OULUN AMMATTIKORKEAKOULU

Joona Karhu, Sami Nousiainen & Carita Niskanen

PC-pelin suunnittelu ja toteutus Unity-pelimoottorilla

Whispering Woods

PC-pelin suunnittelu ja toteutus Unity-pelimoottorilla

Whispering Woods

Joona Karhu, Sami Nousiainen &
Carita Niskanen
Opinnäytetyö
Syksy 2024
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tradenomi (AMK), tietojenkäsittely

Tekijä(t): Joona Karhu, Sami Nousiainen & Carita Niskanen
Opinnäytetyön nimi: PC- pelin suunnittelu ja toteutus Unity- pelimoottorilla
Työn ohjaaja(t): Reima Riihimäki
Työn valmistumislukukausi ja -vuosi: Syyslukukausi 2024 Sivumäärä: 30

Teimme opinnäytetyönä uudelleen Oulun Game Labissa toteutetun mobiilipeli Whispering Woods-projektin PC-alustalle. Opinnäytetyö käsittelee ryhmämme suunnittelu- ja toteutusvaihetta, jossa käydään läpi perinteisten 2D Platformer-pelien historiaa, pelimoottorien vertailua ja käytettyjä kuvanmuokkausohjelmia. Peli toteutettiin Unityllä ja pelin grafiikat piirrettiin suurimmalta osin käsin.

Opinnäytetyössä keskitytään pelin toteutukseen ja kerrotaan käytettyjen ohjelmien toimintaa.
Pelin ohjelmointiin käytettiin C#-ohjelmointikieltä Unityn pelimoottorin kanssa ja pelitaide tehtiin opetellen useita eri kuvanmuokkausohjelmia. Peliin suunniteltiin ja toteutettiin yksi kenttä, yksi iso taistelu päävihollista vastaan ja menu.

Asiasanat: Peligrafiikka, peliohjelmointi, pelimekaniikat, Unity, ohjelmointi, pelimoottori

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Business Administration

Author(s): Joona Karhu, Sami Nousiainen & Carita Niskanen
Title of thesis: PC game design and implementation with Unity game engine
Supervisor(s): Reima Riihimäki
Term and year when the thesis was submitted: Autumn 2024
Number of pages: 30

In this thesis, we redesigned the mobile game Whispering Woods, a project implemented at Oulu Game Lab, for the PC platform. The thesis deals with the design and implementation period of our group, which goes through the history of traditional 2D platformer games, comparison of game engines and the use of image editing software. The game was implemented using Unity and most of the graphics were drawn by hand.

The thesis focuses on the implementation of the game and explains the software used.
The game was programmed using the C# programming language with the Unity game engine and the game art was mostly done with multiple different editing softwares. The game was designed and implemented with one field, one big battle against the main enemy and a menu.

SISÄLLYS

1 JOHDANTO

Opinnäytetyön tavoitteena oli luoda PC-pelidemo käyttäen Unity pelimoottoria ottamalla inspiraatiota aikaisemmin Oulun GameLabissä toteutetusta pelistämme nimeltä ”Whispering Woods”. Peli-idean muututtua käytimme kuitenkin hyväksi joitain alkuperäisen pelin suunnittelupohjia ajan säästämiseksi. Whispering Woodsia tehdessä meillä oli isompi tiimi käytettävissä, johon kuului erilliset audiosuunnittelijat, artistit ja koodarit, kun taas nykyisessä projektissa tekijöitä oli kolme.

Idea pelin uudelleentoteutukseen syntyi yhteisen työharjoittelun aikana, jolloin päätimme kehittää aikaisempi peli paremmaksi uudella kokemuksella ja paremmalla projektipohjalla. Suoritimme ammattiharjoittelun OAMK:in Metapilot Factory -hankkeessa, jossa olimme mukana osana tiimiä kehittämässä PC-opetuspeliä Lapin yliopistolle. Olemme kokeneita pelaajia ja kiinnostuneita pelikehityksestä, ja halusimme nähdä, pystymmekö toteuttamaan demon omalle tietokonepelille. Artistinamme toiminut Carita oli vaihdossa ulkomailla projektin aloitusvaiheessa, mutta saimme aikataulut sovittua etukäteen. Palattuaan hän pääsi liittymään opinnäytetyöhön ottaen vastuun kokonaan visuaalisesta puolesta teknisenä artistina.

Koska aikataulu oli lyhyt, tarkoituksena oli kehittää toimiva demo pelille. Demoon sisältyy tutoriaalikenttä, jossa opetetaan pelaajalle pelin perusmekaniikat. Perusmekaniikat olivat myös pelissä valmiiksi, kuten pelaajan liikkuminen, taistelumekaniikat, kenttien siirtymät, tietojen tallennus pelaajaprofiiliin sekä vihollisten käyttäytyminen.

2 TIETOPERUSTA JA KÄSITTEET

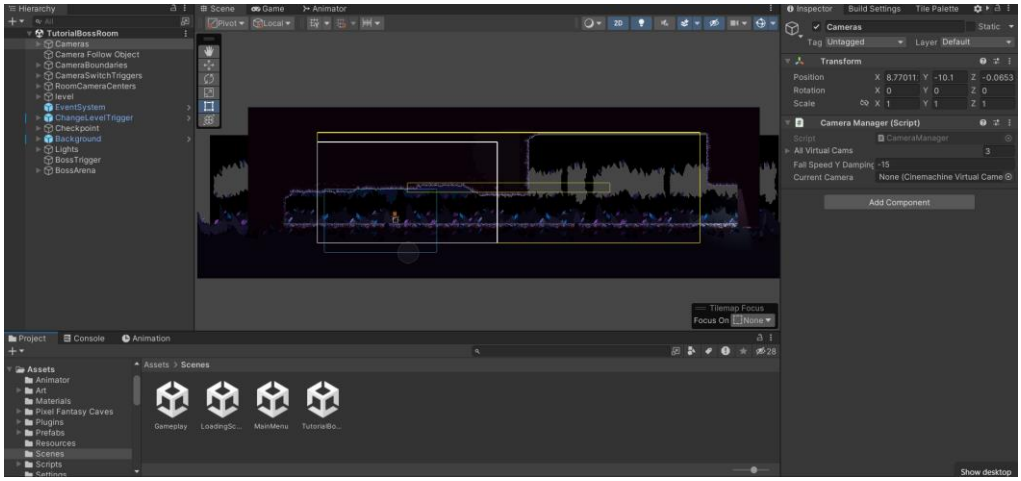
2.1 Pelimoottori

Pelimoottori tarkoittaa ohjelmistokehystä, joka on suunniteltu erityisesti videopelien kehittämiseen. Niiden käyttömahdollisuuden ovat laajat ja niiden avulla voidaan luoda videopelien lisäksi muun muassa mobiilisovelluksia. Tämä tarjoaa kehittäjille valmiita työkaluja ja toiminnallisuuksia, joiden avulla pelien ja sovelluksien luominen on helpompaa ja nopeampaa. Pelimoottorien avulla voidaan kehittää sovelluksia ja pelejä helposti eri alustoille, kuten tietokoneille, mobiililaitteille ja konsoleille. *(Järvinen, N. & Torkkel, J.-M., 2019)*. Tunnetuimpia pelimoottoreita ovat esimerkiksi Unity ja Unreal Engine. Pelimoottorin käyttö säästää huomattavasti aikaa ja resursseja, koska pelin kehittäjien ei tarvitse luoda kaikkia näitä ominaisuuksia alusta alkaen.

2.1.1 Unity

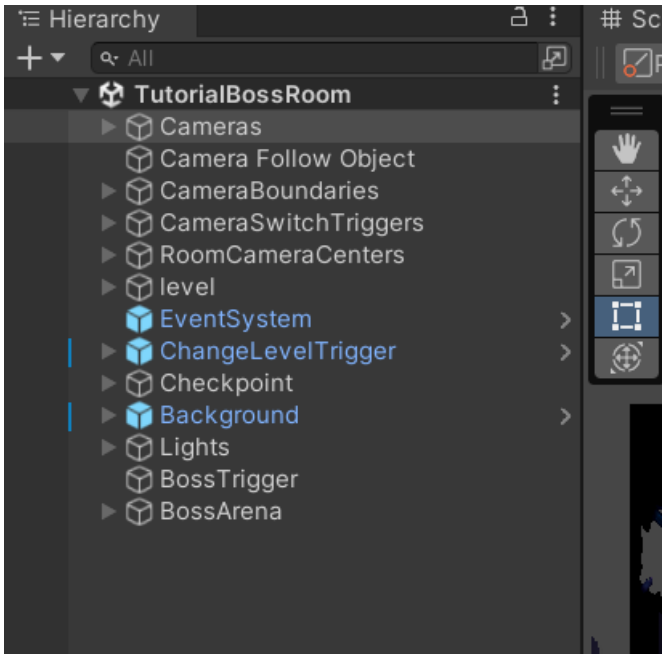
Unity on monipuolinen ja tehokas pelimoottori pelikehitykseen. Se on saavuttanut suuren suosion niin indiekehittäjien kuin suurten pelialan yritysten keskuudessa. Unityn vahvuus piilee sen laajassa työkaluvalikoimassa, joka mahdollistaa sekä 2D- että 3D-pelien kehityksen. Tämä pelimoottori tukee pelien ja sovellusten kehittämistä eri alustoille ja käyttöjärjestelmille, mikä tekee siitä monikäyttöisen työkalun niin peli- kuin ohjelmistokehittäjillekin. Unityn ensisijaisena ohjelmointikielenä toimii C#, joka on monipuolinen ja helppokäyttöinen koodikieli pelien ohjelmoimisessa. Se soveltuu hyvin sekä aloitteleville pelikoodaajille, jotka yrittävät luoda yksinkertaisia pelejä, että kokeneille ohjelmoijille, jotka osaavat luoda monimutkaisempia toiminnallisuuksia pelin sisälle. *(Unity Documentation,2024).*

Unity-editorin päänäkymä (Kuva 1) koostuu useasta tärkeästä ikkunasta, joihin sisältyy muun muassa hierarkiaikkuna, Inspector ja Projektikansio ja konsolinäkymä. Unity-editorin käyttöliittymää pystyy muokkaamaan omien tarpeiden mukaan. Esimerkiksi inspector-näkymässä näkyy lisäksi Build Settings, jossa pystyy muokkaamaan nykyisen peliversion asetuksia, sekä Tile Palette, joka sisältää kenttien luomiseen käytetyt tiilipaketit.



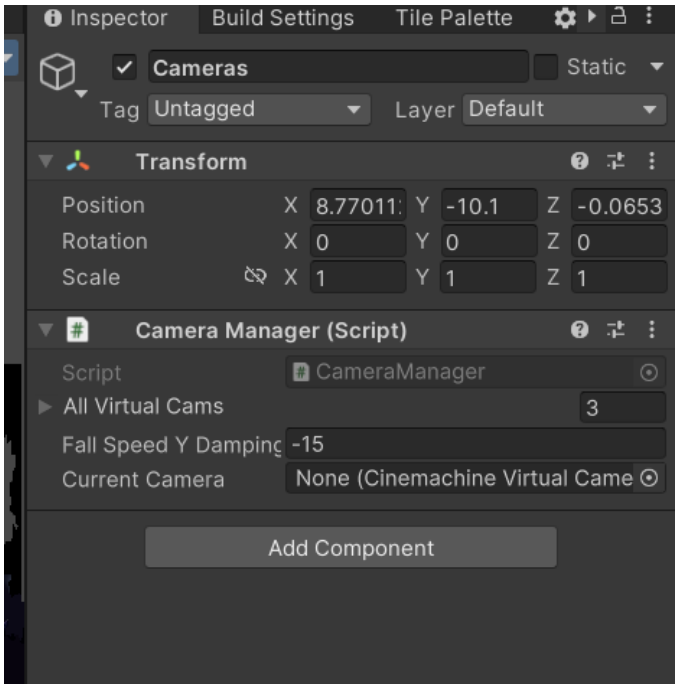
Kuva 1 Unityn UI

Hierarkiaikkuna (*Kuva 2*), jossa näet kaikki kyseisen pelinäkymän sisältämät objektit listana, toimii sisällysluettelona pelinäkymälle. Hierarkiaikkunassa voit luoda ja lisätä objekteja näppärästi pelinäkymään, sekä järjestellä niitä raahaamalla ja pudottamalla. Objekteja voidaan myös ryhmitellä lapsi-isäntä-suhteisiin, jolloin lapsiobjektit perivät isäntäobjektin liikkeet ja muutokset.



