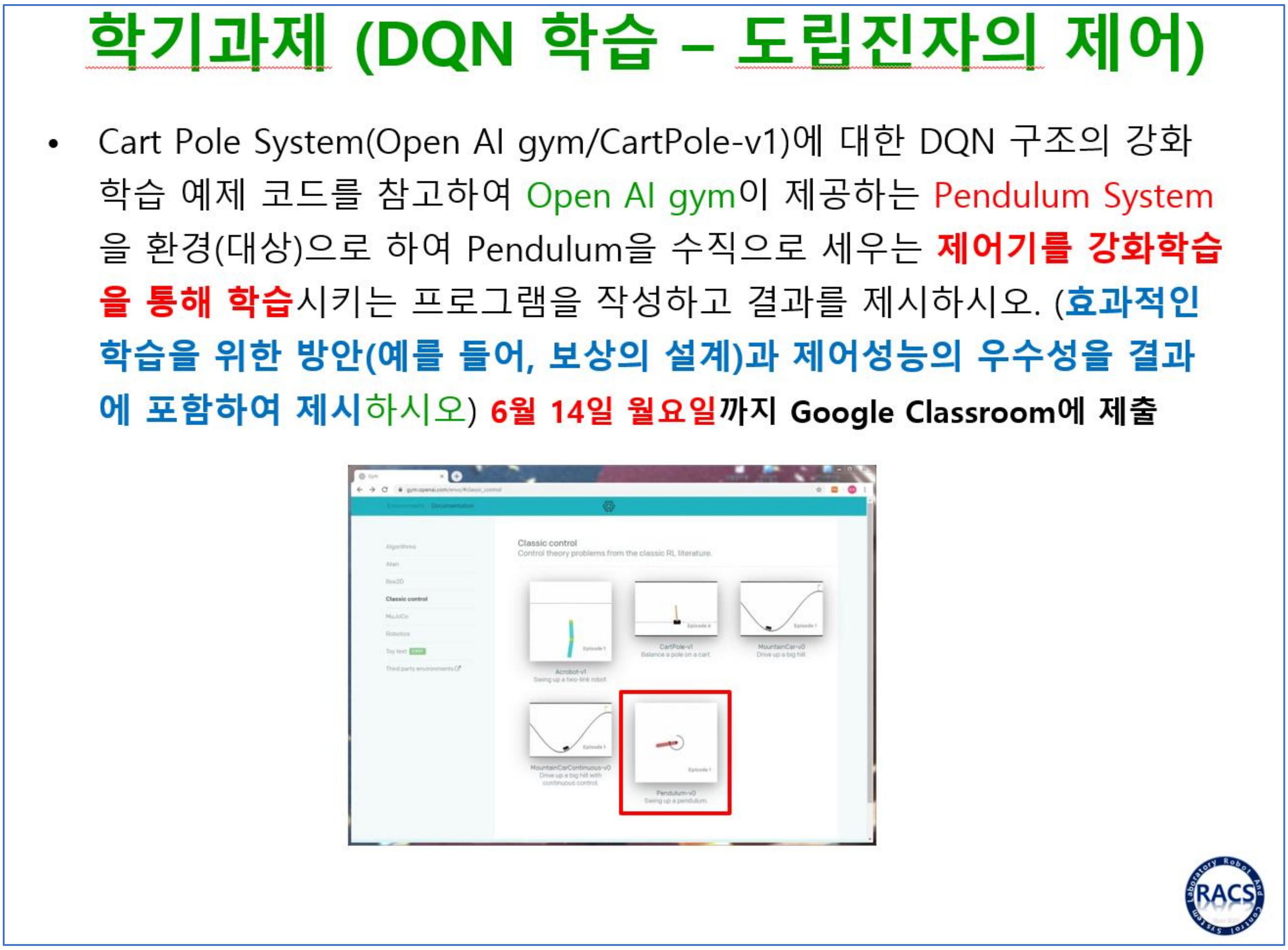
지능시스템

기말대체 과제

2019305059

이현수



학기과제 (DQN 학습 – 도립진자의 제어)

- Cart Pole System(Open AI gym/CartPole-v1)에 대한 DQN 구조의 강화 학습 예제 코드를 참고하여 Open AI gym이 제공하는 Pendulum System을 환경(대상)으로 하여 Pendulum을 수직으로 세우는 **제어기를 강화학습을 통해 학습**시키는 프로그램을 작성하고 결과를 제시하시오. (**효과적인 학습을 위한 방안(예를 들어, 보상의 설계)과 제어성능의 우수성을 결과에 포함하여 제시**하시오) **6월 14일 월요일**까지 **Google Classroom에 제출**

작업공간(이산, 연속) 및 관측공간(이산, 연속)을 보고 알고리즘을 분류할 수 있디.

1. Q-learning 은 개별 작업 공간과 개별 관측 공간을 처리할 수 있다.
2. DQN 은 개별 작업 공간과 이산 및 연속 관측 공간을 모두 처리할 수 있다.
3. DDPG 는 연속 작업 공간과 이산 및 연속 작업 공간을 모두 처리할 수 있다.

진자 환경과 같이 지속적인 작업 공간이있는 경우 Q-learning 만으로는 연속 작업 공간에서 행동하는 방법을 배울 수 없으므로 정책 그라데이션 모델을 ddpg 로 사용해야 한다.

Q-learning 에서는 Q-값을 극대화하는 액션을 선택하려고 한다. 이때 단순한 방법으로 가능한 각 작업을 하나씩 살펴보고 Q 값이 가장 높은 것을 선택하는 것이다. 그러나 연속 공간에서는 무한한 양의 가능한 작업이므로 이 방법을 사용할 수 없다.

지속적인 작업을 처리하는 방법에는 여러 가지가 있다. 정책 그라데이션 방법, 소프트 액터 비평가, DDPG 등과 같은 많은 알고리즘이 있다.

이번 과제에서는 DDPG 를 사용해 구현했다.

