

# Introduction to Reinforcement Learning

Michael Mollel (PhD)

Link to the project: [GitHub](#)

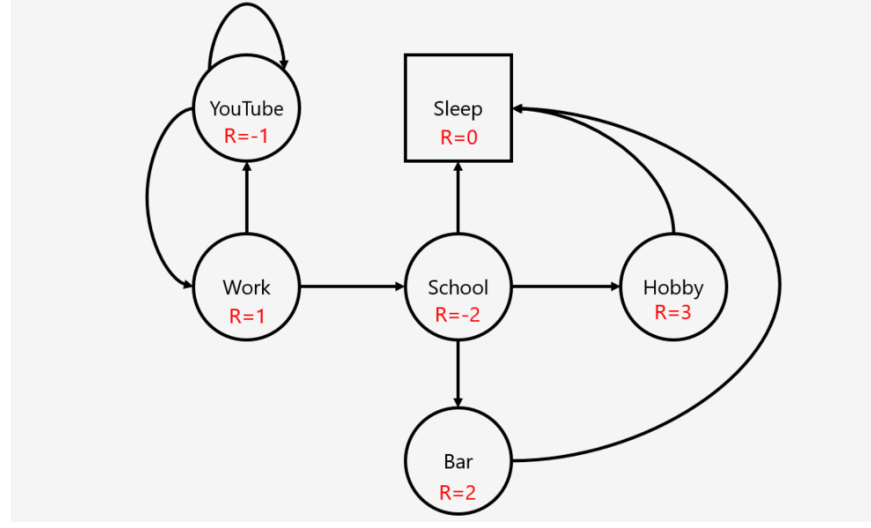
# After Presentation

- ❖ What reinforcement learning (RL) is and why it is useful
- ❖ RL applications in various domains like advertising, finance, healthcare, natural language processing, etc.
- ❖ Key terminology in RL, such as agent, action, state, reward, and environment.
- ❖ Major challenges in RL – sample inefficiency, exploration-exploitation tradeoff, sparse reward problem.
- ❖ Concepts of policies (deterministic and stochastic), trajectories, returns, value functions, and Q-functions.
- ❖ The Q-learning algorithm and how it works, including initializing the Q-table, choosing actions, and updating the table based on rewards received.
- ❖ An example application of Q-learning to the Frozen Lake problem environment

# What is RL

❖ Reinforcement learning is a framework for learning any task.

❖ Any problem that is phrased as a Markov Decision Process (Markov Next Slide)



# Why is RL

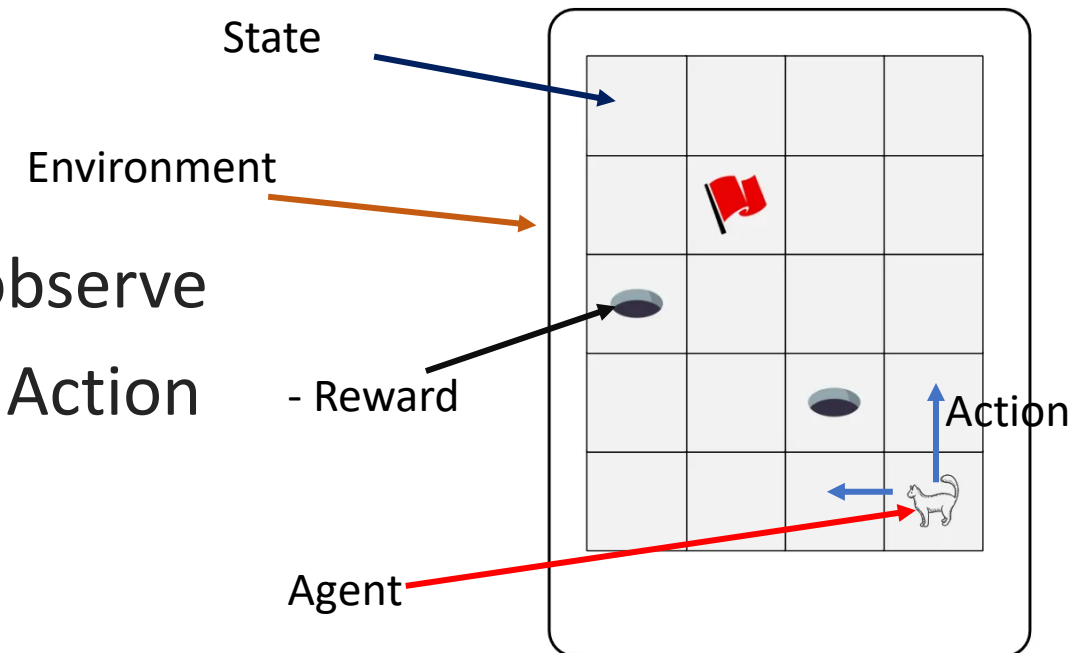
❖ RL can help us with finding novel solutions to problems, without explicitly programming tactics or solution methods.

