1월 5주차 레포트

1. Ray와 RLlib 개념과 설치

(1). Ray란?

Ray는 분산처리를 위한 프로젝트로 파이썬으로 작성

시간이 오래 걸리는 강화학습을 더욱 효율적으로 처리할 수 있다.

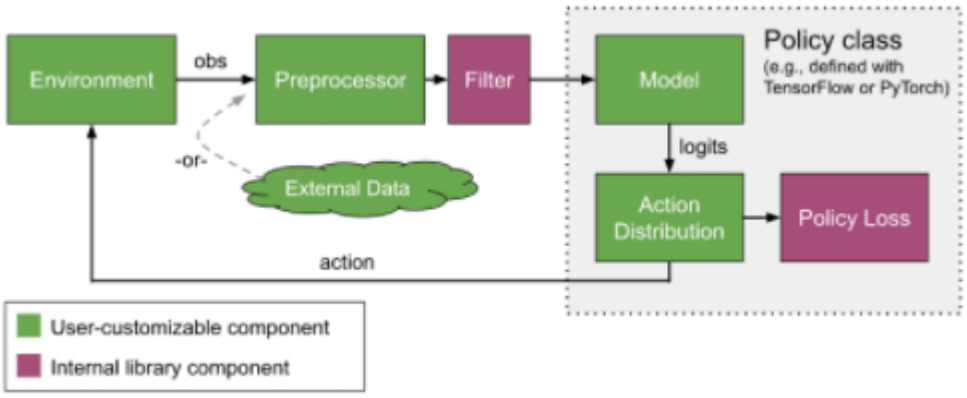
OpenAI Gym과도 연동이 가능하다

(2). Ray 설치

```
pip3 install -U ray
>> Install the latest official version of Ray

pip3 install -U "ray[tune]"
pip3 install -U "ray[rllib]"
pip3 install -U "ray[serve]"
>> Install Ray libraries
```

(3). RLlib이란?

Reinforcement Learning(RL - 강화학습)을 위한 Ray 기반의
Python의 오픈 소스 라이르러리



1. 초록색 : 유저가 바꿀 수 있는 부분
    a. Environment : 환경
    b. Preprocessor : 관측 전처리
    c. Model : Neural Net
    d. Action Distribution : Model의 출력을 해석해 다음 출력을 결정

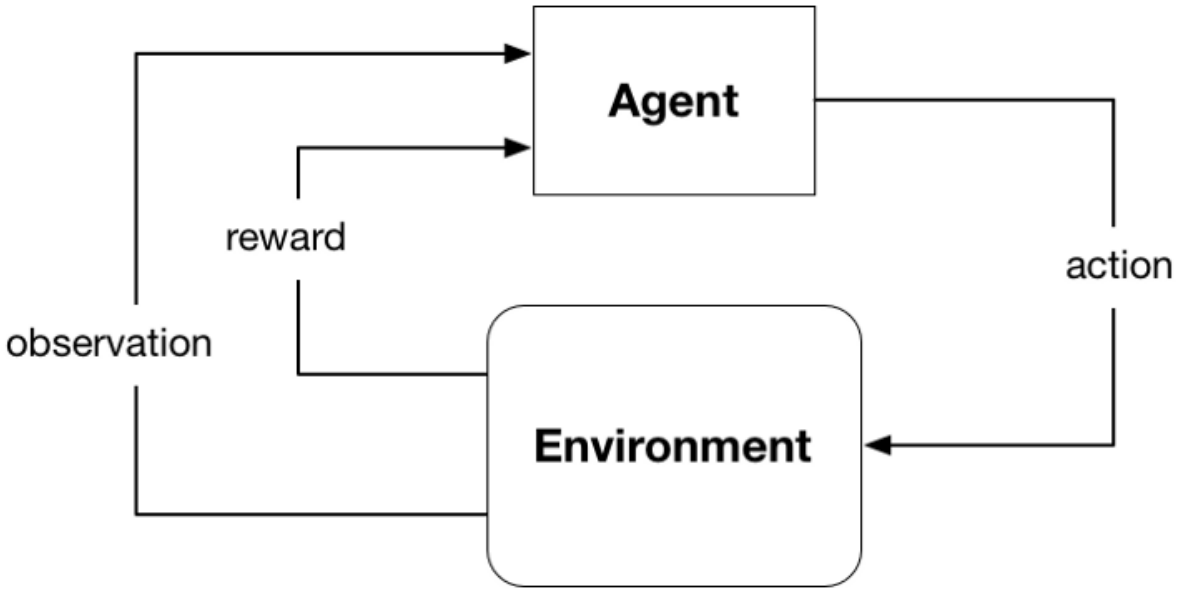
2. 자주색 : 정해진 부분
    a. Filter : 관측 전처리
    b. Policy Loss : 정책에 대한 보상

