Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TEEMU SUOJA

Mobiilipelin kehittäminen

TIETOJENKÄSITTELYN TUTKINTO-OHJELMA
2024

TIIVISTELMÄ

Suoja, Teemu: Mobiilipelin kehittäminen
Opinnäytetyö, AMK
Tietojenkäsittelyn tutkinto-ohjelma
Marraskuu 2024
Sivumäärä: 32

Tämä työ tehtiin osoittamaan, että yksittäinen ohjelmistokehittäjä voi luoda mobiilipelin alusta julkaisuun saakka. Opinnäytetyössä käsitellään yksittäiselle kehittäjälle tärkeitä osa-alueita, kuten projektin suunnittelu, toteutus, julkaisu sekä käytännön projektiesimerkki.

Ensimmäisessä osassa tarkastellaan mitä on Indie-pelikehitys, näkemyksiä millaista kehittäminen on ja löytyykö siitä minkälaisia riskejä ja hyötyjä.

Seuraavassa vaiheessa käsitellään projektiin liittyviä teknologioita ja suunnittelua. Miksi projektissa päädyttiin Unityn käyttämiseen, mitä kehittäjän on syytä ottaa huomioon projektia suunnitellessa, markkinoinnin roolia ja versiohallinnan tärkeyttä.

Lopuksi käydään läpi projektia Android-pelin kehitysprosessi ja sen julkaisu Google Play -alustalle.

Avainsanat: peliohjelmointi, pelisuunnittelu, markkinointi

ABSTRACT

Suoja, Teemu: Mobile Game Development
Bachelor's thesis
Degree Programme in Business Information Systems
November 2024
Number of pages: 32

This work was undertaken to demonstrate that an individual software developer can create a mobile game from start to finish. The thesis covers key aspects relevant to a solo developer, such as project planning, implementation, publishing, and a practical project example.

The first section examines what indie game development is, perspectives on what the development process is like, and the potential risks and benefits involved.

The next phase covers technologies and planning related to the project: why Unity was chosen for the project, what the developer should consider during the planning stage, the role of marketing, and the importance of version control.

Finally, the thesis reviews the Android game development process and its publication on the Google Play platform.

Keywords: game programming, game design, marketing

SISÄLLYS

1 JOHDANTO

Nykypäivän suurin viihdealan jätti ovat pelit. Niiden tarjonta on kasvanut hurjasti ja monipuolistunut merkittävästi. Pelit on luotu viihdyttämään vapaa-ajalla, ja osa on suunniteltu kilpailulliseksi esports-urheiluksi. Pelaamisen suosio ei ole mikään ihme, sillä pelaamisen aloittaminen on todella helppoa, jos käyttäjällä on käytössään mikä tahansa älylaite, tietokone tai konsoli. Lisäksi muiden intohimo pelien harrastamiseen houkuttelee uusia kokeilijoita mukaan. Tämä laaja markkina avaa ovet ohjelmistokehittäjille, joita kiinnostaa pelikehityksen ala.

Tämä opinnäytetyö käsittelee pelikehitystä yksittäisen ohjelmistokehittäjän näkökulmasta, eli indie-kehittäjän näkökulmasta. Tulevissa osioissa käsitellään, mitä indie-kehitys on, miten indie-kehityksen voi aloittaa ja sopiiko se juuri sinulle. Tavoitteena on kertoa yleisesti ja yksityiskohtaisemmin indie-pelikehityksestä ja siitä, mitä asioita kannattaa ottaa huomioon, kun ryhdytään tekemään peliprojekteja. Käymme läpi markkinoinnin merkitystä projektin menestymisen kannalta ja eri tapoja markkinoida. Työ tarjoaa myös näkökulmaa yrittäjähenkiselle henkilölle sekä siitä, millaista on tehdä indie-kehitystä yksin tai mikroyrityksessä. Työssä käytetään Unity-alustaa ja sen ympärille tarvittavia ohjelmistoja. Työn viimeisessä osiossa käsitellään luomaani 2D-pelikehitysprojektia ja sen julkaisua Google Play -kauppaan.

2 INDIE-PELIKEHITYS

2.1 Mikä on Indie-kehitys?

Indie-kehitys tarkoittaa ohjelmistoprojekteja, joita kehittävät itsenäiset ohjelmistokehittäjät. Tällä viitataan kehittäjiin, jotka eivät työskentele suurissa tunnetuissa yrityksissä, kuten EA tai Ubisoftilla, vaan toimivat itsenäisesti omalla toiminimellään, vapaa-ajallaan tai omassa mikroyrityksessään. Indie-kehittäjät ovat siis yksityisyrittäjiä tai pieniä aloittavia yrityksiä pelikehitysalalla. Tyypillisesti indie-studiot koostuvat muutamasta kehittäjästä, eikä henkilöstömäärä yleensä ylitä kymmentä asiantuntijaa. Indie-studiot eivät myöskään käytä julkaisijoita omassa työssään. (Mozolevskaya, 2021)

2.2 Onko sinusta Indie-kehittäjäksi?

Itsenäinen kehitystyö on raskasta, mutta se on erittäin vapaata kuin olisit yrittäjä. Sinun ei kuitenkaan tarvitse luopua aiemmasta työstäsi, sillä indie-kehitystä voi tehdä omalla vapaa-ajalla harrastuksena, kunnes kiinnostus mahdollisesti kasvaa. Vaikka kaikki taidot eivät riittäisi koko projektin toteuttamiseen, indie-kehitystä voi ajatella tilaisuutena oppia uutta alalta. Esimerkiksi, jos hallitset hyvin ohjelmointikielen ja pelimoottorin käytön, mutta laadukkaan visuaalisen sisällön tuottaminen ei vielä onnistu, voit opetella sitä ja kokeilla, onko se mieleistäsi. Projektin suunnittelussa on kuitenkin tärkeää ottaa huomioon pienet tavoitteet ja projektin mittakaava. Itsenäinen kehitys vaatii kykyä olla omatoiminen ja asettaa itselle aikatauluja. Kun nämä perusasiat hallitsee ja saa ensimmäisen projektin valmiiksi, on jo oppinut, miten viedä projekteja loppuun.

Itsetuntemus on tärkeää yksittäiskehittäjän roolissa. On hyvä kartoittaa omat taipumukset ja niiden vaikutus omaan työhön. Jos et pidä tietystä työtehtävästä, on hyvä yhdistää sen rinnalle jotain mieluisampaa tekemistä. Pakolliset asiat on kuitenkin hoidettava omista mieltymyksistä riippumatta. (Robertson, 2022)

2.3 Indie-kehityksen vahvuudet ja heikkoudet

Kuva 1. Kevuru Games -artikkelin kuva indie-kehittäjän eduista. (Mozolevskaya 2021.)

Indie-yrityksillä on täysi vapaus kokeilla epätavallisia ideoita, jotka eivät markkinatasolla ole vielä tunnettuja tai suosittuja. Tämä tekee indie-yrityksistä johtavan kokeilualustan uusille, trendaaville peli-ideoille verrattuna suuriin peliyhtiöihin, jotka keskittyvät tarkkaan suunnitteluun varmistaakseen yritykselle kannattavia ja tuottoisia ratkaisuja. (Mozolevskaya, 2021)

Indie-yritykset koostuvat yleensä pienestä tiimistä, ja heillä on suora yhteys palautteeseen. Suurissa yrityksissä palaute kulkee usein asiakastuen kautta, jolloin se ei aina päädy suoraan kehitysryhmälle. Indie-kehittäjät voivat usein korjata virheitä ja bugeja muutamissa tunneissa ilmoituksen jälkeen, ja pienet yritykset julkaisevatkin yleensä sisältöä nopeammin kuin suuret yhtiöt. (Mozolevskaya, 2021)

Vaikka indie-yrityksillä ei välttämättä ole resursseja tehdä jokaisesta osa-alueesta huippuluokkaa, ne voivat silti loistaa jollakin erityisalueella, kuten visuaalisessa toteutuksessa, ainutlaatuisessa pelimekaniikassa tai tarinankerronnassa. (Mozolevskaya, 2021)

Indie-yritystä ei voi pitää itsenäisenä, jos projekteille asetetaan julkisia määräaikoja; kehitys tapahtuu indie-kehittäjän omassa tahdissa. Kehitykseen kuluva aika on täysin indie-kehittäjän päätettävissä. (Mozolevskaya, 2021)

Indie-yritykset eivät jää suurten julkaisijoiden varjoon, vaan ovat usein trendien luojia. He voivat vapaasti puhua omista projekteistaan, ja parhaimmillaan heidän luomansa uudet ideat voivat saada valtakunnallista huomiota. Esimerkiksi pieni tanskalainen yritys yhdisti tasohyppely- ja pulmapelin, mistä syntyi tunnettu Limbo-peli. (Mozolevskaya, 2021)

Kuva 2. Kevuru Games artikkelin kuva indie-kehittäjän heikkouksista. (Mozolevs-kaya 2021.)

