Lab 2: The Pong Game

Objectives

The main objective of this assignment is to introduce you to the use of classes, objects, and methods, thus applying the related concepts presented in the lectures.
You will do so by building a Pong game. You will first implement two classes that represent the player and a ball in the game. You will then implement the `main` function that runs the game loop.

Pong Game

In this lab, you will implement the **Pong Game** with a single player controlling a paddle to hit incoming balls. The player moves the paddle up or down to prevent the balls from hitting the wall behind them. Each successful hit increases the player's score by 1. If the player misses a ball, the game is over. As the score increases, the paddle size decreases, and the number of balls increases, making the game progressively harder.

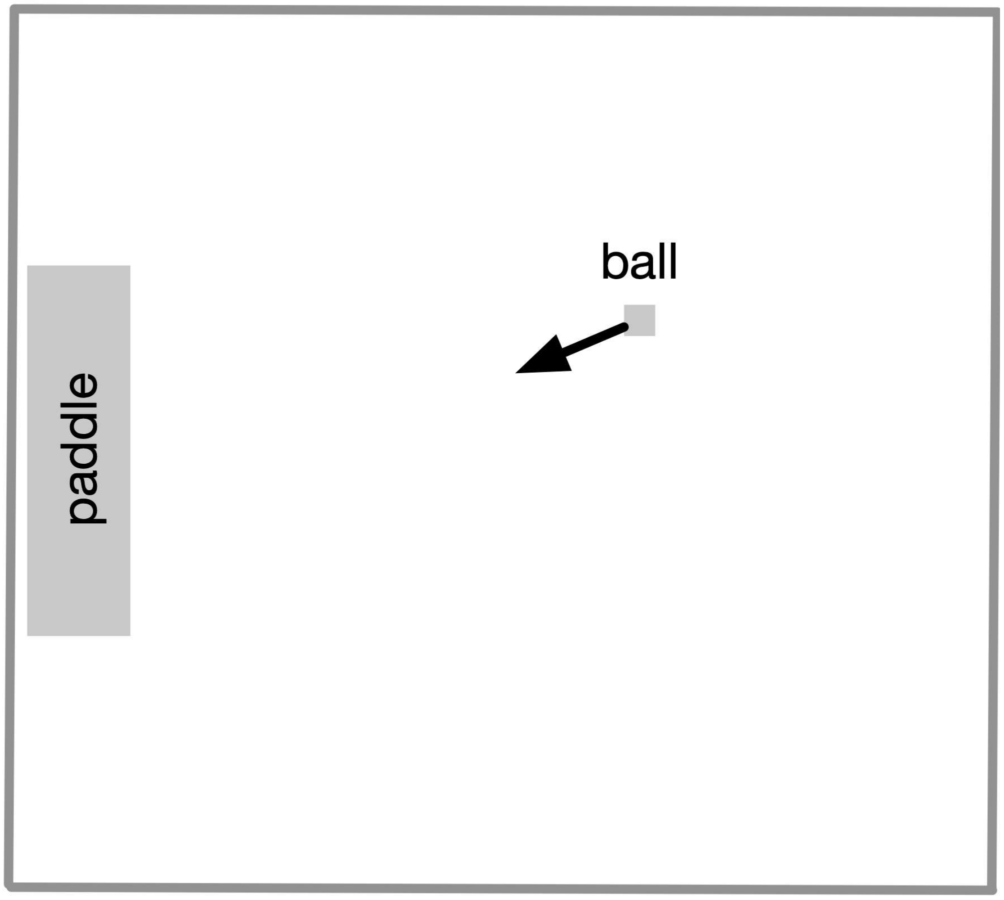


Figure 1: Example run of the pong game where the ball is moving towards the paddle.

Pong Game Program Overview

The Pong Game program implements a simplified version of the classic Pong game using object-oriented programming concepts in `C++`. The game will feature three main classes: `Ball`, `Player` and `Screen`, along with the main function to drive the game loop.

`Ball` Class in `Ball.cpp` and `Ball.h`: This class will represent the balls in the game. It includes the properties like `x` and `y` coordinates, `velocity_x` and `velocity_y` denoting the **velocity** (not speed) in the x and y direction respectively, `width`, `height` and `id` of a ball in the game. The ball's movement will be updated in each time step of the game loop based on these attributes.

