System Design Document for Galactica: Space Force of Justice

Ludvig Andersson Anthony Kalcic Rasmus Lindgren
Markus Pettersson

May 28, 2017
Version 2.0

This version overrides all previous version

Contents

1 Introduction

Galactica: Space Force of Justice is an endless runner game that is able to run on desktop, Android and Google Cardboard.

Since the application runs in both VR and on a normal screen the program supports both stereo- and monoscopic rendering and has the ability to track head orientation in VR mode.

Since the options for user input on a Google Cardboard is limited to head rotation and a single button the application also requires a wired or wireless game pad. The Android game version has a virtual touch pad.

1.1 Definitions, acronyms and abbreviations

All terms regarding the actual gameplay are the same as in the rapid application development document (RAD).

- LibGDX - The framework used for handling graphics, physics, data storage and user input.
- Google Cardboard - Head mounted display that uses a smart phone to run VR applications.
- Passive MVC - A version of the MVC architectural pattern were the controller tells the model and view when to update.
- Normal screen mode - A non vr gameplay mode on the android device that has the same view as the desktop mode but with a joystick and an extra on screen button.
- Tick - One update cycle of the game.
- Rendering - Plotting of virtual elements from the perspective of the camera.

1.2 Design goals

This document describes the design of the application *Galactica: Space Force of Justice* that has been defined in the requirements and analysis document.

- The design must follow the passive MVC architectural pattern.
- The design must be loosely coupled making it possible to switch out all external frameworks.
- The application's core model must be completely free from external libraries.

- The program must be open for extension, making it easy to add new components to the game.
- For usability see RAD.

2 System architecture

The application is written in Java and uses the passive MVC design pattern.

2.1 Platforms

The application will run on a desktop computer or an Android device. On Android the game can be played either in normal screen mode or in a Google Cardboard VR mode.

2.2 LibGDX

Galactica uses the LibGDX framework. LibGDX was chosen because it is written in Java, has widespread usage, and an extension for developing Google Cardboard applications. The framework will handle the tick speed for the game logic and graphical rendering. Data storage and collision physics is handled by LibGDX as well [1].

2.3 Cardboard extension

The project uses the Google Cardboard API for the virtual reality part of the application. In order to combine this API with LibGDX the *Libgdx-cardboard-extensions* by Wei Yang was used. The extension provides functionality for handling stereoscopic rendering and head rotation tracking [2].

2.4 Vecmath

The application uses vectors to represent position and quaternions to represent rotation. The Vecmath library was used for this and since it is part of javax it can essentially be treated as a part of the Java API. Since it is such a basic tool it has been excepted from the design goal of keeping the core package free from external libraries.

2.5 Lombok

Making the project closed for modification and open for extension (open closed principle) created more private variables than desired. This happened because classes

needed to be hidden and unchangeable to ensure that they could stay closed for modification.
Working with private variables creates many getters and setters methods which in turn creates boilerplate code. This is why project Lombok is used. Project Lombok has a plug-in for development environments, that generates getters and setters simply by writing "@Getter" and "@Setter" in front of the variable. This saves a lot of space and makes the code more readable. See figure 1 in Appendix for an example.

3 Subsystem decomposition

The applications code structure involves a package for model, view, controller and utilities. Two main classes can also be found. For an overview of dependencies, see figure 2 in Appendix.

3.1 Model

The *model* is responsible for storing and processing data. The package contains a core package, which handles the logic of the actual game, as well as other packages for menus, game pad data, helper classes et cetera. A dependency overview using STAN and an UML diagram can be found as figures 3 and 4 in Appendix.

3.2 View

The *view* package handles the visual representation of the model, as per definition of the mvc pattern. The view package contains three packages: *parallaxview*, *renderer*, and *sound*.

The *renderer* package handles the rendering of 3D models and particles using the LibGDX framework. The renderer package has no references to any of the model classes.

The main task of the *parallaxview* package is to choose which of the models that will be visually represented to the user, and then tell the *renderer* to render it. The view can be run asynchronous to the model if desired.

The main task of sound is to play audio. The package uses the LibGDX framework to play sounds.