Indie-yritykset kärsivät usein osaamisen puutteesta, mikä voi vaikuttaa pelin laatuun. Tällöin pelaajat saattavat turhautua, jos jokin pelin osa-alue on tehty epäammattimaisesti, mikä voi pilata pelikokemuksen. (Mozolevskaya, 2021)

Indie-kehittäjät kattavat kaikki projektin kustannukset itse, usein omilla henkilökohtaisilla varoillaan tai joukkorahoituksen avulla. Budjetin rajoitukset vaikuttavat huomattavasti pelin lopulliseen julkaisuun. (Mozolevskaya, 2021)

Muutamien indie-pelien menestys on motivoinut muita kehittäjiä julkaisemaan omia versioitaan samankaltaisista peleistä, mikä on johtanut markkinoilla ylikylläisyyteen. (Mozolevskaya, 2021)

Heikko markkinointi voi johtua rahoituksen puutteesta, mutta markkinointia voi toteuttaa myös ilman suuria kuluja. Kuitenkin markkinoinnin sivuuttaminen on suuri virhe, jos tavoitteena on projektin menestys. On tavallista, että markkinointi aloitetaan liian myöhään. Kannattaa muistaa, että projektille on vain yksi julkaisuajankohta, ja tavoitteena on saada mahdollisimman moni tietoiseksi pelin olemassaolosta. (Mozolevskaya, 2021)

Projektin hallinnassa ilmenee aina haasteita, olivatpa ne yksilön tai tiimin tasolla. Jos projektin tarkoituksena on saada tuote markkinoille, aikataulut, työvaiheet ja työmäärät on suunniteltava tarkasti. Hyvä projektinhallinta on avain valmistumiseen ja projektin sujuvuuteen. Valitettavasti kaikilla ei ole resursseja palkata projektisuunnittelijaa, mutta se on tärkeää ottaa huomioon, sillä ilman hallintaa projekti voi päättyä nopeasti. (Mozolevskaya, 2021)

3 PELIMOOTTORI JA KÄYTETTY TEKNIIKKA

3.1 Pelimoottorien monipuolinen tarjonta

Pelimoottoreita löytyy vaikkapa mitä, Gore, GameMaker, Unreal, Godot, CryEngine, Lumberyard, Unity, jne. Voidaan todeta, että pelimoottoreista ei ole pulaa, vaan suurin kysymys on valinnassa ja siinä, mikä niistä sopii parhaiten käyttäjälle. Monet pelimoottorit tarjoavat kattavat hienosäädöt yksilöllisiin tarpeisiin ja kilpailevat asiakaskunnasta. Toiset kuten Cocos2d ja GameMaker keskittyvät 2D-kehitykseen ja Amazon Lumberyard on suunnattu 3D-kehitykseen. Erilaisiin pelimoottoreihin kannattaakin tutustua ja arvioida, mikä niistä vastaa parhaiten omia tarpeita.

Opinnäytetyön tekemiseen on valittu Unity-pelimoottori, sillä se on todella monipuolinen käytöllään ja mitä ikinä opit Unityssä, sillä voi olla hyötyä muissa projekteissa ja moni yritys käyttää Unityä omissa projekteissaan. Työmarkkinoilla voi olla hyödyllistä osata käyttää Unityä. Projektissa voidaan käyttää Unity Asset Store -kaupasta aiemmin hankittuja lisäosia ja materiaaleja. On hyvä havainnollistaa ideansa paperille, mitä resursseja on käytettävissä ja verrata pelimoottoreita keskenään, mikä niistä toteuttaa peli-idean parhaiten.

3.2 Unity pelimoottorina

Unity on erinomainen pelimoottori, joka tarjoaa valmiudet kunnianhimoiseen pelikehitykseen koko alalla. Pelimoottorin omaaminen tuo mahdollisuuden monipuolisesti eri alustoille: PC, Xbox, PS, Android, IOS ja teknologioille kuten 3D, 2D, AR, VR kehitykselle. Unityn oppiminen ei ole myöskään huonoksi tulevaisuuden kannalta. Jos pelikehitys kiinnostaa, mutta et halua julkaista omia pelejäsi tai ne eivät olleet niin onnistuneita, muut kehittäjät voivat olla kiinnostuneita taidoistasi Unityn käyttämisessä. Tämä voi luoda mahdollisuuksia työllistyä alalla. Unity tarjoaa myös kattavan tuen muille ohjelmistoille ja sen käyttö on täysin ilmaista tiettyyn rajaan asti. Jos Unityllä kehitettyjen ohjelmien tuotot ylittävät 100 000 dollaria, täytyy maksaa osuus Unityn kehittäjille. Vuonna

2024 julkaistiin uudet maksuehdot, jotka koskevat Unityn 2023 LTS -versiota ja uudempia: henkilökohtainen käyttäjä voi tienata jopa 200 000 dollaria 12 kuukauden aikana ennen maksujen alkua, mikä koskee tulevaa Unity 6 -versiota. (Unity, n.d.)

Vaikka urasi ei olisi pelikehittäminen, Unityllä voi tehdä suunnittelu- ja simulointiprojekteja. Ympäristön mahdollisuuden antavat kattavat työkalut niiden tekemiseen. Jos esimerkiksi olet taiteilija ja hallitset 3D-mallinnuksen, voit Blenderin ohella käyttää Unityä asetelmien luomiseen omista malleistasi. Jos kehität tekoälyä voit myös kouluttaa agentteja Unityn ympäristössä suorittamaan haluttuja tehtäviä.

Unity on suosittu pelimoottori, joka tarjoaa käyttäjilleen rikkaan dokumentaation ja tukevan yhteisön projektien ongelmien ratkaisemiseksi. Verkossa on runsaasti Unity-videoita ja -opetusmateriaalia, joiden avulla voi oppia paljon. Unityn sanotaan olevan helppo oppia, sillä sisältöä on paljon saatavilla, mutta oppiminen vaatii oman aikansa.

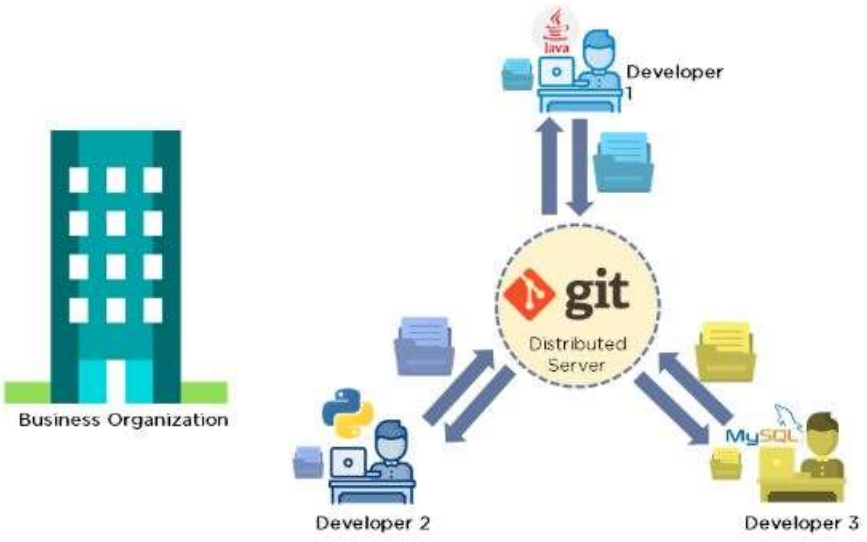
3.3 Git-versiohallinta

Git on ilmainen ja avoimen lähdekoodin versionhallintajärjestelmä, jota käytetään tiedostojen muutosten seuraamiseen. Sitä käytetään yleisesti lähdekoodin hallintaan ohjelmistokehityksessä. Git tarjoaa muun muassa:

