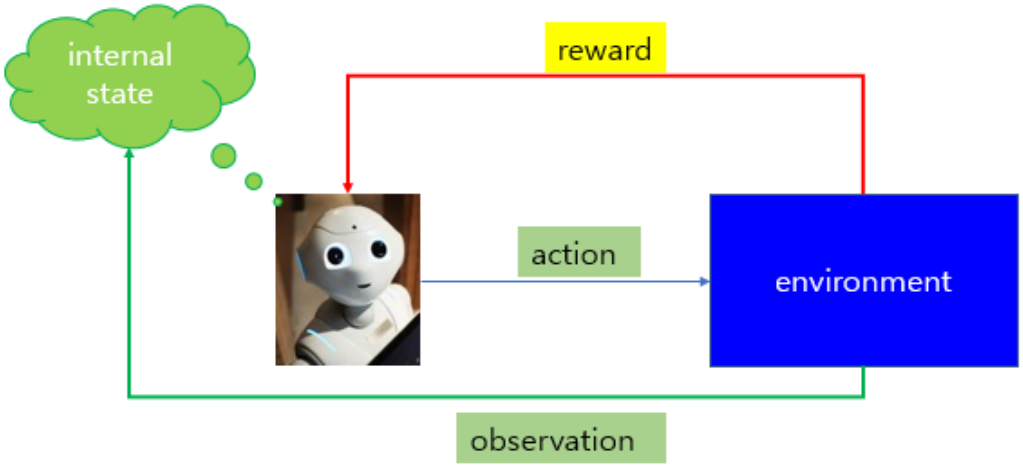
[강화 학습 (Reinforcement Learning)]

환경(세상, Environment)에서 행동하는 액터(Actor)는 환경 속에서의 상태가 변경. 매순간마다 잘했다 못했다는 보상(Reward)를 주어 학습합니다.

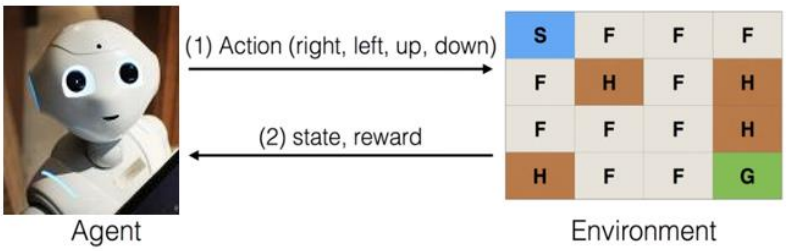


Open AI GYM이라는 FW에 이미 만들어진 환경(Environment)을 사용하여 실습해 보겠습니다.

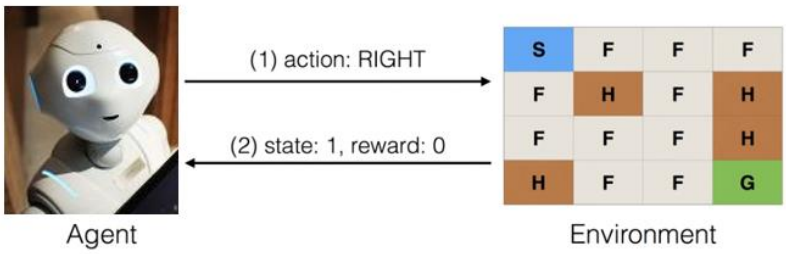
첫번째 환경(Environment) 은 Frozen Lake Game 입니다.
S에서 시작, H는(Hole) 여기에 빠지면 감점. 목표는 G(goal)에 가는 것입니다.
게임의 전체 환경을 볼 수 있다면 다음과 같습니다.

Frozen Lake

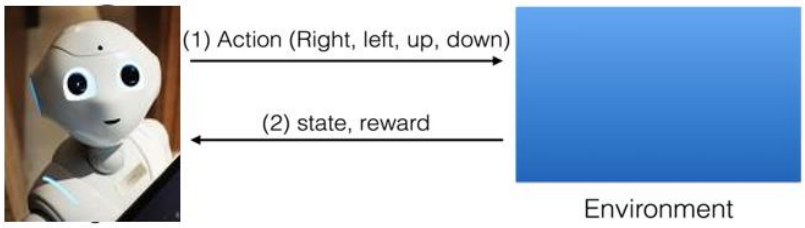
S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G



예) 오른쪽으로 이동 시,



전체 환경을 다 알고 있다면 게임이 쉽겠지만
Agent는 전체 Environment 가 보이지 않습니다.

