiheen tuotoksista toteutettiin toimiva peli. Toteutusvaiheessa siirryttiin Unityn kehitysympäristöön, ja luotiin peliprojekti, johon peli kehitettiin. Toteutusvaiheessa ryhdyttiin luomaan peliobjekteja ja sprite-grafiikoita ja tuomaan niitä Unityn projektiympäristöön. Tässä vaiheessa rakennettiin myös toiminnallisuudet ja pelin logiikka, kunnes peli oli valmis pelattavaksi eikä sisältänyt sen toimintoja estäviä ongelmia. Lopputuloksena oli Unitysta exportoitu, käynnistettävä pelitiedosto, jonka avulla voitiin suorittaa testaus ulkoisten testaajien avulla.

4.2.1 Prosessin kuvaus

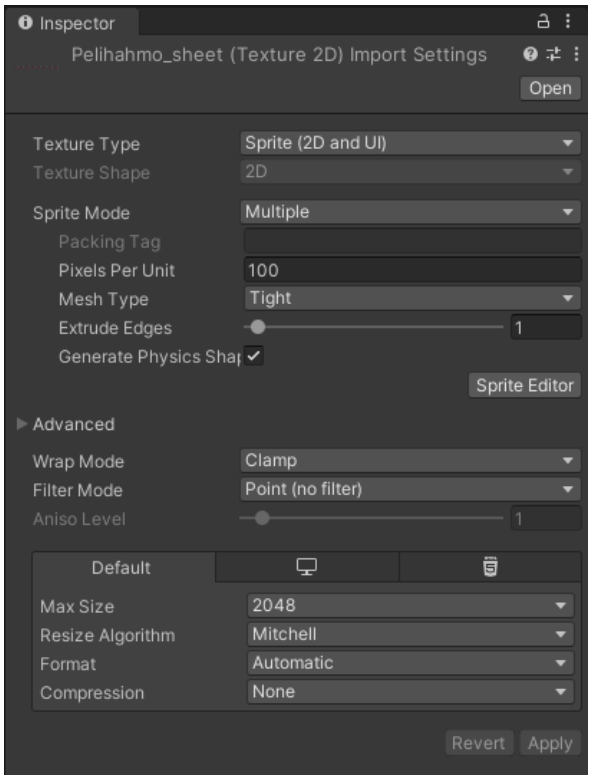
Pelin toteutus tapahtui Scrum-metodologian mukaisissa, sprinttimäisissä jaksoissa kahden kuukauden ajan. Suunnitelman pohjalta voitiin määrittää peliprojektin backlog, eli lista kaikista peliin sisällytettävistä ominaisuuksista. Pelin rakennus tapahtui idean pohjalta yksi backlog-listan jäsen kerrallaan, alkaen pelihahmosta. Prosessin alkaessa pelihahmo oli kaikista selvimmin ideoitu, joten se oli ensimmäinen asia, joka ilmestyi pelin Unity-ympäristöön. Pelihahmolle rakennettiin sen toiminnallisuudet ja animaatiot, jonka jälkeen sen ympärille oli tarkoitus ryhtyä rakentamaan pelimaailmaa. Pelimaailmaan tulivat mukaan metsäinen tausta, viholliset sekä lopuksi mökki. (Bond 2018, 225-226.)

4.2.2 Pelin ulkoasu

Pelin grafiikat ovat itseluotuja pikselipiirroksia. Grafiikoiden luomiseen hyödynnettiin Internetistä löytyvää Pixilart-ilmaistyökalua. Pixilart tarjosi opinnäytetyön grafiikoiden luomiseen tarvittavat työvälineet. Se salli esimerkiksi yksinkertaisten animaatioiden luomisen sekä mahdollisuuden ladata animaatioiden kaikki yksittäiset kuvat (frames), mikä helpotti niiden hyödyntämistä animaatioiden luonnissa Unityssa huomattavasti. Pelihahmo ja vihollinen ovat 64 x 64 mittaisia pikselipiirroksia. Pelin tausta on kooltaan 320 x 180 ja mökki 180 x 180. Pelin sprite-grafiikat löytyvät esiteltynä oheisesta kuviosta 5.

Kuvio 5: Peliin luotuja sprite-grafiikoita

Sprite-grafiikat vietiin Unityyn peliprojektin tiedostoihin ja sieltä varsinaiseen peliin. Ilman minkäänlaista konfigurointia, sprite-grafiikat olivat hieman huonolaatuisia, erityisesti jos niitä suurensi. Tämä korjaantui asettamalla grafiikoiden Filter Mode -asetus Point (no filter) -muotoon. Tämän asetuksen myötä sprite-grafiikoita pystyi suurentamaan ilman laadun huononemista. Myös animaatiotiedostoille täytyi määrittää kyseinen asetus. Animaatiotiedostoille asetettiin lisäksi Sprite Mode -asetus Multiple-muotoon, jotta Unity tietäisi, että kyseinen tiedosto sisältää useamman kuvan. Oheisessa kuviossa 6 esitetään pelihahmon yhden animaatiotiedoston asetukset, jossa Sprite Mode ja Filter Mode ovat asetettuna.

