Joonas Meriläinen

# Pelin kehitys Libgdx-sovelluskehyksellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

23.11.2015

Metropolia

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Joonas Meriläinen<br>Pelin kehitys Libgdx-sovelluskehyksellä<br>33 sivua + 2 liitettä<br>23.11.2015 |
| Tutkinto | Insinööri (AMK) |
| Koulutusohjelma | Tietotekniikka |
| Suuntautumisvaihtoehto | Ohjelmistotekniikka |
| Ohjaaja(t) | Lehtori Miikka Mäki-Uuro<br>Lehtori Simo Silander |




Insinöörityössä luotiin mobiilipeli Libgdx-sovelluskehyksellä. Työn teoriaosuudessa selvitettiin, mikä on Libgdx. Libgdx on pääasiassa kaksiulotteisten pelien kehitykseen tarkoitettu sovelluskehys, joka on rakennettu monien suurien kirjastojen ja rajapintojen päälle. Sen taustajärjestelminä toimii Lightweight Java Game Library, Android SDK, RoboVM, JavaScript sekä WebGL. Tämän lisäksi selvitettiin, että Libgdx on tapahtumaohjattu järjestelmä, ja se koostuu kuudesta erilaisesta moduulista, joiden avulla voidaan hoitaa grafiikka, tiedoston käsittely, ääni, verkko-ominaisuudet sekä syötteenhallinta järjestelmäriippumattomasti.

Teoriaosuuden jälkeen työssä kerrottiin pelin suunnitteluprosessista. Tässä osuudessa selitettiin, millainen peli insinöörityötä varten olisi tarkoitus toteuttaa, miksi tietynlaisiin ratkaisuihin on päädytty ja millä tavalla suunnitteluprosessi toteutettiin. Suunnittelussa päädyttiin toteuttamaan tasohyppelypeli, jonka pääideana oli uudenlainen pelihahmon liikutustapa, jossa liikkumiseen ei käytetä yhtäkään näyttöpainiketta.

Työn lopuksi kerrottiin, miten peli toteutettiin suunnittelun pohjalta. Pelin tavoitteena oli kokeilla, miten suunniteltu tasohyppelyn liikkumismalli toimisi mobiililaitteella, sekä esitellä, millaista Libgdx-sovelluskehyksellä kehittäminen on. Osiossa kerrottiin, miten peli on toteutettu ja minkälaisia Libgdx:n työkaluja pelin tekoon käytettiin.




| | |
|---|---|
| Avainsanat | Libgdx, mobiilipeli, järjestelmäriippumaton kehitys |

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Joonas Meriläinen<br>Game Development with Libgdx-framework<br><br>33 pages + 2 appendices<br>23 November 2015 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information technology |
| Specialisation option | Software Engineering |
| Instructor(s) | Miikka Mäki-Uuro, Senior Lecturer<br>Simo Silander, Senior Lecturer |




In this thesis a mobile game was created with Libgdx-framework. Libgdx is a framework mainly intended for two-dimensional game developing and is built on top of many big libraries and interfaces. The back ends of Libgdx are Lightweight Java Game Library, Android SDK, RoboVM, JavaScript and WebGL. Furthermore, Libgdx is an event-driven system and it consists of six different modules that handles graphics, file handling, audio, networking and input handling platform independently.

The study explains the game design process in detail i.e. what kind of game would be created for the thesis and why certain kinds of solutions were made and how the planning process was performed. The conclusion of the planning was that the game would be a platformer in which the main idea would be new kind of character movement.

The paper also explain how the actual game was developed. The main focus of the game was to try how the planned movement would actually work on mobile platform and to introduce what it is like to develop with Libgdx-framework, including what kind of Libgdx's tools were used on the game.




| | |
|---|---|
| Keywords | Libgdx, mobile game, cross-platform development |

## Sisällys

Lyhenteet

1 Johdanto 1

2 Libgdx 1

2.1 Yleistä 2
2.2 Arkkitehtuuri 3
2.3 Elinkaari 6
2.4 Kirjastot 7
2.4.1 Scene2D 7
2.4.2 Box2D 9
2.4.3 FreeType 10

3 Mobiilipelin toteutus 11

3.1 Suunnittelu 11
3.1.1 Pelin suunnittelu 11
3.1.2 Pelin tavoite 13
3.1.3 Pelin rajaukset 13
3.2 Työskentely 13
3.2.1 Projektin luominen 13
3.2.2 Projektin rakenne 15
3.3 Pelin ominaisuudet 16
3.3.1 Törmäystarkistus 16
3.3.2 Pelihahmon liikkuminen 18
3.3.3 Tason aloituspaikka ja läpäiseminen 22
3.3.4 Pelin häviäminen 22
3.4 Grafiikka 23
3.5 Pelitilan tallennus 27
3.6 Valikot 28
3.7 Hud 31

4 Päätelmät 31

5 Yhteenveto 33

Lähteet 34

Liitteet

Liite 1. Libgdx-projektin rakenne

Liite 2. SaveManager-luokan toteutus

## Lyhenteet

| | |
|---|---|
| API | Application Programming Interface. Ohjelmointirajapinta. |
| DTO | Data Transfer Object. Objekti, joka siirtää dataa. Objektilla ei ole muita toimintoja kuin tiedon jakaminen ja asettaminen. |
| fps | Frames Per Second. Ruudunpäivitysnopeus. |
| GLFW | GL Frame Work. OpenGL-ohjelmointia yksinkertaistamaan luotu rajapinta. |
| GWT | Google Web Toolkit. Avoimen lähdekoodin kirjasto, jolla Java-koodi voidaan kääntää JavaScriptiksi. |
| hud | Head-Up Display. Käyttöliittymä, joka piirretään pelitilan päälle. Se kertoo pelin tilasta esimerkiksi pelaajan jäljellä olevat elämät. |
| JNI | Java Native Interface. Ohjelmointirajapinta, jonka avulla Java-koodi pystyy kutsumaan ja tulla kutsutuksi muista ohjelmistoista, jotka on kirjoitettu esimerkiksi C- tai C++ -kielellä. |
| JSON | JavaScript Object Notation. Tiedonsiirtoon tarkoitettu tiedostomuoto, jossa data on esitetty avain-arvo-pareina. |
| MP3 | MPEG-1 Audio Layer 3. Äänenpakkausmenetelmä. |
| OGG | Tiedostoformaatti, joka sisältää pakattua ääntä tai kuvaa. |
| OpenAL | Open Audio Library. Alusta- ja laitteistoriippumattomaan ääniohjelmointiin tarkoitettu ohjelmointirajapinta. |
| OpenGL | Open Graphics Library. Rajapinta, joka on tarkoitettu grafiikan tuottamiseen. |
| SDK | Software Development Kit. Sovelluksen kehitykseen tarkoitetut työkalut. |
| WebGL | Web Graphics Library. JavaScript-rajapinta grafiikan piirtämiseen. |

| | |
|---|---|
| WAV | Waveform Audio File Format. Tiedostomuoto äänen tallennukseen. |

# 1 Johdanto

Mobiililaitteet ovat yleistyneet huimasti ja tätä myöten myös mobiilipelaaminen. Mobiililaitteita on kuitenkin paljon erilaisia ja niitä löytyy erilaisilla käyttöjärjestelmillä. Saman pelin ohjelmoiminen useille laitteille erikseen vaatisi paljon työtä. Tästä syystä nykyään on olemassa useita erilaisia järjestelmäriippumattomia pelimoottoreita ja sovelluskehyksiä, joiden avulla sama koodi voidaan suorittaa erilaisissa järjestelmissä.

Tämän insinöörityön tavoitteena on tutustua Libgdx-sovelluskehykseen, joka on yksi näistä järjestelmäriippumattomista sovelluskehyksistä. Työn aluksi esitellään sovelluskehys. Esittelyn tarkoitus on selventää lukijalle, mikä Libgdx on. Esittely selvittää, minkälaisille järjestelmille Libgdx:llä voidaan kehittää pelejä, millainen sen arkkitehtuuri on sekä millainen on Libgdx-sovelluksen elinkaari. Työssä tutustutaan tarkemmin myös muutamaan Libgdx:n sisältämään kirjastoon, joita käytetään työn ohella tehdyssä mobiilipelin toteutuksessa.

Esittelyn jälkeen työ käsittelee toteutettua mobiilipeliä. Tämän osion alussa kerrotaan mobiilipelin suunnitteluvaiheesta sekä esitellään kehitetty peli. Loppuosasta selviää, miten pelin ominaisuudet kuten grafiikan piirto, törmäystarkastus ja käyttöliittymä on toteutettu ja miksi ne on toteutettu juuri kyseisellä tavalla. Loppuosassa kuvataan myös ongelmia, joita toteutuksen varrella on kohdattu sekä kerrotaan miten ongelmat on ratkaisu. Aivan työn lopuksi kerron oman mielipiteeni Libgdx-sovelluskehyksestä ja sillä kehittämisestä.

# 2 Libgdx

Tässä luvussa käydään läpi, mikä Libgdx on, mihin se on tarkoitettu ja mitä sillä voidaan tehdä. Luku esittelee Libgdx:n taustalla olevan arkkitehtuurin, Libgdx-sovelluksen elinkaaren sekä kolme Libgdx:stä löytyvää kirjastoa, joita tämän insinöörityön yhteydessä toteutetun pelin tekoon on käytetty.

## 2.1 Yleistä

Libdgx on pääasiassa 2D-peliohjelmointiin tarkoitettu avoimen lähdekoodin sovelluskehys Javalle. Projektin alkuisä on Mario Zechner, mutta Libgdx-tiimiin kuuluu tällä hetkellä 16 henkilöä [1]. Koska Libgdx on avoimen lähdekoodin projekti, on projektiin kuitenkin osallistunut jo yli 300 henkilöä. Libgdx:ää kehitetään aktiivisesti koko ajan. Sen viimeisin versio 1.7.0 julkaistiin 21.9.2015.

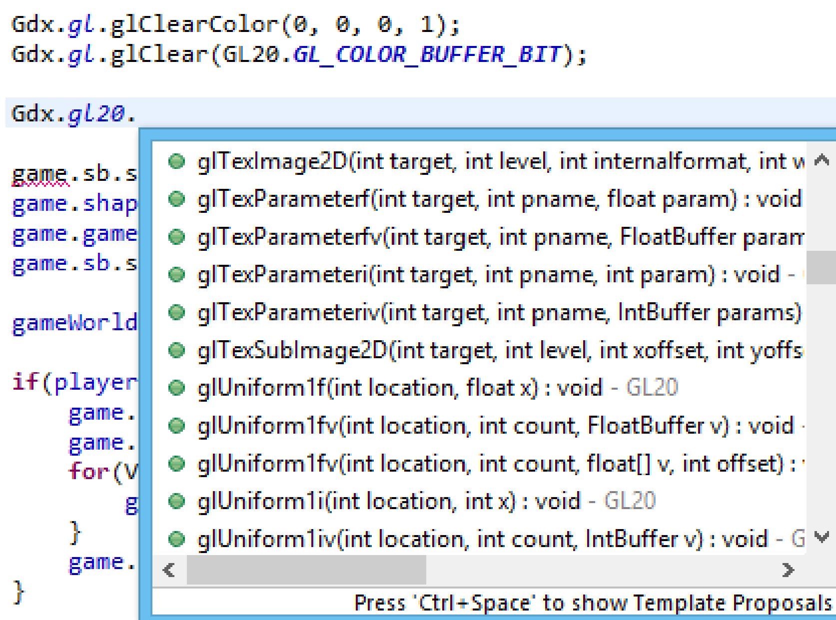
Libgdx on järjestelmäriippumaton sovelluskehys, eli Libgdx:n avulla kehittäjän pitää kirjoittaa koodi vain kertaalleen, jonka jälkeen peli toimii useilla erilaisilla alustoilla ilman koodin muutoksia. Libgdx:llä kirjoitetun koodin voi kääntää Windows-, Linux-, Mac OS X-, Android-, Blackberry-, iOS- ja HTML5-alustoille [2]. Libgdx-sovelluskehys on lisensoitu Apache 2 -lisenssin alle, joka tarkoittaa sitä, että Libgdx:n käyttö on täysin ilmaista sekä ei-kaupallisissa että kaupallisissa projekteissa, eikä käyttäjän tarvitse välttämättä edes mainita sovelluksessaan, että projektissa on käytetty tätä sovelluskehystä [3].

Libgdx:n järjestelmäriippumattomuuden takaamiseksi se käyttää taustallaan neljää erilaista järjestelmää, joiden avulla samat kutsut saadaan toimimaan eri käyttöjärjestelmissä. Windows-, Linux- ja Mac OS X -sovellusten taustalla käytetään Lightweight Java Game Libraryä (LWJGL), joka on matalan tason Java-kirjasto, joka mahdollistaa OpenGL:n, OpenAL:n, GLFW:n ja muiden natiivi API:en käytön Java-ympäristössä. Android-sovellusten pohjana käytetään Googlen kehittämää Android SDK:ta. iOS-sovellusten pohjana käytetään RoboVM:ää, jonka avulla Javan binäärikoodi voidaan kääntää toimimaan iOS-järjestelmissä. Web-sovellusten pohjana toimii JavaScript. Java-koodi käännetään GWT:n ja erilaisten JavaScript-kirjastojen avulla JavaScriptiksi, jolla on HTML5-, WebGL- ja äänentoistotuki [4].

Yleisesti oletetaan, että Java ei sovellu pelikehitykseen, mikä johtuu sisäisen roskienkeruun hitaudesta. Libgdx yrittää kuitenkin tarjota parhaan mahdollisen suorituskyvyn hyödyntämällä JNI-ohjelmointirajapintaa. Tämän avulla saadaan vähennettyä Javan heikkouksia pelikehityksessä, koska paljon suorituskykyä vaativat asiat voidaan tehdä esimerkiksi C++:n avulla ja vähentää Javan oman roskienkeruun kutsumista [5]. Libgdx hyödyntää useita erilaisia tehokkaita matalan tason kirjastoja, kuten OpenGL:ää, OpenAL:ää ja Box2D:tä. Kaikki nämä on kääritty Javalle sopivaan tapaan sekä helposti ymmärrettävästi ja helposti käytettäväksi. Näiden suurten Libgdx:n pohjalla toimivien kirjas-

tojen ja rajapintojen avulla Libgdx tarjoaa työkalut grafiikan piirtämisestä musiikin toistoon sekä JSON:in jäsentelystä trigonometristen laskujen helppoon laskemiseen. Tämän lisäksi kaikkien Java-kirjastojen käyttäminen on täysin mahdollista kehityksessä.

