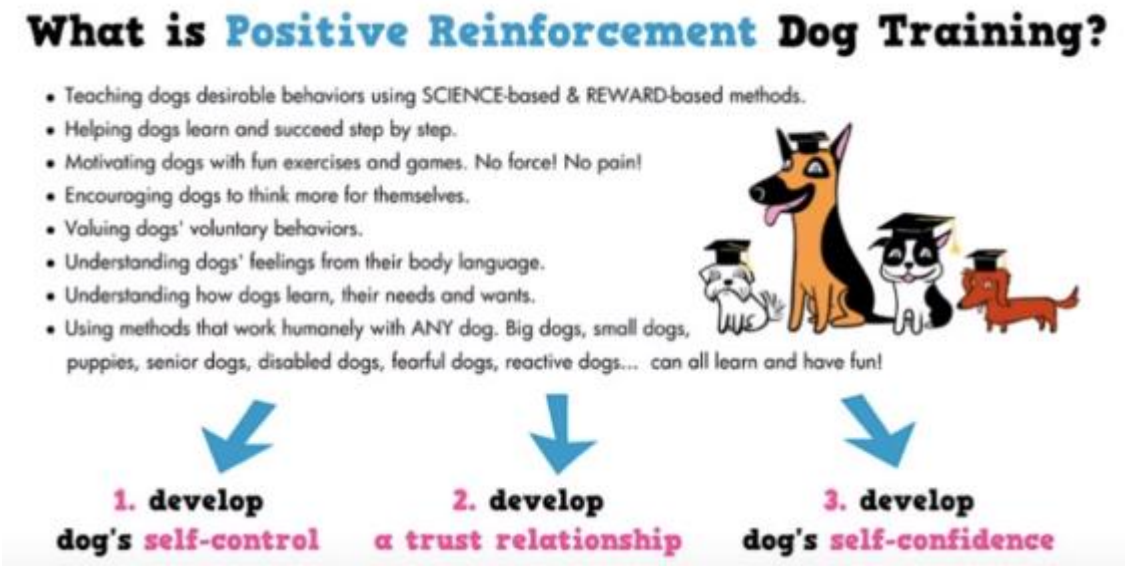
메디치소프트 기술연구소

딥러닝 9일차

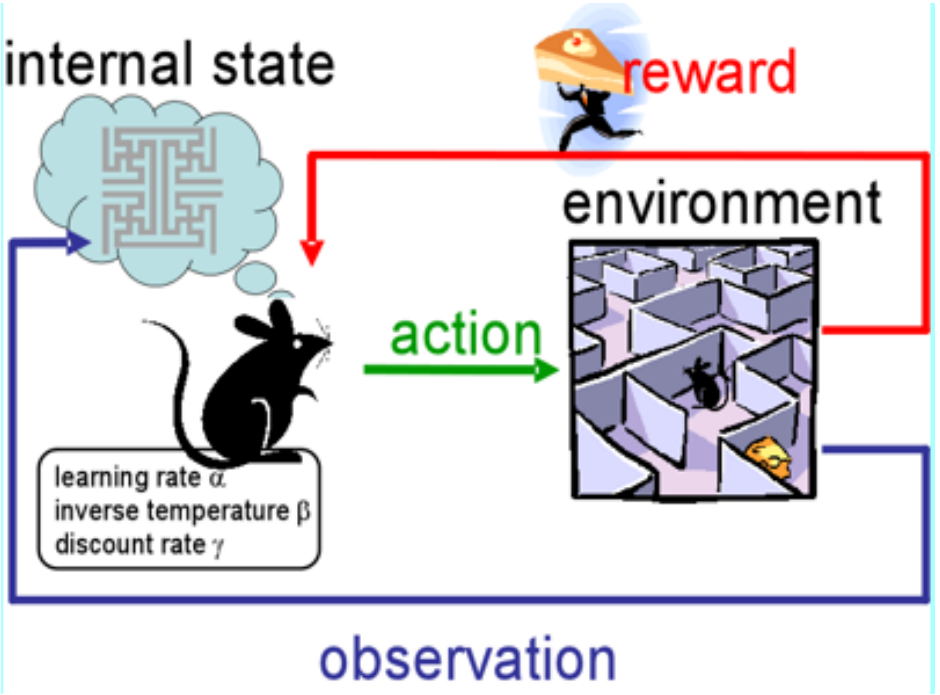
강화 학습(Reinforcement Learning)

