UNIVERSITY OF BIRMINGHAM

Intelligent Interactive Systems: Reinforcement Learning

Mubashir Ali
m.ali.16@bham.ac.uk

Learning Outcomes

By the end of this lesson, you should be able to:

1. What is reinforcement learning?
2. Why we need reinforcement learning?
3. How reinforcement learning works?
4. What is Markov decision process
5. Bellman equation
6. What is Q-learning?
7. How Q-learning works?

What is reinforcement learning

- Reinforcement Learning (RL) is a type of machine learning (ML) where an agent learns to make decisions by interacting with an environment.
- The agent tries different actions and receives feedback in the form of rewards or penalties.
- The goal of the agent is to learn a policy, or a set of rules, that maximizes its total reward over time.

Why we need reinforcement learning

- We need RL because it is particularly useful in situations where we want to train an agent to make decisions based on complex, uncertain, or dynamic environments.

RL: Complex Environment

- Imagine you're building an autonomous vehicle:
  - The vehicle needs to navigate through various traffic conditions, weather patterns, and road obstacles.
  - Simply providing a set of rules for every possible scenario is impractical;
  - the environment is too complex and ever-changing.

- RL allows the vehicle to learn from experience and adapt to new situations as they arise.

RL: Uncertain Environment

- consider the stock market:
  - It's influenced by countless factors like geopolitical events, economic indicators, and investor sentiment.
  - Predicting stock prices with absolute certainty is impossible.

- RL algorithms, however, can learn from historical data and adapt to new market conditions, making them suitable for financial decision-making

RL: Dynamic Environment

- consider the healthcare domain:
  - When diagnosing diseases, medical practitioners must consider an array of symptoms, patient history, and test results.
  - The evolving nature of diseases and medical research makes it challenging to establish a fixed set of rules.

- By employing RL, a diagnostic system could learn from a wide range of patient cases and adjust its decisions based on new information.

Reinforcement Learning

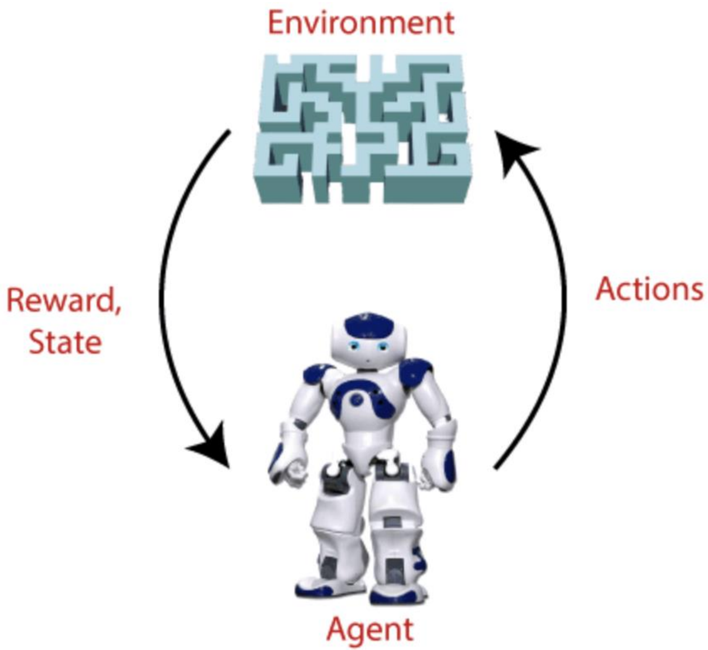
- Game playing
- Robotics
- Autonomous vehicles
- Finance
- Healthcare
- Recommended systems
- Supply chain management
- Energy management
- Natural language processing
- Many more

Reinforcement Learning

- In RL, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, so the agent is bound to learn by its experience only.
- The agent interacts with the environment and explores it by itself.

- The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.

Reinforcement Learning



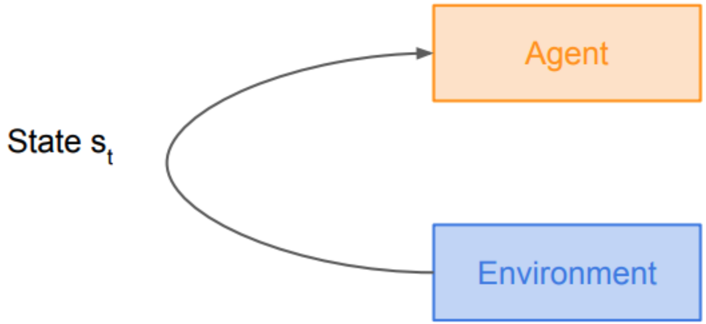
Reinforcement Learning

- The agent is the ML algorithm (or the autonomous system)

- Environment is the adaptive problem space with attributes variables, boundary values, rules, and valid actions