Siitä huolimatta, että natiivikoodi on piilotettu Libgdx:n API:n alle, on kehittäjällä kuitenkin mahdollisuus päästä niin matalalle tasolle, kuin hän itse haluaa. Libgdx tarjoaa korkean tason kutsujen lisäksi myös mahdollisuuden kutsua matalan tason kutsuja. Kuvassa 1 näkyy esimerkki OpenGL:n funktioiden kutsumisesta Libgdx:llä, ja ne ovat varmasti tutun näköisiä OpenGL-kehittäjälle. Libgdx ei yritä olla valmis ratkaisu ja jokaiseen peliin sopiva pelimoottori vaan ennemminkin Libgdx yrittää tarjota mahdollisuuden kirjoittaa peli tavalla, joka kyseiselle pelille on sopivin [2].




Kuva 1. Libgdx:n tarjoamia kutsuja, joilla kutsutaan suoraan OpenGL:n funktioita.

## 2.2 Arkkitehtuuri

Pohjimmiltaan Libgdx koostuu kuudesta erilaisesta ydinmoduulista, jotka muodostavat yhteisen API:n. Nämä ovat funktioita, joita voidaan kutsua kaikissa järjestelmissä, ja ne toimivat samalla tavalla jokaisella alustalla. Kaikki moduulien funktiot ovat julkisia sekä staattisia ja niitä voidaan kutsua Gdx-luokasta. Ydinmoduulit ovat:

- sovellus (Application)

- grafiikka (Graphics)
- tiedosto (Files)
- syöte (Input)
- verkko (Net)
- audio (Audio). [4]

Sovellusmoduuli hoitaa sovelluksen suorituksen ja seuraa siinä tapahtuvia sovellustason tapahtumia kuten esimerkiksi näytön koon muutosta. Sovellusmoduuli tarjoaa funktiot lokitietojen tulostukseen, sovelluksen sammutukseen sekä tavan tallentaa yksinkertaisia tietoja käyttäjän tekemistä muutoksista kuten erilaiset asetustiedot. Sovellusmoduuli tarjoaa myös erilaisia kyselyitä sovelluksen tilasta. Kyselyillä voidaan selvittää esimerkiksi tietoa sovelluksen muistinkäytöstä tai selvittää järjestelmä, jossa sovellus on käynnissä. Kehittäjä pääsee sovellusmoduuliin viitteellä Gdx.app [6].

Grafiikkamoduulin kautta kehittäjä pääsee käsiksi OpenGL-rajapintaan. Grafiikkamoduuli tarjoaa kehittäjälle mahdollisuuden kutsua OpenGL:n tarjoamia funktioita sekä pääsyn säätämään erilaisia videoasetuksia kuten esimerkiksi asettamaan pystytahdistuksen päälle. Moduuli tarjoaa myös funktiot, joilla voidaan selvittää videoasetusten tietoja esimerkiksi ikkunan koon tai fps:n. Grafiikkamoduuliin kehittäjä pääsee käsiksi viitteellä Gdx.graphics ja OpenGL-rajapintaan viitteellä Gdx.graphics.getGL20() tai oikopolkua Gdx.gl20 [7].

Tiedostomoduuli tarjoaa pääsyn kutsuihin, joilla voidaan avata tiedostokahva. Moduulin funktiot on abstrahoitu siten, että eri järjestelmien tiedostonkäsittelyn eroavaisuuksista huolimatta voidaan tiedostot käsitellä samoilla funktioilla joka järjestelmässä. Tiedostokahvan avaamisen jälkeen tiedostoihin voidaan kirjoittaa sekä niitä voidaan lukea, kopioida, siirtää ja poistaa. Tiedostokahvat on jaettu erilaisiin tyyppeihin, mikä johtuu eroista eri järjestelmien tiedoston käsittelyissä. Tyypit ovat sisäiset (internal), ulkoiset (external), lokaalit (local), absoluuttiset (absolute) sekä luokkapolku (classpath). Näistä pelikehityksessä tarpeellisia ovat sisäiset, ulkoiset sekä lokaalit tiedostot. Sisäisiin tiedostoihin kuuluvat kaikki peliin liittyvät tiedostot kuten kuva- ja äänitiedostot, jotka paketoidaan yhteen ohjelman muiden tiedostojen kanssa. Sisäiset tiedostot voidaan avata vain lukemista varten. Lokaaleita tiedostoja tarvitaan, kun halutaan kirjoittaa pieniä tiedostoja kuten esimerkiksi pelin tallennustiedosto. Nämäkin tiedostot on paketoitu yhteen ohjelman mui-

den tiedostojen kanssa. Ulkoisia tiedostoja tarvitaan, kun halutaan käsitellä isoja tiedostoja kuten esimerkiksi kuvakaappauksia. Tällöin tiedostot voidaan tallentaa minne tahansa eikä näitä tiedostoja poisteta samalla, kun poistetaan sovellus. Kehittäjä pääsee tiedostomoduuliin käsiksi viitteellä Gdx.files [8].

Syötemoduuli huolehtii kaikkien erilaisten syötteiden vastaanotosta ja käsittelystä. Se on abstrahoitu siten, että esimerkiksi hiiren painallus tietokoneella ja näytön kosketus mobiililaitteella tulkitaan samanlaiseksi syötteeksi. Syötemoduuli tarjoaa funktiot erilaisten syötteiden vastaanoton seuraamiseen. Kehittäjä voi myös halutessaan lisätä syötemoduuliin erilaisia kuuntelijoita esimerkiksi erilaisten sormieleiden kuten raahaamisen ja nipistyksen kuunteluun, jolloin syötettä ei tarvitse erikseen seurata, koska kuuntelija vastaanottaa syötteen. Kehittäjä pääsee syötemoduuliin käsiksi viitteellä Gdx.input [9].

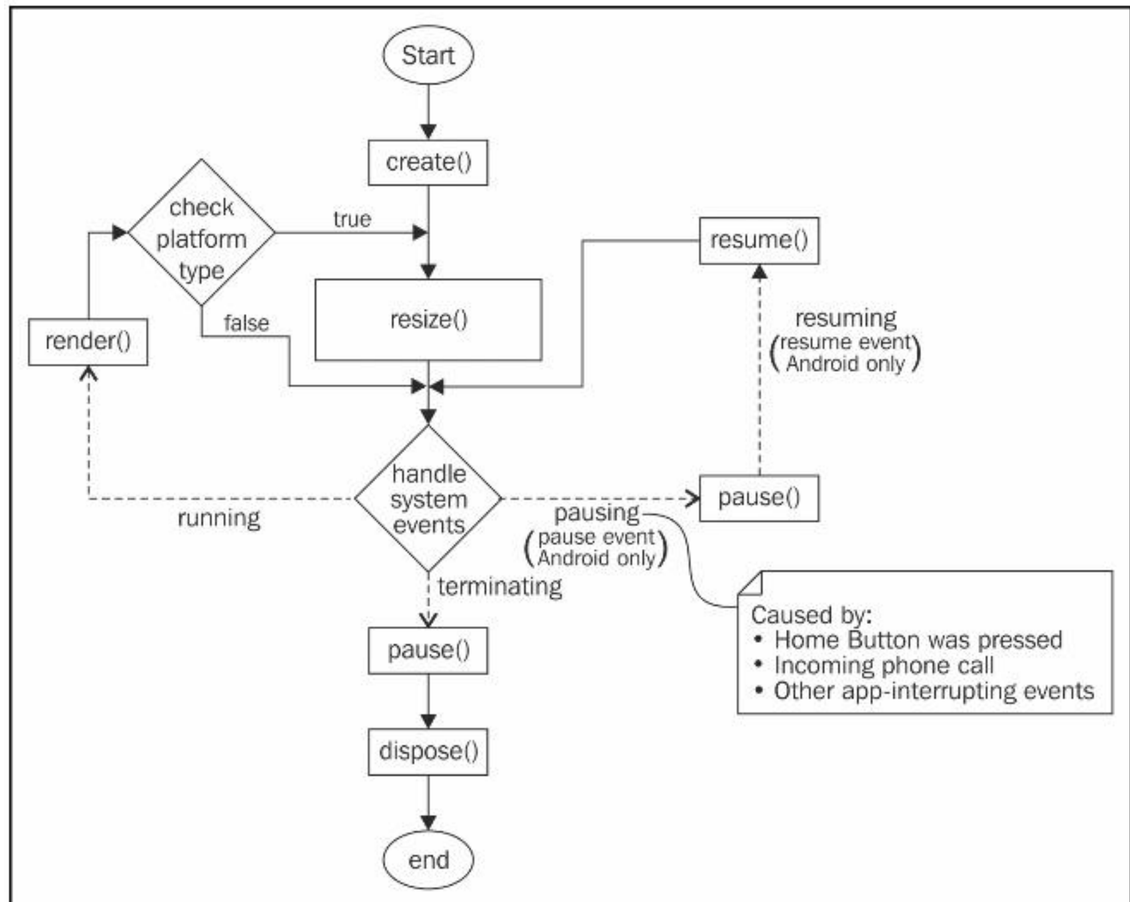
Audiomoduuli hoitaa nimensä mukaisesti kaiken ääniin liittyvät asiat. Libgdx tukee kolmea erilaista äänitiedostojen tallennusformaattia. Nämä ovat MP3, OGG ja WAV. RoboVM kuitenkin rajoittaa sen verran, että se ei tue OGG -muotoa, eli OGG ei toimi iOS-alustoilla [4]. Äänet on jaettu kahteen erilaiseen ryhmään: ääniefekteiksi ja musiikiksi. Ääniefektit ovat pieniä maksimissaan muutaman sekunnin mittaisia ääniä, joita soitetaan vain erikoistilanteissa, kuten esimerkiksi pelihahmon kerätessä esineitä tai ampuessa. Ääniefektitiedostot ladataan kokonaisina suoraan keskusmuistille. Ääniefektitiedoston koko saa olla Androidilla maksimissaan yksi megatavu. Suurempia tiedostoja suositellaan lataamaan musiikkina [10]. Musiikki on tarkoitettu suurempien äänitiedostojen soittoon. Musiikkitiedostoja ei ladata kokonaan välimuistille vaan ne suoratoistetaan levyltä [11]. Kehittäjä pääsee käsiksi audiomoduuliin viitteellä Gdx.getAudio() tai oikopolkua Gdx.audio.

Verkkomoduuli hoitaa kaikki Libgdx:n tarjoamat verkko-ominaisuudet. Sen avulla voidaan lähettää http-pyyntöjä jokaiselle alustalle samoilla käskyillä. Verkkomoduulin avulla voidaan myös avata TCP-asiakkaalle ja palvelimelle socketit, joiden välityksellä ne voivat keskustella. Kehittäjä pääsee käsiksi verkkomoduuliin viitteellä Gdx.net [12].

## 2.3 Elinkaari

Johtuen JavaScriptin ja Android-käyttöjärjestelmän rakenteesta on Libgdx pohjaltaan tapahtumaohjattu järjestelmä. Libgdx-projekteihin ei siis voi luoda perinteistä pääsilmukkaa, jota pyöritetään koko pelin elinkaaren ajan [13]. Sovelluksen käynnistyessä luodaan sovellusmoduuli ja annetaan se kyseisen käyttöjärjestelmän Libgdx-taustajärjestelmälle. Sovellusmoduuliin liitetään sovelluskuuntelija, joka toimii ikään kuin Libgdx-projektin pääluokkana. Tämän jälkeen sovellusmoduuli kutsuu kuuntelijaa aina, kun tapahtuu sovellustason muutoksia. Sovelluskuuntelijan rajapinta koostuu kuudesta erilaisesta funktiosta, jotka kehittäjän on implementoitava omaan sovelluskuuntelijaluokkansa. Nämä funktiot ovat create(), render(), resize(), pause(), resume() ja dispose().

Kuvasta 2 voidaan nähdä, milloin sovellusmoduuli kutsuu kutakin kuuntelijan funktiota. create() kutsutaan, kun ohjelma käynnistyy ensimmäisen kerran. Samalla kutsutaan myös ensimmäisen kerran resize(), kun näytön koko haetaan ensimmäisen kerran. Seuraavaksi kutsutaan render()-funktiota, jossa hoidetaan piirtäminen sekä pelimaailman päivitys. Sovellusmoduuli tarkistaa tässä vaiheessa myös järjestelmän tyypin, jotta se osaa käyttää oikeita funktioita piirron ja päivityksen suorittamiseen. Mikäli pelimaailman päivityksessä on tapahtunut näyttökoon muutoksia, kutsutaan resize() uudelleen. Jos näyttökoon muutosta ei ole tapahtunut, jatketaan odottamaan seuraavia tapahtumia. Suljettaessa sovellusta kutsutaan ensimmäiseksi pause(). Tämän jälkeen kutsutaan dispose(), jossa siivotaan muistista kaikki sovelluksen tiedot, joita Javan oma roskienkeruujärjestelmä ei osaa siivota. pause() kutsutaan Android-järjestelmässä myös silloin, kun käyttäjä poistuu sovelluksesta ”home”-painikkeella tai vastaanottaa puhelun. Tällöin sovellus jää taustalle vielä auki ja voidaan avata uudelleen samasta kohdasta kuin mihin jäätiin ennen kuin kutsutaan dispose(). Kun sovellus palautetaan takaisin, kutsutaan resume() ja aletaan taas odottamaan seuraavia tapahtumia [4].

