Project Report for Tic Tac Toe Game

What is Tic Tac Toe?

Tic Tac Toe is basically a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a diagonal, horizontal, or vertical row is the winner.

Rules of the Game

Here are some rules of this game given below

- The game is to be played between two people (in this program between Player 1 and Player 2).
- One of the player chooses 'O' and the other 'X' to mark their respective cells.
- The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character ('O' or 'X').
- If no one wins, then the game is said to be draw.

Winning Strategy

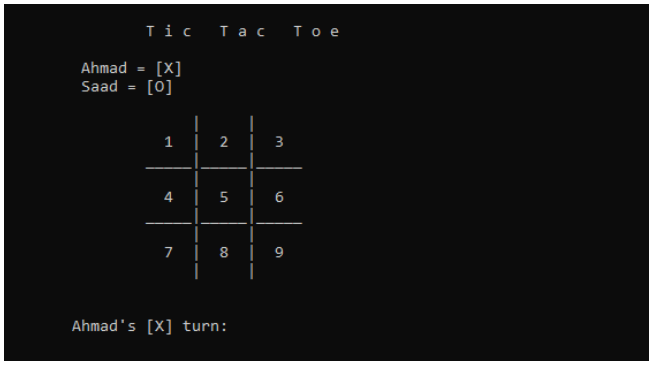
The player who succeeds in placing three of their marks in a diagonal, horizontal, or vertical row is the winner. If both the players play optimally then it is destined that you will never lose ("although the match can still be drawn"). It doesn't matter whether you play first or second. In another ways – "Two expert players will always draw". Isn't this interesting?

Implementation

In our program the moves taken by the player 1 and the player 2 are chosen. I used player_turn() function for this. In this game I used many other functions that'd be explained below.

Input and outputs

In this project only integers will be taken as input which are between 1 and 9. As soon a user give input that box will be changed to X or O depend on which user is playing taking their turn. The output of this program would be shown like in this figure: ↓↓



This output will show many times as our program clears the screen after taking the input from user and implementing on this game.

Which functions I used in this program

I used different functions in this program but all they were defined by myself(user-defined). I didn't use any pre-defined functions from the library. I used functions in this program for ease of myself to so that I can use that piece of code wherever I want to call in this program. All the functions are globally defined so that we can easily use that function in any other function body. Here's the list of my functions that I used in this program:

- display_board()
  this function has a piece of code that displays the board and the box values i.e 1-9
- player_turn()
  the code of this function works to take player's move until the game end.
- game_over()
  this function of this game decides who will win loose the game or draw the game.

