OULUN AMMATTIKORKEAKOULU

BULLET HELL -TYYLISEN PELIN KEHITTÄMINEN GODOT-PELI-MOOTTORILLA

Eemeli Partanen & Pete Kontiokari
Opinnäytetyö (AMK)
Kevät 2025
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä(t): Eemeli Partanen, Pete Kontiokari
Opinnäytetyön otsikko: Bullet Hell -tyylisen pelin kehittäminen Godot-pelimoottorilla
Työn ohjaaja(t): Pasi Mustonen
Työn valmistumislukukausi ja -vuosi: kevät 2025
Sivumäärä: 50 + 1 liite

Opinnäytetyön päätavoite oli luoda kaksiulotteinen peli hyödyntäen Godot -pelimoottorin pääominaisuuksia. Pelistä kerättiin myös käyttäjätutkimus, jonka pohjalta pohdittiin, mitä parannuksia peliin täytyisi tehdä julkaisua varten.

Opinnäytetyössä käydään läpi pelimoottorien historiaa ja keskitytään erityisesti Godot -pelimoottorin ominaisuuksiin sekä tutustutaan sen omaan GDScript -ohjelmointikieleen. Verrataan myös, miten Godot eroaa muista pelimoottoreista ja miksi lopulta valitsimme Godotin jonkin toisen pelimoottorin sijaan.

Opinnäytetyön teoriaosiossa tutustutaan pelien suunnittelun eri vaiheisiin ja suunnitellaan oma, kaksiulotteinen videopeli. Käydään myös läpi mitä asioita peli tarvitsee, että se olisi hyvä ja mukaansatempaava. Tässä osassa käydään myös läpi, miten pelin äänet ja käyttöliittymä voivat välittää pelaajalle tärkeää informaatiota.

Käyttäjätutkimuksesta saadun palautteen pohjalta saatiin tärkeää tietoa, miten pelinkehittäjästä järkevältä vaikuttavat ratkaisut eivät välttämättä ole pelaajan tai pelattavuuden kannalta parhaita ratkaisuja. Tutkimuksen pohjalta saimme myös selville asioita, mitkä kehittämämme peli toteutti oikein, ja mitkä osat pelistä tarvitsevat vielä lisää testausta ja kehittämistä.

Opinnäytetyöprosessin alussa pelille määritetyt vaatimukset saatiin toteutettua onnistuneesti ja tavoitteeseen luoda pelistä alusta loppuun pelattava versio päästiin. Opinnäytetyöprosessin aikana kehitetyn pelin myötä ymmärrys pelin kehittämisen prosessista, sekä erityisesti taidot GDScriptin käytössä parantuivat huomattavasti. Näitä opittuja taitoja voidaan tulevaisuudessa hyödyntää pelin jatkokehityksessä tai muissa projekteissa.

ABSTRACT

Oulu University of Applied Sciences
Degree program in Information Technology
Option of Software development

Author(s): Eemeli Partanen, Pete Kontiokari
Title of thesis: Developing a Bullet Hell –style game using the Godot game engine
Supervisor(s): Pasi Mustonen
Term and year when the thesis was submitted: Spring 2025
Number of pages: 50 + 1 appendice

The main goal of this thesis was to create a two-dimensional game using the Godot game engine. User feedback was also collected about the game, and it was used to consider what improvements should be implemented for the game to be ready for publishing.

The thesis examines the history of game engines and focuses especially on the features of the Godot game engine and the GDScript programming language built for it. It also compares the differences between Godot and other game engines and at the end, why we chose to use Godot instead of any other game engine.

In the theory part of the thesis, we go over the different parts of game design and design our own two-dimensional game. The thesis also goes over what things a game needs, for it to be good and engaging, and how sound and parts of the user interface can be used to convey useful information to the player.

Based on the feedback received, we gained insight on how sometimes when looking at things from a developer's perspective, it might not be the best way to implement in terms of gameplay or player experience. The feedback also gave us information on what we did right when developing the game and which parts still need more testing and development.

The features defined for the game at the start of the thesis process were successfully implemented and the goal of creating a game which you can play from start to finish was achieved. By creating a game as part of the thesis we gained a better understanding of the game development process and especially learned how to better use the GDScript language. The skills learned can be used in the future developing the game or in other projects.

SISÄLLYS

SANASTO

Bullet Hell	Shoot ‘em up peligenren alagenre, jossa pelaajan on tarkoitus väistää suuria määriä ammuksia, joita viholliset ampuvat pelaajaa kohti
FPS	First Person Shooter eli ensimmäisen persoonan ammuntapeli
NPC	Non-Player-Character eli pelin hahmot, joita pelaaja ei ohjaa
OCR-silmukka	Objective, Challenge ja Reward sanoista koostuva sana on pelisuunnittelun malli, jota käytetään kuvaamaan erilaisia pelimotivaatiotasoja jakamalla haasteita pelin pitämiseksi kiinnostavana.
Atomiparametrit	Parametrejä, jotka viittaavat pienin, jakamattomin arvoihin ja muuttujiin, jotka ohjaavat pelin käyttäytymistä, tuntumaa ja tasapainoa.

1 JOHDANTO

Opinnäytetyön tavoitteena on kehittää ja dokumentoida kaksiulotteisen, ylhäältäpäin kuvatun bullet hell –tyylisen videopelin kehitysprosessia. Projektin idea lähti halusta kehittää oma videopeli ja tutustua Godot-pelimoottoriin ja sen käyttämään GDScript ohjelmointikieleen. Pelin moottoriksi valittiin Godot-pelimoottori, koska se on ilmainen avoimen lähdekoodin pelimoottori, jonka suosio on lähiaikoina noussut ja se tarjoaa sopivat työkalut haluamamme videopelin luomiseen. Muita vaihtoehtoja projektissa käytettäväksi pelimoottoriksi olisi voinut olla Unity-pelimoottori sekä Unreal Engine-pelimoottori.

Tavoitteena on myös kehittää peliohjelmoinnin taitoja ja tutustua eri pelimoottoreihin sekä saada syvempi ymmärrys siitä, miten Godot-pelimoottori ja GDscript-ohjelmointikieli toimivat videopelien kehittämisessä. Insinöörityön päätavoite on luoda toimiva pohjaversio videopelille, joka voidaan kehittää tulevaisuudessa julkaisuvalmiiksi tuotteeksi.

Opinnäytetyön yhteydessä tehdystä pelistä on myös tarkoitus tehdä joka viikko oma demoversio, jotta on helpompaa rajata peliprojektin kehittäminen tiettyihin asioihin, helpottaen määritettyjen toimintojen toteuttamiseen keskittymistä.

Pelistä on tarkoitus julkaista pelattava demoversio jollekin verkkoalustalle ja sen pohjalta kerätään käyttäjätutkimuksella käyttäjien mielipiteitä opinnäytetyön ohessa tehdystä pelistä. Käyttäjätutkimuksen pohjalta mietitään, missä onnistuttiin, ja mitä olisi voitu tehdä paremmin sekä pohditaan mahdollisia vaihtoehtoja pelin jatkokehitystä varten.

Opinnäytetyön alussa käydään ensiksi läpi Godot-pelimoottorin ominaisuuksia ja käyttöliittymää ja verrataan, miten se eroaa muista pelimoottoreista sekä tutustaan GDScript-ohjelmointikieleen. Käydään myös läpi pelimoottorien historiaa ja kehitystä. Opinnäytetyön loppupuolella katsotaan videopelin luontia Godot-pelimoottoria hyödyntäen.

2 PELIMOOTTORIT

Pelimoottori on ohjelmistokehitysohjelma tai -ympäristö, jota alun perin käytettiin videopelien kehittämiseen. Nykypäivänä pelimoottoria voidaan myös käyttää esimerkiksi visualisointiin, yhteistyöhön tai johonkin muuhun. Pelimoottorien ominaisuuksiin voi kuulu erilaiset animaatiotyökalut, tekoäly, fysiikka- ja törmäysmoottorit, äänimoottorit tai paljon muuta. (Perforce s.a.)

Pelimoottorit toimivat luomalla kehyksen, jonka avulla käyttäjä pystyy luomaan videopelin helpommin kuin jos heidän täytyisi tehdä se täysin tyhjästä. Pelimoottorin tarjoama kehys vaihtelee pelimoottorista toiseen, mutta tyypillisesti ne sisältävät kaksiulotteisen tai kolmiulotteisen renderöintimoottorin. (Perforce s.a.)

2.1 Pelimoottorien historiaa

1970-luvulla ja 1980-luvun alussa videopelien kehittäminen oli työlästä, sillä jokainen peli rakennettiin alusta alkaen tietyn laitteiston rajoitusten mukaisesti. Peleille luotiin omia koodikantoja, yleensä käyttäen alustalle tehtyä assembly-ohjelmointikieltä käyttäen. Tällainen kehitysprosessi oli työläs ja rajoitti mahdollisuuksia hyödyntää aiemmin kirjoitettua koodia muissa projekteissa. Esimerkiksi Atari -2600 konsolille kehitetyt pelit oli kirjoitettu alustakohtaisella assembly-kielellä, joka teki alustariippumattomasta kehityksestä melkein mahdotonta.

1990-luku puolestaan merkkasi suuren muutoksen videopelin kehityksessä, kun ensimmäiset uudelleenkäytettävät pelimoottorit otettiin käyttöön. Tämä antoi videopelikehittäjille mahdollisuuden rakentaa useita pelejä käyttäen samaa viitekehystä. Merkittävä esimerkki tästä on id Softwaren julkaisema id Tech 1 -pelimoottori, joka tunnetaan paremmin nimellä “Doom Engine”. Pelimoottori tarjosi vankan pohjan FPS-pelien kehittämiselle sekä mahdollisti pelien muokkaamisen ja uuden sisällön luomisen.

Vuonna 1996 id Software vei pelinkehitystä jälleen eteenpäin julkaisemalla “Quake Enginen”, joka toi mukanaan uusia ominaisuuksia, kuten aidon

reaaliaikaisen 3D-renderöinnin ja tuen 3D-laitteistokiihdytykselle. Tämä asetti uudet standardit videopelien kehitykselle. (Gazula, s.a.)

1990-luvun loppupuoli ja 2000-luvun alku merkkasi nopeaa kehitystä videopelien 3D-grafiikassa johtuen grafiikkaprosessorien nopeasta tehon kasvusta (Stockton 28.11.2023). Pelimoottorit kuten Epic Gamesin "Unreal Engine", joka otettiin käyttöön vuonna 1998, toivat mukanaan korkeamman tason pelinkehityksen realismille sekä monimutkaisuudelle muuttaen alaa suuresti (Gazula, s.a.).

Näinä aikoina myös väliohjelmistoratkaisut tulivat esille, tarjoten videopelien kehittäjille erikoistyökaluja fysiikkasimulaatioita, tekoälyä ja äänen prosessointia varten, joka virtaviivaisti pelien kehittämistä ja paransi niiden laatua vielä enemmän.

2000-luvun puolivälissä yleistyivät pelimoottorit, jotka korostivat saavutettavuutta ja alustariippumattomuutta. Unity, joka julkaistiin vuonna 2005, helpotti pelinkehitystä tarjoamalla intuitiivisen käyttöliittymän ja monialustatuen, minkä ansiosta siitä tuli suosittu valinta Indie-kehittäjien keskuudessa (Gazula s.a.).

Samanaikaisesti avoimen lähdekoodin pelimoottorit kuten Godot, joka julkaistiin vuonna 2014 saivat kannatusta mahdollistamalla ilmaisen yhteisön edistämän vaihtoehdon, jossa on vahvat ominaisuudet (Dawe 2014).

Videopelimoottorien evoluutio on siis laajasti vaikuttanut videopelialaan, muuttaen ohjelmoinnin räätälöidystä, laitteistosta riippuvasta ohjelmoinnista monipuolisiin, uudelleenkäytettäviin kehikkoihin antaen videopelien kehittäjille mahdollisuuden luoda monipuolisia ja monimutkaisia pelejä. Tämä on mahdollistanut videopelien luomisen kaikille suurista studioista lähtien pieniin Indie-kehittäjiin, antaen heille työkalut tuoda omat visionsa esiin monilla alustoilla.

2.2 Godot-pelimoottori

Godot-pelimoottori on ilmainen, avoimen lähdekoodin pelimoottori, joka tarjoaa toimivan ympäristön 2D- ja 3D-pelien kehittämiselle. Godot on lisensoitu MIT-lisenssillä, mahdollistaen sen käytön sekä kaupallisissa että ei-kaupallisissa

tarkoituksissa ilman rojalteja tai lisenssimaksuja. (Godot Engine 2025a). Tämä tekee pelimoottorista houkuttelevan vaihtoehdon erityisesti Indie-kehittäjille, pienille tiimeille ja opiskelijoille.

Godotin suunnittelufilosofia keskittyy modulaarisuuteen ja käytettävyyteen. Pelimoottorin kehitysympäristö sisältää valmiiksi kaikki tärkeimmät työkalut, kuten visuaalisen käyttöliittymän rakentajan, animaatiotyökalut, koodieditorin, työkalut debuggaukseen sekä myös kohtauspohjaisen järjestelmän, joka auttaa pelilogiikan organisoimisessa ja selkeydessä. (Godot Engine 2025b). Pelimoottorin yhtenäinen ja skaalautuva käyttöliittymä antaa kehittäjille mahdollisuuden luoda kokonaisia projekteja ilman että heidän tarvitsee turvautua muihin ohjelmistoihin.

