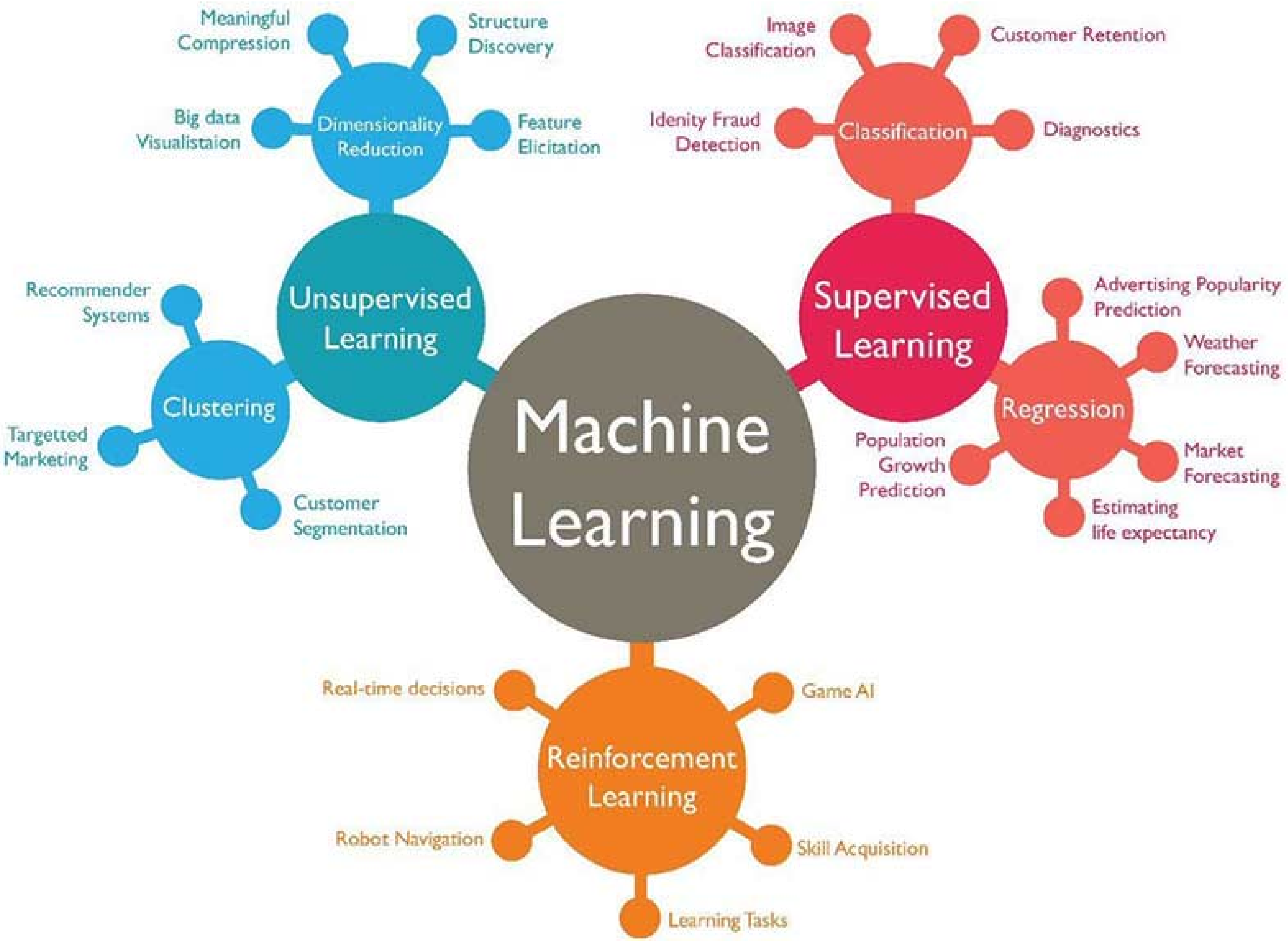
Reinforcement Learning

강 화 학 습

랩 세미나(18.06.01)
민 세 웅

Machine learning



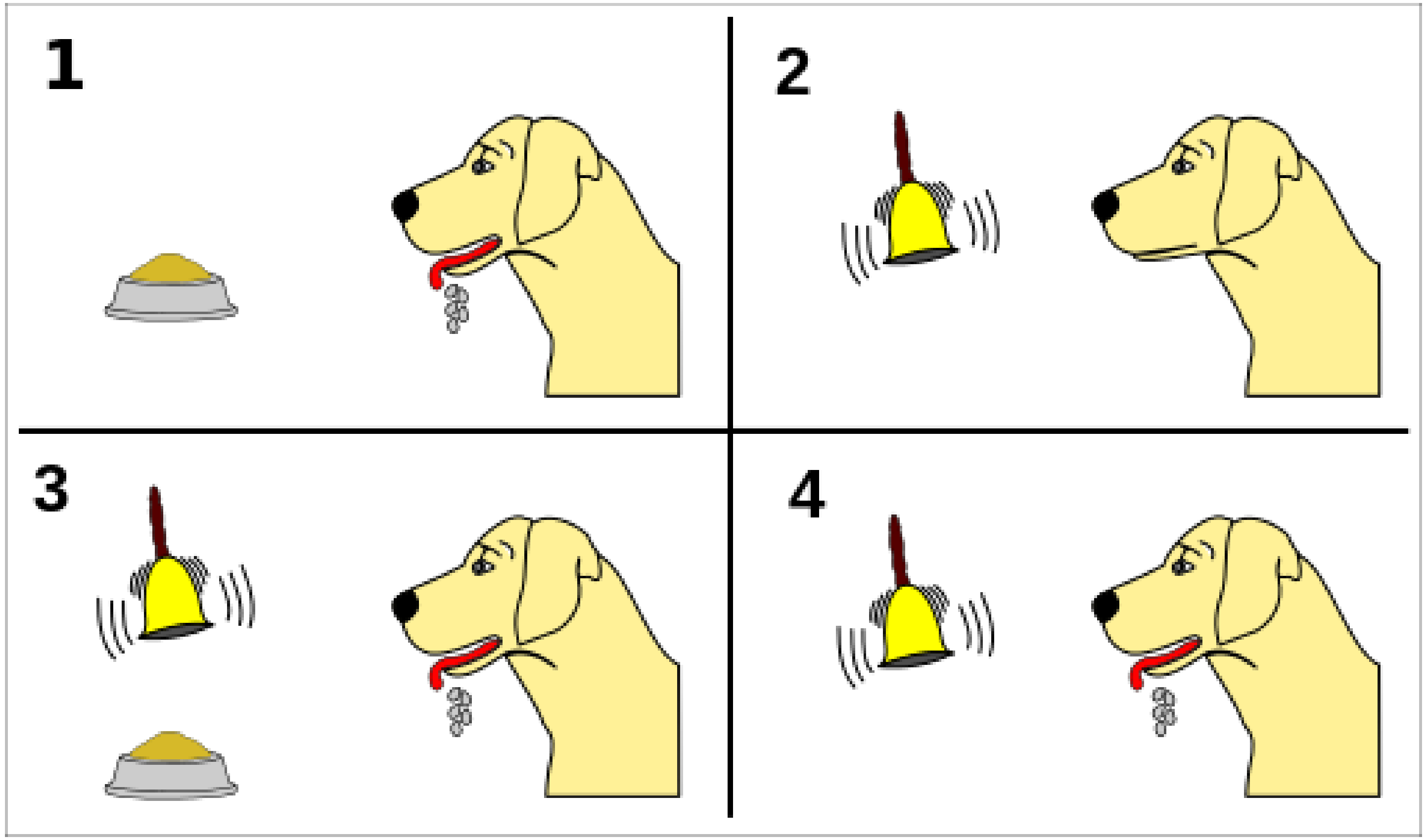
◆ 지도학습
- 예측(Regression)
- 분류(Classificaiton)

