Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Dan Huiko

Peli- ja mainospalvelupaketti mobiilipelien kaupallistamiseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

10.3.2020

Tekijä Otsikko Sivumäärä Aika	Dan Huiko Peli- ja mainospalvelupaketti mobiilipelien kaupallistamiseen 35 sivua 10.3.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Älykkäät järjestelmät ja ohjelmistot
Ohjaaja	Lehtori Antti Laiho

Tämän insinöörityön tavoitteena oli tutkia mobiilipelikehityksen yleisiä tarpeita ilmaispelien kaupallistamisen näkökulmasta ja kehittää sitä varten yleishyödyllinen ja yksinkertainen paketti Unity-pelimoottorille. Tällä paketilla tulisi aloittelevankin pelikehittäjän pystyä ottamaan kaupallistamiseen tarvittavat ominaisuudet käyttöönsä.

Tässä työssä käydään läpi ilmaispelien kaupallistamisen tarpeita ja haasteita sekä paneudutaan Googlen tarjoamiin rajapintoihin peli- sekä mainospalveluita varten Unity-pelimoottorille. Näitä hyödyntäen oli tavoitteena kehittää yhteen Unity-pakettiin kääritty toteutus mobiilikehityksen kaupallistamisen tärkeimmistä ominaisuuksista, joihin sisältyisivät sosiaaliset palvelut, eri tyyppiset mainokset sekä pilvitallennus.

Näiden palvelujen kehittämistä ja testaamista varten kehitettiin myös yksinkertainen peli. Tähän implementoitiin tässä työssä kehitetyt ominaisuudet niin kuin ne julkaisukelpoisissa peleissäkin implementoitaisiin, jotta saataisiin hyvä käsitys siitä, minkälaisia eri ominaisuuksia kehittäjät tältä rajapinnalta saattavat toivoa. Lopputuloksena saatiin aikaiseksi helposti käyttöönotettava paketti, jolla voidaan nopeasti implementoida mobiilipelisovelluksiin Googlen peli- ja/tai mainospalvelut.

Avainsanat	pelikehitys, mobiilipeli, kaupallistaminen, Unity

Author	Dan Huiko
Title	Game and ad service package for mobile game monetizing
Number of Pages	35 pages
Date	10 March 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Smart Systems
Instructor	Antti Laiho, Senior Lecturer

The goal of this final year project was to study the common needs of features of mobile game development from the perspective of free to play games and to develop an easy-to-use package with the features for the Unity game engine. With this package, even a novice game developer should be able to monetize their games.

This thesis examines the needs and challenges of monetizing free mobile games and studies the interfaces provided by Google for the gaming and advertising services for the Unity game engine. Using these, an implementation wrapped in one Unity package was developed. It had the most important features of mobile development commercialization, which includes social services, various types of advertising, and cloud saving.

A simple game was also created to develop and test the features of the services. For this game, the commercialization features that were developed were implemented as they would be in a published game, to get a proper understanding of what different features developers might want from the package. As a result of this project, an easy-to-deploy package was created which enables developers to easily implement Google's game and/or advertising services for mobile games.

Keywords	game development, mobile game, monetization, Unity

Sisällys

Lyhenteet ja käsitteet

GPGS	Google Play -pelipalvelut (engl. Google Play Game Services).
API	Ohjelmointirajapinta (engl. Application Programming Interface).
IAP	Sovelluksen sisäiset ostot (engl. In-app purchases).
UI	Käyttöliittymä (engl. User Interface).
Inspektori	Unity-pelimoottorin ikkuna, joka näyttää valittuun peliobjektiin kiinnitetyt komponentit.
Singleton	Ohjelmoinnissa käytetty suunnittelumalli, jolla varmistetaan, että tietystä luokasta on vain yksi instanssi.
Scene	Unityn tiedostomuoto pelinäkymille.

1 Johdanto

Nykypäivän mobiilipelikehitys on usein hyvinkin hektistä pelimarkkinoiden muuttuessa jatkuvasti. Kun mobiilipelejä kehitetään usein lähes liukuhihnamaisesti, on tarpeellista, että jokaiselle projektille vaadittavia samoja ominaisuuksia ei tarvitse kehittää joka kerta uudelleen. Tällaisia ovat ilmaismobiilipelien kehityksessä usein esimerkiksi kaupallistamisen vaatimat toiminnallisuudet. Vaikka maksullisia pelejä tehdään mobiilialustoille, ovat ilmaissovellukset huomattavasti suositumpi lähestymistapa kaupallistamiseen. Tästä kertoo jo paljon se, että vuonna 2016 Google Play -kaupassa vain kahdeksan prosenttia peleistä oli maksullisia [1].

Tämän insinöörityön tavoitteena oli tuottaa Unity-mobiilipelikehitykseen aikaa säästävä ja kehitystyötä helpottava paketti, joka sisältää monia nykypäivän ilmaismobiilipeleille kaupallistamisen kannalta lähes välttämättömiä ominaisuuksia. Tavoitteena oli sisällyttää tähän pakettiin Googlen mainospalvelu AdMob ja Googlen pelipalvelut. AdMobin avulla voidaan esittää yleisimmin käytössä olevia mainostyyppejä mobiilipeleissä. Googlen pelipalvelut taas mahdollistavat muun muassa käyttäjän todennuksen, saavutukset, tulostaulukot sekä pilvitallentamisen Android-alustalla.

Näiden palveluiden käyttämistä varten oli tarkoituksena kehittää mahdollisimman yksinkertainen rajapinta, jossa otettiin huomioon se, että tätä pakettia saattaa tulevaisuudessa käyttää muitakin kehittäjiä. Näiden ominaisuuksien kehitystä ja testaamista varten kehitettiin myös peli, jossa nämä ominaisuudet otettiin käyttöön käytännön ympäristössä.

2 Suunnittelu ja taustatietoa

Tämä työ tehtiin Unity-pelimoottoria varten, joka on vuonna 2004 julkaistu Unity Technologiesin kehittämä monipuolinen pelimoottori, joka tukee suurimpia pelialustoja, joita on nykypäivänä käytössä. Unityllä voidaan tuottaa niin 2D- kuin 3D-pelejä, erinäisiä ohjelmistoja ja jopa elokuvia. Jo yli 50 prosenttia kaikista mobiilipeleistä on tuotettu Unity-pelimoottorilla [2].

Projektin aluksi täytyi valita Unity-versio ja kun vain toisella projektiin tarvittavista laajennuksista on vaatimuksena vähintään versio Unity 5, valittiin sillä hetkellä viimeisin vakaa versio, joka oli 2018.3.0f2. Koska projektin lopullinen tuotos on tarkoitus tulla vapaaseen jakeluun, kirjoitettiin kaikki koodi ja koodin kommentit englanniksi. Tästä syystä tässä tekstissä viitataan useasti esimerkiksi englanninkielisiin funktioihin.

2.1 Google Play Game Services

Peleille ja varsinkin mobiilipeleille on yleisesti luonteenomaista sisältää joko yksi tai useampia tulostaulukoita, joissa pelaajat voivat kilpailla muiden pelaajien kanssa ympäri maailmaa. Ensimmäisen kerran videopelissä tällainen taulukko nähtiin vuonna 1976 julkaistussa Sea Wolf -pelissä [3], jonka jälkeen ne jäivätkin elämään, varsinkin tarinattomiin peleihin.

Myös saavutukset (engl. achievement) ovat yleinen ominaisuus, joilla voidaan saada esimerkiksi pidempiaikaisia pelaajia palaamaan takaisin peliin suorittamaan niitä. Muun muassa näitä palveluja varten on Apple luonut iOS käyttöjärjestelmälleen GameCenter-palvelun ja Google vastaavasti Androidille Google Play Game Services -palvelun.

Unity tarjoaa sosiaalisten palvelujen käyttöön Social-rajapinnan, joka sisältää GameCenter-toteutuksen, mutta ei GPGS-toteutusta. Tätä varten tuotiin projektiin Googlen kehittämä GPGS-Unity-laajennuspaketti. Tämä paketti mahdollistaa GPGS-rajapinnan kutsut käyttäen Unityn Social-rajapintaa. Versioksi valittiin 0.9.64, koska tämä oli sillä hetkellä uusin.

GPGS-paketti sisältää tuen seuraaville GPGS-rajapinnan ominaisuuksille:

- sisään- ja uloskirjautuminen
- saavutusten paljastus
- saavutusten asteittainen- sekä täysi avaaminen
- tulostaulukkoon lukujen lisäys
- tulostaulukon ja saavutusten oletuskäyttöliittymän näyttäminen
- pilvitallennusten kirjoitus ja luku
- tapahtumat
- pelin videotallentaminen
- vuoropohjainen sekä reaaliaikainen moninpeli.

Paketti sisältää tuen, muttei varsinaista rajapintaa näiden käyttöön. Tässä työssä tarkoituksena ei ollut kaikkien näiden ominaisuuksien toteutuksien luominen vaan keskityttiin niistä olennaisimpiin. Loput ominaisuuksista voi kehittäjä halutessaan itse toteuttaa, eikä niiden tukea ole riisuttu paketista. Paketti sisältää myös Google Play Services Resolver -liitännäisen, joka pitää huolen, että projektissa on tämän paketin käyttöön vaaditut Android-kirjastot.

2.2 Mainospalvelut mobiilipeleissä

Mainokset ovat yhdessä sovelluksen sisäisten ostojen (IAP) kanssa usein tärkein kaupallistamisen muoto nykypäivän mobiilipelikehityksessä. Niitä hyödyntämällä saadaan tuloja myös käyttäjiltä, jotka eivät ole valmiita maksamaan käyttämästään tuotteesta ennen asennusta, tai jos peli tai applikaatio on luonteeltaan sellainen, että siihen ei ole sopivaa lisätä sovelluksen sisäisiä ostoja.

