Tommi Kerola

ANDROID MOBIILIPELIN KEHITYS

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Marraskuu 2016

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Marraskuu 2016	**Tekijä/tekijät** Tommi Kerola
Koulutusohjelma Tietotekniikka		
Työn nimi ANDROID MOBIILIPELIN KEHITYS		
Työn ohjaaja Kauko Kolehmainen		**Sivumäärä** 24 + 3
Työelämäohjaaja		

Opinnäytetyössä kehitettiin Crash Copter -niminen mobiilipeli Android-järjestelmälle, johon lisättiin AdMob-mainoksia. Lopulta peli julkaistiin Google Play -palvelussa, josta kuka tahansa voi sen ladata ilmaiseksi omaan Android-mobiililaitteeseensa.

Teoriaosuudessa tutkittiin Libgdx-nimistä sovelluskehystä, joka toimi runkona omalle projektille. Teoriaosuudessa perehdyttiin sovelluskehyksen toimintaan, ominaisuuksiin ja sovelluskehyksen käyttöönottoon.

Käytännön osuus koostui mobiilipelin koodaamisesta sekä pelissä käytettyjen grafiikoiden, efektien, sekä äänitehosteiden luomisesta.

Asiasanat
Android, Google Play, Java, Libgdx

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Centria University of Applied Sciences	**Date** November 2016	**Author** Tommi Kerola
Degree programme Information Technology		
Name of thesis ANDROID MOBILE GAME DEVELOPMENT		
Instructor Kauko Kolehmainen		**Pages** 24 + 3
Supervisor		

The subject of this thesis was to develop Crash Copter named mobile game for the Android platform. The game also includes AdMob mobile ads and it was uploaded to Google Play service where anyone can download it for free.

Theory part of this thesis was to investigate Libgdx framework. The Libgdx framework works as a backbone for the own project. Theory part investigates how Libgdx works, what it has to offer and how it can be used to develop an own mobile game.

The practical part of this thesis was to write the mobile game and create all the assets for the game, including graphics, sounds and effects.

Key words
Android, Google Play, Java, Libgdx

KÄSITTEIDEN MÄÄRITTELY

API	Application Programming Interface Sovellusrajapinta, jonka avulla eri ohjelmat voivat vaihtaa tietoa keskenään
Framework	Sovelluskehys, joka toimii runkona sen päälle rakennettavalle ohjelmalle
IDE	Integrated Development Environment Ohjelmointiympäristö lähdekoodin tuottamiseen
UML	Unified Modeling Language Graafinen mallinnuskieli oliosuhteiden kuvaamiseen
HUD	Head Up Display Pelitilan päälle piirretty erillinen käyttöliittymä
SDK	Software Development Kit Valikoima sovelluskehitystyökaluja
JDK	Java Development Kit Valikoima ohjelmia Java sovelluskehitykseen
Open GL ES	Open Graphics Library for Embedded Systems OpenGL grafiikkakiihdytyksen sovellusrajapinta sulautetuille järjestelmille
AdMob	Advertising on Mobile Googlen omistama mobiilimainospalvelu

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

1 JOHDANTO 1

2 LIBGDX-FRAMEWORK 2
2.1 Historia 2
2.2 Yleistä 2
2.3 Moduulit 3
2.4 Sovelluksen elinkaari 5
2.5 Projektin luominen 6
2.5.1 Ohjelmointiympäristö 7
2.5.2 Gradle 8
2.5.3 Projektin rakenne 8

3 PELIN TOTEUTUS 9
3.1 Pelin idea 9
3.2 UML-luokkakaaviot 9
3.2.1 Pelin pääluokat 9
3.2.2 Pelin staattiset apuluokat 10
3.3 Pelin valikko 11
3.4 Pelinäyttö 12
3.5 Resoluutio ja kamera 13
3.6 Kenttä 15
3.7 Box2d-fysiikkamoottori 16
3.8 Partikkeliefektit 19

4 ADMOB-MAINOKSET 21

5 PELIN JULKAISU 22

6 POHDINTA 24

LÄHTEET 25
LIITTEET

KUVIOT
KUVIO 1. Moduulit 4
KUVIO 2. Sovelluksen elämänkaari 6
KUVIO 3. Libgx Project Generator -sovellus 7
KUVIO 4. Pääluokkien luokkakaavio 10
KUVIO 5. Staattisten luokkien luokkakaavio 11
KUVIO 6. Pelin valikkonäkymä 12
KUVIO 7. Pelinäyttö 13
KUVIO 8. Resoluution hallinta 14
KUVIO 9. Kameran päivitys interpoloinnin avulla 15
KUVIO 10. Tiled-kenttäeditori 16

KUVIO 11. Box2d-fysiikkamoottori 17
KUVIO 12. Pelaajan määritys koodissa 18
KUVIO 13. Törmäysten käsittely koodissa 19
KUVIO 14. Partikkelieditori 20
KUVIO 15. Mainosyksikön luonti 21
KUVIO 16. Julkaisuversion luonti 22
KUVIO 17. Näkymä Google Play -palvelussa 23

1 JOHDANTO

Mobiilipelaaminen on yleistynyt ja kehittynyt samaan tahtiin, kuin mobiililaitteet, joista on tullut entistä tehokkaampia vuosi vuodelta. Yksinkertaiset 80- ja 90-luvun tietokonepelit ovat heränneet uudelleen henkiin uusina yksinkertaisina mobiilipeleinä. Nykyään kuka tahansa voi kehittää sovelluksen tai pelin ja julkaista sen johonkin mobiilisovellusten sisältöpalveluun, kuten Google Play -palveluun. Sovelluksilla on myöskin mahdollista ansaita mobiilimainosten ohella, jos sovelluksesta tulee suosittu ja sillä on paljon päivittäisiä käyttäjiä.

Mobiilipelien kehittäminen alusta alkaen on kuitenkin hyvin työläs projekti, varsinkin jos peli halutaan julkaista useammalle eri alustalle. Nykyään on monia erilaisia pelimoottoreita sekä sovelluskehyksiä, jotka nopeuttavat ja helpottavat pelien kehittämistä huomattavasti. Pelikehityksessä voidaan keskittyä itse olennaiseen ja ihan kaikkea ei tarvitse tehdä alusta alkaen.

Tämän opinnäytetyön tarkoituksena oli tutkia, kuinka Libgdx-sovelluskehystä voidaan käyttää apuna mobiilipelin ohjelmoinnissa Android-käyttöjärjestelmälle, sillä Android on tällä hetkellä ylivoimaisesti suosituin mobiilikäyttöjärjestelmä. Työn aihe syntyi kiinnostuksesta mobiilipeliohjelmointiin ja halusta luoda oma mobiilipeli Androidille ja myöskin julkaista se Google Play -palveluun. Sieltä kuka tahansa Android-puhelimen omistava pystyy sen ladata ilmaiseksi. Peli on suunniteltu niin, että se toimii lähes kaikissa Android-versioissa ja pelissä käytetyn sovelluskehyksen ansiosta peli on myös mahdollista julkaista muille mobiili- ja työpöytä-käyttöjärjestelmille tulevaisuudessa.

Työssä tutustutaan Libgdx-sovelluskehykseen, sen historiaan, ominaisuuksiin ja toimintaan. Seuraavaksi tutustutaan, kuinka opinnäytetyön ohessa syntynyt Crash Copter -pelin idea ja suunnitteluprosessi sujui ja lähti liikkeelle. Peli on Arcade-tyyppinen helikopteripeli, ja siinä ideana on lentää luolassa, kerätä kolikoita ja väistellä kiviesteitä. Pelin lopussa on maali, johon pelaaja voi laskeutua ja voittaa pelin.

Työssä tutustutaan myös siihen, kuinka Libgdx-projekti luodaan ja mitä se pitää sisällään. Samalla esitellään työkaluja, joita sovelluskehys tarjoaa, kuten esimerkiksi efektien luomiseen tarkoitettu partikkelieditori.

2 LIBGDX-FRAMEWORK

Libgdx on avoimeen lähdekoodiin perustuva Java-pohjainen sovelluskehys, joka on tarkoitettu helpottamaan ja nopeuttamaan peliohjelmointia. Sovelluskehys on järjestelmäriippumaton ja tukee useita eri alustoja (Nair & Oehlke 2015, 1). Tässä kappaleessa tutustutaan tarkemmin Libgdx-sovelluskehyksen historiaan ja tärkeimpiin ominaisuuksiin.

2.1 Historia

Mario Zechner aloitti Libgdx-sovelluskehyksen kehittämisen vuonna 2009, kun hän harjoitteli pelien ohjelmointia Androidille. Sovelluskehys oli aluksi nimeltään AFX (Android Effects). Android-pelien testaaminen puhelimessa oli erittäin hidasta noina alkuaikoina, joten Zechner teki muutoksia AFX-sovelluskehykseen, jotta pelin voisi kääntää suoraan myös työpöytäversioksi. Pelien kehitys nopeutui huomattavasti, kun peliä ei enää tarvinnut ladata puhelimeen tai emulaattoriin pienten muutosten testaamista varten. Vuonna 2010 sovelluskehyksen nimestä tuli Libgx ja siinä oli yksinkertaiset rajapinnat grafiikalle, äänille ja tiedostovirroille. Neljä vuotta myöhemmin kehittämisen alusta sovelluskehyksen 1.0-versio viimein julkaistiin. (Zechner 2014.) Tällä hetkellä uusin versio sovelluskehyksestä on 1.9.4, joka julkaistiin elokuussa 2016 (Zechner 2016).

2.2 Yleistä

Libgdx on järjestelmäriippumaton kaksi- ja kolmiulotteiseen peliohjelmointiin tarkoitettu avoimen lähdekoodin sovelluskehys. Libgdx on tehty tukemaan useita erilaisia laitteita, käyttöjärjestelmiä ja alustoja, joten se on järjestelmäriippumaton. Libgdx on suunniteltu niin, että koodin tarvitsee kirjoittaa vain kerran, jonka jälkeen on mahdollista kääntää projekti lukuisille eri alustoille Gradle-koontityökalun avulla. Tämä mahdollistaa myös sen, että peliä voidaan testata suoraan samalla tietokoneella, jolla sitä kehitetään. Se on huomattavasti nopeampaa kuin jatkuva emulaattorin käyttö, tai testaaminen oikealla Android-laitteella. Tällä hetkellä Ligdx tukee käyttöjärjestelmiä Mac, Linux, Android, iOS ja BlackBerry. Koodi voidaan myös kääntää HTML5-muotoon, jolloin se toimii lähes kaikissa HTML5-teknologiaa (JavaScript/WebGL) tukevissa Web-selaimissa. (Reich 2016.)

