Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Iiro Kiema

Hyperkasuaalipelin julkaisu ja analysointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

13.5.2020

Tekijä Otsikko Sivumäärä Aika	Iiro Kiema Hyperkasuaalipelin julkaisu ja analysointi 35 sivua + 3 liitettä 13.5.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Pelisovellukset
Ohjaaja	Lehtori Miikka Mäki-Uuro

Insinöörityössä oli tarkoituksena luoda hyperkasuaalipeli iOS-alustalle ja integroida siihen analytiikkapalvelu. Hyperkasuaalipelit ovat mobiilipelejä, joissa on tyypillisesti yksinkertaiset pelimekaniikat ja pelaamisen voi aloittaa nopeasti. Lisäksi insinöörityössä oli tarkoitus saada peli Applen TestFlight-palvelun kautta testattavaksi vähintään yhdelle puhelimelle. Pelinkehityksen oppiminen, ja oppimisprosessin tarkkailu, olivat myös yksi insinöörityön tavoitteista. Työn oli tarkoitus olla mahdollisimman kattava paketti eri vaiheista, joita pelinkehitys ja julkaisu vaativat.

Unity-pelimoottorin avulla luotiin peli, jossa on alkuvalikko ja päättymätön päätaso. Päätasossa vihollisia ilmestyi aalloittain satunnaisesti pelaajaa kohti. Vihollisten nopeus ja elämäpisteiden määrä vaihtelivat vihollistyypin mukaan. Jos vihollinen osui pelaajaan, pelaajan elämäpisteet vähenivät. Kun pelaajan elämäpisteiden määrä laski nollaan, peli päättyi. Jos pelaaja sai tuhottua kaikki viholliset, alkoi uusi taso. Pelin vaikeusaste nousi jatkuvasti. Peliin liitettiin myös ääniefektit, jotka olivat ilmaisia ja vapaaseen käyttöön suunnattuja.

Pelistä tehtiin ensin Unityssä koonti (engl. build), ja sitten se lähetettiin Xcoden avulla App Store Connectiin. Siellä koonti annettiin Applen arvioitavaksi tarkoituksena saada se TestFlight-palvelun kautta jaettavaksi testaajille. Julkaisuprosessi App Storeen käytiin teoriatasolla läpi hyödyntäen Applen omia ohjeita ja aiempia kokemuksia App Storeen julkaisusta.

Lopputuloksena peli saatiin onnistuneesti TestFlight-ympäristöön testattavaksi. Peli läpäisi siis Applen vaatimukset TestFlightin osalta. Lisäksi Game Analytics -palvelu saatiin onnistuneesti otettua käyttöön pelissä, ja sen avulla saatiin luettua pelin toiminnan avainlukuja. Pelille ei saatu hankittua montaa testaajaa, joten kunnollisten johtopäätösten tekeminen avainlukujen perusteella oli mahdotonta. Avainlukujen arvoja kuitenkin vertailtiin Game Analyticsin omaan raporttiin vuodelta 2019. Ideana oli saada kokemusta siitä, kuinka lukuja vertaamalla voi vetää johtopäätöksiä pelistä. Tultiin siihen johtopäätökseen, että peliä pitäisi muuttaa niin, että se säilyttää pelaajan mielenkiinnon pidempään.

Avainsanat	iOS, julkaisu, pelianalytiikka, App Store Connect, TestFlight, App Store

Author Title Number of Pages Date	Iiro Kiema Release and analysis of a hyper-casual game 35 pages + 3 appendices 13 May 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Game Applications
Instructor	Miikka Mäki-Uuro, Senior Lecturer

The goal of this thesis was to create a hyper-casual iOS game and to have an analytics service integrated into it. Hyper-casual games are mobile games that typically have simple mechanics and the player can quickly start playing. Another goal of this thesis was that the game should be tested on at least one phone through Apple's TestFlight service. The learning process of game development was also to be observed while working. This thesis was intended to be a thorough package of everything related to creating and publishing an iOS game.

Unity game engine was used to develop a game that had a start menu and an endless level. Enemies spawned in waves and approached the player from random directions. Health amount and movement speed varied based on enemy type. If an enemy reached the player, then the player took damage. Once player's health amount dropped to zero, the game ended. If the player managed to defeat all enemies, then the next level was initiated. In-game difficulty was constantly increasing as the game progressed. Sound effects were also added to the game. They were free and did not require any royalties. A build was made with Unity and opened in Xcode. It was then archived and sent to App Store Connect. At this point the build was put through Apple's review in order to send it to TestFlight. The process of submitting a build into App Store was discussed in theory based on information found on Apple's website and also from the author's previous experiences of submitting an app into App Store.

As a result, the game was successfully shared with players using TestFlight meaning it cleared Apple's TestFlight requirements. Game Analytics service was successfully integrated into the game and it was possible to analyze KPI (Key Performance Indicator) values with it. Unfortunately, there were not enough testers and consequently analyzing KPI values properly was not possible. They were still analyzed using Game Analytics' own report from 2019 as a reference, as it was deemed a good learning experience for trying to draw conclusions from the KPI values. The conclusion was that the game should be improved in order to keep the player interested for longer.

Keywords	iOS, publishing, Game Analytics, App Store Connect, Test Flight, App Store

Sisällys

1 Johdanto

Insinöörityön tavoitteena on käsitellä mahdollisimman kattavasti iOS-alustalle kehitettävän hyperkasuaalipelin julkaisuprosessia. Hyperkasuaalipeli eroaa muista mobiilipeleistä siten, että siinä on yksinkertaiset mekaniikat ja pelaamisen voi aloittaa nopeasti. Analytiikkapalveluksi valittiin Game Analytics, koska sen avulla saa helposti luettavassa muodossa informaatiota pelin toiminnasta.

Insinöörityötä aloittaessa oli jo tiedossa, että sen aikana tehtyä peliä ei oltaisi vielä julkaisemassa App Storeen. App Store -julkaisu käydään silti teoriatasolla läpi, koska se on niin olennainen osa julkaisuprosessin kokonaisuutta. Tiedot julkaisuprosessista raportoimiseen perustuvat kokemukseen toisesta pelistä, jonka julkaisussa App Storeen vuonna 2019 oli oltu mukana. Idea on siinä, että julkaisuprosessi tulisi käytyä kokonaisuudessaan läpi alusta loppuun.

Luvussa 2 käydään läpi, mistä insinöörityön peli sai ideansa, ja myös pelin kehitysprosessia mukaan lukien, mitä osia se pitää sisällään. Lisäksi käydään teoriatasolla läpi monetisaatiota ja järkeviä tapoja toteuttaa se hyperkasuaalipeleissä. Koska monetisaatio on tärkeä osa hyperkasuaalipelin julkaisuprosessia, tämä on syytä käydä läpi, jotta hahmotetaan paremmin sen tarkoitus ja osuus projektiin nähden.

Luvussa 3 perehdytään siihen, mikä on TestFlight, kuinka saadaan koonti (engl. build) TestFlightiin, ja mitä asioita siinä pitää ottaa huomioon. Luvussa 4 käsitellään aluksi pelianalytiikkaa yleisellä tasolla. Tämän jälkeen tarkkaillaan Game Analytics -palvelusta nähtyjä avainlukuja pelistä ja pohditaan, mitä johtopäätöksiä niistä voi vetää.

Luvussa 5 käsitellään pelin teoreettista julkaisua App Storeen ja myös käydään yleisellä tasolla läpi eri vaatimuksia, joita Applella on App Store -sovellusten suhteen. Lisäksi käydään läpi eri ongelmatilanteita, joita voi tulla App Storen tai App Store Connectin kanssa, ja kuinka niistä selvittiin.

Lopuksi luvussa 6 käydään läpi työn tuloksia ja pohditaan, mitä johtopäätöksiä työn tuloksista voi vetää ja kuinka projekti toimi oppimiskokemuksena.

2 Pelin luominen Unity-pelimoottorilla

Tässä luvussa keskitytään itse pelinkehitykseen Unity-pelimoottorilla. Ohjelmointiympäristönä käytettiin JetBrains Rideria [1]. Versionhallintaan käytettiin GitHub Desktop -sovellusta [2]. Aikataulun ja tehtävien hallinnointiin käytettiin Trelloa [3]. Lopuksi käydään myös teoriatasolla läpi monetisaatiota, vaikka se ei projektin tavoitteisiin tai toteutukseen sisältynytkään. Näin hahmotetaan paremmin sen tarkoitus ja osuus projektiin nähden.

2.1 Peli-idea ja referenssipeli

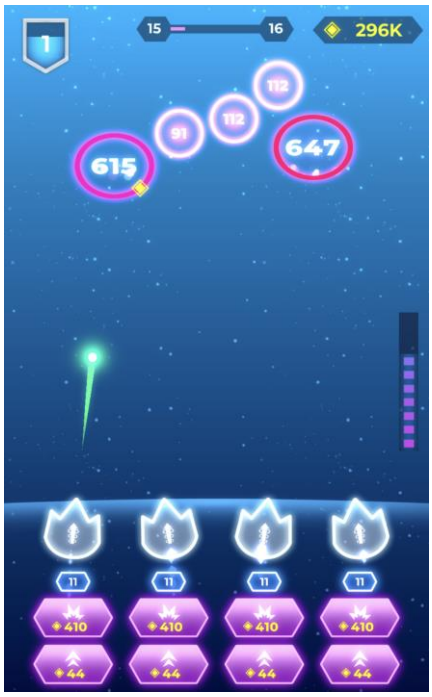
Pelin genreksi valittiin hyperkasuaali (engl. hyper-casual). Kuten mainosteknologiayritys ironSource kertoo blogissaan [4], hyperkasuaalipeleissä on yksinkertaiset mekaniikat ja käyttöliittymä, minkä seurauksena pelin voi aloittaa helposti ja nopeasti. Lisäksi menestyneistä peleistä, kuten esimerkiksi Stack Ball [5], havaitaan, että peli voi menestyä ilman juonta tai näyttävää visuaalista antia. Stack Ball sijoittui korkealle ladatuimpien pelien listalla vuonna 2019 [6].

PubNative käy blogissaan [7] läpi hyperkasuaalipelien kohderyhmää, joita ovat kasuaalit (engl. casual) pelaajat. Heitä ei kiinnosta käyttää tunteja pelin oppimiseen tai keskittymiseen pelaamisen aikana. Blogin mukaan hyperkasuaalipelejä pelataan enimmäkseen työmatkoilla, lounastauoilla ja jopa wc:ssä. Ihmiset pitävät niistä, koska niitä voi pelata missä vain. Suurin osa kasuaalipelaajista ei koe olevansa pelaajia. He myös haluavat pelien olevan ilmaisia. Heitä ei kuitenkaan häiritse katsoa mainoksia saadakseen pelin sisäisiä palkintoja, blogissa todetaan.

Edellä mainittujen syiden perusteella hyperkasuaalipeli soveltuu hyvin tähän insinöörityöhön, koska näin saadaan työmäärältään sopivan haastava peli siten, että sen tekemiseen ei kulu liikaa aikaa.

Referenssiksi valittiin Non-Stop Space Defense iOS-alustalle [8]. Tässä pelissä vihollisia tulee aalloittain ruutua alas, ja niitä näpäyttämällä tykit ampuvat vihollista päin. Tykit ampuvat jonkin verran myös automaattisesti, mutta se ei riitä tasojen läpäisemiseen. Viholliset voivat ilmestyä missä tahansa kohdassa ruudun yläosaa. Kun pelaaja on selvittänyt

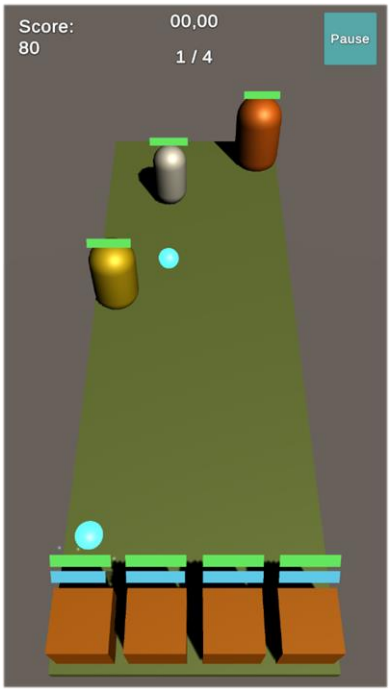
tietyn määrän aaltoja, tulee pomotaistelu. Kuvassa 1 on nähtävissä peli Non-Stop Space Defense.



Kuva 1. Non-Stop Space Defense [8].

