ARTIFICIAL INTELLIGENCE (18CSC305J) LAB

EXPERIMENT 6

MIN MAX ALGORITHM

Harsh Goel
RA1811003010185
CSE-C1

Aim:

To implement min max algorithm and verify result/output.

Problem Description:

Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally. Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.

Problem Formulation (Tic Tac Toe) :

In strategic games, instead of letting the program start the searching process in the very beginning of the game, it is common to use the opening books - a list of known and productive moves that are frequent and known to be productive while we still don't have much information about the state of game itself if we look at the board.

In the beginning, it is too early in the game, and the number of potential positions is too great to automatically decide which move will certainly lead to a better game state (or win).

However, the algorithm reevaluates the next potential moves every turn, always choosing what at that moment appears to be the fastest route to victory. Therefore, it won't execute actions that take more than one move to complete, and is unable to perform certain well known "tricks" because of that. If the AI plays against a human, it is very likely that human will immediately be able to prevent this.

Psedu Code:

```
1. function minimax(node, depth, maximizingPlayer) is
2. if depth ==0 or node is a terminal node then
3. return static evaluation of node
4. if MaximizingPlayer then
```

```
5.  maxEva= -infinity
6.   for each child of node do
7.   eva= minimax(child, depth-1, false)
8.  maxEva= max(maxEva,eva)
9.  return maxEva
10. else
11. minEva= +infinity
12. for each child of node do
13. eva= minimax(child, depth-1, true)
14. minEva= min(minEva, eva
15. return minEva
```

Source Code:

