Test Plan

Alex Guerrero, Keyur Patel and Shafeeq Rabbani

December 8, 2015

Revisions

Table 1: Revisions

Name	Date	Description
Keyur Patel	08/12/2015	Created Revision Table
Keyur Patel	08/12/2015	Added testing for proof of concept demo

Contents

1 Introduction

The test plan is designed to identify the types of tests to perform and helps explain how tests will be performed.

1.1 Test Items

The different items to be tested includes:

- A: The functions and methods of each class of the Model (backend)
- B: The game board against the functional requirements of the product
- C: The graphical interface that implements the Model

2 Software Risk Issues

The pre-existing code is to be redeveloped into modules as per the Model-View-Controller pattern for implementing interfaces. Modularizing the code may prove to be more time consuming than previously anticipated and hence, the code may not be ready in time to meet the deadline. As testing is dependent code, an incomplete code would bring the risk of incomplete or unattempted testing.

The test cases may miss out some scenarios that result in errors in runtime.

3 Features to be Tested

As the first stage of our project redevelopment was to modularize the existing code, every single module in the implementation will be tested.

As per the Model-View-Controller pattern, the model consists of the game environment with all the game rules. The controller is used to input commands into the environment. The view is the GUI of the game made by the graphics available in the PyGame package.

The model portion contains classes Snake, Food, Map and Boundary classes. These classes will have to be tested.

The Controller contains both keyboard inputs and mouse click inputs. This can be tested by giving the controller the appropriate keyboard and mouse inputs and seeing that it responds accordingly.

For the testing of View, the GUI will have to be tested to ensure that it appears appealing and exactly as intended. This will be left for User testing.

4 Features not to be Tested

Functions are to be added to the snake class in the future to allow for the intangibility feature, sacrifice feature and freeze feature. These are not currently being developed and hence are not a part of the current testing.

5 Testing Types

Testing can be broken up into different types, which each have their own role in the testing the product. These test types should be utilized to comprehensively evaluate the quality of the product.

5.1 Structural Testing

Structural testing is also known as white box testing. Structural tests are derived from the program's internal structure. It focuses on the nonfunctional requirements of the product. This type of testing shows errors that occur during the implementation by focusing on abnormal and extreme cases the product could encounter.

5.2 Functional Testing

Functional testing is also known as black box testing. Functional tests are derived from the functional requirements of the program. It focuses less on how the program works and more on the output of the system. These tests are focused on test cases where the product receives expected information.

5.3 Static vs. Dynamic Testing

Static testing simulate the dynamic environment and does not focus on code exectution. This testing involves code walkthroughs and requirements walkthroughs. Static testing is used prevalently in the design stage. In contrast, dynamic testing needs code to be executed.

Dynamic testing involves test cases to be run and checked against expected outcomes. A technique to save time during dynamic testing is to choose representative test cases.

5.4 Manual vs. Automatic Testing

Manual testing is done by people. It involves code walkthroughs and inspection.

Automatic testing can usually be conducted by computers. The tools used to assist with automatic are unit testing tools for the respective programming language. Automatic testing relies on people for testing more qualitative aspects like GUI.

6 Test Factors

6.1 Reliability

- Produces similar results under consistent conditions
- Test Techniques: unit testing, player testing, requirements

6.2 Correctness

- Shows the correct behaviour (expected output) for each input
- Test Techniques: unit testing, player testing, requirements

6.3 Ease of Use

- Ability of a user to naturally operate software
- Test Techniques: player testing, requirements

6.4 Maintainable

- Ease to which the code can be maintained
- Test Techniques: requirements

6.5 Performance

- Responsiveness of system to user input and stability during various workloads
- Test Techniques: stress, player testing

7 Proof of Concept Test

Since the proof of concept is only a partial implementation of the final product, the testing will be fairly informal.

Proof of concept shall be tested for the following:

- The snake object can change directions
- The snake object will grow when it eats food
- The snake object will move in its current direction
- The food object has a random location when instantiated

- The snake object will be flagged as 'dead' when it hits itself or the border of the window (use constants to represent window in backend testing)

8 Testing functional requirements

8.1 Checking status of the game

Test Type

Automatic, dynamic, and functional testing

Test Factors Involved

Initial State

The game board will have the snake and the border coordinates instantiated. The head coordinate of the snake will be adjacent to the border

Inputs

none

Outputs

gameEnded returns the value true

Schedule

Since this test is critical to the simulation, it is scheduled after the Map.py class is coded

Methodology

This test will be conducted automatically by using PyUnit. A game state will be generated and the Snake will be moved into the wall

Test For

This will test the functionality of the gameEnded function

8.2 Game ends if snake collides with itself

Test Type

Manual, functional dynamic test

Test Factors Involved

Correctness

Initial State

Game will be in play state with the snake displayed on screen.

Inputs

Keyboard directions

Outputs

Game over game state

Schedule

Testing will be completed during the initial development stage.

Methodology

This test will be done manually by the developers. The code will be run and tester will start a game. The tester will then ensure that making the snake collide with its own body will end the game.

Test For

The test the functionality of the snake move logic and game state flow.

8.3 The snake grows by one unit when it eats food

Test Type

Manual, structural dynamic test

Test Factors Involved

Correctness

Initial State

Game will be in play state with Food and Snake object instantiated.

Inputs

Keyboard input to get snake to food

Outputs

len(Snake.points) will increase by one

Schedule

Testing will be completed during the initial development stage.

Methodology

This test will be done manually by the developers. The code will be run and tester will start a game. The tester will then ensure that making the snake eat a Food object.

Test For

The test the functionality of the Snake.grow() method.

8.4 Snake must be controlled by the keyboard

Test Type

Manual, functional dynamic test

Test Factors Involved

Correctness

Initial State

Game will be in play state with the snake displayed on screen moving in the default direction.

Inputs

Keyboard buttons: W, A, S, D, and directional buttons (up, down, left, right).

Outputs

Depending on the current direction, the snake will either turn and change direction or do nothing.

Schedule

Testing will be completed during the initial development stage.

Methodology

This test will be done manually by the developers. The code will be run and tester will start a game. The tester will then ensure all keys behave properly for every direction the snake can move.

Test For

The test the functionality of the snake move logic and controller handling keyboard events.

8.5 Calculate a score based on the length of the snake

Test Type

Automatic, dynamic, and functional testing

Test Factors Involved

Correctness

Initial State

The game board will have the snake and the border coordinates instantiated.

Inputs

none

Outputs

game score should be the same as length of the snake

Schedule

This test is critical to the correct functionality of the game, and will be tested once implemented

Methodology

This test will be conducted automatically by using PyUnit. A game state will be generated and the current score will be checked against the current length of the snake

Test For

This will test the functionality ofthe scoring system for the project.

9 Future Testing

9.1 Test Cases for Snake.py

Table 2: Snake.py

Method	Input	Expected Outcome
constructor	none	first 20 points of the snake are generated
changeDir	dir=-1	if current direction is 2 or -2, it will be updated to -1

9.2 Test Cases for Map.py

9.3 Test Cases for Food.py

9.4 Testing for GUI