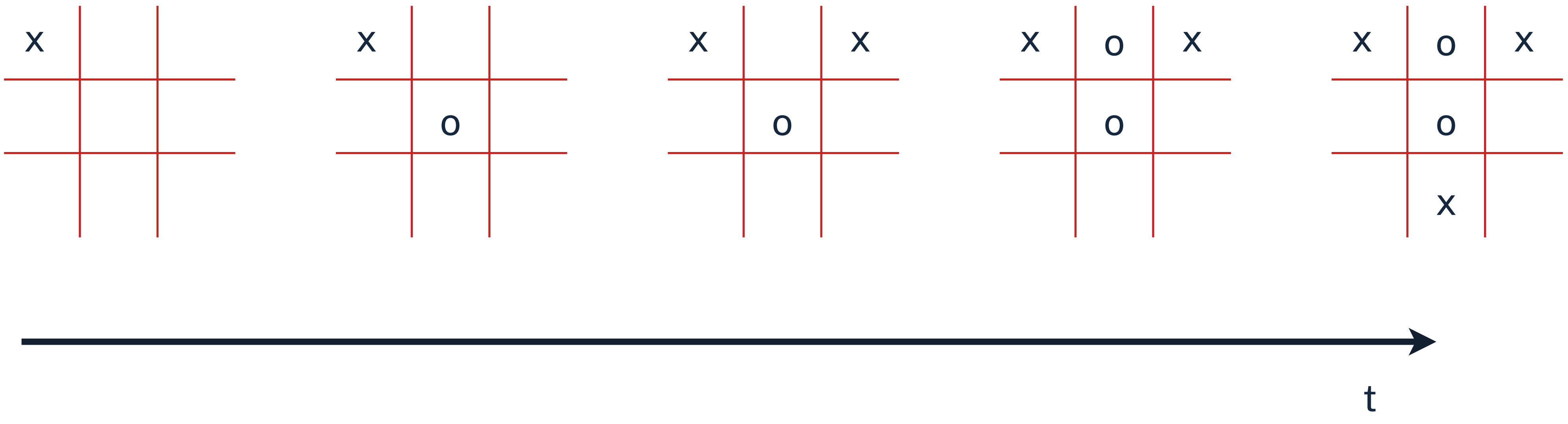
# Reinforcement Learning

# Reinforcement Learning

- Introduction
- Markov Decision Process
- Return, Policy, Value (state value, state-action value)
- RL Categories (model based/free, prediction/control)
- Q Learning
  - Q-Table
  - DQN

# Introduction

- Tic-Tac-Toe: a sequences of decisions



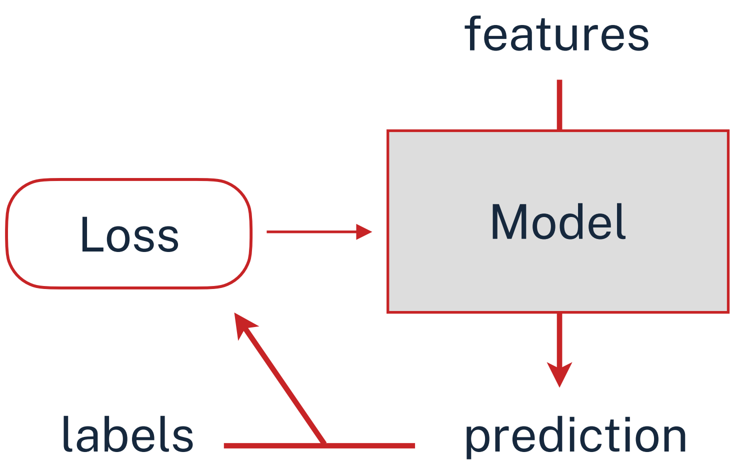

- What’s the winning strategy?

# Related Tasks

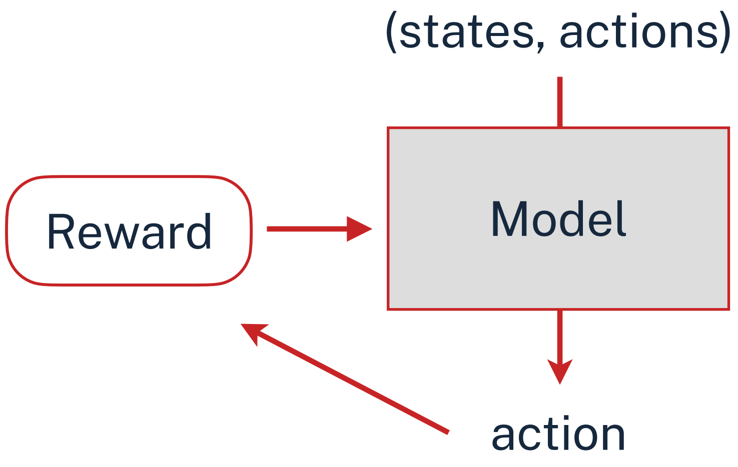
- Robotics:
  - driving
  - limb movement
  - grabbing
- Gaming:
  - Atari
  - Tetris
  - chess
  - Go
  - ... StarCraft :-)

