**REVISION HISTORY**

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 22.12.2024 | 1.0 | Added Static Code Analysis | Doruk Esen  Ömer Emre Bozkurt |
| 22.12.2024 | 1.1 | Added Dependency Analysis and Test Coverage Analysis | Doğa Yağmur Uğut  Eda Nur Yılmaz  Poyraz Köroğlu |

**TABLE OF CONTENTS**

**Revision History 1**

**1** **Introduction 3**

***1.1*** ***Document overview 3***

**2** **Static Code Analysis 4**

***2.1*** ***Tools 4***

***2.2*** ***Results and Discussion 4***

**3** **Dependency Analysis 6**

***3.1*** ***Tools 6***

***3.2*** ***Results and Discussion 6***

**4** **Test Coverage Analysis 7**

***4.1*** ***Tools 7***

***4.2*** ***Results and Discussion 7***

# **Introduction**

## ***Document overview***

This document presents and interprets the code analysis results regarding the SB software development project. Code is analyzed by 3 different tools: 1) Static Code Analysis tool 2) Dependency Analysis tool and 3) Test Coverage tool. The first tool is used to reveal potential bugs that might be overseen during the testing process. The second tool is employed for evaluating the design quality based on the amount of coupling among the software modules and to what extent the code reflects the originally envisioned design. The last tool is used for measuring the coverage of unit tests in the project. Each section below is dedicated to each of these 3 analyses.

# **Static Code Analysis**

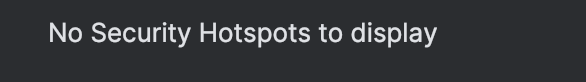
## ***Tools***

We used SonarQube to analyze our project. SonarQube is one of the easiest and most efficient analysis tools. We set up the analysis tool from the Plug-in section inside the IntelliJ IDEA. After we ran the analysis SonarQube gave us results for several classes.

* SonarQube <https://plugins.jetbrains.com/plugin/7973-sonarqube-for-ide>

## ***Results and Discussion***





Analysis of alerts that are reported by the tool:

| **Class** | **Count of instances** | **Severity** | **Count of real bugs** |
| --- | --- | --- | --- |
| Alien.java | 26 | Low | 0 |
| Core.java | 16 | Moderate | 0 |
| Easy\_Alien.java | 6 | Low | 0 |
| EndPage.java | 8 | Low | 0 |
| ExtraHard\_Alien.java | 11 | Low | 0 |
| Game.java | 7 | Moderate | 0 |
| Hard\_Alien.java | 10 | Low | 0 |
| Lwjgl3Launcher.java | 2 | Moderate | 0 |
| Main.java | 2 | Low | 0 |
| Medium\_Alien.java | 9 | Low | 0 |
| Platform.java | 6 | Low | 0 |
| PlayerControlsManager.java | 7 | Low | 0 |
| Power\_Up.java | 10 | Low | 0 |
| Projectile.java | 15 | Low | 0 |
| StartPage.java | 9 | Low | 0 |
| StartupHelper.java | 9 | Low | 0 |
| **Total** | 153 |  | 0 |

