VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Wu Danni

SNIPE BOB: A GAME FOR ANDROID MOBILE PHONE

Technology and Communication

2014

PREFACE

I started to implement the application at the Telecommunication and Information Technology Department, Vaasa University of Applied Science, at January 2014.

The whole thesis is developed on the base of the final result in the course Software Engineering Project, and competed all by myself.

I would like to express my greatest gratitude to my supervisor, Mr. Ghodrat Moghadampour , for his splendid advices. Not only because his supports for my study and research over almost three years, but also academically and emotionally assistances and guidance to finish this thesis. He helped me come up with the thesis topic and guided me through the whole developing process. Whenever I met difficulties he would help me patiently and encourage me to move on independently.

Much gratitude to Mr. Kalevi, Ylinen and all the members of Telecommunication and Information Technology Department in Vaasa University of Applied Science who gave me a hand when I encountering different kinds of problems.

I would also like to thank my classmates and friends. They helped and inspired me a lot when I came across problems.

Thanks for my family. You are always strongest and most important inspirations and spiritual props of me.

Wuhan, 06.05.2014

Wu Danni

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Author	Wu Danni
Title	Snipe Bob: A Game for Android Mobile Phone
Year	2014
Language	English
Pages	70 + 2 Appendices
Name of Supervisor	Ghodrat Moghadampour

Android is an open source operating system based on Linux, which is mainly used in mobile devices. The data at end of 2010 shows that Android system has become the world's largest Smartphone operating system. Furthermore, in the application store, the downloads of game is the largest of all of the application types. According to the situation above, the Android game has large market.

The game development is not only about the programming, but also about the art and creation. The concept is the most significant part of the game. It should be clear and easily understood.

The objective of this thesis was to state the process of development of a shooting game named Snipe Bob for the Android phone operating system. The project was carried out on Sumsung GT-I8262D with Android version4.1.2. By the time writing this thesis, Snipe Bob has been completed and can run smoothly on the phone.

The game's name is Snipe Bob. It has four main screens: menu screen, game screen, high score screen and help screen. The task, player needs to complete, is to get enough score by shooting the bobs, which are moving across the screen one by one. The game has three levels. The player can get into next level after getting enough scores in the current level. On each level the bobs' moving speed will get faster and faster. The player's score will decrease if a squirrel, which is another character moving across the screen, is shot.

Keywords Android, game, OpenGL ES, Java

CONTENTS

PREFACE
ABSTRACT
1 INTRODUCTION..9
2 BACKGROUND..11
2.1 THE SMARTPHONE.. 11
2.2 ANDROID SYSTEM..12
2.2.1 CREATION AND DEVELOPMENT.. 12
2.2.2 OPEN HANDSET ALLIANCE..13
2.2.3 LINUX KERNEL... 15
2.2.4 HARDWARE SUPPORT..16
2.2.5 VERSIONS...17
2.2.6 GOOGLE PLAY...17
2.2.7 APPLICATION DEVELOPMENT..18
2.3 ANDROID GAME..19
2.4 OPENGL ES..19
3 SNIPE BOB.. 20
3.1 CORE GAME MECHANISM.. 20
3.2 SCREENS AND TRANSITIONS..21
3.3 MAIN FUNCTIONS AND STRUCTURES...23
4 IMPLEMENTATION OF SNIPE BOB...29
4.1 THE ASSETS CLASS..29
4.2 THE SETTINGS CLASS..32
4.3 THE MAIN ACTIVITY..34
4.4 THE MAINMENUSCREEN...36
4.5 THE HELP SCREENS..40
4.6 THE HIGH-SCORES SCREEN..43
4.7 THE SIMULATION CLASSES.. 46
4.7.1 THE BALL CLASS...47

4.7.2 THE BOB CLASS.. 48
4.7.3 THE CANNON CLASS.. 50
4.7.4 THE SQUIRREL CLASS..51
4.7.5 THE WORLD CLASS...52
4.7.5.1 WORLD GENERATION... 53
4.7.5.2 COLLISION DETECTION.. 55
4.8 THE GAME SCREEN... 57
4.9 THE WORLDRENDER CLASS... 63
5 TESTING.. 67
6 CONCLUSION... 68
REFERENCES...69
APPENDICES

LIST OF FIGURES AND TABLES

Figure 1. First Android Smartphone HTC T-Mobile G1 p. 11

Figure 2. Android System Architecture p. 13

Figure 3. Android Logos of Each Version p. 14

Figure 4. Initial game mechanics mock -up p. 18

Figure 5. All the screens and transitions of Snipe Bob p. 19

Figure 6. Snipe Bob Use Case Diagram p. 20

Figure 7. Ball-control sequence diagram p. 21

Figure 8. View help sequence diagram p. 21

Figure 9. View high scores sequence diagram p. 22

Figure 10. Turn on/off the sound sequence diagram p. 22

Figure 11. Quit Game sequence diagram p. 23

Figure 12. Assets Class p. 25

Figure 13. Settings Class p. 28

Figure 14. CannonShootGame Class p. 30

Figure 15. MainMenuScreen Class p. 32

Figure 16. HelpScreen Class p. 35

Figure 17. HighscoreScreen Class p. 38

Figure 18. Ball Class p. 41

Figure 19. Bob Class p. 42

Figure 20. Cannon Class p. 44

Figure 21. Squirrel Class p. 45

Figure 22. World Class p. 46

Figure 23. World Generation Process p. 47

Figure 24. Bound objects p. 48

Figure 25. Two overlap situations p. 48

Figure 26. Ready Screen p. 50

Figure 27. Normal-running Screen p. 51

Figure 28. Next-level Screen p. 51

Figure 29. Game-over Screen p. 52

Figure 30. Pause Screen p. 52

Figure 31. Initialize Cannon Ball Speed p. 53

Figure 32. Touch Input p. 54

Figure 33. WorldRender Class p. 55

LIST OF APPENDICES

APPENDIX 1. World class code

APPENDIX 2. GameScreen class code

1 INTRODUCTION

The society has entered an era of Mobile Internet, which has changed every aspect of life, as well as the the way of human’s thinking. Mobile Internet provides access to the Internet on the mobile devices, such as mobile phone, Tablet Personal Computer (PC) and Personal Digital Assistants (PDA). And by present, the Mobile Internet is most widely used on the mobile phone because of it is portable and multifunctional.

There are a variety of mobile applications, in which, entertainment is one of the most attractive fields. A game can provide good entertainment for people to kill the boring fractional time. And because of this feature, the download number of game application is ongoing increasing these years.

Developing a game has more work to do than the normal application development. It also requires the developer to design the game background to let the player get into the game, as well as the image and sound design to act in concert with the background. So it is more about creation. Code programming is a significant part in any process of application development. Building a nice game is not an easy thing, but it is an interesting experience.

To play Snipe Bob, the player needs to use a finger to control the direction and power of the shooter on the lower left corner to shoot objects moving on the screen, and the game will record the score of the player and rank the scores in best five.

In the thesis, Chapter 2 is about the background of the game development, and in that part, the feasibility, needed technology and necessary background of the Android game development is explained. The Chapter 3 of the thesis is about the game introduction, including core mechanism of the game, screens and the transitions between them, main functions and the structure. In the Chapter 4, the implementation will be stated. The last two chapters are about the testing and conclusion.

2 BACKGROUND

This chapter describes the technologies and knowledge background.

2.1 The Smartphone

A smart phone is one kind of mobile phone which owns independent mobile operating system. It can expand its functions by installing applications and games. Its computing power and functionality are all better than the traditional mobile phone.

The initial smart phone is the combination of the Personal Digital Assistants (PDA) and mobile phone. Later, the functions of portable media player, fool digital camera, pocket camcorder and GPS, etc, were added to the combination. And those made this kind of combination as a device with a variety of features.

The statement "smart phone" is mainly related to the "feature phone", which is the collection of titles to the mobile phones that have stronger computing ability and functions.

A smartphone can show the same pattern of the web page as the web page pattern is shown in the computer. Besides, a smartphone can also show the mobile version of the website. It has the independent operating system and good user interface, as well as the strong expansibility of applications. It can install and delete applications easily. In addition, the smartphone has a large high-definition touch screen, can call keyboard to touch handwriting at any time, features multi-tasking operating, and also possesses powerful functions of multimedia, e-mail and internet. These all make the smartphone completely replace the traditional portable devices like mp3, mp4 and PDA, etc. As for the aspect of handling some of the office affairs and other affairs, the smartphone can replace the personal computer. The smartphone can always keep seamlessly connecting with the network, and can synchronize data with computer and laptop and so on.

In the past several years, the sales of the smartphone have already exceeded the other types of the mobile phones in the market. According to one survey in 2012, about half of the mobile consumers in the United States were using smart phone,/8/ in the group of 25 - 34 years old people, the share of the smartphone reached to 62%./10/ The report of NPD Group stated that for the consumers over the age of 18 in the United States, the sales of the smartphones had already reached 59% in the third quarter, 2011./13/

According to the report of comScore, in the European mobile phone market, at the end of the 2012, the total number of mobile Internet users has achieved 241 million in German, France, Italy, Spain and the UK, in which, 57% were smartphones. The trend of using smartphone is very obvious. Besides, in those countries, the penetrations of the smartphone are all over 50%, and it in Spain reached 66%. Up December of 2012, 75% of the consumers buying mobile phones are smartphones in those five countries.

In China, the sales of smartphones were over half up the second quarter of 2012./7/

2.2 Android System

Android is an open source operating system based on Linux, which is mainly used in mobile devices, such as smartphone and tablet PC. At present, the Android operating system is a project ongoing led and developed by Open Handset Alliance (OHA), established by Google. The latest version of Android is Android 4.4.2.

2.2.1 Creation and Development

In October 2003, “the father of Android” Andy Rubin established Android Inc. in Palo Alto, California, United States, and developed the company with Rich Miner, Nick Sears and Chris White.

The initial aim of developing this system is to create an advanced operating system for digital cameras. Later on, they found that the market demand was not big enough to develop, and in addition, the quick growing of the smartphone market, so Android was reformed for smartphones.

On 17th of August, 2005, Android Inc. was bought by Google and became part of it, including a wholly owned subsidiary owned by other founders and all of the staff of the company.[/11/]

Rubin led a team which is responsible for the mobile operating system based on Linux kernel in Google. This development project is the Android operating system. Google's cooperation platform provides a broad market for Android. Google gives a flexible and reliable system upgrade commitment to major hardware manufactures and software developers, and guarantee to give them latest version of operating system.

In September 2007, Google submitted applications for patents of mobile field.

2.2.2 Open Handset Alliance

On 5th of November, 2007, the Open Handset Alliance, OHA was established under the lead of Google. The earliest members include Broadcom, HTC, Intel, LG, Marvell, etc. The objective of OHA is to create a more open and free mobile telephone environment. On the same day with the OHA creation day, the alliance showed its first product, a smartphone with Android operating system using Linux 2.6 as the core foundation. In December 2008, the new batch of members joined OHA, including ARM, HUAWEI, Sony, etc.[/12/] Meanwhile, the Android Open Source Project (AOSP) was created, which is responsible for sustainable development of the Android operating system.[/1/] Beside OHA, Android also has open source communities made up by developers all around the world to develop Android applications and the third-party Android operating system, in order to extend and expand Android's functionality and performance.[/17/]

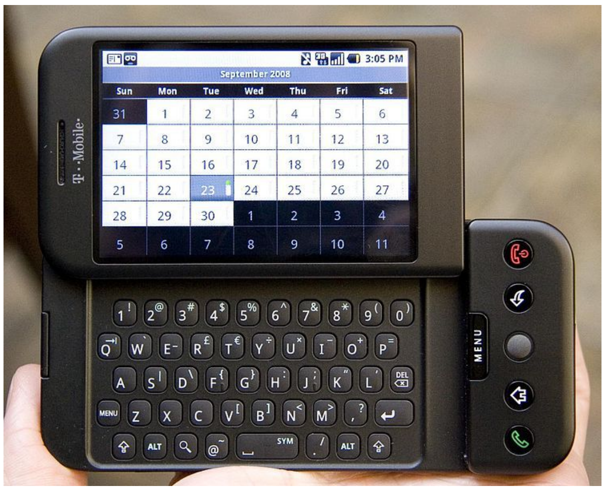

Figure 1. First Android Smartphone HTC T-Mobile G1/19/

The Android operating system uses free open-source license, which means all of the source codes of the Android system are open and free. The majority of Google Android was released in Apache open source terms 2.0, and the under Linux kernel inherited GPLv2 license./3/ AOSP contains the necessary functions of the smartphone, including smartphone network and telephone protocol stack, etc. Google is also continuing to publish questionnaires, open modified list, and renew situation and codes to let anybody see and provide their opinions and comments, so that Google can improve the Android operating system according to the user's requirements.

The Android operating system is completely free and open source, any firm can use the Android operating system freely without the authorization of Google and OHA. But manufacturers cannot use Google's brand and application without the authorization, such as Google Play. Only if the product is in line with Google's compatibility definition file, the manufacturer can pre-load Google Play Store, Gmail, etc, private application of Google and get a compatibility definition file. Furthermore, smartphone vendors can print "With Google" sign on their smartphone./5/

2.2.3 Linux Kernel

The core of the Android operating system is a branch of Linux kernel, so it owns a typical Linux scheduling and function. In addition, in order to let Linux run well in the mobile devices, Google revises and expands it. Android removes X Window System in the Linux, and also does not support the GNU library, which makes it difficult to transplant applications in the Linux platform to the Android platform.[/15/] In 2008, Patrick Brady gave a lecture “Anatomy Physiology of an Android” at Google I/O, and proposed Android HAL chart . HAL exist as the form of *.so files, which can separate the Android framework and the Linux kernel. This kind of intermediary layer way make Android’s operating efficiency higher in mobile devices. This unique system architecture was praised by developer of Linux kernel, Greg Kroah - Hartman and other core maintainers. Google also added a mobile device power management feature named “wakelocks” into the Android core, which is used to manage battery performance in a mobile device. But this feature was not added in mainline open and maintenance of Linux kernel, because the Linux kernel maintainer thought that Google did not show them the intent and codes of the feature.

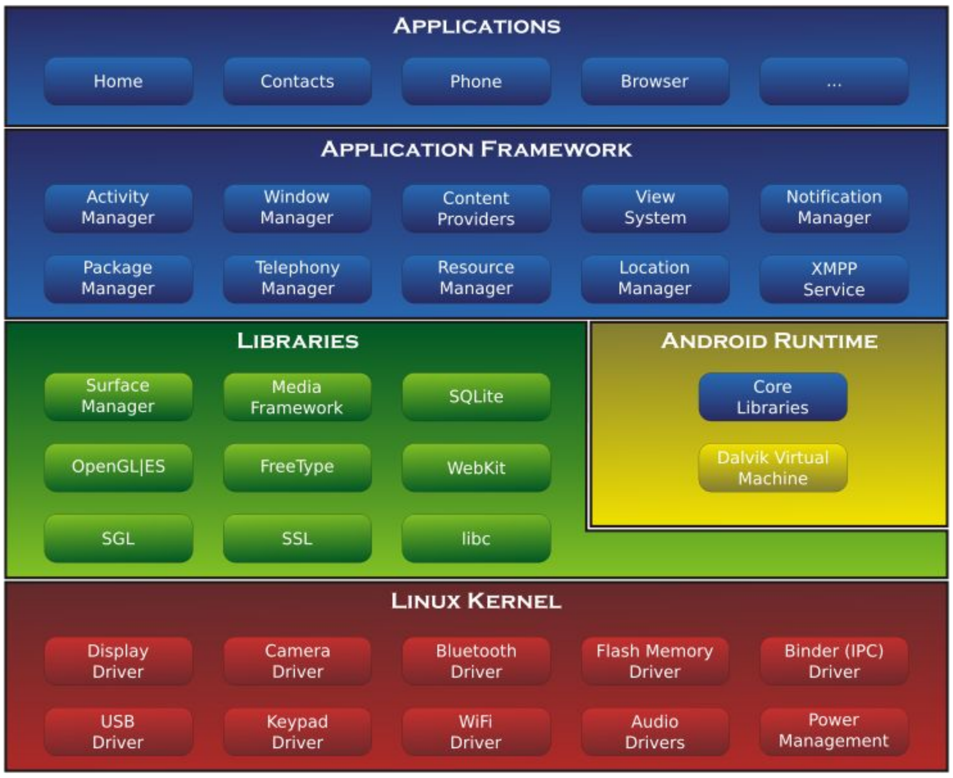

Figure 2. Android System Architecture[/2/]

Linus Torvalds said that the core of Android and Linux would merge together finally, but maybe not in four to five years. The most of the codes was integrated completed in Linux 3.3.[/16/]

2.2.4 Hardware Support

As openness and portability of the Android operating system can be used in most electronic products, including smartphone, tablet personal computer, television, ebook reader, music player, automotive equipment, and other equipments. The Android operating system is mostly mounted in the hardware with the ARM architecture. There are also Android systems which support x86 architecture, such as Google TV. Likewise, iOS equipments, such as iPhone, iPod Touch and iPad, can run the Android operating system by dual-boot tool OpeniBoot or iDroid. So do the Microsoft products.

In the following the versions of the Android operating system will be introduced.

