Pelinkehittäminen Godot avoimen lähdekoodin pelimoottorilla

Luong Tommy Tran

2024 Laurea

Laurea-ammattikorkeakoulu

Pelinkehittäminen Godot avoimen lähdekoodin pelimoottorilla

Luong Tommy Tran

Tietojenkäsittely

Opinnäytetyö

Tammikuu, 2024

Laurea-ammattikorkeakoulu **Tiivistelmä**

Tietojenkäsittelyn koulutus

Tradenomi (AMK)

Luong Tommy Tran

Pelinkehittäminen Godot avoimen lähdekoodin pelimoottorilla

Vuosi 2024 Sivumäärä 38

Tämän opinnäytetyön tarkoituksena oli kehittää julkaisu valmis avoimen lähdekoodin 2D peli Godot-pelimoottorilla. Pelin ohjelmointikielenä oli käytetty GDScriptiä. Pelinkehittämisen ohella opinnäytetyö ottaa kantaa Godot avoimen lähdekoodin pelimoottorin soveltuvuuteen nykyajan pelimoottorimarkkinoilla, joka oli tullut esiin Unityn 2023 syksyn hintamallimuutoksen myötä. Tärkeimpänä asian oli ollut kuitenkin opinnäytetyöntekijän kehittyminen pelinkehittämisessä ja projektinhallinnassa.

Työn teoriaosuudessa käydään läpi digitaalisia oikeuksia, tarkemmin ottaen vapaista- ja avoimen lähdekoodin ohjelmistoista, pelinkehittämisestä yleistasolla ja Godot-pelimoottorista. Kehittämistyönosiossa keskitytään kertomaan pelin toiminnallisuuksista Godot-pelimoottorin näkökulmasta.

Kehitettävä peli oli ensimmäinen graafinen peli, jonka opinnäytetyöntekijä kehitti. Kehitystyö oli kokenut suuria vaikeuksia kehityksen aikana, mikä oli johtunut osin puutteellisista tiedoista uusimmasta Godot-pelimoottorista. Opinnäytetyöntekijä oli tavoitellut liian kunnianhimoista peliä, mutta kehitystyön lopussa oli päädytty yksinkertaisempaan peliin. Godotin bugeista ja ongelmista huolimatta sopii se erinomaisesti pienille peliprojekteille, ja samalla saatiin selville, että avoimen lähdekoodin pelimoottorit voivat olla sopiva vaihtoehto kaupallisille pelimoottoreille.

Asiasanat: avoin lähdekoodi, GDScript, Godot-pelimoottori, pelinkehittäminen

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor’s thesis

Luong Tommy Tran

Game development with Godot open-source game-engine

Year 2024 Pages 38

The purpose of this Bachelor’s thesis was to develop a ready-to-publish open-source 2D game using the Godot game engine. The game was programmed using GDScript. In addition to game development, the thesis addresses the suitability of the Godot open-source game engine in today's game engine market, which became apparent following Unity's price model change in the fall of 2023. However, the focus is on the author's personal development in game development and project management.

The theoretical background of the thesis covered digital rights, specifically free and open-source software, game development at a general level, and the Godot game engine. In the development section, the focus was on discussing the features of the developed game specific to the Godot game engine in game development.

The developed game was the author's first graphical game. The development process was facing significant challenges, partly due to insufficient knowledge of the latest Godot game engine. The author aimed for a too ambitious game, but eventually settled for a simpler game towards the end of the development process. Despite the bugs and issues with Godot, it proved to be suitable for small-scale game projects, and it was discovered that open-source game engines could be a viable alternative to commercial game engines.

Keywords: game development, GDScript, Godot game engine, open source

Sisällys

1 Johdanto

Tässä opinnäytetyön kehittämistyössä on tavoitteena kehittää julkaisuvalmis videopeli PC-alustalle Godot-pelimoottoria käyttäen. Godot-pelimoottorissa käytettävät oletus ohjelmointikielet ovat GDScript ja C#, mutta laajennuksien avulla voidaan hyödyntää muitakin ohjelmointikieliä. Kehitettävä peli ohjelmoidaan GDScript-ohjelmointikielellä.

Opinnäytetyön päätavoite on kehittää osaamistani pelinkehittämisessä, projektinhallinnassa, ohjelmointitaitoja ja ottaa selvää, että onko Godot-pelimoottori sopiva työkalu pelinkehitykseen. Opinnäytetyöllä ei ole ulkoista toimeksiantajaa.

Opinnäytetyön teoriaosuudessa tutustutaan pelinkehittämisen aihealueeseen Godot-pelimoottoriin liittyen. Sitä koskeviin aihealueisiin kuuluu digitaalisten tuotteiden oikeudet, pelinkehittämisprosessi ja pelimoottorit.

2 Työn lähtökohdat

Videopelit ovat olleet aina kiinnostuksen kohteena minulle, ja halu tehdä oma peli on ollut jo monta vuotta, mutta niiden tekeminen on tuntunut ylitsepääsevän vaikealta. Sittemmin Laurea-ammattikorkeakoulun tarjoamien opintojen mukana olen saanut tuntumaa tietotekniikkaan ja siihen koskeviin työskentelymenetelmiin, joita hyödyntäen on pelinkehittäminen tuntunut todellisemmalta. Sen takia on opinnäytetyön aiheena kehittämistyö, jossa kehitetään peli Godot-pelimoottorilla. Aiempaa kokemusta pelinkehittämisestä ei minulla ole.

Godot valittiin pelimoottoriksi pelimaailmassa lähiaikoina tapahtuvien tapahtumien takia. Unityn hintamalli muutos julkaisu vuonna 2023 syksyllä on saanut aikaan kuohun pelinkehittäjien kesken. Unityn syksyllä julkaistu hintamalli muutos tulisi muuttamaan Unityn kaupallisten pelimoottorien tilausmallia siten, että tekijältä veloitetaan pelin lautaskertojen perusteella lisänä tavalliseen kuukausi- tai vuosimaksuun. Lautausperusteinen maksumalli on kyseenalaistettu pelinkehittäjien yhteisössä, sillä selkeyttä sille, mitä ”lataus” kattaa on epäselvä. On arvioitu, että Unityn hintamalli muutoksella voi käydä niin, että pelinkehittäminen ei olisi enää yhtä kannattavaa kuin ennen. Pelkoa on herännyt myös siitä, että latausperusteista hintamallia voitaisiin käyttää väärin, jolloin pahantekijät voisivat tahallaan tuottaa lisäkustannusta pelinkehittäjille.

Unityn sijaan on tuotu esiin vaihtoehtoisia pelimoottoreita, joista yksi niistä on Godot-pelimoottori. Godot-pelimoottori on avoimen lähdekoodin pelimoottori, kun taas Unity on kaupallinen pelimoottori. Pelimoottoreista kerrotaan myöhemmin teoriaosuudessa lisää.

2.1 Kehittämiskohteen kuvaus ja kehittämistavoitteet

Kehittämistyössä tavoitteena on kehittää julkaisuvalmis peli Godot-pelimoottorilla, joka tullaan julkaisemaan itch.io sivulle julkisesti kaikille ilmaiseksi ladattavaksi. Kehitettävä peli tulee olemaan 2D-pohjainen ja sen genrenä on 2D toiminta roguelite (eng. Action roguelite). Toisena tavoitteena on saada tuntumaa Godot-pelimoottorista ja sen perusteella pohtia, että kuinka hyvin se soveltuu 2D pelienkehittämiseen.

2.2 Aihealueen rajaus

Aiheena on pelinkehittäminen Godot-pelimoottorilla ja avoimen lähdekoodin pelimoottorien soveltaminen etenkin indie-kehittäjänä. Alkuperäisenä ideana oli valmistaa ”sama” peli kahdesti Unitylla ja Godotilla, ja julkaista molemmat maksullisina pelinä johonkin pelien jälleenmyyjäpalveluun, kuten Itch.io tai Steam verkkokauppoihin. Julkaisun jälkeen voitaisiin tutkia kahdella eri pelimoottorilla valmistettujen pelien kaupallistaminen, mutta se ei osoittaudu todennäköiseksi, sillä Unityn ilmaisversiolla tehtyjä pelejä saa myydä ilman lisenssiä tai lisämaksuja, kunnes tulot eivät ylitä $100,000.

Kehittämistyön raportoinnissa keskitytään Godotilla pelinkehittämisen erikoisuuksiin esittelemällä kehitystyön rakennetta. Raportoinnissa ei oteta huomioon pelikokemusta, ulkoasua tai vertailua muihin pelimoottoreihin.

2.3 Keskeiset käsitteet

AAA	AAA yritykset ovat suurimpia pelinkehitysyrityksiä, joissa työskentelee yleensä yli 100 henkilöä ja niiden kehittämät pelit ovat AAA pelejä. AAA yritykset voivat saada suurilta julkaisijoilta rahoitusta, taikka olla itse julkaisijoita. (Egenfeldt-Nielsen, Smith & Tosca 2016, 20-24.)
Avoin lähdekoodi	Avoimen lähdekoodin ohjelmisto on ohjelmisto, jota voi jakaa vapaasti, sen mukana tulee luettava lähdekoodi, voidaan muokata haluamallaan tavallaan, on puolueeton ja se sisältää jaettaessa samat lisenssit kuin jaettavalla ohjelmistolla (Open Source Initiative, 2023b).
Copyleft	Copyleft on rajaus, joka rajaa ohjelmiston siten, että ohjelmisto (alkuperäinen tai muokattu), joka jaetaan, tulee saamaan samat oikeudet jakamiseen ja muokkaamiseen (Stallman 2015, 184-186).

Indie	Indie yrityksen määritelmä videopelimarkkinoilla tarkoittaa pelinkehitysyritystä, jonka henkilöstö on pieni tai yksittäinen henkilö ja se ei saa ulkoista rahallista tukea suurilta julkaisijoilta (Kevuru Games 2023; Egenfeldt-Nielsen ym. 2016, 20-21).
Node	Nodet ovat Godot-pelimoottorissa tärkeitä rakennusosia, joista peli koostuu. Nodeja on eri tyyppejä, mutta niillä kaikilla on seuraavat ominaisuudet: nimi, muokattavat ominaisuudet, päivittyvät joka kuvataajuudella, niihin voi lisätä ominaisuuksia ja toimintoja, sekä olla toisten noden lapsia (eng. child) muodostaen puun (eng. tree). (Godot 2024a.)
Pelimoottori	Pelimoottori on kokoelma työkaluja ja teknologioita, jotka auttavat pelinteossa toteutusvaiheessa (Bradfieldt 2023).
Scene	Scenet koostuvat nodeista. Sceneillä on yksi päänode (eng. root node), joten yksittäinen node voi olla scene. Sceneillä on samat ominaisuudet kuin nodeilla, mutta lisäksi niitä voi tallentaa kovalevylle ja lataa uudelleen käyttöön. (Godot 2024a.)
Signal	Godotissa signaalit ovat viestejä, joita nodet lähettävät tietyn tapahtuman jälkeen. Muut nodet voivat yhdistyä (eng. connect) signaaleihin ja aktivoida jonkin metodin, kun signaali otetaan vastaan. (Godot 2024b.)
Vapaa Ohjelmisto	Vapaa ohjelmisto (eng. free software) on idea, aate ja ohjelmisto, jolla on GNU:n mukaan neljä ehdotonta vapautta. Lähes kaikki muut ohjelmistot, jotka eivät ole vapaita ohjelmistoja ovat epävapaita tai omisteisia ohjelmistoja. (GNU 2023.)

3 Vapaa ohjelmisto

Vapaa ohjelmisto (eng. free Software) on idea, ja samalla kuvaus ohjelmiston vapaudesta. Vapaan ohjelmiston idean, sekä lisenssin tarkoituksena on tuoda vapautta ja yhteistyöhalua yhteiskunta- ja yksilötasolla ohjelmistojen ulkopuolellakin. Vapaa ohjelmisto voi olla kaupallinen tai ei-kaupallinen, mutta koskaan se ei voi olla omisteinen ohjelmisto (eng. proprietary software). (Stallman 2015, 3-7.)

