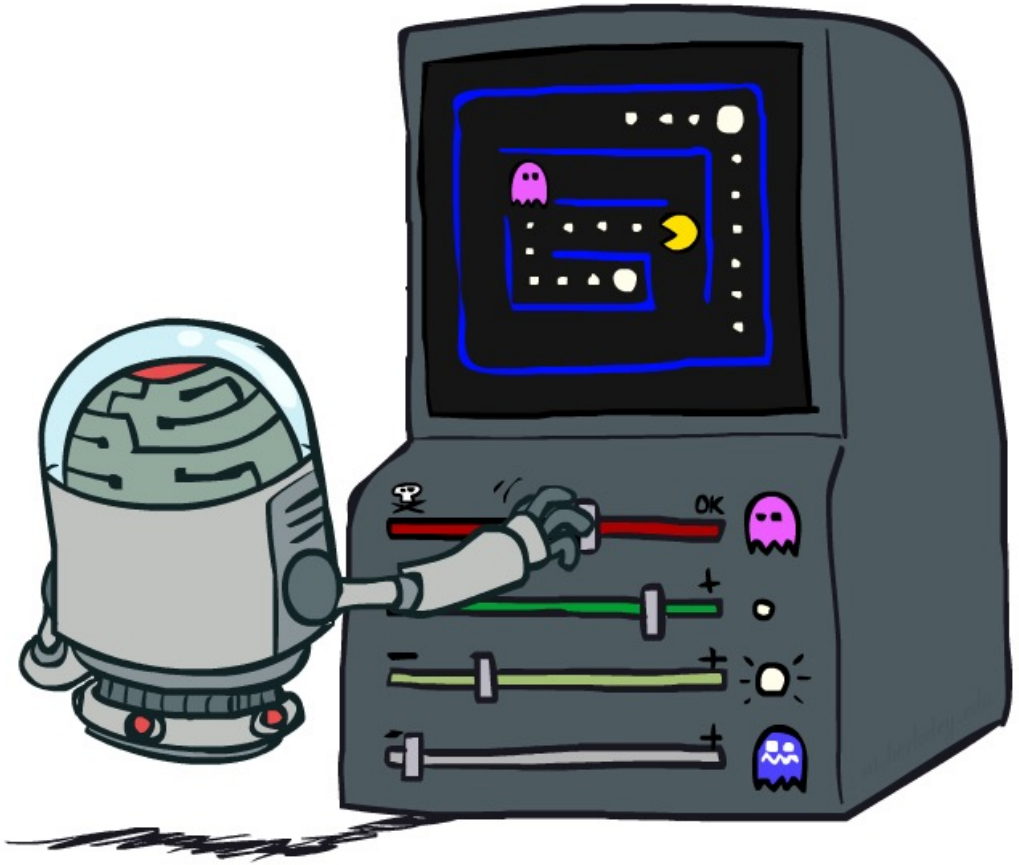
SAMSUNG DISPLAY 강화학습실습

SAMSUNG DISPLAY

발표자 소개

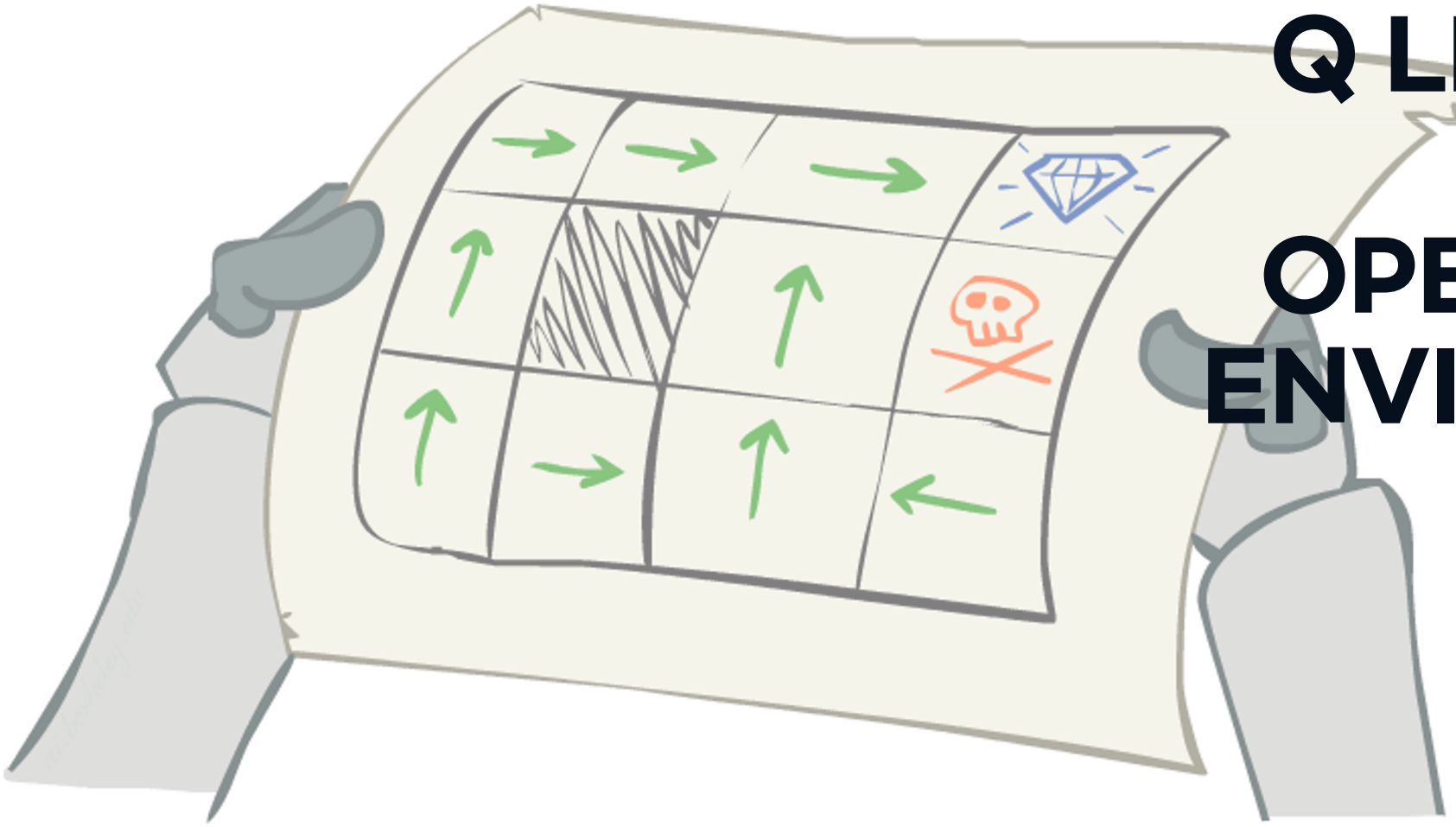
- 김승현
- 성균관대학교 CSI 연구실 석사과정
- Email: kim95175@gmail.com



