Test Report

Alex Guerrero, Keyur Patel and Shafeeq Rabbani

December 8, 2015

1 Revisions

Table 1: Revisions

Name	Date	Description
Keyur Patel	27/11/2015	Created Test Report latex file
Keyur Patel	27/11/2015	Added table template for unit testing AND info
Alex Guerrero	27/11/2015	Edited Structural Testing
Shafeeq Rabbani	27/11/2015	Edited Usability Testing
Shafeeq Rabbani	08/12/2015	Added Coverage Matrix
Shafeeq Rabbani	08/12/2015	Added Summary of Results

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

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

For most of the applicable functions and methods, we tested for robustness by inputting abnormal conditions and extreme domains.

5.1 Testing for Snake.py

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

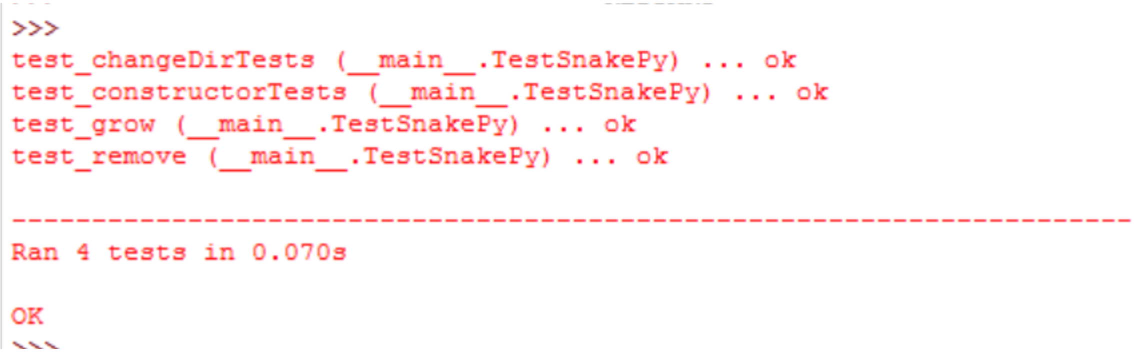
Function Tested	changeDir(newDirection)
Preconditions	Snake object is already instantiated
Expected outcome	The test object's direction is updated if it is a valid input
Function Input	an integer from [-1,1,-2,2]
Test Description	This test uses Snake objects in different directions and calls changeDir 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



```
>>>
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
>>>
```

5.2 Testing for MainMenu.py

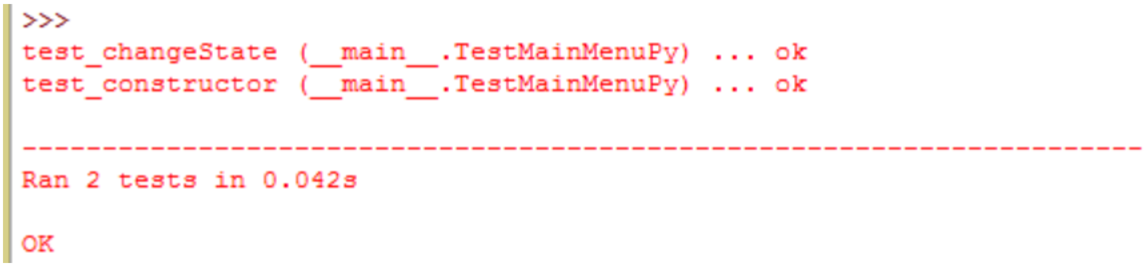
Table 7: Test Case for constructor

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

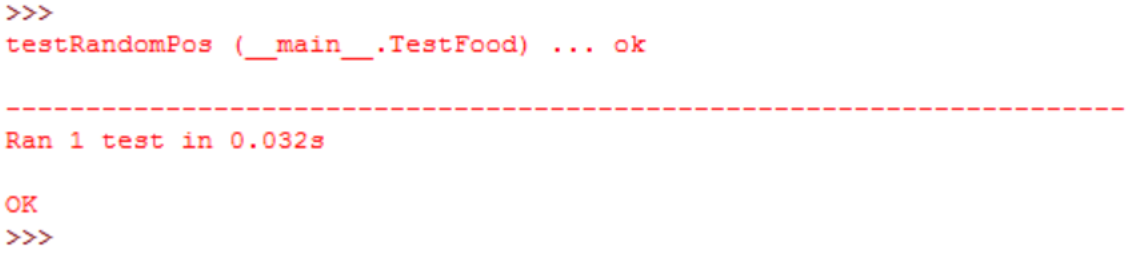


```
>>>
test_changeState (__main__.TestMainMenuPy) ... ok
test_constructor (__main__.TestMainMenuPy) ... ok

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

OK
```

5.3 Testing for Food.py

Table 9: Test Case for constructor

Function Tested	Food()
Preconditions	none
Expected outcome	random x and y position
Function Input	none
Test Description	Assert that two food objects have different positions
Testing Type	Correctness



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

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

OK
>>>
```

5.4 Testing for PlayMap.py

Table 10: Test Case for setDiff

Function Tested	setDiff(difficulty)
Preconditions	none
Expected outcome	difficulty changes to number passed
Function Input	0, 1, and 2
Test Description	Assert that difficult changes after being set
Testing Type	Correctness

Table 11: Test Case for didSnakeHitBoarder

Function Tested	didSnakeHitBoarder()
Preconditions	moving snake head to desired test location
Expected outcome	Return true when snake hits border, False else
Function Input	none
Test Description	Assert that function returns true only when snake hits border
Testing Type	Correctness

Table 12: Test Case for didSnakeHitSelf

Function Tested	didSnakeHitSelf()
Preconditions	moving snake head to desired test location
Expected outcome	Return True when snake hits self, False else

Function Input	none
Test Description	Assert that function returns true only when snake hits self
Testing Type	Correctness

Table 13: Test Case for isSnakeDead

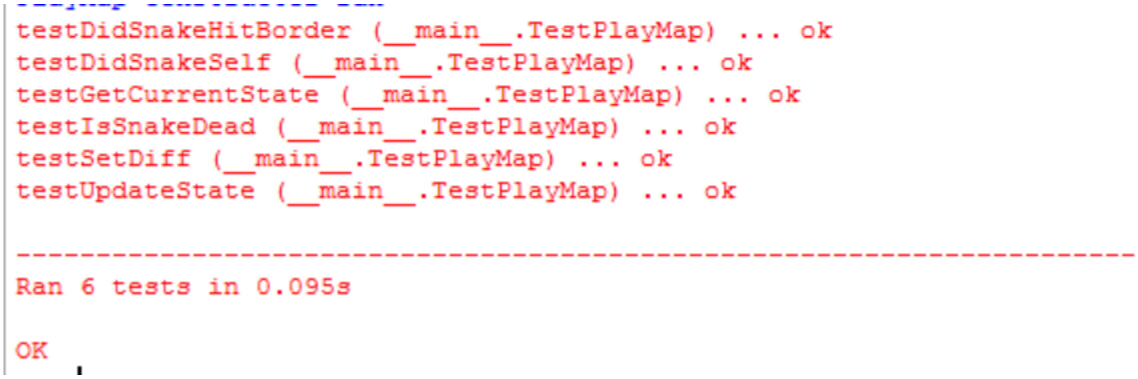
Function Tested	isSnakeDead()
Preconditions	moving snake head to desired test location
Expected outcome	Return True when snake dies, False else
Function Input	none
Test Description	Assert that snake dies when it hits itself or a border
Testing Type	Correctness

Table 14: Test Case for updateState

Function Tested	updateState()
Preconditions	snake and food position
Expected outcome	Snake grows by 1 when it eats food, remains the same length else
Function Input	none
Test Description	Assert that playMap updates correctly
Testing Type	Correctness

Table 15: Test Case for getCurrentState

Function Tested	getCurrentState()
Preconditions	moving snake head to desired test location
Expected outcome	Return -1 when dead, an array of state variables else
Function Input	none
Test Description	Assert that getCurrentState returns correct value
Testing Type	Correctness