Ohjelmointikielenä Godotissa käytetään ensisijaisesti GDScript-kieltä, joka vastaa syntaksiltaan Pythonia ja on optimoitu pelien kehittämistä varten. Lisäksi Godot tukee myös eri ohjelmointikieliä kuten C#:a sekä C++/C-laajennuksia GDExtension teknologian avulla. (Godot Engine 2025b). Tämä antaa pelien kehittäjille mahdollisuuden valita oman projektinsa vaatimuksien ja tarpeiden mukaan sopivimman ohjelmointikielen.

Godot tukee eri alustoja monipuolisesti pelin kehittämisen sekä julkaisun osalta. Pelejä on mahdollista julkaista ja kehittää muun muassa Windowsille, Linuxille, macOS:lle, androidille, iOS:lle, verkkoon (HTML5) sekä useille konsoleille (Godot Engine 2025c). Tämä antaa kehittäjille kyvyn saavuttaa laajan yleisön ilman että heidän tarvitsee käyttää monimutkaisia ratkaisuja ohjelmiston siirtämiseen alustalta toiselle.

Godot on jatkuvassa kehityksessä johtuen sen avoimen lähdekoodin luonteesta ja aktiivisesta yhteisöstä. Yhteisö osallistuu laajasti uusien ominaisuuksien suunnitteluun, bugien korjaukseen ja dokumentaation luomiseen. Dokumentaatio on laaja ja selkeä, joka edesauttaa uusien käyttäjien perehtymistä siihen, miten moottori toimii. (Godot Engine 2025b).

2.2.1 Historia

Godotin lähtökohdat ovat vuodessa 1999, kun Juan ‘reduz’ Linietsky ja Ariel ‘punto’ Manzur perustivat videopelien kehitystä konsultoivan yrityksen nimeltään Codenix. Pelimoottorin ensimmäiset versiot luotiin jo vuonna 2001, kun he aloittivat tekemään pelimoottoria, jota kutsuttiin silloin koodinimellä “Larvotor”. Tätä pelimoottoria lisensoitiin kolmannen osapuolen yrityksille Argentiinassa. Seuraavan kymmenen vuoden aikana pelimoottorilla tulisi olemaan monta erilaista nimeä, kuten “Legacy”, “NG3D ja “Larvita”, ennen kuin kehittäjät päätyisivät nimeen Godot. Pelimoottorin lopulliseen nimeen päädyttiin viittauksena Samuel Beckettin näytelmään “Huomenna hän tulee” (Waiting for Godot), sillä se edusti Godotin perustajien loputonta toivetta uusien ominaisuuksien lisäämisestä pelimoottoriin, vaikka täydellistä tuotetta ei koskaan saavutettaisikaan. (Linietsky 2014a; Wikipedia 2025.)

Vuonna 2014 Godot-pelimoottorista tehtiin avoimen lähdekoodin projekti ja se julkaistiin MIT-lisensoituna GitHub-alustalle (Dawe 2014). Saman vuoden joulukuussa pelimoottorista julkaistaisiin myös sen ensimmäinen stabiili versio, Godot 1.0 (Linietsky 2014b).

Julkaisun jälkeen Godotin kehitys on jatkunut ja se on vuosien varrella saanut useita merkittäviä päivityksiä. Godotin versio 2.0, joka julkaistiin vuonna 2016 keskittyi erityisesti editorin käytettävyyden parantamiseen. Käyttöliittymää uudistettiin yhteisön palautteen pohjalta, tehden siitä selkeämmän ja helpomman käyttää. Päivityksen mukana tuli myös parannuksia resurssienhallintaan ja se lisäsi automaattisen resurssien latauksen, joka nopeutti pelien käynnistymistä. (Linietsky 2016) Mozilla myös myönsi 22. Kesäkuuta 2016 Godotille 20000 dollarin arvoisen Open Source Support (MOSS) “Mission Partners” palkinnon käytettäväksi WebGL 2.0, WebSockets ja WebAssembly tuen lisäämiseksi (Mozilla 22.6.2016).

Tammikuussa 2018 julkaistussa 3.0-versiossa Godot-pelimoottoriin tuli lisää uusia ominaisuuksia ja parannuksia. Pelimoottorin 3.0-versiossa lisättiin esimerkiksi täysin uudistettu renderöintimoottori, joka perustui OpenGL ES 3.0 -rajapintaan mahdollistaen modernimmat visuaaliset efektit kuten dynaamiset varjot ja

parannetun valaistuksen, sekä fysiikkapohjaisen renderöinnin. Lisäksi päivitys toi mukanaan myös tuen mid- ja post-prosessoinnille joihin kuuluvat esimerkiksi sumu, syväterävyysalue ja SSAO. (Linietsky 2018). 3.0-versio lisäsi myös tuen C#-kielen käytölle Microsoftin Mono-ympäristön kautta. Tämä saatiin aikaan Microsoftin lahjoittaman 24000 dollarin ansiosta (Etcheverry 2017, Linietsky 2018).

Maaliskuussa 2019 julkaistussa Godot versiossa 3.1 tuli puolestaan merkittäviä parannuksia pelimoottorin omaan ohjelmointikieleen GDScriptiin. Päivityksiä tuli GDScriptin virheenkäsittelyyn, suorituskykyyn sekä editoriin lisättiin uusia ominaisuuksia, jotka helpottivat skriptien kirjoittamista. (Linietsky 2019) Kehitys jatkui myös aktiivisesti tämän jälkeen ja vuonna 2020 julkaistu versio 3.2 toi pelimoottoriin useita laadullisia parannuksia stabiliteetin ja suorituskyvyn kannalta.

Tammikuussa vuonna 2023 julkaistu Godot-pelimoottorin versio 4.0 oli laaja päivitys, joka sisälsi kokonaan uudelleen rakennetun renderöintimoottorin lisäten tuen Vulkan-ohjelmointirajapinnalle. Päivityksessä tuli myös parannuksia GDScriptin käytettävyyteen ja suorituskykyyn sekä fysiikka- ja animaatiotyökaluihin. Päivitys lisäsi myös monia muita uusia ominaisuuksia ja korjasi bugeja. (Godot Engine 2023) C#-kielen tukea myös parannettiin merkittävästi, sillä päivityksen mukana moottorin aikaisempi Mono-integraatio korvattiin uudella .NET 6 – pohjaisella C#-moduulilla. Tämä parantaa C#-sovellusten suorituskykyä ja helpottaa kehittämistä Visual Studiolla ja muilla .NET-yhteensopivilla työkaluilla. (Santos 2023)

Godot-pelimoottorin uusin versio kirjoitushetkellä on 4.4, jossa lisättiin integraatio Jolt-fysiikkamoottorille, interaktiivinen muokkaaminen peliä pelatessa ja uusia shader-toimintoja (Godot Engine 2025d). Pelimoottorin versio 4.5 on kehityksessä ja sen arvioitu julkaisuaika on vuoden 2025 syksy.

2.2.2 Arkkitehtuuri ja tekniset ominaisuudet

Godot-pelimoottori on rakennettu oliopohjaisen suunnittelun ympärille ja sen keskeisenä rakenteena toimii pelimoottorin joustava kohtaussysteemi sekä solmuhierarkia. Pelejä voidaan rakentaa järjestelemällä solmuja kohtauksiksi, jotka taas puolestaan voivat sisältää muita kohtauksia. Tämä mahdollistaa luotujen

kohtausten uudelleenkäytön ja auttaa modulaarisessa pelin suunnittelussa. Luotuja kohtauksia voidaan liittää toisiin kohtauksiin tai on myös mahdollista periä ominaisuuksia mistä tahansa kohtauksesta, joka mahdollistaa kokonaisuuksien luomisen käyttäen vain visuaalista editoria tai pelkkää koodia, sekä yhdistäen näitä molempia (Godot Engine 2025f).

Godotin kohtaussysteemiä voidaan verrata muiden pelimoottorien esivalmistettujen objektien (prefab) käyttämiseen. Prefabit ovat ennalta määriteltyjä objektipaketteja, joita on mahdollista käyttää pelissä useita kertoja helposti. Godotin kohtaussysteemi toimii vähän kuin sisäkkäin asetellut prefabit, koska sen avulla on mahdollista periä ja yhdistää kohtauksia toisiinsa. Esimerkkinä siitä miten Godotin kohtaussysteemi toimii, voidaan luoda “VälkkyväValo”-kohtaus, joka liitetään “RikkinäinenLamppu”-kohtaukseen. Tämän jälkeen on mahdollista luoda suurempi kohtaus kuten “Kaupunki”-kohtaus missä voi olla kymmeniä rikkinäisiä lamppuja ja kun alkuperäistä “VälkkyväValo”-kohtausta muokataan kaikki sen sisältävät rikkinäiset lamput “Kaupunki”-kohtauksessa myös päivittyvät automaattisesti (Godot Engine 2025f).

Solmut ovat toinen keskeinen osa Godotin arkkitehtuuria ja toimivat pelien tärkeimpinä rakennuspalikoina. Jokaisella solmulla on oma tehtävänsä ja yhdistämällä erilaisia solmuja voidaan monenlaisia kokonaisuuksia. Erityyppisiä solmuja on kymmeniä ja ne voivat esimerkiksi näyttää kuvia, toistaa ääniä, toimia kamerana tai määritellä pelimoottorin fysiikan törmäyspintoja. Kaikilla solmuilla on myös joukko niitä yhdistäviä ominaisuuksia. Näihin kuuluvat nimi, muokattavat ominaisuudet (properties), takaisinkutsun pohjalta päivittyminen joka ruudunpäivityksellä, mahdollisuus laajentaa solmuja uusilla ominaisuuksilla ja funktioilla, sekä niitä on mahdollista liittää toisiin solmuihin lapsisolmuina (Godot Engine 2025g).

Solmujen liittäminen toisiinsa on erityisen tärkeä ominaisuus, sillä solmuja yhdistämällä ne luovat puun, joka toimii tehokkaasti projektien organisoinnissa. Koska erilaisilla solmuilla on omat toimintonsa, niiden yhdistäminen mahdollistaa monipuolisen käyttäytymisen rakentamisen. Esimerkiksi pelihahmon voi luoda yhdistämällä solmuja kuten CharacterBody2D, Sprite2D, Camera2D ja CollisionShape2D. CharacterBody2D-solmu mahdollistaa liikkumisen ja

fysiikkatoimintoja, Sprite2D-solmuun voidaan liittää hahmon graafinen ulkonäkö, Camera2D-solmu toimii kamerana, joka seuraa pelaajaa ja CollisionShape2D-solmu määrittää pelihahmon törmäyspinnan koon ja muodon. Tällä solmujen yhdistelmällä voidaan jo luoda toimiva pohja pelihahmolle, ja sitä voidaan laajentaa lisäämällä solmuja tarpeen mukaan (Godot Engine 2025g).

Signaalit ovat Godot-pelimoottoriin sisäänrakennettu delegointimekanismi, joka antaa pelissä oleville solmuille kyvyn ilmoittaa toisilleen, kun joku tietty tapahtuma tapahtuu. Tällaisia tapahtumia voivat olla esimerkiksi napin painallus tai ajastimen päättyminen. Viestintä toimii ilman että solmut tarvitsevat suoraa viittausta toisiinsa, mikä vähentää koodin sidonnaisuutta ja auttaa sitä pysymään joustavampana (Godot Engine 2025h).

Signaaleja on mahdollista yhdistää joko käyttämällä Godotin editoria tai suoraan koodin kautta. Niitä voidaan myös määritellä itse skripteissä, mikä mahdollistaa esimerkiksi vihollisen tai pelaajahahmon “kuolema”-signaalin lähettämisen, kun hahmon elämäpisteet loppuvat. Godotin signaalit ovat pelimoottorin versio ohjelmoinnin tarkkailijamallista (observer pattern) ja niistä on Godot 4.0 -versiossa tullut “ensiluokkaisia tyyppejä” (first-class types), mikä parantaa niiden automaattista täydennystä ja tekee niistä vähemmän alttiita virheille (Godot Engine 2025h).

Godot-pelimoottorin editori tarjoaa käyttäjälle useita hyödyllisiä ominaisuuksia, jotka helpottavat pelien kehittämistä ja tekevät työnteosta tehokkaampaa. Yksi näistä työkaluista on kohtauspuun editori, jonka avulla on mahdollista hallita projektin kohtauksien solmuhierarkiaa visuaalisesti. Editorista löytyy myös sisäänrakennettu työkalu skriptien muokkaamiseen, joka tukee GDScript-koodin muokkaamista. GDScript-koodin debuggaukseen löytyy myös omat työkalunsa ja myös säikeiden virheenjäljitystä tuetaan (Godot Engine 2025i).

Editori sisältää visuaalisen profilointityökalun, jolla on mahdollista tarkkailla suoritusaikoja renderöintiputkiston eri pisteissä. Käytettävissä on myös työkaluja suorituskyvyn seurantaan ja myös kustomoitujen työkalujen luominen on mahdollista. Kehitystyön aikana editorissa on mahdollista hyödyntää reaaliaikaista skriptien uudelleenlatausta ja kohtauksia on myös mahdollista muokata

reaaliajassa, jolloin ne päivittyvät suoraan editorissa ja säilyvät projektin sulkemisen jälkeen (Godot Engine 2025i).

