3D-pelin kehittäminen Godot Engine -pelimoottorilla

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeenlinnan korkeakoulukeskus, tietojenkäsittelyn koulutusohjelma

Syksy, 2020

Mikko Luoto

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Hämeenlinnan korkeakoulukeskus

Tekijä Mikko Luoto **Vuosi** 2020

Työn nimi 3D-pelin kehittäminen Godot Engine-pelimoottorilla

Työn ohjaaja/t Lauri Salminen

TIIVISTELMÄ

Tämän opinnäytetyön tavoite oli tutustua 3D-pelin kehittämiseen Godot Engine pelimoottorilla. Samalla selvitettiin Godotin vahvuuksia ja heikkouksia pelimoottorina. Projektissa käytiin läpi Godotin perustoimintoja ja ominaisuuksia. Projektin idea syntyi halusta löytää tehokas ilmainen avoimen lähdekoodin pelimoottori korvaamaan Unity. Godot vaikutti hyvältä vaihtoehdolta, sillä se on yksi suosituimmista avoimen lähdekoodin pelimoottoreista ja sitä päivitetään aktiivisesti. Opinnäytetyön menetelmänä oli toiminallinen opinnäytetyö.

Ensiksi käsiteltävässä teoriaosuudessa käytiin läpi perusteita pelinkehityksestä, 3D-malleista ja pelimoottoreista. Lisäksi siinä kerrottiin yleisesti Godotin historiasta ja sen ominaisuuksista. Käytännön osuudessa luotiin yksinkertainen 3D-peli ja käytiin läpi mitä Godotin toimintoja sen kehittämiseen käytettiin. Lopuksi raportissa tehtiin yhteenveto projektin tuloksista.

Työssä ilmeni, että Godot sopii 3D-pelin kehitysalustaksi. Pelin perustoimintojen kehitys onnistui ilman suuria ongelmia ja pelin ensimmäinen taso saatiin valmiiksi. Godot osoittautui olevan laadukas ja tehokas pelimoottori.

Avainsanat Godot**,** peliteknologia, pelimoottori, pelinkehitys

Sivut 28 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Author Mikko Luoto **Year** 2020

Subject Developing a 3D game with Godot Engine

Supervisors Lauri Salminen

ABSTRACT

The goal of this thesis was to become familiar with 3D game development in Godot Engine. Godot's strengths and weaknesses as a game engine were also examined. This project went through Godot's basic functionalities and features. The idea for this project was born from the desire to find a free open source game engine as an alternative to Unity. Godot seemed like a good alternative, because it is one of the more popular open source game engines and gets updated frequently.

The theory part goes through the basics of game development, 3D games and game engines. It also explains general information about Godot. Godot's history, license and features are explored. The practical part of this thesis explains how a simple 3D-game is developed in Godot and how the tools that were used to create it work. The report ends on a summary of the project.

The project revealed Godot to be a suitable platform for 3D game development. The first level of the game and all the game's basic functionalities were completed without any major problems. Godot proved to be a powerful game engine.

Keywords Godot, technology, game engine, game development

Pages 28 pages

SISÄLLYS

SANASTO

Kuvataajuus: Kertoo kuinka monta kuvaa ruudulle piirretään sekunnin aikana.

Käyttöliittymä: Ohjelmiston osa, jota käytetään tuotteen ohjaamiseen, kuten valikot ja painikkeet.

Näytönohjain: Tietokoneen komponentti, jota käytetään kuvan piirtämiseen näytölle.

Pelimaailma: Pelimoottorin luoma virtuaalinen ympäristö, jossa pelaaja voi olla vuoro-vaikutuksessa muiden peliobjektien kanssa.

Renderointi: Kuvan tuottaminen 2D- tai 3D-mallista tietokoneohjelmalla.

Suoritin: Tietokoneen osa, jota käytetään suorittamaan tietokoneohjelman sisältämiä käskyjä.

Tietokonegrafiikka: Tietokoneohjelmalla tuotettuja digitaalisia kuvia.

1 JOHDANTO

Tässä opinnäytetyössä käsitellään yleisesti 3D-pelin kehittämistä Godot Engine -pelimoottorilla ja sitä, kuinka hyvin se sopii siihen tarkoitukseen. Godot Engine on melko uusi pelimoottori, joten on viisasta tutustua siihen, kuinka hyvä se on muihin pelimoottoreihin verrattuna.

Projektin idea syntyi halusta löytää tehokas ilmainen avoimen lähdekoodin pelimoottori korvaamaan Unity. Etsiessäni lisää tietoa Godotista projektia varten selvisi, että Godot tunnetaan pelinkehittäjien keskuudessa tehokkaana 2D-pelimoottorina, mutta sen tehokkuutta 3D-pelien kehittämisessä ei ole testattu yhtä laajasti. Tavoitteena tässä opinnäytetyössä on kehittää yksinkertainen 3D-peli Godot Enginellä, tutustua samalla sen käyttöön ja selvittää sopiiko Godot 3D-pelien kehitysalustaksi.

Aluksi käydään läpi perusteita pelinkehityksestä, pelimoottoreista ja 3D-peleistä. Lisäksi sen jälkeen kerrotaan Godotin ominaisuuksista, lisensseistä, toiminnoista ja historiasta. Toiminnallisessa osassa kerrotaan siitä, kuinka Godotilla kehitetään yksinkertainen 3D-peli ja tutkitaan kuinka siihen tarvittavat työkalut toimivat. Kehitettävässä pelissä ohjataan hahmoa labyrintissa, väistellään esteitä ja kerätään kolikoita.

Opinnäytetyön keskeisimmät tutkimuskysymykset ovat:
- Sopiiko Godot Engine hyvin 3D-pelin kehitysalustaksi?
- Miten 3D-peli luodaan Godotilla?
- Minkälaisia ongelmia esiintyi pelinkehityksen aikana?
- Minkälaisiin projekteihin Godot Engine on suositeltava kehitysalusta?

2 PELINKEHITYS JA PELIMOOTTORI

Luku 2 käsittelee pelien kehittämistä ja pelimoottoreita. Aluksi tutustutaan yleisesti pelinkehitykseen ja sen vaiheisiin. Lopuksi tutustutaan pelimoottoreiden rakenteeseen ja ominaisuuksiin.

Pelinkehitysprosessi tarkoittaa videopelin rakentamista konseptista julkaistavaan muotoon. Kuten mikä tahansa tuotantolinja, pelinkehitysprosessi auttaa järjestämään työnkulun siten, että kaikki tietävät mitä pitää tehdä ja milloin sen pitää olla valmista. Se auttaa hallitsemaan kehittämisaikataulua ja budjettia vähentämällä tehottomuuksia ja pullonkauloja. Vaikka työskentelytavat voivat olla erilaisia projektien ja studioiden välillä, itse prosessi on melko samanlainen niillä kaikilla. Peli muuttuu koko kehityksen ajan ja ideat, jotka kuulostivat hyvältä teoriassa, eivät välttämättä toimi niin hyvin kuin odotettiin. Näin ollen, kehitystyö ei ole aina lineaarinen prosessi. Samaa työtä voi joutua tekemään uudelleen, jos se ei läpäise kaikkia vaatimuksia. Kehitysprosessin on oltava riittävän joustava tällaisten muutoksien varalta. (Stefyn, 2019)

2.1 Pelinkehitysprosessi

Jotta pelin kehittäminen olisi sujuvaa, kehitysprosessi jaetaan yleensä kolmeen eri vaiheeseen. Esituotannossa kehitetään pelin ideaa. Tästä kaikki projektit alkavat. Pohjimmiltaan esituotanto määrittelee pelin aiheen, miksi se pitäisi tehdä ja mitä sen tekeminen vaatii. Peli voi syntyä ideasta tietyn tyyppiselle pelille tai ehkä tarinasta, joka halutaan herättää henkiin. Ehkä halutaan rakentaa sellainen peli, joka hyödyntää tietyn tyyppistä teknologiaa, kuten uutta ohjainta tai virtuaalitodellisuutta. Esituotannossa etsitään vastauksia kysymyksiin, kuten mistä pelissä on kyse, kuka on sen yleisö, onko sille kysyntää, mille alustalle se kehitetään, miten se kaupallistetaan, kuinka kauan kehittäminen vie aikaa, mikä on arvioitu budjetti ja mitä resursseja sen kehittäminen vaatii. Tämä vaihe voi kestää viikosta vuoteen riippuen projektityypistä, käytettävissä olevista resursseista ja rahoituksesta. Tässä vaiheessa työryhmä on melko pieni. Ryhmässä voi esimerkiksi olla tuottaja, ohjelmoija ja konseptitaiteilija. (Stefyn, 2019)

Videopeli on monipuolinen ohjelmisto, joka koostuu monesta eri osasta, kuten grafiikka, pelattavuus, koodi ja ääni. Jos peliä kehitetään ryhmässä, sen jäsenillä on usein eri aiheisiin keskittyvät roolit. Videopelien tuottaja voi hoitaa projektin liiketoimintapuolen, ohjata työryhmää ja toimia ryhmän edustajina. Tuottajan työkuva riippuu siitä, työskenteleekö hän pelin kehittäjille vai pelin julkaisijalle. Konseptitaiteilija asettaa projektille tunnelman jo varhain kehittämällä taidetta ja luonnoksia. Nämä varhaiset kuvitukset auttavat muodostamaan jokaiselle projektin parissa työskentelevälle visuaalisen oppaan pelin yleisestä ilmeestä. (Stefyn, 2019)

Tässä vaiheessa esituotantoa kerätyt tiedot muodostavat perustan pelisuunnitteludokumentille (Game Design Document). Pelisuunnitteludokumentti on pohjimmiltaan elävä asiakirja, joka antaa yleisen ymmärryksen projektista. Se sisältää tietoa pelistä, kuten sen konseptista, tarinasta ja tärkeimmistä pelimekaniikoista. Elävänä asiakirjana sitä päivitetään ja parannetaan jatkuvasti koko tuotannon ajan. Monet kehittäjät, etenkin pienemmät sellaiset, haluavat käyttää ketterämpiä kehittämistekniikoita, jotka liittyvät vähemmän dokumentointiin ja enemmän vain asioiden rakentamiseen. Suuret yritykset, kuten Microsoft, Sony ja Ubisoft vaativat dokumentaatioita. Se on yksi syistä, miksi ne ovat saavuttaneet jatkuvaa menestystä. Hyvä suunnitelma pitää kehityksen järjestäytyneenä ja auttaa tunnistamaan mahdolliset riskit. (Stefyn, 2019)

Videopelien prototyyppi on testi, joka tarkistaa pelin toiminnallisuuden, käyttökokemuksen, pelattavuuden ja pelimekaniikat. Prototyyppien tekeminen tapahtuu esituotannossa, jotta voidaan testata, toimiiko pelin idea ja kannattaako sen kehittämistä jatkaa. Monet ideat eivät pääse tästä vaiheesta ohi. Tavoitteena on saada nopeasti prototyyppi, jolla voi testata toimivatko ideat todella ja onko peli yhtä hauska kuin toivottu. Prototyyppi voi myös paljastaa odottamattomia haasteita, jotka voivat muuttaa koko projektin kulkua. On tärkeää saada muutkin testaamaan prototyyppiä, koska asiat, jotka ovat itsellesi selviä, eivät välttämättä ole muille. (Pickell, 2019)