Kriteereinä vapaalle ohjelmistolle on 4 kappaletta Free Software Foundationin, eli FSF:n mukaan. FSF on yleishyödyllinen säätiö, joka perustettiin vuonna 1985, ja sen päätavoitteena on edistää vapaan ohjelmiston aatetta. Siihen voi liittyä kuka vain vapaan ohjelmiston aatteesta kiinnostuneille. FSF:n toiminta koostuu nykyään vapaa ohjelmistoprojektien kehityksestä, niihin käsittelevien manuaalien ja tukipalveluiden myymisestä, aatteen levittämisestä kansainvälisesti kampanjoiden kautta, ja vapaan ohjelmistojen lisenssien antajana. FSF saa suurimman osan pääomastaan lahjoituksista, vain murto-osa tulee myydyistä tuotteista ja palveluista. (Stallman 2015, 11-14.)

Neljät kriteerit sille, että ohjelmisto on vapaa ohjelmisto, ovat nollasta kolmeen vapaudet:

0. Vapaus: Vapaus käyttää ohjelmistoa tahtomallaan tavalla, mihin vain tarkoitukseen.
1. Vapaus: Vapaus opiskella ohjelmiston toimintoa, ja muokata sitä tahtomallaan tavalla. Pääsy lähdekoodiin on edellytyksenä tälle vapaudelle.
2. Vapaus: Vapaus jakaa kopioita, jotta voit auttaa naapureita.
3. Vapaus: Vapaus jakaa sinun muokattuja kopioitasi muille.

Jos ohjelmisto ei täytä näitä edellyttäviä vapauksia, on ohjelma siten epävapaa. Vapaudet 2 ja 3 koskettavat vapautta jakaa alkuperäistä ja muokattua kopiota muille ilmaiseksi tai maksullisena pyytämättä keneltäkään lupaa. Kellekään ei tarvitse myöskään ilmoittaa, että alkuperäistä ohjelmistoa on muokattu, ja että tällä muokatulla ohjelmistolla on myös nämä 4 vapautta. Jaettua kopiota voidaan muokata ja siirtää toiseen lisenssiin, joka ei ole vapaan ohjelmiston lisenssi tai lisätä rajoituksia, tehden uudesta muokatusta kopiosta epävapaan. (Stallman 3-7.)

On eräitä rajauksia, joita on sallittu lisätä lisenssiin vapaalle ohjelmistolle. Yksi näistä rajauksista on copyleft. Copyleft on rajaus, joka rajaa ohjelmiston siten, että ohjelmisto (alkuperäinen tai muokattu), joka jaetaan, tulee saamaan samat oikeudet jakamiseen ja muokkaamiseen. Yksinkertaisesti selitettynä copyleft takaa sen, että vapaa ohjelmisto pysyy vapaana ohjelmistona, koska jaetulla ohjelmistolla on samat oikeudet kuin jaettavalla vapaa ohjelmistolla. GNU General Public License (GNU GPL) sisältää copyleftin. (Stallman 2015, 184-186.)

Vapaus 0 ja 1 antavat käyttäjälle täyden hallinnan ohjelmiston omaan käsittelyyn. Jotta ohjelmistoa voi käyttää miten tahtoo, täytyy sen edellytyksenä olla se, että ohjelmiston mukana tulee ohjelmiston lähdekoodi. Lähdekoodi pitää olla 1. vapauden mukaan ”luettavissa”, eli binaarikielellä ja hämärtävillä keinoilla piilotetut lähdekoodit (eng. obfuscated) eivät ole kelvollisia, mutta yleisillä ohjelmointikielillä kirjoitetut lähdekoodit ovat kelvollisia. (Stallman 2015, 3-7.)

Vapaan ohjelmiston lisenssejä ovat GNU General Public License (GPL) versiot 2 ja 3, GNU Lesser General Public License (LGPL) versiot 2.1 ja 3 ja monta muuta lisenssiä. Ohjelmistolla ei

tarvitse olla vapaan ohjelmiston lisenssiä ollakseen vapaa ohjelmisto. Yleisenä nyrkkisääntönä vapaan ohjelmiston lisenssille on copyleft-rajoitus ja se, että onko lisenssi yhteensopiva GNU GPL:n kanssa. Eräs tunnettu vapaan ohjelmiston lisenssi, joka ei ole yhteensopiva GNU GPL:n kanssa on Common Public License ja Open Software License. (GNU, 2023.)

Ohjelmistoihin kuuluvat videopelit, joten vapaan ohjelmiston videopelit ovat vapaita, eli niillä on samat neljät vapaudet kuin muillakin vapailla ohjelmistoilla. Vapaan ohjelmiston pelejä ja ohjelmistoja ei kannata sekoittaa avoimen lähdekoodin peleihin ja ohjelmistoihin, vaikka näillä on paljon yhteistä keskenään. Vapaan ohjelmiston pelejä voi löytää hyvin arkistoiduissa vapaiden ohjelmistojen nettisivuista, kuten LibreGameWiki-sivussa. (Stallman 2015, 236-237.)

3.1 Avoin lähdekoodi

Avoin lähdekoodi on idea ja käytäntö, joka ajaa samaa periaatetta vapaan ohjelmiston kanssa. Vuonna 1998 jako vapaaseen ohjelmistoon ja avoimeen lähdekoodiin tapahtui vapaa ohjelmisto yhteisön erimielisyyksien takia. Avoin lähdekoodi on tavoitepitoisempi kuin vapaa ohjelmisto, joka on enemmänkin filosofinen ja poliittinen aate. Sen myötä syntyi FSF:n rinnalle Open Source Initiative (OSI) järjestö, joka ajaa avoimen lähdekoodin ideaa. (Open Source Initiative, 2006.)

OSI on voittoa tavoittelematon järjestö, joten sekin saa suurimman osan varoistaan lahjoituksista FSF-järjestön tavoin. Sen päätavoitteena on edistää avoin lähdekoodin menetelmää ja sen yleisyyttä maailmalla. OSI lupaa laadukkaampaa, kestävämpää, edullisempaa ja joustavampaa ohjelmistokehitystä yrityksille ja yksilöille, jotka käyttävät avointa lähdekoodin menetelmää. OSI antaa FSF-järjestöjen tapaan neuvontaa osapuolille avoimen lähdekoodin menetelmästä ja toimii standardisointielimenä avoimen lähdekoodin käsiteelle, ja lisensseille. (Open Source Initiative, 2023a.)

Jokaisessa vapaassa ohjelmistossa on avoimen lähdekoodin menetelmä käytössä, mutta jokainen avoimen lähdekoodin ohjelmisto ei ole ”vapaa” Stallmanin (2015) mukaan. Kyseessä on ohjelmiston periaate ja lisenssi. Avoimen lähdekoodin ohjelmistoissa ei välttämättä ole copyleftiä, jolloin niiden vapaus ei siirry kopioihin tai muokattuihin kopioihin. Sama käytänne koskee pelejä, ja pelimoottoreita. Godot on avoimen lähdekoodin pelimoottori (Godot 2023b)!

3.2 Omisteinen ohjelmisto

Omisteinen ohjelmisto (eng. proprietary software) on ohjelmisto, joka ei ole avoin lähdekoodin- tai vapaa ohjelmisto. Sanan ja määritelmän ”omisteinen ohjelmisto” keksi FSF. Omisteinen ohjelmisto on oletusmuoto ohjelmistoille, avoin lähdekoodi- ja vapaa ohjelmisto ovat

poikkeus. Omisteinen ohjelmisto lakkaa olemasta omisteinen ohjelmisto, kun sen lisenssi muutetaan OSI-lisenssiksi tai GPL-lisenssiksi. (Stallman 2015, 243-244.)

4 Pelinkehitys

Videopelit ovat ohjelmistoja, joten pelikehitys on ohjelmistokehitystä. Pelille ei ole yhtä selkeää määritelmää, mutta monet ehdotetut määritelmät ovat samaa mieltä siitä, että pelin tarkoitus on turhan vaivan tekemistä asetettuja sääntöjä noudatellen. Ohjelmistoilla on yleensä jokin käyttötarkoitus, joka erottaa ne videopeleistä. Muuten ovat ohjelmistot ja videopelit samankaltaiset, joten niiden kehittämiseen käytetään samoja menetelmiä. (Bond 2023, 63-73.)

Pelienkehittämiseen on yleisesti jaettu kolmeen eri vaiheeseen. Ensimmäisenä vaiheena on esituotanto, toisena tuotanto ja viimeisenä kolmantena vaiheena jälkituotanto. Konseptointi voisi olla myös ensimmäisenä vaiheena pelinkehitystä, tehden pelinkehityksestä nelivaiheisen, mutta yleisesti se on osana esituotantoa. (Bond 2023, 24-26; Bustamante & Cohen 2010, 92-98; Stefyn 2022.). Aikataulutukseen pelinkehityksessä käytetään ohjelmistotuotannossa olevia ohjelmistotuotantoprosesseja (Bustamante & Cohen 2010, 120-124).

Oli pelinkehittäjätiimi yksittäinen kehittäjä, pieni indie-yritys, suuri AAA-yritys tai jokin näiden välistä, voidaan soveltaa edellä mainittuja kolmea vaihetta pelinkehitykseen. Indie yrityksen määritelmä videopelimarkkinoilla tarkoittaa pelinkehitysyritystä, jonka henkilöstö on pieni tai yksittäinen henkilö ja se ei saa ulkoista rahallista tukea suurilta julkaisijoilta (Kevuru Games 2023; Egenfeldt-Nielsen ym. 2016, 20-21.). Opinnäytetyössä toimii täten indie-kehittäjä ja kehitettävä peli on indie-peli, sillä pelinkehitys tapahtuu yksittäisen henkilön toimesta ilman rahallista tukea suurilta julkaisijoilta. AAA yritykset ovat suurimpia pelinkehitysyrityksiä, joissa työskentelee yleensä yli 100 henkilöä ja niiden kehittämät pelit ovat AAA-pelejä. AAA-yritykset voivat saada suurilta julkaisijoilta rahoitusta, taikka olla itse julkaisijoita. Indie-yrityksien ja AAA-yrityksien rinnalla on noussut 2000-luvulla suuren suosioon mobiilipelikehitys yritykset, jotka eivät sovi samaan muottiin kuin edellä mainitut kaksi, sillä niiden pelinkehitystuotanto eroaa laajalti perinteisistä peleistä. Mobiilipeliyrityksien tekemät liikevaihdot voivat vastata indie-yrityksien tai AAA-yrityksien liikevaihtoja, tai jopa ylittää ne pienemmällä kustannuksella. (Egenfeldt-Nielsen ym. 2016, 20-24.)

4.1 Esituotanto

Pelejä on miellyttävä pelata, mutta kaikki pelit eivät ole kaikkien makuun. Jo ennen esituotantoa pitää konseptoida pelin idea. Pelinkehityksessä on tavoitteena yleensä voiton tuottaminen yritykselle, mutta etenkin yksityisenä indie-kehittäjä voi olla tavoitteena muitakin kuin rahallinen voitto, kuten omien pelinkehitystaitojen harjaantuminen, jonkin yhteiskunnallisen

vaikutuksen luominen tai ilon tuottaminen pelaajille (Bond 2023, 197-206.). Esimerkkinä voittoa tavoittelemattomista peleistä ovat jotkut vapaa ohjelmiston- ja avoimet lähdekoodin pelit. Vaikka pelin kopioita jaettaisiin ilmaiseksi, voidaan silti siitä tuottaa voittoa lahjoituksien tai muiden keinojen avulla. Pelin konseptissa tulee esittää pelin tavoitteet, laajuus (budjetti ja aikataulutus), henkilöstön määrä ja sijoittaminen, idea, kohderyhmä, kannattavuus, laitteisto, julkaisualusta, ynnä muuta (Bustamante & Cohen 2010, 92-100; Egenfeldt-Nielsen ym. 2016, 20-24; Kevuru Games 2023.)

