McMaster University

Software Project Management

SFWR ENG 3XA3

Software Test Report

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

November 25, 2015

Contents

List of Tables

List of Figures

Revision History

Revision Number	Revision Date	Description	Author
0	Nov 23	Created Test Report	Josh Voskamp
0	Nov 23	Intro, Scope, Methods	Stephan Arulthasan
0	Nov 25	Non-functional	Mohammad Naveed

Table 1: Revision History

1 Introduction

The purpose of the test report is to specify in detail the test cases implemented for 2048.

2 Scope

This report contains test cases for both the front end and back end of the application. For every component tested we present a short discussion about the test cases chosen, a summary of the results, and address any issues that may have arisen. Types of tests performedinclude unit tests and system tests, all of which are automated.

3 Method of Testing

3.1 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 of type integer in a 4x4 format. Testing for a specific case would be achieved by creating the initial state of the test case, applying the input, and then recording the output. The initial state consists of creating a new instance of the game board and inserting a tile object that is placed at a specific place on the board. The input would be the function that exists within the class which would be applied to the initial state. An 'assertEquals' statement would be written stating the expected output of the test case. For example, testing for a win in 2048 can be done by creating a game board where at least two values contain the tile object with values of 1024. This will be the 'initial state.' Then the left() function can be called to join

the two tiles together. The assertEquals() function can be written to make sure that the 2048 tile is generated at the proper spot its supposed to be at. The script can then be compiled and run to check if the output matches the expected output specified in the assertEquals() statement.

3.2 Manual Testing

4 Testing Results

4.1 Test Case 1: Starting a new game

Initial State: An existing game is running already.
Input: Escape Key
Expected Output: New game board initiated, with two random tiles with a value of 2 or 4 generated.
Output: New game board initiated, with two random tiles with a value of 2 or 4 generated
Method of Testing: Manual testing
Pass/Fail: Pass

4.2 Test Case 2: Exiting a game

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

4.3 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.
Expected Output: The tiles move in the specified direction pressed by the user.
Output: The tiles move in the specified direction pressed by the user.
Method of Testing: Automated and Manual testing
Pass/Fail: Pass

4.4 Test Case 4: The user must be able to win

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.
Expected Output: The two tiles join to create the 2048 tile and the game indicates the user has won. No more moves can be made.
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
Pass/Fail: Pass

4.5 Test Case 5: The user must be able to lose

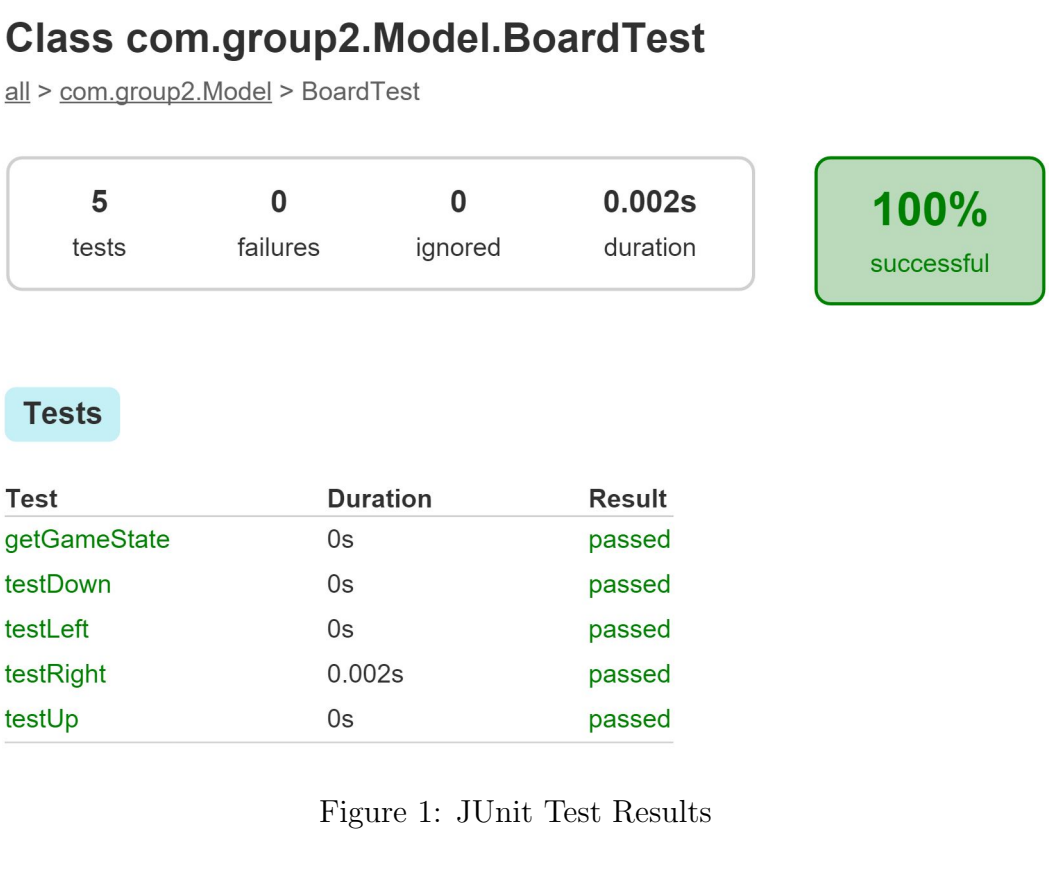
Initial State: The game board must be running in a state where the game board is full of tiles (16) with one more move available, which results in the next state having no more valid moves available, which means no two tiles that are the same are adjacent.
Input: User completes their only valid move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

4.6 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.
Expected Output: Tile does not move off the left side of the board.
Output: Does not allow the tile to move left.
Method of Testing: Automated and Manual testing
Pass/Fail: Pass

4.7 Test Case 7: Testing against Existing Implementation.

Initial State: Original implementation and new implementation, with identical tile locations
Input: User performs the same action on both implementations
Expected Output: Both implementations result in the same board except for the new randomly generated tile
Output: Both implementations result in the same board except for the new randomly generated tile
Method of Testing: Manual testing, compare the resulting board from each implementation of the game
Pass/Fail: Pass



Class com.group2.Model.BoardTest

all > com.group2.Model > BoardTest

5	0	0	0.002s
tests	failures	ignored	duration

100% successful

Tests

Test	Duration	Result
getGameState	0s	passed
testDown	0s	passed
testLeft	0s	passed
testRight	0.002s	passed
testUp	0s	passed

Figure 1: JUnit Test Results

5 Non-Functional Qualities Testing

5.1 Usability

5.1.1 GUI

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.

The feedback from the users will be obtained through a questionnaire in which the questions are based on how easy the GUI and game, was to understand and use. The users will then rate their satisfaction on a scale of 1 to 5, where 5 is being completely satisfied and 1 is being completely unsatisfied. A copy of the questionnaire can be found in the Appendix.