Project Report

➔ Instructions For Building/Running

➔ cmake >= 3.7
- All OSes: click here for installation instructions

➔ 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
- Windows: Click here for installation instructions

➔ 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
- Windows: recommend using MinGW

➔ SDL2 >= 2.0
- Linux: Install using `apt-get` is preferred to building from source.

```
➔   sudo apt-get update
➔   sudo apt-get -y install libsdl2-dev
➔   sudo apt-get -y install libsdl2-ttf-dev
➔   sudo apt-get -y install libsdl2-image-dev

These files are added to .student_bashrc, so it will be automatically configured with
project.
```

- Mac: Install using `homebrew` is preferred.

```
➔   brew install sdl2
➔   brew install sdl2_ttf
➔   brew install sdl2_image
```

- For other platforms, the instructions can be found here

➔ Project Description.

In this project, the functionality is enhanced in term of graphics and also added new strategy that is as follows:, after predefined time , random number of obstacle will appear on the fields, so player have to avoid those obstacle to complete the game. Moreover, initially player would have three lives with the collision the lives will be decremented. Also, there is also other way in which the snake will be dead that is that if he eat itself. In the end, sdl library that support images and text is employed to accomplish the task.

➔ Build Instruction

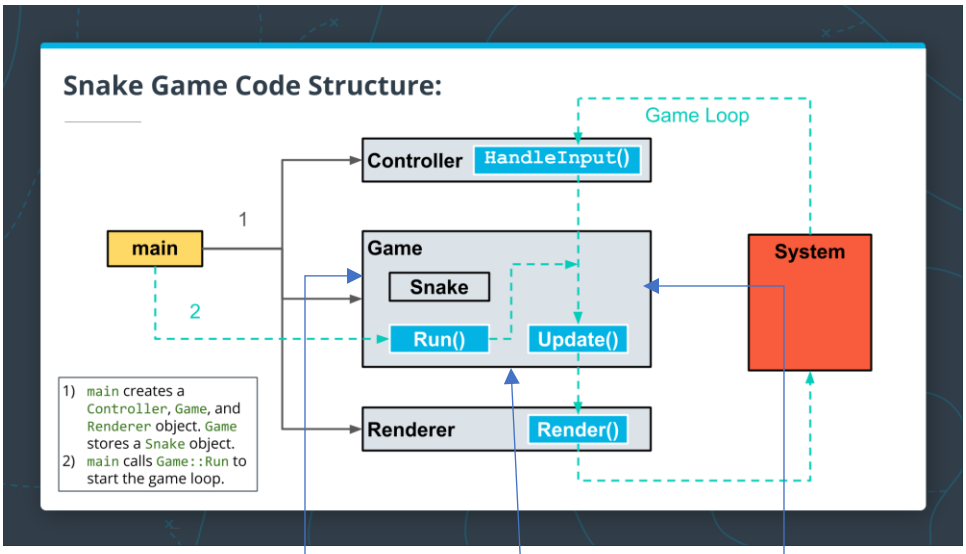
➔ Clone this repo.

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

➔ Compile: `cmake .. && make`

➔ Run it: `./SnakeGame`.

➔ Code Structure



Food
- ➔ Type
- ➔ weightage

Obstacle
- ➔ CheckPos()
- ➔ Panelity()

Bonus
- ➔ Active(bool)
- ➔ Point(x,y)
- ➔ Interval()

➔ Loops, Functions, I/O

```
void Game::Run(Controller const &controller, Renderer &renderer,
               std::size_t target_frame_duration) {
  Uint32 title_timestamp = SDL_GetTicks();
  Uint32 frame_start;
  Uint32 frame_end;
  Uint32 frame_duration;
  long frame_count = 0;
  bool running = true;
  bonus_timer_start = SDL_GetTicks();
  std::vector<obstacle> Obstacles;
  while (running) {
    frame_start = SDL_GetTicks();

    // Input, Update, Render - the main game loop.
    controller.HandleInput(running, snake);
    Uint32 elapsed_time;
          Uint32 bonus_timer_end = SDL_GetTicks();
          elapsed_time =  bonus_timer_end - bonus_timer_start;
          // Place bonus every `interval` seconds
          if (bonus_timer_end - bonus_timer_start >= bonus_interval * 1000) {
```

➔ Object Oriented Programming

Map Header File(Class)

```
#pragma once
#include <string>
#include <map>
// #include "SDL_mixer.h"
#include "SDL_image.h"
#include "SDL_ttf.h"
#include "SDL.h"


class Tilemap {
public:
    static Tilemap* instance() {
    if (_instance == nullptr) {
        _instance = new Tilemap();
    }
    return _instance;
    }

    void init(SDL_Renderer* renderer, int tile_w, int tile_h);
    bool addTile(std::string file_path, std::string id);
    /**
     * Draw single tile at x, y
     */
    void render(std::string id, int x, int y);
    /**
     * Fill rectangle with looping tile
     */
    void fillWith(std::string id, int x, int y, int w, int h);
    void clean();

private:
    static Tilemap* _instance;
    Tilemap() {}
    ~Tilemap() {}
```

Implementation of Map Class:

```
#include "Map.h"
#include <iostream>

Tilemap* Tilemap::_instance = nullptr;

void Tilemap::init(SDL_Renderer* renderer, int tile_w, int tile_h) {
  this->renderer = renderer;
  this->tile_w = tile_w;
  this->tile_h = tile_h;
}

bool Tilemap::addTile(std::string file_path, std::string id) {
  SDL_Surface* temp_surface = IMG_Load(file_path.c_str());

  if (temp_surface == 0) {
    std::cout << "Image load fail\n";
    return false;
  }

  SDL_Texture* texture = SDL_CreateTextureFromSurface(
      renderer, temp_surface);

  if (texture == 0) {
    std::cout << "Could not create texture: " + file_path << "\n";
    return false;
  }

  SDL_FreeSurface(temp_surface);

  if (texture != 0) {
    tiles[id] = texture;
    return true;
  }

  std::cout << "Could not load image: " + file_path << "\n";
  return false;
}

void Tilemap::render(std::string id, int x, int y) {
```

➔ Memory Management

```
#ifndef OBSTACLE_H
#define OBSTACLE_H

#include <vector>
#include "SDL.h"
#include <random>

class obstacle {
 public:
  //Constructor
  obstacle(){}
  obstacle(int xpos, int ypos): _xpos(xpos), _ypos(ypos),engine(dev()), random_w(0, 3){
        _type=random_w(engine);
        }
  obstacle(obstacle &&source) // 4 : move constructor
    {
        _xpos = source._xpos;
        _ypos = source._ypos;
            _type = source._type;
        source._xpos = 0;
        source._ypos = 0;
            source._type = 0;
  }
  ~obstacle() // 1 : destructor
    {
        //std::cout << "DELETING instance of MyMovableClass at " << this << std::endl;
    }

    obstacle(const obstacle &source) // 2 : copy constructor
    {
        _xpos = source._xpos;
        _ypos = source._ypos;
            _type = source._type;
    }

    obstacle &operator=(const obstacle &source) // 3 : copy assignment operator
    {

        if (this == &source)
            return *this;
```