Matti Rantapero

PELIN KEHITYS ANDROID-ALUSTALLE

Tietojenkäsittelyn koulutusohjelma
2017

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

PELIN KEHITYS ANDROID-ALUSTALLE

Rantapero, Matti
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Tammikuu 2017
Ohjaaja: Nuutinen, Petri
Sivumäärä: 35

Asiasanat: android, libgdx, peliohjelmointi

Tämän opinnäytetyön tarkoituksena oli kehittää Android-älypuhelimille tarkoitettu yksinkertainen peli ja sen kautta oppia peliohjelmoinnin perusteita. Peli toteutettiin käyttäen LibGDX-sovelluskehystä.

Työn alussa käsitellään Android-sovelluskehitystä yleisesti sekä käydään läpi LibGDX:n tärkeimpiä ominaisuuksia. Toteutetun pelikehityksen dokumentoinnissa käydään läpi kehityksen eri työvaiheet. Lopuksi tarkastellaan lopputulosta ja pohditaan, miten peliä voisi jatkokehittää.

DEVELOPING A GAME FOR ANDROID

Rantapero, Matti
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Business Information Systems
January 2017
Supervisor: Nuutinen, Petri
Number of pages: 35

Keywords: android, libgdx, game programming

The purpose of this thesis was to develop a simple game for Android smartphones and to learn about the basics of game programming. The game was developed using the LibGDX framework.

The beginning of the thesis examines Android software development in general and the fundamental features of LibGDX. The different stages of the actual game development are reviewed in the fifth main section of the thesis.
.

SISÄLLYS

1 JOHDANTO

Valitsin opinnäytetyöni aiheeksi pelin kehittämisen, koska olen aina ollut kiinnostunut peleistä ja niiden tekemisestä, mutten ollut tehnyt vielä yhtäkään peliä. Ajattelin siis, että saattaisi olla hyvä hetki sille. Android-alustan valitsin, koska pidin siitä ja minulla oli siitä aiempaa kokemusta. Opinnäytetyöni tavoitteena oli luoda yksinkertainen peli Android-alustalle ja samalla oppia peliohjelmoinnin perusteet.

Alun lähteistä haettuun tietoon perustuvassa teoria-osuudessa esittelen pelien kehittämistä yleisesti, Android-alustaa sekä LibGDX-sovelluskehystä. Tämän jälkeen esittelen varsinaisen työni käyttäen hyväksi näytönkaappauksia pelin lähdekoodista ja itse pelistä.

2 PELINKEHITYS YLEISESTI

Pelinkehitys vaatii useita eri taitoja riippuen siitä, kuinka laajasti kehittäjä osallistuu pelintekoon. Jotkut kehittäjät ovat suunnittelijoita, jotka keskittyvät pitkälti pelin konseptin ja ulkoasun suunnitteluun. Toiset kehittäjät taas ovat erikoistuneet ohjelmointiin. Jos kehittäjä haluaa olla monipuolinen pelialan taitaja, tulee hänen osata monia erinäisiä taitoja. Näitä taitoja ovat mm. hyvät suulliset ja kirjalliset kommunikointitaidot, visuaalinen/graafinen suunnittelu, animaatio, vähintään skriptaustason ohjelmointitaidot sekä tietämys laitealustoista, käytettävistä tekniikoista ja pelisuunnittelun teoriasta. Lisäksi pelinkehittäjälle hyödyllisiä ominaisuuksia ovat luovuus ja kiinnostus pelejä kohtaan. (Creative Skillset 2017; Chron 2017.)

Pelejä tulisi suunnitella loppukäyttäjän eli pelaajan näkökulmasta. Huomioitavia asioita ovat mm. haastavuus, hauskuus ja motivaatio. Pelaaja tuntee pelikokemuksen nautinnolliseksi, kun se on sopivan haastava ja hän huomaa kehittyvänsä siinä. Onnistuneesti suunnitellut pelit saattavat saada pelaajan niin sanottuun flow-tilaan, jossa pelaajan keskittyminen kohdistuu täysin hänen pelaamaansa peliin. (Sylvester 2013, 39-40, 63-64.)

Pelien kehittämiseen on saatavilla monia valmiita kehitystyökaluja. Suuret pelialan yritykset luovat yleensä omat työkalunsa, mutta ainakin pienemmät yritykset, aloittelijat ja harrastelijat aloittavat useimmiten valmiilla työkaluilla. Suosituimpia valmiita työkaluja ovat Unity 3D, Cocos2d ja Unreal Engine. Kaikki edellä mainitut työkalut tukevat monia eri kohdealustoja esim. työpöytää, mobiililaitteita ja pelikonsoleita. Unity 3D on sekä 3D- että 2D-peleihin soveltuva pelimoottori, jolla on suuri kehittäjäyhteisö. Sen tukemat kielet ovat C#, JavaScriptiin pohjautuva UnityScript sekä Boo. Cocos2d on 2D-pelimoottori, josta on olemassa eri versioita eri kieliä varten. Sen eri versioiden tukemia kieliä ovat Objective-C, C++, Java, JavaScript ja Ruby. Cocos2d:n C++ -versio tukee suurinta määrää kohdealustoja. Unreal Engine on erittäin suorituskykyinen pelimoottori, jolla on tosin korkea oppimiskynnys. Ainoa sen tukema kieli on C++. (Developer Economics 2014). On olemassa myös pelinkehitystyökaluja, jotka eivät vaadi kovinkaan paljoa ohjelmointitaitoa kehittäjältä ainakaan aluksi. Esimerkki tällaisesta työkalusta on GameMaker. (Kotaku 2013.)

3 ANDROID

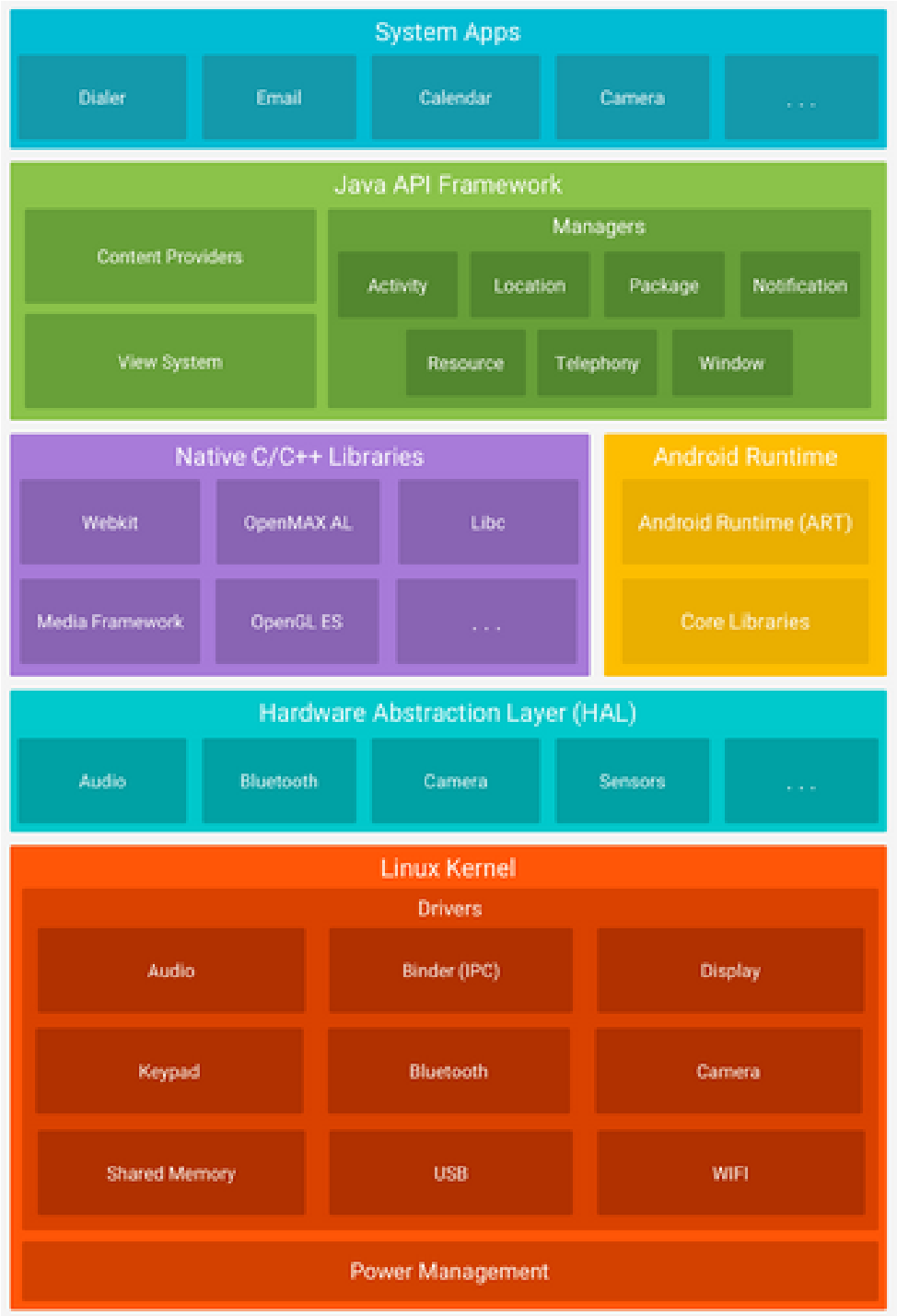
3.1 Yleistä

Android on mobiililaitteille suunnattu avoimen lähdekoodin käyttöjärjestelmä (Gargenta & Nakamura 2014, 1). Sen kehityksestä ja projektin johtamisesta vastaa suurimmalta osin Google (Android Open Source Project 2016). Android tarjoaa monipuolisen sovelluskehyksen sovellusten kehitykseen Java-kielellä (Android Developers 2016).