. 1950년대부터 시작 게임과 기계 제어 분야에서 관심을 끄는 애플리케이션 많이 나옴
. 2013년 영국의 한 스타트업인 딥마인드의 연구원들이 아타리(Atari)게임을 아무 정보 없이 플레이하면서 학습하는 시스템을 시연하면서 혁명이 일어남
. 화면 픽셀에 대한 데이터만 입력, 게임 규칙에 대한 어떤 사전정보 없이 대부분 사람을 능가
. 2017년 5월 알파고가 바둑 세계챔피언 커제에게 승리
. 구글은 2014년에 딥마인드를 5억달러 인수
. 정책 그래디언트, 심층 Q-네트워크(DQN), 마르코프 결정 과정(MDP) 소개



http://angelpawstherapy.org/positive-reinforcement-dog-training.html

강화 학습(Reinforcement Learning)



. 에이전트(agent)는 관측(observation)을 하고 주어진 환경(environment)에서 행동(action)을 합니다.
=> 결과로 보상(reward)을 받습니다.
. 에이전트는 환경 안에서 행동하고 시행착오를 통해 기쁨이 최대가 되고 아픔이 최소가 되도록 학습

https://www.cs.utexas.edu/~eladlieb/RLRG.html

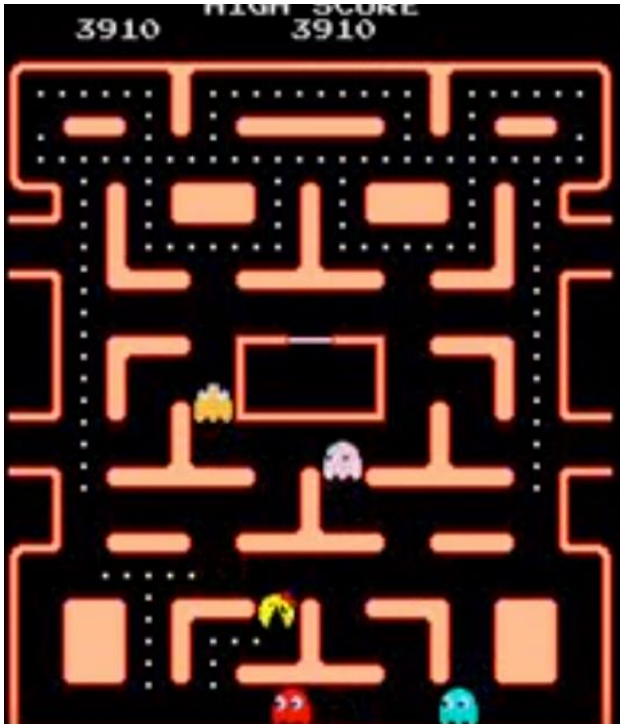
강화 학습(Reinforcement Learning)

. 머신러닝의 한분야로 주어진 환경에서 시간이 지남에 따라 보상이 최대화되는 행동을 할 수 있는
에이전트를 만드는 것을 목적으로 한다.
.지도학습과 비지도 학습의 목적은 데이터에 있는 패턴을 찾아 이를 사용해 예측을 만드는 것
.강화 학습의 목적은 좋은 정책을 찾는 것
. 지도학습과 달리 에이전트에 올바른 정답이 명시적으로 주어지지 않는다.
. 에이전트는 시행착오를 통해 학습
. 비지도 학습과 달리 보상을 통한 감독의 형태가 존재
. 에이전트에 어떻게 작업을 수행하라고 알려주지 않지만 일을 잘했는지 또는 실패 여부 알려줌
. 강화 학습 에이전트는 보상을 얻기 위해 새로운 방식을 찾는 환경의 탐험과 이미 알고 있는 보상 방법
활용하는 것 사이에 적절한 균형을 가져야 함.
. 이와 반대로 지도 학습과 비지도 학습 시스템은 탐험에 대해 신경을 쓸 필요가 없음.
즉 주어진 훈련데이터만 주입
. 지도학습과 비지도 학습에서 훈련 샘플은 일반적으로 독립적
강화 학습에서는 연속된 관측이 보통 독립적이지 않다
에이전트가 잠시 동안 움직이지 않고 환경의 같은 영역에 머물러 있을 수 있다.
연속된 관측은 매우 상호 연관이 되어 있다.
어떤 경우에는 훈련 알고리즘이 독립적인 관측을 얻을 수 있도록 재현 메모리 사용