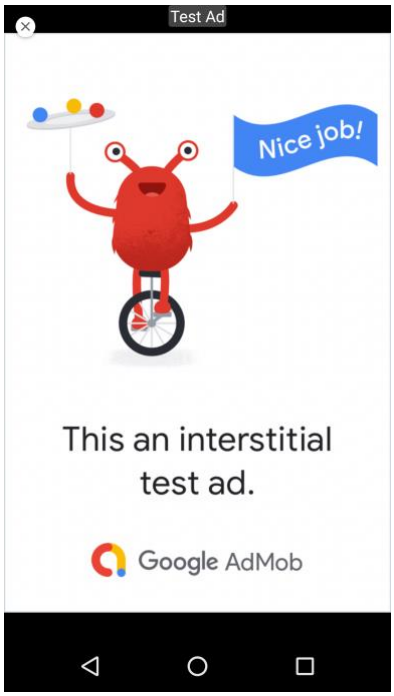
Bannerimainoksia on nähty web- ja mobiilisovelluksissa jo vuosikymmeniä. Ensimmäinen ilmestyi vain 3 vuotta World Wide Webin lanseerauksen jälkeen vuonna 1994 [4]. Vaikka mobiilimainonnassa bannerimainoksien suosio on laskenut videomainoksien ottaessa paikan suurimpana mainostusmuotona [5], löytyy bannerimainoksille vielä paikkansa mobiilipeleissä ja -applikaatioissa.

Kuva 1. Bannerimainos pelinäkymän alareunassa.

Bannerimainokset ovat mobiiliapplikaatioissa usein kuvan 1 mukaisia suorakaiteen muotoisia mainoksia joko näytön ylä- tai alareunassa, joissa esitetään staattisia kuvia ja/tai tekstiä.

Bannerimainokset sopivat näytettäväksi koska tahansa, vaikka pelin aikana. Pienen kokonsa vuoksi bannerimainokset eivät kuitenkaan ole kovinkaan houkuttelevia, vaan jäävät usein huomiotta, eivätkä siksi tuota kovinkaan hyvin. Tämä käy ilmi myös tästä projektista saadusta statistiikasta, kappaleessa 5.

Välimainokset ovat kuvan 2 mukaisia koko ruudun täyttäviä ja graafisesti paljon näyttävämpiä mainoksia, ja ne voivat olla joko staattisia tai videoita. Välimainoksia ei tulisi suinkaan näyttää milloin vain, varsinkaan kun puhutaan peleistä. Välimainoksen n§äyttäminen kesken pelin on kielletty Googlen käyttöehdoissa [6]. Lisäksi se voi jopa pahimmassa tapauksessa saada käyttäjän poistamaan pelin välittömästi laitteeltaan [7, s. 4]. Sen sijaan välimainokset ovat nimensä mukaisesti tarkoitettu näytettäväksi luonnollisissa siirtymä- tai välivaiheissa esimerkiksi siirryttäessä seuraavaan tasoon tai pelaajan pysäyttäessä pelin.



Kuva 2. Välimainos täyttää koko ruudun.

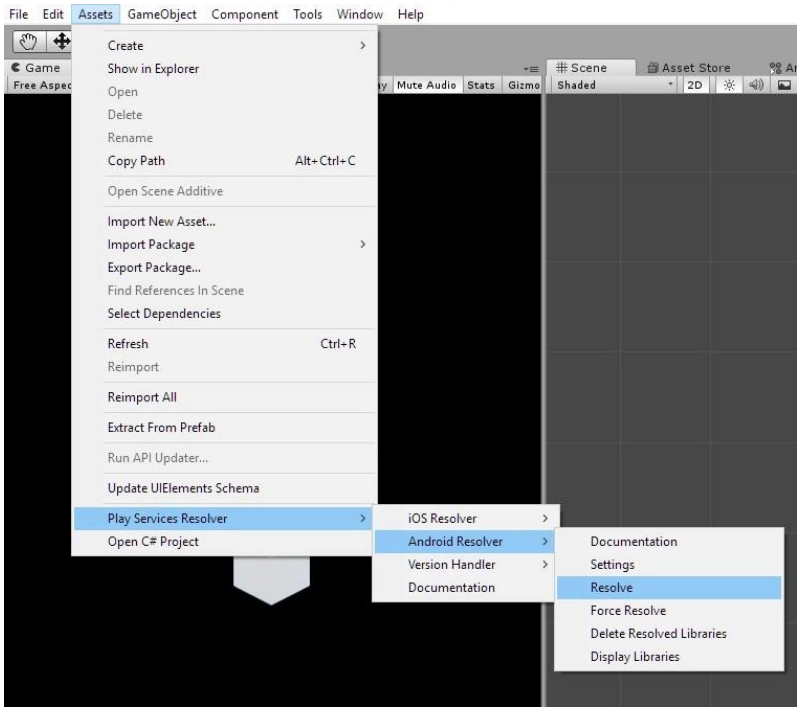
Palkituista mainoksista on tullut viime aikoina erittäin suosittuja pelien kaupallistamisessa, sillä käyttäjät jopa haluavat katsoa niitä. Tapjoyn vuonna 2019 tekemässä tutkimuksessa havaittiin, että vain 10 % 23-28-vuotiaista amerikkalaisista välttää palkittujen mainoksien katselua [8].

Kuva 3. Palkittu mainos tapana antaa pelaajalle sisältöä pelissä [9].

Googlen palkitut mainokset ovat jopa minuutin mittaisia [10] kuvan 3 tapaisia videomainoksia, jotka ovat käyttäjälle tapa maksaa esimerkiksi yllätyslaatikoista, pelin jatkumisesta tai vaikka nopeammasta edistymisestä omalla ajallaan rahan sijasta. Juuri tästä syystä palkittuja mainoksia ei saa näyttää ilman käyttäjän suostumusta, eli pelaajalle täytyy antaa täysi vapaus valita, katsooko mainoksen vai ei. Tärkeää olisi myös tuoda esille, mitä pelaaja katsomallaan mainoksellaan saa palkinnoksi.

Googlella on kaksi eri mainostuspalvelua: AdSense ja AdMob. Näistä AdSense on tarkoitettu verkkosovelluksiin ja AdMob mobiilisovelluksiin. Google tarjoaa Unity-kehitykseen omaa Google Mobile Ads Unity -laajennuspakettiaan, joka käyttää AdMob-alustaa.

Projektiin ladattiin ja tuotiin Google Mobile Ads versio 3.17.0 -paketti. Tämäkin paketti sisältää Unity Play Services Resolver -kirjaston, mutta se on uudempi versio kuin GPGS-paketin sisältämä. Jos projektiin tuo molemmat versiot, niistä ei välttämättä aiheudu virheilmoituksia ja projekti kääntyy, mutta rakennettuna sovellus ei toimi. Tämä aiheutti ongelmia kehitysvaiheessa, kun molemmat paketit toimivat erillään, mutta eivät keskenään. Tähän ratkaisu oli tuoda projektiin ainoastaan uudempi versio resolverista.



Kuva 4. Unity Play Service Resolverin “Resolve”-toiminto.

Kun molemmat paketit oli tuotu projektiin, ajettiin kuvan 4 mukainen resolverin ”resolve”-toiminto, joka kopioi molempien laajennuksien tarvittavat riippuvuudet projektin Assets/Plugins/Android-kansioon. Nämä riippuvuudet eivät tule sisältymään lopulliseen pakettiin, sillä ne ovat kooltaan melko isoja. Paketin käyttäjä saa ne itse samalla tapaa tällä resolve-toiminnolla.

3 Rajapinnan ja Unity-paketin toteutus

Kun tarvittavat paketit oli tuotu projektiin ja konfiguroitu, aloitettiin koodin kirjoittaminen. Jotta käytettävyys olisi mahdollisimman suoraviivaista ja selkeää, sijoitettiin kaikki luokat, joita kehittäjän tarvitsee kutsua Huiko-nimiavaruuteen (engl. namespace).

Näitä luokkia ovat:

- Ads
- SocialServices
- CloudSave.

Näiden lisäksi paketista löytyy Utils-luokka, johon sisältyy staattisia yleishyödyllisiä funktioita, joita muut Huiko-nimiavaruuden luokat käyttävät mutta joita ei ollut mielekästä sijoittaa johonkin tiettyyn luokkaan.

3.1 Sosiaaliset palvelut

Googlen sosiaalisia palveluita varten luotiin luokka nimeltä SocialServices, joka pitää sisällään kaiken toiminnallisuuden näihin palveluihin. Tarkoituksena oli luoda mahdollisimman käyttäjäystävällinen rajapinta, joten kaikki tämän luokan funktiot ovat staattisia, jotta niitä voidaan kutsua ilman instanssia. Lisäksi ne ovat helposti ymmärrettäviä nimiltään ja parametreiltään.

Vaikka moni tämän luokan funktioista kutsuu itse Social-luokan funktioita, joita kutsumalla suoraan saadaan sama lopputulos. Ne päätettiin kuitenkin sisällyttää tähän luokkaan, jotta kaikki sosiaaliset ominaisuudet olisivat saman luokan alla, eikä eroteltuina alustan mukaan.

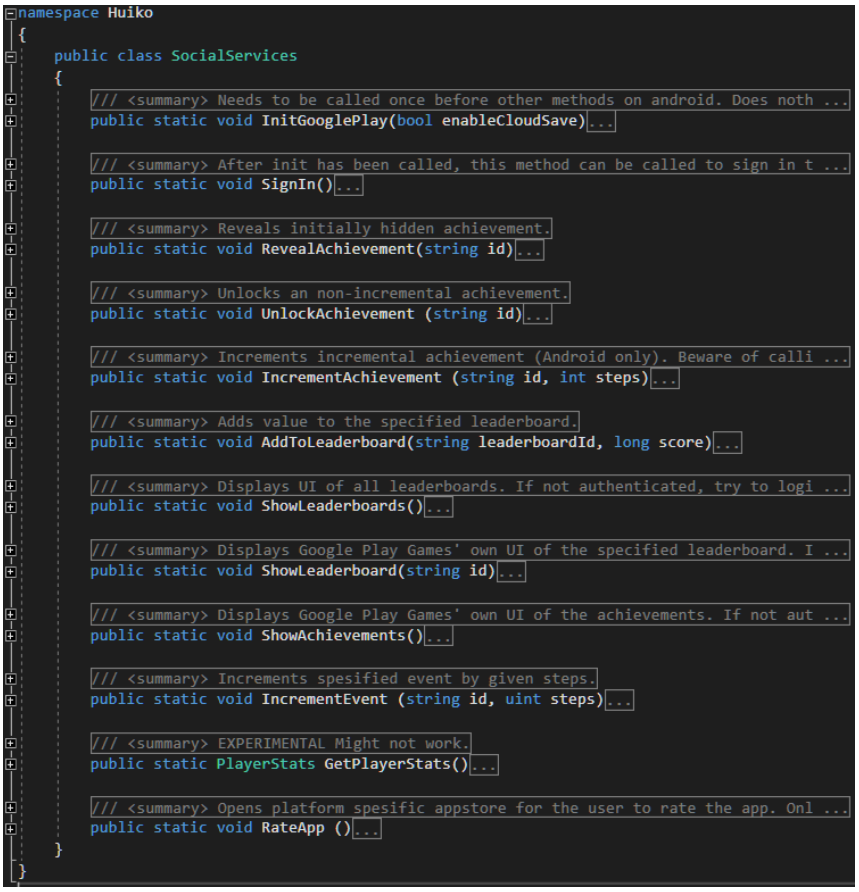
Jotta nämä toiminnallisuudet saadaan käyttöön, täytyy PlayGamesPlatform ensin aktivoida. PlayGamesPlatform on Unityn ISocialPlatform-toteutus, jonka aktivointi korvaa Unityn oletussosiaalialustan tällä toteutuksella. Koska tämä paketti sisältää myös tuen Googlen pilvitallennukselle, täytyy tässä vaiheessa se ottaa käyttöön. Se määritellään, kun luodaan uusi PlayGamesClientConfiguration objekti esimerkkikoodi

