目录

一．问题介绍

本次 RLChina 智能体挑战赛 – 壬寅年秋赛季中，采用的是“奥林匹克-相扑”为基本比赛科目。在这次竞赛中，两个智能体参加相扑竞赛，目标是避免摔下擂台并且争取将对方装撞下擂台从而获得比赛的胜利。

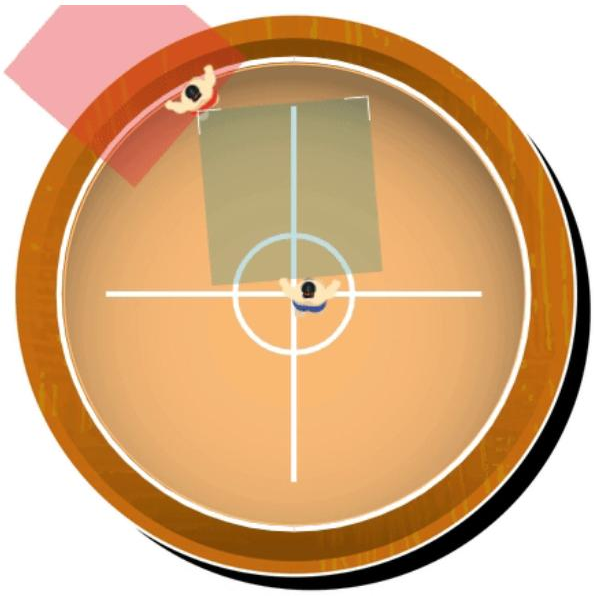

图 1 奥林匹克-相扑

1.1 环境

两个智能体参加相扑竞赛，目标是避免摔下擂台并且争取将对方装撞下擂台。

1.1.1 “AI 奥林匹克-相扑”规则：

- 本游戏共有两方，双方操纵的 agent 各为一个半径为 20，质量为 1 的小球。双方初始化位置为圆形擂台垂直方向的直径上，两球在圆心两侧，与圆心相距 150；
- 比赛开始时，双方智能体能在圆形场地以内自由活动，但不能触碰场地边界，否则视为掉下擂台出界；
- 智能体可以互相碰撞，但根据球体摩擦系数会损失一定的速度；
- 智能体自身有能量，每步消耗的能量与施加的驱动力和位移成正比；
- 智能体能量同时以固定速率恢复，如果能量衰减到零，智能体出现疲劳，导致不能加力；
- 环境结束条件：当有一方摔下擂台时或环境达到最大步数 500 步时环境结束；

1.1.2 交互接口参数说明：

- **observation：**观测是一个字典，键为"obs"和"controlled_player_index"。这个observation 是环境根据当前我方 agent 的位置和旋转角度在游戏环境中采样得到的。该observation 只能显示我方 agent 周围的视角，agent 的中心大概位于 observation 的(19,32)处。每一个像素对应环境中的距离为 5。"obs"对应的值为一个字典，其中包含了 40x40 的二维矩阵和其他与游戏相关的信息。二维矩阵记录了智能体自身朝向的视野内容，在规定视野内智能体能够看见墙壁，地面指示线。observation 中地图的边界、地标线、我方和敌方 agent 分别用四种非零正整数表示，其余位置均为 0。地标线这样无法产生物理碰撞的物体会被智能体遮挡。"controlled_player_index"对应的值（以颜色表示）为控制智能体的编号。

- **action：** agent 需要提供两个数作为当前 step 的 action，分别是 agent 输出的力 force 和旋转角度 angle。 action_space：长度为 n_action_dim 的列表，其中 n_action_dim=2，每个元素为 Gym 当中的 Box 类（Box Link），如[Box(-100.0, 200.0, (1,), float32), Box(-30.0, 30.0, (1,), float32)]。agent_action：2*1 的矩阵，其中 2 表示动作空间的维度数，1 表示每维动作空间的值。

- **is_act_continuous：**布尔型变量，表示动作空间是否连续，此处为 True。

1.1.3 内部机制：

- **energy：** energy 决定 agent 是否还能运动。它只由当前 step 输出的力 $forec_t$ 和这一step 结束时速度的模$|v_t|$决定，具体公式为：

$$\mathrm{energy_t} = \min(\mathrm{energy_{t-1}} + 20 - \mathrm{force_t} * |\mathrm{v_t}|/50, 1000$$

若 energy<0 ，则此后无法再对智能体执行输出任何 action，agent 只能以当前速度保持匀速直线运动或者被碰撞。 本次比赛中， energy 不会再在 observation 中提供，这在之后训练智能体的过程中产生了很多困难。

- **action：** 每一个 step，agent 需要提供两个数作为当前 step 的 action，分别为 agent 输出的力 froce 和旋转的角度 angle。$angle \in [-30,30]$ ，为正则顺时针旋转，为负则逆时针旋转。旋转只影响该个 step 力输出的方向，以及之后环境返回 observation 的角度，不会影响 agent 的速度。 $force \in [-100,200]$ ，为正则向前输出力，为负则向后输出力。力的输出在旋转之后执行。100 的 force 会给 agent 带来 1 的加速度。agent 的速度上限为 10，当速度超过 10 时，会等比例放缩速度的 x，y 分量，使之合速度为 10。

- **reward：**将对手撞下擂台得 100 分，否则得 0 分。

1.2 赛制

采用瑞士轮复式赛制。随机公平地编排第一轮比赛（一般由抽签决定），接着开始比赛，当某一轮比赛结束后，可以得到所有比赛选手的总积分，根据这个总积分的高低，把比赛选手的由高到低排序，接着是高分比高分，低分比低分，上一轮比过的下一轮就不会相遇，如此循环，直到所有轮次结束。

当选手人数为奇数时，中位数的选手此轮轮空；当选手人数为偶数时，按照积分排名的顺序，两两一组进行 PK。

在瑞士轮复式赛制中，所有参赛选手都会在一个赛程表中进行匹配，每场比赛都是针对同一个对手进行的。每场比赛后，胜者会继续挑战下一个对手，而输者则会重新进入赛程表的某个位置继续比赛。这样，所有选手就会在比赛中不断交替，直到最后只剩下一名胜利者。

瑞士轮复式赛制的优点在于，它能够让所有选手都有机会与其他选手进行比输，因此比赛更加公平。另外，由于所有选手都会在比赛中不断交替，因此比赛过程也会更加紧张刺激。

1.3 问题分析

在此次竞赛中，相扑比赛是一个两个智能体直接对抗并尽量使自己获胜、使对方输掉的的比赛，所以可以将它看成是一个博弈对抗的过程。在相扑比赛中，智能体可以使用许多不同的策略来获胜，并且需要根据对手的决策来调整自己的策略。所以采取合适的算法或者说是策略来使自己的智能体获得比赛的胜利是解决此问题的关键。

而这种博弈模型可以更加一步具体为零和博弈，因为在相扑游戏过程中，一方的胜利必然伴随另一方的失败。这就意味着能将两个智能体的总收益之和记为 0。所以在思考智能体与对手决斗时，要考虑如何最大化自己的收益，同时最小化对手的收益。