(4). RLlib 설치

```
pip3 install "ray[rllib]" tensorflow torch

pip3 install -U gym
pip3 install "gym[atari]" "gym[accept-rom-license]" atari_py
>> Atari example 사용용
```

2. Ray & RLlib 사용

(1). 강화학습이란?



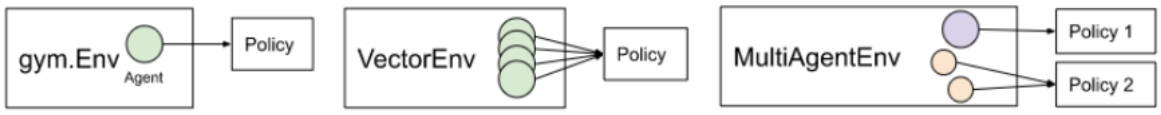
Agent(에이전트) : 환경 상태를 받고 액션을 취하고 보상을 받는 주체

Environment(환경) : 에이전트가 작업할 장소, 보상과 상태, 조치를 제공

Episode(에피소드) : 에이전트가 반복된 시도를 통해서 학습하는 시퀀스

(2). Policy

Agent를 조종하기 위한 Policy 구조



1. 좌측 : Single agent에 대한 하나의 Policy 할당
2. 중간 : Multi agent에 대한 하나의 Policy 할당
3. 우측 : Multi agent에 대한 각각의 다른 Policy 할당

(3). Ray Core API

Ray Core API 정리 By zzsza

API	API 설명	API 예시	
ray.init()	Ray 드라이버 실행 CPU/GPU 갯수, Ray Cluster 연결 정보, 대시보드 정보 등을 인자로 받을 수 있음	ray.init() ray.init(address="108.20.14.23:6379", ignore_reinit_error=True)	
@ray.remote	파이썬 함수나 클래스를 다른 프로세스에서 실행되는 Ray Task로 만들어주는 데코레이터 필요시 인자를 받을 수 있음(num_gpus, max_calls 등)	@ray.remote def func(x): return x	@ray.remote class SomeClass: def method_a(y) return y
.remote()	Remote Function, Remote Class 선언 호출시 사용하는 접미사 기존 함수의 인자를 받을 수 있음 **Remote 작업은 비동기적(Asynchronous)으로 실행**	obj_id = func.remote(x) actor = SomeClass.remote() actor_obj_id = actor.method_a.remote(y)	
ray.put()	Objest Store에 Object를 저장하고 해당 ID 반환함 Remote Function 호출시 인자로 사용할 수 있음 **ray.put()은 동기적(synchronous)으로 실행**	x_id = ray.put(x)	
ray.get()	ObjectRef(Object Id)를 인자로 받아 Object Value를 반환함 **ray.get()은 동기적(synchronous)으로 실행**	result = ray.get(obj_id) result = ray.get(x_id) # ray.put의 결과를 인자로 활용	
ray.wait()	준비된 Object Id와 준비되지 않은 Object Id를 반환함 Task가 완료될 때까지 기다림	ready_ids, not_ready_ids = ray.wait(obj_ids)	
ray.shutdown()	ray.init로 시작된 프로세스를 종료하고 워커와 연결을 끊음 Ray를 사용한 파이썬 프로세스가 종료되면 자동으로 이 코드가 실행됨	ray.shutdow()	

