



CUSTOM PROGRAM DESIGN REPORT

COS10009 INTRODUCTION TO PROGRAMMING



Name: Chan Kwang Yung

Student ID: 101215067

Program: Snake Game

Program's goal

This program is created to be a slightly more advance version of the classic snake game. Almost everyone on earth has heard of the classic snake game. The basics of the game generally consist of a snake or worm “eating” a food item, usually depicted as an apple. The end goal of the classic snake game is that the snake or worm has to move around the plain field chasing after an apple to get longer and longer until the snake’s head either collides with the border of the window or the snake collides with it. The goal of the program that I plan to make is to give the users a fun and exciting time while playing a classic world-renown game that has existed for decades

The first thing that came to mind when planning on the added features of the game is to make it multiplayer based. Therefore, the game will involve two players controlling two different coloured snakes in a competition to see who can eat the most apples. Apart from having a competitive multiplayer based mechanic, the game would also include some other interesting features that will make the players feel like they are playing a snake game on steroids. For example, the snakes will gradually become faster as they move and apart from spawning just an apple, the plain field will also be field with traps that will punish the player when the snake comes in contact with it. The game also includes the basic snake game structures such as the apple spawning in random areas, snakes spawning and moving in one direction unless a key input is inserted and the basic lose conditions such as if the snake hits its own body and if the snake goes over the boundaries.

The game will end when one of the players dies due to meeting the conditions of the lose condition or one of the player manage to accumulate 100 points from eating the apples.

User manual

This game requires 2 players. Player 1 will be controlling a green coloured snake and the controls of controlling it will be W for moving upwards, A for moving to the left side, S for moving downwards and D for moving to the right. Player 2 will be controlling a blue snake with the arrow keys as the direction controlling keys. For example, the up-arrow key will control the snake to move upwards and the other arrow keys will make the snake move corresponding to the direction the arrow is pointing at. The player can view the in-game instructions by holding down the button “I”.

Keyboard input	Use
Player 1	
W	UP
A	LEFT
S	DOWN
D	RIGHT
Player 2	
Up Arrow	UP
Left Arrow	LEFT
Down Arrow	DOWN
Right Arrow	RIGHT
General	
Escape	Close game window
Enter (Only during game ending screen)	Restart the game
I	Show instructions

The end goal of the game is to accumulate 100 points in order for the player to win. Another method the player can win is when the enemy snake either hit the borders of the game window or the snake came in contact with its own body. In order to gain points, the snakes will have to "eat" the apple that spawns randomly inside the window. Once the player comes in contact with the apple, their scores will increase by 10 points and the apple will respawn at another random location. The field will also be filled with 15 skulls to give an extra challenge to the players. If a player comes in contact with the skull, the player's snake will respawn and the player will have to start all over again with 0 points. To start the game, in the command prompt, the player will have to type in `cd mysnae` and then proceed to type `game.rb` to run the game. Once a player lost or a player won, the players have a choice to quit the game by pressing the Esc button or closing the window or the players can play again by pressing the Enter button.

Program design

The design of the program is similar to that of a classic snake game. It involves drawing three different types of screens which are the main game screen, the end game screen and the instructions screen. All the screen will have some sense of unity such as using the same background image and font type to provide some visual identity and so that the players will know they are still playing the same game.

