# Fundamentals of Programming

## End Semester Project

## *~~~ Tick-Tack-Toe ~~~*

**ME-15**

**Section A**


**Group Members:**

**Faizan Ahmad** **(476602)**

**Abdullah Tahseen** **(462573)**

**Muhammad Bin Ahmad** **()** (480779)

## Introduction:

This report explores the implementation of a Tic-Tac-Toe game in C++, featuring a human player and a computer opponent. The game is presented through a console interface and incorporates strategic moves by the computer, providing an engaging experience for the player. The code delves into the mechanics of the game, showcasing elements such as board display, player input validation, and an AI opponent with various move strategies.

## Explanation of the code:

The code is organized into several functions to handle different aspects of the game. Here's an overview of each function:

- displayboard:
  This function is responsible for rendering the Tic-Tac-Toe board on the console. It takes an array representing the board as input and prints the current state with placeholders for X and O.

- wincheck:
  Determines if there is a winner by checking all possible win conditions: three in a row (horizontally, vertically, or diagonally). Returns **true** if a win is detected, otherwise **false**.

- drawcheck:
  Checks if the game is a draw (i.e., no winner and the board is filled). Returns **true** for a draw, otherwise **false**.

- computer_rowmoves, computer_columnmoves, computer_diagonalmoves:
  These functions represent different strategies for the computer's moves. They prioritize blocking the player or winning the game by checking rows, columns, and diagonals.

- random_move:
  Provides a fallback option for the computer to make a random move if the strategic moves are not applicable.

- main:
  The main function orchestrates the game loop, alternating turns between the player and the computer. It utilizes a character array (**board**) to represent the state of the Tic-Tac-Toe board. The player is prompted to input a move (a number corresponding to an empty cell on the board), with thorough validation to ensure a legal move. After

the player's move, the computer makes its move based on the defined strategies. The game continues until there is a winner, a draw, or the player decides to exit.

**Player Input Handling:**

The player's input is handled through a series of **switch** statements, ensuring that the entered move is valid (i.e., an empty cell on the board). If the player enters an invalid move, they are prompted to input a valid one.

**Computer Move Strategies:**

The computer employs strategies to make informed moves:

- **Rows, Columns, Diagonals:** The computer checks for opportunities to complete a row, column, or diagonal and win the game. It also checks for the player doing the same to block them.
- **Random Move:** If no strategic move is available, the computer makes a random move.

**Game Termination:**

The game loop continues until there is a winner or a draw. Once the game concludes, the program prints a corresponding message, acknowledging whether the player won, the computer won, or the game ended in a draw.

**Randomization:**

The **rand()** function, seeded with the current time, is used for generating random numbers. This is primarily used in the **random_move** function to make a random move when needed.