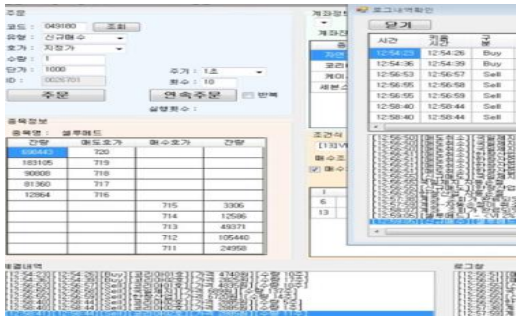
https://www.cs.utexas.edu/~eladlieb/RLRG.html

강화 학습(Reinforcement Learning)

https://www.youtube.com/watch?v=ziWYJCKQALQ



https://www.youtube.com/watch?v=B7MldKAkr6k



1. 에이전트는 보행 로봇을 제어하는 프로그램
   에이전트는 카메라나 터치 센서같은 여러 센서들을 통해 환경 관찰, 행동은 모터를 구동하기 위해 시그널을 전송
   목적지에 도착할 대 양수의 보상을 받고, 시간을 낭비하거나 잘못된 방향으로 향하거나 넘어질 때 음수의 보상을 받도록 프로그램
2. 미스 팩맨: 아타리 게임 시뮬레이션이고, 행동은 가능한 아홉 개의 조이스틱 위치(왼위, 아래, 가운데 등) 관측은 스크린샷이 되고
   보상은 게임의 점수
3. 에이전트가 바둑같은 게임을 플레이
4. 주식시장의 가격을 관찰, 매초 얼마나 사고 팔아야 할지 결정(금전적 이익과 손실)

강화 학습(Reinforcement Learning)

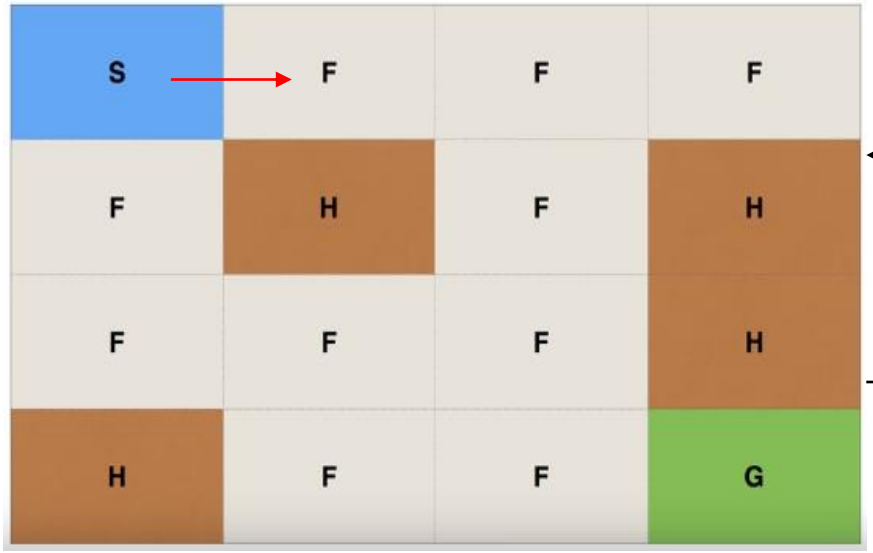
. 소프트웨어 에이전트가 행동을 결정하기 위해 사용하는 알고리즘: 정책(Policy)
. Ex: 관측을 입력으로 받고 수행할 행동을 출력하는 신경망이 정책

. 정책: 매 초마다 어떤 확률P만큼 전진하는 것도 있고,
또는 (1-p)의 확률로 왼쪽 또는 오른쪽으로 랜덤하게 회전
. 회전 각도 : -r과 +r사이의 랜덤한 각도 :
확률적 정책(stochastic policy)
.=> 2개의 정책 파라미터 policy parameter :
확률p와 각도의 범위 r 알고리즘 선택?

