

SE 3XA3: Test Plan
Title of Project

Team 30, VUA
Andy Hameed and hameea1
Usman Irfan and irfanm7
Vaibhav Chadah and chadhav

October 25, 2018

Contents

1	General Information	1
1.1	Purpose	1
1.2	Scope	1
1.3	Acronyms, Abbreviations, and Symbols	1
1.4	Overview of Document	1
2	Plan	1
2.1	Software Description	1
2.2	Test Team	1
2.3	Automated Testing Approach	1
2.4	Testing Tools	1
2.5	Testing Schedule	1
3	System Test Description	2
3.1	Tests for Functional Requirements	2
3.1.1	Area of Testing1	2
3.1.2	Area of Testing2	2
3.2	Tests for Nonfunctional Requirements	2
3.2.1	Look and Feel	2
3.2.2	Usability	3
3.2.3	Performance	3
3.2.4	Operational and Environmental	4
3.2.5	Maintainability and Support Requirements	5
3.2.6	Security and Cultural	5
3.3	Traceability Between Test Cases and Requirements	6
4	Tests for Proof of Concept	6
4.1	Snake Dynamics	6
4.2	Integration and System Testing	7
5	Comparison to Existing Implementation	8
6	Unit Testing Plan	8
6.1	Unit testing of internal functions	8
6.2	Unit testing of output files	8

7 Appendix	9
7.1 Symbolic Parameters	9
7.2 Usability Survey Questions?	9

List of Tables

1 Revision History	ii
2 Table of Abbreviations	1
3 Table of Definitions	1

List of Figures

Table 1: **Revision History**

Date	Version	Notes
10/25/2018	1.0	Andy added section 4.0
Date 2	1.1	Notes

1 General Information

1.1 Purpose

1.2 Scope

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	Definition
Abbreviation1	Definition1
Abbreviation2	Definition2

Table 3: **Table of Definitions**

Term	Definition
Term1	Definition1
Term2	Definition2

1.4 Overview of Document

2 Plan

2.1 Software Description

2.2 Test Team

2.3 Automated Testing Approach

2.4 Testing Tools

2.5 Testing Schedule

See Gantt Chart at the following url ...

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Area of Testing1

Title for Test

1. test-id1

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

2. test-id2

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

3.1.2 Area of Testing2

...

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. test-id1

Type: (Structural, Functional, unit), Dynamic, Manual

Initial State: The game should be installed on the device.

Input/Condition: The game is opened and ran on the device.

Output/Result: The User Interface should open with different buttons, alongside with a playground with a snake in it.

How the test will be performed: The program will manually run on the device and checked by the human eye to see if it meets the criteria.

3.2.2 Usability

1. test-id1

Type: (Structural, Functional, unit), Dynamic, Manual

Initial State: The program will be running for a human nearing 10 years of age or above

Input/Condition: The program will be set on its default settings

Output/Result: The person testing should be able to understand the game and play it. He/she should be able to customize themes and speed of the game.

How the test will be performed: A younger human of nearing age 10 will be asked to operate this game and recorded if he/she is able to operate it successfully or not.

3.2.3 Performance

1. test-id1

Type: (Structural, Functional, unit), Dynamic, Manual

Initial State: The program will be running with the main user interface open.

Input/Condition: The button is pressed.

Output/Result: The response time for button should be less than half a second.

How the test will be performed: It will be performed using human actions. The response would be times to be as precise as possible. Also, it will be taken into consideration that the user doesn't have to wait for a long observable time.

2. test-id1

Type: (Structural, Functional, unit), Dynamic, Manual

Initial State: The snake Game will be running on the device.

Input/Condition: Various speed inputs for the snake.

Output/Result: Different snake speeds according to what the user has decided

How the test will be performed: The game will be played with inputting different speeds. Then, the speed difference will be observed as the game progressed through and taken care that the game goes at the constant speed at each level.

3. test-id1

Type: (Structural, Functional, unit), Dynamic, Manual

Initial State: The snake Game will be running on the device.

Input/Condition: The snake will be moving around and keys will be pressed to change directions.

Output/Result: Snake should change directions promptly.

How the test will be performed: While the game is going on, the buttons will be pressed to change the direction of the snake.

