Zop Test Plan

Ratna Emani, Cole Blanchard, Akshay Mantha

December 7, 2015

Revision History

Developer	Date	Change	Revision Number
Akshay Mantha	December 7 2015	Testing Schedule (Akshay)	9
Cole Blanchard	December 7 2015	Performance Testing	8
Ratna Emani	December 7 2015	Testing Schedule (Ratna), Usability Testing	7
Ratna Emani	November 27 2015	Update Tex file	6
Akshay Mantha	November 27 2015	Update Test plan	5
Cole Blanchard	October 23 2015	3.5 Board Update Test, 3.6 Tile Adjacency Check	4
Ratna Emani	October 23 2015	Reliability, Maintainability, Correctness, Ease of Use, 3.4 Raw Game-play Test	3
Akshay Mantha	October 22 2015	3.1(...)Score Are the Same, 3.2(...)Score Are Not the Same, 3.3 Match of Tiles	2
Ratna Emani	October 22 2015	File Development, Formatting	1

Contents

1 Introduction

1.1 Reliability

Reliability takes precedence over efficiency. It often relies on and involves the redundant checking for exceptional conditions within the program. Many developers often show negligence when testing for reliability. There is a lack of defined depth which programmers deem exceptional conditions necessary, often risking reliability in the process. Given an error it is of the utmost importance that the application must respond appropriately. Errors include values passed by users that are out of the input domain range, warnings that direct users for any potential unwanted behaviours is stimulated by the program. These can be controlled my managing the user accessibility to sensitive values that may cause problems to the internal functions of the program. Also, by keeping the users input domain to a minimum, the program has less possibility to run into errors.

1.2 Maintainability

Software maintainability is the foothold of a successful program. Maintainability is great tool shared amongst developers that allow the application to expand and adopt. With an open-source project like this, it is crucial to keep the program well maintained, allowing other developers to implement furthermore. The lack for maintainability is the main cause for the demise of a program. Once this program has been made by our development team, it will be back up as an open-source project. Support and maintenance will be provided by other developers that will use the game to expand or evolve to new measures.

The game has to potential to change in many different ways. From the addition of new controls, to reinventing the game rules, the possibilities can be endless. As a development team, we chose to use modularity to increase maintainability of the program. By making the application more object-oriented it allows other developers to easily understand and apply their changes with minimum effort. With the use of simple modules and strong internal cohesion the program will maximize the performance as well.

1.3 Correctness

Test cases allows the development team and the client to test the final product for correctness. Correctness is a degree to which a programs behaviour matches its requirements, essentially a tool to measure quality and robustness. It is important for the development team to deliver what was promised in the specifications. However, it is crucial for the program to be error free, to keep the user engaged at all times. The lack of free flow in game design or bugs can easily put off players. Therefore it is crucial that the game does not diverging from a specification and updates its rules to maintain correctness.

1.4 Ease of Use

Ease of use defines the extent to which the program can be used to achieve the results required by the user. Setting a low barrier of entry shuns the users away from feeling conformable with the application. This means that the user interface must be simple and minimal in structure. The program must be accurate and fast, in which the users can play the game and get the results they expect, efficiently. One way to make this possible is to minimize the user input to simple (straightforward) instructions. The game itself uses the clicker input from a frame of the game's interface. Furthermore, the game separates its shell buttons from the actual core gaming, to avoid any accidental disruption during the gaming session. This allows the game to be more engaging and satisfying to use.

2 Automation Plans

Automated PyUnit testing will be used to ensure that the internal functions are operating correctly. Each function and objects will be tested by passing the appropriate values from the input domain range including the extremes. Some of the places automated testing can be used for is checking of the score board, checking if the colors of the selected tiles are matching correctly and if the tiles being generated are all within the color range. ~~White box testing or Structural testing will not be automated for this program~~. We would like to use automation to mainly check functionality of the application.

2.1 Unit Testing

We will make use of the 'unittest' (of Python). We do not need drivers, but we will be using stubs for most of the unit tests. Stubs, mostly, include creating instances and objects of the involved (in testing a specific method) methods. As for code coverage metrics, we did not quantify the coverage, but we tested (covered) all important methods from all modules.