## Code:

```
#include <bits/stdc++.h>

using namespace std;


void displayboard(char board[9]) {
   cout<<"     |     |     \n";
   cout<<"   "<<board[0]<<" | "<<board[1]<<"   |  "<<board[2]<<" \n";
   cout<<"_____|_____|_____\n";
   cout<<"     |     |     \n";
   cout<<"   "<<board[3]<<" | "<<board[4]<<"   |  "<<board[5]<<" \n";
   cout<<"_____|_____|_____\n";
```

```
   cout<<"     |     |     \n";
   cout<<"   "<<board[6]<<" | "<<board[7]<<"   |  "<<board[8]<<" \n";
   cout<<"     |     |     \n";
}

bool wincheck(char board[9]){
if(board[0]==board[1] && board[1]==board[2]){
return true;
}
else if(board[3]==board[4] && board[4]==board[5]){
return true;
}
else if(board[6]==board[7] && board[7]==board[8]){
return true;
}
//                    end of rows check
else if(board[0]==board[3] && board[3]==board[6]){
return true;
}
else if(board[1]==board[4] && board[4]==board[7]){
return true;
}
else if(board[2]==board[5] && board[5]==board[8]){
return true;
}
//                    end of column check
else if(board[0]==board[4] && board[4]==board[8]){
return true;
}
```

```
else if(board[2]==board[4] && board[4]==board[6]){
return true;
}
else {
return false;
}
//                    end of diagonal check
}

bool drawcheck(char board[9]){
if (!wincheck(board)) {
      for (int i = 0; i < 9; i++) {
         if (board[i] != 'X' && board[i] != 'O') {
            return false;
         }
      }
      return true;
   }
   return false;
}

bool computer_rowmoves(char board[9]){
if (board[4]!='X'&& board[4]!='O'){
board[4]='O';
return true;
}
else if(board[0]==board[1] && board[2]=='3' ){
board[3]='O';
return true;
```

```
}
else if (board[1]==board[2] && board[0]=='1'){
board[0]='O';
return true;
}
else if(board[3]==board[4] && board[5]=='6'){
board[5]='O';
return true;
}
else if (board[4]==board[5] && board[3]=='4'){
board[3]='O';
return true;
}
else if(board[6]==board[7] && board[8]=='9'){
board[8]='O';
return true;
}
else if (board[7]==board[8] && board[6]=='7'){
board[6]='O';
return true;
}
else if (board[0]==board[2] && board[1]=='2'){
board[1]='O';
return true;
}
else if(board[3]==board[5] && board[4]=='5'){
board[4]='O';
return true;
}
```

```
else if (board[6]==board[8] && board[7]=='8'){
board[7]='O';
return true;
}
else {
return false;
}
//                    end of rows check



}

bool computer_columnmoves(char board[9]){
if (board[4]!='X'&& board[4]!='O'){
board[4]='O';
return true;
}
else if(board[0]==board[3] && board[6]=='7' ){
board[6]='O';
return true;
}
else if (board[3]==board[6] && board[0]=='1'){
board[0]='O';
return true;
}          // end of column 1
else if(board[1]==board[4] && board[7]=='8'){
board[7]='O';
return true;
}
```

```
else if (board[4]==board[7] && board[1]=='2'){
board[1]='O';
return true;
}           // end of column 2
else if(board[2]==board[5] && board[8]=='9'){
board[8]='O';
return true;
}
else if (board[5]==board[8] && board[2]=='3'){
board[2]='O';
return true;
}           // end of column 3
else if (board[0]==board[6] && board[3]=='4'){
board[3]='O';
return true;
}
else if(board[1]==board[7] && board[4]=='5'){
board[4]='O';
return true;
}
else if (board[2]==board[8] && board[5]=='6'){
board[5]='O';
return true;
}
else{
return false;
}
}
```

```
bool computer_diagonalmoves(char board[9]){
if(board[0]==board[4] && board[8]=='9' ){
board[8]='O';
return true;
}
else if (board[0]==board[8] && board[4]=='5'){
board[4]='O';
return true;
}          // end of column 1
else if(board[4]==board[8] && board[0]=='1'){
board[0]='O';
return true;
}
else if (board[2]==board[4] && board[6]=='7'){
board[6]='O';
return true;
}          // end of column 2
else if(board[4]==board[6] && board[2]=='3'){
board[2]='O';
return true;
}
else if (board[2]==board[6] && board[4]=='5'){
board[4]='O';
return true;
}
else{
return false;
}
}
```

```
bool random_move(char board[9]){
 while (true) {
       int random = rand() % 9;
       if (board[random] == '1' + random) {
          board[random] = 'O';
          return true;
       }
   }
}

int main(){
srand(time(0));
char board[9]={'1','2','3','4','5','6','7','8','9'};
bool valid;
char move;
while(!wincheck(board) && !drawcheck(board)){
cout<<"Your turn, enter the square number which you want to mark.\n";
displayboard(board);
cin>>move;
valid=false;
while (valid==false){
switch(move){
case '1': if(board[0]=='1'){
board[0]='X';
valid =true;
break;
}
else{
```

```
valid=false;
cin>>move;
}
break;
case '2': if(board[1]=='2'){
board[1]='X';
valid =true;
break;
}
else{
valid=false;
cin>>move;
}
break;
case '3': if(board[2]=='3'){
board[2]='X';
valid =true;
break;
}
else{
valid=false;
cin>>move;
}
break;
case '4': if(board[3]=='4'){
board[3]='X';
valid =true;
break;
}
```

```
else{
valid=false;
cin>>move;
}
break;
case '5': if(board[4]=='5'){
board[4]='X';
valid =true;
break;
}
else{
valid=false;
cin>>move;
}
break;
case '6': if(board[5]=='6'){
board[5]='X';
valid =true;
break;
}
else{
valid=false;
cin>>move;
}
break;
case '7': if(board[6]=='7'){
board[6]='X';
valid =true;
break;
```

