Q-learning base Maze Game Agent Implementation

1 Introduction

In this project, we will create a simple Maze game where an agent learns to reach a goal while avoiding obstacles using Q-learning. The primary objective is to showcase how reinforcement learning can be applied to a grid-based environment, and to illustrate its ability to iteratively improve an agent’s policy through trial and error. The Maze game will consist of a grid world where the agent starts at a random position, with the goal of navigating to a specified endpoint. Along the way, the agent must avoid obstacles that penalize its progress if encountered. We will employ Q-learning, a popular off-policy algorithm, to enable the agent to learn optimal actions through repeated exploration of the environment. By analyzing rewards and penalties, the agent will gradually learn which actions yield the highest long-term gain. Our implementation is designed to run in Google Colab, ensuring that all required dependencies can be managed seamlessly without local setup. Users will be able to visualize the agent’s step-by-step movements within the Maze, observe changes in its learned policy, and experiment with different reward configurations and exploration strategies. Ultimately, this project aims to provide a clear and practical introduction to Q-learning in a fun, interactive way.

2 Game Design

In this maze game, a 7x7 grid serves as a testing ground for Q-learning algorithms. The player or agent must reach a defined goal in minimal steps while avoiding obstacles. This section provides a concise overview of the game's design, highlighting its suitability for evaluating Q-learning algorithms in a controlled environment.

2.1 Game Objective and Rules

To successfully complete the game, the player or agent must guide a block from a designated starting position in the top-left corner of the maze to a predefined goal located in the bottom-right corner. Movement is restricted to four cardinal directions (up, down, left, right), and the block cannot traverse through obstacles. The objective is to reach the goal in the fewest possible moves, encouraging strategic planning and efficient pathfinding. Each step taken by the agent is evaluated, with the goal of minimizing the total number of moves required to solve the maze.

2.2 Spaces Definition

The game environment is characterized by its state space and action space, which are fundamental to the Q-learning process.

2.2.1 state space

The state space represents all possible configurations of the maze. In this 7x7 maze, each unique position of the agent within the grid constitutes a distinct state. Additionally, the presence of obstacles further defines the state space, as the agent must consider their location to avoid collisions. Therefore, the state space can be represented as the combination of the agent's position and the maze layout.

2.2.2 action space

The action space defines the set of possible moves the agent can take in any given state. In this game, the agent can move in four cardinal directions: up, down, left, or right. Each action transitions the agent to a new state, provided the move is valid (i.e., it does not lead to an obstacle or outside the maze boundaries). The action space remains consistent throughout the game, offering the agent a fixed set of choices to explore and learn optimal paths.

2.3 Reward Function Design

The reward function is designed to guide the Q-learning agent toward optimal behavior. Upon successfully reaching the goal, the agent receives a +100 reward, incentivizing goal attainment. Conversely, attempting to move into an obstacle results in a −50 penalty, discouraging actions that lead to invalid states. To promote efficiency, a −1 penalty is applied for each step taken, encouraging the agent to minimize the number of moves required to solve the maze. This combination of rewards and penalties ensures a balance between exploration and exploitation, as the agent learns to navigate the maze effectively while avoiding unnecessary actions. Over time, the agent optimizes its policy to maximize cumulative rewards, leading to the discovery of the shortest path to the goal.

2.4 Detailed Game Design

For the detailed game design, several obstacles are strategically placed throughout the map, and encountering any of these obstacles results in a penalty. The game environment is confined to a 7x7 grid, meaning that any action that would move the player or agent outside these boundaries is not permitted. Additionally, there is a strict limitation of 50 steps, which adds a further challenge by forcing players to carefully plan their route. This step restriction encourages efficient navigation while maximizing the use of available moves, ensuring that players focus on avoiding obstacles and reaching the target goal as quickly as possible.

3 Q-Learning Implementation

In order to implement Q-Learning, this project defines a Q-table and introduces essential parameters, including the learning rate, to control the training process. Building on the Q-table update rules, an epsilon-greedy strategy is incorporated to balance exploration and exploitation, enhancing the overall training performance. The following section will detail these components and explain how they are integrated into the game design to achieve a robust Q-Learning implementation.

