 # CPPND: Capstone Project Single Player Ping Pong game

This repo is part of my submission for the Capstone project of Udacity C++ Nano Degree. The code is developed on top of the snake game starter code provided by Udacity. This uses SDL2 for GUI purposes. I have structured the entire game into different classes and utilized necessary initializations and member functions for operations.

This is a retro single-player ping pong game wherein the user can control the horizontal bat at the bottom and slide it left and right as desired. The aim for the player is to keep the ball up and strike it as and when it comes down. Every time the ball hits the bat and goes up, the player scores a point. The user can move the bat using the left and right arrow keys on their keyboard.

Code Structure

The game, renderer, and controller parts of the snake game have been used here. There are two new entities added, namely the bat and the ball. They have been defined in bat.h and ball.h respectively. The member function definitions are provided in ball.cpp, bat.cpp.

In-game.cpp objects of bat and ball are created.
Bat is spawned at the bottom center and kept at rest till user input is received, while the ball is generated at the screen center. Bat motion has been restricted to left and right based on user input.
The ball can move in 360 Degrees; the direction or angle of attack is indicated by the "theta" variable of the ball class. When created, a random value between 0 to 359 is assigned to theta. Update function for bat is run first.

In controller.cpp , the HandleInput function is used to receive user inputs to move the bat. setBatDirection is used to set the direction in which the bat should move.
isRest is a bool that is used to indicate whether the bat is in rest or motion; it is also set to false if user input is received if not, it is set to true.

In bat.cpp, the switch case is used to determine the new bat position based on the direction provided by the user, the setBatPos function is used for this purpose, and speed is given as input to it. If the user demands left, then speed is set to -speed , if right, then +speed, and if nothing is demanded then speed is set to 0. The left and right limit of the bat is used along with screen width to limit the bat going out of the window. Once bat position is updated next Ball position is updated.

The new position of the ball is determined based on the previous position of the ball, the angle of attack, and its speed. The ball will continue to go in its directions till it meets any obstacle, either in the form of a wall or bat. When it comes in contact with the wall, the angle of attack is modified based on the angle of incidence equals the angle of reflectance principle.

When it hits the bat random angle is added or subtracted to the angle of reflection for some uncertainty purposes. If the ball goes below the height of the bat then the GameOver variable of the ball is set to true, which is used by the game.cpp to indicate to the user that the game is over in the command window.

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)
  * 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: `./PongGame
5. Click the GUI and use the keyboard to control the bat's motion.

Rubric points

Loops, Functions, I/O
1. A variety of control structures are used in the project
    - If condition has been used to determine in which direction should the ball be deflected after hitting the bat (ball.cpp, line number 20 onwards)
    - switch case has been used to determine the position of the bat based on the keyboard press from used (bat.cpp, line number 10 onwards)

2. The project code is organized into classes with class attributes to hold the data, and class methods to perform tasks.
    -two separate classes have been created for the entities in the game namely "Ball" and "Bat" which have their own member functions and variables that are manipulated as necessary.

3. All class data members are explicitly specified as public, protected, or private.
    - the variables have been declared as private, public as necessary. (bat.h and ball.h)

4. All class members that are set to argument values are initialized through member initialization lists.
    - the class members of Ball and Bat class have been initialized through the initializer list in the constructor (bat.h-line number 12, ball.h - line number 11)

5. 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.

- the member names are self-explanatory and comments have been given wherever necessary

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

- the member data has been hidden from the user and appropriate setter and getter have been implemented for the users to modify or read a value.