# Machine Learning

Dr Changjiang He, Dr Kuo-Ming Chao
Computer Science| School of Art
University of Roehampton

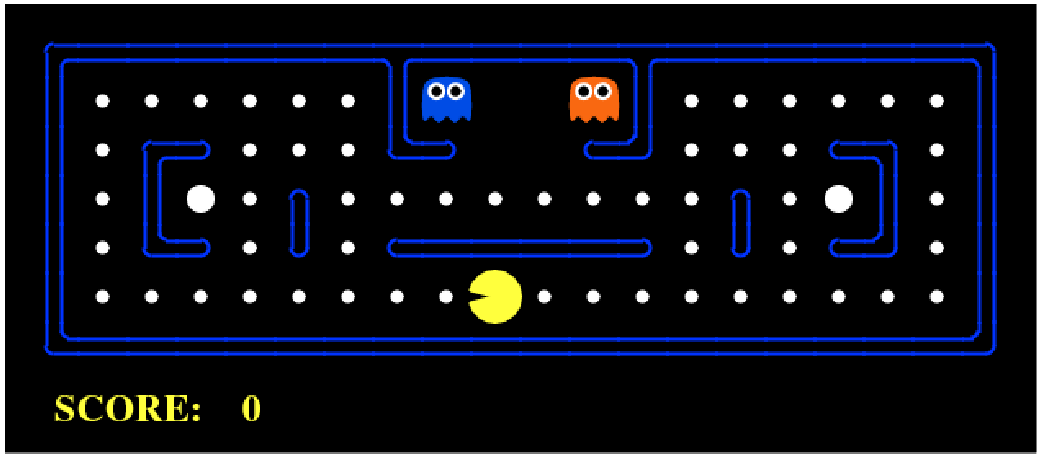
# Lesson 9.2

# Q-Learning

## What is Q-learning?

**<u>Q-learning</u>** is a commonly used model-free approach which can be used for RL problem, e.g., building a self-playing PacMan agent.

- The ‘Q’ in Q-learning stands for quality. Quality here represents how useful a given action is in gaining some future reward.



## What is Q-learning?

- Q-learning is a ***values-based*** learning algorithm.
- Value based algorithms updates the value function based on an equation (particularly Bellman equation).
- Whereas the other type, ***policy-based*** estimates the value function with a greedy policy obtained from the last policy improvement.

## What is Q-learning?

- Q-learning is an ***off-policy learner***. Means it learns the value of the optimal policy independently of the agent’s actions.

- On the other hand, an ***on-policy learner*** learns the value of the policy being carried out by the agent, including the exploration steps and it will find a policy that is optimal, taking into account the exploration inherent in the policy.

## Definition

- **Q*(s,a)** is the expected value (cumulative discounted reward) of doing a in state s and then following the optimal policy.

