華中科技大學

2024

人工智能导论实验报告

专　　业：计算机科学与技术

班　　级：计卓----

学　　号：----------

姓　　名：lcw

电　　话：----------

邮　　件：----------@hust.edu.cn

完成日期：2025 年 1 月 15 日

摘要：强化学习作为人工智能的重要方法，已经在诸如游戏智能体、机器人控制、自然语言处理等领域取得了重大成果。本文通过分析强化学习中的两大算法——深度 Q 网络（DQN）和近端策略优化（PPO），并搭建相应模型用其训练 agent 游玩贪吃蛇游戏，分析两大算法对应 agent 的表现来对比两大算法的特点。

As an important method of artificial intelligence, reinforcement learning has achieved significant results in fields such as game agents, robot control, and natural language processing. This paper analyzes the two major algorithms in reinforcement learning, Deep Q Network (DQN) and Proximal Policy Optimization (PPO), builds the corresponding network and uses it to train the agent to play the game of Snake, and analyzes the performance of the corresponding agent of the two algorithms to compare the characteristics of the two algorithms.

关键词：深度强化学习、DQN、PPO、贪吃蛇游戏、智能体训练。

目　录

1　DQN 算法和 PPO 算法的介绍

1.1 DQN 算法

1.1.1背景

DQN(Deep-Q-Network)算法由 DeepMind 团队于 2013 年在其发表的论文《Playing Atari with Deep Reinforcement Learning》中首次提出，论文中使用 DQN 算法训练了能够根据纯视频图像输入游玩 Atari 系列游戏的 Agent，并且训练出的 Agent 水平普遍高于人类。DQN 算法的提出是强化学习领域的一个重要进展，它将深度学习网络与 Q-learning 结合起来，使得 Agent 能够在更复杂的环境中自主学习并不断优化自己的决策。

1.1.2算法原理

1.1.2.1 Q 值函数与深度神经网络近似

Q 值函数，又称状态-动作价值函数，即$Q(s,a)$，它表示在状态 s 下采取动作 a 时 Agent 能够获得的回报(Reward)的期望，其具体定义为：

$$Q(s,a) = E[R_t|S_t = s, A_t = a]$$

其中R_t表示在状态s下采取动作a后到未来很长一段时间的累加奖励

在传统的 Q-learning 中，Q 值函数 Q(s,a) 被存储在一个查找表中。其中，行代表不同的状态 s，列代表不同的动作 a，这样便建立起了每一对状态和动作与相应 Q 值间的映射关系。但是传统 Q-learning 的缺点是很明显的，即其只能适用于离散且相对较小的状态和动作空间，因此在面对一些状态连续或状态和动作对多且复杂问题时，传统的 Q 表方法就会面临维度爆炸和无法收敛等问题。于是，研究者们开始将深度学习引入到 Q-learning 中，其核心思想是通过深度神经网络来对 Q 值函数进行近似而非直接存储，使得 Q-learning 能够处理各种离散或连续的输入组合，从而克服了传统 Q-learning 在处理高维状态空间和动作空间时的局限性。

引进深度神经网络后的Q-learning被称为DQN，Q值函数重新表示为$Q(s,a;\theta)$，其中θ是神经网络的可学习参数。DQN 网络通常包含以下三层结构：

- **输入层**：接收环境状态的特征，其输入可以是环境的画面帧，也可以是经过特征提取后的环境特征数组，通常使用卷积神经网络对输入的画面帧进行高维特征提取。
- **隐藏层**：通常使用一个或多个隐藏层，这些层会通过非线性激活函数处理

(如 ReLU 或 tanh 等),从而提取出更高维度的特征。

- **输出层**：输出层的输出节点数量通常为动作空间的大小，每个节点的输出对应于该动作的 Q 值，通常使用若干个全连接层得到最终的输出。

1.1.3核心机制

1.1.3.1 经验回放

经验回放(Experience Replay)是 DQN 算法中的一个核心机制。其基本思想是建立一个足够大的回放缓冲区，然后将 Agent 与环境交互过程中收集的经历存储到其中，从而允许在训练模型时使用随机采样或优先经验回放等机制灵活地利用已有的经历来训练 Q 网络。

经验回放池通常是一个双端队列，队列中存储的基本元素是一个四元组：

$$(s, a, r, s') \ \textit{其中：}$$

s: *当前状态*　　a: *智能体采取的动作*

r: *s 中采取动作a获得的奖励*　　s': *状态 s 下执行动作 a 后到达的新状态*

经验回放机制对于 Q 网络的训练的平稳性和收敛速率有很大帮助，一是经验回放池中的四元组全面的描述了 Agent 曾经所处的状态，Agent 可以在此后若个 Epoch 中既学习新的经验，也能反复回顾以往的经验，做到“温故而知新”；二是经验回放池在随机采样和优先回放的加持下很大幅度的降低了训练样本间的相关性，确保了每次训练时所使用的样本来自不同的时间步和场景，这样既能提高模型的泛化能力，避免神经网络对近期的经历过拟合，也能降低训练时反向传播的损失的方差，从而加速模型的收敛并提高训练的稳定性。

1.1.3.2 目标网络

目标网络也是 DQN 的一个重要核心机制，它的引入为 DQN 模型训练的平稳性和收敛性提供了保障。已知 DQN 中需要一个主网络来根据根据环境特征和动作的输入给出相应的 Q 值函数，此主网络承担了动作选择和反向传播更新模型参数任务。在对主网络进行训练时，根据贝尔曼方程，目标 Q 值不仅取决于即时奖励 r，还取决于下一个状态的折扣价值，即：

$$y_j = r_j + \gamma \cdot max_{a'} Q(s', a')$$

如果上式的目标 Q 值通过当前的主 Q 网络$Q(s, a; \theta)$计算，由于主 Q 网络更新频率较快，就会使得对未来的估计波动较大，这就加大了训练过程的方差，使

得模型在训练时出现较大的震荡，影响收敛速度。于是引入目标网络$Q(s,a;\theta^-)$(*结构与主Q网络相同，θ^-为目标Q网络的参数*)，旨在为 DQN 模型提供稳定的估计，从而改善训练过程的性能。上式变为：

$$y_j = r_j + \gamma \cdot max_{a'} Q(s', a'; \theta^-)$$

只需要设置好目标网络的更新间隔，每次达到更新间隔后将主 Q 网络权重参数赋给目标网络，就能就能有效保障 DQN 训练的稳定性与收敛性。

1.2 PPO 算法

1.2.1背景

近端策略优化算法（PPO,Proximal Policy Optimization）算法由 OpenAI 团队于 2017 年在其论文《Proximal Policy Optimization Algorithms》中首次提出。PPO 算法是一种基于策略优化的方法，用于解决强化学习中的连续和离散控制问题。它是对 TRPO（Trust Region Policy Optimization）算法的改进，旨在提高训练稳定性和样本效率，同时减少对计算资源的需求。PPO 算法近年来在多个基准测试中表现出色，尤其在复杂环境中的表现接近或超越了以前的强化学习方法。

1.2.2算法原理

PPO 算法主要使用了两个神经网络，分别用来你和策略函数和价值函数：

•**策略网络**: 策略网络根据当前状态输出每个动作的概率分布,用于选择动作，表示为$\pi(a|s;\theta)$。

•**价值网络**: 价值网络为每个状态估计一个值，表示智能体从该状态开始能获得的长期回报的预期，表示为$V(s;\varphi)$。

1.2.3核心机制

1.2.3.1 剪切目标函数

PPO 引入了一个剪切目标函数来控制策略的更新幅度。它通过利用一个比率（即新策略与旧策略的比率）来调整目标函数，避免过大的策略更新。具体形式为：

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\epsilon)]$$

其中,$r_t(\theta) = \frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{old})}$是比例因子，$\hat{A}_t$是优势估计，$\varepsilon$是一个很小的常数。这个目标函数的设计保证了在策略优化过程中，新的策略不会大幅偏离旧的策略，从而提高了训练的稳定性。

1.2.3.2 熵奖励

类似于 DQN 中的$\varepsilon - greedy$策略，PPO 通过引入熵奖励来增强策略的探索性。熵奖励是对策略输出概率分布的不确定性的度量，鼓励智能体在策略训练的初期多尝试各种动作，从而避免过早确定策略导致的收敛于次优解。

2 DQN 贪吃蛇智能体的建模与设计

2.1 贪吃蛇游戏设计

2.1.1 游戏概述

贪吃蛇游戏是一款经典的电子游戏，其主要玩法为玩家控制一条贪吃蛇在一个二维的网格上移动，网格上会随机刷新食物，玩家的任务是控制贪吃蛇尽可能多的吃到食物，同时避免自己撞到墙壁、障碍物和自己的身体。随着食物的摄入，蛇身会变得越来越长，这样就加大了游戏的难度和挑战性。

2.1.2游戏环境的设计

2.1.2.1 游戏环境设计

贪吃蛇的游戏局面大小为 21x21，具体元素包括以下方面：

蛇的位置：蛇身由其蛇身长度个方格组成，可以使用一个存储二元元组的列表进行表示，通常蛇头坐标为列表的第一个元素。

食物的位置：游戏中食物的位置随机生成，并且在被蛇吃掉后进行更新。

游戏边界和障碍物：贪吃蛇的活动边界为二维网格，其不能够朝上下左右方向越过网格，否则将判定为死亡。此外，游戏网格中部还设置有两行障碍物，同理贪吃蛇碰撞后也会直接死亡。