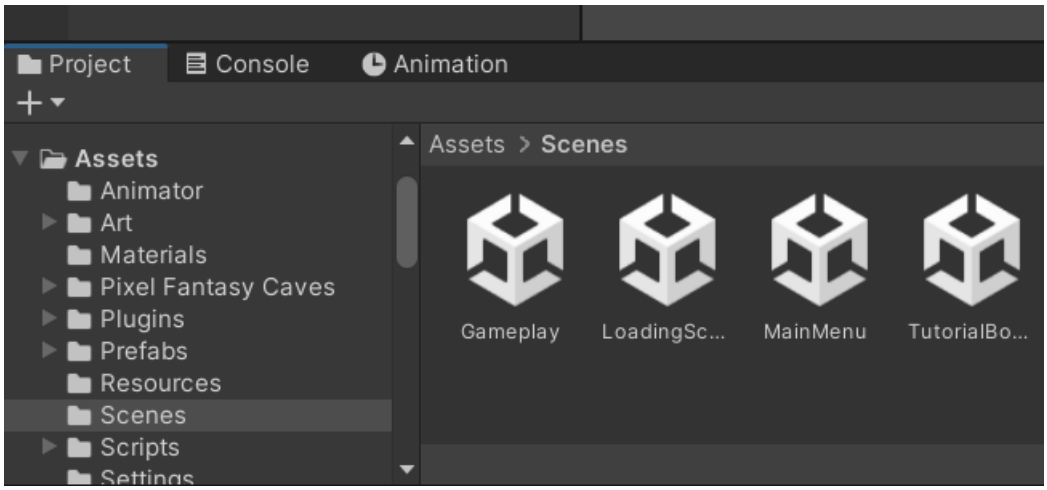
Kuva 2 Unity-editorin hierarkiaikkuna

Inspector-ikkunassa (*Kuva 3*) pystytään muokkaamaan peliobjektien ominaisuuksia, kuten objektien kokoa ja sijaintia, sekä lisäämään objektille komponentteja kuten skriptejä. Kuvassa näkyy inspector-näkymän lisäksi Build Settings. Siinä pystyy muokkaamaan nykyisen peliversion asetuksia, sekä Tile Palette, joka sisältää kenttien luomiseen käytetyt tiilipaketit.



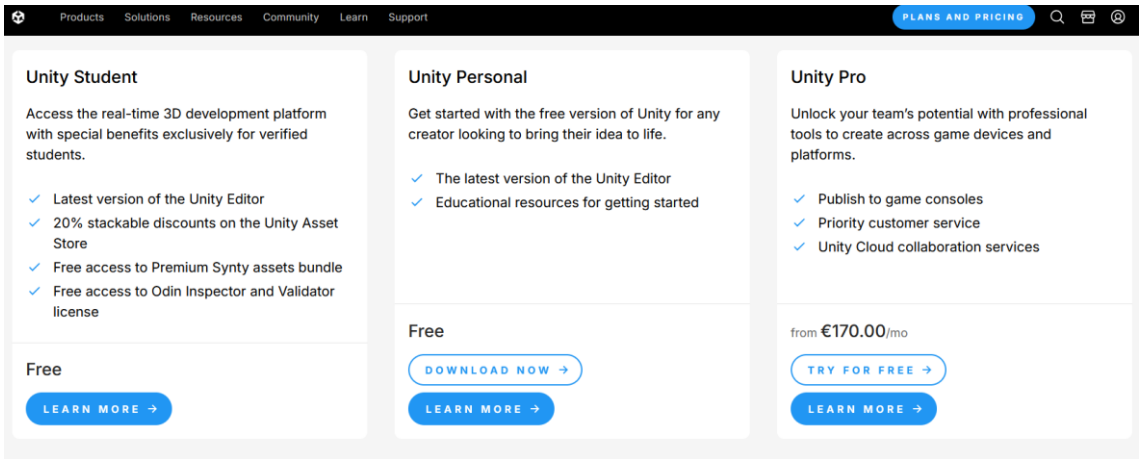
Kuva 3 Inspector-ikkuna

Projektinäkymästä (*Kuva 4*) löytyvät kaikki projektin tiedostot, kuten skriptit, pelinäkymät, kuvat ja äänet. Projektinäkymää käyttäen voit siirtyä kätevästi esimerkiksi pelinäkymästä toiseen. Tästä näkymästä löydät myös projektiin lisätyt laajennukset omista kansioistaan. Kuvan 4 esimerkissä näkyy Pixel Fantasy Caves tillipakettilaajennus, joka on ladattu Unity Asset Storesta.



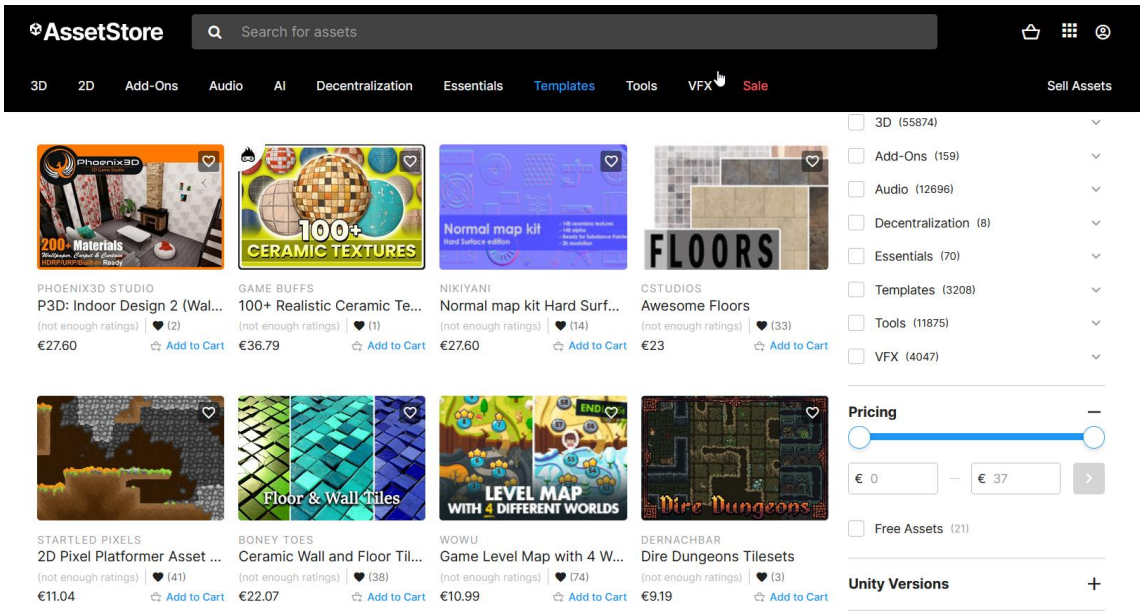
Kuva 4 Unityn Projektinäkymä

Unity tarjoaa erilaisia lisenssivaihtoehtoja pelinkehittäjille (*Kuva 5*). Ilmainen Unity Personal -lisenssi, joka on saatavilla myös opiskelijoille omana Unity Student -versiona, sisältää kaikki perustoiminnallisuudet pelinkehitykseen. Tämän version käyttö edellyttää, että kehittäjän tai organisaation vuositulot pelistä jäävät alle 100 000 dollarin. Kasvaville pelistudioille Unity tarjoaa Plus-lisenssin, jonka vuosituloraja on korkeampi, 200 000 dollaria. Ammattimaiseen pelinkehitykseen suunnattu Unity Pro -lisenssi puolestaan ei aseta rajoituksia tuloille. Nämä maksulliset lisenssit eroavat ilmaisversiosta pääasiassa lisäominaisuuksien, teknisen tuen laajuuden ja brändäyksen osalta. Esimerkiksi Plus- ja Pro-lisenssien käyttäjät voivat halutessaan poistaa Unity-logon peliensä käynnistysnäytöstä, mikä ei ole mahdollista ilmaisversiossa. Lisäksi maksullisiin lisensseihin kuuluu edistyneempiä työkaluja pelin analysointiin ja optimointiin sekä laajempi valikoima pilvipalveluita.



Kuva 5 Unity tarjoaa käyttäjilleen eri lisenssivaihtoehtoja

Unityn omassa Asset Storessa (*Kuva 6*) on tarjolla paljon ilmaisia sekä maksullisia resursseja, kuten grafiikkaa, ääniä, valmiita skriptejä ja Unityn lisäosia, joita voi hyödyntää projektien teossa ajan säästämiseksi. Projektissamme on käytössä Asset Storesta ladattu grafiikkapaketti Pixel Fantasy Caves. Tässä grafiikkapaketissa on pelin teemaan sopivaa pikseligrafiikkaa pelin ensimmäistä tasoa varten.



Kuva 6 Esimerkki Unity Asset Storen valikoimasta

2.1.2 Muita pelimoottoreita

Unityn lisäksi on olemassa lukuisia muita suosittuja pelimoottoreita, kuten Unreal Engine ja GameMaker. Oikean pelimoottorin valinta riippuu peliprojektin teknisistä vaatimuksista sekä pelikehittäjän kokemuksesta ja taidoista.

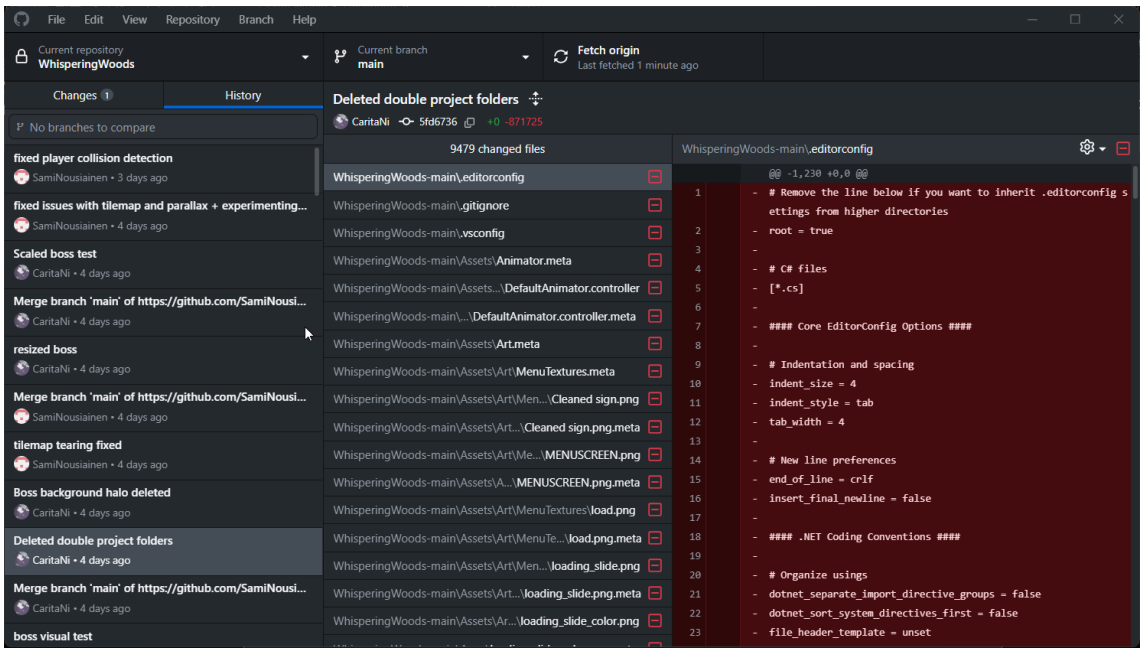
Unreal Engine on Epic Gamesin kehittämä tehokas pelimoottori, joka kilpailee Unityn kanssa pelinkehitysmarkkinoilla. Molemmat ovat suosittuja työkaluja, mutta niillä on merkittäviä eroja. Unreal Engine voi olla Unitya monimutkaisempi työkalu uusille kehittäjille, mutta se tarjoaa laajan valikoiman edistyneempiä ominaisuuksia. Unreal Engine tunnetaan erityisesti vahvasta graafisesta suorituskyvystään, mikä tekee siitä suositun pelimoottorin suurempien AAA-pelien kehityksessä. AAA-pelit ovat korkealaatuisia ja kalliita videopelejä, joilla on suuri budjetti ja jotka on suunniteltu isoille pelialustoille. Ne ovat yleensä alan johtavien peliyhtiöiden kehittämiä huippupelejä, joissa on upea grafiikka ja monipuolinen pelisisältö. Unreal Enginen pääasiallinen ohjelmointikieli on C++.

GameMaker on Mark Overmarsin ja YoYo Gamesin kehittämä pelimoottori, joka on hyvä vaihtoehto aloittelevalle peliohjelmoijalle. GameMakerillä pystytään helposti luomaan yksinkertaisempia 2D-pelejä tietokoneille, konsoleille sekä mobiililaitteille. Ensimmäinen versio julkaistiin marraskuussa 1999, ja päivitetty versio GameMaker 2.0 julkaistiin 2017. GameMakerin ohjelmointikielenä toimii moottorin oma GameMaker Language (GML), joka on yhdistelmä Javascriptistä sekä C#- että C++-kielistä.

2.2 Versionhallinta

Versionhallinta on järjestelmä, joka seuraa ja hallinnoi projektin tiedostoihin tehtyjä muutoksia ajan myötä. Se on erityisen tärkeä ohjelmistokehityksessä, mutta sitä voidaan käyttää minkä tahansa tiedostotyypin hallintaan. Versionhallinta on olennainen osa nykyaikaista ohjelmistokehitystä ja projektinhallintaa, sillä se tehostaa työskentelyä, vähentää virheitä ja mahdollistaa sujuvan yhteistyön kehittäjien välillä. (*The Codest 2024*).

Yksi suosittu versionhallintasovellus on GitHub Desktop (*Kuva 7*). Se on erityisesti GitHub-palvelun käyttöön suunniteltu käyttöliittymä, jonka avulla voit helposti hallita projektin versionhallintaa. GitHub Desktop on helppokäyttöinen sovellus, joka sopii hyvin Git-versionhallinnan opetteluun tai visuaalista käyttöliittymää arvostaville.



Kuva 7 GitHub Desktop sovellus.

Git-projektioiden keskeisin osa on repositorio, eli tietovarasto, joka toimii projektin säilytyspaikkana. Repositorio sisältää kaikki projektin tiedostot ja tiedot tiedostojen muokkaushistoriasta, joka näkyy kuvassa 7. Ryhmäprojekteissa versionhallinta mahdollistaa sen, että jokainen tiimin jäsen voi työstää projektia samanaikaisesti omalla tietokoneellaan. Kun muutokset ovat valmiita, ne voidaan puskea (push) yhteiseen etärepositorioon. Tämän jälkeen muut tiimin jäsenet voivat vetää (pull) nämä päivitykset omiin paikallisiin kopioihinsa projektista.