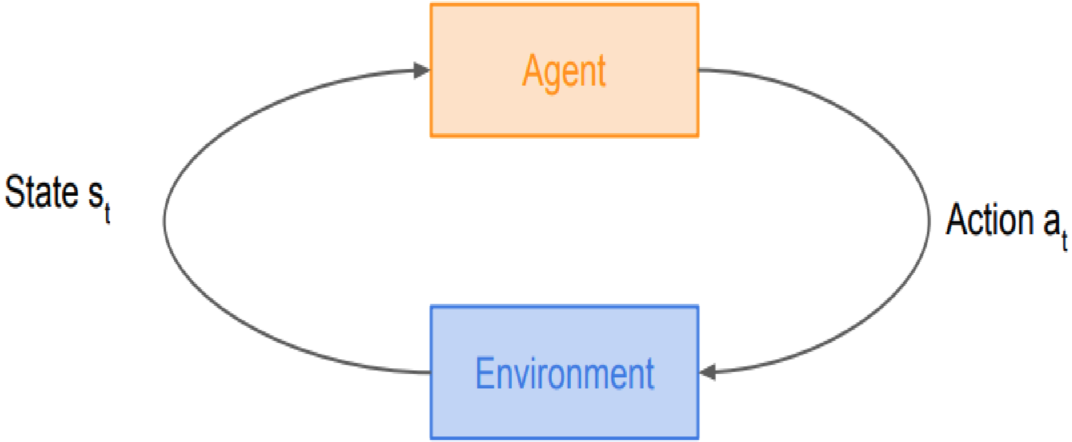
Reinforcement Learning

- The state is the environment at a given point in time



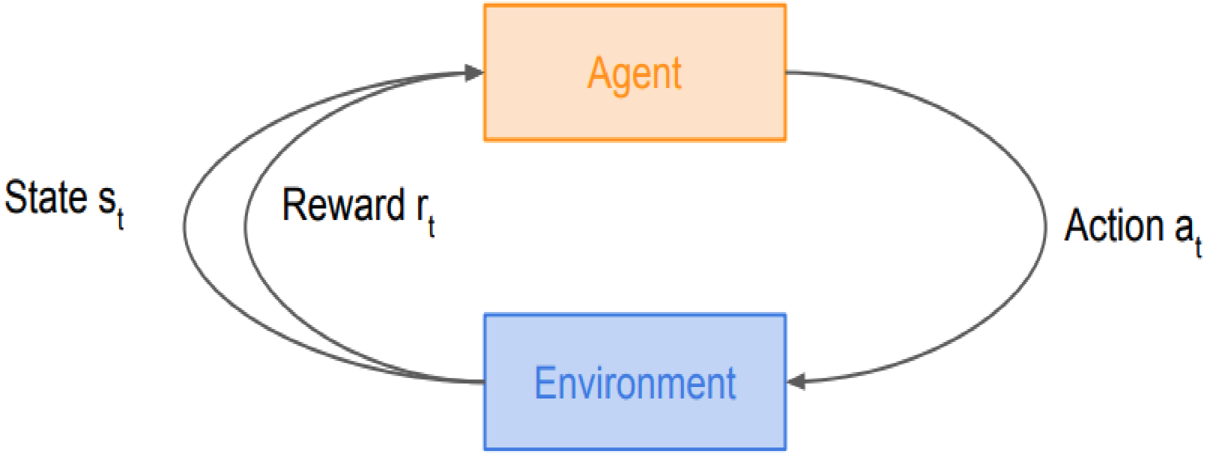
Reinforcement Learning

- The action is a step that the RL agent takes to navigate the environment

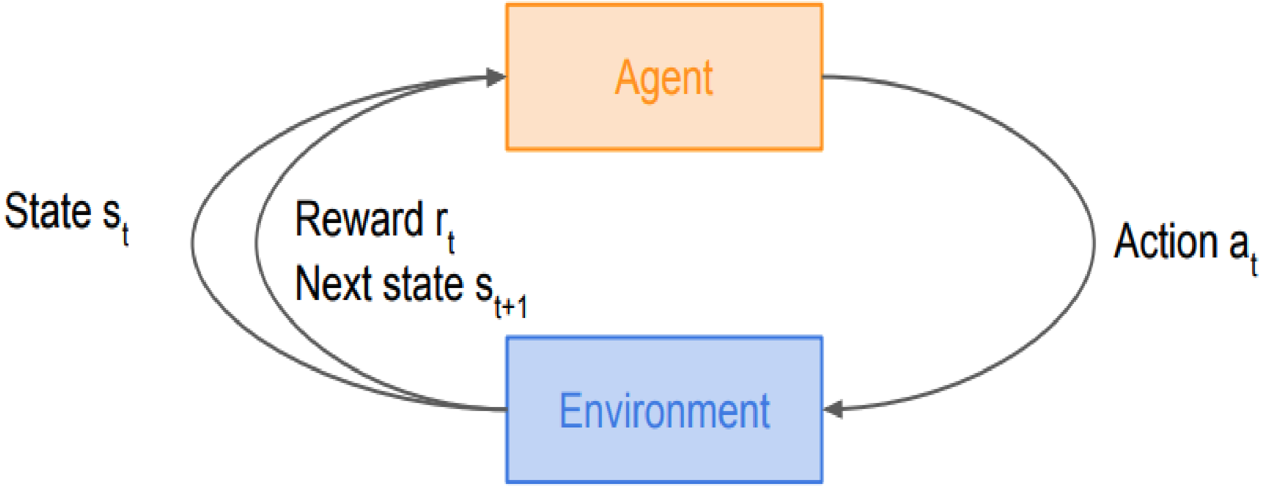


Reinforcement Learning

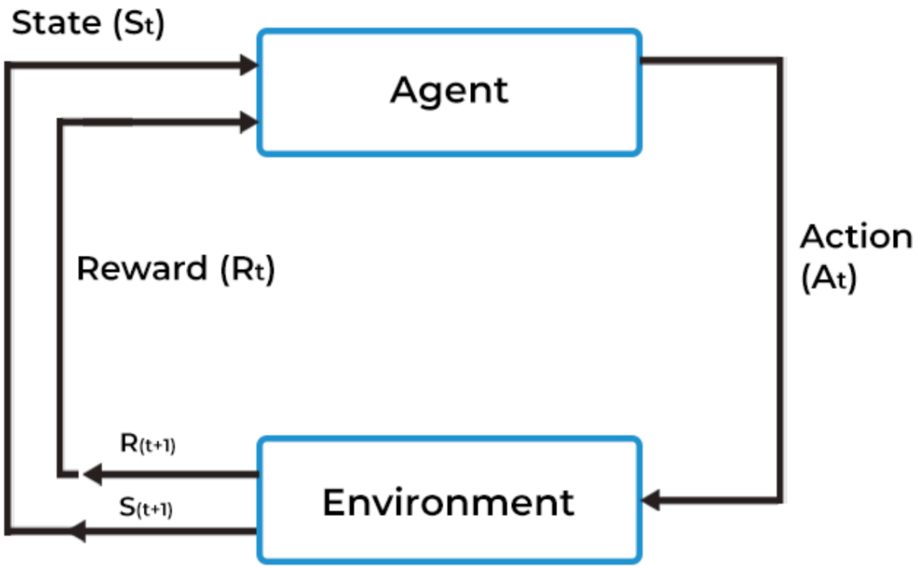
- The reward is the positive, negative, or zero value—in other words, the reward or punishment—for taking an action



