```python
def print_board(board):
    for row in board:
        print(" ".join(row))
    print()

def check_winner(board, player):
    for i in range(3):
        if all(board[i][j] == player for j in range(3)):
            return True
        if all(board[j][i] == player for j in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)):
        return True
    if all(board[i][2 - i] == player for i in range(3)):
        return True
    return False

def is_draw(board):
    return all(board[i][j] != '-' for i in range(3) for j in range(3))

def minimax(board, is_ai_turn):
    if check_winner(board, 'O'): # AI win
        return 1
    if check_winner(board, 'X'): # Player win
        return -1
    if is_draw(board):
        return 0

    if is_ai_turn:
        best_score = -float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == '-':
                    board[i][j] = 'O'
                    score = minimax(board, False)
                    board[i][j] = '-'
                    best_score = max(score, best_score)
        return best_score
    else:
        best_score = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == '-':
                    board[i][j] = 'X'
                    score = minimax(board, True)
                    board[i][j] = '-'
                    best_score = min(score, best_score)
        return best_score
```

```python
def manual_game():
    board = [['-' for _ in range(3)] for _ in range(3)]
    print("Initial Board:")
    print_board(board)

    while True:
        # Input X move
        while True:
            try:
                x_row = int(input("Enter X row (1-3): ")) - 1
                x_col = int(input("Enter X col (1-3): ")) - 1
                if board[x_row][x_col] == '-':
                    board[x_row][x_col] = 'X'
                    break
                else:
                    print("Cell occupied!")
            except:
                print("Invalid input!")

        print("Board after X move:")
        print_board(board)

        if check_winner(board, 'X'):
            print("X wins!")
            break
        if is_draw(board):
            print("Draw!")
            break

        # Input O move
        while True:
            try:
                o_row = int(input("Enter O row (1-3): ")) - 1
                o_col = int(input("Enter O col (1-3): ")) - 1
                if board[o_row][o_col] == '-':
                    board[o_row][o_col] = 'O'
                    break
                else:
                    print("Cell occupied!")
            except:
                print("Invalid input!")

        print("Board after O move:")
        print_board(board)

        if check_winner(board, 'O'):
            print("O wins!")
```

```
            break
        if is_draw(board):
            print("Draw!")
            break

        # AI evaluates the board (from current position)
        cost = minimax(board, True) # AI's turn to move next
        print(f"AI evaluation cost from this position: {cost}")

manual_game()
```

TicTacToe 1BM23CS316.ipynb

File Edit View Insert Runtime Tools Help

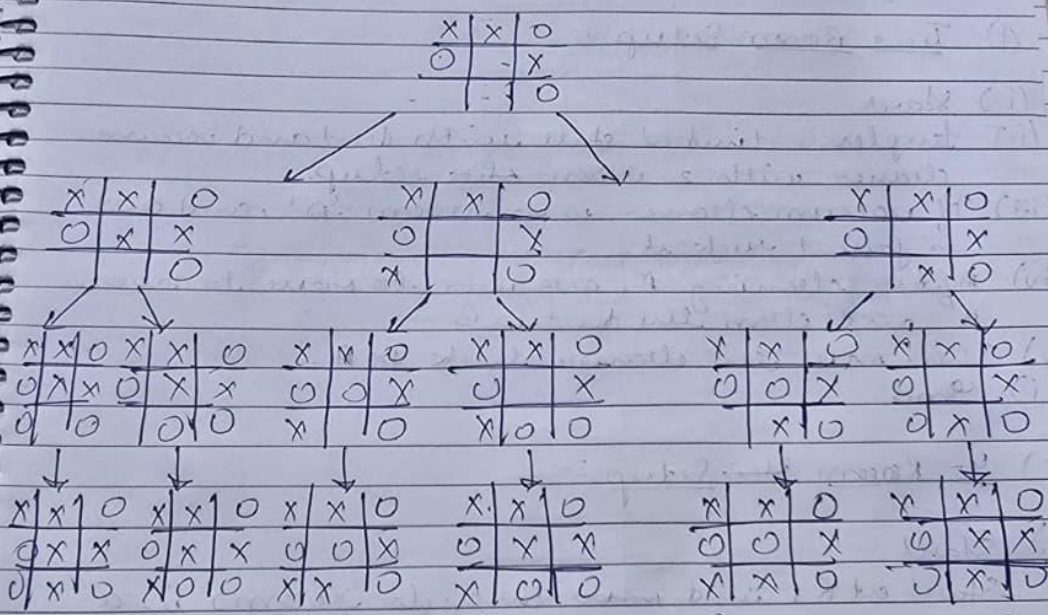
Commands + Code + Text Run all

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

AI evaluation cost from this position: 0
Enter X row (1-3): 2
Enter X col (1-3): 3
Board after X move:
O X O
- X X
- O X

Enter O row (1-3): 2
Enter O col (1-3): 1
Board after O move:
O X O
O X X
- O X

AI evaluation cost from this position: 1
Enter X row (1-3): 3
Enter X col (1-3): 1
Board after X move:
O X O
O X X
X O X

Draw!
```

Page No.:
Date: 18/8/25

LAB-1: Implement Tic Toe Toe

Initial State (X Starts)



Algorithm:

1.) Start
2.) Ask the player to choose between 'X' and 'O'
3.) Decide who goes first
4.) Repeat until game ends:
   - If player's turn:
     - Show board and get player's move
     - update board
     - If player wins announce.
     - If tie, announce and ask the player to play again.
5.) End