AI Essentials

Reinforcement Learning

Ir. Hennion Domien

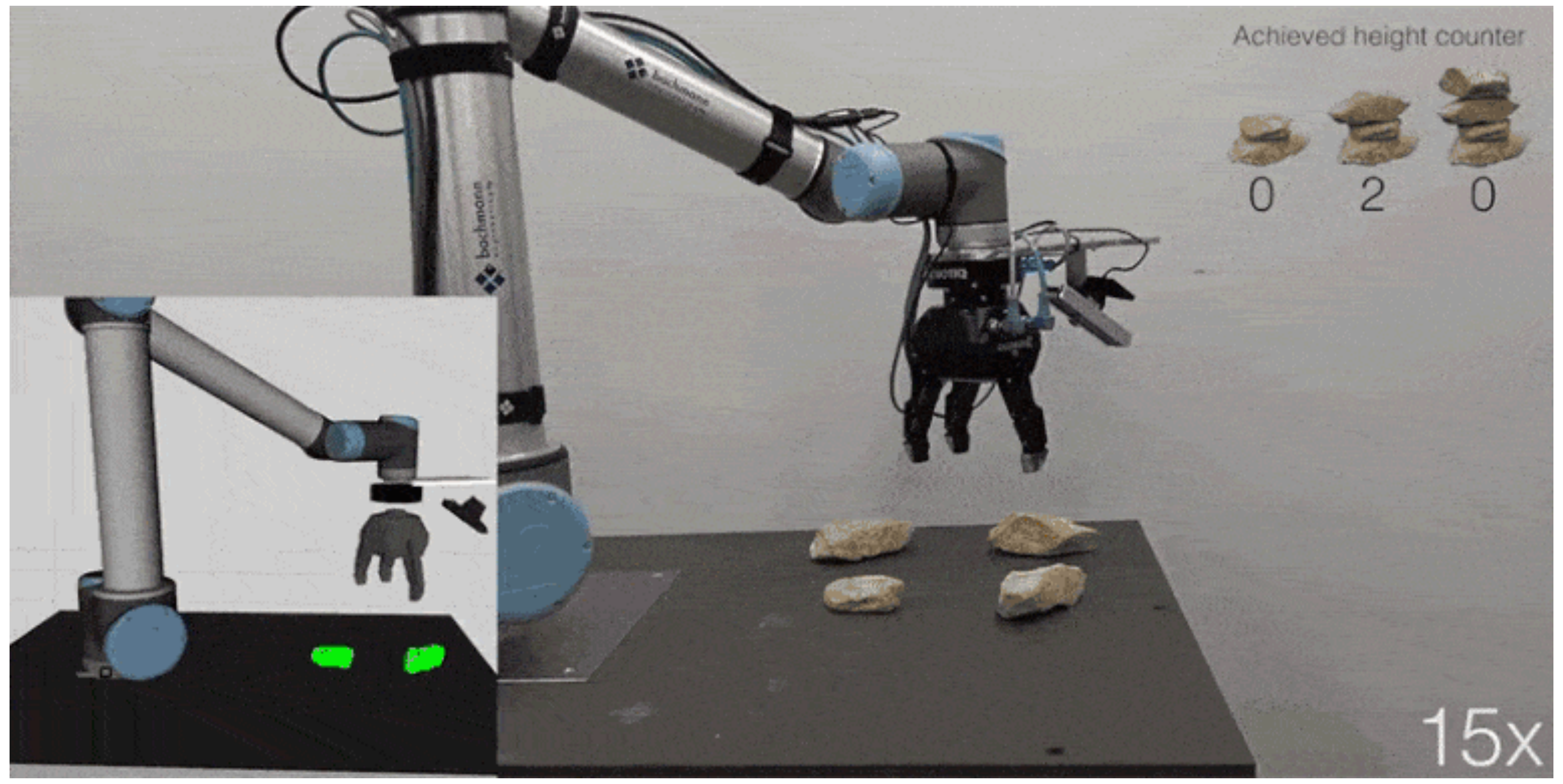
Reinforcement Learning

- Reinforcement Learning is an aspect of Machine learning where an agent learns to behave in an environment, by performing certain actions and observing the rewards/results which it get from those actions.