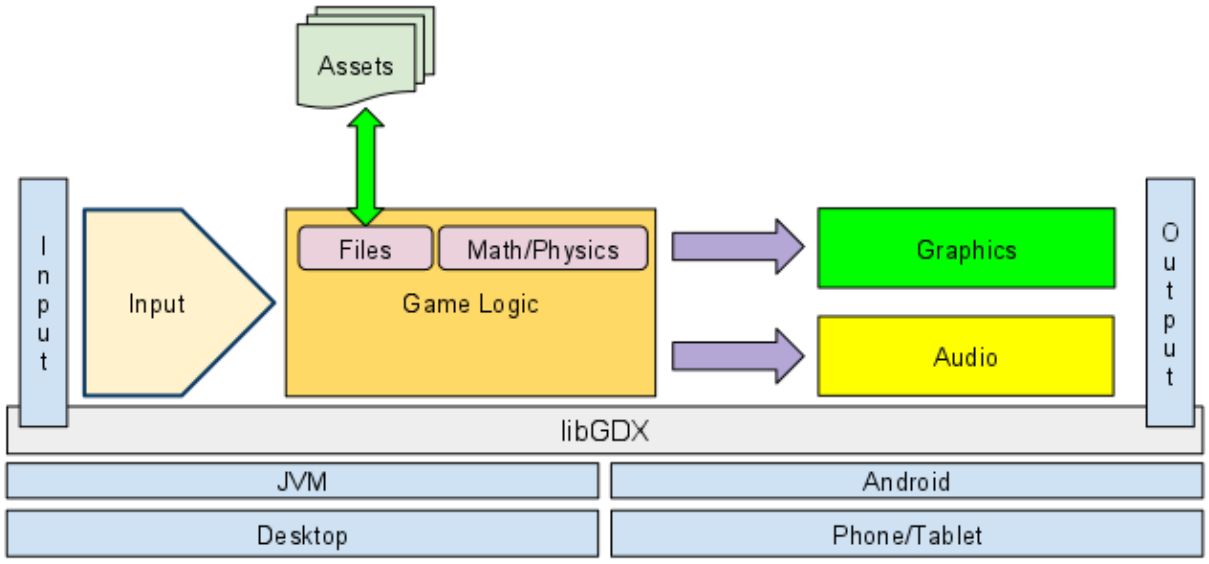
Libgdx käyttää hyväksi erilaisia taustajärjestelmiä järjestelmäriippumattomuuden takaamiseksi. Windows, Linux ja Mac-sovelluksissa käytetään Lightweight Java Game Libraryä (LWJGL). Tämä mahdollistaa muun muassa matalan tason OpenGL-kirjaston käytön pelissä. Androidilla käytetään Googlen tarjoamaa Android SDK:ta, ja iOS-sovelluksissa käytetään RoboVM-rajapintaa. HTML5-sovellukset käännetään Java-kielestä Javascriptiksi. Libgdx-sovelluskehystä käyttäessä lähdekoodi kirjoitetaan Java-ohjelmointikielellä. Osa sovelluskehyksen taustajärjestelmissä käyttää matalamman tason C/C++-kieltä JNI-rajapinnan avulla, joka nostaa suorituskykyä. (Reich 2016.)

Libgdx perustuu vapaaseen lähdekoodiin. Se on lisensoitu Apache 2.0 -lisenssin alle. Tämä tarkoittaa käytännössä sitä, että sovelluskehyksen käyttö on täysin ilmaista ei-kaupallisissa ja kaupallisissa peleissä, eikä pelissä ole välttämättä edes tarvetta mainita koko sovelluskehyksen käytöstä. (Zechner 2010.)

Sovelluskehyksestä on kirjotettu muutamia kirjoja ja sillä on varsin hyvät yhteisön ylläpitämät Wikisivut, jotka ohjeistavat ihan alkuaskelista vaativiin ohjelmointiaiheisiin. Yhteisöltä löytyy myös foorumisivusto kysymyksille ja IRC-kanava reaaliaikaiseen keskusteluun. (Zechner 2013.)

2.3 Moduulit

Sovelluskehys koostuu niin sanotuista ydinmoduuleista, jotka tarjoavat melkeinpä kaikki ne perusominaisuudet, joita tyypilliseen mobiilipelin kehitykseen tarvitsee. (KUVIO 1.) Nämä ominaisuudet toimivat täysin samalla tapaa alustasta riippumatta. Moduulit ovat staattisia ja ne on myöskin määritelty globaaleiksi. Moduulien funktioita voidaan siis kutsua missä tahansa kohtaa koodia Gdx-luokan kautta. (Chen 2015.)



KUVIO 1. Moduulit (Modules overview 2015)

Syötemoduuli mahdollistaa syötteen nappaamisen käyttäjältä. Työpöytäversiossa voidaan käyttää hiirtä ja näppäimistöä, kun taas mobiilisovelluksessa kosketusta ja kiihtyvyysanturia. Mobiililaitteella sormen painallus näytölle tulkitaan samaksi asiaksi kuin tietokoneella hiiren painallus peli-ikkunaan. (Chen, S. 2016.) Syötemoduuliin päästään käsiksi Gdx.input-viittauksella. Parempi tapa syötteen nappaamiseen on kuitenkin implementoida oma InputProcessor-rajapinta ja asettaa se globaaliksi syötteen kuuntelijaksi Gdx.input.setInputProcessor-metodin avulla. (Nair & Oehlke 2015, 89.)

Sovellusmoduulin avulla voidaan selvittää, missä alustassa sovellus on käynnissä. Moduuli saa myös tiedon sovellustason tapahtumista, kuten näytön resoluution muutoksesta. Sovellusmoduulin kautta päästään myös käsiksi log-metodiin, jonka avulla voidaan kätevästi tulostaa tietoa pelin ajonaikaisista tapahtumista ja virheistä. Moduulin kautta voidaan myös tallentaa asetuksia pysyvästi laitteen muistiin, kuten esimerkiksi pelin äänen voimakkuus ja huipputulokset. Moduulin avulla voidaan myös ajaa koodia omassa säikeessä Gdx.app.postRunnable-metodin avulla. Moduulissa on myös exit-metodi, jota kutsumalla sovellus tulee aina lopettaa, jottei sovellus ei vuoda muistia. (Nair & Oehlke 2015, 85-87.)

Grafiikkamoduuli piilottaa taakseen matalan tason OpenGL/OpenGL ES-rajapinnan, jonka avulla kaikki näytölle piirtäminen tapahtuu. Moduulin mukana on myös joukko hyödyllisiä metodeja, joilla saadaan

esimerkiksi tietoa näytön resoluutiosta, pikselitiheydestä, näytön suuntautumisesta ja käytössä olevista OpenGL/OpenGL ES-ominaisuuksista. (Kane 2014.)

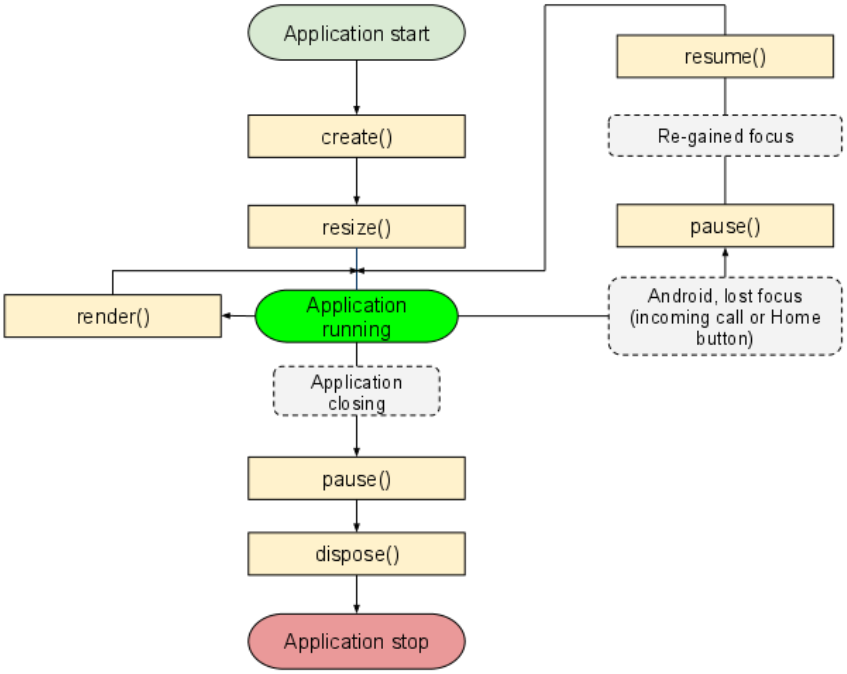
Tiedostomoduuli mahdollistaa helpon tiedostojen käsittelyn. Tällaisia toimitapoja ovat tiedoston lukeminen, kirjoittaminen tiedostoon, tiedoston kopiointi tai siirtäminen ja kansion tiedostojen listaaminen. (Engstler 2014.)

Verkkomoduuli mahdollistaa yksinkertaiset HTTP-pyynnöt ja TCP-asiakas/palvelin-toteutuksen. Moduulin avulla on myös mahdollista avata järjestelmän Web-selain. Esimerkiksi pelissä voi olla linkki pelin kotisivuille, jonka klikkaaminen avaa järjestelmän oman selaimen kyseiseen osoitteeseen. (Nair & Oehlke 2015, 91.)

Äänimoduulin avulla voidaan pelissä toistaa ääntä ja musiikkia. Libgdx tukee tällä hetkellä MP3-, OGG- ja WAV-muodossa olevia äänitiedostoja. Poikkeuksena RoboVM ei tue OGG-tiedostomuotoa lainkaan. Androidissa Libgdx Sound -luokka ei pysty lataamaan yli yhden Megatavun tiedostoja. Isommat tiedostot ladataan Music luokan avulla. (Silverman 2014.)