(4). RLlib 예제 - 택시

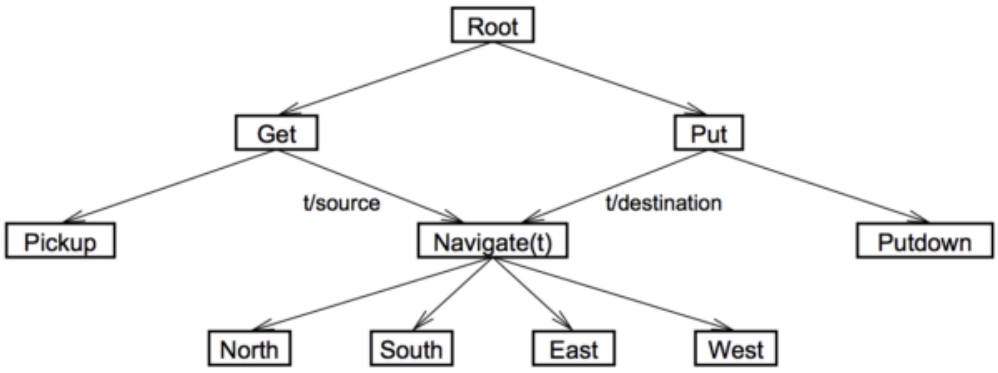


Figure 2: A task graph for the Taxi problem.

[지도 구성]

R : 북서쪽 구석의 Red 목적지

G : 북동쪽 구석의 Green 목적지

Y : 남서쪽 구석의 Yellow 목적지

B : 남동쪽 구석의 Blue 목적지

: : 택시가 운전할 수 있는 셀

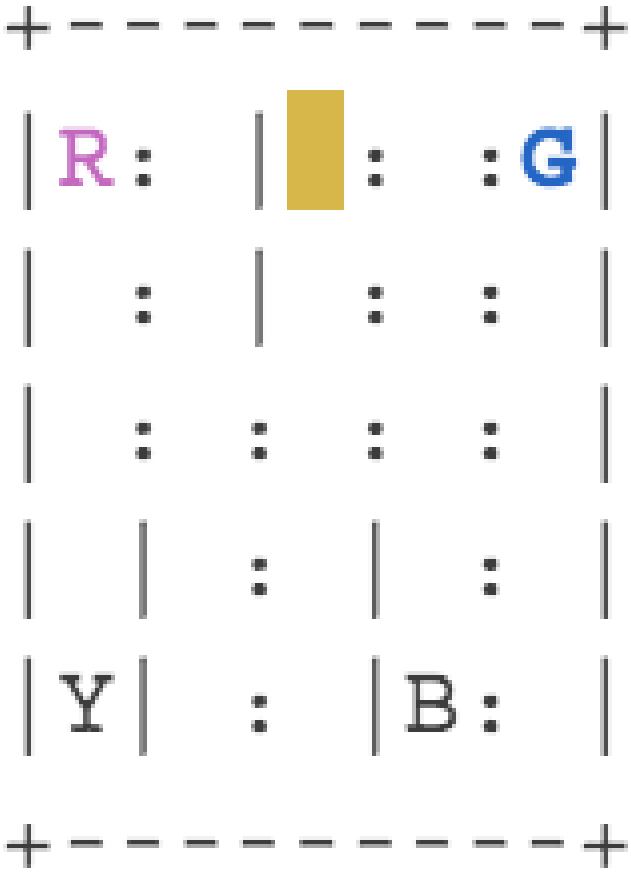
| : 택시가 피해야하는 장애물

파란색 글자 : 현재 승객의 승차 위치

자주색 글자 : 현재 승객의 하차 목적지

노란색 박스 : 비어 있을 때 택시의 위치

초록색 박스 : 승객이 탔을 때 택시의 위치



[액션 구성]