Editori sisältää myös toiminnallisuuden editorin kameran liikuttamiseen ja sen aiheuttamien muutoksien tarkasteluun reaaliaikaisesti käynnissä olevassa pelissä. Editorista löytyy lisäksi sisäänrakennettu offline-dokumentaatio luokkien viitteistä ja se on saatavilla monilla eri kielillä (Godot Engine 2025i).

Godotin editorin toiminnallisuutta on myös mahdollista laajentaa lisäosien avulla. Käyttäjän on mahdollista ladata valmiita lisäosia Godotin Asset Librarysta tai on myös mahdollista luoda omia lisäosia käyttämällä GDScriptiä. Projektinhallinnan kautta voidaan myös ladata kokonaisia projekteja Asset Libraryn kautta ja ottaa ne suoraan käyttöön (Godot Engine 2025i).

2.2.3 GDScript-ohjelmointikieli

GDScript on Godot-pelimoottorin oma, sisäänrakennettu ohjelmointikieli. GDScript on suunniteltu erityisesti pelien kehittämistä varten ja muistuttaa syntaksiltaan lähinnä Pythonia. Tämä tekee siitä helposti lähestyttävän erityisesti, jos on aikaisempaa kokemusta dynaamisesti tyypitettyjen kielten käytöstä (Godot Engine 2024).

GDScript kieli kehitettiin alun perin siksi, että pelimoottorin kehittäjät huomasivat käytössä olleen Lua-skriptikielen olevan monimutkaista ja hidasta käyttää heidän moottorillaan, vaatien suuren määrän koodia. Useimmilla skriptikielillä oli myös huono tuki säikeistykselle, eivätkä ne myöskään tukeneet luokkalaajennuksia, joten niiden mukauttaminen Godotin toimintatyyliin olisi hankalaa. Lisää syitä GDScriptin luonnille olivat muiden kielten huono yhteensopivuus C++ -ohjelmointikielen kanssa, pelinkehitykselle tärkeiden tietotyyppien puute kuten vektorit, roskienkeräyksen aiheuttamat pysäytykset ja suuri muistinkäyttö, sekä hankaluus koodieditorin ominaisuuksien kuten reaaliaikaisen muokkauksen ja automaattisen täydennyksen toteuttamisessa (Godot Engine 2024).

Näiden haasteiden ratkaisua varten GDScript on suunniteltu olemaan tiiviisti

integroitu Godotin sisäiseen toimintaan. Tämä mahdollistaa yksinkertaisemman koodin luomisen ja nopeamman kehityssyklin, sekä pienemmän virhealttiuden ja paremman pelien suorituskyvyn. Oman ohjelmointikielen luominen myös mahdollistaa sen, että kehittäjät pystyvät keskittymään moottorin parantamiseen sen sijaan että aikaa kuluisi pienten inkrementaalisten ominaisuuksien toiminnan varmistamiseen useilla eri ohjelmointikielillä (Godot Engine 2024).

Dynaamisesti tyypitettynä kielenä GDScriptin pääetuja ovat ohjelmointikielen käytön helppo aloittaminen, koodia on helppo iteroida, sekä sen yhteensopivuus Godotin sisäisten ominaisuuksien, kuten vektorityyppien ja solmupohjaisen arkkitehtuurin kanssa. GDScriptillä kirjoitettu koodi ei myöskään vaadi kääntämistä tehden kehitystyöstä nopeaa ja erilaisia asioita voidaan testata välittömästi. Kieli myös tukee asioita kuten luokkien pereyttämistä, ankkatyypitystä sekä polymorfismia (Godot Engine 2025e).

```gdscript
# Member variables.
var a = 5
var s = "Hello"
var arr = [1, 2, 3]
var dict = {"key": "value", 2: 3}
var other_dict = {key = "value", other_key = 2}
var typed_var: int
var inferred_type := "String"

# Constants.
const ANSWER = 42
const THE_NAME = "Charly"

# Enums.
enum {UNIT_NEUTRAL, UNIT_ENEMY, UNIT_ALLY}
enum Named {THING_1, THING_2, ANOTHER_THING = -1}

# Built-in vector types.
var v2 = Vector2(1, 2)
var v3 = Vector3(1, 2, 3)


# Functions.
func some_function(param1, param2, param3):
    const local_const = 5

    if param1 < local_const:
        print(param1)
    elif param2 > 5:
        print(param2)
    else:
        print("Fail!")

    for i in range(20):
        print(i)

    while param2 != 0:
        param2 -= 1
```

KUVA 1. Kuva GDScriptin perussyntaksista.

Vaikka GDScript on tehokas ohjelmointikieli pelien kehittämistä varten, se ei sovellu kaikkiin tilanteisiin ja dynaamisesti tyypitettynä kielenä sillä on myös omia

heikkouksia. Näihin kuuluvat heikompi suorituskyky kuin staattisesti tyypitetyillä ohjelmointikielillä, koodi on hankalampaa refaktoroida, joidenkin virheiden ilmeneminen vasta koodia suorittaessa, sekä vähemmän joustavuutta koodin automaattisessa täydennyksessä. Tarvittaessa esimerkiksi laskennallisesti raskaat tehtävät on mahdollista toteuttaa C++-lisäosien avulla suorituskyvyn parantamiseksi (Godot Engine 2025e).

2.3 Unreal Engine -pelimoottori

Unreal Engine luotiin Epic Gamesin perustajan Tom Sweeneyn johdolla vuonna 1998. Sen ensimmäinen versio julkaistiin ensimmäisen persoonan ammuntapeliin Unrealiin. Unreal Engine oli suunniteltu mahdollistamaan pelien kehittämisen edistyneimmillä grafiikoilla ja edistyneemmällä fysiikkasimulaatiolla. Näiden ominaisuuksien ansiosta, se nousi suureen suosioon ja sitä käytettiin monessa suositussa pelissä, kuten Unreal Tournamentissa, Gears of Warsissa ja Infinity Bladessa. (Gbo Seo Solutions 29.4.2023.)

Unreal Engine on vuosien varrella saanut useita versio päivityksiä. Unreal Enginen uusin versio on versio 5.3 ja se julkaistiin syyskuussa 2023. (Unreal Engine 2023.)

Unreal Engine tarjoaa useita eri ominaisuuksia. Se tarjoaa edistyneet fysiikka simulaatiota, joiden avulla kehittäjät voivat luoda realistisia simulaatioita oikean elämän fysiikasta, kuten painovoimasta ja törmäyksistä. Unreal Engine on myös tunnettu sen korkeatasoisista grafiikoista ja renderöintiominaisuuksista. Se hyödyntää edistyneitä renderöinti tekniikoita, kuten valaistusta ja dynaamista valaistusta luodakseen henkeäsalpaavia visuaalisia tehosteita. Unreal Engine tukee myös eri alustoilla samaan aikaa pelaamista ja monin pelaamista sekä sillä on vahva virtuaalitodellisuutta hyödyntävien pelien kehittämistä.

2.4 Unity-pelimoottori

Unity on vuonna 2004 Unity Techologiesin perustama pelimoottori. Unityn tarina alkoi vuonna 2002 OpenGL foorumeilla, kun Nicholas Francis julkaisi ilmoituksen, jossa hän etsi yhteistyökumppaneita avoimen lähdekoodin grafiikkatyökalun kehittämiseen hänen kaltaisilleen Mac-pohjaisille pelinkehittäjille. Tähän tarjoukseen tarttui Joachim Ante ja David Helgason. Unityn ensimmäinen versio julkaistiin vuonna 2005 nimellä Unity 1.0 ja sillä pystyttiin aluksi julkaisemaan pelejä vain Mac OS X:lle. Vuotta myöhemmin Unity 1.0 sai kuitenkin päivityksen, joka mahdollisti pelien julkaisun myös Windows ja verkkopohjaisille peleille. (Cohen-Peckham 17.10.2019.)

Koska Unityn tiimi oli suureksi osaksi Mac faneja, IPhonen julkaisu vuonna 2007 sai heidät innostumaan sen suurimmaksi osaksi avoimesta tuesta kolmannen osapuolen sovelluksille. Unity lisäsi nopeasti tuen pelieni viennille iOS laitteille, vaikka suuret studiot ja muut pelimoottorit jättivät mobiililaitteet huomitta sen heikkojen teknisten tietojen vuoksi. Indie-pelikehittäjät omaksuivat Unityn luodakseen mobiilipelejä Applen App store:en. Helgason ei osannut arvata, kuinka nopeasti mobiilipeli markkinat voisivat kasvaa. Unity mahdollisti viennin Android laitteille maaliskuussa 2012 pian Unity 3.0 julkaisun jälkeen. Tämä kasvatti sen markkinaosuutta mobiilipelaamisessa kokonaisuudessaan. (Cohen-Peckham 17.10.2019.)

Vuonna 2012 mobiili pelaaminen kattoi 18 % maailmanlaajuisesta pelaamisesta ja vuonna 2018 se nousi 51 %. (Cohen-Peckham 17.10.2019.)

Nykyään Unity tarjoaa tuen yli 20 eri alustalle. Näihin alustoihin kuuluu työpöytä alustat kuten Windows, MAC, Linux ja konsolit kuten PS5, PS4, Xbox One ja Nintendo Switch. (Unity 27.5.2025.)

2.5 Pelimoottorin valinta

Pelin suunnittelemisen alkuvaiheessa meidän täytyi päättää, mitä pelimoottoria haluaisimme käyttää pelin luomiseen. Jokaisella pelimoottorilla on omat vahvuudet ja heikkoudet.

Esimerkiksi Unityn vahvuus on sen suosio ja ennestään laaja käyttö Indie-pelien kehityksessä. Siihen on helppo löytää erilaisia ohjeita ja Unity Asset Storen laaja tarjonta helpottaa ja nopeuttaa pelien kehitystä. Unity tukee myös hyvin kaksi- ja kolmeulotteisten pelien luontia. Esimerkkejä suosituista Unityllä tehdyistä peleistä ovat Among us ja Escape from Tarkov.

KUVA 2. Kuva pelistä Escape from Tarkov.

Unreal Enginen vahvuus on sen edistyneet fysiikkasimulaatiot sekä sen graafiset ominaisuudet, joilla pystyy luomaan visuaalisesti näyttäviä kolmannen ulottuvuuden pelejä, kuten Fortnite ja vastajulkaistu Clair Obscur: Expedition 33.

KUVA 3. Kuva pelin Clair Obscur: Expedition 33 taistelukohtauksesta.

Godot puolestaan keskittyy enemmän kaksiulotteisten pelien kehitykseen. Godot on myös huomattavasti kevyempi, eikä se vaadi erillisiä asennuksia. Sen heikkouksiin kuuluu sen vielä muihin mainittuihin pelimoottoreihin verrattuna heikommat kolmiulotteisten pelien kehittämistä tukevat työkalut. Esimerkkejä Godotilla tehdyistä peleistä on Backpack Battles ja Buckshot Roulette.

KUVA 4. Kuva pelistä Backpack Battles.

Koska suunnittelimme pelin olevan kaksiulotteinen ja halusimme tutustua lähemmin Godotin käyttöön ja sen tarjoamiin ominaisuuksiin, valinta Godotin ja Unityn välillä oli helppo.

3 PELIN SUUNNITTELU

Pelikehityksen voi jakaa seitsemään osaan: Pelin suunnitteluun, Esituotantoon, Prototyypin tekemiseen, Tuotantoon, Testaamiseen, Julkaisuun ja Julkaisun jälkeiseen tukeen. (Ixiegaming s.a.)

Ensimmäinen askel pelinkehityksessä on suunnittelu. Se luo pohjan pelin idealle ja toteutukselle. Pelinkehityksen alkuvaiheissa kannattaa olla tiedossa jo pelin genre, sillä sen ympärille on helpointa rakentaa. Suunnittelun aikana kehitystiimi määrittää pelin konseptin, ja määrittää projektin aikajanan ja budjetin. Suunnittelun aikana vastataan kysymyksiin kuten pelin genreen, minkä ulotteinen peli on, mikä pelin visuaalinen tyyli on ja mitä pelattavuuteen liittyviä mekaanikkoja pelissä on. Nämä ideat muuttuvat useasti suunnitteluvaiheen aikana ja joskus lopullinen tuote voi näyttää joltain aivan muulta kuin mitä alun perin ideoi. Suunnittelun aikana tehdään peli suunnittelu dokumentti, joka palvelee suunnitelmana pelille. Dokumentissa dokumentoidaan asioita kuten pelin ydinsilmukka, pelin keskeisimmät mekaniikat, pelin tarina ja pelin maailma ja sen rakentaminen. Dokumentissa käydään myös läpi pelin visuaalinen ja audiovisuaalinen tyyli ja käyttöliittymä. (Ixiegaming s.a.)

