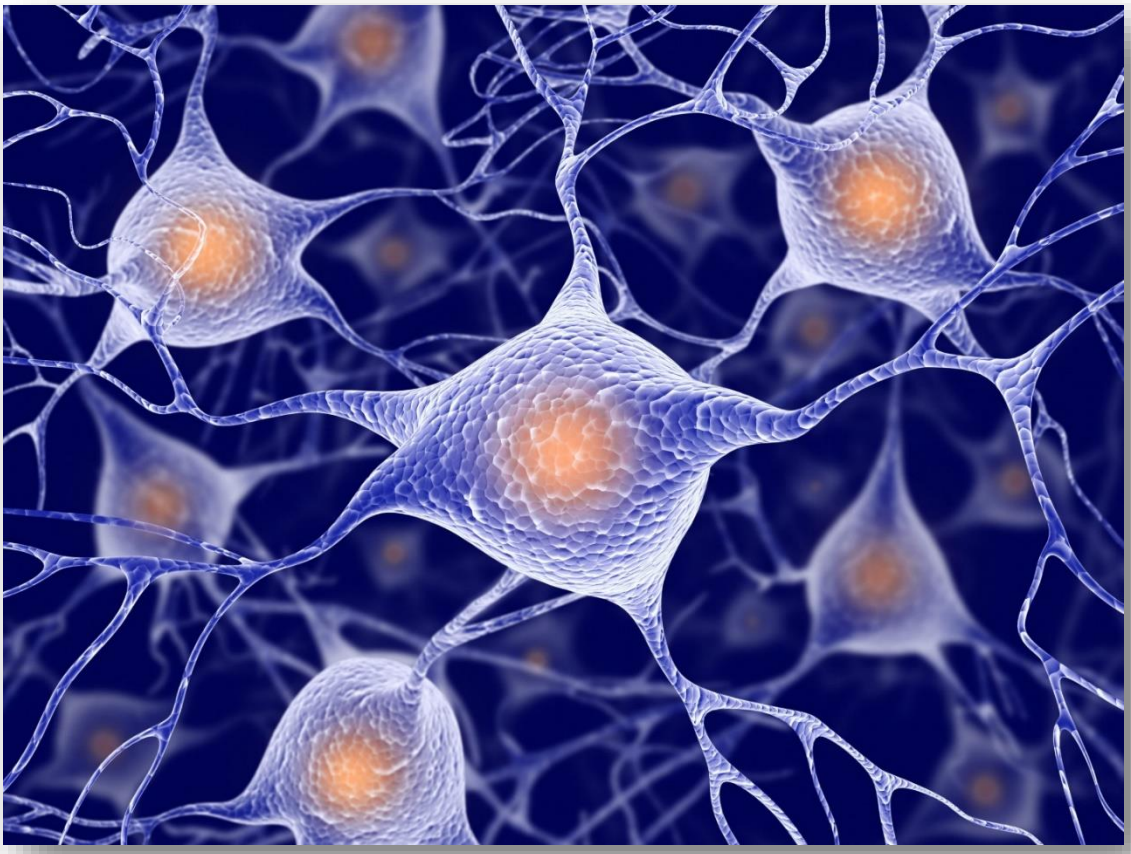

Neuroscience, Learning, Memory, Cognition

DR. Karbalaie

Report Project

Name:

Mohammad Ghafourian

Student number:

99106493

1 Introduction

In $N \times N$ game we only allowed to go right or up, and find a path that has the most numerous final reward, in this RL project we are using ϵ-greedy algorithm(Q-value) a well-known reinforcement learning algorithm.

2 Q-Learning Algorithm

Reinforcement learning (RL) is a branch of machine learning, where the system learns from the results of actions. In this tutorial, we'll focus on **Q-learning**, which is said to be an off-policy temporal difference (TD) control algorithm. It was proposed in 1989 by Watkins.

We create and fill a table storing state-action pairs. The table is called $\boldsymbol{Q}$ or $\boldsymbol{Q - table}$ interchangeably.

Q(S, A) in our Q-table corresponds to the state-action pair for state S and action A. R stands for the reward. t denotes the current time step, hence t+1 denotes the next one. Alpha (α) and gamma (γ) are learning parameters, which we'll explain in the following sections.

In this case, possible values of state-action pairs are calculated iteratively by the formula:

$$Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \rightarrow Q(S_t, A_t)$$

Equation 1: Epsilon-Greedy Q-learning Function

This is called the action-value function or **Q-function**. The function approximates the value of selecting a certain action in a certain state. In this case, Q is the action-value function learned by the algorithm. Q approximates the optimal action-value function $Q*$.

3. Q-Learning Properties

Q-learning is an off-policy temporal difference (TD) control algorithm, as we already mentioned. Now let's inspect the meaning of these properties.

