

A PROJECT REPORT ON TIC TAC TOE GAME

Submitted in partial fulfillment of the requirement for the IV semester of BACHELOR OF TECHNOLOGY IN DESIGN AND ANALYSIS OF ALGORITHM

Submitted By: Shivam Singh (RA2011003010087)

Under the supervision of Dr. M Sindhuja (Asst. Professor)

DEPARTMENT OF COMPUTING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Certificate

This is to certify that the project entitled “**Tic Tac Toe** **Game**” carried out by “**Shivam Singh**” under my

supervision at Department of Computing and Technology, SRM Institute of Technology, Kattankulathur, Chennai.

The work is original, as it has not been submitted earlier either in part or full for any purpose before.

Dr. M Sindhuja (Asst. Professor)

# DECLARATION

I, hereby declare that the work presented in this dissertation entitled “TIC TAC TOE GAME” has been done by me, and this dissertation embodies my own work.

**By -Shivam Singh**

**Approved By:** Dr. M Sindhuja

ACKNOWLEDGEMENTS

I thanks Dr. M SINDHUJA (Asst. Professor) who have been the  great inspiration and who have provided sufficient background knowledge and understanding of this subject.

Our humble prostration goes to her, for providing all the necessary resources and environment ,which have aided me to complete this project successfully.

# **PREFACE**

This report is an introduction to the Tic Tac Toe game in

C programming. Anybody, who doesn’t know even the

 basics of Tic Tac Toe in C ,will be certainly able to understand and gain the great knowledge from this report. The core theme of the report focuses on the development of Tic Tac Toe game in C language.

The report also contains the strategy of making Tic Tac Toe game which serve a good idea to make a Tic Tac Toe game program in C language to the programmer.

The most of the idea of making this game and report is taken from “ Let Us C-  by Yashwant Kanetkar’’ , Schaum’s Outline-C(TMH publications), and Internet

(Wikipedia ,Google ,etc.) .

# **CONTENTS**

1. ACKNOWLEDGEMENTS…………………… 1

1. PREFACE………………..….……………………2
2. ABSTRACT ………………………………………….3
3. WHAT IS TIC TAC TOE ?......................……………4

4.1. STRATEGY……………………………….…. 5

5. PROGRAM FOR TIC TAC TOE GAME ………12

6. CONCLUSION…………………………………..20

7. REFERENCE…………………………………….21

**Abstract**

TIC TAC TOE is a commonly traditional game in Yemen, which consist of two players. One of the player chooses a cell and put X mark in the chosen cell while the second player chose another cell and put 0 mark in this second cell.

The purpose of the paper is to transport this commonly traditional game to get along with modern electronic world. The modern electronic game is important for the country children so that they play the game safely without faces with any diseases. It is also aim to offer children an opportunity to deal with cheap modern techniques. The game designed in this paper based on the microcontroller.

Introduction---

Tic-tac-toe is played on a three-by-three grid by two players, who alternately place the marks X and O in one of the nine spaces in the grid.

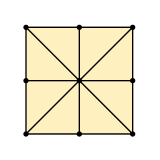
In the following example, the first player (*X*) wins the game in seven steps:

[Game of Tic-tac-toe, won by X](https://en.wikipedia.org/wiki/File:Tic-tac-toe-game-1.svg)

There is no universally-agreed rule as to who plays first, but in this article the convention that X plays first is used.

Players soon discover that the best play from both parties leads to a [draw](https://en.wikipedia.org/wiki/Draw_(tie)). Hence, tic-tac-toe is often played by young children who may not have discovered the optimal strategy.

Because of the simplicity of tic-tac-toe, it is often used as a [pedagogical](https://en.wikipedia.org/wiki/Pedagogical) tool for teaching the concepts of good [sportsmanship](https://en.wikipedia.org/wiki/Sportsmanship) and the branch of [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) that deals with the searching of [game trees](https://en.wikipedia.org/wiki/Game_tree). It is straightforward to write a [computer program](https://en.wikipedia.org/wiki/Computer_program) to play tic-tac-toe perfectly or to enumerate the 765 essentially different positions (the [state space complexity](https://en.wikipedia.org/wiki/State_space_complexity)) or the 26,830 possible games [up to](https://en.wikipedia.org/wiki/Up_to) rotations and reflections (the [game tree complexity](https://en.wikipedia.org/wiki/Game_tree_complexity)) on this space. If played optimally by both players, the game always ends in a draw, making tic-tac-toe a [futile game](https://en.wikipedia.org/wiki/Futile_game).

