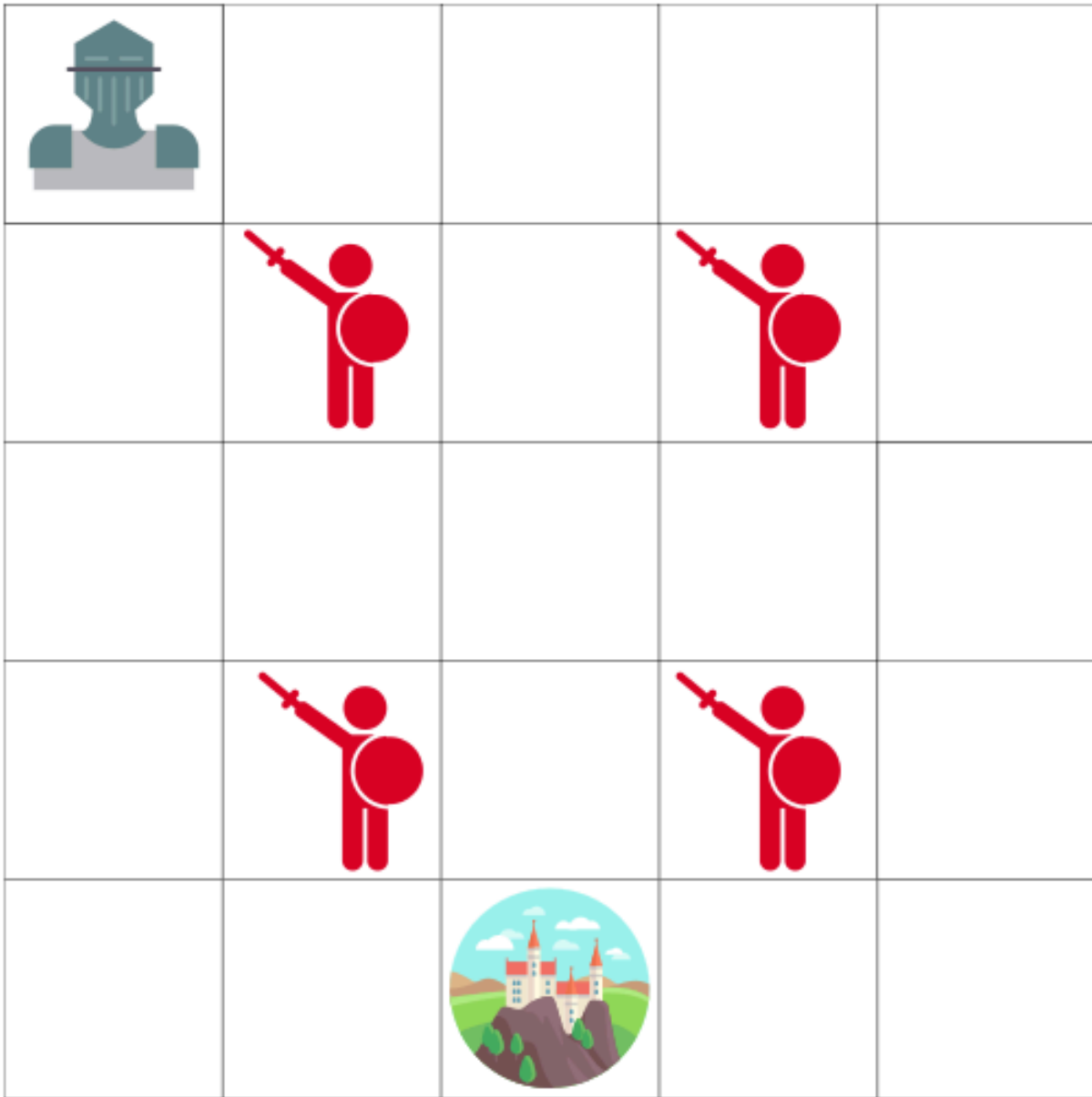


## 2- Diving into Reinforcement Learning with Q-Learning

### Big Picture



Player is a knight (top left) that must avoid landing on a square with a red guard. Player wins by reaching the Princess in the castle (bottom middle). Player moves one tile at a time, and wants to reach castle via the fastest route possible. This all can be quantified with a point-system:

