|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | | | |
| **武汉大学计算机学院**  **本科生课程设计报告** | | | |
|  | | | |
| **面向避障无人车放射源定位应用的**  **多智能体系统实现方案** | | | |
|  | 专业名称： | 空间信息与数字技术 |  |
| 课程名称： | 人工智能前沿技术 |
| 指导教师： | 谢榕 教授 |
| 学生学号： | 2022302191584 |
| 学生姓名： | 周珏帆 |
|  | | | |
| 二〇二四年十二月 | | | |

|  |
| --- |
|  |

目 录

[目 录 II](#_Toc185816155)

[第1章 概述 3](#_Toc185816156)

[1.1 研究意义 3](#_Toc185816157)

[1.2 国内外研究现状 4](#_Toc185816158)

[1.3 研发目标 6](#_Toc185816159)

[第2章 技术方案 7](#_Toc185816160)

[2.1 技术总框 7](#_Toc185816161)

[2.1.1 MATD3算法…………………………………………………………….7](#_Toc185816164)

[2.1.2 MASAC算法……………………………………………………………8](#_Toc185816164)

[2.1.3 MADDPG算法……………………………………………………….…8](#_Toc185816164)

[2.1.4 开发技术栈……………………………………………………………..9](#_Toc185816164)

[2.2 技术细节…………………………………………………………………...10](#_Toc185816162)

[2.2.1 辐射模型………………………………………………………………10](#_Toc185816163)

[2.2.2 传感器模型……………………………………………………………10](#_Toc185816164)

[2.2.3 模型网络架构…………………………………………………………11](#_Toc185816165)

[2.2.4 模型参数设置…………………………………………………………13](#_Toc185816164)

[第3章 系统模拟 16](#_Toc185816166)

[3.1 多智能体场景构建 16](#_Toc185816167)

[3.2 多智能体深度强化学习算法开发 19](#_Toc185816168)

[3.2.1 Q值网络与策略网络………………………………………………….19](#_Toc185816164)

[3.2.2 回放缓冲区…………………………………………………...……….20](#_Toc185816164)

[3.2.3 MASAC算法：高斯策略网络………………………………………..21](#_Toc185816164)

[3.2.4 三种模型定义…………………………………………………………23](#_Toc185816164)

[3.3 模型训练 26](#_Toc185816169)

[3.3.1 模型训练过程 26](#_Toc185816171)

[3.3.2 三种模型训练结果(损失曲线、奖励曲线)对比分析 28](#_Toc185816171)

[3.4 模型测试与结果评估 33](#_Toc185816170)

[3.4.1 常规场景下三种模型性能对比与分析………………………………34](#_Toc185816164)

[3.4.2 复杂场景下三种模型性能对比与分析………………………………35](#_Toc185816164)

[第4章 结论 37](#_Toc185816179)

[参考文献 38](#_Toc185816180)

[附 录 41](#_Toc185816181)

# 概述

## 研究意义

面向避障无人车放射源定位应用的多智能体系统是一项具有重要实际意义的前沿研究，其核心目标是在复杂多变的环境中，实现对放射源的精准、快速定位，同时确保无人车系统的安全性和可靠性。随着核技术的广泛应用，放射源泄漏或丢失的潜在风险增加，如何在短时间内准确找到放射源并实施控制，已经成为环境保护、核安全和公共健康领域的重要问题。无人车因其自主性和高机动性，在放射源定位任务中扮演着重要角色，而将多个无人车集成为协同工作系统，则能有效提高定位效率，扩大应用范围，为核应急响应提供更强大的技术支持。

传统单一智能体定位方法面临诸多局限性，尤其是在动态障碍物密集分布的复杂环境中，其感知范围和决策能力往往无法满足实际需求。多智能体系统通过引入多个智能体共同参与任务，可以实现任务分解、信息共享和分布式协作。每个智能体可基于自身所感知的局部环境信息，自主规划行动路径，同时与其他智能体交换数据，共同优化整个系统的定位方案。这种基于群体智能的协作方式，显著提高了放射源定位的覆盖范围和精度，尤其是在大规模、动态性强的场景中，能够有效解决单一智能体无法完成的复杂任务。

避障能力是无人车系统实现自主运行的重要基础，尤其是在城市环境或工业园区中，障碍物的动态变化对无人车的导航和任务执行提出了严峻挑战。研究避障无人车在多智能体系统中的协同工作机制，可以实现路径规划与障碍规避的高度融合，使得无人车能够实时调整路径，以避免与障碍物发生碰撞，同时维持对放射源的高效搜索和精准定位能力。这种系统通过综合利用环境感知、行为规划和动态决策技术，不仅提高了无人车在复杂环境中的任务完成率，还显著降低了人工干预的需求，为无人系统在实际应用中的安全性和可靠性提供了重要保障。

除此之外，多智能体系统的研究具有重要的理论和技术推动作用。这一领域结合了人工智能、机器人学、分布式计算和环境感知等多个学科的最新进展，通过开发面向实际应用的系统解决方案，不仅为核安全和环境监测领域带来了颠覆性的技术提升，还为其他领域如智慧城市建设、无人化物流运输和智能交通系统提供了可行的技术借鉴。这一研究的开展，不仅能够满足当前放射源定位的迫切需求，还为未来更多应用场景的拓展奠定了坚实的理论基础和技术储备，对推动无人车技术的全面智能化和广泛普及具有深远意义。

