Opinnäytetyö (AMK)

Tradenomi

2024

Mika Lukkarinen

Godot-pelimoottori ja proseduraalinen generointi

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tradenomi

2024 | 46 sivua

Mika Lukkarinen

Godot-pelimoottori ja proseduraalinen generointi

Algoritmit ovat tärkeä osa tietojenkäsittelyä. Ne helpottavat ison datan lukemista ja käsittelyä, mutta algoritmeja voidaan myös hyödyntää datan käsittelyn lisäksi datan luomisena ja tätä kutsutaan proseduraaliseksi generoinniksi.

Opinnäytetyön tavoitteena oli perehtyä, miten proseduraalinen generointia voitaisiin pelillistä viihdekäyttöön Godot-pelimoottorilla ja lisäksi kartoitettiin samalla Godotin vahvuuksia ja heikkouksia. Tavoitteena oli siis kehittää todella selkeä pohja tason generoinnille, joka olisi helppo ottaa käyttöön jatkossa ja kehittää tämän proseduraalisen pohjan ympärille mahdollinen pienin toimiva tuote eli MVP.

Opinnäytetyössä perehdyttiin tarkemmin yhden proseduraalisen mallin kehittämiseen ja kyseisen mallin avulla pelin kartan luomiseen ja miten algoritmin generoimaa dataa voidaan hakea ja hyödyntää jatkokehityksen kannalta. Lisäksi opinnäytetyössä selvitettiin tarkemmin itse pelin kehitystä ja siihen sisältyviä haasteita ja ongelmia. Opinnäytetyössä nähdään myös, miten hyvin sovellus ohjelmoinnin taidot kääntyvät pelinkehityksen puoleen.

Lopputuloksena saatiin tason generointiin toimiva algoritmi, mutta pienin toimiva tuote ei ole isoin peli mekaniikoiltaan. Opinnäytetyössä laadittiin dokumentaatiot pelin grafiikan pohdintaan liittyen ja laatuvaatimukset ovat olleet korkeat, jotta voitaisiin varmistaa, että ei ole virheitä tai visuaalista epäjohdonmukaisuutta pelin ulkonäössä varsinkin, kun pelin kartta on koneen luoma.

Asiasanat:

proseduraalinen generointi, Godot, pelinkehitys, GDScript, MVP

Bachelor’s | Abstract

Turku University of Applied Sciences

Bachelor of Business Administration

2024 | 46 pages

Mika Paavo Johannes Lukkarinen

Godot game engine and procedural generation for level design

Algorithms are an important aspect of data processing as they facilitate the ease reading and handling of copious amounts of data. However, algorithms can also be used for data creation, a process known as procedural generation. The aim of this thesis was to delve deeper into how procedural generation could be utilized for entertainment purposes using Godot Game Engine, while also mapping out the strengths and weaknesses of Godot.

The thesis focused more specifically on the development of one procedural model and the creation of a game map around it, and how this data can be retrieved and utilized for future development. The thesis also delves deeper into the actual game development and what it entails, and it also shows how well application programming skills translate into game development.

The aim of the thesis was to develop a clear basis for level generation, which would be easy to use in the future and to develop a possible MVP also know minimum viable product around it. The result is a functioning algorithm for level generation, but the minimum viable product is not the largest in terms of mechanics or variety, but the quality standards for everything has been high to make sure there’s no bugs or visual incoherence.

Keywords:

Procedural generation, Godot, Game development, GDScript, MVP

Sisältö

Kuvat

Kuviot

Ohjelma

Taulukot

Käytetyt lyhenteet tai sanasto

2D	kaksiulotteisuus (finto, 2024) .
3D	kolmiulotteisuus (finto, 2024).
AI	tekoäly (finto, 2024)
Algoritmi	Prosessi tehtävän suorittamiseen (finto, 2024)
Autokartoitus	Työkalu, joka nopeuttaa laattojen sijoittamista automaattisesti sijoittamalla oikeat laatat kohdilleen. (godotengine, 2023)
Avoin lähdekoodi	Avoin lähdekoodi eli open source on ohjelmisto, jonka lähdekoodi on avoimesti saatavilla ja vapaa, käyttää muokata ja levittää oman tarkoituksen mukaan. Kuka tahansa voi käyttää siis ohjelmistoa omaan tarkoitukseensa ilmaiseksi yleisesti (finto, 2024).
BSP-puut	Binäärinen avaruuden osiointi (tai BSP-puu) on tietorakenne, jota käytetään objektien järjestelyyn avaruudessa. (symbolcraft, 2024)
C#	Microsoftin .Net-alustalle kehitetty ohjelmointikieli (finto, 2024)
GDScript	GDScript on Godot-pelimoottorille suunniteltu korkean tason, oliopohjainen, imperatiivinen ja asteittain tyypitetty ohjelmointikieli. (godotengine, 2023)
Godot	Pelimoottori
Laatta	Projektissa 16 pikseliä eli 8x8 laatta kooltaan, joista rakennetaan peli.

Laattakartta	kokoelma laattoja, joiden avulla rakennetaan pelin kartan grafiikka yhdistelemällä niitä vierekkäin. Myös sisältää dataa törmäyksistä ja jokaiselle laatalle tehdään oma id-luku hakemiseen
MVP	MVP eli pienin toimiva tuote.
Näyttämö	Virtuaalinen ympäristö, jossa objektit sijaitsevat.
Pelaaja	Pelaajana kuvaan pelissä pelattavaa hahmoa ja sen ohjaajaa ja niihin liittyviä sääntöjä
Proseduraalinen generointi	Asian luominen algoritmin kautta ilman ihmisen kosketusta.
Solmu	Pohja jokaiselle objektille pelimottorissa. (godotengine, 2024)
Sprite	osittain läpinäkyvä kuvahahmo, jota käytetään pelissä liikkuvissa tai animoiduissa hahmoissa tai objekteissa.
Tila-automaatti	Tila-automaatti eli äärellinen automaatti, on systeemi, joka automaattisesti siirtyy tehtävien välillä pelissä. Yleisesti käytetty pelaajan animaatiotilan kanssa (Shead, 2018).
Törmäys	Tarkoittaa kohtaa kartassa, mistä pelaaja ei pysty liikkumaan läpi, koska pelaaja törmää objektiin.
Unity	Pelimoottori, joka tunnetaan useiden alustojen tuesta.

1 Johdanto

Algoritmit ovat tärkeä osa tietojenkäsittelyä, ja niiden merkitys on kasvanut tekoälyn ja laajojen datakirjastojen myötä. Ison datan lukemista ja käsittelyä helpotetaan algoritmien ja niiden visualisoinnin avulla, mutta niiden oppiminen on haastavaa. Algoritmeja voidaan hyödyntää lisäksi datan käsittelyssä ja datan luomisessa, jota kutsutaan proseduraaliseksi generoinniksi. Kyseiset teknologiat ovat alkaneet levitä myös viihdekäyttöön.

Opinnäytetyön tavoitteena on tutustua Godot-pelimoottoriin ja kartoittaa sen vahvuuksia ja heikkouksia. Godot'tia verrataan kilpailijoihinsa, ja perustellaan tarkemmin, miksi se on valittu tähän projektiin. Tavoitteena on kehittää Godot-pelimoottorilla proseduraalinen järjestelmä pelin karttojen luomiseen. Generoinnista on tarkoitus kirjoittaa tarkemmin ja selittää sen toiminnallisuus. Generointi johtaa myös erilaisiin päätöksiin luovuuden kannalta, jotka on tehtävä erityisesti pelillistämisen suhteen, kun algoritmia kehitetään viihdekäyttöön. Opinnäytetyössä hyödynnetään Godot'sta 4.2 versiota eli Godot 4.

Teoriaosuudessa avataan laajemmin sitä, mikä Godot on pelimoottorina, mitä proseduraalinen generointi tarkoittaa, miten pelin kehitys aloitetaan algoritmin ympärille ja miten lähdekritiikkiä käytetään peliä kehittäessä. On tärkeää käydä läpi, millaisia päätöksiä on tehtävä rajallisten resurssien myötä ja miten nämä voivat viedä aikaa, kuten kaiken grafiikan tekeminen projektiin eikä ulkoistaminen toiselle.

Opinnäytetyö on toiminnallinen, eli tavoitteena on dokumentoida alusta loppuun, kuinka kyseistä peliä kehitettiin, miten lopputuloksiin päädyttiin tietyllä tavalla, miten ratkaisut tehtiin ja kuinka lopullinen tulos eroaa alkuperäisestä ideasta. Päätavoitteena oli kehittää yksinkertainen luolastoseikkailupeli. Pelin tason luonti toteutetaan siten, että joka kerta kartta näyttää erilaiselta satunnaisuuden ansiosta. Lisäksi selvitettiin, kuinka hyvin sovellusohjelmoinnin taidot toimivat myös pelinkehitykseen.

Turku Game lab toimii opinnäytetyön toimeksiantajana. Game lab on Turun yliopiston ja Turun ammattikorkeakoulun perustama yhteistyöympäristö pelialan koulutuksen ja kehityksen tukemiseksi. Opinnäytetyöstä hyötyy Game lab siten,

että Godot-pelimoottorista ja sen tarjoamista mahdollisuuksista saadaan laajempaa tietämystä tulevaisuuden koulutuksen kannalta.

1.1 Moottorin valinta

Projektin alkuvaiheessa on tärkeää valita, millä pelimoottorilla halutaan kehittää peli. Godot valittiin ilmaisuuden ja avoimen lähdekoodin vuoksi. Tämä tarkoittaa, että omistetaan täysin se, mitä kehitetään, eikä tarvitse huolehtia lisensseistä tai maksuista. Toki myös oman pelimoottorin kehitystä voidaan harkita, mutta tämän projektin kannalta se on liian aikaa vievää ja haastavaa. Projektin kannalta on tärkeää, että moottori tukee 2D-pelejä hyvin, koska pelin assettien tekemiseen hyödynnetään ohjelmaa nimeltä Asesprite. Tällä ohjelmalla luodaan peliin käytettäviä spritejä ja tallennetaan niihin kaikki animaatiotiedot suoraan. Aseprite on lähdekoodilla saatavilla oleva ohjelmisto. Tämä tarkoittaa, että Aseprite ei täytä avoimen lähdekoodin vaatimuksia täysin, vaan enemmän sen lähdekoodi on nähtävissä mutta laillisesti sitä ei saa uudelleen julkaista. Pelin asetit tehdään Asespritessä ja nämä sisältävät animaatio dataa, joten on tärkeää, että voidaan siirtää pelimoottoriin nämä spritet suoraan, joten moottorin on oltava yhteensopiva Asespriten tiedostojen kanssa tai siihen on oltava saatavilla lisäosa tämän hoitamiseen.