在相扑竞赛中会存在面临缺少对手信息的难题，在 AI 领域中，被称为不完全信息问题。不完全信息是一个非常常见的问题，在博弈对抗中，智能体可能缺少对手的信息，并必须基于自己的信息做出决策。为了应对不完全信息的情况，人工智能系统可以使用不同的策略来做出决策。常见的有最小期望策略、纳什均衡策略。在机器学习中也有监督学习与半监督学习的方法，这些方法都能帮助智能体在缺少信息的情况下做出有效的决策。

1.4 难点

1.4.1 博弈对抗

该游戏环境包含两个智能体，两个智能体为竞争关系，属于零和博弈。而且根据游戏的赛制，我们训练得到的智能体需要与其它参赛者的智能体 pk，然后得到排名，所以我们得到的策略模型具体的表现还取决于其它参赛者的策略。

如下图游戏理论：锥体表面为各类策略，横轴表示各类策略的克制关系，纵轴表示各类策略的能力。实际学习过程中，我们希望策略能够沿纵轴提升，达到锥体顶尖的纳什均衡，

对应到现实世界可以理解为：某个游戏顶尖职业玩家的水平，但在学习过程中往往会停滞在某个水平，比如下图的红色区域，这时学习得到的策略克制旧策略，但同时又有其它相同水平的策略克制当前学习到的策略，这些相同水平的策略是传递克制的关系，这时在学习过程中，很可能陷入循环，无法学习得到沿纵轴提升的策略，另一方面，要做出突破，需要较大的探索空间，保证所见过的策略的多样性，只有对当前水平的所有策略有充分的了解，才能得到更好的沿纵轴提升的策略。

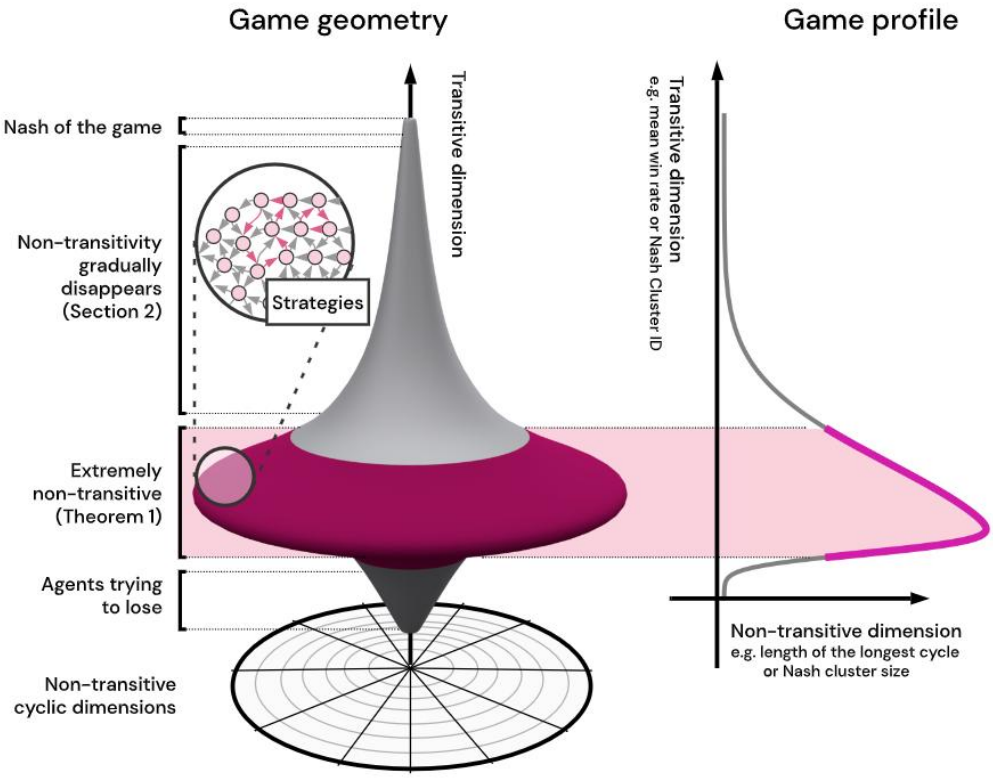

图 2 游戏理论图

1.4.2 不完全信息

在该游戏环境中，所能得到的有效信息仅为一个 40× 40 的方形区域，并没有覆盖全图，得到的信息是不完整的，具体实现时，每个时间步，都需要利用当前不完整的信息做出当前的最优决策。

1.4.2 阅读源码

该比赛文档提供的信息有限，仅包含基本的接口参数的说明，但同时参赛者又可以得到游戏环境的源代码，即为白盒环境，所以整个游戏内部的运行机制都可以通过阅读源码得到。显然在每一步做决策时，当前所了解的信息越多，所做出的决策越接近最优解，但游戏所提供的接口能获得的信息又有限，所以需要阅读游戏环境的源码，了解游戏内部的运行机制，

从而能够还原部分信息，使得能够做出更好的决策。

二、基础知识

2.1 自博弈（Self-Play）

在单人游戏的训练中，要解决的问题很简单，即在环境中获得最大的回报。可以将其视为智能体与环境本身的对抗，而且环境在整个训练过程中不会发生变化。但是在相扑这种双人对抗游戏中，问题就变得复杂了，因为智能体在学习的过程中所能达到的水平与对手也有很大关系。打败随机行动的对手很容易，但这并不意味着在环境中达到了完美的程度。因此，我们必须不断比较网络的当前版本与之前版本，希望智能体不断寻找新的策略来战胜新对手，从而逐渐变得更强大。而利用上述思想解决相扑比赛智能体训练问题的关键则是自我博弈策略。自博弈是一种开放性的强化学习训练框架，出现在多智能体训练的背景下。Self-play 训练方案主要是通过模拟与自己玩游戏并使用训练期间生成的策略来训练 agent 进行学习。

