Data Structures and Object-oriented Programming (EC-204)

DE 44 Mechatronics

Syndicate – C

Project Report

"Tic-Tac-Toe Game"

Team Members:

- NC Muhammad Ahmad Kashif
  Reg # 415199
- NC Saim Raza
  Reg # 412868

Submitted to:

Assoc Prof Dr. Tahir Nawaz

LE Hamza Sohail

Table of Contents

- **Abstract ……..……..…….. 3**
- **Objectives ……..……...…… 3**
- **Project Description ……... 3**
- **Key Features ….…… 4**
- **Code Explanation ....……..... 4**
- **Flowchart Diagram ……. 10**
- **Results and Observations ……... 11**
- **Conclusion ..……. 12**

Tic-Tac-Toe Game

Abstract:

The Tic Tac Toe Game Project is an object-oriented programming (OOP) adaptation of the popular Tic Tac Toe game in C++. The project's goal is to deliver a simple and intuitive console-based gaming experience. The architecture stresses code modularity, reusability, and maintainability by leveraging inheritance and polymorphism principles. This paper delves into the development process, significant features, and insights obtained while making this C++ Tic Tac Toe game.

Objective:

The objective of the C++ Tic Tac Toe Game Project utilizing OOP ideas is to illustrate modular code design, encapsulation, and polymorphism. The project stresses accessibility, error management, and a clear game logic by developing a console-based interface. The goal is to present a realistic example of OOP principles in the form of a simple, instructive game.

Project Description:

The Tic Tac Toe Game Project is a console-based program that aims to recreate the popular Tic Tac Toe game. To structure the codebase, the major programming paradigm used is object-oriented, with an emphasis on encapsulation, inheritance, and polymorphism.

The game is played in a two-player mode, with players taking turns marking cells on a 3x3 grid. Classes are used in the project to model the game board, participants, and the game itself. Inheritance is used to construct specialized classes for various areas of the game, encouraging code reuse and establishing a clear hierarchy.

Key Features:

- **Modular Design:** The project is divided into classes that represent different elements like the game board, players, and the overall game. This modular approach improves code readability and allows for future changes or extensions.
- **Inheritance:** Class ties are established by inheritance. The base class captures general traits and behaviors, whereas derived classes inherit these aspects and add unique functions, hence encouraging code reuse.
- **Polymorphism:** Polymorphism is used to provide versatility in dealing with various game aspects. This allows the software to treat objects of different classes consistently, which simplifies implementation and improves maintainability.
- **Console Interaction:** The game is totally console-based, with a simple and user-friendly interface. Player input is recorded using standard input/output, assuring platform compatibility.
- **Game Logic:** To handle player actions, check for victory criteria, and control the overall flow of the game, the project includes a powerful game logic implementation. This logic is intended to be clear and efficient.
- **Error Handling:** The program provides error-handling features to resolve erroneous user inputs or unforeseen scenarios during gameplay, increasing the application's overall robustness.

Code Explanation:

This C++ program implements a console-based Tic Tac Toe game using object-oriented programming (OOP) principles. Here's a breakdown of the code:

1. **Game Class:**
    - **Attributes:**
        - **board[3][3]:** Represents the Tic Tac Toe board.
        - **turn:** Tracks the current turn.
        - **currentPlayer:** Stores the symbol ('X' or 'O') of the current player.

```
6   class Game
7   {
8   protected:
9       char board[3][3];
10      int turn;
11      char currentPlayer;
```

    - **Methods:**
        - **start():** Initiates the game loop, taking turns between players until a win or tie occurs.

```
public:
    Game()
    {
    }

    virtual void start()
    {
        turn = 0;
        currentPlayer = 'X';
        printBoard();

        for (turn = 0; turn < 3; turn++)
        {
            cout << "Player " << currentPlayer << "'s turn" << endl;
            getMove();
            printBoard();
            if (turn == 2)
            {
                if (checkWin())
                {
                    break;
                }
                turn++;
            }
            togglePlayer();
            cout << "Player " << currentPlayer << "'s turn" << endl;
            getMove();
            printBoard();
            if (turn == 3 && checkWin())
            {
                break;
            }
            if (turn == 3 && checkTie())
            {
                break;
            }
            togglePlayer();
        };
        showResult();
    }
```

        - **printBoard():** Pure virtual function to be implemented by derived classes. Displays the current state of the board.