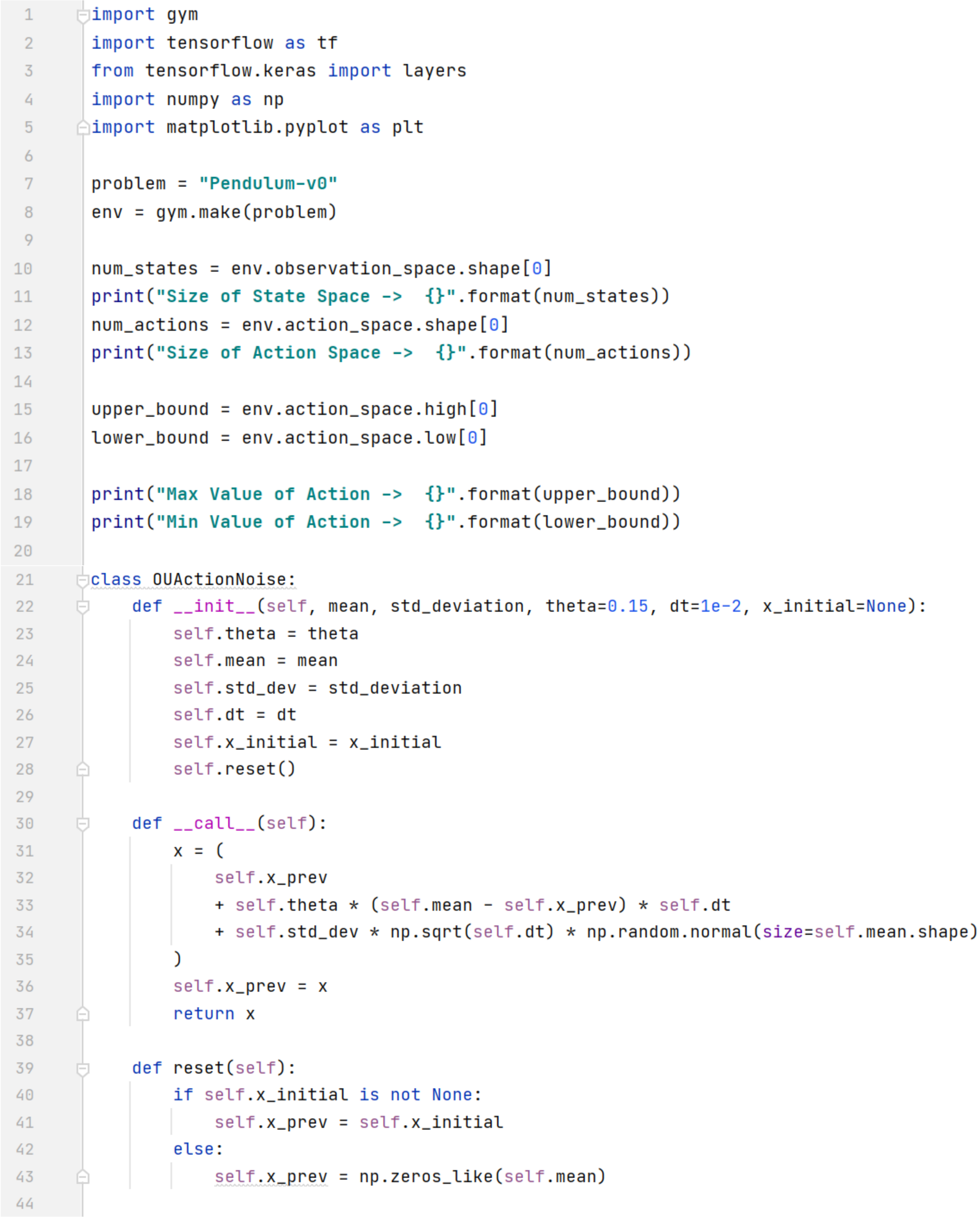
DDPG(심도 결정적 정책 그라데이션)는 연속적인 작업을 학습하기 위한 모델이 없는 오프정책 알고리즘이다.

DPG(결정적 정책 그라데이션)와 **DQN(Deep Q-Network)**의 아이디어를 결합.

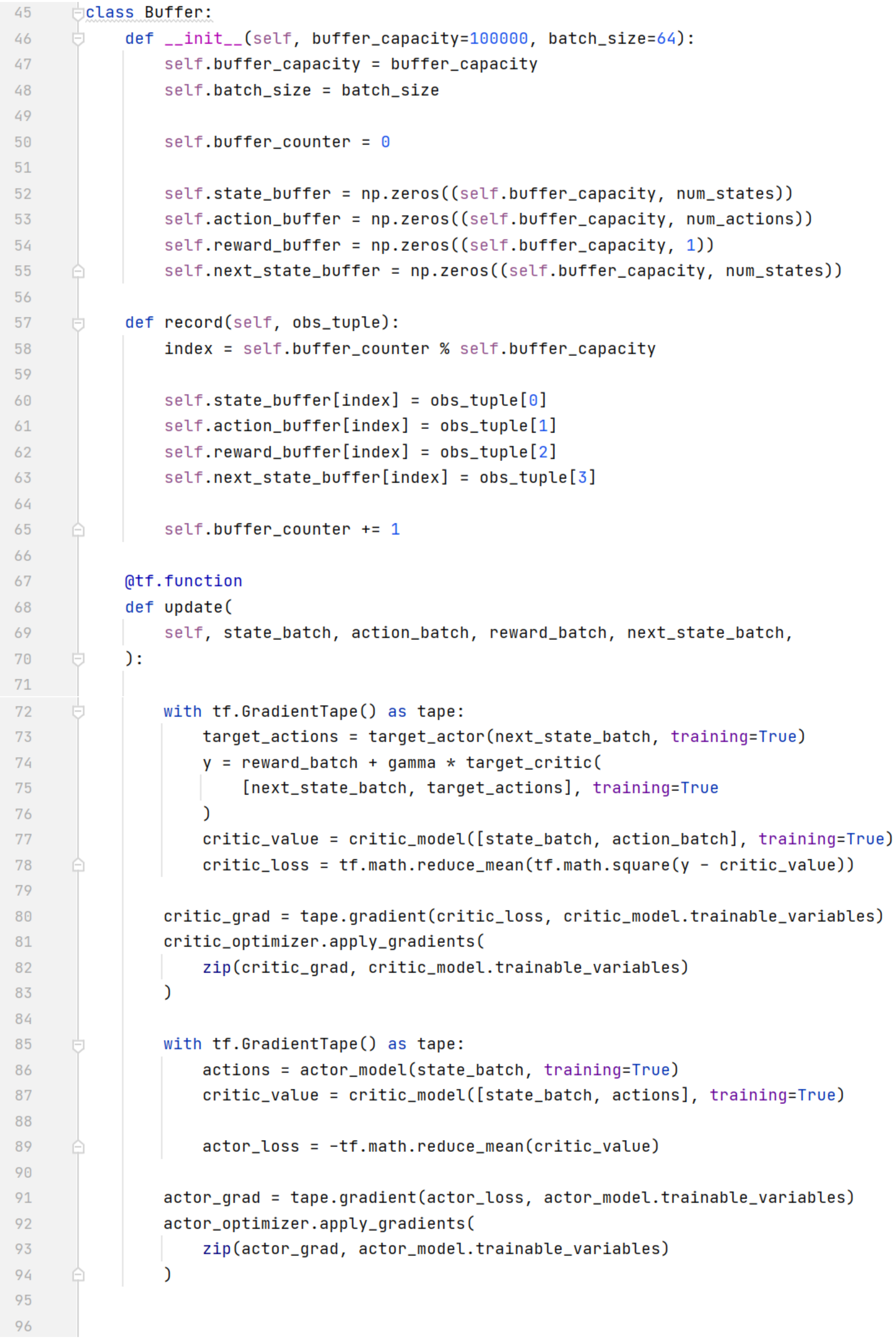
DQN 의 익스피리언스 리플레이 및 슬로우 러닝 타겟 네트워크를 사용하며, 연속 작업 공간을 통해 작동할 수 있는 DPG 를 기반.