2.1.1 基础框架

（1）强化学习基础知识

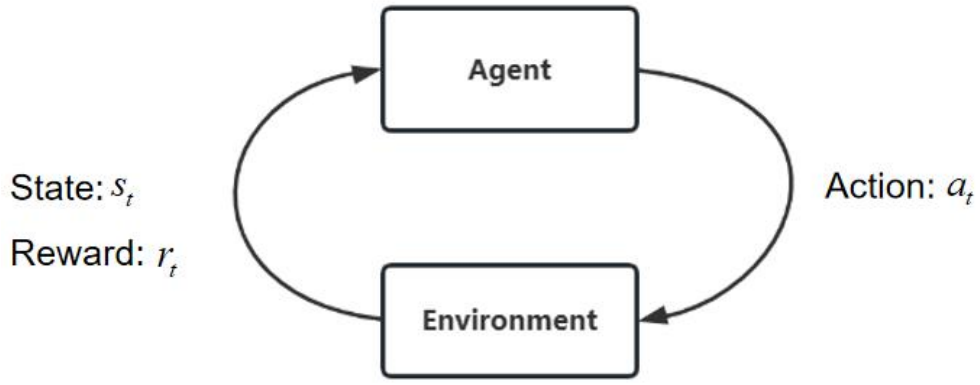

图 3：智能体与环境交互示意图

以多智能体为例：

E：表示具有 n 个 agent 的多代理系统且具有奖励折扣因子 γ。

奖励值随着时间的推移进行折扣，以免陷入局部最优。

S：状态空间，有限状态 state 集合，s 表示某个特定状态，是全局的。

O：联合观测空间，有限 observation 集合，o 表示某个特定观测，是局部的。

如果是多 agent 的场景下，则每个 agent 的 observation 与全局的 state 是不一样的。

A：联合动作空间，有限动作 action 集合，a 表示某个特定动作。

π：联合策略向量，可以看作是在联合观察空间条件下的联合行动空间上的分布。

Reward $R(s,a)$：表示 agent 采取某个动作后的即时奖励。

Policy $P(s,a)$：根据当前 state 和 action 产生下时刻 state 和 observation。

（2）Self-play 算法核心流程

自博弈训练方案可以被视为通过在每一集之前和之后引入功能来扩展 MARL 循环的模块。设 π 是整个 MARL 循环中唯一被训练的策略。SP 方案通过首先决定从一组固定策略 $\pi^{'} \subseteq \pi^{o}$；中获取的哪些策略 $\pi^{'}$ 将定义智能体在下一事件中的行为来包围 MARL 循环。但这不包括其行为由 π 定义的智能体。一旦情节结束，函数 G 决定是否将（可能更新的）策

略 π 引入可用策略池 π^o 中。 该算法在部分可观察随机博弈 (POSG) 循环中提出了一个 SP 方案，定义了一个 n 人、一般和、部分可观察的环境，具体流程见表 1：

表 1：RL Loop with Self-Play 算法流程表

Algorithm:(POSG) RL Loop with Self-Play.

Input： $Environment: (S, A, O, P(\cdot,\cdot|\cdot,\cdot), R(\cdot,\cdot), \rho_0)$

Input： $Self-Play\ Scheme: (\Omega(\cdot|\cdot,\cdot), G(\cdot|\cdot,\cdot))$

Input： $Policy\ to\ be\ trained: \pi \in \Pi_i$

$\pi^o = \{\pi\}$; //Menagerie initialization

for $e = 0,1,2,...$ **do**

  $\pi' \sim \Omega(\pi^o, \pi)$; //Sample from menagerie

  $\pi = \pi' \bigcup \{\pi\}$;

  $s_0, o_0 \sim \rho_0$;

  for $t = 0,..., termination$ **do**

    $a_t \sim \pi(o_t)$;

    $s_{t+1}, o_{t+1} \sim P(s_t, a_t)$;

    $r_t \sim R(s_t, a_t)$;

    $t \leftarrow t+1$;

  end

  $\pi \leftarrow update(\pi)$;

  $\pi^o \sim G(\pi^o, \pi)$; //Curate menagerie

end

return π;

通过形式化动物园 π^o、策略采样分布 Ω 和门函数 G 的概念来定义 SP 训练方案。通过元组 $<\Omega(\cdot|\cdot,\cdot), G(\cdot|\cdot,\cdot)>$ 具体说明：

- $\pi^o \subseteq \Pi_i$；动物园。一组策略，从中可以对智能体的行为进行采样。该集合始终包含当前训练策略 π。对 π^o 施加了约束。它的所有元素都必须至少间接地从被训练的策略 π 中导出。因此，动物园中的所有策略都是 π 策略空间的元素。动物园可以通过下面描述的馆长功能随着训练的进行而改变。
- $\Omega(\pi' \in \Pi_{-i} | \pi^o \subseteq \Pi_i) \in [0,1]$；其中 $\pi' \subseteq \pi^o$；策略抽样分布。动物园 π^o 的概率分布，可用策略集。它以动物园 π^o 和当前正在训练的策略 π 为条件。它选择除 π 之外的哪些策略将会驻留在环境的代理中。
- $G(\pi^{o'} \subseteq \Pi_i | \pi^o \subseteq \Pi_i, \pi \in \Pi_i) \in [0,1]$；动物园的馆长或门控功能。一个可能的随机函数，其参数是当前训练策略 π 和一个动物园 π^o。馆长有两个目的，复杂的馆长可以分为两个功能：
  - G 决定是否将当前策略 π 引入动物园；
  - G 决定动物园中的哪些策略 $\pi \in \pi^o$ 将从动物园中丢弃。

（3）创新点

- **探索课程（EXPLORATION CURRICULUM）**

场景：给定一个没有任何先验运动学知识的人形机器人，如果要从零开始通过自我对抗的方法将它训练成能够后空翻的机器人，规定只有当它完成后空翻时才给予奖励，而完全不给任何指引，那么经过漫无目的的尝试，最终它也许只能在地上撒泼打滚，甚至学不会站立。

探索课程的核心理念为：只有完成最终大目标才获得的奖励称之为稀疏奖励(sparse reward)，假如在每一个与大目标有关的小动作上都给予阶段性的稠密奖励(dense reward)，将最终目的分解为多个小目标，逐步完成最终目标。为训练智能体以完成相扑这种复杂的对抗性运动，可以在训练的前期给予多个小阶段的稠密奖励，使智能体能够逐渐掌握基础的关节动作，例如向前走、站立等，这样的基础性动作在自我对抗的训练中出现的概率很大，也就提高了智能体获得正向奖励的随机概率，我们将这种奖励称为探索奖励(exploration reward)。同时，为了解决探索奖励与竞争奖励的平衡问题，随着自我对抗训练过程的进行，可以将探索奖励逐渐衰减至 0，这样有利于智能体最终顺利完成目标并获得稀疏竞争奖励。

- **对手抽样（OPPONENT SAMPLING）**

背景：如果目标与对手实力悬殊，一直无法取胜获得正向的反馈激励，目标就一直无法进步，一个直观的想法就是让对手放水，即让对手变弱，通过打败一个弱版本的对手来获得奖励，增强学习效果。

因此训练过程中遇到过于强大的对手时，系统会在该对手的早期版本中抽样作为博弈对象，这样让弱者有获胜的机会，在博弈中取得奖励和进步，一旦弱者的水平提升到一定程度，也能反过来训练激励强者使其变得更强。对手太弱的情况也是类似，只需要抽样自己的早期版本供对手训练即可。

- **鲁棒学习策略 （LEARNING ROBUST POLICIES）**

背景：强化学习与深度学习中，常出现的一个问题是过拟合。比如 AlphaGo 最终可能训练出一个能打败柯洁却打不过公园大爷的“围棋高手”。

为了提高学习策略的鲁棒性，可以对环境引入随机效果，例如在相扑中圆形场地的半径可以是随机的，但这样的随机性给早期训练带来了极大的探索可能性，导致智能体的训练时间过长，收敛速度过慢。因此，通过设计一个可变的随机性算子，其以较小的值开始训练，以便智能体能够更快掌握基础动作。除此之外，也可以让智能体同时学习多个策略，引入一个策略池，在每次推出特定策略时，会随机选择其他策略中的一个作为对手，最终就能训练出一个集百家之长的智能体。

2.1.2 使用方法

自博弈算法训练的常用方法是结合 SP 和蒙特卡洛树搜索（MCTS）产生的移动进行监督学习。只需要告诉智能体一个技能的规则并设定一个目标函数，即奖惩机制，除此之外没有任何领域知识。然后通过循环迭代的多次自博弈，逐渐意识到哪些动作能使自己在博弈中

