CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

cspit
Chandubhai S Patel Institute of Technology

Faculty of Technology and Engineering

Chandubhai S. Patel Institute of Technology

Date: / /

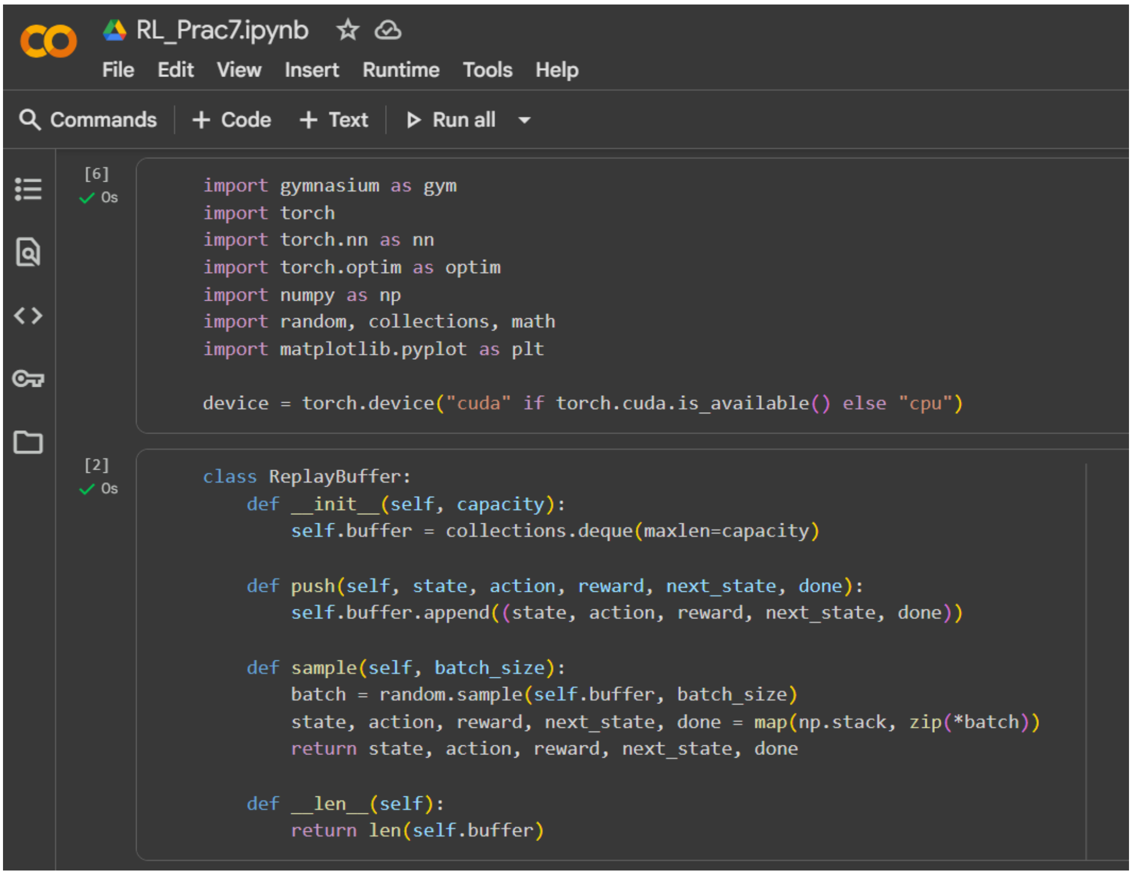
Practical Performa

Academic Year	:	2025-26	Semester	:	7th
Course code	:	OCCSE4001	Course name	:	Reinforcement Learning