DDPG 는 2 개의 네트워크를 보유하여 작동한다.
첫 번째 네트워크는 Q-값을 예측하는 법을 학습하고
두 번째 네트워크는 주어진 상태 작업을 선택하는 방법을 학습한다.

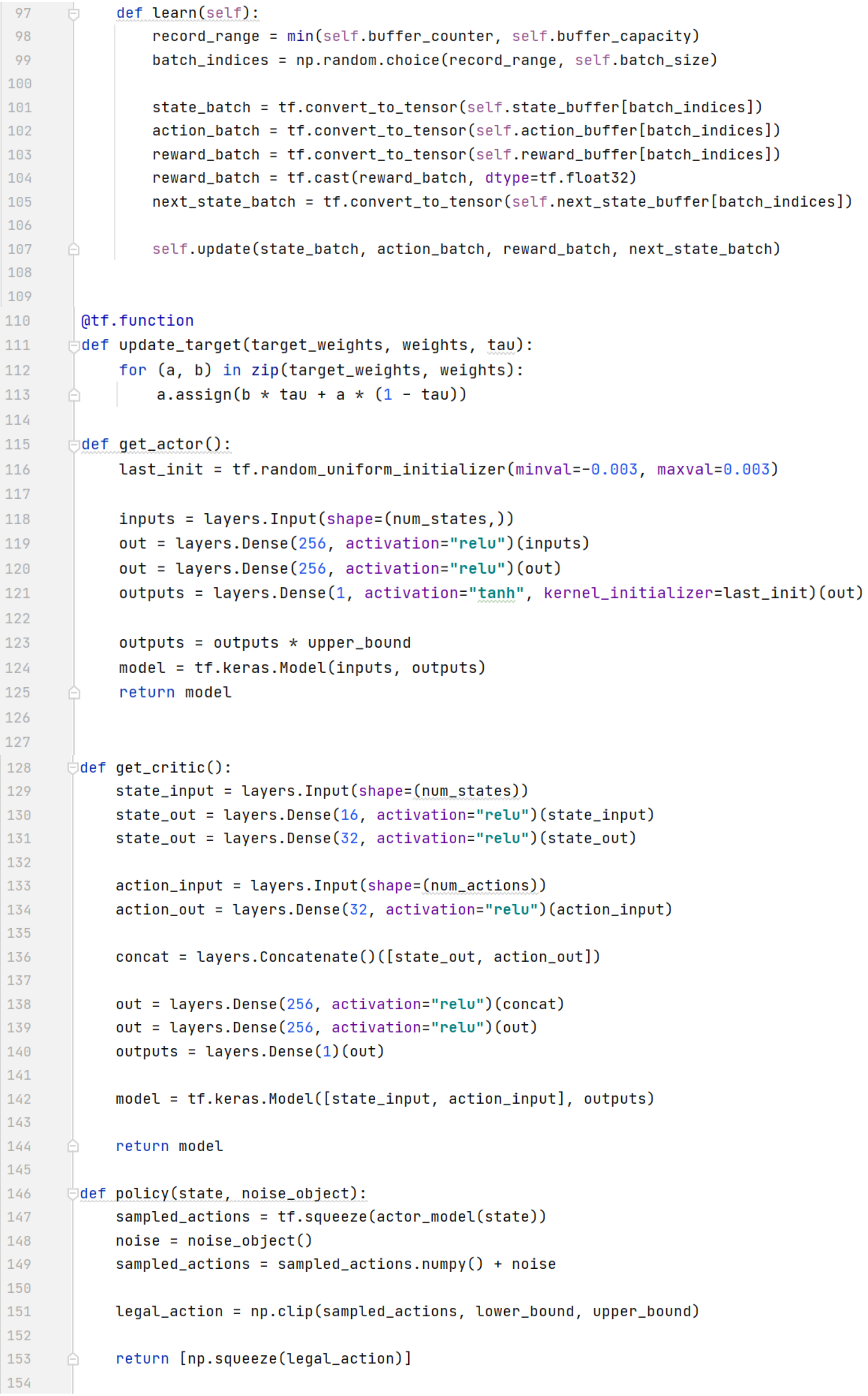
<코드>



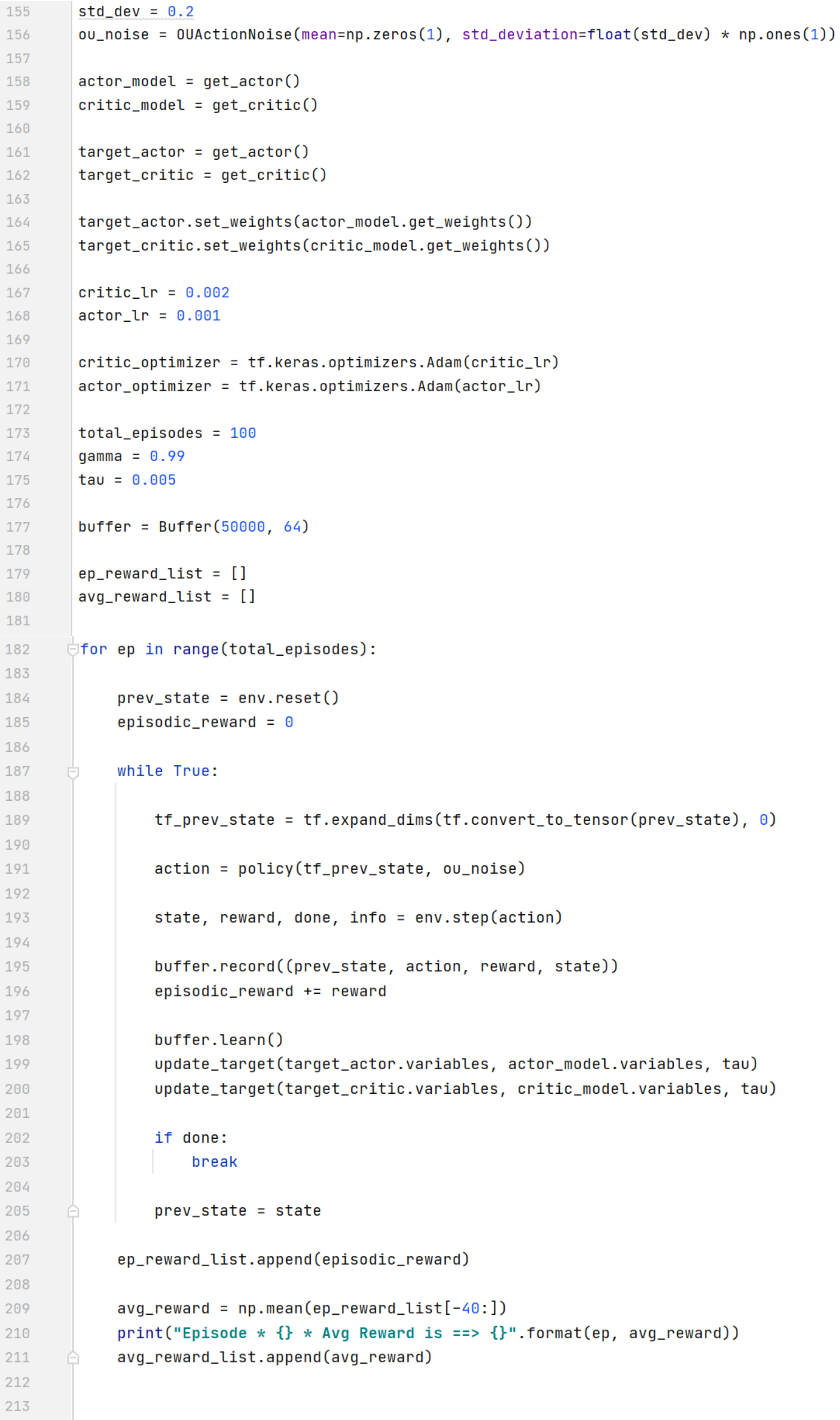
```python
import gym
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt

problem = "Pendulum-v0"
env = gym.make(problem)

num_states = env.observation_space.shape[0]
print("Size of State Space ->  {}".format(num_states))
num_actions = env.action_space.shape[0]
print("Size of Action Space ->  {}".format(num_actions))

upper_bound = env.action_space.high[0]
lower_bound = env.action_space.low[0]

print("Max Value of Action ->  {}".format(upper_bound))
print("Min Value of Action ->  {}".format(lower_bound))

class OUActionNoise:
    def __init__(self, mean, std_deviation, theta=0.15, dt=1e-2, x_initial=None):
        self.theta = theta
        self.mean = mean
        self.std_dev = std_deviation
        self.dt = dt
        self.x_initial = x_initial
        self.reset()

    def __call__(self):
        x = (
            self.x_prev
            + self.theta * (self.mean - self.x_prev) * self.dt
            + self.std_dev * np.sqrt(self.dt) * np.random.normal(size=self.mean.shape)
        )
        self.x_prev = x
        return x

    def reset(self):
        if self.x_initial is not None:
            self.x_prev = self.x_initial
        else:
            self.x_prev = np.zeros_like(self.mean)