```
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
```

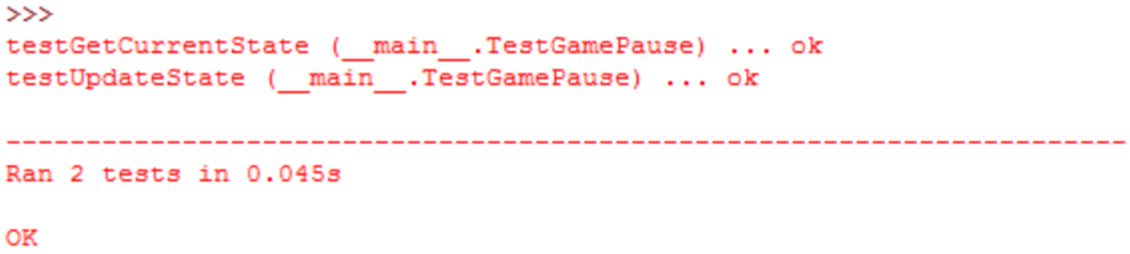
5.5 Testing for GamePause.py

Table 16: Test Case for updateState

Function Tested	updateState(score)
Preconditions	none
Expected outcome	Score variable in pause updates to number passed
Function Input	21
Test Description	Assert that score changes after being updated
Testing Type	Correctness

Table 17: Test Case for getCurrentState

Function Tested	getCurrentState()
Preconditions	none
Expected outcome	Return an array consisting of score, and the 4 buttons on the display
Function Input	none
Test Description	Assert that function returns the proper array of items
Testing Type	Correctness



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

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

OK
```

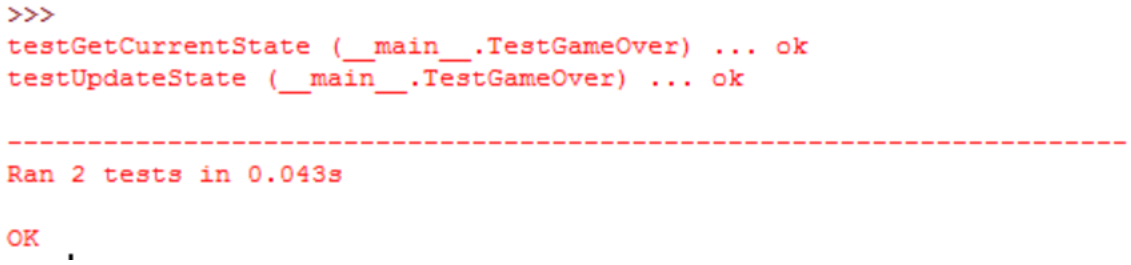
5.6 Testing for GameOver.py

Table 18: Test Case for updateState

Function Tested	updateState(score)
Preconditions	none
Expected outcome	Score variable in pause updates to number passed
Function Input	21
Test Description	Assert that score changes after being updated
Testing Type	Correctness

Table 19: Test Case for getCurrentState

Function Tested	getCurrentState()
Preconditions	none
Expected outcome	Return an array consisting of score, and the 2 buttons on the display
Function Input	none
Test Description	Assert that function returns the proper array of items
Testing Type	Correctness