`Player` Class in `Player.cpp` and `Player.h`: This class is responsible for controlling the paddle, which can move up or down to hit the ball. The `Player` class will manage the paddle's position (`x` and `y` coordinates), size (`width` and `height`) and `speed` at which the paddle moves. Every 2 hits, the height of the paddle will shrink by 1 unit till it reaches a `height` of 3, adding an additional layer of difficulty.

`Screen` Class in `Screen.cpp` and `Screen.h`: The `Screen` class will handle the display of the game. It is responsible for printing the current state of the game, including the positions of the ball, the paddle, and the score, on the console/terminal. The class is implemented for you. You will use the `Screen` class to print the balls and the paddle. The class uses some functions implemented in `io.cpp` and declared in `io.h` in file. **Do not change anything in these files.**

Main Function in `main.cpp`: The main function will control the game loop, initializing the game components, taking input from the user and updating the game state on each iteration. It will manage the game's flow by checking for collisions between the paddle and the balls, updating the score, and ending the game when the player misses a ball.

As the player hits the ball, their score increases, the paddle size decreases, and more balls are introduced into the game. The challenge increases with time, testing the player's reflexes and strategy. If the player misses a ball, the game ends, and the final score is displayed.

Problem Statement

In this lab, you are tasked with implementing the simplified version of the Pong game program. The objective is to simulate a player controlling a paddle that moves vertically as the user moves to hit balls, earning points as the game progresses.

In order to do so, you will implement the methods of the class `Ball`, as defined in `Ball.h`, `Player`, as defined in `Player.h`. You will also implement the `main` function, which "plays" the move indicated by a player's input. The remainder of this section describes: (1) the `Player` class and its methods, (2) the `Ball` class and its methods, (3) the `main` function that implements the game loop.

The `Player` Class

The dynamics of the paddle in the Pong game is represented by an instance of the `Player` class. The definition of this class appears in the file `Player.h`, which is released with the assignment. **You may NOT modify this file** to add to or delete from its content. Modifying the file often results in a mark of zero for the assignment. The class contains the following data members:

- `x` and `y` coordinate of the ~~ball~~ paddle in the canvas/game window, which is managed by the `Screen` class. As shown in Fig.2, the value of `x` ranges between 0 and WIDTH - 1 and `y` ranges between 0 and HEIGHT - 1. The macros WIDTH and HEIGHT are defined in `Globals.h` file, which is also part of the assignment release. **You may NOT modify this file** to add to or delete from its content.
- `width` and `height` are the width and height of the paddle. The width should be set to 1 unit.
- ~~`speed` is the number of units with which we update the y coordinates of the paddle every time the user enters an action.~~

The class contains the following function members that you will implement:

- `getX()` returns the value of the x coordinate
- `getY()` returns the value of the y coordinate
- `getHeight()` returns the height of the paddle
- `getWidth()` returns the width of the paddle
- `decreaseHeight(int delta_to_decrease_by)` decreases the height by `delta_to_decrease_by` amount. The minimum height is 3.
- `update(char c)` updates the `y` coordinates of the paddle according to the value of `c`. If the user presses the up arrow key (represented by `c == 'A'`), the y coordinate should increase by the value of ~~speed~~ 2. Conversely, if the down arrow key is pressed (`c == 'B'`), the y coordinate should decrease by the value of ~~speed~~ 2. The paddle should not fall below the floor or above the ceiling. In Fig. 2, we show the paddle x and y coordinates with respect to the game window to help you figure out the allowed range of the y coordinate.
- `draw(Screen& screen_to_draw_to)` renders the paddle on the game window by invoking `addPixel` on the `screen_to_draw_to` object (an instance of the Screen class). The addPixel function prototype is `void addPixel(double x, double y, char symbol)`, where x and y are the coordinates of the pixel we want to draw, and `symbol` is `#`. To draw the paddle, call addPixel repeatedly on each pixel of the paddle. In Fig. 3, we show the game printed on the terminal. The paddle is shown with `#`.

