TestPlan

SFWRENG 3XA3

Group 30, Team 30
Alan Yin (yins1)
Huajie Zhu (zhuh5)
Junni Pan (panj10)

October 27, 2017

Contents

List of Tables

List of Figures

1 Major Revision History

Date	Version	Note
Oct 22,2017	1.0	draft
Oct 23,2017	1.1	add all information to document
Oct 24,2017	1.2	Final version for test plan rev 0
Dec 06,2017	1.2	Final version for test plan rev 1

2 General Information

2.1 Purpose

This software testing is aimed to verify the functionality of the whole project, and to identify any potential bugs and malfunctions which may occur in program compilation and running time. By completing the test plan, we would like to make sure that our program for the project satisfies the original requirements specified in SRS.

2.2 Scope

This Test Plan includes the procedure for testing all the major functionality and features specified in requirement documents. It includes a variety of test cases to ensure that the Tower Defense Game functions properly, including programming language syntax, game algorithms, user interface, animations and audios, and unit tests for each class and methods. A separate test for the whole program will also be included for integration testing.

2.3 Acronyms, Abbreviations, and Symbols

Term	**Definition**
PoC	Proof of Concept
SRS	Software Requirements Specification
GUI	Graphical User Interface
Slick2D	Slick2D Library
API	Application Programming Interface
HP	Health Points
DPS	Damage Per Second
FS	Functional system test
SS	Structural system test

Table 1: Table of Abbreviations

Term	**Definition**
Structural Testing	Testing derived from the internal structure of the software.
Functional Testing	Testing derived from a description of how the program functions.
Dynamic Testing	Tests that have test cases run while the program is executing.
Static Testing	Testing that does not involve program execution.
Manual Testing	Tests done manually.
Automated Testing	Tests that are done automatically by software.

Table 2: Table of Definitions

2.4 Overview of Document

The project will redevelop and make refinement on an open sourced tower defense game. The software will allows users to build tower in given map to attack the critter. All the software's requirements are numbered in the Requirements Document.

3 Plan

3.1 Software Description

The game allows players to construct towers on the map, to defend the enemies who are supposed to attack through a pre-determined route. The game is implemented in Java.

3.2 Test Team

The individuals responsible for testing are Junni Pan, Alan Yin, and Huajie Zhu.

3.3 Tools Used for Testing

The tool that will be used for this project is JUnit, combined with a built-in tower defense map editor. JUnit is a unit testing framework. It will be used to automate the unit testing. The built-in map editor function will also be used to create proper test cases, to assist integration testing.

3.4 Testing Schedule

Task	**Team Member**	**Date**
Mouse Input & Button Navigation	Junni Pan	October 24th 2017
Menu Animation and Background Music	Junni Pan	October 27th 2017
Loading Animation	Junni Pan	October 30th 2017
Difficulty Selecting	Alan Yin	November 2nd 2017
Tower and Critter Interaction	Alan Yin	November 4th 2017
Enemy & Tower Animation	Alan Yin	November 7th 2017
Map Loading	Huajie Zhu	November 10th 2017
Player Status Determination	Huajie Zhu	November 11th 2017
Game Pressure Test	Huajie Zhu	November 13th 2017

Table 3: Testing Assignments

4 System Test Description

4.1 Tests for Functional Requirements

4.1.1 User Input and the Response

1. FS-IR-1: Main Menu Button Type: Functional, Dynamic, Manual

   Initial State: The game opened successfully

   Input: Mouse left clicks on Start button

   Output: The start button shows the visual effect and the game panel is loaded

   How test will be performed: We will check if the button have a graphic effect when it is been clicked. When the player clicks on the Play Now button, it will go to the difficulty selection stage.

2. FS-IR-3: Check critter info

   Type: Functional, Dynamic, Manual

   Initial State: Game interface

   Input: Mouse left clicks on the Critter info button area

   Output: The corresponding button shows the visual effect and shows the information of the all the critters.

   How test will be performed: We will check if player left clicks Critter info button, there is a visual change on button. And then there is an separate window showing the information of the critters.

3. FS-IR-4: Start Wave Button

   Type: Functional, Dynamic, Manual

   Initial State: Game interface

   Input: Mouse left clicks on the Start Wave button

   Output: The corresponding button shows the visual effect and the next wave starts.

   How test will be performed: We will check if player left clicked Start Wave button, there is a visual change on button. And the button becomes not clickable. The program then generates the corresponding wave of critters starting from the beginning to the end of the map. After all critters died, the Start Wave button becomes clickable and the wave number will increase by one.