◆ 비지도학습
- 차원 감소(Dimension Reduction)
- 분류(Clustering)

◆ 강화학습
- Game
- Robot control

Reinforcement Learning

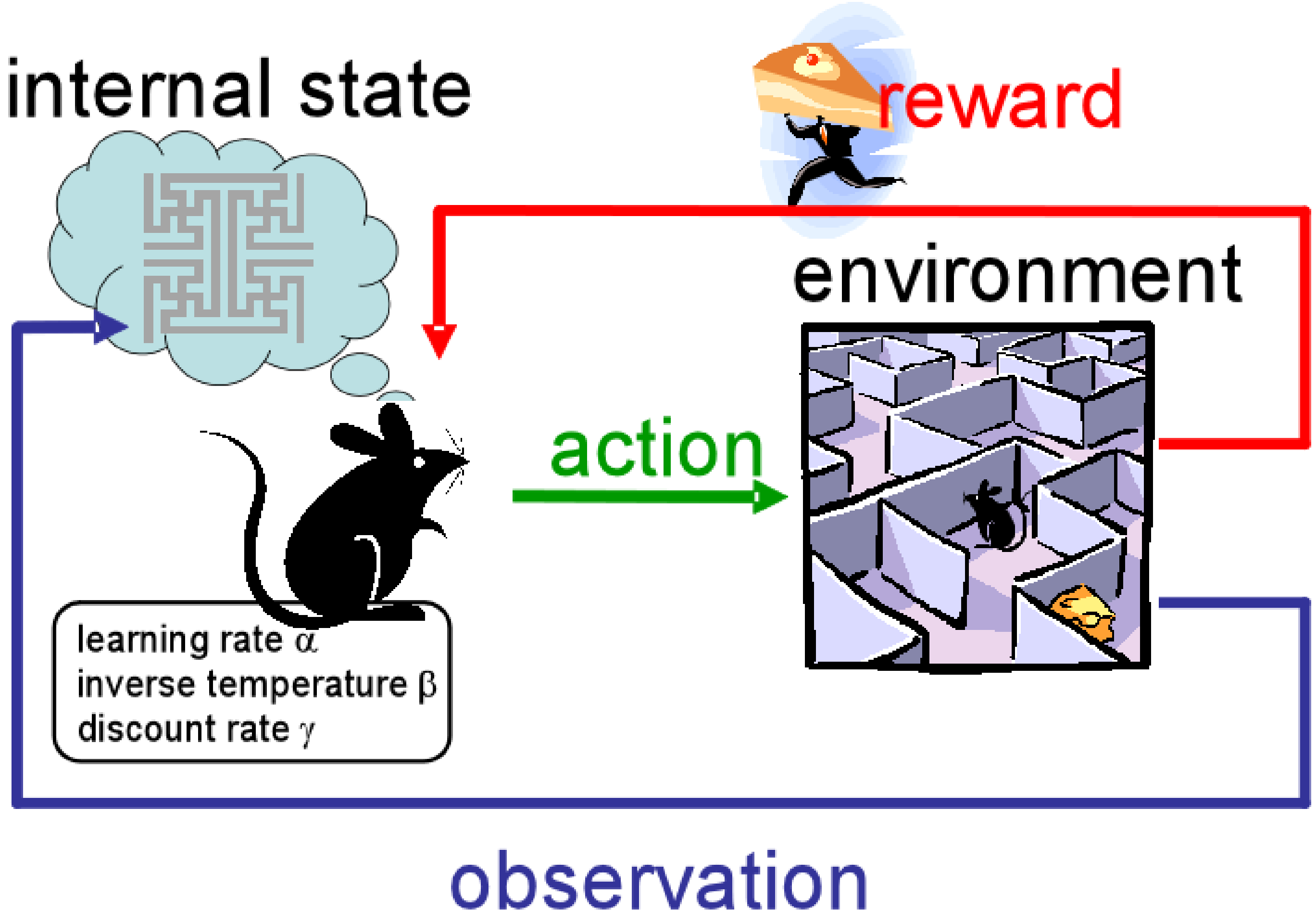
파블로프의 개



◆ 기본적인 컨셉

- 잘한 행동에 대해서 보상을 주자
  (Positive Reinforcement)
- 못한 행동에 대해서 벌을 주자
  (Negative Reinforcement)