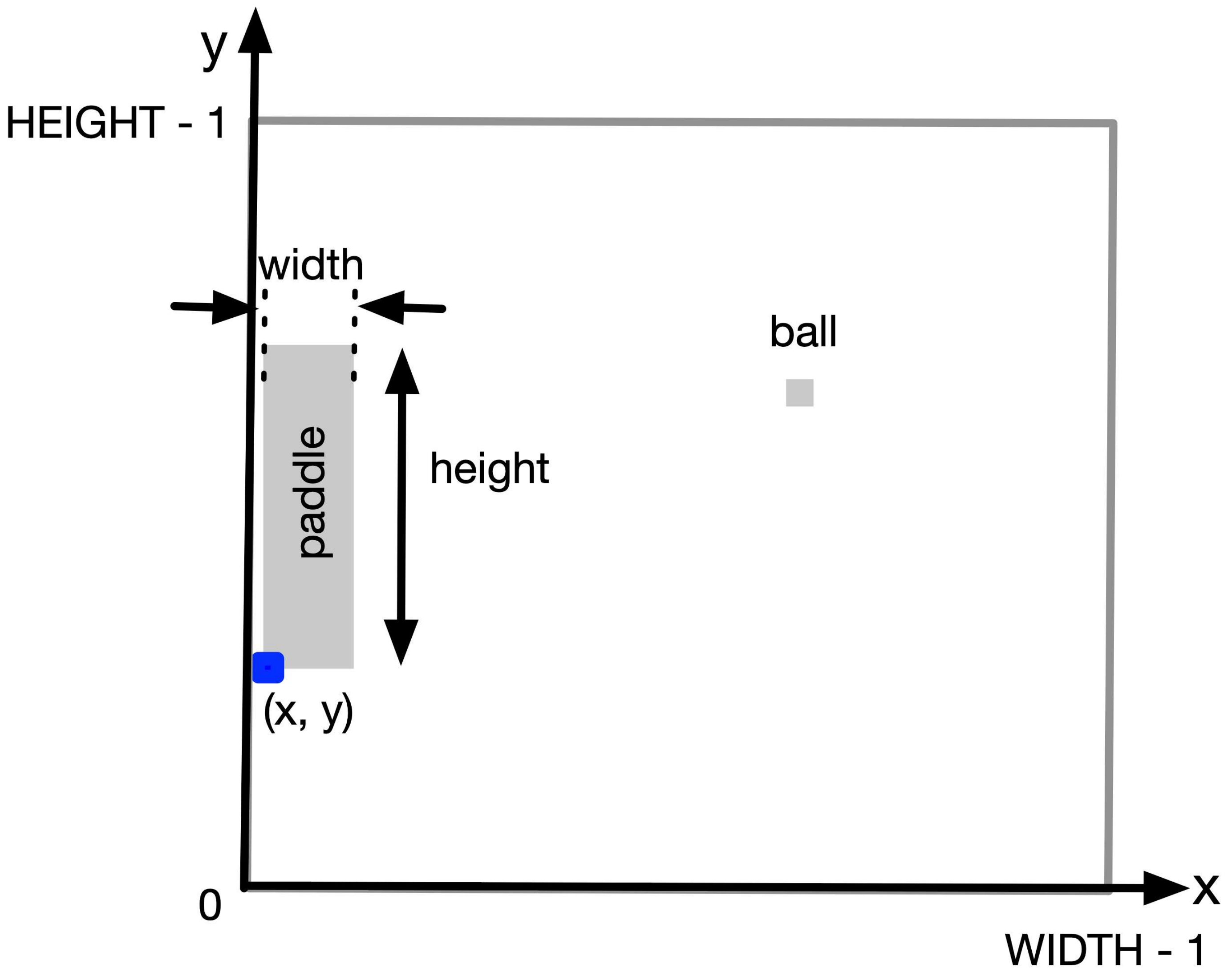
