**Batch: A2** **Roll No.: 16010421063** **Experiment No.: 4**

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

**Program/ Steps:**

```
import random

class TicTacToe(object):
    winning_combos = (
        [0, 1, 2], [3, 4, 5], [6, 7, 8],
        [0, 3, 6], [1, 4, 7], [2, 5, 8],
        [0, 4, 8], [2, 4, 6]
    )

    winners = ('X-win', 'Draw', 'O-win')

    def_init_(self, board=[]):

        if len(board) == 0:
            self.board = [0 for i in range(9)]
        else:
            self.board = board

    def print_board(self):

        for i in range(3):
            print(
                "| " + str(self.board[i * 3]) +
                " | " + str(self.board[i * 3 + 1]) +
                " | " + str(self.board[i * 3 + 2]) + " |"
            )

    def check_game_over(self):

        if 0 not in [element for element in self.board]:
            return True
        if self.winner() != 0:
            return True
        return False

    def available_moves(self):

        return [index for index, element in enumerate(self.board) if element == 0]
```

```
    def available_combos(self, player):
        return self.available_moves() + self.get_acquired_places(player)
    def X_won(self):
        return self.winner() == 'X'

    def O_won(self):
        return self.winner() == 'O'

    def is_tie(self):
        return self.winner() == 0 and self.check_game_over()

    def winner(self):

        for player in ('X', 'O'):
            positions = self.get_acquired_places(player)
            for combo in self.winning_combos:
                win = True
                for pos in combo:
                    if pos not in positions:
                        win = False
                if win:
                    return player
        return 0

    def get_acquired_places(self, player):

        return [index for index, element in enumerate(self.board) if element ==
player]

    def make_move(self, position, player):
        self.board[position] = player
    def minimax(self, node, player):

        if node.check_game_over():

            if node.X_won():
                return -1
            elif node.is_tie():
                return 0
            elif node.O_won():
                return 1
        best = 0
        for move in node.available_moves():
            node.make_move(move, player)
            val = self.minimax(node, get_enemy(player))
            node.make_move(move, 0)
            if player == 'O':
                if val > best:
                    best = val
```

```
            else:
                if val < best:
                    best = val
        return best

def determine(board, player):
    a = 0
    choices = []
    if len(board.available_moves()) == 9:
        return 4
    for move in board.available_moves():
        board.make_move(move, player)
        val = board.minimax(board, get_enemy(player))
        board.make_move(move, 0)
        if val > a:
            a = val
            choices = [move]
        elif val == a:
            choices.append(move)
    try:
        return random.choice(choices)
    except IndexError:
        return random.choice(board.available_moves())


def get_enemy(player):
    if player == 'X':
        return 'O'
    return 'X'


if_name____== "_main_":
    board = TicTacToe()
    print('\n You: X  \n Computer: Y\nBoard positions are like this: ')
    for i in range(3):
        print(
            "| " + str(i * 3 + 1) +
            " | " + str(i * 3 + 2) +
            " | " + str(i * 3 + 3) + " |"
        )
    print('Type position no. for your move')
    while not board.check_game_over():
        player = 'X'
        player_move = int(input("Your Move: ")) - 1
        if player_move not in board.available_moves():
            print('Move not available!')
            continue
        board.make_move(player_move, player)
        board.print_board()
        print()
```

```
        if board.check_game_over():
            break
print('Ai is playing.. ') player =
        get_enemy(player)
computer_move = determine(board, player)
        board.make_move(computer_move, player)
        board.print_board()
    if board.winner() != 0:
if board.winner() == 'X':
            print ("Congrats you win!")
        else:
print('Computer Wins!')
else:
print("Game tied!")
```

---

**Output/Result:**

**Case 1: Game tied**

```
Board positions are like this:
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
Type in the position number you to make a move on..
Your Move: 3
| 0 | 0 | X |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Computer is playing..
| 0 | 0 | X |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
Your Move: 1
| X | 0 | X |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
```

```
Computer is playing..
| X | O | X |
| O | O | O |
| O | O | O |
Your Move: 2
Move not available!
Your Move: 5
| X | O | X |
| O | X | O |
| O | O | O |

Computer is playing..
| X | O | X |
| O | X | O |
| O | O | O |
Your Move: 2
Move not available!
Your Move: 7
Move not available!
Your Move: 9
```

```
Your Move: 7
Move not available!
Your Move: 9
Move not available!
Your Move: 8
| X | O | X |
| O | X | O |
| O | X | O |

Computer is playing..
| X | O | X |
| O | X | O |
| O | X | O |
Your Move: 6
| X | O | X |
| O | X | X |
| O | X | O |

Game tied!
>
```

## Case 2: Human wins

```
You: X
 Computer: Y
Board positions are like this:
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
Type position no. for your move
Your Move: 3
| O | O | X |
| O | O | O |
| O | O | O |

Ai is playing..
| O | O | X |
| O | O | O |
| O | O | O |
Your Move: 9
| O | O | X |
| O | O | O |
| O | O | X |
```

```
Ai is playing..
| O | O | X |
| O | O | O |
| O | O | X |
Your Move: 5
| O | O | X |
| O | X | O |
| O | O | X |

Ai is playing..
| O | O | X |
| O | X | O |
| O | O | X |
Your Move: 6
Move not available!
Your Move: 7
Move not available!
Your Move: 1
```

```
Ai is playing..
| O | O | X |
| O | X | O |
| O | O | X |
Your Move: 6
Move not available!
Your Move: 7
Move not available!
Your Move: 1
| X | O | X |
| O | X | O |
| O | O | X |

Congrats you win!
```

**Case 3: Ai wins**

```
Computer is playing..
| O | O | O |
| O | O | O |
| X | X | O |
Your Move: 9
Move not available!
Your Move: 5
| O | O | O |
| O | X | O |
| X | X | O |

Computer is playing..
| O | O | O |
| O | X | O |
| X | X | O |
Computer Wins!
```

---

**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**

**Ans: b) Adversial search**

**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**
**d) Goal test**
**e) Path cost**

**Ans: All of the above**

---

## Outcomes:

**CO2: Analyze and formalize the problem (as a state space, graph, etc.) and select the appropriate search method and write the algorithm.**

---

## Conclusion (based on the Results and outcomes achieved):

**Learnt about adversarial algorithm and successfully created a Tic-Tac-Toe game using Minmax algorithm and has 2 players –i)Human**
**ii)Ai**

---

## References:


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-

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

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