Reinforcement Learning: Q-Learning

1 Project Introduction

1.1 Background

Reinforcement learning is a machine learning method. Through the interactive feedback system (reward and punishment) between **Agent** and **Environment**, the agent needs to use a series of decisions and state transitions to achieve the preset goal.

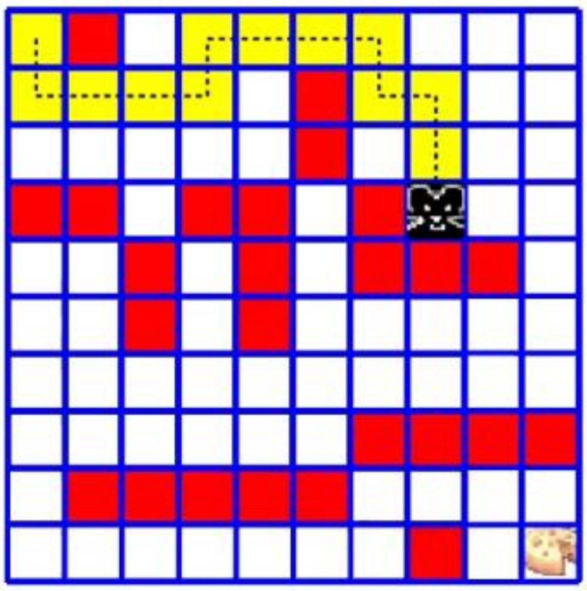
A classic example is training a rat (an intelligence agent) to find the shortest path to a cake in a maze (an environment). Agents use **Exploration** and **Exploitation** of past **Experiences** to achieve their goals. It may fail over and over again, but after a long period of trial and error, the agent can finally find the answer to the problem.

The value of **Accumulated Rewards** can be maximized when an intelligent agent can continuously find an optimal state in the long run. (in short, an algorithm with feedback rewards can be used to induce an intelligent agent to achieve a goal by continuously acquiring rewards.)

In addition, the agent may have to endure many penalties (negative rewards) in order to achieve the goal. For example, the mouse in the maze was given a slap on the wrist for every legal action because we wanted it to take the shortest possible route to reach the target unit, otherwise it would be rewarded for wandering around the maze at will. The shortest path to the target cake can sometimes be long and convoluted, and the agent (the mouse) may have to endure many penalties until it finally reaches the **Delayed Reward** goal.

1.2 Maze Problem

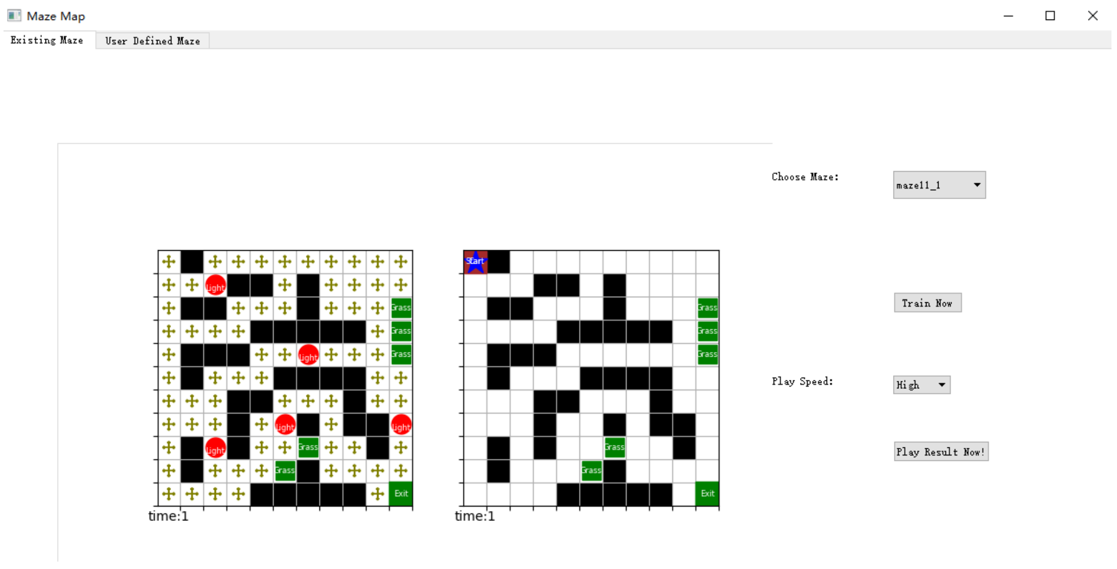
Maze problem has been applied in data structure and algorithm research. The well-known Dijkstra shortest path algorithm is still one of the most practical methods to solve these problems. But because of the intuitive nature of the maze problem, it is well suited to demonstrate and test reinforcement learning techniques.