```



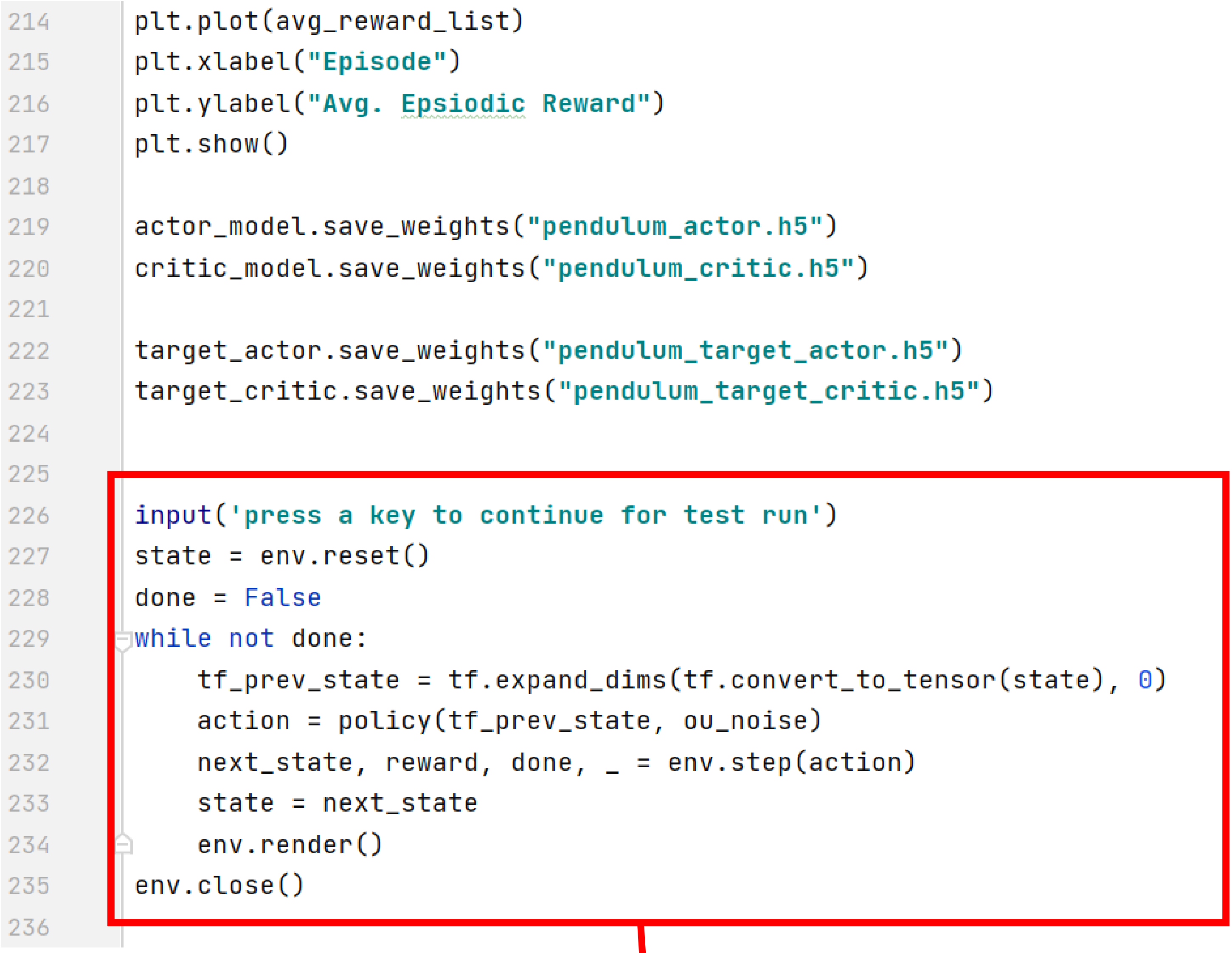
```python
class Buffer:
    def __init__(self, buffer_capacity=100000, batch_size=64):
        self.buffer_capacity = buffer_capacity
        self.batch_size = batch_size

        self.buffer_counter = 0

        self.state_buffer = np.zeros((self.buffer_capacity, num_states))
        self.action_buffer = np.zeros((self.buffer_capacity, num_actions))
        self.reward_buffer = np.zeros((self.buffer_capacity, 1))
        self.next_state_buffer = np.zeros((self.buffer_capacity, num_states))

    def record(self, obs_tuple):
        index = self.buffer_counter % self.buffer_capacity

        self.state_buffer[index] = obs_tuple[0]
        self.action_buffer[index] = obs_tuple[1]
        self.reward_buffer[index] = obs_tuple[2]
        self.next_state_buffer[index] = obs_tuple[3]

        self.buffer_counter += 1

    @tf.function
    def update(
        self, state_batch, action_batch, reward_batch, next_state_batch,
    ):

        with tf.GradientTape() as tape:
            target_actions = target_actor(next_state_batch, training=True)
            y = reward_batch + gamma * target_critic(
                [next_state_batch, target_actions], training=True
            )
            critic_value = critic_model([state_batch, action_batch], training=True)
            critic_loss = tf.math.reduce_mean(tf.math.square(y - critic_value))

        critic_grad = tape.gradient(critic_loss, critic_model.trainable_variables)
        critic_optimizer.apply_gradients(
            zip(critic_grad, critic_model.trainable_variables)
        )

        with tf.GradientTape() as tape:
            actions = actor_model(state_batch, training=True)
            critic_value = critic_model([state_batch, actions], training=True)

            actor_loss = -tf.math.reduce_mean(critic_value)

        actor_grad = tape.gradient(actor_loss, actor_model.trainable_variables)
        actor_optimizer.apply_gradients(
            zip(actor_grad, actor_model.trainable_variables)
        )

```



```
    def learn(self):
        record_range = min(self.buffer_counter, self.buffer_capacity)
        batch_indices = np.random.choice(record_range, self.batch_size)

        state_batch = tf.convert_to_tensor(self.state_buffer[batch_indices])
        action_batch = tf.convert_to_tensor(self.action_buffer[batch_indices])
        reward_batch = tf.convert_to_tensor(self.reward_buffer[batch_indices])
        reward_batch = tf.cast(reward_batch, dtype=tf.float32)
        next_state_batch = tf.convert_to_tensor(self.next_state_buffer[batch_indices])

        self.update(state_batch, action_batch, reward_batch, next_state_batch)


@tf.function
def update_target(target_weights, weights, tau):
    for (a, b) in zip(target_weights, weights):
        a.assign(b * tau + a * (1 - tau))

def get_actor():
    last_init = tf.random_uniform_initializer(minval=-0.003, maxval=0.003)

    inputs = layers.Input(shape=(num_states,))
    out = layers.Dense(256, activation="relu")(inputs)
    out = layers.Dense(256, activation="relu")(out)
    outputs = layers.Dense(1, activation="tanh", kernel_initializer=last_init)(out)

    outputs = outputs * upper_bound
    model = tf.keras.Model(inputs, outputs)
    return model


def get_critic():
    state_input = layers.Input(shape=(num_states))
    state_out = layers.Dense(16, activation="relu")(state_input)
    state_out = layers.Dense(32, activation="relu")(state_out)

    action_input = layers.Input(shape=(num_actions))
    action_out = layers.Dense(32, activation="relu")(action_input)

    concat = layers.Concatenate()([state_out, action_out])

    out = layers.Dense(256, activation="relu")(concat)
    out = layers.Dense(256, activation="relu")(out)
    outputs = layers.Dense(1)(out)

    model = tf.keras.Model([state_input, action_input], outputs)

    return model

def policy(state, noise_object):
    sampled_actions = tf.squeeze(actor_model(state))
    noise = noise_object()
    sampled_actions = sampled_actions.numpy() + noise

    legal_action = np.clip(sampled_actions, lower_bound, upper_bound)

    return [np.squeeze(legal_action)]
```



```python
std_dev = 0.2
ou_noise = OUActionNoise(mean=np.zeros(1), std_deviation=float(std_dev) * np.ones(1))

actor_model = get_actor()
critic_model = get_critic()

target_actor = get_actor()
target_critic = get_critic()

target_actor.set_weights(actor_model.get_weights())
target_critic.set_weights(critic_model.get_weights())

critic_lr = 0.002
actor_lr = 0.001

critic_optimizer = tf.keras.optimizers.Adam(critic_lr)
actor_optimizer = tf.keras.optimizers.Adam(actor_lr)

total_episodes = 100
gamma = 0.99
tau = 0.005

buffer = Buffer(50000, 64)

ep_reward_list = []
avg_reward_list = []

for ep in range(total_episodes):

    prev_state = env.reset()
    episodic_reward = 0

    while True:

        tf_prev_state = tf.expand_dims(tf.convert_to_tensor(prev_state), 0)

        action = policy(tf_prev_state, ou_noise)

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

        buffer.record((prev_state, action, reward, state))
        episodic_reward += reward

        buffer.learn()
        update_target(target_actor.variables, actor_model.variables, tau)
        update_target(target_critic.variables, critic_model.variables, tau)

        if done:
            break

        prev_state = state

    ep_reward_list.append(episodic_reward)

    avg_reward = np.mean(ep_reward_list[-40:])
    print("Episode * {} * Avg Reward is ==> {}".format(ep, avg_reward))
    avg_reward_list.append(avg_reward)