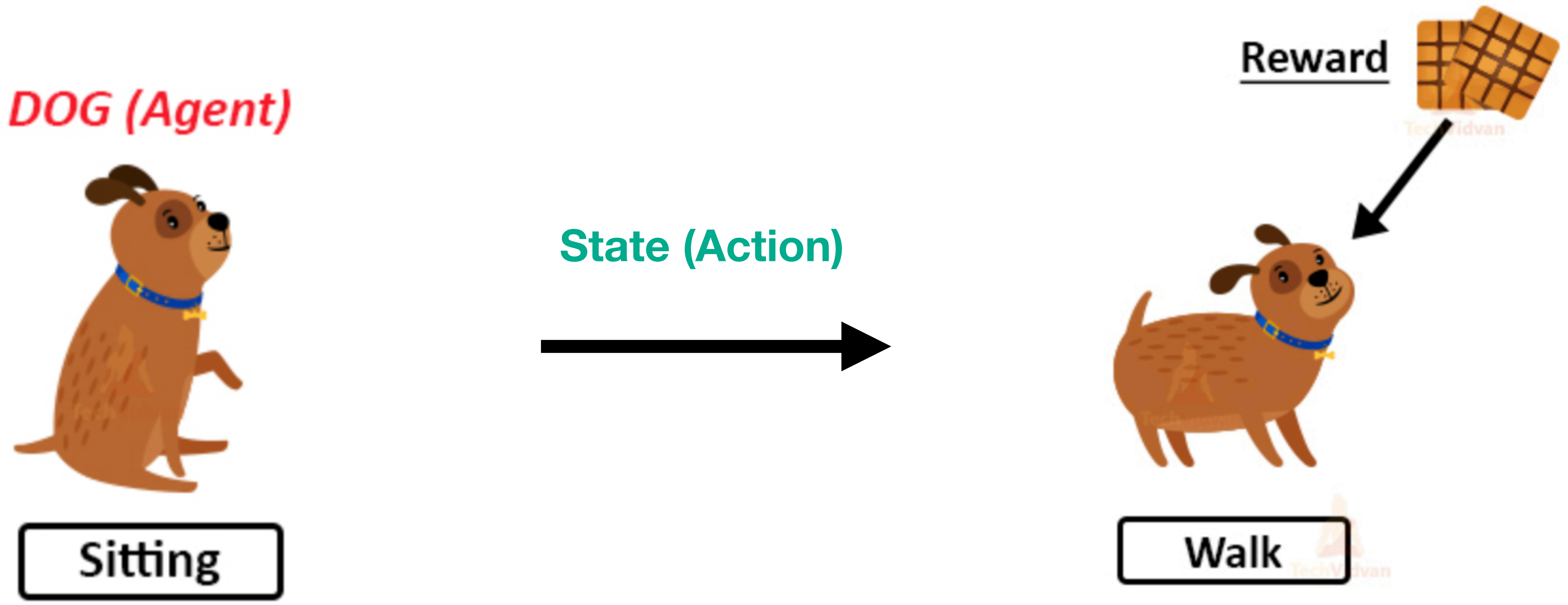
- Robotics Arm Manipulation

- 2016: Google Deep Mind beating Lee Sedol, an Alpha Go Player

- 2018: Open AI team beats world champion DOTA 2

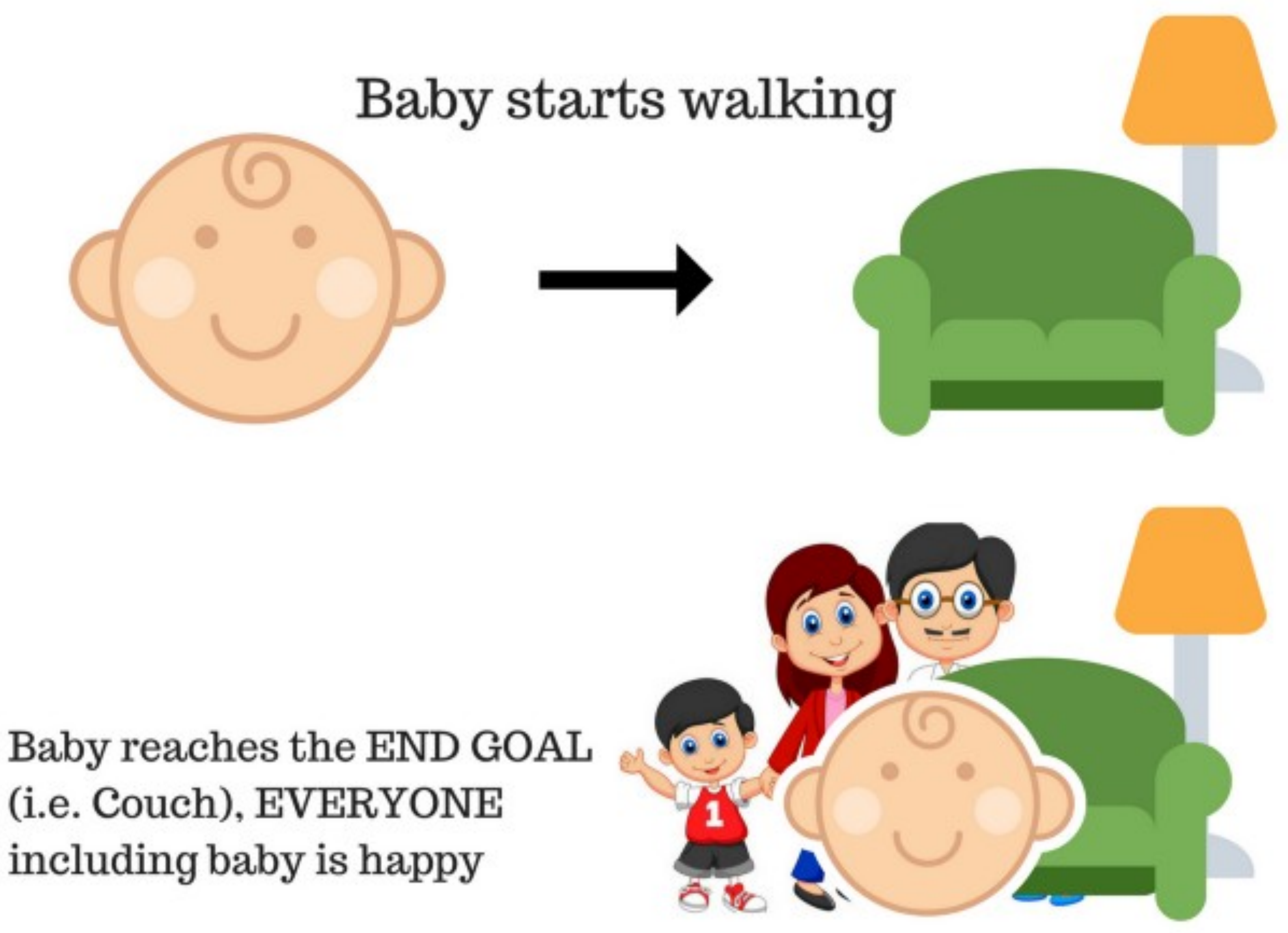


Reinforcement Learning



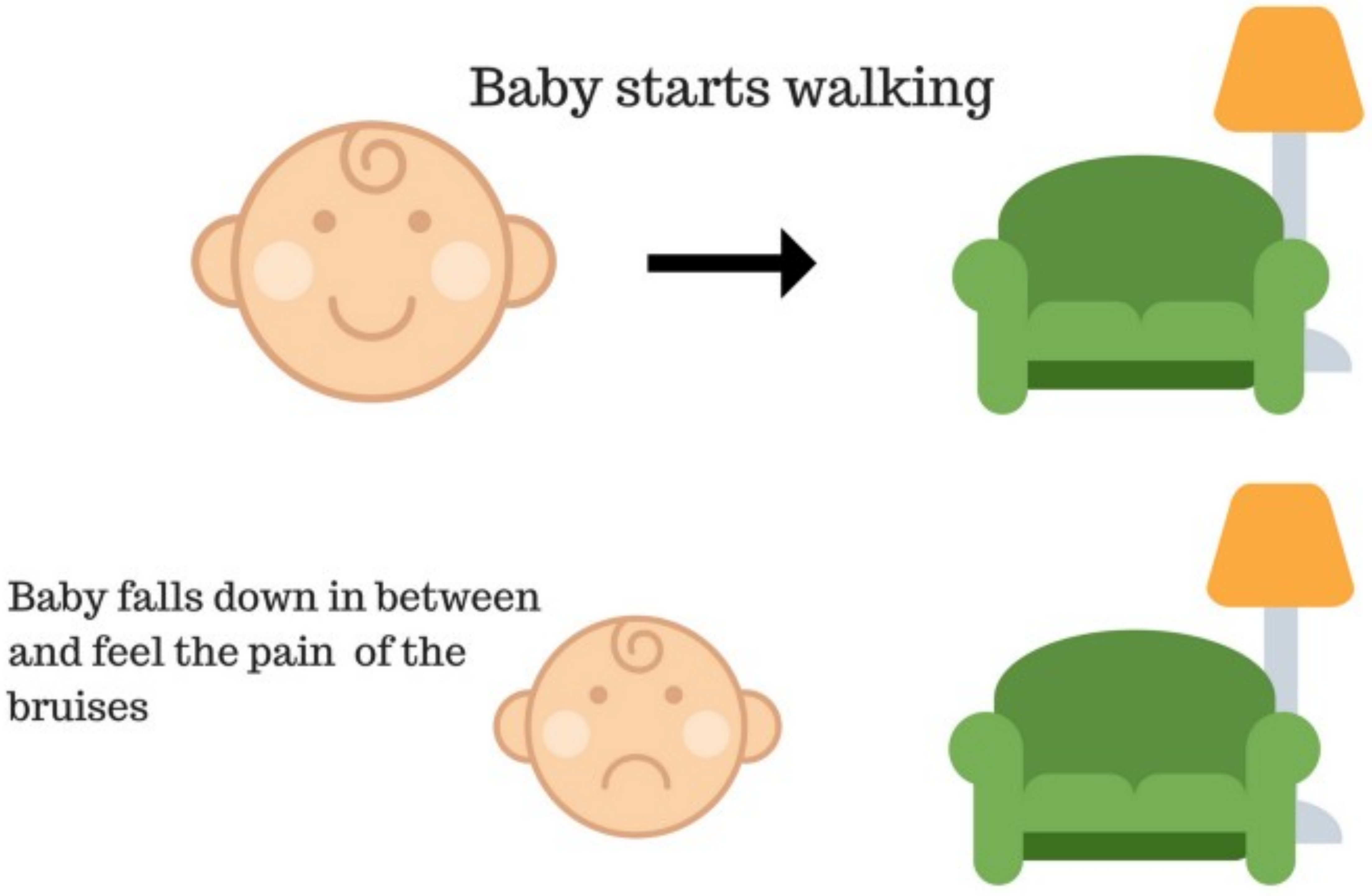
Reinforcement Learning

- The baby is happy and receives appreciation from her parents. It’s positive
- The baby feels good (Positive Reward +n)



