SE 3XA3: Test Plan
GardenDefender

Team 29
Ashley Williams, willia18
Declan Mullane, mullanem
Leo Shi, shiy12

October 26, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
October 25	1.0	Updated section 7.2
October 25	1.1	Updated section 3.1 and all subsections
October 26	1.2	Updated sections 1, 2, 3, 4, 5, 6, 7 and all subsections

1 General Information

1.1 Purpose

The purpose of this document is to provide a detailed description of the test plan for the GardenDefender. The test plan itself aims to prove that the functions of the game execute as specified in the requirements document, that non-functional requirements are met, and to catch and fix any bugs that may arise from testing.

1.2 Scope

Since GardenDefender is a game, the majority of testing will be done manually. It is coded primarily in the JavaScript programming language; as such, Mocha, a test framework designed for JavaScript, will be used for any additional unit testing. Surveys will also be used to test non-functional requirements and usability, where testers will be asked to evaluate properties of the game.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	Definition
GUI	Graphical User Interface
FR	Functional Requirements
NFR	Non-Functional Requirements
FR-Test	Functional Requirements Test
NFR-Test	Non-Functional Requirements Test

1.4 Overview of Document

This document describes the tests to be run for functional and non-functional requirements, to ensure that the game executes as expected. It is broken down as described in the table of contents, with specific tests further broken down as automated or manual, dynamic or static.

Table 3: **Table of Definitions**

Term	**Definition**
JavaScript	Programming language used to build GardenDefender.
Mocha	Testing framework designed for JavaScript unit tests.
User	The individual who is playing the game.
Player	The graphical representation of the player character.
Enemy	The graphical representation of the enemy character.
Projectiles	The weapon the player fires at enemies to destroy them.
Defensive Line	The left boundary of the GUI.
Health	Numerical value denoting player's health value.
Health Bar	Visual representation of the player's health.
Timer	Timer counting down the time until the game play ends.
GUI	Display of the game.
Tester	Individual who will carry out manual testing.
Level	Different stages of progression in the game play.
Game State	The state of the game.
Load State	The initial game state that loads the game assets.
Start State	The game state that allows the user to begin the game.
Main State	The game state where the game play takes place.
Pause State	The game state that allows the user to pause the game play.
Win State	One of the final game states; indicates the user has won.
Lose State	One of the final game states; indicates the user has lost.

2 Plan

2.1 Software Description

GardenDefender is a basic shooter game where a player fires projectiles at oncoming enemies. As such, the software is responsible for generating and displaying both player and enemies, as well as projectiles. The software also displays the player's health and a timer, where the goal of the game is to prevent the health bar from reaching zero before the timer runs out, at which point the player wins the game.

The game is divided into separate game states (load, start, main, pause, win, and lose) that track different aspects of the game.

2.2 Test Team

The testing team will include Ashley Williams, Declan Mullane, and Leo Shi.

2.3 Automated Testing Approach

Automated testing is unsuitable for a large portion of the game code. For the code where automated testing is applicable, unit testing will be implemented with Mocha, and will primarily be utilized to ensure that the game proceeds from one game state to the next.

2.4 Testing Tools

Mocha will be the testing tool for unit testing; a survey will collect data on the satisfaction of non-functional requirements. Manual test cases will also be implemented to supplement cases that cannot be tested automatically.

2.5 Testing Schedule

See Gantt Chart at the following url: GardenDefender Gantt Chart

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Game State Transitions

1. F1-Test

   Load state to start state
   Type: Functional, Dynamic, Manual

   Initial State: Load state

   Input: The start state is invoked through the load state.

   Output: Start state

   How test will be performed: Upon the start of a new game, the tester will determine if the start screen is displayed in the GUI.

2. F2-Test
   Start state to main state

   Type: Functional, Dynamic, Manual

   Initial State: Start state

   Input: The space-bar key is pressed

   Output: Main state

   How test will be performed: At the start screen, the tester will press the space-bar key and then determine if the game has begun (player visible, enemies moving, timer started, full health bar).

