Tampereen ammattikorkeakoulu

2D-pelin kehittäminen Unity:lla

Ossi Haavisto

OPINNÄYTETYÖ
Kesäkuu 2022

Tieto- ja viestintätekniikka
Tietoliikennetekniikka ja tietoverkot

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Tietoliikennetekniikka ja tietoverkot

HAAVISTO, OSSI:
2D-pelin kehittäminen Unity:lla

Opinnäytetyö 35 sivua, joista liitteitä 1 sivu
Kesäkuu 2022

Opinnäytetyössä esitellään pelimoottori Unity ja tutustutaan sen monipuolisuuteen. Työssä perehdytään pelin suunnitteluun ja 2D-videopelin kehittämiseen Unity:lla. Suunnitteluvaiheessa käsitellään tekijöitä, jotka syventävät pelaajan pelikokemusta virtuaalitodellisuuden kanssa, sekä tekijöitä, joita pelisuunnittelussa tulee ottaa erityisesti huomioon. Suunnittelussa esitetyt tekijät käsitellään ja näiden merkityksiin perehdytään. Keskeisiä aiheita suunnittelussa tässä työssä ovat pelin pelattavuus, animaatiot, pelimaailman efektit sekä äänimaailma.

Työssä tutustutaan Unity:n toimintoihin sekä komponentteihin, joita hyödyntämällä ohjelmoidaan alusta loppuun toimiva 2D-peli. Pelin ohjelmoinnissa käsitellään ja perehdytään pelin ohjelmointikoodin. Ohjelmoinnin yhteydessä perehdytään animaatioiden ja tietokonegrafiikan eli tekstuurien käyttämiseen Unity:ssa esittelemällä Unity:n erilaisia työkaluja.

Työ toimii myös esimerkkinä siitä, että jokainen voi vähäisellä kokemuksella suunnitella ja kehittää oman toimivan videopelin. Työn tarkoituksena on antaa innokkaille pelisuunnittelijoille sekä peliohjelmoijille vahva perustaito Unity:n käytöstä ja sen mahdollisuuksista sekä kasvattaa tietämystä pelinsuunnittelijana ja pelinkehittäjänä.

Asiasanat: Unity, pelisuunnittelu, pelikehittäjä

ABSTRACT

Tampere University of Applied Sciences
Degree Program in ICT Engineering Telecommunications and Networks

HAAVISTO, OSSI:
Making a 2D Game with Unity

Thesis 35 pages, including 1 page of attachments
June 2022

The thesis explains how to develop a 2D game using Unity. The Unity game engine is introduced first and the programming languages used by Unity are reviewed. The thesis also provides a discussion on game design and the factors that deepen the interaction between the game and the player. Key topics in the design phase are gameplay, animation, game world effects, and sound world. The sections explain in detail how they affect the enjoyment of the game.

The programming phase introduces different ways with to develop a game with Unity. The thesis presents a clear example of the code and applicable ways to modify programming code in Unity. The purpose of the thesis is to give a strong foundation for every avid game designer and game developer on how to use Unity and what should be considered in game development.

Keywords: Unity, game design, game developer

SISÄLLYS

1 JOHDANTO

Videopelit ovat olleet pitkään suosiossa ja videopelien käyttäjäkunta kasvaa jatkuvasti. Tästä heräsi ajatus kehittää oma videopeli ja tarkastella, mitä kaikkea pelin kehittäminen vaatii. Tämän avulla suurentaisin omaa näkemystä pelisuunnittelua sekä pelikehittämistä kohtaan. Opinnäytetyössä päädyttiin käyttämään Unity-pelimoottoria videopelin kehittämiseen.

Työn ohella on tutkittu kehuttuja videopelejä ja näistä on kerätty huomioita, mitkä tekevät pelistä hyvän. Huomioiden avulla on rakennettu ajatuskartta, mihin keskeiset ajatukset ja ideat kerättiin. Valmiilla suunnitelmalla voitiin siirtyä työssä pelimoottoriin tutustumiseen, jolla videopeli tultaisiin kehittämään. Pelimoottorilla ohjelmoidessa kerättiin ylös selkeät ohjeet, joiden avulla sama prosessi voidaan tehdä uudelleen.

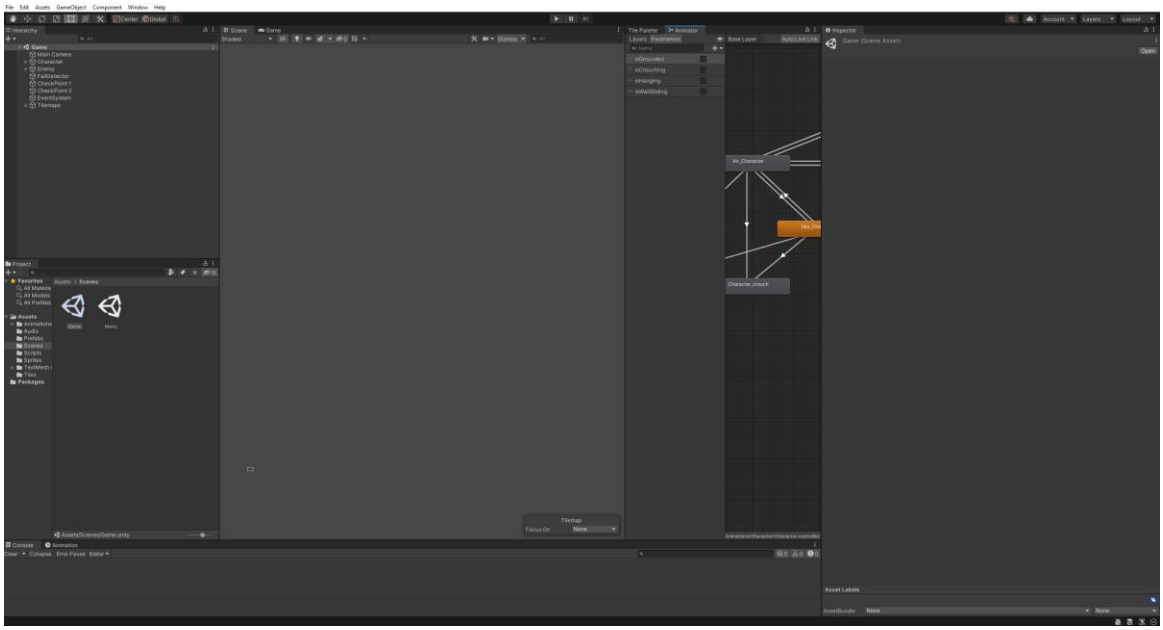
Opinnäytetyöllä tuodaan esiin videopelien keskeisiä aiheita, joita pelien suunnittelijoiden ja kehittäjien tulisi ottaa huomioon. Työn tarkoituksena on kasvattaa ymmärrystä videopelien kehittämisestä ja parantaa näkemystä, kuinka pelaajat saadaan syventymään virtuaalitodellisuuden kanssa, jotta pelikokemus olisi käyttäjälle mieluinen.

2 UNITY

2.1 Pelimoottori Unity

Unity Technologies kehittivät Unity:n, joka on monialustainen pelimoottori (Sinicki 2021). Kyseisellä moottorilla voidaan kehittää kaksi- ja kolmiulotteisia videopelejä monille eri alustoille. Näihin useisiin alustoihin kuuluvat: Windows, Linux/SteamOS, PlayStation 3 ja 4, OS X, Wii U, Xbox One ja monia muita. Unity:lla kehitetyt pelit voidaan jakaa isolle asiakaskunnalle laajan alustavalikoiman avulla, ja tästä syystä se on suosittu pelimoottori. Unity pelimoottorin työikkunan näkyy kuvassa 1.

Jos sinulla on yhtään kiinnostusta pelien kehittämiseen, Unity:n opetteleminen tulisi kuulua sinun oppimisvaiheeseesi pelien kehittämisessä. Mikä on Unity? Yksinkertaisesti, monien pelikehittäjien käyttämä työkalu, jolla pelinkehittäjät voivat luoda ja tehostaa omia luomuksiaan. Unity-ohjelmisto on tehokas, erittäin helppo käyttää sekä ilmainen. Unity on ilmainen siihen asti kun aloitat tekemään tuottoa Unity:lla rakentamistasi peleistä. (Sinicki 2021.)



KUVA 1. Unity:n työikkuna.

2.2 Ohjelmointi Unitylla

Unity tukee pelien ohjelmoinnissa kolmea eri ohjelmointikieltä: C#, Boo ja JavaScript. C# on käytetyin ohjelmointikieli Unity:ssa ja sillä on todettu saavutettavan paras suorituskyky. Tulevaisuudessa JavaScript ja Boo poistuvat valikoimasta ja jäljelle jää vain C#.

Ohjelmointi Unity:n avulla antaa suuret mahdollisuudet jokaiselle vähänkin kiinnostusta omaavalle henkilölle kehittyä peliohjelmoinnin parissa sekä kasvattaa omaa tietämystään pelien suunnittelusta ja kehittämisestä. Suuri käyttäjäkunta mahdollistaa paljon saatavilla olevaa tietoa sekä esimerkkejä. Tämä nähdään vuoden 2021 Unity:n julkaisemista tuloksista omalla nettisivullaan, mistä nähdään käyttäjäkunnan ja Unity:lla tehtyjen pelien määrä (Unity 2022).

3 PELIN SUUNNITTELU

Pelin suunnittelussa on panostettu näihin neljään alueeseen, jotka ovat pelin miellyttävyys, animaatiot, pelimaailma ja äänimaailma. Liitteistä löytyvään ajatuskarttaan on kerätty keskeisiä ajatuksia suunnittelun yhteydessä (Liite 1). Ajatuskartalla on rakennettu vakaa pohja projektille, jonka avulla on helppo nähdä, onko uusi lisäys tarpeellinen, tukeeko se kaikkea muuta tai tuleeko jotain vielä lisätä.

3.1 Pelattavuus

Termillä pelattavuus tarkoitetaan, miten pelaajat ovat vuorovaikutuksessa tietyn video- tai tietokonepelin kanssa. Pelaajan pelikokemus on yksi tärkeimmistä tekijöistä pelattavuuden kanssa ja auttaa määrittämään pelin menestystä. Pelattavuus koostuu monista tekijöistä, jotka mittaavat pelin pelaamisen helppoutta, mukavuutta ja hauskuutta. (Techopedia 2016.)

