Batch: A1(Honours) **Experiment Number:** 4

Roll Number: 16010421059 **Name:** Chinmay Mhatre

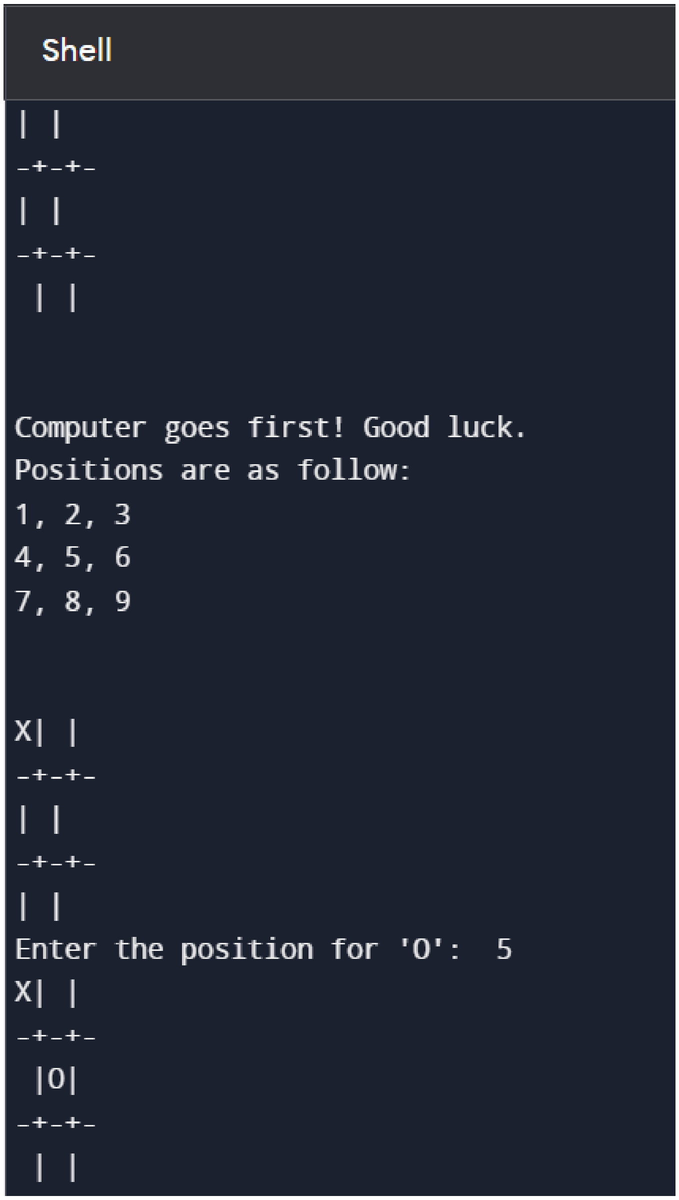
Aim of the Experiment: Implementation of Adversarial algorithm-Min-Max for Tic-Tac-Toe Game

Program/ Steps:

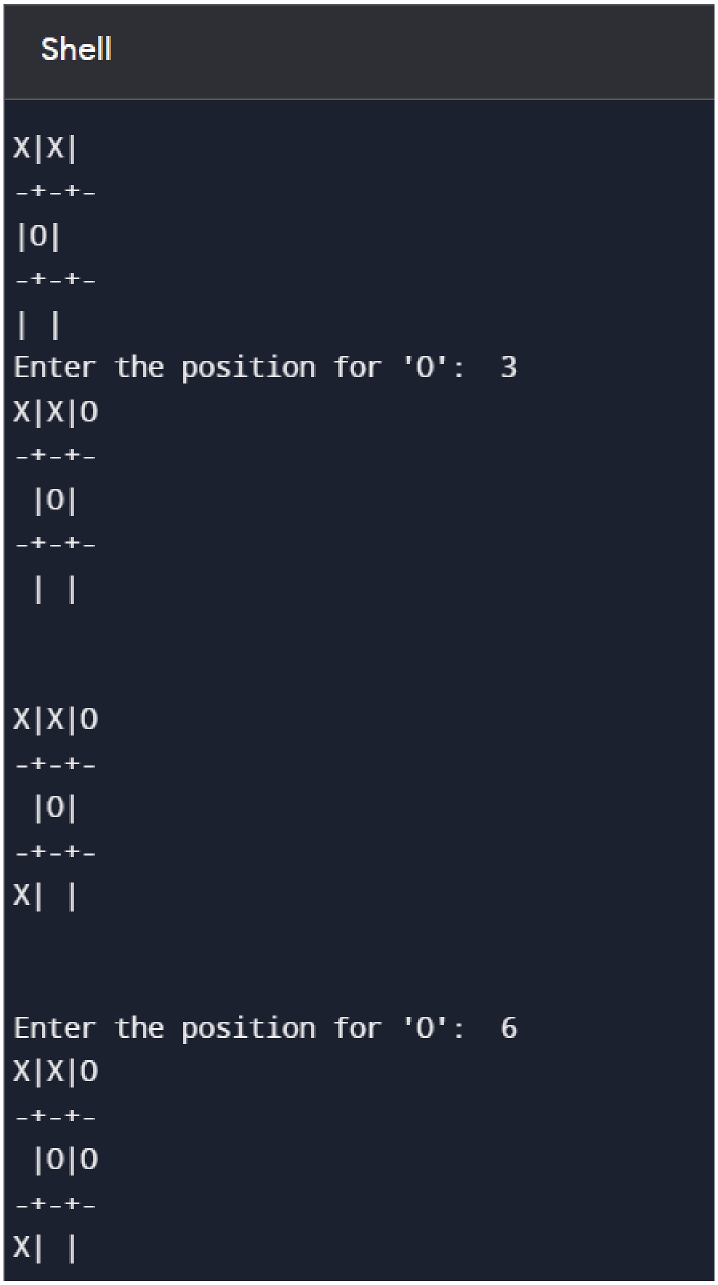
```
def printBoard(board):
    print(board[1] + '|' + board[2] + '|' + board[3])
    print('-+-+-')
    print(board[4] + '|' + board[5] + '|' + board[6])
    print('-+-+-')
    print(board[7] + '|' + board[8] + '|' + board[9])
    print("\n")



def spaceIsFree(position):
    if board[position] == ' ':
        return True
    else:
        return False



def insertLetter(letter, position):
    if spaceIsFree(position):
        board[position] = letter
        printBoard(board)
        if (checkDraw()):
            print("Draw!")
            exit()
        if checkForWin():
            if letter == 'X':
                print("Bot wins!")
                exit()
            else:
                print("Player wins!")
                exit()

        return
    else:
        print("Can't insert there!")
        position = int(input("Please enter new position: "))
        insertLetter(letter, position)
        return


def checkForWin():
    if (board[1] == board[2] and board[1] == board[3] and board[1] != ' '):
        return True
    elif (board[4] == board[5] and board[4] == board[6] and board[4] != ' '):
        return True
    elif (board[7] == board[8] and board[7] == board[9] and board[7] != ' '):
        return True
    elif (board[1] == board[4] and board[1] == board[7] and board[1] != ' '):
        return True
    elif (board[2] == board[5] and board[2] == board[8] and board[2] != ' '):
        return True
    elif (board[3] == board[6] and board[3] == board[9] and board[3] != ' '):
        return True
    elif (board[1] == board[5] and board[1] == board[9] and board[1] != ' '):
        return True
    elif (board[7] == board[5] and board[7] == board[3] and board[7] != ' '):
        return True
    else:
        return False


def checkWhichMarkWon(mark):
    if board[1] == board[2] and board[1] == board[3] and board[1] == mark:
        return True
    elif (board[4] == board[5] and board[4] == board[6] and board[4] == mark):
        return True
    elif (board[7] == board[8] and board[7] == board[9] and board[7] == mark):
        return True
    elif (board[1] == board[4] and board[1] == board[7] and board[1] == mark):
        return True
    elif (board[2] == board[5] and board[2] == board[8] and board[2] == mark):
        return True
    elif (board[3] == board[6] and board[3] == board[9] and board[3] == mark):
        return True
    elif (board[1] == board[5] and board[1] == board[9] and board[1] == mark):
        return True
    elif (board[7] == board[5] and board[7] == board[3] and board[7] == mark):
        return True
    else:
        return False


def checkDraw():
    for key in board.keys():
        if (board[key] == ' '):
            return False
    return True


def playerMove():
    position = int(input("Enter the position for 'O': "))
    insertLetter(player, position)
    return


def compMove():
    bestScore = -800
    bestMove = 0
    for key in board.keys():
        if (board[key] == ' '):
            board[key] = bot
            score = minimax(board, 0, False)
            board[key] = ' '
            if (score > bestScore):
                bestScore = score
                bestMove = key

    insertLetter(bot, bestMove)
    return


def minimax(board, depth, isMaximizing):
    if (checkWhichMarkWon(bot)):
        return 1
    elif (checkWhichMarkWon(player)):
        return -1
    elif (checkDraw()):
        return 0

    if (isMaximizing):
        bestScore = -800
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = bot
                score = minimax(board, depth + 1, False)
                board[key] = ' '
                if (score > bestScore):
                    bestScore = score
        return bestScore

    else:
        bestScore = 800
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = player
                score = minimax(board, depth + 1, True)
                board[key] = ' '
                if (score < bestScore):
                    bestScore = score
        return bestScore


board = {1: ' ', 2: ' ', 3: ' ',
        4: ' ', 5: ' ', 6: ' ',
        7: ' ', 8: ' ', 9: ' '}

printBoard(board)
print("Computer goes first! Good luck.")
print("Positions are as  follow:")
print("1, 2, 3 ")
print("4, 5, 6 ")
print("7, 8, 9 ")
print("\n")
player =  'O'
bot = 'X'


global firstComputerMove
firstComputerMove = True
```

```
while not checkForWin():
   compMove()
   playerMove()
```