- **getMove():** Pure virtual function to be implemented by derived classes. Collects player input for row and column and updates the board accordingly.
- **checkWin():** Pure virtual function to be implemented by derived classes. Checks if the current player has won.
- **checkTie():** Pure virtual function to be implemented by derived classes. Checks if the game is a tie.
- **togglePlayer():** Pure virtual function to be implemented by derived classes. Switches the current player between 'X' and 'O'.
- **showResult():** Pure virtual function to be implemented by derived classes. Displays the result of the game.

```
53
54      virtual void printBoard() = 0;
55      virtual void getMove() = 0;
56      virtual bool checkWin() = 0;
57      virtual bool checkTie() = 0;
58      virtual void togglePlayer() = 0;
59      virtual void showResult() = 0;
60      virtual bool emptyCheck() = 0;
61  };
```

2. **TicTacToe Class (Derived from Game):**

- **Constructor:**
  Initializes the Tic Tac Toe board using the **initializeBoard()** function.

```
63  class TicTacToe : public Game
64  {
65  public:
66      TicTacToe() : Game()
67      {
68          initializeBoard();
69      }
70
71  private:
72      void initializeBoard()
73      {
74          for (int i = 0; i < 3; i++)
75          {
76              for (int j = 0; j < 3; j++)
77              {
78                  board[i][j] = ' ';
79              }
80          }
81      }
82
```

- **Overridden Methods:**
  - **printBoard():** Displays the Tic Tac Toe board with the current state.

```
83      void printBoard() override
84      {
85          cout << "\t\t " << board[0][0] << "\t|" << board[0][1] << "\t|" << board[0][2] << "\n";
86          cout << "\t\t_____________________\n";
87          cout << "\t\t " << board[1][0] << "\t|" << board[1][1] << "\t|" << board[1][2] << "\n";
88          cout << "\t\t_____________________\n";
89          cout << "\t\t " << board[2][0] << "\t|" << board[2][1] << "\t|" << board[2][2] << "\n";
90      }
```

  - **getMove():** Collects player input for row and column, handling exceptions for invalid input.

```
92      void getMove() override
93      {
94          int row, col;
95
96          while (true)
97          {
98              try
99              {
100                 cout << "Enter the row: ";
101                 cin >> row;
102
103                 if (row < 1 || row > 3)
104                 {
105                     throw out_of_range("Row out of range. Please enter a number between 1 and 3.");
106                 }
107
108                 cout << "Enter the column: ";
109                 cin >> col;
110
111                 if (col < 1 || col > 3)
112                 {
113                     throw out_of_range("Column out of range. Please enter a number between 1 and 3.");
114                 }
115
116                 if (board[row - 1][col - 1] == ' ')
117                 {
118                     break;
119                 }
120                 else
121                 {
122                     throw invalid_argument("Cell already occupied. Try again.");
123                 }
124             }
125             catch (const exception &e)
126             {
127                 cerr << "Error: " << e.what() << endl;
128                 cin.clear();
129             }
130         }
131
132         board[row - 1][col - 1] = currentPlayer;
133     }
```

- **checkWin():** Checks for a win by examining rows, columns, and diagonals.

```
135     bool checkWin() override
136     {
137         bool victory = false;
138         for (int i = 0; i < 3 && (turn >= 2 || turn <= 4); i++)
139         {
140             if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && (board[i][0] != ' '))
141             {
142                 victory = true;
143                 return victory;
144             }
145 
146             if (board[0][i] == board[1][i] && board[1][i] == board[2][i] && (board[i][0] != ' '))
147             {
148                 victory = true;
149                 return victory;
150             }
151         }
152 
153         if (board[0][0] == board[1][1] && board[1][1] == board[2][2])
154         {
155             victory = true;
156             return victory;
157         }
158 
159         if (board[0][2] == board[1][1] && board[1][1] == board[2][0])
160         {
161             victory = true;
162             return victory;
163         }
164 
165         return victory;
166     }
```