3.1 Introduction of Q-Learning

Q-learning is a model-free reinforcement learning algorithm that enables an agent to learn an optimal policy through interaction with its environment. It operates by iteratively updating a Q-table, which stores the expected cumulative rewards for each state-action pair. The update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma a' \max Q(s',a') - Q(s,a)]$$

where *a* is the learning rate, γ is the discount factor, *R* is the immediate reward, and *s'* is the next state. The epsilon-greedy strategy is often used to balance exploration and exploitation, ensuring the agent discovers optimal paths efficiently. For further details, refer to Watkins & Dayan[1], who introduced Q-learning, and Sutton & Barto[2], who provide a comprehensive overview of reinforcement learning techniques.

3.2 Q-Learning Implementation

In the following section, we detail our project's Q-learning implementation, outlining Q-table construction, parameter settings, and training strategies for effective maze navigation.

3.2.1 Programming Language Choice

This project uses Python for its simplicity and powerful libraries, such as NumPy, TensorFlow, and pandas, which are essential for implementing machine learning algorithms. Google Colaboratory is chosen as the experimental environment because it provides free, cloud-based computational resources, including GPU support, and facilitates easy collaboration through shared notebooks. This setup allows for efficient coding, testing, and visualization of the Q-learning approach in a user-friendly platform, accelerating development and experimentation.

3.2.2 Q-table Explanation

The Q-table is implemented as a three-dimensional array, where each dimension corresponds to the maze's x-coordinate, y-coordinate, and the four possible actions (up, down, left, right). This structure allows efficient storage and retrieval of Q-values for every state-action pair in the 7x7 maze. By updating these values based on the Q-learning algorithm, the agent learns to associate optimal actions with specific states, ultimately converging toward an optimal policy for navigating the maze.

3.2.3 Parameter Explanation and Epsilon-Greedy Strategy Implementation

Key parameters were carefully selected to optimize the Q-learning process. The epsilon (ε) value was set to 0.1, representing the exploration rate in the epsilon-greedy strategy. This strategy balances exploration (choosing random actions) and exploitation (choosing actions with the highest Q-values). To encourage convergence, epsilon was gradually decreased over training, allowing the agent to shift focus from exploration to exploitation as it learned the optimal policy.

The learning rate (α) was set to 0.1, determining the extent to which new information overrides old information in the Q-table. A lower learning rate ensures stability but slows convergence, while a higher rate accelerates learning but risks instability.

The discount factor (γ) was set to 0.9, prioritizing immediate rewards while still considering future rewards. This value reflects the importance of long-term planning in the agent's decision-making process. Together, these parameters and the epsilon-greedy strategy enabled the agent to efficiently learn and refine its navigation strategy within the maze environment.

3.3 Game Interaction

3.3.1 User Interface

The user interface includes a visualization function, *visualize_mazedisplay.clear_output()* , which employs matplotlib to display the maze environment. Obstacles are shown in bright blue colors, the agent’s location in darker purple color, and the goal in a bright pink color. Utilizing Google Colab’s enables dynamic updates, allowing the maze visualization to refresh in real time during the training process. This setup offers immediate visual feedback on the agent's performance, making it easier to monitor and debug the Q-learning implementation.