Reinforcement Learning

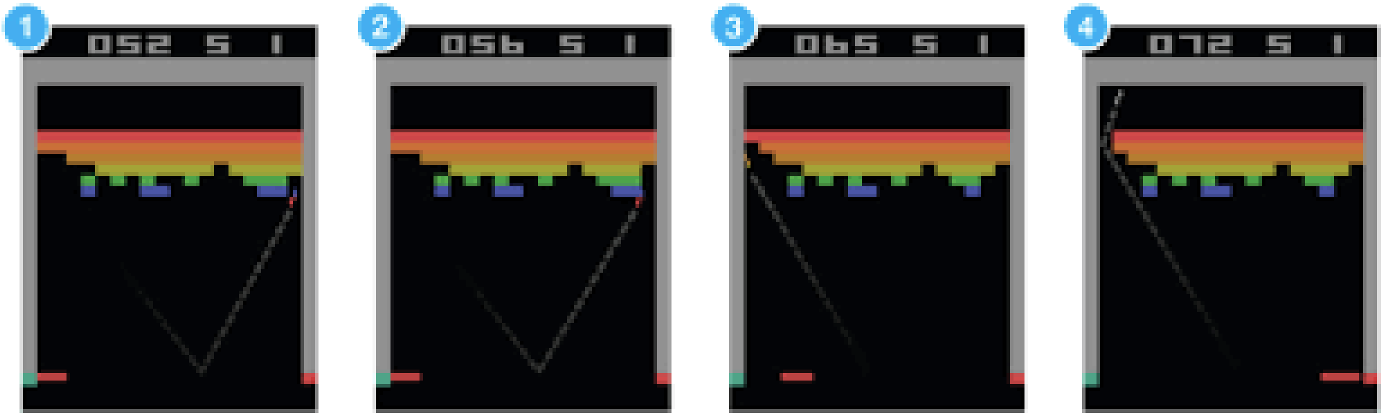


◆ Agent
- 어떤 행동을 하는 개체
- 정책에 따른 액션
- 상태(환경) 감지

◆ Environment
- 행동에 따른 상태 변경
- 리워드 전달

◆ Episode
- 강화학습의 보상을 주는 한 Cycle
  (예) 게임에서 죽기 전까지의 1번의 플레이

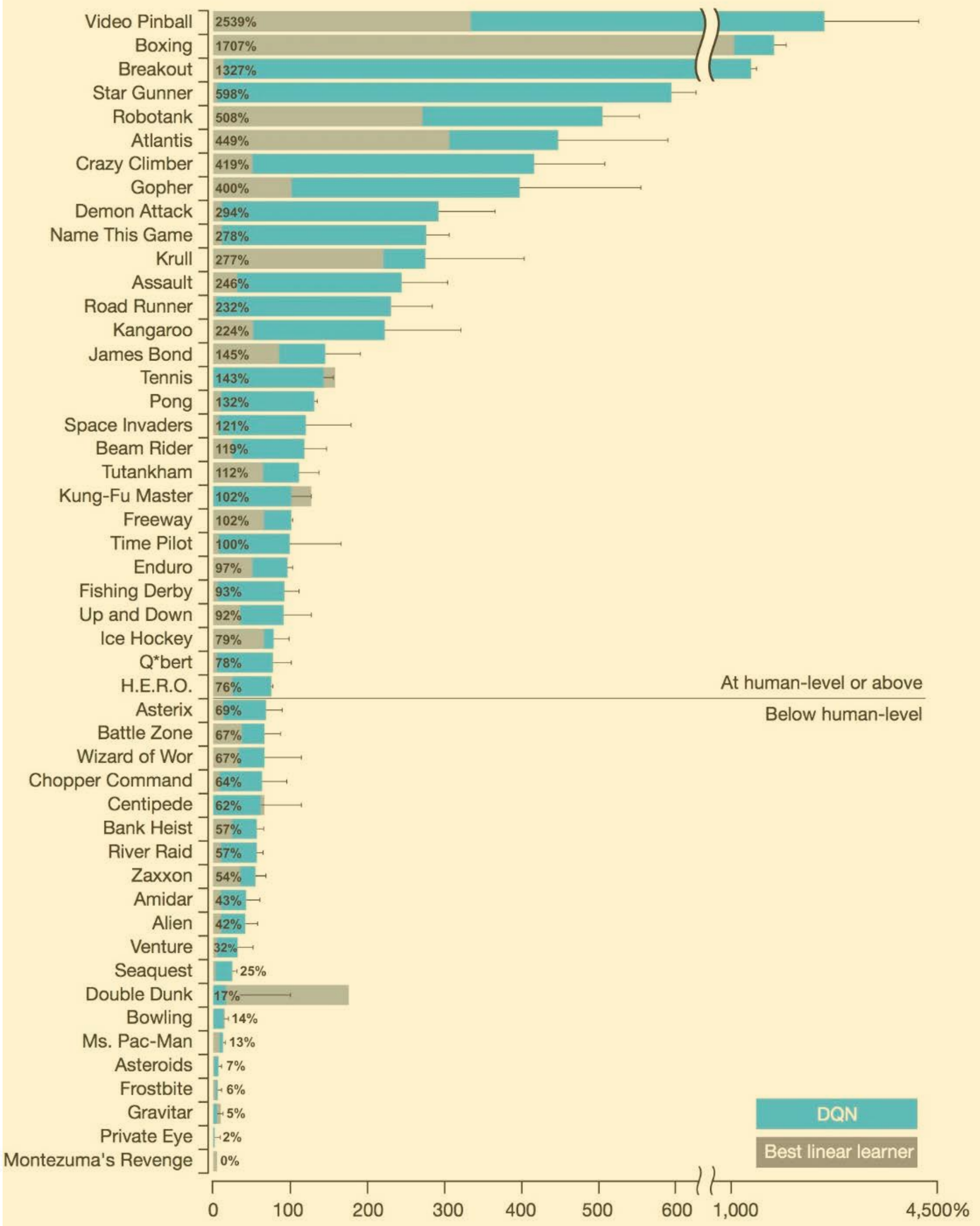
Reinforcement Learning



- 구글 딥마인드(2013, 2015 논문)
  - 기존의 강화학습를 딥러닝과 접목(DQN)
  - CNN과 DQN 접목

- Human-level control through deep reinforcement learning(Nature 26 Feb. 2015)

- 알파고(2016)



Application

Robotics

Autonomous vehicle

Resource allocation

Reinforcement Learning

Finance
Finance

Game Test

Advertisement

- Robotics : Torque at joints
- Business operations
  - Inventory management
  - Resource allocation
- Finance
  - Investment decisions
  - Portfolio design
- E-commerce/media
  - What ads to present to users

Q-learning & Open AI Gym

- Frozen Lake

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Open AI Gym

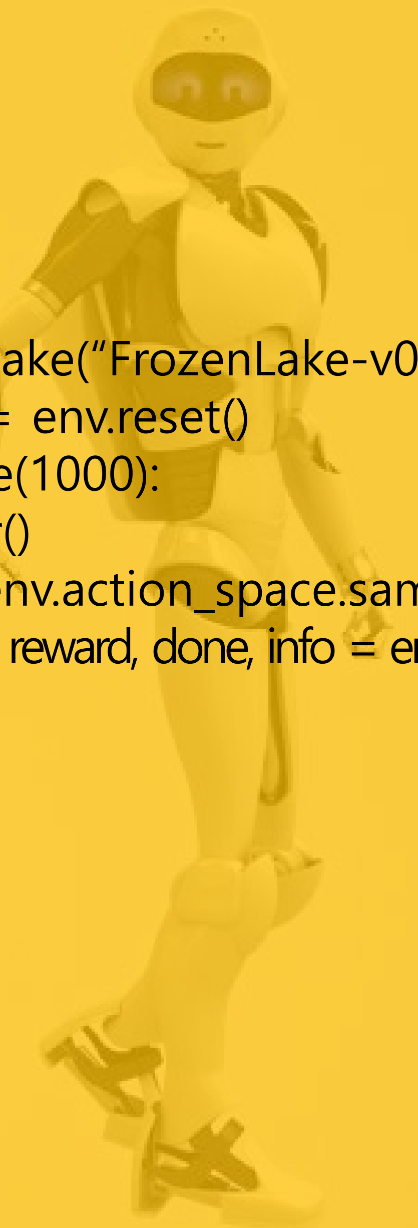
(1) Action (Right, Left, Up, Down)

Agent

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Environment

(2) State, Reward



```
import gym
env = gym.make("FrozenLake-v0")
observation = env.reset()
For _ in range(1000):
    env.render()
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)
```