由于算力限制，在获取游戏环境时我选择不使用 opencv 或者 pygame 的图像帧导出功能，而是使用一个 **21*21 的张量**来存储环境特征，其中各元素在张量中对应的值为：

贪吃蛇头部：9	*贪吃蛇身体：3*
食物：20	*障碍物：-1*

为方便 Agent 了其所处的具体环境，Agent 所处状态 state 是当前时刻往前 4 个时间帧的游戏画面的特征张量，即 Agent 的 state 被设计为一个 **4*21*21 的张量**。

2.2.2.2 动作空间

在贪吃蛇游戏中，动作通常限制为四个方向（上、下、左、右）：

- 上（1）：表示蛇向上移动。
- 下（2）：表示蛇向下移动。
- 左（3）：表示蛇向左移动。
- 右（4）：表示蛇向右移动。

以上四个动作为 Agent 在每一步的所有可能选择，需要注意的是每一时刻 Agent 其实只能三选一，因为蛇不能突然朝反方向运动。

在以上设计的基础上，使用 python 作为编程语言，调用图形库 pygame 就可以实现该贪吃蛇游戏环境，运行时结果如下：

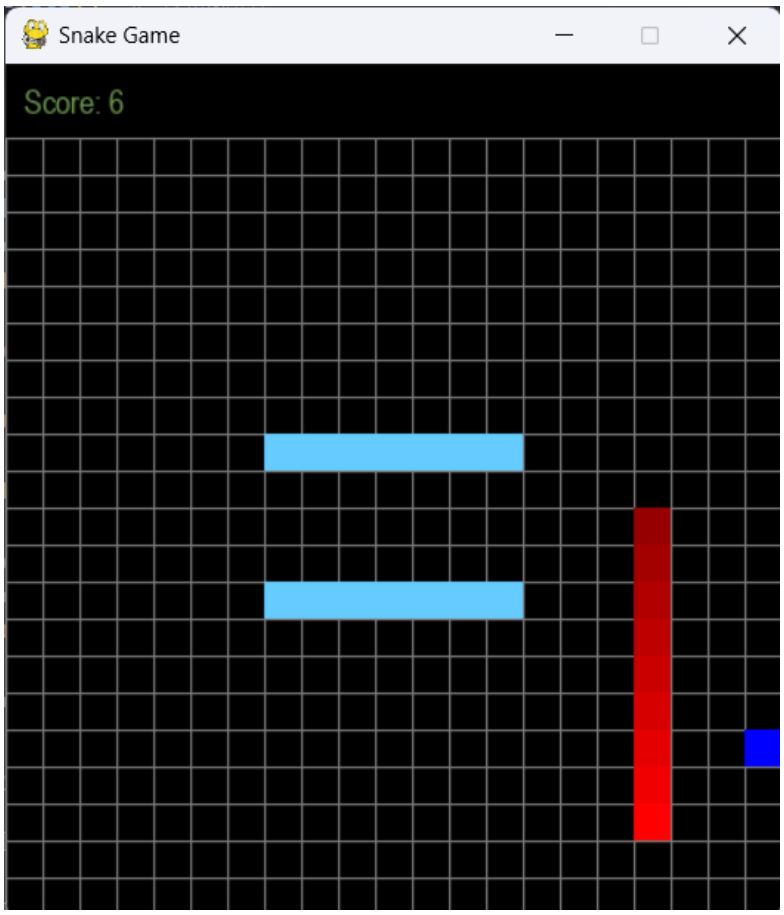

图 2-1 游戏运行时截图

2.2 DQN 网络设计

在游戏环境准备好后，下一步便是搭建 DQN 网络。为加深对于 DQN 网络的理解，这里我使用 pytorch 框架来搭建该网络。该网络在代码中主要组成部分为类 DQN 和类 DQN_AGENT，其中类 DQN 继承自 torch.nn.Module，重写了初始化网络结构的_init_()方法和前向传播的 forward()方法，前向传播时使用的激活函数为 ReLU；类 DQN_AGENT 则在_init_()时设置好训练时的各种超参数，此外还实现了添加经验到经验池、更新主 Q 网络、更新目标 Q 网络、根据 Q 网络选择下一个动作等方法。

在具体设计 DQN 类中的隐藏层时，前后一共有过两种方案，如下表所示：

输入张量 shape	层类型	层参数	输出张量 shape
（，4，21，21）	卷积层	**in_channles** 4 **out_channels** 16 **kernel_size** 7 **stride** 2 **padding** 0	（，16，8，8 ）
（，16，8，8 ）	卷积层	**in_channles** 16 **out_channels** 64 **kernel_size** 3 **stride** 1 **padding** 1	（，64，8，8 ）
（，64，8，8 ）	全连接层		（，512 ）
（，512 ）	全连接层		（，64 ）
（，64 ）	全连接层		（，4 ）

图 2-2 初始方案(超大卷积核)

输入张量 shape	层类型	层参数	输出张量 shape
（，4，21，21）	卷积层	**in_channles** 4 **out_channels** 32 **kernel_size** 3 **stride** 1 **padding** 1	（，32，21，21 ）
（，32，21，21 ）	卷积层	**in_channles** 32 **out_channels** 64 **kernel_size** 3 **stride** 1 **padding** 0	（，64，19，19 ）
（，64，19，19 ）	卷积层	**in_channles** 64 **out_channels** 128 **kernel_size** 3 **stride** 1 **padding** 0	(，128，17，17)
(，128，17，17)	全连接层	**in_channles** 128 **out_channels** 4 **kernel_size** 1 **stride** 1 **padding** 0	（，4，17，17 ）
（，4，17，17 ）	全连接层		(，128)
(，128)	全连接层		（，4 ）

图 2-3 优化后的方案

如上两种隐藏层的设计方案比较大的区别是初始方案的卷积层使用了一个比较大的 7*7 卷积核，因为在设计时我认为卷积层的卷积核越大越能提取到图像的抽象特征。但是之后在课上看到 Yolo-v2 的卷积层设计中使用的都是大小不大于 3 的卷积核，我便对我的隐藏层做了修改，以小核为主，最后一层卷积层也使用了 1*1 的超小卷积核。两种设计方案将在后续进行对比。

2.3 模型训练与对比

在 config.py 中，对超参数进行了如下设置：

```
explore = 4000000  #epsilon-greedy 探索次数
learning_rate=0.001  #DQN 网络学习率
gamma=0.9    #未来状态折扣因子
replace_target_iter=2000  #目标 Q 网络更新频率
buffer_size=20000   #经验缓冲区大小
batch_size=256  #每次训练的 batch 大小
initial_epsilon=0.6  #初始 epsilon 值
final_epsilon=0.001  #最终 epsilon 值
use_gpu=True  #使用 cuda 加速
experience_requirement=1000  #进行训练的最小经验池要求
save_model=False  #是否保存训练得到的模型文件
log_info=True    #是否输出训练日志文件到 wandb
```

DQN 模型使用 epsilon-greedy 策略，其 epsilon 值会从 0.6 开始，经过 4000000 此时间步后线性衰减到 0.001.

对于 Agent，其各种行动获得的奖励设置如下表：

Action	Reward
吃到食物	1
更接近食物	0.05
撞到墙或障碍物	-2
撞到自己的身体	-5
其他	0

此外，若 Agent 在上次吃到食物之后走了超过 30*蛇身长度的步数还没有吃到新的食物，将会直接 Game_Over。

在做好上述设置后，便可以开始对 Agent 的训练，之前对于 DQN 网络的隐藏层有两种不同的设计方案，两种方案对应的模型在经过超过 4000000 个时间步，即大约 50k 个 episode 后被终止训练，训练过程中输出日志的可视化信息如下。