- Lose 1 point at each step (that way, agent is encouraged to reach princess asap)
- Touching enemy loses 100 points, and the game (episode) ends

- If castle is reached, you win with +100 points

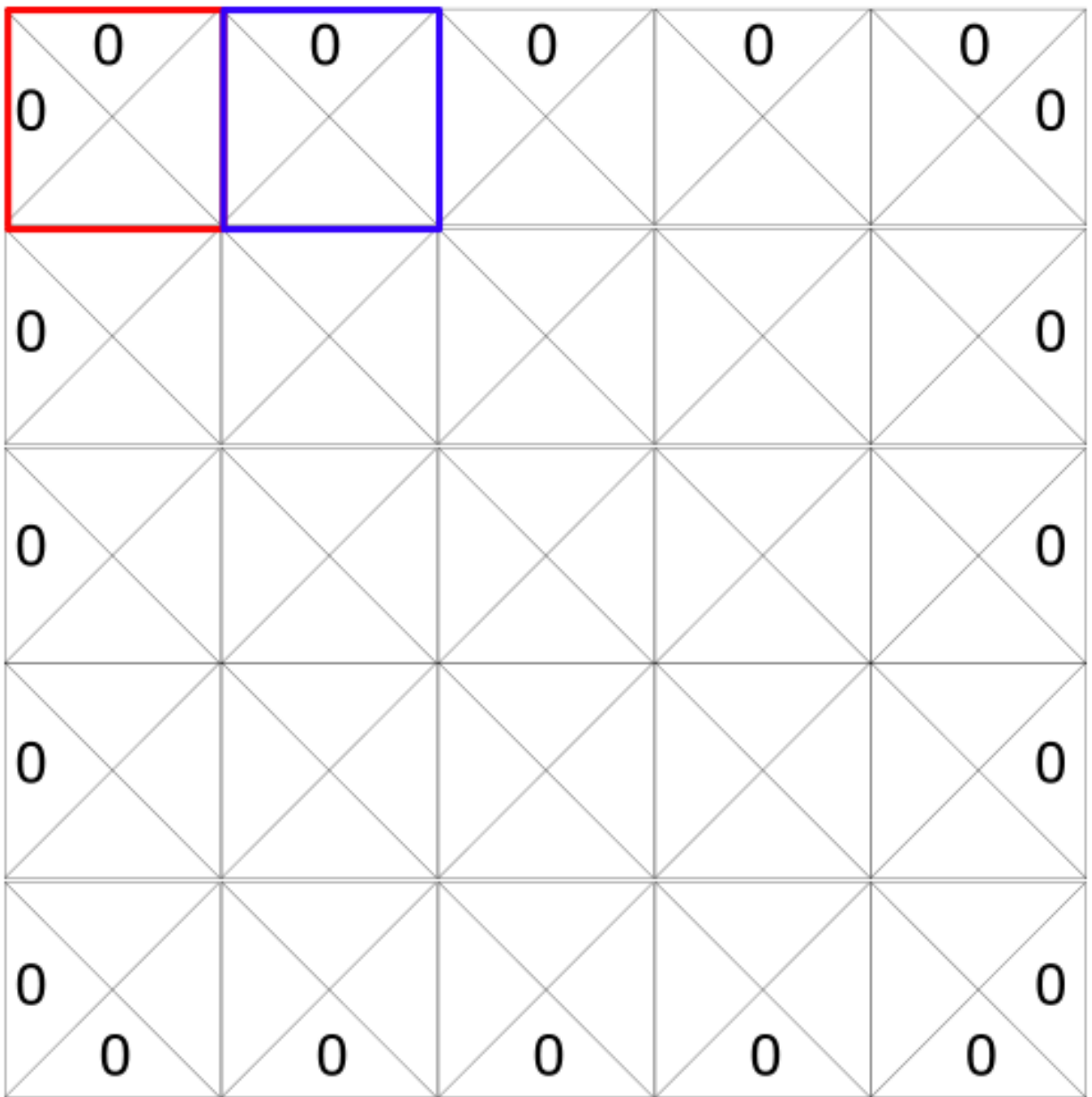
## **Naive Approach**

What if we simply make our agent try to go to each tile and then color each tile red (causes loss) or green (safe). Then, just tell agent to only take green tiles. Issue with this is that there is no sense of which tile is the best to take when green tiles are next to each other. In fact, agent could get stuck in an infinite loop.