2.2 Pelimekaniikka ja -progressio

Projektin peliä suunniteltaessa päätettiin, että viholliset tulevat neljässä eri linjassa näyttöruutua ylhäältä alaspäin ja näytön alaosassa olevia tykkejä näpäyttämällä pelaaja ampuu kyseisellä tykillä ammuksen kohti sen linjan vihollista. Tykeillä on energiaa, joka hupenee jokaisen ampumisen yhteydessä. Jos energiaa ei ole tarpeeksi, ampuminen ei onnistu. Kuvassa 2 on otettu pelistä ajonaikana kuvakaappaus, josta nähdään edellä mainitut ominaisuudet.



Kuva 2. Ajonaikainen näkymä pelistä, jossa tykeillä on ammuttu kohti lähestyviä vihollisia.

Vihollisia tulee aalloissa, ja jokaisen aallon välissä on pieni viive. Kun pelaaja on selvittänyt kaikki aallot, taso päättyy. Näkyviin tulee ikkuna, jossa saa tähtiä yhdestä neljään riippuen siitä, kuinka paljon tykkien elämäpisteet vähenivät tason aikana. Jokaisen tason jälkeen viholliset muuttuvat hieman vahvemmiksi. Peli päättyy, jos minkä tahansa tykin elämäpisteiden määrä laskee nollaan. Pelin päättyessä pelaaja näkee mahdollisen edellisen yrityksen pistemääränsä ja nykyisen pistemäärän.

2.3 Unity-pelimoottorin käyttäminen

Yleisesti ottaen Unityä käytetään niin, että ensin luodaan tietynlaisia näkymätiedostoja (engl. scene), ja näihin näkymiin asetetaan peliobjekteja (engl. GameObject). Peliobjekteihin voi kiinnittää skriptejä tai komponentteja. Skriptit ovat tietynlaisia tekstitiedostoja, joihin ohjelmoidaan haluttuja toiminnallisuuksia käyttäen C#-kieltä. Sen jälkeen skripti kiinnitetään peliobjektiin, jolloin kyseiselle objektille tapahtuu skriptin mukaisia asioita.

Toisaalta skriptiin voi myös antaa referenssinä toisen peliobjektin, jolloin voi esimerkiksi liikuttaa tätä toista objektia, vaikka skripti ei olekaan kiinni siinä. Skripteillä voi tehdä hyvin kattavasti erilaisia asioita, ja yleensä rajoitteena ovatkin vain kehittäjän omat ohjelmointitaidot.

Tyypillisiä komponentteja ovat esimerkiksi partikkelijärjestelmä (engl. Particle System), jolla luodaan visuaalisia efektejä peliin, äänilähde (engl. Audio Source), Collider, joka tarkkailee peliobjektin törmäyksiä, ja Rigidbody, jolla voi säädellä peliobjektin massaa tai siihen vaikuttavia voimia.

2.4 Pelin tekninen toteutus

Pelin yleistä hallinnointia varten tehtiin esimerkkikoodin 1 mukainen singleton-tyyppinen [9] päämanageriluokka, joka säilytti referenssejä muihin alimanageriluokkiin. Tämän yhden pääluokan kautta päästiin siten kätevästi käsiksi muihin tarvittaviin luokkiin.

```
public class GameManager : MonoBehaviour
{
    public static GameManager Instance;

    public ScoreManager ScoreManager { get; private set; }
    public GameUIStateManager GameUIStateManager { get; private set; }
    public SceneHandler SceneHandler { get; private set; }
    public PlayerProgression PlayerProgression { get; private set; }

    private void Awake ()
    {
        if (!Instance)
        {
            Instance = this;
        }

        ScoreManager = GetComponent<ScoreManager> ();
        GameUIStateManager = GetComponent<GameUIStateManager> ();
        SceneHandler = GetComponent<SceneHandler> ();
        PlayerProgression = GetComponent<PlayerProgression> ();
    }

    private void OnDestroy ()
    {
        Instance = null;
    }
}
```

Esimerkkikoodi 1. GameManager-skripti, joka hallinnoi muita manageriluokkia.

Huomionarvoista lienee kuitenkin, että singleton-mallia ei nykyisin suositeltaisi käytettävän syistä, joita Robert Nystrom luettelee Game Programming Patterns -teoksessaan [9]. Hänen mukaansa esimerkiksi staattinen luokka olisi mahdollisesti ollut parempi vaihtoehto tällaisessa tilanteessa.

Tykin ampumista varten tehtiin oma skriptinsä nimeltä CannonSelector (liite 1). Sen toimintaperiaate oli, että pisteestä, jossa sormi osui näyttöön, lähetettiin säde alaspäin, ja jos säde osui johonkin objektiin, tarkastettiin, onko kyseessä jokin neljästä tykistä. Jos oli, niin heti kun pelaaja päästi sormen irti ruudusta, kyseinen tykki ampui ammuksen. Jos pelaaja ei nostanut sormea vaan jatkoi sen pitämistä ruudulla, tykki alkoi ladata superammusta. Kun lataus valmistui, ponnahdusteksti kertoi pelaajalle asiasta. Kun pelaaja lopulta päästi sormen irti, tykki ampui superammuksen.

Kun pelaaja läpäisi tason, uudessa tasossa ei muuttunut visuaalisesti mikään muu, kuin että viholliset olivat vahvempia ja vihollisaaltoja tuli useampia. Tasot toteutettiin siten, että jokainen uusi taso ei olekaan näkymä, vaan yhdessä ainoassa näkymässä pidetään kirjaa siitä, monesko taso on teoriassa menossa. Kun peli päättyy, arvot nollataan ja sama näkymä ladataan uudelleen.

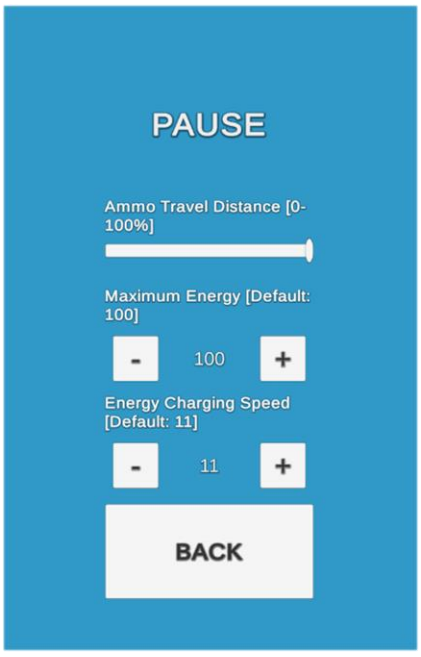
Pelin suorituskyvyn optimointiin käytettiin object pool -tekniikkaa [10], kun luotiin ammuksia ja vihollisia. Tässä tekniikassa peliolioita ei koskaan tuhota vaan ne ainoastaan laitetaan pois päältä, ja myöhemmin niitä voidaan käyttää uudelleen vaihtamalla niiden sijaintia ja laittamalla ne uudelleen päälle. Tällä tavalla kertyy vähemmän työtä C#-kielen jätteenkeräystoiminnolle (engl. garbage collection), jolloin saadaan kasvatettua ruudunpäivitysnopeutta. Skriptistä Enemy1ObjectPooler (liite 2) nähdään, millainen object pool -skriptin toteutus on ensimmäiselle vihollistyypille tässä pelissä. Skripti pohjautui raywenderlich-sivuston [11] ohjeeseen.

2.5 Käyttöliittymä

Pelin käyttöliittymänä toimivat päävalikon ikkunat ja päänäkymän käyttöliittymät. Päävalikossa (kuva 3) on ainoastaan painikkeet pelaamisen aloittamiseen, asetusikkunaan ja pelin lopettamiseen.

Kuva 3. Pelin päävalikko.

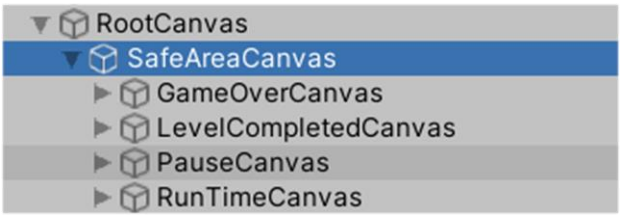
Päänäkymän käyttöliittymänä toimivat ajonaikainen valikko, taukovalikko ja tason päättymisen sekä pelin päättymisen käyttöliittymä. Peliin päätettiin myös tehdä kehittämisen nopeuttamista varten oma säätövalikko, jota pelissä kutsuttiin nimellä Debug Menu (kuva 4).



Kuva 4. Debug Menu, jossa voi pelin aikana kokeilla, miltä eri parametrien arvot tuntuvat pelatessa.

Käyttöliittymiä tehdessä käytettiin apuna Unityn Asset Storesta ladattua ohjelmistokehityspakettia (engl. Software Development Kit) nimeltä Safe Area Helper [12]. Tämä ohjelmistokehityspaketti piti huolen siitä, että esimerkiksi puhelimen näytön pyöreät reunat ja kameraa varten tehty pykälä (engl. notch) eivät peitä käyttöliittymän näkyvyyttä.

Safe Area Helper vaati toimiakseen, että luodaan ensin juurikanvaasi (engl. Root Canvas) ja sen lapseksi toinen turva-aluekanvaasi, johon kiinnitetään SafeArea-skripti. Tälle kanvaasille annettiin projektissa nimeksi SafeAreaCanvas (kuva 5). On tärkeää, että SafeAreaCanvas ja kaikkien sen alle tulevien kanvaasien mittakaava (engl. Scale) on 1.



Kuva 5. Turva-alueelle luotu oma kanvaasi ja sen hierarkia näyttämössä.

On myös mahdollista testata ajonaikaisesti, miltä turva-alueet näyttävät tietyillä eri laitteilla. Tämä onnistuu siten, että lisätään SafeAreaDemo-skripti johonkin peliobjektiin senhetkisessä näkymässä. Kun ajon aikana painaa A-kirjainta, näkee, miltä käyttöliittymä näyttää eri laitteilla. Tämän avulla voi nopeasti kokeilla, millä tavoin käyttöliittymän eri osat kannattaa sijoittaa näytölle, jotta ne näyttävät järkeviltä useilla eri laitteilla.

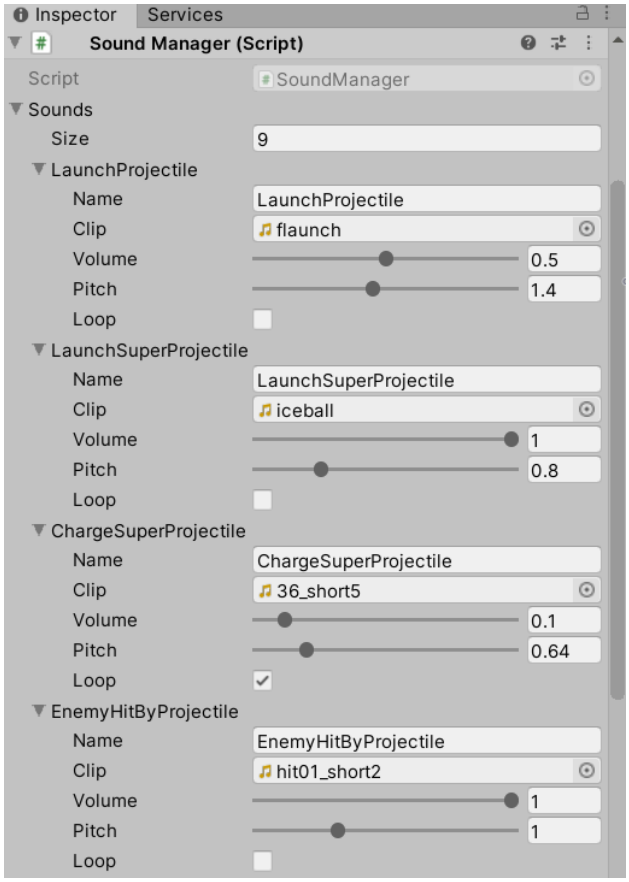
2.6 Äänet

Pelin ääniefekteinä käytettiin OpenGameArt-sivustolta [13] löytyviä ilmaisia ääniefektejä. Niiden hallitsemisesta varten tehtiin SoundManager-niminen skripti (liite 3), joka pohjautui Brackeys-YouTube-kanavan ohjevideoon [14].