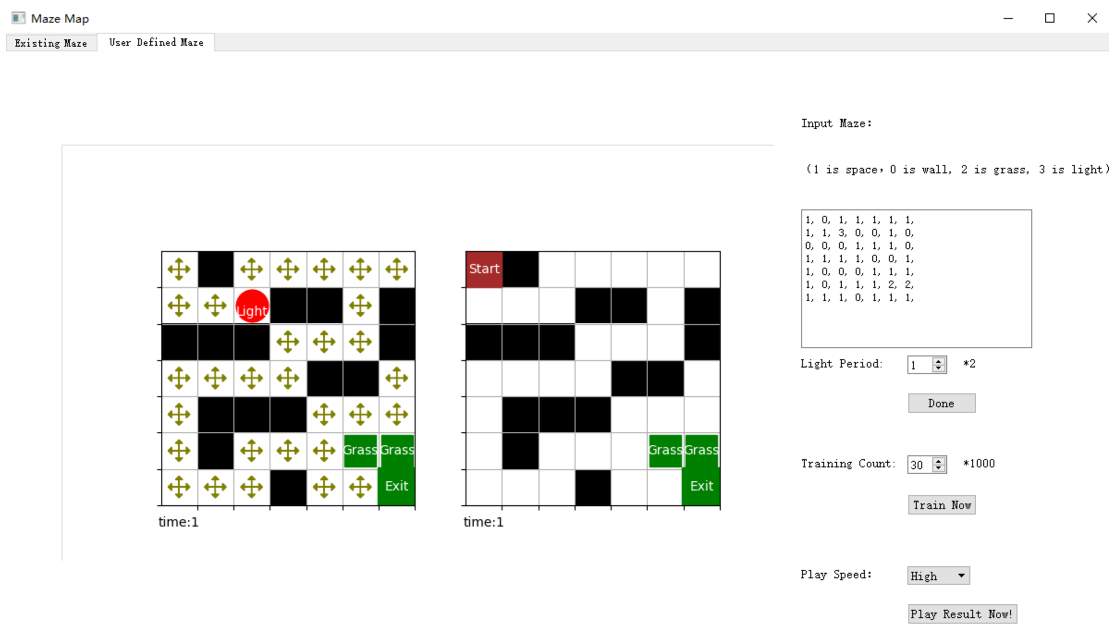
An algorithm with **Feedback Rewards** can be used to induce an intelligent agent to achieve a goal by continuously acquiring rewards.

1.3 My Project

The maze form of this project is shown in the figure, in which the stars are cars, the green square is grass and the red circle is signal light. Cars are not allowed to walk into grass or walls. Cars are not allowed to move when red light appear. The period of the red lights flashing can be defined by the user.



In my project, users can also define their own maze, training times, etc.



Input Maze:

(1 is space, 0 is wall, 2 is grass, 3 is light)

```
1, 0, 1, 1, 1, 1, 1,
1, 1, 3, 0, 0, 1, 0,
0, 0, 0, 1, 1, 1, 0,
1, 1, 1, 1, 0, 0, 1,
1, 0, 0, 0, 1, 1, 1,
1, 0, 1, 1, 1, 2, 2,
1, 1, 1, 0, 1, 1, 1,
```

