# Test Plan for Bomber Clone Application

## Software Engineering 3XA3: Project

Group 12 - The A Team
Gabriel Lopez De Leon, 1310514
Ren-David Dimen, 1222679
Jay Nguyen, 1327828

23 October 2015

# Table of Contents

# 1 Revision History

| Revision # | Revision Date | Description of Change | Author |
|---|---|---|---|
| 8 | Oct 23 2015 | Added Section 17 | Ren-David Dimen |
| 7 | Oct 23 2015 | Added Section 16 and 18 | Gabriel Lopez de Leon |
| 6 | Oct 23 2015 | Added Section 5 | Ren-David Dimen |
| 5 | Oct 23 2015 | Added Section 7-15 | Gabriel Lopez de Leon |
| 4 | Oct 22 2015 | Update Section 3 | Gabriel Lopez de Leon |
| 3 | Oct 22 2015 | Added Sections 4 and 6 | Gabriel Lopez de Leon |
| 2 | Oct 21 2015 | Added Sections 1-3 | Gabriel Lopez de Leon |
| 1 | Oct 21 2015 | Created Test Plan Document | Gabriel Lopez de Leon |

# 2 General Information

The following section will provide an overview of the Bomber Clone test plan. This section explains the main purpose of the document, the test plan identifier, any key terms and acronyms/abbreviations, and any references.

## 2.1 Introduction

This document is the main test plan for the Bomber Clone Application which is a recreation of legacy code for a Bomberman game. This plan will address possible software risks and the features of the game that are to be tested. The main targets for testing include the game states and player states while ensuring there are minimal to no bugs when running the program.

This project will be tested through both unit testing and manual tests of how well the game interacts with user inputs. The details on the two levels of testing will be adressed in further depth in the later sections of this document.

## 2.2 Test Plan Identifier

Bomber Clone Master Test Plan: **Version 0**

## 2.3 Key Terms, Acronyms, and Abbreviations

| Symbol | Description |
|---|---|
| QA | Quality Assurance |
| SRS | Software Requirements Specification |
| PoC | Proof of Concept Plan |
| GUI | Graphic User Interface |
| UI | User Interface |
| OS | Operating System |
| IDE | Integrated Development Environment |
| TA | Teaching Assistant |

## 2.4 References

This document references the IEEE Test Plan Outline, our team's SRS, and PoC which can all be found online or on gitlab.

# 3 Test Items

The following is a list of key aspects of the program to be tested:

A. Game State: which checks if the game has started, if a player has won, and if the board is setup correctly when the game begins.

B. Player State: which checks if the player has spawned at the start of the game, if the player is still alive or dead, and if the player gains a power-up.

Note: the above test items are meant to be tested using JUnit automated testing. The following are manual tests to test aspects of the game which cannot be checked by automated testing.

C. Game Functionality: testing if the user interfce works properly and how well the game responds to user inputs.

## 3.1 Functional Requirements

**Test Type**: Unit Testing (Automated)
**Summary**: The system shall keep track of the state of the arena.
**Test Factors**: Correctness, reliability and continuity of process.
**Rationale**: To update the game as it progresses.

**Test Type**: Unit Testing (Automated)
**Summary**: The game shall have a timer which tracks how long before the game ends.
**Test Factors**: Reliability and continuity of process.
**Rationale**: To limit the time for each game.

**Test Type**: Black Box Test (Manual)
**Summary**: The arena blocks must spawn at random.
**Test Factors**: Correctness and reliability.

**Rationale**: To add diversity between games, as well as change up the destructible/indestructible blocks.

**Test Type**: White Box Test (Manual)
**Summary**: The game shall be able to take multiple inputs.
**Test Factors**: Performance, correctness, useability and reliability.
**Rationale**: To allow for multiple players to use one keyboard.

### 3.2 Non-Functional Requirements

The main non-functional requirements that are to be tested include correctness, reliability, and performance. The game should be able to run properly and execute its functionality correctly and consistently. The performance of the program overall is also very important and needs to be taken into account, especially with regards to how well the game accepts user inputs.

## 4 Plan

The following section provides a description of the software being tested, the team that will perform the testing, the milestones for the testing phase, and the budget allocated to the testing.

### 4.1 Software Description

The software being tested is a re-implementation of legacy java code which simulates bomberman the game. The game is played with one or two players with the possible addition of computer players which will be implemented at a later time. A time limit will be added to the game to limit the length of each match. Power-ups will be an additional feature to be added later on, outside the project timeline for this course.

### 4.2 Test Team

The team that will execute the test cases, write and review the test plan/report consists of : Gabriel Lopez de Leon, Ren-David Dimen, Jay Nguyen.

