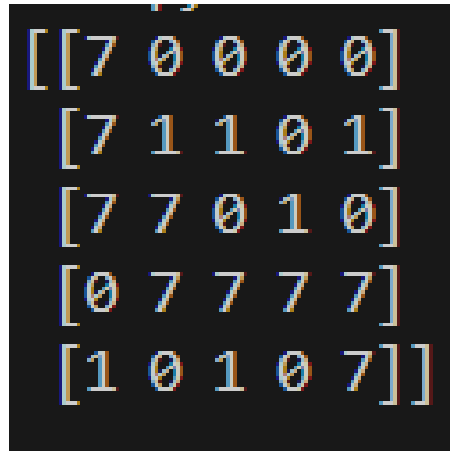


Trần Quang Minh B2304069

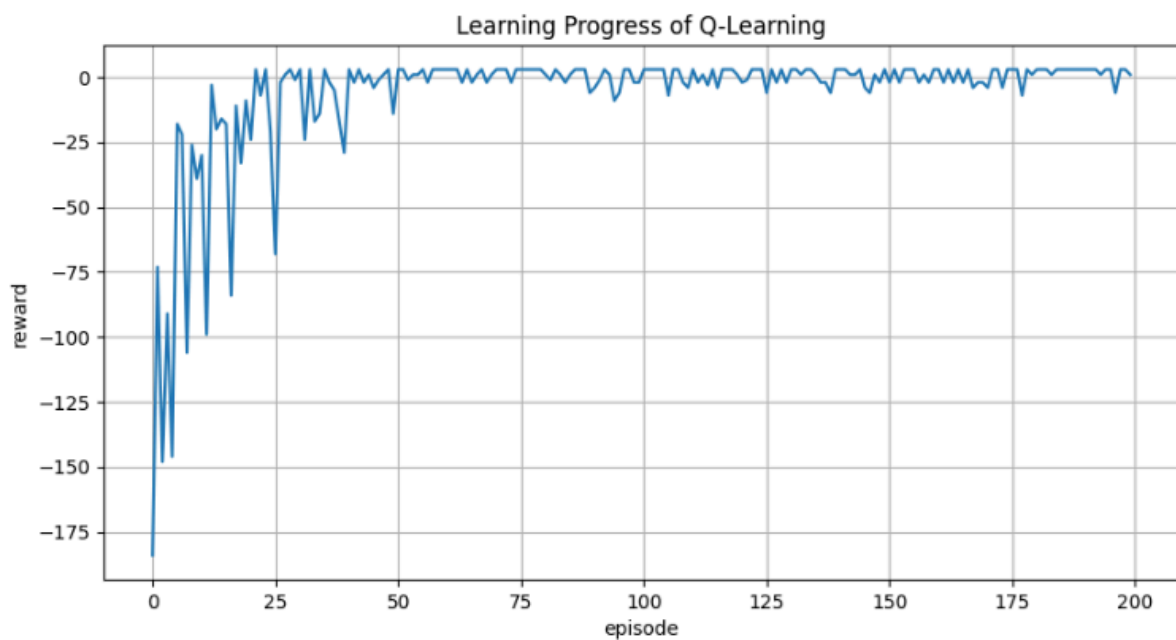
Bài Thực Hành Số 5

a



[7	0	0	0	0]
[7	1	1	0	1]
[7	7	0	1	0]
[0	7	7	7	7]
[1	0	1	0	7]