使用初始方案的 DQN 模型在训练了一整个晚上后，Agent 的得分情况和每局游戏存活步数如下：

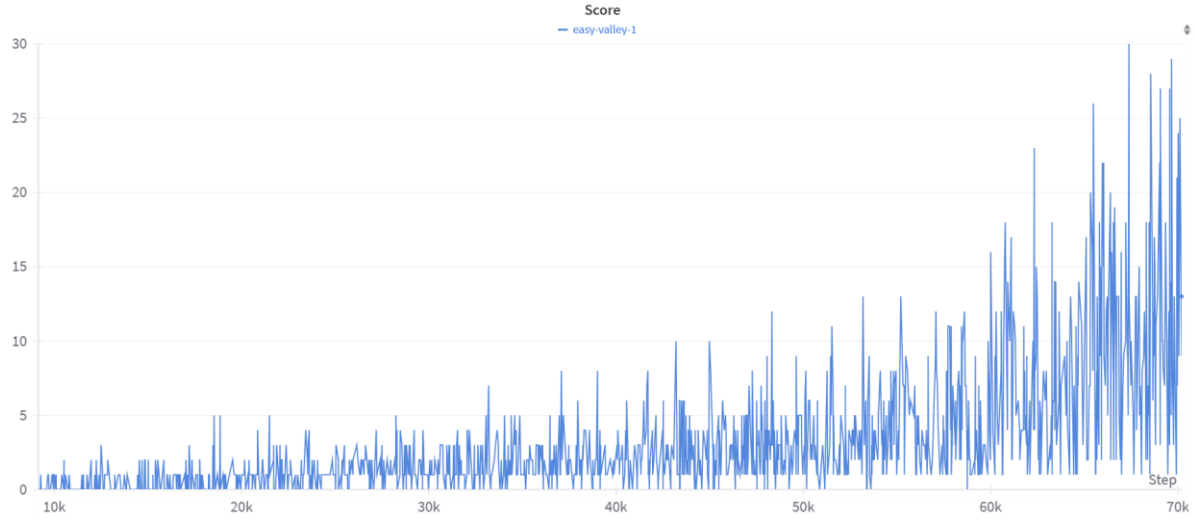

图 2-4 每局得分

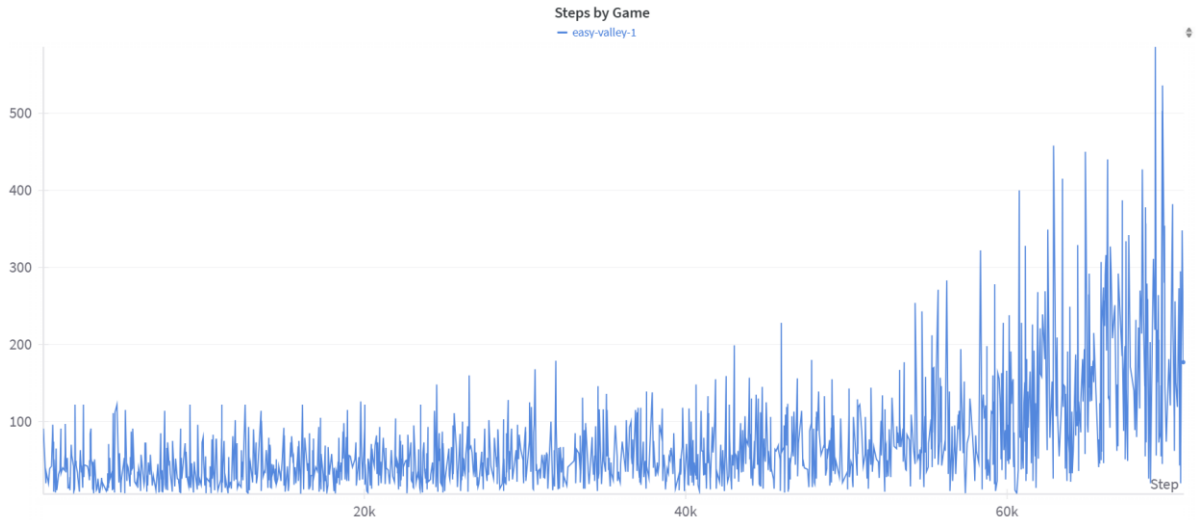

图 2-5 每局存活步数

由图可知，采用较大卷积核的 DQN 模型训练出来的 Agent 在玩了 60k 局游戏，即大约四百万个时间步后，平均得分已经来到了 10 分，而且存活步数也快速升高。但是在之后的训练中，Agent 即使多玩 10k 局也没能有很大的提升，从其最高得分 30 分可以看出，Agent 已经能够很有意识地去吃食物，并且掌握了一定的策略，使其能够在长度达到 20 以上后能够存活。但是受限于 DQN 网络的设计，在大多数尝试中它的分数也只是止步于 15，这说明较大的卷积核设计可能并未能很好地捕获到贪吃蛇所处环境的抽象特征，使得贪吃蛇不能够在自身长度较长时做出规避碰撞自身的动作决策。

优化后的 DQN 模型在训练差不多同样的时间后，可视化训练日志如下：

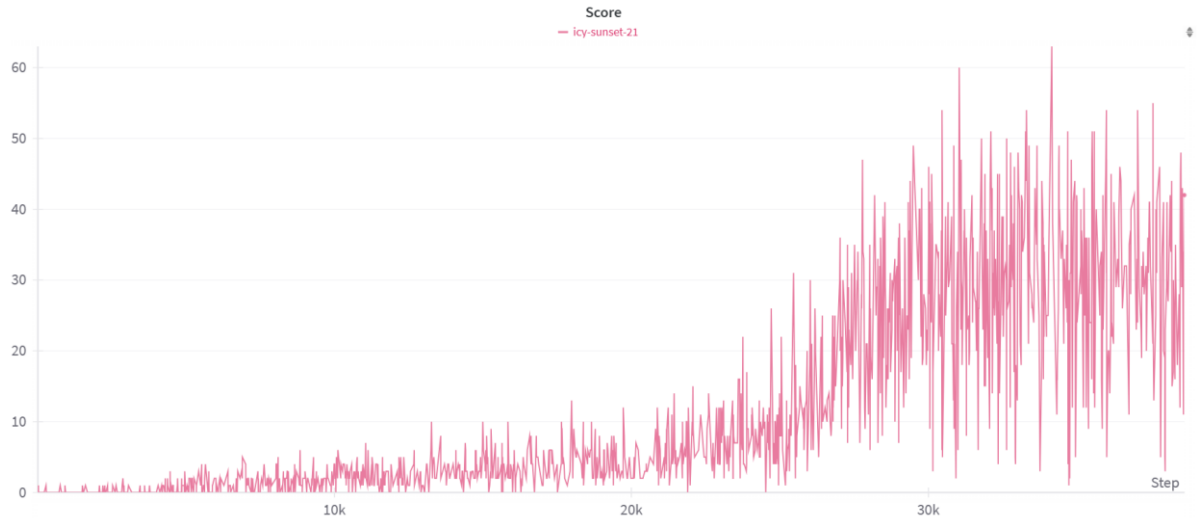

图 2-6 每局得分

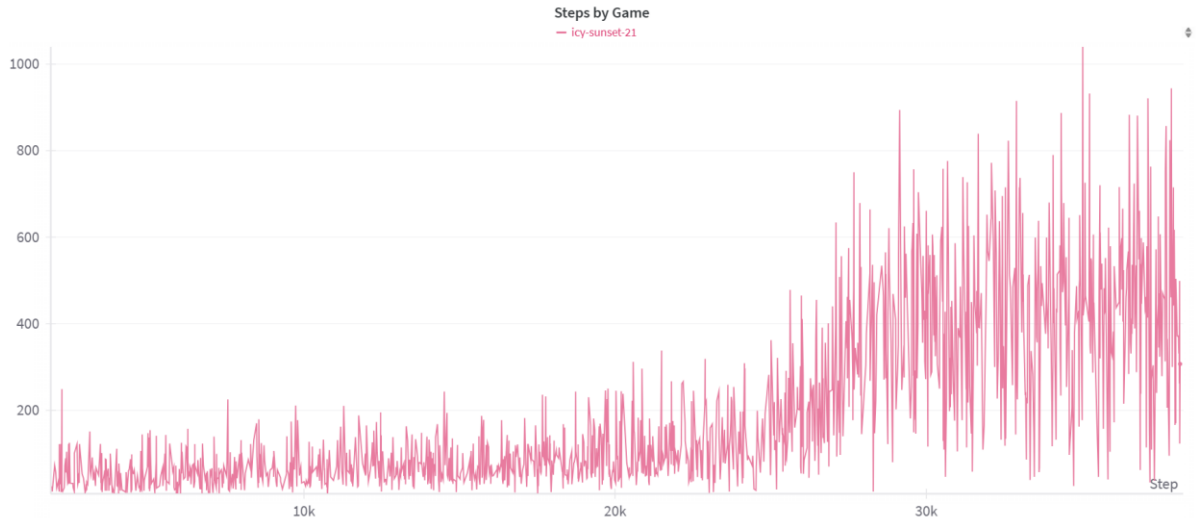

图 2-7 每局存活步数

由图可知，在优化了网络的隐藏层后，Agent 的表现有了很大的提升。最明显的是其在玩了 30k 局以后平均得分就已经能到 30 分以上，做高分更是来到了 60 分，其已经达到甚至超越了人均水平，在更多的训练后或许还能达到更高的水平。

其次与未优化前的模型相对比，可以看出，在几乎相同的时间步中，旧模型玩了 70k 局，而优化后的模型只玩了接近 40k 局，整整少了 30k 局，但是却能拿到更高的分数。此外，旧模型的质变期出现在 60k 局之后，而新模型的质变期出现在 25k 局左右，这说明 DQN 网络在优化后其性能得到了很大的提升，使得 Agent 的学习能力大为增强。综上可以得出以