3.1 Pelin ominaisuudet

Videopeleistä löytyy monenlaisia erilaisia ominaisuuksia. Nämä ominaisuudet voidaan jakaa eri kategorioihin kuten: Pelin perusominaisuudet, pelaajankokemus ominaisuudet, tekniset ominaisuudet, moninpeli ominaisuudet ja metaominaisuudet. Perusominaisuuksiin kuluu pelin mekaniikat kuten hyppiminen ja liikkuminen ja pelin eri tasot tai maailmat. Pelaajankokemus ominaisuuksiin kuuluu asiat kuten pelin tarina ja maailmanrakennus, npc-hahmot ja pelattavan hahmon kustomointi. Teknisiä ominaisuuksia peleissä on asiat kuten grafiikat, musiikki ja käyttöliittymä. Metaominaisuuksiin kuuluu pelaajan pitkäaikaista kiinnostusta edistävät asiat kuten saavutukset, tulostaulukot ja nykyään enemmän yleistyneet pelien sisäiset kausittain vaihtuvat sisältöpaketit, jotka voivat antaa pelaajalle

uusia tehtäviä, esineitä tai kosmeettista sisältöä kannustaen säännölliseen pelaamiseen. (Rabin 2009, 50.)

3.1.1 Pelin idea ja rakenne

Idea pelille lähti halusta rakentaa nopeatempoinen mutta kuitenkin suhteellisen yksinkertainen peli, joka onnistuisi herättämään pelaajan mielenkiinnon. Pelin genre olikin siis helppo valita, sillä bullet hell -genre täyttää juuri nämä kriteerit tarjoamalla nopeatempoisen pelityylin, sekä useasti visuaalisesti näyttäviä vihollishyökkäyksiä.

Pelille tehtiin erillinen suunnitteludokumentti, johon listattiin pelin vaatimuksia ja ominaisuuksia. Näihin vaatimuksiin kuului taso, jossa peliä tultaisiin pelaamaan, ja pelaajan ohjaama hahmo, joka pystyy hyökkäämään hiiren osoittimen suuntaan sekä väistämään tai torjumaan vihollisen hyökkäyksiä. Vihollisia tulisi olla kahdesta neljään ja kaikilla vihollisilla tulisi olla erilaiset hyökkäysmallit. Pelaajalla ja vihollisella tulisi myös olla selvästi näkyvät elämäpisteet pelin käyttöliittymässä. Pelissä tulisi myös olla päävalikko, josta peli käynnistyy sekä päätösruutu, kun pelaaja häviää pelin. Lisäksi dokumenttiin lisättiin ominaisuuksia, jotka tehtäisiin, jos niille riittäisi aika. Näihin ominaisuuksiin kuului pisteytyssysteemi, jossa pelaajalle annettaisiin pisteitä suoriutumisen mukaan. Pelaaja saisi pisteitä riippuen siitä, kuinka nopeasti hän päihittäisi vihollisen ja kuinka monta kertaa ottaisi vahinkoa tasoa päihittäessä. Peliin suunniteltiin myös päivitysominaisuus, jolla pelaaja pystyisi päivittämään hahmoa. Ostamalla päivityksiä pelaaja saisi esimerkiksi enemmän elämäpisteitä tai hänen torjunta- tai väistämiskykynsä kestäisi pidempään, veisi hahmoa pidemmälle tai antaisi pidemmän kuolemattomuus ajan.

Seuraavaksi täytyi päättää pelin ulottuvuus. Koska pelistä haluttiin tehdä suhteellisen yksikertainen, päädyttiin pelistä tekemään kaksiulotteinen ja ylhäältäpäin kuvattu. Päätöstä 2D-ulottuvuuden valintaan helpotti myös Godot- pelimoottorin 2D-pelien kehittämistä tukevat työkalut kuten laattaeditori, joka mahdollistaa kaksiulotteisen maailman luomisen. Muita hyödyllisiä 2D työkaluja on 2D fysiikka rungot ja 2D animoidut hahmokuvat.

Peli ottaa paljon inspiraatiota muista saman genren peleistä kuten Touhou Projectista, Furista, sekä Undertalesta (Touhou Project s.a.; Furi s.a.; Undertale s.a.).

3.1.2 Pelattavan hahmon ominaisuudet

Pelin pelattava hahmo haluttiin pitää ominaisuuksiltaan yksinkertaisena. Tavoitteena oli suunnitella hahmo, jonka ohjaaminen olisi pelaajille helppoa ja että liikkuminen olisi sulavaa sopien pelin nopeatempoiseen ja tarkkaan vihollishyökkäysten väistelyyn. Suunnittelussa päädyttiin siihen, että pelaajalle annettaisiin kahdeksansuuntainen liikkuminen, mikä mahdollistaa vapaan liikkumisen kaikkiin ilmansuuntiin helpottaen vaikeiden hyökkäyskuvioiden väistämistä.

Suunnittelun aikana päädyttiin myös siihen, että pelaajalle luodaan jonkinlainen hyökkäystoiminto, jotta pelaaja pystyy vahingoittamaan vihollisia ja päihittämään heidät. Vaihtoehtona pelaajan hyökkäykselle pohdittiin myös yksinkertaisempaa ratkaisua, jossa kohtaaminen jokaisen vihollisen kanssa olisi ollut ajastettu, mutta pelin kiinnostavuuden kannalta päädyimme nykyiseen ratkaisuun.

Hahmon törmäyslaatikon koko ja hahmon nopeus muuttuivat myös useasti suunnitteluvaiheessa. Jos törmäyslaatikon koko oli liian iso tai jos pelaajan nopeus oli liian hidas, vihollisen ammusten väistämisestä tuli liian vaikeaa, joka johti siihen, että peli oli hyvin vaikea tai lähes mahdoton.

Hahmolle suunniteltiin myös erilaisia elämäpiste järjestelmiä. Aluksi hahmolla oli vain yksi elämä, mutta nopeasti testauksen aika todettiin, että peli olisi hyvin haastava vain yhdellä elämällä. Tämän jälkeen hahmolle annettiin 5 elämäpistettä, mutta se teki pelistä taas hyvin helpon. Lopulta hahmolle päädyttiin antamaan 3 elämäpistettä.

Lisäksi hahmoa suunnitellessa harkittiin, että hahmolle annettaisiin joku aktiivinen toiminto kuten väistöliike tai mahdollisuus torjua vihollisen projektiileja. Väistöliikkeen lisääminen antaisi pelaajalle mahdollisuuden tarvittaessa muuttua kuolemattomaksi ja liikkua nopeasti tietyn pituisen matkan, mikä voisi parantaa pelaajan hallittavuutta vaativissa tilanteissa ja parantaa pelin rytmiä.

3.1.3 Viholliset

Vihollisten suunnitteleminen on keskeinen osa pelimekaniikkojen ja pelin kokemuksen luomista. Suunnittelemisessa ei ole kyse ainoistaan visuaalisesta ilmeestä tai satunnaisista muutoksista, vaan suurelta osin siitä, miten vihollinen vaikuttaa pelaajan päätöksentekoon ja toimintaan. Hyvin suunnitellut viholliset antavat pelaajalle merkityksellisen haasteen, jossa pelaaja hyötyy tilanteen havainnoinnista ja niiden pohjalta tehdyistä reaktioista sen sijaan, että haaste perustuisi täysin sattuman varaan. Bullet hell -peleissä ruudulla on yleensä suuri määrä väistettäviä projektiileja, tehden vihollisten hyökkäyksien suunnittelusta tärkeää. Parhaimmillaan kuviot olisivat helposti luettavissa, mutta samaan aikaan myös vaihtelevia ja haastavia, pitäen pelaajan aktiivisena tuntumatta epäreilulta (Kraj 2020).

Koska peli sijoittuu fantasia-aiheiseen maailmaan, haluttiin vihollisten olevan mytologiaan tai fantasiatarinoihin pohjautuvia olentoja kuten lohikäärme tai minotauri. Jokaisesta vihollisesta haluttiin luoda omalla tavallaan muistettava kokemus, joten suunnitteluvaiheessa haluttiin luoda jokaiselle viholliselle omat hyökkäykset ja käyttäytyminen. Tämän avulla voitaisiin luoda tilanteita, jossa pelaaja joutuisi vihollisen mukaan vaihtamaan heidän pelityyliään soveltuvammaksi vihollisen tekemiin asioihin. Vihollisten erilaisia hyökkäyksiä suunnitellessa käytiin läpi erilaisten bullet hell –genren pelien päävihollisten hyökkäyksiä ja kykyjä saadaksemme paremman käsityksen siitä, minkälaisia hyökkäyskuvioita voitaisiin käyttää tai mitä erikoisominaisuuksia vastustajalla voisi olla. Vihollisten suunnittelun aikana yksi ongelma oli löytää tasapaino vihollisten hyökkäysten nopeudelle ja projektiilien yhdenaikaiselle määrälle pelaajan ruudulla. Liian hitaat hyökkäykset tai vähäinen määrä väistettäviä asioita voivat helposti tehdä pelistä pelaajalle tylsän, kun taas puolestaan liian nopeasti liikkuvat projektiilit tai niiden massiivinen määrä voi vaikuttaa pelaajalle mahdottomalta navigoida. Sopiva tasapaino vihollisten vaikeuden ja kykyjen suunnittelussa tulisi siis olemaan kriittinen osa pelikokemuksen kannalta.

3.1.4 Ääniefektit ja musiikki

Pelin tyypistä tai alustasta riippumatta ääni on olennainen osa pelikokemusta. Peleissä äänet voidaan jakaa kolmeen laajaan kategoriaan: Ääniefekteihin, Ääniraitoihin ja musiikkiin. Lähes jokaisessa videopelissä on vähintään vähän jokaista näistä elementeistä. Joissakin peligenreissä esimerkiksi esiintyy enemmän dialogia, kun taas toisessa ei juuri ollenkaan. Useimmissa peleissä on ainakin yhdenlainen ääniefekti, vaikka tähänkin on poikkeuksia, kuten esimerkiksi 1960-luvun lopulla ja 1970-luvun alussa tehdyt pelit, jotka olivat täysin äänettömiä. (Horowitz & Looney 2014, 16.)

Peliin ei haluttu lisätä liian paljoa erilaisia ääniefektejä, koska niiden ei haluttu häiritsevän pelaamista. Pelin ampumisäänelle annettiin satunnaisesti vaihteleva äänenkorkeus, joka auttoi siinä, että pelaajan korvat eivät väsyisi jatkuvaan samaan ääneen.

Pelin musiikkien haluttiin olevan nopea tempoiset, sillä se vastaisi pelin nopeaa tempoa ja pitäisi pelaajan rytmissä. Musiikin lopulliseksi genreksi valittiin metalli.

3.1.5 Aloitusruutu ja UI-elementit

UI-elementit ovat tärkeä osa peliä, joten selkeän ja helposti navigoitavan käyttöliittymän tekeminen on tärkeää. Valikot ovat ensimmäinen ja viimeinen asia mitä pelaaja näkee, ja minkä kanssa he ovat vuorovaikutuksessa. Hyvin tehty käyttöliittymä luo pysyvän vaikutuksen. Esimerkkejä hyvin tehdyistä päävalikosta on esimerkiksi Halo pelisarja, joissa pelin päävalikko pysyy aina lähestulkoon samanlaisena. (Cherkaoui 17.12.2024.)

KUVA 5. Kuva pelin Halo 3 päävalikosta.

Vaikka moni hyvin tehty valikko on monesti ulkonäöltään pelkistetty, se ei ole vaatimus hyvälle valikolle. Tästä hyvä esimerkki on Persona 5 pelin valikko, jossa valikko on hyvin tyylitelty, mutta on silti hyvin käytännöllinen. Tämmöisen valikon tekeminen ei kuitenkaan ole helppoa ja se vaatii useita eri ohjelmia toimiakseen oikein. (Webster 7.10.2024)

KUVA 6. Kuva pelin Persona 5 taukovalikosta.

4 PELIN KEHITYS

Kun seurataan luovaa prosessia, aloitetaan aina jostain tietystä asiasta, tai tehdään aina samat vaiheet samassa järjestyksessä. Tehtäisiinpä sen tarkoituksella tai ei, noudatetaan silti kaikki erilaisia lähestymistapoja. Tämä sama pätee pelisuunnitteluun. Pelisuunnittelussakin on erilaisia lähestymistapoja. Ensimmäisenä lähestymistapa on iteratiivinen suunnittelu. Se on yleisin lähestymistapa kaikilla luovilla alueilla ja sitä käytetään ihan kuin luonnostaan. Se alkaa ensin olettamalla, jonka jälkeen kokeillaan ja testataan, miten se toimii. Viimeisimmästä iteraatiosta saatua tietoa hyödynnetään ja iteraatioprosessi aloitetaan uudelleen. Iteraatiot ovat tehokkaita, koska epäonnistuminen tapahtuu aina jossain vaiheessa, ellei aina. Tämä prosessi kuitenkin antaa mahdollisuuden luoda paremman tuotteen ja se myös antaa kokemusta ja hioo visiota ja tietämystä. Mitä enemmän kokemusta on, sitä vähemmän tarvitsee tehdä iteraatiota. Tämän lähestymistavan heikkous on se, että monet suunnittelijat ja etenkin opiskelijat sekä vastavalmistuneet usein aliarvioivat ensimmäisen vaiheen tärkeyttä, ja keskittyvät pelkästään kokeilemiseen ja testaamiseen. Siitä seuraa paljon ongelmia, jotka olisi voitu ratkaista aikaisemmin. Lisäksi kokemuksen puutteen takia kehittäjän mututuntuma voi usein olla väärässä. (Querné 2023.)