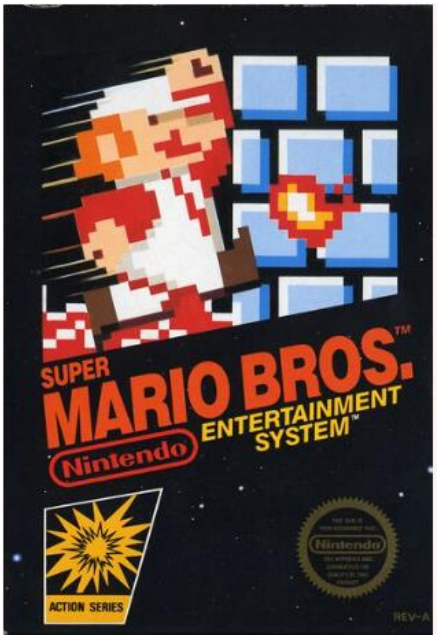
Pelin tulisi olla pelaajan hallinnassa jatkuvasti tilanteesta riippumatta, ja ylimääräiset kohtaukset peleissä voivat rikkoa immersion, koska pelaaja ei itse hallitse tilannetta. Immersiolla tarkoitetaan oman ajatuksen syventymistä virtuaalitodellisuuteen. Tästä syystä kohtaukset tulisi sisällyttää pelin sisälle, eikä erillisinä videokohtauksina, jotka keskeyttävät pelin.

Pelaajalla tulisi myös olla selkeä ohjattavuus, jota pelaaja voisi käyttää ongelmitta. Yleisin ongelma, jonka voi havaita peleissä ohjattavuuden kannalta on viive. Viiveellä tarkoitetaan aikaväliä napin painalluksesta liikkeen tapahtumiseen. Tämä seikka on erittäin tärkeä. Jos pelihahmo tottelee pelaajan käskyjä viiveellä, pelaajan kokemus ja pelaajan syventyminen peliin kärsii.

3.1.1 Tasohyppely

Tasohyppely ja kenttäsuunnittelu ovat suuri osa pelaajan pelikokemusta. Tasohyppely keskittyy pelaajan ohjaamaan pelihahmoon, jonka kyvyillä yritetään voittaa vastaan tulevat esteet ja vastustajat. Kenttäsuunnittelussa käsitellään

kuinka haastavaa ja mieluisaa tasoloikkaa peli pitää sisällään. Jos tasoloikka on liian tarkkaa tai hankalaa, pelaaja voi turhautua ja pahimmassa tapauksessa lopettaa pelin. Peli saa olla hankala, mutta vaikeusasteen tulisi nousta tasaisesti pelissä, jotta pelaajalla olisi aikaa oppia pelin mekaniikat, jolloin pelaaja olisi valmis soveltamaan niitä tulevissa haasteissa. Hyvänä esimerkkinä tasohyppelypelistä sekä kenttäsuunnittelusta toimii Nintendon vuoden 1985 Super Mario Bros. (kuva 2).



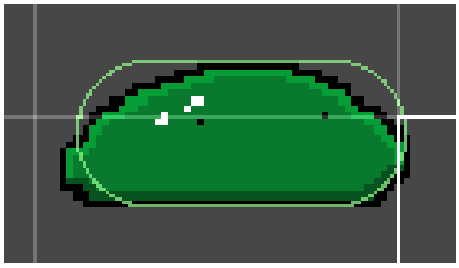
KUVA 2. Klassikkopelin Super Mario Bros. kansikuva.

Tasohyppelypeliä suunniteltaessa tulee myös ottaa huomioon pelihahmon kehittyminen pelin aikana. Voiko pelaaja myöhemmin hypätä korkeammalle tai voiko pelaaja hypätä myöhemmin toisen kerran ilmassa? Tasoloikka tulisi suunnitella tarkasti, jotta kukaan ei pystyisi tekemään mitään peliin kuulumatonta, kuten esimerkiksi ohittamalla pelin voittamisen kannalta tärkeän tapahtuman tai esineen. Kenttäsuunnittelu tulisi olla selkeää ja siihen tulisi pystyä reagoimaan.

3.1.2 Pelihahmo

Pelihahmolla voi olla suuri merkitys pelissä pelaajalle. Jos pelihahmon olemus tai tavoitteet eivät kiinnosta pelaajaa, itse pelikään ei välttämättä herätä kiinnostusta ja pelin maailmaan syventyminen voi jäädä kokematta. Tietenkään kaikki pelit

eivät tarvitse hahmolle tarinaa eikä motiivia, kuten nopeasti tekaistut kännykkäpelit tai tasohyppelypelit. Tässä pelissä on keskitytty pelihahmon yksinkertaisuuteen ja miellyttävään olemukseen (kuva 3). Pelihahmon yksinkertaisuus ei tee yksin pelistä huonoa, mutta jos pelin jokainen osa on jätetty yksinkertaiseksi, pelin tunnelmasta tulee laimea ja pelikokemus voi olla olematon.



KUVA 3. Suunniteltu pelihahmo.

3.1.3 Viholliset

Pelisuunnittelijan tulisi täyttää nämä 3 ehtoa: 1. esitä pelaajalle sarja erilaisia ja haastavia tilanteita 2. anna pelaajalle tarpeeksi tietoa, jotta pelaaja voi reagoida miten toimia erilaisissa tilanteissa 3. tarjoa pelaajalla joukko erilaisia pelin sisäisiä toimintoja, joilla on erilaiset asianmukaiset seuraamukset, ja tämän avulla pelaaja toteuttaa päätöksensä mitä näistä toiminnoista hän käyttää. (Harvey 1999.)

Videopelin vihollisten tulisi olla monipuolisia ja niiden tulisi haastaa pelaajaa erilaisilla toiminnoilla ja tilanteissa. Vihollisistyyppejä tulisi olla myös monta, jotka antavat pelaajalle vaihtelua ja haastavat pelaajan reagointikykyä käyttää oikeanlaista toimintoa. Jos pelissä on monta erilaista vihollista, pelaajan pitäisi pystyä tunnistamaan kyseinen vihollinen ensimmäisen kohtaamisen jälkeen.

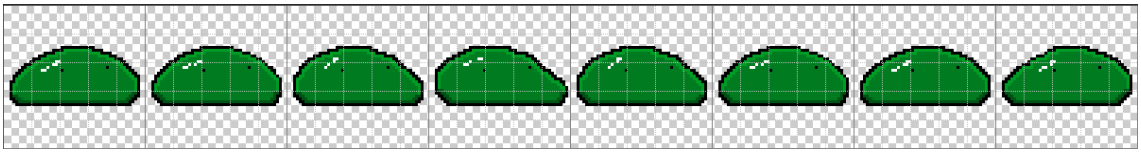
Vihollisten erottaminen toisistaan käy esimerkiksi vaihtamalla näiden ulkonäköä (kuvat 4 ja 5). Pelaaja voi nyt oppia tunnistamaan ja tarkkailemaan vihollisten erilaisia heikkouksia, joilla viholliset olisi helpompi voittaa. Kaikkien samannäköisten vihollisten tulisi toimia samalla tavalla, jotta Harveyn vlogin mukaan pelaaja voisi reagoida erilaisiin tilanteisiin oikealla tavalla.

KUVA 4. Ensimmäinen vihollinen.

KUVA 5. Toinen vihollinen.

3.2 Animaatiot

Mitä on animaatio. Animaatio on tapa valokuvata peräkkäisiä piirroksia, malleja tai jopa nukkeja luodakseen illuusion liikkeestä sarjassa. Koska silmämme voivat säilyttää kuvaa vain noin 1/10 sekunnista, aivomme yhdistävät nopeasti peräkkäiset kuvat yhdeksi liikkuvaksi kuvaksi. (Maio 2020.) Kuvassa 6 näkyy esimerkki animaation peräkkäisten kuvien sarjasta.



KUVA 6. Pelihahmon animaatio kuvasarja.

Animaatiot lisäävät peliin elävyyttä ja vaihtelua pelaajalla. Animaatioilla voidaan myös ohjeistaa pelaajaa. Esimerkiksi vihollisten tarkkailu, jos vihollinen on hyökkäämässä pelaajan kimppuun, tähän voidaan lisätä erillinen animaatio, joka varoittaa pelaajaa.

Mitä enemmän animaatiossa on kuvia, sitä sulavampi ja miellyttävämpi se on. Pelin sulavuudella on keskeinen merkitys pelaajan immersion syventämisen kanssa. Huonolla animaatiolla pelihahmon ohjaaminen voi tuntua oudolta ja huonolta, joten pelistä tulisi saada ulkonäöltään mahdollisimman sulava. Toinen suuri tekijä animaation sulavuuteen on animaation pyörittämisnopeus. Liian nopealla animaatiolla voidaan tuoda pelaajalle sekavuutta, eikä pelaaja välttämättä kerkeäisi ymmärtämään mitä tapahtui. Liian hidas animaatio voi ohjeistaa pelaajaa väärin. Hitaat animaatiot eivät kuitenkaan aina ole huono asia, mutta tasohyppelypeleissä pelihahmon ohjaimet tulisivat olla täsmälliset. Esimerkiksi, täsmällisillä ohjaimilla hyppyanimaatio saadaan vaikuttamaan, että pelihahmo yrittäisi hypätä, vaikka on jo ilmassa.

3.3 Pelimaailman efektit

Näillä efekteillä korostetaan pelimaailmassa tapahtuvia tapahtumia. Esimerkiksi pelin kuvaruudun heiluminen, jolla voidaan ilmoittaa pelaajalle maan tärinästä tai vaikka äkillisestä liikkeestä. Tähän sisältyvät myös kaikki veriroiskeet toimintapeleissä tai pelihahmojen kanssakäyminen ilman sanoja, kuten esimerkiksi myrskypilvi pään päällä. On siis selvää, että tämän vaikutus pelimaailmassa on suuri.

Nämä efektit eivät kuitenkaan ole pelin kannalta pakollisia, koska peli voi hyvin toimia ilman erinäisiä pelin kuvaruudun heilumisia tai roiskeita. Näillä kuitenkin on suuri vaikutus pelaajaan, koska ne antavat pelaajalle selvät indikaatiot, mitä pelaaja tekee tai mitä hänen pelihahmonsa ympärillä tapahtuu. Tämän lisäksi suurin osa pelimaailman efekteistä ovat silmää miellyttäviä.

