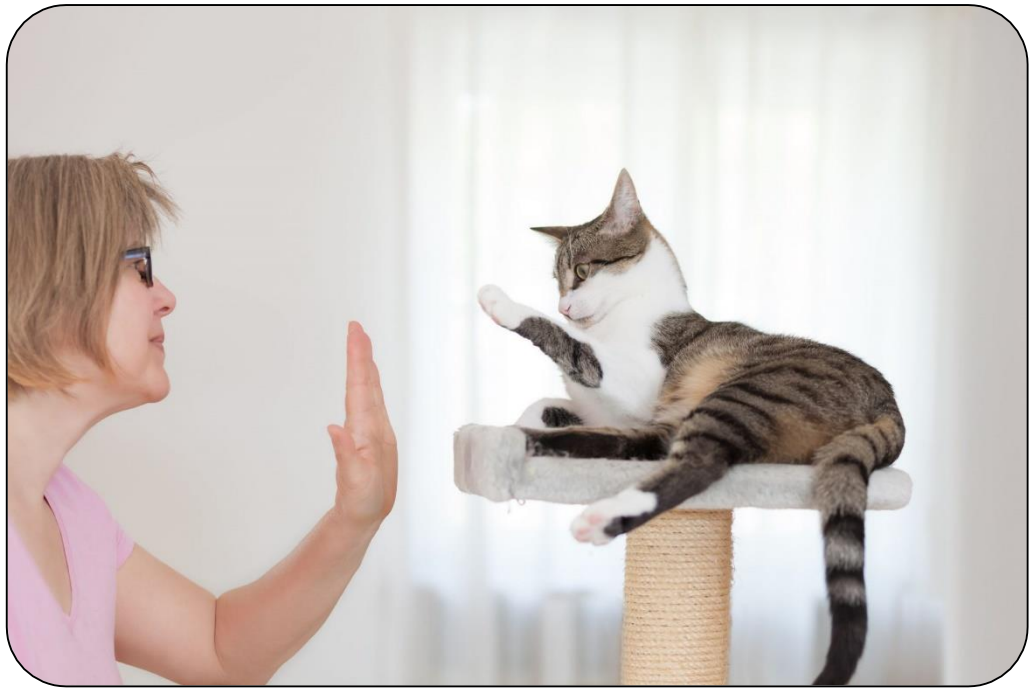
11. RL - Markov Decision Process

Game Engineering & XR Technologies
Prof. HyeongYeop Kang
siamiz@khu.ac.kr
IIIXR LAB

Introduction

Core Concept of Reinforcement Learning

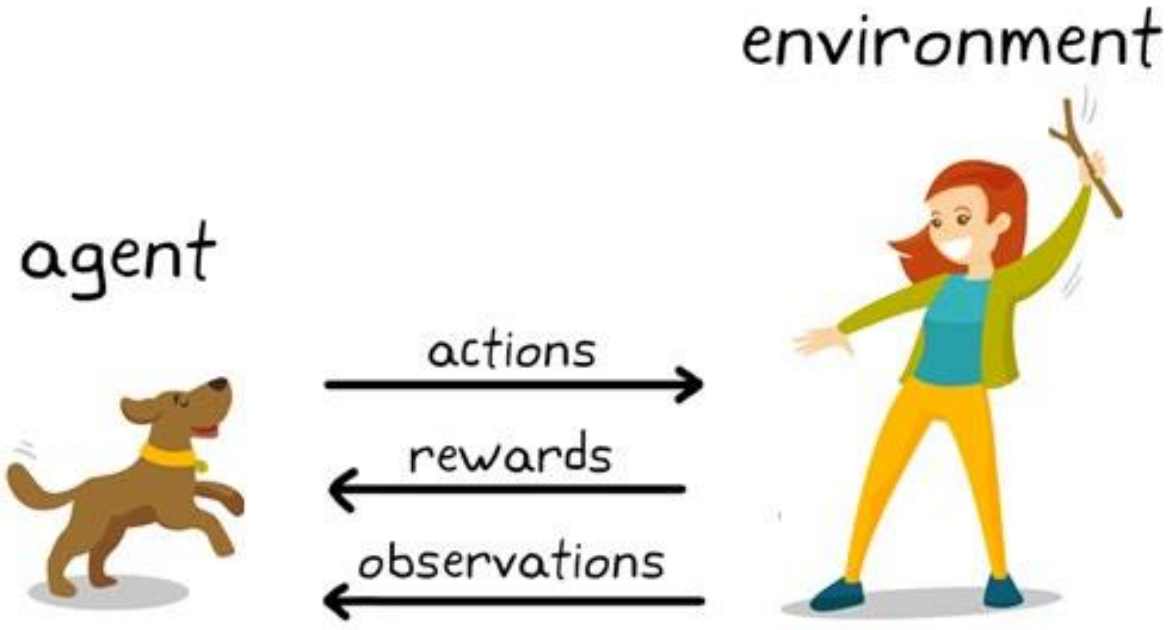
- The term "**reinforcement**" is oriented in the context of animal learning in psychology.
  - The concept is the strengthening of a behavior pattern when an animal receives a stimulus (reinforcement factor).
  - The trial-and-error method of reinforcement learning is called **pleasure-oriented**.



