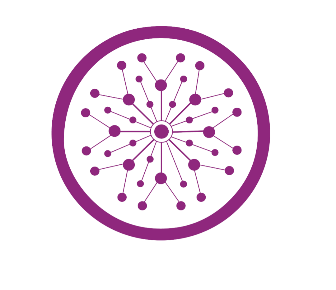
**Object-Oriented Programming (Lab)**

**2nd Semester**

**“BALL GAME”**

**Project Report**



**Lab Instructor:** Miss. Amna Bibi

**Section Name:** 2F

**Group Members:**

|  |  |
| --- | --- |
| **Student ID** | **Student Name** |
| SU92-BSCSM-F23-219 | Inam ul Hassan |
| SU92-BSCSM-F23-052 | Ukasha Zaheer |
| SU92-BSCSM-F23-293 | Kinza Batool |
| SU92-BSCSM-F23-374 | M. Huzaifa |

**Department of Computer Science**

**The Superior University, Lahore** (**Gold Campus**)

* **ABSTRACT:**

The motivation for developing a console-based version of the classic Pong game is to apply and demonstrate Object-Oriented Programming (OOP) ethics in a practical and educational way. Pong, with its simple mechanics like ball movement, paddle control, collision detection, and scoring, is an ideal project to showcase core OOP concepts such as inheritance, encapsulation, and polymorphism. This project highlights how these principles can be used to create modular, scalable, and maintainable code in game development. By building this game, we aim to show how OOP can improve the design and functionality of a simple yet engaging game.

Table of Contents

**Abstract**  …………………………………………………………………………...**2**

**1. Introduction of the Project**..………………………………………………… **4**

**2. Modules of the project** …………..……………………………………………**4**

**3. Tool Used** ….….…..……..…………………………………………………….**5**

**4. Classes, Attributes and Methods** ……………………………………………**6**

**5. Algorithm used to design the code** ...…………...…………………………..**13**

**6. Concepts used to write the code** …………….……………………………...**14**

**7. Output of the code** ..…….…………………………………………………...**18**

**8. Challenges faced during the project** .……………….……………………...**19**

**9. Conclusion**………………………………………………………………...….**19**

# 

1. **Introduction:**

This project involves creating a console-based version of the classic game Pong using C++ and Object-Oriented Programming (OOP) principles. The objective is to demonstrate effective use of OOP concepts like inheritance and encapsulation to model game elements (paddles, ball, game manager), ensuring modularity and scalability. Players control paddles to bounce a ball across the screen using keyboard inputs, while collision detection and scoring mechanics are handled by the game manager. After finishing the project, the goal is to create a fully functional console simulation of Pong. This simulation will feature gameplay that is easy to understand and play, with realistic physics for the movement of the ball and paddles. The aim is to provide an enjoyable experience that closely resembles playing the original Pong game. The project will also feature adjustable difficulty levels, allowing players to choose between varying speeds for the ball and paddle movements. This customization aims to cater to different skill levels and preferences, enhancing the game's accessibility and replay ability.

1. **Modules of the Project:**

* **‘cGameObject’** Class:

This is the **Base class** and it has following functions:

* + - cGameObject
    - void reset ()
    - int getX () int getY ()
    - void resetPosition ()
* **‘cBall’** Class:

This class has following functions:

* void reset ()
* void changeDirection ()
* void randomDirection ()
* void move ()
* **‘cPaddle’** Class:

This class has following functions:

* void reset ()
* void changeDirection ()
* void move ()
* **‘cGameManager’** Class:

This class has following functions:

* + - void Draw ()
    - void Input ()
    - void Logic ()
    - void Gam ()

# **Tool Used:**

We used Visual Studio Code for this project because of its strong support of modern C++ features and robust debugging capabilities, enabling efficient development from the beginning. To ensure compatibility across different operating systems, the project's dependencies and build configurations were carefully managed. This approach aimed to maximize accessibility for potential users while maintaining consistency in performance and functionality.