Reinforcement Learning

- The baby gets hurt and is in pain. It's negative
- The baby cries (Negative Reward -n)

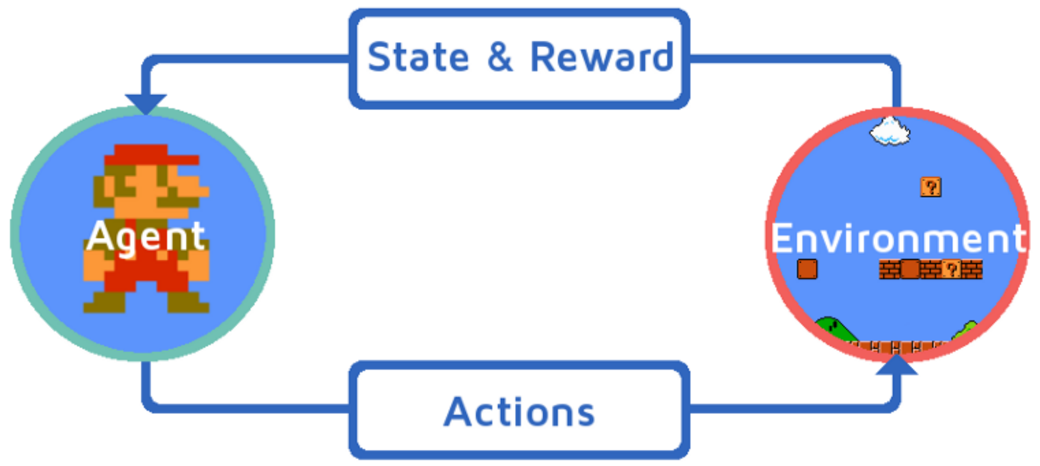


Reinforcement Learning

Super Mario Bros

Reinforcement Learning

Super Mario Bros



- The **RL Agent** receives **state S^0** from the **environment** i.e. Mario. The starting state.

- Based on that **state S^0**, the RL agent takes an **action A^0**. The actions is random.

- Now, the environment is in a **new state S^1** (new frame from Mario or the game engine)

- Environment gives some **reward R^1** to the RL agent. A +1 because the agent is not dead yet. Or a -1 if the agent is dead.


MARIO
000000
×00
WORLD
1-1
TIME
400

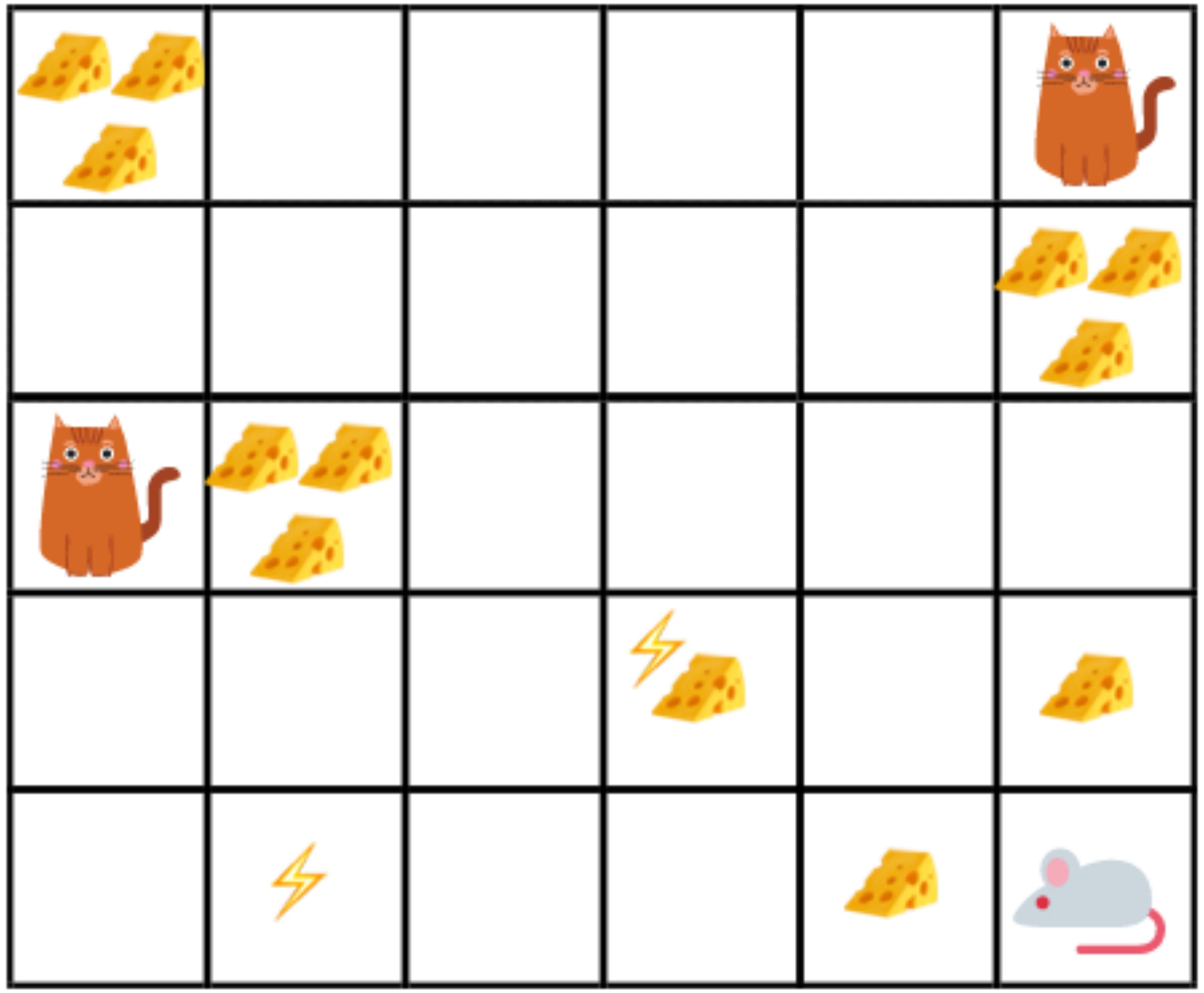
Reward Maximization

- The goal of an RL agent is reward maximization
- Choose the best possible action in order to maximize the reward
- The cumulative rewards of each time step:

$$G_t = \sum_{k=0}^{T} R_{t+k+1}$$

Reward Maximization

- Goal: eat the maximum amount of cheese before being eaten by the cat or getting an electricity shock
- The reward near the cat or the electricity shock, even if it is bigger (more cheese), will be discounted
- This is because of the **uncertainty factor**