Lähin vertailukohta Godotille olisi Unity, koska molemmat pelimoottorit ovat soveltuvia 2D- ja 3D-pelien kehittämiseen ja ovat erittäin monialustaisia. Ne tukevat tietokoneita, mobiililaitteita ja VR-laitteita. Molemmat moottorit käyttävät myös C#-kieltä tai ovat kehitetty tällä kielellä. Taulukko 1. Godot’n ja Unityn vertailu. havainnollistaa eroja Godotin ja Unity:n välillä.

Taulukko 1. Godot'n ja Unityn vertailu.

	Godot	Unity
Avoin Lähdekoodi	Kyllä	Ei
Hinta	Ilmainen	On maksullisia versioita ja lisenssejä, mutta harjoittelija käyttöön ilmainen tiettyyn kohtaan saakka.
2D	Kyllä	Kyllä
3D	Kyllä	Kyllä
Koodaus Kieli	C#, GDScript, Visual Script, C++	C#, JavaScript, Rust
Kielestä tarkemmin	Käyttää GDScriptia ja C#. GDSripti on suunniteltu olemaan helppo lukuista ja käytettävää pythonin tyyliin.	Pääsääntöisesti c#, mutta tarjoaa myös visuaalista strippaamista.
Alustat	Tietokone, XR ja Mobiili	Desktop, Mobile, XR ja p Pelikonsolit
Yhteisö	Aktiivinen yhteisö, varsinkin eri lankapalveluissa ja Godotin omasta foorumista löytyy todella hyvin tukea, YouTube videoita alkanut tulemaan paljon Godot 4 myötä.	Todella iso yhteisö ja paljon materiaalia, assetteja ja yleisesti kanavia löytämään ohjeita löytyy
Yleisyys markkinoilla ja yrityksissä	Kasvava yleisyys pääsääntöisesti Indie tasolla, varsinkin avoimen lähdekoodin sallivan muokkaus mahdollisuuden ansiosta ei isossa käytössä isoissa firmoissa.	Todella yleinen AAA ja Indie peleissä. Löytyy paljon työmarkkinoilla käyttöä.
Tutoriaalit	Löytyy Godot 4 varsinkin aika hyvin, mutta paljon joutuu Godotin dokumentaation kautta selvittää asioita silti	Todella laajasti ja paljon kursseja löytyy esimerkiksi.
Asesprite tuki	Yhteisöltä löytyy monia lisäohjelmia tukemaan tätä	Sisään rakennettu tuki suoraan

Vaikka Unity:lle on helpommin saatavilla resursseja, GDScriptin käyttö Godot'ssa kiinnosti projektin kannalta erityisesti. Se näyttää visuaalisesti helppolukuiselta ja yhdistää C#-kielen ja Pythonin parhaat puolet.

Unitylle on loputtomasti saatavilla resursseja, raportteja ja ohjeita, joten Godot valittiin, jotta saadaan vertailudataa uudesta moottorista. Erityisen kiinnostavaa on, miten se pärjää proseduraalisen generoinnin suhteen. Tämä on tärkeää, kun pelin alussa käydään läpi laajoja ohjelmistokoodeja kaiken luomiseksi. Näin voidaan testata suorituskykyä ja vakautta.

1.2 Godot-version valinta

Godot-foorumilla käyttäjä (Atria, 2023) on tehnyt erinomaisen vertailun Godot 3:n ja 4:n välillä. Lisäksi Godot-tiimin virallinen dokumentaatio sisältää tietoa näiden kahden version eroista (Godot, 2024).

Molemmat lähteet korostavat samoja keskeisiä seikkoja: Godot 3 on vakaa, siitä on enemmän yhteisön laatimaa dokumentaatiota ja se tukee visuaalista ohjelmointia. Toisaalta Godot 4 on uusi versio, jossa ei enää ole visuaalista ohjelmointitukea (paitsi varjostimien osalta). Tämä ei ole ongelma, koska ainoa visuaalinen ohjelmointi, jota on tehty, on Unrealin siniset piirustukset ja Blenderissa solmujen säätäminen. Niinpä Godot 4.2 valittiin tämän projektin käyttöön, koska se on uusin vakaa versio Godot 4:stä. Godot 4.2 tarjoaa uusia ominaisuuksia animaatioiden sekoittamiseen ja uusiin laattakarttatyökaluihin. Erityisesti uudistettu laattakartta-automaattikarttatyökalu, joka tunnetaan nimellä maastotyökalu, sopii hyvin käyttötarkoitukseen. Teoriassa koko pelialue voidaan luoda tätä uutta laattakarttatyökalua käyttäen. Nyt on vain selvitettävä paras tapa toteuttaa se.

2 Proseduraaliset mallit

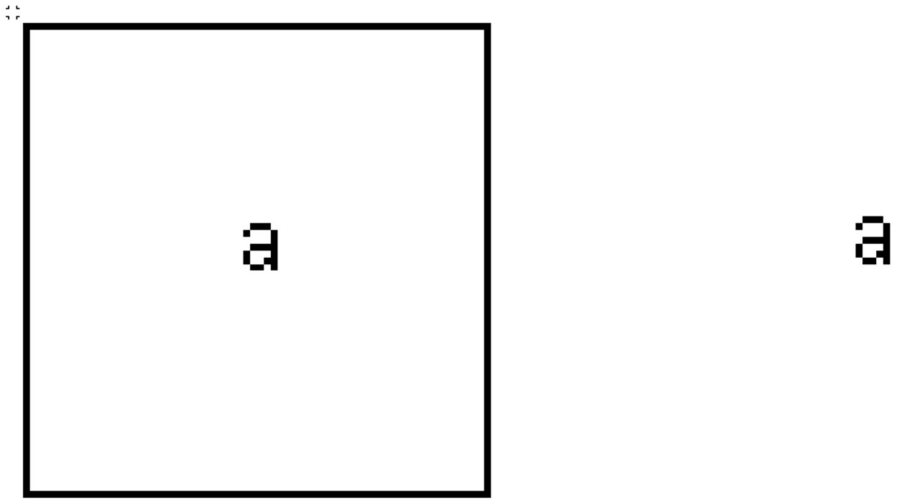
2.1 Mallin valinta

Aluksi on valittava malli, jonka perusteella ongelmaa lähdetään ratkaisemaan. Tässä suuri tekijä on se, mistä löytyy parhaiten dokumentaatiota erityisesti Godot'tiin liittyen. Pääsääntöisesti perehdyttiin alhaalla esiintyvään malliin, mikä koettiin helpoimmaksi toteuttaa mahdollisten resurssien myötä ja mikä parhaiten osuisi visioon siitä, miltä pelin haluttaisiin tuntuvan.

Ideana on pääsääntöisesti saada taso, joka pystyy generoimaan erilaisuutta, mutta ei vaadi paljoa valmiiden asetusten luomista. Mallin tulisi toimia enemmän automaattisena tason pohjan luontina, johon voidaan sitten sijoittaa pelin elementtejä. Huoneita ei tehdä valmiina kohtauksina, jotka sijoitetaan peliin, vaan halutaan automaation myötä tehdä luolaosuus ja nähdä, onko se riittävää pelin kannalta. Binäärinen avaruuden osionnin algoritmi on valittu tätä varten.

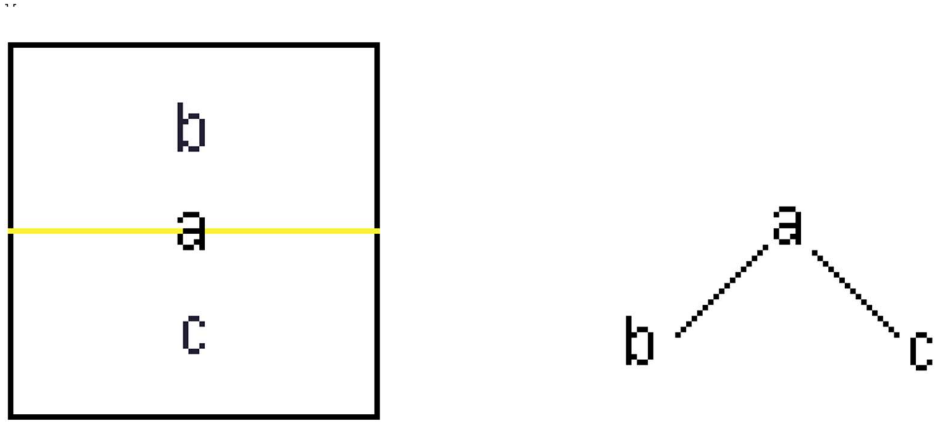
2.2 Binäärinen avaruuden osiointi

BSP tarkoittaa siis binääristä avaruuden osiointia. Ensimmäinen kuva näyttää, miltä yksi puu näyttää. (Kuva 1.)



Kuva 1. Esimerkki yksi puu.

Kaksi puuta näkyy kuvassa 2, jossa a on itse puu ja b ja c ovat oksia. Tätä hyödyntäen voidaan leikata tietty alue moneen osaan ja sijoittaa esimerkiksi niiden keskelle laattoja luomaan pelin tason. Tämän ympärille jouduttaisiin kuitenkin keksimään tapa ohjata pelaaja oksasta oksaan, jotka voitaisiin täyttää laatoilla luoden huoneita. (Kuva 2.)



Kuva 2. Esimerkki kahdesta puun oksasta.

BSP valittiin, koska se luo hyvin huoneen osia ja siihen löytyi paljon paremmin dokumentaatiota luolan sijaan. Se auttaisi suuresti laattakartan suunnittelussa, koska tiedetään, että tarkoituksena on rakentaa huoneita ja käytäviä niiden välillä. Tämä vastasi hyvin visuaalista toivetta mahdolliselle pelin tasojen ulkonäölle.

3 Proseduraalisen mallin kehitys

Proseduraalinen malli tarkoittaa siis algoritmia, joka luo dataa automaattisesti ilman ihmisen kosketusta tai säätöä. Mallista haluttiin tehdä mahdollisimman paljon koneen tekemä, vaikka se saattaisi heikentää pelikokemuksen luovuutta tai monimutkaisuutta. Suurin osa luola seikkailu pelien huoneista koostuu siten, että huoneet ovat valmiiksi ihmisen tekemiä, jotka vain sijoitetaan satunnaisesti. Haluttiin tähän projektiin, että kone sijoittaa kaiken ja tekee kaiken. Myös sen takia, että ymmärrettäisiin paremmin, mitä kaikkea tässä tapahtuu oikeasti.

Mallin kehityksen alussa hyödynnettiin Jono Shieldsin dokumentaatiota (2023), joka käy läpi yksinkertaisen BSP-luolaston kehittämisen Godot’n laattakarttaan, mutta ei käy kaikkea läpi ja loppuu osittain kesken, mutta auttoi minua pääsemään alkuun mallin pohjassa. Shieldsin dokumentaatio loppuu siis käytävien luomiskohtaan ja kannustaa jatkokehitykseen ideoilla. Kyseinen dokumentaatio oli hyvä, koska se käy nopeasti läpi, mitä binäärinen avaruuden osiointi tekee, ei uppoudu liikaa Godot’n käyttöön aloittelijatasolla, ja se hyödyntää Godot’n uusia laattakarttoja erityisesti koodin puolella, mikä antaa hyvän pohjan sen käyttämisen selvittämiseen automaattikartoituksen avulla koodin kautta.