1 **Q Learning and OpenAI Gym Environment**

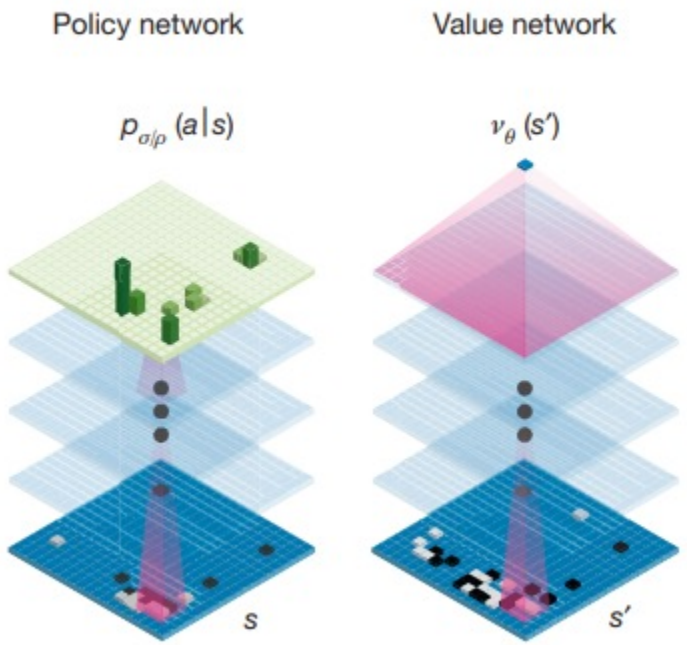
2 **Q Learning with FrozenLake**

3 **Q Learning with Stochastic FrozenLake**



Q LEARNING AND OPENAI GYM ENVIRONMENT

Reinforcement Learning



현재 state에서

어떤 action을 선택하는 것이

최선일까?

Q LEARNING REVIEW

현재 state에서

어떤 action을 선택하는 것이

최선일까?

= Learning with *trial and error*

Q LEARNING REVIEW

- Interacting with Environment using $\epsilon-greedy$ w.r.t. $Q(s,a)$
- Updating the policy using $greedy$ w.r.t. $Q(s,a)$

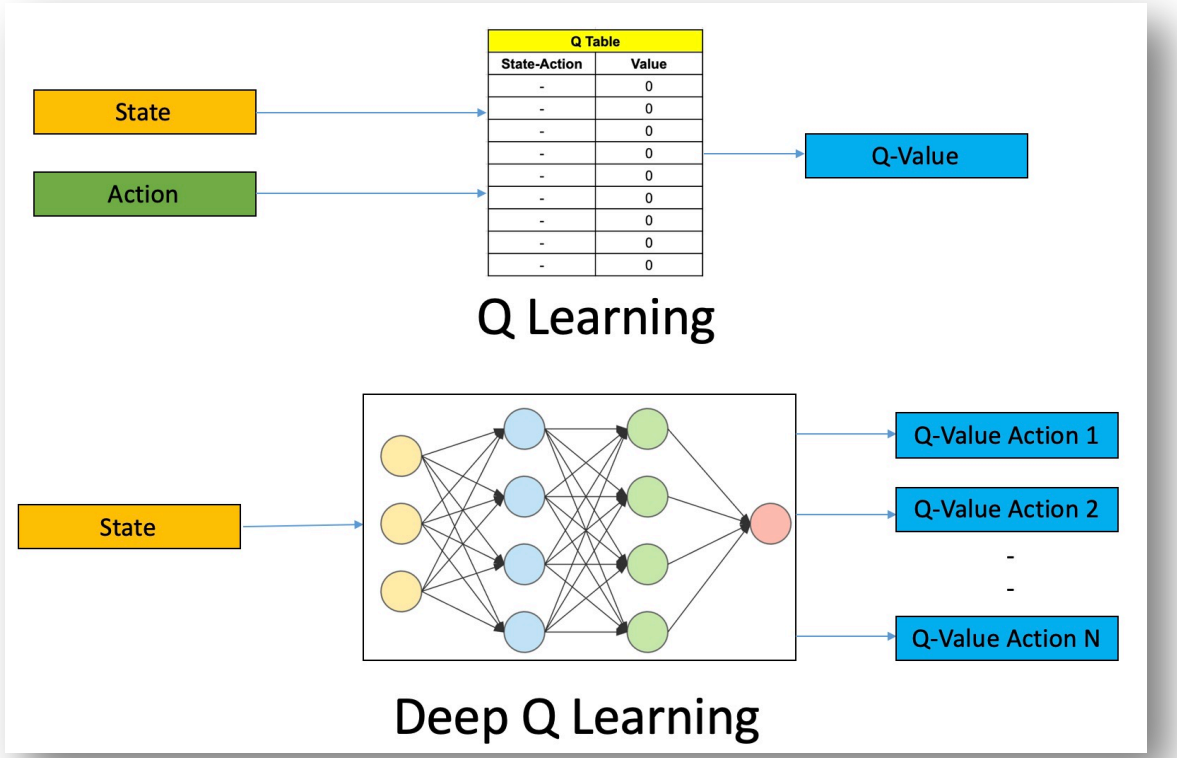
$$\pi(S_t) = \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')$$

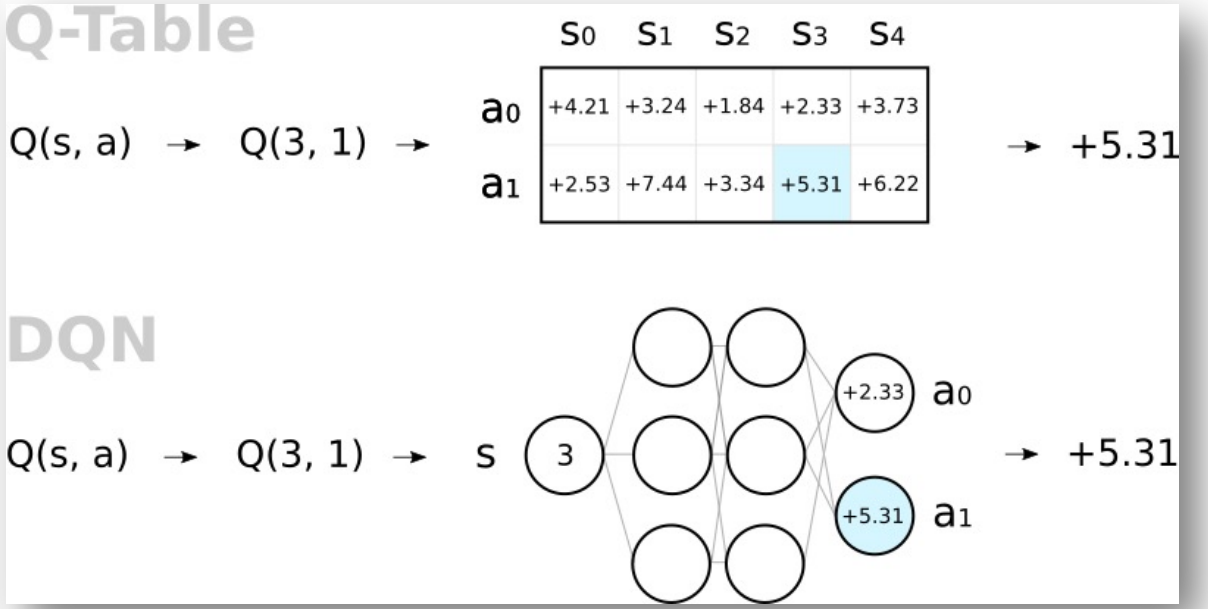
- The Q-learning target then simplifies:

$$R_{t+1} + \gamma Q(S_{t+1}, A') = R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a'}{\operatorname{argmax}}\, Q(S_{t+1}, a')\right)$$