Introduction

Core Concept of Reinforcement Learning

- Reinforcement learning (RL) is a subfield of artificial intelligence (AI) and machine learning (ML).
  - RL focuses on training *agents* to make decisions by interacting with an environment.
  - An agent learns to take *action* in various situations (*environment* – *observed* by their own sensor) to achieve goals, typically by maximizing cumulative *rewards* over time.



Introduction

Core Concept of Reinforcement Learning

- The 7 key components of reinforcement learning include:
  1) **Agent**: The entity that makes decisions and takes actions in the environment.
  2) **Environment**: The context or world in which the agent operates, characterized by a set of states.
  3) **State**: A representation of the current situation or context in the environment.
  4) **Action**: A decision or move that the agent takes in response to a particular state.
  5) **Reward**: The feedback signal provided to the agent after taking an action, which indicates how good or bad the action was with respect to the goal.
  6) **Policy**: A mapping from states to actions that defines the agent's behavior. The policy can be deterministic (a specific action for each state) or stochastic (a probability distribution over possible actions).
  7) **Value function**: A function that estimates the expected cumulative reward for each state, given the agent's current policy.

Introduction

Core Concept of Reinforcement Learning

- The objective of RL is to learn an optimal policy:
  - Optimal policy is a strategy that maximizes the expected cumulative reward over time.
  - To achieve this, the agent needs to explore the environment by trying different actions, observe the outcomes (including rewards), and adjust its policy based on the observed consequences.
  - There are various algorithms and techniques for solving reinforcement learning problems, including model-based methods, model-free methods (e.g., Q-learning, SARSA), and policy gradient methods (e.g., REINFORCE, DDPG).

Introduction

Core Concept of Reinforcement Learning

- Does the agent explore the environment by trying different actions?
  - Based on the observation the agent receives from the environment at a given time, the agent selects an action.

- The reinforcement learning loop typically follows these steps:
  1) The agent observes the current state of the environment (observation).
  2) Based on the observation, the agent selects an action according to its current policy.
  3) The agent takes the chosen action, which alters the environment's state.
  4) The agent receives a reward and a new observation from the environment, reflecting the action’s consequences.
  5) The agent updates its knowledge (e.g., policy, value function, or action-value function) using the reward and observation information.
  6) The process repeats until a termination condition is met, such as reaching a goal or a maximum number of steps.

Finite Markov Decision Process

RL & Markov Decision Process (MDP)

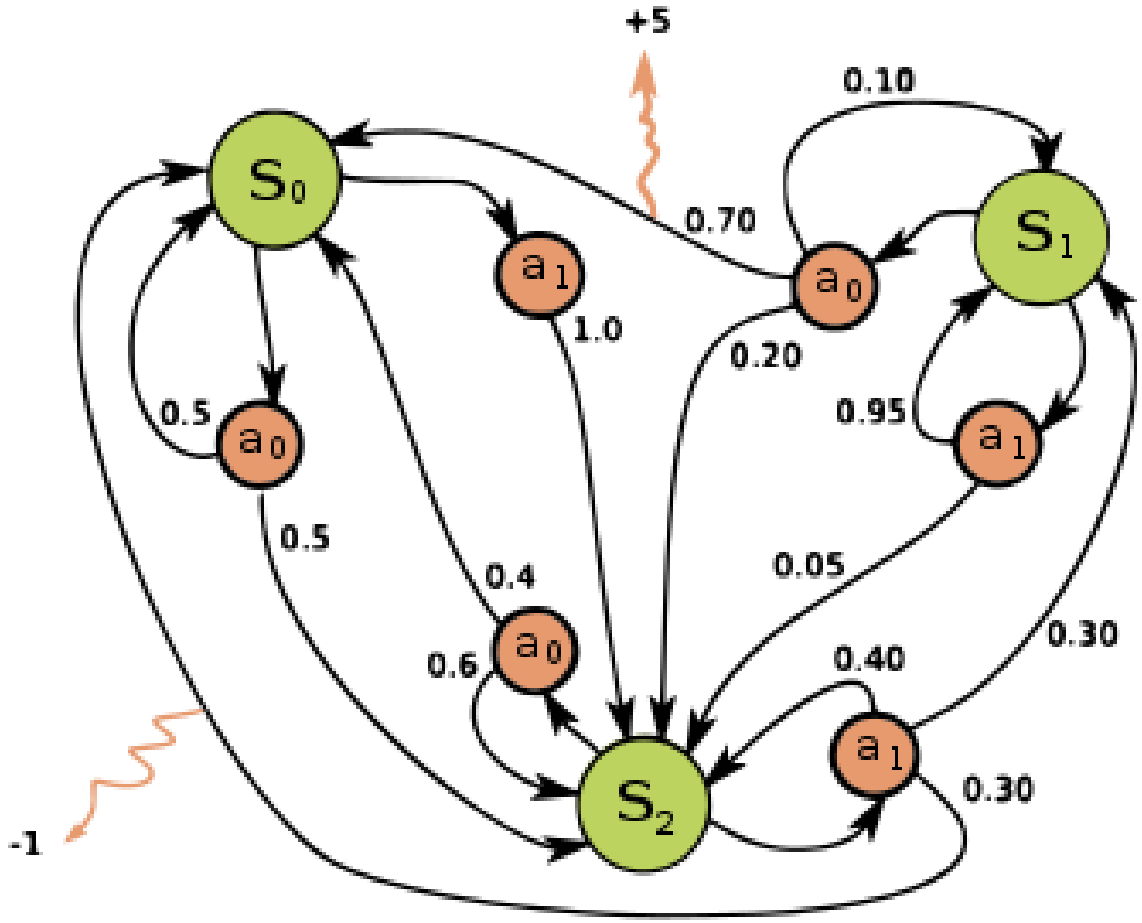
- The environment used in RL is often defined as finite MDP.
  - MDP is a mathematical framework used to model decision making problems.
  - In MDP, the entity that learns and makes decisions is called the agent, while everything outside of the agent that interacts with it is called the environment.