Game screen

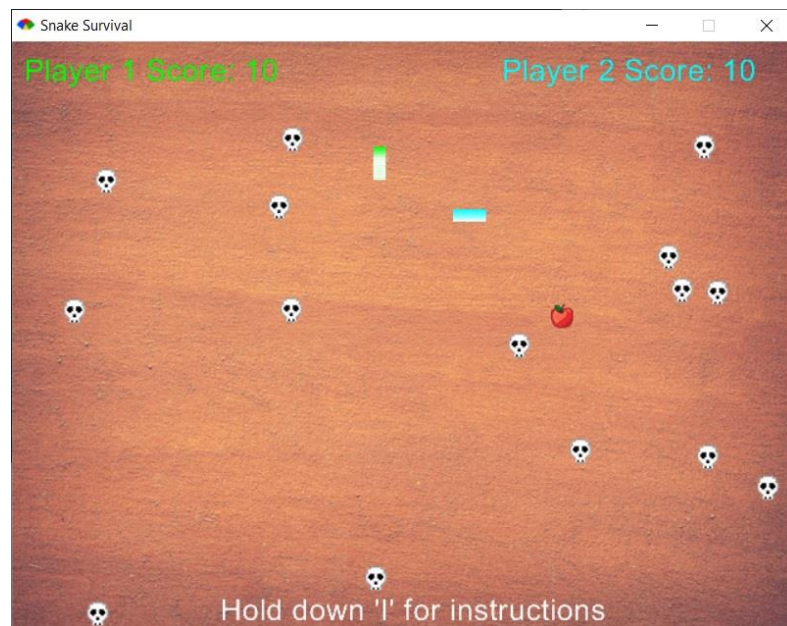


Figure 1 Game Screen

As depicted in figure 1, the snake game made will look like that where it consists of two green and blue coloured snakes chasing after an apple while in a field of skulls. The game screen will look very similarly to the classic snake game where it involves a snake, and an apple as shown in the figure below.

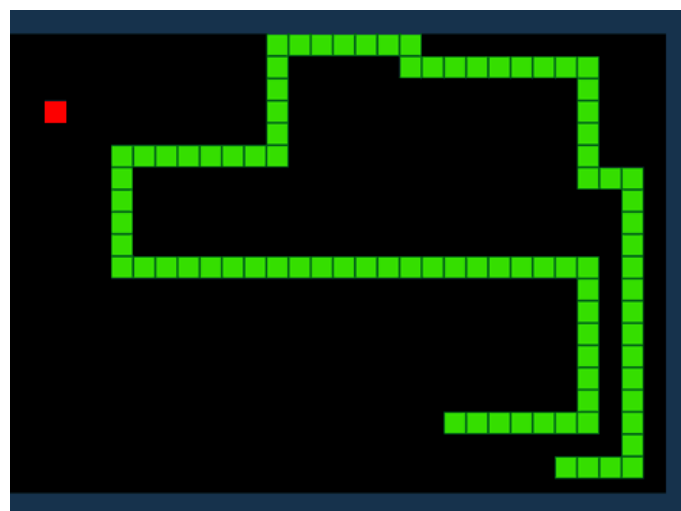
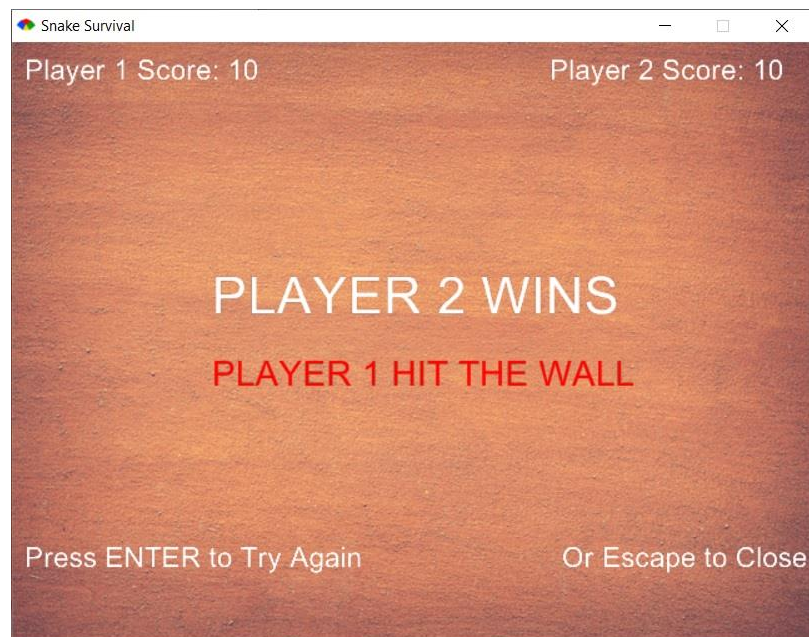
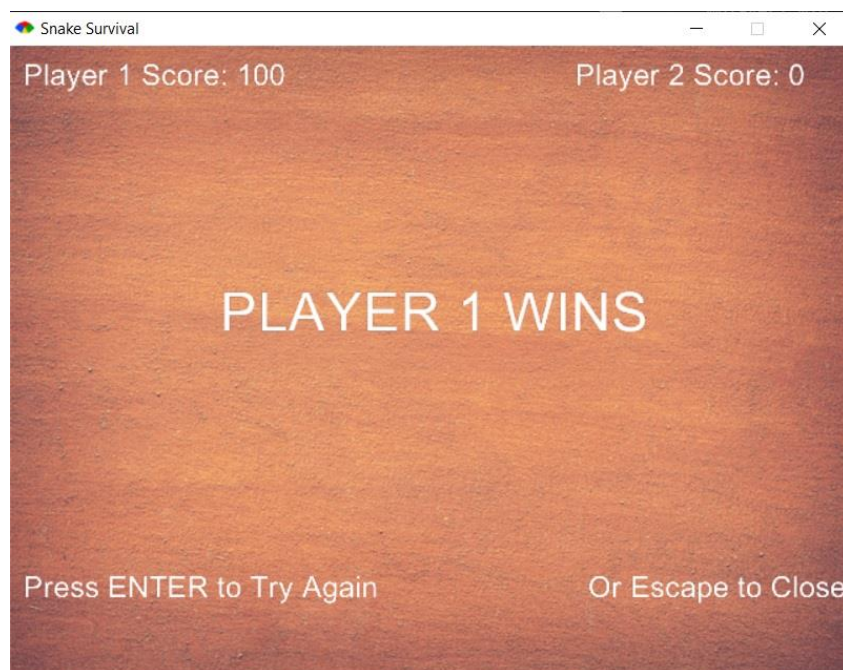