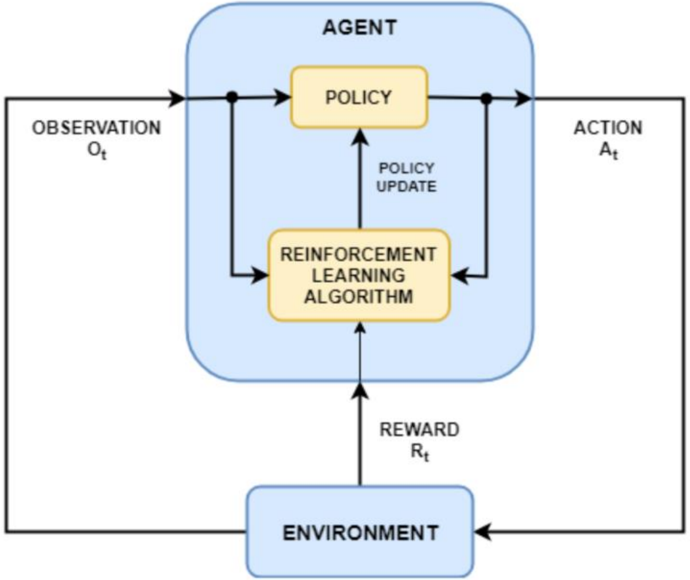
Reinforcement Learning



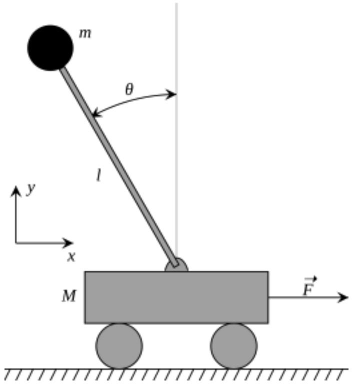
Reinforcement Learning



Reinforcement Learning



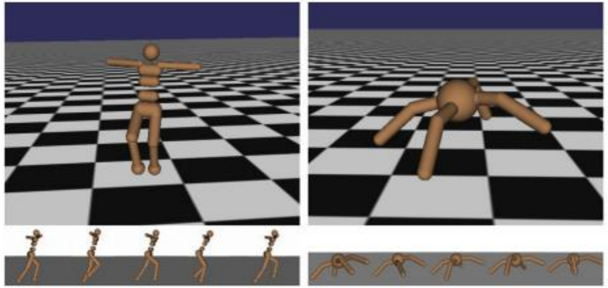
RL - Cart-Pole Problem



Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

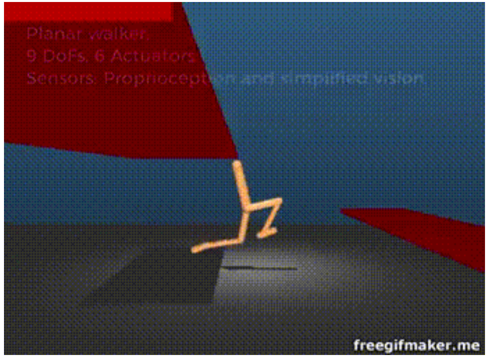
RL - Robot Locomotion

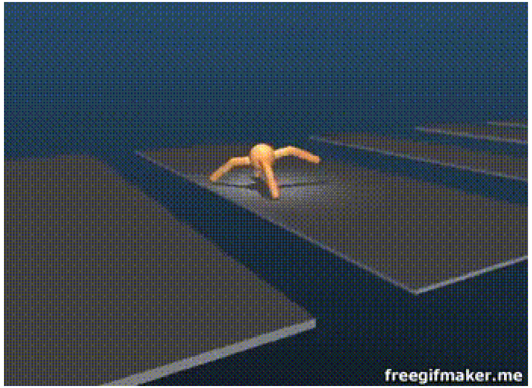


Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

RL - Robot Locomotion



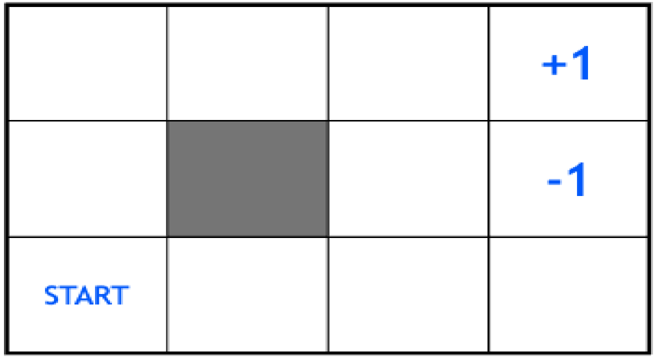


RL - Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

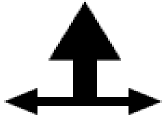
A simple MDP: Robot in a room



actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

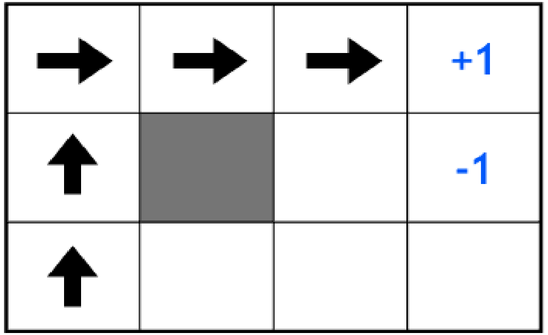


reward +1 at [4,3], -1 at [4,2]
reward -0.04 for each step

- states
- actions
- rewards

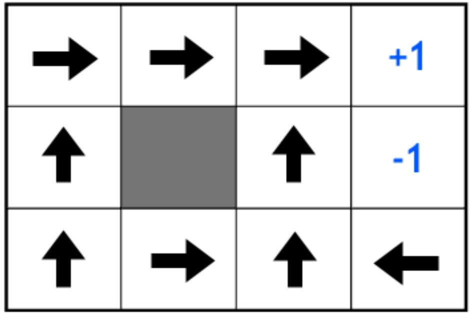
- what is the solution?

Is this a solution?

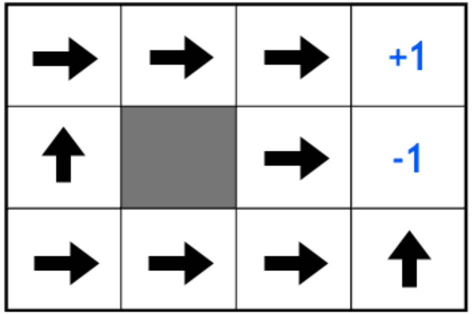


- only if actions deterministic
  - not in this case (actions are stochastic)
- solution/policy
- A policy **π** is a function from S to A that specifies what action to take in each state