Open AI Gym

(1) Action : Right

Agent

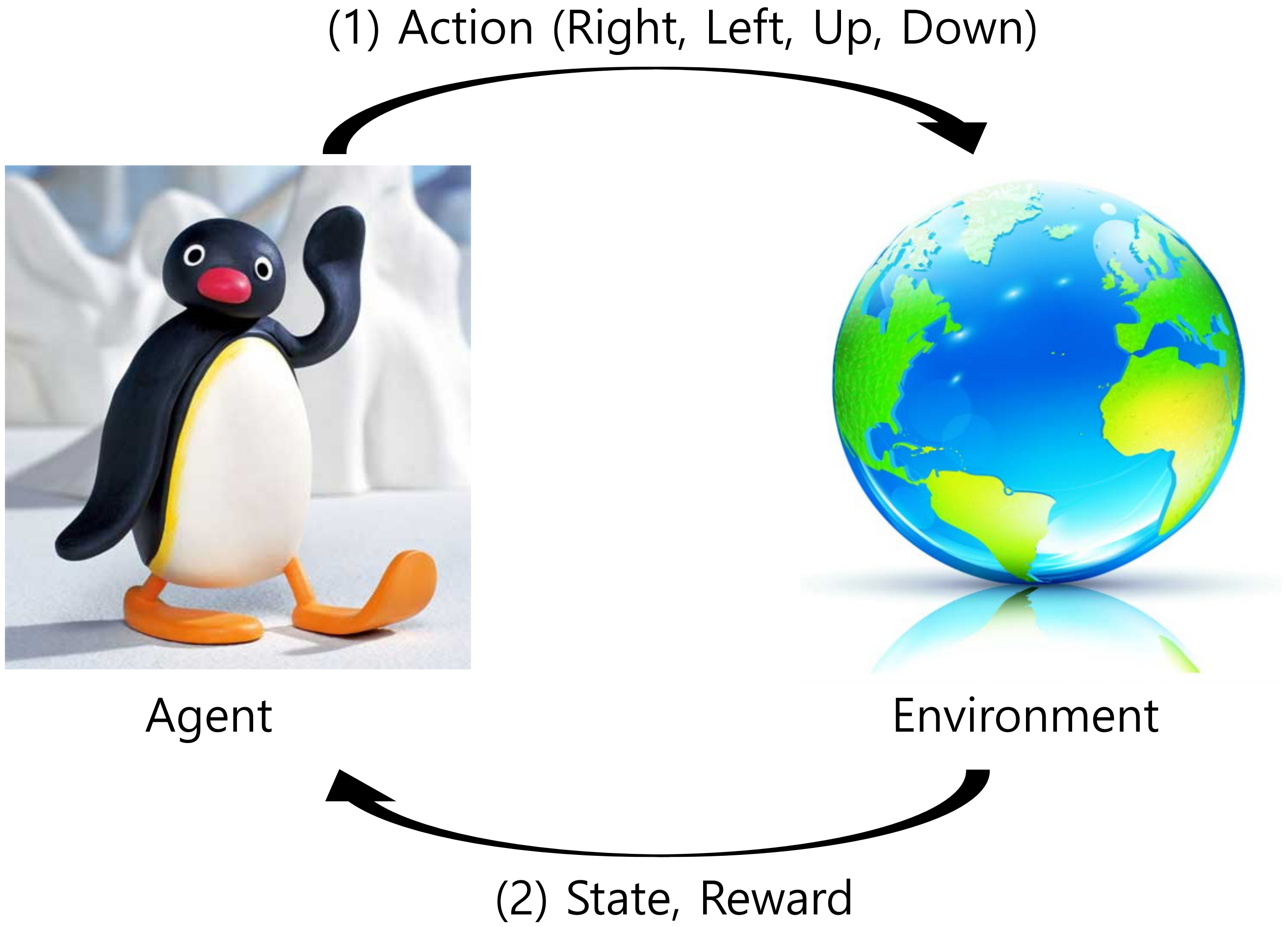
S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Environment

(2) State : 1, Reward : 0

```
import gym
env = gym.make("FrozenLake-v0")
observation = env.reset()
For _ in range(1000):
    env.render()
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)
```

Open AI Gym




```
import gym
env = gym.make("FrozenLake-v0")
observation = env.reset()
For _ in range(1000):
    env.render()
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)
```

Q-learning

Q-function (action-value function)

(1) state

(2) action

(3) quality(reward)

Q(state, action)