[](https://en.wikipedia.org/wiki/File:Three_Men%27s_Morris_variant_board.svg)

[Incidence structure](https://en.wikipedia.org/wiki/Incidence_structure) for tic-tac-toe

The game can be generalized to an *[m](https://en.wikipedia.org/wiki/M,n,k-game" \o "M,n,k-game)*[,](https://en.wikipedia.org/wiki/M,n,k-game" \o "M,n,k-game)*[n](https://en.wikipedia.org/wiki/M,n,k-game" \o "M,n,k-game)*[,](https://en.wikipedia.org/wiki/M,n,k-game" \o "M,n,k-game)*[k](https://en.wikipedia.org/wiki/M,n,k-game" \o "M,n,k-game)*[-game](https://en.wikipedia.org/wiki/M,n,k-game" \o "M,n,k-game), in which two players alternate placing stones of their own color on an *m*-by-*n* board with the goal of getting *k* of their own color in a row. Tic-tac-toe is the 3,3,3-game. [Harary's generalized tic-tac-toe](https://en.wikipedia.org/wiki/Harary%27s_generalized_tic-tac-toe" \o "Harary's generalized tic-tac-toe) is an even broader generalization of tic-tac-toe. It can also be generalized as an *[n](https://en.wikipedia.org/wiki/Nd_game" \o "Nd game)[d](https://en.wikipedia.org/wiki/Nd_game" \o "Nd game)*[game](https://en.wikipedia.org/wiki/Nd_game" \o "Nd game), specifically one in which *n* equals 3 and *d* equals 2. It can be generalised even further by playing on an arbitrary [incidence structure](https://en.wikipedia.org/wiki/Incidence_structure), where rows are [lines](https://en.wikipedia.org/wiki/Line_(geometry)) and cells are [points](https://en.wikipedia.org/wiki/Point_(geometry)). Tic-tac-toe's incidence structure consists of nine points, three horizontal lines, three vertical lines, and two diagonal lines, with each line consisting of at least three points.

Tic-tac-toe is not a very challenging game for human

 beings. If you’re an enthusiast, you’ve probably moved

from the basic game to some variant like three

dimensional tic-tac-toe on a larger grid. If you sit down right now to play ordinary three-by-three tic-tac-toe with a friend, what will probably happen is that every game will come out a tie. Both you and your friend can

 probably play perfectly, never making a mistake that

would allow your opponent to win. But can you describe

how you know where to move each turn? Most of the time,you probably aren’t even aware of alternative

 possibilities; you just look at the board and instantly know

where you want to move. That kind of instant knowledge

is great for human beings, because it makes you a fast

 player. But it isn’t much help in writing a computer

 program. For that, you have to know very explicitly what

your strategy is.

STRATEGY—

The highest-priority and the lowest-priority rules seemed obvious to me right away.The highest-priority are these: 1. If I can win on this move,do it.

1. If the other player can win on the next move,block that winning square.

Here are the lowest-priority rules,used only if there is nothing suggested more strongly by the board position:

n-2. Take the center square if it’s free.

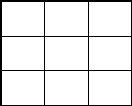
n-1. Take a corner square if one is free.

n. Take whatever is available.

The highest priority rules are the ones dealing with the most urgent situations:either I or my opponent can win on the next move.The lowest priority ones deal with the least urgent situations,in which there is nothing special about the moves already made to guide me.

What was harder was to find the rules in between.I knew that the goal of my own tic-tac-toe strategy was to set up a fork, a board position in which I have two winning moves,

so my opponent can only block one of them. Here is an example:



x o x

x o

X can win by playing in square 3 or square 4.It’s O’s turn,

 but poor O can only block one of those squares at a time.

Whichever O picks, X will then win by picking the other

one. Given this concept of forking, I decided to use it as the next highest priority rule:

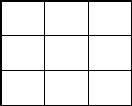
1. If I can make a move that will set up a fork for myself, do it.That was the end of the easy part. My first attempt at writing the program used only these six rules.