Konseptin hyväksynnästä edetään varsinaiseen esituotantoon, jossa laaditaan Game Design Document (GDD), johon kirjataan koko pelinkehitysprosessin aikana pelinkonsepti ja siinä esitetyt asiat yksityiskohtaisemmin. GDD toimii pelinkehityksessä elävänä manuaalina, jota seuraamalla saadaan pelinkehitys sujuvaksi ja se voidaan esittää potentiaalisille investoijille investoimaan pelinkehitystä. GDD:n tärkeys vähenee pienemmissä yrityksissä ja joissain käytetyissä tuotantoprosesseissa. Pienemmissä tiimeissä on helpompaa kommunikoida keskenään, jolloin pelinkehitykseen voi vaikuttaa suoremmin kuin suurissa tiimeissä, ja muutokset voidaan hyväksyttää nopeammin. Sama asia koskee ketteriä menetelmiä, joissa on nopeat kehityssyklit, mitkä tuovat elävämmän kehitysprosessin kuin vankka GDD. (Bustamante & Cohen 2010, 95-97; Kevuru Games 2023.)

Prototyyppi luodaan kehitettävästä pelistä esituotannossa, jotta saadaan selville pelintuntuma käytännössä esiin. Prototyypistä voidaan päätellä pelin kannattavuus. Kannattavuus tulkitaan pelin miellyttävyydestä pelata ja siitä mahdollisesti tulevista kuluista. Jos peli ei ole miellyttävä pelata tai se tulee ylittämään budjetin, voidaan pelin GDD muokata, kunnes pelistä tulee miellyttävä pelata, ja sen budjetti on kohdallaan. Prototyyppi on todella karkea kuvaa antava teos lopullisesta pelistä, siitä puuttuu suurin osa viimeisistä graafisista resursseista, ja sen pituus on vain murto-osa lopputeoksesta. (Egenfeldt-Nielsen ym. 2016, 20-24; Kevuru Games 2023.)

4.2 Tuotanto

Suurimmassa osassa yksinpeleistä menee aikaa eniten pelinkehityksessä tuotantovaiheessa. Tuotanto vaiheessa ohjelmoidaan peli, luodaan visuaaliset- ja ääniominaisuudet, kehitetään pelin ideaa (juoni, miljöö, käsikirjoitus, jne.) ja tehdään markkinointia. Tuotannossa on muutamia tavoitekohtia, joita tullaan tekemään, jotta tuotanto voisi siirtyä seuraavaan tavoitekohtaan. Nämä tavoitekohdat (eng. milestones, kts. kuvio 1) ovat prototyyppi (jo esituotannossa tehty), ensimmäinen pelattava (eng. first playable), vertical slice, pre-alpha, alpha, beta ja gold master. (Egenfeldt-Nielsen ym. 2016, 24-26; Kevuru Games 2023.)

Kuvio 1: Pelinkehityksen tuotantovaiheen tavoitekohdat

Prototyyppi on jo esitetty kappaleessa 4.1 esituotanto. Ensimmäinen pelattava on prototyyppi, joka on kuin ensimmäinen prototyyppi, mutta siihen on panostettu enemmän, yleensä lisäämällä graafisia ominaisuuksia ja ääniresursseja. Vertical slice-prototyyppi esittelee pelin kokonaisuudessaan tietyssä rajatussa kohdassa koko pelistä, eli sen tarkoitus on näyttää millainen valmis peli tulisi olemaan. Vertical slice-prototyypissä on pelilliset, visuaaliset ja ääniominaisuudet valmiina rajatussa kohdassa koko peliä. Pre-alpha jalostaa vertical slice-prototyyppiä koko pelin mittaiseksi. Alpha on ”valmis” peli, siinä on kaikki pelin ydinominaisuudet valmiina ja pelin pystyy pelaamaan alusta loppuun. (Bustamante & Cohen 2010, 98-100; Egenfeldt-Nielsen ym. 2016, 24-26; Kevuru Games 2023.)

Beta-versiossa on alpha versiosta hiottu suurin osa testeistä paljastuneet bugit ja toimivuudet. Ennen beta-versiota on kaikki aiemmat prototyypit testattu yrityksen sisällä, mutta beta-versiossa annetaan peli myös ulkoisille osapuolille testattavaksi. Ulkopuoliset testaajat voivat olla kohderyhmä, joille peli tullaan pääosin mainostamaan. Näitä ulkopuolisia testaajia kutsutaan beta-testaajiksi, ja on yleistä, että he eivät saa palkkaa, mutta sen sijaan muita etuja, kuten ilmaisen kopion valmiista pelistä julkaisuaikana. Beta-testaajat voivat myös toimia pelin suurlähettiläinä (eng. ambassador), levittäen sanaa pelistä, tuoden pelille ilmaisesti julkisuutta. (Bustamante & Cohen 2010, 98-100; Egenfeldt-Nielsen ym. 2016, 24-26; Kevuru Games 2023.)

Beta-testeistä saaduista tuloksista parannellaan peli Gold Master-versio (GM, tunnetaan myös nimellä GMC eli Gold Master Candidate)(Bustamante & Cohen 2010, 98-100). GM-versio pelistä on täysin valmis, joten sen voi esitellä julkaisijoille hyväksyttäväksi julkaisuun julkaisualustoille. GM-versio pelistä sisältää vain kaikki tarvittavat ominaisuudet pelin toimimiseen sille

tarkoitetulle alustalle, eli lähdekoodia ja muita materiaaleja ei ole GM-versiossa. Kun GM on hyväksytty ja peli julkaistu, siirrytään pelinkehityksessä viimeiseen vaiheeseen, eli jälkituotantoon. (Egenfeldt-Nielsen ym. 2016, 24-26; Kevuru Games 2023.)

4.3 Jälkituotanto

Jälkituotannon pituus vaihtelee tuotetun pelin perusteella. Jotkut pelit vaativat päivityksiä ja lisäominaisuuksia julkaisun jälkeen, jolloin voidaan pelin jälkituotantoa jatkaa, kunnes pelin tuki päätetään tai peli koetaan valmiiksi. Jälkituotanto voi olla pelin tuottavin vaihe, sillä pelissä voi olla sisäisiä mikromaksuja, ja peliin voidaan tehdä maksullisia lisäosia (eng. expansion) tai ladattavia sisältöjä (eng. DLC). Kuitenkin jokaisen julkaisun jälkeen kuuluu tehdä jälkipuinti, jossa kehitystiimi tekee yleiskatsauksen kehityksestä. Parannusideoita ja tulevaisuuden suunnitelmia tullaan keskustelemaan jälkipuinnissa. Kehitystiimi voi siirtyä seuraavaan projektiin, jatkaa pelin tukea, tehdä jatko-osan pelille, tai hajota tiiminä. (Bustamante & Cohen 2010, 98-100; Egenfeldt-Nielsen ym. 2016, 24-26.)

5 Godot-pelimoottori

Pelinkehittämisen alkuun tarvitaan konsepti, jotta voidaan siirtyä esituotantoon. Konseptissa ilmenee peliin kuuluvat ”ideaaliset” asiat, jotka erottavat pelin muista peleistä, mutta siinä on konkreettisiakin asioita listattuna, kuten ohjelmointiteknologia, pelialusta, sekä pelimoottori. (Egenfeldt-Nielsen ym. 2016, 24-26; Kevuru Games 2023-)

Pelimoottori on kokoelma työkaluja ja teknologioita, jotka auttavat pelinteossa toteutus vaiheessa. Pelimoottorin avulla säästetään aikaa, sekä osaamista, sillä siinä on valmiina toimintoja ja ratkaisuja, jotka voivat olla erittäin hankalia toteuttaa kehittäjille. Pelimoottorissa olevat ominaisuudet, jotka ovat erittäin vaikeita kehittäjille itse toteuttaa ovat pääasiassa: renderöinti, fysiikkamoottori, pelialustatuki ja yleinen kehitysympäristö. Renderöinti on prosessi, jossa peliä näytetään pelaajan näyttöön. Pelimoottorin renderöinti soveltaa pelin näyttämistä eri laitteistoille (eng. hardware), joka on aikaa vievää toteuttaa itse ohjelmoimalla. Fysiikkamoottori vaikuttaa pelissä tapahtuvien objektien simuloimiseen. Fysiikkamoottori on todella vaikea tehdä itse. Pelimoottoreissa on helppoa tuoda (eng. export) valmistettava peli eri pelialustoille, ilman, että pelin koodia pitäisi kirjoittaa uudestaan jokaiselle eri pelialustalle. Pelimoottoria käyttäessä kehittyy myös osaaminen sen hallintaan, jota voi käyttää muissa pelinkehitys projekteissa. (Bradfield 2023, 3-4.)

Valmiit pelimoottorit soveltuvat erinomaisesti indie-pelinkehittäjille, sillä näillä voi puuttua syvä ohjelmointitaito, jolloin voivat ne keskittyä enemmän pelinkehittämiseen kuin ohjelmointiin. Isot yritykset pystyvät luomaan oman pelimoottorin omiin tarpeisiinsa, mutta niillekin on yleensä kannattavaa käyttää valmiita pelimoottoreita: Unity, Unreal Engine,

GameMaker Studio, jne. Monet pelimoottorit ovat kaupallisia tuotteita, ja niissä vaaditaan yleensä lisenssirajoituksia sekä rojaltimaksuja. Pelimoottorien hintamallistot voivat vaihdella minä hetkenä hyvänsä, joten voi käydä niin, että kehitteillä oleva peli ei olekaan yhtä kannattavaa kuin entisen hintamallin kanssa. Kaupallisten pelimoottorien rinnalle on sen sijaan lähiaikoina tuotu esiin ei-kaupalliset- ja avoimen lähdekoodin pelimoottorit, kuten Godot-pelimoottori. (Bradfield 2023, 3-4.)

Godot-pelimoottori on täysin ilmainen, rojaltivapaa, sen lähdekoodi on julkisesti esillä, ja tulee pysymään sellaisena, sillä Godot-pelimoottorilla on MIT-lisenssi (Godot 2023a). MIT-lisenssin takia voidaan Godot-pelimoottorista tehdä omia versioita ja jakaa niitä eteenpäin ilman kysymättä keneltäkään lupia, mutta näin harvoin toteutuu. Godot-pelimoottoria kehittää Godot yhteisö, jotka ovat kaikki vapaaehtoisia osallistujia. Tämä yhteisö parantelee pääosin yhtä virallista versiota Godotista, jonka pääversioita julkaistaan tietyin välein. (Godot 2023b.)

Godot-pelimoottorin pääpiirteisiin kuuluu ”block”-perusteinen ohjelmointi, avoin lähdekoodi yhteisöllisyys ja koodattavat objektit, joita kutsutaan sanalla resurssi (eng. resource). Avoin lähdekoodi mahdollistaa Godot-pelimoottorin muokkaamisen omiin tarpeisiin, ja yhteisöltä voi saada apua ja valmiita komponentteja, joita voi lisätä suoraan omaan Godot-pelimoottoriin, toisinkuin kaupallisissa pelimoottoreissa. Godot-pelimoottori käyttää oletusohjelmointikielenään GDScriptiä, joka muistuttaa suurimmaksi osaksi Python-ohjelmointikieltä. C# on myös valmiina soveltavaksi Godot-pelimoottoriin, mutta muitakin ohjelmointikieliä on mahdollista käyttää, ja näihin on tukikirjastoja tehty yhteisön toimesta ”language binding”-teknologiaa käyttäen. Language binding on teknologia, jonka avulla toista ohjelmointikieli käännetään haluttuun ohjelmointikieleen, jotta se toimii kuin halutun ohjelmointikielen. Godot-pelimoottori sopii niin 3D- kuin 2D pelienkehittämiseen, sekä se tukee pelin kehitystä ja julkaisua tietokone-, mobiili- ja pelikonsolialustoille. (Godot 2023b; Bradfield 2023, 3-4 & 225-228.)

6 Pelinkehittämisen toteutus

Peli kehitettiin yksin, joten opinnäytetyöntekijällä oli täysivalta koko pelinkehittämisen ajan. Pelinkehittäminen oli jaettu kolmeen eri vaiheeseen, esituotanto, tuotanto ja jälkituotanto. Esituotanto tapahtui jo opinnäytesuunnitelmaa tehtäessä. Esituotantoon kuuluu konseptointi ja yleisesti GDD (Game Design Document) luominen, mutta GDD nähtiin liian raskaalta toimelta yksintyöhön, joten se jätettiin väliin. Tässä vaiheessa tutustuttiin myös Godot-pelimoottorin käyttämiseen ja se tapahtui Godotin virallisella sivulla olevan tutoriaalin suorittamisella. Siinä opastettiin Godotin perustoiminnot, kuten graafisen käyttöliittymään tutustuminen, node ja scene-rakenteet ja ohjelmointi GDScriptillä peliprojektia tehden. Virallisen Godot tutoriaalin suoritettua kokeiltiin myös muita Godot tutoriaaleja, joita Godot yhteisön jäsenet ovat valmistaneet.

