**🎯 Assignment: Understanding the Tic-Tac-Toe Problem in AI**

**🧠 What is the Tic-Tac-Toe Problem?**

Tic-Tac-Toe is a simple two-player game played on a 3x3 grid. Players take turns placing their symbol (usually **X** and **O**) in empty squares. The first player to get **three of their symbols in a row** — horizontally, vertically, or diagonally — wins. If all 9 squares are filled without any player winning, the game is a **draw**.

In Artificial Intelligence, **Tic-Tac-Toe** is often used to teach:

* Game playing agents
* Search trees
* Minimax algorithm
* Utility-based decision making

**🧩 Components of a Tic-Tac-Toe Search Problem**

* **Initial State**: Empty 3x3 board.
* **Actions**: Place X or O in any empty cell.
* **Goal Test**: Check if the player has won (3 in a row) or if the board is full (draw).
* **Path Cost**: Usually not considered here, but each move could be counted as one step.
* **Type**: Goal-based problem, deterministic, fully observable.

**📘 Tasks**

**Task 1: Representing the Game**

1. Draw an initial empty 3x3 board.
2. List all possible first moves for Player X.
3. How many possible unique board states are there after the first move?

Initial Empty 3x3 Board:

1 | 2 | 3

---------

4 | 5 | 6

---------

7 | 8 | 9

Possible First Moves for Player X:

The possible first moves for Player X are the 9 positions on the board:

1, 2, 3, 4, 5, 6, 7, 8, 9

Possible Unique Board States after the First Move:

There are 9 possible unique board states after the first move, one for each position where Player X can place their mark (X).

For example:

- If X places their mark in position 1: X | 2 | 3 / 4 | 5 | 6 / 7 | 8 | 9

- If X places their mark in position 5: 1 | 2 | 3 / 4 | X | 6 / 7 | 8 | 9

Each position corresponds to a unique board state.

**Task 2: Game Tree Exploration**

1. Create a small **search tree** of Tic-Tac-Toe up to depth 2:
   * Root node: initial board
   * Depth 1: All possible moves by X
   * Depth 2: All possible responses by O
2. Label each node with the board state.

Game Tree Exploration

Root Node (Initial Board):

1 | 2 | 3

---------

4 | 5 | 6

---------

7 | 8 | 9

Depth 1: All Possible Moves by X

Let's consider the possible moves by X:

1. X in position 1: X | 2 | 3 / 4 | 5 | 6 / 7 | 8 | 9

2. X in position 2: 1 | X | 3 / 4 | 5 | 6 / 7 | 8 | 9

3. X in position 3: 1 | 2 | X / 4 | 5 | 6 / 7 | 8 | 9

4. X in position 4: 1 | 2 | 3 / X | 5 | 6 / 7 | 8 | 9

5. X in position 5: 1 | 2 | 3 / 4 | X | 6 / 7 | 8 | 9

6. X in position 6: 1 | 2 | 3 / 4 | 5 | X / 7 | 8 | 9

7. X in position 7: 1 | 2 | 3 / 4 | 5 | 6 / X | 8 | 9

8. X in position 8: 1 | 2 | 3 / 4 | 5 | 6 / 7 | X | 9

9. X in position 9: 1 | 2 | 3 / 4 | 5 | 6 / 7 | 8 | X

Depth 2: All Possible Responses by O

For each of the 9 possible moves by X, O can respond in 8 different positions (since one position is already occupied by X).

Let's take the example of X in position 5: 1 | 2 | 3 / 4 | X | 6 / 7 | 8 | 9

O's possible responses:

1. O in position 1: O | 2 | 3 / 4 | X | 6 / 7 | 8 | 9

2. O in position 2: 1 | O | 3 / 4 | X | 6 / 7 | 8 | 9

3. O in position 3: 1 | 2 | O / 4 | X | 6 / 7 | 8 | 9

4. O in position 4: 1 | 2 | 3 / O | X | 6 / 7 | 8 | 9

5. O in position 6: 1 | 2 | 3 / 4 | X | O / 7 | 8 | 9

6. O in position 7: 1 | 2 | 3 / 4 | X | 6 / O | 8 | 9

7. O in position 8: 1 | 2 | 3 / 4 | X | 6 / 7 | O | 9

8. O in position 9: 1 | 2 | 3 / 4 | X | 6 / 7 | 8 | O

This process would be repeated for each of the 9 possible moves by X, resulting in a total of 9 \* 8 = 72 possible board states at depth 2.