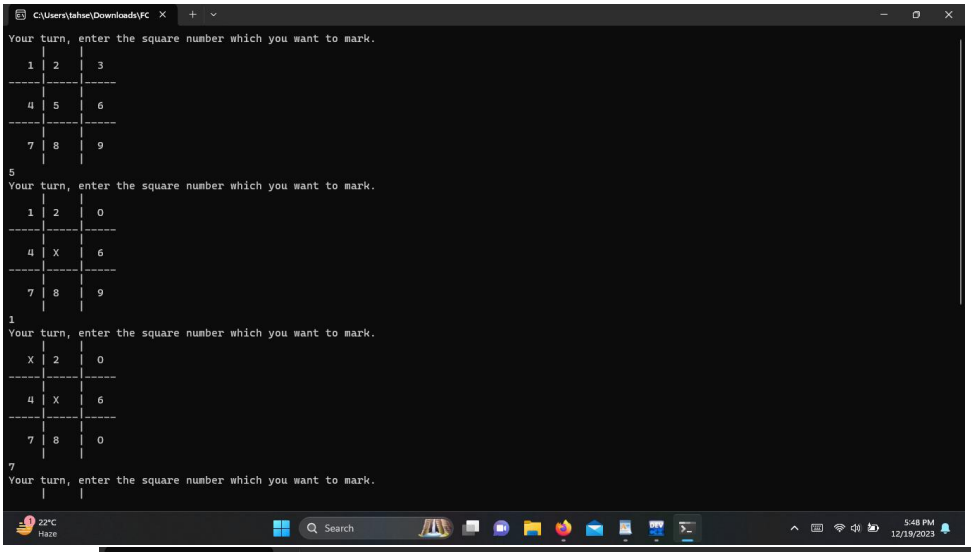
```
}
else{
valid=false;
cin>>move;
}
break;
case '8': if(board[7]=='8'){
board[7]='X';
valid =true;
break;
}
else{
valid=false;
cin>>move;
}
break;
case '9': if(board[8]=='9'){
board[8]='X';
valid =true;
break;
}
else{
valid=false;
cin>>move;
}
break;
default: cout<<"Invalid move, enter again.\n";
cin>>move;
break;
```

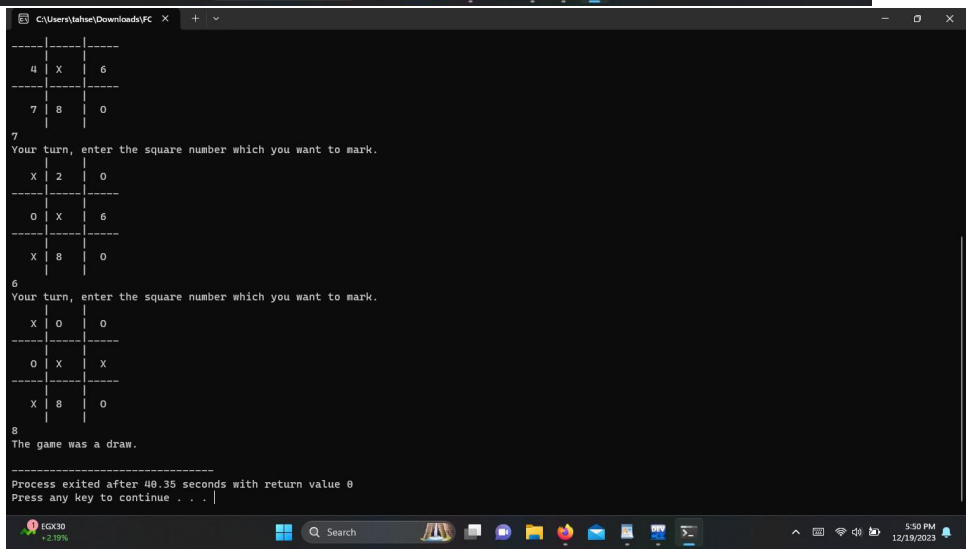
```
}
}
valid=true;
if(wincheck(board)){
displayboard(board);
cout<<"CONGRATULATIONS, YOU JUST BEAT MY PROGRAM WHICH I WOKRED ON FOR DAYS.";
break;
}
if (drawcheck(board)){
cout<<"The game was a draw.\n";
break;
}
if (!computer_rowmoves(board)) {
      if (!computer_columnmoves(board)) {
         if(!computer_diagonalmoves(board)){
                        random_move(board);
}
}
      }

if(wincheck(board)){
displayboard(board);
cout<<"The AI won, better luck next time.\n";
break;
}
}
}
```

