Author: Baruch Chau

Date: July 19, 2022

How to Build:

The given CMAKE file can be used to build the project. Change directory to '/build' folder then run 'make' command. Run './SnakeGame' to run the game.

Description:

This is a modified two player snake game based on the given sample code provided by Udacity. The target score is entered at the beginning of the game in the command terminal, which then launches the window with the game. The goal of the game is to either eat enough food to reach the goal first, or eat the other snake. Eating yourself is still possible and if the two snake's heads touch, the game results in a tie. Whichever snake reaches the target score first or eats the other snake wins. At the end of the game, both player scores are displayed and the winning snake is shown.

Class Structure:

The 'Main' class is where the program begins. The game launches by first initializing the size and layout of the window. The user is then prompted to enter the target score for the players to achieve. Next the other classes which run the game, controller and graphics render are initialized. The main game loop is then run. At the end of the game, the final scores are displayed and based on the scores or status of the snake, the winner is decided.

The 'Game' class holds the game engine where the main game loop and required functions run from. The class is initialized with the grid size and snake players. In addition, initial positions and directions are given here to make the game fairer. In the Run function, a thread which runs the Check_Collision function that checks the snake's positions with respect to each other and monitors game ending events. When the main game loop is running, the linear procedure of looking for a key input by the user, update the snake positions, rendering the food, rendering snake1, rendering snake2 then clearing the screen is done. At the end of the renders, a frame rate calculation is done to determine at how many frames per second the entire process is running at. If the main game loop exits because of a game ending condition, the collision thread is stopped. As for the necessary functions, a PlaceFood function simply checks for available spots to place food targets for the snake. The Update function tracks the game progress and extends the game when applicable. The Check_Collision function monitors the possible live interactions that end the game such as the snake eating itself or eating another snake.

The 'Controller' class parses the keyboard inputs where the user controls the snake. There are no initializations required for this class. The 'HandleInput' function monitors the designated keyboard inputs for each player and assigns an enumerated value which correlates to the snake moving in the desired direction. Additionally, a 'ChangeDirection' function prevents the snake from moving backwards on itself if a body exists.

The 'Renderer' class handles rendering the different aspects of the game. When the renderer initializes, the window is created with the desired dimensions sent from the Main. At the end of the game, the deconstructor destroys the window. Previously the renderer had a single render function which renders the food, snake and clears the screen in a sequence. Now food and clear screen are rendered in their

own functions RenderFood and ClearScreen respectively for clarity and Render has become the RenderSnake function, dedicated to rendering the snake. Additionally, an UpdateWindowTitle function keeps the score updated.

The 'Snake' class creates an object which contains the physical attributes of each snake played in the game such as head position, body position, movement speed, and alive status, etc. When initialized, the size of the blocks are passed to the class. The functions within the Snake class are used to update and keep track of the member variables. When the user provides direction input, the head direction is updated. Additionally, when the body is updated constantly as it moves along the grid and the grid is checked to see if it consists of the snake body.

Rubric Points:

1. The project demonstrates an understanding of C++ functions and control structures.

    Main.cpp: Lines 46-63 - If conditions to check game winner

    Game.cpp: Lines 128-136 - Added second snake eating food condition and checking if either player's score has reached the target score

2. The project accepts user input and processes the input.

    Main.cpp: Lines 24-34 - Uses IOStream Cout and Cin to prompt user to enter target score

    Controller.cpp: Lines 48-65 - Added additional key inputs for second player

3. The project uses Object Oriented Programming techniques.

    Renderer.cpp: Lines 50-53 - Moved clear screen from render to its own function

    Renderer.cpp: Lines 55-69 - Moved food rendering to its own function

    Game.cpp: Lines 139-187 - Moved self-collision checking from Snake Class to a Game function. Expanded to including checking against collision of other snake in game.

4. Classes use appropriate access specifiers for class members.

    Snake.h: Line 43 - Added new variable to identify player as public so other classes can access state easily

    Game.h: Line 60 - Moved running to public so other classes can easily monitor if the game is stopped or running

5. All class members that are set to argument values are initialized through member initialization lists.

    Game.cpp: Line 26 - snake2 initialized in constructor

6. All class member functions document their effects, either through function names, comments, or formal documentation. Member functions do not change program state in undocumented ways.

    All classes: comments added to document function effects.

7. At least two variables are defined as references, or two functions use pass-by-reference in the project code.

Renderer.h: 23 - Changed the way the variables are passed to the function by using reference instead of value.

Renderer.cpp: 102-105 - Changed the way the variables are passed to the function by using reference instead of value.

Snake.h: 32 – Changed the way the variables are passed to the function by using reference instead of value.

Snake.cpp: 68-78 – Changed the way the variables are passed to the function by using reference instead of value.

8. The project uses multiple threads in the execution.

Game.cpp: Lines 52 and 88 - Check_Collision class is run parallel to the main game loop to monitor game ending situations.