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

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

OK
```

6 Testing functional requirements

7 Usability Testing

Usability testing is carried get response from gamers on their experience of the game. Testing was carried by allowing youth between the age of 18 to 25. The comments and ratings given by this focus group reflect the interests and needs of youth of today.

Table 20: User 1

Number of times played	5
Rate entertainment (from 1 to 10)	8
Rate Power Up feature (from 1 to 10)	11
Rate graphics (from 1 to 10)	8
Suggested Improvements	There must be a way of knowing which difficulty level has been chosen. Response of keys was slow. The game would be more interesting had it been multiplayer.

Table 21: User 2

Number of times played	2
Rate entertainment (from 1 to 10)	7.5
Rate Power Up feature (from 1 to 10)	10
Rate graphics (from 1 to 10)	8
Suggested Improvements	There should be more menu options.

Table 22: User 3

Number of times played	6
Rate entertainment (from 1 to 10)	6
Rate Power Up feature (from 1 to 10)	7
Rate graphics (from 1 to 10)	7
Suggested Improvements hline	The game should be more colorful.

Table 23: User 4

Number of times played	5
Rate entertainment (from 1 to 10)	6
Rate Power Up feature (from 1 to 10)	7
Rate graphics (from 1 to 10)	2
Suggested Improvements	There appears to be a lag. Make the score board at the top of the screen more noticeable.

Table 24: User 5

Number of times played	2
Rate entertainment (from 1 to 10)	7.5
Rate Power Up feature (from 1 to 10)	10
Rate graphics (from 1 to 10)	8
Suggested Improvements	There should be more options in the options menu.

Table 25: User 6

Number of times played	8
Rate entertainment (from 1 to 10)	7
Rate Power Up feature (from 1 to 10)	8

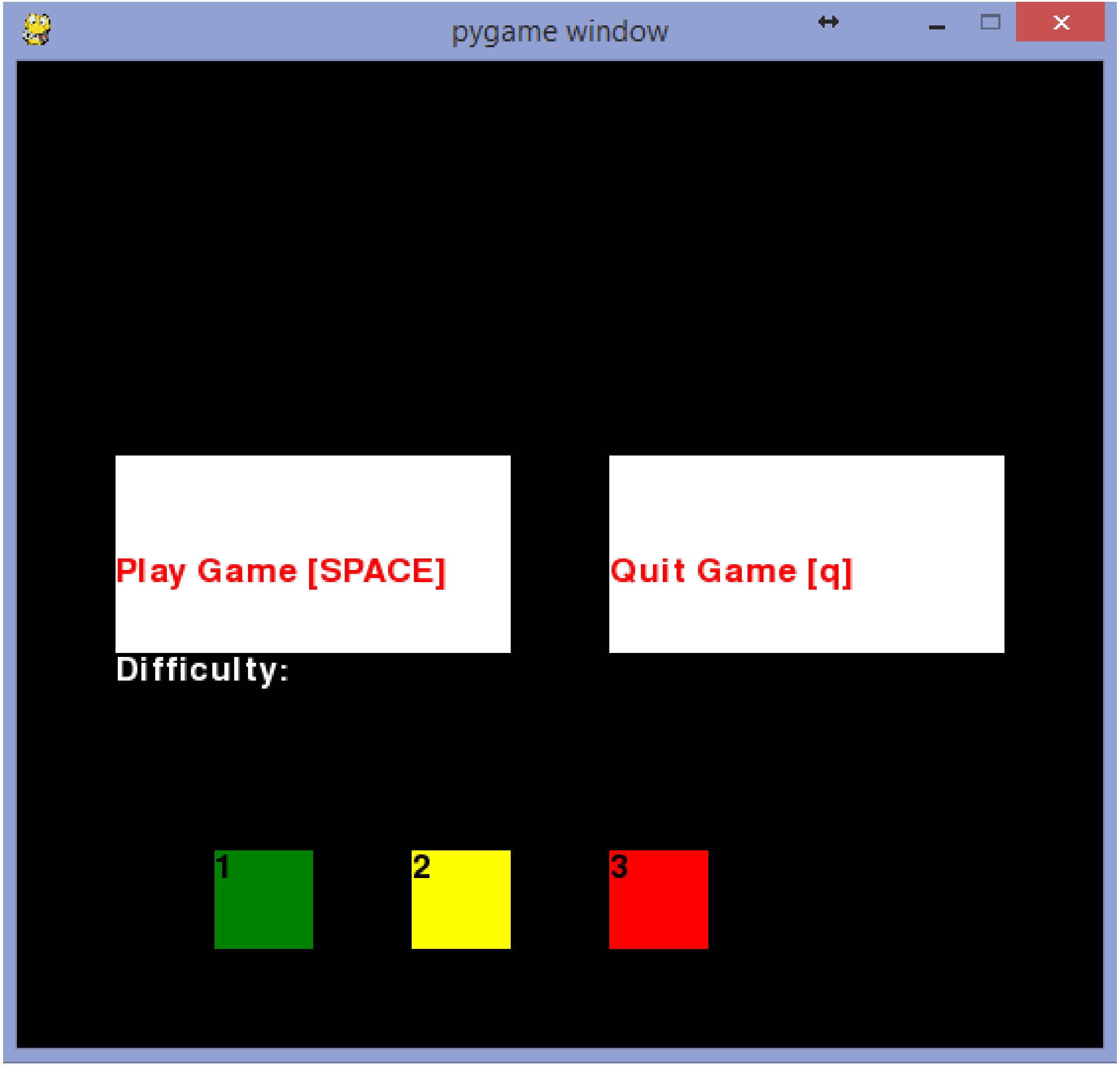
Rate graphics (from 1 to 10)	5 .
Suggested Improvements	The top ten scores ever should be saved

Table 26: User 7

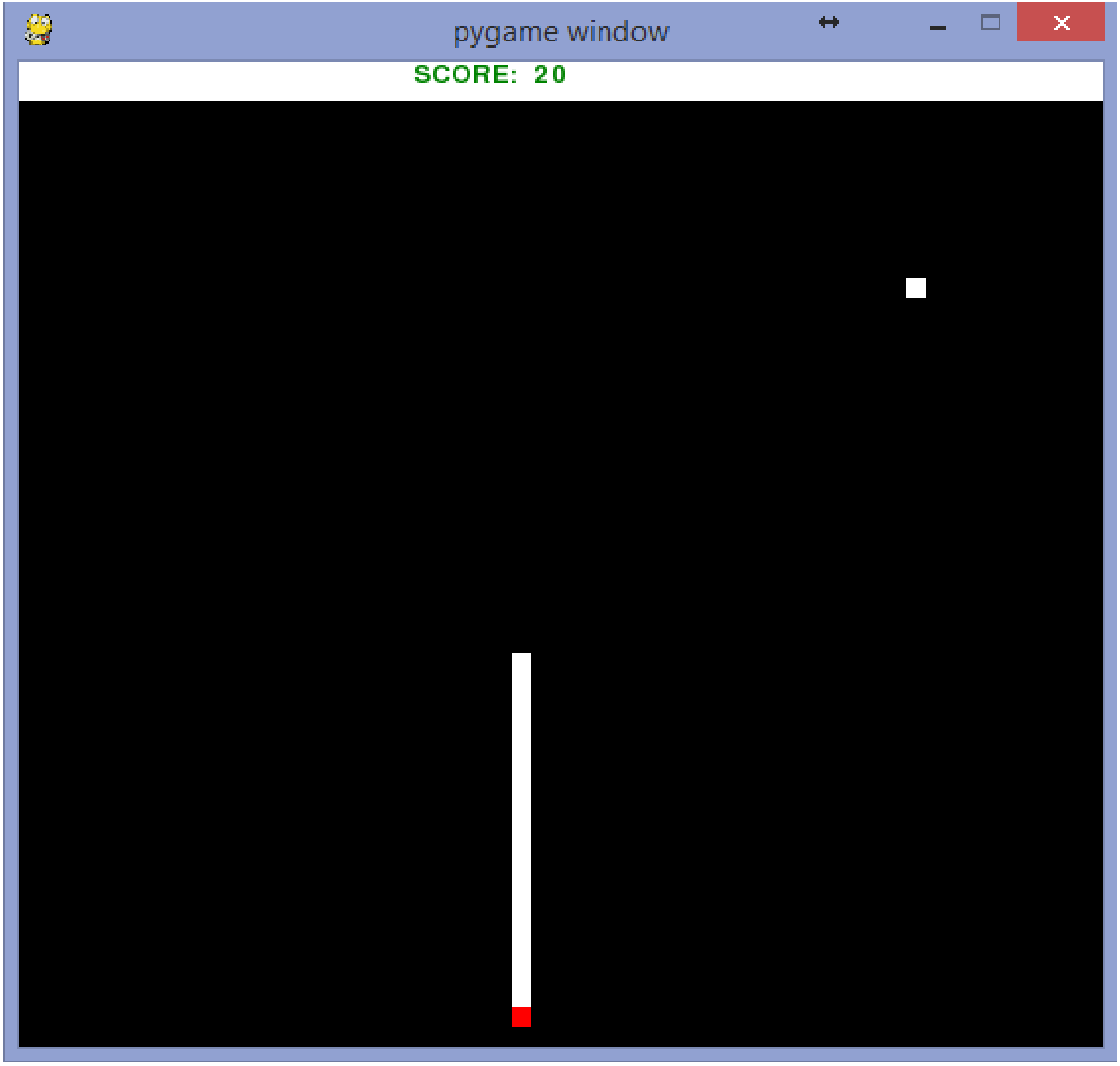
Number of times played	3
Rate entertainment (from 1 to 10)	6
Rate Power Up feature (from 1 to 10)	2