$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

Q LEARNING REVIEW





Q LEARNING ALGORITHM

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

0. **Initialize** Q *table* and parameters

Q LEARNING ALGORITHM

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

1. **Select action** using epsilon-greedy

Q LEARNING ALGORITHM

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

2. **Take action** and get observations

Q LEARNING ALGORITHM

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

3. **Update** q table

ENVIRONMENT

Definition

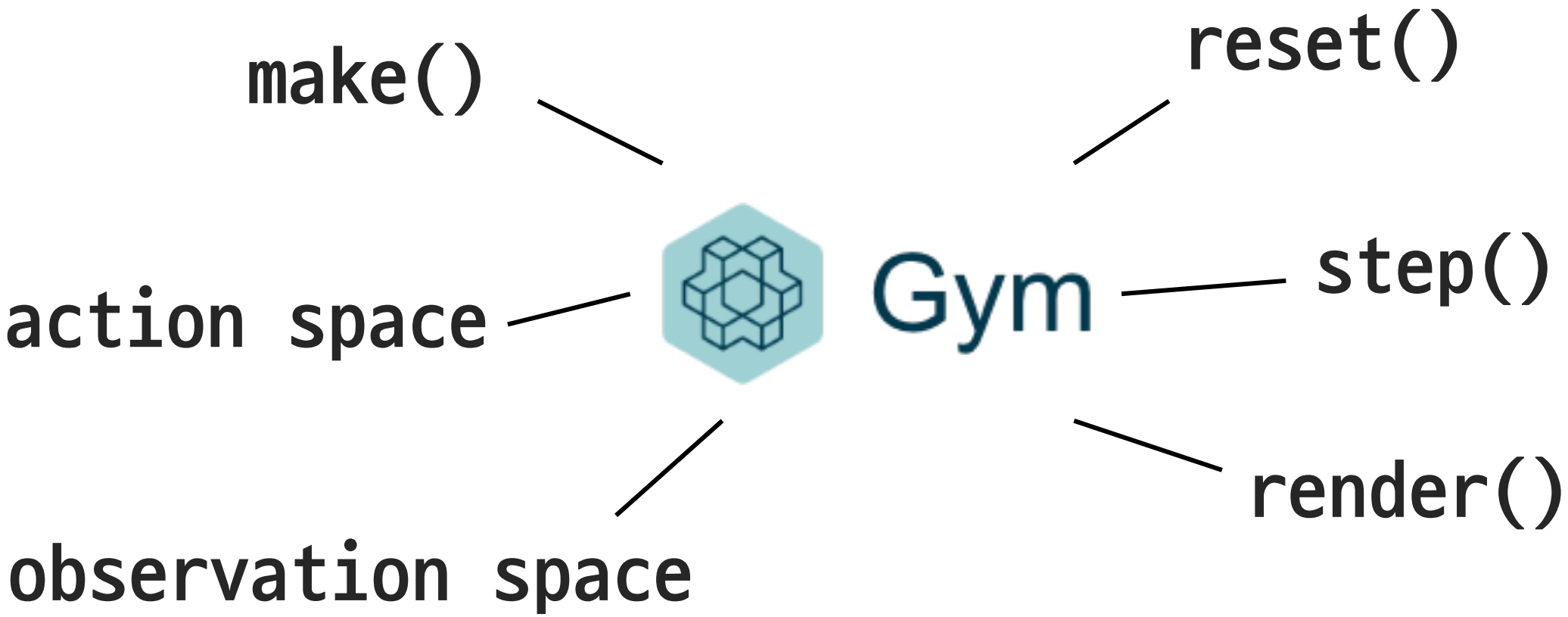
A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$.

OPENAI GYM

OpenAI Gym toolkit provides a set of physical simulation environments, games and robot simulators that we can play with and design reinforcement learning agents for.

OPENAI GYM



OPENAI GYM


```
gym.make( “{env_name}” )
```

등록된 environment를 만드는 함수

추가적인 argument도 사용할 수 있음

env = gym.make(‘FrozenLake-v0’, is_slippery=Fal

se)

OPENAI GYM

observation space
action space

state와 action이 정의되는 집합을 나타냄

개수가 명확히 정해져 있는 *Discrete*

연속적인(continuous) 공간들을 나타내는 *Box*

env.observation_space

env.action_space

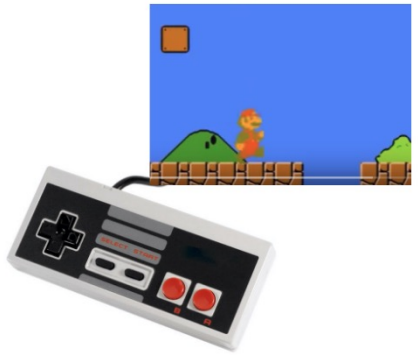
	Space Types	About
1.	Box	A *n* dimensional box, where each coordinate lies between a bound defined by [low,high].

Independent bound for each dimension:
>>> space = gym.spaces.Box(low=np.array([-1.0, -2.0]), high=np.array([2.0, 4.0]), dtype=np.float32)
>>> print(space, space.sample())
Box(2,) [0.37700886 -0.12235405], returns a 2 element vector, each corresponding to one dimension, the 1st dimension value is sampled from interval [-1.0, 2.0] and the 2nd dimension sampled from interval [-2.0, 4.0].
Identical bound for each dimension:
>>> space = gym.spaces.Box(low=-1.0, high=2.0, shape=(3, 4), dtype=np.float32)
>>> print(space,space.sample())
Box(3, 4) [[0.29874545 0.84560674 1.102614 1.6697267]
[1.4066843 1.8870455 0.48101822 1.7307336]
[-0.49271297 0.8122731 -0.7641185 -0.45864782]]
returns a tensor or a 3 x 4 multidimensional matrix.		

2.	Discrete	The space consists of n distinct points each mapped to an integer value in the interval [0, n-1].
>>> gym.spaces.Discrete (4), **Discrete(4), t**he defined space will contain 4 discrete points with each point mapped to an integer in the interval [0,3].		