Reward Maximization

With a discount rate

- A discount rate between 0 and 1
- Lager γ —> smaller discount
- Smaller γ —> larger discount

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \, where \gamma \in [0, 1)$$

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots$$

Sigma (Sum up)

Discount rate

Rewards received at each state

Expanded form of the Equation

Task in RL

- A **task** is a single instance of a reinforcement learning problem
- 2 types: Continuous and episodic tasks

Task in RL

Continuous tasks

- Tasks that continue forever
- The agent learns to choose the best actions and simultaneously interacts with the environment
- There is no starting and end state
- The agent keeps running until we decide to stop it
- Ex.: Crypto trading bots

Task in RL
Episodic tasks

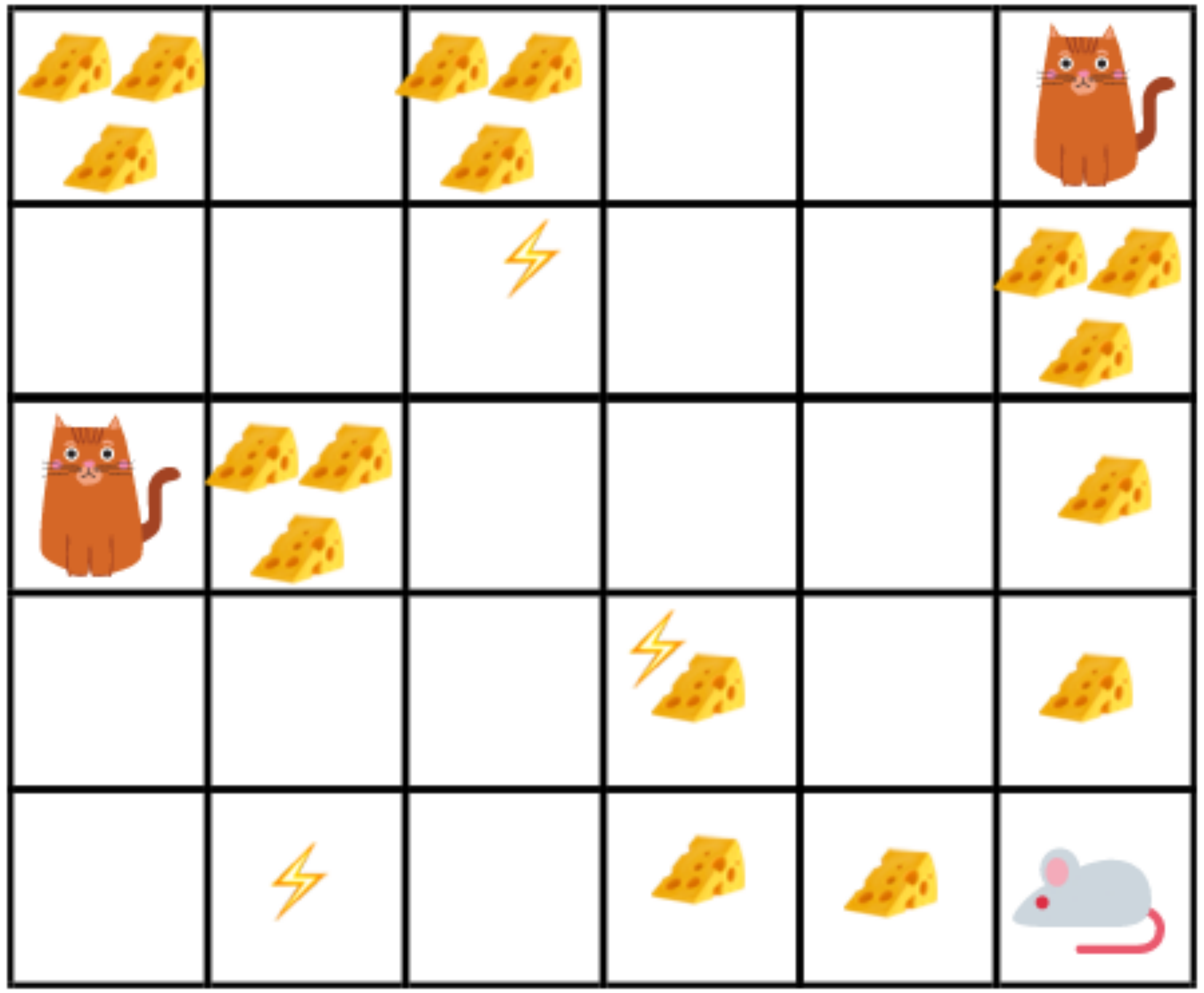
- There is a start and end point (terminal state)
- This creates an **episode**: a list of states **S**, Actions **A** and Rewards **R**
- Ex.: RL in Mario, DOTA, Counter strike, …

Exploration and exploitation

- **Exploration** is all about finding more information about an environment
- **Exploitation** is exploiting already known information to maximize the rewards

Exploration and exploitation trade-off

- The mouse can a good amount of small cheese (+1 each). But there is a big sum of cheese at the top (+100).
- Staying at the nearest reward —> the mouse will exploit
- If the mouse does some exploration it can find bigger rewards



Approaches to solve RL problems

Policy-based approach

- Learn a policy which we need to optimize
- The policy defines how the agent behaves

$$a = \pi(s)$$

a : Actions

s : State

π: Policy Func.

- 2 types of policies:
  - **Deterministic**: a policy at a given state (s) will always return the same action (a)
  - **Stochastic**: a distribution of probability over different actions