Kun tuntui siltä, että Godot oli tullut tarpeeksi tutuksi, siirryttiin tuotantoon. Esituotannon ja tuotannon aikana ei edetty suunnitelmallisesti. Sitten kun päähän tuli idea, niin sitä mentiin heti toteuttamaan. Tuotannon aikana pelinkonsepti, joka piti olla lyöty lukkoon, koki suuria muutoksia. Jälkituotantoon se ei tosin vaikuttanut. Jälkituotannossa tehtiin pelistä markkinaversio, jossa on vain pelin pelaamiseen tarvittavat asiat, eli tässä tapauksessa exe-tiedosto. Pelin markkinaversio julkaistiin itch.io sivulle ilmaisena pelinä, eli siitä ei voi maksaa, vaikka niin toivoisikin (itch.io sivulla on ”päätä itse hinta” vaihtoehto). Peliä saatetaan päivittää tulevaisuudessa, mutta todennäköisemmin siirrytään mieluummin seuraavan pelin kehittämiseen.

6.1 Pelinkehittämismenetelmä

Pelinkehittämiseen käytettiin koodaa ja korjaa-mallia (eng. code and fix model). Koodaa ja korjaa-malli on luontaisin ohjelmoinnin vaihejakomalli, jota jokainen koodari soveltaa koodatessaan tiedostamatta. Koodaa ja korjaa-mallia käyttäviä kehittäjiä kutsutaan ”cowboy” koodareiksi, sillä näiden työskentelyllä ei ole tarkkaa ohjausta muilta tahoilta, vaan he pääsevät suoraan kehittämään työtä vähäisellä alustuksella. Tässä menetelmässä on kaksi vaihetta: koodaus ja korjaus. Koodaus vaiheessa kehitellään sovellusta itsenäisesti kehitystiimissä, kunnes koetaan ongelmia koodissa, jolloin alkaa korjaus vaihe. Korjaus vaiheessa koodin ongelmia, eli bugeja koitetaan korjata. Kun ongelmat ovat saatu korjattua, palataan takaisin koodaus vaiheeseen. Tätä sykliä jatketaan, kunnes ohjelma on valmis. (Mkrtchyan 2019.)

6.2 Pelin konsepti

Pelin konseptina oli kehittää ilmainen avoimen lähdekoodin peli PC alustalle Windows-käyttöjärjestelmälle Godot-pelimoottorilla. Pelinkehittämisen aikaan uusin versio Godotista (24.12.2023) oli Godot v4.2.1. Versionhallintaan käytettiin Git-versionhallinta systeemiä. Git kautta pidetään myös Github repository päivitettynä, jotta peli olisi avoimen lähdekoodin peli. Godot projekteja on helppo jakaa ja muokata. Godot projekteja voidaan suoraan tuoda lähdekoodista ja tehdä niistä pelaamis- tai editointivalmiita, joten tässä projektissa kehitettyä peliä on helppo jatkokehittää muiden toimesta.

Pelin oli tarkoitus olla alun perin ”2D ylhäältäpäin kuvattu toiminta roguelite” (eng. 2D top down action roguelite), mutta se muuttui peliä kehittäessä” vampire survivor like”- genreksi, joka on toisaalta nimeltään ”Bullet Heaven”. Genren siirtymä ei ollut hetken mielenjohde, vaan pitkäaikainen pohdinta. Roguelite-genre on tunnusomaisesti satunnaismuuttajaan pohjautuva genre, mutta sen tarkat vaatimukset ovat puutteelliset. Pelinkehityksen aikana on tullut monestikin pohdittua, että mihin pitäisi lisätä satunnaismuuttujia, jotta peli olisi roguelite. Mitä enemmän alkoi pelin perusrunko valmistua, sitä enemmän ymmärrettiin, että tarvittavien satunnaismuuttujien tekeminen olisi liian aikaa vievää ja hankalaa ensimmäiseksi peliprojektiksi, joten pelin konseptia piti muuttaa yksinkertaisemmaksi. Pelinkehityksen

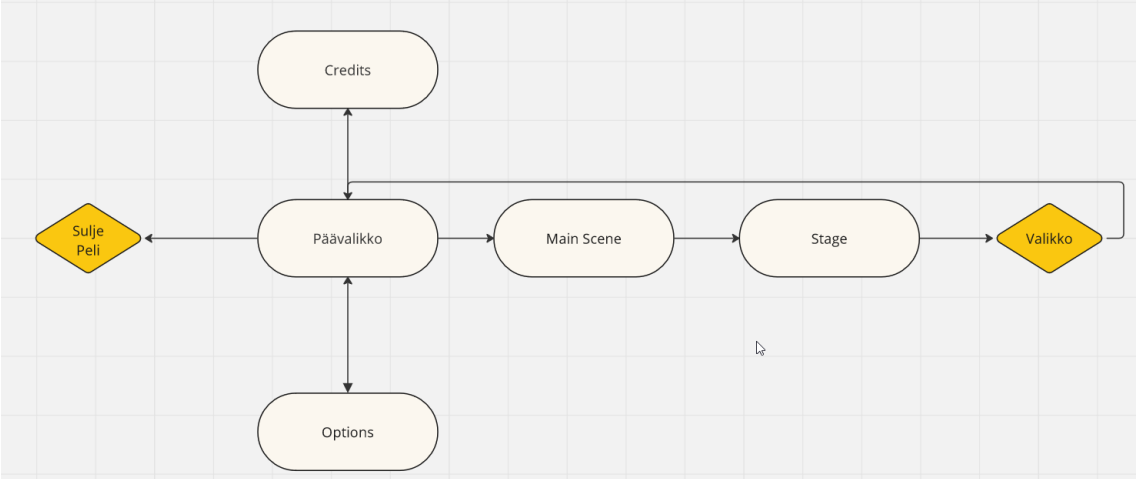
aikana saatiin inspiraatio sen hetken uudesta suosioon tullusta genrestä, joka oli ”Bullet Heaven”. ”Bullet Heaven”-tyyppiset pelit käyttävät hyväkseen myös satunnaismuuttujoita, mutta omaavat yleensä vähemmän ominaisuuksia kuin roguelite-pelit, jolloin sellainen olisi helpompi kehittää.

Pelin vanhana konseptina oli taistella ylhäältäpäin kuvaten huoneissa vihollisia, siirtyä seuraavaan huoneeseen ja toistaa tätä, kunnes pelaajan hahmo menehtyy. Huoneet ja viholliset olisivat satunnaismuuttujilla sekoitettu siten, että jokainen huone tuntuisi erilaiselta. Kun pelaaja läpäisisi huoneen, eli tuhoaisi kaikki viholliset huoneesta, saisi hän valita kolmesta satunnaisesta parannuksesta. Huoneissa olisi myös aseita satunnaisesti lojumassa, jotta pelaaja kykenisi vaihtamaan pelityyliään. Pelihahmon menehdyttyä näytetään pelaajalle piste, joka koostuu huoneiden määristä. Idea oli mainio, mutta toteutus osoittautui erittäin hankalaksi jo pelaajan hyökkäystapaa kehittäessä, eli lähes alussa. Vihollisten ja huoneiden satunnaistuottaminen olivat selkeästi mahdottomia aikataulua nähden, joka oli tuolloin yksi kuukausi, joten pelin konsepti tarvitsi muutoksen. Tuona aikana alkoi idea toiseen genreen siirtymisestä, joka tuli olemaan ”Bullet Heaven”.

Uuden genre mukana piti luoda uusi peli konsepti. Samat tekniset seikat pidettiin, kuten avoin lähdekoodi, Windows-järjestelmä ja Godot-pelimoottori, mutta pelin kulkua muutettiin. Sen sijaan, että luotaisiin satunnaisia huoneita ja vihollisia, olisi vain yksi vakituinen huone, jossa syntyisi vihollisia satunnaisiin paikkoihin näyttöä. Hyökkäämiseen ei käytetä pelaajan syöttämiä komentoja, vaan hyökkäykset tapahtuisivat itsestään. Vihollisilla olisi yksinkertainen ohjelmointikoodi, ne liikkuisivat pelkästään pelaajahahmoa kohti ja tuottaisivat vahinkoa koskettamalla pelaajahahmoon. Vihollisten menehdyttyä syntyisi niiden päälle tavara, jota ottamalla saisi pelaaja kokemuspisteitä ja tarpeeksi kokemuspisteitä kerättyä saisi pelaaja valita kolmesta satunnaisesta parannuksesta, joihin kuuluvat aseet. Peli päättyy, kun pelaaja päihittää ”pomo”-vihollisen.

6.3 Pelilogiikka ja scene-rakenne

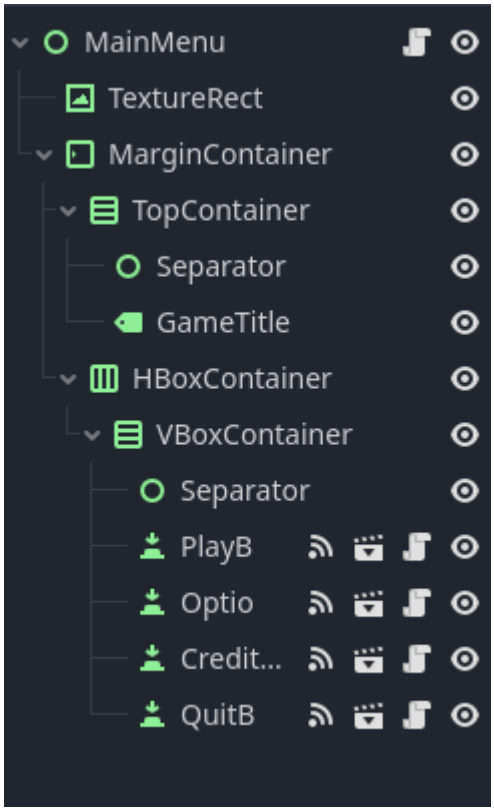
Godot-pelien rakenteeseen kuuluvat scene-rakenne. Scenet koostuvat nodeista. Sceneillä on yksi root node, joten yksittäinen node voi olla scene. Sceneillä on samat ominaisuudet kuin nodeilla, mutta lisäksi niitä voi tallentaa kovalevylle ja lataa uudelleen käyttöön. Godotissa yksi sceneistä pitää olla main scene, eli scene, josta peli alkaa. Tässä pelissä main scenenä toimii päävalikko-scene. Kuvio 2 näyttää kehitetyn pelin scene-rakenteen. Päävalikosta voidaan siirtyä sceneihin: Credits, Main Scene ja Options. ”Sulje peli” osio kuviossa ei ole scene, vaan toiminto, joka sulkee pelin. Kun Päävalikosta on päästy Main Scene-sceneen, voi pelaaja siirtyä siitä Stage-sceneen, jossa sitten on Valikko-toiminto, mikä tulee silloin kuin pelaaja joko häviää tai voittaa pelin. Valikko-toiminnosta pääsee pelaaja takaisin Päävalikko-sceneen.