- A MDP decision process is a 4-tuple (S, A, P_a, R_a):
  - S = (s_0, s_1, ..., s_n) is a set of states called the state space,
  - A = (a_0, a_1, ..., a_n) is a set of actions called the action space,
  - $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$.
  - $R_a(s, s')$ is the immediate reward (or expected immediate reward) received after transitioning from state s to state s', due to action a.

- MDP goal:
  - A policy function π is a (potentially probabilistic) mapping from state space (S) to action space (A).
  - The goal in MDP is the find a good policy for decision maker.

Finite Markov Decision Process

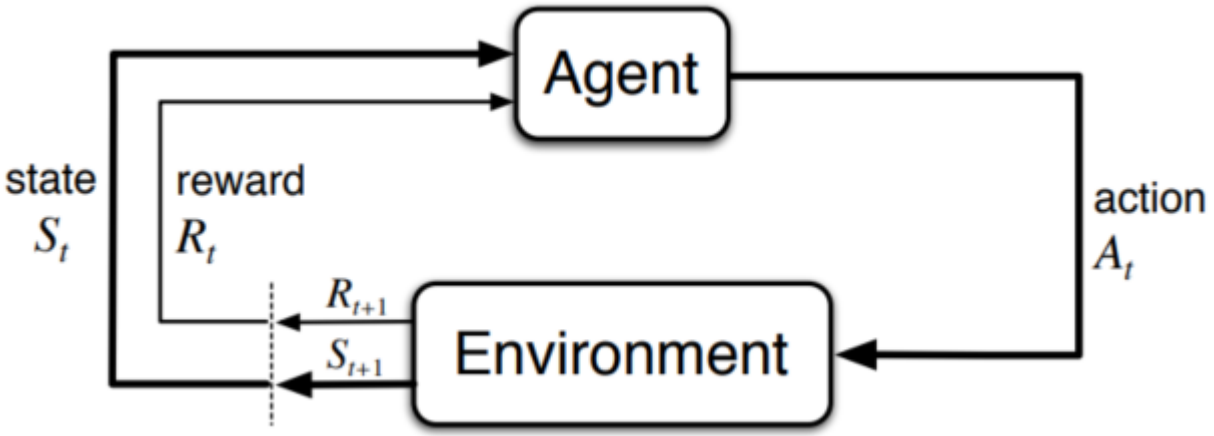
EXAMPLE[mdpw]



Finite Markov Decision Process

The Agent – Environment Interface[sutton]

- In MDP, agent-environment interactions are repeated until the goal is reached.
  - The agent selects actions
    → The environment responds to these actions, presents new situations to the agent, gives rise to rewards.
  - More specifically, at each time step t ($t \in 0, 1, 2, 3, \ldots$), the agent receives some representation of the environment's state, $S_t \in S$.
    → Based on the S_t, agent selects an action, $A_t \in A(s)$.
    → One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in R \subset \mathbb{R}$ and find itself in a new state, S_{t+1}.
  - This gives rise to a sequence or trajectory that begins like: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$



Finite Markov Decision Process

The Agent – Environment Interface[sutton]

- In finite MDP, the sets of states, actions, and rewards all have a finite number of elements.
  - Then, the random variables R_t and S_t have well defined discreate probability distributions dependent only on the preceding state and action:

$$p(s', r \mid s, a) \doteq \Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$$

[for all s', $s \in S$, $r \in R$, and $a \in A(s)$]

[note. $\doteq$ is used for "is defined as"]

  - This is a dynamic function p: $S \times R \times S \times A \rightarrow [0,1]$. In other words, p specifies a probability distribution for each choice of s and a, that is:

$$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1$$

- Markov property
  - In MDP, the probabilities given by p completely characterize the environment's dynamics.
  - That is, the probability of each possible value for S_t and R_t depends only on the S_{t-1} and R_{t-1}, and not at all on earlier states and actions.
  - This means that the state must include information about all aspects of the past agent-environment interaction that make a difference for the future.
  - Then, the state is said to have the Markov property.
  - $p(s_{t+1}, r_{t+1} \mid s_0, a_0, r_1, s_1, a_1, \ldots, r_{t-1}, s_{t-1}, a_{t-1}, r_t, s_t, a_t) = P(s_{t+1}, r_{t+1} \mid s_t, a_t)$

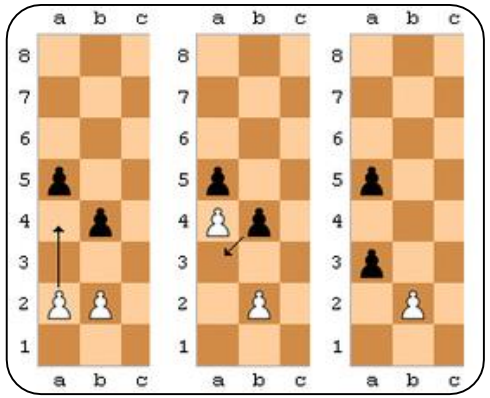
Finite Markov Decision Process

Why do we use Markov Property?