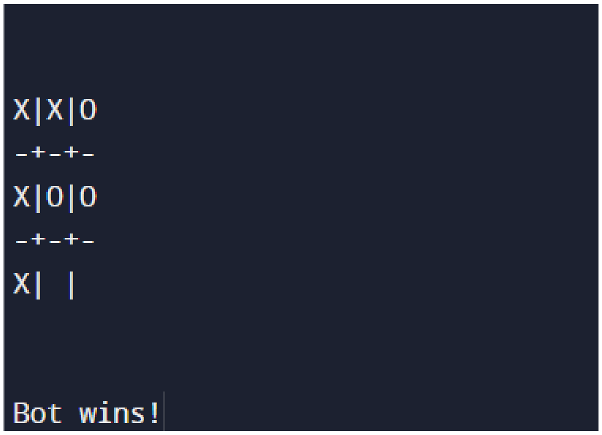
Output/Result:



```
| |
-+-+-
| |
-+-+-
 | |


Computer goes first! Good luck.
Positions are as follow:
1, 2, 3
4, 5, 6
7, 8, 9


X| |
-+-+-
| |
-+-+-
| |
Enter the position for 'O':  5
X| |
-+-+-
 |O|
-+-+-
 | |
```



```
X|X|
-+-+-
|O|
-+-+-
| |
Enter the position for 'O':  3
X|X|O
-+-+-
 |O|
-+-+-
 | |


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


Enter the position for 'O':  6
X|X|O
-+-+-
 |O|O
-+-+-
X| |
```



```


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


Bot wins!
```

Post Lab Question-Answers:

1. **Game playing is often called as an**

   a) Non-adversarial search
   b) Adversarial search
   c) Sequential search
   d) None of the above

   Ans: b) Adversarial 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: b) Operators defining legal moves

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):

We successfully implemented Adversarial algorithm-Min-Max for Tic-Tac-Toe Game.

References:

1. 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 9d690bad4b37/#:~:text=A%20Minimax%20algorithm%20can%20be,on%20each%20available%20spot%20(recursion)
2. Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, 2ndEdition, Pearson Publication
3. Elaine Rich, Kevin Knight, Artificial Intelligence, Tata McGraw Hill, 1999.