Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

EEVERTTI HEIKKILÄ

2D-mobiilipeli Unity-pelimoottorilla

TIETOJENKÄSITTELYN KOULUTUSOHJELMA
2020

Tekijä(t)
Heikkilä, Eevertti

Julkaisun laji
Opinnäytetyö, AMK

Päivämäärä
18.4.2020

Sivumäärä
33

Julkaisun kieli
Suomi

Julkaisun nimi
2D-mobiilipeli Unity-pelimoottorilla

Tutkinto-ohjelma
Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

Tämä opinnäytetyö kertoo mobiilipelistä, joka kehitettiin animaatio- ja pelifirmalle Nopia Oy:lle käyttäen Unity-pelimoottoria.

Teksti paneutuu myös työkaluihin ja palveluihin, joita kehittämisessä käytettiin, sekä kuvailee kyseisen mobiilipelin kehittämisen prosessia ja esittelee pelin päätoimintoja. Lisäksi tekstissä käsitellään mobiilipelejä ja niiden kehittämistä yleisellä tasolla.

Peli saatiin valmiiksi ja julkaistiin onnistuneesti, ja prosessista opittiin paljon.

Avainsanat
Mobiilipeli, Unity, peliohjelmointi, C#

Author(s) Heikkilä, Eevertti	Type of Publication Bachelor's thesis	Date 18.4.2020
	Number of pages 33	Language of publication: Finnish

Title of publication
2D-mobilegame in Unity-game engine

Degree programme
Degree programme in Business Information Technology

Abstract

This bachelor's thesis describes a mobile game which was developed in the Unity game-engine for Nopia Oy, an animation- and game studio.

The text will go through the tools and services which were used in the development of the game. It will describe the process of creating the game as well as explain its main features. The text will also address mobile games and their development on a general level.

The game was completed and released successfully, and a lot was learned from the process.

Key words
Mobilegame, Unity, game programming, C#

SISÄLLYS

LÄHTEET

1 JOHDANTO

Mobiilipelit ovat viimeisten kymmenen vuoden aikana kehittyneet rajua tahtia, ja ne ovat tänä päivänä käytetyimpiä puhelinsovelluksia. Pelit ovat merkittävä osa ihmisten kulttuuria.

Tämä opinnäytetyö kertoo mobiilipelistä, jonka kehitin Nopia Oy:lle kesällä 2019. Nopia Oy on animaatio- ja peliyritys, joka on aikaisemmin kehittänyt ja julkaissut muutamia pelejä eri asiakkaille. Tämä projekti oli siis yksi ensimmäisistä kokonaan omatoimisesti kehitetyistä peleistä Nopialle. Pelin kehittämisessä käytettiin Unity-pelimoottoria, koska siitä minulla ja Nopialla on eniten kokemusta. Pelin oli tarkoituksena toimia eräänlaisena markkinatestinä, jolla selvitettäisiin ihmisten kiinnostusta pelin aiheeseen.

Koska peliprojektista tuli lopulta hyvin laaja, ei tämän työn tarkoituksena ole paneutua jokaiseen pieneen yksityiskohtaan. Yritän kertoa asioista pääpiirteittäin ja antaa samalla käsityksen pelin päämekaniikoista ja toiminnoista.

Teoriaosuudessa esittelen lyhyesti mobiilipelien historiaa ja sitä, millainen merkitys niillä on nykypäivänä. On hyvä tietää, mistä kaikki alkoi. Tämän jälkeen käsittelen hieman ideointiprosessia ja työkalun valintaa. Vaikka teksti muuten keskittyy Unityn käyttöön, käsittelen muitakin vaihtoehtoja, koska niiden olemassaolo on hyvä tiedostaa. Sitten esittelen eri työkaluja ja palveluja, joita projektissa tuli käytettyä.

Teorian jälkeen kerron itse mobiilipelistä ja sen kehittämisprosessista. Kuvailen pelin pääpiirteitä ja esittelen, miten se toimii. Lopuksi kerron pelin julkaisuprosessista ja siitä, mitä tein vielä julkaisun jälkeen.

2 KATSAUS MOBIILIPELEIHIN

2.1 Mobiilipelien lyhyt historia

Jotta ymmärtäisi paremmin millaisista markkinoista on kyse, on ensin syytä tarkastella mobiilipelien historiaa ja kehitystä, jota käymme läpi tässä luvussa.

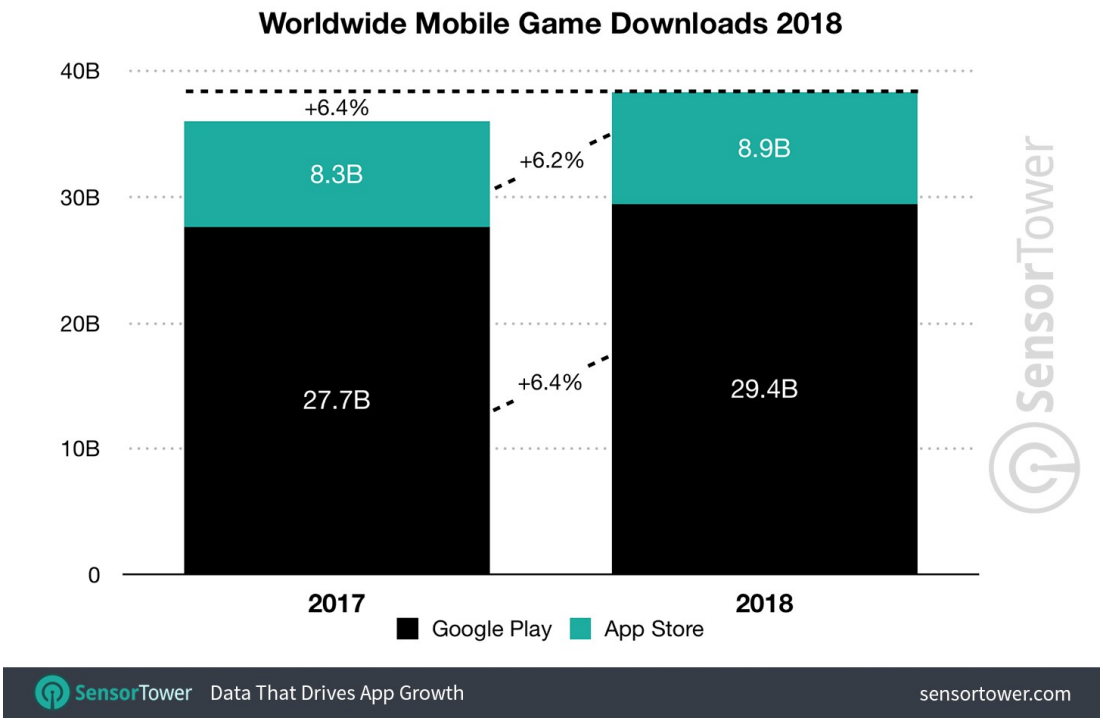
Ensimmäiset viralliset mobiilipelit eivät juurikaan muistuta nykypäivän käsitystä siitä, millaisia mobiilipelit ovat. Yksi tunnetuimpia ensimmäisiä mobiilipelejä oli vuonna 1997 julkaistu Snake. Siinä pelaaja ohjaa palikoista muodostunutta käärmettä, jonka tarkoituksena on syödä ruudulle ilmestyviä pisteitä kasvattaakseen häntäänsä.
Ensimmäisten mobiilipelien ongelma oli se, että ne tulivat puhelimelle suoraan asennettuna, eikä niitä voinut päivittää tai ladata lisää. Vasta 1990-luvun lopussa yleistynyt WAP- eli Wireless Application Protocoll -teknologia mahdollisti mobiilipelien helpon lataamisen ja jakamisen. Mobiilimarkkinat jatkoivat kehitystä tulevina vuosina, mutta suurin mullistus tapahtui vuonna 2007, kun Apple julkaisi ensimmäisen kosketusnäytöllisen iPhonen. Tämän myötä Applen oma mobiilisovellusten jakamisalusta, App Store, avattiin 2008. App Storesta käyttäjät pystyivät lataamaan ja ostamaan appeja helposti. (Crews 2016.)

App Store tarjosi hyviä rahallisia mahdollisuuksia myös kehittäjille. App Storen avaamisen aikaan kehittäjät saivat 70 prosentin osuuden omien appiensa tuotoista, mikä kasvatti heidän kiinnostustaan mobiilikehittämistä kohtaan. (Markoff & Holson 2008.)

Applen menestyneen App Storen myötä oli vain ajan kysymys, milloin muutkin teknologiajätit saapuisivat markkinoille. Paras esimerkki on samana vuonna, eli 2008 Googlen julkaisema Android Market, josta käyttäjät pystyivät lataamaan pelejä ja appeja Android-laitteille. Android Marketin julkaisun jälkeen Google avasi muitakin palveluja, esimerkiksi Google eBookstoren vuonna 2008 ja Google Musicin vuonna 2011. Vuonna 2012 Google kuitenkin päätti yhdistää kaikki palvelunsa yhdeksi eli Google Playksi. (Callaham 2017.)