택시가 북쪽으로 한 칸 이동

택시가 남쪽으로 한 칸 이동

택시가 동쪽으로 한 칸 이동

택시가 서쪽으로 한 칸 이동

승객 승차

승객 하차

[보상 구성]

+20 : 택시가 승객을 올바른 하차를 완료했을 시

-10 : 택시가 승객의 승/하차를 올바르게 못 했을 시

[코드]

```
// 라이브러리 및 Map 구성

import sys
from contextlib import closing
from io import StringIO
from typing import Optional

import numpy as np
from gym import Env, spaces, utils
from gym.envs.toy_text.utils import categorical_sample

MAP = [
    "+---------+",
    "|R: | : :G|",
    "| : | : : |",
    "| : : : : |",
    "| | : | : |",
    "|Y| : |B: |",
    "+---------+",
]
```

```
// Environment 구성

class TaxiEnv(Env) :

  def __init__(self):
      ...

  def encode(self, taxi_row,taxi_col,pass_loc,dest_idx):
      ...

  def decode(self, i):
      ...
```

```
  def step(self, action):
      ...

  def reset(self, *, send : Optional[int] = None, options : Optional[dict] = None):
      ...

  def render(self, mode="human"):
      ...
```

3. Gym 사용하기

(1). Gym 환경 클래스 구현

```
1. gym.Env를 상속
2. _reset, _step, _render을 구현
3. Gym 에이전트의 메소드 구성
  reset
  step
  render
```

(2). Agent 구성

1. HumanAgent
    a. 사람이 렌더링 결과를 보고 숫자를 입력해 플레이
    b. 문제를 풀기전 환경 파악
    c. 강화학습 에이전트 학습 후 검증에도 사용

2. BaseAgent
    a. 간단한 룰 기반 에이전트
        i. 알을 두어서 이기는 곳이 있으면 그곳에 둔다
        ii. 아니라면 랜덤한 위치에 둔다
    b. 최소한의 가이드라인 역할

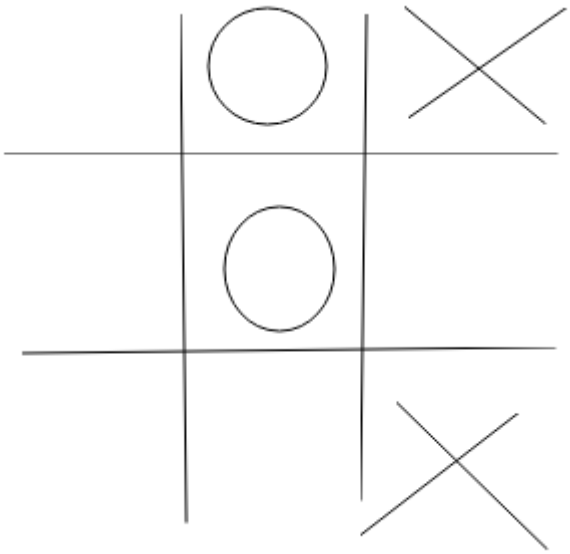
(3). 보상의 방식

1. ‘O’의 승리 >> ‘X’의 패배
    a. ‘O’의 승리 보상 : +1
    b. ‘X’의 승리 보상 : -1
    c. 게임 진행 및 무승부 : 0

2. ‘O’는 보상을 최대화 하려하고, ‘X’는 보상을 최소화 하려한다.

