Tic-Tac-Toe with RL

Riyanshi Goyal - 12041240
Ankita Kumari - 12040220
Lahari Sreeja - 12041570
Yedla Usha Sri - 12041780

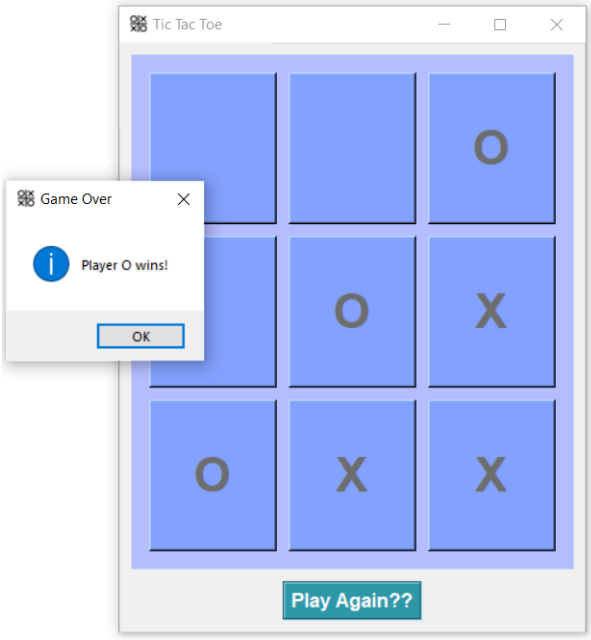
Introduction

What is Tic-Tac-Toe?
A game in which two players seek in alternate turns to complete a row, a column, or a diagonal with either three O's or three X's drawn in the spaces of a grid of nine squares

What is Reinforcement Learning?
Reinforcement learning is a machine learning training method based on rewarding desired behaviors and punishing undesired ones.
In this project, we used different reinforcement learning approaches that enable a model to iteratively learn and improve over time by taking the correct action.
The approaches used:

1. Q-Learning
2. SARSA (State Action Reward State Action)
3. MCTS (Monte Carlo Tree Search)

This is how our interactive tic-tac-toe looks like:



Q-Learning

Optimal action-selection policy for finite Markov decision process (MDP)

In this approach, the project aims to implement a Tic-Tac-Toe game where one of the players is an intelligent agent utilizing Q-learning, a reinforcement learning algorithm, playing against a human.

Key Components:

- **Game Interface:** Tkinter-based GUI with a 3x3 grid for user interaction.
- **Modes :**

    1. Computer Vs Computer (For training the Q-Player)
        ➔ This approach is designed in such a way that two players ('X' and 'O') are trained together by sharing a common Q-table to learn and enhance their strategies through mutual competition.
    2. Human Vs Computer (The trained Q-player then competes against a human)
        ➔ Here, ‘X’ refers to the Human, while ‘O’ refers to the Computer
    3. Human vs. Human

We ran Q-learning for 20,000 interactions, which gave very good results. It's almost impossible to beat the Q-player.

Q-learning Parameters:

Epsilon: 0.9 (Exploration rate - probability of choosing a random move)
Alpha: 0.3 (Learning rate - controls the weight given to new information)
Gamma: 0.9 (Discount rate - determines the importance of future rewards)
Number of Episodes: 20,000 (In the training phase, we make the system play for 20,000 episodes to allow sufficient learning)

This approach is designed in such a way that two players are trained together by sharing a common Q-table. Through mutual competition, both players iteratively enhance their strategies. This method offers an advantage compared to playing against a random opponent since both players develop competence in making strategic moves. Additionally, this approach contributes to a more efficient training process by reducing the iterations needed to refine Q-values.

SARSA Algorithm

State-Action-Reward-State-Action technique, is an On Policy and uses the action performed by the current policy to learn the Q-value.
In this approach, the project aims to implement a Tic-Tac-Toe game where one of the players is an intelligent agent utilizing SARSA, a reinforcement learning algorithm.