2.3 Piirtotyökalut

Projektimme taidetyylinä on pikselitaide, joka on digitaalista taidetta, jossa jokainen kuvapiste eli pikseli suunnitellaan huolellisesti. Pikselitaide luodaan usein yksinkertaisilla kuvankäsittelyohjelmilla, mutta se ei suinkaan ole helppoa; väripaletin valinta ja värien käyttö ovat avainasemassa halutun lopputuloksen saavuttamiseksi. *(Redacción Tokio, 13.02.2023).*

Pikselitaide perustuu tarkkaan rasterointiprosessiin, joka mahdollistaa videopelin kaikkien visuaalisten elementtien luomisen pienten värillisten mosaiikkien avulla. 1980- ja 1990-luvut olivat pikselitaiteen kulta-aikaa, ja nykyinen retrotrendi on herättänyt tämän taidemuodon uudelleen henkiin. Pikselitaide oli päätyyppi varhaisissa tietokone- ja konsolipeleissä, kuten esimerkiksi kuuluisa Nintendon julkaisema tasohyppelypeli Super Mario Brothers ja varhaiset Legend Of Zelda- sarjan pelit. *(Trewhitt, Max 22.3.2023).* Nykyaikana pikselitaide peleissä on saavuttanut suurta suosiota pelien, kuten Noita, Undertale ja Stardew Valley myötä.

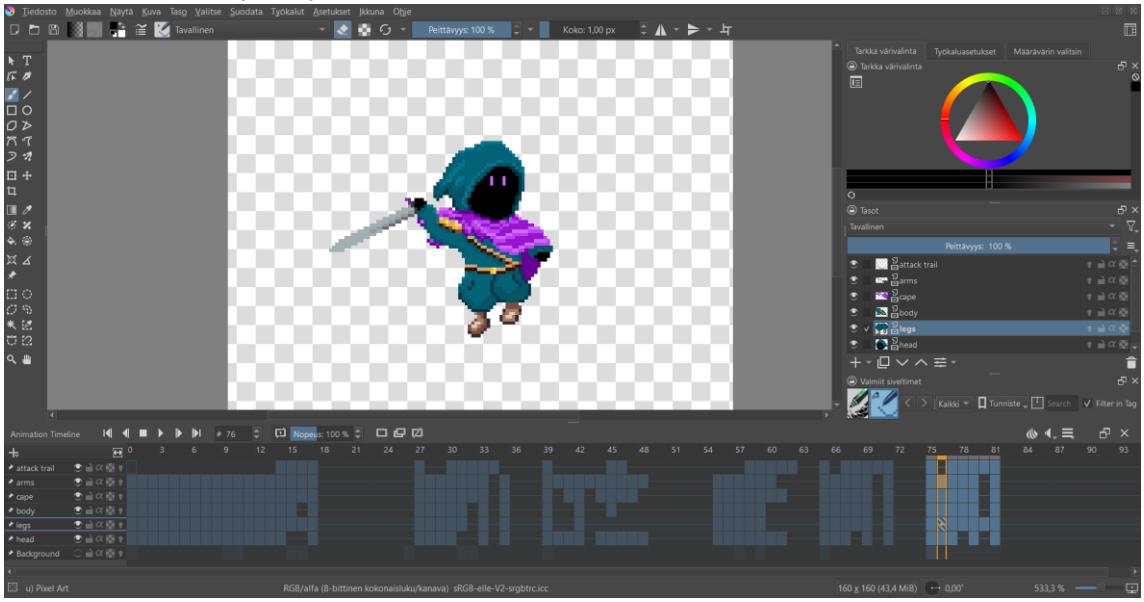
Piksel artin tekemisessä vietetään paljon aikaa kynä- ja viivatyökalujen parissa. Pikselitaiteilijat käyttävät Fill- ja Brush-työkaluja harkiten ja säästeliäästi, sillä yksittäinen pikseli vaikuttaa merkittävästi lopputulokseen. Ohjelmiston valinnassa on tärkeää, että se mahdollistaa näiden työkalujen tarkan ja helpon käytön. *(Monserrate, Emi Adobe 2024).*

PNG- tai GIF-tiedostot ovat yleisimpiä tallennusmuotoja. JPG on yleinen tiedostotyyppi ja usein oletusmuotona, mutta sen käyttämä pakkaus voi heikentää pikselitaiteen laatua ja häiritä tarkkaa pikselikohtaista työtä. PNG on tarkempi pikselitaiteen muuttumattomuuden kannalta ja antaa myös näin mahdollisuuden tallentaa teokset ilman taustaa. (Adobe, 2024).

PNG tulee sanoista ‘’Portable Network Graphic”, joka on suosittu ja patentoimaton tiedostotyyppi. Tämä merkitsee sitä, että PNG tiedostoja voi avata ja luoda melkein millä tahansa ohjelmalla. PNG voi käsitellä 16 miljoonaa eri väriä ja tarjoaa kuvien tallentamisen täysin sekä osittain läpinäkyvän taustan kanssa. PNG-kuvaa voidaan pakata ilman, että kuvan laatu heikkenee tai tietoja menetetään. Tämä tekee sen tallentamisesta ja siirtämisestä helpompaa. Tämä on merkittävä etu verrattuna häviöllisiin formaatteihin, kuten JPEG-tiedostoihin, joissa osa kuvan tiedoista poistuu pakkaamisen yhteydessä. (Adobe, 2024). Taustattoman taiteen saavuttaa esimerkiksi Gimpillä muuttamalla Alpha-kanavan taustan kokonaan läpinäkyväksi.

2.3.1 Krita

Valitsimme Kritan yhdeksi piirto-ohjelmaksi, koska se on ilmainen ja siinä on hyvät työkalut pikseligrafiikan tekemiseen. Piirtonäkymässä (*Kuva 8*) voi ottaa ruudukon käyttöön ja siveltimen saa maalaamaan yhden pikselin kerrallaan. Käyttöliittymä on helposti muokattavissa itselle sopivaksi.



Kuva 8 Kritan piirtonäkymä

Käyttöliittymän alareunassa on aikajana animaatioiden tekemistä varten. Aikajanalla seuraava avainkuva pysyy samana kuin edellinen kunnes siihen tehdään muutoksia. Tämä helpottaa huomattavasti animaatioiden tekemistä. Krita tukee useimpia kuvatiedostomuotoja ja sen asiakirjatiedostomuoto on .kra. Pelaajahahmo ja sen animaatiot on tehty Kritalla. *(Krita Manual, 2024).*

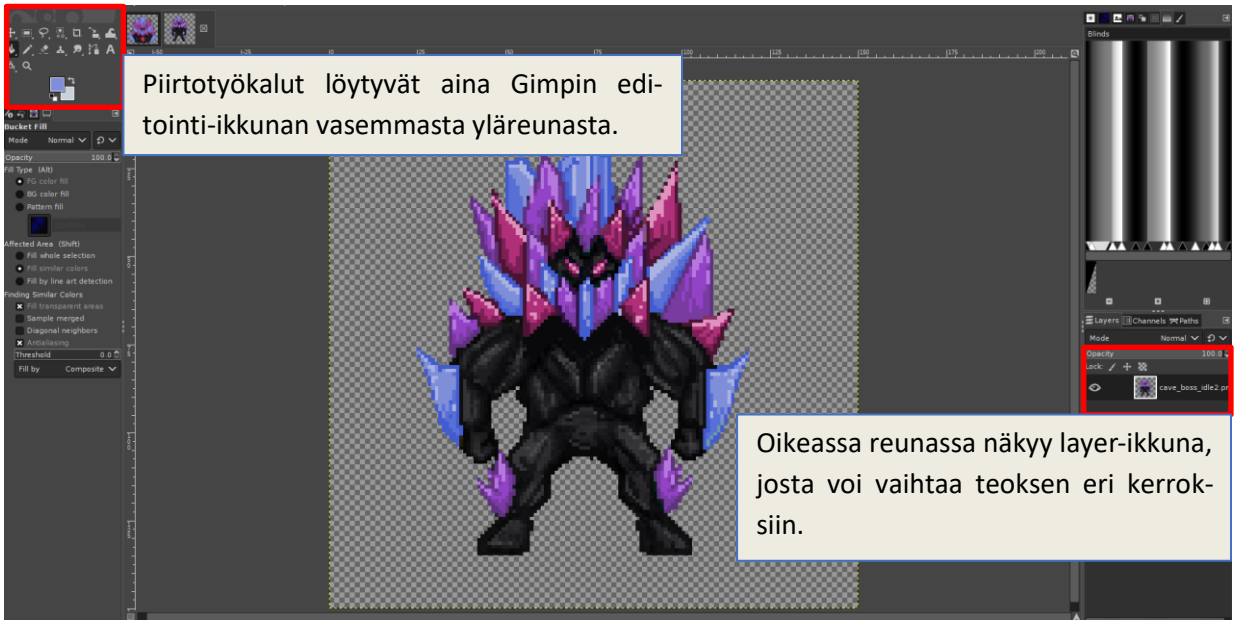
2.3.2 GIMP

Toinen käytössä oleva piirto-ohjelma on GIMP, joka on ilmainen avoimen lähdekoodin kuvaeditori. Projektissa pelaajahahmo tehtiin ennen pääartistin liittymistä mukaan, minkä vuoksi käytämme kahta eri ohjelmaa. Krita oli aloittelevalle piirtäjälle helposti lähestyttävä pikselitaiteeseen keskittynyt ohjelma, kun taas artistimme oli tottunut jo käyttämään kuvaeditoreja. Gimpin kanssa työskennellessä on mahdollista tehdä animaatiot myös ohjelman sisäisesti, mutta opinnäytetyössä on käytetty Unityn animaattoria.

Kun piirtäminen tapahtuu pikselin tarkkuudella, ei ohjelmalta vaadita myöskään mitään tiettyjä toimintoja. GIMPillä voi kuitenkin asettaa ruudukon (grid) taustan päälle ja piirtää sen avulla tarkkuuden takaamiseksi. Voimme myös tehdä erinäisiä efektejä esimerkiksi vihollisille tarvittaessa ja tallentaa projektit PNG- tiedostoina ilman taustaa.

Gimpillä on monia muita ominaisuuksia. Sitä voidaan käyttää muun muassa yksinkertaisena maalausohjelmana, online-eräkäsittelyjärjestelmänä, massatuotantokuvien renderöintilaitteena sekä kuvaformaatin muuntimena. Tuettuja tiedostomuotoja ovat bmp, gif, jpeg, mng, pcx, pdf, png, ps, psd, svg, tiff, tga, xpm ja monet muut. (*Gimp's Team, 2024).*

Editori on myös selkeä ja helppokäyttöinen. Gimpin perusversio on yksinkertaisen selkeä ja se tarjoaa vain perinteiset editointi-, ja piirtovälineet. Tasot (layers) on myös helposti selkeästi näkyvissä ja muokattavissa alusta alkaen (*Kuva 9*).



Kuva 9 Gimpin piirtonäkymä

2.3.3 Smack Studio

Animoidessamme erästä vihollishahmoa käytimme Smack Studiota, joka hyödyntää pikselitaidetta. Smack Studio on Steamissa myynnissä oleva "sandbox"-tyyppinen taistelupeli, jossa on sisäänrakennettu hahmonluontityökalu. Pikselitaiteen animointi on yleensä hidas ja taitoa vaativa prosessi, mutta Smack Studio pyrkii helpottamaan sitä tarjoamalla automatisoituja työkaluja, jotka nopeuttavat prosessia merkittävästi niin harrastajille kuin ammattilaisille.

Smack Studiossa animaatioiden luominen onnistuu siten, että jokainen kehon osa piirretään vain kerran, minkä jälkeen ne kiinnitetään valmiiseen animaatiorigiin. Ohjelmisto huolehtii kehon osien uudelleen piirtämisestä automaattisesti. (*ThirdPixel Interactive, 2024).*

Smack Studio antaa myös käyttäjilleen luvan hyödyntää pelissä luotuja hahmoja ja animaatioita muissa projekteissa. Tekijänoikeuskysymyksiin liittyen he vastaavat verkkosivuillaan seuraavasti:
"Export your animations for use in any project you like: we hold no legal claim over any IP created in Smack Studio, and we'd love for artists to create, share, monetize, and do what they please with their creations" (ThirdPixel Interactive, 2024).

Ohjelman käyttöön päädyimme toisen projektin innoittamana, jossa Smack Studiota hyödynnettiin pelin sisäisten hahmojen luomiseen. Automatisointi säästi paljon aikaa, mikä mahdollisti päävihollisen manuaalisesti tehtyjen animaatioiden tarkemman viilaamisen.

3 PELIN SUUNNITTELU

3.1 Peli-idean suunnitteleminen

Tässä osiossa käymme läpi Game design -dokumenttia, jossa avaamme pelin perusrakennetta ja sisältöä. Selitämme tarkemmin pelimme gameplay-elementeistä, genrestä ja kenttien rakenteesta. GameLab-pelikehityskurssin aikana loimme Metroidvania-tyylisen pelin nimeltä ”Whispering Woods”. Pidimme peli-ideasta, mutta lyhyen aikataulun ja vähäisen kokemuksen takia emme saaneet toteutettua pelistä sellaista kuin alun perin suunnittelimme.