1:n mukaisesti. Kehittäjä voi päättää, haluaako hän ottaa pilvitallennuksen käyttöön kutsuessaan initialisointifunktiota antamalla tälle parametriksi binäärimuuttujan.

```
public static void InitGooglePlay(bool enableCloudSave)
{
#if UNITY_ANDROID
       PlayGamesClientConfiguration config;
       if (enableCloudSave)
       {
           config = new PlayGamesClientConfiguration.Builder()
               .EnableSavedGames()
               .Build();
       }
       else
       {
           config = new PlayGamesClientConfiguration.Builder()
               .Build();
       }
       PlayGamesPlatform.InitializeInstance(config);
       PlayGamesPlatform.Activate();
#endif
}
```

Esimerkkikoodi 1. Sosiaalisten palvelujen alustaminen.

Tämä koodi on SocialServices-luokan InitGooglePlay-funktio (kuva 5), jota täytyy kutsua kerran ajon aikana ennen kuin voidaan kutsua muita SocialServices-luokan funktioita. Kun PlayGamesPlatform on aktivoitu, korvaa se Unityn Social-luokan toteutuksen. Tämän vuoksi moni SocialServices-luokan funktioista toimii molemmilla alustoilla, sillä ne kutsuvat Social-luokan funktioita. Toisaalta osa toteuttaa vain Androidilla toimivia ominaisuuksia. Tällaiset funktiot käärittiin esikäsittelydirektiiviin, joka pitää huolen, että kyseiset funktio ei tee mitään iOS-alustalla.



```
namespace Huiko
{
    public class SocialServices
    {
        /// <summary> Needs to be called once before other methods on android. Does noth ...
        public static void InitGooglePlay(bool enableCloudSave)...

        /// <summary> After init has been called, this method can be called to sign in t ...
        public static void SignIn()...

        /// <summary> Reveals initially hidden achievement.
        public static void RevealAchievement(string id)...

        /// <summary> Unlocks an non-incremental achievement.
        public static void UnlockAchievement (string id)...

        /// <summary> Increments incremental achievement (Android only). Beware of calli ...
        public static void IncrementAchievement (string id, int steps)...

        /// <summary> Adds value to the specified leaderboard.
        public static void AddToLeaderboard(string leaderboardId, long score)...

        /// <summary> Displays UI of all leaderboards. If not authenticated, try to logi ...
        public static void ShowLeaderboards()...

        /// <summary> Displays Google Play Games' own UI of the specified leaderboard. I ...
        public static void ShowLeaderboard(string id)...

        /// <summary> Displays Google Play Games' own UI of the achievements. If not aut ...
        public static void ShowAchievements()...

        /// <summary> Increments spesified event by given steps.
        public static void IncrementEvent (string id, uint steps)...

        /// <summary> EXPERIMENTAL Might not work.
        public static PlayerStats GetPlayerStats()...

        /// <summary> Opens platform spesific appstore for the user to rate the app. Onl ...
        public static void RateApp ()...
    }
}
```

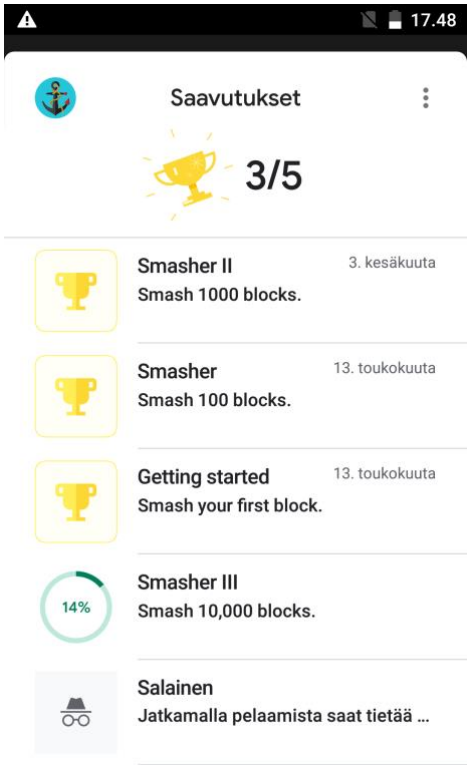
Kuva 5. SocialServices-luokan rakenne.

Aktivoinnin jälkeen voidaan kutsua kuvan 5 mukaista SignIn-sisäänkirjautumisfunktiota, joka toteuttaa sisäänkirjautumisen molemmille alustoille käyttäen Unityn Social-luokkaa. Onnistuneen sisäänkirjautumisen jälkeen on mahdollista kutsua kaikkia muita luokan funktioita. Kaikissa muissa luokan funktioiden alussa tarkastellaan, onko käyttäjä kirjautunut sisään, virheiden välttämiseksi. Toisin kuin RateApp-funktio, joka yksinkertaisesti avaa hyperlinkin pelikauppaan.

Saavutukset

Saavutuksia on iOS- ja Android-alustoilla yhteensä neljä erilaista: näistä kaksi on samanlaisia molemmilla alustoilla, paljastettu sekä piilotettu. Näiden lisäksi Android-alustalla on paljastetuista ja piilotetuista saavutuksista asteittaiset versiot.

Paljastettu saavutus on saavutus, jonka pelaaja voi nähdä, vaikkei olisi pelannut vielä ollenkaan. Piilotetut saavutukset ovat sellaisia, joiden olemassaolosta ilmoitetaan pelaajalle, mutta ei näytetä, mitä niiden suorittamiseen vaaditaan. Asteittaisista saavutuksista pelaaja näkee prosentteina, kuinka paljon saavutuksesta on suoritettu.



Kuva 6. Näkymä saavutuksista ISKOS-pelissä.

Kuvassa 6 on nähtävissä kaikki nämä eri saavutustyypit, osa suoritettuina, yksi osittain suoritettuna sekä yksi piilotettu suorittamaton saavutus.

Saavutusten käyttämistä varten luotiin SocialServices-luokkaan neljä funktiota: RevealAchievement, UnlockAchievement, IncrementAchievement sekä ShowAchievements (kuva 5). RevealAchievement eli saavutuksen paljastusfunktio ottaa parametrinä vain saavutuksen tunnusnumeron. Tällä tunnuksella, nolla-arvolla sekä binäärimuuttuja-tyyppisellä lambda-ilmaisulla, kutsutaan Social-rajapinnan ReportProgress-funktiota, jolla tarkastellaan, onnistuiko kutsu. Onnistunut kutsu

paljastaa kyseisen saavutuksen, muttei kasvata tai suorita sitä. Piilotettu saavutus paljastuu myös, jos se suoritetaan, eli sitä ei ole pakko ensin paljastaa.

UnlockAchievement-funktio suorittaa asteettoman saavutuksen, sen toteutus on muuten sama kuin paljastusfunktionkin, mutta ReportProgress-funktiokutsuun syötetään arvoksi 100.

Koska asteittaiset saavutukset ovat vain Android-alustalla, on myös niiden suorittamiseen tehty funktio IncrementAchievement hieman erilainen toteutukseltaan. Ensinnäkin se on käärityt esikäsittelydirektiiviin, joka pitää huolen, että vaikka sitä kutsuttaisiinkin muilla alustoilla, mitään ei kuitenkaan suoriteta. Se ottaa parametreikseen myös saavutuksen tunnusnumeron, mutta myös asteiden kasvatettavan määrän kokonaislukuna. Social-rajapinnasta ei siis löydy toteutusta asteittaiselle saavutukselle, joten täytyy kutsua PlayGamesPlatform-rajapinnan IncrementAchievement-funktiota. Sitä kutsutaan samalla tavalla kuin ReportProgress-funktiota, mutta sille syötetään parametrinä saatu lukuarvo. Tämä kasvattaa kyseistä saavutusta niin monta astetta kuin parametrinä annettiin.

Tätä funktiota tulee käyttää harkiten, eikä sitä tule kutsua esimerkiksi joka kerta, kun tehdään jokin toiminto, sillä jos tätä toimintoa toistetaan useasti pienen ajan sisällä, se voi olla laitteesta ja internetyhteyden laadusta riippuen todella raskasta. Näissä tapauksissa tulisi kerätä näiden toimintojen määrä muuttujaan ja antaa se parametrinä, kun sen arvo ylittää tietyn rajan tai kun pelissä tulee luonnollinen tauko.