OPENAI GYM

Discrete vs Continuous



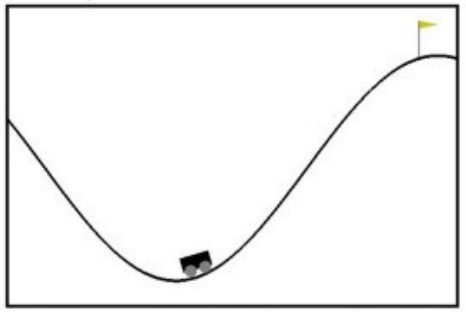
Discrete action
[up, down, A, B] = [0, 0, 1, 0]

Types of State and Action Spaces
(Observation space & State space are same for most gym environments.)

Grid World

Discrete State/ Observation Space
[1,2,...,6 non terminal states]
Discrete Action Space
[Up, Down, Left & Right]

MountainCarContinuous-v0*



Continuous State/ Observation Space
[Car location, velocity] = [(-1.2 to 0.6), (-0.07 to 0.07)]
Continuous Action Space
[-1.0 to 1.0]

* MountainCar-v0 is the discrete action space version of this.

Continuous action
[축 1, 축 2, 축 3] = [0.1, 0.5, 0.3]

OPENAI GYM

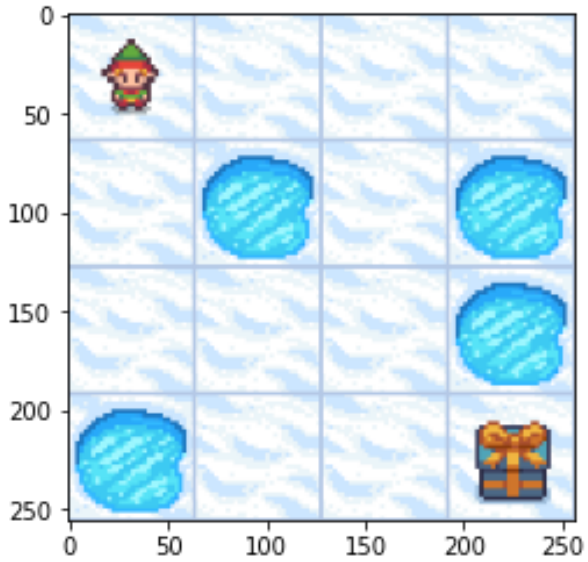
env.reset( )

환경을 초기 상태로 초기화하는 함수

episode를 시작할 때 호출

첫 state를 return함

state = env.reset()

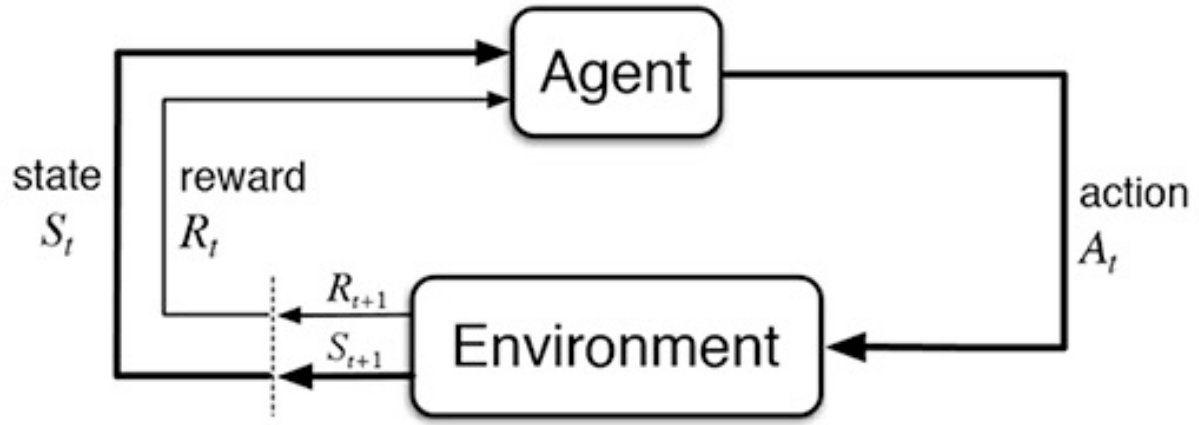


OPENAI GYM

`env.step(action)`

현재 state에서 action을 행하는 함수

next_state, reward, done, info를 반환함



OPENAI GYM

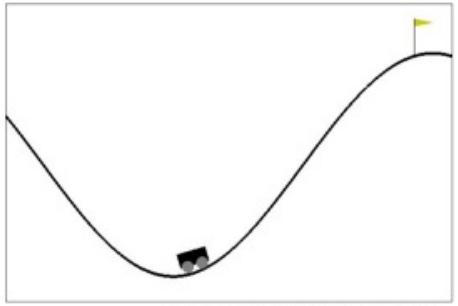
env.render( )

현재 environment 상태를 visualize하는 함수

Environment가 잘 작동하는지

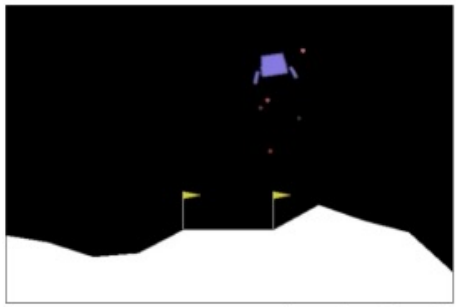
Agent가 잘 학습되었는지

디버깅에 큰 도움이 됨

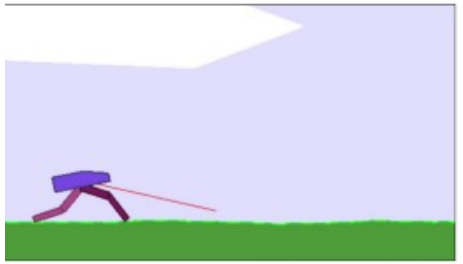


MountainCarContinuous-v0

]



LunarLander-v2



BipedalWalker-v2

Q LEARNING ALGORITHM

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

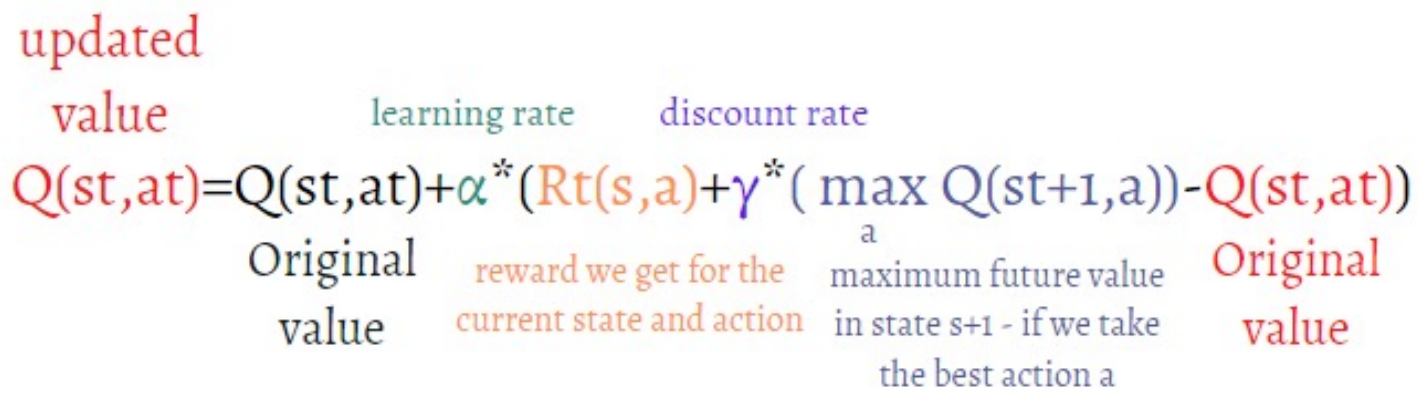
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

