**Reinforcement Learning**

**Tic-Tac-Toe**

* import numpy as np: This imports the NumPy library, which provides a powerful set of functions for working with arrays and matrices.
* import random: This imports the random library, which provides functions for generating random numbers.
* board = np.array([[' ' for \_ in range(3)] for \_ in range(3)]): This creates a NumPy array to represent the Tic-Tac-Toe board.
* Q = {}: This creates an empty dictionary to store the Q-table.
* learning\_rate = 0.8: This defines the learning rate, which controls how much the agent updates its Q-table after each move.
* discount\_factor = 1.0: This defines the discount factor, which controls how much the agent values future rewards compared to immediate rewards.
* exploration\_prob = 0.2: This defines the exploration probability, which controls how often the agent explores new moves instead of choosing the best move according to its Q-table.
* epochs = 50000: This defines the number of training epochs.

**Functions:**

* check\_win(player, board): This function checks if the given player has won the game.
* available\_moves(board): This function returns a list of all available moves on the given board.
* choose\_move(board): This function chooses a move for the agent.
* train\_agent(): This function trains the agent by playing against itself.
* play\_game(): This function plays a game of Tic-Tac-Toe between the agent and the user.

**Explanation of how the code works:**

The code starts by defining the Tic-Tac-Toe board and the agent's Q-table. Then, it defines the hyperparameters for the Q-learning algorithm.

The train\_agent() function trains the agent by playing against itself for a specified number of epochs. At each epoch, the agent chooses a move, observes the state of the board after the move, and receives a reward. The agent then updates its Q-table based on the reward and the new state of the board.

The play\_game() function plays a game of Tic-Tac-Toe between the agent and the user. The user takes turns making moves, and the agent chooses moves according to its Q-table. The game ends when a player wins or there is a draw.

**Here is a more detailed explanation of each function:**

check\_win(player, board):

This function iterates over the rows, columns, and diagonals of the board to check if the given player has won. If the player has three of their marks in a row, column, or diagonal, the function returns True. Otherwise, it returns False.

available\_moves(board):

This function iterates over the board and returns a list of all positions where a move can be made. A position is only available if it does not already contain a mark.

choose\_move(board):

This function chooses a move for the agent. If the exploration probability is greater than a random number, the agent chooses a random move. Otherwise, the agent chooses the move with the highest Q-value.

train\_agent():

This function trains the agent by playing against itself for a specified number of epochs. At each epoch, the agent starts with a fresh board and plays until the game is over. After each move, the agent updates its Q-table based on the reward and the new state of the board.

play\_game():

This function plays a game of Tic-Tac-Toe between the agent and the user. The user takes turns making moves, and the agent chooses moves according to its Q-table. The game ends when a player wins or there is a draw.

**Thank You**