Light Period: 1 *2

Training Count: 30 *1000

Play Speed: High

2 Project Modelling

2.1 MDP

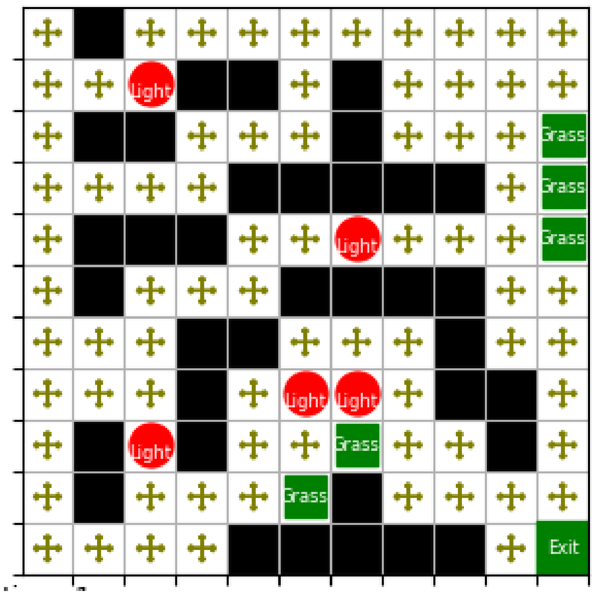
The framework of the Markov Decision Process MDP consists of the environment and the intelligence agent acting in the environment.

In our example, the environment is a classic square maze with five types of squares:

- Wall
- The blank space
- Target square (where the Exit is)
- Light
- Grass

Our agent is a car that is only allowed to move on a blank space for the sole purpose of finding an exit.

Four actions: up, down, left, right.



2.2 Reward

With each step (an action) the car takes in the maze, the maze returns a reward, the game status (win or lose), and the next position for the car.

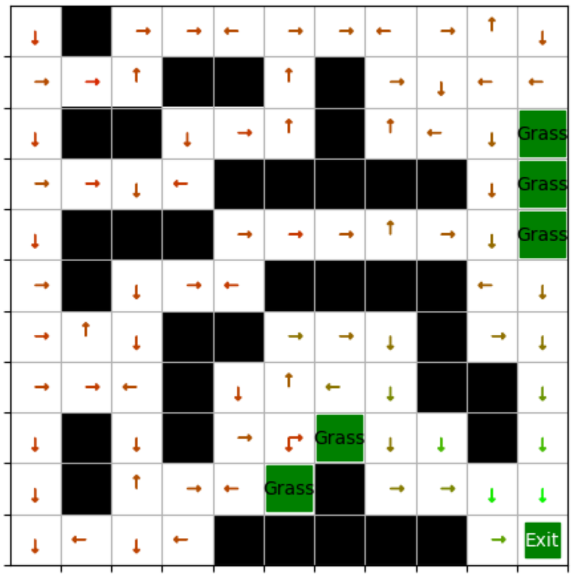
- A reward is almost always a negative value, except for the action of reaching the destination.
- Moving reward is set to a negative value, it is to reduce the length of the mouse's route and reach the end as soon as possible.
- To prevent the game from going on forever, the game will end when the reward has accumulated enough small.

Action	Reward
Move	-0.05
Knock the Wall	-0.85
Repeat the Path that Walked	-0.3
Die	-2
Arrive the End	1

2.3 Q-Table

The Q-Learning algorithm is used for training to obtain the Q table: **Action Value Function**, which has an action value corresponding to each action in each state. When making decisions, the action with the highest action value in the current state will be selected.

Choose the one with the largest Q value from the four actions in each state and draw it with an arrow. The color indicates the size of the Q value, and the Q value increases from red to green.



2.4 Q-Learning

Q-Learning is an **Offline Strategy of Learning**, behavioral strategy ***ε*-greedy strategy** is used, is beneficial to explore. The goal strategy uses the greedy strategy, to make full use of the experience of learning.

