TEAM MEMBERS:

NAME	Registration No.
GOURAV SWAIN	20010454
OM PRAKASH MOHANTA	20010448
ADITYA SAUMYA	20010473
SONALI RAJ	20010456
ABHILIPSHA SAHOO	20010445

Content:

Declaration

*I hereby declare that the project work entitled "**Tic-tac-toe game using DFS algorithm**" submitted to the CV Raman Global University, is a record of an original work done by our team under the guidance of **Prof. Santosh Sharma Sir.***

NAME OF THE STUDENTS

Gourav Swain
Om Prakash Mohanta
Aditya Saumya
Sonali Raj
Abhilipsha Sahoo

ACKNOWLEDGEMENT

Firstly, we would like to thank our professor

Santosh Sharma Sir *for giving us this opportunity to express our views on this project. And also thank him for continuous guidance, monitoring, and constant encouragement throughout this project.*

We are thankful to those who have helped usthroughout our project work.

Lastly, we thank all our group members for finalizingthis project within a limited time frame, without whom this project would not be possible.

Abstract

Learning to the program cobly be analogous to acquiring expertise in abstract mathematics, which may be boring ordulljority of students. Thus, among the countless options to approach learning coding , acquiring conceps through game creation could pe the most enriching experience for students. Consequently, it is important to select a lucid and familiar game for students. Then, the following step is to choose a language that introduces the basic concepts of Depth First Search(DFS). Forthis paper, we chose the game of Tic-Tac-Toe, which is straight-forward for most people. The programming language chosen here is C.

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 developmentof Tic Tac Toe game in C language.

Introduction :

The Tic Tac Toe game is a game for two players, called "X" and "O", who take turns marking the spaces in a 3×3 grid. The player who succeeded in placing three respective marks in a horizontal, vertical, or diagonal row wins the game. The Tic Tac Toe is a great way to pass your free time whether you're standing in a line or spending time with your kids. Stop wasting paper and save trees. Because of the simplicity of Tic Tac Toe, it is often used as a pedagogical tool for teaching the concepts of good sportsmanship and the branch of artificial intelligence.

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

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 downright now to play ordinary three-by-three tic-tac-toe witha friend, what will probably happen is that every gamewill come out a tie. Both you and your friend can probably play perfectly, never making a mistake thatwould allow your opponent to win. But can you describehow you know where to move each turn? Most of thetime, you probably aren’t even aware of alternative possibilities; you just look at the board and instantly knowwhere you want to move. That kind of instant knowledgeis great for human beings, because it makes you a fast player. But it isn’t much help in writing a computer program.

The tic-tac-toe game is played on a 3x3 grid the game is played by two players, who take turns. The first player marks moves with a circle, the second with a cross. The player who has formed a horizontal, vertical, or diag-onal sequence of three marks wins.

Problem Statement:

Since the 1970s, people started to take interest in using their computers as an entertainment environment, thus, the multi billion game industry was starting to take shape. Having presented earlier the sum of money this industry produces, I decided to have a go and create a game of my own. As a kid, I was always fascinated by the idea of becoming a game developer, but, as yearswent by, I have realized this is not exactly what programming and computer science, as a practice,are about and I dropped the idea. However, the third year project offered me the possibility to try and achieve one of my childhood's dreams and I couldn't resist the temptation

Objectives Of The Gaming System:

The game is developed for full-time entertainment and enthusiasms. It teaches the Gamer to bealert at every situation he/she faces, because if the Gamer is not fully alert and notice thesaucerfire he/she must be hit by the saucer-bombs. Though the proposed game is an action game, it doesn't involve direct violence. No zombie killing, animal killing, or human killing is performed in the game. So it can also be viewed as a non-violencegame. Kids can also play this game, because the design of the game is very simple, controlling the gameis very easy – pressing some neighboring keys of the keyboard

Strategy/Algorithm Used in Code

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.

2. If the other player can win on the next move,block thatwinning square.Here are the lowest-priority rules,used only if there isnothing 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.

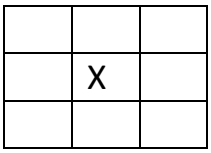
The highest priority rules are the ones dealing with themost urgent situations:either I or my opponent can win onthe next move.The lowest priority ones deal with the leasturgent situations,in which there is nothing special aboutthe moves already made to guide me.What was harder was to find the rules in between.I knewthat the goal of myown tic-tac-toe strategy was to set up afork, a board position in which I havetwo winningmoves, so my opponent can only block one of them.