```

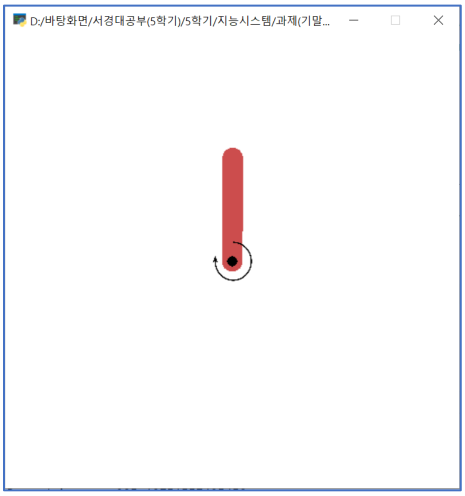


```
plt.plot(avg_reward_list)
plt.xlabel("Episode")
plt.ylabel("Avg. Epsiodic Reward")
plt.show()

actor_model.save_weights("pendulum_actor.h5")
critic_model.save_weights("pendulum_critic.h5")

target_actor.save_weights("pendulum_target_actor.h5")
target_critic.save_weights("pendulum_target_critic.h5")


input('press a key to continue for test run')
state = env.reset()
done = False
while not done:
    tf_prev_state = tf.expand_dims(tf.convert_to_tensor(state), 0)
    action = policy(tf_prev_state, ou_noise)
    next_state, reward, done, _ = env.step(action)
    state = next_state
    env.render()
env.close()
```

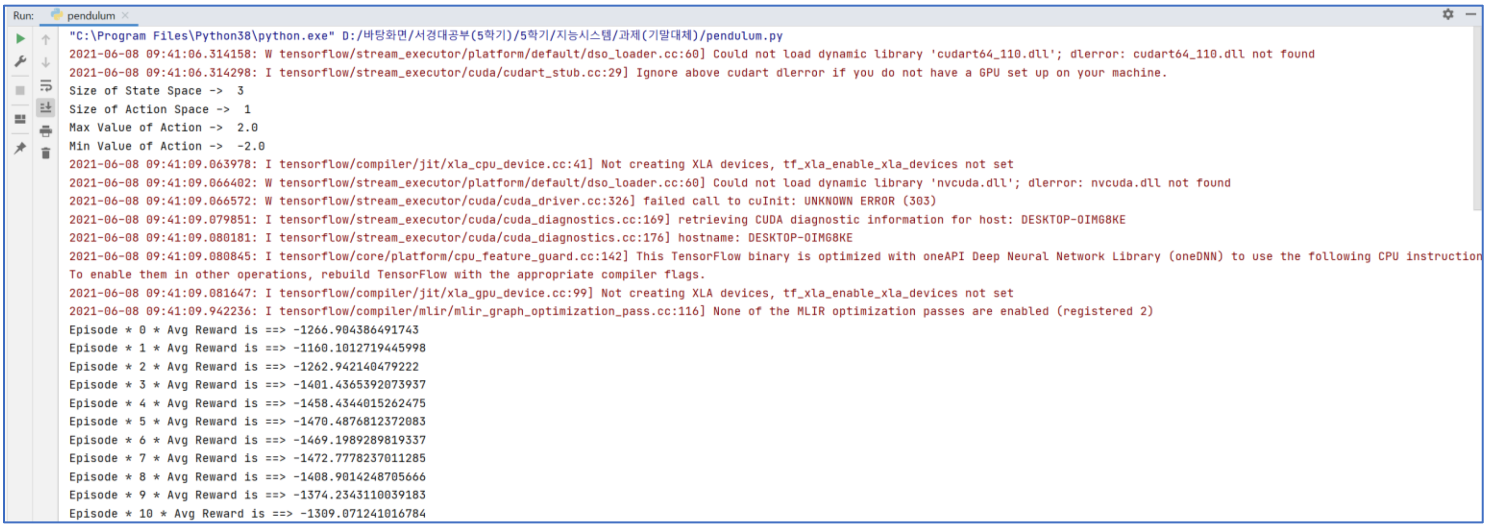
직접 추가한 코드

학습결과를 render메소드를 통해 비주얼로 표현한다.

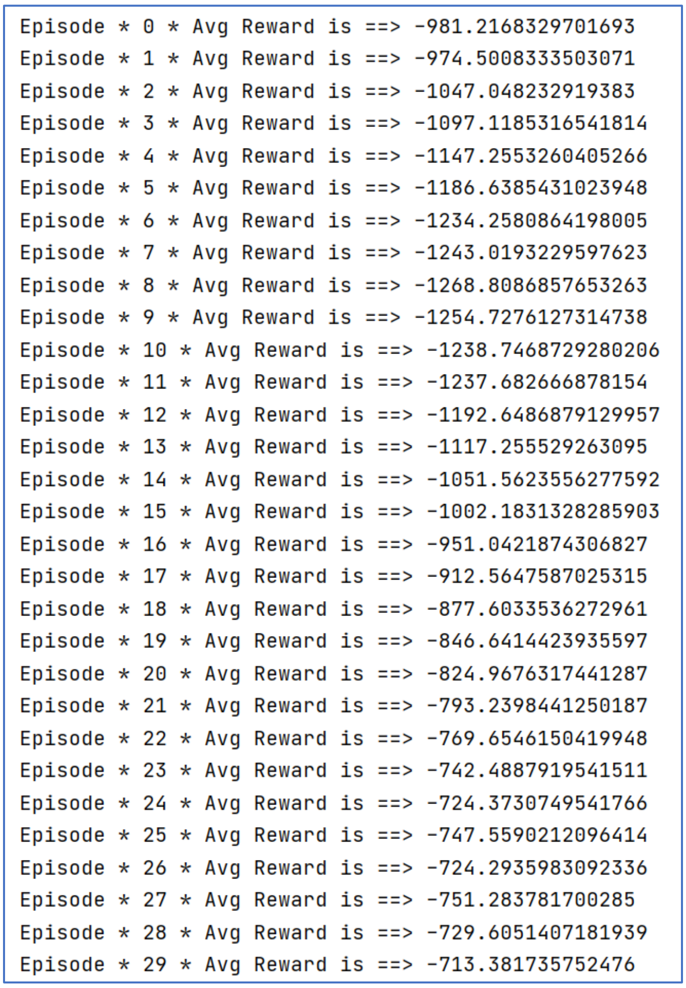
Ex)