3. F3-Test
   Main state to pause state

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: The esc key is pressed.

   Output: Pause state

   How test will be performed: In the main state, the tester will press the esc key and determine if the pause screen is subsequently displayed in the GUI.

4. F4-Test
   Pause state back to main state

   Type: Functional, Dynamic, Manual

   Initial State: Pause state

   Input: The esc key is pressed.

   Output: Main state (game resumed)

   How test will be performed: In the pause state, the tester will press the esc key and determine if their game has been subsequently resumed (player at previous position, timer resumes at previous value, health value hasn't changed, enemies continue their previous path of travel).

5. F5-Test
   Pause state to start state

   Type: Functional, Dynamic, Manual

   Initial State: Pause state

   Input: The enter key is pressed.

   Output: Start state

   How test will be performed: In the pause state, the tester will press the enter key and determine if the initial start screen is subsequently displayed in the GUI.

6. F6-Test
   Main state to lose state

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: Health value equal to zero before the timer ends.

   Output: Lose state

   How test will be performed: The tester will allow enemies to surpass the defensive line until the health value becomes zero and then determine if the lose screen is displayed in the GUI.

7. F7-Test
   Win state to start state

Type: Functional, Dynamic, Manual

Initial State: Win state

Input: The space-bar key is pressed.

Output: Start state

How test will be performed: In the win state, the tester will press the space-bar key and determine if the initial start screen is subsequently displayed in the GUI.

8. F8-Test
   Lose state to start state

   Type: Functional, Dynamic, Manual

   Initial State: Lose state

   Input: The space-bar key is pressed.

   Output: Start state

   How test will be performed: In the lose state, the tester will press the space-bar key and determine if the initial start screen is subsequently displayed in the GUI.

3.1.2 Character Movement and Controls

1. F9-Test
   Player can move up

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: The 'W' key is pressed.

   Output: The player moves up in the GUI.

   How test will be performed: In the main state, the tester will press the 'W' key and determine if the player moves upwards in the GUI.

2. F10-Test
   Player can move down

   Type: Functional, Dynamic, Manual

   Initial State: Main state

Input: The 'S' key is pressed.

Output: The player moves down in the GUI.

How test will be performed: In the main state, the tester will press the 'S' key and determine if the player moves downwards in the GUI.

3. F11-Test
   Player can move left

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: The 'A' key is pressed.

   Output: The player moves to the left in the GUI.

   How test will be performed: In the main state, the tester will press the 'A' key and determine if the player moves to the left in the GUI.

4. F12-Test
   Player can move right

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: The 'R' key is pressed.

   Output: The player moves to the right in the GUI.

   How test will be performed: In the main state, the tester will press the 'R' key and determine if the player moves to the right in the GUI.

5. F13-Test
   Player can aim and shoot

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: Mouse click

   Output: A projectile moves in the GUI with the correct input trajectory.

   How test will be performed: In the main state, the tester will aim with the mouse and click to shoot. The tester will determine if a projectile is subsequently displayed in the GUI and moves in the correct trajectory.

6. F14-Test
   Enemies are moving towards defensive line

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: N/A

   Output: Enemies are moving towards the defensive line in the GUI.

   How test will be performed: In the main state, the tester will determine if enemies are moving towards the defensive line in the GUI.

3.1.3 Game Logic

1. F15-Test
   Player cannot move out of bounds

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: The player is moved to the boundaries of the GUI (via 'W','A','S','D' key presses).

   Output: The player remains visible in the GUI.

   How test will be performed: The tester will move the player to the left, right, top and bottom boundaries of the GUI. The tester will verify whether the player remains within display in the GUI at these boundaries.

2. F16-Test
   Enemy behaviour at defensive line

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: An enemy reaches the defensive line.

   Output: The enemy disappears from the GUI and the health value decreases.