See Appendix, figure 5, for more information about dependency overview and 6 for UML diagram.

3.3 Controller

The *controller* package has responsibility over user input, updating the model and the view. It handles the user input and changes the model depending on the input. The controller also decides which view should be shown to the user and which models that are being updated.

See Appendix, figure 7, for the STAN dependency overview and figure 8 for the UML diagram.

3.4 Utilities

The *utilities* package contains helper classes for the other packages. It is responsible for loading resources and contains static math functions that are used across the package structure.

3.5 Main classes

The program has two main classes, one for Cardboard devices and one for launching normal applications. The classes are both responsible for initializing the controllers necessary to run the game. After that, they proceed to update the controllers when the framework (LibGDX) tells them to do so.

3.6 Sequence diagrams

Two sequence diagrams depicting the cannon shooting and the application starting on Android is included in the document. Pictures of these diagrams can be found as figures 9, 10 and 11 in the Appendix.

Figures 9 and 10 represent the normal flow of the use case "Cannon" described in the RAD. The diagrams show class interaction, and methods that are involved. 9 shows the classes needed to generate a cannon shot, whilst 10 shows the updating of a corresponding projectile. Figure 10 shows the three updating calls (origin of these calls can be found in the LibGDX engine clock) made for moving the cannon projectile previously generated. After the time representing a projectiles life cycle has been exceeded and the "death" process has started.

Figure 11 represents the normal flow for the sequence diagram ”App start”, which can also be found in the RAD. The diagram shows Java related methods that the application uses to generate an Android screen. Firstly the initial screen is generated, showing buttons for the player to choose from. In the normal flow, the player presses the button labeled "normal" and the application switches and shows a new screen.

3.7 Code quality

A variety of different code analyzing tools have been used to confirm that the code is written according to the standard Java conventions. The tools that were used are: PMD, FindBugs and Check Style. These tools all have different built in analyzing options that a user can pick and choose from. A majority of the options have been enabled but those that were excessive or made plug-ins give errors were disabled. A list of removed tests can be found in the documents tab in the Git repository. The result after working towards solving the warnings can be seen as figure 12 in the Appendix.

3.8 Testing

All the tests can be found in the test package. The testing of the application is limited to the Model package. This is because the Model package handles most of the logic in the game and therefore is the most important package to test. The JaCoCo tool is used for checking how much of the code that is covered by tests. This also helps when testing private methods since they can not be tested in the same manner as a public method. The amount of code covered is shown in figure 13, in the Appendix.

4 Persistent data management

The file system used for the application is the same for all platforms the game is designed for. The file system is handled by the game framework LibGDX.

4.1 Filetypes

The application uses five different file types:

G3db files are used for the 3D models in the game. Every 3D model that can be seen on the screen is loaded from a .g3db file. G3db files are the primary file-type used by LibGDX.

Mp3 files are used to play audio from the game, both sound effects and background music. LibGDX uses .mp3, .ogg and .wav files. Galactica uses the mp3 file type because of its wide spread usage.

Png files are used for textures, both for 3D models and menu backgrounds. Galactica uses .png because of its wide spread use. Compared to .jpg, it is losslessly compressing meaning that it does not lose information when compressed.

P files are used for particle effects. It stores the data about how the particle effect should be emitted. The file type used is not required and could be a .txt file as the file only contains text. The directions given by LibGDX were not clear as they used

the file types .pfx and .p interchangeably. The .p file type was chosen because .pfx interfered with the "Personal Information Exchange" file type in windows.

Wavefront OBJ files are used to create collision models in the game by reading the vertices coordinates.

4.2 Naming

3D model files are named according to what they are, for example if a spacecraft is named Agelion then the 3D model is named agelion.g3db.

3D model textures used for the 3d models are named texture.png and are placed in the same folder as the 3D model that is using the texture.

Hitboxes for 3D models are placed in the same folder as their respective 3D model and are all named hitbox.obj.

Audio files are named according to what their audio is, for example an explosion sound is named ”explosion.mp3”. The different songs used as background music are named after their original titles.

