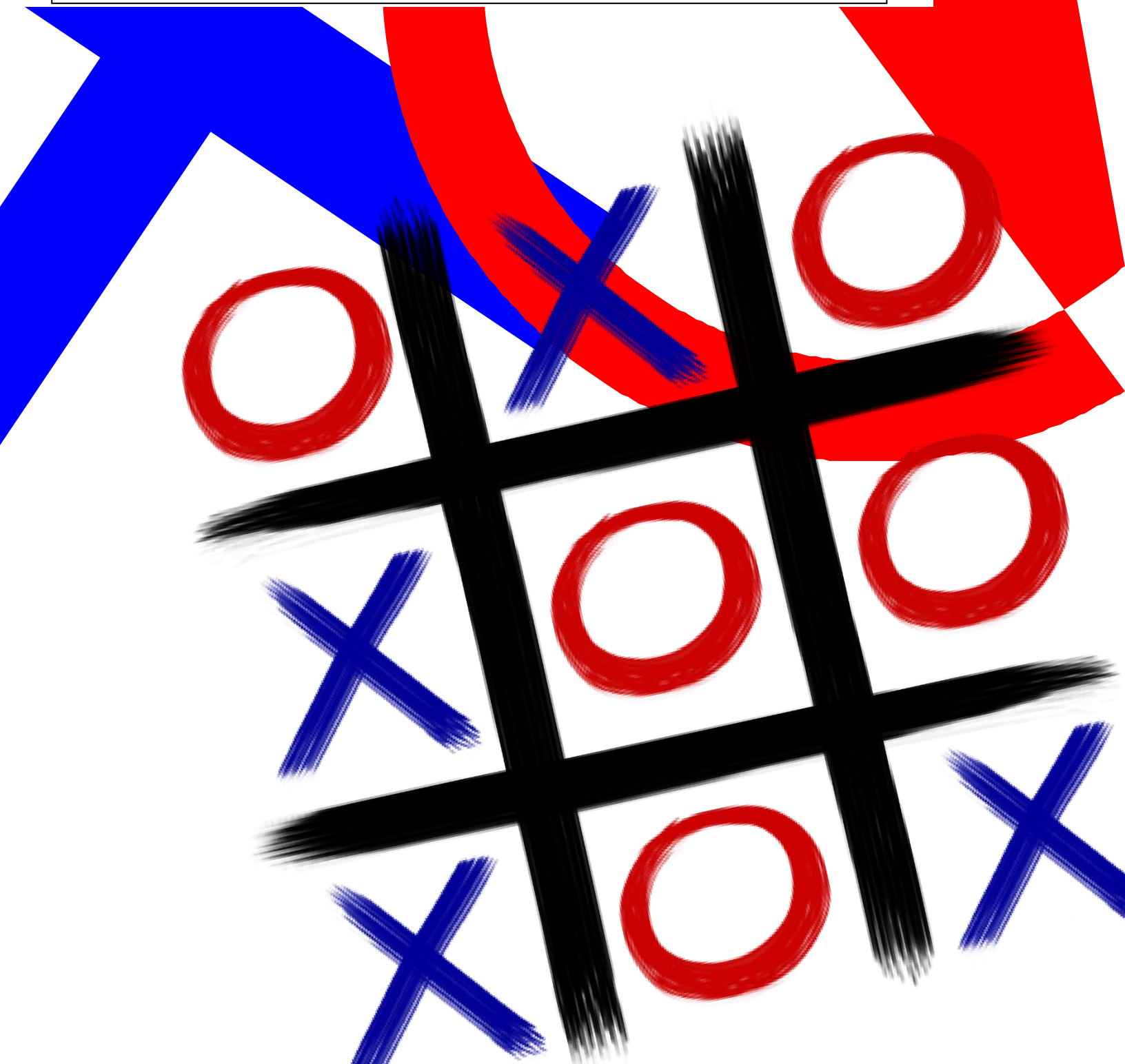
TIC TAC TOE

SAAD ATIF 156, Qazi MUHAMMAD USMAN 149



Introduction

For our project we decided to choose the first option, Tic Tac Toe. As provided in the project instructions, the idea was implemented using a Minimax Algorithm. It is a backtracking/recursive algorithm, used in decision making and game theory. It is commonly used in game playing in AI as it provides the most optimal move for the player assuming that the opponent is also playing optimally. Let's see how the game functions further.

