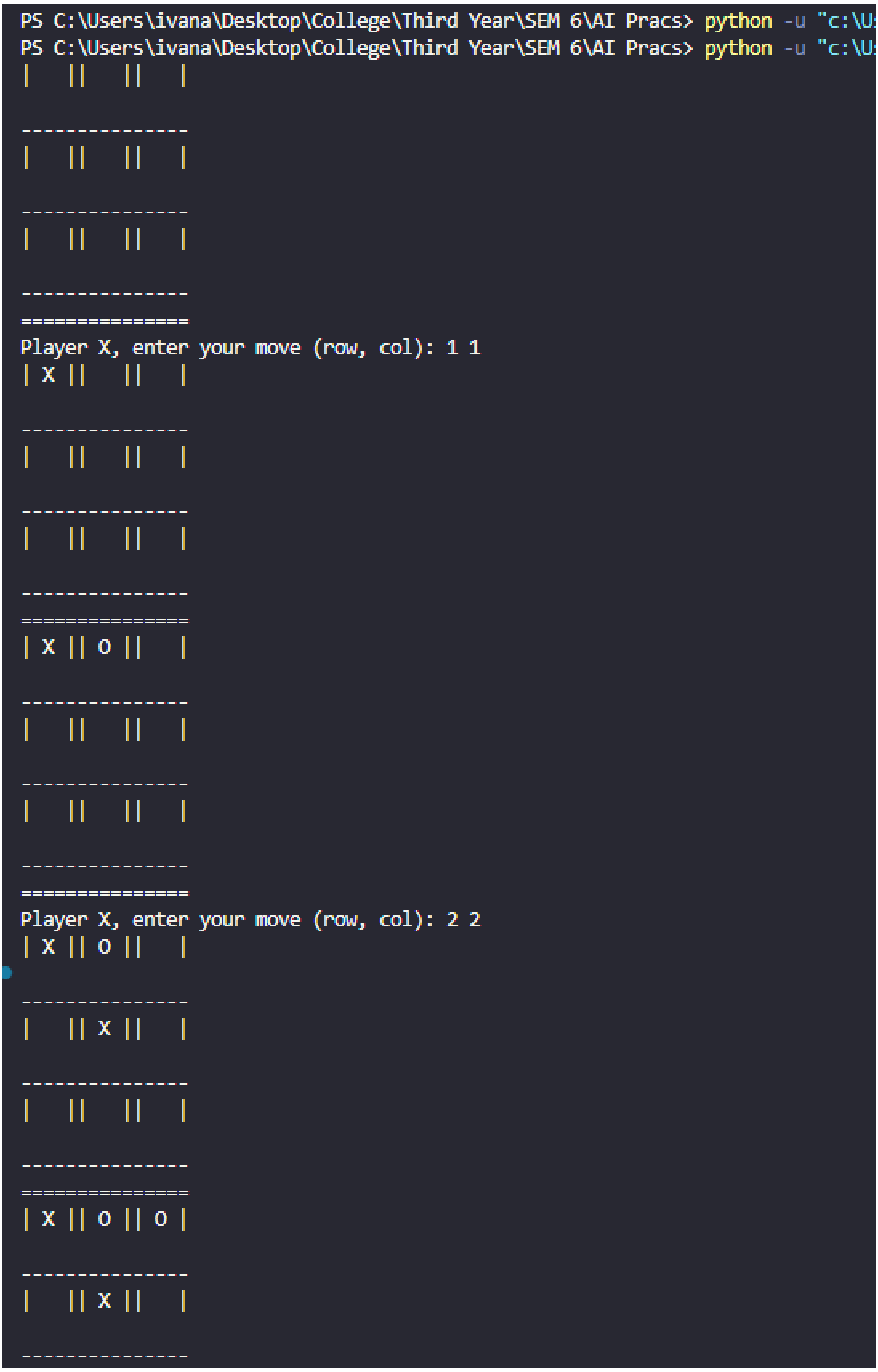
Experiment 1

- ***Ivan Dsouza - 9601-***
- *T.E. COMPS A(Batch C)*

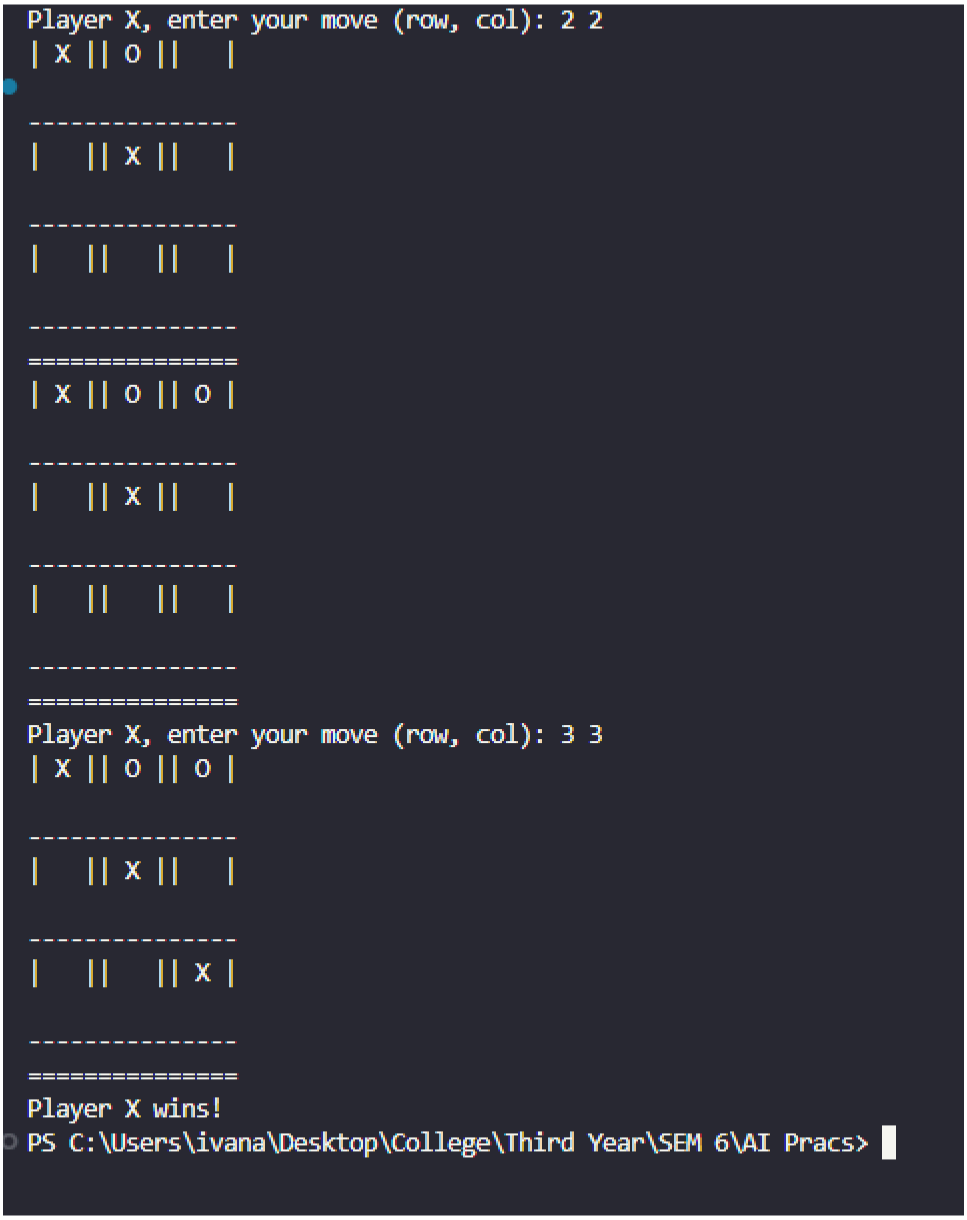
A) Using Brute Force method

```
from random import choice
from math import inf


def create_board():
    return [[" " for _ in range(3)] for _ in range(3)]


def print_board(board):
    for row in board:
        print("| " + " || ".join(row) + " | ")
        print("\n" + "---------------")
    print("===============")


def check_win(board, player):
    win_conditions = [
        [board[0][0], board[0][1], board[0][2]],
        [board[1][0], board[1][1], board[1][2]],
        [board[2][0], board[2][1], board[2][2]],
        [board[0][0], board[1][0], board[2][0]],
        [board[0][1], board[1][1], board[2][1]],
        [board[0][2], board[1][2], board[2][2]],
        [board[0][0], board[1][1], board[2][2]],
        [board[0][2], board[1][1], board[2][0]],
    ]
    return any(
        all(cell == player for cell in condition) for condition in win_conditions
    )


def check_tie(board):
    return all(cell != " " for row in board for cell in row)


def get_available_moves(board):
    return [
        (row, col) for row in range(3) for col in range(3) if board[row][col] == " "
    ]
```

```
def get_random_move(board):
    available_moves = get_available_moves(board)
    return random.choice(available_moves) if available_moves else None


def make_move(board, player, move):
    row, col = move
    board[row][col] = player


def play_game():
    board = create_board()
    current_player = "X"

    while True:
        print_board(board)

        if current_player == "O":
            best_move = None
            best_score = float("-inf")
            for move in get_available_moves(board):
                temp_board = [row[:] for row in board]
                make_move(temp_board, "O", move)
                score = minimax(temp_board, "X")
                if score > best_score:
                    best_score = score
                    best_move = move
            make_move(board, "O", best_move)
        else:
            try:
                row, col = map(
                    int,
                    input(
                        f"Player {current_player}, enter your move (row, col): "
                    ).split(),
                )
                move = (row - 1, col - 1)