2.4 Sovelluksen elinkaari

Libgdx -ohjelman elinkaari on tapahtumaohjattu johtuen Android-käyttöjärjestelmän rakenteesta. Kuvio 2 kuvaa ohjelman elinkaarta. Kun ohjelma käynnistetään, ensimmäisenä suoritetaan create-metodi, jossa yleensä ladataan tiedostot ja luodaan kaikki objektit myöhempää käyttöä varten. Seuraavaksi ajetaan resize-metodi, jossa on parametreina leveys ja korkeus pikseleinä. Seuraavana suoritetaan render-metodi, joka on pelin pääsilmukka. Silmukassa yleensä napataan syötettä käyttäjältä, päivitetään objektit, pelimekaniikka ja piirretään grafiikkaa näytölle. Pause-metodiin siirrytään, kun Android-käyttäjä saa vaikkapa puhelun tai siirtyy sovelluksesta toiseen hetkeksi. Tietokoneella pause-metodiin siirrytään, kun peli-ikkuna ei ole aktiivisena ikkunana. Vain Android-laitteissa kutsutaan resume-metodia, kun palataan peliin. Aina kun sovellus suljetaan, kutsutaan dispose-metodia, jossa tuhotaan objektit ja vapautetaan varattu muisti. (Chen 2015.)



KUVIO 2. Sovelluksen elämänkaari (The life cycle 2015)

2.5 Projektin luominen

Libgdx -projektin luontia varten tarvitaan Libgdx Project Generator -niminen ohjelma, joka on vapaasti ladattavissa Libgdx-projektin kotisivuilta. (KUVIO 3.) Ohjelmassa määritetään projektin nimi, Java-paketin nimi, kansio johon projekti luodaan ja myös Android SDK:n sijainti. Projektiin voi ottaa myös mukaan lukuisia kolmannen osapuolen lisäosia, kuten Box2d-fysiikkamoottori, jota käsitellään tarkemmin myöhemmässä luvussa.

KUVIO 3. Libgx Project Generator -sovellus

2.5.1 Ohjelmointiympäristö

IDE:ksi valitsin Googlen tarjoaman Android Studion, joka perustuu JetBrains-yhtiön IntelliJ IDEA nimiseen Java-ohjelmointiympäristöön. Muita hyviä ohjelmistoympäristöjä, jotka toimivat yhdessä Libgdx-projektin kanssa olisi ollut esimerkiksi Eclipse tai Netbeans. Koska projekti perustuu Gradleen, IDE ei ole täysin pakollinen. Se on kuitenkin hyvin suotuisaa, sillä IDE:t tarjoavat luonnollisesti paljon ominaisuuksia, jotka helpottavat ohjelmoijan työtä. Gradle -koontityökalun käyttö mahdollistaa myös

sen, että jos peliä kehitetään tiimissä. Tällöin kaikilla tiimin jäsenillä ei ole pakko olla sama ohjelmointiympäristö käytössä, vaan jokainen jäsen voi käyttää kehitykseen juuri sitä ympäristöä, mistä itse eniten pitää.

2.5.2 Gradle

Gradle on komentoriviohjelmana toimiva Java-ohjelmien koontityökalu, joka pyrkii yhdistämään vanhempien Apache Ant- ja Apache Maven-työkalujen hyviä puolia yhteen. Gradle:lla on *build.gradle*-niminen koontitiedosto, joka on kirjoitettu Groovy-nimisellä scriptikielellä. Gradle osaa hakea riippuvuudet suoraan tietovarastosta, ja Gradle tukee sekä Mavenin että Antin tietovarastoja.

Lisäkirjastoja saa siis kätevästi lisättyä omaan projektiin lisäämällä niitä Gradlen koontietiedostoon. Gradlella voi kirjoittaa taskeja, jotka voivat olla myös riippuvaisia toisistaan. Kaikki taskit saadaan selville komentorivistä komennolla gradle tasks. Oma peli on paketoitava, jotta sen voisi suorittaa esimerkiksi Windows-tietokoneella. Pelistä voi luoda suoritettavan JAR-paketin ajamalla komentorivistä gradle desk:dist-komennon. Komento luo JAR-paketin projektin desktop/dist/libs-kansioon. Monissa IDE-työkaluissa on kuitenkin graafinen liittymä näiden taskien suorittamiseen.

2.5.3 Projektin rakenne

Libgdx-projektin rakenne koostuu laajasta kansiorakenteesta, jossa alustakohtainen koodi ja tiedostot on sijoitettu oman kansiorakenteen sisälle ja kaikille alustoille yhteinen koodi on core-nimisen kansion alla. Oletuksena pelin grafiikat ladataan android-nimisen kansion sisällä olevasta assets-kansiosta. Liite 1 kuvaa Libgdx-projektin kansiorakennetta.

3 PELIN TOTEUTUS

3.1 Pelin idea

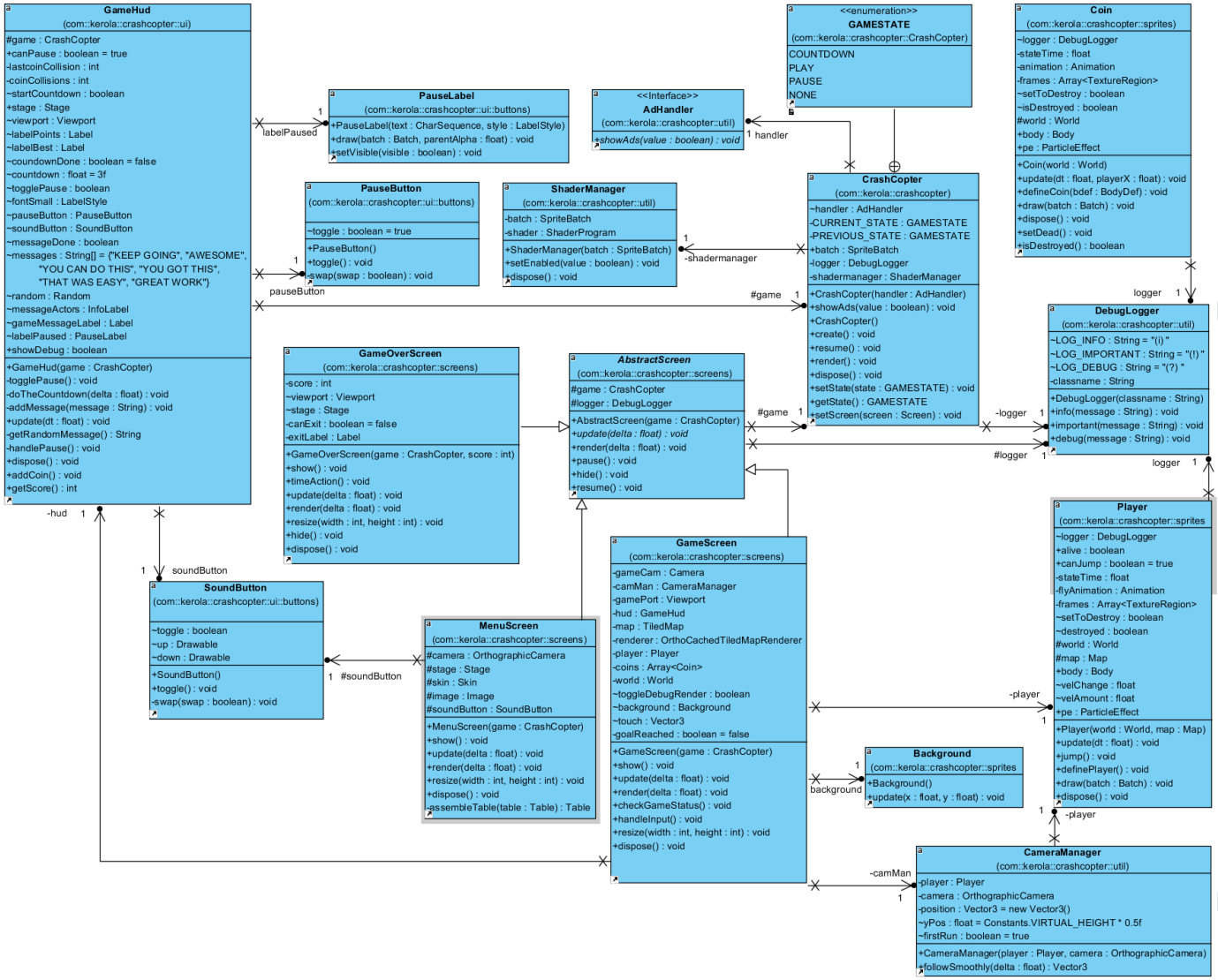
Pelin ideana oli toteuttaa klassinen haastava kaksiulotteinen peli, jossa lennetään rikkinäisellä helikopterilla luolassa väistellen luolan seinämiä. Helikopteria voidaan nostaa koskettamalla näyttöä. Helikopteri alkaa laskeutua välittömästi, kun kosketus lakkaa näytöltä. Pelin haastavuus tulee siitä, että tulee osata arvioida ja ajoittaa nouseminen ja laskeminen oikein. Luolassa on myös kolikoita, joita pelaajan tulee kerätä. Pelin ideana oli olla haastava, muttei kuitenkaan mahdoton, joten peli on mahdollista päästä läpi selviytymällä tarpeeksi pitkään hengissä. Lopussa tulee maali, johon helikopteri laskeutuu.

3.2 UML-luokkakaaviot

Pelin toiminnan tarkastelu on helpoin aloittaa UML-luokkakaavioista, jotka antavat nopeasti yleiskuvan pelissä olevista luokista, niiden välisistä suhteista ja vuorovaikutteista. Luokkakaaviot on jaettu kahteen eri osaan. Kuvio 4 osoittaa pelin tärkeimmät pääluokat ja kuvio 5 osoittaa pelin staattiset luokat, jotka ovat suurimmaksi osaksi apuluokkia.