Approaches to solve RL problems

Value based approach

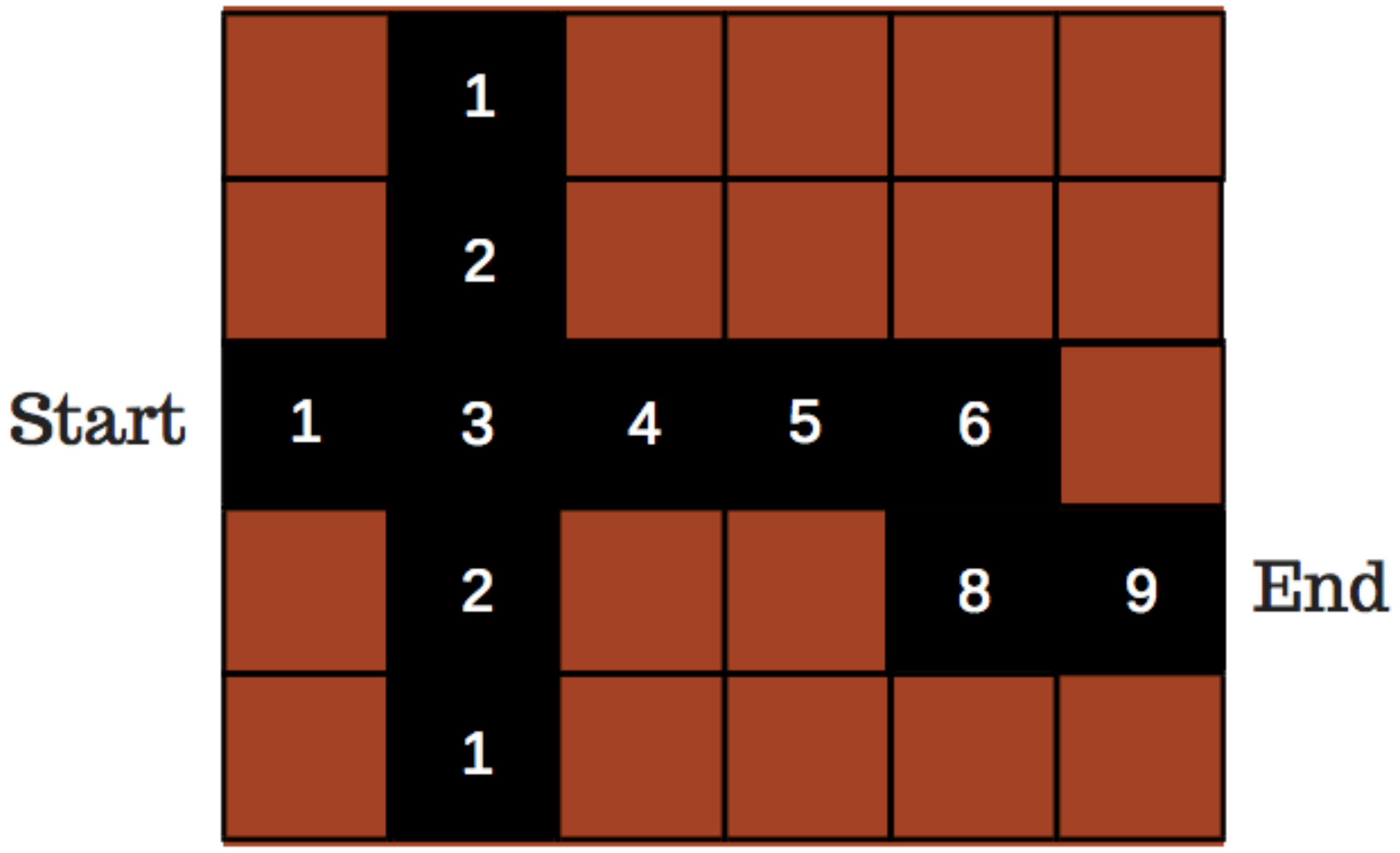
- Optimize the value function V(s)
- A function which tells us the maximum expected future reward the agent shall get at each state
- The agent will always take the state with the highest value