Practical- No. 7

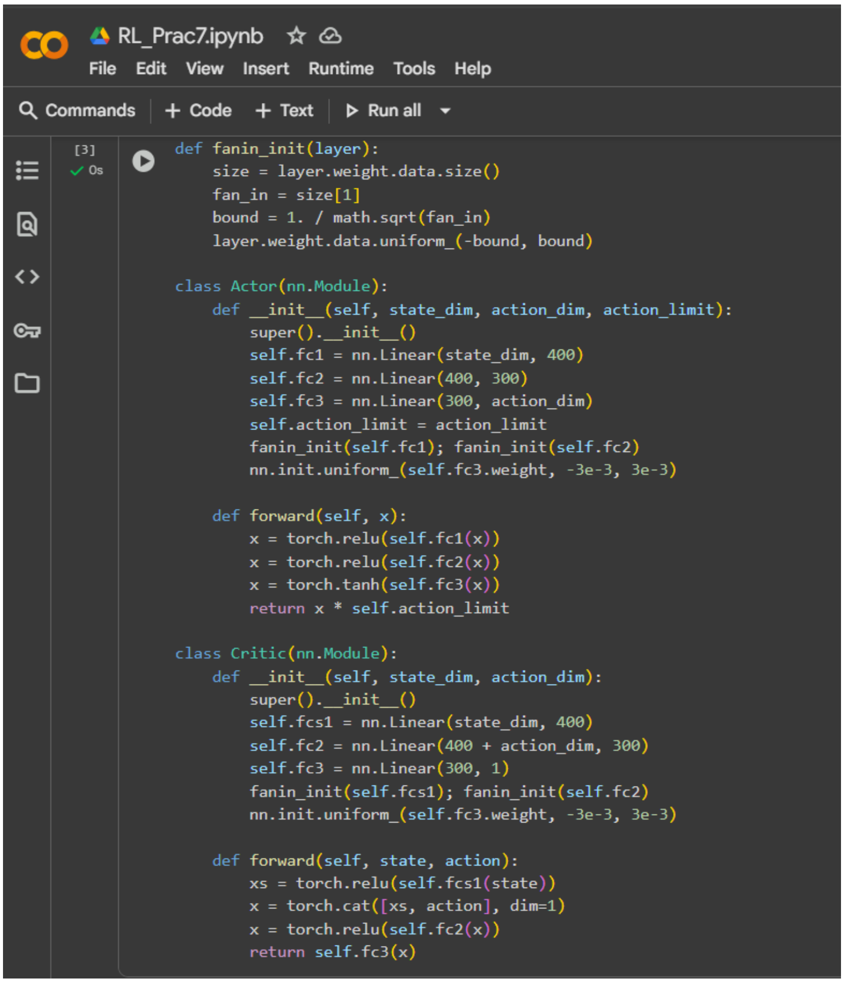
Aim: To implement and analyze the Deep Deterministic Policy Gradient (DDPG) algorithm for environments with continuous action spaces.

Code:

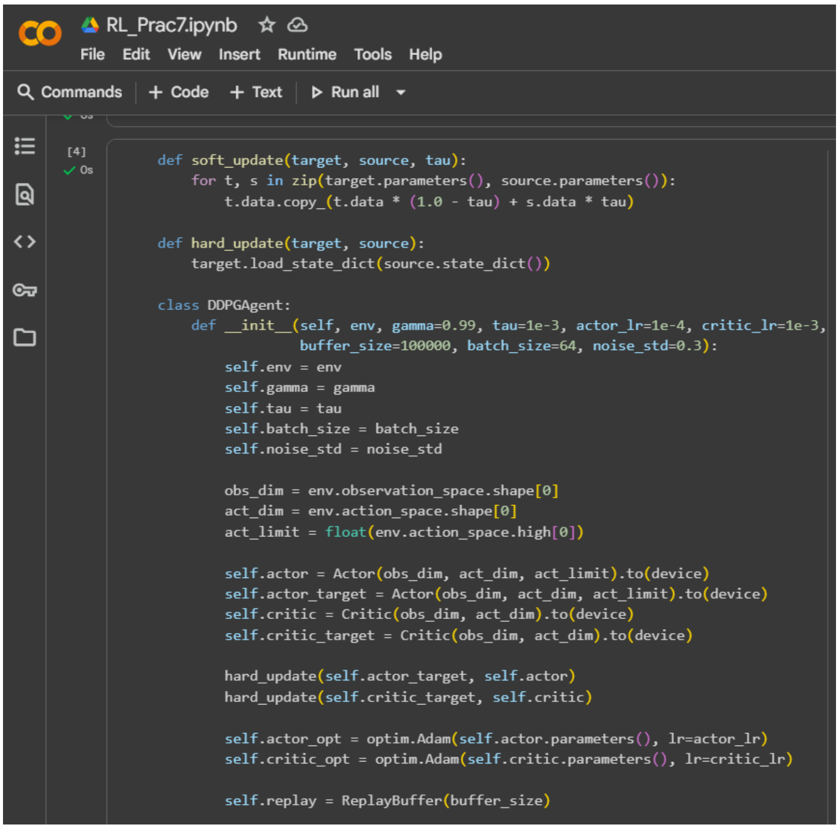

RL_Prac7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
[6] ✓ 0s
[2] ✓ 0s

```python
import gymnasium as gym
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import random, collections, math
import matplotlib.pyplot as plt

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

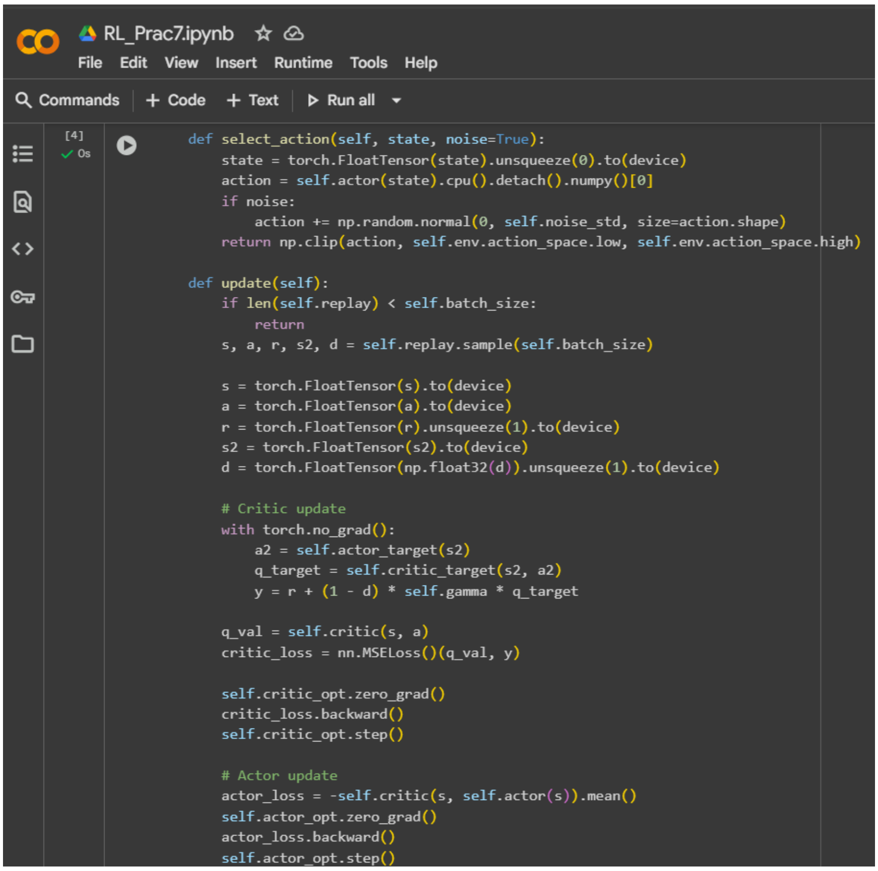
```python
class ReplayBuffer:
    def __init__(self, capacity):
        self.buffer = collections.deque(maxlen=capacity)

    def push(self, state, action, reward, next_state, done):
        self.buffer.append((state, action, reward, next_state, done))

    def sample(self, batch_size):
        batch = random.sample(self.buffer, batch_size)
        state, action, reward, next_state, done = map(np.stack, zip(*batch))
        return state, action, reward, next_state, done