# Application of RL

- ❖ Advertising (The ads you see on your browser)
- ❖ Finance (when to buy and when to sell)
- ❖ Healthcare (Drug design)
- ❖ NLP Large Language Models (Model alignment)

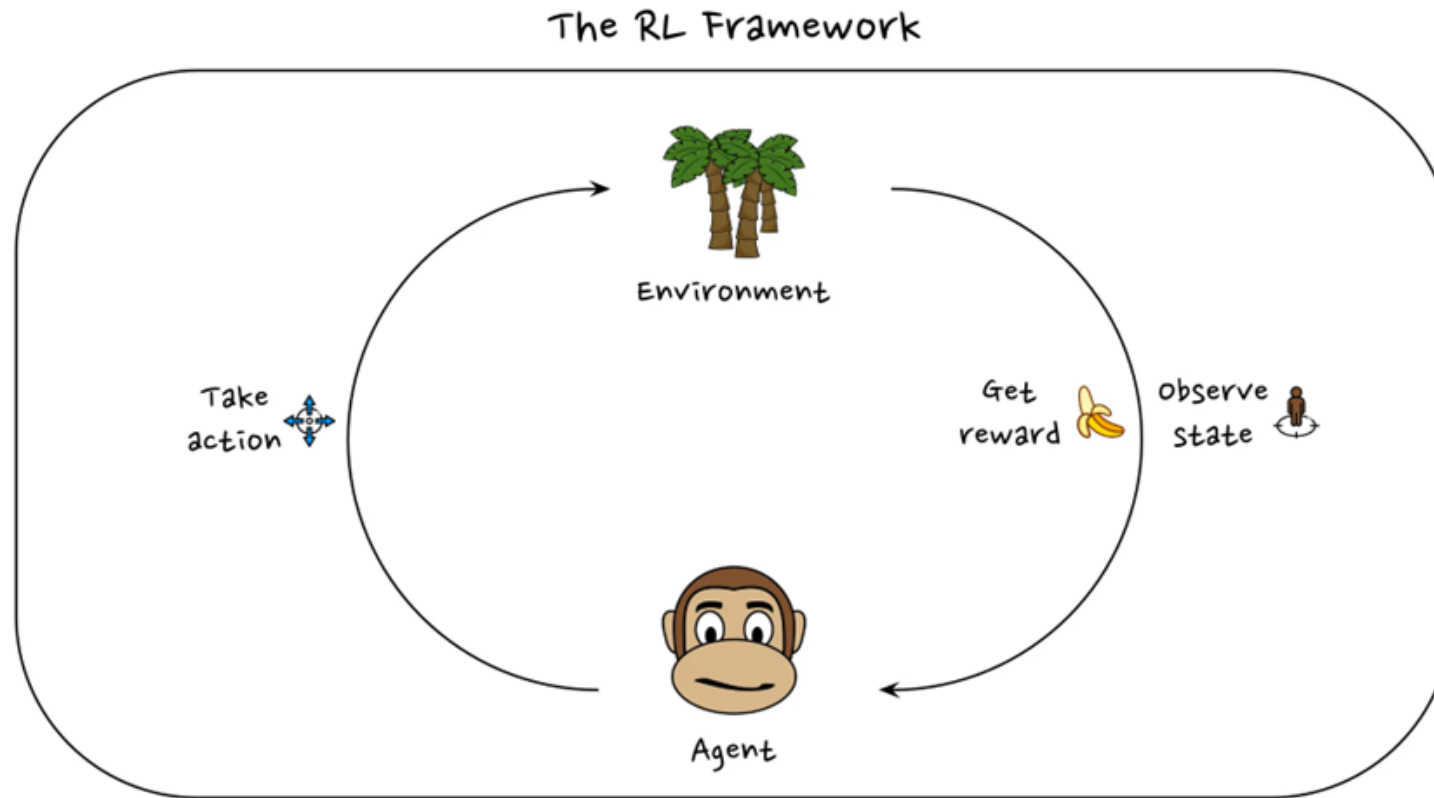
## RL - Terminology and Basic

- Agent: A thing that tries to solve a problem
- Action: a decision taken by the agent
- State: A circumstance that the agent sees or observe
