



基于腾讯开悟教学平台采用 PPO 算法进 行的智能体决策 1V1 实验

课程名称： 神经网络和深度学习
授课教师： 赵卫东
实验小组： DayDayUp
张广东 24262010055
余盛龙 24262010053
小组成员： 许丽瑄 24262010046
甘成武 24262010010
黄正宇 24262010018

目录

一 实验目的	2
二 实验环境及实验要求	2
2.1 实验环境描述	3
2.2 实验要求	14
三 设计思想	15
3.1 PPO 算法	15
3.2 奖励函数	17
3.3 学习策略	27
四 实验结果与分析	28
4.1 基准实验	28
4.2 修改奖励函数	30
4.3 修改学习策略	30
五 结论	33
5.1 PPO 学习方法的优缺点	33
5.2 可能的优化与改进方向	34

一、实验目的

强化学习（Reinforcement Learning, RL）是一种机器学习方法，智能体通过与环境的交互来学习如何最大化某种累积奖励的最佳决策。基于策略的强化学习，其目标是找到一种策略（Policy），使得在环境中的行为能够最大化累积奖励。策略是一个映射状态到行为的函数，它直接决定了在给定状态下智能体应该采取的行为。

REINFORCE 算法是策略梯度方法的典型代表，智能体根据当前策略和环境交互，通过采样得到的轨迹数据直接计算出策略参数的梯度，进而更新当前策略，使其向最大化期望回报的目标靠近。基于策略的强化学习比基于价值的强化学习算法的优化目标（一般是时序差分误差的最小化）要更加直接。REINFORCE 算法理论上是能保证局部最优的，它实际上是借助蒙特卡洛方法采样轨迹来估计动作价值，这种做法的一大优点是可以得到无偏的梯度。但是，因为使用了蒙特卡洛方法，REINFORCE 算法的梯度估计的方差很大，可能会造成一定程度上的不稳定。

TRPO (Trust Region Policy Optimization) 一种策略梯度方法，它在策略梯度的基础上引入了约束条件，以确保策略的变化在一定范围内并且是单调变化的。TRPO 的目标是在满足约束条件的前提下，找到能够最大化累积奖励的策略。TRPO 方法通过对策略梯度进行优化，以实现策略的更新。

PPO (Proximal Policy Optimization) 是 TRPO 算法的改进版。PPO 主要有两种形式：一种是 PPO-惩罚，通过 KL 散度的惩罚项来限制更新；另一种是 PPO-截断，即通过截断比值的方式来限制策略的变化。常用的是后者，也就是 PPO-clip，它通过裁剪概率比来防止过大的更新，这样避免了计算二阶导数，用一阶优化方法就能实现，更易于实现和应用。

实验的目的是理解基于策略梯度的强化学习方法，验证 PPO 算法在稳定性和计算速度方面的改进，测试超参数的敏感性，观察移除截断后策略发生的崩溃现象，并通过修改价值回报理解 PPO 的应用场景和局限性。

二、实验环境及实验要求

本次实验在腾讯开悟教学平台中进行。开悟平台中提供了机器学习所需硬件平台，基于王者荣耀建立的智能体模拟环境，网页版的代码开发调试环境，以及训练管理和评估环境。

2.1 实验环境描述

本实验通过算法训练一个智能体，在地图获取经济和经验来提高英雄的攻击力，率先摧毁对方的水晶以获得胜利。若己方水晶被摧毁，则对方获得胜利。

2.1.1 问题描述

腾讯开悟教学平台中的智能体决策 1V1 实验，基于游戏王者荣耀建立，问题要求通过算法训练一个智能体，让其在游戏回合中学习游戏操作策略，通过击杀对方角色、士兵和野怪积累经济和经验，拆毁水晶塔取得胜利。其中经济用于购买角色装备，提升攻击力和抗性；经验用于提升角色等级，增加角色攻击力和抗性，以及提高技能的参数。



图 1 地图

智能体决策 1V1 的地图如 图 1 所示。地图主体为墨家机关直道，从左下到右上建筑依次为：我方泉水法阵，我方水晶，我方防御塔，中央战场，敌方防御塔，敌方水晶，敌方泉水法阵。其中泉水法阵为英雄出生和复活地点，泉水法阵会定期刷新双方士兵。防御塔会攻击进入一定范围内的敌方阵容，防御塔下会生成治疗符文，英雄拾取治疗符文后回复一定量的生命值，治疗符文被拾取后经过固定的冷却时间刷新。治疗符文不区分阵容，治疗符文未拾取时不会生成多个。中央广场的丛林中会定期刷新丛林野怪，丛林野怪没有阵营。丛林野怪会跟随并攻击最后一个攻击自己的角色，丛林野怪有固定的巢穴范围，跟随攻击目标不会超出巢穴范围并且有最大跟随间距，超出巢穴范围或超出最大跟随间距时放弃跟随并返回初始位置，同时清除对攻击目标的记忆。游戏中需要拆毁防御塔之后的水晶取得胜利。实验中进行了简化游戏回合，在拆毁对方防御塔后判定取胜。

实验平台中提供三个英雄角色，每个英雄角色有一个被动技能和三个不同的主动技能，被动技能会自动触发，主动技能需要智能体采取动作才会触发。不

同英雄的技能不同，此外每个英雄还有通用的闪现技和召回技能，以及最基础的普通攻击和移动。

技能	描述
空	未定义
空	未定义
移动	移动英雄
普通攻击	释放普通攻击
技能一	释放英雄技能一
技能二	释放英雄技能二
技能三	释放英雄技能三
治疗技能	释放治疗技能
自选技能	释放自选技能：闪现
召回	经过一定时间后传送英雄回到泉水，攻击或者操作会被打断
技能四	释放技能四，英雄技能需要的辅助操作
装备技能	释放特定装备提供的技能

表 1 技能列表

在智能体决策 1V1 实验中，英雄的动作被分成三个维度来定义：种类，方位，目标。动作种类包括：移动，攻击，三个英雄主动技能，两个通用技能闪现和召回，其定义如 表 1 所示。其次是动作方位，包括移动方位和技能释放方位。动作方位是否有效取决于选定的动作，需要按照动作类型进行过滤。通常动作方位由方向和强度两部分组成，移动方位直接用 16x16 的矩阵在直角坐标中表示，技能释放方位用目标和 16x16 的偏移矩阵表示，如 图 2 所示。最后是动作目标，单体的攻击和治疗动作通常仅需要指定对象，而范围技能的方位依赖与动作目标和偏移。

游戏回合中，视野是跟随英雄移动的，如 图 3 所示，双方英雄看到的视野不同。游戏引擎会返回当前帧的状态数据，从中可以获取到英雄血量、技能信息、防御塔信息等数值，如 图 3 所示。观测到的特征经过视野过滤，观测数据中仅提供英雄视野中的数据，当敌方英雄不在视野中时，敌方英雄部分特征会被置为默认值。环境中观测到的特征信息如 表 2 所示。主要包括英雄状态、非英雄角色状态、子弹装备物品等信息。

智能体决策 1V1 实验中游戏回合的目标是摧毁敌方水晶（实际设定是防御

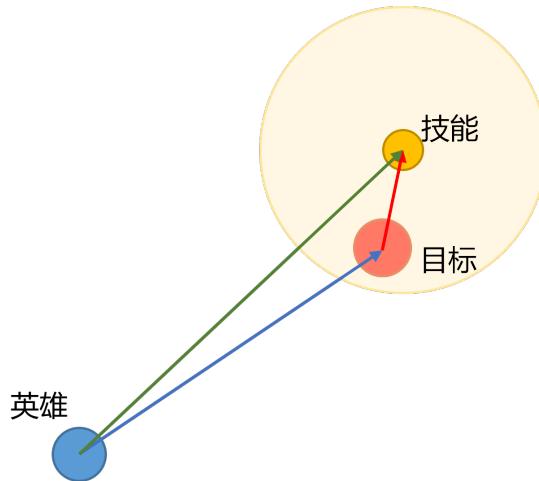


图 2 技能释放方位的计算

字段名	字段类型	备注
frameNo	signed int	帧号
hero_states	vector, HeroState	英雄状态列表
npc_states	vector, ActorState	非英雄角色状态列表
bullets	vector, Bullet	子弹列表
cakes	vector, Cake	功能物件列表 (血包等)
equip_infos	vector, EquipInfo	装备信息列表
frame_action	vector, FrameAction	死亡事件

表 2 状态信息

塔), 除此之外不计算任何得分, 游戏结果只有胜负, 最终结果按照胜负比例计算, 超时不计算胜负。游戏回合中击杀对方阵容角色或者从林野怪, 通过提升己方攻击能力和削弱对方攻击能力, 来间接的影响胜负。

2.1.2 环境配置

智能体决策 1V1 实验中需要配置的信息包括敌方模型和双方英雄。为了统一, 通常选定蓝方进行算法的训练和评估, 并进行过程监控。对局模式中的红方, 可以指定基础 AI 模型或者其他特定的智能体模型, 也可以指定为自对局模式, 即红蓝双方都使用同一个智能体进行训练和评估。每个算法都可以自定义环境配置, 通过 user_conf 字典实现, 格式为:

```

1 usr_conf = {
2     "diy": {
3         "monitor_side": 0,

