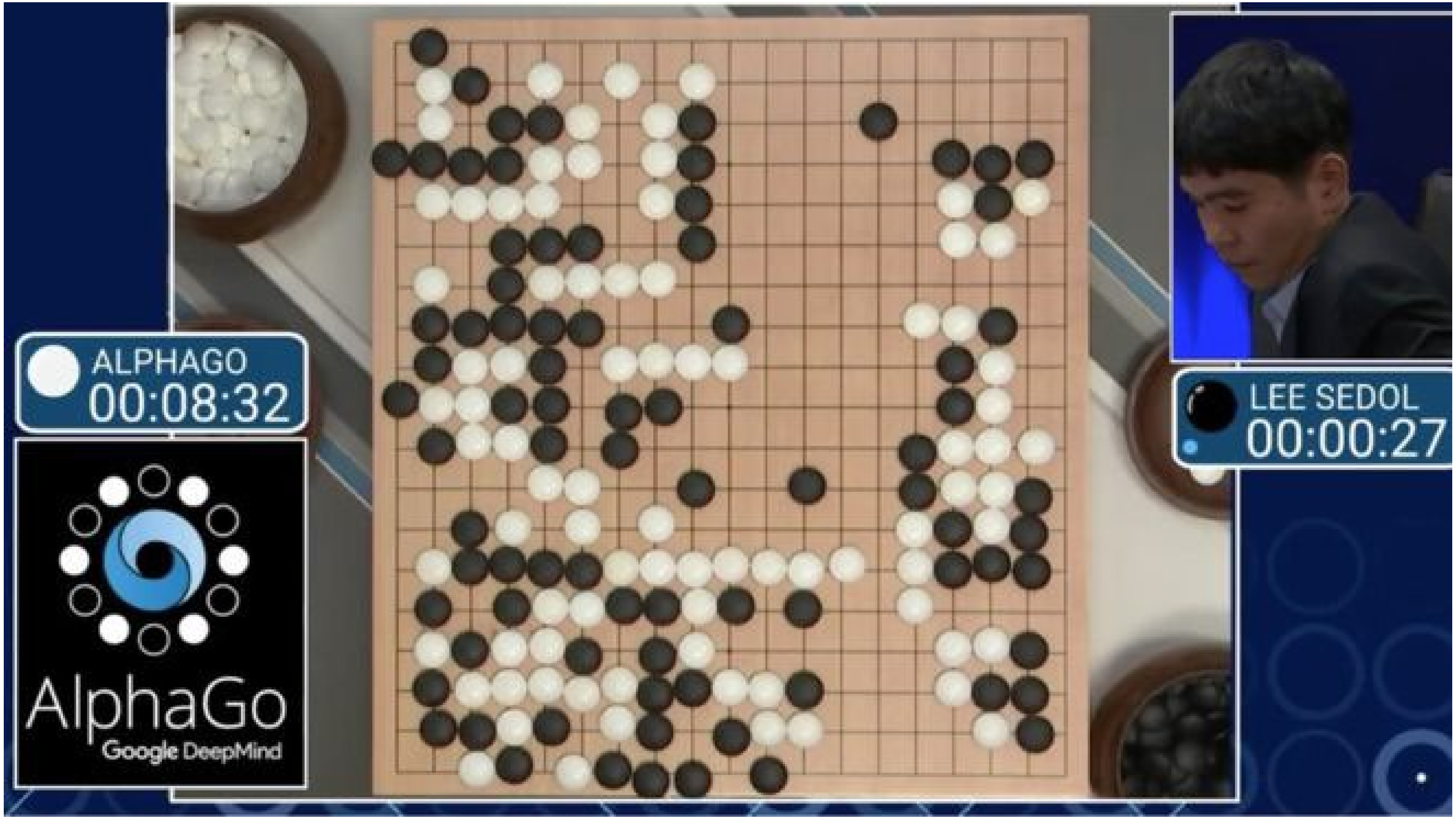
# Reinforcement Learning

Adapted from slides by Shusen Wang at Stevens Institute of Technology

http://wangshusen.github.io/

# AlphaGo



# A little bit probability theory…

# Random Variable

- Random variable: unknown; its values depends on outcomes of a random event.
- Uppercase letter $X$ for random variable.

| Random Variable | Possible Values | Random Events | Probabilities |
| --- | --- | --- | --- |
| $X =$ | 0 | | $\mathbb{P}(X = 0) = 0.5$ |
| | 1 | | $\mathbb{P}(X = 1) = 0.5$ |

# Random Variable

- Random variable: unknown; its values depends on outcomes of a random event.
- Uppercase letter $X$ for random variable.
- Lowercase letter $x$ for an observed value.
- For example, I flipped a coin 4 times and observed:
  - $x_1 = 1$,
  - $x_2 = 1$,
  - $x_3 = 0$,
  - $x_4 = 1$.

# Probability Density Function (PDF)

- PDF provides a relative likelihood that the value of the random variable would equal that sample.

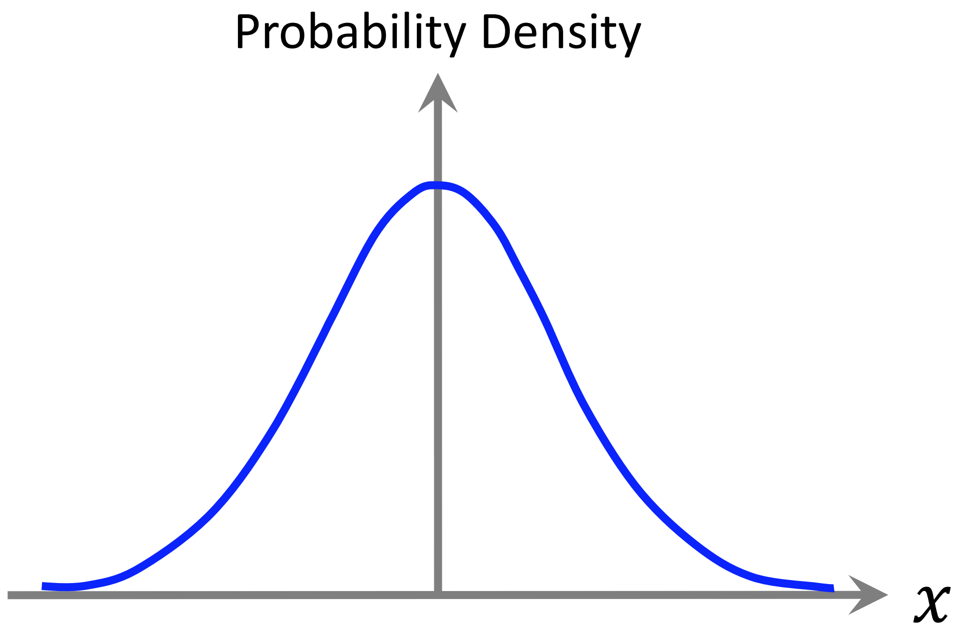
# Probability Density Function (PDF)

- PDF provides a relative likelihood that the value of the random variable would equal that sample.

**Example:** Gaussian distribution

- It is a continuous distribution.
- PDF:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\,\sigma^2}\right).$$



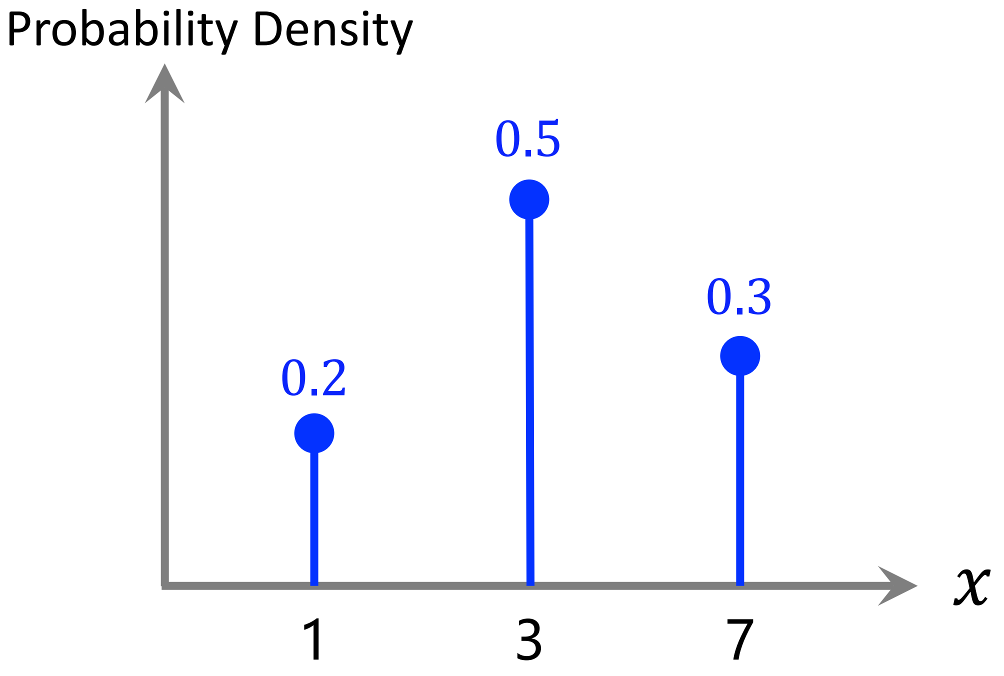
# Probability Mass Function (PMF)

- PMF is a function that gives the probability that a discrete random variable is exactly equal to some value

**Example**

- Discrete random variable: $X \in \{1, 3, 7\}$.
- PDF:

$$p(1) = 0.2,$$
$$p(3) = 0.5,$$
$$p(7) = 0.3.$$



# Properties of PDF/PMF

- Random variable $X$ is in the domain $\mathcal{X}$.
- For continuous distribution,

$$\int_{\mathcal{X}} p(x)\, dx = 1.$$

- For discrete distribution,

$$\sum_{x \in \mathcal{X}} p(x) = 1.$$

# Expectation

- Random variable $X$ is in the domain $\mathcal{X}$.
- For continuous distribution, the expectation of $f(X)$ is:

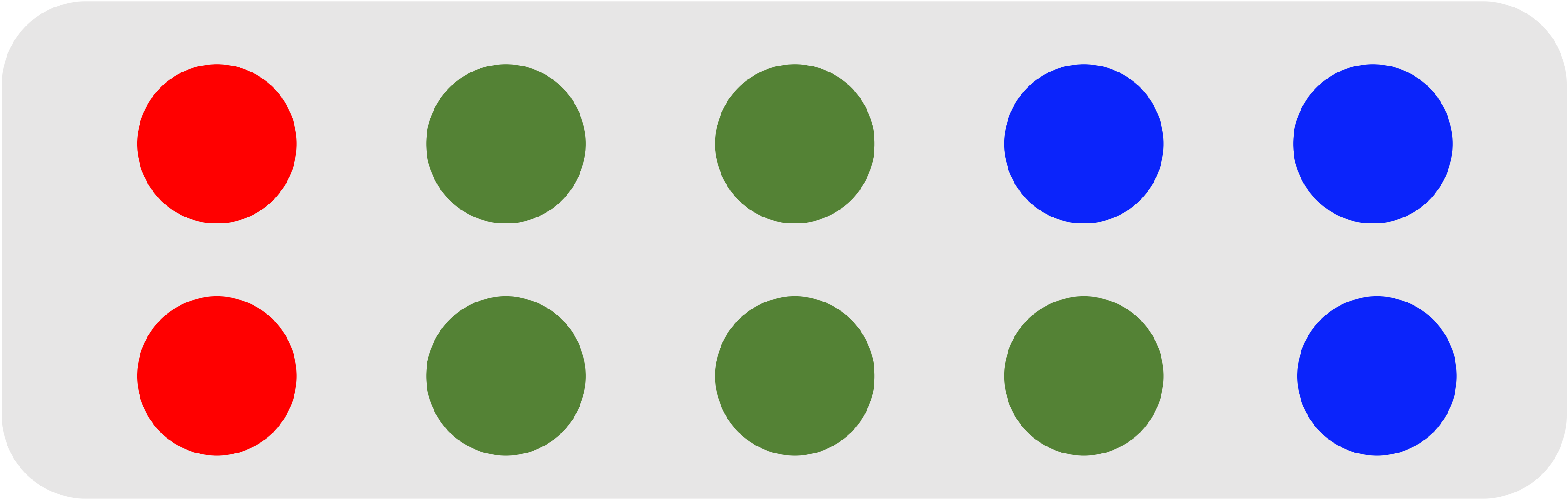
$$\mathbb{E}\left[f(X)\right] = \int_{\mathcal{X}} p(x) \cdot f(x)\, dx.$$

- For discrete distribution, the expectation of $f(X)$ is:

$$\mathbb{E}\left[f(X)\right] = \sum_{x \in \mathcal{X}} p(x) \cdot f(x)\,.$$

# Random Sampling

- There are 10 balls in the bin: 2 are red, 5 are green, and 3 are blue.
- Randomly sample a ball.
- What will be the color?



# Random Sampling

- Sample red ball w.p. 0.2, green ball w.p. 0.5, and blue ball w.p. 0.3.
- Randomly sample a ball.
- What will be the color?

# Random Sampling

- Sample red ball w.p. 0.2, green ball w.p. 0.5, and blue ball w.p. 0.3.
- Randomly sample a ball.
- What will be the color?

```
from numpy.random import choice

samples = choice(['R', 'G', 'B'], size=100, p=[0.2, 0.5, 0.3])
print(samples)
```

```
['R' 'G' 'R' 'R' 'R' 'R' 'B' 'B' 'B' 'G' 'G' 'B' 'G' 'B' 'B' 'G' 'B' 'G'
 'B' 'B' 'G' 'B' 'G' 'B' 'B' 'G' 'B' 'B' 'G' 'B' 'G' 'G' 'G' 'G' 'G' 'B'
 'B' 'B' 'B' 'B' 'B' 'G' 'G' 'B' 'R' 'R' 'B' 'R' 'B' 'G' 'R' 'G' 'R' 'G'
 'R' 'R' 'B' 'G' 'G' 'G' 'B' 'R' 'G' 'B' 'G' 'R' 'G' 'G' 'G' 'B' 'B' 'R'
 'G' 'G' 'B' 'B' 'R' 'B' 'B' 'B' 'R' 'B' 'G' 'B' 'R' 'B' 'R' 'G' 'B' 'R'
 'B' 'B' 'G' 'G' 'G' 'R' 'R' 'B' 'R' 'G']
```

# Terminologies

# Terminology: state and action

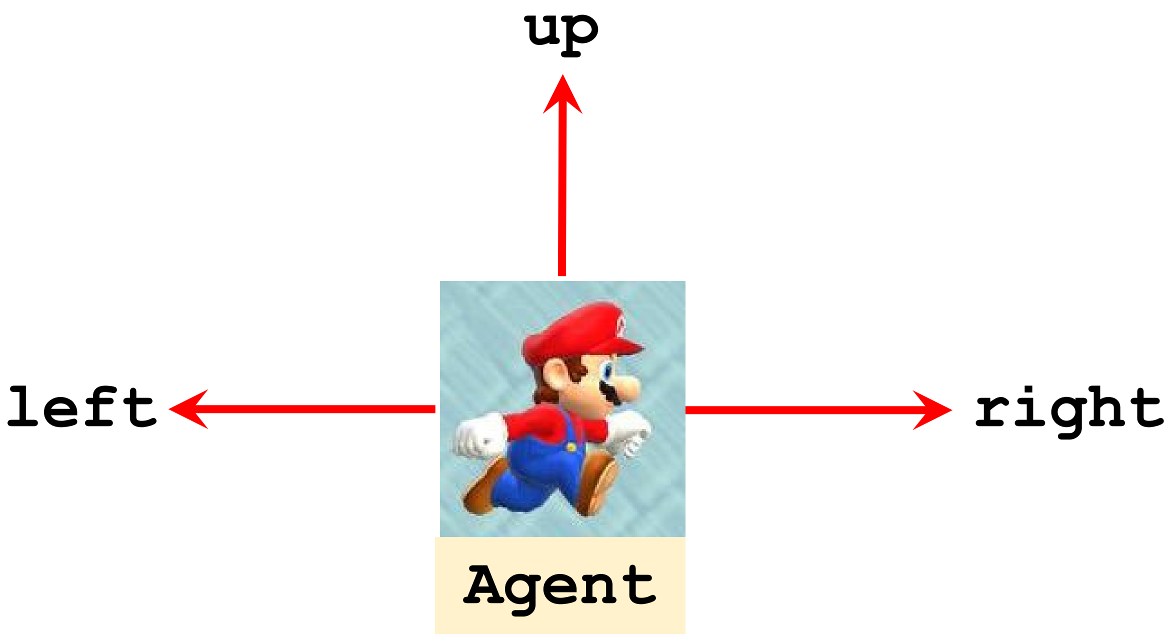
state $s$ (this frame)