取胜的概率最大化，并最终掌握这项技能。其关键结构说明如下：

Neural network

- 输入是局面 s，输出是预测的 action 概率向量 p 和胜率 v；
- 目标就是最小化联合损失，就是让神经网络的预测跟 MCTS 的搜索结果尽量接近。

MCTS

- 接收 game，net，参数等信息、蒙特卡罗树搜索（选择，拓展，模拟，反向传播）；
- 通过 MCTS 搜索树优化神经网络参数，从而指导 MCTS 下一步节点的高效选择；
- 利用蒙特卡洛树搜索建立一个模型提升器。

Players

根据输入局面信息配合神经网络对象给出的估值结合蒙特卡洛树搜索给出应对方法。

Board

包括游戏规格、执行 player 的行动、定义 player 的行为（如飞行、发射等）。

Game

获取 board 信息、定义有效行动、获取某时刻可能的行为数量、得到下一时刻状态、游戏结束逻辑、获取 player 的得分。

Self-play

- 循环接收以 MCTS 策略的 player 的行动及对应可能性，存储当前数据并执行行动直到游戏结束，得到的数据是一串局面从开始到结束的数据集合，用于输入神经网络。
- 在自博弈过程中，利用提升器指导模型提升，进而进一步提高提升器的能力。

2.2 PPO

2.2.1 算法原理

使用策略梯度算法获得好的结果需要仔细调整步长 (Schulman et al., 2015a)。此外，大多数策略梯度方法对每个采样轨迹执行一次梯度更新并且具有高样本复杂度。为解决这两个问题，Schulman 等人（2017）提出了 PPO 算法。 这使用了一个最大化的替代目标，同时惩罚政策的大变化。令 $l_t(\theta)=\dfrac{\pi_\theta(a_t\mid s_t)}{\pi_{\theta_{old}}(a_t\mid s_t)}$ 表示似然比。然后 PPO 优化目标：$L=E[\min(l_t(\theta)\hat{A}_t,\mathrm{clip}(l_t(\theta),1-\varepsilon,1+\varepsilon)\hat{A}_t)]$，其中 $\hat{A}_t$ 是广义优势估计，$\mathrm{clip}(l_t(\theta),1-\varepsilon,1+\varepsilon)$ 在区间 $[1-\varepsilon,1+\varepsilon]$ 中修剪 $l_t(\theta)$。该算法在从策略中采样多个轨迹和在采样数据集上执行 SGD 的几个时期之间交替，以优化这个替代目标。由于状态价值函数也同时被逼近，因此价值函数逼近的误差也被添加到智能体目标以计算完整的目标函数。PPO 算法执行的具体流程见表 2。

表 2：clip version 的 PPO 算法流程表

Algorithm:PPO-clip
Hyperparameters：clip factor ε **,the number of sub-iterations** M,B
Input：initial policy parameters θ **.initial value function parameters** ϕ
for $k=0,1,2,\ldots$ **do**
Collect set of trajectories $D_k=\{\tau_i\}$ by running policy π_{θ_k} in the environment
Compute rewards-to-go $\hat{G}_t$

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the Current value function V_{ϕ_k}

for $m \in \{1,...,M\}$ **do**

$$l_t(\theta') = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}$$

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min(l_t(\theta'),1-\varepsilon,1+\varepsilon) A^{\pi_{\theta_{old}}}(S_t, A_t),$$

$$\text{clip}(l_t(\theta'),1-\varepsilon,1+\varepsilon) A^{\pi_{\theta_{old}}}(S_t, A_t))$$

typically via stochastic gradient ascent with Adam

end for

for $b \in \{1,...,B\}$ **do**

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(S_t) - \hat{G}_t)^2$$

typically via some gradient descent algorithm

end for

end for

2.2.2 使用原因

（1）离散动作空间

之所以选用 PPO 算法进行相扑比赛智能体的训练，而非其他连续强化学习算法（例如 SAC，DDPG 等），是因为目前在很多实际的项目中，基本没有用连续动作强化学习算法的。而且如果遇到了连续动作空间，大家往往第一步就会采用离散化动作空间。例如对于 $a \in [-1,1]$,工程上第一件事是将其离散化为 $a = \{-1,-0.5,0,0.5,1\}$ 。因为连续动作产生的效应是一个非常尖锐的尖峰分布，不具备连续的属性，动作很容易卡在边界（例子中的-1,1）。

（2）大规模训练时的分布式强化学习

在进行大规模的多智能体强化学习训练时，一般会采用分布式的训练方式，这允许我们在训练期间使用非常大的批量，在一定程度上解决了差异问题，同时也有助于探索。在分布式强化学习的训练中，actor 端负责和环境进行交互生成数据，而 learner 端则负责用收集到的数据训练模型，然后同步给 actor。这个过程中总会产生时延等因素，导致 actor 和 learner 端的模型不同步，而采用 PPO-分布式算法则无需进行任何同步操作，就可以非常完美地解决这个问题：

$$Loss_p = clip(\frac{\pi(s,a;\theta_{learner})}{\pi(s,a;\theta_{actor})},1-\varepsilon,1+\varepsilon) A(s,a)$$

2.3 霍夫变换

霍夫变换是一种图像处理算法，用于在图像中检测直线、圆形和椭圆等几何形状。它可以在图像中识别出直线的位置和方向，并可以检测出图像中的几何形状，即使这些形状被模糊或压缩过也能够检测出来。

霍夫变换是由霍夫提出的，有几种不同的变换算法，如标准霍夫变换、概率霍夫变换和逆霍夫变换等。它们的共同点是都使用了极径-极角坐标系来表示图像中的几何形状，并利用了极径-极角坐标系中的性质来检测出图像中的几何形

2.3.1 cv2.HoughLinesP

cv2.HoughLinesP 是 OpenCV 中的一个函数，用于检测图像中的直线。它使用了概率霍夫变换算法来实现。

cv2.HoughLinesP()函数的参数如下：

- **image**：输入图像，要求是灰度图像。
- **rho**：表示极径的分辨率，即直线的分辨率。 具体的，如果图像的分辨率为 (rows, cols)，则直线的分辨率为 (2*rho+1)。 rho 越大，直线的分辨率越低，检测出的直线越少，反之则越多。
- **theta**：表示极角的分辨率，即直线的角度分辨率。 theta 越大，直线的角度分辨率越低，检测出的直线越少，反之则越多。
- **threshold**：直线检测阈值，表示在一条直线上至少应该有多少个像素，才能被认为是一条真正的直线。 这个参数可以帮助排除图像中的噪声。
- **minLineLength**：最小线段长度，表示一条直线所应该具有的最小长度。 如果一条直线的长度小于这个值，则认为它是噪声，并且不会被检测出来。
- **maxLineGap**：最大直线间隔，表示两条直线之间的最大间隔。 如果两条直线之间的距离大于这个值，则认为它们不属于同一条直线，即使它们共线也是如此。