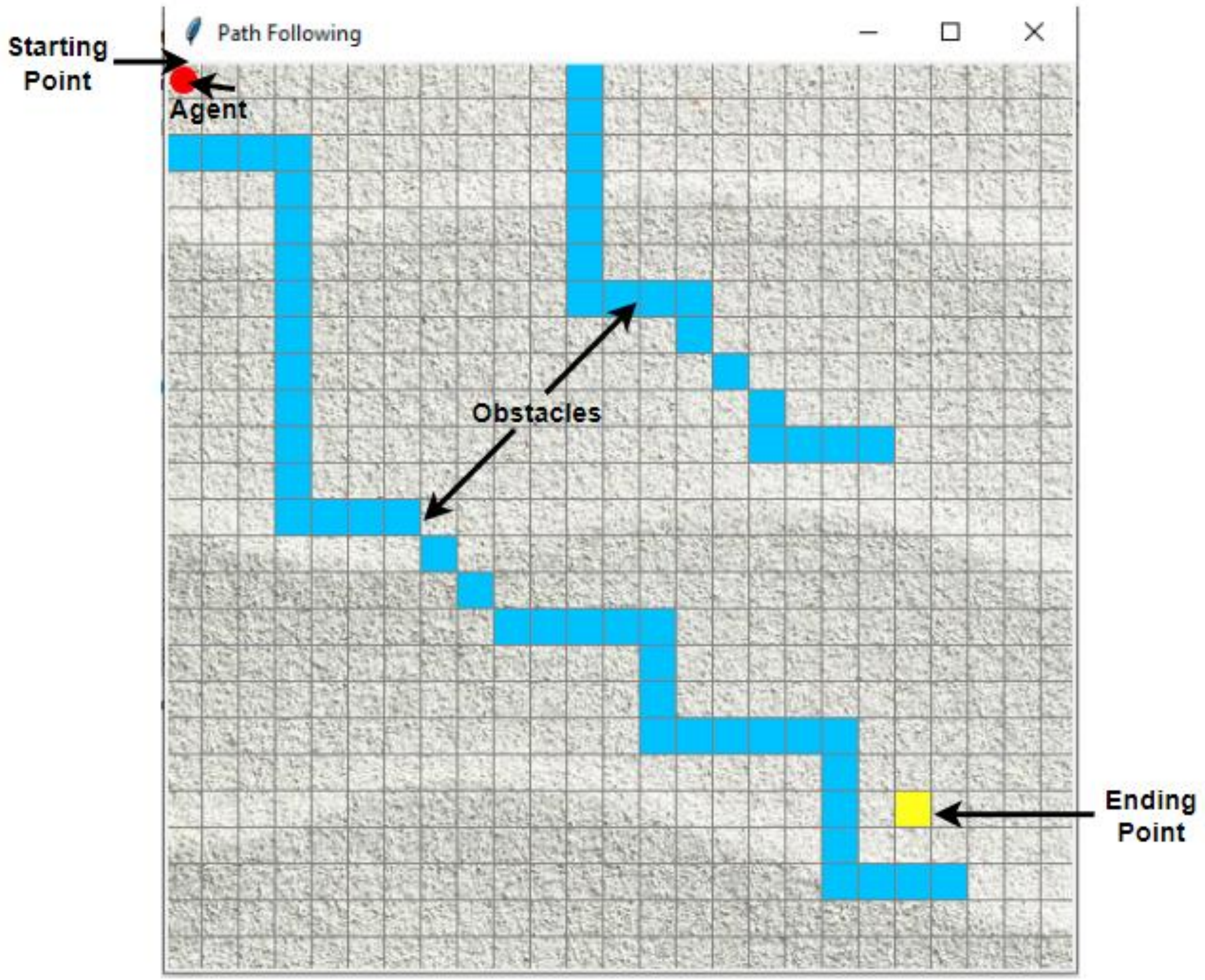
- Q-learning uses **Temporal Differences(TD)** to estimate the value of Q*(s,a). Temporal difference is an agent learning from an environment through episodes with no prior knowledge of the environment.

- The agent maintains a table of **Q[S, A]**, where **S** is the set of **states** and **A** is the set of **actions**.

- Q[s, a] represents its current estimate of Q*(s,a).

## Example

- Let’s say an agent has to move from a starting point to an ending point along a path that has obstacles.
- Agent needs to reach the target in the shortest path possible without hitting in the obstacles and he needs to follow the boundary covered by the obstacles.



