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

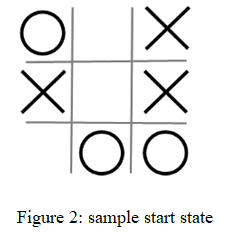
**Roll Number: 16010422234 Name: Chandana Ramesh Galgali**

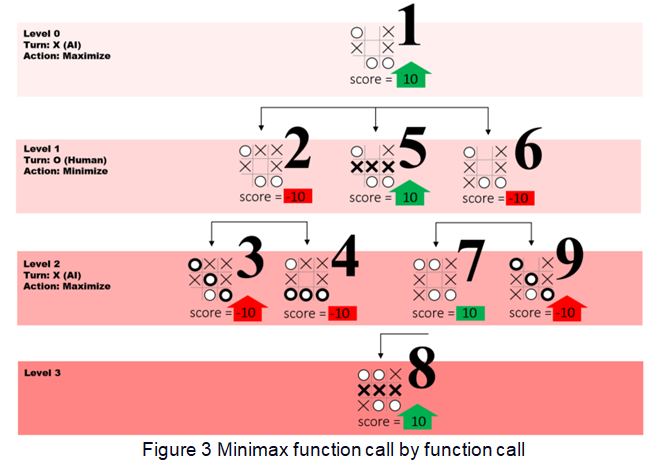
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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.]





\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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**

**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!")**

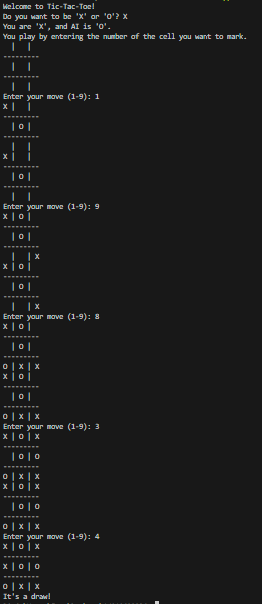
**break**

**if \_\_name\_\_ == "\_\_main\_\_":**

**game = TicTacToe()**

**game.play()**

**Output/Result:**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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