Figure 2 Classic Snake Game

End game screen

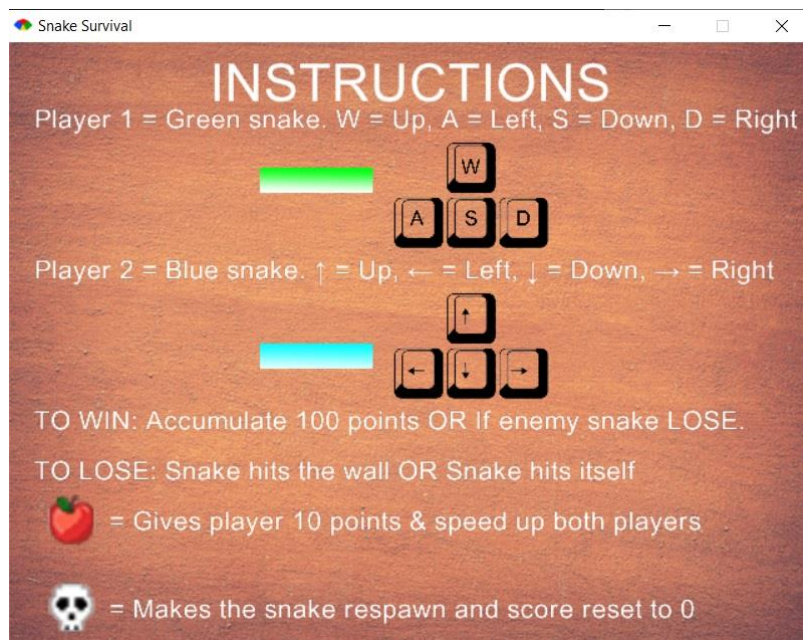


The end game screen will look something like the above image. There will be texts displaying on the screen showing which player won, the reason why the other player lost, the scores of the players and some text to tell what the player should do next for user-friendliness.