(4). 보드판에서의 상태표현 방식

1. 보드의 상태를 마크코드 리스트로
    a. ‘O’ : 1
    b. ‘X’ : 2
    c. 예제 : [0,1,2,0,1,0,0,2]



2. 누구의 차례인지에 따라 보상의 가치가 다르다
    a. ‘O’의 차례 : ‘O’의 승리
    b. ‘X’의 차례 : ‘X’의 승리

3. 보드의 상태 + 차례 로 상태를 구성

a. 예제 : [0,1,2,0,1,0,0,0,2], ‘O’

(5). 상태 & 동작 공간

(6). 코드

```
// Environment

class TicTacToeEnv(gym.Env):
>> gym.Env를 받는다

  metadata = {'render.modes' : ['human']}
  >> 렌더링 모드를 표시한다.

  def step(self,action) :
  >>> 매 스탭마다 agent의 action에 의한 정보들을 저장

  def reset(self) :
  >>> environment를 초기화

  def render(self,mode='human',close=False):
  >>> step으로 주어진 정보들을 토대로 렌더링 실행

>>> Environment은 위와 같이
  step, reset, render 로 구성되어있다.
```

```
// Reset

def reset(self) :
>> Environment를 초기화

  self.board = [0] * 9
  >> 틱택토가 그려질 보드를 전부 초기화

  self.mark = '0'
  self.done = Flase

  return self._get_obs()

def _get_obs(self) :
>> 관측한 상태 (보드 상태 + 다음 차례인 마크 형식)
  return tuple(self.board), self.mark
```

```
// Step

def step(self,action) :
>> 스탭마다 에이전트에 의한 액션들을 참고하여 정보 추출

  loc = action
  reward = NO_REWARD
  self.board[loc] = tocode(self.mark)

  status = check_game_status(self.board)
  >> 승/패가 결정되면 마크에 맞는 리워드 반환

  if status >=0 :
    self.done = True
    if status in [1,2] :
      reward = O_REWARD if self.mark == '0' else X_REWARD

  self.mark = next_mark(self.mark)
  >> 다음 플레이어(마크)로 교체

  return self._get_obs(), reward, self.done, None
  >> 새 상태, 보상, 에피소드 종료 여부 반환
```

```
// Render

def render(self, mode='human', close=False):
>> 렌더링
  if mode == 'human' :
  >> 'human' 모드에서만 보드를 표시하도록 설정
    self._show_board(print)

def _show_board(self,showfn):
>> 보드에 그리기

  for j in range(0,9,3):
    print('|'.join([tomark(self.board[i])])
    >> 마크 코드를 문자로
      >> 0 : ''
      >> 1 : 'O'
      >> 2 : 'X'

    for i in range(j,j+3):
      if j <6  :
        print('--------')
```

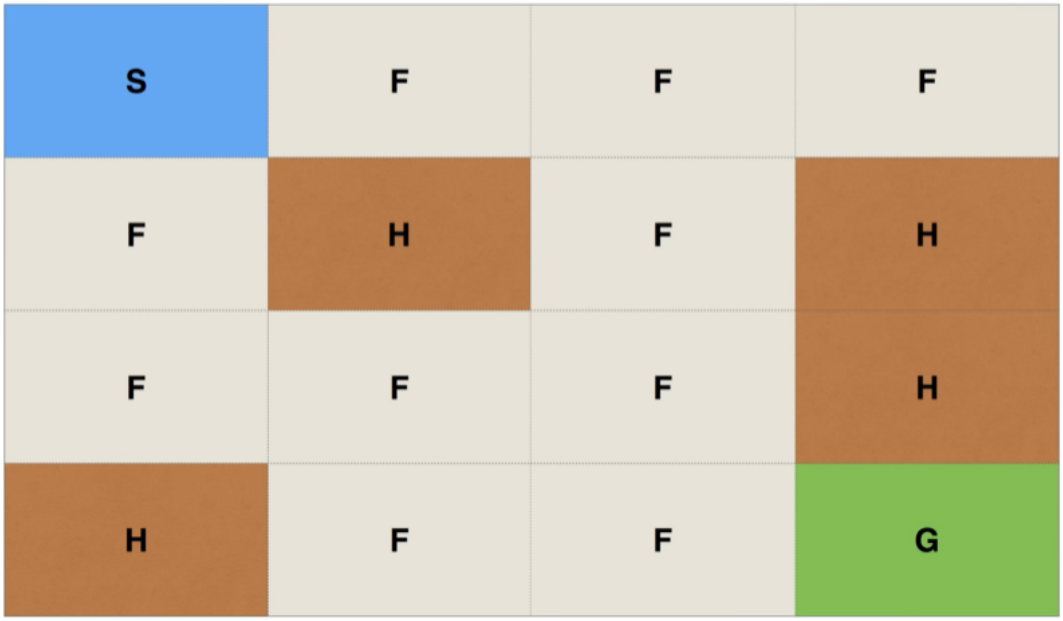
```
// Play(main)
```

```
>>> 아직 이 부분까지 못 넘어옴

def play(show_number):
  env = TicTacToeEnv(show_number=show_number)
  agents = [HumanAgent('O'), HumanAgent('X')]
  episode = 0

  while True :
    state = env.reset()
    _, mark = stae
    done = False
    env.render()

    while not done :
      agent = agent_by_mark(agents,next_mark(mark))
      env.show_turn(True,mark)
      ava_actions = env.available_actions()
      action = agent.act(ava_actions)

      if action is None :
        sys.exit()

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

      priint('')
      env.render()

      if done :
        env.show_result(True,mark,reward)
      else :
        _, mark = state

  episode += 1
```