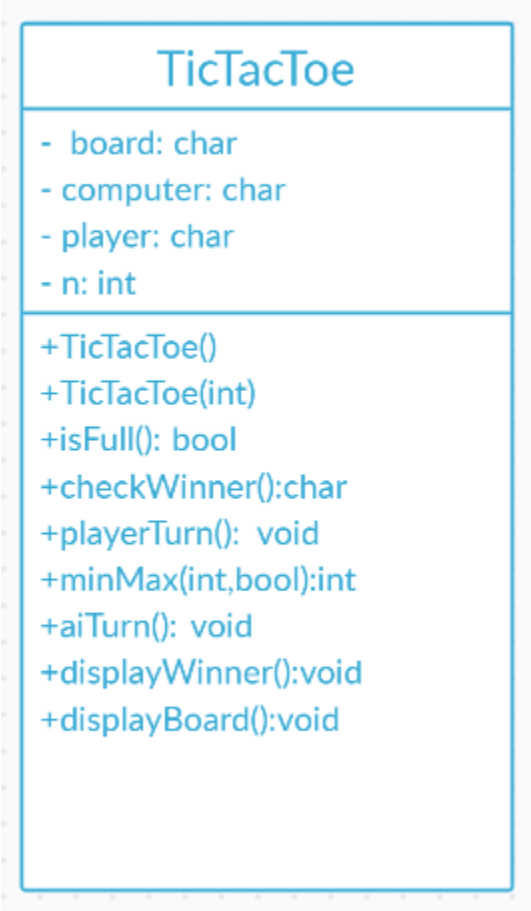
Problem Statement

To create a Tic Tac Toe game using the Minimax Algorithm.

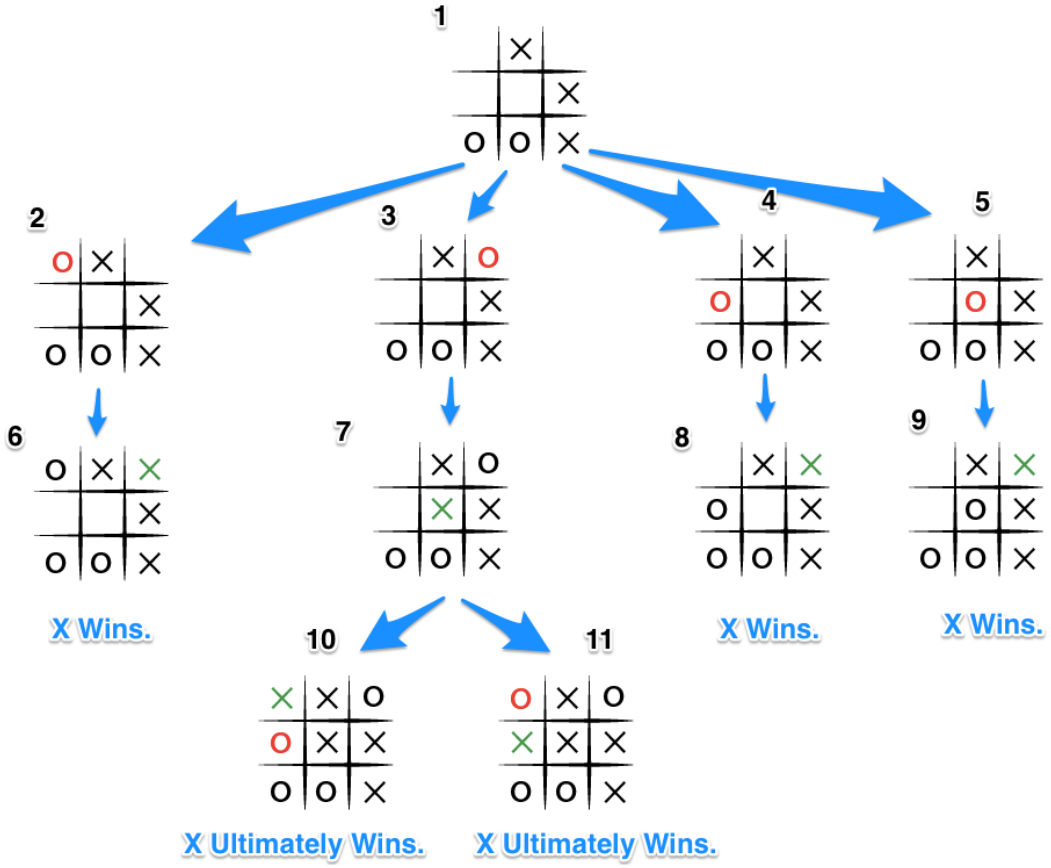
Objective

- This project's purpose is to build a game named tic-tac-toe using a game tree.
- To understand the Minimax Algorithm and implement it in the game.

