Date: 6 - 02 – 2026
Assignment No.: 3

Name: Varghese K James
Reg No.: 3122245001192

Excercise 3 – Tic-Tac-Toe Game using Minimax and AlphaBeta Pruning Search

Aim: Implement a Tic-Tac-Toe Game using Minimax and Alpha-Beta Pruning Search Methods

Source Code

```
import math

class TicTacToeGame:

    def __init__(self):
        self.board = [' ' for _ in range(9)]
        self.human = 'X'
        self.ai = 'O'
        self.current_player = self.human
        self.states_explored = 0

    def display_board(self):
        print("\n-------------")
        for i in range(0, 9, 3):
            print(f"| {self.board[i]} | {self.board[i+1]} | {self.board[i+2]} |")
            print("-------------")
        print()

    def is_moves_left(self, board) -> bool:
        if (' ' in board):
            return True
        else:
            return False

    def check_winner(self, board, player):
        winning_combos = [
            (0, 1, 2), (3, 4, 5), (6, 7, 8), #row
            (0, 3, 6), (1, 4, 7), (2, 5, 8), #column
            (0, 4, 8), (2, 4, 6) #diag
        ]

        for a, b, c in winning_combos:
            if board[a] == board[b] == board[c] == player:
                return True
        return False

    def is_terminal(self, board):
        return (self.check_winner(board, self.ai) or
                self.check_winner(board, self.human) or
                not self.is_moves_left(board))

    def utility(self, board) -> int:
        if self.check_winner(board, self.ai):
            return 1
```

```
        elif self.check_winner(board, self.human):
            return -1
        else:
            return 0

    def generate_successor_states(self, board, player):
        successors = []
        for i in range(9):
            if board[i] == ' ':
                new_state = list(board)
                new_state[i] = player
                successors.append(new_state)
        return successors

    def minimax(self, board, depth, is_maximizing):
        self.states_explored += 1

        score = self.utility(board)

        if self.is_terminal(board):
            return score

        if is_maximizing:
            best_val = -math.inf
            successors = self.generate_successor_states(board, self.ai)
            for state in successors:
                value = self.minimax(state, depth + 1, False)
                best_val = max(best_val, value)
            return best_val
        else:
            best_val = math.inf
            successors = self.generate_successor_states(board, self.human)
            for state in successors:
                value = self.minimax(state, depth + 1, True)
                best_val = min(best_val, value)
            return best_val

    def alpha_beta(self, board, depth, alpha, beta, is_maximizing):
        self.states_explored += 1

        if self.is_terminal(board):
            return self.utility(board)

        if is_maximizing:
            best_val = -math.inf
            successors = self.generate_successor_states(board, self.ai)

            for state in successors:
                value = self.alpha_beta(state, depth + 1, alpha, beta, False)
                best_val = max(best_val, value)
                alpha = max(alpha, best_val)

                if beta <= alpha:
                    break
            return best_val
        else:
```

```
            best_val = math.inf
            successors = self.generate_successor_states(board, self.human)

            for state in successors:
                value = self.alpha_beta(state, depth + 1, alpha, beta, True)
                best_val = min(best_val, value)
                beta = min(beta, best_val)

                if beta <= alpha:
                    break
            return best_val

    def find_best_move(self, algo):
        self.states_explored = 0

        best_val = -math.inf
        best_move = -1

        for i in range(9):
            if self.board[i] == ' ':
                self.board[i] = self.ai

                if (algo == 1):
                    move_val = self.minimax(self.board, 0, False)
                else:
                    move_val = self.alpha_beta(self.board, 0, -math.inf, math.inf, False)

                self.board[i] = ' '

                if move_val > best_val:
                    best_move = i
                    best_val = move_val

        print(f"States explored: {self.states_explored}")
        return best_move

    def play_game(self):
        print("<- Tic-Tac-Toe ->")

        algo = int(input(f"Minimax (1) or AlphaBeta Pruning (2): "))
        print("\nYou are 'X' and the AI is 'O'.")
        print("Enter positions 0-8 to play.")
        self.display_board()

        while not self.is_terminal(self.board):
            if self.current_player == self.human:
                while True:
                    try:
                        user_input = input(f"Enter move for {self.human} (0-8): ")
                        move = int(user_input)
                        if 0 <= move <= 8 and self.board[move] == ' ':
                            self.board[move] = self.human
                            self.current_player = self.ai
                            break
                        else:
                            print("Invalid move.")
```