## 国内外研究现状

**国内研究现状**

在国内，避障无人车放射源定位算法的研究近年来取得了长足进展。清华大学智能交通与控制研究中心基于多模态传感器融合技术，开发了一种放射源定位与避障算法。该算法利用激光雷达、深度相机和红外探测器，结合路径规划和强化学习，显著提升了无人车在复杂动态环境中的鲁棒性。实验结果显示，该技术在核电站内部环境中表现出优异的障碍物规避和高精度定位能力，被广泛认为是国内技术前沿的代表。

百度Apollo无人驾驶平台虽然以城市无人驾驶为主，但其研发的路径规划和障碍物检测算法已延伸至核设施放射源监测领域。在模拟核泄漏演练中，Apollo系统通过高精度地图和多传感器数据融合实现了放射源快速定位和避障任务。另一个典型案例是北京航空航天大学与中国核工业集团合作的无人车放射源探测项目，结合无人车集群技术和分布式定位算法，实现了在复杂室内环境中的高效避障和目标定位。

此外，中科院自动化研究所也开发了一种基于强化学习和SLAM技术的动态避障算法，用于放射源定位。该技术已在某核设施巡检机器人中试运行，在实时性和可靠性方面均表现出色。比如，在某核泄漏模拟实验中，该机器人在2分钟内完成了对目标区域的全覆盖搜索，成功绕过动态障碍物并定位放射源。

**国外研究现状**

国外研究在避障无人车放射源定位算法领域走在前列。例如，Google的Waymo团队开发的路径规划算法虽然主要应用于城市环境，但其在放射源定位场景中同样表现出色。在一项模拟核泄漏的应急演练中，Waymo系统利用多模态传感器数据融合和深度强化学习算法，在高辐射区域中快速规避障碍物并精确定位放射源。此外，美国洛斯阿拉莫斯国家实验室（Los Alamos National Laboratory）开发了一种基于概率图模型的放射源定位算法，通过实时传感器数据融合和路径优化，在复杂环境中实现了高效的动态避障。

英国牛津大学的Oxford Robotics Institute专注于多智能体协作算法的研究，开发了“分布式任务分配与定位系统”。该系统通过无人车之间的实时通信，动态调整每辆车的任务，使得整个集群可以在复杂环境中实现快速放射源定位。在Sellafield核电站的一次实验中，这一系统成功完成了对多个放射源的联合定位和障碍规避任务。

德国的马普智能系统研究所（Max Planck Institute for Intelligent Systems）开发了一种利用强化学习和进化算法相结合的放射源定位技术。这项技术特别适用于动态障碍密集的核设施环境，例如，在德国某核废料处理厂的测试中，该系统在10分钟内成功完成了对目标区域的全面扫描和放射源定位。

**国内外研究差异**

国内外在避障无人车放射源定位算法的研究侧重点有所不同,国外研究更注重技术的实际应用。例如，Waymo系统已经在多城市环境中通过大量实地测试验证了其路径规划和障碍物规避能力，并将其技术拓展至核事故应急场景。相较之下，国内的研究更多集中在理论层面和实验室验证。例如，清华大学和北京航空航天大学的研究主要在仿真环境中进行，实际部署场景较少。

在多智能体协作方面，国外的技术优势更为显著。例如，Oxford Robotics Institute的分布式协作算法在高辐射环境中展示了卓越的任务分配和协同能力，而国内类似研究在信息共享和实时通信方面仍有提升空间。此外，在高精度传感器硬件和复杂算法实现上，国外的研究机构如洛斯阿拉莫斯国家实验室拥有更成熟的设备支持，而国内尚处于设备改进和技术集成的阶段。

**未来展望**

未来，随着5G通信、边缘计算和人工智能技术的发展，避障无人车放射源定位算法将在实际应用中迎来更广阔的前景。一个重要方向是无人车与无人机的联合应用。例如，通过无人机的全局监测与无人车的精准定位相结合，可以在核事故应急中实现更高效的放射源探测。此外，多智能体系统的协同工作也将成为研究重点，特别是利用5G网络实现多车实时通信和任务分配，提升协作效率。

在工业应用中，无人车放射源定位技术有望进一步拓展至核设施巡检、环境放射性监测和废料处理等领域。例如，在某次核电站泄漏演练中，结合无人车和无人机的协同算法，仅用时5分钟便完成了全覆盖搜索并精准定位泄漏源。这种高效、智能化的技术应用将大幅减少人工干预，降低核泄漏环境下的人员风险，为核安全领域提供重要保障。

最终，通过深度学习与强化学习技术的结合，未来的避障无人车系统有望实现更高的自主性和智能性。例如，无人车可以通过不断学习复杂环境中的模式，在动态环境中快速调整策略，从而适应更多类型的任务需求，进一步推动这一领域的技术发展和产业化应用。

## 研发目标

本实验聚焦于“基于避障无人车的放射源定位”技术，采用三种多智能体深度强化学习算法——MADDPG（Multi-Agent Deep Deterministic Policy Gradient）、MASAC（Multi-Agent Soft Actor-Critic）和MATD3（Multi-Agent Twin Delayed Deep Deterministic Policy Gradient）进行开发与验证。这三种算法将分别被用于设计无人车多智能体系统，使其能够在复杂环境中自主探索、智能避障、高效协作并精准定位放射源，同时实现路径的动态优化和多车体间的智能分工。

