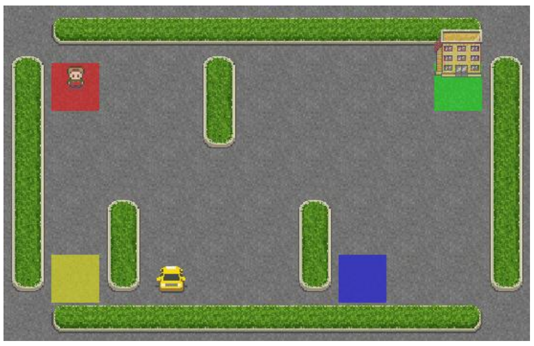
Project2

1. DQN (Deep Q-Network)

The **Taxi** environment is a grid-based problem with the goal of moving a taxi to pick up and drop off a passenger at designated locations. There are four designated locations: Red, Green, Yellow, and Blue, in a 5x5 grid world. The taxi starts at a random square, and the passenger starts at one of the designated locations. The goal is to move the taxi to the passenger's location, pick up the passenger, move to the passenger's desired destination, and drop off the passenger. Once the passenger is dropped off, the episode ends. When illegally executing "pickup" and "drop-off" actions, reward is -10. And when the task is completed, you will receive +20 reward. In addition, the reward for each step is -1.
(You can learn basic information about the environment through the gym documentation:
https://gymnasium.farama.org/environments/toy_text/taxi/)



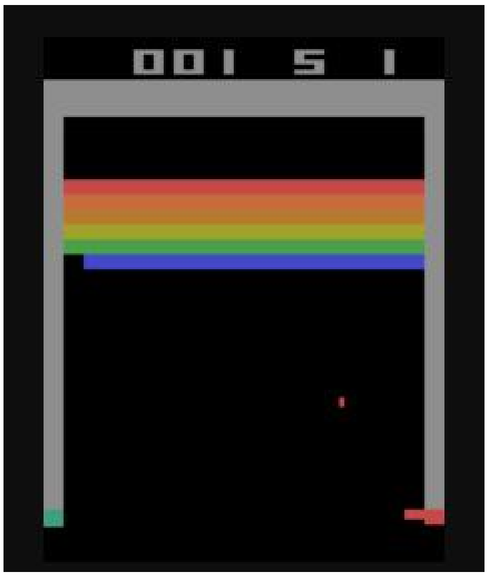
In this environment, the state space will be relatively large (about a few hundred), and if Q-learing algorithm is used to record the q value in each state, it will take up a lot of space. For this situation, we need to use a function fitting method to estimate the Q value, that is, treating this complex Q value table as data and using a parameterized function to fit these data. Obviously, this method of function fitting suffers from accuracy loss, so it is called an approximation method. The DQN algorithm you are about to implement can be used to solve discrete action problems in large (or continuous) state spaces.

Q1: Implement the DQN algorithm on the Taxi environment in *dqn.ipynb*
The deep neural network structure and ReplayBuffer(in rl_utils.py) have been provided, please do not modify them and use directly. Your main task is to complete the basic update iteration process in the DQN algorithm, noting that the target network needs to be updated after a specified number of steps.

2. PPO (Proximal Policy Optimization)

The Arcade Learning Environment (ALE), commonly referred to as Atari, is a framework that allows researchers and hobbyists to develop AI agents for Atari 2600 roms. Its built on top of the Atari 2600 emulator Stella and separates the details of emulation from agent design. You can interact with the games through the Gym API.
(Please refer to this website to learn how to use ALE in Gym:
https://ale.farama.org/gymnasium-interface)

This project uses the game **Breakout** from ALE. The dynamics are similar to pong: You move a paddle and hit the ball in a brick wall at the top of the screen. Your goal is to destroy the brick wall. You score points by destroying bricks in the wall. You can try to break through the wall and let the ball wreak havoc on the other side, all on its own! You have five lives.
(You can learn basic information about the environment through the ALE documentation:
https://ale.farama.org/environments/breakout/)



The PPO algorithm is based on the Actor-Critic framework. The Actor-Critic algorithm may encounter unstable training situations, such as when the policy module is a Deep Neural Network, updating parameters along the policy gradient may result in a sudden deterioration of the policy due to a long step size, thereby affecting the training effectiveness. In response to the above issues, we consider finding a trust region during the update process, where updating policies can provide security guarantees for certain policy performance. This is called TRPO. PPO is an improvement based on TRPO, with simpler algorithm implementation and better performance.