. Open AI 짐(https://gym.openai.com)
- 에이전트가 훈련시키기 위해 먼저 작업 환경을 마련
- 훈련을 위한 최소한의 시뮬레이션 환경 필요
- 다양한 종류의 시뮬레이션 환경(아타리 게임, 보드 게임, 2D와 3D 물리 시뮬레이션 등)을 제공하는 툴
- 에이전트를 훈련시키고 이들을 비교 또는 새로운 RL 알고리즘을 개발

강화 학습(Reinforcement Learning) –OpenAI GYM Games

Frozen Lake



Environment

Action
(right, left,up,down)

State, reward

강화 학습(Reinforcement Learning) –OpenAI GYM Games

. 설치 방법

- Python
- TensorFlow
  - sudo apt-get install python-pip python-dev
  - pip install tensorflow (or pip install tensorflow-gpu)
- OpenAI Gym
  - sudo apt install cmake
  - apt-get install zlib1g-dev
  - sudo -H pip install gym
  - sudo -H pip install gym[atari]

강화 학습(Reinforcement Learning) –OpenAI GYM Games

. 키보드를 이용한 게임

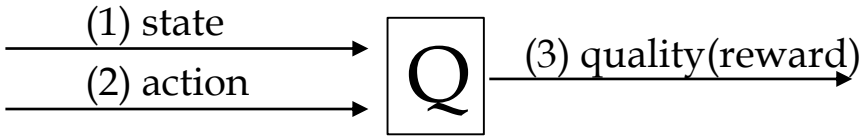
```
import readchar  # pip3 install readchar
# MACROS
LEFT = 0
DOWN = 1
RIGHT = 2
UP = 3
# Key mapping
arrow_keys = {
    '\x1b[A': UP,
    '\x1b[B': DOWN,
    '\x1b[C': RIGHT,
    '\x1b[D': LEFT}
# Register FrozenLake with is_slippery False
register(
    id='FrozenLake-v3',
    entry_point='gym.envs.toy_text:FrozenLakeEnv',
    kwargs={'map_name': '4x4', 'is_slippery': False}
)
env = gym.make('FrozenLake-v3')
env.render()  # Show the initial board
```

```
while True:
    # Choose an action from keyboard
    key = readchar.readkey()
    if key not in arrow_keys.keys():
        print("Game aborted!")
        break
    action = arrow_keys[key]
    state, reward, done, info = env.step(action)
    env.render()  # Show the board after action
    print("State: ", state, "Action: ", action,
          "Reward: ", reward, "Info: ", info)

    if done:
        print("Finished with reward", reward)
        break
```

```
SFFF
FHFH
F←[41mF←[0mFH
HFFG
State:  9 Action:  2 Reward:  0.0 Info:  {'prob': 1.0}
  (Down)
SFFF
FHFH
FFFH
H←[41mF←[0mFG
State:  13 Action:  1 Reward:  0.0 Info:  {'prob': 1.0}
  (Right)
SFFF
FHFH
FFFH
HF←[41mF←[0mG
State:  14 Action:  2 Reward:  0.0 Info:  {'prob': 1.0}
  (Right)
SFFF
FHFH
FFFH
HFF←[41mG←[0m
State:  15 Action:  2 Reward:  1.0 Info:  {'prob': 1.0}
Finished with reward 1.0
```

강화 학습(Reinforcement Learning) –Q-learning(Table)

Q-function(state-action value function)



Q(state, action)

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

Max Q = $\max_{a'} Q(s, a')$. argmax Q(s1,a) -> right

$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$

Optimal Policy

학습???

