Chapter 1: Introduction and Problem Statement:

This project is a replica of the famous 1972 arcade game “PONG” where two players each control two bats and try to score as much goals as possible in each other by throwing the ball at each other’s nets, simultaneously trying to defend their nets from goals in the process.

The purpose of this project is to create an interactive and enjoyable gaming experience while demonstrating our capabilities of OOP programming approach and software engineering practices in the world of game design. And the use of C++ to create clean and quick programs that can run without the hassle of other heavy third-party engines.

The programs used in this program where Visual Studio IDE and C++ programming language, in addition to the free open source graphic library SFML (Simple and Fast Multimedia Library) which was used as an alternative easier method than having to use OpenGL.

Some of the technologies in the program involves the usage of but not limited to:

- Polymorphism
- Encapsulation
- Modularization
- Constructors
- Static functions
- Static attributes
- Dynamic memory allocation
- File handling
- Exception handling
- Global constants

Some of the functionalities of the project where:

1. Implementation of classes for different entities of the game, currently having a class for the bat and the ball with its own header and .cpp file, while also having a “utility” score class, with static functions and attributes to handle the scoring system dynamically.
2. Each of *bat.h* and *ball.h* contain functions to handle certain properties of the game such as setting color, speed, spawn location etc.

3. Scoring system that allows the highest score to be saved to a separate file right before closing the game, and it can be retrieved on the next session to always have the highest score of all time displayed on the screen.
4. Exception handling was used in case loading or saving scores have failed for any reason regarding the files.
5. The game uses “delta time” technology which ensures the game always runs at the same speed to all players, no matter the framerates and the updates frequency of the different computers. making the game always fair
6. There is few none used functions regarding a single player mode of the game, where the player plays against the wall.
7. The game color pallet was set and chosen for more aesthetic feeling for the game. While also having functions to make it easier to change the pallet later so it’s not hardcoded into the classes.

Chapter 2: Solution Design and Implementation:

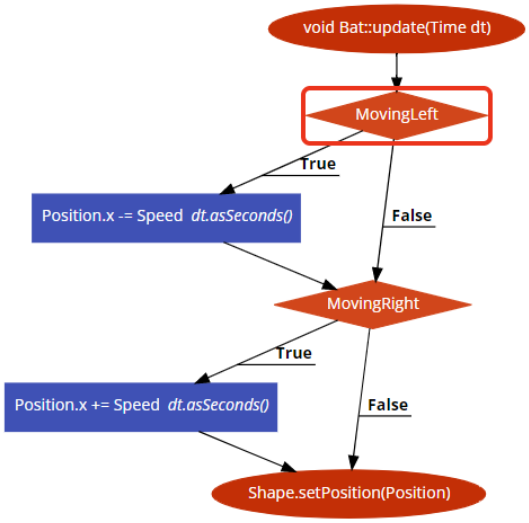
Design Choices:

OOP:

The object oriented approach was adopted to build the game. The bat and ball entities have been identified as classes and a score class was created just to handle some functions regarding scoring to make the code more manageable and easier to read.

<u>The bat class:</u>

- Has a constructor with arguments to chose the position of spawning and sets a the default size, speed and shape of the bat
- The variables are encapsulated and there are functions to access the position, shape and to edit the color of the bat
- The functions to handle the movement work as: as long the player is holding the button, the movement Boolean is true, and as he lets go of the button, the movement Boolean is false
- The update function uses the delta time given, movement Booleans, and the speed to move the bat on the screen. The final speed is calculated as the product of dt and speed



Flowchart of the Bat::update function, which will be continuously called as long as the game is running in main later

<u>The ball class:</u>

- Has a constructor with arguments to set the position of spawning and sets default parameters to the ball such as speed, size and color
- The variables are encapsulated and there are functions to access the position, shape, velocity and edit the color of the ball
- There are functions to handle the physics of the ball: when the ball hits one of the left or right walls, its direction on the x axis is inverted and it moves to the other direction

When the ball hits the bat of one of the players, there is a function to invert the ball on the y axis, and slightly increase the speed of the ball. This effect can be stacked till the ball reach a very high speed of 2000.0f

- When one of the player scores, there is a function to “reset” the game. It takes the arguments of the screen size and resets the ball to the default velocity and teleports it to the middle of the screen, it also inverts its direction on the y axis giving the losing player a fair advantage.
- The update function uses the same delta time method as the bat but doesn’t need Booleans, as there is no keystrokes involved in moving the ball, it is handled by the program.

