System Programming Course
Project Documentation: Tic Tac Toe Game

Student:Seidaly Rauan
Group: IT2-2104
GitHub Project

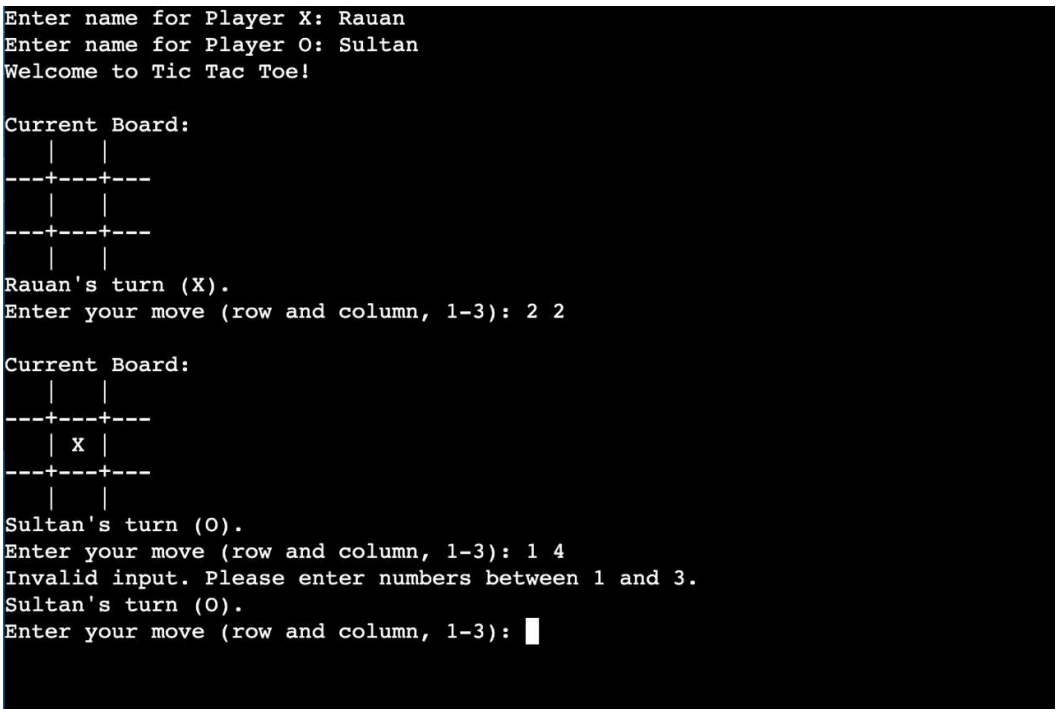
Almaty
2024

Introduction

The **Tic Tac Toe Game** is a simple console-based game developed in C++ that allows two players to play the classic 3x3 grid-based game. The program provides a turn-based gameplay experience with features like checking for wins, draws, and replaying the game. This project demonstrates the use of object-oriented programming concepts, including classes, encapsulation, and member functions.

Features

1. **Two-player gameplay:** Players X and O take turns making moves.
2. **Custom player names:** Players can enter their names at the start of the game.
3. **Dynamic board display:** The board updates after every valid move.
4. **Win and draw detection:** The game checks for winning conditions or a draw after every move.
5. **Replay option:** Players can choose to play again after a game ends.
6. **Input validation:** Ensures proper move inputs and prevents overwriting cells.



Requirements

- **Programming Language:** C++
- **Compiler:** Any standard C++ compiler supporting C++11 or later.
    - Examples: GCC, Clang, or MSVC.
- **Development Environment:** Optional (Visual Studio, CLion, Code::Blocks, etc.)

Libraries Used

This project uses standard libraries from the C++ Standard Library:

1. **<iostream>**
    - For input and output operations.
    - Example: cin and cout.
2. **<vector>**
    - For representing the 3x3 Tic Tac Toe board as a 2D grid.
3. **<string>**
    - For storing and manipulating player names.

Example:

```
#include <iostream>

#include <vector>

#include <string>

using namespace std;

class TicTacToe {

    vector<vector<char>> board;

    char currentPlayer;

    string playerXName;

    string playerOName;
```

---

additional 1

Class Overview

Class: TicTacToe *(additional 1)*

Encapsulates the game logic and data, ensuring a modular design.

Attributes:

- vector<vector<char>> board: 2D vector representing the game board.
- char currentPlayer: Keeps track of the current player (X or O).
- string playerXName: Name of Player X.
- string playerOName: Name of Player O.