**Task 3: Classifying the Problem**

Answer the following questions:

1. Is this a **goal-based agent** problem?
2. Is Tic-Tac-Toe a **deterministic** game? Why?
3. Is it a **fully observable** environment?
4. Is it a **single-agent** or **multi-agent** problem?

1. Is this a goal-based agent problem?

Yes, Tic-Tac-Toe is a goal-based agent problem. The agent's goal is to win the game or, at the very least, draw. The agent's actions are guided by this goal, and it makes decisions based on the current state of the board to achieve the desired outcome.

2. Is Tic-Tac-Toe a deterministic game? Why?

Yes, Tic-Tac-Toe is a deterministic game. Given the current state of the board and the actions taken by both players, the outcome of the game is uniquely determined. There are no random elements or uncertainties in the game, and the next state of the board is entirely determined by the current state and the actions taken.

3. Is it a fully observable environment?

Yes, Tic-Tac-Toe is a fully observable environment. Both players have complete knowledge of the current state of the board, including the positions of all X's and O's. There is no hidden information, and both players can observe the entire board.

4. Is it a single-agent or multi-agent problem?

Tic-Tac-Toe is a multi-agent problem. There are two agents (players) interacting with each other in the same environment (the game board). Each agent makes decisions based on the current state of the board and the actions taken by the other agent. The agents have conflicting goals (winning the game), which makes it a competitive multi-agent environment.

**Task 4: Python Mini-Project**

Implement a simple **Tic-Tac-Toe game** in Python:

* Two-player mode (X and O input manually)
* Show the board after every move
* Detect win or draw conditions

(*Optional*: Use a 2D list and functions like check\_win(board))

Task 4: Python Mini-Project

Tic-Tac-Toe Game Implementation:

def print\_board(board):

print(f" {board[0]} | {board[1]} | {board[2]} ")

print("---+---+---")

print(f" {board[3]} | {board[4]} | {board[5]} ")

print("---+---+---")

print(f" {board[6]} | {board[7]} | {board[8]} ")

def check\_win(board):

winning\_combos = [(0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6), (1, 4, 7), (2, 5, 8), (0, 4, 8), (2, 4, 6)]

for combo in winning\_combos:

if board[combo[0]] == board[combo[1]] == board[combo[2]] != " ":

return board[combo[0]]

if " " not in board:

return "Draw"

return False

def main():

board = [" "] \* 9

current\_player = "X"

while True:

print\_board(board)

move = input(f"Player {current\_player}, enter your move (1-9): ")

if board[int(move) - 1] != " ":

print("Invalid move, try again.")

continue

board[int(move) - 1] = current\_player

result = check\_win(board)

if result:

print\_board(board)

if result == "Draw":

print("It's a draw!")

else:

print(f"Player {result} wins!")

break

current\_player = "O" if current\_player == "X" else "X"

if \_\_name\_\_ == "\_\_main\_\_":

main()

How to Play:

1. Run the program.

2. Players X and O take turns entering their moves (1-9).

3. The game board will be displayed after every move.

4. If a player tries to make an invalid move (e.g., a spot already occupied), they will be asked to try again.

5. The game ends when one player wins or when the board is full (a draw).

This implementation uses a list to represent the game board and includes functions for printing the board and checking for win or draw conditions.

**✅ Submission Guidelines**

* Submit your answers to conceptual questions as a Word doc.
* Python code (if attempted) with the Word doc must be pushed to your github repositories.