Android on käytössä monenlaisissa eri laitteissa, kuten älypuhelimissa, tableteissa, televisioissa ja älylaseissa. Se on yleistymässä myös mm. autoteollisuudessa, kodinkoneissa ja robotiikassa. (Jackson 2014, xxvii.) Androidin osuus kaikkien myytyjen älypuhelimien käyttöjärjestelmistä on kasvanut erittäin paljon lähivuosina. Vuonna 2010 sen osuus oli vain 22,7 prosenttia, kun taas vuonna 2015 se oli noussut jo 81,61 prosenttiin. (Statista 2016.)

3.2 Arkkitehtuuri

Android-käyttöjärjestelmä koostuu useista eri tasoista. Tasot eivät aina ole täysin erillään toisistaan, vaikka niillä jokaisella on oma ja selkeä tarkoituksensa. (Gargenta & Nakamura 2014, 31.) Tasot ovat Linux-kerneli (Linux kernel), Natiivi taso (HAL, natiivit C-ja C++-kirjastot, Android Runtime), sovellusten hallintataso (Java API Framework / application framework) ja sovellukset. Tätä ns. ohjelmistopinoa hahmottaa kuva 1. (Android Developers 2016.)



Kuva 1. Androidin ohjelmistopino (Android Developers 2016).

Android on rakennettu Linux-kernelin päälle. On olemassa monta syytä, miksi juuri Linuxia päätettiin käyttää Androidin pohjana. Näitä syitä ovat mm. siirrettävyys, turvallisuus sekä hyödylliset ominaisuudet, kuten muistin- ja virranhallinta sekä verkkotoiminnot. Linux-kerneli on pitkälti vastuussa Android-järjestelmän tietoturvasta esim. jokainen Android-sovellus suorittuu erillisenä prosessinaan kernelin hallinnoimilla oikeuksilla. (Gargenta & Nakamura 2014, 32.)

Hardware abstraction layer (HAL) tarjoaa laitevalmistajille rajapinnan, jota noudattamalla he voivat toteuttaa toimintoja muokkaamatta Android-järjestelmän ylempiä tasoja (Android Open Source Project 2016). Ylemmän tason Java API:t saavat laitteiston (mm. kameran) toimintoja käyttöönsä HAL:in rajapintojen kautta. HAL koostuu useista moduuleista, joista jokainen toteuttaa rajapinnan jollekin tietylle laitteistokomponentille. (Android Developers 2016.)

Android-laitteiden, joiden järjestelmän versio on 5.0 tai uudempi, jokainen sovellus suorittuu omassa Android Runtime (ART)-instanssissaan (Android Developers 2016). ART suorittaa Dex-tavukoodia sekä Dalvik Executable-formaatin koodia. ART kykenee siis suorittamaan useimmat sen edeltäjälle Dalvikille suunnitellut sovellukset. (Android Open Source Project 2016.)

Natiivien C/C++-kirjastojen (libraries) yksi tärkeimmistä tehtävistä on tukea Java API Frameworkia tarjoamalla toiminnallisuutta, joka on Java API:en kautta sovellusten käytettävissä. Natiiveja kirjastoja ovat mm. web-selainmoottori Webkit, SQL-tietokanta SQLite ja OpenGL-grafiikkakirjastot. (Android Developers 2016; Gargenta & Nakamura 2014, 34-35.)

Java API Framework on monille Android-sovelluskehittäjille tutuin Android-arkkitehtuurin taso, koska se tarjoaa kehittäjille Java-kirjastoja, joita koodissaan käyttämällä he saavat käyttöönsä koko Android-käyttöjärjestelmän toiminnallisuuden. Java API Framework abstrahoi alemman tason järjestelmän komponentit, kirjastot ym. yksinkertaistaen niiden käyttöä kehitetyn sovelluksen Java-koodissa. Itse kehitettyjen tai Google Play-sovelluskaupasta ladattujen sovellusten lisäksi Android-järjestelmässä on myös valmiiksi asennettuja sovelluksia, jotka toteuttavat mm. älypuhelinten perustoimintoja, kuten osoitekirjan/puhelut, emailin ja kameran. (Gargenta & Nakamura 2014, 39; Android Developers 2016.)

3.3 Ohjelmistokehityksen aloittaminen

3.3.1 Tarvittavat ohjelmistot

Android-ohjelmistokehityksen aloittamiseen tarvitaan JDK (Java Development Kit) sekä Android SDK. Vaikka pelkillä edellä mainituilla ohjelmistoilla on mahdollista luoda Android-sovelluksia, kehitystä helpottaa Android-kehitykseen tarkoitettu Android Studio-IDE, joka perustuu IntelliJ IDEA-nimiseen IDE:en. IntelliJ IDEA tukee myös Android-kehitystä, joten sitä voi käyttää Android Studion sijaan. (SitePoint 2016.)

3.3.2 Android SDK-työkalut

Android SDK-työkaluja on kahta eri tyyppiä: SDK-työkaluja ja alustatyökaluja (platform tools). SDK-työkalut asennetaan Android SDK:n mukana ja ne päivitetään tietyin aikavälein. Ne ovat Android-sovelluskehityksen kannalta välttämättömiä. Alustatyökalujen tarve taas riippuu kohdealustasta. (Android Developers 2016.)

Tärkeimpiä SDK-työkaluja ovat SDK Manager, AVD Manager, emulaattori ja Dalvik Debug Monitor Server (ddms). SDK Managerin avulla otetaan käyttöön uusia saatavilla olevia työkaluja. AVD Manager tarjoaa graafisen käyttöliittymän Android-virtuaalikoneiden luomiseen ja hallintaan. Luodut virtuaalikoneet suorittuvat SDK-työkaluihin kuuluvassa emulaattorissa, joka tarjoaa sovellusten debuggaukseen ja testaamiseen soveltuvan Android-ajoympäristön (Android run-time environment). (Android Developers 2016.)

3.4 Android-sovelluksen rakenne

3.4.1 Yleistä

Android-sovellukset koostuvat useista komponenteista, joista keskeisimmät ovat activityt, servicet, content providerit ja broadcast receiverit. Muita komponentteja ovat

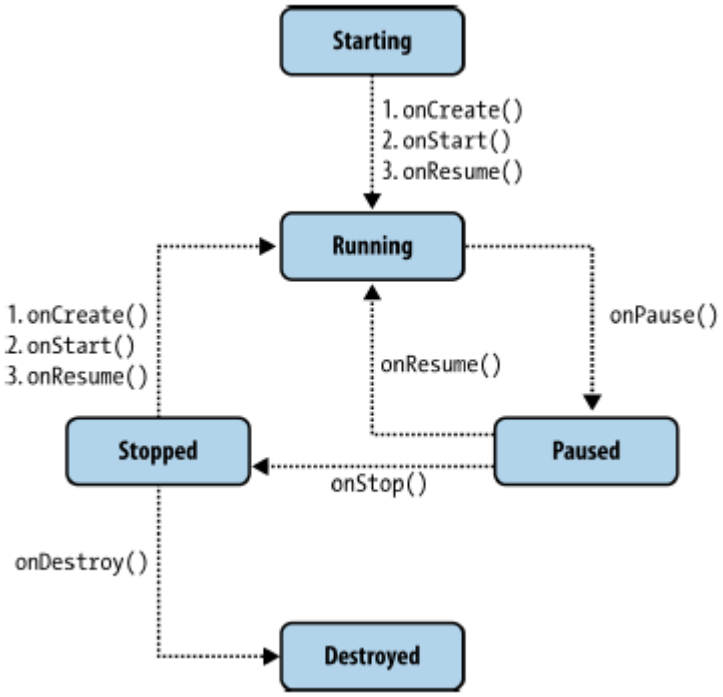
mm. intentit, fragmentit, viewt ja layoutit. Jokaisella komponentilla on selkeä tarkoituksensa ja elinkaarensa. Jotkut komponenteista ovat riippuvaisia toisista komponenteista. (Android Developers 2016; Tutorialspoint 2016.)