                if move not in get_available_moves(board):
                    print(
                        "Invalid move. The cell is already taken or out of bounds. Try again."
                    )
                    continue
                make_move(board, current_player, move)
            except ValueError:
                print(
                    "Invalid move. Please enter row and column as numbers separated by a space."
                )
                continue
        if check_win(board, current_player):
            print_board(board)
            print(f"Player {current_player} wins!")
            break
        elif check_tie(board):
            print_board(board)
            print("It's a tie!")
            break

        current_player = "O" if current_player == "X" else "X"


def minimax(board, player):
    if check_win(board, "X"):
        return 1
    elif check_win(board, "O"):
        return -1
    elif check_tie(board):
        return 0

    scores = []
    for move in get_available_moves(board):
        temp_board = [row[:] for row in board]
        make_move(temp_board, player, move)
        if player == "X":
            score = minimax(temp_board, "O")
        else:
            score = minimax(temp_board, "X")
        scores.append(score)

    return max(scores) if player == "X" else min(scores)


if __name__ == "__main__":
    play_game()
```



```
PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\AI Pracs> python -u "c:\U
PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\AI Pracs> python -u "c:\U
|   ||   ||   |

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

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

---------------
===============
Player X, enter your move (row, col): 1 1
| X ||   ||   |

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

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

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

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

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

---------------
===============
Player X, enter your move (row, col): 2 2
| X || O ||   |

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

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

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

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

---------------
```