4. FS-IR-5: Speed Bar

   Type: Functional, Dynamic, Manual

   Initial State: Game interface Input: Mouse left click and drag the speed adjustment bar to the speed level from 1 to 5.

   Output: The drag bar shows the visual effect and the game speed changes.

   How test will be performed: We will check if player left clicks and drags the speed bar to the corresponding speed(1 to 5), the running speed of the game changes accordingly.

5. FS-IR-6: Pause Button

   Type: Functional, Dynamic, Manual

   Initial State: Game interface

   Input: Mouse left clicks on the Pause button Output: The corresponding button shows the visual effect and the action of all the critters and towers stops. How test will be performed: We check if player left clicks pause button, there is a visual change on button and the action of all critters and towers stop moving. The player can still do the building, upgrading and selling operations.

6. FS-IR-7: Main Menu Button

   Type: Functional, Dynamic, Manual

   Initial State: Game interface

   Input: Mouse left clicks on the Main Menu button

   Output: The corresponding button shows the visual effect and the game will return to the main menu.

   How test will be performed: We check if player left clicks Main menu button, there is a visual change on button, and the current wave ends then it goes to the initial main menu.

7. FS-IR-8: Attack Strategy Drop-down Box

   Type: Functional, Dynamic, Manual

   Initial State: After the player choose a tower

   Input: Mouse left clicks on the tower attack strategy drop-down menu.

   Output: The corresponding button shows the visual effect and shows 5 strategies to choose.

   How test will be performed: We check if player chooses the tower and left clicks on the tower attack strategy drop-down menu, there are 5 tower attack strategies to choose(Closest, Farthest, Fastest, Weakest, Strongest).

8. FS-IR-9: Upgrade Button

   Type: Functional, Dynamic, Manual

   Initial State: After the player choose a tower

   Input: Mouse left clicks on the Tower upgrade button.

   Output: The corresponding button shows the visual effect and the tower can be upgraded if conditions are met.

   How test will be performed: We check if player chooses the tower and left clicks on the Tower upgrade button, there is a visual change on button. And if possible, the required amount of money will be deducted from user and the tower will upgrade to the next level.

9. FS-IR-10: Sell Button

   Type: Functional, Dynamic, Manual

   Initial State: After the player choose a tower

   Input: Mouse left clicks on the Tower sell button.

   Output: The corresponding button shows the visual effect and the selected tower will be sold.

   How test will be performed: We check if player chooses the tower and left clicks on the Tower sell button, there is a visual change on button. At the same time, the player's money will increase according to selling price and the tower will disappear from the map.

10. FS-IR-11: Fire Tower Button

    Type: Functional, Dynamic, Manual

    Initial State: Game interface

    Input: Mouse left clicks on the fire tower button.

    Output: The corresponding button shows the visual effect and the player can choose a place to build a fire tower if possible.

    How test will be performed: We check if the player has enough money and left clicks on the fire tower button, there is a visual change on button. And the player can move the mouse on the screen, there is a fire tower icon behind the mouse. If possible, the player can build the fire tower and the required amount of money will be deducted.

11. FS-IR-12: Ice Tower Button

    Type: Functional, Dynamic, Manual

    Initial State: Game interface

    Input: Mouse left clicks on the ice button

    Output: The corresponding button shows the visual effect and the player can choose a place to build a ice tower if possible

    How test will be performed: We check if the player has enough money and left clicks on the ice tower button, there is a visual change on button. And the player can move the mouse on the screen, there is a ice tower icon behind the mouse. If possible, the player can build the ice tower and the required amount of money will be deducted.

12. FS-IR-13: Laser Tower Button

    Type: Functional, Dynamic, Manual

    Initial State: Game interface

    Input: Mouse left clicks on the laser button

    Output: The corresponding button shows the visual effect and the player can choose a place to build a laser tower if possible

    How test will be performed: We check if the player has enough money and left clicks on the laser tower button, there is a visual change on button. And the player can move the mouse on the screen, there is a laser tower icon behind the mouse. If possible, the player can build the laser tower and the required amount of money will be deducted.

13. FS-IR-14: Area Attack Tower Button

    Type: Functional, Dynamic, Manual

    Initial State: Game interface

    Input: Mouse left clicks on the Area Attack button.

    Output: The corresponding button shows the visual effect and the player can choose a place to build a area attack tower if possible.