2.2.5 Versions

Before the official release of the Android, it had two internal test version, named as the famous robot name, Astro Boy (AndroidBeta) and Clockwork Robot (Android 1.0). Later on, as it related to copyright issues, Google changed its naming rules as a dessert. The sizes of desserts become bigger and bigger as the updates of every version. Each name of versions shown in the Figure 3.

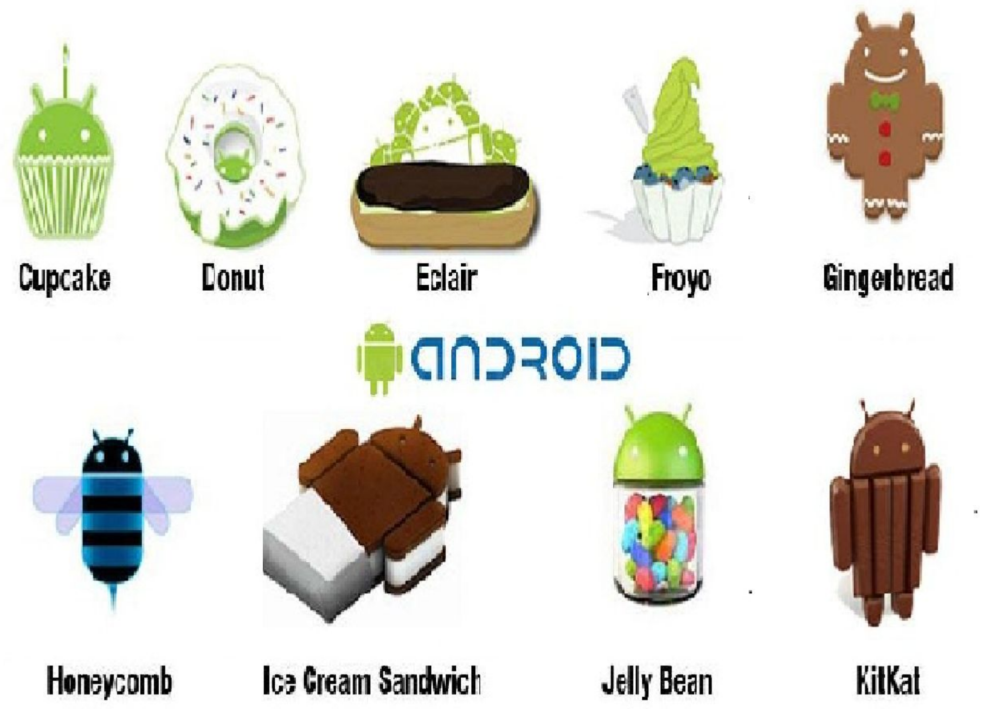

Figure 3. Android Logos of Each Version[14]

2.2.6 Google Play

Google provides applications and games for the users by Google Play which is an online store platform whose predecessor is Android Market. Up to July of 2013, the number of official certification applications in Google Play has broken through 100 millions, which surpassed Apple Store and became the biggest application store in the world./4/

In February of 2009, Google released an online application store, Android Market. Users can find, buy, download and evaluate the applications and other content through this platform web. Third-party software developers and freedom developers can also release the applications they developed through Android Market. In December of 2011, the applications downloads in Android Market had exceeded ten billion. Simultaneously, there has been 130 million Android devices had downloaded applications in Android Market, which is called Google Play now.

The pay programs in Google Play are provided in many countries and regions.

Besides Google Play, there are other application market, such as Amazon Appstore of Amazon, Samsung application store of Samsung, Fetch, AppBrian, and Pea pod in China, etc./9/

In addition, Google Play also provides service, the verification of installed applications in backstage of Android system, so that the prevention of the malware can be the maximum.

2.2.7 Application Development

In the early days of Android application development, Java in Android SDK (Software Development Kit) was usually used. In addition, developers can also use C or C++ in Android NDK (Native Development Kit). Simultaneously, Google released language, Simple, which is suitable for beginners. Furthermore, Google also released Google App Invertor development kit, which can build applications quickly, and is easy for novice developers.

2.3 Android Game

Android Game is the game running on the devices with the Android operating system, including not only Android mobile phones, but also Tablet PCs.

There are a variety of games, such as sports game, action game, card game, shooting game, and puzzle game, etc.

Chinese Smartphone Games Market Annual Consolidated Report 2012 shows that, in China, the market size of smartphone games firstly exceeded other games'. The market share of smartphone games was up to 51.16%. And in the 2013, the market share kept stable. With the increase of the market share of the Android operating system, this number will be higher. Moreover, Android Game also occupies a large share of the Android applications.

2.4 OpenGL ES

OpenGL ES (OpenGL for Embedded System) is subset of OpenGL three-dimensional graphics API (Application Programming Interface). OpenGL ES is mainly designed for embedded devices, such as mobile phone, PDA and game console, etc. This API is defined and promoted by Khronos Group, which is a graphics hardware and software industry association. The association main concern open standards of graphics and multimedia.

OpenGL ES is royalty-free, cross-platform and fully functional 2D and 3D graphics API. It has created a flexible and powerful low-level interface between software and graphics acceleration./18/

OpenGL ES includes system descriptions of floating-point and fixed-point arithmetic. OpenGL ES 1.X is designed for fixed-function hardware, and provides acceleration support, graphics quality and performance standards. And OpenGL ES 2.X provides fully programmable 3D graphics algorithms, including cover technology. /18/

3 SNIPE BOB

In this part, the Android shooting game Snipe Bob will be introduced, including the core game mechanism, the screens and transitions of the whole game, as well as main functions and structure.

3.1 Core Game Mechanism

Snipe Bob is a shooting game. The player should use a finger to control the direction and power of the shooter on the lower left corner to shoot objects moving on the screen. The game will record the score of the player and rank the scores in best five. The game will end if the score the player gets in one level is not enough. The new score will be saved if it can be one of the highest five. And the following are the details:

- The target characters, bob and squirrel, are constantly moving from the left of the screen to right.
- The ball is shot from the cannon on the left bottom corner of the screen. The angle and the initial speed is decided by the touch point of the player’s finger.
- The score will be increased when the ball has shot a bob, and will be decreased when the squirrel is shot.
- The speed of bobs will be increased when the level is higher.
- If the score the player gets in one level is equal or bigger than the target score, then the game will go into the next level. And if not, then the game is over.
- When the player pass all of the three levels, the game is over.

Figure 4 shows the initial mock-up of the core principles.

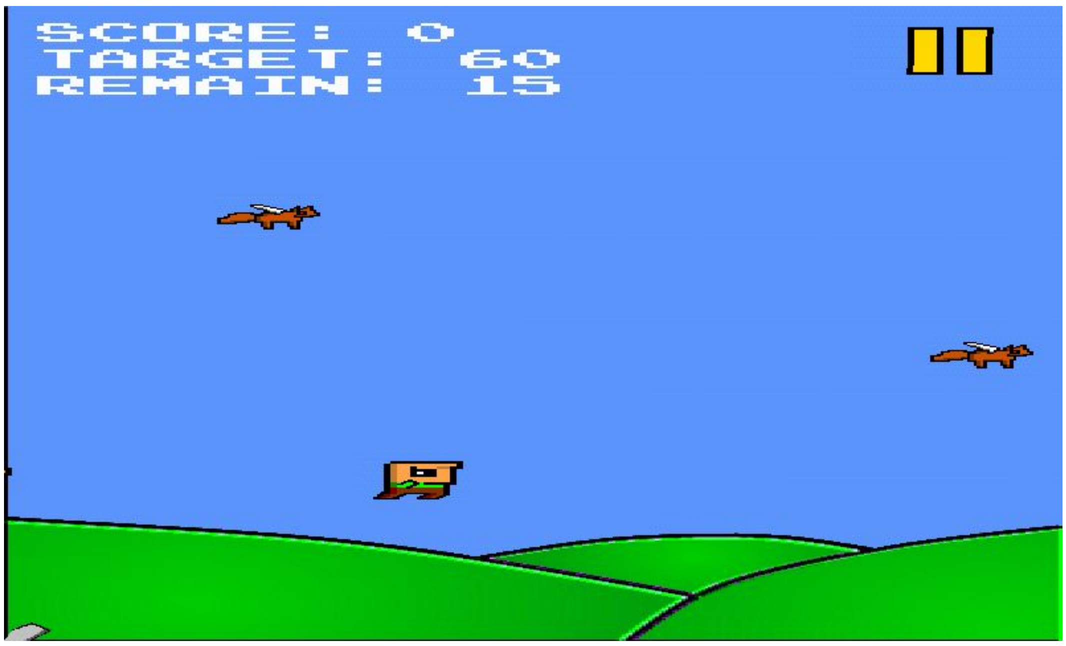

Figure 4. Initial game mechanics mock -up

3.2 Screens and Transitions

For the screens and transitions, the game follows the formula as:

- The game will have a main screen; PLAY, HIGHSCORES, and HELP menu items; and a button to disable and enable sound.
- The game will have a game screen which will ask the player to get ready and handle running, pause, game over, and next level states gracefully.
- The game will have a high-scores screen which will show the top five scores the player has achieved so far.
- The game will have help screens which present the game mechanics and goals to the player. It will leave out a description of how to control the cannon.

Figure 5 shows all screens and transitions.

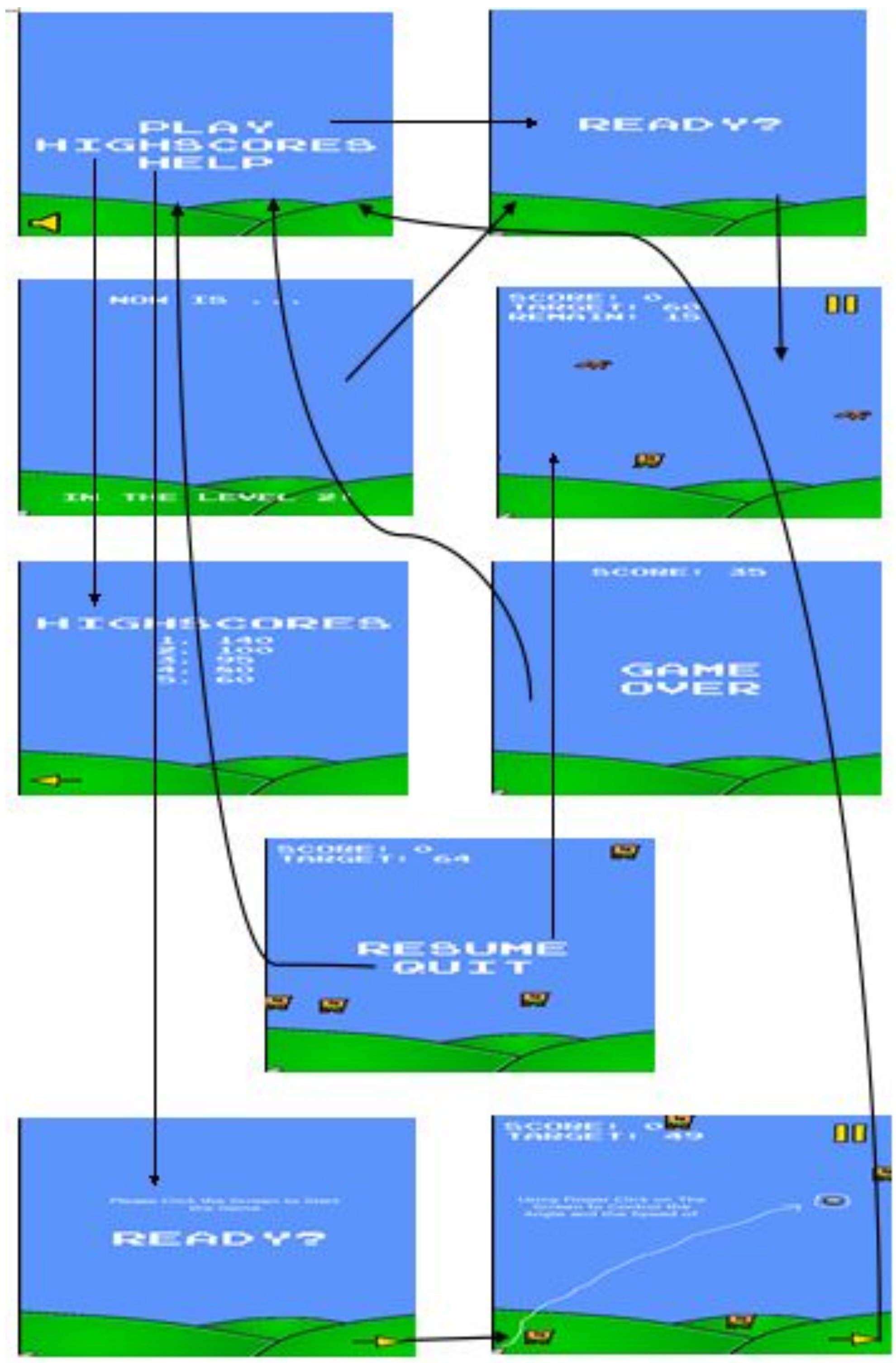

Figure 5. All the screens and transitions of Snipe Bob

3.3 Main Functions and Structures

As mentioned in the previous chapters about the creation of the game, the main functions and the structures of the application are designed as the follows.

The main functions:

- Control the ball shooting by touching the screen
- View help
- View high scores
- Turn on/off the sound
- Quit Game

The following is the use case diagram of the game.

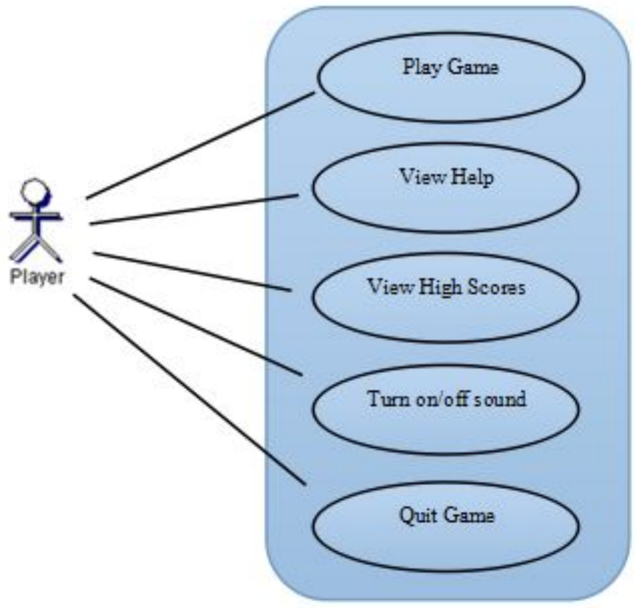

Figure 6. Snipe Bob Use Case Diagram

The speed of the shooting ball and movement is controlled by the touch of the player's finger, which is the core of playing the game. The application will calculate the distance between the touch point and the original point, as well as the angle between the x-axis and the line connecting the touch point and the original point. The speed of the ball is initiated according to the value of the distance and angle. The speed at y direction will be decreased because of gravity. And then the speed and position of the ball will be updated.

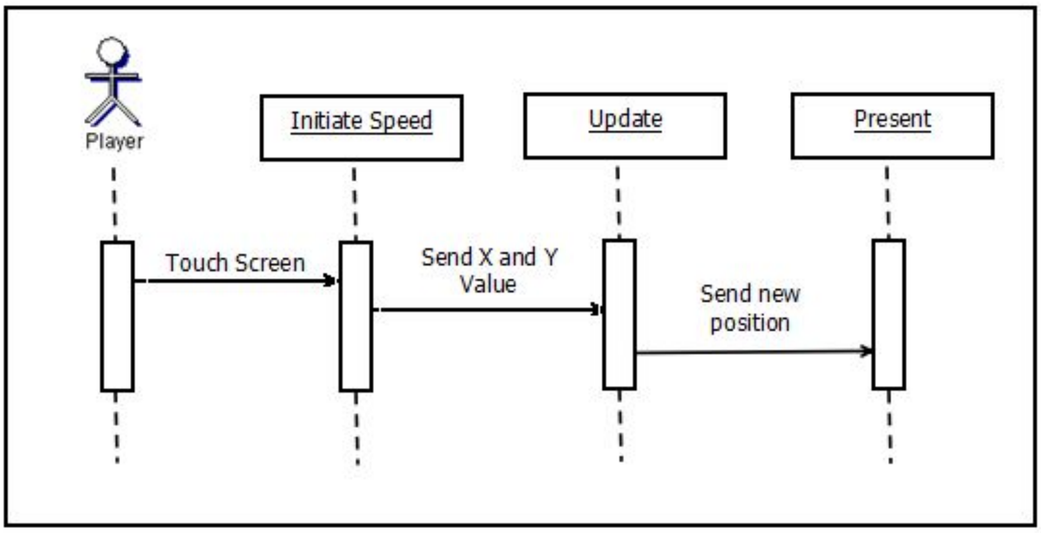

Figure 7. Ball-control sequence diagram

The function, "View help", is responsible for showing the tutorial of the game and teaching the player how to play the game. When the player presses the help screen button in the main menu screen, the signal of transiting screen will be sent. Then the game will turn to the help screen.

