Sami Matero

MIKROMAKSUT UNITY-PELIKEHITYSIYMPÄRISTÖSSÄ

MIKROMAKSUT UNITY-PELIKEHITYSYMPÄRISTÖSSÄ

Sami Matero
Opinnäytetyö
Syksy 2015
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä(t): Sami Matero
Opinnäytetyön nimi: Mikromaksut Unity-pelikehitysympäristössä
Työn ohjaaja(t): Asmo Saloranta
Työn valmistumislukukausi ja -vuosi: Syksy 2015 Sivumäärä: 31 + 3 liitettä

Opinnäytetyön tavoitteena oli tutkia mikromaksujärjestelmien toimintaa mobiilipeleissä sekä toteuttaa vastaavanlainen järjestelmä Frozen Visionin Nepcar Racing -mobiilipeliin, joka julkaistiin Windows Phone -alustalle.

Työssä tutkittiin erilaisia mobiilisovelluskauppoja ja niiden toimintaa. Tämän tiedonpohjalta etsittiin ratkaisu, jolla pystyttiin toteuttamaan mikromaksujärjestelmä Windows Phone-, Android- ja iOS-järjestelmille käyttäen Unity-pelinkehitysohjelmistoa.

Lopputuloksena oli tilaajalle soveltuva käytäntö, jota pystytään käyttämään myös tulevissa mobiilipeleissä, jotka on tehty Unity-pelinkehitysalustalla.

Asiasanat: pelikehitys, mikromaksut, Unity

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author(s): Sami Matero
Title of thesis: Micropayments in Unity mobile game development environment
Supervisor(s): Asmo Saloranta
Term and year when the thesis was submitted: Pages: 31 + 3 appendices

---

Goal of this thesis was to research and implement micropayment system for Frozen Vision's Nepcar Racing game, which has been released on Windows Phone store.

In the work different mobile phone appstore were analyzed and from that information, found solution that could be used in Unity game development software.

Keywords: game development, micropayments, Unity

SISÄLLYS

1 JOHDANTO

Opinnäytetyön tavoitteena oli toteuttaa Frozen Visionin Nepcar Racing -peliin suunniteltu mikromaksusisältö Unity-pelimoottorilla Windows Phone- ja Google Play -sovelluskauppoihin.

Opinnäytetyö toteutettiin samanaikaisesti Nepcar Racing -pelin tuotannon kanssa Windows Phone -alustalle, jossa peli julkaistiin sekä varmistettiin valmiudet Google Play -sovelluskauppaan.

Projektin toteutus esitetään esimerkkiprojektin avulla, jossa on käytetty samoja työkaluja ja menetelmiä kuin Frozen Visionin Nepcar Racing -pelissä. Tämän tarkoituksena on esittää toteutuksen prosessia selkeämmin, kuin tuotantokoodilla olisi voinut.

2 MIKROMAKSUJÄRJESTELMIEN TOIMINTA

2.1 Yleistä mikromaksuista

Mikromaksut on viime vuosina yleistynyt sovellusten ansaintamalli. Mikromaksut tarkoittavat pieniä maksuja, joiden kokoluokka on yleensä kymmenistä senteistä muutamiin euroihin. Mikromaksujen tarkoituksena on antaa myyjille mahdollisuus kaupallistaa tuotteitaan pienemmässä mittakaavassa ja tarjota asiakkaille sisältöä, jonka asiakas mahdollisesti haluaa ostaa ilman, että hän joutuu ostamaan isomman paketin sisältöä.

Yleisiä mikromaksukohteita pelien lisäksi ovat blogit ja nettilehdet, jotka voivat myydä pääsylippuja vain tiettyihin artikkeleihin sen sijaan, että asiakas tilaa koko lehden tai pääsyn kaikkiin blogin artikkeleihin. Mikromaksuista on tullut vuosien varrella pääsääntöinen tapa tarjota digitaalista sisältöä internetissä ja sovelluksissa. Yleensä mikromaksuja löytää ilmaisista sovelluksista, mutta niitä löytää myös maksullisista sovelluksista, jotka haluavat tarjota lisäsisältöä maksua vastaan. Sovellusten sisäisistä ostoista käytetään termejä In App Purchase (IAP) ja In App Billing (IAB).

2.2 Mikromaksut peleissä

Mikromaksut ovat yleistyneet peleihin Free to Play (F2P) -ansaintamallin myötä. F2P-malli tuli suosioon, kun useat peliyhtiöt alkoivat tarjoamaan pelejään ilmaiseksi ja pelin ansaintalogiikkana oli mainosten näyttäminen pelaamisen yhteydessä. F2P-malliin tuli lisäksi sovelluksen sisäiset ostokset, kun peleihin alettiin lisäämään pelin sisäisiä tuotteita, joilla voidaan nopeuttaa pelin kulkua tai lisätä peliin ominaisuuksia, joita ei normaalisti pelaajalla olisi käytössä.

Tyypillisiä mikromaksujen kohteita peleissä ovat seuraavanlaisia:

- Pelinsisäinen valuutta
- Pelivaikutus
    - Pelimekaaniikkaa nopeuttavat boosterit
    - Pelivaluutan keräämistä helpottavat boosterit
    - Pelihahmon kokemuksen keräämistä helpottavat boosterit
- Esineet
    - Pelihahmolla käytettävä esine
    - Keräilykortit tai esineet
    - Muotiesineet
- Lisäosat
    - Kartta-paketti
    - Hahmo-paketti
    - Uudet pelimekaniikat.

2.3 Mikromaksujen käyttötavat peleissä

Mikromaksuja käytetään usealla eri tavalla pelin ansaintamallin mukaisesti. Peleissä käytetään kolmea yleismallia mikromaksujen toteuttamisessa. (1.)

2.3.1 Suorat ostot

Suorissa ostoissa pelaaja menee pelin sisäiseen kauppaan, jossa näkyy suoraan myyntihinta pelaajan paikallisessa valuutassa. Tämän tyyppisiä ostoja käytetään, kun halutaan myydä pelaajalle yksittäisiä kokonaisuuksia, kuten karttapaketit, tai muuta lisäsisältöä. Yleensä peleissä joissa on tämäntyyppisiä ostoksia, ei käytetä useita pelivaluuttoja ja peli voidaan usein ostaa mainosvapaaksi, jos mainoksia näytetään pelissä (1).