```

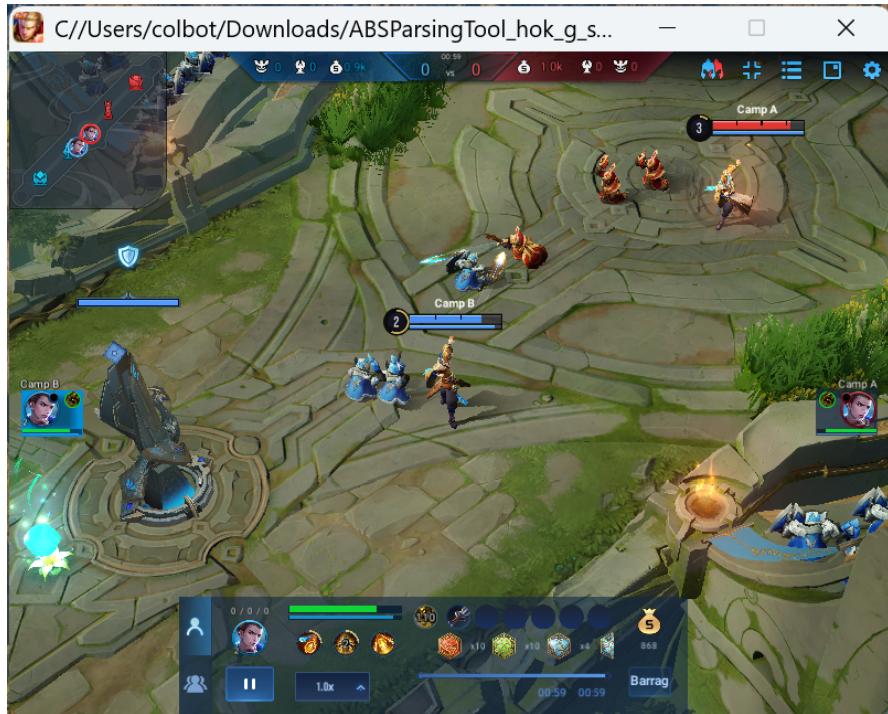


图 3 游戏画面

```
4     "monitor_label": "common_ai",
5     "lineups": [[{"hero_id": 133}], [{"hero_id": 508}]],
6   }
7 }
```

配置字典中各个字段信息解释如下：

1. monitor_side：选择的阵营，统计指标以该阵营为第一视角，0 为蓝方阵营，1 为红方阵营
2. monitor_label：对手配置，三种类型：
 - (a) "selfplay"，自对弈
 - (b) "common_ai"，对手采用 common_ai
 - (c) 自定义的模型 id，如“1234”，对手选用指定模型
3. lineups：阵容配置，分别代表蓝方和红方英雄 id,hero_id 范围:[133, 199, 508]

由于智能体决策 1V1 的评估中红蓝双方的英雄都仅支持 133 狄仁杰，因此将 monitor_side 为 0，同时设置 lineups 为[[{"hero_id":133}], [{"hero_id":133}]]。在训练中仅需要修改 monitor_label 来选择对局的对手模型即可。

2.1.3 代码框架

智能体决策 1V1 实验提供了基础的代码框架，包含已经实现的 PPO 算法，以及一个可以自己实现的 diy 算法。以 PPO 算法为例，各代码文件作用为：

文件	用途
train_test	调试入口
ppo/config	算法配置文件
ppo/train_workflow	算法流程
ppo/algorithm/agent	智能体实现
ppo/feature/definition	状态观测和状态变化检测
ppo/feature/reward_manager	计算奖励
ppo/model/model	神经网络模型

其他与环境交互的代码包被隐藏防止修改，以保证游戏回合的公平性。实验者无需关心和环境交互的实现部分。

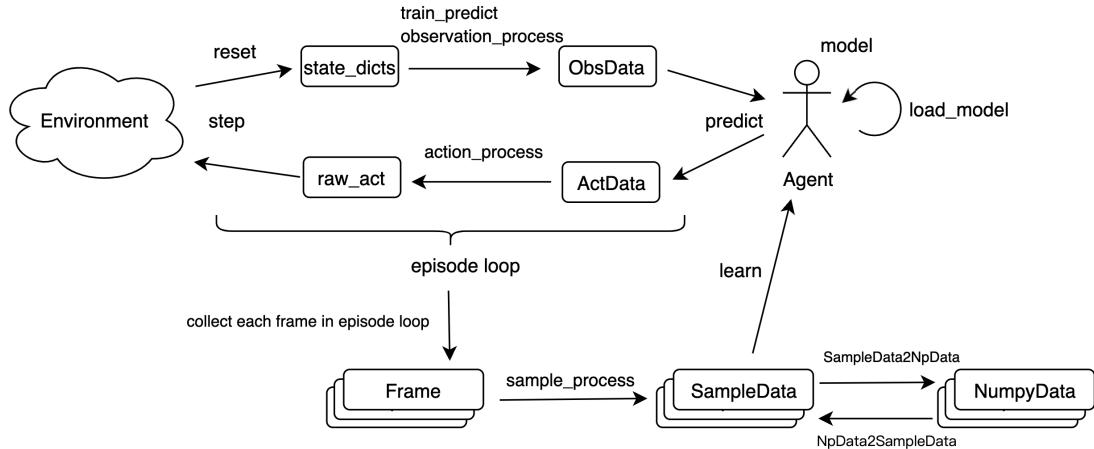


图 4 智能体训练流程

实验中的智能体包含一个可被训练的模型，智能体可以对环境给出的观测进行决策，这个决策作用于环境产生新的观测，此过程通过训练流程控制，不断循环。训练流程还要收集循环过程中产生的每一帧数据，将他们组合成样本数据，智能体可以根据这些样本作为算法的输入，通过算法更新模型。由于 Hok1v1 采用分布式训练，会启动多个容器，样本需要通过网络通信发送到训练容器（learner）中进行训练，所以需要对样本进行编码方便网络发送，另外智能体需要将 learner 容器上的模型同步回来。以上任务描述如 图 4 所示。

平台上提供的代码框架中已经实现了一个版本的 PPO 算法，并且可以运行训练得到结果。基于已有的 PPO 算法，可以在 diy 目录中自己实现一个智能体，然后进行训练和评估。由于平台的限制，代码目录中仅能识别 ppo 和 diy 算法目录，不支持新增或自定义其他算法实现。

2.1.4 模型训练

The screenshot shows the 'Training Management' section of the experimental control panel. It displays two training tasks:

- DIY-gd-0001** (Running): Task ID #37406, Version V14.2.57. It has a runtime of 30min (10h) and is using DIY distributed mode. Description: Both parties chose Ding Junjie as the opponent,对手为common_ai, 基础训练.
- PPO-gd-0001** (Automatic Release): Task ID #37231, Version V14.2.57. It has a runtime of 10h (10h) and is using PPO distributed mode. Description: Basic model, no changes made.

图 5 训练管理界面

模型训练在实验的控制台主页上，通过左侧菜单选择训练管理标签进入。在代码修改并测试结束后，通过新增训练任务，使用最新的代码进行模型训练。训练管理界面如 图 5 所示。

The dialog box is titled 'Add Training Task'. It contains the following fields:

- Task Name:** DIY
- Use Pre-trained Model:** (radio button selected)
- Pre-trained Model:** PPO-gd-0001_33280
- Algorithm:** PPO
- Training Mode:** Distributed
- Training Steps:** 33280
- Training Duration:** 10h27s
- Task Configuration:** (Algorithm, Training Mode, and Pre-trained Model remain consistent)
- Algorithm:** PPO
- Training Mode:** Distributed
- Duration:** (Maximum 10h) Input fields for hours (请输入) and minutes (请输入).