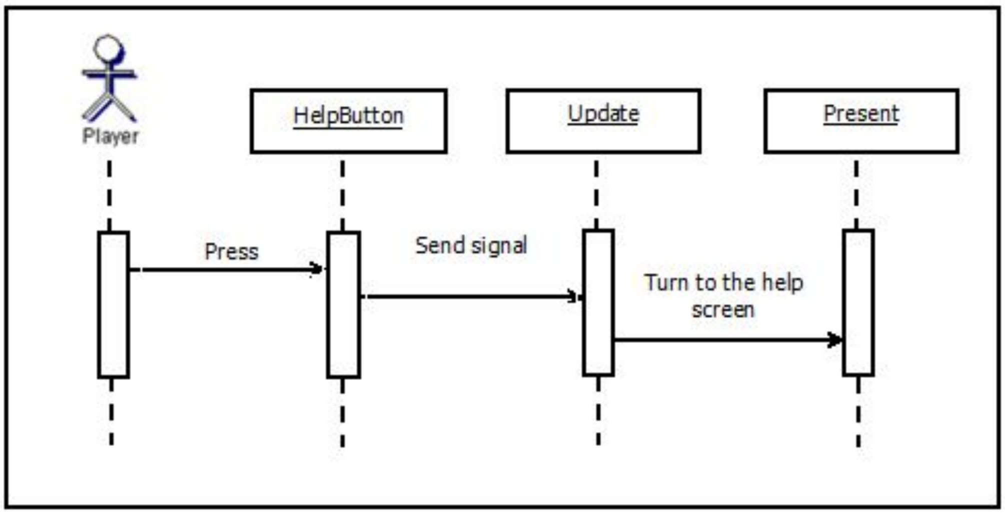

Figure 8. View help sequence diagram

The "View high scores" function is for presenting the highest five scores in the game. Players can view them by pressing the high scores screen button. Then the game will receive the signal for transiting the screen to the high scores.

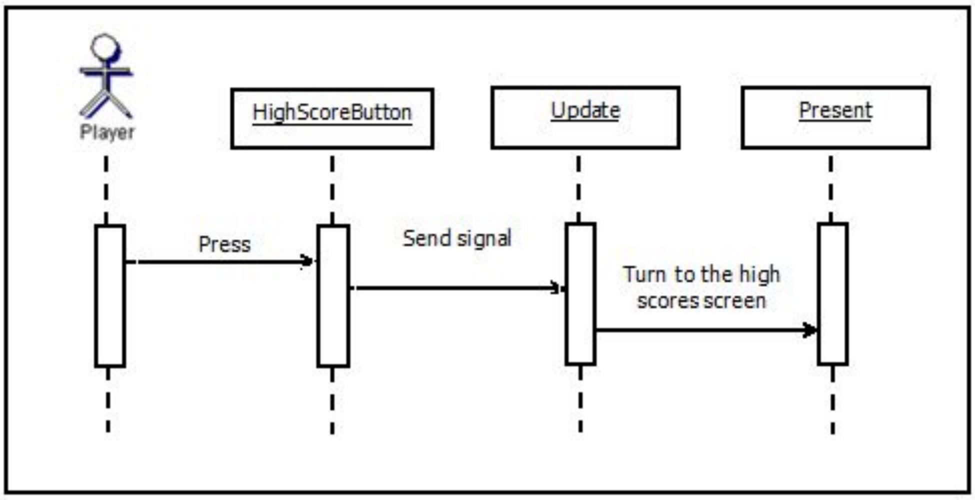

Figure 9. View high scores sequence diagram

The loudspeaker button is responsible for turning on and off the sound of game. There will be two types of speaker icons for the states of on and off. After pressing the speaker button, the icon will be changed according to the state of sound.

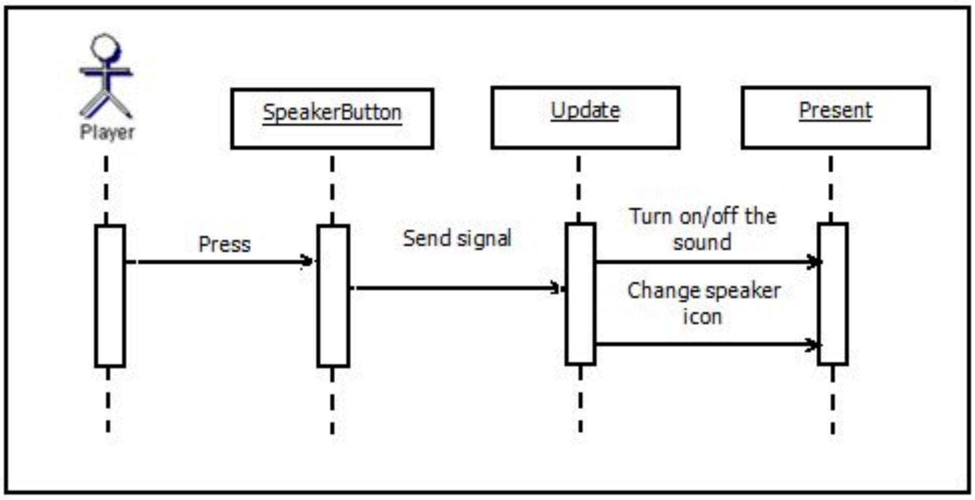

Figure 10. Turn on/off the sound sequence diagram

After pausing the game, if the player presses the Quit game button, then it will receive the quit signal and stop the game and the screen will transit to the starting menu.

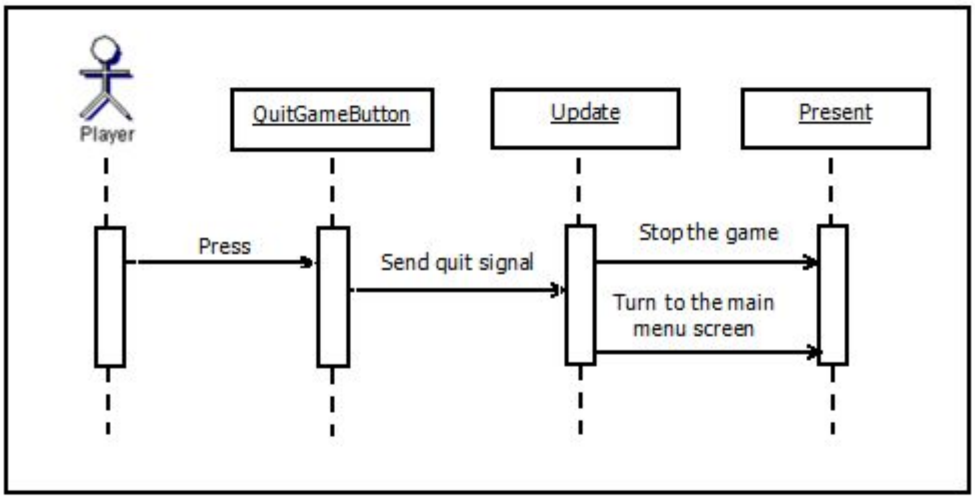

Figure 11. Quit Game sequence diagram

Snipe Bob will have five packages. Each package will deal with different functions.

The first package of the project is to build several basic interfaces for the game, including sound for play the sound effect, fileIO for input and output the score to the file, graphics for drawing and object for applying characters in the game, etc.

The second package is for the implementation of OpenGL ES, handling the problems about graphics displaying, such as animation, texture and font and so on.

In the third package, the math problem will be processed. To detect collision, characters in the game will be bound with rectangles and test the overlap of them. This package is responsible for the binding and overlap testing.

Then finally, the last package is the core of the application, the implementation of the main screens and game mechanism of the game. It will contain all of the screens and characters in the game. There will be four types of characters in the game, named bob, squirrel, ball and cannon. They are responsible for the initialization and update of the state. The classes for the main menu screen, game screen, high scores screen and help screen, will update and display different component of the game. In all of these screens, the game screen is the core part. It will achieve the main functions of the game. There will be one specific class named world for handling the main logic of the game world, including the generation and updating of all the characters in the game, collision detection, and the state changing of the game.

All of these screen classes have three main functions: a function whose name is the same as the screen name, update(), and present(). The first function is mainly responsible for the initialization of necessary content (variables, load images and sound). The function update() is responsible for monitoring and updating the states of the components, such as the speed and position of the characters. Finally, the function present() is responsible for rendering all of the components in screens.

The world class has most of the important functions of the game:

- World(WorldListener listener, int l): the constructor function which use WorldListener and the number of level as the parameters.
- generateLevel(): generates the objects in the world.
- updateBob(): updates the state and position of character bob.
- updateSquirrel(): updates the state and position of the character squirrel.
- updateCannon(): updates the angle of the cannon.
- updateBall(): updates the velocity and the position of the shooting ball.
- checkCollisions(): detects the collision between the ball and the characters, bobs and squirrels.
- checkLevelEnd(): checks whether the present level is at the end and converts the state of the game world.

4 IMPLEMENTATION OF SNIPE BOB

4.1 The Assets Class

The Assets class holds references to all the Texture, TextureRegion, Animation, Music, and Sound instances which are needed throughout game.

The Assets class has three methods:

- load(): is responsible for populating all the static members of the class.
- reload(): reloads the members when the application is resumed.
- playSound(): plays the sound.

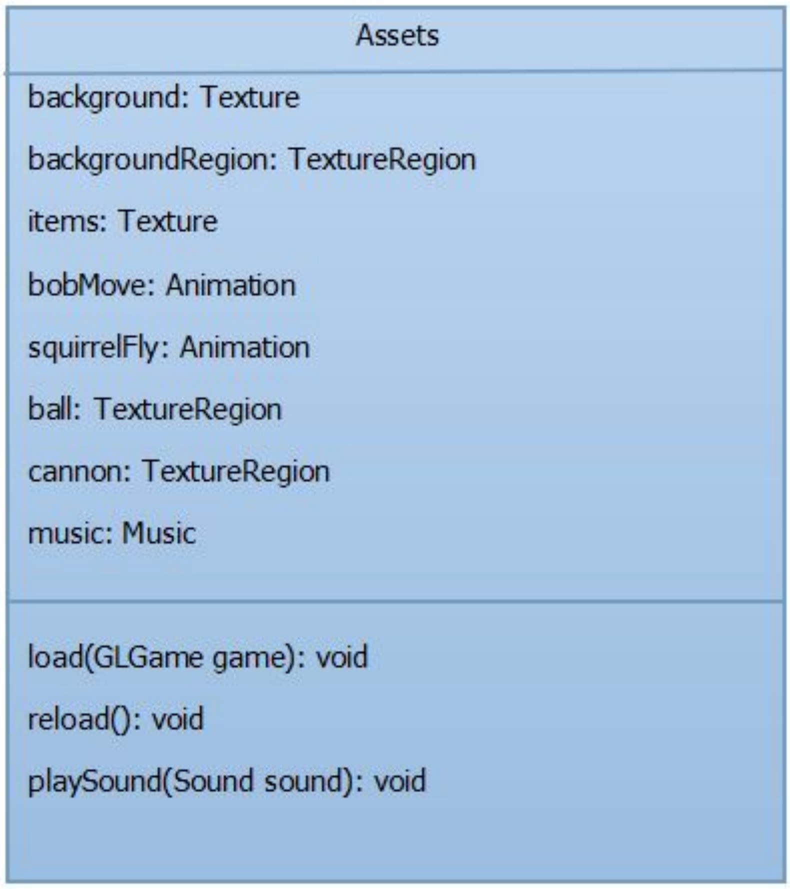

Figure 12. Assets Class

Background and items are textures, while backgroundRegion, ball and cannon are all TextureRegions, which means the place the region of the texture that need to be put. Furthermore, bobMove and squirrelFly are animations, whose position will be updated while the game is running. The Assets class also includes other parameters, but they are not the main parts and the types are the same as what are listed.

The following code is the assets-class code.

```
public class Assets {
            public static Texture background;
            public static TextureRegion backgroundRegion;
            public static Texture atlas;
            public static TextureRegion cannon;
            public static TextureRegion ball;

            public static Animation squirrelFly;
            public static Animation bobMove;
            public static Texture items;
            public static TextureRegion mainMenu;
            public static TextureRegion pauseMenu;
            public static TextureRegion ready;
            public static TextureRegion gameOver;
            public static TextureRegion highScoresRegion;
            public static TextureRegion soundOn;
            public static TextureRegion soundOff;
            public static TextureRegion arrow;
            public static TextureRegion pause;
            public static TextureRegion squirrel;
            public static TextureRegion bob;
            public static TextureRegion bobHit;
            public static TextureRegion bobFast;
            public static Font font;
            public static Music music;
            public static Sound hitSound;
            public static Sound clickSound;

            /*
             * The load() method, which will be called once at the
             * start of our game, is responsible forpopulating all the
             * static members of the class.
             */
            public static void load(GLGame game) {
               background = new Texture(game, "background.png");
               backgroundRegion = new TextureRegion(background,
                           0, 0, 320, 480);
               atlas = new Texture(game, "atlas.png");
               cannon = new TextureRegion(atlas, 0, 0, 64, 32);

               items = new Texture(game, "items.png");
               ball = new TextureRegion(items, 128, 160, 24, 24);
               bob = new TextureRegion(items, 64, 128, 32, 32);

               bobMove = new Animation(0.2f,
                 new TextureRegion(items, 64, 128, 32, 32),
                 new TextureRegion(items, 32, 128, 32, 32));

               squirrelFly = new Animation(0.2f,
                 new TextureRegion(items, 0, 160, 32, 32),
                 new TextureRegion(items, 32, 160, 32, 32));

               mainMenu = new TextureRegion(items, 0, 224, 300, 110);
               pauseMenu = new TextureRegion(items, 224, 128, 192, 96);

               ready = new TextureRegion(items, 320, 224, 192, 32);
               gameOver = new TextureRegion(items, 352, 256, 160, 96);
               highScoresRegion = new TextureRegion(Assets.items, 0, 257,
                 300, 110 / 3);

               soundOff = new TextureRegion(items, 0, 0, 64, 64);
               soundOn = new TextureRegion(items, 64, 0, 64, 64);
               arrow = new TextureRegion(items, 0, 64, 64, 64);
               pause = new TextureRegion(items, 64, 64, 64, 64);

               squirrel = new TextureRegion(items, 0, 160, 32, 32);

               font = new Font(items, 224, 0, 16, 16, 20);

               music = game.getAudio().newMusic("music.mp3");
               music.setLooping(true);
               music.setVolume(0.5f);
               if(Settings.soundEnabled)
                  music.play();
               hitSound = game.getAudio().newSound("hit.ogg");
               clickSound = game.getAudio().newSound("click.ogg");
            }

           /*
            * The OpenGL ES context will get lost when our
            * application is paused. We have to reload the textures
            * when the application is resumed.
            */
            public static void reload() {
               background.reload();
```

```
                items.reload();
                if(Settings.soundEnabled)
                    music.play();
            }

           /*
            * The final method of this class is a helper method we'll
            * use in the rest of the code to play back audio. Instead
            * of having to check whether sound is enabled
            * everywhere, we encapsulate that check in this method.
            */
            public static void playSound(Sound sound) {
                if(Settings.soundEnabled)
                    sound.play(1);
            }
}
```

Snippet 1. Defining assets

4.2 The Settings Class

The settings-class is responsible for the loading the saving of the file. The scores the player gets will be saved into a file named ".cannon" in the file folder "sdcard".

The Settings class has three methods:

- load(): loads the saving file
- save(): saves scores into the file
- addScore(): adds the new high score into the file

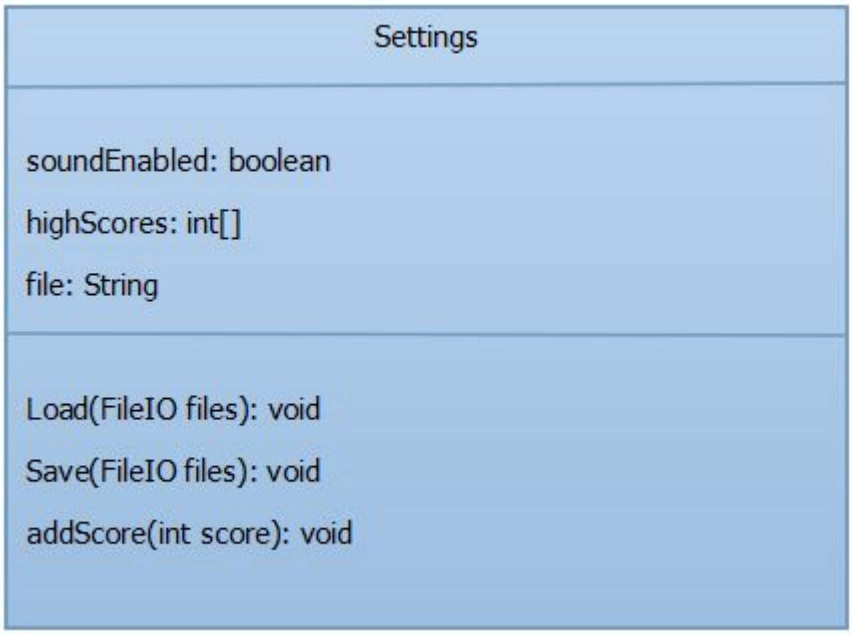

Figure 13. Settings Class

The int array highScores saves the highest five scores the player got in the game, and saves this information into the ".cannon" file by the method save().