3.4 Äänimaailma

Peli saa äänien kautta todella paljon syvyyttä ja saa pelaajan uppoutumaan oikeanlaisen äänimaailman kanssa peliin paremmin. Äänien tulisi kuitenkin vastata todellisuutta, jotta pelaaja voisi samaistua pelin äänimaailman kanssa. Esimerkiksi jos, pelissä käytetään samaa ääniefektiä puisen ja metallisen objektin

kanssa, äänimaailma ei sopisi pelimaailman kanssa ja näin peliin syventyminen voi heikentyä. Äänimaailmaa kehittäessä tulisi ottaa huomioon pelin grafiikka, jotta ne sopisivat yhteen. Yhteensopivalla äänimaailmalla ja grafiikalla peliin on helpompi keskittyä, jolloin pelikokemus on huomattavasti miellyttävämpi. Ristiriitainen tunnelma, kuten pimeä luola ja iloinen musiikki eivät tue toisiansa ja näin pelikokemus kärsii.

Pelimusiikilla on suuri merkitys pelattavuuden kannalta. Musiikilla voidaan ohjeistaa pelaaja tietynlaisissa tilanteissa. Esimerkiksi rauhallisella musiikilla voidaan ilmoittaa pelaajan olevan turvassa, kun taas nopeatempoisella jännittävällä musiikilla voidaan ilmoittaa mahdollisesta vaarasta. Tempolla voidaan luoda pelaajalle erilaisia tunteita, jotka parhaassa tapauksessa johtaa haluttuihin reaktioihin. Musiikissa tempolla tarkoitetaan kappaleen nopeutta. Tempolla on siis helppo luoda tilanteeseen sopivampaa musiikkia. Vaarallisessa tilanteessa nostetaan tempoa ja rauhallisessa/turvallisessa tilanteessa tempoa lasketaan.

Äänimaailmalla on hyvin keskeinen merkitys pelien luomisessa. Tällä saadaan pelin tunnelmaan enemmän syvyyttä. Pelimusiikki kulkee käsikädessä ääniefektien kanssa ja nämä tukevat pelattavuuden rytmiä ja vaikuttavat myös olennaisesti pelaajan pelikokemukseen. Pelin äänimaailmaan ei välttämättä kiinnitetä huomiota yhtä paljon kuin pelin muihin elementteihin, mutta sen puuttuminen huomataan välittömästi. Musiikilla ja ääniefekteillä vaikutetaan voimakkaasti myös pelaajan immersioon. (Huiberts 2010.)

3.4.1 Musiikki

Pelimusiikki on usein interaktiivista musiikkia. Interaktiivisella musiikilla tarkoitetaan, että musiikki mukautuu pelin tapahtumiin tai pelaajan toimintaan. Esimerkiksi pelissä alueelta siirtyminen toiselle alueelle. Siirrytään rauhallisesta viidakosta, vaikka tulivuoren tuntumaan, missä tunnelman tulisi mukautua tilanteeseen sopivaksi. Interaktiivisella musiikilla saadaan pelaajalle rakennettua syvempää immersiota.

Ohjelmoinnillisesta näkökulmasta pelimusiikin interaktiivisuus voidaan toteuttaa esimerkiksi hyödyntämällä laukaisimia. Käytännössä laukaisin toimii eräänlaisena laukaisevana tekijänä, jolla saadaan ääniraidat toistumaan oikeassa ajankohdassa. (Brandon 2005.) Esimerkiksi jos pelihahmo ylittää jonkin tietyn rajan, lisätään tähän väliin laukaisin, jolla saadaan musiikki vaihtumaan tunnelmaan sopivammaksi.

3.4.2 Ääniefektit

Ääniefektit ovat pelissä lähes yhtä tärkeä osa kuin pelin musiikki. Ääniefekteillä saadaan äänimaailmaa syvennettyä vielä lisää ja se antaa kaikelle pelin sisäiselle tekemiselle lisäpotkua! Pienetkin lisäykset elävöittävät maailmaa paljon, kuten ponnistus maasta tai laskeutuminen taas maankamaralle.

Peleissä musiikki ja äänet vaikuttavat pelaajan immersioon, vaikka pelaaja ei sitä huomaisikaan (Huiberts 2010). Esimerkiksi jos kiveä lyödään puisella mailalla ja sama ääniefekti kuuluu lyötäessä samaa kiveä rautalapiolla. Pelaaja usein vertaa ääniä elämänsä aikana kuuleviinsa ääniin, joten jos pelin äänimaailma on ristiriidassa pelaajan oman ajatusmaailman kanssa, immersio voi heikentyä.

Valtava osa viestinnästä pelin sisällä tapahtuu äänimaailmalla, peliympäristö tuottaa ääntä, jonka voi linkittää helposti pelimaailman fiktiiviseen äänimaailmaan. Näillä äänillä kommunikoidaan pelin omaa maailmaa sekä siinä esiintyviä esineitä. Esimerkkejä näistä äänistä ovat moottoreiden äänet kilpapeleissä, pelihahmon käveleminen mudassa, ammuntapelit ja ukkosmyrskyt sekä sateet. (Huiberts 2010.)

4 PELIN OHJELMOINTI

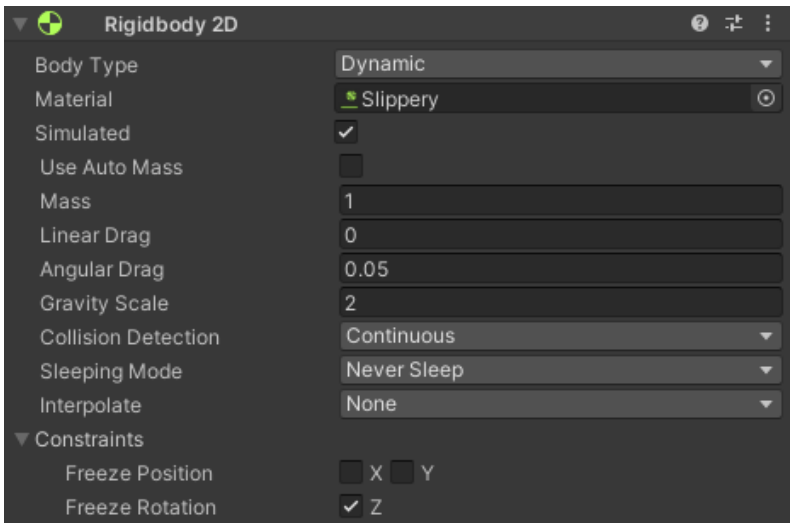
Ohjelmoinnissa kannattaa miettiä tarkkaan, mitä on ohjelmoimassa, mitä kaikkea siinä tarvitsee ohjelmoida ja miten sen toteuttaisi. Hyvällä suunnittelulla voidaan ennakoida tulevia haasteita, jotta ohjelmointivaiheessa ei jäätäisi jokaiseen ongelmakohtaan. Hyvällä suunnittelulla vältetään myös ohjelmointivirheitä sekä ohjelmointikoodin osien lisäämistä jälkeen päin. Ohjelmointikoodia muuttaessa ohjelmointikoodi tulee tarkistaa, että se toimii samalla tavalla kuin aikaisemmin.

4.1 Fysiikat

Pelissä pitää tietenkin olla säännöt ja pelin fysiikat tuovat ne hyvin esiin. Kuinka nopeasti pelaaja voi liikkua tai kuinka korkealle pelaaja voi hypätä? Pelaajan pitäisi pystyä vaikuttamaan ”kaikkeen” pelissä, lukuun ottamatta taustaan ja kentän tekstuureihin. Mitä enemmän pelaaja voi jättää jälkiä pelissä, sitä paremmin pelaaja voi uppoutua peliin ja nauttia pelistä vielä enemmän!

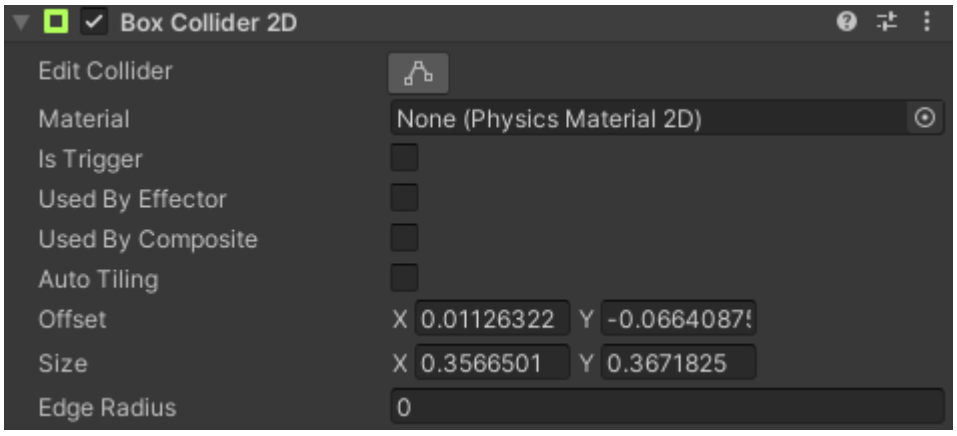
4.1.1 Unity:n fysiikat

Unity pitää sisällään valmiita komponentteja, joita lisäämällä objekteille saa erilaisia ominaisuuksia. Yksi Unity:n tärkeimmistä komponenteista on ”Rigidbody” ja tämän pelin yhteydessä ”Rigidbody2D”, koska kyseessä on 2D-peli. Rigidbody lisää objektille perusfysiikoita, mitkä näkyvät kuvassa 7. Tärkeimpiä näistä ominaisuuksista pelin tekemisessä on painovoima, massa, objektin materia ja törmäyksen havaitseminen (collision detection), joka tunnistaa, jos objekti koskee johonkin toiseen objektiin.



KUVA 7. Rigidbody2D-komponentin ominaisuudet.

Toinen fysiikoiden kannalta tärkeä komponentti on törmäyskone (collider), joka havaitsee muut objektit. Tätä komponenttia voidaan vapaasti muokata kyseisen objektin kokoiseksi muokkaa törmäyskonetta-näppäimellä (Edit Collider), jotta kaikki objektin reunat täsmäisivät annetun kuvahahmon (sprite) reunoja. Tällä voidaan myös lisätä objekteille ominaisuus, etteivät ne voi mennä toistensa läpi, kuten esimerkiksi pelihahmo ja kaikki tasot millä pelihahmo voi seistä. Yksinkertaisen 2D-laatikkotörmäyskoneen (Box Collider 2D) valikko näkyy kuvassa 8.



KUVA 8. 2D-laatikkotörmäyskoneen komponentin ominaisuudet.