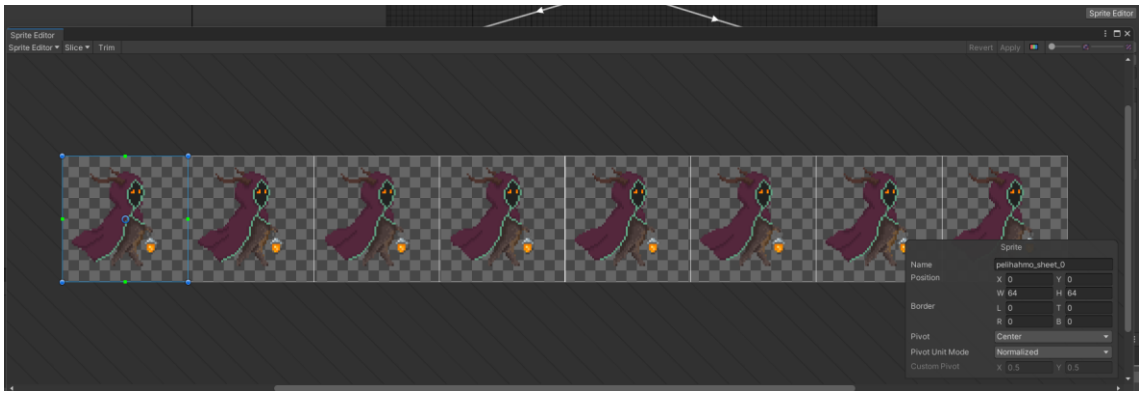


Kuvio 6: Pelihahmon animaation asetukset, jossa Sprite ja Filter Mode asetettu

Pelissä esiintyy useampia animaatioita. Esimerkiksi pelihahmolla on kolme animaatiota; niin sanottu idle-animaatio, joka on käynnissä aina silloin, kun pelihahmo seisoo paikallaan, sekä kävely-animaatiot (vasemmalle ja oikealle), jotka käynnistyvät pelihahmon liikkuessa. Lisäksi vihollisilla on idle-animaatio, jossa ne lipuvat ylös ja alas paikallansa. Metsästä löytyy myös silmäparit, jotka aukeutuvat ja sulkeutuvat.

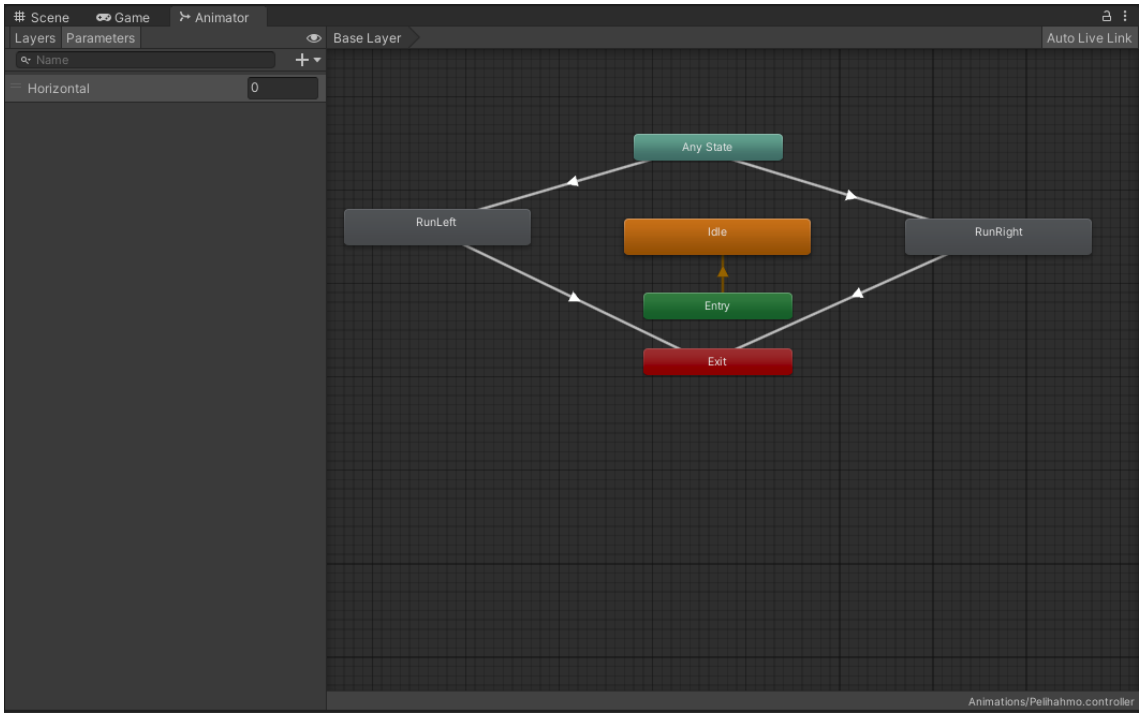
Animaatioiden tuominen peliin onnistui Unityn animaatiotyökalujen avulla. Kun Pixilartista ladattu animaatiotiedosto tuodaan Unityyn, se täytyy pilkkoa palasiksi Sprite Editor -työkalulla. Näin tuodaan Unitylle selväksi, mitkä animaation yksittäiset kuvat (frames) ovat. Pilkottuna nämä kuvat voidaan siirtää Unityssa luotuun animaatioon, jossa voidaan määrittää,

kuinka paljon aikaa yksittäisten kuvien välillä on. Tämä vaikuttaa siihen, miten nopea tai hidas animaatio on. Alla olevassa kuviossa 6 esitetään animaatiotiedoston pilkkomista Sprite Editor -työkalulla.



Kuvio 7: Animaation pilkkominen Unity-ympäristössä

Animaatioita luodessa Unity luo myös Animator Controller -nimisen ominaisuuden animoitavalle peliobjektille. Esimerkiksi pelihahmolla on oma Animator Controller, jonka avulla pystytään konfiguroimaan sen animaatioita. Pelihahmolla on kolme animaatiota, joiden sisälle on viety pilkotut animaatiotiedostot. Animoimisen voi päättää tähän niiden peliobjektien osalta, joilla on vain yksi toistuva animaatio. Pelihahmolla on kuitenkin kolme erillistä animaatiota. Tästä syystä, pelihahmolle piti määritellä, milloin se käynnistää minkäkin animaation. Tämä tapahtui sen Animator Controllerin avulla. Oheisessa kuviossa 7 esitellään pelihahmon Animator Controllerin sisältöä, jossa on määritetty pelihahmon tilat ja niihin kuuluvat animaatiot.