Esituotannon jälkeen siirrytään päävaiheeseen eli tuotantoon, jossa itse peliä aletaan kehittämään. Tuotanto on kehitysprosessin pisin vaihe. Se kestää yleensä noin 1-4 vuotta ja on vaihe missä peli alkaa todella rakentumaan. Tarinaa hiotaan, hahmot ja ympäristöt luodaan, pelisäännöt päätetään ja koodi kirjoitetaan. Lähes kaikki sisältö videopelissä on tietoinen päätös. Tämä sisältää jokaisen hahmon, ympäristön, värin, äänen, vaikeustason ja säännön. Alkuperäiset ideat eivät kuitenkaan aina toimi niin hyvin todellisuudessa, joten kehityksen ohessa peliä testataan ja parannetaan jatkuvasti. (Stefyn, 2019)

Pienten työryhmien työntekijöillä on oltava useita tehtäviä, kun taas suuressa pelistudiossa on enemmän henkilökuntaa, joista monet erikoistuvat tiettyyn tuotannon osaan. Suunnittelijan tehtäviin sisältyy mielenkiintoisten tavoitteiden, sääntöjen ja haasteiden luominen. Suunnittelijat ovat vastuussa pelin hauskuudesta ja sen rakenteesta. Kenttäsuunnittelija vastaa mielenkiintoisten ja hauskojen tasojen luomisesta. Graafikko on vastuussa pelin visuaalisesta ilmeestä. He ovat taiteilijoita, jotka voivat esimerkiksi keskittyä konseptitaiteeseen, animointiin tai 3D-mallinnukseen. Ääniteknikot vastaavat äänitehosteista. Testaajat ovat välttämättömiä pelin kehitysprosessissa. He testaavat pelejä, etsivät vikoja ja varmistavat pelin toimivan oikein. (Stefyn, 2019)

Pelin tuotannossa on useita vaiheita. Jotkut pelit eivät koskaan pääse prototyyppivaiheen ohi. First playable on pelin ensimmäinen pelattava versio.

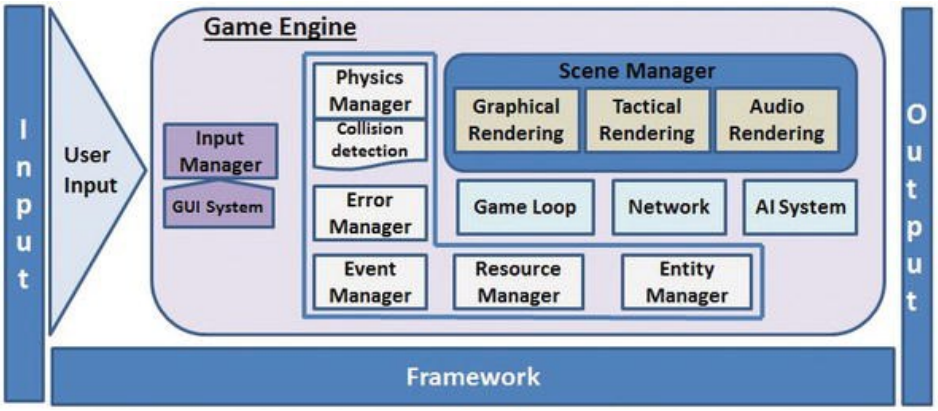
Se antaa paljon paremman kuvan pelin ulkoasusta ja pelattavuudesta kuin prototyyppi, sillä peliin aletaan lisätä valmiita pelielementtejä ja sisältöä. Alfassa kaikki pelin päätoiminnot on lisätty ja se on täysin pelattavissa alusta loppuun. Joitakin asioita tarvitsee ehkä vielä lisätä, mutta tärkeimpien toimintojen pitäisi toimia kunnolla. Betassa kaikki sisältö ja ominaisuudet on lisätty ja keskitytään virheiden korjaamiseen. Gold Masterissa peli on valmis ja se lähetetään julkaisijalle. Julkaisija on yritys, joka julkaisee videopelejä. (Stefyn, 2019)

Kun tuotanto on valmis ja peli on julkaistu, pelin kehitysprosessi jatkuu. Kehittäjät voivat keskittyä pelin ylläpitoon, kuten virheiden korjaamiseen tai lisäsisällön luomiseen. Tätä vaihetta kutsutaan jälkituotannoksi. Osa kehittäjistä voi siirtyä kehittämään jatko-osaa tai seuraavaa projektia. Koko kehitysprosessista voidaan tehdä selvitys, jossa käydään läpi mitä voitaisiin tehdä paremmin seuraavassa projektissa. Kaikki suunnitteludokumentit, sisältö ja koodi viimeistellään, kerätään ja varastoidaan. (Pickell, 2019)

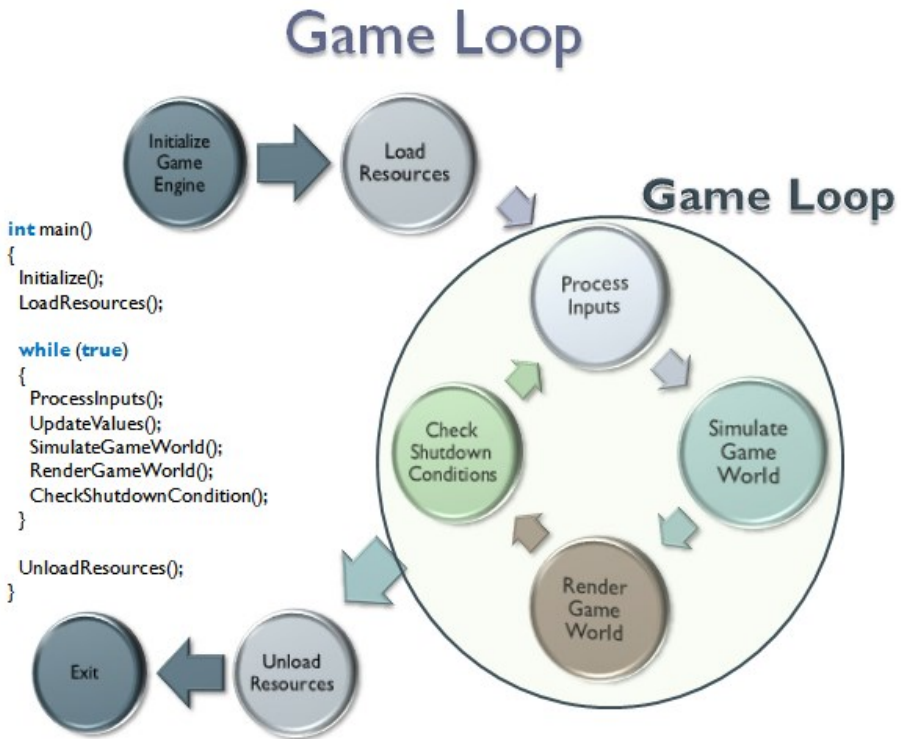
2.2 **Pelimoottorin perusteet**

Pelimoottori on kehitysympäristö videopelin luomista varten. Se mahdollistaa pelikehityksen ilman, että tekijältä vaaditaan erittäin vahvoja ohjelmointitaitoja ja aikaa tehdä jokainen toiminto itse. Pelimoottori myös helpottaa monelle eri alustalle kehittämistä. Pelin tekemiseen vaadittava työmäärä vähenee, koska kehittäjä voi hyödyntää pelimoottorin valmiita toiminnallisuuksia. Vaikka pelimoottori tarjoaa monenlaisia toiminnallisuuksia, jos pelissä halutaan käyttää monimutkaisempia toimintoja, ne täytyy ohjelmoida itse. Pelimoottorin kanssa käytetäänkin yleensä erillistä koodieditoria, kuten Visual Studiota. (Weimann, 2018)

Pelimoottori on ohjelmistokehys, joka sisältää pelin kehitykseen tarvittavia työkaluja. Se on vastuussa esim. grafiikan renderoinnista, tekoälystä, törmäyksen tunnistamisesta ja äänestä. Pelimoottorin pääohjelma vastaa pelin logiikasta ja renderointimoottori grafiikan tuottamisesta. Fysiikkamoottori vastaa fysiikan lakien toteuttamisesta ja sen avulla tunnistetaan peliobjektien törmäykset. Kuva 1 on esimerkki siitä minkälaisia prosesseja ja toimintoja pelimoottori voi sisältää. Kuva 2 luonnehtii sitä mitä pelimoottori tekee pelin ollessa käynnissä. Aluksi se käynnistyy ja lataa kaikki vaaditut resurssit. Seuraavaksi pelimoottori prosessoi käyttäjän antamat komennot, simuloi pelimaailman sen perusteella ja renderoi sen. Pelimoottori toistaa tätä silmukkaa, kunnes se saa käskyn lopettaa. (Baker, 2016)



Kuva 1. Esimerkki pelimoottorin rakenteesta. (Zarrad, 2018)



Kuva 2. Esimerkki pelisilmukasta eli siitä mitä pelimoottori tekee pelin ollessa käynnissä. (Karim, 2013)

Nykyään markkinoilla on monta eri pelimoottoria, joista jokaisella on omat toimintonsa ja ominaisuutensa. Lisäksi pelimoottoreiden hinnat ja lisenssityypit kannattaa ottaa huomioon valintaa tehdessä. Lisenssi määrittelee, minkälaiseen kaupalliseen käyttöön pelimoottoria saadaan käyttää. Pelimoottorin lisenssi voi olla joko avoimen lähdekoodin, ilmainen tai kaupallinen. Avoimen lähdekoodin moottorit ovat kaikkien käytettävissä ilmaiseksi ja niiden lähdekoodia voi muuttaa vapaasti. Ilmaiset pelimoottorit ovat jaossa ilmaiseksi internetissä, mutta niiden lähdekoodi ei ole välttämättä saatavilla. Kaupallisia pelimoottoreita myydään monilla eri

tavoilla. Niitä voidaan kaupata esimerkiksi kertaostoksena, tilauksena tai pelin tuotosta peritään tietty prosenttimäärä. (Salama & ElSayed, 2018)

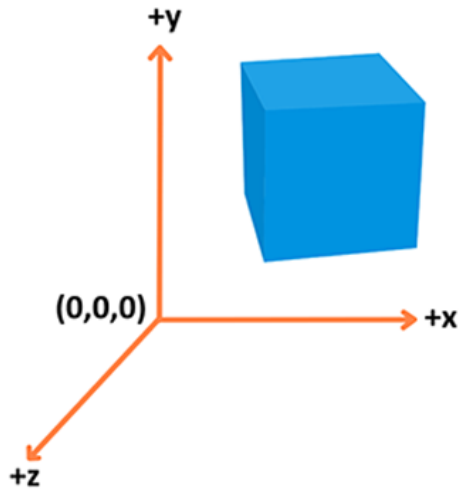
Nämä kaikki asiat voivat tehdä pelimoottorin valitsemisesta haastavaa. Pelimoottori kannattaakin valita sen perusteella, minkälaista peliä on itse kehittämässä ja mille alustalle haluaa julkaista sen. Pelimoottorien tukemat alustat, joille niiden pelejä voi julkaista vaihtelee erittäin paljon. Kannattaa ottaa huomioon myös pelimoottorin käytettävyys, kuten onko sen perusteet helppo oppia ja onko se tehokas käyttää. Suositulle pelimoottorille löytyy todennäköisesti helposti käyttöohjeita ja oppaita, koska käyttäjien määrä on suuri. (Zarrad, 2018)