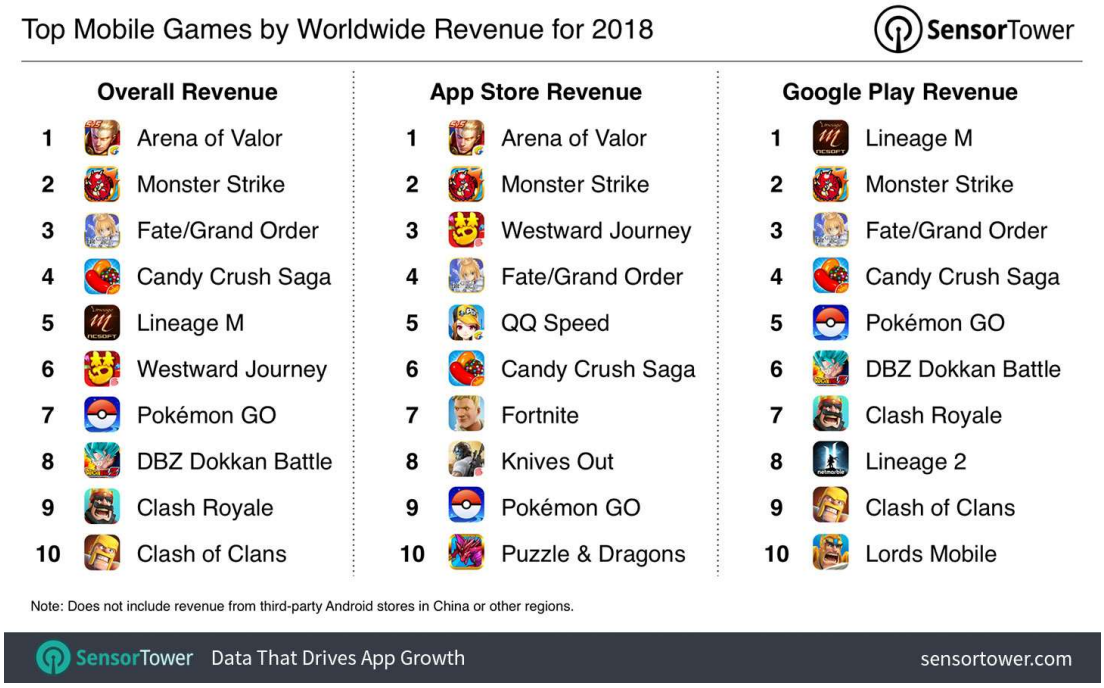
2.2 Mobiilipelien asema nykypäivänä

Nykypäivänä App Store ja Google Play ovat kaksi suurinta appien jakelualustaa koko maailmassa. Vuoteen 2019 mennessä appien latauskerrat saavuttivat 194 miljardia kertaa. Suosituin kategoria oli pelit. Mobiilipelien merkittävyyttä markkinoilla ei siis voi kieltää. Pelit muodostivat 39% Google Playn kaikista latauskerroista ja 30% App Storen latauskerroista. Molemmat latausmäärät kokivat noin 6% kasvun vuoteen 2017 verrattuna kuvan 1 osoittamalla tavalla. (Nelson 2019a; Iqbal 2019.)



Kuva 1. Mobiilipelien latauskertojen kasvu vuosina 2017-2018. (Nelson 2019a.)

Mobiilipelimarkkinoista puhuessa täytyy ottaa huomioon myös Kiina ja sen merkitys. Arvion mukaan vuoteen 2019 mennessä Yhdysvalloissa oli noin 203 miljoonaa pelaajaa ja Kiinassa noin 459 miljoonaa pelaajaa. Kiinan sisäinen käyttäjäkunta on niin suuri, että se näkyy myös tuottavimpien mobiilipelien listassa (Kuva 2). Listan kärjessä oleva Arena of Valor on ainoastaan Kiinan sisäisiä markkinoita varten kehitetty. Kiinan melkein kokonaan suljetut markkinat ja suuri käyttäjäkunta lisäävät länsimaalaisten halua jakaa pelejänsä kyseisessä maassa. (Nelson 2019b; Iqbal 2019.)

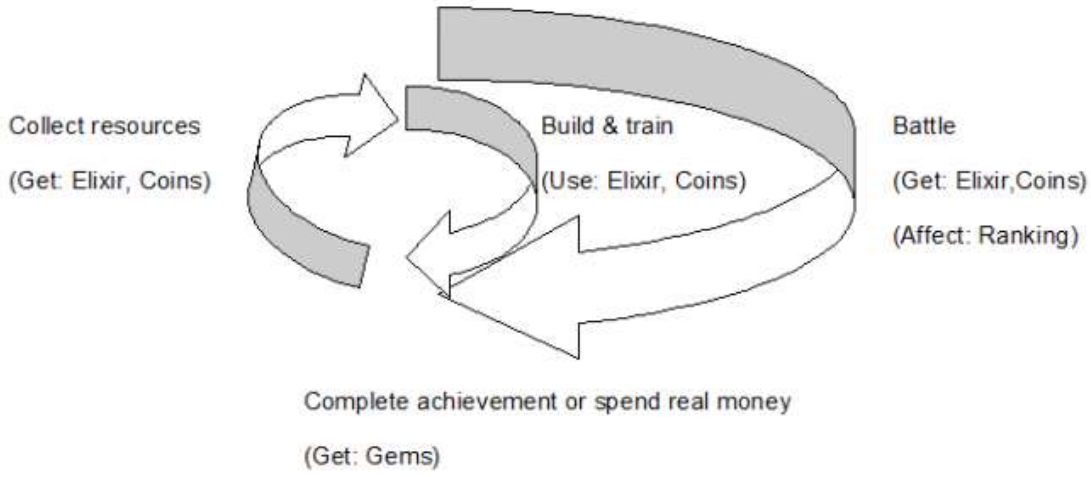


Kuva 2. Tuottavimmat mobiilipelit vuonna 2018. (Nelson 2019b.)

Mobiilipelit ovat tunnetusti erilaisia verrattuna tavallisiin PC- tai konsolipeleihin. Yleistäen voisi sanoa, että mobiilipelejä pelaa eniten sellainen joukko ihmisiä, jotka haluavat kuluttaa aikaa syystä tai toisesta. On kuitenkin myös sellaisia pelaajia, jotka varta vasten istuvat alas pelaamaan puhelimella, eikä heidän vaikutustaan alaan kannata vähätellä. Oli syy pelaamiseen mikä tahansa, mobiilipelit ovat yleensä yksinkertaisia tai luotu niin, että niitä on tarkoitus pelata vain vähän kerrallaan.

Kaikkia suosittuja mobiilipelejä yleensä yhdistää se, että ne on suunnattu “kasuaalipelaajille” enemmän kuin varsinaisille peliharrastajille, eli niitä on vaivatonta pelata aina vähän kerrallaan. Tämä tulee varsin hyvin ilmi, kun tarkastelee mobiilipelien yleisimpiä rahastusmalleja.

Otetaan esimerkiksi Supercellin suosittu Clash of Clans. Sen lisäksi, että peli näyttää visuaalisesti mukaansatempaavalta, se on helppo aloittaa ja sen gameplay loop eli pelaamisen kiertokulku on tyydyttävä. Keräämällä resursseja saa kolikoita ja eliksiiriä, joilla voi päivittää ja hankkia joukkoja, joita vuorostaan voi käyttää taistelemisessa arvon parantamiseksi. Gameplay loop on visualisoitu kuvassa 3. (Koistila 2014.)



Kuva 3. Clash of Clans -pelin gameplay loop. (Koistila 2014.)

Tämän gameplay loopin ympärillä kaikkeen liittyy kuitenkin raha. Rakennusten rakentaminen ja joukkojen kehittäminen vie esimerkiksi aikaa, jonka voi ohittaa käyttämällä oikealla rahalla ostettavia jalokiviä. Mitä pidemmälle pelissä etenee, sitä vähemmän näitä jalokiviä saa ilmaiseksi, mikä tarkoittaa sitä, että tarve kuluttaa oikeaa rahaa kasvaa. (Koistila 2014.) Ihmiset, jotka kuluttavat yhteen peliin rahaa, ovat yleensä pelin suurin tulon lähde.

Vaikka suuri osa rahaa tuottavista mobiilipeleistä seuraa samaa kaavaa kuin Clash of Clans, on kuitenkin olemassa myös toisenlainen joukko mobiilipelejä. Tämä joukko ei välttämättä pyri saamaan pelaajia pysymään pelin parissa, vaan käyttää mainoksia rahan tuottamiseen. Tällaiset pelit ovat yleensä hyvin yksinkertaisia, ja kehittäjillä on tapana tehdä niitä runsaasti tulojen kasvattamista varten. Esimerkkejä tällaisista peleistä ovat yleensä viraalit pikkupelit, kuten vuonna 2013 julkaistu Flappy Bird.

3 PELIN KEHITTÄMISESTÄ YLEISESTI

Tässä luvussa käyn ensin läpi, mitä kannattaa tehdä, ennen kuin aloittaa itse pelin rakentamisen. Vaikka mobiilipelit eroavat PC- tai konsolipeleistä, on silti hyvä ymmärtää joitakin perusasioita, jotta pelistä tulee pelaajalle nautinnollisempi kokemus.
Tämän jälkeen työssä keskitytään enemmän kehittämisen tekniseen puoleen.

3.1 Idea ja suunnittelu

Jesse Schellin (2015, 1) toteaa alkusanoissaan: ”Suunnittele pelejä. Aloita heti! Älä odota! Älä edes jatka tätä keskustelua, vaan aloita suunnittelu heti! Nyt!”.