图 6 新增训练任务

在新增训练任务的界面可以选择使用的算法实现。智能体决策 1V1 实验的框架支持加载已经训练好的模型继续进行训练。最长训练时间为 10 小时，需要

合理的设计网络模型，调整学习率和奖励函数，注意模型的收敛时间。新增训练任务的界面如 图 6 所示，可以选择从头开始训练或者从一个基础模型开始训练。

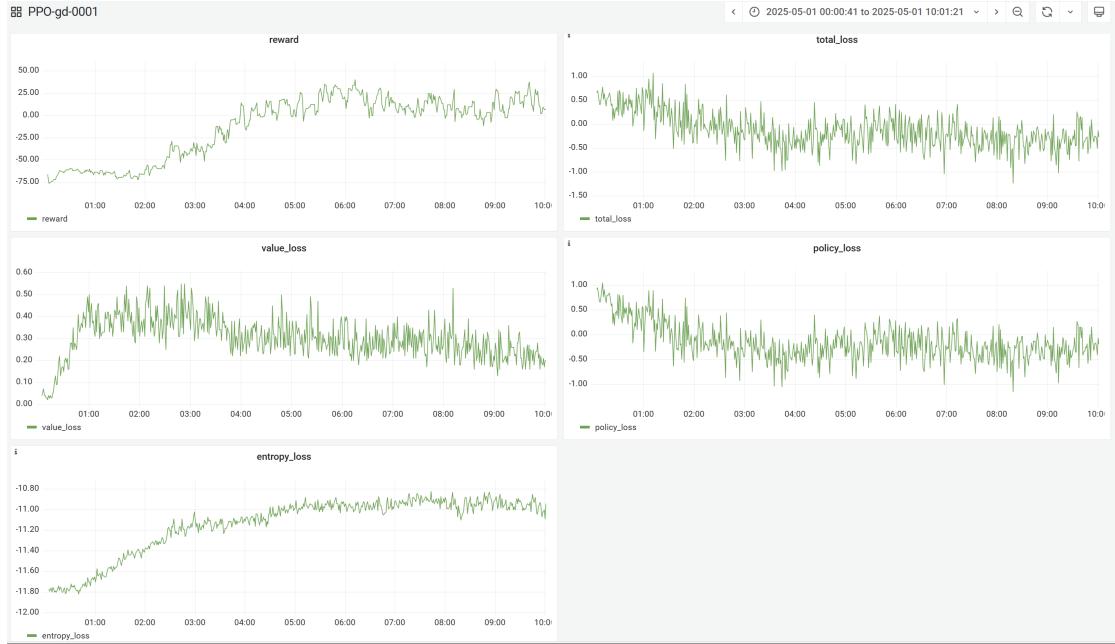


图 7 训练监控界面

模型的训练过程可以提前中止。模型训练过程中每隔一定时间会保存一个模型文件。在模型训练的监控界面，可以看到模型训练过程中的数据变化，监控模型是否收敛以及收敛速度是否符合预期，可以提前中止模型训练以节省时间。训练监控界面如 图 7 所示。默认配置的模型保存时间为 30 分钟。初始训练时间较长，模型的保存时间间隔可以较长防止占用过多的磁盘空间。后期基于已有模型进行训练时，训练时间较短，可以采用较短的保存间隔。

× 模型列表				
PPO-gd-0001				
仅保存最近7天内生成的模型，请及时查看				
训练时长	训练步数	大小	操作	
10h27s	33280	10.53M	下载	提交至模型管理
9h52min24s	32833	10.52M	下载	提交至模型管理
9h21min44s	31125	10.53M	下载	提交至模型管理

图 8 训练过程模型列表

在模型训练任务的模型列表界面可以看到训练过程中保存的所有模型，可以根据监控数据找到对应的模型，并提交到模型管理中以备后用，防止模型被清理。模型列表如 图 8 所示。

2.1.5 模型管理

模型	大小	算法	训练模式	训练步数	训练时长	操作
PPO-gd-0001_33280 检测成功 #80446	10.53M	PPO	分布式	33280	10h	新增评估 新增训练 ...
0426ppo_33346 检测成功 #79620	10.58M	PPO	分布式	33346	10h	新增评估 新增训练 ...
0420ppo_18984 检测成功 #79542	10.58M	PPO	分布式	18984	5h44min	新增评估 新增训练 ...

图 9 模型管理界面

模型管理菜单中显示所有已经提交至模型管理中的模型，提交到模型管理中可以保证模型不会被清理。模型管理中的模型可以进行评估和下载操作。模型提交到模型管理中后，首先会进行检测，检测成功后才可以进行评估。模型管理界面如 图 9 所示。

2.1.6 模型评估

任务名称	输入任务名称	模型	状态	版本	更多筛选
基准测试 已完成 #221016 小组可见 V14.2.57		阵营模型 PPO-gd-0001_33280 狄仁杰 10h27s PPO 分布式	完成	V14.2.57	查看详情
DayDayUp (张广东) 提交于 2025-05-01 15:06:21		阵营模型 0426ppo_33346 狄仁杰 10h26s PPO 分布式	未开始		
0426ppo_02 已完成 #219989 小组可见 V14.2.57		阵营模型 0426ppo_33346 狄仁杰 10h26s PPO 分布式	完成	V14.2.57	查看详情
DayDayUp (黄正宇) 提交于 2025-04-26 23:17:27		阵营模型 0420ppo_18984 狄仁杰 5h43min46s PPO 分布式	未开始		

图 10 模型评估管理界面

模型评估是对已经训练结束的模型运行回合查看结果。评估任务提交后会进行排队等待运行，运行结束后可以查看结果。模型评估任务管理界面如 图 10 所示。

添加模型评估任务界面如 图 11 所示。添加模型评估任务时可以配置的参数包括：任务名称，双方英雄，双方智能体模型，对战回合数量。其中，英雄只能选择狄仁杰，双方智能体模型可以选择模型管理中的任意模型，不支持 common_ai



图 11 添加模型评估任务

模型。智能体决策 1V1 实验中不提供通用或者公用的模型，因此需要找到一个合理的评估基准，此处选用原始版本代码训练得到的模型作为基准。评估会作为红蓝阵营分别进行，最大选择评估回合数为 5，则作为红蓝阵营分别进行 5 个回合，总计 10 个游戏回合。由于地图的对称性，理论上作为红蓝双方阵营应该是没有区别的。

在模型评估管理界面看可以看到评估任务的具体配置信息，如 图 12 所示。包含对战双方选择的模型信息和英雄信息。

在模型评估结果的详情页面中，可以看到本次评估任务的具体信息。如 图 13 所示。该页面展示了游戏回合中通常需要关注的信息，如经验、经济、击杀和死亡等信息。需要注意的是，在智能体决策 1V1 的实验中，智能体操纵的英雄发育成长的优劣通常通过经验、经济、击杀和死亡这些信息来评估，但游戏的胜负是按照拆毁水晶计算。英雄的发育成长为拆毁水晶提供了基础。

在模型评估任务的详情页面，可以观看游戏回合的快速预览。预览采用俯视图的模式，显示完整的游戏地图，如 图 14 所示。其中红色和蓝色的点代表双方阵营的士兵和英雄，较大的点代表英雄。快速预览中可以观察到英雄和士兵在



图 12 模型评估任务配置

地图中的位置分布。更具体的游戏信息可以通过下载录像的方式进行查看。

在模型评估任务的详情页中，可以下载评估任务每个游戏回合的回放。观看游戏回放前需要先下载安装游戏回放的播放器，然后将下载的游戏回放文件复制到 auto_replay 文件夹中，执行 SGame 程序，自动开始回放。下载得到的播放器如 图 15 所示。播放过程中不能控制播放进度和速度。播放完成后会自动删除录像文件，因此在播放完成前可以结束播放器，避免录像文件被删除。游戏录像回放界面如 图 3 所示，可以通过鼠标滚轮缩放视角或者通过按键移动画面中心位置，以查看英雄角色的具体表现。

2.1.7 分析调试

开悟平台提供了基于网页版 vscode 的在线开发和调试环境。通过实验右上角的“进入开发”按钮可以打开代码开发调试环境。代码开发调试环境空闲 10 分钟会被回收。开发调试环境的操作和标准的 vscode 基本一致。在开发调试环境中修改代码保存后，就可以到训练管理界面采用最新的代码进行训练。训练开始时会首先对当前的代码打包生成快照，打包过程中会进行文件大小的检测，如果有较大的文件会打包失败。编写代码时应该注意拆分文件保证大小符合要求。

