Saimaa University of Applied Sciences
Technology Lappeenranta
Degree programme in Information Technology
Information System Development

Joonas Pirttiaho

Cross-platform mobile game development

Thesis 2014

Tiivistelmä

Joonas Pirttiaho
Järjestelmäriippumaton mobiilipelikehitys, 30 sivua, 1 liite
Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikan koulutusohjelma
Tietojärjestelmien kehitys
Opinnäytetyö 2014
Ohjaaja: lehtori Mikko Huhtanen, Saimaan ammattikorkeakoulu

Opinnäytetyössä tutkittiin miten on mahdollista kehittää applikaatioita tai pelejä järjestelmäriippumattomasti yleisimmille mobiilikäyttöjärjestelmille. Opinnäytetyön kirjoitushetkellä käytetyimmät mobiilikäyttöjärjestelmät olivat Googlen Android ja Applen iOS.

Opinnäytetyö aloitettiin selvittämällä erilaisia tekniikoita, joilla on mahdollista kehittää mobiiliapplikaatioita tai -pelejä järjestelmäriippumattomasti. Erilaiset vaihtoehdot käydään läpi ja Libgdx-ohjelmistokirjastoa esitellään tarkemmin sekä toteutetaan esimerkki projekti kyseisellä ohjelmistokirjastolla.

Lopputuloksena valittuja kehitystyökaluja hyödyntämällä oli mahdollista toteuttaa mobiilipeli Android- ja iOS-käyttöjärjestelmille. Opinnäytetyössä esitellään myös kehitysympäristö työkaluineen sekä esimerkki projektin asennus oikealle laitteelle.

Avainsanat: mobiilipelien kehitys, järjestelmäriippumaton, Libgdx

Abstract

Joonas Pirttiaho
Cross-platform mobile game development, 30 pages, 1 appendix
Saimaa University of Applied Sciences
Technology Lappeenranta
Degree Programme in Information Technology
Information System Development
Bachelor´s Thesis 2014
Instructor: Senior Lecturer Mikko Huhtanen, Saimaa University of Applied Sciences

The main purpose of this thesis was to find and learn about a framework for cross-platform mobile game development for modern mobile operating systems. The target mobile platforms were Google's Android and Apple's iOS.

The thesis was started by finding out different types of solutions for cross-platform application and game development for mobile devices. Different types of solutions should be briefly described and the found solution for cross-platform development should be more extensively covered.

As the result of this thesis the found types for cross-platform development options are listed and one framework, Libgdx, is more extensively covered from the required development environment to the stage of deploying the game to an actual device.

Keywords: mobile game development, cross-platform, Libgdx

Table of contents

Terms and abbreviations

ADT	Android Development Tools is an Eclipse plugin for supporting Android development
Android	Google's mobile operating system
API	Application Programming Interface
App Store	Apple's digital distribution platform
Apple	Apple Inc. produces consumer electronics and develops operating systems
Eclipse	Open source IDE
Google	Google Corporation develops Google search engine and the Android operating system
Google Play	Google's digital distribution platform
HTML5	HyperText Markup Language 5 is used to create modern websites and mobile applications
IDE	Integrated Development Environment
iOS	Apple's mobile operating system
Java	Programming language
Libgdx	Game development framework
Platform	Operating system, hardware or combination of both
Plugin	Software component that brings new features to existing software
RoboVM	Software that translates Java bytecode into native code for iOS

1 Introduction

Nowadays mobile applications and games are used on a broad range of different kind of devices. Devices are from a variety of manufacturers, come in different sizes and hardware components. Even the operating system and its version can vary from one device to another. Therefore when developing an application or a game the developer needs to plan what are the target platforms and minimum device requirements.

In the beginning of the development of an application or a game the developer chooses which technologies are going to be used for the development. The chosen technology can consist of programming languages, software development kits, third party software libraries and frameworks. The chosen technologies usually limit the available target platforms. Typically the application is created for one platform and can be ported to another platform by recreating parts or the whole application with different technologies. The goal of porting is to be able to run the application in a new platform which supports the new technologies. Developing for a variety of platforms can be very time consuming, if the application porting requires a lot work. For commercial projects this means extra costs but also broadens the customer base.

The main goal of this thesis is to explore the solutions for cross-platform development for mobile games. With one framework an example game is created from the beginning to a stage of deploying to a live device. The target platforms for the game are Google's Android, Apple's iOS and the desktop platform. In this thesis the desktop platform consists of Microsoft's Windows, Apple's Mac OS X and Linux based operating systems.

2 Cross-platform development

Creating graphical user-interface objects, drawing pictures or controlling the device's camera is done by using specific methods described in the platform's API (Application Programming Interface). Each platform has its own API, so that an application which has been created for a specific platform cannot be installed on a different platform because the APIs are different. Usually the supported

programming languages and SDKs (Software Development Kits) are also different between platforms.

Because consumers use devices with different operating systems, in order to reach the largest possible audience for an application or a game it should be able to run on as many devices as possible. Cross-platform applications usually provide identical (or nearly identical) functionality on each platform although the graphical user-interface can be different with each supported platform due to different design guidelines for individual platforms. Mobile application development for multiple platforms can be roughly divided into the following techniques.

2.1 Native code for each platform

Different native codebase for each platform of the same application is created simultaneously or an existing application from a specific platform can be ported to another platform. Porting is done by modifying the existing application, so that it is able to be run on a new target platform. Further updates to the application require that all the projects are updated separately with their own platform supported technologies.

Each platform has its own supported IDEs (Integrated Development Environments), programming languages and SDKs. Some IDEs or programming languages can also be used with different platforms. For example, Eclipse IDE supports Windows, Mac OS X and Linux based operating systems but Apple's Xcode IDE supports only their own OS X.