# **Classes, Attributes and Methods:**

**Subclass of base class**

cPaddle

**Subclass of base class**

cBall

**Base class**

cGameObject

**Derived Class**

cGameManger

**void resetPosition ()** { x = originalX; y = originalY; }

**Methods**

**void reset () override** { resetPosition (); dir = STOP; } **void changeDirection**(eDirection direction) { dir = direction; } **void randomDirection**() { srand(time(0)); dir = static\_cast<eDirection>((rand () % 6) +1); } **inline eDirection getDir**() const { return dir; }

**Attributes**

eDirection dir;

**subclass**

c ball

**inline int getX() const** { return x; } **inline int getY() const** { return y; }

**virtual** ~cGameObject () { } **virtual** void reset () = 0;

**Attributes**

int x, y; int originalX,

int originalY;

**Methods**

cGameObject(int posX, int posY) { originalX = posX; originalY = posY; x = posX; y = posY; }

**Base class**

cGameObject

**Methods**

**void move ()** { switch (dir) { case STOP: break; case RIGHT: x++; break; case UPRIGHT: x++; y--; break; case DOWNRIGHT: x++; y++; break; case LEFT: x--; break; case UPLEFT: x--; y--; break; case DOWNLEFT: x--; y++; break; default: break; }

**subclass**

c Paddle

**Attributes**

eControler joyStick;

**Methods**

**cPaddle(int posX, int posY) ;**

cGameObject(posX, posY) { joyStick = PAUSE; } void reset() override { resetPosition(); joyStick = PAUSE; } void changeDirection(eControler dir) { joyStick = dir; }

void move() { switch (joyStick) { case UP: y--; break; case DOWN: y++; break; case PAUSE: break; default: break; } }

**void Draw ()** {SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), {0, 0}); for (int i = 0; i < width + 2; i++) { cout << "\xB2"; } for (int i = 0; i < height; i++) { for (int j = 0; j < width; j++) { int ballX = ball->getX (); int ballY = ball->getY (); int player1x = player1->getX (); int player1y = player1->getY (); int player2x = player2->getX (); int player2y = player2->getY (); if (j == 0) { cout << "\xB1"; } if (ballX == j && ballY == i) { cout << "o"; } else if (player1x == j && (player1y >= i && player1y < i + 4)) { cout << "\xDB";} else if (player2x == j && (player2y >= i && player2y < i + 4)) { cout << "\xDB"; } else {cout << " "; if (j == width - 1) {cout << "\xB1";} } } cout << endl}for (int i = 0; i < width + 2; i++) {cout << "\xB2”;} cout << endl; for (int i = 0; i < width / 2 - 3; i++) {cout << " ";} cout << score1 << “: " << score2 << endl;**}**

**Methods**

**cGameManager ()** { quit = false; height = 15; width = 50; score1 = score2 = 0; ball = new cBall(width / 2, height / 2); player1 = new cPaddle (1, height / 2); player2 = new cPaddle (width - 2,

height / 2); }

**~cGameManager ()** { delete ball; delete player1; delete player2; } **void ScoreUp(cPaddle \*player)** { if (player == player1) { score1++; } else if (player == player2) { score2++; } ball->reset (); player1->reset (); player2->reset (); }

**Derived Class**

cGameManger

**Attributes**

int width, height; int score1, score2; bool quit; cBall \*ball; cPaddle \*player1; cPaddle \*player2;

**Methods**

**void Input()** { int player1y = player1->getY(); int player2y = player2->getY(); if (\_kbhit()) {

if (ball->getDir() == STOP) {ball->randomDirection() } switch (\_getch()) {case 'w': if (player1y - 3 > 0)

}

{ player1->changeDirection(UP); } break; case 's': if (player1y + 1 < height) {

player1->changeDirection(DOWN); } break; case 'i': if (player2y - 3 > 0) { player2->changeDirection(UP); } break; case 'j': if (player2y + 1 < height) { player2->changeDirection(DOWN); } break; case 'q': quit = true; break; default: break; } }

