SE 3XA3: Test Plan
SnakeGame Project

Team L03G09
Longwei Ye (yel16)
Qiang Gao (gaoq20)
Zhiwei Li (liz342)

March 12, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2022/3/9	1.0	Adding contexts for Section 1
2022/3/9	1.0.1	Adding contexts for Section 2
2022/3/10	1.0.2	Adding contexts for Section 6
2022/3/11	1.0.3	Adding contexts for Section 4
2022/3/11	1.0.4	Adding contexts for Section 3, 5

1 General Information

1.1 Purpose

The purpose for the testing of the SnakeGame project is to confirm that the codes for building this project is implemented properly.

1.2 Scope

The test plan is testing the functionality of the re-implementation of SnakeGame project, which includes the objective of controlling the snake to move and to grow up by "eating" elements.

As the project is developed on HTML with JavaScript, this test plan is implemented by Mocha.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
SRS	Software Requirements Specification
JS	JavaScript
PCB	Preset Color Button
GUI	Graphical User Interface
OS	Operation System
FR	Functional Requirement
NFR	Non Function Requirement

1.4 Overview of Document

This project is re-building the open source SnakeGame project, which is the game that allows the player to control the "snake" to gain points by eating different kinds of boosters. All the requirements are listed in the SRS file.

Table 3: **Table of Definitions**

Term	**Definition**
HTML	The code that is used to structure a web page and its content
JavaScript	A programming language used to implement core script for World Wide Web
Mocha	A JavaScript test framework
Navigator object	An object contains information about the browser.

2 Plan

2.1 Software Description

The software is stimulating a game situation, and it will allow the user to control the "snake" displayed on the chessboard. It will be implemented by HTML with JavaScript.

2.2 Test Team

The members that is taking responsibility for this the Testing is Longwei Ye, Qiang Gao, and Zhiwei Li.

2.3 Automated Testing Approach

Automated Testing are approached by setting up unit testing cases through Mocha.

2.4 Testing Tools

This test plan is consisting of manually testing and the unit testing framework Mocha.

2.5 Testing Schedule

Please see the Gantt chart(schedule.pdf) at the same folder.

Task	Team Member	Date
Start the game	Zhiwei Li and Qiang Gao	2022/3/17
Playing the game	Longwei Ye	2022/3/17
End the game	Zhiwei Li	2022/3/21
During the game	Qiang Gao	2022/3/21
Look and Feel Requirements	Longwei Ye	2022/3/24
Performance Requirements	Zhiwei Li	2022/3/28
Cultural Requirements	Qiang Gao	2022/3/28

Table 4: **Testing Assignments**

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Start the game

3.1.1.1 FR1

1. STG-FR1-1

   Type: Functional, Dynamic, Manual

   Initial State: The SnakeGame.html file is open by the player.

   Input: One of the PCB(Red, Green, Blue, Default) is clicked. Then start game button is clicked.

   Output: The snake changes into corresponding color in the new started game.

   How test will be performed: The function that changes the snake's color will be executed after the PCB is clicked and the change will be demonstrated in the next new game. Therefore when a new game is started, we will check if there is a change of the corresponding color to the snake.

2. STG-FR1-2

   Type: Functional, Dynamic, Manual

Initial State: The SnakeGame.html file is open by the player.

Input: A 6-digit hex number is typed into the "Your Own Color:" text box starting with a # sign, and set button is clicked. Then start game button is clicked.

Output: The snake changes into corresponding color in the new started game.

How test will be performed: The function that changes the snake's color will be executed after the set button is clicked and the change will be demonstrated in the next new game. Therefore when a new game is started, we will check if there is a change of the corresponding color to the snake.

3. STG-FR1-3

Type: Functional, Dynamic, Manual

Initial State: The SnakeGame.html file is open by the player.

Input: A wrong type of the color number is typed into the "Your Own Color:" text box(i.e. the length of the number is not 6, not starting with # or they are not hex numbers), and set button is clicked. Then start game button is clicked.