Kuvio 2: Pelin scene-rakenne

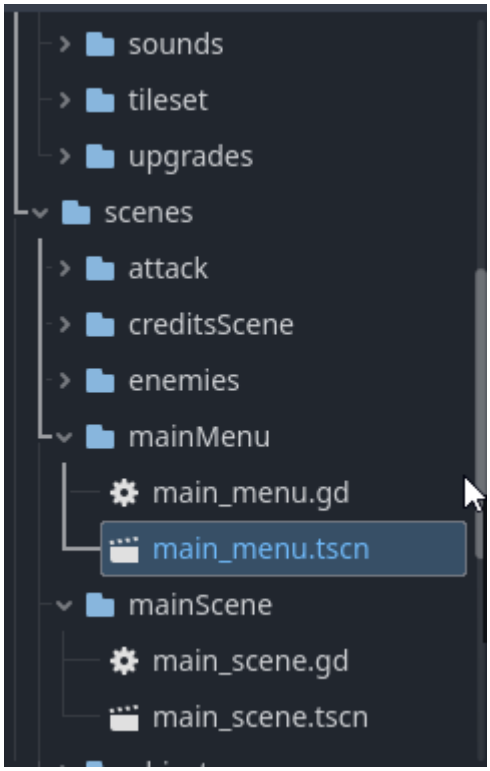
6.3.1 Päävalikko ja scene-siirtymät

Päävalikon-scenen rakenne näytetään kuviossa 3, ja se miten se näkyy näytöllä pelatessa kuviossa 4. Kuviossa 3 on MainMenu-noden ja button-nodejen kohdalla oikealla puolella kirjakäärön näköinen symboli. Se osoittaa sen, että näissä nodeissa on koodia, eli ohjelmointitiedosto liitettynä. Kuviossa 5 nähdään MainMenu-sceneen liitetyn koodin, joka on tiedostossa main_menu.gd, ja scene itse on tiedostona tallennettuna main_menu.tscn. Gd- ja tscn-tiedostotyypit ovat ominaisia Godot-peleille. Tscn-tiedostot sisältävät scene-tiedoston ja gd-tiedostot sisältävät koodia.



Kuvio 3: Päävalikko-scenen rakenne

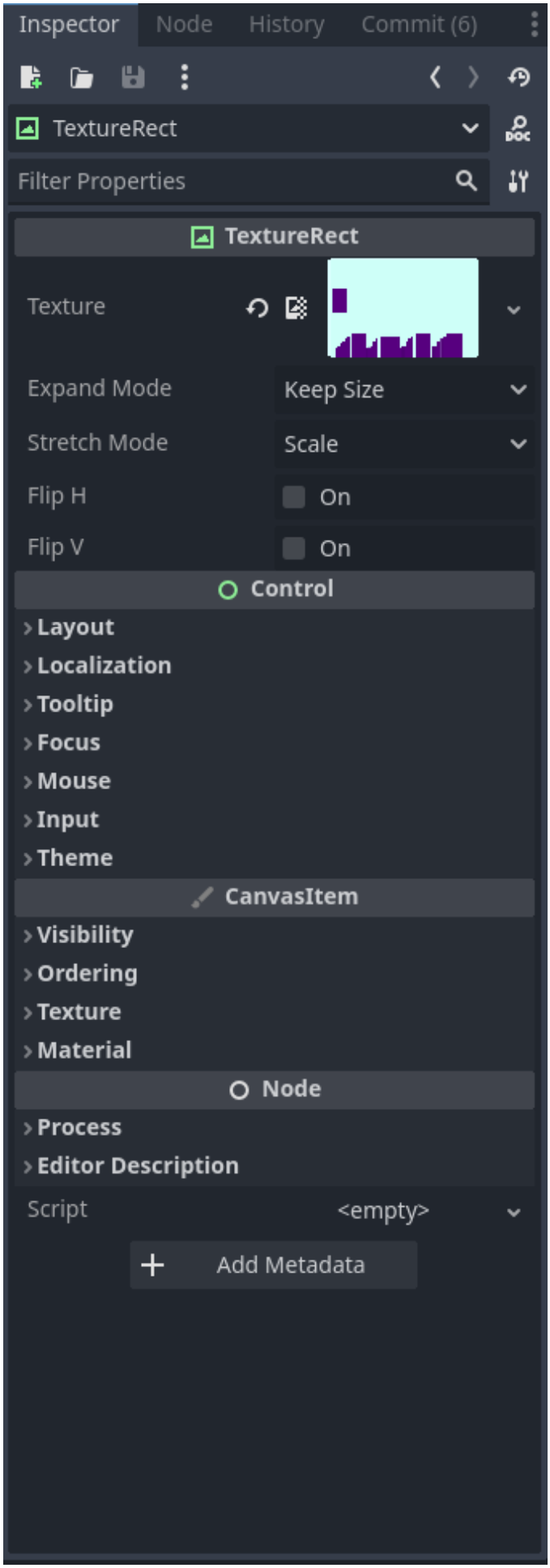
Kuvio 4: Päävalikko pelatessa



Kuvio 5: Päävalikko-scenen tiedosto ja scripti

Kuvio 3, eli päävälikko-scenen rakenne voidaan tulkita niin, että root nodena toimii Main-Menu-node, joka on tyyppiä Control (jokaisella nodella on oma tyyppi, joka päättää sen mitä ominaisuuksia nodella on). Sen lapsina ovat TextureRect- ja MarginContainer-nodet, joilla on myös lapsinodeja. TextureRect-nodessa sisältää kuvadatan, joka näkyy kuviossa 4 päävalikon taustakuvana. MarginContainer-noden lapsinodeina ovat TopContainer- ja HboxContainer-nodet. TopContainer-nodepuu on tässä scenessä otsikkoa näyttävä node, ja HboxContainer-nodepuu näyttää painikkeet, joissa on toimintoja. Separator-node on tyhjä node, eli siinä ei ole mitään ominaisuuksia, mutta se toimii tässä scenessä marginaaleina.

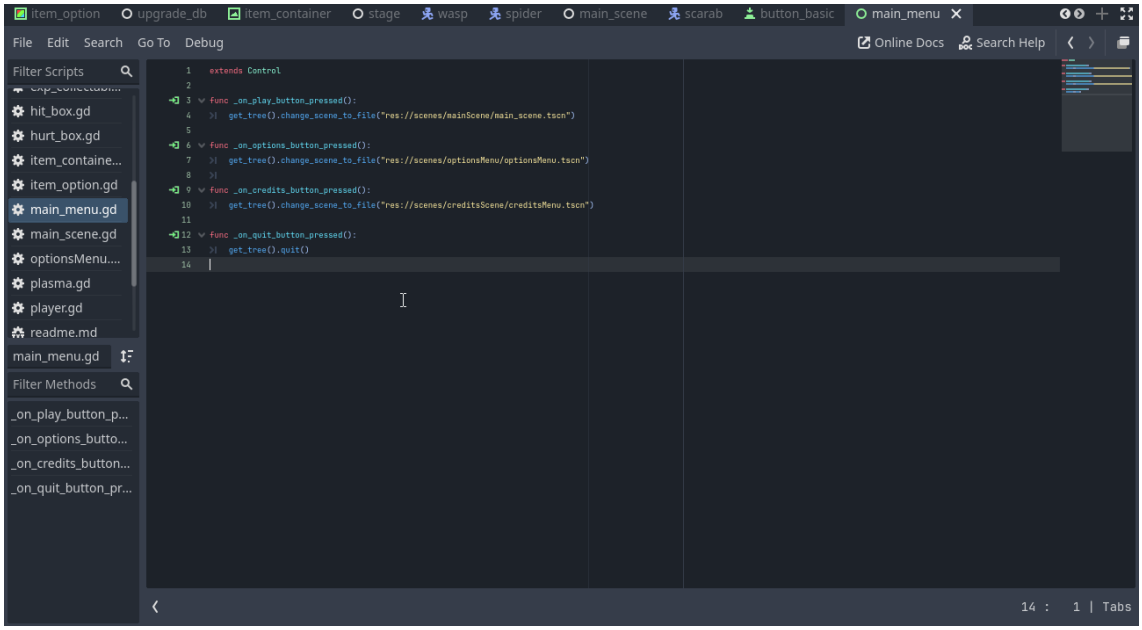
Jokaiseen noden ominaisuuksiin päästään käsiksi node inspectorilla, joka näkyy kuviossa 6. Kuviossa 6 on TextureRect-node tarkasteltavana. Texture-kohdassa voidaan lisätä kuva, joka tulee esiintymään pelissä. Siinä on muitakin asioita, joita voi vaikuttaa, mutta oletusarvot ovat yleensä tarpeeksi sopivat node-tyyppiä valitessa, joten niitä ei tarvitse muuttaa.


Inspector
Node
History
Commit (6)
TextureRect
Filter Properties
TextureRect
Texture
Expand Mode
Keep Size
Stretch Mode
Scale
Flip H
On
Flip V
On
Control
Layout
Localization
Tooltip
Focus
Mouse
Input
Theme
CanvasItem
Visibility
Ordering
Texture
Material
Node
Process
Editor Description
Script
<empty>
Add Metadata

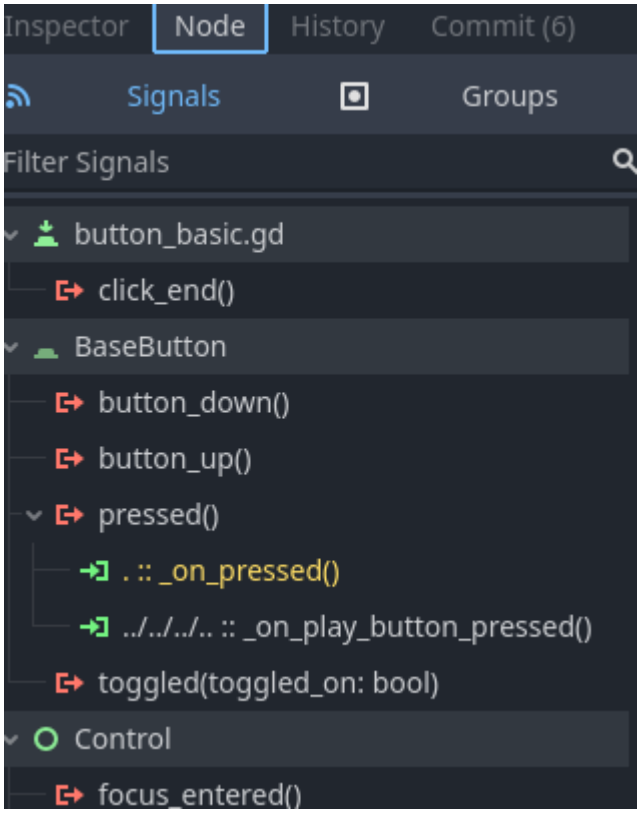
Kuvio 6: Node inspector

Jotta MainMenu-scenestä päästäisiin muihin sceneihin, pitäisi luoda scene siirtymät. Scene-siirtymät saadaan aikaan koodin kautta. Godot oletusohjelmointikielenä toimii niiden oma tekemä GDScript ja C#. Muitakin ohjelmointikieliä voitaisiin käyttää laajennuksien avulla, mutta tässä projektissa pidättäydyttiin GDScriptissä, sillä muut saatavilla olevat ohjelmointikielet olivat vieraita opinnäytetyöntekijälle. Godot ohjelmointi alkaa aina avainsanalla ”extends __”, missä ”__” on node, jota koodi muokkaa, mikä on tässä tapauksessa MainMenu-node (kuvio 3 & 7).