3.1. Model-Free Reinforcement Learning

Q-learning is a model-free algorithm. We can think of model-free algorithms as trial-and-error methods. The agent explores the environment and learns from outcomes of the actions directly, without constructing an internal model or a Markov Decision Process.

In the beginning, the agent knows the possible states and actions in an environment. Then the agent discovers the state transitions and rewards by exploration.

3.2. Temporal Difference

It is a TD algorithm, where the predictions are reevaluated after taking a step. Even incomplete episodes generate input for a TD algorithm. Overall, we measure how the last action is different from what we estimated initially, without waiting for a final outcome.

In Q-learning, Q-values stored in the Q-table are partially updated using an estimate. Hence, there is no need to wait for the final reward and update prior state-action pair values in Q-learning.

3.3. Off-Policy Learning

Q-learning is an off-policy algorithm. It estimates the reward for state-action pairs based on the optimal (greedy) policy, independent of the agent's actions. An off-policy algorithm approximates the optimal action-value function, independent of the policy. Besides, off-policy algorithms can update the estimated values using made up actions. In this case, the Q-learning algorithm can explore and benefit from actions that did not happen during the learning phase.

As a result, Q-learning is a simple and effective reinforcement learning algorithms. However, due to greedy action selection, the algorithm (usually) selects the next action with the best reward. In this case, the action selection is not performed on a possibly longer and better path, making it a short-sighted learning algorithm.

4 Epsilon-Greedy Q-Learning Algorithm

Now let's look at how the Q-learning algorithm works.

Algorithm 1: Epsilon-Greedy Q-Learning Algorithm

Data: α: learning rate, γ: discount factor, ϵ: a small number
Result: A Q-table containing Q(S,A) pairs defining estimated optimal policy π^*

```
/* Initialization                                          */
```
Initialize Q(s,a) arbitrarily, except Q(terminal,.);
Q(terminal,.) $\leftarrow$ 0;
```
/* For each step in each episode, we calculate the
   Q-value and update the Q-table                          */
```
for *each episode* **do**
```
   /* Initialize state S, usually by resetting the
      environment                                          */
```
   Initialize state S;
   for *each step in episode* **do**
      do
```
         /* Choose action A from S using epsilon-greedy
            policy derived from Q                         */
```
         A $\leftarrow$ SELECT-ACTION(Q, S, ϵ);
         Take action A, then observe reward R and next state S';
         Q(S, A) $\leftarrow$ Q(S, A) + α [R + $\gamma \max_a$ Q(S', a) - Q(S, A)];
         S $\leftarrow$ S';
      while *S is not terminal*;
   end
end

Figure 1: Epsilon-Greedy Q-Learning Algorithm

As we take action A, our state changes from S to S'. In the meantime, we get a reward of R. Then we use these values to update our Q-table entry $Q(S, A)$. We repeat until we reach a terminal state.

5 Action Selection

First we are going to introduce Multi-Armed Bandit Problem

Multi-Armed Bandit Problem

The multi-armed bandit problem is used in reinforcement learning to formalize the notion of decision-making under uncertainty. In a multi-armed bandit problem, an agent(learner) chooses between k different actions and receives a reward based on the chosen action.

The multi-armed bandits are also used to describe fundamental concepts in reinforcement learning, such as rewards, time steps, and values.

For selecting an action by an agent, we assume that each action has a separate distribution of rewards and there is at least one action that generates maximum numerical reward. Thus, the probability distribution of the rewards corresponding to each action is different and is unknown to the agent(decision-maker). Hence, the goal of the agent is to identify which action to choose to get the maximum reward after a given set of trials.

5.1. Exploration vs. Exploitation Tradeoff

To better understand how the action selection is performed, we need to first understand the concepts of exploration and exploitation.

In reinforcement learning, the agent tries to discover its environment. As mentioned above, model-free algorithms rely on trial-and-error. During these trials, an agent has a set of actions to select from. Some of the actions are previously selected and the agent might guess the outcome. On the other hand, some actions are never taken before.

In a multi-armed bandit problem, the agent initially has none or limited knowledge about the environment. The agent can choose to explore by selecting an action with an unknown outcome, to get more information about the environment. Or, it can choose to exploit and choose an action based on its prior knowledge of the environment to get a good reward.

The concept of exploiting what the agent already knows versus exploring a random action is called the exploration-exploitation trade-off. When the agent explores, it can improve its current knowledge and gain better rewards in the long run. However, when it exploits, it gets more reward immediately, even if it is a sub-optimal behavior. As the agent can’t do both at the same time, there is a trade-off.

As we already stated, initially the agent doesn’t know the outcomes of possible actions. Hence, sufficient initial exploration is required. If some actions lead to better rewards than others, we want the agent to select these options. However, only exploiting what the agent already knows is a dangerous approach.