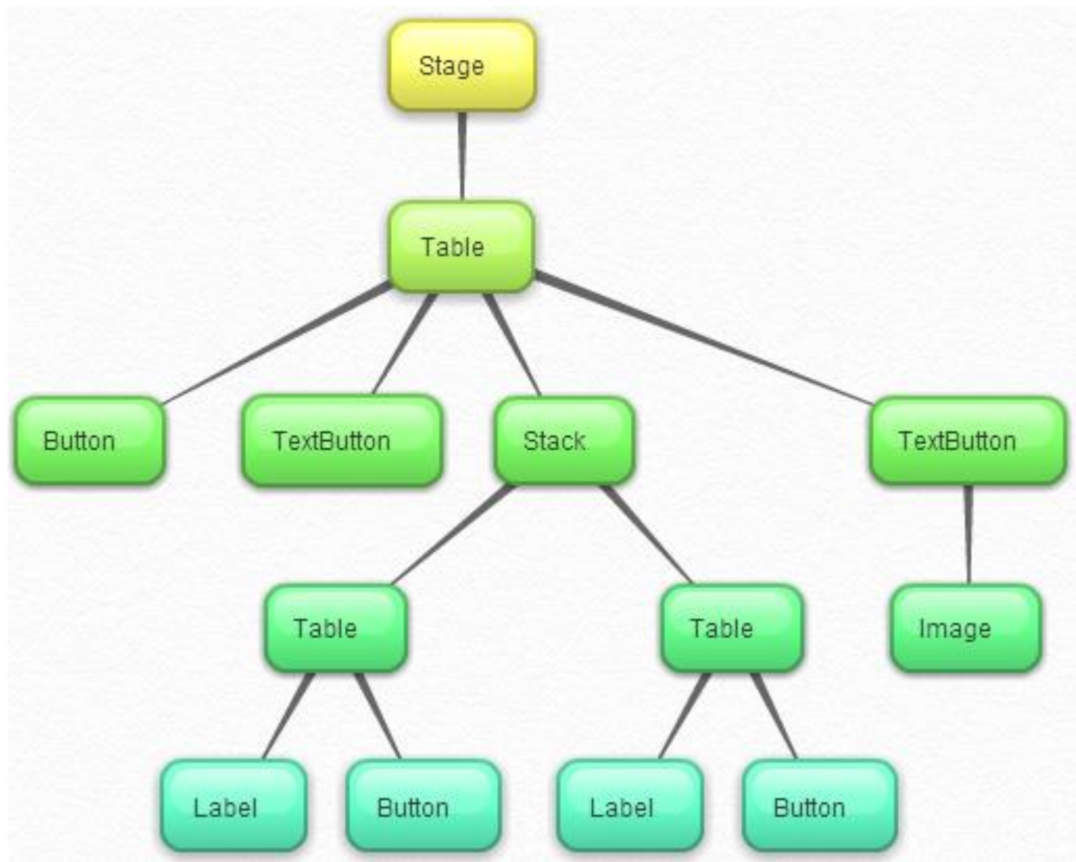



Kuva 2. Libgdx-sovelluksen elinkaari [4].

## 2.4 Kirjastot

### 2.4.1 Scene2D

Scene2D on Libgdx:n sisäänrakennettu ohjelmointirajapinta. Se on suunniteltu erilaisten scene graph -tietorakenteiden luomiseen. Scene graph on tietorakenne, jossa solmut on asetettu hierarkkisesti puurakenteeseen. Solmu voi olla myös ryhmä, johon liittyy useampia solmuja, jolloin puurakenne voi levitä miten tahansa. Kaikki alimman tason solmuun kohdistuvat vaikutukset vaikuttavat myös kaikkiin lapsisolmuihin [14]. Stage-objekti on tietorakenteen juuriolio Scene2D:ssä. Siihen voidaan liittää erilaisia Actor-objekteja, jotka toimivat tietorakenteen solmuina. Mistä tahansa luokasta voidaan tehdä Actor-objekti perimällä Actor-luokka. Kuvassa 3 on esimerkki scene graph -tietorakenteesta, jossa Stageen on liitetty erilaisia Actoreita Scene2D UI paketista.




Kuva 3. Esimerkki scene graph -tietorakenteesta.

Stage-objektilla on joko oma tai sille annettu kamera tai Viewport-objekti, joka hoitaa kameran toiminnan. Erilaisten Viewporttien avulla Stage osaa itse skaalata kuvan siten, että Stage ja sen Actorit toimivat kaikilla resoluutioilla samalla tavalla. Actoreiden piirtäminen ja toiminta hoidetaan myös Stagen kautta. Kun kutsutaan Stage.act(), kutsutaan myös kaikkien siihen lisättyjen Actoreiden act()-funktiota, jolloin Actorit suorittavat niihin liittyvät toiminnot. Samaan tapaan myös Stage.draw() kutsuu kaikkien actoreiden draw()-funktiota, jolloin ne piirretään. Stage seuraa myös käyttäjän syötettä. Kun se vastaanottaa syötteen, se kutsuu syötteen vastaanottanutta Actoria ja sen kuuntelijaa.

Actoreihin voidaan siis liittää erilaisia kuuntelijoita, jotka seuraavat, liittyikö saatu syöte kyseiseen Actoriin. Kaikki Actorit sisältävät myös listan, johon voidaan liittää erilaisia toimintoja. Tällaisia ovat esimerkiksi Actorin liikuttaminen pisteestä a pisteeseen b tai värin muutos. Toiminnot voidaan toteuttaa joko välittömästi, tai ne voidaan toteuttaa esimerkiksi siten, että kun Actoria painetaan, liikkuu se sekunnissa pisteestä (0,0) pisteeseen (60,0). Toimintoja voidaan myös ketjuttaa, eli voidaan luoda yksi toiminto, jossa Actor

liikkuu pisteestä (0,0) pisteeseen (60,0), ja kun se on perillä, suoritetaan toinen toiminto, jossa liikutaan pisteestä (60, 0) pisteeseen (60, 60) [15].

Scene2D:n avulla voidaan rakentaa kokonaisia pelejä, mutta se sopii erittäin hyvin myös vain pelin käyttöliittymän tekoon. Käyttöliittymän rakentamista varten on Scene2D:hen lisätty paketti Scene2D UI, joka sisältää erilaisia käyttöliittymäelementtejä. Tällaisia elementtejä ovat esimerkiksi tekstikenttä ja erilaiset painikkeet. Se tarjoaa myös erilaisia säiliöitä, joihin elementtejä voidaan laittaa, jolloin säiliö hoitaa elementtien asettelun. Kaikki Scene2D UI:n elementit ovat Actor-objekteja ja kaikki ne ovat säiliöitä, joihin voidaan lisätä Actoreita. Tämä tarkoittaa sitä, että esimerkiksi painikkeeseen voidaan liittää toinen painike, jos käyttöliittymään sellainen halutaan. Scene2D UI:lla voidaan tehdä melkein minkälaisia käyttöliittymiä tahansa [16].

Erilaisten käyttöliittymäelementtien muotoiluun saattaa joutua käyttämään paljon koodirivejä. Tätä ongelmaa auttamaan löytyy Scene2D UI -kirjastosta Skin-luokka. Siihen voidaan säilöä kaikki käyttöliittymän tarvitsemat resurssit, kuten esimerkiksi painikkeiden kuvat tai taustakuvat. Tämän lisäksi Skin-luokkaan voidaan liittää JSON-tiedosto, johon voidaan luoda valmiiksi erilaisia tyylipohjia eri käyttöliittymäelementeille. Kuvassa 4 näkyy TextButton-objektin tyylimäärittely. Siinä kerrotaan, että kun painiketta ei ole painettu, näytetään skiniin lisätystä tekstuuriatlaksesta kuva nimeltään ”button.up”. Kun painiketta painetaan, näytetään kuva ”button.down” ja siirretään painikkeen tekstiä kaksi pikseliä alaspäin. TextButtonin fonttina käytetään aikaisemmin nimellä ”font” määritettyä fonttia. Kun TextButtonin tyyli on määritelty valmiiksi JSON-tiedostossa, voidaan tyylin mukainen painike luoda helposti, antamalla sille parametreina haluttu teksti, viite Skin-objektiin sekä tyylipohjan nimi eli kuvan tapauksessa ”button” [17].

```
"com.badlogic.gdx.scenes.scene2d.ui.TextButton$TextButtonStyle": {
    "button": { "up": "button.up", "down": "button.down", "pressedOffsetY": -2, "font": font }
}
```

Kuva 4. TextButton-objektin tyylimäärittely JSON-muotoisesti.

### 2.4.2 Box2D

Box2D on erittäin suosittu fysiikkamoottori, jolla simuloidaan kaksiulotteisten jäykkien kappaleiden realistista liikettä. Se on alun perin kirjoitettu C++-kielellä, mutta se on toteutettu useille eri kielille ja pelimoottoreille. Box2D:ta käyttää esimerkiksi suomalainen

erittäin suosittu Angry Birds -peli. Myös Libgdx:stä löytyy mahdollisuus käyttää pelin fysiikkamoottorina Box2D:ta. Libgdx:n Box2D-toteutus ei ole juurikaan abstrahoitu, eli Java-funktiot muistuttavat hyvin paljon alkuperäisen C++-toteutuksen funktioita [18].

Box2D-simulaatio tapahtuu omassa maailmassaan. Kaikki Box2D-objektit pitää lisätä maailmaan, jotta ne otetaan huomioon simulaatiossa. Jokaisella Box2D-objektilla on body. Body sisältää tiedon esimerkiksi objektin sijainnista, massasta ja nopeudesta. Bodyt on jaettu kolmeen erilaiseen tyyppiin, jotka toimivat eri tavoilla. Staattinen body ei reagoi voimiin, eikä sille voida asettaa nopeutta. Tällaista bodyä on hyvä käyttää tasoissa, joiden päällä pelihahmo voi kävellä. Dynaaminen body on staattisen bodyn vastakohta. Se reagoi erilaisiin voimiin kuten painovoimaan, ja se pystyy liikkumaan sekä pyörimään voimien vaikutuksesta. Kinemaattinen body on kahden aiemman body-tyypin yhdistelmä. Siihen eivät vaikuta erilaiset voimat, mutta se pystyy kuitenkin liikkumaan ja pyörimään. Kinemaattista bodyä voidaan käyttää esimerkiksi liikkuvien alustoiden tekemiseen [19].

Tiedot kappaleen muodosta, kimmoisuudesta, kitkasta sekä tiheydestä löytyvät fixturesta. Fixturen muoto on se, jolla Box2D tarkistaa, onko body törmännyt johonkin. Bodyyn pitää siis liittää vähintään yksi fixture, jotta body voi törmätä johonkin. Fixtureille voidaan myös lisätä filttereitä, jotka määrittävät kappaleet, joihin fixturen halutaan ja ei haluta törmäävän. Yhteen bodyyn voidaan lisätä useita fixtureita, joilla kaikilla voi olla erilaiset ominaisuudet. Tämä mahdollistaa sen, että Box2D-objekti voi olla hyvinkin monimutkainen [20]. Fixture voidaan määritellä myös sensoriksi. Sensorit eivät fyysisesti törmää mihinkään, mutta niiden kosketus toisten fixtureiden kanssa rekisteröidään ja voidaan käsitellä [21].

Kaikki fixtureiden kontaktit rekisteröidään kontaktikuuntelijalle. Kuuntelija sisältää funktiot BeginContact() ja EndContact(), jotka kutsutaan nimiensä mukaisesti kontaktin alkaessa tai kontaktin loppuessa. Tämän lisäksi se sisältää funktiot PreSolve() ja PostSolve(), joita kutsutaan kontaktin ajan. Näissä funktioissa voidaan suorittaa asiat, joita kontaktin halutaan aiheuttavan [22].

### 2.4.3 FreeType

Jotta pelissä voidaan piirtää tekstiä, täytyy siihen olla lisätty joku käytettävä fontti. Yleensä tätä varten luodaan valmiiksi bitmap-tiedosto, josta löytyvät kaikki halutut merkit,

joita tekstissä tullaan käyttämään. Jokaista merkkiä kohden piirretään aina bitmap-tiedostosta löytyvä merkkiä vastaava kuva. Tällaista fonttia kutsutaan bitmap-fontiksi [23]. Bitmap-tiedostot ovat aika suuria sekä niiden skaalaaminen erilaisille resoluutioille voi aiheuttaa virheitä kuviin, jolloin teksti saattaa joillain resoluutioilla pikselöityä niin pahasti, että tekstin lukemisesta tulee vaikeaa.

Bitmap-fontin ongelmien ratkaisemiseksi on Libgdx:ään toteutettu kääre C-pohjaiselle FreeType moottorille, joka hoitaa erilaisten fonttien käsittelyn. FreeType mahdollistaa bitmap-fonttien generoimisen TrueType font -tiedostoista. Generoitavalle fontille voidaan syöttää erilaisia arvoja kuten haluttu väri, koko tai varjostuksen määrä. Tällä tavalla voidaan yhdestä TrueType font -tiedostosta luoda monia erilaisia ja erikokoisia bitmap-fontteja, eikä pelin mukana tarvitse viedä erikseen jokaiselle eri resoluutiolle sopivia fontteja [24].

# 3 Mobiilipelin toteutus

Tässä luvussa esitellään insinöörityön yhteydessä toteutettu Libgdx:llä toteutettu peli. Luku kertoo, millainen toteutettu peli on ja miksi peli haluttiin tehdä. Luvusta selviää, miten Libgdx:ää käytetään pelinkehityksessä ja millaisia valmiita ratkaisuja se tarjoaa. Luku esittelee myös erilaisia ratkaisuja, joihin pelin ominaisuuksien toteutuksessa on päädytty, ja selitetään, miksi kyseisiin ratkaisuihin on päädytty.

## 3.1 Suunnittelu

### 3.1.1 Pelin suunnittelu

Olen aina ollut tasohyppelypelien ystävä. Pidän erityisesti peleistä, jotka ovat vaikeita, mutta kuitenkin reiluja. Tällä tarkoitan siis peliä, jossa kaikki virheet, joita pelaaja tekee, johtuvat pelaajasta itsestään, eivätkä huonosti tehdystä liikkumisesta tai epäreilusti asetetuista piilotetuista esteistä. Super Meat Boy on yksi lempipeleistäni. Se on juurikin peli, jota olen edellä kuvannut. Siinä liikkuminen on hiottu viimeisen päälle hyvään kuntoon, ja pelaaja tuntee koko ajan olevansa ohjaksissa. Olen jo pidemmän aikaa pohtinut, olisiko mobiililaitteille mahdollista tehdä tasohyppelypeliä, jonka ohjaaminen tuntuu oikeasti hyvältä ja mukavalta. Nyt kehittämäni peliprototyypin idea onkin lähtöisin näistä ajatuksista.