UML



Game Tree



Code:

```
#include <iostream>
#include<fstream>
using namespace std;
class node
{
public:
	personal data;
	node* next;
	node* root = NULL;
	void insert_new_node()
	{
		data.insert();
		node* temp = new node;
		temp->data = data;
		temp->next = NULL;
		if (root == NULL)
		{
			root = temp;
		}
		else
		{
			node* p = root;
			while (p->next != NULL)
			{
				p = p->next;
			}
```

```
                        p->next = temp;
                }

        }

};

class personal
{
public:
        string name;
        int age;
        string status;
        void insert()
        {
                ofstream file;
                file.open("Example.txt");
                cout << "Enter the name of the player.....";
                cin >> name;
                file << "Name:";
                file << name;

                cout << "\nEnter the age: ";
                cin >> age;
                file << endl;
                file << "Age ..."<<age;
                file << status;
                file.close();
        }
};
class TicTacToe {
public:
        node head;
private:
        char** board; /* = { {'x', 'o', 'x'},
        {'x', 'o', 'o'},
        {'o', 'o ', 'x'}
        };*/
        int n; //store the size of grid
        char player = 'x'; //x for player
        char computer = 'o'; //o for computer
public:
        TicTacToe() { //function to initilze the board to empty
                this->n = 3;
                board = new char* [n];
                for (int i = 0; i < n; i++) {
                        board[i] = new char[n];
                }
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                board[i][j] = 0;
                        }
                }
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                board[i][j] = ' ';
                        }
                }
        }
        TicTacToe(int n) { //function to set the size of grid and intilize it
to empty
```

```
		this->n = n;
		board = new char* [n];
		for (int i = 0; i < n; i++) {
			board[i] = new char[n];
		}
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < n; j++) {
				board[i][j] = 0;
			}
		}
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < n; j++) {
				board[i][j] = ' ';
			}
		}
	}
	bool isFull() { //function to check whether board is full
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < n; j++) {
				if (board[i][j] == ' ')
					return false;
			}
		}
		return true;
	}
	char checkWinner()
	{ //return the winner if there is one
		int player_win_counter = 0;
		int computer_win_counter = 0;
		for (int i = 0; i < n; i++) { //check horizontally
			for (int j = 0; j < n; j++) {
				if (board[i][j] == player && player_win_counter <
n)
					player_win_counter++;
				else if (board[i][j] == computer &&
computer_win_counter <
					n)
					computer_win_counter++;
				if (player_win_counter == n) {
					//cout << "Player Won";
					return player;
				}
				if (computer_win_counter == n) {
					//cout << "Computer Won";
					return computer;
				}
			}
			player_win_counter = 0;
			computer_win_counter = 0;
		}
		player_win_counter = 0; //reset for next case
		computer_win_counter = 0; //reset for next case
		for (int i = 0; i < n; i++) { //check vertically
			for (int j = 0; j < n; j++) {
				if (board[j][i] == player && player_win_counter <
n)
					player_win_counter++;
				else if (board[j][i] == computer &&
computer_win_counter <
					n)
					computer_win_counter++;
```