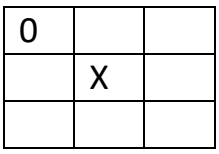
Here is an example:

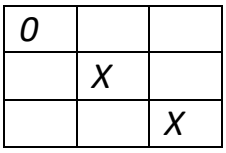
X	0	
	X	
X		0

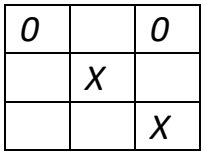
X can win by playing in square 3 orsquare 4. It's O's turn,but poor O can only block one of those squares at a time. Whichever O picks, Xwill then win by picking the other one. Given this concept of forking, I decided to use it asthe next highest priority rule:3. If I can make a move that will set up 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. Ineeded to add something, but I had trouble finding a goodrule 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 movinginto the square thatis common to his two winning combinations.

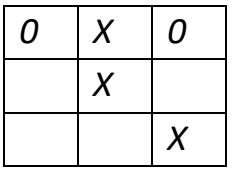
In other words, apply the same search technique to the opponent's position that I applied to my own.Thisstrategy works well in many cases, but not all. Forexample, here is a sequence of moves under thisstrategy,with the human player moving first:



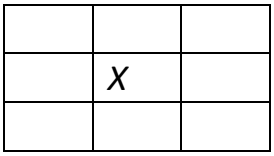


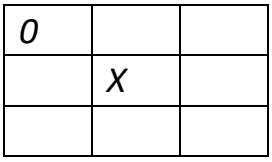


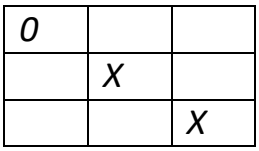


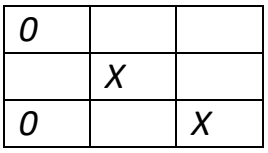


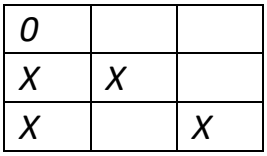
In the fourth grid, the computer (playing O) has discovered that X can set up a fork by moving in square 6, between the winning combinations 456and 369. The computer moves to block this fork.Unfortunately,X can also set up a fork by moving in squares 3, 7, or 8. The computer's move in square 6 has blocked 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 and789.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 square6? No. Here is something O could have done:

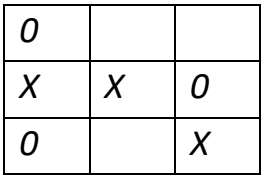












In this sequence, the computer's second move is in square7. This move also blocks a fork, but it wasn't chosen for that reason. Instead, it was chosen to force X's next move. In the fifth grid, X has had to move in square 4, to prevent an immediate win by O. The advantage of this situation for O is that 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 thanfor 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 square7. 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:4. If I canmake 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. Whatif 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 thatthere is only one situation in which this can happen:

X		

X		
	0	

X		
	0	
		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		

X		
	O	

X		
	O	
		X

X		O
	O	
		X

X		O
	O	
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 form 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 thinkeither about forcing alone (rule 4b) or about the opponent's forks alone (rule 4a). As it turns out, the board 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 verticalcombination is always okay. Therefore, I was able toimplement 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.

DFS(Depth First Search)Algorithm:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT

Pseudocode:

```
DFS(G,v) ( v is the vertex where the search starts )
    Stack S := {};  ( start with an empty stack )
    for each vertex u, set visited[u] := false;
    push S, v;
    while (S is not empty) do
      u := pop S;
      if (not visited[u]) then
```

```
      visited[u] := true;
      for each unvisited neighbour w of uu
         push S, w;
    end if
  end while
END DFS()
```

Source Code:

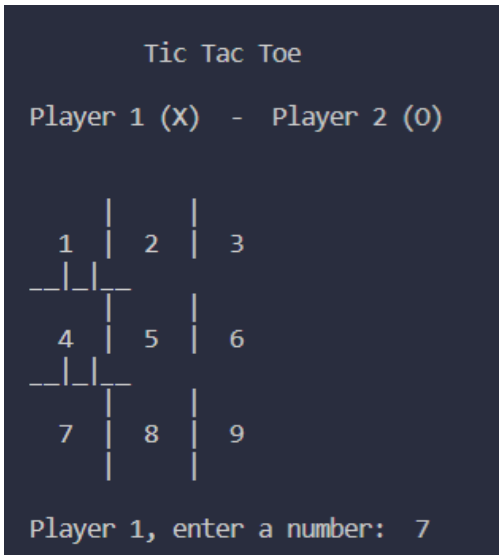
```
#include <stdio.h>
#include <conio.h>

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;
```

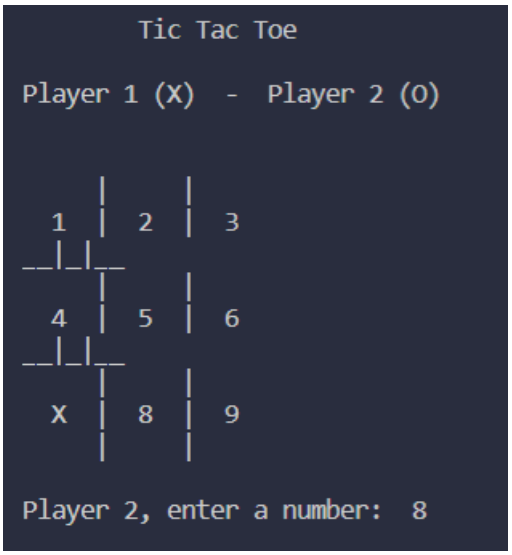
```
        else if (choice == 4 && square[4] == '4')
            square[4] = mark;

        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;
}

int checkwin()
{
    if (square[1] == square[2] && square[2] == square[3])
        return 1;
```

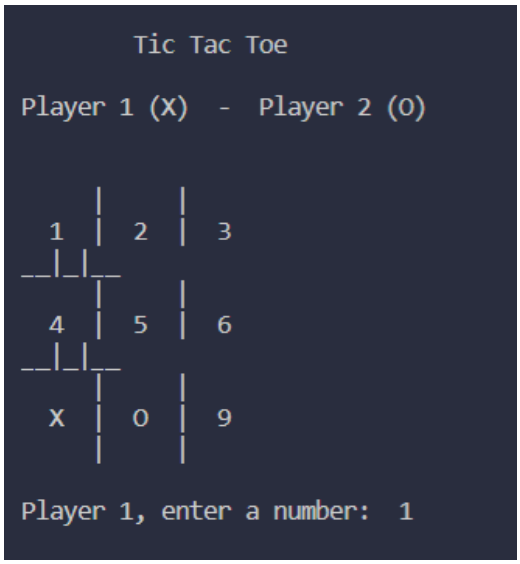
```
    else if (square[4] == square[5] && square[5] == square[6])
        return 1;

    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;
}

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]);
```