For example, a greedy agent can get stuck in a sub-optimal state. Or there might be changes in the environment as time passes. As a result, we wish to keep a balance between exploration and exploitation; not giving up on one or another.

5.2. Epsilon-Greedy Action Selection

In Q-learning, we select an action based on its reward. The agent always chooses the optimal action. Hence, it generates the maximum reward possible for the given state.

In epsilon-greedy action selection, the agent uses both exploitations to take advantage of prior knowledge and exploration to look for new options:

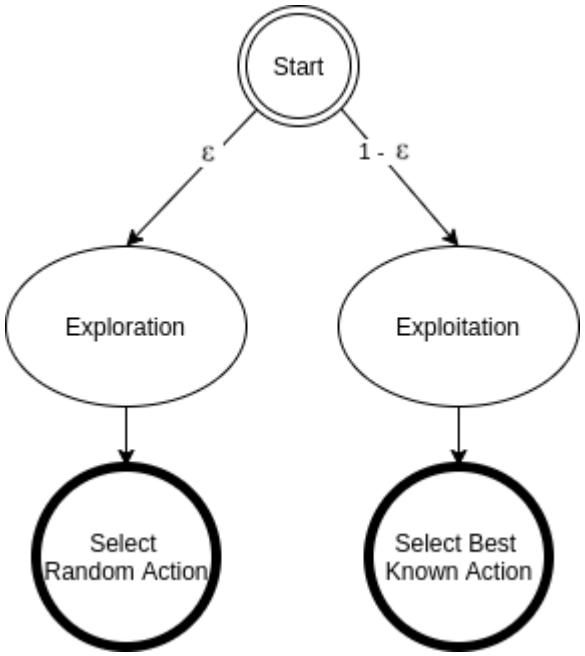

Figure 2: Epsilon-Greedy Action Selection

The epsilon-greedy approach selects the action with the highest estimated reward most of the time. The aim is to have a balance between exploration and exploitation. Exploration allows us to have some room for trying new things, sometimes contradicting what we have already learned.

With a small probability of ϵ, we choose to explore, i.e., not to exploit what we have learned so far. In this case, the action is selected randomly, independent of the action-value estimates.

If we make infinite trials, each action is taken an infinite number of times. Hence, the epsilon-greedy action selection policy discovers the optimal actions for sure.

Now let’s consider the implementation:

Algorithm 2: Epsilon-Greedy Action Selection

Data: Q: Q-table generated so far, : a small number, S: current state

Result: Selected action

Function *SELECT-ACTION(Q, S, ϵ)* **is**

    n ← uniform random number between 0 and 1;
    if $n < \epsilon$ **then**
        A ← random action from the action space;
    else
        A ← maxQ(S,.);
    end
    return selected action A;

end

Figure 3: Epsilon-Greedy Action Selection Algorithm

As we've discussed above, usually the optimal action, i.e., the action with the highest Q-value is selected. Otherwise, the algorithm explores a random action. An epsilon-greedy algorithm is easy to understand and implement. Yet it's hard to beat and works as well as more sophisticated algorithms.

6 Epsilon-Greedy Q-learning Parameters

As we have mentioned the algorithm previously, the algorithm needs three parameters two of them which are α, γ (alpha and gamma) are used in Q-learning and the third one which is ϵ (epsilon) is used in action selection

$$Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \rightarrow Q(S_t, A_t)$$

Equation 1: Epsilon-Greedy Q-learning Function

Where Q represents the Q-Value corresponding to a specific pair of states and actions, α is the learning rate, R represents the immediate reward of a certain state, and γ is the discount factor.

6.1. Alpha ($\boldsymbol{\alpha}$)

Similar to other machine learning algorithms, alpha ($\boldsymbol{\alpha}$) defines the learning rate or step size. As we can see from the equation above, the new Q-value for the state is calculated by incrementing the old Q-value by alpha multiplied by the selected action's Q-value.

Alpha is a real number between zero and one $(0 < \alpha \leq 1)$. If we set alpha to zero, the agent learns nothing from new actions. Conversely, if we set alpha to 1, the agent completely ignores prior knowledge and only values the most recent information. Higher alpha values make Q-values change faster.

6.2. Gamma (γ)

Gamma (γ) is the discount factor. In Q-learning, gamma is multiplied by the estimation of the optimal future value. The next reward's importance is defined by the gamma parameter.

Gamma is a real number between 0 and 1 $(0 \leq \gamma \leq 1)$. If we set gamma to zero, the agent completely ignores the future rewards. Such agents only consider current rewards. On the other hand, if we set gamma to 1, the algorithm would look for high rewards in the long term. A high gamma value might prevent conversion: summing up non-discounted rewards leads to having high Q-values.

