SE 3XA3: Test Report
Snake 2.0

Team # 34, Team Send Help
George Mo (moz)
Harrison Lau (lauh3)
Vanessa Truong (truonv1)

December 5, 2017

Contents

List of Tables

List of Figures

This document will summarize the results of the tests for the requirement specifications of our implementation, Snake 2.0, with respect to the SRS and test plan documents. It will outline all the tests conducted on functional and non-functional requirements, and describe the testing approaches that have been taken.

1 Functional Requirements Evaluation

The purpose of the following tests was to ensure that our final product accomplishes the functional requirements outlined by the SRS. It includes testing on the following areas: the menu GUI, the in-game score counter, and the in-game power-ups. Along with the areas of testing are the details and results of the tests.

1.1 Menu GUI

The menu GUI was tested to validate that it performs the correct functions (i.e. taking the user to the right page with respective button clicks). All the menu GUI tests were conducted manually.

1. FR-P-1

   Description: Ensures that the user can activate and play the game from the menu.

   Results: The user is able to play the game from clicking on the 'Play' button in the menu.

2. FR-HTP-1

   Description: Ensures that the user can access the game instructions (how to play) page from the menu.

   Results: The user is able to access the game instructions page from clicking on the 'Help' button in the menu.

3. FR-HTP-2

   Description: Ensures that the user can re-access the menu after accessing the game instructions page.

Table 1: **Revision History**

Date	**Version**	**Notes**
Dec 4	1.0	Started Test Report
Date 5	1.1	Finished all sections for Rev 1

Results: The user is able to re-access the menu from clicking on the 'Back' button on the game instructions page.

4. FR-HTP-3

   Description: Ensures that the user can activate and play the game from the game instructions page.

   Results: The user is able to activate and play the game from clicking on the 'Play' button on the game instructions page.

5. FR-CM-1

   Description: Ensures that the user can access the settings page to change the map size.

   Results: The user is able to access the settings page from clicking on the 'Settings' button on the menu.

6. FR-CM-2

   Description: Ensures that the user can re-access the menu from the settings page.

   Results: The user is able to re-access the menu by clicking on the 'Back' button on the settings page.

7. FR-CM-3

   Description: Ensures that the user can select Map 1 to activate and play the game with the applied map.

   Results: The user is able to select Map 1 which will activate the game and allow the user to play with that map applied to the game.

8. FR-CM-4

   Description: Ensures that the user can select Map 2 to activate and play the game with the applied map.

   Results: The user is able to select Map 2 which will activate the game and allow the user to play with that map applied to the game.

9. FR-CM-5

   Description: Ensures that the user can select Map 3 to activate and play the game with the applied map.

   Results: The user is able to select Map 3 which will activate the game and allow the user to play with that map applied to the game.

1.2 In-Game Score Counter

The in-game score counter was tested to check if there were any errors in the code for calculating the current score. Factors that affect the current score include the in-game power-ups and the in-game timer itself. All of the in-game score counter tests were conducted with JUnit unit testing.

10. FR-SCRCTR-1

    Description: Ensures that the default score when the game is initially activated is 100.

    Results: The game is activated with the user having a default score of 100.

11. FR-SCRCTR-2

    Description: Ensures that the in-game timer will correctly decrement the current score by 1 everytime the snake moves 7 steps.

    Results: The game is activated with the user having a default score of 100, and the current score decreases to 99 after the snake moves 7 steps.

12. FR-SCRCTR-3

    Description: Ensures that the current score is always incremented by 10 whenever the Snake consumes a pellet.

    Results: The current score in the game is incremented by 10 for every pellet that the Snake consumes.

1.3 In-Game Power-ups

The in-game power-ups were tested to ensure that the power-ups are correctly applied to the in-game Snake. Some of the power-ups were tested using JUnit unit testing while others were tested manually.