4. 다른 예제들

(1). 예제 - FrozenLake

Frozen Lake



[이미지 설명]

S : 시작점 (Starting Point)

F : 얼은 면 (Frozen Surface)

H : 구멍 (Hole)

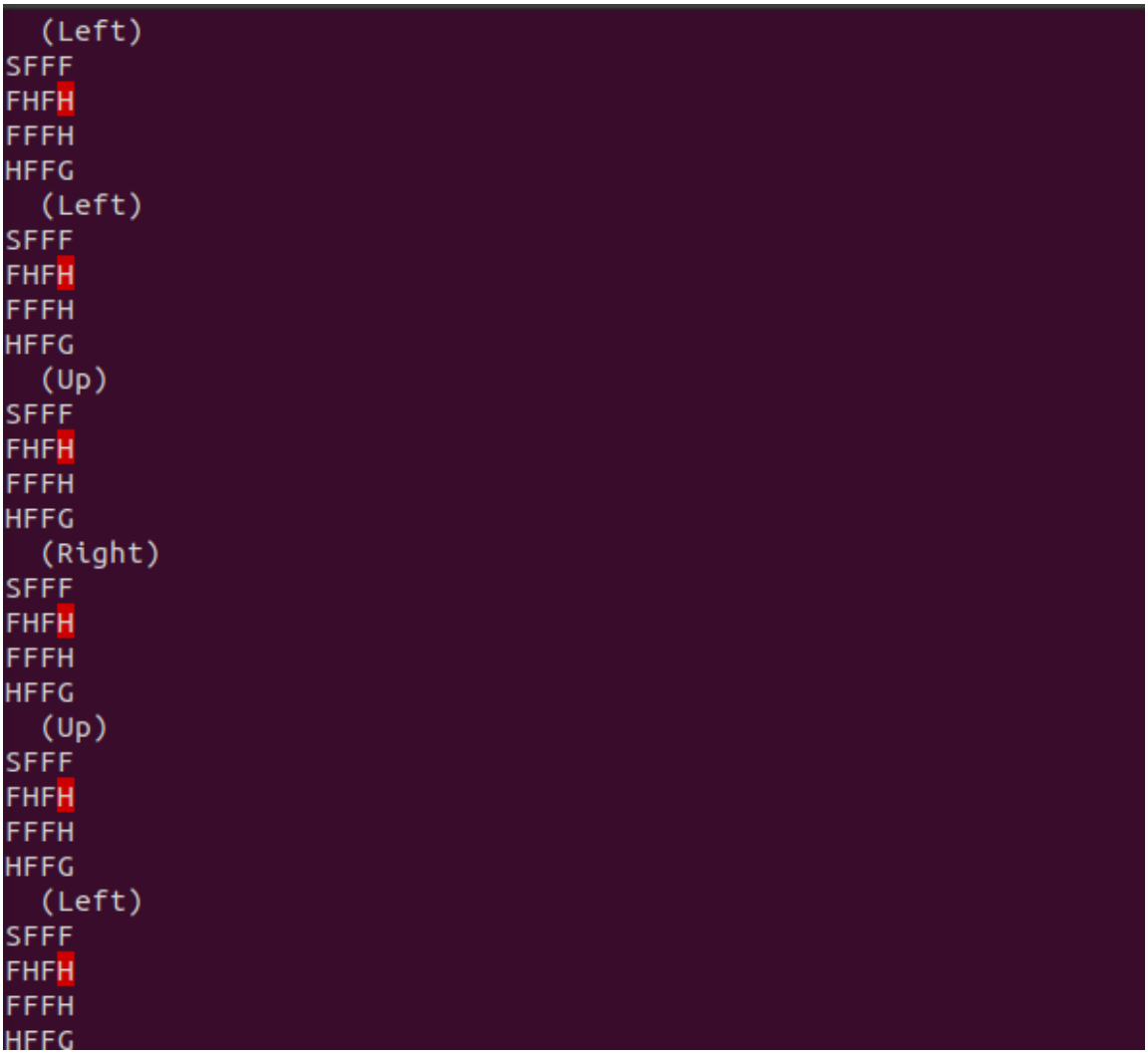
G : 목표지점 (Goal)