Q LEARNING

0. (state, action)의 Quality를 뜻하는 Q Table 생성

np.ones([*# of State*, *# of Action*])

1. Epsilon-greedy 를 통해 Action 선택
2. Take Action
3. Update Q Value



EPSILON-GREEDY EXPLORATION

일정 확률🎲로

random.random()

랜덤한 액션 (exploration)

env.action_space.sample()

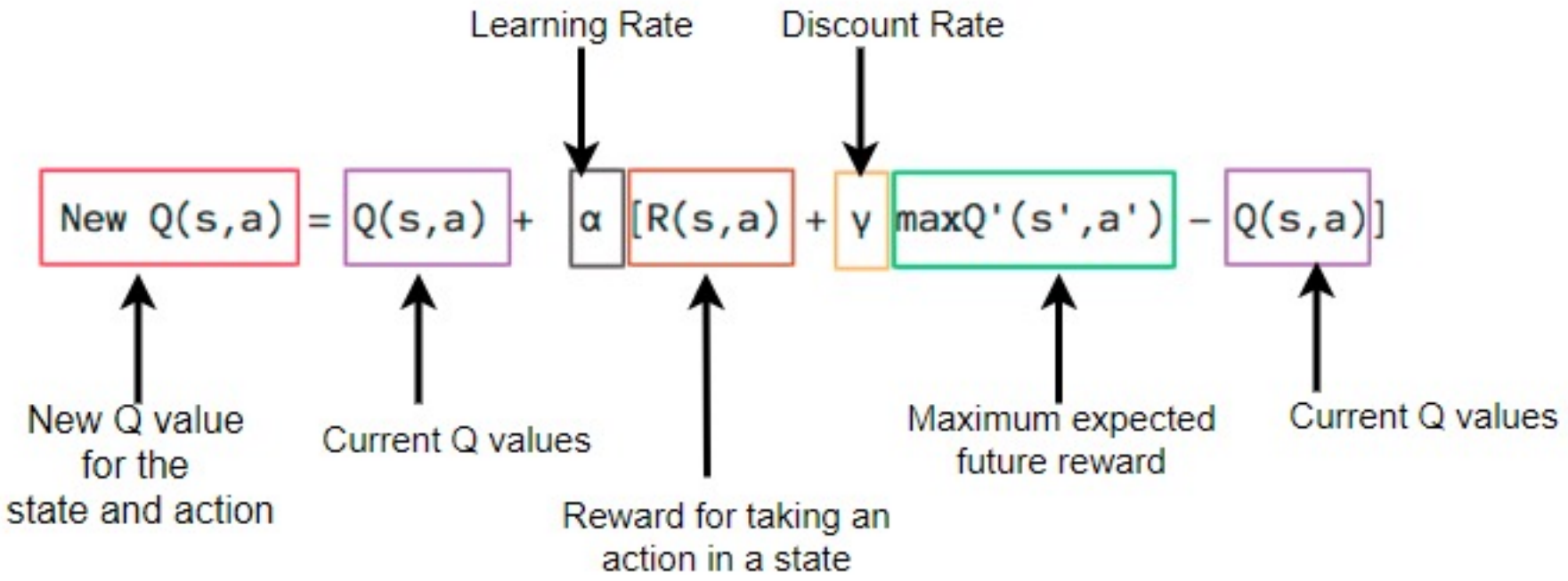
최선의 액션을 선택 (exploitation)

np.argmax(q_table[state])

```
eps = 0.1

if rand() > eps :
        a = argmax( Q(s,a) )
else:
        a = random_action()
```