Kaikkia komponentteja varten tulee luoda AndroidManifest.xml-tiedostoon oma elementtinsä, jossa määritellään kyseisen komponentin koodissa toteuttava luokka. Lisäksi määritellään intentit, joita komponentti pystyy käsittelemään. (Android Developers 2016.)

3.4.2 Activity

Android-sovelluksessa on yksi tai useampi activity-komponentti. Activityt ovat ikään kuin sovelluksen eri näkymiä tai sivuja, joiden kautta käyttäjä vuorovaikuttaa sovelluksen kanssa esim. syöttää tietoja, painaa painikkeita tai tarkastelee sovelluksen esittämiä tietoja. Activity-komponentti luo ikkunan, johon kehittäjä määrittelee sille UI:n View-komponentin muodossa metodilla setContentView. On suositeltavaa luoda yksi aktiviteetti kutakin tehtävää/toimintoa varten. Jokaisessa Android-sovelluksessa on yksi pää-activity (main activity), joka otetaan käyttöön, kun sovellus käynnistetään eli se on sovelluksen aloitusikkuna. (Aftab & Karim 2014, 24-25; Android Developers 2016.)

Activity-komponentilla, kuten muillakin komponenteilla, on tarkkaan määritelty elinkaari, joka koostuu useasta eri tilasta (kuva 2). Activity sisältää monta eri callback-metodia, yksi jokaista activityn elinkaaren vaihetta varten. Järjestelmä kutsuu tiettyä callback-metodia activityn siirtyessä uuteen tilaan. Callback-metodien koodissa määritellään, miten tilasta toiseen siirtymiseen reagoidaan. (Android Developers 2016.)



Kuva 2. Activityn elinkaari (Gargenta & Nakamura 2014, 65).

3.4.3 Intent

Intentit toimivat viesteinä komponenttien, kuten aktiviteettien, välillä. Ne voivat mm. käskeä aktiviteettiä käynnistymään ja käskeä serviceä käynnistymään tai pysähtymään. Intenttejä on sekä implisiittisiä että eksplisiittisiä. Eksplisiittisten intenttien tapauksessa lähettävä komponentti määrittää selvästi, minkä komponentin tulisi se vastaanottaa. Implisiittisen intentin tapauksessa lähettäjä määrittää vain vastaanottajan tyypin, jolloin mikä tahansa määrittelyä vastaava komponentti/sovellus voi vastaanottaa ko. intentin. (Gargenta & Nakamura 2014, 68.)

3.4.4 Service

Service-komponentit suorittuvat taustalla, eikä niillä ole minkäänlaista graafista käyttöliittymää, mutta ne pystyvät toteuttamaan samoja toimintoja kuin aktiviteetit. Ne ovat siis hyödyllisiä silloin, kun halutaan suorittaa jotakin prosessia taustalla riippumatta siitä, mitä on näkyvissä näytöllä. (Gargenta & Nakamura 2014, 68.) Jokin toinen komponentti voi käynnistää servicen ja myös sitoa itsensä siihen ja olla sen kanssa

vuorovaikutuksessa. Service voi esimerkiksi suorittaa tiedosto-I/O:ta, soittaa musiikkia tai olla vuorovaikutuksessa content provideriin. Toinen komponentti voi käynnistää servicen kutsumalla joko startService- tai bindService-metodia. startService-metodilla käynnistetty service on käynnissä kunnes se kutsuu omaa stopSelf-metodiaan tai toinen komponentti pysäyttää sen stopService-metodilla. Järjestelmä poistaa pysäytetyn servicen. bindService-metodilla käynnistettyyn serviceen voi olla yhteydessä monta eri komponenttia samanaikaisesti, jolloin komponentit ovat yhteydessä serviceen IBinder-rajapinnan kautta. Tällainen service poistetaan muistista, kun siihen ei ole yhteydessä enää yhtäkään toista komponenttia (clienttiä). (Android Developers 2016.)

3.4.5 Content provider

Content provider hallinnoi jaettua dataa, jonka sovellus on tallentanut tiedostojärjestelmään, SQLite-tietokantaan, webbiin tai muuhun pysyvään sijaintiin. Content providerin voi asettaa sallimaan turvallisesti muiden sovellusten pääsy sen oman sovelluksen dataan. Se on myös standardi rajapinta prosessissa olevan datan yhdistämisessä toisen prosessin koodiin. (Android Developers 2016.)

3.4.6 Broadcast receiver

Broadcast receiver mahdollistaa sovelluksessa järjestelmätason tapahtumiin reagoimisen. Järjestelmän itsensä käynnistämät broadcast-tapahtumat (ilmoitukset) voivat liittyä mm. näytön sammumiseen ja akkuvirran vähissä olemiseen. Sovellukset voivat myös itse käynnistää broadcast-tapahtumia, joihin muut sovellukset voivat reagoida. Vastaanottavat sovellukset määrittävät erilaisten filtterien kautta minkälaisiin tapahtumiin niissä reagoidaan. Broadcast receivereillä ei ole minkäänlaista visuaalista esitystä, vaan ne suorittuvat taustalla aktivoiduttuaan. Broadcasteja lähetetään muiden sovellusten saataville intenttien muodossa. (Android Developers 2016; Gargenta & Nakamura 2014, 72, 223.)

4 LIBGDX

4.1 Yleistä

LibGDX on avoimen lähdekoodin sovelluskehys, joka on lähinnä tarkoitettu pelien kehittämiseen Java-kielellä. Java-kielen lisäksi LibGDX hyödyntää C-kielellä luotuja kirjastojaan joidenkin suorituskyvyn kannalta kriittisten tehtävien suorittamisessa ja muiden C-kirjastojen liittämisessä itseensä. Nämä C-kirjastot mahdollistavat myös cross-platform toiminnallisuuden. Kehittäminen monelle eri alustalle samalla koodilla (cross-platform) on mahdollista myös, koska LibGDX abstrahoi alustojen erot API:n (Application Programming Interface) taakse. (Nair & Oehlke 2015, 10.)

LibGDX ei ole pelimoottori, johon kuuluu yleensä monia työkaluja mm. kenttäeditori (level editor) ja jokin työkalu, joka ennalta määrittää työnkulun. LibGDX on sen sijaan sovelluskehys ja vaatii enemmän suunnittelua kehittäjältä itseltään, mutta antaa tälle myös enemmän vapauksia. Kehittäjä määrittää itse oman työkulkunsa ja pystyy itse säätämään matalammalla tasolla (low-level) mm. OpenGL-kutsuja.
(Nair & Oehlke 2015, 10.)

LibGDX hyödyntää toteutuksessaan myös monia third-party -kirjastoja. Näitä kirjastoja ovat mm:

- OpenGL (Grafiikan renderöinti)
- FreeType (Fonttien renderöinti)
- mpg123 (MPEG-soitin- ja dekooderi)
- xiph.org (Pakattu audio-formaatti)
- Soundtouch (Tempon ym. muuttaminen audio-jonoissa- ja tiedostoissa)
- Box2D (2D-fysiikkamoottori)
- LWJGL (Java-kirjasto pelinkehitystä varten)
- OpenAL (Audio API)
- KissFFT.

(Badlogic Games 2016.)

LibGDX:n tukemat alustat ovat tällä hetkellä Windows, Linux, Mac OS X, Android, Blackberry, iOS ja HTML5. (Badlogic Games 2016.)

4.2 Asennus

LibGDX käyttää Gradle-nimistä ns. build automation -ohjelmistoa riippuvuuksien hallinnoimiseen, koontiprosessin (build process) suorittamiseen sekä IDE-integraatioon. LibGDX-sovelluskehyksen lisäksi tarvitaan muutamia oheisohjelmistoja sovelluskehityksen aloittamiseksi. (Badlogic Games 2016.)

Windowsille, Linuxille, Androidille ja HTML5:lle kehittämiseen tarvitaan seuraavat ohjelmistot:

- Java Development Kit 7+ (JDK)
- Jokin seuraavista Java-IDE:istä: IntelliJ IDEA / Android Studio, Eclipse tai NetBeans
- Android SDK
- Mahdolliset pluginit IDE:stä riippuen:
    - Eclipselle ADT-plugin
    - Eclipse Integration Gradle
    - NetBeansille NBAndroid-pluginit
    - NetBeansille ”Gradle Support For Netbeans”.

iOS:lle tarvitaan puolestaan seuraavat ohjelmistot / laitteet:

- Mac-tietokone, johon on asennettu Mac OS X ja Xcode 7+
- RoboVM-plugin.