- seurannan muutoksille lähdekoodissa
- hajautetun työkalun lähdekoodin hallintaan
- mahdollisuuden useamman kehittäjän työskentelyn yhdessä
- tuen epälineaariselle kehitykselle rinnakkaisten haarojen kautta (Perveez, 2022)

Ennen Gitin käyttöä kehittäjät lähettivät koodinsa keskuspalvelimelle ilman, että heillä oli omia kopioita. Muutokset, jotka lähtivät palvelimelle olivat tuntemattomia muille kehittäjille, ja niiden kanssa kommunikointi oli hankalaa. Gitin saapuessa julkiseen käyttöön se ratkaisi ongelmia. Jokaisella kehittäjällä on

mahdollisuus kopioida koko projektikoodi paikalliseen järjestelmään. Jokaisen kehittäjän tekemiä muutoksia pystytään seuraamaan ja kehittäjien välille on muodostunut säännöllistä viestintää. (Perveez, 2022)



Kuva 3. Git kolmen kehittäjän käytössä. (Perveez 2022)

Nykypäivänä, mitä ikinä teetkään digitaalisessa muodossa, on tärkeää pitää siitä versiota. Unity-projektin tapauksessa tai missä muussakaan ohjelmisto-kehitysprojektissa väitän versiohallinnan pitämisen pakollisena. Se palvelee tekijää ja kaikkia projektin osallisia. Opinnäytetyön projektissa käytämme Git-Hubia pitämään tiedostomme pilvessä. GitHub tarjoaa valmiina .gitignore-tiedoston, joka estää ei-välttämättömien tiedostojen lataamisen versiohallintaan. Arkiston luominen palveluun on yksinkertaista ja käyttöönotto onnistuu helposti esimerkiksi GitHub-työpöytäohjelmiston avulla.

4 PELIKEHITYSPROJEKTIN ALOITTAMINEN

Hyvin suunniteltu projekti on jo puoliksi valmis sovellus. Suunnitteluvaiheessa on hyvä merkitä ylös projektin tavoitteet, luoda aiheesta ajatuskartta ja nostaa esiin tärkeät pääkohdat. Kun tavoitteet on merkitty, on aika hahmotella, miten niitä lähdetään ratkaisemaan. Alussa se ei ole helppoa, eikä välttämättä oikeita ratkaisuja synny heti. Suunnitteluvaihe on osa koko projektityötä ja vie usein enemmän aikaa kuin moni osaa kuvitella. Hyvin suunniteltu projekti voi pelastaa paljon aikaa ja mahdollistaa myöhemmin lisäkehitysmahdollisuuksia.

Voidaan ajatella tilannetta, jossa mietitään mikä kieli tulee käyttöliittymään ja tehdään ratkaisu, että se kirjoitetaan ohjelmakoodiin suomeksi ja huomataan, että ohjelmalla olisi käyttöä myös ulkomailla. Tämä olisi voinut ottaa huomioon suunnitteluvaiheessa, jossa olisi luotu ID-arvot tekstikentille tai luotu valmiiksi lokalisaatiojärjestelmä siltä varalta, jos ohjelmaa paikallistettaisiin muihin maihin.

Suunnittelussa ota huomioon omat vahvuutesi ja mitkä asiat kiinnostavat sinua. Projektit ovat yhtä lailla mahdollisuuksia oppia uutta, mutta se ei tarkoita, että kaikkia pitäisi opetella. Valmiita sisältöjä kannattaa harkita suunnitteluvaiheessa ja kartoittaa, mitä voidaan hyödyntää etukäteen. Jos projekti tarvitsee musiikkia, voiko sitä löytää jostain ja millaisilla oikeuksilla sitä voidaan käyttää? Tai et ole löytänyt vielä sopivaa visuaalista ulkonäköä projektillesi, etkä sitä itse halua tehdä, mistä sellaisen löytäisin? Haluanko siihen panostaa rahalla vai löydänkö ilmaisen? Tai teenkö yksinkertaisen pelin, jossa visuaalit eivät ole lainkaan oleellisia.

Oman aikataulun pitäminen ja pienten tavoitteiden asettaminen antaa motivaatiota työtä tehdessä. Voit asettaa, että viikon aikana teet yhden tavoitteen, kuten toimivat aloitussivun nappulat tai Blenderi-miekka Unityyn. Tavoitteet voivat olla omien mieltymysten mukaan haastavia tai helppoja. Laadittu tavoite toimii ohjeena sille, mitä tulisi saada valmiiksi. Visuaalinen merkintä

tavoitteesta auttaa motivaation säilymisessä, kun näet, missä vaiheessa projektin tavoite etenee.

Yleisesti pitkästyminen tapahtuu, jos projekti jää seisomaan tai omaa työtä ei voi nähdä käytännössä. Siksi niille on hyvä laittaa jonkinlainen paikka, johon voit merkata työn tehdyksi. Se edesauttaa näkymätöntä työtä. Yksinäisenä kehittäjänä on tärkeää osata suorittaa laadittuja tavoitteita loppuun. Tavoitetta suorittaessa ei kannata tarttua monen asian hoitamiseen yhtäaikaisesti, vaan suorittaa asetettu tehtävä ennen seuraavaa. Kehitysvaiheessa tämä tarkoittaa, että ohjelmointi, suunnittelu ja grafiikan luominen kannattaa ajoittaa omille päivilleen tai erillisiksi työpaketeiksi. Usean asian tekemistä samanaikaisesti ei aina voida välttää, mutta monien eri asioiden tekeminen yhtä aikaa voi johtaa useampiin virheisiin ja keskittymisen heikentymiseen.

Lyhyesti mitä kannattaa ottaa huomioon projektin suunnitelmaa tehdessä:

1. Idea
- Kerrotaan mistä on kyse. Projektin idea ja mitä siinä tapahtuu.
2. Tavoite
- Kerrotaan millaiset tavoitteet projektissamme on.
3. Resurssit
- Kerrotaan paljonko käytetään resursseja ja hyödynnetäänkö aikaisempia projekteja.
4. Aika
- Kerrotaan aikataulutus, jos sellainen on.
5. Julkisuus
- Otetaan selkeä asenne, sille tehdäänkö projekti jonka tarkoitus on tulla jossain vaiheessa julkiseksi.

Hyviin käytäntöihin kuuluu työn dokumentointi, vaikka tekisit työtä itsellesi. Dokumentoinnilla teet työstäsi ymmärrettävän jokaisessa projektin vaiheessa ulkopuolisille ja tulevaisuuden itsellesi. On suositeltavaa luoda projektille oma Git-versiohallinta, johon kirjataan muutokset koko matkan ajalta puhtaasti ja selkeästi kirjoitettuna. (Robertson, 2022)

5 MARKKINOINTI

Indie-kehittäjänä itsensä mainostaminen on erittäin tärkeää, vaikka suurta budjettia mainostamiseen ei olisi tai sitä ei ole lainkaan. Yksi tapa on muodostaa oma yhteisö pelien pariin. Tämä voi tarkoittaa ilmaisen Twitch- tai Youtube-tilin avaamista ja yhteisön kasvattamista alustalla tekemällä videoita tai striimaamalla itselleen mieluisia asioita. Tämän lisäksi on mahdollista laajentaa sitä Instagramiin tai Twitteriin. Nämä ovat vain esimerkkipaikkoja, sillä alustoja missä voit mainostaa projektiasi on loppumattomasti. (Karnes, 2021)

Kuitenkin pelin löytäminen on yhtä tärkeää kuin sen mainostaminen. Puhutaan ASO:sta (App Store Optimization) eli tuotekaupan optimoinnista, jossa tarkoituksena on kirjoittaa tuotteelle nimi, avainsanat, kuvaus ja sovelluskuvakkeet. Myöhemmin sovelluksen arvostelut ovat osana ASO:a. Tätä tulemme tekemään käytännössä projektin loppuosassa, jossa julkaisemme pelin Google Play -kauppaan. (Karnes, 2021)

Jokaisen indie-kehittäjän täytyy osata markkinoida, jos haluaa projektinsa onnistuvan. Ilman kunnollista markkinointia projektisi näkyvyyttä ei juurikaan ole eikä kunnollisia tuloksia. Vaikka projektin tarkoituksena ei olisi panostaa markkinointiin vaan projektin ominaisuuksiin, se silti katoaa muiden kilpailevien projektien sekaan, jos sen näkyvyys ei pääse käyttäjille tietoon. Tämän vuoksi moni alusta on rakentanut ”wishlist” -järjestelmän tai seuraa ominaisuuksia, jossa kehittäjän projektia voi seurata tilapäivityksillä.