实验旨在系统地比较三种算法在多智能体场景中的性能表现，包括定位精度、避障能力、任务完成时间以及在动态复杂环境中的稳定性和鲁棒性。通过在多种仿真环境中的实验测试和实际场景的验证，研究每种算法在协作策略、计算效率和适应性上的差异，为无人车避障与目标定位的实际应用提供理论和实践支持。

在实验过程中，我将深入优化每种算法的核心结构与参数配置，如奖励函数设计、策略网络架构和更新频率等，探索多智能体系统中的通信机制、协作方式和环境感知能力对算法效果的影响。特别地，通过对比分析三种算法在不同任务负载、环境复杂度以及资源约束条件下的表现，挖掘其各自的优势和适用场景，为设计高效的多智能体协作框架提供指导。

本实验的最终目标是构建一个高效、稳定、具有强鲁棒性和适应性的无人车多智能体系统，不仅能够在复杂的动态环境中完成放射源的精准定位任务，还能为后续无人车技术在紧急救援、环境监测以及智能交通等领域的应用提供可靠的算法基础和实践经验。同时，通过全面对比分析三种算法的表现，为多智能体强化学习算法的选择与优化提供实用的决策依据。

# 技术方案

## 技术总框

**2.1.1 MATD3算法**

在我的研究中，MATD3被定义为一种基于行动者和评论者结构的多智能体深度强化学习方法。它是对单智能体算法TD3的扩展，能够适应多智能体环境。该方法采用集中式训练与分布式执行的混合模式，值函数的输入包括每个智能体自身以及其他智能体的观测和动作信息，而策略函数只依赖于智能体自身的观测。作为MADDPG的优化版，MATD3通过使用两套独立的Q值网络（即双Q网络）来计算值函数，并取其中的最小值作为目标更新，从而有效抑制了高估问题。这种方法的总体框架如图2.1所示，清晰地展示了核心的实现思路和算法流程：