- **checkTie():** Checks for a tie by verifying if there are no empty cells left.

```
168         bool checkTie() override
169         {
170             if (!checkWin() && emptyCheck())
171             {
172                 return true;
173             }
174             return false;
175         }
```

- **togglePlayer():** Switches the current player between 'X' and 'O'.

```
177         void togglePlayer() override
178         {
179             currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
180         }
181 
```

- **showResult():** Displays the result of the game.

```
182     void showResult() override
183     {
184         if (checkWin())
185         {
186             cout << "Player " << currentPlayer << " wins!" << endl;
187         }
188         else
189         {
190             cout << "It's a tie!" << endl;
191         }
192     }
```

- **Additional Method:**
  - **emptyCheck():** Checks if the board has any empty cells.

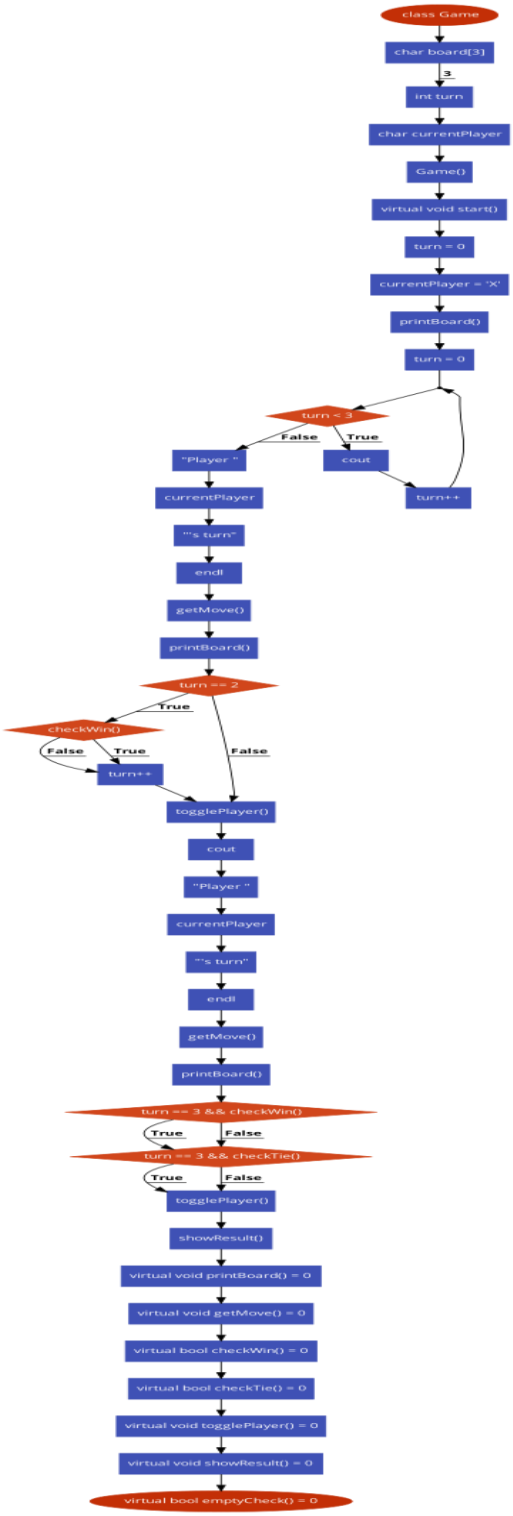
```
193     bool emptyCheck() override
194     {
195         bool empty = false;
196         for (int i = 0; i < 3; i++)
197         {
198             for (int j = 0; j < 3; j++)
199             {
200                 if (board[i][j] == ' ')
201                 {
202                     empty = true;
203                     return empty;
204                     break;
205                 }
206             }
207             if (empty)
208             {
209                 break;
210             }
211         }
212         return empty;
213     }
```

3. **Main Function:**

   Creates an instance of the **TicTacToe** class and initiates the game by calling **start()**.