2.2 HTML5 based technologies

HTML5 (HyperText Markup Language) based mobile development technologies relay on HTML and JavaScript programming languages, which are commonly used with website development. The HTML5 based application operates on the platforms web runtime, also known as a web view, or it can be run on the web browser the same way as a normal website would. Packaging the HTML5 application in to a native binary allows the application to be distributed, installed and launched in the same way as a native application on any platform. The applica-

tion can control the device's capabilities and be displayed in a full-screen view. (Intel Developer Zone, Building Cross-Platform Apps with HTML5.)

Accessing devices capabilities is done by using an API provided by a HTML5 mobile application framework. Some of these frameworks are Titanium Appcelerator, Sencha Touch and Kendo UI. The frameworks also provide the possibility to build the application in to a native binary for each supported platform. (GAJOTRES, Top 7 mobile application HTML5 frameworks.)

2.3 Cross-platform compilers

These types of cross-platform development technologies offer the developer a unified API and a single programming language for each target platform. The application code is created once and then it is possible to compile the application for each target platform separately.

For example, Xamarin framework allows the developer to create the application in C# programming language and then build the application for Android, iOS and Windows Phone mobile devices as a native application (Xamarin, Tour). By using Libgdx framework games are programmed with Java and the projects can be built for Android and iOS devices as native games. Where Xamarin is built for mobile application development, Libgdx is focused on mobile game development.

3 Libgdx

Libgdx is an open source game development framework, which provides a unified API to target supported platforms. The code is written once and the application can be ported to all platforms without any platform specific modifications. Writing platform specific code is also supported if necessary. Libgdx has been developed in Java, but the performance critical parts are written in C and C++ due to higher level of performance. (Libgdx, Goals and Features)

Supported target platforms for Libgdx are Windows, Linux based operating systems, Mac OS X, Android, iOS, BlackBerry and HTML5. Libgdx applications are written in Java and the graphics are drawn by using OpenGL via OpenGL ES

1.x or 2.0 interfaces. For game development the framework includes common game related APIs to ease the development such as rendering text, building user-interfaces and playing sounds or music. (Libgdx Developer's Guide, Introduction.)

The development of a Libgdx based game is done on the desktop by writing the code with an IDE. The game can also be run and debugged on the desktop. But because of computers usually are a lot more powerful than mobile devices the application should be periodically tested on different mobile devices, so that the performance of the game can be verified.

3.1 Supported platforms

3.1.1 Android

Android is an operating system based on the Linux kernel. Primarily Android is used on mobile devices such as smartphones and tablet computers. Applications for Android are most commonly developed in Java programming language with Android software development kit (SDK). Android SDK includes development tools such as a debugger, necessary software libraries, a device emulator and sample code with tutorials. Applications operating on Android are run on the Dalvik Virtual Machine as Dalvik bytecode compiled from Java. (Android Developer, Android the world's most popular mobile platform.)

3.1.2 iOS

The operating system used in Apple's mobile devices is named iOS. The first version of iOS was released with the first iPhone in 2007. Contrary to Android, iOS is not available to be used by any other device manufacturer.

Mobile application development for iOS devices is possible with an Intel based Macintosh computer running Mac OS X operating system with the iOS SDK. Deploying the application also requires Xcode IDE, since it comes with many necessary tools such as the Application Loader for uploading the finished application to the App Store. Native iOS applications are written in Objective C programming language. (iOS Developer Library, About iOS App Programming.)

3.1.3 Desktop

Supported desktop platforms are Windows, Mac OS X and Linux based operating systems. The desktop platform is used for creating and debugging the game in an IDE. The desktop platform differs from the mobile platform with mostly different hardware. For example the desktop environment usually has a very large screen compared to a mobile device but on the downside the desktop environment does not have GPS positioning, accelerometer or possibility to vibrate the computer like a smartphone would. Nevertheless, the desktop can fully be one of the game's target platforms or even the only one.

4 Java programming language

Java is a pure object-orientated programming language, that was evolved from a language named Oak. The main idea of the language was to be platform independent, so it can run with different devices and platforms. The syntax for Java is similar to the popular C++ programming language. Java is used in Libgdx, but it is also used with applets in websites, electronic devices, desktop and server applications. (History of Java programming language, History.)

Native applications targeted for a specific platform are run on the native operating system whereas Java compilers produce platform independent byte code which is operating in Java Virtual Machine (JVM) on each of the supported platforms. Therefore Java applications are written once and deployed to all platforms with the same code. (Groups.enging.umd.emich.edu, Java The Programming Language.)

5 Creating the Libgdx example project

The development can be done with Windows, Mac OS X or with a Linux based operating system. Deploying the game to an iOS simulator or on to an actual iOS device can only be done in Mac OS X due to Apple's policy not to support any other operating system for development besides their own Mac OS X. (Libgdx Developer's Guide, Prerequisites.)

For example, the game can be created in Windows operating system and the iOS deployment can be done by importing the game project in to a Mac OS X development environment. Vice versa the whole development process and deploying to all platforms can be done with using only Mac OS X. Libgdx projects can also be developed with different IDEs than mentioned in this thesis, for example Android Studio or IntelliJ IDEA can also be used.

5.1 Required software

For targeting Android, desktop and HTML5 platforms the following software needs to be installed for the development machine.

- Eclipse IDE and Eclipse ADT (Android Development Tools) plugin
- JDK (Java Development Kit)
- Android SDK
- Google Web Toolkit (explicitly for the HTML5 project)

Additionally for targeting Apple's iOS platform Xcode IDE and RoboVM Eclipse plugin must be installed on a development machine running Mac OS X. (Libgdx Developer's Guide, Prerequisites).

