Lab Assignment 5

Subject: Artificial Intelligence **Guided by:** Dr. Anuradha Yenkikar

Student name: Neha Sawant

Experiment Name: Implement minmax algorithm for game playing.

Objective:

To implement the Minimax algorithm to enable an AI to play Tic-Tac-Toe optimally. The goal is for the AI (Player X) to play against the human (Player O) and always find the best move.

Theory:

The Minimax Algorithm is a decision-making algorithm, especially used in two-player games like Tic-Tac-Toe. The algorithm simulates all possible moves of the game, and it chooses the move that maximizes the AI's chance of winning while minimizing the chance of the opponent's winning. It ensures that AI plays optimally and never loses (at worst, it ties).

Key Concepts:

- Maximizing Player (AI): Tries to maximize the score (Player X).
- Minimizing Player (Human) Tries to minimize the score (Player O).
- The AI evaluates every possible move using recursion and backtracking until it finds the best possible move.

Algorithm Implementation Steps:

1.Check for a Winner: Evaluate if there’s a winner by checking rows, columns, or diagonals.

2. Check for a Draw: Verify if the board is completely filled, resulting in a draw.

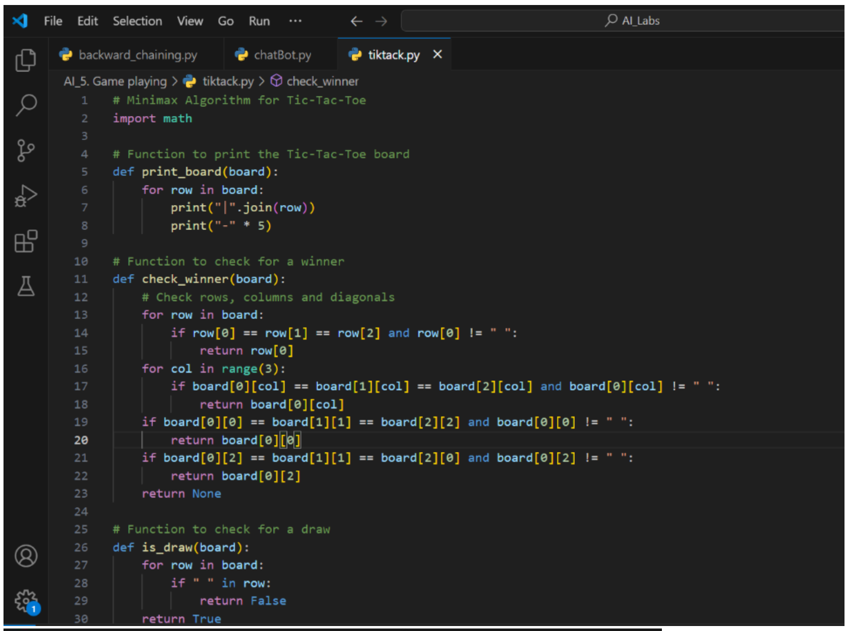
3. Minimax Function:
   - It recursively evaluates all potential moves.
   - Assigns scores: `+1` if AI wins, `-1` if the human wins, `0` for a draw.
   - AI maximizes its score, while the human minimizes it.

4. Best Move Finder: The AI selects the best move by evaluating the result of the Minimax function.

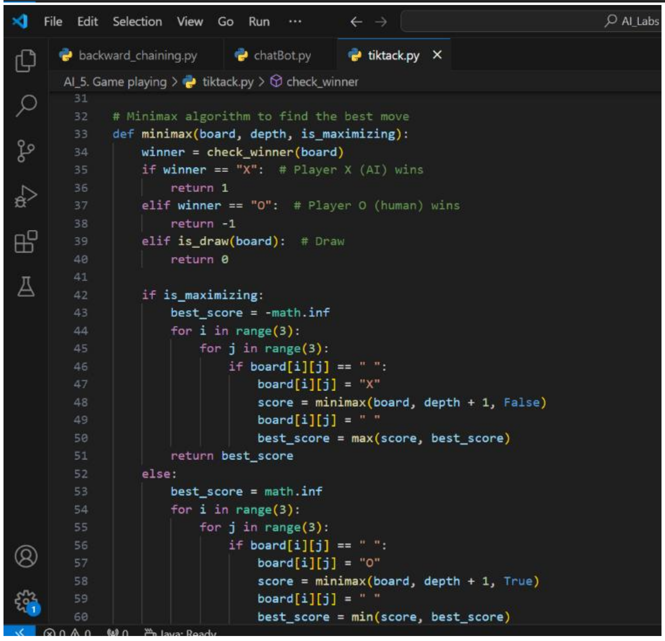
5.Game Play:

- The human plays first (Player O).
- The AI (Player X) responds using the Minimax algorithm.

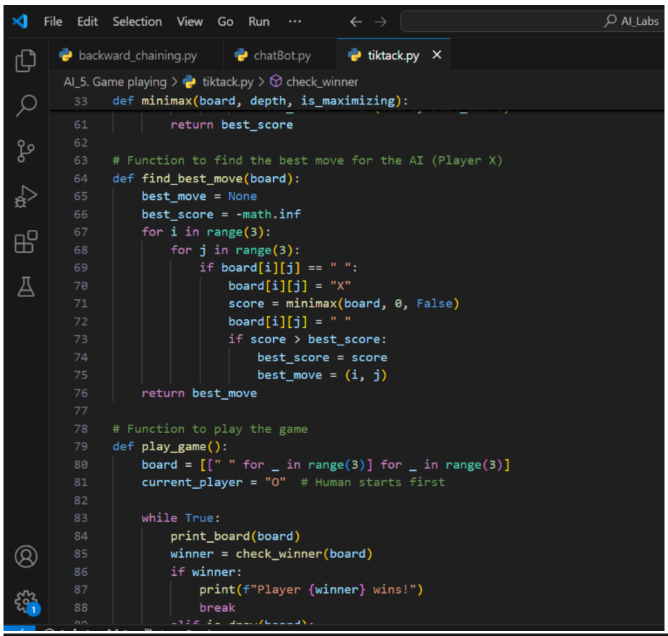
Code Explanation:


File Edit Selection View Go Run ···
AI_Labs
backward_chaining.py
chatBot.py
tiktack.py
AI_5. Game playing > tiktack.py > check_winner

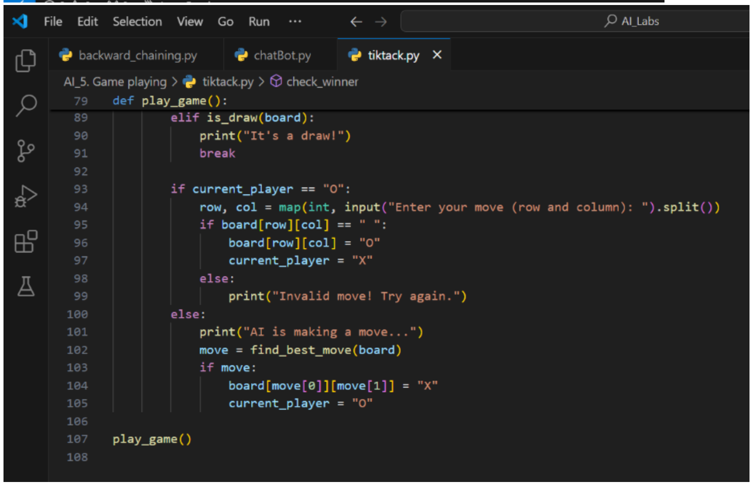
```python
# Minimax Algorithm for Tic-Tac-Toe
import math

# Function to print the Tic-Tac-Toe board
def print_board(board):
    for row in board:
        print("|".join(row))
        print("-" * 5)

# Function to check for a winner
def check_winner(board):
    # Check rows, columns and diagonals
    for row in board:
        if row[0] == row[1] == row[2] and row[0] != " ":
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != " ":
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != " ":
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != " ":
        return board[0][2]
    return None

# Function to check for a draw
def is_draw(board):
    for row in board:
        if " " in row:
            return False
    return True
```