Kuvio 8: Pelihahmon Animator Controller -kartta

Pelihahmon Animator Controllerissa sijaitsee kartta, jonka avulla se päättää sen hetkisen animaationsa. Animator Controller tarkkailee pelihahmon horisontaalista liikettä, jota kautta se määrittelee, liikkuuko pelaaja oikealle vai vasemmalle, ja käynnistää tämän mukaan tietyn animaation. Tätä varten pelihahmon skriptiin on lisätty koodipätkä, jota kautta tämä tarkkailu tapahtuu. Alla sijaitsevassa kuviossa 8 esitetään pelihahmon skripti ja lisätty koodi. Karttaan on myös määritetty idle-tila, johon pelaaja asettuu aina, kun se ei liiku mihinkään suuntaan.

```
6   {
7
8       public Animator animator;
9
10      void Start()
11      {
12
13      }
14
15      void Update()
16      {
17          animator.SetFloat("Horizontal", Input.GetAxis("Horizontal"));
18
19          Vector3 horizontal = new Vector3(Input.GetAxis("Horizontal"), 0.0f, 0.0f);
20          transform.position = transform.position + horizontal * Time.deltaTime;
21      }
22  }
23
```

Kuvio 9: Pelihahmon animaatio-koodi skriptissä

4.2.3 Pelin toiminnallisuus

Pelin toiminnallisuuksia rakennetaan Unityssa skriptien avulla, kuten aiemmin pelihahmon Animator Controllerin kanssa. Skriptit sisältävät koodia, jotka luovat peliobjekteille toiminnallisuuksia. Skripti voidaan liittää peliobjektiin viemällä se peliobjektin komponentiksi, jolloin sen sisältö vaikuttaa kyseiseen peliobjektiin.

Pelihahmolle kuuluvia toiminnallisuuksia ovat kävely vasemmalle ja oikealle sekä hyppy. Nämä toteutettiin lisäämällä koodia pelihahmon skriptiin. Tämä koodi tarkkailee pelaajan painamia nappeja ja siirtää pelihahmoa sen mukaan. Pelihahmon hypyn kanssa esiintyi ongelmia, jotka täytyi ratkaista. Hyppyyn täytyi määrittää, että jos pelihahmo on vielä ilmassa edellisestä hypystä, pelaaja ei voi hypätä uudestaan. Ongelma hypyssä oli nimittäin se, että painamalla hyppy-nappia useamman kerran peräkkäin pelaaja pystyi lentämään, sillä pelihahmo ei ehtinyt palata maahan hyppyjen välissä. Korjaus onnistui lisäämällä koodiin funktion, joka tarkkaili sitä, oliko pelihahmo takaisin maassa vai ei. Jos pelihahmo ei ollut takaisin maassa, hyppyä ei voinut suorittaa.

Pelihahmolle annettiin myös oma Box Collider 2D -ominaisuus, joka määrittää sen ympärille alueen, jolla se voi olla kosketuksissa toisen peliobjektin Colliderin kanssa. Esimerkiksi pelimaailmaan lisättiin maa-peliobjekti, jolla on myös oma Colliderinsa. Pelihahmon Collider törmää maan Colliderin kanssa, mikä tarkoittaa sitä, että pelihahmo ei voi kulkea maan läpi. Ilman maata, tai jonkinlaista alustaa, pelihahmo tippuisi näkymättömiin siihen vaikuttavan gravitaatiovoiman takia. Pelihahmolle annettu Rigidbody 2D -ominaisuus tuo pelihahmon fysiikan lakien alaisuuteen, minkä takia se käyttäytyy painovoiman mukaisesti. Rigidbody-ominaisuus mahdollistaa hyppyvoiman kohdentamisen pelihahmoon, jonka ansiosta hyppy onnistuu. Oheisessa kuviossa 9 näkyy pelihahmon skripti, josta voidaan huomata muun muassa pelihahmon hyppy-toiminnon koodi.

```
    public Animator animator;
    public Rigidbody2D rb;
    public float jumpAmount;
    public bool isJump;

    void Start()
    {

    }

    void Update()
    {
        animator.SetFloat("Horizontal", Input.GetAxis("Horizontal"));

        Vector3 horizontal = new Vector3(Input.GetAxis("Horizontal"), 0.0f, 0.0f);
        transform.position = transform.position + horizontal * Time.deltaTime;

        if (this.transform.position.y > 0) {
            isJump = true;
        } else {
            isJump = false;
        }

        if (Input.GetKeyDown(KeyCode.Space) && isJump==false) {
            rb.AddForce(Vector2.up * jumpAmount, ForceMode2D.Impulse);
        }

    }
}
```

Kuvio 10: Pelihahmon hyppy-koodi skriptissä

Pelaajan täytyy myös nähdä pelihahmo tämän matkatessa pelikenttää pitkin. Pelin kamera tuli valmiiksi luotuna peliprojektin mukana. Oletuksena kamera on liikkumaton. Tämä aiheuttaa sen, että kun pelihahmo liikkuu kameran alueen ulkopuolelle, pelaaja ei enää näe pelihahmoa. Kameran täytyy siis liikkua pelihahmon mukana, jotta pelihahmo ei häviäisi pelaajan näkökentästä. Kaikista helpoiten tämän saisi toteutettua asettamalla kameran pelihahmon lapseksi peliobjektien hierarkia-listassa. Tässä projektissa tämä kuitenkin toteutettiin luomalla kameralle skriptin, joka asettaa sen seuraamaan pelihahmoa. Skriptissä määritetään kameran sijainniksi pelihahmon sijainti horisontaalisesti. Vertikaalisesti kameran sijainti ei muutu, vaan pelihahmon hypätessä kamera pysyy samalla korkeudella. Kameran vertikaalinen liikkumattomuus parantaa pelikokemusta, sillä pelaaja näkee pelikentän paremmin ja osaa väistää vihollisia paremmin, kun hän pystyy näkemään ne pelihahmon hypätessäkin.