Kun manageriluokat saatiin valmiiksi, alettiin editoida äänitiedostoja paremmin peliin sopivaksi. Huomattiin, että jos äänitiedostoja käytti sellaisenaan pelissä, niin esimerkiksi ammuksen osuessa viholliseen ja siitä havaittavan äänen välillä oli häiritsevän pitkä

viive. Tätä yritettiin korjata Audacity-ohjelman [15] avulla leikkaamalla tyhjää osaa pois äänitteen alusta ja tarvittaessa käyttämällä Fade-In -efektiä, mikäli tiedoston alussa kuului säröytynyttä ääntä. Kuitenkin esimerkiksi ampumisen äänessä tämä toimenpide ei kunnolla onnistunut, joten siihen jäi viivettä.

Unityssä jatkettiin äänitiedostojen hiomista säätämällä niiden parametreja kuvan 6 mukaisesti siten, että ne kuulostivat pelissä järkeviltä.



Kuva 6. Äänitiedostojen säätäminen Unityssä.

Esimerkkikoodissa 2 on Sound-luokka, josta näkee, miten parametrit toteutettiin skriptissä.

```
using UnityEngine;

[System.Serializable]
public class Sound
{
    public string Name;
    public AudioClip Clip;
    [Range(0f, 1f)]
    public float Volume;
    [Range(.1f, 3f)]
    public float Pitch;
    [HideInInspector]
    public AudioSource Source;
    public bool Loop;
}
```

Esimerkkikoodi 2. Sound-luokka.

2.7 Analytiikka

Analytiikkapalveluna päätettiin käyttää Game Analyticsiä [16]. Sen käyttöönotto tapahtui siten, että ensin ladattiin Game Analyticsin ohjelmistokehityspaketti Unityn Asset Storesta [17]. Sen jälkeen seurattiin palvelun omia asennusohjeita sivustolla [18]. Ohjeisiin sisältyivät muun muassa seuraavat toimenpiteet:

- ohjelmistokehityspaketin tuonti (engl. Import SDK)
- asennuksen onnistumisen varmistaminen
- tilin luonti ja sisäänkirjautuminen
- alustus (engl. Initialization)
- GameAnalytics-peliobjektin luominen.

Game Analyticsin toimintaperiaate on, että lähetetään tapahtumia (engl. event) palvelimelle sellaisissa pelitilanteissa, joita halutaan myöhemmin tarkkailla. On monia erityyppisiä tapahtumia, joita Game Analyticsin palvelimille voi lähettää. Niitä ovat muun muassa Business-, Ad-, Resource-, Progression-, Error- ja Design-tapahtumat. Tässä pelissä käytettiin ainoastaan Progression- ja Design-tapahtumia.

Pelin etenemisen seuraamista varten käytettiin Progression-tapahtumia (engl. Progression Event), joista yksi tapaus on nähtävissä esimerkkikoodissa 3.

```
public void IncrementCurrentLevel()
{
      currentLevel += 1;
      PlayerPrefs.SetInt ("CurrentLevel", currentLevel);
      PlayerPrefs.Save ();
      GameAnalytics.NewProgressionEvent (GAProgressionStatus.Start,
      $"Level_{currentLevel}");
}
```

Esimerkkikoodi 3. Nykyisen tason kasvattamismetodi, jossa on Progression-tapahtuma mukana.

Käyttöliittymän painikkeiden tarkkailua varten käytettiin Design-tapahtumia (engl. Design Event), jotka ovat nähtävissä esimerkkikoodissa 4.

```
public class MainMenuUIAnalytics : MonoBehaviour
{
      public void PlayButtonClicked()
      {
            GameAnalytics.NewDesignEvent ("Play_Button");
      }

      public void SettingsButtonClicked()
      {
            GameAnalytics.NewDesignEvent ("Settings_Button");
      }

      public void QuitButtonClicked()
      {
            GameAnalytics.NewDesignEvent ("Quit_Button");
      }
}
```

Esimerkkikoodi 4. Päävalikon analytiikkaa varten tehty luokka, jossa käytetään Design-tapahtumia.

Game Analytics vaati toimiakseen, että pelin ensimmäiseen näkymään lisättiin GameAnalytics-peliobjekti valikosta Window/GameAnalytics/Create GameAnalytics Object. Tässä projektissa se lisättiin MainMenu-nimisen näyttämön lisäksi myös EndlessLevel-näyttämöön, koska tällä tavoin pystyttiin editorissa näkemään konsolista, toimiiko analytiikka myös tilanteessa, jossa peliä ei aloitettu ensimmäisestä näkymästä.

2.8 Monetisaatio

Insinöörityössä tehtyyn peliin ei lisätty monetisaatiota, mutta sitä käydään nyt hieman läpi teoriatasolla. Monetisaatio tarkoittaa sitä, kuinka kehittäjä ansaitsee rahaa pelin

avulla. Hyperkasuaalipelissä monetisaation muotona ovat usein mainokset ja suuri määrä käyttäjiä, Game Analyticsin blogista [19] käy ilmi. Myös ristiinmainostaminen (engl. Cross Promoting) on tärkeää siksi, että saadaan kasvatettua pelin latausten lukumäärää. Muiden pelien mainostaminen ei tässä tapauksessa ole haitaksi kehittäjälle, koska tavallisesti pelaajat pelaavat useita pelejä samaan aikaan.

Mainosten täytyy toimia sulavasti pelin kanssa, eivätkä ne saa heikentää käyttäjäkokemusta, blogissa mainitaan. On muutamia eri keinoja, joilla mainoksia voidaan näyttää pelaajille ilman, että he turhautuvat. Yksi tapa on näyttää pelattavia mainoksia. Toinen on esimerkiksi antaa pelaajalle uusi yritys pelin päättymisen jälkeen, jos hän suostuu katsomaan mainoksen. Kolmas tapa, blogin mukaan, on luoda mainoksia pelin sisään natiivisti, jolloin mainos on osa peliä.

Yksi esimerkki, miten mainoksia olisi voitu lisätä insinöörityön peliin, olisi ollut luoda jokin keino kehittää pelaajan tykkejä vahvemmiksi. Maksutapana olisi sitten ollut mainosten katsominen. Kuitenkin menestyvän hyperkasuaalipelin luontiin pitäisi keksiä paljon enemmän tilanteita, joissa hyödyntää mainontaa. Muutoin peli ei tuota tarpeeksi rahaa pelinkehittäjälle.

Suosittuja mainospalveluiden tarjoajia ovat esimerkiksi ironSource [20] ja Unity Ads [21]. Tässä projektissa Unity Ads olisi voinut olla järkevä vaihtoehto, koska se olisi integroitunut hyvin Unity-pelimoottorin kanssa.

3 TestFlight-koonnin tekeminen

Tässä luvussa käydään läpi Applen TestFlight [22] -toiminnallisuuteen liittyviä asioita. TestFlight on Applen luoma sovellus, jolla pelaajat voivat testata beetavaiheessa olevaa peliä ja auttaa täten kehittäjää tekemään pelistä parempi. Luvussa käydään läpi vaihe vaiheelta, kuinka peli saadaan Unitystä pelaajalle testattavaksi.

3.1 Koontiasetukset ja koonnin tekeminen Unityssä

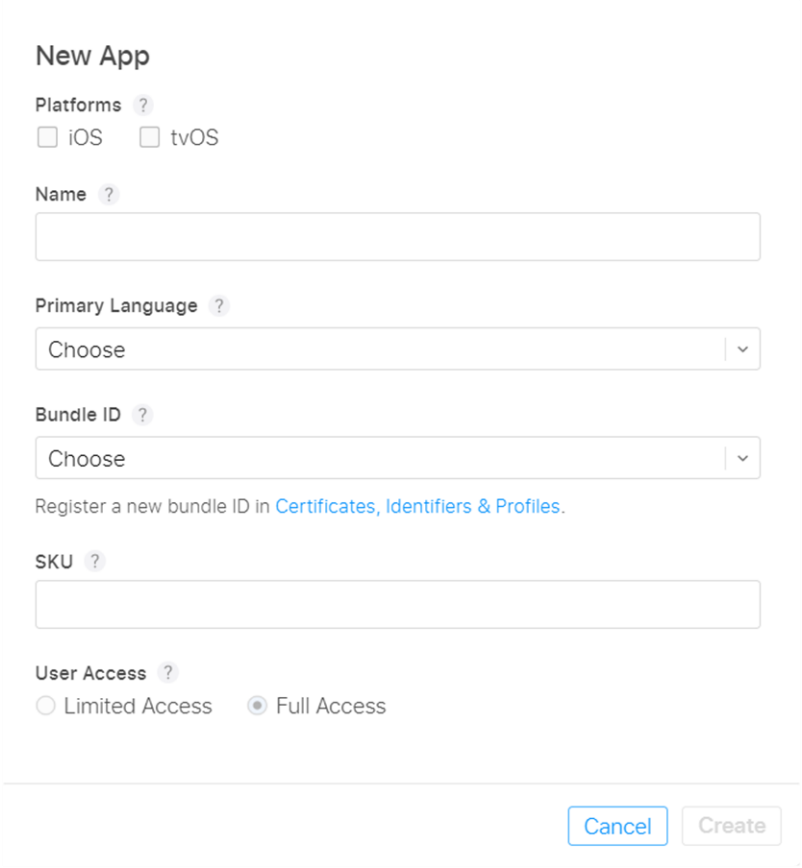
Aiemmilla yrityksillä luoda koonti (engl. build) huomattiin, että jos Unity Recorder oli asennettuna Unityssä iOS-koontia tehdessä, myöhemmin Xcodessa koonnin arkistointi epäonnistui. Sen vuoksi tarkistettiin, että projektista on poistettu Unity Recorder. Se tehtiin valitsemalla Window/Package Manager ja etsimällä sieltä Unity Recorder ja painamalla Remove-nappulaa.

Unityn koontiasetusten säätäminen tehtiin kohdasta Edit/Project Settings/Player. Tuettujen iOS-versioiden joukkoon valittiin iOS 10.0 ja uudemmat versiot. Kohdasta Edit/Project Settings/Player/Other Settings valittiin, että Auto Graphics API on pois päältä. Graphics API -kentässä plusnappia painamalla lisättiin rajapinnat OpenGLES2 ja OpenGLES3. Myös Static Batching -asetus otettiin pois päältä. Architecture-kentässä valittiin käyttöön ARM64-arkkitehtuuri.

Lopuksi tehtiin koonti valitsemalla File/Build Settings. Kohdassa ”Run in Xcode as” valittiin Release ja painettiin Build-nappia.

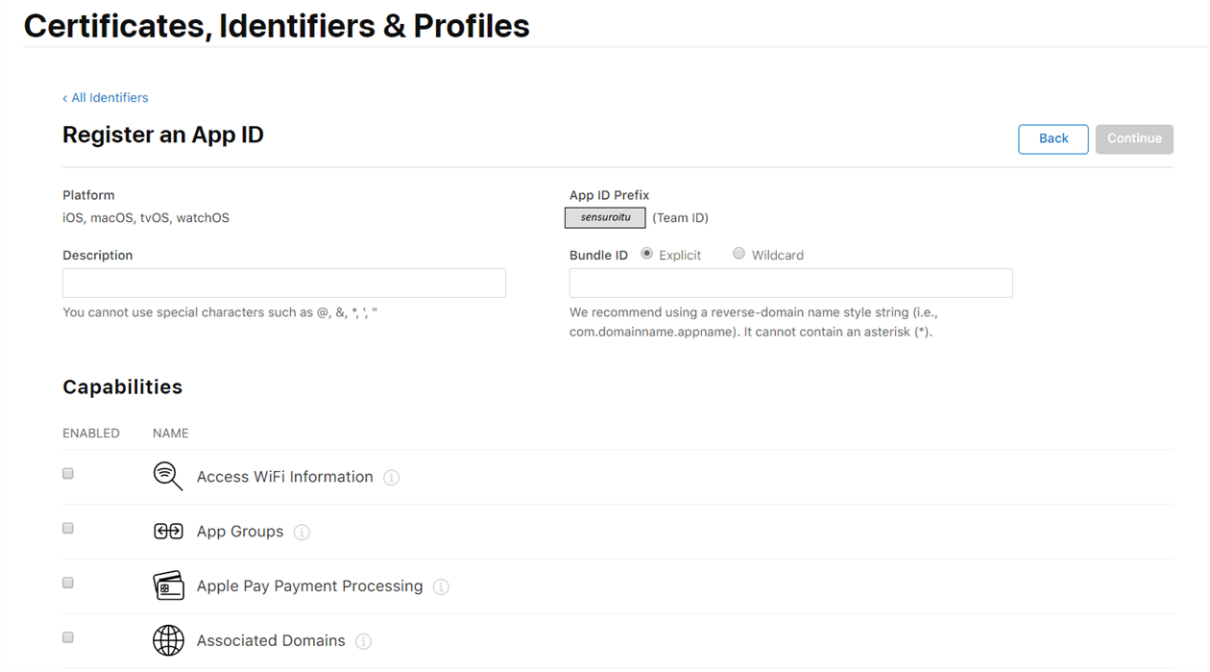
3.2 Uuden sovelluksen rekisteröiminen App Store Connectissa

TestFlightin käyttäminen edellytti, että luodaan uusi sovellus App Store Connectiin [23]. Se tehtiin kohdasta My Apps valitsemalla New App ja täyttämällä kuvan 7 mukaiset tyhjät kentät. Saatavilla ei ollut sopivaa bundle id:tä, joten päätettiin luoda sellainen itse manuaalisesti.