# Different Paradigms

## supervised learning




## reinforcement learning

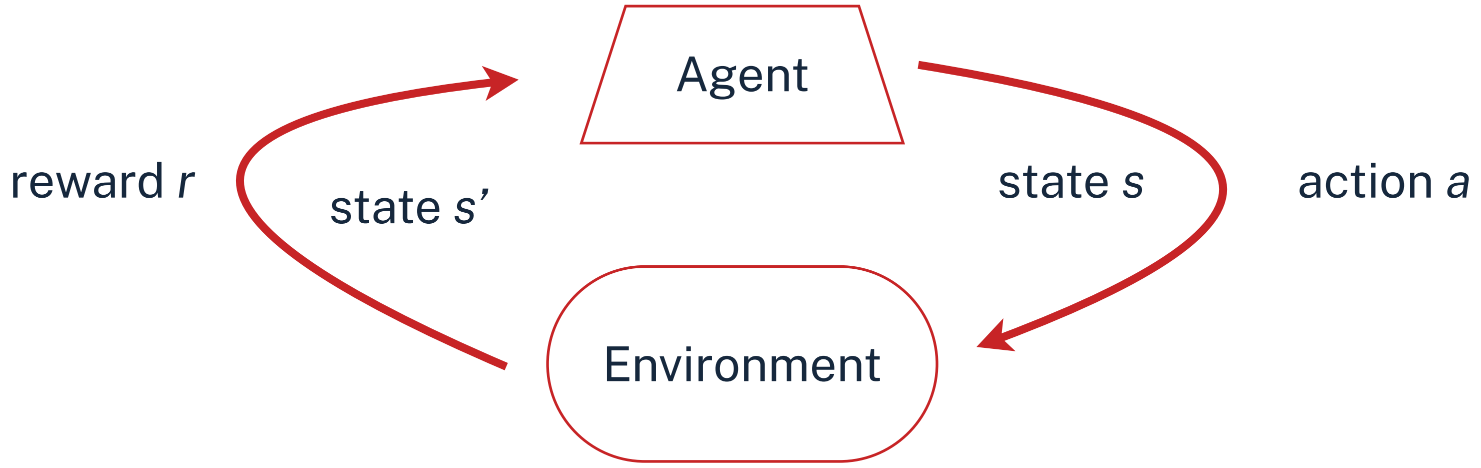


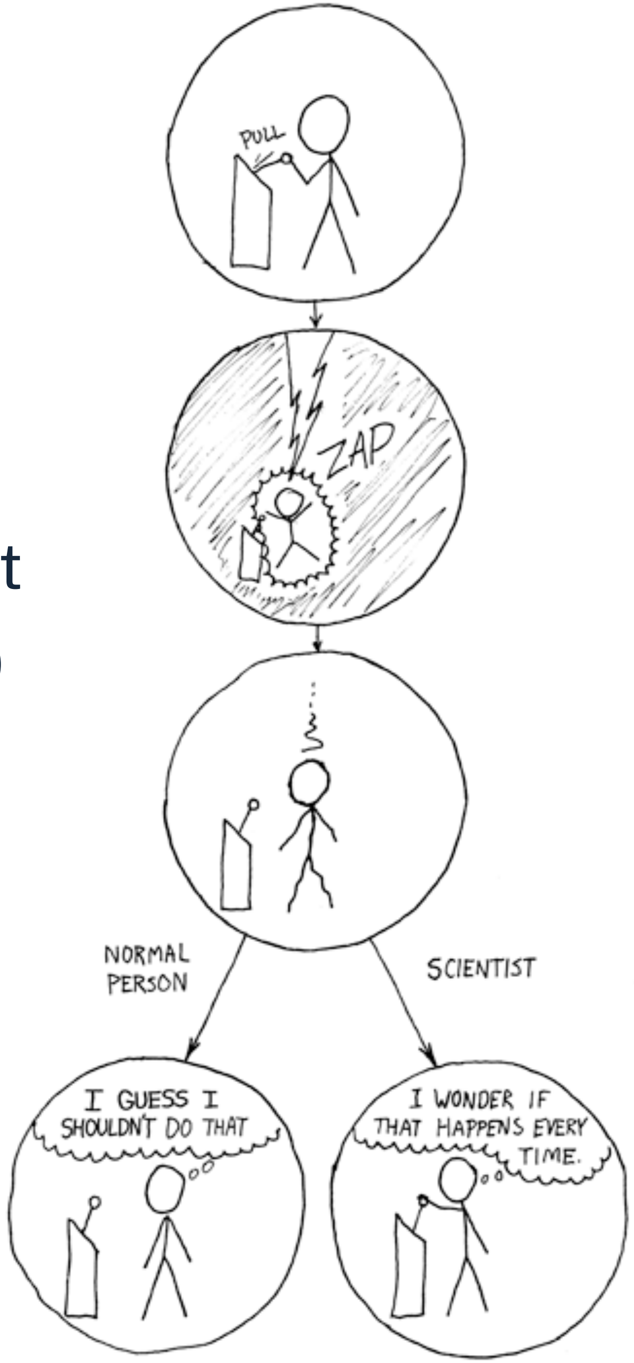
# RL vs. Supervised Learning

- No labels to compute loss (instead: reward)
- (Typically) No large pre-recorded dataset but interaction with environment
- Sequence of decisions, incorporating the changing environment

# Basic Principle

- Learn through trial and error (much like an infant…)
  - Try some action
  - Receive some feedback/reward from the environment
  - Repeat process until converging to positive results :-)




https://xkcd.com/242/

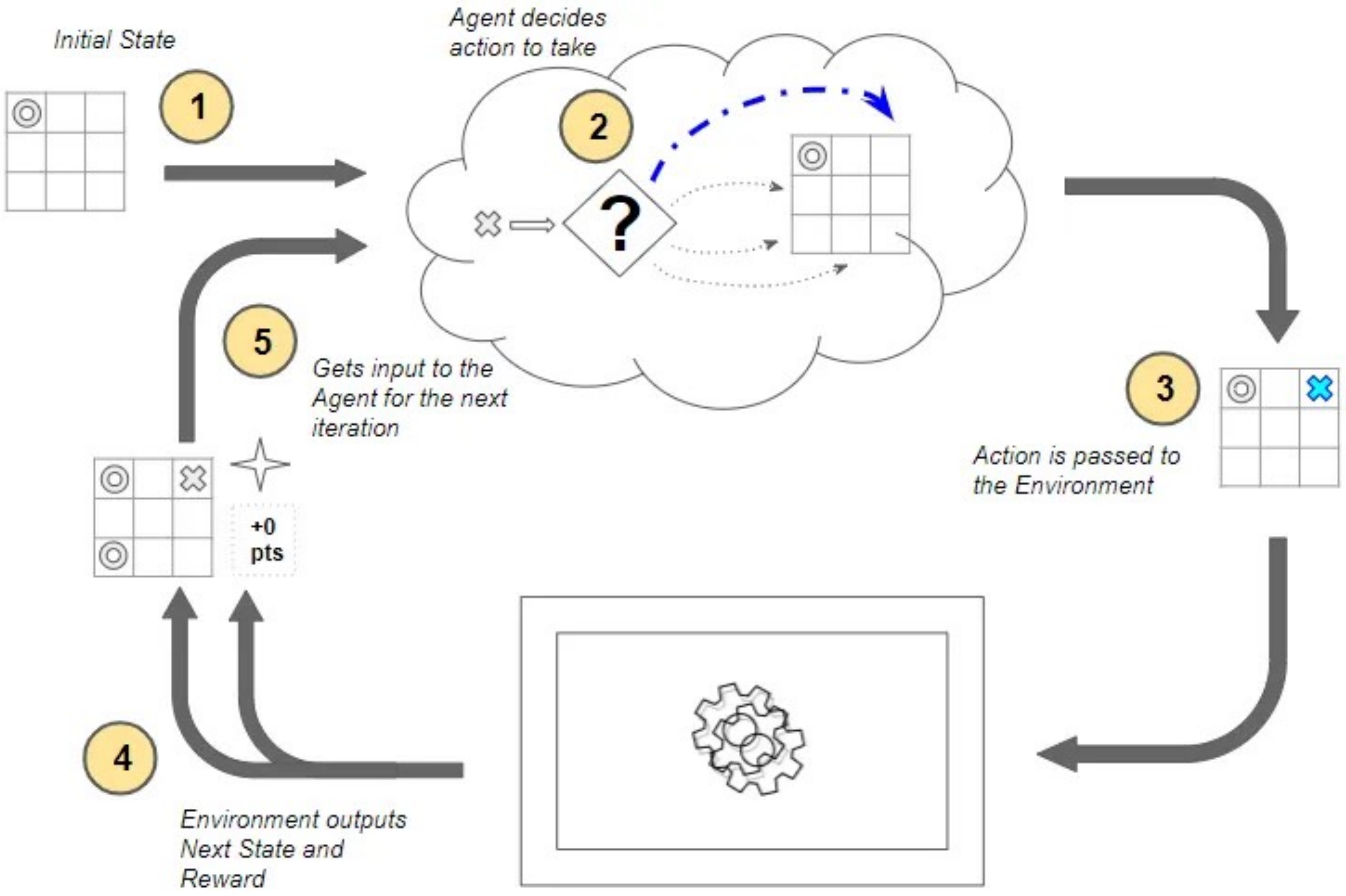
ohm

# Key Concepts

- **Agent**: the system that we try to learn
- **Environment**: the real-world environment that the agent operates in
- **State**: representation of the current state of the environment. This could be a finite set or in infinite space
- **Action**: set of actions that the agent can perform to alter the environment
- **Reward**: positive or negative reinforcement following the action