- Reward: Incentive given to agent after taking Action
- Environment: a complete picture of the world



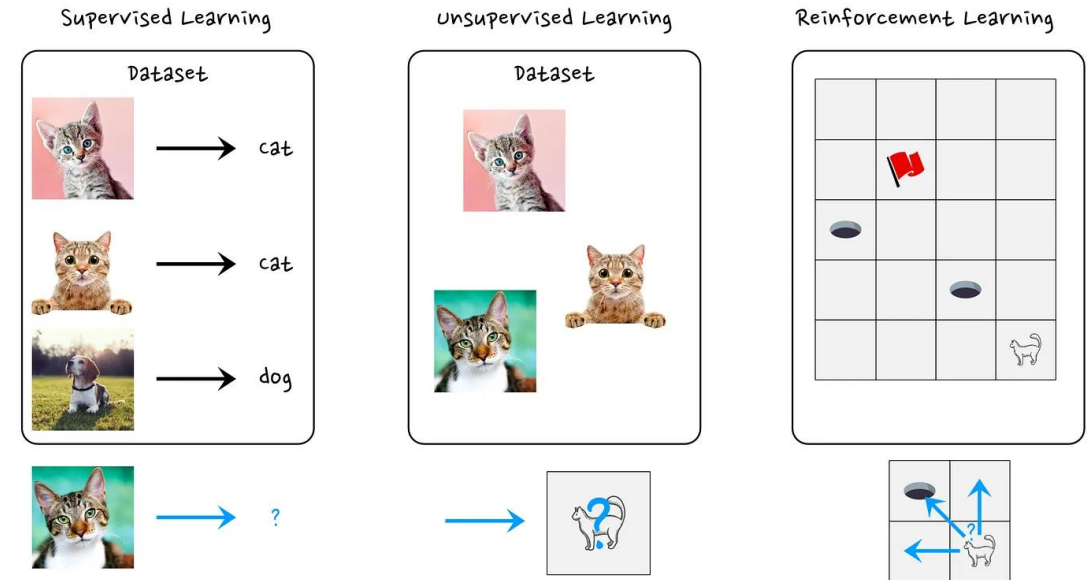
# Basic Description of RL

1. An agent that tries to solve a task in a particular environment and state  $s$
2. At every timestep  $t$ , the agent needs to choose an action  $a$ .
3. After this action, it might receive a reward  $r$
4. And get a new observation of its states