代码开发环境同时还可以进行调试。调试脚本的入口为 train_test.py 函数。由于代码中目前无法加载模型，调试时只能加载一个初始化的模型从头进行训

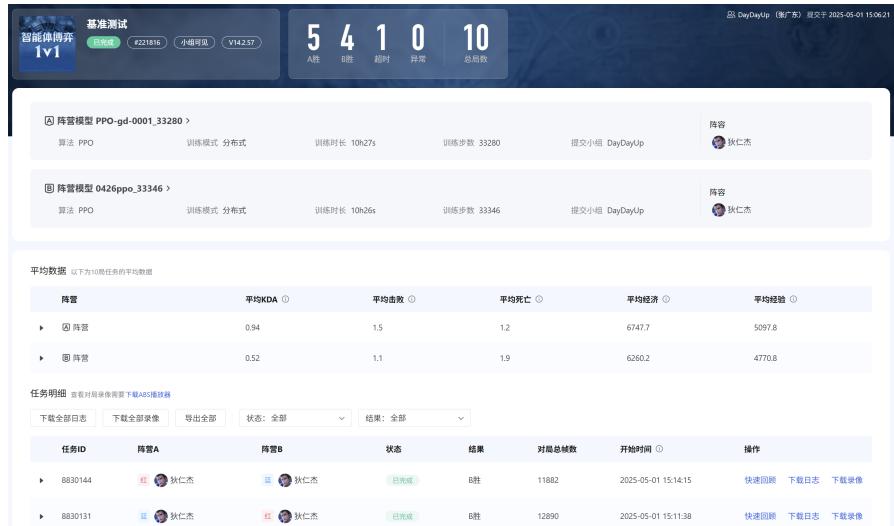


图 13 模型评估结果列表

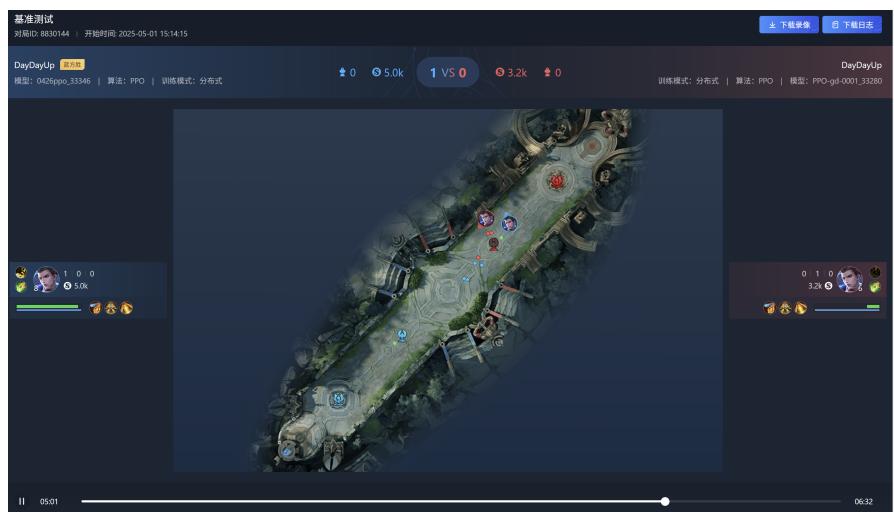


图 14 模型评估游戏回合快速预览

练。开发环境中已经配置好了调试的配置文件，从左侧功能栏切换到到调试界面直接运行即可。调试时需要注意，train_test.py 脚本属于每次启动开发环境时通过模板创建，该文件中更改的代码在开发环境关闭后会丢失。每次进入开发环境调试时需要确认采用的算法名称是需要进行调试的算法。调试时在终端区域可以看到日志输出，调试功能和标准 vscode 基本一致。需要注意开发环境是每次启动时生成的，只有 ckpt, conf, diy, ppo, log 这几个目录是链接到用户的存储目录中，因此开发环境无法用 git 进行代码版本管理，修改代码时注意保存。

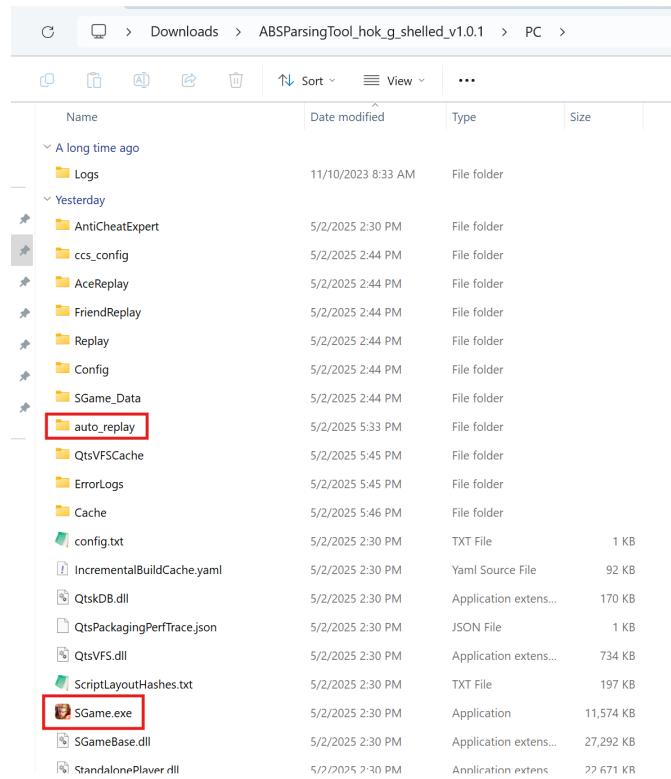


图 15 游戏录像播放器

2.2 实验要求

2.2.1 掌握深度强化学习的基本知识

传统的强化学习通常使用表格方法或线性函数，适用于状态空间较小、定义明确的问题。深度强化学习通过使用深度神经网络扩展了强化学习，使其能够处理像图像这样的高维度、非结构化输入，并学习更复杂的策略。虽然深度强化学习使用的是深度学习模型，但其学习机制却有本质区别。大多数深度学习应用涉及监督学习（从有标签的数据中学习）或无监督学习（在无标签的数据中寻找模式）。深度强化学习通过互动和反馈（奖励）进行学习，每一步都没有明确的正确操作标签。深度强化学习和监督学习、无监督学习也有区别。监督学习要求数据集具有正确的输入输出对（训练数据）。无监督学习寻求数据的内在结构。深度强化学习在互动环境中运行，通过基于标量奖励信号的探索和利用，学习最佳行为。

2.2.2 理解基于策略的强化学习方法

强化学习算法分为基于值和基于策略两种方式。基于策略是指，输入环境状态，直接输出各个动作的概率或动作强度。训练刚开始时，动作概率分布没有明显差异，趋向于随机探索。策略梯度算法的思想是先将策略表示成一个和奖励有

关的连续函数，然后用连续函数的优化方法去寻找最优的策略，优化目标是最大化连续函数，最常用的是优化方法是梯度上升法（与最小化 loss 的梯度下降相对）。

PPO 算法属于策略优化算法的一种改进型。传统方法可能存在训练过程中策略更新过大导致的不稳定性问题，而 PPO 通过限制策略更新的幅度来解决这一点。PPO 算法基于 TRPO 算法可信区域的思想，但是其算法实现更加简单。大量的实验结果表明，与 TRPO 相比，PPO 能学习得一样好（甚至更快），这使得 PPO 成为非常流行的强化学习算法。如果我们想要尝试在一个新的环境中使用强化学习算法，那么 PPO 就属于可以首先尝试的算法。PPO 算法限制学习策略的方式有两种，惩罚和截断，通常选用截断。

2.2.3 理解并应用 PPO 算法

PPO 算法是对 TRPO 算法的改进，TRPO 算法是对 Actor-Critic 算法的优化。PPO 算法包含两个网络，策略网络和价值网络。PPO 算法的流程描述如下：

```
1 初始 化策 略网 络 和 价 值 网 络
2 循 环 直 到 收 敛：
3   收集 轨 迹数 据：使 用 当 前 策 略 与 环 境 交 互 N 步
4   计 算 优 势 估 计：通 过 GAE 方 法 计 算 各 状 态 的 优 势 值
5   优 化 阶 段 (执 行 K 个 回 合)：
6     随 机 小 批 量 采 样 数 据
7     计 算 新 策 略 概 率 比
8     计 算 策 略 损 失
9     计 算 价 值 损 失
10    计 算 熵 损 失
11    联 合 更 新
12    定 期 同 步 旧 策 略 参 数
```

三、设计思想

3.1 PPO 算法

在智能体决策 1V1 实验中，环境产生的原始观测数据不能直接作为智能体模型的输入，并且不同的用户开发的智能体一般是不一样的，显然不同的智能体的决策、学习方法的输入输出也是不一样的，所以应该先定义智能体模型输入输出的数据结构。包括特征 (ObsData)、动作 (ActData)、样本 (SampleData)，其中 ObsData 和 ActData 分别作为智能体 predict 方法的输入和输出，SampleData 作为智能体 learn 方法的输入。此外需要实现一个智能体，能够处理 ObsData 和 ActData 类型，由于可能会定义不同的数据结构，同时环境接口输入输出的数据

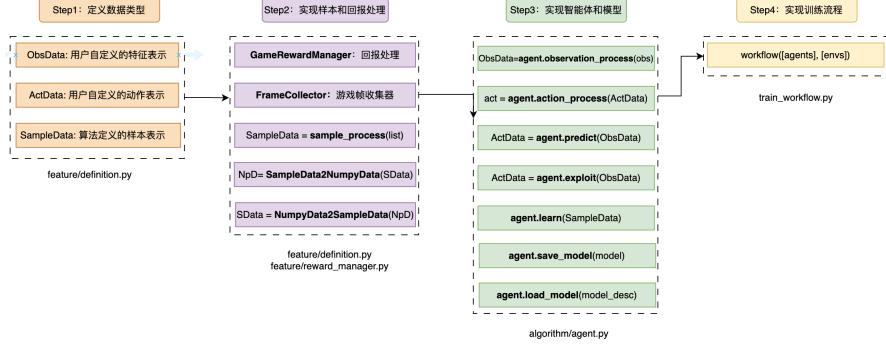


图 16 代码结构及功能

结构是固定的，因此环境接口的输入输出数据和智能体接口的输入输出数据需要进行转换，所以还需要实现这些数据结构的转换方法，包括：observation_process 和 action_process。此外，智能体还需要包含一个模型（一般是神经网络模型），智能体负责与环境交互，产生预测动作并消费样本来训练模型。在实现了 数据结构，数据处理函数，模型和 智能体 以及回报设计等之后，还需要实现一个强化学习的训练流程 workflow，将所有组件组合起来完成强化学习训练，即智能体通过不断的与环境交互，获取样本数据，更新并迭代模型，直到模型收敛到期望的效果。平台中提供的代码框架各部分功能如 图 16 所示。