Jokainen peli alkaa ideasta. Mutta mistä idean saa? Idean saa inspiraatiosta. Inspiraation vuorostaan saa tarkastelemalla maailmaa ja asioita, joita rakastaa. Inspiraatio itsessään muutetaan hyväksi pelidesigniksi ajattelemalla sitä ongelman kautta. Aluksi täytyy yrittää keksiä, mikä ongelma on kyseessä. Miten esimerkiksi voisi tehdä selainpelin, josta teinit pitävät? Monet ihmiset hyppäävät suoraan johtopäätökseen ajattelemalla asiaa heti ratkaisun kannalta, vaikka ongelman esittäminen selkeästi kasvattaa luovuutta ja auttaa keksimään ratkaisuja, joita muut eivät välttämättä edes ajattelisi. Ongelman esittäminen auttaa myös tiimityöskentelyssä, koska kaikki voivat tällöin etsiä ratkaisua selkeästi asetettuihin reunaehtoihin. (Schell 2015, 70-73.)

Kun on päättänyt, millaisen pelin tekee, on siitä syytä tehdä prototyyppejä. Ideoiden ja mekaniikkojen iterointi on avain hyvään lopputulokseen. Vaikka olisi houkuttelevaa tehdä peli niin, että aluksi suunnittelee kunnolla ja sitten toteuttaa, tämä ei kuitenkaan ole mahdollista. Tällainen ratkaisu muistuttaisi 1970-luvulla kehitettyä projektityöskentelyn vesiputousmallia, joka ei sovi tähän tilanteeseen. Vaikka vesiputousmalli kannusti pitempään suunnittelujaksoon, sen ongelmana oli, että iteraatioita ei käytetty. Vasta myöhemmin yleistyneet ketterät mallit, esimerkiksi scrum ja sen variaatiot, sopivat pelin kehittämiseen paremmin. (Schell 2015, 93-99.)

3.2 Työkalun valinta

Markkinat ovat täynnä erilaisia pelimoottoreita, ohjelmointikehyksiä ja kirjastoja, joista osa soveltuu paremmin mobiilikehittämiseen ja osa ei. Vaikka keskityn tarkemmin Unityn käyttöön, käyn seuraavaksi läpi eri vaihtoehtoja ja listaan niiden ominaisuuksia. Ensin erittelen kuitenkin, mitä eroa edellä mainituilla kolmella termillä tässä kontekstissa on.

Kirjasto tarkoittaa jonkinlaista pakettia koodia, jota on tarkoitus käyttää uudelleen. Kirjasto voi olla sovelluslaajennus (DLL) tai suoraa lähdekoodia. Yleensä yksi kirjasto

keskittyy yhteen aihealueeseen, esimerkiksi äänen toistamiseen tai grafiikkojen piirtämiseen. Ohjelmointikehys tarkoittaa kokoelmaa kirjastoja ja työkaluja, joiden tarkoitus on yhdessä ratkaista jokin ongelma. Moottorin erottaminen ohjelmointikehyksestä on epäselvempää, mutta moottorit ovat yleensä kehittyneempiä ohjelmointikehyksiä, jotka sisältävät sisäänrakennetun editorin. (GameFromScratch.com 2015.)

3.2.1 Game Maker Studio 2

Game Maker Studio 2 (jatkossa pelkkä Game Maker) on viimeisin YoYo Gamesin kehittämä ja julkaisema pelimoottori. Game Maker tarjoaa käyttäjälleen aloittelijaystävällisen 'Drag and Drop' -tavan rakentaa pelilogiikkaa, mutta myös sisäänrakennetun C-kieleen perustuvan GML-kielen (Game Maker Language).
Game Maker sisältää monta sisäänrakennettua työkalua, esimerkiksi objektieditorin, skriptieditorin, tiilipohjaisen huone-editorin ja grafiikkaeditorin. Game Makerilla voi kehittää pelejä monelle eri alustalle, esimerkiksi mobiilille, mutta jotkin niistä vaativat erillisen lisenssin. (YoYo Games.com 2020a; YoYo Games.com 2020b.)

Game Maker mielletään yleensä pääasiassa 2D-moottoriksi, koska GML ei tarjoa valmiiksi rakennettuja 3D-funktioita. Ne täytyy itse ohjelmoida. Esimerkkejä peleistä, jotka on tehty Game Makerilla, ovat Undertale, Forager, Spelunky (YoYo Games.com 2020c).

3.2.2 Unity

Unity on Unity Technologiesin kehittämä ja julkaisema pelimoottori, mutta sillä voi myös tehdä muutakin kuin pelejä, esimerkiksi animaatioita ja muunlaisia appeja. Unity käyttää skriptikielenä hieman muokattua JavaScriptiä, joka tunnetaan myös nimellä UnityScript, tai puhdasta C#-kieltä. Tästä syystä ulkopuolisten kirjastojen käyttäminen on myös helppoa. Unityn avulla peliä voi kehittää reaaliajassa, mikä tarkoittaa sitä, ettei koko projektia tarvitse rakentaa jokaista testiä varten, vaan testauksen voi aloittaa suoraan näkymästä siitä kohdasta mistä haluaa. Muuttujia voi myös muokata reaaliajassa. Unitylla voi kehittää pelejä monelle eri alustalle, ja maksaa tarvitsee vasta,

kun oma peli myy tietyn verran vuodessa. Tarjolla on myös kuukausimaksullisia lisenssejä, joiden mukana tulee enemmän ominaisuuksia. (Unity.com 2020a.)

Unity on pääasiassa tarkoitettu 3D-kehittämiseen, mutta siitä löytyy myös työkalut 2D-pelejä varten. Esimerkkejä peleistä, jotka on tehty Unitylla, ovat Hollow Knight, Cuphead, Subnautica (Unity.com 2020b).

3.2.3 MonoGame

MonoGame on MonoGame Teamin kehittämä ja julkaisema peliohjelmointikehys. MonoGame on kokonaan ilmainen ja vapaata lähdekoodia, pääasiassa C#-kielelle tarkoitettu ohjelmointikehys. MonoGame mahdollistaa kehittämisen monille eri alustoille. Koko ohjelmointikehyksen voi myös halutessaan siirtää muille alustoille. (MonoGame.net 2020a.)

MonoGamella tehdään pääasiassa 2D-pelejä, mutta sen 3D-mahdollisuudet ovat kehittyneet paljon viime aikoina yhteisön toimesta. Esimerkkejä peleistä, jotka on tehty MonoGamella, ovat Stardew Valley, Celeste, Bastion (MonoGame.net 2020b).

4 KÄYTETYT TEKNIIKAT

Tässä luvussa esittelen kaikki ne palvelut ja työkalut, joita projektissa tuli käytettyä. Aloitan lyhyellä kuvauksella itse Unitystä, jonka jälkeen katsaus Amazon Web Servicesiin ja sen tietokantaratkaisuun, ja lopuksi muita yksittäisiä kirjastoja. Näiden käyttöön projektin kannalta paneudutaan vasta myöhemmin.

4.1 Unity

Kun Unityn käynnistää ensi kertaa, täytyy käyttäjän valita 2D- tai 3D-pohja sen perusteella, millaisen projektin aikoo tehdä. Näissä kahdessa ei ole muuta eroa kuin se, että 3D-pohja luo valmiiksi pelin valo-objektin ja valmistelee kameran 3D:ta varten. Pohja

määrittelee myös sen, tuodaanko grafiikat tekstuureina vai sprite-muodossa. 2D-peleissä grafiikka-resurssit ovat oletuksena sprite-muodossa. (Unity3d.com 2020a.)

Unityn käyttöliittymä koostuu erilaisista telakoitavista ikkunoista. Puhtaassa projektissa keskellä on Scene, oikealla Inspector, vasemmalla Hierarchy, ylhäällä Toolbar ja alhaalla Project. Unityn näkymät koostuvat peliobjekteista, jotka näkyvät aina näkymän Hierarchy-ikkunassa. Peliobjektien komponentit näkyvät Inspector-ikkunassa. Project-ikkunassa pystyy tarkastelemaan, tuomaan, ja muokkaamaan resursseja ja niiden asetuksia. Toolbar säilyttää työkaluja näkymässä liikkumista varten, ja painikkeet, joilla käynnistää ja lopettaa testauksen. (Unity3d.com 2020b.)

Kaikki projektissa käytettävät tiedostot ovat resursseja. Resursseja voi joko tuoda ulkopuolelta tai luoda Unityn sisällä. Tällaisia sisäisiä resursseja ovat esimerkiksi Controller-resurssit. Resurssien import-asetukset näkyvät Inspector-ikkunassa. Import-asetukset vaikuttavat siihen, miten resurssit käyttäytyvät pelin aikana, ja missä muodossa tai miten ne pakataan lopulliseen projektiin. (Unity3d.com 2020c.)

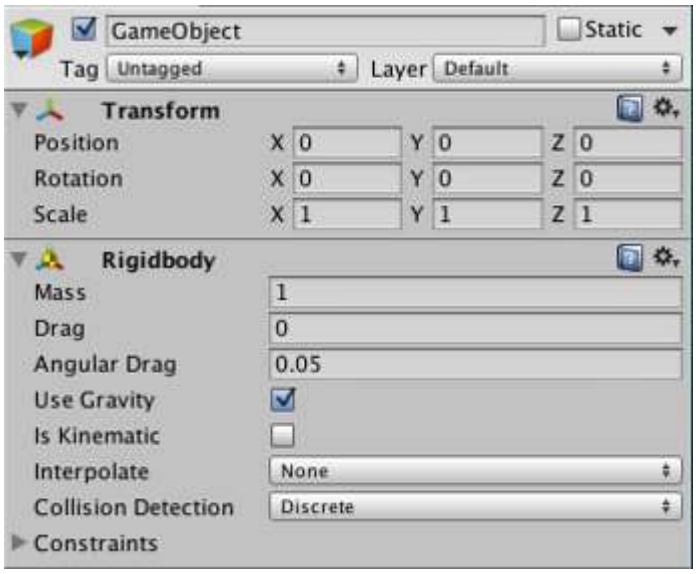
4.1.1 Komponentit ja skriptit

Jokainen peliobjekti on kuin tyhjä kangas, joka saadaan tekemään erilaisia asioita komponenttien avulla. Yksi komponentti mahdollistaa esimerkiksi sen, että pelaaja voi liikuttaa kyseistä peliobjektia, ja toinen mahdollistaa sen, että peliobjekti reagoi viholisiin. Unity sisältää paljon erilaisia komponentteja erilaisia tilanteita varten, mutta kaikki niistä ovat pohjimmiltaan kuitenkin skriptejä. Unity käyttää skriptikielenä C#-kieltä tai omaa variaatiota Javascriptistä, joka tunnetaan nimellä Unityscript (Unity3d.com 2020d).