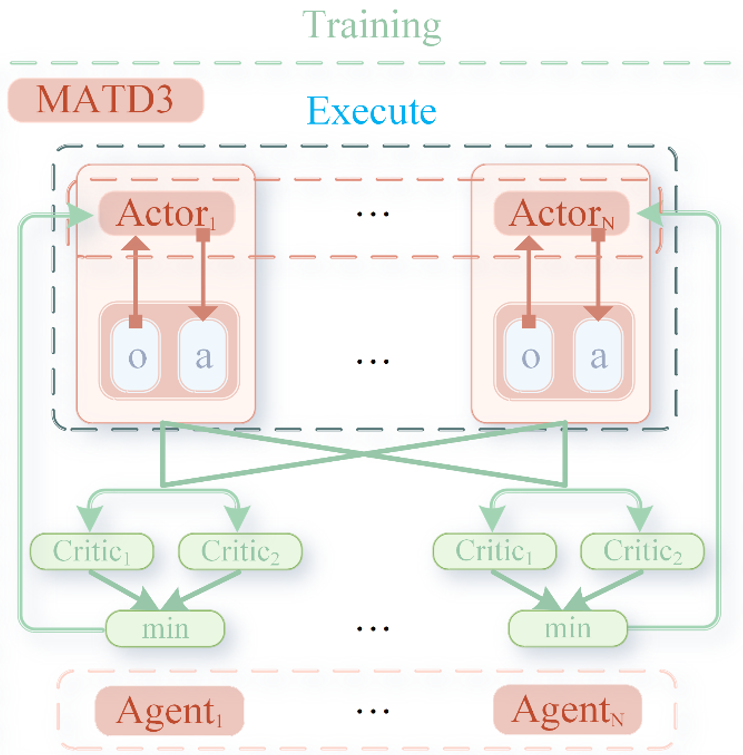


图2.1

如图2.1所示，为MATD3多智能体强化学习算法框架。其中“o”代表智能体的本地观测，“a”是策略网络（Actor）生成的输出。

**2.1.2 MASAC算法**

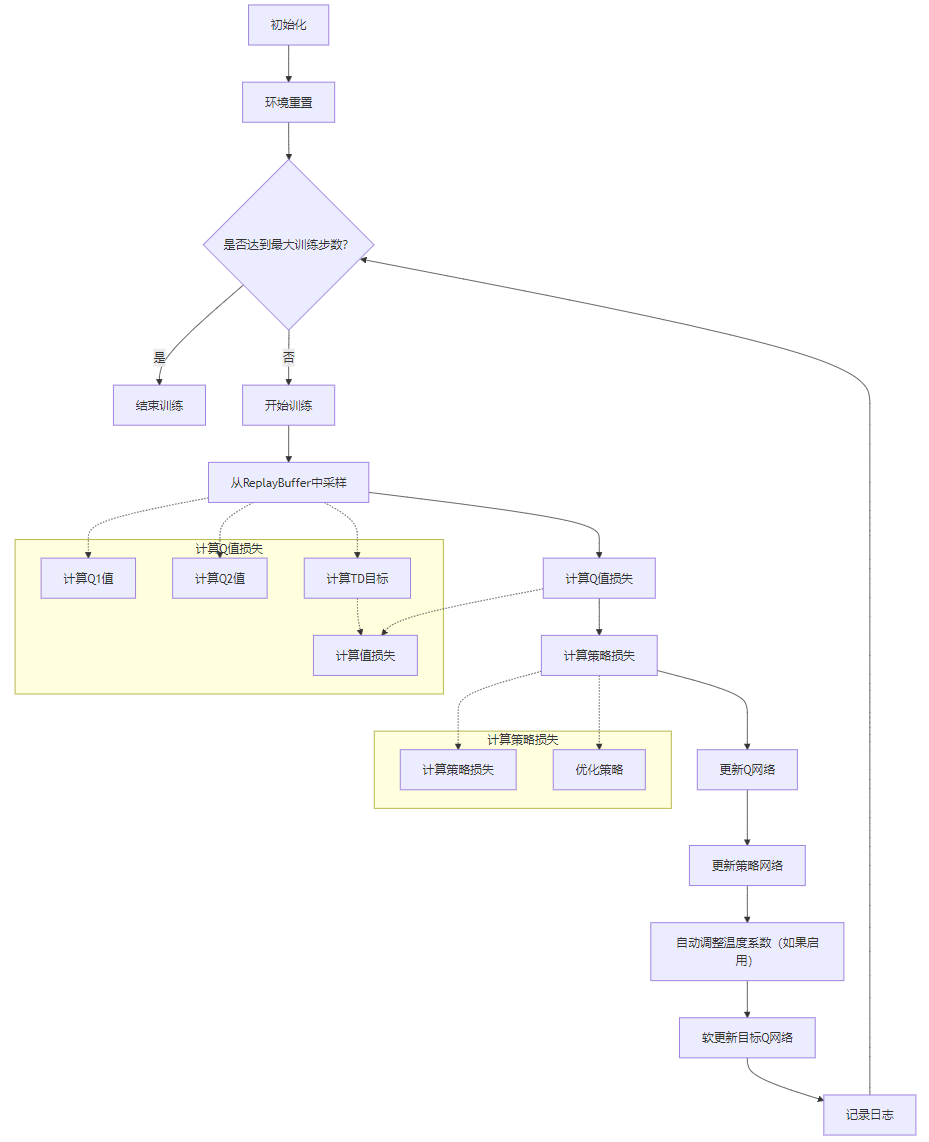


图2.2

如图2.2所示，MASAC算法的训练流程包括初始化环境和模型后，通过与环境的交互不断采样数据，并计算Q值和策略损失。算法利用Replay Buffer中的历史经验进行训练，更新Q网络和策略网络，优化策略的表现。训练过程中还可以自动调整温度系数（如果启用），并进行目标Q网络的软更新。训练过程持续进行，直到达到最大训练步数为止，并记录日志以跟踪训练进度和结果。

**2.1.3 MADDPG算法**

MADDPG算法的流程如图2.3所示，首先初始化环境和代理，并配置TensorBoard用于记录训练过程中的奖励和损失等指标。在每个回合开始时，重置环境，代理根据当前的训练步骤选择动作。若训练步骤不足以进行策略选择，则随机选择动作；否则，基于策略选择动作。代理执行动作后，将环境的转移（状态、动作、奖励、新状态）存储到回放缓冲区。然后，利用回放缓冲区的数据更新代理，并记录当前回合的统计信息和奖励。每20个回合保存一次模型，如果当前回合的奖励超过最大奖励，则保存最佳模型。这个过程持续进行，直到满足训练结束的条件：

