UDACITY

< Return to Classroom

C++ Capstone Project

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear student

You did a great job on this project 👍 and your project meets all the specifications.

- Well done for adding another snake to the game and using multi-threading.
- Great work for displaying each player's score at the end of the game.
- Moreover, your `readme` is so clear 👌

About using multi-threading:

- The best way to implement your idea for controlling snakes by multi-threads is to use the thread communications technique
- The same technique you have learned in the traffic simulation program
- You can do it by creating a generic message class called `message` and making the controller thread notify the main thread when it has an update with a player movement.

About creating two players:

- There is nothing wrong you did here.
- You can also create an enum inside it snake1 and snake2 to make your code more compact.

Some suggestions:

- You can allow users to enter their names and save them in a text file with their scores, so you can get the top score.

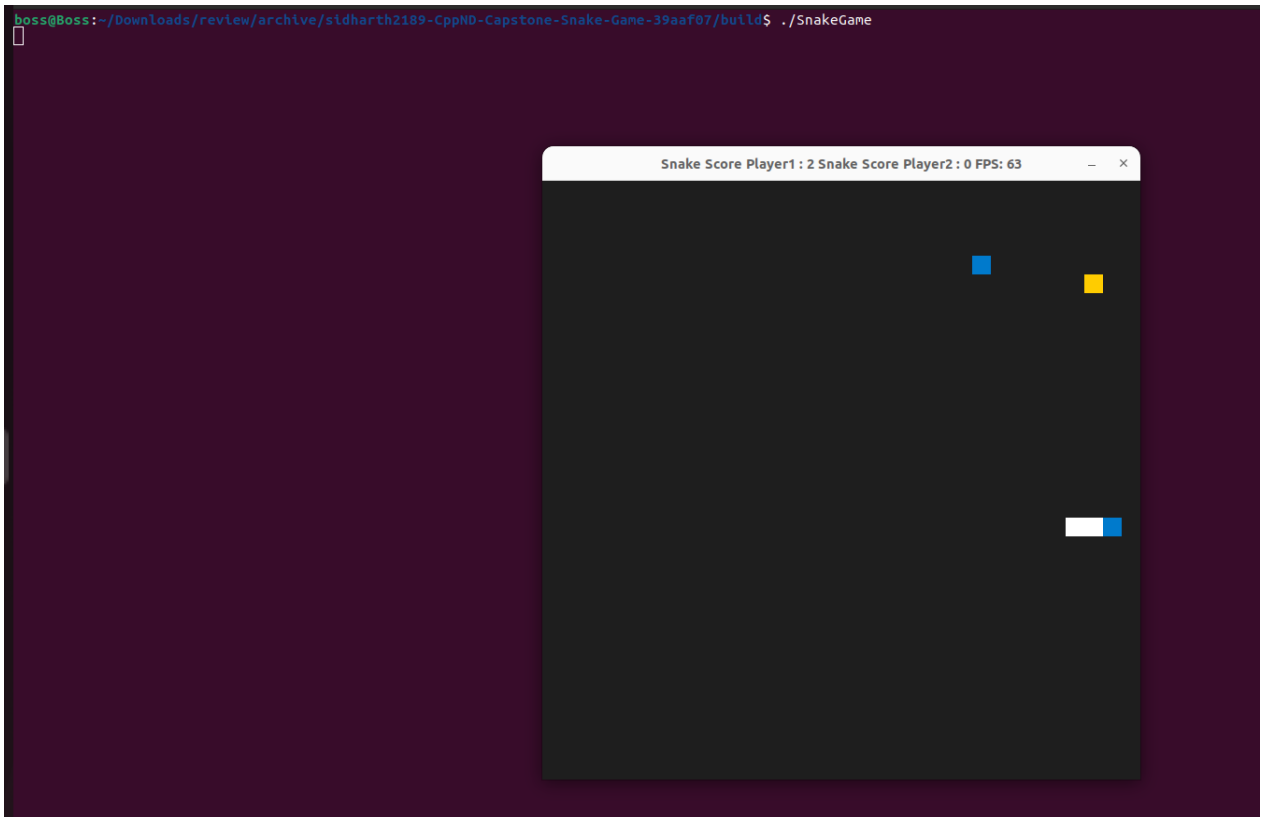
- You can add some obstacles to the game.
- You can allow the user to select the initial speed (Slow - Fast).
- You can add **another snake** to the game and allow the computer to control it using **A* search algorithm**.