Output: A notification will shown to the player, telling that the type of the color input is wrong.

How test will be performed: The function that examines the color input will be executed after the set button is clicked, and if the input type is wrong, the variable for current snake's color will not be changed and a pop up will be shown to tell that. So when set button is clicked with wrong type input in "Your Own Color:" text box, we will check if there is a popup shown and the snake's color variable is kept unchanged.

3.1.1.2 FR2

4. STG-FR2-1

Type: Functional, Dynamic, Manual

Initial State: A new game gets started.

Input: A valid string the user want to use as the username. (Maximum length of username is 10).

Output: The username will be displayed on the right side of the game interface while playing the game.

How test will be performed: We will manually check if the username inputted is displayed on the screen.

5. STG-FR2-2

Type: Functional, Dynamic, Manual

Initial State: A new game gets started.

Input: A invalid string(the length is over 10) the user want to use as the username. (Maximum length of username is 10).

Output: A prompt saying "Invalid Username" pops out.

How test will be performed: We will manually check if there is a prompt saying "Invalid Username" pops out after inputting a string whose length is over 10.

3.1.1.3 FR3

6. STG-FR3-1

Type: Functional, Dynamic, Manual

Initial State: The SnakeGame.html file is open by the player.

Input: The "History" button is clicked.

Output: The corresponding "History.txt" file pops out.

How test will be performed: The latest history(including PlayerName and Score) are recorded in the "History.txt" file in the ./Record directory when 1 round of game ends. So when the "History" button is clicked, we will check if the "History.txt" file pops out.

7. STG-FR3-2

Type: Functional, Dynamic, Manual

Initial State: The SnakeGame.html file is open by the player and several games rounds have been played.

Input: The "History" button is clicked.

Output: The corresponding "History.txt" file pops out and the past records are displayed.

How test will be performed: The latest playing history(including PlayerName and Score) are recorded in the "History.txt" file in the ./Record directory when a round of game ends. So when the "History" button is clicked, we will check if the records displayed in the "History.txt" file are correct.

3.1.1.4 FR4

8. STG-FR4-1

Type: Functional, Dynamic, Manual

Initial State: The SnakeGame.html file is open by the player.

Input: N/A

Output: The rules of the game are displayed on the GUI.

How test will be performed: The rules for the game are always displayed on the GUI. So we will check if the rules are displayed when we open the game file.

3.1.1.5 FR5

9. STG-FR5-1

Type: Functional, Dynamic, Manual

Initial State: The SnakeGame.html file is opened by the player.

Input: N/A

Output: The user interface displays a short cartoon.

How test will be performed: We will manually check if the screen displays a short cartoon once the SnakeGame.html file is opened.

3.1.1.6 FR6

10. STG-FR6-1

    Type: Functional, Dynamic, Manual

    Initial State: The SnakeGame.html file is opened by the player.

    Input: User clicks the preset button for different difficulties, then clicks the start game button.

    Output: The snake game is performed based on the chosen difficulties.

    How test will be performed: The preset difficulty button will change the inside clock cycle refresh frequency variable to change the difficulties. We will manually check if the refresh frequency variable is modified correctly and the in-game screen is refreshed on that frequency.

3.1.2 Playing the game

3.1.2.1 FR7

11. PTG-FR7-1

    Type: Functional, Dynamic, Manual

    Initial State: The SnakeGame.html file is opened by the player.

    Input: The "start" button is clicked.

    Output: The gaming zone will be refreshed, a snake will be printed in a preset position and a supplement is printed in a random position.

    How test will be performed: After clicking the "start game" button, The function will initiate the gaming zone with a background, a snake in the preset position, and a supplement in a random position. We will check to see if the background and the snake can be printed properly, and a supplement is printed in a random position.

3.1.2.2 FR8

12. PTG-FR8-1

    Type: Functional, Dynamic, Manual

    Initial State: The SnakeGame has been started and in the current clock cycle, the game is not ended.

    Input: A direction key(i.e. Up, down, left, right) is pressed. The pressed direction key is in-line with the snake's current moving direction.