下结论：

1.小卷积核（如 1x1 或 3x3）在捕捉局部特征方面表现优越，能够专注于图像中的细微变化，提取更多高维特征。

2.较小的卷积核使得网络结构更深，能够学习到更复杂的特征表示，从而有助于提高模型的学习能力。

3.小卷积核的计算量相对较小，使得模型能在保持较高表达能力的同时，减少计算负担，从而提升训练和推理速度

3.PPO 智能体贪吃蛇的设计

3.1 训练环境配置

此次实验的 PPO 模型部分直接使用 stablebaselines3 中的 PPO 模型。在使用该模型训练时,需要使用 gym 中自带的环境或自定义符合其接口规定的环境类。由于 gym.snake（openai 提供的贪吃蛇环境）已经不满足最新的接口规范，需要自定义贪吃蛇环境。在已有的贪吃蛇游戏类 SnakeGame 的基础上，可以很快地创建环境类 SnakeGymEnv(继承自 gymnasium.Env)，主要内容是为该类实现环境中每一个操作步的方法 step()、一个 episode 结束后的 reset()方法以及训练过程中游戏画面渲染的 render()方法。在实现完上述方法后，使用 gym.check_env()方法对 SnakeGymEnv 进行检查，检查通过便可以进行后续训练。

3.2 奖励设计及状态设计

奖励设计如下：

action	reeard
吃到食物	1
撞到障碍物或自己	-1
超出指定步数没吃到食物	-0.01

状态设计：

由于 sb3 中的 PPO 模型的环境特征提取的 CnnPolicy(卷积神经网络策略)使用的是 8x8 的大卷积核和 4 的步长，不适用于该贪吃蛇游戏，故在训练 PPO 模型时我使用了 MlpPolicy（多层感知机策略），于是环境的状态观测空间便不能是整个游戏地图，而是被简单的表示为一个长度为 6 的张量，能帮助 agent 大致地了解环境信息：

上方是否有危险	左方是否有危险	下方是否有危险	右方是否有危险	蛇头与食物相对 X 距离	蛇头与食物相对 Y 距离

以上状态设计较为简单，只能让 agent 感知到自身前后左右的环境和自身与食物的距离，而不能让其捕捉到自身身体的长度信息及其盘绕结构。当蛇身达到一定长度后局限性将比较明显。

3.3 模型训练及结果分析

训练时对 PPO 设置的参数如下：

```
model = PPO(
    "MlpPolicy",  #使用多层感知机策略
    env,  #自定义的环境实例
    device="cuda",  #使用 cuda 加速
    n_steps=2048,  #每次更新的样本数量
    batch_size=512,  #样本批量大小
    n_epochs=4,  #每个训练步骤中的更新轮数
    gamma=0.94,  #奖励折扣因子
    learning_rate= linear_schedule(1.5e-3, 2.5e-4),  #线性衰减的学习率
    clip_range= linear_schedule(0.150, 0.025)
    #线性衰减的剪切范围
)
```

在训练了两个小时，一共 2500 局游戏后，agent 的表现如下：

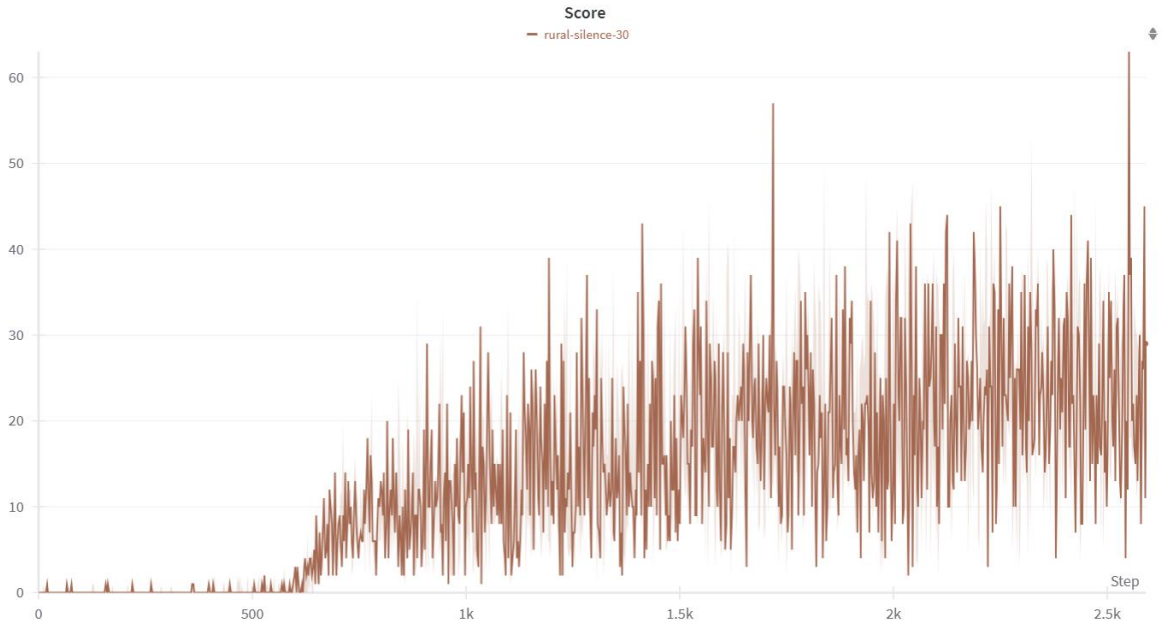

图 3-1 每局得分

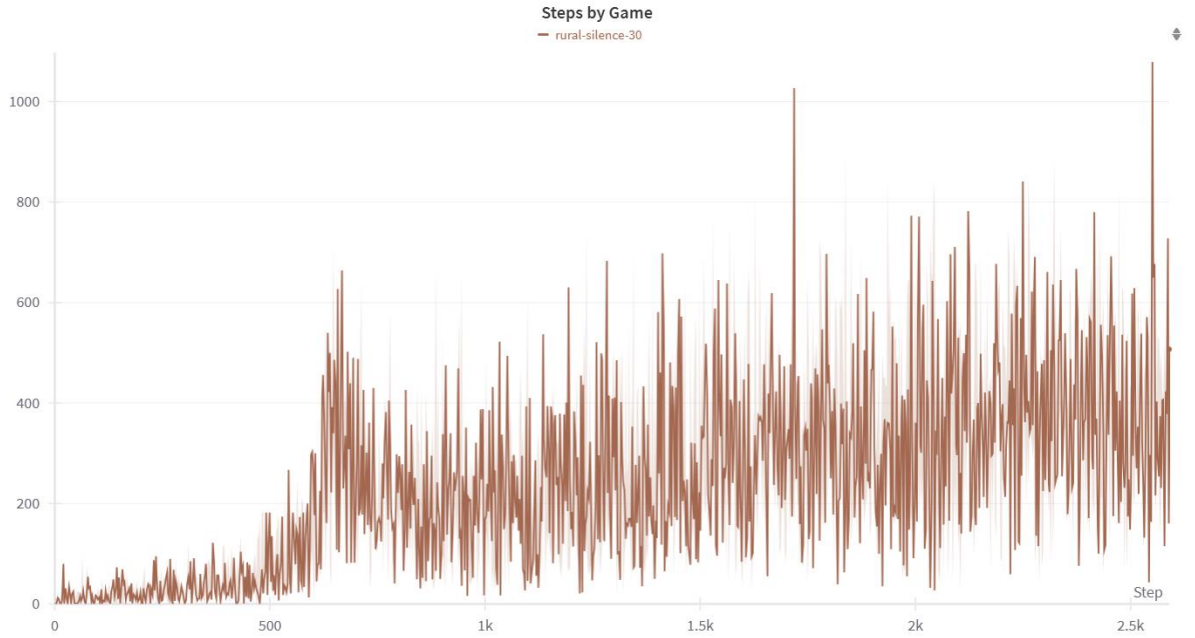

图 3-1 每局存活步数

由训练结果可知，PPO 模型训练出的 agent 在进行了 500 局游戏后能力就发生了质变，之后的分数虽然依旧有在缓慢增长，但受限于 agent 的环境观测空间不全，其在面对 30 以上长度的蛇身时会比较吃力，而且进步空间受限，但即使如此其也能得到最高 60 分的成绩，这已经达到了预期。