Here are some useful links related to the topics covered in this course

- Check this link for more information About the README file
- Standard library is an amazing source for many common used method, Don't miss checking them out.
- Check out this link to know some useful C++ tricks for competitive programming.
- Follow this link to learn some advanced C++ techniques.

Don't forget to rate my work as a project reviewer! Your detailed feedback is very helpful and appreciated - thank you!
Note that there is no code review section for this project on the Udacity review page

Wish you all the best and success



README (All Rubric Points REQUIRED)

The README is included with the project and has instructions for building/running the project.

If any additional libraries are needed to run the project, these are indicated with cross-platform installation instructions.

You can submit your writeup as markdown or pdf.

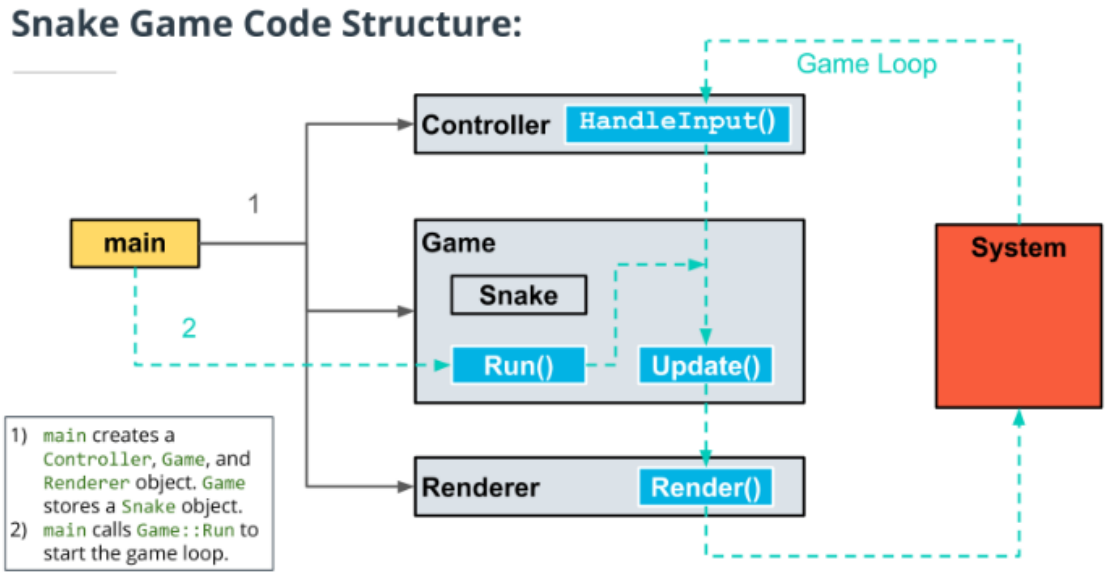
- The project contains a clear instructions for **building and running** the project
- Also you can check this link for more information about writing **a good** `readme` file

```
## Dependencies for Running Locally
* cmake >= 3.7
  * All OSes: [click here for installation instructions](https://cmake.org/install/)
* 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)
* SDL2 >= 2.0
  * All installation instructions can be found [here](https://wiki.libsdl.org/Installation)
  * [SDL Wiki](https://wiki.libsdl.org/APIByCategory)
  >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 - [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`.
```

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.

- The description of the project was **so clear** 👍
- Well done for drawing a diagram for the project structure
- You can also learn more about how you can draw *a diagram* for your project from this page



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.