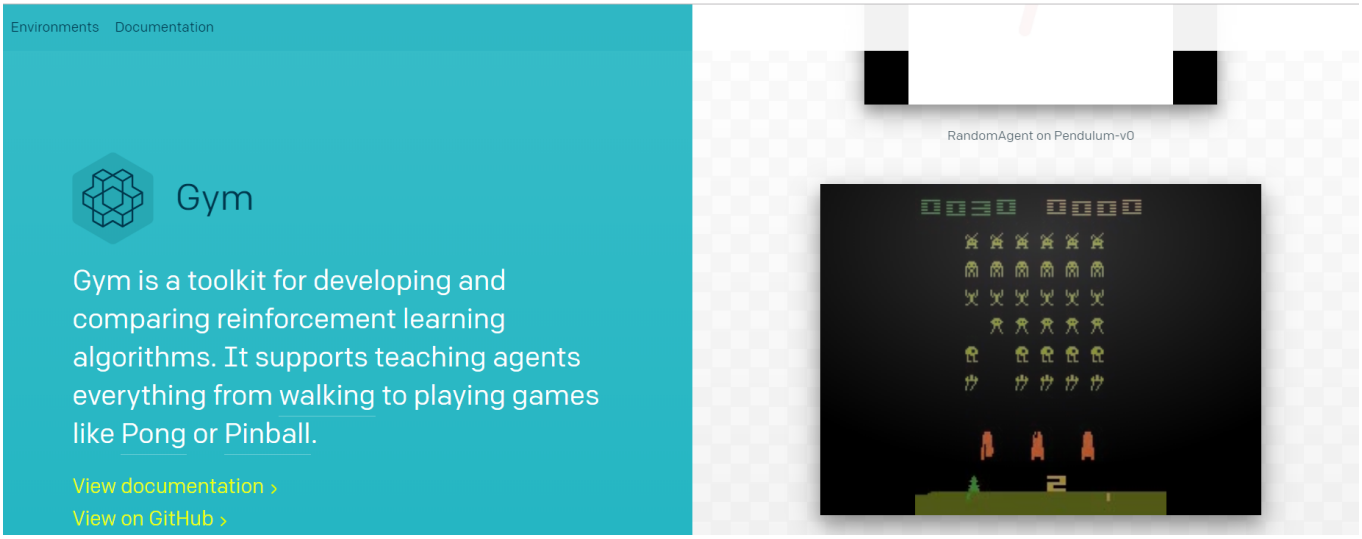


Agent가 행동을 해야 한 단계씩 Environment 정보를 알 수 있습니다.

Open AI GYM

환경을 쉽게 만들 수 있는 **Framework**

https://gym.openai.com/



실습) OPEN AI GYM 설치

아나콘다 콘솔에서 다음을 실행합니다.

```
(base) C:₩Windows₩system32>activate machine_learning_env
(machine_learning_env) C:₩Windows₩system32>pip install gym
```

실습) Playing OpenAI Gym Games

공유 폴더의 ex01_play_frozenlake_det.py 파일이 게임 프로그램 실행 파일입니다.
(참고 - ex01_play_frozenlake_det.py
이 파일을 실행하겠습니다.
키보드 입력을 받기 위한 라이브러리 reachkey도 함께 설치합니다.

아나콘다 콘솔에서 다음을 실행합니다.

```
(base) C:₩..>activate machine_learning_env
(machine_learning_env) C:₩..>pip install readchar
(machine_learning_env) C:₩..>python "C:₩....₩ex01_play_frozenlake_det"
```

화살표를 오른쪽, 왼쪽, 위, 아래로 이동하여 Goal에 도달해봅니다.

```
←[41mS←[0mFFF
FHFH
FFFH
HFFG
  (Right)
S←[41mF←[0mFF
FHFH
FFFH
HFFG
State:  1 Action:  2 Reward:  0.0 Info:  {'prob': 1.0}
  (Right)
SF←[41mF←[0mF
FHFH
FFFH
HFFG
State:  2 Action:  2 Reward:  0.0 Info:  {'prob': 1.0}
```

```
import gym                      # gym 설치 후, import
env = gym.make("Taxi-v1")       # 환경 생성.    환경 이름 : Taxi-v1
observation = env.reset()       # 환경 초기화 (env.reset()) 후 첫 상태(observation)를 가지고 옵니다.

for _ in range(1000):
        env.render()                    # 환경을 출력
        action = env.action_space.sample()
        #  환경에 따라 적절한 action을 선정하는 함수를 앞으로 만들어야 합니다.
        observation, reward, done, info = env.step(action) #환경에 선정된 action을 행동하는 함수.
        # action 후 상태(observation), reward , done(게임이 끝났는지 알려줌), info(추가 정보)
```

[Q-learning (table)]

Random?

Even if you know the way, ask. "아는 길도, 물어가라"

Q			

Q-function

현재 상태에서 취할 수 있는 모든 종류의 action 후 얻을 수 있는 reward의 최대 값과,
그 최대값을 갖을 수 있는 action 얻기 :

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

$\pi^*(s) = \arg\max_a Q(s, a)$

Q (현재상태, 다음 action)

예) Q (s1, LEFT): 0

Q (s1, RIGHT): 0.5

Q (s1, UP): 0

Q (s1, DOWN): 0.3

위 예에서 현재 상태 s1에서 취할 수 이는 모든 action (Left, right, up, down) 중 최대 reward를 얻을 수 있는 action은 right.가 된다.