(Badlogic Games 2016.)

LibGDX-sovelluskehyksen mukana tulee ”gdx-setup.jar”-niminen tiedosto, joka on sekä graafisen käyttöliittymän omaava ohjelma että komentorivityökalu. Työkalun saa käyttöön suorittamalla edellä mainitun JAR-tiedoston. Työkalu pyytää määrittämään sovelluksen nimen, Java-paketin nimen, sovelluksen pääluokan, projektille halutun tiedostosijainnin sekä Android SDK:n tiedostosijainnin. Lisäksi tulee vielä määrittää

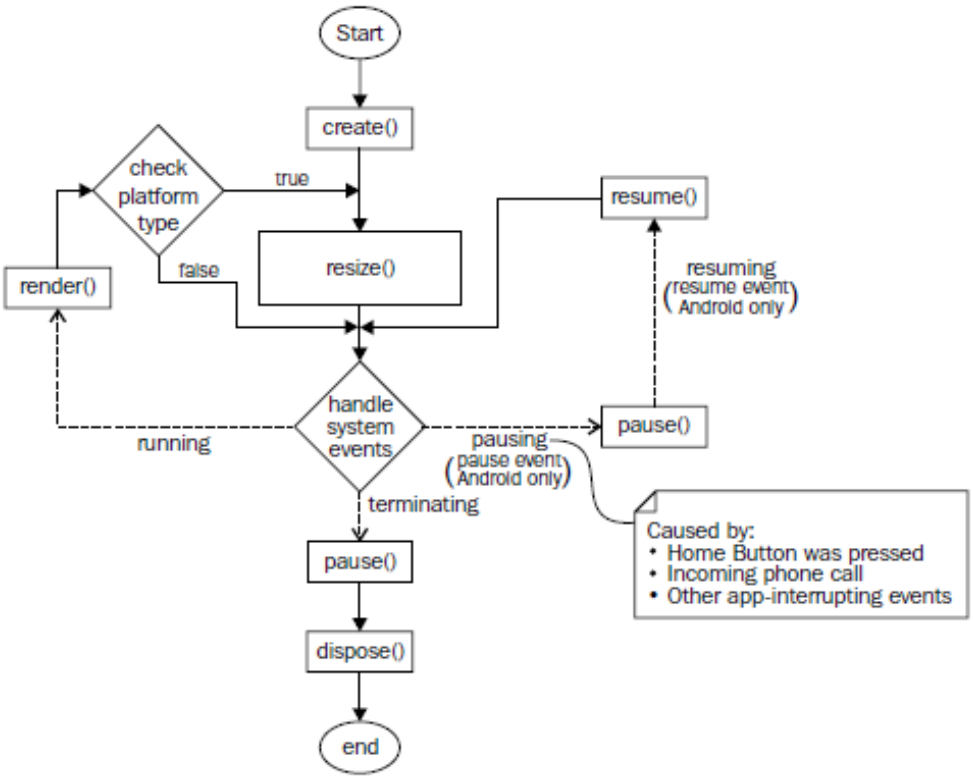
mille alustoille aiotaan kehittää ja mitä laajennuksia aiotaan käyttää. Kuvassa 3 näkyy kyseinen työkalu syötetyillä esimerkkitiedoilla. (Badlogic Games 2016.)

Kuva 3. LibGDX:n setup-työkalu (Badlogic Games 2016).

4.3 Sovelluskehyksen rakenne

4.3.1 Elinkaari

LibGDX-sovelluksilla on tarkkaan määritelty elinkaari (kuva 4), johon kuuluvat seuraavat tilat (states): create, resize, render, pause, resume ja dispose. LibGDX-sovellus alkaa aina create-metodista, jossa ladataan tarvittavat resurssit (kuvatiedostot ym.) muistiin ja luodaan pelimaailman alkutila. Resize-metodia kutsutaan aina create-metodin jälkeen ja myös kun pelinäkymän (screen) kokoa muutetaan. Se vastaanottaa parametreinaan näkymän korkeuden ja leveyden pikseleinä. Sovelluksen pelisilmukka (game loop) kutsuu render-metodia aina, kun pelimaailman tila tulisi päivittää ja näyttö renderöidä. Pause-metodia kutsutaan Android-alustalla käyttäjän painaessa home-painiketta ja työpöytäsovelluksissa juuri ennen dispose-metodin kutsumista sovellusta suljettaessa. Siihen on siis suositeltavaa sijoittaa pelitilan tallennus. Resume-metodia kutsutaan Android-alustalla sovelluksen jatkuessa pause-tilasta. Sovelluksen suoritus päättyy dispose-metodiin, jota kutsutaan sovellusta suljettaessa. Se vapauttaa kaikki sovelluksen käyttämät resurssit muistista. (Badlogic Games 2016.)



Kuva 4. LibGDX-sovelluksen elinkaari (Nair & Oehlke 2015, 75).

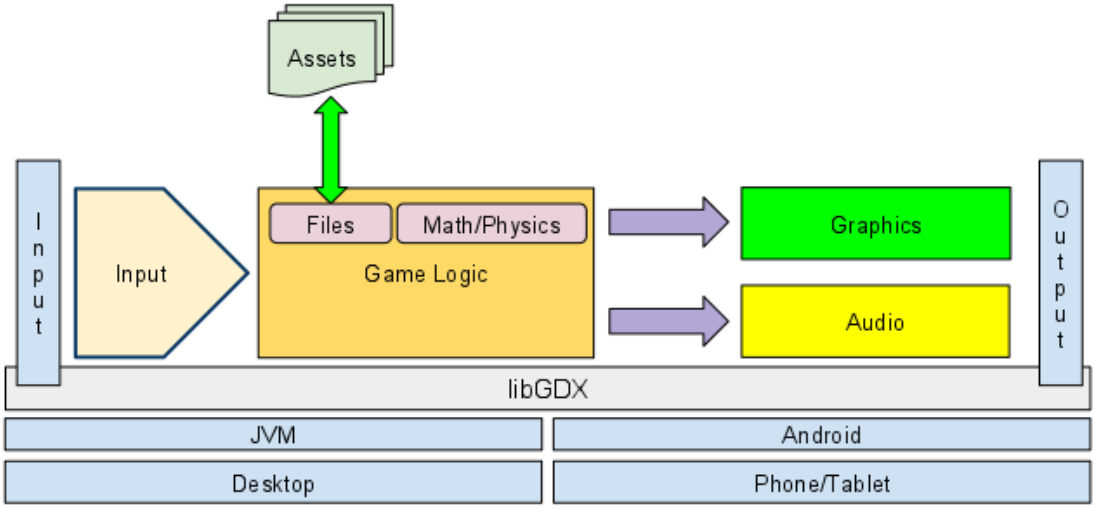
LibGDX-sovellus on pitkälti tapahtumapohjainen, eikä sisällä varsinaista pääsilmukkaa (main loop). Lähimpänä tällaista silmukkaa on kuitenkin ApplicationListener.render() -metodi. ApplicationListener-toteutuksessa (kuva 5) kehittäjä määrittelee, mitä sovellus tekee sen kussakin elinkaaren vaiheessa. (Badlogic Games 2016.)

```
public class MyGame implements ApplicationListener {
    public void create () {
    }

    public void render () {
    }

    public void resize (int width, int height) {
    }

    public void pause () {
    }

    public void resume () {
    }

    public void dispose () {
    }
}
```

Kuva 5. ApplicationListener-rajapinnan toteuttava luokka (Badlogic Games 2016).

4.3.2 Moduulit

Pelinkehitykseen tarvittavat ominaisuudet on LibGDX:ssä jaettu useaan eri moduuliin (kuva 6), joihin on pääsy Gdx-luokan kenttien kautta eli jokaista moduulia kohden on Gdx-luokassa oma kenttänsä. Nämä moduulit ovat app, input, graphics, files, audio ja networking. (Nair & Oehlke 2015, 67-73.)



Kuva 6. LibGDX:n moduulit (Badlogic Games 2016).

App-moduuli eli application-moduuli tarjoaa pääsyn log-toimintoihin, metodin sovelluksen hallittuun sulkemiseen sekä mahdollisuuden tallentaa dataa preferences-tiedostoon. Lisäksi se tarjoaa mahdollisuuden kysellä tiedon Android API-versiosta, alustatyypistä ja muistinkäytöstä. (Nair & Oehlke 2015, 68.)