5.2 Creating the Libgdx project

Libgdx projects can be created with an application called Libgdx Setup (figure 1), which was developed by Aurelion Ribon. Libgdx Setup downloads the latest version of Libgdx, creates the folders for each platform project and links the projects correctly to be used with Eclipse. The created projects can also be updated in the future with the newest version Libgdx framework by using the update functionality. (Aurelion Ribon's dev Blog, Libgdx Project setup v3.0.0!)

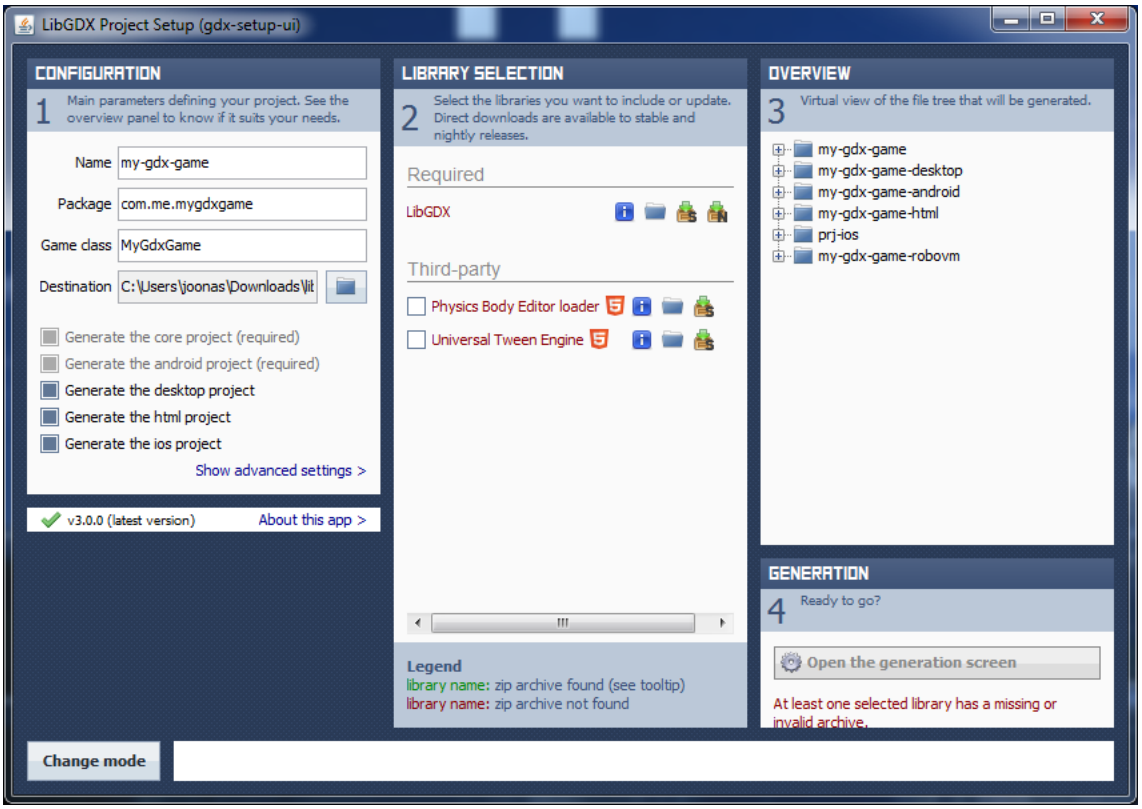

Figure 1 LibGDX Project Setup

5.2.1 Main project

The main project contains all the code for the game, except the starter classes, platform specific libraries and optional third-party SDKs for each platform. For example, the Android project can utilize Google Play's In-app Billing service or the iOS version can use the Apple's Game Center. All the other projects are linked to this main project.

All the application code will be written in the src folder as java files as shown in figure 2. Java code files are typically organized in Java packages by classes belonging to the same category or by providing similar functionality. Libgdx library files are located in the libs folder. Other platform specific libraries are located within the respective projects libs folder.

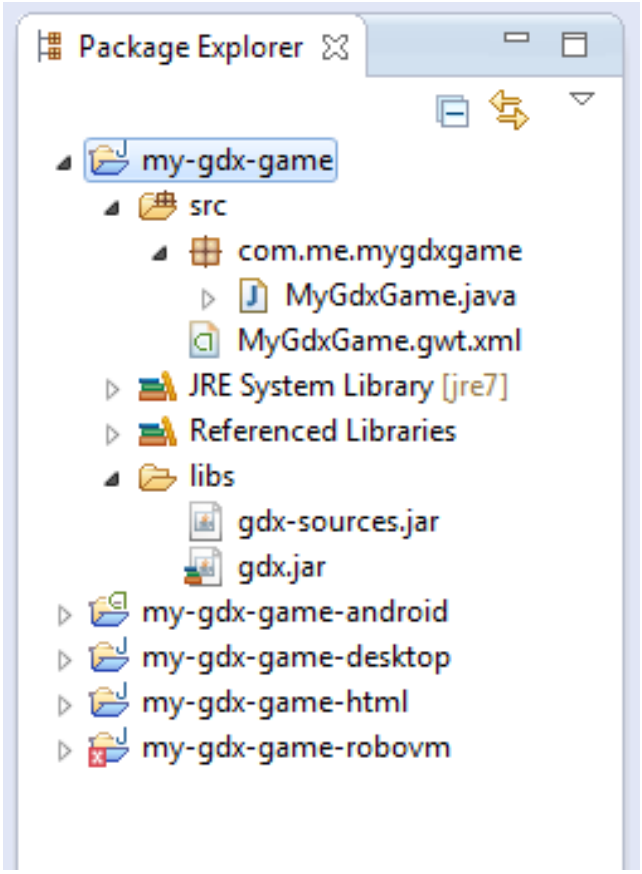

Figure 2 Tree view of the main project

5.2.2 Android project

Android project contains the starter class MainActivity.java in Java package com.me.mydgxgame and necessary files to run the Android application. Also all images, sounds and other resource files for the game are stored in Android project's (figure 3) assets folder.