## Executes and Explanations:

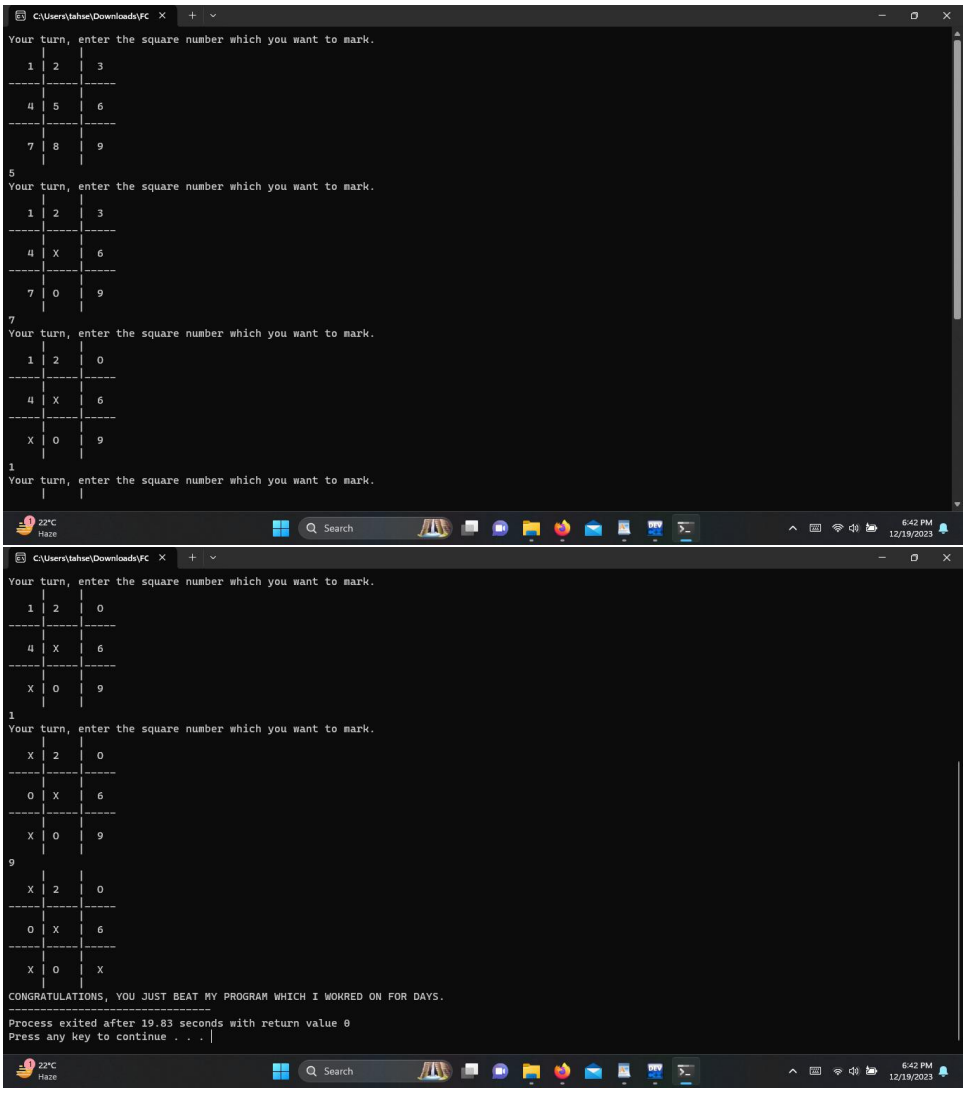
Case 1 (Draw):