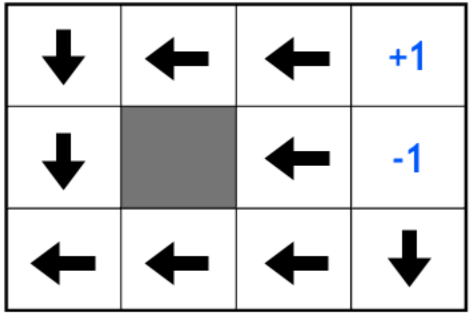
Reward for each step: -0.1



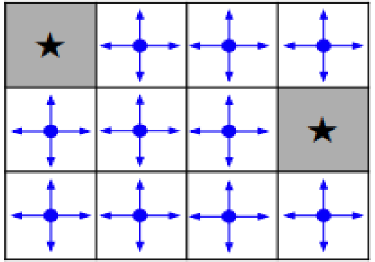
Reward for each step: -2



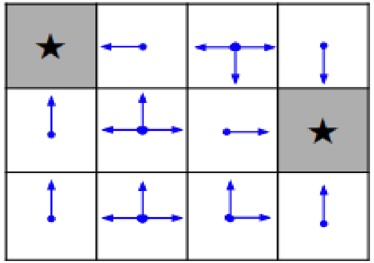
Reward for each step: +0.01



Random policy vs optimal policy



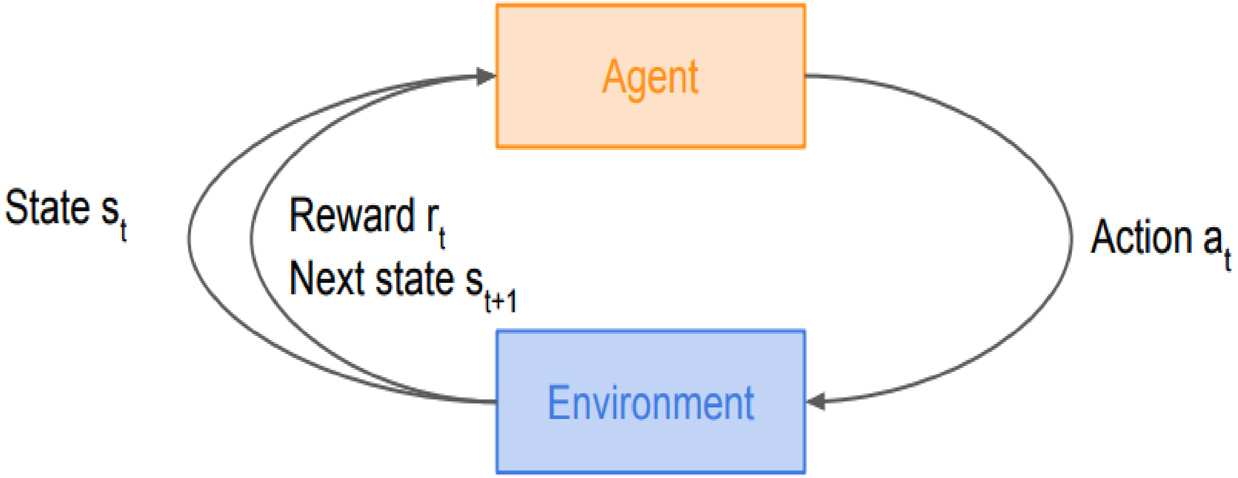
Random Policy



Optimal Policy

We want to find optimal policy π^* that maximizes the sum of rewards.

How can we mathematically formalize the RL problem?



Markov Decision Process (MDP)

- Mathematical formulation of the RL problem
- Set of states S
- Set of actions A
- Initial state S_0
- transition model P(s,a,s')
  - P([1,1], up, [1,2]) = 0.8
- Reward function r(s) (and discount γ)
  - r([4,3]) = +1

Markov Decision Process (MDP)

- A policy π is a function from S to A that specifies what action to take in each state.
- **Objective**: find policy π^* that maximizes cumulative discounted reward $\sum_{t \geq 0} \gamma^t r_t$

Definitions: optimal policy **π***

- We want to find optimal policy **π*** that maximizes the sum of rewards.

- How do we handle the randomness (initial state, transition probability…)? Maximize the **expected sum of rewards!**

- Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$

Definitions: episode

- An episode refers to a sequence of interactions between an agent and an environment that begins with the agent starting from an initial state, taking actions based on its policy, and ends when a termination condition is reached.
  - “game over” after N steps

Definitions: additive and discounted rewards

- Additive rewards:
  - $V(s0, s1, \ldots) = r(s0) + r(s1) + r(s2) + \ldots$
- Discounted rewards
  - $V(s0, s1, \ldots) = r(s0) + \gamma * r(s1) + \gamma^2 * r(s2) + \ldots$

Definitions: value function

- **state value function: $V^{\pi}(s)$**
  - The value function denoted as V(s), estimates the expected cumulative future reward an agent can expect to receive from a given state s onwards, by following a certain policy.

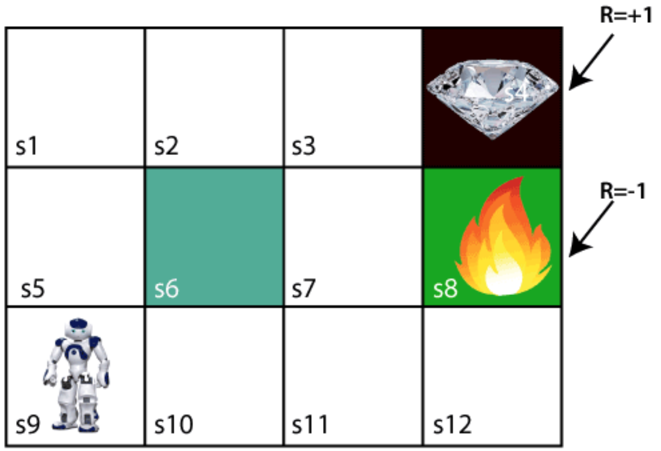
$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

Definitions: Q-value function

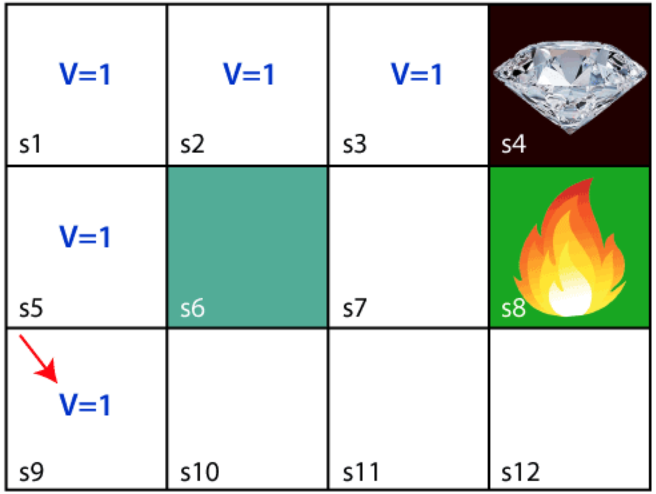
- **state-action value function: $Q^\pi(s,a)$**
  - The state-action value function, also known as the Q-function or Q-value, denoted as Q(s,a), estimates the expected cumulative future reward an agent can expect to receive by taking action a in state s, and then following a certain policy.