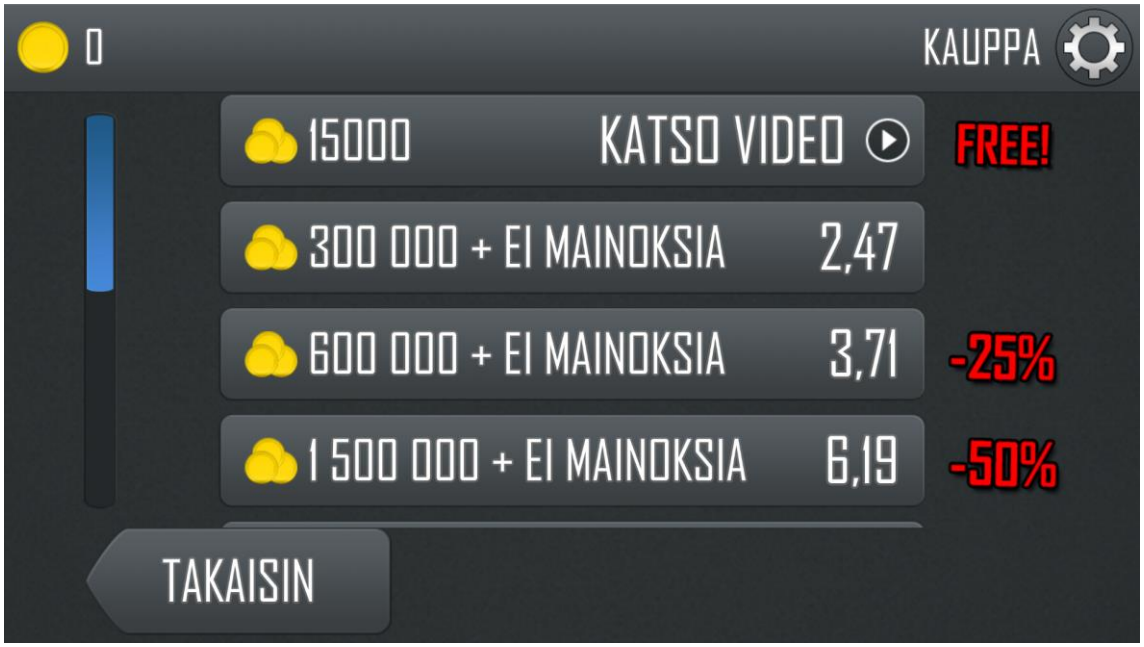
Hyvänä esimerkkinä toimii NDEMIC-studion tuottama Plaque Inc -peli (2), jossa sisältöä voi ostaa auki selkeällä hinnalla (kuva 1).

KUVA 1. Plaque Inc. Pelinsisäinen kauppa (2).

2.3.2 Yksivaluuttajärjestelmä

Yksivaluuttajärjestelmässä pelaaja kerää kolikoita tai vastaavaa pelivaluuttaa, jota pelaaja käyttää ostaakseen itselleen pelissä käytettäviä hyödykkeitä, joilla pelaaja etenee pelissä. Tämän kaltaisiin peleihin kuuluu olennaisesti rahakauppa, josta pelaaja voi ostaa itselleen lisää kolikoita, jotka edistävät pelaajan etenemistä. Yksivaluuttajärjestelmä on hyvin yksinkertaistettu malli, jossa pelaajalla pysyy hyvin hallussa ostoksen arvo ja pelaaja pystyy hyvin hahmottamaan, mitä on ostamassa. (1.)

Esimerkkinä tälläisestä järjestelmästä toimii Fingersoftin Hill Climb Racing (3.), jossa edetään ostamalla uusia kulkuneuvoja käyttämällä pelinsisäistä valuuttaa, jota saadaan lisää pelaamalla tai käyttämällä oikeaa rahaa. (kuva 2.)



KUVA 2. Hill Climb Racing -pelin valuuttakauppa (3).

2.3.3 Monivaluuttajärjestelmä

Monivaluuttajärjestelmässä käytetään yleensä kahta tai useampaa valuuttatyyppiä:

- Pehmeä valuutta on pelissä tienattu rahayksikkö, jota saadaan pelaamalla sekä kulutetaan pelaamisen yhteydessä.
- Kova valuutta, jota yleensä kutsutaan premium-valuutaksi, myydään pelin yhteydessä erikokoisissa paketeissa, joiden hinta vaihtelee puolesta eurosta viiteenkymmeneen euroon.
- Premium-valuutalla yleensä ostetaan pelissä lisäominaisuuksia ja pelin kulkua nopeuttavia esineitä.

Erottamalla pelin sisäiset ostokset oman valuutan alle voidaan pelinsisäisiä tuotteita, kuten karttapaketteja, myydä pelin sisällä ilman tuotteiden rekisteröimistä pelin käyttämään sovelluskauppaan. Tämä mahdollistaa tilannekohtaisten tarjousten tekemisen pelaajalle. (4.)

Yleensä sovelluskauppaan rekisteröidään vain premium-valuuttapaketit, joita ostamalla pelaaja saa peliinsä lisää premium-valuuttaa, jota pelaaja voi vaihtaa pelinsisäisessä kauppapaikassa esineisiin ja ominaisuuksiin, joita pelinkehittäjä on sinne lisännyt.

Esimerkkinä tälläisestä järjestelmästä toimii Supercellin tuottama Boom Beach (5), jossa pelaaja käyttää perusvaluuttaa ja sen lisäksi premium-valuutta, jolla pelaaja voi nopeuttaa pelin eri ominaisuuksia (kuva 3).

KUVA 3. Boom Beachin premium-valuuttakauppa (5).

2.4 Mikromaksut mobiiliympäristöissä

Eri sovelluskauppapalvelut ovat järjestäneet omat mikromaksujärjestelmänsä sovelluskauppansa yhteyteen.

Yleisenä toimintatapana on, että sovelluskaupassa oleva sovellus rekisteröi sovelluksessa käytettävät ostokset ennalta sovelluskaupan tietokantaan, johon sovelluksen sisäisissä ostoissa viitataan ostosta tehdessä.

2.4.1 Yleinen ostotapahtuman kulku

Jokaisella sovelluskaupalla on oma järjestelynsä, mutta yleinen kaava on seuraavanlainen:

1. Ohjelmassa kutsutaan ostotapahtumaa
2. Ostotapahtuma ohjataan maksupalveluun
3. Sovelluksen hallinta siirtyy maksupalvelulle
4. Maksupalvelu käsittelee tapahtuman
5. Maksupalvelu palauttaa ostotapapahtuman tiedot sekä palauttaa ohjelman hallinnan takaisin sovellukselle.
6. Sovellus käsittelee ostotapahtuman palautteen, sekä toimii sen mukaisesti.

2.4.2 Yleisimmät IAP tuotetyypit

Hallitut tuotteet
Hallituissa tuotteissa sovelluskauppa pitää omassa tietokannassaan kirjaa tuotteen ostotiedoista sekä kuittitiedoista.

Kertaostotuote on tuote, jonka voi ostaa vain kerran. Sovelluskauppa pitää sisällään ostokuitin, josta voi tarkistaa onko tuote jo ostettu.

Kulutettava tuote on tuote, jota voi ostaa useamman kerran, mutta se pitää kuluttaa, ennen kuin sen voi ostaa uudestaan. Sovelluskaupan tietokanta pitää sisällään tiedon onko tuote ostettu ja voidaanko se kuluttaa.

Kuluttamisen yhteydessä peli lähettää sovelluskauppaan kulutuspyynnön ja sovelluskauppa palauttaa takaisin viestin, joka kertoo, onko tuotteen kuluttaminen onnistunut. Tämän viestin perusteella peli pystyy reagoimaan viestin mukaisesti.

Tilaustuote on yleensä kuukausimaksulisenssi, jolla voidaan kaupata pelipalveluun pääsyä. Sovellus pystyy tarkistamaan tilauksen tilanteen ja rajoittamaan sovellukseen tai palveluun pääsyä sen mukaisesti

Tilausta pystyy yleensä hallitsemaan sovelluskaupasta sekä pelin sisältä.

Hallitsemattomat tuotteet
Hallitsemattomista tuotteista sovelluskauppa ei pidä mitään kirjanpitoa niiden tilasta. Hallitsemattomia tuotteita käytetään yleensä, kun pelaajalle halutaan kaupata premium-valuutta, jolla pelaaja tekee pelin sisällä ostoksia. Tällöin ostosten hallinta vastuu siirtyy täysin sovelluskaupalta pelinkehittäjälle.