# Example: Tic Tac Toe

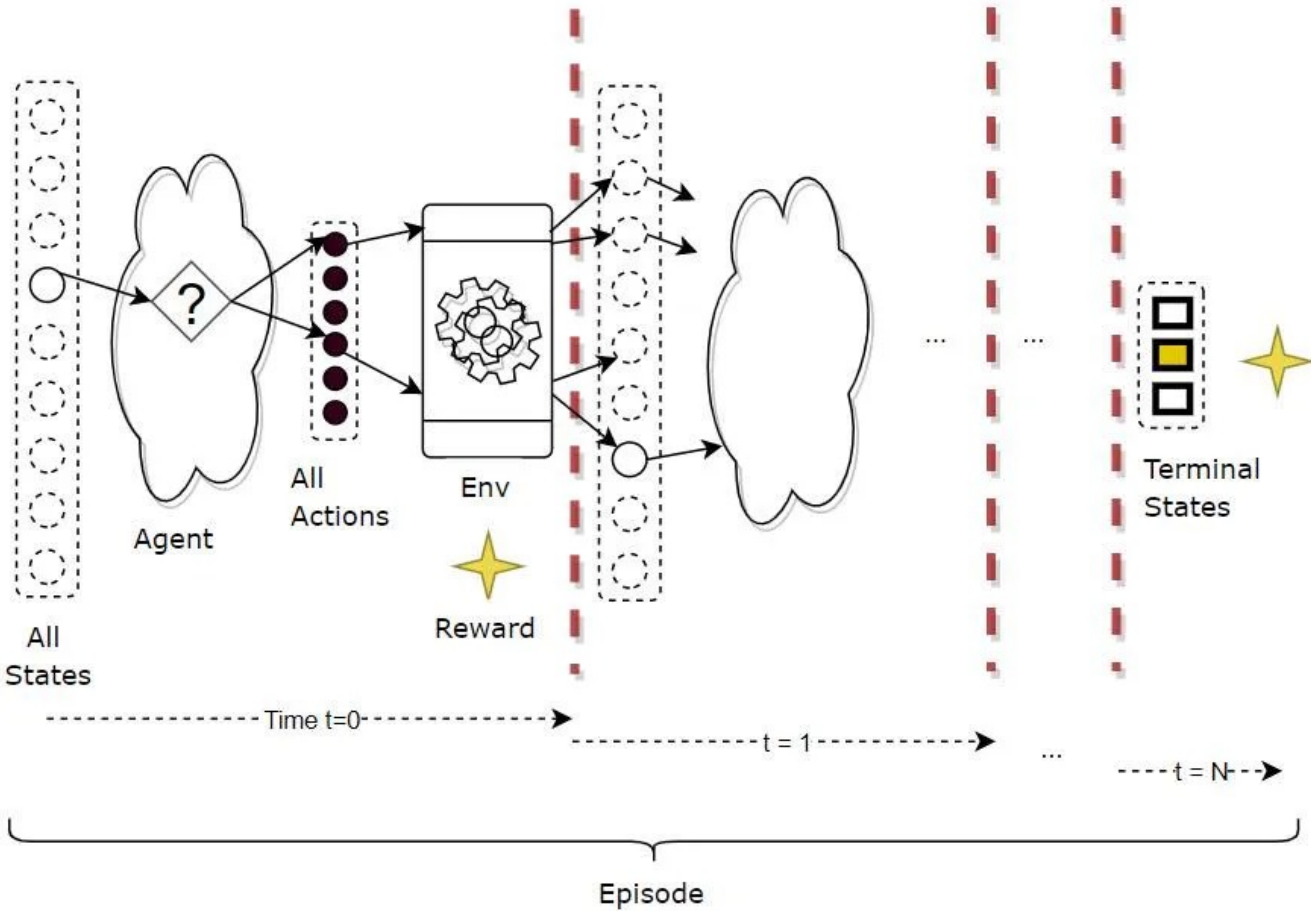
- Agent?
- Environment/State?
- Actions?
- Reward?



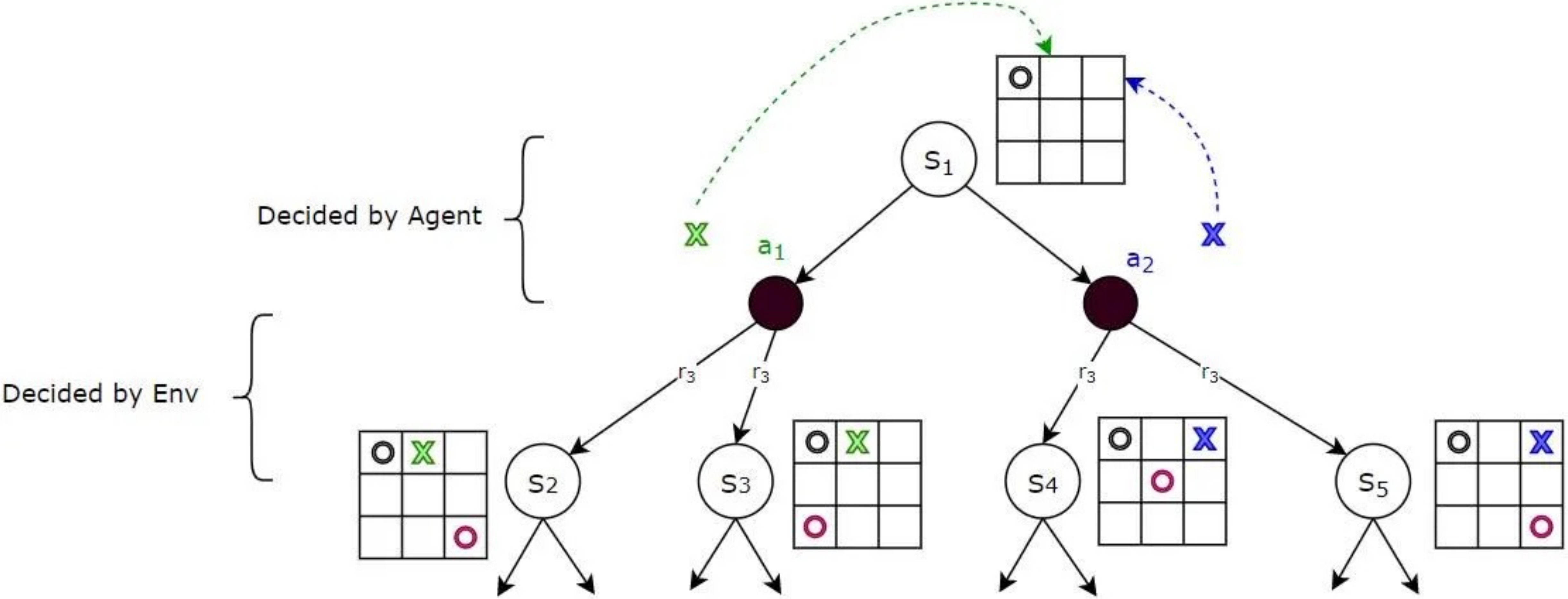

https://towardsdatascience.com/reinforcement-learning-made-simple-part-1-intro-to-basic-concepts-and-terminology-1d2a87aa060

https://towardsdatascience.com/reinforcement-learning-made-simple-part-1-intro-to-basic-concepts-and-terminology-1d2a87aa060

