

SE 3XA3:Test Plan: Revision 0

Team 03, Pongthusiastics
Adwity Sharma - sharma78
Arfa Butt - buttaa3
Jie Luo - luoj3

Contents

1	General Information	3
1.1	Purpose	3
1.2	Scope	3
1.3	Acronyms, Abbreviations, and Symbols	3
1.4	Overview of Document	4
2	Plan	4
2.1	Software Description	4
2.2	Test Team	4
2.3	Testing Tools	5
2.4	Testing Schedule	5
3	System Test Description	5
3.1	Tests for Functional Requirements	5
3.1.1	Mode Selection	5
3.1.2	Game State	10
3.2	Tests for Nonfunctional Requirements	11
3.2.1	Usability	11
3.2.2	Operating System Support	12
3.2.3	Spelling and Grammar	12
3.2.4	Hardware Requirement	12
3.2.5	Entertainment	13
3.2.6	Challenge	13
3.2.7	Contols	13
4	Tests for Proof of Concept	14
4.1	Game Modes	14
4.2	Paddle and Ball Movement	15
5	Comparison to Existing Implementation	16
6	Unit Testing Plan	16
6.1	Unit Testing of Internal Functions	16
6.2	Unit Testing of Output Files	16

1 General Information

1.1 Purpose

The purpose of this report is to verify that the software has been tested properly and that it was implemented correctly.

1.2 Scope

This document provides a basis for testing the functionality and the properties of the ping pong game after re-implementation it. It tests the abilities and limits of the game. It also documents the aspects of the game that are to be tested. Setting the testing criteria makes it possible to gauge the degree of success or failure of the software. Also completing this document before finalizing the project helps determine what aspects can lead to failure of the software.

1.3 Acronyms, Abbreviations, and Symbols

Terms	Definitions
SRS	Software Requirements Specification
Users	Players of the game
The Project	The pong game that is being reconstructed.
Product	The game that is being developed.
Java	Java programming language.
Git	The GitLab website.
Windows	Microsoft windows.
Customer	Anyone who would like to use this game.
3XA3 team	Professor, course coordinators, teaching assistant and any other personnel responsible for running of the 3XA3 course.

Structural testing Structural testing is the testing of the internal structure of the software. Also called white box testing.

Functional testing Functional testing is the testing of the program's function. Also called black box testing.

Dynamic testing Testing done by running the program and checking the result against expected behaviour.

Static testing Testing done without executing a program and it is generally done in the requirements and design stage.

Manual testing Testing of a software manually by hand.

Automated testing Testing is done automatically by the software.

1.4 Overview of Document

The fault in our pong provides a improved reimplementatation of the pong game found in this link: <https://github.com/mihneadb/Pong>. The objectives of this game has been detailed in the SRS document, found in the gitlab repository for this project. The SRS document details all the requirements and functionality of the project that we hope to achieve. This document details the testing of those requirements and functionality.

2 Plan

2.1 Software Description

This software will allow users to play the classic game of Ping Pong, with different features added to it, if they desire. The game will give the user the option to either play a normal ping pong game, or choose one of the new modes that have been added to this game. These modes will include a multiplayer option as well as a ping pong game with obstacles added to the playing field. This software will allow users to play an updated version of ping pong while still preserving the spirit of the original game. The entire game will be implemented in Java.

2.2 Test Team

The entire testing process, including writing and executing test cases, will be done by the following members:

- Adwity Sharma
- Arfa Butt
- Jie Luo

2.3 Testing Tools

The Junit package, available in Java by default, is the tool that will be used for automated testing. It will be used to validate all major classes and function.

2.4 Testing Schedule

See Gantt Chart at the following url:

<https://gitlab.cas.mcmaster.ca/Group3/FaultInOurPong/blob/master/ProjectSchedule/GanttChart.pdf>

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Mode Selection

3.1.1.1 Open New Game

1. FS-NG-1: When user chooses new game, a new page with the options for selecting a new game should open

Type: Functional, Dynamic, Manual

Initial State: Menu page

Input: Button click

Output: New game page

How test will be performed: Run the program and check if appropriate page opens.