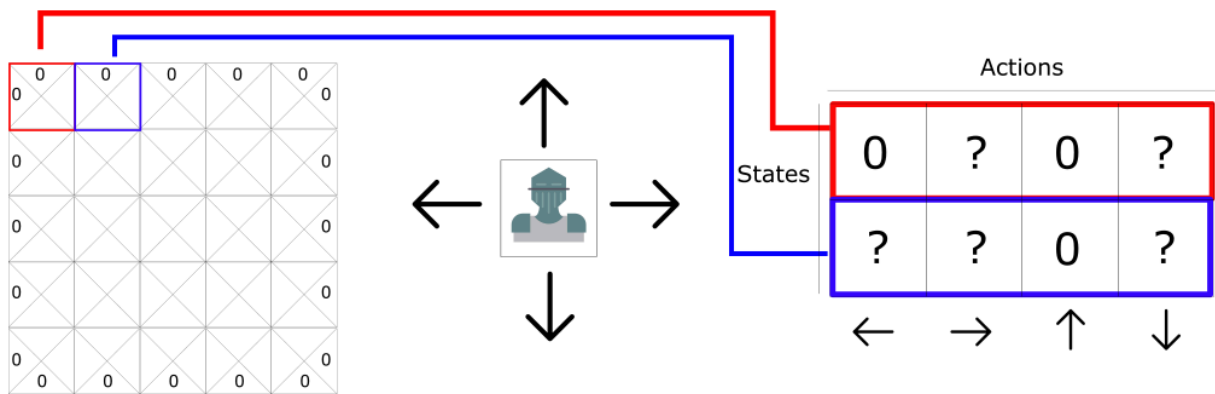
## **Introducing the Q-table**

A smarter strategy is to create a table where we will calculate maximum expected future reward, for each action at each state. This way, we will have a sense for what action is the best to take for each state.

Each state (tile on the board) allows four possible actions (moving left, right, up, or down). At first this can be visualized as a grid, where each tile is divided into four possible actions. 0 means that a particular action is impossible (like moving out of bounds).



This grid can then be transformed into a table:



This is called a Q-table ("Q" for quality of the action). The columns will be the

four actions, and the rows will be the states. The value of the action/state cell will be the maximum expected future reward for that given state and action.

Each Q-table score will be the maximum expected future reward if that action is taken at that state with the **best policy given**.

Here, we say "best policy given" because we don't actually implement one. Instead, we improve the Q-table to always choose the best action.

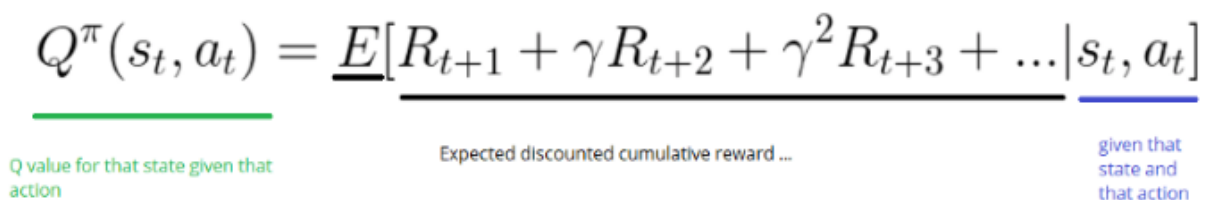
In this sense, Q-table is basically a cheat sheet.

Now, we have to learn each value of the Q-table using the **Q Learning Algorithm**.