3.2.1 Pelin pääluokat

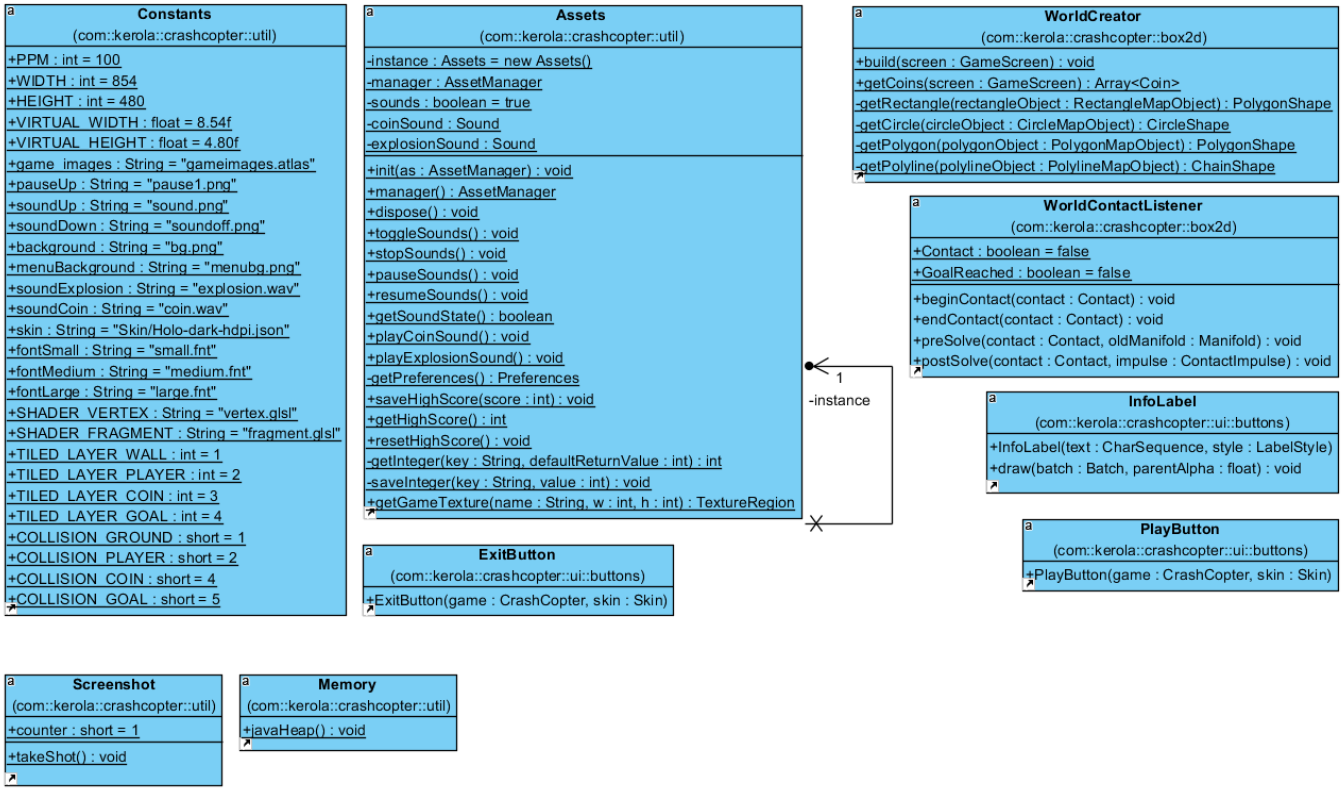
Kuvio 4 osoittaa pelin päätoiminnan ja luokkien riippuvuussuhteen UML-luokkakaaviona. Pelin pääluokkana toimii CrashCopter-luokka, joka periytyy Libgdx Game-luokasta. Luokka sisältää pelitilat, joita käytetään hyväksi pelinäytössä. Pääluokan viittaus annetaan jokaiselle näyttöluokalle. Näyttöluokkia pelissä ovat pelin valikko, itse peli ja lopetusnäyttö. Näyttöluokat periytyvät Libgdx Screen-luokasta, lisäksi on käytetty abstraktista luokkaa apuna, jonka avulla näyttöluokkiin on lisätty update-metodi, joka suoritetaan aina render-metodin alussa. Lisätyn metodin avulla on pelissä erotettu päivitys ja piirtäminen toisistaan.



KUVIO 4. Pääluokkien luokkakaavio

3.2.2 Pelin staattiset apuluokat

Kuvio 5 osoittaa pelin staattiset luokat, jotka ovat niin sanottuja apuluokkia. Assets-luokassa hoidetaan pelin resurssien lataus, äänten toisto ja myös asetusten ja pisteiden tallennus laitteen pysyvään muistiin. Ainokainen (Singleton) suunnittelumalliin perustuva Assets-luokka varmistaa sen, että siitä on ainoastaan yksi konkreettinen ilmentymä pelin ajonaikana. Constants-luokka sisältää kaikki tiedostojen nimet, pelimaailman oikeat mitat ja virtuaaliset mitat. Luokassa on myös Box2d-olioille tarkoitetut törmäysbitit, joiden avulla määritellään, mitkä oliot voivat törmätä toistensa kanssa. WorldCreator-luokka luo pelimaailman ja WorldContactListener-luokka saa tiedon pelissä tapahtuvista törmäyksistä ja käsittelee ne. Törmäystietojen mukana on myöskin viitteet niihin objekteihin, jotka olivat mukana törmäystapahtumassa.



KUVIO 5. Staattisten luokkien luokkakaavio

3.3 Pelin valikko

Pelin valikko on ensimmäinen pelinäkymä, joka tulee näkyville pelin käynnistyksen jälkeen. (KUVIO 6.) Valikkonäkymässä on taustakuva ja kolme painonappia. Valikkoruudussa alkaa myös leijailla hiukkasia pienen viiveen jälkeen. Hiukkasefektitien toteutus on kuvailtu myöhemmässä kappaleessa. Pelaaja voi siirtyä peliin, lopettaa pelin ja vaihtaa äänien tilaa. Valikko on toteutettu Libgdx Scene2D -kirjaston avulla, joka on tarkoitettu yksinkertaisten kaksiulotteisten näyttökomponenttien toteutukseen. Painonapeissa on myöskin animaatio joka liu'uttaa ne näytölle pienellä viiveellä. Äänten tila tallennetaan laitteen pysyvään muistiin Assets-apuluokan avulla. Liitteenä 2 on koodiesimerkki play-napin toteutuksesta.

KUVIO 6. Pelin valikkonäkymä

3.4 Pelinäyttö

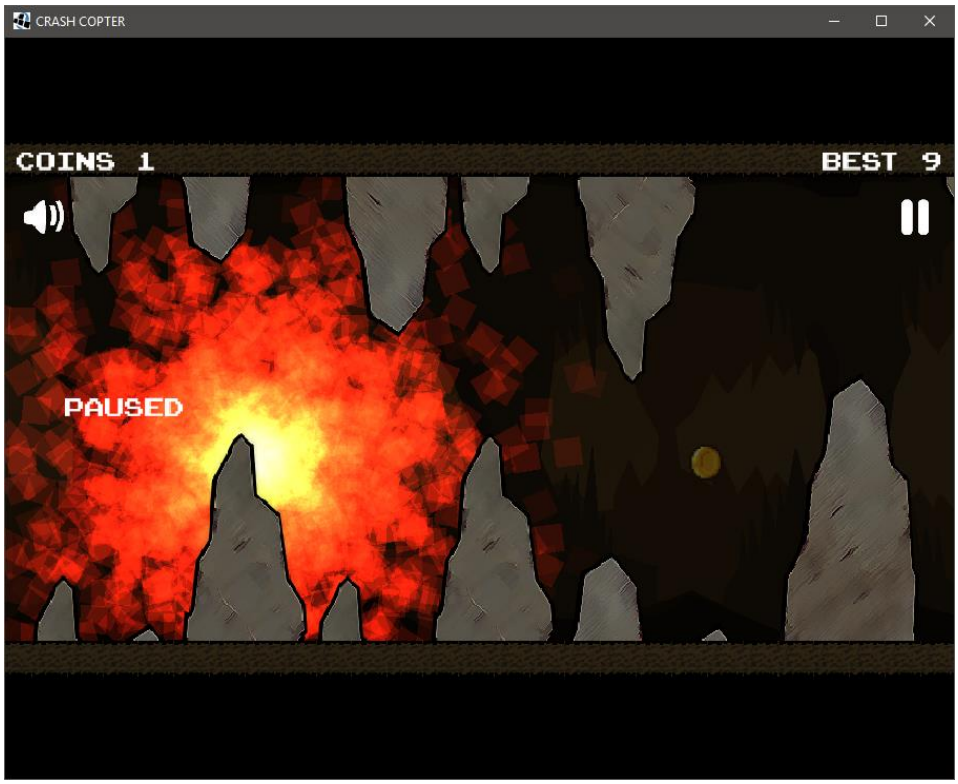
Pelinäytössä helikopteri on paikoillaan, kunnes lähtölaskenta on suoritettu. Lähtölaskennan jälkeen helikopteri alkaa laskeutua hiljalleen ja pelaajan tulee koskettaa näyttöä, jotta helikopteri alkaisi nousta hyvin hitaasti. Näytölle ilmestyy kolikoita, joita keräämällä voi kasvattaa pelipisteitä.

Pelissä on erillinen HUD (Head-Up Display), joka ilmoittaa pelaajan nykyiset pisteet ja aikaisemman huipputuloksen. (KUVIO 7.) HUD sisältää myös äänten mykistyksen ja pause-napin. HUD:in piirron päivitys on rajoitettu 30 kertaan sekunnissa. Itse peliä pyritään päivittämään 60 kertaa sekunnissa. HUD sisältää myös pelin viestijärjestelmän. Pelissä pelaaja saa 7 kolikon välein kannusteviestin, joka kannustaa pelaajaa jatkamaan pelaamista. Näytölle tulee myös ilmoitus, jos pelaaja on mykistänyt äänet tai laittanut pelin pause-tilaan. Viestijärjestelmä on tehty käyttäen jono-tietorakennetta, uusi viesti näytetään vasta, kun vanha viesti on poistunut näytöltä.

KUVIO 7. Pelinäyttö

3.5 Resoluutio ja kamera

Nykyään on paljon erilaisia mobiililaitteita eri kokoisilla näytöillä, ja se onkin otettava huomioon pelinteossa. Libgdx tarjoaa resoluution hallintaan useita eri ratkaisuja Viewport-luokilla. Näitä näkymäporttiluokkia ovat ScreenViewport, FitViewport ja ExtendViewport. Pelissä päädyin käyttämään Fitviewport-luokkaa, joka sovittaa skaalaamalla pelin näytölle aina oikeassa kuvasuhteessa, jos laitteessa on erilainen kuvasuhde, kuin pelissä niin ilmestyy peliin mustat reunat. Kuviosta 8 voi huomata mustat reunat pelin ylä- ja alapuolella, kun peli on yritetty istuttaa väärään resoluutioon.