4.1.2 Fysiikoiden ohjelmointi

Unity:n sisäisiä fysiikoita voidaan muokata ohjelmointikoodin sisällä kutsumalla kyseisen objektin komponenttia. Esimerkiksi pelihahmolle on lisätty komponentti

Rigidbody2D ja sitä kutsutaan ohjelmointikoodissa kuvan 9 mukaisesti. Ohjelmoinnissa Rigidbody2D:tä on käytetty nimellä ”rb”, joka saa arvonsa heti ohjelmointikoodin alussa Unity:n Rigidbody2D:ltä kuvan 10 mukaisesti.

```
65 references | 16 references
private Rigidbody2D rb;
```

KUVA 9. Rigidbody2D:n kutsuminen ohjelmointikoodin sisällä.

```
void Start(){
    rb = GetComponent<Rigidbody2D>();
```

KUVA 10. Rigidbody2D antaa arvonsa muuttujalle rb.

Tätä on käytetty tässä projektissa esimerkiksi painovoiman suuruuden muuttamiseksi. Painovoiman muuttaminen toimii käyttämällä juuri luotua muuttujaa rb, jonka jälkeen ohjelmointikoodissa ilmoitetaan mitä arvoa komponentista halutaan muuttaa (kuva 11).

```
// Painovoiman kääntäminen
1 reference
void Gravity(){
    rb.gravityScale *= -1;
    startGravity *= -1;
    Rotation();
}
```

KUVA 11. Ohjelmoinnin osa ”Painovoiman kääntäminen”.

4.2 Hahmon ohjelmointi

Ohjelmoinnin alussa annetaan kaikille muuttujille arvo, joka vaihtuu ohjelmointikoodissa tarpeen vaatiessa esimerkiksi, kuinka nopeasti hahmolla voidaan liikkua tai kuinka korkealle pelaaja haluaa hypätä.

Unity ymmärtää myös vaaka- ja pystysuunnan liikkeen, mitä käytettiin apuna hahmon liikkumiseen x-akselilla. Kun pelaaja haluaa liikkua, tulee ohjelmointikoodissa tässä tapauksessa nostaa vaakatason voimaa. Ohjelmointikoodin tulee

myös tietää, kumpaan suuntaan pelaaja haluaa liikkua. Tämä tiedetään vaakatason arvosta. Arvo saadaan joko painamalla A- tai D-näppäintä tai nuolinäppäimillä vasen tai oikea. Vasenta nuolinäppäintä tai A-näppäintä painamalla arvo on -1, jolloin pelihahmo liikkuu vasemmalle. Painamalla D-näppäintä tai oikeata nuolinäppäintä arvo muuttuu 1:ksi, jolloin pelihahmo liikkuu oikealle.

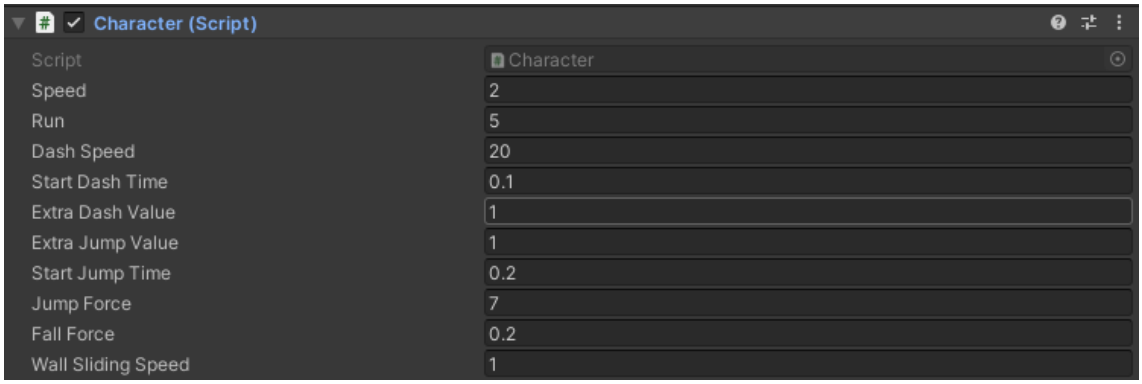
Kuvassa 12 ohjelmointikoodi katsoo vaakatason arvon muuttujan ”moveInput” sisään ja seuraavaksi laitetaan hahmo liikkumaan kertomalla arvot yhteen. Nopeutta siis voidaan vaihtaa muuttaessa nopeuden (speed) suuruutta. Koska kyse on vaakasuorasta (Horizontal) liikkeestä y-akselin arvoa ei muuteta. Painamalla vasenta Shift-näppäintä, nopeuden arvo ”speed” vaihdetaan arvoksi ”run” kuvan 13 mukaisella tavalla. Näitä arvoja voidaan muuttaa Unity:n sisällä tarkastajaikkunassa (kuva 14).

```
moveInput = Input.GetAxisRaw("Horizontal");
rb.velocity = new Vector2(moveInput * speed, rb.velocity.y);
```

KUVA 12. Vaakatason arvon tarkistaminen ja kertominen nopeuden kanssa.

```
if(Input.GetKey(KeyCode.LeftShift)){
    moveInput = Input.GetAxisRaw("Horizontal");
    rb.velocity = new Vector2(moveInput * run, rb.velocity.y);
}
```

KUVA 13. Vaakatason nopeuden muuttaminen painamalla Shift-näppäintä.



KUVA 14. Pelihahmon ohjelmointikoodi Unity:n tarkastaja-ikkunassa.

Hahmon hypyssä muutetaan vain y-akselin voimaa, jolloin hahmon hyppiminen ei vaikuta liikkumisen nopeuteen. Hahmon tulisi siis hypätä, kun jotain tiettyä painiketta painetaan. Kuvan 15 mukaisesti välilyöntiä painamalla hahmon voima 2-

dimension maailmassa kasvaa ylöspäin. Voiman suureneminen ylöspäin antaa hahmolle kyvyn hyppiä. Hypyssä on otettu huomioon, kuinka kauan pelaaja painaa välilyöntiä, joka vaikuttaa hypyn korkeuteen.

Pelissä on myös mahdollista hahmolla hypätä kerran ilmassa, joka antaa mahdollisuuden tuplahyppyyn. Ohjelmointikoodissa tarkastetaan, koskeeko pelihahmo maata vai onko pelihahmo ilmassa. Pelihahmon ollessa maassa, pelaaja voi hyppiä normaalisti, mutta ilmassa ohjelmointikoodi tarkistaa onko tuplahyppy jo käytetty. Jos pelaaja on hypännyt ilmassa jo kerran, tulee pelihahmon laskeutua takaisin maahan, ennen kuin pelihahmo voi hypätä uudelleen.

```
// Hyppy ja tuplahyppy
if(Input.GetKeyDown(KeyCode.Space) && extraJumps > 0 && rb.gravityScale > 0){
    rb.velocity = Vector2.up * jumpForce;
    isJumping = true;
    jumpTime = startJumpTime;
    --extraJumps;
} else if(Input.GetKeyDown(KeyCode.Space) && extraJumps == 0 && isGrounded == true && rb.gravityScale > 0){
    rb.velocity = Vector2.up * jumpForce;
    isJumping = true;
    jumpTime = startJumpTime;
}
```

KUVA 15. Pelihahmon hyppy ja tuplahyppy ohjelmointikoodissa.

Kuvissa 16 ja 17 käydään läpi hahmon ampumista. Ampuminen tapahtuu H-näppäimestä. Kun H-näppäintä painetaan, siirrytään ohjelmointikoodissa funktion ”Shoot” kohdalle, joka tarkistaa lentoradan mahdollisuudet ja lähettää ammuksen liikkeelle. Lentoradan mahdollisuuksilla tarkoitetaan, mihin suuntaan pelaaja katsoo, katsooko pelaaja ylöspäin vai ilmassa alaspäin. Luodin esivalmistelu (bulletPrefab) pitää sisällään ammuksen, johon on liitetty oma nopeutensa ja Rigidbody2D.

```
// Ampuminen
if(Input.GetKeyDown(KeyCode.H)){
    Shoot();
}
```

KUVA 16. Jos H-näppäintä painetaan, ohjelmointikoodi siirtyy ”Shoot” funktion sisään.

```
void Shoot(){

    // Painovoiman ollessa < 0
    if(rb.gravityScale < 0 && aimingUp == true){
        float angle = facingRight ? 270f : 270f;
        Instantiate(bulletPrefab, firePointUp.position, Quaternion.Euler(new Vector3(0f, 0f, angle)));
    } else if(rb.gravityScale < 0 && aimingDown == true){
        float angle = facingRight ? 90f : 90f;
        Instantiate(bulletPrefab, firePointDown.position, Quaternion.Euler(new Vector3(0f, 0f, angle)));
    } else if(rb.gravityScale < 0 && isWall == true && aimingUpwards == false){
        float angle = facingRight ? 0f : 180f;
        Instantiate(bulletPrefab, firePointBack.position, Quaternion.Euler(new Vector3(0f, 0f, angle)));
    } else if(rb.gravityScale < 0 && isWall == false && aimingUpwards == true){
        float angle = facingRight ? 225f : 315f;
        Instantiate(bulletPrefab, firePointUpwards.position, Quaternion.Euler(new Vector3(0f, 0f, angle)));
    } else if(rb.gravityScale < 0 && isWall == true && aimingUpwards == true){
        float angle = facingRight ? 315f : 225f;
        Instantiate(bulletPrefab, firePointBackUpwards.position, Quaternion.Euler(new Vector3(0f, 0f, angle)));
    } else if(rb.gravityScale < 0 && isWall == false && aimingUpwards == false){
        float angle = facingRight ? 180f : 0f;
        Instantiate(bulletPrefab, firePointFront.position, Quaternion.Euler(new Vector3(0f, 0f, angle)));
    }
```

KUVA 17. Shoot-funktio.

4.3 Vihollisten ohjelmointi

Viholliselle ohjelmoitiin samaan ohjelmointikoodiin monta erilaista vaihtoehtoa, mitkä voidaan ottaa käyttöön Unity:n tarkastaja ikkunassa (kuva 18). Vaihtamalla näitä muuttujia todeksi tai vääräksi vihollinen reagoi pelin sisällä eri tavoin. Kuitenkin jos mitään näistä ei valita, vihollinen ei tee mitään.

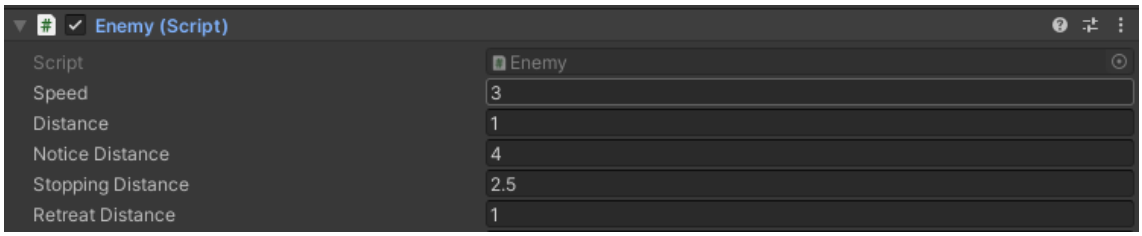
KUVA 18. Vihollisen ohjelmointikoodi Unity:n sisällä.

