Implementing Q-Learning and Deep Q-Learning in Maze Navigation

This paper explores and compares two reinforcement learning techniques, as applied to the problem of maze navigation with multiple traps. The objective is to train agents to efficiently navigate through mazes across various maze complexities.

This project applies two reinforcement learning methods, Q-Learning and Deep Q-Learning, to the specific task of maze navigation. The aim is to enable agents to find the shortest path through a maze, avoiding traps and reaching a designated goal, thus comparing the efficacy of these two approaches in a controlled environment.

1. AI Concepts and Methods

Reinforcement learning involves learning what actions to take in various states, based on rewards and punishments. In this project are two key reinforcement learning techniques: Q-Learning and Deep Q-Learning.

- **Q-Learning:** A value-based learning algorithm, Q-Learning works by learning a function Q(s, a), representing the quality of a particular action 'a' in a given state 's'. This algorithm is particularly effective in scenarios with a discrete, finite state space.
- **Deep Q-Learning (DQL):** An extension of Q-Learning, DQL utilizes deep neural networks to approximate the Q-function. This approach is suited for problems with larger, more complex state spaces, where a table-based representation (as used in Q-Learning) is impractical.

These methods learn from their environment to make decisions, with a notable difference being DQL's ability to handle larger and more complex state spaces, thanks to its neural network-based approach.

2. Description of the Code

Maze Class

This class creates a maze environment. It defines the maze's size, start and goal positions, and places traps randomly. The maze is represented as a grid, with 0s indicating free paths, 1 indicating the goal, and -1 indicating traps.

Deep Q-Learning (DQL) Components

DQL (Deep Q Network): A neural network class that takes an input (the state of the maze) and outputs Q-values for each action. It uses several linear layers with **ReLU** activation.

DQLAgent: This class implements the DQL algorithm. It stores experiences in memory (replay buffer) and uses them to train the DQL. The agent selects actions based on the epsilon-greedy policy, explores the maze, and updates its network based on the reward received.

The train method includes a visualization feature using Pygame to show the maze and the agent's progress. The train_without_visualization method excludes visualization.

Q-Learning Components

QLearningAgent: Implements the tabular Q-Learning algorithm. It maintains a table of Q-values and updates them based on the rewards received from the environment.

Like DQLAgent, this agent also has methods for training with and without visualization.

Testing

Main function: Sets up the maze and agents and runs training sessions for both DQL and Q-Learning. It initializes the maze size, number of traps, number of episodes, and prints the results of each training session.

3. Programming Challenges:

In Deep Q-Learning implementation for maze navigation, I encountered an error due to a mismatch in tensor dimensions during the model's training phase. The issue was resulting from incorrect reshaping of state tensors, which did not align with the neural network's expected input dimensions. To resolve this, I modified the **replay** method in the **DQLAgent** class to correctly reshape the states and next_states into single numpy arrays before converting them into tensors. This ensured that they matched the size expected by the neural network's input layer. This adjustment fixed the dimension mismatch, enabling the neural network to process the state information accurately.