Hallitsemattomat tuotteet on tähdätty tuotteisiin, joita voidaan ostaa useampi peräkkäin kuten valuuttapaketit. Erona hallittujen tuotteiden kulutettavaan tuotteeseen on, että kulutettava tuote pitää erikseen kuluttaa ennen kuin uuden voi ostaa. (6.)

Yleinen esimerkki ostotapahtuman kulusta on seuraavanlainen: Pelaaja ostaa sovelluskaupasta itselleen premium-valuutta, tämän jälkeen pelaaja siirtyy pelinsisäiseen kauppaan, josta pelaaja ostaa itselleen haluamansa peliesineet. Pelin sisäisen oston jälkeen peli rekisteröi itselleen ostotapahtumat ja luovuttaa tuotteet pelaajan käyttöön.

2.5 Yleisimmät sovelluskaupat

Jokaisella mobiilialustalla on yksi tai useampi sovelluskauppa josta käyttäjä saa ladattua sovelluksen ja suorittaa sovelluksen sisäisiä ostoja. Näistä yleisimpiä ovat seuraavat:

2.5.1 Google Play

Google Play on Googlen Android-käyttöjärjestelmälle suunniteltu sovelluskauppa. Sovelluskauppa tarjoaa sovelluksen sisäisiä ostoja Google Wallet -palvelun kautta. (7.)

Maksutavat

Play-kauppa tarjoaa seuraavia maksutapoja (8):

- Luotto- tai maksukortit
- Operaattorin suoralaskutus
- Prepaid-kortit tai lahjakortit
- Paypal.

Mikromaksujen tuotetyypit

Play-kauppa tarjoaa seuraavia tuotetyyppejä (9):

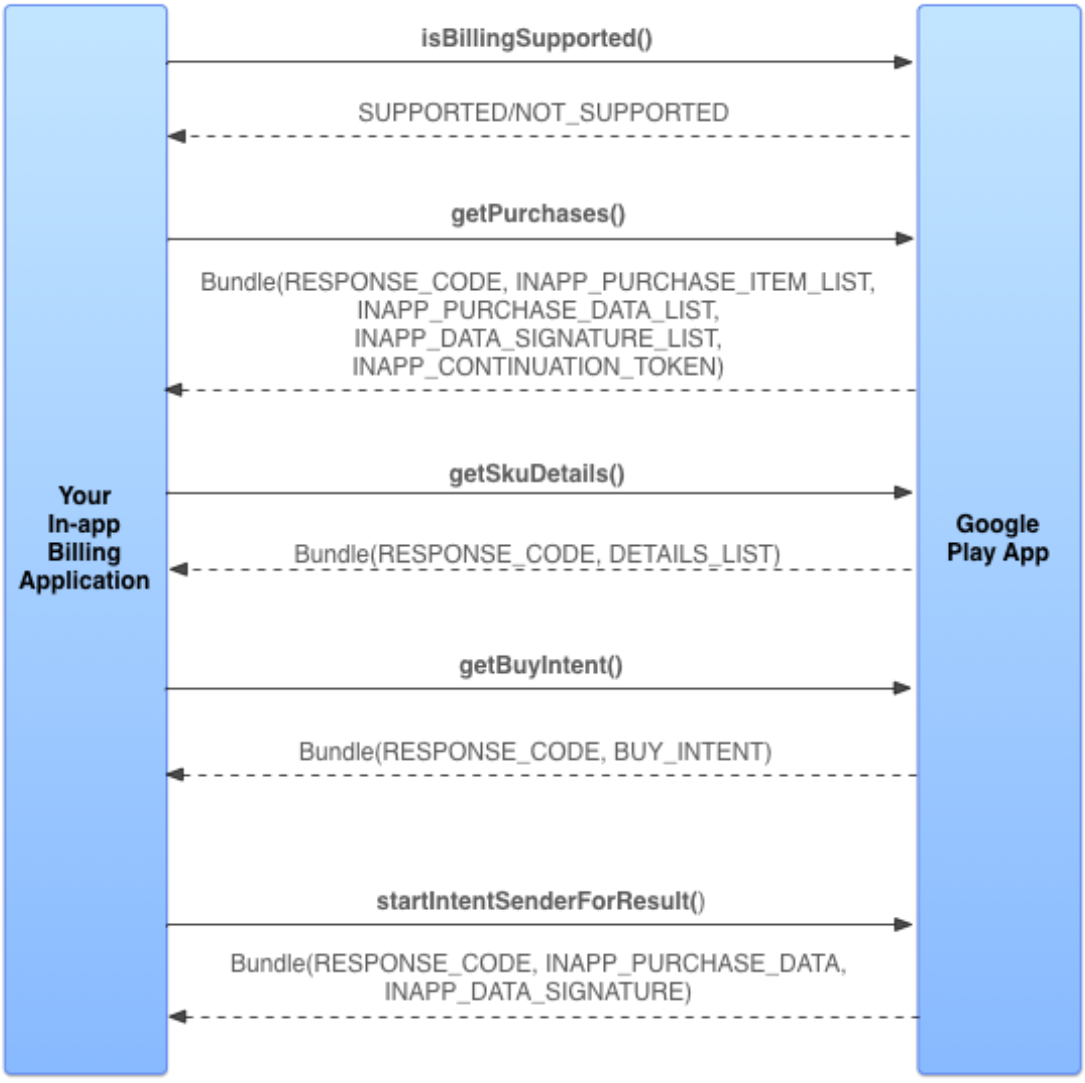
- Hallitut tuotteet
    - Kulutettava tuote
    - Kertaostotuote
- Tilaukset.

Sovelluskaupan integrointi

Play-kauppa tarjoaa oman rajapinnan IAP-ostoksien tekemiseen Android-alustalla.

In-app Billing API pitää sisällään sovelluksen sisäisen välimuistin ostoksista, joka nopeuttaa ostosten tilan tarkastelua ja sallii sovelluksen käyttämisen verkosta irti (9).

Play-kaupan ostotapahtumat kulkee kuvan 4 osoittaman kaavan mukaan.



KUVA 4. Play-kaupan ostotapahtuman kulku. (9)

2.5.2 Apple App Store

App Store on Applen kehittämä sovelluskauppa-alusta Applen puhelimille sekä tietokoneille (10).

Maksutavat

App Store tarjoaa seuraavat maksutavat (11):

- Luotto- tai maksukortit
- Kaupan krediitit / Valmiiksi ladatut käyttörahat
- Lahjakortit / lahjakoodit
- ClickandBuy, PayEase, Paypal.