Mobiililaitteille on olemassa paljon erilaisia tasohyppelypelejä, mutta kaikissa ainakin itse pelaamissani pelihahmon ohjaaminen on toteutettu näytölle piirretyillä painikkeilla. Näyttöpainikkeista on toki olemassa hyviä ja huonoja versioita. Huonoimmat ovat vain näytölle piirrettyjä painikkeita, jotka eivät anna minkäänlaista ärsykettä pelaajalle, jotta pelaaja tietää osuneensa painikkeeseen. Huonot painikkeet eivät myöskään anna ollenkaan pelaajalle anteeksi sitä, että hän painaa hieman ohi painikkeesta. Paremmin toteutetut painikkeet värisyttävät hieman näyttöä, jotta pelaaja tietää painaneensa painiketta. Mielestäni myös virtuaalinen peliohjain hahmon liikutukseen on myös parempi vaihtoehto kuin pelkät painikkeet. Pelihahmon ohjaamiseen tarkoitettu virtuaalinen peliohjain voidaan tehdä myös niin, että se ilmestyy sinne, minne pelaaja painaa näytöllä, jolloin pelaaja ei voi vahingossa painaa ohi painikkeesta.

Itse en kuitenkaan pidä juurikaan yhdestäkään yllä olevasta vaihtoehdoista. Virtuaalisilla painikkeilla ohjaaminen ei ole lähellekään niin vaivatonta kuin esimerkiksi fyysisellä peliohjaimella, ja pelaaja turhautuu helposti vääriin painalluksiin tai ohipainalluksiin. Tällöin pelistä katoaa hauskuus, vaikka se olisi muuten todella hyvä ja innovatiivinen. Tätä insinöörityötä varten tahdoin kokeilla, onnistuisiko tasohyppelypelin tekeminen mobiilipelille ilman yhtäkään virtuaalista painiketta tai peliohjainta. Tavoitteena kuitenkin se, että pelihahmon liikuttaminen tuntuu hyvältä ja että tasohyppelyn määrittely toteutuu. Pelin ei tule olla pelkkä yhteen suuntaan juokseminen, jossa ainoa toiminto on, että näyttöä painamalla pelihahmo hyppää, vaikka toki tällaisetkin pelit voivat olla hauskoja.

Peli-idean suunnittelu alkoi siten, että piirsin paperille muutaman hyvin geneerisen tasohyppelypelin tason. Tämän jälkeen aloin pohtia, miten hahmon saisi ohjattua tasojen läpi ilman, että tarvitaan virtuaalisia painikkeita. Otin mallia yhteen suuntaan juoksemispeleistä liikkumiseen siten, että pelihahmo liikkuu koko ajan oikealle tai vasemmalle eikä liikettä voida pysäyttää. Kun liikkuminen oli päätetty, oli aika miettiä, miten pelihahmon hyppäämisen voisi toteuttaa. Päädyin ratkaisuun, että näyttöä pyyhkäisemällä pelihahmon liike pysähtyisi ja hyppääminen tehtäisiin Angry Birds -pelin linnun laukaisemisen tavalla eli sormea liikuttamalla määritetään hypyn voima ja suunta ja päästämällä sormi irti näytöltä hyppää pelihahmo määritettyyn suuntaan määritetyllä voimalla.

Koska olin luonut jo aikaisemmin muutaman peliprototyypin Libgdx-sovelluskehyksen avulla, tiesin jo valmiiksi suurin pirtein, minkälaisia lisäkirjastoja ja aputyökaluja tulisin tarvitsemaan sopivan prototyypin kasaamiseen, joten näiden kanssa ei ollut ongelmia.

### 3.1.2 Pelin tavoite

Pelissä tavoitteena on ohjata pelihahmo lähtöpisteestä maalipisteeseen mahdollisimman nopeasti. Matkalla on esteitä kuten piikkejä, joita pelaajan pitää väistellä. Maaliin päästäkseen pelaajan pitää myös selvittää taso esimerkiksi tasapainoilemalla liikkuvan alustan päällä ja hyppimällä seiniä pitkin päästäkseen ylöspäin.

Pelin periaatteellinen tavoite on testata, miten hyvin tasohyppelypelin ohjaamisen pystyy toteuttamaan ilman virtuaalisia painikkeita ja samalla myös esitellä osa Libgdx:n tarjoamista mahdollisuuksista pelinkehityksessä, kuten Box2D-integraatio, Scene2D UI:n käyttö käyttöliittymänluomisessa sekä pelin kääntö yhdestä koodista usealle eri alustalle.

### 3.1.3 Pelin rajaukset

Peli on rajattu tarkasti, mikä johtuu siitä, että tarkoituksena luoda aluksi vain prototyyppi ohjauksesta ja tarkastella, onko liikkumisidea tarpeeksi hyvä ja varsinkin tarpeeksi hauska, jotta peliä kannattaa jatkokehittää. Myös insinöörityön aikataulu luo oman haasteen projektin loppuun saamiseksi, joten tarkka rajaaminen on tarpeen.

Tätä insinöörityötä varten peli toteutetaan sellaiseen kuntoon, että sitä voidaan pelata. Tämä tarkoittaa siis sitä, että peliin luodaan vähintään yksi pelattava taso, joka sisältää aloituspisteen sekä maalin. Tämän lisäksi pelistä löytyy vähintään yksi este, esimerkiksi piikit, joita voidaan sijoittaa pelin tasoihin ja pelaajan törmätessä niihin peli hävitään. Eniten pelissä panostetaan pelihahmon liikkumiseen. Liikkuminen hiotaan suunnitelman mukaiseen kuntoon parhaalla mahdollisella tavalla. Peliin luodaan myös yksinkertainen käyttöliittymä, joka koostuu päävalikosta ja tasonvalintanäytöstä. Peli toteutetaan siten, että sen kehitystä on helppo jatkaa tulevaisuudessa pidemmälle ja lopulta jopa julkaista.

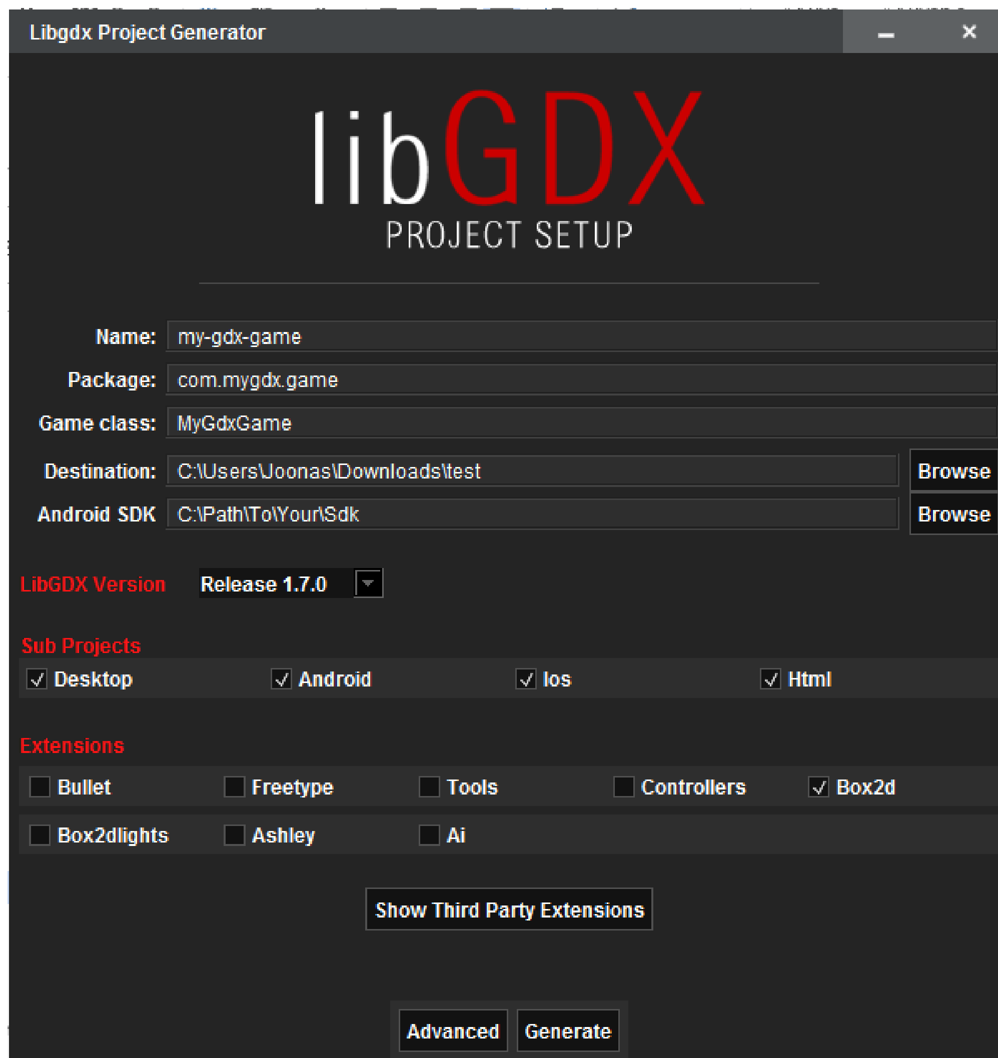
## 3.2 Työskentely

### 3.2.1 Projektin luominen

Libgdx-projekti käyttää Gradlea riippuvuuksien hallintaan sekä koontiprosessiin. Tämä mahdollistaa sen, että Libgdx-projekti toimii missä tahansa Java-kehitysympäristössä, ja projektin tiiminjäsenet voivat kehittää sovellusta myös erilaisilla ympäristöillä. Jotta projektin koonti onnistuu, täytyy kehitysympäristöstä löytyä muutama ylimääräinen sovellus.

Libgdx-projekti vaatii toimiakseen vähintään version 7 JDK:n. Jos halutaan, että sovellus toimii Androidilla, pitää kehitysympäristöstä löytyä Android SDK. Jos halutaan, että sovellus voidaan koota iOS:lle, pitää kehitys tehdä Mac-tietokoneella ja kehitysympäristöstä pitää löytyä XCode sekä RoboVM. Näiden lisäksi kehitysympäristön pitää pystyä käsittelemään Gradle-projekteja, joten ainakin Eclipseen - kehitysympäristö, jota itse käytän projektissa - pitää ladata erillinen Gradle-lisäosa [25].

Libgdx-projekti luodaan erillisellä luontisovelluksella. Ilman sitäkin projekti on mahdollista luoda, mutta luontisovelluksen avulla luominen on niin helppoa, ettei ole oikeastaan mitään syytä, miksi sitä ei käyttäisi. Kuvassa 5 näkyy luontisovelluksen käyttöliittymä. Luontisovelluksessa käyttäjä kertoo, mihin projekti asennetaan, Java-paketin nimen, pääluokan nimen sekä Android SDK:n sijainnin tietokoneella. Näiden lisäksi käyttäjä valitsee alustat, joille sovellus halutaan sekä haluamansa Libgdx:n tarjoamat lisäkirjastot. Sovellus generoi näistä tiedoista automaattisesti Gradle-projektin, joka sisältää kaikki riippuvuudet (esimerkiksi lisäosat), jotka projektiin liittyvät. Kun Gradle-projekti avataan kehitysympäristössä, lataa Gradle automaattisesti kaikki kirjastot projektia varten, ja projektin kehitys voi alkaa. Mikäli jostain syystä ei haluta käyttää Gradle-projektia voi luontisovelluksella luoda myös Eclipse- tai IDEA-kehitysympäristöjen projektitiedostot ilman Gradlea.




Kuva 5. Libgdx-projektin luontisovellus.

### 3.2.2 Projektin rakenne

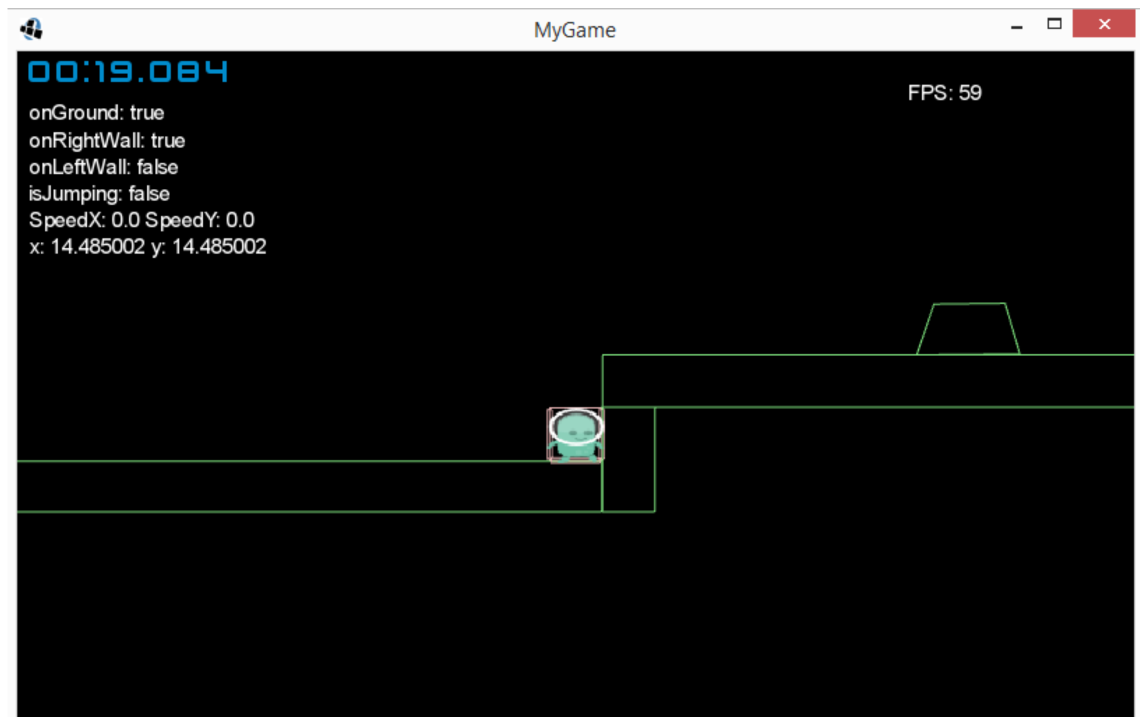
Libgdx-projekti koostuu useista Java-projekteista. Liitteessä 1 näkyy projektin rakenne Eclipsen näkymässä. Jokaiselle alustalle, jolle peli halutaan, luodaan oma Java-projekti. Nämä projektit sisältävät tarpeelliset tiedostot koodin kääntämiseen kyseiselle alustalle sekä pääluokan, jolla voidaan käynnistää sovellusmoduuli. Näiden lisäksi luodaan vielä yksi projekti, johon itse pelin koodi tulee. Mikäli Android on valittu yhdeksi halutuksi alustaksi, luodaan sen Java projektin alle ”assets”-kansio. Tällöin kaikki peliin liittyvät ulkoiset tiedostot, kuten tekstuurit ja äänet pitää laittaa tähän kansioon. Kansio linkitetään kaikkiin muihin projekteihin, jotta ne löytävät myös tiedostot. Jos Androidia ei valittu, voidaan tiedostot tallentaa samaan tapaan kuin tavallisesti Javassa.