Action $a \in$ {left, right, up}

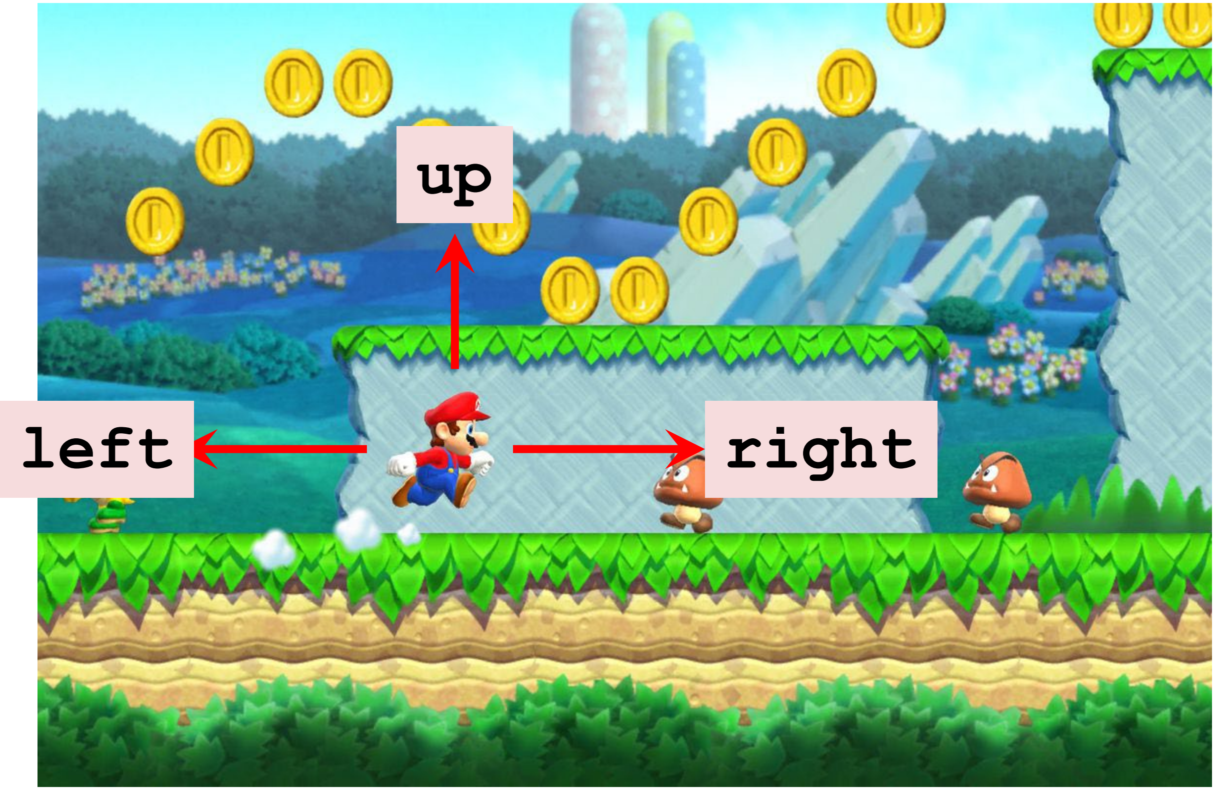
# Terminology: state and action

**state $s$ (this frame)**

Action $a \in$ {left, right, up}



# Terminology: policy



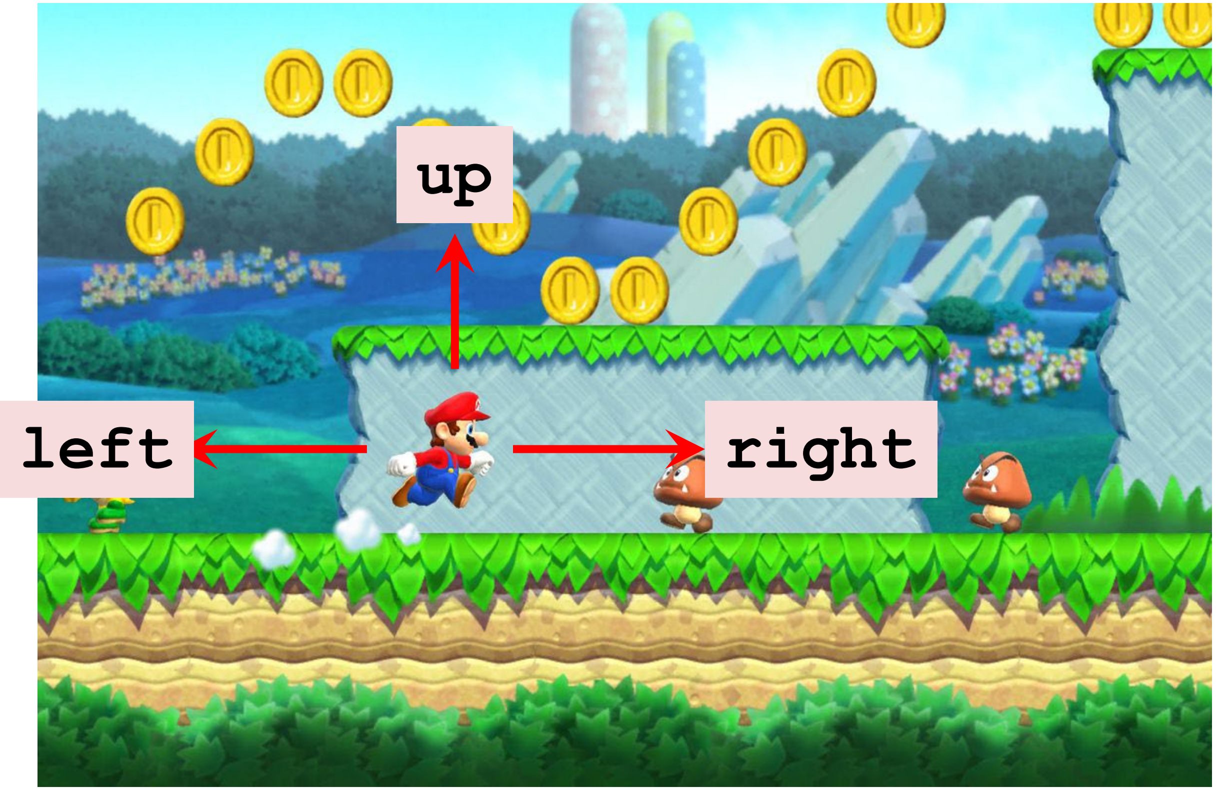

**policy $\pi$**

- Policy function $\pi: (s, a) \mapsto [0,1]$:

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

- It is the probability of taking action $A = a$ given state $s$, e.g.,
  - $\pi(\text{left} \mid s) = 0.2$,
  - $\pi(\text{right} \mid s) = 0.1$,
  - $\pi(\text{up} \mid s) = 0.7$.
- Upon observing state $S = s$, the agent's action $A$ can be random.

# Terminology: policy



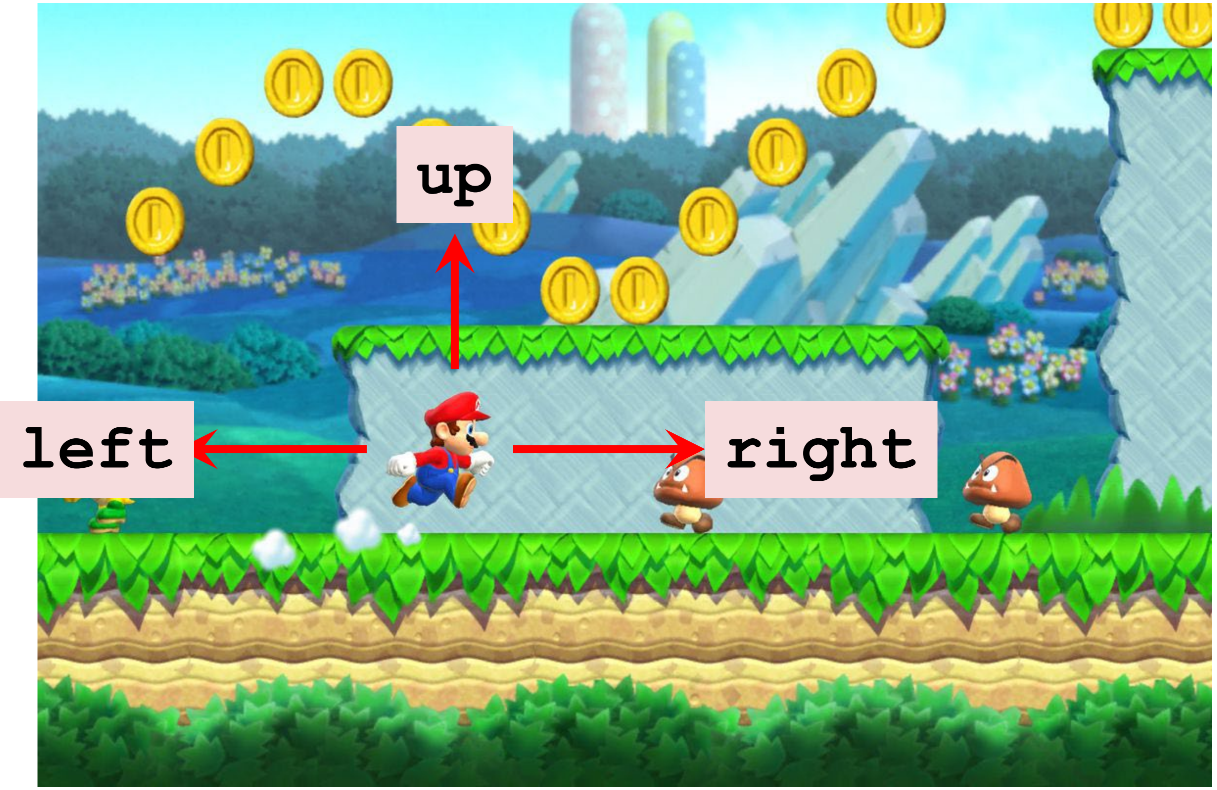

**policy $\pi$**

- Policy function $\pi: (s, a) \mapsto [0,1]$:

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

- It is the probability of taking action $A = a$ given state $s$, e.g.,
  - $\pi(\text{left} \mid s) = 0.2$,
  - $\pi(\text{right} \mid s) = 0.1$,
  - $\pi(\text{up} \mid s) = 0.7$.
- Upon observing state $S = s$, the agent's action $A$ can be random.

# Terminology: policy



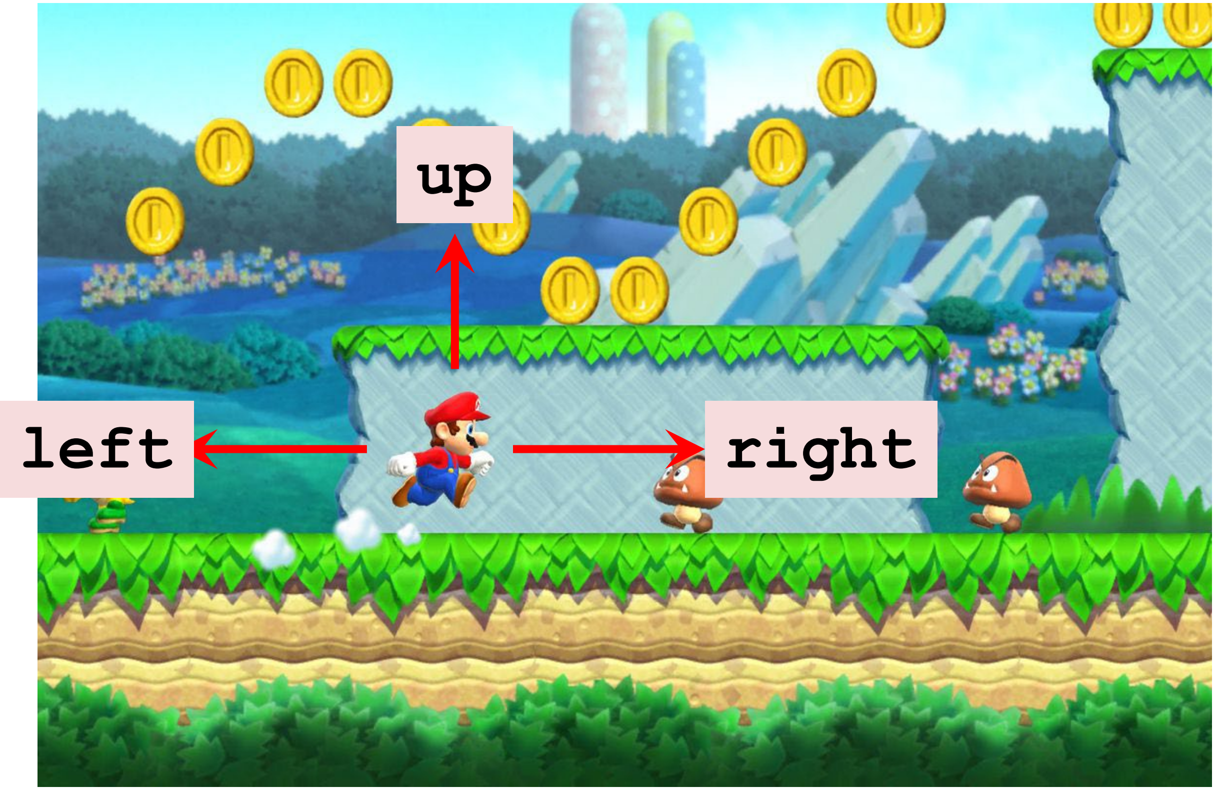

**policy $\pi$**

- Policy function $\pi: (s, a) \mapsto [0,1]$:

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

- It is the probability of taking action $A = a$ given state $s$, e.g.,
  - $\pi(\text{left} \mid s) = 0.2$,
  - $\pi(\text{right} \mid s) = 0.1$,
  - $\pi(\text{up} \mid s) = 0.7$.
- Upon observing state $S = s$, the agent's action $A$ can be random.

# Terminology: policy




## policy $\pi$

- Policy function $\pi: (s, a) \mapsto [0,1]$:

$$\pi(a \mid s) = \mathbb{P}(A = a \mid S = s).$$

- It is the probability of taking action $A = a$ given state $s$, e.g.,
  - $\pi(\text{left} \mid s) = 0.2$,
  - $\pi(\text{right} \mid s) = 0.1$,
  - $\pi(\text{up} \mid s) = 0.7$.
- Upon observing state $S = s$, the agent's action $A$ can be random.

# Terminology: reward

**reward $R$**

- Collect a coin: $R = +1$

# Terminology: reward

**reward $R$**

- Collect a coin: $R = +1$
- Win the game: $R = +10000$

# Terminology: reward

## reward $R$

- Collect a coin: $R = +1$
- Win the game: $R = +10000$
- Touch a Goomba: $R = -10000$ (game over).

# Terminology: reward

**reward** $R$

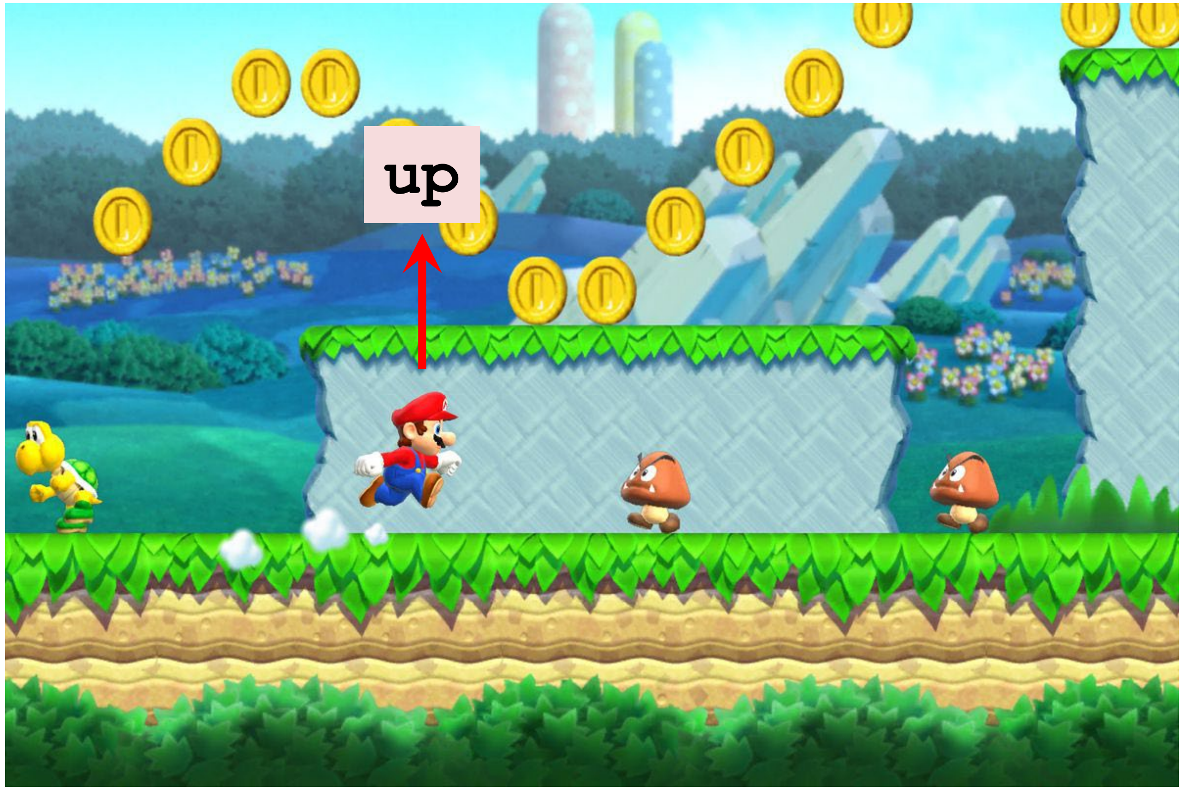
- Collect a coin: $R = +1$
- Win the game: $R = +10000$
- Touch a Goomba: $R = -10000$ (game over).
- Nothing happens: $R = 0$