Language- **Python**

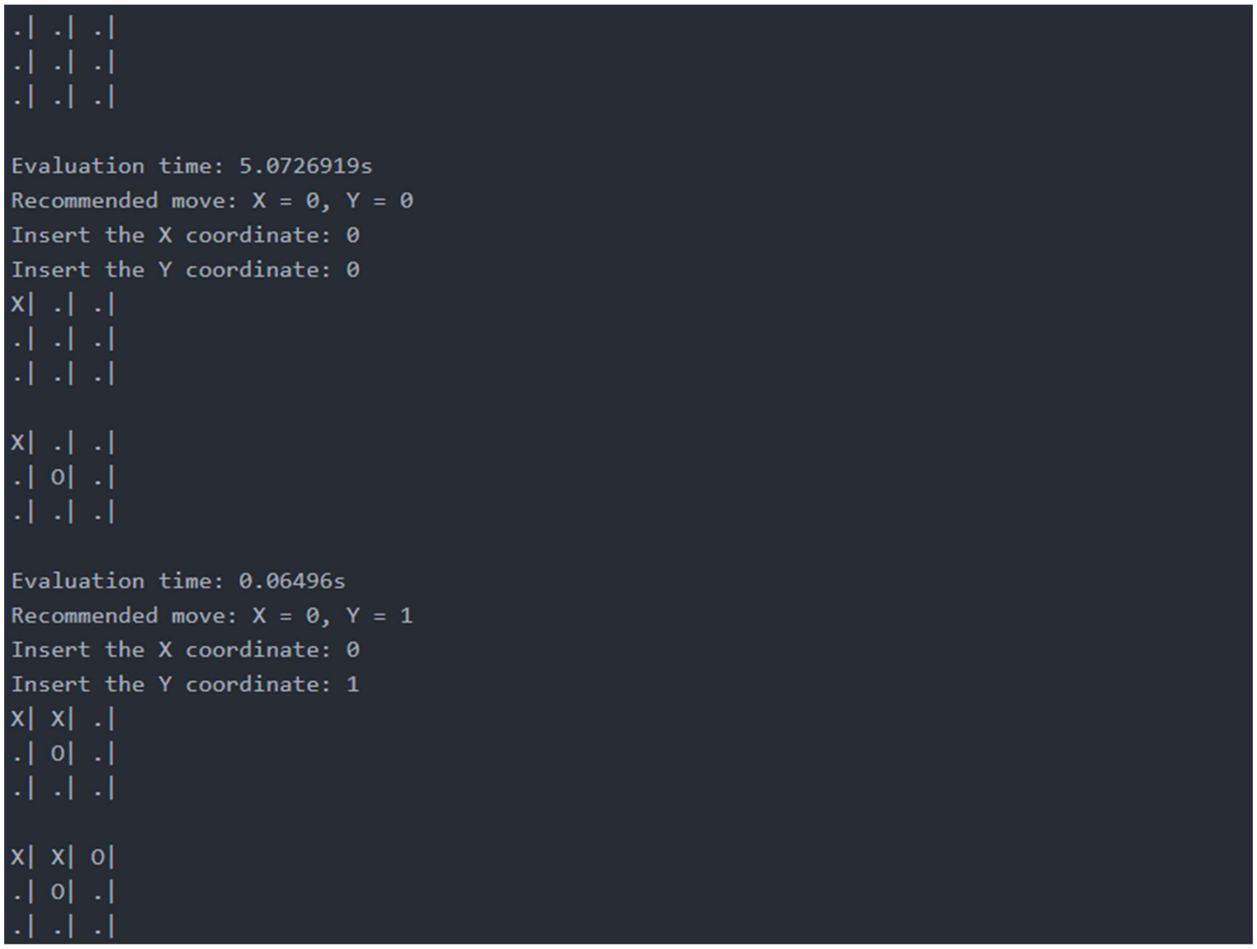
```python
class Game:
    def __init__(self):
        self.initialize_game()

    def initialize_game(self):
        self.current_state = [['.','.','.'],
                              ['.','.','.'],
                              ['.','.','.']]

        # Player X always plays first
        self.player_turn = 'X'

    def draw_board(self):
        for i in range(0, 3):
            for j in range(0, 3):
                print('{}|'.format(self.current_state[i][j]), end=" ")
            print()
        print()

    # Determines if the made move is a legal move
    def is_valid(self, px, py):
        if px < 0 or px > 2 or py < 0 or py > 2:
            return False
        elif self.current_state[px][py] != '.':
            return False
        else:
            return True
```

```python
# Checks if the game has ended and returns the winner in each case
def is_end(self):
    # Vertical win
    for i in range(0, 3):
        if (self.current_state[0][i] != '.' and
            self.current_state[0][i] == self.current_state[1][i] and
            self.current_state[1][i] == self.current_state[2][i]):
            return self.current_state[0][i]

    # Horizontal win
    for i in range(0, 3):
        if (self.current_state[i] == ['X', 'X', 'X']):
            return 'X'
        elif (self.current_state[i] == ['O', 'O', 'O']):
            return 'O'

    # Main diagonal win
    if (self.current_state[0][0] != '.' and
        self.current_state[0][0] == self.current_state[1][1] and
        self.current_state[0][0] == self.current_state[2][2]):
        return self.current_state[0][0]

    # Second diagonal win
    if (self.current_state[0][2] != '.' and
        self.current_state[0][2] == self.current_state[1][1] and
        self.current_state[0][2] == self.current_state[2][0]):
        return self.current_state[0][2]

    # Is whole board full?
    for i in range(0, 3):
        for j in range(0, 3):
            # There's an empty field, we continue the game
            if (self.current_state[i][j] == '.'):
                return None

    # It's a tie!
    return '.'

# Player 'O' is max, in this case AI
def max(self):

    # Possible values for maxv are:
    # -1 - loss
    # 0  - a tie
    # 1  - win

    # We're initially setting it to -2 as worse than the worst case:
        maxv = -2

        px = None
        py = None

        result = self.is_end()

        # If the game came to an end, the function needs to return
        # the evaluation function of the end. That can be:
        # -1 - loss
        # 0  - a tie
        # 1  - win
        if result == 'X':
            return (-1, 0, 0)
        elif result == 'O':
            return (1, 0, 0)
        elif result == '.':
            return (0, 0, 0)

        for i in range(0, 3):
            for j in range(0, 3):
                if self.current_state[i][j] == '.':
                    # On the empty field player 'O' makes a move and cal
ls Min
                    # That's one branch of the game tree.
                    self.current_state[i][j] = 'O'
                    (m, min_i, min_j) = self.min()
                    # Fixing the maxv value if needed
                    if m > maxv:
                        maxv = m
                        px = i
                        py = j
                    # Setting back the field to empty
                    self.current_state[i][j] = '.'
        return (maxv, px, py)


    # Player 'X' is min, in this case human
    def min(self):

        # Possible values for minv are:
        # -1 - win
        # 0  - a tie
        # 1  - loss

        # We're initially setting it to 2 as worse than the worst case:
        minv = 2
```

```python
        qx = None
        qy = None

        result = self.is_end()

        if result == 'X':
            return (-1, 0, 0)
        elif result == 'O':
            return (1, 0, 0)
        elif result == '.':
            return (0, 0, 0)

        for i in range(0, 3):
            for j in range(0, 3):
                if self.current_state[i][j] == '.':
                    self.current_state[i][j] = 'X'
                    (m, max_i, max_j) = self.max()
                    if m < minv:
                        minv = m
                        qx = i
                        qy = j
                    self.current_state[i][j] = '.'

        return (minv, qx, qy)


    def play(self):
        while True:
            self.draw_board()
            self.result = self.is_end()

            # Printing the appropriate message if the game has ended
            if self.result != None:
                if self.result == 'X':
                    print('The winner is X!')
                elif self.result == 'O':
                    print('The winner is O!')
                elif self.result == '.':
                    print("It's a tie!")

                self.initialize_game()
                return

            # If it's player's turn
            if self.player_turn == 'X':

                while True:
                    (m, qx, qy) = self.min()
                    print('Recommended move: X = {}, Y = {}'.format(qx,
qy))

                    px = int(input('Insert the X coordinate: '))
                    py = int(input('Insert the Y coordinate: '))

                    (qx, qy) = (px, py)

                    if self.is_valid(px, py):
                        self.current_state[px][py] = 'X'
                        self.player_turn = 'O'
                        break
                    else:
                        print('The move is not valid! Try again.')

            # If it's AI's turn
            else:
                (m, px, py) = self.max()
                self.current_state[px][py] = 'O'
                self.player_turn = 'X'

g = Game()
g.play()
```