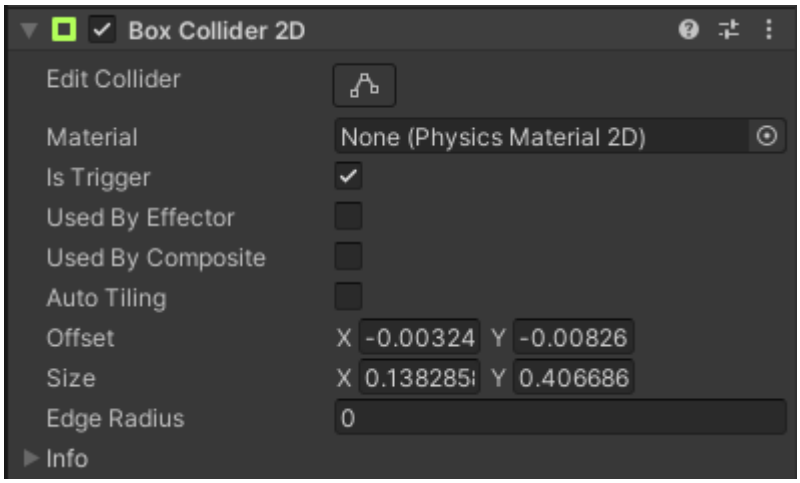
Kameran asettamisessa esiintyviä vaikeuksia olivat muun muassa se, että pelihahmon liikkuessa kameran liike oli nykivää. Se ei siis liikkunut sulavasti, niin kuin pitäisi. Tämä korjautui vaihtamalla skriptissä sijaitsevan funktion Update-funktiosta LateUpdate-funktioksi. LateUpdate-funktio suoritetaan aina Update-funktion sisältöjen jälkeen, mikä auttaa kameran liikkeen sulavuutta, sillä kamera seuraa kappaleita, jotka ovat saattaneet esimerkiksi liikkua

Update-funktion suorituksen aikana. Alla olevassa kuviossa 10 esitetään kameran skripti, jossa sen toiminta määritetään.

```
public class PlayerCamera : MonoBehaviour
{
    public Transform player;
    public float xOffset;
    public float yOffset;

    void Start()
    {

    }

    void LateUpdate()
    {
        transform.position = new Vector3(player.position.x + xOffset, player.position.y + yOffset, transform.position.z);
    }
}
```

Kuvio 11: Kameran toiminta skriptissä

Muita pelin toiminnallisuuksia ovat vihollisten ja pelin päämääränä toimivan mökin toiminnallisuudet. Viholliset ja mökki toimivat jokseenkin samalla tavalla. Vihollisille ja mökille on annettu omat Colliderinsa, mikä tarkoittaa sitä, että ne voivat törmätä pelihahmon Colliderin kanssa. Vihollisten ja mökin Colliderit ovat kuitenkin määritetty niin, että pelihahmo voisi kulkea niiden Collidereiden läpi ongelmitta. Normaalisti Colliderit törmäisivät toisiinsa, eikä pelaaja voisi kävellä niiden läpi. Vihollisten ja mökin Colliderit ovat määritelty Is Trigger -muotoisiksi, mikä tarkoittaa, että ne voivat toimia liipaisina erinäisille toiminnallisuuksille, eivätkä toimi niin ikään näkymättöminä seininä, kuten Colliderit normaalisti. Is Trigger -tyyppinen Collider voi muun muassa tunnistaa, jos pelihahmo liikkuu sen alueelle. Täten sillä saa luotua toiminnallisuuksia vihollisille, joihin pelihahmo ei saa osua, sekä mökille, jonka luokse päästyään peli on voitettu. Oheisessa kuviossa 11 näkyy Is Trigger -asetus vihollisen Colliderin asetuksissa.