Texture files for on screen buttons or backgrounds are named after their usage in the application.

Packages follow a standardized name convention set by Oracle[3], using only lowercase ASCII letters.

4.3 Filestructure and storage

The resources that are used in Galactica are stored in the same assets folder within the game folder. The file structure for resources in Galactica is mainly divided into four sections.

3D models contain sub-folders that correspond to every 3D model in the game, containing a 3D model and optionally a texture file for the specific 3D model.

Sounds contains two sub-folders; sound effects and music. “sounds effects” is for shorter audio tracks that are loaded into the primary memory, while “music” contains larger files that are streamed into the game.

Images contains sub-folders that contain images that are used in Galactica. Subfolders have a name that describes what kind of images they contain.

Particles contains all the particle files and an image that is used by all particles.

4.3.1 Local data storage

The player score is the only data stored to permanent memory. The application handles score linked with a player's name. After a finished game, the data is written to storage.

File commonly found here: "This PC > Windows (C:) > Users > [Name] > .prefs"

5 Access control and security

The application doesn't store any critical usage data such as passwords or log-in information and there is no use for an Admin in the application due to it only being played locally, thus this is not applicable.

6 Appendix

```
@Getter
private final ExitButton3D exitButton;
@Getter
private final StartButton3D startButton;
@Getter
private final List<IPowerUp> powerUps;

@Getter
@Setter
private Vector3f aimDirection;
```

Figure 1: A common use of the Project Lombok extension

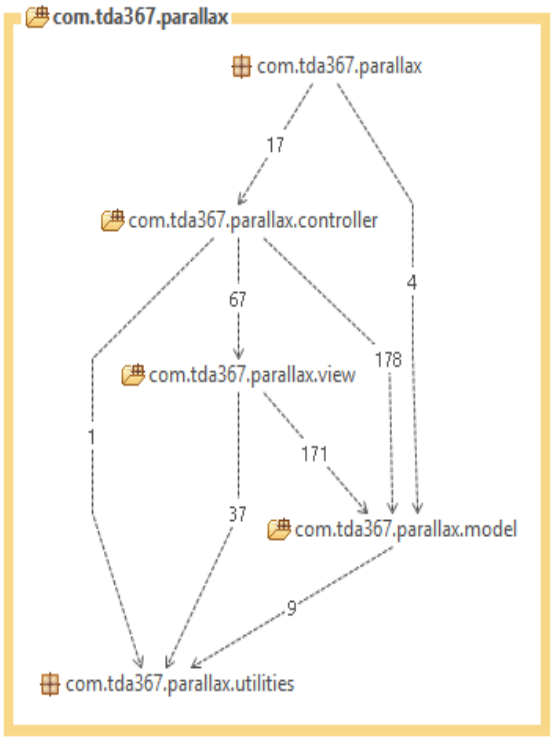

Figure 2: Dependency analysis (by STAN) for the top level packages of the application