   How test will be performed: In the main state, the tester will allow an enemy to reach the defensive line. They will then determine if the enemy disappears from the GUI when it reaches the defensive line and that the health bar decreases accordingly.

3. F17-Test
   Enemies are killed when hit by projectile

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: An enemy and projectile collide.

   Output: The enemy disappears from the GUI.

   How test will be performed: In the main state, the tester will shoot an enemy with a projectile. The tester will determine whether the enemy disappears upon collision.

4. F18-Test
   Next level starts after winning one level

   Type: Functional, Dynamic, Manual

   Initial State: Main state

   Input: For one level, the health value does not reach zero before the timer ends.

   Output: The next level is started.

   How test will be performed: For one level, the tester must maintain the health value above zero until the timer ends (fend off enemies until the timer ends) and then determine if a new level is started afterwards (timer reset, full health bar).

5. F19-Test
   Winning final level

   Type: Functional, Dynamic, Manual

   Initial State: Main state (final level)

   Input: The health value does not reach zero before the timer of the final level ends.

   Output: Win state

   How test will be performed: In the last level, the tester must maintain the health value above zero until the timer ends (fend off enemies until the timer ends) and then determine if the win screen is subsequently displayed in the GUI.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Requirement Testing

1. NF1-Test

   Type: Structural, Static, Manual

   Initial State: Game launched on a web browser and surveys handed out to our testers.

   Output/Result: More than 80% of testers agree that the images used in the game are visually pleasing.

   How test will be performed: Our game and survey will be provided to our testers. After exploring the game, they will fill out a survey and give their feedback on the images of the game. Survey data will be collected and analyzed, with the test succeeding if more than 80% of users agree that the images are pleasing.

3.2.2 Usability Requirement Testing

1. NF2-Test

   Type: Structural, Static, Manual

   Initial State: Game accessed via CAS student server.

   Input/Condition: Testers launch game on their own computer.

   Output/Result: Game launches successfully.

   How test will be performed: Our testers will launch the game on their own computer. This test passes if none of the surveys indicate that the game fails to launch.

2. NF3-Test

   Type: Structural, Static, Manual

   Initial State: Game launched on a web browser and surveys handed out to testers.

   Input/Condition: Testers give their opinion on if the game is simple and easy to play.

Output/Result: More than 80% of testers agree that the game is easy to play.

How test will be performed: Our game and survey will be provided to our testers. After playing the game, they will fill out a survey and give their feedback on the game's usability and playability. Survey data will be collected and analyzed; this test succeeds if more than 80% of testers agree that the game is easy to play.

3.2.3 Performance Testing

1. NF4-Test

   Type: Structural, Dynamic, Manual

   Initial State: Game hosted on CAS student server. Surveys handed out to testers.

   Input/Condition: Tester starts and plays the game.

   Output/Result: None of the tester survey results indicate game lag when tester attempts to move the character or fire a projectile.

   How test will be performed: After the tester has attempted to play the game on their computer, they will fill out the survey and indicate if the game's performance lagged. The test passes if more than 80% of survey responses indicate no lag.

3.2.4 Stress Testing

1. NF5-Test

   Type: Structural, Dynamic, Manual

   Initial State: Game hosted on CAS student server.

   Input/Condition: Assign twenty users to run the game simultaneously.

   Output/Result: The game runs successfully for each user.

   How test will be performed: Twenty users will be asked to launch the game on the game website simultaneously. This test passes if none of the survey results indicate that the host fails to operate with the load of thirty users.