KUVIO 8. Resoluution hallinta

Pelissä on käytetty sovelluskehyksen tarjoamaa ortografista kameraa, joka seuraa tasaisesti helikopteria. Kameraa ei voi päivittää suoraan pelaajan koordinaatteihin, koska jollain laitteella kuva saattaisi näyttää siltä, että se tökkii pahasti. Pelin pääsilmukassa saadaan selville delta-nimiseen muuttujaan tieto siitä, kuinka monta sekuntia viimeisestä ruudunpäivityksestä on mennyt. Tämä arvo voi vaihdella hieman laitekohtaisesti, joka pahimmillaan voi aiheuttaa pelissä hyvinkin tökkivän tunnelman. Deltamuuttujaa käytetään apuna lineaarisessa interpoloinnissa, jonka avulla saadaan kamera liikkumaan tasaisesti, vaikka kuvanpäivitys olisi hieman epätasaista.

Pelissä käytetty metodi kameran päivittämiseen on esitetty kuviossa 9.

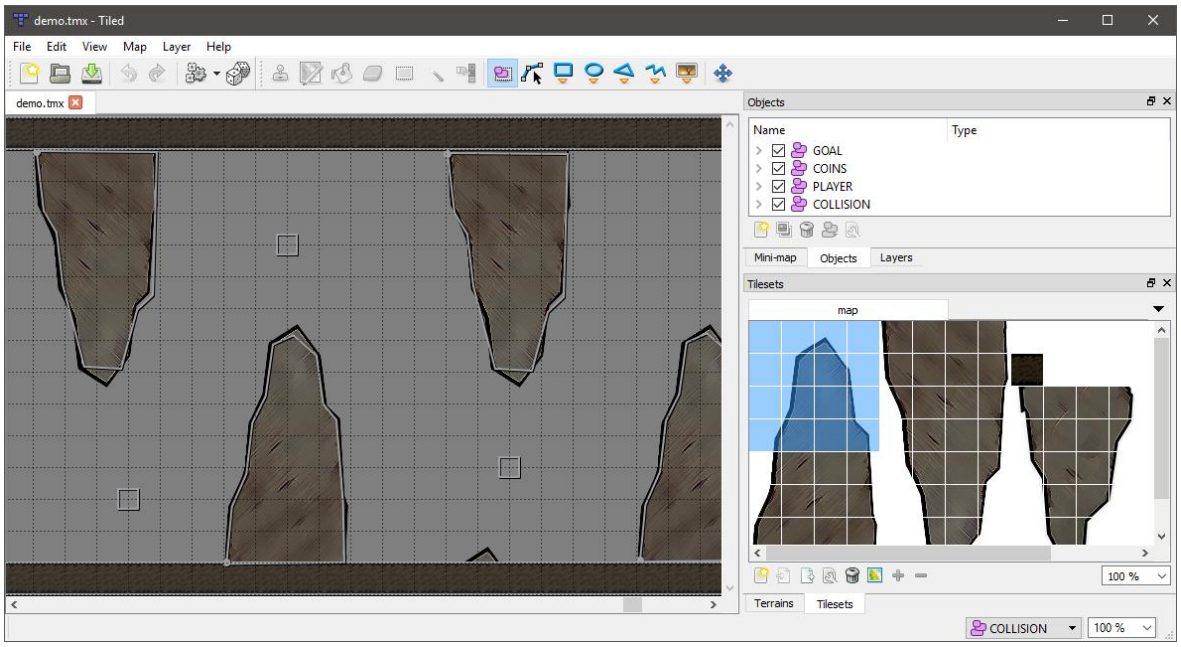
```
public Vector3 followSmoothly(float delta) {

    position = camera.position.lerp(
        new Vector3(player.body.getPosition().x, yPos, 0),
    .3f * delta);

    return position;
}
```

KUVIO 9. Kameran päivitys interpoloinnin avulla

3.6 Kenttä

Pelin kenttä on tehty Tiled-nimisellä ohjelmalla, joka perustuu vapaaseen lähdekoodiin. Tiled on yleismaailmallinen kaksiulotteisten pelien kenttäeditori, joka toimii useiden eri pelimoottoreiden kanssa. Kentän rakentaminen editorilla on huomattavasti helpompaa entä sen teko suoraan pelin lähdekoodissa. (KUVIO 10.)

Pelin kaikki kenttägrafiikat on pakattu yhteen kuvaan resurssien säästämisen ja myöskin yksinkertaisuuden takia. Kenttä koostuu 32x32 pikselin ruudukoista ja kenttä on 15 ruudukkoa korkea ja 800 ruudukkoa pitkä. Kuvien päälle voi lisätä kerroksia, kuten rajoja törmäyksentunnistukseen ja mihin kohti kenttää kolikoiden ja pelaajan tulisi ilmestyä. Pelissä on vain yksi kenttä, mutta se on hyvin pitkä. Lopussa tulee kuitenkin vastaan maali, johon helikopteri laskeutuu. Pelaaja saa maaliin päästyä erilaisen loppuruudun, jossa onnitellaan pelin läpäisystä.



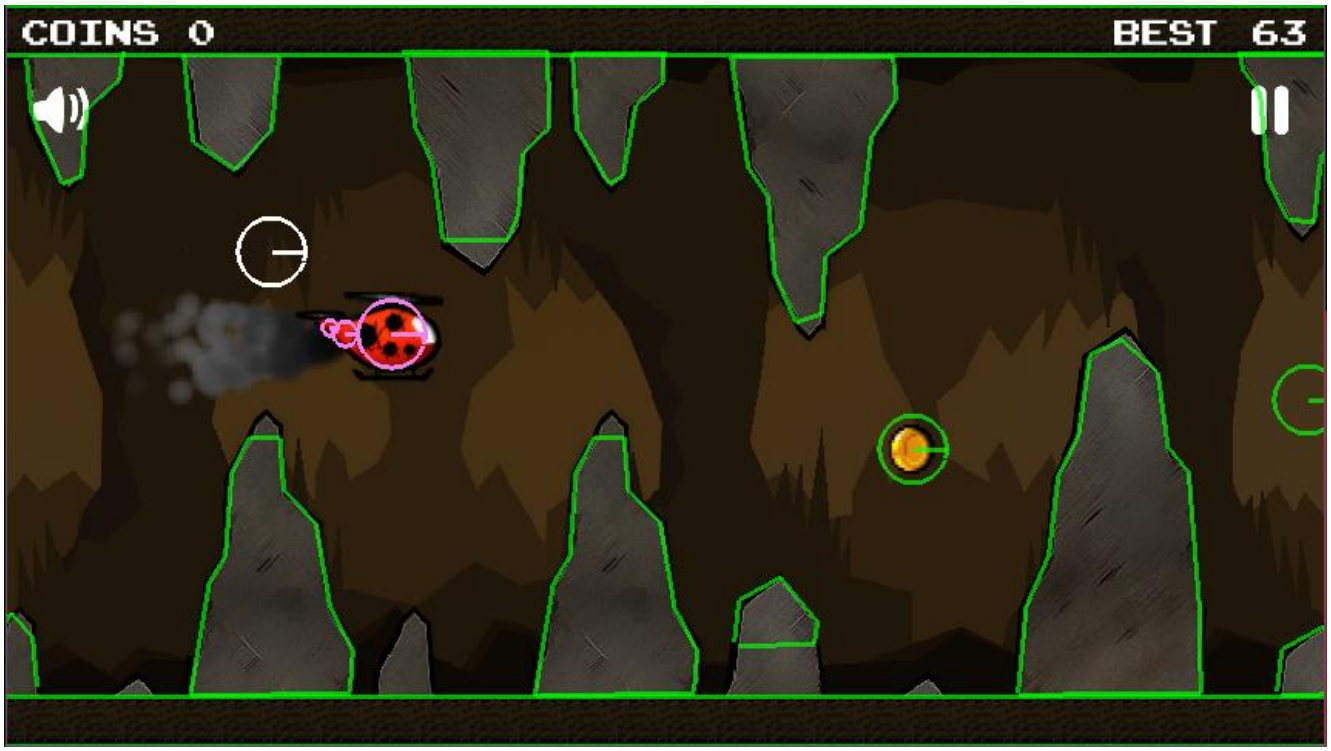
KUVIO 10. Tiled-kenttäeditori

3.7 Box2d-fysiikkamoottori

Pelin fysiikat ja törmäyksentunnistus on toteutettu Box2d-fysiikkamoottorilla. Box2d:n avulla voidaan simuloida kaksiulotteisia jäykkiä objekteja. Simulaatio tapahtuu omassa maailmassa, johon kaikki objektit tulee lisätä, jotta ne olisivat mukana simulaatiossa. Jokaisella Box2d-objektilla tulee olla body, joka sisältää sijainnin ja fysikaaliset ominaisuudet, kuten esimerkiksi sen massan, nopeuden ja kitkan. Box2d-objektilla voi olla staattinen, dynaaminen tai kinemaattinen body. Dynaaminen body on tarkoitettu esimerkiksi pelaajalle. Siihen vaikuttavat kaikki voimat, ja se voi liikkua ja pyöriä voimien vaikutuksesta. Staattinen body ei taas reagoi mihinkään voimiin, ja sitä käytetään yleisesti kiinteissä tasoissa, jotka eivät muutu tai liiku lainkaan. Kinemaattinen body on näiden kahden edellä mainitun välimuoto. Se ei reagoi mihinkään voimiin, mutta sille voidaan määrätä tietty liikerata. Se voi olla pelissä esimerkiksi kiinteä alusta, jolle on ennalta määritelty edestakainen liikerata.

Kappaleen muoto ja ominaisuudet ovat määrätty body-objektiin liitetystä fixturesta. Kappale voi koostua yhdestä tai useammasta muodosta, kuten pelin pelaaja koostuu kolmesta eri ympyrämuodosta. (KUVIO 11.) Fixturen voi määritellä myös sensoriksi, jolloin se ei törmää fysikaalisesti toisiin objekteihin, mutta törmäys silti käsitellään. Sensoreita on käytetty pelin kolikoissa. Kolikot halutaan tunnistaa, mutta niihin ei haluta törmätä.