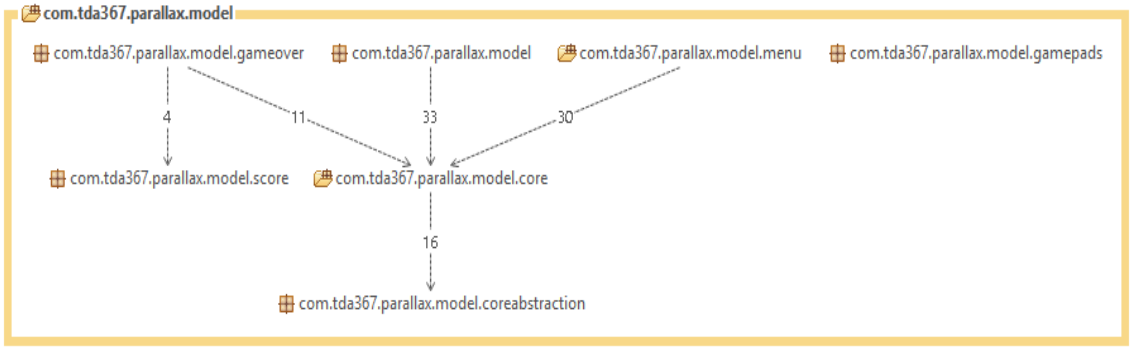

Figure 3: Dependency analysis (by STAN) for model packages of the application

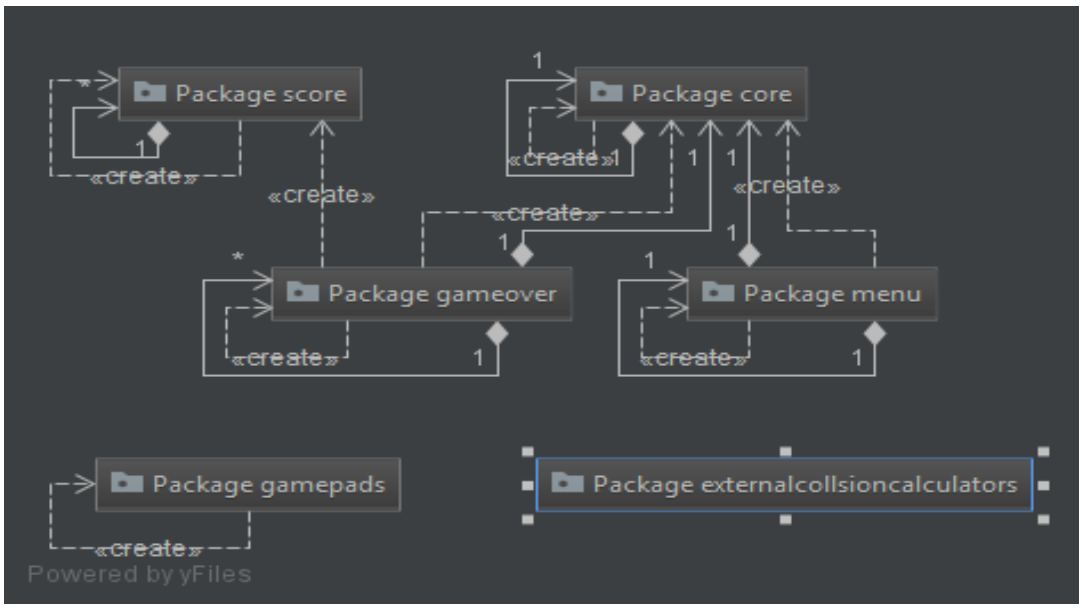

Figure 4: UML overview of the model package. For a enlarged image, visit `https://drive.google.com/open?id=0B_iEOvynwJHOV09KTi1qaC1ZeHc`