    def __len__(self):
        return len(self.buffer)
```



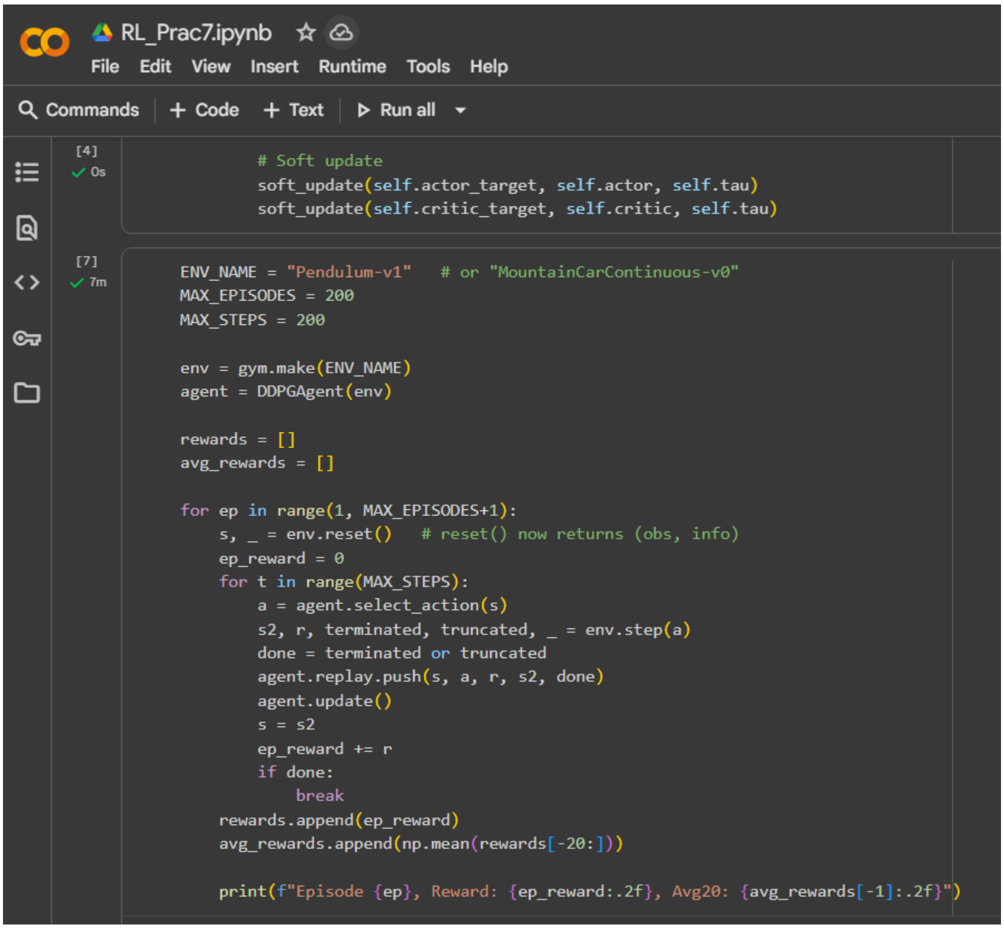
```
def fanin_init(layer):
    size = layer.weight.data.size()
    fan_in = size[1]
    bound = 1. / math.sqrt(fan_in)
    layer.weight.data.uniform_(-bound, bound)

class Actor(nn.Module):
    def __init__(self, state_dim, action_dim, action_limit):
        super().__init__()
        self.fc1 = nn.Linear(state_dim, 400)
        self.fc2 = nn.Linear(400, 300)
        self.fc3 = nn.Linear(300, action_dim)
        self.action_limit = action_limit
        fanin_init(self.fc1); fanin_init(self.fc2)
        nn.init.uniform_(self.fc3.weight, -3e-3, 3e-3)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        return x * self.action_limit

class Critic(nn.Module):
    def __init__(self, state_dim, action_dim):
        super().__init__()
        self.fcs1 = nn.Linear(state_dim, 400)
        self.fc2 = nn.Linear(400 + action_dim, 300)
        self.fc3 = nn.Linear(300, 1)
        fanin_init(self.fcs1); fanin_init(self.fc2)
        nn.init.uniform_(self.fc3.weight, -3e-3, 3e-3)

    def forward(self, state, action):
        xs = torch.relu(self.fcs1(state))
        x = torch.cat([xs, action], dim=1)
        x = torch.relu(self.fc2(x))
        return self.fc3(x)
```