13. FR-SPD-1

    Description: Ensures that the Snake temporarily speeds up when a speed-up pellet is consumed. Automated unit testing was used to test whether the stepCount variable in the code is incremented up to a limit value, representing how long the Snake will embody the speed-up power-up.

    Results: When the Snake consumes a speed-up pellet, the stepCount variable is incremented up to a value of 35 and the in-game timer delay is changed to speed up the Snake for 35 steps. The Snake then returns to its default speed after reaching 35 steps.

14. FR-SPD-2

    Description: Ensures that a speed-up pellet will randomly spawn at some point in time after its last spawn. This was tested manually by checking whether a speed-up pellet would respawn in an undefined sequence.

    Results: A speed-up pellet will eventually respawn at some point in time after it has been spawned. It will respawn after a random number of times.

15. FR-SPD-3

    Description: Ensures that the Snake temporarily slows down when a slow-down pellet is consumed and that the in-game score counter decrements at a slower rate. Automated unit testing was used to test whether the stepCount variable in the code is incremented up to a limit value and if the decreased_rate variable is changed, representing how long the Snake will embody the slow-down power-up and the new rate at which the counter is decremented.

    Results: When the Snake consumes a slow-down pellet, the stepCount variable is incremented up to a value of 35 and the in-game timer delay is changed to slow down the Snake for 35 steps. The decreased_rate variable is changed to a higher value so that the current score is decremented every 17 steps instead of 7. The Snake then returns to its default speed after reaching 35 steps.

16. FR-SPD-4

    Description: Ensures that a slow-down pellet will randomly spawn at some point in time after its last spawn. This was tested manually by checking whether a slow-down pellet would respawn in an undefined sequence.

    Results: A slow-down pellet will eventually respawn at some point in time after it has been spawned. It will respawn after a random number of times.

17. FR-BNDRY-1

    Description: Ensures that the self-invincibility pellet will apply to the Snake whenever it collides with its body while moving in the RIGHT direction. This was tested manually.

    Results: When the Snake consumes a self-invincibility pellet, it is able to move through its body going in the RIGHT direction.

18. FR-BNDRY-2

    Description: Ensures that the self-invincibility pellet will apply to the Snake whenever it collides with its body while moving in the LEFT direction. This was tested manually.

    Results: When the Snake consumes a self-invincibility pellet, it is able to move through its body going in the LEFT direction.

19. FR-BNDRY-3

    Description: Ensures that the self-invincibility pellet will apply to the Snake whenever it collides with its body while moving in the UP direction. This was tested manually.

    Results: When the Snake consumes a self-invincibility pellet, it is able to move through its body going in the UP direction.

20. FR-BNDRY-4

    Description: Ensures that the self-invincibility pellet will apply to the Snake whenever it collides with its body while moving in the DOWN direction. This was tested manually.

    Results: When the Snake consumes a self-invincibility pellet, it is able to move through its body going in the DOWN direction.

21. FR-BNDRY-5

    Description: Ensures that a self-invincibility pellet will randomly spawn at some point in time after its last spawn. This was tested manually by checking whether a self-invincibility pellet would respawn in an undefined sequence.

    Results: A self-invincibility pellet will eventually respawn at some point in time after it has been spawned. It will respawn after a random number of times.

22. FR-BNDRY-6

    Description: Ensures that the Snake can temporarily move through its own body when a self-invincibility pellet is consumed. Automated unit testing was used to test whether the immunityCount variable in the code is incremented up to a limit value, representing how long the Snake will embody the self-invincibility power-up.

    Results: When the Snake consumes a self-invincibility pellet, the immunityCount variable is incremented up to a value of 35. The Snake will not be able to move through its own body after a fixed amount of time.

2 Nonfunctional Requirements Evaluation

The purpose for running the following tests was to ensure that all the non-functional requirements are accomplished by the final product, with respect to the SRS document. The non-functional requirements testing covers the areas of usability and performance.

2.1 Usability

The usability tests were conducted to measure the ease of use, understandability, and overall satisfaction that application provides for different users.