### Q-Learning Algorithm: learning the Action Value Function

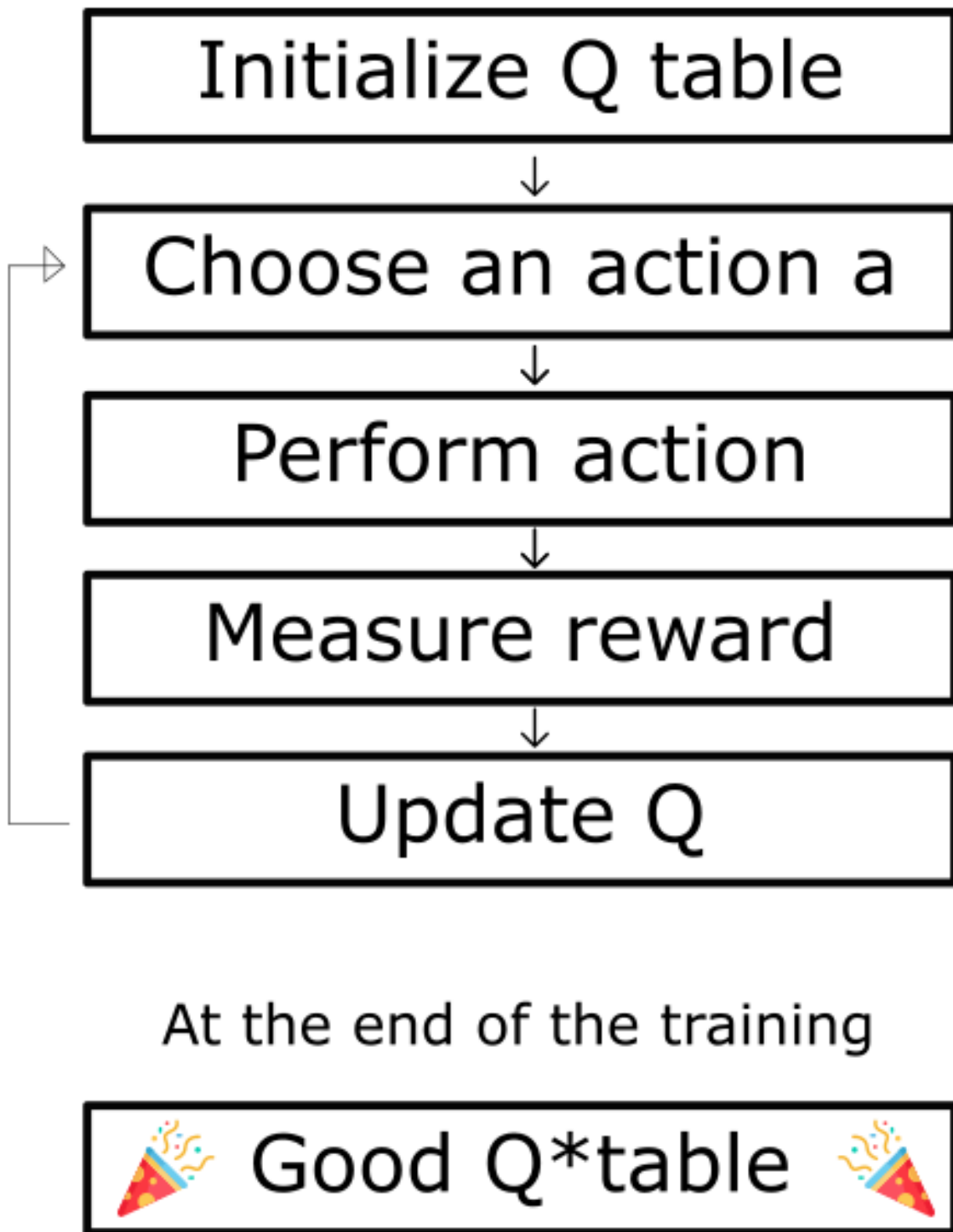
The action value function (or "Q-function") is a multivariable function on the state and action. It returns the expected future reward of that action at that state:

$$Q^{\pi}(s_t, a_t) = \underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$$



Essentially, the Q-function looks through the Q-table to find the line associated with the state, and the column associated with our action. Then it returns the Q value from the matching cell (which is the expected future reward).

Before exploring the environment, the Q-table gives the same arbitrary fixed value (usually 0). As we explore, Q-table iteratively updates and will give a better approximation. The Q-table is improved with the **Bellman Equation**.



1. Initialize Q-values ( $Q(s,a)$ ) arbitrarily for all state-action pairs (usually initialize to 0)
2. For life or until learning stops:
  1. Choose action ( $a$ ) in the current world state ( $s$ ) based on current Q-value estimates ( $Q(s,*)$ ).

2. Take the action (a) and observe the outcome state (s') and reward (r)
3. Update following Bellman Equation

$$Q(s, a) := Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

Above is the basic pseudo-code for the Q-learning algorithm.