State, action, reward

$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

state

action

reward

Terminal state

Future reward

$$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

state

action

reward

Terminal state

$$R = r_1 + r_2 + r_3 + \cdots + r_n$$

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

강화 학습(Reinforcement Learning) Q-러닝

```
import gym
import numpy as np

possible_actions = [[0, 1, 2], [0, 2], [1]]

def policy_random(state):
    return np.random.choice(possible_actions[state])

n_states = 3
n_actions = 3
n_steps = 20000
alpha = 0.01
gamma = 0.99

exploration_policy = policy_random
q_values = np.full((n_states, n_actions), -np.inf)
transition_probabilities = [
        [[0.7, 0.3, 0.0],[1.0, 0.0, 0.0],[0.8, 0.2, 0.0]], # s0에서, 행동 a00이 선택되면 0.7의 확률로 상태 s0로 가고 0.3의 확률로 상태 s
        [[0.0, 1.0, 0.0],None,           [0.0, 0.0, 1.0]],
        [None,           [0.8, 0.1, 0.1], None],
         ]
```

Q-러닝은 에이전트가 플레이하는 것(가령, 랜덤하게)을 보고 점진적으로 Q-가치 추정을 향상시킵니다. 정확한 (또는 충분히 이에 가까운) Q-가치가 추정되면 최적의 정책은 가장 높은 Q-가치(즉, 그리디 정책)를 가진 행동을 선택하는 것이 됩니다.

강화 학습(Reinforcement Learning) Q-러닝

```
rewards = [
        [[+10, 0, 0], [0, 0, 0], [0, 0, 0]],
        [[0, 0, 0], [0, 0, 0], [0, 0, -50]],
        [[0, 0, 0], [+40, 0, 0], [0, 0, 0]],
    ]
def run_episode(policy, n_steps, start_state=0, display=True):
    env = MDPEnvironment()
    if display:
        print("상태 (+보상):", end=" ")
    for step in range(n_steps):
        if display:
            if step == 10:
                print("...", end=" ")
            elif step < 10:
                print(env.state, end=" ")
        action = policy(env.state)
        state, reward = env.step(action)
        if display and step < 10:
            if reward:
                print("({})".format(reward), end=" ")
    if display:
        print("전체 보상 =", env.total_rewards)
    return env.total_rewards
```

강화 학습(Reinforcement Learning) Q-러닝

```
def optimal_policy(state):
    return np.argmax(q_values[state])

class MDPEnvironment(object):
    def __init__(self, start_state=0):
        self.start_state=start_state
        self.reset()
    def reset(self):
        self.total_rewards = 0
        self.state = self.start_state
    def step(self, action):
        next_state = np.random.choice(range(3), p=transition_probabilities[self.state][action])
        reward = rewards[self.state][action][next_state]
        self.state = next_state
        self.total_rewards += reward
        return self.state, reward
```