Update Q Value



Q table

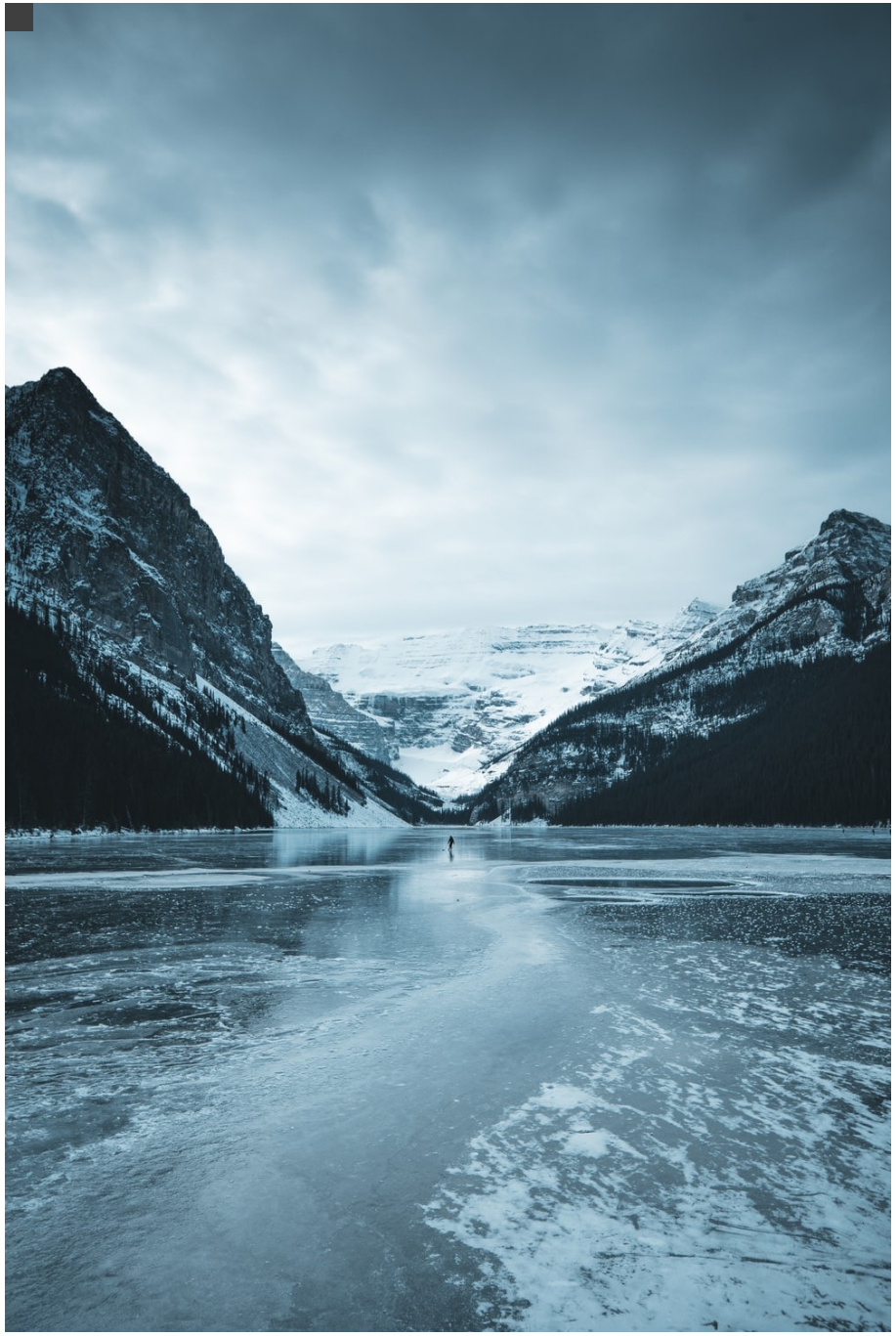
state, action

next state

reward

learning rate

Discount rate



Q LEARNING WITH FROZEN LAKE

FROZNE LAKE WITH Q LEARNING

GOAL

Q Learning Code 구현

OpenAI Gym FrozenLake-v0

환경에 적용

ADVANCED

Non-Deterministic (Stochastic)

환경에 적용

FROZEN LAKE

출발지에서 도착지로 이동하는 것이 목표

상하좌우로 이동 가능

구멍에 빠지면 Game Over

FROZEN LAKE

State
= Discrete(16)

0~15의 정수로 결정되는 Discrete Space

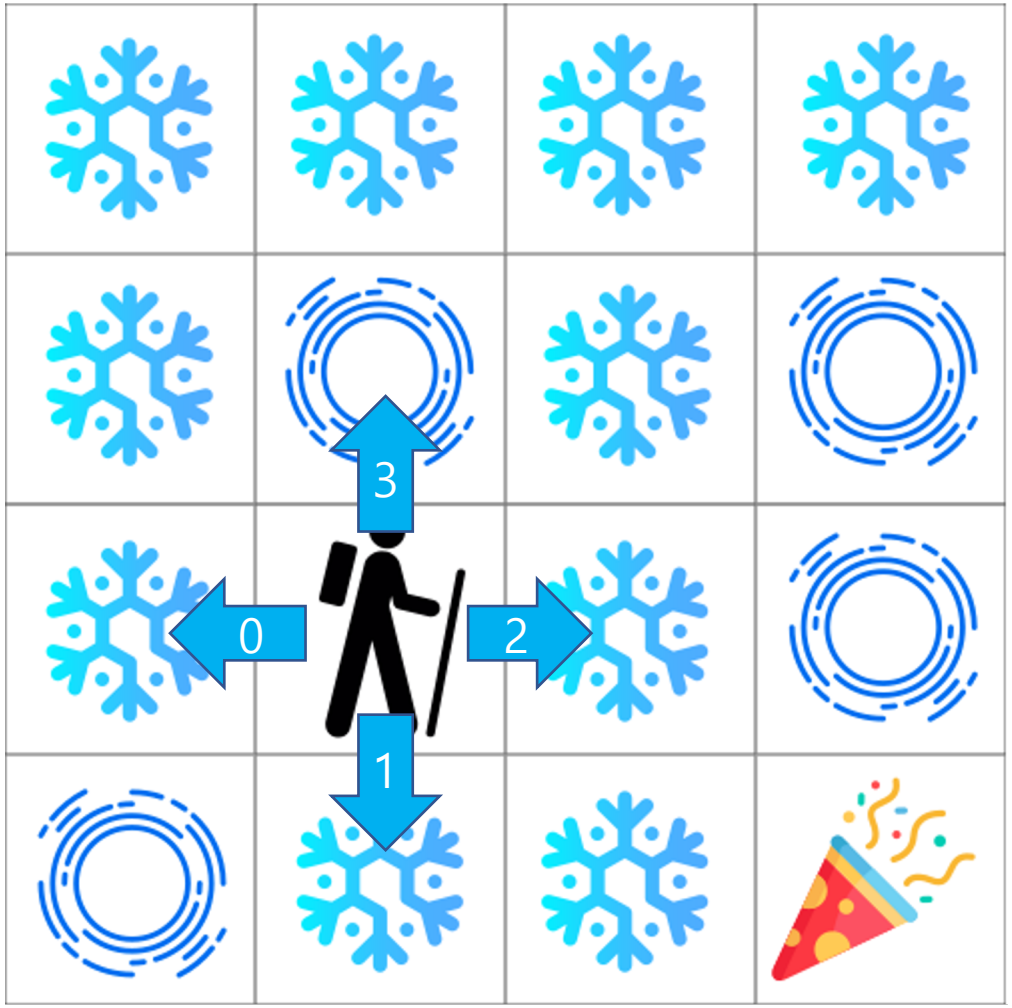
현재 좌표 (x, y)

state = y * 4 + x

FROZEN LAKE

Action
= Discrete(4)