    Output: No direction changed will be made. Then snake's position will be reprinted in the next clock cycle.

    How test will be performed: If the pressed direction key is in-line with the snake's current moving direction, no direction changed will be made. If not, the snake's moving direction will be made same to the pressed key. Then snake's position will be reprinted in the next clock cycle. After clicking the direction button, The function will examine whether the direction variable needed to be changed. If so, it will store the change. Then it will refresh the following snake positions based on the variable. So we will check to see if the variable is kept unchanged and snake is moving with the original direction in the following clock cycles.

13. PTG-FR8-2

    Type: Functional, Dynamic, Manual

    Initial State: The SnakeGame has been started and in the current clock cycle, the game is not ended.

    Input: A direction key(i.e. Up, down, left, right) is pressed. The pressed direction key is not in-line with the snake's current moving direction.

    Output: The snake's moving direction will be made same to the pressed key. Then snake's position will be reprinted in the next clock cycle.

How test will be performed: If the pressed direction key is in-line with the snake's current moving direction, no direction changed will be made. If not, the snake's moving direction will be made same to the pressed key. Then snake's position will be reprinted in the next clock cycle. After clicking the direction button, The function will examine whether the direction variable needed to be changed. If so it will store the change. Then it will refresh the following snake positions based on the variable. So we will check to see if the variable is changed to the needed direction and snake is moving with the modified direction in the following clock cycles.

3.1.2.3 FR9

14. PTG-FR9-1

Type: Functional, Dynamic, Manual

Initial State: The SnakeGame has been started and in the current clock cycle, the game is not ended.

Input: N/A

Output: The snake's position and supplements' position will be re refreshed based on the stored clock cycle variable in the system.

How test will be performed: The gaming zone is refreshed based on a preset clock cycle, which is stored in clock cycle variable. We will start the game with different clock cycle to see if the game is refreshed based on that given frequency.

3.1.2.4 FR10

15. PTG-FR10-1

Type: Functional, Dynamic, Manual

Initial State: The SnakeGame has been started and in the current clock cycle, the game is not ended. The snake's head is next to the boundary of game zone, and in the snakes moving direction.

Input: In the current clock cycle, no direction key is pressed, or the pressed key direction is in-line with the snake current moving direction.

Output: The gaming zone will be frozen and a popup show up telling that the game is over with a score result.

How test will be performed: The function will check the moving direction variable, snake head block position and gaming zone boundary in the current clock cycle. If the snake will hit the boundary in next clock cycle and keep the moving direction unchanged, the function will freeze the gaming zone and have a popup shown to display the score result. So we will check to see whether there is popup shown with the correct score, and the gaming zone is frozen to prevent further modification if we control the snake to hit the boundary.

3.1.2.5 FR11

16. PTG-FR11-1

Type: Functional, Dynamic, Manual

Initial State: The SnakeGame has been started and in the current clock cycle, the game is not ended. The snake's head is next to a supplement, and in the snakes moving direction.

Input: In the current clock cycle, no direction key is pressed, or the pressed key direction is in-line with the snake current moving direction.

Output: The snake's position will cover the supplement block. The score, snake's length and speed variable will be changed according to the kind of supplement eaten by the snake.

How test will be performed: The function will check the moving direction variable, snake head block position and existing supplement block positions in the current clock cycle. If the snake will hit any supplement block in next clock cycle and keep the moving direction unchanged, the function will update the score variable, speed variable immediately, and snake's length in the following clock cycles according to the kind of supplement eaten by the snake. So we will check to see whether there is change to the relative variables, and the change is made under the rule we set for the game.

3.1.3 End the game

3.1.3.1 FR12

17. ETG-FR12-1

    Type: Functional, Dynamic, Automated

    Initial State: The game has ended.

    Input: N/A

    Output: Refreshing rate of the screen is 0

    How test will be performed: The function can be tested by using a navigator object that is supported by most modern browsers to assure the screen is not refreshing.

