Snake Game Idea

The game has two snakes trying to eat the food available. The more food the snakes eat the bigger and faster they get. The winning snake is the one that eats more food and grows faster and bigger.

Snakes Description and Control

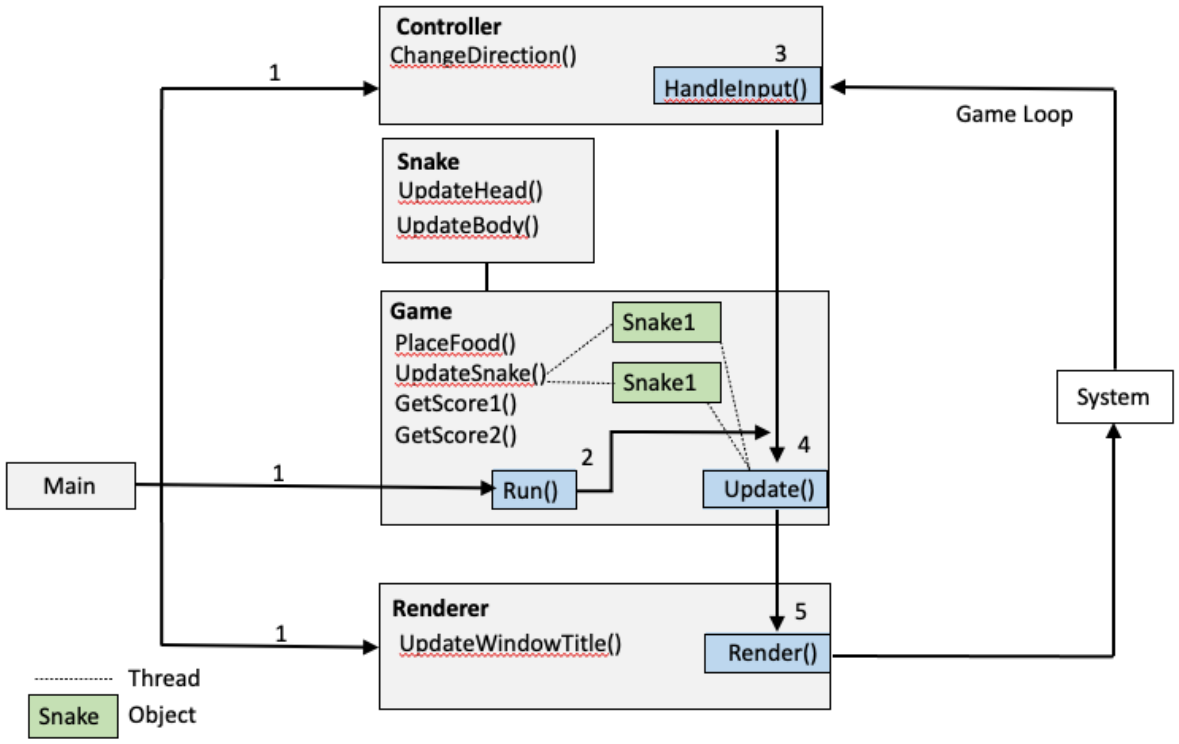
Snake1's head is colored green, and it is controlled with the following keys: upper arrow, lower arrow , left arrow, and right arrow respectively corresponding to up, down, left, and right.
Snake2's head is colored blue, and it is controlled with the following keys: s, d, e, and f respectively corresponding to left, right, up, and down.

Game Termination

The game terminates in two ways: gracefully and ungracefully. Graceful termination is when one of the snakes dies. This happens when the snake's body grows to the point that it steps into its body.
Ungraceful termination is when the one of the players hits control C or closes the game window. In both cases the winner will be announced on the terminal screen.

Code Structure

The graph below represents the code structure.