3 3D-GRAFIIKKA PELEISSÄ

Tässä luvussa käsitellään 3D-grafiikkaa ja sen käyttöä peleissä. Aluksi tutustutaan 3D-mallien rakenteeseen ja niiden luomiseen. Lopuksi kerrotaan 3D-pelien historiasta. 3D-malleja käytetään asioiden visualisoimiseen esim. taiteessa, viihteessä, simuloinnissa ja suunnittelussa. 3D-mallit ja mallinnus ovat tärkeitä monella eri toimialalla, kuten videopeleissä, 3D-tulostuksessa, markkinoinnissa, elokuvissa ja tietokoneavusteisessa suunnittelussa. 3D-grafiikkaobjekteja kutsutaan 3D-malleiksi. 3D-mallin tiedot sisältyvät graafiseen tiedostoon. Malli voidaan näyttää kaksiulotteisena kuvana 3D-renderointiprosessin kautta tai sitä voidaan käyttää ei-graafisissa tietokonesimulaatioissa ja laskelmissa. (Slick, 2018)

3.1 3D-grafiikka

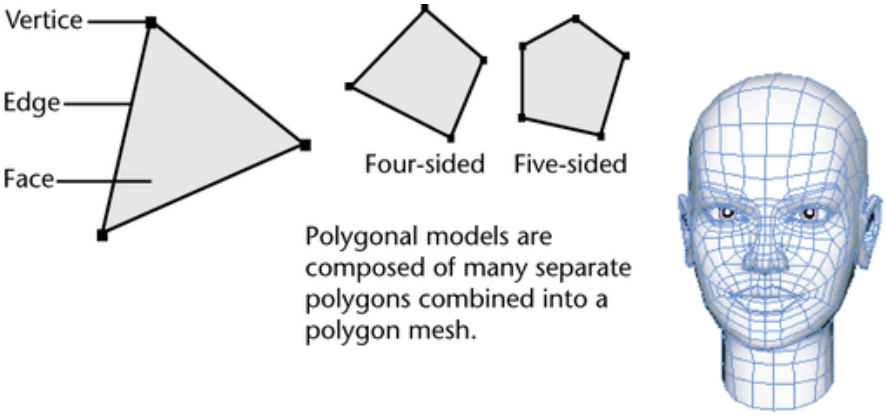
3D tarkoittaa kolmiulotteista eli asiaa, jolla on leveys, korkeus ja syvyys, kuten kuvassa 3 näkyy. Ihminen elää itsekin kolmiulotteisessa maailmassa ja pystyy tunnistamaan asioiden etäisyyden toisistaan syvyysnäön avulla. (MDN contributors, 2019)



Kuva 3. Asioiden sijainti kolmiulotteisessa tilassa määritellään koordinaattien avulla. Nämä koordinaatit ovat yleensä X, Y ja Z. (MDN contributors, 2019)

Videopeleissä käytettävän 3D-mallin pohjarakenne on polygonimalli (polygon mesh) eli verkko, joka koostuu polygoneista. Polygonit ovat geometrisia rakennuspalikoita, joiden avulla rakennetaan 3D-malleja. Polygonit ovat muotoja, jotka määritellään kolmiulotteisilla pisteillä (vertice) ja nämä pisteet yhdistetään suorilla viivoilla (edge). Näillä muodoilla on vähintään kolme sivua. Polygonin sisäistä aluetta kutsutaan termillä pinta (face). Vertice, edge ja face ovat polygonin olennaiset rakenneosat, kuten kuvassa 4 näkyy. Näiden rakenneosien avulla saadaan muokattua polygonia. Toisin

kuin 2D-grafiikassa, 3D-malleja pystyy katselemaan haluamastaan kuvakulmasta ja niiden rakennetta muuttamaan vapaasti. (Slick, 2020)



Kuva 4. Kuvan 3D-malli rakentuu polygoneista. (Autodesk, 2015)

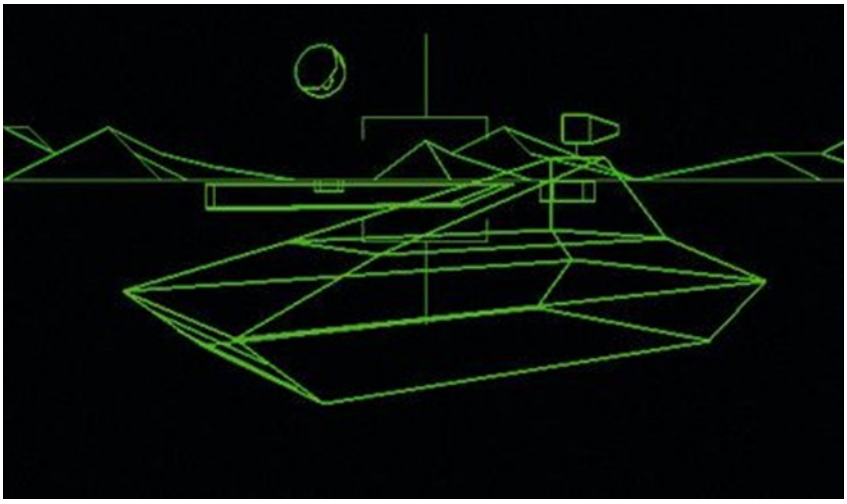
3D-mallin luomisprosessia kutsutaan 3D-mallinnukseksi. Tämä prosessi tehdään siihen tarkoitetussa 3D-mallinnusohjelmassa. Kaikista monimutkaisinkin 3D-malli on ollut alun perin yksinkertainen geometrinen muoto, kuten kuutio, pallo tai lieriö. Nämä yksinkertaiset geometriset muodot mallinnetaan ja muotoillaan mallinnusohjelmassa siksi mitä ikinä yritetäänkään luoda. Mallinnukseen käytettävien ohjelmistojen hallitseminen voi olla haastavaa ja niiden käyttö vaatii paljon opettelua. Markkinoilla on monia 3D-mallinnusohjelmia. Suosituimpia ovat AutoCAD, ZBrush, 3DS Max ja Blender. (Slick, 2018)

Tekstuureilla (texture) ja varjostimilla (shader) voi vaikuttaa 3D-mallin ulkonäköön. Varjostin on 3D-malliin sovellettava joukko ohjeita, joiden avulla tietokone tietää, miltä sen tulisi näyttää. Useimmissa 3D-ohjelmistopaketeissa on työkaluja, joiden avulla taiteilija voi säätää varjostimen parametrejä helposti. Näitä työkaluja käyttämällä taiteilija voi hallita tapaa, jolla mallin pinta on vuorovaikutuksessa valon kanssa, mukaan lukien opasiteetti, heijastuskyky ja kiiltävyys. Tekstuureilla voidaan vaikuttaa paljon polygonimallin ulkonäköön. Ne ovat kaksiulotteisella kuvia, jotka asetetaan 3D-mallin pinnalle. Tätä prosessia kutsutaan teksturoinniksi. Tekstuureja voidaan käyttää 3D-mallin pinnan yksityiskohtien ja näyttävyyden lisäämiseen. Tekstuurit voivat vaihdella yksinkertaisista tasaisesta väritekstuurista aina täysin fotorealistisiin pinnan yksityiskohtiin. Teksturointi ja varjostimet ovat tärkeä osa 3D-grafiikkaa. Texture- ja shader-taiteilijat ovat yhtä tärkeitä elokuvan tai pelin yleisessä ilmeessä kuin mallintajat tai animaattorit. (Slick, 2020)

3.2 **3D-pelien historia**

Ensimmäinen menestynyt 3D-peli oli vuonna 1980 julkaistu tankkipeli Battlezone, joka näkyy kuvassa 5. Pelissä pystyi liikkumaan vapaasti, suojautua

hyökkäyksiltä ja taistella vihollisia. Freescape oli varhainen 3D-pelimoottori. Se mahdollisti kolmiulotteisen ympäristön luomisen, joka koostui lattiasta ja niin monesta yksinkertaisesta muodosta kuin tietokoneen muisti ja suoritin sallii. Varhaisten 3D-pelien yleinen ongelma oli matala kuvataajuus, joka huononsi pelikokemusta. Suurin osa varhaisista 3D-peleistä olivat melko yksinkertaisia ja pelkästään kolmiulotteiselta näyttämistä pidettiin vaikuttavana. Suuri rajoittaja 3D-pelien grafiikalle on aina ollutkin sitä ajavan järjestelmän suoritusteho. Yksi temppu millä sitä vältettiin, oli antaa illuusio kolmiulotteisuudesta. Yksi esimerkki siitä on Wing Commander, jossa avaruudessa lentämisen illuusio luotiin säätämällä 2D-pikseligrafiikan kokoa. (PC Plus 2010)



Kuva 5. Battlezone on yksi varhaisimpia 3D-pelejä. (PC Plus, 2010)

Ampumapelit vaativat nopeutta, joten ne pidettiin yksinkertaisina. Vuonna 1992 julkaistu Wolfenstein 3D on peli, jossa kaikki tasot olivat täysin litteitä ja pelaajan interaktio pelin kanssa rajoitettu ampumiseen ja ovien avaamiseen. Samana vuonna julkaistu Ultima Underworld, sisälsi muun muassa kehittyneen valaistuksen, pelihahmoille puhumisen, pulmapelejä, fysiikkamoottorin, 3D-objekteja ja kyvyn katsoa ylös ja alas. Ultima Underworld pystyi sisältämään nämä kaikki ominaisuudet, koska se ei roolipelinä vaatinut samanlaista nopeutta ja tehokasta kuvataajuutta kuin ampumapeli. Kuvassa 6 näkyy minkälaista grafiikka Ultima Underword sisälsi. (PC Plus 2010)

Kuva 6. Ultima Underworldin laadukas 3D-grafiikka heikensi pelin suorituskykyä ja on syy pienelle peli-ikkunalle. (PC Plus, 2010)

Suurin osa peleistä eivät olleen täysin 3D-pelejä vaan 2.5D-pelejä, jotka hyödynsivät osittain 3D-grafiikkaa. Vasta vuonna 1996 julkaistu täysin 3D-grafiikkaa hyödyntävä ampumapeli Quake aloitti 2.5D-pelien suosion laskun. Pelien grafiikoiden parantuessa niiden laitteistolta vaatima suoritusteho myös kasvoi. Erillisen näytönohjaimen tarve alkoi ilmetä ja niiden suosia alkoi kasvamaan vähitellen. Laitteiston suoritustehon kasvaessa yleisesti, pelintekijät pystyivät alkamaan hyödyntämään sitä entistä laadukkaimmilla peligrafiikoilla. (PC Plus, 2010)