# Terminology: state transition

**state transition**

old state —action→ new state

# Terminology: state transition

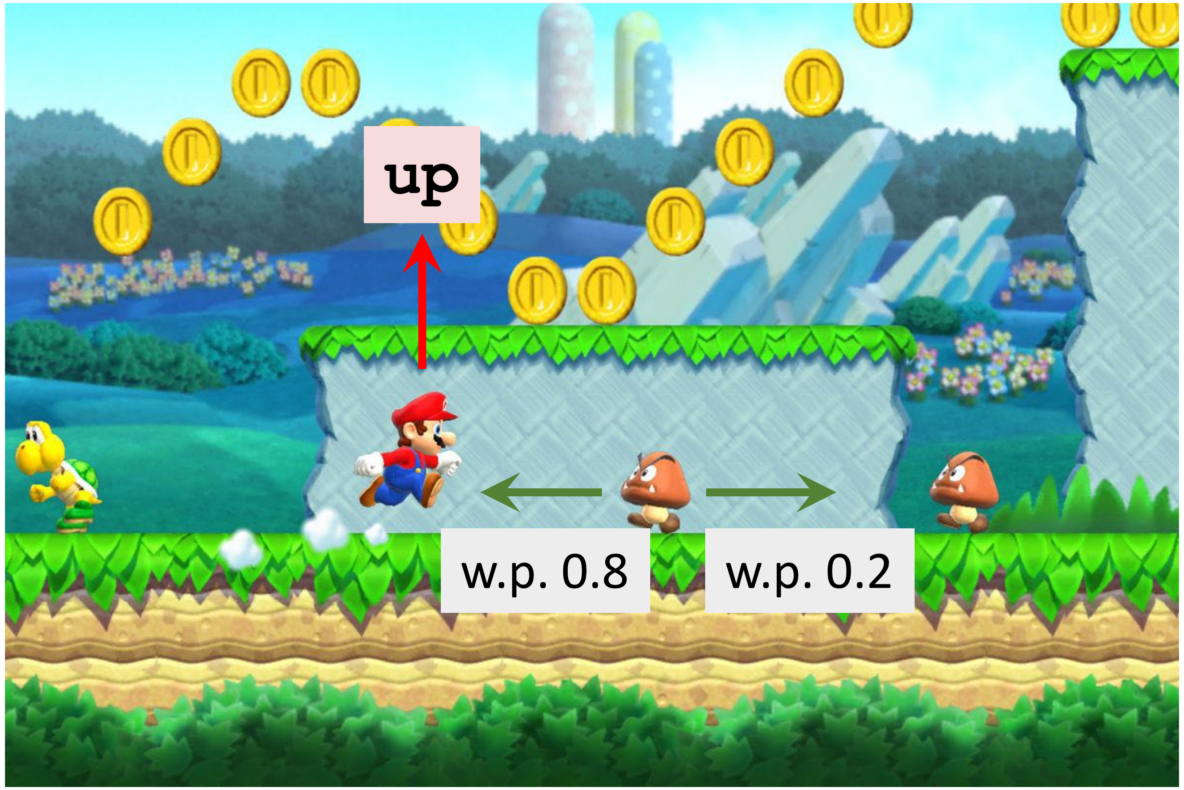



**state transition**

old state —action→ new state

- E.g., “up” action leads to a new state.

# Terminology: state transition

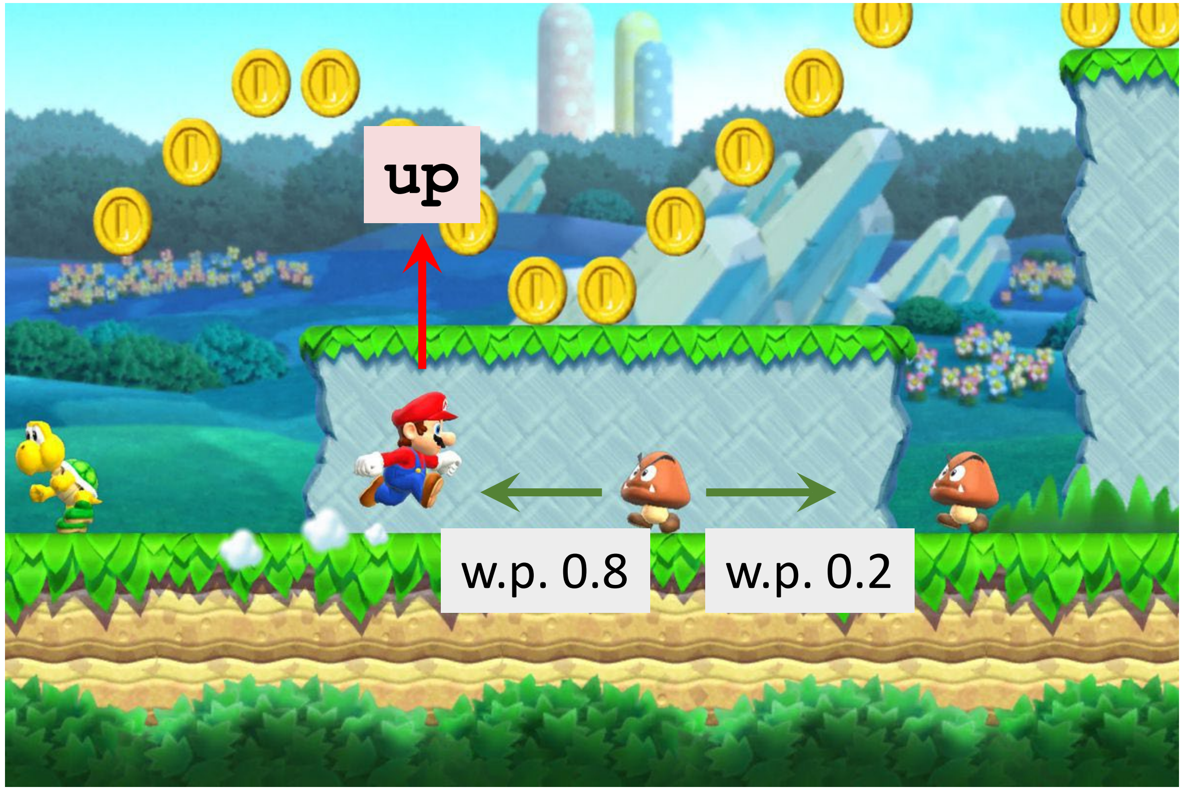



**state transition**


- E.g., "up" action leads to a new state.

- State transition can be random.
- Randomness is from the environment.

# Terminology: state transition

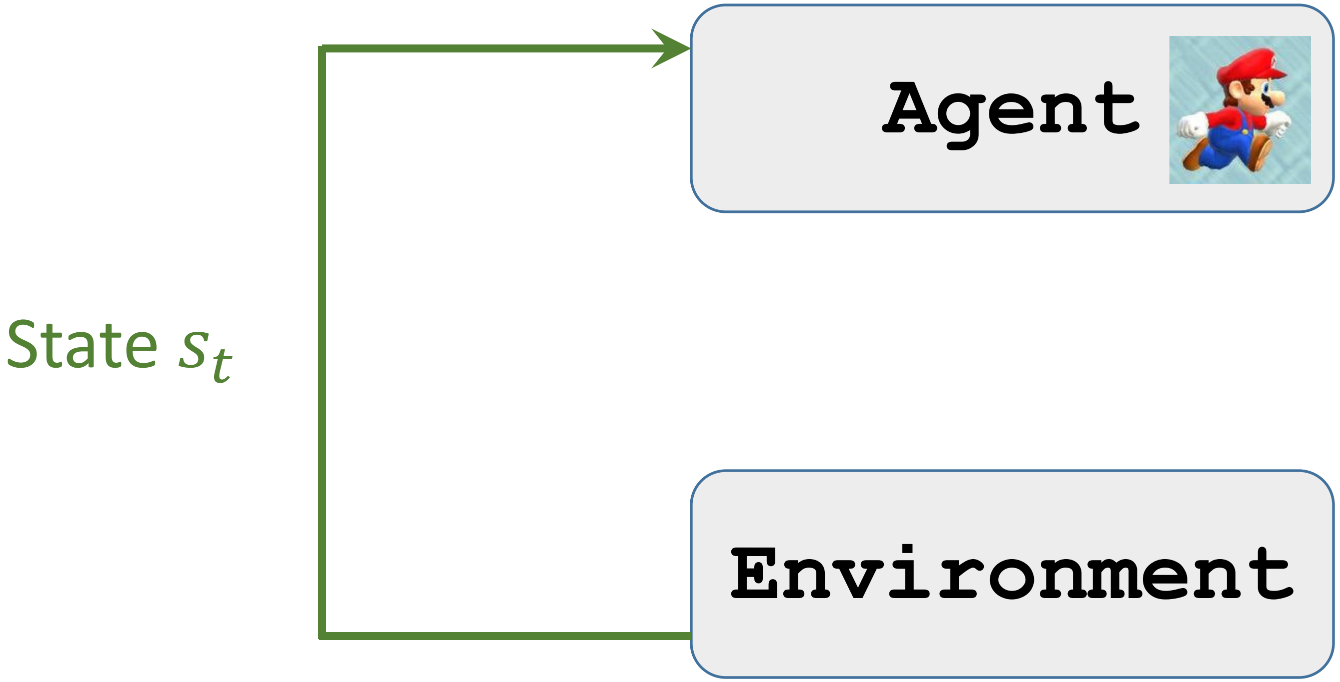



## state transition

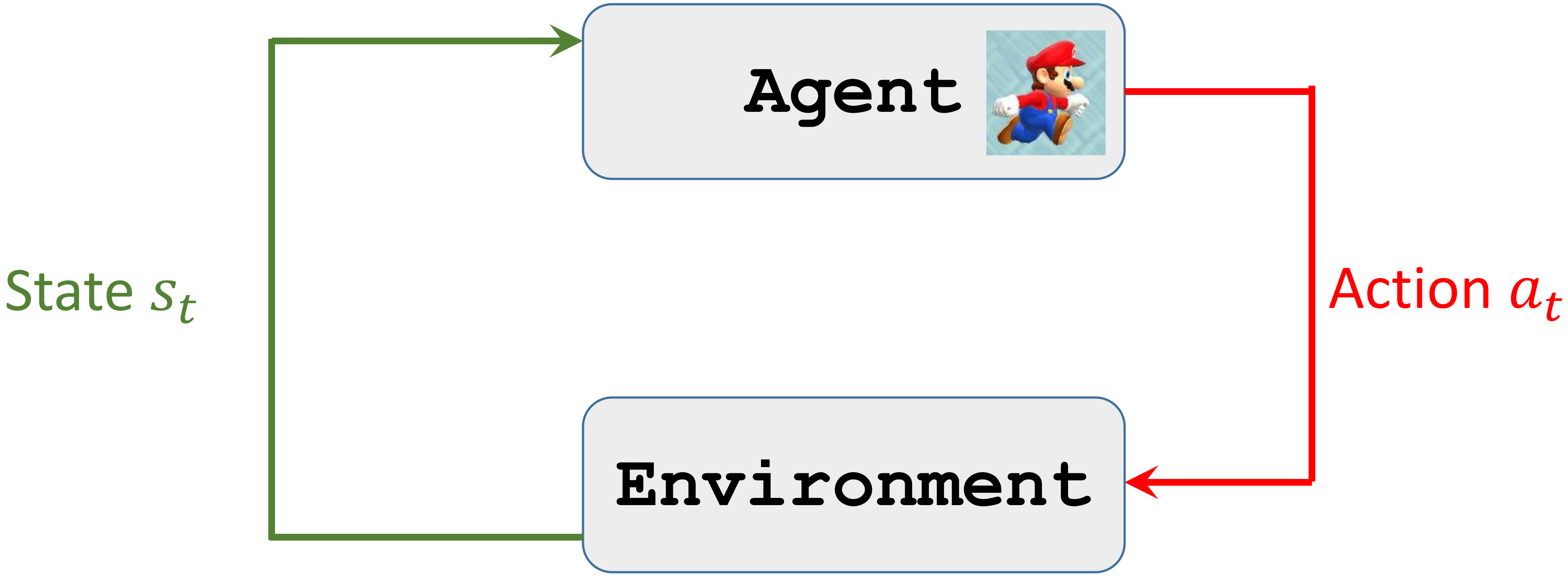
old state —action→ new state

- E.g., "up" action leads to a new state.
- State transition can be random.
- Randomness is from the environment.
- $p(s'|s, a) = \mathbb{P}(S' = s'|S = s, A = a)$.

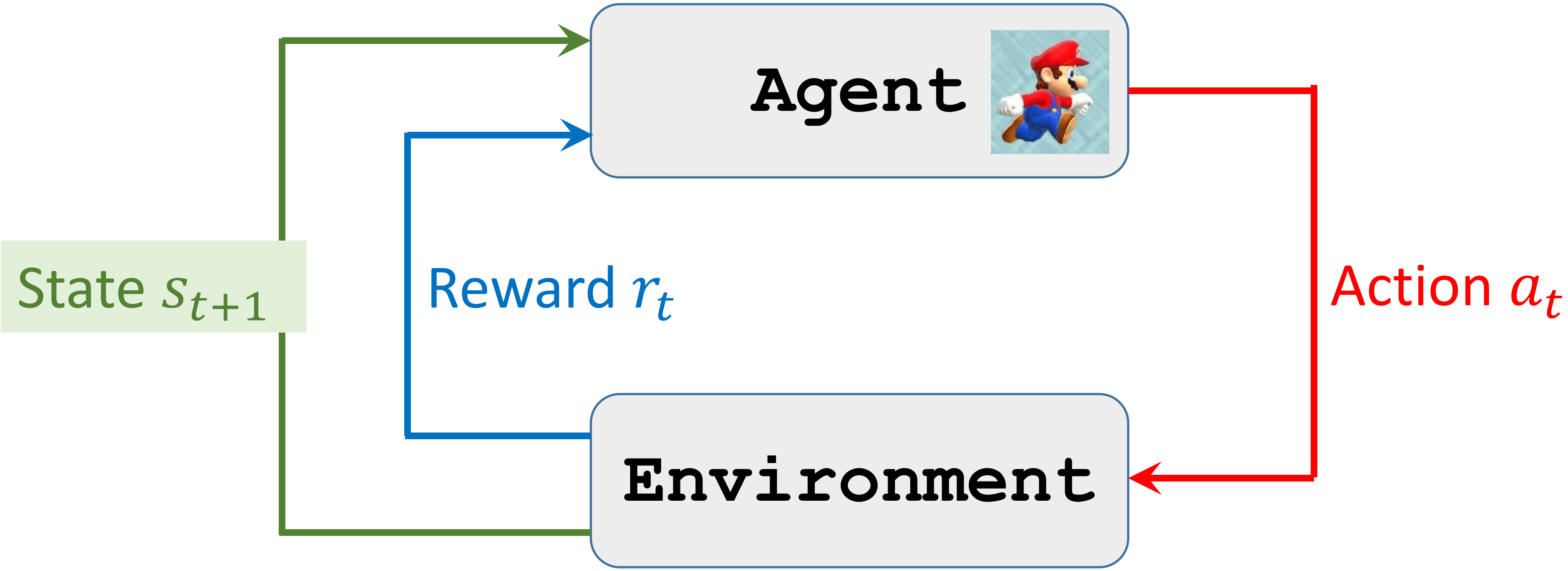
# Terminology: agent environment interaction



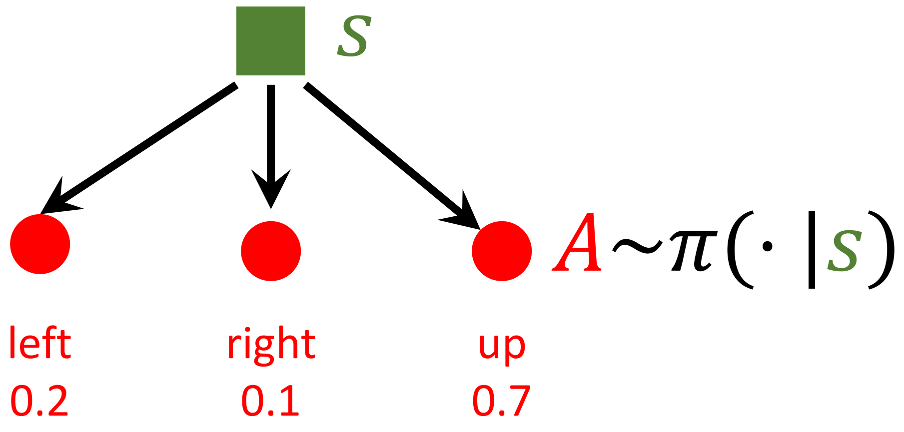
# Terminology: agent environment interaction



# Terminology: agent environment interaction



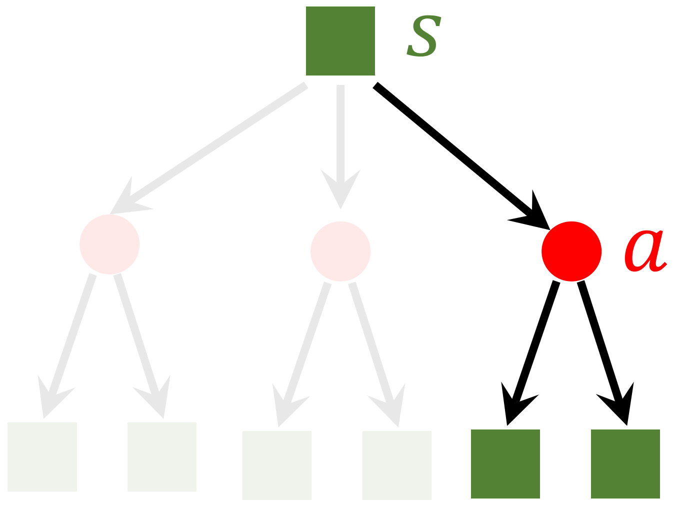
# Randomness in Reinforcement Learning




Actions have randomness.