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right]$$

Expected

Reward
discounted

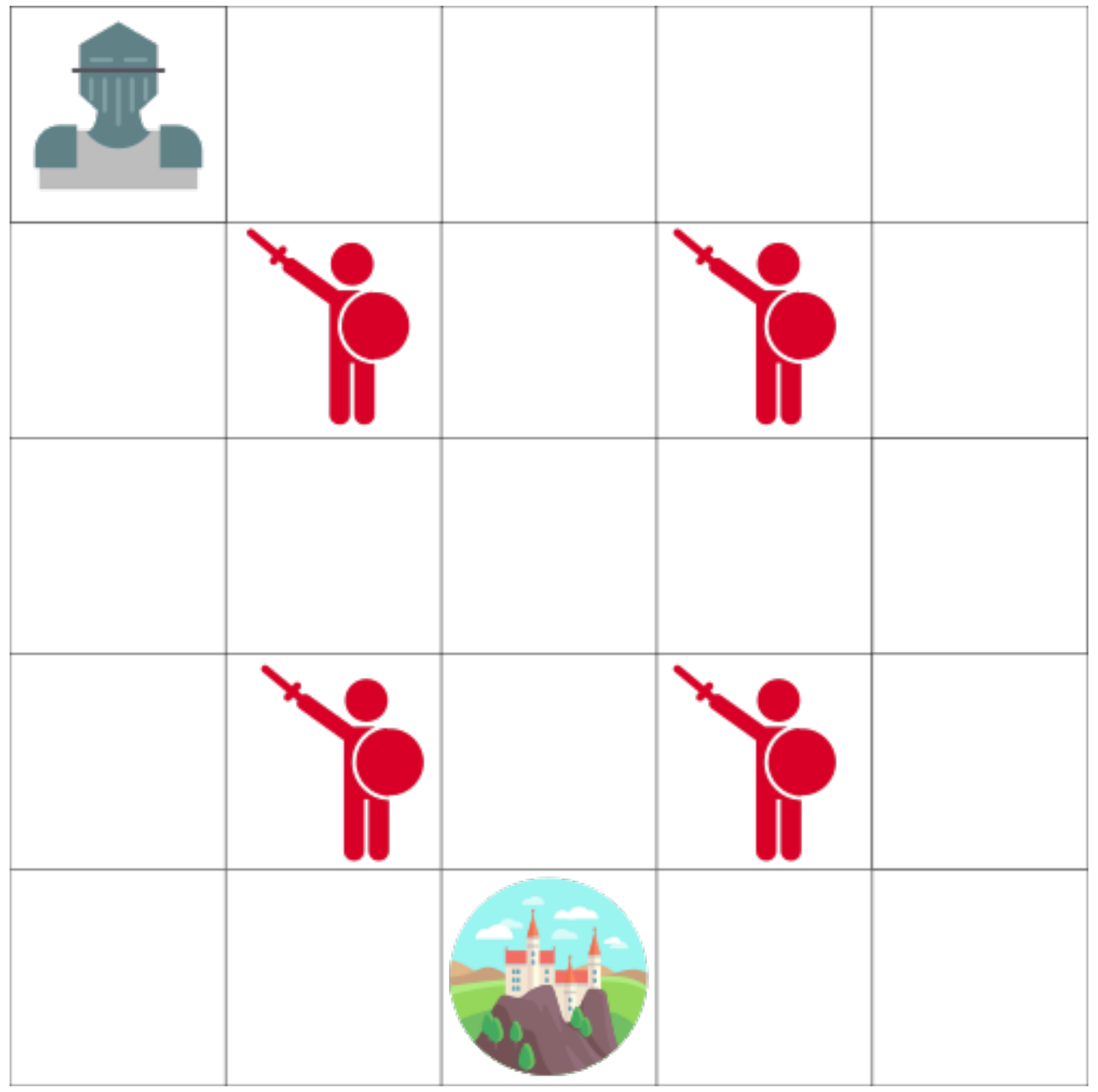
Given that state



Q-learning

A value based RL algorithm

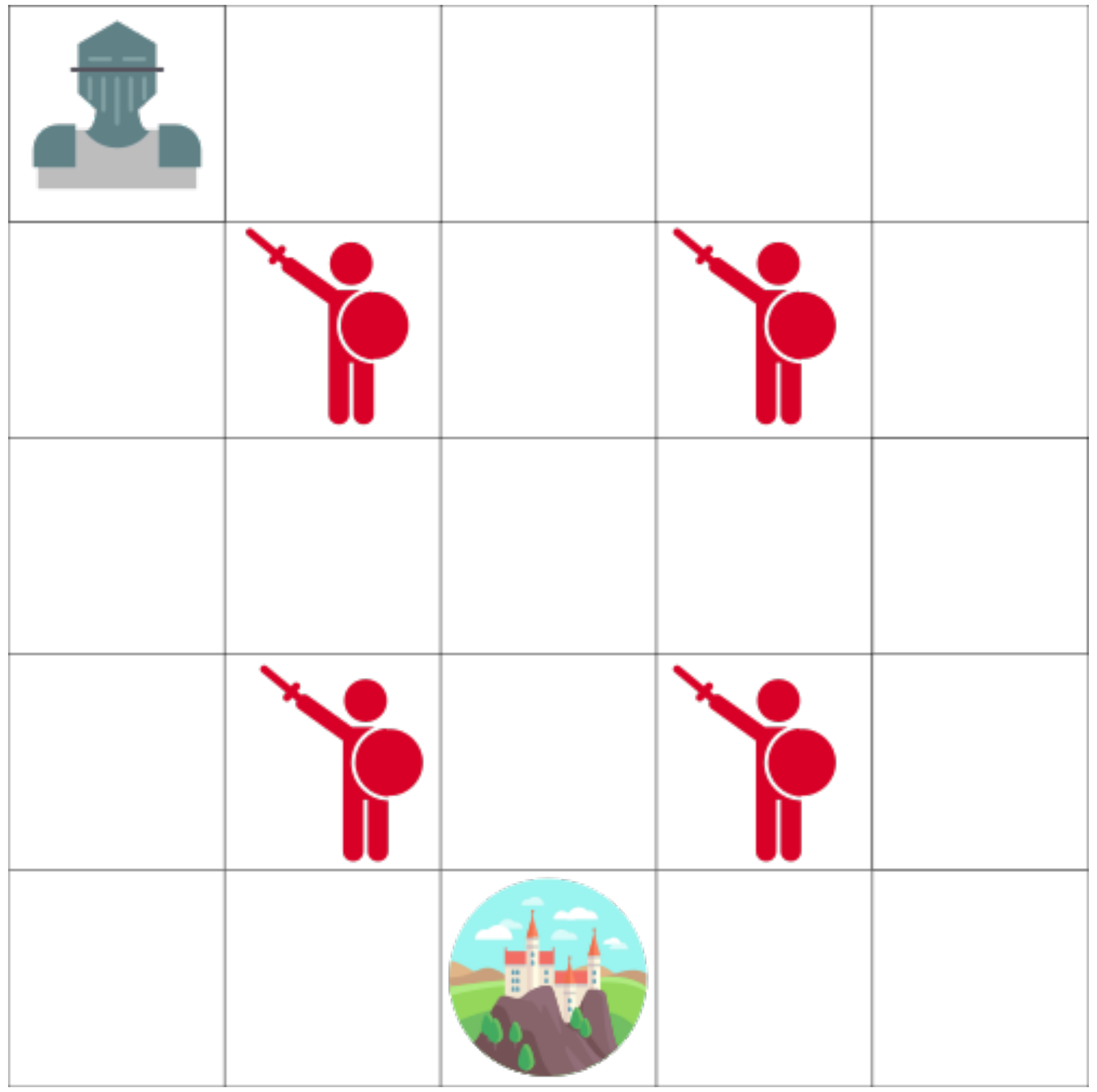
- A knight needs to save the princess trapped in the castle shown on the map below
- Your goal is to go the castle by the fastest route possible
- Scoring system
  - -100 if you touch an enemy and the episode ends
  - +100 if you reach the castle and you win
  - -1 for each step you take (this helps to agent to be as fast as possible)

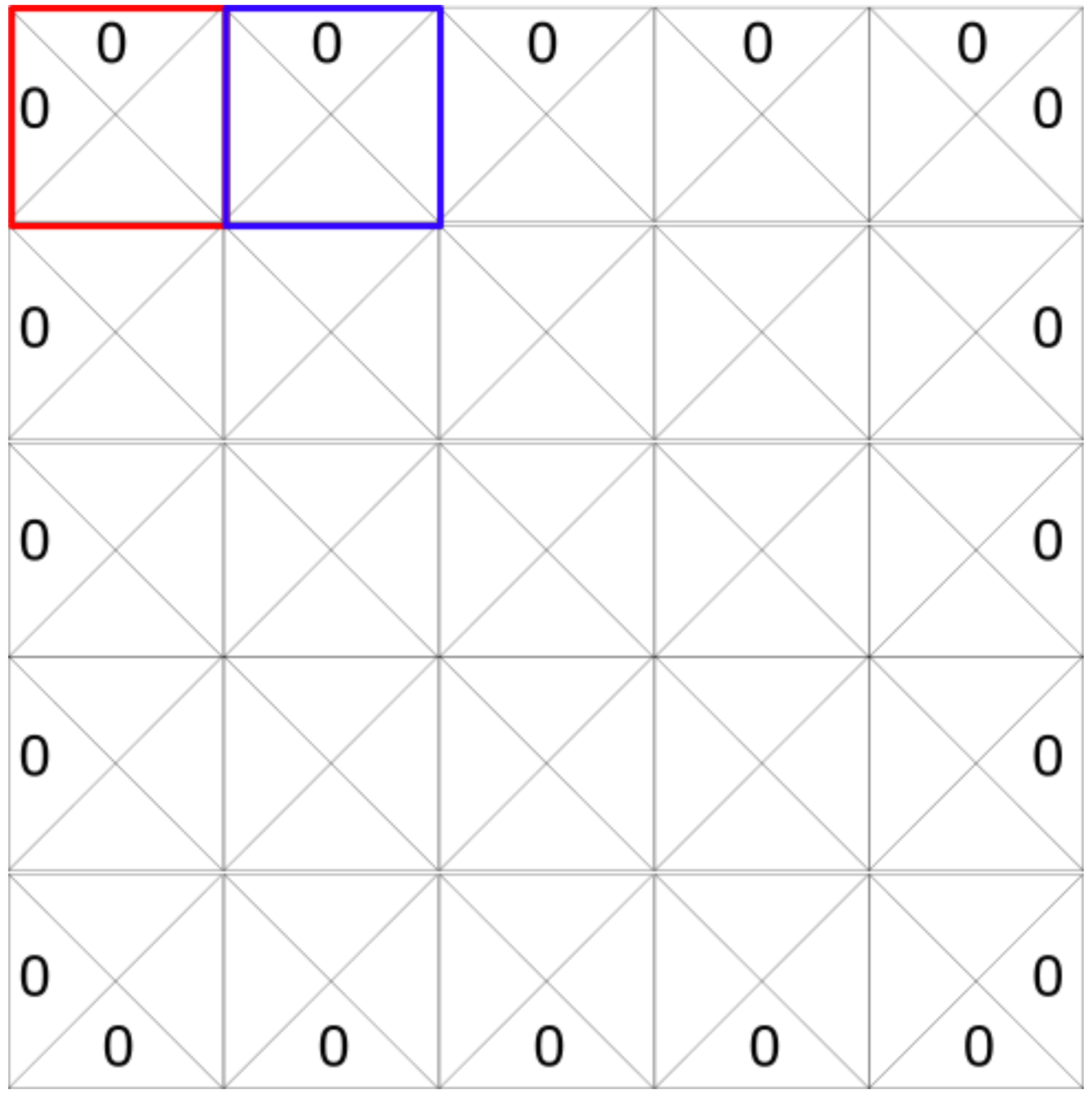


Q-learning

Q-table

- Create a table where we'll calculate the maximum expected future reward, for each action at each state