Unity tukee suurinta osaa C#-kielen ominaisuuksista, joten skriptejä voi pääasiassa kirjoittaa normaalilla C#-tietämyksellä. Kaikkien skriptien, joissa halutaan käytettävän Unityn ominaisuuksia, täytyy kuitenkin periytyä Unityn MonoBehaviour-luokasta. MonoBehaviour mahdollistaa esimerkiksi Unityn perusfunktioiden Start() ja Update() käyttämisen. Nämä funktiot ja niiden variaatiot ovat perusta kaikelle pelilogiikalle. Start-funktioon laitetaan kaikki, jonka halutaan tapahtuvan vain kerran, kun peliobjekti

ensikertaa aktivoidaan. Update-funktio sisältää kaiken, minkä halutaan tapahtuvan joka pelikuvaruudun aikana, eli esimerkiksi pelaajan komentoihin reagoimisen. (Unity3d.com 2020e.)

Unityn vahva komponenttisysteemi mahdollistaa sen, että skriptit voivat helposti keskustella keskenään. Skriptin sisällä julkisiksi määritellyt muuttujat tulevat näkyviin komponentin kohdalle peliobjektin Inspector-ikkunassa, joka on esitetty kuvassa 5. Näiden julkisten muuttujien arvoja voi helposti muokata toisten skriptien kautta tai manuaalisesti suoraan Inspector-ikkunasta. Tämä mahdollistaa esimerkiksi myös sen, että tiimityöskentelytilanteissa ohjelmoijat tekevät pohjatyön, jonka päälle esimerkiksi kenttäsuunnittelijat rakentavat oman osuutensa ilman, että heidän tarvitsee kirjoittaa itse koodia. (Unity3d.com 2020f.)



Kuva 5. Peliobjektin komponentit Inspector-ikkunassa. (Unity3d.com 2020f.)

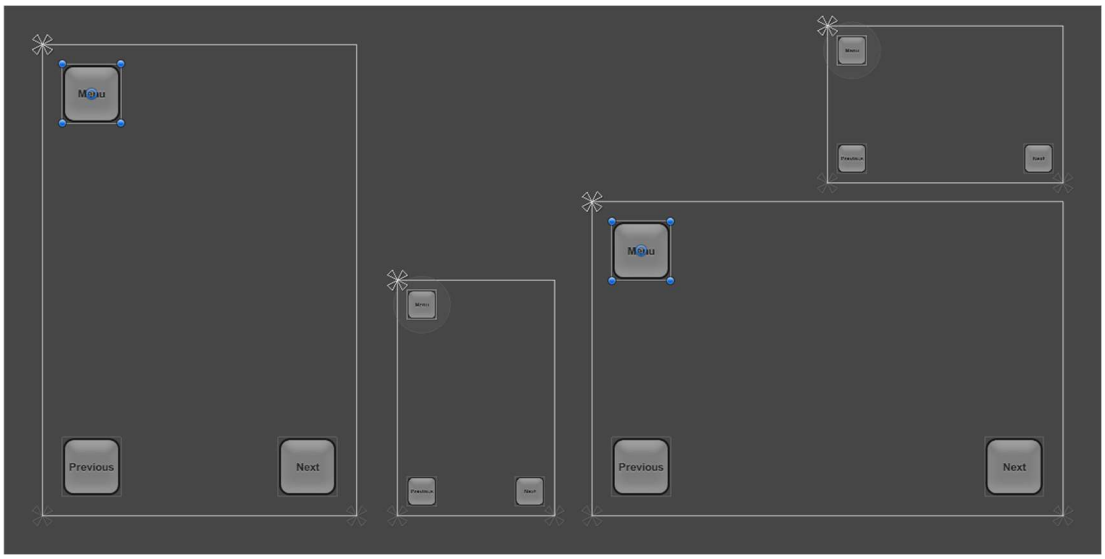
4.1.2 Käyttöliittymä

Käyttöliittymä, tarkoittaa pelissä jonkinlaista systeemiä, jonka avulla pelaaja käyttää peliä. Esimerkkejä tällaisista voivat olla esimerkiksi valikot, ruudulla näkyvät kontrollit ja erilaiset tilastot. Koska puhelimilla ei tavanomaisesti ole erillisiä peliohjaimia tai

vastaavia ohjauslaitteita, on mobiilipeleille tullut tavaksi esittää kontrollit ruudun kulmissa erilaisina painikkeina. Tämän takia on hyvin tärkeää hallita Unityn käyttöliittymän rakennustyökalut, jotta peliohjaimista tulisi mahdollisimman mukavat pelaajaa kohtaan.

Unityssa kaikkien UI-elementtien täytyy olla Canvas-peliobjektin lapsia. Canvas-peliobjektilla on normaalisti kolme eri renderöinti tilaa, jotka vaikuttavat siihen, millä tavalla Canvas-peliobjektin elementit piirretään ruudulle. ‘Screen Space – Overlay’ piirtää Canvas-peliobjektin elementit näkymän päälle, ja vaihtaa elementtien kokoa ruudun koon mukaan. ‘Screen Space – Camera’ on sama kuin edellinen, mutta se piirtää Canvas-peliobjektin jonkin tietyn kameran eteen. Kolmas, eli ‘World Space’, aiheuttaa sen, että Canvas-peliobjekti käyttäytyy samalla tavalla kuin mikä tahansa muukin peliobjekti. (Unity3d.com 2020g.)

Canvas-peliobjektin lapset koostuvat yleensä erilaisista valmiiksi tehdyistä UI-komponenteista. Näitä komponentteja ovat esimerkiksi teksti, kuva, nappi, lista, jne. Jokainen UI-komponentti korvaa normaalin Transform-komponentin RectTransform-komponentilla, jota tarvitaan ankkuroimaan elementti oikein johonkin kohtaan ruutua. Oikeilla asetuksilla elementeistä saa sellaisia, että ne skaalautuvat oikein jokaisen laitteen ruudulle, kuvan 6 osoittamalla tavalla. (Unity3d.com 2020g.)



Kuva 6. Esimerkki hyvin skaalautuvista UI-elementeistä erilaisilla canvaksilla. (Unity3d.com 2020h.)

4.2 Unity Services

Unity Services on Unityn tarjoama paketti palveluja, jotka ovat yleensä käytännöllisiä mobiilipelikehittämistä varten. Näitä palveluja ovat Unity Ads, Unity Analytics, Unity Cloud Build ja Unity Multiplayer. Unity Services otetaan käyttöön projektikohtaisesti linkittämällä se uniikkiin Unity Services Project ID:hen. (Unity3d.com 2020i.)

4.2.1 Unity Ads

Unity Ads tuo mukanaan “Advertisements”-kirjaston, jonka avulla mainokset voi toteuttaa. Advertisement.Initialize()-funktio valmistelee mainoksen, ja Advertisement.Show() -funktio näyttää mainoksen, jos sellainen on valmiina. Palkinnon antavia mainoksia varten luokan täytyy toteuttaa IUnityAdsListener-rajapinta, joka sisältää metodit tarkastamista, palkitsemista ja virheiden käsittelyä varten. Banner-mainokset toimivat samalla tavalla, mutta niiden näyttämistä varten tarvitaan Coroutine, joka koko ajan tarkistaa mainosten saatavuuden. (Unity3d.com 2020j.)

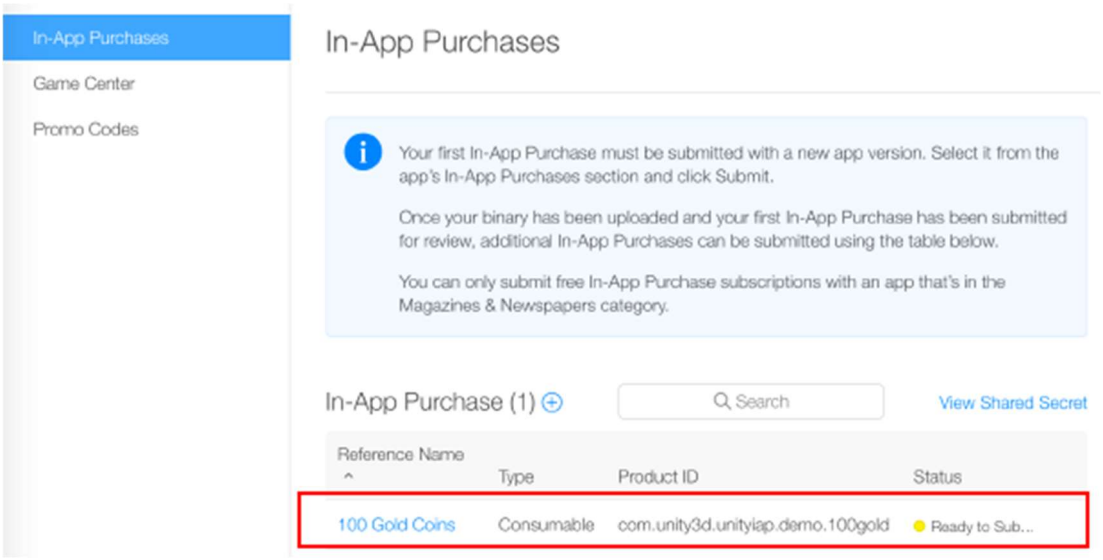
4.2.2 Unity IAP

Unity IAP tuo mukanaan “Purchasing”-kirjaston, joka sisältää kaiken ostotapahtumiin liittyvän. Halutun luokan täytyy toteuttaa IStoreListener-rajapinta, joka sisältää metodit oston käsittelyä ja tuotteiden esittelyä varten. Jokaiselle tuotteelle tarvitsee antaa uniikki ID, joka koostuu pelin paketin nimestä ja tuotteen nimestä, kuvan 7 osoittamalla tavalla. (Unity3d.com 2020k.)

```
public class MyStoreClass : MonoBehaviour, IStoreListener {
    void Start() {
        var module = StandardPurchasingModule.Instance();
        ConfigurationBuilder builder = ConfigurationBuilder.Instance(module);
        builder.AddProduct("com.unity3d.storeguidedemo.100gold", ProductType.Consumable);
        UnityPurchasing.Initialize(this, builder);
    }
```