File Edit Selection View Go Run ···
AI_Labs
backward_chaining.py
chatBot.py
tiktack.py
AI_5. Game playing > tiktack.py > check_winner
Java: Ready

```python

# Minimax algorithm to find the best move
def minimax(board, depth, is_maximizing):
    winner = check_winner(board)
    if winner == "X":  # Player X (AI) wins
        return 1
    elif winner == "O":  # Player O (human) wins
        return -1
    elif is_draw(board):  # Draw
        return 0

    if is_maximizing:
        best_score = -math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "X"
                    score = minimax(board, depth + 1, False)
                    board[i][j] = " "
                    best_score = max(score, best_score)
        return best_score
    else:
        best_score = math.inf
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "O"
                    score = minimax(board, depth + 1, True)
                    board[i][j] = " "
                    best_score = min(score, best_score)
```


File Edit Selection View Go Run ···
AI_Labs
backward_chaining.py
chatBot.py
tiktack.py
AI_5. Game playing > tiktack.py > check_winner

```python
def minimax(board, depth, is_maximizing):
        return best_score

# Function to find the best move for the AI (Player X)
def find_best_move(board):
    best_move = None
    best_score = -math.inf
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = "X"
                score = minimax(board, 0, False)
                board[i][j] = " "
                if score > best_score:
                    best_score = score
                    best_move = (i, j)
    return best_move

# Function to play the game
def play_game():
    board = [[" " for _ in range(3)] for _ in range(3)]
    current_player = "O"  # Human starts first

    while True:
        print_board(board)
        winner = check_winner(board)
        if winner:
            print(f"Player {winner} wins!")
            break
```


File Edit Selection View Go Run ···
AI_Labs
backward_chaining.py
chatBot.py
tiktack.py
AI_5. Game playing > tiktack.py > check_winner

```python
def play_game():
        elif is_draw(board):
            print("It's a draw!")
            break

        if current_player == "O":
            row, col = map(int, input("Enter your move (row and column): ").split())
            if board[row][col] == " ":
                board[row][col] = "O"
                current_player = "X"
            else:
                print("Invalid move! Try again.")
        else:
            print("AI is making a move...")
            move = find_best_move(board)
            if move:
                board[move[0]][move[1]] = "X"
                current_player = "O"

play_game()
```

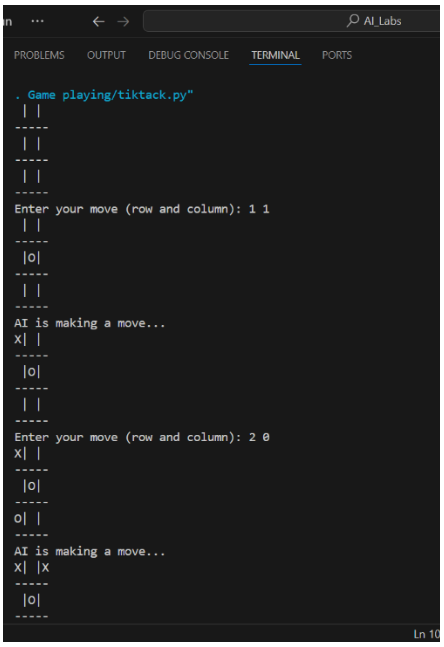
Instructions to Run:

1. Install Python (if not already installed).

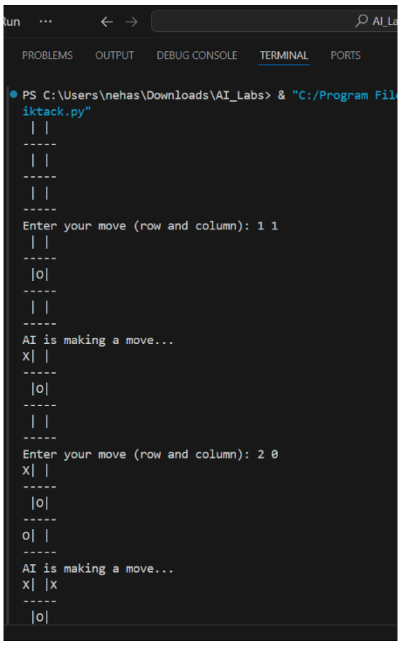
2. Save the code into a file, e.g., `tictactoe.py`.