Mikromaksujen tuotetyypit

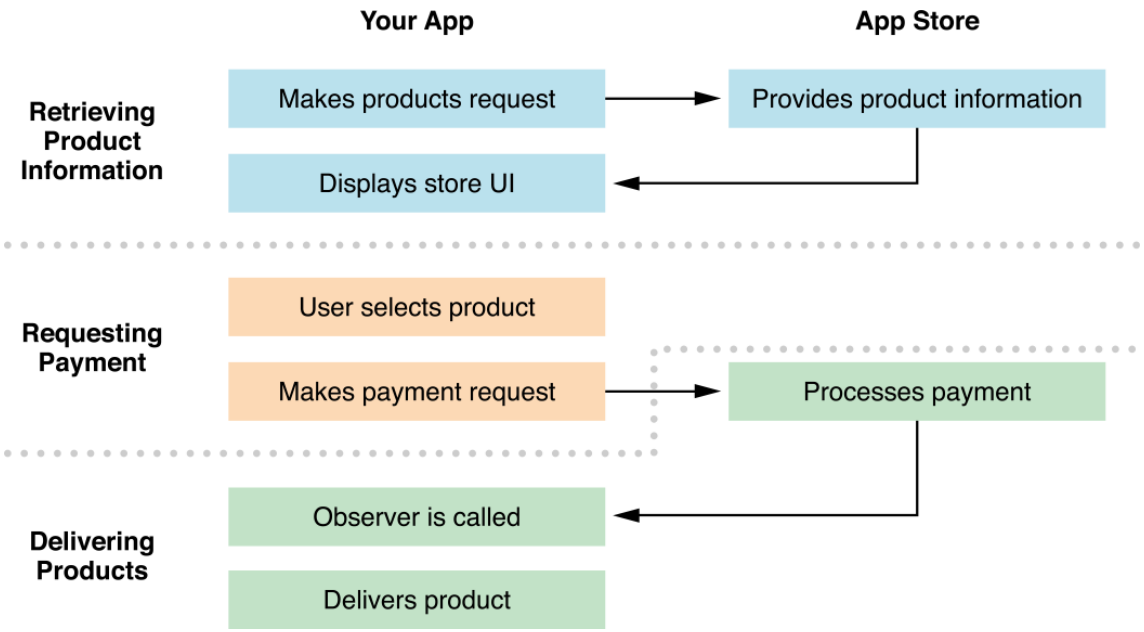
App Store tarjoaa seuraavia tuotetyyppejä (12):

- Kulutettavat tuotteet
- Kertaostotuotteet
- Automaattisesti uusiutuvat tilaukset
- Ei-uusiutuvat tilaukset
- Ilmaiset tilaukset.

Sovelluskaupan integrointi

Apple tarjoaa StoreKit-rajapinnan IAP-ostoksia varten (13)

Ostotapahtuman kulku App Storessa on kuvan 5 mukainen.



KUVA 5. App storen ostotapahtuman kulku. (13)

2.5.3 Windows Store

Windows Store on Microsoftin kehittämä sovelluskauppa Windows Phone -laitteille (14).

Maksutavat

Windows Store tarjoaa seuraavia maksutapoja (15):

- Luotto- tai maksukortit
- Prepaid
- Lahjakortit
- Operaattorin suoralaskutus
- Paypal, Alipay, INICIS.

Mikromaksujen tuotetyyppejä

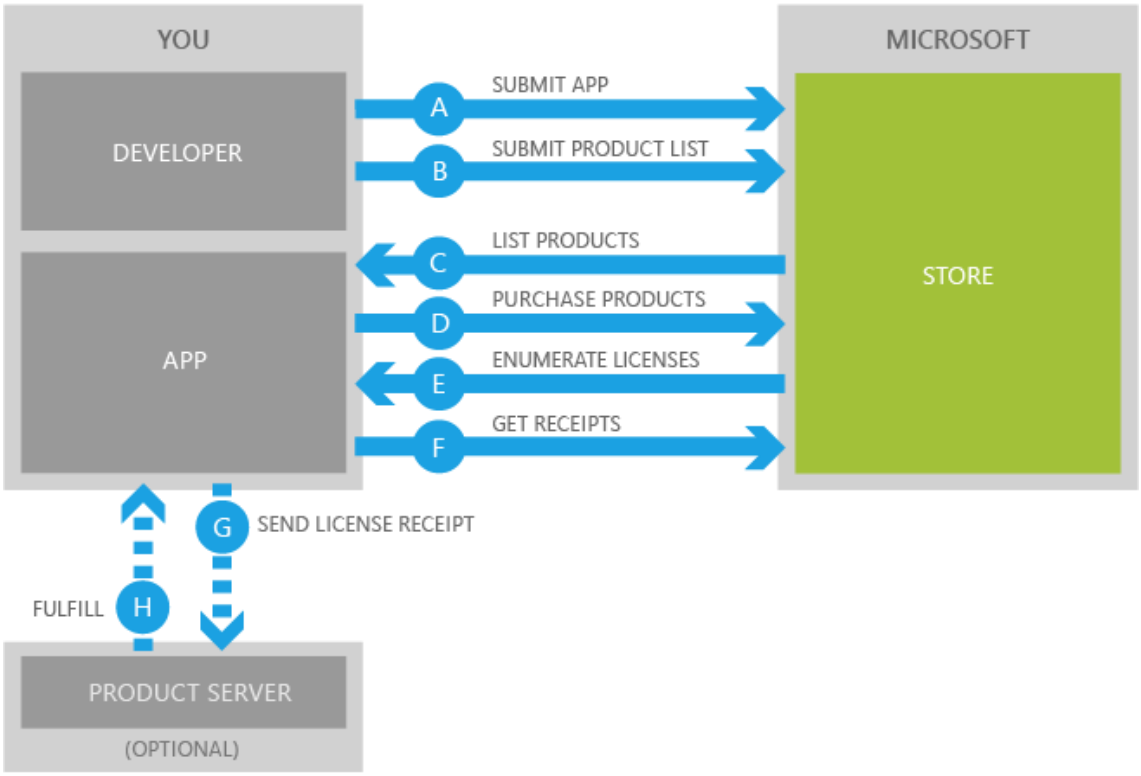
Windows Storen tukemat tuotetyypit ovat (16):

- Kulutettava tuote
- Kertaostotuote.

Sovelluskaupan integrointi

Microsoft tarjoaa oman rajapinnan sovelluskaupalleen (15).

Ostoksen kulku Windows Storessa menee kuvan 6 mukaisesti.



Kuva 6. Windows Store ostotapahtuman kulku (14).

3 UNITY-SOVITUS

Nepcar Racing kanssa päätettiin käyttää valmista liitännäistä mikromaksujen toteuttamiseen. Tarkoituksena oli nopeuttaa toteutusta sekä välttää pahimpia aloittelijavirheitä.

Unity-liitännäisen valinnassa oli 4.2.2014 valintakriteereinä hinta, alustojen kattavuus sekä alustojen välinen yhtenäisyys. Vertailun tulokset näkyvät taulukossa 1. Alustoiksi valittiin Google Play, Windows Store sekä Apple App store niiden laajan käyttäjäkunnan sekä helppokäyttöisen sovelluskaupan takia.

Koodipohjan yhtenäisyys valittiin toiseksi kriteeriksi. Koodipohjan yhtenäisyydellä tarkoitetaan, että samalla liitännäisellä voidaan käyttää useampaa sovelluskauppa eikä jokaiselle sovelluskaupalle tarvitse kirjoittaa omaa hallintaa.