# Markov Decision Process: Visually



# Agent and Environment



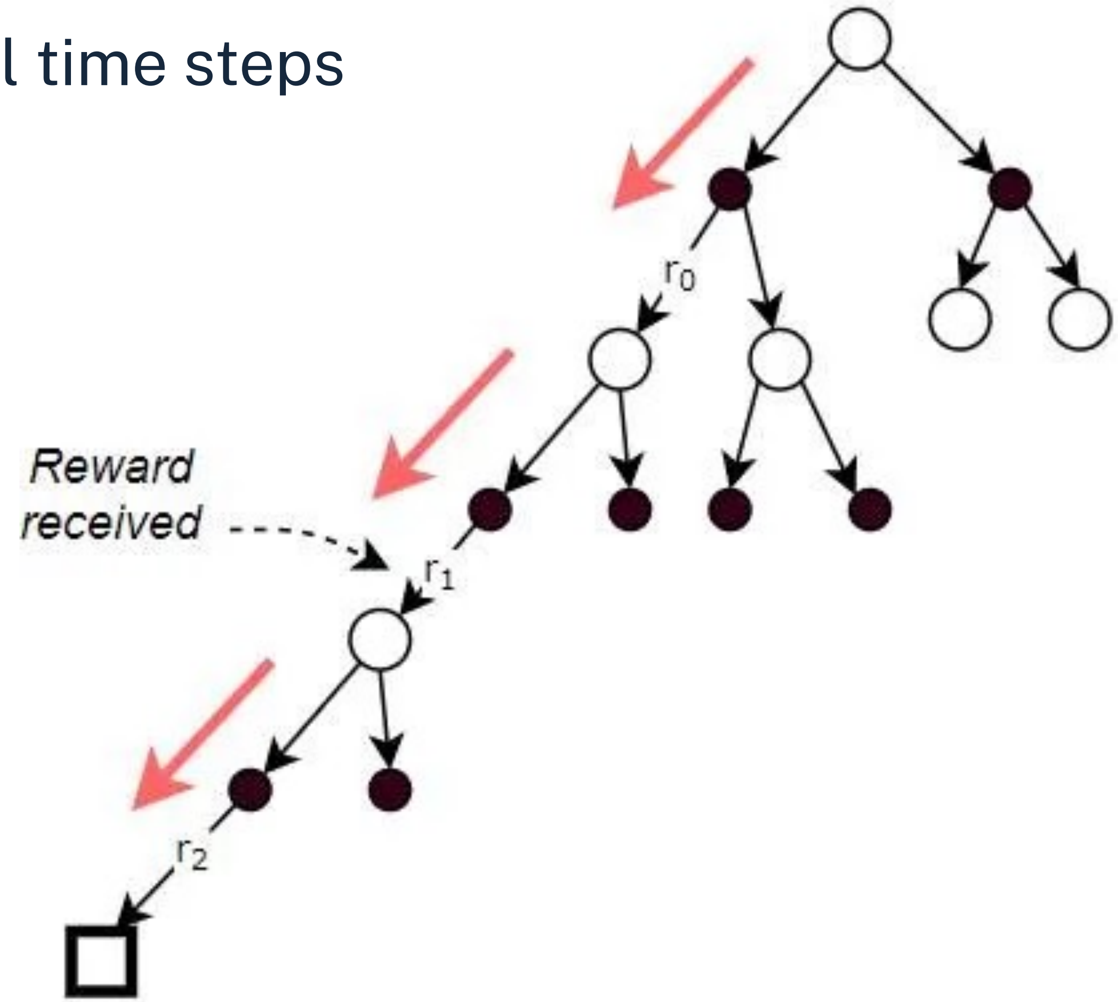

https://towardsdatascience.com/reinforcement-learning-made-simple-part-1-intro-to-basic-concepts-and-terminology-1d2a87aa060

# Recap: Markov Decision Processes

- The environment is represented as a Markov decision process (MDP) $\mathcal{M}$.
- Markov assumption: all relevant information is encapsulated in the current state
- Components of an MDP:
  - initial state distribution $p(\mathbf{s}_0)$
  - transition distribution $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$
  - reward function $r(\mathbf{s}_t, \mathbf{a}_t)$
- policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)$ parameterized by $\boldsymbol{\theta}$
- Assume a fully observable environment, i.e. $\mathbf{s}_t$ can be observed directly

# Key Concepts: Return

- Total reward over all time steps



# Finite and Infinite Horizon

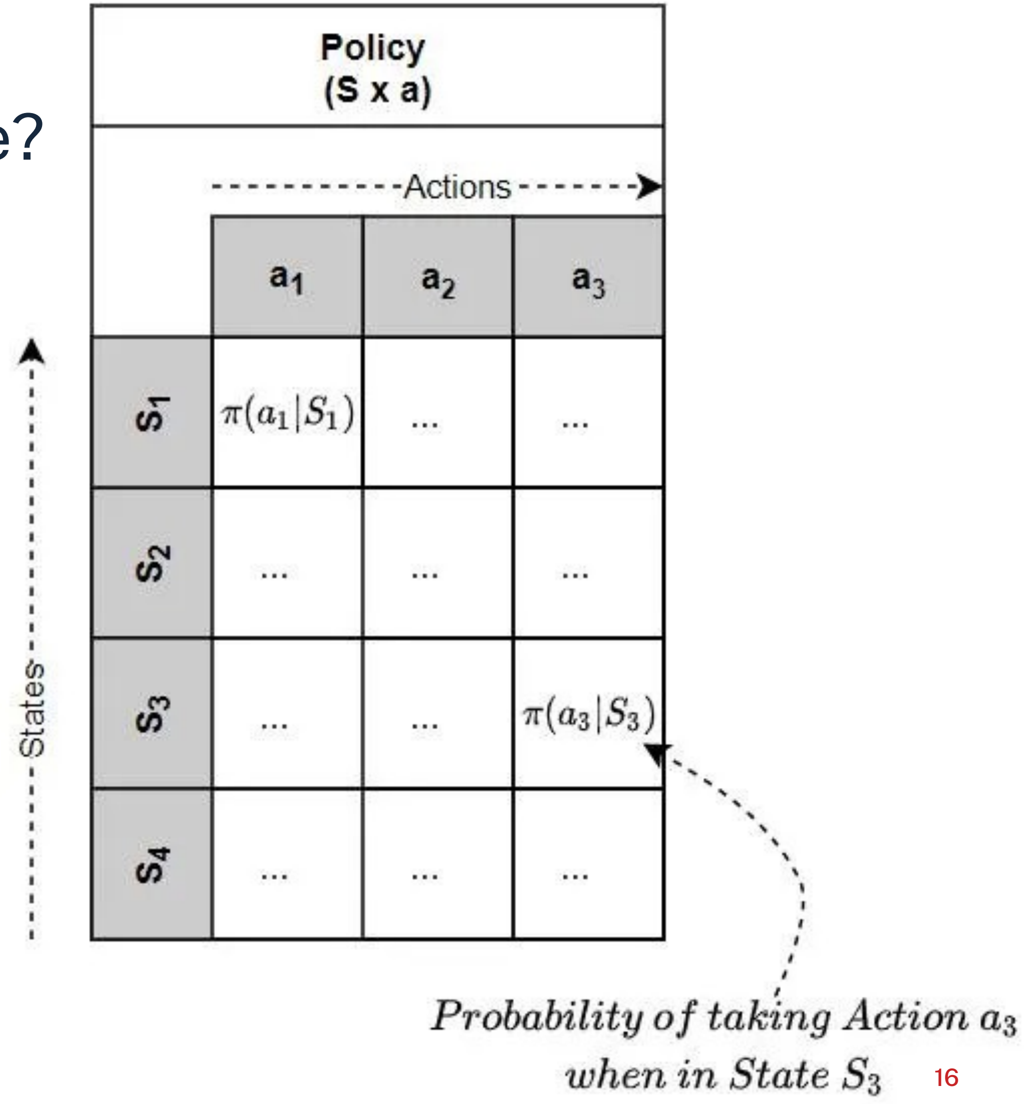
- assume infinite horizon
  - We can't sum infinitely many rewards, so we need to discount them: \$100 a year from now is worth less than \$100 today
  - Discounted return

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

  - Want to choose an action to maximize expected discounted return
  - The parameter $\gamma < 1$ is called the discount factor
    - small $\gamma =$ myopic
    - large $\gamma =$ farsighted