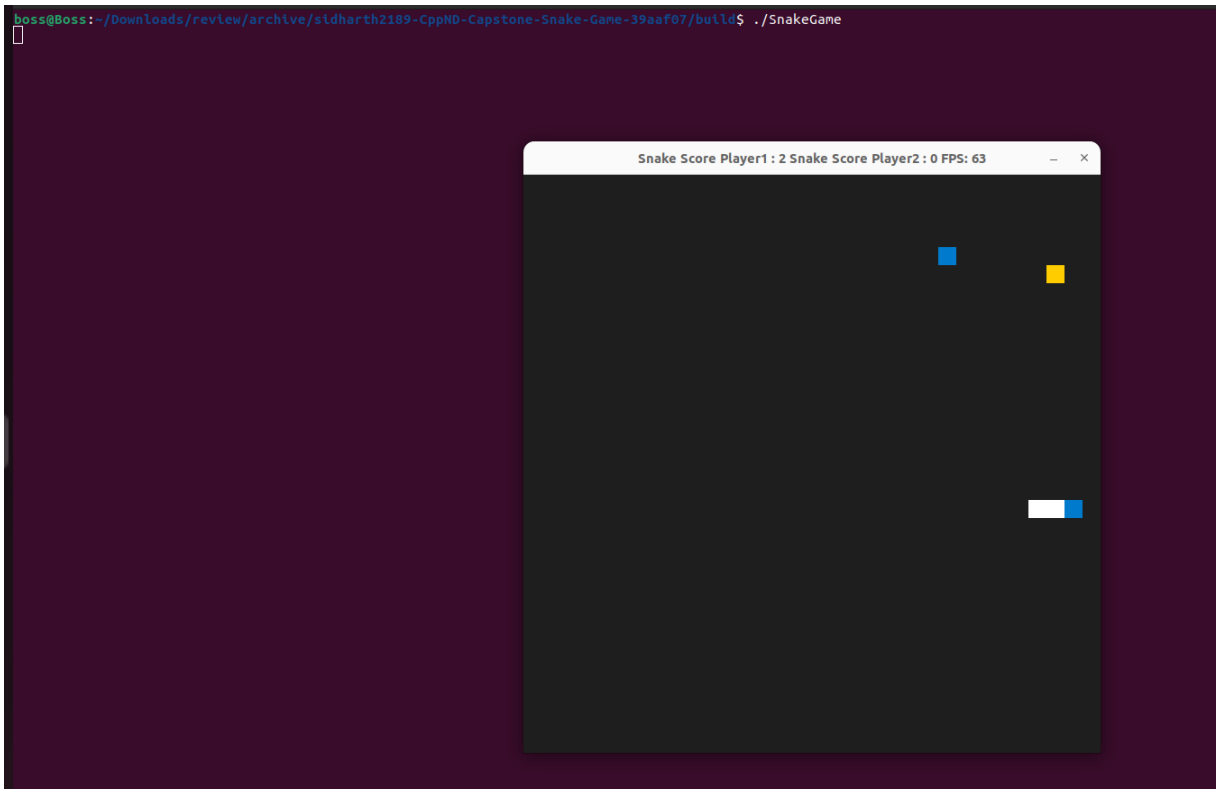
You already implemented **more than 5 rubric** points. Well done 👍

Compiling and Testing (All Rubric Points REQUIRED)

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.

- Your code compiles without any errors 👍
- Here are some useful links for the Make file topic, you will need it entire life being a software engineer!
    - How to Build a CMake-Based Project
    - Introduction to CMake by Example
    - Cmake tutorial



Loops, Functions, I/O

A variety of control structures are used in the project.

The project code is clearly organized into functions.

- Control Structures are just a way to specify flow of control in programs.
- Any algorithm or program can be more clear and understood if they use self-contained modules called as logic or control structures.
- It basically analyzes and chooses in which direction a program flows based on certain parameters or conditions.