<실행결과>

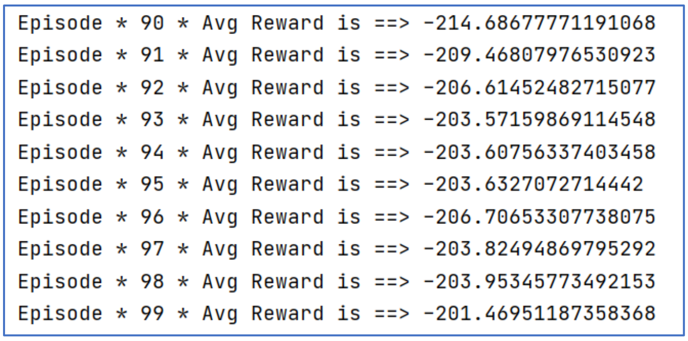


```
"C:\Program Files\Python38\python.exe" D:/바탕화면/서경대공부(5학기)/5학기/지능시스템/과제(기말대체)/pendulum.py
2021-06-08 09:41:06.314158: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2021-06-08 09:41:06.314298: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
Size of State Space ->  3
Size of Action Space ->  1
Max Value of Action ->  2.0
Min Value of Action ->  -2.0
2021-06-08 09:41:09.063978: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_enable_xla_devices not set
2021-06-08 09:41:09.066402: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic library 'nvcuda.dll'; dlerror: nvcuda.dll not found
2021-06-08 09:41:09.066572: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOWN ERROR (303)
2021-06-08 09:41:09.079851: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: DESKTOP-OIMG8KE
2021-06-08 09:41:09.080181: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: DESKTOP-OIMG8KE
2021-06-08 09:41:09.080845: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instruction
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-06-08 09:41:09.081647: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_enable_xla_devices not set
2021-06-08 09:41:09.942236: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
Episode * 0 * Avg Reward is ==> -1266.904386491743
Episode * 1 * Avg Reward is ==> -1160.1012719445998
Episode * 2 * Avg Reward is ==> -1262.942140479222
Episode * 3 * Avg Reward is ==> -1401.4365392073937
Episode * 4 * Avg Reward is ==> -1458.4344015262475
Episode * 5 * Avg Reward is ==> -1470.4876812372083
Episode * 6 * Avg Reward is ==> -1469.1989289819337
Episode * 7 * Avg Reward is ==> -1472.7778237011285
Episode * 8 * Avg Reward is ==> -1408.9014248705666
Episode * 9 * Avg Reward is ==> -1374.2343110039183
Episode * 10 * Avg Reward is ==> -1309.071241016784
```