# Basic Branches of ML and Why RL

- ☐ Supervised
- ☐ Unsupervised
- ☐ Semi-Supervised
- ☐ Reinforcement



RL – can be applied to all MDP Tasks

➤ Answer: **Yes**

## Challenges

1. Sample (in-)efficiency
2. The exploration-exploitation trade-off
3. The sparse-reward problem

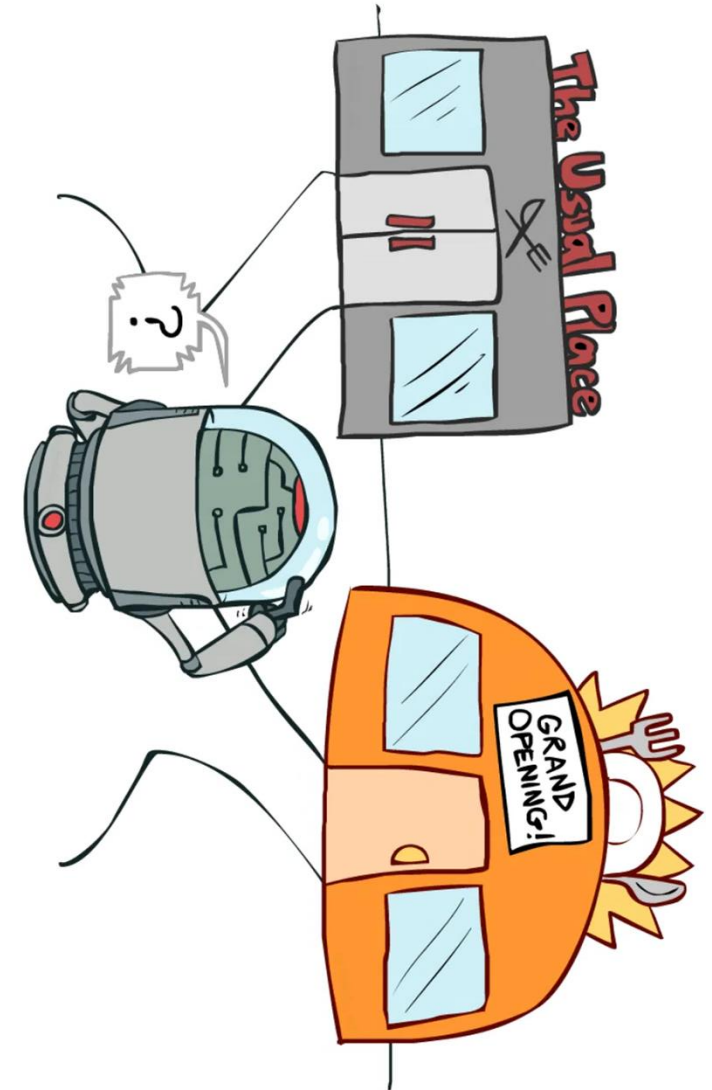
# Sample (in-) Efficiency

- ❖ Sample is an interaction with the environment.
- ❖ Most RL Algo needs a lot of samples/interactions to be able to solve a task.
- ❖ For a human, it takes a few moves to understand good moves or bad.
- ❖ Human can reuse some of the knowledge and skills already acquired from other experiences throughout their life.
- ❖ RL-agent in contrast, starts the learning process without any assumptions.



# Exploration-exploitation trade-off

- ❖ Choose between usual and unusual places. What if I choose an unusual place and get rich or get killed?
- ❖ How often I can choose to explore the unusual place
- ❖ This is the exploration-exploitation trade-off, we want an automated way to strike a good balance between letting the agent explore and taking actions for which it already knows what they will lead to.
- ❖ For a lot of problems, it is quite possible that the agent gets stuck in a local optimum
- ❖ *One of the hardest problems for RL to solve*





# The sparse-reward problem

- ❖ RL-agent receives so little rewards, that it actually gets no feedback on how it should improve.
- ❖ If we only give our agent a reward (a positive feedback signal) when we have reached the flag, it might not ever get a positive feedback signal, simply because it might never reach the flag by taking random actions (exploration).
- ❖ Reward shaping: modify the reward such that the agent gets more feedback signals to learn from
- ❖ reward shaping is not a scalable solution



# RL - Terminology Advance

- **Policies:** “behavior” of our agent
  - ❖ This itself is the algorithm that we want to learn or the agent needs to learn. It answers the fundamental question: Given a state, what should be the next action the agent takes?
  - ❖ Two types of Policies exist
    1. Deterministic Policy:  $a_t = \mu_{\theta}(s_t)$ 

In this formula,  $a_t$  is an action at timestep  $t$ ,  $s_t$  is the state at timestep  $t$ , and  $\mu$  is our policy function with parameters  $\theta$ . The parameters  $\theta$  influence the output of the function, so we want to change these such that our agent behaves in a way that optimally solves the problem.
    2. Stochastic Policy:  $a_t \sim \pi_{\theta}(\cdot \mid s_t)$ 