TAULUKKO 1. Unity-mikromaksuliitännäisten vertailu

Nimi	In App Purchahing combo	Unibill	Soomla	Roar Engine
Tekijä	Prime31	Outline Games	Opensource	Roar Engine
Hinta euroina	130	166	Ilmainen	495
Google Play	Kyllä	Kyllä	Kyllä	Kyllä
Apple App store	Kyllä	Kyllä	Kyllä	Kyllä
Windows Phone Store	Ei	Kyllä	Ei	Ei
Yhtenäinen koodipohja	Ei	Kyllä	Kyllä	Kyllä
Linkki	https://prime31.com/plugins	http://outlinegames.com/unibill.htm	http://project.soom.la/	http://roarengine.com

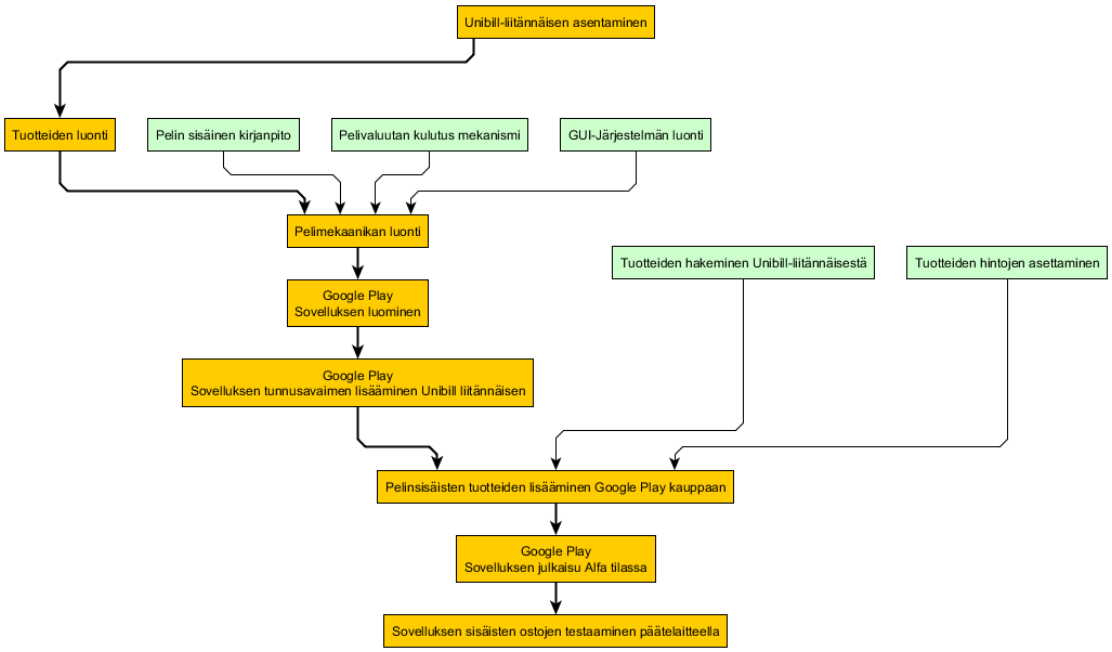
Valitsin Nepcar Racing projektiin Unibill-liitännäisen, koska se tarjosi helposti käytettävän rajapinnan kolmelle sovelluskaupalle, joita projektissamme käytettiin, sekä jättää varaa muille sovelluskaupoille, joita vertailussa ei käytetty.

4 PROJEKTIN TOTEUTUS GOOGLE PLAY MARKETISSA

Projektin toteutus esitetään esimerkkiprojektin avulla, jossa on käytetty samoja työkaluja ja menetelmiä kuin Frozen Visionin Nepcar Racing -pelissä. Tämän tarkoituksena esittää toteutuksen prosessia selkeämmin, kuin tuotantokoodilla olisi voinut.

Nepcar Racingssä käytettiin yksivaluuttajärjestelmää, jossa pelaajalla oli käytössä Nep coin niminen pelin sisäinen valuutta, jolla pelaaja pystyi päivittämään autoaan paremmaksi. Tätä valuuttaa pelaaja tienasi pelaamalla tai ostamalla sitä lisää sovelluskaupasta valuutta paketteina.

Esimerkkiprojektissa tehdään yksinkertainen peli, jossa käytetään yksivaluuttajärjestelmää. Tällä kuvastetaan prosessia jossa pelaaja ostaa pelin valuuttaa, joka muutetaan peliesineiksi, jotka kulutetaan pois. Esimerkissä pelaajalla on käytössään luoteja, joita voi ostaa lisää käyttämällä timantteja. Timantteja pelaaja voi ostaa lisää Google Play kaupasta. Esimerkkiprojektin kulku menee kuvan 7 osoittamalla tavalla



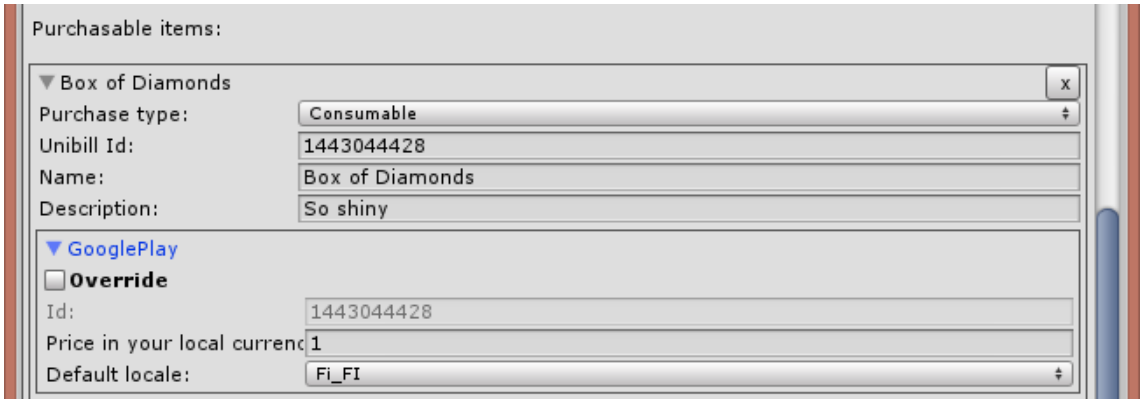
Kuva 7. Esimerkkiprojektin työvaiheet

4.1 Projektin sekä sovellustuotteiden luonti

Luodaan tyhjä Unity-projekti. Tämän jälkeen avataan Asset Store, josta haetaan Unibill-liitännäinen.

Esimerkkiprojektissa käytetään yksi valuuttajärjestelmää, joten luodaan Unibill Inventory Editorissa valuutaksi timantit ja ostettavaksi tuotteeksi Box of diamonds. Sen ostamalla pelaaja saa kymmenen timanttia jotka voidaan myöhemmin vaihtaa pelissä käytettäviin luoteihin.

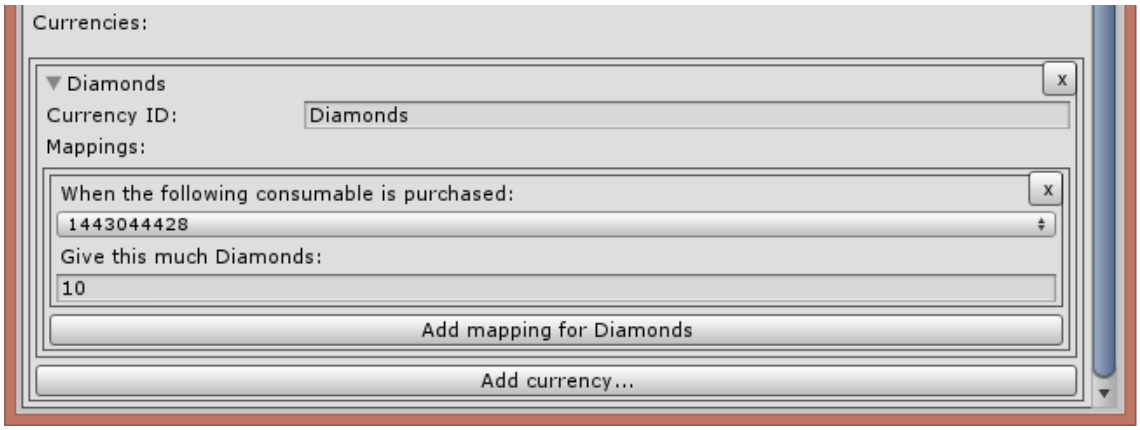
Avataan Inventory Editor ja siirrytään Purchasable items-kohtaan, jossa lisätään uusi ostettava tuote painamalla Add item... -painiketta. Tämän jälkeen valitaan tuotetyypiksi Consumable eli kulutettava tuote, lisätään vielä nimi ja kuvaus. Tuotteen hinnan määrittely tapahtuu kauppa kohtaisella välilehdellä. Esimerkissä avataan GooglePlay-välilehti ja asetetaan hinnaksi 1, joka tarkoittaa yhtä euroa, Default locale -kohtaan laitetaan valinta Fi_FI kuvan 8 mukaisesti.