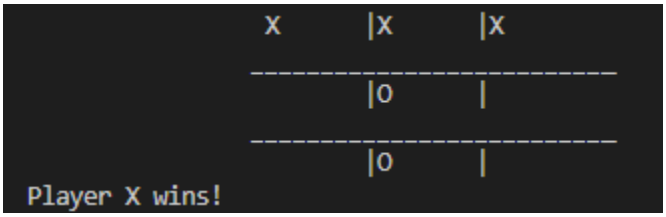
```
216 int main()
217 {
218     TicTacToe game;
219     game.start();
220
221     return 0;
222 }
```

Flowchart Diagram:

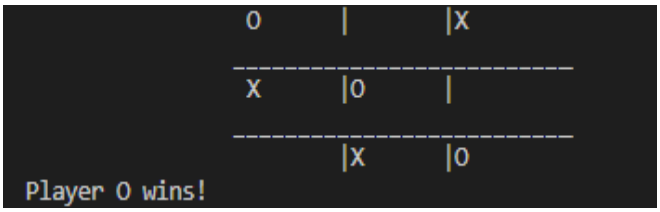


Results and Observations:

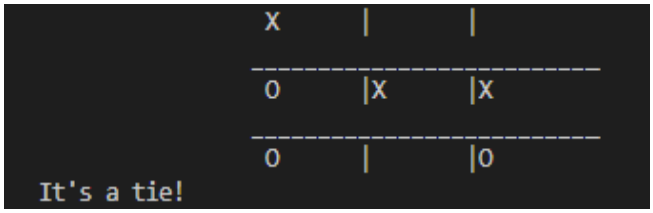
1. Player X wins:



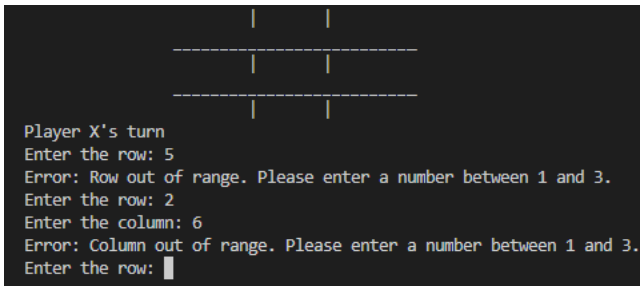
2. Player O wins:



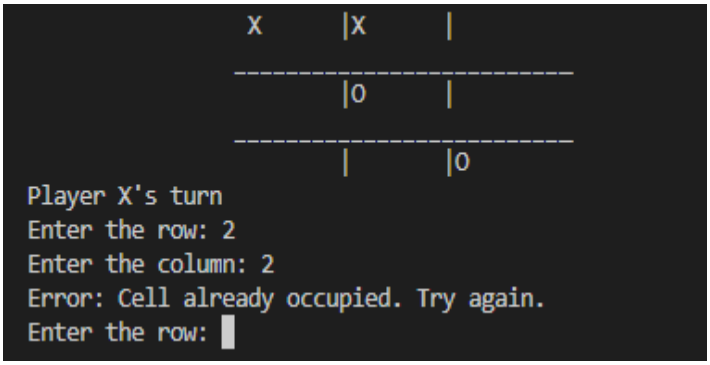
3. Tied:



4. Error: Invalid Entry:



5. **Error: Occupied Move:**



Consclusion:

In conclusion, this program shows how to construct a Tic Tac Toe game utilizing OOP concept such as inheritance and polymorphism. The game is played on a console, with participants taking turns entering their movements until a winner or a tie is established. Error management is included to ensure correct user input.