McMaster University

Software Project Management

SFWR ENG 3XA3

Software Test Plan

Authors:
Mohammad Naveed **1332196**
Josh Voskamp **1319352**
Stephan Arulthasan **1308004**

November 4, 2015

Contents

List of Tables

List of Figures

Revision History

Revision Number	Revision Date	Description	Author
0	Oct 21 2015	Created Test Plan Document	Stephan Arulthasan
0	Oct 21 2015	Functional Tests	Everyone
0	Oct 22 2015	Tools and Types of Tests	Josh Voskamp
0	Oct 23 2015	Non Functional Tests	Mohammad Naveed
0	Oct 23 2015	Proof of Concept	Josh Voskamp
0	Oct 23 2015	Project Plan	Mohammad Naveed
0	Oct 23 2015	Method of Testing	Stephan Arulthasan

Table 1: Revision History

1 Introduction

1.1 Document Overview

This document describes the verification and validation process that will be used to test the game 2048. This document will be used as a reference for future testing.

2 Types of Tests

2.1 Structural Testing

Structural testing is also known as white box testing. Structural testing uses an understanding of the internal code of the program to select test cases. The purpose is to test how the program handles extreme cases.

2.2 Functional Testing

Functional testing is also known as black box testing. Functional testing focuses on giving a specific input and verifying that the program returns the expected output. The purpose is to test the functional requirements.

2.3 Unit Testing

Unit tests involves testing individual units of code independently to make sure that inputs give their expected outputs. Unit testing can be done both, manually or automatically.

2.4 Static vs Dynamic Testing

Static testing is a testing method focused on verification. This is done by reading through the code looking for errors rather than executing the program. Dynamic testing is performed in a runtime environment. Dynamic testing monitors the program's interactions with system memory, CPU and overall system performance. Dynamic testing is focused on validation.

2.5 Manual vs Automated Testing

Manual testing is performed by people, where test cases are preformed manually. Manual testing is more flexible in its test cases. Automated testing requires automated tools to perform tests quickly and effectively returning results. Manual testing is inefficient to perform regression testing, which involves constantly executing the same or similar tests.

3 Test Tools

3.1 JUnit

JUnit is an automated unit testing framework for Java. It will be used in the testing of the game 2048.

4 Method of Testing

4.1 Unit Testing

The application is broken down into a Model, View, Controller (MVC) format, in which all the data and objects are placed within the Model class. The visualization of the application is placed within the View class, and the logic to change the states of the application is separated into the Control class. Using this format, we can test the correctness and functionality of the application in the form of JUnit testing. This separates the test cases into smaller ones, and tests every single class with specific cases to ensure overall correctness. The process of creating a new game of 2048 in its current implementation, involves a ”game board” array that contains all the values of the current state. An array contains 16 values, as the game board holds ”tile” objects in a 4x4 format. Testing for a specific case would be achieved by creating a new array that contains a predetermined combination of values, inserting it into the main class, and then compiling and running the game to see how the application responds. For example, testing for a win in 2048 can be done by creating an array where at least two values contain the tile object with a values of 1024. This will be the ”initial state.” The game can then run and be recorded to see what the output is after a specific set of inputs. This entire process can be automated, by creating a script with several different arrays to cover each test case and then recording the output of the ”assertEquals” function.

5 System Tests

5.1 Test Case 1: Starting a new game

Initial State: An existing game is running already.
Input: Escape Key
Output: New game board initiated, with two random tiles (2,4) generated.
Method of Testing: Manual testing

5.2 Test Case 2: Exiting a game.

Initial State: The game board is running already.
Input: User presses the red ”X” that closes the window.
Output: The window closes.
Method of Testing: Manual testing

5.3 Test Case 3: The user must be able to make moves.

Initial State: Game board is running with any set of tiles.
Input: User presses either of the 4 arrow keys.
Output: The tiles move in the specified direction pressed by the user.
Method of Testing: Automated and Manual testing

5.4 Test Case 4: The user must be able to win the game.

Initial State: The game board must be running with at least 2 1024 tiles beside each other.
Input: The user presses one of the arrow keys that would join the two tiles together.
Output: The two tiles join to create the 2048 tile and the game indicates the user has won. No more moves can be made.
Method of Testing: Automated and Manual testing

5.5 Test Case 5: The user must be able to lose the game.

Initial State: The game board must be running in a state where the game board is full of tiles (16) with one more move available, which results in the next state having no more valid moves available, which means no two tiles that are the same are adjacent.
Input: User completes their only valid move
Output: The game ends and displays ”You Lose” message on screen.
Method of Testing: Automated and Manual testing

5.6 Test Case 6: The user must not be able to move off the edge of the board.

Initial State: The game board must be running in a state where there is only one tile in the bottom left corner.
Input: User presses the left key.
Output: Does not allow the tile to move left.
Method of Testing: Automated and Manual testing

5.7 Testing against Existing Implementation.

This version of the game will be tested against the existing implementation by running the same sequence of moves on identical initial game boards, and then compare the resulting game board. This will be tested through automated testing.