Rate graphics (from 1 to 10)	1
Suggested Improvements	Fix the lag. Add more modes such as a mode to make the snake go through one wall and come out from the other side. Add obstacles for the snake. Reward 'bonus' food points which appear for 5 seconds and disappear if not eaten by snake within this time.

8 GUI Testing

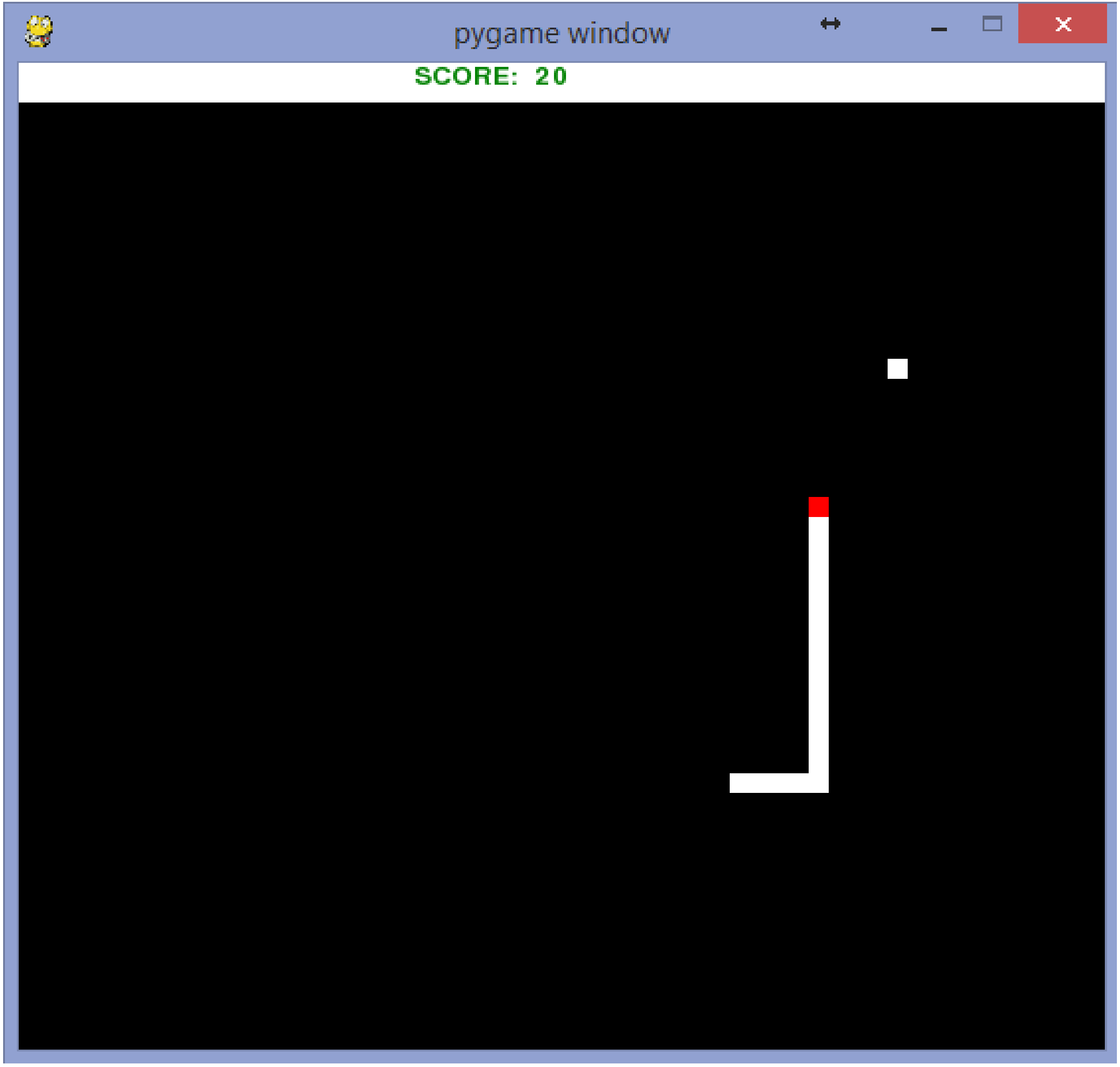
All features of the graphical user interface were tested to see that they correctly respond to the inputs let they be from mouse or the keyboard.
Test Input: The program is first run.
Expected: The option menu appears.
Output: Pass.



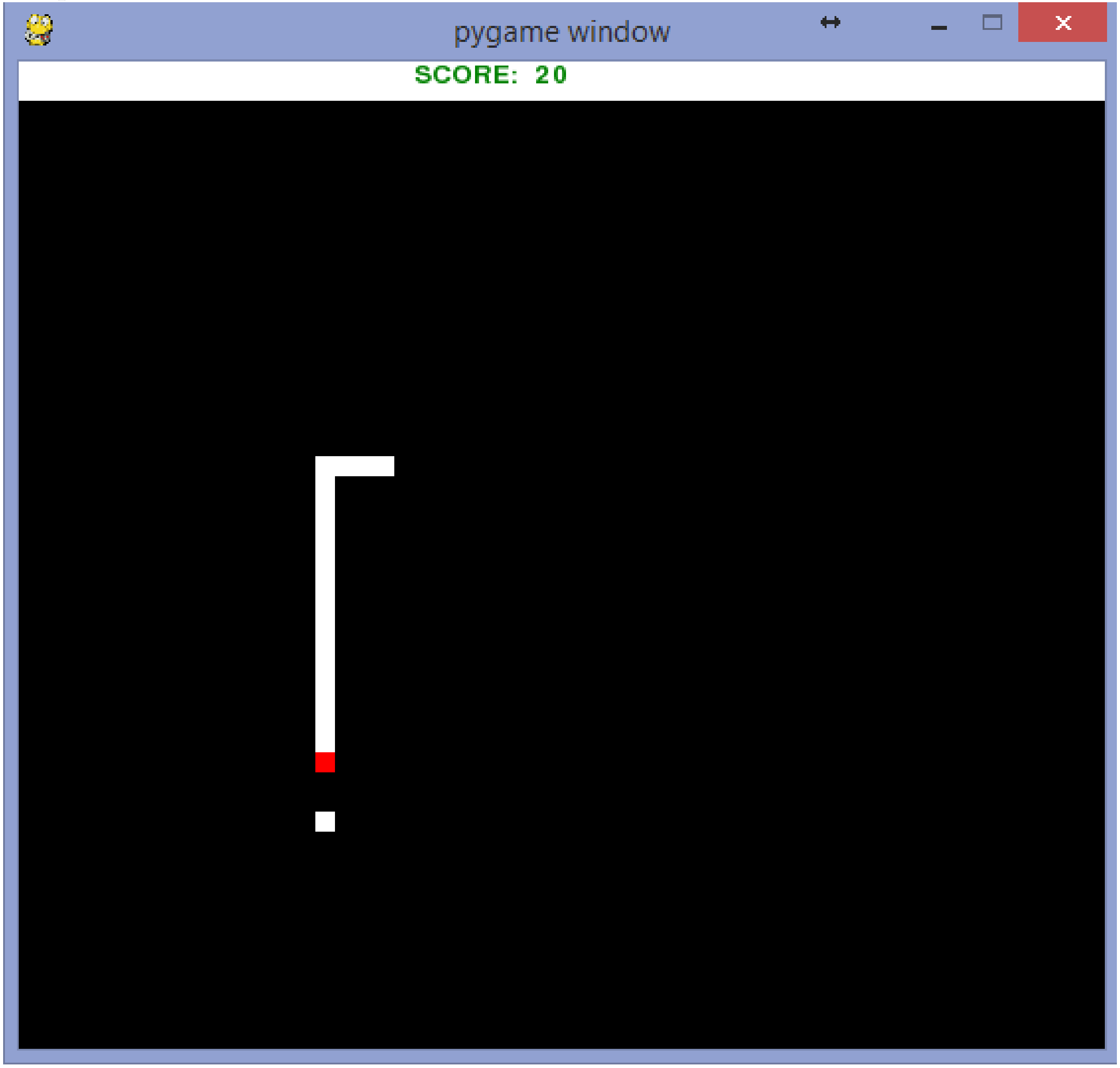
Test Input: In the option menu, Play Game is clicked or the space bar is pressed.
Expected: The snake game begins.
Output: Pass.



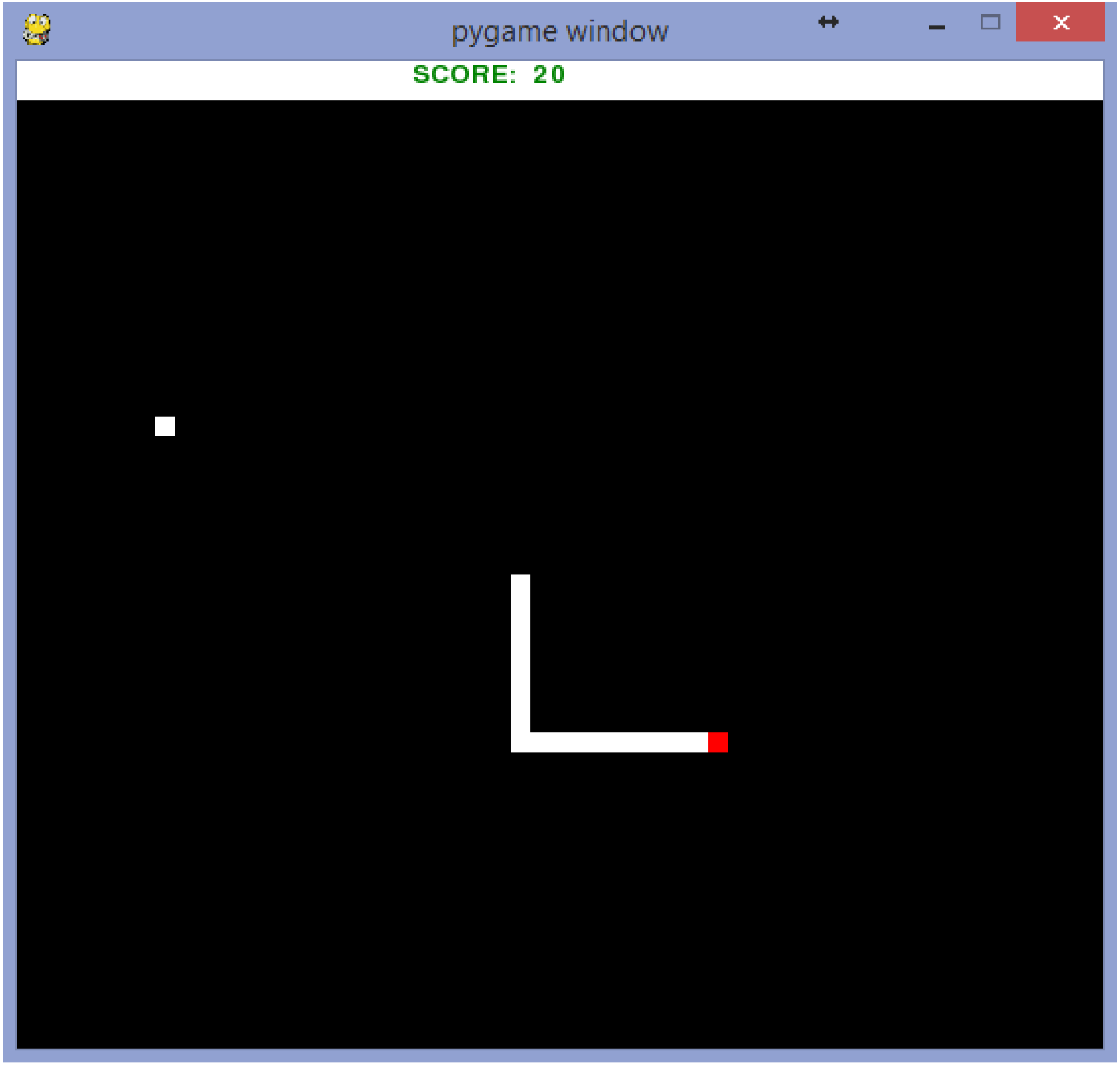
Test Input: In the game, UP arrow key is pressed while the snake is horizontally positioned.
Expected: The snake turns up.
Output: Pass.



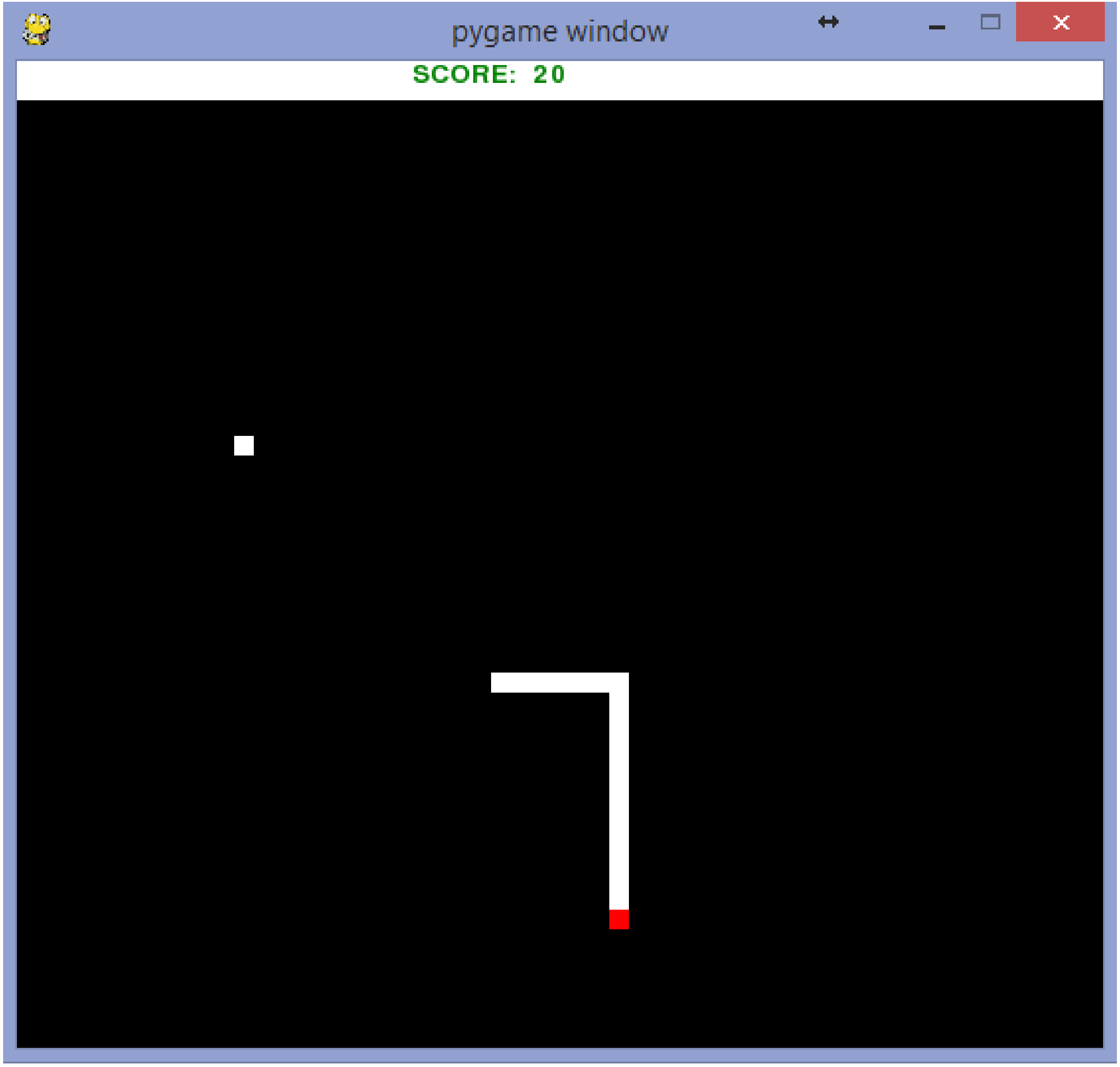
Test Input: The DOWN arrow key is pressed when the snake is horizontally positioned.