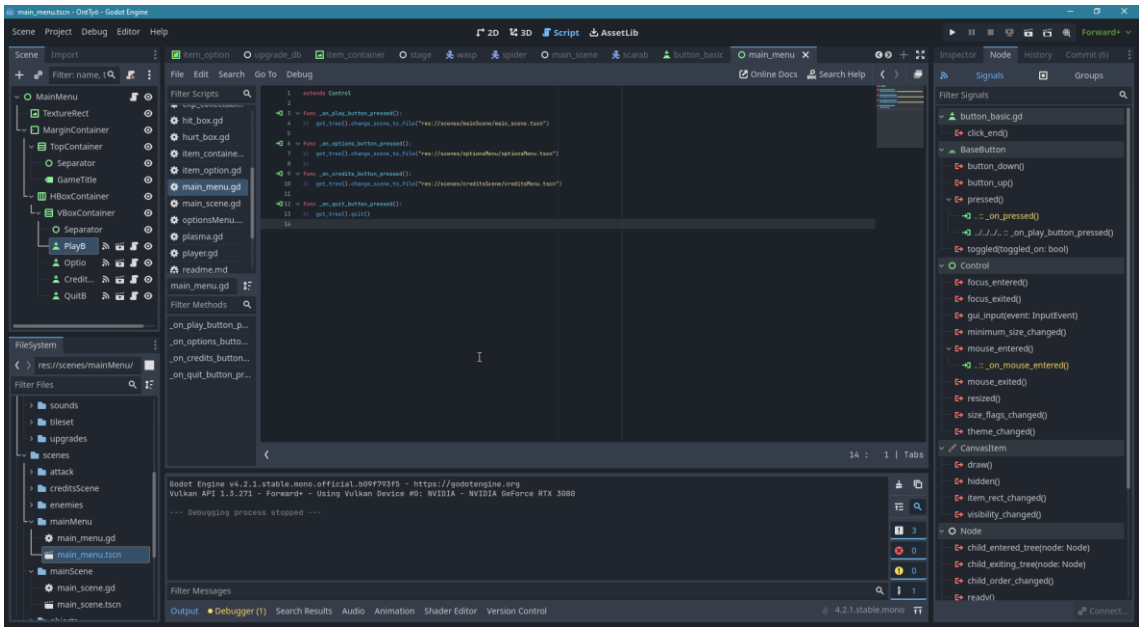
Kuviossa 8 näkyy vihreitä nuolia, jotka osoittavat neliöön koodilinjan ulkopuolella. Nämä vihreät nuolet ovat yhdistymiä (eng. connects), ne ottavat vastaan signaaleja, joita nodet lähettävät, mitkä ovat tässä tapauksessa nappi-nodet. Signaaleja voidaan lisätä koodin kautta, mutta myös visuaalisella työkalulla node inspectoria käyttäen (kuvio 8). Jokaisella node-tyypillä on omat valmiit signaalit, joita voidaan lisätä node-inspectorin kautta. Button-tyyppisillä nodeilla on pressed()-signaali, joka laukaisee sitä painaessa. Kun signaali laukaistaan, lähettää se signaalin (eng. emit signal), jota voidaan käyttää aktivoimaan jokin metodi. Esimerkkinä PlayButton-nappia painaessa käynnistyy metodi ”_on_play_button_pressed()”. Metodilla ”get_tree().change_scene_to_file()” tehdään scene-siirtymä. Metodi ”get_tree().quit()” sulkee selaimen. Kuviossa 9 nähdään Godot-pelimoottorin pelinkehittämisen graafinen ulkoasu kokonaisuudessaan.



Kuvio 7: main_menu.gd koodi



Kuvio 8: Node signaalit

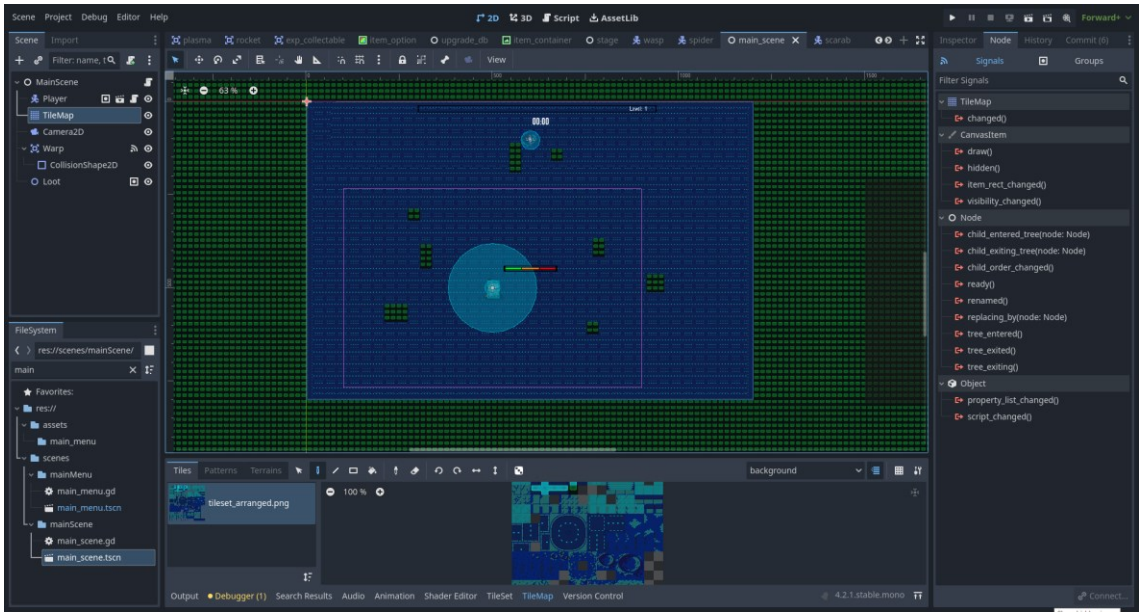


Kuvio 9: Godot-pelimoottorin käyttöliittymä

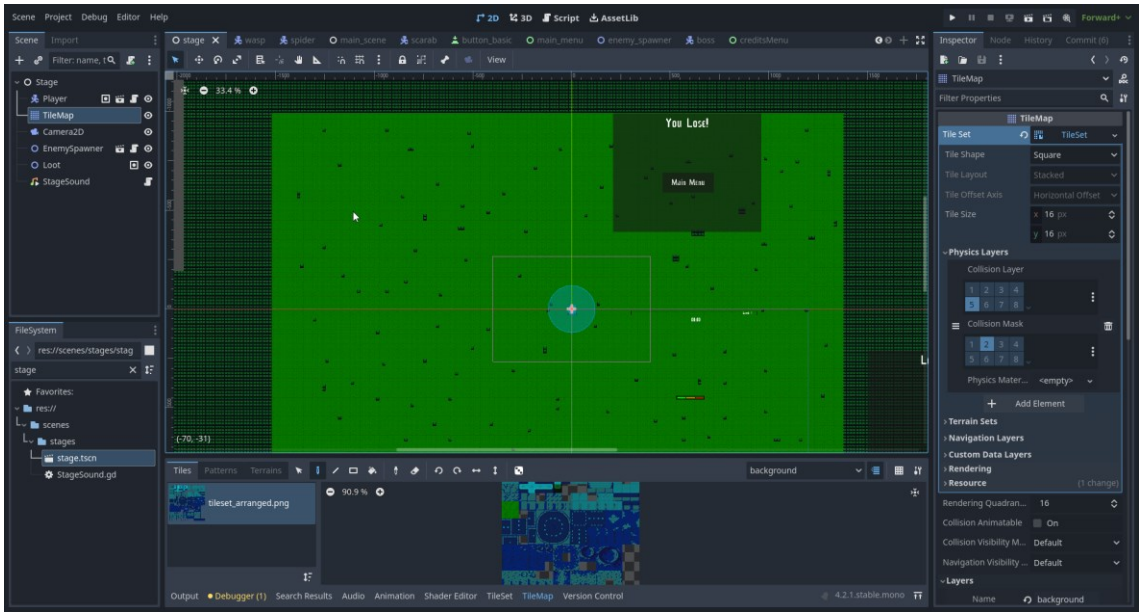
6.3.2 Ympäristö, collision ja maski

Päävalikko-scenestä siirrytään sceneen main_scene.tscn (kts. kuvio 10). Tämän scenen oli tarkoitus sisältää metaprogressioparannuksia ja aseiden/hahmojen vaihtomahdollisuuksia, mutta ”Bullet Heaven” kaltaiset pelit eivät niitä tarvitse, joten tuloksena jäi melko tyhjä taso, jossa pelaaja voi kokeilla hahmon ohjattavuutta. Tasossa on yksi kohta, johon pelaajahahmon siihen astuttua siirrytään stage.tscn-sceneen (kts. kuvio 11).

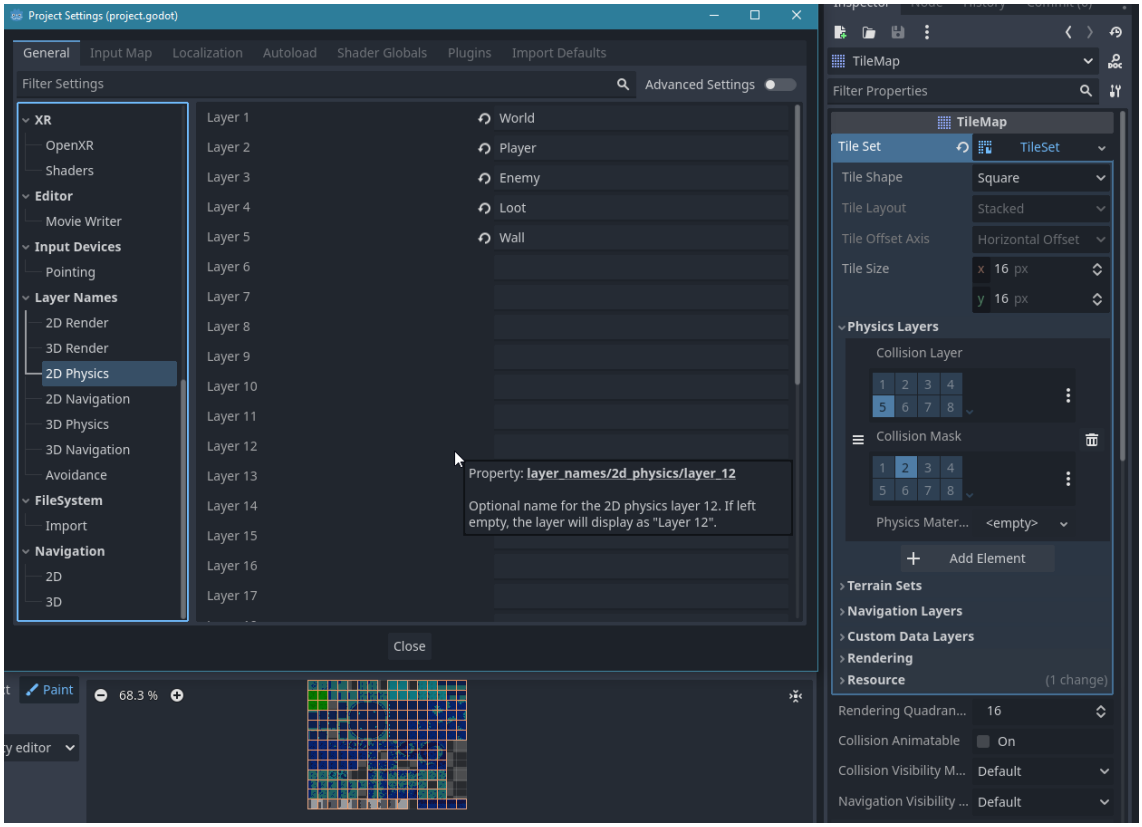
Main_scene- ja stage-scenet molemmat sisältävät Player-scenen, joka on oma scenensä: player.tscn. Niissä on myös TileMap-node, jonka avulla asetetaan peliympäristö. TileMap-nodeen lisätään node inspectorissa tileset, joka on kuva, jossa on rakennuspalasia 2D maailman luomiseen. Kuviossa 10 ja 11 alapuolella on tileset, josta voidaan hiirellä napsauttaa haluttu pala ruudukosta ja lisätä se maailmaan. Godotissa on tehokas automaattitunnistin tileset-tyyppisille kuville. Tilesetin ruudukon määrityksen voi tehdä itsekin, ja paljon muutakin ominaisuuksia voi muokata Godot-työkalulla, mutta tässä projektissa tärkeintä oli muokata fyysistä kerrosta (eng. physic layer). Jokaiselle laatalle voi laittaa fyysinen kerros, eli se kerros, jonka kanssa laatta (eng. tile) vuorovaikuttaa. Fyysinen kerros asetetaan projektin asetuksista kuvio 12 lailla. Sceneissä ja nodeissa ne muokataan koodin tai node inspectorin kautta (kts. kuvio 12). Collision-kerros (eng. collision layer) asettaa fyysisen kerroksen, jossa collision objekti on ja maski (eng. mask) asettaa fyysisen kerroksen, jonka kanssa tämä vuorovaikuttaa. Projektin esimerkkinä on kuviossa 10 oleva vihreä seinä, joka on kerroksessa 5, mikä on Wall-kerroksessa (kts. kuvio 12). Jos pelaaja, joka on fyysisellä kerroksella 2 koskettaa seinään ja sillä on maski kerroksella 5, ei se pääse sen läpi, sillä fysiikka simulaatio estää kahden saman fyysisen kerroksen objektin läpäisyä. Viholliset ovat kerroksella 3, joten nämä voivat läpäistä seinän.



