Experiment Report

**Deep Q-Learning**

Learning Approach:

* The Deep Q-Learning algorithm is implemented through a class **DQNAgent**. Key features of this agent include:
  + A neural network model is built for function approximation. The network consists of two hidden layers with 24 neurons each, using ReLU activation. The output layer has a size equal to the number of actions and uses linear activation.
  + Key parameters: discount rate (**gamma**), exploration rate (**epsilon**), learning rate, and a memory buffer for experience replay.
  + Methods for choosing actions (**act**), storing experiences (**remember**), and learning from experiences (**replay**).

Evaluation Metrics:

* The performance of the agent is evaluated based on the rewards per episode and the loss during training.
* The agent is trained over multiple episodes, with rewards accumulated per episode. Additionally, the average loss during replay (experience replay) is calculated.
* There is also a test phase where the trained agent is evaluated over a number of episodes to compute the average test reward in order to test the robustness of the learned policy.

Graphs/Visualizations:

* Two plots are generated:
  1. Training Rewards per Episode: This plot visualizes the total reward accumulated in each training episode, providing insight into how the agent's performance improves over time.
  2. Training Loss per Replay: This plot shows the loss incurred during each replay (experience replay), indicating how well the agent is learning from past experiences.

**Approximate Q-Learning**

Learning Approach:

* The Approximate Q-Learning algorithm is implemented through a class **ApproximateQLearningAgent**. Key features include:
  + Linear function approximation with weights for estimating Q-values.
  + Key parameters: learning rate (**alpha**), discount rate (**gamma**), exploration rate (**epsilon**), and weight matrix for Q-values.
  + Methods for action selection (**choose\_action**), updating Q-values (**update**), and updating exploration rate (**update\_epsilon**).
* The training process involves iterating over episodes, selecting actions, updating the agent based on rewards and next states, and adjusting the exploration rate.

Evaluation Metrics:

* The performance of the agent is evaluated in two phases:
  1. **Training Phase**: Rewards are accumulated per episode during training. The focus is on how the agent's policy improves over time.
  2. **Testing Phase**: The agent is tested over a set number of episodes, calculating the average reward and average steps per episode which lets us know how robust is the policy learned.

Graphs/Visualizations:

* A single plot is generated to visualize the average rewards per episode during the training phase. This plot helps in understanding the learning progression and effectiveness of the agent's policy over time.

**Comparative Analysis:**

* **Learning Approach**:
  + Deep Q-Learning uses a neural network for approximating Q-values, suitable for complex state spaces. Approximate Q-Learning uses a simpler linear function approximation.
  + Deep Q-Learning includes an experience replay mechanism, enhancing learning from past experiences, which is absent in Approximate Q-Learning.
* **Evaluation Metrics**:
  + Both approaches use rewards per episode as a key metric. Deep Q-Learning also incorporates loss during replay as an additional metric.
  + The test phase in Approximate Q-Learning includes average steps per episode, providing an additional performance dimension.
* **Graphs/Visualizations**:
  + Deep Q-Learning provides more detailed visual insights with two plots (rewards and loss), while Approximate Q-Learning focuses solely on rewards.

In summary, the Deep Q-Learning approach is more complex and potentially more powerful due to its neural network-based approximation and experience replay. In contrast, Approximate Q-Learning offers a simpler yet effective method for environments where linear approximation is sufficient. Due to this difference in this experiment, I was able to get the lighter and simpler approach to converge and find the optimal policy without running out of memory.