<u>The score class:</u>

- It uses file management and dynamic memory allocation to save the highest score of each player, and loads it back every time a new game is initiated. It uses two functions for saving and loading.
- There are two static attributes that always store the highest score ever reached for each player1 and player2, with player one being the one on the bottom of the screen.
- *static void saveScores(int scorePlayer1, int scorePlayer2);*
  it allocates memory for a new file and opens the “score.txt” file in the game files, it then compares the score of the two players in the current game, and the highest score (which is the same as the one in the file) and based on that, it decides to save either the new highest score or keep the old highest score in the file. After that it closes the opened file and deallocates the memory for it as its no longer needed.
- *static void loadScores(int& scorePlayer1, int& scorePlayer2);*
  same as saveScores, it allocates memory for a new file and opens “score.txt” but this time it takes the scores in the file and plugs them into the static attributes for them, meaning that it tells the game what the highest score was ever reached based on what is written in the file. After that it closes the file and deallocates the memory.

```
function saveScores(scorePlayer1: int, scorePlayer2: int)
  new file = ofstream pointer
  file.open("scores.txt")

  if file.isOpen() then check if: {
    if scorePlayer1 > currentScore1 then
          save scorePlayer1
    else
          save currentScore1
    if scorePlayer2 > currentScore2 then
          save scorePlayer2
    else
      save currentScore2
    }
    file.close()
}
  else
    throw error("Failed to open scores file.")
  delete file
end
```

Initializing the game (when first opening):

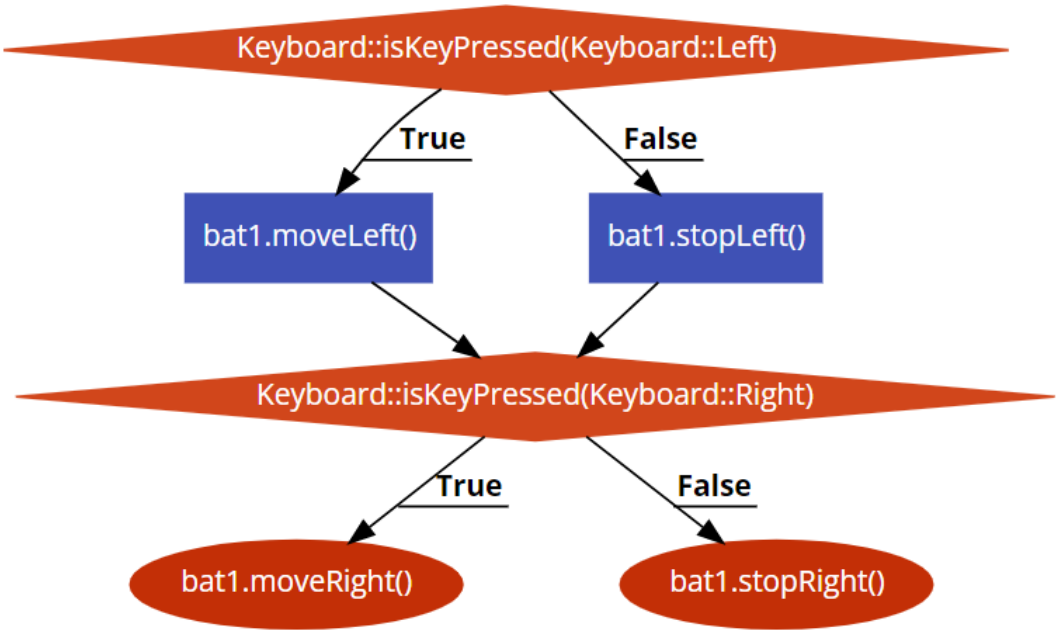
- The window resolution is set and a window with the given resolution is made.
- 2 objects of bat and 1 object of ball are created.
- We set the color of each object according to our color pallet.
- We run the loading scores function in a try block to get our highest score
- We prepare the hud of the game by adding text objects and choosing the correct fonts, character size, location to be set on screen.
- We create the clock, which will be useful later in updating the game

```
Text hud1, hud2;
Text lastScore1, lastScore2;
Text dashes;

Font font;
font.loadFromFile("fonts/Enrique-Round.ttf");
hud1.setFont(font);
hud2.setFont(font);
lastScore1.setFont(font);
lastScore2.setFont(font);
```

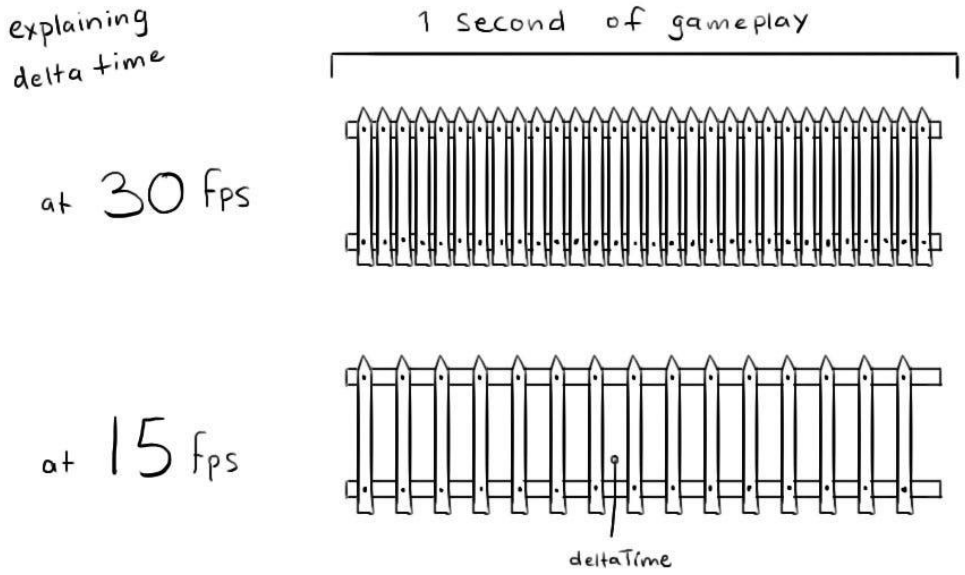
Update loop:

- As long as the game running, we check and make sure to update various things accordingly, this is also where we draw everything to the screen after updating.
- We check if the player wants to close the game, if he decides to exit we make sure to save the scores with saveScore first.
- We handle the movement of the two bat objects by identifying the keystrokes and by using the previous functions and Boolean variables that we made before



- We check if the ball ever hit the top or the bottom of the screen (one of the player scored) if so, we update the score variable and text. We also use the function we made before for the ball to reset it to the center of the screen and start a new round.
- We check if the ball hit the left or the right sides of the screen, in that case we use one of the functions we made in ball to bounce it back again and keep playing.
- We check if the ball ever hit the bat, in that case we use the function we made for that in ball, to bounce the ball back in the other direction.
- ***Now that all checking is done, we make our delta time:***
  Delta time is the time difference it takes for our clock to restart, we update our clock once every while loop, meaning that it is the time taken for every update done to the game. We need this number to stabilize the movement of the objects in the game. That way if the game is running at a slow speed, with few updates, the bat objects become faster than the update rate. And if the game is running at

very fast speed with lots of updates, the objects become slower. That way the game seems to be running at correct speed without having the updates speed affect the gameplay experience.



- We then use this delta time as arguments to the functions that updates the position of the ball and bats objects.
- We update the texts that display the scores with the latest scores.
- After all of these updates, we must make sure to actually draw all of these objects to the screen. We use the window.draw(&object) for that
- Another function is needed after window.draw, which is window.display(), it takes no arguments and displays everything in our window object to the screen of the user.

```
// Updating the whole window
// Clear old window and set its color our color
window.clear(bgColor);
//Draw all of our objects
window.draw(hud1);
window.draw(hud2);
window.draw(lastScore1);
window.draw(lastScore2);
window.draw(dashes);
window.draw(bat1.getShape());
window.draw(bat2.getShape());
window.draw(ball.getShape());
//Display it to screen
window.display();
```

Chapter 3: External Solutions and Deviations:

The only external solution used was SFML library, and it was used as a third party multimedia library. Other libraries could have been used such as SDL2 or raylib

Chapter 4: Testing and Verification:

To test the stability of the game, the first test case was the game begin tried with 1 player playing as both players multiple times. This continued till the player reached a score of 20 on each side.

Second test case was 2 players playing together on the same computer, this continued till each player scored more than 10 points

Third test case was trying the saving score mechanism, this was by playing for a few rounds and resetting the game to check how the scores were loaded and saved.

Fourth test case was changing the accessibility settings of the scores.txt file in the game and checking how the scoring system would react

Chapter 5: Results, Discussion, and Future Enhancements

Results:

- The game showed stability on the first 2 test cases, except that when hitting the ball from the side of the bat, a collision is detected and the ball speed is increased, but that happens quicker than the ball can bounce. Which leads to the ball flying faster than usual. This isn't intended to happen but can be seen as an interesting mechanic rather than a bug
- The game worked as intended for the third test case.
- The game threw an exception and failed to save/load scores on test case four.

Discussion:

The game showed success and proved to be what its meant to be. It's a good application for implementing various oop principles and an interesting demonstration of various functionalities of C++.

Future Enhancements:

- Implementing GameObject class to work as a parent for both bat and ball with common functions
- Implement GameMaster class to store some of the work that is done in the main function and have other useful functions to overall make the code cleaner and easier to deal with
- Make the game design more symmetrical
- Implement a main menu and single player mode with lives where the ball hits the wall till he misses against himself and runs out of lives
- Add sound effects
- Add "frenzy" gamemode where after a number of goals, the number of balls on the screen increases and the player has to watch out for multiple balls at once
- Add power ups such as: speed up, larger bat, slower ball

Chapter 6: Conclusion and References:

Working on the project was very satisfying for me, as after a lot of hard work and time. It was well spent; as long as I was working I was learning new things about the functionalities of C++ and OOP. I also had fun learning about implementing of GUI and trying to set the correct coordinates of each object and drawing each shape.

I hit some roadblocks such as using of static attributes globally and drawing the UI of the game correctly. But thankfully they were solved.

My lab instructor provided a lot of suggestions and help with some of the roadblocks I faced.

I used the official documentation of SFML as a reference and learning material for the library

https://www.sfml-dev.org/documentation/2.5.1/annotated.php