Ammattiharjoittelun myötä pelikehityskokemus kasvoi, joten saimme idean toteuttaa kyseinen peli uudestaan paremmalla projektipohjalla. Idean kehittyessä huomasimme, että työstämämme peli on hyvin erilainen alkuperäisestä pelistä, joten päätimme luoda kokonaan uuden pelin vanhan pohjalta.

Alkuperäinen peli luotiin mobiilipeliksi, ja siinä oli vahvana elementtinä mimic-tyyliset viholliset, jotka maastoutuvat puiksi, pensaiksi, kiviksi ja muiksi ympäristöelementeiksi. Idea kyseisessä pelissä oli arvaamattomuus ja yllätyselementit, joten pelaaja ei voinut tietää, onko vastaantuleva puu vihollinen vai tavallinen puu. Nykyisessä pelissä nojaamme enemmän klassisiin fantasiaelementteihin. Tämän lisäksi pelimme on suoraviivaisempi kuin Whispering Woods, jossa oli enemmän vapautta pelattavien kenttien järjestyksellä.

3.2 Game Design Document

Game design document (GDD) on yksityiskohtainen suunnitelma pelin konseptista, jossa määritellään pelin kaikki keskeiset ominaisuudet, mekaniikat, tarinat ja tavoitteet. Se toimii ikään kuin pelin "käsikirjoituksena", jota kehitystiimi käyttää apuna pelin suunnittelussa ja toteutuksessa, varmistaen että kaikki tiimin jäsenet ymmärtävät pelin vision.

3.2.1 Gameplay

Pelin kulku on selkeää tasohyppelyä. Tämä tarkoittaa, että hahmo liikkuu vaakasuunnassa eteenpäin ja voi hyppiä tasojen, esteiden ja vihollisten yli. Koska peli kuvataan sivulta, pelaaja näkee helposti, mitä tapahtuu hahmon edessä. Pelaaja ohjaa hahmoa näppäimistön WASD-näppäimillä tai nuolinäppäimillä. Jos pelaaja haluaa, voi käyttää myös peliohjainta. Tämä auttaa suunnittelemaan liikkeitä ja välttämään vaaroja.
Pelissä voi olla erilaisia haasteita, kuten hyppiminen esteiden yli, vihollisten väistäminen ja erilaisten tasojen tutkiminen. Pelaajahahmon liikkumiseen käytetään Unityn Rigidbody2D-komponenttia, jolla voidaan antaa fysiikat peliobjekteille. Pelaaja saadaan liikkumaan lisäämällä Rigidbody2D:lle nopeutta haluttuun suuntaan. Collider2D-komponentti tarkistaa pelaajahahmon törmäykset sen ympäristön kanssa. Tällä komponentilla voidaan valita mihin kaikkeen pelaaja voi törmätä. Törmäykset voidaan säätää taulukosta Unityn asetuksissa tai koodissa, jos halutaan esimerkiksi tietty törmäys päälle tai pois vain hetkellisesti.
Pelissä on tarkoitus toteuttaa opinnäytetyöhön yksi tutoriaalikenttä ja yksi päävihollisen huone. Tutoriaalikentässä keskitytään tasohyppelyyn ja pelissä etenemiseen. Päävihollisen huone tulee kentän lopussa, jonne pelaajan täytyy päästä kiertämällä sortunut kenttäosio kiertotien kautta.

3.2.2 Peligenre

Pelin genrenä toimii Metroidvania. Metroidvania yhdistää elementtejä kahdesta klassisesta pelisarjasta: Metroidista ja Castlevaniasta. Muita genren modernimpia klassikkopelejä ovat muun muassa Hollow Knight, Dead Cells ja Ori and the Blind Forest. Peligenren nimi kuvaa pelejä, joilla on tietyt yhteiset ominaisuudet ja pelimekaniikat. Yksi metroidvanian keskeisiä ominaisuuksia on laaja ja yhtenäinen maailma, joka on useasti luotu 2D-grafiikalla. Tämän lisäksi genren keskeisiä piirteitä ovat pelihahmon kehittyminen ja uusien kykyjen saaminen, jotka avaavat pääsyn uusille alueille. Pelimaailma avautuu vähitellen pelaajan tutkiessa ympäristöä. *(Bitmob 2010)*

3.2.3 Taistelumekaniikat

Pelaajalla on käytössään erilaisia työkaluja vihollisten päihittämistä varten. Pelin alussa pelaajalla on tavallinen miekka, jolla voi lyödä eteen, ylös ja pelaajan ilmassa ollessa myös alaspäin. Myöhemmin pelaaja voi löytää myös muita lyömäaseita, kuten suuren ja hitaan kahden käden miekan tai lyhyen mutta nopean tikarin. Näillä vaihtoehdoilla pelaaja saa valita omaan pelityyliinsä sopivan aseen. Pelin edetessä pelaaja saa käyttöönsä myös taikoja, joiden avulla vihollisiin voidaan vaikuttaa kauempaakin. Taikojen käyttäminen kuluttaa taikapisteitä, joita saa takaisin käyttämällä siihen tarkoitettuja kulutettavia esineitä tai käymällä tallennuspisteellä.

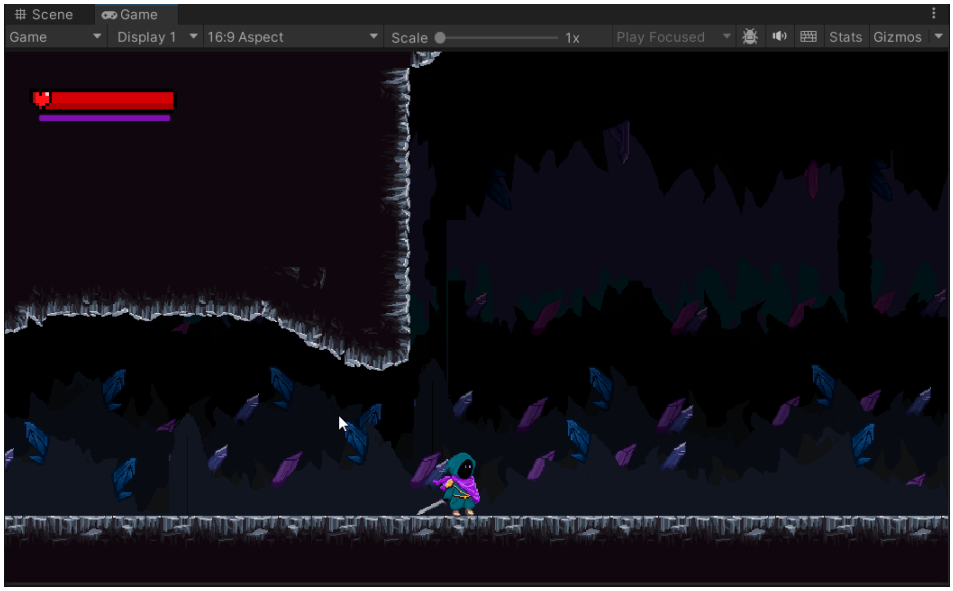
Pelaaja voi väistää vihollisten iskut liikkumalla, hyppäämällä tai käyttämällä syöksyominaisuutta. Syöksy liikuttaa nopeasti pelaajahahmoa lyhyen matkan sen katsomissuuntaan. Syöksyn aikana pelaaja on immuuni vahingolle, mutta sen aikana ei voi tehdä mitään muuta. Vihollisten osuessa pelaajaan iskuillaan pelaaja menettää elämäpisteitä. Elämäpisteitä saa takaisin käyttämällä siihen tarkoitetun esineen tai käymällä tallennuspisteellä. Jos pelaajan elämäpisteet loppuvat, pelaajahahmo kuolee ja joutuu takaisin edelliselle tallennuspisteelle.

3.2.4 Kamera

Kameran käyttäytymisestä halusimme tehdä mahdollisimman luonnollisen tuntuisen. Otimme inspiraatiota Hollow Knightin kameran toiminnallisuuksista suunnitellessa omaa kamerasysteemiä. Kamera liikkuu hieman pelaajahahmon edellä antaen pelaajalle enemmän aikaa reagoida kaikkeen edessä olevaan. Se lähtee liikkeelle pelaajan liikkuessa pienellä viiveellä ja vaihtaa suuntaansa sulavasti, kun pelaaja kääntyy katsomaan toiseen suuntaan. Näillä ominaisuuksilla kameran liike saadaan tuntumaan vähemmän kankealta. Kameran liikettä halutaan myös rajoittaa eri tilanteissa. Tietyissä huoneissa kamera keskitetään huoneen keskelle ja lukitaan paikalleen riippumatta siitä, missä pelaaja on. Tätä ominaisuutta käytetään pääasiassa salahuoneissa ja päävihollisten huoneissa. Ahtaissa tiloissa kameraa estetään liikkumasta pystysuunnassa. Pelikenttiin asetetaan kohtia, joissa kameran keskitystä siirretään haluttuun suuntaan. Tällä ominaisuudella voidaan näyttää pelaajalle haluttuja ympäristön avainkohtia.

3.2.5 Kenttäsuunnittelu

Peli alkaa tutoriaalikenttänä toimivasta luolakentästä (*Kuva 10*), jossa pelaajalle opetetaan pelin perustoiminnallisuus. Luolan lopusta löytyy huone, jossa on pelin ensimmäinen päävastus eli boss *(kuva 11)*, joka pitää päihittää ennen kuin voi edetä varsinaiseen pelimaailmaan. Päätimme erotella kenttien sisäiset alueet omiin pienempiin huoneisiin. Kun kentät ladataan pienempinä huoneina, se pienentää kerralla ladattavien asioiden määrää parantaen suorituskykyä. Samalla tämä saa maailman tuntumaan isommalta ja alueet tuntuvat olevan yhteyksissä toisiinsa.



Kuva 10 Kuvankaappaus keskeneräisestä luolakentästä.

Kuva 11 Hahmottelua bossihuoneesta – kokoerot ja skaalaus

Luolakentän läpäistyään pelaaja etenee luolan ulkopuolella sijaitsevaan kyläalueeseen, jossa hän voi asioida kauppiaiden ja muiden ei-pelattavien hahmojen (NPC-hahmojen) kanssa. Tämä alue toimii pelaajalle lepoalueena, jossa voi päivittää hahmoaan, ostaa tarvikkeita ja saada lisätietoa maailmasta NPC-hahmoilta. Kylästä avautuu useita reittejä, joiden kautta pelaaja voi edetä ensimmäiseen isompaan kenttään.

3.2.6 Pelaajahahmo

Pelaajahahmo on pysynyt hyvin lähellä alkuperäistä hahmosuunnitelmaa. Hahmon suunnitteli graafikkomme Carita edellisessä peliprojektissa, ja sen toteutti myöhemmin toinen silloisen ryhmän graafikko. Tässä projektissa Sami piirsi ja animoi hahmon sopimaan paremmin päivitettyyn taidetyyliin.

Alkuperäinen suunnitelma (*Kuva 12*) oli yksinkertainen luonnos pelaajahahmon ulkonäöstä. Tämä luonnos toimi inspiraationa lopulliselle hahmon ulkomuodolle. Halusimme pitää hahmon mystisyyden peittämällä hänen kasvonsa, joka pitää hahmon ulkomuodon myös yksinkertaisena ja neutraalina. Kaapumainen vaatetus ja viitta tuo seikkailijamaista ulkomuotoa (*Kuva 13*).

Kuva 12 Ensimmäinen suunnitelma pelaajahahmosta, joka muutti muotoaan pelin kehittyessä.

Kuva 13 Pelaajahahmon suunnittelun loppuvaihe - värikartoitus

3.2.7 Päävihollinen (Boss)

Päävihollliset ovat tärkeä osa miljöötä ja pelikokemusta. Halusimme, että päävihollliset olisivat perusvihollisista poiketen pelaajahahmoa suurempia. Tästä alettiin miettimään visuaalisesti miljööhön sopivia luonnoksia. Kenttä on maanalainen luolasto, joka antoi aihealueen. Inspiraatiota haettiin tekoälyä hyödyntäen.

Persoonallisen ja uniikin ulkomuodon takaamiseksi tekoälyn tarjoamat kuvat ovat inspiraation lähteenä, mutta kaikki designit ovat kuitenkin graafikon omaa käsialaa (*Kuva 14*). Ensimmäinen päävastus on pelaajahahmoa muistuttava viitallinen, aavemainen hahmo. Päävihollisen pään päällä on kolme kristallia, joista suurin toimii yhdessä hyökkäyksessä vihollisen vahinkoa tuottavana objektina. Päävihollisen nimi, ‘’Gatekeeper”, erottaa bossin myös normaaleista, alempitasoisista vihollisista. Päävihollisilla on useampia erilaisia hyökkäyksiä.

Kuva 14 Raakasuunnitelma Gatekeeperin syöksyhyökkäyksestä.

Toisen päävihollisen suunnitelma on myöhemmin pelissä esiintyvä suuri kristallijättiläinen, joka on mahtipontisesti kristallien peittämä. Se on myös merkittävästi suurempi pelaajahahmoa joka tuo uhan ja voimakkuuden tunnetta paljon enemmän aikaisempaan päävastukseen verrattuna. Tämä on valmiiksi ideoitu myöhempää kehittämistä varten.