**void Game ()** { string p1, p2; cout << "Player1: "; cin >> p1; cout << "Player2: "; cin >> p2; while (! quit) { Draw (); Input (); Logic (); Sleep (50); } system("cls"); cout << "Player1 : " << p1 << " - " << score1 << " ||| Player2 : " << p2 << " - " << score2 << endl; **else { cout << "Player2 win! "; } Sleep(5000); }};**

if (score1 == 10) { cout << "Player1 win! "; } else { cout << "Player2 win! "; } Sleep(5000); }};

else if (ballY == player2y) { ball->changeDirection(DOWNLEFT); } } if (ballY == height - 1) { ball->changeDirection(ball->getDir() == DOWNRIGHT ? UPRIGHT: UPLEFT); } else if (ballY == 0) { ball->changeDirection(ball->getDir() == UPRIGHT ? DOWNRIGHT: DOWNLEFT); } else if (ballX == width - 1) { ScoreUp(player1); } else if (ballX == 0) { ScoreUp(player2); } if (player1->getY() - 4 < 0) { player1->changeDirection(PAUSE); } else if (player1->getY() + 2 > height) { player1->changeDirection(PAUSE); } if (player2->getY() - 4 < 0) { player2->changeDirection(PAUSE); } else if (player2->getY() + 2 > height) { player2->changeDirection(PAUSE); } if (score1 == 10 || score2 == 10) { quit = true; } }

**void Logic ()** { ball->move(); player1->move (); player2->move (); int ballX = ball->getX (); int ballY = ball->getY (); int player1x = player1->getX (); int player1y = player1->getY (); int player2x = player2->getX (); int player2y = player2->getY (); if (ballX == player1x + 1) { if (ballY == player1y - 3) { ball->changeDirection(UPRIGHT); } else if (ballY == player1y - 2 || ballY == player1y - 1) { ball->changeDirection(static\_cast<eDirection>((rand() % 3) + 1)); } else if (ballY == player1y) { ball->changeDirection(DOWNRIGHT); } } else if (ballX == player2x - 1) { if (ballY == player2y - 3) { ball->changeDirection(UPLEFT); } else if (ballY == player2y - 2 || ballY == player2y - 1) { ball->changeDirection(static\_cast<eDirection>((rand () % 3) + 4)); }

1. **Algorithm:**

* Take input from both players to enter their names.
* Initialize the game screen with paddles, ball, and score display.
* Set the positions of the paddles and ball to the center at the start of the game.
* Capture user input for Player 1’s paddle movement using "w" and "s".
* Capture user input for Player 2’s paddle movement using "i" and "j".
* If key = "w", move Player 1’s paddle up.
* If key = "s", move Player 1’s paddle down.
* If key = "i", move Player 2’s paddle up.
* If key = "j", move Player 2’s paddle down.
* If key = "q", quit the game.
* Update the ball's movement direction in every iteration.
* Check for collisions between the ball and paddles.
* Reflect the ball if it hits the top or bottom walls.
* Increment player scores if the ball passes through the left or right walls.
* If score1 == 10 and score2 == 10, display "It's a tie!".
* Else if score1 == 10, display "Player 1 wins".
* Else if score2 == 10, display "Player 2 wins".
* If the user quits (key = 'q'):
* If score1 > score2, display "Player 1 wins".
* Else if score2 > score1, display "Player 2 wins".
* Else, display "It's a tie!".
* Reset the positions of the ball and paddles after each score.
* The game loop continues until one of the players reaches a score of 10, or both reach 10 simultaneously, or the user chooses to qu.

# 

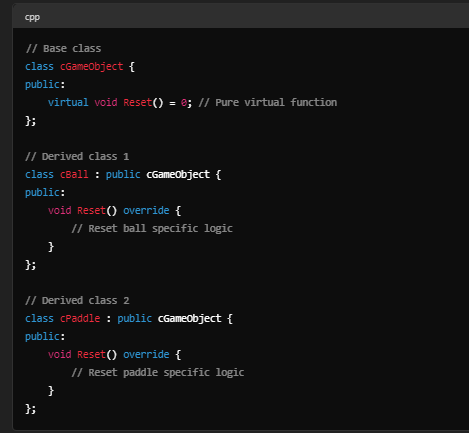
# **6.Concepts used to write the Code:**

1. **Inheritance:**

* **Base Class:** cGameObject serves as the base class with common properties like position and methods for managing these properties.
* **Derived Classes:** cBall and cPaddle **inherit** from cGameObject, gaining access to its methods and properties. They extend these by adding specific functionalities like ball movement and paddle control.