Kuvio 12: Vihollisen Colliderin asetukset, joissa Is Trigger määritetty

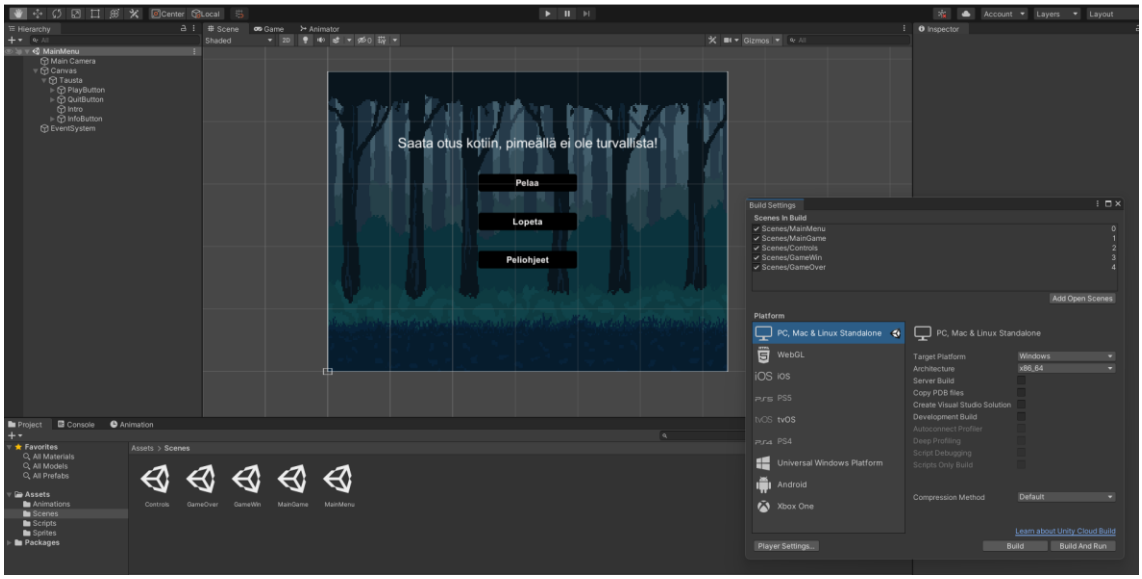
Pelkkä Is Trigger -määrittely ei kuitenkaan riitä toiminnallisuuden luomiseksi. Tästä syystä määritellään vihollisten ja mökin skripteissä, että jos toinen Collider astuu niiden alueiden sisälle, tapahtuu haluamamme toiminto. Vihollisilla tämä toiminto on, että peli siirtyy Game Over -kohtaukseen ja peli on hävitty. Mökillä taas toiminto on, että peli siirtyy kohtaukseen, jossa peli on voitettu. Tämä tapahtuu lataamalla pelin kohtausvalikosta sopivan kohtauksen, kohtausten järjestysnumeroiden avulla. Pelin kohtaukset ovat viety kohtausvalikkoon, jossa niille määräytyy järjestysnumero. Alla sijaitsevassa kuviossa 12 esitetään vihollisen skripti, jossa peli asetetaan vaihtamaan kohtausta pelaajan osuessa siihen.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
6
7   public class EnemyController : MonoBehaviour
8   {
9       void OnTriggerEnter2D(Collider2D other) {
10          SceneManager.LoadScene(4);
11      }
12
13  }
14
```

Kuvio 13: Vihollisen skripti, jossa kohtauksen vaihto määritetään

Pelin sisältöjen välillä liikkuminen tapahtuu kohtauksien avulla. Yllä kuvatusti, vihollisten ja mökin kautta siirrytään häviö- ja voittokohtauksiin, jotka sisältävät valikkoruudut kyseisille tilanteille, mutta pelissä on myös muitakin kohtauksia. Peli alkaa Main Menu -kohtauksesta, joka sisältää pelin päävalikon. Päävalikko-kohtauksessa, kuten kaikissa muissakin kohtauksissa, jotka eivät sisällä varsinaista peliä, on rakennettu eräänlainen käyttöliittymä pelaajalle. Kohtausten peliobjekti-listaan on luotu peliobjekti nimeltä Canvas, jonka alle on rakennettu muun muassa nappeja, joista voi aloittaa tai lopettaa pelin. Canvas ja muut valikon elementit ovat lisätty suoraan Unityn UI (User Interface) -valikoimasta. Tämän ansiosta napeilla on sisäänrakennetut On Click () -toiminnot Komponentti-valikossa. Nappien On Click () -toimintoihin täytyy kuitenkin vielä määrittää, mitä mistäkin napista tapahtuu. Tässä pelissä napit ovat määritetty vaihtamaan kohtausta painaessa. Esimerkiksi Pelaa-nappi vaihtaa peli-kohtaukseen ja Peliohjeet-nappi vaihtaa kohtaukseen, joka sisältää peliohjeet. Lopeta-napille on kuitenkin määritetty toiminto, joka sulkee pelin sitä painaessa. Nämä toiminnot ovat luotu napeille skriptin avulla, jossa määritetään kohtauksen vaihto sekä pelin lopetus. Toimintojen lisäämiseksi nappien On Click () -toimintoihin on lisätty viittaus

kyseiseen skriptiin ja haluttuun toimintoon. Oheisessa kuviossa 13 esitellään pelin kohtauksia eri näkymissä.

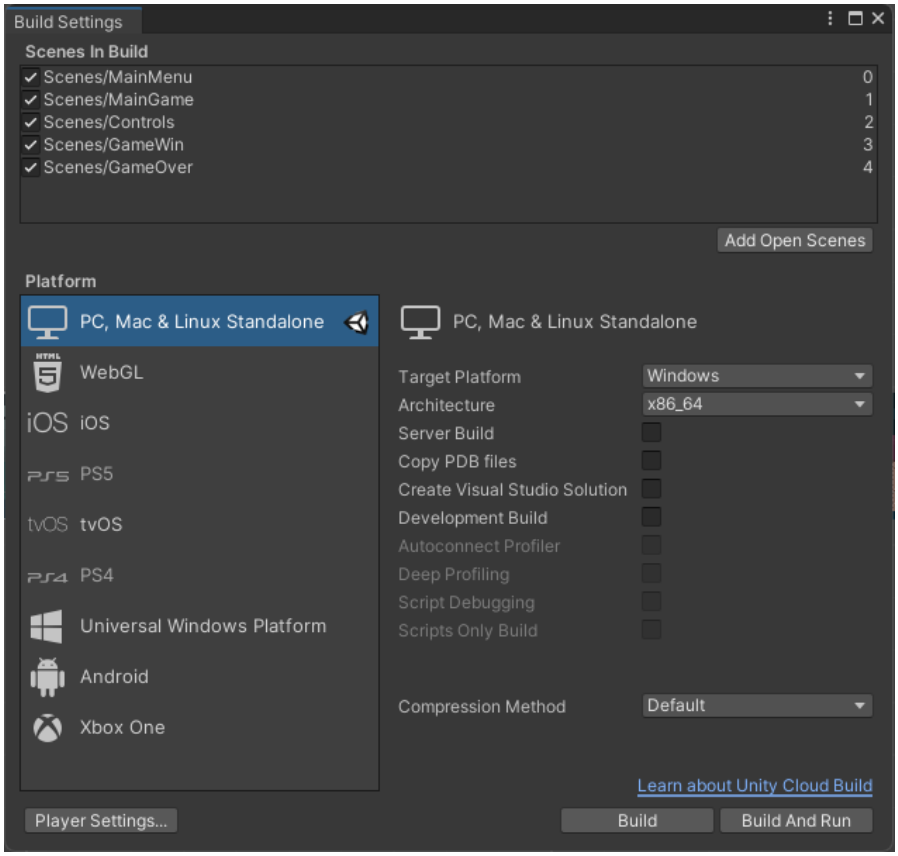


Kuvio 14: Pelin kohtaukset tiedostonäkymässä (alareuna), kohtausvalikossa (oikea) sekä Päävalikko-kohtaus kohtausnäkymässä

Peliä kehittäessä täytyy miettiä peliä pelaajan näkökulmasta. Peliympäristön tulee ohjata pelaajaa oikeaan suuntaan, eikä se saa saattaa pelaajaa tarkoituksettomiin tilanteisiin. Tätä varten peliympäristö täytyi tarkastaa sellaisia ominaisuuksia varten, joiden kautta pelaaja voisi kiertää pelin tarkoitetun toiminnan. Esimerkiksi pelin alussa, tarkoitettu menosuunta on vasemmalta oikealle, mutta mitä tapahtuisi, jos pelaaja lähtisikin kulkemaan oikealta vasemmalle. Estääkseen pelaajan kulun väärään suuntaan, peliin lisättiin näkymätön seinä, joka pysäyttää pelaajan kulun, ennen kun hän putoaisi pelialueelta. Seinä on asetettu pelaajan taakse, pelikentän vasempaan reunaan. Näkymätön seinä on peliin lisätty peliobjekti, jonka näkyvyys on asetettu läpikuultavaksi. Seinällä on oma Collider, joka estää sen läpi kävelemistä. Seinän korkeus estää pelaajan myös hyppäämästä sen yli. Täten pelaajan on pakko lähteä kulkemaan tarkoitettuun suuntaan, vasemmalta oikealle. Alla sijaitsevassa kuviossa 14 esitetään pelin pelikenttä kokonaisuudessaan.

Kuvio 15: Pelikenttä kokonaisuudessaan sekä rajana toimiva seinä