Kuva 7. Uuden sovelluksen luomisikkuna App Store Connectissa.

Bundle id:n luominen onnistui sivustolta developer.apple.com/account kohdasta Certificates, Identifiers & Profiles. Avautuneessa näkymässä (kuva 8) Description-kenttään kirjoitettiin pelin nimi, valittiin Bundle ID -tyypiksi eksplisiittinen (engl. explicit) ja arvoksi reverse-domain-tyyppinen nimi eli muotoa com.domainname.appname. Tarvittaessa tässä näkymässä olisi voitu myös lisätä toiminnallisuuksia (engl. Capabilities), mutta nyt niitä ei lisätty.

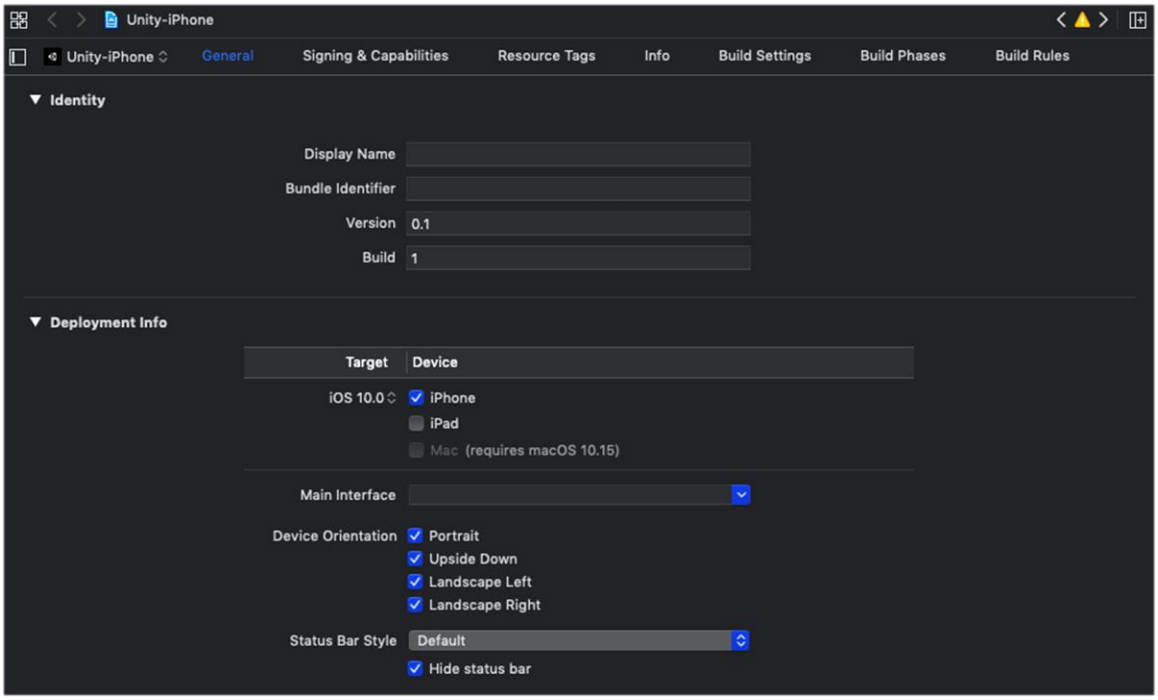


Kuva 8. Sovelluksen rekisteröinnin ikkuna developer.apple.com/account-sivustolla.

Seuraavalla sivulla vain varmistettiin tietojen olevan oikein. Register-painikkeella saatiin luotua uusi sovellustunnus.

3.3 Koonti arkistointi Xcodessa

Kuvassa 9 nähtävällä General-välilehdellä Identity-osioon kirjoitettiin arkistoitavan sovelluksen nimi, Bundle Identifier sekä versio- ja koontinumero.



Kuva 9. Xcoden Identity-näkymä.

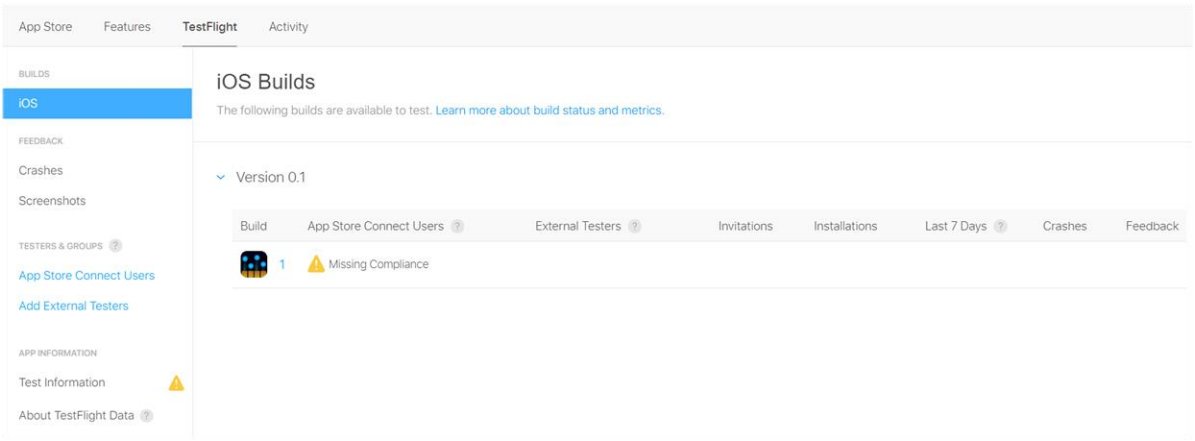
Arkistoinnin aloittaminen tehtiin valitsemalla Product/Archive. Avautuvassa ikkunassa valittiin haluttu koonti ja painettiin Distribute App -painiketta. Tässä vaiheessa tuli peräkkäin useita uusia ikkunoita, joiden kanssa eteneminen selitetään alla.

Ensin valittiin jakelutavaksi App Store Connect ja toimenpiteeksi Upload (suom. lähettäminen). Sen jälkeen valittiin, että sisällytetään iOS-sisältöön bittikoodi (engl. Include bitcode for iOS content) ja että lähetetään sovelluksen symbolit, jotta Apple voi lähettää kehittäjälle symbolisia raportteja (engl. Upload your app's symbols to receive symbolicated reports from Apple). Lopuksi otettiin käyttöön automaattinen sisäänkirjautuminen (engl. Automatically manage signing) ja tarkistettiin, että tiedot ovat oikein.

Tämän jälkeen Xcode aloitti koonnin lähettämisen App Store Connectiin. Jos kävi niin, että aiemmin ei ollut muistettu poistaa Unity Recorderia käytöstä, tässä vaiheessa tuli virheilmoitus ”fccore.bundle has conflicting provisioning settings”. Sitten ei ollut muuta vaihtoehtoa kuin aloittaa koko prosessi alusta Unitystä lähtien niin, että Unity Recorder on poistettu.

3.4 App Store Connectin kanssa toimiminen

Ensin kirjauduttiin sisään App Store Connect -sivustolle ja valittiin kohta My Apps. Sitten valittiin haluttu sovellus ja mentiin TestFlight-välilehdelle. Kun koonnin prosessointi oli valmistunut, tuli näkyviin Missing Compliance -varoitus (kuva 10), joka piti hoitaa kuntoon. Varoituksessa on kyse siitä, että Applelle ei ole annettu Export Compliance -tietoja. Nämä tiedot täytyy antaa, että pääsee eteenpäin TestFlight-prosessissa.

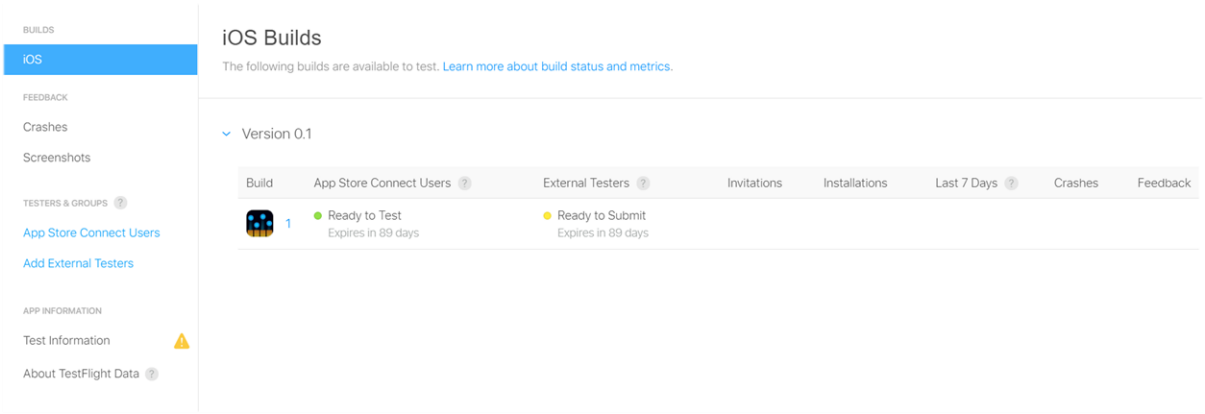


Kuva 10. Koonnin näkymä TestFlight-välilehdellä juuri arkistoinnin jälkeen. Näkyvissä keltaisella Missing Compliance -varoitus.

Export Compliance tarkoittaa tässä tapauksessa sitä, käyttääkö sovellus salausta vai ei. Tämän pelin mikään osa, Game Analytics mukaan lukien, ei käytä salausta, joten ilmoitettiin asia Applelle. Tämän jälkeen Missing Compliance -varoitus katosi.

3.5 Käyttäjien kutsuminen App Store Connectissa

Testaajia voi lisätä joko yksittäisinä henkilöinä tai ryhminä App Store Connectissa. Tässä projektissa päätettiin käyttää ryhmää. Ryhmä luotiin menemällä TestFlight-välilehdelle (kuva 11) ja valitsemalla kohta Add External Users.



Kuva 11. Näkymä TestFlight-välilehdellä.

Näkyviin tuli ikkuna, jossa voi luoda uuden ryhmän. Ryhmän nimeksi annettiin Externals. Tämän jälkeen vasempaan laitaan ilmestyi Externals-niminen kenttä. Sieltä löytyy Testers-niminen osio, jossa käyttäjiä voi lisätä.

Jotta tietylle koonnille saatiin annettua testaajia, täytyi tehdä vielä yksi vaihe. iOS Builds -näkymästä valittiin haluttu koonti. Kohtaan Group lisättiin aiemmin luotu ryhmä nimeltä Externals. Seuraavassa näkymässä täytettiin Test Information -kentän tiedot. Tämä peli ei käytä sisäänkirjautumista, joten se kohta jätettiin valitsematta. Viimeisessä näkymässä täytyi kirjoittaa jokin kuvaus siitä, mihin testaajien tulisi kiinnittää huomiota pelissä.

4 Game Analyticsin informaation tulkinta

Tässä luvussa käydään ensin läpi yleisesti, mihin pelianalytiikalla pyritään ja mitkä ovat tärkeimpiä mitattavia avainlukuja. Sen jälkeen käydään läpi, kuinka Game Analyticsin sivustolta tarkkaillaan avainlukuja. Lisäksi vertaillaan niitä alan suosituksiin ja pohditaan, mitä johtopäätöksiä asiasta voi vetää.

4.1 Pelianalytiikka yleisesti

Kuten Game Analyticsin blogissa [24] todetaan, ensimmäisistä kolikkopeleistä lähtien aina uusimpiin mobiilipeleihin asti pelianalyysin tehtävänä on ollut mitata, mitä pelaajat

pitävät hauskana. Pelianalytiikan avulla voidaan seurata, miten pelaajien valinnat pelissä vaikuttavat taloudellisesti. Toisessa Game Analyticsin blogissa [25] puolestaan käydään läpi tärkeimpiä mitattavia avainlukuja, jotka selitetään alla.