Kuva 8. Valmis tuote Inventory Editorissa

Valuutta luodaan samassa ikkunassa alempana Currencies osuudessa painamalla Add currency... -painiketta painamalla. Tämän jälkeen määritellään valuutaksi Diamonds kirjoittamalla se Currency ID kohtaan.

Unibill tarjoaa automaattisen valuutan ostonhallinnan, jota käytämme esimerkissä. Tämä ominaisuus lisätään painamalla Add mapping for Diamonds -painiketta, jolloin voimme valita tuotteen, jonka ostamalla saadaan tätä valuuttaa lisää. Valitaan juuri luotu tuote sekä asetetaan että kyseisen tuotteen ostamalla saa kymmenen kappaletta timantteja. Kuvassa 9 näkyy valuutan asetukset.



KUVA 9. Valmis valuutta Inventory Editorissa

4.2 Unibill-toiminnallisuus

Unibill toiminnallisuus lisätään tekemällä komponentti, joka käynnistyy pelin yhteydessä alustaen Unibiller-luokan, joka ottaa yhteyden pelissä käytettävään sovelluskauppaan. Unibiller-luokka tarjoaa kehittäjälle usean event-kutsun josta voidaan seurata maksutapahtumaa.

Unibiller vastaanottaa ostotapahtumasta tulevan viestin, jonka perusteella tuote realisoidaan pelaajalle käytettäväksi tavaraksi. Tuotteen ostonhallinnan voi jättää Unibillin hallittavaksi tai tapahtuman voi itse hallita unibiller-luokasta tulevan viestin perusteella.

Koska sovelluksemme tarvitsee ainoastaan yhden sovellustuotteen valuutalle, voidaan jättää ostotapahtuman hallinnoiminen Unibillille joka muuttaa Box of Diamonds tuotteen kymmeneksi timantiksi.

Unibill mahdollistaa myös sovelluskaupasta riippuen kuittien haku ominaisuuden, jolloin voidaan hakea pelaajan aikaisemmat ostokset ja palauttaa nämä, jos pelaaja on esimerkiksi asentanut sovelluksen uudestaan.

Esimerkissä luodaan Shop-niminen luokka, joka alustaa Unibiller-luokan, sekä hallinnoi pelituotteiden ostotapahtumia. Tämä komponentti liitetään tyhjään peliobjektiin, jolloin se alustuu pelin käynnistyessä. (Liite 1.)

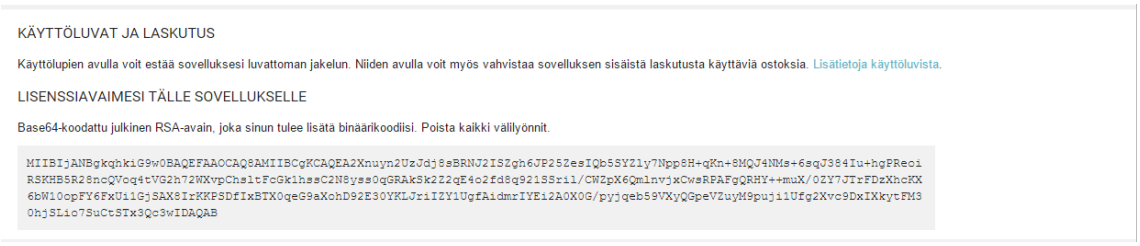
Tämän lisäksi luodaan PlayerInventory-luokka, joka hallinnoi pelaajan pelissä olevia esineitä eli tässä tapauksessa luotien saamista ja kuluttamista. (Liite 2.)

Viimeisenä lisätään graafinen käyttöliittymä peliin, johon liitetään Shop-luokan timanttien ja luotien ostofunktiot sekä PlayerInventory-luokan luotien kuluttamis funktio. Tätä varten esimerkkiprojektiin luotiin GUIHandler-luokka joka hallitsee käyttöliittymää pelitietojen perusteella, sekä päivittää käyttöliittymässä näkyviä tietoja. (Liite 3.)

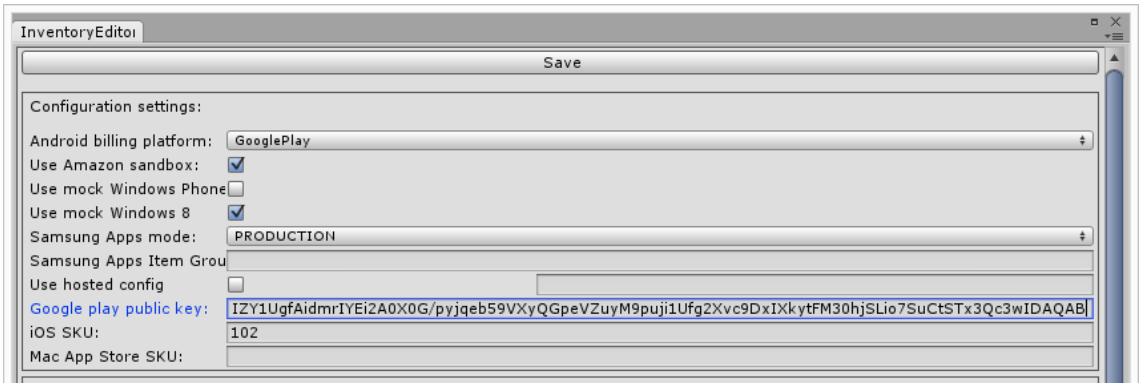
4.3 Projektin liittäminen Google Play -kauppaan

Projektissa käytetään olemassa olevaa Googlen kehittäjätiliä, johon luodaan uusi sovellus. Tähän sovellukseen lisätään julkaisua varten tarvittavat tietueet, joiden lisäämisessä Google Play -kauppa ohjeistaa.

Ostettavien tuotteiden luonnissa käytetään Unibill Inventory Editorin Google Play export -ominaisuutta. Tätä ominaisuutta varten on haettava Google Play -lisenssiavain (kuva 10) ja liitettävä se Inventory Editoriin (kuva 11).



