Test Report

Alex Guerrero, Keyur Patel and Shafeeq Rabbani

November 28, 2015

1 Revisions

Table 1: Revisions

Name	Date	Description
Keyur Patel	27/11/2015	Created Test Report latex file
¡¡¡¡¡¡¡ HEAD Keyur Patel	27/11/2015	Added table template for unit testing AND info
======= Alex Guerrero	27/11/2015	Edited Structural Testing

2 Structural (White Box) Testing

2.1 Unit Tests for Food

Table 2: Revisions

Test Case	Initial State	Expected Output	Output
testRandomPos.1	foodA and foodB randomly placed	positions compared and not equal	pass
testRandomPos.2	foodC randomly placed	——	pass
testRandomPos.3	foodD randomly placed	——	pass
¿¿¿¿¿¿¿ 5999134f4920b5ddc8bf8bfe8614070f6cb1bf9b			

Contents

3 Features that were Tested

- 1:The functional requirements of the product
- 2:The classes and methods of the product (Model)
- 3:The GUI of the product

4 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.

4.1 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.

4.2 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.

4.3 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.

4.4 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.

5 Automated Unit Testing

5.1 Testing for Snake.py

```
>>>
test_changeDirTests (__main__.TestSnakePy) ... ok
test_constructorTests (__main__.TestSnakePy) ... ok
test_grow (__main__.TestSnakePy) ... ok
test_remove (__main__.TestSnakePy) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.070s

OK
>>>
```

Table 3: Test Case for constructor

Function Tested	Snake()
Preconditions	none
Expected outcome	a Snake() object is instantiated
Function Input	none
Test Description	This test asserts equality of two Snake() objects once in
Testing Type	Correctness

Table 4: Test Case for changeDir

Function Tested	changeDir(newDirection)
Preconditions	Snake object is already instantiated
Expected outcome	The test object's direction is updated if it is a valid input
Function Input	an integer from [-1,1,-2,2]

Test Description	This test uses Snake objects in different directions and calls changeDir on them with all possible direciton inputs
Testing Type	Correctness and Robustness

Table 5: Test Case for grow

Function Tested	grow
Preconditions	there is an instantiated Snake() object
Expected outcome	The snake's length increases by 1
Function Input	none
Test Description	This test asserts equality between pregrown Snake objects and newly grown objects
Testing Type	Correctness

Table 6: Test Case for remove

Function Tested	remove
Preconditions	a Snake object is instantiated
Expected outcome	every point in the snake after the inputted index is removed
Function Input	integer value corresponding to the index

Test Description	This test asserts equality between the length of a Snake object that has remove executed at various indexes and said indexes+1. This test also tests for abnormal and extreme values
Testing Type	Correctness,Robustness

5.2 Testing for MainMenu.py

```
>>>
test_changeState (__main__.TestMainMenuPy) ... ok
test_constructor (__main__.TestMainMenuPy) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.042s

OK
```

Table 7: Test Case for constructor

Function Tested	MainMenu()
Preconditions	none
Expected outcome	a MainMenu object is instantiated
Function Input	none
Test Description	constructor equality test
Testing Type	Correctness

Table 8: Test Case for changeState

Function Tested	changeState
Preconditions	a MainMenu object has been instantiated
Expected outcome	the state is updated if input is valid
Function Input	string value corresponding to the new state
Test Description	This test asserts equality between the inputted newState and the state of the MainMenu object after running changeState on it
Testing Type	Correctness,Robustness

```
>>>
testRandomPos (__main__.TestFood) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.032s

OK
>>>
```

```
testDidSnakeHitBorder (__main__.TestPlayMap) ... ok
testDidSnakeSelf (__main__.TestPlayMap) ... ok
testGetCurrentState (__main__.TestPlayMap) ... ok
testIsSnakeDead (__main__.TestPlayMap) ... ok
testSetDiff (__main__.TestPlayMap) ... ok
testUpdateState (__main__.TestPlayMap) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.095s

OK
```

```
>>>
testGetCurrentState (__main__.TestGamePause) ... ok
testUpdateState (__main__.TestGamePause) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.045s

OK

>>>
testGetCurrentState (__main__.TestGameOver) ... ok
testUpdateState (__main__.TestGameOver) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.043s

OK
```

6 Testing functional requirements