Experiment No: 7 Date:

Tic Tac Toe

Aim: To Implement the Tic Tac Toe Problem using MinMax Algorithm

Theory:

Tic Tac Toe is a classic two-player game where each player takes turns marking spaces on a 3x3 grid. The goal is to get three of your symbols (either "X" or "O") in a row, column, or diagonal.

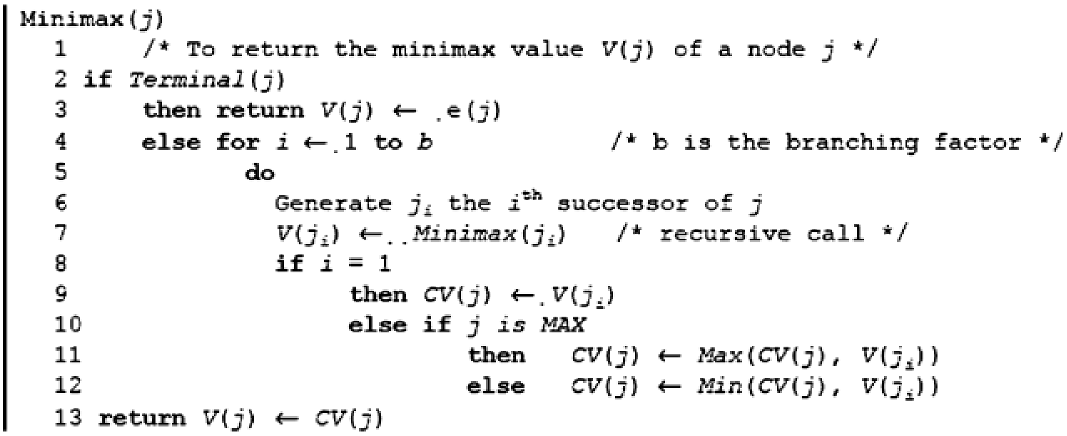
The Minimax algorithm is a decision-making algorithm used in game theory and decision theory for minimizing the possible loss for a worst-case scenario. It is commonly used in turn-based games like Tic Tac Toe to determine the best move for a player.

The Minimax algorithm works by recursively exploring all possible moves that can be made from the current state of the game. It evaluates each possible move by assuming that both players will make optimal moves and then choosing the move that maximizes the player's chances of winning or minimizes the opponent's chances of winning. This process continues until a terminal state (win, lose, or draw) is reached.

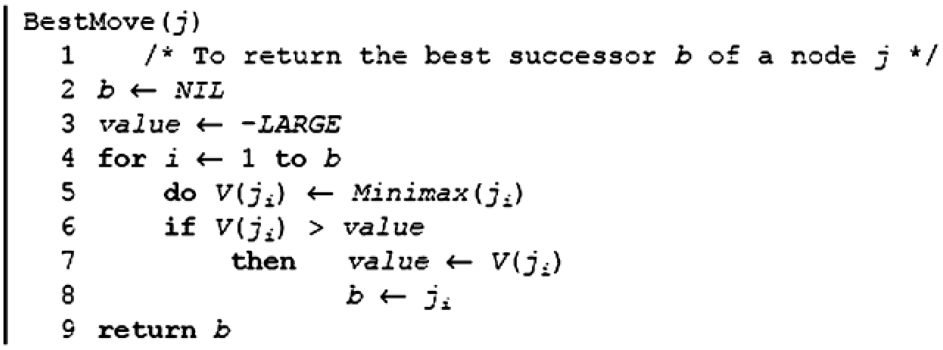
In Tic Tac Toe, the Minimax algorithm can be implemented by assigning a score to each possible move and then choosing the move with the highest score for the maximizing player (usually the computer) and the move with the lowest score for the minimizing player (the human player).

The algorithm considers all possible future game states by recursively exploring the game tree until it reaches a terminal state. It then assigns a score to each terminal state based on whether it results in a win, loss, or draw. These scores are propagated back up the tree, and the algorithm selects the move that leads to the best possible outcome for the current player.