- There are three basic types of logic, or flow of control, known as:
    - Sequence logic, or sequential flow
    - Selection logic, or conditional flow
    - Iteration logic, or repetitive flow

- For more information, you can check this link

```
if (e.type == SDL_QUIT) {
  std::lock_guard<std::mutex> lck(_mtxController);
  running = false;
} else if (e.type == SDL_KEYDOWN) {
  if (e.key.keysym.sym == _up) {
    ChangeDirection(snake, Snake::Direction::kUp,
                    Snake::Direction::kDown);
  }

  else if (e.key.keysym.sym == _down) {
    ChangeDirection(snake, Snake::Direction::kDown,
                    Snake::Direction::kUp);
  }

  else if (e.key.keysym.sym == _left) {
    ChangeDirection(snake, Snake::Direction::kLeft,
                    Snake::Direction::kRight);
  }

  else {
    ChangeDirection(snake, Snake::Direction::kRight,
                    Snake::Direction::kLeft);
  }
}
```

The project reads data from an external file or writes data to a file as part of the necessary operation of the program.

You can allow users to **enter their names** and **save it in a text file** with their scores, so you can get **the top score.**

- You can capture the date too.
- Check this link for more information about how you could do that

The project accepts input from a user as part of the necessary operation of the program.

You can allow the user to select the initial speed `(Slow - Fast).`

Object Oriented Programming

Object Oriented Programming

The project code is organized into classes with class attributes to hold the data, and class methods to perform tasks.

Well done for **structuring** your project into **classes**

All class data members are explicitly specified as public, protected, or private.

- **Public and private** access modifiers were used.
- You can check this link for more information about their usages.

```
class Controller {
 public:
  Controller (SDL_Keycode up, SDL_Keycode down, SDL_Keycode left, SDL_Keycode right):
  _up(up),
  _down(down),
  _left(left),
  _right(right) {}
  void HandleInput(bool &running, Snake &snake) const;

 private:
  void ChangeDirection(Snake &snake, Snake::Direction input,
                       Snake::Direction opposite) const;
  SDL_Keycode _up;
  SDL_Keycode _down;
  SDL_Keycode _left;
  SDL_Keycode _right;
  static std::mutex _mtxController; // useful to lock the shared resource "runninh"
};
```

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

- The initialization list is used for
  1) For initialization of non-static const data members
  2) For initialization of reference members
  3) For initialization of member objects which do not have a default constructor
  4) For initialization of base class members
  5) When the constructor's parameter name is the same as the data member
  6) For Performance reasons
- Check this link to know more about **When do we use Initializer List in C++?**

```
Game::Game(std::size_t grid_width, std::size_t grid_height)
    : snake_1(grid_width, grid_height),
      snake_2(grid_width, grid_height),
      engine(dev()),
      random_w(0, static_cast<int>(grid_width - 1)),
      random_h(0, static_cast<int>(grid_height - 1)) {
  // initialize position of snake 2
  while (snake_2.head_x == snake_1.head_x && snake_2.head_y == snake_1.head_y)
  {
    snake_2.head_x = random_w(engine); // random start point (x) of snake 2
    snake_2.head_y = random_h(engine); // random start point (y) of snake 2
  }
```

```
  PlaceFood();
}
```

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.

- Documenting projects is one of the most important topics in C++
- It helps others to understand your work and build on top of it
- Check this link to learn more about this topic

Appropriate data and functions are grouped into classes. Member data that is subject to an invariant is hidden from the user. State is accessed via member functions.

Inheritance hierarchies are logical. Composition is used instead of inheritance when appropriate. Abstract classes are composed of pure virtual functions. Override functions are specified.

- This point was not addressed in the project.
- Check this link for more information about **Why and when to use inheritance?**

One function is overloaded with different signatures for the same function name.

- C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.
- This link contains a brief introduction about that topic.

One member function in an inherited class overrides a virtual base class member function.