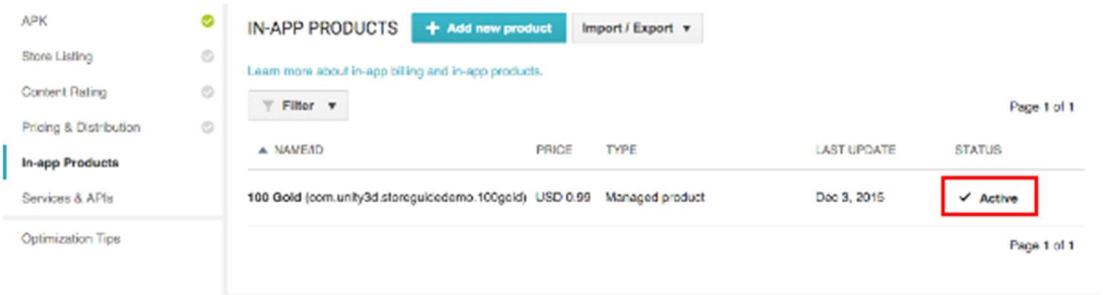
Kuva 7. Esimerkki tuotteen esittelystä IStoreListener-rajapinnassa. (Unity3d.com 2020l.)

Jokainen tuote täytyy yksitellen määritellä myös jokaisen käytettävän kaupan asetuksissa. App Storessa tuotteiden määrittely löytyy rekisteröidyn pelin asetuksista. Tuotteen määritteiden, kuten nimen ja hinnan täytyy vastata täysin koodissa määriteltyjä tuotteita kuvan 8 osoittamalla tavalla. Tuotteita voi tämän jälkeen testata maksutta käyttämällä määriteltyjä testaajia. (Unity3d.com 2020l.)



Kuva 8. App Storen IAP määrittelysivu. (Unity3d.com 2020l).

Google Playn kanssa periaate on täysin sama, mutta tuotteet määritellään Google Play Consolen kautta pelin asetuksista kuvan 9 osoittamalla tavalla. (Unity3d.com 2020m).



Kuva 9. Google Playn IAP-määrittelysivu. (Unity3d.com 2020m).

4.3 Amazon Web Services

Amazon Web Services (jatkossa pelkkä AWS) on Amazonin tarjoama pilvialusta, joka sisältää yli 175 erilaista pilvipalvelua eri tarpeisiin. AWS:n tarjoamat palvelut käsittävät esimerkiksi tietokannat, koneoppimisen, tekoälyn, data-analytiikan ja IoT:n. (Amazon.com 2020a.)

4.3.1 DynamoDB

DynamoDB on AWS:n tarjoama noSQL-ietokanta, joka on toteutettu (avain,arvo)- ja dokumenttitekniikalla. DynamoDB skaalautuu automaattisesti käyttäjän tarpeisiin, ja hajauttaa käyttäjän datan tulevan liikenteen eri AWS:n alueille nopeampaa käyttöä varten. (Amazon.com 2020b.)

Aivan ensiksi käyttäjän täytyy luoda tauluja. Taulujen luonti tapahtuu selaimella DynamoDB:n konsolissa. Jokaiselle taululle täytyy syöttää yksi perusavain. Toimiakseen Unityn kanssa täytyy projektiin ensin ladata AWS Mobile SDK. Ennen operaatioiden tekemistä DynamoDB API täytyy yhdistää tietokantaan, joka tehdään kuvan 10 osoittamalla tavalla. (Amazon.com 2020b.)

```
var credentials = new CognitoAWSCredentials(IDENTITY_POOL_ID, RegionEndpoint.USEast1);
AmazonDynamoDBClient client = new AmazonDynamoDBClient(awsCredentials);
DynamoDbContext Context = new DynamoDBContext(client);
```

Kuva 10. Esimerkki tietokantayhteyden luomisesta. (Amazon.com 2020b.)

DynamoDB API mahdollistaa kätevän tavan tiedon tallentamiseen, jossa tietokantaa varten räätälöidyt luokat voi kirjoittaa suoraan tauluun (Amazon.com 2020b). Tällaisesta luokasta ja sen käytöstä näkyy esimerkit kuvissa 11 ja 12.

```
[DynamoDBTable("ProductCatalog")]
    public class Book
    {
        [DynamoDBHashKey]   // Hash key.
        public int Id { get; set; }
        [DynamoDBProperty]
        public string Title { get; set; }
        [DynamoDBProperty]
        public string ISBN { get; set; }
        [DynamoDBProperty("Authors")]    // Multi-valued (set type) attribute.
        public List<string> BookAuthors { get; set; }
    }
```

Kuva 11. Esimerkki tietokantaa varten räätälöidystä luokasta. (Amazon.com 2020b.)

```
private void PerformCreateOperation()
{
    Book myBook = new Book
    {
        Id = bookID,
        Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
        ISBN = "111-1111111001",
        BookAuthors = new List<string> { "Author 1", "Author 2" },
    };

    // Save the book.
    Context.SaveAsync(myBook,(result)=>{
        if(result.Exception == null)
            resultText.text += @"book saved";
    });
}
```

Kuva 12. Esimerkki objektin tallentamisesta tietokantaan. (Amazon.com 2020b.)

4.3.2 Lambda

Lambda on AWS:n tarjoama pilvipalvelu, joka mahdollistaa valmiiden skriptien suorittamisen joko automaattisesti tai manuaalisesti. Tiettyä Lambda-palvelua voi kutsua json-muodossa olevalla syötteellä, tai ohjelmoida Lambda-palvelu reagoimaan itsestään AWS:ssä tapahtuviin muutoksiin, esimerkiksi rivin lisäämiseen tietokantaan. (Amazon.com 2020c.)

Lambda tarvitsee toimiakseen erilaisia oikeuksia, esimerkiksi oikeuden käsitellä tiettyjä tietokannan tauluja. Nämä oikeudet määritellään AWS:n rooli-ikkunassa JSON-muodossa. Itse Lambda-funktiot kirjoitetaan JavaScriptillä ja ladataan pilveen AWS:n konsolin kautta. Lambda-funktioita kutsutaan kuvien 13 ja 14 osoittamalla tavalla. (Amazon.com 2020c.)

```
var request = new InvokeRequest()
{
    FunctionName = "hello-world",
    Payload = "{\"key1\" : \"Hello World!\"}",
    InvocationType = InvocationType.RequestResponse
};
```

Kuva 13. Lambda-kutsu -objektin luonti. (Amazon.com 2020c.)

```
Client.InvokeAsync(request, (result) =>
{
    if (result.Exception == null)
    {
        Debug.Log(Encoding.ASCII.GetString(result.Response.Payload.ToArray()));
    }
    else
    {
        Debug.LogError(result.Exception);
    }
});
```

Kuva 14. Lambda-kutsu ja tuloksen käsittely. (Amazon.com 2020c.)

4.4 Json.NET

Json.NET on Newtonsoftin kehittämä ohjelmointikehys, joka mahdollistaa helpon muunnoksen .NET objektien ja JSONin välillä. Json.NET toimii myös kätevästi C#-kielen LINQ:n (Language Integrated Query) kanssa, jos haluaa vain tiettyjä tietoja ulos JSON-objektista ilman, että tekee sitä varten oman luokan. Kuvassa 15 on esimerkki serialisoinnista. (Newtonsoft.com 2020.)

```
Product product = new Product();

product.Name = "Apple";
product.ExpiryDate = new DateTime(2008, 12, 28);
product.Price = 3.99M;
product.Sizes = new string[] { "Small", "Medium", "Large" };

string output = JsonConvert.SerializeObject(product);
//{
//  "Name": "Apple",
//  "ExpiryDate": "2008-12-28T00:00:00",
//  "Price": 3.99,
//  "Sizes": [
//    "Small",
//    "Medium",
//    "Large"
//  ]
//}

Product deserializedProduct = JsonConvert.DeserializeObject<Product>(output);
```

Kuva 15. Esimerkki JSON-muunnoksesta. (Newtonsoft.com 2020.)