Kun kaikki tarvittu sisältö on luotu peliin, se voidaan viedä (export) suoritettavaan tiedostoon. Tämä tapahtuu samasta valikosta, josta löytyy pelin kohtaukset järjestettynä. Oheisessa kuviossa 15 esitetään tämä valikkonäkymä. Ennen pelin viemistä tulee varmistaa, että kaikki valmiiseen peliin halutut kohtaukset ovat tässä listassa, muuten ne jäisivät puuttumaan. Lueteltujen kohtausten alta on mahdollista määrittää, mille alustalle peli on tarkoitettu. Tässä pelissä tarkoitettu alusta on Windows, joten asetukset säädettiin sen mukaisiksi. Kun kaikki kohtaukset oli luettu mukaan ja pelin alusta asetettu, voitiin painaa Build-nappia ja rakentaa pelistä suoritettava tiedosto. Valmis tuotos, joka koostui suoritettavan pelitiedoston sisältävästä kansiosta, latautui tämän jälkeen koneelle. Pelin ensimmäinen versio oli nyt valmis jaettavaksi muille testaustarkoitukseen.



Kuvio 16: Build Settings -menu, josta peli viedään suoritettavaan tiedostoon

4.3 Testausvaihe

Testausvaiheessa on tarkoitus testata pelin toiminnallisuuksia sekä kokonaisuutta. Bond (2018, 145) kuvaa pelitestaajien neljää piiriä. Ensimmäinen piiri sisältää pelinkehittäjän itse, josta piirit laajenevat ystäviin ja perheenjäseniin, tuttaviin ja lopulta Internetiin. Tässä opinnäytetyössä testaus tulee tapahtumaan ensimmäisen ja toisen piirin toimesta. Ensimmäisen prototyypin valmistumiseen saakka tässä työssä kehitetyn pelin testaajana on toiminut pelinkehittäjä itse. Pelin ensimmäistä versiota testasi ystävät ja perheenjäsenet. Bond (2018, 145) suosittelee tuttavien piiriin siirtymistä silloin, kun pelinkehityksen iteratiivinen prosessi on käyty useamman kerran läpi, ja peli on lähempänä haluttua lopputulosta. (Bond 2018, 145.)

4.3.1 Suunnitelma

Pelin testaajina toimivat pelinkehittäjän lisäksi kolme henkilöä. Testaajien kokemus videopelien parissa oli hajanaista. Yksi testaajista pelaa pelejä säännöllisesti, yksi on pelannut ennen, mutta ei pitkään aikaan ja kolmas ei ole tottunut pelaamaan minkäänlaisia pelejä. Testaajat saivat pelin kokonaisuudessaan testattavaksi, ja saivat itsenäisesti käydä läpi sen toimintoja. Testaajat suorittivat testauksen omilla tietokoneillaan, testatakseen

pelin sopivuutta eri alustoilla. Pelitestaajille ei kerrottu pelin toiminnoista tai päämäärästä etukäteen mitään, vaan testattiin pelin ohjeiden riittävyyttä. Tämänkaltaisessa testausmenetelmässä, black box -testauksessa, testaajat pyrkivät etsimään pelistä ongelmia ilman, että he tietävät mitään pelin sisäisistä toiminnoista tai koodista. Testaajat hyödyntävät black box -testauksessa vain heille tarkoitettuja välineitä, joita ovat tämän pelin osalta näppäimistösyötteet. (Schultz & Bryant 2017, 120.)

Testaus suoritettiin kahden henkilön kanssa paikan päällä, ja yksi testaus suoritettiin etäyhteyksillä. Testauksesta syntyvä palaute kirjattiin muistiinpanoihin testauksen aikana. Testauksella pyrittiin saamaan vastaukset muun muassa seuraaviin kysymyksiin:

- Onko pelin vaikeustaso kohdallaan?
- Ovatko pelin ohjeistukset riittävät?
- Estääkö jokin asia pelin pelaamisen?

4.3.2 Testauksen suoritus

Testauksen keskimääräinen suoritusaika per testaaja oli noin 30 minuuttia. Testaajat saivat pelin eteensä suoritettavana tiedostona, ilman mitään tietoa pelin sisällöstä tai toiminnasta. Testaajat tutustuivat peliohjeisiin ennen pelin aloittamista. Testaajat huomasivat päävalikossa sijaitsevan tekstin, joka selittää pelin päämäärän, ja aloittivat pelin onnistuneesti.

Kaksi testaajista ryhtyi liikkumaan heti oikeaan suuntaan, vasemmalta oikealle. Yksi testaaja, joka oli testaajista kokenein videopelien osalta, liikkui ensin vasempaan suuntaan, jossa kulun estämiseksi asetettu seinä pysäytti hänet. Törmätessä vihollisiin, testaajat joutuivat yrittämään uudestaan muutamaan kertaan. Hyppyjen ajoitus koettiin haastavaksi, mutta kaikki kolme testaajaa pääsivät lopulta onnistuneesti vihollisten ohi. Kaikki kolme testaajaa myös saattoi pelin onnistuneesti loppuun kulkemalla mökin luokse.

Testien aikana ei noussut esiin pelin estäviä ongelmia. Välittömiä korjauksia ei täten tarvinnut suorittaa. Kun testaajat olivat suorittaneet pelin onnistuneesti, kysyttiin heiltä kysymyksiä liittyen pelin positiivisiin ja negatiivisiin puoliin, sekä mahdollisiin parannusehdotuksiin.

4.3.3 Testauksen tulokset