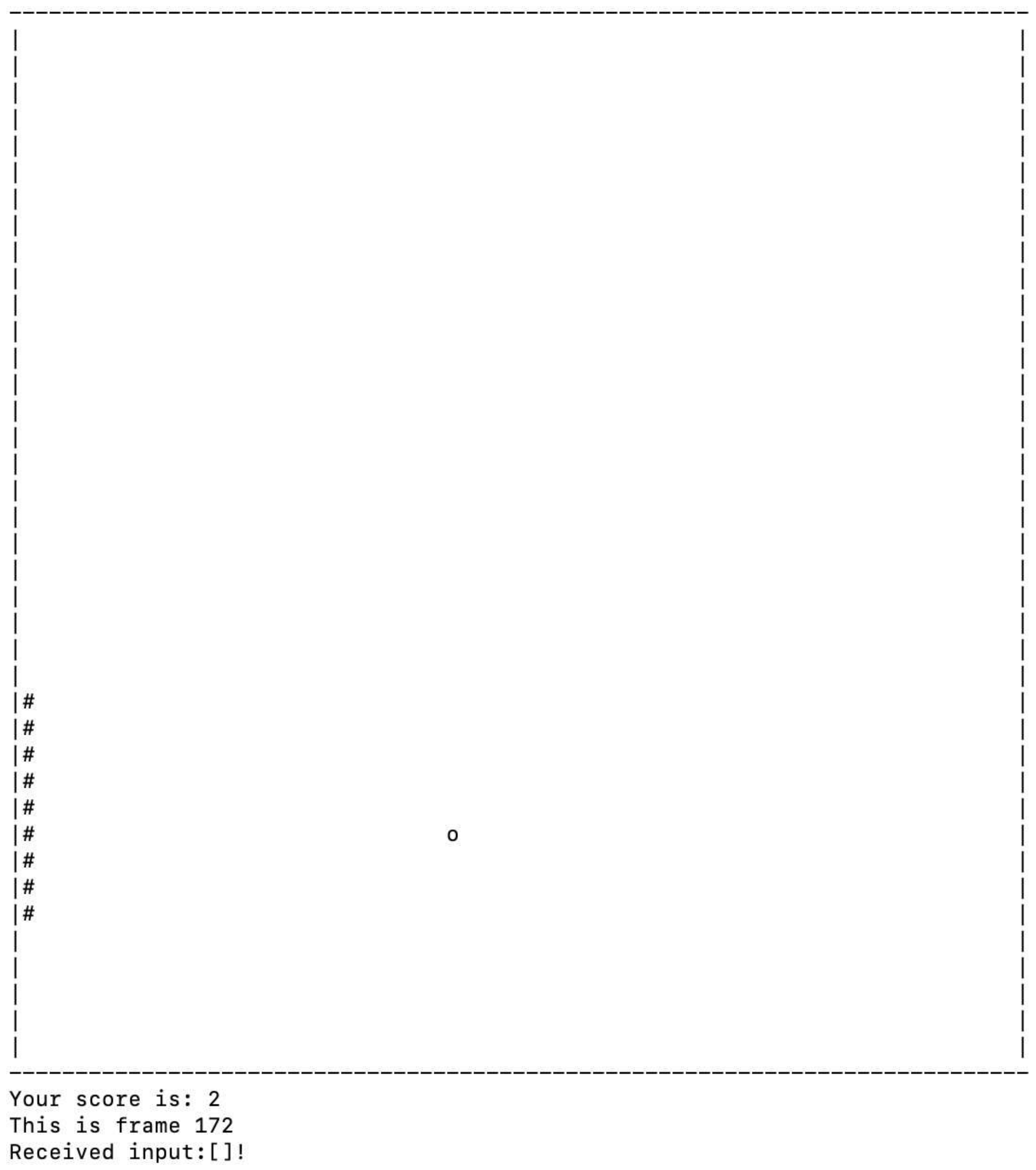