4 DQN 策略与 PPO 策略的对比

4.1 模型训练结果分析

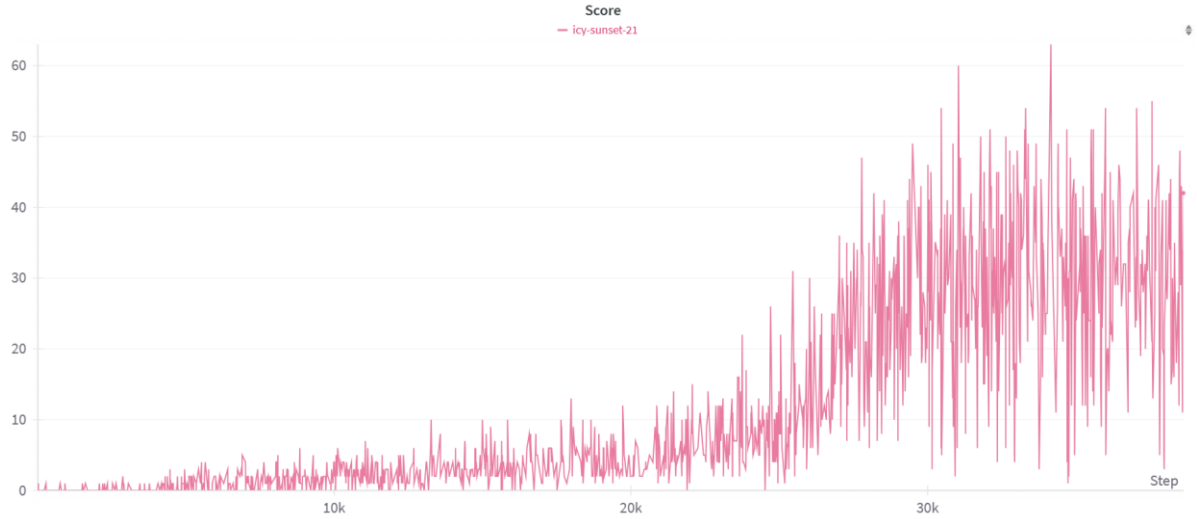

图 4-1 DQN 训练得分结果

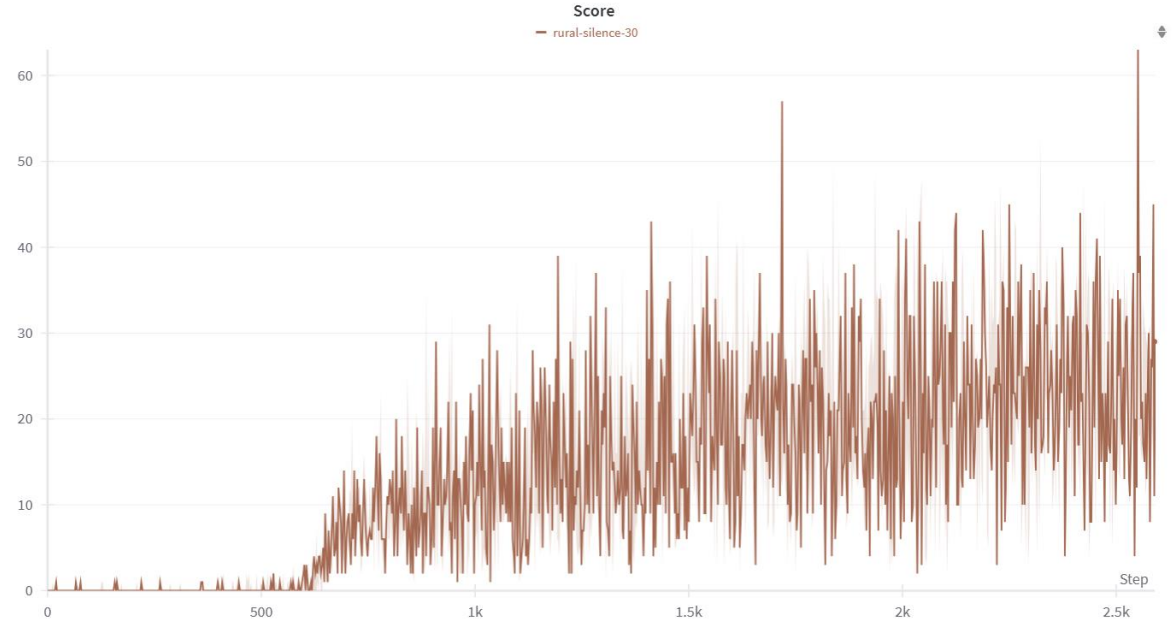

图 4-2 PPO 训练得分结果

1. DQN 算法的学习曲线呈现出较为平稳的增长趋势，其在前 20k 局游戏中的表现都不是很理想，在进行了 30k 局游戏时才达到高分区域，这说明 DQN 算法的收敛速度较慢，需要更多的训练才能使 agent 达到理想水平。

2. PPO 算法的学习曲线在早期阶段得分增长更快，其在进行了约 600 局游戏后分数就能达到 10 分以上，1k 局后就能达到高分区域，且在 3k 局以

前就已经有收敛的趋势，这说明 PPO 算法的收敛速度要远快于 DQN。
根据上述分析，可能原因如下：

1. DQN 算法基于价值学习，使用深度神经网络近似 Q 值函数，为稳定训练过程其使用了经验回放池机制，但降低样本的相关性的同时也使得其需要花费更多的时间才能收敛，主要是由于更新过程中使用的样本不再是最新的状态-动作对，可能导致不一致的状态估计和较高的方差；而 PPO 算法是基于策略学习的算法，其通过直接优化策略函数学习当下的最优策略，同时 PPO 算法只使用一次采样到的经验，这样使得训练这使得训练过程更加高效和灵活，并能够较快地收敛到一个稳定的策略。

2. 在探索方面，DQN 使用$\epsilon - greedy$算法，使 agent 能在初期有效的进行探索，但是其对ϵ值得依赖性较大，当ϵ较大时会导致 agent 在后期过多的选择劣质的动作而影响其学习过程，当ϵ较小时 agent 会因为探索不足而导致其无法发现更优的策略，受其影响 DQN 算法需要花更多时间才能收敛到一个不错的策略；而 PPO 算法使用熵奖励作为探索机制，增加策略分布的不确定性，从而鼓励智能体选择多样化的动作，这种机制是基于 agent 当前的动作和环境反馈进行的，使得智能体能够动态适应环境的变化，持续进行有效的探索，故其受超参数的影响较小，相较于 DQN 算法其能够更快的找到一个较优的策略并能避免收敛到一个不良的策略。

3. 训练稳定性方面，DQN 算法在更新 Q 值时使用下一状态的最高 Q 值作为未来奖励，这样使得 Q 值容易被较高估计，即使得 agent 倾向于高估当前状态下某一个动作的价值而不能做出真正有全局性的选择，使得 agent 体训练过程中容易出现震荡，同时也增加了模型收敛的用时；PPO 算法通过使用剪切目标函数限制了策略函数的更新幅度，从而使得训练过程更平滑和更稳定。

总结展望和思考

此次实验是我和队友自选的主题，使用主流强化学习方法去训练玩贪吃蛇的 agent，我负责实现 DQN 和 PPO 算法下 agent 的训练，而 A3C 算法理论上队友应该实现了。

无论如何，这次实验对于相当于初次接触人工智能的我来说无疑是一次很大的挑战，之前我对人工智能的理解仅仅是局限于强化学习，但是也一直没有上手实践过。在经过了本学期的人工智能导论课的学习后，我才知道原来人工智能涵盖的知识面这么广泛，启发式搜索、逻辑推理和知识表达原来都是属于人工智能的范畴，自己所了解的不过是冰山一角，好在此次课程让我对人工智能这座大山有了大致的认知。

实验中的 DQN 模型是我自己搭建的，当时也花了一天时间去理解他人代码中对 tensor 的各种神奇操作，然后一点点的堆砌出整个模型并跑通了训练代码，那时的我沾沾自喜，还不知道搭建完网络只是一个开始罢了。在之后的训练过程中，对于任何一个超参数的改动，就算使用 cuda 加速也得训练上一两个小时才能看出调整后的模型是否表现得更好，这种试错成本我之前是没有体验过的。在那段时间里我先是在上课前开始一轮训练，然后在课上经常会偷瞄一眼 wandb 上的训练过程可视化图像，期待着 agent 能带给我新的惊喜。在调整了一个星期后，模型终于达到了预期的效果。

由于时间比较紧迫，PPO 模型部分我直接调用了 stablebaselines3 中的 PPO 模型，在减轻工作量的同时训练出来的模型也取得了不错的效果，同时因为 PPO 算法的高效性，调整参数的试错成本也降低了不少。

但是我的模型其实还是有很多不足之处的，例如无论是 DQN 还是 PPO 的模型，我都对 agent 所处环境的表示做出了简化，使得训练出来的模型还无法在自身长度很长的环境下有效存活。同时我给 agent 设置的奖励机制也可能过于简单，还不能更好地指导 agent 去更好的发现最优策略和获得更高的分数。

总而言之，DQN 和 PPO 都是强化学习领域中很强大的算法，其在现实生活中的应用不计其数，用它们来训练一个玩贪吃蛇的 agent 或许真的是有点大炮打蚊子的意味了，但是若是因此而好高骛远，又未免会变得眼高手低。此次人工智能实验对于我来说就是一个很好的开始，也将激励我在人工智能的道路上继续探索和前进。

参考资料：

【1】 Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. arXiv preprint arXiv:1312.5602.

【2】 van Hasselt, H., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. arXiv preprint arXiv:1509.06461.

【3】 Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.

