SE 3XA3: Test Plan
Return of the Bomberman

Team #18, REM
Miles Jackson jacksa7
Eitan Yehuda yehudae
Ridhwan Chowdhury chowdr11

March 5, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

This document will give an overview of the testing that will be done throughout the implementation of Return of the Bomberman.

1 General Information

1.1 Purpose

The purpose of this document is to describe a plan which verifies that the product fulfills all the requirements through testing.

1.2 Scope

This document describes all the tests that will be performed to validate the functional requirements, non-functional requirements, proof of concept and unit tests. Additionally, this document outlines the associated software used to complete these tests.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
POC	Proof of Concept
SRS	Software Requirement Specification

Table 3: **Table of Definitions**

Term	**Definition**
Node.js	Open-source, back-end JavaScript runtime environment
Mocha	JavaScript test framework running on Node.js
Cells	Square spaces on gameboard
Board	Gameboard made up of NxM cells including hard and soft walls

1.4 Overview of Document

This document will provide a walk-through of the test plan for Return of the Bomberman. This includes a plan for how the testing will be performed, a system test description that outlines the tests for functional and non-functional requirements, and a test plan for the POC and unit tests.

2 Plan

2.1 Software Description

This software in an extension of a bomber-man web browser game to be implemented as an online multiplayer web game written in Javascript using Node.js.

2.2 Test Team

Test team consists of Miles Jackson, Eitan Yehuda and Ridhwan Chowdhury.

2.3 Automated Testing Approach

Much of our project relies on user interaction, for this reason the majority of the automated testing will be made for the base functionalities of our game. A test suite will be created for the functionalities such as input commands and collisions between objects. All test cases passing will ensure the base functionality of our game responds and functions according to the requirements.

2.4 Testing Tools

The testing for this project will be done using Mocha which is a JavaScript test framework for Node.js programs, featuring browser support.

2.5 Testing Schedule

See Gantt Chart at the following url ... **Gantt Chart**

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User Input

1. **test-FR1**: Test Up Movement

   Type: Functional, Dynamic, Automated.

   Initial State: Character is stationary.

   Input: The W key will be pressed and released on user keyboard.

   Output: Character moves up one space on the board.

   How test will be performed: An automated test will mimic a W key press and check the character's position on the board before and after ensuring the character moved one cell upwards on the board.

2. **test-FR2**: Test Down Movement

   Type: Functional, Dynamic, Automated.

   Initial State: Character is stationary.

   Input: The S key will be pressed and released on user keyboard.

   Output: Character moves down one space on the board.

   How test will be performed: An automated test will mimic a S key press and check the character's position on the board before and after ensuring the character moved one cell downwards on the board.

3. **test-FR3**: Test Left Movement

   Type: Functional, Dynamic, Automated.

   Initial State: Character is stationary.

   Input: The A key will be pressed and released on user keyboard.

   Output: Character moves up one space on the board.

   How test will be performed: An automated test will mimic a A key press and check the character's position on the board before and after ensuring the character moved one cell to the left on the board.

4. **test-FR4**: Test Right Movement

   Type: Functional, Dynamic, Automated.

   Initial State: Character is stationary.

   Input: The D key will be pressed and released on user keyboard.

   Output: Character moves right one space on the board.

   How test will be performed: An automated test will mimic a D key press and check the character's position on the board before and after ensuring the character moved one cell to the right on the board.

5. **test-FR5**: Test Place Bomb

   Type: Functional, Dynamic, Automated.

   Initial State: Character is stationary.

   Input: The Space-bar will be pressed and released on user keyboard.

   Output: A bomb is dropped on the same cell as the Character.

   How test will be performed: An automated test will mimic a Space-bar key press and check the new bomb's position on the board and character's position on the board ensuring the bomb and character are on the same cell.

3.1.2 Game Mechanics

1. **test-FR6**: Test Power-ups

   Type: Functional, Dynamic, Manual.

   Initial State: Character has no power-up or a different power-up.

   Input: The character touches a new power-up.

   Output: Character now has new power-up.

   How test will be performed: A tester will visually observe that if the character touches a new power-up, they get that power-up.

2. **test-FR7**: Test Bomb Detonation

   Type: Functional, Dynamic, Automated.

   Initial State: Bomb is not present on board.

   Input: The character places a bomb.

   Output: Bomb explodes after 3 seconds.

   How test will be performed: The place bomb function will be executed on a character, after a 3 second delay the cell the bomb was placed on will be checked to ensure it no longer has a bomb, but an explosion will be in the cell instead.