```
def soft_update(target, source, tau):
    for t, s in zip(target.parameters(), source.parameters()):
        t.data.copy_(t.data * (1.0 - tau) + s.data * tau)

def hard_update(target, source):
    target.load_state_dict(source.state_dict())

class DDPGAgent:
    def __init__(self, env, gamma=0.99, tau=1e-3, actor_lr=1e-4, critic_lr=1e-3,
                 buffer_size=100000, batch_size=64, noise_std=0.3):
        self.env = env
        self.gamma = gamma
        self.tau = tau
        self.batch_size = batch_size
        self.noise_std = noise_std

        obs_dim = env.observation_space.shape[0]
        act_dim = env.action_space.shape[0]
        act_limit = float(env.action_space.high[0])

        self.actor = Actor(obs_dim, act_dim, act_limit).to(device)
        self.actor_target = Actor(obs_dim, act_dim, act_limit).to(device)
        self.critic = Critic(obs_dim, act_dim).to(device)
        self.critic_target = Critic(obs_dim, act_dim).to(device)

        hard_update(self.actor_target, self.actor)
        hard_update(self.critic_target, self.critic)

        self.actor_opt = optim.Adam(self.actor.parameters(), lr=actor_lr)
        self.critic_opt = optim.Adam(self.critic.parameters(), lr=critic_lr)

        self.replay = ReplayBuffer(buffer_size)
```

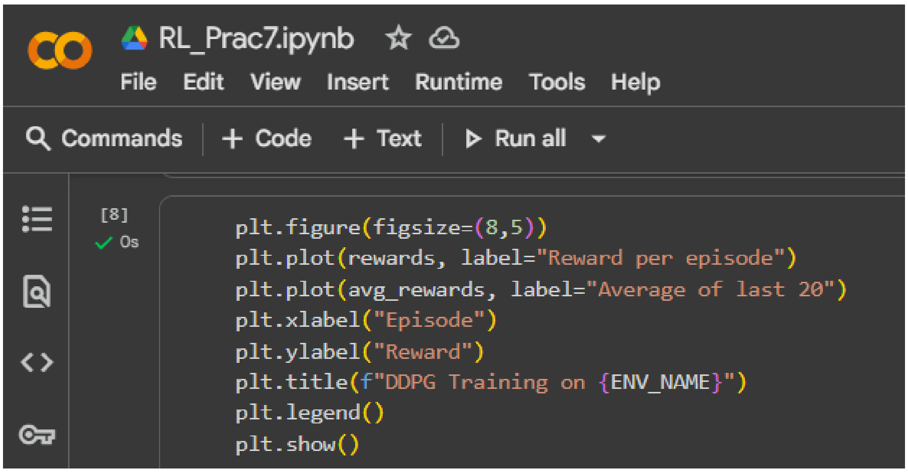


```
    def select_action(self, state, noise=True):
        state = torch.FloatTensor(state).unsqueeze(0).to(device)
        action = self.actor(state).cpu().detach().numpy()[0]
        if noise:
            action += np.random.normal(0, self.noise_std, size=action.shape)
        return np.clip(action, self.env.action_space.low, self.env.action_space.high)

    def update(self):
        if len(self.replay) < self.batch_size:
            return
        s, a, r, s2, d = self.replay.sample(self.batch_size)

        s = torch.FloatTensor(s).to(device)
        a = torch.FloatTensor(a).to(device)
        r = torch.FloatTensor(r).unsqueeze(1).to(device)
        s2 = torch.FloatTensor(s2).to(device)
        d = torch.FloatTensor(np.float32(d)).unsqueeze(1).to(device)

        # Critic update
        with torch.no_grad():
            a2 = self.actor_target(s2)
            q_target = self.critic_target(s2, a2)
            y = r + (1 - d) * self.gamma * q_target

        q_val = self.critic(s, a)
        critic_loss = nn.MSELoss()(q_val, y)

        self.critic_opt.zero_grad()
        critic_loss.backward()
        self.critic_opt.step()

        # Actor update
        actor_loss = -self.critic(s, self.actor(s)).mean()
        self.actor_opt.zero_grad()
        actor_loss.backward()
        self.actor_opt.step()
```