Pelissä on käytetty virtuaalista resoluutiota, koska Box2d toimii metriarvoilla ja aidot pikseliarvot olisivat aivan liian suuria sille. Fysiikkamoottorin manuaalissa suositellaan käyttämään 0.1 – 10 metrin kokoisia Box2d-olioita parhaan suorituskyvyn takaamiseksi. Pelin alkuperäinen resoluutio on 854x480 pikseliä ja siitä on tehty virtuaalinen resoluutio fysiikkamoottorille skaalaamalla pikselit metreiksi käyttäen keksittyä lukuarvoa, jolla Box2d-objekteista tulee sopivan kokoisia. Kaikki pikseliarvot on jaettu lukuarvolla 100, jotta ne olisivat sopivia fysiikkamoottorille. Pelissä pelaaja on dynaaminen, luolan seinämät ja kivet ovat staattisia, kun taas kolikot ja maali ovat sensoreita. (KUVIO 11.)



KUVIO 11. Box2d-fysiikkamoottori

Pelin törmäyksentunnistus on toteutettu siten, että Box2d-objektien fixturelle on annettu kategoriabitit ja törmäysbitit. Kategoriabitit määräävät käytännössä objektin identiteetin, mihin ryhmään se kuuluu. Törmäysbittien avulla määrätään, minkä ryhmien kanssa kyseinen ryhmä saa törmätä. Seuraavassa koodiesimerkissä kuvataan, kuinka pelaaja määritellään Box2d-maailmaan ja määritellään, että pelaaja pystyy törmätä kolikoiden, maan ja maalin kanssa.

Pelaajan Box2d-objektin määrittely koodissa:

```
// Luodaan dynaaminen kappale simulaatioon

BodyDef bdef = new BodyDef();
bdef.type = BodyDef.BodyType.DynamicBody;
body = world.createBody(bdef);

// Määritetään muoto ja koko
CircleShape shape = new CircleShape();
shape.setRadius(22f/Constants.PPM);
shape.setPosition(shape.getPosition().add(0.15f, 0.05f));

[ . . . ]

// Annetaan kategoriabitit, törmäysbitit ja viittaus itseensä

FixtureDef fdef = new FixtureDef();
fdef.shape = shape;
fdef.filter.categoryBits = Constants.COLLISION_PLAYER;
fdef.filter.maskBits = Constants.COLLISION_GROUND | Constants.COLLISION_COIN
                | Constants.COLLISION_GOAL;
body.createFixture(fdef).setUserData(this);

[ . . . ]
```

KUVIO 12. Pelaajan määritys koodissa

Kun törmäys tapahtuu, niin staattinen WorldContactListener -luokka saa tiedon näistä kahdesta objektista, jotka ovat törmänneet ja törmäys käsitellään kyseisen luokan avulla. Objektien törmätessä tarkastellaan ensin, onko esimerkiksi kolikko törmännyt pelaajaan vai pelaaja kolikkoon. Kolikko asetetaan kuolleeksi ja myöhemmin kuolleeksi asetetut kolikot tuhotaan simulaatiosta ja pelaajan pisteitä kasvatetaan samalla.

Törmäysten käsittely pelissä on toteutettu seuraavalla tavalla:

```
@Override
 public void beginContact(com.badlogic.gdx.physics.box2d.Contact contact) {


     Fixture fixA = contact.getFixtureA();
     Fixture fixB = contact.getFixtureB();

     int collisionbits = fixA.getFilterData().categoryBits |
     fixB.getFilterData().categoryBits;

     if ( collisionbits == (Constants.COLLISION_PLAYER |
                     Constants.COLLISION_COIN) ) {

         if(fixA.getFilterData().categoryBits == Constants.COLLISION_COIN)
             ((Coin) fixA.getUserData()).setDead();
         else
             ((Coin) fixB.getUserData()).setDead();

     } else if (collisionbits == (Constants.COLLISION_PLAYER |
                     Constants.COLLISION_GROUND)) {
         if(!GoalReached)
             Contact = true;

     } else if (collisionbits == (Constants.COLLISION_PLAYER |
                     Constants.COLLISION_GOAL)) {
         GoalReached = true;
     }

 }
```

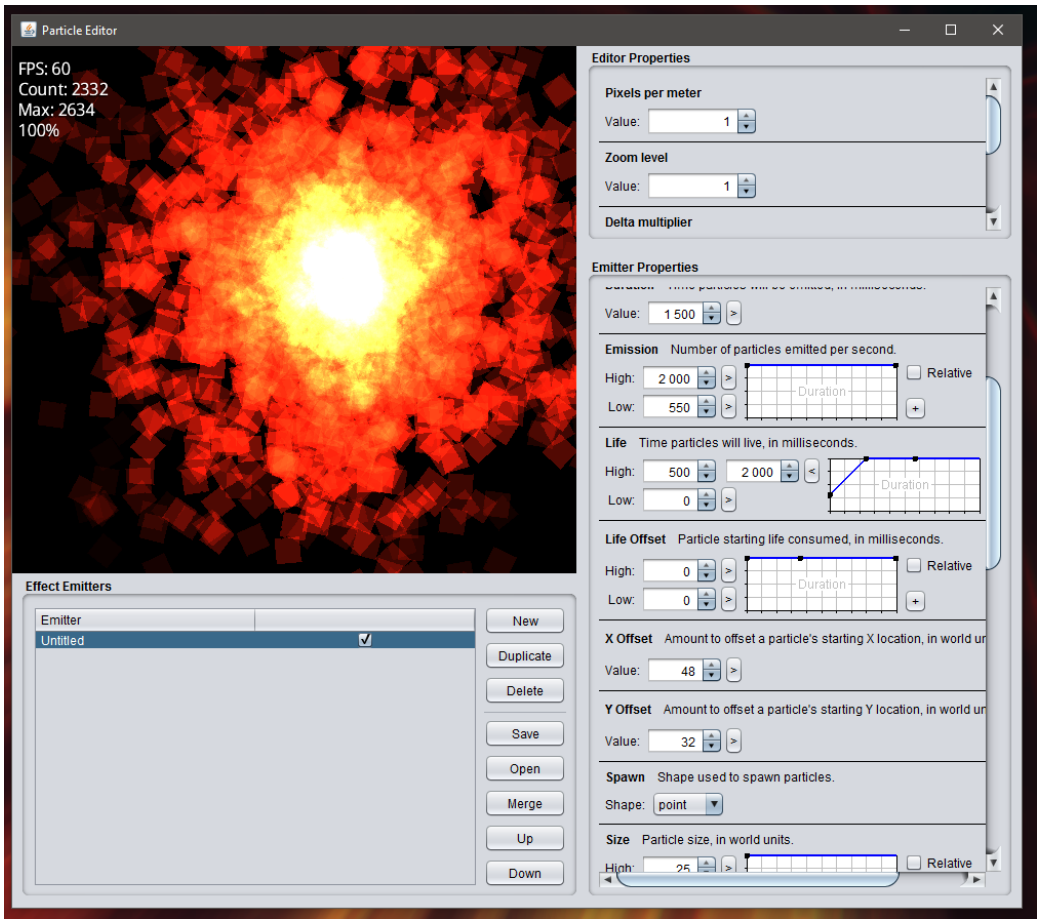
KUVIO 13. Törmäysten käsittely koodissa

3.8 Partikkeliefektit

Pelissä on käytetty Libgdx-sovelluskehyksen omaa 2D-partikkelieditoria, joka on vapaasti ladattavissa projektin kotisivuilta. Pelissä on useita erilaisia partikkeliefektejä. Pelin valikossa leijailee partikkeleita,

pelaajan helikopterista tulee savua ja sen osuessa kallionseinämiin tulee räjähdysefekti. Näytölle ilmestyvissä kolikoissa on myös efekti, joka saa ne kimaltelemaan ja kun helikopteri osuu niihin, syntyy poksahdus efekti. Nämä efektit on luotu sovelluskehyksen omalla partikkelieditorilla.

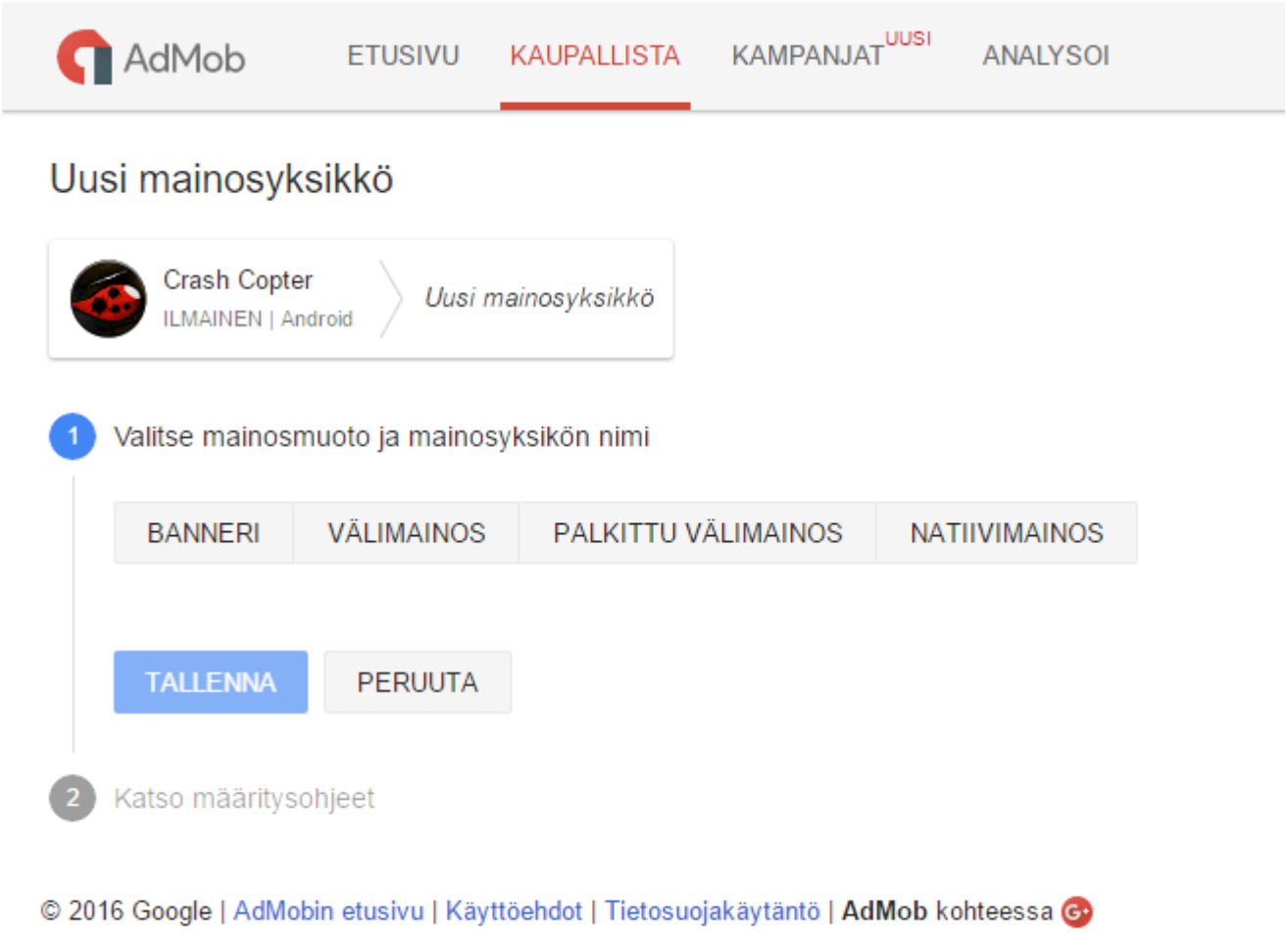
Editorilla voi suunnitella haluamansa efektin hyvinkin tarkasti. Voidaan määrittää esimerkiksi partikkelien väri, koko, määrä, suunta, nopeus ja elinikä. Määrityksessä annetaan korkein ja matalin arvoja, joiden avulla saadaan jokainen efekti myös näyttämään hieman erilaiselta. Pelaajan osuessa kallionseinämiin tulee räjähdys, joka on jokaisella kerralla hieman erilainen.