[Learning Q(s, a)]

Q 학습 시키기

16x4 Table 총 16개 상태(states) and 4 actions (up, down, left, right)

S0	S1	S2	S3
S4	S5	S6	S7
S8	S9	S10	S11
S12	S13	S14	S15

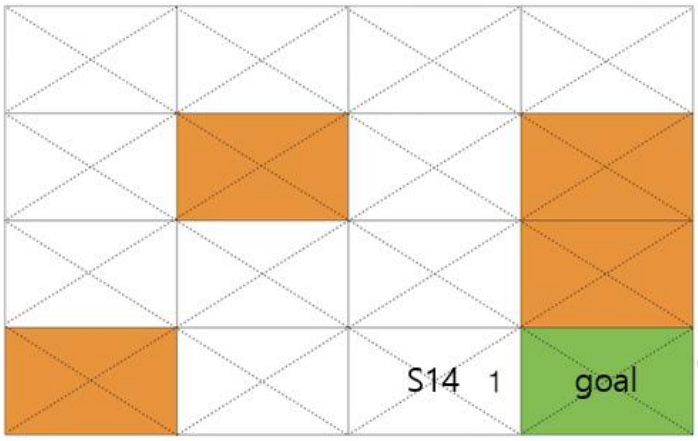
초기 Q table은 모두 0

0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

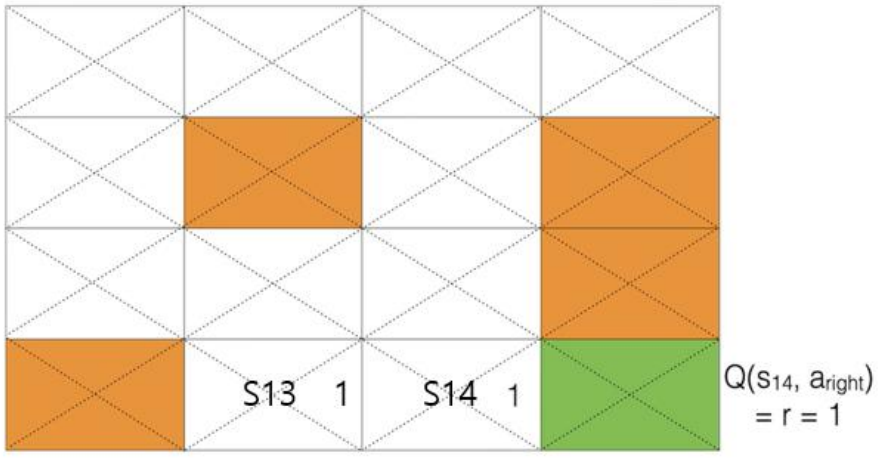
$$\hat{Q}(s,a) \leftarrow r + \max_{a'} \hat{Q}(s',a')$$

Q(s상태에서 a Action 시) ← reward(마지막 Goal에 도달 했을 때만1)
+ 다음 상태S′ 의 여러 action의 reward 중 최대값

(단, r 은 이 게임에서 마지막 Goal에 도달 했을 때만 1 reward를 받을 수 있다.)



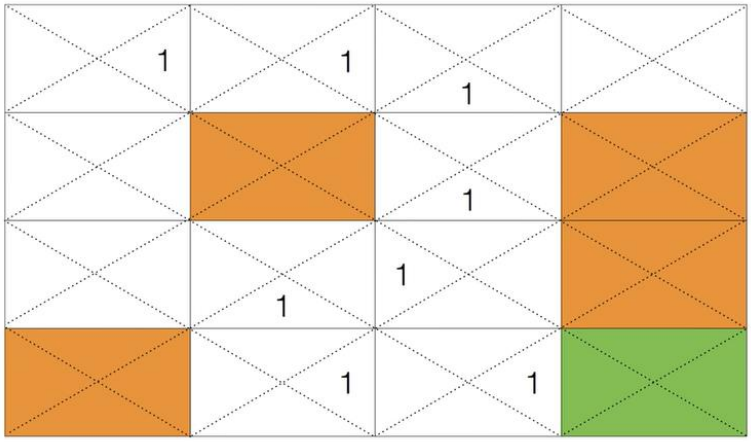
1) 초기 학습에 랜덤하게 움직이다가 우연히, Q(S14)에 오른쪽으로 움직여 Goal에 다다르면 reward 1을 받게 된다.



$Q(s_{13}, a_{right}) = r + \max(Q(s_{14}, a)) = 0 + \max(0, 0, 1, 0) = 1$

2) 랜덤하게 움직이다가 우연히, Q(S13)에 오른쪽으로 움직이면 S14의 max(0,0,1,0) 값인 1을 얻게된다.

3) 학습을 거듭하면 다음과 같은 Q table이 만들어 질 수 있다.



[Q-learning 구현]

최대값이 모두 같을 때 random하게 return하도록 작성한 코드.

```
def rargmax(vector):
    m = np.max(vector)
    indices = np.nonzero(vector == m)[0]
    return random.choice(indices)
```

```
#np.nonzero 함수는 요소들 중 0 이 아닌 값들의 index 들을 반환해 주는 함수
a=np.array([1, 0, 2, 3, 0])
print(np.nonzero(a)) #[0, 2, 3]

#np.nonzero 함수는 요소들 중 3 인값들의 index 들을 반환해 주는 함수
print(np.nonzero(a==3)) #[3]

#np.nonzero 함수는 요소들 중 1 인값들의 index 들을 반환해 주는 함수
print(np.nonzero(a==1)) #[0]
```

'FrozenLake-v3' 환경을 만드는 부분

```
register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name' : '4x4', 'is_slippery': False}
)
env = gym.make('FrozenLake-v3')
```

