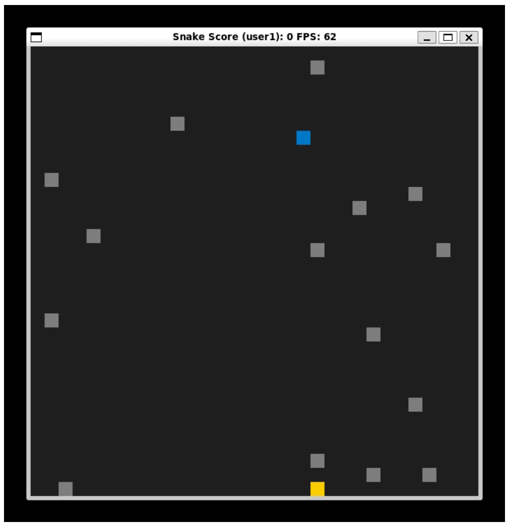
C++ ND Capstone Project - Snake Game

This is a repo for the Capstone project in the Udacity C++ Nanodegree Program. The code for this repo was inspired by this excellent StackOverflow post and set of responses.



Setup Guide

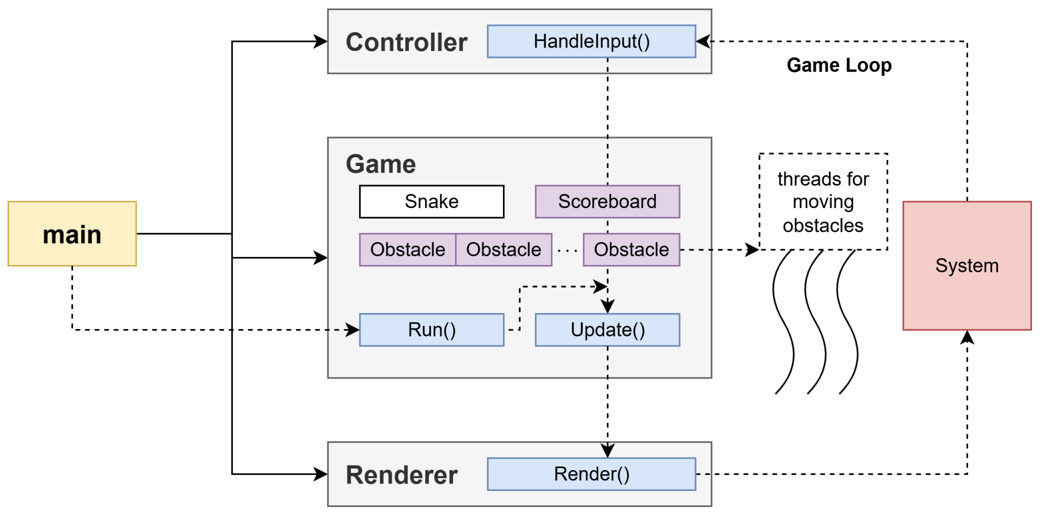
Dependencies

- 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
- SDL2 >= 2.0
  - All installation instructions can be found here
  - Linux: `sudo apt-get install libsdl2-dev`
- 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

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`

Overview of Code Structure



- `main.cpp`: Entry point of the program.
  - It will constructs `Controller`, `Game`, and `Renderer` objects, and starts the game.
- `game.h/cpp`: It holds `Snake`, `Scoreboard`, and `Obstacle` objects, and runs the game loop.
  - `Scoreboard` is a newly added class (definitions can be found in `scoreboard.h/cpp`), which is responsible for asking players for user names, and maintaining highest scores.
  - A vector of `unique_ptr`s to obstacles (`ObstacleBase`) has been added as a member to the `Game` class. Locations of obstacles will be randomized at the beginning. Separate threads will be launched for `MovingObstacle`s to change moving directions (definitions can be found in `obstacle.h/cpp`).
- `snake.h/cpp`: It contains logics for the snake movements. `alive` will be set to false when hitting an obstacle.
- `renderer.h/cpp`: It renders objects used in the game. Obstacles rendering code has been added.
- `controller.h/cpp`: It takes inputs from the player. No change has been made.

New Features

In this project, we added 2 new features to the snake game:

- New Feature 1: allowed players to enter their names and save their highest scores to a text file
  - After starting the game (by running `./SnakeGame`), the player will be asked to type in a user name (empty spaces will be dropped) from the console.
  - Under the hood, a `Scoreboard` object will be created, which will store the highest scores for each player.
  - When the game ends, `Scoreboard` will compare player's score from the current run with historical score (stored in `out/scoreboard.txt`), keep the highest score, and write the data back to `out/scoreboard.txt`.
- New Feature 2: added fixed and moving obstacles to the game
  - Obstacles have been added to the game (in `src/obstacle.h` and `src/obstacle.cpp`).
  - Locates of these obstacles will be randomly generated after starting the game, and part of these obstacles will be moving around.
  - When an obstacle hits another obstacle, it will turn around.
  - If the snake hits any obstacle, the game will end.

Project Rubric

- Please search for `STUDENT CODE` in source files to find implementations for the following rubric points.

Loops, Functions, and IO

Criteria	Implementation
The project accepts and processes user input.	`scoreboard.cpp:17-21`
The project reads data from and writes data to a file.	`scoreboard.cpp:38-53,55-64`

Object-Oriented Programming

Criteria	Implementation
New classes are added to the project with appropriate access specifiers for class members.	`scoreboard.h`, `obstacle.h`
Class constructors utilize member initialization lists.	`obstacle.h:21-23,65-72`
Classes follow an appropriate inheritance hierarchy with virtual and override functions.	`ObstacleBase`, `FixedObstacle`, and `MovingObstacle` in `obstacle.h`

Memory Management

Criteria	Implementation
The project makes use of references in function declarations.	`renderer.h:16-17`
The project uses destructors appropriately.	`renderer.cpp:37-39`
The project uses smart pointers instead of raw pointers.	`game.h:46`

Concurrency

Criteria	Implementation
The project uses multithreading (for moving obstacles).	`obstacle.cpp:21-24`
A mutex or lock is used in the project (to protect read/update for obstacles).	`obstacle.h:35`