How different functions are inter-linked

No functions are interlinked in this program

Algortihm_Tic_Tac_Toe

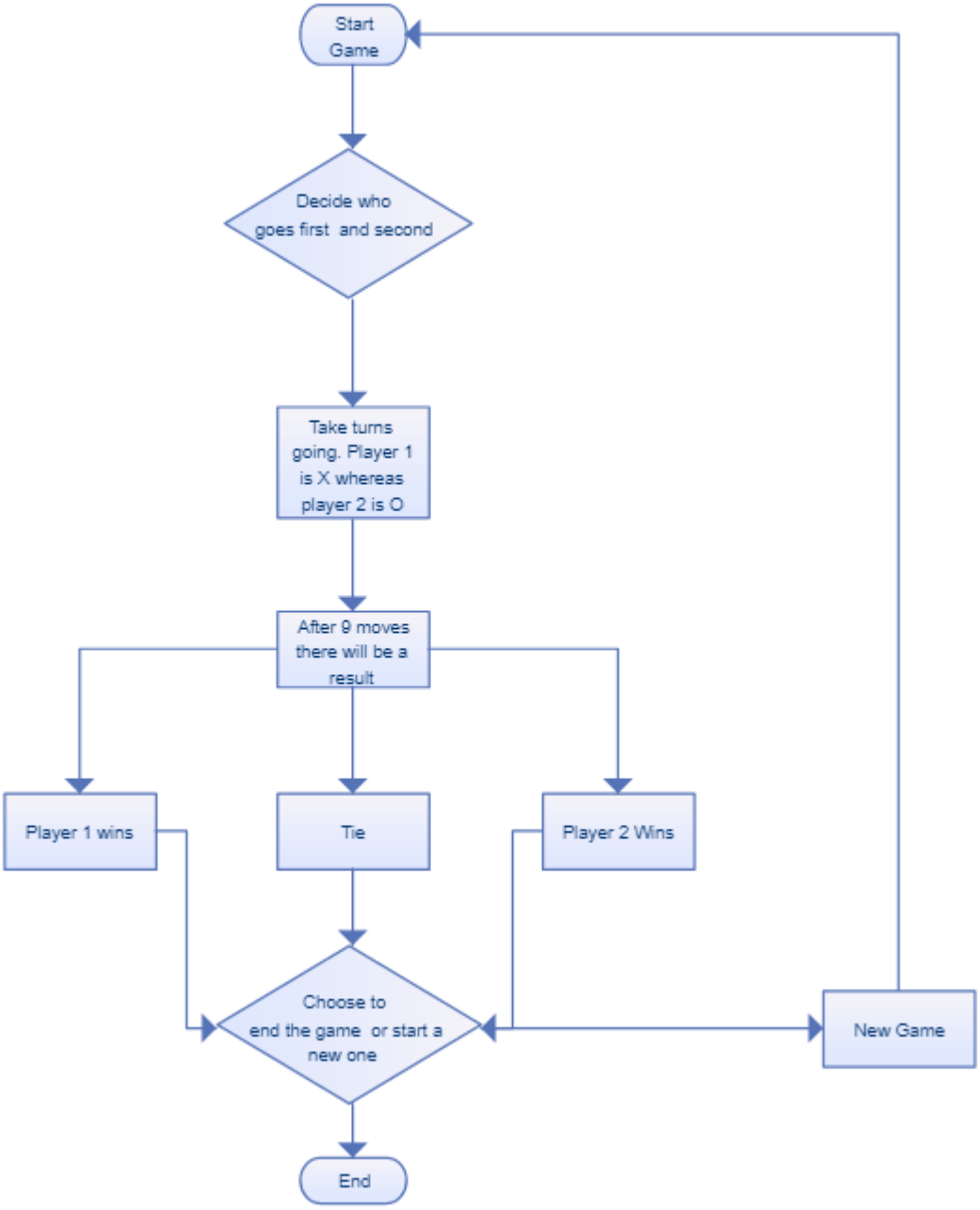
Start

1. Declare Structure
2. Declare variables in the structure
3. Initialize structure with a
4. Define Function display_board()
   4.1. Print game name
   4.2. Display player's name
   4.3. Display checkboard of 3x3
   4.4. Set boxes with array index
5. Define player_turn()
   5.1. Check if turn == X
   5.2. Display player's turn
   5.3. Check if turn == O
   5.4. Display opponent's turn
   5.5. Input: choice
   5.6. Set switch for choice
   5.7. If case 1: row=0 column=0
   5.8. If case 1: row=0 column=1
   5.9. If case 1: row=0 column=2
   5.10. If case 1: row=1 column=0
   5.11. If case 1: row=1 column=1
   5.12. If case 1: row=1 column=2
   5.13. If case 1: row=2 column=0

5.14. If case 1: row=2 column=1
5.15. If case 1: row=2 column=2
5.16. Default: display invalid input
5.17. Set if statement to check player's move
5.18. Else display box already filled
5.19. Call display_board()

6. Define function game_over()
   6.1. Set for loop for 0-3
   6.2. Check if horizontal and vertical boxex are equal
   6.3. Check if diagonally boxes are equal
7. Start main function
   7.1. Prompt player 1 name
   7.2. Prompt player 2 name
   7.3. Set while loop
      7.3.1. Call display_board()
      7.3.2. Call player_turn()
      7.3.3. Call game_over()
   7.4. Set if to check player winning
   7.5. Else draw the game
8. End

Flowchart



Logic

A player can play a perfect game (that means win or draw) if he/she follows several simple rules in placing the next mark on the grid. The rules are described above, where you can also find the optimal strategy for the player that makes the first move.

The optimal strategies for both the first and second player are available on an xkcd drawing. Though it has several errors (it misses making winning moves in several situations and at least in one case is missing an X mark), I will be using this version for the playing strategy (with fixes the

errors that I was able to find). Bear in mind, this means the computer will always play a perfect game. If you implement such a game, you probably want to also let the users win, in which case you need a different approach. But for the purpose of this article, this should suffice.