3.2.4 Operational and Environmental

1. test-id1

Type: (Structural, Functional, unit), Dynamic, Manual

Initial State: The program will be moved on a USB.

Input/Condition: The USB will be inserted into any other working computer/ Desktop.

Output/Result: The game should be able to run on it as long as the device is powered and in working state.

How test will be performed: Many different laptops, alongside with desktops, will be used to test. The game will be played on different devices with different specifications to make sure that the game is playable regardless of the specs of the device.

3.2.5 Maintainability and Support Requirements

1. test-id1

Type: (Structural, Functional, unit), Dynamic, Manual

Initial State: The program will be moved to a Windows, Mac OS and Linux operating devices.

Input/Condition: The program will be executed.

Output/Result: The game should run.

How test will be performed: The game will be taken and transferred to the systems operating on different OS's. For this, the target is Windows device, Mac OS device and a Linux Device.

3.2.6 Security and Cultural

1. test-id1

Type: (Structural, Functional, unit), Dynamic, Manual

Initial State: The program will be running.

Input/Condition: All the interfaces running.

Output/Result: No offensive or illegal content on the entire application.

How test will be performed: The application will be executed and each page and option will be approached to make sure there is no offensive or illegal content. Also, there is a Static module to this requirement where all the files (including code) will be looked to make sure about no offensive or illegal content.

3.3 Traceability Between Test Cases and Requirements

4 Tests for Proof of Concept

4.1 Snake Dynamics

Snake Movement and Speed

1. T1D1

Type: Dynamic

Initial State: The snake body - graphically represented by a red square - is initially motionless. It exists somewhere within the frame of the window.

Input: Keyboard Event - user clicks on one of the directions on the keyboard arrow pad.

Output: Snake moves according to the direction chosen. This can logically be represented by the expression `keyboardEvent.direction == snakeMovementDirection`. Note that the variables used are arbitrary and are dependant on Python syntax.

How test will be performed:

- A method will be created under the POC test class where the keyboard event is manually set to the code representing each of the directions on the arrow keypad - up, down, left and right. The direction inserted will be asserted equal to the direction of the moving snake, set by some variable.
- After starting the game, the user will click on each one of the four directions and verify whether or not the snake is moving in the corresponding direction, using the graphical interface created with Pygame.

2. T1D2

Type: Dynamic, Functional testing

Initial State: The snake body - graphically represented by a red square - is initially motionless. It exists somewhere within the frame of the window.

Input: Keyboard Event - user clicks on one of the directions on the keyboard arrow pad.

Output: Snake moves accurately according to the speed set in the snake module. Statically, this can be represented for the vertical movement of the snake by this expression:

$$(snakeFinalPosition - snakeInitPosition) == (speed \times timeElapsed) * vel$$

where *vel* is the distance defined for 1 single step and *speed* is the delay between each step in milliseconds

How test will be performed: A method will be created under the POC test class where the keyboard event is manually set to the code representing each of the directions on the arrow keypad - up, down, left and right. The logical expression above is implemented into an assert statement verifying that the distance moved corresponds to the speed and velocity that were used as well as the time that has elapsed - this can be obtained from the time object in Pygame.

4.2 Integration and System Testing

1. T1D3

Type: Integration, System testing

Initial State: The game menu is loaded onto the window with options for starting the game and quitting the game.

Input: The following sequence of inputs

- (a) User clicks start game
- (b) Keyboard Event - user clicks on one of the directions on the keyboard arrow pad.
- (c) user exits the window by clicking the exit tab on the top right corner of the screen

Output: Snake game runs as intended. The "start game" option leads the user to the game screen where the snake body sits motionless. The user moves the snake body using the keyboard arrow pad, moving in directions that correspond appropriately to the arrows clicked and in the correct sequence. The game window is closed once the user clicks the exit button on the top right corner.

How test will be performed: Several peers will be asked to test the game from start to finish for this integration and system test.

5 Comparison to Existing Implementation

6 Unit Testing Plan

6.1 Unit testing of internal functions

6.2 Unit testing of output files

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for `SYMBOLIC_CONSTANTS`. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions?

This is a section that would be appropriate for some teams.