4 GODOT ENGINE

Tässä luvussa käsitellään Godot Enginen perustietoja ja ominaisuuksia. Godot Engine on alun perin Ariel Manzurin ja Juan Linietskyn luoma pelimoottori. Godot Engineä alettiin kehittää vuonna 2007 kaupallisena ja suljettuna pelimoottorina. Se julkaistiin myöhemmin uudelleen vuonna 2014 avoimen lähdekoodin MIT-lisenssillä. Godotin MIT-lisenssi tarkoittaa sitä, että sitä saa käyttää mihin tarkoitukseen ikinä haluaa. Se antaa luvan tutkia kuinka Godot toimii ja muuttaa sen toimintatapoja. Lisäksi se antaa luvan jakaa muuttamattomia ja muutettuja Godot Engine -versioita kaupallisesti ja eri lisenssillä. Godotilla tehdyn pelin tekijänoikeudet ovat täysin pelintekijän omassa omistuksessa. Ainoa ehto on, että ohjelman dokumentaation täytyy sisältää Godotin tekijänoikeusilmoitus ja lisenssiselvitys, jotka löytyvät Godotin verkkosivuilta. (Linietsky, Manzur & contributors, 2020)

Godotin ensimmäinen vakaa versio 1.0 julkaistiin joulukuussa 2014. Siitä lähtien Godot on saanut aktiivisesti päivityksiä. Versio 2.0 julkaistiin helmikuussa 2016 ja 3.0 tammikuussa 2018. Godotin kehitystä on tuettu rahapalkinnoilla. Godot sai vuonna 2016 Mozilla Open Source Support palkinnon, joka on arvoltaan 20 000 dollaria. Epic Games lahjoitti arvoltaan 250 000 dollarin palkinnon helmikuussa 2020. Microsoft rahoitti Mono ja C#-tuen lisäämistä 24 000 dollarin lahjoituksella. (Linietsky, 2020)

4.1 Ominaisuudet

Godot on tarkoitettu 2D- ja 3D-pelien luontiin. Godot pyrkii sisältämään kaikki työkalut mitä pelin tekemiseen voisi tarvita. Tuettuja alustoja ovat Linux, macOS, Windows, UWP, BSD, Android, iOS, HTML5 ja WebAssembly. Ennen 3.0 versiota ainoa tuettu skriptauskieli oli GDScript. GDScript on Godotin oma kieli, joka muistuttaa Pythonia. Godot sisältää oman debuggauksella varustetun koodieditorin. Nykyään tuettuja kieliä on neljä. Ne ovat GDScript, Visualscript, C++ ja C#. Pääkielet ovat GDScript ja Visualscript, koska ne ovat parhaiten integroitu Godot Engineen. C# ja C++ vaativat erillisen ohjelmointiympäristön. (Linietsky, Manzur & contributors, 2020)

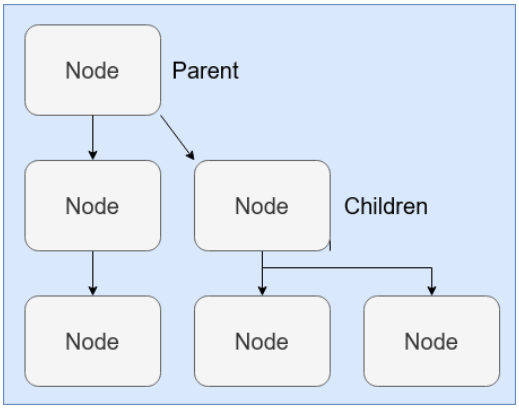
Grafiikan renderointiin Godot käyttää OpenGL ES 3.0 -rajapintaa. Myös OpenGL ES 2.0 käyttäminen on mahdollista. Godotin mukaan vanhemman 2.0 version käyttäminen on suositeltavaa, jos haluaa kehittää graafisesti kevyttä peliä, joka toimii suurimmalla osaa alustoista. Moottori tukee normaalikartoitusta, spekulaarisuutta, dynaamisia varjoja ja koko näytön jälkikäsittelytehosteita, kuten FXAA, bloom, DOF, HDR ja gammakorjaus. Mukana on myös yksinkertaistettu shader-kieli, joka muistuttaa GLSL-kieltä. Shadereita voidaan käyttää materiaaleissa ja jälkikäsittelyyn. Godot sisältää myös animointijärjestelmän. Fysiikkamoottorina toimii Bullet. (Linietsky, Manzur & contributors, 2020)

Godotista tarjotaan kahta eri versiota, standard ja mono. Mono eroaa perusversiosta niin, että se tukee C#-ohjelmointikieltä. Monoa kehitetään erikseen ja se on siitä syystä hieman standardia jäljessä päivityksissä. Godot koostuu yhdestä .EXE -tiedostosta ja se tekee siitä erittäin helposti siirrettävän ohjelman. Godot on tiedostokooltaan melko pieni ja Mono versio vain hieman standardia suurempi C#-tuen takia. (Linietsky, Manzur & contributors, 2020)

Godot-projektin tavoitteena on kehittää tehokas avoimen lähdekoodin pelimoottori, jolla voi kehittää pelejä ilman minkäänlaisia käyttörajoituksia. Lisäksi Godotissa panostetaan helppokäyttöisyyteen. Tämä käy ilmi esimerkiksi siitä, kuinka alun perin Godotissa ei ollut tapaa viedä projekteja. Kehittäjät käänsivät oikeat binaarit ja rakensivat paketit jokaiselle alustalle manuaalisesti. Kun kehittäjien määrä alkoi kasvaa ja yritys alkoi ottaa enemmän hankkeita samaan aikaan, kävi selväksi, että tämä oli pullonkaula kehitysprosessissa. Nykyään projektin viemisestä on tehty erittäin yksinkertaista. (Godot community, Linietsky & Manzur, 2020)

4.2 **Skenet ja solmut**

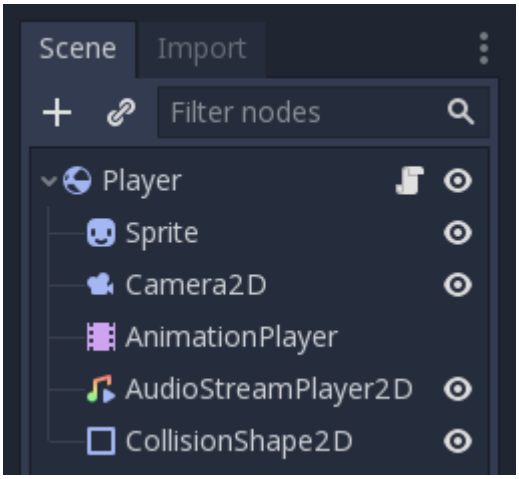
Godotissa solmu (node) on pelin luomisessa käytettävä rakennuspalikka. Solmulla pystyy suorittamaan monenlaisia erikoistuneita toimintoja. Jokaisella solmulla on aina nimi, sen ominaisuuksia pystyy muuttamaan ja se voi käsitellä vastakutsun jokaisella renderoidulla hetkellä. Lisäksi sitä voi laajentaa uusilla funktioilla ja sen voi lisätä toisen solmun alisolmuksi. Solmuilla on mahdollista olla muita solmuja alisolmuina. Kun tekee niin, solmuista muodostuu puu, kuten kuvassa 7 näkyy. Solmujen yhdistely on tehokas työkalu, koska erilaisia solmuja yhdistelemällä voi luoda monimutkaisia toimintoja. (Godot community, Linietsky & Manzur, 2020)



Kuva 7. Solmuista ja niiden alisolmuista muodostuu puu. (Godot community, Linietsky & Manzur, 2020)

Skene (scene) on ryhmä solmuja järjestettynä hierarkkisesti. Skenellä on aina yksi solmu, joka toimii juurena. Kuva 8 esittää pelaajahahmoa solmuna, jolla on sen kaikki muut ominaisuudet alisolmuina. Skene voidaan

tallentaa levylle tiedostona. Pelin ajaminen on sama asia kuin skenen ajaminen. Pelin ajaminen vaatii, että yksi skene on valittu pääskeneksi, joka käynnistyy ensimmäisenä. Pohjimmiltaan Godotin käyttö on skenen ja sen solmujen editointia. Moni tehokas tapa käyttää skriptejä, skenejä ja solmuja on soveltaa olio-ohjelmoinnin periaatteita niihin. Yksi olennaisimmista tavoista käyttää skriptejä Godotissa on niiden kiinnittäminen solmuihin. Kun skriptin kiinnittää solmuun, pystyy sen avulla määrittelemään sille monenlaisia ominaisuuksia, kuten mitä tapahtuu kyseisen solmun törmätessä muihin peliobjekteihin. Skriptin lisääminen solmuun onnistuu helposti painamalla solmua hiiren oikealla näppäimellä ja valitsemalla Attach Script-toiminnon. (Godot community, Linietsky & Manzur, 2020)



Kuva 8. Skene ja sen sisältämät solmut. Player-solmu toimii juurena sen alla oleville solmuille.

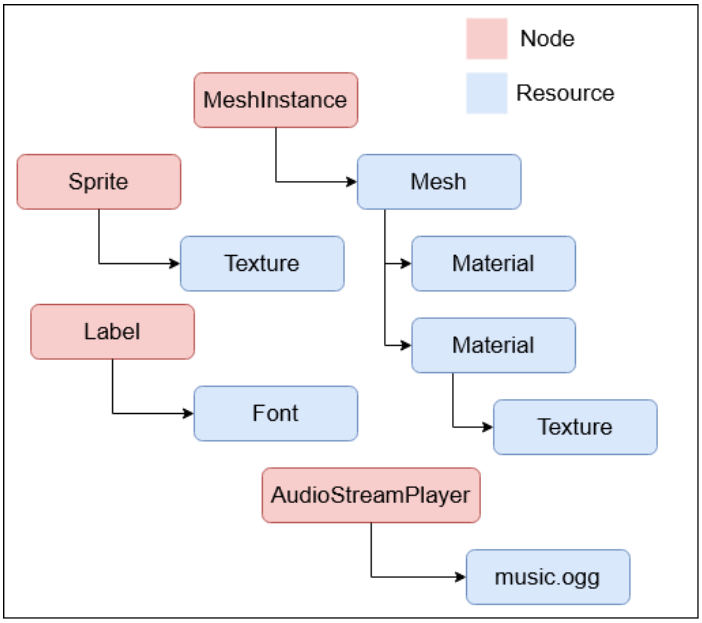
Pelin koon kasvaessa kehityksen aikana koko pelin rakentaminen yhden skenen sisään on huono idea. On suositeltavaa, että rakentaa pelin osia niiden omissa skeneissä. Jos solmun rakentaa sen omassa skenessä, sitä voi käyttää helposti uudelleen monessa eri skenessä. Tätä kutsutaan instansoimiseksi. Kaikki muutokset instanssin alkuperäiseen skeneen vaikuttavat sen instansseihin muissa skeneissä. (Godot community, Linietsky & Manzur, 2020)

