Assignment: Implementing a Q-Learning Agent for Tic-Tac-Toe (3%)

Introduction

The objective of this assignment is to design and implement a reinforcement learning agent that can learn to play Tic-Tac-Toe optimally using the **Q-Learning** algorithm.

In this assignment, we will develop an autonomous Tic-Tac-Toe agent capable of learning through self-play using **tabular Q-Learning**. The agent will explore different game states, update its Q-values based on outcomes, and gradually improve its strategy until it can consistently avoid losing and play optimally. We will use AI based tools like chat-GPT to design

1. Game Environment

- Representation of the Tic-Tac-Toe board and valid game actions
- Win/loss/draw detection
- Turn-based interaction between the learning agent and an opponent

2. Q-Learning Algorithm

- Q-table structure storing the value of **(state, action)** pairs
- ε-greedy action selection for exploration vs. exploitation
- Q-value updates using the standard Q-Learning rule
  $Q(s,a) = Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

3. Reward System

- **+10** for a win
- **–10** for a loss
- **0** for a draw
- **0** for non-terminal moves

4. Training Procedure

- Self-play episodes
- Exploration decay over time
- Convergence observation (e.g., win rates improving)

5. Evaluation

After training, evaluate the agent by testing it against a random player and against your self

Deliverables

- Python source code for the Q-Learning agent and game environment
- Training logs or plots showing learning progress (e.g., win/draw/loss rates)
- Results of playing against random player and human player as the training improves

Assignment: Implementing a Q-Learning Agent for Tic-Tac-Toe (3%)

Introduction

The objective of this assignment is to design and implement a reinforcement learning agent that can learn to play Tic-Tac-Toe optimally using the **Q-Learning** algorithm.

In this assignment, we will develop an autonomous Tic-Tac-Toe agent capable of learning through self-play using **tabular Q-Learning**. The agent will explore different game states, update its Q-values based on outcomes, and gradually improve its strategy until it can consistently avoid losing and play optimally. We will use AI based tools like chat-GPT to design

1. Game Environment

- Representation of the Tic-Tac-Toe board and valid game actions
- Win/loss/draw detection
- Turn-based interaction between the learning agent and an opponent

2. Q-Learning Algorithm

- Q-table structure storing the value of **(state, action)** pairs
- ε-greedy action selection for exploration vs. exploitation
- Q-value updates using the standard Q-Learning rule
  $Q(s,a) = Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

3. Reward System

- **+10** for a win
- **–10** for a loss
- **0** for a draw
- **0** for non-terminal moves

4. Training Procedure

- Self-play episodes
- Exploration decay over time
- Convergence observation (e.g., win rates improving)

5. Evaluation

After training, evaluate the agent by testing it against a random player and against your self

Deliverables

- Python source code for the Q-Learning agent and game environment
- Training logs or plots showing learning progress (e.g., win/draw/loss rates)
- Results of playing against random player and human player as the training improves