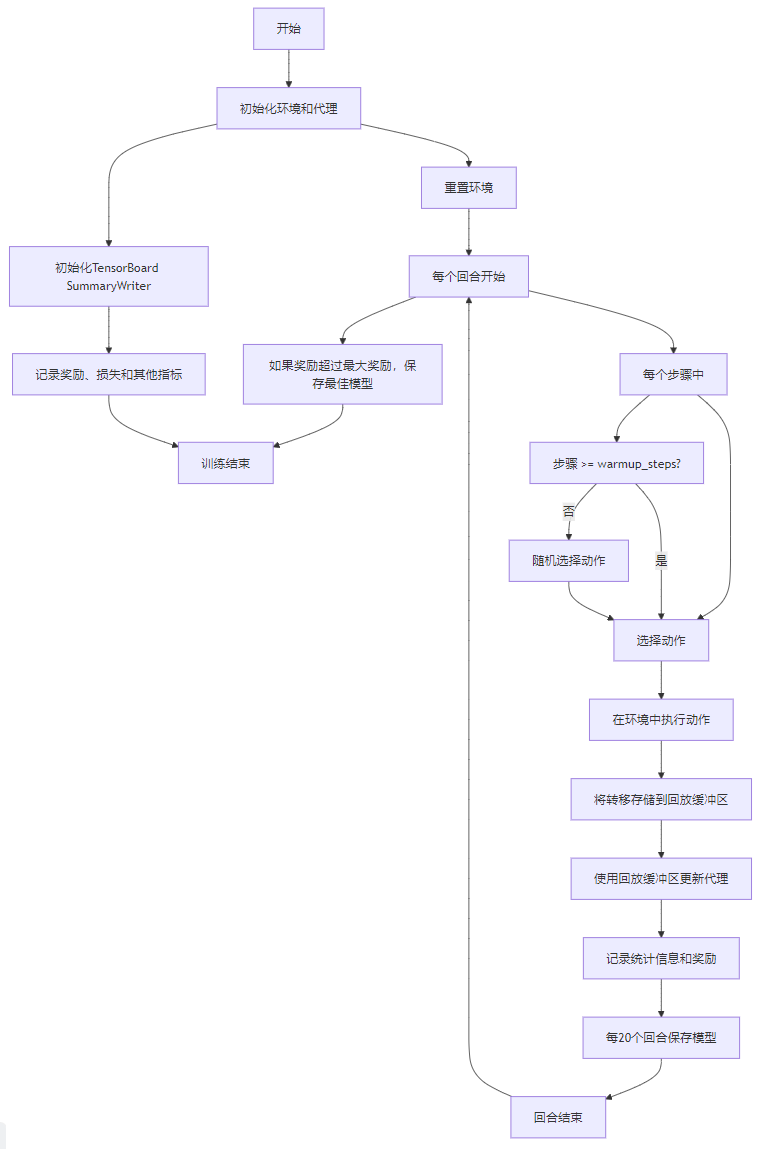


图2.3

**2.1.4 开发技术栈**

|  |  |
| --- | --- |
| 技术 | 描述 |
| 编程语言 | Python |
| 深度学习框架 | PyTorch |
| 开发环境 | |  | | --- | |  |   Python 3.9.21/torch 2.5.1+cu124  /torchvision 0.20.1 |
| 计算加速单元 | NVIDIA RTX 2050  CUDA 12.6 |
| 数据可视化 | Matplotlib，tensorboard |
| 图像处理 | PIL |
| 数据分析 | Numpy、tqdm |
| 强化学习环境接口 | gym、pettingzoo、gymnasium |

表2.1

## 技术细节

**2.2.1 辐射模型**

该辐射模型假设辐射源为各向同性点状伽马源，即源的线性尺寸远小于从源到感兴趣点的距离。搜索任务涉及代理在指定区域内寻找失踪的放射源（用表示）。其中，为源的状态向量，()表示源在二维平面中的位置，而I\_s代表放射源的强度。

(1)

在式(1)中,表示传感器的响应强度；S代表传感器的有效探测面积，为其探测效率。此外，为伽马射线穿透第i种介质的路径长度，则是第i种物质对伽马射线的衰减系数。变量R为传感器与放射源之间的距离，是背景辐射的计数值，即在没有样本的情况下探测器记录的计数。

**2.2.2 传感器模型**

尽管传感器的测量会受到噪声的影响，每次对辐射强度的测量值都会有所不同，但这些测量值通常围绕一个平均值波动。为了对辐射强度进行统计描述，采用泊松分布模型进行建模。因此，在τ秒的时间内，辐射传感器记录到来自每秒平均发射µ计数的源的z∈Z+计数的概率为：

(2)

其中，表示传感器测量到的实际强度，并且是泊松分布的参数。随机变量的均值和方差均为。因此，当传感器位于已知位置时，单位时间内的辐射计数测量的似然函数可以通过以下公式表示：

(3)

本研究考虑了一个多智能体协同搜索的场景，目标是寻找丢失的放射源。正如图2.4所示，放射源可能位于不同位置，如房间的角落、中央区域或靠近墙壁的位置。每个智能体都配备了激光雷达和辐射探测器，任务是协作完成搜索，尽可能快速地定位放射源，同时避免与障碍物或其他智能体发生碰撞。