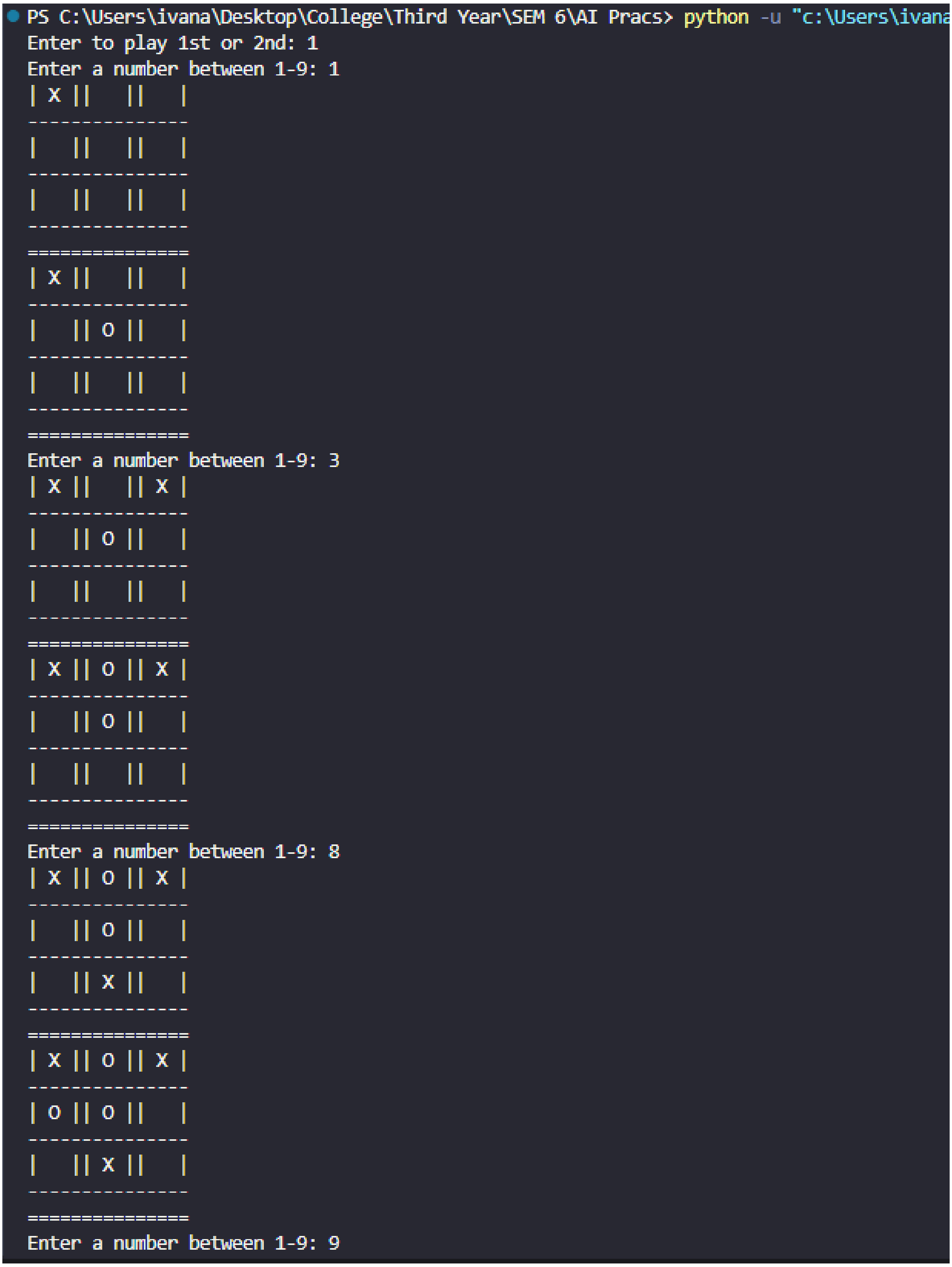


```
Player X, enter your move (row, col): 2 2
| X || O ||   |

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

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

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

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

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

---------------
===============
Player X, enter your move (row, col): 3 3
| X || O || O |

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

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

---------------
===============
Player X wins!
PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\AI Pracs>
```