4. Test Results

```
Deep Q Learning Results:
Maze: 5 Traps: 2 Episodes: 1000
Success Rate: 0.992
Average Steps per Episode: 8.639
Wins: 992, Episodes: 1000

Q Learning Results:
Maze: 5 Traps: 2 Episodes: 1000
Success Rate: 0.998
Average Steps per Episode: 8.26
Wins: 998, Episodes: 1000
```

```
Deep Q Learning Results:
Maze: 10 Traps: 6 Episodes: 10000
Success Rate: 0.9932
Average Steps per Episode: 18.1456
Wins: 9932, Episodes: 10000

Q Learning Results:
Maze: 10 Traps: 6 Episodes: 10000
Success Rate: 0.9976
Average Steps per Episode: 18.3074
Wins: 9976, Episodes: 10000
```

```
Deep Q Learning Results:
Maze: 15 Traps: 5 Episodes: 1000
Success Rate: 0.94
Average Steps per Winning Episode: 32.47
Wins: 944, Episodes: 1000

Q Learning Results:
Maze: 15 Traps: 5 Episodes: 1000
Success Rate: 0.981
Average Steps per Episode: 40.01
Wins: 981, Episodes: 1000
```

```
Deep Q Learning Results:
Maze: 18 Traps: 7 Episodes: 5000
Success Rate: 0.9914
Average Steps per Episode: 35.329
Wins: 4957, Episodes: 5000

Q Learning Results:
Maze: 18 Traps: 7 Episodes: 5000
Success Rate: 0.9944
Average Steps per Episode: 41.846
Wins: 4972, Episodes: 5000
```

```
Deep Q Learning Results:
Maze: 18 Traps: 7 Episodes: 500
Success Rate: 0.952
Average Steps per Episode: 38.74
Wins: 476, Episodes: 500

Q Learning Results:
Maze: 18 Traps: 7 Episodes: 500
Success Rate: 0.94
Average Steps per Episode: 119.64
Wins: 470, Episodes: 500
```

```
Deep Q Learning Results:
Maze: 20 Traps: 5 Episodes: 500
Success Rate: 0.97
Average Steps per Winning Episode: 51.23
Wins: 487, Episodes: 500

Q Learning Results:
Maze: 20 Traps: 5 Episodes: 500
Success Rate: 0.962
Average Steps per Episode: 176.084
Wins: 481, Episodes: 500
```

```
Deep Q Learning Results:
Maze: 30 Traps: 10 Episodes: 50
Success Rate: 0.74
Average Steps per Episode: 118.04
Wins: 37, Episodes: 50

Q Learning Results:
Maze: 30 Traps: 10 Episodes: 50
Success Rate: 0.22
Average Steps per Episode: 825.16
Wins: 11, Episodes: 50
```

```
Deep Q Learning Results:
Maze: 30 Traps: 10 Episodes: 1000
Success Rate: 0.86
Average Steps per Episode: 131.997
Wins: 860, Episodes: 1000

Q Learning Results:
Maze: 30 Traps: 10 Episodes: 1000
Success Rate: 0.96
Average Steps per Episode: 274.084
Wins: 960, Episodes: 1000
```

5. Discussion and Analysis

The comparative analysis between Q-Learning and Deep Q-Learning (DQL) across various maze complexities and training durations presents insightful findings. The results indicate that for smaller mazes with fewer traps, Q-Learning slightly outperforms DQL in terms of success rate and average steps per episode. It also performs better in terms of success rate with extended training. This suggests that Q-Learning's memorization approach becomes effective in smaller, less complex environments and when given sufficient time to explore all state-action pairs exhaustively.

However, as the complexity of the mazes increases, DQL begins to show its strengths. In more complex mazes (e.g., 30x30 with 10 traps), DQL not only achieves a higher success rate but also requires fewer steps per episode to find the goal, especially with less training (e.g., 50 episodes). This supports the hypothesis that DQL can generalize better in larger, more complex environments due to its ability to abstract and learn from high-dimensional input spaces.

Notably, when both algorithms are trained for longer periods (e.g., 1,000 episodes on a 30x30 maze), Q-Learning manages to achieve a higher success rate. This highlights the importance of

training duration in environments where Q-Learning can still manage the state-action space size. It's also worth mentioning that the average steps per episode for Q-Learning remain significantly higher compared to DQL, indicating less efficient pathfinding despite its high success rate.

The data also reveals the limitations of Q-Learning in terms of scalability. When the maze size increases to 30x30 with 10 traps, the average steps per episode for Q-Learning dramatically increase, suggesting difficulty in navigating the environment efficiently. In contrast, DQL maintains a more consistent performance, reinforcing its suitability for larger-scale problems.

These findings imply that the selection of an appropriate algorithm should be contingent on the specific characteristics of the problem domain. For simple and well-defined problems where computational resources allow exhaustive exploration, Q-Learning may be adequate. For more complex, dynamic, or uncertain environments, DQL offers a more robust and scalable solution.

Github:

https://github.com/emadsidd/AI_final_project.git