    How test will be performed: We check if the player has enough money and left clicks on the area attack tower button, there is a visual change on button. And the player can move the mouse on the screen, there is a area attack tower icon behind the mouse. If possible, the player can build the area attack tower and the required amount of money will be deducted.

14. FS-IR-15: N Button

    Type: Functional, Dynamic, Manual

    Initial State: After the player finishing building a tower

    Input: Mouse left clicks on the N button.

    Output: The corresponding button shows the visual effect and the icon behind the mouse disappear.

    How test will be performed: We check if user left clicks on the N button, there is a visual change on button. After then, the player moves the mouse on the screen, there is no tower icon behind the mouse.

4.1.2 Game Logic

15. FS-GL-1: Enemy track

    Type: Functional, Dynamic, Manual

    Initial State: A wave is running.

    Description: All the enemies will walk along the right path one by one.

    How test will be performed: We check if a wave started, the enemies move on the right path, and no exceptions occur.

16. FS-GL-2: Tower Attack

    Type: Functional, Dynamic, Manual

    Initial State: A wave is running.

    Description: The tower will attack the enemies within its attack range based on the chosen strategy.

    How test will be performed: We check if the enemy entering the tower's attack range, the tower attacks on an enemy chosen by its strategy.

17. FS-GL-3: Attack Strategy

    Type: Functional, Dynamic, Manual

    Initial State: A tower is attacking

    Input: Selected Strategy

    Output: The tower will attack the enemy based on the chosen strategy.

    How test will be performed: We check if the attack way of a tower works as the strategy description(Closest, Farthest, Fastest, Weakest, Strongest). Besides, each tower can have a unique strategy.

18. FS-GL-4: Tower Strengthen

    Type: Functional, Dynamic, Automated

    Initial State: A tower has been upgraded

    Description: The tower will become stronger after upgrading, its damage per hit will increase.

    How test will be performed: We set different level of waves and check if the upgraded tower could do high damage than normal tower. We will use JUnit here for accurate result.

19. FS-LG-5: Difficulty Increase

    Type: Functional, Dynamic, Automated

    Initial State: A wave is ended

    Input: New wave started

    Output: A stronger wave of enemies will come to the path.

    How test will be performed: We will use JUnit here to read data from all the possible waves to find out if every wave is stronger than the previous one.

20. FS-LG-6: Stage Change

    Type: Functional, Dynamic, Manual

    Initial State: One stage is clear

    Input: The player enters the next stage.

    Output: A new map which is harder and more complex will be loaded and the game will be reset.

    How test will be performed: We check if one stage is clear, the next stage is loaded, all the tower on the map will be deleted and the player's money is reset to the default amount.

4.1.3 Animation Effect

21. FS-ANE-1: Enemy Animation

    Type: Functional, Dynamic, Manual

    Initial State: The game state.

    Input: The player start a new wave.

    Output: The enemies move along the path with its own distinct animation effect.

    How test will be performed: We will check if all 6 kinds of enemy have their own distinct animation when they walk along the path.

22. FS-ANE-2: Tower Animation

    Type: Functional, Dynamic, Manual

    Initial State: The game state.

    Input: The player upgrade a new tower and there exists enemy in the tower attack range

    Output: The tower's trajectory is strengthened after upgrade.

    How test will be performed: We will check if all 4 kinds of tower's trajectory is upgraded as the tower upgrades, by manually upgrade the towers.

4.1.4 Audio Effect

23. FS-AUE-1: Main Menu Audio

    Type: Functional, Dynamic, Manual

    Initial State: The game installed successfully.

    Input: The player opens the game.

    Output: The audio of main menu works properly.

    How test will be performed: We will check if the audio of main menu works properly after we open the game.

24. FS-AUE-2: Game Stage Audio

    Type: Functional, Dynamic, Manual

    Initial State: Level selecting menu.

    Input: The player enter a game stage.

    Output: The audio of game stage works properly.

    How test will be performed: We will check if the audio of game stage works properly after we enter a stage

4.2 Tests for Nonfunctional Requirements

4.2.1 Usability

25. SS-1

    Type: Structural, Dynamic, Manual

    Initial State: Program installed onto system

    Input/Condition: Launch program on Windows OS, Mac OS, or Linux.

    Output/Result: The game successfully runs on Window OS, Mac OS, or Linux.

    How Test Will Be Performed: The program will be installed on computers. we will try to launch program under Windows, Mac OS, and Linux to check the capability.