2. FS-NG-2: Single Player Mode chosen
Type: Functional, Dynamic, Manual
Initial State: New game page
Input: Button click
Output: New single player mode game is started
How test will be performed: Run the program and check if appropriate game mode starts.
3. FS-NG-3: Advanced Single Player Mode chosen
Type: Functional, Dynamic, Manual
Initial State: New game page
Input: Button click
Output: New advanced single player mode game is started
How test will be performed: Run the program and check if appropriate game mode starts.
4. FS-NG-4: Multiplayer Mode chosen
Type: Functional, Dynamic, Manual
Initial State: New game page
Input: Button click
Output: New multiplayer mode game is started
How test will be performed: Run the program and check if appropriate game mode starts.
5. FS-NG-5: Back button to go back to the menu page
Type: Functional, Dynamic, Manual
Initial State: New game page
Input: Button click
Output: Menu page
How test will be performed: Run the program and check if appropriate page opens.

3.1.1.2 Load Game

1. FS-LG-1: Load a saved game instead of starting a new one

Type: Functional, Dynamic, Automated

Initial State: Menu page

Input: Load previously saved game state

Output: The scores aren't reset to zero

How test will be performed: Check that the scores do not start at zero through automated testing. This testing approach may give us errors if the user saved a game state with a zero score. However, this can be overcome by implementing it so that a saved game state with a score of zero will be treated as a new game.

2. FS-LG-2: Game loaded is same as the game state that was saved the last time

Type: Functional, Dynamic, Manual

Initial State: Menu page

Input: Button click

Output: The scores, game mode, and the speed should be the exact same as ones in the last saved game.

How test will be performed: Run the program and check if the data from the saved game state matches the data for the game that is loaded.

3.1.1.3 Change Speed

1. FS-CS-1: Open speed change page with fast, normal, and slow options

Type: Functional, Dynamic, Manual

Initial State: Menu page

Input: Button click

Output: Speed change page

How test will be performed: Run the program and check if appropriate page opens.

2. FS-CS-2: Once option is chosen, go back to menu page so user can start a new game
Type: Functional, Dynamic, Manual
Initial State: Speed change page
Input: Button click (one of the three speeds)
Output: Menu page
How test will be performed: Choose a speed, and check if appropriate page opens.
3. FS-CS-3: Check that single player mode has the updated speed
Type: Functional, Dynamic, Manual
Initial State: Menu page
Input: Button clicks
Output: Single player mode game starts with the ball having the appropriate speed.
How test will be performed: Run the program, start a single player mode game and verify the speed.
4. FS-CS-4: Check that advanced single player mode has the updated speed
Type: Functional, Dynamic, Manual
Initial State: Menu page
Input: Button clicks
Output: Advanced single player mode game starts with the ball having the appropriate speed.
How test will be performed: Run the program, start an advanced single player mode game and verify the speed.
5. FS-CS-5: Check that multiplayer mode has the updated speed
Type: Functional, Dynamic, Manual
Initial State: Menu page
Input: Button clicks

Output: Multiplayer mode game starts with the ball having the appropriate speed.

How test will be performed: Run the program, start a multiplayer mode game and verify the speed.

3.1.1.4 Highscores

1. FS-HS-1: Open highscores page when option is chosen from the menu page

Type: Functional, Dynamic, Manual

Initial State: Menu page

Input: Button click

Output: Highscores page

How test will be performed: Run the program, and check if appropriate page opens.

2. FS-HS-2: Add a new highscore and check if it is added at the right place in the highscores list

Type: Functional, Dynamic, Manual

Initial State: Menu page

Input: Button clicks

Output: Highscores page

How test will be performed: Run the program, play a single player mode game, and make a highscore. Go back to the menu page, open the highscores list and check if the new highscore was added or not. Also, make sure that the highscore was added at the correct rank.

3.1.1.5 Tutorial

1. FS-TU-1: Open tutorial page when option is chosen from the menu page

Type: Functional, Dynamic, Manual

Initial State: Menu page

Input: Button click

Output: Tutorial page

How test will be performed: Run the program, and check if the tutorial page opens.