Policy using Q-function

State-action value function

Q(s1, LEFT) : 0
Q(s1, RIGHT) : 0.5
Q(s1, UP) : 0
Q(s1, DOWN) : 0.3

$\max_{\{a\}} Q(S1, a)$

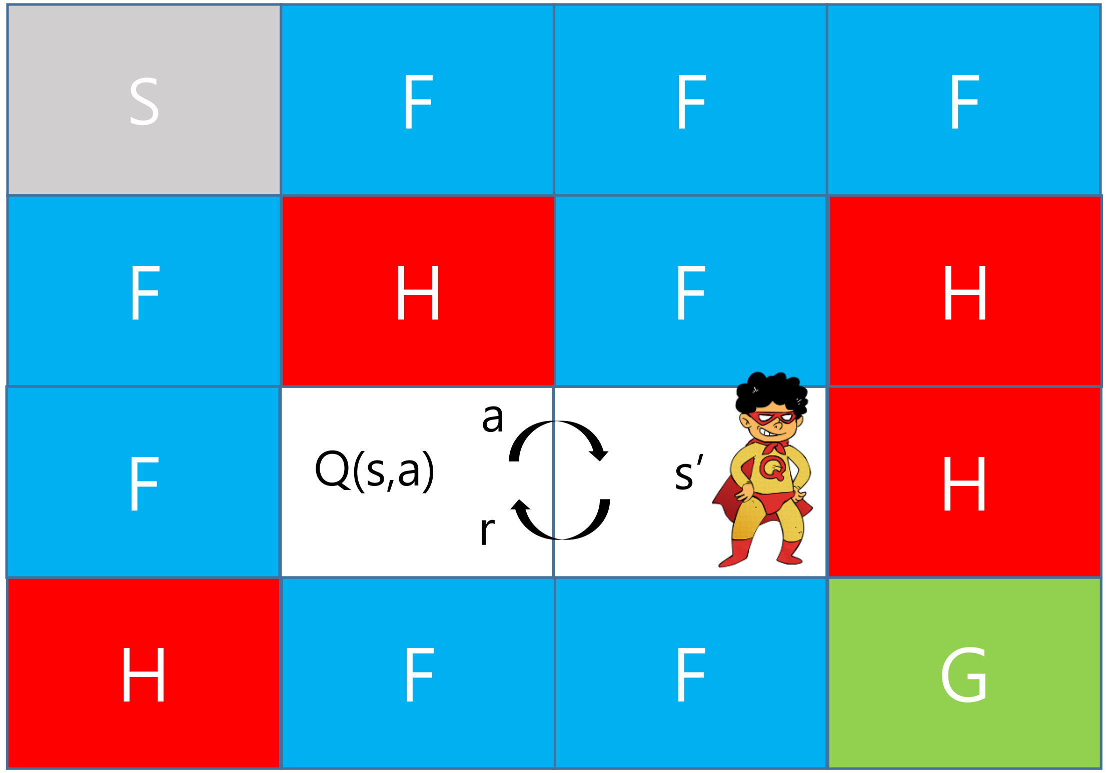
RIGHT <-$\arg\max_{\{a\}} Q(S1, a)$

Optimal Policy with Q

Max Q = $max_{\{a'\}} Q(s, a')$

$\Pi^*(s) = \arg\max_{\{a\}} Q(s, a)$

Q-learning

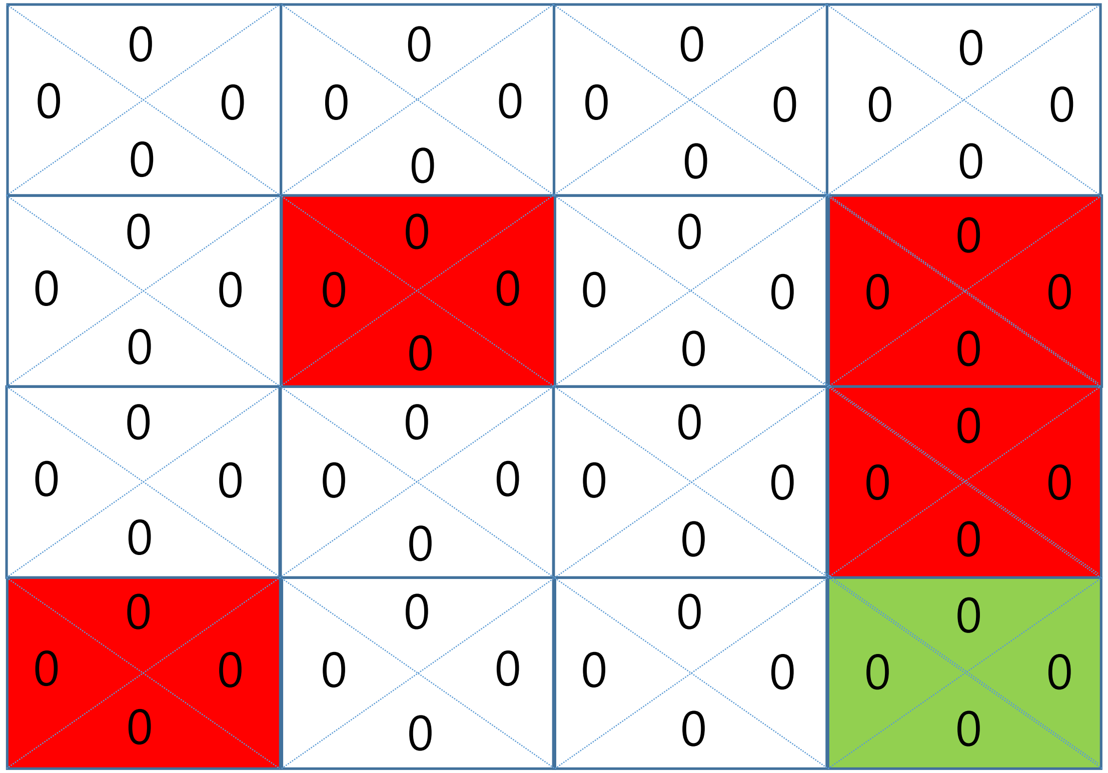


Assume Q in s′ exists!

- My condition
  - I am in **s**
  - When I do action **a,** I'll go to **s'**
  - When I do action **a**, I'll get reward **r**
  - Q in s′, Q(s′, a′) exist!
- $Q(s, a) = r + maxQ(s', a')$

Q-learning

- Learning Q(s,a) : Q-table(16x4)
  16 states and 4 actions (up, down, left, right)



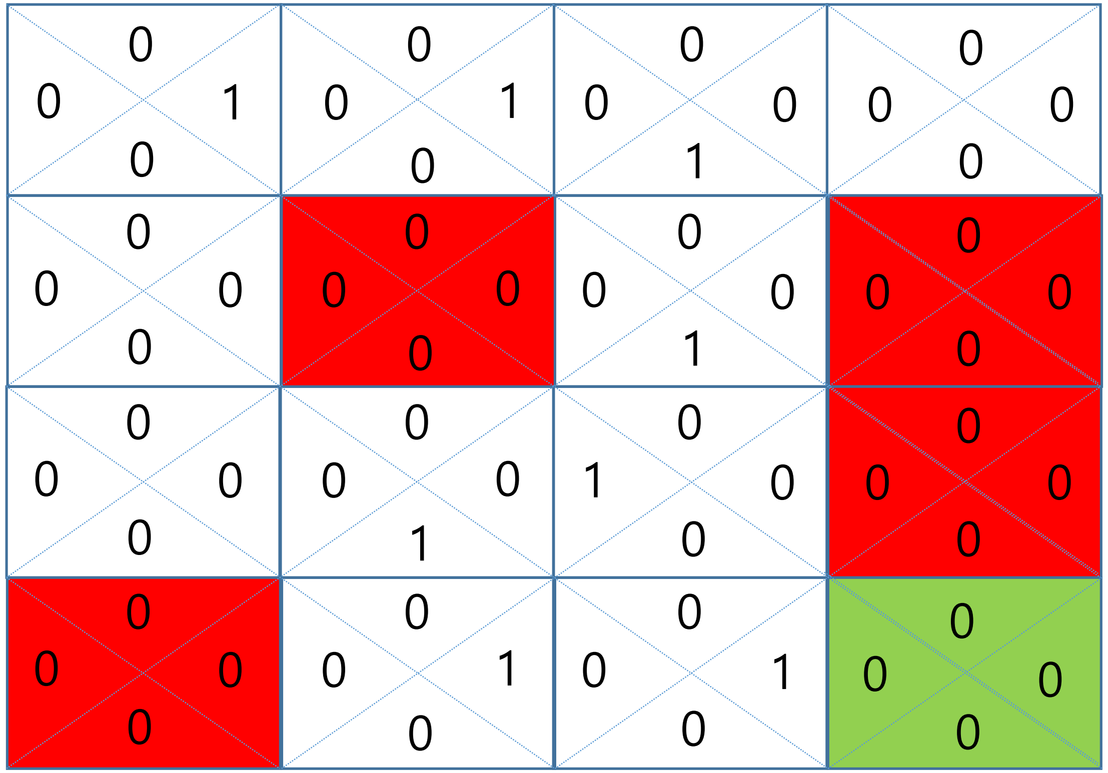
Assume Q in s′ exists!

- My condition
  - I am in **s**
  - When I do action **a,** I'll go to **s′**
  - When I do action **a**, I'll get reward **r**
  - Q in s′, Q(s′, a′) exist!