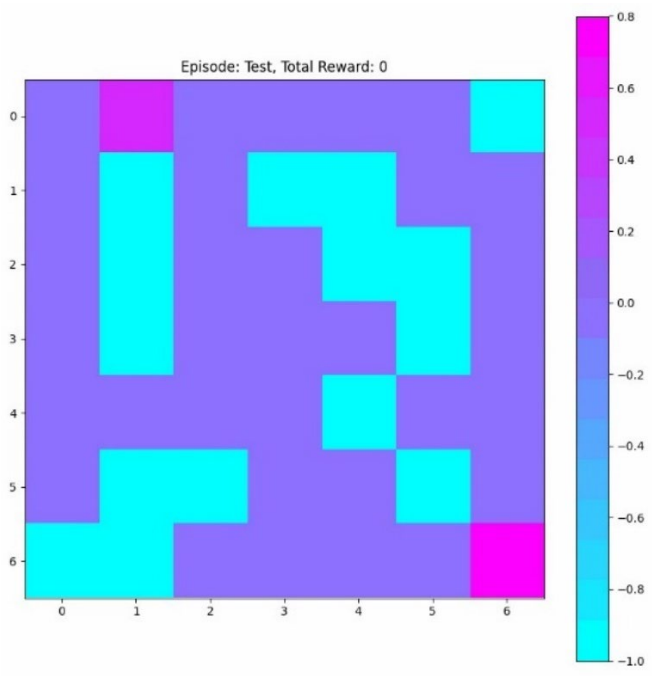

Fig. 3.1 User Interface demonstration

4 Agent Test and Evaluation

With the user interface and dynamic visualization established, we now transition to the Agent Test and Evaluation section. Here, we assess the agent's performance in various maze scenarios, analyze learning progress, and gather key metrics to verify the effectiveness of our Q-learning approach, setting the stage for iterative optimization.

4.1 Agent performance

After training, the agent performed admirably in testing. It reliably avoided obstacles and navigated the maze with near-optimal speed, reaching the target in minimal steps. The Q-learning framework, along with the well-tuned parameters and epsilon-greedy strategy, facilitated effective decision-making throughout. These results indicate that our approach enables robust learning and efficient navigation, forming a solid foundation for further refinement and application in more complex environments.

4.2 Evaluation

The learning curve, plotted with episodes on the x-axis and total reward on the y-axis, provides valuable insights into the Q-learning process. Initially, the curve exhibits significant fluctuations during the first 100 episodes, reflecting the agent's exploration phase as it discovers the environment and learns from rewards and penalties. However, by episode 200, the curve stabilizes around a reward value of 90, indicating convergence to an optimal policy.

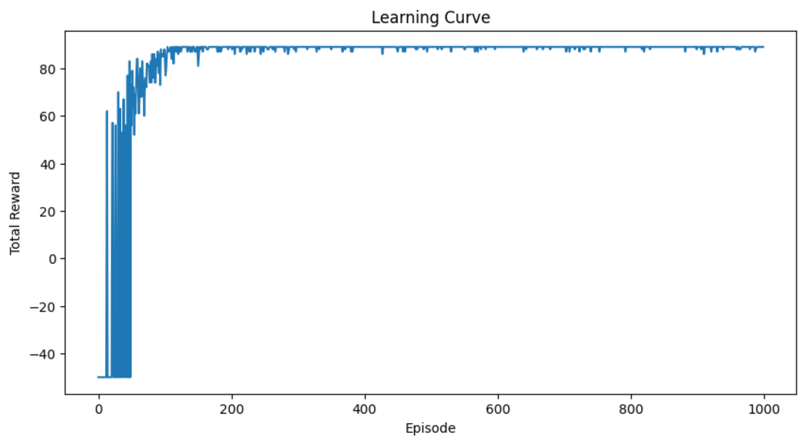

Fig.4.1 Learning Curve plot

Subsequent episodes show minor oscillations between 85 and 90, with the amplitude of these fluctuations decreasing over time. This behavior suggests that the agent has refined its strategy and is consistently achieving near-optimal performance. The rapid convergence and stabilization of the learning curve highlight the efficiency of Q-learning in this maze environment, demonstrating its ability to balance exploration and exploitation effectively.
In conclusion, the learning curve confirms that the agent successfully learned to navigate the maze while minimizing steps and avoiding obstacles. The observed convergence and reduced variability over episodes underscore the robustness and reliability of the Q-learning algorithm for solving this task.

5 Discussion

In conclusion, this project successfully demonstrates the application of Q-learning within a maze environment using a 7x7 grid. The agent, guided by a carefully structured Q-table, learned to navigate the maze efficiently while avoiding obstacles, as evidenced by its robust performance during testing. By implementing essential parameters such as epsilon, alpha, and gamma along with an epsilon-greedy exploration strategy, the learning curve converged steadily, affirming the effectiveness of our approach. The dynamic visualization in Google Colab further allowed for real-time monitoring and iterative debugging, enhancing the overall development process.
Looking ahead, there is potential to extend this work by exploring more complex environments and fine-tuning parameters to further optimize navigation strategies. Future research could also integrate advanced reinforcement learning algorithms or incorporate deep learning methods for higher-dimensional problems. This project not only offers a solid foundation for understanding basic Q-learning mechanics but also serves as a springboard for more sophisticated applications in artificial intelligence and robotics.

References:

[1] Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279 – 292.
[2] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.