- It enables theoretical proofs of RL.
  - The reason for assuming this property is for efficiency.
  - Considering all past states and considering only the current state have the same probability = memoryless property
  - If you need to know the information of all past states to decide actions from the current state, the size of the state space would grow without limit.

Example

- Chess
  - In chess, you can make the same decisions as if you know the past, just by knowing the current situation, without needing to know the previous situations.
- Inventory management
  - Consider a retailer that needs to manage the inventory of a single product. The future inventory level and costs depend only on the current inventory level and the order decision.



Finite Markov Decision Process

The Agent – Environment Interface[sutton]

- State transition and reward functions
  - From the four-argument dynamics function, $p(s', r \mid s, a)$, a three-argument state-transition probability function can be obtained, p: $S \times S \times A \rightarrow [0,1]$:

$$p(s' \mid s, a) \doteq \Pr(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) = \sum_{r \in R} p(s', r \mid s, a)$$

  - The expected rewards for state-action pairs as a two-argument function can be obtained: r: $S \times A \rightarrow \mathbb{R}$:

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r \mid s, a)$$

  - The expected rewards for state-action-next state triples as a three-argument function: r: $S \times A \times S \rightarrow \mathbb{R}$:

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in R} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

Finite Markov Decision Process

The Agent – Environment Interface Example – Recycling Robot[sutton]

A mobile robot has the job of collecting empty soda cans in an office environment. It has sensors for detecting cans, and an arm and gripper that can pick them up and place them in an onboard bin; it runs on a rechargeable battery. The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper. High-level decisions about how to search for cans are made by a reinforcement learning agent based on the current charge level of the battery.

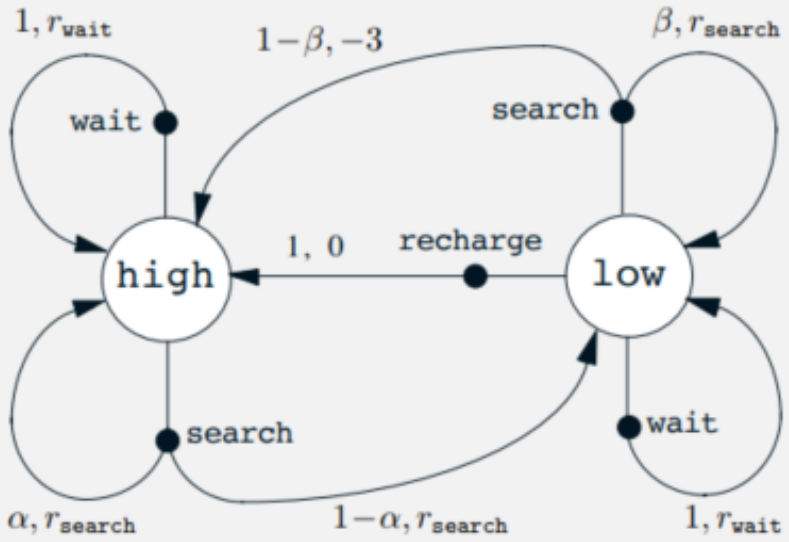
- Robot has two state defined by battery charge level: S = {**high**, **low**}
- The action sets are then A(**high**) = {**search**, **wait**} and A(**low**) = {**search**, **wait**, **recharge**}
- The rewards are zero most of the time, but become positive when the robot secures an empty can (r =1), or negative if the battery runs all the way down (r = -3).
- If the energy level is **high**, a searching leaves the energy level high with probability $\boldsymbol{\alpha}$ and reduces it to low with probability **1-**$\boldsymbol{\alpha}$. On the other hand, a period of searching undertaken when the energy level is **low** leaves it low with probability $\boldsymbol{\beta}$ and depletes the battery with probability **1-**$\boldsymbol{\beta}$.
- Let $r_{search} > r_{wait}$, where r denotes the expected number of cans the robot will collect (expected reward).

Finite Markov Decision Process

The Agent – Environment Interface Example – Recycling Robot[sutton]

- Robot has two state defined by battery charge level: S = {**high**, **low**}
- The action sets are then A(**high**) = {**search**, **wait**} and A(**low**) = {**search**, **wait**, **recharge**}
- The rewards are zero most of the time, but become positive when the robot secures an empty can (r >= 1), or negative if the battery runs all the way down (r = -3).
- If the energy level is **high**, a searching leaves the energy level high with probability α and reduces it to low with probability **1-α**. On the other hand, a period of searching undertaken when the energy level is **low** leaves it low with probability β and depletes the battery with probability **1-β** .
- Let $r_{search} > r_{wait}$, where r denotes the expected number of cans the robot will collect (expected reward).