【4】 zouyih. (2018). DQN-tensorflow-gluttonous_snake. GitHub. https://github.com/zouyih/DQN-tenso

附录

config.py

```
class CONFIG:
    explore = 4000000
    learning_rate=0.001
    gamma=0.9
    replace_target_iter=2000
    buffer_size=20000
    batch_size=256
    initial_epsilon=0.6
    final_epsilon=0.001
    use_gpu=True
    save_model_frequency=10000
    experience_requirement=1000
    save_model=True
    log_info=False
```

snake_game.py

```
import pygame
import numpy as np
import random
import sys


class SnakeGame:
    def __init__(self):
        pygame.init()
        self.X_BLOCK_NUM =21
```

```
        self.Y_BLOCK_NUM =21
        self.BLOCK_SIZE = 20
        self.WINDOW_WIDTH = self.X_BLOCK_NUM * self.BLOCK_SIZE
        self.WINDOW_HEIGHT = self.Y_BLOCK_NUM * self.BLOCK_SIZE
        self.HEADER_HEIGHT = 40

        self.BG_COLOR = (0, 0, 0)
        self.SNAKE_COLOR = (255, 0, 0)
        self.SNAKE_HEAD_COLOR = (255,255,255)
        self.FOOD_COLOR = (0, 0, 255)
        self.BORDER_COLOR = (125, 125, 125)
        self.OBSTACLE_COLOR = (102,204,255)
        self.TEXT_COLOR = (117, 162, 89)

        self.SPEED = 10

        self.game_display = pygame.display.set_mode((self.WINDOW_WIDTH,
self.WINDOW_HEIGHT + self.HEADER_HEIGHT))
        pygame.display.set_caption('Snake Game')
        self.clock = pygame.time.Clock()
        self.clock_tick = 10

        self.snake_pos = [(5, 5), (5, 4), (5, 3)]
        self.obstacle_pos= [(i,8) for i in range(7,14)]+[(i,12) for i in
range(7,14)]
        self.snake_direction = (1, 0)  # 初始方向（向右）
        self.food_pos = self.generate_food()

        self.score = 0
```

```
        self.game_over = False

    def generate_food(self):
        while True:
            food_x = random.randint(0, (self.WINDOW_WIDTH // self.BLOCK_SIZE)
- 1)
            food_y = random.randint(0, (self.WINDOW_HEIGHT // self.BLOCK_SIZE)
- 1)
            if (food_x, food_y) not in self.snake_pos and (food_x, food_y) not
in self.obstacle_pos:
                return (food_x, food_y)
    def is_collision(self,pt=None):
        if pt==None:
            pt=self.snake_pos[0]
        if (pt in self.snake_pos[:-1]) or pt in self.obstacle_pos or (pt[0] <
0) or (pt[0] >= self.WINDOW_WIDTH // self.BLOCK_SIZE) or (pt[1] < 0) or
(pt[1] >= (self.WINDOW_HEIGHT // self.BLOCK_SIZE)):
            return True
        return False


    def move_snake(self):
        if not self.game_over:
            head_x, head_y = self.snake_pos[0]
            new_head = (head_x + self.snake_direction[0], head_y +
self.snake_direction[1])

            if self.is_collision(new_head):
                self.game_over = True
            else:
```

```
                self.snake_pos.insert(0, new_head)  # 在蛇头插入新位置

                if new_head == self.food_pos:
                    self.score += 1
                    self.food_pos = self.generate_food()  # 生成新的食物
                else:
                    self.snake_pos.pop()  # 移除蛇尾

    def draw_snake(self):
        colors = np.linspace(self.SNAKE_COLOR, (150,0,0),
len(self.snake_pos)).astype(int)
        #渐变色蛇身
        pygame.draw.rect(self.game_display, self.SNAKE_COLOR,
                         (self.snake_pos[0][0] * self.BLOCK_SIZE,
                         self.snake_pos[0][1] * self.BLOCK_SIZE +
self.HEADER_HEIGHT,
                         self.BLOCK_SIZE, self.BLOCK_SIZE))

        for i, pos in enumerate(self.snake_pos[1:]):
            pygame.draw.rect(self.game_display, colors[i + 1],
                             (pos[0] * self.BLOCK_SIZE,
                             pos[1] * self.BLOCK_SIZE + self.HEADER_HEIGHT,
                             self.BLOCK_SIZE, self.BLOCK_SIZE))
    def draw_food(self):
        pygame.draw.rect(self.game_display, self.FOOD_COLOR,
                        (self.food_pos[0] * self.BLOCK_SIZE, self.food_pos[1]
* self.BLOCK_SIZE + self.HEADER_HEIGHT,
                         self.BLOCK_SIZE, self.BLOCK_SIZE))
    def draw_obstacle(self):
```

```
        for pos in self.obstacle_pos:
            pygame.draw.rect(self.game_display, self.OBSTACLE_COLOR,
                             (pos[0] * self.BLOCK_SIZE, pos[1] *
self.BLOCK_SIZE + self.HEADER_HEIGHT,
                              self.BLOCK_SIZE, self.BLOCK_SIZE))

    def display_score(self):
        font = pygame.font.SysFont('Arial', 18)
        score_surface = font.render(f'Score: {self.score}', True,
self.TEXT_COLOR)
        self.game_display.blit(score_surface, (10, 10))

    def draw_grid(self):
        for x in range(0, self.WINDOW_WIDTH, self.BLOCK_SIZE):
            pygame.draw.line(self.game_display, self.BORDER_COLOR, (x,
self.HEADER_HEIGHT), (x, self.WINDOW_HEIGHT + self.HEADER_HEIGHT))
        for y in range(0, self.WINDOW_HEIGHT, self.BLOCK_SIZE):
            pygame.draw.line(self.game_display, self.BORDER_COLOR, (0, y +
self.HEADER_HEIGHT), (self.WINDOW_WIDTH, y + self.HEADER_HEIGHT))

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP and self.snake_direction != (0,
1):
                    self.snake_direction = (0, -1)  # 向上
```

```
                elif event.key == pygame.K_DOWN and self.snake_direction !=
(0, -1):
                    self.snake_direction = (0, 1)  # 向下
                elif event.key == pygame.K_LEFT and self.snake_direction !=
(1, 0):
                    self.snake_direction = (-1, 0)  # 向左
                elif event.key == pygame.K_RIGHT and self.snake_direction !=
(-1, 0):
                    self.snake_direction = (1, 0)  # 向右
                # 使用 W, A, S, D 控制方向
                elif event.key == pygame.K_w and self.snake_direction != (0,
1):
                    self.snake_direction = (0, -1)  # 向上
                elif event.key == pygame.K_s and self.snake_direction != (0, -
1):
                    self.snake_direction = (0, 1)  # 向下
                elif event.key == pygame.K_a and self.snake_direction != (1,
0):
                    self.snake_direction = (-1, 0)  # 向左
                elif event.key == pygame.K_d and self.snake_direction != (-1,
0):
                    self.snake_direction = (1, 0)  # 向右

    def reset_game(self):
        self.snake_pos = [(5, 5), (5, 4), (5, 3)]
        self.snake_direction = (1, 0)
        self.food_pos = self.generate_food()
        self.score = 0
        self.game_over = False
```

```
    def game_loop(self):
        while True:
            self.handle_events()
            self.move_snake()

            self.game_display.fill(self.BG_COLOR)
            self.draw_grid()
            self.draw_food()
            self.draw_obstacle()
            self.draw_snake()
            self.display_score()

            if self.game_over:
                font = pygame.font.SysFont('Arial', 36)
                game_over_surface = font.render('Game Over!', True,
self.TEXT_COLOR)
                self.game_display.blit(game_over_surface, (self.WINDOW_WIDTH
// 4, self.WINDOW_HEIGHT // 2))
                pygame.display.update()
                pygame.time.wait(500)
                self.reset_game()

            pygame.display.update()
            self.clock.tick(self.SPEED)

if __name__ == "__main__":
    game = SnakeGame()
    game.game_loop()
```