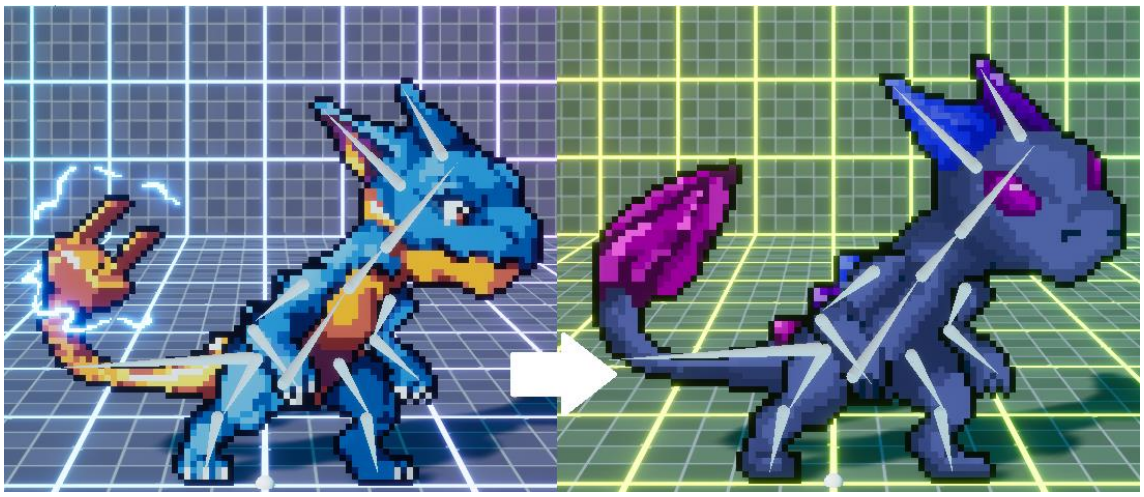
3.2.8 Pienemmät viholliset (Enemy entities)

Pieniä vihollisia on kahta tyyppiä. Paikallaan pysyvä passiivinen sieni hyökkää vain pelaajan ollessa tarpeeksi lähellä tekemällä viiveellä kaasupilven, joka tekee pientä vahinkoa pelaajaan. Sieneen koskiessa pelaaja ottaa myös vahinkoa. Sienen hyökkäysmekanismi on kuitenkin helppo myös väistää. Tällä opetetaan pelaajaa väistämään vihollisten hyökkäyksiä ja ympäristöön asetettuja esteitä, mutta ei vielä haasteta pelaajaa oikeaan taisteluun.

Toinen pieni vihollistyyppi oli tärkeä saada jotenkin reaktiiviseksi, mikä auttaisi pelaajaa tottumaan liikkumisen lisäksi taistelemaan. Kuvassa 15 suunnittelimme lisää hahmoja ideoinnin tueksi. Vihreä limahahmo olisi melee-tyyppinen lähitaisteluun perustuva hahmo, kuten myös alareunassa oleva eläintä muistuttava hahmo. Ylhäällä oikealla oli ainoa kaukohyökkäyshahmo. Se muistuttaa ulkoisesti päävastusta ja ampuu pelaajaa kaukohyökkäyksellä, joita väistelemällä pelaajan pitää päästä lähelle hahmoa lyömään sitä.

Kuva 15 Hahmosuunnittelua vihollisista

Smack Studiossa ihastuimme valmiiseen “Galvan”- nimiseen hahmoon (*Kuva 16*), jonka hyödynsimme omalla piirtotyylillämme melee- tyyppiseksi viholliseksi. Se hyökkää lyömällä pelaajaa käsillään. Tämä vihollinen partioi sille asetetulla alueella ja lähtee jahtaamaan pelaajaa, jos pelaaja tulee tarpeeksi lähelle sitä.



Kuva 16 Galvanin alkuperäinen muoto ja muokattu muoto

4 PELIN TOTEUTUS

Tämän pelin tekemiseen valitsimme pelimoottoriksi Unityn, koska sen käytöstä meillä on aikaisempaa kokemusta aikaisempien projektien teossa. Unity on monipuolinen ja helppokäyttöinen työkalu sekä PC - että mobiilipelien luomiseen. Sitä on käytetty monien menestyneiden pelien teossa, kuten Hollow Knight ja Blasphemous, jotka toimivat inspiraation lähteenä omassa pelissämme.

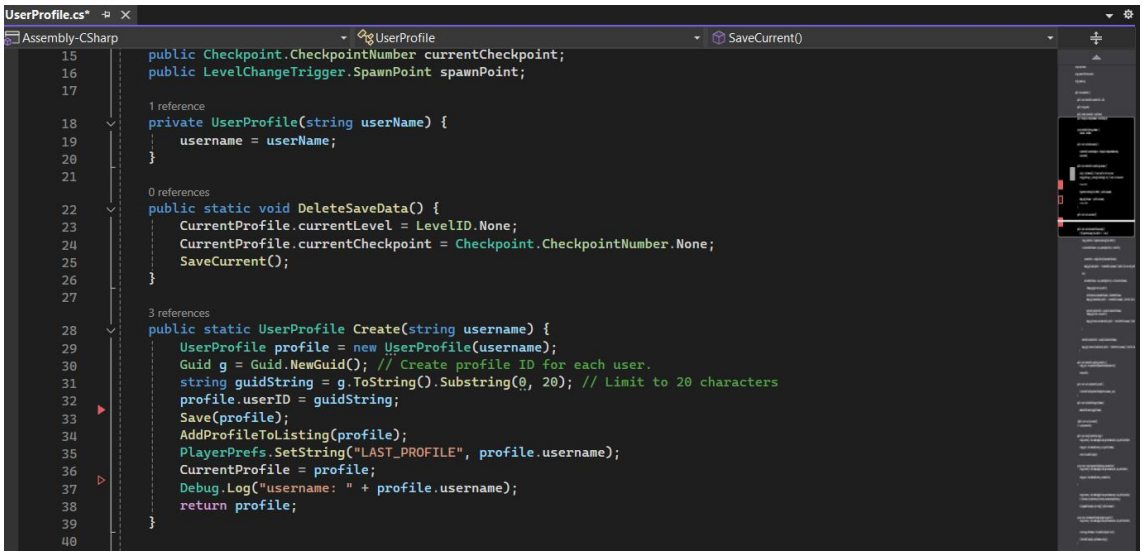
Versionhallintaan valitsimme työkalun nimeltä GitHub Desktop, jota käytimme aikaisemmissakin projekteissamme. GitHub Desktopin kautta GitHubin tietovarastoon tallennetaan projektin tiedostoihin tehdyt muutokset. Tämän avulla projektin jäsenet voivat työstää yhtäaikaisesti samaa projektia, ja puskea omat valmiit muutokset tietovaraston päähaaraan sekä vetää muiden jäsenten tekemät muutokset. Tämä aiheuttaa toisinaan ongelmia, jos useampi henkilö työstää samaa tiedostoa yhtäaikaisesti.

GitHub Desktop kuitenkin sisältää hyviä työkaluja ja ominaisuuksia muutoksien yhdistämiseen useasta eri lähteestä, joten nämä ongelmat korjaantuvat usein nopeasti ja helposti. Nämäkin ongelmat voidaan ehkäistä kommunikoimalla selkeästi mitä tiedostoja kukakin työstää sillä hetkellä.

4.1 Projektipohjan lisääminen

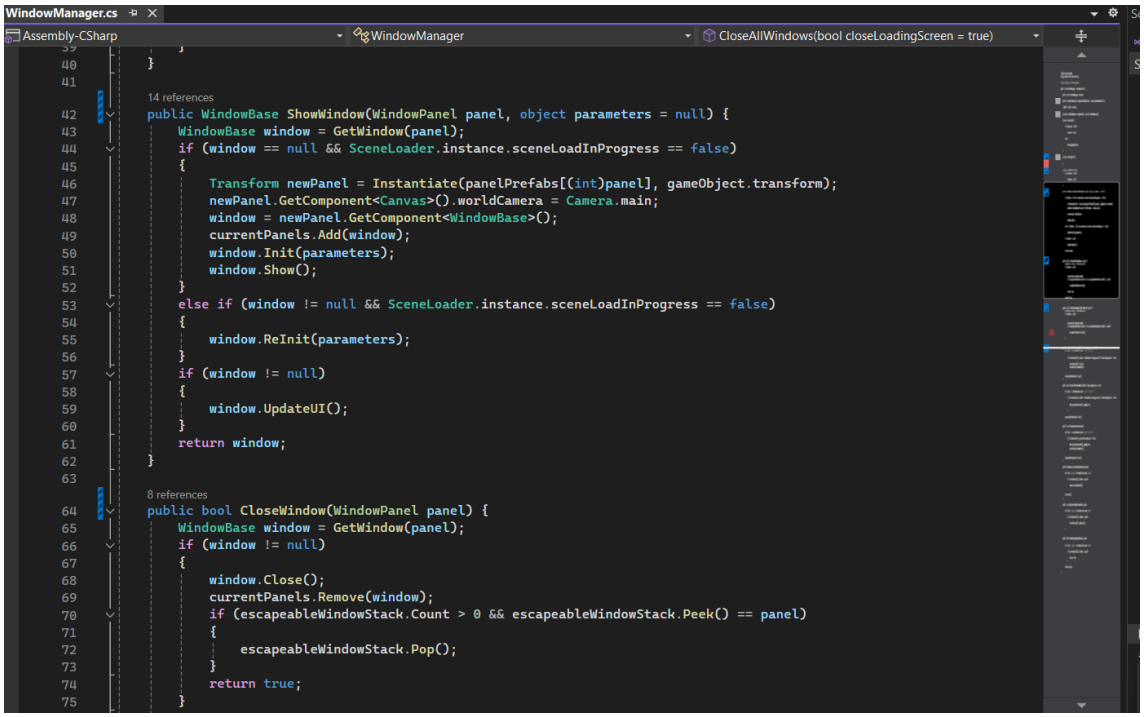
Projektipohja on projektin runko, joka sisältää keskeisten skriptien pohjat. Selkeä projektipohja nopeuttaa ja helpottaa peliprojektin luontia. Tähän projektiin adoptoimme OAMK:in MetaPilot Factory- peliprojektissa käyttämämme projektipohjan todettuamme sen monipuoliseksi ja toimivaksi. Tässä osiossa avaamme projektipohjan tärkeämpiä osioita ja selitämme miten ne toimivat.

Userprofile-skripti (*Kuva 17)* sisältää logiikat pelaaja profiilin luomiseen, olemassa olevien profiilien hakemiseen, profiilin valitsemiseen sekä valittuun profiiliin tietojen tallennukseen. Kuvassa 17 näkyy esimerkki userprofile skriptin sisältämistä funktioista. Create-funktiossa luodaan uusi pelaajaprofiili. DeleteSaveData-funktiossa määritetään mitä tietoja valitusta pelaajaprofiilista poistetaan funktiota kutsuttaessa.



Kuva 17 Userprofile-skripti

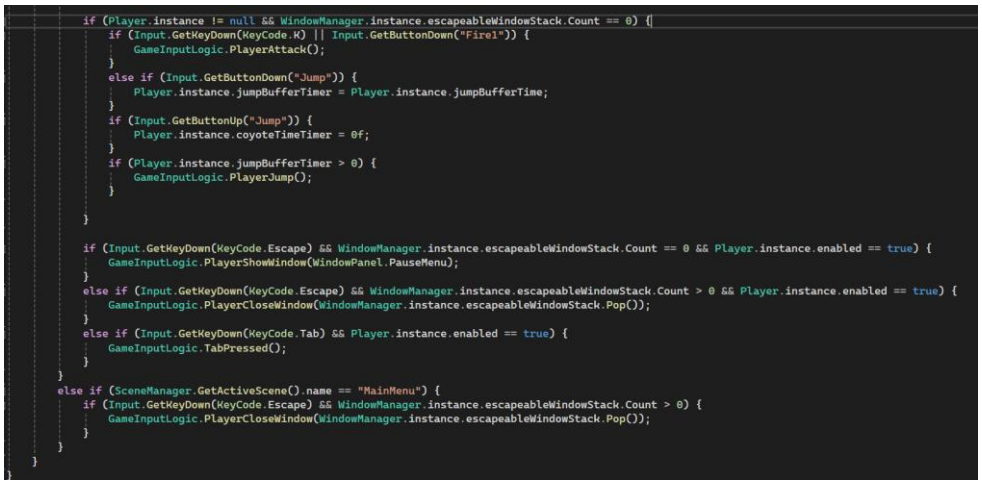
Windowmanager (*Kuva 18*) käsittelee kaikkia UI-ikkunoita, kuten Pelaajahahmon HUD:ia ja taukovalikkoa. ShowWindow-funktiota kutsumalla voidaan avata haluttuja ikkunoita, kuten taukovalikkoikkuna tai inventaarioikkuna, kun taas CloseWindow-funktiota käyttämällä ne voidaan sulkea.



Kuva 18 Windowmanager-skriptin “ShowWindow” ja “CloseWindow”-funktiot

Enum (lyh. enumeraatio) on C#-ohjelmointikielessä käytettävä tietotyyppi, jota voidaan käyttää muuttujissa, parametreissä ja paluuarvoissa. Projektipohjassa tallensimme kaikki enumit samaan tiedostoon helpottaen näiden kutsumista muissa skripteissä.