- Given state $s$, the action can be random, e.g., .
  - $\pi(\text{“left”}|s) = 0.2$,
  - $\pi(\text{“right”}|s) = 0.1$,
  - $\pi(\text{“up”}|s) = 0.7$.

# Randomness in Reinforcement Learning




$S' \sim p(\cdot \mid s, a)$

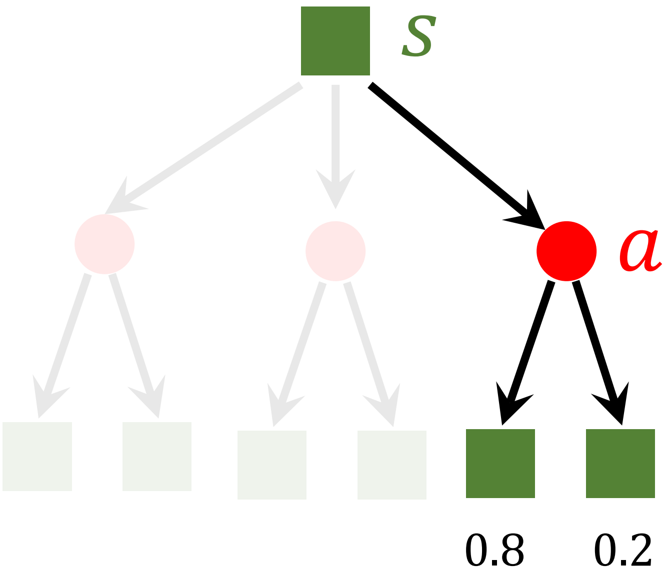
Actions have randomness.

- Given state $s$, the action can be random, e.g., .
  - $\pi(\text{“left”} \mid s) = 0.2$,
  - $\pi(\text{“right”} \mid s) = 0.1$,
  - $\pi(\text{“up”} \mid s) = 0.7$.

State transitions have randomness.

- Given state $S = s$ and action $A = a$, the environment randomly generates a new state $S'$.

# Randomness in Reinforcement Learning




Actions have randomness.

- Given state $s$, the action can be random, e.g., .
  - $\pi(\text{“left”}|s) = 0.2$,
  - $\pi(\text{“right”}|s) = 0.1$,
  - $\pi(\text{“up”}|s) = 0.7$.

State transitions have randomness.

- Given state $S = s$ and action $A = a$, the environment randomly generates a new state $S'$.

# Play the game using AI

- Observe a frame (state $s_1$)
- ➔ Make action $a_1$ (left, right, or up)
- ➔ Observe a new frame (state $s_2$) and reward $r_1$
- ➔ Make action $a_2$
- ➔ …

# Play the game using AI

- Observe a frame (state $s_1$)
- ➔ Make action $a_1$ (left, right, or up)
- ➔ Observe a new frame (state $s_2$) and reward $r_1$
- ➔ Make action $a_2$
- ➔ …

- (state, action, reward) trajectory:

$$s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T.$$

# Rewards and Returns

# Return

**Definition:** Return (aka cumulative future reward).

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

# Return

**Definition:** Return (aka cumulative future reward).

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

**Question:** Are $R_t$ and $R_{t+1}$ equally important?

- Which of the followings do you prefer?
  - I give you \$100 right now.
  - I will give you \$100 one year later.

# Return

**Definition:** Return (aka cumulative future reward).

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

**Question:** Are $R_t$ and $R_{t+1}$ equally important?

- Which of the followings do you prefer?
  - I give you \$100 right now.
  - I will give you \$100 one year later.

- Future reward is less valuable than present reward.
- $R_{t+1}$ should be given less weight than $R_t$.

# Return

**Definition:** Return (aka cumulative future reward).

- $U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $\gamma$: discount rate (tuning hyper-parameter).
- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

# Randomness in Returns

**Definition:** Discounted return (at time step $t$).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

At time step $t$, the return $U_t$ is random.

- Two sources of randomness:
  1. Action can be random: $\mathbb{P}[A = a \mid S = s] = \pi(a|s)$.
  2. New state can be random: $\mathbb{P}[S' = s' | S = s, A = a] = p(s'|s, a)$.

# Randomness in Returns

**Definition:** Discounted return (at time step $t$).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

At time step $t$, the return $U_t$ is random.

- Two sources of randomness:
  1. Action can be random: $\mathbb{P}[A = a \mid S = s] = \pi(a|s)$.
  2. New state can be random: $\mathbb{P}[S' = s' | S = s, A = a] = p(s'|s, a)$.
- For any $i \geq t$, the reward $R_i$ depends on $S_i$ and $A_i$.
- Thus, given $s_t$, the return $U_t$ depends on the random variables:
  - $A_t, A_{t+1}, A_{t+2}, \cdots$ and $S_{t+1}, S_{t+2}, \cdots$.

# Value Functions

# Action-Value Function $Q(s, a)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma \, R_{t+1} + \gamma^2 \, R_{t+2} + \gamma^3 \, R_{t+3} + \cdots$

# Action-Value Function $Q(s, a)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

**Definition:** Action-value function for policy $\pi$.

- $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$.

- Return $U_t$ depends on states $S_t, S_{t+1}, S_{t+2}, \cdots$ and actions $A_t, A_{t+1}, A_{t+2}, \cdots$.

# Action-Value Function $Q(s, a)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

**Definition:** Action-value function for policy $\pi$.

- $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$.

- Return $U_t$ depends on states $S_t, S_{t+1}, S_{t+2}, \cdots$ and actions $A_t, A_{t+1}, A_{t+2}, \cdots$.
- Actions are random: $\mathbb{P}[A = a \mid S = s] = \pi(a|s)$. (Policy function.)
- States are random: $\mathbb{P}[S' = s' | S = s, A = a] = p(s'|s, a)$. (State transition.)

# Action-Value Function $Q(s, a)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

**Definition:** Action-value function for policy $\pi$.

- $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$.

**Definition:** Optimal action-value function.

- $Q^\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$.

# State-Value Function $V(s)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

**Definition:** Action-value function for policy $\pi$.

- $Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right].$

**Definition:** State-value function.

- $V_\pi(s_t) = \mathbb{E}_A\left[Q_\pi(s_t, A)\right]$

# State-Value Function $V(s)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

**Definition:** Action-value function for policy $\pi$.

- $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$.

**Definition:** State-value function.

- $V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$. (Actions are discrete.)

Taken w.r.t. the action $A \sim \pi(\cdot|s_t)$.

# State-Value Function $V(s)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

**Definition:** Action-value function for policy $\pi$.

- $Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right]$.

**Definition:** State-value function.

- $V_\pi(s_t) = \mathbb{E}_A\left[Q_\pi(s_t, A)\right] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$. (Actions are discrete.)
- $V_\pi(s_t) = \mathbb{E}_A\left[Q_\pi(s_t, A)\right] = \int \pi(a|s_t) \cdot Q_\pi(s_t, a)\, da$ . (Actions are continuous.)

# Understanding the Value Functions

- Action-value function: $Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right]$.
- Given policy $\pi$, $Q_\pi(s, a)$ evaluates how good it is for an agent to pick action $a$ while being in state $s$.
- $Q^\star(s_t, a_t)$ evaluates how good it is for an agent to pick action $a$ while being in state $s$ no matter what the policy is.

# Understanding the Value Functions

- Action-value function: $Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | S_t = s_t, A_t = a_t]$.
- Given policy $\pi$, $Q_\pi(s, a)$ evaluates how good it is for an agent to pick action $a$ while being in state $s$.
- $Q^\star(s_t, a_t)$ evaluates how good it is for an agent to pick action $a$ while being in state $s$ no matter what the policy is.

- State-value function: $V_\pi(s) = \mathbb{E}_A\,[Q_\pi(s, A)]$
- For fixed policy $\pi$, $V_\pi(s)$ evaluates how good the situation is in state $s$.
- $\mathbb{E}_S[V_\pi(S)]$ evaluates how good the policy $\pi$ is.

# Play games using reinforcement learning

# How does AI control the agent?

Suppose we have a good policy $\pi(a|s)$.

- Upon observing the state $s_t$,
- random sampling: $a_t \sim \pi(\cdot\,|s_t)$.

# How does AI control the agent?

Suppose we have a good policy $\pi(a|s)$.

- Upon observing the state $s_t$,
- random sampling: $a_t \sim \pi(\cdot|s_t)$.

Suppose we know the optimal action-value function $Q^{\star}(s, a)$.

- Upon observe the state $s_t$,
- choose the action that maximizes the value: $a_t = \text{argmax}_a Q^{\star}(s_t, a)$.

# OpenAI Gym

- Gym is a toolkit for developing and comparing reinforcement learning algorithms.
- https://gym.openai.com/

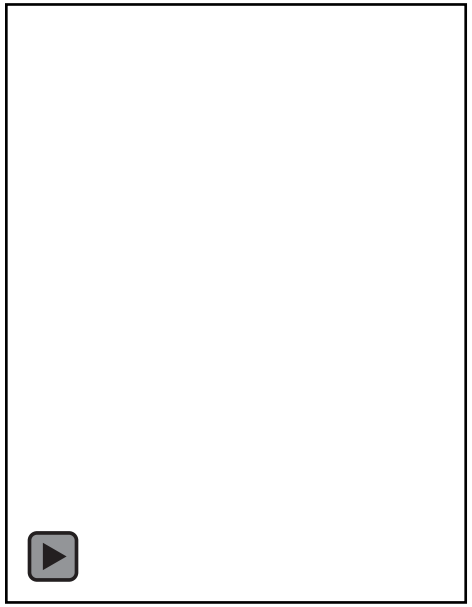
**ontrol pr**

Cart Pole

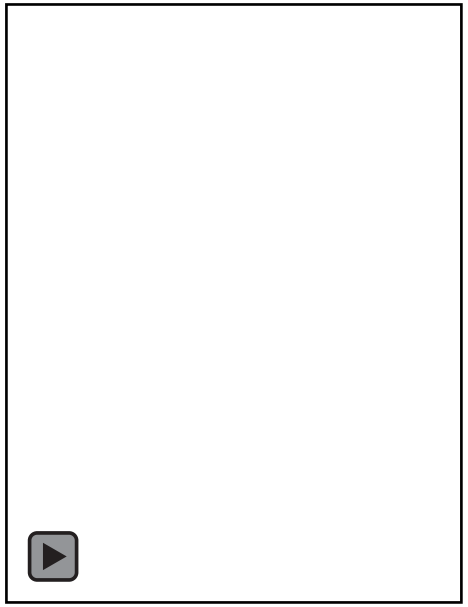
# OpenAI Gym

- Gym is a toolkit for developing and comparing reinforcement learning algorithms.
- https://gym.openai.com/

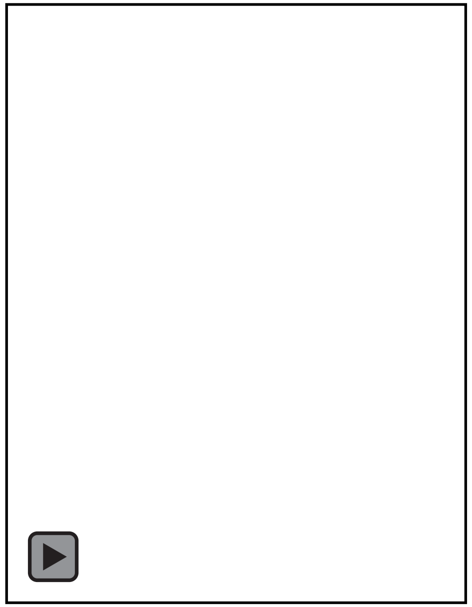
**Atari Games**



Pong



Space Invader

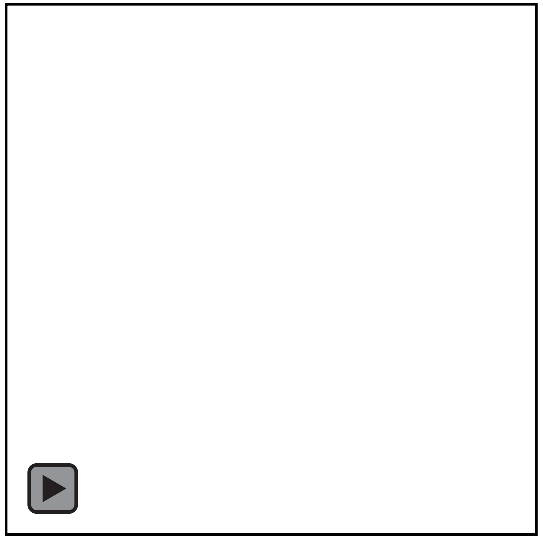


Breakout

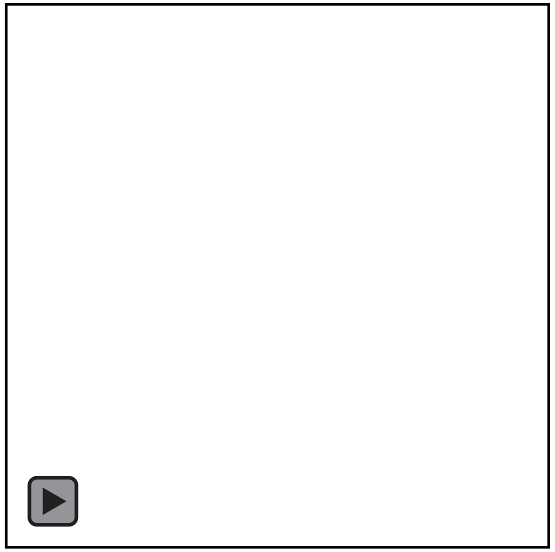
# OpenAI Gym

- Gym is a toolkit for developing and comparing reinforcement learning algorithms.
- https://gym.openai.com/