Source code

```
#include <iostream>
using namespace std;
struct layout
{
int choice;
char board[3][3] =  {{'1','2','3'},{'4','5','6'},{'7','8','9'}};
int row, column;
char turn = 'X';
bool draw =false;
string player1, player2;
};

layout a;

void display_board()
{
	system("cls");
	cout<< "\n\t\tT i c   T a c   T o e\n";
	cout<<"\t==================================="<<endl;
	cout<< "\n\t "<<a.player1<<" = [X] \n \t "<<a.player2<<" = [O]\n\n";

	cout<<"\t\t     |     |     "<<endl;
	cout<<"\t\t  "<<a.board[0][0]<<"  |  "<<a.board[0][1]<<"  |  "<<a.board[0][2]<<"  "<<endl;

	cout<<"\t\t_____|_____|_____"<<endl;

	cout<<"\t\t     |     |     "<<endl;

	cout<<"\t\t  "<<a.board[1][0]<<"  |  "<<a.board[1][1]<<"  |  "<<a.board[1][2]<<"  "<<endl;

	cout<<"\t\t_____|_____|_____"<<endl;

	cout<<"\t\t     |     |     "<<endl;

	cout<<"\t\t  "<<a.board[2][0]<<"  |  "<<a.board[2][1]<<"  |  "<<a.board[2][2]<<"  "<<endl;
```

```
	cout<<"\t\t    |    |    "<<endl;
}

void player_turn()
{
	if (a.turn == 'X')
	cout<<"\n\n\t"<<a.player1<<"'s [X] turn: ";
	if (a.turn == 'O')
	cout<<"\n\n\t"<<a.player2<<"'s [O] turn: ";
	cin>> a.choice;
	switch(a.choice)
	{
		case 1: a.row = 0; a.column = 0; break;
		case 2: a.row = 0; a.column = 1; break;
		case 3: a.row = 0; a.column = 2; break;
		case 4: a.row = 1; a.column = 0; break;
		case 5: a.row = 1; a.column = 1; break;
		case 6: a.row = 1; a.column = 2; break;
		case 7: a.row = 2; a.column = 0; break;
		case 8: a.row = 2; a.column = 1; break;
		case 9: a.row = 2; a.column = 2; break;

		default:
			cout<<"Invalid Input !";
			break;
	}
	if (a.turn == 'X' && a.board[a.row][a.column] != 'X' && a.board[a.row][a.column] != 'O')
	{
		a.board[a.row][a.column] = 'X';
		a.turn = 'O';
	}
	else if (a.turn == 'O' && a.board[a.row][a.column] != 'X' && a.board[a.row][a.column] != 'O')
	{
		a.board[a.row][a.column] = 'O';
		a.turn = 'X';
	}
	else
	{
		cout<< "\n\tBox already filled \n\tChoose another box";
		player_turn();
	}
	display_board();
}
```

```
bool game_over()
{
	for (int i=0; i<3; i++)
	if (a.board[i][0] == a.board[i][1] && a.board[i][0] == a.board[i][2] || a.board[0][i] ==
a.board[1][i] && a.board[0][i] == a.board[2][i])
	return false;

	if(a.board[0][0] == a.board[1][1] && a.board[0][0] == a.board[2][2] || a.board[0][2] ==
a.board[1][1] && a.board[0][0] == a.board[2][0])
	return false;

	for(int i=0; i<3; i++)
	for(int j=0; j<3; j++)
	if (a.board[i][j] != 'X' && a.board [i][j] != 'O')
	return true;

	a.draw = true;
	return false;
}



int main()
{
	cout<<"What's your name?\n";
getline(cin,a.player1);
cout<<"What's your opponent name?\n";
getline(cin,a.player2);
	while(game_over())
	{
	display_board();
	player_turn();
	game_over();

}

if(a.turn== 'X' && a.draw == false)
cout<<a.player2<<" [O] wins!!\n";
else if(a.turn== 'O' && a.draw == false)
cout<<a.player1<<" [X] wins!!\n";
else
{
	cout<< "GAME DRAW !!";
```

```
}

}
```

Sample Output