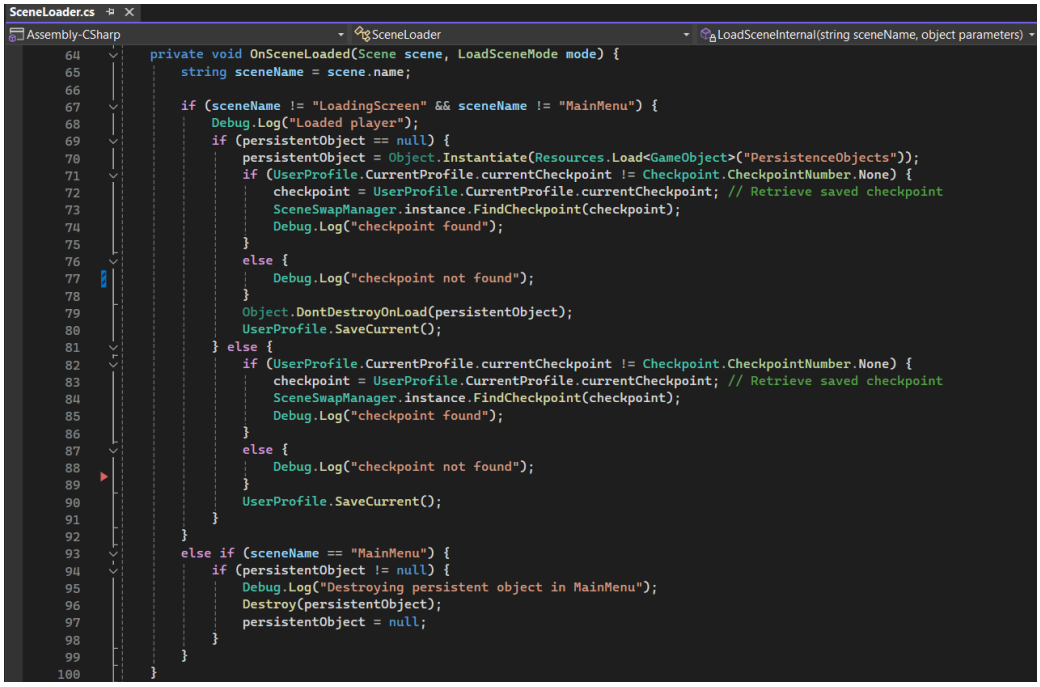
KeyInputManagerissa (*Kuva 19*) määritetään funktiokutsut tiettyä näppäintä painaessa. Skriptissä voidaan määrittää ehtoja funktiokutsuille, kuten esimerkiksi sen, että jos taukovalikkoikkuna on auki, inventaarioikkunaa ei voi avata yhtäaikaisesti. Mutta jos muita ikkunoita ei ole auki, inventaarion saa auki painamalla Tab-nappia. Jos pelaajahahmo on olemassa eikä ikkunoita ole auki, pelaaja voi lyödä painamalla lyöntinappia tai hypätä painamalla hyppynappia.



```
        if (Player.instance != null && WindowManager.instance.escapeableWindowStack.Count == 0) {
            if (Input.GetKeyDown(KeyCode.K) || Input.GetButtonDown("Fire1")) {
                GameInputLogic.PlayerAttack();
            }
            else if (Input.GetButtonDown("Jump")) {
                Player.instance.jumpBufferTimer = Player.instance.jumpBufferTime;
            }
            if (Input.GetButtonUp("Jump")) {
                Player.instance.coyoteTimeTimer = 0f;
            }
            if (Player.instance.jumpBufferTimer > 0) {
                GameInputLogic.PlayerJump();
            }

        }

        if (Input.GetKeyDown(KeyCode.Escape) && WindowManager.instance.escapeableWindowStack.Count == 0 && Player.instance.enabled == true) {
            GameInputLogic.PlayerShowWindow(WindowPanel.PauseMenu);
        }
        else if (Input.GetKeyDown(KeyCode.Escape) && WindowManager.instance.escapeableWindowStack.Count > 0 && Player.instance.enabled == true) {
            GameInputLogic.PlayerCloseWindow(WindowManager.instance.escapeableWindowStack.Pop());
        }
        else if (Input.GetKeyDown(KeyCode.Tab) && Player.instance.enabled == true) {
            GameInputLogic.TabPressed();
        }
    }
    else if (SceneManager.GetActiveScene().name == "MainMenu") {
        if (Input.GetKeyDown(KeyCode.Escape) && WindowManager.instance.escapeableWindowStack.Count > 0) {
            GameInputLogic.PlayerCloseWindow(WindowManager.instance.escapeableWindowStack.Pop());
        }
    }
}
```

Kuva 19 KeyInputManager-skripti

Tarvitsemme skriptin skenejen vaihtamiseen ja lataamiseen, koska jaoimme kentät ja huoneet omiin skeneihinsä. Kuvassa 20 näkyy SceneLoader-skriptin OnSceneLoaded-funktio, jonka sisälle voimme asettaa erilaisia ehtolauseita ladatuille skeneille nimen perusteella. Esimerkiksi funktion ensimmäinen ehtolause tarkistaa, että jos nykyisen skenen nimi ei ole ”LoadingScreen” tai ”MainMenu”, silloin ladataan pelaajaobjekti. Userprofile-skriptissä tarkistetaan, onko valittuun pelaajaprofiiliin tallennettu ”checkpoint” eli tallennuspiste. Jos sellainen löytyy, kutsutaan funktio, joka siirtää pelaajaobjektin tallennuspisteen kohdalle. Jos tallennuspistettä ei löydy, oletetaan uuden pelin aloitus, ja pelaaja siirretään pelin alkuun.



SceneLoader.cs

Assembly-CSharp | SceneLoader | LoadSceneInternal(string sceneName, object parameters)

```
64      private void OnSceneLoaded(Scene scene, LoadSceneMode mode) {
65          string sceneName = scene.name;
66
67          if (sceneName != "LoadingScreen" && sceneName != "MainMenu") {
68              Debug.Log("Loaded player");
69              if (persistentObject == null) {
70                  persistentObject = Object.Instantiate(Resources.Load<GameObject>("PersistenceObjects"));
71                  if (UserProfile.CurrentProfile.currentCheckpoint != Checkpoint.CheckpointNumber.None) {
72                      checkpoint = UserProfile.CurrentProfile.currentCheckpoint; // Retrieve saved checkpoint
73                      SceneSwapManager.instance.FindCheckpoint(checkpoint);
74                      Debug.Log("checkpoint found");
75                  }
76                  else {
77                      Debug.Log("checkpoint not found");
78                  }
79                  Object.DontDestroyOnLoad(persistentObject);
80                  UserProfile.SaveCurrent();
81              } else {
82                  if (UserProfile.CurrentProfile.currentCheckpoint != Checkpoint.CheckpointNumber.None) {
83                      checkpoint = UserProfile.CurrentProfile.currentCheckpoint; // Retrieve saved checkpoint
84                      SceneSwapManager.instance.FindCheckpoint(checkpoint);
85                      Debug.Log("checkpoint found");
86                  }
87                  else {
88                      Debug.Log("checkpoint not found");
89                  }
90                  UserProfile.SaveCurrent();
91              }
92          }
93          else if (sceneName == "MainMenu") {
94              if (persistentObject != null) {
95                  Debug.Log("Destroying persistent object in MainMenu");
96                  Destroy(persistentObject);
97                  persistentObject = null;
98              }
99          }
100     }
```

Kuva 20 Esimerkki SceneLoader-skriptin funktiosta

4.2 Pelin perustoiminnallisuuksien luominen

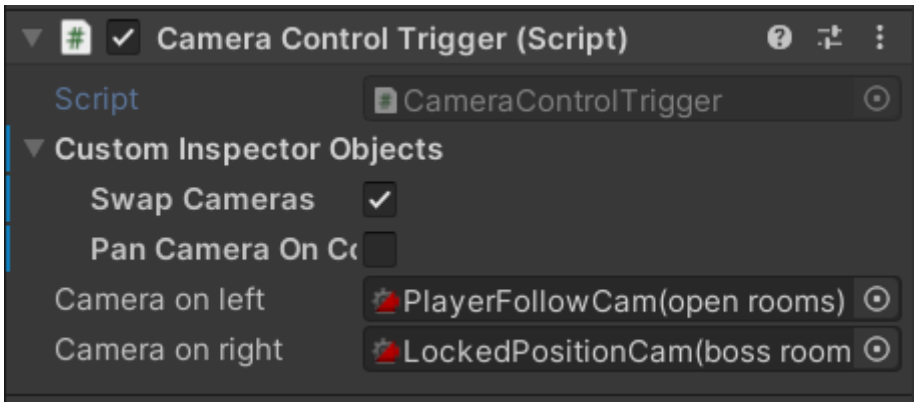
Ohjelmointi aloitettiin pelaajan liikkumisesta. Pelissä on käytetty Unityn sisäisiä fysiikoita, joiden avulla pelaajan liikkumisnopeutta ja painovoimaa säädellään. Pelaajaa liikutetaan lisäämällä nopeutta suuntanäppäintä painettaessa sen mukaiseen suuntaan *(Kuva 21).* Hyppääminen lisää liikkumisnopeutta ylöspäin ja pelaajahahmon massa tiputtaa sen takaisin maahan. Näppäimistöllä käytetään A ja D- näppäimiä liikkumiseen ja välilyöntiä hyppäämiseen. Peliohjaimella käytetään vasenta sauvaa liikkumiseen ja hyppäämiseen alinta toimintapainiketta.

```
void Move() {
    if (canMove == true) {
        moveInputX = Input.GetAxisRaw("Horizontal");
        moveInputY = Input.GetAxis("Vertical");

        if (isAttacking == true && Grounded() == true) {
            rb.velocity = Vector2.zero;
        }
        else {
            if (moveInputX >= 0.1f) {
                rb.velocity = new Vector2(moveSpeed, rb.velocity.y);
            }
            if (moveInputX <= -0.1f) {
                rb.velocity = new Vector2(-moveSpeed, rb.velocity.y);
            }
            if (moveInputX == 0f) {
                rb.velocity = new Vector2(0f, rb.velocity.y);
            }
        }
    }
}
```

Kuva 21 Funktio, jolla pelaajahahmoa liikutetaan

Liikkumisen jälkeen tehtiin kamerasysteemi. Peli käyttää pääasiallisen kameran lisäksi virtuaalisia kameroita, joiden avulla voidaan pelin aikana hallita kameran käyttäytymistä. Kameraa varten on tehty erillinen objekti *CameraFollowObject*, jota se seuraa. Useimmissa huoneissa käytössä on kamera, joka keskittää pelaajahahmon ruudun keskelle ja seuraa sitä.

Pystysuunnassa ahtaita käytäviä varten on kamera, joka seuraa pelaajaa vain vaakasuunnassa. Tämän kameran ollessa käytössä CameraFollowObject pysyy lukittuna pystysuunnassa pelaajahahmon korkeudella. Tiettyjä huoneita, kuten päävastus- tai salahuoneita varten on oma virtuaalinen kamera, joka keskittää kuvan huoneen keskelle seuraamatta pelaajahahmoa. Tällaisiin huoneisiin on lisätty objekti, jonka kohdalle CameraFollowObject siirretään. Kenttiin on asetettu kameran hallintatriggereitä (Kuva 22), joiden avulla voidaan helposti määrittää, mikä kamera aktivoidaan pelaajan liikkuessa sen kohdalle. Triggeriin voidaan asettaa eri kamera riippuen siitä, kummasta suunnasta pelaaja siihen kävelee.



Kuva 22 Kameran hallintatriggerin skripti Unityn käyttöliittymässä.

Kameran hallintatriggerissä on myös toinen käyttötapa, jonka avulla voidaan siirtää kameran keskitystä, kun pelaaja on tietyssä paikassa. Tällä voidaan esimerkiksi vihjata pelaajalle kiertotiestä esteen ohittamiseen, näyttää, että kielekkeeltä on turvallista hypätä alas, tai näyttää pelaajalle jotain tärkeää, mikä on sillä hetkellä saavuttamattomissa.

Pelaajahahmolle on tehty modernin metroidvania-pelin taistelumekaniikat. Se voi lyödä miekalla eteen, ylös ja ilmassa ollessa alaspäin. *(Kuva 23)* Kaikki lyönnin alueella olevat viholliset menettävät kestopisteitä pelaajan nykyisen lyöntivoiman perusteella. Miekkalyönnissä on tiettyjä rajoituksia. Sen aikana ei voi kääntyä, ja maassa ollessa pelaaja pysähtyy lyönnin ajaksi. Hyökkäyspainiketta painaessa lyönti menee ajastimelle, jonka aikana pelaaja ei voi lyödä. Näin pelaaja ei voi hyökätä uudestaan liian nopeasti. Jos pelaaja osuu viholliseen alaspäin lyönnillä, pelaaja hyppää osumasta ylöspäin. Tällä estetään pelaajan osuminen viholliseen silloin, kun pelaaja on lyömässä vihollista sen yläpuolelta.

```
1 reference
public void Attack() {
    if (attackCooldownTimer <= 0f) {
        //dealing damage is started and stopped with animation triggers
        animator.SetBool("isAttacking", true);
        if (moveInputY <= -0.5f && Grounded() == false) {
            animator.Play("LongswordDown");
        }
        else if (moveInputY >= 0.5f && Grounded() == false) {
            animator.Play("LongswordUp"); //mid air upward slash
        }
        else if (moveInputY >= 0.5f && Grounded() == true) {
            animator.Play("LongswordUp2"); //upward slash while grounded
        }
        else {
            if (Grounded() == false) {
                animator.Play("LongswordJump");
            }
            else {
                animator.Play("Longsword");
            }
        }
        attackCooldownTimer = attackRate;
    }
}
```

Kuva 23 Pelaajan hyökkäysfunktio miekalla.

