

# *Object Oriented Programming*

## Semester Project

**Instructor:**  
**Mr. Tariq Mahmood**

Prepared By:

**Name:** Abdul Moeez

**Reg#:** 241804

**Chess Game in C++ Using SFML**

## 1. Introduction

This project is a two-player 2D chess game developed using **C++ and SFML (Simple and Fast Multimedia Library)**. The goal of this project is to demonstrate object-oriented programming principles, user interaction design, and simple game logic. Players can input their names, play a full game of chess with legal move enforcement, highlighting, and winning detection, and navigate between menus using intuitive controls.

## 2. Dependencies and Setup Requirements

To run the game successfully, you need the following:

- **SFML 2.5 or higher**
- **C++17 compatible compiler (G++, Clang++, MSVC)**
- **CMake (optional)**
- **Assets folder** containing piece textures: Assets/White Pieces/, Assets/Black Pieces/
- **Fonts folder** containing a TTF font for UI text
- **IDE** such as Visual Studio, CLion, or Code::Blocks (or terminal-based build)

---

## 3. Module Breakdown with Code

### 3.1 Game Initialization and Window Setup

```
RenderWindow window(VideoMode(1000, 800), "Chess Game");
Font font;
font.loadFromFile("Assets/Font/arial.ttf");
```

The window is initialized to 1000x800 pixels. A custom font is loaded from the assets folder for use in buttons, labels, and messages.

---

### 3.2 Board Rendering Module

```
RectangleShape board[8][8];
for (int r = 0; r < 8; r++)
    for (int c = 0; c < 8; c++) {
        board[r][c].setSize(Vector2f(tileSize, tileSize));
        board[r][c].setPosition(boardStartX + c * tileSize, boardStartY + r *
tileSize);
        board[r][c].setFillColor((r + c) % 2 == 0 ? Color(118, 150, 86) :
Color(238, 238, 210));
    }
```

This code generates an 8x8 chessboard by alternating light and dark squares.

---

### 3.3 Texture Loading for Pieces

```
Texture textures[12];
textures[W_P].loadFromFile("Assets/White Pieces/Pawn.png");
// ... similar for all other pieces ...
```

Piece textures for all white and black pieces are loaded from appropriate folders.

---

### 3.4 Game State and UI Elements

```
Text turnText("", font, 28);
turnText.setPosition(boardStartX, boardStartY + boardHeight + 20);
Button menuButton(Vector2f(120, 40), Vector2f(boardStartX + boardWidth - 120,
boardStartY + boardHeight + 20), "Main Menu", font);
```

This sets up UI elements like turn indicators, the menu button, and winning messages.

---

### 3.5 Main Menu and Name Input Handling

```
if (gameState == MAIN_MENU) {
    newGameButton.update(mousePos);
    exitButton.update(mousePos);
    if (event.type == Event::MouseButtonPressed && event.mouseButton.button ==
Mouse::Left) {
        if (newGameButton.isClicked(mousePos)) gameState = NAME_INPUT;
    }
}
```

This block handles UI transitions from the main menu to name input or exits the game.

---

### 3.6 Name Input Screen

```
if (gameState == NAME_INPUT) {
    player1Input.update(mousePos);
    player2Input.update(mousePos);
    if (startGameButton.isClicked(mousePos)) {
        player1Name = player1Input.content;
        player2Name = player2Input.content;
        gameState = PLAYING;
        setupPieces(pieces, textures, boardStartX, boardStartY, tileSize);
    }
}
```

Allows players to input their names and initializes the chessboard when both names are entered.

---

### 3.7 Gameplay Logic and Piece Movement

```
if (!pieceSelected) {
    Piece *p = pieces[pos.y][pos.x];
    if (p && p->isWhite() == whiteTurn) {
        selected = pos;
        pieceSelected = true;
        moves = p->getLegalMoves(pos, pieces);
    }
} else {
    for (auto &m : moves) {
        if (m == pos) {
            delete pieces[pos.y][pos.x];
            pieces[pos.y][pos.x] = pieces[selected.y][selected.x];
            pieces[selected.y][selected.x] = nullptr;
            pieces[pos.y][pos.x]->setPosition(...);
            whiteTurn = !whiteTurn;
        }
    }
    pieceSelected = false;
    moves.clear();
}
```

This handles click-based piece selection and legal movement validation. Capturing is done by deleting the existing piece at the target square.

---

### 3.8 Win Checking

```
if (checkForWin(pieces, whiteWins)) {  
    gameOver = true;  
    string winner = whiteWins ? player1Name + " Wins!" : player2Name + " Wins!";  
    winMessage.setString(winner);  
}
```

The game checks for a win after every move and displays the winner message if a condition is met.

---

### 3.9 Cleanup on Exit

```
cleanupPieces(pieces);  
return 0;
```

Releases dynamically allocated memory for all pieces and cleanly exits the application.

---

## 4. Conclusion

This 2D chess game project successfully combines graphical rendering with object-oriented game logic. With modular design, texture loading, user input, and legal chess move enforcement, it provides a clean example of game development using SFML and C++. The game includes:

- A menu and name input system
- Interactive gameplay with legal move validation
- UI feedback for current turn and winner announcement

This project can be extended further by implementing advanced features such as:

- AI opponents
- Move history
- Checkmate/stalemate logic
- Save/load functionality

The project demonstrates practical C++ and SFML integration and provides a good foundation for intermediate game development projects.

**AIR UNIVERSITY**