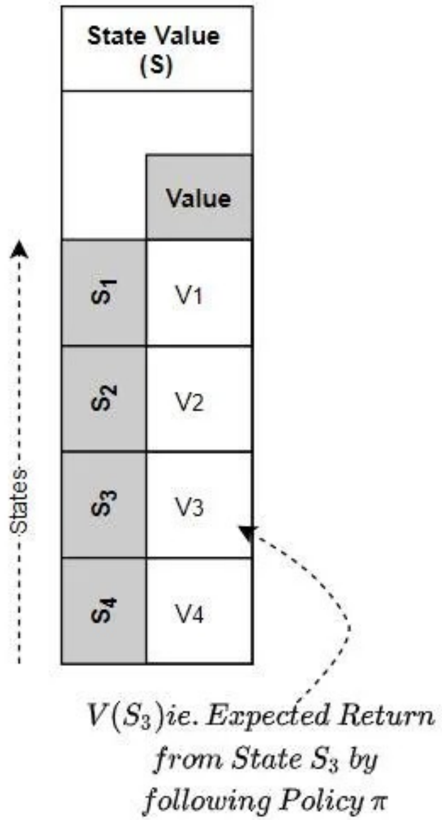
# Key Concept: Policy

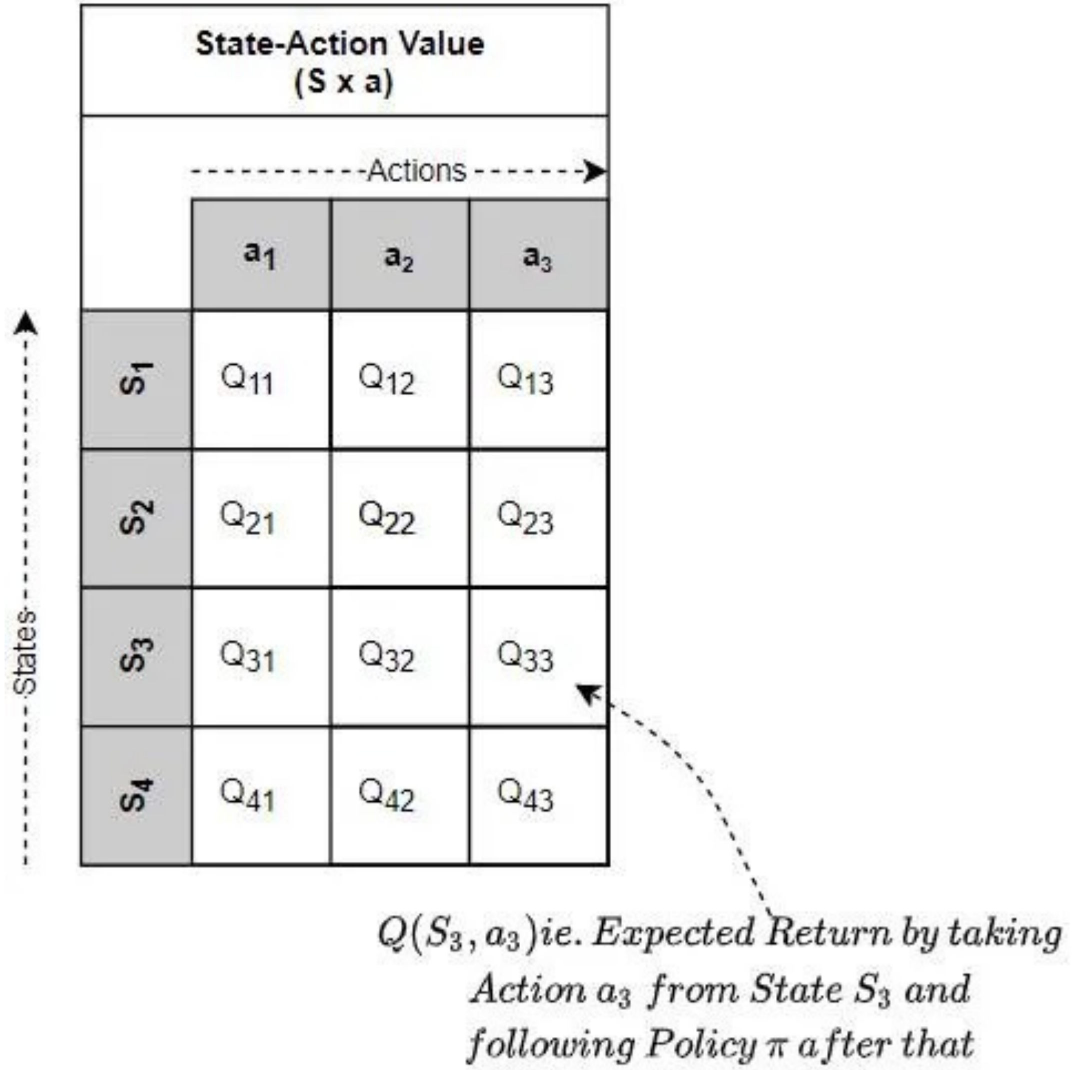
- How does the agent decide which action to take?
- Examples
  - Always pick random action
  - Always pick action to reach the next state with highest reward
  - Avoid negative reward
  - …
- If states and actions are finite: look-up table



# Key Concept: Value

- Expected return following a certain policy
- State Value vs. State-Action Value



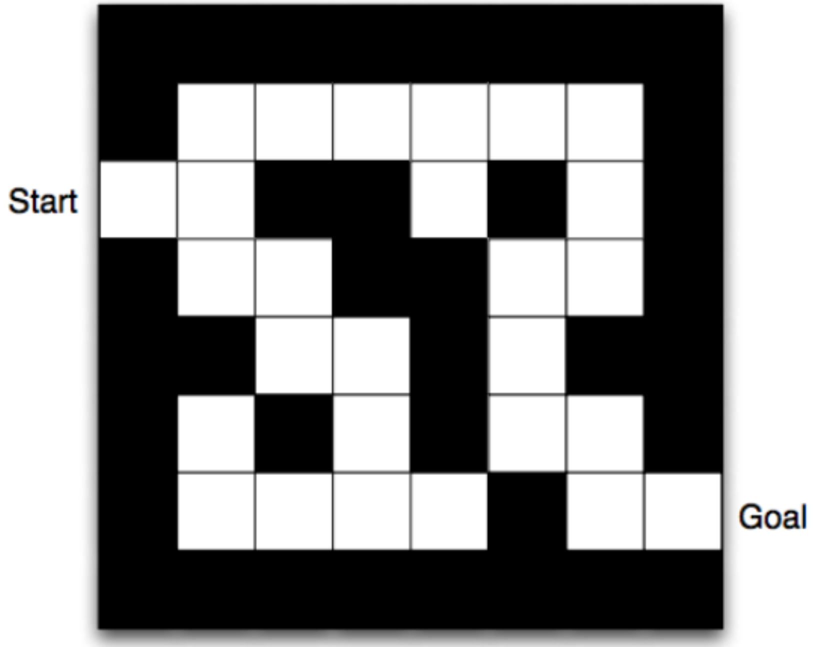
# Value Function

- Value function $V^\pi(\mathbf{s})$ of a state $\mathbf{s}$ under policy $\pi$: the expected discounted return if we start in $\mathbf{s}$ and follow $\pi$

$$V^\pi(\mathbf{s}) = \mathbb{E}[G_t \mid \mathbf{s}_t = \mathbf{s}]$$
$$= \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid \mathbf{s}_t = \mathbf{s}\right]$$

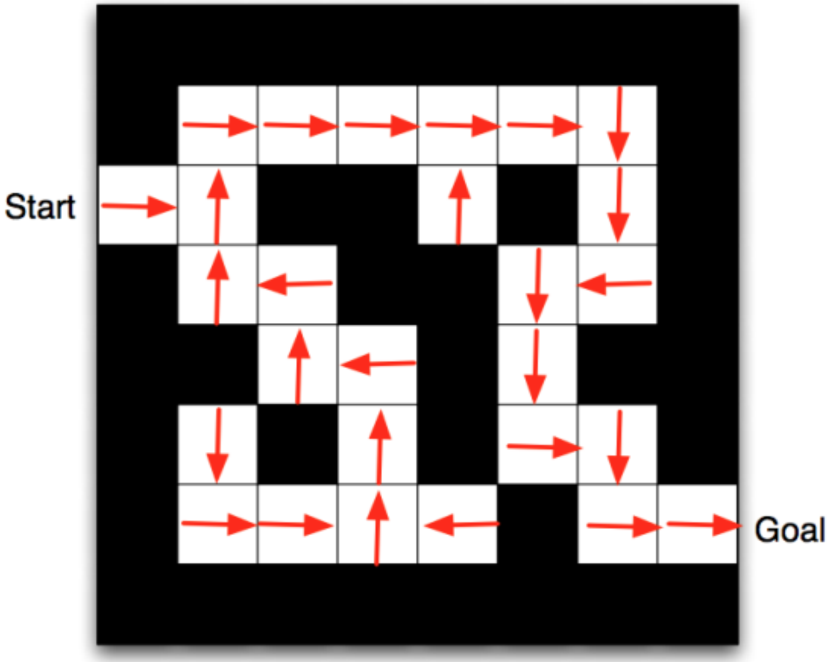
- Computing the value function is generally impractical, but we can try to approximate (learn) it
- The benefit is credit assignment: see directly how an action affects future returns rather than wait for rollouts