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

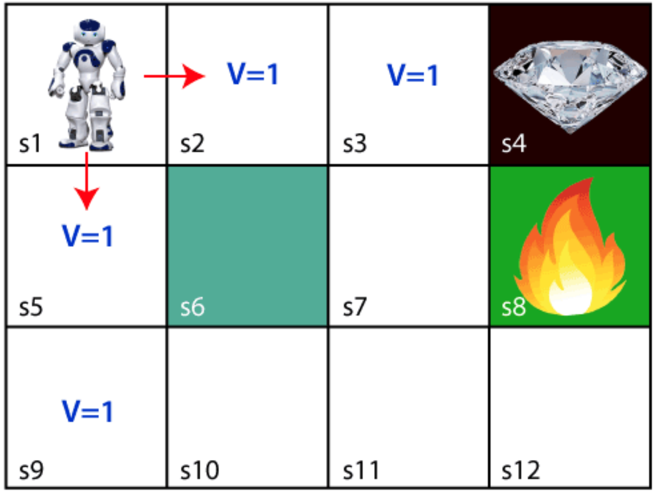
How Does Reinforcement Learning Work?



How Does Reinforcement Learning Work?



How Does Reinforcement Learning?



Bellman equation

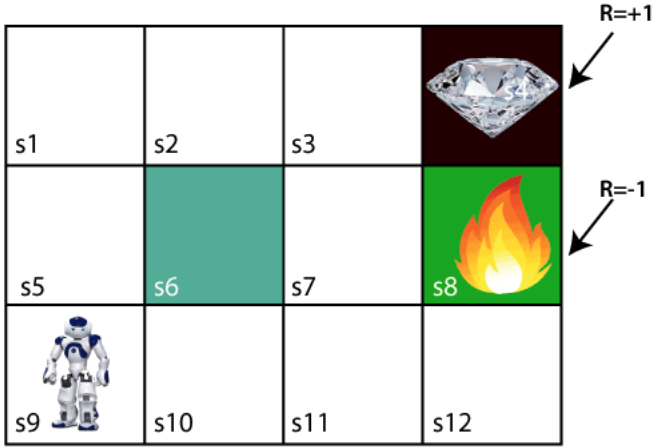
- Bellman equation is one of the main building blocks in reinforcement learning

$$V(s) = \max[R(s,a) + \gamma V(s')]$$

- V(s) = value calculated at a particular point.
- R(s,a) = Reward at a particular state s by performing an action a.
- γ = Discount factor
- V(s') = The value at the previous state.
- In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

Bellman equation

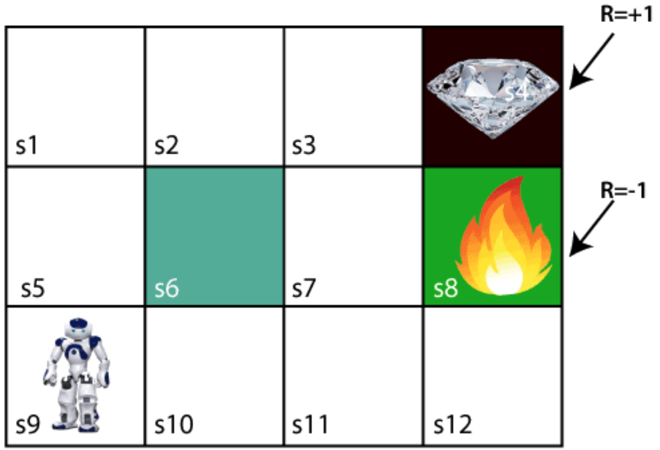
$$V(s) = \max[R(s,a) + \gamma V(s')]$$



Bellman equation

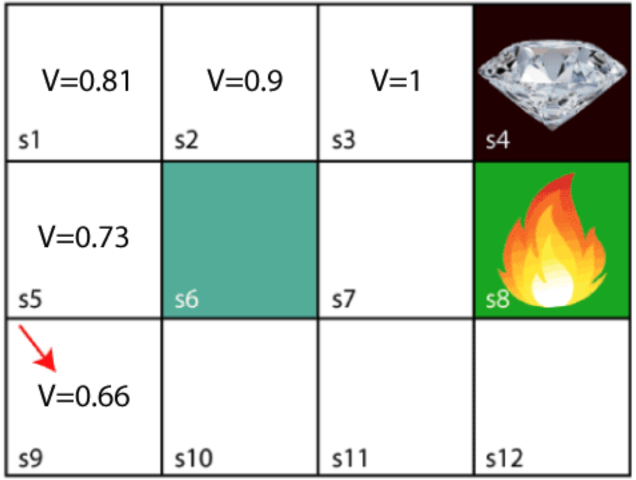
$$V(s) = \max[R(s,a) + \gamma V(s')]$$

- γ = 0.9
- V(s) = max[1] + 0
- V(s) = 1



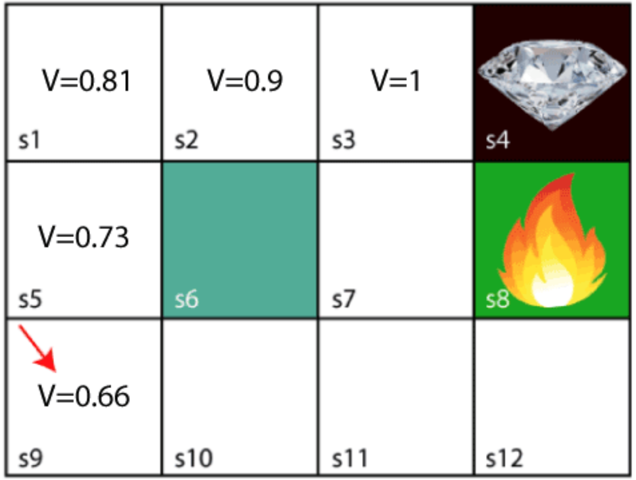
Bellman equation

- **For S3 block:**
- $V(s3) = \max[R(s,a) + \gamma V(s`)]$,
- here $V(s') = 0$ because there is no further state to move.
- $V(s3) = \max[R(s,a)] => V(s3) = \max[1] => V(s3) = 1.$