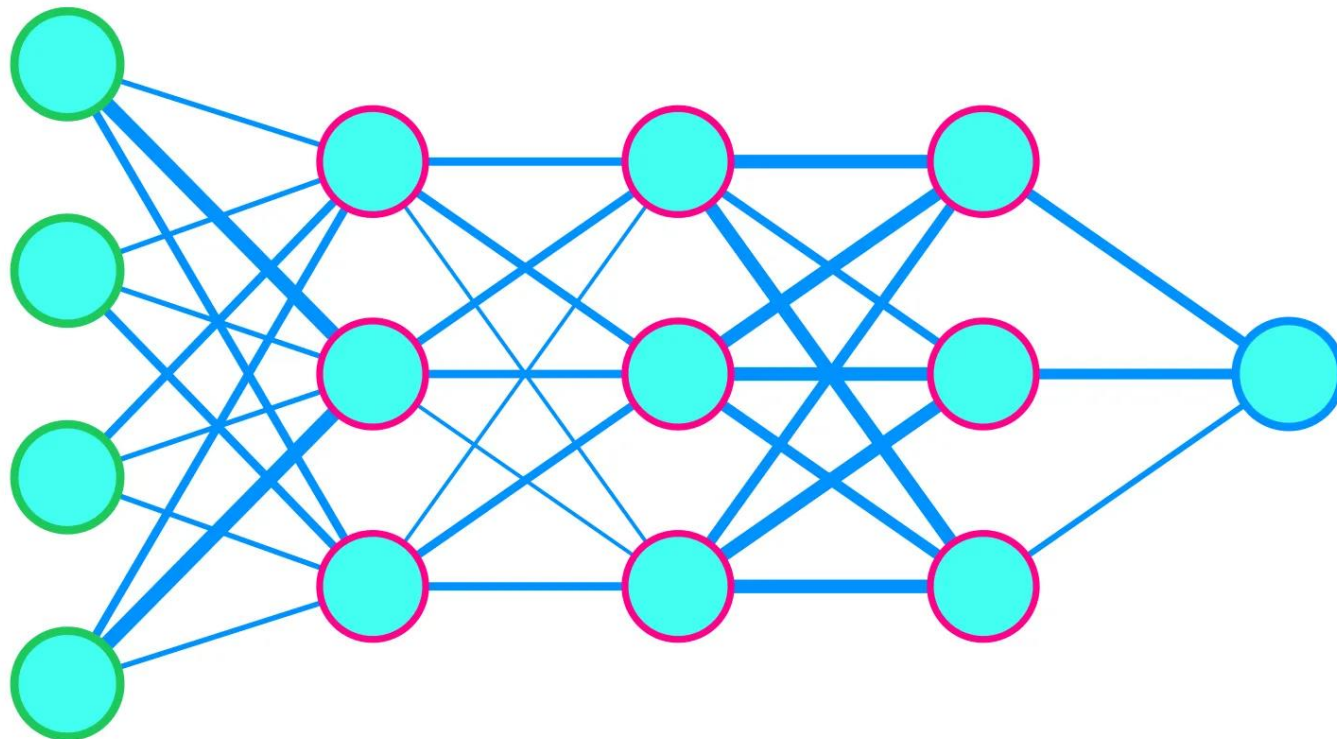
The difference with the deterministic policy is that we select action  $a_t$  with a certain probability, but it could very well be the case that the next time we are in a similar state, we select a different action instead.