my_dqn.py

```
import torch
import torch.nn as nn
from collections import deque
import random
from math import exp
from config import CONFIG

device=torch.device("cuda" if CONFIG.use_gpu and torch.cuda.is_available()
else "cpu")
print(f"using device:{device}")

class DQN(nn.Module):
    def __init__(self):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=4, out_channels=32, kernel_size=3,
stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
kernel_size=3, stride=1, padding=0)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128,
kernel_size=3, stride=1,padding=0)
        self.conv4 = nn.Conv2d(in_channels=128, out_channels=4,
kernel_size=1, stride=1,padding=0)
        self.fc1 = nn.Linear(4*17*17,128)
        self.fc2=nn.Linear(128,4)
    def load_model_weights(self, file_path):
        self.load_state_dict(torch.load(file_path))
        print('Weights loaded')
```

```
    def forward(self, x ,batch_size=1):
        x = torch.relu(self.conv1(x))
        #print(x.shape)
        x = torch.relu(self.conv2(x))
        #print(x.shape)
        x = torch.relu(self.conv3(x))
        x = torch.relu(self.conv4(x))
        x = x.view(x.size(0), -1)
        #print(x.shape)
        x = torch.relu(self.fc1(x))
        #print(x.shape)
        x=self.fc2(x)
        return x
class DQN_AGENT():
    def __init__(self,epsilon=CONFIG.initial_epsilon):
        self.model=DQN().to(device)
        self.target_model=DQN().to(device)
        self.total_steps=0
        self.total_circles=0
        self.learning_rate=CONFIG.learning_rate
        self.saved_state=[torch.zeros((21,21),device=device) for _ in
range(4)]
        self.saved_experience=deque(maxlen=CONFIG.buffer_size)
        self.epsilon=epsilon
        self.optimizer = torch.optim.Adam(self.model.parameters(),
lr=self.learning_rate)
    def select_next_action(self,game_state,epsilon_greedy=True):
        self.saved_state.pop(0)
        self.saved_state.append(torch.tensor(game_state).to(device))
```

```
        state=torch.stack(self.saved_state,dim=0)


        self.total_steps += 1
        self.epsilon=max(CONFIG.final_epsilon,self.epsilon)
        self.epsilon -= (CONFIG.initial_epsilon-
CONFIG.final_epsilon)/CONFIG.explore
        if epsilon_greedy and random.random() < self.epsilon:
            action=random.randint(1,4)
        else:
            with torch.no_grad():
                q_values=self.model(state.unsqueeze(0))
                action=torch.argmax(q_values).item()+1
        return action
    def add_experience(self,state,action,reward,next_state,is_over):
        state_tensor=torch.stack(state,dim=0).to(dtype=torch.float32,device=d
evice)
        next_state_tensor=torch.stack(next_state,dim=0).to(dtype=torch.float3
2,device=device)
        experience=(state_tensor,action,reward,next_state_tensor,is_over)
        self.saved_experience.append(experience)
    def
update_q_network(self,batch_size=CONFIG.batch_size,gamma=CONFIG.gamma):
        if len(self.saved_experience) < batch_size:
            return
        batch=random.sample(self.saved_experience,batch_size)


        states=torch.stack([exp[0] for exp in
batch],dim=0).to(dtype=torch.float32,device=device)
```

```
        actions=torch.tensor([exp[1] for exp in
batch],dtype=torch.int,device=device)
        rewards=torch.tensor([exp[2] for exp in
batch],dtype=torch.float32,device=device)
        next_states=torch.stack([exp[3] for exp in
batch]).to(dtype=torch.float32,device=device)
        dones=torch.tensor([exp[4] for exp in
batch],dtype=torch.bool,device=device)

        q_values=self.model(states)
        next_q_values=self.target_model(next_states)

        expected_q_values = q_values.clone()
        max_next_q_values = torch.max(next_q_values, dim=1).values
        target = rewards + gamma * max_next_q_values * (~dones)

        expected_q_values[torch.arange(len(batch)), actions - 1] = target

        loss = nn.MSELoss()(q_values, expected_q_values.detach())

        self.optimizer.zero_grad()
        loss.backward()
        nn.utils.clip_grad_norm_(self.model.parameters(), 1.0)
        self.optimizer.step()
    def
DDQN_update_q_network(self,batch_size=CONFIG.batch_size,gamma=CONFIG.gamma):
        #DDQN 算法的尝试，训练过程确实比 DQN 要平稳
        if len(self.saved_experience) < batch_size:
            return
```

```
        batch=random.sample(self.saved_experience,batch_size)

        states=torch.stack([exp[0] for exp in
batch],dim=0).to(dtype=torch.float32,device=device)
        actions=torch.tensor([exp[1] for exp in
batch],dtype=torch.int,device=device)
        rewards=torch.tensor([exp[2] for exp in
batch],dtype=torch.float32,device=device)
        next_states=torch.stack([exp[3] for exp in
batch]).to(dtype=torch.float32,device=device)
        dones=torch.tensor([exp[4] for exp in
batch],dtype=torch.bool,device=device)

        q_values=self.model(states)

        max_q_value_action = torch.argmax(self.model(next_states),dim=1)
        next_q_values = self.target_model(next_states)
        target_next_q_values =
next_q_values[torch.arange(len(batch)),max_q_value_action]

        expected_q_values = q_values.clone()
        target = rewards + gamma * target_next_q_values * (~dones)
        expected_q_values[torch.arange(len(batch)), actions - 1] = target

        loss = nn.MSELoss()(q_values, expected_q_values.detach())

        self.optimizer.zero_grad()
        loss.backward()
        nn.utils.clip_grad_norm_(self.model.parameters(), 1.0)
```

```
        self.optimizer.step()


    def update_target_network(self):
            self.target_model.load_state_dict(self.model.state_dict())
```

dqn_train.py

```
from snake_game import SnakeGame
from my_dqn import DQN_AGENT,CONFIG
import numpy as np
import torch
import wandb
import pygame
import math
import copy
import os


os.environ["WANDB_MODE"] = "offline"


device=torch.device("cuda" if CONFIG.use_gpu and torch.cuda.is_available()
else "cpu")
print(f"using device:{device}")


class GmaeAgent(SnakeGame):
    def __init__(self,agent):
        super().__init__()
        self.dqn_agent=agent
```

```
        self.action=3  #1 up 2 down 3 left 4 right
        self.reward=0
        self.total_reward=0
        self.step=0
        self.show_snake=True
        self.clock_tick=5000
        self.previous_score_step=0
        self.max_tolerance_step=40
        # 初始化一个 W&B 实验/项目
        if CONFIG.log_info:
            wandb.init(
                project='Snake_DQN_Agent_final',
                config={
                    'model': "DQN",
                    'learning_rate': self.dqn_agent.learning_rate,
                    'epsilon': self.dqn_agent.epsilon
                }
            )


    def move_snake_and_get_reward(self,action):
        if action not in [1,2,3,4]:
            return
        if action == 1 and self.snake_direction != (0, 1):
            self.snake_direction = (0, -1)  # 向上
        elif action == 2 and self.snake_direction != (0, -1):
            self.snake_direction = (0, 1)  # 向下
        elif action == 3 and self.snake_direction != (1, 0):
            self.snake_direction = (-1, 0)  # 向左
        elif action == 4 and self.snake_direction != (-1, 0):
```

```
            self.snake_direction = (1, 0)  # 向右
        if not self.game_over:
            head_x, head_y = self.snake_pos[0]
            new_head = (head_x + self.snake_direction[0], head_y +
self.snake_direction[1])

            if (new_head[0] < 0) or (new_head[0] >= self.WINDOW_WIDTH //
self.BLOCK_SIZE) or (new_head[1] < 0) or (new_head[1] >= (self.WINDOW_HEIGHT
// self.BLOCK_SIZE))  or new_head in self.obstacle_pos:
                self.game_over = True
                self.reward = -2
            elif new_head in self.snake_pos[:-1]:
                self.game_over = True
                self.reward = -5
            else:
                self.snake_pos.insert(0, new_head)  # 在蛇头插入新位置

                if new_head == self.food_pos:
                    self.reward = 1
                    self.score += 1
                    self.previous_score_step = self.step
                    self.food_pos = self.generate_food()  # 生成新的食物
                else:
                    self.reward = 0
                    self.snake_pos.pop()  # 移除蛇尾

    def display_steps(self):
        font = pygame.font.SysFont('Arial', 18)
```

```
        score_surface = font.render(f'Step: {self.dqn_agent.total_steps}',
True, self.TEXT_COLOR)
        self.game_display.blit(score_surface, (230, 10))
    def display_circle(self):
        font = pygame.font.SysFont('Arial', 18)
        score_surface = font.render(f'Circle:
{self.dqn_agent.total_circles}', True, self.TEXT_COLOR)
        self.game_display.blit(score_surface, (100, 10))

    def draw_game(self):
        self.game_display.fill(self.BG_COLOR)

        self.draw_grid()
        self.draw_food()
        self.draw_obstacle()
        self.draw_snake()
        self.display_score()
        self.display_steps()
        self.display_circle()
        pygame.display.update()

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                exit()
            if event.type == pygame.KEYDOWN:
                #切换画面帧数，5000 用于训练，10 用于观察
                if event.key == pygame.K_DOWN:
```

```
                    if self.clock_tick == 5000:
                        self.clock_tick = 10
                    elif self.clock_tick == 10:
                        self.clock_tick = 5000
    def reset_game(self):
        self.dqn_agent.total_circles+=1
        if CONFIG.log_info:
            wandb.log({
                'Total Steps': self.dqn_agent.total_steps,
                'Score': self.score,
                'Steps by Game': self.step,
                'epsilon':self.dqn_agent.epsilon,
                "Reward":self.total_reward
            })
        super().reset_game()
        self.reward=0
        self.action=3
        self.step=0
        self.previous_score_step=0
        self.total_reward=0
    def get_game_state(self):
        game_state=torch.ones((self.WINDOW_HEIGHT // self.BLOCK_SIZE,
self.WINDOW_WIDTH // self.BLOCK_SIZE),device=device)
        game_state[self.snake_pos[0][0]][self.snake_pos[0][1]]=9
        for x,y in self.snake_pos[1:]:
            game_state[x][y]=3
        for x,y in self.obstacle_pos:
            game_state[x][y]=-1
        game_state[self.food_pos[0]][self.food_pos[1]]=20
```

```
        return game_state
    def agent_play_game(self):
        while True:
            self.handle_events()
            state=self.get_game_state()

            food_distance = math.dist(self.snake_pos[0],self.food_pos)

            if self.step>0:
                self.action=self.dqn_agent.select_next_action(state)
            previous_state=copy.deepcopy(self.dqn_agent.saved_state)

            self.move_snake_and_get_reward(self.action)
            new_food_distance = math.dist(self.snake_pos[0],self.food_pos)

            if not self.game_over:
                if self.step - self.previous_score_step >
self.max_tolerance_step*len(self.snake_pos) :
                    self.game_over = True
                elif new_food_distance < food_distance:
                    self.reward = 0.05
            self.step+=1
            self.total_reward += self.reward

            self.draw_game()

            next_state=copy.deepcopy(self.dqn_agent.saved_state)
            next_state.pop(0)
            next_state.append(self.get_game_state())
```

```
            self.dqn_agent.add_experience(previous_state,self.action,self.rewa
rd,next_state,self.game_over)

            if self.dqn_agent.total_steps % 100 == 0 and
len(self.dqn_agent.saved_experience) > CONFIG.experience_requirement:
                self.dqn_agent.update_q_network()
            if self.dqn_agent.total_steps % CONFIG.replace_target_iter == 0
and self.dqn_agent.total_steps != 0:
                self.dqn_agent.update_target_network()

            if CONFIG.save_model and
self.dqn_agent.total_steps %CONFIG.save_model_frequency == 0 and
self.dqn_agent.total_steps != 0:
                torch.save(self.dqn_agent.model.state_dict(),"~/temp/saved_mod
el.pth")
                torch.save(self.dqn_agent.target_model.state_dict(),"~/temp/sa
ved_target_model.pth")

            if self.game_over:
                self.reset_game()
            self.clock.tick(self.clock_tick)
if __name__ == '__main__':
    dqn_agent = DQN_AGENT()
    # dqn_agent.model.load_model_weights("~/temp/saved_model.pth")
    #
dqn_agent.target_model.load_model_weights("~/temp/saved_target_model.pth")
    game = GmaeAgent(dqn_agent)
    game.agent_play_game()
```