Input-moduuli mahdollistaa input-tilojen kyselyn (polling) jokaisella LibGDX:n tukemalla alustalla. Sen avulla saadaan selville näppäinten, hiiren ja kosketusnäytön tila. Polling suoritetaan jaksoittaisesti tietyin aikavälein. Input-moduulia pääsee koodissa käyttämään Gdx.input-kentän kautta. (Badlogic Games 2016.) Kuvassa 7 olevassa esimerkissä input-kentän kautta selvitetään kosketusnäytön tila eli kosketaanko näyttöä kyselyn suoritushetkellä, jonka jälkeen tulos tallennetaan isTouched-nimiseen boolean-tyyppiseen muuttujaan.

```
boolean isTouched = Gdx.input.isTouched();
```

Kuva 7. Esimerkki Input-moduulin käytöstä GDX.input-kentän kautta (Badlogic Games 2016).

Jatkuvan tietyin aikavälein tapahtuvan kyselyn sijasta voidaan myös määrittää erillinen tapahtumankäsittelijä (listener), joka käsittelee input-tapahtumat kronologisessa järjestyksessä. Tapahtumankäsittelijän määritteleminen on suositeltavaa, jos pelin kontrollit ovat monimutkaiset. Sen määritteleminen tapahtuu käytännössä antamalla

Input-rajapinnan setInputProcessor-metodille parametriksi halutun tapahtumankäsittelijän, jonka tulee olla InputProcessor-rajapinnan toteuttavan luokan olio (kuva 8). Oman custom-tapahtumankäsittelijän pystyy siis luomaan tekemällä oman luokan, joka toteuttaa InputProcessor-rajapinnan tai vaihtoehtoisesti periytyy InputAdapter-luokasta. Edellä mainittu InputAdapter-luokka toteuttaa kaikki InputProcessor-rajapinnan vaatimat metodit, jolloin kehittäjän tarvitsee omassa periytetyssä luokassaan määrittää toteutukset vain tarvitsemilleen metodeille. (Badlogic Games 2016.)

```
MyInputProcessor inputProcessor = new MyInputProcessor();
Gdx.input.setInputProcessor(inputProcessor);
```

Kuva 8. InputProcessor-tapahtumankäsittelijän määritteleminen (Badlogic Games 2016).

Graphics-moduuli tarjoaa tietoa näytön tilasta, sovelluksen ikkunasta ja senhetkisestä OpenGL-kontekstista. Tähän tietoon kuuluu mm. näytön koko, pikselitiheys (pixel density) sekä frame-buffer-ominaisuudet kuten värisyvyys (color depth), syvyyspuskurit (depth buffers) ja anti-aliasing eli reunojen pehmentäminen. (Badlogic Games 2014.)

Files-moduuli tarjoaa alustariippumattoman keinon kirjoittaa tiedostoihin ja lukea niistä. Se sisältää seuraavat toiminnot:

- Tiedostosta lukeminen
- Tiedostoon kirjoittaminen
- Tiedoston kopioiminen
- Tiedoston poistaminen
- Tiedostojen ja hakemistojen listaaminen
- Tiedoston/hakemiston olemassaolon testaaminen

(Badlogic Games 2016.)

Yleisin käyttötarkoitus files-moduulille on assettien (tekstuurit, äänitiedostot ym.) lataaminen samasta alihakemistosta kaikkien kohteena olevien alustojen käyttöön. Se on myös hyödyksi pelitietojen (pelitilat, huipputulokset ym.) tallennukseen. (Badlogic Games 2016.)

LibGDX:ään on määritellyt viisi eri tiedostotyyppiä ovat classpath, internal, local, external ja absolute. Classpath-tiedostot tallennetaan suoraan lähdekoodin tiedostoihin, pakataan jar-tiedostoihin ja ovat aina "read-only". Muiden kuin classpath-tiedostojen käyttö on useimmissa tapauksissa suositeltavaa. Internal-tiedostot ovat read-only ja sijaitsevat PC:llä sovelluksen juurihakemistossa ja Androidilla sovelluksen assets-hakemistossa. Jos haettua tiedostoa ei löydy internal-hakemistoista, siirtyy files-moduuli etsimään sitä classpath-hakemistoista. Internal-tiedostoihin tulisi peleissä tallentaa assetit. Local-tiedostot tallennetaan sovelluksen juurihakemistoon ja Androidilla sovelluksen internal-hakemistoon, johon muilla sovelluksilla ei ole oikeuksia. Niitä voi käyttää pienien tiedostojen, kuten pelitilojen tallentamiseen. External-tiedostot tallennetaan Androidilla SD-kortille ja PC:llä senhetkisen käyttäjän kotihakemistoon. Absolute-tiedostojen tallentamiseksi pitää määritellä koko tiedostopolku, minkä takia niitä käyttävä sovellus on vaikeammin siirrettävä. (Badlogic Games 2016.)

4.3.3 Konfigurointi

Jokaiselle eri LibGDX:n tukemalle alustalle on oma aloitusluokkansa, josta sovelluksen suoritus alkaa. Jokainen aloitusluokka luo alustalleen sopivan Application-rajapinnan toteutuksen instanssin sekä sovelluslogiikan toteuttavan ApplicationListenerin. (Badlogic Games 2016.)

Android-sovellukset vaativat ainakin yhden Activity-luokan, josta sovelluksen suoritus alkaa. LibGDX-setup-työkalu luo automaattisesti kyseisen activityn (kuva 9). Tämän activityn suoritus alkaa varsinaisesti sen sisältämästä onCreate-metodista. Sen suorittamisen jälkeen luodaan uusi AndroidApplicationConfiguration-instanssi, jonka jälkeen kutsutaan AndroidApplication.initialize()-metodia, jolle annetaan parametreina ApplicationListener ja luotu konfiguraatio. (Badlogic Games 2016.)

```
package com.me.mygdxgame;

import android.os.Bundle;

import com.badlogic.gdx.backends.android.AndroidApplication;
import com.badlogic.gdx.backends.android.AndroidApplicationConfiguration;

public class MainActivity extends AndroidApplication {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        AndroidApplicationConfiguration cfg = new AndroidApplicationConfiguration();

        initialize(new MyGdxGame(), cfg);
    }
}
```

Kuva 9. LibGDX-sovelluksen MainActivity-luokka (Badlogic Games 2016).

Android-sovelluksissa voi olla monta activitya, mutta LibGDX-sovellusten tulisi yleensä sisältää niitä vain yksi. Erilaisia näkymiä sovellukseen ei luoda uusien activityjen muodossa. Tämä johtuu siitä, että uusi activity vaatii uutta OpenGL-kontekstia, mikä on aikaa vievää ja tarkoittaa sitä, että kaikki graafiset resurssit pitää ladata uudelleen. (Badlogic Games 2016.)

Kuten muutkin Android-sovellukset, Androidille suunnattu LibGDX-sovellus vaatii AndroidManifest.xml-tiedoston, jonka application-elementin activity-lapsielementteihin tulee määrittää kaikki sovelluksen activityt, joita LibGDX-sovelluksessa tulisi siis olla vain yksi. (Badlogic Games 2016.)

5 TOTEUTETUN PELIKEHITTÄMISEN VAIHEET

5.1 Peli-idea

Peli-idean kehittelyssä oli tärkeää ottaa huomioon alustasta johtuvat rajoitteet, kuten näppäinten puuttuminen ja pieni näyttö. Älypuhelinten alhainen suorituskyky suh-

teessa esim. pöytäkoneisiin ei ollut ongelma, koska peli ei tulisi vaatimaan paljon suorituskykyä laitteelta. Tämä johtuu siitä, että peli tulisi olemaan yksinkertainen niin graafisesti kuin muutenkin. Myös kokemattomuus peliohjelmoinnissa sai päätymään varsin yksinkertaiseen peli-ideaan. Tässä työssä keskitytään enemmän pelin koodiin kuin sen grafiikkaan, joten peliobjektien ulkoasu toteutetaan yksinkertaisina geometrisina kuvioina. Selkeät värierot näytöllä näkyvien objektien välillä ovat siis tärkeitä. Pelin kontrolleina toimivat sormen liu'uttaminen (drag) kosketusnäytöllä sekä valikkojen kosketuskontrollit.

Pelin ideana on, että ruudun ylälaidasta putoaa peliobjekteja, jotka ovat joko "pisteitä" tai "vihollisia". Pisteet ovat ulkomuodoltaan ympyröitä ja viholliset kolmioita. Pelaaja yrittää kerätä mahdollisimman suuren pistemäärän liu'uttamalla sormella omaa pelihahmoaan vapaasti ruudulla niin, että osuu mahdollisimman moneen pisteeseen samalla väistellen pistemäärää vähentäviä vihollisia. Vihollisia on ruudulla vähemmän kuin pisteitä, mutta ne laskevat pistemäärää suhteessa enemmän kuin pisteet nostavat sitä. Ruudun ylälaidassa näkyy jäljellä oleva aika sekä senhetkinen pistemäärä.