## Q-table

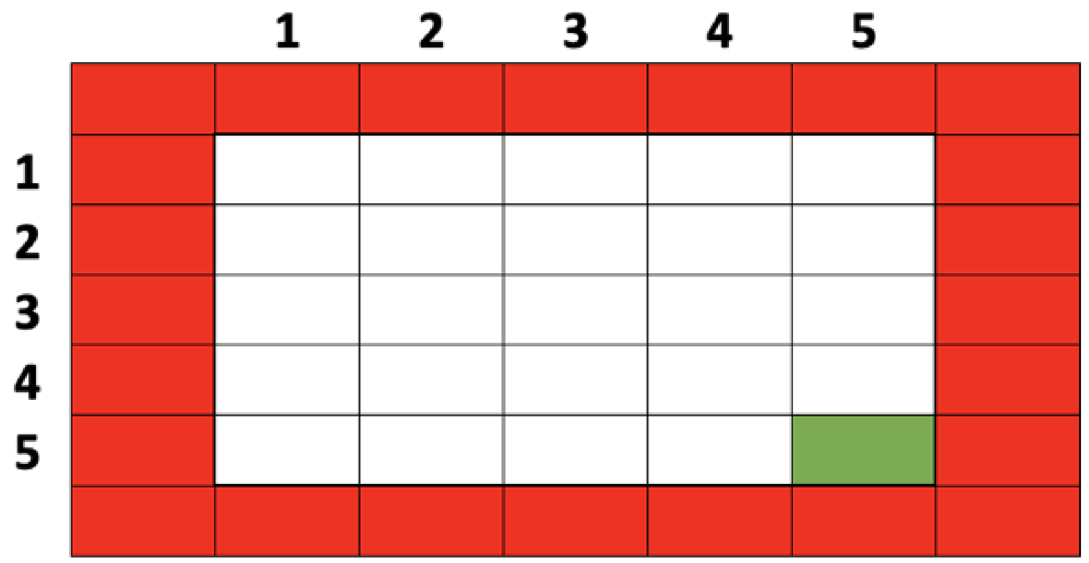
- Q-Table is the data structure used to calculate the maximum expected future rewards for action at each state. Basically, this table will guide us to the best action at each state.

- To learn each value of the Q-table, Q-Learning algorithm is used. The Q-function uses the Bellman equation and takes two inputs: state (s) and action (a).

$$\underbrace{\boxed{Q^{\pi}(s_t, a_t)}}_{\text{Q-Values for the state given a particular state}} = \underbrace{\boxed{\underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots}}_{\text{Expected discounted cumulative reward}} \underbrace{\boxed{|s_t, a_t}]}_{\text{Given the state and action}}$$

## Basic Idea

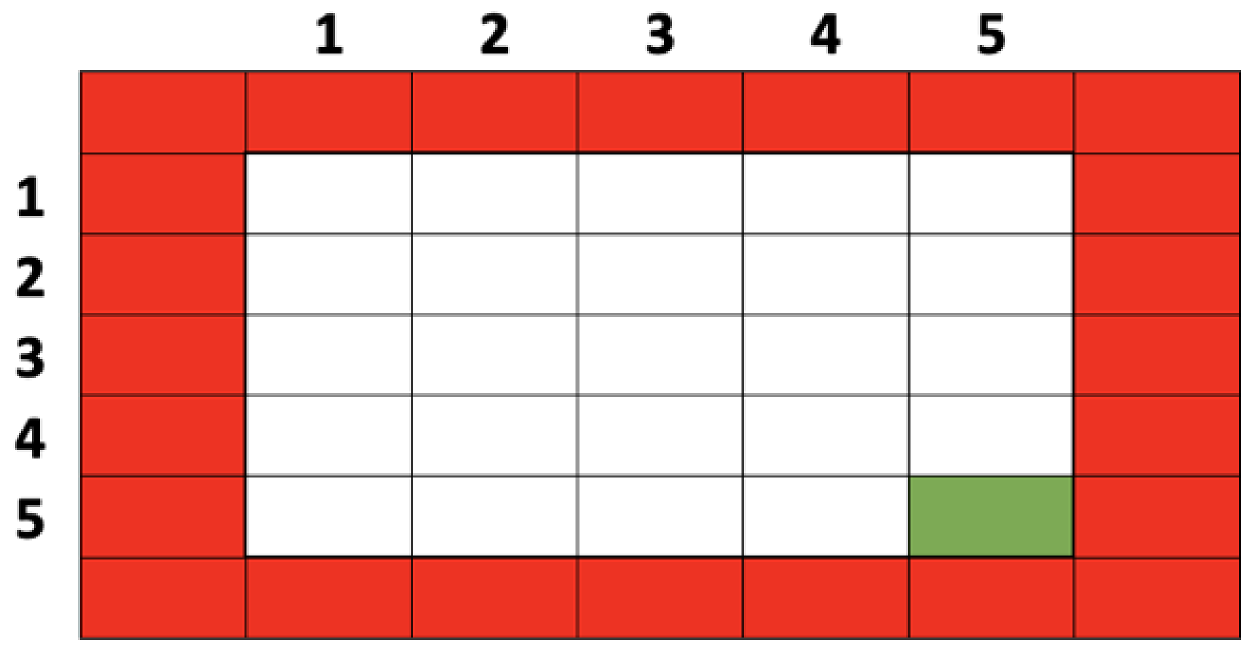
- Imagine a board with 5 rows and 5 columns — all the white cells in Figure 1. We can start in any white cell. The goal will be to travel and reach the green cell (5,5) at the bottom right corner using as little steps as possible.