Maailmalla tunnettu peliyritys Ubisoft kehitti ratkaisun iteratiiviseen suunnittelun ongelmaan. Rationaalinen pelisuunnittelu keskittyy pelikokemuksen ja pelijärjestelmien purkamiseen ratkaistakseen suurimman osan suunnittelun ongelmista ennen sen tekemistä ja testaamista. Se luotiin analysoimalla eri tieteenaloja. (Querné 2023.)

Rationaalinen pelisuunnittelu koostuu työkalujen ja prosessien käytöstä siten, että ne kuroisivat kuilun umpeen pelijärjestelmien ja pelaajan välillä. Työkalujen, kuten vaikeusmatriisin käyttö eri taitojen ja atomiparametrien tunnistamiseksi pelimekaanikkojen vaikeustason säätämistä varten tai varieteettimatriisin käyttö erilaisten tasoskenaarioiden luontiin. Myös OCR-silmukan käyttö pelaajan pelinsisäisen motivaation tunnistamiseksi tai palautteen kartoitus auttavat oikein

tehtynä ratkaisemaan suurimman osan “helpoista” ongelmista jo ennen suunnittelun toteutusta. (Querné 2023.)

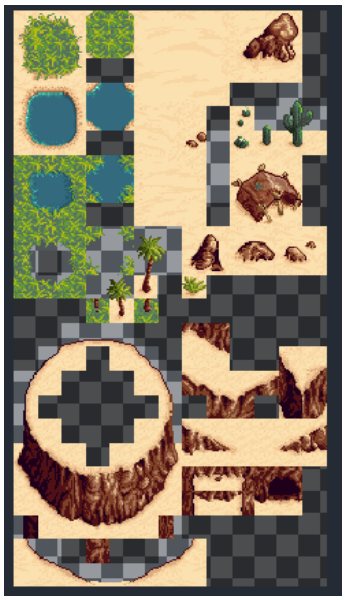
Tällä lähestymistavalla on kaksi eri haittapuolta. Ensimmäinen on se, että jotkin suunnittelijat ajattelevat, että rationaalinen pelisuunnittelu on ikään kuin taikakaava. Tämä on kuitenkin väärin ja muita lähestymistapoja tarvitaan silti, jotta rationaalinen suunnittelu olisi tehokasta, koska yksinään se on vain pussi työkaluja pelien analysointiin. Toisena haittapuolena on se, että rationaaliselle pelisuunnittelulle ei ole suurta määrää resursseja. Täytyy kaivaa todella syvälle löytääkseen tietoa siitä, tai etsiä henkilö, joka seuraa Ubisoftin akateemista ohjelmaa, koska muuten löytyy resursseja vain matemaattisten lähestymistapojen käytöstä pelisuunnittelussa, jotka eivät ole rationaalisen suunnittelun ydin. (Querné 2023.)

Godot oli loistava alusta hyödyntää iteratiivista suunnittelua. Sen nopeat rakennusajat auttoivat nopeaan testaamiseen ja sen pohjalta uudelleen suunnitteluun. Kun jokin asia ei toiminut odotetulla tavalla, oli helppoa ja nopeea käydä muokkaamassa koodia, ja testata sitä uudelleen.

Opinnäytetyön osana suunnittelun lisäksi myös kehitettiin videopeli. Suunnittelun jälkeen videopeliä lähdettiin kehittämään ensin rakentamalla areena, jossa pelaaja ja vastustajat taistelisivat ja luomalla hahmon, jota pelaaja pystyy liikuttamaan. Tämän jälkeen peliin luotiin useampi erilainen vihollinen, joille annettiin erilaisia hyökkäyksiä. Lopuksi pelille tehtiin käyttöliittymä ja peliin lisättiin erilaisia ääniefektejä sekä musiikit.

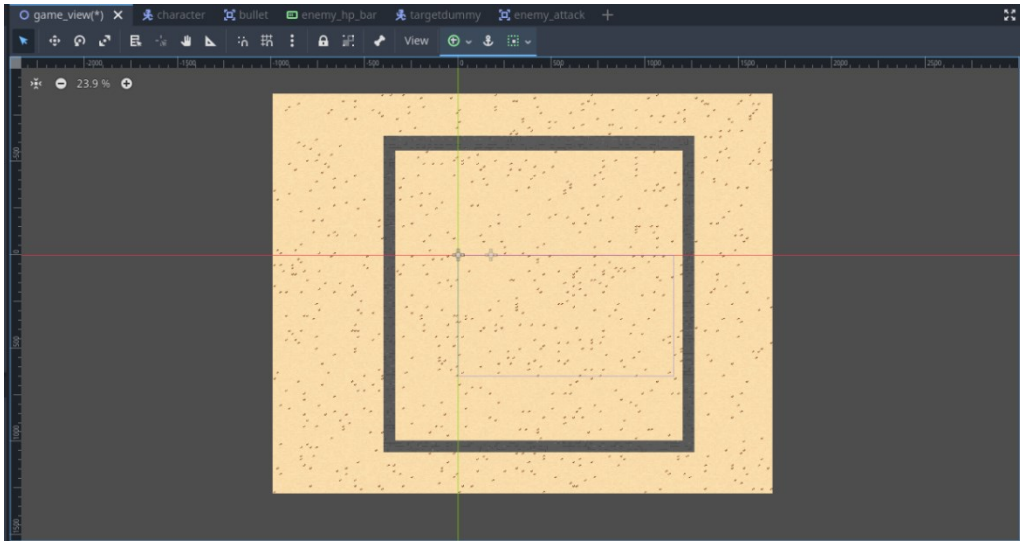
4.1 Tason luonti

Pelin tason luonti alkoi ensimmäiseksi etsimällä peliin sopivat tekstuurit. Koska pelin teemana oli fantasiamaailmaan sijoittuva colosseum, valittiin maan tekstuuriksi hiekkainen ja kivinen tekstuuri.



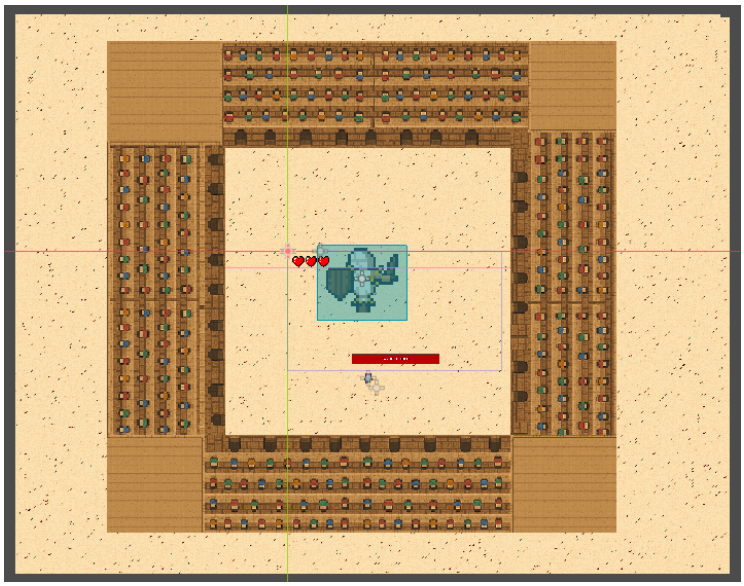
KUVA 7. Kuva peliin valitusta tekstuurista

Tason muoto muuttui myös useasti sen luonti vaiheessa. Ensimmäisessä versiossa taso oli timantin muotoinen, mutta se tuntui olevan liian ahdas. Toisessa versiossa taso oli jo enemmän avarampi ja kulmat olivat pyöristettyjä. Tämän areenan ongelmana oli se, että hahmo saattoi jäädä kulmaan jumiin väistäessään vihollisen ammuksia. Lopulta tason muodoksi valittiin neliö, sillä se ei häiritse pelissä liikkumista tai muita toimintoja.



KUVA 8. Ensimmäinen versio pelin tasosta

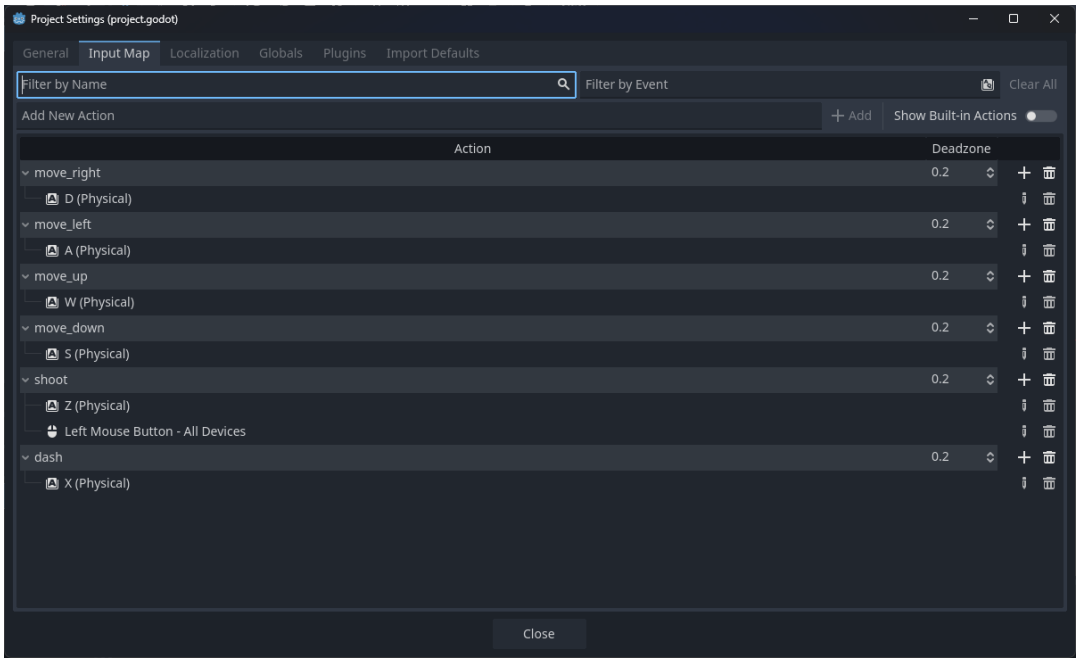
Myöhemmin pelin tasolle lisättiin katsojia ja tason seinistä tehtiin enemmän colosseum teemaan sopivammat.



Kuva 9. Kuva pelin lopullisesta tasosta

4.2 Pelattavan hahmon kontrollit

Pelattavan hahmon kontrollit tehtiin käyttämällä Godotin input mappingiä. Input mapping mahdollisti hahmon ohjaamisen näppäimistöä ja hiirtä käyttäen määräämällä jokaiselle toiminnolle oman näppäimen.



KUVA 10. Kuvassa näkyy Godotin input map ruutu ja hahmon ohjaamiselle annetut näppäimet.

Sen jälkeen, kun hahmon eri kontrolleille oli annettu nimi ja sitä vastaava näppäin, täytyi niitä kutsua koodista.

```
func get_input():
    var input_direction = Input.get_vector("move_left", "move_right", "move_up", "move_down")
    velocity = input_direction * speed
```

KUVA 11. Kuva hahmon liikkumista ohjaavasta koodista.

Hahmon liikkumiselle tehtiin myös kävelyanimaatiot neljään eri suuntaan, joita ohjattiin koodilla.

```
func update_animation():
    if velocity.length() > 0:
        last_direction = velocity.normalized()
        anim.play()

        if abs(velocity.x) > abs(velocity.y):
            anim.animation = "walk"
            anim.flip_h = velocity.x < 0
            anim.flip_v = false
        elif velocity.y < 0:
            anim.animation = "up"
            anim.flip_h = false
            anim.flip_v = false
        else:
            anim.animation = "down"
            anim.flip_h = false
            anim.flip_v = false
    else:
        anim.stop()
```

Kuva 12. Kuva hahmon animaatioita ohjaavasta koodista.

Hahmolle annettiin myös kyky ampua hiiren painalluksella sekä väistämisliike X-näppäintä painamalla. Väistämisliike teki pelaajasta hetkellisesti kuolemattoman. Tämän avulla pelaaja kykeni helpommin väistämään vihollisen luoteja.