3. Open a terminal or command prompt, navigate to the folder where your file is saved, and run:

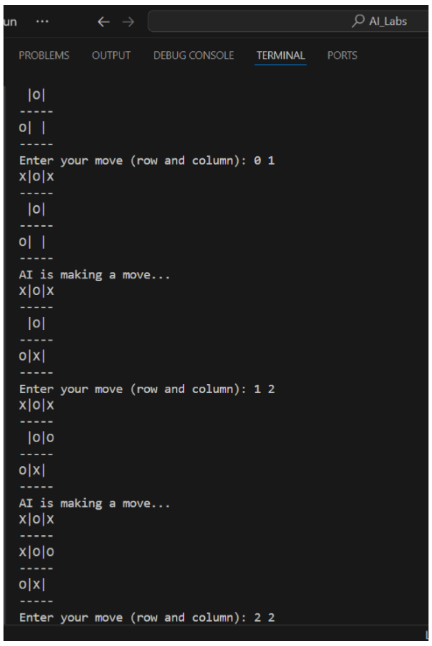
4. The game will prompt you to input your move as two integers (row and column) between `0` and `2`. Example:



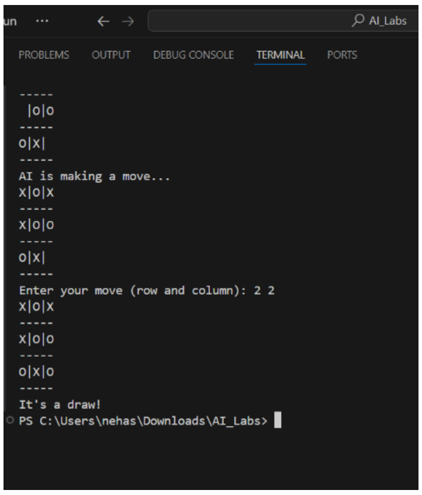
```
. Game playing/tiktack.py"
 | |
-----
 | |
-----
 | |
-----
Enter your move (row and column): 1 1
 | |
-----
 |o|
-----
 | |
-----
AI is making a move...
x| |
-----
 |o|
-----
 | |
-----
Enter your move (row and column): 2 0
x| |
-----
 |o|
-----
o| |
-----
AI is making a move...
x| |x
-----
 |o|
-----
```



```
PS C:\Users\nehas\Downloads\AI_Labs> & "C:/Program Fil
iktack.py"
 | |
-----
 | |
-----
 | |
-----
Enter your move (row and column): 1 1
 | |
-----
 |o|
-----
 | |
-----
AI is making a move...
x| |
-----
 |o|
-----
 | |
-----
Enter your move (row and column): 2 0
x| |
-----
 |o|
-----
o| |
-----
AI is making a move...
x| |x
-----
 |o|
```



```
 |o|
-----
o| |
-----
Enter your move (row and column): 0 1
x|o|x
-----
 |o|
-----
o| |
-----
AI is making a move...
x|o|x
-----
 |o|
-----
o|x|
-----
Enter your move (row and column): 1 2
x|o|x
-----
 |o|o
-----
o|x|
-----
AI is making a move...
x|o|x
-----
x|o|o
-----
o|x|
-----
Enter your move (row and column): 2 2
```



```
-----
 |o|o
-----
o|x|
-----
AI is making a move...
x|o|x
-----
x|o|o
-----
o|x|
-----
Enter your move (row and column): 2 2
x|o|x
-----
x|o|o
-----
o|x|o
-----
It's a draw!
PS C:\Users\nehas\Downloads\AI_Labs>
```

Conclusion:

The Minimax algorithm implemented in this lab ensures the AI never loses in Tic-Tac-Toe. It evaluates all possible moves and chooses the one that maximizes its chances of winning, resulting in optimal gameplay.