Julkaisu on aina tärkeä, sillä onnistunut julkaisu on askel kohti menestystä. Jos julkaisu on onnistunut, voidaan odottaa passiivista näkyvyyttä. Tällöin julkaisualusta mainostaa sovellusta kohdennetusti käyttäjille, jotka saattaisivat olla kiinnostuneita julkaisun tyypistä.

6 PROJEKTI NEOGON

6.1 Projektin aloittaminen

Projektin aloittamisessa oli useita ideoita. Ajatuksina oli muun muassa opetella VR-kehittämistä, tekoälyagentteja Unityssä, valmistaa harrastuksen taustalta Dungeons & Dragons inspiroitunutta peliä, mutta kuitenkaan ei päädytty luomaan niin kunnianhimoista peliä. On helppo valita suuri maali ja asettaa epärealistisia odotuksia, siksi tämän projektin tärkeä rooli on asettaa selkeä maali, jotta ei syntyisi ikuisuusprojektia. Siksi ideoiden havainnollistaminen paperille ja niiden vertailu keskenään on suunnittelussa tärkeää.

Suunnittelu aloitettiin ensimmäisellä aiheella Clarasos Entrepeneur -nimellä, se perustui luotuun fantasiamaailmaan, jossa olisi toimittu yrittäjänä tavoitteena rikastua keräten kuuluisuutta ja rahaa. Kuitenkin projekti oli selkeästi liian iso kooltaan toteutuakseen asettamissa ajoissa ja nykyisen elämäntilanteen takia.

Päädyttiin toiseen ratkaisuun Neogoniin. Neogon on rento yksinkertainen mobiilipeli ja paljon parempi vaihtoehto työn toteuttamiselle. Pelissä olet palomuuritekoäly, jonka tehtävänä on torjua hyökkäyksiä värikkäissä neonvaloissa. Laadittiin alustava suunnitelma digitaaliseen muotoon, jossa sanallisesti kerrottiin miten peli tulisi toteuttaa ja millaisia ominaisuuksia peli sisältää. Suunnitelma siirtyi paperille ajatuskartan muodosssa, joka oli tekemisen hahmottamiselle erinomainen idea, sillä se auttoi hahmottamaan kokonaisuutta.

Tekniset valmistelut jatkuivat luomalla projektille oman GitHub-säiliön ja Unity-version 2022.3.1.

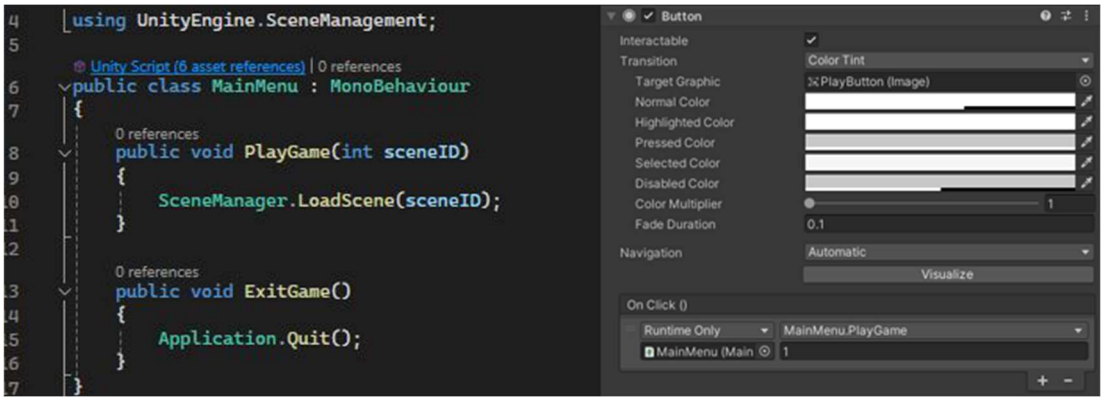
6.2 Valikko

Uuden projektin aluksi luotiin yksinkertainen valikko, joka sisältää muutamia painikkeita, kuten Play, Settings ja Exit. Kuvassa 4 esitellään Unityllä luotu Scene Main Menu, jonka CanvasHolder sisältää valikon objektit. Valikko koostuu kuvista, teksteistä, painikkeista ja MainMenu.cs -skriptistä.

Jokaisen painikkeen takaa löytyy Unityyn integroitu OnClick-ominaisuus, jonka avulla painikkeista saadaan toiminnallisia. Esimerkiksi painettaessa Settings-painiketta, Unity piilottaa nykyisen näkymän ja paljastaa CanvasHolder-objektista SettingsMenu-objektin. Vastaavasti painettaessa Play-painiketta, OnClick-ominaisuus käyttää MainMenu-objektiin kiinnitettyä skriptiä pelin käynnistämiseksi. Skriptissä on kaksi metodia: PlayGame() ja ExitGame().

Skripti vaatii toimiakseen Unityn SceneManagement-kirjaston, joka vastaa pelissä tason vaihtamisesta. Tässä tapauksessa käytetään int-parametriä, sillä build order kertoo, että varsinainen peli sijaitsee id-numerolla 1. Tämän voisi myös toteuttaa string-muodossa kirjoittamalla vastaavan Scene-nimen parametriksi. Kuvassa 5 painikeobjektin OnClick-tapahtuma sisältää linkitetyn MainMenu-skriptin, ja se kutsuu luokan metodia PlayGame() arvolla 1.

Kuva 4. Valikon ulkonäkö ja hierarkian rakenne.



Kuva 5. MainMenu.cs ja OnClick() painike.

Pelin asetuksiin on myös lisätty logiikkaa. Hierarkiasta löytyy peliin kulkeutuva SoundFXManager-objekti, johon on komponentiksi lisätty oma skripti SoundFXManager. Tämä objekti saadaan kulkeutumaan muihin skeneihin Unityn pelimoottorimetodin Awake avulla. Tämä erikoinen metodi kutsutaan aina automaattisesti, kun pelimoottori lataa uuden instanssin. Sen tarkoituksena on alkuvalmistella tehtävät, jotka tapahtuvat ennen pelin alkua tai objektin aktivointia.

Asetuksissa liukusäädintä käyttämällä voidaan määrittää pelille haluttu äänen taso. Äänen voimakkuuden säätö tapahtuu logaritmisesti, mikä tarkoittaa, että pienet muutokset äänen tasossa vaikuttavat eri tavoin suurilla voimakkuuksilla. Tällä saavutetaan paljon tarkempi äänen skaalautuvuus kuin lineaarisesti. Haluttiin, että kaikki peliäänet tulevat tämän kautta, joten on luotu metodit PlaySoundFXClip() ja PlayRandomSoundFXClip(), joita voidaan kutsua pelissä. Niiden toimintana on luoda pelikentälle äänitiedosto käyttäen parametreinään ääniraitaa, paikkaa ja äänitasoa. Kun ääni on kuunneltu loppuun, metodi vastaa myös sen poistamisesta.

PlayRandomSoundFXClip() eroaa sillä, että ääniraitaparametrin sijasta käytetään taulukkoa (Array) yksittäisen tiedoston sijaan.

