**Reinforcement Learning Game Design – Tic Tac Toe**

**CDS524 Assignment 1**

**MA Xiaorui (1165950)**

**1. Game Design**

**1.1 Objective & Rules**

The objective of the Tic Tac Toe game is simple: the goal is for a player to align three marks (either “X” or “O”) in a row, either horizontally, vertically, or diagonally, within a 3x3 grid. The game alternates between two players, one represented by "X" and the other by "O". The player who first aligns three of their marks in any direction wins the game. If all squares are filled without either player aligning three marks, the game is declared a tie. The game proceeds with each player making moves alternately until a winner is found, or the grid is full.

**1.2 State Space & Action Space**

* **State Space (S)**: The state of the Tic Tac Toe game is represented by the 3x3 grid, where each square can either be empty, marked with "X", or marked with "O". Additionally, the state includes the player whose turn it is to move, either "X" or "O". Therefore, a state can be represented as a 3x3 matrix with values {9, 1, 0}, where 9 represents an empty square, 1 represents "X", and 0 represents "O". The total number of states is .
* **Action Space (A)**: The action space consists of all possible moves a player can make during their turn, which are the coordinates of all empty cells on the grid at the state.

**1.3 Reward Function**

The reward function defines how much the agent (player "X") benefits from taking certain actions in specific states. Based on the game's outcome:

* A reward of **1.0** is given if player "X" wins.
* A reward of **-1.0** is given if player "O" wins.
* A reward of **0.5** is given in the event of a tie, when all squares are filled without a winner.

The reward is designed to encourage the agent to maximize the value when it plays as "X".

**2. Q-learning Implementation**

图示, 应用程序

AI 生成的内容可能不正确。

Figure 1. Overall Framework.

**2.1 Q-learning Algorithm**

Q-learning is a model-free reinforcement learning algorithm used to learn the optimal policy. The agent updates its Q-values, which represent the expected future rewards of taking a certain action from a given state. The update is done using the formula:

where:

* denotes the **Current State**, and denotes the **Action** the agent takes from that state.
* denotes the **Next State** after taking action , and denotes the **Best Next Action** the agent can take in the next state .
* denotes the **Reward** the agent receives after taking action in state .
* **γ** is **Discount Factor**, a value between 0 and 1 that balances the weight between immediate rewards and future rewards.
* **α** is **Learning Rate**, determining the extent to which new information updates the existing Q-values, indicating how much the new knowledge influences the old ones.
* is the **Expected Value**.

In the implementation, Q is structured as a dictionary where the keys represent the game's states (defined by the board status and the player), and the values are inner dictionaries, with keys of each possible move (i.e., the empty squares) and values of current Q-value. An example can be:

{“091191090X”: {(0, 1): -0.8999651254169656, (1, 1): 1.0, (2, 1): -0.8999944254739062}},

where “091191090X” is the games’ state, with “X” as 1, “O” as 0 and blank as 9 showing the current board and “O” or “X” at the end to show whose turn. The board is shown in Figure 1. In addition, (0, 1), (1, 1) and (2,1) are the possible moves and 1.0 and -0.8999651254169656 are the Q values for the corresponding move.

Bellman's equation is applied within the *learn\_Q* method of the *Game* class, which is called after each move. The default Q-values are set to be **1**.

**2.2 Learned Policy**

图片包含 图示

AI 生成的内容可能不正确。

Figure 2. The visualization of the learned policy.

As shown in Figure 1, there are two processes:

1. **Training: Learning the optimal policy by competing with itself.**

This is processed by making computer control both player “X” and player “O”, competing each other, with "X" seeking to maximize the Q values, whereas player "O" seeking to minimize them. To realize the competition two settings are discussed in Section 5. After being trained 100,000 times against itself, a Q dictionary with the best action choice for each condition is constructed. An example is shown in Figure 2.

1. **Testing: Play with human players.**

After the best action-selection policy is learned, humans can test the agent.