# RL - Terminology Advance

- Policies Example: Neural Network

State

3.1
1.2
3.4
2.8

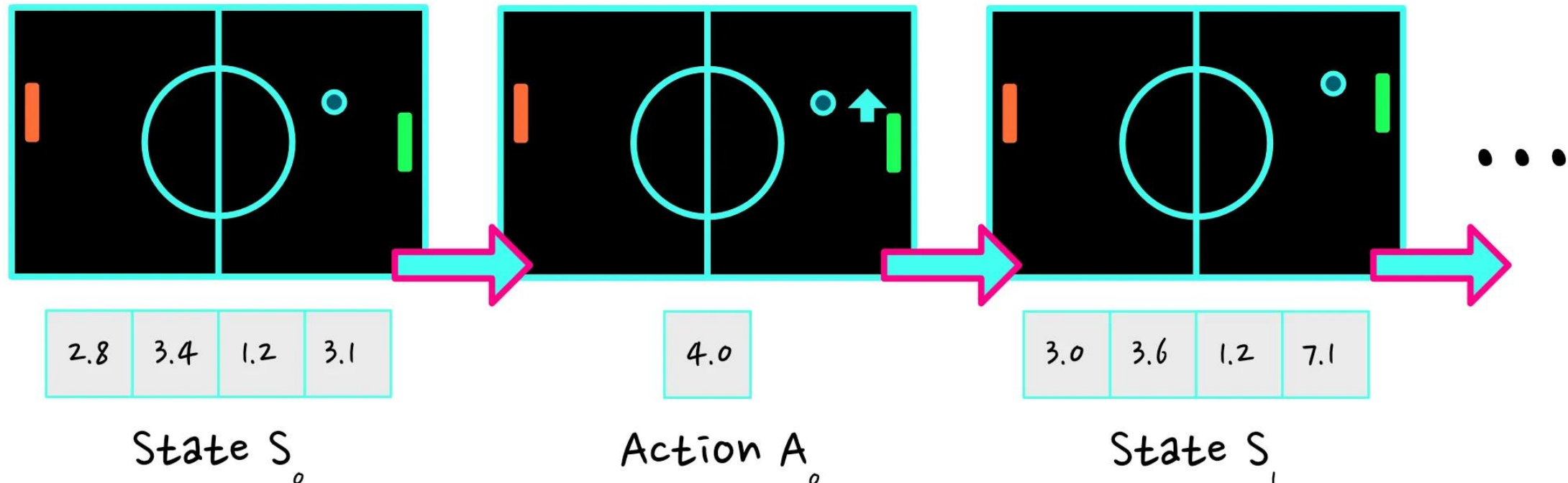


Action

5.0
-----

# RL - Terminology Advance

- **Trajectories and returns:** is simply referring to a certain sequence of states and actions:  $\tau = (s_0, a_0, s_1, a_1, \dots)$
- **Finite Trajectory:**  $R(\tau) = \sum_{t=0}^T r_t$
- **Infinite Trajectory:**  $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$  where  $\gamma$  is **discount factor**
- Gamma is always between 0 and 1 (inclusive). The lower the discount factor, the less valuable future rewards will be.