DAU (engl. Daily Active Users) tarkoittaa, kuinka monta uniikkia aktiivista käyttäjää aloittaa vähintään yhden session päivän aikana. DAU-lukumäärä on vain otos tietyllä ajanhetkellä. Ympäröivä konteksti voi olla aivan yhtä tärkeää, ellei jopa tärkeämpää kuin suuri käyttäjäkunta.

Sessiot (engl. Sessions) tarkoittaa, kuinka monta kertaa kuka tahansa käyttäjä käynnistää sovelluksen. Kuten DAU, sessioiden kokonaismäärä vaatii kontekstin ollakseen hyödyllinen. Erityisesti olisi hyvä tarkastella sessioiden keskimäärää suhteessa DAU:hun, koska tämä avainluku voi kertoa, kuinka sitoutuneita käyttäjät ovat peliin. Pelin genrellä on vaikutusta suhteeseen sessiot/DAU. Jos käyttäjät palaavat peliin 5–10 kertaa päivässä, on turvallista olettaa heidän nauttivan pelistä. Jos käyttäjät avaavat pelin ainoastaan 1–2 kertaa päivässä, peli ei todennäköisesti säilytä heidän mielenkiintoaan pitkään, blogissa todetaan.

DAU/MAU (engl. DAU/Monthly Active Users) on suhde, joka kertoo, kuinka hyvin sovellus säilyttää käyttäjiä. Avainluku DAU/MAU voi saada arvoja ainoastaan yhden ja nollan väliltä. Arvot lähempänä yhtä tarkoittavat, että käyttäjät avaavat sovelluksen korkeammalla prosentuaalisella osuudella päivistä. Blogin mukaan suositut sosiaalisen median verkostoitumissovellukset, kuten Facebook, ovat raportoineet DAU/MAU-suhteesta jopa 50 %:n suuruisena. Suosituimmilla peleillä suhde on lähempänä 20:tä prosenttia.

Retentio (engl. Retention) on mahdollisesti tärkein avainluku free-to-play-pelissä, blogissa todetaan. Retention mittaamiseksi käyttäjät jaetaan ryhmiin sovelluksen latauspäivän perusteella. Latauspäiväksi merkitään päivä nolla. Jos käyttäjä avaa sovelluksen seuraavana päivänä, käyttäjä merkitään säilytetyksi. Muuten häntä ei merkitä säilytetyksi. Tämä toimenpide suoritetaan jokaiselle käyttäjäjoukolle jokaisena päivänä latauspäivästä eteenpäin. Yleisiä retention tarkkailupäiviä blogin mukaan ovat 1, 3, 7 ja 30.

Pelaajakato (engl. Churn) tarkoittaa karkeasti retention vastakohtaa eli sitä, kuinka monta sovelluksen ladannutta pelaajaa lopettaa pelaamisen. Tyypillisesti pelaajakato mitataan käyttäjänä, joka ei ole pelannut 28 päivään, blogissa kerrotaan.

Konvertointisuhde (engl. Conversion Rate) mittaa tietyllä aikavälillä ostoksen tehneiden uniikkien pelaajien prosentuaalista suhdetta kaikkiin pelaajiin. Free-to-play-peleissä voi mitata myös esitettyjen mainosten konvertointisuhdetta.

ARPDAU (engl. Average Revenue Per Daily Active User) on yksi eniten keskustelluista avainluvuista mobiilipeleissä, blogin mukaan. ARPDAU:n avulla ymmärtää, miten peli suoriutuu päivittäin tarkasteltuna.

ARPPU (engl. Average Revenue Per Paying User) mittaa vain sellaisten käyttäjien joukkoa, jotka ovat suorittaneet oston pelissä. Tämä avainluku voi vaihdella huomattavasti pelin genren mukaan. Usein hardcore-peleillä eli monimutkaisemmilla peleillä on korkeammat monetisaatioavainluvut, kuten ARPPU, mutta niillä ei ole kasuaalipelien kaltaista massaviehätystä, blogissa todetaan.

Lähde (engl. Source) tarkoittaa paikkaa, jossa käyttäjä voi ansaita valuuttaa. Game Analyticsin Dashboard-näkymässä lähdeavainluku mittaa, kuinka paljon valuuttaa käyttäjä on ansainnut. Se sisältää myös mahdolliset valuutat, joita käyttäjä on lahjoittanut pelintekijälle.

Sink eli vapaasti suomennettuna kaivo, tarkoittaa lähteen vastakohtaa. Nämä ovat pelissä paikkoja, joissa pelaajat käyttävät hankkimiaan valuutoita. Lähde ja kaivo voivat viitata kovaan tai pehmeään valuuttaan (engl. hard/soft currency). Nämä kaksi valuuttatyyppiä täytyy pitää irrallaan toisistaan analyysin aikana, blogissa kerrotaan.

Virtaus (engl. Flow) tarkoittaa lähteen ja kaivon yhdistelmää. Virtaus on pelaajan ansaitseman ja käyttämän valuutan kokonaistilanne. Yleisesti ottaen virtauksen tulisi olla melko tasapainossa. Jos virtauskäyrä lähtee jyrkkään nousuun, pelaajilla on liikaa valuuttaa eikä siten tarvetta monetisoida. Jos virtauskäyrä kallistuu kohti nollaa, pelaajilla ei ole tarpeeksi resursseja tehdä mitään pelissä.

Aloitus (engl. Start) tarkoittaa sitä, montako kertaa pelaaja aloittaa uuden tason. Epäonnistuminen (engl. Fail) tarkoittaa, montako kertaa pelaaja aloittaa tason, mutta ei läpäise sitä. Läpäisy (engl. Complete) tarkoittaa, montako kertaa pelaajat läpäisevät tietyn tason. Näiden kolmen avainluvun yhdistelmä auttaa analysoimaan tasoja pelissä, blogissa todetaan.

Taulukossa 1 on esitetty vielä yhteenvetona kaikki yllä läpikäydyt avainluvut.

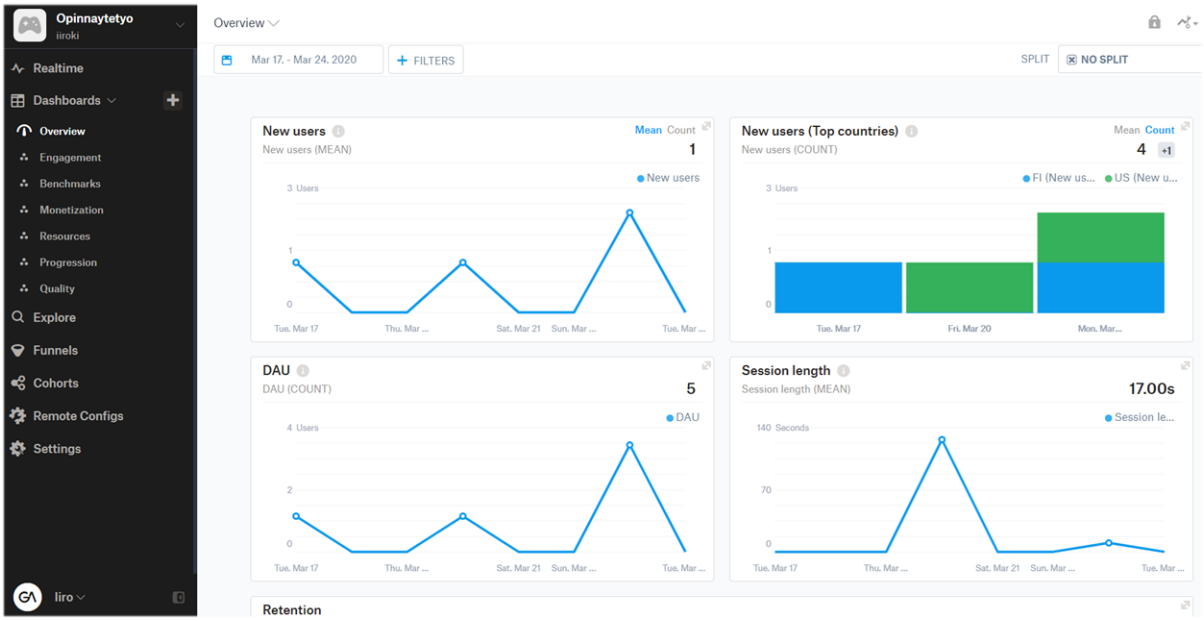
Taulukko 1. Tärkeimpiä avainlukuja pelianalytiikassa Game Analyticsin blogin [25] mukaan.

	Vapaa suomennos	Englanniksi
1.	DAU	DAU (Daily Active Users)
2.	Sessiot	Sessions
3.	DAU/MAU	DAU/MAU (Daily Active Users / Monthly Active Users)
4.	Retentio	Retention
5.	Pelaajakato	Churn
6.	Konvertointisuhde	Conversion Rate
7.	ARPDAU	ARPDAU (Average Revenue Per Daily Active User)
8.	ARPPU	ARPPU (Average Revenue Per Paying User)
9.	Lähde	Source
10.	Kaivo	Sink
11.	Virtaus	Flow
12.	Aloitus	Start
13.	Epäonnistuminen	Fail
14.	Läpäisy	Complete

Kaivossa mainituista kovasta ja pehmeästä valuutasta selvennyksenä, että kova valuutta tarkoittaa, artikkelin [26] mukaan, oikealla rahalla ostettua pelin sisäistä valuuttaa. Pehmeä valuutta puolestaan tarkoittaa yksinomaan pelaamalla hankittua pelin sisäistä valuuttaa.

4.2 Pelin toimimisen tarkkaileminen Game Analyticsissa

Kun oltiin kirjautuneena sisään Game Analyticsin sivustolle [27], valittiin ensin tarkasteltavaksi haluttu peli. Tässä näkymässä oli useita eri vaihtoehtoja, joista lähteä tarkkailemaan pelin tilastoja. Aluksi valittiin kuvassa 12 nähtävä yleiskatsaus (engl. Overview).



Kuva 12. Yleiskatsausnäkymä Game Analyticsin sivustolla [27].

Yleiskatsausnäkymässä näkee

- uusien käyttäjien määrän
- mistä eri maista käyttäjät ovat
- päivittäiset aktiiviset käyttäjät (engl. DAU = daily active users)
- pelisession keston
- pelin retention.

Näistä eri avainluvuista (engl. Key Performance Indicator) voitiin päätellä mahdollisia ongelmakohtia, joihin on syytä puuttua.

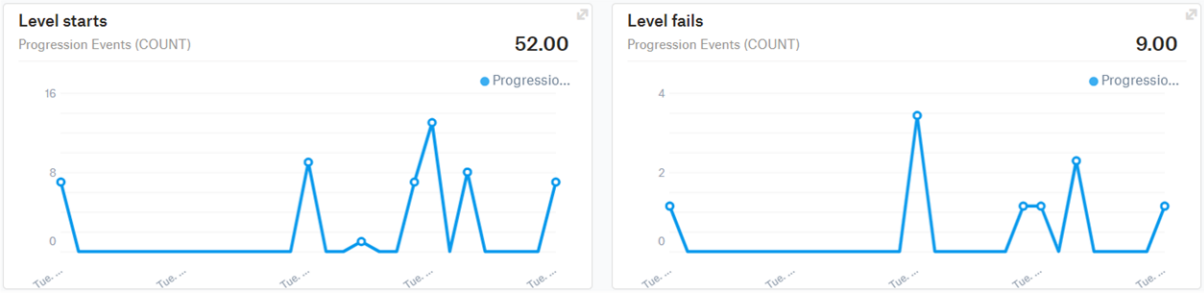
4.3 Ongelmien tunnistaminen ja korjaaminen Game Analyticsissa

Koska pelillä oli hyvin vähäinen määrä käyttäjiä, ei pelin avainlukujen arvoista voinut vetää merkityksellisiä johtopäätöksiä. Kuitenkin oppimiskokemuksen nimissä niitä päätettiin vertailla alan suosituksiin, joiden referenssinä käytettiin Game Analyticsin omaa raporttia vuodelta 2019 [28].

Game Analyticsin raporttiin vertaamalla havaittiin, että useissa avainluvuissa olisi parannettavaa. Alla on vertailtu tyypillisiä avainlukujen arvoja, kun kyseessä on menestynyt lajityypin kasuaali (engl. casual) peli.

- Päivän 1 retention suositus on yli 35 % [28, s. 7]. Pelissä se oli 1,85 %.
- Päivän 7 retention suositus on yli 11 % [28, s. 9]. Pelissä se oli 0 %.
- Päivän 28 retention suositus on yli 4 % [28, s. 11]. Pelissä se oli 0 %.
- Session pituuden suositus on yli 10 minuuttia [28, s. 13]. Pelissä se oli 40,8 sekuntia.