Algorithm:

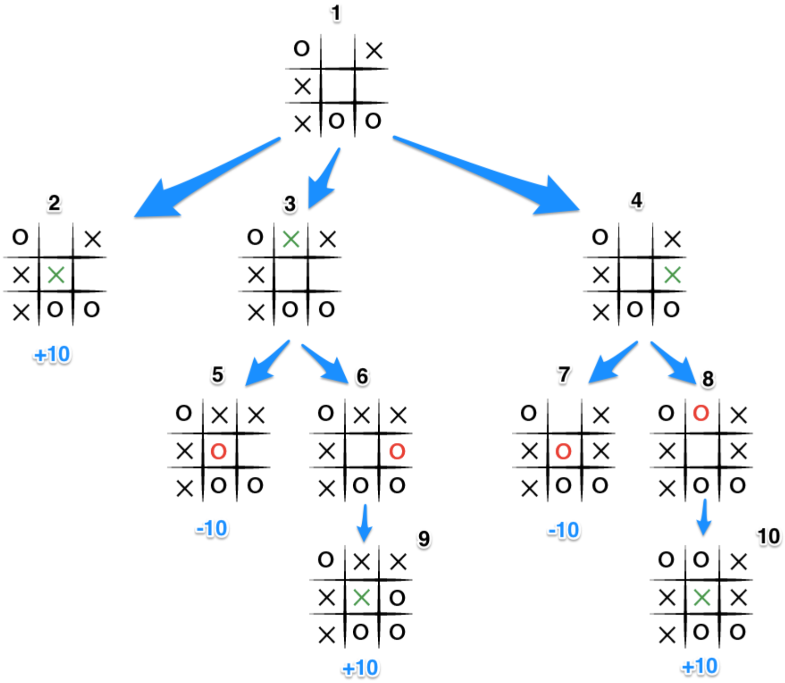


```
Minimax(j)
     /* To return the minimax value V(j) of a node j */
  if Terminal(j)
     then return V(j) ← e(j)
     else for i ← 1 to b            /* b is the branching factor */
            do
              Generate $j_i$ the $i^{th}$ successor of j
              $V(j_i)$ ← Minimax($j_i$)   /* recursive call */
              if i = 1
                 then CV(j) ← $V(j_i)$
                 else if j is MAX
                        then   CV(j) ← Max(CV(j), $V(j_i)$)
                        else   CV(j) ← Min(CV(j), $V(j_i)$)
  return V(j) ← CV(j)
```



```
BestMove(j)
    /* To return the best successor b of a node j */
b ← NIL
value ← -LARGE
for i ← 1 to b
    do V(j_i) ← Minimax(j_i)
    if V(j_i) > value
        then   value ← V(j_i)
               b ← j_i
return b
```

Example:



In the Minimax algorithm, the scoring system for Tic Tac Toe typically assigns scores as follows:

- If the computer (maximizing player) wins, assign a positive score (e.g., +10).
- If the opponent (minimizing player) wins, assign a negative score (e.g., -10).
- If the game ends in a draw, assign a neutral score (e.g., 0).
-

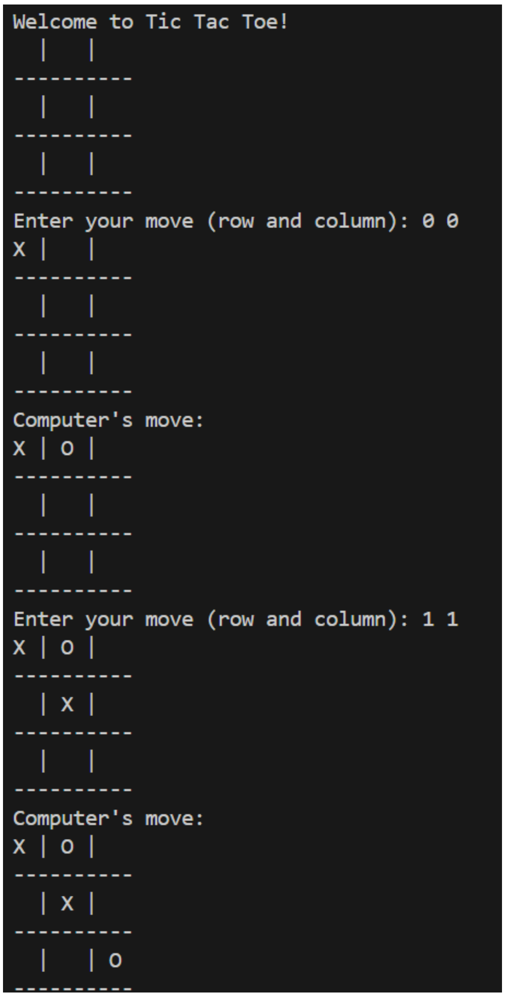
These scores represent the desirability of a particular game state from the perspective of the current player. The algorithm aims to maximize its score (if it's the maximizing player) or minimize its score (if it's the minimizing player) by recursively exploring all possible moves.