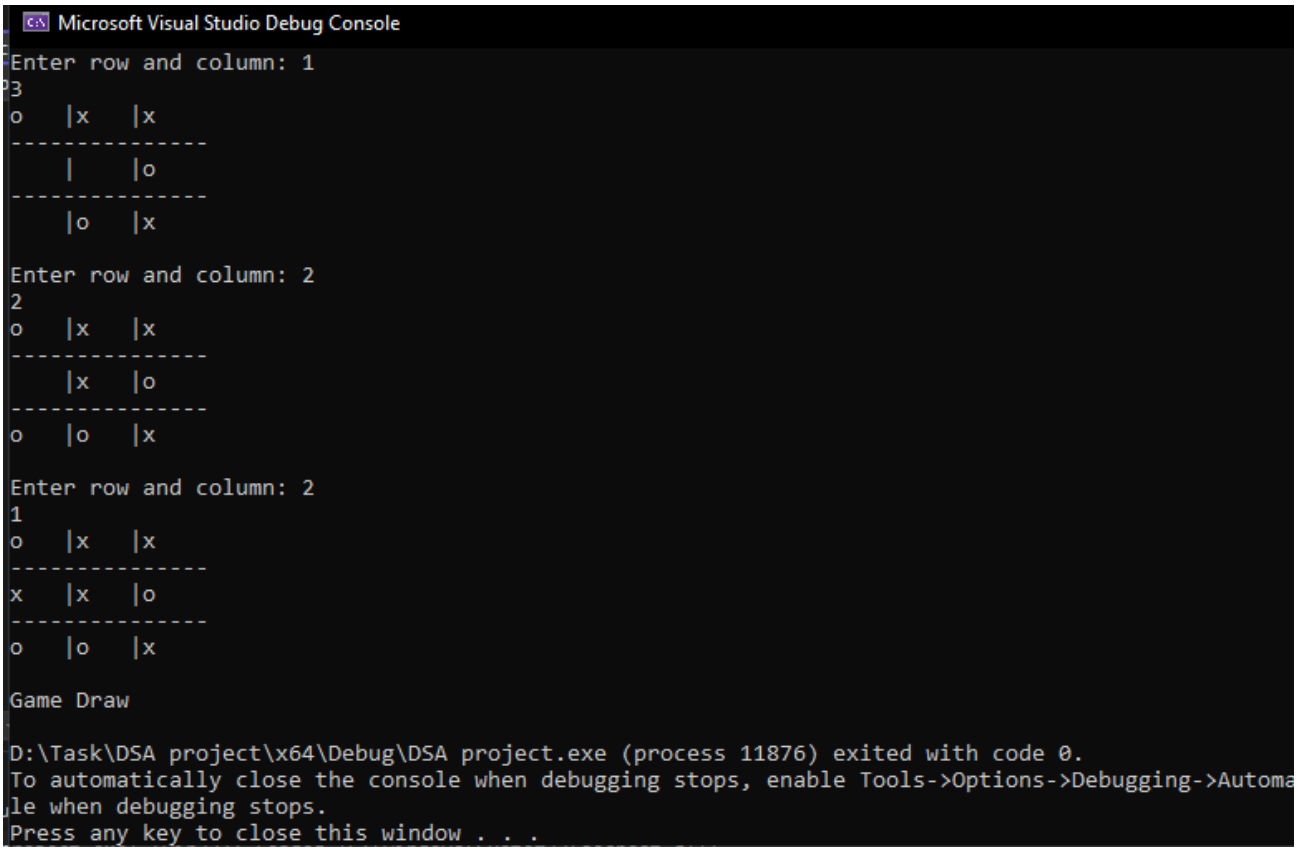
```
                        if (player_win_counter == n) {
                              //cout << "Player Won";
                              return player;
                        }
                        if (computer_win_counter == n) {
                              //cout << "Computer Won";
                              return computer;
                        }
                  }
                  player_win_counter = 0;
                  computer_win_counter = 0;
            }
            player_win_counter = 0; //reset for next case
            computer_win_counter = 0; //reset for next case
            for (int i = 0; i < n; i++) { //check main diagonal
                  for (int j = 0; j < n; j++) {
                        //if condition to traverse only diagonal
                        if (i == j) {
                              if (board[j][i] == player &&
player_win_counter <
                                    n)
                                    player_win_counter++;
                              else if (board[j][i] == computer &&
                                    computer_win_counter < n)
                                    computer_win_counter++;
                              if (player_win_counter == n) {
                                    //cout << "Player Won";
                                    return player;
                              }
                              if (computer_win_counter == n) {
                                    //cout << "Computer Won";
                                    return computer;
                              }
                        }
                  }
            }
            player_win_counter = 0; //reset for next case
            computer_win_counter = 0; //reset for next case
            for (int i = 0; i < n; i++) { //check secondary diagonal
                  for (int j = 0; j < n; j++) {
                        //if condition to traverse only secondary diagonal
                        if (((i + j) == (n - 1))) {
                              if (board[j][i] == player &&
player_win_counter <
                                    n)
                                    player_win_counter++;
                              else if (board[j][i] == computer &&
                                    computer_win_counter < n)
                                    computer_win_counter++;
                              if (player_win_counter == n) {
                                    //cout << "Player Won";
                                    return player;
                              }
                              if (computer_win_counter == n) {
                                    //cout << "Computer Won";
                                    return computer;
                              }
                        }
                  }
            }
            if (isFull()) {
```

```
                        //cout << "Game Drew";
                        return 't';
                }
                return NULL;
        }
        void playerTurn() { //function for players turn
                displayboard();
                int row = -1, col = -1;
                bool check = true; //used in while condition to iterate the loop
until valid input is entered
                if (!isFull()) {
                        while (check) {
                                cout << "Enter row and column: ";
                                cin >> row >> col;
                                if (row > n || col > n) { //if out of range of
board
                                        cout << "Please enter a valid input.\n";
                                        continue;
                                }
                                else if (board[row - 1][col - 1] != ' ') { //if
spot already filed
                                        cout << "Spot already filled. Try again.\n";
                                        continue;
                                }
                                board[row - 1][col - 1] = player;
                                check = false; //terminate loop
                        }
                }
                //checkGameState(); //see if player won
        }
        int minMax(int depth, bool max_player) {
                int result = checkWinner();
                int bestScore;
                if (result != NULL) { //check whether there is a winner
                        int score;
                        if (result == player) //if player wins return +10
                                score = 10;
                        else if (result == computer) { //if computer wins return
-10
                                score = -10;
                        }
                        else if (result == 't') { //return 0 for a tie
                                score = 0;
                        }
                        return score - depth; //subtract the depth to find the
most optimal move
                }
                if (max_player) { //for maximizing player
                        bestScore = INT_MIN;
                        for (int i = 0; i < n; i++) {
                                for (int j = 0; j < n; j++) {
                                        if (board[i][j] == ' ') { //check if spot is
empty