- **A virtual function** is a member function that you expect to be redefined in derived classes.
- When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.
- You can save this link in your notes to come back to it whenever you want to revise this topic quickly

One function is declared with a template that allows it to accept a generic parameter.

- This point was not addressed in the project.
- Check this link for more information about **templates**

- Check this link for more information about **templates**

Memory Management

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

- Pass by reference was used in the project.
- Check this link for more information about the difference between the differences between pass by reference and pass by values

```
void Snake::UpdateBody(SDL_Point &current_head_cell, SDL_Point &prev_head_cell, Snake const &otherSnake) {
  // Add previous head location to vector
  body.push_back(prev_head_cell);

  if (!growing) {
    // Remove the tail from the vector.
    body.erase(body.begin());
  } else {
    growing = false;
    size++;
  }
```

At least one class that uses unmanaged dynamically allocated memory, along with any class that otherwise needs to modify state upon the termination of an object, uses a destructor.

- Allows remembering any `new` operator in your project must have a `delete` operator encounter to avoid memory leaking
- At the end of any class, check if you allocated memory in the heap
- If **YES**, free this memory in the **class destructor** using the `delete` operator

The project follows the Resource Acquisition Is Initialization pattern where appropriate, by allocating objects at compile-time, initializing objects when they are declared, and utilizing scope to ensure their automatic destruction.

If you still don't know what is the meaning of **RAII**, check this discussion

For all classes, if any one of the copy constructor, copy assignment operator, move constructor, move assignment operator, and destructor are defined, then all of these functions are defined.

Quick revision for the the difference between the shallow and deep copying

- There are two different types of copying
- The first is the **shallow** copying, which is commonly used in the **move** operator method.
  EX: `_image = source._image;`
- The second is a **deep** copying, which is commonly used for copying the memory in the **heap** in the

copy operator method.
Ex: `_image = new wxBitmap(*source._image);`

For classes with move constructors, the project returns objects of that class by value, and relies on the move constructor, instead of copying the object.

- Move semantics is about transferring resources rather than copying them when nobody needs the source value anymore.
- Check this link to learn more about **Why Move Constructors are used?**

The project uses at least one smart pointer: `unique_ptr`, `shared_ptr`, or `weak_ptr`. The project does not use raw pointers.

- A smart pointer is a class that wraps a 'raw' (or 'bare') C++ pointer, to manage the lifetime of the object being pointed to.
- Smart pointers should be preferred over raw pointers.
- If you feel you need to use pointers, you would normally want to use a smart pointer as this can alleviate many of the problems with raw pointers, mainly forgetting to delete the object and leaking memory.

Concurrency

The project uses multiple threads in the execution.

Well done for using multi-threads to control the two snakes and update the game

```
futures.emplace_back(std::async(&Controller::HandleInput, player1, std::ref(running), std::ref(snake_1)));
futures.emplace_back(std::async(&Controller::HandleInput, player2, std::ref(running), std::ref(snake_2)));
futures.emplace_back(std::async(&Game::Update, this));
std::for_each(futures.begin(), futures.end(), [](std::future<void> &ftr) {ftr.wait();});
```

- **std::thread** is the thread class that represents a single thread in C++.
- To start a thread we simply need to create a new thread object and pass the executing code to be called (i.e, a callable object) into the constructor of the object.
- Once the object is created a new thread is launched which will execute the code specified in callable.
- Check these Tutorials, it contains some useful examples

A promise and future is used to pass data from a worker thread to a parent thread in the project code.

- There is a difference between **promise and future**
- **A promise:** is used to set a value, a notification or an exception.
- **A future:** is used to pick up the value from the promise.

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.

You can check this link to know more about the difference between `std::lock_guard` and `std::unique_lock`

A `std::condition_variable` is used in the project code to synchronize thread execution.

The **condition_variable** class is a synchronization primitive that can be used to block a thread, or multiple threads at the same time, until another thread both modifies a shared variable (the condition), and notifies the condition_variable.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START