실습 ex02_0_q_table_frozenlake_det.py

```
import gym
import numpy as np
import matplotlib.pyplot as plt
from gym.envs.registration import register
import random

#최대값이 모두 같을 때 random하게 return하도록 작성한 코드.
def rargmax(vector):
    m = np.max(vector)
    indices = np.nonzero(vector == m)[0]
    return random.choice(indices)

register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name' : '4x4', 'is_slippery': False}
)
env = gym.make('FrozenLake-v3')
```

ex02_0_q_table_frozenlake_det.py (계속)

```
# Q를 모두 0으로 초기화.  Q[16,4]
Q = np.zeros([env.observation_space.n, env.action_space.n])
num_episodes = 2000

# create lists to contain total rewards and steps per episode
rList = []
for i in range(num_episodes): # 여러번 반복 학습
    state = env.reset() # 환경 reset 후, 첫번째 상태 얻음
    rAll = 0
    done = False

    # The Q-Table learning algorithm
    while not done:
        #현재 state의 Q중 최대 reward를 얻을 수 있는 action을 구함.
        action = rargmax(Q[state, :])

        # 환경에서 action 후, new_state와 reward를 얻음
        new_state, reward, done, _ = env.step(action)

        # Update Q-Table with new knowledge using learning rate
        Q[state, action] = reward + np.max(Q[new_state, :])

        rAll += reward
        state = new_state
    rList.append(rAll)


print("Success rate: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print("LEFT DOWN RIGHT UP")
print(Q)

plt.bar(range(len(rList)), rList, color="b", alpha=0.4)
plt.show()
```

(출력결과)

Success rate: **0.952**

Final Q-Table Values

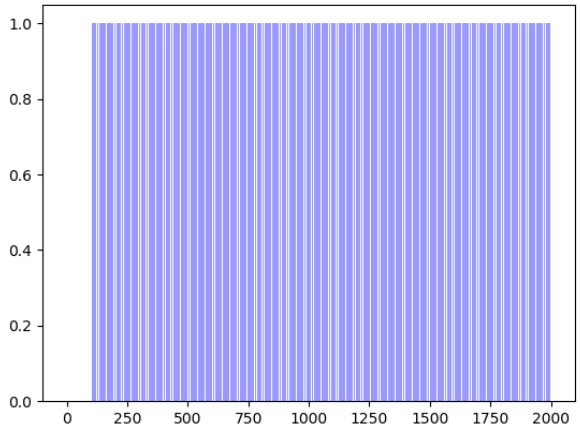
LEFT DOWN RIGHT UP

[[0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]

[0. 0. 0. 0.]

[0. 0. 1. 0.]

[0. 0. 0. 0.]]

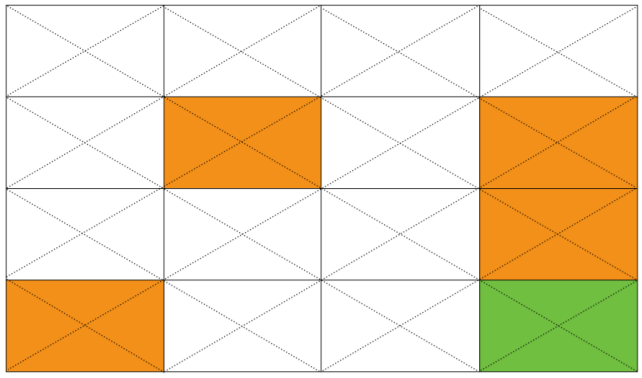


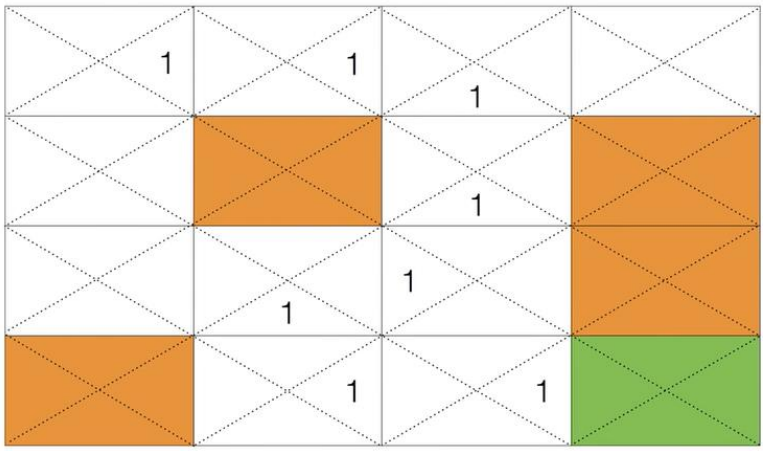
실행 결과 다음과 같은 Q table이 만들어졌습니다.

```
Q = np.zeros([env.observation_space.n, env.action_space.n])
```

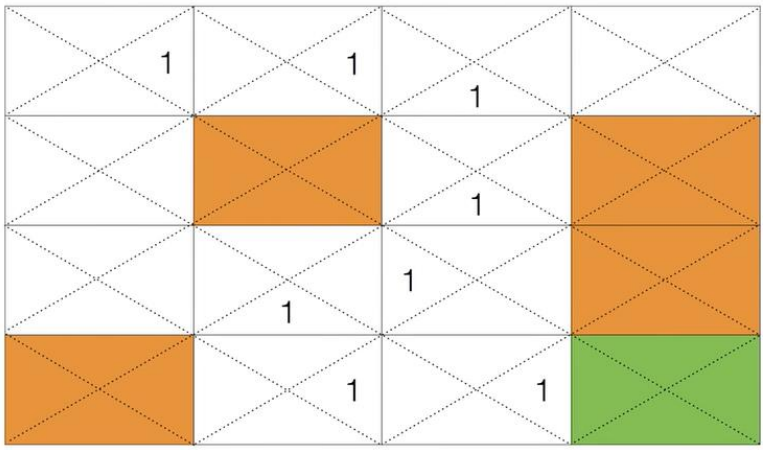
```
print(Q)
```

```
LEFT DOWN RIGHT UP
[[ 0.  0.  1.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  0.]]
```