KUVIO 14. Partikkelieditori

4 ADMOB-MAINOKSET

AdMob on Googlen tarjoama mobiilimainospalvelu. Ilmaispelillä on mahdollista myös ansaita mainosten ohella, jos pelillä on tarpeeksi päivittäisiä käyttäjiä. Mainoksia varten tulee rekisteröityä https://apps.admob.com-osoitteessa, jonka jälkeen lisätään pelin tiedot sinne joko manuaalisesti tai Google Play-palvelun kautta ja luodaan uusi mainosyksikkö. (KUVIO 15.) Mainosmuotoja on banneri, välimainos, palkittu välimainos sekä natiivimainos. Bannerimainoksen ja natiivimainoksen koon ja sijainnin saa valita. Natiivimainokset ovat sovelluksen ulkoasua vastaaviksi muokattuja mainoksia. Välimainokset ovat koko sivun mainoksia, jotka voivat sisältää tekstiä ja videota.



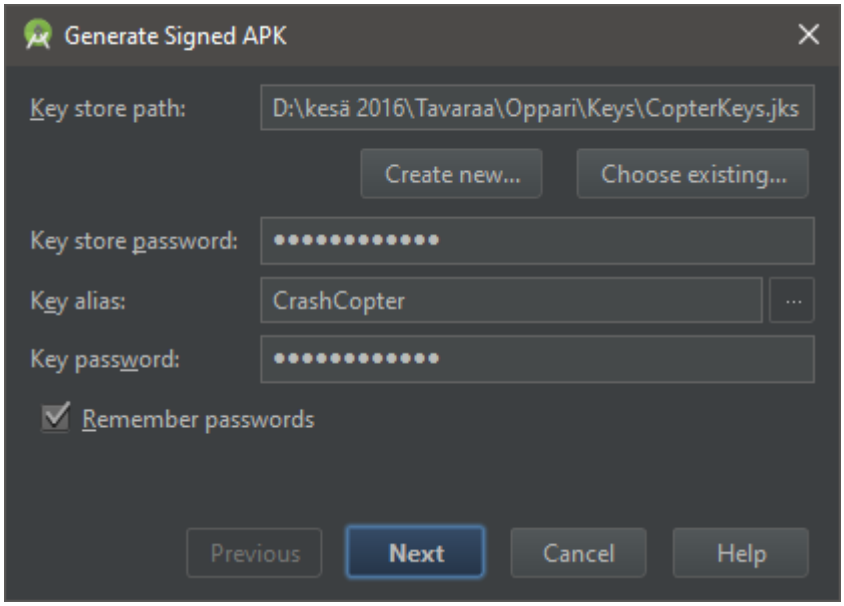
KUVIO 15. Mainosyksikön luonti

Kun mainosyksikkö on luotu, niin saadaan selville sen tunnus ja integrointiohjeet koodiosuudelle. Tunnusta tarvitaan, kun mainososuus koodataan peliin. Koodissa tarkistetaan ensin, onko pelaajan laite yhteydessä verkkoon, jos ei, niin mainosta ei haeta eikä näytetä. Pelin välimainos näytetään joka toinen kerta, kun pelissä siirrytään loppuruudusta pelivalikkoon. Liitteessä 3 on koodiesimerkki metodista, joka lataa ja näyttää välimainoksen sekä tarkistaa, onko laite yhteydessä mobiili- tai WLAN-verkkoon.

5 PELIN JULKAISU

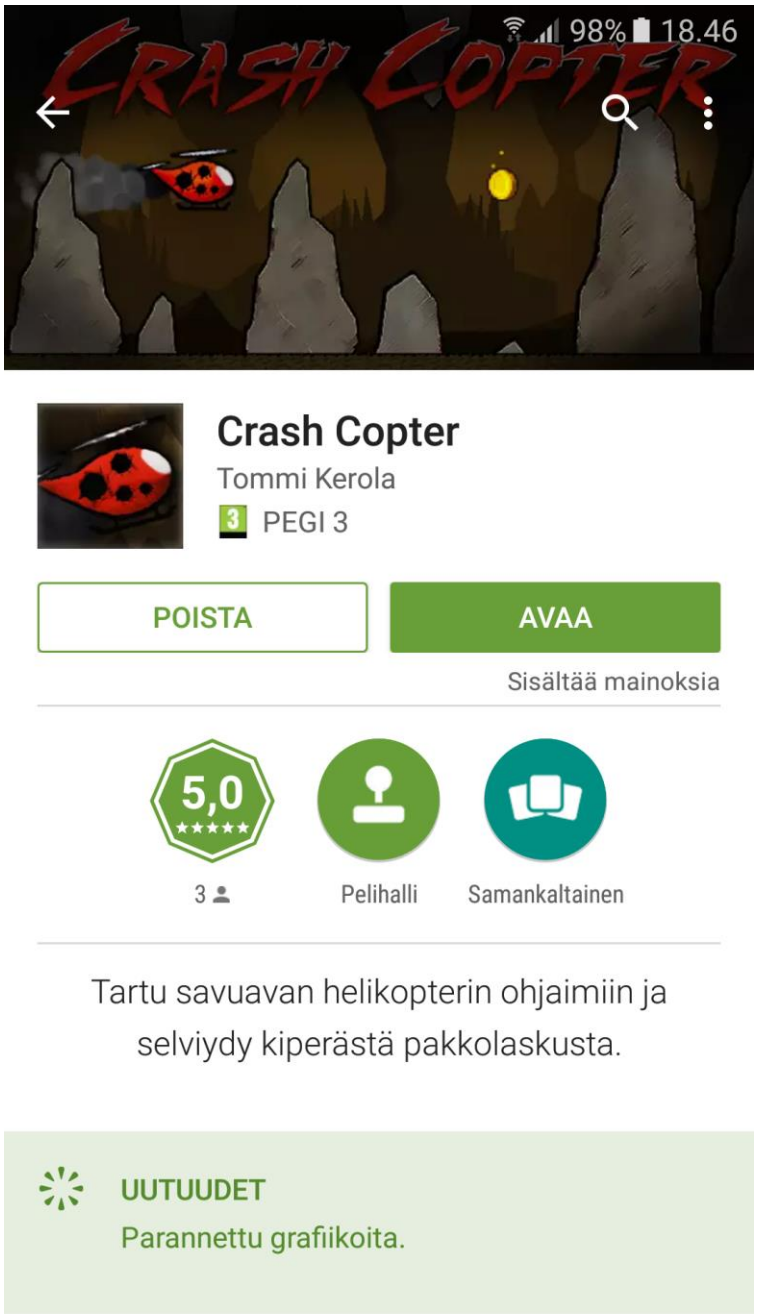
Pelin lataaminen Google Play -palveluun vaatii rekisteröitymisen Google Play Developer Consolessa, joka maksaa 25 USD. Rekisteröityminen viralliseksi sovelluskehittäjäksi tapahtuu osoitteessa https://play.google.com/apps/publish/signup/. Rekisteröinti tapahtuu Googlen Gmail-osoitteella ja maksaminen käy varsin nopeasti, jos on jo ostanut jotain aikaisemmin Google Play -palvelusta. Maksu on elinikäinen kertamaksu. Rekisteröitymisen jälkeen palveluun voi ladata rajattomasti omia sovelluksia ja pelejä.

Pelistä tulee tehdä lopullinen julkaisuversio, ennen kun sen voi ladata Google Play -palveluun. Android Studiossa löytyy Build-valikko, jossa on vaihtoehto Generate Signed APK (Kuvio 16.) Apuohjelmalla luodaan salasana ja samalla lopullinen julkaisuversio, jonka voi sitten lähettää Google Play -palveluun.



KUVIO 16. Julkaisuversion luonti

Peliä lisätessä Google Play -palveluun tulee pelille antaa nimi, lyhyt kuvaus ja täysi kuvaus. Nimi on pelin nimi, jolla pelin myös löytää palvelusta. Lyhyt kuvaus on maksimissaan 80 merkkiä pitkä kuvaus pelistä ja pitkä kuvaus on taas maksimissaan 4000 merkkiä pitkä kuvaus. Pelistä tulee myös antaa kuvakaappauksia, korkean resoluution kuvake sekä ominaisuuskuva. Kuvio 14 kuvaa, miltä peli näyttää palvelussa puhelimella katsottuna. Kuvassa ominaisuuskuva on ylimpänä ja sen alla on korkean resoluution kuvake ja pelin nimi. Alempana tulisi myös kuvakaappaukset ja pelin pitkä kuvaus.