Vaihtamalla vartiointi (Patrol) muuttujan todeksi, vihollinen vahtii kyseisen laattarivin eli laatoilla piirretyn tason päällä, millä vihollinen on. Ohjelmointikoodissa tunnistetaan, jos vihollinen saapuu joko laattarivin reunalle tai vastaan tulee seinä, jolloin vihollinen kääntyy ja jatkaa vartiointia toiseen suuntaan (kuva 19).

```
if(groundInfo.collider == false || isFront == true){
    if(movingRight == true){
        transform.eulerAngles = new Vector3(0, -180, 0);
        movingRight = false;
    } else {
        transform.eulerAngles = new Vector3(0, 0, 0);
        movingRight = true;
    }
}
```

KUVA 19. Jos vihollinen saapuu laattarivin reunalle tai vihollinen kohtaa esteen, vihollinen kääntyy.

Vaihtamalla puskemisen (Push) muuttujan todeksi, jos pelaaja tulee vihollisen huomaamisen etäisyydelle (Notice Distance), vihollinen muuttaa suuntansa kohti pelihahmoa, kunnes pelihahmo poistuu tältä etäisyydeltä (kuvat 20 ja 21). Vaihtamalla perääntymisen (Retreat) muuttujan todeksi, vihollinen seuraa pelihahmoa sen huomattuaan, mutta pysähtyy pysähdysetäisyydellä (Stopping Distance) ja pakenee pelihahmoa tämän tullessa peruutusetäisyydelle (Retreat Distance) (kuva 22).



KUVA 20. Viholliselle luodut arvot ohjelmointikoodissa.

```
if((Vector2.Distance(transform.position, Player.position) < noticeDistance)){
```

KUVA 21. Jos pelihahmo on lähempänä, kuin huomaamisen etäisyyden arvo.

```
if(Vector2.Distance(transform.position, Player.position) > stoppingDistance){
    transform.position = Vector2.MoveTowards(transform.position, Player.position, speed * Time.deltaTime);
} else if(Vector2.Distance(transform.position, Player.position) < stoppingDistance && Vector2.Distance(transform.position, Player.position) > retreatDistance && Retreat == true)
    transform.position = this.transform.position;
} else if(Vector2.Distance(transform.position, Player.position) < retreatDistance && Retreat == true){
    transform.position = Vector2.MoveTowards(transform.position, Player.position, -speed * Time.deltaTime);
} else if(Push == true) {
    transform.position = Vector2.MoveTowards(transform.position, Player.position, speed * Time.deltaTime);
}
```

KUVA 22. Vihollisen erilaiset toiminnat eri muuttuja valinnoilla.

Vaihtamalla ampumisen (Shoot) muuttujan todeksi, vihollinen laukaisee ammuksen kohti pelihahmoa ohjelmointikoodissa luodun muuttujan aika ammusten välissä (timeBetweenShots) arvon mukaisesti. Ammus lentää tämänhetkiseen kohtaan, missä pelihahmo sijaitsee ja katoaa sen saavutettuaan (kuva 23).

```
if(Shoot == true){
    if(timeBetweenShots <= 0 && Vector2.Distance(transform.position, Player.position) < noticeDistance){
        Instantiate(Projectile, transform.position, Quaternion.identity);
        timeBetweenShots = startTimeBetweenShots;
    } else {
        timeBetweenShots -= Time.deltaTime;
    }
}
```

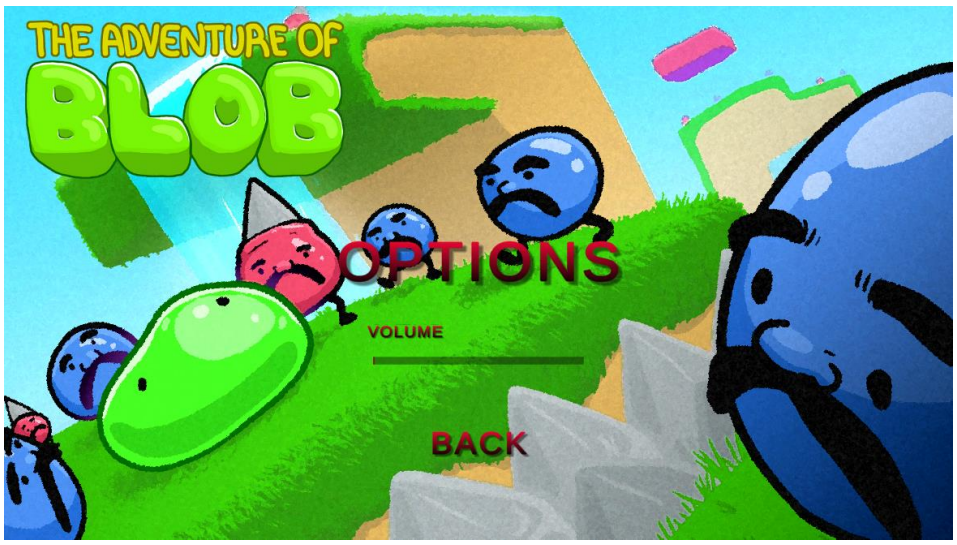
KUVA 23. Vihollisen ampumistoiminto.

4.4 Valikot

Otsikkoruutu avautuu pelissä ensimmäisenä, jonka näkymä näkyy kuvassa 24. Painamalla mitä tahansa näppäintä peli siirtyy alkuvalikkoon. Alkuvalikossa on 3 mahdollista painiketta, jotka korostuvat siirryttäessä hiirellä niiden päälle. 3 painiketta ovat pelaa, asetukset ja lopeta (kuva 25). Pelaa-painikkeesta peli-ikkuna avautuu ja peli alkaa. Asetukset-painikkeesta avautuu näkymä, asetukset, missä voidaan säätää äänen voimakkuutta sekä palata alkuvalikkoon painamalla palaa-painiketta (kuva 26). Lopeta-painikkeesta sovellus sulkeutuu.

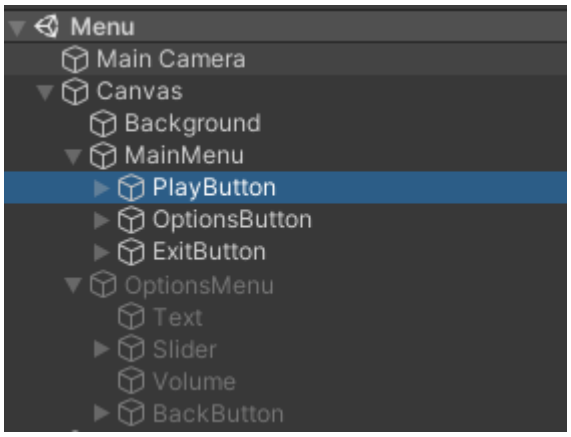
KUVA 24. Pelin otsikkoruutu.

KUVA 25. Pelin alkuvalikko.



KUVA 26. Pelin asetukset valikko.

Painikkeet luodaan Unity:n sisällä ja niiden vaikutusta voidaan vaihtaa tarkastaja-ikkunan sisällä (kuvat 27 ja 28). Tarkastaja-ikkunan sisällä valitaan, milloin halutaan että jotain tapahtuu ja mitä tapahtuu. Alkuvalikosta voidaan jatkaa kulkua peliin ja asetuksiin tai sammuttaa peli.



KUVA 27. Alkuvalikko ja asetukset valikko rakennettuna valikkonäkymän alle.

KUVA 28. Asetukset-painiketta painamalla asetukset valikko muuttuu aktiiviseksi ja alkuvalikko muuttuu inaktiiviseksi.

Näkymien vaihto tapahtuu ohjelmointikoodin sisällä kuvien 29 ja 30 mukaisesti. Näkymä vaihtuu, kun nostetaan Unity:n sisällä rakennus indeksiä isommaksi. Näkymien indeksin suuruus riippuu, missä järjestyksessä ne halutaan tuoda näkyviin pelissä (kuva 31).

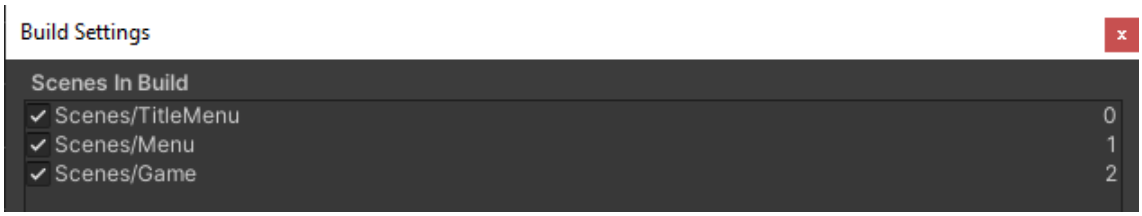
```
if(Input.anyKey){
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```

KUVA 29. Otsikkoruudusta siirtyminen seuraavaan rakennus indeksiin.