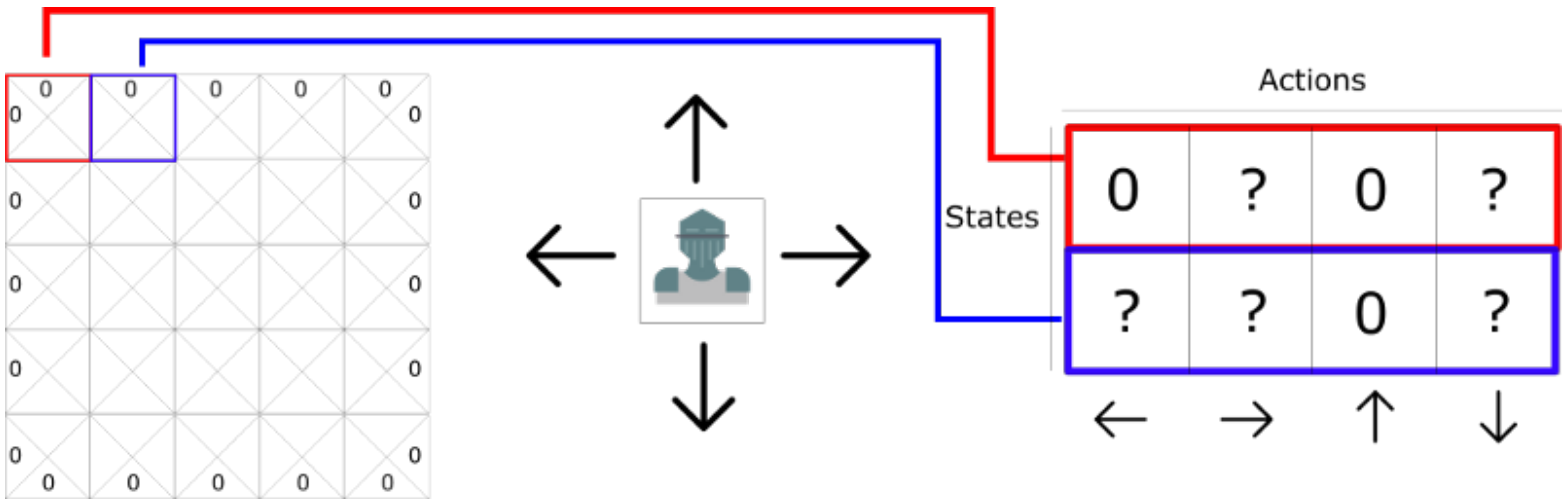




Q-learning

Q-table

- Each Q-table score will be the maximum expected future reward that the agent will get if it takes that action at that state.
- We will keep improving our Q-table values to always choose the best action
- The Q-table is a “cheat sheet” for the agent

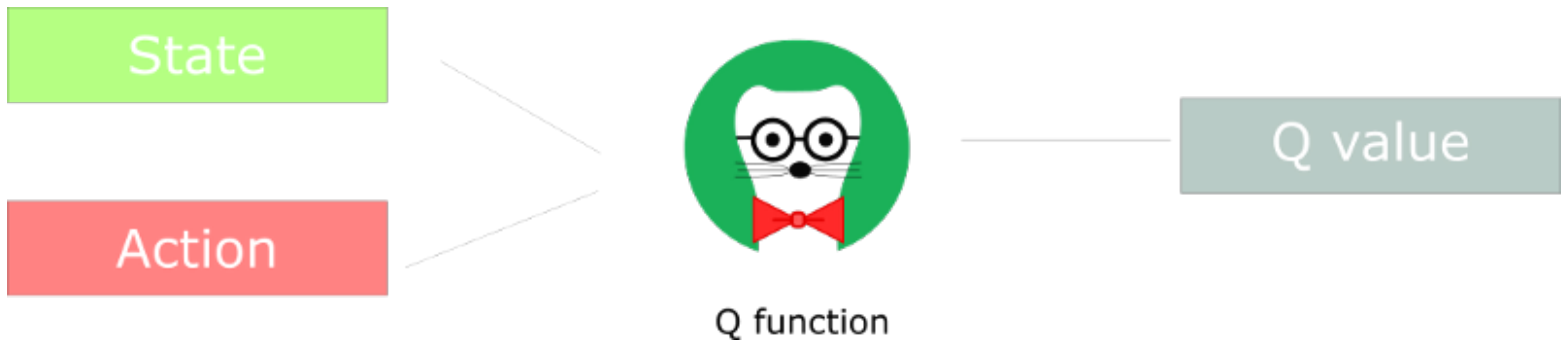


Q-learning

Action Value Function

- How do we calculate the values for each element of the Q table?
- The Q function above is a reader the scrolls thought our Q-table and finds the value associated with our state (s) and action (a)

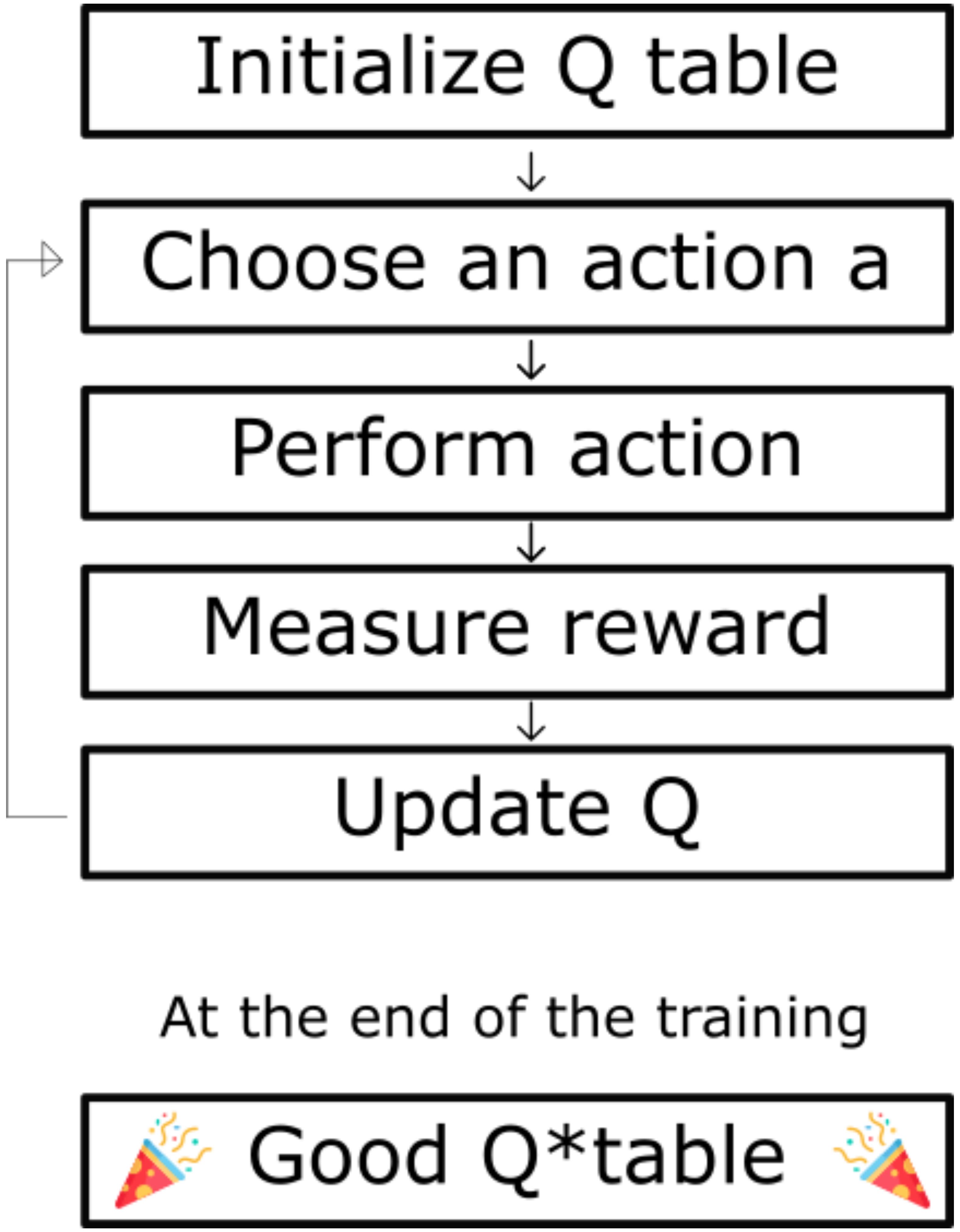
$$\underbrace{Q^{\pi}(s_t, a_t)}_{\text{Q value for that state given that action}} = \underline{E}[\underbrace{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots}_{\text{Expected discounted cumulative reward ...}} | \underbrace{s_t, a_t}_{\text{given that state and that action}}]$$