[목적]

시작 지점에서 구멍에 빠지지 않고 목표지점에 도착

```
import gym
env = gym.make("FrozenLake-v0")
>> make 함수를 통해서 환경 생성
>> 이미 만들어져 있는 환경을 불러오는 것

observation = env.reset()
>> 환경을 사용하기 전에 reset() 필요

for _ in range(1000) :
  env.render()
```

```
  >> 현재의 환경을 렌더링

  action = env.action_space.sample()
  >> 에이전트 액션 실행

  observation, reward, done, info = env.step(action)
  >> 액션을 매 스탭마다 취한 결과 추출
```




```
// 전체 코드 - main.py //

import gym

env = gym.make("FrozenLake-v1")

observation = env.reset()

for _ in range(1000):
  env.render()
```

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

```
// frozen_lake.py - 라이브러리, 랜덤 맵 생성 //

import sys
from contextlib import closing
from io import StringIO
from typing import Optional

import numpy as np
from gym import Env, spaces, utils
from gym.envs.toy_text.utils import categorical_sample

LEFT = 0
DOWN = 1
RIGHT = 2
UP = 3

MAPS = {
    "4x4": ["SFFF", "FHFH", "FFFH", "HFFG"],
    "8x8": [
        "SFFFFFFF",
        "FFFFFFFF",
        "FFFHFFFF",
        "FFFFFHFF",
        "FFFHFFFF",
        "FHHFFFHF",
        "FHFFHFHF",
        "FFFHFFFG",
    ],
}

def generate_random_map(size=8, p=0.8):
    """Generates a random valid map (one that has a path from start to goal)
    :param size: size of each side of the grid
    :param p: probability that a tile is frozen
    """
    valid = False

    # DFS to check that it's a valid path.
    def is_valid(res):
        frontier, discovered = [], set()
        frontier.append((0, 0))
        while frontier:
            r, c = frontier.pop()
            if not (r, c) in discovered:
                discovered.add((r, c))
                directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
                for x, y in directions:
                    r_new = r + x
                    c_new = c + y
                    if r_new < 0 or r_new >= size or c_new < 0 or c_new >= size:
```

```
                        continue
                    if res[r_new][c_new] == "G":
                        return True
                    if res[r_new][c_new] != "H":
                        frontier.append((r_new, c_new))
        return False
```

```
// frozen_lake.py - Environment 설정 //

class FrozenLakeEnv(Env):

  def __init__(self, desc=None, map_name="4x4", is_slippery=True):
    ....

  def step(self, action) :
    ....

  def reset(self,*,seed : Optional[int] = None, options : Optional[dict] = None):
    ....

  def render(self, mode = "human"):
    ....
```