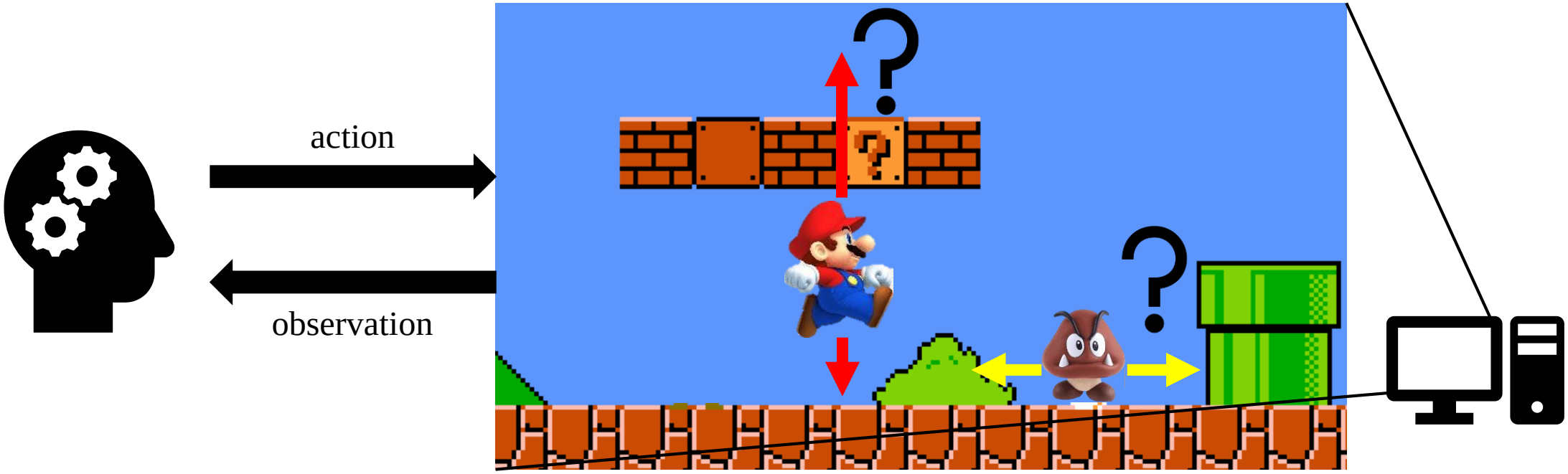
s	a	s'	$p(s'\|s,a)$	$r(s,a,s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-



Markov Property and State Design

State Design

- There is no general method to check whether the Markov property is satisfied.
  - Unfortunately, most real-world problems do not satisfy the Markov property.
  - For instance, we cannot know Mario's exact state just by looking at a single still image (observation).
  - You need more information to make the right decision, like whether the jump is going up or down, which direction Mario is moving, and which way Goomba is going.



Markov Property and State Design

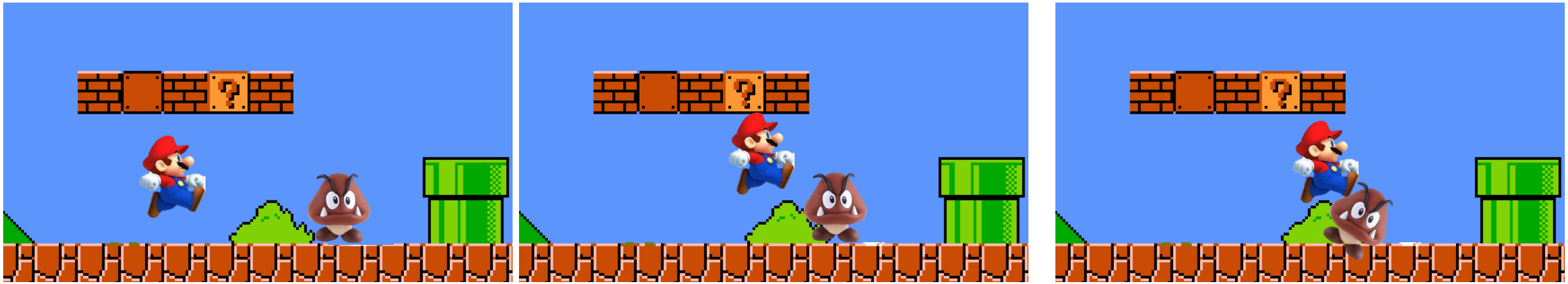
State Design

- Therefore, we need more information.
  - In practice, we try to reflect time-sensitive information to the state, making it satisfy the Markov property.
  - Now, a state is defined by [a snapshot of a particular point in time, a piece of information].
  - State doesn’t necessarily have to contain only the information about events that occurred at current time step *t*.
  - For instance, you can include the previous 10 frames of images from *t*-1, *t*-2, *t*-3, … *t*-9 in state *t*, or incorporate information that contains time attributes like velocity or acceleration.
  - You are free to compose state t to include information from previous time steps.