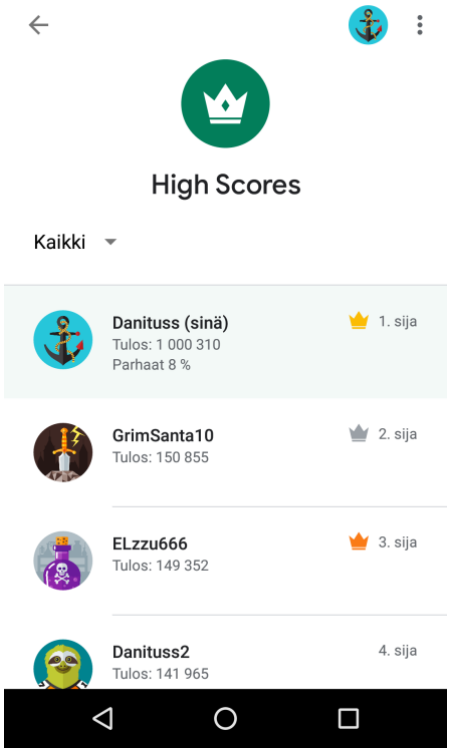
Tulostaulukot

Tulostaulukoiden toteutukseen tarvitaan käytännössä kaksi funktiota, yksi pisteiden lisäämiseen taulukolle sekä yksi taulukoiden näyttämiseen. Koska tulostaulukoita voi olla myös useampia ja niistä voidaan haluta näyttää vain tietty, tarvitaan tähän tehtävään vielä kolmas funktio.

Pisteiden lisääminen tulostaulukkoon tapahtuu AddToLeaderboard-funktiolla. Tämä funktio ottaa parametreinä tulostaulukon tunnusnumeron sekä lisättävät pisteet kokonaislukuna. Tässä funktiossa kutsutaan Social-luokan ReportScore-funktiota, johon

syötetään parametreinä saadut arvot ja tarkastellaan lambda-ilmaisussa kutsun onnistumista.

Tulostaulukoiden näyttämistä varten luotiin funktiot ShowLeaderboards sekä ShowLeaderboard. Näistä Showleaderboards kutsuu Social-luokan toteutusta, joka näyttää GPGS:n vakiotulostaulukkonäkymän, jossa näkyy kaikki taulukot, jos niitä on useampi kuin yksi.



Kuva 7. High Scores -tulostaulukko ISKOS-pelissä.

Kun halutaan esittää useista tulostaulukoista vain yksi, ilman että pelaaja joutuu itse avaamaan sen muiden seasta, voidaan käyttää ShowLeaderboard-funktiota. Se ottaa parametrinä yksittäisen tulostaulukon tunnusnumeron, jota käyttäen kutsutaan PlayGamesPlatform-luokan ShowLeaderboardUI-funktiota, koska sille ei löydy Social-luokasta toteutusta. Tämä avaa halutun tulostaulukon kuvan 7 mukaisella Google Playn vakio- UI:lla.

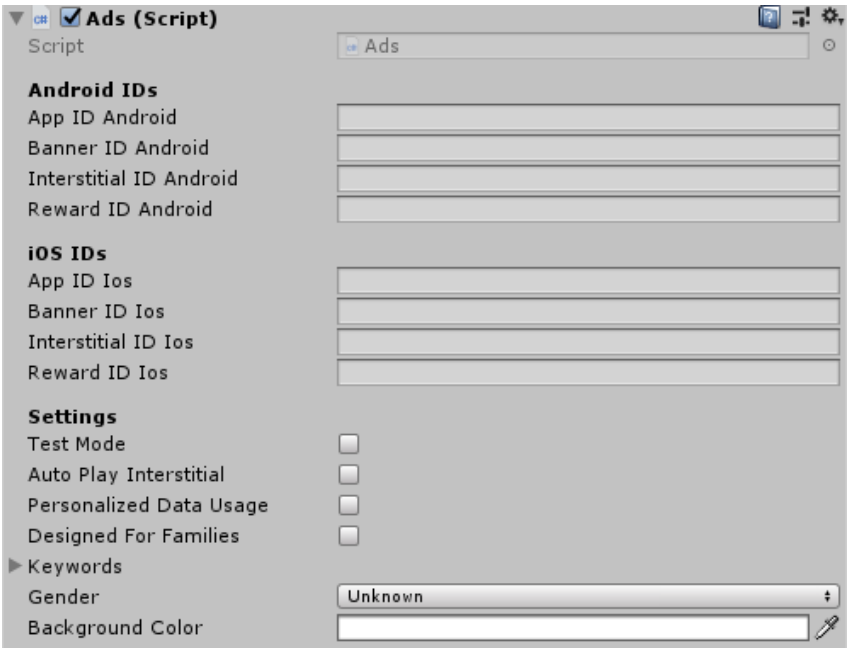
Ekstrat

Edellä mainittujen ominaisuuksien lisäksi SocialServices-luokka sisältää funktiot tapahtuman kasvattamiseen (IncrementEvent), pelaajan statistiikan tarkasteluun (GetPlayerStats) sekä applikaation arvioimiseen pelikaupoissa (RateApp).

3.2 Mainokset

Mainosten konfigurointi ja käyttöönotto

Mainospalveluita varten luotiin Ads-luokka. Koska se vaatii käyttäjältä hieman konfiguroimista, perii se MonoBehaviour-luokan, jolla vältetään kovakoodaamisen tarve, kun konfiguraatiot voidaan tehdä jokaisessa projektissa editorissa (kuva 8). Tästä syystä Ads-scripti täytyy kiinnittää komponenttina peliobjektiin. Tähän olisi myös elegantimpi ratkaisu, mutta ajan puutteessa se jäi kuitenkin tulevaisuuden suunnitelmaksi. Siihen palataan myöhemmin luvussa 6.



Kuva 8. Ads-luokka editorissa.

Luokasta on paljastettu Unityn inspektori-näkymään käyttäjän määrityksiä varten kaikki mahdolliset AdMobin tunnusnumerot. Näistä mikään ei ole pakollinen, vaan käyttäjä voi määritellä ne tarpeidensa mukaan, esimerkiksi vain Android-sovellus- ja banneri-tunnusnumerot. Nämä tunnusnumerot käyttäjä saa luomalla AdMob-projektin sekä siihen haluamansa mainosyksiköt.

Näiden lisäksi näkyvissä on muutamia hyödyllisiä asetuksia. Jos Test Mode -asetuksen asettaa päälle, määrittelee se tunnusnumeroiksi AdMobin testinumerot, jolloin näytettävät mainokset eivät ole oikeita vaan testimainoksia. Tähän luotiin vielä turvaominaisuus, joka asettaa Test Mode -asetuksen päälle, jos Unity-projektista rakennettiin sovellus kehitystilassa. Lisäksi näkymästä löytyy määrityksiä mainoskutsuja varten.

Kun Ads-scripti on kiinnitetty aktiiviseen peliobjektiin, pelin käynnistyessä initialisoi se itsensä käyttäjän tekemien konfiguraatioiden mukaisesti. Siitä tehdään myös singleton, eli objekti, josta voi olla vain yksi instanssi, jota kutsutaan Instance-nimisestä muuttujasta.

Mainosyksiköt ja mainoskutsu

AdMob tarjoaa neljä erilaista mainostuksen muotoa, joita kutsutaan mainosyksiköiksi. Näitä ovat:

- bannerimainos
- välimainos
- palkittu mainos
- natiivimainos.

Tässä työssä implementointiin näistä vain kolme ensimmäistä, sillä natiivimainoksien Unity-tuki on kirjoitushetkellä vasta kehitysasteella.

Jotta mainoksia voidaan ladata, tarvitsee mainos mainoskutsun, joka pitää sisällään kaiken sen tiedon, millaista mainosta kutsutaan. Tämä on hyvin tärkeää esimerkiksi lapsille suunnatun sisällön mainoksissa. Toisaalta on myös kaikille osapuolille suotuisampaa, kun näytetty mainos kohtaa oikean yleisön.

```
adRequest = new AdRequest.Builder()
      .AddKeywords(keywords)
      .SetGender(gender)
      .TagForChildDirectedTreatment(designedForFamilies)
      .AddExtra("tag_for_under_age_of_consent", personalizedDataU
      sage.ToString())
      .AddExtra("color_bg", Utils.ColorToHex(backgroundColor))
      .Build();
```

Esimerkkikoodi 2. Mainoskutsun luominen.

Ads-luokassa mainoskutsut on toteutettu luomalla objekti yksityiseen AdRequest-tyyppiseen luokkamuuttujaan luokan alustusvaiheessa esimerkkikoodin 2 mukaisesti. Tähän objektiin asetetaan kaikki käyttäjän inspektorissa määrittelemät asetukset. Tätä mainoskutsua käytetään parametrinä aina kun kutsutaan jokaisen mainosyksikön latausfunktiota, jolloin kutsuttava mainosyksikkö saa käyttäjän määrittelemät asetukset.

Bannerimainos

Bannerimainosta ei tarvitse ladata etukäteen kuten muita mainostyyppejä, vaan sille voitiin luoda yksi funktio ”ShowBanner”, jota kutsumalla saadaan uusi bannerimainos näkyviin. Tästä tosin luotiin kaksi versiota: parametriton sekä parametrillinen, joka ottaa parametreikseen AdSize-objektin sekä AdPosition-enumin. Näistä parametrittomalle on kovakoodattu kooksi AdSize.SmartBanner sekä sijainniksi AdPosition.Bottom.

Tämä ShowBanner-funktio tarkistaa ensin, ovatko mainokset käytössä, jonka jälkeen tuhotaan mahdollinen aikaisempi bannerimainos. Tämän jälkeen luodaan uusi BannerView-tyyppinen mainosyksikköobjekti, joka asetetaan julkiseen luokkamuuttujaan, jotta käyttäjä voi asettaa omia toteutuksia tapahtumakäsittelijöihin. Tämän objektin konstruktorille annetaan parametreiksi bannerimainoksen tunnusnumero, koko sekä sijainti. Tähän objektiin määritellään tapahtumakäsittelijät tapahtumille, joita ovat

- OnAdLoaded
- OnAdFailedToLoad
- OnAdOpening
- OnAdClosed

- OnAdLeavingApplication.

Tämän jälkeen tästä objektista kutsutaan LoadAd-funktiota käyttäen aiemmin luotua mainoskutsua parametrinä, joka lataa ja esittää bannerimainoksen.

Bannerimainoksen poistoa varten luotiin funktio RemoveBanner, joka kutsuu BannerView-objektin Destroy-funktiota, jos se on olemassa. Jos Banneri halutaan vain päivittää, voidaan kutsua uudestaan ShowBanner-funktiota, joka poistaa vanhan ja luo uuden mainoksen tämän tilalle. Bannerimainoksen päivittäminen onkin usein järkevää tietyin väliajoin, jotta käyttäjälle tulee tarjottua mahdollisimman paljon häntä koskettavia mainoksia.