4.3 **Resurssit**

Solmujen lisäksi on olemassa toinen tietotyyppi, joita kutsutaan resursseiksi (resource). Resurssit ovat yhtä tärkeitä Godotin käytössä kuin solmut. Solmut sisältävät kaikenlaisia toimintoja, kuten piirtävät 3D-grafiikan. Resurssit sisältävät dataa ja niiden tarkoitus on toimia välineenä datan säilömiseen. Resurssit eivät tee yksinään mitään, mutta sen sijaan solmut käyttävät resurssien sisältämää dataa. Kaikki mitä Godot tallentaa ja lataa levyltä on resurssi. Resurssi voi olla esimerkiksi skene, tekstuuri, skripti tai fontti. Kuva 9 on esimerkki siitä, kuinka solmut käyttävät resursseja. AudioStreamPlayer on solmu, jota käytetään äänten toistamiseen. Music.ogg

on äänitiedosto, jota sillä toistetaan eli kyseisen solmun käyttämä resurssi. (Godot community, Linietsky & Manzur, 2020)

Jokainen objekti, oli se sitten solmu tai resurssi, voi lähettää jonkun ominaisuuksistansa muualla käytettäväksi. Ominaisuuksia on monen tyyppisiä, kuten MeshInstance-solmun polygonimallin pinta voi sisältää tekstuurin, joka on oma resurssinsa. Tämä tarkoittaa sitä, että sekä solmut että resurssit voivat sisältää resursseja ominaisuuksina, kuten kuvassa 9 näkyy. (Godot community, Linietsky & Manzur, 2020)



Kuva 9. Solmut ja resurssit voivat sisältää resursseja ominaisuuksina. (Godot community, Linietsky & Manzur, 2020)

Resursseja voidaan tallentaa kahdella eri tapaa. Resurssi voi olla skenen ulkopuolella, eli tallennettu tietokoneen levylle omana tiedostonaan. Toinen vaihtoehto on tallentaa resurssi niin, että se on tallennettu skenen sisälle, johon se on kiinnitetty. Skene tallennetaan aina tscn- tai scn-tiedostoksi, jota se on silloin osa. Tässä projektissa käytetään molempia tapoja resurssien tallentamiseen. (Godot community, Linietsky & Manzur, 2020)

4.4 GDScript

GDScript on korkean tason, dynaamisesti tyypitetty ohjelmointikieli, jota käytetään Godot Enginessä skriptien eli komentosarjojen kirjoittamiseen. Se on syntaksiltaan erittäin lähellä Pythonia. GDScript on optimoitu ja integroitu käytettäväksi Godotissa. Alkuaikoina moottori käytti Lua-skriptikieltä. Lua on nopea, mutta sen yhdistäminen järjestelmään oli monimutkaista, hidasta ja vaati valtavan määrän koodia. Myös Python ilmeni kokeilujen jälkeen vaikeaksi integroida. Viimeinen kolmannen osapuolen

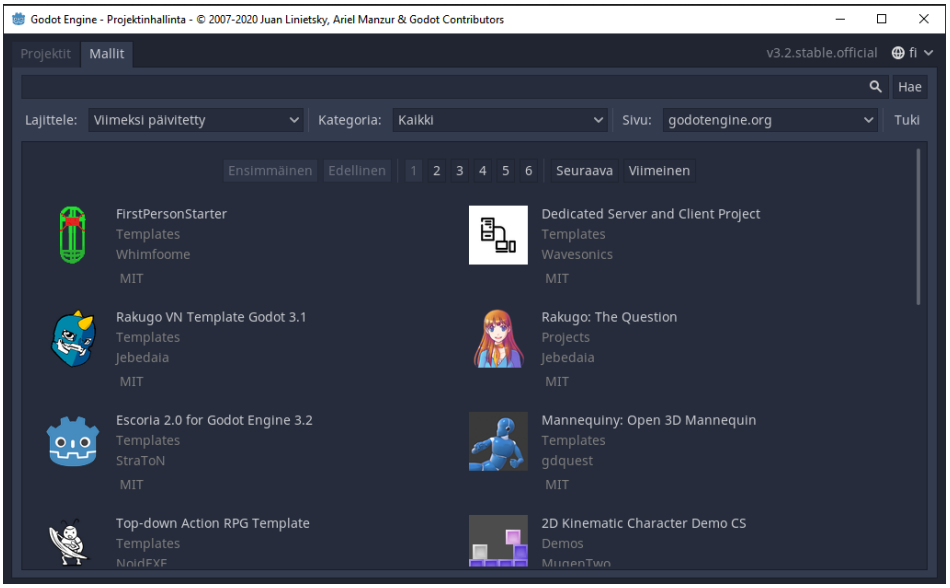
ohjelmointikieli, jota kokeiltiin, oli Squirrel, mutta myös se hylättiin. Siinä vaiheessa kävi ilmeiseksi, että pelkästään Godotille tehty oma skriptikieli voisi optimaalisesti hyödyntää sen arkkitehtuuria. Godot upottaa skriptit solmuihin ja suurinta osaa kielistä ei ole suunniteltu tällaista käyttöä ajatellen. (Godot community, Linietsky & Manzur, 2020)

Godot käyttää useita sisäänrakennettuja tietotyyppejä 2D- ja 3D-matematiikan laskemiseen. Skriptikielet eivät tue sellaista ja sen lisääminen oli kehittäjien mielestä epäkäytännöllistä. Godotilla on oma muistinhallintamalli resursseille, mutta suurin osa skriptikielistä tarjoaa sen päälle myös niiden oman sellaisen, mikä johtaa virheisiin. Koodin yhdistäminen on aina hankalaa ja johtaa yleensä moniin odottamattomiin ohjelmointivirheisiin ja huonoon ylläpidettävyyteen. Nämä kaikki syyt johtivat GDScript-kielen luomiseen. GDScriptin kieli ja tulkitsija olivat lopulta pienempiä kuin Lua- ja Squirrel-sidontakoodit, vaikka niillä oli yhtäläinen toiminnallisuus. Ajan myötä sisäänrakennetun kielen käyttö on osoittautunut valtavaksi eduksi. (Godot community, Linietsky & Manzur, 2020)

5 3D-PELIN KEHITTÄMINEN

Tavoitteena oli kehittää yksinkertainen 3D-peli Godotilla ja tutustua samalla sen käyttöön. Pelin ideana on liikuttaa hahmoa labyrintissa, kerätä kolikoita ja vältellä esteinä toimiviin vihollisiin koskemista. Ennen pelin kehittämisen aloittamista tutkin Godotin mukana tulevia valmiita peliprojektipohjia ja sitä, kuinka ne toimivat. Katsoin myös paljon opasvideoita pelinkehittämisestä Godotilla. Godotin sivuilta löytyvä dokumentaatio auttoi paljon sen käytön opettelussa ja oli tärkein tiedonlähde tässä projektissa. 2D-pelien kehittäminen Godotilla on paljon paremmin dokumentoitu ja suositumpaa kuin 3D-pelien, joten pidin 3D-pelin kehittämistä mielenkiintoisempana aiheena projektille.

Godotin käynnistäessä ensimmäisenä aukeaa projektinhallinta ikkuna. Siinä pystyy lisäämään, luomaan ja poistamaan peliprojekteja. Myös ohjelman kielen vaihtaminen on mahdollista. Kuvassa 10 näkyvältä mallit-välilehdeltä (templates) voi ladata avoimen lähdekoodin projektipohjia Asset Library -tietokannasta, jotta pääsee nopeammin alkuun. Niiden käyttämisestä on tehty erittäin helppoa. Tarvitsee vain valita haluttu projektipohja, napsauttaa lataa, asentaa se ja valita sille sijainti. Asset Library, joka tunnetaan myös nimellä AssetLib, on säilytyspaikka käyttäjien jakamille lisäosille, skripteille, työkaluille ja muille resursseille. Niiden lataaminen onnistuu pelimoottorin ja Godotin verkkosivujen kautta. Kaikki nämä resurssit ovat jaossa ilmaiseksi avoimen lähdekoodin lisenssillä.

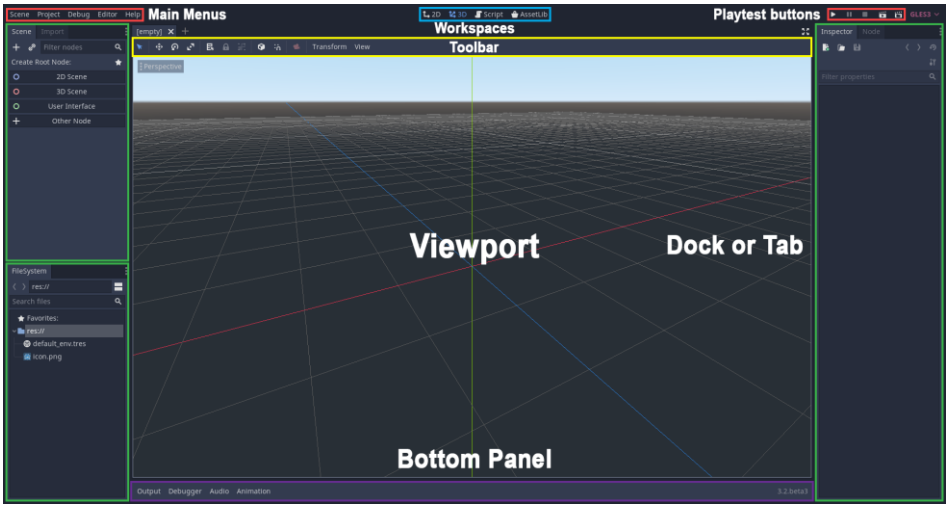


Kuva 10. Mallit-välilehti sisältää valmiita projektipohjia.

Uusi projekti luodaan napsauttamalla oikealla olevaa New Project -painiketta. Siitä avautuvassa ikkunassa projektille annetaan nimi, valitaan tyhjä kansio tallennussijainniksi ja päätetään mitä versiota OpenGL-ohjelmasta käytetään grafiikoiden renderointiin. OpenGL ES 3.0 tarjoaa

laadukkaimman grafiikan renderoinnin, mutta ei ole yhteensopiva kaikkien vanhempien alustojen kanssa. Sitä ei myöskään suositella käytettävän selainpeleissä. Näissä edellä mainituissa tapauksissa 2.0-version käyttäminen on suositeltavaa. Renderointimoottorin versiota voi muuttaa myöhemmin projektiasetuksista, jos sille on tarve. Tässä projektissa käytetään versiota 3.0, koska peliä kehitetään PC-alustalle ja halutaan saada kokonainen käsitys Godotin tehokkuudesta renderoida 3D-grafiikkaa.

Editorin käyttöliittymä koostuu valikoista (main menus), työtilasta (workspaces), pelitestauspainikkeista (playtest buttons), työkalupalkista (toolbar), näkymästä (viewport), välilehdistä (dock or tab) ja alapaneelista (bottom panel). Editorin käyttöliittymä näkyy kuvassa 11.