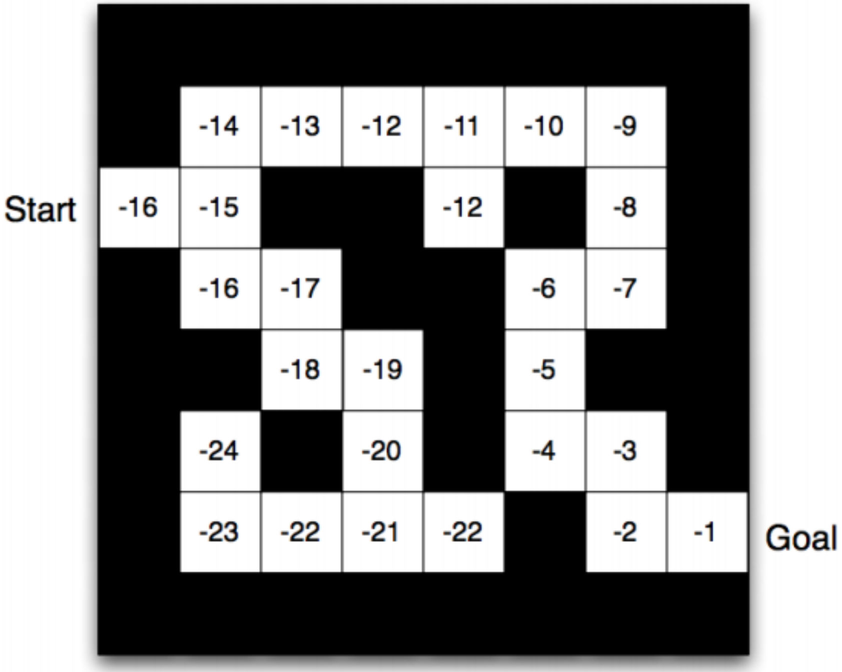
# Value Function




- Rewards: -1 per time step
- Undiscounted ($\gamma = 1$)
- Actions: N, E, S, W
- State: current location

# Value Function



# Action-Value Function

- Can we use a value function to choose actions?

$$\arg\max_{\mathbf{a}} r(\mathbf{s}_t, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)}[V^\pi(\mathbf{s}_{t+1})]$$

- Problem: this requires taking the expectation with respect to the environment's dynamics, which we don't have direct access to!
- Instead learn an action-value function, or Q-function: expected returns if you take action **a** and then follow your policy

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}[G_t \mid \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}]$$

- Relationship:

$$V^\pi(\mathbf{s}) = \sum_{\mathbf{a}} \pi(\mathbf{a} \mid \mathbf{s}) Q^\pi(\mathbf{s}, \mathbf{a})$$

- Optimal action:

$$\arg\max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$$

# Relationship Reward, Return and Value

- *Reward* is the immediate reward obtained for a single action.
- *Return* is the total of all the discounted rewards obtained till the end of that episode.
- *Value* is the expected return over many episodes

# Bellman Equation

- The Bellman Equation is a recursive formula for the action-value function:

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \, \pi(\mathbf{a}' \mid \mathbf{s}')}[Q^{\pi}(\mathbf{s}', \mathbf{a}')]$$

- There are various Bellman equations, and most RL algorithms are based on repeatedly applying one of them.

# Optimal Bellman Equation

- The optimal policy $\pi^*$ is the one that maximizes the expected discounted return, and the optimal action-value function $Q^*$ is the action-value function for $\pi^*$.
- The Optimal Bellman Equation gives a recursive formula for $Q^*$:

$$Q^*(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})} \left[ \max_{\mathbf{a}'} Q^*(\mathbf{s}_{t+1}, \mathbf{a}') \mid \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a} \right]$$

- This system of equations characterizes the optimal action-value function. So maybe we can approximate $Q^*$ by trying to solve the optimal Bellman equation!

# Q-Learning

- Let $Q$ be an action-value function which hopefully approximates $Q^*$.
- The Bellman error is the update to our expected return when we observe the next state $\mathbf{s}'$.

$$\underbrace{r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a})}_{\text{inside } \mathbb{E} \text{ in RHS of Bellman eqn}} - Q(\mathbf{s}_t, \mathbf{a}_t)$$

- The Bellman equation says the Bellman error is 0 in expectation
- Q-learning is an algorithm that repeatedly adjusts $Q$ to minimize the Bellman error
- Each time we sample consecutive states and actions $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$:

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha \underbrace{\left[ r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a}) - Q(\mathbf{s}_t, \mathbf{a}_t) \right]}_{\text{Bellman error}}$$

# Exploration-Exploitation Tradeoff

- Notice: Q-learning only learns about the states and actions it visits.
- Exploration-exploitation tradeoff: the agent should sometimes pick suboptimal actions in order to visit new states and actions.
- Simple solution: $\epsilon$-greedy policy
  - With probability $1 - \epsilon$, choose the optimal action according to $Q$
  - With probability $\epsilon$, choose a random action
- Believe it or not, $\epsilon$-greedy is still used today!

# Exploration-Exploitation Tradeoff

- Q-learning is an off-policy algorithm: the agent can learn $Q$ regardless of whether it's actually following the optimal policy
- Hence, Q-learning is typically done with an $\epsilon$-greedy policy, or some other policy that encourages exploration.

# Q-Learning

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;
    until $S$ is terminal

# Q Learning Visually

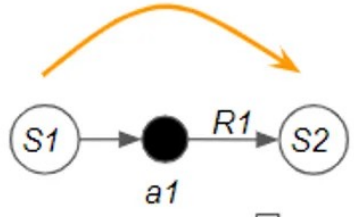
**1** Initialise Q-Value (ie. State Action value) estimates with zero value, and pick the initial state.

| | a1 | a2 | a3 | a4 |
|---|---|---|---|---|
| S1 | 0 | 0 | 0 | 0 |
| S2 | 0 | 0 | 0 | 0 |
| S3 | 0 | 0 | 0 | 0 |
| S4 | 0 | 0 | 0 | 0 |
| S5 | 0 | 0 | 0 | 0 |

S1 + initial state

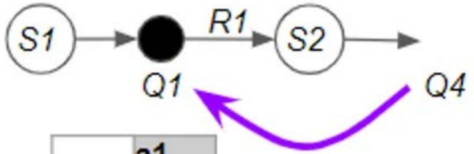
**2** Agent picks an **action to execute** from the current state using ε-greedy policy.