Välimainos

Välimainos luodaan samalla tavalla kuin bannerimainoskin, mutta sen luonti tehdään LoadInterstitial-funktiossa. Sitä ei esitetä suoraan ladattaessa, ellei käyttäjä ole asettanut autoPlayInterstitial-muuttujaa todeksi. Tämän muuttujan tilaa tarkastellaan OnAdLoaded-tapahtumakäsittelijässä, joka kutsuu ShowInterstitial-funktiota ollessaan tosi. Muussa tapauksessa täytyy käyttäjän kutsua itse ShowInterstitial-funktiota halutessaan näyttää mainos. Hyvä tapa onkin ladata mainos ennen kuin sitä tarvitaan, jotta se on varmasti valmis esitettäväksi silloin, kun ShowInterstitial-funktiota kutsutaan.

Palkittu mainos

Kuten välimainos, tulee palkittu mainos ladata ennen kuin se voidaan esittää. Kuten banneri- ja välimainoksissa, palkittujen mainoksien luominen tapahtuu lähes samalla tavalla. Ensin luodaan RewardedAd-tyyppinen mainosyksikköobjekti, johon asetetaan tapahtumakäsittelijät. Palkituilla mainoksilla tosin on vielä yksi uniikki tapahtumakäsittelijä, OnUserEarnedReward. Tällä on parametrina Reward-tyyppinen objekti, joka sisältää muun muassa mainosyksikköön määritellyn lukuarvollisen palkinnon.

Tämän jälkeen ladataan mainos käyttäen samaa mainoskutsua kuin muissakin mainosyksiköissä. Luonnin jälkeen voidaan mainos näyttää kutsumalla

ShowRewardedAd-funktiota, joka tulee aina asettaa pelissä johonkin nappiin, jotta pelaaja voi itse halutessaan katsoa sen.

Jos palkituissa mainoksissa halutaan käyttää yksinkertaista lukuarvollista palkintoa, esimerkiksi pelin sisäistä valuuttaa, voidaan se antaa yksinkertaisesti asettamalla se muuttujaan OnUserEarnedReward-tapahtumakäsittelijässä.

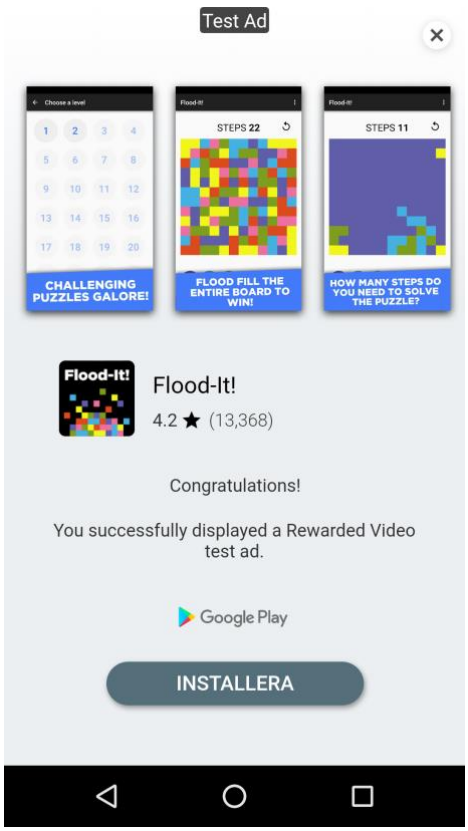
Monesti palkituilla mainoksilla ei kuitenkaan ole tällaista lukuarvollista palkintoa, vaan ne voivat olla lähestulkoon mitä tahansa. Tällaisissa tapauksissa kehittäjän tarvitsee itse kirjoittaa vielä mitä tulisi tapahtua, kun palkittu mainos on katsottu. Tässä tulee huomioida että näitä tapahtumakäsittelijöitä ei ajeta Unityn säikeissä, joten niissä ei voi kutsua Unityn API:a. Tällaisissa tapauksissa voidaan esimerkiksi asettaa tapahtumakäsittelijässä binäärimuuttuja, jota voidaan pollata Unityn Update-funktiossa. Esimerkkikoodi 3 on esimerkkitapa toteuttaa palkitseminen, kun palkintona on jotain muuta kuin lukuarvollinen palkinto.

```
bool rewarded, rewardedClosed;

public void UserOptedToWatchAd ()
{
      Ads.Instance.rewardedAd.OnUserEarnedReward += RewardedAdComplete;
      Ads.Instance.rewardedAd.OnAdClosed += RewardedAdClosed;

      Ads.Instance.ShowRewardedAd();
}

public void RewardedAdComplete (object sender,
                                 GoogleMobileAds.Api.Reward reward)
{
      rewarded = true;
}

public void RewardedAdClosed (object sender, EventArgs args)
{
      rewardedClosed = true;
}

public void Update ()
{
      if (rewardedClosed)
      {
            if (rewarded)
            {
                  // Give the reward here.
            }
            else
            {
                  // Handle case where the ad was not watched completely.
            }
            rewarded = false;
```

```
                rewardedClosed = false;
            }
        }
```

Esimerkkikoodi 3. Palkitun mainoksen käyttäminen pelissä.

Tässä koodissa on neljä funktiota, joista UserOptedToWatchAd ajetaan, kun pelaaja painaa nappia halutessaan katsoa palkitun mainoksen. Siinä asetetaan palkitun mainoksen tapahtumakäsittelijöiksi funktiot kahdelle tapahtumalle sekä esitetään kyseinen mainos. Toinen näistä tapahtumakäsittelijöistä on OnUserEarnedReward, johon asetetaan RewardedAdComplete-funktio. Tätä tapahtumaa kutsutaan, kun pelaaja on katsonut mainoksen loppuun asti, joka tässä tapauksessa asettaa vain rewarded-binäärimuuttujan todeksi. OnAdClosed-tapahtumaan asetetaan RewardedAdClosed-funktio ja se ajetaan, kun pelaaja sulkee mainoksen, kuvassa 9 näkyvästä napista ruudun oikeassa yläreunassa.



Kuva 9. Palkittu mainos palkinnon saamisen jälkeen.

Tämä funktio asettaa toisen binäärimuuttujan, rewardedClosedin todeksi, jota Update-funktio pollaa, jotta voidaan kutsua Unity API:n funktioita.

3.3 Pilvitallennus ja sovelluksen sisäiset ostot

Mobiilipeleissä sovelluksen sisäiset ostot ovat usein erillisten lisäsisältöjen sijaan erikoisvaluuttaa, jota pelaaja voi ostaa ja käyttää haluamallaan tavalla. Tällaista valuuttaa myydään yleensä monissa erikokoisissa paketeissa, ja ne maksavat yleensä muutamasta eurosta jopa yli sataan euroon. Google Play -kaupassa hintahaarukka on kuitenkin 0,50-350 euroa [11]. Nämä paketit ovat erillisiä tuotteita jakelukaupoissa. Tällainen tapa on kehittäjällekin helpompi toteuttaa, kun erilaisista tuotteista ei tarvitse tehdä erikseen maksullisia tuotteita jakelukauppoihin.

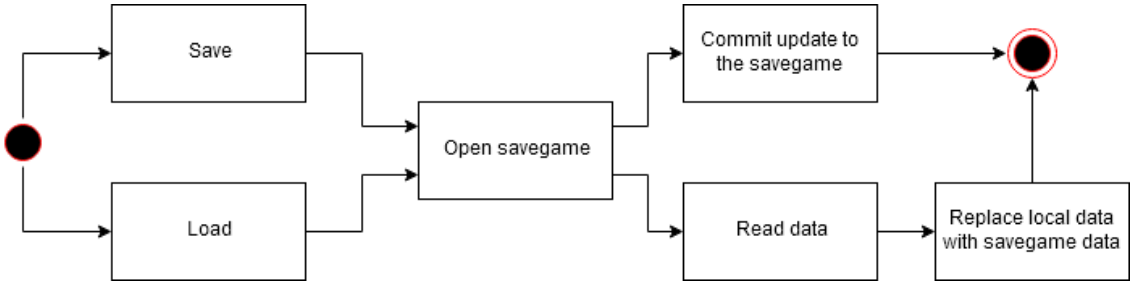
Tätä varten tarvitaan pilvitallennusjärjestelmä, jotta pelaajan ostama erikoisvaluutta pysyy tämän tilillä, vaikka tämä poistaisi paikalliset tallennustiedostot tai vaihtaisi uuteen mobiililaitteeseen. Googlen pelipalvelut sisältää tallennetut pelit -ominaisuuden, jota voidaan käyttää tähän tarkoitukseen sekä minkä tahansa pelin sisäisen datan, esimerkiksi pelin etenemisen tallentamiseen.

Tätä tallennusjärjestelmää varten luotiin kaksi luokkaa, CloudSave ja SaveData, joista CloudSave kuuluu Huiko-nimiavaruuteen, mistä kehittäjä voi yksinkertaisesti kutsua Save- tai Load-funktioita tallentaakseen tai ladatakseen tallennusdatan.

Googlen tallennusjärjestelmä, joka toteutettiin CloudSave-luokkaan, ottaa tallennusdatan tavukokoelmana (engl. byte array). Tästä syystä oli yksinkertaisin tapa toteuttaa datan hallinnointi luomalla sarjallistettava (engl. serializable) luokka. Tämä SaveData-luokka pitää sisällään kaiken, jonka kehittäjä haluaa tallennettavan. Luokan muodostin hakee itseensä pelin sen hetkisen datan, jolloin yksinkertaisesti luomalla uusi SaveData-objekti saadaan tallennusta varten päivitetty data.

Kun halutaan tallentaa tai ladata pelitallenne, avataan ensin ISavedGameClient-tyyppinen pelitallenneobjekti. Tämä saadaan automaattisesti joko verkosta tai paikallisesta muistista riippuen siitä, onko laite yhdistetty verkkoon. Tätä avatessa asetetaan binäärimuuttujan arvo sen mukaan, ollaanko lataamassa vai tallentamassa.

Kun tallenne on avattu, voidaan tähän joko kirjoittaa uusi data ja kommitoida se, tai lukea tämän data ja käsitellä sitä (kuva 10).



Kuva 10. Yksinkertaistettu kaavio tallennusjärjestelmästä.