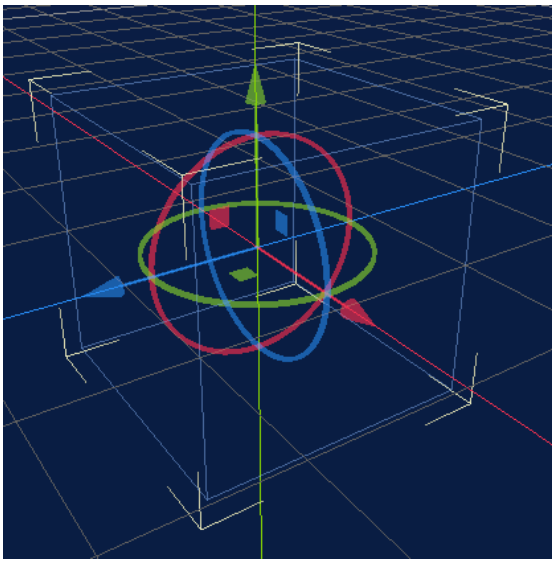


Kuva 11. Editorin käyttöliittymä jaettuna alueisiin. (Godot community, Linietsky & Manzur, 2020)

FileSystem-telakkaa käytetään projektin tiedostojen hallintaan. Sen yläpuolella on skene-telakka, joka sisältää aktiivisena olevan skenen sisällön. Oikealla sijaitsevassa Inspector-välilehdessä pystyy muuttamaan skenen peliobjektien ominaisuuksia. Pelinäkymän yläpuolella on työkalupalkki, josta löytyy työkalut skenessä olevien objektien liikuttamiseen, koon muuttamiseen ja lukitsemiseen. Alapaneelista löytyy debuggauskonsoli, animaatioeditori ja audiomikseri. Työtila painikkeita on neljä: 2D, 3D, Script ja AssetLib. 2D tilassa kehitetään 2D-pelejä ja käyttöliittymät. 3D-työtilassa kehitetään 3D-pelejä ja pääsee työskentelemään 3D-mallien, valojen ja kenttäsuunnittelun kanssa. 3D-työtilassa perspective-painiketta voidaan käyttää pelin näyttötavan muuttamiseen editorissa. Script-painike avaa koodieditorin. AssetLib on kirjasto ilmaisia lisäosia, skriptejä ja tiedostoja.

Godotin 3D-tilassa työskennellessä peliobjektin sijainti pelimaailmassa määritellään koordinaattien avulla. Y-koordinaatti on peliobjektin korkeus, X-koordinaatti leveys ja Z-koordinaatti syvyys. Peliobjektin asetuksissa sijaitseva Transform-välilehti sisältää nämä koordinaatit. Peliobjektien

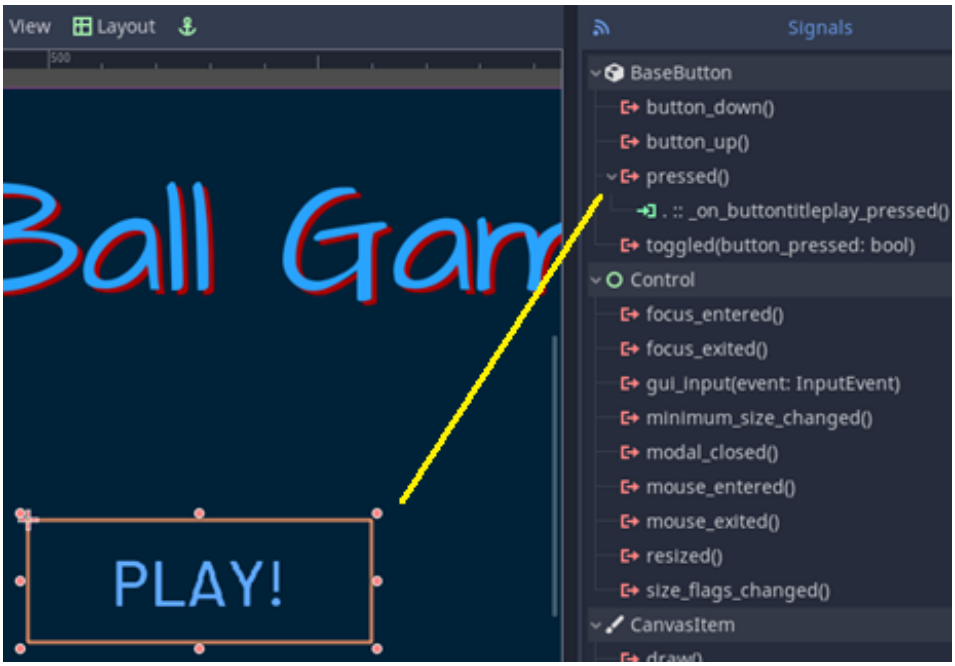
siirtäminen 3D-näkymässä tapahtuu liikuttamistyökaluilla (gizmo). Näissä työkaluissa jokaista akselia edustaa väri, kuten kuvassa 12 näkyy. Peliobjektien liikuttaminen onnistuu tietokoneen hiirellä raahaamalla, mutta tarkkuutta vaativissa siirroissa on suositeltavaa peliobjektin Transform-koordinaattien numeroarvojen muuttaminen.



Kuva 12. Punainen väri edustaa X-akselia, vihreä väri Y-akselia ja sininen väri Z-akselia.

5.1 Käyttöliittymän luominen

Tietty skene saadaan avautumaan ensimmäisenä pelin käynnistyessä, kun se valitaan pelin pääskeneksi. Tässä projektissa pääskeneksi valittiin pelin päävalikko. Päävalikko luodaan 2D-näkymässä. Käyttöliittymän luontiin käytettävien solmujen juurena toimii Control-solmu. Control-solmuun lisätään muita alisolmuja sen mukaan mitä pelissä tarvitaan. UI-elementit renderoidaan ylhäältä alas, eli skenessä olevian solmujen luettelossa alin UI-elementti on päällimmäisin itse pelissä. Käytin tekstikenttänä toimivaa label-solmua pelin nimen kirjoittamiseen ruudulle ja button-solmua pelin käynnistävän napin luomiseen. Yhdistin nappiin skriptin, joka avaa pelin ensimmäisen tason skenen. Painamalla nappia lähetetään signaali, joka kutsuu napin skriptissä aktiivisen olevan skenen vaihtoon käytettävää komentoa, kuten kuvassa 13 näkyy. Signaali on toiminto, jonka voi lisätä solmulle. Sen avulla solmu voi lähettää viestejä muille solmuille ja ne voivat vastata siihen halutulla tavalla. On käytännöllisempää lähettää signaali vain silloin kun nappia on painettu, kuin tarkistaa jatkuvasti napin tila. Päävalikon värien ja fonttien muuttaminen onnistuu helposti valitun käyttöliittymäsolmun asetuksista. Pelin häviämisruutu toteutettiin melkein täysin samalla tavalla kuin aloitusruutu. Häviö-skene ladataan pelaajan osuessa viholliseen. Häviämisruutu on samanlainen kuin päävalikko, vain tekstikenttien sisällöt ovat muutettu.



Kuva 13. Painamalla play-nappia lähetetään signaali, joka kutsuu napin skriptiä.

Peli sisältää myös pistelaskurin. Kerättyjen pisteiden määrä näkyy pelatessa ruudun oikeassa yläkulmassa. Pistelaskurin juurena toimii käyttöliittymäsolmujen yläluokka Control-solmu, joka on lisätty pelattavana olevaan skeneen. Käyttöliittymään käytettävät solmut ovat 2D-objekteja, jotka asettuvat ruudulla 3D-pelinäkymän päälle. Pisteet kirjoitetaan label-solmun tekstikenttään, jonka sisältö päivitetään aina skriptin kautta, kun pelaaja kerää kolikon. Pistelaskuri sisältää myös pienen kuvan kolikosta, joka lisättiin Sprite-solmun avulla. Se on solmu, jota voidaan käyttää yleisesti kaikenlaisten 2D-kuvien näyttämiseen.

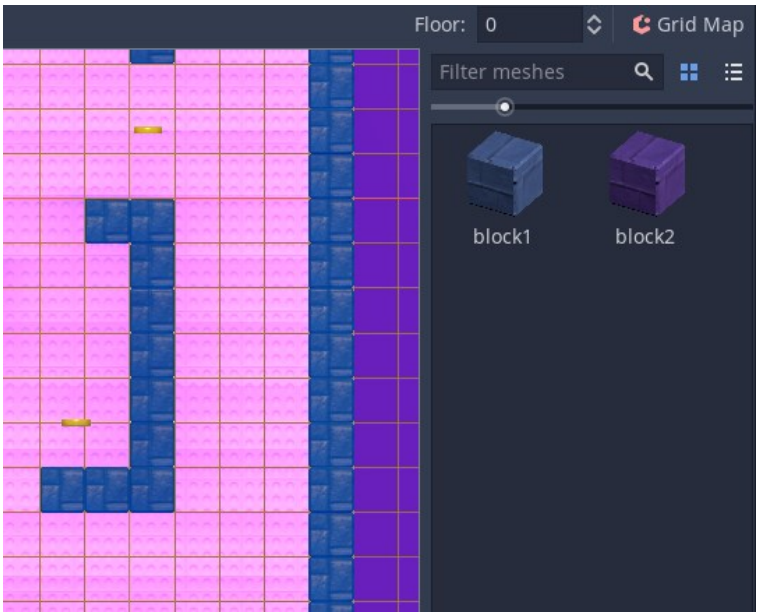
5.2 Ensimmäisen tason luominen

3D-pelejä luodessa kaikki siihen tarvittavat työkalut löytyvät Spatial-solmun aliluokista. Spatial-solmu toimii juurena kaikille sen alaisille solmuille. Loin tason, jonka päällä pelaaja liikkuu StaticBody-solmusta. StaticBody-rakenne on tarkoitettu peliobjekteille, jotka eivät liiku. Tällainen objekti ei kuluta paljon CPU-resursseja. StaticBody on yksi PhysicsBody-solmun aliluokista. Kaikki PhysicsBodyn alaluokat ovat erilaisia pohjarakenteita peliobjektille. Niillä on omat ominaisuutensa ja käyttötarkoituksensa. Staticbody-solmu tarvitsee CollisionShape-solmun, että se voi törmätä muihin peliobjekteihin. Peliobjekti menee läpi muista objekteista, kuten lattiasta, jos sille ei anna solmua, joka tunnistaa törmäykset. CollisionShape-solmulle pitää antaa muoto. Tässä projektissa pelin maan haluttiin olevan muodoltaan yksi suuri neliö, joten CollisionShape sai kuution muodon. StaticBody tarvitsee myös polygonimallin, että sen näkee pelissä.

Polygonimallin saa lisättyä MeshInstance-solmulla. MeshInstance voi käyttää Godotin luomaa yksinkertaista 3D-mallia, kuten palloa tai kuutiota, mutta myös 3D-mallin tuominen mallinnusohjelmasta on mahdollista. Polygonimallin tuonti projektiin esimerkiksi Blenderistä onnistuu siirtämällä sen gltf-tiedosto pelin projektikansioon. Polygonimallin pintaa pystyy muuttamaan vielä lisää sen materiaaliasetuksista (material), kuten antamalla sille epätasaisen pinnan. Kuvassa 14 näkyvä materiaalivalikko sisältää monenlaisia asetuksia, joita voi käyttää polygonimallin pinnan muuttamiseen, kuten teksturointiin.