Android project's tree view is similar to the main project's tree view. New additions are Android specific folders and files. All the Android project's settings such as the application name, version number and required permissions are located in the AndroidManifest.xml file. The assets folder will contain all the non-code related files such as images and sounds of the project.

ProGuard is a tool which can shrink, optimize and obfuscate Android applications code. Obfuscating is done by removing the unused code, or by renaming classes, fields and methods. The result is a smaller application size and it is more difficult to reverse engineer. Using ProGuard is optional but highly rec-

ommended. ProGuard can be enabled and configured from the proguard.cfg config file. (Android Developer, ProGuard.)

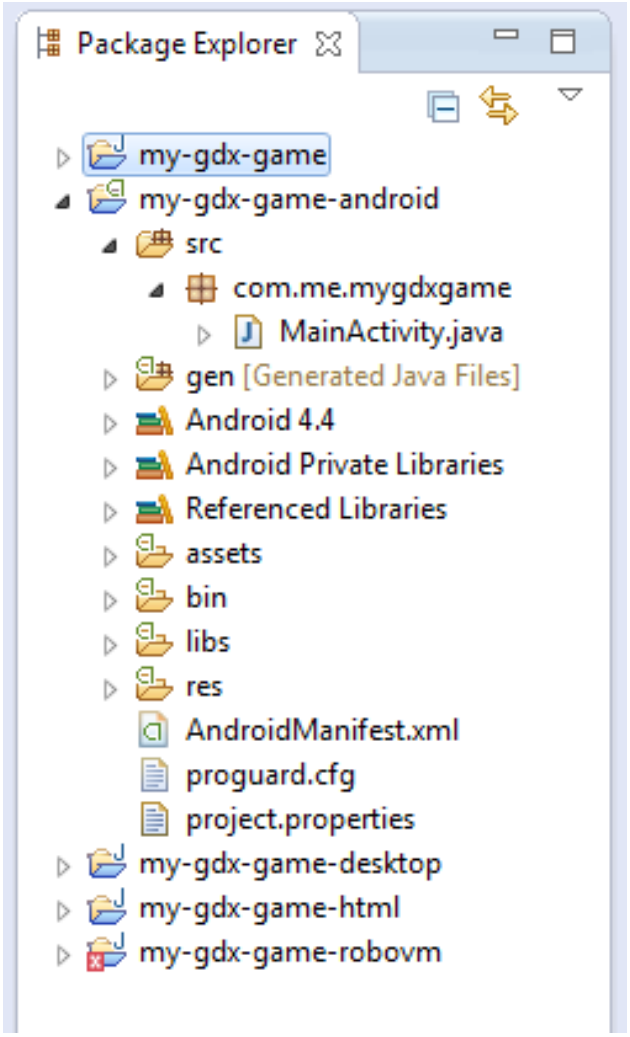

Figure 3 Tree view of the Android project

5.2.3 Desktop project

The desktop project contains the starter class Main.java found in Java package com.me.mygdxgame to run and debug the application on the desktop. The desktop project is linked to the Android project's assets folder so that it is unnecessary to keep duplicates of the resource files.

The desktop version of the project is executed (figure 4) by selecting Run As Java Application. Configuration, such as the game resolution, for the desktop can be changed from the starter class Main.java. This is very practical since it is

possible to test the game's appearance with different device resolutions, without needing the actual devices.

Figure 4 Libgdx application running on Windows 7

5.2.4 RoboVM project

RoboVM project contains the starter class to run the application on iOS devices and on the iOS simulators for iPad and iPhone. RoboVM project is linked to the Android project's assets folder like the desktop project. RoboVM project also contains configuration files for the iOS. This project is the only one which does not support debugging due to RoboVMs limitations at the time of writing. Figure 5 lists the RoboVM specific settings files Info.plist.xml and robovm.xml.

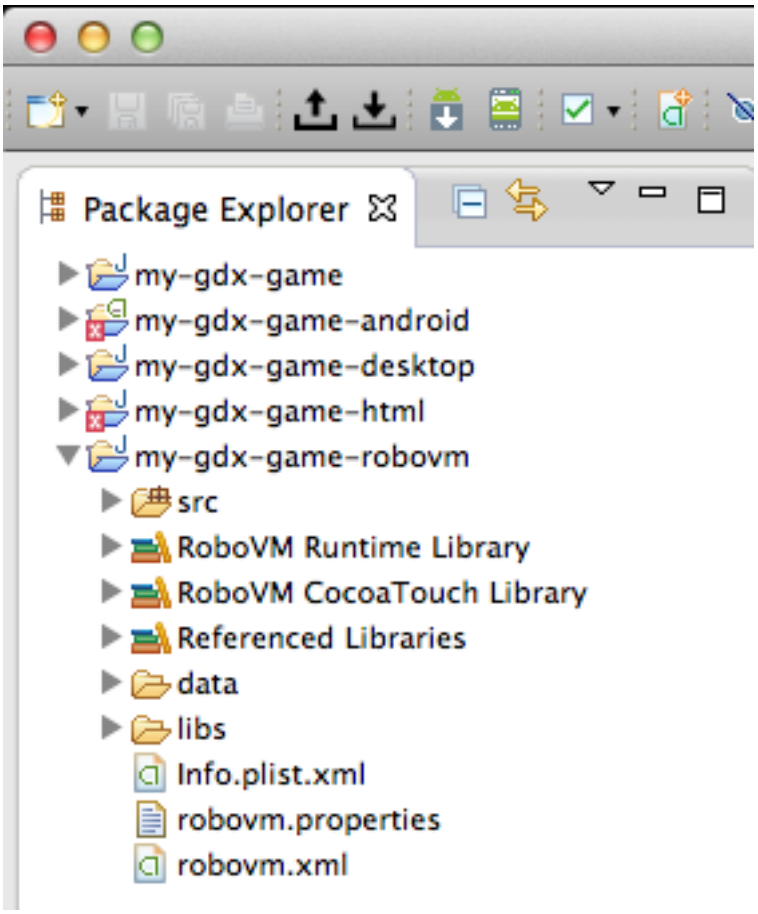

Figure 5 The projects in Eclipse on Mac OS X operating system

5.2.5 HTML5 project

HTML5 project contains the starter class to run the application as a HTML5 application and it is also linked to the Android project's assets folder. The project can be run in development mode where the actual Java code is run via GWT plugin and therefore allows debugging. Running in production mode requires a web server that supports serving compiled files.