Bellman equation

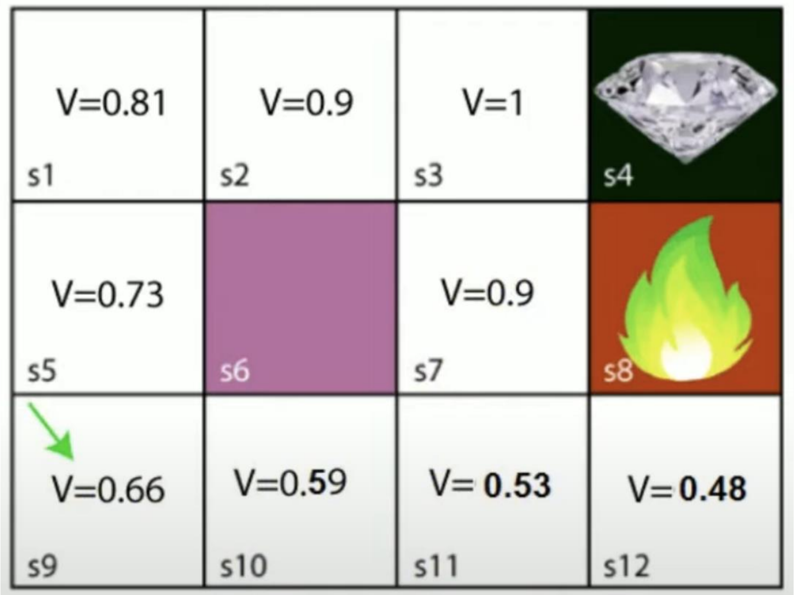
- **For S2 block:**
- $V(s2) = \max[R(s,a) + \gamma V(s`)]$,
- here $\gamma = 0.9, V(s') = 1, and\ R(s,a) = 0$, because there is no reward at this state.
- $V(s2) = \max[0.9(1)] => V(s2) = \max[0.9] => V(s2) = 0.9$



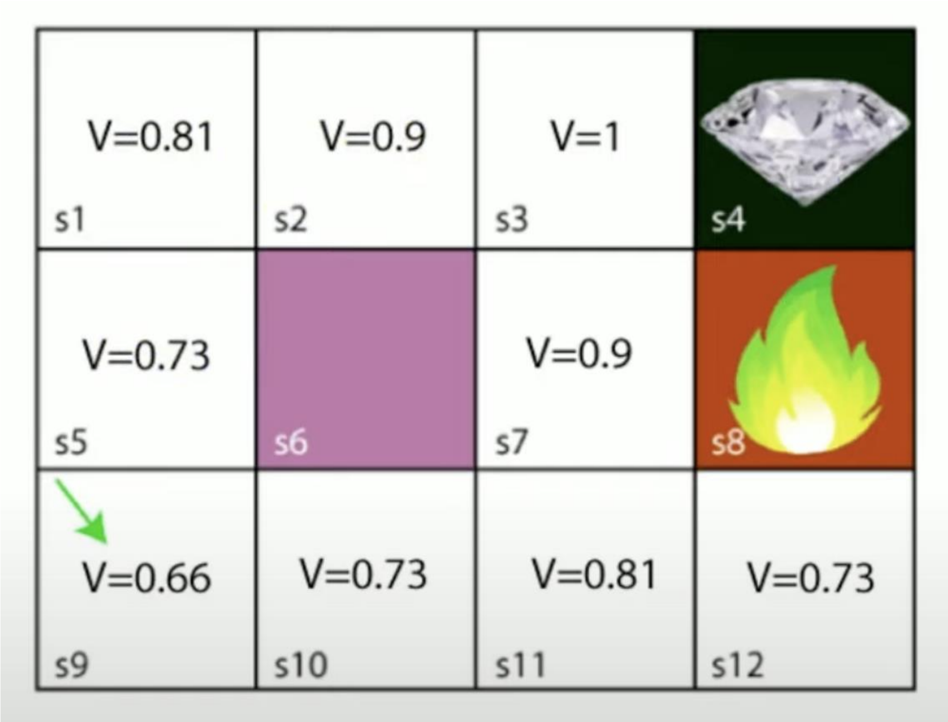
Bellman equation

- **For S7 block:**

- $V(s7) \ = \ \max[R(s,a) \ + \ \gamma V(s`)]$,
- here $\gamma = \ 0.9, V(s') = 1 \ and \ R(s,a) = \ 0$, because there is no reward at this state also.
- $V(s7) = \ \max[0.9(1)] => \ V(s7) = \ \max[0.9] =>$ $V(s7) \ = 0.9$

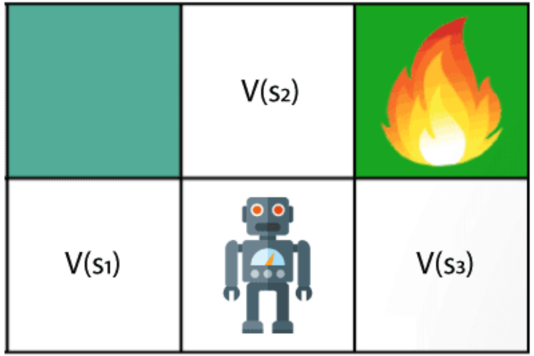


Bellman equation



Reinforcement Learning Algorithm: Q-Learning

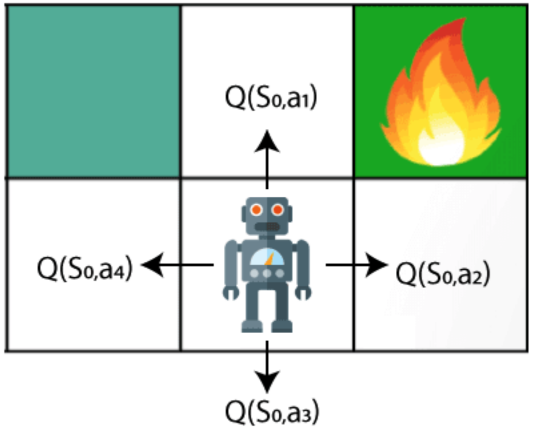
Reinforcement Learning Algorithm: Q-Learning



Here agent will take a move as per probability bases and changes the state

But if we want some exact moves?

Reinforcement Learning Algorithm: Q-Learning



So for this, we need to make some changes in terms of Q-value.

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.

Q-Learning

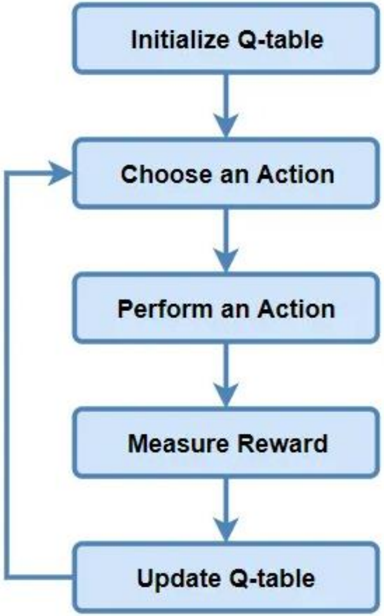
- **Model-free:**
  - In Q-learning, being model-free means that the algorithm doesn't require knowledge of the underlying dynamics or transition probabilities of the environment.
- **Value-based:**
  - Q-learning is a value-based method because it focuses on learning the value function Q(s,a), which estimates the expected cumulative reward of taking action a in state s and then following the optimal policy thereafter.
- **Off-policy**
  - off-policy indicates that it updates its Q-values using data generated by following a different policy, typically an exploratory one.