[매번 알고 있는 같은 길로만 가야 할까? VS 모험]



1) E-greedy

#10% 확률로 random, 90%는 아는 길로

e = 0.1
if (0~1 난수발생) < e :
    a = random 하게 선택
else:
    a = argmax(Q(s, a)) #아는 길로

2) decaying E-greedy

초반에는 random. 뒤로 갈수록 아는 길로.

for i in range (1000)
    e = 0.1 / **(i+1)**
    if (0~1 난수발생) < e:
        a = random 하게 선택
    else:
        a = argmax(Q(s, a)) #아는 길로

실습 ex02_1_q_table_frozenlake_det.py

```
import gym
import numpy as np
import matplotlib.pyplot as plt
from gym.envs.registration import register
import random

#최대값이 모두 같을 때 random하게 return하도록 작성한 코드.
def rargmax(vector):
    m = np.max(vector)
    indices = np.nonzero(vector == m)[0]
    return random.choice(indices)

register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name' : '4x4', 'is_slippery': False}
)

env = gym.make('FrozenLake-v3')


# Q를 모두 0으로 초기화.  Q[16,4]
Q = np.zeros([env.observation_space.n, env.action_space.n])

# Set learning parameters
num_episodes = 2000

# create lists to contain total rewards and steps per episode
rList = []
for i in range(num_episodes):  # 여러번 반복 학습
    state = env.reset()    # 환경 reset 후, 첫번째 상태 얻음
    rAll = 0
    done = False

    e = 1. / ((i // 100) + 1)

    # The Q-Table learning algorithm
    while not done:
        # Choose an action by e-greedy
        # 현재 state의 Q중 최대 reward를 얻을 수 있는 action을 구함.
        # 단, 알려진 길로만 가지 않기 위해서 random 값이 e보다 적은 경우는 아무렇게나 action
        # 학습 후반부로 갈 수록 e의 값은 작아져, 정해질 길로 가게 됩니다.
        if np.random.rand(1) < e:
            action = env.action_space.sample()
        else:
            action = rargmax(Q[state, :])

        # Get new state and reward from environment
        new_state, reward, done, _ = env.step(action)
```

```
        # Update Q-Table with new knowledge using decay rate
        Q[state, action] = reward + np.max(Q[new_state, :])

        rAll += reward
        state = new_state
    rList.append(rAll)

print("Success rate: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print("LEFT DOWN RIGHT UP")
print(Q)
plt.bar(range(len(rList)), rList, color='b', alpha=0.4)
plt.show()
```

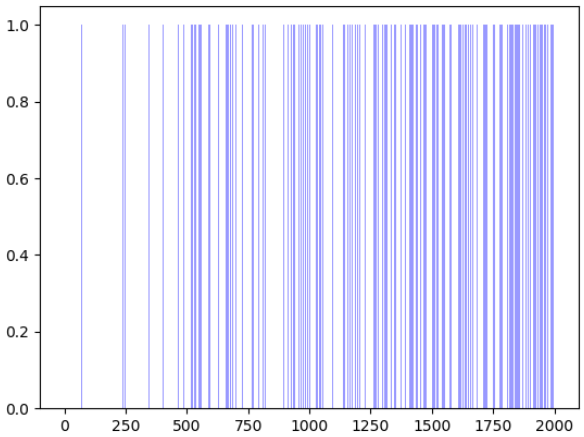
(출력결과)

Success rate: 0.8025

Final Q-Table Values

LEFT DOWN RIGHT UP

[[1. 1. 1. 1.]
 [1. 0. 1. 1.]
 [1. 1. 1. 1.]
 [1. 0. 1. 1.]
 [1. 1. 0. 1.]
 [0. 0. 0. 0.]
 [0. 1. 0. 1.]
 [0. 0. 0. 0.]
 [1. 0. 1. 1.]
 [1. 1. 1. 0.]
 [1. 1. 0. 1.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 1. 1. 1.]
 [1. 1. 1. 1.]
 [0. 0. 0. 0.]]



3) add random noise

a = argmax(Q(s, a) + random_values)

a = argmax([0.5 0.6 0.3 0.2 0.5]+[0.1 0.2 0.7 0.3 0.1])

4) 여기에 **decaying** 적용할 수 있습니다. 그래서 학습 후반으로 갈수록 random 값의 영향이 적어집니다.