4.5 Google Play Games

Google Play Games on Googlen kehittämä ja ylläpitämä vapaan lähdekoodin Unity-lisäosa, joka mahdollistaa kaiken kommunikoinnin Google Playn kanssa. Lisäosa integroituu Unityn Social-rajapintaan, joka normaalisti toimii ainoastaan App Storen kanssa. Tämän ansiosta samoja Social-rajapinnan toimintoja voi käyttää myös Google Playn kanssa. Näitä toimintoja ovat esimerkiksi käyttäjän tunnistautuminen, saavutukset ja erilaisten tietojen pilveen tallentaminen. (Github.com 2020.)

5 PROJEKTIN ALOITTAMINEN

Projekti, jota kutsun tästä lähin sen koodinimellä “Okupeli”, alkoi siitä, että minulle kerrottiin pohjatiedot ja reunaehdot. Minulle esiteltiin Adoben Flash-tekniikkaan perustuva 2D-peli nimeltään “Leo’s Red Carpet Rampage” (Kuva 16), josta projektin olisi tarkoitus hankkia inspiraatiota. Okupelistä tulisi 2D ns. ikuinen juoksija (”infinite runner”) Androidille ja iOS:lle. Tarkoituksena on siis ohjata automaattisesti juoksevaa

hahmoa, joka nappia painamalla hyppää esteiden yli. Okupeli on jaettu satunnaisesti generoituihin juoksuratoihin, jotka vaikeutuvat, mitä pitempään selviää, eli tasot muuttuvat nopeammiksi ja esteitä ilmestyy enemmän. Myöhemmillä tasoilla pelaaja saa myös kyvyn ampua laser säteitä, joilla voi tuhota tiettyjä vastaan tulevia esteitä. Jokaisen tason välissä tulee pieni minipeli. Jokainen minipeli kestää hyvin vähän aikaa, ja koostuu jostakin yksinkertaisesta tehtävästä. Okupelin artisti, eli grafiikoista vastaava henkilö, oli ennen aloittamistani tehnyt jo muutaman mallinnuksen pelin visuaalisesta ilmeestä, joita käytin apuna ideoimisessa.

Kuva 16. Leo's Red Carpet Rampage, josta Okupeli inspiroitui. (Redcarpetrampage.com)

Näitä kaikkia pohjatietoja lukuun ottamatta minulle annettiin hyvin vapaat kädet Okupelin suhteen. Suurin osa ideoista kehittyi ajan myötä minun tai muiden toimesta sen perusteella, mikä tuntui hyvältä.

6 PELIN TOTEUTTAMINEN

Tässä luvussa käyn läpi, miten Okupeli toteutettiin käytännössä. Käsittelen myös pelin julkaisua Google Playssa ja App Storessa, ja tämän jälkeistä ylläpitämistä.

6.1 Juoksutasot

Juoksutasot ovat Okupelin pääidea. Pelihahmo juoksee automaattisesti niin, että pysyy koko ajan ruudun vasemmassa laidassa. Pelaajan vastuulla on hypätä vastaan tulevien esteiden yli, tai vaihtoehtoisesti ampua ne lisäpisteitä tienatakseen. Vastaan tulee välillä myös tavaroita, joista saa pisteitä tai lisää energiaa. Kuvassa 17 näkee kuvakaappauksen normaalista juoksutasosta.

Kuva 17. Kuvakaappaus juoksutasosta.

6.1.1 Pelihahmo

Koska pelihahmon liike on niin yksinkertaista, se käyttää totuusarvoja tilan määrittelemiseen aidon tilakoneen sijasta. Kaikki pelaajan fysiikat, kuten painovoima ja syötteiden tarkkailu, tapahtuvat Update-funktiossa, eli kerran joka pelikuvaruudun aikana. Pääperiaate kaikelle liikkumiselle on se, että pelaajalla on float-tyyppinen (desimaaliluku) muuttuja ”yspeed”, joka kertoo, kuinka paljon pelaaja liikkuu y-akselilla yhden framen aikana. Suoritusjärjestys Update-funktion sisällä tapahtuu seuraavasti kuvan 18 osoittamalla tavalla. Ensin yspeed päivitetään painovoimalla ja pelaajan aiheuttamalla hypyllä. Sitten tehdään yhteentörmäystarkistus maan kanssa, eli pelaajan yspeed-muuttuja asetetaan nollaan, jos pelaaja osuu maahan. Vasta lopuksi itse pelaajan

koordinaatteja muutetaan yspeed-muuttujan verran. Samaa logiikkaa käytettäisiin myös ”xspeed”-muuttujalle, jos pelaajan annettaisiin liikkua x-akselilla. Kuvassa 18 näkyy myös, että yspeed-muuttuja on aina sitä muutettaessa kerrottu arvoilla 60f ja Time.deltaTime. Time.deltaTime on se aika, joka on kulunut tämän ja edellisen pelikuvaruudun välillä ja 60f tarkoittaa pelin tarkoitettua ruudunpäivitysnopeutta. Näillä arvoilla kertominen pitää huolen siitä, että peli toimii samalla vauhdilla, vaikka laitteen fps (frames per second) olisi pienempi kuin 60.

```
//Gravity
if (!onGround) {
    //If you hold in the air you can 'float' longer
    if (jumpHeld && floatTimer > 0) {
        yspeed = Mathf.Clamp(yspeed -= 0.003f * 60f * Time.deltaTime, 0, 1);
        floatTimer -= 1f * 60f * Time.deltaTime;
    } else {
        yspeed -= 0.003f * 60f * Time.deltaTime;
    }
    if (yspeed < 0 && !animator.GetCurrentAnimatorStateInfo(0).IsName("Anim_oku_fall")
                   && !animator.GetCurrentAnimatorStateInfo(0).IsName("Anim_oku_hurt")
                   && !animator.GetCurrentAnimatorStateInfo(0).IsName("Anim_oku_fallG")
                   && !animator.GetCurrentAnimatorStateInfo(0).IsName("Anim_oku_hurtG")) {
        PlayAnim("Anim_oku_fall");
    }
}

//Stop falling
if (transform.position.y+yspeed <= ystart) {
    if (!onGround) {
        onGround = true;
        transform.position = new Vector2(transform.position.x, ystart);
        yspeed = 0;
        doubleJump = true;
        PlayAnim("Anim_oku_run");
        bounceCombo = 0;
        bounceTimes = 0;
        Master.instance.PlaySound(sndLanding);
    }
} else {
    onGround = false;
}

//Add speed
transform.position = new Vector2(transform.position.x+(finalspd * 60f * Time.deltaTime), transform.position.y + (yspeed*60f*Time.deltaTime));
```

Kuva 18. Palanen pelaajan Update-funktion sisällöstä.

6.1.2 Esteet ja tavarat

Juoksutasot ovat satunnaisesti generoituja, eli esteitä ja tavaroita tulee vastaan satunnaisesti. Mitä pidemmälle pelaaja pääsee, sitä nopeammin ja tiheämmin niitä myös tulee. Tämä on malliesimerkki pelin vaikeus käyrästä (difficulty curve), eli pelin vaikeutumisesta ajan myötä.

Alkuperäinen idea oli se, että peli valitsisi erilaisia valmiiksi rakennettuja esteasetelmia ja lähettäisi niitä satunnaisesti, mutta tämän idean hylkäsin aikaisin ajan säästämisen takia. Melkein täysi satunnaisuus oli nopeampi toteuttaa kuin kaikkien esteasetelmien erikseen luominen.

Lopullinen esteiden luominen tapahtuu kuvan 19 osoittamalla tavalla. Luominen tapahtuu aina, kun muuttuja enemyInterval saavuttaa nollan tai alle. Luonti pitää huolen siitä, että normaalisti kahta samaa estettä ei voi tulla peräkkäin, mutta mitä pidemmällä ollaan, sitä suurempi mahdollisuus on, että jotkin esteet kloonataan. Kaikki intervalliarvot, kuten minInterval ja maxInterval asetetaan kentän alussa, ja ne riippuvat siitä, kuinka pitkällä pelissä ollaan. Tavaroiden luonti tapahtuu samalla periaatteella, mutta ilman duplikaattivarmistusta.

```
if (enemyInterval <= 0) {
    if (Master.instance.endTimer > 0.075f) {
        //Decide on what to spawn
        int obsIndex = -1;
        while (true) {
            obsIndex = Random.Range(0, maxSpawnIndex + 1);
            if (obsIndex != previousObstacle || obsIndex == 0) break;
        }
        previousObstacle = obsIndex;
        //Spawn
        SpawnSwitch(obsIndex, 0);
        //Check if able to dublicate
        //Only dublicate ninjas and cars
        if (obsIndex == 0 || obsIndex == 1 || obsIndex == 3) {
            //Limit for dube
            int cap = Mathf.Clamp(30 - Master.instance.stage, 5, 30);
            int dube = Random.Range(0, cap + 1);
            if (dube == 0) {
                //Spawn car a bit further away than ninjas
                SpawnSwitch(obsIndex, (obsIndex == 1 ? 1.5f : 1f));
            }
        }

        //Wait for a random time (can be affected with level number later)
        enemyInterval = Random.Range((int)minInterval, (int)maxInterval);
    }
} else {
    enemyInterval-=1f * 60f * Time.deltaTime;
}
```

Kuva 19. Esteiden luonti.

6.1.3 Pomotaso

Joka kymmenes juoksutaso on pomotaso, jossa pelaajan täytyy taistella vastaan tulevaa robottia vastaan. Tason näkee kuvassa 20. Pomotaso kestää normaalista poiketen siihen asti, kunnes pelaaja on tuhonnut robotin. Robotti ampuu ajastimella kahta erilaista hyppivää ammusta. Toinen ammuksista kulkee matalalla, eli sen yli pitää hypätä, ja toinen kulkee korkealla, eli sen ali pitää mennä. Ammuksien tiheys kasvaa myöhemmillä pomotasoilla. Ampumistoistorakenne on esitetty kuvassa 21.