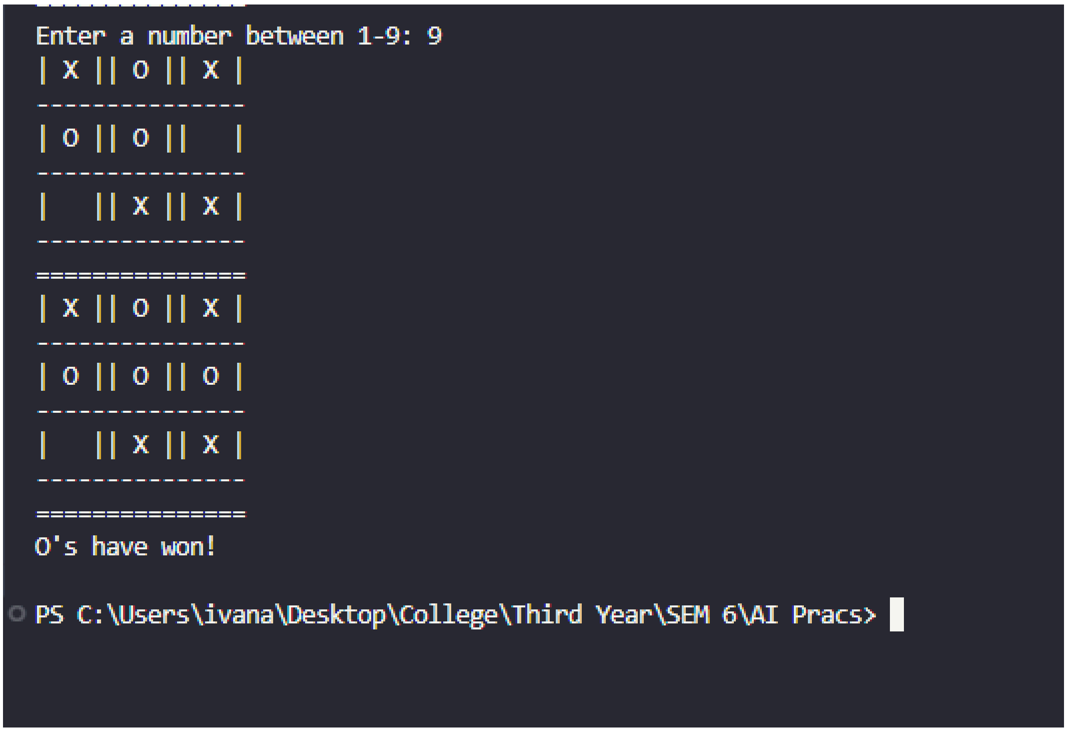
B) Using Heuristic Approach

```
from random import choice
from math import inf

board = [[0, 0, 0],
         [0, 0, 0],
         [0, 0, 0]]

def print_board(board):
    chars = {1: 'X', -1: 'O', 0: ' '}
    for x in board:
        for y in x:
            ch = chars[y]
            print(f'| {ch} |', end='')
        print('\n' + '---------------')
    print('===============')

def Clearboard(board):
    for x, row in enumerate(board):
        for y, col in enumerate(row):
            board[x][y] = 0

def check_win(board, player):
    win_conditions = [[board[0][0], board[0][1], board[0][2]],
                      [board[1][0], board[1][1], board[1][2]],
                      [board[2][0], board[2][1], board[2][2]],
                      [board[0][0], board[1][0], board[2][0]],
                      [board[0][1], board[1][1], board[2][1]],
                      [board[0][2], board[1][2], board[2][2]],
                      [board[0][0], board[1][1], board[2][2]],
                      [board[0][2], board[1][1], board[2][0]]]

    if [player, player, player] in win_conditions:
        return True

    return False

def gameWon(board):
    return check_win(board, 1) or check_win(board, -1)

def printResult(board):
    if check_win(board, 1):
        print('X has won! ' + '\n')

    elif check_win(board, -1):
        print('O\'s have won! ' + '\n')
    else:
        print('Draw' + '\n')

def blanks(board):
    blank = []
    for x, row in enumerate(board):
        for y, col in enumerate(row):
            if board[x][y] == 0:
                blank.append([x, y])

    return blank

def boardFull(board):
    if len(blanks(board)) == 0:
        return True
    return False

def setMove(board, x, y, player):
    board[x][y] = player

def playerMove(board):
    e = True
    moves = {1: [0, 0], 2: [0, 1], 3: [0, 2],
             4: [1, 0], 5: [1, 1], 6: [1, 2],
             7: [2, 0], 8: [2, 1], 9: [2, 2]}
    while e:
        try:
            move = int(input('Enter a number between 1-9: '))
            if move < 1 or move > 9:
                print('Invalid Move! Try again!')
            elif not (moves[move] in blanks(board)):
                print('Invalid Move! Try again!')
            else:
                setMove(board, moves[move][0], moves[move][1], 1)
                print_board(board)
                e = False
        except(KeyError, ValueError):
            print('Enter a number!')

def getScore(board):
    if check_win(board, 1):
        return 10

    elif check_win(board, -1):
        return -10

    else:
        return 0
```

```
def minmax(board, depth, alpha, beta, player):
    row = -1
    col = -1
    if depth == 0 or gameWon(board):
        return [row, col, getScore(board)]

    else:
        for cell in blanks(board):
            setMove(board, cell[0], cell[1], player)
            score = minmax(board, depth - 1, alpha, beta, -player)
            if player == 1:
                # X is always the max player
                if score[2] > alpha:
                    alpha = score[2]
                    row = cell[0]