```
        # Soft update
        soft_update(self.actor_target, self.actor, self.tau)
        soft_update(self.critic_target, self.critic, self.tau)
```

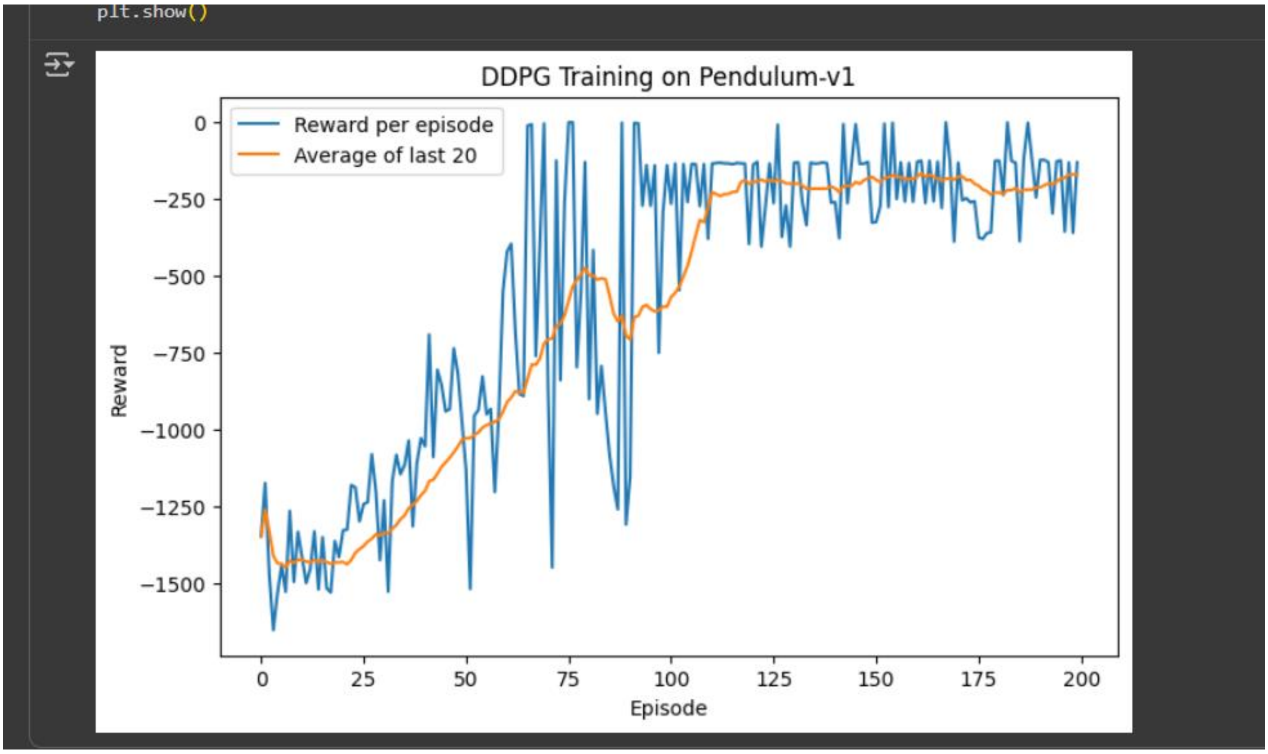
```
ENV_NAME = "Pendulum-v1"   # or "MountainCarContinuous-v0"
MAX_EPISODES = 200
MAX_STEPS = 200

env = gym.make(ENV_NAME)
agent = DDPGAgent(env)

rewards = []
avg_rewards = []

for ep in range(1, MAX_EPISODES+1):
    s, _ = env.reset()   # reset() now returns (obs, info)
    ep_reward = 0
    for t in range(MAX_STEPS):
        a = agent.select_action(s)
        s2, r, terminated, truncated, _ = env.step(a)
        done = terminated or truncated
        agent.replay.push(s, a, r, s2, done)
        agent.update()
        s = s2
        ep_reward += r
        if done:
            break
    rewards.append(ep_reward)
    avg_rewards.append(np.mean(rewards[-20:]))

    print(f"Episode {ep}, Reward: {ep_reward:.2f}, Avg20: {avg_rewards[-1]:.2f}")
```



RL_Prac7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[8]
0s

```
plt.figure(figsize=(8,5))
plt.plot(rewards, label="Reward per episode")
plt.plot(avg_rewards, label="Average of last 20")
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.title(f"DDPG Training on {ENV_NAME}")
plt.legend()
plt.show()
```

Output:



```
plt.show()
```

Grade/Marks **Sign of Lab Teacher with Date**

(_____ / 10)