18. ETG-FR12-2

    Type: Functional, Dynamic, Manual

    Initial State: The game has ended.

    Input: N/A

    Output: Refreshing rate of the screen is 0

    How test will be performed: The function can be tested by adding a function which makes pixels flashing while the screen is refreshing. Manuel testing can be achieved by checking if the pixels are flashing.

3.1.3.2 FR13

19. ETG-FR13-1

    Type: Functional, Dynamic, Manual

    Initial State: The game has ended.

    Input: N/A

    Output: A prompt showing "Game Over" pops out.

How test will be performed: The function can only be tested by manual testing, by check if a prompt showing "Game Over" displayed on the screen.

3.1.3.3 FR14

20. ETG-FR14-1

    Type: Functional, Dynamic, Automated

    Initial State: The game has ended.

    Input: N/A

    Output: The score will be recorded in history.

    How test will be performed: A unit test will be used to test this function. The tester will go through the file that records history and check if the newest record is exactly for the game that just ended.

21. ETG-FR14-2

    Type: Functional, Dynamic, Manual

    Initial State: The game has ended.

    Input: N/A

    Output: The score will be recorded in history.

    How test will be performed: We manually test if the newest scores recorded and displayed on the screen.

3.1.4 During the game: snake eats a booster element

3.1.4.1 FR15

22. DTG-FR15-1

    Type: Functional, Dynamic, Manual

Initial State: A new game has been started, and the game is not ended in the current clock cycle.

Input: A blue supplement block is in the snake's heading direction, and is reached in the current clock cycle.

Output: In the next clock cycle, the snake's current speed is doubled, and 2 score is added into current score.

How test will be performed: The function that changes the snake's speed and score will be executed, and change will be demonstrated in the next clock cycle. We will check the inside variable that if there is a double change of the corresponding speed variable, and 2 score is added to the current score variable, and if the change is demonstrated in the next clock cycle.

3.1.4.2 FR16

23. DTG-FR16-1

Type: Functional, Dynamic, Manual

Initial State: A new game has been started, and the game is not ended in the current clock cycle.

Input: A red supplement block is in the snake's heading direction, and is reached in the current clock cycle.

Output: In the next 3 clock cycles, the snake's length is gaining 3 in each of the cycles, and 5 score is added into current score in the next clock cycle.

How test will be performed: The function that changes the snake's length will be executed in next 3 cycles, and the function that changes the score will be executed in the next clock cycle. The change will be demonstrated in the next 3 clock cycles. We will check the inside variable that if the snake's length is added by 1 in the next 3 clock cycles, and 5 score is added to the current score variable, and if the change is demonstrated in the next 3 clock cycles.

3.1.4.3 FR17

24. DTG-FR17-1

Type: Functional, Dynamic, Manual

Initial State: A new game has been started, and the game is not ended in the current clock cycle.

Input: A white supplement block is in the snake's heading direction, and is reached in the current clock cycle.

Output: In the next clock cycles, the snake's length is gaining 1, and 1 score is added into current score in the next clock cycle.

How test will be performed: The function that changes the snake's length will be executed in next cycle, and the function that changes the score will be executed in the clock cycle. The change will be demonstrated after the next clock cycle. We will check the inside variable that if the snake's length is added by 1, and 1 score is added to the current score variable, and if the change is demonstrated in the next clock cycle.

3.2 Non-functional Requirements

3.2.1 Look and Feel Requirements

25. LFR-NFR1-1

Type: Structural, Dynamic, Manual

Initial State: Program installed onto the system and a test group of 5 people is invited.

Input: Each tester opens SnakeGame.html and scales the GUI with points between 1 and 5.

Output: The average rating from test group is higher than 2.5.

How test will be performed: Each of the user in the test group will be provided with a questionnaire that includes the following:

(a) 1 Very Bad

(b) 2 Bad

(c) 3 So-so

(d) 4 Good

(e) 5 Very Good

The test group will be invited to rate the GUI of the program with the provided scale and the average rating point is calculated. The point must be greater than 2.5 to pass the test.