6.3. Epsilon (ϵ)

Epsilon (ϵ) parameter is related to the epsilon-greedy action selection procedure in the Q-learning algorithm. In the action selection step, we select the specific action based on the Q-values we already have. The epsilon parameter introduces randomness into the algorithm, forcing us to try different actions. This helps not getting stuck in a local optimum.

If epsilon is set to 0, we never explore but always exploit the knowledge we already have. On the contrary, having the epsilon set to 1 force the algorithm to always take random actions and never use past knowledge. Usually, epsilon is selected as a small number close to 0.

7 Code Implementation

First of all, let's to mention our parameters in final correct answer. In final answer our episode equals to 60000, as you know this number is much less than 118264581564861424 which is total paths we can test ($\frac{60!}{30!\times 30!}$). the alpha, gamma and epsilon equal to 0.6, 0.8 and 0.1 respectively

Measure the Qs with

```
reward = get_reward(next_state)
q_table[state[0], state[1], action] = (1 - alpha) * q_table[state[0], state[1], action] + alpha * (reward + gamma * np.max(q_table[next_state[0], next_state[1], :]))
state = next_state
```

code 1: Q-value Calculator

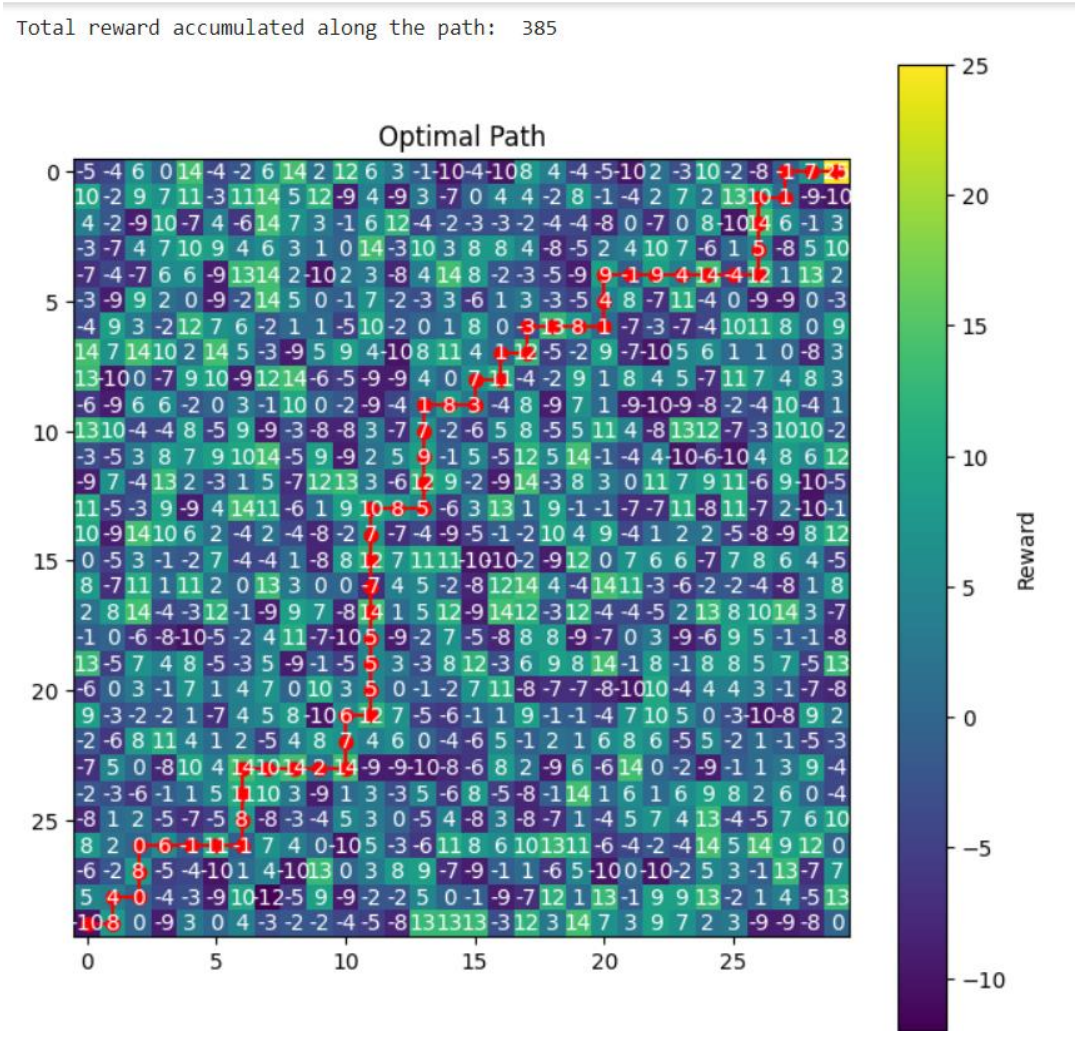

Figure 4: Final Result

Now let's go to see other results (the code is explained completely in the jupyter note book)

```
alpha = 0.1  # Learning rate
gamma = 0.9  # Discount factor
num_episodes = 2000  # Total number of episodes
```

