**[Sim.py]**

from string import ascii\_uppercase

from draw\_utils import \*

from pyglet.gl import \*

import numpy as np

import pandas as pd

import os

# reward

move\_reward = 0.1

obs\_reward = 0.1

goal\_reward = 10

print('reward:' , move\_reward, obs\_reward, goal\_reward)

local\_path = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_)))

class Simulator:

def \_\_init\_\_(self):

'''

height : 그리드 높이

width : 그리드 너비

inds : A ~ Q alphabet list

'''

# Load train data

self.files = pd.read\_csv(os.path.join(local\_path, "./data/factory\_order\_train.csv"))

self.height = 10

self.width = 9

self.inds = list(ascii\_uppercase)[:17]

def set\_box(self):

'''

아이템들이 있을 위치를 미리 정해놓고 그 위치 좌표들에 아이템이 들어올 수 있으므로 그리드에 100으로 표시한다.

데이터 파일에서 이번 에피소드 아이템 정보를 받아 가져와야 할 아이템이 있는 좌표만 -100으로 표시한다.

self.local\_target에 에이전트가 이번에 방문해야할 좌표들을 저장한다.

따라서 가져와야하는 아이템 좌표와 end point 좌표(처음 시작했던 좌표로 돌아와야하므로)가 들어가게 된다.

'''

box\_data = pd.read\_csv(os.path.join(local\_path, "./data/box.csv"))

# 물건이 들어있을 수 있는 경우

for box in box\_data.itertuples(index = True, name ='Pandas'):

self.grid[getattr(box, "row")][getattr(box, "col")] = 100

# 물건이 실제 들어있는 경우

order\_item = list(set(self.inds) & set(self.items))

order\_csv = box\_data[box\_data['item'].isin(order\_item)]

for order\_box in order\_csv.itertuples(index = True, name ='Pandas'):

self.grid[getattr(order\_box, "row")][getattr(order\_box, "col")] = -100

# local target에 가야 할 위치 좌표 넣기

self.local\_target.append(

[getattr(order\_box, "row"),

getattr(order\_box, "col")]

)

self.local\_target.sort()

self.local\_target.append([9,4])

# 알파벳을 Grid에 넣어서 -> grid에 2Dconv 적용 가능

def set\_obstacle(self):

'''

장애물이 있어야하는 위치는 미리 obstacles.csv에 정의되어 있다. 이 좌표들을 0으로 표시한다.

'''

obstacles\_data = pd.read\_csv(os.path.join(local\_path, "./data/obstacles.csv"))

for obstacle in obstacles\_data.itertuples(index = True, name ='Pandas'):

self.grid[getattr(obstacle, "row")][getattr(obstacle, "col")] = 0

def reset(self, epi):

'''

reset()은 첫 스텝에서 사용되며 그리드에서 에이전트 위치가 start point에 있게 한다.

:param epi: episode, 에피소드 마다 가져와야 할 아이템 리스트를 불러올 때 사용

:return: 초기셋팅 된 그리드

:rtype: numpy.ndarray

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

items : 이번 에피소드에서 가져와야하는 아이템들

terminal\_location : 현재 에이전트가 찾아가야하는 목적지

local\_target : 한 에피소드에서 찾아가야하는 아이템 좌표, 마지막 엔드 포인트 등의 위치좌표들

actions: visualization을 위해 에이전트 action을 저장하는 리스트

curloc : 현재 위치

'''

# initial episode parameter setting

self.epi = epi

self.items = list(self.files.iloc[self.epi])[0]

self.cumulative\_reward = 0

self.terminal\_location = None

self.local\_target = []

self.actions = []

# initial grid setting

self.grid = np.ones((self.height, self.width), dtype="float16")

# set information about the gridworld

self.set\_box()

self.set\_obstacle()

# start point를 grid에 표시

self.curloc = [9, 4]

self.grid[int(self.curloc[0])][int(self.curloc[1])] = -5

self.done = False

return self.grid

def apply\_action(self, action, cur\_x, cur\_y):

'''

에이전트가 행한 action대로 현 에이전트의 위치좌표를 바꾼다.

action은 discrete하며 4가지 up,down,left,right으로 정의된다.

:param x: 에이전트의 현재 x 좌표

:param y: 에이전트의 현재 y 좌표

:return: action에 따라 변한 에이전트의 x 좌표, y 좌표

:rtype: int, int

'''

new\_x = cur\_x

new\_y = cur\_y

# up

if action == 0:

new\_x = cur\_x - 1

# down

elif action == 1:

new\_x = cur\_x + 1

# left

elif action == 2:

new\_y = cur\_y - 1

# right

else:

new\_y = cur\_y + 1

return int(new\_x), int(new\_y)

def get\_reward(self, new\_x, new\_y, out\_of\_boundary):

'''

get\_reward함수는 리워드를 계산하는 함수이며, 상황에 따라 에이전트가 action을 옳게 했는지 판단하는 지표가 된다.