Ensimmäiseksi on aloitettava puumallin puun luominen, joten tarvitaan runko, ja ideana on, että rungolla voi olla joko 0 tai 2 lasta. Aloitus tehtiin siten, että luotiin 2D-pääsolmu näyttämön vanhemmaksi ja sen lapseksi laattakarttasolmu. Algoritmin kehittämiseksi pääsolmua laajennettiin ja tämän laajennuksen sisällä luotiin algoritmi oksien luomiseen, jota hyödynnetään myöhemmin pääsolmussa siten, että kyseistä algoritmia käytetään täyttämään laattakarttaan laattoja sen sääntöjen mukaan.

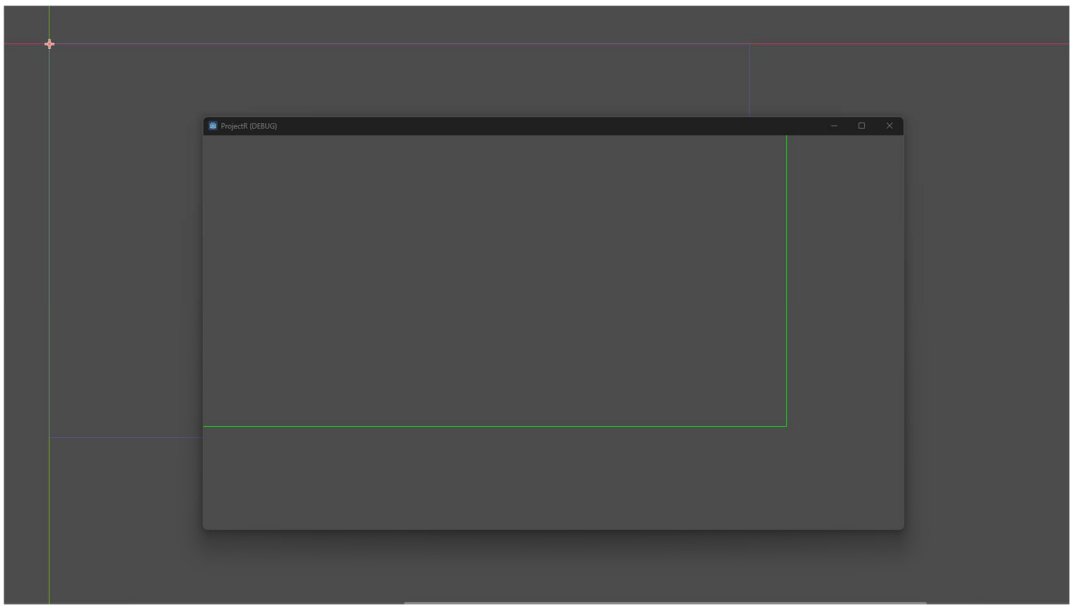
Yksinkertainen toteutusfunktio on luotu, joka käynnistetään alussa. Siinä määritellään, että sijainti ja koko ovat solmun oma sijainti ja koko. Godot’ssa "minä" on pääsääntöisesti sama kuin "tämä" muissa ohjelmointikielissä, mikä antaa mahdollisuuden käyttää objektin metodia myös muissa osioissa.

Seuraava koodin pätkä demonstroi Ohjelma 1 lehtien hakeminen GDScriptillä kirjoitettu ohjelmakoodi, miten lehdet haetaan yksinkertaisesti. Jos lehtiä ei ole, kuten alhaalla olevassa esimerkissä, se antaa vain koko alueen, minne se loisi lehdet, jos lehtiä olisi.

Ohjelma 1 Lehtien hakeminen GDScriptillä kirjoitettu ohjelmakoodi.

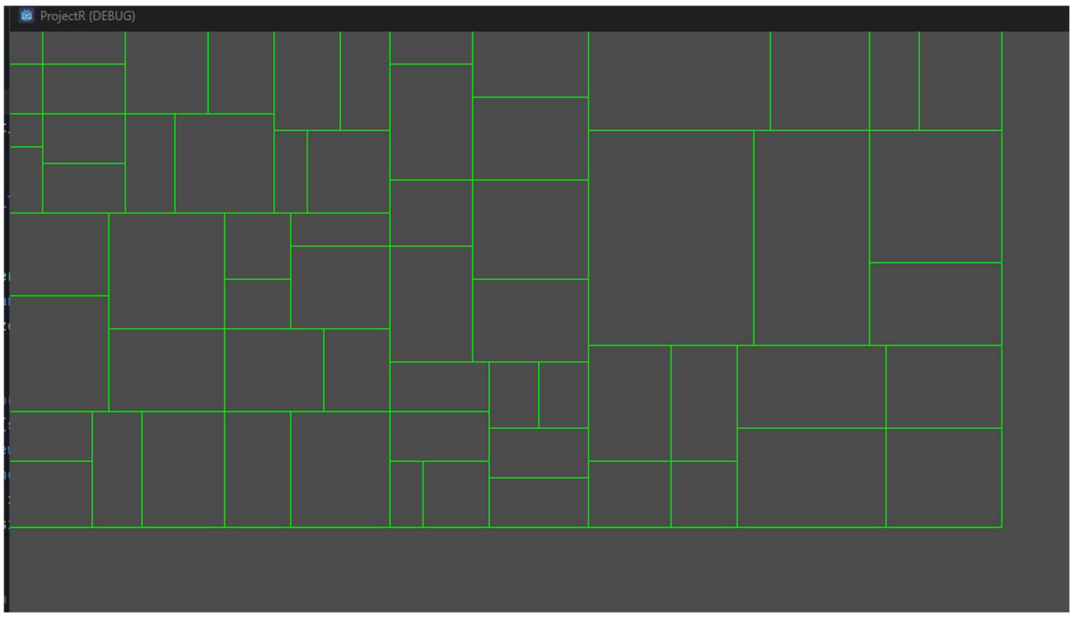
```
func get_leaves(): #Hae lehdet
            if not(left_child && right_child):
                        return [self]
            else:
                        return          left_child.get_leaves()          +
right_child.get_leaves()
```

Ensimmäisessä testissä ei ole yhtään lehtiä, joten palautetaan vain laattakartan koko. Alue piirretään siis vain ulos, minkä sisälle lehdet voivat muodostua. (Kuva 3.)



Kuva 3. Esimerkki, ilman dataa.

Kyseinen ohjelma (Ohjelma 2) on lopullisesta algoritmista, jossa oksat jaetaan 40 %- 60 % välillä, luoden melkein neliöitä, mutta ei aivan. Päädyttiin suhteellisen tasaisiin arvoihin pääsääntöisesti sen takia, että huoneiden haluttiin olevan suunnilleen samankokoisia erityisesti testaamista varten. Tilanteita alkoi syntyä, joissa jotkut huoneet olivat vain pari laattaa leveitä tai korkeita, luoden liian huonoja huoneen osia tai liian monta pieneen alueeseen, mikä ei toiminut pelin kannalta. Myös koodin osuudessa kaikki ovat suurelta osin kokonaislukuja, koska se auttaa suuresti laattakarttaan sijoittamisessa. Alhaalla oleva kuva näyttää, miltä algoritmin luomat oksat näyttävät (Kuva 4.).



Kuva 4. Miltä näyttää BSP malli datan kanssa.

Ohjelma 2 Algoritmi

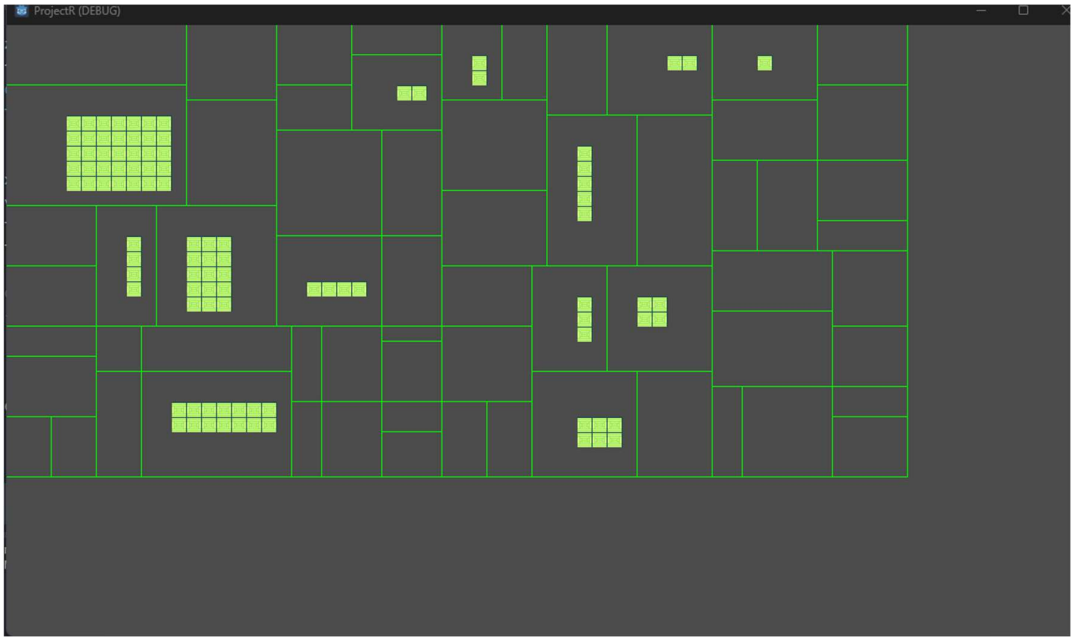
```
func split(remaining, paths):

	var rng = RandomNumberGenerator.new()

	var split_percent = rng.randf_range(0.40,0.60) #Oksien katkaisut tulee olemaan 30%-70%

	var split_horizontal = size.y >= size.x #tarkastetaan että ei ole korkeampi kuin leveä

	if(split_horizontal):

		#horizontal - vaakasuora

		var left_height = int(size.y * split_percent)

		left_child = Branch.new(position, Vector2i(size.x, left_height))

		right_child = Branch.new(

			Vector2i(position.x, position.y + left_height),

			Vector2i(size.x, size.y - left_height))

	else:

		#Vertical - Pystysuora

		var left_width = int(size.x * split_percent)

		left_child = Branch.new(position, Vector2i(left_width, size.y))

		right_child = Branch.new(

			Vector2i(position.x + left_width, position.y),

			Vector2i(size.x - left_width, size.y))
```

Oksien sisälle on luotava laattoja, minkä vuoksi erilaisia sääntöjä piti muodostaa ja sopivat mitat ja säännöt piti löytää kokeilun ja virheiden kautta, erityisesti sen suhteen, miltä pelin haluttiin näyttävän. Tässä käytetään itse tehtyä laattakarttaa, jonka luomiseen perehdytään tarkemmin seuraavassa osiossa, koska laattakartalle piti luoda omat törmäykset ja järjestelmä automaattikartoitukseen.