Figure 5: Parameter 1

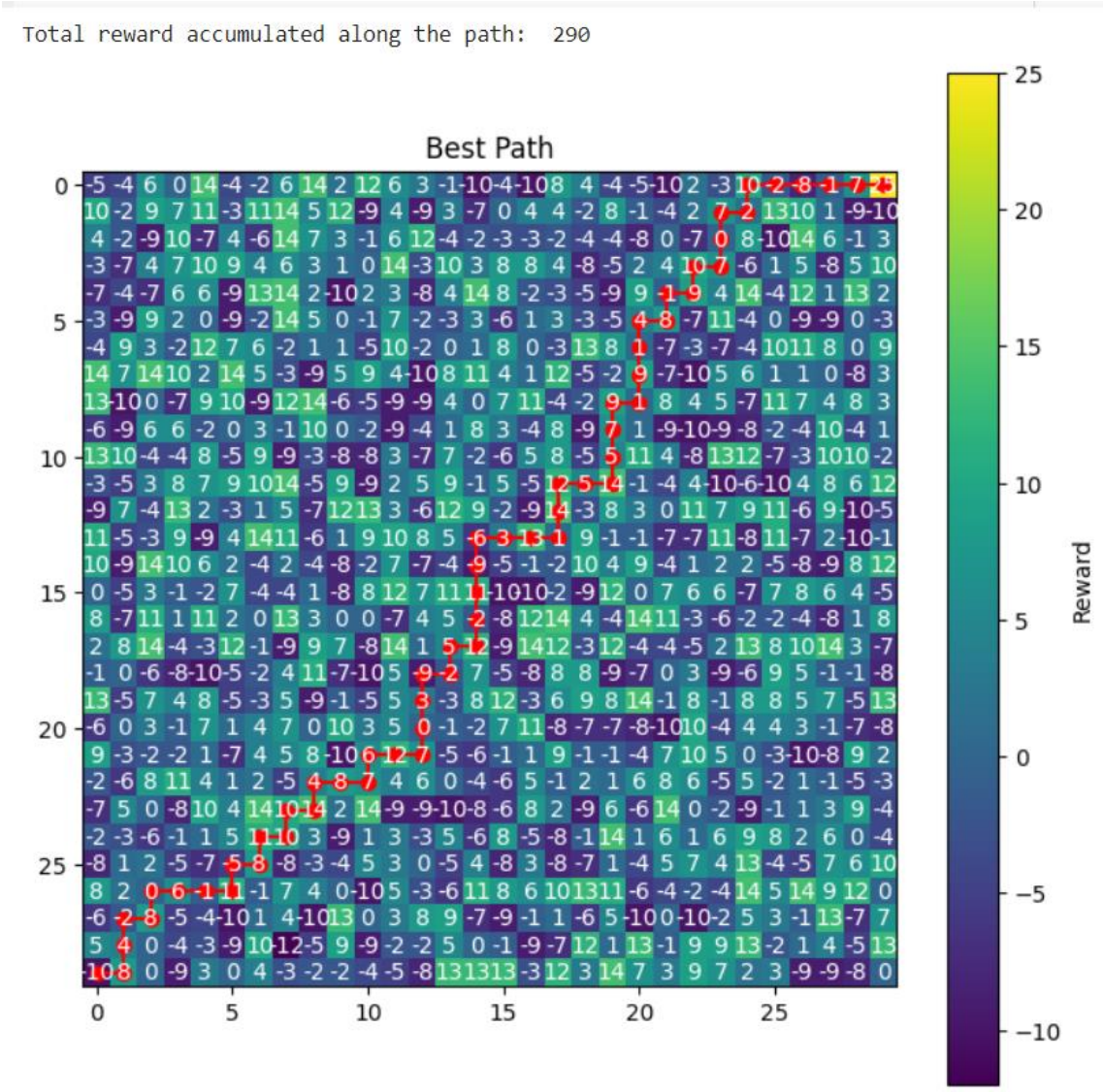

Figure 6: Result 1.1

Episode = 20000

```
Total reward accumulated along the path:  318
```