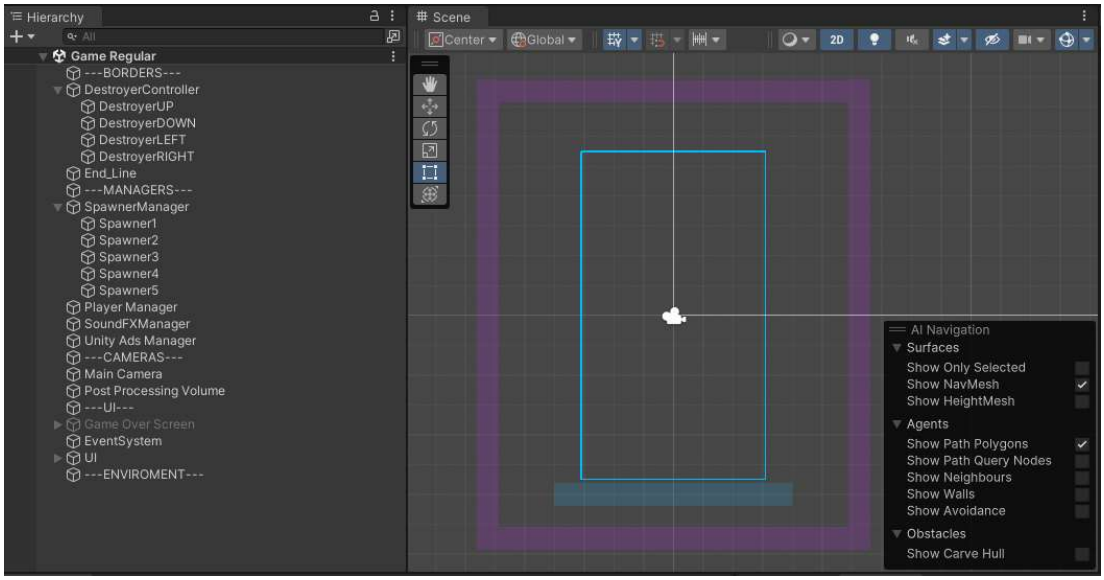
```
Unity Message | 0 references
private void Awake()
{
    if (Instance == null)
    {
        Instance = this;

        DontDestroyOnLoad(gameObject);
    }
    else
    {
        Destroy(gameObject);
    }
}

5 references
public void PlaySoundFXClip(AudioClip audioClip, Transform spawnTransform, float volume)
{
    AudioSource audioSource = Instantiate(soundFXObject, spawnTransform.position, Quaternion.identity);
    audioSource.clip = audioClip;
    audioSource.volume = volume;

    audioSource.Play();

    float clipLenght = audioSource.clip.length;
    Destroy(audioSource.gameObject, clipLenght);
}
```

Kuva 6. SoundFXManager-luokka.

6.3 Kenttä

Peliä tehdessä 2D-muodossa ei tarvinnut paljon ajatella visuaalista viehättävyyttä. Suunnitelmassa oli selkeä kuva, miten kenttä tulisi asettaa, se tarvitsi vain seinät ympärille, jotka rajaavat pelialueen ja siivoavat ylimääräiset objektit. Kuvassa 7 violetti laatikko toimii DestroyerController-objektin alla, joka

sisältää skriptin DestroyerController.cs. Tämä violetti raja pitää huolen, jos mikä tahansa muu asia taikka esine koskee siihen, se tullaan poistamaan pelistä OnCollisionEnter2D-metodin ansiosta. Esimerkiksi kun pelaaja ampuu ohi ja ammus osuu violettiin seinään DestroyerController tarkistaa törmäyksen. Jos törmänneellä peliobjektilla on Rigidbody2D, se poistetaan pelistä.



Kuva 7. Pelimuodon Regular Scene.

Pelille luotiin myös sininen alue End_Line, joka toimii skriptillä EndLineController.cs. Tämä luokka pitää kirjaa paljonko vihollisia pääsee läpi ja rankaisee pelaajaa vähentämällä tämän kestopisteitä. Skripti sisältää OnCollisionEnter2D-metodin, joka laukaisee tapahtumaketjun. Kun vihollinen törmää alueelle, sen peliobjekti tuhotaan, ja siitä luodaan visuaalinen indikaattori kutsumalla PlayDestructionParticles-metodia. Pelaajan kestopisteitä vähennetään DecreasePlayerHealth-metodilla. Tämä metodi etsii PlayerController-luokan pelaajaobjektia, ja jos se löytyy, kutsutaan playerController.TakeDamage(1) rankaisemaan pelaajaa, mikä aiheuttaa peliin äänimerkin.

Kenttä tarvitsi paikat, joihin viholliset syntyvät, joten SpawnerManager-objektin alle luotiin viisi eri syntymäpistettä. Nämä alueet sijaitsevat kuvassa 7 kirkkaan sinisen laatikon yläreunan ja violetin laatikon yläreunan välissä. Viisi spawneria on järjestetty linjaan, mikä tuo satunnaisuutta siihen, mistä vihollinen voi tulla seuraavaksi.

Visuaalinen toteutus työssä jäi pieneksi, mutta sitä elävöitettiin pääkameraan kiinnitetyllä jälkikäsittelykomponentilla, jossa Bloom-ominaisuus loi peliin värikkäitä hohtovaloja. Toinen tavoite oli temppuilla materiaalien ja animaatioiden kanssa, jotka toisivat hologrammisen muodon värikkäille vihollisille. Tämä tavoite kuitenkin jatkuu tulevaisuuden versioon.

6.4 Ampuminen

Pelin tärkein ominaisuus, ampuminen, tapahtuu ruudun alaosassa. Alun perin pelaaja pystyi ampumaan mistä tahansa ruudun kohdasta näpäyttämällä, mutta koettiin tarpeelliseksi rajoittaa ampumista ruudun puolivälistä alaspäin. Alkuperäisessä ideassa oli tarkoitus, että ammus liikkuu samoilla urilla kuin viholliset, mutta päädyttiin antamaan ammukselle vapaa liikkuvuus. Pelin testaamisen aikana huomattiin, että oli mielekkään siistiä, kun pelaaja pystyi tuhoamaan kaksi kohdetta, mikä vaati ennakointia ja ajoitusta. Tämän vuoksi vapaa liikkuvuus päätettiin säilyttää pelissä (Ohjelma 1).

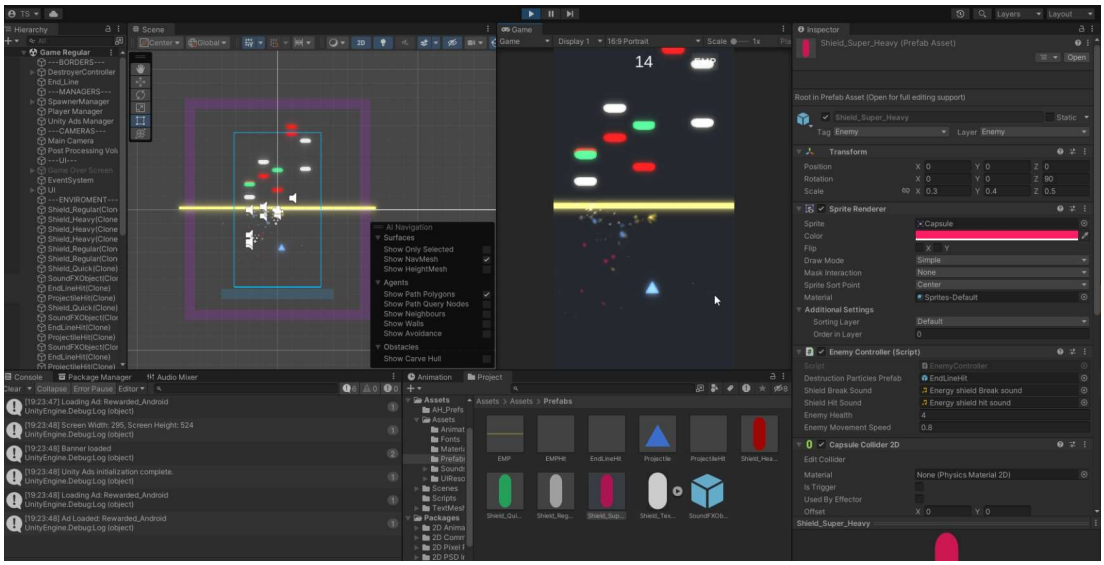
```
    void Update()
    {
        if (Input.GetMouseButtonDown(0) && CanShoot() &&
!isDead)
        {
            Vector2 mousePosition = Camera.main.Screen-
ToWorldPoint(Input.mousePosition);
            if (mousePosition.y < 0f)
            {
                ShootProjectile(mousePosition);
                lastShotTime = Time.time;
            }
        }
    }

    bool CanShoot()
    {
        return Time.time - lastShotTime >= shotCooldown;
    }

    void ShootProjectile(Vector2 position)
    {
        if (projectilePrefab != null)
```

```
        {
            Instantiate(projectilePrefab, position, Qua-
ternion.identity);
            SoundFXManager.Instance.PlaySoundFX-
Clip(shootingSound, transform, 0.5f);
        }
    }
```

Ohjelma 1. Ampumisen toiminta PlayerController-luokassa.

Ampuminen oli kuitenkin liian helppoa, sillä pelaaja pystyi painamaan ruutua niin paljon, että pelissä ei ollut lainkaan haastetta. Tästä syystä päätettiin rajoittaa ammuntanopeutta lisäämällä pieni odotusaika ammusten välille. Ampumisen tuli silti pysyä helppona, sillä normaalin pelimuodon tarkoitus ei ollut rankaista pelaajaa. Suunnitelmassa on myös toinen pelimuoto, jossa pelaaja ei saisi ampua ohi kohteistaan.