## Basic Idea

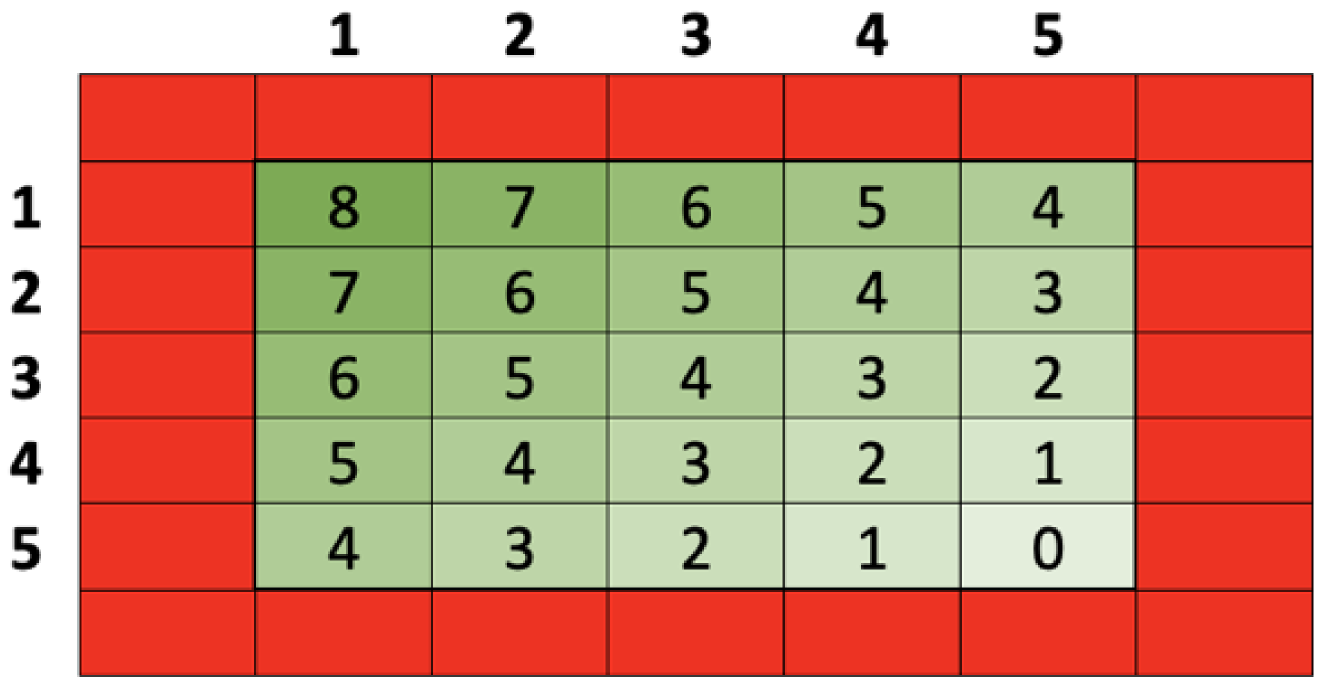
University of Roehampton London

- We can travel either Up, Down, Left or Right. We can only move one step at a time. And we cannot fall out of the board, i.e. move into the red cells, and if we do so, we die and we lose the game.