The following is the code of Setting class.

```
public class Settings {
        public static boolean soundEnabled = true;
        public final static int[] highscores = new int[] { 100, 80, 50, 30, 10 };
        //The path of the file is /sdcard/.cannon
        public final static String file = ".cannon";
        public static void load(FileIO files) {
                BufferedReader in = null;
                try {
                        In=newBufferedReader(new
                                InputStreamReader(files.readFile(file)));
                        soundEnabled = Boolean.parseBoolean(in.readLine());
                        for(int i = 0; i < 5; i++) {
                        highscores[i] = Integer.parseInt(in.readLine());
                        }
                } catch (IOException e) {
                        // :( It's ok we have defaults
                } catch (NumberFormatException e) {
                        // :/ It's ok, defaults save our day
                } finally {
```

```
                            try {
                                    if (in != null)
                                in.close();
                    } catch (IOException e) {
            }
        }
    }

    public static void save(FileIO files) {
        BufferedWriter out = null;
        try {
            out = new BufferedWriter(new OutputStreamWriter(files.writeFile(file)));
            out.write(Boolean.toString(soundEnabled));
            out.write("\n");
            for(int i = 0; i < 5; i++) {
                out.write(Integer.toString(highscores[i]));
                out.write("\n");
            }
        } catch (IOException e) {
        } finally {
            try {
                if (out != null)
                    out.close();
            } catch (IOException e) {
            }
        }
    }

    public static void addScore(int score) {
        for(int i=0; i < 5; i++) {
            if(highscores[i] < score) {
                for(int j= 4; j > i; j--)
                    highscores[j] = highscores[j-1];
                highscores[i] = score;
                break;
            }
        }
    }
}
```

Snippet 2. Settings-class code.

4.3 The Main Activity

To enter the game, we need a start point, and the class CannonShootGame will provide this start point.

The CannonShootGame class also has three methods:

- getStartScreen(): provides the start point of the game
- onSurfaceCreated(): initializes the materials the game need to use
- onPause(): pauses the music

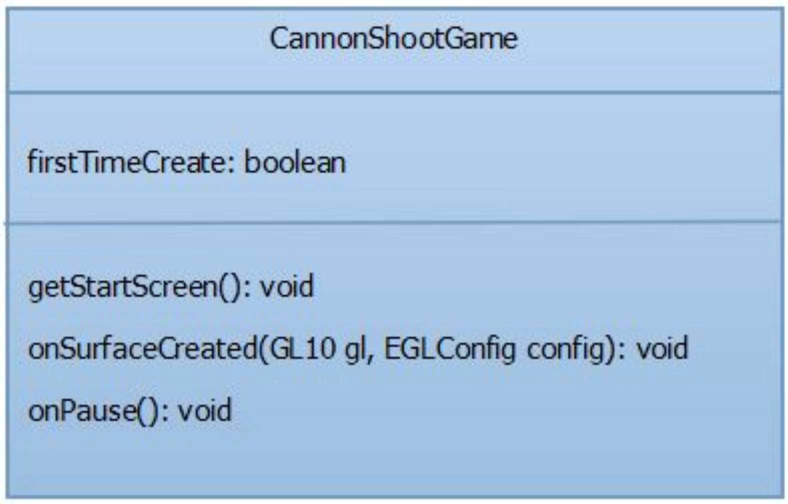

Figure 14. CannonShootGame Class

The parameter firstTimeCreate is used to judge if the game is created for the first time created. If it is, then the file and the assets are loaded; if it is not, then just the assets are reloaded.

The following is the code of CannonShootGame class.

```
public class CannonShootGame extends GLGame {
   boolean firstTimeCreate = true;
   //Setting the starting screen as main menu screen
   public Screen getStartScreen() {
      return new MainMenuScreen(this);
   }

   @Override
   public void onSurfaceCreated(GL10 gl, EGLConfig config) {
      super.onSurfaceCreated(gl, config);
      if(firstTimeCreate) {
            Settings.load(getFileIO());
            Assets.load(this);
            firstTimeCreate = false;
        } else {
            Assets.reload();
        }
    }

    @Override
    public void onPause() {
        super.onPause();
        if(Settings.soundEnabled)
            Assets.music.pause();
    }
}
```

Snippet 3. CannonShootGame-class code

4.4 The MainMenuScreen

This screen is returned by CannonShootGame.getStartScreen(), so this screen is the first screen when the game starts. It shows the starting menu of the game and provides the interfaces to other screens, including the game screen, the high scores screen, as well as the help screen.

The MainMenuScreen has four methods:

- MainMenuScreen(): constructor method responsible for setting up all of the members
- update(): monitors the touch events
- present(): renders the background and UI elements
- pause(): makes sure the settings are saved into the SD card

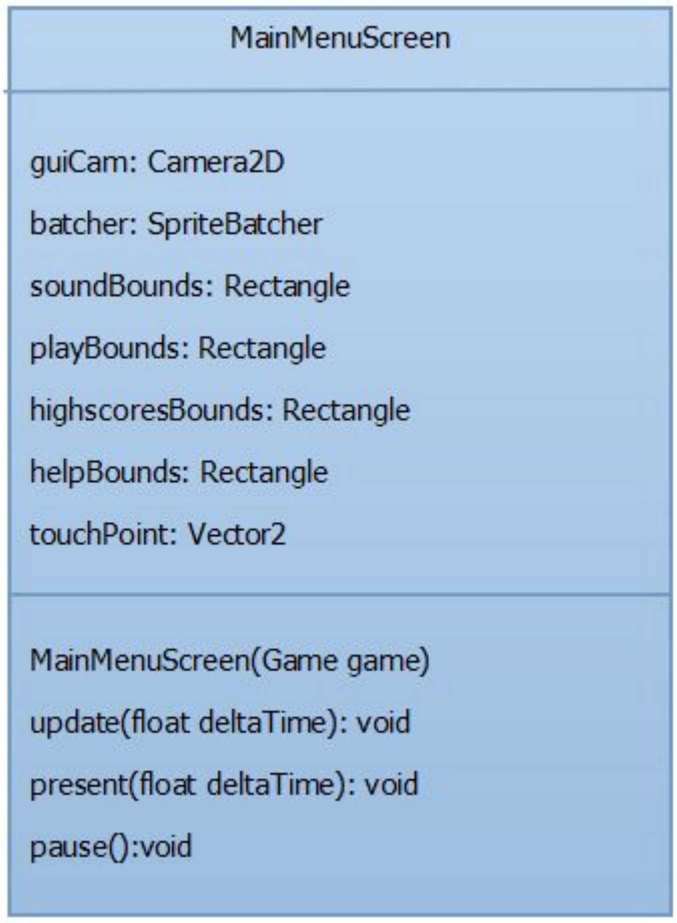

Figure 15. MainMenuScreen Class

Rectangles are used to determine if the user touches a UI element. Since the Camera2D is used, the Vector2 instance is needed to transfer the touch coordinates to world coordinates.

The following is the MainMenuScreen-class code.

```
public class MainMenuScreen extends GLScreen {
    Camera2D guiCam;
    SpriteBatcher batcher;
    Rectangle soundBounds;
    Rectangle playBounds;
    Rectangle highscoresBounds;
    Rectangle helpBounds;
    Vector2 touchPoint;
```

```
public MainMenuScreen(Game game) {
   super(game);

   guiCam = new Camera2D(glGraphics, 320, 480);
   batcher = new SpriteBatcher(glGraphics, 100);
   soundBounds = new Rectangle(0, 0, 64, 64);
   playBounds = new Rectangle(160 - 150, 200 + 18, 300, 36);
   highscoresBounds = new Rectangle(160 - 150, 200 - 18, 300, 36);
   helpBounds = new Rectangle(160 - 150, 200 - 18 - 36, 300, 36);
   touchPoint = new Vector2();
}

@Override
/*
 *  We loop through the TouchEvents returned by our Input
 *  instance and check for touch-up events. In case we have such an
 *  event, we first translate the touch coordinates to world coordinates.
 *  Since the camera is set up in a way so that we work in our target
 *  resolution, this transformation boils down simply to flipping the y-
 *  coordinate on a 320  480 pixel screen.
 */
public void update(float deltaTime) {
   List<TouchEvent> touchEvents =
              game.getInput().getTouchEvents();
   game.getInput().getKeyEvents();

   int len = touchEvents.size();
   for(int i = 0; i < len; i++) {
      TouchEvent event = touchEvents.get(i);
      if(event.type == TouchEvent.TOUCH_UP) {
         touchPoint.set(event.x, event.y);
         guiCam.touchToWorld(touchPoint);
         //Press play button to play the game
         if(OverlapTester.pointInRectangle(playBounds, touchPoint)) {
            Assets.playSound(Assets.clickSound);
            game.setScreen(new GameScreen(game));
            return;
         }
         //Turn to the high scores screen
         if(OverlapTester.pointInRectangle(highscoresBounds,
              touchPoint))         {
            Assets.playSound(Assets.clickSound);
            game.setScreen(new HighscoresScreen(game));
            return;
         }
         //Turn to the help screen
         if(OverlapTester.pointInRectangle(helpBounds, touchPoint)) {
            Assets.playSound(Assets.clickSound);
            game.setScreen(new HelpScreen(game));
            return;
        }
        //Turn on/off the sound
        if(OverlapTester.pointInRectangle(soundBounds, touchPoint)) {
            Assets.playSound(Assets.clickSound);
            Settings.soundEnabled = !Settings.soundEnabled;
            if(Settings.soundEnabled)
                Assets.music.play();
            else
                Assets.music.pause();
        }
    }
  }
}

@Override
/*
 * We clear the screen, set up the projection matrices via the camera,
 * and render the background and UI elements.
 * Note that the UI elements are rendered in a coordinate system with
 * the origin in the lower left of the screen and the y-axis pointing
 * upward.
 */
public void present(float deltaTime) {
    GL10 gl = glGraphics.getGL();
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
    guiCam.setViewportAndMatrices();
    //Enable rendering the texture
    gl.glEnable(GL10.GL_TEXTURE_2D);
    //Draw background
    batcher.beginBatch(Assets.background);
    batcher.drawSprite(160, 240, 320, 480, Assets.backgroundRegion);
    batcher.endBatch();

    gl.glEnable(GL10.GL_BLEND);
    gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
    batcher.beginBatch(Assets.items);
    //Draw the buttons on the background
    batcher.drawSprite(160, 200, 300, 110, Assets.mainMenu);
    batcher.drawSprite(32, 32, 64, 64, Settings.soundEnabled?
      Assets.soundOn:Assets.soundOff);

    batcher.endBatch();

    gl.glDisable(GL10.GL_BLEND);
}

@Override
/*
```

```
     * we make sure that the settings are saved to the SD card since the
     * user can change the sound settings on this screen.
     */
    public void pause() {
        Settings.save(game.getFileIO());
    }
}
```

Snippet 4. MainMenScreen-class code

4.5 The Help Screens

There are total of two help screens which all work the same: to load the help screen image, to render it along with the arrow button, and wait for a touch of the arrow button to move to the next screen.

There are four functions in the HelpScreen class:

- HelpScreen(): constructor method
- resume(): loads the actual help screen texture and creates a corresponding TextureRegion for rendering with the SpriteBatcher
- update(): simply checks whether the arrow button is pressed
- present(): clears the screen, sets up the matrices, renders the help image in one batcher, and then renders the arrow button

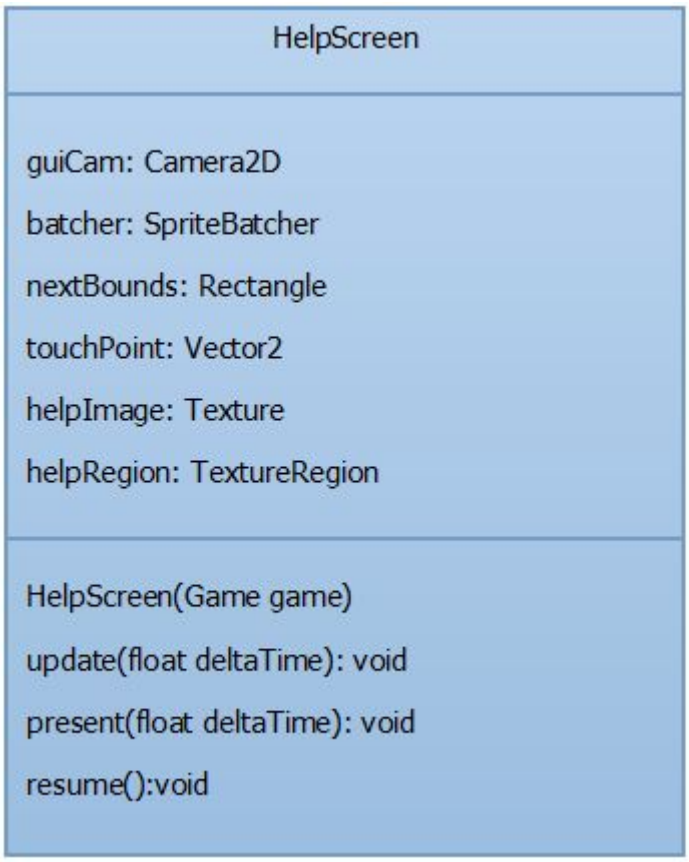

Figure 16. HelpScreen Class

The only thing where the help screens differ is the image that they each load and the screen to which they transit. So here only present the code of the first help screen is given.

The following is the HelpScreen-class code.

```
public class HelpScreen extends GLScreen {
    Camera2D guiCam;
    SpriteBatcher batcher;
    Rectangle nextBounds;
    Vector2 touchPoint;
    Texture helpImage;
    TextureRegion helpRegion;

    public HelpScreen(Game game) {
        super(game);
        guiCam = new Camera2D(glGraphics, 320, 480);
        nextBounds = new Rectangle(320 - 64, 0, 64, 64);
        touchPoint = new Vector2();
        batcher = new SpriteBatcher(glGraphics, 1);
    }

    @Override
    public void resume() {
        helpImage = new Texture(glGame, "help1.png" );
        helpRegion = new TextureRegion(helpImage, 0, 0, 320, 480);
    }

    @Override
    public void pause() {
        helpImage.dispose();
    }

    @Override
    public void update(float deltaTime) {
        List<TouchEvent> touchEvents = game.getInput().getTouchEvents();
        game.getInput().getKeyEvents();
        int len = touchEvents.size();
        for(int i = 0; i < len; i++) {
            TouchEvent event = touchEvents.get(i);
            touchPoint.set(event.x, event.y);
            guiCam.touchToWorld(touchPoint);
            //If touch the arrow button, then turn to the next help screen
            if(event.type == TouchEvent.TOUCH_UP) {
                if(OverlapTester.pointInRectangle(nextBounds, touchPoint))
                {
                    Assets.playSound(Assets.clickSound);
                    game.setScreen(new HelpScreen2(game));
                    return;
                }
            }
        }
    }

    @Override
    public void present(float deltaTime) {
        GL10 gl = glGraphics.getGL();
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
        guiCam.setViewportAndMatrices();
        //Render the help image
        gl.glEnable(GL10.GL_TEXTURE_2D);

        batcher.beginBatch(helpImage);
        batcher.drawSprite(160, 240, 320, 480, helpRegion);
        batcher.endBatch();
```

```
        //Render the arrow button
        gl.glEnable(GL10.GL_BLEND);
        gl.glBlendFunc(GL10.GL_SRC_ALPHA,
                GL10.GL_ONE_MINUS_SRC_ALPHA);

        batcher.beginBatch(Assets.items);
        batcher.drawSprite(320 - 32, 32, -64, 64, Assets.arrow);
        batcher.endBatch();

        gl.glDisable(GL10.GL_BLEND);
    }
}
```

Snippet 5. HelpScreen-class code

4.6 The High-Scores Screen

This screen is used to view the highest five history scores the player got in the game. So this class will renders the high scores saved in Settings. And also, there will be an arrow button for the player to get back to the main menu.

The class has three methods:

- HighScoreScreen(): constructor for the initialization of all the parameters
- update(): checks whether the arrow button was pressed, then transit to the main menu screen
- present(): clears the screen, sets the matrices, render the background, renders the high scores

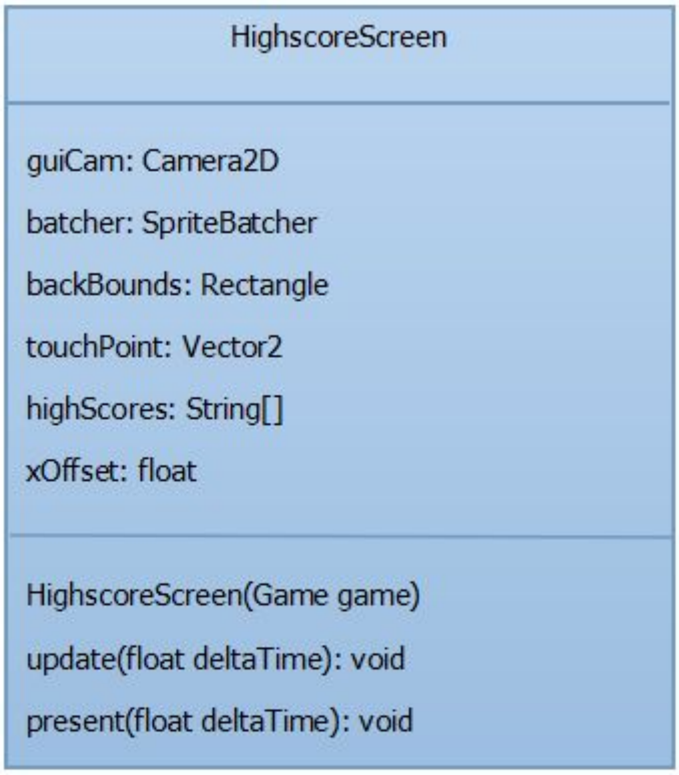

Figure 17. HighscoreScreen Class

The parameter xOffset is a value which is computed to offset the rendering of each line so that the lines are centered horizontally. In the constructors, besides all members will be set up, xOffset value is computed, which is done by evaluating the longest string out of the five strings created for the highest five scores. Since the bitmap font is fixed-width, it is easy to calculate the number of pixels needed for a single line of text by multiplying the number of characters with the width.