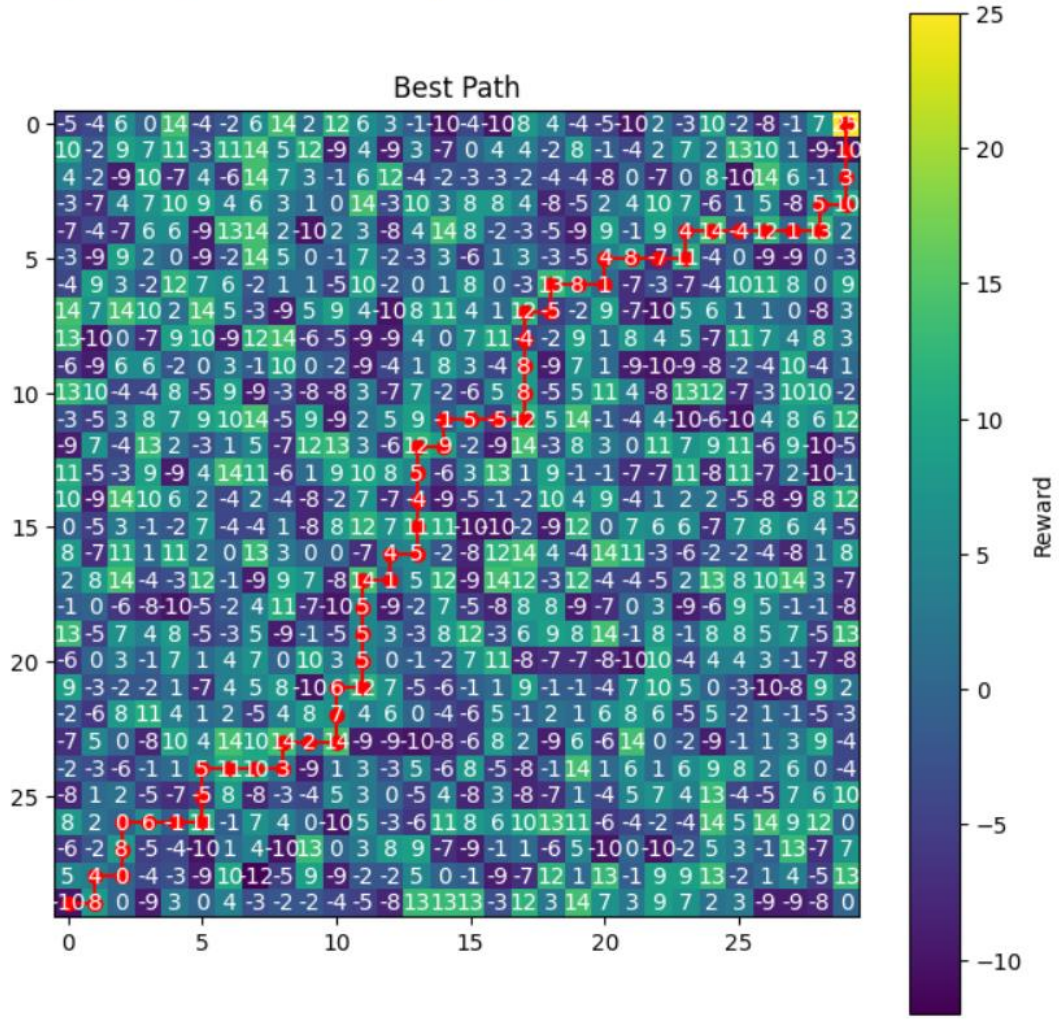

Figure 7: Result 1.2

```
alpha = 0.6  # Learning rate
gamma = 0.8  # Discount factor
num_episodes = 20000  # Total number of episodes
```