                                                board[i][j] = player; //place x
                                                int score = minMax(depth + 1, false);
//call the function for minimizing player
                                                board[i][j] = ' '; //revert move
                                                bestScore = max(score, bestScore);
                                                //store greater value in best score
                                        }
                                }
```

```
                        }
                        return bestScore; //return the best score for maximizing
player
                }
                else { //for minimizing player
                        bestScore = INT_MAX;
                        for (int i = 0; i < n; i++) {
                                for (int j = 0; j < n; j++) {
                                        if (board[i][j] == ' ') { //check if spot is
empty
                                                board[i][j] = computer; //place o
                                                int score = minMax(depth + 1, true);
//call the function for maximizing player
                                                board[i][j] = ' '; //revert move
                                                bestScore = min(score, bestScore);
                                                //score lesser score in best score
                                        }
                                }
                        }
                        return bestScore; //return the best score for minimizing
player
                }
        }
        void aiTurn() { //function for computers turn
                int bestScore = INT_MIN;
                int row = -1, col = -1;
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                if (board[i][j] == ' ') { //check if spot is empty
                                        board[i][j] = player; //place the player
symbol
                                        int score = minMax(0, false); //call minMax
on that spot
                                        board[i][j] = ' '; //revert move
                                        if (score > bestScore)
                                        { //store the best move and best score
                                                row = i;
                                                col = j;
                                                bestScore = score;
                                        }
                                }
                        }
                }
                board[row][col] = computer; //make the best move for the ai
        }
        void displayWinner() {
                ofstream file;
                char winner = checkWinner();
                if (winner == player)
                {

                        file.open("Example.txt");
                        cout << "Player Won" << endl;
                        file << "Player Won";
                }
                else if (winner == computer)
                {
                        cout << "Computer won" << endl;
                        file << "computer Won";
                }
                else
```

```
                {
                        cout << "Game Draw" << endl;
                        file << "Game Tie ";

                }
                file.close();
        }
        void displayboard() {
                int temp = 0; /*Variable to store the count of required dashes
after
                each
                line. Required because size of board is is
                customizable*/
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                if (j < n - 1) { //doesn't need to print | for last
case
                                        cout << board[i][j] << "   |";
                                        if (i == 0) { //only count for first
iteration
                                                temp += 2; //since two element are
printed (value and | )
                                        }
                                }
                                else { //for last case
                                        cout << board[i][j] << endl;
                                        if (i == 0) {
                                                temp++; //only one is needed
                                        }
                                }
                        }
                        for (int dash = 0; dash < temp && i < n - 1; dash++)
                                cout << "---"; //print dashes
                        cout << endl; //next line
                }
        }
};
int main()
{
        TicTacToe t;
        t.head.insert_new_node();
        system("cls");
        while (!t.checkWinner())
        {
                t.playerTurn();
                if (!t.checkWinner())
                        t.aiTurn();
        }
        t.displayboard();
        t.displayWinner();
        return 0;

}
```

Output:



Conclusion

The game Tic-tac-toe is implemented using the Minimax Algorithm. This is done by creating a class Tic-tac-toe. We made this algorithm a member function in the class and the function was called to use. Besides this we have multiple functions such as to display the game board, for the players to do their turns, to check who won and more for a fully functioning tic tac toe game.