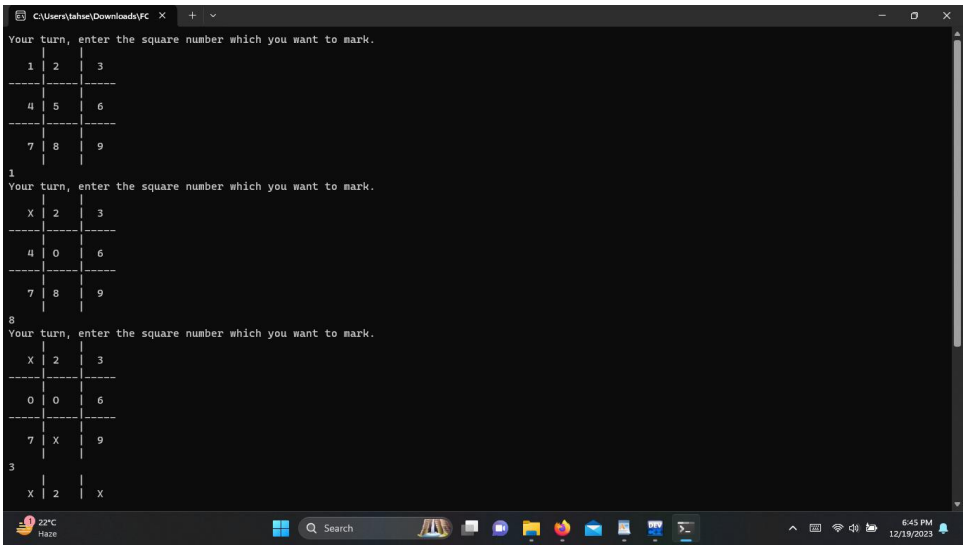

The above screenshots show the case when both the player and AI both try their best to block each other leading to an outcome where no one wins.

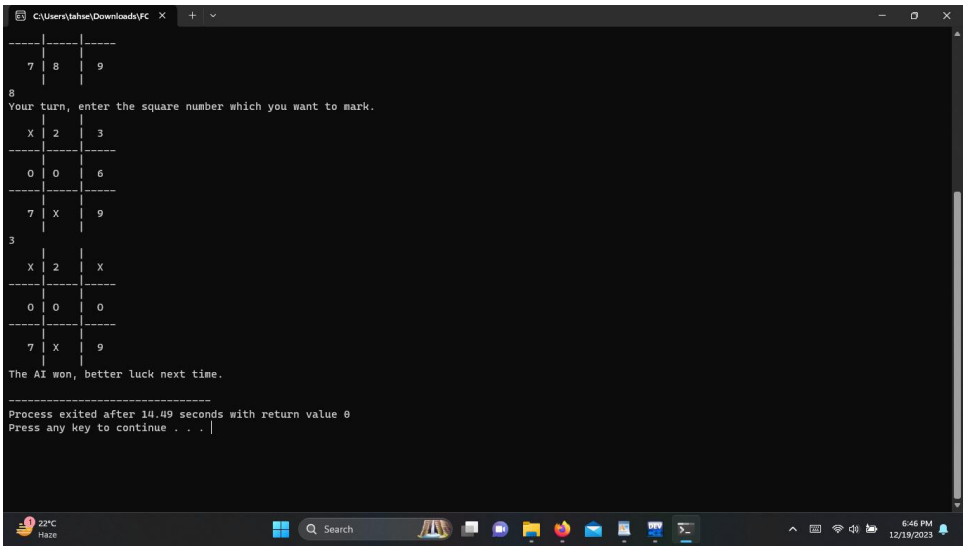
Case 2 (Win):




Since or “AI” doesn't have access to all the possible variations of the board, hence the AI can't always make the best moves. In the above example the user trapped the Ai leaving it with no choice but to lose.

Case 3 (Lose):




Our AI will always try to connect 2 “O” to win, as seen in the above screenshots.

## **Future Improvements:**

Future iterations of this code could incorporate additional features, such as an improved AI with more advanced strategies, a graphical user interface for enhanced user experience, or the ability to play against other human opponents over a network. The game serves as a foundational example for beginners learning C++ game development, providing a practical demonstration of concepts like user input handling, decision-making algorithms, and modular code structure.

## Conclusion:

This code provides a practical example of a console-based Tic-Tac-Toe game in C++, demonstrating fundamental programming concepts such as user input handling, decision-making algorithms, and modular code structure. It serves as a valuable resource for learning and can be a foundation for developing more complex board games or educational tools.