Figure 8: Parameter 2

We can see a huge improvement in the result

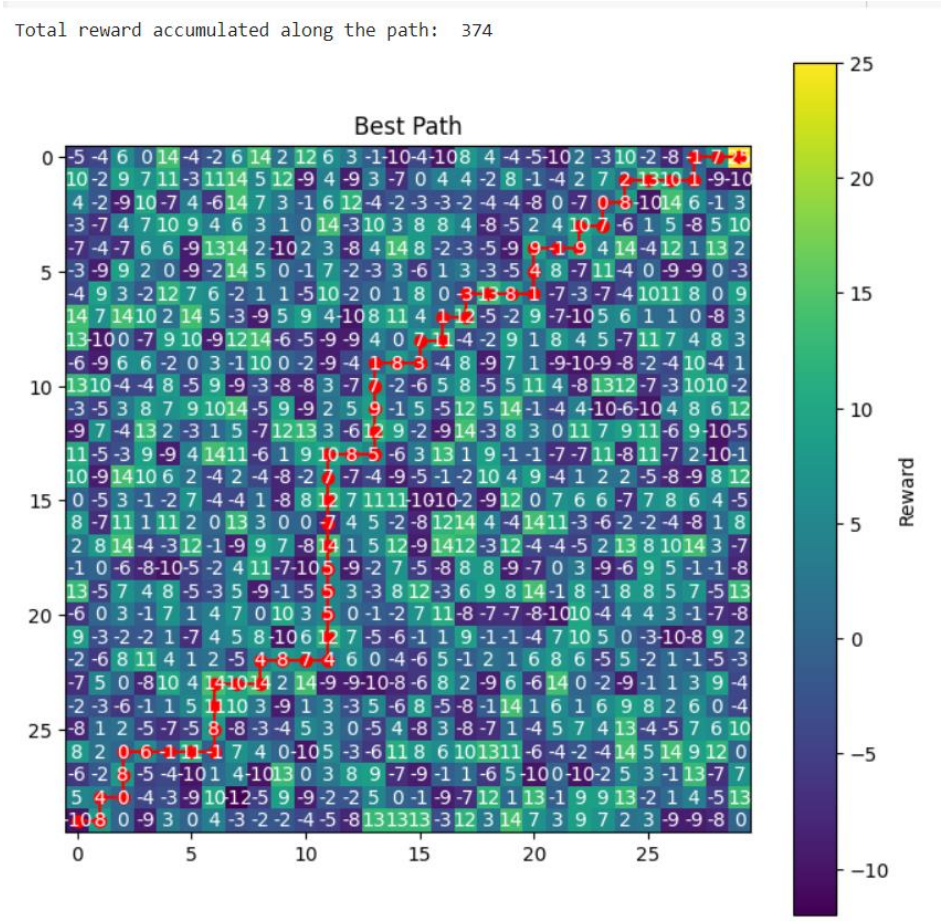

Figure 9: Result 2

```
alpha = 0.5  # Learning rate
gamma = 0.9  # Discount factor
num_episodes = 60000  # Total number of episodes
```