Name: Varghese K James
Reg No.: 3122245001192

```
                    except ValueError:
                        print("Please enter a valid integer between 0 and 8.")
            else:
                move = self.find_best_move(algo)
                self.board[move] = self.ai
                self.current_player = self.human
                print(f"AI placed at pos {move}")

            self.display_board()

        if self.check_winner(self.board, self.ai):
            print("Game Over. AI Wins!")
        elif self.check_winner(self.board, self.human):
            print("Game Over. You Win!")
        else:
            print("Game Over. It's a Draw!")

if __name__ == "__main__":
    game = TicTacToeGame()
    game.play_game()
```

Date: 6 - 02 – 2026
Assignment No.: 3

Name: Varghese K James
Reg No.: 3122245001192

Input/Output

```
vicfic@NAG3:~/stud/ai/ex3$ python tic-tac-toe.py
<- Tic-Tac-Toe ->
Minimax (1) or AlphaBeta Pruning (2): 1

You are 'X' and the AI is 'O'.
Enter positions 0-8 to play.

-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------

Enter move for X (0-8): 4

-------------
|   |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------

States explored: 55504
AI placed at pos 0

-------------
| O |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------

Enter move for X (0-8): 1

-------------
| O | X |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------

States explored: 1054
AI placed at pos 7

-------------
| O | X |   |
-------------
|   | X |   |
-------------
|   | O |   |
-------------
```

```
vicfic@NAG3:~/stud/ai/ex3$ python tic-tac-toe.py
<- Tic-Tac-Toe ->
Minimax (1) or AlphaBeta Pruning (2): 2

You are 'X' and the AI is 'O'.
Enter positions 0-8 to play.

-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------

Enter move for X (0-8): 4

-------------
|   |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------

States explored: 6138
AI placed at pos 0

-------------
| O |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------

Enter move for X (0-8): 1

-------------
| O | X |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------

States explored: 373
AI placed at pos 7

-------------
| O | X |   |
-------------
|   | X |   |
-------------
|   | O |   |
-------------
```

```
Enter move for X (0-8): 3

-------------
| O | X |   |
-------------
| X | X |   |
-------------
|   | O |   |
-------------

States explored: 50
AI placed at pos 5

-------------
| O | X |   |
-------------
| X | X | O |
-------------
|   | O |   |
-------------

Enter move for X (0-8): 2

-------------
| O | X | X |
-------------
| X | X | O |
-------------
|   | O |   |
-------------

States explored: 4
AI placed at pos 6

-------------
| O | X | X |
-------------
| X | X | O |
-------------
| O | O |   |
-------------

Enter move for X (0-8): 8

-------------
| O | X | X |
-------------
| X | X | O |
-------------
| O | O | X |
-------------

Game Over. It's a Draw!
vicfic@NAG3:~/stud/ai/ex3$
```

```
Enter move for X (0-8): 3

-------------
| O | X |   |
-------------
| X | X |   |
-------------
|   | O |   |
-------------

States explored: 38
AI placed at pos 5

-------------
| O | X |   |
-------------
| X | X | O |
-------------
|   | O |   |
-------------

Enter move for X (0-8): 2

-------------
| O | X | X |
-------------
| X | X | O |
-------------
|   | O |   |
-------------

States explored: 4
AI placed at pos 6

-------------
| O | X | X |
-------------
| X | X | O |
-------------
| O | O |   |
-------------

Enter move for X (0-8): 8

-------------
| O | X | X |
-------------
| X | X | O |
-------------
| O | O | X |
-------------

Game Over. It's a Draw!
vicfic@NAG3:~/stud/ai/ex3$
```

Date: 6 - 02 – 2026
Assignment No.: 3

Name: Varghese K James
Reg No.: 3122245001192

Learning Outcomes

- Implemented the Minimax and Alpha-Beta Pruning Search Methods.
- Learned to write a algorithmic opponent in an adversarial environment
- Understood the efficiancy that the Alpha Beta Pruning approach has over general Minimax in the amount of states explored.