1. SS-1

   Description: Ensures that the user can run Snake 2.0 on any Windows 10 OS via the executable file, with all of the dependent files installed on their computer. This will be conducted manually by using a USB stick to transfer the files across all of the team members' laptops to ensure that it will run on any Windows 10 OS device.

   Results: The application successfuly runs as an executable file on all team member laptops.

2. SS-2

   Description: Ensures that the application will be easy to use for 60% of users without assistance. This will be conducted by allowing a group of selected people to use the application and perform the requested function.

   Results: Of a group of 5 people, all 5 individuals were able to use the application to its fullest extent within 5 minutes.

3. SS-3

   Description: Ensures that the application will be easy to install for 60% of users without assistance. This will be conducted by providing a group of selected people the installation files for the application, and asking them to install it onto their devices to play.

   Results: Of a group of 5 people, 3 individuals were able to install the application on their device and play the game.

4. SS-4

   Description: Users will be asked to rate the application out of 10 on the following three categories: ease of use, understandability, and overall satisfaction.

   Results: Of a group of 5 people, the average ratings for ease of use, understandability, and overall satisfaction are 10, 9, and 8.5 respectively.

5. SS-5

Description: Users will be asked to rate both the original Snake game and our reimplemtation, Snake 2.0, out of 10 on the following three categories: ease of use, understandability, and overall satisfaction. The results will be compared to check whether our implementation exceeds the standards of the original ga ein each category.

Results: Of the ratings gathered from a group of 5 people, the ratiings for ease of use, understandability, and overall satisaction for our implementation are equal or higher than those for the original application.

2.2 Performance

The performance tests were conducted to measure the response and execution times of different events that would occur with using the application.

1. SS-6

   Description: Users will run the application's executable file 5 times on different devices, and the time taken for the application to start up will be recorded to ensure that it does not exceed 5 seconds.

   Results: The application opens up with 1-3 seconds of running the executable file.

2. SS-7

   Description: Users will press the arrow keys to move the Snake around the board. This will be performed a number of times to ensure that the Snake responds to arrow key presses within half a second and moves in the respective direction.

   Results: The Snake responds to the arrow key presses within a millisecond and moves in the requested direction with respect to the arrow key pressed.

3. SS-8

   Description: Users will each of the 3 available map sizes in different trials, for a total of 5 trials. The time taken for the application to transition into in-game mode after the user clicks the selected map size will be recorded to ensure that the time does not exceed 3 seconds.

   Results: The application takes approximately half a second to start the in-game mode with the applied map size.

3 Comparison to Existing Implementation

While Snake 2.0 does incorporate a fair amount of additional features, it still shares the basic game objectives with the exisiting implementation. In the original implementation, the in-game Snake can only move in one of the four cardinal directions, will grow by a unit of 1 whenever it consumes a pellet, and will die if it collides with one of the four windowing borders or its body. These are all functionalities that have been carried over to our reimplementation. On top of these core functions, Snake 2.0 adds more elements to the game such as power-ups, map sizing changes, and an altered score tracking system.

For our project, testing was primarily conducted on the additional features that exclude the basic functionalities of the existing implementation. Although manual testing was still conducted on these functioinalities to ensure that our copy of the existing implementation functioned properly, it is assumed that the code for the original game has already been tested, therefore making it redundant to again test those basic functionalities in our implementation. Most of our testing focus was placed on ensuring that each power-up had an actual effect on the Snake, the map sizes would change in response to the user's request, and the in-game score counter was properly updating the score and responding to the power-ups which affect it.

Test case SS-5 under the non-functional requirements testing section was conducted to compare the ease of use, understandability, and overall satisfaction of the existing implementation with our reimplementation. With the addition of our gaming enhancements to the original game, Snake 2.0 received equal or higher ratings to the original. Although our implementation is more complex and less simplistic than the original, the added challenges to playing the game definitely creates a more enjoyable and entertaining experience for the players, as indicated with the results of this test case.