PPO comes in two forms: PPO-Penalty and PPO-Clip. PPO-Penalty incorporates the constraint of KL divergence into the objective function; PPO-Clip directly imposes restrictions in the objective function to ensure that the difference between the new and old parameters is not too large. **In this project, you only need to implement PPO-Clip.**

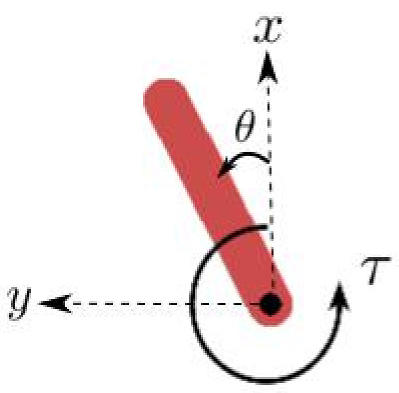
```
Algorithm 1 PPO-Clip
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, ...$ do
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
       on the current value function $V_{\phi_k}$.
6:     Update the policy by maximizing the PPO-Clip objective:
       $$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$
       typically via stochastic gradient ascent with Adam.
7:     Fit value function by regression on mean-squared error:
       $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$
       typically via some gradient descent algorithm.
8: end for
```

Q2: Implement the PPO(Clip) algorithm on the Breakout environment in *ppo.ipynb*

The deep neural network structure of actor and critic have been provided, please do not modify them and use directly(To accelerate training convergence, actor and critic share the same network). Your main task is to complete the basic update iteration process in the PPO algorithm with PPO-Clip method.

3. DDPG (Deep Deterministic Policy Gradient)

The **pendulum** is part of the Classic Control environments in gym. The inverted pendulum swingup problem is based on the classic problem in control theory. The system consists of a pendulum attached at one end to a fixed point, and the other end being free. The pendulum starts in a random position and the goal is to apply torque on the free end to swing it into an upright position, with its center of gravity right above the fixed point. The action is the torque applied to free end of the pendulum, the state is the x-y coordinates of the pendulum's free end and its angular velocity. The reward is Non-Positive value, the closer the pendulum is to an upright position and the smaller the angular velocity and torque, the greater the reward (maximum is 0).
(You can learn basic information about the environment through the gym documentation: *https://www.gymlibrary.dev/environments/classic_control/pendulum/*)



Since PPO is an on-policy algorithm, its sample efficiency is relatively low. DQN directly estimates the optimal function Q and can achieve off-policy learning, but it can only handle environments with finite action space, because it needs to select an action with the largest Q value from all actions. The DDPG algorithm constructs a deterministic policy and uses the gradient ascent method to maximize the Q value, which can handle environments with infinite action space.

Algorithm 1 Deep Deterministic Policy Gradient

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
3: repeat
4:     Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5:     Execute $a$ in the environment
6:     Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:     Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:     If $s'$ is terminal, reset environment state.
9:     if it's time to update then
10:        for however many updates do
11:            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:            Compute targets
                   $y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$
13:            Update Q-function by one step of gradient descent using
                   $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$
14:            Update policy by one step of gradient ascent using
                   $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$
15:            Update target networks with
                   $\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi$
                   $\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$
16:        end for
17:    end if
18: until convergence
```

Q3: Implement the DDPG algorithm on the pendulum environment in *ddpg.ipynb*

The deep neural network structure of PolicyNet and QValueNet have been provided, please do not modify them and use directly. Note that multiple environments will be run in parallel. Your main task is to complete the basic update iteration process in the DDPG algorithm.