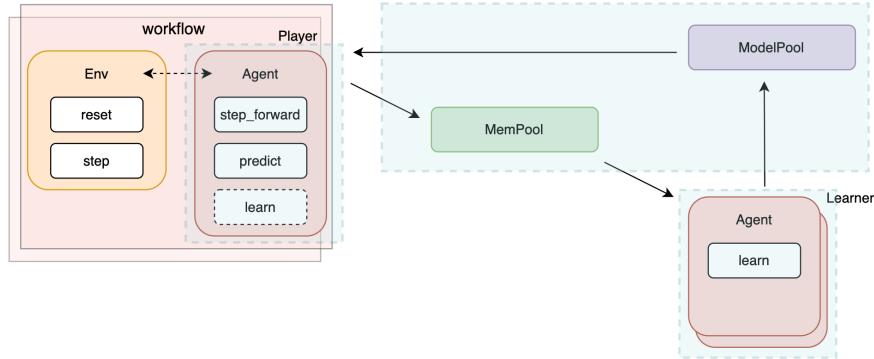


图 17 分布式训练架构

开悟平台采用分布式训练，在训练开始后启动一个样本池，一个模型同步服务。分布式训练架构如 图 17 所示。学习容器会定期保存模型，`agent.learn(samples)` 调用将会把样本发送到样本池，训练容器会从样本池中采样样本 samples 将其传入 `agent.learn(samples)` 进行训练，此过程是自动的，用户无需开发额外代码。智能体决策 1V1 实验虽然提供了两个智能体，但由于在 selfplay 模式下红蓝双方的两个智能体是同构的，所以只需训练一个模型，即训练容器（learner）中仅有一个模型实例，所以样本池也只有一个，两个智能体调用 `agent.learn()` 将发

送样本到同一个样本池；同样地，模型同步服务也只同步一个模型，用户可以按需在恰当时机从模型同步服务加载模型。用户可以为两个智能体加载不同的历史模型，调用 `agent.load_model(id="latest")` 将会加载最新模型，若希望加载随机模型则调用 `agent.load_model(id="random")`，若训练过程希望加载某个模型用于评估则可以指定模型 id。

环境返回的观测信息 `obs` 对应与 PPO 算法中的状态 `s`。开悟平台的实验环境中已经对这部分数据做了预处理，但是智能体需要的状态信息 `s` 和环境观测信息 `obs` 之间存在一定差异，这部分由 `observation_process` 函数将环境提供的观测信息转换为智能体需要的状态信息。同理，`action_process` 函数用于将智能体选择的动作转换为环境可以识别的动作。`sample_process` 用于将环境交互数据和奖励打包成样本数据，用于经验回放。在分布式训练中，样本数据需要通过网络传输至样本池，因此需要对 `SampleData` 进行序列化和反序列化，此处应该注意序列化编码和解码的对应，防止数据出错。

3.2 奖励函数

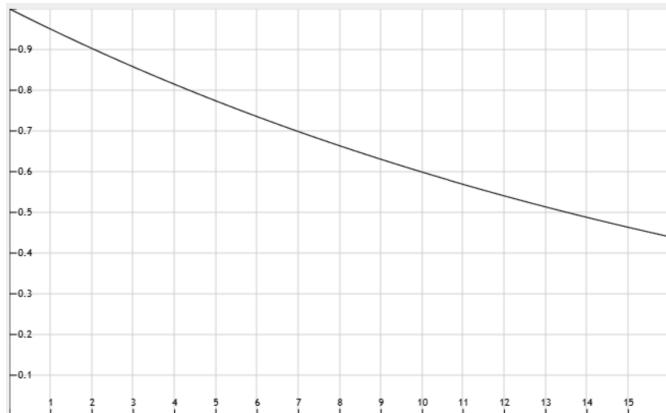


图 18 英雄等级衰减因子

智能体决策 1V1 实验的游戏回合状态比较复杂，因此其奖励函数的设计需要考虑的因素也比较多。考虑游戏中英雄的奖励应该是对拆毁敌方水晶有帮助的因素，主要包括：经济，经验，生存状态，攻击状态，英雄位置，技能使用的条件，空闲时的行为，以及对获胜条件的考量。

考虑到对战回合中随着游戏的进行和英雄等级的提升，同等金钱和经验对英雄的提升逐渐降低，构造一个随着英雄等级变化的动态因子 `level_factor` 来处理奖励随英雄等级变化的衰减。此处选用幂函数即可，衰减因子定义为：

$$f = 0.95^x \quad (1)$$

英雄共有 15 个等级，衰减因子曲线如 图 18 所示。从开局的 1.0 衰减到 15 级时的约 0.46。

3.2.1 状态检测

智能体决策 1V1 是对战游戏，因此在做状态检测时应该考虑敌我双方的状态对比，而不仅是状态的绝对值。代码框架中使用 GameRewardManager 来管理所有奖励，从游戏的帧状态信息 frame_data 提取所有需要关注的数据，并对比前后两帧的数据变化和敌我双方的数据对比。

随着游戏进行，士兵的生命和攻击强度会逐步提升，英雄的生命和攻击输出也会逐渐增加，在游戏回合中需要关注双方士兵和英雄的状态，用于动态计算奖励。从 frame_data 中获取敌我双方英雄的代码如下所示：

```

1 main_hero, enemy_hero = None, None
2 hero_list = frame_data["hero_states"]
3 for hero in hero_list:
4     hero_camp = hero["actor_state"]["camp"]
5     if hero_camp == camp:
6         main_hero = hero
7     else:
8         enemy_hero = hero

```

敌方英雄不在视野中时，敌方英雄的数据会是默认值，需要进行区分。其次是双方防御塔的状态，获取状态的代码如下：

```

1 main_tower, main_spring = None, None
2 enemy_tower, enemy_spring = None, None
3 npc_list = frame_data["npc_states"]
4 for organ in npc_list:
5     organ_camp = organ["camp"]
6     organ_subtype = organ["sub_type"]
7     if organ_camp == camp:
8         if organ_subtype == "ACTOR_SUB_TOWER": # 21 is tower
9             main_tower = organ
10        elif organ_subtype == "ACTOR_SUB_CRYSTAL": # 24 is
11            crystal
12            main_spring = organ
13        else:
14            if organ_subtype == "ACTOR_SUB_TOWER": # 21 is tower
15                enemy_tower = organ
16            elif organ_subtype == "ACTOR_SUB_CRYSTAL": # 24 is
17                crystal
18                enemy_spring = organ

```

3.2.2 经济奖励

游戏回合中，英雄的经济主要表现为获得的金钱数据，金钱用于购买装备提升生命值、攻击输出和伤害抗性。英雄的装备购买列表和顺序已经固定，当金钱足够时会自动进行购买，无需关注金钱的使用。智能体决策 1V1 实验中设置了特殊模式，购买装备只计算已经获得金钱总额，不会进行扣减。同时，注意到同等经济在回合初期和回合后期对英雄的提升有区别，回合后期英雄本身具有较高的属性，同样的金钱购买的装备在回合初期对英雄提升较多而在回合后期对英雄提升较少。因此对经济的奖励应该按照英雄等级进行折算。计算经济奖励的代码实现如下：

```
1 if reward_name == "money":  
2     money = main_hero["moneyCnt"]  
3     reward_struct.cur_frame_value = money * level_factor
```

3.2.3 经验奖励

游戏回合中，英雄的经验用于升级，升级后可以增加自身属性。实验中每个英雄属性的分配方式和顺序已经固定，无需关注。同样随着英雄等级的提升，升级所需要的经验会逐渐上升，经验的作用减弱。因此同样对经验的奖励按照等级折算。另外升级会消耗英雄经验值，因此计算英雄获取的总经验时应该加上当前等级已经消耗的升级经验。计算经验奖励的代码实现如下：

```
1 def calculate_exp_sum(self, this_hero_info, level_factor):  
2     level_exp_cost = [  
3         0, 160, 458, 904, 1428,  
4         2041, 2754, 3579, 4529, 5617,  
5         6857, 8263, 9848, 11626, 13610]  
6     hero_level = this_hero_info["level"]  
7     base_exp = level_exp_cost[hero_level]  
8     exp_sum = base_exp + this_hero_info["exp"]  
9     exp_sum = exp_sum * level_factor  
10    return exp_sum
```

3.2.4 生存状态

英雄的生存状态包括生命值和法力值两项，为了游戏回合中数据的一致性，对生命值和法力值做归一化，计算为生命值和法力值的百分比。生存状态既可以直计算奖励，也可以和动作同时计算奖励，比如血量较低时应该优先考虑后退和召回。也可以在动作的 legal 中加入生存状态的考量。计算生命值和法力值的代码如下：

```

1 if reward_name == "hp_point":
2     reward_struct.cur_frame_value = math.sqrt(math.sqrt(1.0 *
3         main_hero_hp / main_hero_max_hp))
4 elif reward_name == "ep_rate":
5     if main_hero_max_ep == 0 or main_hero_hp <= 0:
6         reward_struct.cur_frame_value = 0
7     else:
8         reward_struct.cur_frame_value = main_hero_ep / float(
9             main_hero_max_ep)

```

3.2.5 攻击状态

英雄击杀对方英雄时应该获得奖励，自身死亡时进行惩罚（负奖励）。战绩可以通过英雄状态中 "killCnt" 和 "deadCnt" 字段获得。另外，英雄完成对士兵、敌方英雄和丛林野怪死亡前的最后一击，可以获得高额的金钱和经验，该行为称为“补刀”。需要在游戏中让英雄尽可能多的完成补刀行为，以获取较高的经济和经验成长。此处补刀行为仅考虑对方士兵即可，因为己方士兵通常不会攻击敌方英雄和丛林野怪，而且击杀敌方英雄和丛林野怪已经有远超击杀敌方士兵的金钱和经验奖励。计算补刀奖励的代码如下：

```

1 reward_struct.cur_frame_value = 0.0
2 frame_action = frame_data["frame_action"]
3 if "dead_action" in frame_action:
4     dead_actions = frame_action["dead_action"]
5     for dead_action in dead_actions:
6         if (
7             dead_action["killer"]["runtime_id"] == main_hero[""
8                 actor_state"]["runtime_id"]
9             and dead_action["death"]["sub_type"] == ""
10                ACTOR_SUB_SOLDIER"
11         ):
12             reward_struct.cur_frame_value += 1.0
13         elif (
14             dead_action["killer"]["runtime_id"] == enemy_hero[""
15                 actor_state"]["runtime_id"]
16             and dead_action["death"]["sub_type"] == ""
17                ACTOR_SUB_SOLDIER"
18         ):
19             reward_struct.cur_frame_value -= 1.0