Ensiksi piti luoda täytteet, jotta huoneiden välille tulisi oikeat välit eikä koko laattakartta olisi vain yhtä lattialaattaa. Tätä varten piti kokeilla eri täytearvoja, ensimmäiset arvot olivat 2 ja 3 välillä, mutta se johti ongelmaan, jossa joihinkin laattoihin ei vain muodostunut mitään, varsinkin kun solujen koot olivat aiemmin 30 % - 70 %, mikä ei toiminut lainkaan.

(Kuva 5) huoneet täytetään leikkaamalla huoneista reunat. Esimerkissä täyte on ilmeisesti liian suuri, mikä huomataan siitä, että pieniin ruutuihin ei muodostu laattoja. Täyte on asetuksissa yhdestä kolmeen laattaan, eli se valitsee jonkin

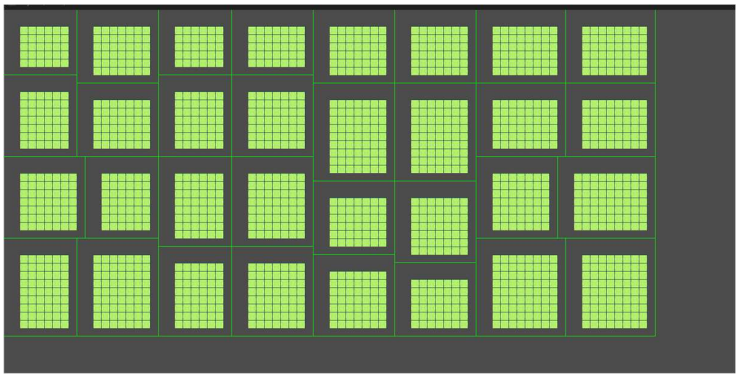
luvun tällä välillä ja sen verran laattoja ei satunnaisesti suunnasta riippuen muodostu reunoille. Yksi laatta on 16 pikseliä.



Kuva 5. Ruutujen täyttäminen laatoilla.

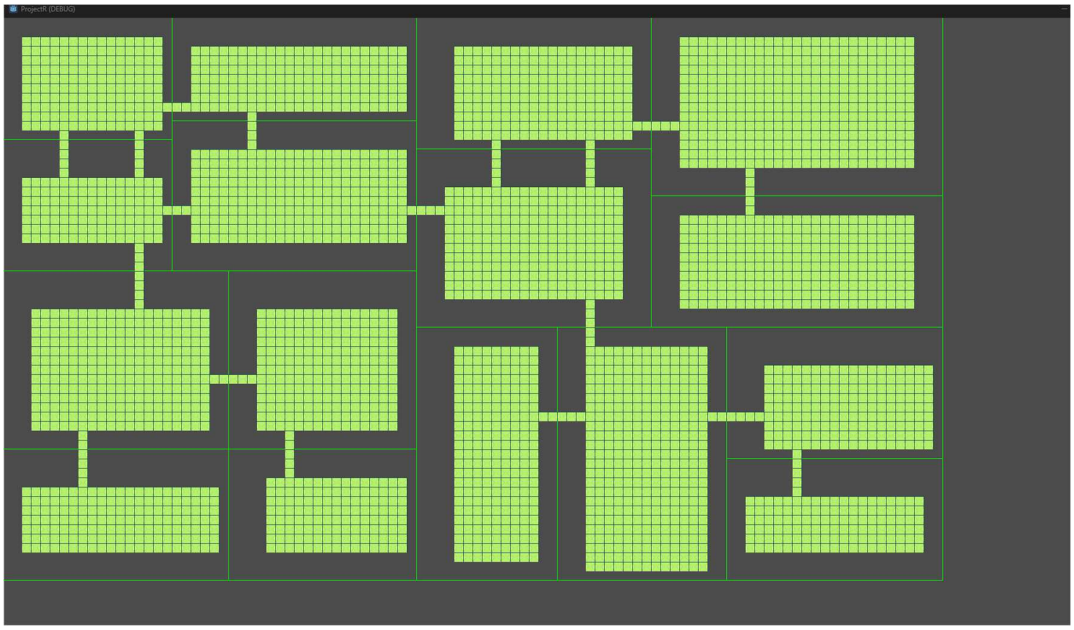
Kuva 6. Toimivat mitat ruuduille 45 % - 55 % mitoilla testattiin mahdollisimman lähellä olevia arvoja saadakseen ensin pääsääntöisesti melkein neliöitä. Kun käytävät lisätään, järjestys muuttuu suuresti. Myös 40 % - 60 % mahdolliset leikkauskohteet pienennettiin. Haluttiin pitää huoneet mahdollisimman tasakokoisina pienellä heittovälillä, jotta ei syntyisi niin sanottuja kolmesta neljään laatan huoneita, jotka ovat niin pieniä, että ne aiheuttavat mahdollisia automaation ongelmia, kun laattojen päälle luodaan maasto automaattisesti kartoittamaan laatat joko seiniin tai lattioihin.

Kun päädyttiin arvoihin 40 % - 60 %. Alettiin luomaan huoneiden välille käytäviä, ne luodaan algoritmissa pääsääntöisesti hyödyntäen tätä koodin pätkää, jossa kerääntyy keskipisteet, käytävien leikkausten määrä määrittyy siten, että ne leikkaantuvat 3 kertaa huoneiden määrän mukaan. Luoden huoneiden keskipisteistä yhdistyviä polkuja



Kuva 6. Toimivat mitat ruuduille.

Kuva 7. Käytävät kartassa näyttää, miten käytävät pääsääntöisesti muodostuvat nyt luotuihin huoneisiin. Eri reittejä muodostuu huoneiden välille, joita pitkin pelaaja pystyy liikkumaan huoneesta huoneeseen etsiessään maalia. Käytävät etsivät pääsääntöisesti solujen/ruutujen keskipisteet ja luovat niiden välille yhteyksiä joko x- ja y-akselin suuntaan (Ohjelma 3). Mitä enemmän jaettuja alueita, sitä enemmän käytäviä. Kolme vaikutti hyvältä määrältä, jossa ei mennä liiansokkeloiseksi miettiä, miten päästä minnekin, eikä joka huoneesta pääse kaikkialle.



Kuva 7. Käytävät kartassa.

Ohjelma 3 Käytävien generointi

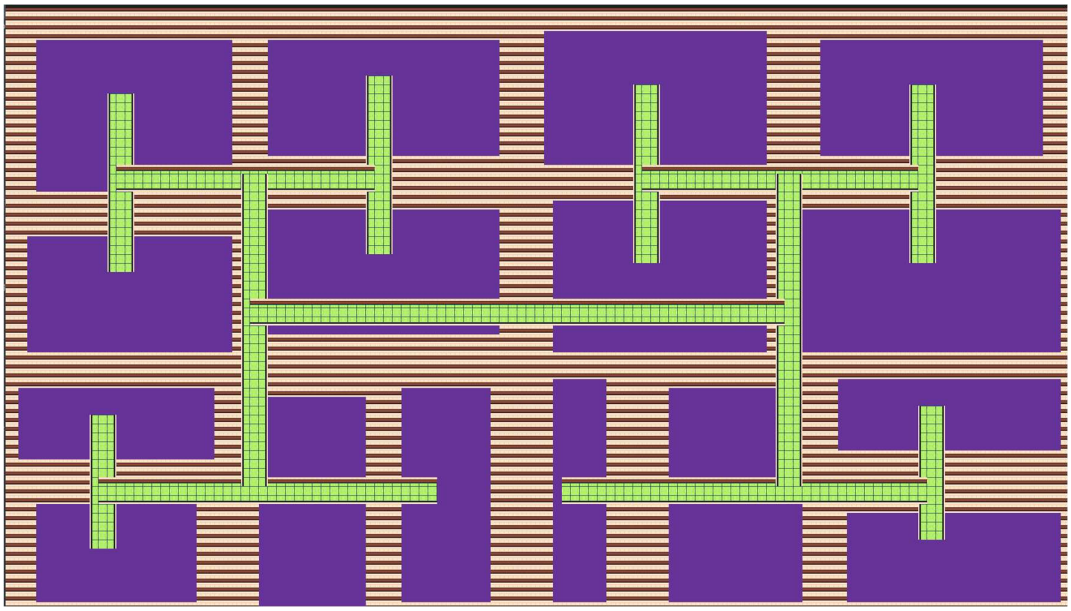
```
if(remaining > 0):

                    left_child.split(remaining - 1, paths)

                    right_child.split(remaining - 1, paths)

          paths.push_back({'left':  left_child.get_center(),  'right':
right_child.get_center()})

          pass
```

Käytävien välille luotiin seinät (Ohjelma 4) myöskin ohjelmassa määritetään, miten, piirrämme algoritmin kautta käytävät.

Seinien luominen tehdään siten, että ennakkoon on määritelty käytävät 3 laattaa leveiksi ja seinien päälle lisätään automaattisen laattojen sijoittaminen luoden seinät, jolloin se näyttää havainnollistavalta kuvalta (Kuva 8).