Pelaaja menettää kestopisteitä osuessaan vihollisiin tai niiden hyökkäyksiin. Kun pelaajahahmo vahingoittuu, se lentää vastakkaiseen suuntaan vahingon lähteestä ja pelaaja menettää kontrollin hahmosta hetkellisesti. Tämän aikana pelaaja on immuuni kaikelle vahingolle ja ei voi menettää kestopisteitä. Jos pelaajan kestopisteet loppuvat, pelaaja kuolee ja joutuu takaisin edelliselle tallennuspisteelle.

Miekan lisäksi pelaajalle on tarkoitus tehdä taikoja ja muita työkaluja, joiden avulla voidaan vaikuttaa vihollisiin kauempaa. Taikoja varten pelaajalla on taikamittari, joka vähenee niitä käytettäessä. Jos taikamittari on tyhjä, pelaajan pitää täyttää se käymällä tallennuspisteellä tai käyttämällä esineitä.

4.3 Pelisisällön luonti

4.3.1 Viholliset

Kaikkia vihollisten perustoiminnallisuuksia varten on tehty Enemy-skripti *(Kuva 24)*. Tämä skripti sisältää muuttujat ja funktiot kaikille ominaisuuksille, jotka ovat samoja vihollistyypistä riippumatta. Uniikkien vihollisten omat skriptit perivät Enemy- skriptistä nämä ominaisuudet. Tämä käytäntö auttaa välttämään toistuvaa koodia eri skripteissä.

```
public class Enemy : MonoBehaviour {

    public float health = 50f;
    public float damage = 10f;
    public bool hasTakenDamage;
    [SerializeField]
    private SpriteRenderer spriteRenderer;
    private Color originalColor;

    Unity Message | 0 references
    void Awake() {
        spriteRenderer = GetComponent<SpriteRenderer>();
        originalColor = spriteRenderer.color;
    }

    5 references
    public virtual void TakeDamage(float damage) {
        hasTakenDamage = true;
        health -= damage;
        Debug.Log(health);
        StartCoroutine(FlashRed());
        if (health <= 0) {
            Debug.Log("enemy dead");
            Die();
        }
    }
```

Kuva 24 Esimerkki kaikille vihollisille yhteisestä ominaisuudesta

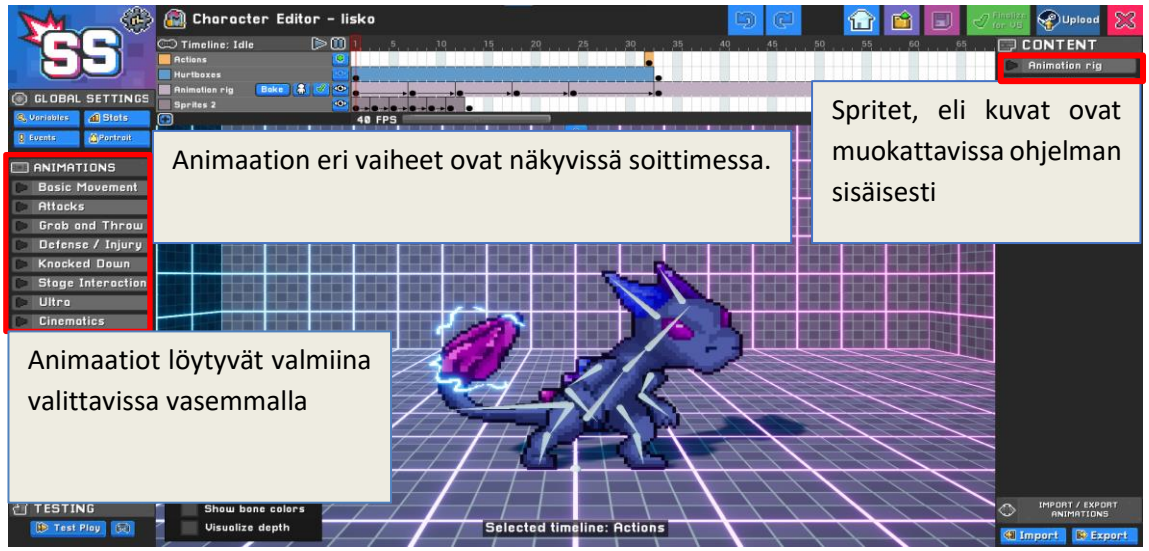
Ensimmäinen pelaajan kohtaama vihollinen on paikallaan oleva sieni (*Kuva 25*), joka ei aktiivisesti hyökkää pelaajaa kohtaan. Tällä vihollisella vain yksi hyökkäys. Pelaajan kävellessä tarpeeksi lähelle, se päästää ympärilleen myrkkykaasua, joka vahingoittaa pelaajaa (*Kuva 26*).

Kuva 25 Passiivinen sieniviholliinen, joka sulautuu ympäristöön.

```
1 reference
private void Attack() {

    Instantiate(poisonMist, transform.position, Quaternion.identity);

    Collider2D player = Physics2D.OverlapCircle(transform.position, attackRadius, LayerMask.NameToLayer("Player"));
    if (player != null) {
        Player.instance.TakeDamage(damage, transform);
    }
    attackTimer = attackCooldown;
}
```

Kuva 26 Sieniviholliisen hyökkäysfunktio

Smack Studiolla teimme tutoriaalikentässä näkyvän kristalliliskon. Animoiminen oli automatisoitu, mutta muokkasimme sitä pelissä näkyvien animaatioiden tyyliseksi yksinkertaistamalla aikajanaa. Käytimme valmista ja animoitua mallia, johon teimme oman designin (*Kuva 27*). Tämä vihollinen tarvitsi 4 eri animaatiota: paikallaan seisominen, hyökkäys, kävely ja kuoleminen. Piirtämistä vaati 5 eri kuvaa, jonka sitten Smack Studion animointityökalu viimeisteli valmiiksi animaatioksi.



Kuva 27 Smack Studion editointinäkymä.

Kristalliliskolle asetetaan kenttään kaksi pistettä, joiden väliä se kulkee edestakaisin. Jos pelaaja kävelee tarpeeksi lähelle, se lähtee jahtaamaan pelaajaa. Pelaajan saavuttaessa se yrittää lyödä pelaajaa. Nämä toiminnallisuudet on toteutettu määrittämällä liskolle eri tilat (Kuva 28) ja vaihtamalla niitä tarpeen mukaan.

```
switch (currentState) {
    case EnemyState.Idle:
        currentState = EnemyState.Patrolling;
        break;
    case EnemyState.Patrolling:
        Patrol();
        break;
    case EnemyState.Attacking:
        AttackPlayer();
        break;
    case EnemyState.Chasing:
        ChasePlayer();
        break;
}
```

Kuva 28 Kristalliliskon eri tilat koodissa.

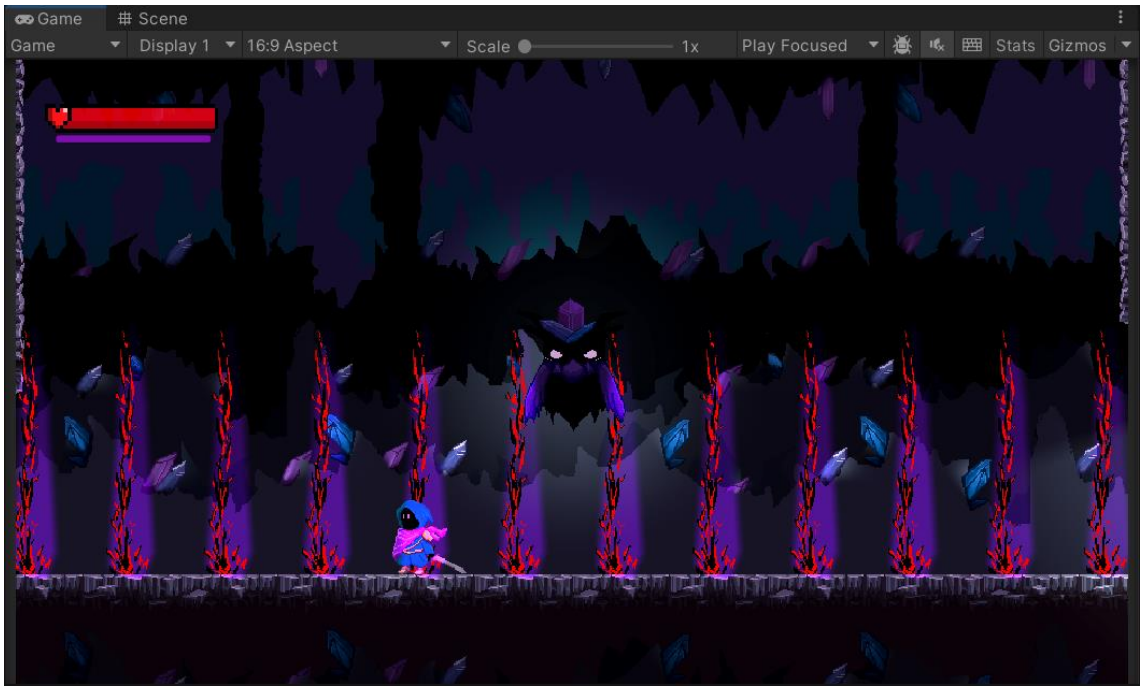
Demon päävihollisella on tehty oma huone. Kun pelaaja kävelee huoneessa olevan triggerin läpi, pelaaja lukitaan huoneeseen ja päävihollinen ilmestyy. Päävihollisella on kolme ajastimella tehtävää hyökkäystä, jotka ovat monimutkaisempia kuin tavallisten vihollisten hyökkäykset. Tässä vaiheessa projektia koodissa arvotaan numero yhden ja kolmen väliltä ja valitaan hyökkäys sen perusteella. Hyökkäyksille on tarkoitus tehdä jatkossa erilaisia ehtoja, kuten tietyn hyökkäyksen valitseminen tietyltä etäisyydeltä. Päävihollinen ei myöskään tee vielä mitään väistöliikkeitä.

Ensimmäinen hyökkäys tiputtaa viisi kristallia katosta sekunnin välein suoraan pelaajan yläpuolelta riippumatta siitä, missä pelaaja sillä hetkellä on (*Kuva 29*). Kristalleissa on rigidbody2D- komponentti, jonka avulla painovoima tiputtaa ne ruudun yläreunasta alas. Collider2D- komponentin avulla tarkistetaan mahdolliset osumat pelaajaan ja aiheutetaan vahinkoa pelaajan elämäpisteisiin. Kristallit tippuvat suoraan alas, joten pelaajan tarvitsee vain liikkua sivusuunnassa hyökkäyksen keston ajan väistääkseen ne.

```
public void FallingCrystals() {
    Vector2 crystalSpawnPos = new Vector2(Player.instance.transform.position.x, BossArena.instance.arenaBoundaries.bounds.max.y - 1);
    if (crystalPrefab != null) {
        Instantiate(crystalPrefab, crystalSpawnPos, Quaternion.identity);
    }
}

1 reference
private IEnumerator CrystalAttack() {
    attackTimer = attackCooldown;

    for (int i = 0; i < fallingCrystalsAmount; i++) {
        FallingCrystals();
        animator.Play("idle_1");
        isAttacking = false;

        yield return new WaitForSeconds(projectileInterval);
    }
}
```

Kuva 29 Tippuvien kristallien ilmestyminen pelaajan yläpuolelle

Toinen hyökkäys aiheuttaa salamaniskuja tasaisin välimatkoin koko huoneen leveydeltä. Tämän hyökkäyksen animaatio on eritelty kahteen osaan. Animaation ensimmäinen osa on indikaatio siitä, mihin salamat ovat iskemässä ja ei vielä aiheuta vahinkoa, vaikka pelaaja olisi jonkin niistä kohdalla. Ensimmäinen osa kestää sekunnin, jonka jälkeen pelaaja vahingoittuu, mikäli on salamaniskun kohdalla animaation toisen puoliskon aikana (*Kuva 30*). Salamat lyövät kahdesti peräkkäin eri kohtiin (*Kuva 31*), joten pelaajan on väistettävä ne liikkumalla tarkasti sivusuunnassa.



Kuva 30 Päävihollisen salamaniskuhyökkäys

```
1 reference
private IEnumerator LightningAttack() {
    if (lightningPrefab != null) {
        attackTimer = attackCooldown;

        GameObject lightning = Instantiate(lightningPrefab, BossArena.instance.transform.position, Quaternion.identity);

        yield return new WaitForSeconds(1);

        GameObject lightning2 = Instantiate(lightningPrefab, new Vector2(BossArena.instance.transform.position.x + lightningOffset,
          BossArena.instance.transform.position.y), Quaternion.identity);

        Destroy(lightning);

        yield return new WaitForSeconds(1);

        Destroy(lightning2);
        animator.Play("idle_1");
        isAttacking = false;
    }
}
```

Kuva 31 Salamaniskujen spawnaaminen peräkkäin eri kohtiin

Kolmannessa hyökkäyksessä päävihollinen ilmestyy huoneen toiseen reunaan riippuen siitä, kummalla puolella huonetta pelaaja on keskikohdasta katsottuna. Se ilmestyy aina lattian tasolle huoneen vastakkaiselle puolelle pelaajasta katsottuna (*Kuva 32*) ja lähtee syöksymään pelaajaa päin. Jos se osuu pelaajaan syöksyn aikana, pelaaja menettää elämäpisteitä. Kun päävihollinen saavuttaa huoneen toisen päädyn, se ilmestyy takaisin huoneen keskelle ilmaan. Tämän hyökkäyksen voi väistää hyppäämällä sen yli syöksyn aikana.

```
private IEnumerator TeleportToDash() {
    Vector2 center = BossArena.instance.transform.position;
    Vector2 playerPos = Player.instance.transform.position;
    collFloating.enabled = false;
    spriteRenderer.enabled = false;
    spotLight.enabled = false;
    Instantiate(teleportParticles, transform.position, Quaternion.identity);
    yield return new WaitForSeconds(1f);
    if (playerPos.x <= center.x) {
        transform.position = new Vector2(BossArena.instance.arenaBoundaries.bounds.max.x, BossArena.instance.spawnPoint.transform.position.y);
        Instantiate(teleportParticles, transform.position, Quaternion.identity);
        Dash();
    }
    if (playerPos.x > center.x) {
        transform.position = new Vector2(BossArena.instance.arenaBoundaries.bounds.min.x, BossArena.instance.spawnPoint.transform.position.y);
        Instantiate(teleportParticles, transform.position, Quaternion.identity);
        Dash();
    }
}
```

Kuva 32 Tällä funktiolla tarkistetaan, kummalle puolelle huonetta päävihollinen ilmestyy ennen syöksyä.