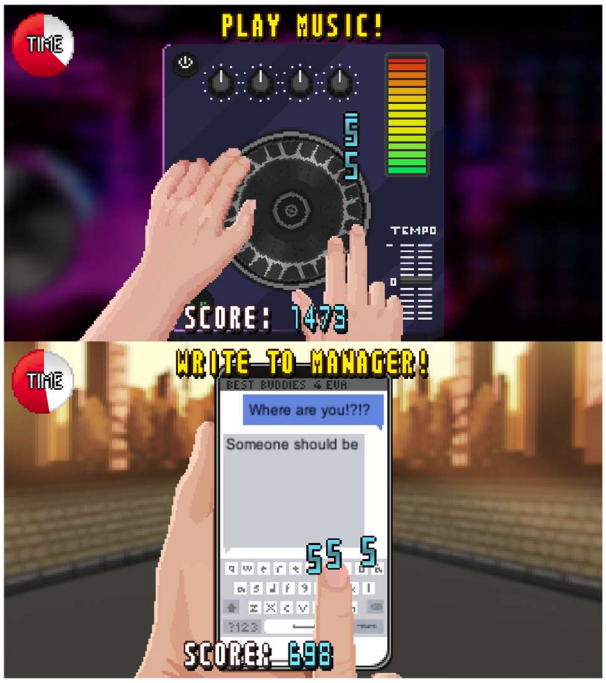
Kuva 20. Pomotaso.

```
private IEnumerator AttackPattern() {
    yield return new WaitForSeconds(1f);

    while (Master.instance.bossHp > 0) {

        Master.instance.PlaySound(sndLoad);
        for (int i=0; i<30; i++) {
            //Telegraph effect
            Effect p = Instantiate(prefabParticle, new Vector2(body.position.x - 0.3f, body.position.y - 0.15f), Quaternion.identity).GetComponent<Effect>();
            p.xspeed = Random.Range(-0.05f, -0.01f);
            p.yspeed = Random.Range(-0.01f, 0.005f);

            yield return null;
        }

        Master.instance.PlaySound(sndShoot);
        Vector3 pos = new Vector3(transform.position.x - 0.2f, transform.position.y - 0.05f, transform.position.z);
        Cannonball cb = Instantiate(prefabCannonball, pos, Quaternion.identity).GetComponentInChildren<Cannonball>();
        int r = Random.Range(0, 3);
        if (r == 0) {
            cb.jumpspeed = 0.02f;
        } else if (r == 1) {
            cb.jumpspeed = 0.032f;
        } else if (r == 2) {
            cb.jumpspeed = 0.06f;
        }

        yield return new WaitForSeconds(2f);
    }
}
```

Kuva 21. Ampumistoistorakenne.

6.2 Minipeli

Jokaisen juoksutason välissä tulee lyhyt minipeli, jossa tarkoituksena on vain kerätä lisäpisteitä. Minipelejä on viisi erilaista, ja seuraavana tuleva arvotaan aina satunnaisesti. Arvonnassa otetaan kuitenkin huomioon se, ettei sama minipeli voi tulla kahta kertaa peräkkäin. Kuvassa 22 on esitetty muutama minipeli.



Kuva 22. Muutama pelissä esiintyvä minipeli.

Kaikki eri minipelit on sijoitettu samaan näkymään, mutta jokainen on oma peliobjektinsa, joka aktivoidaan tai deaktivoidaan sen mukaan, mikä minipeli on valittu. Tästä huolehtii yleismanageri, joka vastaa myös minipelin ajastamisesta. Jokaisella minipelillä on myös oma managerinsa, joka vastaa kyseisen minipelin toiminnasta. Koska minipelit ovat niin yksinkertaisia, oli järkevämpää laittaa ne saman näkymän alle.

6.3 Tietokanta

Okupeli käyttää aiemmin tekstissä esiteltyä Amazonin AWS:n tarjoamaa DynamoDB:tä tietokantatoimintoihin. Tätä käytin lähinnä sen vuoksi, että Nopialla oli kokemusta siitä muiden projektien kautta. Okupeli käyttää pääasiassa vain yhtä taulua,

johon tallennetaan käyttäjien id, käyttäjänimi, pisteet, viimeksi käytetyt vaatteet ja ostot. Näitä arvoja ei käsitellä suoraan koodista, vaan kutsutaan välikätenä olevia Lambda-funktioita. Kutsut tapahtuvat kuvan 23 osoittaman metodin avulla. Ensimmäinen parametri ”obj” on jokin tiettyä lambda-funktiota varten räätälöity luokka (Esimerkki kuvassa 24), joka muutetaan JSON-muotoon ja lähetetään kutsun mukana. Lambda-funktio palauttaa samanlaisen JSON-objektin, johon peli reagoi tietyllä tavalla.

```
//Call for lambda
public void InvokeLambda<T>(T obj, string lambdaName, Action<LambdaResponse> callback) {
    if (Master.instance.oldVersion) {
        callback(null);
    } else {
        var request = new InvokeRequest() {
            FunctionName = lambdaName,
            Payload = JsonConvert.SerializeObject(obj),
            InvocationType = InvocationType.RequestResponse
        };
        LambdaClient.InvokeAsync(request, (result) => {
            if (result.Exception == null) {
                LambdaResponse r = JsonConvert.DeserializeObject<LambdaResponse>(Encoding.ASCII.GetString(result.Response.Payload.ToArray()));
                callback(r);
            } else {
                //Debug.LogError(result.Exception.Message);
                callback(null);
            }
        });
    }
}
```

Kuva 23. Lambda-funktioiden kutsumetodi.

```
public class AwsRequestStartup {
    public string id;
    public string version;
}
public class AwsRequestGameover {
    public string id;
    public string name;
    public int new_score;
    public int new_money;
    public int hatt;
    public int topp;
    public int bott;
}
```

Kuva 24. Lambda-funktiota varten räätälöity luokka.

Kuvassa 25 näkee osan lambda-funktiosta, joka päivittää pelaajan tiedot, kun peli on ohi. Kyseinen lambda-funktio sisältää myös tarkistuksia sille, että vastaanotettu data on oikeellista, mutta niitä ei tässä kuvassa näytetä.

```
let newPlayerParams = {
    TableName: "okugame_users",
    Key: {
        id: event.id
    },
    UpdateExpression: "set #nam = :r, score = :p, #tim = :s, hatt = :ha, topp = :to, bott = :bo",
    ExpressionAttributeValues:{
        ":r":event.name,
        ":p":event.new_score,
        ":s":Date.now(),
        ":ha":event.hatt,
        ":to":event.topp,
        ":bo":event.bott
    },
    ExpressionAttributeNames:{
        "#nam":"name",
        "#tim":"time"
    }
};
//Push
client.update(newPlayerParams, function(err){
   if (err) { Respond(false,err.message); } else { AddMoney(); }
});
```

Kuva 25. Tapa, jolla lambda-funktio lisää pelaajan datan tauluun.

6.4 IAP ja mainokset

Okupelin sisäiset ostokset käyttävät Unityn omia IAP-palveluja, joihin on yhdistetty hieman Lambda-kutsuja pelaajan datan tallentamiseen. Tiettyjen ostojen tallentaminen oli kätevämpää tehdä tietokannan tauluun, koska peli toimii sekä iOS:lla ja Androidilla. Google Play- ja App Store -rajapintoja käytetään pelkästään käyttäjän autentikointiin, eikä kyseisten kauppapaikkojen pilviin tallenneta enempää dataa kuin on pakko.

Unityn IAP toteutetaan siten, että luodaan luokka, joka toteuttaa IStoreListener-rajapinnan ja sen metodit. Rajapinta on generalisoitu niin, että se automaattisesti käyttää Google Playn tai App Storen toimintoja alustasta riippuen, kunhan tuotteille on määritelty samat nimet. Jos ostotapahtuma on onnistunut, kutsutaan Lambda-funktiota, joka hoitaa tiedon päivittämisen tietokantaan.

Mainokset toimivat Unity Ads -palvelun avulla. Loin yhden mainoksista vastaavan luokan, jonka metodeja voi kutsua mistä tahansa, jotta niiden näyttäminen olisi helppoa. Mainoksia on kahdenlaisia: normaali mainos ja palkintomainos. Pelaajalle annetaan elämien loppuessa mahdollisuus katsoa palkintomainos. Jos mainoksen katsoo kokonaan, pelaaja saa yhden lisäelämän. Normaaleja mainoksia näytetään satunnaisesti joka tason välissä. Nämä pelaaja voi ohittaa koska tahansa.

Kuvassa 26 näkyvät tärkeimmät mainosmetodit. Metodit toimivat melkein samalla tavalla, mutta erona on Monetization.IsReady()-testi, jolla tarkistetaan, että mainoksia on tarjolla. Palkintomainoksen tapauksessa mainoksen kutsun yhteydessä välitetään myös metodi, joka reagoi mainoksen tulokseen.

```
//Show a normal add
public void ShowAdSingle() {
    if (Monetization.IsReady(videoad)) {
        ShowAdPlacementContent ad = null;
        ad = Monetization.GetPlacementContent(videoad) as ShowAdPlacementContent;
        if (ad != null) {
            ad.Show();
        }
    }
}

//Show a rewarded ad
public void ShowAdRewarded() {
    if (Monetization.IsReady(rewardedad)) {
        ShowAdPlacementContent ad = null;
        ad = Monetization.GetPlacementContent(rewardedad) as ShowAdPlacementContent;
        if (ad != null) {
            ad.Show(DidRewardedAdFinish);
        }
    }
}

public void DidRewardedAdFinish(ShowResult result) {
    switch (result) {
        case ShowResult.Finished: Master.instance.Respawn(); break;
        case ShowResult.Skipped: Master.instance.AdSkipped(); break;
        case ShowResult.Failed: Master.instance.AdSkipped(); break;
    }
}
```

Kuva 26. Mainosten näyttämiseen tarkoitetut metodit ja niiden toiminta.