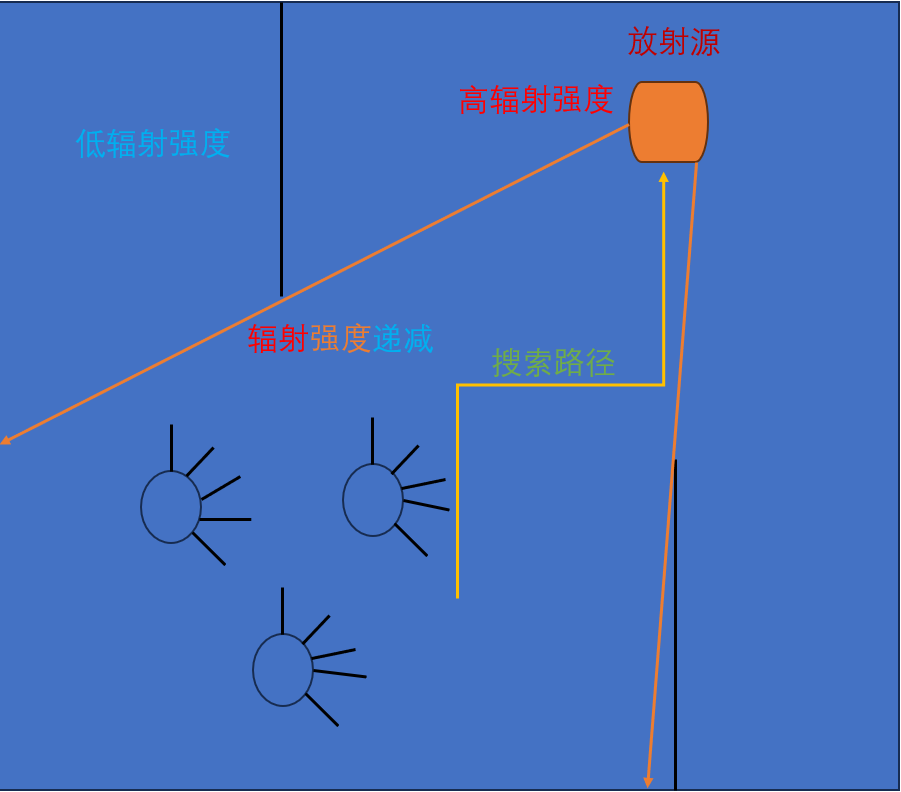


图2.4

**2.2.3 模型网络架构**

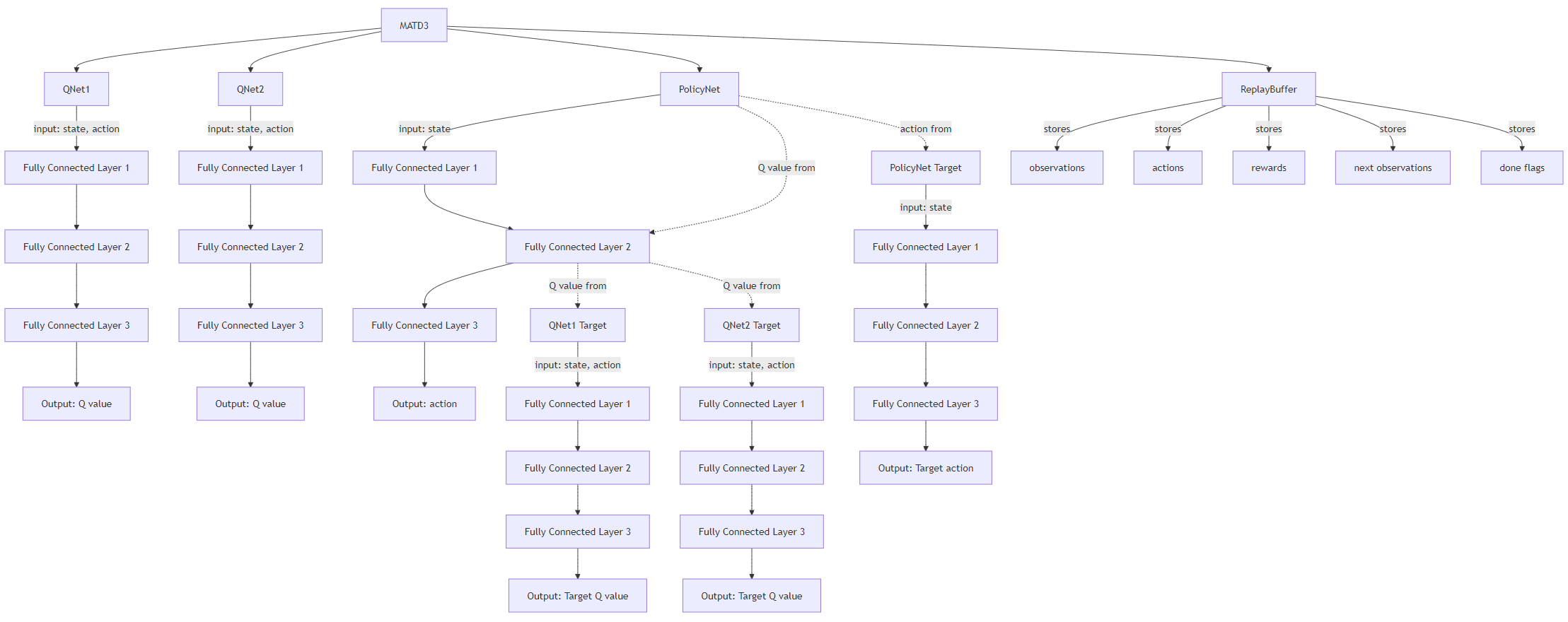


图2.5

如图2.5，MATD3（Multi-Agent Twin Delayed Deep Deterministic Policy Gradient）模型的网络架构包括多个关键组成部分。每个智能体都配备一个Q网络（QNet）和一个策略网络（PolicyNet）。QNet负责估计每个智能体在当前状态和动作下的Q值，其由多个全连接层（Fully Connected Layers）构成，输出为Q值。PolicyNet则根据当前状态输出一个动作，用于指导智能体的行为，通常通过多个全连接层生成动作的确定性策略。此外，MATD3还包含一个重放缓冲区（ReplayBuffer），用于存储和管理经验数据（例如，观察、动作、奖励等），并通过经验回放机制提升学习效率。网络通过双重延迟更新（Twin Delayed Updates）策略，采用目标Q网络和目标策略网络来稳定训练过程，减少过估计问题。整体架构通过这些组件相互协作，实现多智能体环境下的有效学习和决策。