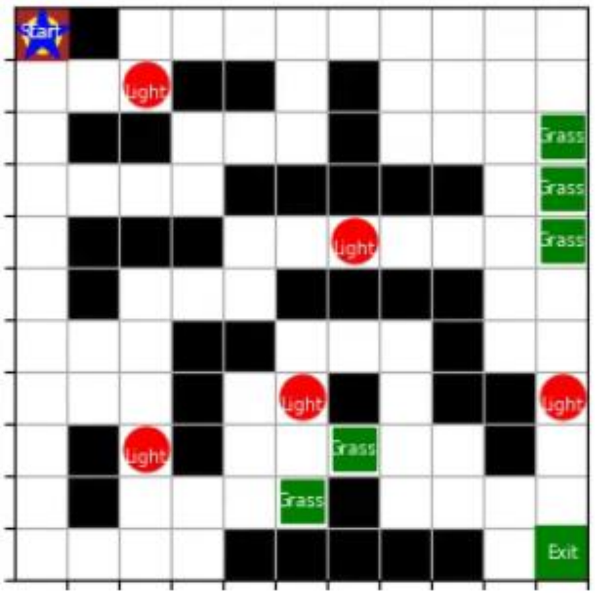
Offline Strategy: The method used to update the Q value is to follow the new policy. The Max action in the Q table allows the update of the Q table not to be based on the current experience

***ε*-greedy:** Each state is explored with a probability of (that is, a random choice to go or not to go), and the remaining probability of 1- selects the action with a higher utility value in the current state

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
        Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $a$, observe $r$, $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
        $s \leftarrow s'$;
    until $s$ is terminal
```

2.5 Time Factor

The time factor is introduced and the flashing interval of the signal light changes accordingly. We can see the signal light flashes at a certain period in this figure.



3 Project Implementation

3.1 Data Structure

Maze

```
'maze11_1': [
    [1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 3, 0, 0, 1, 0, 1, 1, 1, 1],
    [1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 2],
    [1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 2],
    [1, 0, 0, 0, 1, 1, 3, 1, 1, 1, 2],
    [1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1],
    [1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 1, 0, 1, 3, 0, 1, 0, 0, 3],
    [1, 0, 3, 0, 1, 1, 2, 1, 1, 0, 1],
    [1, 0, 1, 1, 1, 2, 0, 1, 1, 1, 1],
    [1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1],
```

Reward

```
REWARD = {"move": -0.05, "finish": 1.0, "wall": -0.85, "bound": -0.85, "repeat": -0.3, "dead": -2}
```

Q-Table

```
def __init__(self, my_maze, epsilon=0.1, learning_rate=0.1, gamma=0.9):
    self.Q_table = dict()
    self.my_maze = my_maze  # 迷宫
    self.epsilon_ = epsilon  # 探索因子，通常设置为0.1，意味着在每10次action中，有一次汽车将采取完全随机的动作
    self.learning_rate = learning_rate  # 学习率
    self.gamma = gamma  # 折现系数
    self.hsize = my_maze.maze.size // 2
```

3.2 Train

```
# 训练的次数
for epoch in range(epoch_N):
    if self.my_maze.period == 0:
        car_cell = random.choice(self.my_maze.free_cells)
    else:
        car_cell = (0, 0)
    self.my_maze.reset(car_cell, 0)
    game_over = False

    state = self.my_maze.get_current_state_simple()

    n_episodes = 0
    while not game_over:
        valid_actions = self.my_maze.valid_actions()
        if not valid_actions: break
        state_now = state

        # epsilon-贪心算法
        if np.random.rand() < self.epsilon_:
            action = random.choice(valid_actions)
        else:
            action = self.predict(state_now)
```

```
# 实施action
state_next, reward, game_status = self.my_maze.act(action,
                                                    self.my_maze.get_current_state_simple)

if (state, action) not in self.Q_table.keys():  # 确保存在值(state, action)以避免密钥错误
    self.Q_table[(state, action)] = 0.0

max_next_Q = max([self.Q_table.get((state_next, a), 0.0) for a in ACTIONS])
self.Q_table[(state, action)] += self.learning_rate * (
        reward + self.gamma * max_next_Q - self.Q_table[(state, action)])

if game_status == 'win':
    win_history.append(1)
    game_over = True
elif game_status == 'lose':
    win_history.append(0)
    game_over = True
else:
    game_over = False