2. FS-TU-2: Back button to go back to the menu page

Type: Functional, Dynamic, Manual

Initial State: Tutorial page

Input: Button click

Output: Menu page

How test will be performed: Run the program and check if appropriate page opens.

3.1.2 Game State

1. FS-GS-1: Paddle movement

Type: Functional, Dynamic, Manual

Initial State: Game

Input: Left key pressed

Output: User's paddle is moved left

How test will be performed: Run the program, open a game and press left key. Check that the paddle moved left on the console.

2. FS-GS-2: Paddle movement

Type: Functional, Dynamic, Manual

Initial State: Game

Input: Right key pressed

Output: User's paddle is moved right

How test will be performed: Run the program, open a game and press right key. Check that the paddle moved right on the console.

3. FS-GS-3: Increment scores

Type: Functional, Dynamic, Manual

Initial State: Game

Input: One of the users misses a turn

Output: The other user's score should increase by 1

How test will be performed: Run the program, open a multiplayer game and miss one of the player's turn. Check that the other player's score increased by 1.

4. FS-GS-4: Decrease lives (in single player mode)

Type: Functional, Dynamic, Manual

Initial State: One of the single player modes game

Input: Miss the ball (when thrown by the computer)

Output: User's life should decrease by 1

How test will be performed: Run the program, open one of the single player modes game and miss one turn. Check that the player's lives decreased by 1.

3.2 Tests for Nonfunctional Requirements

3.2.1 Usability

1. FN-1: This testing is done to ensure that the re-designing of the game has made the game better than it was originally.

Type: Structural, static, manual

Initial State: Users group have already downloaded and played the original version of the pong game. After playing the game they have also filled in the survey that they were asked to fill in, so that we could have something to compare our game's ability against. The usability and entertainment factors are the areas we are most focused on for the game.

Input: the same group of user are given our re-designed game and asked to fill in the same survey.

Output/ result: The user response for each major category of the survey is tallied, for both the surveys. Based of that tally, the original game and the redesigned version are given a rating for the usability and entertainment factor of the game.

How test will be performed: The same group of the user who participated in playing the initial version of the game are made to play the new updated version of the game. They are then given the same survey questions to fill in, as they had filled in after the original game. The result for both versions are tallied. If the updated version receives more favourable responses from the user for the game, then it can be said that this re-design of the game is successful.

3.2.2 Operating System Support

2. FN-2: This test is done to confirm that the game runs in all major operating systems such as Windows, Mac OS, Linux etc.

Type: Functional (dynamic, manual)

Tester: Development team

Pass: The game can be compiled and run in each of the platform and change the game to make sure that it runs in each operating system.

3.2.3 Spelling and Grammar

3. FN-3: Making sure that the game does not have any grammar or spelling errors.

Type: Functional (dynamic, manual)

Tester: Development team

Pass: The development team checks to see if the game has any grammar or spelling errors in it and all the errors are corrected.

3.2.4 Hardware Requirement

4. FN-4: Making sure that the speed of game doesn't change in different machines.

Type: Functional (dynamic, manual)

Tester: Development team

Pass: The development team runs it in their respective computers at the same time to see that the game runs at the same speed in all the computers.

3.2.5 Entertainment

5. FN-5: This testing is done to determine if the game is entertaining enough for the user.

Type: Functional (dynamic, manual)

Tester: Testing group

Pass: First the testing group is made to play the game and complete the survey relating to it. Then the survey response is tallied and if the average vote says that they deem it entertaining then we pass this test.

3.2.6 Challenge

6. FN-6: This testing is done to determine if the game is challenging enough for the user.

Type: Functional (dynamic, manual)

Tester: Testing group

Pass: First the testing group is made to play the game and complete the survey relating to it. Then the survey response is tallied and if the average vote says that they rate it medium for challenging then we pass this test. A medium means that the game isn't too hard but isn't so easy that it gets boring.