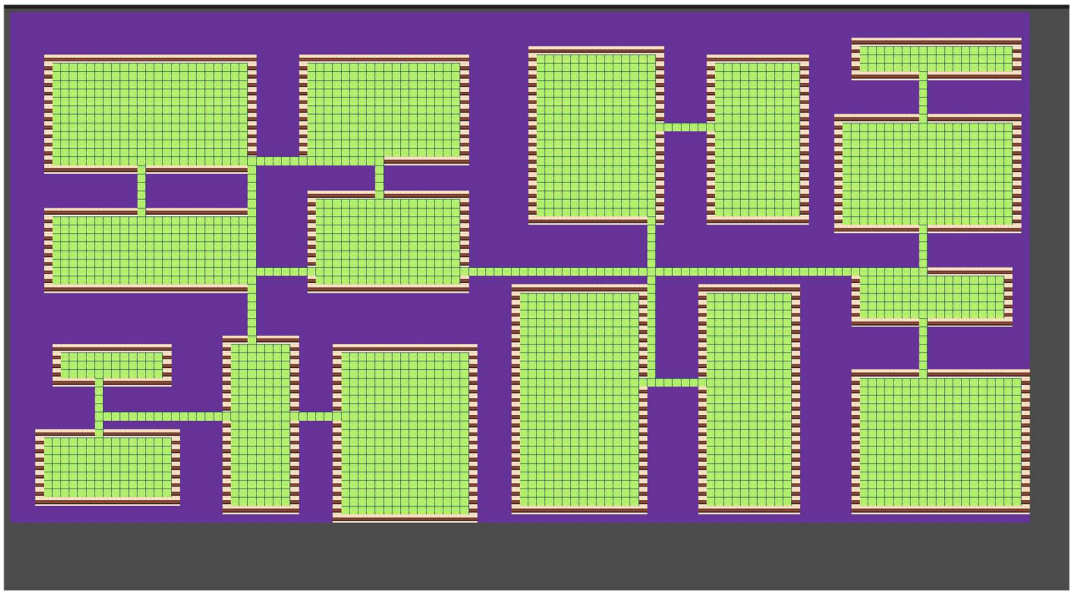
Ohjelma 4 Käytävien generointi ja laattojen automaatio

```
for path in paths:
                    if path["left"].y == path["right"].y:
                                #horizontal
                                        for i in
range(path["right"].x - path["left"].x):
#figuroi parempi tapa esim patternit käytävän seinille
                                tilemap.set_cell(0,
Vector2i(path["left"].x+i,path["left"].y -1), 0, Vector2i(17, 12))
#seinä
          tilemap.set_cell(0, Vector2i(path["left"].x+i,path["left"].y
+1), 0, Vector2i(16, 14)) #ala seinä
          tilemap.set_cell(0,
Vector2i(path["left"].x+i,path["left"].y), 0, Vector2i(17, 13))
          Room_AutoMap.append(Vector2i(path["left"].x+i,path["left"].
y -1))
          Room_AutoMap.append(Vector2i(path["left"].x+i,path["left"].
y +1))
          Room_AutoMap.append(Vector2i(path["left"].x+i,path["left"].
y))
```



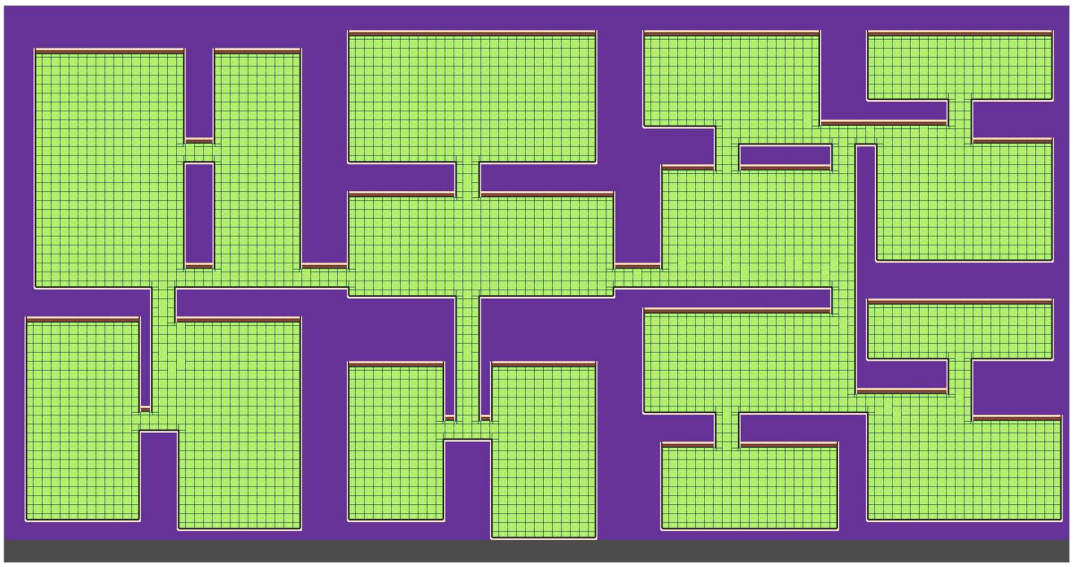
Kuva 8. Havainnollistava kuva käytäväistä, jossa huoneet ovat poistettu.

Seinien luomiseen luotiin pääsääntöisesti joka suuntaan yksi isompi laatta huoneeseen ensin, jonka päälle sitten sijoitettiin lattia. Tämä antoi mahdollisuuden sille, että jos käytävä tulisi huonossa kulmassa, ohjelma osaisi silti kartoittaa sen kuntoon (Kuva 9).



Kuva 9. Huoneiden ympärille seinien lisäys.

Tässä automaattikartoitus on sijoitettu luodun huoneen päälle. Tätä varten kaikki laatat kartoitettiin taulukkoon ja Godotille annettiin käyttöön valmiiksi määritetty maasto automaattikartoitusta varten, ja se kartattaisi kaikki laatat uudelleen. Tämä loi alhaalla näkyvän lopputuloksen huoneille (Kuva 10).



Kuva 10. Automappaaminen.

Myös on pystyttävä sijoittamaan esimerkiksi pelaaja, vihollisia ja mahdollisia uusia laattasettejä valmiiseen huoneeseen jatkokehityksen kannalta, vaikka huoneille, objekteille ja kaikelle muulle. Tätä varten luotiin funktio, jota voidaan hyödyntää ohjelma 5 koodi, joka sijoittaa halutun kohteeseen. Pääsääntöisesti on kaksi funktiota, joista toinen sijoittaa satunnaisen laatan objektin ja toinen sijoittaa objektit yksitellen taulukosta (Ohjelma 6) jos on esimerkiksi taulukko kaikille vihollisille, huoneen lisäyksille ja muulle, jolloin satunnainen lattialaatta synnyttää esineitä.

Pelaaja ja vihollinen voivat teoriassa ilmestyä samalle laatalle, mutta pelaajaan verrattuna vihollinen aktivoituu peliin vasta vähän ajan kuluttua. Ratkaisu tähän ongelmaan on ottaa laatta, jolle pelaaja ilmestyy pois laskuista tai korvata ilmestymislaatat eri laatoilla, joilla on eri atlasnumero, jolloin niihin ei voi enää ilmestyä. Atlasnumero tarkoittaa laattakartassa määritettyä id-numeroa, joka on

laatan hakemista varten. Eli kyseinen laatta otetaan pois taulukosta, johon kerätään kaikki mahdolliset laatat, ja myös pois atlas-id-arvosta, jota haetaan.

Ohjelma 5 Ohjelmistokoodi objektin sijoittaminen laatalle

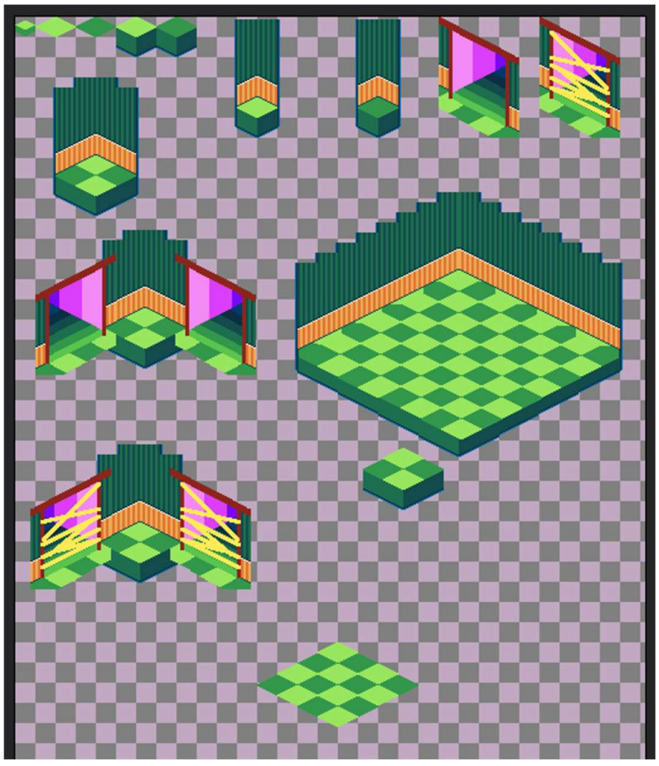
```
#Functioon handlaamaan random sijoittaminen, jotta voidaan mahdollisesti hyödyntää vihollisille.

func place_on_tile(object):

	Random_Tile += tilemap.get_used_cells_by_id(0, 0, Vector2i(17, 13)) #hae kaikki lattia tilet

	var random_pos = Vector2i(Random_Tile.pick_random()) * 16

	object.position = random_pos #sijoita random tileen objekti

	pass
```

Ohjelma 6 Ohjelmistokoodi taulukon sijoittaminen laatalle

```
func place_array_on_tile(array):

	Random_Tile += tilemap.get_used_cells_by_id(0, 0, Vector2i(17, 13)) #hae kaikki lattia tilet

	for i in array:

		var random_pos = Vector2i(Random_Tile.pick_random()) * 16

		i.position = random_pos #sijoita random tileen objekti arraysta

	pass
```

4 Pelinkehitys

Pelin ideana on luoda toimiva pohja luolastoseikkailupelille. Alkuperäinen idea oli luoda isometrinen peli, joka hyödyntäisi alhaalla näkyvää laattakarttaesimerkkiä (Kuva 11). Päädyttiin kuitenkin perinteiseen yläkuvanäkymään, koska isometrisen laattakartan automaattikartoituksen saaminen toimimaan Godotissa koettiin liian työlääksi erityisesti koodin kautta. Lisäksi isometrisen hyödyntämiseen ei löydetty paljon dokumentaatiota ja pääsääntöisesti piti hyödyntää perinteistä ristiruudukkoa ja yrittää muovata sen toimimaan. Tämä ei tuntunut johdonmukaiselta alusta lähtien, kun jokaiseen laatan sijoitukseen joutui lisätä laskuihin tai funktioihin aina, että kaikki on käännetty 45 astetta. Tämä käännös myös piti olla aina kun yritti lisätä jotain uutta ottaa huomioon myös.

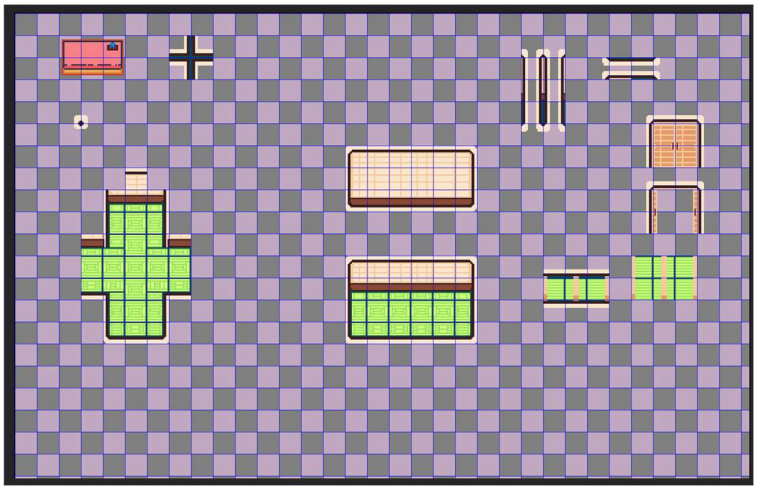


Kuva 11. Alkuperäinen pelin visuaalinen suunnitelma huoneelle.