Methods:

- **TicTacToe()**: Constructor that initializes the board and sets the current player. (*additional 2*)

```
TicTacToe() : board(3, vector<char>(3, ' ')), currentPlayer('X') {}
```

---

additional 2

- **void initializeGame()**: Resets the board and gets player names. *(additional 3)*

```
void initializeGame() {
        for (auto& row : board)
            fill(row.begin(), row.end(), ' ');
        cout << "Enter name for Player X: ";
        cin >> playerXName;
        cout << "Enter name for Player O: ";
        cin >> playerOName;
              currentPlayer = 'X
```

---

additional 3

- **void displayBoard()**: Displays the current state of the board.

```
void displayBoard() {
        cout << "\nCurrent Board:\n";
        for (int i = 0; i < 3; ++i) {
            for (int j = 0; j < 3; ++j) {
                cout << " " << board[i][j] << " ";
                if (j < 2) cout << "|";
            }
            cout << endl;
            if (i < 2) cout << "---+---+---\n";
        }
    }
```

---

additional 4

- **bool makeMove(int row, int col)**: Marks the board with the current player's symbol if the move is valid. *(additional 5)*

```
bool makeMove(int row, int col) {
        if (row < 0 || row >= 3 || col < 0 || col >= 3 ||
board[row][col] != ' ') {
            cout << "Invalid move. Try again.\n";
            return false; }
        board[row][col] = currentPlayer;
        return true; }
```

---

additional 5

- **bool checkWin()**: Checks if the current player has won. *(additional 6)*

```
 bool checkWin() {

        for (int i = 0; i < 3; ++i) {

            // Check rows and columns

            if ((board[i][0] == currentPlayer && board[i][1] ==
currentPlayer && board[i][2] == currentPlayer) ||

                (board[0][i] == currentPlayer && board[1][i] ==
currentPlayer && board[2][i] == currentPlayer))

                return true;  }

        // Check diagonals

        if ((board[0][0] == currentPlayer && board[1][1] ==
currentPlayer && board[2][2] == currentPlayer) ||

            (board[0][2] == currentPlayer && board[1][1] ==
currentPlayer && board[2][0] == currentPlayer))

            return true;

        return false;  }
```

---

additional 6

- **bool checkDraw()**: Checks if the game is a draw. *(additional 7)*

```
bool checkDraw() {

        for (const auto& row : board) {

            for (char cell : row) {

                if (cell == ' ') return false;

            }

        }

        return true; }
```

---

additional 7

- **void switchPlayer()**: Switches the turn to the other player. (*additional 8)*

```
void switchPlayer() {
        currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
    }
```

additional 8

- **char getCurrentPlayerSymbol()**: Returns the current player's symbol. (*additional 9)*

```
 char getCurrentPlayerSymbol() const {
        return currentPlayer;
    }
```

additional 9

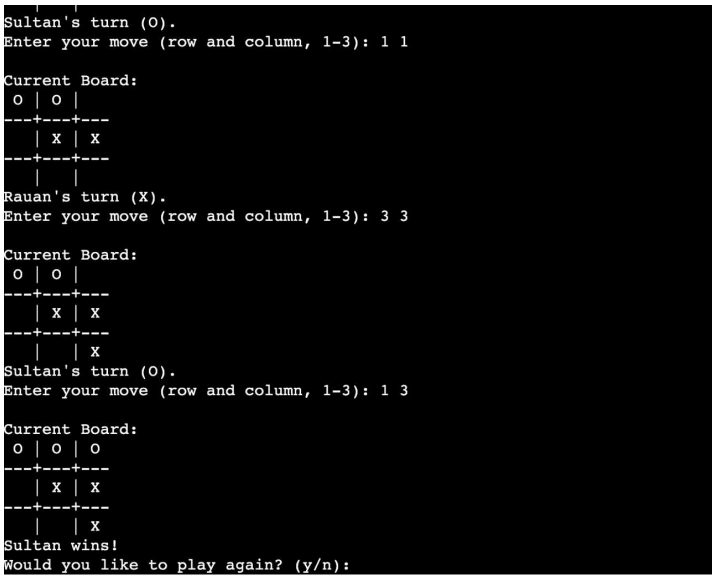
- **string getCurrentPlayerName()**: Returns the current player's name. (*additional 10)*

```
string getCurrentPlayerName() const {
        return (currentPlayer == 'X') ? playerXName : playerOName;
    }
```

additional 10

Game Flow

1. **Initialization:**
    - The playGame() function starts the game loop.
    - Players are prompted to enter their names.
    - The board is displayed.
2. **Gameplay Loop:**
    - Each player takes turns making a move.
    - Input is validated to ensure correctness.
    - After a move:
        - The board updates.
        - The program checks for a win or draw.
        - If neither occurs, the turn switches to the other player.
3. **Game End:**
    - If a win or draw is detected, the result is displayed.
    - Players are given the option to replay.



additional 11

Conclusion

This project provides a hands-on implementation of fundamental programming concepts in C++. It is a fun and interactive way to practice OOP, input validation, and basic game logic while setting the stage for more complex game development projects.