Toinen tapa, millä pelin toimivuutta voitiin tarkkailla, oli etenemistapahtumat (engl. Progression Event). Ne ohjelmoitiin pelin koodin sekaan siten, että ne tapahtuvat halutulla ajanhetkellä pelin aikana. Kuvassa 13 nähdään, miltä pelitason aloitus- ja epäonnistumistapahtumista saadut arvot näyttävät Game Analyticsin sivustolla.



Kuva 13. Tason aloitus- ja epäonnistumistapahtumat Game Analyticsin Progression-välilehdellä.

Kun verrattiin tasojen aloitus- ja epäonnistumismääriä, huomattiin, että ne eivät täsmää. Tästä voitiin päätellä, että pelaaja todennäköisesti lopetti usein pelin ennen pelin päättymisruutua. Tämän perusteella voisi olla järkevää miettiä, miten pelissä saisi säilytettyä pelaajan mielenkiinnon pidempään ja korjata siten pelin toimintaa.

5 iOS-pelin julkaisu App Storeen

Tässä luvussa käydään aluksi App Store -vaatimukset yleisellä tasolla läpi. Sen jälkeen käydään läpi julkaisuprosessi App Storeen, mutta se ei kuitenkaan perustu opinnäytetyön aikana tehtyyn peliin, vaan toiseen vuonna 2019 julkaistuun peliin, jota raportin kirjoittaja on ollut mukana tekemässä. Julkaisun apuna käytettiin Applen omia ohjeita [29].

5.1 App Store -vaatimukset yleisesti

Ennen koonnin lähetystä App Storeen täytyy pelinkehittäjän huolehtia monista eri asioista, jotka löytyvät Applen ohjesivustolta [30]. Apple itse luokittelee ohjeistukset viiteen eri osa-alueeseen, jotka ovat turvallisuus, suorituskyky, liiketoiminta, design ja lakiasiat.

Ennen koonnin lähetystä App Store -arviointiin, tulee pelinkehittäjän huolehtia seuraavista asioista:

- Pelissä ei ole ilmiselviä virheitä.
- Kaikki sovelluksen tiedot ovat oikein, ja metadata on annettu ja täsmää pelin kanssa.
- Yhteystiedot on päivitetty, jotta Apple saa otettua pelinkehittäjään yhteyttä ongelmatilanteissa.
- Aktiivinen demotili kirjautumistietoineen on annettu Applelle, mikäli tietoja tarvitaan sovelluksen testaamiseen perusteellisesti.
- Pelin backend-palvelut ovat päällä arvioinnin aikana.
- Applelle on toimitettu yksityiskohtaiset selitykset ominaisuuksista, jotka eivät ole itsestään selviä, ja myös sovellusten sisäisistä ostoista.

Yllä on vain pieni osa kaikista vaatimuksista, jotka löytyvät ohjesivustolta [30].

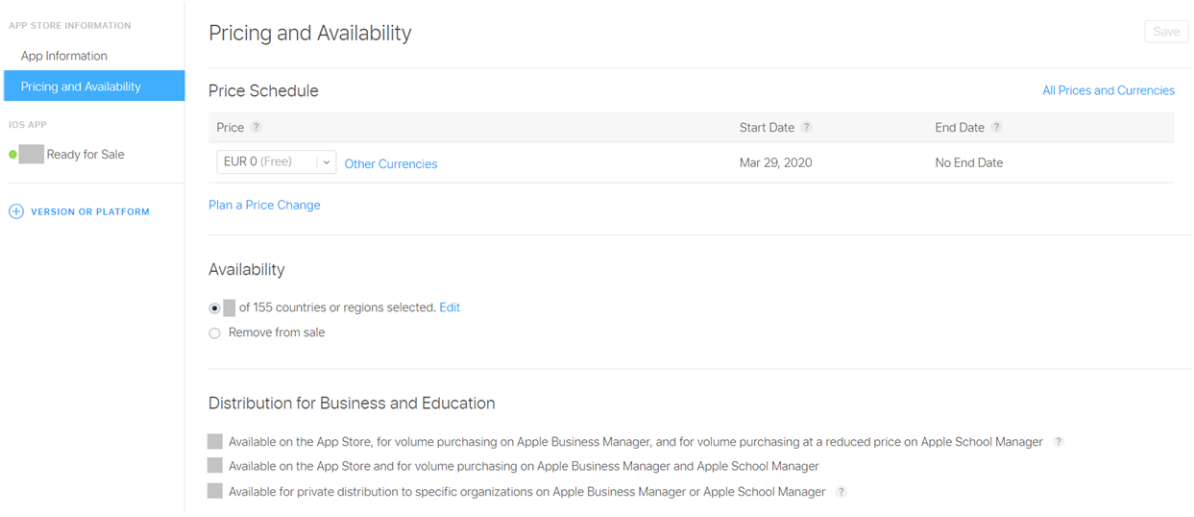
5.2 Alkutoimenpiteet App Store Connectissa

Kun oltiin kirjautuneena sisään App Store Connectiin, valittiin sieltä haluttu sovellus ja käytiin täyttämässä App Information -kohtaan seuraavat tiedot:

- nimi (engl. Name)
- linkki tietosuojakäytäntöön (engl. Privacy Policy URL)
- kategoria (engl. Category).

Seuraavaksi valittiin kohta Pricing and Availability (kuva 14) eli hinnoittelu ja saatavuus. Koska kyseessä oli free-to-play-peli, valittiin hinnaksi nolla euroa. Tässä vaiheessa, kun peliä ei vielä ollut julkaistu, voitiin valita myös, sallitaanko pelin ennakkotilaus vai ei. Availability-kenttään valittiin haluttu jakelualue eli se, minkä maiden App Storeihin peli

halutaan saada. Kohdassa Distribution for Business and Education (suom. jakelu yritys- ja opetuskäyttöön) valittiin omaan sovellukseen sopiva vaihtoehto.



Kuva 14. Hinnoittelu- ja saatavuusasetukset App Store Connectissa, kun peli on julkaistu.

Seuraavat työvaiheet tehtiin kohdassa Prepare for Submission (suom. valmistele jättö). Julkaisun jälkeen tämän osion nimeksi tuli Ready for Sale (suom. valmis myyntiin).

5.3 App Storeen tulevien kuvakaappausten ja videoiden kriteerit

App Storeen tulevat kuvakaappaukset ja esittelyvideo tehtiin noudattamalla Applen määrittämiä kriteereitä [31]. 6,5-tuuman iPhonelle tarkoitetuissa kuvissa käytettiin 1242 x 2688 -resoluutiota. 5,5-tuuman iPhonelle tarkoitetuissa kuvissa resoluutio oli 1242 x 2208.

Esittelyvideo (engl. App preview) tehtiin 6,5-tuumaiselle iPhonelle. Video käytti mp4-formaattia, resoluutio oli 886 x 1920, äänen näytteenottotaajuus oli 48 kHz ja videon pituus mitoitettiin välille 15–30 sekuntia.

Oli tärkeää, että kuvien ja videoiden sisältö vastasi tarkalleen peliä sen uusimmassa koonnissa App Storessa. Tämän seurauksena, jos pelinkehittäjä teki visuaalisia muutoksia peliin ja latasi uuden koonnin App Storeen, täytyi hänen ottaa myös kuvat ja videot

pelistä uudelleen. Muussa tapauksessa Apple ei hyväksynyt koonnin lähettämistä App Storeen.

5.4 Tietojen keräämiseen liittyvät kysymykset App Store Connectissa

Kun oltiin valmiita laittamaan koonti App Storeen, mentiin kohtaan Prepare for Submission. Painiketta Submit for Review (suom. jätä arvosteluun) painamalla tuli näkyviin uusi ikkuna, jossa kysyttiin viennin vaatimustenmukaisuudesta (engl. Export Compliance) ja mainostunnisteesta (engl. Advertising Identifier).

Export Compliance -kohdassa Apple halusi tietää, onko pelissä mahdollisesti käytettävään salaukseen tehty muutoksia edelliseen ilmoitukseen verrattuna. Lisäksi Apple varoittaa, että mikäli tässä asiassa ei toimita oikein, tulee vakavia seuraamuksia.

Advertising Identifier -kohdassa Apple halusi tietää, käyttääkö sovellus mainostunnistetta. Tekstissä kerrotaan, että mainostunniste on uniikki tunniste jokaiselle iOS-laitteelle, ja se on ainoa tapa tarjota kohdennettuja mainoksia (engl. Targeted Ads). Käyttäjät voivat rajoittaa mainosten kohdentamista iOS-laitteissaan.

Tekstissä jatketaan, että jos sovellus käyttää kohdennettua mainontaa, tulee kehittäjän tarkistaa sovelluksen koodi, mukaan lukien sen sisältämät kolmansien osapuolten koodit, ennen koonnin jättöä. On kehittäjän omalla vastuulla pitää huoli siitä, että sovelluksen koodi noudattaa kehittäjän valitsemia kohtia, jotka luetellaan alla vapaasti suomennettuna. Sovellus käyttää mainostunnistetta

- mainosten näyttämiseen sovelluksessa (engl. Serve advertisements within the app)
- määrittämään tämän sovelluksen asennus aiemmin näytettyyn mainokseen (engl. Attribute this app installation to a previously served advertisement)
- määrittämään tässä sovelluksessa suoritettu toiminto aiemmin näytetylle mainokselle (engl. Attribute an action taken within this app to a previously served advertisement).

Lopuksi tekstissä halutaan kehittäjältä vahvistus siihen, että sovellus ja sen sisältämät kolmansien osapuolten käyttöliittymät käyttävät mainostunnistetarkistusta. Lisäksi niiden tulee noudattaa käyttäjän asetuksia liittyen mainosseurannan rajoittamiseen.

Mikäli käyttäjä kytkee päälle Limit ad tracking -asetuksen iOS:ssä, sovellus ei saa käyttää mainostunnistetta eikä mitään sen kautta saatua informaatiota, muuten kuin rajoitettuun mainoskäyttötarkoitukseen (engl. Limited Advertising Purposes). Tämä mainoskäyttötarkoitus on määritelty iOS Developer Program License Agreement -lisenssisopimuksessa. Lisenssisopimuksen pystyi lukemaan tekstin linkistä, mutta raportin kirjoitushetkellä siihen ei päästy käsiksi, että sitä olisi voitu käyttää lähteenä tässä.

Kun kysymyksiin oli vastattu, vahvistettiin koonnin jättö painamalla Submit-painiketta. Tässä vaiheessa jäätiin odottamaan Applen arviointia, kelpaako koonti App Storeen. Arvostelun tilaa pystyi seuraamaan Activity -välilehdeltä kohdassa App Store Versions.