**3** Agent obtains observation data from the environment ( S1, a1, R1, S2 )




**4** Update the **current Q-value** using the observed reward and the **target Q-value.** The target Q-value is the action with the max Q-value from the next state.

$$Q(S,A) = Q(S,A) + \alpha(R + \gamma\, max\, Q(S',a) - Q(S,A))$$




| | a1 |
|---|---|
| S1 | q' |

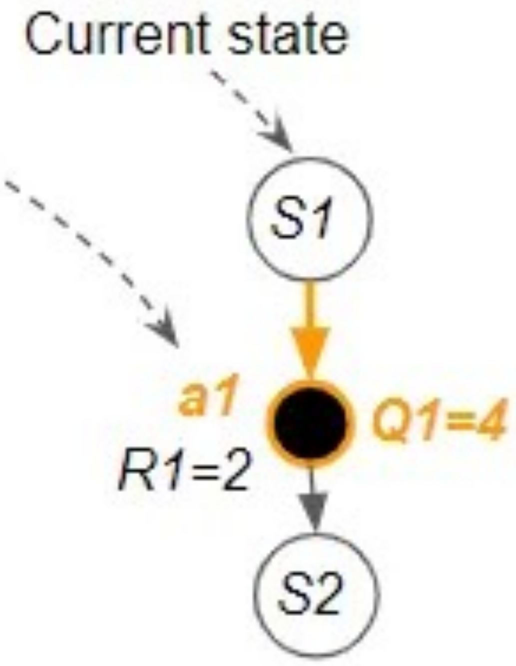
S2 Next state becomes the Current State

ohm

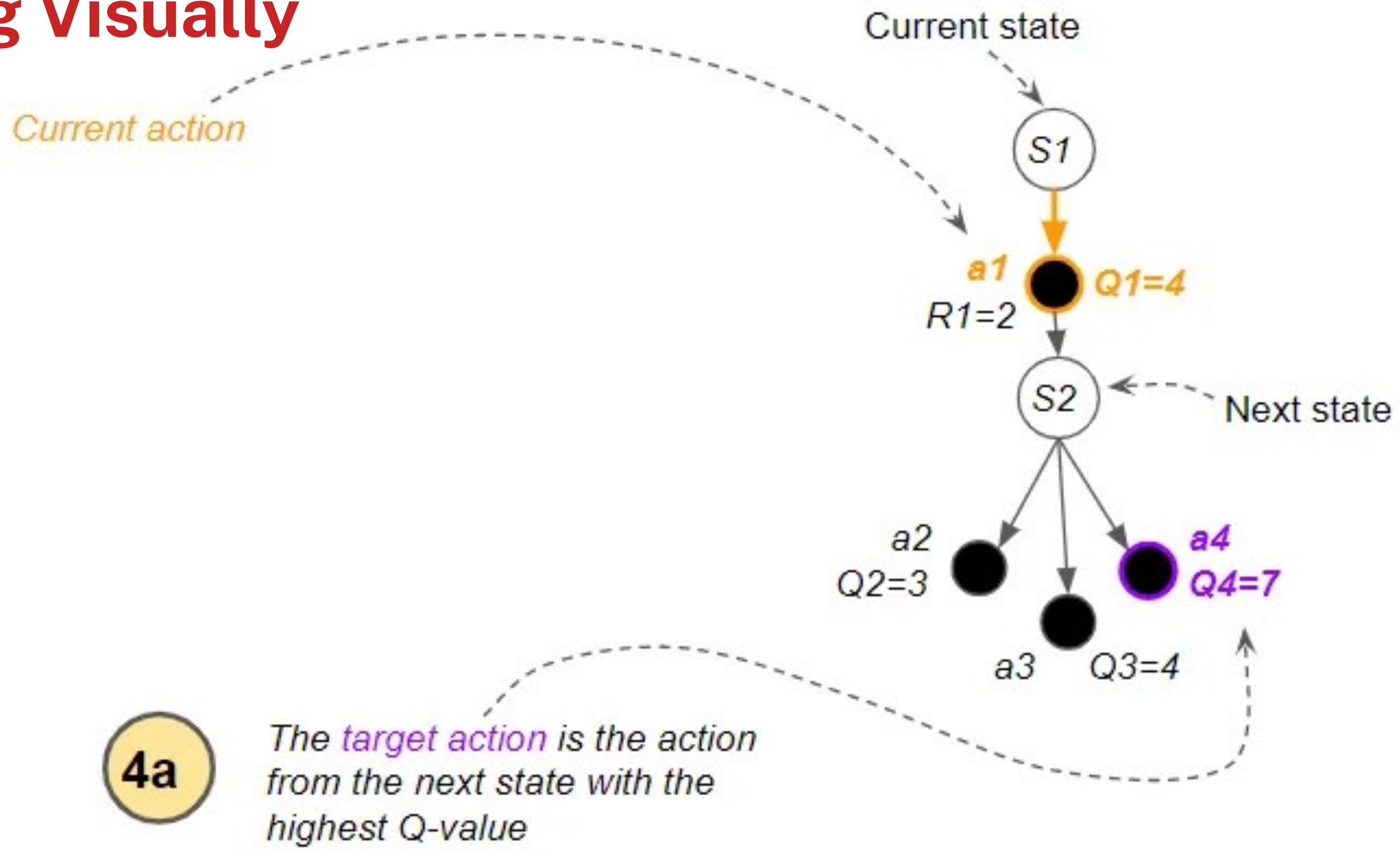
# Q-Learning Visually

**2** Current action picked using an ε-greedy policy.

| | a1 | a2 | a3 | a4 |
|---|---|---|---|---|
| **S1** | 4 | 9 | 2 | 3 |
| **S2** | 0 | 3 | 4 | 7 |

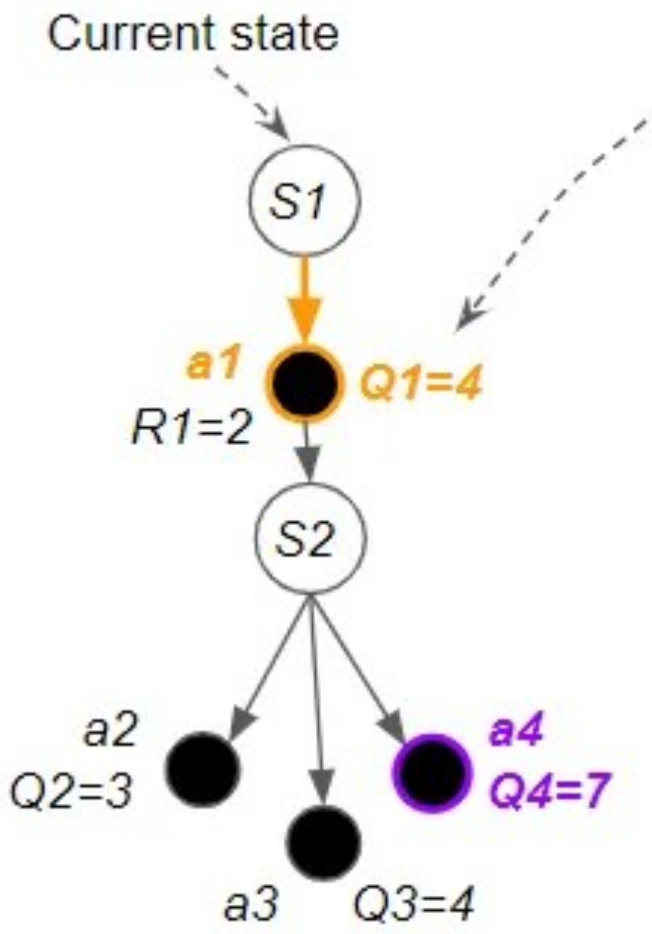


# Q-Learning Visually




| | a1 | a2 | a3 | a4 |
|---|---|---|---|---|
| S1 | 4 | 9 | 2 | 3 |
| S2 | 0 | 3 | 4 | 7 |

# Q-Learning Visually




**4b** The current Q-value is updated

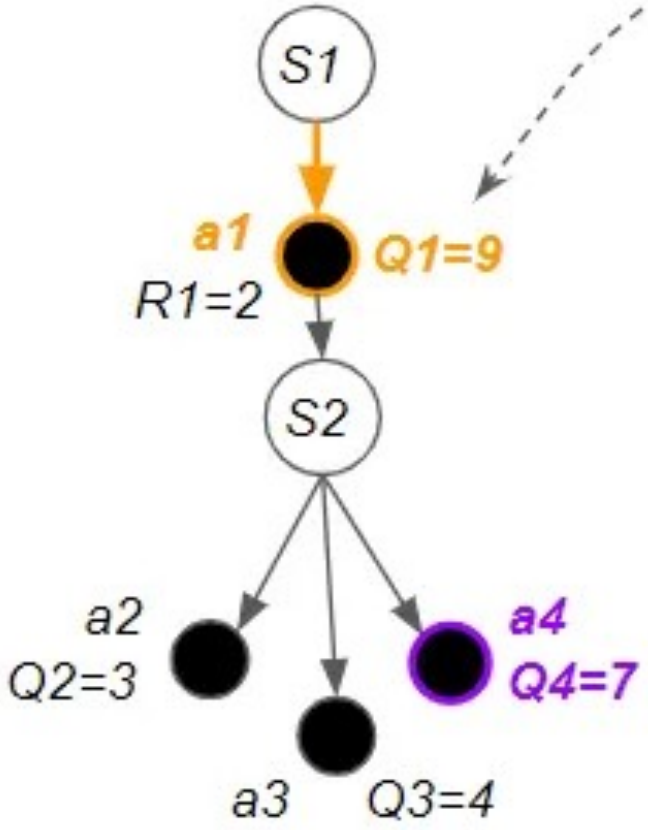
$$Q(S,A) = Q(S,A) + \alpha(R + \gamma \, max \, Q(S',a) - Q(S,A))$$