Further information about HTML5 project and publishing HTML5 projects can be found from the Libgdx documentation.

5.3 Creating the example game

The following example is a simple game in which the player tries to click on falling red objects. When the player manages to click on an object, the object disappears and the player scores one point. If the object falls the down player los-

es one point from the score. Objects appear on top of the screen at one second intervals. The whole source code for the game is also attached as an appendix.

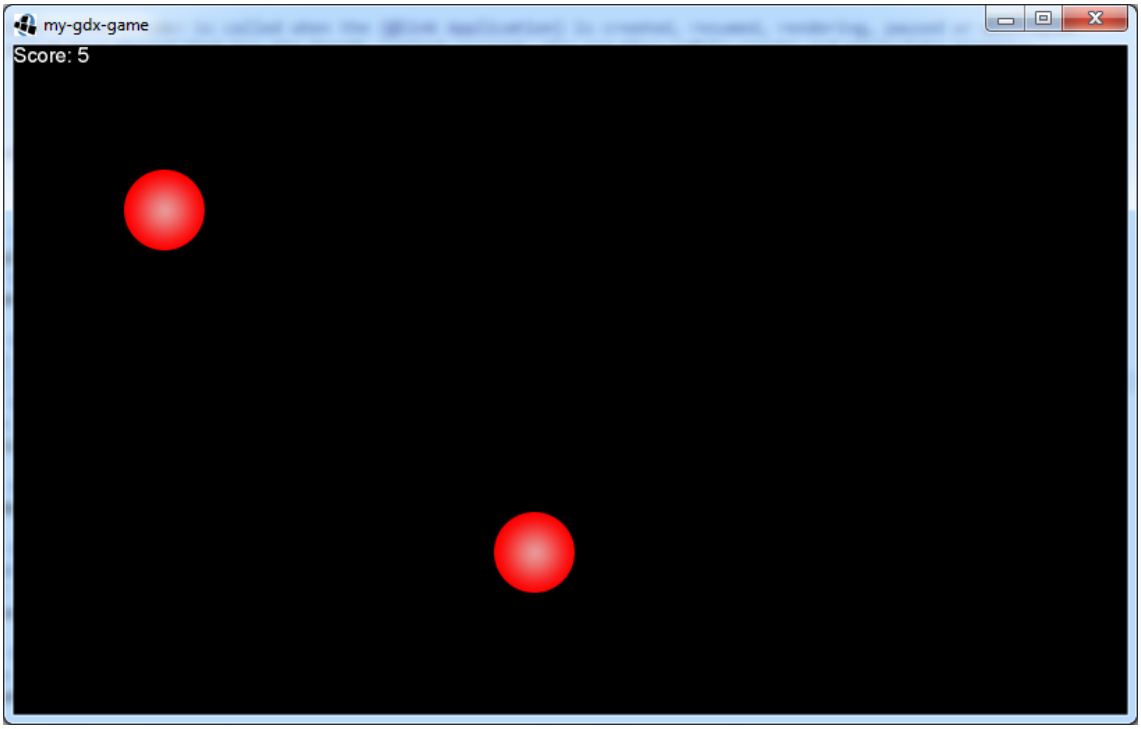

Figure 6 Running the example game on desktop

In figure 6 the game is running on Windows 7 platform. A mouse click on the game screen works in the same way when a user clicks on the screen on a touch screen mobile device.

5.3.1 The game loop

From a programming perspective the main standpoint of a game is the game loop from where the game is run. The game loop (figure 7) reacts to user input, updates everything within the game world and draws the game. Before the game loop starts usually game assets and other file input or output operations are performed in a loading screen. Loading the resources can also be done from the game loop as they are needed. A simple example of a game loop is in the following figure.