Expected: The snake turns down.
Output: Pass.



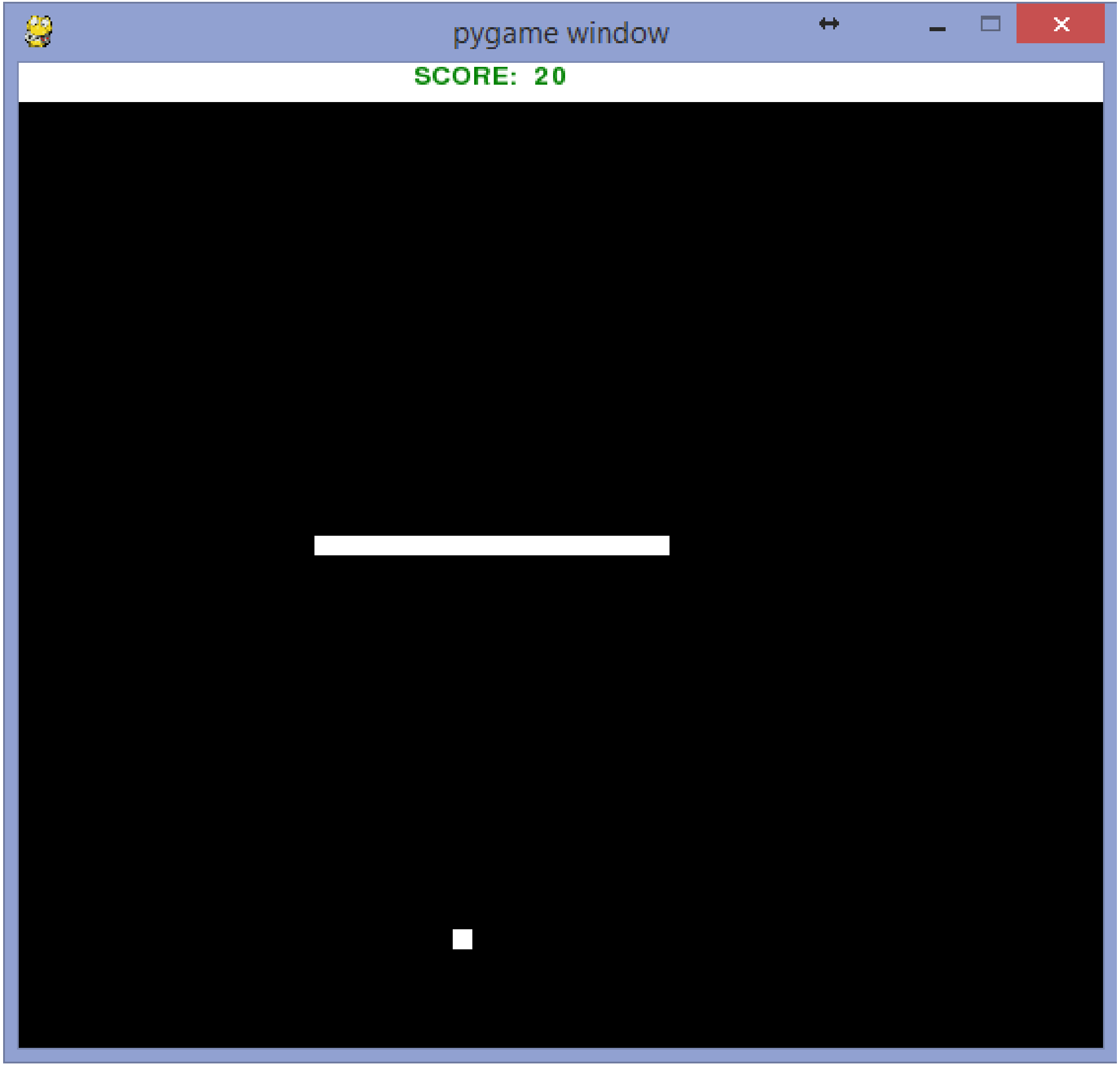
Test Input: the RIGHT arrow key is pressed when the snake vertically positioned.
Expected: The snake turns right.
Output: Pass.



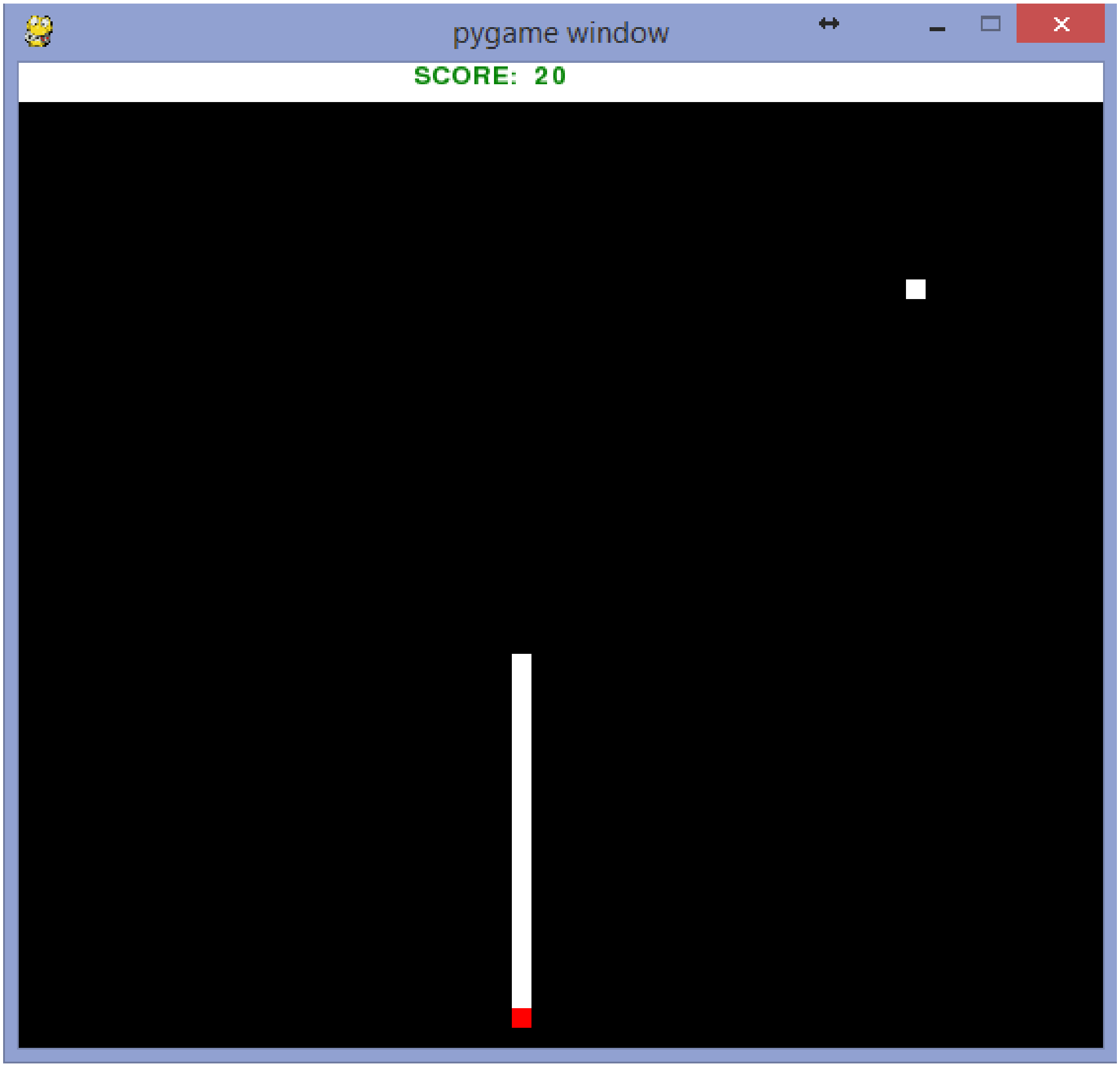
Test Input: The LEFT arrow key is pressed when the snake is vertically positioned.
Expected: The snake turns left.
Output: Pass.



Test Input: The snake crashes into itself.
Expected: ,It's Power up will be used, the size of the snake will shrink and the red head disappears.
Output: Pass.

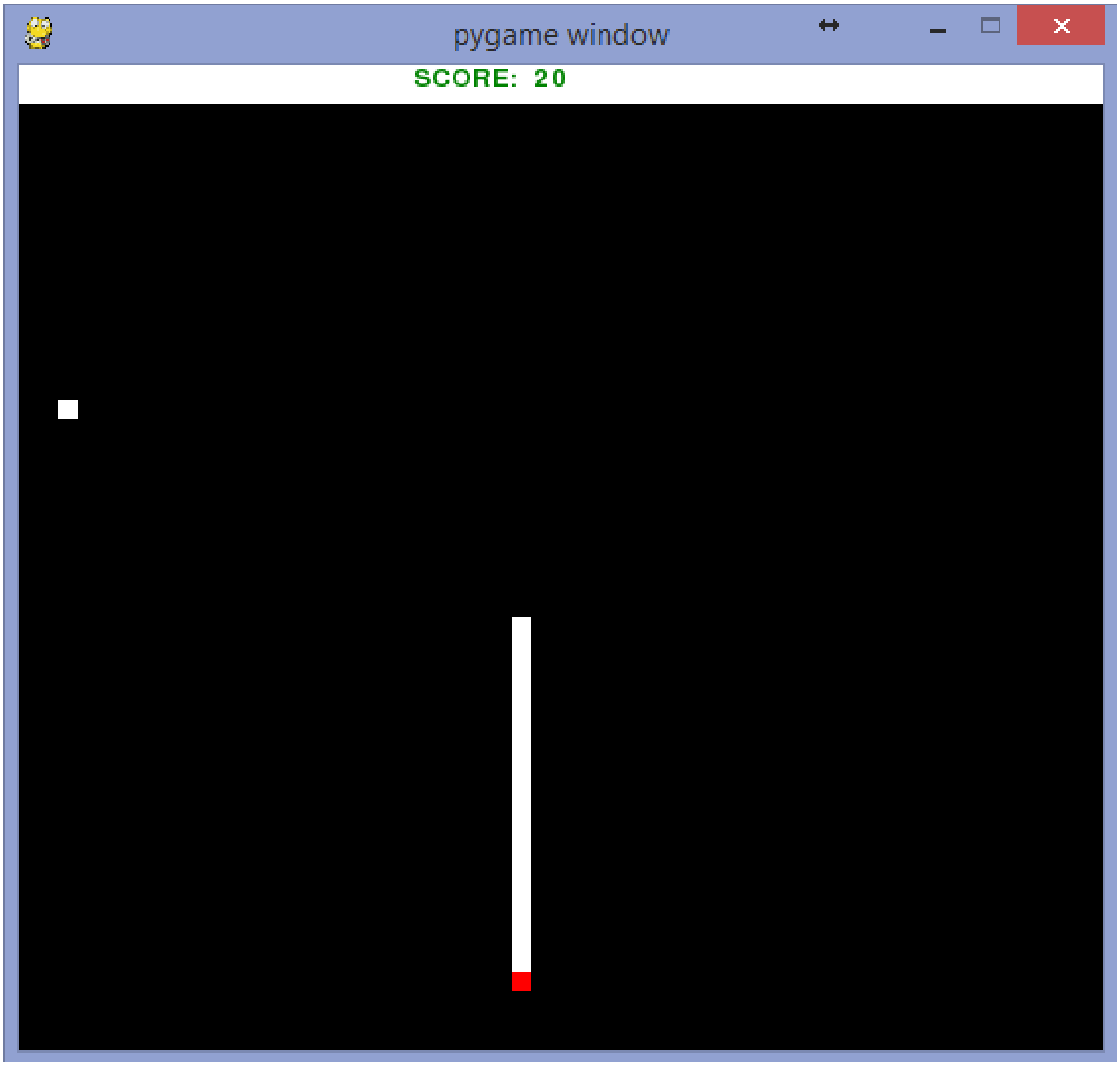


Test Input: The snake crashes the border.
Expected: The Game Over screen pops up and the game ends. The screen displays the score and options to either restart or quit the game.
Output: Pass.

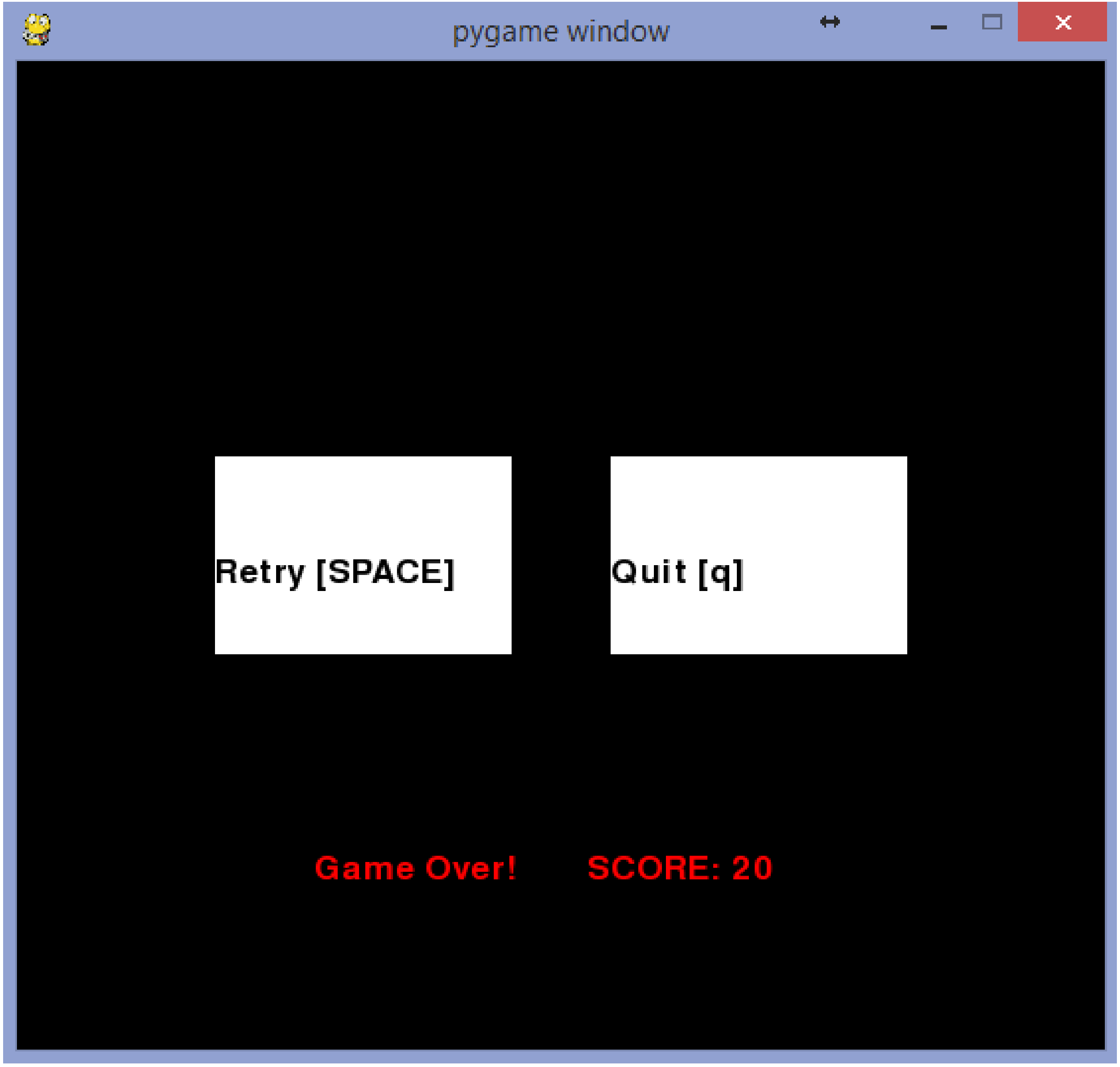
Before: 

SCORE: 20

After:



Test Input: In the game over menu, the space bar is pressed.
Expected: The game successfully restarts.
Output: Pass.



Game Over! SCORE: 20