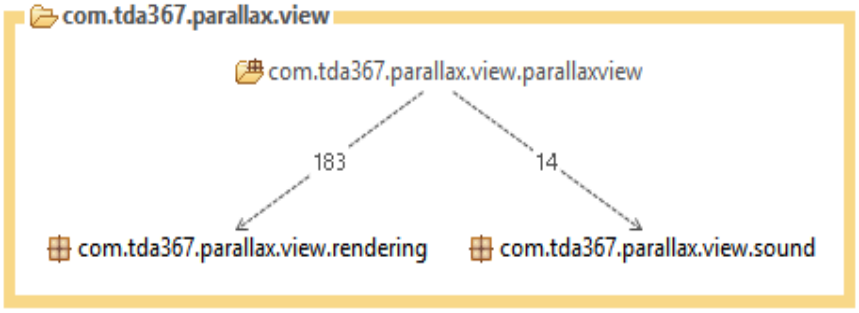

Figure 5: Dependency analysis (by STAN) for view packages of the application

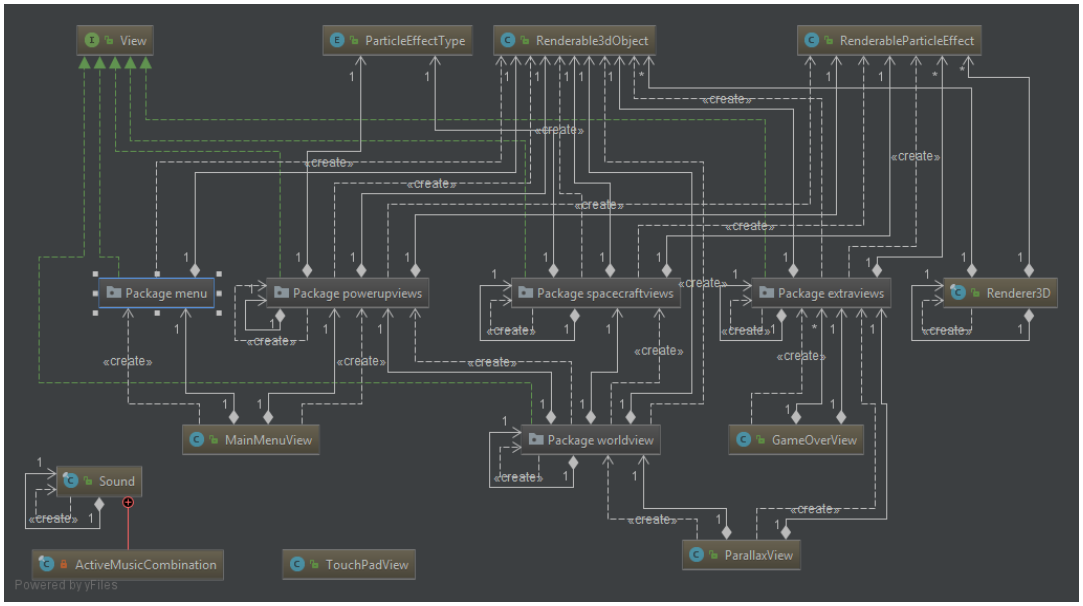

Figure 6: UML overview of the model package. For a enlarged image, visit `https://drive.google.com/open?id=0B_iEOvynwJHOV09KTi1qaC1ZeHc`

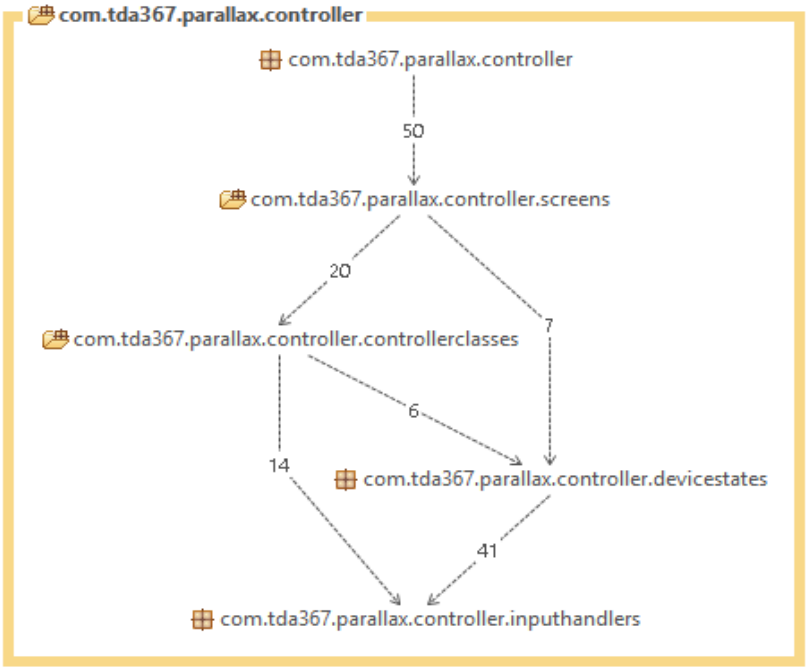

Figure 7: Dependency analysis (by STAN) for controller packages of the application