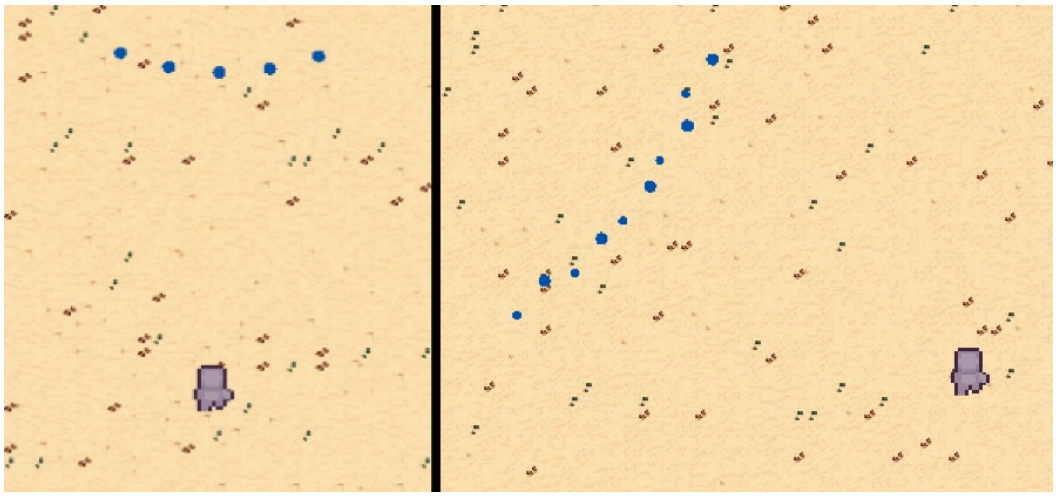
4.3 Vihollisten luonti

Pelin ensimmäiseen demoversioon luotiin yksinkertainen vihollinen, jolla on kyky valita satunnaisesti yksi kolmesta sille luodusta hyökkäyskuviosta. Vihollinen ei pysty liikkumaan, vaan sen roolina on toimia johdatuksena pelin mekaniikkoihin ja antaa pelaajalle mahdollisuus harjoitella vihollisten hyökkäysten väistämistä. Vihollisen hyökkäykset koostuvat liikkuvasta spiraalikuviosta, pelaajaa kohti ammuttavasta hyökkäyksestä sekä ympyränmuotoisesta kuviosta. Viholliselle lisättiin elämäpisteet ja kyky ottaa vahinkoa pelaajan luodeista. Kun elämäpisteet loppuvat, vihollisen hyökkäykset pysähtyvät ja se poistetaan pelin areenalta. Tämä vihollinen toimii pohjana uusien vihollisten ja ominaisuuksien kehittämiselle.

Ensimmäisen demoversion jälkeen vihollisten hyökkäyslogiikka siirrettiin omaan kohtaukseensa, joka sisältää skriptin ja AttackController-solmun johon skripti liitettiin. Tämä mahdollistaa hyökkäysten hallitsemisen yhdestä paikasta ja tekee luoduista hyökkäyksistä helposti muokattavia ja uudelleenkäytettäviä eri vihollisilla. AttackController-solmu voidaan liittää vihollisten lapseksi, jolloin se hoitaa automaattisesti hyökkäysten valinnan ja suorittamisen vihollisen skriptissä määriteltyjen parametrien mukaan.

Toiseen demoversioon luotiin pelin seuraava vihollinen, joka on minotauri. Minotaur-vihollisen vaikeustasoa haluttiin nostaa aiempaan viholliseen verrattuna, joten sen hyökkäyksistä tehtiin monimutkaisempia ja hankalampia väistää. Viholliselle luotiin myös uusi pelimekaniikka tekemällä yksinkertainen tilakone, joka hallitsee vihollisen käytöstä sen jäljellä olevien elämäpisteiden mukaan. Taistelu alkaa siitä, että vihollinen on normaalissa tilassa, jossa se pysyy paikallaan ja sen hyökkäykset määräytyvät satunnaisesti AttackController-solmun avulla. Kun sen elämäpisteet laskevat määritellyn rajan alapuolelle, vihollinen siirtyy DASH-tilaan kutsumalla start_dash_phase()-funktiota. DASH-tila keskeyttää satunnaiset hyökkäykset ja vihollinen liikkuu nopeasti ennalta määritettyjen pisteiden välillä. Liikkumisen aikana se kutsuu DASH-tilaa varten luotua hyökkäyskuviota ja vaiheen päätyttyä se palaa normaaliin tilaan. Minotaur-vihollisen roolina on olla pelin ensimmäinen vihollinen, jolla on hankalampia bullet hell -elementtejä ja jonka käyttäytyminen muuttuu merkittävästi taistelun aikana.

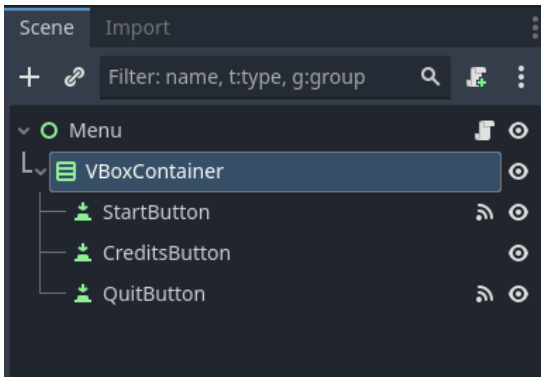
Kolmannesta vihollisesta tehtiin monivaiheinen taistelu, jonka edetessä pelaajalle esiteltäisiin jatkuvasti uusia pelimekaniikkoja. Ensimmäisen vaiheen aikana vihollinen pyrkii seuraamaan ja liikkumaan pelaajan ympärillä, samanaikaisesti ampumaan pelaajaa kohti hyökkäyksillä, joita pelaaja on jo aikaisemmin kokenut. Toisen vaiheen aikana taistelu pysyy suurilta osin samana, mutta vihollinen muuttaa ammukset olemaan kaksi pisteen ympärillä pyörivää luotia aikaisemmin käytetyn yksinkertaisen luodin sijaan. Kolmannessa vaiheessa vihollinen muuttuu tilapäisesti kuolemattomaksi ja siirtyy peliareenan keskelle, jossa se käyttää areenan ympäri pyörivää hyökkäystä ja areenalle syntyy pieniä hämähäkkivihollisia, jotka seuraavat pelaajaa. Viimeinen vaihe toimii loppuhuipennuksena, jossa vihollinen saa osan elämäpisteistään takaisin ja aloittaa hyökkäyksen, jossa pelaajan pitää liikkua luotien välissä olevan, kapenevan välin mukana.



KUVA 13. Vasemmalla normaali haulikkohyökkäys ja oikealla kolmannen vastustajan yhden pisteen ympärillä pyörivä hyökkäys

4.4 Käyttöliittymän luonti

Pelin valikot haluttiin pitää selkeinä ja minimalistisina. Koska peli on nopea tempoinen ja suhteellisen haastava, pyrittiin luomaan valikot, jotka mahdollistivat pelaajan nopea pääsyn takaisin peliin. Valikot luotiin tekemällä peliin uusi näkymä, jonka jälkeen näkymälle tehtiin käyttöliittymä solmu. Tämän jälkeen solmulle annettiin lapsisolmuksi VBoxContainer-objekti, jolle annettiin lapsisolmuksi vielä 3 nappia, jotka vastaavat päävalikon valintoja.



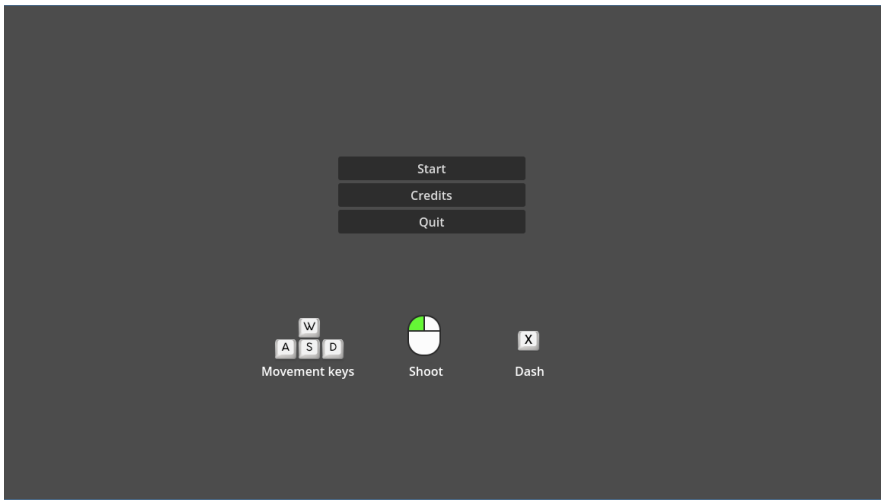
KUVA 14. Päävalikon napit näkymän editorissa.

Seuraavaksi valikolle kirjoitettiin koodi, jonka avulla käyttäjä pystyisi selaamaan valikkoa. Valikon napeille luotiin signaali, jonka avulla saatiin tietoon, kun nappia painettiin. Tämän jälkeen jokaiselle napille annettiin näkymää vaihtava koodinpätkä, joka kutsuu näkymäpuuta vaihtamaan tämänhetkisen näkymän toiseen käyttämällä “get_tree()” -komentoa.

```
1   extends Control
2
3
4   func _on_start_button_pressed() -> void:
5       get_tree().change_scene_to_file("res://Scenes/game_view.tscn")
6
7
8
9   func _on_quit_button_pressed() -> void:
10      get_tree().quit()
11
12
13  func _on_credits_button_pressed() -> void:
14      get_tree().change_scene_to_file("res://Scenes/credits.tscn")
```

KUVA 15. Kuva päävalikon toimintaa ohjaavasta koodista.

Päävalikkoon lisättiin näkyville myös pelattavan hahmon kontrollit, että pelaaja tietäisi saman tien, miten hahmoa ohjattaisiin.



KUVA 16. Kuva pelin päävalikosta

Pelin Käyttöliittymä on myös hyvin minimalistinen. Siinä on näkyvillä vain pelattavuuden kannalta pelaajalle oleelliset asiat, eli pelaajan elämäpisteet ja vihollisen elämäpisteet. Nämä pelaajalle tärkeät käyttöliittymän osat tehtiin käyttämällä Godotin ProgressBar -objektia ja lyhyttä skriptiä. Pelaajan elämäpisteet näytettiin pelissä sydäminä, jotka vähenivät aina, kun pelaaja otti vahinkoa.

KUVA 17. Pelaajan elämäpisteiden tekstuuri

Tämä tehtiin ensin lisäämällä sydämille tekstuuri, jotka asetettiin CanvasLayer -objektiin. Tämän jälkeen pelaajalle tehtiin lyhyt skripti, jossa ensin CanvasLayerilla olevat sydämet laitettiin taulukkoon, jonka jälkeen pelaajan ottaessa vahinkoa, kutsuttiin “update_heart_display” nimistä funktiota, joka päivitti näkyvien sydämien määrän vastaamaan pelaajan elämäpisteitä.

```
func update_heart_display():
    for i in range(hearts_list.size()):
        hearts_list[i].visible = i < playerHealth
```

KUVA 18. Elämäpisteiden visuaalista määrää päivittävä koodi

Vihollisen elämäpisteet tehtiin ensin tekemällä kaksi erillistä ProgressBar solmua. Ensimmäinen ProgressBar nimettiin vahinkopalkiksi ja siitä tehtiin läpinäkyvä harmaalla taustalla ja toinen ProgressBar nimettiin terveyspalkiksi ja siitä tehtiin punainen. Näiden palkkien tehtävänä oli luoda elävä terveyspalkki, joka näyttäisi pelaajan tekemän vahingon purskeissa. Tätä varten terveyspalkille täytyi antaa myös ajastin, jolle annettiin signaali, asetettaisi vahinkopalkin terveyspalkin arvoksi.

```
 9  func _set_health(new_health):
10      var prev_health = health
11      health = min(max_value, new_health)
12      value = health
13
14      if health <= 0:
15          queue_free()
16      if health < prev_health:
17          timer.start()
18      else:
19          damage_bar.value = health
20
21  func init_health(_health):
22      health = _health
23      max_value = health
24      value = health
25
26      damage_bar.max_value = health
27      damage_bar.value = health
28
29
30
31  func _on_timer_timeout():
32      damage_bar.value = health
33
```

KUVA 19. Vihollisen terveyspalkkia ohjaava koodi

Lopulta vastustajan terveyspalkki näyttää tältä sen ottaessa vahinkoa.

KUVA 20. Kuvassa näkyvä punainen palkki on vastustajan tämänhetkiset elämäpisteet, vaaleanharmaa palkki on juuri otettu vahinko ja tummanharmaa palkki on aikaisemmin menetetyt elämäpisteet.

5 PELIN JULKAISUSUUNNITELMA

Koska Godot mahdollistaa julkaisun usealla eri alustalle kuten Android, Windows, iOS, Linux ja HTML5 on pelin julkaiseminen mobiili laitteille tai pöytäkoneelle olisi mahdollista. Jos peli haluttaisiin julkaista maksullisena versiona, täytyisi siihen tehdä jotain muutoksia, koska osa pelissä käytetyistä tekstuureista ja äänistä ovat hyödynnettävissä vain, jos niitä ei käytetä kaupallisesti.

Pelistä julkaistiin demoversio Itch.io sivulle käyttäjätutkimuksen keräämistä varten. Vaikka emme aluksi suunnitelleet pelin julkaisemista web-versiona, päädyimme julkaisemaan pelin siellä, koska alustalle julkaiseminen oli helppoa ja ilmaista, vaatien vain käyttäjän luonnin ja pelin tiedostojen lataamisen sivulle. Jotta käyttäjätutkimukseen saataisiin enemmän vastauksia, päädyttiin ratkaisuun, jossa peli vietiin Itch.io alustalle HTML5-pohjaisena selaimessa toimivana versiona. Tämä oli mahdollista, koska pelin demoversio oli kohtalaisen kevyt ja suorituskyky ei oikein kärsinyt siitä, että peliä pelattiin selaimessa.