Unfortunately, it lost in many different situations. I needed to add something, but I had trouble finding a good rule to add. My first idea was that rule 4 should be the defensive equivalent of rule 3, just as rule 2 is the defensive equivalent of rule 1:

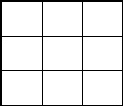
4a. If, on the next move, my opponent can set up a fork,block that possibility by moving into the square that is common to his two winning combinations.

In other words, apply the same search technique to the opponent’s position that I applied to my own.This strategy works well in many cases, but not all. For

example, here is a sequence of moves under this strategy,with the human player moving first:

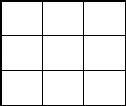


x



o

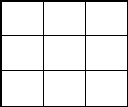
x



o

x

x

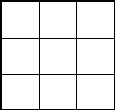


o

o

x

x



o x o x

x

In the fourth grid, the computer (playing O) has discovered that X can set up a fork by moving in square 6,

 between the winning combinations 456and 369. The

computer moves to block this fork.

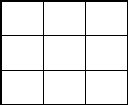
Unfortunately,X can also set up a fork by moving in squares 3, 7, or 8. The computer’s move in square 6 has

 blocked one combination of the square-3 fork,but X can

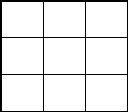
still set up the other two. In the fifth grid, X has moved in

square 8. This sets up the winning combinations 258 and 789.The computer can only block one of these, and X will win on the next move.

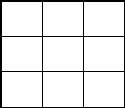
Since X has so many forks available, does this mean that the game was already hopeless before O moved in square 6? No. Here is something O could have done:



x



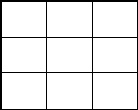
o



o

x

x

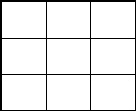


o

x

o

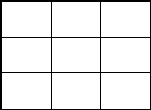
x



o

x x

o x



o

x x o

o x

In this sequence, the computer’s second move is in square 7. This move also blocks afork, but it wasn’t chosen for that reason. Instead, it was chosen*to force X’s next move* .

In the fifth grid, X has had to move in square 4, to prevent

an immediate win by O. The advantage of this situation for O is that square 4 was not one of the ones with which X could set up a fork. O’s next move, in the sixth grid, is also forced. But by then the board is too crowded for either player to force a win; the game ends in a tie, as usual.

This analysis suggests a different choice for an intermediate-level strategy rule, taking the offensive: 4b.If I can make a move that will set up a winning

combination for myself, do it. Compared to my earlier try,this rule has the benefit of simplicity. It’s much easier

for the program to look for a single winning combination than for a fork, which is two such combinations with a common square.

Unfortunately, this simple rule isn’t quite good enough. In the example just above,the computer found the winning combination in which it already had square 1,and the other two were free. But why should it choose to move in square 7 rather than square 4? If the program did choose square 4, then X’s move would still be forced, into square 7.

We would then have forced X into creating a fork, which

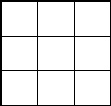
would defeat the program on the next move.

It seems that there is no choice but to combine the ideas from rules 4a and 4b:

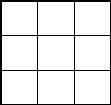
1. If I can make a move that will set up a winning combination for myself, do it. But ensure that this move does not force the opponent into establishing a fork. What this means is that we are looking for a winning combination in which the computer already owns one square and the other two are empty. Having found such a combination, we can move in either of its empty squares. Whichever we choose, the opponent will be forced to choose the other one on the next move. If one of the two empty squares would create a fork for the opponent, then the computer must choose that square and leave the other for the opponent.

What if both of the empty squares in the combination we find would make forks for the opponent? In that case,

we’ve chosen a bad winning combination. It turns out that there is only one situation in which this can happen:

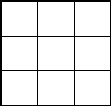


x



x

o

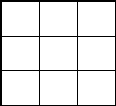


x

o

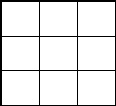
x

Again, the computer is playing O. After the third grid, it is looking for a possible winning combination for itself. There are three possibilities: 258,357 and 456. So far we have not given the computer any reason to prefer one over another. But here is what happens if the program happens to choose 357:



x

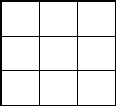
o



x

o

x

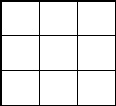


x

o

o

x



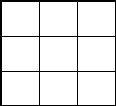
x

o

o

x

x



x

By this choice, the computer has forced its opponent into a fork that will win the game for the opponent. If the computer chooses either of the other two possible winning combinations, the game ends in a tie. (All moves after this choice turn out to be forced.)

