Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

«Методы машинного обучения в автоматизированных системах обработки информации и управления»

Лабораторная работа №7

«Алгоритмы Actor-Critic»

ИСПОЛНИТЕЛЬ:

Васильев Д.А.
Группа ИУ5-21М
____

"__"___________2023 г.

Москва 2023

1. Задание

- Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

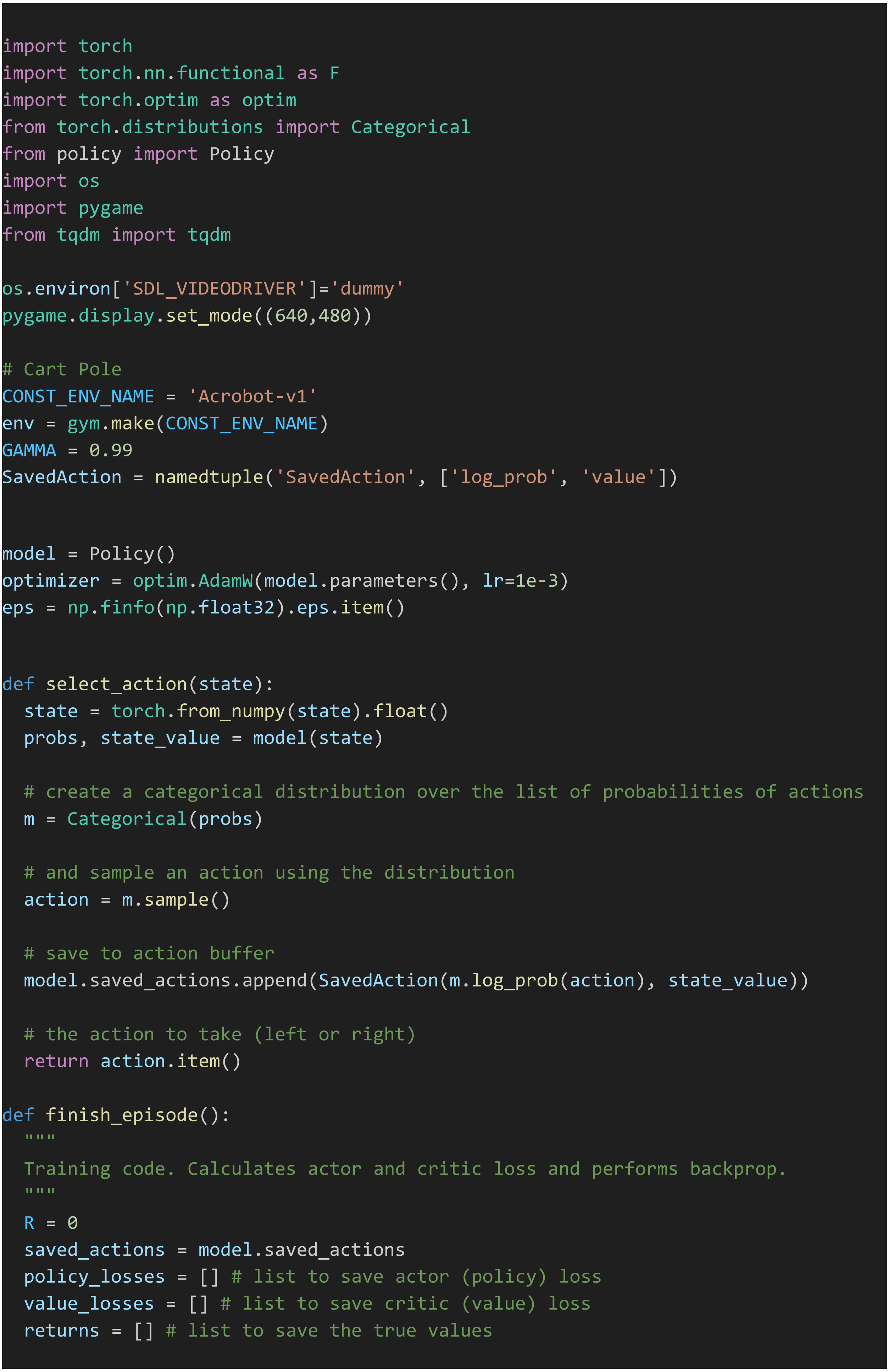
2. Листинг

2.1. policy.py

```
import torch.nn as nn
import torch.nn.functional as F

class Policy(nn.Module):
  def __init__(self):
    super(Policy, self).__init__()
    self.affine1 = nn.Linear(6, 128)

    # actor's layer
    self.action_head = nn.Linear(128, 3)

    # critic's layer
    self.value_head = nn.Linear(128, 1)

    # action & reward buffer
    self.saved_actions = []
    self.rewards = []

  def forward(self, x):
    x = F.relu(self.affine1(x))

    # actor: choses action to take from state s_t
    # by returning probability of each action
    action_prob = F.softmax(self.action_head(x), dim=-1)

    # critic: evaluates being in the state s_t
    state_values = self.value_head(x)

    # return values for both actor and critic as a tuple of 2 values:
    # 1. a list with the probability of each action over the action space
    # 2. the value from state s_t
    return action_prob, state_values
```

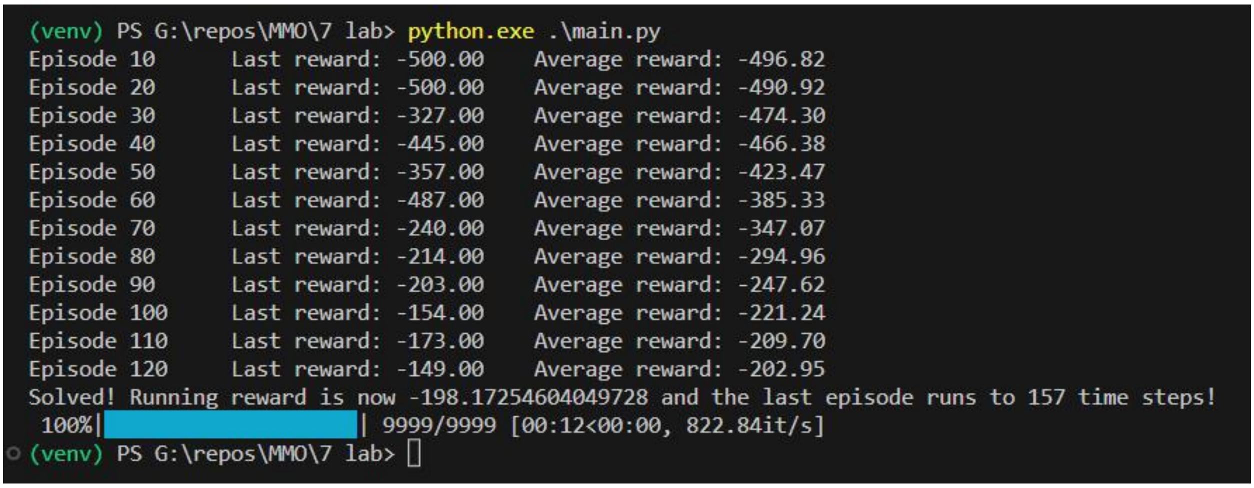
2.2. main.py

```
import gymnasium as gym
import numpy as np
from itertools import count
from collections import namedtuple
```



```
import torch
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Categorical
from policy import Policy
import os
import pygame
from tqdm import tqdm

os.environ['SDL_VIDEODRIVER']='dummy'
pygame.display.set_mode((640,480))

# Cart Pole
CONST_ENV_NAME = 'Acrobot-v1'
env = gym.make(CONST_ENV_NAME)
GAMMA = 0.99
SavedAction = namedtuple('SavedAction', ['log_prob', 'value'])


model = Policy()
optimizer = optim.AdamW(model.parameters(), lr=1e-3)
eps = np.finfo(np.float32).eps.item()


def select_action(state):
  state = torch.from_numpy(state).float()
  probs, state_value = model(state)

  # create a categorical distribution over the list of probabilities of actions
  m = Categorical(probs)

  # and sample an action using the distribution
  action = m.sample()