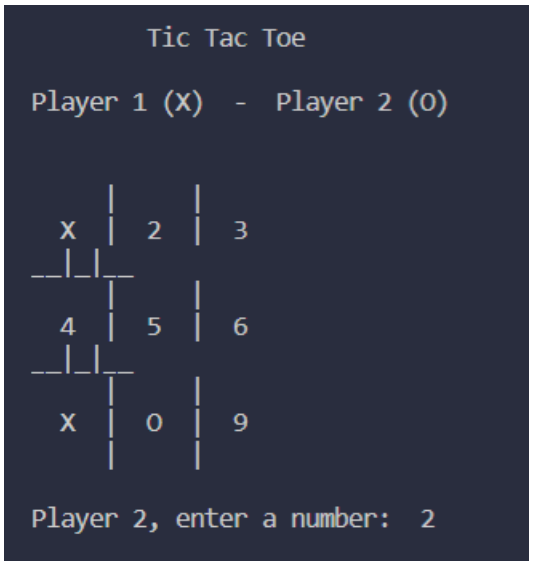
```
    printf("__|_|__\n");
    printf("    |   |    \n");

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

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

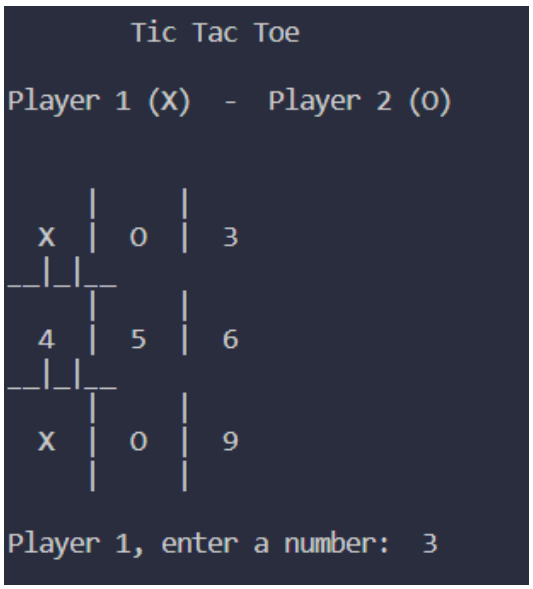
Output:

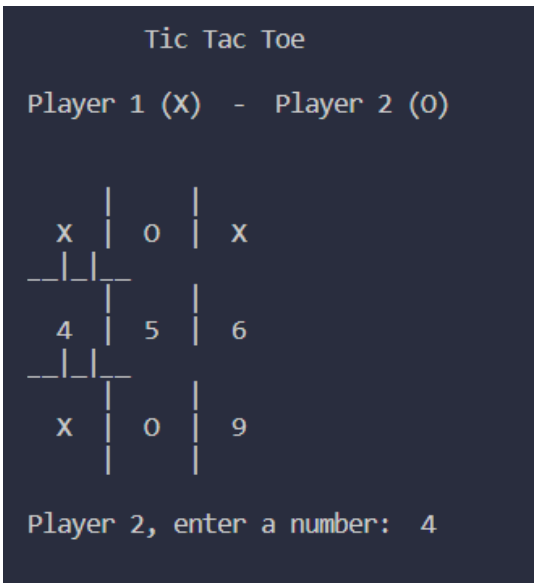


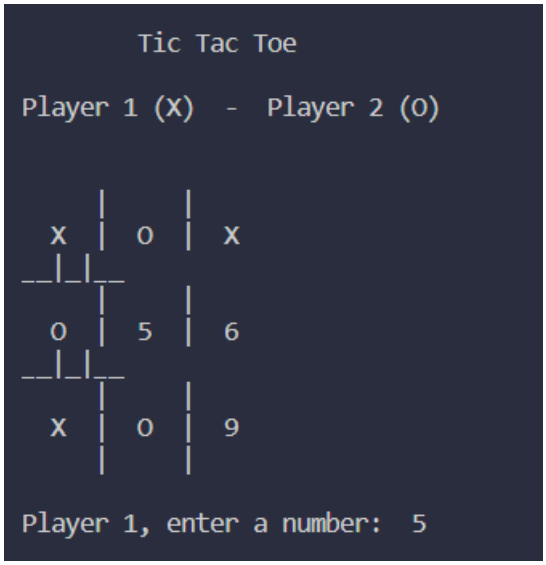


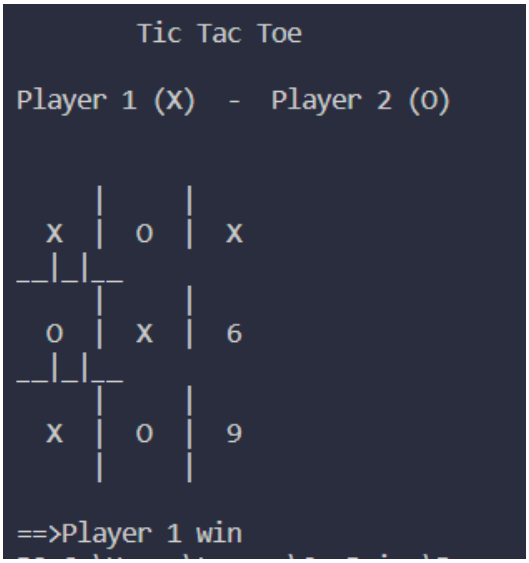












Conclusion:

The Tic Tac Toe game is most familiar among all the age groups. Intelligence can be a property of any purpose-driven decision maker. This basic idea has been suggested many times. An algorithm of playing Tic Tac Toe has been presentedand tested that works in efficient way. Overall the system works without any bugs.