Peliin luodut spritet ja laattakartta on luotu hyödyntämällä työkalua nimeltä Asesprite. Se on maksullinen pikselitaiteen työkalu, jonka kautta voidaan viedä peliin Spritet ja laattakartta valmiiksi käytettävässä muodossa pelimoottorille, jolloin kaikki data Asepritessa, esimerkiksi animaatioille, säilyy myös pelimoottorissa. Kaikki pelissä olevat varat ovat luotuja, ja tämä osoittautui hyödylliseksi oppimisen kannalta, mutta myös aikaa vieväksi, mikä hidasti suuresti pelinkehitysosuutta ja johti usein jatkuviin korjauksiin, jos virheitä tapahtui tai mitat olivat vääriä.

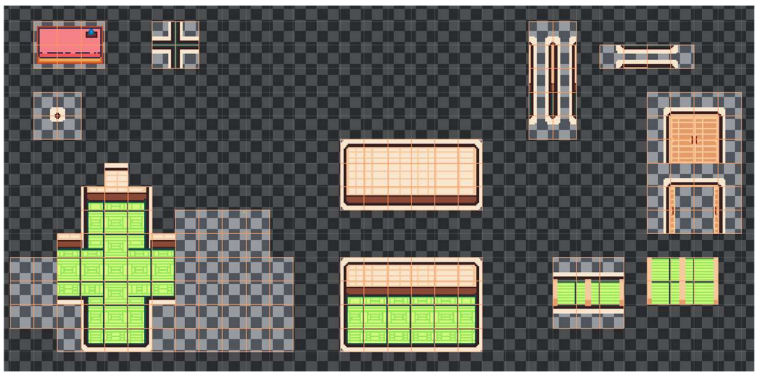
4.1 Laattakartta ja automaatio

Ensimmäinen vaihe oli laattakartan luominen, jota voitaisiin hyödyntää luolaston generoinnin aikana ja automaattikartoituksen määrittelyssä. Laattakartta on luotu Kuvan 12 mukaan, ja siitä ei hyödynnetty kaikkia laattoja enää lopullisessa pelissä, mutta siihen tehtiin toimiva maasto eli automaattikartoitus. Maaston tekeminen ei ole haastavaa, mutta ei myöskään helppoa. Paljon muutoksia piti tehdä saadakseen sen toimimaan suoraan koodista, joten mitä yksinkertaisempi rakenne, sitä helpommin se toimi. Tämä on syy, miksi päädyttiin selkeisiin suorakulmiohuoneisiin.



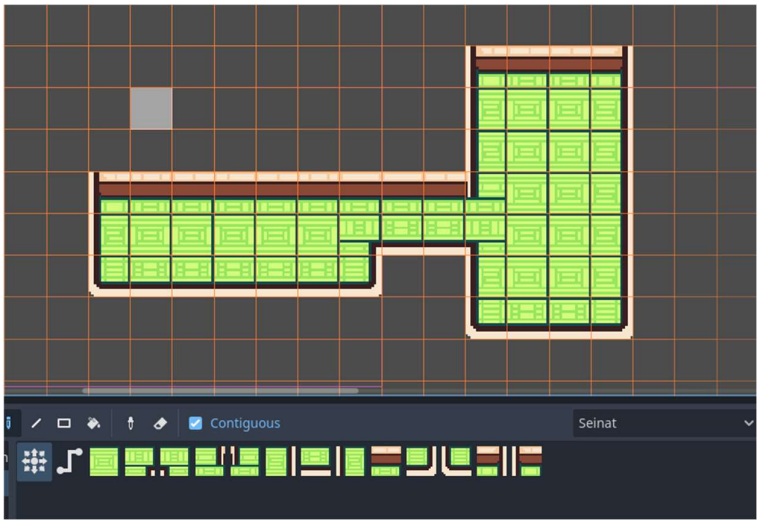
Kuva 12. Nykyinen laattakartta.

Laattakarttaan luodaan siis atlas-kartta (kuva 13), jossa valitaan kaikki laatat, joita voidaan käyttää, kuten alhaalla näkyvässä esimerkissä. Ensimmäinen laatta olisi atlaksessa laattakartassa sijainnissa x2 ja y1 eli (2,1).



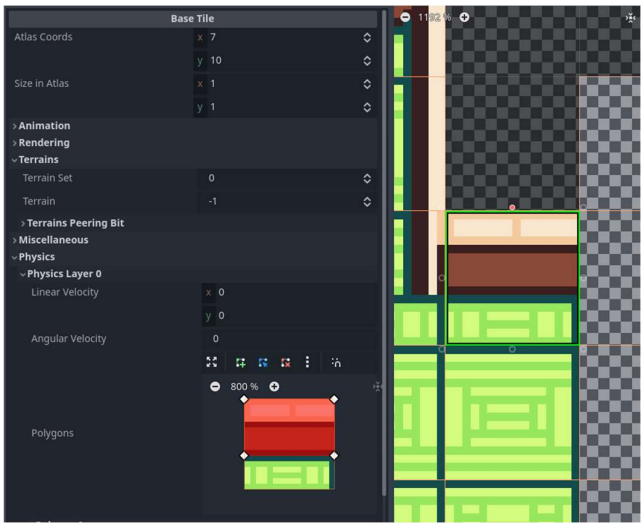
Kuva 13. Miten valitaan laatat ja luodaan atlas id niille.

Automaattikartoitus (Kuva 14) tapahtuu siten, että piirretään jokaiselle laatalle, jota käytetään automaatioon huoneen kartoittamiseen alueet, minkä mukaan luokitellaan lattiat ja seinät. Nämä piirretyt alueet pitää yhdistää oikein, jonka seurauksena pelimoottori osaa piirtäessä yhdistää laatat toisiinsa, muodostaen seiniä ja käytäviä.



Kuva 14 Yleinen automaattikartoitus.

Jokaiseen seinälliseen laattaan määritellään rajat törmäysmahdollisuudelle. Tämä toteutetaan siten, että on tehtävä manuaalisesti piirtämällä rajat. Tämä on todella hidasta ja vie aikaa, mutta se on tehtävä huolellisesti pikselin tarkkuudella, jotta ei synny yllättäviä kohtia, joista pelaaja pääsee joko kävelemään läpi tai jää jumiin (Kuva 15).



Kuva 15. Törmäyksen tekeminen laattaan.

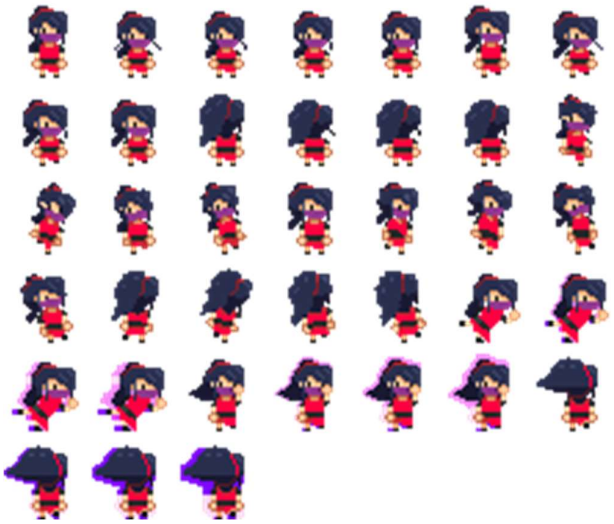
4.2 Assetti

(AdamCYounis, 2024) YouTube-kanava oli iso apu spritejen tekemiseen. Vaikka hänen videonsa suuresti on Unity keskittyviä silti samat teoriat pätevät yleisesti spritejen tekemiseen. soittolista pikselianimaatioon liittyen varsinkin pätee aika universaalisti ja Nä

mä olivat suuresti pohja selvittä, miten tehdä Sprite assetteja peliin.

Pelaajahahmon visuaalisen ulkonäön on tarkoitus muistuttaa Kunoichia eli naispuolista Shinobia tai naispuolista ninjaa (yamatomagazine, 2020). Inspiraationa hahmojen ulkonäköön otettiin pääsääntöisesti japanilaisista kansanperinteistä, koska laattakartasta tuli mielestäni tatamilattian näköinen, joka on perinteisesti japanilainen. Tarkoitus oli siis pitämään ulkonäkö melko

universaalina keskittyen Japanin kansantarinoihin pienellä popkulttuurin joustovaralla.

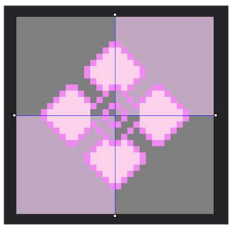


Kuva 16. Pelaaja.

Projektin ensimmäiseksi viholliseksi on valittu kasa-obake tai karakase-obake (MythologyWorldwide, 2024), joka on japanilaisen kansanperinteen mukainen haamu. Tämä hahmo muistuttaa ulkonäöltään sateenvarjoa tai lamppua. Tämä vihollinen on valittu, koska sen katsottiin sopivan hyvin ensimmäiseksi viholliseksi. Sen tehtävänä on yksinkertaisesti päästä tarpeeksi lähelle pelaajaa.

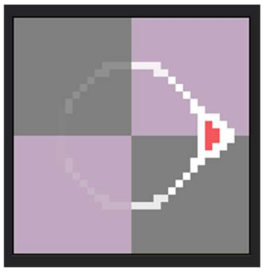
Kuva 17. Vihollinen.

Projektiin on myös kehitetty oma hiiren ulkonäkö (Kuva 18. Hiiri), joka tekee hiiren sijainnin näkemisestä selkeämpää, varsinkin kun käytössä oleva tietokoneen hiiri on melko pieni.



Kuva 18. Hiiri.

Projektissa on luotu indikaattori pelaajan alle (Kuva 19. Pelaajan alhaalla osoitin hiiren suuntaan). Indikaattorin tarkoituksena on osoittaa hiirtä kohti, mikä auttaa hahmottamaan hiiren sijaintia. Jos jatkokehityksessä halutaan lisätä ohjaintuki, indikaattorin pyöriminen on liitetty ammuksen ampumiseen. Tämän avulla ohjaimen tuki voitaisiin kehittää jatkossa.



Kuva 19. Pelaajan alhaalla osoitin hiiren suuntaan.