SARSA (State-Action-Reward-State-Action) is a reinforcement learning algorithm that is used to learn a policy for an agent interacting with an environment. It is a type of on-policy algorithm, which means that it learns the value function and the policy based on the actions that are actually taken by the agent.

This difference between Q-learning and SARSA technique is visible in the difference of the update statements for each technique:-

1. **Q-Learning:**
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$
2. **SARSA:**
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Key Components:

- **Game Interface:** Tkinter-based GUI with a 3x3 grid for user interaction.
- **Modes :**
    4. Computer Vs Computer (For training the Q-Player)
    5. Human Vs Computer (The trained Q-player then competes against a human)
        ➔ Here, 'X' refers to the Human, while 'O' refers to the Computer

MCTS

It is a probabilistic and heuristic driven search algorithm that combines tree search implementations with reinforcement learning.

The Monte Carlo tree search (MCTS) algorithm consists of four phases:
➔ Selection, Expansion, Rollout/Simulation, Backpropagation.
The implementation includes a TreeNode class for the search tree and covers selection, expansion, simulation, and backpropagation phases.

Key Components:

- TreeNode Class:
    ➔ Represents a node in the search tree with game board details.
    Tracks terminal state, expansion status, parent node, visits, score, and children

nodes.

- MCTS Class:
  - ➔ Manages the MCTS algorithm for Tic-Tac-Toe.
    Implements selection, expansion, simulation, and backpropagation phases.
- Phases:
  - ➔ Selection Phase: Iteratively selects nodes using the UCB1 formula.
  - ➔ Expansion Phase: Generates child nodes for legal moves.
  - ➔ Simulation Phase: Simulates games with random moves until a terminal state.
  - ➔ Backpropagation Phase: Updates node visits and scores, propagating information to the root.

Results:
MCTS effectively selects optimal moves in various game states.

RESULTS

We ran the Q-learning for 20000 iterations, and it gave very good results. It's almost impossible to beat the Q-player. Either the match ends with a draw, or the Q-player wins.

Comparing Monte Carlo Tree Search (MCTS), SARSA technique and Q-learning involves considering various factors, and the choice between them often depends on the characteristics of the problem or game environment. Here are some factors to consider when comparing MCTS and Q-learning:

- **Exploration vs. Exploitation:**
  MCTS: MCTS inherently balances exploration and exploitation through the selection and expansion phases. It is particularly suitable for scenarios where the optimal move may not be known in advance.
  Q-learning: Q-learning relies on a learned Q-table or function, which may struggle in scenarios with a large state or action space. It might need additional exploration strategies.
  SARSA technique : SARSA tends to be more conservative in exploration because it updates its Q-values based on the actions it actually takes.

- **Model-free vs. Model-based:**
  MCTS: Is a model-based approach as it builds and explores a model of the environment (search tree). It doesn't require a pre-defined model of the environment.
  Q-learning: Is a model-free approach. It directly learns the optimal policy or action-value function from interactions with the environment.

- **Implementation Complexity:**
  MCTS: Implementation is often simpler and more intuitive, especially for discrete action spaces.

Q-learning: Can be more complex to implement, especially for continuous action spaces. It requires designing a function approximator and managing exploration. SARSA technique : It updates Q-values based on the agent's own policy, which might make the learned policy more interpretable. If interpretability is a priority, SARSA might be preferable.

Recommendations:

- **MCTS Iterations:**
  Depending on the complexity of the Tic-Tac-Toe game, 1000 iterations might be sufficient for MCTS to converge to a reasonable solution.

- **Q-learning Iterations:**
  Q-learning, especially with neural network approximations, often benefits from a larger number of iterations. 20000 iterations can allow the agent to explore a broader range of states and actions, improving the accuracy of the learned Q-function.

- **SARSA Iterations:**
  40000 iterations may allow the agent to explore a broader range of states and actions, improving the accuracy of the learned function.

NOVELTY

Implemented different approaches for developing the Tic-tac-toe game and compared the advantages and limitations of all the three approaches.