Alkuperäinen arsenaali osoittautui kuitenkin liian pieneksi, joten peliin lisättiin myös EMP-ase, joka tuhoaa kaikki kohteet kentällä, mutta sen käyttö maksaa pelaajalle pisteitä ja kustannus tuplaantuu jokaisella käyttökerralla. Kuvassa 8 nähdään keltainen linja, joka liikkuu kentän alhaalta ylös tuhoamalla kaikki kohteet tieltään ja jättäen jälkeensä vihollisesteiden pölyä. Tämä ase on aktivoitavissa pelin oikeasta yläkulmasta, kun painikkeen tausta hohkaa keltaisella värillä.



Kuva 8. Käynnissä oleva peli.

6.5 Viholliset

Pelin ensimmäisessä testivaiheessa käytettiin valkoisia kapselikohteita, jotka toimivat vihollisina pelissä. Näille kohteille asetettiin nopeus ja kestopistemäärä. Yhden kohteen käyttö kävi tylsäksi, ja variaatioita oli mahdollista tehdä näillä kahdella muuttujalla, joten lisättiin uusia prefabeja vihollisista.

Tehtiin punainen vihollinen, joka on yhtä nopea kuin valkoinen, mutta omaa tuplasti enemmän kestopisteitä. Vihreä vihollinen on nopeampi kuin mikään muu kohde, mutta sillä on vain yksi kestopiste. Aniliininpunaisella vihollisella on neljä kertaa enemmän kestopisteitä, mutta se on kaikista hitain. Oli mahtavaa nähdä, kuinka vain kahdella muuttujalla, nopeudella ja kestopisteillä, voidaan luoda erilaisia variaatioita tuhottavista kohteista.

Viholliset tuli myös kategorisoida, mutta niissä ilmeni ongelma, jossa ne törmäsivät toisiinsa. Tämän ongelman korjaaminen vaati layer-peitteen käyttöä, ja jokaiselle vihollispeliobjektille annettiin "Enemy" tag-merkintä. Tämä ratkaisi ongelman, jossa nopeat viholliset työntävät hitaampia vihollisia, sillä niille mahdollistettiin kulkeminen toisten objektien läpi, joilla oli sama "Enemy" tag (Ohjelma 2).

```
int enemyLayer = LayerMask.NameToLayer("Enemy");
Physics2D.IgnoreLayerCollision(enemyLayer, enemyLayer,
true);
```

Ohjelma 2. Tason numeraalin hankkiminen nimellä ja törmäyksen evääminen tasossa sijaitseville kohteille.

Pelissä on tärkeää antaa käyttäjälle palautetta. Osumisen tapahtuessa on luotu indikaattori, joka havainnollistaa tapahtumaa. Tässä vaiheessa Particle System mahdollistaa animaation luomisen, jota voidaan kutsua koodissa. Kun vihollisvariaatio tuhoutuu, koodi muuttaa hiukkasten värit variaatiomuodon mukaan. Kuvassa 9 on esitelty ohjelmakoodi PlayDestructionParticles-metodista. Tämä metodi käyttää kahta parametria: sijaintia ja väriä.

Sen toiminnallisuus tarkistaa, onko prefab olemassa; jos sellainen löytyy, toteutetaan seuraavat komennot: luodaan uusi peliobjekti haluttuun sijaintiin, joka sisältää Particle Systemin. Käytämme GetComponent-komentoa muokataksemme peliobjektin sisäistä Particle Systemiä. Tällä tavoin pääsemme käsiksi hiukkasjärjestelmän päämoduuliin, johon asetamme uuden aloitusvärin. Lopuksi varmistamme, että peliobjekti poistetaan kahden sekunnin kuluessa.

```
1 reference
void PlayDestructionParticles(Vector2 position, UnityEngine.Color color)
{
    if (destructionParticlesPrefab != null)
    {
        GameObject particles = Instantiate(destructionParticlesPrefab, position, Quaternion.identity);
        ParticleSystem particleSystem = particles.GetComponent<ParticleSystem>();
        var main = particleSystem.main;
        main.startColor = new ParticleSystem.MinMaxGradient(color);
        Destroy(particles, 2f);
    }
}
```

Kuva 9. PlayDestructionParticles-metodi EnemyController-luokassa.

```
Unity Script (4 asset references) | 4 references
public class EnemyController : MonoBehaviour
{
    [SerializeField] GameObject destructionParticlesPrefab;
    [SerializeField] private AudioClip shieldBreakSound;
    [SerializeField] private AudioClip shieldHitSound;
    [SerializeField] int enemyHealth = 1;
    [SerializeField] public float enemyMovementSpeed = 1.0f;


    2 references
    public void DecreaseHealth(int amount, Color enemyColor)
    {
        enemyHealth -= amount;
        if (enemyHealth <= 0)
        {
            SoundFXManager.Instance.PlaySoundFXClip(shieldBreakSound, transform, 0.5f);
            PlayDestructionParticles(transform.position, enemyColor);
            Destroy(gameObject);
            PlayerController playerController = FindObjectOfType<PlayerController>();
            if (playerController != null)
            {
                playerController.AddPoints(1);
            }
        }
        else
        {
            SoundFXManager.Instance.PlaySoundFXClip(shieldHitSound, transform, 0.5f);
        }
    }

    Unity Message | 0 references
    private void Update()
    {
        transform.Translate(Vector3.left * enemyMovementSpeed * Time.deltaTime);

    }
```

Kuva 10. EnemyController-luokka.

Vihollisille tehtiin satunnainen ilmaantuminen aiemmin kentälle asetetuista spawn-kohteista. Tätä kaikkea hallinnoi SpawnerManager-luokka. Luokan vastuulla on pitää listaa erityyppisistä vihollisista, todennäkyisyyksistä ja milloin ja mistä kohteiden kuuluisi ilmaantua pelikentälle.

EnemyController ja PlayerController tekevät tiiviisti yhteistyötä, sillä ne pitävät sisällään pelin oleellisinta tietoa, vahinkoa. Vihollisten tehtyä kolmen kestopisteen verran vahinkoa pelaajaan, se aktivoi PlayerControllerissa pelin pysähdyksen ja GameOver-metodin kutsun. Pelin pääteruutu astuu kuvaan, kun pelaaja on niin sanotusti kuollut.

```
Unity Script (1 asset reference) | 0 references
public class SpawnerManager : MonoBehaviour
{
    public List<GameObject> enemyPrefabs;
    public List<float> spawnProbabilities;
    [SerializeField] public float initialSpawnInterval = 2f;
    [SerializeField] public float spawnTimeDecreaseRate = 0.1f;
    [SerializeField] public float minSpawnInterval = 0.5f;
    [SerializeField] public float maxSpawnSpeed = 5f;

    private float nextSpawnTime;
    private float currentSpawnSpeed;

    public Transform[] spawnPoints;
    Unity Message | 0 references
    private void Start()
    {
        InitializeSpawnPoints();
        ResetSpawnTimer();
        currentSpawnSpeed = 0f;
    }
    Unity Message | 0 references
    void Update()
    {
        if (Time.time >= nextSpawnTime)
        {
            SpawnObstacle();
            DecreaseSpawnTime();
            ResetSpawnTimer();
        }
        currentSpawnSpeed = Mathf.Min(currentSpawnSpeed + Time.deltaTime * spawnTimeDecreaseRate, maxSpawnSpeed);
    }

    2 references
    void ResetSpawnTimer()
    {
        nextSpawnTime = Time.time + initialSpawnInterval;
    }

    1 reference
    void DecreaseSpawnTime()
    {
        initialSpawnInterval = Mathf.Max(initialSpawnInterval - Time.deltaTime * spawnTimeDecreaseRate, minSpawnInterval);
    }
```