3.3 Traceability Between Test Cases and Requirements

Test Cases	Requirements
NF1-Test	NFR-01
NF2-Test, NF3-Test	NFR-02
NF4-Test	NFR-04
F1-Test	FR-01
F2-Test	FR-02
F3-Test	FR-12
F4-Test	FR-13
F5-Test	FR-14
F6-Test	FR-10
F7-Test	FR-11, FR-13
F8-Test	FR-10
F9-Test	FR-03
F10-Test	FR-03
F11-Test	FR-03
F12-Test	FR-03
F13-Test	FR-05
F14-Test	FR-06
F15-Test	FR-04
F16-Test	FR-07
F17-Test	FR-08
F18-Test	FR-09
F19-Test	FR-11

Table 4: Traceability Between Test Cases and Requirements

4 Tests for Proof of Concept

Tests for the Proof of Concept will focus on displaying all game graphics, main game functions (player and enemy movement, projectile firing, collisions, and health decrements), and switching between game states.

4.1 Graphic Display and Game State

F2-Test
Start state to main state
Type: Functional, Dynamic, Manual
Initial State: Start state
Input: The space-bar key is pressed
Output: In main state
How test will be performed: At the start screen, the tester will press the space-bar key and then determine if the game has begun (player visible, enemies moving, timer started, full health bar).

4.2 Game Functionality

F13-Test
Player can aim and shoot
Type: Functional, Dynamic, Manual
Initial State: Main state
Input: Mouse click
Output: A projectile moves in the GUI with the correct input trajectory.
How test will be performed: In the main state, the tester will aim with the mouse and click to shoot. The tester will determine if a projectile is subsequently displayed in the GUI and moves in the correct trajectory.

F17-Test
Enemies are killed when hit by projectile
Type: Functional, Dynamic, Manual
Initial State: Main state
Input: An enemy and projectile collide.
Output: The enemy disappears from the GUI.
How test will be performed: In the main state, the tester will shoot an enemy with a projectile. The tester will determine whether the enemy disappears upon collision.

F16-Test
Enemy behaviour at defensive line
Type: Functional, Dynamic, Manual
Initial State: Main state

Input: An enemy reaches the defensive line.
Output: The enemy disappears from the GUI and the health value decreases.
How test will be performed: In the main state, the tester will allow an enemy to reach the defensive line. They will then determine if the enemy disappears from the GUI when it reaches the defensive line and that the health bar decreases accordingly.

5 Comparison to Existing Implementation

Our implementation of GardenDefender retains all of the same functionality of the original existing implementation, with some additional functional and non-functional requirements. For instance, we added a user guide and start screen, different levels, pause and resume functions, and split the game up into separate game states to track different game aspects.

6 Unit Testing Plan

All unit testing will be conducted using Mocha.

6.1 Unit testing of internal functions

Most of our tests are manual tests because it's impractical to implement certain unit tests, as many of the functions rely on user input. However, we plan to use unit tests to verify a few parts of our code.

Specifically, we plan to use unit tests to verify that the game switches between game states correctly. We should be able to write tests that call for a state switch and assert that the game state has changed by comparing the current state with the expected state. Additionally, we will test that all of the game assets load in our load state, so that even though a manual test will be required to ascertain that the images have appeared within the GUI, we will know that they have at least loaded and should be present within the GUI.

6.2 Unit testing of output files

7 Appendix

This is where you can place additional information.

7.1 Usability Survey Questions?

To test the usability of our project, a survey will be administered to 20 people to garner feedback on some of the project's non-functional requirements. The survey will be structured so that the individuals surveyed will respond with "disagree, neutral, or agree" to the survey statements, which are as follows:

1. The game loads and displays the start screen in your browser of choice.
2. The start screen of the game loaded quickly (in less than three seconds).
3. The rules of the game are easy to understand.
4. The aesthetics (imagery) of the game is appealing.
5. The aesthetics (imagery) and text used in the game are not offensive.
6. The difficulty level of the game is appropriate for most players age 7 and older.
7. The game runs without delay - it neither lags nor loads slowly - in your browser of choice.
8. The game did not freeze or stop responding while running.
9. Playing the game was, overall, an enjoyable experience.
10. The game is enjoyable enough to replay.
11. Playing the game was not, in any way, distressing.