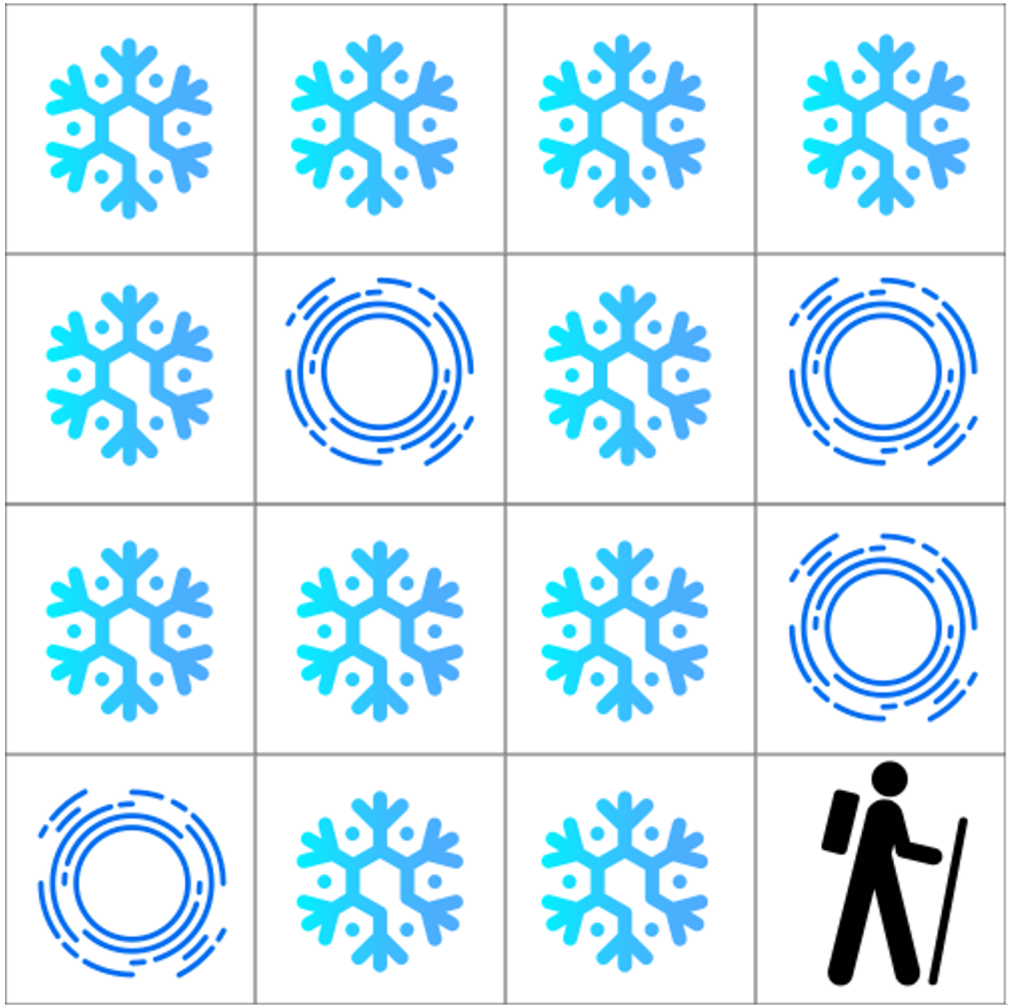
↑
← ↓ →



FROZEN LAKE

Reward

목적지에 도착할 경우 +1



TEST AND DEBUG

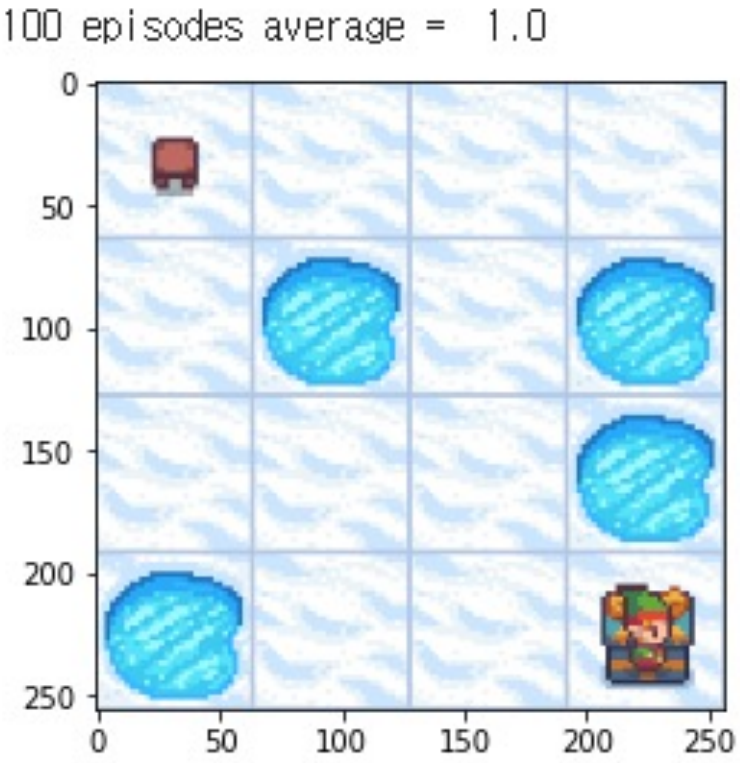
에이전트가 잘 학습되었는지 평가하는 단계

Test시 epsilon-greedy를 사용하지 않음

env.render()을 사용해서 visualization

Environment에 이상이 없는가

Agent에 이상이 없는가



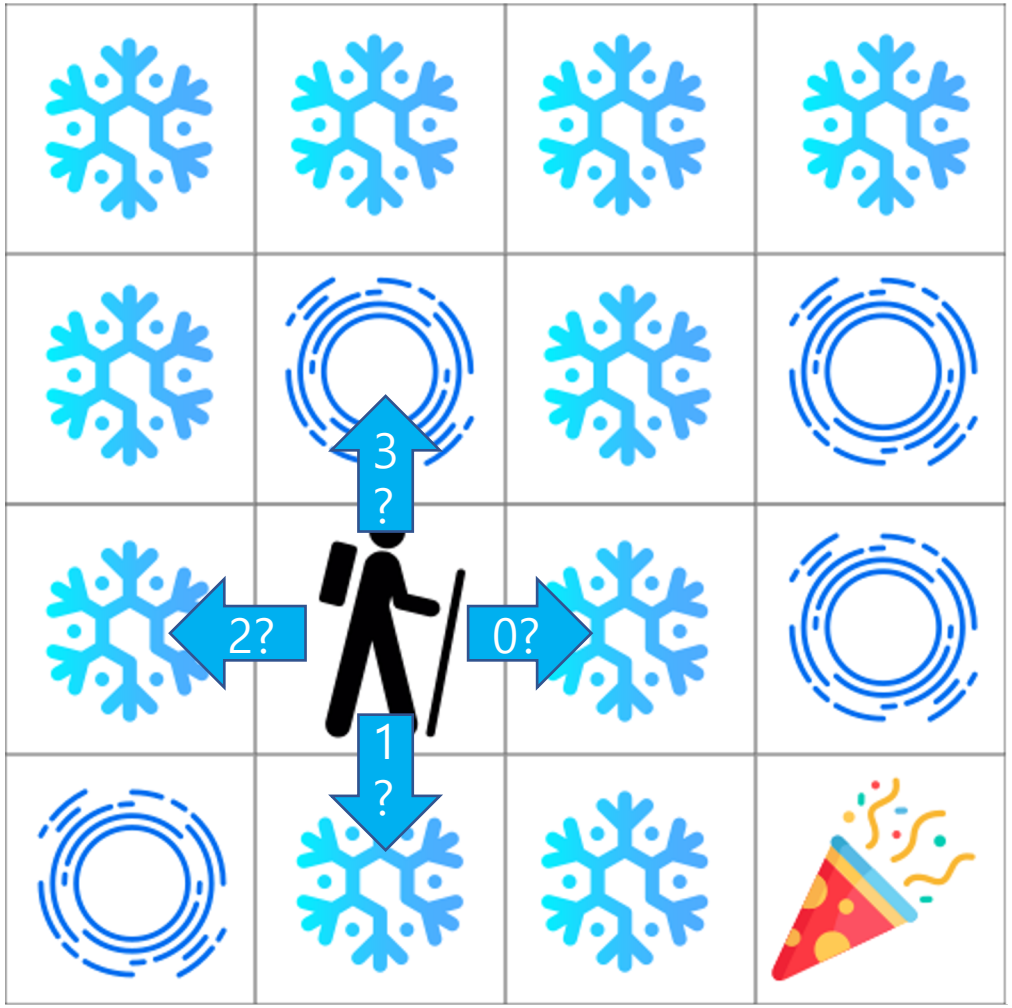
-> Go to Practice! 📝

NON-DETERMMINISTIC

Stochastic Env