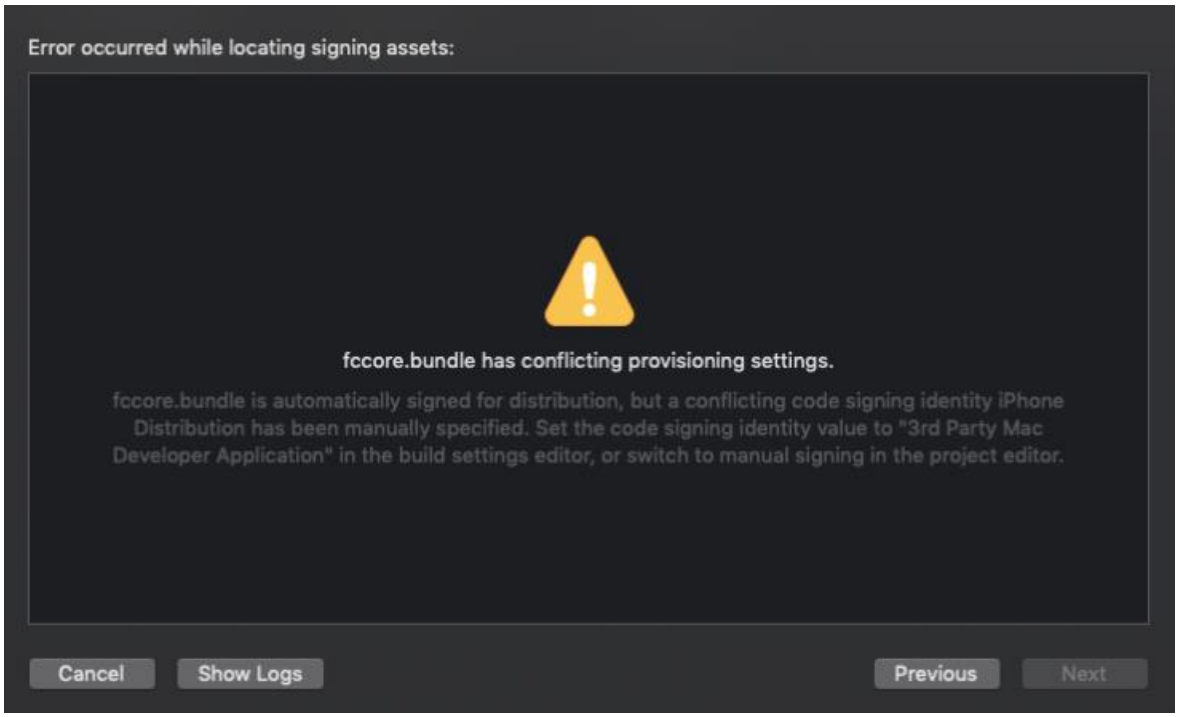
5.5 Ongelmatilanteet koonnin lähetyksen kanssa

Ajoittain tuli vastaan tilanne, jossa oli tehty jokin vaihe väärin, ja sen vuoksi Apple ei hyväksynyt koonnin lähettämistä App Store Connectiin tai App Storeen. Näissä tilanteissa kuitenkin tuli aina jokin virheilmoitus asiaan liittyen. Virheilmoituksen perusteella pystyi usein ymmärtämään tai päättelemään, mikä meni väärin, ja korjaamaan sitten asian. Myös käyttämällä jotakin hakukonetta, kuten Googlea, pystyi saamaan lisätietoa virheilmoituksesta, jolloin ymmärsi paremmin, mitä pitää korjata.

Esimerkiksi oli tilanne, jossa yritettiin testausta varten luoda koontia suoraan Xcodesta iPhoneen. Koonti epäonnistui, ja tuli näkyviin virheilmoitus, jossa kerrottiin, että Xcoden nykyinen versio ei ehkä ole yhteensopiva kyseisen iOS-version kanssa. Ohjeessa neuvottiin päivittämään Xcode, ja se ratkaisikin ongelman.

Aina virheilmoitukset eivät kuitenkaan antaneet suoraa vastausta siihen, mikä ongelman aiheutti. Raportissa aiemmin mainittu Unity Recorder -liitännäisen asennus aiheutti Xcodessa arkistoinnin epäonnistumisen, ja virheilmoituksesta (kuva 15) ei käynyt suo-

raan selväksi, mikä ongelman aiheutti. Tässä tilanteessa Googlea käyttämällä löytyi kuitenkin ratkaisu ongelmaan keskustelupalstalta, jossa muut kehittäjät olivat ihmetelleet samaa asiaa.



Kuva 15. Virheilmoitus Xcodessa arkistointia tehdessä.

Oli tilanteita, joissa yritettiin saada koontia App Storeen ja se ei mennyt läpi Applen tarkistuksesta. Näissä tilanteissa Apple kuitenkin lähetti kehittäjälle viestin, jossa kerrottiin syyt, joiden vuoksi koonti hylättiin. Jos kävi niin, että Apple oli ymmärtänyt väärin sovelluksen toiminnan, asiasta pystyi lähettämään Applelle tarkentavan viestin, ja tällä tavoin asia voitiin myös ratkaista.

5.6 Uuden peliversion lisääminen App Storeen

Applen ohjeissa [32] kerrotaan, että uuden peliversion lisääminen App Storeen on mahdollista vain silloin, kun nykyisen version tila (engl. status) on Ready for Sale tai Developer Removed from Sale. Lisäksi on syytä huomioida, että jos uudessa versiossa on ongelma, ei ole mahdollista palata takaisin vanhaan versioon.

Kuten ensimmäisen koonnin kanssa, ensin tehtiin uusi koonti Unityssä ja avattiin projekti Xcode-ohjelmalla. Sen jälkeen arkistoitiin koonti. App Store Connectissa mentiin kohtaan App Store Connect/My Apps ja valittiin peli, joka halutaan päivittää. Plus-nappulalla avautui uusi ikkuna, johon syötettiin uuden koonnin versionumero, ja näin saatiin luotua uusi versio pelistä App Store Connectiin. Tästä eteenpäin menettelytapa eteni samalla tavalla kuin ensimmäisen version kanssa.

Kun uusi koonti meni Applen tarkastuksesta läpi, se korvasi kokonaan vanhan version App Storessa. Tämä on syytä pitää mielessä erityisesti tilanteissa, joissa on poistettu tuki vanhemmalta iOS-versiolta. Tällöin kyseisen iOS-version käyttäjät eivät enää pysty lataamaan uutta peliversiota App Storesta.

Kuitenkin App Store Connectissa kohdassa Pricing and Availability/Last-Compatible Version Apple kertoo säilyttävänsä tietynlaista arkistoa vanhoista peliversioista. Tällöin käyttäjät, jotka eivät voi asentaa uutta versiota, voivat silti jatkaa pelaamista vanhalla versiolla. Kehittäjä voi halutessaan poistaa jakelusta näitä vanhoja versioita kohdassa Last-Compatible Version. Näkymässä kerrotaan, että ainoa syy, miksi niitä kannattaa poistaa, on sellainen tilanne, jossa yhdessä tai useammassa versiossa on merkittäviä juridisia tai käytettävyyteen liittyviä ongelmia.

6 Johtopäätökset

Insinöörityön aikana saatiin kehitettyä tavoitteiden mukaisesti hyperkasuaalipeli. Peli sisälsi pelaajan näkökulmasta tärkeitä ominaisuuksia, kuten nopean ja helpon tavan aloittaa pelaaminen sekä paljon pelattavaa. Lisäksi pelissä oli käyttöliittymän valikot ja äänet. Kehittäjän näkökulmasta peli sisälsi myös tärkeitä ominaisuuksia, kuten suorituskyvyn optimointia ja analytiikkapalvelun integraation.

Peli saatiin TestFlight-palvelun kautta käyttäjille testattavaksi, kuten oli aikomuskin. Lisäksi peli läpäisi Applen tarkastuksen ensimmäisellä yrityksellä, mikä oli myönteinen yllätys. Projektin tavoitteiden mukaisesti pelin toiminnan avainlukuja voitiin seurata Game Analyticsin sivustolta.

Monetisaation integroiminen peliin ei sisältynyt projektin tavoitteisiin. Se on kuitenkin hyvin oleellinen osa-alue koko hyperkasuaalipelin kehitysprosessia, joten siihen perehdyttiin teoriatasolla. Sen toteutus päätettiin jättää pois, koska mainosten kunnollinen integroiminen peliin olisi vaatinut merkittävästi lisää suunnittelua ja toiminnallisuuksia, joiden oheen mainokset olisi liitetty. Muutoin vaarana olisi pelin käyttäjäkokemuksen heikentyminen.

Projekti mitoitettiin alusta lähtien sellaiseksi, että pelissä ei ollut tarvetta käyttää 3D-malleja, jotka kaupallista peliä tehdessä voivat monesti olla aika tärkeitä. On tosin syytä huomioida, että hyperkasuaalipeleissä 3D-mallit eivät ole ehdoton edellytys menestymiselle. Tästä hyvänä esimerkkinä on raportissa aiemmin mainittu Stack Ball, joka menestyi hyvin siitä huolimatta, että siinä käytettiin niukasti 3D-malleja.

Tavoite oli saada peli vähintään yhdelle laitteelle testattavaksi, ja tässä onnistuttiin. Toki olisi ollut hyödyllistä, jos testaajia olisi ollut esimerkiksi tuhat tai enemmän. Se olisi vaatinut jonkinlaista käyttäjänhankintakampanjaa, joka puolestaan olisi ollut epäkäytännöllistä toteuttaa näin pienessä yhden henkilön projektissa. Tämä kuitenkin aiheutti sen, että pelin avainlukuja ei voitu vertailla merkityksellisesti Game Analyticsin viitearvoihin. Toisaalta se ei kuulunutkaan projektin tavoitteisiin. Oppimistarkoituksessa arvoja kuitenkin vertailtiin.

Oppiminen ja sen analysoiminen olivat yksi insinöörityön tavoitteista. Pelinkehitysprosessi oli itsessään todella opettavainen kokemus, ja se todettiin paremmaksi oppimistyökaluksi verrattuna internetistä löytyvien ohjevideoiden (engl. tutorial) mukana tekemiseen. Syynä tähän oli, että aiempien kokemusten pohjalta ohjevideoiden mukana tekemällä lähinnä kopioi, mitä videosta näytettiin, eikä yrittänyt ensin ajatella itse, miten jokin toiminnallisuus olisi järkevää toteuttaa.

Siinä, että ajattelee kaiken itse, on tietysti vaarana, että oppimisprosessista tulee hidas, epämiellyttävä ja epämotivoiva. Kuitenkin huomattiin, että motivaatio säilyi hyvin läpi projektin sen ansiosta, että oli todella motivoivaa luoda jotain itse keksittyä verrattuna siihen, että kopioi toisen henkilön keksimää toteutusta videon mukana. Samalla tuli onnistumisen kokemuksia, mikä antoi mahdollisuuden pitkäkestoiseen motivaation säilymiseen.

Kuitenkin muutamissa tilanteissa kehittäjän taidot eivät riittäneet jonkin toiminnallisuuden toteuttamiseen, minkä seurauksena oli pakko turvautua ohjeeseen.

Ratkaisevana tekijänä yhtälössä oli kuitenkin hyperkasuaalipelien yksinkertaisuus, jolloin eri työvaiheet eivät käyneet ylivoimaisen vaikeiksi. Lisäksi pelin yksinkertaisuus antoi mahdollisuuden lisätä peliin muita toiminnallisuuksia, kuten analytiikkapalvelun käyttöönoton. Valmista projektia jälkikäteen tarkasteltaessa havaittiin, että hyperkasuaalipelin kehittäminen on erinomainen tapa oppia pelinkehitystä.

7 Yhteenveto

Insinöörityössä käytiin läpi pelin kehittämisen eri vaiheet ja se, kuinka koonti saatiin ensin Unitystä Xcodeen ja sieltä App Store Connectiin. Sitten puitiin TestFlight-sovelluksen käyttöä ja kuinka App Store Connectista saadaan peli testattavaksi TestFlightiin. Lisäksi käytiin teoriatasolla läpi, kuinka julkaisuprosessi App Storeen tapahtuu ja mitä asioita siinä tulee ottaa huomioon. Raportissa myös sivuttiin, miten ongelmatilanteita saatiin ratkaistua.

Kaikki tärkeimmät insinöörityön alussa suunnitellut ominaisuudet ja osa-alueet saatiin tehtyä. Niihin sisältyivät iOS-alustan peli, jossa on analytiikkapalvelu käytössä, TestFlight-palvelun käyttöönotto ja pelin analytiikkapalvelun avainarvojen tulkinta. Monetisaatio, jonka varsinainen integrointi peliin ei sisältynyt projektin tavoitteisiin, käytiin teoriatasolla läpi. Lisäksi projektin visio oli alusta lähtien sellainen, että siinä ei ollut tarvetta käyttää 3D-malleja, jotka kaupallista peliä tehdessä voivat olla tärkeitä, pelistä riippuen.

Yksi insinöörityön tavoitteista oli selvittää, toimiiko se hyvänä oppimistyökaluna pelinkehitykseen. Tässä asiassa projekti onnistui. Projektin tekijä oppi todella paljon niin Unity-pelimoottorin kuin Game Analytics -palvelun osalta. Ainakin osatekijänä oppimisessa uskottiin olevan sen, että koetettiin miettiä mahdollisimman pitkään itse ratkaisua ongelmiin ennen opetusvideoihin turvautumista. Oli myös paljon motivoivampaa perehtyä pelianalytiikan termeihin, kun sai samalla integroida niitä itse tehtyyn peliin.

Vaikka hyperkasuaalipelin tekeminen itsessään voi olla suhteellisen helppoa verrattuna muihin pelien lajityyppeihin, sen vuoksi on myös helpompi yhdistellä siihen muita toiminnallisuuksia, kuten vaikkapa analytiikka- tai mainospalvelu. Tämä projekti antoi hyvän pohjan, jonka päälle jatkaa pelinkehityksen opettelua.