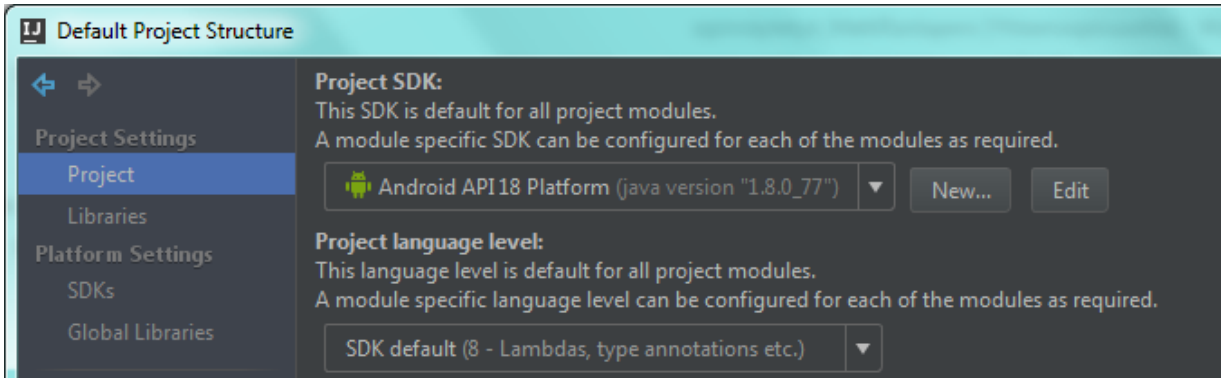
Pelin olisi tarkoitus sisältää seuraavat näkymät: menu (valikko), game (itse peli), score (tulos) ja mahdollinen pause.

5.2 Kehitysympäristön valmistelu

IDE:jen, kielien, sovelluskehysten ym. valikoima pelinkehitystä varten on varsin kattava. IDE:t ja sovelluskehykset ovat kielikohtaisia, niinpä haluttu kieli saattaa vaikuttaa merkittävästi valintaan niiden välillä. Haluttiin koodata Java-kielellä, jolle ei ole tarjolla kovin montaa eri vaihtoehtoa Android-pelinkehitykseen sovelluskehysten osalta. Yksi varteenotettava vaihtoehto oli kuitenkin LibGDX. Toinen vaihtoehto olisi ollut Unity 3D, joka ei tosin ole Java-sovelluskehys, vaan kokonainen kehitysympäristö mukana tulevalla IDE:llä. Unityä käyttäen olisi voinut koodata joko C#:lla tai JavaScriptilla, mikä ei sinänsä olisi ollut este. Unity 3D on myös cross-platform -kehys niin kuin LibGDX:kin. Päädyttiin LibGDX:ään, koska siinä kehittäminen tapahtuu puhtaammin koodin kautta, eikä graafisilla työkaluilla ole niin suurta osaa, paitsi tietenkin tekstuurien ym. tekemisessä. Java-IDE:jä on monia, kuten Eclipse, NetBeans ja

IntelliJ IDEA. Valittu sovelluskehys LibGDX olisi tukenut kaikkia edellä mainituista, mutta päädyttiin IntelliJ:hin, koska se käyttää LibGDX:n vaatimaa Gradle-ohjelmistoa projekteissaan ilman tarvittavia plugineja. Eclipse ja NetBeans olisivat vaatineet erillisen pluginin.

Päätösten jälkeen oli aika ladata ja asentaa tarvittavat työkalut. Ensimmäisenä ladattiin IntelliJ:n ilmainen Community Edition osoitteesta https://www.jetbrains.com/idea/. Kun se oli asennettu tarkistettiin, että kehitystyössä käytettävässä tietokoneessa oli asennettuna riittävän uudet versiot Java JDK:sta ja Android SDK:sta. Tämän jälkeen määritettiin mitä Android SDK:n ja Javan versiota IDE käyttäisi oletusarvoisesti (kuva 10). Android SDK-version pääsee määrittämään IntelliJ:ssä alkuvalikossa olevasta configure-valikosta, josta valitaan ”Project Defaults” ja siitä ”Project Structure”.



Kuva 10. Käytettävän Android SDK:n version määrittäminen IntelliJ IDE:ssä.

Sitten ladattiin LibGDX-setup-ohjelma (kuva 11). Projektin ja pelin pääluokan nimeksi annettiin OT ja Java-paketin nimeksi com.mattirantapero.ot noudattaen yleisiä Java-pakettien nimeämiskäytäntöjä. Kohdealustaksi valittiin Androidin lisäksi myös työpöytä siksi, että peliä voisi mahdollisesti olla nopeampaa testata. Laajennuksista otettiin käyttöön Box2d-fysiikkamoottori, siltä varalta, että haluttaisiin lisätä peliin fysiikan mallinnusta kohdistuen ylhäältä putoaviin peliobjekteihin.

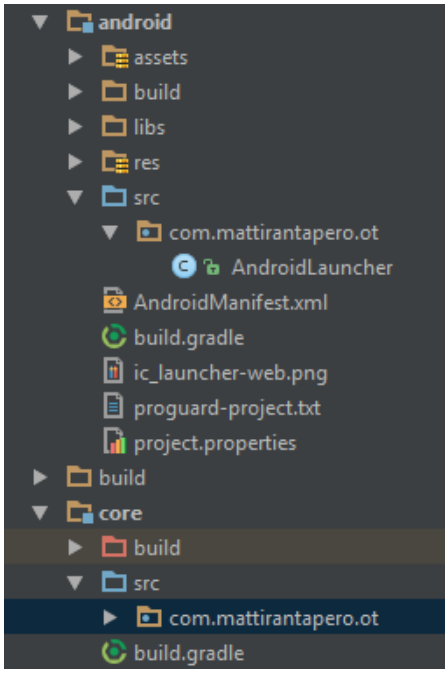
Kuva 11. LibGDX-projektin luominen setup-ohjelmalla.

Projektin luomisen jälkeen avattiin ko. projekti IDE:ssä. LibGDX-setupin luoma projekti sisältää valmiiksi aloitusluokat omina aliprojekteinaan oleville eri kohdealustoille. Android-aliprojekti sisältää lisäksi AndroidManifest.xml-tiedoston, jossa oikea aloitusluokka on valmiiksi määritelty activityksi. Pelin ApplicationListener-rajapinnan toteuttava pääluokka ei myöskään ole tyhjä, vaan se sisältää koodin, joka saa sovelluskehyksen mukana tulleen kuvan näkymään näytöllä sovellusta suoritettaessa. Se on tavallaan testi, joka kertoo, onko projektin luominen ja sen konfiguraatio sujunut virheettömästi.

Testin jälkeen luotiin Android-virtuaalikone AVD-managerilla, mutta huomattiin sen toimivan hitaasti käytetyllä tietokoneella ja vievän liikaa laiteresursseja (prosessorin kuormitus ja keskusmuisti). Päätettiin siis, että testattaisiin jatkossa sovellusta virtuaalikoneen sijaan Android-älypuhelimella. Asennettiin siis tietokoneeseen tarvittavat ajurit, jotta voitaisiin debugata ADB:n avulla usb-kaapelin välityksellä puhelimessa.

5.3 Ohjelmointi

Varsinainen sovelluslogiikan sisältävä koodi LibGDX-projektissa sijoitetaan core-aliprojektin src-kansioon (kuva 12).



Kuva 12. Osa ns. project treestä.

5.3.1 Game- ja Screen-luokat

Päätettiin periyttää pelin pääluokka LibGDX:n ApplicationListener-rajapinnan toteuttavasta Game-luokasta. Game-luokalla saa toteutettua hyvin pelin eri näkymät ja niiden välillä siirtymiset. Tämä johtuu siitä, että Game-luokka delegoi pelin pääsilmukkana (main loop) toimivan render-metodin suorituksen Screen-rajapinnan toteuttaville luokille. Eri näkymiä saa siis luotua tekemällä näitä Screen-luokkia ja siirtymällä nii-

hin halutessaan kutsumalla Game-luokan setScreen-metodia. Screen-luokkia ei poisteta automaattisesti muistista, joten uuteen Screen-luokkaan siirryttäessä tulisi poistaa edellinen näkymä sen dispose-metodilla.

Pelin niin sanottu pääluokka on Game-luokasta periytetty OT (”OpinnäyteTyö”), joka sisältää create, render ja dispose-metodien lisäksi vakioina (static final-kentät) pelin virtuaalisen leveyden ja korkeuden sekä SpriteBatch-tyyppisen kentän. Edellä mainittuihin on pääsy kaikilla Screen-luokilla, koska setScreen-metodille annetaan sitä kutsuttaessa parametriksi Game-instanssi, joka sijoitetaan Screen-luokassa erilliseen kenttään.