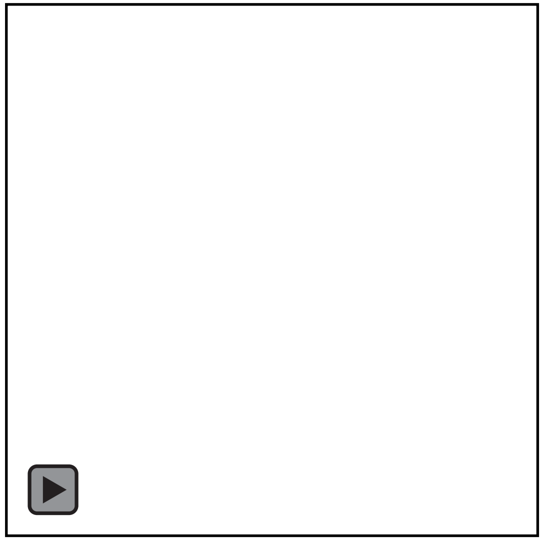
**MuJoCo (Continuous control tasks.)**



Ant

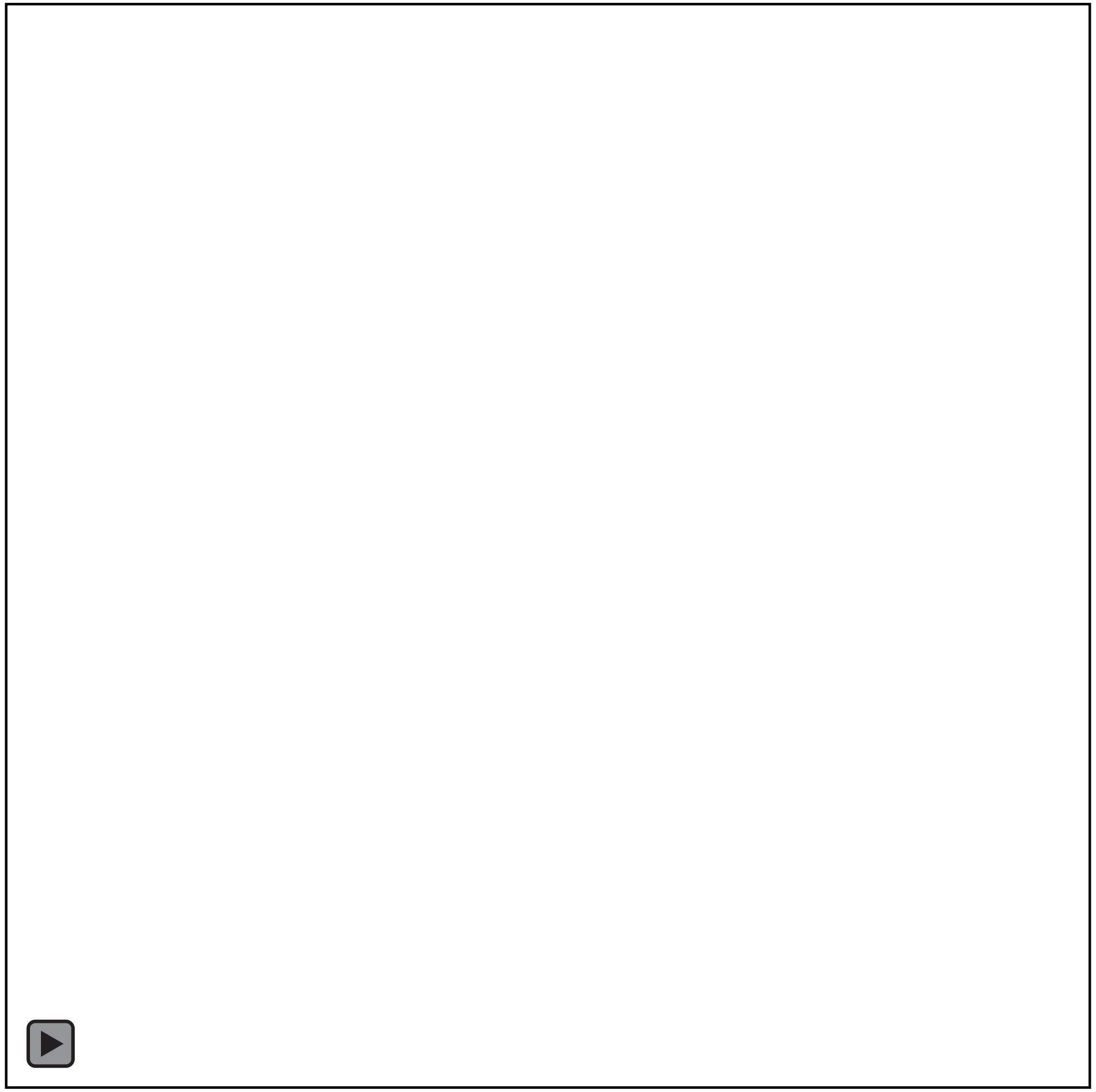


Humanoid



Half Cheetah

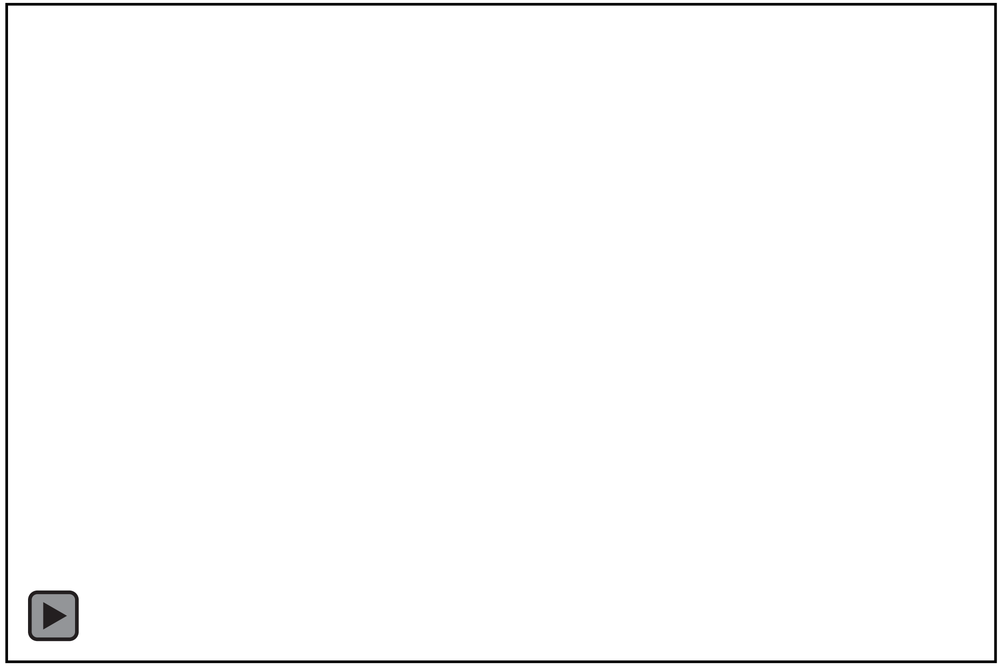
# OpenAI Gym



# Play CartPole Game

```python
import gym
env = gym.make('CartPole-v0')
```

- Get the environment of CartPole from Gym.
- “env” provides states and reward.



# Play CartPole Game

```
state = env.reset()

for t in range(100):
    env.render()
    print(state)

    action = env.action_space.sample()
    state, reward, done, info = env.step(action)

    if done:
        print('Finished')
        break

env.close()
```

A window pops up rendering CartPole.

A random action.

"done=1" means finished (win or lose the game)

# Summary

# Summary

**Terminologies**

- Agent 
- Environment
- State $s$.
- Action $a$.
- Reward $r$.
- Policy $\pi(a|s)$
- State transition $p(s'|s, a)$.

# Summary

## Terminologies

- Agent 
- Environment
- State $s$.
- Action $a$.
- Reward $r$.
- Policy $\pi(a|s)$
- State transition $p(s'|s,a)$.

## Return and Value

- Return:

$$U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \cdots$$

- Action-value function:

$$Q_\pi(s_t, a_t) = \mathbb{E}\,[U_t | s_t, a_t].$$

- Optimal action-value function:

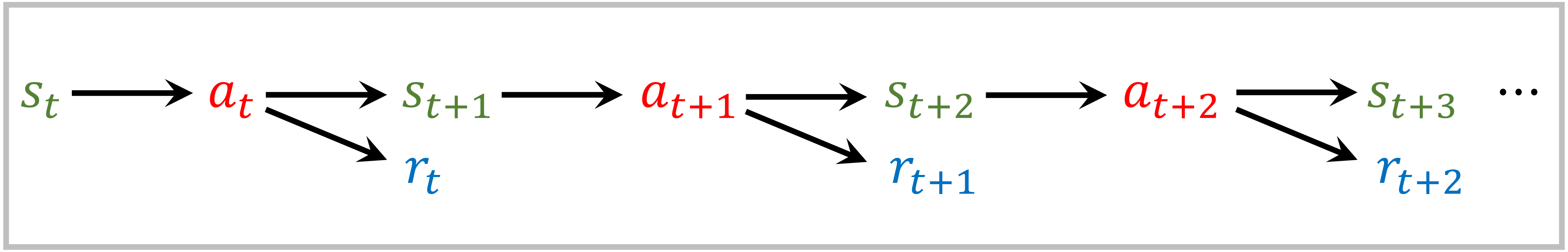
$$Q^\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t).$$

- State-value function:

$$V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)].$$

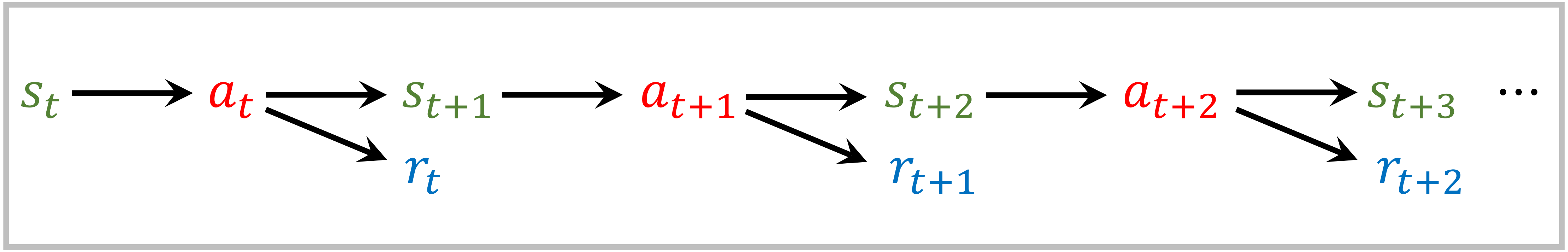
# Play game using reinforcement learning

- Observe state $s_t$, make action $a_t$, environment gives $s_{t+1}$ and reward $r_t$.



# Play game using reinforcement learning

- Observe state $s_t$, make action $a_t$, environment gives $s_{t+1}$ and reward $r_t$.




- The agent can be controlled by either $\pi(a|s)$ or $Q^{\star}(s, a)$.

# We are going to study…

2. Value-based learning.
   - Deep Q network (DQN) for approximating $Q^{\star}(s, a)$.
   - Learn the network parameters using temporal different (TD).

3. Policy-based learning.
   - Policy network for approximating $\pi(a|s)$.
   - Learn the network parameters using policy gradient.

4. Actor-critic method. (Policy network + value network.)

5. Example: AlphaGo

# Value-Based Reinforcement Learning

**Shusen Wang**

# Action-Value Functions

# Discounted Return

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma\, R_{t+1} + \gamma^2\, R_{t+2} + \gamma^3\, R_{t+3} + \cdots$

- The return depends on actions $A_t, A_{t+1}, A_{t+2}, \cdots$ and states $S_t, S_{t+1}, S_{t+2}, \cdots$
- Actions are random: $\mathbb{P}[A = a \mid S = s] = \pi(a|s).$ (Policy function.)
- States are random: $\mathbb{P}[S' = s' | S = s, A = a] = p(s'|s, a).$ (State transition.)

# Action-Value Functions $Q(s, a)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

**Definition:** Action-value function for policy $\pi$.

- $Q_\pi(s_t, a_t) = \mathbb{E}\left[U_t | S_t = s_t, A_t = a_t\right].$

  - Taken w.r.t. actions $A_{t+1}, A_{t+2}, A_{t+3}, \cdots$ and states $S_{t+1}, S_{t+2}, S_{t+3}, \cdots$
  - Integrate out everything except for the observations: $A_t = a_t$ and $S_t = s_t$.

# Action-Value Functions $Q(s, a)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$

**Definition:** Action-value function for policy $\pi$.

- $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | S_t = s_t, A_t = a_t]$.

**Definition:** Optimal action-value function.

- $Q^\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$.
- Whatever policy function $\pi$ is used, the result of taking $a_t$ at state $s_t$ cannot be better than $Q^\star(s_t, a_t)$.

# Deep Q-Network (DQN)

# Approximate the Q Function

**Goal:** Win the game (≈ maximize the total reward.)

**Question:** If we know $Q^{\star}(s, a)$, what is the best action?

# Approximate the Q Function

**Goal:** Win the game (≈ maximize the total reward.)

**Question:** If we know $Q^\star(s, a)$, what is the best action?

- Obviously, the best action is $a^\star = \underset{a}{\operatorname{argmax}}\, Q^\star(s, a)$.

$Q^\star$ is an indicator of how good it is for an agent to pick action $a$ while being in state $s$.

# Approximate the Q Function

**Goal:** Win the game (≈ maximize the total reward.)

**Question:** If we know $Q^\star(s, a)$, what is the best action?

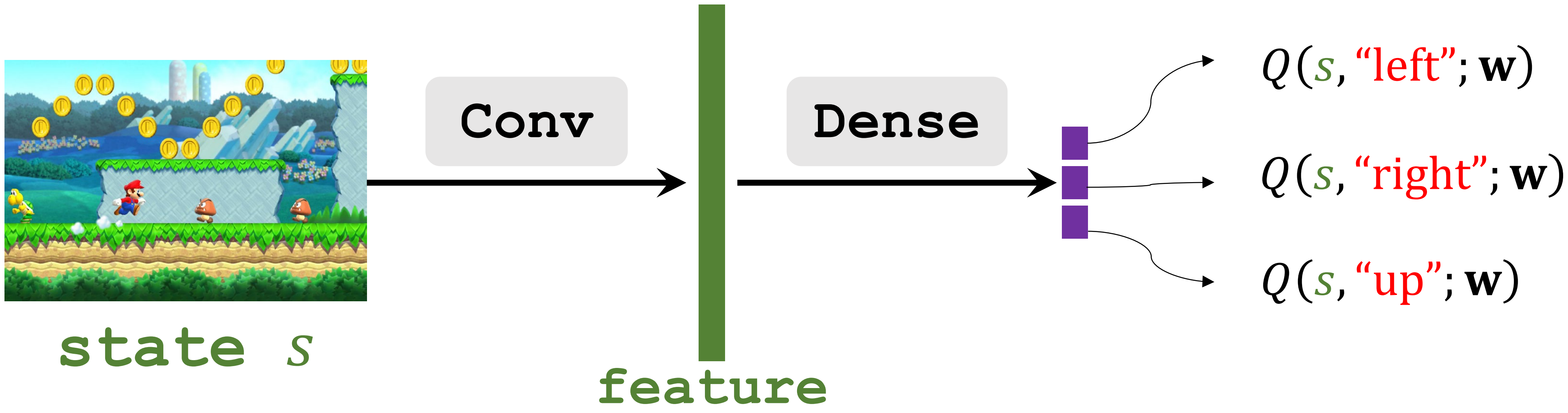
- Obviously, the best action is $a^\star = \underset{a}{\operatorname{argmax}}\, Q^\star(s, a)$.

**Challenge:** We do not know $Q^\star(s, a)$.

- Solution: Deep Q Network (DQN)
- Use neural network $Q(s, a; \mathbf{w})$ to approximate $Q^\star(s, a)$.

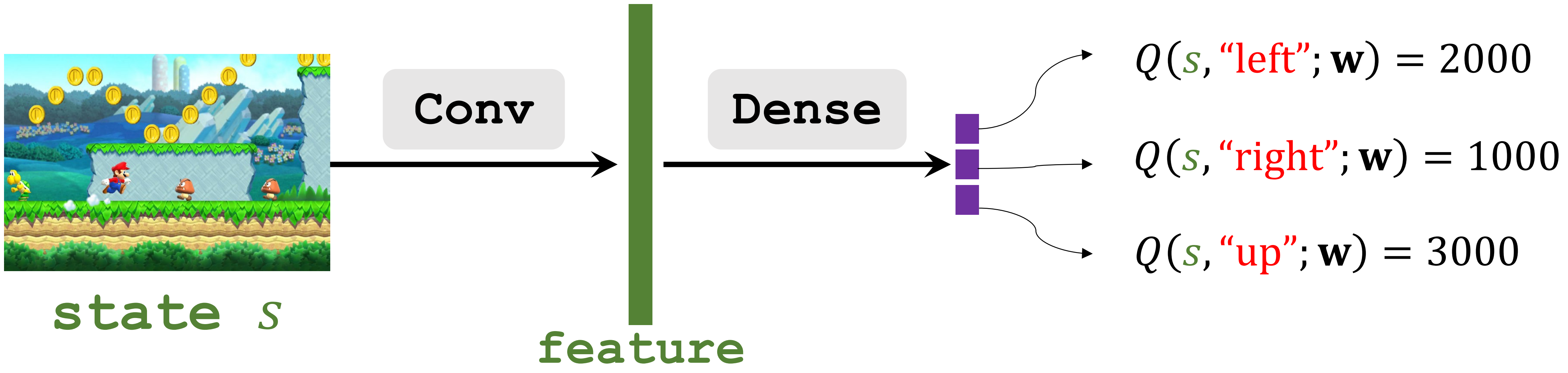
# Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape:  dimension of action space.



# Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape: dimension of action space.