## Basic Idea

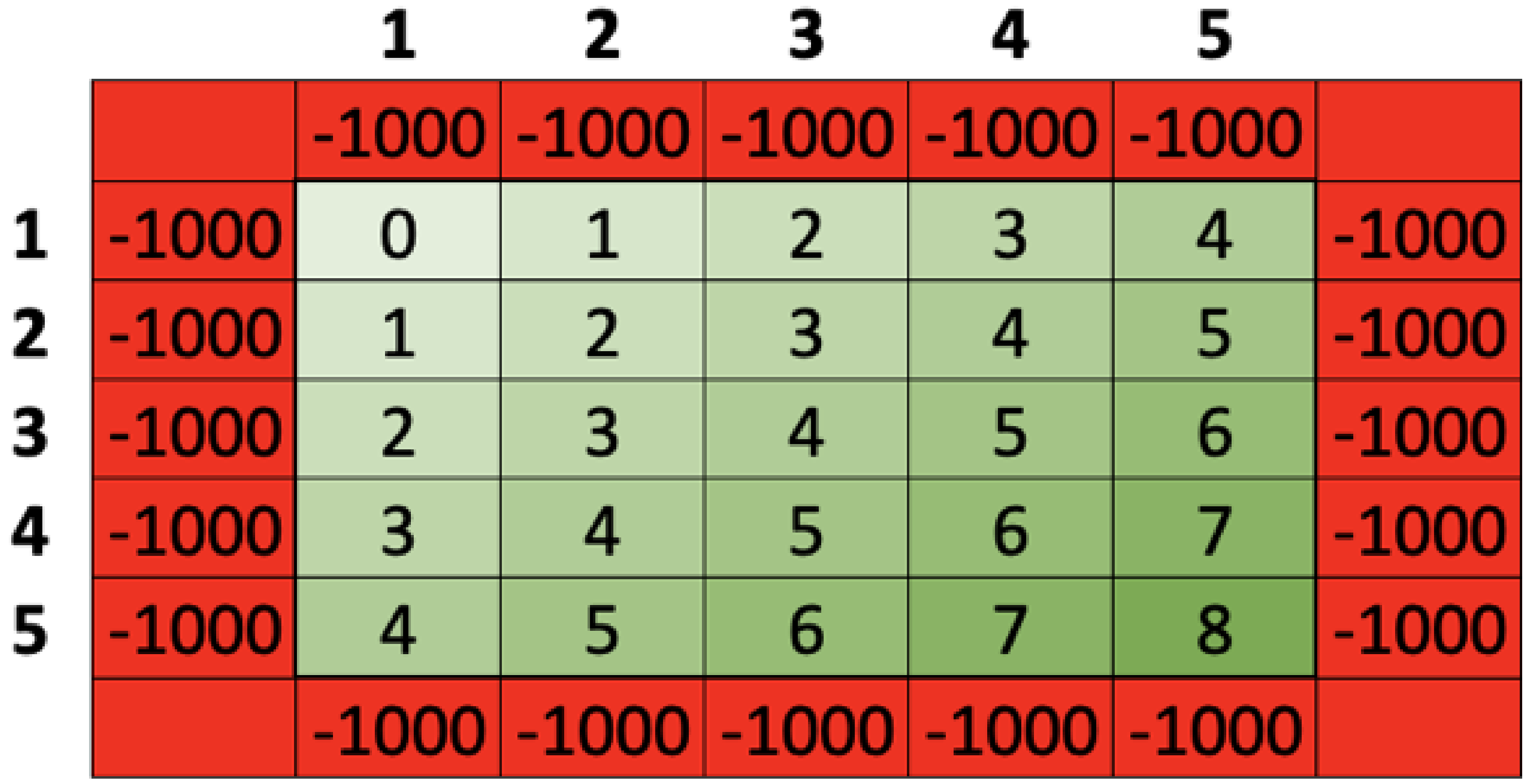
- So how can we quantify this notion of “towards” and “away”? We can start with assigning a “distance” value to all the cells.



| | | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|
| | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 |
| 1 | 🟥 | 8 | 7 | 6 | 5 | 4 | 🟥 |
| 2 | 🟥 | 7 | 6 | 5 | 4 | 3 | 🟥 |
| 3 | 🟥 | 6 | 5 | 4 | 3 | 2 | 🟥 |
| 4 | 🟥 | 5 | 4 | 3 | 2 | 1 | 🟥 |
| 5 | 🟥 | 4 | 3 | 2 | 1 | 0 | 🟥 |
| | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 | 🟥 |

## Basic Idea

- Also, going off the grid here means we have lost and so should be penalized.



| | | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|
| | | -1000 | -1000 | -1000 | -1000 | -1000 | |
| 1 | -1000 | 0 | 1 | 2 | 3 | 4 | -1000 |
| 2 | -1000 | 1 | 2 | 3 | 4 | 5 | -1000 |
| 3 | -1000 | 2 | 3 | 4 | 5 | 6 | -1000 |
| 4 | -1000 | 3 | 4 | 5 | 6 | 7 | -1000 |
| 5 | -1000 | 4 | 5 | 6 | 7 | 8 | -1000 |
| | | -1000 | -1000 | -1000 | -1000 | -1000 | |

## Basic Idea

- Let’s expand out what we have presented into an “Actions vs States” table. This expanded “Actions vs States” representation is the Q-Learning table.

| | 11 | 12 | 13 | 14 | 15 | 21 | 22 | 23 | 24 | 25 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Up** | -1000 | 0 | 1 | 2 | 3 | -1000 | 1 | 2 | 3 | 4 | ... |
| **Down** | 1 | 2 | 3 | 4 | -1000 | 2 | 3 | 4 | 5 | -1000 | ... |
| **Left** | -1000 | -1000 | -1000 | -1000 | -1000 | 0 | 1 | 2 | 3 | 4 | ... |
| **Right** | 1 | 2 | 3 | 4 | 5 | 2 | 3 | 4 | 5 | 6 | ... |

## Basic Idea

- In fact, the expected value of total rewards, the Q-value, is really the expected value of the total reward over all possible successive steps, starting from the current state.

- And what we posted here, really is just the reward based on the distance of that particular cell, i.e. “immediate reward”.

| | 11 | 12 | 13 | 14 | 15 | 21 | 22 | 23 | 24 | 25 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Up** | -1000 | 0 | 1 | 2 | 3 | -1000 | 1 | 2 | 3 | 4 | ... |
| **Down** | 1 | 2 | 3 | 4 | -1000 | 2 | 3 | 4 | 5 | -1000 | ... |
| **Left** | -1000 | -1000 | -1000 | -1000 | -1000 | 0 | 1 | 2 | 3 | 4 | ... |
| **Right** | 1 | 2 | 3 | 4 | 5 | 2 | 3 | 4 | 5 | 6 | ... |

## Q-learning

- Now let’s say we know the expected reward of each action at every step. This would essentially be like a cheat sheet for the agent! Our agent will know exactly which action to perform.

- It will perform the sequence of actions that will eventually generate the maximum total reward. This total reward is also called the Q-value and we will formalise our strategy as:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

## Q-learning

- The above equation states that the Q-value yielded from being at state *s* and performing action *a* is the immediate reward r(s,a) plus the highest Q-value possible from the next state *s'*.

- Gamma here is the discount factor which controls the contribution of rewards further in the future.

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

## Q-learning

- Q(s’,a) again depends on Q(s”,a) which will then have a coefficient of gamma squared.
- So, the Q-value depends on Q-values of future states as shown here:

$$Q(s,a) \rightarrow \gamma \mathrm{Q}(\mathrm{s}',\mathrm{a}) + \gamma^2 Q(s'',a) \ldots \ldots \ldots \gamma^n Q(s''^{\ldots n},a)$$

- Adjusting the value of gamma will diminish or increase the contribution of future rewards.