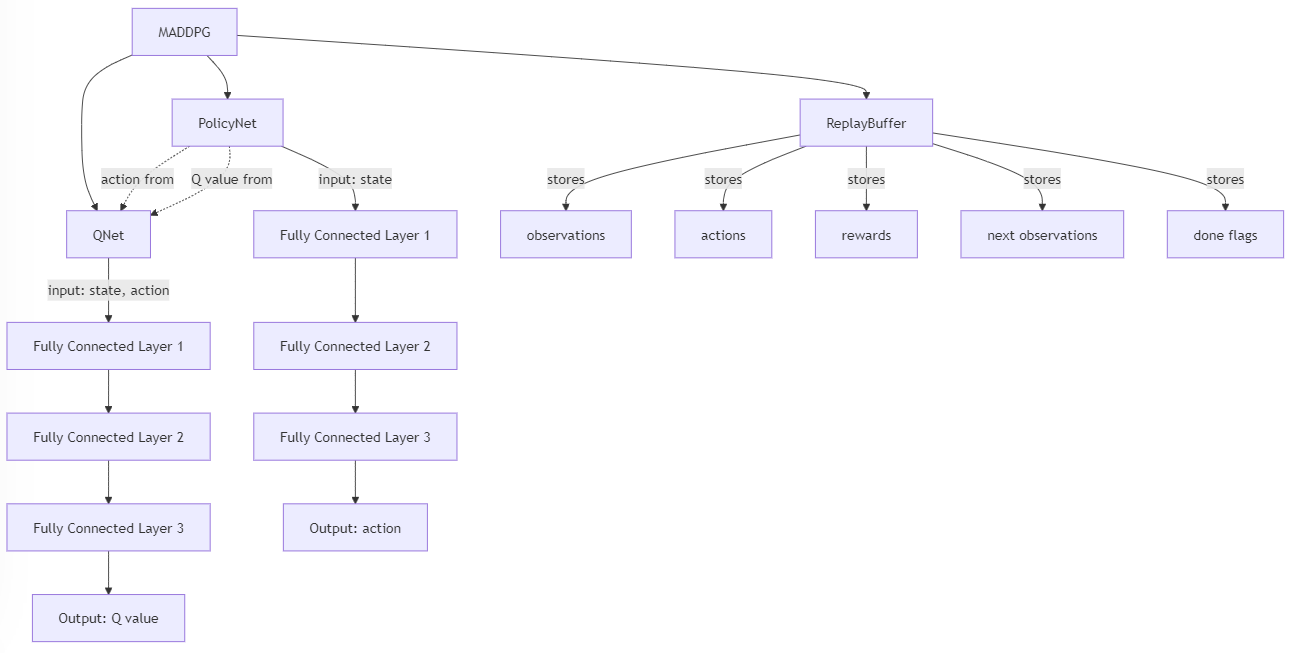


图2.6

如图2.6，MADDPG（Multi-Agent Deep Deterministic Policy Gradient）模型的网络架构由多个智能体的策略网络（PolicyNet）和Q值网络（QNet）组成。每个智能体有一个策略网络，用于根据当前的环境状态输出一个确定性的动作。策略网络由多个全连接层构成，通过最大化长期回报来优化策略。Q值网络则评估每个智能体在当前状态和动作下的Q值，通常包含多个全连接层。为了适应多智能体的环境，MADDPG中的每个智能体不仅依赖于自己的状态信息，还会将其他智能体的动作作为输入。重放缓冲区（ReplayBuffer）用于存储智能体的经验（如状态、动作、奖励等），通过经验回放来提升学习效率。此外，MADDPG使用延迟更新策略，独立更新每个智能体的策略和Q值网络，并通过目标网络（Target Networks）来稳定训练过程。整体架构通过多个智能体协同工作，提升在多智能体环境中的学习能力和性能。