3.2.1.1 Performance Requirements

26. PR-NFR2-1

Type: Structural, Dynamic, Manual

Initial State: Program installed onto the system and the system built-in task manager application is also opened.

Input: User start playing the game

Output: The usage of the memory showed on the task manager do not exceed 100MB.

How test will be performed: The built-in tasks manager shows the status of the user's computer, including the RAM. So we will check if the computer's memory runs out as the game is running.

3.2.1.2 Cultural Requirements

27. CR-NFR7-1

Type: Structural, Dynamic, Manual

Initial State: Program installed onto the system and a test group of 5 people is invited.

Input: Each tester opens SnakeGame.html and scales the words used in the GUI with points between 1 and 5 after playing several rounds.

Output: The average rating from test group is higher than 4.

How test will be performed: Each of the user in the test group will be provided with a questionnaire that includes the following:

(a) 1 Many Offensive words (> 10)

(b) 2 Some Offensive Words ($[5, 10]$)

(c) 3 Few Offensive Words ($[0, 5]$)

(d) 4 No Offensive Words ($=0$)

The test group will be invited to rate the GUI of the program with the provided scale and the average rating point is calculated. The point must be greater than 4 to pass the test.

3.3 Traceability Between Test Cases and Requirements

Test cases are set up according to the FRs and NFRs mentioned in the SRS document. They are organized and numbered based on the order of original requirements for traceability convenience.

4 Tests for Proof of Concept

4.1 Background: Potential Risks

When developing the SnakeGame project, the following risks should be considered and be overcome when the implementation is completed:

1. The SankeGame is designed for running on computers that runs on different operation systems. Therefore a potential risk for this project is that the compatibility to different kinds of OS.

2. The game record is recorded in the "History.txt" file, therefore a risk that could fail the project is not supporting .txt file in some specific operation system.

4.2 Proof of Concept Test

4.2.0.1 Testing for OS Compatibility

1. POC-OSC-1

   Type: Functional, Dynamic, Manual

Initial State: Major OS(e.g. Windows 11, MacOS X, Ubuntu) are pre-installed.

Input: The SnakeGame.html file is open by the tester.

Output: The game is running properly.

How test will be performed: We will check the project's compatibility by running it on different kinds of OS. If the game is running as expect, this risk is overcome.

2. POC-OSC-2

   Type: Functional, Dynamic, Manual

   Initial State: Major OS(e.g. Windows 11, MacOS X, Ubuntu) are pre-installed and several rounds of game has been executed.

   Input: "History" button is clicked by the tester.

   Output: "History.txt" file is opened and pop-up properly.

   How test will be performed: This risk can be eliminated as long as the game history file is supported universally. So we will check if the "History.txt"(.txt file extension) is supported among different kinds of OS.

5 Comparison to Existing Implementation

The following are the test cases that compare the program to the existing implementation of the project:

1. test STG-FR3-1
2. test STG-FR2-2
3. test LFR-NFR1-1

6 Unit Testing Plan

6.1 Unit Testing of Internal Functions

JavaScript is used for implementing the internal functions of the game and the Mocha test framework will be used. Unit tests would be separated into a dedicated file named *SnakGame.test.js*, instead of sharing the same file with original codes. As for those functions that have specific inputs and outputs, we will create a series of unit tests to check if the outputs are the results we expect. To test those functions that do not have clear outputs, it is hard to identify if they are correct. We will match one of these functions with one of those with inputs and outputs and test them together to manifest the results. This game will not need any stubs or drivers to be imported specifically for testing. The coverage metrics will be automatically generated by Mocha, which would be used to measure and monitor our tests. Our goal is to test over 90% of our internal functions.

6.2 Unit Testing of User Interface

For this game, CSS and HTML are used for constructing the user interface. However, unit testing is not accessible and appropriate for the user interface since the web page of the game is hard to be modulized and testers are not able to automatically check if the web shows what we expect. Therefore, the approach we use for testing the user interface is manual testing rather than unit testing.

7 Appendix

N/A