3. **test-FR8**: Test Bomb Placing Limit

   Type: Functional, Dynamic, Automated.

   Initial State: Character is present on board.

   Input: The character tries to place a bomb, move one cell and place another.

   Output: Bomb is only placed on initial cell.

   How test will be performed: A bomb will be placed on a cell by the character who will then move one cell and try to place a second bomb while the first is still active. The second cell will then be checked to ensure that no bomb was place on it.

4. **test-FR9**: Test Bomb Radius

   Type: Functional, Dynamic, Automated.

   Initial State: Character is stationary with a soft wall 2 spaced away in the upward direction and 3 away moving to the right.

   Input: The character places a bomb.

   Output: Bomb explosion spans 2 cells in vertical and horizontal direction which would only destroy the wall in the upward direction while the one to the right remains on the board.

   How test will be performed: A bomb will be placed on a cell by the character, after 3 seconds it will ensure the soft wall that was in the

upward direction was destroyed by the explosion, while the explosion did not reach the soft wall on the right.

5. **test-FR10**: Test Soft Wall Destruction

   Type: Functional, Dynamic, Automated.

   Initial State: Character is standing within bomb radius (vertical or horizontal) of a soft wall .

   Input: The character places a bomb.

   Output: soft wall gets destroyed.

   How test will be performed: A bomb will be placed on a cell by the character. After 3 seconds when the bomb explodes the soft wall within the bomb radius will be checked to ensure it is destroyed and is removed from the board.

6. **test-FR11**: Test Hard Wall Indestructibility

   Type: Functional, Dynamic, Automated.

   Initial State: Character is standing within bomb radius (vertical or horizontal) of a hard wall .

   Input: The character places a bomb.

   Output: hard wall doesn't gets destroyed.

   How test will be performed: A bomb will be placed on a cell by the character. After 3 seconds when the bomb explodes the hard wall within the bomb radius will be checked to ensure it is not destroyed and is remains on the board.

7. **test-FR12**: Test Movement Through Walls

   Type: Functional, Dynamic, Automated.

   Initial State: Character is standing next to a hard and soft wall in different directions.

   Input: The character attempts to move in direction of a wall.

   Output: State stays unchanged, character doesn't move.

How test will be performed: The character will be placed next to a hard and soft wall. An input command will be sent to move toward the hard wall, the character position should then be checked to ensure it is the same. A command will then be sent in the direction of the soft was with the same check after to ensure no movement was made.

8. **test-FR13**: Test Player Death

   Type: Functional, Dynamic, Automated.

   Initial State: Character is standing within blast radius (vertical or horizontal) of a placed bomb.

   Input: Bomb explodes and explosion hits character.

   Output: Character dies.

   How test will be performed: A Character will be on the board one cell adjacent to a bomb, after 3 seconds it will ensure the character died and is no longer present on the board.

9. **test-FR14**: Test Win Condition

   Type: Functional, Dynamic, Automated.

   Initial State: Two players on board.

   Input: a bomb explodes and kills one player leaving only one player standing.

   Output: Remaining player wins the game.

   How test will be performed: A Character will be on the board one cell adjacent to a bomb while the other is placed at a safe distance. After 3 seconds it will ensure the character died and is no longer present on the board and that the game over condition is met.

3.1.3 Connectivity

1. **test-FR15**: Test Host Connectivity

   Type: Functional, Dynamic, Manual.

   Initial State: Hosted game in progress, player is host.

Input: Host player leaves game lobby.

Output: Other players get kicked out of game lobby and game ends.

How test will be performed: A tester will host a game lobby with multiple instances (players) in that lobby. Tester will ensure that when the host player leaves, lobby closes and game ends for all players.

2. **test-FR16**: Test Guest Connectivity

Type: Functional, Dynamic, Manual.

Initial State: Hosted game in progress, player is guest.

Input: Guest player leaves game lobby.

Output: Other players can keep playing and game lobby still running.

How test will be performed: A tester will host a game lobby with multiple instances (players) in that lobby. Tester will ensure that when the guest player leaves, lobby stays open and game continues for the rest of the players.

3.2 Tests for Nonfunctional Requirements

3.2.1 Performance Requirements

1. **test-PR1**: Input Response

Type: Dynamic, Manual

Initial State: The game is accessed on the internet and a game is started.

Input/Condition: The tester will perform all available movements in the game as well as dropping bombs.

Output/Result: The tester will ensure the character moves according to the given input within 0.2 seconds.

How test will be performed: The tester will play the game giving all available inputs making sure the game responds correctly within the given time frame.