The following is the HighscoreScreen-class code.

```
public class HighscoreScreen extends GLScreen {
	Camera2D guiCam;
	SpriteBatcher batcher;
	Rectangle backBounds;
	Vector2 touchPoint;
	String[] highScores;
	float xOffset = 0;
```

```
public HighscoreScreen(Game game) {
    super(game);

    guiCam = new Camera2D(glGraphics, 320, 480);
    backBounds = new Rectangle(0, 0, 64, 64);
    touchPoint = new Vector2();
    batcher = new SpriteBatcher(glGraphics, 100);
    highScores = new String[5];
    for(int i = 0; i < 5; i++) {
        highScores[i] = (i + 1) + ". " + Settings.highscores[i];
        xOffset = Math.max(highScores[i].length() *Assets.font.glyphWidth, xOffset);
    }
    //Subtracts half of the longest line width from 160 (the horizontal
    //center of our target screen of 320  480 pixels) and adjusts it
    //further by subtracting half of the width.
    xOffset = 160 - xOffset / 2;
}

@Override
public void update(float deltaTime) {
    List<TouchEvent> touchEvents = game.getInput().getTouchEvents();
    game.getInput().getKeyEvents();
    int len = touchEvents.size();
    for(int i = 0; i < len; i++) {
        TouchEvent event = touchEvents.get(i);
        touchPoint.set(event.x, event.y);
        //Transfer the coordinates to the world coordinates
        guiCam.touchToWorld(touchPoint);
        //If touch the arrow button, then return to the main menu screen
        if(event.type == TouchEvent.TOUCH_UP) {
            if(OverlapTester.pointInRectangle(backBounds, touchPoint)) {
                game.setScreen(new MainMenu(game));
                return;
            }
        }
    }
}

@Override
public void present(float deltaTime) {
    GL10 gl = glGraphics.getGL();
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
    guiCam.setViewportAndMatrices();

    gl.glEnable(GL10.GL_TEXTURE_2D);
    //Render the background
    batcher.beginBatch(Assets.background);
    batcher.drawSprite(160, 240, 320, 480, Assets.backgroundRegion);
    batcher.endBatch();
```

```
        gl.glEnable(GL10.GL_BLEND);
        gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
        //Render the strngs of scores
        batcher.beginBatch(Assets.items);
        batcher.drawSprite(160, 360, 300, 33, Assets.highScoresRegion);
        //Output the scores stored in the array highScores
        float y = 240;
        for(int i = 4; i >= 0; i--) {
            Assets.font.drawText(batcher, highScores[i], xOffset, y);
            y += Assets.font.glyphHeight;
        }
        //Render the arrow button
        batcher.drawSprite(32, 32, 64, 64, Assets.arrow);
        batcher.endBatch();

        gl.glDisable(GL10.GL_BLEND);
    }
}
```

Snippet 6. HighscoreScreen-class code

4.7 The Simulation Classes

There are several objects in the game, and each of them is achieved by the simulation classes.

- Ball
- Bob
- Cannon
- Squirrel

In which, bob, squirrel and ball can move, while cannon is static. These classes are responsible for the manipulation of the characters in the game: to store the position, velocity, the state and the length of time the object has been in that state, bound shape of the object and so on.

4.7.1 The Ball Class

The cannon ball class is initialized first at the original point, the lower-left point of the screen, and it gets the initial speed according to the player's touch on the screen. So the ball class needs to provide a method to initial the velocity. At the same time, the movement of the ball is influenced by the gravity, so it will have a gravity simulation. As for the collision detection, it is achieved in the world class.

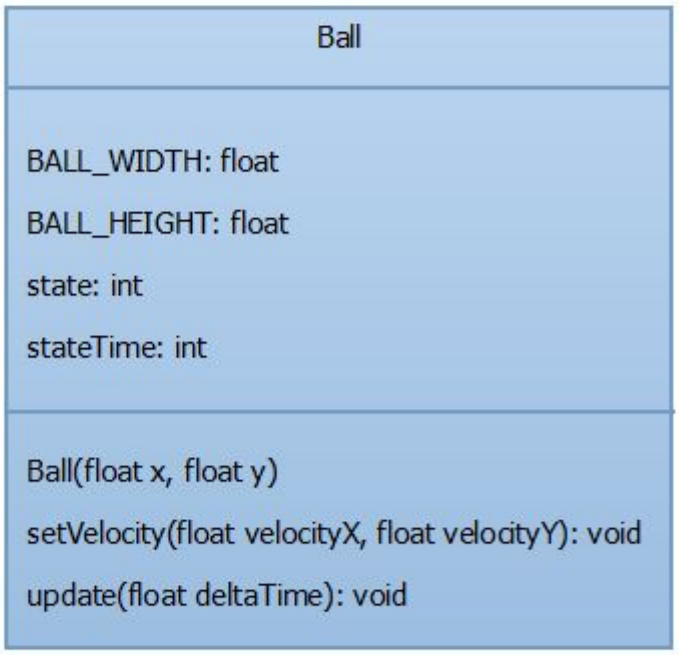

Figure 18. Ball Class

The parameters, BALL_WIDTH and BALL_HEGHT, are used to define the width and height of the ball, and they are static final parameters. The method setVelocity() is for receiving the initial velocities of x and y axis and giving them to the ball. The velocity initialization process is achieved in the GameScreen class.

The following is the Ball-class code.

```
public class Ball extends DynamicGameObject{
   public static final float BALL_WIDTH = 0.5f;
   public static final float BALL_HEIGHT = 0.5f;
```

```
    int state;
    float stateTime;

    public Ball(float x, float y) {
       super(x, y, BALL_WIDTH, BALL_HEIGHT);
       state = BALL_FLY;
       stateTime = 0;
    }
    //Initialize the velocity of the ball
    public void setVelocity(float velocityX, float velocityY) {
         velocity.set(velocityX, velocityY);
    }

      public void update(float deltaTime) {
        //Update the velocity by gravity
       velocity.add(World.gravity.x * deltaTime, World.gravity.y * deltaTime);
       //Update the position by velocity
       position.add(velocity.x * deltaTime, velocity.y * deltaTime);
       //Bound the rectangle to ball
       bounds.lowerLeft.set(position).sub(bounds.width / 2, bounds.height / 2);

       stateTime += deltaTime;
    }
}
```

Snippet 7. Ball-class code.

4.7.2 The Bob Class

The Bob is the target which the player need to shoot during the game. Bobs move from the left of the screen to the right automatically.

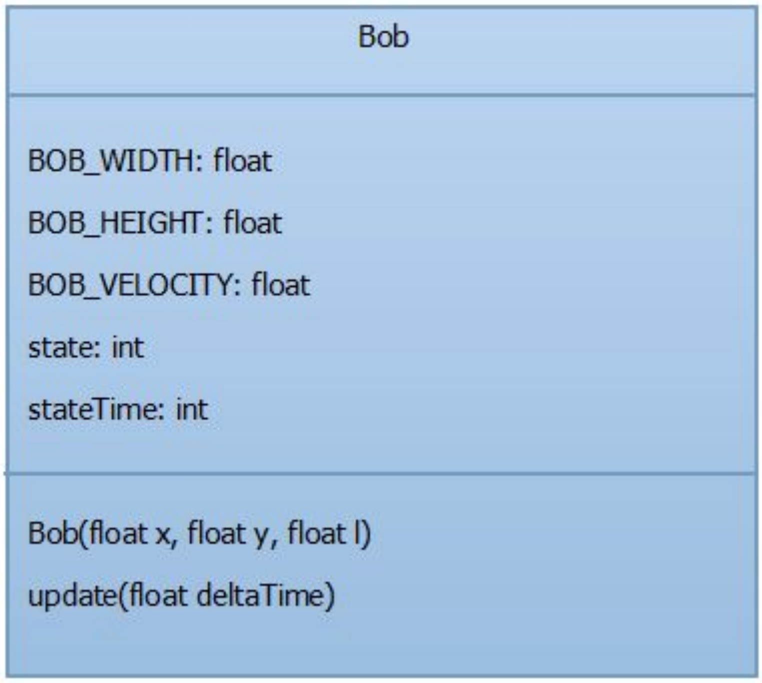

Figure 19. Bob Class

In the constructor, the parameters, x and y, means the position coordinates of bob, and the parameter, l, represents for the level, The velocity of the bobs is increased as the level becomes higher. So the difficulty of the game will increase. The static parameter BOB_VELOCITY is the base velocity of the bob.

The following is the Bob-class code.

```
public class Bob extends DynamicGameObject{
    public static final float BOB_WIDTH = 0.8f;
    public static final float BOB_HEIGHT = 0.8f;
    public static final float BOB_VELOCITY = 2f;

    int state;
    float stateTime;

    public Bob(float x, float y, float l) {
        super(x, y, BOB_WIDTH, BOB_HEIGHT);
        velocity.set((l - 1f) + BOB_VELOCITY, 0);
        stateTime = 0;
    }
    public void update(float deltaTime) {
        //Update the position by velocity
        position.add(velocity.x * deltaTime, velocity.y * deltaTime);
        //Bound the rectangle to bob
        bounds.lowerLeft.set(position).sub(BOB_WIDTH / 2, BOB_HEIGHT / 2);
        stateTime += deltaTime;
    }
}
```

Snippet 8. Bob-class code

4.7.3 The Cannon Class

The cannon class is responsible for initializing the cannon at the lower-left corner of the screen, as well as the starting shooting angle.The update of the cannon angle is achieved in the class GameScreen class.

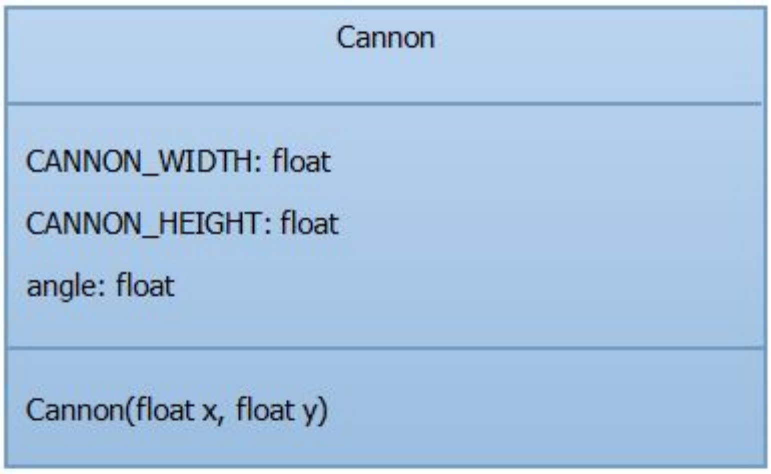

Figure 20. Cannon Class

Following is the Cannon-class code.

```
public class Cannon extends GameObject {
    public static final float CANNON_WIDTH = 4f;
    public static final float CANNON_HEIGHT = 2f;
    public float angle;      //Shooting angle

    public Cannon(float x, float y) {
        super(x, y, CANNON_WIDTH, CANNON_HEIGHT);
        angle = 45;
    }
```

}

Snippet 9. Cannon-class code.

4.7.4 The Squirrel Class

The squirrel class is somewhat similar to the bob class.

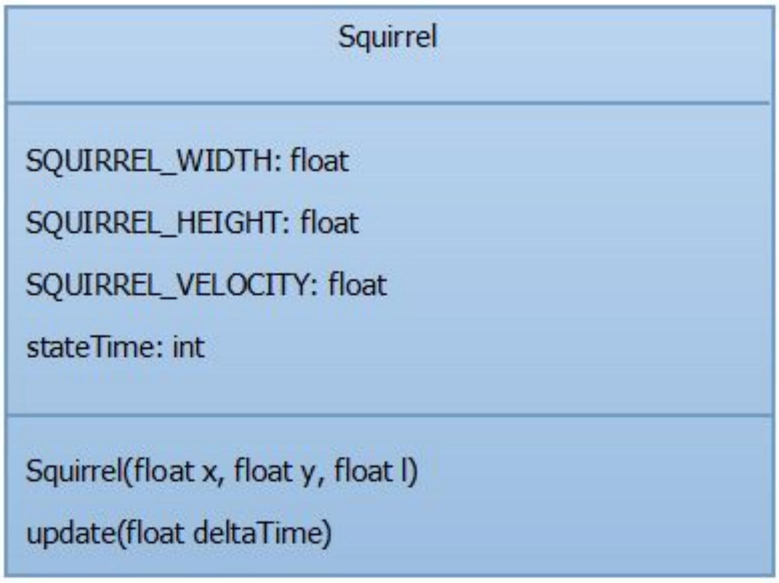

Figure 21. Squirrel Class

Following is the Squirrel-class code.

```
public class Squirrel extends DynamicGameObject {
   public static final float SQUIRREL_WIDTH = 1;
   public static final float SQUIRREL_HEIGHT = 0.6f;
   public static final float SQUIRREL_VELOCITY = 3f;

   float stateTime = 0;

   public Squirrel(float x, float y) {
      super(x, y, SQUIRREL_WIDTH, SQUIRREL_HEIGHT);
      velocity.set(SQUIRREL_VELOCITY, 0);
   }

     public void update(float deltaTime) {
      //Update the position by velocity
      position.add(velocity.x * deltaTime, velocity.y * deltaTime);
```

```
        //Bound the rectangle to squirrel
        bounds.lowerLeft.set(position).sub(SQUIRREL_WIDTH / 2, SQUIRREL_HEIGHT
                /2);
        stateTime += deltaTime;
    }
}
```

Snippet 10. Squirrel-class code.

4.7.5 The World Class

This class is the last class discussed in this chapter, and also the most important class.

The world class is responsible for generating the whole game world, including creating and placing the objects in the world. Furthermore, the class also processes the collision detection.

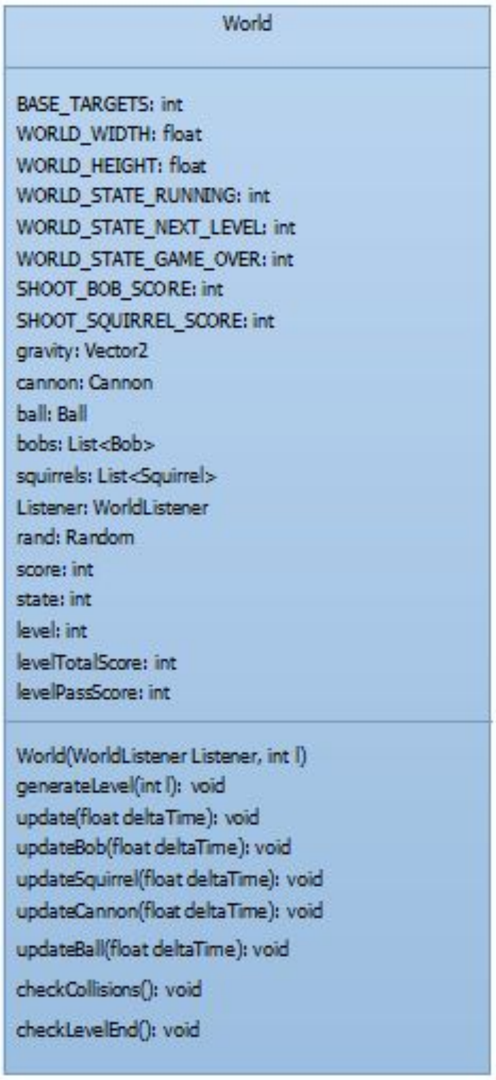
World
BASE_TARGETS: int
WORLD_WIDTH: float
WORLD_HEIGHT: float
WORLD_STATE_RUNNING: int
WORLD_STATE_NEXT_LEVEL: int
WORLD_STATE_GAME_OVER: int
SHOOT_BOB_SCORE: int
SHOOT_SQUIRREL_SCORE: int
gravity: Vector2
cannon: Cannon
ball: Ball
bobs: List<Bob>
squirrels: List<Squirrel>
Listener: WorldListener
rand: Random
score: int
state: int
level: int
levelTotalScore: int
levelPassScore: int
World(WorldListener Listener, int l)
generateLevel(int l): void
update(float deltaTime): void
updateBob(float deltaTime): void
updateSquirrel(float deltaTime): void
updateCannon(float deltaTime): void
updateBall(float deltaTime): void
checkCollisions(): void
checkLevelEnd(): void

Figure 22. World Class

4.7.5.1 World Generation

The generation process of bobs and squirrels is achieved in the method called generateLevel(), which is not the same in different levels.