```

3.2.6 英雄位置

在游戏中，英雄通常需要不断向敌方推进。在对战地图中，无论红蓝阵营，己方位于地图左下角，敌方位于右上角。英雄的出生位置位于左下方，如图 19 所示。因此大体上英雄应该朝向右上方前进，或者为了通用性，英雄应该沿着双



图 19 英雄出生位置

方水晶连线方向作为进攻方向。在没有其他需要考虑的情况下，英雄的位置按照靠近敌方防御塔来计算基础奖励，此时英雄未进入战场。基础位置奖励的计算代码如下：

```

1 def calculate_forward(self, main_hero, main_tower, enemy_tower):
2     main_tower_pos = (main_tower["location"]["x"], main_tower["location"]["z"])
3     enemy_tower_pos = (enemy_tower["location"]["x"], enemy_tower["location"]["z"])
4     hero_pos = (
5         main_hero["actor_state"]["location"]["x"],
6         main_hero["actor_state"]["location"]["z"],
7     )
8     forward_value = 0
9     dist_hero2emy = math.dist(hero_pos, enemy_tower_pos)
10    dist_main2emy = math.dist(main_tower_pos, enemy_tower_pos)
11    hero_hp = main_hero["actor_state"]["hp"] / main_hero[
12        "actor_state"]["max_hp"]
13    if hero_hp > 0.75:
14        forward_value = (dist_main2emy - dist_hero2emy) /
15        dist_main2emy
16    return forward_value

```

另外，在接触到敌方角色时，英雄需要考虑攻击和受伤害的范围，选择合适的战场位置，这个位置通常被称为“站位”。站位需要考虑的几个因素包括：兵线，防御塔范围。士兵通常会攻击距离最近的敌方阵营，因此在对战时，英雄应该站在士兵后方位置，己方士兵离敌方攻击单位最近的位置，称为兵线。如图图 20 所示，通常英雄不应该跨过该线。

英雄越过己方兵线后会被敌方阵营优先攻击，如 图 21 所示，此时智能体应该考虑控制角色后退，回到兵线后或者己方防御塔范围内。



图 20 兵线



图 21 英雄被对方士兵优先攻击

防御塔会对进入防御范围内的敌方阵营发动攻击，通常选择攻击离防御塔最近的敌方单位，但是在己方英雄遭受攻击且发动攻击的敌方处于防御塔范围内时会优先进行反击。防御塔的伤害比较高，通常英雄进入防御塔范围内时应该处于兵线的保护下。如果没有兵线保护，会被防御塔攻击，称为抗塔，如图 22 所示，这种情况下英雄会很快死亡。

大多数时候英雄不应该越过敌方防御塔，因为己方士兵会在遇到敌方阵营时进行攻击并停止前进，英雄越过敌方防御塔会失去己方兵线的保护，并不断的遭遇后续敌方阵营的攻击，如图 23 所示，这种行为称为“越塔”。通常越塔有固定的使用场景和攻击模式，即在追击敌方英雄且有兵线保护的情况下，进入敌方防御塔范围内，此时攻击敌方英雄会被防御塔优先反击，可以继续追踪到越塔后开始攻击敌方英雄，同时使用闪现技能迅速跳出防御塔攻击范围，击杀敌方英



图 22 英雄被防御塔攻击



图 23 英雄越过敌方防御塔

雄后使用召回技能传送回己方泉水法阵。在多英雄对战时可以考虑这种行为模式，单英雄对战时，越塔追踪并击杀敌方英雄获得的优势并不超过跟随兵线攻击敌方防御塔。

综合以上几点，在遭遇敌方角色时英雄应该站在兵线后，没有敌方角色时，英雄只在有兵线保护的情况下进入防御塔范围，此时称为英雄进入战场。英雄战场位置的奖励分为遭遇检测，兵线检测，位置惩罚三部分。其中遭遇检测判断英雄是否进入敌方阵营的攻击范围，由于无法获得敌方英雄的攻击范围，敌方英雄的对战奖励由伤害检测来反馈。具体的检测代码如下：

```

1 def is_encounter(loc, enemy):
2     enemy_pos = (enemy['location']['x'], enemy['location']['z'])
3     dist = int(math.dist(loc, enemy_pos))
4     return dist <= int(enemy['attack_range']), dist

```

兵线通过检测己方阵营到指定敌方目标的距离实现，如果有更接近目标的己方单位，则英雄处于安全位置。具体的检测代码如下：

```
1 def is_safe(enemy, teams, dist):
2     enemy_pos = (enemy['location']['x'], enemy['location']['z'])
3     for member in teams:
4         member_pos = (member['location']['x'], member['location']
5                         ['z'])
6         if int(math.dist(enemy_pos, member_pos)) < dist:
7             return True
8     return False
```

战场位置惩罚按照进入攻击范围的距离来计算，如果英雄处于所有敌方阵营的攻击范围之外，那么英雄是安全的。具体的计算代码如下：

```
1 def position_penalty(enemy, dist):
2     attack_range = float(enemy['attack_range'])
3     if float(dist) > attack_range:
4         return 0.0
5     return (attack_range - float(dist)) / attack_range
```

另外考虑到防御塔的攻击远远超出普通士兵，在位置惩罚中对敌方防御塔带来的惩罚加上较高的权重。



图 24 治疗符文

双方防御塔下会定期刷新治疗符文，如图 24 拾取后可以迅速回复大量生命值。当英雄生命值较低时，应该主动朝治疗符文位置移动去进行拾取。另外，治疗符文不区分阵营，在英雄靠近敌方防御塔下治疗符文且有兵线存在的情况下，应该主动拾取敌方防御塔下的治疗符文，达到削弱敌方的目的。对战中，在生命值低于 30% 时应该考虑后退寻找己方防御塔下治疗符文，在生命值低于 75% 且视野范围内有安全的治疗符文时，应该考虑拾取。治疗符文的数据仅当符文出现在英雄视野中时才有相关信息。数据结构如下：

```

1 [
2     'configId': 5,
3     'collider': {
4         'location': {
5             'x': 15340,
6             'y': 48,
7             'z': 15100
8         },
9         'radius': 0
10    }
11 },
12 [
13     'configId': 5,
14     'collider': {
15         'location': {
16             'x': -15220,
17             'y': 48,
18             'z': -15120
19         },
20         'radius': 0
21    }
22 ]

```

寻找最近的治疗符文并且判断是否需要拾取的代码实现如下：

```

1 def calculate_cake_lure(self, main_hero, cakes, pass_by=5000):
2     if cakes is None or len(cakes) < 1:
3         return False, (0, 0)
4     hero_state = main_hero['actor_state']
5     hp = float(hero_state['hp']) / float(hero_state['max_hp'])
6     if hp > 0.75:
7         return False, (0, 0)
8     target_cakes = []
9     hero_pos = hero_state['location']['x'], hero_state['location']
10    ['z']
11    for cake in cakes:
12        cake_pos = cake['collider']['location']['x'], cake['
13            'collider']['location']['z']
14        cake_dist = math.dist(hero_pos, cake_pos)
15        target_cakes.append((cake_dist, cake_pos))
16    target_cakes.sort(key=lambda x: x[0])
17    best_dist, best_pos = target_cakes[0]
18    get_cake = best_dist < float(pass_by) or hp < 0.3
19    return get_cake, best_pos

```

整合以上三点，最终英雄移动和站位的奖励函数实现如下：

```

1 forward_reward = self.calculate_forward(main_hero, main_tower,
2                                         enemy_tower)
3 wartime, position_reward = self.calculate_wartime_lane(main_hero,
4                                         main_tower, enemy_tower, main_roles, enemy_roles)
5 need_cake, cake_dist = self.calculate_cake_lure(main_hero, cakes)
6 cake_reward = 1.0 - cake_dist/10000.0
7 if need_cake:
8     reward_struct.cur_frame_value = cake_reward

```

```

7 elif wartime:
8     reward_struct.cur_frame_value = position_reward
9 else:
10    reward_struct.cur_frame_value = forward_reward