                    col = cell[1]

            else:
                if score[2] < beta:
                    beta = score[2]
                    row = cell[0]
                    col = cell[1]

            setMove(board, cell[0], cell[1], 0)

            if alpha >= beta:
                break

        if player == 1:
            return [row, col, alpha]

        else:
            return [row, col, beta]

def o_comp(board):
    if len(blanks(board)) == 9:
        x = choice([0, 1, 2])
        y = choice([0, 1, 2])
        setMove(board, x, y, -1)
        print_board(board)

    else:
        result = minmax(board, len(blanks(board)), -inf, inf, -1)
        setMove(board, result[0], result[1], -1)
        print_board(board)

def x_comp(board):
    if len(blanks(board)) == 9:
        x = choice([0, 1, 2])
        y = choice([0, 1, 2])
        setMove(board, x, y, 1)
        print_board(board)

    else:
        result = minmax(board, len(blanks(board)), -inf, inf, 1)
        setMove(board, result[0], result[1], 1)
        print_board(board)

def makeMove(board, player, mode):
    if mode == 1:
        if player == 1:
            playerMove(board)

        else:
            o_comp(board)
    else:
        if player == 1:
            o_comp(board)
        else:
            x_comp(board)

def play():
    while True:
        try:
            order = int(input('Enter to play 1st or 2nd: '))
            if not (order == 1 or order == 2):
                print('Please pick 1 or 2')
            else:
                break
        except(KeyError, ValueError):
            print('Enter a number')

    Clearboard(board)
    if order == 2:
        currentPlayer = -1
    else:
        currentPlayer = 1

    while not (boardFull(board) or gameWon(board)):
        makeMove(board, currentPlayer, 1)
        currentPlayer *= -1

    printResult(board)

play()
```



```
PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\AI Pracs> python -u "c:\Users\ivana
Enter to play 1st or 2nd: 1
Enter a number between 1-9: 1
| X ||   ||   |
---------------
|   ||   ||   |
---------------
|   ||   ||   |
---------------
===============
| X ||   ||   |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
===============
Enter a number between 1-9: 3
| X ||   || X |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
===============
| X || O || X |
---------------
|   || O ||   |
---------------
|   ||   ||   |
---------------
===============
Enter a number between 1-9: 8
| X || O || X |
---------------
|   || O ||   |
---------------
|   || X ||   |
---------------
===============
| X || O || X |
---------------
| O || O ||   |
---------------
|   || X ||   |
---------------
===============
Enter a number between 1-9: 9
```



```
---------------
Enter a number between 1-9: 9
| X || O || X |
---------------
| O || O ||   |
---------------
|   || X || X |
---------------
===============
| X || O || X |
---------------
| O || O || O |
---------------
|   || X || X |
---------------
===============
O's have won!

PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\AI Pracs>
```