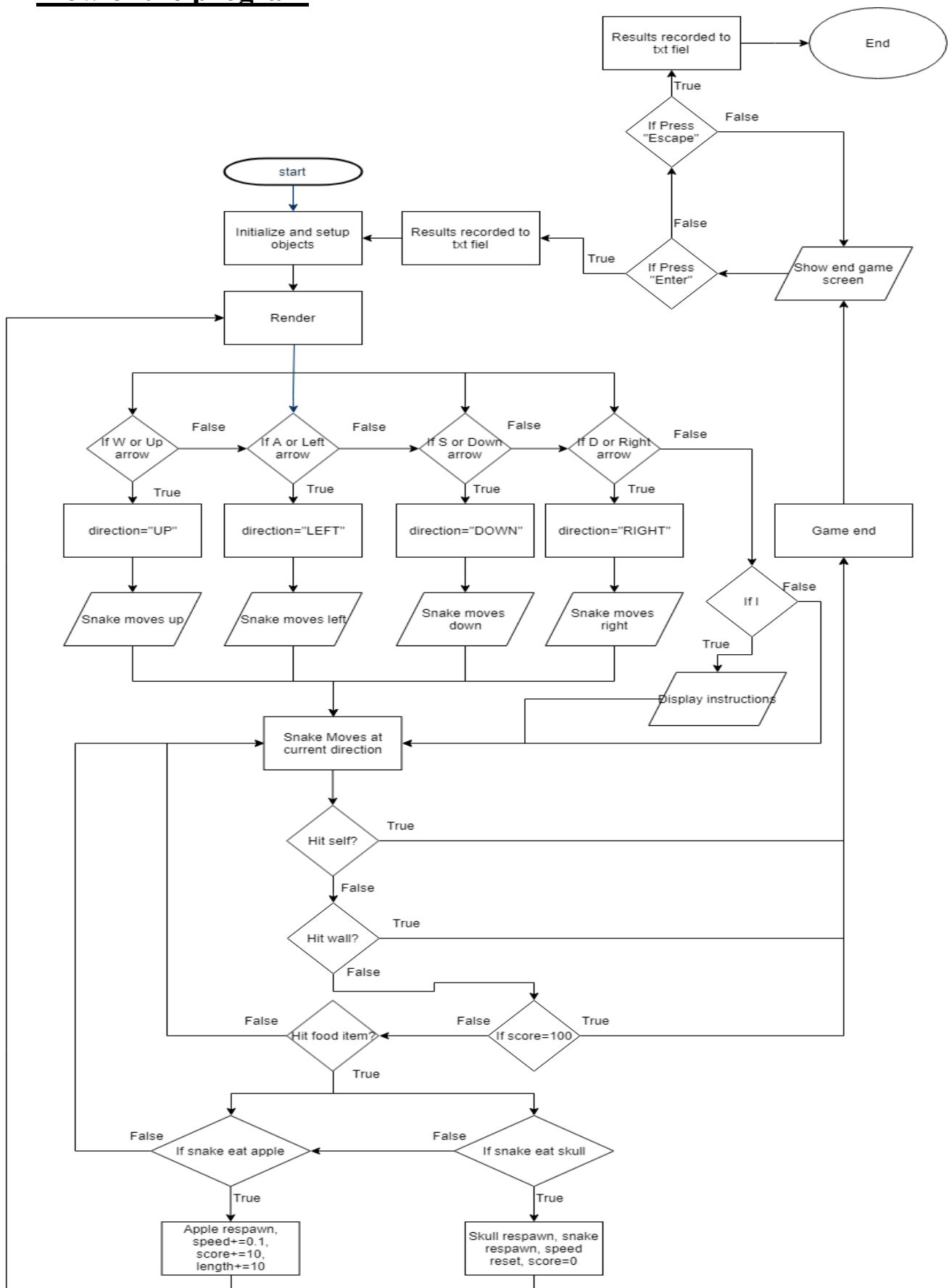
Another kind of end game screen will be drawn if one of the players manages to accumulate 100 points. This screen does not show the reason why the other player lost.

Instructions screen

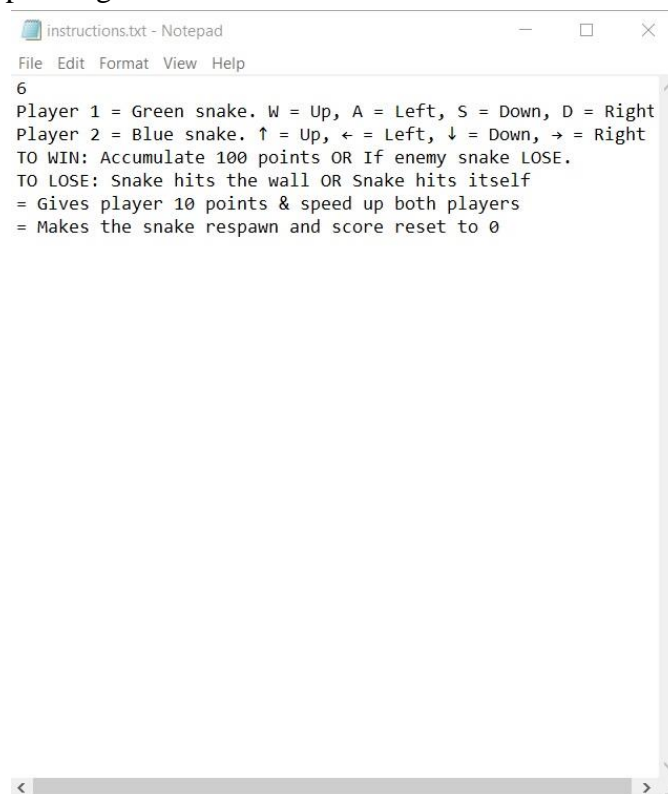


The instructions will be depicted as such where it tells the players the basics of the game. The texts are arranged neatly and images are used so that players can easily understand it.