Lähteet

1 JetBrains Rider. 2020. Verkkoaineisto. JetBrains. <https://www.jetbrains.com/rider/>. Luettu 6.4.2020.

2 GitHub Desktop. 2020. Verkkoaineisto. GitHub. <https://desktop.github.com/>. Luettu 18.4.2020.

3 Trello Home Page. 2020. Verkkoaineisto. Trello. <https://trello.com/>. Luettu 18.4.2020.

4 What are hyper casual games and how do you monetize them. 2019. Verkkoaineisto. ironSource. < https://www.ironsrc.com/blog/what-are-hyper-casual-games-and-how-do-you-monetize-them/>. Luettu 8.4.2020.

5 Stack Ball 3D. 2020. Verkkoaineisto. App Store Preview. <https://apps.apple.com/us/app/stack-ball-3d/id1456732568>. Luettu 8.4.2020.

6 Top Hyper-Casual Games Worldwide for May 2019 by Downloads. 2019. Verkkoaineisto. Sensor Tower. <https://sensortower.com/blog/top-hyper-casual-games-worldwide-may-2019-downloads>. Luettu 8.4.2020.

7 How to Market a Hyper-Casual Game – 5 Tips. 2020. Verkkoaineisto. PubNative. <https://pubnative.net/blog/how-to-market-a-hyper-casual-game-5-tips/>. Luettu 9.4.2020.

8 Non-Stop Space Defense. 2020. Verkkoaineisto. App Store Preview. <https://apps.apple.com/us/app/non-stop-space-defense/id1434276380>. Luettu 7.4.2020.

9 Nystrom, Robert. 2014. Singleton. Verkkoaineisto. <http://gameprogrammingpatterns.com/singleton.html>. Luettu 10.4.2020.

10 Nystrom, Robert. 2014. Object Pool. Verkkoaineisto. <http://gameprogrammingpatterns.com/object-pool.html>. Luettu 10.4.2020.

11 Placzek, Mark. 2016. Object Pooling in Unity. Verkkoaineisto. <https://www.raywenderlich.com/847-object-pooling-in-unity>. 23.11.2016. Luettu 13.4.2020.

12 Safe Area Helper. 2020. Verkkoaineisto. Unity Asset Store. <https://assetstore.unity.com/packages/tools/gui/safe-area-helper-130488>. Luettu 10.4.2020.

13 OpenGameArt Home Page. 2020. Verkkoaineisto. OpenGameArt. < https://opengameart.org/>. Luettu 10.4.2020.

14 Introduction to AUDIO in Unity. 2017. Verkkoaineisto. YouTube. <https://www.youtube.com/watch?v=6OT43pvUyfY&t=680s>. Luettu 4.3.2020.

15 Audacity Home Page. 2020. Verkkoaineisto. Audacity. <https://www.audacityteam.org/>. Luettu 10.4.2020.

16 Game Analytics Home Page. 2020. Verkkoaineisto. GameAnalytics. < https://gameanalytics.com/>. Luettu 10.4.2020.

17 GameAnalytics - Fast, Free Game Analytics. 2020. Verkkoaineisto. GameAnalytics. < https://assetstore.unity.com/packages/add-ons/services/analytics/game-analytics-fast-free-game-analytics-6755>. Luettu 10.4.2020.

18 Unity SDK. 2020. Verkkoaineisto. GameAnalytics. < https://gameanalytics.com/docs/item/unity-sdk>. Luettu 10.4.2020.

19 Hyper-Casual Games 101: How To Get The Most Out Of Your Players. 2018. Verkkoaineisto. GameAnalytics. <https://gameanalytics.com/blog/monetizing-hyper-casual-games.html>. Luettu 18.4.2020.

20 ironSource Home Page. 2020. Verkkoaineisto. ironSource. <https://www.ironsrc.com/>. Luettu 21.4.2020.

21 Unity Ads. 2020. Verkkoaineisto. Unity. <https://unity3d.com/unity/features/ads>. Luettu 18.4.2020.

22 Testing Apps with TestFlight. 2020. Verkkoaineisto. Apple. <https://testflight.apple.com/>. Luettu 10.4.2020.

23 App Store Connect. 2020. Verkkoaineisto. Apple. <https://appstoreconnect.apple.com/>. Luettu 10.4.2020.

24 Games and Analytics: A look through time. 2015. Verkkoaineisto. GameAnalytics. <https://gameanalytics.com/blog/games-and-analyticsa-look-through-time.html>. Luettu 11.4.2020.

25 15 Metrics All Game Developers Should Know by Heart. 2015. Verkkoaineisto. GameAnalytics. <https://gameanalytics.com/blog/metrics-all-game-developers-should-know.html>. Luettu 11.4.2020.

26 Here are 6 tips for a better game monetization strategy. 2019. Verkkoaineisto. Woobi. <https://woobi.com/2019/05/6-tips-for-game-monetization-strategy/>. Luettu 11.4.2020.

27 Game Analytics Login. 2020. Verkkoaineisto. GameAnalytics. <https://go.gameanalytics.com/login>. Luettu 14.4.2020.

28 Mobile Gaming Benchmarks. 2019. Verkkoaineisto. GameAnalytics. <https://pages.gameanalytics.com/rs/686-EPV-320/images/H1-2019-Mobile-Benchmarks-Report-GameAnalytics.pdf>. Luettu 31.3.2020.

29 App Store Connect Help. 2020. Verkkoaineisto. Apple. <https://help.apple.com/app-store-connect/. Luettu 29.3.2020.

30 App Store Review Guidelines. 2020. Verkkoaineisto. Apple. <https://developer.apple.com/app-store/review/guidelines/>. Luettu 11.4.2020.

31 Screenshot specifications. 2020. Verkkoaineisto. Apple. <https://help.apple.com/app-store-connect/#/devd274dd925>. Luettu 30.3.2020.

32 Create a new version. 2020. Verkkoaineisto. Apple. <https://help.apple.com/app-store-connect/#/dev480217e79>. Luettu 17.4.2020.

CannonSelector-skripti

```
using System.Collections;
using TMPro;
using UnityEngine;

public class CannonSelector : MonoBehaviour
{
    private bool hasFullyChargedShot;
    private bool isPlaying;

    private float elapsedChargingTime;

    private GameObject objectHitByRay;

    private ParticleSystem fullyChargedParticleSystem;

    private CannonParticleSystemReferences cannonParticleSystemReferences;
    private ParticleSystem chargingParticleSystem;
    [SerializeField] private ParticleSystem[] chargingParticleSystems;
    [SerializeField] private TMP_Text superProjectileReadyText;

    private ProjectileExpander projectileExpander;

    private void Start ()
    {
        elapsedChargingTime = 0;
    }

    void Update()
    {
        if (Input.GetKey (KeyCode.Mouse0))
        {
            elapsedChargingTime += Time.deltaTime;

            if (elapsedChargingTime >= 0.3f)
            {
                var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
                RaycastHit hitInfo;
                if (Physics.Raycast (ray, out hitInfo))
                {
                    objectHitByRay = hitInfo.collider.gameObject;
                    cannonParticleSystemReferences = objectHitByRay.GetCompo-
nent<CannonParticleSystemReferences> ();
                    if (objectHitByRay && !isPlaying && cannonParticleSys-
temReferences)
                    {
                        chargingParticleSystem = cannonParticleSystemRefer-
ences.GetChargingParticleSystem ();
                        if (!chargingParticleSystem.gameObject.activeSelf) //
This is just because the particle systems were constantly playing in editor,
so now they are disabled by default and need to be re-enabled before use.
                            chargingParticleSystem.gameObject.SetActive
(true);
                        chargingParticleSystem.Play();
                        isPlaying = true;
                        SoundManager.Instance.Play ("ChargeSuperProjectile");

                        projectileExpander = objectHitByRay.GetCompo-
nentInParent<ProjectileExpander> ();
```

```
                        if (projectileExpander)
                        {
                            projectileExpander.HideAndResetChargedProjectile
();
                            projectileExpander.StartChargingProjectile ();
                        }
                        else
                        {
                            Debug.Log ("Couldn't find projectileExpander: " +
projectileExpander);
                        }
                    }
                }

                if (elapsedChargingTime >= 1.5f && !hasFullyChargedShot)
                {
                    SoundManager.Instance.Play ("SuperProjectileReady");
                    hasFullyChargedShot = true;
                    superProjectileReadyText.enabled = true;
                    StartCoroutine(HideTextAfterDelay (superProjectileReady-
Text));
                    if (projectileExpander)
                    {
                        projectileExpander.StopChargingProjectile ();
                    }
                    else
                    {
                        Debug.Log ("Couldn't find projectileExpander: " + pro-
jectileExpander);
                    }
                }
            }
        }

        if (Input.GetMouseButtonUp (0))
        {
            var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
            RaycastHit hitInfo;
            if (Physics.Raycast (ray, out hitInfo))
            {
                objectHitByRay = hitInfo.collider.gameObject;
                if (hasFullyChargedShot)
                {
                    var superProjectileLauncher = objectHitByRay.GetCompo-
nentInParent<SuperProjectileLauncher> ();
                    if (superProjectileLauncher)
                    {
                        superProjectileLauncher.TryToLaunchSuperProjectile ();
                        if (projectileExpander)
                        {
                            projectileExpander.HideAndResetChargedProjectile
();
                        }
                        else
                        {
                            Debug.Log ("Couldn't find projectileExpander: " +
projectileExpander);
                        }
                    }
                }
                else
                {
```

```
                    var projectileLauncher = objectHitByRay.GetCompo-
nentInParent<ProjectileLauncher> ();
                    if (projectileLauncher)
                    {
                        SoundManager.Instance.Stop ("ChargeSuperProjectile");
                        projectileLauncher.TryToLaunchProjectile ();
                        if (projectileExpander)
                        {
                            projectileExpander.HideAndResetChargedProjectile
();
                        }
                        else
                        {
                            Debug.Log ("Couldn't find projectileExpander: " +
projectileExpander);
                        }
                    }
                }
            }

            for (int i = 0; i < chargingParticleSystems.Length; i++)
            {
                chargingParticleSystems[i].Stop();
            }

            elapsedChargingTime = 0;
            isPlaying = false;
            hasFullyChargedShot = false;
        }
    }

    IEnumerator HideTextAfterDelay (TMP_Text text)
    {
        yield return new WaitForSeconds (1);
        text.enabled = false;
    }
}
```

Enemy1ObjectPooler-skripti

```
using System.Collections.Generic;
using UnityEngine;

public class Enemy1ObjectPooler : MonoBehaviour
{
    public static Enemy1ObjectPooler SharedInstance;
    private List<GameObject> PooledObjects;
    public GameObject ObjectToPool;
    public int AmountToPool;
    public bool ShouldExpand = true;

    private void Awake ()
    {
        SharedInstance = this;
    }

    void Start()
    {
        PooledObjects = new List<GameObject> ();
        for (int i = 0; i < AmountToPool; i++)
        {
            GameObject obj = Instantiate (ObjectToPool);
            obj.SetActive (false);
            PooledObjects.Add (obj);
        }
    }

    public GameObject GetPooledObject ()
    {
        for (int i = 0; i < PooledObjects.Count; i++)
        {
            if (!PooledObjects[i].activeInHierarchy)
            {
                return PooledObjects[i];
            }
        }

        if (ShouldExpand)
        {
            GameObject obj = Instantiate (ObjectToPool);
            obj.SetActive (false);
            PooledObjects.Add (obj);
            return obj;
        }
        else
        {
            return null;
        }
    }
}
```

SoundManager-skripti

```
using System;
using UnityEngine;

public class SoundManager : MonoBehaviour
{
    public Sound[] Sounds;
    public static SoundManager Instance;


    private void Awake ()
    {
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            Destroy (gameObject);
            return;
        }

        DontDestroyOnLoad (gameObject);

        foreach (Sound sound in Sounds)
        {
            sound.Source = gameObject.AddComponent<AudioSource> ();
            sound.Source.clip = sound.Clip;

            sound.Source.volume = sound.Volume;
            sound.Source.pitch = sound.Pitch;
            sound.Source.loop = sound.Loop;
        }
    }

    public void Play (string name)
    {
        Sound s = Array.Find (Sounds, sound => sound.Name == name);
        if (s == null)
        {
            Debug.LogWarning ("Sound: " + name + " not found.");
            return;
        }

        s.Source.Play();
    }

    public void Stop (string name)
    {
        Sound s = Array.Find (Sounds, sound => sound.Name == name);
        if (s == null)
        {
            Debug.LogWarning ("Sound: " + name + " not found.");
            return;
        }

        s.Source.Stop ();
```

```
        }

        public void StopAllSounds ()
        {
            foreach (var s in Sounds)
            {
                s.Source.Stop ();
            }
        }
    }
```