Kuvio 10: main_scene ja tileset



Kuvio 11: stage

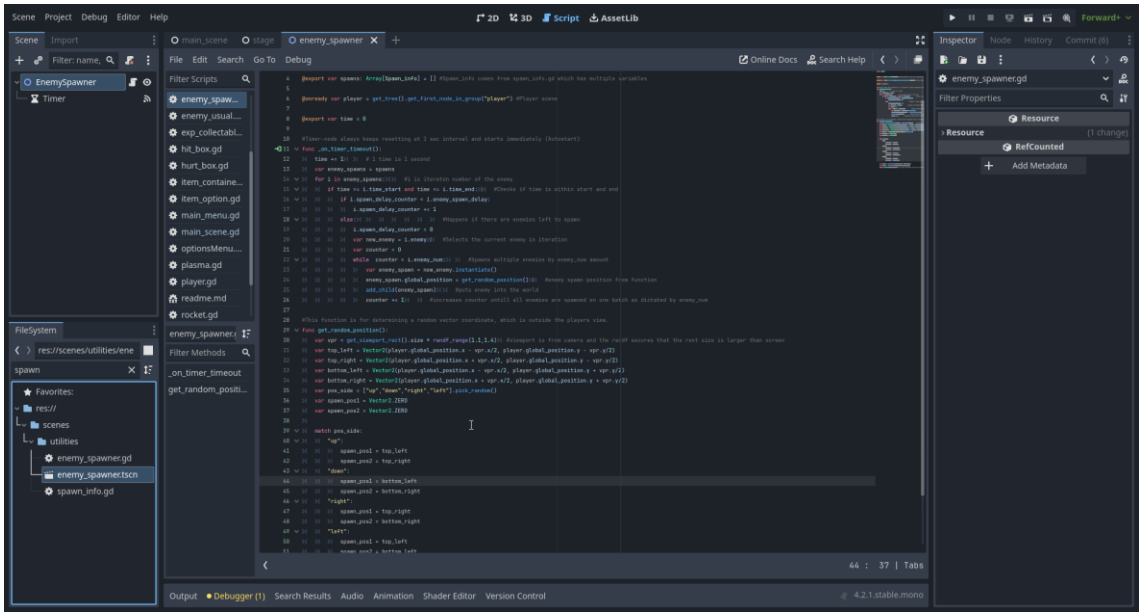


Kuvio 12: fyysinen kerros ja maski

Camera2D on pelitasossa, eikä pelaajassa, koska joskus on tarve erottaa pelikamera pelaajasta, kuten välianimaation (eng. cutscene) aikana. Tässä projektissa ei ollut tarve erottaa kamera pelaajasta, mutta se on olemassa tulevaa kehitystä varten. Main_scene-scenessä oleva Warp-node on scene-siirtymää varten. Kuviossa 10 ja 11 sinisellä näkyvät ympyrät ovat collision-alueita.

6.3.3 Vihollisten luominen

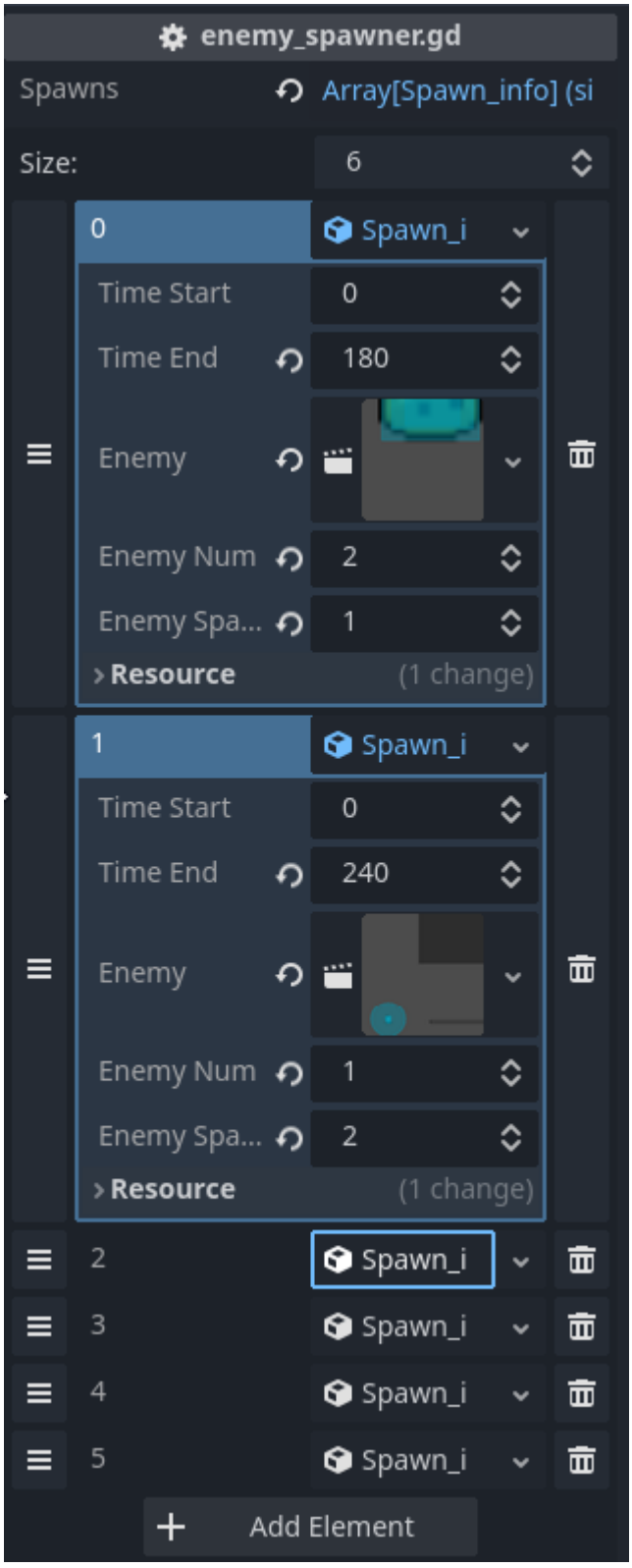
Vihollisten luominen satunnaisiin sijainteihin stage-scenessä tapahtuu EnemySpawner-nodessa, joka on oma scenensä. Kuviossa 13 on kaksi gd-tiedostoa, spawn_info.gd ja enemy_spawner.gd. Spawn_info.gd on muotoiltu JSON-tiedoston kaltaiseksi. Se itsessään ei ole liitettynä mihinkään sceneen tai nodeen, kuten aiemmat esitellyt gd-tiedostot. Sitä käytetään luomaan kuvio 15 kaltainen lista luotavista vihollisista. Avainsana @export lisää node inspectoriin muokattavia ominaisuuksia (kuvio 14 ja 15).



Kuvio 13: Vihollisten luomisprosessi

```gdscript
extends Resource

class_name Spawn_info

@export var time_start:int
@export var time_end:int
@export var enemy:Resource
@export var enemy_num:int
@export var enemy_spawn_delay:int

var spawn_delay_counter = 0

```

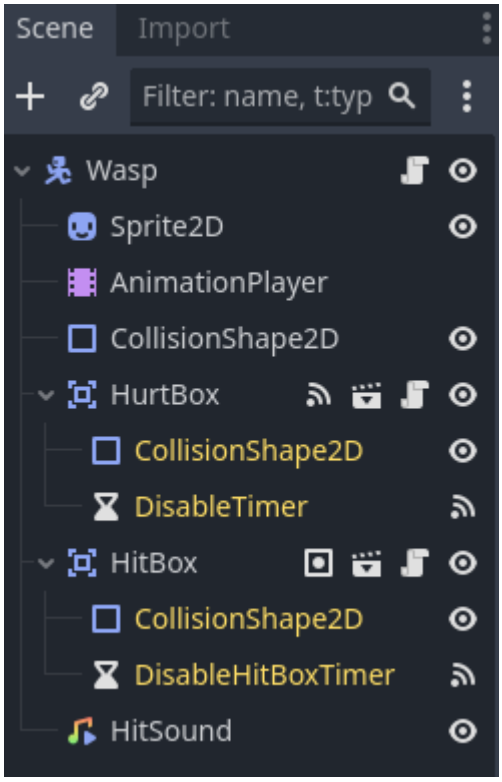
Kuvio 14: Spawn_info.gd



Kuvio 15: Vihollisten luontitiedot

Metodi ”_on_timer_timeout()” on yhdistettynä Timer-nodeen, ja se laukaisee silloin kun Timer-nodessa asetettu aika on kulunut. Metodi katsoo EnemySpawner-noden node inspectorilla

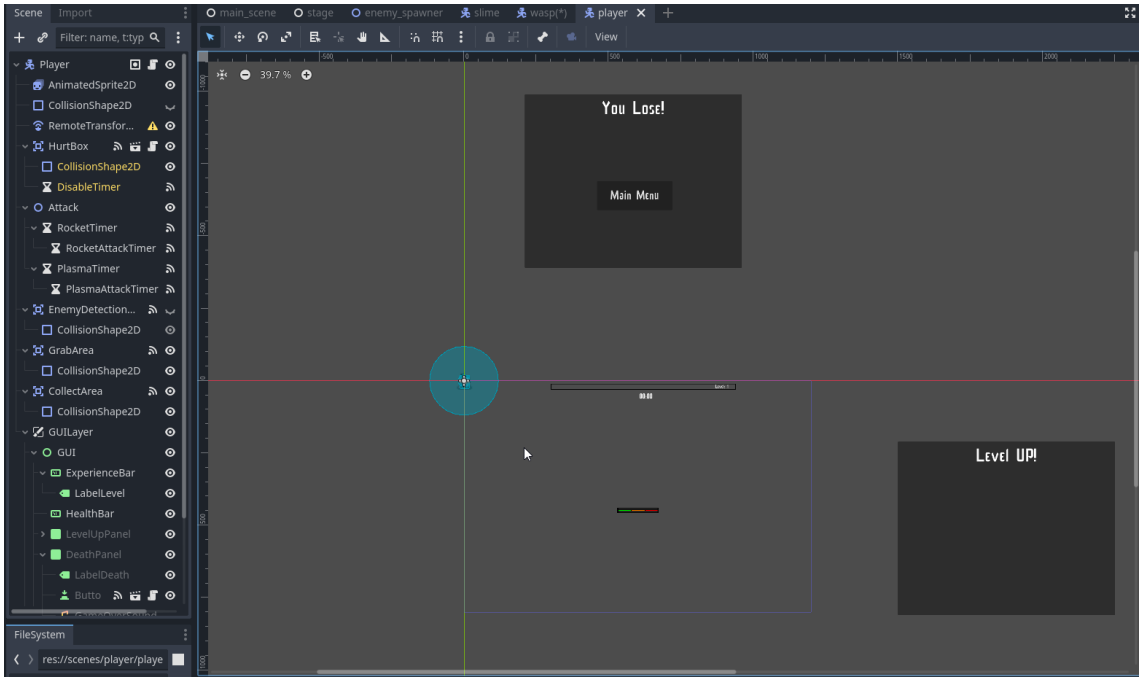
asetettujen vihollisten luontitiedot. Instantiate() luo noden ja add.child() metodi lisää vihollis-nodet lapsinodeina sceneen. Metodissa ”get_random_position()” tapahtuu laskenta vihollisten paikalle, joka on aina poissa pelaajan kamerasta. Spawn_info.gd:ssä on kohta ”@export var enemy:Resource”. ”Resource” viittaa mitä vain tiedostoa Godotissa, kuten sceneä tai kooditiedostoa. Tässä tapauksessa viitataan vihollis-sceneen, joka näyttää kuvion 16 kaltaiselta. Kuvion 16 Wasp-scene sisältää animaatiot, collision objektit, äänet ja koodin. Pelissä on muitakin vihollisia.