```
public void PlayGame(){
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}

public void QuitGame(){
    Debug.Log ("QUIT!");
    Application.Quit();
}
```

KUVA 30. Alkuvalikosta siirtyminen peli-ikkunaan ja lopetusnäppäimen lopetustoiminto.



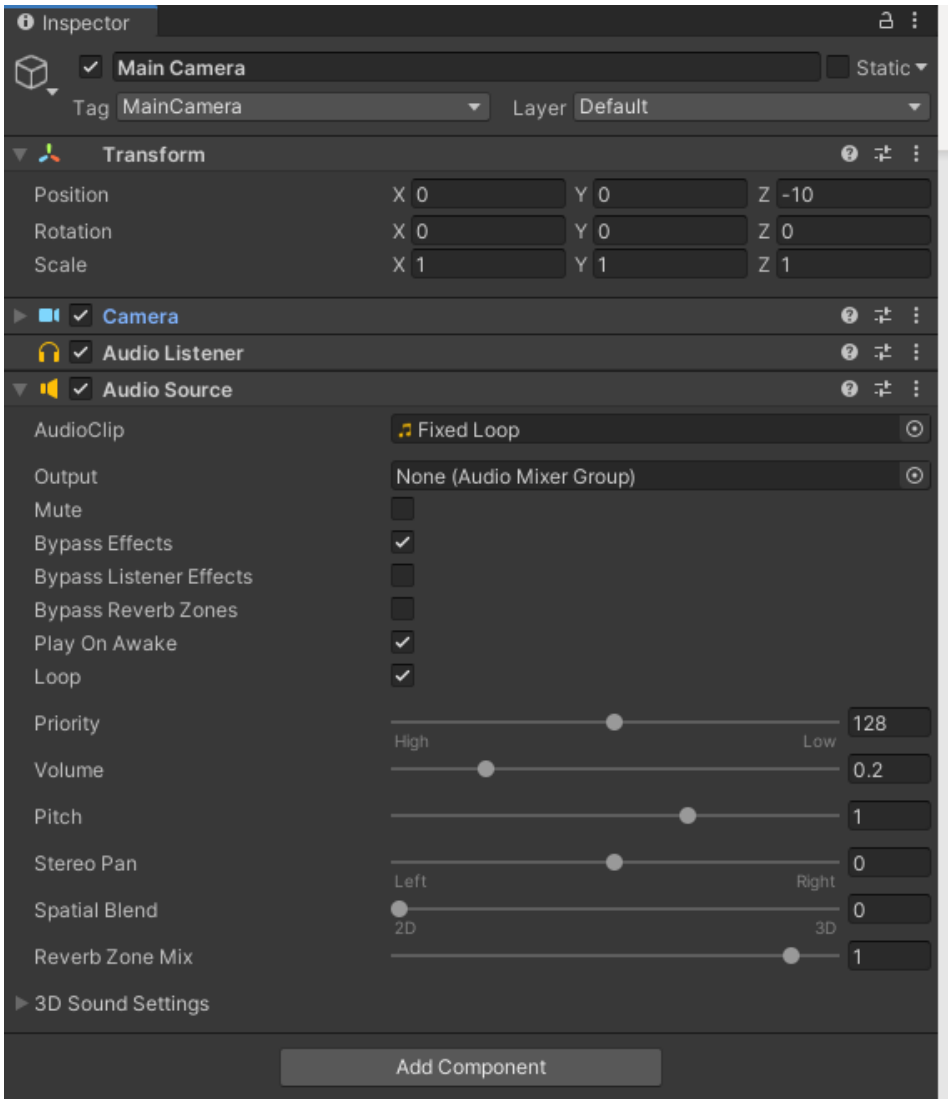
KUVA 31. Rakennus asetukset ja näkymien rakennus indeksit.

4.5 Äänet ja efektit

Äänien ohjelmointi peliin Unity:n avulla on helppoa. Musiikin tapauksessa ääniraita liitetään johonkin tiettyyn objektiin, jonka halutaan soittaa kyseistä raitaa. Pitää kuitenkin muistaa, miten äänimaailmaa halutaan ohjata. Kuuluuko musiikki kaiken aikaa ja mistä suunnasta musiikki kuuluu?

Työssä ääniraita on liitetty pelin kameraan, jotta musiikki on aina mukana. Kuvasta 32 nähdään asetus tilan vaihto ”Spatial Blend”, jolla voidaan vielä säätää

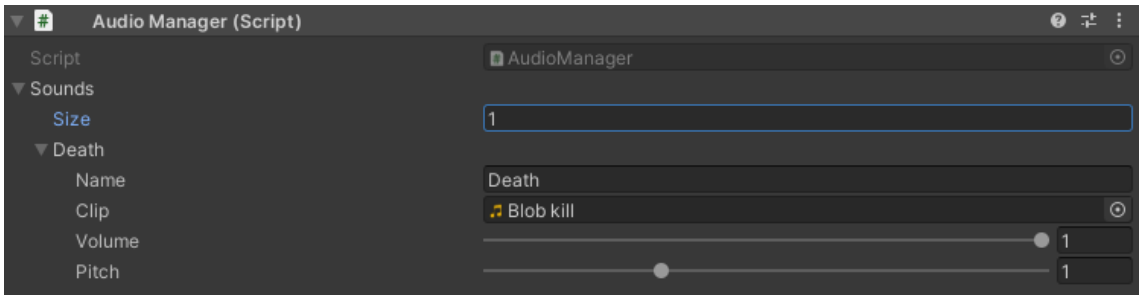
erikseen ääniraidan tulevan 2D:nä tai 3D:nä. Tätä voidaan esimerkiksi käyttää, jos halutaan jonkin ääniraidan alkavan kuulua kauempaa ja vahvistua pelaajan lähestyessä kohdetta. 3D-äänimaailmaa voi myös käyttää varoitukseen nopeasti lähestyvästä ammuksesta, mitä pelaaja ei välttämättä vielä näe.



KUVA 32. Kameraan lisätyt äänikomponentit.

Koska työssä on kaksi näkymää, tarvitsevat molemmat näkymät omat äänen kuuntelija ja äänilähde (Audio Listener, Audio Source) -komponentit, joilla voidaan pelin ääniraidat soittaa. Äänilähteen sisältä löytyy myös mahdollisuus asettaa ääniraita pyörimään silmukassa. Asettamalla ääniraita silmukkaan tulee tarkistaa ääniraidan alku- ja loppusointu, jotta ääniraita jatkaisi kulkuansa ongelmitta ja korvaa ärsyttämättä.

Ääniraitoja voidaan myös kiinnittää objekteihin, joita kutsutaan usein, kuten esimerkiksi pelin kaikki ammukset. Työssä on myös ohjelmoitu mahdollisuus sijoittaa ääniraidat ohjelmoinnin avulla oikeisiin tilanteisiin. Ohjelmoinnilla on myös luotu uusi ikkuna Unity:n sisään, missä voidaan muuttaa äänenvoimakkuutta sekä sävelkorkeutta. Kuvan 33 äänenhallinta (Audio Manager) voidaan kutsua ohjelmointikoodissa ja nimetä ääniraita, mikä halutaan soittaa (kuva 34).



KUVA 33. Äänenhallinta, johon on liitetty pelihahmon kuolema ääniefekti.

```
void OnTriggerEnter2D(Collider2D collision){
    if(collision.tag == "HurtBox"){
        transform.position = respawnPoint;

        FindObjectOfType<AudioManager>().Play("Death");

    } else if(collision.tag == "CheckPoint"){
        respawnPoint = transform.position;
    }
}
```

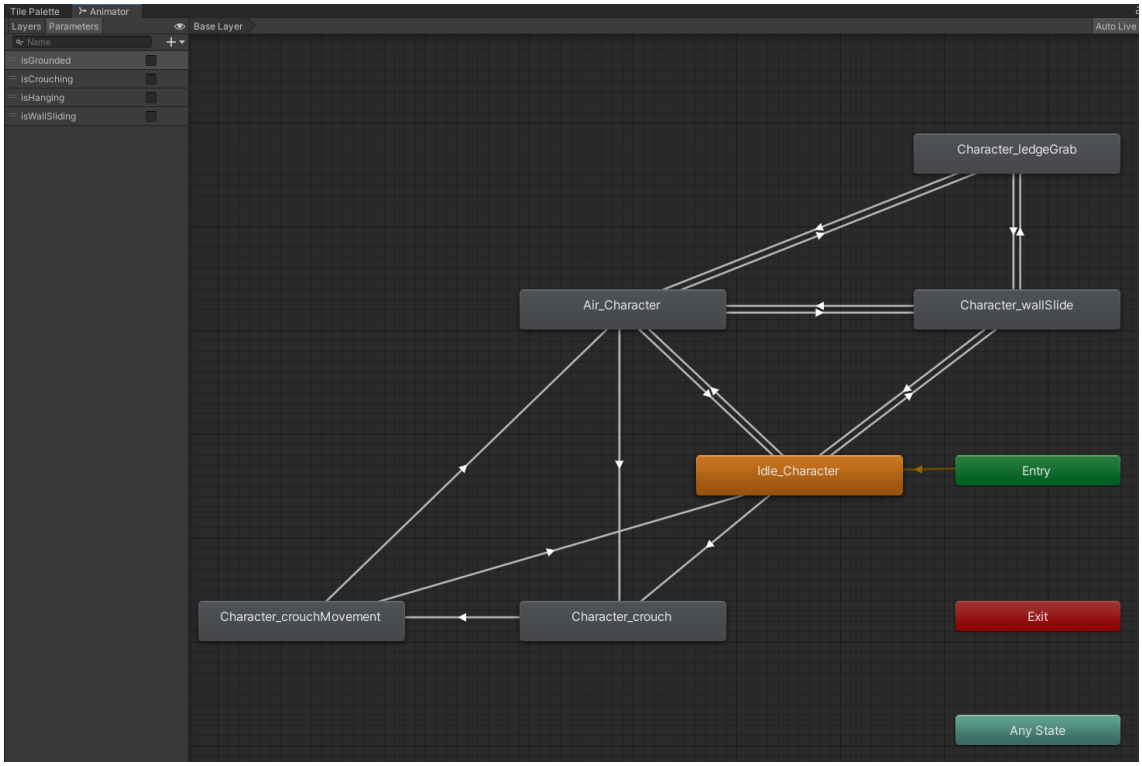
KUVA 34. Ääniraidan kutsuminen ohjelmointikoodissa.

4.6 Animaatiot

Animaatioissa on käytetty apuna Unity:n omaa animaattorityökalua (animator). Tällä valitaan objekti, mihin halutaan lisätä animaatioita. Tämän jälkeen animaatioiden nopeutta voidaan muuttaa Unity:n sisällä. Unity piirtää myös näistä objektiin lisätyistä animaatioista oman animaatiokartan.

Kuvasta 35 nähdään eri animaatiot linkitettyinä toisiinsa animaatiokartassa. Jokaisessa linkissä näkyy suuntanuoli, mikä kuvaa mistä animaatiosta voidaan siir-

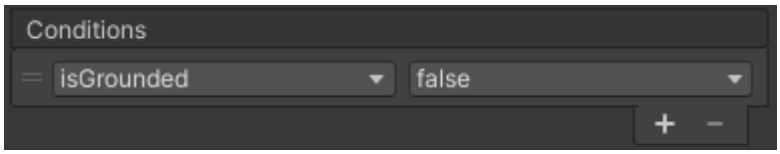
työä toiseen animaatioon. Jotta pelissä osataan kutsua oikea animaatio, tulee ohjelmointikoodissa viitata animaattori-ikkunaan kuvassa 36 näytetyllä tavalla ja muuttaa sen parametreja halutulla tavalla aina tarvittaessa. Esimerkiksi jos halutaan siirtyä ”idle”-animaatiosta ”air_character”-animaatioon tulee linkkien ehtojen vastata tämänhetkisiä parametreja (kuva 37).



KUVA 35. Animaattori ikkunan animaatiokartta.