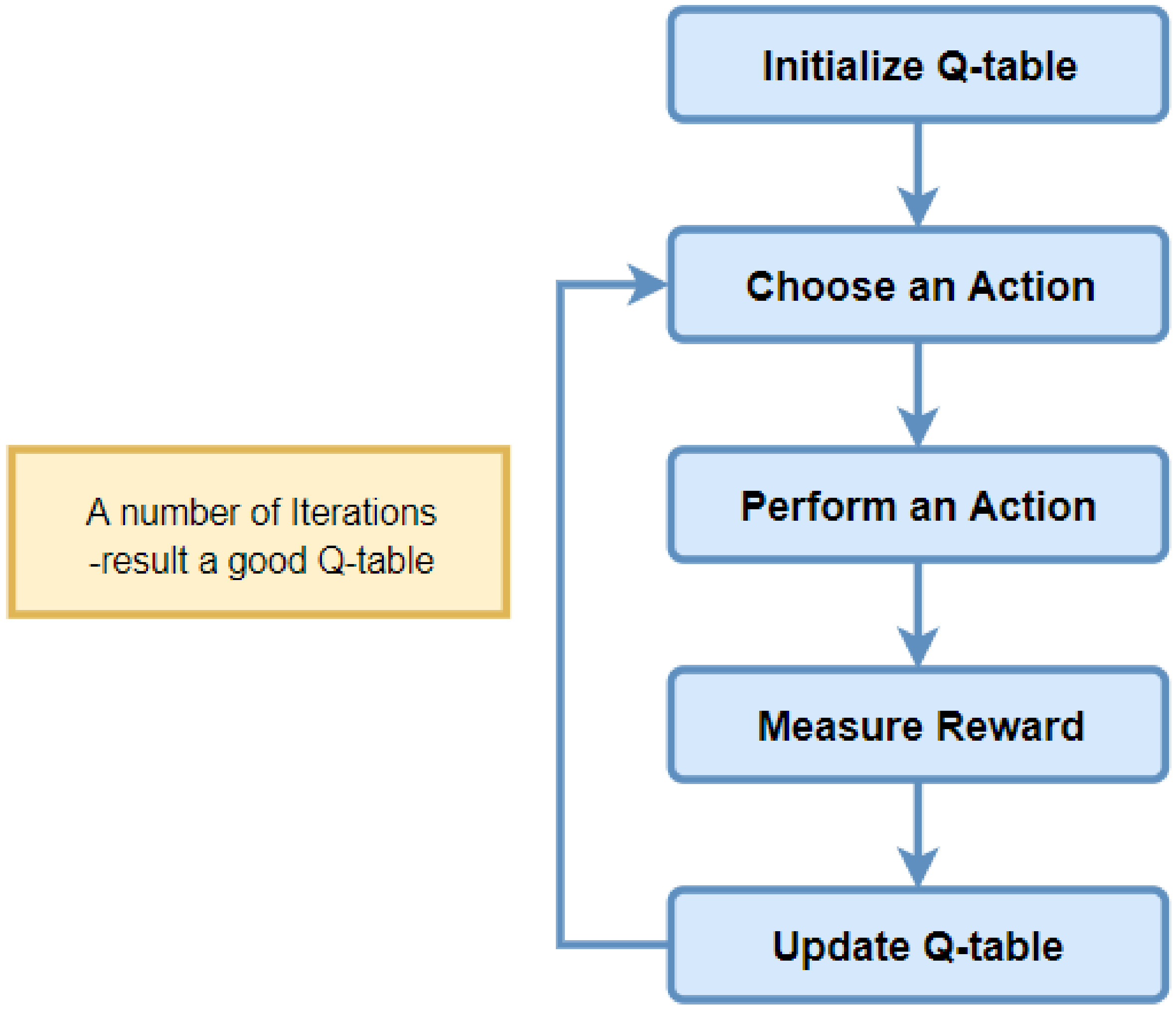
## Q-learning

- Since this is a recursive equation, we can start with making arbitrary assumptions for all q-values. With experience, it will converge to the optimal policy. In practical situations, this is implemented as an update:

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

- where alpha is the learning rate or step size. This simply determines to what extent newly acquired information overrides old information.

## Q-table


Initialize Q-table
Choose an Action
Perform an Action
Measure Reward
Update Q-table
A number of Iterations
-result a good Q-table