Pelinkehityksessä osoitinta on hyödynnetty siten, että kaikki ampumiseen liittyvät tärkeät tiedot on kiinnitetty siihen (Ohjelma 7). Tämä mahdollistaa ammusten kääntymisen oikein hiiren mukaan osoittimen avulla (Ohjelma 8). Jmbiv YouTubeen (2020) tekemää video sarjaa ylhäältä kuvatun ammunta pelin kehittämiseen on pääsääntöisesti hyödynnetty video osia 3-5. Tämän kautta on selvitetty toimiva tapa saada ammukset kääntymään hiirensuuntaan. Tässä tapauksessa osoitin toimii periaatteessa pelaajana, joka pystyy kääntymään vapaasti joka asentoon, kun taas pelaaja pystyy kääntymään vain 4 suuntaan.

Kyseisen video sarjan ongelma on, että se on kehitetty Godot 3 varten, joten siihen ei voi luottaa täysin. Suurin osa koodista joudutaan kääntämään Godot 4 mukaan.

Ohjelma 7 Ampuminen

```
func shoot():

            var bullet_instance = Bullet.instantiate()

            var    direction    =    (shoot_rotation.global_position    -
shoot_point.global_position).normalized()

            emit_signal("player_fired_bullet",          bullet_instance,
shoot_point.global_position, direction)
```

Ohjelma 8 Miten Godotissa osoittaa hiireen.

```
func _physics_process(delta): #osoita hiireen

            look_at(get_global_mouse_position())
```

Ammus, mitä pelaaja heittää on heittoveitsi nimeltä Kunai (Kuva 20) sen takia, koska se on 1500 luvulta lähtöisin oleva japanilainen teräsase ja tunnettu yleisesti ninjojen suhteen (Ashlybrine, 2024).

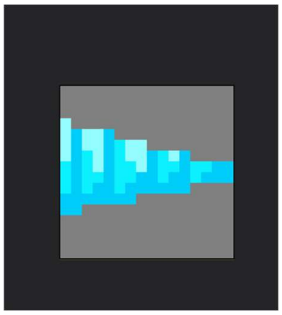
Kuva 20. Pelaajan heittämä Kunai.

Jatkokehityksen kannalta on harkittu lisää vihollisia, mutta projektissa on rajoituttu vain pelaajaan ja yhteen viholliseen. Tämä johtuu siitä, että kaiken animaation tekeminen koettiin liian aikaa vieväksi. Animaation laadun vaihtelu tarkasti animoidusta siihen, että vain PNG-kuva liikkuu näytöllä, koettiin oudoksi.

Toinen vihollistyyppi siis jäi kesken, koska sen oli tarkoitus perustua ensimmäisen vihollisen tekoälyyn. Nykyisen järjestelmän koettiin kuitenkin hahmottavan vihollisen osumat ja muut paremmin, joten ennen uuden vihollisen lisäämistä haluttiin luoda vakaampi pohja (Kuva 21).

Kuva 21. Toinen vihollinen.

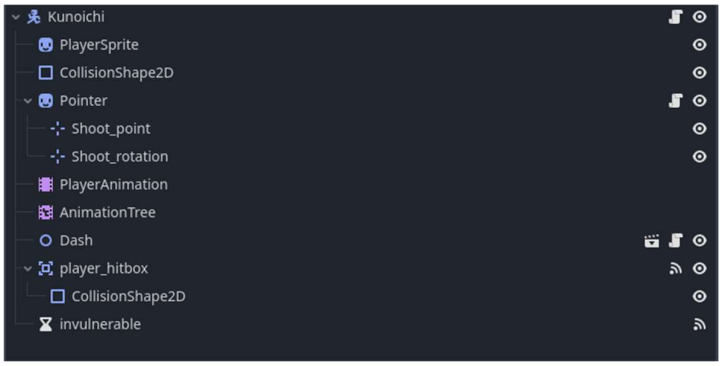
Mahdollinen eri ammuksen Sprite (Kuva 22). Ideana oli, tehdä kyseisestä (Kuva 21. Toinen vihollinen) aavemaisesta hahmosta toinen vihollistyyppi, joka ampuu pidemmän matkan päästä pelaajaa, eikä vain jahtaa aivottomasti Tällöin pelaaja joutuisi miettimään, muuta kuin karkuun juoksemista vain jahtaavilta vihollisilta, ammusten takia ja se myöskin kannustaisi pelaajaa käyttämään syöksy toimintoa saamaan suojauksen osumilta sekunnin ajaksi väistääkseen osumat ammuksilta.



Kuva 22. Vihollisen ammus.

4.3 Peli

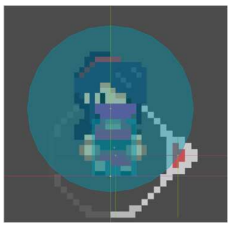
Godotissa jokainen näkymä on periaatteessa oma solmu, johon lisätään lapsisolmuja. Pelaajasolmun alla hoidetaan useita asioita. Näitä ovat pelaajan Sprite ja sen animaatiotiedot, törmäykset, osoitin eli aikaisemmin tehty indikaattori pelaajan alla, animaation hallinta, animaatiopuu, logiikka syöksylle sekä ajastin, joka määrittää, voiko pelaaja ottaa osumaa. Lisäksi on olemassa Area2D, joka tarkistaa, onko vihollinen tarpeeksi lähellä pelaajaa, jolloin pelaaja ottaa osumaa.



Kuva 23. Pelaajan solmut.

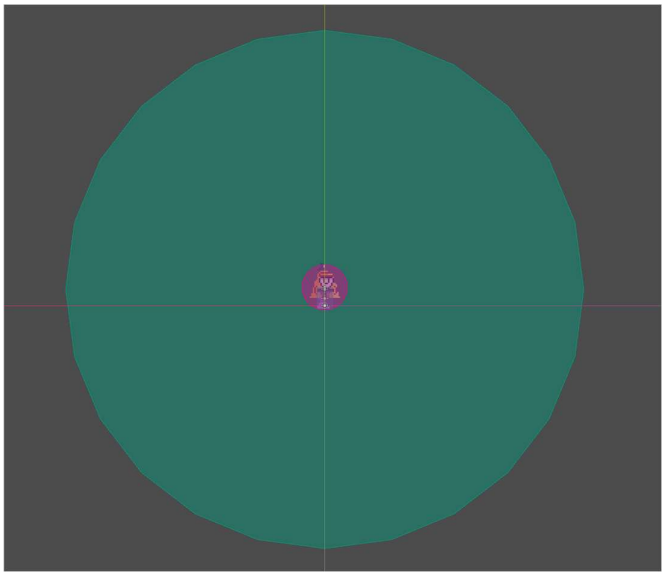
Pelaajahahmossa on monta funktiota kuten törmäystunnistus kehossaan, sprite-grafiikkaan kiinnitettyä animaatio tiloja, joista on logiikka animaation vaihtamiselle tai kuvakehyksien vaihtamiselle tietyn tilan mukaan. Lisäksi pelaajahahmon alla on pyörivä osoitin, jossa on 2D-merkit kohdille, joiden

mukaan ammutaan projektiileja. Nämä merkit laskevat sen, että pelaaja pystyy ampumaan mihin tahansa suuntaan ja osaavat kääntää projektiilin oikein päin (Kuva 24).



Kuva 24. Pelaaja.

Vihollisen nykyinen versio on tehty siten, että sillä on erittäin yksinkertainen tekoäly. Kun pelaaja menee sisälle vihreään ympyrään (Kuva 25), vihollinen alkaa seurata pelaajaa tietyllä nopeudella. Jos pelaaja liikkuu ulos alueelta, vihollinen jäätyy paikoilleen. Jatkokehityksen kannalta olisi tarkoitus parantaa tätä tekoälyä hyödyntämällä Navigointiverkkoa (godotengine, 2023), jolloin vihollinen osaisi väistää paremmin esimerkiksi seiniä. Tämän toteuttamiseksi pitäisi selvittää, miten määrittää navigointiverkko jälkikäteen eikä suoraan pelin alussa, koska pelin taso generoituu pelin alussa eikä ole valmis navigointiverkon sijoittamiselle kehitystilassa.

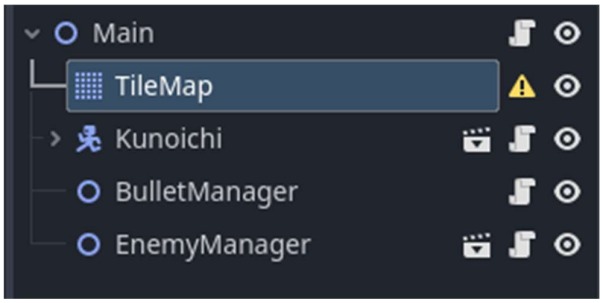


Kuva 25. Vihollisen ja havaitsemisalue.

Vihollisen solmuun (Kuva 26) kuuluu samaan tapaan kuin pelaajaan solmuihin, animaation manageri, animaatiopuun manageri ja alueet pelaajan havaitsemiseksi. Jos pelaaja osuu viholliseen ammuksella, alue tunnistaa myös tämän. Lisäksi vihollisen kehossa on törmäyksenestäjä estämässä seinän läpi kävelyn.

Kuva 26. Vihollisen solmut.

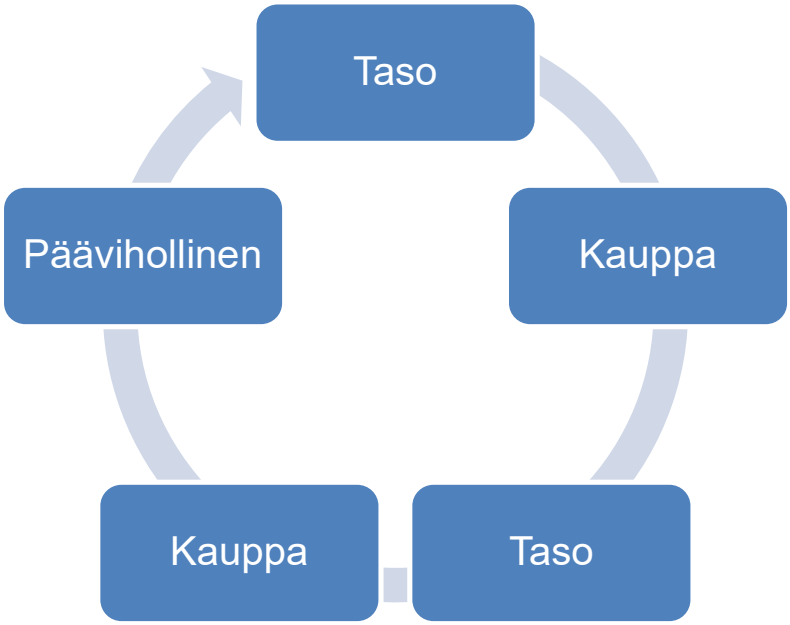
Pelin kannalta pääkohtaukseen (Kuva 27) sijoitetaan itse laattakartta, johon luodaan taso, Kunoichi eli pelaajahahmolle, luotienhallinta, joka käsittelee projektiilien luomisen, jotta ei synny ongelmia objektin ja sen lapsen välillä. Viimeisenä on Vihollisten Hallinta, joka sijoittaa viholliset karttaan ja muut objektit. Tätä käytetään tällä hetkellä melko paljon kaiken sijoittamiseen karttaan, joten nimi ei sinänsä ole enää kuvaava. Parempi nimi olisi Ilmestymis-Hallinta.