Markov Property and State Design

State Design Example – Super Mario

- Markov property design
  - You can compose state *t* with images from *t*-1, *t*-2, and *t*-3 frames.
  - Then, the movement information can be obtained in S_t without requiring previous state information.
    - ✓ For example, velocity and acceleration can be obtained in state *t*.
  - Consequently, you can make a decision only with the S_t,which means that the Markov property is satisfied.

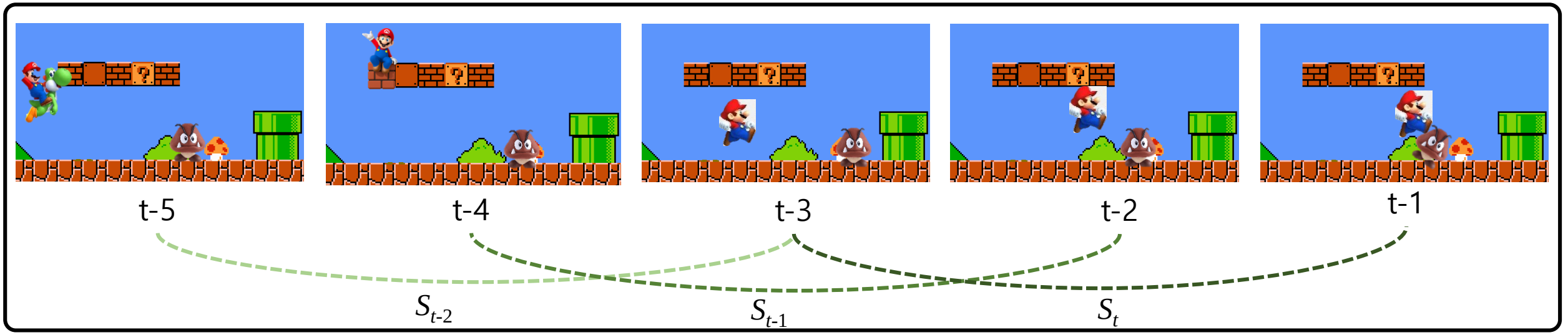


t-3 *t*-2 *t*-1

Markov Property and State Design

State Design Example – Super Mario

- Markov property design
  - Note that including the images from frames t-1, t-2, and t-3 in S_t does not mean using S_{t-1}, S_{t-2}, and S_{t-3}, directly.
  - The S_t represents what information the agent receives at time step t, not the information generated at t.
  - It is also possible to include information of S_{t-1} and S_{t-2} to S_t – this depends on the designer.
  - An important design goal is to make the future state of the environment depend only on the current state and action, not on the sequence of previous states and actions.



MDP - Goals and Rewards

Maximizing Rewards

- The reward is a formalized signal about the agent's goal.
  - A reward is a scalar value that the agent receives from the environment as feedback for taking an action in a particular state.
  - At each time step, the reward signal is passed from the environment to the agent.
  - Reward is a simple number, $R_t \in \mathbb{R}$.
  - In the recycling robot example, we gave the robot a reward of 0 most of the time, and then a reward of +1 for each can collected.
  - To make the robot move constantly, we can give a reward of -1 for every time step that the robot stops.

- The agent always learns to maximize its reward.
  - If we want the agent to do something for us, we must provide rewards to it in such a way that in maximizing them the agent will also achieve our goals.
  - Therefore, it is thus critical that the rewards we set up truly indicate what we want to be accomplished.
  - The reward signal is your way of communicating to the agent what you want it to achieve, not how you want it achieved.

MDP – Returns and Episodes

Maximizing Cumulative Reward over Episodes

- The return is the sum of the rewards.
  - We have said that the agent's goal is to maximize the cumulative reward the agent receives in the long run.
  - Formally, this is represented by seeking to maximize the expected return (G_t).
  - In the simplest case the return is the sum of the rewards:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \quad \text{[where, } T \text{ is a final time step]}$$

- The episode is a sequence of actions, states, and rewards.
  - An episode is a sequence of actions, states, and rewards, starting from an initial state and ending at a terminal state or a predefined time limit.
  - The agent's learning process consists of multiple episodes, and it aims to maximize the cumulative reward (return) it receives across these episodes.

- What if T is infinite? – To address this problem..

MDP – Return Discounting

Adjusting the importance of future rewards relative to immediate rewards.

- The discount factor (γ) is a value between 0 and 1 that determines the degree of discounting:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$G_t \doteq R_{t+1} + \gamma G_{t+1}$$

- A higher discount factor gives more importance to future rewards, whereas a lower discount factor emphasizes immediate rewards.

- Note that even if the G is a sum of an infinite number of terms, G can be determined in the context of finite if the reward is nonzero, constant, and $\gamma < 1$.
  - ✓ If the reward is constant +1, $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$

- Discounting if often used for a few reasons:
  - ✓ **Modeling uncertainty:** Discounting future rewards can help model the uncertainty of an agent's ability to estimate long-term consequences accurately. As we look further into the future, predictions become more uncertain.
  - ✓ **Fast convergence**: When learning an optimal policy, discounting can help the agent focus on more immediate rewards and converge to a solution faster.
  - ✓ **Finite tasks**: In tasks with infinite or very long time horizons, discounting can ensure that the cumulative rewards remain finite and prevent potential issues in the learning process.