This particular game sequence was very troublesome for me because it goes against most of the rules I had chosen earlier. For one thing, the correct choice for the program is any edge square, while the corner squares must be avoided. This is the opposite of the usual priority. Another point is that this situation contradicts rule 4a (prevent forks for the other player) even more sharply than the example we considered earlier. In that example, rule 4a wasn’t enough guidance to ensure a correct choice, but the correct choice was at least with the rule. That is, just blocking a fork isn’t enough, but threatening a win and blocking a fork is better than just threatening a win alone. This is the meaning of rule 4. But in this new situation, the corner square (the move we have to avoid)

 block a fork, while the edge square (the correct move)

 block a fork!

When I discovered this anomalous case, I was ready to

give up on the idea of beautiful, general rules. I almost decided to build into the program a special check for this

 precise board configuration. That would have been pretty

ugly, I think. But a shift in viewpoint makes this case

easier to understand: What the program must do is force

the other player’s move, and force it in a way that helps the computer win. If one possible winning combination doesn’t allow us to meet these conditions, the program should try another combination. My mistake was to think either about forcing alone (rule 4b) or about the

opponent’s forks alone (rule 4a).

As it turns out, the boar d situation we’ve been considering

is the only one in which a possible winning combination

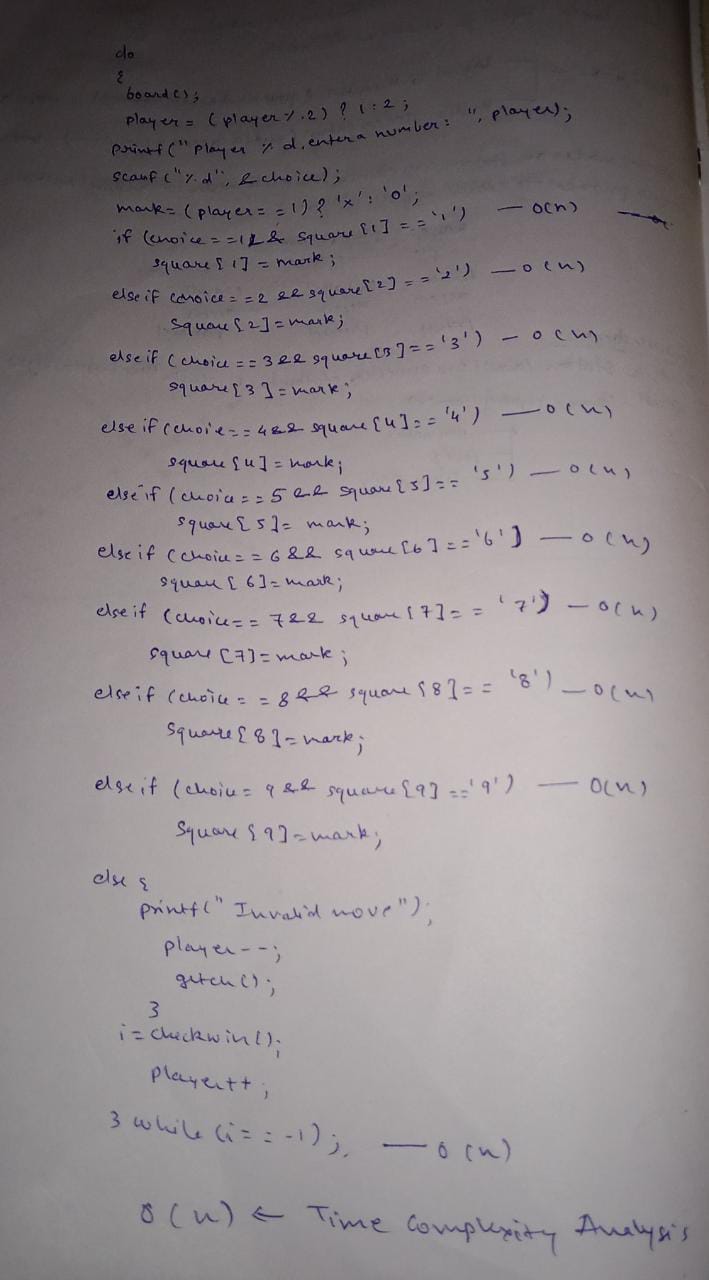
could include two possible forks for the opponent. What’s more, in this board situation, it’s a diagonal combination that gets us in trouble, while a horizontal or vertical combination is always okay. Therefore, I was able to implement rule 4 in a way that only considers one

