Batch: IAI-2 **Experiment Number: 4**

Roll Number: 16010422234 **Name: Chandana Ramesh Galgali**

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

---

Program/Steps:

1. Implement two players, 1) AiPlayer and 2) HuPlayer [AI and Human player] for a tic-tac-toe game.
2. For AiPlayer implement Minmax algorithm. [For simplicity first consider the start state as given in the figure 2 below. Once the program is working fine with this start state then change the start state to blank game board.]

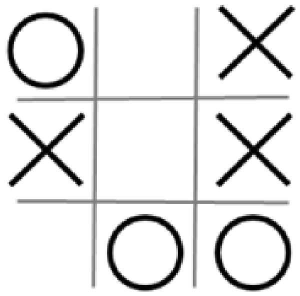


Figure 2: sample start state

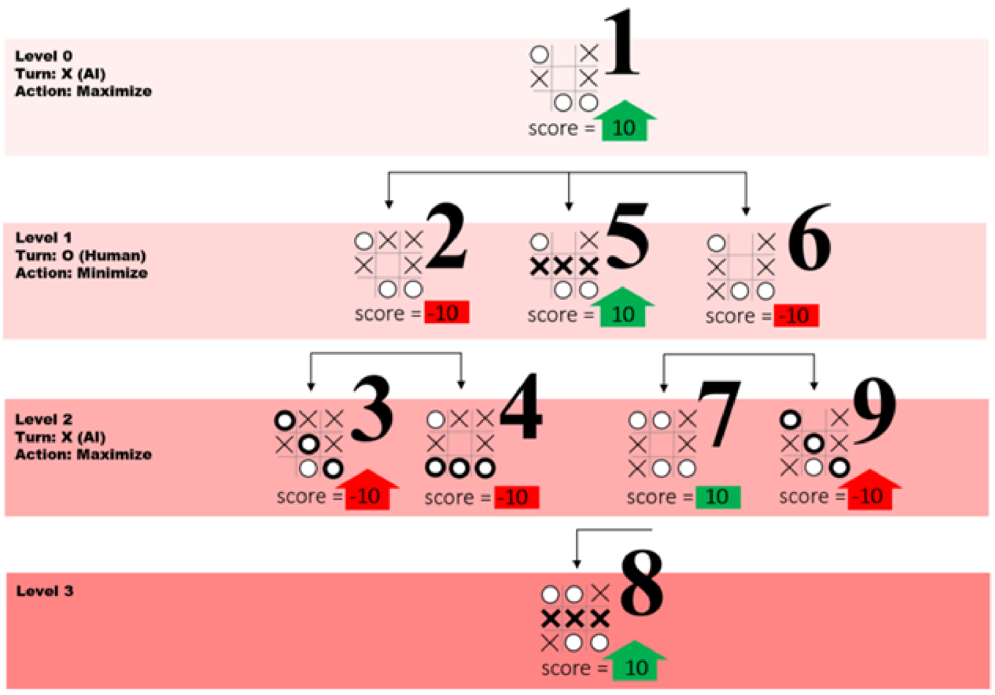


Figure 3 Minimax function call by function call

Code:

```
import math

class TicTacToe:
    def __init__(self):
        self.board = [' ']*9
        self.ai_player = None
        self.hu_player = None

    def print_board(self):
        for i in range(0, 9, 3):
            print(self.board[i], '|', self.board[i+1], '|',
self.board[i+2])
            if i < 6:
                print('---------')

    def empty_cells(self):
        return [i for i, cell in enumerate(self.board) if cell == ' ']

    def check_winner(self, player):
        win_conditions = [(0, 1, 2), (3, 4, 5), (6, 7, 8),
                          (0, 3, 6), (1, 4, 7), (2, 5, 8),
                          (0, 4, 8), (2, 4, 6)]
        for condition in win_conditions:
            if all(self.board[i] == player for i in condition):
                return True
        return False

    def check_draw(self):
        return ' ' not in self.board

    def game_over(self):
        return self.check_winner(self.ai_player) or
self.check_winner(self.hu_player) or self.check_draw()

    def minimax(self, depth, player):
        if player == self.ai_player:
            best = [-1, -math.inf]
        else:
            best = [-1, math.inf]
        if depth == 0 or self.game_over():
            score = self.evaluate()
            return [-1, score]

        for cell in self.empty_cells():
            self.board[cell] = player