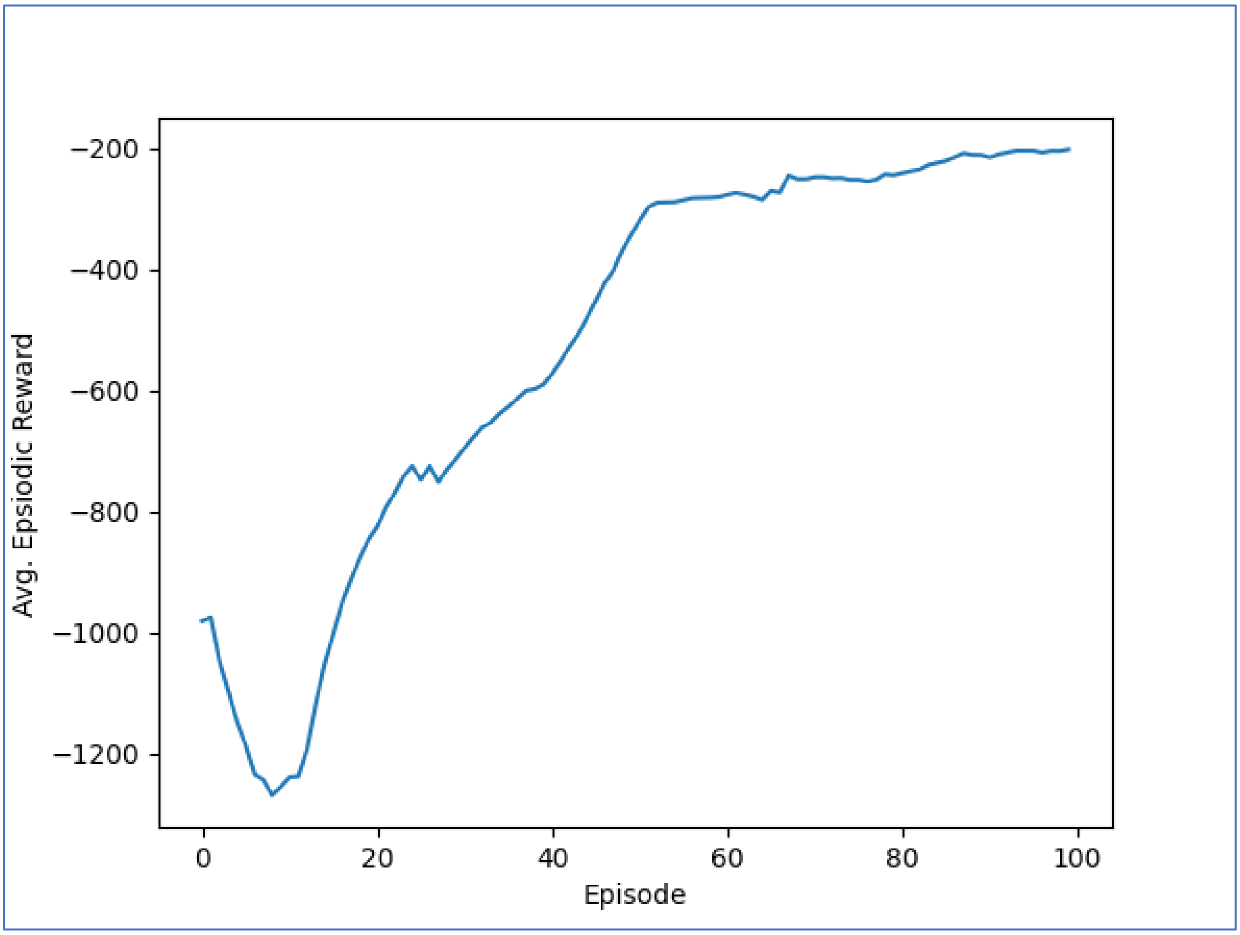


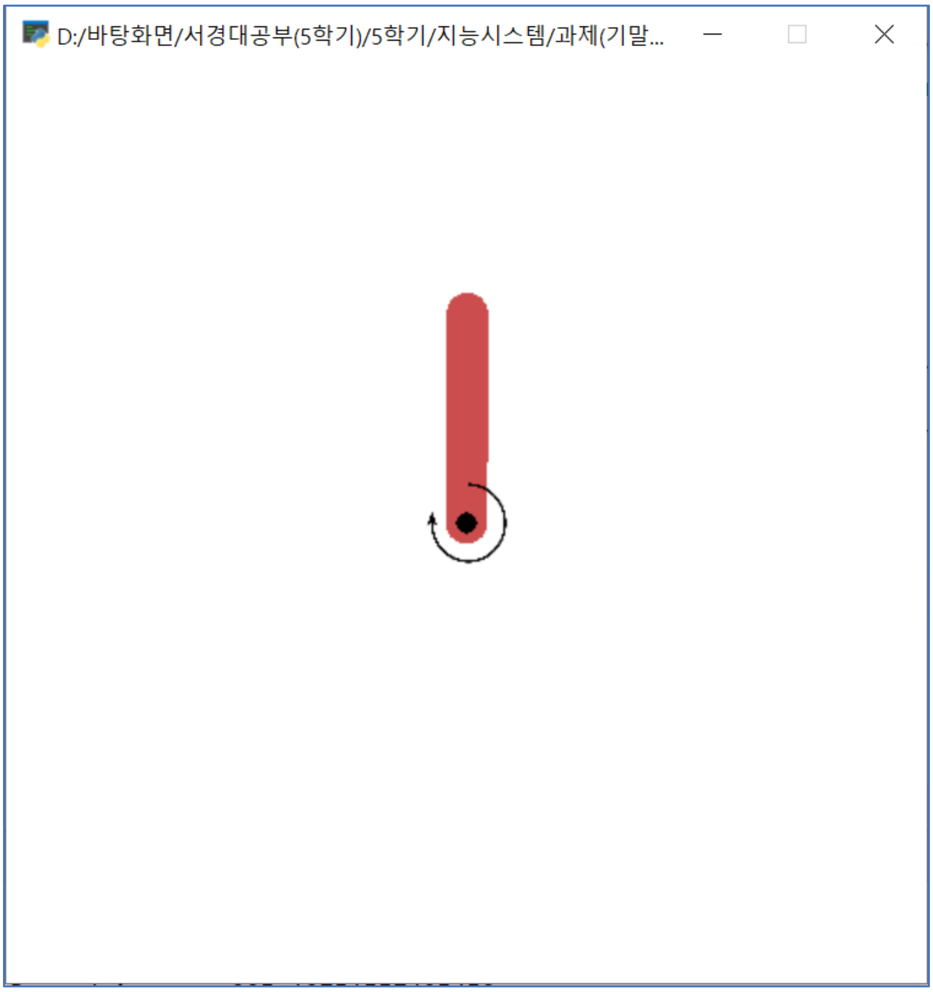
```
Episode * 0 * Avg Reward is ==> -981.2168329701693
Episode * 1 * Avg Reward is ==> -974.5008333503071
Episode * 2 * Avg Reward is ==> -1047.048232919383
Episode * 3 * Avg Reward is ==> -1097.1185316541814
Episode * 4 * Avg Reward is ==> -1147.2553260405266
Episode * 5 * Avg Reward is ==> -1186.6385431023948
Episode * 6 * Avg Reward is ==> -1234.2580864198005
Episode * 7 * Avg Reward is ==> -1243.0193229597623
Episode * 8 * Avg Reward is ==> -1268.8086857653263
Episode * 9 * Avg Reward is ==> -1254.7276127314738
Episode * 10 * Avg Reward is ==> -1238.7468729280206
Episode * 11 * Avg Reward is ==> -1237.682666878154
Episode * 12 * Avg Reward is ==> -1192.6486879129957
Episode * 13 * Avg Reward is ==> -1117.255529263095
Episode * 14 * Avg Reward is ==> -1051.5623556277592
Episode * 15 * Avg Reward is ==> -1002.1831328285903
Episode * 16 * Avg Reward is ==> -951.0421874306827
Episode * 17 * Avg Reward is ==> -912.5647587025315
Episode * 18 * Avg Reward is ==> -877.6033536272961
Episode * 19 * Avg Reward is ==> -846.6414423935597