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

실습 ex02_2_q_table_frozenlake_det.py

```python
import gym
import numpy as np
import matplotlib.pyplot as plt
from gym.envs.registration import register

register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name' : '4x4', 'is_slippery': False}
)
env = gym.make('FrozenLake-v3')

# Q를 모두 0으로 초기화.  Q[16,4]
Q = np.zeros([env.observation_space.n, env.action_space.n])

#  Learning parameters
num_episodes = 2000

# create lists to contain total rewards and steps per episode
rList = []
for i in range(num_episodes): # 여러번 반복 학습
    state = env.reset()   # 환경 reset 후, 첫번째 상태 얻음
    rAll = 0
    done = False

    # The Q-Table learning algorithm
    while not done:
        #현재 state의 Q중 최대 reward를 얻을 수 있는 action을 구함.
        #단, 알려진 길로만 가지 않기 위해서 random 값 add.
        # 학습 후반 부로 갈 수로 random 값의 영향을 적게 하기위해   random/(i+1)
        action = np.argmax(Q[state, :] + np.random.randn(1, env.action_space.n) / (i + 1))

         # 환경에서 action 후, new_state와 reward를 얻음
        new_state, reward, done, _ = env.step(action)
```

```
        # Update Q-Table with new knowledge using decay rate
        Q[state, action] = reward + np.max(Q[new_state, :])

        rAll += reward
        state = new_state
    rList.append(rAll)

print("Success rate: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print("LEFT DOWN RIGHT UP")
print(Q)
plt.bar(range(len(rList)), rList, color='b', alpha=0.4)
plt.show()
```

(출력결과) Success rate: 0.99

Final Q-Table Values

LEFT DOWN RIGHT UP

[[0. 1. 0. 0.]

[0. 0. 0. 0.]

[0. 0. 0. 0.]

.. 중략

[0. 0. 1. 0.]

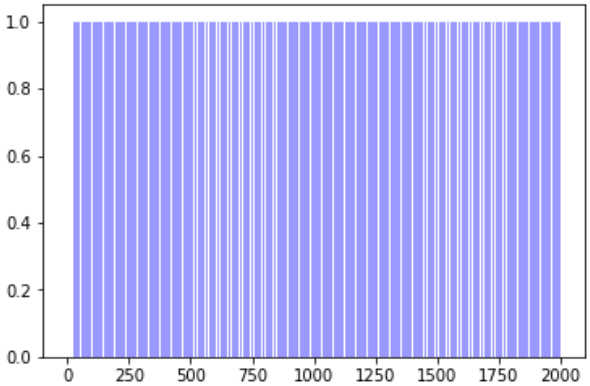
[0. 1. 0. 0.]

[0. 0. 0. 0.]

[0. 0. 0. 0.]

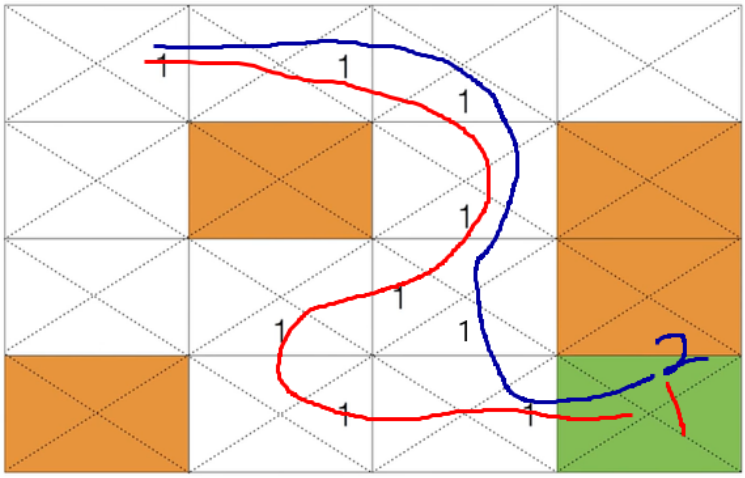
[0. 0. 0. 0.]

[0. 0. 1. 0.]



[0. 0. 0. 0.]]

[1번길이 좋을까요? 2번길이 좋을까요?]



1) Learning Q (s, a) with discounted reward

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

r : 바로 얻을 수 있는 reward

max ... : 미래의 reward

아이디어 :

a) 미래의 reward는 discount 하자

b) 결과적으로 reward를 빨리 받을 수 있는 곳으로 이동하게 함.

- Future reward $R = r_1 + r_2 + r_3 + \cdots + r_n$

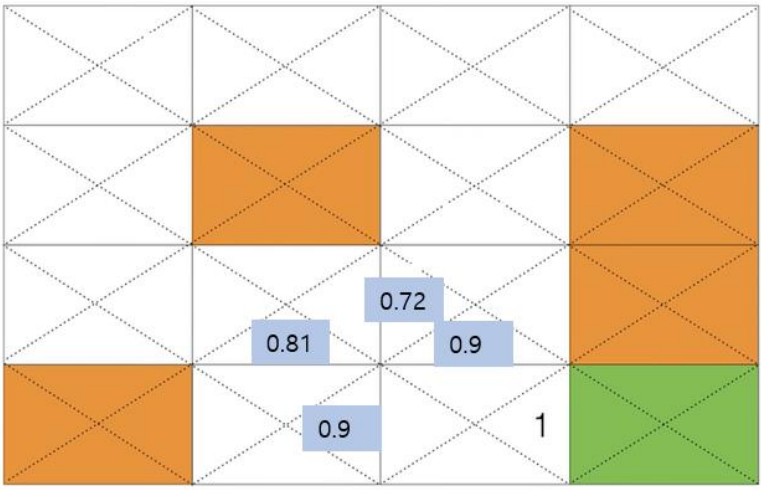
$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$$

- Discounted future reward (environment is stochastic)

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \\ &= r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \cdots)) \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

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

Discounted reward = 0.9



$\hat{Q}$ denote learner's current approximation to Q.

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

$\hat{Q}$ converges to Q.