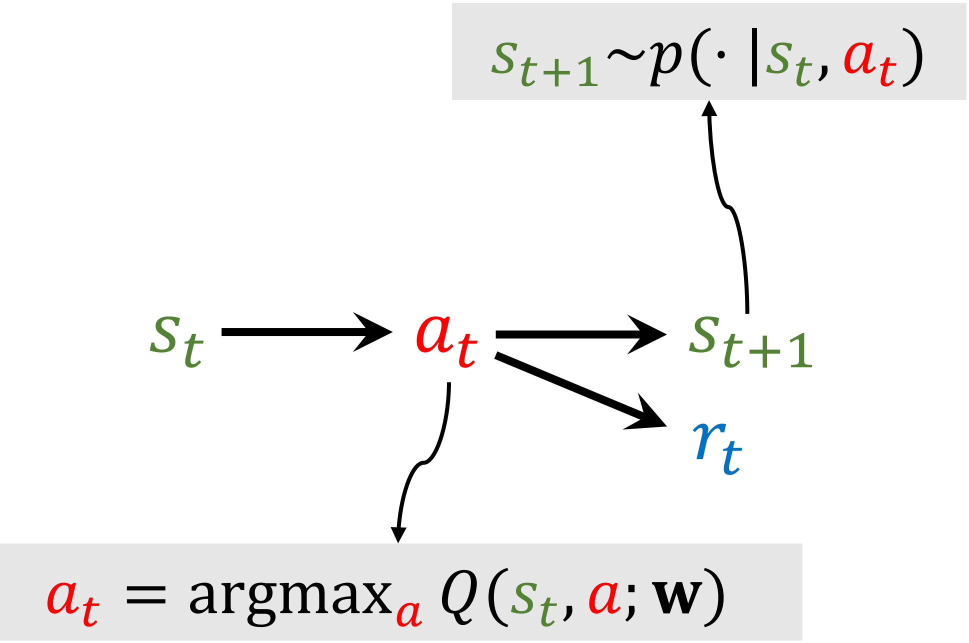

**Question:** Based on the predictions, what should be the action?

# Apply DQN to Play Game

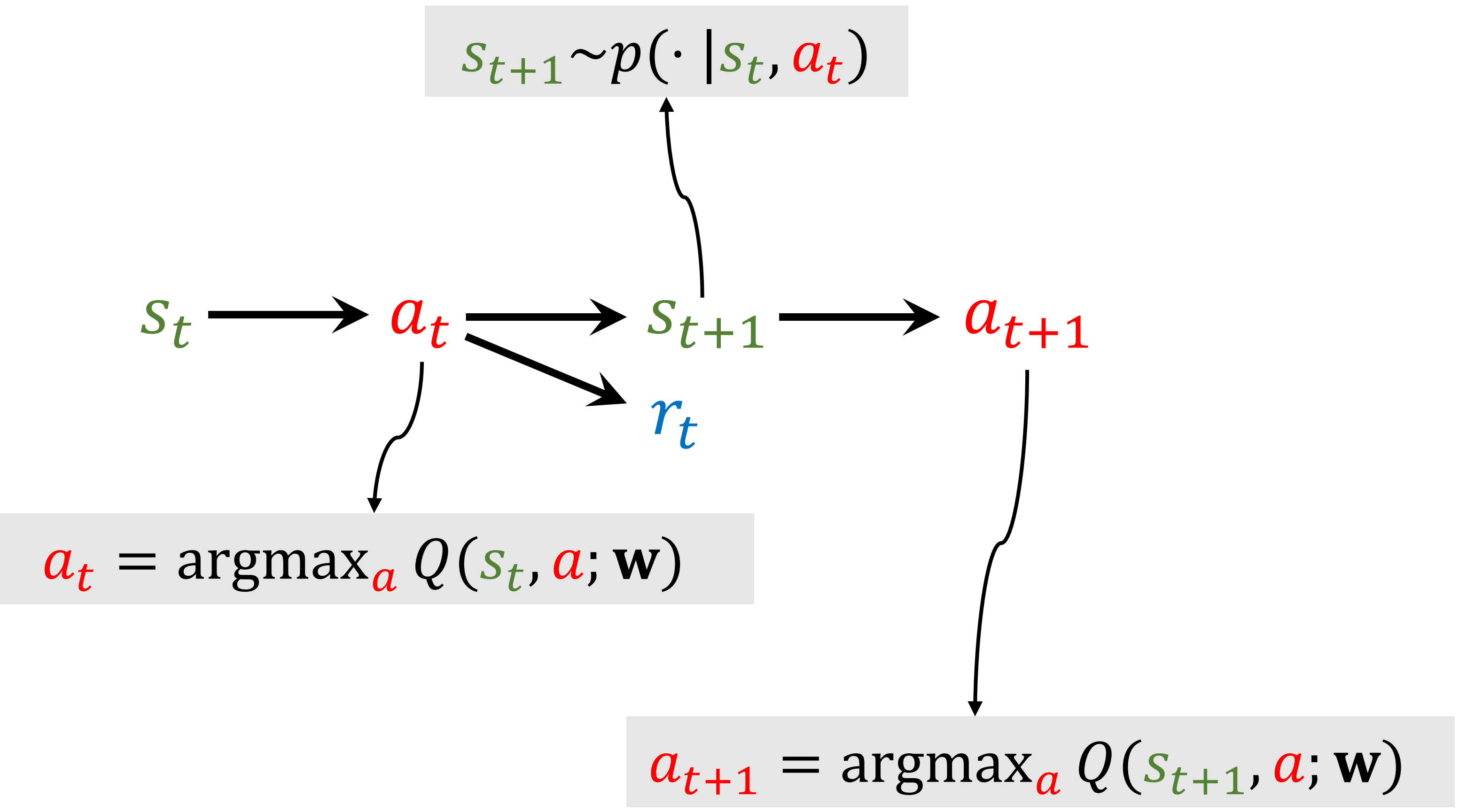
$s_t \longrightarrow a_t$

$$a_t = \mathrm{argmax}_a \, Q(s_t, a; \mathbf{w})$$

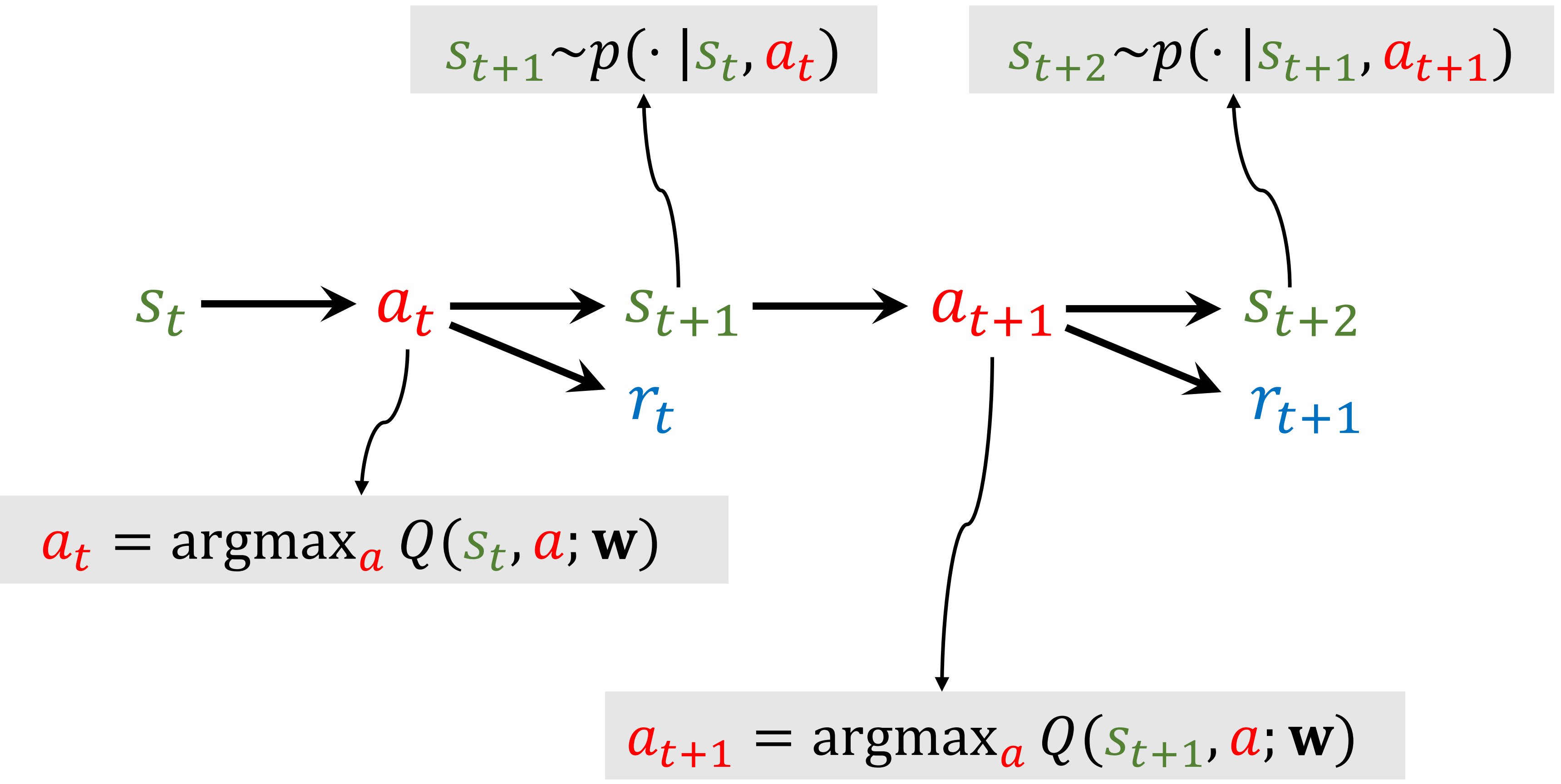
# Apply DQN to Play Game



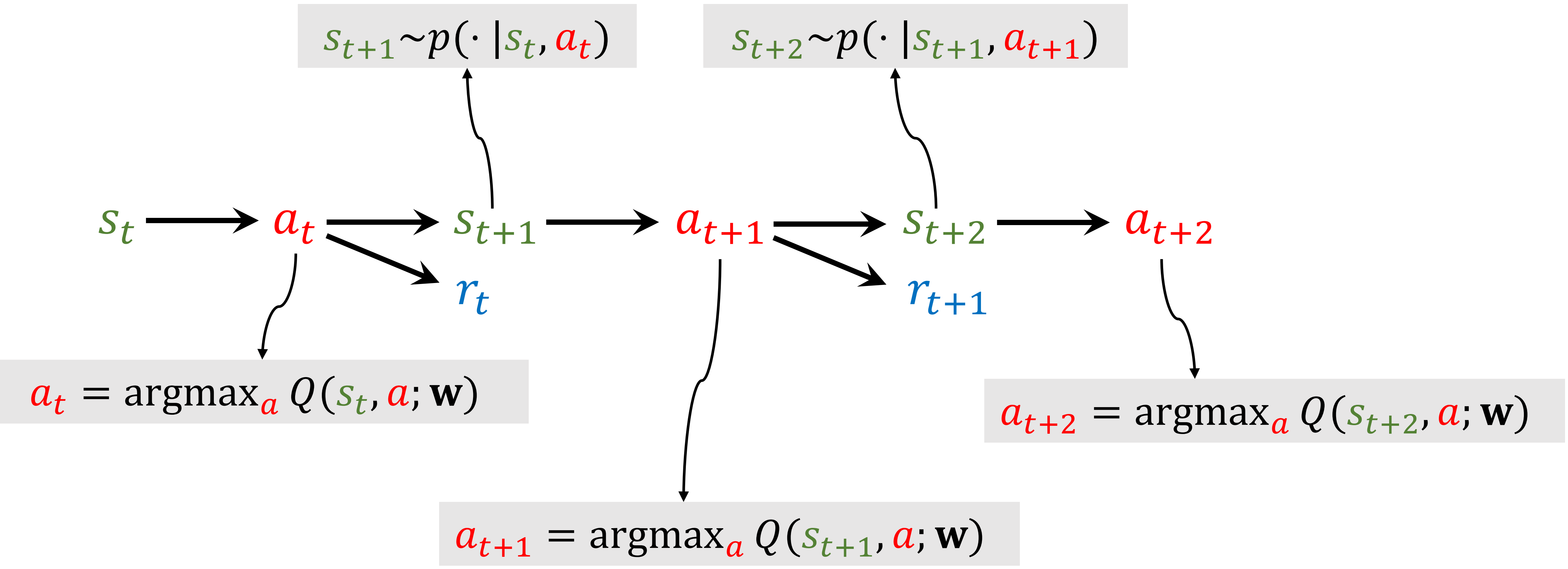
# Apply DQN to Play Game



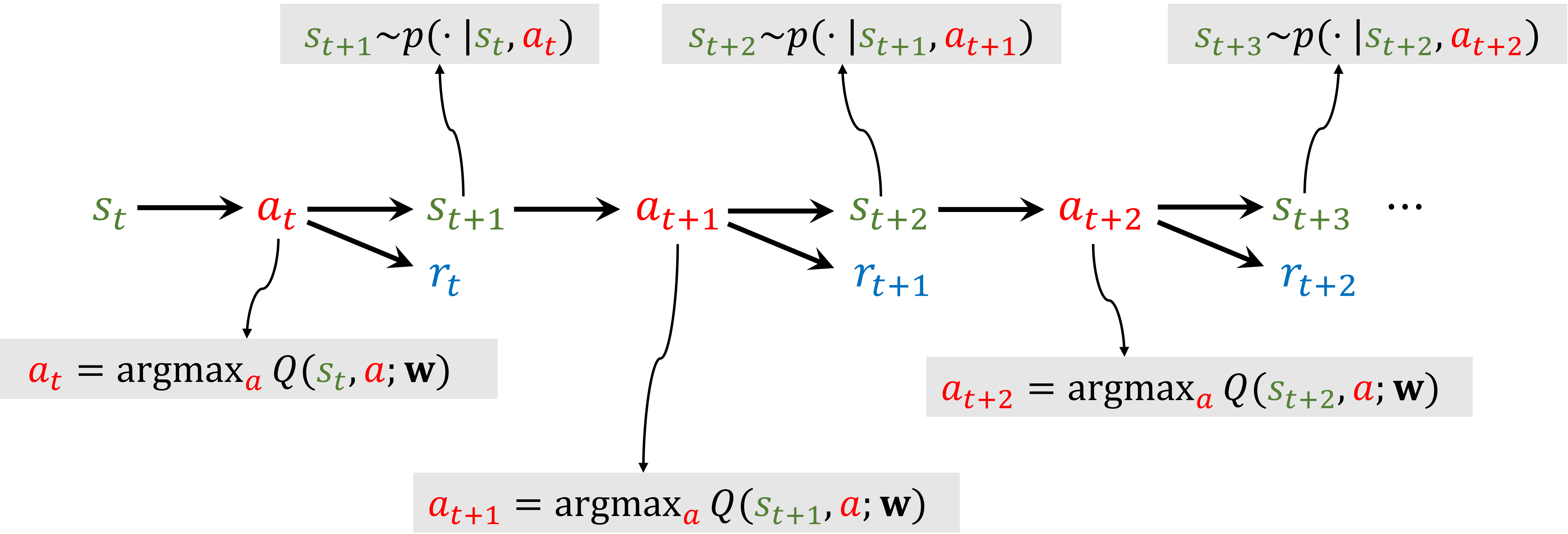
# Apply DQN to Play Game



# Apply DQN to Play Game



# Apply DQN to Play Game



# Temporal Difference (TD) Learning

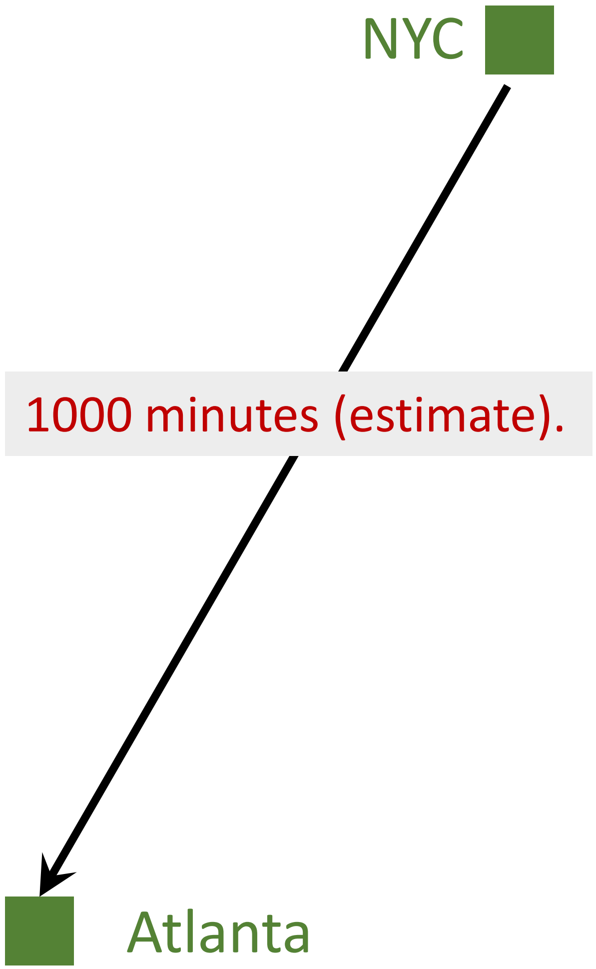
**Reference**


1. Sutton and others: A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In *NIPS*, 2008.
2. Sutton and others: Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *ICML*, 2009.