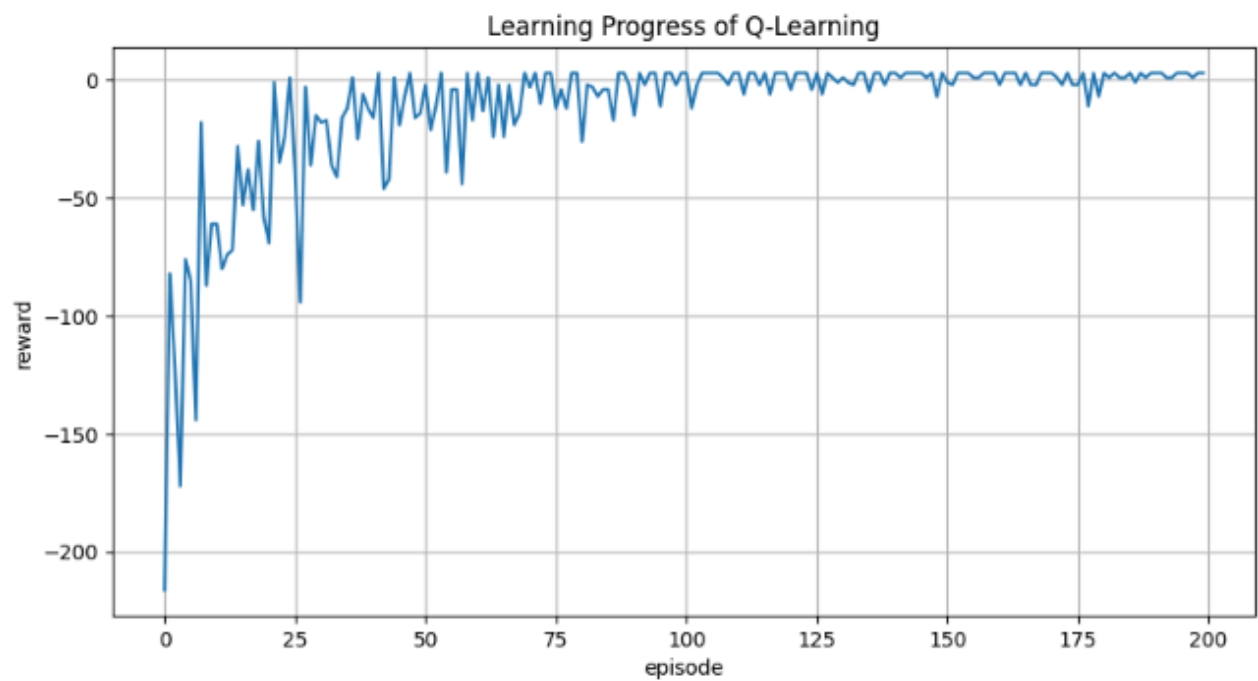
Số 7 tượng trưng cho agent



b

```
[[7 0 0 0 0]
 [7 1 1 0 1]
 [7 7 0 1 0]
 [0 7 7 7 0]
 [1 0 1 7 7]]
```

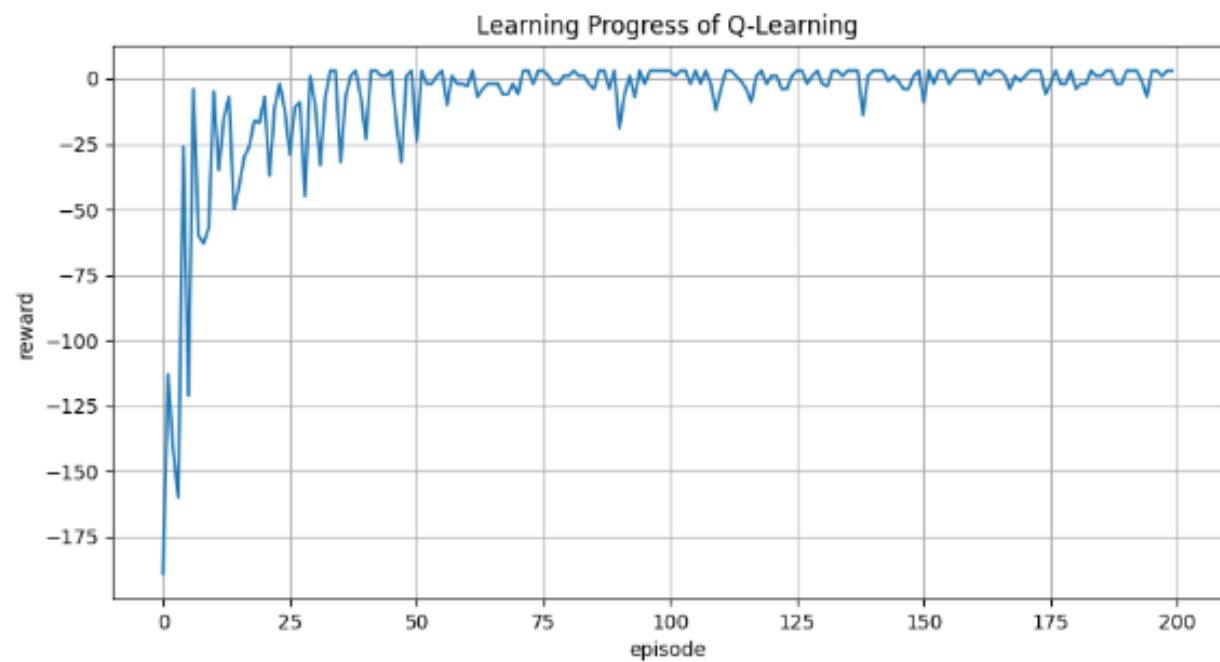
Số 7 tượng trưng cho agent



C

```
[[7 0 0 0 0]
 [7 1 1 0 1]
 [7 7 0 1 0]
 [0 7 7 7 7]
 [1 0 1 0 7]]
```

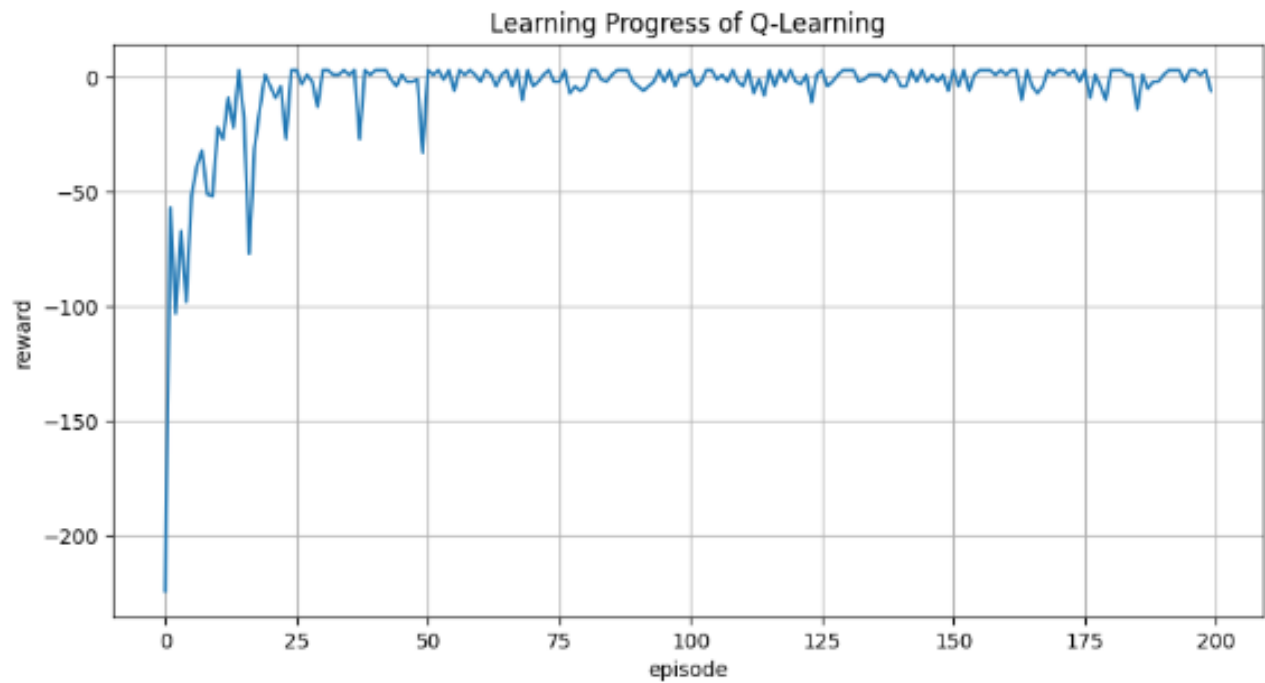
Số 7 tượng trưng cho agent



d

```
[[7 0 0 0 0]
 [7 1 1 0 1]
 [7 7 0 1 0]
 [0 7 7 7 7]
 [1 0 1 0 7]]
```

Số 7 tượng trưng cho agent



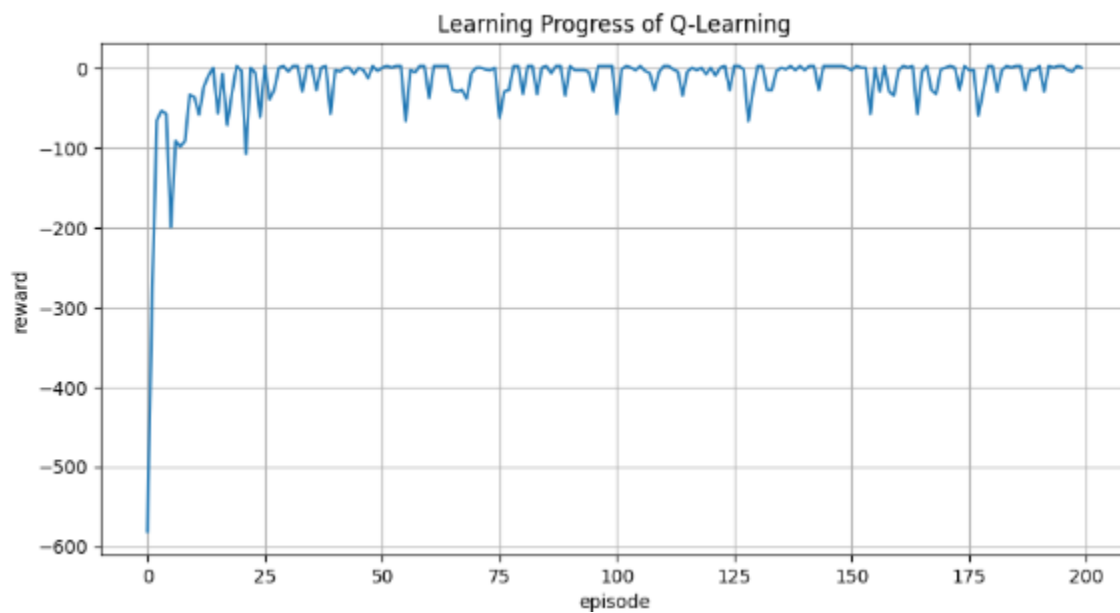
e

Sự khác nhau ở đồ thị tích lũy tổng phần thưởng b, c, d là do sự khác nhau của 3 tham số alpha, gamma, epsilon. Mỗi tham số đều đóng góp vào việc lựa chọn nước đi của agent trong thuật Q-Learning

d

```
[[7 0 0 0 0]
 [7 1 1 0 1]
 [7 1 0 1 0]
 [7 7 7 7 7]
 [1 0 1 0 7]]
```

Số 7 tượng trưng cho agent



f

```
import numpy as np
import matplotlib.pyplot as plt

learning_rate = 0.15
gamma = 0.9
epsilon = 0.15

maze = np.array([
    [0, 0, 0, 0, 0],
    [0, 1, 1, 0, 1],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 0, 0],
```

```

        [1, 0, 1, 0, 0]