```

3.2.7 获胜奖励

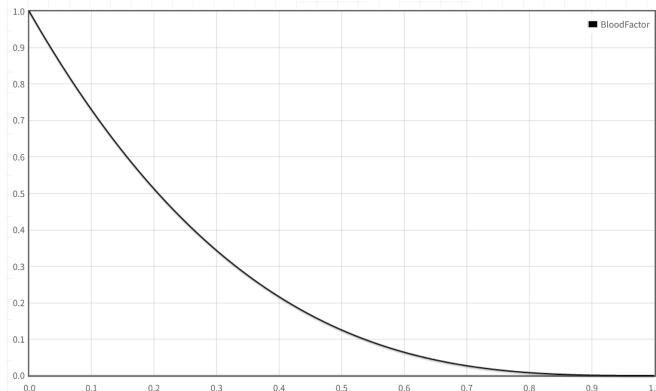


图 25 敌方防御塔生命值奖励因子

当英雄在兵线的保护下进入敌方防御塔范围内时，补刀之外，应该考虑攻击敌方防御塔。此外当敌方防御塔血量较低时，英雄应该提高攻击敌方防御塔的优先级，以达到快速获胜的目的。当英雄攻击敌方防御塔时，按照防御塔的血量百分比，以指数方式计算奖励。公式为：

$$r = (1 - h)^3 \quad (2)$$

依据敌方防御塔生命值百分比，奖励因子从 0 到 1，如 图 25 所示。具体的获胜奖励实现代码为：

```

1 def calculate_tower_hit(self, main_hero, enemy_tower):
2     hit_target = main_hero['hero_states']['actor_state']['
3         hit_target_info']
4     enemy_tower_id = enemy_tower['config_id']
5     hit_tower = False
6     for target in hit_target:
7         if target['hit_target'] == enemy_tower_id:
8             hit_tower = True
9             return
10    enemy_tower_hp = float(enemy_tower['hp']) / float(enemy_tower
11        ['max_hp'])
12    target_reward = math.pow(1.0 - enemy_tower_hp, 3.0)
13    return hit_tower, target_reward

```

3.3 学习策略

PPO 算法的关键超参数包括学习率、clip 范围、熵系数、价值函数系数、GAE 参数等。这些参数会影响 PPO 的效果和稳定性，收敛速度等，通常和具体的问题有关。几个关键超参数的作用和推荐值如下：

- **学习率**：控制策略和价值网络参数的更新步长；过大导致震荡，过小时收敛较慢；典型范围为 1×10^{-5} 到 1×10^{-3} 。
- **裁剪范围**：限制策略更新幅度，保持新旧策略相近；值太小限制更新，值太大失去约束效果；典型值为 $0.1 \sim 0.3$ ，通常用 0.2。
- **熵系数**：鼓励探索，防止策略过早收敛；前期可以稍大，后期逐渐缩小；典型范围是 $0.00 \sim 0.02$ ，通常取 0.01。
- **价值函数系数**：平衡策略优化与价值函数学习；典型范围 $0.5 \sim 1.0$ ，通常取 1.0。
- **GAE 参数**：通常折扣因子 γ 取 $0.900 \sim 0.999$ ，偏差方差权衡比率 λ 取 $0.90 \sim 0.98$ 。

3.3.1 PPO 截断

PPO 算法属于策略梯度方法，通过限制每次策略更新的幅度来保证稳定性。TRPO 算法使用复杂的优化过程来约束更新，而 PPO 算法通过剪辑概率比率来简化这个过程。裁剪的作用主要是防止过大的策略更新，从而导致训练崩溃。裁剪是 PPO 算法比较重要的一个特征，通过对 PPO 裁剪系数进行修改，验证不同裁剪范围时 PPO 算法的表现。

智能体决策 1V1 中初始的 PPO 裁剪范围配置为 0.2，将裁剪范围修改为 0.3 观察训练过程和结果的差异，理解 PPO 裁剪对训练过程的影响。

3.3.2 奖励趋势调整

智能体决策 1V1 实验中，英雄角色在对战中不同动作的作用不同，移动属于基础动作，而攻击和技能释放属于使用频率较低的动作，此外，攻击和释放技能也应该在英雄进入合适的位置后进行。因此在训练的周期中，应该首先考虑训练智能体控制英雄角色的基本动作，在此基础上，进行攻击和技能释放的训练，最后可以修改英雄的战斗策略来提高胜率。整体上训练可以分为以下三个阶段：

- 训练前期，偏重具体行为相关的稠密奖励，引导智能体学会基本操作
- 训练中期，增强与对局结果强相关的稠密奖励，引导智能体在单局中建立优势
- 训练后期，调高稀疏奖励权重，引导智能体直接关注最终胜负

由于实验中训练时间和游戏熟悉程度的原因，本次实验关注前两个阶段，验证训练中不同时期应该关注的问题。

四、实验结果与分析

在算法实验的过程中，通常先运行一次基础版训练和评估作为基准，然后在此基础上进行优化。这里采用开悟平台提供的框架代码中的默认实现作为基准。

4.1 基准实验

为了方便实验中对代码进行改动，采用现有 PPO 算法的代码作为基础。将 PPO 算法目录中的代码文件拷贝到 diy 目录中进行覆盖，然后修改代码文件中导入时的包名称。在提交训练任务时选择 diy 算法，进行 10 小时训练，训练结果的模型作为基准模型。为了便于后续对比，基准模型的训练对手采用 common_ai，后期可以清楚的观察到模型的收敛时间和胜率变化等现象。



图 26 基准实验的评估结果

如 图 26 所示，即使在自对弈模式下，红蓝双方的结果数据也并不会完全一致。由于双方都是依据环境动态决策，当其中一方先做决策后，另一方会采用不同的策略。回顾游戏视频可以发现，英雄角色在对战中会出现越过兵线，抗兵，抗塔等现象；另外一些时间段会出现英雄没有任何进攻或防御行为，脱离或远离战斗区域，浪费了成长机会；最后，英雄在血量较低的时候，没有后退或者

躲避伤害的行为，知道英雄死亡。这些行为也是符合预期的，在后续的优化中，会针对这些问题进行修改。

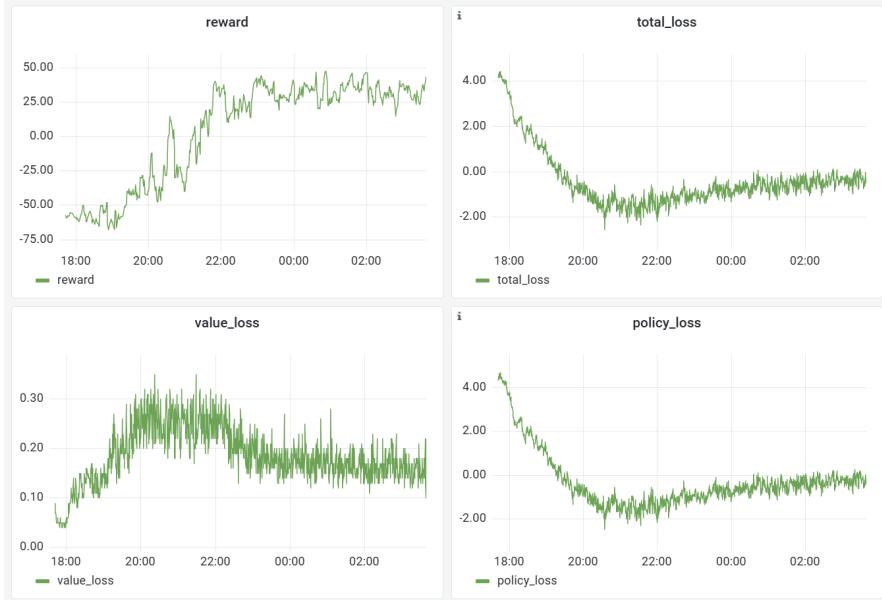


图 27 基准训练监控数据

如图 27 所示，从监控面板可以看到，奖励曲线基本在稳定的上升，但是中间也会出现较大的波动，这比较符合基于策略的强化学习模式，通常有较大的方差比较难收敛，而 TRPO 和 PPO 就是为了解决方差大和收敛慢的问题。后续可以考虑调整学习率和截断参数的，来验证收敛速度和方差问题。



图 28 基准训练监控比分数据

如图 28 所示，观察监控中的胜率和防御塔血量数据可以发现，在训练开始后大约 5 小时，胜率和敌方防御塔血量基本已经稳定，这说明模型已经基本收敛。在后续的训练中，己方防御塔剩余血量继续增加，说明智能体后续的训练在持续的提高己方优势，只是由于对手太弱，已经无法表现在胜率上。

4.2 修改奖励函数

智能体决策 1V1 实验中可以加载已有的模型进行继续训练，这为智能体模型优化提供了便利，可以针对当前模型的问题进行优化然后进行增量训练。按照前面实验设计中设计的奖励函数进行修改后，并将模型的保存时间修改为 600 秒，即每十分钟保存一个模型，在模型的优化时通常采用较短的保存时间。在基准模型的基础上训练 2 小时。