## 4.3 Milestones

### 4.3.1 Location

The location where the testing will be performed is Hamilton, Ontario. The institution that will be performing the testing is The A Team, group 12 in McMaster University's Software Engineering 3XA3 Lab Section L01.

### 4.3.2 Dates and Deadlines

Test Case:
The creation of the test cases for both system testing and unit testing have begun already as of October 21, 2015.

Test Case Completion:
Implementing code for automated testing such as JUnit is scheduled to begin immediately and to continue throughout the software development process. The test cases should be completed by the Test Report due on November 27, 2015.

Test Report Revision 0:
The Test Report Revision 0 is to be written by November 27, 2015.

## 4.4 Budget

There is no budget for the testing of the system, the only constraint for testing would be in regards to the time it would take to implement the various tests.

# 5 Software Risk Issues

The game is designed to run on the user's personal computers and as such may present some risks when running on the device or within the code itself. In some instances the following software issues may arise and should be tested for.

1. The program may contain an infinite loop that may deplete the capabilities of the device it is running on. This may cause the game

to crash or in a more serious situation, cause the computer to freeze.

2. The game is built to have many sections of the program to be updating and running at once. This may create a case of deadlock. If this were to happen, the game may stop running.

3. Both the user and the game's memory will be updating the board displayed. This presents a possibility of interference between what the user and what the memory is sending to the board resuting in an inaccurate output.

4. Some components in the game's programming will be asynchronous. As such, the game may take some time to update sections of the code causing a delay in the updating of the board state or it may interfere with other data stored within the game's memory.

5. The game is written so that most modern computers may run it however, there may exist some devices that may not meet the software requirments to run the sysyem.

6. Several packages will be used in the implementation of the program. All packages should be executable when running the program.

# 6 Features To Be Tested

The following is a list of the areas to be focused on during testing of the applicaiton:

A. Addition of a game timer.

B. Redesigned/improved implementation of the board and possible game states.

C. Redesigned/improved inputs, and controls in game. (compared to legacy code)

# 7 Features Not To Be Tested

The following is a list of areas that will not be specifically addressed. All testing in these areas will be indirect as a result of other testing efforts:

A. Performance on Various OS

This non-functional requirement is not one of the main priorities during the testing phase and is not a key aspect that needs to be implemented. Though it is ideal if the program is able to run on various OS, it is not the main focus of the project.

B. GUI and their Functionality

The GUI such as menus, windows, and buttons are assumed to be implemented properly and will be tested indirectly while testing the overall program.

# 8 Approach

## 8.1 Test Tools

The main test tool used during the testing period will be JUnit which is built into the Eclipse IDE for Java programming.

## 8.2 Meetings

Meetings to work on the assignment will be mainly during lab periods each week, any further meetings to work collectively as a group will take place when members see it fit. However, most of the files needed for the project are being shared between group members through GitLab online, so working together on the project does not always require members to be with each other in person.

## 8.3 Testing Levels

### 8.3.1 Types of Tests

There are various types of testing which will be used throughout this testing process. The various types of tests include, white box, black box, unit, manual and automated testing. White box testing will be used mainly to test functionality of complex methods and methods which relate to gameplay, ones that cannot be tested using automated testing. Block box testing will be used for simpler functions to see if they work or not, disregarding how it is implemented. Most of the white box and black box testing will be done manually, and JUnit testing will be automated through the Eclipse IDE.

### 8.3.2 Game State Testing

The testing for our group's game project will consist of the various types of tests above with a main focus on automated testing. The test team, which consists of all the members of group 12, will all test parts of the code throughout its implementation and will be revised/approved of by the other members of the team. Proof of the tests will be seen through documentation and through JUnit files which will also be uploaded into the GitLab repository. Throughtout the testing process, the test team will be relying on JUnit as the main test tool for unit testing.

Our group will be using unit testing to test the various possible game states which occur as the program runs. The rationale behind using unit testing is because we feel that using automated testing for a key aspect of the game will be more efficient and allows for consistent checks to see if the reliability and continuity of process requirements are still being met. Furthermore, using unit testing provides instant feedback when the JUnit file is run. The user is immediately notified by a green light if the expected output is still being outputted after any changes while as a red light would mean that the functionality that the specific unit test is checking is no longer producing the correct output. Consistent testing of the state of the game is important as it is the backbone of the application, the game relies on this to check if a match has begun, how the arena changes and if a player has won.

During the proof of concept demonstration, the game state will be tested by presenting a working implementation of the game which should show how

the requirements are addressed and how the unit tests result in a positive outcome (green light for JUnit tests). The inputs for the unit tests will be random placements of bombs in the arena with an expected output of them affecting the board. Another test will take an input of a bomb placement to eliminate the enemy player where the expected output should be a player win. These tests will be performed by the whole test team and will be demonstrated to the TA.