Figure 10: Parameter 3

```
Total reward accumulated along the path:  381
```

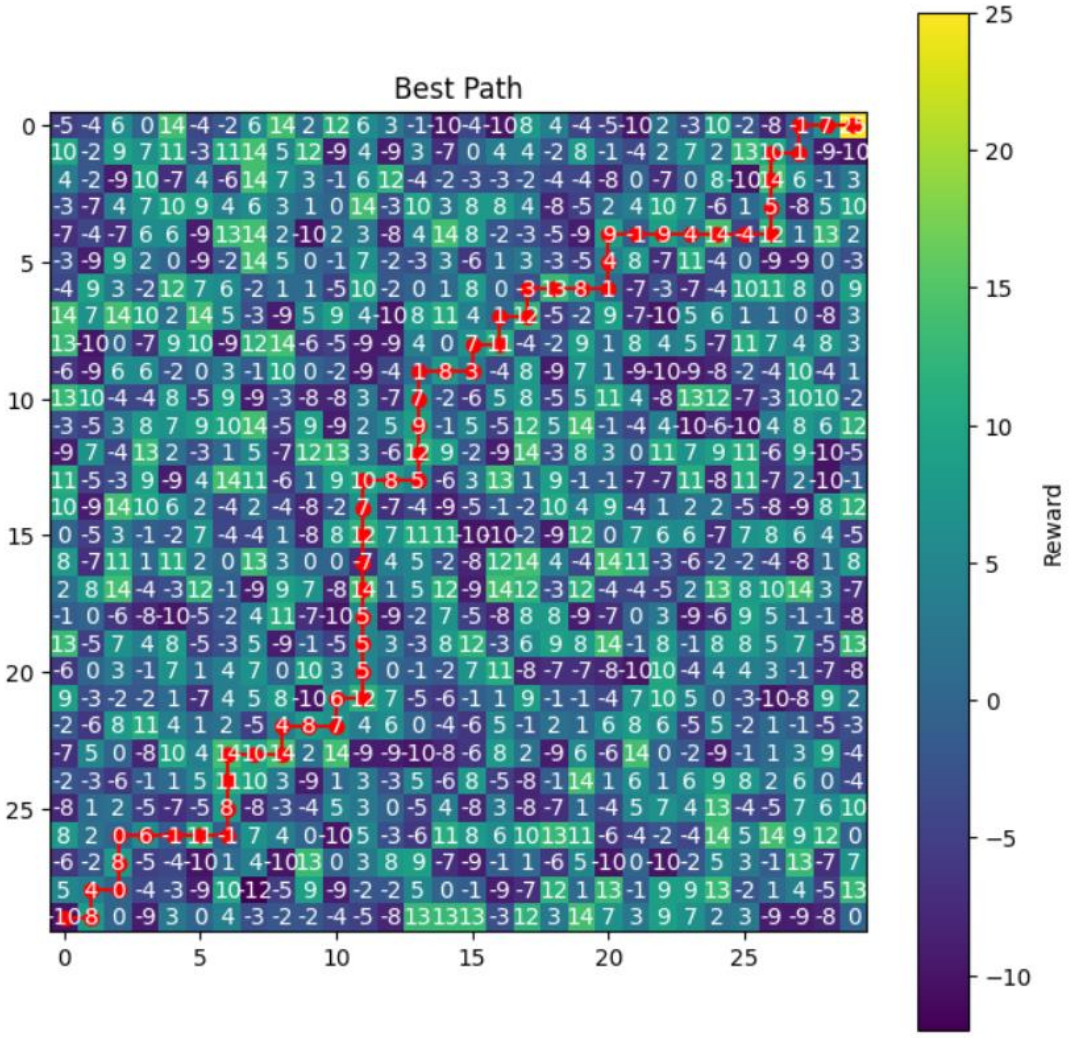

Figure 11: Result 3

```
alpha = 0.6  # Learning rate
gamma = 0.8  # Discount factor
num_episodes = 60000  # Total number of episodes
```

Figure 12: Parameter 4 (Final Parameter)

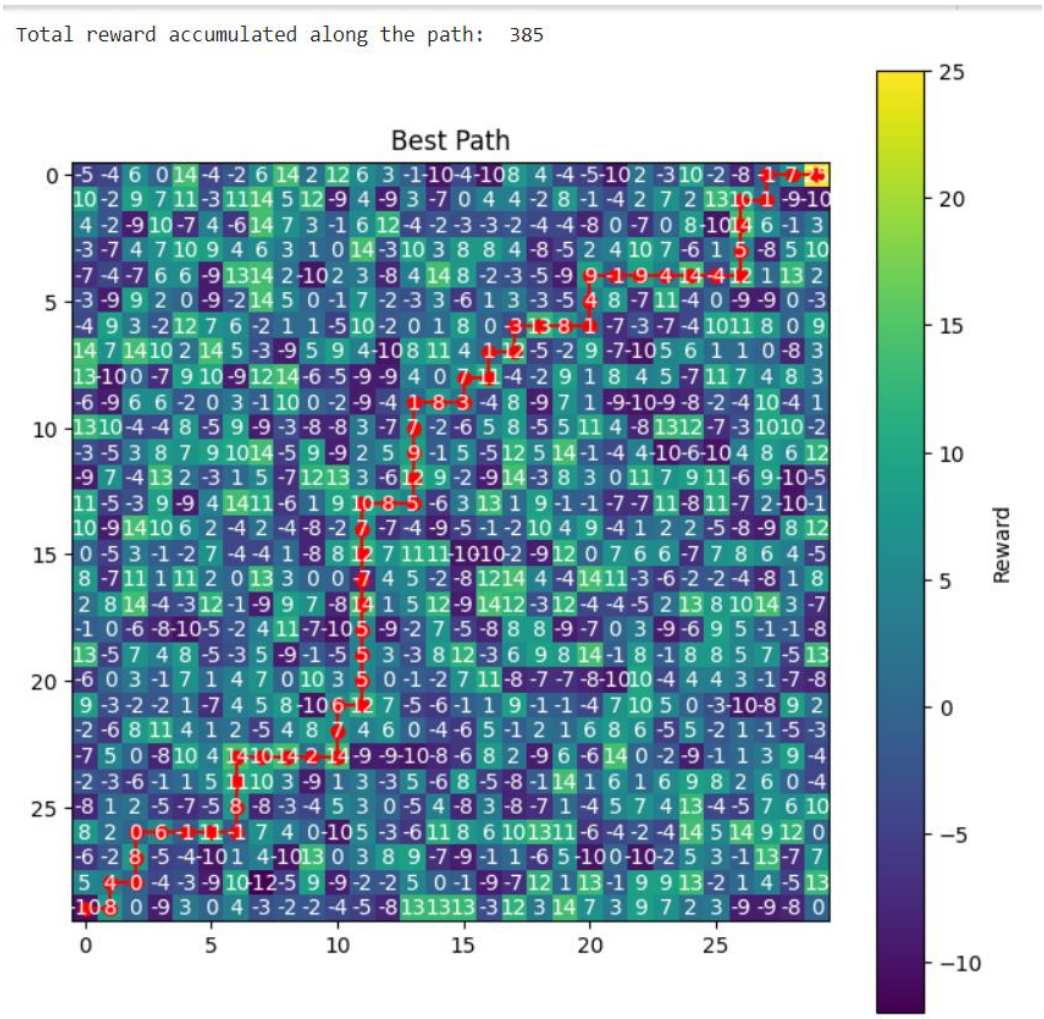

Figure 13: Result 4 (Final Parameter)

References

https://www.baeldung.com/cs/epsilon-greedy-q-learning

https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/