SID:3160368
Name:XIE WEIRU

Maze game based on Q-Learning algorithm

1 introduction

In the field of artificial intelligence, reinforcement learning, as an important learning paradigm, optimizes behavioral strategies through the interaction between agents and the environment to maximize long-term cumulative rewards. Q-Learning algorithm is a classical algorithm in reinforcement learning, which is excellent in solving complex decision problems. This report designs and implements a maze game based on Q-Learning algorithm. It aims to gain an in-depth understanding of reinforcement learning principles and demonstrate its application to path planning problems.

2 Game design

2.1 Game goal

The core goal of the game is to guide the agent through a complex maze environment. Starting from the entrance to the maze, the agent can only move one space at a time in four directions, up, down, left and right. The agent needs to find the shortest path to the exit without moving out of the maze boundary or entering the trap area. This goal provides a clear direction for the agent to learn and act.

2.2 State space

The state space is the set of all possible states that an agent can be in during a game. In this 6x6 maze, each cell represents a possible state. There are 36 cells in the maze, which means there are 36 different states of the agent. These states are precisely represented in the form of coordinates. Each state corresponds to the specific position of the agent in the maze, and this position information determines the next action the agent can take and the reward it can get.

2.3 Action space

The action space defines the set of actions that the agent can perform in each state. In this game, the action space consists of four basic actions: up, down, left, and right. No matter which cell the agent is in the maze, it can choose one of these four actions to perform. These actions determine the direction of the agent's movement in the maze, which in turn affects its exploration path in the maze and whether it can reach the end successfully.

2.4 Reward function

The reward function is the key mechanism to guide the decision-making of the agent in maze game. In the game, when the agent reaches the end point, it will receive a positive reward of 100. This is an affirmation of its completion of the task and motivates it to explore towards the end. If the agent enters the trap area, it receives a negative reward of -10, which alerts it to avoid the dangerous area. However, when moving on the ordinary path, only -1 negative reward can be obtained each time to avoid meaningless wandering of the agent on the ordinary path. Through this reward setting, the agent can learn through trial and error and gradually master the optimal strategy. While avoiding traps, quickly find a route to the finish line to maximize rewards.

3 Q-Learning algorithm implementation

3.1 Q-Learning algorithm implementation

Q-learning is a model-free reinforcement learning algorithm designed to allow agents to learn optimal strategies through interaction with the environment. It uses Q values to assess the long-term value of performing a particular action in a certain state. In the maze game, when the agent selects an action in each state, it will refer to the Q-value table, which is updated according to the Bellman equation. In the process of exploration, the agent gradually learns the optimal strategy by constantly trying different actions and updating the Q value table according to the reward feedback.

In order to balance exploring new moves and exploiting existing experience, the epsilon-greedy strategy is used. The agent randomly selects an action for exploration with a probability of $\varepsilon=0.1$, and chooses the action with the highest Q value in the current state with a probability of $1-\varepsilon$. As the training progresses, the agent gradually increases the use of existing experience and reduces random exploration.

3.2 Algorithm execution and results

3.2.1 Execution process

Before training begins, initialize the Q value table, setting the Q value of all state-action pairs to 0. At the same time, set the maximum number of training rounds to 500, and the maximum number of steps per round to 100.

During each training turn, the agent moves from the beginning of the maze. At each step, the agent selects an action according to the epsilon-greedy strategy, and after executing the action, obtains a new state and the corresponding reward. Then, the Q-value table is updated according to the Q-learning update formula. In this process, the Pygame library is used to draw the maze and the position of the agent in real time, and the information such as current action, cumulative reward and current step number is displayed on the interface.

3.2.2 Training results

At the beginning of training, since the agent mainly conducts random exploration, the cumulative reward is low, and it often enters the trap. As the training progresses, the agent gradually learns effective strategies, and the cumulative reward begins to steadily rise. After about 200 training sessions, the cumulative reward growth trend leveled off, indicating that the agent had learned a more stable strategy. Through the analysis of the training results, it is found that the agent can get from the starting point to the end point with fewer steps in the late training period, and the average step number is about 10 steps. This proves that Q-learning algorithm successfully learns the optimal path in this game. In addition, the results obtained by repeated training are similar, which indicates that the algorithm has good stability.