    ])
    start = (0, 0)
    goal = (4, 4)
    max_steps = 100
    episodes = 200

    q_table = np.zeros((maze.shape[0]*maze.shape[1], 4))

    def state_to_index(state):
        return state[0]*maze.shape[1] + state[1]

    def step(state, action):
        x, y = state
        if action == 0:
            next_x, next_y = x-1, y
        elif action == 1:
            next_x, next_y = x+1, y
        elif action == 2:
            next_x, next_y = x, y-1
        else:
            next_x, next_y = x, y+1

        if (next_x < 0 or next_x >= maze.shape[0] or next_y < 0 or next_y >=
maze.shape[1]):
            return state, -5

        if (maze[next_x, next_y] == 1):
            return state, -30

        next_state = (next_x, next_y)
        if next_state == goal:
            reward = 10
        else:
            reward = -1

        return next_state, reward

    def get_action(state):

```

```

    if np.random.rand() < epsilon:
        action = np.random.randint(4)
    else:
        action = np.argmax(q_table[state_to_index(state)])

    return action

def learn(state, action, next_state, reward):
    q_value = q_table[state_to_index(state), action]
    next_max = np.max(q_table[state_to_index(next_state)])
    q_table[state_to_index(state), action] = q_value + learning_rate*(reward +
gamma*next_max - q_value)

### Train
total_rewards = []
for _ in range(epochs):
    state = start
    total_reward = 0

    for _ in range(max_steps):
        action = get_action(state)
        next_state, reward = step(state, action)

        learn(state, action, next_state, reward)

        state = next_state
        total_reward += reward
        if state == goal:
            break

    total_rewards.append(total_reward)

# Test
state = start
maze[start] = 7
step_counter = 0
while state != goal and step_counter < max_steps:
    action = np.argmax(q_table[state_to_index(state)])
    state, _ = step(state, action)

```

```
    step_counter += 1
    maze[state] = 7

print(maze)

# Plot
plt.figure(figsize=(10, 5))
plt.title("Learning Progress of Q-Learning")
plt.xlabel("episode")
plt.ylabel("reward")
plt.grid(True)
plt.plot(total_rewards)
plt.savefig("plot_e.png")
```