Figure 2: A drawing showing a paddle location and dimensions in a pong game.



```
-----------------------------------------------------------------------------
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|#                               o                                          |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
-----------------------------------------------------------------------------
Your score is: 2
This is frame 172
Received input:[]!
```

Figure 3: The game printed on the terminal.

The `Ball` Class

The dynamics of the balls in the Pong game is represented by an instance of the `Ball` class. The definition of this class appears in the file `Ball.h`, which is released with the assignment. **You may NOT modify this file** to add to or delete from its content. Modifying the file often results in a mark of zero for the assignment. The class contains the following data members:

- `x` and `y` coordinate of the ball in the canvas/game window, which is managed by the `Screen` class. The value of x ranges between 0 and WIDTH - 1 and y ranges between 0 and HEIGHT - 1. The macros WIDTH and HEIGHT are defined in `Globals.h` file, which is also part of the assignment release. **You may NOT modify this file** to add to or delete from its content.

- `width` and `height` are the width and height of the ball, should be set to 1 unit for all balls.
- `velocity_x` and `velocity_y` are the velocities of the ball in the x and y direction. `velocity_x` and `velocity_y` are the number of units a ball moves in the x direction and y direction (respectively) in one time step or iteration. For example, if the `velocity_x` is negative, this means the ball's x position is decreasing by `velocity_x`'s magnitude every time step. The time step is defined in `Globals.h` as the macro `timeStep`. **Decreasing x coordinates means the ball is moving to the left. Decreasing y coordinate means the ball is moving down as illustrated in Fig. 2.**
- **id** is a unique identifier for each ball in the game. It can be set to the ball's number. It will be used to check if a ball is colliding with **another** ball.

The class contains the following function members that you will implement:

- `getX()` returns the value of the x coordinate
- `getID()` returns the value of the id of the ball
- `update()` would first account for the change in the velocity in the y direction due to gravity, then updates the x and y coordinates of the ball for one time step `timeStep`. The velocity in the y direction becomes more negative with acceleration of 0.02 units per (time step)^2, i.e. `velocity_y = velocity_y - 0.02 * timeStep`. The displacement of the ball in a particular direction is its velocity in that direction $\times$ the time step.
- `overlap(Ball& b)` checks if the ball of the current instance collides with ball b and if it collides with it, the function determines in which direction the collision is happening. The function returns an `int`, which is either HORIZONTAL_OVERLAP, VERTICAL_OVERLAP or NO_OVERLAP. The values of these macros are defined in `Globals.h`. If there is no overlap, the function should return NO_OVERLAP. In Fig. 4, we show how to measure the dimensions of the overlap. If the vertical overlap is larger than the horizontal, the function should return VERTICAL_OVERLAP. Otherwise, it should return HORIZONTAL_OVERLAP.
- `overlap(Player& p)` checks if the ball of the current instance collides with the player p. The logic of the function is the same as `overlap(Ball& b)`.
- `bounce(Ball arr[], int ballCount, Player player)` checks if the ball of the current instance bounces of any other ball, wall or the paddle of the player. The coordinates of the game window range from 0 to WIDTH - 1 in the x direction, and from 0 to HEIGHT - 1 in the y direction. If the current ball location is at or beyond the vertical walls, the sign of the velocity in the x direction should be flipped. If the current ball location is on the ground or beyond, the sign of the velocity in the y direction should be flipped. To bounce off another ball, you need to call the `overlap` function on all balls. If two balls overlap horizontally, flip the x-velocity; otherwise, flip the y-velocity.
- `draw(Screen& screen_to_draw_to)` renders the ball on the game window by calling addPixel on the `screen_to_draw_to` object (an instance of the Screen class). The addPixel function prototype is `void addPixel(double x, double y, char symbol)`, where x and y are the coordinates, and symbol is `'o'` (lower case letter O). Since the width and height of the ball is 1, we only call addPixel once.