## 3.3 Pelin ominaisuudet

### 3.3.1 Törmäystarkistus

Pelin fysiikkamoottorina käytetään Box2D:ta, jonka toiminta on kuvattu kappaleessa 2.4.2. Pelihahmolla on dynaaminen body. Bodyyn on liitetty fixture, joka on neliön muotoinen. Fixturen leveys ja korkeus on määritelty yhdeksi, joka tarkoittaa sitä, että Box2D-maailmassa pelihahmo on metrin korkea ja metrin leveä neliön muotoinen kappale. Pelihahmolla ei ole ollenkaan kitkaa eikä kimmoisuutta. Nämä ovat turhia ominaisuuksia johtuen hahmon liikkumistavasta. Fixturen lisäksi pelihahmon bodyyn on liitetty kolme sensoria. Ne on sijoitettu fixturen vasempaan ja oikeaan laitaan sekä alapuolelle siten, että puolet sensorista tulee ulos fixturesta. Ne eivät myöskään ole aivan sivun pituisia tai korkeita. Tällä vältetään se, että ei synny tilannetta, jossa kaksi sensoria osuu samaan kohteeseen samaan aikaan. Sensorit eivät myöskään tule liikaa ulos fixturesta, jotta sensorit eivät havaitse törmäystä liian aikaisin.

Pelin tasot luodaan Tiled-ohjelmalla. Sillä on mahdollista piirtää tiilikartta sekä piirtää erilaisia objekteja esimerkiksi törmäysaluetta varten. Taso tallennetaan tmx-formaattiin, joka sisältää kaikkien tason elementtien datan xml-muodossa. Pelin staattiset bodyt sekä niiden fixturet parsitaan tmx-tiedostoista Libgdx:n kanssa tulevalla, utils-kirjastosta löytyvällä, Box2DMapObjectParser-luokalla. Sen avulla voidaan parsia tmx-tiedoston törmäysalueiden elementit ja niiden ominaisuudet sekä luoda ominaisuuksien mukainen Box2D-objekti. Lisäsin itse vielä apuluokan Box2DMapObjectParserHelper, joka perii Box2DMapObjectParser-luokan. Apuluokassa lisätään törmäysalueiden elementeille ominaisuudet, jotka muuten joutuisi syöttämään käsin Tiledissä. Kaikille staattisille Box2D-objekteille on asetettu törmäysfiltterit siten, että ne voivat törmätä vain pelihahmon kanssa. Kuvassa 6 näkyy pelihahmon tekstuurin ympäröivä fixture sekä tmx-tiedostosta parsitut fixturet, joihin pelihahmon fixture voi törmätä.




Kuva 6. Pelin tason Box2D-maailma.

Kontaktikuuntelijana toimii luokka MyContactListener, joka perii ContactListener-luokan. Luokassa tarkastellaan, osuvatko pelihahmon sensorit johonkin. Jos ne osuvat, päivitetään pelihahmon tilaa. Esimerkiksi, jos pelihahmon alla oleva sensori osuu johonkin, tiedetään, että pelihahmo seisoo maassa. Mikäli kontakti loppuu, tiedetään, että pelihahmo on irronnut maasta. Kuvassa 7 näkyy toteutus, jossa kuuntelijassa tutkitaan, osuuko pelihahmon alla oleva sensori johonkin. Samaan tapaan toimivat myös pelihahmon fixturen vasemmalla ja oikealla puolella olevat sensorit, mutta ne kertovat, onko pelihahmo kiinni seinässä. Törmäyksiin, joissa ei ole väliä, mikä osa pelihahmosta osuu kohteeseen esimerkiksi maaliin tullessa, ei tutkita sensoreiden kontaktia vaan pelihahmon oikean fixturen kontaktia.

```
@Override
public void beginContact(Contact contact) {

    Fixture fa = contact.getFixtureA();
    Fixture fb = contact.getFixtureB();

    // Ground sensor
    if(fa.getUserData() != null && fa.getUserData().equals("groundSensor")) {
        Player player = (Player) fa.getBody().getUserData();
        player.setOnGround(true);
    } else if(fb.getUserData() != null && fb.getUserData().equals("groundSensor")) {
        Player player = (Player) fb.getBody().getUserData();
        player.setOnGround(true);
    }
```

Kuva 7. Pelihahmon alla olevan sensorin kontaktin tutkiminen.

### 3.3.2 Pelihahmon liikkuminen

Fysiikkamoottorin takia hoidetaan liikkuminen asettamalla pelihahmolle erilaisia voimia riippuen siitä, miten pelihahmon halutaan liikkuvan. Pelihahmon liikkuminen on toteutettu Player-luokan update()-funktiossa, jota kutsutaan aina, ennen kuin päivitetään Box2D-maailma. update()-funktiossa tarkistetaan tila, jossa pelaaja on ja päätetään sen mukaan, mitä pelaajan pitää tehdä.

Koska pelihahmon on tarkoitus liikkua koko ajan jompaankumpaan suuntaan, kutsutaan funktiota move() aina, jos pelihahmo ei ole erikoistilanteessa. Siellä tarkistetaan pelihahmon suunta ja asetetaan voima, jolla pelihahmon halutaan liikkuvan. Asetettava voima lasketaan kaavalla 1. Kaavassa otetaan huomioon nopeuden muutos, koska halutaan, että nopeus muuttuu välittömästi, eikä haluta, että kiihdytetään hitaasti maksiminopeuteen.

$$F = m * \frac{\Delta v}{\Delta t} \qquad (1)$$

$F$ on voima
$m$ on pelihahmon massa
$\Delta v$ on nopeuden muutos
$\Delta t$ on ajan muutos

Kuvassa 8 on nähtävissä koko move()-funktio. Pelihahmon suuntaa muutetaan painamalla näyttöä halutulta puolelta. Näytön vasenta puolta painamalla hahmon suunta muuttuu vasemmalle ja vastaavasti oikeaa puolta painamalla suunta muuttuu oikealle.

```
private void move() {
    Vector2 curSpeed = body.getLinearVelocity();

    float maxVel = 0;

    if(movingRight) {
        maxVel = Constants.DEFAULTVELOCITY;
    } else if(movingLeft) {
        maxVel = -Constants.DEFAULTVELOCITY;
    }

    // f = ma -> f = mv/t -> want to change speed instantly so ->  f = m * (max_v - current_v) / t
    float velChange = maxVel - curSpeed.x;
    float force = body.getMass() * velChange / delta;

    body.applyForceToCenter(new Vector2(force, 0), true);
}
```

Kuva 8. Pelihahmon tavalliseen liikkumiseen käytettävä move()-funktio.

Jatkuva liike ja suunnan välitön muutos aiheuttivat ongelman, joka tuli ilmi vasta testatessa peliä mobiililaitteella. Alun perin suunnan muutos tapahtui, kun kuuntelija kutsui tap()-funktiota, eli näyttöä painetaan ja sormi nostetaan ylös näytöltä liikuttamatta sitä ulos maksimisäteen verran alkuperäisestä painokohdasta. Tämä toimi hyvin hiirellä ohjatessa, sillä hiiren klikkauksen teko on nopeaa. Testatessani puhelimella tuntui tämä kuitenkin aika kankealta. Vaihdoin suunnan muutoksen tapahtumaan kuuntelijan touchdown()-funktioon, joka kutsutaan, kun näyttöä painetaan. Liikkuessa normaalisti tämä tuntui huomattavasti paremmalta, mutta aiheutti jälleen uuden ongelman. Touchdown()-funktiota kutsutaan aina, kun näyttöä kosketetaan, myös silloin kun halutaan tehdä pyyhkäisyele. Tällöin pelihahmo ehti liikkua hetken, ennen kuin siirryttiin tilaan, jossa hyppy säädetään. Tästä seurasi se, että seinähyppyä tehdessä pelihahmo saattoi lähteä liikkumaan irti seinästä, vaikka pelaaja halusi hypätä. Lopulta ratkaisin ongelman muuttamalla suuntaa tap()-funktiossa, mikäli pelihahmo on kiinni seinässä ja ilmassa. Muussa tapauksessa suunnan muutos tapahtuu touchdown()-funktiossa.

Hyppääminen tapahtuu pyyhkäisemällä näyttöä. Kun näyttöä pyyhkäistään, tarkistetaan saadaanko hyppy suorittaa. Hypyn suorituksen ehtona on, että pelihahmon ei saa olla jo hyppytilassa. Tämän lisäksi, jotta hyppy voidaan suorittaa pitää pelaajan olla kiinni joko seinissä tai lattiassa. Jos hyppy voidaan suorittaa, siirtyy pelihahmo hypyn suunnittelutilaan. Tällöin lopetetaan move()-funktion kutsuminen, nollataan hahmon x- ja y-akselin suuntaiset nopeudet. Hypyn impulssi asetetaan sen mukaan, miten pitkälle ja mihin suuntaan pelaaja on raahannut sormea näytöllä pyyhkäisyn alkupisteeseen nähden. Hyppäämisen impulssia päivitetään aina, kun pelaaja liikuttaa sormeaan näytöllä nostamatta sitä. Impulssille on määritetty x- ja y-akselin suuntaiset maksimi-impulssit, joilla estetään, ettei pelaaja voi hypätä liian pitkälle tai liian korkealle. Tämän lisäksi rajoitetaan

pelaajan hyppyä, kun pelaaja on tekemässä seinähyppyä. Tällöin pelaaja pakotetaan hyppäämään irti seinästä. Jos pakotusta ei tehtäisi, voisi pelaaja hyppiä suoraan seinää pitkin seinää ylös.

Impulssin laskemisen lisäksi hypyn suunnittelutilassa piirretään näytölle pisteitä, jotka näyttävät sen hetkisen hypyn lentoradan. Pisteiden sijaintia päivitetään aina samalla, kun hypyn voima muuttuu. Lentoradan pisteiden paikat lasketaan kaavoilla 2 ja 3. Pisteiden paikkojen laskussa otetaan huomioon samat rajoitukset kuin hypylläkin on, jotta saadaan piirrettyä todenmukainen lentorata.

$$F_x(t) = x_0 + v_x * \Delta t \qquad (2)$$

$F_x(t)$ on x-akselin suuntainen voima ajan suhteen
$x_0$ on hahmon x-akselin suuntainen paikka ajan hetkellä t
$v_x$ on hahmon x-akselin suuntainen nopeus
$\Delta t$ on ajan muutos

$$F_y(t) = y_0 + v_y * \Delta t + \frac{1}{2} * g * \Delta t^2 \qquad (3)$$

$F_y(t)$ on y-akselin suuntainen voima ajan suhteen
$y_0$ on hahmon y-akselin suuntainen paikka ajan hetkellä t
$v_y$ on hahmon y-akselin suuntainen nopeus
$g$ on painovoima
$\Delta t$ on ajan muutos

Kun pelaaja on tyytyväinen hypyn voimaan, hän päästää sormella irti näytöstä. Tällöin kutsutaan jump()-funktiota. Tämä vastaanottaa parametrina suunnittelussa lasketun impulssin. Jotta hyppy menisi täysin lentoradan laskukaavan mukaisesti, täytyy impulssi kertoa vielä pelihahmon massalla kaavojen 4 ja 5 tavalla. Tämä on muuten sama kaava kuin tavallisessa liikkumisessa, mutta koska voima annetaan impulssina, voidaan aikamuuttuja jättää pois kaavasta. Impulssia annettaessa Box2D osaa itse ottaa huomioon ajan.

$$F_x = m * v_x \qquad (4)$$

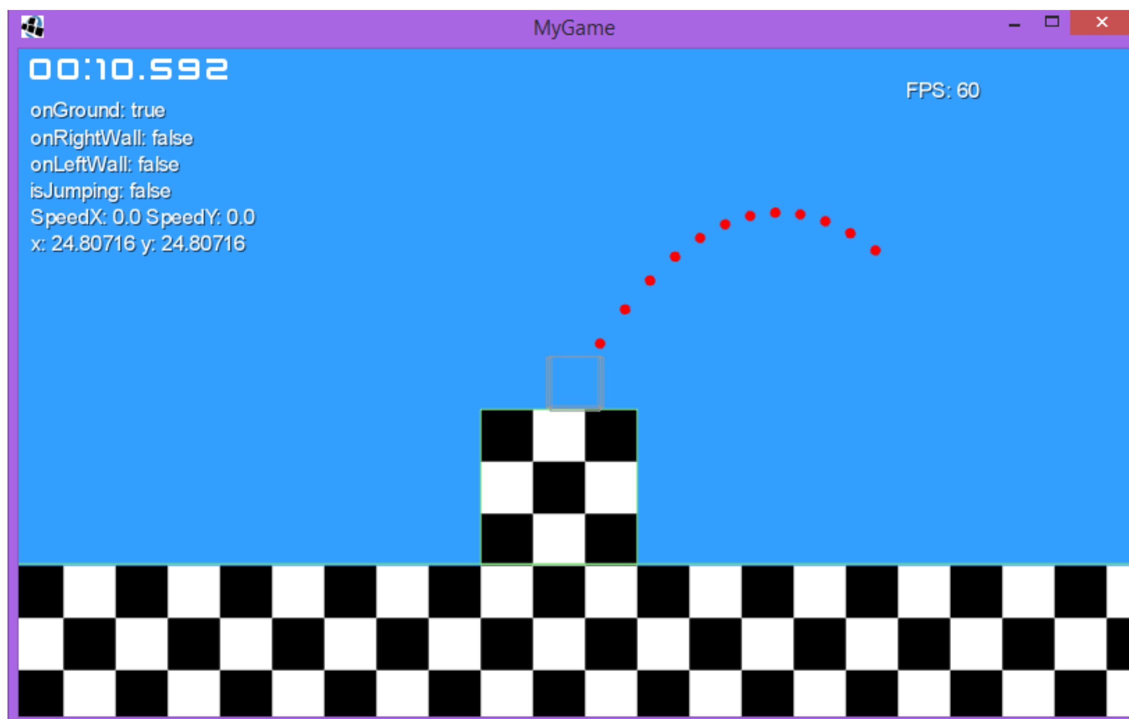
$F_x$ on x-suuntainen voima
$m$ on pelihahmon massa
$v_x$ on x-suuntainen nopeus

$$F_y = m * v_y \quad (5)$$

$F_y$ on y-suuntainen voima
$m$ on pelihahmon massa
$v_y$ on y-suuntainen nopeus

Kun hyppy on suoritettu, alkaa update()-metodi pyörittämään checkIfJumpIsDone()-funktiota. Tässä funktiossa tutkitaan, onko hyppy jo loppunut. Hyppy todetaan loppuneeksi, jos hahmo osuu jompaankumpaan seinään tai lattiaan. Tämän lisäksi hypyllä on pieni viive, jonka ajan sen pitää vähintään kestää. Viive on lisätty sitä varten, että pelihahmo ei irtoa välittömästi lattiasta, jolloin saatetaan ehtiä tarkistamaan, että hyppy on jo suoritettu, ennen kuin pelihahmo on edes ehtinyt irrota lattiasta. Tämä mahdollistaisi hyppäämisen ilmassa. Kun hyppy on loppunut, siirtyy update()-funktio takaisin suorittamaan move()-funktiota.