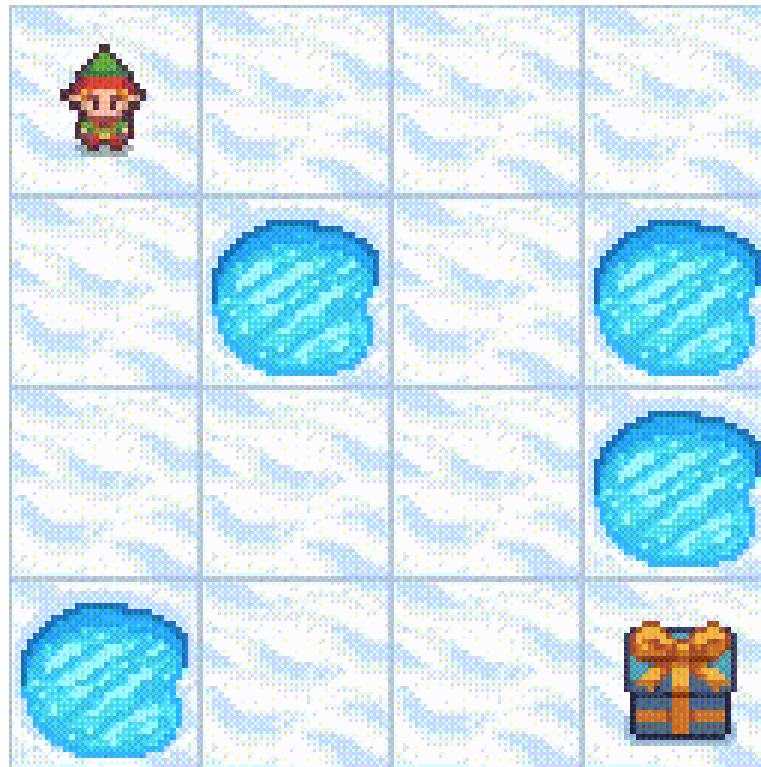
# Value function and Q-function

- **Objective:** to find a policy (a behavior) for our agent such that it will receive the highest possible cumulative reward.
- ❖ **Value function:**  $V^\pi(s) = \mathbb{E}_{\tau \sim \pi}[R(\tau) \mid s_0 = s]$ 
  - The value of being in state  $s$  *while acting under policy  $\pi$  is equal to the reward we get from the trajectory  $\tau$  that we expect to follow from policy  $\pi$ , given that state  $s$  is the starting state* (**how good it is to be in a certain state**).
- ❖ **Q- function:**  $Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi}[R(\tau) \mid s_0 = s, a_0 = a]$ 
  - The value of being in state  $s$  and taking action  $a$ , while acting under policy  $\pi$  is equal to the reward we get off the trajectory  $\tau$  that we expect to follow from policy  $\pi$ , given that state  $s$  is the starting state and the first action we take is a
  - (**how good is it to be in a state and to take a certain action**)

# Q-function -> Q-Table -> Q-Learning

- Q-learning is a model-free, value-based, off-policy algorithm that will find the best series of actions based on the agent's current state. The “Q” stands for quality. Quality represents how valuable the action is in maximizing future rewards.

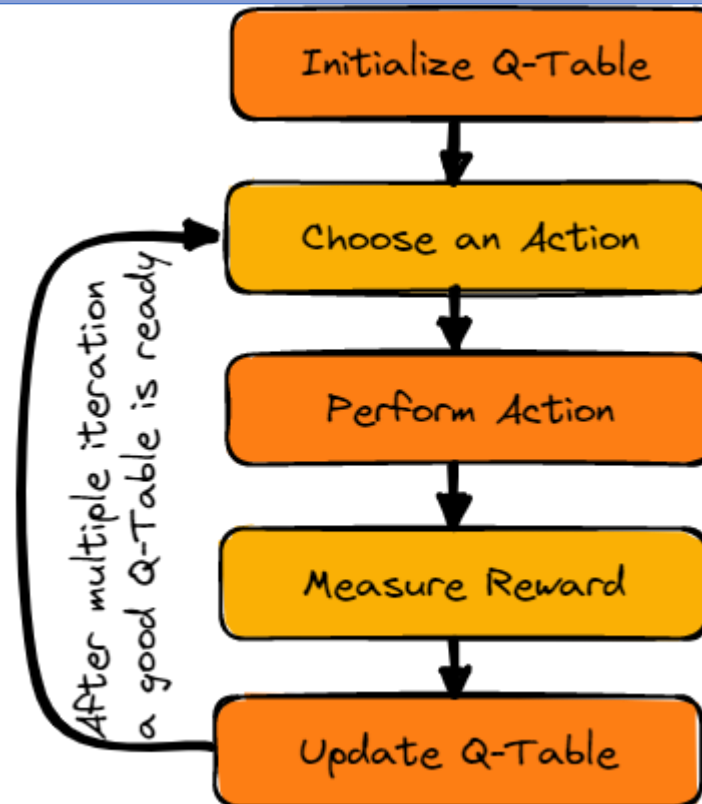
## Frozen Lake Problem



# Q-Table

- The agent will use a Q-table to take the best possible action based on the expected reward for each state in the environment. Simply put, a Q-table is a data structure of sets of actions and states, and we use the Q-learning algorithm to update the values in the table.

## Q-Learning Algorithm







# Initialize Q-Table

- first, initialize the Q-table. We will build the table with columns based on the number of actions and rows based on the number of states.

## Q-Table Initialization ()

State	UP	Action		
		DOWN	LEFT	RIGHT
(0,0)	0	0	0	0
(0,1)	0	0	0	0
(0,2)	0	0	0	0
(0,3)	0	0	0	0
(1,0)	0	0	0	0
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(2,0)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(3,0)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0





				
<b>Start</b>	0	0	0	0
<b>Idle</b>	0	0	0	0
<b>Hole</b>	0	0	0	0
<b>End</b>	0	0	0	0



# Choose and Action

- At the start, the agent will choose to take the random action(down or right), and on the second run, it will use an updated Q-Table to select the action.