```
animator = GetComponent<Animator>();
```

KUVA 36. Animaattori-työkalun kutsuminen ohjelmointikoodissa.



KUVA 37. Animaatio linkin ehtoikkuna.

Työssä on käytetty parametrien vaihtamiseen bool-muuttujia, mitkä tarkistavat onko jokin asia totta. Parametri ”isGrounded” tarkistaa onko pelihahmo maassa, jos pelihahmo koskettaa maata toistetaan ”idle_Character”-animaatio. Parametrien muuttaminen ohjelmoinnin sisällä toimii kuvan 38 mukaisella tavalla. Kuvan

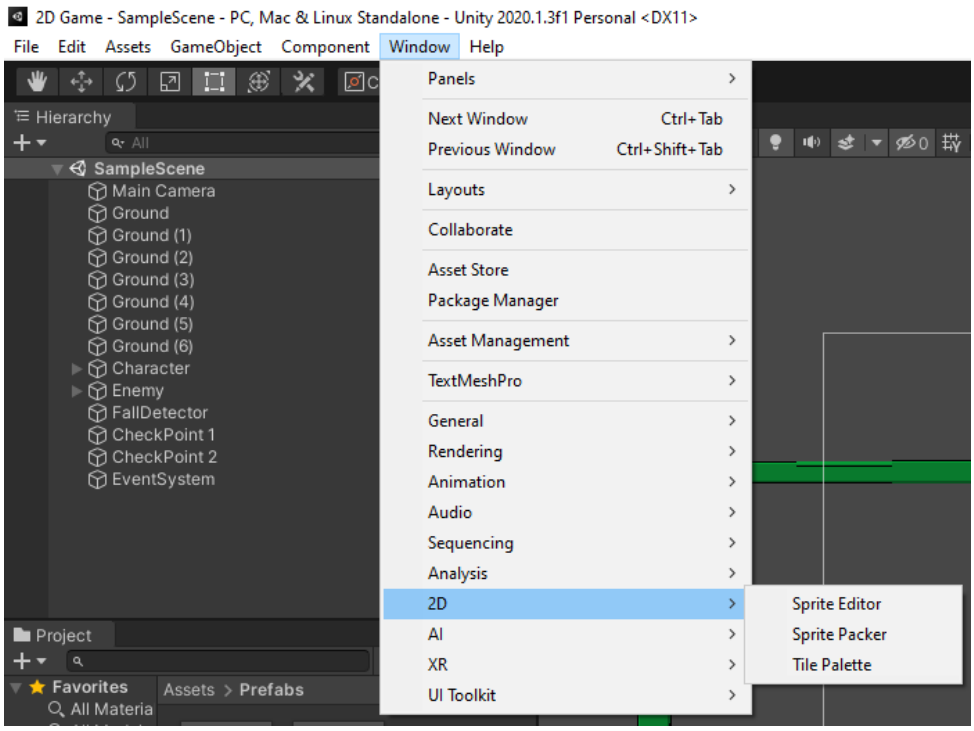
mukaan vain ”isHanging” on totta, jolloin pelihahmon roikkua-animaatiota toistetaan.