On erittäin tärkeää, että tasohyppelypelin liikkuminen tuntuu hyvältä ja ohjautuu hyvin. Tästä syystä peliin rakennettiin heti alkuun testiympäristö, jossa pystyy testaamaan esimerkiksi, millaisilla hypyn maksimivoimien arvoilla pelihahmo hyppää miten korkealle, miten pitkälle pelaaja voi parhaimmillaan hypätä sekä millainen vauhti on sopiva tavallisessa liikkumisessa. Sen avulla pystyy myös selvittämään hyvin ilmaantuneita virheitä ja varmistamaan, että liikkuminen olisi mahdollisimman virheetöntä. Kuvassa 9 on kuvakaappaus testiympäristöstä.




Kuva 9. Liikkumista varten rakennettu testiympäristö, jossa voidaan hioa liikkeen arvot sopiviksi.

### 3.3.3 Tason aloituspaikka ja läpäiseminen

Tason alkupiste sekä maalialue lisätään karttaan Tiled-editorissa. Kuten tason Box2D-elementit lisätään nekin omalle objektitasolleen. Lähtöpiste lisätään ympyränmuotoisena objektina, joka on nimeltään ”Spawnpoint”. Kun taso ladataan peliin, haetaan kyseinen ympyrä ja palautetaan objektin keskipiste aloituspaikaksi ja luodaan pelihahmo tähän pisteeseen. Maalialue lisätään samalle objektitasolle kuin alkupiste. Se on nelikulmion muotoinen objekti nimeltään ”Goal”. Nelikulmion data parsitaan ja siitä luodaan staattinen Box2D-objekti. Kun kontaktikuuntelija huomaa, että pelaaja osuu maaliin, asetetaan Player-luokassa lippu, joka kertoo, että taso on voitettu. Tällöin lopetetaan pelimaailman päivitys, tallennetaan tarvittaessa pelitilanne ja näytetään voittoruutu. Jos tmx-dataa parsittaessa huomataan, että alkupistettä ja maalialuetta ei löydy luodusta tasosta, heitetään virhe, jossa kerrotaan, että nämä pitää löytyä kartasta ja lopetetaan pelin suoritus.

### 3.3.4 Pelin häviäminen

Piikit toimivat tällä hetkellä pelin ainoina esteinä. Nekin lisätään Tiled-editorissa karttaan piirtämällä piikkien kuvien ympärille omalle objektitasolle polygoni objekti. Objektin ominaisuudet parsitaan ja niiden perusteella piikkien ympärille luodaan staattinen Box2D-

objekti. Kun pelaajan havaitaan törmänneen piikkeihin, asetetaan Player-luokasta löytyvä boolean ”alive” lippu falseksi. Tällöin lopetetaan pelihahmon piirtäminen ja piirretään hiukkasefekti kuvaamaan pelihahmon kuolemaa. Pelihahmon kuollessa peli todetaan hävityksi, joten lopetetaan pelimaailman päivittäminen ja siirrytään lopettamaan peliä. Kun peli lopetetaan, annetaan ensin hiukkasefektin valmistua ja sen jälkeen näytetään häviöruutu. Hiukkasefektin annetaan ensin valmistua, jotta häviöruutua ei piirretä hiukkasefektin päälle, jolloin pelaaja saattaisi hämmentyä siitä, miksi peli loppui.

## 3.4 Grafiikka

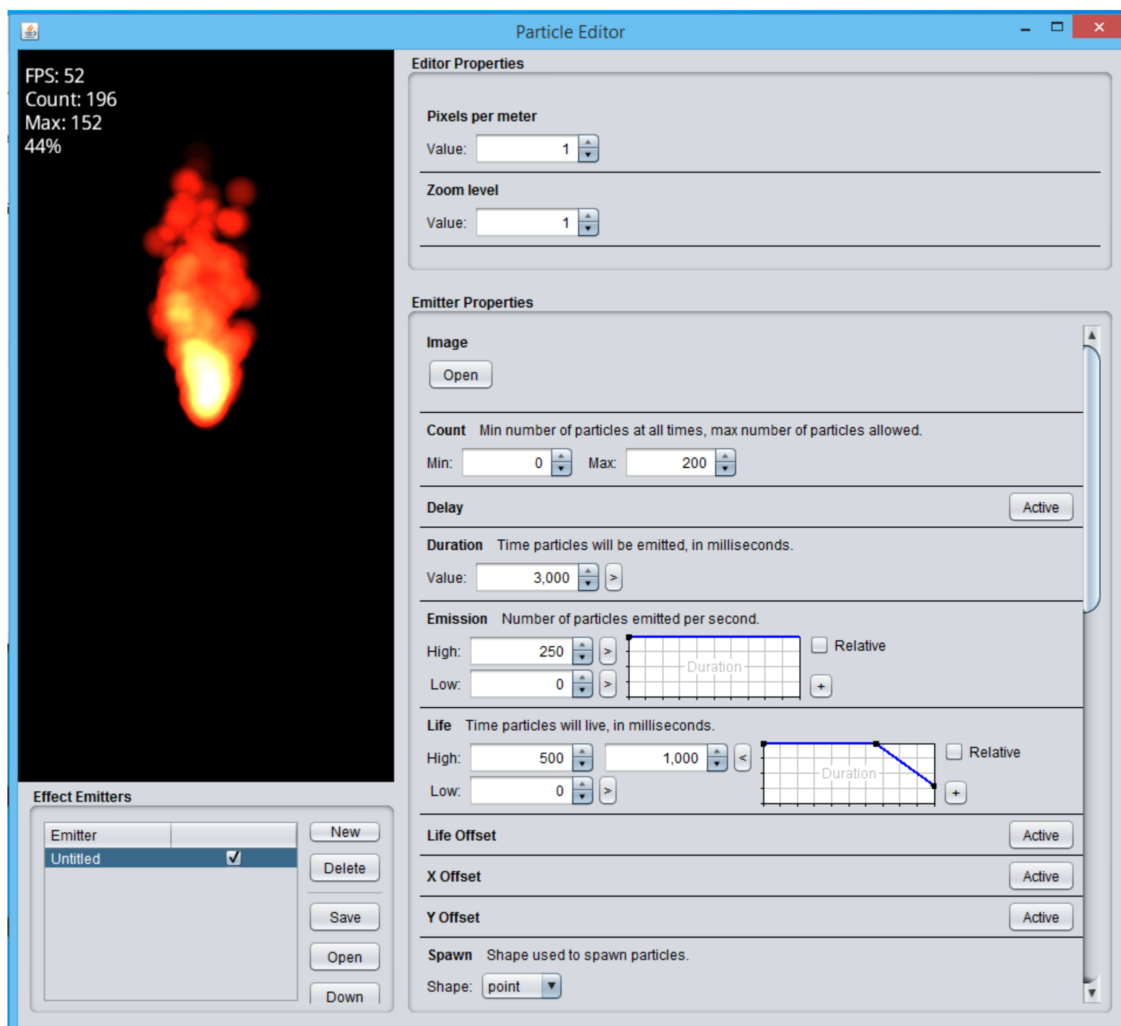
Kaikki pelin grafiikka on png-formaatissa. Libgdx tarjoaa Texture-luokan kuvatiedostojen dekoodaamiseen sekä lataamiseen näytönohjaimen muistille. Koska pienten yksittäisten kuvien piirtäminen erikseen on raskasta, ladataan esimerkiksi käyttöliittymän kuvat sekä pelihahmon kaikki kuvat yhdessä kuvassa. Tällaista kuvaa kutsutaan tekstuuriatlakseksi. Libgdx tarjoaa pienten kuvien yhdistämiseen tekstuuriatlakseksi oman työkalun. Se tarjoaa myös tällä työkalulla luodulle tekstuuriatlakselle apuluokan, jonka avulla atlaksesta voidaan hakea yksittäisiä kuvia tiedostonnimen perusteella, eikä tarvitse syöttää erikseen kuvan koordinaatteja ja kokoa hakeakseen kuvan. Kun kuva haetaan atlaksesta, palauttaa se TextureRegion-luokan objektin, joka on nimensä mukaisesti pala isompaa tekstuuria. Tätä voidaan käyttää samaan tapaan kuin yksittäistä Texture-luokan objektiakin. TextureRegionin voi siis asettaa esimerkiksi painikkeen kuvaksi tai niitä voidaan kasata useampia yhteen ja luoda animaatio.

Kuten Libgdx:n esittelystä käy ilmi, Libgdx käyttää grafiikan piirtämiseen OpenGL-rajapintaa, ja pelin kaikki piirtäminen tehdään sovellusmoduulin jatkuvasti kutsumassa render()-funktiossa. Piirtämisen optimoimiseksi Libgdx tarjoaa luokan SpriteBatch. Se kerää erän tekstuureja sekä tekstuurien sijainnit ja lähettää kerralla erän näytönohjaimelle piirrettäväksi. Tämä on huomattavasti tehokkaampaa kuin tekstuurien yksikerrallaan lähettäminen. Kaikki piirtokutsut tapahtuvat SpriteBatchin begin()- ja end()-funktioiden välissä. Kun kutsutaan begin(), SpriteBatch valmistellaan piirtoa varten ja aktivoidaan sekoitus sekä teksturointi. Tämän jälkeen voidaan suorittaa piirtokutsut, jolloin piirretään halutut TextureRegion-objektit. Piirtokutsujen jälkeen lopetetaan piirtäminen eli kutsutaan SpriteBatchin end()-funktiota. Tällöin poistetaan käytöstä sekoitus ja teksturointi. SpriteBatch helpottaa huomattavasti piirtämistä, koska se hoitaa konepellin alla

OpenGL:n säädöt, eikä piirtämistä varten ole pakko kutsua mitään OpenGL:n omia kutsuja suoraan lukuun ottamatta näytön tyhjennystä. Näytön tyhjennys pitää kutsua, jottei edellisen piirtokierroksen kuva jää näkyviin.

Pelin tasojen piirtämistä varten Libgdx tarjoaa TiledMapRenderer-rajapinnan. Tämän rajapinnan toteuttaa luokka OrthogonalTiledMapRenderer, joka koostaa kartan tmx-tiedoston perusteella ja tarkistaa, missä kohtaa minkäkin tekstuurin pitää sijaita. Luokka ottaa parametrina myös pelin SpriteBatch-ilmentymän ja hoitaa itse piirron aloituksen ja lopetuksen. Tason piirto voidaan tehdä siis yksinkertaisesti kutsumalla OrthogonalTiledMapRenderer.render().

Hiukkasefektit ovat tärkeä osa pelin tunnelman luomista. Myös näiden luomiseen Libgdx tarjoaa erillisen työkalun ja rajapinnat, joiden avulla hiukkasefektin toteutus on helppo tehdä. Kuvassa 9 näkyy hiukkasefektityökalun käyttöliittymä. Sen avulla voidaan helposti muokata hiukkasefektin ominaisuuksia, ja ominaisuudet näkyvät heti näytöllä. Kun efekti on valmis, tallennetaan sen tiedot tekstitiedostoon. Hiukkasefektin käsittelyyn löytyy ParticleEffect-luokka. Efekti ladataan load()-funktiolla, jolle annetaan parametrina tiedostokahva tekstitiedostoon sekä kansioon, jossa efektin kuva tai kuvat sijaitsevat. Jotta hiukkasefektin animaatio toimii, pitää se käynnistää funktiolla start(). Tämän jälkeen se voidaan piirtää yksinkertaisesti draw()-funktiolla, jolle annetaan SpriteBatch sekä aika, joka viime piirtämisestä on kulunut. Ajan perusteella ParticleEffect-luokka huolehtii itse efektin animaation päivityksestä piirtokutsua kutsuttaessa.




Kuva 10. Libgdx:n tarjoama työkalu hiukkasefektien luomiseen.

Pelimaailma on suurempi kuin mitä yhteen ruutuun mahtuu, joten pelistä pitää löytyä jonkinlainen kamerasysteemi. Pelissä käytetäänkin Libgdx:n tarjoamaa ortografista kameraa. Kamera hoitaa automaattisesti projektio- ja näkymämatriisien muutokset esimerkiksi zoomattaessa tai liikutettaessa kameraa. Näitä on luotu peliä varten kaksi erilaista, toinen itse peliä varten ja toinen käyttöliittymää varten. Käyttöliittymän kamera pidetään koko ajan samassa paikassa, ja sen näkymäportin koko on sama kuin näytön. Peliä varten luotu pääkamera asetetaan siten, että pelihahmo on sen keskellä, ja kamera seuraa koko ajan pelihahmoa. Tästä on kuitenkin poikkeustapaus. Jos pelihahmo on lähellä reunoja, ei kameraa siirretä siten, että pelaaja näkee pelimaailman reunojen yli, vaan rajataan kamera pelimaailman sisään. Näillä alueilla pelihahmo voi liikkua ilman, että kamera seuraa sitä. Heti, kun pelihahmo siirtyy tarpeeksi kauas reunasta, alkaa kamera taas seuraamaan pelaajaa. Pääkameran päivitys hoidetaan Play-luokan updateCamera()-funktiossa, jossa päivitetään sen sijaintia yllä olevien sääntöjen mukaan, ja kutsutaan

kameran funktiota update(), jolloin kamera laskee projektio- ja näkymämatriisit uudelleen annettujen arvojen kanssa.