Jos pelin olisi halunnut julkaista mobiililaitteille, olisi siihen pitänyt lisätä kosketusnäytölle sopivat kontrollit. Julkaisu App Storeen ja Google play storeen olisi myös vaatinut monia eri asioita.

App storeen julkaisu vaatii erillisen Apple kehittäjä käyttäjän, joka maksaisi $99 vuodessa. Se vaatisi myös, että peli rakennettaisiin iOS laitteille Godotissa sekä pelistä pitäisi olla kuvia, kuvaus sekä applikaatio kuva. (Apple, 2025)

Google play storeen pelin julkaisu on kehittäjälle huomattavasti halvempaa. Yksittäisen applikaation pystyy julkaisemaan Googlen play storessa hintaan $25. Peli täytyisi rakentaa Android laitteille ja siitä pitäisi olla Google play storen sivulle nimi, kuvaus, kuvia, kategoria ja applikaatio kuva. Applikaatiolle täytyisi myös täyttää sisällön ikärajoitekysely ja annettava tietosuojakäytännön URL-osoite. (Google, 2025)

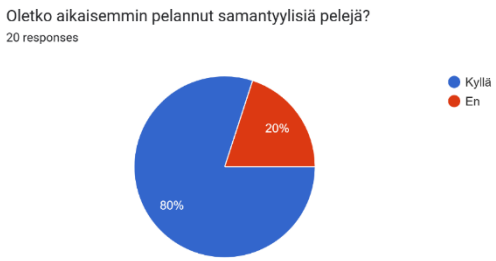
Pelin julkaisu käyttöjärjestelmille kuten Windows, Linux ja macOS ei vaatisi suurempia muutoksia itse peliin, julkaisu suuremmille alustoille kuten Epic Store and Steam, vaatisi sen, että peliä kehitettäisiin pidemmälle.

Pelin julkaisu Epic Storeen vaatisi myös $100 hakemusmaksun ja pelille pitäisi hankkia ikäräjäkoalition ikäraja luokitus. Lisäksi Epic Storeen täytyisi luoda käyttäjä ja täyttää vero- ja maksulomakkeet. (Epic Games, 2023.)

Pelin julkaisu Steamiin vaatisi $100 Steam Direct –maksun ja sille pitäisi luoda kauppasivu Steamiin. Lisäksi täytyisi täyttää erilliset lomakkeet maksutiedoilla sekä verotiedoilla. (Steamworks, s.a.)

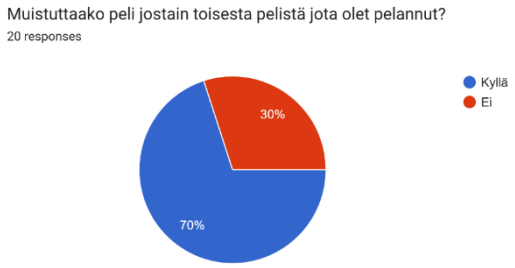
6 KÄYTTÄJÄTUTKIMUS

Opinnäytetyön ohessa tehdystä pelistä kerättiin Googlen forms:ia käyttäen käyttäjätutkimus. Tutkimuksessa kysyttiin käyttäjiltä heidän mielipidettänsä pelistä, sekä mitä kehitysmahdollisuuksia pelistä vielä löytyisi. Peliä pelasi 64 henkilöä ja tutkimukseen vastasi 20 peliä testannutta. Tutkimuksen ensimmäisenä kysymyksenä käyttäjiltä kysyttiin, ovatko käyttäjät aikaisemmin pelanneet samankaltaisia pelejä.



KUVA 21. Ympyrädiagrammi käyttäjien vastauksista ensimmäiseen kysymykseen.

Diagrammista käy selville, että suurin osa käyttäjistä on aikaisemmin pelannut samantyylisiä pelejä. Seuraavaksi käyttäjiltä kysyttiin, että muistuttaako opinnäytetyön ohessa tehty peli jostain muusta pelistä, ja jos muistutti niin mistä.

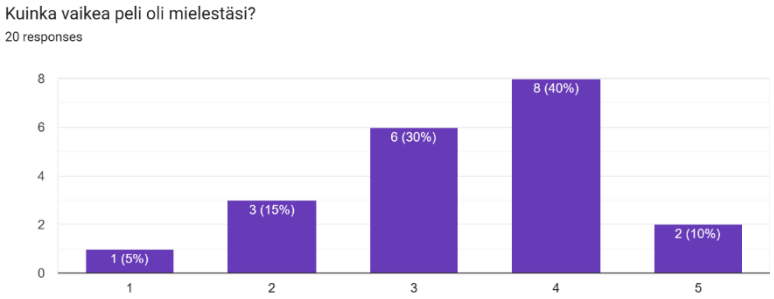


KUVA 22. Ympyrädiagrammi käyttäjien vastauksesta toiseen kysymykseen.

Tästä diagrammista käy selville, että vaikka suurelle osalle käyttäjistä, jotka ovat pelanneet samankaltaisia pelejä, opinnäytetyön ohessa tehty peli muistutti jotain niistä. Siltikään jokaisen samankaltaisen pelin pelaajan mielestä peli ei

muistuttanut mitään muuta samankaltaista peliä. Käyttäjiltä kysyttiin myös, mitä peliä opinnäytetyön ohessa tehty peli muistutti. Tähän kysymykseen vastattiin usealla eri pelillä, mutta eniten käyttäjät vastasivat “Touhou Project”, “Binding of Isaac”, “Enter the Gungeon”, “Realm of the Mad God” ja “Vampire Surviors”. Käyttäjien vastaukset kuvastivat hyvin paljon pelejä, joista opinnäytetyön ohella tehty peli otti inspiraatiota.

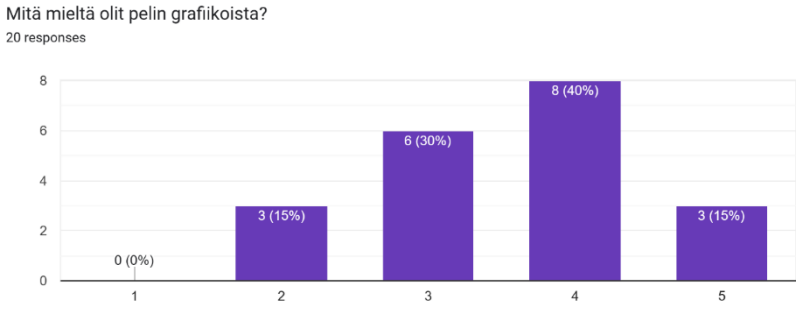
Seuraavaksi käyttäjiltä kysyttiin enemmän itse pelistä. Kolmantena kysymyksenä käyttäjiltä kysyttiin pelin vaikeustasosta.



KUVA 23. Käyttäjien vastaukset pelin vaikeustasosta.

Pylväsdiagrammista näkee, että peli oli yleisesti suhteellisen haastava pelaajille. Kun vastauksia katsoo tarkemmin, moni vastaaja, joka vastasi pelin olevan vaikea, ei ollut aikaisemmin pelannut samantyylisiä pelejä.

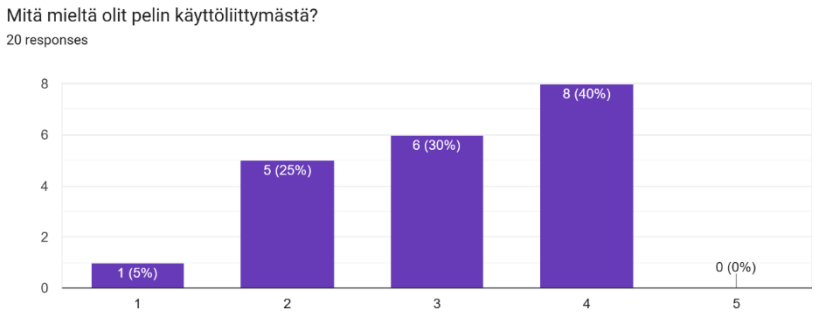
Seuraavaksi käyttäjiltä kysyttiin, mitä mieltä he olivat pelin grafiikoista.



KUVA 24. Pylväsdiagrammi, jossa näkyy käyttäjien vastaukset pelin grafiikoista.

Vaikka pelin grafiikoihin ei ollut käytetty hirveästi aikaa, sai peli silti suhteellisen hyvät arvostelut sen graafisesta puolesta.

Viidentenä kysymyksenä käyttäjiltä kysyttiin, mitä mieltä he olivat pelin käyttöliittymästä ja valikoista.



KUVA 25. Pylväsdiagrammi pelin käyttöliittymän saamista arvosteluista.

Käyttöliittymän arvostelujen pohjalta tehdyistä diagrammeista saadaan selville, että pelin käyttöliittymä ja valikot olivat ihan hyviä, mutta selvästi niiden tekemiseen olisi voinut käyttää enemmän aikaa.

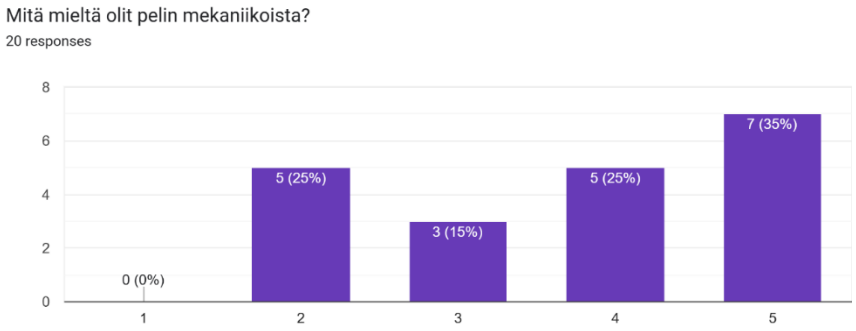
Tutkimuksen kuudes kysymys kysyi käyttäjien mielipiteitä peliin tehdyistä vastustajista ja niiden eri hyökkäyksistä.

KUVA 26. Pylväsdiagrammi pelin vihollisten saamista arvosteluista.

Tästä diagrammista käy selville, että peliin tehdyt viholliset olivat suhteellisen hyvin tehty. Kun saatuja vastauksia lähtee tutkimaan tarkemmin, nähdään, että

usea, jonka mielestä peli oli haastava, arvosteli peliin tehdyt vastustajat heikommin, kuin ne, joiden mielestä peli oli helppo tai keskivaikea.

Seuraavaksi käyttäjiltä kysyttiin heidän mielipidettänsä pelin mekaniikoista.



KUVA 27. Pylväsdiagrammi pelin mekaniikkojen saamista arvosteluista.

Tästä diagrammista näkee, että peliin tehdyt mekaniikat olivat selvästi pelin yksi vahvimmista osista. Vaikka mekaniikat saivat hyvät arvostelut, kyselyn vapaa sana -kohdassa kävi ilmi, että moni pelin mekaniikka olisi kaivannut pieniä muutoksia, että ne olisivat olleet käyttäjäystävällisemmät.

Tutkimuksen viimeisessä kysymyksessä käyttäjiltä kysyttiin heidän mielipidettänsä pelin musiikeista ja ääniefekteistä.

KUVA 28. Pylväsdiagrammi, jossa näkyy käyttäjien arvostelu peliin valitusta musiikista

Diagrammista näkee, että pelin nopea tempoinen musiikki ja vähäiset, mutta vaikuttavat ääniefektit olivat selvästi käyttäjien mielestä mielekkäät.

Viimeiseksi käyttäjille annettiin mahdollisuus vapaaseen sanaan pelistä. Tähän moni käyttäjä antoi paljon erilaisia kehitysmahdollisuuksia. Suuri osa käyttäjistä olisi halunnut, että pelissä oleva hyökkäys toiminto olisi toiminut pitämällä nappia pohjassa, sen jatkuvan klikkaamisen sijaan. Käyttäjät löysivät tähän myös porsaanreiän. Jos käyttäjä käytti jotain ohjelmaa, joka mahdollisti usean klikkauksen simuloinnin yhdellä napin painalluksella, käyttäjä kykeni ampumaan loputtoman määrän sekunnissa. Tämä teki pelistä todella helpon, sillä jokainen vihollinen kuoli melkein heti syntyessään.

Toinen paljon palautetta saanut asia oli se, kun viholliset syntyvät kentälle, ne ovat hetken kuolemattomia, tätä ei kuitenkaan pelissä tuoda esille mitenkään, joten vihollisille olisi voitu tehdä jonkin tapainen indikaattori siitä, että ne ovat kuolemattomia.

Viimeinen paljon huomiota saanut asia oli pelaajan väistöliike, joka oli monen mielestä hyvin vahva eikä siinä ollut visuaalista indikaattoria kuolemattomuuden lopulle. Tämä ongelma voitaisiin korjata helposti lyhentämällä väistön jälkeistä kuolemattomuutta ja lisäämällä indikaattori kuolemattomuudelle ja väistön lataus ajalle.