Flow of the program



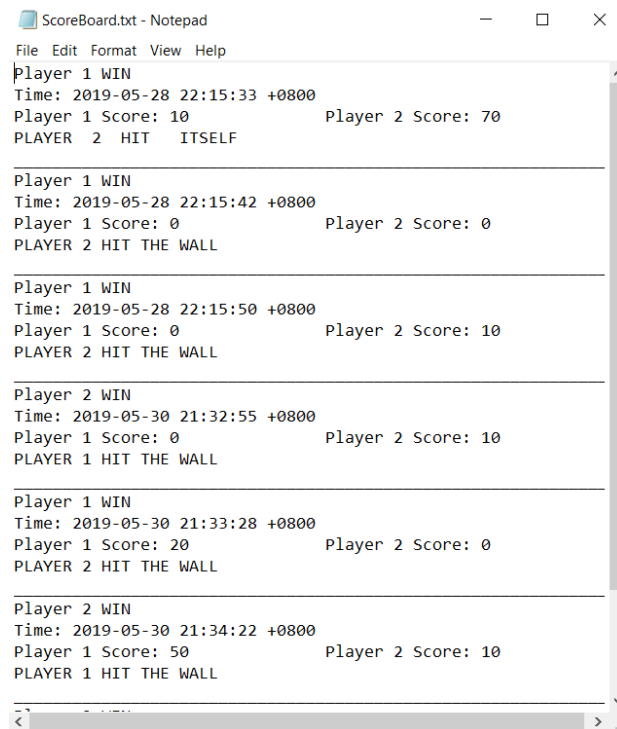
The game will start by initializing everything that is needed by the program to run. Next, it will render the codes so that objects and images are generated and displayed. Next, after everything is drawn and setup, the user will be able to see the GUI and the images. The player can then insert key inputs in order to control the snake and change its direction of movement. The snake on default will move only on the right direction, and can only change direction if the player inputs the appropriate key input to change its direction. For example, if the player one presses the “W” button, the snake will change the direction its heading to up. This applies to all the directional keys set by the programmer such as W, A, S, D, up arrow, left arrow, down arrow and left arrow keys. After the player has inputted the appropriate key inputs, the snake will change the direction and maintain that direction until another input is keyed in by the player. If the player press “I” instead, the programme will read from a text file and display the information in an instructions page. The user will have to hold down the “I” button to view the instructions instead of pressing it once.



```
6
Player 1 = Green snake. W = Up, A = Left, S = Down, D = Right
Player 2 = Blue snake. ↑ = Up, ← = Left, ↓ = Down, → = Right
TO WIN: Accumulate 100 points OR If enemy snake LOSE.
TO LOSE: Snake hits the wall OR Snake hits itself
= Gives player 10 points & speed up both players
= Makes the snake respawn and score reset to 0
```

3 Text file containing the instructions

Next, the program will consider the conditions that will assess the player to either a win or a loss. The first thing it would evaluate is that if the snake collides with its own body. If the evaluation returns a true, the game will end and an ending screen is then displayed. During this stage, the user has a choice to press enter to restart the game or to press another key input. If enter is pressed, the results from the game will be recorded into a text file and the game will reinitialize everything and start from scratch. If the user did not press enter, the user can also press the escape button. If the button is pressed, the results from the game will be recorded into a text file as well and the window will close and the programme ends, if not the game screen will keep on being drawn until either an enter button or escape button is pressed.



```
ScoreBoard.txt - Notepad
File Edit Format View Help
Player 1 WIN
Time: 2019-05-28 22:15:33 +0800
Player 1 Score: 10      Player 2 Score: 70
PLAYER 2 HIT ITSELF

Player 1 WIN
Time: 2019-05-28 22:15:42 +0800
Player 1 Score: 0      Player 2 Score: 0
PLAYER 2 HIT THE WALL

Player 1 WIN
Time: 2019-05-28 22:15:50 +0800
Player 1 Score: 0      Player 2 Score: 10
PLAYER 2 HIT THE WALL

Player 2 WIN
Time: 2019-05-30 21:32:55 +0800
Player 1 Score: 0      Player 2 Score: 10
PLAYER 1 HIT THE WALL

Player 1 WIN
Time: 2019-05-30 21:33:28 +0800
Player 1 Score: 20      Player 2 Score: 0
PLAYER 2 HIT THE WALL

Player 2 WIN
Time: 2019-05-30 21:34:22 +0800
Player 1 Score: 50      Player 2 Score: 10
PLAYER 1 HIT THE WALL
```

4Records of the game

The same goes for the next condition which is whether or not the snake's head collides with the borders of the window. If it is true, the game will end, if not the game will evaluate the next clause which is if the player's score reaches 100. If the player successfully accumulated up to 100 points, the game will end as well, or else the game will wait until either a snake dies or a snake manages to accumulate up to 100 points.