            score = self.minimax(depth - 1, 'O' if player == 'X' else 'X')
            self.board[cell] = ' '

            score[0] = cell

            if player == self.ai_player:
                if score[1] > best[1]:
                    best = score
            else:
                if score[1] < best[1]:
                    best = score

        return best

    def ai_turn(self):
        depth = len(self.empty_cells())
        if depth == 0 or self.game_over():
            return

        if depth == 9:
            cell = 0
        else:
            cell = self.minimax(depth, self.ai_player)[0]
        self.board[cell] = self.ai_player

    def hu_turn(self):
        while True:
            move = int(input('Enter your move (1-9): ')) - 1
            if move in self.empty_cells():
                self.board[move] = self.hu_player
                break
            else:
                print('Invalid move! Try again.')
    def evaluate(self):
        if self.check_winner(self.ai_player):
            return 1
        elif self.check_winner(self.hu_player):
            return -1
        else:
            return 0

    def play(self):
        print("Welcome to Tic-Tac-Toe!")
        player_choice = input("Do you want to be 'X' or 'O'? ").upper()
        if player_choice == 'X':
            self.ai_player = 'O'
            self.hu_player = 'X'
        else:
            self.ai_player = 'X'
            self.hu_player = 'O'

        print(f"You are '{self.hu_player}', and AI is
'{self.ai_player}'.")
        print("You play by entering the number of the cell you want to
mark.")

        while not self.game_over():
            self.print_board()
            if self.hu_player == 'X':
                self.hu_turn()
            else:
                self.ai_turn()

            if self.check_winner(self.hu_player):
                self.print_board()
                print("Congratulations! You win!")
                break
            elif self.check_draw():
                self.print_board()
                print("It's a draw!")
                break
```