 possible winning combination by setting up the

 program’s data structures so that diagonal combinations

are the last to be chosen. This trick makes the program’s

design less than obvious from reading the actual program but it does save the program some effort.



Code---

#include <stdio.h>

#include <conio.h>

char square[10] = { 'o', '1', '2', '3', '4', '5', '6', '7', '8', '9' };

int checkwin();

void board();

int main()

{

int player = 1, i, choice;

char mark;

do

{

board();

player = (player % 2) ? 1 : 2;

printf("Player %d, enter a number: ", player);

scanf("%d", &choice);

mark = (player == 1) ? 'X' : 'O';

if (choice == 1 && square[1] == '1')

square[1] = mark;

else if (choice == 2 && square[2] == '2')

square[2] = mark;

else if (choice == 3 && square[3] == '3')

square[3] = mark;

else if (choice == 4 && square[4] == '4')

square[4] = mark;

else if (choice == 5 && square[5] == '5')

square[5] = mark;

else if (choice == 6 && square[6] == '6')

square[6] = mark;

else if (choice == 7 && square[7] == '7')

square[7] = mark;

else if (choice == 8 && square[8] == '8')

square[8] = mark;

else if (choice == 9 && square[9] == '9')

square[9] = mark;

else

{

printf("Invalid move ");

player--;

getch();

}

i = checkwin();

player++;

}while (i == - 1);

board();

if (i == 1)

printf("==>\aPlayer %d win ", --player);

else

printf("==>\aGame draw");

getch();

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FUNCTION TO RETURN GAME STATUS

1 FOR GAME IS OVER WITH RESULT

-1 FOR GAME IS IN PROGRESS

O GAME IS OVER AND NO RESULT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int checkwin()

{

if (square[1] == square[2] && square[2] == square[3])

return 1;

else if (square[4] == square[5] && square[5] == square[6])

return 1;

else if (square[7] == square[8] && square[8] == square[9])

return 1;

else if (square[1] == square[4] && square[4] == square[7])

return 1;

else if (square[2] == square[5] && square[5] == square[8])

return 1;

else if (square[3] == square[6] && square[6] == square[9])

return 1;

else if (square[1] == square[5] && square[5] == square[9])

return 1;

else if (square[3] == square[5] && square[5] == square[7])

return 1;

else if (square[1] != '1' && square[2] != '2' && square[3] != '3' &&

square[4] != '4' && square[5] != '5' && square[6] != '6' && square[7]

!= '7' && square[8] != '8' && square[9] != '9')

return 0;

else

return - 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FUNCTION TO DRAW BOARD OF TIC TAC TOE WITH PLAYERS MARK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void board()

{

system("cls");

printf("\n\n\tTic Tac Toe\n\n");

printf("Player 1 (X) - Player 2 (O)\n\n\n");

printf(" | | \n");

printf(" %c | %c | %c \n", square[1], square[2], square[3]);

printf("\_\_|\_|\_\_\n");

printf(" | | \n");

printf(" %c | %c | %c \n", square[4], square[5], square[6]);

printf("\_\_|\_|\_\_\n");

printf(" | | \n");

printf(" %c | %c | %c \n", square[7], square[8], square[9]);

printf(" | | \n\n");

}

**CONCLUSION**

The Tic Tac Toe is a games for two players, called “X” and “O” , who take turns making the spaces in a 3x3 grid. The player who succeeded in placing three respective marks in a horizontal ,vertical, or diagonal row wins the game .We make Tic Tac Toe game successfully with the help of C language and it is very entertaining.Some of my friends also help me in understanding the concept of this tic tac toe game as I know also about the concept of this game from my childhood because I use to play with my friends.

Reference-

Wikipedia

Let us c

Google

Stack flow

Youtube