Q-Learning

Q-table

	south	north	east	west	pickup	dropoff
1	0	0	0			
2						
3						
...						

Q-Learning


Initialize Q-table
Choose an Action
Perform an Action
Measure Reward
Update Q-table

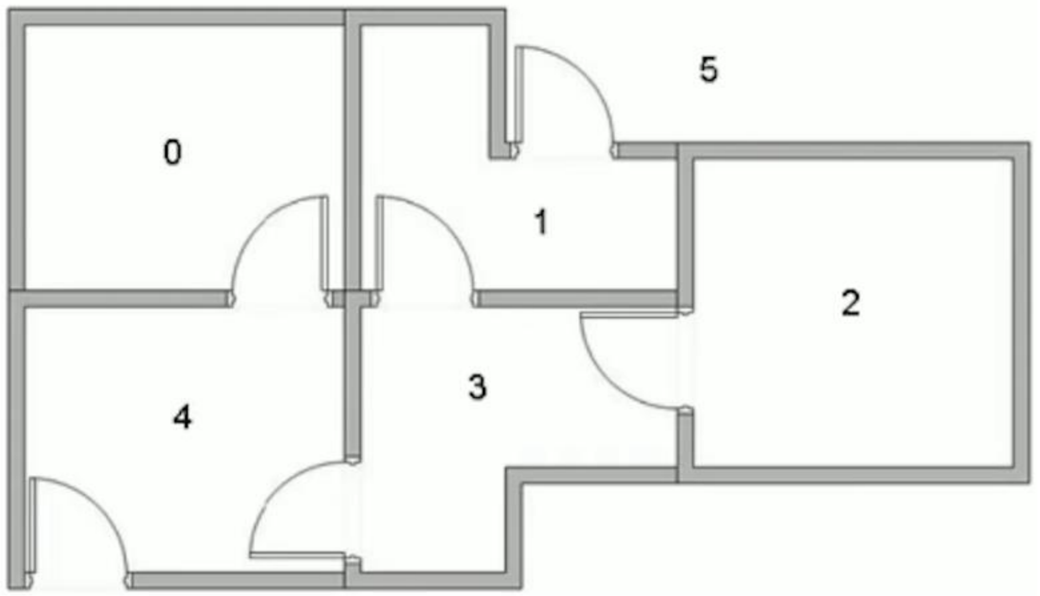
Q-Learning algorithm

- For each s, a initialize the table entry $\hat{Q}$(s, a) to zero.
- Observe the current state s
- Do forever:
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'
  - Update the table entry for $\hat{Q}$ (s, a) as follows:

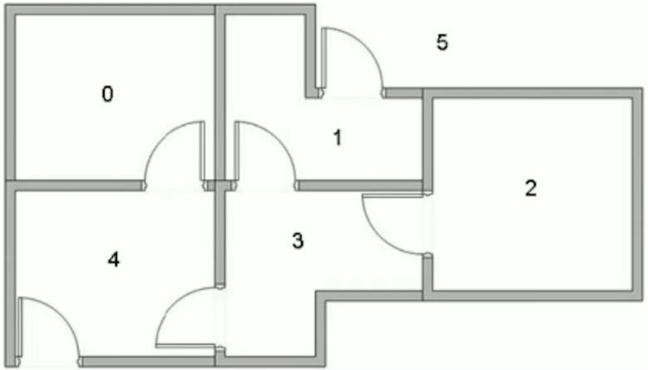
$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

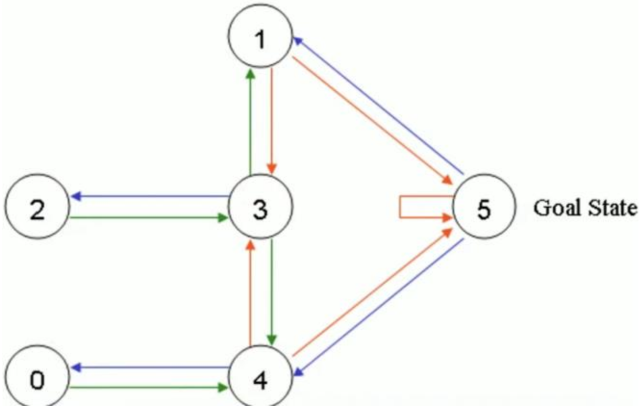
  - S ← s'