$Q1 = Q1 + \alpha \, (R1 + \gamma * Q4 - Q1)$

$Q1 = 4 + (2 + 7 - 4) = 9$

NB: taking $\alpha = \gamma = 1$ for simplicity

# Q-Learning Visually

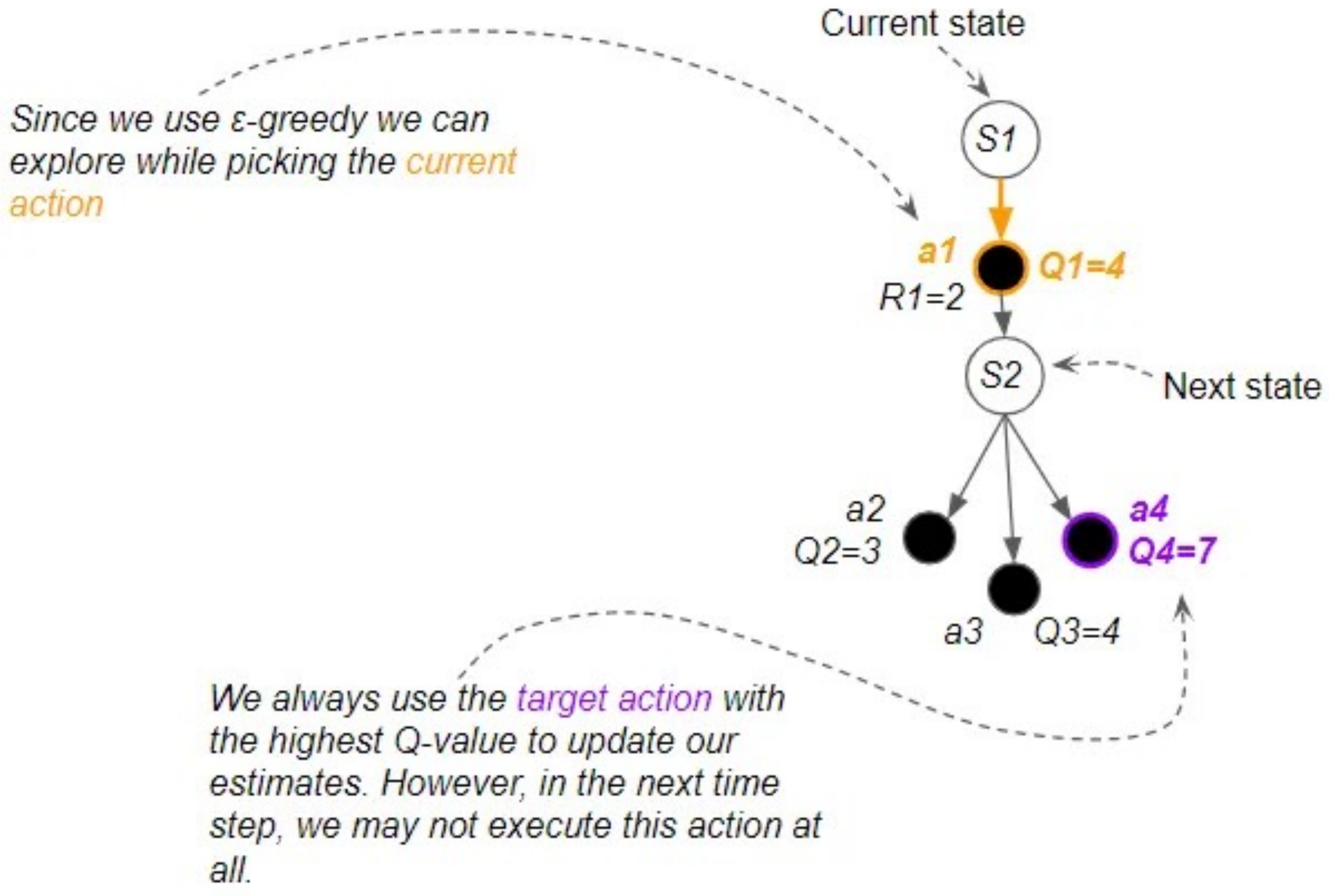



**4b** The current Q-value is updated

9

| | a1 | a2 | a3 | a4 |
|---|---|---|---|---|
| S1 | 4 | 9 | 2 | 3 |
| S2 | 0 | 3 | 4 | 7 |

# Q-Learning: Duality of Current and Target Action



# Q-Learning

- Initialize Q-Table with zeros
- Reward will “populate” the Q-Table (and propagate…)
- but…
  - Q-Table may become very large
  - States may not be enumerable (ie. discrete)

# Deep Q Networks (DQN)

- Recall Bellman's update rule

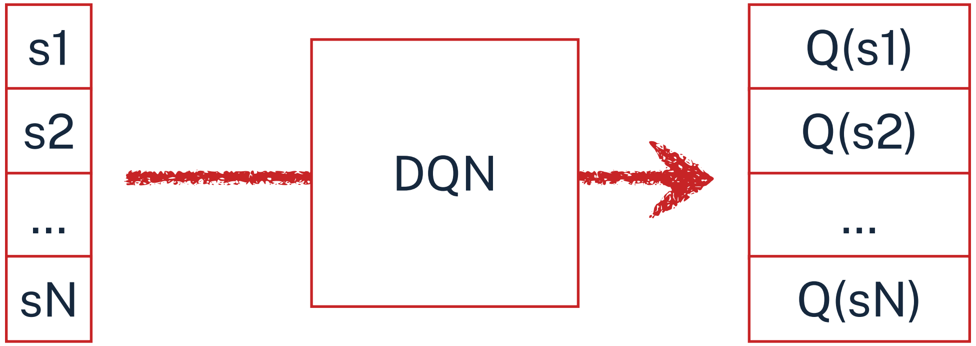
$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

- Replace Q-Table with function approximation (ie. a neural net)
- Idea: We're looking for an approximation where the above equation is true

$$Cost = \left[ Q(s,a;\theta) - \left( r(s,a) + \gamma \max_a Q(s',a;\theta) \right) \right]^2$$

# DQN: Modeling Choices (1)

- Technically, our DQN should map (state, action) → Q value
- Would require separate inference for each action
- Instead: Predict Q value for all actions at the output layer




- Train using backpropagation

https://arxiv.org/abs/1312.5602

ohm

# DQN: Modeling Choices (2)

- Mnih et al., 2013: DQN
  - CNN applied to 5 consecutive frames (downsampled to 84x84) to model state
  - 4-18 output values (one Q for each valid action)
  - experience replay buffer: cache $e_t = (s_t, a_t, r_t, s_{t+1})$ for efficient minibatch
  - Q/Q-Target networks, ie. keep target network constant

# Summary

- Reinforcement learning learns from sequences of actions and their reward
- Underlying theory
  - Markov Decision Process (MDP)
  - Bellman's equation
- Regular Q-Learning (Q-Table) requires coding of states and actions
- Deep Q-Learning (DQN) allows to use encoding layers (eg. CNN for images) to model state
- Experience replay helps to speed up minibatch training