$0 < P(s' \mid s, a) < 1$ 인 값이 존재함

단순한 길 찾기 알고리즘으로 해결할 수 없음

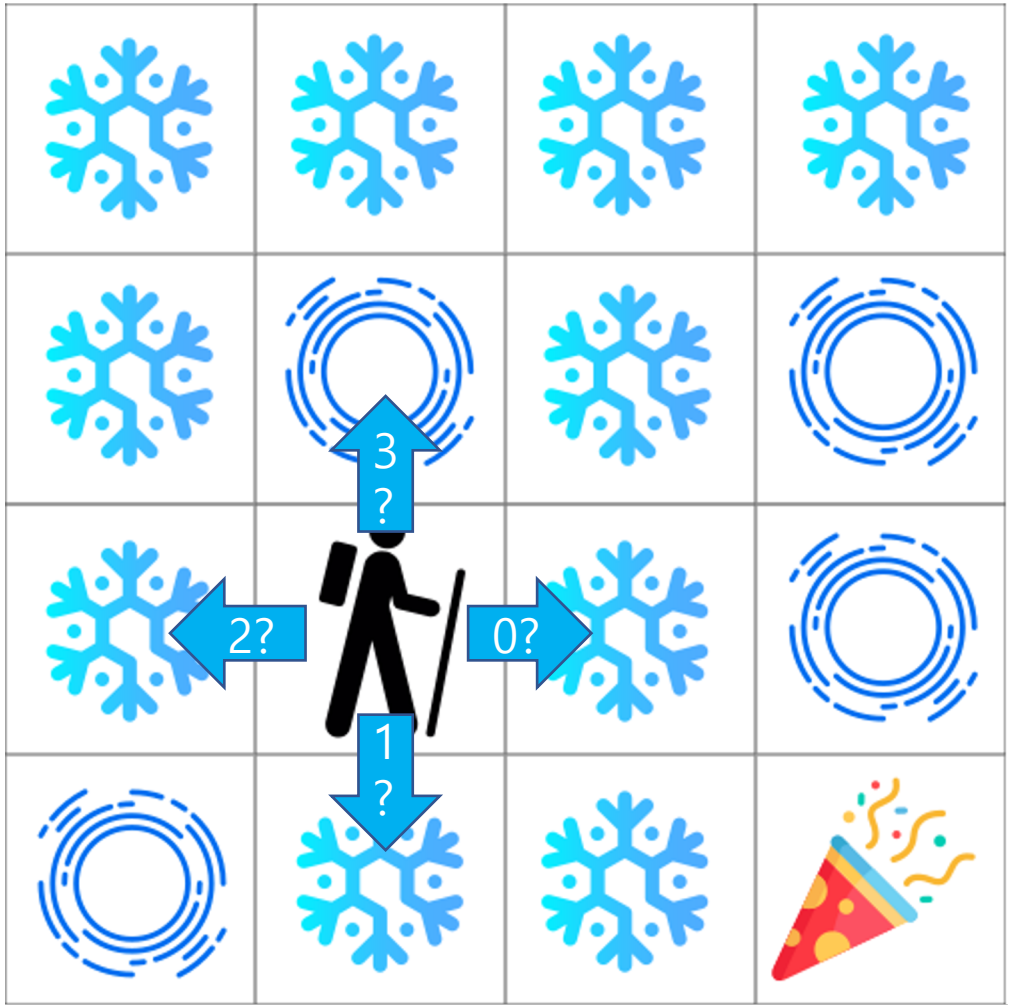


NON-DETERMMINISTIC

Slipper FrozemLake

$0 < P(s' \mid s, a) < 1$ 인 값이 존재함

단순한 길 찾기 알고리즘으로 해결할 수 없음



NON-DETERMMINISTIC

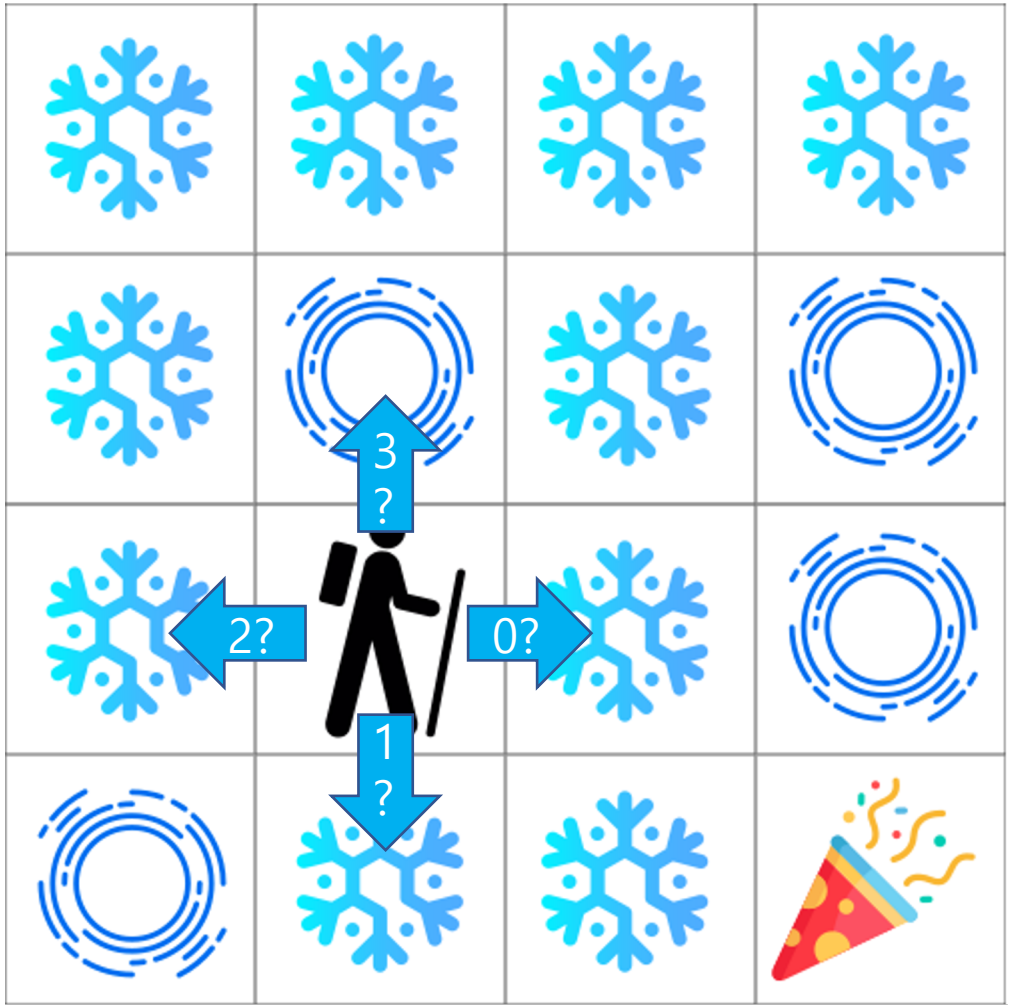
Solution

구멍의 반대 방향으로 이동한다면

적어도 구멍에 빠지지는 않음

처음에 아래로 가는 길이 가장 안전함

q_table을 통해 확인해보자



-> Go to Practice! 📝