KUVA 10. Google Play -sovellusavain



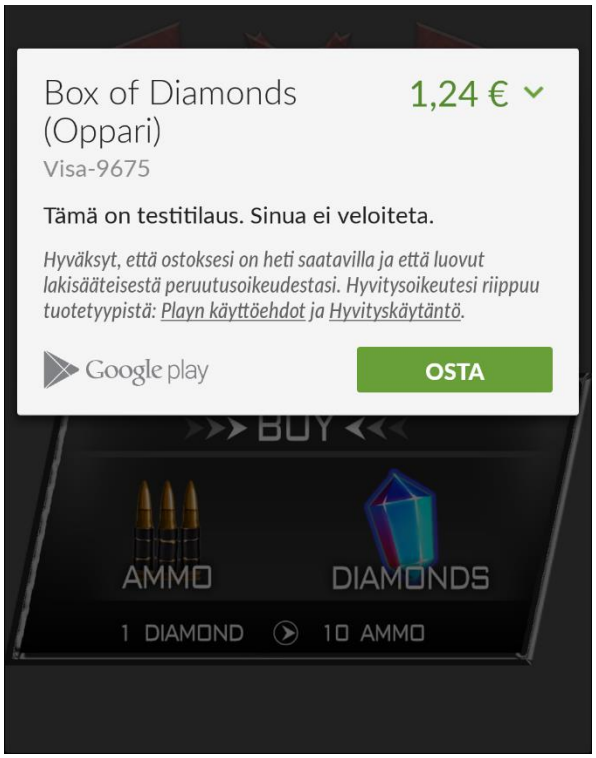
KUVA 11. Sovellusavaimen paikka Inventory Editorissa

Tämän jälkeen voidaan hakea Unibillin generoima MassImportCSV.txt-tiedosto hakemistosta “<projektin kansio>\Assets\Plugins\unibill\generated\googleplay\” ja Google Play sovelluksen Sovellustuotteet välilehden Tuo sovelluksen sisäisiä tuotteita CSV-tiedoston avulla toiminnallisuutta hyväksi käyttäen luodaan sovellustuotteet Google Play kauppaan.

4.4 Maksutapahtuman testaaminen laitteessa

Google Play sallii Alfa-tilassa julkaistujen sovellusten testata sovelluksen sisäisiä ostoja laitteessa. Alfa-tilassa julkaistujen sovellusten ostot kulkee täysin samaa kaavaa kuin täysin julkaistujen sovellusten ostotapahtumat, paitsi alfa-tilassa Google ei laskuta käyttäjää. Tämä mahdollistaa sovelluksen eri skenaarioiden testaamista jolloin voidaan löytää mahdollisia ongelmakohtia sovelluksesta.

Jotta päästään testaamaan maksutapahtumia, tulee sovellus julkaista Alfa-tilassa. Tämä onnistuu kääntämällä projekti Android-asetuksella, jolloin saadaan apk-asennustiedosto. Tämä tiedosto lisätään Google Play sivuilla APK sivulla alfa-testauksessa -välilehden alla Lähetä uusi APK alfa-vaiheeseen -napin kautta jolloin sovellus julkaistaan Alfa-vaiheessa. Kun Google Play on julkaissut sovelluksen, voidaan sovellus asentaa puhelimeen Google Play sivustolta ja testata peliä (kuva 12).



KUVA 12. Ostamisen testaaminen laitteessa

4.5 Huomioitavia asioita sekä ongelmakohtia

Kehittäjän on tärkeä huomioida pelaajan oston takaaminen. Ostotapahtuman aikana voi tapahtua useita tekijöitä, kuten internet yhteyden katkeaminen, jonka seurauksena pelaajalta voi lähteä rahat, mutta pelille ei tule viestiä ostotapahtuman suoriutumisesta.

Tätä tilannetta voidaan torjua hakemalla sovelluskaupasta viimeisimmät kuitit, jos on syytä epäillä, että ostotapahtuma ei ole kulkenut aivan loppuun. Tällöin sovelluksessa pidetään kirjaa tunnetuista kuiteista ja verrataan niitä sovelluskaupasta haettuihin kuitteihin, jolloin voidaan palauttaa kadonnut ostos.

Toinen suuri ongelma on piratismi, jonka yksi yleisimpiä muotoja mobiilialustalla on sovelluskauppa emulaattorit joihin ostotapahtuman ohjataan proxy menetelmällä ja emulaattori palauttaa viestin onnistuneesta ostotapahtumasta. Tällaista hyökkäystä voi yrittää torjua ostokuittien tarkastelulla, mutta selvää ratkaisua ei vielä ole.

5 YHTEENVETO

Opinnäytetyön tavoitteena oli etsiä ratkaisu Frozen Visionin Nepcar Racing pelin mikromaksu tarpeisiin. Työn aikana tutkittiin hyvin eri puhelinalustojen kauppapaikat, niiden ominaisuudet sekä heikkoudet.

Tämän tiedon pohjalta pystyin etsimään ratkaisun jolla pystyin tekemään peliin mikromaksujärjestelmän joka toimi usealla eri alustalla sekä niiden yleisimmillä kauppapaikoilla. Lopputuloksena löytyi käytäntö jota pystyttiin käyttämään tulevien Unity-pohjaisten mobiilipelien mikromaksusisällön tuottamisessa.

Projekti oli mielenkiintoinen katsaus mobiilipelien suosituimpaan ansaintamalliin sekä erilaisten sovelluskauppojen toimintatapoihin. Alan monimuotoisuus teki tästä haastavan urakan löytää juuri meidän pelille sopiva ratkaisu, sillä sovelluskauppoja on useita alustasta riippuen. Tulevaisuudessa lisää haasteita riittää Aasian sovelluskaupoista, joita ei projektin puitteissa tarvinnut miettiä, sillä pelin julkaisu keskittyi lähinnä länsimaalaisille markkinoille.

LÄHTEET

1. Looking at In-Game Currencies. 2014. Game Sparks. Saatavissa: http://www.gamesparks.com/blog/looking-at-in-game-currencies/. Hakupäivä 9.3.2015.
2. Plague Inc. 2012. Google Play. Saatavissa: https://play.google.com/store/apps/details?id=com.miniclip.plagueinc&hl=fi. Hakupäivä 9.3.2015.
3. Hill Climb Racing. 2012. Google Play. Saatavissa: https://play.google.com/store/apps/details?id=com.fingersoft.hillclimb&hl=fi. Hakupäivä 9.3.2015
4. Murphy, David 2015. Gondola Brings Dynamic Pricing to Mobile Games. PC Mag. Saatavissa: http://www.pcmag.com/article2/0,2817,2480206,00.asp. Hakupäivä 14.5.2015.
5. Boom Beach. 2014. Google Play. Saatavissa: https://play.google.com/store/apps/details?id=com.supercell.boombeach. Hakupäivä 9.3.2015.
6. Blundell 2012. Difference between managed and unmanaged in-app product android? Stackoverflow. Saatavissa: http://stackoverflow.com/questions/9391123/difference-between-managed-and-unmanaged-in-app-product-android. Hakupäivä: 14.5.2015.
7. Sovellusten ja digitaalisen sisällön asentaminen tai ostaminen. 2015. Google Play. Saatavissa: https://support.google.com/googleplay/answer/113409?hl=fi. Hakupäivä 20.5.2015.
8. Hyväksytyt maksutavat. 2015. Google Play Ohjeet. Saatavissa: https://support.google.com/googleplay/answer/2651410?hl=fi. Hakupäivä 20.5.2015.
9. In-app Billing API. 2012. Android Developers. Saatavissa: http://developer.android.com/google/play/billing/api.html. Hakupäivä 20.5.2015.

10. In-App Purchace for Developers. 2013. Apple. Saatavissa: https://developer.apple.com/in-app-purchase/. Hakupäivä 20.5.2015.
11. iTunes Storessa, Mac App Storessa, App Storessa ja iBooks Storessa hyväksytyt maksutavat. 2013. Apple. Saatavissa: https://support.apple.com/fi-fi/HT202631. Hakupäivä 20.5.2015.
12. Getting Started with In-App Purchase. 2013. Apple. Saatavissa: https://developer.apple.com/in-app-purchase/In-App-Purchase-Guidelines.pdf. Hakupäivä 20.5.2015.
13. About In-App Purchase. 2013. Apple. Saatavissa: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Introduction.html. Hakupäivä 20.5.2015.
14. In-app purchase for Windows Phone 8. 2014. MSDN. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/jj206949(v=vs.105).aspx. Hakupäivä 20.5.2015.
15. Windows Phone -kaupan maksutavat. 2014. Microsoft. Saatavissa: http://www.windowsphone.com/fi-fi/how-to/wp8/accounts-and-billing/payment-methods-for-the-windows-phone-store. Hakupäivä 20.5.2015.
16. In-App Purchase API overview for Windows Phone 8. 2015. MSDN. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/jj206950(v=vs.105).aspx. Hakupäivä 20.5.2015.