MDP – Return Discounting

Adjusting the importance of future rewards relative to immediate rewards.

- If discounting is not used (i.e., $\gamma = 1$), the agent will treat all future rewards with equal importance to immediate rewards.

- This can lead to several issues:
  - ✓ **Slow convergence**: The agent may take longer to learn an optimal policy, as it considers long-term consequences equally important to immediate outcomes.
  - ✓ **Infinite returns**: In tasks with infinite or very long time horizons, the cumulative rewards may become infinite, making it challenging to compare different policies effectively.
  - ✓ **Overemphasis on long-term consequences**: The agent may prioritize actions with long-term benefits at the expense of immediate rewards, potentially leading to suboptimal behavior.

- If $\gamma = 0$, the agent is myopic in being concerned only with maximizing immediate rewards.

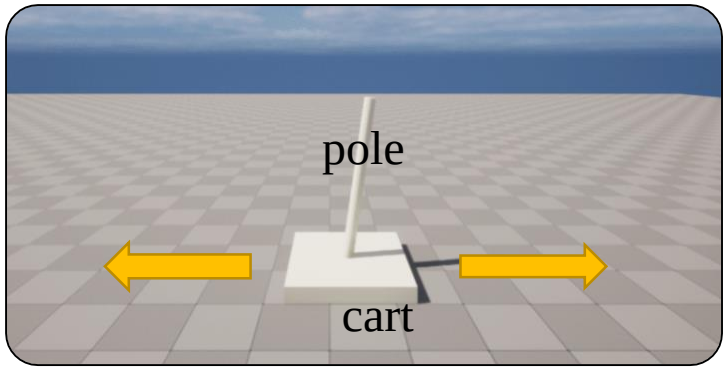
- Including the possibility that $T = \infty$ or $\gamma = 1$ (but not both), the general form of return can be written as follows:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

MDP – Return Discounting

Return Example with Pole-Balancing.

- Problem Definition
  - Objective: Apply forces to a cart moving along a track so as to keep a pole hinged to the cart from falling over.
  - Failure 1: if the pole falls past a given angle from vertical.
  - Failure 2: if the cart runs off the track.
  - After failure: The pole is reset to vertical after each failure.



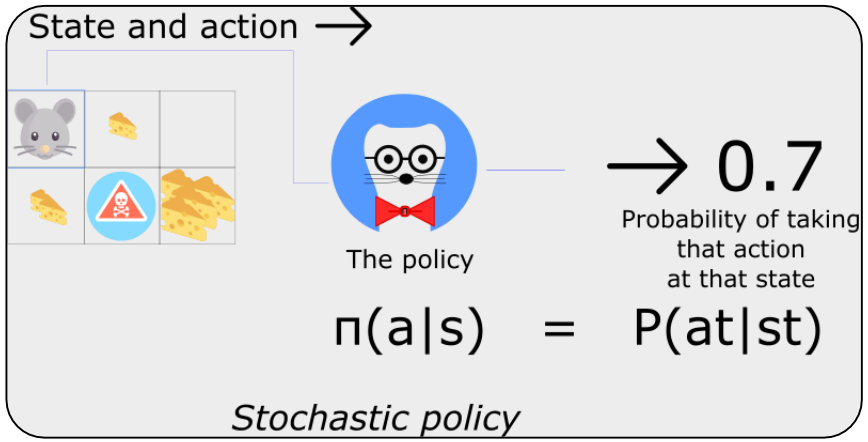
- Design as Episodic
  - The cart repeatedly attempts to balance the pole over sequence of episodes.
  - In this case, reward could be +1 for every time step on which failure did not occur.
  - The return at each time would be the number of steps until failure.
  - Successful balancing forever would mean a return of infinity.

- Design as Continuing Task
  - In this case, reward could be -1 on each failure and zero at all other times.
  - The return at each time would then be related to $-\gamma^k$, where k is the number of time steps before failure.

MDP – Policies and Value Functions

How good it is to perform a given action in a given state.

- Policy
  - A policy π is a mapping from states to probabilities of selecting each possible action.
  - In other action, it is a rule or strategy that defines the agent's behavior in an environment.
  - There are two primary types of policies:
    - **Deterministic policy**: The agent takes a specific action for each state, with no randomness involved: $a = \pi(s)$.
    - **Stochastic policy**: The agent selects actions probabilistically, based on a probability distribution over possible actions for each state: $P(a|s) = \pi(a|s)$.



- Value functions
  - Value functions are used to estimate the *expected cumulative rewards* or *expected return* an agent can receive from a given state or a state-action pair.
  - This help the agent evaluate the desirability of state or actions, guiding the agent's decision-making process to maximize its cumulative rewards over time.

MDP –Value Functions

There are two types of value functions.

- State-value function (V-function)
  - The state-value function, $v_\pi(s)$, represents the expected return an agent can achieve starting from state s and following a specific policy π.
  - This is formally defined as:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s], \text{ for all } s \in S,$$

    where $\mathbb{E}_\pi[\ ]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step.