After that, the next evaluation that takes place is that when the snake has not win or lose, the snake can collide with food items such as apples or the skulls. If the snake did not collide with these food items, the game will loop the process of moving the snake and evaluating its win and lose conditions until it does collide with a food item. For the food items, if the snake collides with a skull, the player will be punished by having the score reset and the snake respawn on a different location. The snake too will be reset into its initial state when the game starts. If the snake did not collide with the skulls but instead with the apples, the player will be rewarded with a small speed boost on both players and a +10 in score and length for the player who ate the apple. If not, the programme will keep looping to where the snakes move until the one of the conditions are met. After the snake has collided with either of the food items, the food items will disappear and be rendered again at a different location. For when the snake eats the skulls, the snake itself will disappear and be rendered again at a different location. This is to create the effects of respawning objects.

Core program functionalities

Colliding with objects

This function is quite simple. In order to make the body of the snake, square blocks are placed into an array. This can be done by using `.push()`. For the snake to collide with objects head on, the first index in the array will be the head of the snake. This is so that the program can judge that only when the head of the snake collides with objects, some action will be taken. In this case, when the snake collides with food items such as the apple or the skulls, action will be taken. How the programme judges if the snake collides with the food item is by using `Gosu::distance()` which is used to calculate the distance between the coordinates of two objects. In this case, if the distance between the coordinates of the head of the snake and the coordinates of the food items is lesser than 14, a true will be returned. This will then be used in the main file to make the programme take action when the statement is true.

Snake moving

As mentioned before, the snake's body is made of an array of square blocks. The head of the snake acts as a pointer to where the snake is heading as well. The snake moves by constantly adding new blocks to the array by drawing the blocks and adding it in the array using `.push()`. The snake can maintain its length and no get longer as it moves due to the use of `.shift` which removes the first item in the array which in this cases would be the oldest block added to the snake's body. To move the snake to certain directions, square blocks are drawn according to the new coordinates. For example, if the snake moves left, new blocks will be drawn where the x-coordinate is increasing for each block drawn and appended into the array.

Lengthening of snake

The snake lengthens by having a ticker that decreases. For example, when a snake eats an apple, the ticker will be increased by 10 which means some time is given to stop the `.shift` method mentioned earlier from running up until the value of the ticker falls back to 0. This will in turn increase the length of the snake as new blocks are still being drawn constantly while the `.shift` method is temporarily muted.

Win or lose conditions

The game has 1 win condition and 2 lose conditions that applies to both players. The player is judged a winner if the player manages to accumulate up to 100 points without colliding with any skulls. The player is judged to lose when the snake's head either collides with the borders of the window or itself. This can be done by using if else statements. For how the programme knows when the snake collides with itself, `.pop()` method is used to remove the head segment of the snake and `Gosu::distance` is used again to evaluate if the distance between the head and the body is lesser than set, return true. For how the game evaluates when the snake collides with the border, if else statement is used to compare if the coordinates of the head of the snake is more than or less than the available coordinates of the window provided. For example, the window size used for this game is 480 in height and 640 in width, if the snake's head moves over the given window size, actions will be executed.

Data Dictionary

Field	Data type	Description
Class: Segment		
@window	Window	Used to enable object to be drawn inside of the window
@x	Array of integers	Used to index an array of integers
@y	Array of integers	Used to index an array of integers
Class: Snake		
@window	Window	Used to enable object to be drawn inside of the window
@x	Integer	X-coordinates of snake
@y	Integer	Y-coordinates of snake
@segments	Array	An array for the body of the snake
@direction	String	This string is used for the programmer to determine which direction the snake is facing so that the coordinates can change accordingly
@speed	Floating point	Used to determine the distance between each square block drawn in the body of the snake when it moves
@length	Integer	As a reference point to how long the snake is
@ticker	Integer	Used to control the length of the snake to either maintain its length or become longer
Class: Food		
@image	Object	Used to initialize an object to be drawn on the window
@x	Integer	X-coordinates of food
@y	Integer	Y-coordinates of food