- Q(s, a) = r + $maxQ(s',a')$

Q-learning

- Learning Q(s,a) : Q-table(16x4)
  16 states and 4 actions (up, down, left, right)



For each s, a initialize table entry $\hat{Q}(s,a)$<-0
Observe current state s
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s,a)$ as follows:

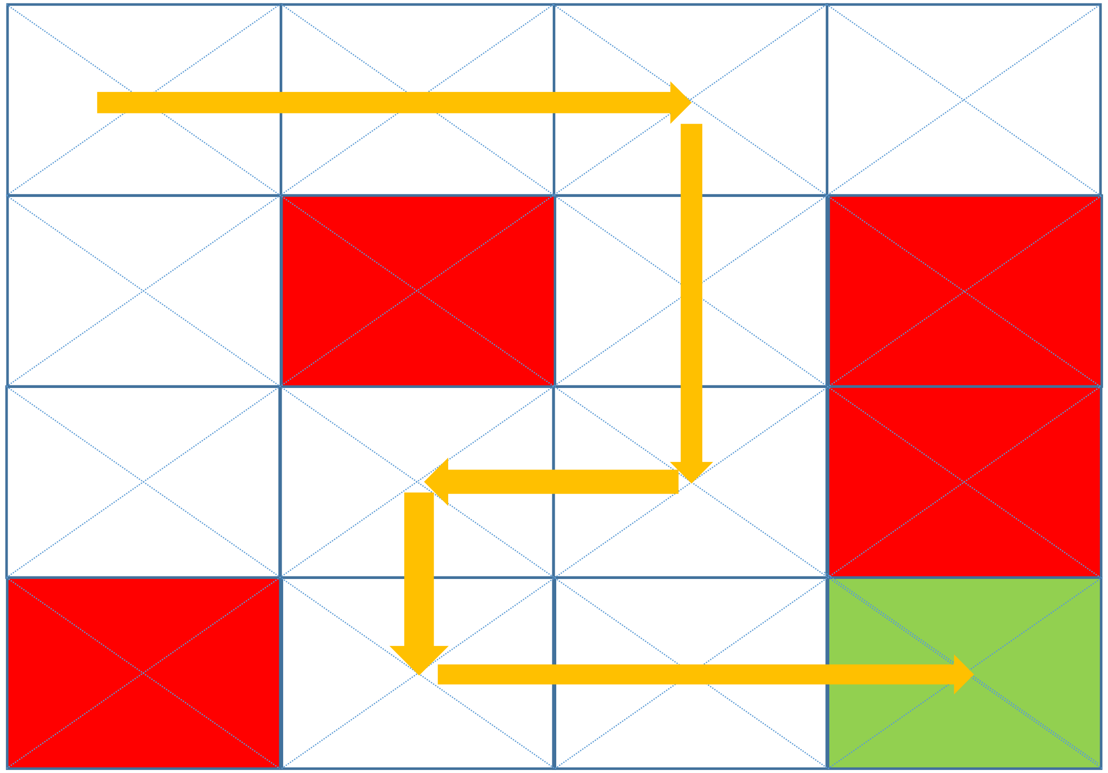
$$\hat{Q}(s,a) \leftarrow r + \max_{\{a'\}} \hat{Q}(s',a')$$

- $s \leftarrow s'$

Exploit VS Exploration: E-greedy

- Learning Q(s,a)

Is it a optimal solution?



E - greedy

```
e = 0.1
if rand < e:
    a = random
else
    a = argmax(Q(s,a))
```

Decaying E - greedy

```
for i in range(1000)
    e = 0.1 / (i+1)
    if rand < e:
        a = random
    else
        a = argmax(Q(s,a))
```