Peli sisältää kolme eri näkymää: MenuScreenin, PlayScreenin ja ScoreScreenin. MenuScreen on hyvin yksinkertainen. Siinä pelaaja koskettaa näyttöä aloittaakseen pelin, jolloin siirrytään MenuScreenistä PlayScreeniin. PlayScreenistä siirrytään ScoreScreeniin peliajan päättyessä. Jokaisen Screen-luokan konstruktorissa luodaan uusi ortografinen kamera (OrtographicCamera) ja viewport (Viewport), joille annetaan parametreina pelin virtuaalinen leveys ja korkeus, jotka saadaan siis Game-instanssin vakioista.

PlayScreenissä tapahtuu varsinainen peli. Siinä on kenttinä eri peliobjektien kokoelmia, tekstuureita ja niiden leveydet ja korkeudet vakioina. Sen konstruktorissa luodaan monia pelin aikana tarvittavia olioita. PlayScreenin render-metodissa puolestaan piirretään näytölle kaikki tarvittava ja kutsutaan sitä ennen update-metodia, jossa suoritetaan erinäisten peliobjektien tilojen päivittäminen ja tarkistetaan osuvatko ne toisiinsa.

5.3.2 Peliobjektit ja niiden toiminta

Peliobjektit pelissä ovat joko yhden kerran yksittäisen pelin aikana luotavia tai jatkuvasti pelin aikana luotavia ja tuhottavia. Jatkuvasti pelin aikana luotavia objekteja on useampi kuin yksi ja, koska ne olisivat olleet sisällöltään hyvin samanlaisia, päätettiin luoda niitä varten abstrakti luokka koodiduplikaattien välttämiseksi. Yhden kerran luotavia peliobjekteja on tässä vaiheessa pelin kehitystä ainoastaan yksi, joka on PlayerEntity-luokka ja sen olio on pelaajan liikuttama hahmo. Ajateltiin kuitenkin, että

peliin voitaisiin lisätä sujuvammin uusia tämän kaltaisia peliobjekteja, jos luotaisiin yhden kerran luotaville objekteillekin oma abstrakti luokkansa.

Yhden kerran luotavia peliobjekteja varten tehtiin Entity-niminen abstrakti luokka, joka sisältää kaikille yhden kerran luotaville peliobjekteille yhteisiä kenttiä ja metodeita. Sen sisältämät kentät ovat Vector2-tyyppinen position (sijainti näytöllä) sekä Rectangle-tyyppinen bounds (rajat). Metodeita ovat mm. konstruktori sekä objektin sijainnin määritykseen tarkoitettu setPosition.

Jatkuvasti luotavia objekteja varten luotiin PoolableEntity-niminen abstrakti luokka, joka toteuttaa LibGDX:n Pool.Poolable-rajapinnan. Sitä käytettiin apuna object pool-suunnittelumallin toteuttamiseksi. PoolableEntityn sisältämiä kenttiä ovat mm. Entitynkin sisältämien positionin ja boundsin lisäksi boolean-tyyppinen visible, jonka arvon tulee kertoa se, että onko kyseinen olio liikkunut näytön reunusten ulkopuolelle. Sen sisältämiä metodeja ovat mm. abstrakti init, jota kutsutaan, kun pool ottaa kyseisen olion käyttöön, sekä reset, jota kutsutaan, kun pool ottaa kyseisen olion pois käytöstä. Tällaisia PoolableEntity-luokasta periytettäviä luokkia ovat PositiveEntity ja NegativeEntity. PositiveEntityn olioon osuessaan pelaaja saa pisteitä ja NegativeEntityn olioon osuessaan pelaaja puolestaan menettää pisteitä.

Varsinaisen pelin alkaessa PlayScreenin konstruktorissa ladataan peliobjekteja varten muistiin kaikki tarvittavat resurssit (lähinnä tekstuurit) sekä luodaan PlayerEntityn instanssi pelaajan hahmoksi. Lisäksi luodaan ArrayList-kokoelmat PositiveEntityn ja NegativeEntityn instansseja varten sekä erillinen pool kummankin tyyppisille instansseille. PlayScreenin render-metodissa kutsuttavassa update-metodissa suoritetaan uusien jatkuvasti luotavien olioiden luominen, käyttöön ottaminen ja käytöstä poistaminen. Uusia oliota otetaan käyttöön, jos edellisen olion käyttöön ottamisesta on kulunut riittävän paljon aikaa. Olioita otetaan pois käytöstä, jos ne ovat ylittäneet näytön rajat tai pelaaja on osunut niihin hahmollaan.

```
PositiveEntity pCur;
Iterator<PositiveEntity> iter = plusses.iterator();

while(iter.hasNext()){
    pCur = iter.next();
    pCur.update(Gdx.graphics.getDeltaTime(), game.HEIGHT);
    if(pCur.bounds.overlaps(player.bounds)){
        scorebar.addToScore(PLUS_SCORE_VALUE);
        iter.remove();
        pluspool.free(pCur);
    }
    else if(!pCur.visible){
        iter.remove();
        pluspool.free(pCur);
    }
}
```

Kuva 13. PlayScreenin update-metodin osa, jossa tarkistetaan osuuko pelaaja pisteitä antaviin peliobjekteihin.

5.3.3 Kontrollit

PlayScreen periytettiin LibGDX:n InputAdapter-luokasta, joka sisälsi false-arvon palauttavat oletustoteutukset InputProcessor-rajapinnan vaatimille metodeille. Tämän pelin kontrollien toteuttamiseen tarvittiin ainoastaan touchDown-, touchUp- ja touchDragged-metodit. Jotta kosketukseen reagoitaisiin vain silloin kun pelaajan sormi on pelihahmon päällä, luodaan tätä tarkistusta varten erillinen metodi, jota kutsutaan jokaisen eri kosketustapahtuman yhteydessä (kuva 14).

```
@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
    if(isTouchOnPlayer(screenX, screenY)) {
        player.setPosition(screenX, screenY);
        return true;
    }
    return false;
}

private boolean isTouchOnPlayer(int screenX, int screenY){
    return Math.abs(screenX - player.getX()) <= player.getWidth()*2
            && Math.abs(screenY - player.getY()) <= player.getHeight()*2;
}
```

Kuva 14. Pelaajan kosketukseen reagoiminen.

5.3.4 Aika- ja pistelaskuri

Näytön ylälaidassa sijaitsevalle jäljellä olevan ajan ja pistemäärän näyttävälle palkille luotiin oma luokka nimeltä Scorebar, jonka konstruktorissa määritetään sen ulkoasu ja rakenne LibGDX:n Scene2d-elementeillä (kuva 15). Scorebar toteuttaa LibGDX:n rajapinnan Disposable eli se sisältää erillisen dispose-nimisen metodin ko. luokan olion poistamiseksi, koska Scorebarin käyttämiä Scene2d-elementtejä ei poisteta automaattisesti muistista. Näin ollen pitää itse muistaa poistaa Scorebar-oliot omassa koodissa kutsumalla dispose-metodia. Scorebarin dispose-metodia kutsutaan poistettaessa PlayScreen, koska siinä luodaan Scorebar-olio.

```
stage = new Stage(port, batch);
Table table = new Table();
table.top();
table.setFillParent(true);

timeLabel = new Label("TIME", new Label.LabelStyle(new BitmapFont(), Color.WHITE));
timeLabel.setFontScale(2f);
scoreLabel = new Label("SCORE", new Label.LabelStyle(new BitmapFont(), Color.WHITE));
scoreLabel.setFontScale(2f);
timeCountLabel = new Label(String.format("%02d", timeCount), new Label.LabelStyle(new BitmapFont(), Color.WHITE));
timeCountLabel.setFontScale(2f);
scoreCountLabel = new Label(String.format("%06d", score), new Label.LabelStyle(new BitmapFont(), Color.WHITE));
scoreCountLabel.setFontScale(2f);

table.add(timeLabel).expandX();
table.add(timeCountLabel).expandX();
table.add(scoreLabel).expandX();
table.add(scoreCountLabel).expandX();

stage.addActor(table);
```

Kuva 15. Osa Scorebar-luokan konstruktoria.