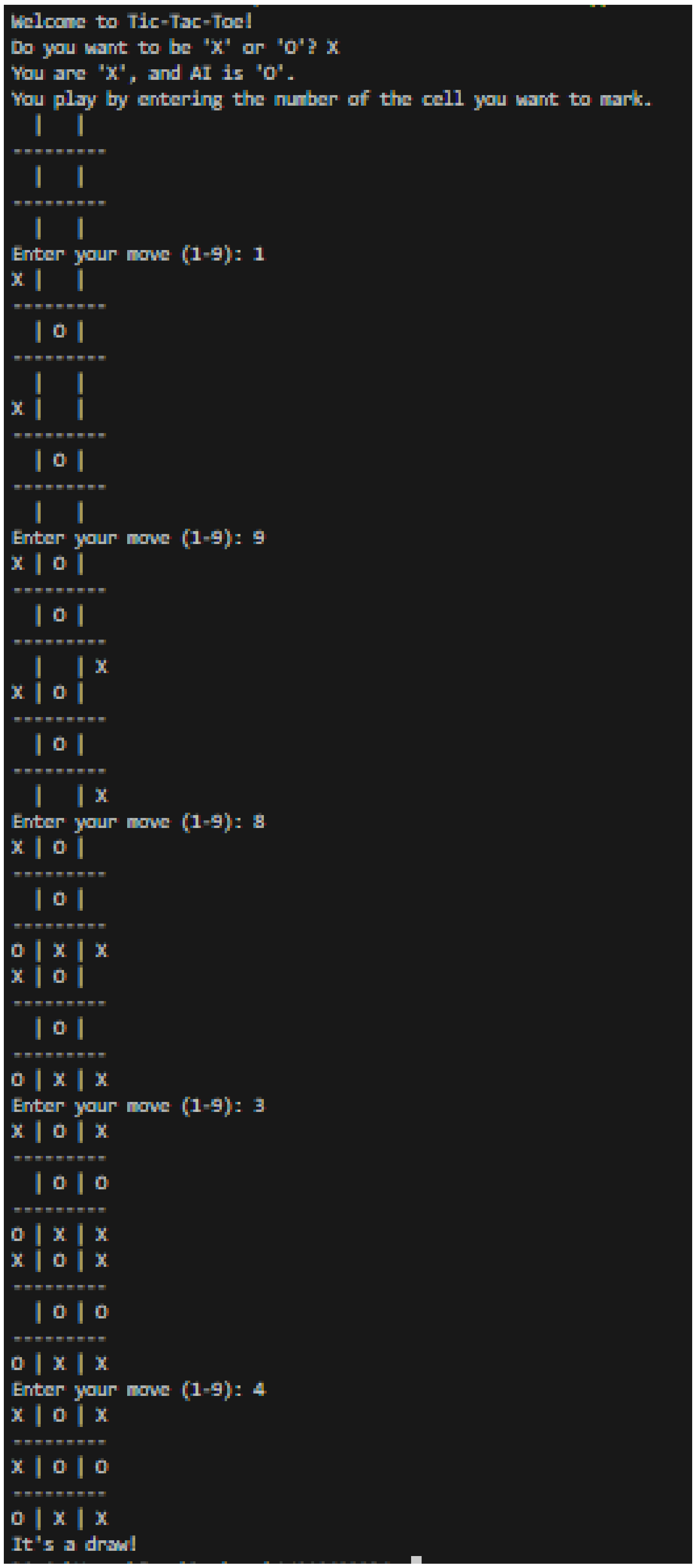
```python
            if self.check_winner(self.ai_player):
                self.print_board()
                print("AI wins! Better luck next time.")
                break
            elif self.check_draw():
                self.print_board()
                print("It's a draw!")
                break

            self.ai_turn()
            if self.check_winner(self.ai_player):
                self.print_board()
                print("AI wins! Better luck next time.")
                break
            elif self.check_draw():
                self.print_board()
                print("It's a draw!")
                break

            self.print_board()
            if self.check_winner(self.hu_player):
                print("Congratulations! You win!")
                break
            elif self.check_draw():
                print("It's a draw!")
                break

if __name__ == "__main__":
    game = TicTacToe()
    game.play()
```

Output/Result:



```
Welcome to Tic-Tac-Toe!
Do you want to be 'X' or 'O'? X
You are 'X', and AI is 'O'.
You play by entering the number of the cell you want to mark.
  |   |
---------
  |   |
---------
  |   |
Enter your move (1-9): 1
X |   |
---------
  | O |
---------
  |   |
X |   |
---------
  | O |
---------
  |   |
Enter your move (1-9): 9
X | O |
---------
  | O |
---------
  |   | X
X | O |
---------
  | O |
---------
  |   | X
Enter your move (1-9): 8
X | O |
---------
  | O |
---------
O | X | X
X | O |
---------
  | O |
---------
O | X | X
Enter your move (1-9): 3
X | O | X
---------
  | O | O
---------
O | X | X
X | O | X
---------
  | O | O
---------
O | X | X
Enter your move (1-9): 4
X | O | X
---------
X | O | O
---------
O | X | X
It's a draw!
```

Post-Lab Questions:

1. Game playing is often called as an

a) Non-adversarial search

b) Adversarial search

c) Sequential search

d) None of the above

2. What are the basic requirements or needs 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

Outcomes: 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):
The implementation of the Min-Max algorithm for the Tic-Tac-Toe game demonstrated its effectiveness in achieving optimal decision-making in adversarial environments. While exhibiting promising results, further research and experimentation are warranted to address scalability challenges and explore avenues for algorithmic improvements in more complex game domains.

References:

1. Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Second Edition, Pearson Publication
2. Luger, George F. Artificial Intelligence : Structures and strategies for complex problem solving, 2009, 6th Edition, Pearson Education