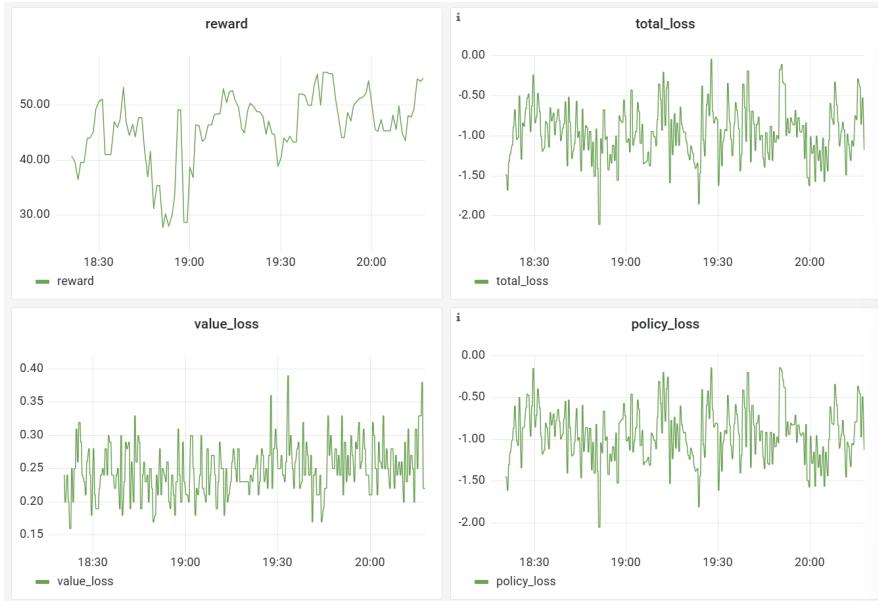


图 29 修改奖励函数后的训练过程监控

如 图 29 所示，可以看到在两个小时的训练时间内，reward 略有上升。策略和价值整体波动变化不明显。

观察训练过程中的对战数据可以发现，在保持胜率的基础上，帧数有稳定的下降趋势，即结束游戏回合所用的时间变短，平均每帧的经济也有下降，如 图 30 所示。这说明修改的奖励函数产生了效果，英雄更注重尽快结束游戏对战，而不是专注于发展经济和提升等级。

4.3 修改学习策略

针对 PPO 算法的学习策略，验证裁剪范围的影响，和分阶段训练的效果。

4.3.1 PPO 截断

开悟平台上智能体决策 1V1 实验代码框架中初始配置的 PPO 裁剪范围是 0.2，训练效果为基准实验中所示。通过将裁剪范围修改为 0.3，其他部分保持和基准实验一致，从头开始进行 10 小时训练。



图 30 修改奖励函数后的训练过程比分监控

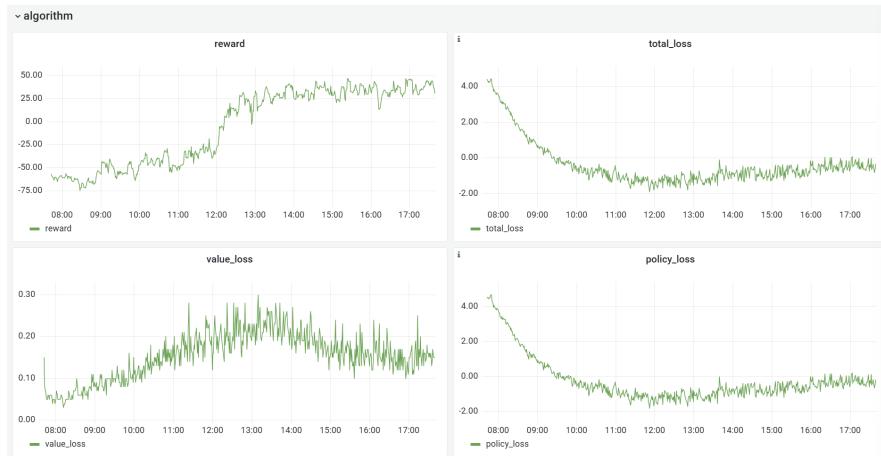


图 31 PPO 裁剪范围 0.3 的训练监控

如图 31 所示，修改 PPO 裁剪范围为 0.3 后，可以看到 reward 和 value_loss 波动更明显，方差变大，这点符合 PPO 裁剪变化的预期。总体上模型的训练过程仍然是收敛的，没有出现崩溃现象。

对比基准实验和本次实验的胜率变化，可以看到 PPO 裁剪范围 0.2 的时候，训练过程大约在 4~5 小时收敛，而 PPO 裁剪范围 0.3 的时候，训练过程大约在 7 小时收敛，如图 32 所示。PPO 裁剪范围变大的情况下，训练过程比较激进，导致收敛速度较慢。

将本次实验的最终模型提交到模型管理后，进行评估，对手选择为基准实验生成的模型。从结果来看，相对于基准模型，胜率、击杀、经验和金钱均有略微优势，如图 33 所示。这是由于 PPO 裁剪范围较大时，训练时可以探索更多的

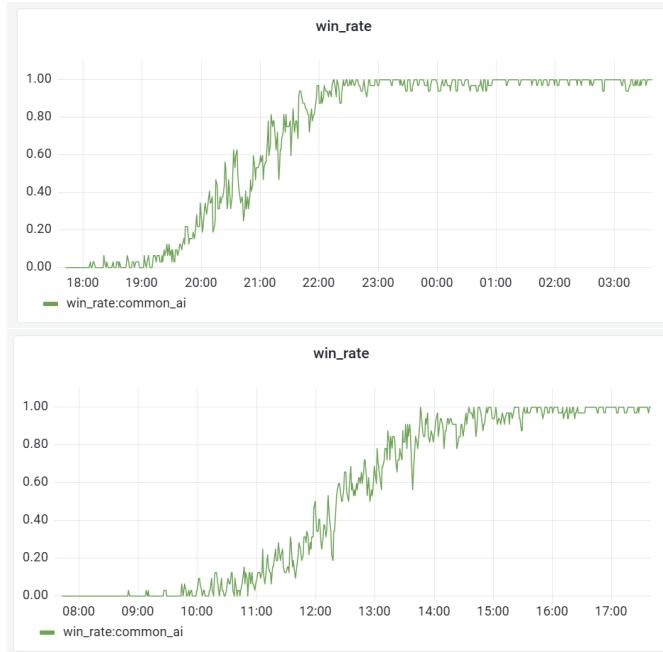


图 32 PPO 裁剪范围 0.2(上) 和裁剪范围 0.3(下) 的胜率对比

策略，这也导致训练过程收敛较慢。

4.3.2 分阶段奖励

从训练过程中来看，训练初期，智能体主要学习如何控制英雄走向中央广场，并在战斗前停止在中央广场的合适的位置。此后，训练智能体控制英雄寻找并攻击敌方阵营和躲避敌方攻击。最后，训练智能体选择合适的攻击目标和取胜策略。

本次实验中，通过先从初始化模型开始，连续训练 10 小时，得到一个基本的模型，该模型主要学习英雄移动和攻击等基本行为。在此模型的基础上，训练



图 33 PPO 裁剪参数 0.3 的评估结果

智能体控制英雄角色选择合适的攻击目标，躲避敌方伤害等较高逻辑等级的行为。通常基础训练需要较长时间才能收敛，而后续的优化训练可以使用较高的奖励，快速实现优化模型的效果。本次实验中优化训练选择 2 小时，从评估结果的录像回放中能看到，英雄已经具有较高的逻辑行为表现。

五、结论

PPO (Proximal Policy Optimization) 是一种强化学习算法，属于策略梯度方法的一种改进版。PPO 提出的主要目的是解决传统策略梯度方法训练不稳定的问题。TRPO 虽然通过约束来保证更新步长，但计算起来比较复杂。PPO 通过引入剪裁的替代目标函数，简化了实现，同时保持了稳定性。相比于其他如 A3C 等算法，PPO 可能更有效地利用样本，减少与环境交互的次数。此外，PPO 在实验中表现出的鲁棒性也是一个优势，适用于多种任务，不像有些算法需要大量调参。

5.1 PPO 学习方法的优缺点

与其他深度强化学习算法相比，PPO 算法的优点有：

1. 训练稳定性：通过引入裁剪替代目标函数 (Clipped Surrogate Objective)，限制策略更新幅度，避免传统策略梯度方法的发散风险，相比 TRPO 更易于实现。
2. 样本效率高：支持多轮次 (mini-batch) 策略更新，在 Atari 等复杂任务中比 A3C 等算法更高效。
3. 适应性广泛：对超参数相对鲁棒，在连续控制 (如机械臂)、游戏 (如 Dota2) 等领域均有成功应用。
4. 实现简单：无需复杂的二阶优化 (如 TRPO)，仅需一阶优化即可实现稳定训练。

同时，PPO 算法的缺点有：

1. 超参数敏感：裁剪阈值 (如 $\epsilon = 0.2$) 和学习率需要精细调节，不同任务可能需要重新调参。
2. 局部最优陷阱：在稀疏奖励环境中，可能因策略更新过于保守而陷入次优解。

3. 高维动作空间挑战：对于超高维动作空间（如多机器人协同），需结合其他技术（如自注意力机制）提升效果。
 4. 探索能力受限：相比 Q-learning 类算法（如 DQN），对探索机制的依赖更隐式，可能需要显式熵奖励辅助。
- PPO 因其平衡了实现难度与性能，成为目前强化学习的基准算法之一。

5.2 可能的优化与改进方向

PPO 裁剪保证了策略的稳定上升，但同时也容易陷入局部最优解，通常需要结合实际经验，调整奖励函数，显示的考虑长期奖励来辅助。比如在智能体决策 1V1 中，需要考虑一些特殊的对战策略，并且需要通过状态变化提前预测奖励和损失，以使奖励函数尽可能的平滑和单调。

此外，PPO 算法在训练中比较难预测收敛时间，可以考虑结合经验召回，在支持较大的 PPO 裁剪区域的情况，考虑对经验按照回合进行评估，抛弃较差的结果。PPO 算法目前使用的裁剪因子都是静态的，可以考虑结合训练流程和收敛情况，动态调整裁剪因子，以达到快速收敛和探索机制的平衡。