### 8.3.3 Player State Testing

Player state will also be tested using unit testing as it is another very important part of the program which needs to be consistently checked to ensure it works correctly and with its intended functionality. The state of the player must be tested with JUnit as immediate feedback is once again necessary. The state of the player can change at anytime during a match, so the system must keep checking for the key states such as if the player has spawned correctly at the start of the game, if the player has collected a power-up, and if the player is alive or dead. This testing can be done by taking in an input of the game starting with expected output as the player appearing on the arena. Another input would be placing a bomb next to a player to see if the player's state changes upon the bomb exploding.

### 8.3.4 General Functionality

Our group also plans on testing the overall functionality of the game manually by running and playing the game to ensure all the features are present and everything works as intended. A description of how to test the program manually will likely be added later on and updated throughout the software development process. Overall, these tests are simple to execute and only take the user input from keyboard listeners and will require the user to check and see if the game did what it is supposed after a certain key is pressed or after a certain interaction occurs.

# 9 Item Pass/Fail Criteria

With respect to the unit testing for the Game and Player State, the test will be passed once the Eclipse IDE give the green light signaling that the expected output matches the actual output for the given unit test. JUnit

code will be what tests to see if it passes or not as it validates the output. As for other testing, mainly referring to the manual tests, these tests pass if the game responds well to user input and translates it to an action in game which it is supposed to do.

# 10 Suspension and Resumption of Testing

Testing will be taking place throughout the software development process and no suspension or delay of testing are expected. The only possible criteria for suspension include the delay time when the program is being coded and is unable to be run. The resumption requirement would simply be to complete the code enough to allow for it to run so that various tests either manual or automated can be completed.

# 11 Test Deliverables

The following is a list of the deliverable related to testing:
A. Test Plan Revision 0
B. Test Report Revision 0
C. Test Plan Revision 1
D. Test Report Revision 1

# 12 Remaining Tasks

| Task | Assigned To | Status |
|---|---|---|
| Complete Test Plan Revision 0 | Test Team | In Progress |
| Create Test Report Revision 0 | Test Team | Not Started |
| Create Test Plan Revision 1 | Test Team | Not Started |
| Create Test Report Revision 1 | Test Team | Not Started |
| Complete Proof of Concept Demo | Group 12 | In Progress |

Note: Test Team and Group 12 consist of the same members but the two names are used to identify the type of the task, whether it relates to testing, programming or the overall project.

# 13 Environmental Needs

Not much is needed to support testing in our project, the main requirement would be to ensure that the JUnit framework is installed for unit testing in the Eclipse IDE. The rest of the testing mainly requires the test team to just run the program and see the results for black box tests and use manual tests for some of the white box testing.

# 14 Staffing and Training Needs

Further training for this project is not necessary, it is within the scope of capabilites of the project team. Each member of the test team have experience with Java programming, the Eclipse IDE and unit testing using JUnit. With regards to staffing, each member of the test team will be required to consistently run tests on the code as it is being implemented.

# 15 Responsibilities

There is no set leader of the project or testing phase, each group member is expected to contribute and work together to complete the tests. All of the work done in the testing period will be split amongst the test team ensuring that all manual and automated tests are completed and properly documented.

# 16 Schedule

The following is a description of the allocated time for different testing activities:

A. Development of master test plan Revision 0: due on October 23, 2015
B. Review of test plan before implementing tests: scheduled for October 24/25, 2015
C. Review of SRS and PoC to ensure the code is implemented correctly: scheduled for October 24/25, 2015
D. Break in the testing process to complete most of code implementation:

scheduled for November 19-23, 2015

E. Unit testing: due by Test Report, on the November 27, 2015

# 17 Planning Risks and Contingencies

A. Limited Working Staff

The staff assigned to complete the project is extremely limited and losing or having a member reassigned would severely hinder the progress of the project.

Should this occur, the Project Supervisor and the Professor will be updated on the situation and updates to the project's deadlines and requirements will be adjusted accordingly.

B. The Scope of the Project is too Big

The desired results of the program as desired by the client may be larger than feasible given the time constraint. As such, it may be difficult to complete the project as required.

To help minimize this risk, tasks will be alloted a specific amount of time to which it shall be completed. If the tasks can not be completed in time, the working team shall review the tasks and see what is essential to the project and what can be removed. This shall help simplify what is needed to be done.

C. Technical Complexity

As the programmers are familiar with the project in question, it may seem like it will be simple to implement. Underestimating the complexity of the task at hand may lead to simple errors in the program.

To combat this problem, the program will be periodically compiled and ran to ensure that the project is up to standard.

# 18 Approvals

| | |
|---|---|
| Project Supervisor - Hediyeh Motaghian (TA) | |
| Professor - Spencer Smith | |