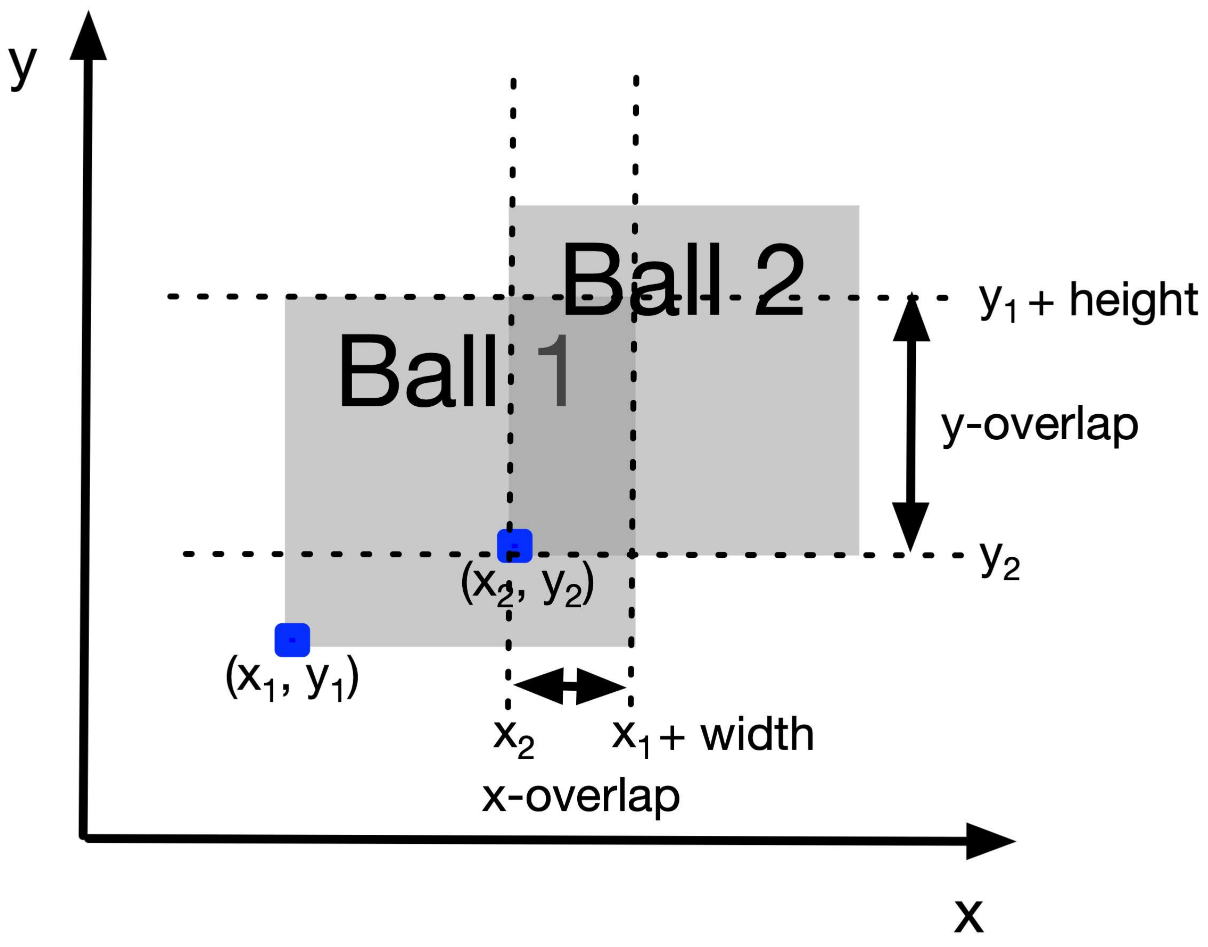


Figure 4: The overlap between two balls.

The Game Controller

The game controller is implemented in the `main` function of the game, contained in the file main.cpp. Its purpose is to first create the necessary game elements, including the player and balls, and then control the main game loop. Within this loop, the controller handles user input, updates the physics of the game (player and ball movements), and manages game state transitions such as scoring and game over conditions.

The game is a series of frames printed to the user until the game ends. A frame refers to one complete cycle of drawing the game's state on the screen. In a typical game loop, the game processes input, updates the position of objects, handles physics, and then renders the updated state to the display. The number of frames rendered per second is called frames per second (FPS), referred to as `screen_fps` in `Globals.h`. For example, a game running at 60 FPS updates the visuals on the screen 60 times per second.

In the main function, for each frame/cycle in the game loop, we run `simulation_fps` number of iterations. Each iteration simulates one `timeStep`. In one iteration/time step, we check for input, update balls and paddle positions. After `simulation_fps` number of iterations, we will wait for the time of one frame to pass, and then print the new frame as already implemented in the main function.

