**Experiment:2**

**Aim: Write a Program to Implement** **Tic-Tac-Toe game using Depth First Search algorithm.**

**Description of Tic-Tac-Toe game:**

Tic-tac-toe (also known as noughts and crosses or Xs and Os) is 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 horizontal, vertical, or diagonal row wins the game. The following example game is won by the first player, X:

A game with red circles and blue xs

Description automatically generated

Players soon discover that the best play from both parties leads to a draw. Hence, tic-tac-toe is most often played by young children. 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 that deals with the searching of game trees. It is straightforward to write a computer program to play tic-tac-toe perfectly or to enumerate the 765 essentially different positions (the state space complexity) or the 26,830 possible games up to rotations and reflections (the game tree complexity) on this space.

**Algorithm to decide the next move:**

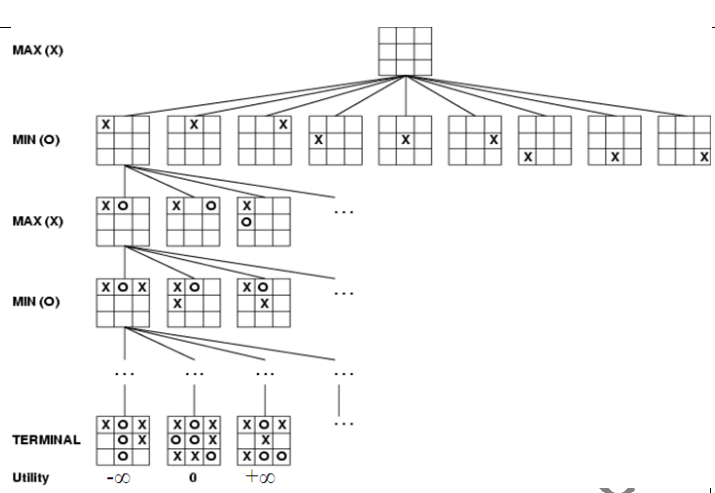
**Board Position:** A structure containing a nine-element vector representing the board, a list board positions that could result from the next move, and a number representing an estimate of how likely the board position is to lead to an ultimate win for the player to move.

**The Algorithm:** To decide on the next move, look ahead at the positions that result from each possible move. Decide which position is best, make the move that leads to that position, and assign the rating of that best move to the current position. To decide which of a set of board positions is best, do the following for each of them:

1. See if it is a win. If so, call it the best by giving it the highest possible rating.

2. Otherwise, consider all the moves the opponent could make next. See which of them is worst for us. Assume the opponent will make that move. Whatever rating those move has, assign it to node we are considering.

3. The best node is then the one with the highest rating.



**Implementation**

It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is like the BFS algorithm.

The step-by-step process to implement the DFS traversal is given as follows -

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back, and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

**Implementation of DFS on Tic-Tac-Toe game -:**

This technique verifies whether the game is ended by seeing if the board is filled or if one player won. If so, give back the game's outcome. Go over each board square once. Go on to the next square if the current one is occupied. Depending on the colour of the player in question, set the square to either a "X" or a "O”, to find the game's result, recursively run the same procedure with the modified board and the turn boolean switched. Update this branch's best result, aiming to optimize the outcome for the active player.

# Set up the game board as a list

board **=** ["-", "-", "-",

         "-", "-", "-",

         "-", "-", "-"]

# Define a function to print the game board

**def** print\_board():

**print**(board[0] **+** " | " **+** board[1] **+** " | " **+** board[2])

    print(board[3] **+** " | " **+** board[4] **+** " | " **+** board[5])

**print**(board[6] **+** " | " **+** board[7] **+** " | " **+** board[8])

# Define a function to handle a player's turn

**def** take\_turn(player):

**print**(player **+** "'s turn.")

    position **=** input("Choose a position from 1-9: ")

**while** position **not** **in** ["1", "2", "3", "4", "5", "6", "7", "8", "9"]:

        position **=** input("Invalid input. Choose a position from 1-9: ")

    position **=** int(position) **-** 1

**while** board[position] !**=** "-":

        position **=** int(input("Position already taken. Choose a different position: ")) **-** 1

    board[position] **=** player

    print\_board()

# Define a function to check if the game is over

**def** check\_game\_over():

    # Check for a win

**if** (board[0] **==** board[1] **==** board[2] !**=** "-") **or** \

       (board[3] **==** board[4] **==** board[5] !**=** "-") **or** \

       (board[6] **==** board[7] **==** board[8] !**=** "-") **or** \

       (board[0] **==** board[3] **==** board[6] !**=** "-") **or** \

       (board[1] **==** board[4] **==** board[7] !**=** "-") **or** \

       (board[2] **==** board[5] **==** board[8] !**=** "-") **or** \

       (board[0] **==** board[4] **==** board[8] !**=** "-") **or** \

       (board[2] **==** board[4] **==** board[6] !**=** "-"):

**return** "win"

    # Check for a tie

**elif** "-" **not** **in** board:

**return** "tie"

    # Game is not over

**else**:

**return** "play"

# Define the main game loop

**def** play\_game():

    print\_board()

    current\_player **=** "X"

    game\_over **=** False

**while** **not** game\_over:

        take\_turn(current\_player)

        game\_result **=** check\_game\_over()

**if** game\_result **==** "win":

            print(current\_player **+** " wins!")

 game\_over **=** True

**elif** game\_result **==** "tie":

            print("It's a tie!")

            game\_over **=** True

**else**:

            # Switch to the other player

            current\_player **=** "O" **if** current\_player **==** "X" **else** "X"

# Start the game

play\_game()

**Output**

- | - | -

- | - | -

- | - | -

X's turn.

Choose a position from 1-9: 1

X | - | -

- | - | -

- | - | -

O's turn.

Choose a position from 1-9: 3

X | - | O

- | - | -

- | - | -

X's turn.

Choose a position from 1-9: 5

X | - | O

- | X | -

- | - | -

O's turn.

Choose a position from 1-9: 2

X | O | O

- | X | -

- | - | -

X's turn.

Choose a position from 1-9: 9

X | O | O

- | X | -

- | - | X

X wins!

**Date of experiment performed:**

**Day of experiment performed:**

**Date of experiment submission:**

**Day of experiment Submission:**

Faculty Co-ordinator Signature