霍夫直线变换的数学原理是求解图像中直线的方程。 对于一条直线，它的方程可以表示为 $y = kx + b$，其中 k 是斜率，b 是截距。 因此，我们要做的就是求出图像中所有直线的 k 和 b 值。在霍夫直线变换中，我们将图像中的每个像素看作是一个点，然后计算出这些点之间的关系，从而推断出图像中存在的直线。 具体来说，我们假设图像中有一条直线，并求出这条直线的 k 和 b 值。 然后，我们计算出这条直线在图像中所有点的 y 坐标，如果这些点的 y 坐标与实际的 y 坐标相差不大，则认为这条直线是合法的。我们重复上述过程，求出图像中所有可能的直线，然后根据实际情况选择最合适的直线。

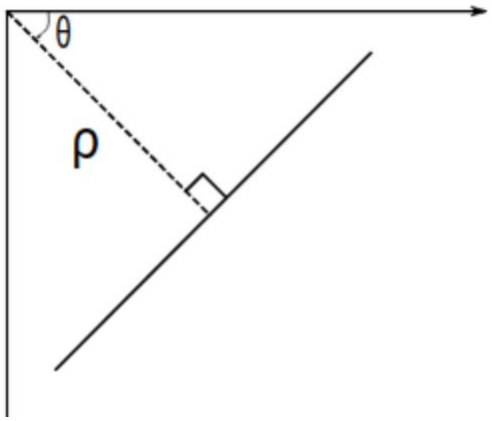

图 4 极坐标形式

在 cv2.HoughLinesP() 函数中，我们可以通过参数 rho 和 theta 采取极坐标的形式来控制对直线的搜索精度。即 rho = x cos (theta) + y sin (theta)。这样，最后，(rho,theta) 单元格累积数越多,说明过这条线的点越多,当大于某一阈值 threshold 的时候,我们说这是我们要找的一条直线。其中参数 minLineLength 和 maxLineGap 分别用来控制直线的最小长度和最大间隔。

调用 cv2.HoughLinesP()函数后，会返回一个包含所有检测到的直线的列表 lines。每条直线都是一个 4 元组（x1, y1, x2, y2），表示直线的两个端点坐标。

2.3.2 cv2.HoughCircles

cv2.HoughCircles()是一种用于在图像中检测圆的方法，它使用的是 "霍夫圆变换" 算法。

霍夫圆变换是一种用于在图像中检测圆的方法。 它的基本思想是，将图像中的每个像素看作一个点，然后对这些点进行统计，以查找像素之间的关系，从而推断出图像中存在的圆。

cv2.HoughCircles()函数使用霍夫圆变换算法来检测图像中的圆。 它的参数包括：

- **image:** 输入图像，应该是灰度图像或二值图像。
- **method:** 定义使用哪种方法来检测圆，目前支持的方法有 cv2.HOUGH_GRADIENT 和 cv2.HOUGH_MULTI_SCALE。
- **dp:** 图像的分辨率与圆的分辨率之比，具体的，如果图像的分辨率为 (rows, cols)，圆的分辨率为 (2*dp+1)，则 dp 越大，圆的分辨率越低，检测出的圆越少，反之则越多。
- **minDist:** 两个圆之间的最小距离，如果两个圆的圆心之间的距离小于 minDist，则认为这两个圆是同一个圆。

- **param1:** 第一个辅助参数，具体用途取决于 method 参数的值。 对于 cv2.HOUGH_GRADIENT 方法，param1 表示阈值 1，即检测圆的阈值。 这个参数可以帮助排除图像中的噪声。

- **param2:** 第二个辅助参数，具体用途取决于 method 参数的值。 对于 cv2.HOUGH_GRADIENT 方法，param2 表示阈值 2，即圆心检测的阈值。 这个参数越大，检测出的圆越少，反之则越多。

- **minRadius:** 最小圆半径，表示圆的半径所应该具有的最小值。 如果圆的半径小于这个值，则认为它是噪声，并且不会被检测出来。

- **maxRadius:** 最大圆半径，表示圆的半径所应该具有的最大值。 如果圆的半径大于这个值，则认为它是噪声，并且不会被检测出来。

霍夫圆变换的数学公式是求解图像中圆的方程。 对于一个圆，它的方程可以表示为 (x - a)^2 + (y - b)^2 = r^2，其中 (a, b) 是圆心坐标，r 是圆的半径。 因此，我们要做的就是求出图像中所有圆的圆心坐标和半径。

在霍夫圆变换中，我们将图像中的每个像素看作是一个点，然后计算出这些点之间的关系，从而推断出图像中存在的圆。具体来说，我们假设图像中有一个圆，并求出这个圆的圆心坐标和半径。 然后，我们计算出这个圆在图像中所有点的距离，如果这些点的距离与实际的圆的距离相差不大，则认为这个圆是合法的。我们重复上述过程，求出图像中所有可能的圆，然后根据实际情况选择最合适的圆。

cv2.HoughCircles()函数返回值是一个包含圆信息的列表，每个元素都是一个三元组 (x, y, r)，表示圆的圆心坐标 (x, y) 和圆的半径 r。

三、解决方法

解决方法分两部分，第一部分是使用强化学习的方法对智能体进行训练，主要是使用结合自博弈方法的 PPO 算法来对模型进行训练，从而得到智能体的策略模型；第二部分是通过观察训练得到的策略模型的实际表现，发现训练得到的策略存在的问题，在此基础上添加一些固定规则，作为策略模型的补充。在实际应用时，大部分情况下使用策略模型的策略，在一些特殊情况下，使用固定规则。

3.1 结合自博弈策略的 PPO 模型

3.1.1 observation 处理

由于 observation 中有效数据只有代表边界、地标线、我方 agent 和敌方 agent 的数据，我们将 observation 从 $40*40$ 处理成 $40*40*4$ ，即 4 个 $40*40$ 的像素图，每张对应一种类型的数据，若该类型数据在该点不为 0，则置为 1，否则为 0。同时，为了获取历史信息，使模型能够了解敌我双方速度或预测敌方位置，我们将四个 step 的 observation 的数据进行叠加，最终输入模型的 state 的维度为 $4*40*40*4$ 。

3.1.2 action 处理

由于我们使用 PPO 算法，最终输出的动作原本是离散的。因而我们将动作空间离散化为 31 个离散的 force 和 angle ，基本可以覆盖整个动作空间。

3.1.3 reward 设计

由于比赛环境代码开源，因而我们可以不仅仅根据是否获胜得到一个 sparse 的 reward，而可以根据环境的内部数据，如智能体的位置等设计稠密的奖励。对于每一个 step，定义 L_t^{my} 为当前 step 和上一 step 自己的位置到中心距离的平方差的平方： $L_t^{my} = (l_t^{my})^2 - (l_{t-1}^{my})^2$ 。