Kutsuttaessa CloudSave-luokan Save-funktiota luodaan uusi SaveData-objekti, joka muutetaan ensin JSON-merkkijonoksi, jonka jälkeen se muutetaan tavukokoelmaksi. Tämä tavukokoelma annetaan CloudSave-luokan WriteData-funktiolle parametrinä, jota Save kutsuu. Tämä avaa pelitallenteen tallennustilassa käyttäen automaattista konfliktin käsittelijää, joka päättää, valitaanko vanha data välimuistista vai pilvestä. Tämä valitaan pisimmän peliajan mukaan, joka saadaan tallenteen metadatasta. Kun tallenne on avattu onnistuneesti, rakennetaan uusi tallenne uudella datalla esimerkkikoodin 4 mukaisesti, joka lopuksi kommitoidaan ja joka korvaa vanhan tallenteen.

```
private void CommitData(ISavedGameMetadata game, byte[] data)
{
      ISavedGameClient savedGameClient =
      PlayGamesPlatform.Instance.SavedGame;

      SavedGameMetadataUpdate.Builder builder
            = new SavedGameMetadataUpdate.Builder();

      byte[] pngData = GetScreenshot().EncodeToPNG();

      builder = builder
            .WithUpdatedDescription("Saved game at " + DateTime.UtcNow)
            .WithUpdatedPlayedTime(game.TotalTimePlayed.Add
                  (new TimeSpan(0, 0, (int)Time.realtimeSinceStartup)))
            .WithUpdatedPngCoverImage(pngData);

      SavedGameMetadataUpdate updatedMetadata = builder.Build();

      savedGameClient.CommitUpdate
      (game, updatedMetadata, data, OnSavedGameWritten);
}
```

Esimerkkikoodi 4. Funktio, joka rakentaa ja kommitoi uuden tallenteen.

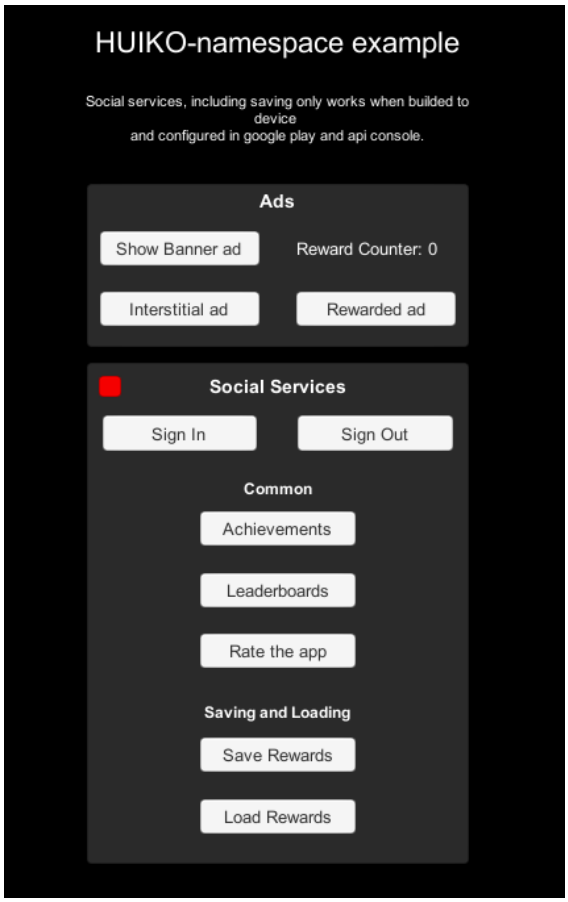
Load-funktion kutsuminen vastaavasti avaa tallenteen, mutta lataustilassa. Kun tallenne on auki, voidaan siitä lukea binääridata, joka saadaan tavukokoelmana. Tämä tavukokoelma muutetaan JSON-merkkijonoksi, joka taas muutetaan SaveData-tyyppiseksi objektiksi, joka asetetaan SaveData-tyyppiseen luokkamuuttujaan. Tämän jälkeen siitä kutsutaan SetData-funktiota, joka asettaa kehittäjän määrittelemän datan pelin muuttujiin.

CloudSave-luokkaan toteutettiin myös kehittäjälle paljastetut tapahtumakäsittelijät onnistuneelle tallentamiselle sekä lataamiselle. Näillä kehittäjä voi esimerkiksi päivittää ladatun datan pelinäkymään. Nämä toimivat samalla syntaksilla kuin mainosten tapahtumakäsittelijät yhtenäisyyden säilyttämiksesi.

3.4 Unity-paketti

Tämän projektin perimmäinen tarkoitus oli lopulta luoda tästä kaikesta edellä mainitusta paketti, joka voitaisiin tuoda eri projekteihin ja ottaa käyttöön mahdollisimman nopeasti. Muun muassa tällaisia paketteja varten Unitystä löytyy unitypackage-tiedostoformaatti, johon voidaan pakata kaikki assetit, ohjelmakoodit, kirjastot sekä muut mahdolliset tiedostot. Tämä pakattu tiedosto voidaan avata missä tahansa Unity-projektissa, joka purkaa sen kyseisen projektin Assets-kansioon.

Loin paketin nimeltä HUIKO, joka sisältää Google Mobile Ads ja GPGS -pluginit, Play Services Resolverin sekä näiden käyttöön kehitetyn rajapinnan. Näiden lisäksi kehitin pakettiin esimerkkiscenen (kuva 11), jonka avulla tämän paketin käyttäjä voi tutustua rajapinnan ominaisuuksiin.



Kuva 11. Esimerkkiscene, joka sisältyy HUIKO-unitypakettiin.

Tähän sceneen rakennettiin UI-pohjaisia nappeja, joilla voidaan testata rajapinnan tärkeimpiä ominaisuuksia. Kaikkien näiden nappien toiminallisuus toteutettiin yhteen Example-luokkaan. Koska koko rajapinta toteutettiin niin, että suurinta osaa toiminnallisuuksista voi käyttää yhdellä rivillä koodia, on tämä luokka suurelta osin hyvin yksinkertainen.

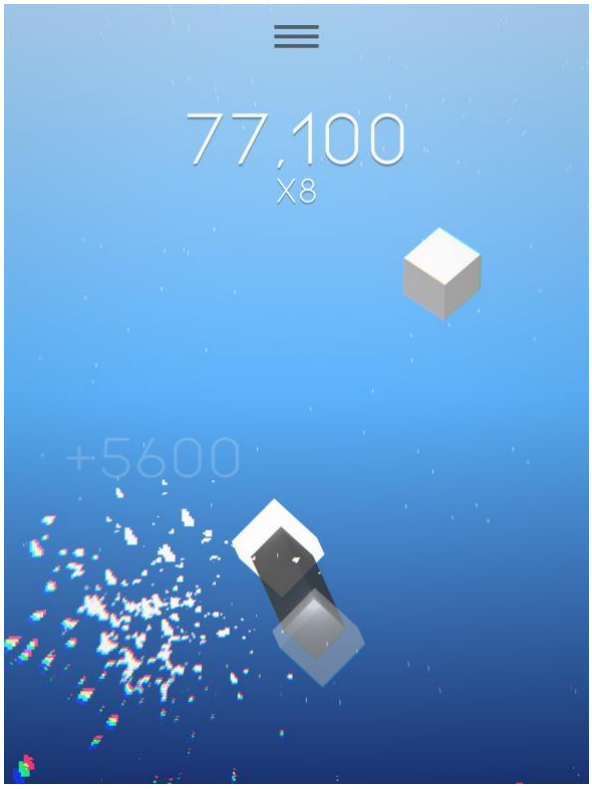
Koska mainosten tapahtumakäsittelijät olivat kehitysvaiheessa suuri kompastuskivi, tähän esimerkkiin tehtiin mahdollisimman selväksi, kuinka väli- sekä palkintomainosten tapahtumia käytetään. Tätä varten tähän luokkaan toteutettiin toiminallisuus, joka muuttaa joko väli- tai palkintomainoksen napin värin vihreäksi, kun kyseinen mainosyksikkö on ladattu. Samaan tapaan se muutetaan punaiseksi, kun mainos suljetaan. Myös palkittujen mainosten palkinto annetaan sitä säilyttävään luokkamuuttujaan sekä päivitetään sitä esittävään tekstikenttään.

Tallennusjärjestelmän testausta varten käytettiin hyväksi palkitun mainoksen palkintoa, joka voidaan tallentaa ja ladata kunhan käyttäjä on kirjautunut sisään. Kun tallenne on ladattu, data päivitetään sitä esittävään tekstikenttään käyttäen OnSaveLoaded-tapahtumakäsittelijää.

4 Paketin käyttöönotto testisovelluksessa

4.1 ISKOS-mobiilipeli

ISKOS on yksinkertainen mobiilipeli, jonka perimmäinen tarkoitus oli toimia kehitysalustana tämän paketin toteutusta varten. Tämä osoittautui hyväksi lähestymistavaksi kehittämiseen, sillä siitä saatiin tärkeää tietoa, minkälaisia ominaisuuksia mobiilipelit tarvitsevat todellisessa ympäristössä esimerkiksi mainosten sekä pelin sisäisten tuotteiden toteutusta varten.



Kuva 12. Kuvakaappaus ISKOS-pelin pelinäkymästä.

Pelin ideana on tuhota tippuvia laatikoita pyörittämällä kaksipäistä palkkia, jonka aktiivinen pää vaihtuu jokaisella lyönnillä (kuva 12). Laatikoita tuhoamalla kerätään pisteitä pelin nopeutuessa niin kauan, kunnes pelaaja ei pysy enää mukana, jolloin peli loppuu.

4.2 Käyttöönotto Unityssä

Jotta tämä paketti voidaan ottaa käyttöön, tarvitsee käyttäjän tehdä muutama toimenpide. Ensimmäisenä tulee vaihtaa projektin alusta Androidiksi, jonka jälkeen voidaan tuoda HUIKO-paketti projektiin.