6 Test Factors

The test factors refer to the testing approach used to test software quality factors, which are the *non-functional requirements* of the system. Some of the test factors and their most common testing techniques are shown in the table below:

Test Factor	Meaning	Test Techniques
Ease of Use	Ability of the user to readily perform a successful task	Compliance, requirements, manual support, unit testing, usability testing
Performance	Systems responsiveness and reaction to standard and non standard workloads	Stress, execution, compliance, unit testing
Reliability	Ability to perform its functions and maintain performance through circumstances	Execution, recovery, requirements, error handling,unit testing
Portability	Ability to compile and run on different platforms	Operations, compliance
Maintainability	Ease at which the product can be maintained	Compliance, unit testing
Coupling	How modules within the system rely on each other	Operations, inter-systems control

Table 2: Test Factors

6.1 Factors to be tested in 2048

6.1.1 Ease of Use

The software should be easy to use and understand. Users of the software should be able to determine what they can and cannot do with ease, and the interface should also be intuitive.

Rationale:
Testing the user interface will give us a good insight as to what the average user might find too complicated or too easy, and this will allow us to make further potential improvements on the user interface. Furthermore, properly testing the interface and ensuring that it is easy to use will enhance the users experience with the product.

Method of Testing:
Testing the ease of use will be done through manual testing. It will be done by getting a set of users that have no background in computer science or any

related fields, and getting feedback from them about the interface and how easy the game was to play.

6.1.2 Performance

The performance of 2048 should be high enough to keep the users engaged in the game by responding quickly to the users actions. Since our game does not require an IP/TCP connection to the internet, performance in this case refers to the response time of the game to both valid and non valid actions made by the users.

Rationale:
Testing the performance of 2048 will give a good idea of the response time of the game to the users actions. This will allow us to make improvements if necessary. Furthermore, testing performance of this version of 2048 could be used as a benchmark for future versions of the game.

Method of Testing:
Testing the performance of 2048 will be done through both automated testing and manual testing. Automated testing will be done through JUnit test cases in which each test case models the permissible user actions. The response time of the game to the test cases will also be recorded. Manual testing will be done through a set of users that play the game and have the response time of each action recorded.

6.1.3 Portability

The software should be portable in that it has the ability to run and compile on different operating systems, as this will allow us to target a much larger audience.

Rationale:
Testing the portability of 2048 will ensure that the game will work on all major platforms such as Mac OSX, Linux, and Windows.

Method of Testing:
Testing the portability of 2048 will be done manually. The game will be

run on all three different operating systems and results will be recorded (i.e. whether the game compiles and runs or not).

7 Proof of Concept Test

7.1 User Interface Scaling

Since there are many screen resolutions available, the user interface must scale to allow the user to easily see the game. This will be tested manually by running the user interface on different screen resolutions.

7.2 Joining of Tiles

The user should be able to join 2 tile of equal value. Ex (2 and 2). This will be tested manually by playing through the game. Joining of tiles will also be tested using the automated Java unit testing framework, JUnit.

Test Case
Initial State: The game board must be running with at least 2 tiles of equal value next to each other.
Input: The user presses the arrow key in the direction that would join the two tiles together.
Output: The tiles join and become a new tile with double the value.

8 Project Plan

This section provides details on the system to be tested, the team members, the milestones and scheduling for the testing process.

8.1 Software Description

The software being tested is the puzzle game 2048. The game takes as input the directional keys on the keyboard, and outputs the corresponding state of the game board after a user action. The goal of the game is to get the 2048 tile by adding smaller tiles that are all multiples of 2. The game ends when there are no valid moves left for the use to make and the board is full.

8.2 Team Members

The team that will execute the test cases consists of:

- Mohammad Naveed
- Josh Voskamp
- Stephan Arulthasan

8.3 Milestones

8.3.1 Location

The location where the testing and validation will take place is in Hamilton, Ontario. At McMaster University.

8.3.2 Dates and Deadlines

Test Cases: The creation of the system test cases and JUnit test cases is scheduled to begin on October 18th, 2015. The deadline for the test cases is October 23rd, 2015. All three team members will responsible for coming up with both the system test cases and the JUnit test cases.
Test Case Implementation: The implementation of the the JUnit automated testing will begin on October 20th, 2015. The deadline for the implementation will be on October 25th, 2015. Josh Voskamp will be responsible for the implementation of the test cases.
Test Report: The test report is scheduled to begin October 20th, 2015 and the deadline will be on November 27th, 2015. The whole team will be responsible for the test report.

9 Project Approval

Professor - Spencer Smith	
TA - Li Peng	