Java the Hutt's Yahtzee

Test and Validation Plans

Java the Hutt

(Dominic Orsi, Conner Diede, Ryan Gavino, and Cameron Williamson)

Course: CPSC 224 - Software Development

Instructor: Aaron S. Crandall

I. Introduction

I.1. Project Overview

The software being tested is a multi-player Yahtzee game with an interactive GUI. Multiple users can play against each other just like traditional Yahtzee to see who can score the most points. Testing is crucial for all aspects of this program from all of the objects and their functionality to the GUI. Major functionalities to test are the start game button, roll button, dice selection after each roll, score line selection after each turn, turn swapping, and the quit button.

I.2. Scope

The purpose of this document is to summarize how we will go about testing our software. We will implement small unit tests on specific functionalities of the objects within the program as well as elements of our GUI such as various buttons for example. We will also use integration testing in our software, mainly to ensure that our code works as expected when we combine it with other group members. Finally, we will go about system testing by playing our game once we are close to a finished product. This will help us find smaller bugs within our system as it all works together.

II. Testing Strategy

1. Identify tests to check the functionality of a class after each commit.
2. Write the unit tests.
3. Group reviews the tests and its software in pull requests.
4. Group comes up with a plan for integration testing on multiple pulled classes.
5. Document bugs and put them in the issues tab on Github for resolving.
6. Repeat steps 3-5 after each round of successful unit tests.
7. Once the model is complete, wire it up to the GUI to start system tests.
8. Have someone un-involved with the project development play the game.

III. Test Plans

Unit testing plans are that for each section of code we write we will also write the unit test for it. By writing the unit test for the code we write it will allow us to debug it quicker if a test fails as the same person who has written the code will have written the unit test for it.

1. Write the method.
2. Then write the unit test for the method just written.
3. Test and make sure the method works properly along with the test.
4. Make sure everything is working and if not it is documented and shared with the team.

Integration testing plans are that the tests will be written together by each person who has written the source code we are merging together to help make the process as simple as possible.

1. Write the integration test during a meeting when merging the code.

2. Make sure everything is working and if not it is documented and shared with the team.

System testing will be done by all of us through the lifecycle of the project. This is to ensure that the GUI is self explanatory and is compatible with all the code written.

1. Before writing code a test will be done to make sure everything works
2. After code has been written another system test will occur to make sure everything again works before pushing.
3. Make sure everything is working and if not it is documented and shared with the team.

Functional testing will be done by the person who wrote the code for the button and another person to make sure it works correctly along with our other tests verifying that it works properly.

1. After writing some GUI code, it will be tested using all the other tests mentioned.
2. A full run through of the GUI will be done before committing.
3. Make sure everything is working and if not it is documented and shared with the team.

Performance testing will be done by everyone to ensure that the code works properly on a variety of machines. This will also help with troubleshooting and debugging in the performance aspects of the project.

1. Will be tested after code for the GUI has been written.
2. Make sure everything is working and if not it is documented and shared with the team.

User acceptance testing will be done by a 3rd party not involved with us to make sure everything works properly when they are testing it. This will also allow us to test our code's performance on different machines as well.

1. Will be given to a user outside the group.
2. They will run the program.
3. Then fill out a sheet documenting everything that happened.
4. We will read over the sheet and then test to try and replicate any problems and fix them.

III.1. Unit Testing

Unit tests will be written and pushed along with the software as it is written. Therefore the unit tests for a specific area of code will be written by the person who wrote that code. Since we are pushing our tests with our software, the unit tests can be reviewed by others in the pull request before it is added to the repo.

III.2. Integration Testing

Integration testing will be used to make sure that our code works together as expected when we merge it. Since this process involves multiple people and more communication than unit testing, we will plan and write most of these tests during our meetings on Mondays and Thursdays.

III.3. System Testing

We will perform system testing by testing our software through the GUI. Once we have a primitive version of our game with a working GUI we can start this type of testing to help us find bugs that weren't found from prior testing.

III.3.1. Functional testing:

Test that all of the buttons and text input areas provide the desired behavior specified in the functional requirements.

III.3.2. Performance testing:

Tests that specified non-functional requirements are working correctly by playing the game. These include features such as the color of the scorecard changing based on whose turn it is and the roll animation.

III.3.3. User Acceptance Testing:

We will have people or groups of people who were not involved with our development process play our game and report any errors found in the game play.

IV. Glossary

Define technical terms used in the document.

V. References

R. Niedermayr, E. Juergens and S. Wagner, "Will My Tests Tell Me If I Break This Code?," 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), 2016, pp. 23-29, doi: 10.1109/CSED.2016.013.

"What Is the Software Testing Life Cycle? A Complete Guide,"
https://www.testim.io/blog/software-testing-life-cycle/