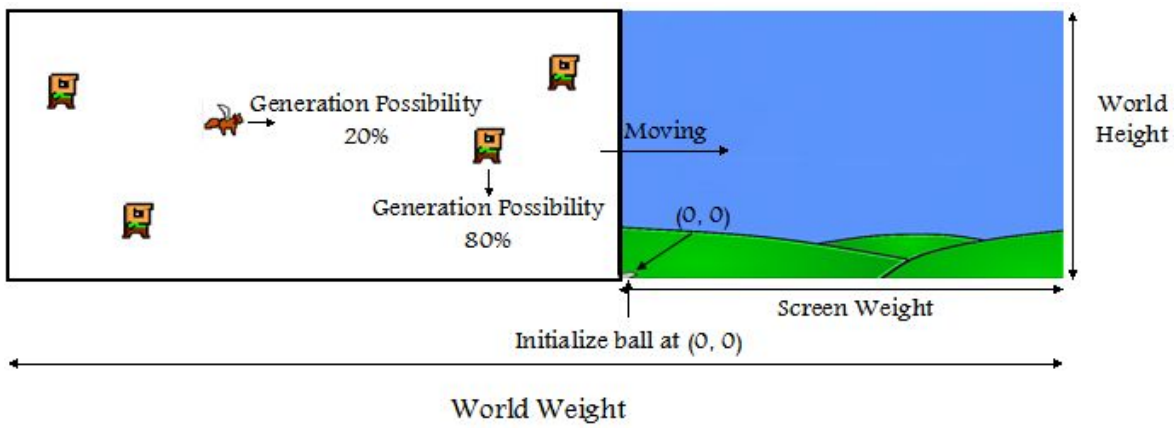

Figure 23. World Generation Process

As it can be seen in Figure 22, the method randomly generates the x and y positions of characters, bob and squirrel. The values of x are negative, so that the characters can be put left behind the screen. The generation possibility of bobs are 80%, squirrels are 20%. The number of all the moving characters and the velocity of bobs are generated according to the level, which are increased with the higher level. And after the game is began, all of the bobs and squirrels move from the left of the screen to right.

The following is the code of generateLevel().

```
private void generateLevel(int l) {
        for(int i = 0; i < BASE_TARGETS + (l - 1) * 5; i++) {
                //Generate Bobs in the world randomly according to the level
                if(rand.nextFloat() < 0.8f) {
                        float x = (float)Math.random() * (2 + l) * WORLD_WIDTH;
                        float y = (float)Math.random() * WORLD_HEIGHT * 4 / 5;
                        //The velocity of the bob will be increased in the
                        //higher level according to the level
                        Bob bob = new Bob(-x, y, l);
                        bobs.add(bob);
                }
                //Generate Squirrel, the number is according to the level
                else {
                float x = (float)Math.random() * (2 + l) * WORLD_WIDTH;
                float y = (float)Math.random() * WORLD_HEIGHT * 4 / 5;
                Squirrel squirrel = new Squirrel(-x, y);
                squirrels.add(squirrel);
        }
}
```

Snippet 11. The code of generateLevel().

In the method, the static parameter "BASE_TARGETS" represents the base number of the targets, and parameter "l" represents the number of level. It can be seen that the number of targets and the width of the world will be increased according to the value of "l". The increasing of the whole world width is to guarantee the generation frequency of the targes, so that the different interval of each target is not to big.

4.7.5.2 Collision Detection

The collision detection process is achieved in the method called checkCollisions().

Figure 24. Bound objects

This process has two steps. Firstly, bounding the characters with axis-aligned rectangles and then to check whether the two rectangles are overlapped. So what the method needs to do is to detect the overlapping between rectangles instead of the irregular shape, which will make the detection easier.

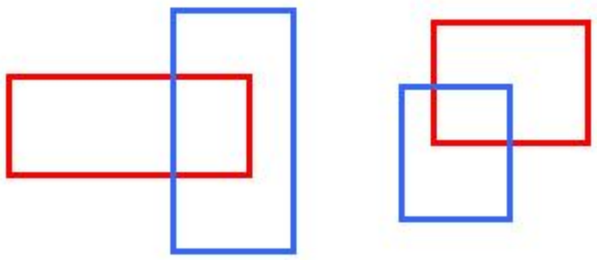

Figure 25. Two overlap situations

As if can be seen in the Figure 24, there are two situations when two rectangles overlap. The code in the OverlapTester class ensures the two situations.

The following is the code snippet of how to test the overlap.

```
public static boolean overlapRectangles(Rectangle r1, Rectangle r2) {
      if(r1.lowerLeft.x < r2.lowerLeft.x + r2.width &&
         r1.lowerLeft.x + r1.width > r2.lowerLeft.x &&
         r1.lowerLeft.y < r2.lowerLeft.y + r2.height &&
         r1.lowerLeft.y + r1.height > r2.lowerLeft.y)
          return true;
       else
          return false;
}
```

Snippet 12. Testing the overlap of two rectangles

The first two conditions ensure the first situation, and the last two conditions ensure the second one. The first condition states that the left edge of the first rectangle must be to the left of the right edge of the second rectangle. The next condition states that the right edge of the first rectangle must be to the right of the left edge of the second rectangle. The other two conditions state the same for the top and bottom edges of these two rectangles. If all these conditions are met, then the two rectangles overlap.

The following is the code of checkCollisions().

```
private void checkCollisions() {
         //Check if the ball collides the bob
         for(int i = 0; i < bobs.size(); i ++) {
                  Bob collider = bobs.get(i);
                  if(OverlapTester.overlapRectangles(ball.bounds,
                  collider.bounds)) {
                           bobs.remove(collider);
                           ball = new Ball(0, 0);
                           score += SHOOT_BOB_SCORE;
                  }
         }
         //Check the collision between ball and squirrel
         for (int i = 0; i < squirrels.size(); i++) {
               Squirrel squirrel = squirrels.get(i);
```

```
                if (OverlapTester.overlapRectangles(squirrel.bounds, ball.bounds)) {
                    squirrels.remove(squirrel);
                    ball = new Ball(0, 0);
                    score += SHOOT_SQUIRREL_SCORE;
                }
            }
        if(ball.position.x > 10) ball = new Ball(0, 0);
}
```

Snippet 13. The code of checkCollisions().

The method checks every bob and squirrel to judge if the rectangles of cannon ball and any target are overlapped. And if they are overlapped, then the target is removed from its list, renew the record is renewed and the ball is created at the original point (0, 0).

Method checkLevlEnd() also checks whether the game is over. If the present score of the player is less than the target score or the player has passed all of the levels, the game will be over.

The integral world-class code is presented in the appendices.

4.8 The Game Screen

The game screen is the core screen of the game, which will present the actual world to the player and allows it to interact with it. The game screen consists of five different screens, the ready screen, the normal running screen, the next-level screen, the game-over screen, and the pause screen.

Figure 26. Ready Screen

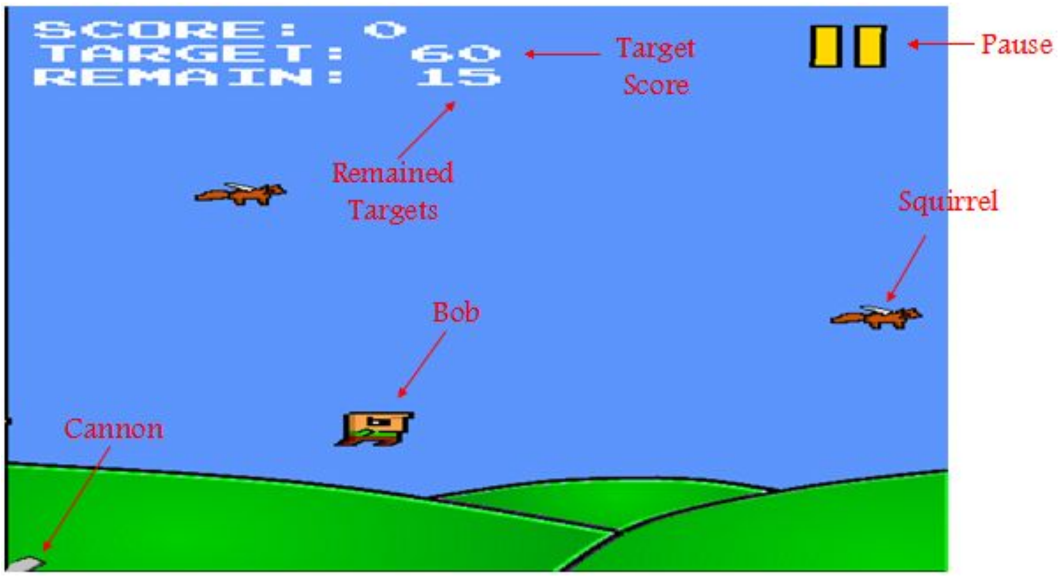

Figure 27. Normal-running Screen

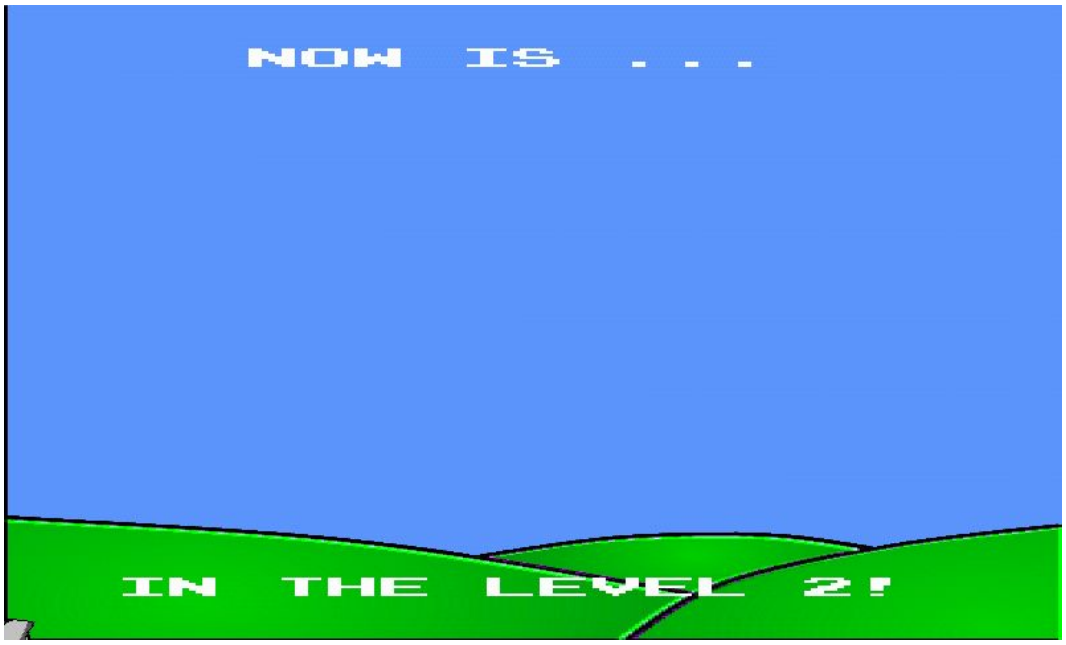

Figure 28. Next-level Screen

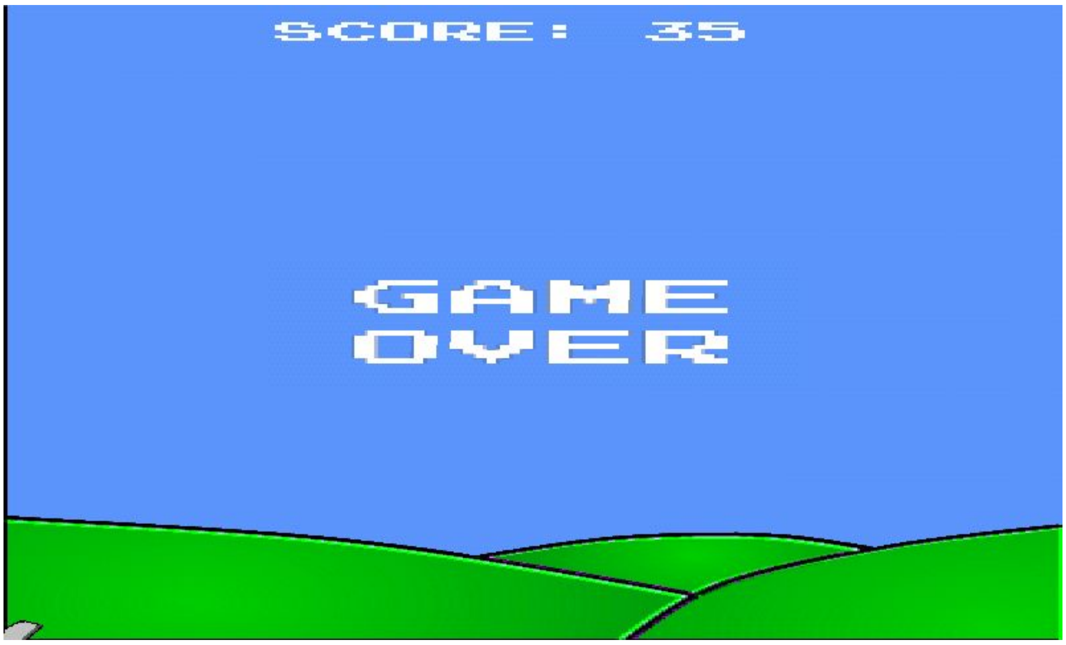

Figure 29. Game-over Screen

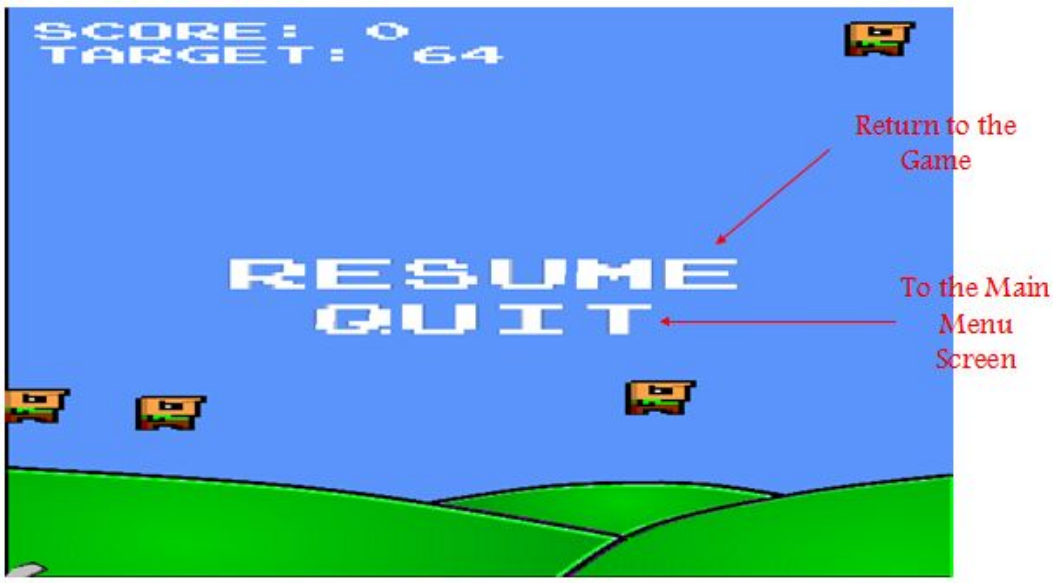

Figure 30. Pause Screen

The update and present methods for all subscreens that update and render the game world are separated, as well as the UI elements that are part of the subscreens.

The initialization of the speed of the ball is also achieved in the GameScreen class. The initial speed of the ball is determined by the position of touch point on the screen. So how to set the initial speed of the cannon ball?

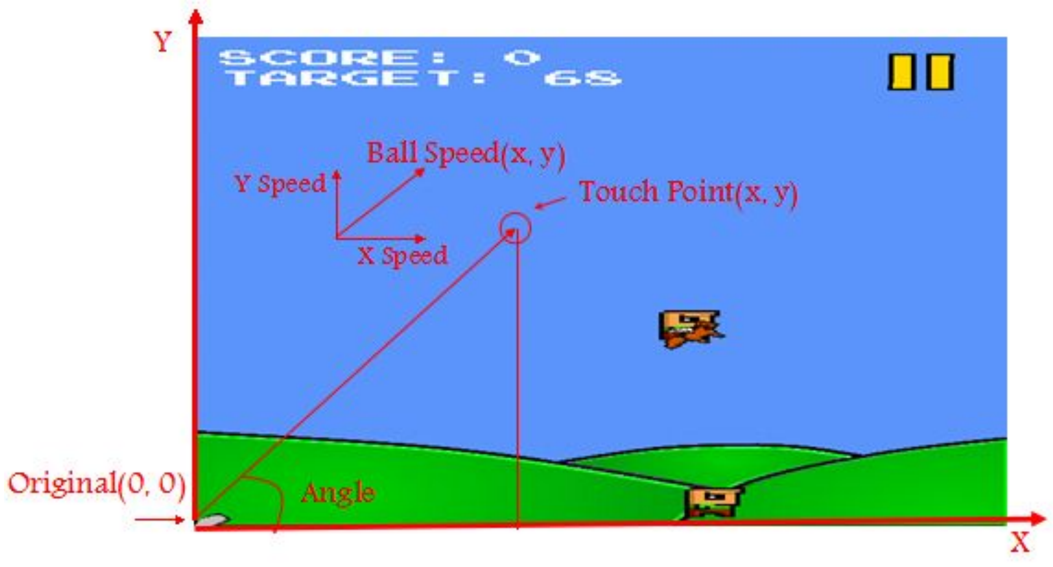

Figure 31. Initialize Cannon Ball Speed

The position of the point on the screen is expressed as a vector. The player uses one finger to touch a point on the screen, and then the application will receive the value of x and y position of the point, calculate the distance of the line between touch point and original point, as well as the angle between the x-axis and the line. Thus the shooting angle is the angle between the x-axis and the line and the shooting speed, which is also expressed as a vector, is determined by the value of the line's length. As in the real world, there also is a gravity working on the ball. So the y-axis value of the speed is decreased while the ball is flying. Then the application will updates the position of the ball according to the speed. So the ball will fly as a parabola instead of a straight line.