:param new\_x: action에 따른 에이전트 새로운 위치좌표 x

:param new\_y: action에 따른 에이전트 새로운 위치좌표 y

:param out\_of\_boundary: 에이전트 위치가 그리드 밖이 되지 않도록 제한

:return: action에 따른 리워드

:rtype: float

'''

# 바깥으로 나가는 경우

if any(out\_of\_boundary):

reward = obs\_reward

else:

# 장애물에 부딪히는 경우

if self.grid[new\_x][new\_y] == 0:

reward = obs\_reward

# 현재 목표에 도달한 경우

elif new\_x == self.terminal\_location[0] and new\_y == self.terminal\_location[1]:

reward = goal\_reward

# 그냥 움직이는 경우

else:

reward = move\_reward

return reward

def step(self, action):

'''

에이전트의 action에 따라 step을 진행한다.

action에 따라 에이전트 위치를 변환하고, action에 대해 리워드를 받고, 어느 상황에 에피소드가 종료되어야 하는지 등을 판단한다.

에이전트가 endpoint에 도착하면 gif로 에피소드에서 에이전트의 행동이 저장된다.

:param action: 에이전트 행동

:return:

grid, 그리드

reward, 리워드

cumulative\_reward, 누적 리워드

done, 종료 여부

goal\_ob\_reward, goal까지 아이템을 모두 가지고 돌아오는 finish율 계산을 위한 파라미터

:rtype: numpy.ndarray, float, float, bool, bool/str

(Hint : 시작 위치 (9,4)에서 up말고 다른 action은 전부 장애물이므로 action을 고정하는 것이 좋음)

'''

self.terminal\_location = self.local\_target[0]

cur\_x,cur\_y = self.curloc

self.actions.append((cur\_x, cur\_y))

goal\_ob\_reward = False

new\_x, new\_y = self.apply\_action(action, cur\_x, cur\_y)

out\_of\_boundary = [new\_x < 0, new\_x >= self.height, new\_y < 0, new\_y >= self.width]

# 바깥으로 나가는 경우 종료

if any(out\_of\_boundary):

self.done = True

goal\_ob\_reward = True

else:

# 장애물에 부딪히는 경우 종료

if self.grid[new\_x][new\_y] == 0:

self.done = True

goal\_ob\_reward = True

# 현재 목표에 도달한 경우, 다음 목표설정

elif new\_x == self.terminal\_location[0] and new\_y == self.terminal\_location[1]:

# end point 일 때

if [new\_x, new\_y] == [9,4]:

self.done = True

self.local\_target.remove(self.local\_target[0])

self.grid[cur\_x][cur\_y] = 1

self.grid[new\_x][new\_y] = -5

goal\_ob\_reward = True

self.curloc = [new\_x, new\_y]

else:

# 그냥 움직이는 경우

self.grid[cur\_x][cur\_y] = 1

self.grid[new\_x][new\_y] = -5

self.curloc = [new\_x,new\_y]

reward = self.get\_reward(new\_x, new\_y, out\_of\_boundary)

self.cumulative\_reward += reward

if self.done == True:

if [new\_x, new\_y] == [9, 4]:

if self.terminal\_location == [9, 4]:

# 완료되면 GIFS 저장

goal\_ob\_reward = 'finish'

height = 10

width = 9

display = Display(visible=False, size=(width, height))

display.start()

start\_point = (9, 4)

unit = 50

screen\_height = height \* unit

screen\_width = width \* unit

log\_path = "./logs"

data\_path = "./data"

render\_cls = Render(screen\_width, screen\_height, unit, start\_point, data\_path, log\_path)

for idx, new\_pos in enumerate(self.actions):

render\_cls.update\_movement(new\_pos, idx+1)

render\_cls.save\_gif(self.epi)

render\_cls.viewer.close()

display.stop()

return self.grid, reward, self.cumulative\_reward, self.done, goal\_ob\_reward

if \_\_name\_\_ == "\_\_main\_\_":

sim = Simulator()

files = pd.read\_csv("./data/factory\_order\_train.csv")

for epi in range(2): # len(files)):

items = list(files.iloc[epi])[0]

done = False

i = 0

obs = sim.reset(epi)

actions = [0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1]

while done == False:

i += 1

next\_obs, reward, cumul ,done, goal\_reward = sim.step(actions[i])

obs = next\_obs

if (done == True) or (i == (len(actions)-1)):

i =0