1- The Main function creates a controller, game, and a renderer objects.
2- The main calls Game::Run to start the game main loop of controller, update, and renderer.
3- The controller HandleInput() function uses ChangeDirection () function to get the keys inputs of both snakes.
4- Updating the game occurs inside game where the snakes objects are stored. For each snake a thread is created to update the snake in SnakeUpdate(). A promise and future are used to determine if the snake's body has grown or not.
5- Rendering the game happens in Render().

Build Instruction

1. Clone the repo.
2. For thread support please add the following line to the end of CMakeLists.txt
   set(CMAKE_CXX_FLAGS "-std=c++17 -pthread")
3. Make a build directory in the top-level directory:
   mkdir build && cd build
4. Compile it
   cmake .. && make
5. Run it

./SnakeGame

Dependencies for Running Locally

- cmake >= 3.7
  - All OSes: https://cmake.org/install/
- make >= 4.1 (Linux, Mac), 3.81 (Windows)
  - Linux: make is installed by default on most Linux distros
  - Mac: https://developer.apple.com/xcode/features/
  - Windows: http://gnuwin32.sourceforge.net/packages/make.htm)
- SDL2 >= 2.0
  - All installation instructions can be found in https://wiki.libsdl.org/Installation. Note that for Linux, an `apt` or `apt-get` installation is preferred to building from source.
- gcc/g++ >= 5.4
  - Linux: gcc / g++ is installed by default on most Linux distros
  - Mac: same deal as make - https://developer.apple.com/xcode/features/
  - Windows: recommend using MinGW

C++ Capstone Project Rubric

Criteria	Meets Specifications	Comments/ Examples
README		
A README with instructions is included with the project	The README is included with the project and has instructions for building/running the project. The README indicates which rubric points are addressed. The README also indicates where in the code (i.e., files and line numbers) that the rubric points are addressed.	README.pdf
The README indicates which project is chosen.	The README describes the project you have built. The README also indicates the file and class structure, along with the expected behavior or output of the program.	README.pdf
The README includes information about each rubric point addressed.	The README indicates which rubric points are addressed. The README also indicates where in the code (i.e., files and line numbers) that the rubric points are addressed.	README.pdf
Compiling and Testing (All Rubric Points REQUIRED)		
The submission must compile and run.	The project code must compile and run without errors. We strongly recommend using cmake and make, as provided in the starter repos. If you choose another build system, the code must compile on any reviewer platform.	Done. Steps: 1- cmake .. &&make 2-./Snakegame
Loops, Functions, I/O		
The project demonstrates an understanding of C++ functions and control structures.	A variety of control structures are used in the project. The project code is clearly organized into functions.	game.cpp Lines 136 to 158

The project reads data from a file and process the data, or the program writes data to a file.	The project reads data from an external file or writes data to a file as part of the necessary operation of the program.	game.cpp Lines 107 to 111 Lines 142 to 154
Object Oriented Programming		
The project uses Object Oriented Programming techniques.	The project code is organized into classes with class attributes to hold the data, and class methods to perform tasks.	Example Snake class which contains snake attributes and class methods such as GrowBody().
Classes use appropriate access specifiers for class members.	All class data members are explicitly specified as public, protected, or private.	Done in the header files
Class constructors utilize member initialization lists.	All class members that are set to argument values are initialized through member initialization lists.	renderer.cpp Lines 5 to 12 game.cpp Lines 14 to 19 snake.cpp Lines 5 to 17
Classes abstract implementation details from their interfaces.	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.	Done
Memory Management		
The project makes use of references in function declarations.	At least two variables are defined as references, or two functions use pass-by-reference in the project code.	controller.cpp Line 6 Line 15
Concurrency		
The project uses multithreading.	The project uses multiple threads in the execution	game.cpp Lines 91 to 113
A promise and future is used in the project.	A promise and future is used to pass data from a worker thread to a parent thread in the project code.	game.cpp Lines 91 to 113 Lines 117 to 134
A mutex or lock is used in the project.	A mutex or lock (e.g. std::lock_guard or `std::unique_lock) is used to protect data that is shared across multiple threads in the project code.	game.cpp Lines 123 to 133