Q-Learning

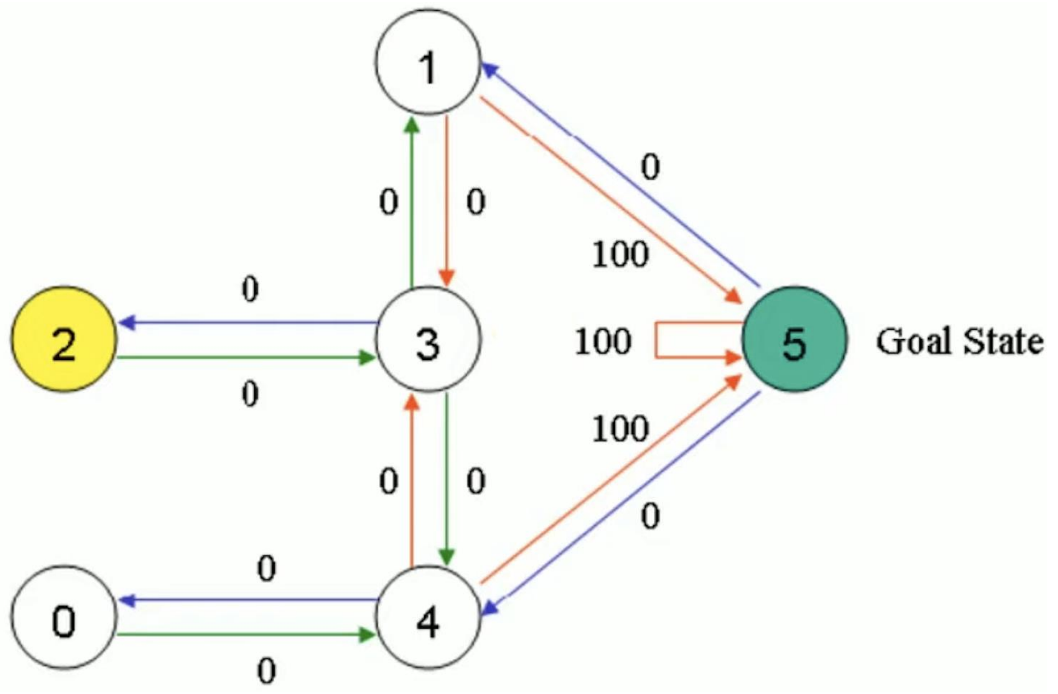


Q-Learning

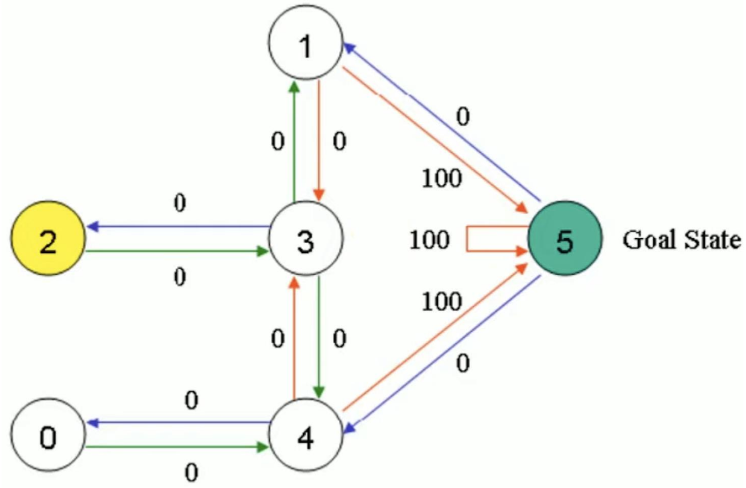




Q-Learning



Q-Learning



$$
R = \begin{array}{c c}
 & \text{Action} \\
\text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix}
\end{array}
$$

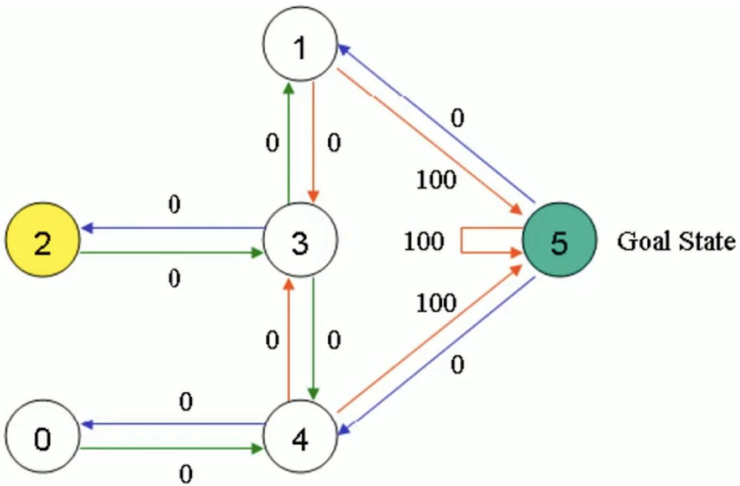
Q-Learning

$$
R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix}
\end{array}
$$

$$
Q = \begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Q-Learning

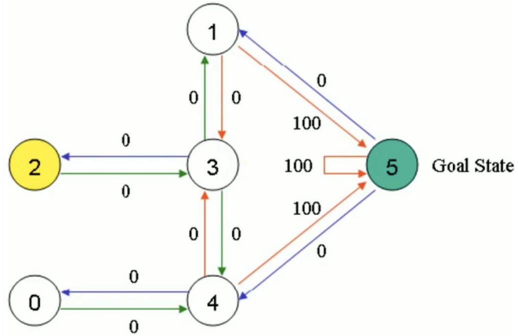
- Let's take the state 1 as start state
- Look at the second row (state 1) of matrix R.
- There are two possible actions for the current state 1: go to state 3, or go to state 5.
- By random selection, we select to go to 5 as our action.



$$
R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}
$$

Q-Learning

- Now let's imagine what would happen if our agent were in state 5 (next state).
- Look at the sixth row of the reward matrix R (i.e. state 5).
- It has 3 possible actions: go to state 1, 4 or 5.
- Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
- Q(1,5)= R(1, 5) + 0.8* Max[Q(5, 1), Q(5, 4), Q(5,5)]= 100+ 0.8* 0= 100



$$
R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}
\qquad
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}
$$

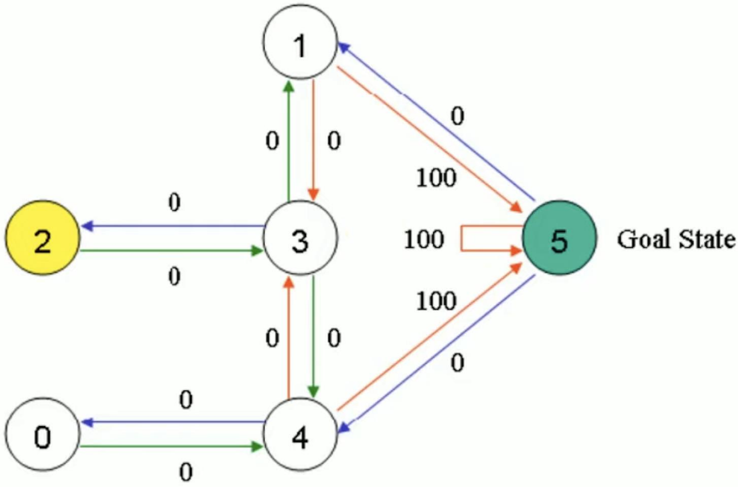
Q-Learning

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-Learning

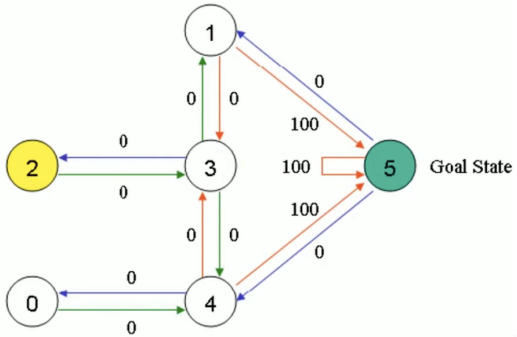
- For the next episode, we randomly choose the initial state - say 3 (can go to 1, 2 & 4)



$$
R = \begin{array}{c c}
 & \text{Action} \\
\text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix}
\end{array}
$$

Q-Learning

- Now we imagine that we are in state 1 (next state).
- Look at the second row of reward matrix R (i.e. state 1).
- It has 2 possible actions: go to state 3 or state 5.
- Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
- Q(3, 1)= R(3, 1) + 0.8* Max[Q(1, 3), Q(1,5)]= 0+ 0.8 * Max(0, 100) = 80



$$R = \begin{array}{c|c} & \text{Action} \\ \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c|c} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-Learning

$$R = \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \textbf{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Q-Learning

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$