Each iteration is a time step `timeStep`, which is defined in `Globals.h`. In each iteration as long as the game did not end, we will:

- Take the user input through `get_input()` function implemented in `io.cpp`. If it is `'q'`, the game ends. The function returns `'\0'` if the user did not enter input. If the user input is not `'\0'`, we update the paddle using the methods in Player class.
- Update the position of each ball and check for collisions/overlaps with the player using methods in Ball class. If a ball hits the paddle, we increment the score by 1.
- Every 2 hits, the paddle size decreases by 1 using a method in Player class. Every 5 hits, we increase the number of balls by 1 till we have a maximum of 5 balls.
- New balls created should have `(x, y) = (30.0, 30.0)`, `velocity_x = 0.9` and `velocity_y = 0`.

After the iterations for a frame are over, we draw the balls and player using the `draw` methods implemented in Ball and Player class.

We repeat the above process as long as the game did not end. The game ends when the player quits or a ball bouncing off the left wall.

Important! The skeleton code in main.cpp will render the updated game state to the screen by drawing the player, the balls and the score. It will also go through another game cycle after the time of one frame has passed. You are not required to edit the lines in the main.cpp. You are only required to add your game loop in the main function.

Include Files

In addition to the Ball.h and Player.h files described above, the file Globals.h has global definitions for multiple macros such as simulation_fps and HORIZONTAL_OVERLAP. You should include this file if you plan to use it in Ball.cpp and/or Player.cpp.

We also have io.h and Screen.h function implementations for input/output and screen printing. Again, **you may NOT modify these files** to add to or delete from their content. Modifying the files commonly results in a mark of zero for the assignment.

We also released a `tester` directory that you are not required to edit. It will be used on ECF during testing.

Reference Executable

A reference executable can be run on ECF by running `~ece244i/public/game-ref`. You can use this reference to play the game.

Please note that this reference executable works only on ECF machines. It will not work in Window machines or Macs. Thus, to use it, either use it on ECF machines or connect to ECF, as is described in the "Background" handout.

Procedure

Create directory called lab2 in your ece244-labs directory. Make sure that the permissions of this new directory are such that it is readable by none other than you (refer to lab 1 for how to do so). Download the zip file containing the assignment release files and place it in this lab2 directory. Unzip the file, which will create the assignment files in the directory. You can move the zip file out of the directory or remove it after this step.

There are ten assignment files: Ball.cpp, Ball.h, Player.h, Player.cpp, Screen.h, Screen.cpp, io.h, io.cpp, Globals.h and main.cpp. You will add your code in three files: Player.cpp, Ball.cpp and main.cpp. The first contains the implementation of the methods of the class Player. The second contains the implementation of the methods of the class Ball. The third contains the main function that implements the game loop. Some skeletal code is given in main.cpp.

Remember that **you may NOT modify any other file**. Modifying the other files commonly results in a mark of zero for the assignment.

Finally, there is a `Makefile` that is given for you to compile your code.

To test your code **locally**, you can simply type **`make game`** in the command line of the directory containing this `Makefile`. This will generate an executable named `game`. In the command line type:

```
./game
```

To test your code on ECF with the test cases, run `make` in your ece244-labs/lab2 directory after transferring your code to ECF. An executable named `autotester` will be generated. In the command line type:

```
./autotester
```

The autotester command will let you know if your code has errors by providing it with several test cases.

Marking and Deliverables

You must submit your code for autotesting for 7 marks of your grade. The autotester will be used to check the correctness of your Player, Ball class implementation and your main function. Thus, you need to submit only the three files: Player.cpp, Ball.cpp and main.cpp. Your copies of the .h files are ignored, since you are not allowed to modify them.

To submit your code, make sure you are in the directory that contains the source files, i.e., your `ece244-labs/lab2`. Submit your Player.cpp, Ball.cpp and main.cpp files as lab 2 using the command:

```
~ece244i/public/submit 2
```

For 3 marks of your grade, you must visit your lab session anytime from September 16 to October 11, and ask the TA to grade you. The TA will ask you to demonstrate your code, and ask you some questions about your code and understanding.

Appendix

The printed game as you run `./game` executable.

```
-----------------------------------------------------------------------------
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|#                                       o                                  |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|#                                                                          |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
|                                                                           |
--------------------------------------------------------------------------------
Your score is: 2
This is frame 177
Received input:[]!
```