3 System Tests

Systems tests will, primarily, utilise the Logic.py module, which in turn utilises the Board.py module (which itself utilises the Tile.py module). This is a way in that the existing implementation will be tested.

3.1 Tile(s) Removed and Value is set to 0

3.1.1 Test type

Automated, unit test

3.1.2 Initial state

The board is initialized, size of 6 by 6; 36 blocks in place.

3.1.3 Input

Tile(s) removed (after matching).

3.1.4 Output

The value of tile(s) removed is 0.

3.1.5 Schedule

This feature will be implemented for the Proof of Concept Demonstration.

3.1.6 Methodology

This test can be done using PyUnit (unit-testing framework) in PyCharm (an IDE). PyUnit should pass the test case.

3.1.7 Test tool

Integration testing can be used as a tool to see how this function communicates with other functions of the system.

3.1.8 Test for

This test validates the ”happy path” scenario of updated score requirement.

3.2 Tile(s) Removed and Value is not set to 0

3.2.1 Test type

Automated, unit test

3.2.2 Initial state

The board is initialized, size of 6 by 6; 36 blocks in place.

3.2.3 Input

Tile(s) removed (after matching).

3.2.4 Output

The value of tile(s) removed is not 0.

3.2.5 Schedule

This feature will be implemented for the Proof of Concept Demonstration.

3.2.6 Methodology

This test can be done using PyUnit (unit-testing framework) in PyCharm (an IDE). PyUnit should pass the test case.

3.2.7 Test tool

Integration testing can be used as a tool to see how this function communicates with other functions of the system.

3.2.8 Test for

This test validates the negative scenario of removing tile (leads to unupdated score) requirement.

3.3 Colour Match of Tiles

3.3.1 Test type

Automated, unit test

3.3.2 Initial state

The board is initialized, size of 6 by 6; 36 tiles in place.

3.3.3 Input

Coordinates of tiles are given to the function.

3.3.4 Output

A Boolean value; true when the selected tiles' colour is the same.

3.3.5 Schedule

This feature will be implemented for the Proof of Concept Demonstration.

3.3.6 Methodology

This test can be done automatically using PyUnit. The tester will give PyUnit the funciton. Then, PyUnit should execute the test and pass it.

3.3.7 Test tool

Unit testing framework can be used as a tool to see how this function works within the scope of the system.

3.3.8 Test for

This test validates the requirement of selecting tiles of same colour to obtain a score.

3.4 Board Update Test

3.4.1 Test type

Automated, Structural test

3.4.2 Initial state

State after removing tiles from the board (some spaces are empty).

3.4.3 Input

6x6 array (empty spaces agged).

3.4.4 Output

6x6 array (no empty spaces remaining, tiles above move down to ll empty spaces).

3.4.5 Schedule

This feature will be implemented for the Proof of Concept Demonstration.

3.4.6 Methodology

Using PyUnit, the test will remove 0-6 blocks in a column. PyUnit should execute and ensure the tiles above move into the correct space in the next state of the board. Then it will ensure there are no more empty spaces on the board.

3.4.7 Test tool

Unit testing can be used to see how the function works within the system.

3.4.8 Test for

The test validates that all empty spaces in the previous state of the board have been lled with the tiles above it in the next state as well as randomly coloured blocks to ll in spaces which do not have tiles above.

3.5 Tile Adjacency Check

3.5.1 Test type

Automated, unit test

3.5.2 Initial state

Board is initialized, 6x6 array.

3.5.3 Input

Coordinates of blocks are given

3.5.4 Output

Boolean value is given; True when blocks are directly next to each other, False otherwise.

3.5.5 Schedule

This feature will be implemented for the Proof of Concept Demonstration.

3.5.6 Methodology

Using PyUnit, the test will ensure that any tiles selected by the user are next to each other, in one direct path (i.e. no overlapping).

3.5.7 Test tool

Unit testing can be used to see how the function works in the system.

3.5.8 Test for

The test validates that tiles being removed are next to each other, in a direct path (horizontal or vertical only).

3.6 Type Exception Check (While Moving Tiles Down)

3.6.1 Test type

Automated, Structural test

3.6.2 Initial state

Board is initialized; 6x6 array.

