Code Coverage Analysis

# Instructions

1. Enter your team letter and name: **2C**
2. Create screenshots of each tier-level page of the JaCoCo coverage report. Crop these screenshots as necessary so that the pertinent information is readable.
3. In your analysis, provide suggestions for improvements in the unit tests for each tier which are more insightful than the obvious "Write more tests" suggestion. There will be significant gaps in coverage due to the fact that you might not have unit tests for all of the project’s components. That’s expected.
4. In the last section of this file, identify two specific components, i.e. classes, that were unit tested. You will identify one with good code coverage and one with poor coverage.
5. Upload the final Word file to the *Code Coverage* dropbox in the **Team** **Exercises** category.

# UI Tier

This is our analysis at the UI Tier code for the project.

## 

## Analysis

Many of the classes in this tier are tested, but they are not necessarily tested well. This is evidenced by the fact that only 60% of the methods are tested and only 55% of the lines are covered. This is partially due to the fact that at the time this report is being written, individuals are still working on writing test classes for some of the components in this package. For the classes test classes that are not currently in progress, the tests tend to have good coverage of the possible scenarios that the code can be called in.

# Application Tier

This is our analysis at the UI Tier code for the project.

## 

## Analysis

This package contains 4 classes, GameCenter, Player, PlayerLobby, and PlayerLobbyExceptions. From these 4 classes, about 64% of the lines are being tested and accounted for. From there, an average of 61% of the methods are being tested. Within the classes themselves, Player and PlayerLobbyExceptions are the best tested. These two files are considered the “friendly files” of the package, as they are going to be actively used throughout the development of the application. To improve, more tests can be written to improve upon the percentages.

# Model Tier

This is our analysis at the UI Tier code for the project.

## 

## Analysis

## The package only has a single class, Board, in it and as such only this class is being tested. The tests for this class do cover the class well as shown by the 100% coverage of both the methods and the lines. The coverage of this class is further expanded on in the “Well-tested Component” section of this document.

# Well-tested Component

This is our analysis of a well-tested component: Board.java



## Analysis

The reason as to why the board class is a well tested class is because of how often it is used throughout the application. This class has 100% method and line coverage. We wanted Board.java to be a friendly class. Each written test accomplishes its necessary task to find any impurities in the program. Testing of this class also led to functionality implementations in the class. One such example was to add a setter for ActivePlayer which details who gets to start the next move.

# Poorly-tested Component

This is our analysis of a poorly-tested component: getGameRoute.java



## Analysis

One thing that we can do to improve upon this test coverage is to consider more cases. Right now, even though our coverage shows 100% on class and method, it does not consider enough cases. This is reflected in the 53% line percentage. It does not test the validity and reliability of when a player attempts to challenge another player currently in a game. It also does not test whether the players enter correctly, not whether the flipped board is correct.