```
animator.SetBool("isCrouching", false);
animator.SetBool("isGrounded", false);
animator.SetBool("isWallSliding", false);
animator.SetBool("isHanging", true);
```

KUVA 38. Animaatio parametrien vaihtaminen ohjelmointikoodin sisällä.

5 PELIN TEKSTUURIT

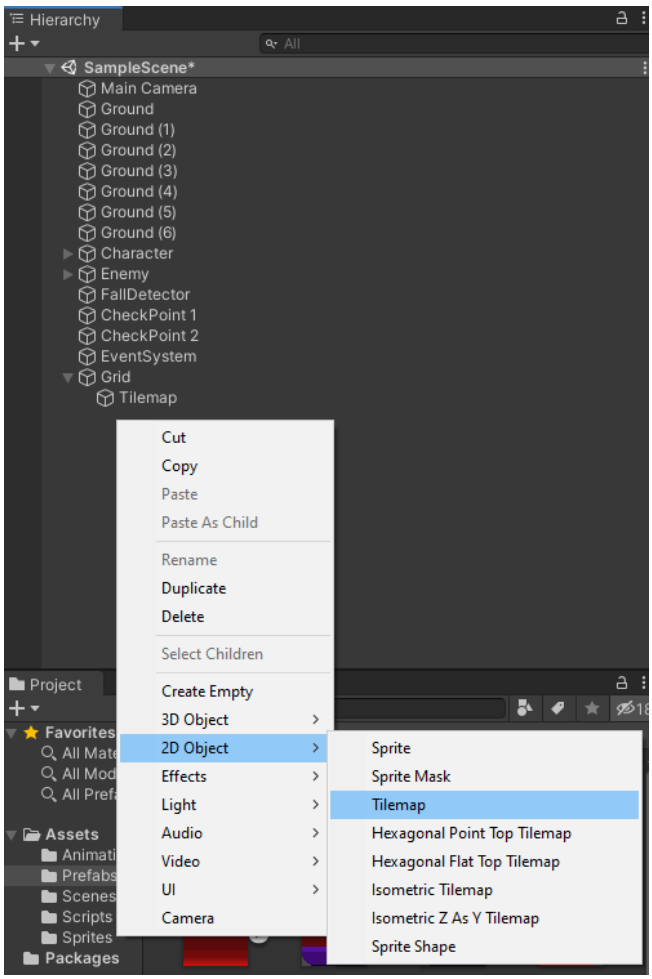
Tekstuureissa on käytetty niin sanottua pikselitaidetta. Nimensä mukaan laattojen luominen tapahtuu pikselitasolla. Laattoja liittämällä yhteen luodaan laattarivi, joka siis on vain osa pelin tasosta. Työssä on luotu oma laattasarja (tileset), joka sisältää kaikki laatat yhdessä paketissa. Laatat tuodaan Unity:n puolelle ja liitetään työkaluun nimeltään laattapaletti (Tile Palette) (kuva 39). Nyt laattapaletti-ikkunassa voidaan luoda uusi paletti, johon tuodaan luodut laatat.



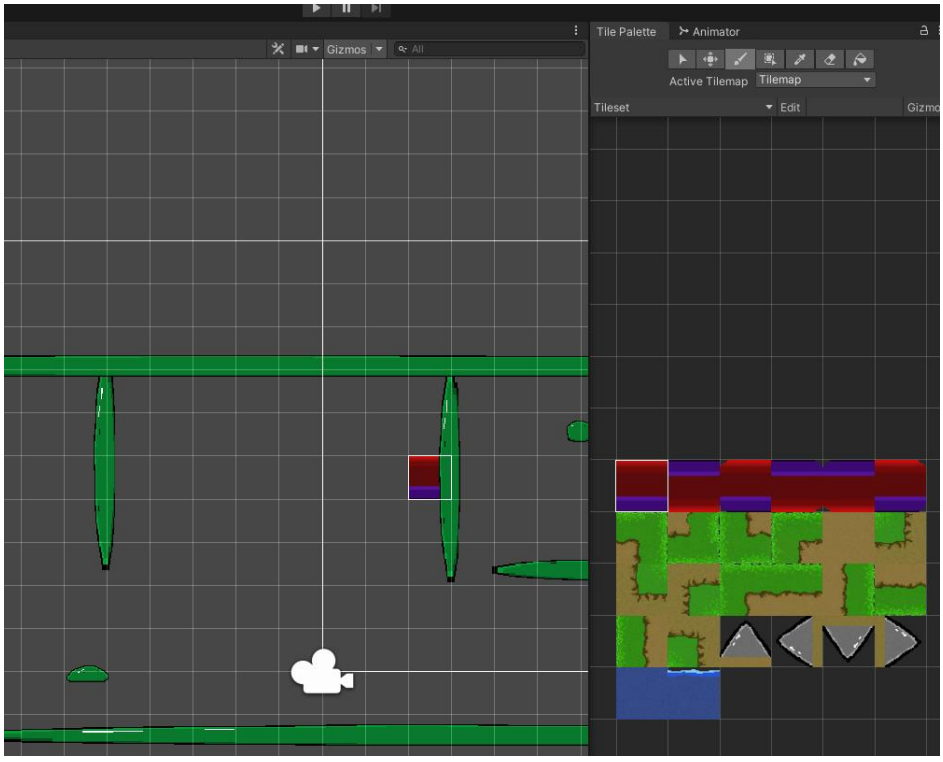
KUVA 39. Laatta palettityökalu.

Tason piirtämiseksi tarvitaan vielä käyttöön laattakartta (Tilemap), mihin voidaan asettaa laattoja haluamalla tavalla (kuva 40). Tämä laattakartta piirtää ruudukon näkymäikkunaan (Scene) kuvan 41 mukaisesti. Ruudukko tulee mitoittaa samankokoiseksi laattojen kanssa, jotta piirrettyyn kenttään ei jäisi rakoja tai laatat eivät menisi päällekkäin.

Laattojen piirtäminen toimii valitsemalla haluamasi laattaa, jonka jälkeen valitaan sivellin ja valitsemalla haluamasi ruutu ruudukosta. Laattoja ei voi kääntää, joten laattoja pitää luoda kaikkiin mahdollisiin tilanteisiin (esimerkiksi yksinkertaisesti kääntämällä ensimmäinen laatta ylösalaisin kattoa varten).



KUVA 40. Uusi 2D-objekti nimeltään laattakartta (Tilemap).

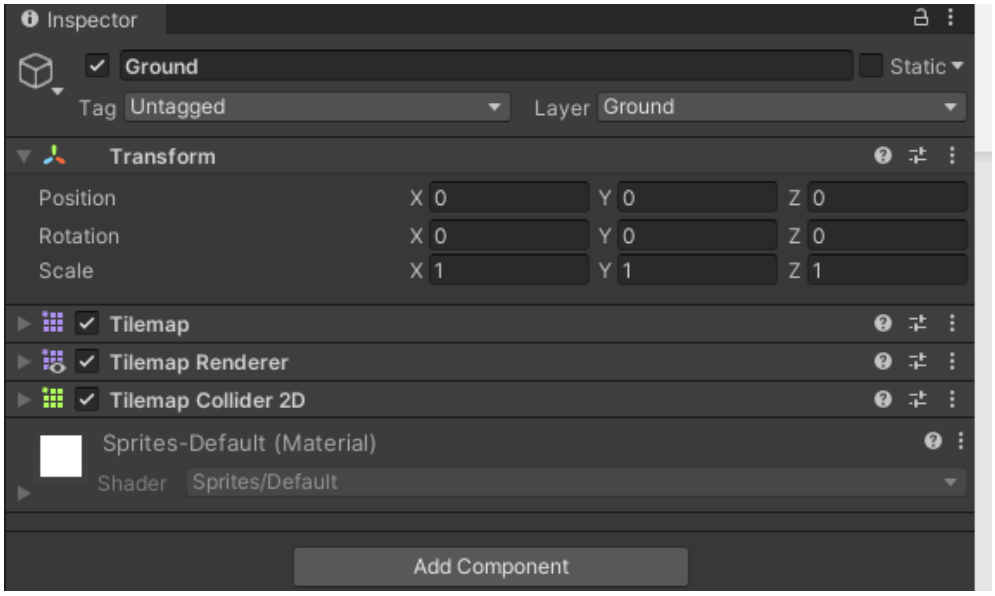


KUVA 41. Laattakartan ruudukko, sekä laatta paletti laattoineen.

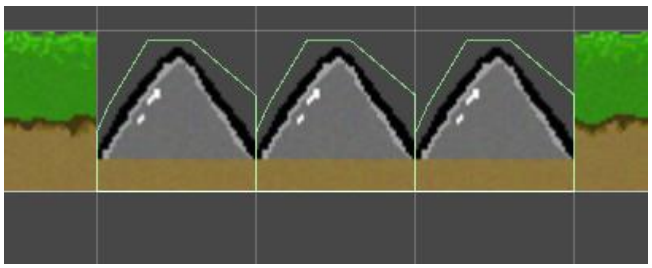
Laattakarttoja voi tehdä useampia (kuva 42), jotka helpottavat pelin ohjelmointia huomattavasti. Jokaiselle laattakartan voi merkata tai antaa niille kerrosmerkinnän. Piirtämisessä tulee kuitenkin muistaa, mikä laattakartta on juuri nyt valittuna. Väärällä laattakartalla voidaan muuttaa laattojen ominaisuuksia esimerkiksi piikit eivät vahingoita pelaajaa.

KUVA 42. Pelin maan ja piikkien laattakartat.

Ominaisuuksia voidaan lisätä laattakartoille ohjelmoimalla tai lisäämällä Unity:n erillisiä komponentteja objektille. Kuvassa 43 näkyy maa-laattakartan ominaisuudet. Maalle ei ole annettu mitään tagia, mutta kerrosmerkintä on annettu, jotta pelihahmo tunnistaa milloin se on kosketuksissa maan kanssa. Unity piirtää törmäyskoneen laatoille laattakartassa, lisäämällä laattakarttaan 2D laattakartan törmäyskoneen (”Tilemap Collider 2D) kuvan 45 mukaisesti.

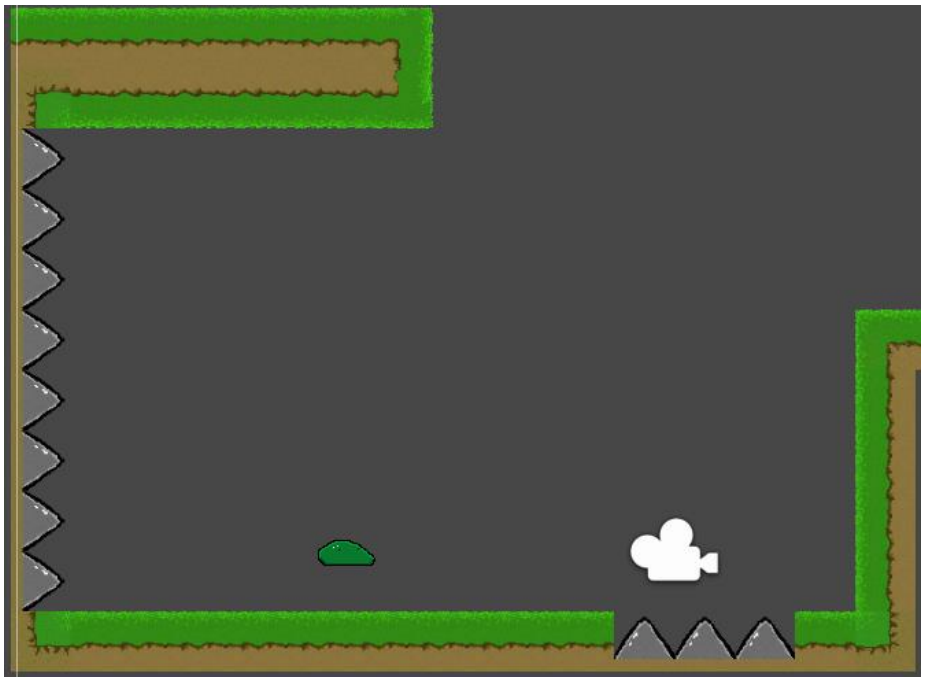


KUVA 43. Maalaattakartan tarkastelu ikkuna.



KUVA 44. Törmäyslaatikoiden automaattinen piirtäminen.

Pelin taiteella on suuri merkitys pelikokemuksen kanssa, joten jokaisen laatan tulisi kuvata itseään pelaajalle selkeästi. Kuvassa 45 nähdään pelin muutamia laattoja käytettynä, mistä saadaan heti selkeä kuva, missä on turvallista sekä missä on vaara.



KUVA 45. Luonnos mahdollisesta pelimaailmasta.

6 POHDINTA

Opinnäytetyön tavoitteena oli suunnitella ja kehittää toimiva 2D-videopeli Unity-pelimoottorilla, sekä antaa kiinnostuneille pelisuunnittelijoille ja pelikehittäjille ymmärtämys Unity-pelimoottorin käytöstä ja 2D-videopelien kehittämisestä. Työn tuloksena saatiin kehitettyä toimiva 2D-tasohyppelypeli toimivalla ohjattavuudella, helposti muokattavalla kenttäsuunnittelulla ja toimivalla ohjelmointikoodilla, jonka muuttujia voidaan helposti ohjata Unity:ssa.

Työssä onnistuttiin loistavasti, mutta ajallisen haasteen vuoksi kaikkia mahdollisia ideoita ei sisälletty 2D-videopelin ohjelmointiin. Ohjelmoinnin aikana ilmeni pelin kehittämisessä eniten haasteita, koska uuden ohjelmointikoodi osion lisäämisen jälkeen huomattiin videopelissä uusia ohjelmointivirheitä, jotka aiheuttivat joko pelihahmon liikkumisen kanssa ongelmia tai peli ei toiminut kuten se oli suunniteltu.

Tätä työtä on hyvä käyttää esimerkkinä aloittamaan pelisuunnittelu ja pelikehittäminen. Työssä selitetyt peliin rakennetut mekaniikat ovat helppo ymmärtää ja niitä on helppo muokata haluamallansa tavalla. Ohjelmointikoodissa on käytetty monia erilaisia tapoja, joka antaa lukijalle suuremman näkemyksen ohjelmoinnin mahdollisuuksista.

LÄHTEET

Brandon, A. 2005. Audio for Games: Planning, Process and Production. New Riders

Harvey S, 1999. Designing Enemies With Distinct Functions. Viitattu 7.6.2022 https://www.gamedeveloper.com/design/designing-enemies-with-distinct-functions

Huiberts, S, 2010. Captivating Sound. The Role of Audio for Immersion in Computer Games. Viitattu 15.3.2022 http://download.captivating-sound.com/Sander_Huiberts_CaptivatingSound.pdf

Maio A, 2020. What is Animation? Definiton and Types of Animation. Viitattu 1.6.2020 https://www.studiobinder.com/blog/what-is-animation-definition/

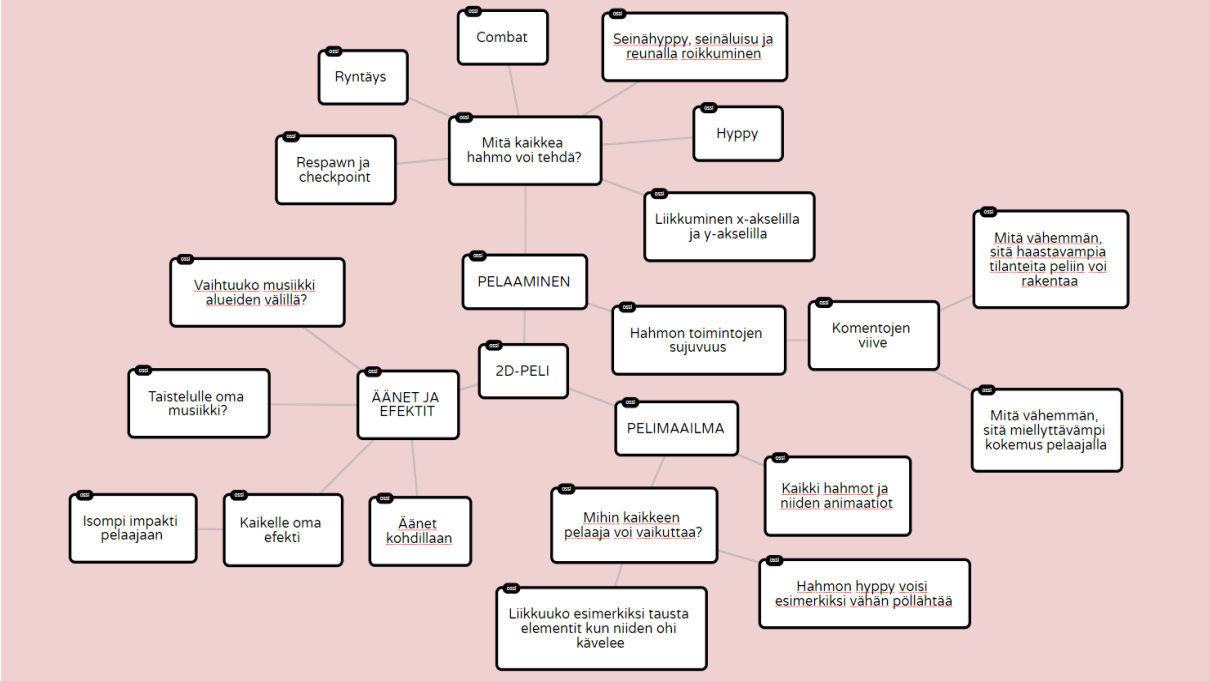
Sinicki A, 20.3.2021. What is Unity? Everything you need to know. Viitattu 1.6.2022 https://www.androidauthority.com/what-is-unity-1131558/

Techopedia, 2016. Gaming. Gameplay. Viitattu 1.6.2022 https://www.techopedia.com/definition/1911/gameplay

Unity. 2022, Our Company. Our impact by the numbers. Viitattu 1.6.2022 https://unity.com/our-company

LIITTEET

Liite 1. Ajatuskartta



LIITE 1. 2D-pelin ajatuskartta suunnittelua varten.