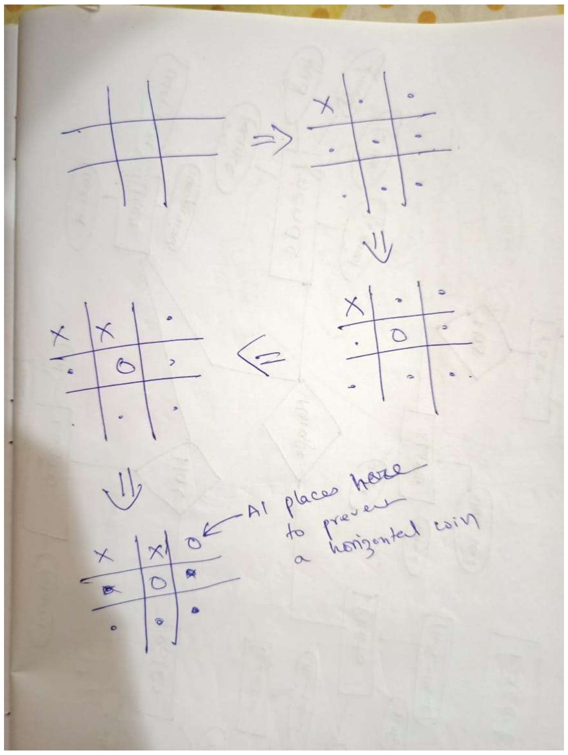
TEST CASE:

Input from User and Output from User



```
.| .| .|
.| .| .|
.| .| .|

Evaluation time: 5.0726919s
Recommended move: X = 0, Y = 0
Insert the X coordinate: 0
Insert the Y coordinate: 0
X| .| .|
.| .| .|
.| .| .|

X| .| .|
.| O| .|
.| .| .|

Evaluation time: 0.06496s
Recommended move: X = 0, Y = 1
Insert the X coordinate: 0
Insert the Y coordinate: 1
X| X| .|
.| O| .|
.| .| .|

X| X| O|
.| O| .|
.| .| .|
```

Verification:



Result: Hence, successfully implemented both A* and Best First Search algorithms and verified test cases.