- Action-value function (Q-function)
  - The action-value function, Q(s, a), estimates the expected return an agent can achieve starting from state s, taking action a, and then following a specific policy π for the subsequent states.
  - This represents the long-term value of taking a particular action in a specific state while adhering to the given policy.
  - This is formally defined as:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a].$$

MDP –Value Functions

Value functions can be estimated from experience.

- Experience?
  - The agent's experience comes from the sequence of states, actions, and rewards it encounters as it interacts with the environment over time.
  - By observing the rewards and transitions between states as a result of its actions, the agent can incrementally update its estimates of the value functions, getting closer to their true values.
  - There are some popular methods for estimating value functions: Monte Carlo, Temporal Difference, etc.

- Monte Carlo (MC) methods:
  - MC methods use the average of the actual returns observed from multiple episodes to estimate the value functions.
  - MC methods work with complete episodes and require the task to be episodic.
  - To estimate the state-value function $v_\pi(s)$, the agent can average the returns from all the episodes where the s was visited.
  - To estimate the action-value function $q_\pi(s, a)$, the agent can average the returns from all the episodes where the state-action pair (s, a) was visited.
  - If the number of state or actions is very large, MC may not be practical. To overcome this, parameterized function approximator can be used (but this will not be addressed in our lecture).

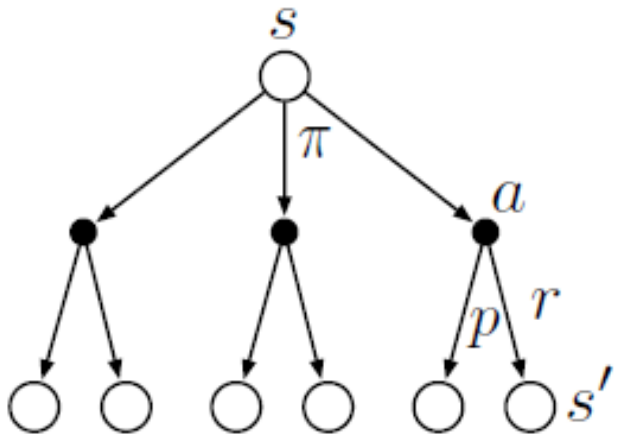
MDP –Value Functions

Bellman Equation

- Recursive relationships
  - For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)\,[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma v_\pi(s')] \end{aligned}$$

  - We call the above *Bellman equation* for $v_\pi(s)$.
- Backup diagram
  - Open circle: state & solid circle: state-action pair.
  - Starting from state s, the root node at the top, the agent could take any of some set of actions (three are shown in the diagram) based on its policy π.
  - From each of these, the environment could respond with one of several next states, s′ (two are shown in the figure), along with a reward, r, depending on its dynamics given by the function p.
  - The Bellman equation averages over all the possibilities, weighting each by its probability of occurring.
  - It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.



MDP – Optimal Policies and Value Functions

Finding a policy that achieves a lot of reward over the long run

- Optimal policy?
  - A policy π is defined to be better than π' if its *expected return* is greater than to that of π' for all states.
  - In other words, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$, for all $s \in S$.
  - There is always at least one policy that is better than or equal to all other policies: optimal policy.
  - We denote all the optimal policies by π_*.
  - They share the same state-value function, called the optimal state value function, denoted v_*:

$$v_*(s) \doteq \max_\pi v_\pi(s), \text{ for all } s \in S.$$

  - Optimal polices also share the same optimal action-value function, denoted q_*:

$$q_*(s) \doteq \max_\pi q_\pi(s, a), \text{ for all } s \in S \text{ and } a \in A(s)$$

  - For the state-action pair (s, a), this function gives the expected return for taking action a in state s and thereafter following an optimal policy:

$$q_*(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

MDP – Optimal Policies and Value Functions

Finding Optimal *policies* and *value functions* is Indeed a challenging.

- High computational complexity
  - Even if we have a complete and accurate model of the environment's dynamics, it is usually not possible to simply compute an optimal policy by solving the Bellman optimality equation.
  - The amount of computation is always insufficient.

- Memory shortage
  - A large amount of memory is often required to build up approximations of value functions, policies, and models.
  - In small tasks, it is possible to form these approximations using arrays or tables with one entry for each state (or state-action pair) - We call this *tabular case*, and the corresponding methods we call *tabular methods*.
  - However, many real world problems include far more states than could possibly be entries in a table.
  - To overcome this, more compact parametrized function representation is required.

- Partial observability
  - In some environments, the agent may not have complete information about the state of the environment, leading to partial observability.
  - This makes estimating value functions more difficult, as the agent must infer the hidden aspects of the environment from the observed information.
  - Methods such as partially observable Markov decision processes (POMDPs) and recurrent neural networks (RNNs) can be used to handle partial observability.

Reference

[mdpw] https://en.wikipedia.org/wiki/Markov_decision_process
[sutton] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Contributors

List of Contributions
[initial flow, some images]

DongHeun Han

Research Fields
- Character Animation
- Physical Animation
- Reinforcement Learning

Email: hand32@khu.ac.kr

List of Contributions
[flow reorganization, RL project images]

JuYeong Hwang

Research Fields
- Character Animation

dudyyyy4@khu.ac.kr