Program:

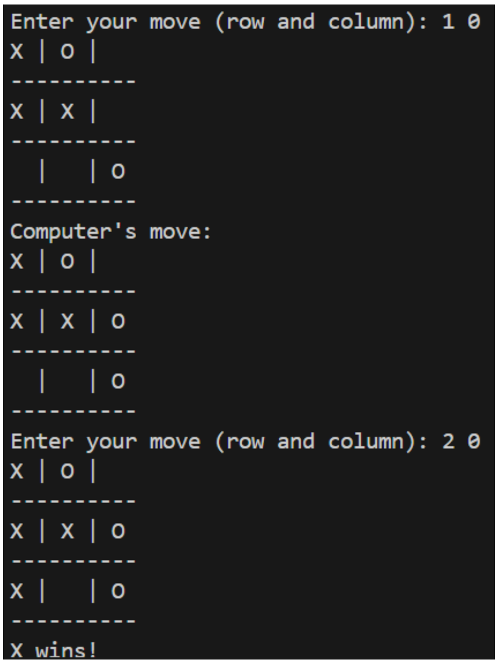
```
import sys

def print_board(board):
   for row in board:
      print(" | ".join(row))
      print("-" * 10)

def evaluate(board):
   # Check rows
   for row in board:
      if row.count("X") == 3:
         return 10
      elif row.count("O") == 3:
         return -10

   # Check columns
   for col in range(3):
      if board[0][col] == board[1][col] == board[2][col]:
         if board[0][col] == "X":
            return 10
         elif board[0][col] == "O":
            return -10

   # Check diagonals
   if board[0][0] == board[1][1] == board[2][2]:
      if board[0][0] == "X":
         return 10
      elif board[0][0] == "O":
         return -10
   if board[0][2] == board[1][1] == board[2][0]:
      if board[0][2] == "X":
         return 10
      elif board[0][2] == "O":
         return -10

   # If no one wins
   return 0

#check if any move left
def is_moves_left(board):
   for row in board:
      if " " in row:
         return True
   return False
```

```
def minimax(board, depth, is_max):
    score = evaluate(board)

    # If maximizer wins
    if score == 10:
        return score - depth

    # If minimizer wins
    if score == -10:
        return score + depth

    # If there are no moves left and no one wins
    if not is_moves_left(board):
        return 0

    # If it's the maximizer's move
    if is_max:
        best = -sys.maxsize
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "X"
                    best = max(best, minimax(board, depth + 1, not is_max))
                    board[i][j] = " "
        return best
    else:
        best = sys.maxsize
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "O"
                    best = min(best, minimax(board, depth + 1, not is_max))
                    board[i][j] = " "
        return best

def find_best_move(board):
    best_val = -sys.maxsize
    best_move = (-1, -1)

    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = "X"
                move_val = minimax(board, 0, False)
                board[i][j] = " "
                if move_val > best_val:
                    best_move = (i, j)
                    best_val = move_val

    return best_move

def play_game():
    board = [[" " for _ in range(3)] for _ in range(3)]
    player = "X"

    print("Welcome to Tic Tac Toe!")
    print_board(board)

    while True:
        if player == "X":
            row, col = map(int, input("Enter your move (row and column): ").split())
            if board[row][col] != " ":
                print("Invalid move. Try again.")
                continue
            board[row][col] = player
        else:
            print("Computer's move:")
            row, col = find_best_move(board)
            board[row][col] = player

        print_board(board)
        result = evaluate(board)
        if result == 10:
            print("X wins!")
            break
        elif result == -10:
            print("O wins!")
            break
        elif not is_moves_left(board):
            print("It's a draw!")
            break

        player = "O" if player == "X" else "X"

play_game()
```

Output



```
Welcome to Tic Tac Toe!
  |   |
----------
  |   |
----------
  |   |
----------
Enter your move (row and column): 0 0
X |   |
----------
  |   |
----------
  |   |
----------
Computer's move:
X | O |
----------
  |   |
----------
  |   |
----------
Enter your move (row and column): 1 1
X | O |
----------
  | X |
----------
  |   |
----------
Computer's move:
X | O |
----------
  | X |
----------
  |   | O
----------
```



```
Enter your move (row and column): 1 0
X | O |
----------
X | X |
----------
  |   | O
----------
Computer's move:
X | O |
----------
X | X | O
----------
  |   | O
----------
Enter your move (row and column): 2 0
X | O |
----------
X | X | O
----------
X |   | O
----------
X wins!
```

Conclusion: Solved Tic Tac Toe problem using MinMax Algorithm with successful execution of programs.