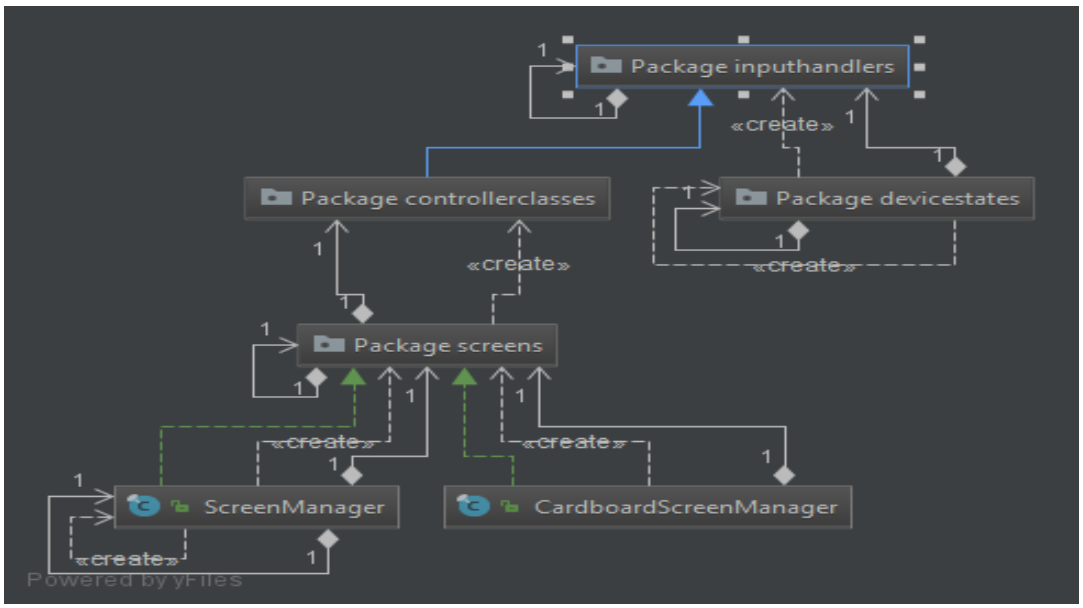

Figure 8: UML overview of the controller package. For a enlarged image, visit `https://drive.google.com/open?id=0B_iEOvynwJHOV09KTi1qaC1ZeHc`

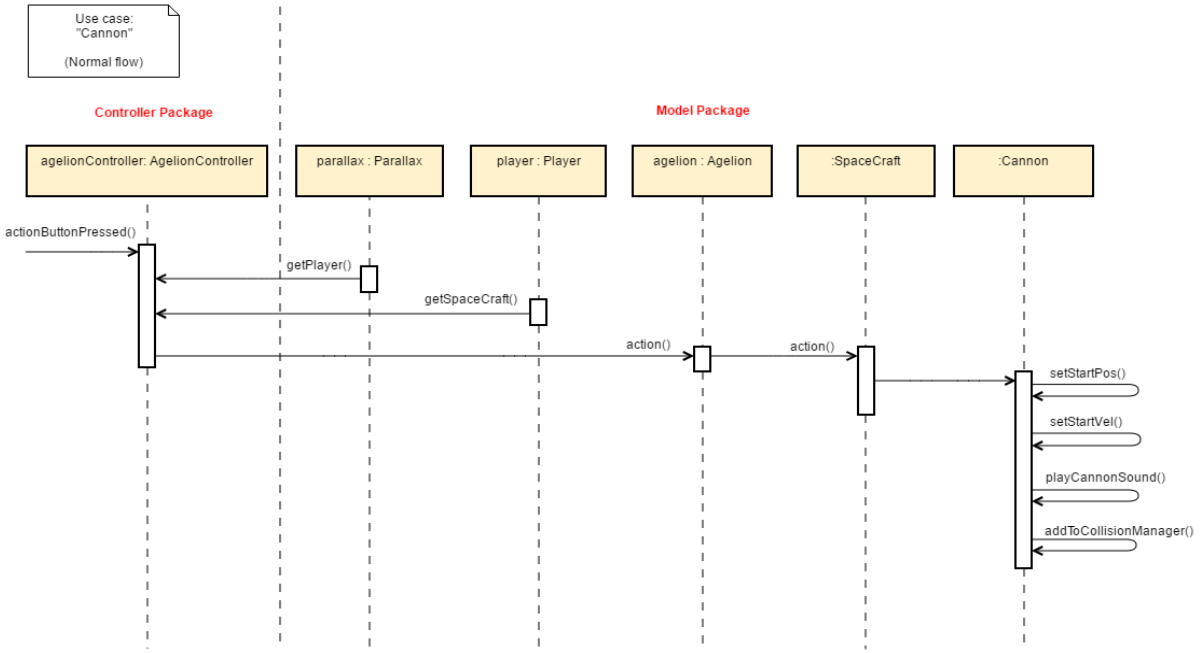

Figure 9: Sequence diagram for the producing of a cannon projectile, based on the ”Cannon” use case