Koska mobiililaitteita on nykyään todella paljon erilaisia, ja eri mobiililaitteiden resoluutiot vaihtelevat paljon, täytyy resoluutioiden vaihtelut ottaa huomioon. Mikäli ei haluta, että luodaan jokaiselle resoluutiolle omat erikokoiset tekstuurit, täytyy resoluution vaihtelu käsitellä jollain muulla tavalla. Tätä varten pelissä käytetään Libgdx:n tarjoamaa Viewport-luokkaa, joka hoitaa näkymäportin koon hallinnan. Pääkameralle sekä käyttöliittymän kameralle luodaan omat Viewport-ilmentymät. Käyttöliittymän näkymäportin kooksi asetetaan pelin käynnistävän laitteen resoluutio. Pääkameran näkymäportin kooksi asetetaan virtuaalinen resoluutio, jotta kamera pysyy samanlaisena laitteen resoluutiosta riippumatta. Virtuaalinen resoluutio on leveydeltään 480 ja korkeudeltaan 320. Jotta Viewport osaa hallita näytön koon oikein, pitää sovellusmoduulin kutsumassa resize()-funktiossa kutsua Viewport-luokan funktiota update(), jolle annetaan parametrina laitteen resoluution leveys ja korkeus. Tällöin Viewport skaalaa kuvan ja säätää siihen liitetyn kameran näkymäportin koon.

Libgdx tarjoaa useita erilaisia valmiita näkymäporttivaihtoehtoja. Oletuksena käytössä on ScreenViewport, joka ei tue virtuaalista resoluutiota, vaan sen koko vastaa aina ruudun kokoa eikä skaalausta tapahdu. Tämän lisäksi vaihtoehtoina on esimerkiksi rajata näkymäportin koko siten, että jos näytön resoluutio on liian iso, piirretään sivuille mustat alueet ja pidetään kuvan koko samana tai mahdollisesti venytetään kuva siten, että koko näyttö aina täyttyy. Itse päädyin käyttämään ExtendViewportia. Tämä asettaa näkymäkoon siten, että mitä isompi resoluutio laitteella on, sitä enemmän pelaaja näkee maailmasta. Tähän voisi lisätä myös maksimikoon, jonka jälkeen sivuilla näytetään mustat alueet, mutta tähän peliin se ei ole tarpeen.

Kameran ja näkymäportin laskennat pitää ottaa huomioon aina ennen piirtämistä, jotta ne vaikuttavat johonkin. Ennen piirtokutsuja ja SpriteBatchin begin()-komentoa pitää SpriteBatchin projektiomatriisiksi asettaa kameran projektiomatriisi. Tämän lisäksi pitää myös kutsua Viewportin metodi apply(), jotta asetetaan näkymäportin kooksi Viewportin laskema skaalattu koko. Ennen käyttöliittymän piirtokutsuja vaihdetaan projektiomatriisi ja näkymäportti käyttöliittymän omiksi versioiksi.

## 3.5 Pelitilan tallennus

Pelitilan tallennusta varten on luotu SaveManager-luokka. Se sisältää funktiot tiedoston lataamiseen ja tallentamiseen. Liitteessä 2 näkyy koko SaveManager-luokan toteutus. Sen konstruktorissa avataan ensin lokaali tiedostokahva tallennustiedostoon. Tiedostokahva avataan lokaalisti, koska kuten tiedostomoduulin selityksessä kappaleessa 2.2. kerrottaan on lokaaliin tiedostoon mahdollista kirjoittaa. Tallennustiedosto on JSON-muotoinen. Konstruktorissa tämä tallennustiedoston JSON luetaan Libgdx:n tarjoamaan ObjectMap-luokkaan, siten, että jokainen JSON:in sisältämä objekti lisätään omaksi avain-arvopariksi. Pelissä avaimena toimii tason nimi, jotka on yksinkertaistettu muotoon ”level#”, jossa # on kyseisen tason numero. Arvona käytetään luokkaa Level, joka toimii DTO:na. Luokka sisältää vain muuttujat tallennettavista tiedoista, sekä niiden funktiot tietojen hakuun ja asettamiseen.

Tallennustiedostoa ei ole koodattu millään salauksella, koska on huomattavasti helpompaa selvittää ongelmia, kun tallennustiedostoa voi muokata käsin ja sitä on selkeä lukea. Myöhemmin tallennustiedostoon voidaan lisätä esimerkiksi base64-koodaus, jolla estetään pelin helppo huijaaminen. Tätä vahvempaa salausta ei mielestäni kuitenkaan ole tarpeellista tehdä, koska pelissä ei ole tulosten vertailua eri pelaajien välillä, vaan pelaaja kilpailee vain itseään vastaan. Vahvemman salauksen luominen veisi aikaa, ja se pieni osa pelaajista, joka oikeasti haluavaa huijata pelissä, voi murtaa myös vahvemmat salaukset, jolloin salaukseen käytetty aika menisi hukkaan. Pieni salaus on kuitenkin tarpeen, jotta vältetään houkutus huijata pelissä, sillä ilman salausta huijaus on helppoa.

Tietojen lataaminen luokkiin tapahtuu loadDataValue()-funktiolla. Se ottaa parametrina avaimen sekä luokkatyypin, joka halutaan ladata, ja jos avaimella löytyy dataa, se palauttaa halutun luokan olion. Tallennus tapahtuu saveDataValue()-funktiolla. Tälle annetaan parametrina avain sekä olio, joka halutaan tallentaa. Tieto lisätään ObjectMappiin. Jos ObjectMapista löytyy jo avaimella olio, se korvataan. Muuten uusi tieto lisätään muiden joukkoon. Tämän jälkeen ObjectMap-kirjoitetaan kokonaan uudelleen save.json-tiedostoon.

Pelissä ei ole mitään kerättäviä bonuksia, eikä hahmolla ole erikoisia avattavia kykyjä. Pelin tasot ovat myös lyhyitä ja nopeasti läpäistävissä, joten pelaajan sijaintia tasossa ei

tarvitse tallentaa, vaan voidaan aina aloittaa alusta. Näistä syistä saadaan pelin tallennusjärjestelmä pidettyä hyvin yksinkertaisena. Ainoat tallennettavat tiedot ovat kentän läpäisy, onko tasoa mahdollista pelata, sekä läpäisyyn kulunut aika.

Pelin tallennus tapahtuu, kun pelaaja saavuttaa tason maalin. Tällöin kutsutaan Play-luokan funktiota saveGame(). Funktiossa tarkastetaan, onko pelaaja jo aikaisemmin läpäissyt tason, jos ei ole, asetetaan taso läpäistyksi. Tämän lisäksi asetetaan myös seuraava kenttä avatuksi. Vain parhaasta kentän läpäisyajasta pidetään kirjaa. SaveGame()-funktiossa tarkistetaan siis myös, onko pelaaja parantanut aikaa ja korvataan tarvittaessa aikaisempi paras aika uudella. Jos jompikumpi tason tiedoista on muuttunut, kutsutaan SaveManagerin saveDataValue()-funktiota ja tallennetaan tiedot, muuten ei tallenneta turhaan samoja tietoja.

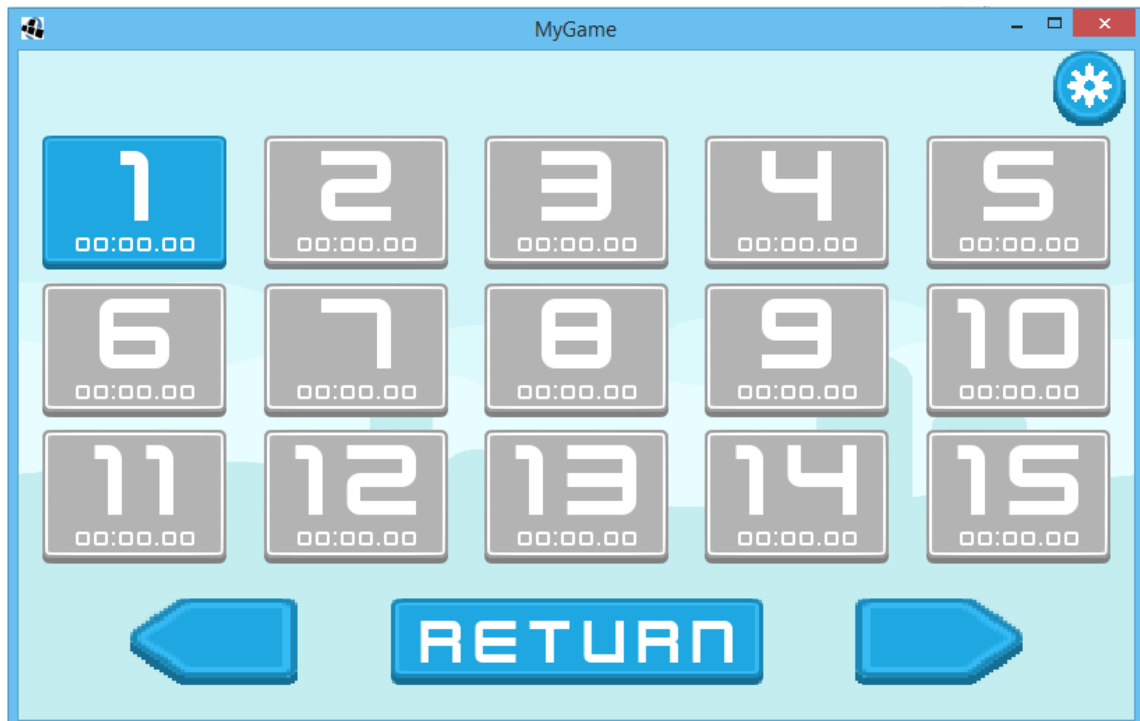
## 3.6 Valikot

Pelin valikot on toteutettu aikaisemmin esitellyn Scene2d UI -kirjaston avulla. Pelissä on kaksi päävalikkonäyttöä, jotka on toteutettu omiin luokkiinsa. Ensimmäinen päävalikkonäyttö aukeaa heti, kun peli käynnistyy.  Se sisältää vain painikkeet peliin jatkamiseen, pelin asetusnäytölle pääsyyn ja pelin lopetukseen. Koska asetukset on rajattu pois, ei painike tee vielä mitään.

Näyttö koostuu viidestä Actor-objektista, jotka on lisätty Stage-objektiin. Alimmaisena Stageen on lisätty Table-objekti eli taulukko, joka hoitaa elementtien yksinkertaisen asettelun. Se on kooltaan maksimissaan näytön kokoinen. Kuvassa 11 näkyvät tauluun lisätyt objektit omilla riveillään. Ylimmällä rivillä on Label-objekti, jolla otsikko on kirjoitettu. Seuraaviin kolmeen riviin on lisätty TextButton-objektit, jotka ovat painikkeita, joihin voi lisätä myös tekstiä. Jokaiseen painikkeeseen on lisätty oma kuuntelija, joka seuraa painalluksia. Otsikolle ja painikkeille luodaan painikkeiden koosta (jotka riippuvat näytön resoluutiosta) riippuvat fontit FreeType-lisäosan avulla. Kun play-painiketta painetaan, siirrytään tasonvalitsemisnäytölle. Ennen näytön vaihtoa suoritetaan kuvan himmennys action, joka kestää 0.3 sekuntia. Tällä saadaan näytönvaihto näyttämään vähän sulavammalta.

Kuva 11. Pelin päävalikkonäyttö.

Kuvassa 12 näkyvä tasonvalintanäyttö on hieman monimutkaisempi rakennelma kuin päävalikkonäyttö. Scene graph -tietorakenteen ylimmällä tasolla ovat TextButton-objektit, joita on jokaiselle pelin tasolle omat. Jokainen painike sisältää tekstinä kentän numeron, sekä nopeimman ajan, joka pelaajalta on kulunut tason suoritukseen. Mikäli tasoa ei vielä ole läpäisty, näytetään aika ”00:00:00”. Pelin alussa vain ensimmäisen tason painiketta on mahdollista painaa. Loput painikkeet ovat pois käytöstä. Kun pelaaja on läpäissyt tason, löytyy tallennustiedostosta tasolle tieto, että tasoa on mahdollista pelata ja painike avataan käyttöön. Tasonvalintapainikkeet on pakattu taulukkoon siten, että yhdessä taulukossa on maksimissaan 15 painiketta, ja ne on jaettu kolmeen riviin. Nämä taulukot on pakattu Stack-objektiin. Se asettelee taulukot päällekkäin eri tasoille. Näistä näytetään kerrallaan vain yksi taso.




Kuva 12. Pelin tasonvalintanäyttö.

Scene graph -tietorakenteen alimmaisella tasolla on päävalikkonäytön tapaan taulukko, joka on maksimissaan koko näytön kokoinen. Tähän taulukkoon liitetään Stack-objektin lisäksi, oikeaan yläkulmaan ImageButton-objekti, joka on painike, johon voidaan lisätä kuva. ImageButtonin olisi tarkoitus avata pelin asetusnäyttö, mutta tälläkään ei ole vielä mitään lisättyä toiminnollisuutta. Tämän lisäksi Stack-objektin alapuolelle lisätään Text-Button-objekti sekä kaksi Button-objektia. Button on peruspainike, jossa ei ole valmista paikkaa tekstille tai kuvalle.

TextButtonia painamalla siirrytään päävalikkonäytölle ja suoritetaan 0,3 sekunnin himmennys-action kuten näytölle tultaessakin. Nuolipainikkeita painamalla voidaan selata pelin tasoja. Nuolen painaminen laukaisee kaksi erilaista tapahtumaketjua. Yksi ketju siirtää näkyvän Stack-objektin tason näytön koon verran vasemmalle tai oikealle (riippuen siitä kumpaa nuolta painettiin) puolessa sekunnissa. Siirron jälkeen taso asetetaan piilotetuksi. Toinen ketju siirtää ensin piilotetun Stack-objektin tason välittömästi ulos näytöltä, joko vasemmalle tai oikealle puolelle. Tämän jälkeen se asetetaan näkyväksi ja siirretään keskelle näyttöä samalla puolen sekunnin viiveellä kuin aikaisempi taso vietiin näytöltä pois. Tällä ketjutuksella saadaan aikaiseksi animaatio, joka näyttää siltä kuin Stackin tasoja siirrettäisiin aina jompaankumpaan sivuun, vaikka ne ovat oikeasti vain

päällekkäin. Pelin tason käynnistäminen tapahtuu tasonvalintapainikkeesta. Tällöin siirrytään suorittamaan Play-luokkaa ja annetaan sille parametrina valitun tason Level-objekti.