Test Input: The letter 'q is pressed in the options menu, during the game and in the game over menu.
Expected: Pressing q quits the program.
Output: Pass.
Program window successfully closed in all three test cases.

9 Coverage Matrix

Everything from the list of functional requirements was tested to ensure that it was operating as intended. The table below shows the coverage matrix with the functional requirements and whether the particular requirement was met.

Table 27: Coverage Matrix Table

Requirement	Test Result
R1: The game must start with a main menu screen with a play game button, quit game button, and three difficulty buttons from 1-3	Pass
R2: When the play game button is pressed, and instruction will appear and wait for user input.	Pass
R3: The snake must be controlled by the keyboard. w, a, s, d or directional keys.	Pass
R4: If the snake goes over the same location of a food object, a new food object will be generated and the snake will grow.	Pass
R5: Preceeding the instructions screen, the gameboard will appear with a single snake and food object.	Pass
R6: Pressing the esc key during the game brings up a pause screen that displays resume, main menu and quit game buttons.	Pass
R7: As the player advances in the game, the snake moves faster.	Pass
R8: When the snake hits the border or itself (after power up is used), the game is over.	Pass
R9: The game calculates a score that is based on the length of the snake.	Pass
R10: The game over screen displays the score and a retry buttons and quit game button.	Pass
R11: The snake will start with a power up that allows the player to collide with the snake body once without consequence.	Pass

10 Summary of Results

The code of the Snake game has been split up into modules. This allows for unit testing to occur. All the modules were thoroughly tested and checked for robustness.

Usability testing was also carried out by allowing gamers to have a chance to play the game first hand and fill out the a survey about their experience playing the game.

The results of the tests were promising and it can be concluded that the Snake Game is ready to be released.