KUVIO 17. Näkymä Google Play -palvelussa

6 POHDINTA

Opinnäytetyön teoriaosuudessa tutkittiin, mikä Libgdx on ja miten se toimii, sekä kuinka siitä on hyötyä mobiilipelin kehityksessä. Käytännönosuudessa syntyi Crash Copter -niminen peli Android-alustalle. Pelistä tuli joka puolin toimiva vaikkakin hieman yksinkertaisen näköinen. Siitä huolimatta syntyi sille yllättävän paljon lähdekoodia. Peliin lisättiin lopuksi AdMob-välimainoksia ja se julkaistiin Google Play -palveluun.

Ennen opinnäytetyötä itselläni ei ollut kovin paljon aikaisempaa kokemusta mobiilipeliohjelmoinnista, mutta kiinnostus aiheeseen on ollut kova jo pitkään. Valitsin Libgdx-sovelluskehyksen sen avoimen lisenssin, järjestelmäriippumattomuuden ja nopeasti kasvaneen suosion perusteella. Sovelluskehyksen käytöstä löytyi paljon materiaalia ja alkuun pääseminen oli helppoa. Ongelmia pelinteon aikana oli se, ettei pelillä ollut aluksi selvää suuntaa ja en ole aiemmin tehnyt projektia, jossa on paljon eri olioita vuorovaikutussuhteessa toistensa kanssa. Olioiden keskinäisiä suhteita tuli mietittyä ja tehtyä uusiksi jonkin verran. Kehityksen aikana syntyi paljon erilaisia prototyyppejä, kun kokeilin tehdä asioita paremmin eritavoilla. Grafiikoiden tuotossa oli myös hyvin paljon ongelmia, sillä itselläni ei ole ollenkaan lahjakkuuksia niiden tuottamiseen. Hienompien grafiikoiden avulla pelistä olisi tullut huomattavasti parempi, koska peliin olisi voinut lisätä enemmän sisältöä. Näin jälkeenpäin ajateltuna grafiikat olisi myös kannattanut tehdä ensin vektorimuodossa, jotta niiden skaalaaminen oikeaan kokoon olisi ollut vaivatonta.

Mielestäni työ oli sopivan haastava sekä monipuolinen ja opin paljon uusia asioita mobiilipeliohjelmoinnista, sekä Javan olio-ohjelmoinnista. Uskon, että tulevaisuudessa tulen julkaisemaan lisää pelejä Google Play -palveluun harrastuksen ohella, joissa on käytetty apuna Libgdx-sovelluskehystä.

LÄHTEET

Zechner, M. 2014. libGDX 1.0 released. Saatavissa: http://www.badlogicgames.com/wordpress/?p=3412. Viitattu 3.6.2016.

Zechner, M. 2016. libGDX 1.9.4 released. Saatavissa: http://www.badlogicgames.com/wordpress/?p=3973.Viitattu 3.6.2016.

Reich, M. 2016. Introduction. Saatavissa: https://github.com/libgdx/libgdx/wiki/Introduction. Viitattu 3.6.2016.

Zechner, M. 2010. libgdx changes its license. Saatavissa: http://www.badlogicgames.com/wordpress/?p=777. Viitattu 3.6.2016.

Zechner, M. 2013. Goals and Features. Saatavissa: http://libgdx.badlogicgames.com/features.html. Viitattu 3.6.2016.

Nair, B. & Oehlke, A. 2015. Learning LibGDX Game Development - Second Edition, Packt Publishing.

Chen, S. 2015. Modules overview. Saatavissa: https://github.com/libgdx/libgdx/wiki/Modules-overview. Viitattu 3.6.2016.

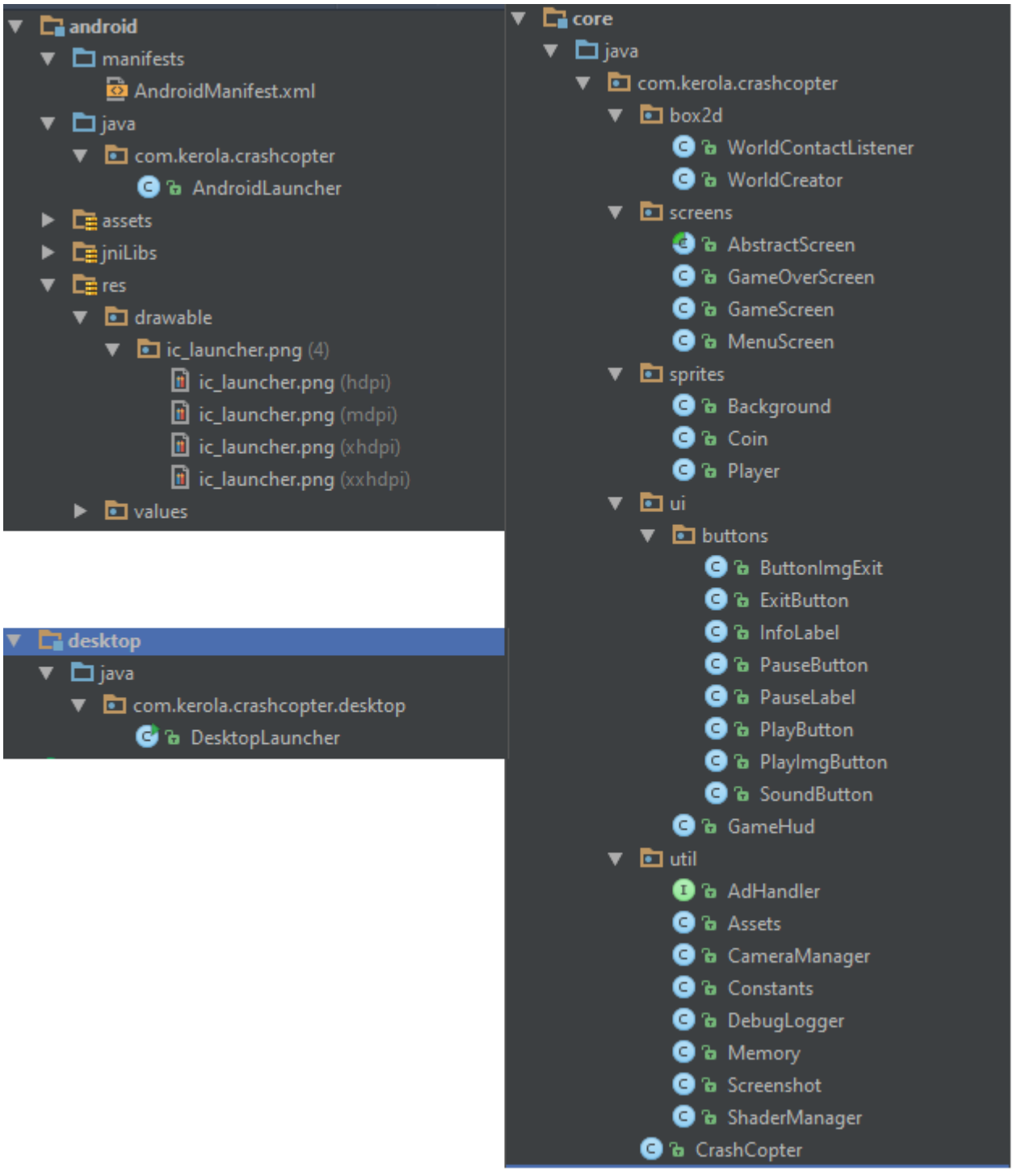
Chen, S. 2015. The life cycle. Saatavissa: https://github.com/libgdx/libgdx/wiki/The-life-cycle. Viitattu 3.6.2016.

Chen, S. 2016. Input handling. Saatavissa: https://github.com/libgdx/libgdx/wiki/Input-handling. Viitattu 3.6.2016.

Kane, R. 2014. Graphics. Saatavissa: https://github.com/libgdx/libgdx/wiki/Graphics. Viitattu 3.6.2016.

Engstler, P. 2014. File module. Saatavissa: https://github.com/libgdx/libgdx/wiki/File-module. Viitattu 3.6.2016.

Silverman, D. 2014. Sound effects. Saatavissa: https://github.com/libgdx/libgdx/wiki/Sound-effects. Viitattu 3.6.2016.


android
manifests
AndroidManifest.xml
java
com.kerola.crashcopter
AndroidLauncher
assets
jniLibs
res
drawable
ic_launcher.png (4)
ic_launcher.png (hdpi)
ic_launcher.png (mdpi)
ic_launcher.png (xhdpi)
ic_launcher.png (xxhdpi)
values
desktop
java
com.kerola.crashcopter.desktop
DesktopLauncher
core
java
com.kerola.crashcopter
box2d
WorldContactListener
WorldCreator
screens
AbstractScreen
GameOverScreen
GameScreen
MenuScreen
sprites
Background
Coin
Player
ui
buttons
ButtonImgExit
ExitButton
InfoLabel
PauseButton
PauseLabel
PlayButton
PlayImgButton
SoundButton
GameHud
util
AdHandler
Assets
CameraManager
Constants
DebugLogger
Memory
Screenshot
ShaderManager
CrashCopter

```
public class PlayImgButton extends ImageButton {

    public PlayImgButton() {
        super(new Image((Texture) Assets.manager()
        .get(Constants.PlayUp)).getDrawable(),
                    new Image((Texture) Assets.manager()
        .get(Constants.PlayDown)).getDrawable());


        setTransform(true);

        addAction(Actions.sequence(
                Actions.scaleTo(0, 0),
                Actions.fadeOut(0),
                Actions.delay(.4f),

                Actions.parallel(
                        Actions.scaleTo(1.0f, 1.0f, 0.3f,
                        Interpolation.linear),
                        Actions.alpha(1.0f, 0.3f))));

    }
}
```

```
@Override
public void showInterstitialAds() {

    if( ! isConnectedToNetwork()) return;

    try {
        runOnUiThread(new Runnable() {
            public void run() {
                if (interstitialAd.isLoaded()) {
                    interstitialAd.show();
                }
                else {
                    AdRequest adRequest =
                            new AdRequest.Builder()
                            .addTestDevice(TESTILAITE_TUNNUS)
                            .build();

                    interstitialAd.loadAd(adRequest);
                }
            }
        });
    } catch (Exception e) {
        Gdx.app.log("Error - showInterstitialAds", e.getMessage());
    }
}



@Override
public boolean isConnectedToNetwork() {
    ConnectivityManager cman = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = cman.getActiveNetworkInfo();
    return (networkInfo != null && networkInfo.isConnected());
}
```