Step 1: Initialize Q-values

Build a Q-table of m x n columns and rows. m=number of actions, n=number of states. Initialize all values to 0:

Actions					
States					
	0	0	0	0	
	0	0	0	0	
	0	0	0	0	
	■ ■ ■				
	0	0	0	0	

Step 2: Repeat steps 3-5 for life or until learning is stopped:

Exit step 2 once maximum number of episodes is reached (that max is specified by user) or until user manually stops training.

Step 3: Choose an Action

Choose an action  $a$  in the current state  $s$  based on the current Q-value estimates.

But if we initialize entire table to 0 in the beginning, what action do we take?

This is where exploration/exploitation comes in to play. We need to explore so our Q-table is useful.

Idea: In the beginning of training, use the **epsilon greedy strategy**:

- Define an exploration rate "epsilon" which is set to 1 at the beginning. This is essentially the rate of steps we'll do randomly.
  - In the beginning, this value must be at its highest value, because we want to explore as much as possible to learn and make Q-table useful.
- Generate a random number. If this number  $>$  epsilon, do "exploitation" (using what we already know to select best action at each step). Else, we'll do exploration.
- Idea is that we must have a big epsilon at the beginning of the training of the Q-function. Then, reduce it progressively over time as agent's Q-table becomes better (agent is more confident at estimating Q-values).



Steps 4-5: Evaluate

Take action  $a$  and observe the outcome states  $s'$  and reward  $r$ . Now update

function  $Q(s,a)$ .

Bellman Equation used to update  $Q(s,a)$  uses current  $Q$  value, learning rate, discount rate, reward for taking action  $a$  at state  $s$  and the maximum expected future reward given the new state  $s'$  and all possible actions at that new state:

$$NewQ(s, a) = \underbrace{Q(s, a)}_{\text{Current Q value}} + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{R(s, a)}_{\text{Reward for taking that action at that state}} + \underbrace{\gamma}_{\text{Discount rate}} \underbrace{\max_{a'} Q'(s', a')}_{\text{Maximum expected future reward given the new } s' \text{ and all possible actions at that new state}} - \underbrace{Q(s, a)}_{\text{Current Q value}}]$$

So, our new  $Q$  value = Our current  $Q$  value +  $lr * [Reward + discount\ rate * (highest\ Q\ value\ between\ possible\ actions\ from\ the\ new\ state\ s') - Current\ Q\ value]$

Walking through an example:



We are the mouse. The one cheese tile is +1 points. The two cheese tile is +2 points. The big pile of cheese is +10 points and the end of the episode. Eating rat poison is -10 points, the end of the episode.

Step 1: Init  $Q$ -table:



	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Step 2: Choose an Action

In beginning, we know nothing, but have big epsilon rate, so we take random move. Let's say we move right:



	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Now we can update Q-value of being at the start and going right, using Bellman Equation.

Step 4-5: Update Q-function:

$$NewQ(s, a) = \underbrace{Q(s, a)}_{\text{New Q value for that state and that action}} + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{R(s, a)}_{\text{Reward for taking that action at that state}} + \underbrace{\gamma}_{\text{Discount rate}} \underbrace{\max Q'(s', a')}_{\text{Maximum expected future reward given the new s' and all possible actions at that new state}} - \underbrace{Q(s, a)}_{\text{Current Q value}}]$$

$$NewQ(start, right) = Q(start, right) + \alpha[\Delta Q(start, right)]$$

$$\Delta Q(start, right) = R(start, right) + \gamma \max Q'(1cheese, a') - Q(start, right)$$

$$\Delta Q(start, right) = 1 + 0.9 * \max(Q'(1cheese, left), Q'(1cheese, right), Q'(1cheese, down)) - Q(start, right)$$

$$\Delta Q(start, right) = 1 + 0.9 * \underline{0} - \underline{0} = 1$$

$$NewQ(start, right) = 0 + 0.1 * 1 = 0.1$$

- First calculate change in Q value  $\Delta Q(start, right)$
- Then add initial Q value to the change in Q value multiplied by a learning rate.

Here, learning rate is how quickly the network abandons the earlier value for the new one. If our learning rate is 1, then new estimate will be the new Q-value.

Now do this repeatedly until learning is stopped.