Pelin positiivinen palaute koski muun muassa pelin ilmapiiriä ja pikseligrafiikoita. Palautteesta kävi ilmi, että tarkoitettu tunnelma välittyi testaajiin pelin sprite-grafiikoiden välityksellä. Pelin ohjeistukset koettiin myös riittäviksi. Testaajat ymmärsivät pelin päämäärän sekä peliohjeet, eikä näiden osalta noussut esiin ongelmia. Ohjeistusten sijaan

testauksen aikana ilmentyi kysymyksiä pelin tarinasta ja pelihahmosta, mikä on positiivinen merkki pelimaailman vuorovaikutteisuudesta.

Testauksen palautteesta korostui myös pelin haastavuus. Vihollisten yli hyppäämisessä koettiin haastavaksi hyppyjen ajoitus. Vihollisten ohi pääsy pakotti kaikki testaajat kokeilemaan muutaman kerran uudestaan. Tätä ei kuitenkaan koettu täysin negatiiviseksi, sillä tämän koettiin myös parantavan pelin voiton tyydyttävyyttä. Pelin vaikeustason voisi kuitenkin palautteen perusteella vielä tarkastaa.

Konkreettisia parannusehdotuksia palautteesta löytyi muun muassa pelin lopulle. Yksi testaajista ehdotti pelin loppuun lisäyksiä. Näitä mahdollisia lisäyksiä olivat esimerkiksi se, että pelihahmon päästessä mökin luokse vaihtuisi näkymä mökin sisälle, ja peli päättyisi vasta tähän. Vaihtoehtoisesti ehdotettiin, että ennen pelin päättymistä ruudulle ilmestyisi jonkinlainen osoitus pelin onnistuneesta päättämisestä ennen pelivalikkoon palaamista. Nämä ovat täysin mahdollisia muutoksia, joita on syytä pohtia tarkemmin.

5 Jatkokehitysideat

Testauspalautteen perusteella pelin vaikeustaso saattaa tarvita hiomista. Vaikeustaso ei tarvitse suuria muutoksia, sillä tietty haastavuus koettiin myös positiiviseksi. Kuitenkin esimerkiksi hyppy-mekaniikkaa voidaan hioa, jotta onnistuminen ei olisi niin pienestä kiinni. Haastavuus saattaa myös johtua pelihahmon tai vihollisen Colliderista. Toteutusvaiheessa nämä Colliderit säädettiin siten, että ne kattaisivat pelihahmon ja vihollisen muodot, mutta kattaisi silti mahdollisimman vähän ylimääräistä tilaa välttääkseen epäreiluja häviötilanteita pelaajalle. Vaikeustason säätäminen vaatii siis hyppy-mekaniikan jatkuvaa testausta sekä Colliderien tarkastuksen.

Testauksessa toivotut lisäykset loppuun vaativat suunnittelua. Lisäyksistä ilmeni kaksi vaihtoehtoa, josta suunnittelun voi aloittaa. Joko pelin loppuun täytyy lisätä toiminto, joka onnittelee pelaajaa pelin voitosta ja siirtyy sitten valikkoon, tai peliin täytyy lisätä mahdollisuus siirtyä mökin sisälle. Mökin sisälle siirtyminen vaatisi uuden sprite-grafiikan sekä mahdollisten uusien animaatioiden luomista. Pelin päämäärän mukaisesti pelin päättyminen mökin sisällä voisi olla osuvampi loppu.

Muita jatkokehitysideoita voisi olla muun muassa lisätä peliin niin sanottu parallax-tausta. Parallax-taustassa tausta jaetaan kerroksiin, jotka asetetaan eri etäisyyksille ja liikkumaan eri nopeuksilla kameraan nähden. Testauksessa ilmentyneet mielipiteet pelimaailmaa kohden olivat positiivisia, joten suuret muutokset eivät ole välttämättömiä. Kuitenkin, parallax-tausta voi kehittää nykyistä pelimaailmaa entistä mukaansatempaavammaksi, mikä koettiin myös testauksessa positiiviseksi.

Jos pelistä haluaisi kehittää laajemman, voisi siihen myös kehittää toisen tason. Tämä toinen taso voisi sijoittua pelimaailmassa eri paikkaan, jolloin siihen tulisi piirtää uudet sprite-grafiikat peliympäristöksi. Toinen taso voisi sisältää lisää väistettäviä vihollisia, tai se voisi myös sisältää jonkin täysin uuden tehtävän pelaajalle. Jos peliä ryhtyisi laajentamaan, mahdollisuudet ovat lopulta rajattomia.

6 Oman työn arviointi

Opinnäytetyön tarkoituksena oli tutustua pelinkehitykseen ja Unityn käyttöön ja oppia pelinkehityksen perusteita. Pelin kehittämisen aikana tämä toteutui. Lähtötilanteeseeni verrattuna olen paljon tietoisampi pelinkehityksen käytännöistä sekä Unityn toiminnasta. Erityisesti kehitystä tapahtui Unity-pelimoottorin käytön osalta, ja uskon voivani jatkossa kehittää Unitylla varmemmin.

Pelin kehittäminen sujui pääosin sujuvasti. Sain peliin lisättyä kaikki elementit, joita suunnittelin, eikä pelinkehityksessä tullut vastaan kehitystä estäviä ongelmia. Varsinainen pelin kehittämisen ajoittaminen ja tehtävien jakaminen onnistuivat myös hyvin. Scrum-metodologia sopi peliprojektin kehitysmenetelmäksi oivasti.