Kuva 14. Materiaaliasetukset sisältävät monipuolisia toimintoja polygonimallin pinnan muokkaamiseen.

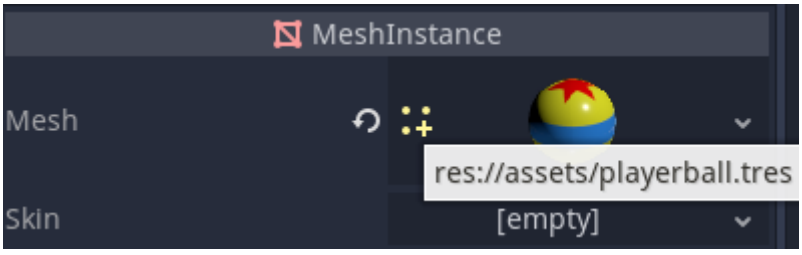
Gridmap on työkalu 3D-pelimaailmojen luomiseen, jolla voi asetella valmiiksi määriteltyjä peliobjekteja suoraan pelimaailmaan kuin rakennuspalikoita. GridMap käyttää MeshLibrary-toimintoa, joka on luettelo näistä valmiiksi käytettävistä polygonimalleista. Meshlibrary auttaa pelin rakentamisessa, koska samoja rakennuspalikoita voi käyttää helposti uudelleen. Tässä projektissa Gridmapia käytettiin esimerkiksi pelin seinien asetteluun. Editorin perspektiivin asettaminen Perspective-asetuksista Orthogonal-tilaan on suositeltavaa, sillä se helpottaa peliobjektien asettelussa huomattavasti. Orthogonal-tilassa editori tasoittaa pelimaailman, kuten kuvassa 15 näkyy.



Kuva 15. Gridmap nopeuttaa pelimaailman luomista huomattavasti.

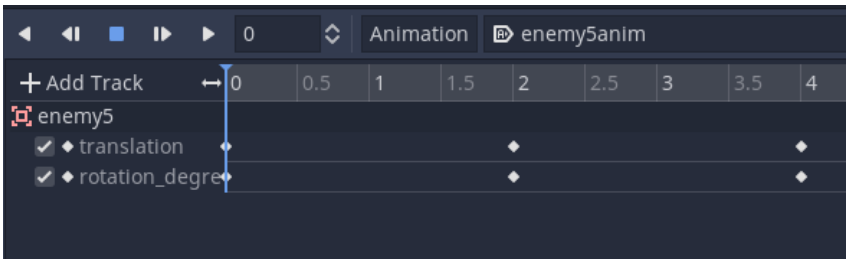
5.3 Pelaajahahmon ja vihollisen luominen

Pelissä ohjattava hahmo on värikäs pallo. Loin pelaajahahmon sen omassa skenessä, josta se on helppo siirtää muihin pelin skeneihin Instance-nappia painamalla. Peliobjektia on helppo käyttää uudelleen, jos sen luo omana skenenään. Skene tallentuu tscn-tiedostoksi. Pelaajahahmon rakenne kannattaa luoda KinematicBody-solmusta. KinematicBody on yksi PhysicsBody-solmun alaluokista. Se on tarkoitettu pelaajan ohjaamille objekteille. Lisäsin pelaajahahmon KinematicBody-solmulle alisolmuksi neliönmuotoisen CollisionShape-solmun, että pelaajan törmäykset muihin peliobjekteihin voidaan tunnistaa. Käytin myös MeshInstance-solmua pelaajahahmon polygonimallin lisäämiseen. Minulla oli Blenderissä tehty gltf-polygonimalli, jonka avasin Godotissa ja tallensin sen tres-tiedostoksi. Tres-tiedostoksi muutettuna polygonimallia voi käyttää MeshInstance-solmussa, kuten kuvassa 16 näkyy. Lisäsin pelaajahahmon solmulle myös oman skriptin, jossa on sen liikuttamiseen tarvittava koodi.



Kuva 16. Polygonimalli lisättynä MeshInstance-solmuun.

Pelin vihollisten solmut muistuttavat monella tapaa pelaajan ohjaaman hahmon solmua, mutta niiden käyttötapa pelimaailmassa poikkeaa paljon. Vihollisten pohjarakenteena toimii Area-solmu. Area-solmu on yleiskäyttöinen alue, joka on tarkoitettu 3D-peliobjektien tunnistamiseen ja niihin vaikuttamiseen. Sen alisolmuksi lisättiin myös CollisionShape-solmu tunnistamaan törmäykset pelaajaan ja MeshInstance-solmu näyttämään vihollisen 3D-malli, joka tuotiin samalla tavalla Blenderistä kuin pelaajahahmon. Vihollisten tarkoitus pelissä on toimia esteinä, joihin pelaaja ei saa koskea ja samalla liikkua samalla ennalta määrättyjä reittejä pelimaailmassa. Vihollisen liikuttamiseen käytettiin AnimationPlayer-solmua. AnimationPlayer-Solmulla pystyy toistamaan animaatioita. Tässä projektissa se lisättiin vihollisen alisolmuksi toistamaan animaatiota, joka muuttaa vihollisen sijaintia pelimaailmassa. Tämä tehtiin muuttamalla vihollisen koordinaatteja, kuten kuvassa 17 näkyy. Tämä animaatio pyörii silmukkana, joten viholliset toistavat samaa liikerataa ikuisesti.



Kuva 17. Translation on objektin sijaintikoordinaatti pelimaailmassa. Jokainen ruutukuvio on uusi sijaintikoordinaatti johon AnimationPlayer siirtää peliobjektia animaation edetessä.

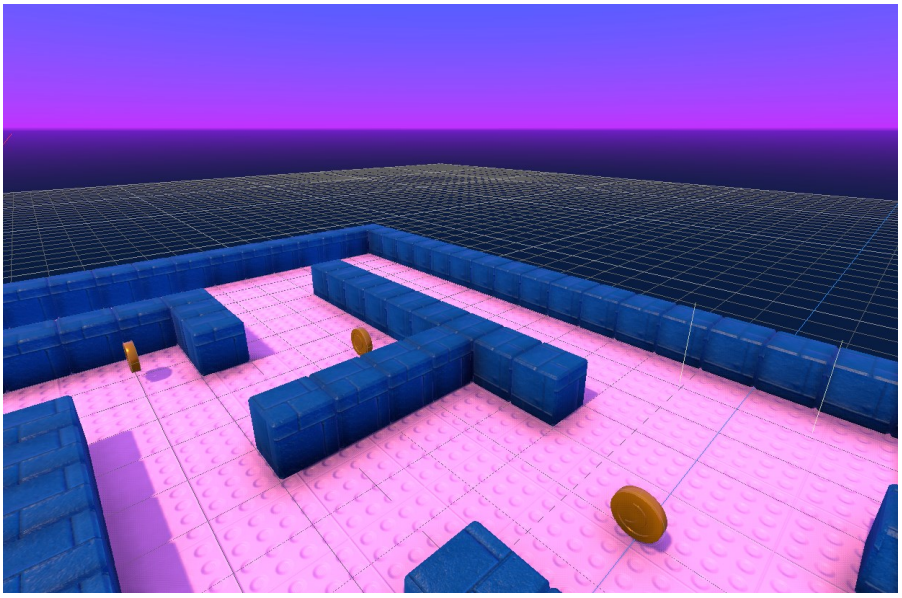
Ainoa tapa hävitä pelissä on koskea viholliseen. Pelaajan koskettaessa viholliseen lähetetään signaali sen skriptiin, joka avaa häviöruutu-skenen. Pelimaailmaan on aseteltu kolikkoja, joita keräämällä tietyn määrän pelaaja voittaa pelin. Kolikon rakenteena toimii Area-solmu, kuten vihollisilla. Kolikoilla on yhteinen skripti, joka päivittää ruudun yläkulmassa olevan pistelaskurin, kun pelimoottori tunnistaa kolikon törmäyksen pelaajahahmoon. Tämä tehtiin CollisionShape-solmulla, jonka avulla törmäykset pelaajahahmoon tunnistetaan. Kolikolla on MeshInstance-solmulla lisätty polygonimalli, joka näyttää tyypilliseltä kultakolikolta. Kolikon polygonimalli tuotiin Blenderistä samalla tavalla kuin vihollisen ja pelaajahahmon. Kolikolla on myös AnimationPlayer-solmu, joka käynnistää pienen animaation, aina kun pelaaja kerää kolikon. Tämän avulla pelaaja näkee, että kolikko tuli todellakin kerättyä. Kolikkojen yhteinen skripti tarkistaa kerättyjen kolikoiden määrän aina kun sellainen kerätään ja se myös avaa voittoruutu-skenen, kun oikea määrä kolikkoja on kerätty.

5.4 Kamera, valaistus ja ympäristö

Pelinäkymä muodostuu kamerasta, joka seuraa pelaajahahmoa. 3D-peleissä käytettävät kamerat löytyvät Spatial-solmun alta. Camera on solmu,

jonka kautta pelaaja näkee pelin. Jos Camera-solmun lisää pelaajahahmon alisolmuksi se lukittautuu siihen ja seuraa sitä pelaajan liikkuessa. Se ei ollut tähän projektiin sopivin vaihtoehto, sillä kameran liike ei ollut kovin sujuvaa ja pehmeää. Oli parempi käyttää InterpolatedCamera-solmua. InterpolatedCamera on kamera, joka liikkuu kohti sille asetettua kohdetta. Sujuvampi kameran liike saadaan lisäämällä tyhjä Spatial-solmu pelaajahahmon alisolmuksi ja asettamalla se lyhyen etäisyyden päähän pelaajahahmosta. Tämä pelaajan taakse asetettu solmu lisätään InterpolatedCameran kohteeksi, jota kohti se sitten liikkuu. Näin kameran liikkeeseen saadaan pieni viive, joka tekee sen liikkeestä pehmeämpää.

Seuraavaksi peliin tarvittiin lisätä valonlähde. Tähän peliin sopi parhaiten valonlähde, joka valaisee koko pelialueen. Tällaiseen tarkoitukseen sopii DirectionalLight-solmu. Se valaisee pelialueen kaukaa yhdestä suunnasta auringon tapaan, kuten kuvassa 18 näkyy. Tästä syystä itse solmun sijainnilla pelimaailmassa ei ole vaikutusta valaistukseen, vaan valaistuksen suuntaa muutetaan solmun asetuksista.



Kuva 18. DirectionalLight-solmu valaisee koko skenen.