2. **test-PR2**: Refresh Rate

   Type: Dynamic, Manual

   Initial State: The game is accessed by 2 testers on the internet and a game is started.

   Input/Condition: One tester will perform as many movements as possible withing a 3 second time frame.

   Output/Result: The second tester should see at least 15 movements proving the game will refresh at least 5 times per second while playing multiplayer over the web.

   How test will be performed: One tester will perform as many movements as possible over 3 seconds as the other tester counts the movements to ensure at least 15 are made.

3. **test-PR3**: Multiplayer Synchronization

   Type: Dynamic, Manual

   Initial State: The game is accessed by 2 testers on the internet and a game is started.

   Input/Condition: One tester will perform movements at incremental times known to both players.

   Output/Result: The second tester should see the movements made from the first tester within 0.4 seconds.

   How test will be performed: Two testers will be on a call, synchronizing when the first tester will move. The seconds tester will time it until he sees the movement come through on his screen ensuring it is within 0.4 seconds.

3.2.2 Operational and Environmental Requirements

1. **test-OE1-1**: Windows Compatibility

   Type: Dynamic, Manual

   Initial State: The tester will be using a Windows based computer.

Input/Condition: The tester will launch the game on Safari, Google Chrome, and Microsoft Edge and ensure all movement and input commands respond accordingly.

Output/Result: The tester should see no difference between any of the browsers with all functioning according the the requirements.

How test will be performed: The tester will open the game using one of the browsers and ensure base game functionality responds correctly then will try again on another browser until all have been tested.

2. **test-OE1-2**: MacOS Compatibility

   Type: Dynamic, Manual

   Initial State: The tester will be using a Mac-OS based computer.

   Input/Condition: Same as OE1-1

   Output/Result: Same as OE1-1

   How test will be performed: Same as OE1-1 to ensure the game also functions on Mac computers.

3. **test-OE1-3**: Linux Compatibility

   Type: Dynamic, Manual

   Initial State: The tester will be using a Linux based computer.

   Input/Condition: Same as OE1-1

   Output/Result: Same as OE1-1

   How test will be performed: Same as OE1-1 to ensure the game also functions on Linux based computers.

3.3 Traceability Between Test Cases and Requirements

All test case id's were given to match the id of the SRS they are testing. Example, Function Requirement FR2 is tested in test case test-FR2.

4 Tests for Proof of Concept

4.1 Multiplayer Lobbies

1. **test-POC1**: Multiplayer

   Type: Functional, Dynamic, Manual

   Initial State: 2 testers connect to an online version of the game.

   Input: Each tester will take turns making movements in the game.

   Output: The testers will ensure the online lobbies are functioning correctly and movements can be seen by both players.

   How test will be performed: 2 testers connect to an online lobby, tester 1 will then make a few movement and will confirm tester 2 can see the movements, then it will be repeated with the other tester moving around in the game.

The rest of the testing for our POC has already been covered in the Functional and Non-Functional Requirements Testing Sections. Included in our POC was getting functional multiplayer lobbies as well as browser support. The functional multiplayer lobbies are tested in cases POC1 as well as PR1, PR2, and PR3 with the functional requirements for the base movements. Browser support testing is covered in the 3 test cases OE1-1, OE1-2, and OE1-3 which tests both browser support along with OS compatibility.

5 Comparison to Existing Implementation

The original implementation of the game was on offline single player game with no real objective. All that was available was to move around the board and place bombs to destroy soft walls with the player unable to be killed. Our implementation will upgrade the game to be online including multiple players adding a real objective to the game to destroy all other players and be the last one alive. It will also add additional functionalities to the game such as power-ups to add to the complexity of the game so it is more enjoyable to players.

6 Unit Testing Plan

6.1 Unit testing of internal functions

Mocha will be used for automated unit testing in this project. There will be automated unit tests for core game functionality such as movement of characters, bomb placement, bomb detonation, wall collisions, etc. These tests will ensure that these individual units are functioning as expected making further modifications easier. These automated tests also free up time for testers to test more complicated functionality like host and guest connection to lobby.

6.2 Unit testing of output files

Not applicable as our program include no output files.

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

No SYMBOLIC_CONSTANTS were used in this document.

7.2 Usability Survey Questions?

The following are some questions the users will be asked to ensure usability requirements of the product are met.

1. Are the keyboard controls of the game self explanatory?
2. Did the language in the game makes sense?
3. How enjoyable was the game? Rate 1-10
4. Did you experience any bugs during game-play? If so what were they?