Exploit VS Exploration

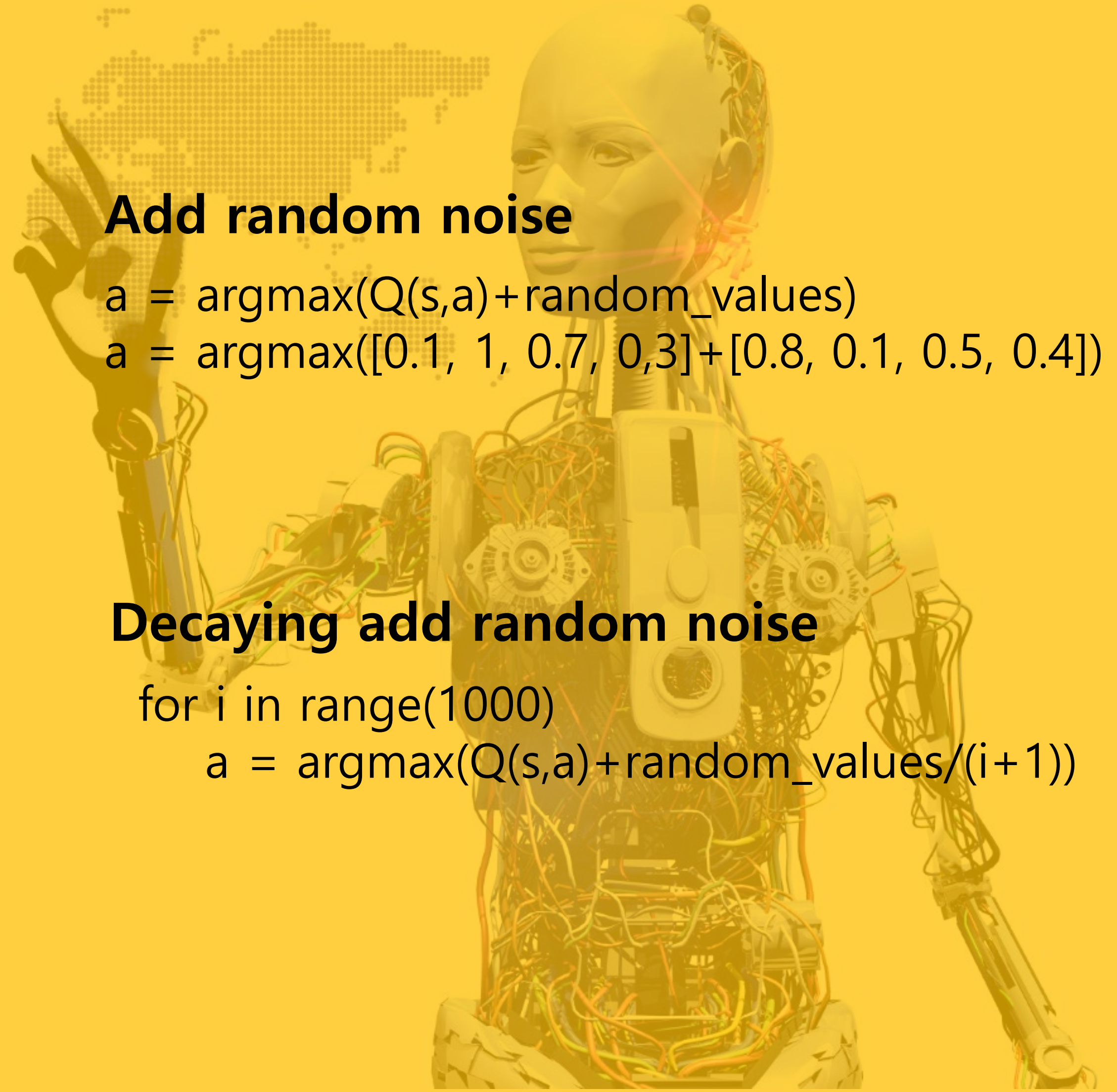
- Add random noise

Up　Q : 0.3 + 0.4

Left　Q : 0.1 + 0.8

Right　Q : 0.7 + 0.5

Down　Q : 1 + 0.1



Add random noise

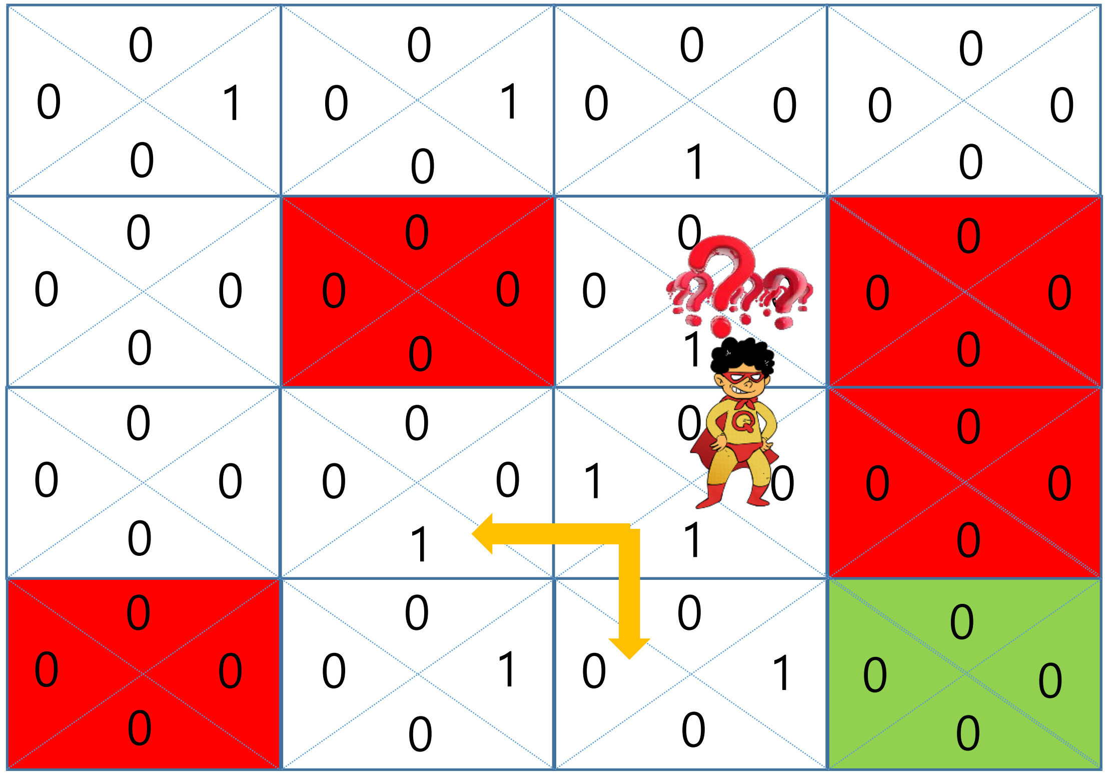
```
a = argmax(Q(s,a)+random_values)
a = argmax([0.1, 1, 0.7, 0,3]+[0.8, 0.1, 0.5, 0.4])
```

Decaying add random noise

```
for i in range(1000)
    a = argmax(Q(s,a)+random_values/(i+1))
```