4 Game interaction

The game UI is built using the Pygame library. The whole is divided into maze display area and information display area. The maze display area accurately presents the 6×6 maze, with different colors to distinguish the starting point, the end point, the trap and the ordinary path, so that the player can intuitively understand the maze layout. The function pane displays the current status, actions performed by the agent, and rewards and punishments obtained by the agent in real time. For example, information such as Action: up and Total Reward: -5 is displayed. Using Pygame library graphics drawing function, draw each cell and agent of the maze. The event processing mechanism implements the interaction logic between the agent and the environment to ensure that the agent's moving operations can respond in time. The whole UI design is simple and intuitive, color matching is reasonable, and the feedback of agent status changes is timely, so that players can clearly observe the game progress.

5 Evaluation result

In the multiple rounds of experiments on Q-learning algorithm in games, the learning rate, discount factor and exploration rate are emphatically adjusted. At first, these parameters are set to 0.8. After that, the learning rate is gradually reduced to 0.1, the discount factor is increased to 0.9, and the exploration rate is reduced to 0.1. The entire training process is fixed at 500 iterations.
The experimental results show that before parameter adjustment, the agent's exploration efficiency in the maze is low, and the probability of successfully exiting the maze is not high. With the optimization of parameters, the learning effect of the agent becomes more and more remarkable in 500 iteration training. In the game's criteria, if the agent can get out of the maze in 10 moves, its learning is judged to be effective. In the early stage of training, the agent mainly relies on random exploration. Agents often get trapped, the average number of steps out of the maze is far more than 10, the probability of successful maze, and the cumulative reward is low. However, with the progress of training and parameter adjustment, the agent gradually finds a better strategy, and the probability of successfully walking out of the maze is greatly increased.
By observing the change of Q value, it can be seen that the Q value fluctuates sharply at the beginning of training. This is because the agent constantly tries various actions in the exploration phase, and has not yet formed a stable strategy, so the Q value is extremely unstable. With the progress of training, the agent accumulates more experience, and the Q value begins to converge slowly. At the later stage of training, the Q value of most state-action pairs is getting smaller and smaller, and tends to be stable. This change clearly reflects the agent's increasing mastery of the optimal strategy. The ability to make better decisions in the maze environment and the probability of successfully completing the maze in less than 10 moves is also increasing.

6 Challenges and solutions

In the course of the experiment, I encountered some challenges. The first is the parameter tuning problem. Learning rate, discount factor and exploration rate affect each other, and improper value will make the learning effect of the agent poor. At the beginning of the design of the maze process, the intelligence will take a few steps forward and then backward, always wandering. After checking the problem, it was found that the exploration times were small and the experience value was insufficient. To this end, we use the method of multiple rounds of comparative experiments. By setting different parameter combinations for training, record the cumulative reward, number of steps and Q value convergence of the agent, so as to determine the appropriate parameters. Changing the explore value to 500 or more significantly improves the effect. The second problem is UI interface design. When designing the UI interface, I first encountered the problem that the agent went to the exit but did not display it. Therefore, modify and improve the UI interface design, when the agent walks to the exit, the agent layer floats above the exit layer.

7 Conclusion

This project designs and implements a maze game based on Q-Learning algorithm. Through clear game design, effective algorithm implementation, friendly user interface and reasonable evaluation and analysis, the effectiveness of Q-Learning algorithm in solving path planning problem is demonstrated. In the future, the algorithm can be further optimized. Such as introducing more complex exploration strategies or combining deep learning techniques to improve the learning efficiency and adaptability of agents. At the same time, enrich the game scenarios and tasks, and expand the application of reinforcement learning in more fields.

8 appendix

https://github.com/Nwnono/Assignment-1.git
https://youtu.be/vtHGpzJac8E