定义 L_{t}^{opp} 为当前 step 和上一 step 对手位置到中心位置的差的平方：$L_{opp} = (l_t^{opp})^2 - (l_{t-1}^{opp})^2$ ，将 L_t^{my} 和 L_t^{opp} 的差作为基础奖励，这样设计可以鼓励我方 agent 将对方 agent 往边界推，同时我方 agent 保持靠近中心位置。使用平方而非绝对值，是为了使 agent 在 map 上不同位置的奖励存在差别。这样当二者到中心距离的差保持不变时，越靠近 map 中心奖励越小，越靠近边界奖励越大，鼓励 agent 将对手完全撞出擂台。 由于能量在 observation 中不再提供，我们还需要使 agent 学会节约能量，但又不能过度约束智能体的行为，使其畏首畏尾。因而设定一个能量阈值（ th_{energy} ），当 agent 的能量小于能量阈值时，会产生一个指数增长的惩罚，使 agent 在能量较小时能够降低力的输出，恢复能量。 能量计算公式为：

$$ep_t^{my} = \begin{cases} 0 & if\ energy_t^{my} > th_{energy} \\ \mathrm{e}^{\frac{th_{energy} - energy_t^{my}}{17}} & if\ \mathrm{energy}_t^{my} \le th_{energy} \end{cases}$$

此外，我们还需要对两项奖励进行归一化，同时将双方奖励作差，使奖励零和。整体奖励公式为：

$$reward_t = \begin{cases} \frac{1}{2250}(L_t^{opp} - L_t^{my} - ep_t^{opp} + ep_t^{my}) & if \quad not\ finished \\ 40 & if \quad win \\ -40 & if \quad lose \\ 0 & if \quad tie \end{cases}$$

3.1.4 模型结构

所采用的模型结构如下图所示：

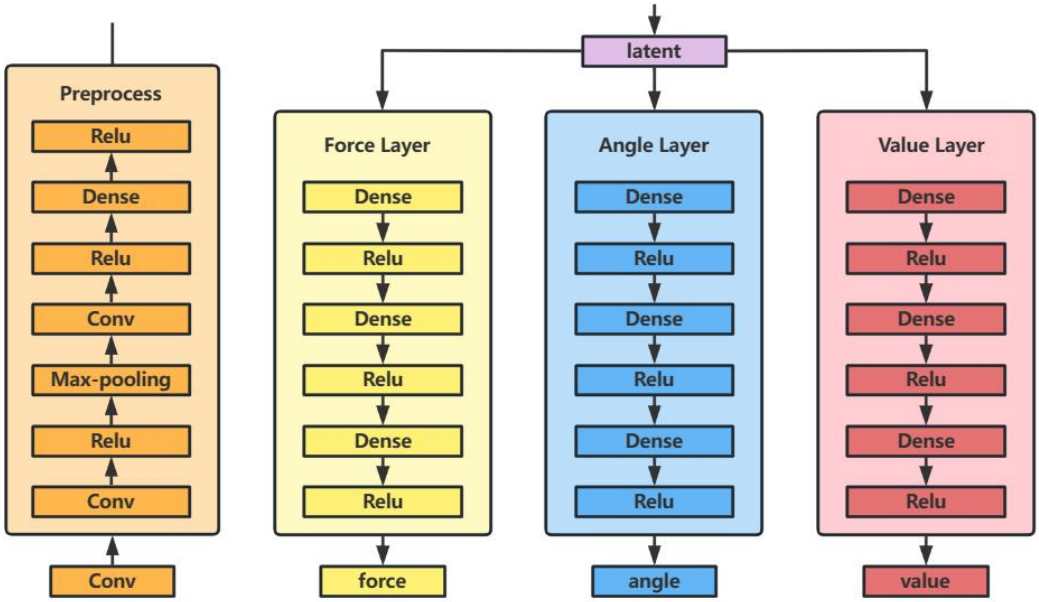

图 5 模型基础结构图

3.1.5 训练算法

模型采用 PPO 算法，先自博弈进行训练。当模型具备一定水平后，采用固定对手模型，训练一端模型的方式，使模型水平不断爬升。

下图展示了如何建立一个系统，通过自博弈 PPO，训练神经网络玩双人游戏。

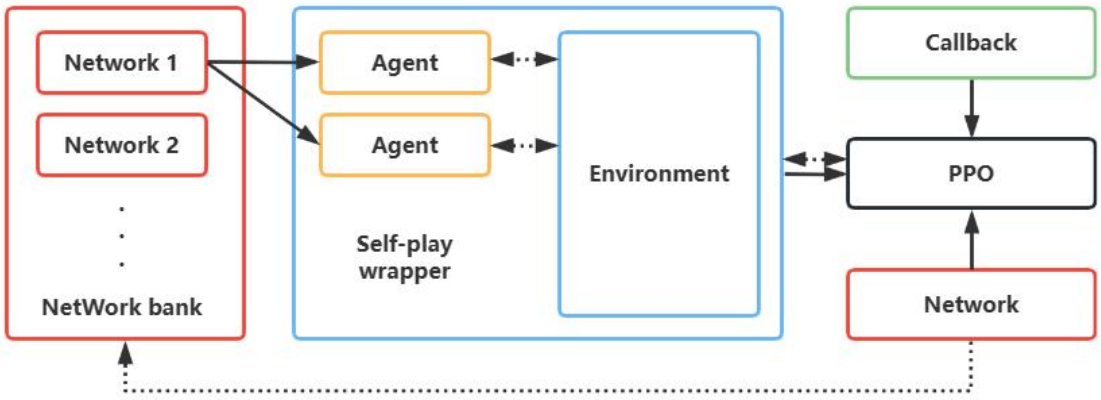

图 6 自博弈 PPO 算法原理图

其中的关键是将游戏环境打包到了一个“Self-play wrapper”（自博弈打包器）中。这个打包器有三个重要的作用：

- 在每次重置环境时，随机加载某个先前的版本，作为正在接受训练的网络的下一个对手。
- 通过策略网络采样，逐个挑战对手。
- 延迟向 PPO 引擎返回奖励，直到轮到其他智能体。

从本质上来说，自我博弈打包器将双人游戏环境转换成了不断变化的单人游戏，因为在每次游戏结束后，它都会创建并加载新的对手。此外，在双人相扑游戏中存在延迟奖励，即如果你是先移动的一方，那么你的动作不会立即产生奖励，你需要等待对手的移动情况，才能对自身奖励进行更新。因此，为了处理这种延迟，在将奖励返回给接受训练的 PPO 智能体之前，自我博弈打包器必须记录该智能体的动作，以及所有其他对手的动作。

综上所述，加载之前的网络、记录对手的动作以及延迟奖励的结合是所设计的结合自博弈策略的 PPO 模型训练算法的关键。

3.2 固定规则

上述自博弈方法设计的奖励函数会对智能体较低的能量给予惩罚，防止智能体在比较危险的情况下因为能量较低而失去控制，从而导致被撞下擂台。上述奖励函数的特点导致自博弈算法训练得到的智能体不会猛烈进攻，而是采用一种迂回战术、躲避和进攻相结合，可以维持能量不被消耗，但是也对对手威胁不大。

基于上述情况，设计本节的固定规则作为对自博弈方法的补充，即在一般情况下使用自博弈模型的策略得到动作，在特定情况下，使用本节设计的固定规则。

本节设计的固定规则主要包含两部分：进攻规则、防御规则。当对手出现在视野中时，可以精确计算对方的位置，然后据此封锁对方智能体，将其逼出边界，即进攻规则；当智能体在擂台边界时，向擂台中心移动，即防御规则。