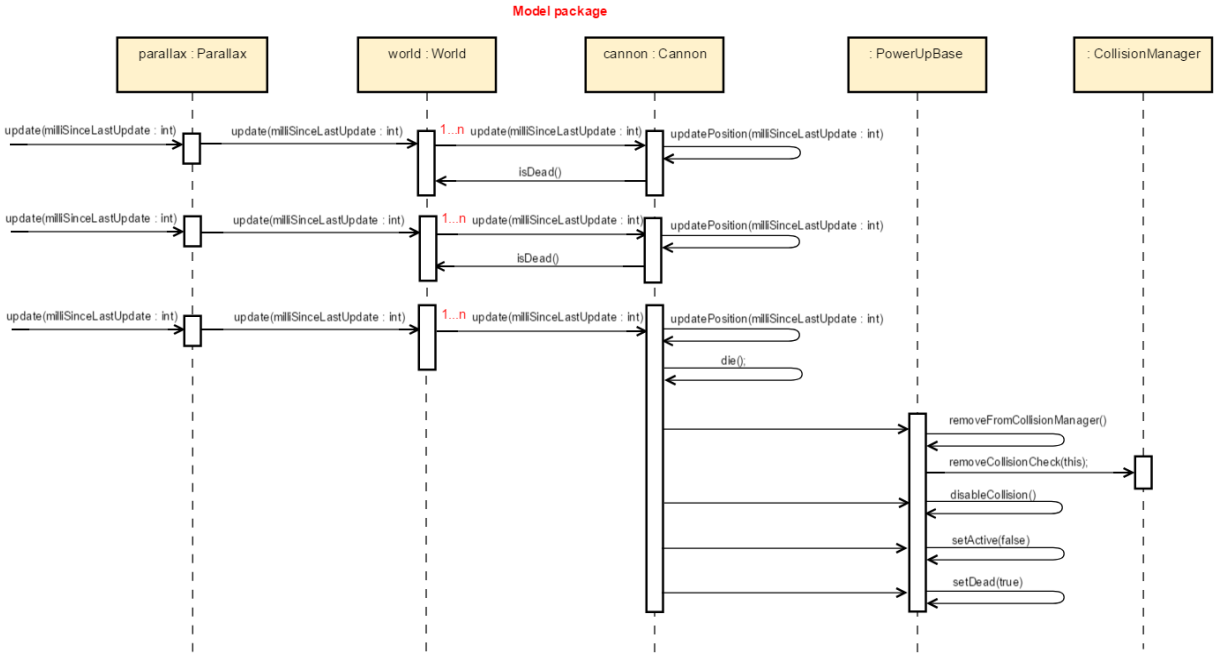

Figure 10: Sequence diagram for the updating of the cannon projectile

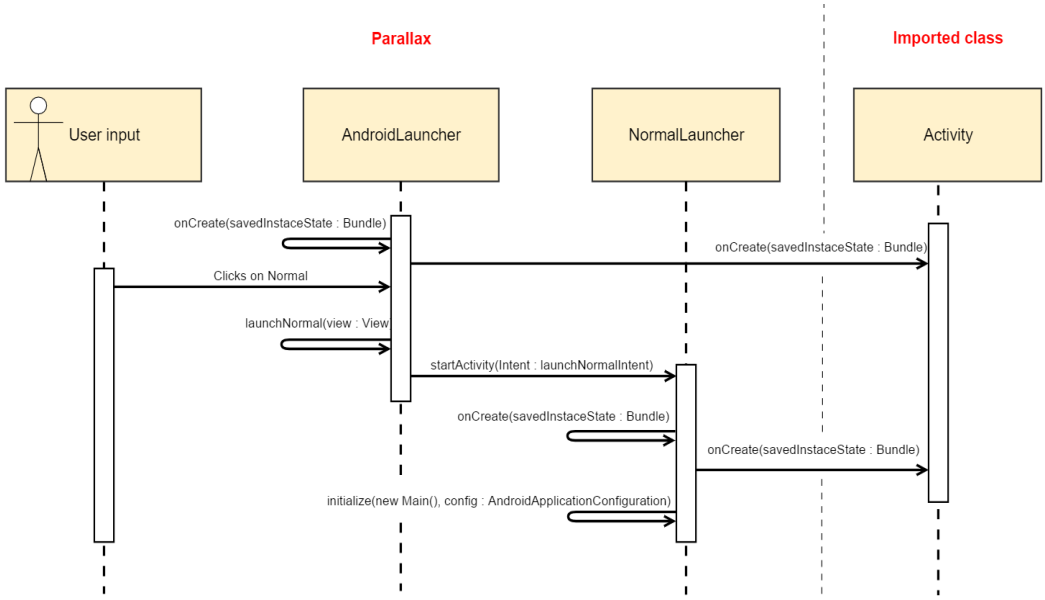

Figure 11: Sequence diagram for ”App Start” use case

Project:
parallax
Scope:
Directory '.../core/src - [core]'
Profile:
Support3
Results:
Enabled coding rules: 814

- FindBugs: 449
- PMD: 252
- Checkstyle: 113

Problems found: 109

- FindBugs: 11
- PMD: 74
- Checkstyle: 24

Time statistics:

- Started: Sun May 28 23:43:19 CEST 2017
  - Initialization: 00:00:00.015
  - Checkstyle: 00:00:02.391
  - FindBugs: 00:00:09.713
  - PMD: 00:00:02.083
  - Gathering results: 00:00:00.281
- Finished: Sun May 28 23:43:33 CEST 2017