4 Unit Testing

The JUnit testing framework was used to conduct unit testing for this project.

4.1 Unit Testing of Game start-up and Reset

The initial state of the game (when in-game mode is activated) was tested by asserting that the default current score was 100, the snake's body was a length of 3 units, and that the initial direction of the Snake was in the RIGHT direction. The unit test for the game start-up was conducted by setting up the board game environment, initiating the game, and then passing the expected values for the above variables into assert statements. The test was run several times to ensure that all test cases passed.

The reset method of the game was tested by setting up the board game environment, initiating the game, calling the move method on the Snake a couple of times along with having the snake consume several pellets to alter its body length and current score value. Then, the reset method is invoked and assert statements are used to check whether the current score value is resetted to 100 and that the body length is resetted to 3.

4.2 Unit Testing of Pellet Consumption

Unit testing was conducted on pellet consumption to ensure that for every pellet that the Snake consumed, the current score would increase by 10. This was done by setting up the board game environment, having the Snake consume a pellet, and then passing the values for the expected current score into an assert stateent to validate that it has increased by 10. The test was also performed in the scenario where the current score has been decremented by 1 after the Snake makes 7 moves, and the Snake then consumes a pellet. Thus, unit tests were conducted to verify that the points would increase by 10 relative to the current score.

4.3 Unit Testing of Power-ups

Testing was conducted on each pellet power-up to ensure that the correct effect would take place on the in-game Snake. There were a total of three power-up pellets: a speed-up pellet, a slow-down pellet, and a self-invincibility pellet. Testing on these three power-ups were conducted in three separate methods.

Each method sets up the board game environment, along with setting the respective pellet to active by using boolean values. If the respective pellet value was true (active), then that pellet's effect would take place on the Snake. Then, by setting the values for the Snake's head location and the pellet location to identical values, the scenario of the Snake consuming that power-up pellet is successfully set up. Then, the expected values for the in-game variables for tracking the points decrementation rate and the step counts are passed through assert statements. This verifies that the active pellet along with its intended effect are functioning.

4.4 Unit Testing of Score Counter

The in-game score counter was tested on the following aspects: the decrementation rate, the slow-down pellet rate, and pellet consumption. After setting up the initial board game environment, the decrementation value was tested by passing the expected value for the decrementation rate into an assertment statement to ensure that it is set to the default value. Another unit test was written to test the decrementaton rate of the score counter after a slow-down pellet is consumed by the Snake. Consuming a slow-down pellet will change the decrementation rate of the score counter to double its default value, which slows down the rate in which points are decremented. The expected value (the default decrementation rate multiplied by 2) is passed through an assert statement after the slow-down pellet is set to active in the test case. Lastly, pellet consumption is tested with regards to the score counter to ensure that the score counter updates every time the Snake consumes a pellet. The expected values for the current score after the Snake has consumed several pellets are passed through assert statements to validate this.

5 Changes Due to Testing

There were not many changes to any of the previously stated test cases in the Test Plan document. However, additional test cases were added to cover the additional functional requirements to our implementation, and a slight change in one of our power-up functions had to be accounted for in its respective test case.

Test cases had to be written and conducted for the addition of an in-game score counter. The in-game score counter brought the functionalities of pellets adding 10 points to the score, as well as bringing in an in-game timer that decrements the score periodically. Thus, additional testing was planned and conducted to verify that the in-game score counter functioned correctly in the absence of bugs.

Furthermore, there was also a change to the power-up pellet for boundary invincibility. Instead of having that pellet provide the Snake with boundary collisioin invincibility, the pellet provides the Snake with self-collision invincibility, allowing it to move through its body, which would otherwise end the game. In response to this, the test cases for testing if the Snake could collide with the LEFT, RIGHT, UPPER and LOWER walls without reaching a game over were changed to checking whether the Snake could move through its body going in those 4 cardinal directions without reaching a game over.