Sosiaalisten palveluiden käyttöönottoa varten täytyy ajaa Unityn window-valikosta Google Play Games->Setup->Android Setup. Jos tämä ei tule näkyviin, tulee projekti avata uudestaan. Tämä avaa ikkunan, johon täytyy syöttää Google Play -konsolin generoimat projektikohtaiset Android-resurssit (esimerkkikoodi 5). Tätä varten tarvitsee siis luoda Google Play -konsoliin uusi projekti, jos sellaista ei ennestään ole, johon määritellään halutut pelipalveluominaisuudet. Tämä luo projektiin GPGSIds-nimisen scriptin, josta löytyy Google Play -konsoliprojektiin määritellyt projektispesifit staattiset tunnusnumerot projektin nimelle, saavutuksille, tulostaulukoille sekä tapahtumille. Näitä tunnusnumeroita käytetään kutsuttaessa SocialServices-luokan funktioita.

```
<?xml version="1.0" encoding="utf-8"?>
<!--
Google Play game services IDs.
-->
<resources>
  <!-- app_id -->
  <string name="app_id" translatable="false">603418033007</string>
  <!-- package_name -->
  <string name="package_name"
translatable="false">com.Test.Testi</string>
  <!-- achievement Test achievement -->
  <string name="achievement_test_achievement"
translatable="false">CgkI7_aR9McREAIQAw</string>
  <!-- leaderboard Test Scoreboard -->
  <string name="leaderboard_test_scoreboard"
translatable="false">CgkI7_aR9McREAIQAA</string>
  <!-- event Test event -->
  <string name="event_test_event"
translatable="false">CgkI7_aR9McREAIQBA</string>
</resources>
```

Esimerkkikoodi 5. Esimerkki GPGS-resurssit.

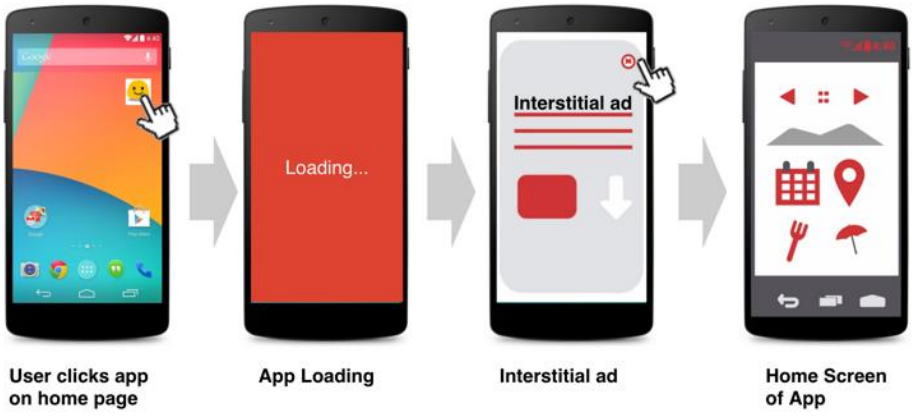
Kun Android Setup on suoritettu, käynnistää se automaattisesti resolverin, joka tuo projektiin tarvittavat kirjastot. Tämä tosin saattaa jäädä jumiin. Jos näin käy, voi resolver-ikkunan sulkea ja ajaa sen manuaalisesti kuvan 4 mukaisesti. Tämän jälkeen kaikki tarvittava Unityn puolella on tehty ja tämän toimivuutta voidaan testata paketin sisältämässä testiscenessä, kun se ajetaan Android-laitteelle.

4.3 Erilaiset käyttötavat ISKOS-pelissä

Eri mainosyksiköiden käyttötavat

ISKOS-peliin otettiin käyttöön kaikki tässä työssä läpikäydyt mainosyksiköt, kuitenkin siten, ettei mainoksia ole niin paljon, että ne vaikuttaisivat negatiivisesti pelaajan pelikokemukseen.

Välimainoksia käytetään pelissä kuvan 13 mukaisella tavalla, eli peliä käynnistäessä avautuu latausnäkymä, joka alustaa pelin järjestelmiä, muun muassa mainokset sekä pelipalvelut. Samalla yritetään ladata välimainos, ja jos se ehtii latautua kolmen sekunnin aikana, esitetään se pelaajalle. Suljettuaan mainoksen tai tämän kolmen sekunnin jälkeen siirrytään automaattisesti pelin päävalikkoon (kuva 15).



Kuva 13. ISKOS-pelissä käytetty mainostustapa välimainoksella [6].

Kun pelaaja siirtyy päävalikosta pelinäkymään, lataa se bannerimainoksen, joka tulee näkyviin heti, kun se on ladattu. Tämä bannerimainos sijaitsee ruudun alareunassa, kuitenkin siten, ettei se peitä mitään pelillisiä elementtejä kuvan 6 tapaan.

Palkittua mainosta käytetään ISKOS-pelissä antamalla pelaajalle mahdollisuus jatkaa peliä yhden kerran hävittyään. Kun pelaaja häviää pelin, tulee hänelle näkyviin kuvan 14 mukainen aikarajallinen nappi, jonka toiminnallisuus on selitetty tekstillä.

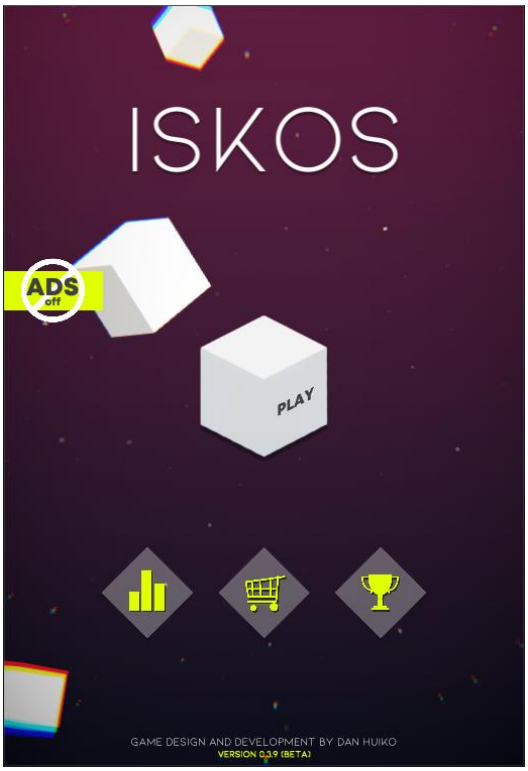
Kuva 14. Pelaajalle näytettävä valinta palkitun mainoksen katsomiseen.

Painettuaan nappia avautuu mainos, jonka kokonaan katsottuaan saa pelaaja jatkaa pelaamista siitä kohdasta, mihin aiemmin jäi.

GPGS-palvelut pelissä

Heti sovelluksen käynnistyttyä alustetaan pelipalvelut ja yritetään automaattista kirjautumista. Jos pelaaja ei ole kirjautunut sisään tai peli käynnistetään ensimmäistä kertaa, pelaajalle esitetään Google Playn kirjautumisnäkymä, josta pelaaja voi valita haluamansa käyttäjän. Pelaajan täytyy myös päättää, ottaako käyttöön automaattisen kirjautumisen.

Peliin toteutettiin googlen pelipalveluilla yksinkertainen huipputulostaulukko, johon tallennetaan automaattisesti pelin loppuessa pelaajan huipputulos. Tätä taulukkoa voi tarkastella painamalla sitä osoittavaa nappia päävalikossa (kuva 15), joka avaa kuvan 4 mukaisen näkymän.

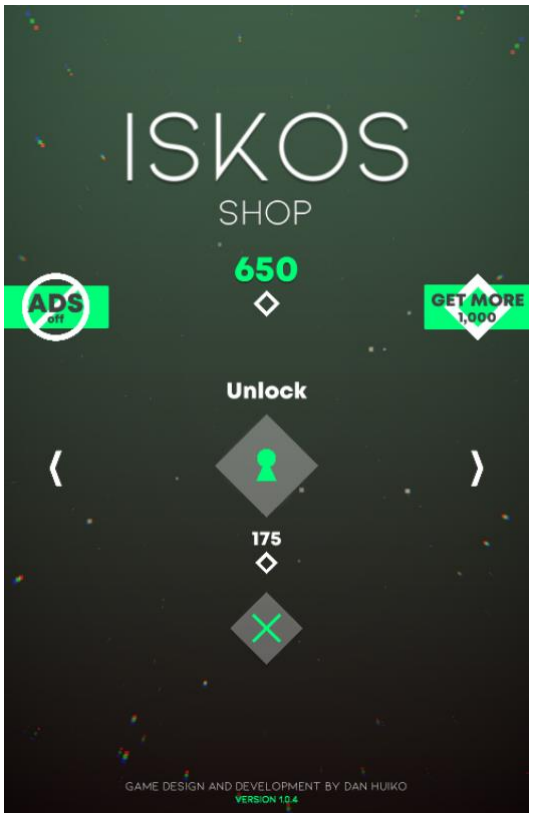


Kuva 15. ISKOS-mobiilipelin päävalikkonäkymä

Myös saavutuksia tehtiin hyväksikäyttäen kaikkia GPGS:n erityyppisiä saavutuksia. Näitä on yksi perusmuotoinen paljastettu saavutus, jonka pelaaja saa tuhotessaan ensimmäisen laatikon. Lisäksi pelistä löytyy asteittaisia saavutuksia eri määrille tuhottuja laatikoita sekä yksi piilotettu saavutus, joka aukeaa, kun pelaaja katsoo ensimmäisen kerran palkitun mainoksen. Näitä saavutuksia ja niiden edistymistä voi myös tarkastella päävalikosta (kuva 15) löytyvän napin takaa, joka avaa kuvan 3 mukaisen näkymän.

Pilvitallentaminen pelissä

Peliin toteutettiin kuvan 16 mukainen kauppa, josta on mahdollista ostaa pelin sisäisiä tuotteita, joko pelaamalla saaduilla tai oikealla rahalla ostettavilla timanteilla. On ehdottoman tärkeää, että nämä timantit tai niillä ostetut tuotteet säilyvät pelaajalla, jos pelin tiedot katoavat hänen laitteeltaan. Hyvä ratkaisu tähän on tallentaa nämä tiedot pilveen, jossa ne ovat aina saatavilla, sidottuna pelaajan google-tiliin.



Kuva 16. ISKOS-pelin kauppanäkymä.

Tallennettavan dataan kuuluu kokonaisluku timanteista sekä binäärimuuttujalista jokaisen tuotteen lukituksen tilasta. Kun päävalikko ladataan, ladataan automaattisesti tämä data, joka asetetaan pelin sisäisiin vastaaviin muuttujiin. Aina kun pelissä nämä arvot muuttuvat, eli kun pelaaja saa lisää timantteja tai käyttää niitä tuotteisiin, tallennetaan muuttunut data.