3.6.3 Input

Column number, instance of boad, and an integer representing the number of empty spaces (in a column).

3.6.4 Output

Assertion should throw a 'TypeError' if either the column number is not an integer, some other object/class is instantiated instead of board, and there is no integer to represent the number of empty spaces (in a column).

3.6.5 Schedule

This feature will be implemented for the Final Presentation.

3.6.6 Methodology

PyUnit will see, if parameters of the moveDown method are of correct type or not.

3.6.7 Test tool

Unit testing can be used to see how robust the function is with respect to parameters.

3.6.8 Test for

The test validates throwing exceptions.

4 Non Functional Testing

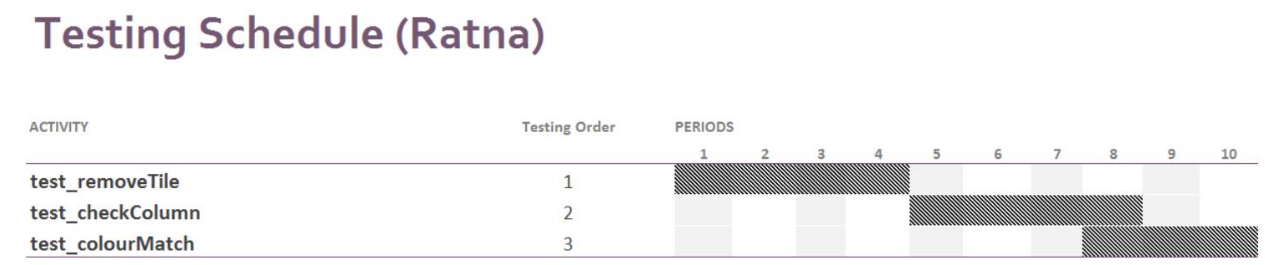
4.1 Usablity Testing

Usability testing is a proficient way of measure real-user experience of any application. Companies that launch an software products usually dedicate a team of users called a focus group to test this. Followed by Alpha, and Beta testing done by select users/consumers. For usablility testing of Zop, we decided to ask few friends to play the game and write their suggestions/concerns as feedback.

4.2 Performance Testing

Performance testing is a good way of seeing how efficient the program it running. It measures the speed of functions and how many times they are called throughout the program. In Zop, performance testing will be measured by how many times functions are called throughout the program. This will accomplish seeing how we can make the program better for the final product.

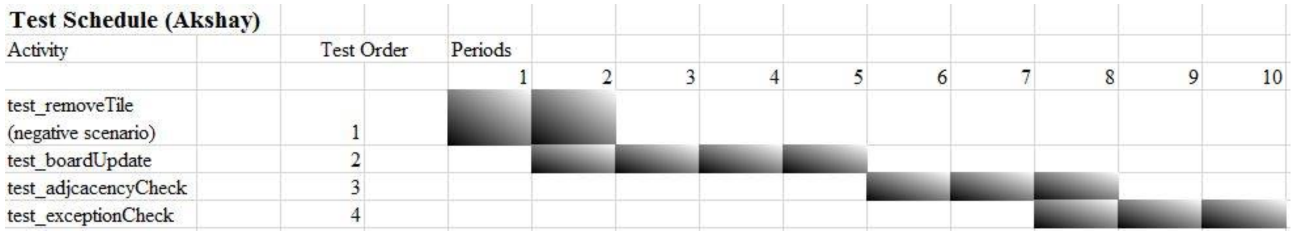
5 Test Scheduling



Testing Schedule (Ratna)

ACTIVITY	Testing Order	PERIODS									
		1	2	3	4	5	6	7	8	9	10
test_removeTile	1	▨	▨	▨	▨						
test_checkColumn	2					▨	▨	▨	▨		
test_colourMatch	3								▨	▨	▨

Table 1: Testing Schedule 1



Test Schedule (Akshay)

Activity	Test Order	Periods									
		1	2	3	4	5	6	7	8	9	10
test_removeTile (negative scenario)	1	▨	▨								
test_boardUpdate	2		▨	▨	▨	▨					
test_adjcacencyCheck	3						▨	▨	▨		
test_exceptionCheck	4								▨	▨	▨

Table 2: Testing Schedule 2