6 Automated Testing

There were no automation testing tools used during the implementation of this project. As a good portion of our code is GUI-related and much of our testing was easily and efficiently accomplished through manual or unit testing, automated testing was not necessary. However, it may be worth looking into for the future if we decide to add more internal functions and expand on our project. With a broader project scope and wider time constraints, automated testing tools would definitely be beneficial to invest in.

7 Trace to Requirements

The following traceability matrix shows the traceability between the test cases to the requirements stated in the SRS document.

Test	**Requirements**
	Functional Requirements Testing
FR-P-1	FR4, FR7
FR-HTP-1	FR4, FR7
FR-HTP-2	FR4
FR-HTP-3	FR4
FR-CM-1	FR4, FR5, FR7
FR-CM-2	FR4
FR-CM-3	FR4, FR5, FR7
FR-CM-4	FR4, FR5, FR7
FR-CM-5	FR4, FR5, FR7
FR-SCRCTR-1	FR6, FR7
FR-SCRCTR-2	FR6, FR7
FR-SCRCTR-3	FR6
FR-SPD-1	FR2, FR3, FR6
FR-SPD-2	FR2, FR3
FR-SPD-3	FR2, FR3, FR6
FR-SPD-4	FR2, FR3
FR-BNDRY-1	FR1, FR3
FR-BNDRY-2	FR1, FR3
FR-BNDRY-3	FR1, FR3
FR-BNDRY-4	FR1, FR3
FR-BNDRY-5	FR1, FR2, FR3
FR-BNDRY-6	FR1, FR2, FR3, FR6
	Non-functional Requirements Testing
SS-1	NFR4, NRF5, NFR6
SS-2	NFR1, NFR2, NFR7
SS-3	NFR1, NFR2, NFR3, NFR4, NFR7, NFR8, NR9
SS-4	NFR1, NFR2, NFR4, NFR5, NFR7, NFR8, NR9
SS-5	NFR1, NFR2, NFR4, NFR5, NFR7, NFR8, NR9
SS-6	NFR3, NFR4, NFR5
SS-7	NFR3, NFR5
SS-8	NFR3

Table 2: Trace Between Test Cases and Requirements

8 Trace to Modules

The following traceability matrix shows the traceability between the test cases and the modules stated in the Module Guide document..

Test	**Modules**
	Functional Requirements Testing
FR-P-1	M1, M2, M3, M4
FR-HTP-1	M3, M4
FR-HTP-2	M3, M4
FR-HTP-3	M3, M4
FR-CM-1	M3, M4
FR-CM-2	M3, M4
FR-CM-3	M3, M4
FR-CM-4	M3, M4
FR-CM-5	M3, M4
FR-SCRCTR-1	M4, M5
FR-SCRCTR-2	M4, M5
FR-SCRCTR-3	M4, M5
FR-SPD-1	M4
FR-SPD-2	M4
FR-SPD-3	M4
FR-SPD-4	M4
FR-BNDRY-1	M4
FR-BNDRY-2	M4
FR-BNDRY-3	M4
FR-BNDRY-4	M4
FR-BNDRY-5	M4
FR-BNDRY-6	M4
	Non-functional Requirements Testing
SS-1	M1, M2, M3, M4, M5
SS-2	M3, M4, M5
SS-3	M1
SS-4	M1, M3, M4, M5
SS-5	M1, M3, M4, M5
SS-6	M1, M2
SS-7	M4
SS-8	M3, M4

Table 3: Trace Between Test Cases and Modules

9 Code Coverage Metrics

Many of the functional and non-functional requirements of our implementation were conducted manually, particularly those functions involving interactions with the menu GUI as well as mouse and keyboard inputs. Thus, the code coverage for unit testing was only around 60%, which is approximately 15% less than we initially anticpated, as a majority of our code is GUI and I/O device relevant (i.e. menu GUI set-up, scoreboard set-up, keypressed Snake movement, mouse clicks for menu interaction).