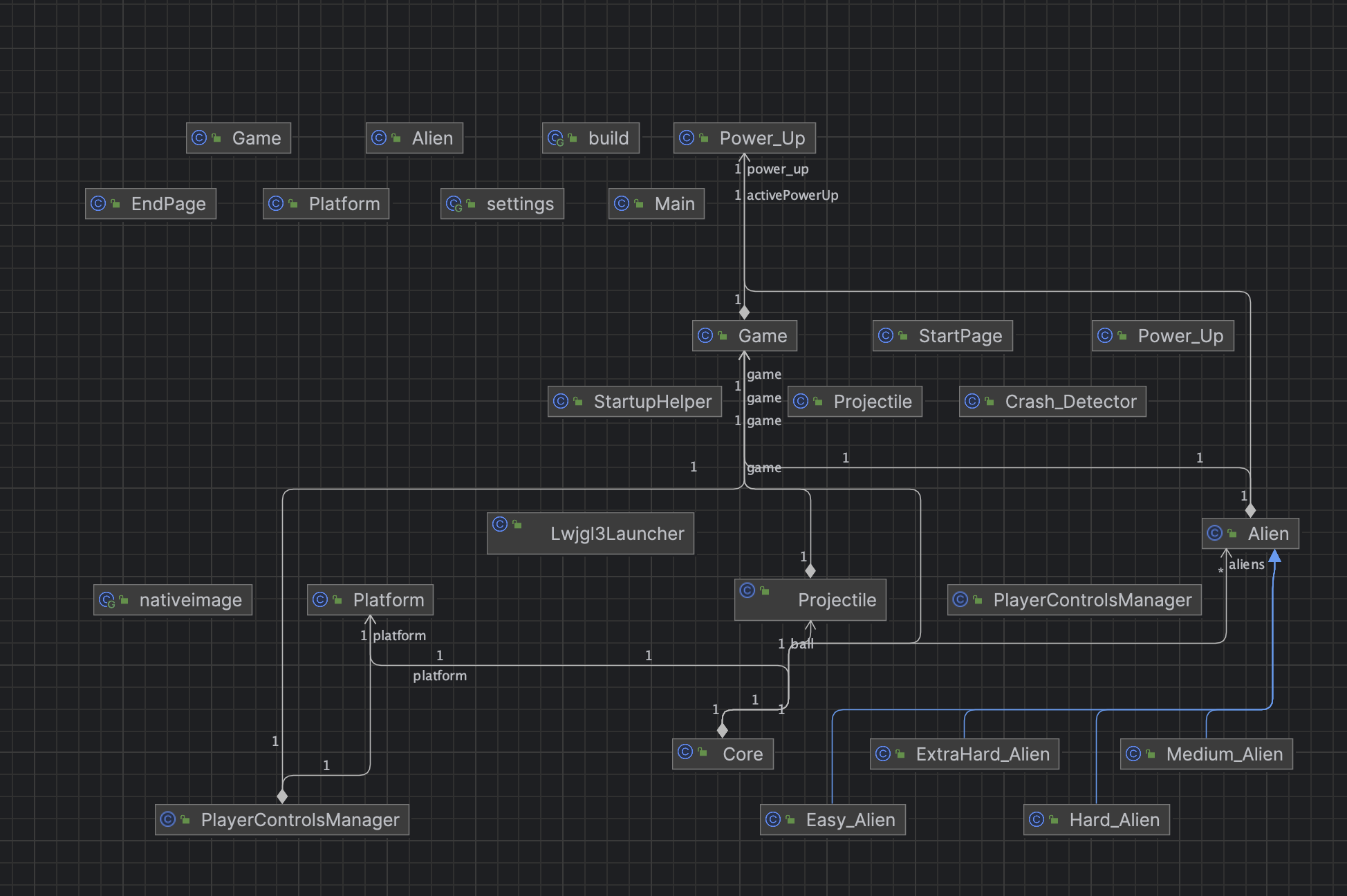
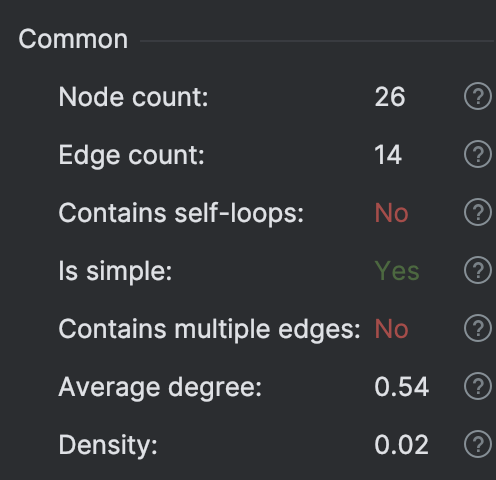
Also there are no security concerns regarding our project and implementation.

# **Dependency Analysis**

## ***Tools***

We used a Java dependency viewer from IntelliJ IDEA Ultimate. It did not require additional setups or downloads and gave accurate results.

## ***Results and Discussion***



Number of edges: 14

Number of nodes: 26

Edge-to-node Ratio = 14 / 26 = 0.538

Tree Impurity = 2 \* (14 – 26 + 1) / (26 – 1) \* (26 – 2) = -0.0366

As we can see, tree impurity is tiny. That means that our design has a very good structure. The edge-to-node ratio is also not big. Alien and Projectile classes are more coupled nodes than the other ones. We also have some nodes that are not coupled by any nodes. The coupling can be handled by refactoring and decomposing to reduce density.,

**4 Test Coverage Analysis**

***4.1*** ***Tools***

As a Test Coverage Analysis tool, we used Intellij IDEA’s built-in coverage analysis software.

(<https://www.jetbrains.com/help/idea/code-coverage.html#report>)

***4***.***2*** ***Results and Discussion***

We have covered each class properly. However, some of our lines are not fully covered. We’ve dealt with Lwjgl3Launcher class with perfection. StartupHelper is a class developed by developers of LibGDX. Since that class has direct access to source code we did not want to make any adjustments in that class.