6.5 Julkaisu ja ylläpito

Viikot ennen Okupelin julkaisua ovat yleensä kehittämisen hektisimmät vaiheet. Tämän projektin kanssa asia ei onneksi ollut näin, koska minulla oli ollut hyvin aikaa viimeistellä kaikkia pelin osa-alueita ennen julkaisua. Okupeli julkaistiin Google Playssa ja App Storessa. Kumpaakin julkaisua varten peliä piti testata.

Google Playssa julkaisuprosessi oli vaivattomampi. Peli ladattiin palveluun, jossa pystyi määrittelemään testaajien sähköpostiosoitteet. Näin testaajat pystyivät lataamaan

pelin suoraan Google Playsta. App Storessa julkaiseminen ja testaaminen oli hankalampaa. Peli pitää kääntää Applen XCode-ohjelmalla, jonka jälkeen se pitää testata jollakin iOS-laitteella. Tämän jälkeen pitää odottaa muutama päivä ennen julkaisua, koska Apple tarkistaa pelin manuaalisesti, jottei se sisällä laittomuuksia. Pelin julkaisu sujui kokonaisuudessaan melkein moitteettomasti.

Okupelin ylläpitäminen koostui lähinnä muutamista päivityksistä ja bugien korjaamisesta. Okupelin ensimmäinen versio ei edes sisältänyt vielä IAP:eja, ja käytti tästä syystä vähemmän lambdoja. Suuren osan tietokannan hyödyntämisestä implementoin vasta päivityksessä, joka tuli viikkoja julkaisun jälkeen. Voisi siis sanoa, että ensimmäisenä julkaistu versio oli puutteellinen verrattuna siihen, millainen peli on nyt.

7 LOPUKSI

Pelinkehittämisprosessi on hyvin laaja ja monipuolinen. Olin tämän projektin ohjelmoija, mutta kaiken muun, kuten grafiikat, musiikit, äänitehosteet, testaamisen, markkinoinnin hoitivat pääasiassa muut. Tämä opinnäytetyökään ei kata kaikkea, koska jokaisen pienen yksityiskohdan ja logiikan läpi käyminen ei olisi mielekästä. Tämän takia halusin paneutua pääasioihin ja yrittää antaa yleiskuvan siitä, millainen prosessi oli.

Tämä oli ensimmäinen virallisesti julkaistu peli, jonka tein alusta loppuun asti itse. Minulla oli jo ennestään paljon kokemusta Unityn käytöstä, mutta opin silti hyvin paljon uutta. Opin erityisesti käyttämään ja integroimaan lisäosia, kuten AWS, Unity IAP ja Unity Ads. Voisi siis sanoa, että sain paljon kokemusta ”backend”-kehittämisestä ja pelin tietokantaintegraatiosta. Pääsin myös kokemaan, millainen on mobiilipelien julkaisuprosessi käytännössä. Oma johtopäätökseni on se, että Androidille kehittäminen on paljon vaivattomampaa kuin iOS:lle kehittäminen. Välttääkseen mahdollisesti turhaa vaivaa, kannattaa peli ensin julkaista Androidilla ja sitten vasta kääntää iOS:lle, jos mahdollista.

Omasta mielestäni suoriuduin projektista hyvin. Näin jälkeenpäin katsottuna koodi alkoi muistuttaa sitä enemmän viidakkoa, mitä pidemmälle se eteni, mutta tämä on kai tavallista tällaisten projektien kanssa. Seuraavalla kerralla osaan ennakoida ja yrittää varautua paremmin. Tämä ei ollut ensimmäinen pelini, eikä se myöskään jää viimeisekseni.

LÄHTEET

Amazon.com 2020a. What is AWS? Viitattu 30.3.2020. https://aws.amazon.com/what-is-aws/

Amazon.com 2020b. AWS DynamoDB. Viitattu 30.3.2020. https://docs.aws.amazon.com/mobile/sdkforunity/developerguide/dynamodb.html

Amazon.com 2020c. AWS Lambda. Viitatty 30.3.2020. https://docs.aws.amazon.com/mobile/sdkforunity/developerguide/lambda.html

Callaham, J. 2017. From Android Market to Google Play: a brief history of the Play Store. Viitattu 18.1.2020. https://www.androidauthority.com/android-market-google-play-history-754989/

Crews, E. 2016. A Short History of Mobile Games. Viitattu 18.1.2020. https://geekinsider.com/short-history-mobile-games/

GameFromScratch.com 2015. Gamedev Glossary: Library Vs Framework Vs Engine. Viitattu 25.1.2020. https://www.gamefromscratch.com/post/2015/06/13/GameDev-Glossary-Library-Vs-Framework-Vs-Engine.aspx

Github.com 2020. Google Play Games plugin for Unity. Viitattu 31.3.2020. https://github.com/playgameservices/play-games-plugin-for-unity

Iqbal, M. 2019. App Download and Usage Statistics (2019). Viitattu 18.1.2020. https://www.businessofapps.com/data/app-statistics/

Koistila, P. 2014. Game monetization design: Analysis of Clash of Clans. Viitattu 20.1.2020. https://www.gamasutra.com/blogs/PeteKoistila/20140415/215503/Game_monetization_design_Analysis_of_Clash_of_Clans.php

Markoff, J. & Holson, L. 2008. Apple's Latest Opens a Developer's Playground. Viitattu 18.1.2020. https://www.nytimes.com/2008/07/10/technology/personaltech/10apps.html?_r=5&ref=technology

MonoGame.net 2020a. MonoGame. Viitattu 25.1.2020. http://www.monogame.net/

MonoGame.net 2020b. Showcase. Viitattu 25.1.2020. http://www.monogame.net/showcase/

Nelson, R. 2019a 'Global App Revenue Grew 23% in 2018 to More Than $71 Billion on iOS and Google Play'. Sensor Tower Blog. 16.1.2019. Viitattu 18.1.2020. https://sensortower.com/blog/app-revenue-and-downloads-2018

Nelson, R. 2019b 'The Top Mobile Apps, Games, and Publishers of 2018: Sensor Tower's Data Digest'. Sensor Tower Blog. 16.1.2019. Viitattu 18.1.2020. https://sensortower.com/blog/top-apps-games-publishers-2018

Newtonsoft.com 2020. Introduction. Viitattu 31.3.2020. https://www.newtonsoft.com/json/help/html/Introduction.htm

Redcarpetrampage.com 2020. Roni's Red Carpet Rampage. Viitattu 2.4.2020. http://redcarpetrampage.com/

Schell, J. 2015. The Art of Game Design A Book of Lenses. 2. uud. p. Florida. CRC Press.

Unity.com 2020a. Unity Core Platform. Viitattu 25.1.2020. https://unity.com/products/core-platform

Unity.com 2020b. Made With Unity. Viitattu 25.1.2020. https://unity.com/madewith

Unity3d.com 2020a. 2D or 3D projects. Viitattu 26.1.2020. https://docs.unity3d.com/Manual/2Dor3D.html

Unity3d.com 2020b. Learning the interface. Viitattu 26.1.2020. https://docs.unity3d.com/Manual/LearningtheInterface.html

Unity3d.com 2020c. Importing. Viitattu 26.1.2020. https://docs.unity3d.com/Manual/ImportingAssets.html

Unity3d.com 2020d. GameObjects. Viitattu 26.1.2020. https://docs.unity3d.com/Manual/GameObjects.html

Unity3d.com 2020e. Scripting. Viitattu 26.1.2020. https://docs.unity3d.com/Manual/ScriptingSection.html

Unity3d.com 2020f. Using Components. Viitattu 26.1.2020. https://docs.unity3d.com/Manual/UsingComponents.html

Unity3d.com 2020g. Canvas. Viitattu 30.1.2020. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html

Unity3d.com 2020h. Designing UI For Multiple Resolutions. Viitattu 30.1.2020. https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/HOWTO-UIMultiResolution.html

Unity3d.com 2020i. Setting Up Your Project For Unity Services. Viitattu 31.1.2020. https://docs.unity3d.com/Manual/SettingUpProjectServices.html

Unity3d.com 2020j. Integration Guide for Unity. Viitattu 31.1.2020. https://unityads.unity3d.com/help/unity/integration-guide-unity#basic-implementation

Unity3d.com 2020k. Initialization. Viitattu 31.1.2020. https://docs.unity3d.com/Manual/UnityIAPInitialization.html

Unity3d.com 2020l. Configuring for Apple App Store and Mac App Store. Viitattu 31.1.2020. https://docs.unity3d.com/Manual/UnityIAPAppleConfiguration.html

Unity3d.com 2020m.Configuring for Google Play Store. Viitattu 31.1.2020. https://docs.unity3d.com/Manual/UnityIAPGoogleConfiguration.html

YoYo Games.com 2020a. Game Maker Studio 2. Viitattu 25.1.2020. https://www.yoyogames.com/gamemaker

YoYo Games.com 2020b. Game Maker Studio 2. Viitattu 25.1.2020. https://www.yoyogames.com/gamemaker/features

YoYo Games.com 2020c. Made With Game Maker Showcase. Viittattu 25.1.2020. https://www.yoyogames.com/showcase