上述固定规则的想法不是很复杂，但是使用上述规则的前提是能够得到智能体的位置信息，在使用进攻规则时还要能得到对手智能体的位置信息，因为智能体只能得到周围环境的像素图，所以需要根据局部的观测来得到智能体全局的位置，因而只能根据视野中的边界信息和地标线信息来判断当前位置。分析后，我们最终才用了边界、垂直交线和小同心圆三种方法来确定擂台的中心，从而确定两个智能体的位置。

3.2.1 智能体全局坐标计算

3.2.1.1 坐标系

本节介绍如何根据智能体的局部观测信息得到智能体在擂台上的位置信息，在此之前，需要建立坐标系，方便对智能体的位置信息进行量化描述。

如下图为简化后的可视化界面，如下界面中建立两套坐标系：全局坐标系 XOY，以可视化界面左上角顶点为坐标原点，向右为 X 轴，向下为 Y 轴；局部坐标系$X'O'Y'$，以智能体正方形视野的左上角为坐标原点，向右为X'轴，向下为Y'轴。

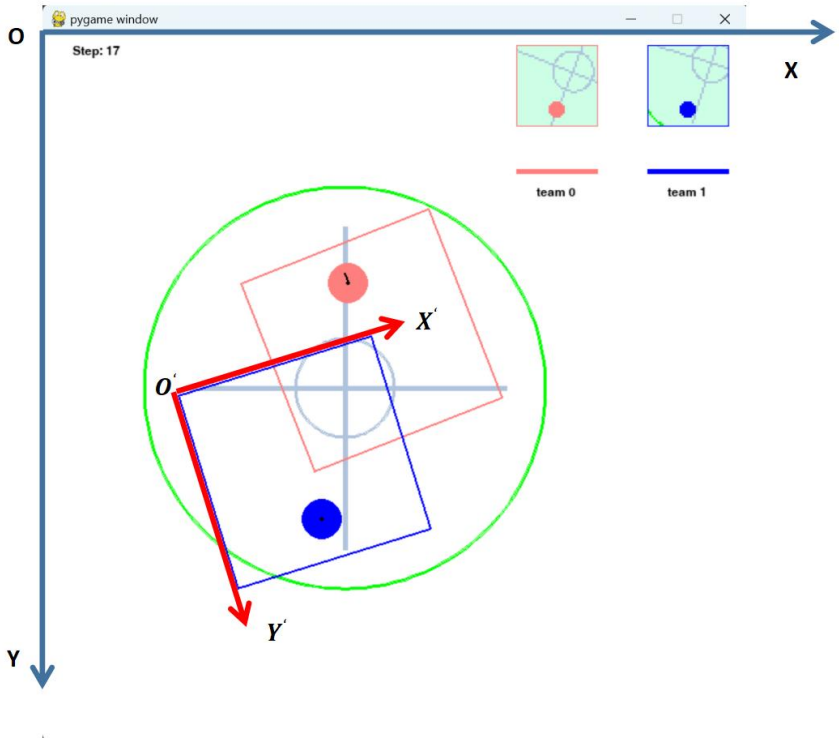

图 7 坐标系图

在该游戏环境中有两种距离度量方式，像素距离与实际渲染距离。智能体的局部观测为 40 × 40 的像素图，在对智能体的局部观测进行分析时，常用像素距离来度量。渲染时，一个像素对应实际距离 5，后续在分析智能体全局坐标时，多用实际渲染距离分析。

可视化界面的一些长度信息：使用实际渲染距离：擂台半径为 200；使用像素距离，智能体的视野范围为 40 × 40，所以智能体正方形视野的边长刚好与擂台半径相等。

可视化界面的一些关键坐标：使用像素距离：智能体中心的局部坐标为（19,32）；使用实际渲染距离，擂台中心的全局坐标为（300,350），红色智能体的初始全局坐标为（300,200），蓝色智能体的初始全局坐标为（300,500）。

3.2.1.2 边界法

若视野中出现擂台边界，则可根据擂台边界来确定擂台中心的局部坐标。具体首先遍历视野中的边界像素点，找到相距最远的两个点，由此可以确定视野范围内关于擂台边界的最长的一条弦，由于边界圆的半径已知，则可以计算得到擂台中心到该弦的距离，进一步可以得到擂台中心的局部坐标。（计算得到擂台中心的局部坐标，又智能体的局部坐标、擂台中心的全局坐标都是固定的，已知这三个坐标就可以得到智能体的全局坐标，具体见 3.2.1.5）

通过下图，对该方法做具体解释。下图点 O 为擂台中心，假定当前智能体的视野为下图的 FHIJ 正方形区域。通过两两比较视野中所有的边界点，可以得到相距最远的两个边界点 A、B，A、B 均在擂台边界圆 O 上，且 AB 为视野范围内最长的一条弦。已知 A、B 的

局部坐标(x_A, y_A)、(x_B, y_B)，则可以得到 AB 中点 H 的局部坐标$(x_H, y_H) = (\frac{(x_A+x_B)}{2}, \frac{(y_A+y_B)}{2})$，并可以得到 $\theta = \angle BAC = \arctan(\frac{y_A-y_B}{x_B-x_A})$，容易证明$\angle FOH = \angle BAC = \theta$。AB 的距离$L_{AB}$也可以计算得到，又擂台半径 OB 已知，记为 r，则可以得到 OH 的距离：$L_{OH} = \sqrt{(r^2 - \frac{L_{AB}^2}{2})^2}$。

已知 H 的局部坐标、$\angle FOH = \theta$、L_{OH}，则可以得到擂台中心 O 的局部坐标：$(x_H - L_{OH} \cdot \sin\theta, y_H - L_{OH} \cdot \cos\theta)$

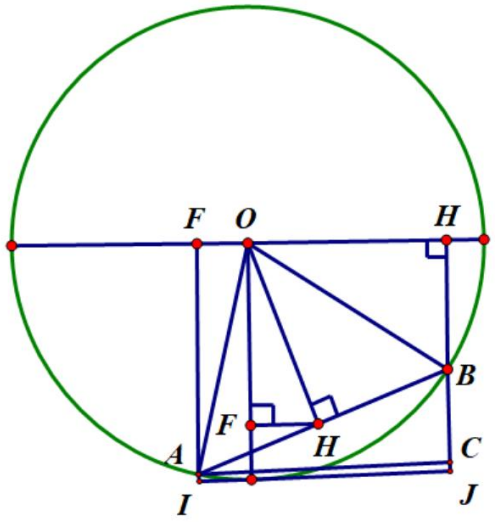

图 8 边界法示意图

已知擂台中心的局部坐标则可以通过 3.2.1.5 中方法得到智能体的全局坐标。

3.2.1.3 垂直交线法

利用 opencv 的霍夫变换 cv2.HoughLineP，可以找到地标线中的直线，两条直线的交叉点即为擂台中心点。由于地标线可能被我方或者敌方 agent 遮挡，或者未出现在 agent 的视野中，因而需要将霍夫变换中对于直线线长的阈值降低，但这样又有可能将小同心圆的弧检测为线段。最终在判定垂直交线时，需要限定两直线斜率的范围，同时选择最长两条直线相交产生的中心点作为擂台中心点。