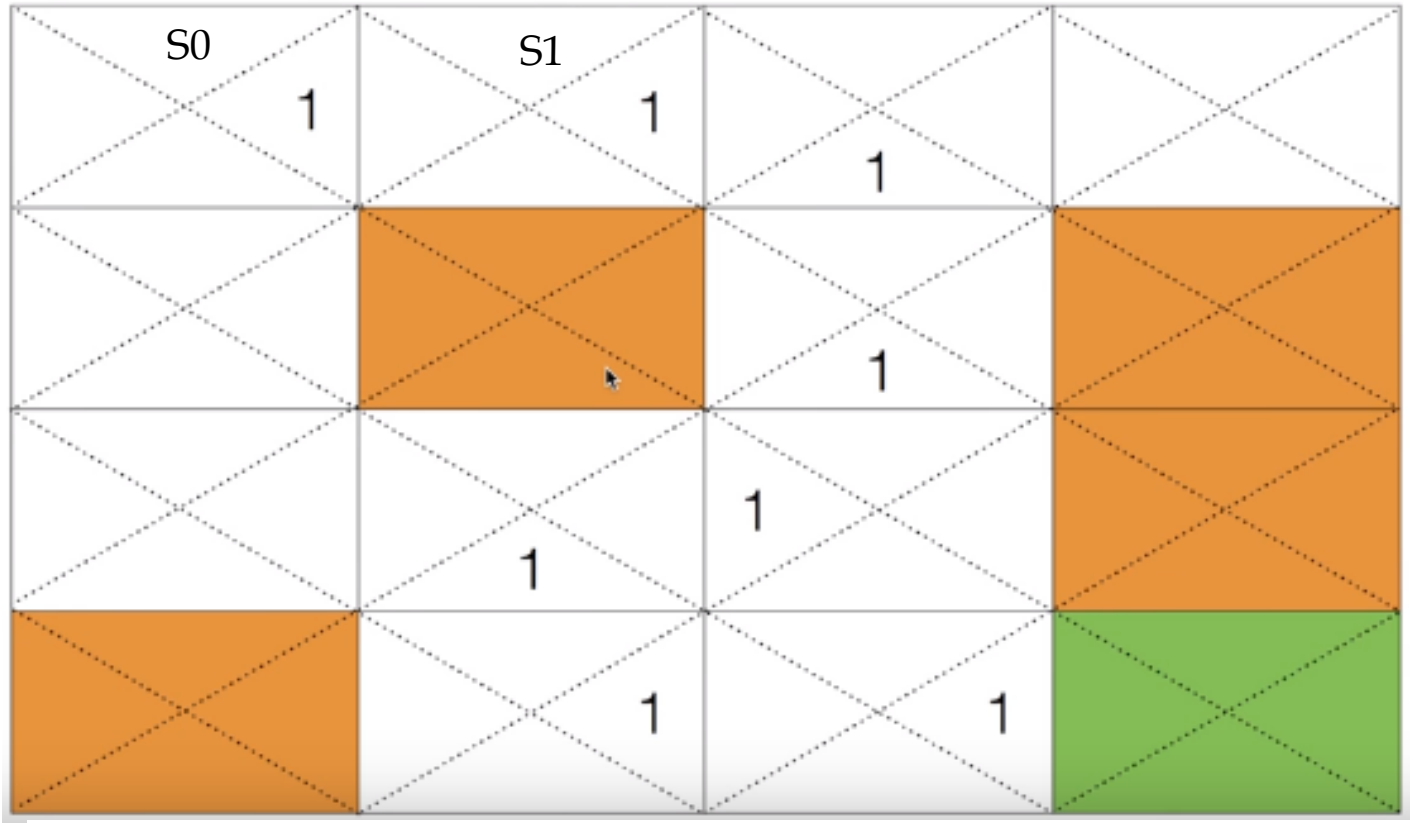
강화 학습(Reinforcement Learning) Q-러닝

```
for state, actions in enumerate(possible_actions):
    q_values[state][actions]=0
env = MDPEnvironment()

for step in range(n_steps):
    action = exploration_policy(env.state)
    state = env.state
    next_state, reward = env.step(action)
    next_value = np.max(q_values[next_state]) # 그리디한 정책
    q_values[state, action] = (1-alpha)*q_values[state, action] + alpha*(reward + gamma * next_value)

all_totals = []
for episode in range(1000):
    all_totals.append(run_episode(optimal_policy, n_steps=100, display=(episode<5)))
print("요약: 평균={:.1f}, 표준 편차={:1f}, 최소={}, 최대={}".format(np.mean(all_totals), np.std(all_totals), np.min(all_totals), np.max(all_totals)))
print()
```

강화 학습(Reinforcement Learning)- Dummy Q-learning (table)

. Learning Q(s, a) Table : one success!

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$




```
Final Q-Table Values
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. 1. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 0.]]
```

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

강화 학습(Reinforcement Learning)- Dummy Q-learning (table) 예제

```
import gym
import numpy as np
import matplotlib.pyplot as plt
from gym.envs.registration import register
import random as pr
# https://gist.github.com/stober/1943451
def rargmax(vector):
    """ Argmax that chooses randomly among eligible maximum indices.
    m = np.amax(vector)
    indices = np.nonzero(vector == m)[0]
    return pr.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')
```

강화 학습(Reinforcement Learning)- Dummy Q-learning (table) 예제

```
# Initialize table with all zeros
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):
    # Reset environment and get first new observation
    state = env.reset()
    rAll = 0
    done = False

    # The Q-Table learning algorithm
    while not done:
        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 learning rate
        Q[state, action] = reward + np.max(Q[new_state, :])

        rAll += reward
        state = new_state
```

16 X 4

게임이 끝날 때까지

$$\hat{Q}(s,a) \leftarrow 0$$

. s, a initialize table
. current state a
. For a문
- action a 선택 실행
- reward r 즉시 보상
- 새로운 s'에 대해 관찰
- update table

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

강화 학습(Reinforcement Learning)- Dummy Q-learning (table) 예제

```
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 = rargmax(Q[state, :])

        # 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 + 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="blue")
plt.show()
```