**2.3 Hyper-parameters**

* **Epsilon (ϵ)**: This parameter controls the exploration-exploitation trade-off. During training, ϵ is set to a high value (e.g., 0.9) to encourage the agent to explore new actions. In competition with the human player, ϵ is set to 0 to ensure that the agent always selects the best action based on its learned Q-values.
* **Learning Rate (α)**: The learning rate controls how much new information affects the Q-value update. A high learning rate leads to faster updates, but may cause instability, while a low learning rate results in more gradual learning. In this implementation, a learning rate of α=0.3 is typically used.
* **Discount Factor (γ)**: The discount factor determines the importance of future rewards in the Q-value update. In this case, γ is set to be 0.9, meaning the agent values future rewards slightly less than immediate rewards. Its value is less critical in Tic Tac Toe since the game is deterministic and has a finite timesteps.

**3. Game Interaction**

The user interface (UI) for this Tic Tac Toe implementation is built using the **Tkinter** library. The UI consists of a 3x3 grid representing the board. Each grid square is clickable, allowing the human player to place their mark ("X") on the board. The computer (player "O") makes its moves based on the learned Q-values. The game can be restarted by clicking the Reset button.

As shown in Figure 3, the Tkinter interface displays the current state of the board, updates it after each move, and indicates the game's outcome (win, loss, or tie) and the cumulative winning rate. The boar

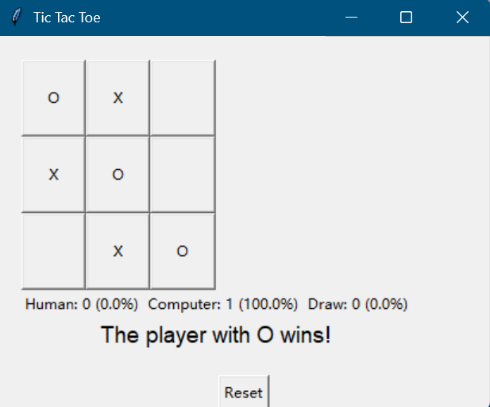


Figure 3. The Tkinter interface.

**4. Evaluation Results**

The number of training episodes 100,1000, 10000, 100000 are tested. After learning 100000 episodes, the agent becomes unbeatable.

**5. Challenges & Solutions**

In this implementation of Tic Tac Toe, rather than making the agent learn from the human by tuning the hyperparameters and playing lots of episodes, I automate the optimal-policy-learning process by making the computer repeatedly competing itself. This allows the model to update its Q-values without the need for human interaction.

To ensure that both players learn to make optimal moves, there are two points to be emphasized.

1. **Reward Function**

In this implementation, the reward is positive for Player X and negative for Player O. Specifically:

* A reinforcement of +1.0 is awarded when Player X wins.
* A reinforcement of -1.0 is awarded when Player O wins.
* A reward of 0.5 is given in the case of a tie.

This reward scheme encourages Player X to maximize the total Q-value (since a win is rewarded positively), while Player O attempts to minimize it (since a loss is rewarded negatively). This "maximization vs. minimization" dynamic simulates the competitive nature of the game, ensuring that both players are pushing towards optimal play but with opposing goals.

1. **Q-values Updating**

Another key point in the code implementation for differentiating the objectives of these two players is that during the update in Q values, player X will predict O’s action with minimal Q values, which is reflected in the computation of Expected Value as . In contrast, Player O will predict X’s action with maximal Q value, reflecting in Expected Value as .

With this setup, the Q-learning algorithm can iteratively update the Q-values for each state-action pair as it simulates games between Player X and Player O. By alternating between actions that maximize and minimize the Q-values, the model converges towards an optimal policy for both players, learning the best strategies through self-play.

**Github Repo**: <https://github.com/XiaoruiMaLU/CDS-524-Assignment-1>

**Demo Video**: <https://youtu.be/1abRTwLivew>