Kuva 27. Pää näkymän solmut.

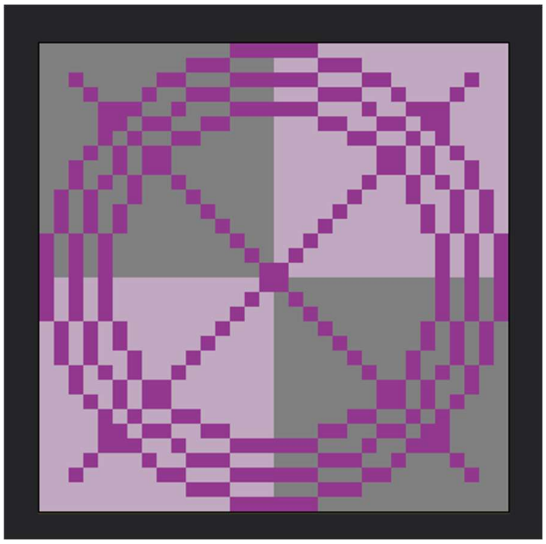
Alkuperäinen peli-idea oli, että on taso, johtaa kauppaan, mistä pääsee takaisin eri tasoon (Kuvio1) ja tämä toistuu niin pitkään, kunnes päädytään

päävihollisen. Kuitenkin projektin kehitysvaihe rajoitettiin vain tasoon, koska kaupan kehittäminen omalla tietokannalla kaikille esineille ja oman rahajärjestelmän luominen peliin koettiin liian aikaa vieväksi. Myös päävihollisen osalta koettiin samoin, koska se vaatisi kaupan koko kehityksen alusta lähtien, jotta tiedettäisiin, kuinka suureksi asettaa sen arvot elämän ja vahingon osalta.



Kuvio 1 Pelin kulun idea.

Pelissä loppu pisteenä toimii tällä hetkellä kyseinen kuva 28. Peli sijoittaa kyseisen grafiikan johonkin tilen päälle dungeonin ja jos pelaaja astuu siihen, peli on läpi ja loppuu. Jatkossa olisi ideana se, että tämä veisi pelaajan kauppaan, jossa voidaan kehittää hahmoasi rahan vastineella mitä saat, kun eliminoit vihollisia.



Kuva 28 Nykyinen grafiikka loppupisteelle.

5 Lopuksi

Opinnäytetyön tavoitteena oli tutustua Godot 4 -pelimoottoriin ja selvittää sen vahvuuksia ja heikkouksia sekä tutkia, miten proseduraalista generointia voidaan hyödyntää pelissä ja mahdollisesti tehdä sen ymmärtämisestä helpompaa. Tavoitteessa voidaan sanoa onnistutun, sillä selvisi hyvin, kuinka suuri heikkous dokumentaation ja opetusmateriaalin puute pelimoottorissa todella on. Tämä hidasti projektin etenemistä merkittävästi. Samanaikaisesti havaittiin monia vahvuuksia, kuten se, että kaikki kehitetty omistetaan täysin ja sovelluksen asentaminen ei vaadi paljon, koska se toimii kannettavana laitteesta laitteeseen ilman asennus tarvetta, joten koko projektin voi helposti siirtää USB-tikulla toiselle tietokoneelle.

Proseduraalinen malli kehitettiin, mutta sen ymmärtäminen on edelleen haastavaa. Suurin osa jatkokehityksestä meni usein kokeilun ja virheiden kautta. Tiedetään pääpiirteittäin, mitä BSP tekee: se jakaa rungon moneen oksaan, ja näiden oksien keskialueet ovat lehtiä tai soluja, joihin generoidaan laatat. Käytävien osalta etsitään näiden solujen keskipisteet ja yhdistetään ne samalla jakotyylillä jakamalla ne osiin y- ja x-akselin suunnissa. Koodista voidaan todennäköisesti näyttää, missä mikäkin tapahtuu, mutta ei olla täysin varmoja asiasta. Jatkossa olisi hyvä kehittää samanlainen idea eri algoritmilla ja katsoa, avaisiko se kokonaiskuvaa selkeämmin.

Jatkokehityksen kannalta on monia vaihtoehtoja. Eräs vaihtoehto on kokeilla erilaisia algoritmeja, jotka syötetään projektiin luomaan erilaisia luolaston ulkonäköjä. Alun perin ajateltiin, jos pelaaja pelaisi tason, jonka jälkeen pelaaja pääsisi kauppaan ostamaan päivityksiä ja lopulta yrittäisi saada itsensä tarpeeksi vahvaksi päävihollista varten. Projektin mittakaavaa pienennettiin kuitenkin pelkkään tasoon, koska päävihollisen ja kaupan tekemisen koettiin vievän liikaa aikaa ja uudelleen generoituvan tason kehittäminen havaittiin olevan haastavampaa kuin suunniteltiin aikataulun kannalta. Jatkokehityksen kannalta voitaisiin kuitenkin alkaa kehittämään näitä kahta muuta osiota, jotka

puuttuvat, ja saada siten enemmän kokonaiskuvaa siitä, ovatko luolastot liian pieniä, ja saada myös enemmän käyttäjätestauspalautetta.

Nykyinen testaus on suurelta osin ollut projektin tekemistä, eli luodaan luolasto, tarkistetaan, onko generoinnissa virheitä, ja jos on, yritetään korjata ne. Luodaan pelaaja, liikutetaan pelaajaa tasossa ja lasketaan, meneekö liian kauan mennä pisteestä a pisteeseen b, ja jos menee, pienennetään luolaston generointialuetta ja yritetään mahdollisesti pienentää huoneiden kokoa. Testaus on suurelta osin ollut sitä, että yritetään saada kaikki tason mitat tuntumaan sopivalta pelaajalle. Mutta pelin kannalta olisi hyvä saada peli testattavaksi laajasti eri ihmisille paremman palautteen saamiseksi. Testauksen myötä voitaisiin selvittää tuntuuko luolasto liian pieneltä tai yksinkertaiselta pelaajille ja kuinka paljon eri mekaanikoita pitäisi lisätä peliin sen tuntuman parantamiseksi.

Lähteet

AdamCYounis. (2024). *Youtube*. Haettu 23. 4 2024 osoitteesta AdamCYounis: https://www.youtube.com/@AdamCYounis

AdamCYounis. (2024). *Youtube*. Noudettu osoitteesta Pixel Art Class: https://www.youtube.com/playlist?list=PLLdxW--S_0h4dlWUpl-TzBp-ulqK3NiM_

Ashlybrine. (22. 4 2024). *Medium*. Noudettu osoitteesta Unveiling the Kunai Knife: History, Design and Applications!: https://medium.com/@ashlybrine29/unveiling-the-kunai-knife-history-design-and-applications-0af0b16abdad

Atria. (14. 9 2023). *Godot Community*. Noudettu osoitteesta Godot 3 or Godot 4: Which Version Should You Choose?: https://godot.community/topic/83/godot-3-or-godot-4-which-version-should-you-choose

finto. (2024). *Algoritmi*. Noudettu osoitteesta https://finto.fi/cer/fi/page/?uri=http%3A%2F%2Ftieteentermipankki.fi%2Fwiki%2FClean_Energy_Research%3Aalgorithm

finto. (2024). *avoin lähdekoodi*. Noudettu osoitteesta https://finto.fi/afo/fi/page/?uri=http%3A%2F%2Fwww.yso.fi%2Fonto%2Fyso%2Fp17089

finto. (2024). *c#*. Noudettu osoitteesta http://www.yso.fi/onto/yso/p13591

finto. (2024). *finfo*. Noudettu osoitteesta http://www.yso.fi/onto/yso/p1978

finto. (2024). *finfo*. Haettu 2024 osoitteesta kolmiulotteisuus: http://www.yso.fi/onto/yso/p1978

finto. (2024). *finfo*. Noudettu osoitteesta Suomalainen asiasanasto ja ontologiapalvelu: https://finto.fi/udcs/fi/page/014277

finto. (2024). *finto*. Noudettu osoitteesta finto: http://www.yso.fi/onto/yso/p22765

Godot. (2024). *Godot Docs*. Noudettu osoitteesta Upgrading From Godot 3 to Godot 4: https://docs.godotengine.org/en/stable/tutorials/migrating/upgrading_to_godot_4.html

godotengine. (7. 11 2023). *Godot Docs*. Noudettu osoitteesta Using TileMaps: https://docs.godotengine.org/en/stable/tutorials/2d/using_tilemaps.html

godotengine. (11 2023). *Godot Docs*. Noudettu osoitteesta GDScript reference: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html

godotengine. (11. 7 2023). *Godot Docs*. Noudettu osoitteesta NavigationMesh: https://docs.godotengine.org/en/stable/classes/class_navigationmesh.html

godotengine. (2024). *Godot Docs*. Noudettu osoitteesta Node.

jmbiv. (3. 5 2020). *Youtube*. Noudettu osoitteesta How to make a Top-down Shooter in Godot | Playlist: https://www.youtube.com/playlist?list=PLpwc3ughKbZexDyPexHN2MXLliKAovkpl

MythologyWorldwide. (29. 2 2024). *Mythology Worldwide*. Noudettu osoitteesta The Enigmatic Kasa-Obake: The Umbrella Monster In Japanese Yokai Lore: https://mythologyworldwide.com/the-enigmatic-kasa-obake-the-umbrella-monster-in-japanese-yokai-lore/

Noveltech. (17. 7 2023). *Noveltech*. Noudettu osoitteesta Generating a 2d map using the Random Walk algorithm: https://mathworld.wolfram.com/RandomWalk2-Dimensional.html

Shead, M. (11. 2 2018). *blog.markshead*. Noudettu osoitteesta State Machines - Basics of Computer Science: https://blog.markshead.com/869/state-machines-computer-science/

Shields, J. (13. 8 2023). *jonoshields*. Noudettu osoitteesta Procedural Dungeon Generation In Godot: https://jonoshields.com/post/bsp-dungeon

symbolcraft. (2024). *symbolcraft*. Noudettu osoitteesta Mikä on BSP-puu?: https://www.symbolcraft.com/products/bsptrees/finnish/

unity. (2024). *Game development terms*. Noudettu osoitteesta unity: https://unity.com/how-to/beginner/game-development-terms

yamatomagazine. (11. 3 2020). *Yamato Magazine*. Noudettu osoitteesta Japan's Deadly Female Ninjas: Walkint the Path of the Kunoihi: https://yamatomagazine.home.blog/2020/03/11/japans-deadly-female-ninjas-walking-the-path-of-the-kunoichi/