# Example

- I want to drive from NYC to Atlanta.
- Model $Q(\mathbf{w})$ estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

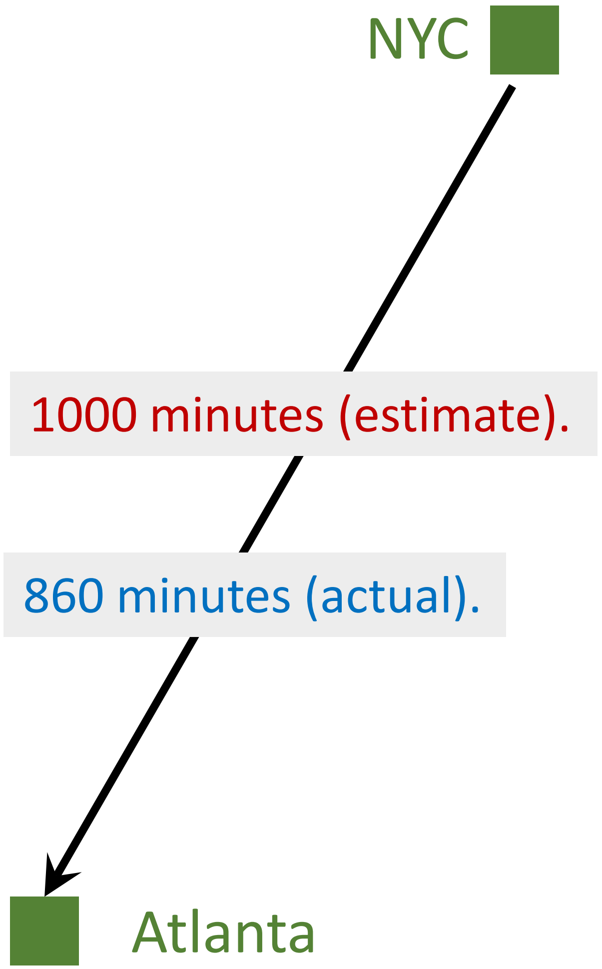


# Example

- I want to drive from NYC to Atlanta.
- Model $Q(\mathbf{w})$ estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

- Make a prediction: $q = Q(\mathbf{w})$, e.g., $q = 1000$.
- Finish the trip and get the target $y$, e.g., $y = 860$.

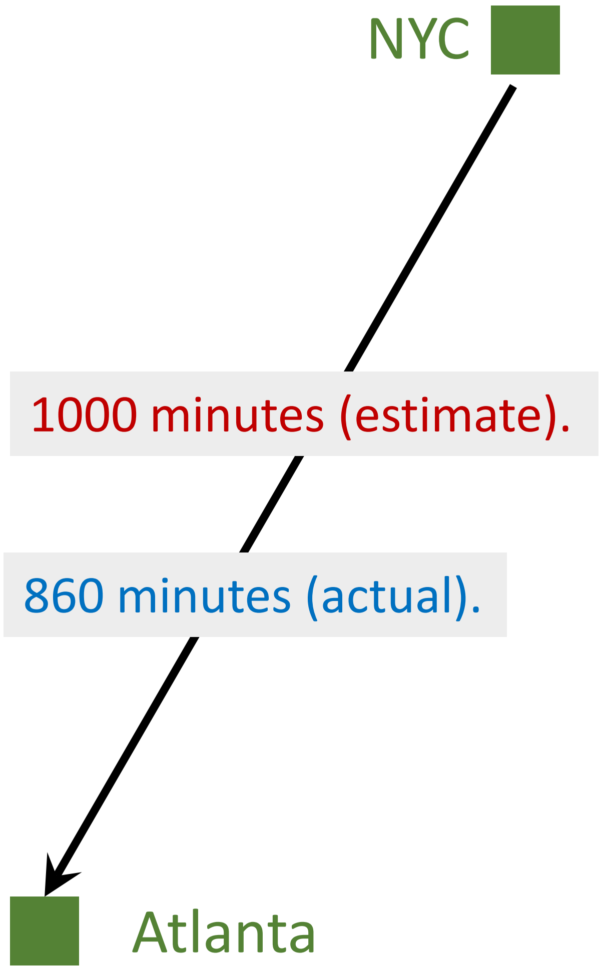


# Example

- I want to drive from NYC to Atlanta.
- Model $Q(\mathbf{w})$ estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

- Make a prediction: $q = Q(\mathbf{w})$, e.g., $q = 1000$.
- Finish the trip and get the target $y$, e.g., $y = 860$.
- Loss: $L = \frac{1}{2}(q - y)^2$.
- Gradient: $\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial q}{\partial \mathbf{w}} \cdot \frac{\partial L}{\partial q} = (q - y) \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.

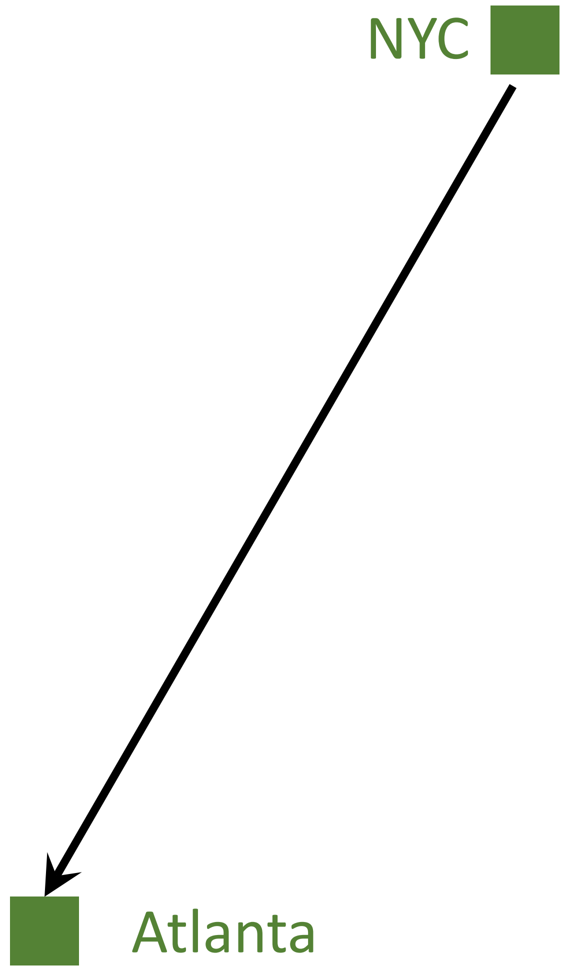


# Example

- I want to drive from NYC to Atlanta.
- Model $Q(\mathbf{w})$ estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

- Can I update the model before finishing the trip?

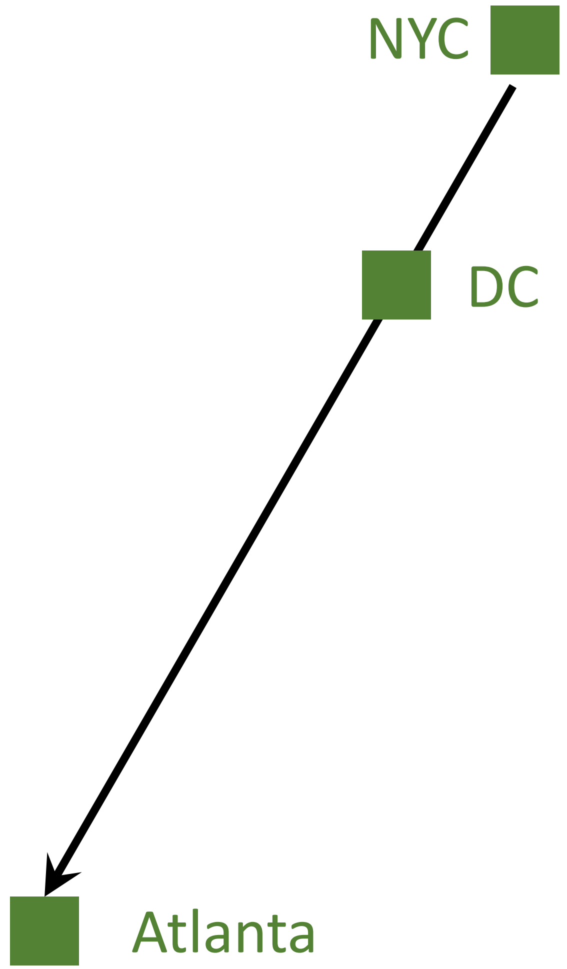


# Example

- I want to drive from NYC to Atlanta (via DC).
- Model $Q(\mathbf{w})$ estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

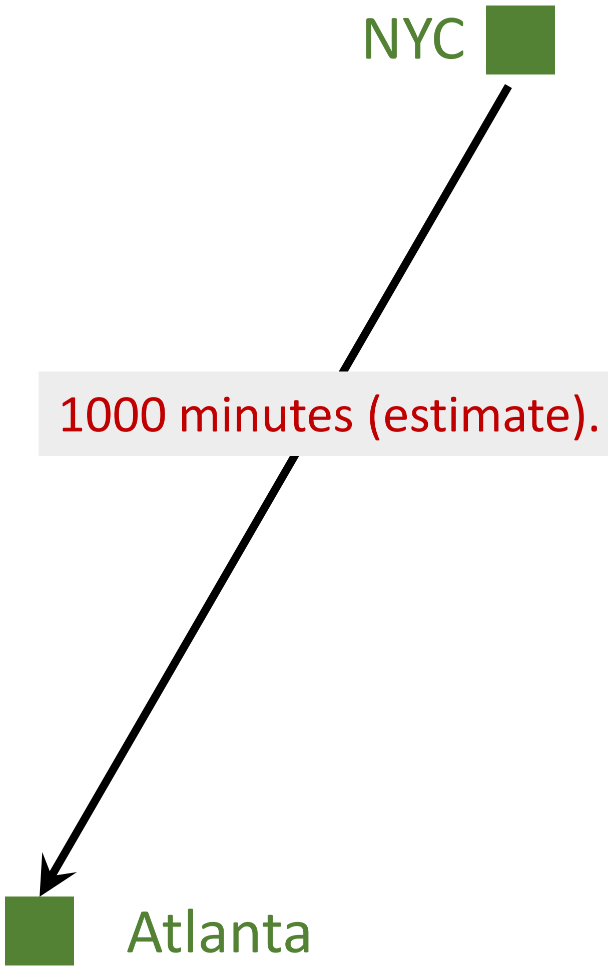
- Can I update the model before finishing the trip?
- Can I get a better $\mathbf{w}$ as soon as I arrived at DC?



# Temporal Difference (TD) Learning

- Model's estimate:

  NYC to Atlanta: 1000 minutes (estimate).



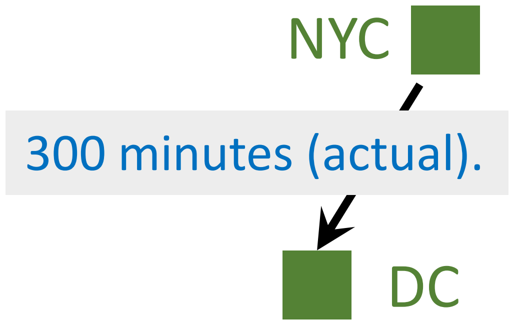
# Temporal Difference (TD) Learning

- Model's estimate:

  NYC to Atlanta: 1000 minutes (estimate).

- I arrived at DC; actual time cost:

  NYC to DC: 300 minutes (actual).



# Temporal Difference (TD) Learning

- Model's estimate:

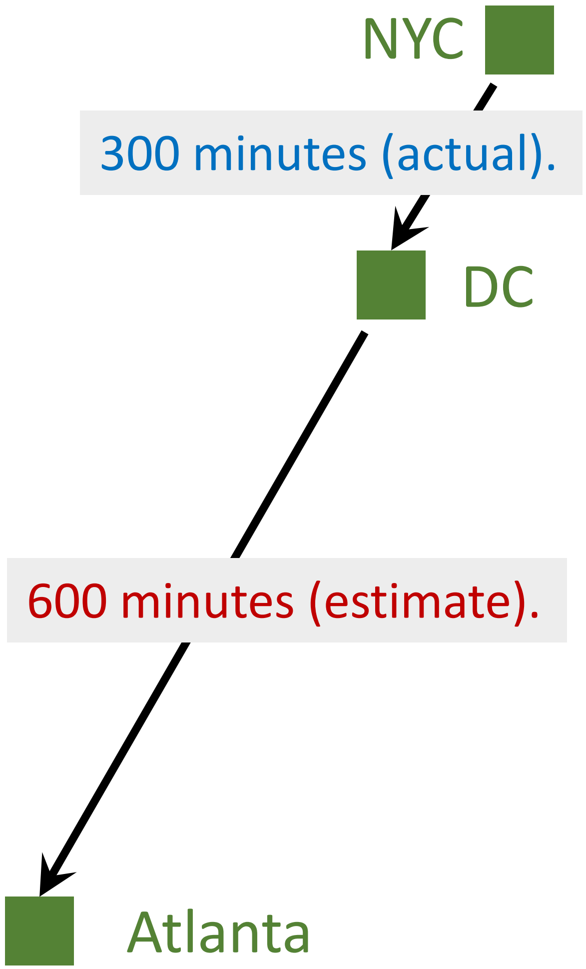
  NYC to Atlanta: 1000 minutes (estimate).

- I arrived at DC; actual time cost:

  NYC to DC: 300 minutes (actual).

- Model now updates its estimate:

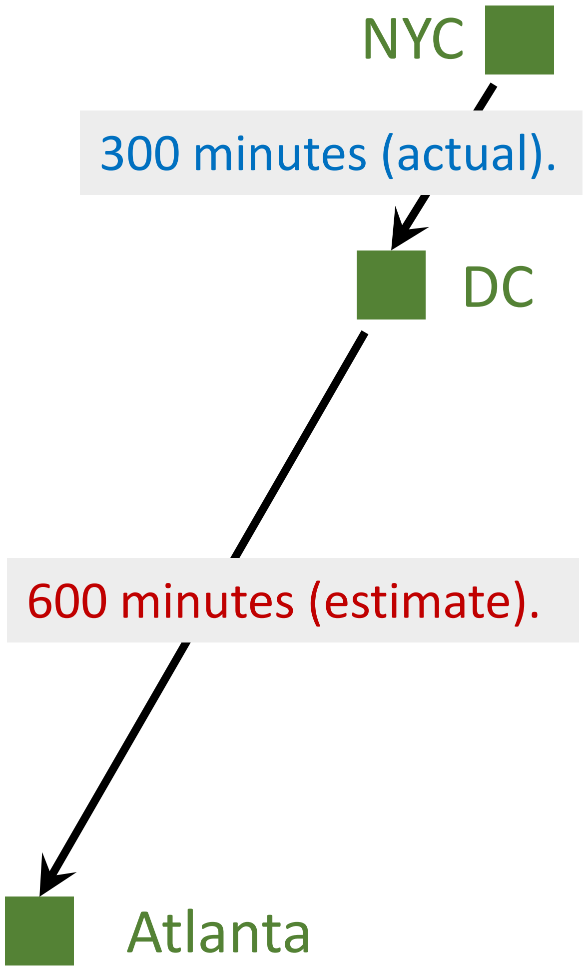
  DC to Atlanta: 600 minutes (estimate).



# Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 1000$ minutes.
- Updated estimate: $300 + 600 = 900$ minutes.

TD target.

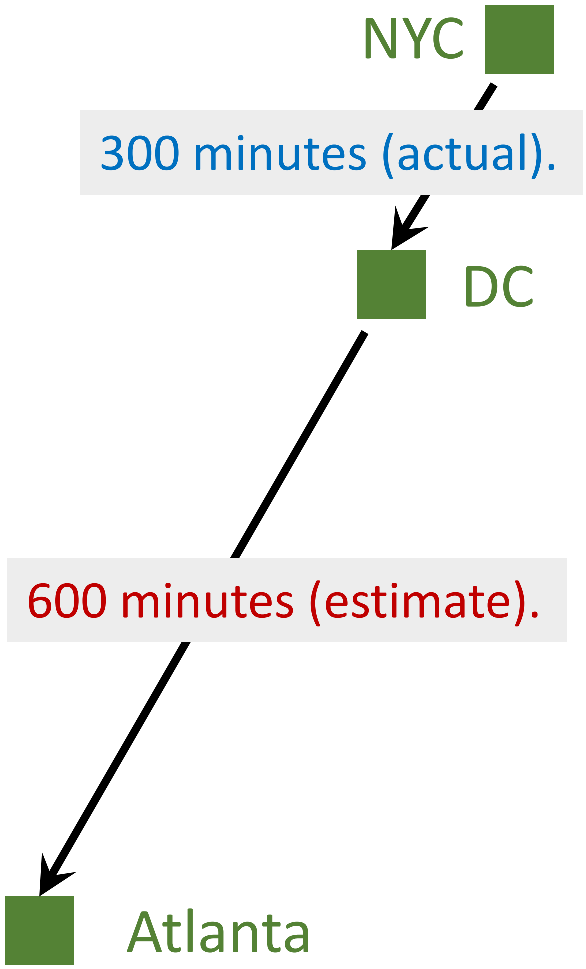


# Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 1000$ minutes.
- Updated estimate: $300 + 600 = 900$ minutes.

TD target.

- TD target $y = 900$ is a more reliable estimate than 1000.