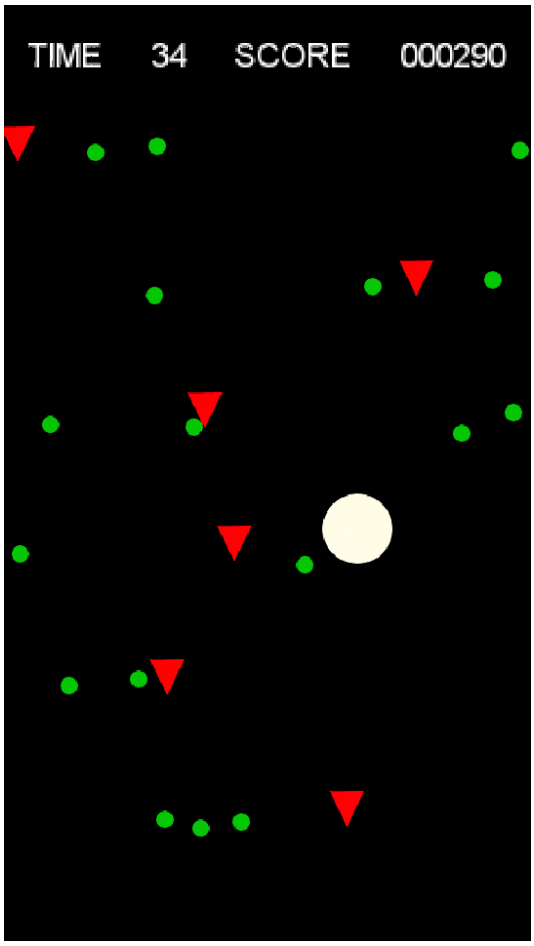
Scorebarissa suoritetaan pelin jäljellä olevan ajan laskeminen sekä pisteiden lisääminen ja vähentäminen (kuva 16). Ajan laskeminen tapahtuu update-metodissa, joka ottaa parametrina vastaan edellisestä näytön päivityksestä kuluneen ajan eli delta timen, jota hyödynnetään sekuntien laskemisessa. Update-metodissa myös reagoidaan peliajan loppumiseen, jolloin asetetaan timesUp-kentän arvoksi true. PlayScreenissä kutsutaan Scorebar-olion timesUp-kentän arvon palauttavaa timesUp-metodia jokaisella näytönpäivityksellä, jotta tiedetään, koska päättää sen hetkinen peli ja siirtyä ScoreScreeniin.

Pisteitä lisätään kutsumalla PlayScreenin update-metodissa Scorebar-olion addToScore-metodia, joka ottaa parametrinaan vastaan lisättävän arvon. Parametrin arvo lisätään score-kenttään.

```
public void update(float delta){
    if(!timesUp) {
        timer += delta;
        if (timer >= 1) {
            timeCount--;
            timeCountLabel.setText(String.format("%02d", timeCount));
            timer = 0;
        }
        if (timeCount <= 0)
            timesUp = true;
    }
}

public void addToScore(int value){
    if(!timesUp){
        score += value;
        scoreCountLabel.setText(String.format("%06d", score));
    }
}
```

Kuva 16. Scorebar-luokan update- ja addToScore-metodit.

5.4 Lopputulos



Kuva 17. Näytönkaappaus kehitetystä pelistä.

Peli saatiin toimimaan hyvin, mutta ylhäältä ”putoavien” peliobjektien nopeudessa ja koossa voisi olla vielä parannettavaa. Lisäksi pelaajahahmon ja muiden peliobjektien törmäysten tunnistusta (collision detection) voisi hieman säätää; rajat voisivat olla suppeammat kuin mitä tekstuurit ovat.

Jatkokehitystä ajatellen pelin ulkoasua voisi parantaa, tehdä selkeät valikot ja ehkä lisätä jonkinlaista fysiikanmallinnusta esim. peliobjektien törmäykset vaikuttaisivat niiden liikerataan. Lisäksi pisteitä antavia objekteja voisi olla useampia erilaisia. Samanlaisia objekteja peräkkäin kerätessä pelaaja saisi enemmän pisteitä.

6 LOPUKSI

Pelistä saatiin tehtyä toimiva ja mielestäni opin peliohjelmoinnin perusteita eli työn tavoitteet täyttyivät. Luotu peli on tosin yksinkertainen toteutukseltaan ja olen ajatellut jatkokehittää sitä ennen kuin aloitan mahdollisesti kehittämään toteutukseltaan haastavampia pelejä.

LÄHTEET

Aftab, M.U. & Karim, W. 2014. Learning Android Intents. Birmingham: Packt Publishing Ltd.

Android Developers. Activity. Päivitetty 20.10.2016. Viitattu 22.12.2016. https://developer.android.com/reference/android/app/Activity.html

Android Developers. App Manifest. Päivitetty 12.10.2016. Viitattu 20.12.2016. https://developer.android.com/guide/topics/manifest/manifest-intro.html

Android Developers. Application Fundamentals. Päivitetty 12.11.2016. Viitattu 20.12.2016. https://developer.android.com/guide/components/fundamentals.html

Android Developers. Content Providers. Päivitetty 20.12.2016. Viitattu 21.12.2016. https://developer.android.com/guide/topics/providers/content-providers.html

Android Developers. Platform Architecture. Päivitetty 12.10.2016. Viitattu 26.12.2016. https://developer.android.com/guide/platform/index.html

Android Developers. Services. Päivitetty 15.12.2016. Viitattu 22.12.2016. https://developer.android.com/guide/components/services.html

Android Developers. The Activity Lifecycle. Päivitetty 22.12.2016. Viitattu 23.12.2016. https://developer.android.com/guide/components/activities/activity-lifecycle.html#lc

Android Open Source Project. Android Interfaces and Architecture. Päivitetty 23.8.2016. Viitattu 26.12.2016. https://source.android.com/devices/index.html

Android Open Source Project. ART and Dalvik. Päivitetty 23.8.2016. Viitattu 26.12.2016. http://source.android.com/devices/tech/dalvik/index.html

Android Open Source Project. Welcome to the Android Open Source Project. Päivitetty 19.12.2016. Viitattu 27.12.2016. http://source.android.com/

Badlogic Games. Event handling. Päivitetty 11.1.2016. Viitattu 7.11.2016. https://github.com/libgdx/libgdx/wiki/Event-handling

Badlogic Games. File Handling. Päivitetty 18.8.2016. Viitattu 7.11.2016. https://github.com/libgdx/libgdx/wiki/File-handling

Badlogic Games. Graphics. Päivitetty 12.1.2014. Viitattu 7.11.2016. https://github.com/libgdx/libgdx/wiki/Graphics

Badlogic Games. Input handling. Päivitetty 11.1.2016. Viitattu 7.11.2016. https://github.com/libgdx/libgdx/wiki/Input-handling

Badlogic Games. Introduction. Päivitetty 2.6.2016. Viitattu 3.11.2016. https://github.com/libgdx/libgdx/wiki/Introduction

Badlogic Games. Project Setup Gradle. Päivitetty 1.3.2016. Viitattu 5.11.2016. https://github.com/libgdx/libgdx/wiki/Project-Setup-Gradle

Badlogic Games. Setting up your Development Environment (Eclipse, Intellij IDEA, NetBeans). Päivitetty 19.9.2016. Viitattu 5.11.2016. https://github.com/libgdx/libgdx/wiki/Setting-up-your-Development-Environment-%28Eclipse%2C-Intellij-IDEA%2C-NetBeans%29

Badlogic Games. Starter classes & configuration. Päivitetty 27.3.2016. Viitattu 10.11.2016. https://github.com/libgdx/libgdx/wiki/Starter-classes-%26-configuration

Badlogic Games. The life cycle. Päivitetty 20.12.2015. Viitattu 6.11.2016. https://github.com/libgdx/libgdx/wiki/The-life-cycle

Chron. What Knowledge Do You Need to Be a Game Designer. Päivitetty 1.1.2017. Viitattu 1.1.2017. http://work.chron.com/knowledge-need-game-designer-13974.html

Creative Skillset. Game Designer. Päivitetty 1.1.2017. Viitattu 1.1.2017. http://creativeskillset.org/job_roles/331_game_designer

Developer Economics 2014. Top Game Development Tools: Pros and Cons. Päivitetty 1.1.2017. Viitattu 1.1.2017. https://www.developereconomics.com/top-game-development-tools-pros-cons

Jackson, W. 2014. Android Apps for Absolute Beginners. 3. painos. Apress Media LLC.

Kotaku 2013. A Beginner's Guide to Making Your First Videogame. Päivitetty 1.1.2017. Viitattu 1.1.2017. http://kotaku.com/5979539/a-beginners-guide-to-making-your-first-video-game

Nair, S.B. & Oehlke A. 2015. Learning LibGDX Game Development. 2. painos. Birmingham: Packt Publishing Ltd.

SitePoint. Starting Android Development, Creating a Todo App. Päivitetty 12.4.2016. Viitattu 7.12.2016. https://www.sitepoint.com/starting-android-development-creating-todo-app/

Statista. Global market share held by smartphone operating systems from 2009 to 2015. Päivitetty 27.12.2016. Viitattu 27.12.2016. https://www.statista.com/statistics/263453/global-market-share-held-by-smartphone-operating-systems/

Sylvester, T. 2013. Designing Games. Sebastopol: O'Reilly Media Inc.

Tutorialspoint. Android – Application Components. Päivitetty 22.12.2016. Viitattu 22.12.2016. https://www.tutorialspoint.com/android/android_application_components.htm