Figure 12: Amount of remaining problems found by the code analysis tools: PMD, Check Style & FindBugs. For a enlarged image, visit `https://drive.google.com/open?id=0B_iEOvynwJHOV09KTi1qaC1ZeHc`

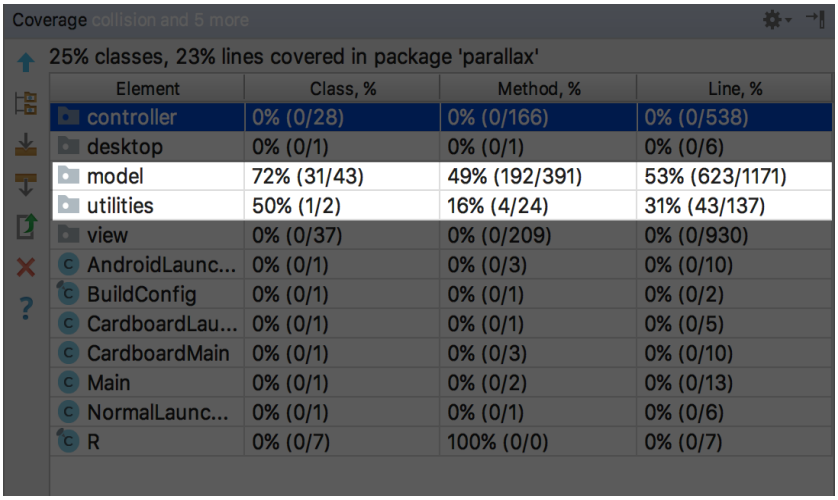

Figure 13: The amount of test coverage according to JaCoCo

References

[1] (2013). Libgdx, Badlogicgames, [Online]. Available: https://libgdx.badlogicgames.com/features.html.

[2] W. Yang. (2016). Libgdx-cardboard-extension, [Online]. Available: https://github.com/yangweigbh/Libgdx-CardBoard-Extension.

[3] I. Sun Microsystems. (1999). 9 - naming conventions, [Online]. Available: http://www.oracle.com/technetwork/java/codeconventions-135099.html.