**CPPND: Capstone Snake Game**

**Dependencies for Running Locally**

1. cmake >= 3.7
   * All OSes: [click here for installation instructions](https://cmake.org/install/)
2. make >= 4.1 (Linux, Mac), 3.81 (Windows)
   * Linux: make is installed by default on most Linux distros
   * Mac: [install Xcode command line tools to get make](https://developer.apple.com/xcode/features/)
   * Windows: [Click here for installation instructions](http://gnuwin32.sourceforge.net/packages/make.htm)

3. SDL2 >= 2.0

* All installation instructions can be found [here](https://wiki.libsdl.org/Installation)
* Note that for Linux, an `apt` or `apt-get` installation is preferred to building from source.

4. gcc/g++ >= 5.4

* Linux: gcc / g++ is installed by default on most Linux distros
* Mac: same deal as make - [install Xcode command line tools](https://developer.apple.com/xcode/features/)
* Windows: recommend using [MinGW](<http://www.mingw.org/>)

**Basic Build Instructions**

1. Clone this repo.

2. Make a build directory in the top level directory: `mkdir build && cd build`

3. Compile: `cmake .. && make`

4. Run it: `./SnakeGame`.

**Functionality**

In this project, the purpose is to create two snakes in the snake game instead of one. This is done so that two players can play at the same time. The input controllers to the snake game are mouse roller for one snake and keyboard for the other. Both snakes results will be updated in the window to see the final winner. When you run the project you should see two snakes moving with a cell of food. One snake can be controlled by keyboard and other with mouse roller. You should be able to see the updated scores of both the scores in the window. The results are also available in output.txt.

**Classes**

1. **main.cpp** now reads the user input from the input file to initialize the game and outputs the result to the output file.
2. **game.h** and **game.cpp**. This class creates a shared pointer for two snake object creations. It runs the game where both the snakes grows their body after eating the food
3. **snake.h** and **snake.cpp**. In this class, functions are used to update the snake’s body and head.
4. **Rendrer.cpp and renderer.h.** In this class, the windows, the window title, two snakes and food are created to be displayed on the
5. screen

**Rubric Points Addressed**

**Loops, Functions, I/O**

| **CRITERIA** | **MEETS SPECIFICATIONS** |
| --- | --- |
| The project reads data from a file and process the data, or the program writes data to a file. | File – main.cpp  Lines from 13 to 68  Instead of the static constant expressions, I used these parameters to input from a file input.txt. I also output into another file output.txt. |
| The project accepts user input and processes the input. | File – main.cpp  Lines from 13 to 68  The project accepts user input and processes the input from the input file. |

**Object Oriented Programming**

| **CRITERIA** | **MEETS SPECIFICATIONS** |
| --- | --- |
| Overloaded functions allow the same function to operate on different parameters. | File – renderer.cpp  Lines from 92 to 96  Function overloading is used in function UpdateWindowTitle(), to update a different title on the window |
| Derived class functions override virtual base class functions. | File – controller.cpp, controller.h  Lines from 46 to 79, 7 to 25  Created a virtual function HandleInput() in the base class Controller so that input can be handled either through keyboard or mouse roller |
| Templates generalize functions in the project. | File – snake.h  Lines from 22 to 35  Template is used in function SnakeCell() so that it can take different data types for the measurement of the body of the snake cells |

**Memory Management**

| **CRITERIA** | **MEETS SPECIFICATIONS** |
| --- | --- |
| The project uses smart pointers instead of raw pointers. | File – game.h  Lines from 24  The two snakes object created are shared pointer objects. |

**Concurrency**

| **CRITERIA** | **MEETS SPECIFICATIONS** |
| --- | --- |
| The project uses multithreading. | File – game.cpp  Lines from 41-42  The function Update() in game is made a thread, which runs parallelly to the main() function. This is created as a thread because both the snakes will access this function anytime. |
| A promise and future is used in the project. | File – game.cpp  Lines from 41-42  Std::async is used for thread creation which has an underlying promise and future implementation |
| A mutex or lock is used in the project. | File – game.cpp  Line 100  Mutex lock\_guard is used in Update() function which is a thread, so that the snakes don’t try to access the food at the same time. |