- In deterministic worlds
- In finite states

실습 ex03_0_q_table_frozenlake.py

```
import gym
import numpy as np
import matplotlib.pyplot as plt
from gym.envs.registration import register

register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name' : '4x4', 'is_slippery': False}
)
env = gym.make('FrozenLake-v3')

# Initialize table with all zeros
Q = np.zeros([env.observation_space.n, env.action_space.n])

# Set learning parameters
dis = .99
num_episodes = 2000
```

```
# create lists to contain total rewards and steps per episode
rList = []
for i in range(num_episodes):
    # Reset environment and get first new observation
    state = env.reset()
    rAll = 0
    done = False

    # The Q-Table learning algorithm
    while not done:
        action = np.argmax(Q[state, :] + np.random.randn(1, env.action_space.n) / (i + 1))

        # Get new state and reward from environment
        new_state, reward, done, _ = env.step(action)

        # Update Q-Table with new knowledge using learning rate
        Q[state, action] = reward + dis * np.max(Q[new_state, :])
        state = new_state

        rAll += reward

    rList.append(rAll)

print("Success rate: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print("LEFT DOWN RIGHT UP")
print(Q)
plt.bar(range(len(rList)), rList, color='b', alpha=0.4)
plt.show()
```

```
        # Get new state and reward from environment
        new_state, reward, done, _ = env.step(action)

        # Update Q-Table with new knowledge using learning rate
        Q[state, action] = reward + dis * np.max(Q[new_state, :])
        state = new_state

        rAll += reward

    rList.append(rAll)

print("Success rate: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print("LEFT DOWN RIGHT UP")
print(Q)
plt.bar(range(len(rList)), rList, color="blue")
#plt.bar(range(len(rList)), rList, color='b', alpha=0.4)
plt.show()
```

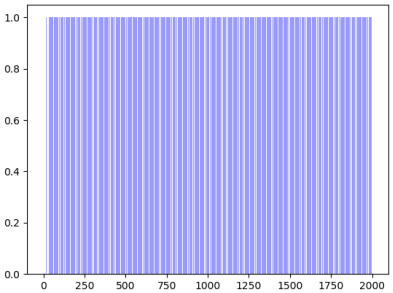
(출력 결과)

Success rate: 0.9525

Final Q-Table Values

LEFT DOWN RIGHT UP

```
[[0.         0.         0.95099005 0.        ]
 [0.         0.         0.96059601 0.        ]
 [0.         0.970299   0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.9801     0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.99       0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.         0.         0.99       0.        ]
 [0.         0.         1.         0.        ]
 [0.         0.         0.         0.        ]]
```



Windy Frozen Lake Nondeterministic world!

Deterministic VS Stochastic (nondeterministic)

- In deterministic models the output of the model is fully
  determined by the parameter values and the initial conditions initialconditions
- Stochastic models possess some inherent randomness.
  The same set of parameter values and initial conditions will lead to an ensemble of different outputs.

Deterministic 게임

```
register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name' : '4x4', 'is_slippery': False}
)
```

Stochastic (nondeterministic) 게임

```
register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name' : '4x4', 'is_slippery': True}
)
```

Stochastic (non-deterministic) worlds

- Unfortunately, our Q-learning (for deterministic worlds) does not work anymore
- Why not?

실습 ex03_0_q_table_frozenlake.py 코드의 환경을 다음으로 바꾸고 실행 시

env = gym.make('**FrozenLake-v0**')

Success rate: 0.0145 가 나옵니다.

- **Solution?**
  - Listen to Q (s`) (just a little bit)
  - Update Q(s) little bit (learning rate)

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$ 이 식을 다음과 같이 바꿉니다.

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

이때 알파는 Learning rate = 0.1

실습 ex04_q_table_frozenlake.py

```
import gym
import numpy as np
import matplotlib.pyplot as plt

env = gym.make('FrozenLake-v0')

# Initialize table with all zeros
Q = np.zeros([env.observation_space.n, env.action_space.n])

# Set learning parameters
learning_rate = .85
dis = .99
num_episodes = 2000

# create lists to contain total rewards and steps per episode
rList = []
for i in range(num_episodes):
    # Reset environment and get first new observation
    state = env.reset()
    rAll = 0
    done = False

    # The Q-Table learning algorithm
    while not done:
        action = np.argmax(Q[state, :] + np.random.randn(1, env.action_space.n) / (i + 1))

        # Get new state and reward from environment
        new_state, reward, done, _ = env.step(action)

        # Update Q-Table with new knowledge using learning rate
        Q[state, action] = (1-learning_rate) * Q[state, action] \
            + learning_rate*(reward + dis * np.max(Q[new_state, :]))

        rAll += reward
        state = new_state

    rList.append(rAll)
print("Success rate: " + str(sum(rList) / num_episodes))
print("Final Q-Table Values")
print("LEFT DOWN RIGHT UP")
print(Q)
plt.bar(range(len(rList)), rList, color='b', alpha=0.4)
plt.show()
```

(출력결과)

Success rate: 0.6305

Final Q-Table Values

LEFT DOWN RIGHT UP

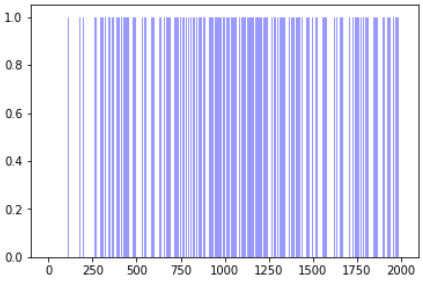
[[4.69434689e-01 6.71704966e-03 6.83730147e-03 2.23282712e-03]