### 3.7 Hud

Pelin hud on toteutettu omaan Hud-luokkaansa. Se luodaan Play-luokassa, samalla kun luodaan muutkin pelin objektit. Hud on myös toteutettu Scene2D UI -kirjaston avulla. Hudin rakenne on hyvin yksinkertainen. Se on vain Stage, joka piirretään pelin päälle. Stageen on liitetty näytön kokoinen taulukko, jonka vasempaan yläreunaan on lisätty Label-objekti, jossa näytetään tason suoritusaika. Aika näytetään muodossa 00:00.000, jossa kaksi ensimmäistä numeroa ovat minuutteja, seuraavat kaksi sekunteja ja viimeiset kolme millisekunteja. Aikaa päivitetään jokaisella pelin päivityskierroksella.

Myös voitto- ja häviöruudut on toteutettu Hud-luokkaan. Ne ovat molemmat Dialog-objekteja. Dialog on ponnahdusikkuna, joka sisältää valmiiksi kaksi taulukkoa. Ylempi taulukko on suunniteltu tekstiä varten. Alempi taulukko on tarkoitettu painikkeita varten. Voitto- ja häviöruutu näytetään kutsumalla Hud-luokan showEndingStatusDialog()-funktiota, jolle annetaan parametrina boolean. Parametri on true, jos pelaaja voitti tason, ja false, jos pelaaja hävisi tason. Mikäli pelaaja voitti tason, näytetään tekstinä onnittelu, tason suoritukseen kulunut aika ja pelaajan paras aika. Häviötapauksessa näytetään vain teksti ”kuolit, yritä uudelleen”. Dialog sisältää kolme painiketta. Yhdestä painikkeesta voidaan palata takaisin tasonvalintanäytölle. Toisesta painikkeesta voidaan käynnistää sama taso uudelleen. Kolmannesta painikkeesta voidaan siirtyä seuraavaan tasoon, mutta vain, jos seuraava taso on jo avattu.

## 4 Päätelmät

Itselleni ei jäänyt juurikaan pahaa sanottavaa Libgdx:stä. Se on hyvin monipuolinen sovelluskehys kaksiulotteisten pelien kehitykseen. Sen tarjoamien valmiiden korkean tason ratkaisujen avulla, on pelin kehitys nopeaa ja suhteellisen helppoa. Korkean tason ratkaisujen avulla myös ohjelmoijat, joilla ei ole paljon kokemusta pelikehityksestä, voivat helposti päästä sisään pelin kehitykseen Libgdx:llä. Oman pelini kehityksessä ei tullut tarpeeseen päästä käyttämään matalamman tason funktiota, mutta on hyvä, että Libgdx

tarjoaa ne. Tämän ansiosta vältetään kehittäjän ratkaisujen rajoittaminen. Pidin paljon siitä, että esimerkiksi Box2D-kirjasto on vain kevyesti kääritty Java-toteutus, jolloin ongelmiin pystyy etsimään esimerkiksi C++-kielen ratkaisuja ja ratkaisut on helppo soveltaa omaan Libgdx-projektiin.

Libgdx:n tarjoamat työkalut kuten tekstuuriatlasten tai hiukkasefektien luomiseen tarkoitetut sovellukset ovat hyvin käteviä. Koska ne on suunniteltu juuri Libgdx:ää varten, löytyy Libgdx:stä myös helppo tapa, jolla niillä luotuja tiedostoja voidaan käsitellä. Esimerkiksi yksinkertaisen hiukkasefektin luominen ja sen tuominen ohjelmaan oli helppoa, vaikka koko hiukkasefektikonsepti oli ennestään minulle tuntematon. Sovellus, josta kuitenkin eniten pidin, on projektin luomiseen tarkoitettu sovellus. Sen avulla on todella helppo luoda uusia Libgdx-projekteja ja sitä käyttämällä vältetään lähes kaikki kehitysympäristön asetusten säätäminen.

Koko Libgdx:n paras puoli on mielestäni se, että koska peli toimii sekä tietokoneella että mobiililaitteilla, voidaan pelin toteutusta testata suorittamalla peli suoraan tietokoneella. Näin testaukseen ei tarvitse käyttää emulaattoria, jonka käyttö voi välillä olla hyvin turhauttavaa, koska esimerkiksi sen käynnistäminen voi kestää useita minuutteja. Silloin, kun peliä halutaan testata mobiililaitteella, on pelin siirtäminen Android-puhelimelle hyvin yksinkertaista. Valitettavasti iOS-laitteelle pelin siirtäminen vaatii Mac-tietokoneen, jota itselläni ei ole saatavilla, joten en päässyt testaamaan, miten helposti peli kääntyy iOS-laitteille.

Libgdx on hyvin dokumentoitu, ja sen oma wiki kattaa suurimman osan sen tarjoamista ominaisuuksista. Wiki sisältää myös monia esimerkkiprojekteja, joista voi ottaa mallia erilaisten ominaisuuksien toteuttamiseen. Esimerkkiprojektit ovat kuitenkin välillä hieman liian monimutkaisia, jotta ne soveltuisivat hyvin opastukseen. Mikäli dokumentoinnista ei löydy ohjeita, on Internet pullollaan erilaisia Libgdx-oppaita, ja esimerkiksi Youtubesta löytyy paljon erilaisia video-oppaita pelin kehitykseen Libgdx:llä. Tämän lisäksi Libgdx:n foorumeilta, ainakin oman kokemukseni mukaan, saa helposti vastauksen erilaisiin kysymyksiin, joihin ei muualta onnistu vastausta kaivamaan.

## 5 Yhteenveto

Tämän insinöörityön tarkoituksena oli esitellä Libgdx-sovelluskehys, miten se toimii ja minkälaisten osien päälle se on kasattu. Esittely ei ole täydellisen kattava, mutta mielestäni toteutuksen kanssa kertoo hyvin siitä, millaista Libgdx:n kanssa on työskennellä sekä mitä Libgdx:llä on tarjota pelinkehitykseen. Libgdx on avoimen lähdekoodin projekti ja sillä on hyvin aktiivinen kehittäjäkunta, joten esimerkiksi lisäosia on huomattavasti enemmän kuin tässä työssä esitellyt muutamat. Lisäosien esittely onkin rajattu niihin, joita pelin toteutuksessa on käytetty.

Työn ohessa toteutettiin myös peli Libgdx-sovelluskehystä käyttämällä. Pelin tarkoituksena oli testata, miten ideoimani tapa tasohyppelypelin liikkumisessa toimiva mobiililaitteella sekä tutkia, ja esitellä, miten Libgdx toimii pelin kehityksessä. Pelin toteutus valmistui rajausten mukaiseen tavoitteeseen. Pelissä on toimiva liikkumismekaniikka, alkupiste, maali sekä piikkejä, joihin osuessa pelihahmo kuolee ja peli hävitään. Tämän lisäksi pelissä on toimiva käyttöliittymä. Toteutusta on siis jo mahdollista oikeasti pelata.

Mielestäni pelihahmon liikkumistapa toimi hyvin ja pelihahmon ohjaaminen on sopivan tarkkaa. Peli on myös toteutettu siten, että sitä on helppo laajentaa. Suunnitelmissani onkin, että peli jossain vaiheessa saadaan niin hyvään kuntoon, että se voidaan julkaista. Siihen pisteeseen pääsemiseen peli tarvitsee vielä paljon lisää ominaisuuksia ja kaikki väliaikaiset grafiikat pitää korvata oikeilla grafiikoilla.

## Lähteet


1 Bose Juwal, 12.2014. LibGDX Game Development Essentials, Packt Publishing.

2 Badlogic Games. Introduction, 2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Introduction>. Päivitetty 29.7.2015. Luettu 10.10.2015.

3 Badlogic Games. Libgdx Githubin README.md-tiedosto. Verkkodokumentti. <https://github.com/libgdx/libgdx> . Päivitetty 9.6.2015. Luettu 10.10.2015.

4 Nair, Suryakumar Balakrishnan, 1.2015. Learning LibGDX Game Development-Second Edition, Packt Publishing.

5 Cook James, 8.2015. LibGDX Game Development By Example, Packt Publishing.

6 Badlogic Games. The application framework, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/The-application-framework>. Päivitetty 27.7.2015. Luettu 11.10.2015

7 Badlogic Games. Graphics, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Graphics>. Päivitetty 12.1.2014. Luettu 11.10.2015.

8 Badlogic Games. File handling, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/File-handling>. Päivitetty 10.9.2015. Luettu 11.10.2015.

9 Badlogic Games. Input handling, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Input-handling>. Päivitetty 11.2.2014. Luettu 11.10.2015.

10 Badlogic Games. Sound effects, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Sound-effects>. Päivitetty 14.8.2014. Luettu 15.10.2015.

11 Badlogic Games. Streaming music, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Streaming-music>. Päivitetty 14.9.2013. Luettu 15.10.2015.

12 Networking, 12.1.2014. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Networking>. Päivitetty 14.5.2015. Luettu 15.10.2015.

13 Badlogic Games. The life cycle, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/The-life-cycle>. Päivitetty 12.3.2015. Luettu 11.10.2015.

14 Scene graph. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Scene_graph>. Päivitetty 19.7.2015. Luettu 18.10.2015.

15 Badlogic Games. Scene2d, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Scene2d>. Päivitetty 8.6.2015. Luettu 18.10.2015.

16 Scene2d.ui, 23.12.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Scene2d.ui>. Päivitetty 13.9.2015. Luettu 18.10.2015.

17 Badlogic Games. Skin, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Skin#overview>. Päivitetty 13.4.2015. Luettu 2.11.2015.

18 Badlogic Games. Box2d, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Box2d>. Päivitetty 22.10.2015. Luettu 24.10.2015.

19 Box2D C++ tutorials - Bodies. Verkkodokumentti. <http://www.iforce2d.net/b2dtut/bodies>. Päivitetty 14.7.2013. Luettu 24.10.2015.

20 Box2D C++ tutorials - Fixtures. Verkkodokumentti. <http://www.iforce2d.net/b2dtut/fixtures>. Päivitetty 14.7.2013. Luettu 24.10.2015.

21 Box2D C++ tutorials – Sensors. Verkkodokumentti. <http://www.iforce2d.net/b2dtut/sensors>. Päivitetty 14.7.2013. Luettu 24.10.2015.

22 Box2D C++ tutorials - Anatomy of a collision. Verkkodokumentti. <http://www.iforce2d.net/b2dtut/collision-anatomy>. Päivitetty 7.5.2014. Luettu 24.10.2015.

23 Badlogic Games. Bitmap fonts, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Bitmap-fonts>. Päivitetty 29.10.2015. Luettu 1.11.2015.

24 Badlogic Games. Gdx freetype, 14.9.2013. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Gdx-freetype>. Päivitetty 6.7.2015. Luettu 1.11.2015.

25 Setting up your Development Environment, 26.3.2014. Verkkodokumentti. <https://github.com/libgdx/libgdx/wiki/Setting-up-your-Development-Environment-%28Eclipse%2C-Intellij-IDEA%2C-NetBeans%29>. Päivitetty 21.10.2015. Luettu 25.10.2015.

## Libgdx-projektin rakenne

### HTML5-projekti

- Platformer-html
  - src
    - com.jndm.game
    - com.jndm.game.client
      - HtmlLauncher.java
  - JRE System Library [jre1.8.0_45]
  - Gradle Dependencies (persisted)
  - GWT SDK [GWT - 2.6.0]
  - build
  - war
  - webapp
  - build.gradle

### Työpöytä-projekti

- Platformer-desktop
  - src
    - com.jndm.game.desktop
      - DesktopLauncher.java
  - JRE System Library [jre1.8.0_45]
  - Gradle Dependencies (persisted)
  - assets
  - build
  - build.gradle

### iOS-projekti

- Platformer-ios
  - src
    - com.jndm.game
      - IOSLauncher.java
  - JRE System Library [jre1.8.0_45]
  - Gradle Dependencies (persisted)
  - build
  - data
  - build.gradle
  - Info.plist.xml
  - robovm.properties
  - robovm.xml

### Android-projekti:

- Platformer-android
  - src
    - com.jndm.game.android
      - AndroidLauncher.java
  - Gradle Dependencies (persisted)
  - Android 5.1.1
  - Android Private Libraries
  - gen [Generated Java Files]
  - assets
  - bin
  - build
  - libs
  - res
  - AndroidManifest.xml
  - build.gradle
  - ic_launcher-web.png
  - proguard-project.txt
  - project.properties

### Pelin koodia varten luotu projekti:

- Platformer-core
  - src
    - com.jndm.game
      - MyGame.java
    - com.jndm.game.gui
    - com.jndm.game.handlers
    - com.jndm.game.resources
    - com.jndm.game.saving
    - com.jndm.game.screens
    - com.jndm.game.utils
    - Game.gwt.xml
  - JRE System Library [jre1.8.0_45]
  - Gradle Dependencies (persisted)
  - build
  - build.gradle

## SaveManager-luokan toteutus

```
public class SaveManager {
    private FileHandle savefile;
    private Save save;

    public SaveManager() {
        savefile = Gdx.files.local("bin/save.json");
        save = getSave();
    }

    public static class Save {
        public ObjectMap<String, Object> data = new ObjectMap<String, Object>();
    }

    private Save getSave() {
        Save save = new Save();
        if (savefile.exists()) {
            Json json = new Json();
            save = json.fromJson(Save.class, savefile.readString());
        }
        return save;
    }

    public void saveToJson() {
        Json json = new Json();
        json.setOutputType(OutputType.json);
        savefile.writeString(json.prettyPrint(save), false);
    }

    @SuppressWarnings({ "unchecked", "rawtypes" })
    public <T> T loadDataValue(String key, Class type){
        if(save.data.containsKey(key)) {
            return (T) save.data.get(key);
        } else {
            return null;
        }
    }

    public void saveDataValue(String key, Object object){
        save.data.put(key, object);
        saveToJson();
    }

    public ObjectMap<String, Object> getAllData(){
        return save.data;
    }

}
```