  # save to action buffer
  model.saved_actions.append(SavedAction(m.log_prob(action), state_value))

  # the action to take (left or right)
  return action.item()

def finish_episode():
  """
  Training code. Calculates actor and critic loss and performs backprop.
  """
  R = 0
  saved_actions = model.saved_actions
  policy_losses = [] # list to save actor (policy) loss
  value_losses = [] # list to save critic (value) loss
  returns = [] # list to save the true values
  # calculate the true value using rewards returned from the environment
  for r in model.rewards[::-1]:
    # calculate the discounted value
    R = r + GAMMA * R
    returns.insert(0, R)

  returns = torch.tensor(returns)
  returns = (returns - returns.mean()) / (returns.std() + eps)

  for (log_prob, value), R in zip(saved_actions, returns):
    advantage = R - value.item()

    # calculate actor (policy) loss
    policy_losses.append(-log_prob * advantage)

    # calculate critic (value) loss using L1 smooth loss
    value_losses.append(F.smooth_l1_loss(value, torch.tensor([R])))

  # reset gradients
  optimizer.zero_grad()

  # sum up all the values of policy_losses and value_losses
  loss = torch.stack(policy_losses).sum() + torch.stack(value_losses).sum()

  # perform backprop
  loss.backward()
  optimizer.step()

  # reset rewards and action buffer
  del model.rewards[:]
  del model.saved_actions[:]

def main():
    running_reward = -500

    # run infinitely many episodes
    for i_episode in count(1):
        # print(running_reward)
        # reset environment and episode reward
        state, _ = env.reset()
        ep_reward = 0
        # for each episode, only run 9999 steps so that we don't
        # infinite loop while learning
        for t in range(1, 9999):
            # select action from policy
            action = select_action(state)
            # take the action
            state, reward, done, truncated, _ = env.step(action)
            model.rewards.append(reward)
            ep_reward += reward
            if done or truncated:
                break
        # print(ep_reward)
        # update cumulative reward
        running_reward = 0.05 * ep_reward + (1 - 0.05) * running_reward
