Battleship

Test and Coordination Plans

Git Gud

Rebekah Blazer, Ryder Gallagher, Jack Nealon

Course: CPSC 224 - Software Development

Instructor: Aaron S. Crandall

I. Introduction ----------

I.1. Project Overview

The objective of this group project is to develop a program to run a game of Battleship. First, we will develop a written plan on how we want to approach our program. This includes UML charts, sequence diagrams and other functional requirements identified before starting to code all of which can be found in our FP2 document. We will also develop a comprehensive testing plan to ensure that our program's behavior works correctly and that all methods produce intended results (this document). This will help with smooth development and minimal errors throughout the project.

The main structure of the game will have two players play against each other on a 10x10 game board. Each player will have 5 ships: one carrier (size 5), one battleship (size 4), one cruiser (size 3), one submarine (size 3) and one destroyer (size 2). The ships will be placed on the game board and each player will take turns guessing where the opposing player's ships are. Each player will take turns selecting coordinates to attack. If the player's guess hits one of the opposing player's ships, the coordinate square will be marked with an 'X'. If the attack misses, the coordinate square will be marked with an 'O'. A battleship is sunk when there is an 'X' on every coordinate square that the ship occupies. The game will continue until one player has sunk all of the opposing player's ships. At the end of the game, the winner will be displayed and the option to play again will be given to the user.

The program will utilize an external GUI, which will display each player their respective game boards and provide an interactive experience for the players. The game will be developed using the java programming language and will utilize git and GitHub for collaborative file sharing.

I.2. Scope

This project requires a solid understanding of programming concepts including algorithmic data structures, object-oriented programming, and a familiarity with the Java programming language. Project collaboration is key to plan and design the game, identify potential issues, and to create a development roadmap that outlines the milestones, and progress for the project. Testing will also be a major part of the project as it is crucial to ensure the game's functionality and stability to provide a great end user experience.

Additionally, the purpose of this document is to provide a test plan outline that describes the kinds of testing that will be used to validate our battleship program's functionality and also to provide a guide for the overall progress of the project. More specifically, this document details unit, integration and system testing strategies that we plan to use throughout the development process.

II. Testing Strategy ----------

1. Before functions are created for each class, there will first be unit tests to determine whether or not the default cases for the class are functioning as expected. Each coder will do so for the class they were assigned.
2. If tests fail, add an issue and rework the code until the tests succeed and close the issue.
3. Issues should be detailed and name the function not working as well as what aspect isn't functioning as expected and how it is misbehaving.
4. Start coding functions starting with the highest priority functions.
5. After each function is completed, test it. Testing includes reactions to input, changing variables, etc.
6. Add issues and edit code as needed.

7. Again, issues should be detailed, naming the function that is not working and specifying how it is not working.
8. Once functioning properly, close the issue and push to main.
9. Team members will then test if the classes interact with each other properly.
10. Any errors will be marked with an issue and resolved as a team.

III. Test Plans ----------

III.1. Unit Testing

Unit testing will start as an individual assignment. Team members are to do the tests for the class they are assigned to code and should start with testing the default values of the class. These tests should be made and run before any other functions for the class are written and the coder shall not work on the functions until default values work properly. Once default values are running properly, the team member will start working on the main functions for the class. Tests are recommended for each function to minimize the need for code review when pushing to main, however, not required. Once code is pushed to main, team members will test classes' ability to interact with one another. Any errors found will be worked on together as a team. Once all classes are finished and run error-free, the program will be run and tested as a whole. Again, any errors found are to be resolved as a team.

III.2. Integration Testing

Initially, integration testing will be conducted individually as each person develops their assigned class alongside its respective unit test cases. As unit tests are implemented, the respective integration tests will follow to demonstrate that despite passing unit tests, the program behavior will still function properly to ensure a working program. Additionally, once unit tests are completed and as overlap happens, collaborative integration testing will then fall on the two programmers whose classes (methods or variables or behaviors) overlap. As a framework, some integration tests that can be determined now include:

1. Verify that the game is displayed correctly on the user interface.
    a. That players can interact with the GUI.
2. Test that the game accurately processes player actions and correctly updates the game board.
3. Verify that the game rules are implemented correctly.
    a. Attempts against the game rules are properly addressed (re-enter, skip, etc)
4. Test that the game ends when one person has sunk all of the opposing ships.
5. Verify that players can take turns attacking and receive feedback on the result of the attack.

III.3. System Testing

System testing will be done with a few hard coded scenarios. This will be implemented after integration testing, so we can assume that the game's individual functionality should work, so these tests will see if the game functions as intended. A few things we could try would be:

1. Hard coded version of the game that has the players performing the intended "normal actions" such as putting in strings as their names, placing the pieces away from each other, and guessing both wrong and right locations for the opponent's battleships.

2. Hard coded version of the game where the players clearly make some odd decisions like putting in weird values for their names, guessing everywhere except for the ships, placing ships too close together, etc.

3. Hard coded version of the game where the players make the worst decisions possible like trying to make their name 100 characters long, placing ships on top of each other, etc.

III.3.1. Functional testing:

1. Test if players can take at least 3 turns each, while guessing coordinates (doesn't need to have any ships it just needs to allow the player to guess and to show where they already guessed).
2. Test to see if the player can guess the same spot over and over or if the proper error message will pop up
3. Test to see if the board starts off blank (with no ships on it)
4. Hard code placing ships, then test to see if the board only shows the ships that you placed and if the board is clear of any hit or miss markers
5. Test to see if a ship will sink if all its coordinates are hit
6. Test to see if all the opponent's ships are sunk, and that the player is declared a winner.

III.3.2. User Acceptance Testing:

The last step of testing for us will be to select 2 friends to play the game with us and to take note of any bugs we have (if any). If there are any bugs, we will work to rectify those issues before repeating the testing process. If we have extra time, we may take suggestions and make cosmetic fixes and improvements to optimize the UI.

IV. Glossary ----------

- **Unit test:** A test that focuses on a small piece of the program (a unit).
- **Integration test:** A test for program behaviors and processes.
- **Functional test:** A test for mandatory methods and behaviors that are essential to make the program run.
- User acceptance testing: the final stage of software development where QA testers test the program before launching the final product.
- **Default case:** When the class is implemented, what are the automatic states of the class?
- **High priority function:** Functions that are detrimental to the game running properly and fulfilling functional requirements.
- **Issue:** GitHub issue. Assigned to people and used to keep track of problems and any requirements/tasks that need to be completed.
- **Class:** An object of the program. In our cases, "Player", "Ship", "Board", etc.
- **Function:** A portion of the class that fulfills a certain task.
- **Bug(s)**: An error, flaw or fault in the design, development, or operation of the program

V. References ----------

Hasbro. "Battleship Game Instructions." Hasbro, 2009. https://www.hasbro.com/common/instruct/battleship.pdf.