**Report Outline and Explanation of Each Step**

**Project Requirements and Code Steps**

1. **Simple Python Program to Play MiniPong without Reinforcement Learning (RL)**
   * **Requirement**: The first task is to create a simple Python program that plays MiniPong by selecting actions based on the value of dzdzdz (the relative position of the ball to the paddle). This task should modify the existing code within mini\_pong.py to make decisions without RL.
   * **Code Analysis**: I'll check the code in task3-1.py to see how it calculates or retrieves dzdzdz, and whether it performs action selection based on the value of dzdzdz. This code likely checks if the paddle should move up or down depending on the position of the ball relative to the paddle and selects an action accordingly.
2. **Reinforcement Learning (RL) Agent for MiniPong Using Tabular or Deep RL**
   * **Requirement**: Create an RL agent using either tabular or deep RL techniques. For TD-learning (such as Q-learning or SARSA), the agent should start with an exploration rate ϵ=1\epsilon = 1ϵ=1 (pure exploration) and gradually reduce ϵ\epsilonϵ to a minimum of 0.10.10.1 during training, encouraging the agent to explore less over time.
   * **Code Analysis**: I will examine task3-2.py for the RL setup, which likely includes the implementation of an RL algorithm (Q-learning, Deep Q-Network, etc.) and an ϵ\epsilonϵ-greedy strategy to manage exploration versus exploitation. The code should also have a mechanism to decrease ϵ\epsilonϵ over time.
3. **Training the Agent with Episode Resets and Tracking Rewards**
   * **Requirement**: During training, the code should reset the environment after each episode and record the sum of rewards per episode. After training, this data should be used to plot the rewards per episode, showing the agent’s performance improvements over time.
   * **Code Analysis**: I’ll check how the code resets the environment after each episode and accumulates the episode rewards. The code should also store these rewards, which are later used for plotting. The goal of the plot is to visualize the agent’s learning progress.
4. **Testing the Trained Agent and Computing Statistics**
   * **Requirement**: After training is complete, run 50 test episodes with ϵ=0.0\epsilon = 0.0ϵ=0.0 (no exploration) to evaluate the policy. For each episode, the environment should be reset at the start, and the sum of rewards should be calculated. The Test-Average and Test-Standard-Deviation of the rewards over these episodes indicate the agent's performance.
   * **Code Analysis**: I'll examine the testing loop and the method of calculating the average and standard deviation of rewards over 50 episodes.
5. **Impact of Additional Information (Level 2 of MiniPong)**
   * **Requirement**: In Level 2 of MiniPong, the observed state includes both the ball’s yyy-coordinate and dzdzdz. Discuss whether this additional information (the yyy-coordinate) helps or hinders learning.
   * **Explanation**: In the report, I’ll provide an analysis of how the yyy-coordinate might impact learning. Typically, additional information can make it easier for the agent to learn a precise policy, but it can also complicate learning if the information is not directly useful.

**Next Steps**

I’ll go through each file now and examine the code for these steps. Once complete, I’ll provide a detailed report for each requirement and how the code meets or addresses it.

Based on the contents of the two files, here’s a breakdown of how each project requirement is met and what the code does at each stage.

**Report on Code and Requirements Fulfillment**

**1. Simple Python Program to Play MiniPong without Reinforcement Learning**

**Code File**: task3-1.py

**Implementation**:

* The code imports the MiniPongEnv environment and initializes it with level 1 and a size of 5.
* A function, strategy(dz), is defined to select actions based on the relative position dzdzdz of the ball. The strategy is:
  + Move left (return 1) if dz<0dz < 0dz<0,
  + Move right (return 2) if dz>0dz > 0dz>0,
  + Do nothing (return 0) if dz=0dz = 0dz=0.
* The play\_epi function runs a single episode, selecting actions based on dzdzdz and accumulating rewards.
* A loop runs this function for multiple episodes (num\_episodes = 3), storing and printing the total reward for each episode.

**Explanation**:

* This file meets the requirement for a simple non-RL program to play MiniPong by using a rule-based strategy based on dzdzdz.
* The program does not employ any learning but selects actions based on the immediate position of the ball relative to the paddle, demonstrating a simple manual policy.

**2. Reinforcement Learning (RL) Agent for MiniPong Using Tabular or Deep RL**

**Code File**: task3-2.py

**Implementation**:

* A QLearningAgent class is defined to implement Q-learning.
  + **Q-table**: Initialized with bins for discretizing states and actions.
  + **Epsilon-greedy**: The exploration rate ϵ\epsilonϵ starts at 1.0 and decays with each episode using epsilon\_decay=0.995.
  + **Discretization**: The discretize function maps continuous state values to discrete bins.
* The train function iterates through episodes, selecting actions based on the Q-table and ϵ\epsilonϵ-greedy policy, updating Q-values using the temporal difference (TD) update rule, and decaying ϵ\epsilonϵ after each episode.

**Explanation**:

* This class fulfills the requirement for a learning agent using tabular Q-learning with a decaying ϵ\epsilonϵ-greedy approach. It starts with high exploration, gradually shifting towards exploitation, enabling the agent to learn from its environment.
* The agent is trained over a user-defined number of episodes (episodes=1000 in this case).

**3. Training the Agent with Episode Resets and Tracking Rewards**

**Code File**: task3-2.py

**Implementation**:

* During training, the total reward per episode is recorded in rewards\_per\_episode.
* After training, the total rewards for each episode are plotted to visualize the learning progress (Training Reward per Episode).

**Explanation**:

* This section of the code addresses the requirement to track rewards during training and visualize the cumulative rewards per episode, indicating the agent’s learning improvement over time.
* The plot shows how rewards change over episodes, which helps evaluate the effectiveness of the learning process.

**4. Testing the Trained Agent and Computing Statistics**

**Code File**: task3-2.py

**Implementation**:

* The test function evaluates the trained agent over 50 episodes with ϵ=0\epsilon = 0ϵ=0 (no exploration, only exploitation).
* It calculates the average (test\_average) and standard deviation (test\_std) of the rewards over these episodes.
* A plot is generated to show the rewards per test episode (Test Reward per Episode).

**Explanation**:

* This part fulfills the testing requirement by calculating the average performance (Test-Average) and variability (Test-Standard-Deviation) of the agent’s performance, showing how well the learned policy performs without exploration.
* The reward plot provides a visual representation of the agent's consistency across test episodes.

**5. Impact of Additional Information in Level 2 of MiniPong**

**Requirement**: In Level 2 of MiniPong, the observed state includes both yyy-coordinate and dzdzdz.

**Explanation**:

* With the yyy-coordinate, the agent receives additional information, which may help it learn more effectively by providing more precise spatial positioning. However, additional state information may also increase the complexity of learning, as the agent must now incorporate more variables into its decision-making. For a tabular Q-learning agent, this could lead to a larger Q-table and potentially slower learning if state-space discretization is not well optimized.