        # perform backprop
        finish_episode()
        # log results
        if i_episode % 10 == 0:
            print(f"Episode {i_episode}\tLast reward: {ep_reward:.2f}\tAverage
reward: {running_reward:.2f}")
        # check if we have "solved" the cart pole problem
        if running_reward > env.spec.reward_threshold * 2:
            print(f"Solved! Running reward is now {running_reward} and the last
episode runs to {t} time steps!")
            break
    env2 = gym.make(CONST_ENV_NAME, render_mode='human')
    # reset environment and episode reward
    state, _ = env2.reset()
    ep_reward = 0
    # for each episode, only run 9999 steps so that we don't
    # infinite loop while learning
    bar = tqdm(range(1, 10000), bar_format=' {l_bar}{bar:20}{r_bar}{bar:-10b}',
colour='CYAN')
    for t in bar:
        # select action from policy
        action = select_action(state)
        # take the action
        state, reward, done, _, _ = env2.step(action)
        model.rewards.append(reward)
        ep_reward += reward
        if done:
            bar.update(10000-t)
            bar.refresh()
            bar.close()
            break
if __name__ == '__main__':
    main()
```

3. Экранные формы



```
(venv) PS G:\repos\MMO\7 lab> python.exe .\main.py
Episode 10      Last reward: -500.00    Average reward: -496.82
Episode 20      Last reward: -500.00    Average reward: -490.92
Episode 30      Last reward: -327.00    Average reward: -474.30
Episode 40      Last reward: -445.00    Average reward: -466.38
Episode 50      Last reward: -357.00    Average reward: -423.47
Episode 60      Last reward: -487.00    Average reward: -385.33
Episode 70      Last reward: -240.00    Average reward: -347.07
Episode 80      Last reward: -214.00    Average reward: -294.96
Episode 90      Last reward: -203.00    Average reward: -247.62
Episode 100     Last reward: -154.00    Average reward: -221.24
Episode 110     Last reward: -173.00    Average reward: -209.70
Episode 120     Last reward: -149.00    Average reward: -202.95
Solved! Running reward is now -198.17254604049728 and the last episode runs to 157 time steps!
 100%|████████████████████| 9999/9999 [00:12<00:00, 822.84it/s]
○ (venv) PS G:\repos\MMO\7 lab> 
```