5 Tulokset

Tästä projektista saatiin lopulta tuotettua suunnitellun mukainen Unity-paketti, joka sisältää osan Googlen peli- ja mainospalveluiden toteutuksista sekä rajapinnan niiden käyttöä varten. Paketti tukee rajapinnan kautta pelipalveluista sisään- ja uloskirjautumisen, saavutuksia, tulostaulukoita, tapahtumia sekä pilvitallentamista. Mainospalveluista käytettävissä on banneri-, väli- sekä palkitut mainokset.

Kehitetty rajapinta on helppokäyttöinen, mutta sitä olisi voinut vielä parannella ainakin kommentoinnin ja yhtenäisyyden osalta. Esimerkiksi sosiaalisia palveluita kutsutaan staattisten funktioiden kautta, kun taas mainospalveluita ja pilvitallennusta instanssin kautta. Sosiaalisista palveluista puuttuu myös samanlaiset tapahtumakäsittelijät kuin mitä mainoksista ja pilvitallennuksesta löytyy.

Paketin kehitystä ja testaamista varten kehitettiin myös peli. Pelillisesti se ei ole kovinkaan erikoinen, eikä siltä odotettu kaupallista menestystä. Siitä huolimatta se kuitenkin julkaistiin Google Play -kaupassa, jotta nähtäisiin tämän paketin toimivan sen todellisessa käyttöympäristössä. Tällöin saatiin myös todellista statistiikkaa muun muassa mainospalveluiden toiminnasta (kuva 17).

Sovellus	Mainosyksikkö	Näyttökerrat, tuhannen näyttökerran tulot	Osuvat pyynnöt	Näyttöarvo	Impressiot
ISKOS Ilmainen | An...	Banner	1,09 €	763	70,77 %	540
ISKOS Ilmainen | An...	Loading screen	4,42 €	372	47,31 %	176
ISKOS Ilmainen | An...	Continue playing	5,63 €	497	25,35 %	126
Yhteensä		**2,46 €**	**1 632**	**51,59 %**	**842**

Kuva 17. ISKOS-pelin mainosstatistiikkaa julkaisusta kirjoitushetkeen.

Tästä statistiikasta nähdään esimerkiksi, että vaikka bannerimainoksilla on eniten impressioita, sen tuhannen näyttökerran tulot ovat joukon pienimmät. Palkitulla mainoksella taas on vähiten impressioita, mutta se tuottaa selvästi eniten tuhatta näyttökertaa kohden välimainosten pudotessa näiden väliin.

Tämän lisäksi testattiin vielä, kuinka helposti pelkästään mainokset voidaan ottaa käyttöön tuotannossa olevaan peliin. Pelinä toimi Speed Tester 3D -peli, joka on julkaistu Google Play -kaupassa. Pelissä oli jo käytössä Unity Ads -mainospalvelu, mutta se haluttiin vaihtaa käyttämään Googlen mainospalveluita.

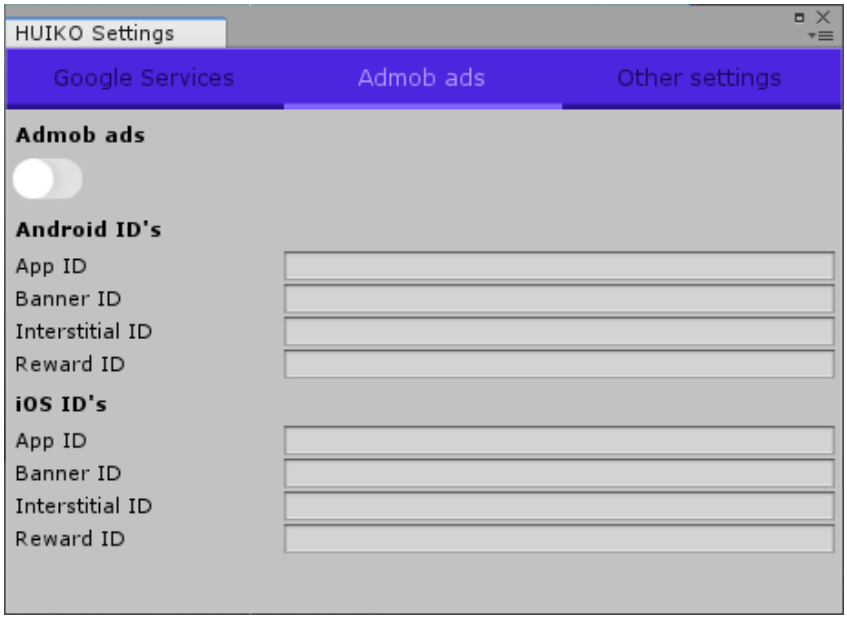
Paketti tuotiin tähän Unity-projektiin, jonka jälkeen konfiguroitiin mainosyksikkö- ja tuotetunnukset. Koodiin lisättiin kaksi riviä koodia, mainoksen lataus- sekä näyttökutsu. Aikaa kului vain minuutteja, jonka jälkeen peli rakennettiin ja päivitettiin pelikauppaan.

6 Yhteenveto ja tulevaisuus

Tämän projektin aloitushetkellä Unityn oma mainostuspalvelu oli hyvinkin vajavainen eikä tukenut kaikkia samoja mainosyksiköitä kuin Googlen palvelu. Tähän on viime aikoina tullut muutoksia, ja nykyään Unity Ads on paljon laajempi kokonaisuus. En kuitenkaan näe, että työni olisi ollut siltä osin turha, sillä myös sen käyttöönotto vaatii melko paljon koodin kirjoittamista, kun taas tällä paketilla voidaan esittää mainoksia jopa vain yhdellä rivillä koodia. Tämän lisäksi Unity Ads on hankalampi alusta aloittavalle kehittäjälle, sillä mainostulojen nostamista varten täytyy kehittäjällä olla yritystunnus.

Yksi tämän projektin merkittävimmistä tavoitteista oli mahdollisimman yksinkertainen käyttöönotto kaikille sen ominaisuuksille. Tätä ajatellen ryhdyttiin toteuttamaan käyttäjäystävällistä Unity-editori-laajennusta, josta voitaisiin asettaa kaikki asetukset sekä kytkeä vain halutut ominaisuudet päälle tai pois, esimerkiksi pilvitallennuksen.

Tämä laajennus saatiin kuvan 18 mukaiseen pisteeseen asti toteutettua, eli käytännössä vain käyttöliittymän osalta valmiiksi. Toiminnallisuuden osalta kävi kuitenkin hyvin nopeasti selväksi, että sen toteuttaminen ei olisikaan aivan yksinkertaista vaan vaatisi huomattavasti enemmän suunnittelua ja kehitystyötä, johon aikaa ei enää riittänyt.



Kuva 18. Huiko-nimiavaruutta varten tehty Unity-editori-ikkuna.

Kuten luvussa 2.1 mainittiin, GPGS sisältää vielä enemmän toiminnallisuuksia kuin tässä paketissa toteutettiin. Näitä voisi tulevaisuudessa toteuttaa ja lisätä pakettiin. Toisaalta niille voi tarvittaessa myös paketin käyttäjä itse tehdä toteutuksen.

Vaikka lopullisen paketin ominaisuuksista moni toimii suoraan iOS-käyttöjärjestelmällä, en voi taata sitä, sillä käytössäni ei ole Applen kehittäjälisenssiä. Tämän vuoksi olen pystynyt kehittämään ja testaamaan toimivuutta vain Android-alustalla. Tämä koskee todennäköisesti vain Googlen pelipalvelujen sekä pilvitallennuksen toiminnallisuuksia, eikä mainoksia.

Kaiken kaikkiaan uskon paketistani olevan hyötyä tulevaisuudessa itselleni, pelikehityksen opiskelijoille sekä muille, jotka haluavat ottaa nämä ominaisuudet nopeasti käyttöönsä ja keskittyä pelinsä kehittämiseen.

Lähteet

1 Share of mobile games on Google Play. 2016. Statista, Inc. Verkkoaineisto. <https://www.statista.com/statistics/673464/free-paid-mobile-games-android-share> Päivitetty 7.12.2016. Luettu 31.7.2019.

2 Public Relations. 2018. Unity technologies. Verkkoaineisto. <www.unity3d.com/public-relations>. Luettu 30.7.2019.

3 Ben Medler. 2009. Journal for Computer Game Culture. Verkkoaineisto. <https://www.eludamos.org/index.php/eludamos/article/view/vol3no2-4/127> Luettu 29.1.2020.

4 Robert Katai. 2018. Evolution of Digital Banner Ads: From Flash to HTML5. Bannersnack. Verkkoaineisto. <https://blog.bannersnack.com/evolution-of-digital-banner-ads/> Luettu. 19.8.2019.

5 Matt Kaplan. 2019. 52 In-App Advertising Statistics You Should Know. Verkkoaineisto. <https://www.inmobi.com/blog/2019/05/21/52-in-app-advertising-statistics-you-should-know> Luettu 19.8.2019.

6 Google. 2019. Disallowed interstitial implementations. Verkkoaineisto. <https://support.google.com/admob/answer/6201362?hl=en> Luettu 28.8.2019

7 Jiaping Gui, Meiyappan Nagappan, William G. J. Halfond. 2017. What Aspects of Mobile Ads Do Users Care About? An Empirical Study of Mobile In-app Ad Reviews. Verkkoaineisto. <https://pdfs.semanticscholar.org/a5a7/a6bbe25c1d3293b0338e42c1afb0edda9e3d.pdf> 22.2.2017. Luettu 28.1.2020.

8 Tapjoy. 2019. Modern Mobile Gamer Personas 2019. Verkkoaineisto. <https://www.tapjoy.com/lp/mobile-gamer-demographics-2019> Luettu 28.1.2020.

9 Applift. 2018. School of Ad Tech: Understanding the Difference Between Rewarded Video and Rewarded Ads. Verkkoaineisto. <https://applift.com/blog/difference-rewarded-video-and-rewarded-ads>

10 Google. 2020. Create a rewarded ad unit. Verkkoaineisto. <https://support.google.com/admob/answer/7311747?hl=en> Luettu 29.1.2020.

11 Google. 2020. Supported locations for distribution to Google Play users. Verkkoaineisto. <https://support.google.com/googleplay/android-developer/table/3541286> Luettu 29.1.2020.