Annetun palautteen pohjalta voidaan tulla lopputulokseen, että pelin testaajat olivat yleisesti tyytyväisiä peliin ja sen mekaniikkoihin. Myös pelin grafiikoihin ja äänimaailman oltiin tyytyväisiä. Pelaajat kokivat pelin myös tarpeeksi haastavana, joka on hyvä, sillä pelin halutaan olevan uudelleen pelattava. Tutkimuksen pohjalta saatiin myös tieto pelin mekaniikkojen puutteista. Käyttäjätutkimuksesta selvisi myös, että pelaajat, jotka olivat ennestään pelanneet samankaltaisia pelejä, arvostelivat pelin eritavoin kuin he, joilla ei ollut aikaisempaa kokemusta tästä peligenrestä. Tutkimuksen perusteella selvisi, että pelille olisi selvästi oma markkinarako, mutta se ei välttämättä toisi uusia pelaajia genreen, joten jatkokehityksen kannalta voisi olla kannattavaa keskittyä bullet hell -genrestä kiinnostuneiden pelaajien kokemuksen parantamiseen.

7 POHDINTA

Opinnäytetyön tavoitteena oli suunnitella ja luoda bullet hell –tyyppinen peli sekä tutustua Godot –pelimoottorin käyttöön. Pelin suunnitteluvaiheessa pelille määriteltiin tietyt tekniset ominaisuudet, jotka haluttiin toteuttaa ja lisäksi mietittiin mahdollisia lisäyksiä, joita voitaisiin toteuttaa, kun peliä lähdettäisiin kehittämään pidemmälle. Peli valmistui ajallaan ja sille asetettuihin vähimmäistavoitteisiin päästiin. Viimeistelty demoversio on alusta loppuun asti pelattavissa ja siihen luodut ominaisuudet toimivat suunnitellusti.

Godotin ja GDScriptin oppiminen tapahtui yllättävän nopeasti, vaikka kummallakaan ei ollut aikaisempaa kokemusta niiden käytöstä. Godot osoittautui hyvin toimivaksi ja tehokkaaksi tämän tyylisen pelin luomiseen.

Kehityksen aikana pelistä luotiin neljä erilaista demoa. Aluksi määriteltyyn joka viikkoiseen demoon ei aivan päästy, vaan peliä demottiin joka toinen viikko, koska osa kehitysajasta meni raportin kirjoittamiseen ja aikaisemmilla viikoilla tehtyjen toimintojen hiomiseen. Godotista puuttui myös Unityn kaltainen ominaisuus, jonka avulla tiedostoja olisi voinut muokata samanaikaisesti toisen henkilön kanssa, joka hidasti tekemistä. Projektin versiohallintana käytettiin Github-alustaa, johon viikoittaiset demoversiot saatiin helposti talteen.

Pelistä kerättiin myös käyttäjätutkimus, jonka perusteella peliä pystyttäisiin kehittämään jatkossa pidemmälle. Tutkimus auttoi myös määrittämään, kuinka hyvin pelin toteutus onnistui. Olemme tyytyväisiä käyttäjätutkimuksesta saatuun palautteeseen ja se myös korosti tarvittavia muutoksia pelin jatkokehityksen kannalta. Tutkimus toi myös esille että, osa kehittämisvaiheessa järkevältä tuntunut asia ei kuitenkaan ollut pelin testaajille tarpeeksi selvästi esitetty ja jälkeenpäin selvätkin asiat kuten hyökkäysnapin jatkuva uudelleen painaminen tekee käyttäjän pelikokemuksesta huonomman. Pelaajan saama visuaalinen ja äänipohjainen palaute pelin tapahtumista on yksi asia missä on vielä parannettavaa ja on yksi asia mistä käyttäjätutkimuksessa tuli palautetta.

Pelin kehittäminen osana opinnäytetyötä paransi selvästi ymmärrystä pelin suunnittelu- ja kehitysprosessista. Näitä opittuja taitoja pystyy myös soveltamaan tulevaisuudessa erilaisten pelien kehityksessä alustasta tai pelimoottorista riippumatta. Vaikka peli täytti kaikki perusvaatimukset, mielestämme se ei ollut täysin julkaisuvalmis suuremmille alustoille kuten Steam ja siihen olisi pystynyt lisäämään vielä monia eri ominaisuuksia. Olemme kuitenkin tyytyväisiä opinnäytetyön aikana kehitetyn pelin lopputulokseen. Mielestämme Godotin ja GDScriptin osaamisemme on parantunut niin, että tulevaisuuden projektit tai pelin jatkokehittäminen tapahtuisi huomattavasti nopeammalla tahdilla.

LÄHTEET

Apple 2025. Membership program enrollment. Luettavissa: https://developer.apple.com/help/account/membership/program-enrollment. Luettu: 17.5.2025

Cherkaoui, I. 2024. The Hidden Boss: UX Why Game Menus Matter More Than You Think. Luettavissa: https://medium.com/design-bootcamp/the-hidden-boss-ux-why-game-menus-matter-more-than-you-think-3b90678949ec. Luettu: 18.5.2025

Cohen-Peckham, E. 17.10.2019. “How Unity built the world’s most popular game engine”. Luettavissa: https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/. Luettu: 9.5.2025

Dawe, L. 2014. “Godot Game Engine Is Now Open Source” GamingOnLinux. Luettavissa: https://www.gamingonlinux.com/2014/02/godot-game-engine-is-now-open-source/. Luettu: 9.4.2025

Epic Games. 3.9.2023. Epic Games Store Launches Self-Publishing Tools for Game Developers and Publishers. Luettavissa: https://store.epicgames.com/en-US/news/epic-games-store-launches-self-publishing-tools-for-game-developers-and-publishers. Luettu: 2.6.2025

Etcheverry, I. 2017. “Introducing C# in Godot” Godot Engine. Luettavissa: https://godotengine.org/article/introducing-csharp-godot/. Luettu: 22.4.2025

Furi s.a. Furi. Luettavissa: https://www.thegamebakers.com/furi/. Luettu: 21.4.2025

Gazula, S. s.a. Game Engines: All you Need to Know. Gameopedia. Luettavissa: https://gameopedia.com/blogs/game-engines-all-you-need-to-know. Luettu: 31.03.2025

Gbo Seo Solutions. 2023. “Unreal Engine: The Ultimate Game Development Platform”. Luettavissa: https://medium.com/%40gbo.seo.solutions/unreal-engine-the-ultimate-game-development-platform-7cb1312651cc. Luettu: 18.5.2025

Godot Engine 2023. Godot 4.0 sets sail: All aboard for new horizons. Luettavissa: https://godotengine.org/article/godot-4-0-sets-sail/ Luettu: 22.4.2025

Godot Engine 2024. What is GDScript and why should I use it? Luettavissa: https://docs.godotengine.org/en/stable/about/faq.html#what-is-gdscript-and-why-should-i-use-it. Luettu: 5.5.2025

Godot Engine 2025a. Godot Engine license. Luettavissa: https://godotengine.org/license/. Luettu: 6.4.2025

Godot Engine 2025b. Introduction to Godot. Luettavissa: https://docs.godotengine.org/en/stable/getting_started/introduction/introduction_to_godot.html. Luettu: 7.4.2025

Godot Engine 2025c. Features. Luettavissa: https://godotengine.org/features/. Luettu: 7.4.2025

Godot Engine 2025d. Godot 4.4, A unified experience. Luettavissa: https://godotengine.org/releases/4.4/. Luettu: 22.4.2025

Godot Engine 2025e. GDScript: An introduction to dynamic languages. Luettavissa: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_advanced.html. Luettu 5.5.2025

Godot Engine 2025f. Godot's design philosophy. Luettavissa: https://docs.godotengine.org/en/stable/getting_started/introduction/godot_design_philosophy.html. Luettu 15.5.2025

Godot Engine 2025g. Nodes and Scenes. Luettavissa: https://docs.godotengine.org/en/stable/getting_started/step_by_step/nodes_and_scenes.html. Luettu: 15.5.2025

Godot Engine 2025h. Using signals. Luettavissa: https://docs.godotengine.org/en/stable/getting_started/step_by_step/signals.html. Luettu: 19.5.2025

Godot Engine 2025i. List of features. Luettavissa: https://docs.godotengine.org/en/stable/about/list_of_features.html. Luettu: 19.5.2025

Google 2025. Create and set up your app. Luettavissa: https://support.google.com/googleplay/android-developer/answer/9859152?hl=en. Luettu 17.5.2025

Horowitz, S & Looney, S. 2014. The essential guide to game audio: the theory and practice of sound for games. Focal Press Taylor & Francis group. New York and London.

Ixiegaming s.a. iXie's Guide to Mastering the 7 stages of Game Development. Luettavissa: https://www.ixiegaming.com/ebooks/guide-to-mastering-the-7-stages-of-game-development.pdf. Luettu: 21.4.2025

Kraj, N. 2020. Keys to Rational Enemy Design. GDKeys. Luettavissa: https://gdkeys.com/keys-to-rational-enemy-design/. Luettu: 20.5.2025

Linietsky, J. 2014a. “Godot History in Images!” Godot Engine. Luettavissa: https://godotengine.org/article/godot-history-images/. Luettu: 9.4.2025

Linietsky, J. 2014b. “Godot Engine reaches 1.0, first stable release” Godot Engine. Luettavissa: https://godotengine.org/article/godot-engine-reaches-1-0/. Luettu: 9.4.2025

Linietsky, J. 2018. “Godot 3.0 is out and ready for the big leagues” Godot Engine. Luettavissa: https://godotengine.org/article/godot-3-0-released/. Luettu: 22.4.2025

Linietsky, J. 2019. “Godot 3.1 is out, improving usability and features” Godot Engine. Luettavissa: https://godotengine.org/article/godot-3-1-released/#gdscript-typing. Luettu: 22.4.2025

Maxine. 27.5.2025. What platforms are supported by Unity? Luettavissa: https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity. Luettu: 2.6.2025

Mozilla 22.6.2016. Mozilla Awards $385,000 to Open Source Projects as part of MOSS “Mission Partners” Program. Luettavissa:

https://blog.mozilla.org/en/mozilla/mozilla-awards-385000-to-open-source-projects-as-part-of-moss-mission-partners-program/. Luettu: 11.4.2025

Querné, J. 30.5.2023. An explanation of different Game Design Approaches. Luettavissa: https://medium.com/@josselin.querne/an-explanation-of-different-game-design-approaches-12b0937c8ebd. Luettu: 20.5.2025

Rabin, S. 2009. Introduction to Game Development. Toinen painos. Charles River Media. Boston

Santos, R. 2023. “What’s new in C# for Godot 4.0” Godot Engine. Luettavissa: https://godotengine.org/article/whats-new-in-csharp-for-godot-4-0/. Luettu: 22.4.2025

Steamworks s.a. Steam Direct Steamworks-jakeluohjelmaan liittyminen. Luettavissa: https://partner.steamgames.com/steamdirect. Luettu 18.5.2025

Stockton, M.D. 28.11.2023. The Evolution of Graphics Cards: A Historical Perspective. Medium. Luettavissa: https://medium.com/@stocktonmattdavies/the-evolution-of-graphics-cards-a-historical-perspective-609be03e421c. Luettu: 31.03.2025

Touhou Project s.a. What is the Touhou Project? Luettavissa: https://touhou-project.news/touhou_project/. Luettu: 21.4.2025

Undertale s.a. Undertale. Luettavissa: https://undertale.com/about/. Luettu: 21.4.2025

Unreal Engine. 2023. “Unreal Engine 5.3 is now available!”. Luettavissa: https://www.unrealengine.com/en-US/blog/unreal-engine-5-3-is-now-available. Luettu: 18.5.2025

Verschelde, R. 2020. “Here comes Godot 3.2, with quality as priority” Godot Engine. Luettavissa: https://godotengine.org/article/here-comes-godot-3-2/. Luettu: 22.4.2025

Webster, A. 2024. Persona director says making beautiful menus is ‘actually really annoying’. Luettavissa: https://www.theverge.com/2024/10/7/24262357/persona-metaphor-menu-design. Luettu: 18.5.2025

Wikipedia 2025. Godot (game engine). Luettavissa: https://en.wikipedia.org/wiki/Godot_(game_engine). Luettu: 9.4.2025

LIITTEET

Pelinsuunnitteludokumentti LIITE 1

Pelisuunnitelma

Pelin vaatimukset

2D-Peli jossa:

- Yksi areena

-Pelaaja pystyy liikuttamaan ja hyökkäämään hahmolla

-Automaattinen hyökkääminen

-Hiiri tähtäimenä?

-Vihollisen hyökkäysten väistäminen / torjunta?

-Muutama vihollinen, jolla useampi eri hyökkäys

-Pelaajalla ja vihollisella omat elämäpisteet näkyvissä (Pelaajalla 1 elämäpiste vai useampi)?

-Main menu

Mahdollisia lisäyksiä:

Useampi areena

-Pistesysteemi (aika, pisteet tms?)

-upgrade-systeemi:

-Vihollisten välissä huone, jossa kauppias tai vastaavaa?

Peli ottaa inspiraatiota muista bullet hell shoot ‘em up tyyppisistä peleistä kuten Touhou project, Furi ja Undertale.

Teemana colosseum / gladiaattori areena, jossa erilaisia vastustajia


Kagiyama Hina
38
怨霊「ミスフォーチュンズホイール」
Bonus 0833020 History 00/01
HiScore 010000000
Score 004105852
Player
Power 4.30
東方神風録
090870
Enemy
36.8fps