Q-learning

The process

- As we **explore** the environment, the Q-table will give us a better and better approximation by **iteratively updating Q(s,a)** using the **Bellman Equation**



1. Initialize Q-values ($Q(s, a)$) arbitrarily for all state-action pairs.
2. For life or until learning is stopped...
3. Choose an action (a) in the current world state (s) based on current Q-value estimates ($Q(s, \cdot)$).
4. Take the action (a) and observe the the outcome state (s') and reward (r).
5. Update $Q(s, a) := Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$

Q-learning

Step 1: Initialize Q-values

- We build a Q-table, with m cols (m= number of actions), and n rows (n = number of states). We initialize the values at 0.

	Actions			
States	0	0	0	0
	0	0	0	0
	0	0	0	0
		...		
	0	0	0	0

Q-learning

Step 2: For life (or until learning is stopped)

- Steps 3 to 5 will be repeated until we reached a maximum number of episodes (specified by the user) or until we manually stop the training.

Q-learning

Step 3: Choose an action

- Which actions should we take when everything is 0?
  —> Exploration/exploitation trade-off
- Use simulated annealing to start (or epsilon-greedy)
- The chance to take a random action (Epsilon rate) is high at the beginning and decreases over time

Q-learning

Steps 4–5: Evaluate

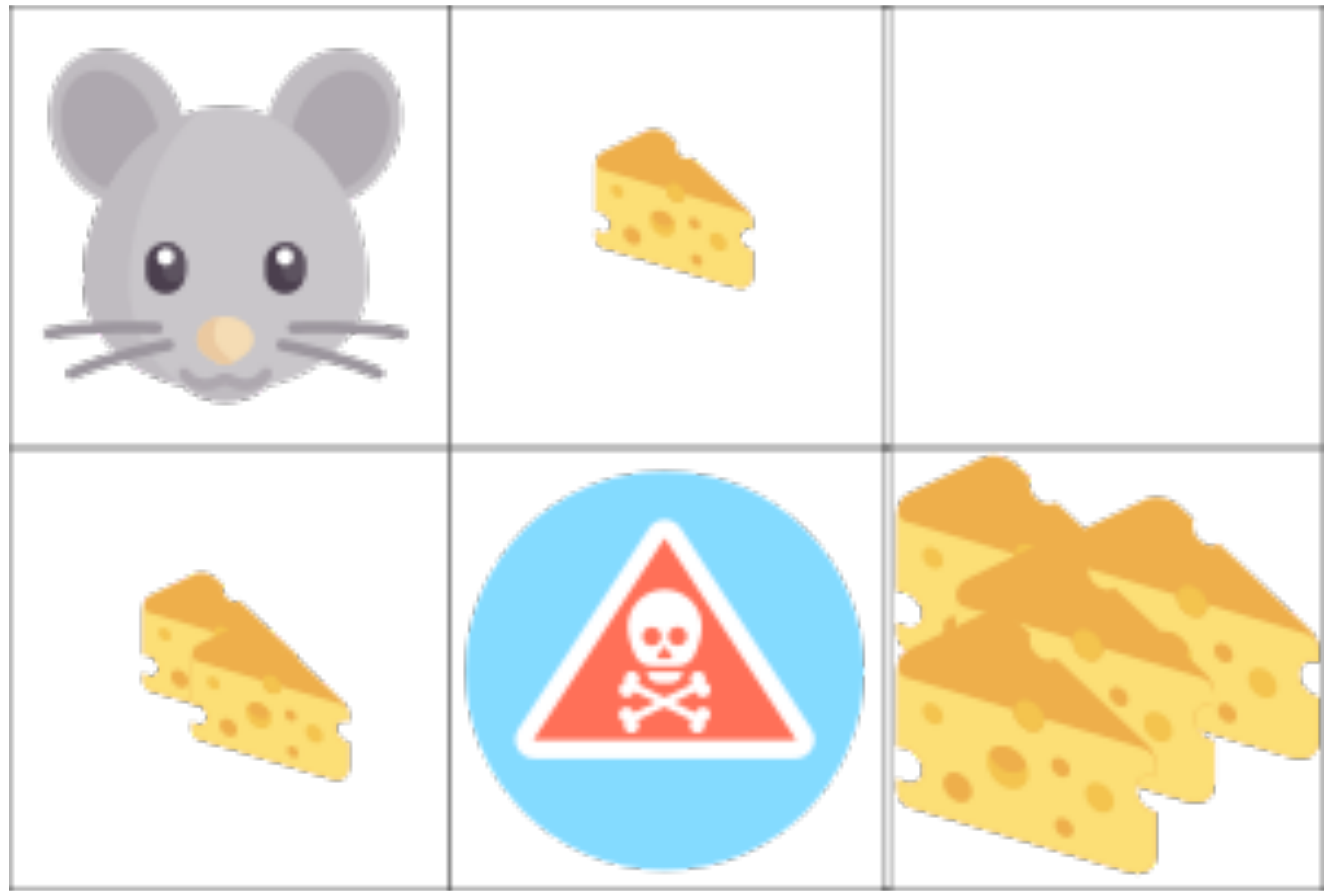
- Take the action a and observe the outcome state s’ and reward r. Now update the function Q(s,a) with the **Bellman equation**:

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

Q-learning

An example

- One cheese = +1
- Two cheese = +2
- Big pile of cheese = +10 (end of the episode)
- If you eat rat poison =-10 (end of the episode)



Q-learning

Step 1: Initialize Q-values

	←	→	↑	↓
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

Q-learning

Step 2: Choose an action

- At the beginning we choose randomly with the Simulated Annealing method
- We choose 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

Q-learning

Steps 4–5: Update the Q-function

- New Q = 0 + 0.1 x [1 + 0.9 x 0 - 0] = 0 + 0.1 x 1 = 0.1

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

	←	→	↑	↓
Start	0	0.1	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