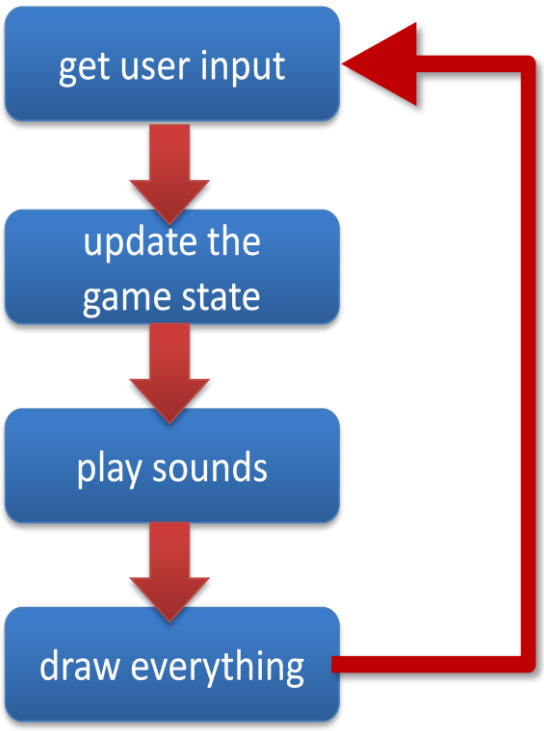

Figure 7 Simple example of a game loop

5.3.2 Variables and create method

The create method, shown in figure 8, is run only once when the game is launched. Typically setting up the game world, setting the player to the starting position and similar starting the game related actions are done in this method.

```
public class MyGdxGame implements ApplicationListener {
    private OrthographicCamera camera;
    private SpriteBatch batch;  // draws 2D textures

    private Array<Rectangle> objects; // holds the objects that player tries to click
    private Rectangle object = new Rectangle(); // objects are defined as Rectangles
    private Texture objectImage;
    private float time = 0;  // time tracking for making objects appear
    private float lastSpawnTime = 0;

    private int score;  // keeps the players score
    private BitmapFont font; // default font which comes with Libgdx

    private Vector3 touchCordinates = new Vector3();
    private int i;

    @Override
    public void create() {
        camera = new OrthographicCamera();
        camera.setToOrtho(false, 800, 480); // cameras viewport width and height
        batch = new SpriteBatch();

        // loads the image for the object from the assets folder
        objectImage = new Texture(Gdx.files.internal("object.png"));

        // clickable objects array
        objects = new Array<Rectangle>();

        // font to draw the score
        font = new BitmapFont();
    }
```

Figure 8 Setting up variables and the create method

5.3.3 Updating the game state

The updateGameState method (figure 9) calls the createNewObject method within one second intervals, moves the objects on the screen downwards and takes care of the object removal once an object has gone below the gameplay area.

```
private void updateGameState() {
    time += Gdx.graphics.getDeltaTime();

    // if a second has passed since the previous object was made
    if(time - lastSpawnTime > 1){
        createNewObject();
        lastSpawnTime = time;
    }
    // moves the objects downwards and removes one once
    // the object is below bottom of the screen
    for(i = 0; i < objects.size; i++){
        objects.get(i).y -= 250 * Gdx.graphics.getDeltaTime();
        if(objects.get(i).y + objectImage.getHeight() < 0){
            objects.removeIndex(i); // removes the object
            score--; // -1 to the score
        }
    }
}
```

Figure 9 Updating the state of the game

The falling objects are created in createNewObject method as Rectangles and stored in the objects array as shown in figure 10.

```
private void createNewObject() {
    // creates new object as a Rectangle with given dimensions and positions
    // it to a random location on top of the screen
    object = new Rectangle();
    object.setSize(64, 64);
    object.x = MathUtils.random(0, Gdx.graphics.getWidth() - object.getWidth());
    object.y = Gdx.graphics.getHeight();
    objects.add(object);
}
```

Figure 10 Creating the objects

5.3.4 Processing the user input

In this example game the user input is only the occasion when the user taps on the screen. From the position of the tap, a new Rectangle is created (figure 11) and then it is possible to check for overlapping between the tap and the falling objects. Alternative to this method would be creating an onClickListener for the objects.

```
private void processUserInput() {
    // finds out if the user has clicked on a object
    if(Gdx.input.isTouched()) {
        //user touched this area
        touchCordinates.set(Gdx.input.getX(), Gdx.input.getY(), 0);
        camera.unproject(touchCordinates);
        Rectangle touchArea = new Rectangle(touchCordinates.x, touchCordinates.y, 10, 10);

        // goes trough all the objects
        for(i = 0; i < objects.size; i++){
            // if the object overlaps with the touch on screen
            if(objects.get(i).overlaps(touchArea)){
                objects.removeIndex(i); //remove from the array
                score++; //add +1 to score
            }
        }
    }
}
```

Figure 11 Processing the user input

5.3.5 Drawing the game

All the images and the font are drawn to the screen by using OpenGL. In Libgdx 2D textures can be drawn by using a SpriteBatch (figure 12). Each object and the font for the score need to be drawn for every single frame.

```
private void drawGame() {
    // updates the camera for each frame
    camera.update();

    // batch renders the view with dimensions specified by the camera
    batch.setProjectionMatrix(camera.combined);

    batch.begin();
        //goes through the objects array one by one
        for(Rectangle object: objects) {
            //and draws each object on to the screen
            batch.draw(objectImage, object.x, object.y);
        }
        //draws the current score on to the top left corner of the screen
        font.draw(batch, "Score: " + score, 0, Gdx.graphics.getHeight());
    batch.end();
}
```

Figure 12 Drawing the objects and the score

The render() method shown in figure 13 is called constantly through the life-time of the game in a rate of frames per second. All the input processing, game state

updates and drawing is done in this method by calling the appropriate methods since they require constant updating.

```
@Override
public void render() {
    // each frame is cleared with black color, RGB (0, 0, 0, alpha)
    Gdx.gl.glClearColor(0, 0, 0, 1);
    Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

    processUserInput();
    updateGameState();
    drawGame();

}
```

Figure 13 Render() utilizes the previously created methods

5.3.6 Disposing assets

Although the modern mobile operating systems free up memory automatically once needed it is still necessary to dispose the loaded assets when possible to avoid memory leaks. The following dispose (figure 14) method is automatically called by the Libgdx framework when the application is going to get terminated by the user or by the operating system.

```
@Override
public void dispose() {
    // unloading the disposable resources from memory
    objectImage.dispose();
    font.dispose();
    batch.dispose();
}
```

Figure 14 Dispose method

5.4 Debugging on desktop

Development is mostly done by adding new lines of code, or changing something from the old lines of code. After certain amount of changes, the new features need to be tested and verified that they work as indented. Compiling the

application for the desktop is the fastest way to run the application since it only takes a few seconds to deploy. Deploying to a mobile device is a lot slower.

The desktop version of the application can be started by right clicking the desktop project and selecting Run As Java Application. Debug mode can be started by selecting the Debug As option or using the Debug icon from the toolbar.