Post Lab Assignment Answers:

Q) The easiest trick to win Tic Tac Toe:

1. Focus on creating winning opportunities for yourself while blocking your opponent's potential wins.
2. Take the center square if available, as it gives you control over more lines.
3. Aim to control two lines at once, making it easier to create a win.
4. Block your opponent's potential win by placing your mark before them.
5. Pay attention to corners and edges, as they offer more options for creating lines.
6. Algorithm to win a 5*5 Tic Tac Toe:

Q) The basic principles of winning remain the same as in 3*3 Tic Tac Toe.

1. Focus on controlling more lines and blocking your opponent's options.
2. Utilize the additional squares to create more potential winning combinations.
3. Consider advanced strategies like "forking" (creating two potential wins at once) and "blocking chains" (preventing your opponent from creating multiple win options).
4. Algorithms like minimax with pruning can be applied to explore future moves and identify the best path to victory.

Q) Is there a way to never lose at Tic-Tac-Toe?

1. Yes, playing perfectly against another perfect player will always result in a draw.
2. This means always making the optimal move and blocking all potential opponent wins.
3. The complexity of achieving perfect play increases with larger boards.

Q) What can Tic-Tac-Toe help you with?

1. Tic-Tac-Toe can be a simple yet effective tool for:
2. Developing critical thinking skills: It requires planning, strategizing, and anticipating your opponent's moves.
3. Learning basic concepts of game theory: Players can understand concepts like win conditions, optimal moves, and decision-making under uncertainty.
4. Improving problem-solving abilities: Finding creative solutions to create winning scenarios and block opponents' moves.
5. Enhancing spatial reasoning skills: Visualizing the board layout and potential lines of play.
6. Promoting sportsmanship and fair play: The game teaches respecting rules, taking turns, and accepting defeat gracefully.