Pelin kehittämisessä tuli kuitenkin vastaan monenlaisia haastavia tilanteita. Nämä haastavat tilanteet koskivat muun muassa peliobjektien skriptausta ja animointia. Skriptaukseen liittyvät haasteet vaativat ongelmanratkaisua, sillä skriptatut ominaisuudet eivät aina toimineet halutulla tavalla. Aiempi ohjelmointikokemus tuli kyseisissä tilanteissa hyödyksi. Animaatioissa haaste johtui omasta kokemattomuudesta animaatioiden tekemisessä. Animoinnissa kuitenkin opin käytänteitä, jotka varmasti helpottavat niiden tekemistä jatkossa.

7 Yhteenveto

Iteratiivisen suunnittelun mukaisesti, testausvaiheen päättyessä siirryttäisiin takaisin kehitysprosessin aikaisempiin vaiheisiin. Testausvaiheessa esiintyneen palautteen perusteella pelin olemassaolevassa sisällössä on kehitettävää. Konkreettinen suunnitelma pelin jatkolle olisi siirtyä uudestaan suunnitelmavaiheeseen, ja käydä läpi pelin seuraavia toteutettavia ominaisuuksia. Muun muassa pelin vaikeustason hiominen sekä pelin jatkokehitysideat vaatisivat suunnittelua.

Pelinkehityksen työvälineinä käytetyt menetelmät, kuten Iteratiivisen suunnittelun periaatteet ja MDA-viitekehys tukivat peliprojektin etenemistä. Ne auttoivat selventämään

pelinkehityksen prosessia ja antoivat prosessille konkreettiset raamit, joiden sisällä kehitys ja suunnittelu voi tapahtua. Erittäin hyödylliseksi menetelmäksi korostui Iteratiivinen suunnittelu. Vaikka tässä opinnäytetyössä ei siirrytty seuraaviin Iteratiivisen suunnittelun vaiheisiin, se osoittautui silti luonnolliseksi tavaksi kehittää peliä eteenpäin. Lisäksi Scrum-metodologia auttoi kehittämisen jäsentelyssä ja ajoittamisessa backlog-listan ansiosta.

Opinnäytetyön tavoitteen mukaisesti peliprojektin lopputuloksena oli toiminnallinen 2D-peli. Pelin kehitys tapahtui suunnitelman mukaisessa ajanjaksossa ja siihen sisällytettiin suunnitelman mukaisesti kaikki halutut ominaisuudet. Toiminnallisen pelin onnistunut kehitys osoittaa, että olen tutustunut tavoitteen mukaisesti pelinkehitykseen ja Unity-pelimoottorin käyttöön. Lopputulos on myös osoitus siitä, että Unity-pelimoottori soveltuu hyvin ensimmäisen peliprojektin kehitystyökaluksi.

Lähteet

Painetut

Bond, J. G. 2018. Introduction to Game Design, Prototyping, and Development: from concept to playable game with Unity and C#. 2. Painos. Addison-Wesley.

Chandler, H. & Chandler, R. 2011. Fundamentals of Game Development. Jones & Bartlett Learning.

Godbold, A. & Jackson, S. 2016. Mastering Unity 2D Game Development. Packt Publishing Ltd.

Schultz, C. P. & Bryant, R. D. 2017. Game Testing All In One. 3. Painos. Mercury Learning and Information.

Sähköiset

FreeCodeCamp. 2019. What is Game Development? Viitattu 27.9.2022. https://www.freecodecamp.org/news/what-is-game-development/

Full Scale. 2021. What is a Game Engine? Viitattu 23.9.2022. https://fullscale.io/blog/what-is-game-engine/

Hollister, S. 2020. Unity's IPO filing shows how big a threat it poses to Epic and the Unreal Engine. Viitattu 21.9.2022. https://www.theverge.com/2020/8/24/21399611/unity-ipo-game-engine-unreal-competitor-epic-app-store-revenue-profit

Starloop. 2022. 3D Vs. 2D: The Eternal Battle to Develop Video Games. Viitattu 24.9.2022. https://starloopstudios.com/3d-vs-2d-the-eternal-battle-to-develop-video-games/

Unity Developers. 2020. Evolution of Unity and how it became a great tool also for Artists and Designers. Viitattu 30.9.2022. Viitattu 3.10.2022. https://unitydevelopers.co.uk/evolution-of-unity-and-how-it-became-a-great-tool-also-for-artists-and-designers/

Unity Documentation. Unity Manual. Viitattu 3.10.2022. https://docs.unity3d.com/Manual/Quickstart2D.html

Unity3D. Downloading and installing Editors and modules with the Unity Hub. Viitattu 2.10.2022. https://docs.unity3d.com/hub/manual/InstallEditors.html

Unity. Download Unity. Viitattu 3.10.2022. https://unity.com/download

Unity Store. Plans and pricing. Viitattu 2.10.2022. https://store.unity.com/

Kuviot