Episode * 20 * Avg Reward is ==> -824.9676317441287
Episode * 21 * Avg Reward is ==> -793.2398441250187
Episode * 22 * Avg Reward is ==> -769.6546150419948
Episode * 23 * Avg Reward is ==> -742.4887919541511
Episode * 24 * Avg Reward is ==> -724.3730749541766
Episode * 25 * Avg Reward is ==> -747.5590212096414
Episode * 26 * Avg Reward is ==> -724.2935983092336
Episode * 27 * Avg Reward is ==> -751.283781700285
Episode * 28 * Avg Reward is ==> -729.6051407181939
Episode * 29 * Avg Reward is ==> -713.381735752476
```

(중간 생략)



```
Episode * 90 * Avg Reward is ==> -214.68677771191068
Episode * 91 * Avg Reward is ==> -209.46807976530923
Episode * 92 * Avg Reward is ==> -206.61452482715077
Episode * 93 * Avg Reward is ==> -203.57159869114548
Episode * 94 * Avg Reward is ==> -203.60756337403458
Episode * 95 * Avg Reward is ==> -203.6327072714442
Episode * 96 * Avg Reward is ==> -206.70653307738075
Episode * 97 * Avg Reward is ==> -203.82494869795292
Episode * 98 * Avg Reward is ==> -203.95345773492153
Episode * 99 * Avg Reward is ==> -201.46951187358368
```





결과를 보면 학습이 정상적으로 되어 목표하는 결과가 나왔다.