Kuvio 16: Wasp vihollinen-scene

6.3.4 Player-scene, UI, hyökkääminen ja hahmoparannukset

Player-scenessä on paljon asioita tiukasti sidottuna. Siinä on pelaajahahmon kuvat, animaatiot, äänet, collision objektit, hyökkäykset, hahmon parannukset, pelin voitto- ja häviö kytkimet, sekä pelin UI, eli User Interface (kts. kuvio 17 ja kuvio 18). Näiden asioiden avaaminen veisi monta sivua, joten niiden esittely on jätetty väliin. Asiasta kiinnostuneet voivat tarkastella lähdekoodia liitteessä 1 olevan Github-linkin kautta.

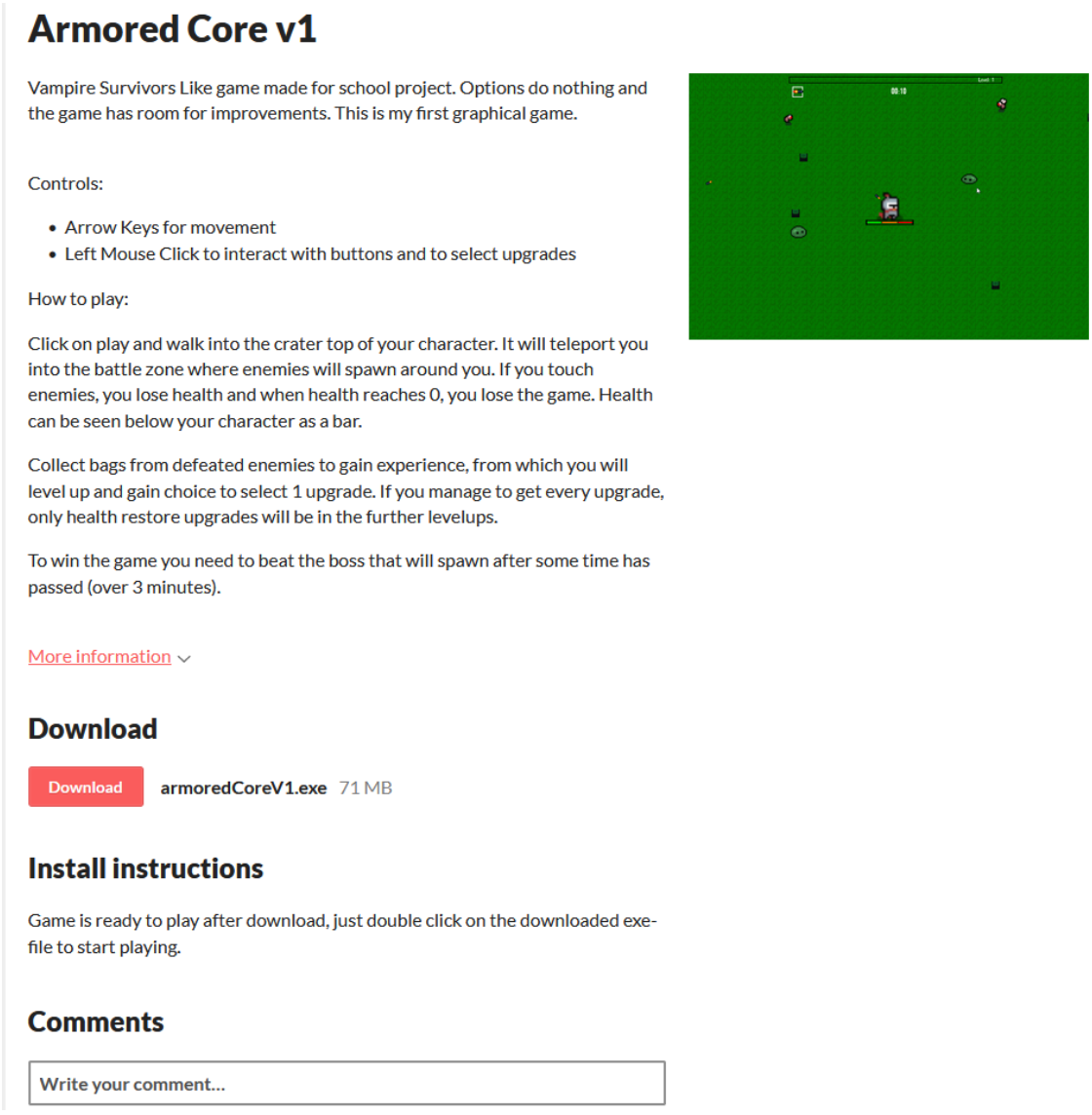


Kuvio 17: Player-scene

Kuvio 18: Hahmon parannus mahdollisuudet

6.4 Pelin julkaisu

Markkinoilla olevien pelien ei tarvitse sisältää pelin toimimisen kannalta muita asioita, joten Godot-peliprojekti piti muuttaa markkinaversioksi, joten projekti tuotiin (eng. export) exe-tiedostoksi. Peli julkaistiin Github- ja itch.io-sivuille. Github-sivulla on pelin lähdekoodi, ja itch.io-sivulla on pelin lyhyt esittely ja exe-tiedosto lataus. Peliä ei ollut testattu ulkopuolisten toimin ennen pelin julkaisua. Pelillä ei ole tarkoitus tienata voittoa, joten se julkaistiin itch.io-sivulle täysin ilmaisena pelinä.



Kuvio 19: Julkaistu peli itch.io-sivulla

7 Kehittämistyön tulos

Kehittämistyönä saatiin aikaan valmis avoimen lähdekoodin peli, jota pystytään pelaamaan alusta loppuun saakka. Pelissä ei ollut testaajia, joten pelin toiminen äärimmäisten tapauksien kanssa ei ole taattua, joten pelin toimivuus saadaan selville vasta kun pelin pelaajilta on saatu palautteita. Tavoitteena oli luoda 2D ylhäältäpäin kuvattu toiminta roguelite, mikä vastaa laajalti kehitettyä peliä. Peliä kehittäessä opin, että alussa olevat kunnianhimoiset tavoitteet muuttuvat kehityksen aikana realistisemmaksi, kun kokemusta pelinkehittämisessä karttuu.

Nykyisen version Godot-pelimoottorissa on vielä silmään ponnahtavia bugeja esiintynyt kehittäessä. Scenen lisääminen resurssina nodeen näyttää jonkin toisen scenen sprite-hahmoa, vanhan gd-tiedoston poistaminen ja uuden linkittäminen nodeen ei päivitä uuden gd-tiedoston nimeä heti, vaan silloin kuin Godot käynnistetään uudestaan. Muitakin bugeja on tullut vastaan. Godot 4 julkaistiin vuoden alkupuolella, jolloin netistä saattaa olla vaikea löytää päivitettyä tietoa uusimmasta Godot versiosta, joka hankaloitti pelin kehittämistä paljon. Godot 3 ja 4 välillä tapahtui suuria muutoksia, jolloin Godot 3 ratkaisut eivät toimineet enää Godot 4:ssä ja graafinen käyttöliittymä oli tehty uusiksi. Näistä seikoista huolimatta on Godot-pelimoottorilla 2D pelin kehittäminen suotuisaa. Lähtökohtana oli Unityn hintamallimuutos, mutta Godot työympäristö vaikutti soveltuvalta pienille 2D peliprojekteille, joten uskon tulevaisuudessakin kehittävän uudet pelit Godot-pelimoottorilla.

8 Yhteenveto

Opinnäytetyön teoriaosiossa tutustuttiin vapaa- ja avoimen lähdekoodin ohjelmistojen käsitteisiin ja pelinkehittämiseen yleisellä tasolla. Kehitystyössä testattiin Godot-pelimoottorin toimintoja ja sen käytettävyyttä 2D pelien kehittämiseen. Kehitystyönä oli 2D ylhäältäpäin kuvattu toiminta roguelite, joka muuttui ”Bullet Heaven”-tyyppiseksi peliksi, mikä oli todennäköisempi kehittää ensityönä opiskelijalle.

Godot-pelimoottorin GDScript ohjelmointikieleen en päässyt syvälle tutustumaan, mutta kokeiltua tuli ja helpolta se vaikutti. C# ohjelmointikieltä kehotetaan käyttämään Godot-pelejä kehittäessä, sillä se on paremmin optimoitu. Seuraavissa projekteissa tulen käyttämään C# ohjelmointikieltä, sillä muissa pelinkehitysyrityksissä nähdään C# hyödyllisempänä kuin GDScript, koska sitä voi soveltaa muuhunkin kuin Godot-pelien kehittämiseen. Aikataulutuksessa oli haasteita. Kehitystyö, jonka kuului kestää n. 1 kuukausi piteni kahteen. Jotta kehittyisin pelinkehittäjänä, tulisi minun käyttää ohjelmoinnin vaihejakomalleja aikataulutukseen. Pieniin pelinkehittämisryhmiin sopivat ketterät menetelmät, joten niitä olisi hyvä opiskella.

Parannettavaa on pelissä vielä laajalti. Vähemmän kriittisenä, mutta tärkeänä asiana parannusta olisi tiedostojen järjestely ja koodin optimointi. Kehittäessä koitin pidättäytyä hyviksi tunnetuissa ohjelmointitavoissa, mutta huolimattomuuden takia saatoin tehdä hutiloituja ratkaisuja, jotka turhaan monimutkaistivat kehittämistä. Pelin resurssien löytäminen osoittautui hankalaksi, ja niiden hyödyntämiseen tarvitsisin lisää harjoittelua. Kehitystyössä paljastui pelinteon olevan paljon enemmänkin kuin ohjelmointia ja pelimoottorin käyttöä.

Lähteet

Painetut

Bradfield, C. 2023. Godot 4 Game Development Projects: Build five cross-platform 2D and 3D games using one of the most powerful open source game engines. 2. painos. Iso-Britannia: Packt Publishing.

Sähköiset

Bond J.G, 2023. Introduction to Game Design, Prototyping, and Development From Concept to Playable Game with Unity and C#. 3. Painos. E-kirja. Lontoo: Pearson.

Bustamante II. & Cohen D. 2010. Producing Games; From Business and Budgets to Creativity and Design. E-kirja. Burlington: Elsevier.

Egenfeldt-Nielsen S, Smith J.H & Tosca S.P. 2016. Understanding video games the essential introduction. 3. painos. E-kirja. New York: Routledge.

GNU, 2023. Various Licenses and Comments about Them. Viitattu 10.12.2023. https://www.gnu.org/licenses/license-list.html.

GNU, 2023. What is Free Software? Viitattu 10.12.2023. https://www.gnu.org/philosophy/free-sw.en.html.

Godot, 2023a. License. Viitattu 14.12.2023. https://godotengine.org/license/.

Godot, 2023b. Why Godot is right for you. Viitattu 14.12.2023. https://godotengine.org/features/.

Godot, 2024a. Nodes and Scenes. Viitattu 16.2.2024. https://docs.godotengine.org/en/stable/getting_started/step_by_step/nodes_and_scenes.html.

Godot, 2024a. Using Signals. Viitattu 17.2.2024. https://docs.godotengine.org/en/stable/getting_started/step_by_step/signals.html.

Kevuru Games, 2023. Indie game development: guide to revenues, most profitable genres & monetization [+10 best indie games 2023]. Viitattu 12.12.2023. https://kevurugames.com/blog/indie-game-development-the-all-you-need-guide-to-revenues-most-profitable-genres-monetization-bonus-top-10-best-indie-games-2020/.

Mkrtchyan Rafayel, 2019. The Code and Fix Model. Viitattu 28.2.2024. https://productcoalition.com/the-code-and-fix-model-2cabd4c48166.

Open Source Initiative, 2006. History of the OSI. Viitattu 10.12.2023. https://opensource.org/history/.

Open Source Initiative, 2023a. About the Open Source Initiative. Viitattu 11.12.2023. https://opensource.org/about/.

Open Source Initiative, 2023b. The Open Source Definition. Viitattu 10.12.2023. https://opensource.org/osd/.

Stallman Richard M. 2015. Free Software, Free Society: Selected Essays of Richard M. 3. Painos. E-kirja. Boston: Free Software Foundation.

Stefyn Nadia, 2022. How video games are made: the game development process. Viitattu 12.12.2023. https://www.cgspectrum.com/blog/game-development-process.

Kuviot

Liitteet

Liite 1: Github repository ja itch.io-linkki

Github-linkki: https://github.com/nmnn33/Armored-Core-game

Itch.io-linkki: https://ultrasts5.itch.io/armored-core-v1