4.3.2 Kenttädesign

Pelin kenttädesign on toteutettu Metroidvania-peligenren tyylisesti, ja se koostuu useista eri alueista, joista jokainen on jaettu pienempiin huoneisiin. Ensimmäistä huonetta lukuun ottamatta jokaisesta huoneesta löytyy tallennuspiste, jossa pelaaja voi levätä ja tallentaa edistymisensä. Pelaajan kuollessa he siirtyvät takaisin viimeiselle tallennuspisteelle, mistä peliä voi jatkaa ilman, että täytyisi aloittaa alusta joka kerta kuoleman jälkeen.

Demoversiota varten loimme luolakentän, joka sisältää kaksi huonetta. Ensimmäinen huone toimii tutoriaalikenttänä, jossa pelaajalle opetetaan pelin perustoiminnallisuudet, kuten liikkuminen ja taistelumekaniikat. Ensimmäisessä huoneessa on muutamia helppoja vihollisia, jotka pelaajan täytyy päihittää ennen siirtymistä seuraavaan huoneeseen.

Seuraava huone on luolakentän viimeinen, ja sieltä löytyy pelin ensimmäinen päävihollinen. Päävihollisen päihittämisen jälkeen pelaaja voi siirtyä seuraavaan kenttään, jonka on tarkoitus olla rauhallinen kylä luolan ulkopuolella. Siellä pelaaja voi ostaa uusia varusteita ja päivittää hahmoaan. Tämä kyläkenttä ei kuitenkaan sisälly demoversioon.

5 POHDINTA

Tässä opinnäytetyössä käsittelimme pelin suunnittelua ja kehitystä, keskittyen nostalgisen pikseligrafiikalla toteutetun tasohyppelypelin luomiseen. Tavoitteenamme oli suunnitella peli, josta pelaaja voisi nauttia ajattomasti. Valitsimme tasohyppelygenren, koska se tarjoaa suoraviivaisen ja selkeän lähestymistavan pelinkehityksen perusteiden oppimiseen ja toteutukseen.

Ryhmässä keskityimme selkeään kommunikaatioon ja yhteistyöhön, joka mahdollisti projektin sujuvan etenemisen. Tapasimme kasvotusten kerran viikossa ja työskentelimme muuten etänä Discordia hyödyntäen. Kirjasimme kaikki ideat ylös Discord-kanavalle ja kävimme läpi realistiset tavoitteet yhdessä. Keskityimme omiin vastuualueisiimme, jotka sovittiin projektin alkuvaiheessa jokaisen vahvuudet huomioon ottaen.

Pelin suunnitteluvaiheessa päätimme käyttää vanhan projektin GDD:tä, jonka pohjalta aloimme työstämään tämän projektin tavoitteita. Päätimme silloin demon sisällön laajuuden, joka tulisi olemaan tutoriaalikenttä ja päävihollisen huone. Päävalikko on peräisin edellisestä projektista. Pelihahmon design pysyisi enimmäkseen samanlaisena, mutta sen ulkomuotoa päivitettäisiin. Viholliset ja kenttädesign tehtäisiin kokonaan uudelleen. Vaikka vanha GDD auttoi kehitysprosessissa, olisi uuden tekeminen ennen varsinaisen pelikehityksen aloittamista nopeuttanut työntekoa entisestään. Esimerkiksi vihollisten ja kentän tarkempi suunnittelu olisi helpottanut kehitystä. Iterointi on kuitenkin osa kehitysprosessia ja tähän mennessä käyttämättömiä ideoita voidaan hyödyntää myöhemmin jatkokehityksessä.

Opimme myös paljon uutta ohjelmointiin liittyvää demoa tehdessä. Peliohjelmoijana toimivan projektipohjan hyödyntäminen uutta peliprojektia luodessa nopeuttaa merkittävästi projektin aloitusvaihetta. Projektipohjat sisältävät usein pelinkehityksen kannalta keskeisiä perustoiminnallisuuksia, kuten hahmon liikkumisen, pelinäkymien hallinnan tai käyttöliittymän perustan, mikä säästää ohjelmoijalta aikaa ja vaivaa. Valmiin projektipohjan avulla pelin perustoiminnallisuuksien lisääminen on huomattavasti helpompaa kuin niiden luominen alusta asti tyhjään projektiin. Lisäksi projektipohjaa voidaan laajentaa ja mukauttaa uuden projektin tarpeisiin lisäämällä siihen uusia toiminnallisuuksia ja ominaisuuksia. Esimerkiksi mukautettavuus peligenren vaatimuksiin, kuten Metroidvanian monimutkaiseen tasodesigniin tai roguelike-elementteihin, tekee projektipohjasta joustavan työkalun. Projektipohjan käyttäminen antaa kehitystiimille mahdollisuuden keskittyä suoraan pelin ainutlaatuisiin piirteisiin ja mekaniikkoihin sen sijaan, että aikaa kuluisi perustoimintojen toteuttamiseen.

Opimme toteuttamaan monet genrelle ominaisista asioista paremmin kuin vanhassa projektissa. Kaikki pelaajan toiminnallisuudesta vihollisten käyttäytymiseen on suunniteltu alusta asti jatkokehitystä ajatellen. Myöhemmin pelaajalle on helppo lisätä uusia toiminnallisuuksia ilman, että se vaikuttaa mihinkään olemassa olevaan. Myös uusia vihollisia on helppo tehdä jo olemassa olevien pohjalta. Toteutettu kamerasysteemi on myös varustettu useammilla ominaisuuksilla kuin vanhassa projektissa. Pelin mekaniikoissa on kuitenkin vielä parantamisen varaa. Esimerkiksi pelaajan lyöntien osumatunnistusta on tarkoitus parantaa osana jatkokehitystä. Kenttäsuunnittelu yksi osa-alue, joka vaatii yllättävän paljon aikaa ja kokeilua. Aiheen opiskelun ja kokeilun kautta tuli selväksi kuinka tärkeää on lisätä ympäristöön elementtejä, jotka kannustavat pelaajaa tutkimaan. Reittien tulee olla myös tarpeeksi selkeitä, että pelaaja tietää oikean etenemissuunnan.

Artistin näkökulmasta työskentely hahmoanimaatioiden parissa oli erityisen opettavaista. Pikselitaiteessa erityisen tärkeää oli pitää hahmojen pikselimäärä mahdollisimman samankaltaisena, jotta grafiikka olisi mahdollisimman yhtenäistä. Edellisessä projektissa kahden eri artistin työskentely aiheutti eroja grafiikan

yhteneväisyydessä, joihin kiinnitettiin huomiota vasta projektin valmistumisen jälkeen. Tässä projektissa pidimme huolta, että pikselimäärät olivat aina suhteutettuna pelaajaan. Staattisen värimaailman luominen auttoi myös pitämään miljöön ja hahmot visuaalisesti yhtenäisinä.

Visuaaliset valinnat, kuten värimaailma ja hahmojen yksinkertaiset animaatiot, olivat tietoisia päätöksiä pelin tunnelman ja pelaajan kokemuksen tukemiseksi. Pikseligrafiikan käytön päätavoite oli luoda ajaton ja tunnelmallinen peliympäristö, jossa visuaalinen tyyli tukee pelin mekanismeja. Tämä yksinkertaisuus mahdollisti myös pelaajan keskittymisen pelin toimintaan sen sijaan, että hän olisi häiriintynyt liiallisista yksityiskohdista.

Animaatioiden tekeminen oli myös nyt paljon nopeampaa edellisen projektin kokemuksen myötä. Kokeilin ensimmäistä kertaa myös animaatioiden automatisointia. Opin yksinkertaistamaan aikajanaa, mutta huomasin, että automatisointi ei aina tuottanut toivottuja tuloksia.

Pikselitaiteen historia oli mielenkiintoinen ja opettavainen. Artistina on tärkeää ymmärtää, mistä tietyt termit, työskentelytavat ja tiedostotyyppien suosio juontavat juurensa. Aiheeseen perehtyminen auttoi sisäistämään omat ajatuksemme pelin keskeisistä elementeistä ja siitä, mitä haemme sen visuaalisesta kokonaisuudesta.

Tämä projekti on antanut minulle uutta ymmärrystä animaatioiden automatisoinnista ja pelin visuaalisesta suunnittelusta. Vaikka automaation käyttö ei aina tuottanut toivottuja tuloksia, opin tärkeitä asioita aikajanan optimoinnista ja sen vaikutuksesta animaatioiden sujuvuuteen. Tulevaisuudessa aion panostaa enemmän automaation ja käsin piirretyn animaation tasapainottamiseen, jotta voin saavuttaa nopeampia tuotantoprosesseja ilman, että visuaalinen laatu kärsii.

Pelin visuaalinen ilme ja animaatiot eivät ole vain ulkokuorta, vaan ne vaikuttavat suoraan siihen, miten pelaaja kokee pelimaailman. Tunnelman luominen pikseligrafiikan avulla mahdollisti sen, että peli voi olla yhtä aikaa nostalginen ja moderni. Tulevaisuudessa aion syventää tätä pelikokemusta entistä dynaamisemmilla animaatioilla ja ympäristöjen interaktiivisuudella, jotta pelaaja voisi kokea maailmamme entistä elävämmäksi.

Pelin jatkokehitykseen on suunnitteilla paljon lisää ominaisuuksia ja sisältöä. Kaikki tämä on tarkoitus dokumentoida yksityiskohtaisesti uudessa GDD:ssä. Tarinaa ei ole juuri mietitty tähän mennessä ja se on tarkoitus kirjoittaa jatkokehityksen alkupuolella. Tarinan olemassaolo selkeyttää ja nopeuttaa kehitysprosessia. Tarinan pohjalta voidaan piirtää konseptitaidetta, jonka pohjalta taas on helpompi alkaa suunnitella kenttiä, hahmoja ja vihollisia. Demossa on vain yksi pelattava alue, jossa on yksi kenttä. Valmiissa pelissä maailma on tarkoitus jakaa useaan alueeseen, joissa kaikissa on useampia kenttiä. Jokaisen alueen loppuun on suunnitteilla oma uniikki päävihollinen. Pelaajahahmolle on suunnitteilla lisää aseita, taikoja ja kykyjä. Näitä varten tarvitaan myös tavaraluettelo pelin käyttöliittymään. Tavaraluetteloa voitaisiin käyttää myös muita kerättäviä esineitä varten.

LÄHTEET

Järvinen, N. & Torkkel, J.-M. 2019. Avaa silmäsi pelimoottoreiden hyötykäytölle. HAMK Unlimited Professional 1.4.2019. Hakupäivä 10.9.2024.
https://unlimited.hamk.fi/teknologia-ja-liikenne/pelimoottorit-hyotykaytto/.

Unity documentation 2024. Docs and guides to work with the Unity ecosystem 2024. Hakupäivä 10.9.2024.
https://docs.unity.com/.

The Codest 2024. Versionhallinta. Hakupäivä 5.11.2024.
https://thecodest.co/fi/sanakirja/versionhallinta/.

Trewhitt, Max 2023. The beauty of Pixel Art and why it's every bit as relevant today 22.3.2023. Hakupäivä 19.9.2024.
https://www.flaticon.com/blog/the-beauty-of-pixel/.

Monserrate, Emi 2024. Learn how to make Pixel art: Tutorial with tips & tools. Hakupäivä 19.9.2024.
https://www.adobe.com/creativecloud/design/discover/pixel-art.html.

Adobe 2024. What are PNG files and how do you open them? Hakupäivä 20.9.2024
https://www.adobe.com/creativecloud/file-types/image/raster/png-file.html.

Redacción Tokio 2023. What is Pixel art? 13.02.2023. Hakupäivä 19.9.2024.
https://www.tokioschool.com/en/news/what-is-pixel-art/.

GIMP's Team 2024. About GIMP. Hakupäivä 20.9.2024.
https://www.gimp.org/about/introduction.html.

Bitmob 2010. Metroidvania: Super Metroid and the Definition of a Genre 24.4.2010. Hakupäivä 23.10.2024.
https://venturebeat.com/games/metroidvania-super-metroid-and-the-definition-of-a-genre/.

Salomäki, Timo 2019. IT-sanasto. Hakupäivä 19.07.2024.
https://github.com/TimoSalomaki/IT-sanasto/blob/master/ohjelmointisanasto.md.

ThirdPixel Interactive,2024. SmackStudio Press. Hakupäivä 25.11.2024.
https://smackstudio.com/press/

Krita Manual 2024. Hakupäivä 12.12.2024.
https://docs.krita.org/en/index.html