state = state_next

n_episodes += 1
```

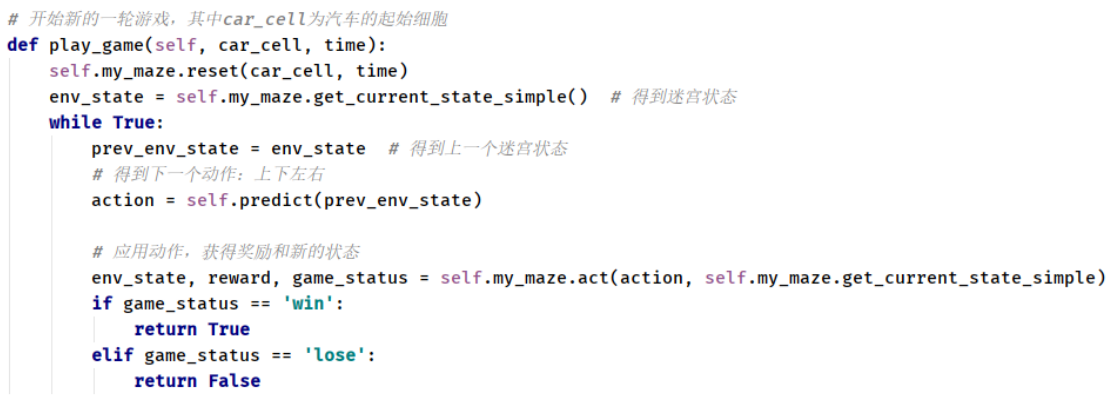
3.3 Completion Check

In this function, we will examine each step of each cell in the maze to see if any errors have occurred. This function for smaller mazes, but for larger mazes, the time cost is greater.

```
# 完成度检测, 检验每一个cell的动作是否不会出现任何错误
def completion_check(self):
    if self.my_maze.period == 0:
        period_temp = 1
    else:
        period_temp = self.my_maze.period
    for time_ in range(period_temp):
        for cell in self.my_maze.free_cells:
            if not self.my_maze.valid_actions(cell):
                return False
            if not self.play_game(cell, time_):
                return False
    return True
```

3.4 Run the Game

Simulate the entire game based on the given model.



```
# 开始新的一轮游戏，其中car_cell为汽车的起始细胞
def play_game(self, car_cell, time):
    self.my_maze.reset(car_cell, time)
    env_state = self.my_maze.get_current_state_simple()  # 得到迷宫状态
    while True:
        prev_env_state = env_state  # 得到上一个迷宫状态
        # 得到下一个动作：上下左右
        action = self.predict(prev_env_state)

        # 应用动作，获得奖励和新的状态
        env_state, reward, game_status = self.my_maze.act(action, self.my_maze.get_current_state_simple)
        if game_status == 'win':
            return True
        elif game_status == 'lose':
            return False
```

4 Conclusion

For this assignment, I tried the Q table model. At the beginning, I intuitively thought that the supervised learning model was better, but after the completion, I found that the training speed was slow and the algorithm was complex.

I implemented the Q table model, which is fast in training and intuitive in algorithm. This made me realize the importance of problem analysis and literature research. Different solutions should be adopted for different problems, from difficult to easy.

I completed and designed the UI interface. Users have a great degree of freedom in the UI interface and can customize the maze and maze parameters.

Although this big assignment took me a long time, I realized the pleasure of completing a complete project independently and gained a sense of achievement. This not only improved my programming ability, deepened my understanding of reinforcement learning, but also improved my ability to solve problems and work under pressure. I got a full harvest, and my efforts were rewarded.

Finally, thanks very much for your teaching!

References:

[1] Deep Reinforcement Learning for Maze Solving

[2] Q-Learning

[3] Creating Autonomous Vehicle Systems 2018, by Shaoshan Liu, Liyun Li, Jie Tang, Shuang Wu, Jean-Luc Gaudiot

[4] Autonomous Driving-Technical, Legal and Social Aspects 2015, Markus Maurer, J. Christian Gerdes, Barbara Lenz, Hermann Winner (Editors)