The following is the code of this process.

```
//update the angle of the cannon
world.cannon.angle = touchPoint.sub(world.cannon.position).angle();
//Initialize the speed
if(event.type == TouchEvent.TOUCH_UP) {
            float radians = world.cannon.angle * Vector2.TO_RADIANS;
             float ballSpeed = touchPoint.len() / 28;
             world.ball.position.set(world.cannon.position);
             float velocityX = FloatMath.cos(radians) * ballSpeed;
             float velocityY = FloatMath.sin(radians) * ballSpeed;
             //Set the speed to the ball
             world.ball.setVelocity(velocityX, velocityY);
             //Bound the rectangle to the ball
             world.ball.bounds.lowerLeft.set(world.ball.position.x - 0.1f,
                   world.ball.position.y - 0.1f);
}
```

Snippet 14. Set the angle and speed

It can be seen from the code that the application first gets the shooting angle of the cannon, which is saved in the cannon class, and then transforms the value of angle to Pi value. Calculating the speed according to the length, and the speed values in x and y direction is done by using the sin and cos value of the speed.

As a game for the touch-screen mobile phone, the most of the functions of the game need to be achieved by the touching.

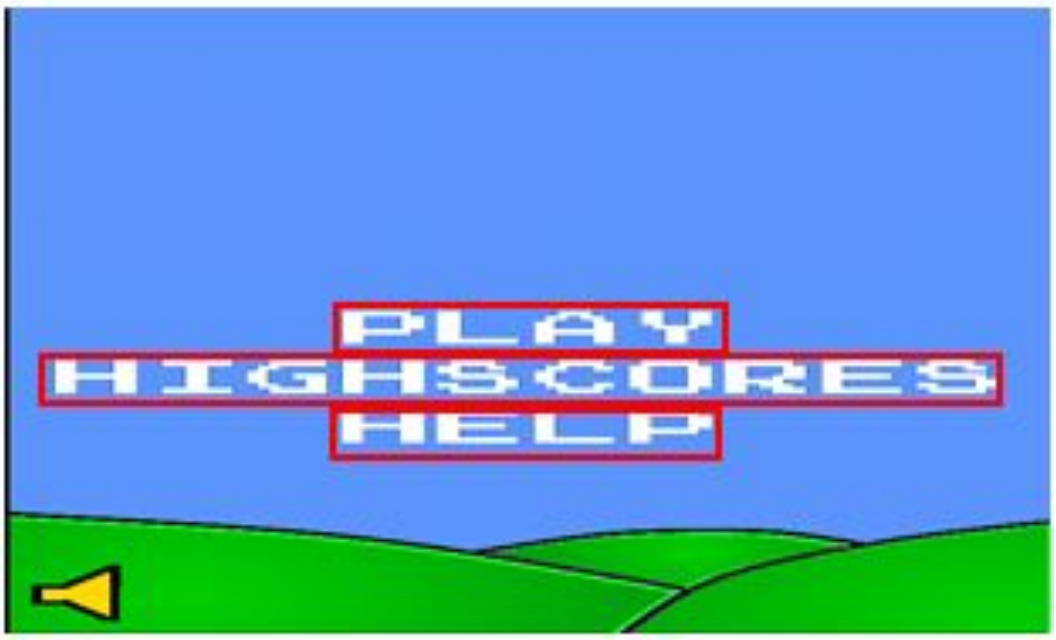

Figure 32. Touch Input

The principle of the touch input is similar to the collision detection. It also needs to bind rectangles to the buttons, and then detect whether the touch point is in any of the rectangles.

The whole is GameScreen-class code are presented in the appendices.

4.9 The WorldRender Class

This class is used to render the world.

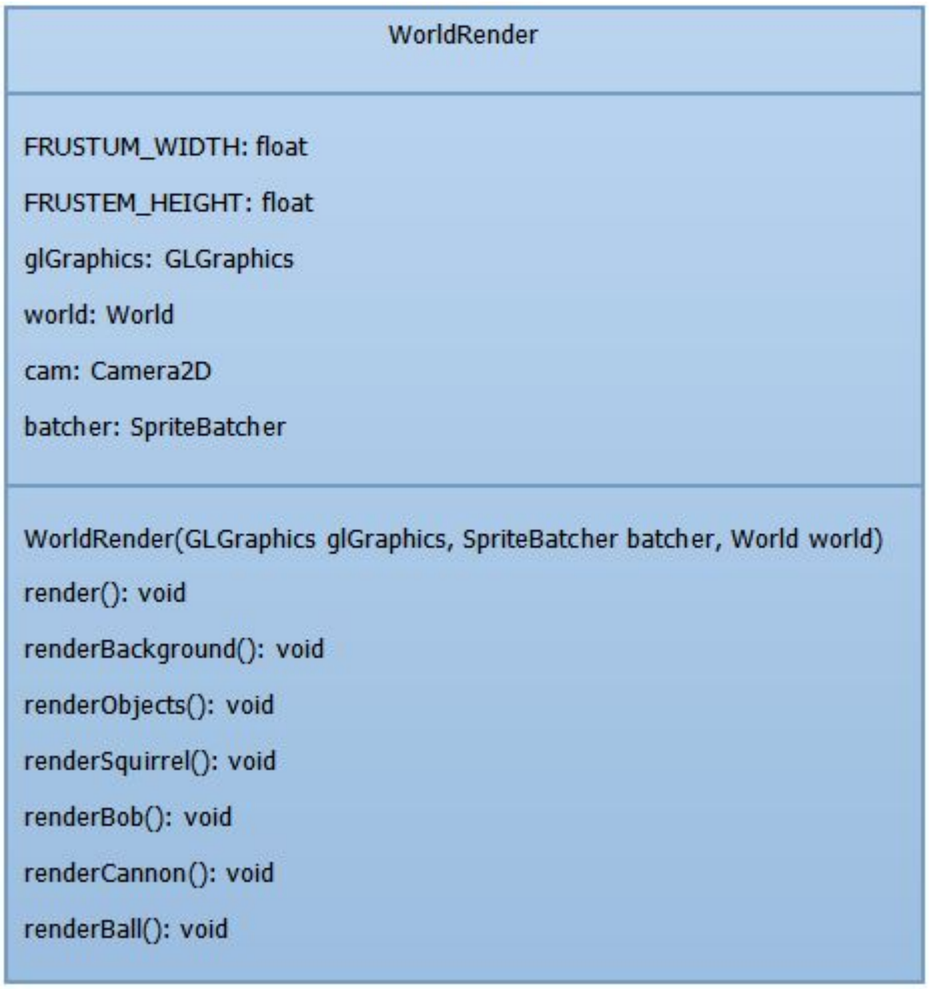

Figure 33. WorldRender Class

In this class, the render() method splits up the rendering into two batchers: background image and all the objects in the world. The camera is fixed because of the stationary background. The renderBackground() method simply render the background of the game on the screen. The renderObjects() method is responsible for rendering the objects in the world. Each object is rendered in a single batch. In the constructor of GameScreen class, the SpriteBatcher is initialized so that there are 1000 sprites in a single batch can be used, which is more than enough for the game world. Moreover, for each type of object, there are also separate rendering methods.

The following is the WorldRender-class code.

```
public class WorldRenderer {
   static final float FRUSTUM_WIDTH = 10;
   static final float FRUSTUM_HEIGHT = 15;
   GLGraphics glGraphics;
   World world;
   Camera2D cam;
   SpriteBatcher batcher;

   /*
    * In this case, it's the view frustum's width and height, which we
    * define as 10 and 15 meters. We also have a couple of members—
    * namely a GLGraphicsinstance, a camera, and the
    * SpriteBatcherreference we get from the game screen.
    */
     public WorldRenderer(GLGraphics glGraphics, SpriteBatcher batcher,
     World world) {
      this.glGraphics = glGraphics;
      this.world = world;
      this.cam = new Camera2D(glGraphics, FRUSTUM_WIDTH, FRUSTUM_HEIGHT);
      this.batcher = batcher;
   }

   /*
    * The render() method splits up rendering into two batches: one for
    * the background image and another one for all the objects in the
    * world. It also updates the camera position.
    */
   public void render() {
      //the position of the camera is the center of the world
      cam.position.x = FRUSTUM_WIDTH / 2;
      cam.position.y = FRUSTUM_HEIGHT / 2;
    cam.setViewportAndMatrices();
    renderBackground();
    renderObjects();
}

/*
 * The renderBackground()method simply renders the background so
 * that it follows the camera.
 */
public void renderBackground() {
    batcher.beginBatch(Assets.background);
    batcher.drawSprite(cam.position.x,cam.position.y, FRUSTUM_WIDTH,
       FRUSTUM_HEIGHT, Assets.backgroundRegion);
      batcher.endBatch();
}

/*
 * The renderObjects()method is responsible for rendering the second
 * batch.
 */
public void renderObjects() {
    GL10 gl = glGraphics.getGL();
    gl.glEnable(GL10.GL_BLEND);
    gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
    batcher.beginBatch(Assets.items);
    renderBob();
    renderBall();
    renderSquirrel();
    batcher.endBatch();
    renderCannon(gl);
    gl.glDisable(GL10.GL_BLEND);
}

//Render squirrels
public void renderSquirrel() {
       int len = world.squirrels.size();
    for(int i = 0; i < len; i++) {
        Squirrel squirrel = world.squirrels.get(i);
        TextureRegion keyFrame = Assets.squirrelFly.getKeyFrame(
                  squirrel.stateTime, Animation.ANIMATION_LOOPING);
        batcher.drawSprite(squirrel.position.x, squirrel.position.y, 1, 1, keyFrame);
    }
}

/*
 * The method renderBob()is responsible for rendering Bob.
 */
public void renderBob() {
        int len = world.bobs.size();
        for(int i = 0; i < len; i ++) {
                Bob bob = world.bobs.get(i);
                TextureRegion keyFrame = Assets.bobMove.getKeyFrame(
                        bob.stateTime, Animation.ANIMATION_LOOPING);
                batcher.drawSprite(bob.position.x, bob.position.y, 1, 1, keyFrame);
        }
    }
    //Bind the cannon texture with the cannon character
    public void renderCannon(GL10 gl) {
        Cannon cannon = world.cannon;
        Texture texture = Assets.atlas;
        Vertices cannonVertices = new Vertices(glGraphics, 4, 6, false, true);
        cannonVertices.setVertices(new float[] { -0.5f, -0.25f, 0.0f, 0.5f,
            0.5f, -0.25f, 1.0f, 0.5f,
            0.5f,  0.25f, 1.0f, 0.0f,
           -0.5f,  0.25f, 0.0f, 0.0f },
           0, 16);

        cannonVertices.setIndices(new short[] {0, 1, 2, 2, 3, 0}, 0, 6);

        gl.glLoadIdentity();
        gl.glTranslatef(cannon.position.x, cannon.position.y, 0);
        gl.glRotatef(cannon.angle, 0, 0, 1);

        texture.bind();
        cannonVertices.bind();
        cannonVertices.draw(GL10.GL_TRIANGLES, 0, 3);
        cannonVertices.unbind();
    }
    //Bind the ball texture with the ball character
    public void renderBall() {
        Ball ball = world.ball;
        TextureRegion keyFrame = Assets.ball;
        batcher.drawSprite(ball.position.x, ball.position.y, 0.5f, 0.5f,
                keyFrame);
    }
}
```

Snippet 15. WorldRender-class code

5 TESTING

The game testing was done by the Samsung GT-I8262D with Android version 4.1.2. During the development, after each function had been completed, the corresponding test was carried out immediately.

One significant problem during the game development was that the cannon could not shoot when the player touched the screen, and could not change the shooting angle, neither. Furthermore, when the last target passed by the screen, the game would end.

The point of these problem is that the texture used in the game should be put in the only one picture, or it will have the problem mentioned above.

6 CONCLUSION

Developing a game is like developing application to a certain extent. Every application development requires understanding the customer's need, discussing the application demands, designing the user interfaces, and programming and so on. However, game development also has higher requirements, such as image and sound decision. Game development is more about art and creation than the general application development.

The Snipe Bob is my first attempt to develop an integral game. Basically, it is a successful one. It has several levels for the player, and the difficulty of the game will be higher and higher as levels are increasing, which increases the interest of the game. Currently, Snip Bob has three levels and two types of targets. In the future, it can be extended to more levels and more target types.

The game is an Android mobile game. It ran smoothly during the test. The structure of the game is described in the book Beginning Android 4 Games Development[/6/], as well as some other sources mentioned in the book.

The most challenging part in developing this game was to solve the cannon's shooting problem. This problem was solved by calculating the values related to the touch point position. And one place needs to be attention is that all of the textures need to be in one and the same picture.

The experience with developing this game was valuable for studying Android application development and helped gain deeper understanding of Android applications.

REFERENCES

/1/ About the Android Open Source Project. Accessed 15 November 2010.
http://source.android.com/source/index.html

/2/ Anatomy Physiology of an Android.
https://sites.google.com/site/io/anatomy--physiology-of-an-android

/3/ Android issues reviewed. Accessed 08 August 2011.
http://code.google.com/p/android/issues/list?q=status%3AReviewed

/4/ Android Market reaches 500,000 app mark. Accessed 23 October 2011.
http://www.t3.com/news/android-market-reaches-500000-app-mark

/5/ Android Open Source Project Frequently Asked Questions: Compatibility. Source.android.com. Accessed13 March 2011.
http://source.android.com/faqs.html#compatibility

/6/ Beginning Android 4 Games Development. Mario Zechner, Robert Green.

/7/ Beyondbrics. Apple in China: not as cool as before. ft.com. Accessed
30 August 2012.
http://blogs.ft.com/beyond-brics/2012/08/24/apple-in-china-not-as-cool-as-before/

/8/ Devindra Hardawar. The magic moment: Smartphones now half of all U.S. mobiles. VentureBeat. VentureBeat MobileBeat. 29 March 2012. Accessed 27 June 2013.
http://venturebeat.com/2012/03/29/the-magic-moment-smartphones-now-half-of-all-u-s-mobiles/

/9/ Ganapati, Priya. Independent App Stores Take On Google’s Android Market. Wired News. June 11, 2010. Accessed 2 February 2011.
http://www.wired.com/gadgetlab/2010/06/independent-app-stores-take-on-googles-android-market/

/10/ Generation App 62% of Mobile Users 25-34 Own Smartphones. Nielson. The Nielsen Company. 11 March 2011. Accessed 27 June 2013.
http://www.nielsen.com/us/en/newswire/2011/generation-app-62-of-mobile-users-25-34-own-smartphones.html

/11/ Google Buys Android for Its Mobile Arsenal. Businessweek.com. 2005-08-17. Accessed 29 October 2010.
http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm

/12/ Kharif, Olga. Google's Android Gains More Powerful Followers. BusinessWeek. McGraw-Hill. 2008-12-09. Accessed 13 December 2008.
http://www.businessweek.com/the_thread/techbeat/archives/2008/12/googles_android_2.html

/13/ Lee Graham. As Smartphone Prices Fall, Retailers Are Leaving Money on the Table, According to the NPD Group. NPD Group. The NPD Group, Inc. 14 November 2011. Accessed 27 June 2013.
https://www.npd.com/wps/portal/npd/us/news/press-releases/pr_111114a/

/14/ List of Android Versions and their Features.
http://wethedevelopers.com/list-of-android-versions-and-their-features/

/15/ Paul, Ryan. Dream(sheep++): A developer's introduction to Google Android. Ars Technica. 23 February 2009. Accessed 7 March 2009.
http://arstechnica.com/open-source/reviews/2009/02/an-introduction-to-google-android-for-developers.ars

/16/ Patch/Feature Status Chart.
http://elinux.org/Android_Mainlining_Project#Patch.2FFeature_Status_Chart

/17/ Shankland, Stephen. Google's Android parts ways with Java industry group. CNET News. 12 November 2007.
http://www.cnet.com/news/googles-android-parts-ways-with-java-industry-group/

/18/ The Standard for Embedded Accelerated 3D Graphics.
http://www.khronos.org/opengles/

/19/ T-Mobile's G1 phone (HTC Dream), using Google's Android software.
http://zh.wikipedia.org/wiki/File:T-Mobile_G1_launch_event_2.jpg