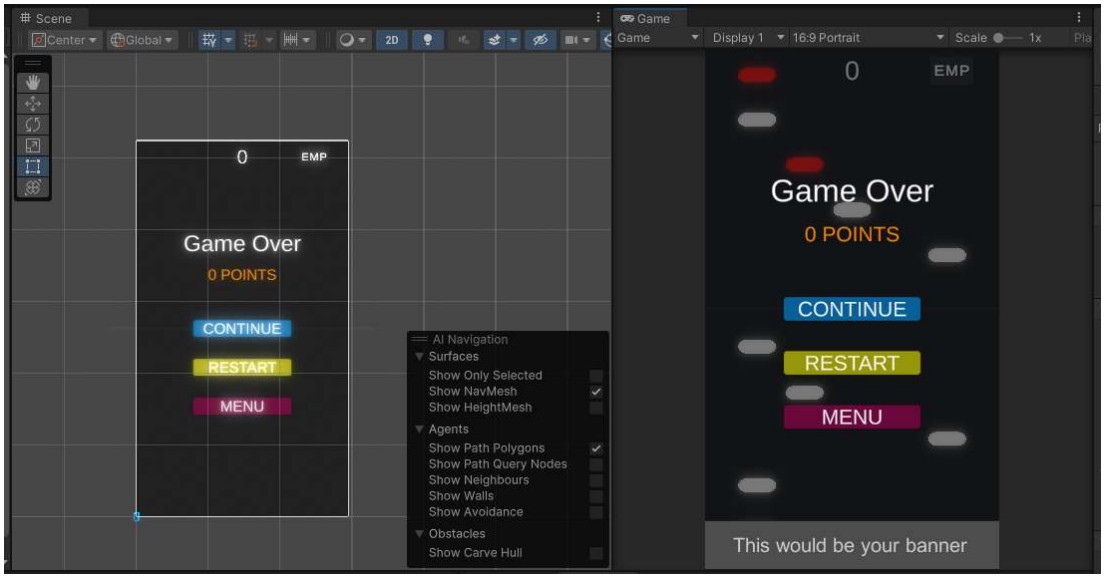
Kuva 11. SpawnerManager-luokka.

6.6 Pisteytys ja mainokset

Pelissä tarvitsee olla jokin maali ja haluttiin, että pelaaja kerää pisteitä tuhoamalla saapuvia kohteita. Mitä enemmän niitä saa tuhottua, sitä enemmän

pisteet kasvavat. Jokainen tuhottu vihollinen vastaa yhtä pistettä ja tällä hetkellä ei ole annettu erityyppisten vihollisten päihittämisestä ylimääräisiä pisteitä. Kuitenkin vaikeampien vihollisten päihittäminen tulisi tulevaisuuden versioissa kompensoida paremmin. Kaikki pelin aikana kerätyt pisteet tallennetaan PlayerController-luokan playerCurrentPoints-muuttujaan, jonka voi nähdä muuttuvan reaaliajassa käyttöliittymän yläreunassa (Kuva 8).

Kuvassa 12 pelin pääteikkuna sisältää kolme painiketta, jotka toimivat seuraavasti. Continue-painike antaa mahdollisuuden jatkaa hävittyä peliä yhdellä elämällä, jos käyttäjä katsoo Unity Ads -palvelun ylläpitämän palkintomainoksen. Restart-painike luonnostaan antaa mahdollisuuden aloittaa pelin täysin alusta ja Menu johdattaa käyttäjän takaisin aloitusvalikolle. Pääteikkunan alareunaan on laitettu Unity Ads -palvelun bannerimainos, joka esiintyy aina, kun pelaaja joutuu pääteruudulle. Luonnostaan pelissä pitää myös näyttää lopullinen tulos ja oranssilla tekstillä käyttäjä näkee viimeisimmän tuloksensa. Tarkoituksena on tulevaisuudessa lisätä tähän Google Playn leaderboard.



Kuva 12. Pelin pääteikkuna.

Unity Ads implementoiminen vaatii kirjautumisen Unity Cloud -palveluihin ja uuden projektin luomisen. Projektin luomisen jälkeen käytetään sen Grow-ominaisuuksia valitsemalla Unity Ads Monetization. Sivustolta tarvitsemme Ad units-osion Android Game ID:n. Tällä ID:llä osoitamme Unitylle oikean

projektin ja käytämme sitä pelimoottorin luodussa AdsInitializer-luokan sisällä. Tämä luokka valmistelee kaikki pelissä toimivat mainokset kuten palkinto- ja bannerimainoksen.

Mainosten implementoinnissa on otettava huomioon, että mainos on päässyt latautumaan ennen sen näyttämistä. Varmistin tämän käyttämällä Unityn Start-metodia, joka käynnistetään kerran ennen Update-metodin alkua ja sijoittamalla AdsInitializer:lle halutun painikkeen ja UI-kankaan. Tämä mahdollisti pelin jatkaminen palkintomainoksen jälkeen ja pääteruudun ilmaantumisessa bannerimainoksen näkyvyyden.

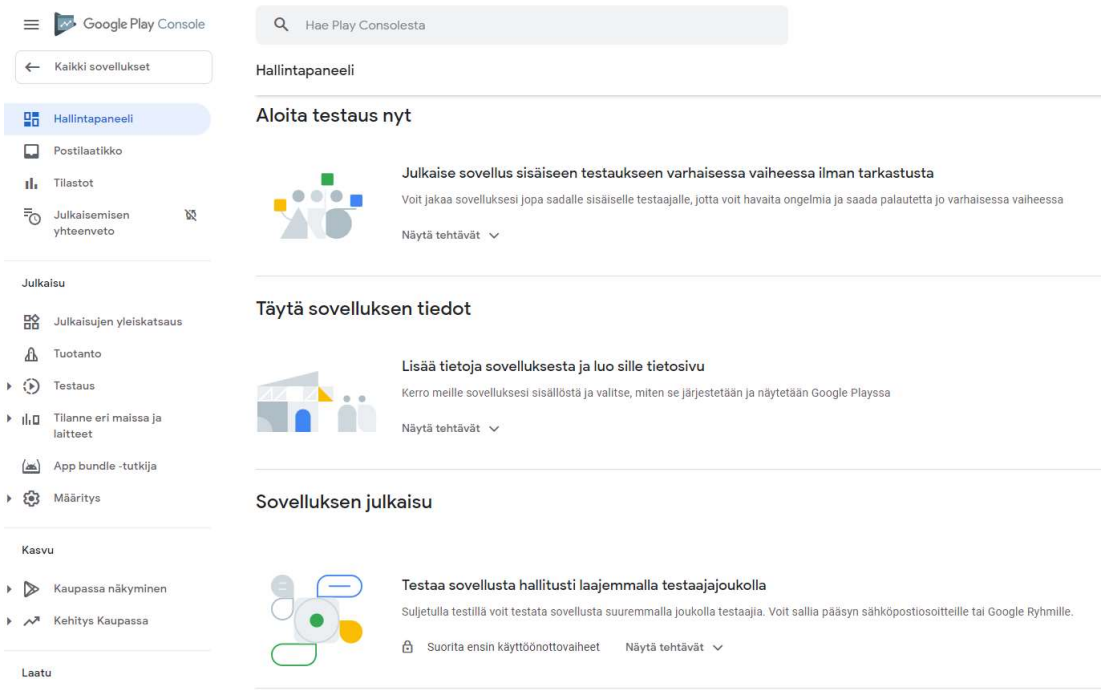
Päänäkymää ohjaa GameOverScreen-luokka, joka huolehtii pelaajan seuraavista toiminnoista. Se pitää kirjaa siitä, onko pelaaja ottanut vastaan ylimääräisen elämän, milloin mainos pitää kutsua tai piilottaa, ja riippuen siitä, jatketaanko keskeneräistä peliä, se valmistelee myös tuhoamaan kaikki "Enemy" -tagilla varustetut peliobjektit.

7 JULKAISEMINEN

Pelin julkaisemisen voi toteuttaa monella tavalla, mutta jakelun kannalta Google Play -kauppa on käyttäjille ystävällisempi ja se vastaa projektin tavoitteisiin julkaista Android-mobiilipeli. Google Play on myös oletuksena jokaisella Android-laitteella. Ennen pelin julkaisemista tarvitsee selvittää seuraavat vaiheet: Tilin luominen, sovelluksen luominen, sovelluksen materiaalien asettaminen, kaupan listaus ja sovelluksen julkaiseminen.

Tilin luominen tapahtuu Google-kehittäjäsivuilta, jossa rekisteröitymiseen tarvitaan vähintään 25 euroa. Kun tilin muut ehdot on hyväksytty, päästään omaan Google-kehittäjänäkymään. Näkymästä oikeasta ylänurkasta voidaan löytää luo uusi sovellus. Kun uusi sovellus määritellään, se täytyy luokitella peliksi tai muuksi sovellukseksi. Sovelluksella tulee olla nimi ja oletuskieli. Uuden sovelluksen luominen edellyttää myös Googlen kehittäjien ohjelmasääntöjen, App signingin ja Yhdysvaltojen vientilakien hyväksymistä. (Google, 2024)

Kun uusi sovellus on luotu, Google ohjaa seuraavaan prosessiin hallintapaneelissa. Sovellusta valmistellessa hallintapaneeli esittää tarvitsemat toimenpiteet. Näet esimerkiksi suosituksia sovelluksen ylläpitoon, testaukseen ja mainontaan liittyen. Kun yksi kohta on valmis, voit palata hallintapaneeliin katsomaan muita ehdotuksia.