通过上述方式则可得到擂台中心的局部坐标，与边界法类似，可以通过 3.2.1.5 得到智能体的全局坐标。

3.2.1.4 同心圆法

利用 opencv 霍夫变换 cv2.HoughCircles，可以找到地标线中的小同心圆，其圆心即为中

心点。检测时可能在中线小同心圆检测到多个圆，因此需要对找到的多个圆心取平均值。由于大多数时候视野中只有圆弧，所以也需要降低检测的阈值，但这样可能会将圆弧和直线的相交处判定为圆弧，因此小同心圆的检测一般只作为辅助检测。

通过上述方式则可得到擂台中心的局部坐标，与上述方法类似，可以通过 3.2.1.5 得到智能体的全局坐标。

3.2.1.5 全局坐标计算

如下图，点 O 为擂台中心，其全局坐标(X_0, Y_0)是一个固定值，为(300,350)，点 H 为智能体所在位置，正方形 ABCD 为智能体的视野范围，OL 方向为蓝色智能体初始化朝向的方向，HM 与 AD、BC 平行，交 OL 于 K 点，∠LKM 记为 angle，即为当前智能体的朝向方向。每个时间步，智能体采取两个动作：一个是力的大小、一个是施加力的角度，将之前所有时间步施加力的角度叠加就可以得到∠LKM 即 angle。使用像素距离，点 H 的局部坐标为$(x_H, y_H) = (19,32)$，是固定的；点 O 的局部坐标通过上述的三种方法可以得到，记为(x_O, y_O)。HN、ON 分别与正方形视野的边界 AB、AD 平行，可以得到$\theta = \angle HON = \arctan(\frac{y_H - y_O}{x_O - x_H})$，同时 OH 的长度$L_{OH}$也可以计算得到。显然 $ON//MH$，所以$\angle NOR = \angle HKR = \angle LKM = \theta$，则$\angle HOR = angle + \theta$。

则对蓝色智能体，其全局坐标(x, y)可以通过如下方式得到：

$$HR = \Delta x = L_{OH} \cdot \sin(\theta + angle)$$

$$OR = \Delta y = L_{OH} \cdot \sin(\theta + angle)$$

$$(x, y) = (X_0 - \Delta x * 5, Y_0 + \Delta y * 5)$$

上式中得到的 Δx 与Δy 为像素距离，在转为全局坐标时，使用实际渲染距离，所以对Δx、Δy 乘 5。

类似的，只要视野中出现对手，则可以得到对手的局部坐标，则可以使用上述方法计算得到对手的全局坐标。

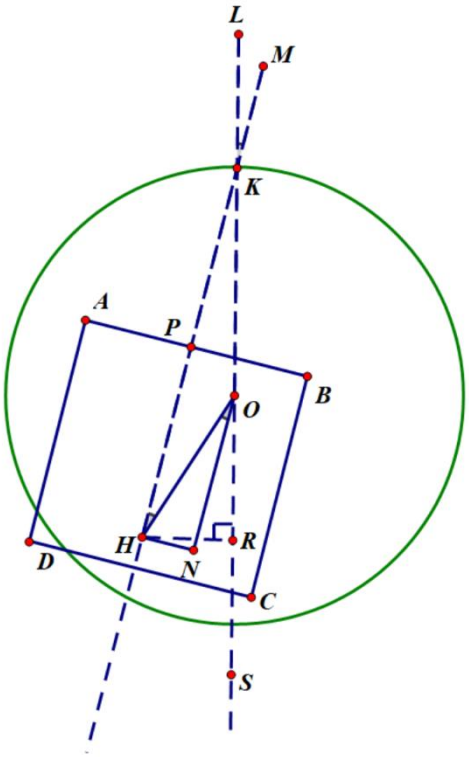

图 9 全局坐标计算示意图

3.2.2 防御规则

当视野中出现擂台边界，或者当前智能体与擂台中心相距较远，则使用防御规则，具体如下：

若在视野中存在边界，则采用固定规则向地图中心移动：若 agent 朝向中心则直接朝向中心前进；若背对中心且速度也背向中心，则反向输出力倒退移动；否则不输出力，并旋转朝向中心。

3.2.3 进攻规则

当视野中出现对手时，可以精确计算得到对手智能体的坐标，这时使用固定规则，具体如下：计算两个智能体分别到中心的距离L_{my}、L_{opp}

若$L_{my} > L_{opp}$，则智能体朝擂台中心移动，即$(300,350)$；

若$L_{my} \leq L_{opp}$：计算两智能体与擂台中心连线的夹角，即两智能体的圆心角，若该圆心角足够小，且我方智能体比敌方智能体靠近中心一个球身，即$L_{opp} - L_{my} > 20$，则朝向敌方智能体与擂台中心连线距离敌方 30 处运动；若领先不够多，则朝向擂台中心运动。若圆心角较大，则朝敌方智能体与擂台中心连线距离敌方L_{my}处运动。

四、效果

用户名：瓦瑞尔

排名：榜眼（第二）

排名链接：http://www.jidiai.cn/env_detail?envid=74

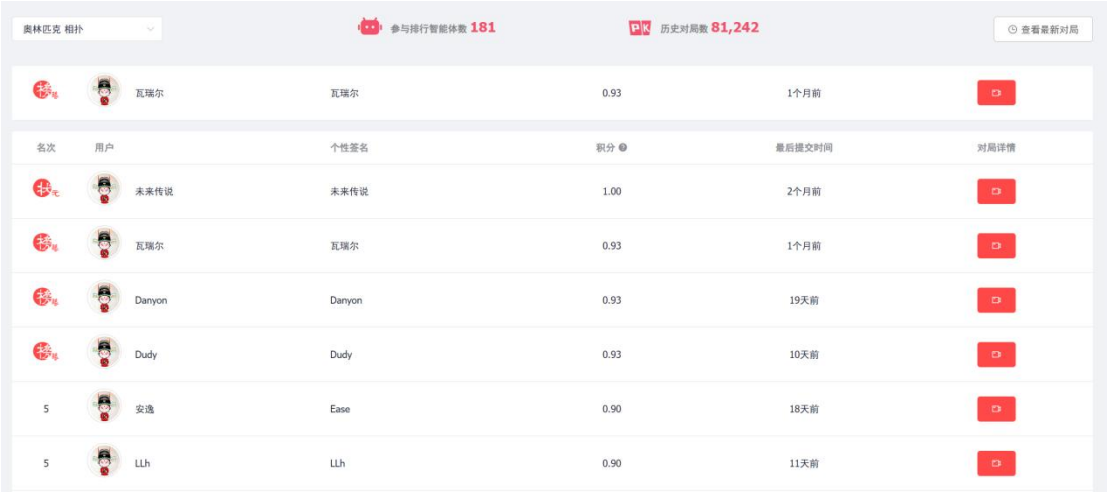

图 10 排名截图

排名说明：大作业开始的时候，这个竞赛只剩大约一周就结束了，上面的截图是在及第提供的测试平台上提交后的排名。（根据正赛的报名信息可以看到，测试平台排名第一的用户在正赛是排名第三，可以作为一个排名的参考）