APPENDIX 1

```
public class World {
        public interface WorldListener {
                public void shoot();

                public void hit();

    }
    /*
    * The WORLD_WIDTHand WORLD_HEIGHTspecify the extents of
    * our world horizontally and vertically.
    * Our view frustum will show a region of 10  15 meters of our
    * world.
    * our world will span 20 view frustums or screens horizontally.
    */
    public static final int BASE_TARGETS = 15;
    public static final float WORLD_WIDTH = 10;
    public static final float WORLD_HEIGHT = 15;
    public static final int WORLD_STATE_RUNNING = 0;
    public static final int WORLD_STATE_NEXT_LEVEL = 1;
    public static final int WORLD_STATE_GAME_OVER = 2;
    public static final Vector2 gravity = new Vector2(0, -10);

    public static final int SHOOT_BOB_SCORE = 10;
    public static final int SHOOT_SQUIRREL_SCORE = -5;

    /*
     * Next up are all the members of the Worldclass. It keeps track of
     * every characters in the game
     * Additionally, it has a reference to a WorldListenerand an instance of
     * random, which we'll use
     * to generate random numbers for various purposes.
     */
    public final Cannon cannon;
    public Ball ball;
    public final List<Bob> bobs;
    public final List<Squirrel> squirrels;
    public final WorldListener listener;
    public final Random rand;

    public int score;
    public int state;
    public int level;

    private int levelTotalScore;
    private int levelPassScore;

    /*
     * The constructor initializes all members and also stores the
     * WorldListener and level passed
     * as a parameter. Cannon is placed in the left-bottom corner of the
     * world at (0, 0).
     */
    public World(WorldListener listener, int l) {
       this.cannon = new Cannon(0, 0);
       this.ball = new Ball(0, 0);
       this.bobs = new ArrayList<Bob>();
       this.squirrels = new ArrayList<Squirrel>();
       this.listener = listener;
       rand = new Random();

       this.level = 1;
       generateLevel(l);

       this.score = 0;
       this.state = WORLD_STATE_RUNNING;
       //The total scores until the end of this level
       levelTotalScore = (BASE_TARGETS * l + l * (l - 1) * 5 / 2) * 10;
       //The pass score need to be the 2/5 of the total score
       levelPassScore = levelTotalScore * 2 / 5;
    }

    /*
     * Generate the objects in the world
     */
    private void generateLevel(int l) {
        for(int i = 0; i < BASE_TARGETS + (l - 1) * 5; i++) {

                //Generate Bobs in the world randomly
                if(rand.nextFloat() < 0.8f) {
                        float x = (float)Math.random() * (2 + l) *
                                WORLD_WIDTH;
                        float y = (float)Math.random() * WORLD_HEIGHT * 4
                                / 5;
```

```
                        //The velocity of the bob will be increased in the
                        //higher level.
                        Bob bob = new Bob(-x, y, l);
                        bobs.add(bob);
                }

        //Generate Squirrel
                else {
                float x = (float)Math.random() * (2 + l) * WORLD_WIDTH;
                float y = (float)Math.random() * WORLD_HEIGHT * 4 / 5;
                Squirrel squirrel = new Squirrel(-x, y);
                squirrels.add(squirrel);
        }

    }
}

public void update(float deltaTime) {
    updateBob(deltaTime);
    updateSquirrel(deltaTime);
    updateCannon(deltaTime);
    updateBall(deltaTime);

    if (ball.state != Ball.BALL_HIT)
      checkCollisions();
    checkLevelEnd();
}

private void updateBob(float deltaTime) {
      int len = bobs.size();
     for (int i = 0; i < len; i++) {
      Bob bob = bobs.get(i);
      bob.update(deltaTime);
    }
}

private void updateSquirrel(float deltaTime) {
      int len = squirrels.size();
                for (int i = 0; i < len; i++) {
                        Squirrel squirrel = squirrels.get(i);
                        squirrel.update(deltaTime);
                }
}
```

```
    private void updateCannon(float deltaTime) {
        cannon.update(deltaTime);
    }

    private void updateBall(float deltaTime) {
        ball.update(deltaTime);
    }

    private void checkCollisions() {
        //Check if the ball collide the bob
        for(int i = 0; i < bobs.size(); i ++) {
                Bob collider = bobs.get(i);
                if(OverlapTester.overlapRectangles(ball.bounds,
collider.bounds)) {
                        bobs.remove(collider);
                        ball = new Ball(0, 0);
                        score += SHOOT_BOB_SCORE;
                }
        }
        //Check the collision between ball and squirrel
        for (int i = 0; i < squirrels.size(); i++) {
               Squirrel squirrel = squirrels.get(i);
               if (OverlapTester.overlapRectangles(squirrel.bounds,
ball.bounds)) {
                squirrels.remove(squirrel);
                ball = new Ball(0, 0);
                score += SHOOT_SQUIRREL_SCORE;
               }
           }

        if(ball.position.x > 10) ball = new Ball(0, 0);
    }

      private void checkLevelEnd() {
      //if bob passed the screen then remove it from the list
        for(int i = 0; i < bobs.size(); i ++) {
                Bob bob = bobs.get(i);
                if(bob.position.x > 10)
                        bobs.remove(bob);
        }
        //if squirrel passed the screen then remove it from the list
        for(int i = 0; i < squirrels.size(); i ++) {
                Squirrel squirrel = squirrels.get(i);
                if(squirrel.position.x > 10)
                        squirrels.remove(squirrel);
        }
        //If all of targets have passed the screen and the final score is
        //bigger than the target score, then the game will turn into next
        //level.
        if(bobs.isEmpty() && squirrels.isEmpty()) {
                if(score >= levelPassScore) {
                        state = WORLD_STATE_NEXT_LEVEL;
                        level ++;
                }
//Check if the game is over. The condition is the present score is less than
//the target score, or all of the levels are passed.
                if(score < levelPassScore || level > 3) state =
WORLD_STATE_GAME_OVER;
        }
    }

    public int getLevelPassScore() {
        return levelPassScore;
    }
}
```

APPENDIX 2

```
public class GameScreen extends GLScreen{
    static final int GAME_READY = 0;
    static final int GAME_RUNNING = 1;
    static final int GAME_PAUSED = 2;
    static final int GAME_LEVEL_END = 3;
    static final int GAME_OVER = 4;
    int state;
    Camera2D guiCam;
    Vector2 touchPoint;
    SpriteBatcher batcher;
    World world;
    WorldListener worldListener;
    WorldRenderer renderer;
    Rectangle pauseBounds;
    Rectangle resumeBounds;
    Rectangle quitBounds;
    int lastScore;
    String scoreString;

    int lastLevel;
    int remainTargets;
    int remainBobs;
    int remainSquirrels;
    String targetString;
    String targetScoreString;
    FPSCounter fpsCounter;

    public GameScreen(Game game) {
          super(game);
          state = GAME_READY;
        guiCam = new Camera2D(glGraphics, 320, 480);
        touchPoint = new Vector2();
        batcher = new SpriteBatcher(glGraphics, 1000);
        worldListener = new WorldListener() {
          public void shoot() {
                   Assets.playSound(Assets.clickSound);
          }

          public void hit() {
                   Assets.playSound(Assets.hitSound);
          }
        };
        lastLevel = 1;
        world = new World(worldListener, lastLevel);
        renderer = new WorldRenderer(glGraphics, batcher, world);
        pauseBounds = new Rectangle(320- 64, 480- 64, 64, 64);
        resumeBounds = new Rectangle(160 - 96, 240, 192, 36);
        quitBounds = new Rectangle(160 - 96, 240 - 36, 192, 36);
        lastScore = 0;
        scoreString = "score: 0";
        remainTargets = world.BASE_TARGETS;
        targetString = "Target Remain: " + remainTargets;
        targetScoreString = "Target Score: " + world.getLevelPassScore();
        fpsCounter = new FPSCounter();
    }

    /*
     * It  make sure any user input is processed correctly and will also
     * update the World instance
     * if necessary.
     */
          @Override
          public void update(float deltaTime) {
          if(deltaTime > 0.1f)
              deltaTime = 0.1f;

          switch(state) {
          case GAME_READY:
              updateReady();
              break;
          case GAME_RUNNING:
              updateRunning(deltaTime);
              break;
          case GAME_PAUSED:
              updatePaused();
              break;
          case GAME_LEVEL_END:
              updateLevelEnd();
              break;
          case GAME_OVER:
              updateGameOver();
              break;
          }
```

```
}

/*
 * The updateReady() method is invoked in the paused subscreen.
 * All it does is wait for a
 * touch event, in which case it will change the state of the game
 * screen to the GAME_RUNNING state.
 */
private void updateReady() {
    if(game.getInput().getTouchEvents().size() > 0) {
        state = GAME_RUNNING;
    }
}

/*
 * In the updateRunning()method, we first check whether the user
 * touched the pause button in the upper-right corner. If that's the
 * case, then the game is put into the GAME_PAUSEDstate.
 * Otherwise, we update the World instance with the current delta
 * time and the x and y axis value of the accelerometer, which are
 *responsible for moving ball.
 */
private void updateRunning(float deltaTime) {
    List<TouchEvent> touchEvents =
         game.getInput().getTouchEvents();
    int len = touchEvents.size();
    for(int i = 0; i < len; i++) {
        TouchEvent event = touchEvents.get(i);
        if(event.type != TouchEvent.TOUCH_UP)
            continue;

        touchPoint.set(event.x, event.y);
        guiCam.touchToWorld(touchPoint);

        if(OverlapTester.pointInRectangle(pauseBounds, touchPoint))
       {
            Assets.playSound(Assets.clickSound);
            state = GAME_PAUSED;
            return;
        }

        //update the angle of the cannon
        world.cannon.angle =
            touchPoint.sub(world.cannon.position).angle();
    /*
     * The initial velocity of the cannonball will be a combination
     * of the cannon’s direction and the speed the cannonball has
     * from the start. The speed is equal to the distance between
     * the cannon and the touch point. The further away you
     * touch, the faster the cannonball will fly.
     */
    if(event.type == TouchEvent.TOUCH_UP) {
        float radians = world.cannon.angle * Vector2.TO_RADIANS;
        float ballSpeed = touchPoint.len() / 28;
        world.ball.position.set(world.cannon.position);
        float velocityX = FloatMath.cos(radians) * ballSpeed;
        float velocityY = FloatMath.sin(radians) * ballSpeed;
        world.ball.setVelocity(velocityX, velocityY);
        world.ball.bounds.lowerLeft.set(world.ball.position.x - 0.1f,
            world.ball.position.y - 0.1f);
    }

}
//update the World instance
world.update(deltaTime);
//update the number of remained targets
remainBobs = world.bobs.size();
remainSquirrels = world.squirrels.size();
remainTargets = remainBobs + remainSquirrels;
targetString = "Remain: " + remainTargets;

targetScoreString = "Target: " + world.getLevelPassScore();
//check whether our score string needs updating
if(world.score != lastScore) {
    lastScore = world.score;
    scoreString = "score: " + lastScore;
}
//Store the current level
if(world.level != lastLevel) lastLevel = world.level;
//check whether time is up, in which case we enter the
//GAME_NEXT_LEVELstate,
//which will show the message in the top left screen
if(world.state == World.WORLD_STATE_NEXT_LEVEL) {
    state = GAME_LEVEL_END;
}
/*
```

```
     * In case the game is over, we set the score string to either
     * score: #score or new highscore: #score, depending on
     * whether the score achieved is a new high score. We then add
     * the score to the Settings and tell it to save all the settings to
     * the SD card. Additionally, we set the game screen to the
     * GAME_OVERstate.
     */
    if(world.state == World.WORLD_STATE_GAME_OVER) {
        state = GAME_OVER;
        if(lastScore >= Settings.highscores[4])
            scoreString = "new highscore: " + lastScore;
        else
            scoreString = "score: " + lastScore;
        Settings.addScore(lastScore);
        Settings.save(game.getFileIO());
    }
}

/*
 * In the updatePaused()method, we check whether the user has
 * touched the RESUME or QUIT UI elements and react accordingly.
 */
private void updatePaused() {
    List<TouchEvent> touchEvents =
         game.getInput().getTouchEvents();
    int len = touchEvents.size();
    for(int i = 0; i < len; i++) {
        TouchEvent event = touchEvents.get(i);
        if(event.type != TouchEvent.TOUCH_UP)
            continue;

        touchPoint.set(event.x, event.y);
        guiCam.touchToWorld(touchPoint);

        if(OverlapTester.pointInRectangle(resumeBounds, touchPoint))
        {
            Assets.playSound(Assets.clickSound);
            state = GAME_RUNNING;
            return;
        }

        if(OverlapTester.pointInRectangle(quitBounds, touchPoint)) {
            Assets.playSound(Assets.clickSound);
            game.setScreen(new MainMenuScreen(game));
            return;

        }
    }
}

/*
 * In the updateLevelEnd() method, we check for a touch-up event;
 * if there has been one, we create a new World and
 * WorldRendererinstance. We also tell the Worldto use the score
 * achieved so far and set the game screen to the
 * GAME_READYstate, which will again wait for a touch event.
 */
private void updateLevelEnd() {
    List<TouchEvent> touchEvents =
          game.getInput().getTouchEvents();
    int len = touchEvents.size();
    for(int i = 0; i < len; i++) {
        TouchEvent event = touchEvents.get(i);
        if(event.type != TouchEvent.TOUCH_UP)
            continue;
        world = new World(worldListener, lastLevel);
        renderer = new WorldRenderer(glGraphics, batcher, world);
        world.score = lastScore;
        world.level = lastLevel;
        state = GAME_READY;
    }
}

/*
 * In the updateGameOver()method, we again just check for a
 * touch event, in which case we simply transition back to the main
 * menu.
 */
private void updateGameOver() {
    List<TouchEvent> touchEvents =
          game.getInput().getTouchEvents();
    int len = touchEvents.size();
    for(int i = 0; i < len; i++) {
        TouchEvent event = touchEvents.get(i);
        if(event.type != TouchEvent.TOUCH_UP)
            continue;
        game.setScreen(new MainMenuScreen(game));
    }
}

/*
 * Rendering of the game screen is done in two steps. We first
 * render the actual game world via the WorldRendererclass, and
 * then render all the UI elements on top of the game world
 * depending on the current state of the game screen. The
 * render()method does just that.
 */
@Override
public void present(float deltaTime) {
    GL10 gl = glGraphics.getGL();
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
    gl.glEnable(GL10.GL_TEXTURE_2D);

    renderer.render();

    guiCam.setViewportAndMatrices();
    gl.glEnable(GL10.GL_BLEND);
    gl.glBlendFunc(GL10.GL_SRC_ALPHA,
          GL10.GL_ONE_MINUS_SRC_ALPHA);

    batcher.beginBatch(Assets.items);
    switch(state) {
    case GAME_READY:
        presentReady();
        break;
    case GAME_RUNNING:
        presentRunning();
        break;
    case GAME_PAUSED:
        presentPaused();
        break;
    case GAME_LEVEL_END:
        presentLevelEnd();
        break;
    case GAME_OVER:
        presentGameOver();
        break;
    }
    batcher.endBatch();
    gl.glDisable(GL10.GL_BLEND);
    fpsCounter.logFrame();
}

/*
 * The presentReady()method just displays the pause button in the
 * top-right corner, as well as the score string in the top-left corner.
 */
private void presentReady() {
    batcher.drawSprite(160, 240, 192, 32, Assets.ready);
}

/*
 * In the presentRunning() method, we simply render the pause
 * button and the current score string.
 */
private void presentRunning() {
    batcher.drawSprite(320 - 32, 480 - 32, 64, 64, Assets.pause);
    Assets.font.drawText(batcher, scoreString, 16, 480-20);
    Assets.font.drawText(batcher, targetScoreString, 16, 480-40);
    Assets.font.drawText(batcher, targetString, 16, 480-60);
}

/*
 * The presentPaused() method displays the pause menu UI
 * elements and the score again.
 */
private void presentPaused() {
    batcher.drawSprite(160, 240, 192, 96, Assets.pauseMenu);
    Assets.font.drawText(batcher, scoreString, 16, 480-20);
    Assets.font.drawText(batcher, targetString, 16, 480-40);
}
/*
 * The presentLevelEnd()method renders the string THE PRINCESS
 * IS … at the top of the
 * screen and the string IN ANOTHER CASTLE! at the bottom of the
 * screen.
 */
private void presentLevelEnd() {
    String topText = "now is ...";
    String bottomText = "in the level " + world.level + "!";
    float topWidth = Assets.font.glyphWidth * topText.length();
```

```
            float bottomWidth = Assets.font.glyphWidth *
                bottomText.length();
            Assets.font.drawText(batcher, topText, 160 - topWidth / 2, 480 -
                40);
            Assets.font.drawText(batcher, bottomText, 160 - bottomWidth /
                2, 40);
        }

        /*
         * The presentGameOver()method displays the game-over UI
         * element as well the score
         * string. Remember that the score screen is set in the
         * updateRunning()method to either
         * score: #score or new highscore: #value.
         */
        private void presentGameOver() {
            batcher.drawSprite(160, 240, 160, 96, Assets.gameOver);
            float scoreWidth = Assets.font.glyphWidth * scoreString.length();
            Assets.font.drawText(batcher, scoreString, 160 - scoreWidth / 2,
                480-20);
        }
    @Override
    public void pause() {
        if(state == GAME_RUNNING)
          state = GAME_PAUSED;
    }
}
```