26. SS-2

    Type: Structural, Dynamic, Manual

    Initial State: Program installed onto system

    Input/Condition: launch program

    Output/Result: An average frame rate per second

    How Test Will Be Performed: First, program will be installed onto computers.Then we launch program and run the game. Next, check the average frame rate per second to determine if it meets the minimum stable frame rate.

4.2.2 Performance

27. SS-3

    Type: Structural, Dynamic, Manual

    Initial State: Program installed onto system

    Input/Condition: launch program and start game.

    Output/Result: the game will not crash after 10 rounds.

    How Test Will Be Performed: First, program will be installed onto computers.Then we launch program and start the game. We would like to check whether this game crashes with in 10 rounds.

5 Tests for Proof of Concept

Proof of Concept testing will mainly concentrate on verifying the functionality of external gaming library, which can be specified into animation, audio and user interface implementation.

5.1 Animation usage

28. PoC-1: Main Menu Animation

    Type: Functional, Dynamic, Manual

Initial State: The game installed successfully.

Input: The player opened the game.

Output: The animation of main menu shows properly.

How test will be performed: We will check if the animation of main menu shows properly after we open the game.

29. PoC-2: Enemy Animation

    Type: Functional, Dynamic, Manual

    Initial State: Inside a game stage.

    Input: The player starts a new wave.

    Output: The enemy moves on the path with its own distinct animation effect.

    How test will be performed: We will check if all 6 kinds of enemy have their own distinct animation when they move along the path.

30. PoC-3: Tower Animation

    Type: Functional, Dynamic, Manual

    Initial State: Inside a game stage.

    Input: The player build a new tower and there exists an enemy in the tower attack range.

    Output: The tower's trajectory has its animation effect.

    How test will be performed: We will check if all 4 kinds of tower's trajectory have their own distinct animation when the tower attack a enemy.

5.2 Audio usage

31. PoC-4: Main Menu Audio

    Type: Functional, Dynamic, Manual

    Initial State: The game installed successfully.

    Input: The player opens the game.

    Output: The audio of main menu works properly.

How test will be performed: We will check if the audio of main menu works properly after we open the game.

5.3 user interface implementation

32. PoC-5: user interface

    Type: Functional, Dynamic, Automated

    Initial State: The game installed successfully

    Input: The player opens the game.

    Output: User interface appear on the screen properly.

    How test will be performed: We will check if the proper combination of slick2D window and Java Swing interface is shown on the screen, with all the images, buttons, drop-down menus and drag bars positioned correctly.

6 Unit Testing Plan

The JUnit framework will be used to do unit testing for this project.

6.1 Unit testing of internal functions

We will create unit tests for the internal functions in order to have all the methods tested. we will give the input values and set corresponding output values, and unit tests will automatically test and give detailed tests results. Our project do not need to import any stubs and drivers for testing. Coverage metrics are used to check how much are covered by our codes. We are trying to cover as much as possible codes to make sure the tests function well.

7 Traceable Matrices

7.1 Trace to Modules

Modules	Tests
M5,M16,M17,M18,M19,M20	FS-IR-1, FS-AUE-1
M3,M6,M7,M8	FS-ANE-1, FS-ANE-2, UT-1, UT-2, UT-3, UT-4, UT-5
M3,M7,M10,M16	FS-GL-1, FS-GL-2, FS-GL-3, FS-GL-4
M5,M2,M16,M17,M9	FS-AUE-2, FS-IR-2 14
M1,M4,M17	FS-LG-5, FS-LG-6
M17,M18,M19	FS-IR-2 14
M16	FS-IR-2 14
M1	SS-1
M2,M4	SS-3
M1	SS-1

Table 4: Trace Between Modules and Tests

7.2 Trace to Requirements

Requirements	Tests
FR1	FS-IR-1, FS-AUE-1
FR2	FS-ANE-1, FS-ANE-2, UT-1, UT-2, UT-3, UT-4, UT-5
FR3	FS-GL-1, FS-GL-2, FS-GL-3, FS-GL-4
FR4	FS-AUE-2, FS-IR-2 14
FR5	FS-LG-5, FS-LG-6
NF1	FS-IR-2 14
NF3	SS-1
NF2	FS-IR-2 14
NF4	SS-3
NF5	SS-1

Table 5: Trace Between Requirements and Tests

8 Code Coverage Metrics

The tower defense development team, Group 30, has managed to cover over 90% of all the code in this project in the tests. Referring to the traceable matrices above, all the modules and corresponding requirements are going to be tested, with most of them tested more than once. This proves that almost all the code can be tested and verified.