Env4PPO.py

```python
import gymnasium as gym
from gymnasium import spaces
from snake_game import SnakeGame
from config import CONFIG
from stable_baselines3.common.env_checker import check_env
import numpy as np
import wandb
import math
import torch
import pygame



class SnakeGymEnv(gym.Env):
    def __init__(self):
        super(SnakeGymEnv, self).__init__()
        self.game = SnakeGame()  # 初始化自定义的贪吃蛇游戏
        self.action_space = spaces.Discrete(4)  # 动作空间：上下左右
        self.SPEED = 5000
        self.steps = 0
        self.total_steps = 0
        self.total_reward = 0
        self.previous_score_step=0
        self.max_tolerance_step=30

        if CONFIG.log_info:
            wandb.init(
                project='Snake_PPO_Agent_final',
```

```
            config={
                'model': "PPO"
            }
        )


    self.observation_space = gym.spaces.Box(
        low=-1.0, high=1.0, shape=(6,), dtype=np.float32
    )

def reset(self, seed=None, options=None):
    if CONFIG.log_info:
        wandb.log({
            'Total Steps': self.total_steps,
            'Score': self.game.score,
            'Steps by Game': self.steps,
            "Reward":self.total_reward
        })
    super().reset(seed=seed)
    self.game.reset_game()
    self.reward = 0
    self.steps = 0
    self.total_reward = 0
    self.previous_score_step = 0

    state = self.get_state()
    return state, {}  # 返回状态和附加信息（新接口要求）
def move_snake(self):
    if not self.game.game_over:
        head_x, head_y = self.game.snake_pos[0]
```

```
            food_distance =
math.dist(self.game.snake_pos[0],self.game.food_pos)
            new_head = (head_x + self.game.snake_direction[0], head_y +
self.game.snake_direction[1])
            new_food_distance = math.dist(new_head,self.game.food_pos)

            if self.game.is_collision(new_head):
                self.reward = -1
                self.game.game_over = True
            else:
                self.game.snake_pos.insert(0, new_head)  # 在蛇头插入新位置

                if new_head == self.game.food_pos:
                    self.game.score += 1
                    self.reward=1
                    self.previous_score_step=self.steps
                    self.game.food_pos = self.game.generate_food()  # 生成新的
食物
                else:
                    self.game.snake_pos.pop()  # 移除蛇尾
                    if self.steps - self.previous_score_step >
len(self.game.snake_pos)*2*self.max_tolerance_step:
                        self.reward = -0.05
                        self.game.game_over = True
                        print("for oversteps")
                    elif self.steps- self.previous_score_step >
len(self.game.snake_pos)*self.max_tolerance_step:
                        self.reward = -0.01
                    else :
```

```
                    self.reward = 0

    def step(self, action):
        if action == 0 and self.game.snake_direction != (0,1):  # 向上
            self.game.snake_direction = (0, -1)
        elif action == 1 and self.game.snake_direction != (0,-1):  # 向下
            self.game.snake_direction = (0, 1)
        elif action == 2 and self.game.snake_direction != (1,0):  # 向左
            self.game.snake_direction = (-1, 0)
        elif action == 3 and self.game.snake_direction != (-1,0):  # 向右
            self.game.snake_direction = (1, 0)

        food_distance = math.dist(self.game.snake_pos[0],self.game.food_pos)

        self.move_snake()
        self.steps += 1
        self.total_steps += 1
        self.total_reward += self.reward
        self.render()

        state = self.get_state()

        done = self.game.game_over

        info = {"score": self.game.score}

        return state, self.reward, done, False, info  # 新接口需要返回 `done`
和 `truncated`
```

```
# def get_state(self):
#     game=self.game
#     head = game.snake_pos[0]
#     point_l = (head[0] - 1, head[1])
#     point_r = (head[0] + 1, head[1])
#     point_u = (head[0], head[1] - 1)
#     point_d = (head[0], head[1] + 1)

#     dir_u = game.snake_direction == (0, -1)
#     dir_d = game.snake_direction == (0, 1)
#     dir_l = game.snake_direction == (-1, 0)
#     dir_r = game.snake_direction == (1, 0)

#     state = [
#         # Danger straight
#         (dir_r and game.is_collision(point_r)) or
#         (dir_l and game.is_collision(point_l)) or
#         (dir_u and game.is_collision(point_u)) or
#         (dir_d and game.is_collision(point_d)),

#         # Danger right
#         (dir_u and game.is_collision(point_r)) or
#         (dir_d and game.is_collision(point_l)) or
#         (dir_l and game.is_collision(point_u)) or
#         (dir_r and game.is_collision(point_d)),

#         # Danger left
#         (dir_d and game.is_collision(point_r)) or
#         (dir_u and game.is_collision(point_l)) or
```

```
    #         (dir_r and game.is_collision(point_u)) or
    #         (dir_l and game.is_collision(point_d)),

    #         # Move direction
    #         dir_l,
    #         dir_r,
    #         dir_u,
    #         dir_d,

    #         # Food location
    #         game.food_pos[0] < head[0],  # food left
    #         game.food_pos[0] > head[0],  # food right
    #         game.food_pos[1] < head[1],  # food up
    #         game.food_pos[1] > head[1],  # food down
    #         (game.food_pos[0] - head[0])/22,
    #         (game.food_pos[1] - head[1])/22
    #         ]

    #     return torch.tensor(state,dtype=torch.float32)
    def get_state(self):
        head=self.game.snake_pos[0]
        [xhead, yhead] = [head[0], head[1]]
        [xfood, yfood] = [self.game.food_pos[0], self.game.food_pos[1]]
        deltax = (xfood - xhead) / self.game.X_BLOCK_NUM
        deltay = (yfood - yhead) / self.game.Y_BLOCK_NUM
        checkPoint = [[xhead,yhead-1],[xhead-
1,yhead],[xhead,yhead+1],[xhead+1,yhead]]
        tem = [0,0,0,0]
        for coord in self.game.snake_pos[1:-1]+self.game.obstacle_pos:
```

```
            if [coord[0],coord[1]] in checkPoint:
                index = checkPoint.index([coord[0],coord[1]])
                tem[index] = 1
        for i,point in enumerate(checkPoint):
            if point[0]>=self.game.X_BLOCK_NUM or point[0]<0 or
point[1]>=self.game.Y_BLOCK_NUM or point[1]<0:
                tem[i] = 1
        state = [deltax,deltay]
        state.extend(tem)
        return state
    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    self.show_snake= not self.show_snake
                if event.key == pygame.K_DOWN:
                    if self.clock_tick == 5000:
                        self.clock_tick = 10
                    elif self.clock_tick == 10:
                        self.clock_tick = 5000
    def render(self, mode="human"):
        # self.game.handle_events()
        self.game.handle_events()

        self.game.game_display.fill(self.game.BG_COLOR)
```

```
        self.game.draw_grid()
        self.game.draw_food()
        self.game.draw_obstacle()
        self.game.draw_snake()
        self.game.display_score()
        pygame.display.update()
        self.game.clock.tick(self.SPEED)

    def close(self):
        pygame.quit()

if __name__ == "__main__":
    check_env(SnakeGymEnv)
```

PPO_train.py

```
import gymnasium as gym
from stable_baselines3 import PPO
from stable_baselines3.common.callbacks import CheckpointCallback
from Env4PPO import SnakeGymEnv

def linear_schedule(initial_value, final_value=0.0):
    if isinstance(initial_value, str):
        initial_value = float(initial_value)
        final_value = float(final_value)
        assert (initial_value > 0.0)

    def scheduler(progress):
        return final_value + progress * (initial_value - final_value)
```

```
    return scheduler


def PPO_train():
    env = SnakeGymEnv()
    lr_schedule = linear_schedule(1.5e-3, 2.5e-4)
    clip_range_schedule = linear_schedule(0.150, 0.025)

    model = PPO(  "MlpPolicy",
                env,
                device="cuda",
                verbose=1,
                n_steps=2048,
                batch_size=512,
                n_epochs=4,
                gamma=0.94,
                learning_rate=lr_schedule,
                clip_range=clip_range_schedule
                )



    model.learn(
        total_timesteps=int(2000000),
    )

if __name__ == "__main__":
    PPO_train()
```