 [5.60824180e-04 8.28324283e-05 8.87832913e-04 2.51383738e-01]

 [4.36906632e-03 9.89871287e-04 2.79017072e-03 6.78027769e-01]

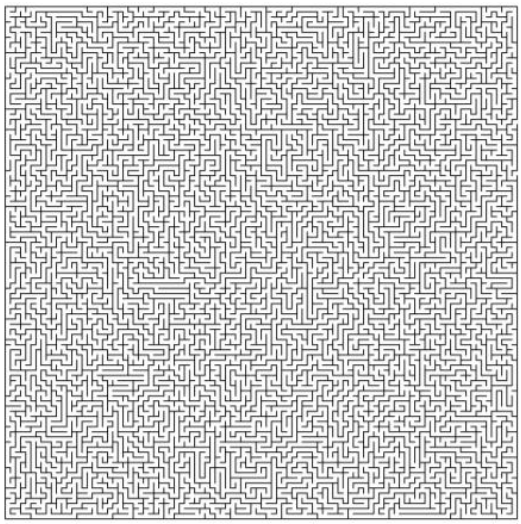
.. 중략

 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]



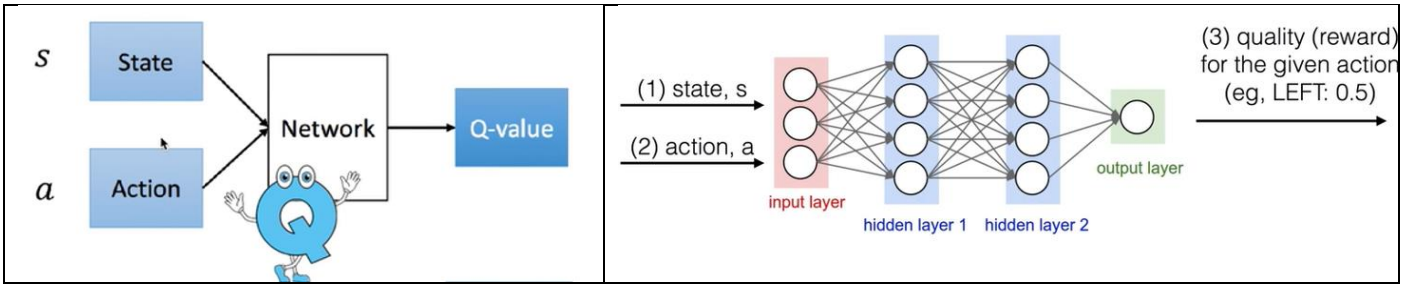
참고) http://computingkoreanlab.com/app/jAI/jQLearning/

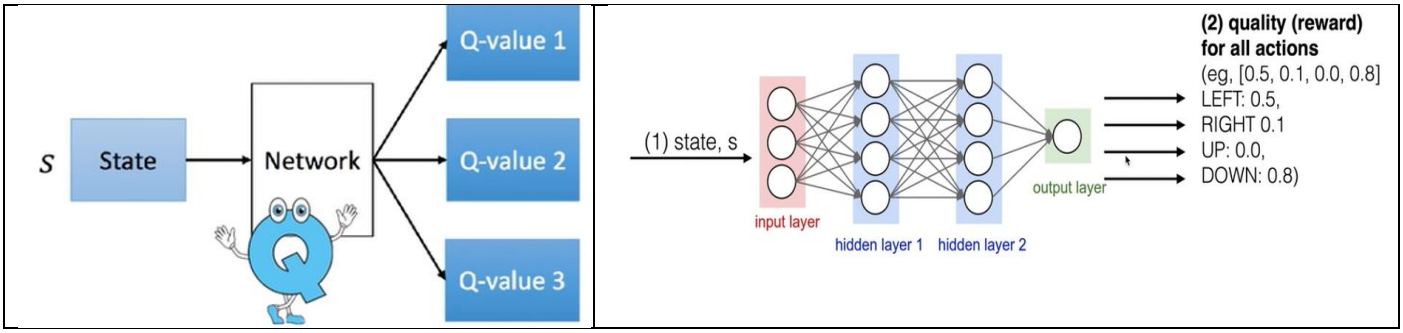
만약, 다음과 같은 100*100 미로라면 Q table은 몇 개의 값을 가져야 할까요?



[Q-Network]

Q-function Approximation

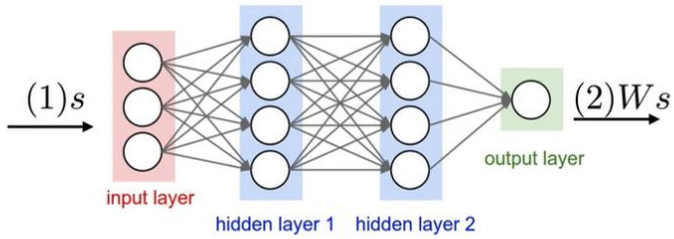




두번째 모델을 사용합니다.

Q-Network training (linear regression)

$$H(x) = Wx \qquad cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$



$$y = r + \gamma \max Q(s')$$