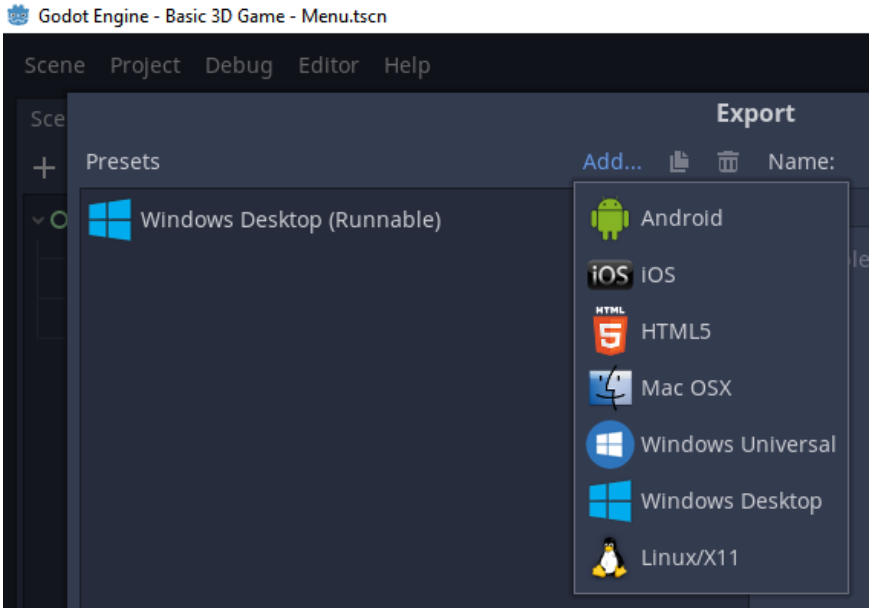
Ympäristö (environment) on resurssi, jonka voi lisätä kameraan. Se sisältää asetuksia, joita voi käyttää peliympäristön renderoinnin muuttamiseen. Tässä projektissa sitä käytettiin taivaan ja ympäristövalaistuksen muokkaamiseen, mutta se sisältää myös monia muita asetuksia, kuten kuva 19 näkyy. Projektin skeneille voi asettaa oletusympäristön projektin asetuksista, jos haluaa tiettyä ympäristöä käytettävän kaikissa skeneissä.

Kuva 19. Environment-resurssi sisältää useita asetuksia ympäristön muuttamiseen.

5.5 **Pelin julkaisu**

Kun pelin saa valmiiksi ja toimivaksi, seuraava vaihe on sen jakaminen muille. Ei ole käytännöllistä pyytää muita lataamaan Godot vain sen takia, että he voivat avata peliprojektin. Sen sijaan projektin voi viedä (export) eli muuntaa se muotoon, jonka kuka tahansa voi ajaa. Pelin vientitapa riippuu siitä mille alustalle peli on kohdistettu. Tämän projektin peli julkaistiin PC-alustalle. PC-alustalle pelin viemisestä on tehty Godotissa erittäin helppoa. Se vaatisi pieniä muutoksia, jos peli tehtäisiin puhelinta tai tablettia varten.

Projektin vieminen helpommin jaettavaan muotoon tehdään Export-valikosta, joka löytyy editorin ylärivin projektivälilehdeltä. Export-valikon avautuessa ensimmäistä kertaa se on tyhjä. Tämä johtuu siitä, että Godotiin on lisättävä viennin esiasetus. Viennin esiasetus luodaan napsauttamalla vientivalikon yläosassa olevaa Add-painiketta. Tämä avaa luettelon alustoista, joista valitaan se, jolle peli halutaan kehittää. Tämä lista näkyy kuvassa 20.



Kuva 20. Lista alustoista, joille peli voidaan julkaista. Kuvassa näkyy valmis viemisen esiasetus Windowsille.

Oletusasetukset olivat tarpeeksi hyvät vientiin, joten niiden muuttaminen ei ollut tarpeen. Monet alustat vaativat lisätyökalujen, kuten ohjelmistokehityspaketin (software development kit) asentamisen vientiä varten. Lisäksi Godotin tarvitsee asentaa vientipohja. Vientivalikko ilmoittaa, kun jotain tarvittavaa puuttuu, eikä anna käyttäjän viedä valitulle alustalle ennen kuin ongelma on ratkaistu. Tässä projektissa Godot ilmoitti vientipohjan puuttuvan. Sen pystyy asentamaan Godotin sisällä. Vientipohja on tpz-tiedosto, joka voidaan ladata ja asentaa käyttämällä editorin valikosta löytyvää Manage Export Templates -painiketta.

Vientiä suoritettaessa Godot tekee luettelon kaikista vietävistä tiedostoista ja luo sitten niistä paketin. Projektin vienti voidaan tehdä kolmella eri tavalla. Ensimmäinen tapa on viedä jokainen tiedosto. Tämä tapa vie kaikki projektin tiedostot. Sillä on hyvä testata, onko jotain unohdettu, mutta kehittäjillä on usein projektihakemistossa paljon turhia tiedostoja, mikä tekee siitä huonon vaihtoehdon. Toinen vaihtoehto on viedä vain käytetyt resurssit. Useimmissa projekteissa se riittää. Kolmas vaihtoehto on viedä vain valitut resurssit. Siinä vain listaan valitut tietyt resurssit viedään. Tämä on luultavasti ylilyönti useimmissa projekteissa, mutta joissakin tapauksissa se on viisasta, kuten suurissa projekteissa. Tämä tapa tarjoaa täyden kontrollin siitä mitä viedään. Yksittäisiä resursseja voidaan valita ja riippuvuuden havaitseminen suoritetaan sen varmistamiseksi, että kaikki tarvittava lisättiin. Tämän projektin vienti tehtiin käyttäen tapaa, joka vei vain tarvitut resurssit.

6 TULOKSET JA YHTEENVETO

Pelin kehitys onnistui ilman mitään suuria ongelmia. Projektissa Godot ilmeni toimivan kevyen 3D-pelin kehitysalustana hyvin. Kehitettyä peliä voisi jatkokehittää ja parantaa monella eri tapaa. Peliin voi esimerkiksi lisätä uusia tasoja. Projektin voidaan katsoa onnistuneen, sillä suurinta osaa Godotin perustoiminnoista tuli käytettyä.

Kehityksen aikana ilmeni pieniä ongelmia. Nämä ongelmat olivat yleensä epäselvyyksiä ja väärinymmärryksiä. Nämä ongelmat kuitenkin ratkesivat itsestään, nopeasti tai eivät haitanneet kehitystä erityisesti. Jotkut ohjelman toiminnot olivat piilossa monen valikon alla tai niiden toimintaperiaatetta ei ollut helppo ymmärtää. Tällaiset ongelmat vaativat ylimääräistä selvittämistä dokumentaatiosta. Monimutkaiseen ongelmaan ratkaisun löytäminen voi olla hankalaa, sillä verkosta löytyvien oppaiden määrä on melko pieni, joka johtuu käyttäjien pienestä määrästä. Pelimoottorin perusteet on dokumentoitu erittäin hyvin sen verkkosivuilla. Godotin dokumentaatiosta saa kattavan kuvan siitä, kuinka se toimii. Projektin aikana dokumentaatiota tuli hyödynnettyä erittäin paljon.

Godot sopii 3D-pelin kehitysalustaksi ainakin, jos tavoitteena on kehittää pieni ja yksinkertainen peli. Godot tarjoaa pelinkehitykseen vaadittavat perustoiminnot ja sen käytöllä ei ole minkäänlaisia rajoitteita. Jos tavoitteena olisi kehittää laajempaa ja kunnianhimoisempaa peliä, uskoisin pidemmälle kehitetyn pelimoottorin, kuten Unityn ja Unreal Enginen olevan parempi vaihtoehto. Ne ovat vielä tällä hetkellä paremmin dokumentoituja, suositumpia ja yhtä tehokkaita käyttää. Tällä hetkellä ei ole vielä julkaistu yhtäkään erittäin menestynyttä peliä, joka on kehitetty Godotilla. Yksi suuri syy tälle on se, että Godot on vielä melko uusi pelimoottori. Godot 3.0 on vasta noin pari vuotta vanha ja pelien kehittäminen voi viedä useita vuosia. On siis erittäin todennäköistä, että tulevaisuudessa julkaistaan Godotilla kehitettyjä pelejä, jotka menestyvät kaupallisesti.

Godotin tulevaisuus vaikuttaa hyvältä. Se on saanut paljon rahallista tukea ja sitä päivitetään aktiivisesti. Godotin käyttäjämäärän kasvaessa se tulee varmasti saamaan entistä enemmän oppaita, toimintoja ja ominaisuuksia. Godot on jo nyt yksi tehokkaimmista pelimoottoreista ja sen saaman aktiivisen tuen takia se tulee olemaan vielä tehokkaampi tulevaisuudessa.

LÄHTEET

Autodesk (2015). Introduction to polygons. Viitattu 9.2.2020 https://knowledge.autodesk.com/support/maya-lt/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/MayaLT/files/Polygons-overview-Introduction-to-polygons-htm.html

Baker, M. (2016). How Do Game Engines Work? Viitattu 2.2.2020 https://interestingengineering.com/how-game-engines-work

Dawe, L. (2014). Godot Game Engine Is Now Open Source. Viitattu 9.2.2020 https://www.gamingonlinux.com/articles/godot-game-engine-is-now-open-source.3096

Godot community, Linietsky, J. & Manzur, A. (2020). Godot Docs – 3.2 branch. Viitattu 9.2.2020 https://docs.godotengine.org/en/3.2/index.html

Karim, W. (2013). Inside the Game – Game vs. Software. Viitattu 1.3.2020 https://gamyguru.wordpress.com/2012/07/13/inside-the-game-game-vs-software/

Linietsky, J. (2020). Godot Engine was awarded an Epic MegaGrant. Viitattu 9.2.2020 https://godotengine.org/article/godot-engine-was-awarded-epic-mega-grant

Linietsky, J., Manzur, A. & contributors (2020). Godot Engine Features. Viitattu 9.2.2020 https://godotengine.org/features

MDN contributors (2019). Explaining basic 3D theory. Viitattu 5.2.2020 https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Basic_theory

PC Plus (2010). The evolution of 3D games. Viitattu 9.2.2020 https://www.techradar.com/news/gaming/the-evolution-of-3d-games-700995

Pickell, D. (2019). The 7 Stages of Game Development. Viitattu 1.2.2020 https://learn.g2.com/stages-of-game-development

Salama, R. & ElSayed, M. (2018). Basic elements and characteristics of game engine. Viitattu 2.2.2020 https://un-pub.eu/ojs/index.php/gjcs/article/download/4023/3943

Slick, J. (2018). What Is 3D Modeling? Viitattu 7.2.2020 https://www.lifewire.com/what-is-3d-modeling-2164

Slick, J. (2020). 3D Model Components—Vertices, Edges, Polygons & More. Viitattu 8.2.2020
https://www.lifewire.com/3d-model-components-1952

Stefyn, N. (2019). How Video Games Are Made: The Game Development Process. Viitattu 19.4.2020
https://www.cgspectrum.com/blog/game-development-process

Weimann, J. (2018). How to Get Started in Game Development. Viitattu 1.2.2020
https://simpleprogrammer.com/started-game-development/

Zarrad, A. (2018). Game Engine Solutions. Viitattu 2.2.2020 https://www.intechopen.com/books/simulation-and-gaming/game-engine-solutions