LIITTEET

Liite 1 Shop.cs

Liite 2 PlayerInventory.cs

Liite 3 GUIHandler.cs

```
using System;
using System.Linq;
using UnityEngine;

public class Shop : MonoBehaviour
{
    public PlayerInventory Inventory;
    private bool IsInitialized;

    #region Unity events

    private void Start()
    {
        InitiateUnibill();
    }

    private void OnDestroy()
    {
        RemoveUnibill();
    }

    #endregion

    #region Unibill

    /// <summary>
    ///     Subscripes into unibill events and initializes unibill
    /// </summary>
    private void InitiateUnibill()
    {
        //Subscripe into unibill events
        Unibiller.onBillerReady += onBillerReady;
        Unibiller.onPurchaseCancelled += onCancelled;
        Unibiller.onPurchaseFailed += onFailed;
        Unibiller.onPurchaseCompleteEvent += onPurchased;
        Unibiller.onPurchaseDeferred += onDeferred;

        //Initialize unibill after subsciping into events to ensure all events working
        correctly
        Unibiller.Initialise();
    }

    /// <summary>
    ///     Removes subscriped events when destroying object
    /// </summary>
    private void RemoveUnibill()
    {
        Unibiller.onBillerReady -= onBillerReady;
        Unibiller.onPurchaseCancelled -= onCancelled;
        Unibiller.onPurchaseFailed -= onFailed;
        Unibiller.onPurchaseCompleteEvent -= onPurchased;
        Unibiller.onPurchaseDeferred -= onDeferred;
    }

    #endregion

    #region Unibill events

    /// <summary>
    ///     Purchase is deferred to get approval from parental system
```

```
    /// </summary>
    /// <param name="obj">Item that user was purchasing</param>
    private void onDeferred(PurchasableItem obj) { Debug.Log("Purchase is Defered"); }

    /// <summary>
    ///     This event is called when user purchased item succesfully
    /// </summary>
    /// <param name="obj">Item that user was purchasing</param>
    private void onPurchased(PurchaseEvent obj) { Debug.Log("Purchase succesfull"); }

    /// <summary>
    ///     This event is called when user failed to buy item.
    /// </summary>
    /// <param name="obj">Item that user was purchasing</param>
    private void onFailed(PurchasableItem obj) { Debug.Log("Purchase failed"); }

    /// <summary>
    ///     This event is called when player cancells purchase
    /// </summary>
    /// <param name="obj">Item that user was purchasing</param>
    private void onCancelled(PurchasableItem obj) { Debug.Log("User Cancelled"); }

    /// <summary>
    ///     This event is called when unibill is initialized fully
    /// </summary>
    /// <param name="obj">Object with state of unibill</param>
    private void onBillerReady(UnibillState obj)
    {
        switch (obj)
        {
            case UnibillState.SUCCESS:
                IsInitialized = true;
                break;
            case UnibillState.SUCCESS_WITH_ERRORS:
                IsInitialized = true;
                break;
            case UnibillState.CRITICAL_ERROR:
                IsInitialized = false;
                break;
            default:
                throw new ArgumentOutOfRangeException("obj");
        }
    }

    #endregion
```

```
    #region ShopFunctions

    /// <summary>
    ///     Uses unibill to use diamonds for buying bullets
    /// </summary>
    public void BuyBullets()
    {
        //Exits if Inventory is null so we won't waste diamonds
        if (Inventory == null) return;

        //If succesfully consumed diamonds give bullets to player
        if (Unibiller.DebitBalance("Diamonds", 1))
        {
            Inventory.GiveBullets(10);
        }
        else
        {
            Debug.LogError("Not enough Diamonds for bullets");
        }
    }

    /// <summary>
    ///     Initiates purchase of diamonds
    /// </summary>
    public void BuyDiamonds()
    {
        //Checks if Unibill is ready
        if (IsInitialized)
        {
            //Gets item code that we want
            PurchasableItem item =
                Unibiller.AllPurchasableItems.FirstOrDefault(
                    allPurchasableItem => allPurchasableItem.name.Equals("Box of
Diamonds"));
            //Uses item info to initiate purchase
            Unibiller.initiatePurchase(item);
        }
        else
        {
            Debug.LogError("Unibill is not initialized");
        }
    }

    #endregion
}
```

```
using UnityEngine;

public class PlayerInventory : MonoBehaviour
{
    public int Bullets;

    #region Unity events

    private void Start()
    {
        //Loads players inventory when script is started
        LoadInventory();
    }

    private void OnDestroy()
    {
        //Saves inventory when script is destroyed
        SaveInventory();
    }

    #endregion

    #region saving and loading

    public void SaveInventory()
    {
        //Saves amount of bullets into playerprefs for later use
        PlayerPrefs.SetInt("Ammunition", Bullets);
    }

    public void LoadInventory()
    {
        //Gets ammount of bullets from playerprefs with default value of 0
        Bullets = PlayerPrefs.GetInt("Ammunition", 0);
    }

    #endregion

    /// <summary>
    /// Used from Shop after purchasing bullets via Diamonds
    /// </summary>
    /// <param name="amount"></param>
    public void GiveBullets(int amount)
    {
        Bullets = Bullets + amount;
    }

    /// <summary>
    /// Used from GUI to consume bullets
    /// </summary>
    public void UseBullets()
    {
        if (Bullets > 0)
        {
            Bullets--;
        }
        else
        {
            Debug.Log("No enough bullets");
        }
    }
}
```

```
using UnityEngine;
using UnityEngine.UI;

public class GUIHandler : MonoBehaviour
{
    //These variables are filled from unity editor
    public Text AmmoText;
    public Text DiamondText;

    public bool OutOfAmmo;
    public Image OutOfAmmoImage;

    public PlayerInventory PInventoryInstance;
    public Button ShootButton;
    public Shop ShopInstance;

    private void Update()
    {
        //Updates our information about diamonds and bullets
        OutOfAmmo = PInventoryInstance.Bullets <= 0;
        UpdateAmmoCounter();
        UpdateDiamondCounter();

        //Checks ammount of bullets and locks shoot button if neccessary
        if (OutOfAmmo)
            LockShootButton();
        else
            UnlockShootButton();
    }

    /// <summary>
    ///     Gets bullet count from PlayerInventory and updates text field
    /// </summary>
    private void UpdateAmmoCounter()
    {
        AmmoText.text = PInventoryInstance.Bullets.ToString();
    }

    /// <summary>
    ///     Gets Diamond count from Unibiller and updates text field
    /// </summary>
    private void UpdateDiamondCounter()
    {
        //Unibill keeps count of its currencies in its own database
        var diamonds = (int) Unibiller.GetCurrencyBalance("Diamonds");
        DiamondText.text = diamonds.ToString();
    }

    private void UnlockShootButton()
    {
        OutOfAmmoImage.enabled = false;
        ShootButton.interactable = true;
    }

    private void LockShootButton()
    {
        OutOfAmmoImage.enabled = true;
        ShootButton.interactable = false;
    }
}
```