Batch: A1 **Experiment Number: 4**

Roll Number: 16010422012 **Name: Ayaan Bhoje**

Aim of the Experiment: Implementation of Adversarial algorithm-Min-Max for Tic-Tac-Toe Game

Program/ Steps:

```
class TicTacToe:
    def __init__(self):
        self.board = [['X', 'O', 'X'],
                      ['O', 'X', 'O'],
                      [' ', ' ', ' ']]

    def display_board(self):
        for row in self.board:
            print("|".join(row))
            print("-----")

    def is_winner(self, player):
        for i in range(3):
            if all(cell == player for cell in self.board[i]) or \
               all(row[i] == player for row in self.board) or \
               all(self.board[i][i] == player for i in range(3)) or \
               all(self.board[i][2 - i] == player for i in range(3)):
                return True
        return False

    def is_draw(self):
        return all(cell != ' ' for row in self.board for cell in row)

    def is_game_over(self):
        return self.is_winner('X') or self.is_winner('O') or self.is_draw()


class AiPlayer:
    def __init__(self, symbol):
        self.symbol = symbol

    def get_move(self, board):
        best_score = float('-inf')
        best_move = None

        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = self.symbol
                    score = self.minimax(board, False)
                    board[i][j] = ' '

                    if score > best_score:
                        best_score = score
                        best_move = (i, j)

        return best_move

    def minimax(self, board, is_maximizing):
        if TicTacToe().is_winner('X'):
            return -1
        elif TicTacToe().is_winner('O'):
            return 1
        elif TicTacToe().is_draw():
            return 0

        if is_maximizing:
            best_score = float('-inf')
            for i in range(3):
                for j in range(3):
                    if board[i][j] == ' ':
                        board[i][j] = self.symbol
                        score = self.minimax(board, False)
                        board[i][j] = ' '
                        best_score = max(score, best_score)
            return best_score
        else:
            best_score = float('inf')
            for i in range(3):
                for j in range(3):
                    if board[i][j] == ' ':
                        board[i][j] = 'X' if self.symbol == 'O' else 'O'
                        score = self.minimax(board, True)
                        board[i][j] = ' '
                        best_score = min(score, best_score)
            return best_score


class HuPlayer:
    def __init__(self, symbol):
        self.symbol = symbol
    def get_move(self):
        row = int(input("Enter row (0, 1, 2): "))
        col = int(input("Enter column (0, 1, 2):
        ")) return row, col


tic_tac_toe = TicTacToe()
ai_player = AiPlayer('O')
hu_player = HuPlayer('X')

while not tic_tac_toe.is_game_over():
    tic_tac_toe.display_board()
    row, col = hu_player.get_move()
    tic_tac_toe.board[row][col] =
    hu_player.symbol if
    tic_tac_toe.is_game_over():
        break
    ai_row, ai_col =
    ai_player.get_move(tic_tac_toe.board)
    tic_tac_toe.board[ai_row][ai_col] = ai_player.symbol

tic_tac_toe.display_board()
```

Output/Result:

```
X|O|X

-----

O|X|O

-----

 | |

-----

Enter row (0, 1, 2): 1

Enter column (0, 1, 2): 2

X|O|X

-----
```

```
O|X|X

-----

O| |

-----

Enter row (0, 1, 2): 2

Enter column (0, 1, 2): 0

X|O|X

-----

O|X|X

-----

X| |

-----
```

Post Lab Question-Answers:

1. **Game playing is often called as an**

    a) Non-adversial search
    b) Adversial search
    c) Sequential search

    d) None of the above

2. **What are the basic requirements or need of AI search methods in game playing?**

    a) Initial State of the game

b) Operators defining legal moves
c) Successor functions

Outcomes: Analyse and formalized the problem and select the appropriate search method and write the algorithm.

Conclusion (based on the Results and outcomes achieved):

In this implementation, we've developed a simple Tic Tac Toe game where a human player can play against an AI player using the Minimax algorithm. The game provides a console-based interface where players can input their moves, and the game state is displayed after each move.

The `TicTacToe` class handles the game logic, including checking for a winner, detecting a draw, and determining if the game is over. The `AiPlayer` class represents the AI player, which utilizes the Minimax algorithm to make optimal moves. The `HuPlayer` class represents the human player.

The game proceeds with each player taking turns making moves until either one player wins, the game ends in a draw, or the player quits.

This implementation provides a basic demonstration of using the Minimax algorithm in a simple game scenario and can be extended further with additional features such as improved AI strategies, graphical user interface, and multiplayer support.

References:

1. How to make your Tic Tac Toe game unbeatable by using the minimax algorithm:
https://www.freecodecamp.org/news/how-to-make-your-tic-tac-toe-game-unbeatable
by-using-the-minimax-algorithm
9d690bad4b37/#:~:text=A%20Minimax%20algorithm%20can%20be,on%20each%20
a vailable%20spot%20(recursion)

2. Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach,
2ndEdition, Pearson Publication

1. Elaine Rich, Kevin Knight, Artificial Intelligence, Tata McGraw Hill, 1999.