Q-learning

- Discounted future reward



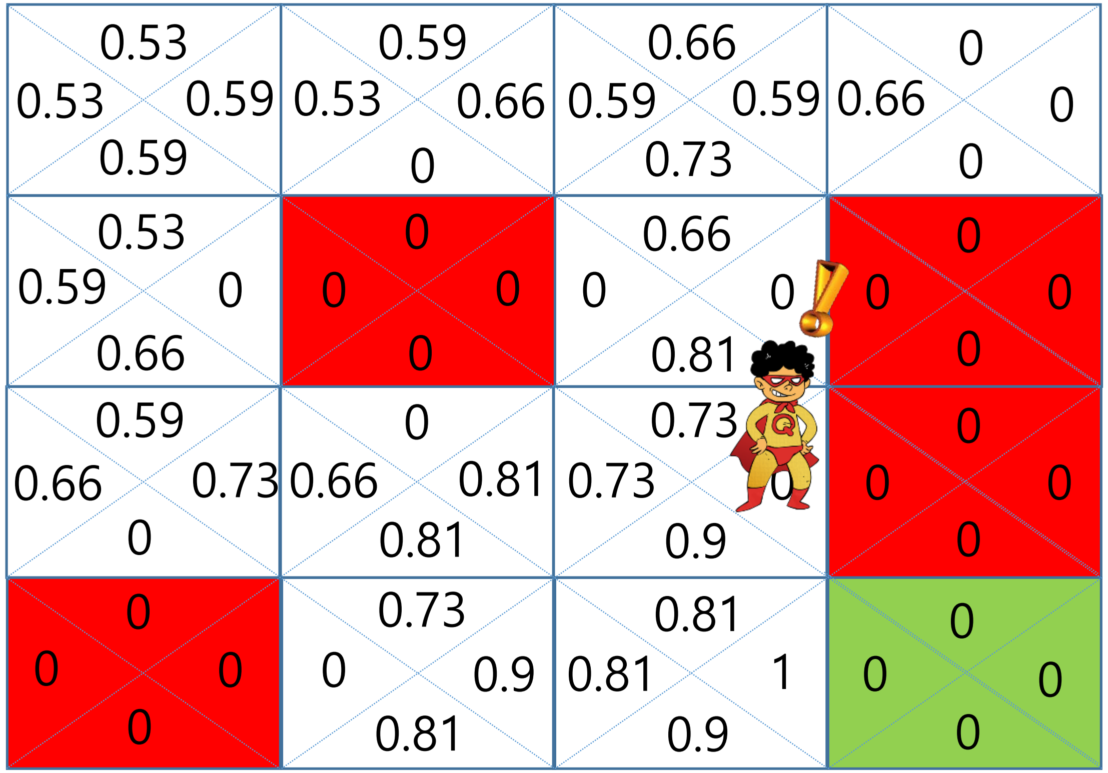
Discounted future reward

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{\{a'\}} \hat{Q}(s',a')$$

보통 $\gamma = 0.9$ 정도 사용

Q-learning

Discounted future reward



Discounted future reward

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{\{a'\}} \hat{Q}(s',a')$$

보통 $\gamma = 0.9$ 정도 사용

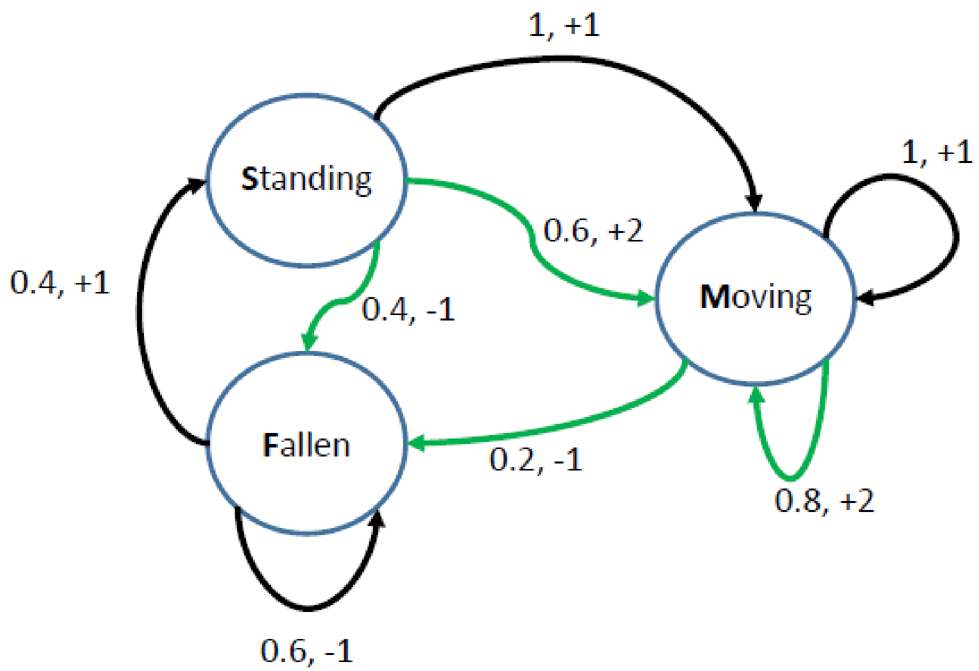
Convergence

$\hat{Q}$ converges to Q

- In deterministic worlds
- In finite states

Stochastic(non-deterministic) world

MDP(Markov Decision Process))



slow action (black)
fast action (green)

Learning incrementally

Learning rate, α

- $\alpha = 0.1$

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{\{a'\}} Q(s',a')]$$

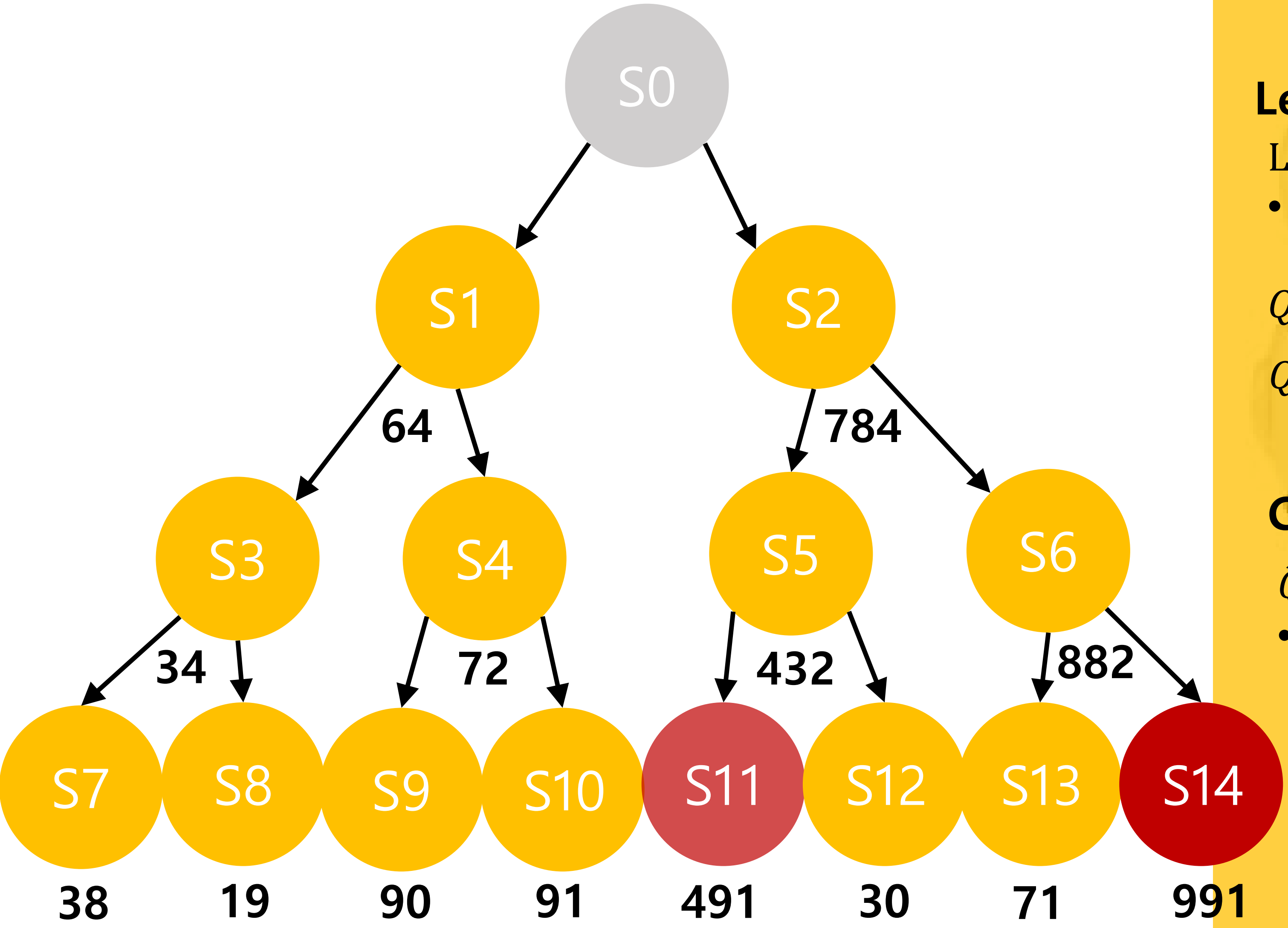
$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{\{a'\}} Q(s',a') - Q(s,a)]$$

Convergence

$\hat{Q}$ converges to Q

- Can still prove convergence of $\hat{Q}$ to Q [Watkins and Dayan, 1992]

EX (State-Value function)



Learning incrementally

Learning rate, α

- $\alpha = 0.1$

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{\{a'\}} Q(s',a')]$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{\{a'\}} Q(s',a') - Q(s,a)]$$

Convergence

$\hat{Q}$ converges to Q

- Can still prove convergence of $\hat{Q}$ to Q [Watkins and Dayan, 1992]