## Apply Action and Update Table

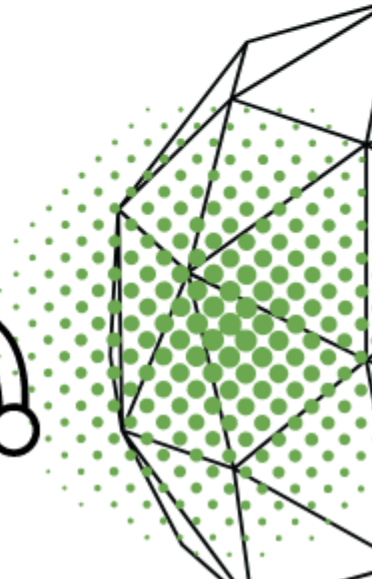
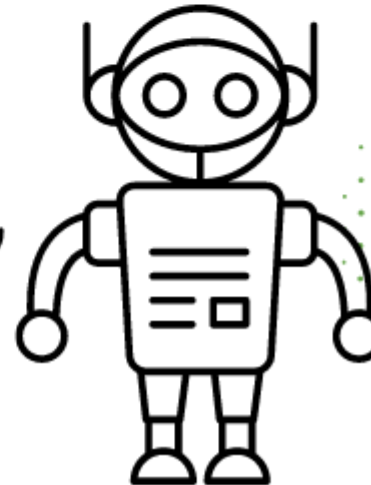
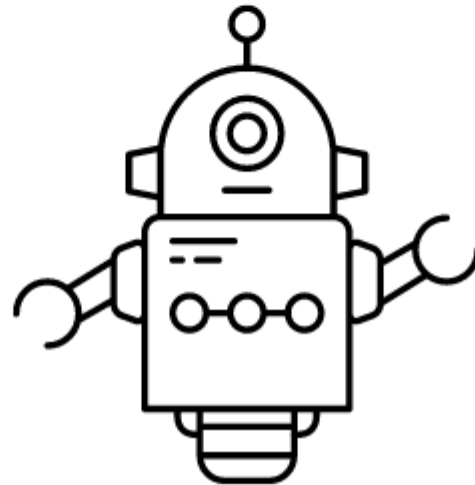
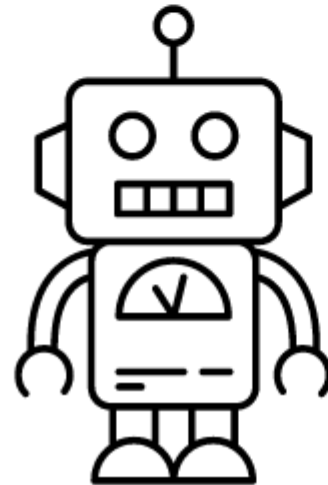
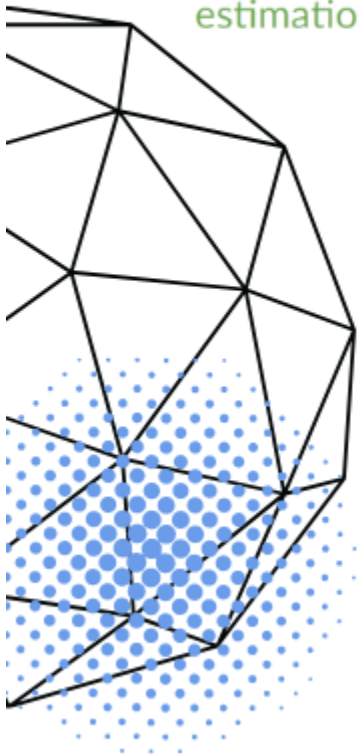
				
<b>Start</b>	0	0	0	1
<b>Idle</b>	0	0	0	0
<b>Hole</b>	0	0	0	0
<b>End</b>	0	0	0	0

# The Update Function

$$\underbrace{Q(S_t, A_t)}_{\text{New Q-value estimation}} \leftarrow \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}} + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma \max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}} - \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}}]$$

TD Target

TD Error



What left: Practice and More lectures

# Group of RL Algorithm