Kuva 13. Google-kehittäjä näkymän hallintapaneeli, ehdotukset. (Google Play Console)

Sovelluksen materiaalinen asettelu ja niiden määrittäminen on pakollista. Google ei salli sovellusta julkaisuun, jos käyttöönottovaatimukset eivät ole tehty. Sovellukselta vaaditaan sisällön kuvaus, ulkoasu, rakenne ja sisältääkö sovellus kaupallisia elementtejä. Arvioidaan, onko sovelluksessa jotain mikä ei sopisi kaikenikäisille käyttäjille, mikä on sovelluksen luokka ja löytyykö sieltä mainoksia. Jokainen sovellus vaatii kuvauksen mitä se pitää sisällään.

Kauppalistaus voidaan jättää pois kokonaan, mutta sen käyttöönottaminen vaatii useampia toimenpiteitä, kuten kauppatilin tekemisen. Ominaisuus mahdollistaa sovelluksen hinnoittelun sovelluksen tai sen sisäisen materiaalin ostoissa. Tilaukset ja ostojen seuraukset havaitaan konsolinäkymästä.

Google omaa erinomaiset työkalut kehittäjille testauttaa sovellustaan ennen suurempaa julkaisua. Testaustapoja on monia erilaista tarpeen mukaan, kuten sisäinen testaus, suljettu betatestaus, avoin betatestaus, kevyt julkaisu, ennakkorekisteröinti ja virallinen julkaisu. Mahdollisuutena on käyttää ennakkojulkaisuja, jolloin sovellus on vielä kehitysvaiheessa, mutta sitä voidaan testata

käytännössä. Kehittäjä voi saada arvokasta palautetta ilman, että se vaikuttaisi sovelluksen arvosteluihin julkisella tasolla. (Crovetto, 2020)

Asettaessa sovellusta Google Play-kauppaan on tietynlaisia kriteereitä tilien käytöstä. On siis hyvä tuoda esiin Googlen linjaukset kehittäjien kohdalle. Palvelun pääasiallinen linjaus on, että tilit ovat tarkoitettu vain kehittäjille, jotka aktiivisesti julkaisevat tai ylläpitää applikaatioita Google Playssä. Google jakaa kriteerit kahteen lohkoon. Jos kummastakaan lohkosta löytyy poikkeama, se on syy poistaa kehittäjätilisi. Suositellaan näiden seikkojen sisäistämistä sovelluksen ylläpitoa huomioitaessa.

Epäaktiivinen kehittäjä ilman aplikaatioita:

- Tili on luotu yli vuosi sitten
- Ei ole koskaan lähettänyt aplikaatiota tarkasteluun

Epäaktiivinen kehittäjä aplikaatioilla:

- Tili on luotu yli vuosi sitten
- Kaikkien julkastujen aplikaatioiden käyttäjämäärät yhdistettynä alittavat 1000 latausta
- Kehittäjä ei ole varmistanut puhelinnumeroa ja sähköpostiosoitetta
- Kehittäjä ei ole käyttänyt Play Consolea 180 päivän aikana

Jos nämä asetetut kriteerit eivät täyty, Google on yhteydessä kehittäjään ja antaa kahden kuukauden ajan korjauksiin. On siis tavoituksellista tähdätä 1000 lataukseen julkaisussa, pitää kehittäjätiedot päivitettynä ja käydä vähintään kerran tarkistelemassa sovelluksia Play Console palvelussa, jotta tiliä ei poisteta. (Google, 2024)

Julkaisussa Google on myös tarkka mikä on pelin Android-rajapinnan versio. Projektissa jouduttiin tekemään muutamia ylimääräisiä asennuksia Unityn puolella, jotta saataisiin uusimmat versiot julkaisupakettien tekemiseen. Pelin skaalautuvuus julkaisupakettia testaillessa koitui ongelmaksi, mutta se tarvitsi projektiin pienen lisäyksen ViewPortHandlerin muodossa.

8 YHTEENVETO

Indie-kehittäjän pääsääntöinen tavoite ei ole menestyä, vaan oppia viemään projekteja loppuun ja kehittyä omassa osaamisessaan. Menestys on näiden tulos. Tämä on ainakin oma näkemykseni, mutta jokaisella saa olla oma asettamansa maali. Olin todella tyytyväinen siihen, että projekti on saatu päätökseen pitkän ajan jälkeen. Äkilliset elämänmuutokseni vaikuttivat työn kulkuun ja sen saamisesta loppuun.

Olen aina ollut innostunut pelialan kehityksestä opintojeni ohessa, ja opinnäytetyö peliprojektista oli opettavainen kokemus. Projekti opetti paljon Unityn ja C#-ohjelmointikielen käytöstä, se paransi suunnittelutaitojani ja opetti paljon itsestäni, miten työtä kuuluisi tehdä. Pelin kehittäminen voi siis olla niin vaikeaa kuin kukin sen itse tekee.

Pelissä on vielä paljon asioita, joita haluan lisätä, kuten erilaisia pelimuotoja, musiikkia, kauppa, Google-leaderboards ja moninpeli kaksintaistelu. Nämä kaikki ovat kuitenkin listattuna pelin omassa GitHubissa, joka on mainio paikka pitää kirjaa oman pelin versioista ja tulevaisuudensuunnitelmista. Tällä hetkellä peli on ensimmäisessä versiossaan ja odottaa uusia päivityksiä pelimekaniikkoihin ja grafiikkaan.

Lähdeluettelo

Crovetto, F. (9. 10 2020). *Pre-launch testing for mobile games: tools and best practices on Google Play*. Haettu 18.4.2022 osoitteesta https://medium.com/googleplaydev/test-pre-launch-96d0ef3c4d51

Crump, T. (6. 12 2018). *20 Indie Dev Tips for Awesome Game Design*. Haettu 18.1.2022 osoitteesta https://www.buildbox.com/20-tips-for-awesome-game-design/

Estevez, C. (29. 10 2015). *Project planning for solo game developers*. Haettu 19.1.2022 osoitteesta https://hacknplan.com/project-planning-for-solo-game-developers/

Google. (2024). *Closure of inactive developer accounts*. Haettu 4.11.2024 osoitteesta https://support.google.com/googleplay/android-developer/answer/11605267?hl=en#:~:text=Google%20Play%20Developer%20accounts%20are,fee%20will%20not%20be%20refunded.

Karnes, K. (25. 5 2021). *Indie Game Marketing: A 2021 Approach*. Haettu 31.12.2021 osoitteesta https://clevertap.com/blog/indie-game-marketing/

Mozolevskaya, V. (9. 3 2021). *Indie Game Development: Guide to Revenues, Most Profitable Genres & Monetization*. Haettu 30.12.2021 osoitteesta https://kevurugames.com/blog/indie-game-development-the-all-you-need-guide-to-revenues-most-profitable-genres-monetization-bonus-top-10-best-indie-games-2020/

Ninichi. (12. 9 2017). *11 Places to Publish & Release Your Indie Game*. Haettu 1.2.2022 osoitteesta https://ninichimusic.com/blog/2017/9/1/11-places-to-publish-release-your-indie-game

Perveez, S. (12. 7 2022). *What is Git: Features, Commands, Branch and Workflow in Git*. Haettu 6.10.2022 osoitteesta https://www.simplilearn.com/tutorials/git-tutorial/what-is-git#what_is_git

Robertson, M. (25. 4 2022). *The Effective Solo Developer*. Haettu 4.10.2022 osoitteesta https://betterprogramming.pub/the-effective-solo-dev-8407d86c8a9e

Unity. (2022). *Unity User Manual 2022.3 (LTS)*. Noudettu osoitteesta https://docs.unity3d.com/Manual/index.html

Unity. (n.d.). *Changes to Unity plans and pricing*. Haettu 11.4.2024 osoitteesta https://unity.com/pricing-updates#:~:text=Unity%20subscription%20plans%3F-,Unity%20Personal,-Will%20the%20Runtime