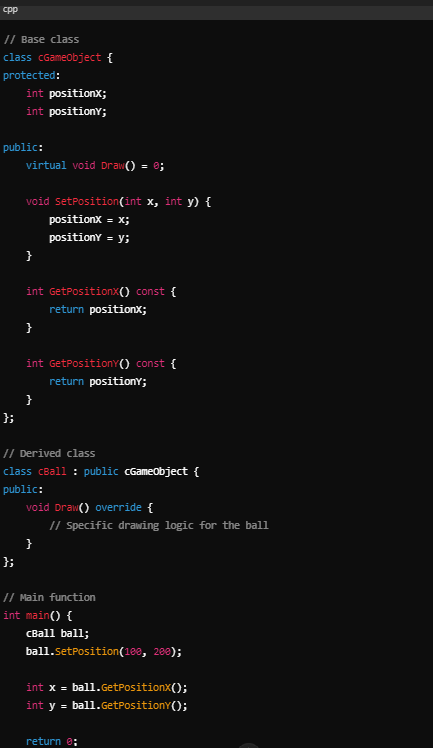
1. **Polymorphism:**

**cGameObject** is an abstract base class with a pure virtual function Reset (). Both **cBall** and **cPaddle** inherit from cGameObject and provide their own implementations of the **Reset ()** function, showcasing **polymorphism** where objects of different derived types (cBall and cPaddle) can be treated as objects of the base class (cGameObject).



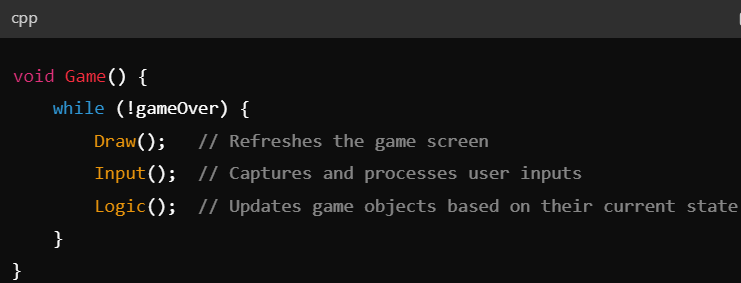
1. **Data Abstraction:**

cGameObject provides a base class with properties (positionX and positionY) and methods (SetPosition, **GetPositionX, GetPositionY**) for managing these properties. The Draw method is declared as a pure virtual function (= 0), enforcing **abstraction** and requiring derived classes like cBall to implement their own Draw method. This structure hides the internal details of how positions are managed and accessed, promoting code clarity and **encapsulation.**



1. **Loops:**

The Game () method in cGameManager class uses a **while** **loop**, ensuring that the game continuously updates the screen (**Draw ()**), handles user input (**Input ()**), and updates the game logic (**Logic ()**) until the game is over (game over condition becomes true).



1. **Conditional Statements (If-else):**

 In cGameManager, conditional statements are used to check for collisions the ball and paddles (if (ball collides with paddle)), walls (if (ball collides with top or bottom wall)), and to update the game state based on these conditions (if (score condition is met)).

1. **Arrays**:

Arrays are not directly utilized in the ball game code.

1. **File Handling/Database:**

The provided code focuses on real-time game interaction and does not involve file handling or database operations.

1. **APIs:**

The provided code does not utilize external APIs. It relies on standard C++ libraries for basic input/output operations and console manipulation.

# **Output of the Code:**

# **Challenges faced during the Project:**

In the pursuit of developing a functional and engaging game, several challenges were encountered across different phases of the project.

* Carefully designing game components using Object-Oriented Design (OOD) was essential to ensure clarity, maintainability, and scalability. By breaking down the game into manageable modules—such as paddles, the ball, and game rules—the development process was structured for easier maintenance and future updates.
* Ensuring players could easily control the game on any device was essential for a smooth gaming experience. We achieved this by carefully designing and thoroughly testing how players interacted with the game. This ensured controls were consistently responsive, enhancing gameplay across all platforms.
* Smooth transitions between game states—such as playing, pausing, or ending—were pivotal in maintaining a consistent and engaging game flow. Extensive testing was conducted to identify and resolve any issues, ensuring uninterrupted gameplay and enhancing overall player satisfaction.

1. **Conclusion:**

This project underscores the importance of designing software with a focus on maintainability, extensibility, and user experience. By leveraging OOP, the Pong game demonstrates how encapsulation, inheritance, and polymorphism can contribute to building a robust and enjoyable gaming experience.