Debug mode allows hot swapping of code, which is the fastest way to change or try new code in the application. No compiling or restarting of the application is required between changes. The changes are instantly visible within the application.

5.5 Logging

Debug messages can be printed with "Gdx.app.log("message-tag", "message");". While running the desktop project debug messages are shown in the Eclipses Console view and when running on Android the messages are printed in LogCat, which is also commonly used with Android development. LogCat also provides all system debug output's from the Android device. The output can be filtered with a tag to show only the relevant output. All errors and exceptions will also be printed to these views. (Android Developer, logcat.)

5.6 Deploying to a device

It is also necessary to test the application on real devices. Modern computers have more processing power and better graphics processing units than mobile devices. A good application should be created so that it can be run with even older mobile devices with lower-end components. The procedure for deploying to a device varies from the platform.

5.6.1 Deploying to Android

The easiest way to deploy to an Android device is to connect the device through a usb-cable with the development computer. Depending on the device a usb driver for Windows might be required from the devices manufacturer. Android requires a Usb debugging option to be enabled from the device's developer options.

The application is deployed as an apk (Android application package) file to the device by right clicking the Android project and selecting Run As > Android Application or by using the Run icon from the toolbar. Applications sent to digital distribution platforms need to be signed when exported as apk files. This can be achieved by using the Export as Android application functionality in Eclipse.

5.6.2 Deploying to iOS

Apple's Mac OS X is required when deploying to an iOS device or a simulator. Deployment cannot be done in Windows or Linux based operating systems since the necessary Xcode is only supported in OS X. Deploying to an iOS device also requires purchase of Apple Developer Program license for iOS. Deploying to an iOS simulator is possible without the license.

After a proper Development Certificate, Application identifier and Provisioning Profile have been acquired to the development environment it is possible to deploy the application to an iOS device (iOS Developer Library, Maintaining Your Signing Identities and Certificates). The required Certificate, Identifier and Profile can be obtained from Apple's Developer portal after purchasing the necessary license.

Deployment to a device is done by right clicking the RoboVM project and selecting iOS Device App from Run As. The Package for App Store / Ad-hoc distribution selection in RoboVM tools allows to build the application for the App Store. The built application file (.ipa) can be sent for evaluation for the App Store with a tool named Application Loader.

6 Publishing the application

Finished applications are usually published on digital distribution platforms where consumers can download the application for free or purchase the application for a price. Distribution platforms have their own rules and demands (such as application quality, content or price) for the submitted applications.

6.1 Publishing for Android

Google allows Android applications to be published to any digital distribution platform, shared from any website directly, or be sent as email attachments as an installation package (.apk). Google's own digital distribution platform is called Google Play, formerly known as Android Market. (Android Developer, Open Distribution.)

6.2 Publishing for iOS

Applications for iOS can only be published to the official Apple App Store. All applications must meet Apple's guidelines for quality and content before acceptance to the App Store. In order to publish an application it must be built using the Distribution Provisioning Profile, which can be created after purchasing the iOS Developer program license. Applications are submitted for approval through Application Loader and iTunes Connect website. (iOS Developer Library, About App Distribution).

6.3 Publishing for desktop

Libgdx applications for the desktop are created with the Eclipse's Export functionality. Exported applications are created as executable .jar files which can be launched by opening the file.

Exported applications can be freely distributed through websites or distribution services such as Valve's Steam or Apple's Mac App Store. These commercial distribution platforms have their own submit and approval processes like Google Play and iOS App Store.

7 Falling Stars - a game with Libgdx

This thesis also developed a mobile cross-platform game (figure 16) with Libgdx named Falling Stars. The game was developed for Android and iOS mobile phones and tablets. Although a desktop version is possible with Libgdx, only the mobile platforms Android and iOS were the main target platforms.

Figure 16 Falling Stars game play

The playable character is a flying squirrel and the object of the game is to complete various challenges and to collect the stars from the sky for points and fuel. Points can be used to purchase upgrades and power-ups.

Figure 17 Main menu of Falling Stars game

The development was done by using many tools, community add-ons and features of Libgdx such as light effects from box2dlights, particle effects were created by using the Particle editor and object physics were achieved by using the Box2D extension. Also Google Play Game Services (icons in figure 17) leaderboards and badges were integrated to the game. The player can use these services by logging in with a Google+ account within the game itself.

The game was successfully created by reading tutorials from the Libgdx community forums and by watching Libgdx video tutorials from YouTube. Libgdx also comes with comprehensive documentation, which was proven to be a great resource for development.

Figure 18 Falling Stars game on different platforms

In above figure 18 the game is running on iOS devices iPad 2 and iPhone 4 and also Android devices Asus TF300 and Samsung S3.

8 Summary

The main purpose of this thesis was to find and learn about a framework, which enables development of cross-platform games for mobile devices. Libgdx proved to be a good choice for the developed game Falling Stars, since it was possible to develop the game for the target platforms from the same codebase.

In the future more mobile platforms like Firefox OS, Tizen, Sailfish or Ubuntu Touch are likely to be also target platforms for popular mobile games and applications. Developing applications or games to all of these platforms in native code is unlikely because of the amount of time and expertise required for each platform. For commercial applications this requires more work hours and programmers who are experienced with developing to these platforms.

Alternative technologies such as HTML5 based technologies are likely to be more common choices for development in the future because of the commonly used JavaScript and HTML languages. Currently the main issue holding back the spreading of HTML5 based development is the performance of HTML5 based mobile applications. Native applications are generally more responsive than HTML5 based applications. Especially with games it is important to achieve the best possible performance from the available device.

List of references

1. Intel Developer Zone. Building Cross-Platform Apps with HTML5. http://software.intel.com/en-us/html5/articles/building-cross-platform-apps-with-html5 Read 3.1.2014.