3.2.7 Contols

7. FN-7: This testing is done to determine if the game controls are intuitive.

Type: Functional (dynamic, manual)

Tester: Testing group

Pass: First the testing group is made to play the game and complete the survey relating to it. Then the survey response is tallied and if the average vote is high then we pass the test.

4 Tests for Proof of Concept

Proof of concept focuses on the automated testing of the software and verifying that the results are the same. This testing process is more accurate than manual testing because human bias or human errors do not affect the results of this type of testing. Automated testing would require the ability to check if the various modes of the games are rigorously implemented and that the direction and paddles used for the game changes directions accordingly.

4.1 Game Modes

1. PC-1: Checking to see that the games starts with scores set at 0

Type: Functional, Dynamic, Automated

Initial State: New game page

Input: make a new game state

Output: the score at the beginning of the game

How test will be performed: A new game state is created and the score at the beginning of the game is returned and the results are evaluated to see that it matches the expected result. If the results match, then the test unit returns true.

2. PC-2: Checking to see that the load game mode starts correctly

Type: Functional, Dynamic, Automated

Initial State: Load game page

Input: make a load game state

Output: the score at beginning of the loaded game and end of the old game

How test will be performed: A load game state is created and the score at the beginning of the game is returned and result is compared with the score returned at the end of the save game. If the results match, then the test unit returns true.

3. PC-3: Checking to see that the high score function is implemented correctly

Type: Functional, Dynamic, Automated

Initial State: New game page

Input: make a new game state

Output: the score at the end of the game

How test will be performed: A new game state is created and the score at the end of the game is returned and the over all high scores stored is checked to see if the new high score is added to the list or not. It returns true if the new high score is added only if player beat the old saved records.

4.2 Paddle and Ball Movement

1. PC-4: Checking to see that the direction of the ball changes

Type: Functional, Dynamic, Automated

Initial State: New game page

Input: initial speed and position of the ball

Output: the x and y coordinates and speed of the ball

How test will be performed: A new game state is created and the position of the ball at the beginning of the game is returned and then position and speed of the ball at various stages are returned. If the ball changes direction, then the coordinates should go to negative. If this is happening, then the test function returns true. Otherwise, it returns false.

2. PC-5: Checking to see if the paddle changes direction correctly

Type: Functional, Dynamic, Automated

Initial State: New game page

Input: the initial position of the paddle

Output: the coordinates of the paddle at various stages of the game.

How test will be performed: A new game state is created and the initial position of the paddle is returned in the test unit. When the paddle

changes direction, the new coordinates of the paddle is returned. If the coordinates change, then the test function would return true.

5 Comparison to Existing Implementation

There is one test that compares the modified game to the original game. Please refer to:

- test FN-1 in Tests for Usability in Nonfunctional Requirements

6 Unit Testing Plan

JUNIT testing from the java built in function will be used to test this programme.

6.1 Unit Testing of Internal Functions

For the testing of the internal functions of this project, function that return a value can be tested automatically. We will take each function and give it an input and see if the input matches the expected output in the testing unit. We will try to include all the boundary cases as input for our unit testing, so that we know all the limitations of the function. For the game to be certified through unit testing of internal functions, the game should return the same result as expected for at least 95% of the functions that are tested. Also we can perform a white box testing of the functions to see that the code has been well formatted and that the naming convention is followed properly. In addition, we will try to make sure that the mvc model has been implemented correctly while we are conducting the white box testing.

6.2 Unit Testing of Output Files

To accurately test the output files of the project, the output of the game for each of the function will be compared against the expected output for the files. Automated testing can be implemented for some of the files. Automated testing ensures that the output is accurate to the desired outputs. Playing the game and checking each component of the game. And making sure that the functionality of the game is same is important. We use an action, expected

and result table to conduct this testing. Each feature or possible scenario in the game is listed in the action column. The expected result for each action is listed in the expected column. The actual output for the action is listed in the result column. If the expected and result match, then we can say that the result is pass. To ensure accuracy of the game's functionality, we should get pass result in at least 95% of the test results.