Multiplayer GUI Yahtzee

Test and Validation Plans

Blue Bulldogs

Hailey Boe, Anna Cardinal, Tyler Kamphouse, Jonathan Smoley

Course: CPSC 224 - Software Development

Instructor: Aaron S. Crandall

I. Introduction

I.1. Project Overview

As previously mentioned, the subject of this project is Yahtzee. More specifically, the goal is to build a multiplayer version of Yahtzee that includes a GUI with interactions scalable to the number of players in the group. To ensure that this program will work properly, several different testing methods will be deployed. Due to the GUI serving as the sole way for user(s) to see the game progressing, one of the major functions in need of testing is the navigation between windows. Because the number of windows changes will depend on how many players are involved, an equivalently important function to test is the ability to pick how many players can play. This function connects a broader requirement that the game settings be customizable. Once the configurable aspect of this Yahtzee implementation is tested, the components that depend on user settings can be tested. These include both the size and roll count provided to the hand as well as the size of each player's scorecard. While it is important that the functions already listed work, their working is of little importance if the players cannot see their progress through the game. Therefore, a final function that needs to be tested is whether individual components are in view throughout the course of the game.

I.2. Scope

The testing of this project will evaluate the functionality and completeness of each required behavior listed above. Below there is a collection of the specific methods that will be utilized in pursuit of the complete state of this program. Generally, the unit testing will cover individual methods whereas integration testing will cover these methods combined into a class. System testing will ensure that the necessary classes work together in reaching the overall required specifications. Functional and performance testing are needed to determine whether the specific functional and non-functional requirements provided in the project plan are met. Finally, User acceptance testing will mostly be performed to catch bugs and test the overall operation of Yahtzee.

II. Testing Strategy

1. Identify the requirements to be tested. All test cases shall be derived using the current Software Requirements Specification.
2. Identify which particular unit test(s) will be used to test each module. And what Integration tests will be used between modules
3. Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.
4. Identify the expected results for each unit test.
5. Document unit test case configuration, test data, and expected results.
6. Perform the unit test(s).
7. Successful unit testing is required before the unit is eligible for component integration/system testing.
8. Create Integration Tests using unit tests after confirming their success.
9. Identify the expected results for each integration test.
10. Document integration test case configuration, test data, and expected results.
11. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.

III. Test Plans

III.1. Unit Testing

The primary goal of unit testing is to take the smallest unit of testable software in the application, isolate it from the remainder of the code, and test it for bugs and unexpected behavior. If doing unit testing, who will write them, and how will you be reviewing progress with your unit testing? Will other team members be looking at the output of the test harness and when?

Unit testing will be utilized by our group. We plan on having at least two people write and modify existing unit tests from previous projects to fit with this project. These unit tests will be used to show our progress on issues on GitHub, and additionally be used to help for Integration tests that will show our progress on milestones. Every team member at some point should be viewing the unit tests as a way to track progress on what needs to be accomplished next and to hold each other accountable for getting functioning work.

Our group expects to have at least one unit test for every class, and classes such as “Hand” will have more to be able to test important functions like “roll hand” or “keep selected dice”. These types of unit tests will be taken from group member’s tests they have already written and lightly modified for this assignment. While unit tests for new classes like “Player” and “LeaderBoard” will need unit tests written from scratch. These will test abilities such as “get player score” or “set player name”.

III.2. Integration Testing

For integration testing, we are trying to strategically use integration testing to focus on catching bugs that haven't been detected by the unit tests. The areas we are going to focus on using integration testing the most is the junction between our settings frame and our game play frame, and between our main frame and the end frame. There are a lot of setting features, such as the number of players, in the settings page that are vital to the function requirements for the rest of the code. For that junction, we might test to see if the number of selected players in the startWindow changes the number of player’s names to be filled out in the nameWindow. We might also check that the MainWindow cycles through the appropriate number of players received in the startWindow when the user selects the number of players from drop down in the frame before. For the scorecard and main game play through junction, we will test to see that the scorecard has appropriately assembled itself in the endWindow, displaying the rankings of the players with the proper player scorecard data. We will be building more integration tests as we keep working.

When we work on code separately, every time we try to push back to the repo in gitHub, we will ensure that the tests are passing in JUnit. Every time we meet together in group meetings, we will also amply discuss bug issues and potential areas where there might be problems. In these meetings, we will also be able to talk through current bug problems. Overall, the most important thing we will do is add onto new code slowly and will test frequently with manually made data and tests. In these tests, the data sets we make to test our code will be inclusive of all test cases, such as testing for zeros, negative values, and other unordinary input data. After producing these data sets, we will then analyze the different structures of the two joint functions and determine the most vulnerable attributes. Afterwards, we will be able to better identify and remedy areas of our code that need to be strengthened.

III.3. System Testing

III.3.1. Functional testing:

Functional testing, in general, is highly dependent upon the various functional requirements included in the project plan. As such, the plan is to have a test for those requirements that are more likely to produce errors when faced with a user. For instance, while a title is required for this project, it has a low chance of failing to appear to the user. Therefore, testing resources (i.e. time) would be better spent on a requirement like tracking a user request for the number of players.

Once again, this type of testing determines if the actions performed when running this program are in accordance with the requirements set prior to development. With the combination of unit testing and integration testing, the only action-wise testing left to be completed pertains specifically to the functions required to make Yahtzee run correctly.

III.3.2. Performance testing:

We do not plan on having many performance tests (or any at all). The closest thing we might test for that constitutes a performance tests may be something like keeping a high score or checking that a players name gets displayed instead of just a number for them. These tests will cover aspects of code that do not functionally affect the program.

III.3.3. User Acceptance Testing:

Every member of the group will run through the program several times throughout the coding process and even when the code is “complete” to check for any bugs that slipped through unit or integration tests. Ideally each member will try to do different things through each playthrough to better test every aspect of the code. Additionally, we will have people we know play through the game and give feedback. These people will probably roommates, friends, family, etc.

IV. Glossary

Unit Testing:

a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation.

Integration Testing:

the phase in software testing in which individual software modules are combined and tested as a group.

Functional Testing:

a type of testing that seeks to establish whether each application feature works as per the software requirements.

Performance Testing:

a software testing process used for testing the speed, response time, stability, reliability, scalability and resource usage of a software application under particular workload.

User Acceptance Testing:

the last phase of the software testing process actual software users test the software to make sure it can handle necessary tasks in real-world scenarios, according to specifications.

V. References

“Integration Testing - Javatpoint.” *Www.javatpoint.com*, https://www.javatpoint.com/integration-testing.

Hamilton, Thomas. “Performance Testing Tutorial: What Is, Types, Metrics & Example.” *Guru99*, 24 Mar. 2022, https://www.guru99.com/performance-testing.html.