2. GAJOTRES, Top 7 mobile application HTML5 frameworks. http://www.gajotres.net/top-7-mobile-application-html5-frameworks/. Read 3.1.2014.

3. Xamarin. Tour. http://xamarin.com/tour Read 3.1.2014

4. Libgdx. Goals and Features. http://libgdx.badlogicgames.com/features.html. Read 3.1.2014.

5. Libgdx Developer's Guide. Introduction. https://github.com/libgdx/libgdx/wiki/Introduction Read 12.12.2013.

6. Android Developer. Android, the world's most popular mobile platform. http://developer.android.com/about/index.html. Read 17.11.2013.

7. iOS Developer Library. About iOS App Programming. https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007072-CH1-SW1 Read 10.12.2013.

8. History of Java programming language. History. http://www.freejavaguide.com/history.html. Read 4.2.2014.

9. Groups.enging.umd.emich.edu, Java The Programming Language. http://groups.engin.umd.umich.edu/CIS/course.des/cis400/java/java.html Read 2.1.2014.

10. Libgdx Developer's Guide, Prerequisites. https://github.com/libgdx/libgdx/wiki/Prerequisites. Read 5.2.2014.

11. Aurelion Ribon's dev Blog. Libgdx Project setup v3.0.0! http://www.aurelienribon.com/blog/2012/09/libgdx-project-setup-v3-0-0/. Read 7.2.2014.

12. Android Developer, ProGuard. http://developer.android.com/tools/help/proguard.html Read 4.1.2014.

13. Android Developer, logcat. http://developer.android.com/tools/help/logcat.html Read 15.12.2013.

14. iOS Developer Library. Maintaining Your Signing Identities and Certificates. https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html. Read 20.10.2013.

15. Android Developer, Open Distribution. http://developer.android.com/distribute/open.html Read 2.12.2013.

16. iOS Developer Library. About App Distribution. https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html Read 8.12.2013.

```
package com.me.mygdxgame;

import com.badlogic.gdx.ApplicationListener;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.GL10;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.math.MathUtils;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.math.Vector3;
import com.badlogic.gdx.utils.Array;

public class MyGdxGame implements ApplicationListener {
   private OrthographicCamera camera;
   private SpriteBatch batch;       // draws 2D textures

   private Array<Rectangle> objects; // holds the objects that player tries to click
   private Rectangle object = new Rectangle(); // objects are defined as Rectangles
   private Texture objectImage;
   private float time = 0;  // time tracking for making objects appear
   private float lastSpawnTime = 0;

   private int score;  // keeps the players score
   private BitmapFont font; // default font which comes with Libgdx

   private Vector3 touchCordinates = new Vector3();
   private int i;

   @Override
   public void create() {
       camera = new OrthographicCamera();
       camera.setToOrtho(false, 800, 480); // cameras viewport width and height
       batch = new SpriteBatch();

       // loads the image for the object from the assets folder
       objectImage = new Texture(Gdx.files.internal("object.png"));

       // clickable objects array
       objects = new Array<Rectangle>();

       // font to draw the score
       font = new BitmapFont();
   }

   @Override
   public void render() {
       // each frame is cleared with black color, RGB (0, 0, 0, alpha)
       Gdx.gl.glClearColor(0, 0, 0, 1);
       Gdx.gl.glClear(GL10.GL_COLOR_BUFFER_BIT);

       processUserInput();
       updateGameState();
       drawGame();
   }
```

```
private void processUserInput() {
   // finds out if the user has clicked on a object
  if(Gdx.input.isTouched()) {
     //user touched this area
     touchCordinates.set(Gdx.input.getX(), Gdx.input.getY(), 0);
     camera.unproject(touchCordinates);
     Rectangle touchArea = new Rectangle(touchCordinates.x, touchCordinates.y, 10,
     10);

     // goes trough all the objects
     for(i = 0; i < objects.size; i++){
      // if the object overlaps with the touch on screen
       if(objects.get(i).overlaps(touchArea)){
          objects.removeIndex(i); //remove from the array
          score++; //add +1 to score
       }
     }
   }
}

private void updateGameState() {
   time += Gdx.graphics.getDeltaTime();

    // if a second has passed since the previous object was made
    if(time - lastSpawnTime > 1){
          createNewObject();
          lastSpawnTime = time;
    }
    // moves the objects downwards and removes one once
    // the object is below bottom of the screen
    for(i = 0; i < objects.size; i++){
      objects.get(i).y -= 250 * Gdx.graphics.getDeltaTime();
      if(objects.get(i).y + objectImage.getHeight() < 0){
          objects.removeIndex(i); // removes the object
          score--; // -1 to the score
      }
    }
}

private void drawGame() {
   // updates the camera for each frame
   camera.update();

   // batch renders the view with dimensions specified by the camera
   batch.setProjectionMatrix(camera.combined);

   batch.begin();
   //goes through the objects array one by one
   for(Rectangle object: objects) {
          //and draws each object on to the screen
          batch.draw(objectImage, object.x, object.y);
   }
   //draws the current score on to the top left corner of the screen
   font.draw(batch, "Score: " + score, 0, Gdx.graphics.getHeight());
   batch.end();
}
```

```
    private void createNewObject() {
        // creates new object as a Rectangle with given dimensions and positions
        // it to a random location on top of the screen
        object = new Rectangle();
        object.setSize(64, 64);
        object.x = MathUtils.random(0, Gdx.graphics.getWidth() - object.getWidth());
        object.y = Gdx.graphics.getHeight();
        objects.add(object);
    }

    @Override
    public void dispose() {
        // unloading the disposable resources from memory
        objectImage.dispose();
        font.dispose();
        batch.dispose();
    }

    @Override
    public void resize(int width, int height) {}
    @Override
    public void pause() {}
    @Override
    public void resume() {}
}
```