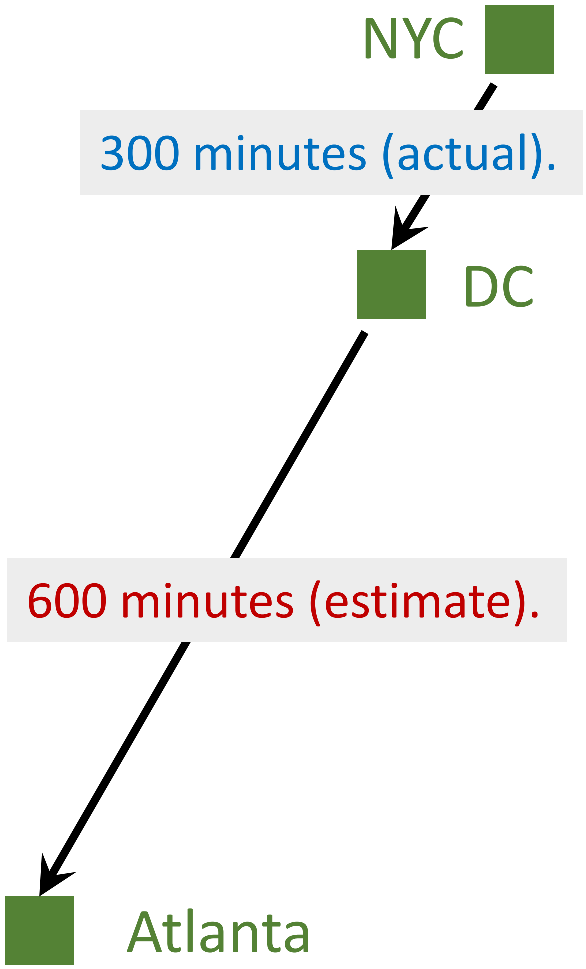
# Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 1000$ minutes.
- Updated estimate: $300 + 600 = 900$ minutes.

TD target.

- TD target $y = 900$ is a more reliable estimate than 1000.
- Loss: $L = \frac{1}{2}(Q(\mathbf{w}) - y)^2$.

TD error

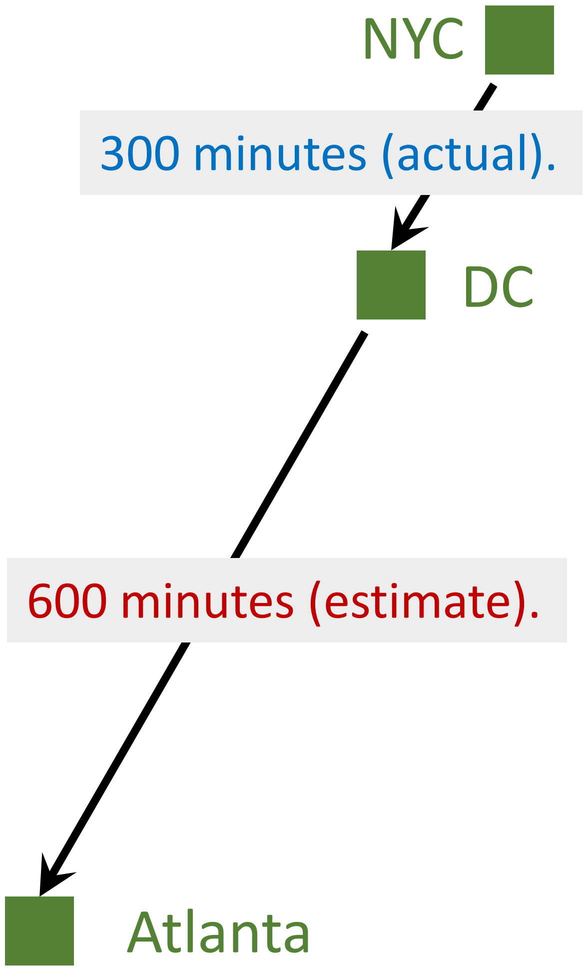


# Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 1000$ minutes.
- Updated estimate: $300 + 600 = 900$ minutes.

TD target.

- TD target $y = 900$ is a more reliable estimate than 1000.
- Loss: $L = \frac{1}{2}(Q(\mathbf{w}) - y)^2$.
- Gradient: $\frac{\partial L}{\partial \mathbf{w}} = \underbrace{(1000 - 900)}_{\text{TD error}} \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$.

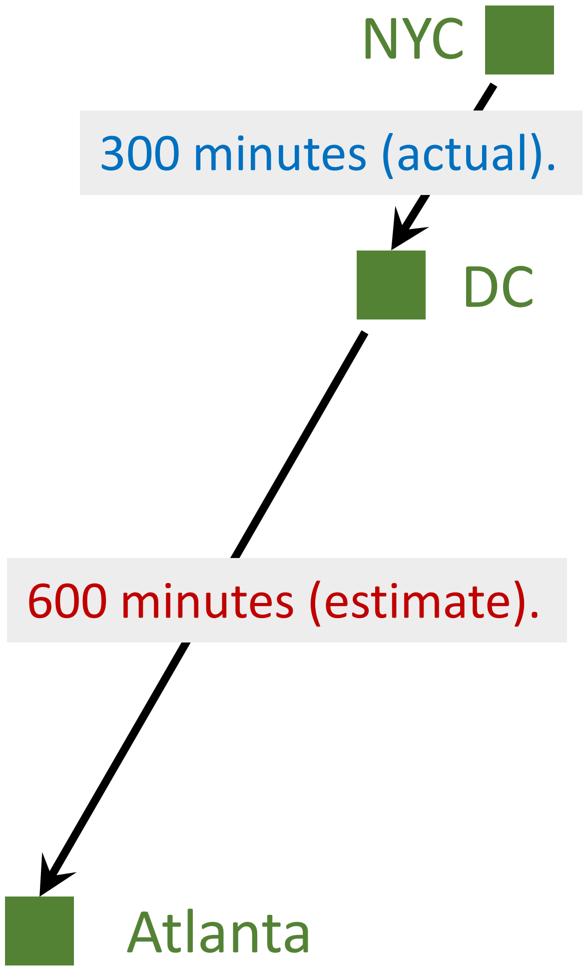


# Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 1000$ minutes.
- Updated estimate: $300 + 600 = 900$ minutes.

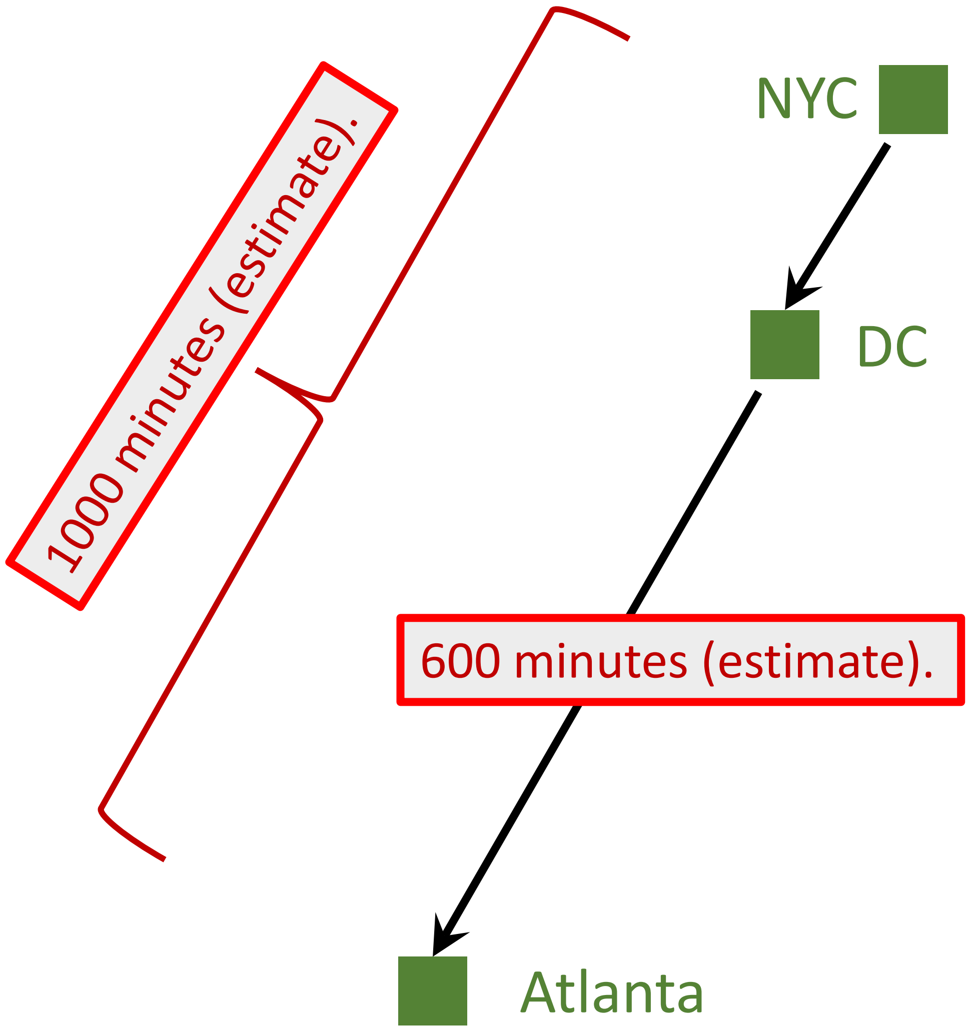
TD target.

- TD target $y = 900$ is a more reliable estimate than 1000.
- Loss: $L = \frac{1}{2}(Q(\mathbf{w}) - y)^2$.
- Gradient: $\frac{\partial L}{\partial \mathbf{w}} = (1000 - 900) \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.



# Why does TD learning work?

- Model's estimates:
  - NYC to Atlanta: 1000 minutes.
  - DC to Atlanta: 600 minutes.
  - ➔ NYC to DC: 400 minutes.

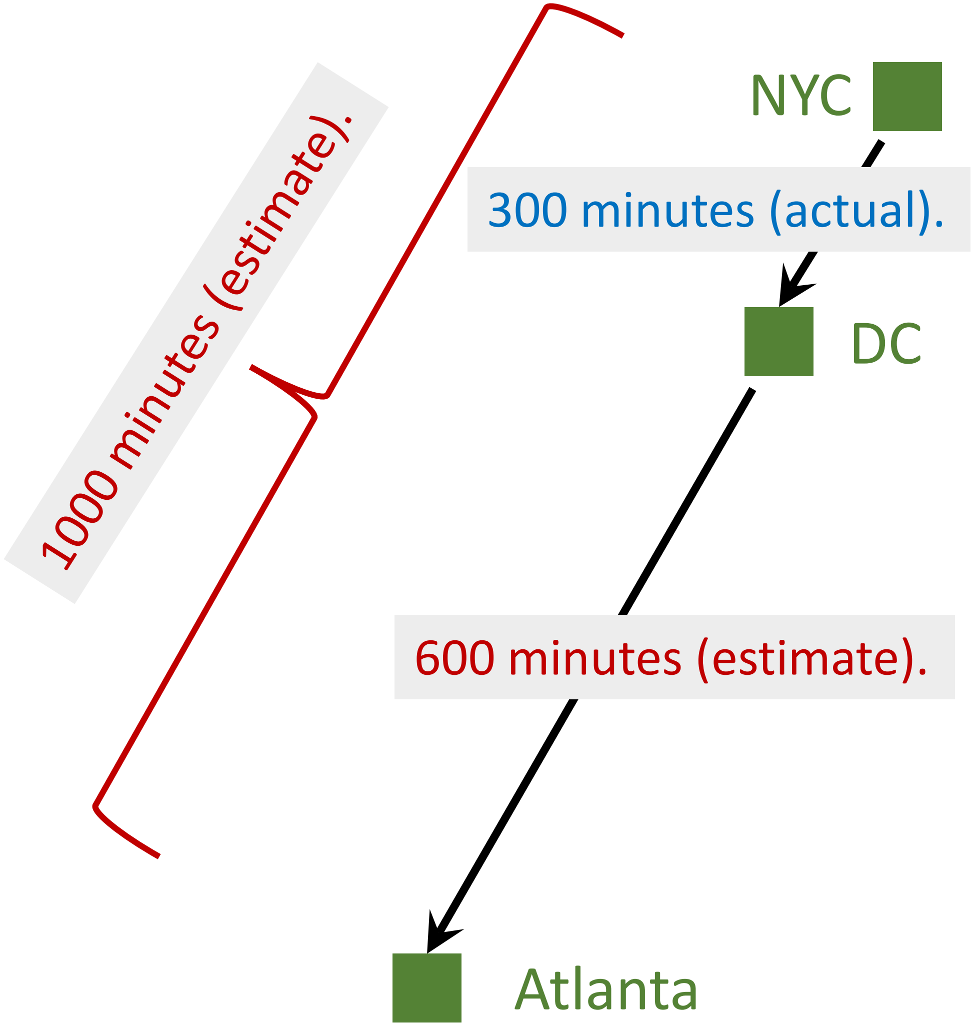


# Why does TD learning work?

- Model's estimates:
  - NYC to Atlanta: 1000 minutes.
  - DC to Atlanta: 600 minutes.
  - ➔ NYC to DC: 400 minutes.

- Ground truth:
  - NYC to DC: 300 minutes.

- TD error: $\delta = 400 - 300 = 100$



# TD Learning for DQN

# How to apply TD learning to DQN?

- In the "driving time" example, we have the equation:

$$T_{\text{NYC}\rightarrow\text{ATL}} \approx T_{\text{NYC}\rightarrow\text{DC}} + T_{\text{DC}\rightarrow\text{ATL}} \,.$$

Model's estimate — Actual time — Model's estimate

# How to apply TD learning to DQN?

- In the "driving time" example, we have the equation:

$$T_{\text{NYC}\rightarrow\text{ATL}} \approx T_{\text{NYC}\rightarrow\text{DC}} + T_{\text{DC}\rightarrow\text{ATL}}.$$

Model's estimate | Actual time | Model's estimate

- In deep reinforcement learning:

$$Q^*(s_t, a_t) \approx r_t + \gamma \cdot \max_a Q^*(s_{t+1}, a).$$

# How to apply TD learning to DQN?

**Identity:** $U_t = R_t + \gamma \cdot U_{t+1}$.

**TD learning for DQN:**

- DQN's output, $Q(s_t, a_t; \mathbf{w})$, is an estimate of $U_t$.
- DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$, is an estimate of $U_{t+1}$.
- Thus, $\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{estimate of } U_t} = \mathbb{E}\left[r_t + \gamma \cdot \underbrace{\max_a Q(S_{t+1}, a; \mathbf{w})}_{\text{estimate of } U_{t+1}}\right]$.

# How to apply TD learning to DQN?

**Identity:** $U_t = R_t + \gamma \cdot U_{t+1}.$

**TD learning for DQN:**

- DQN's output, $Q(s_t, a_t; \mathbf{w})$, is an estimate of $U_t$.
- DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$, is an estimate of $U_{t+1}$.
- Thus,

$$\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{Prediction}} \approx \underbrace{r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w})}_{\text{TD target}}.$$

# Train DQN using TD learning

- Prediction: $Q(s_t, a_t; \mathbf{w}_t)$.
- TD target:

$$y_t = r_t + \gamma \cdot \max_{a} Q(s_{t+1}, a; \mathbf{w}_t).$$

# Train DQN using TD learning

- Prediction: $Q(s_t, a_t; \mathbf{w}_t)$.
- TD target:

$$y_t = r_t + \gamma \cdot \max_{a} Q(s_{t+1}, a; \mathbf{w}_t).$$

- Loss: $L_t = \frac{1}{2}[Q(s_t, a_t; \mathbf{w}) - y_t]^2$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial\, L_t}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.

# Explore the Environment

- **ε-greedy policy:**
  - With probability **ε**, the agent chooses a random action (exploration).
  - With probability **1 - ε**, the agent chooses the action that has the highest predicted Q-value (exploitation).
- **Decaying ε:** Often, DQN uses an **annealing strategy** where ε starts high (favoring exploration) and gradually decreases over time (favoring exploitation as the agent learns more).

# Summary

# Temporal Difference (TD) Learning

**Algorithm:** One iteration of TD learning.

1. Observe state $S_t = s_t$ and perform action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w}_t)$.
3. Differentiate the value network: $\mathbf{d}_t = \frac{\partial\, Q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.

# Temporal Difference (TD) Learning

**Algorithm:** One iteration of TD learning.

1. Observe state $S_t = s_t$ and perform action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w}_t)$.
3. Differentiate the value network: $\mathbf{d}_t = \frac{\partial\, Q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.
4. Environment provides new state $s_{t+1}$ and reward $r_t$.
5. Compute TD target: $y_t = r_t + \gamma \cdot \max_{a} Q(s_{t+1}, a; \mathbf{w}_t)$.

# Temporal Difference (TD) Learning

**Algorithm:** One iteration of TD learning.

1. Observe state $S_t = s_t$ and perform action $A_t = a_t$.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w}_t)$.
3. Differentiate the value network: $\mathbf{d}_t = \frac{\partial\, Q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.
4. Environment provides new state $s_{t+1}$ and reward $r_t$.
5. Compute TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$.
6. Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t$.

# Process of Q Learning

- Initialize network $Q(s, a; \mathbf{w})$
- Repeat:
  - Observe the current state $s_t$
  - Choose an action (**ε-greedy** strategy): select action $a_t$ using an **exploration policy**:
    - With probability $\varepsilon$, choose a random action (exploration).
    - With probability $1 - \varepsilon$, choose the action with the highest $Q(s_t, a_t; \mathbf{w})$ (exploitation).
  - Take the action and observe the reward
  - Update $Q(s, a; \mathbf{w})$ using TD learning
- After training, the **optimal policy** is:

$$\pi^*(s) \coloneqq \arg\max_a Q(s, a; \mathbf{w})$$

# Play Breakout using DQN

(The video was posted on YouTube by DeepMind)

# Thank you!