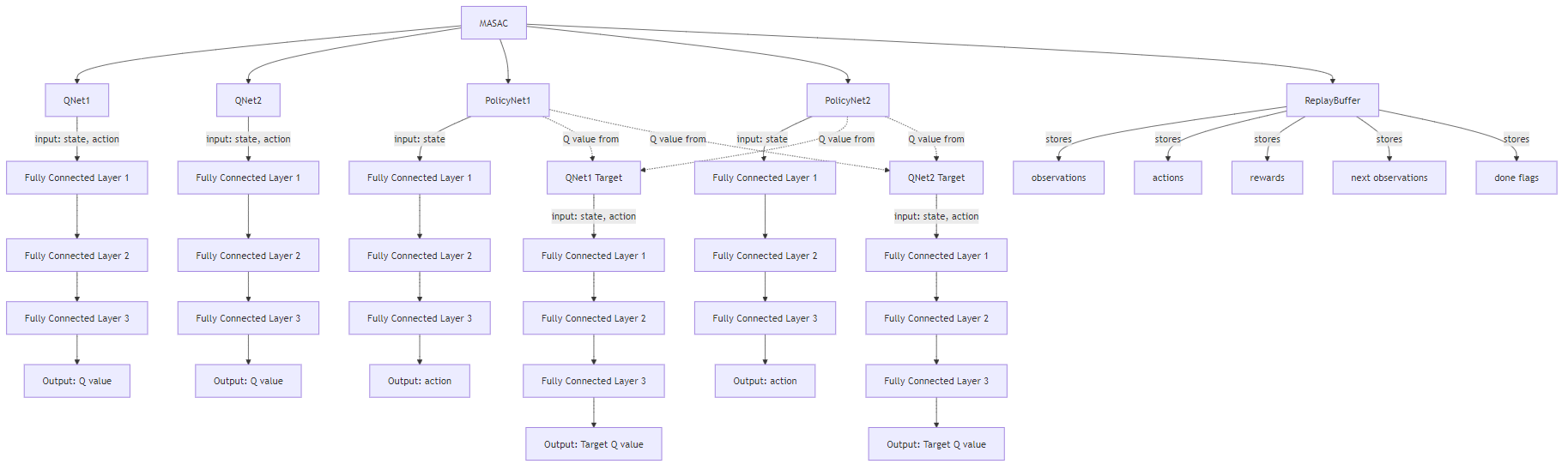


图2.7

如图2.7，MASAC（Multi-Agent Soft Actor-Critic）模型的网络架构由多个智能体共享的策略网络、Q值网络和价值网络组成。每个智能体的策略网络（PolicyNet）输出一个动作分布，用于指导智能体在环境中进行决策。Q值网络（QNet）评估每个智能体在当前状态下采取某个动作的价值，通常采用双Q网络来减少过估计偏差。价值网络（ValueNet）则估计状态的价值，帮助指导策略的更新。MASAC采用最大熵强化学习方法，旨在通过最大化预期回报和策略的熵，从而平衡探索与利用。每个智能体的训练通过使用经验回放池（ReplayBuffer）进行，智能体通过对其他智能体行为的反应进行学习。为了提高训练的稳定性，MASAC使用了目标网络（Target Networks）来软更新Q值网络和价值网络，从而实现更加平稳的策略优化。

**2.2.4 模型参数设置**

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 含义 | 示例值 | 来源 |
| num\_agents | |  | | --- | |  |  |  | | --- | | 智能体数量 | | 3 | args.num\_agents |
| hidden\_dim | |  | | --- | |  |  |  | | --- | | 隐藏层的维度大小 | | 64 | args.hidden\_dim |
| buffer\_size | 经验回放池的容量 | int(1e6) | args.buffer\_size |
| batch\_size | 每次训练采样的批量大小 | 1024 | args.batch\_size |
| device | 使用的计算设备（CPU 或 GPU） | 'cuda' | args.device |
| use\_orthogonal\_init | 是否使用正交初始化 | True | args.use\_orthogonal\_init |
| lr\_c | 评论器网络（Q 网络）的学习率 | 1e-3 | args.lr\_c |
| lr\_a | 策略网络（Policy 网络）的学习率 | 1e-3 | args.lr\_a |
| gamma | 折扣因子 | 0.95 | args.gamma |
| tau | 软更新参数 | 0.01 | |  | | --- | |  |  |  | | --- | |  |   args.tau |
| max\_action | 动作空间的最大值 | 1.0 | args.max\_action |
| episode\_num | 训练的总回合数 | 50000 | |  | | --- | |  |   args.episode\_num |
| warmup\_steps | 热身步骤数，用于随机动作采样以填充经验回放池 | 10000 | |  | | --- | |  |   args.warmup\_steps |
| add\_model | 是否加载已训练的模型 | False | args.add\_model |
| output\_dir | |  | | --- | |  |  |  | | --- | | 模型保存的目录路径 | | ./output | args.output\_dir |
| detector | 用于额外监测功能的参数 | 动态设置 | env.reset() 的返回值 |

表2.2 MADDPG 模型参数设置

MASAC相对于MADDPG模型，增加了以下参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 含义 | 示例值 | 来源 |
| alpha | 熵正则化系数，控制探索与开发的权衡 | 0.2 | args.alpha |
| alpha\_lr | 熵正则化系数的学习率 | 3e-4 | |  | | --- | |  |   args.alpha\_lr |
| |  | | --- | |  |   MIN\_LOG | 最小对数值，用于控制数值范围 | -20 | args.MIN\_LOG |
| MAX\_LOG | 最大对数值，用于控制数值范围 | 2 | args.MAX\_LOG |
| automatic\_entropy\_tuning | 是否启用自动熵调节 | True | args.automatic\_entropy\_tuning |
| policy\_update\_freq | |  | | --- | |  |  |  | | --- | | 策略更新的频率 | | 2 | args.policy\_update\_freq |

表2.3 MASAC模型新增参数

MATD3相对于MADDPG模型，增加了以下参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 含义 | 示例值 | 来源 |
| policy\_noise | 目标策略平滑噪声，用于缓解过度拟合 | 0.1 | args.policy\_noise |
| noise\_clip | 噪声裁剪值，用于控制噪声幅度 | 0.25 | args.noise\_clip |
| |  | | --- | |  |   policy\_update\_freq | 策略更新的频率 | 2 | args.policy\_update\_freq |

表2.4 MATD3模型新增参数

在MASAC (Multi-Agent Soft Actor-Critic)模型中，相比于 MADDPG，多出的参数主要与熵正则化和策略更新频率相关。alpha 控制了熵正则化系数，它决定了探索与开发之间的权衡。较高的值有助于探索更多的策略，而较低的值则倾向于选择更确定的动作。 alpha\_lr 是熵正则化系数的学习率，控制其更新的速度。MIN\_LOG 和 MAX\_LOG 分别限制了对数值的最小和最大范围，确保训练过程中数值不出现异常。automatic\_entropy\_tuning 允许自动调整熵系数，从而在训练过程中优化探索策略。通过设置 policy\_update\_freq，可以控制策略更新的频率，从而平衡策略更新的稳定性和训练速度。这些参数的引入有助于提升模型的探索能力，使得 MASAC 在复杂环境中能够更有效地找到优化策略。

相比之下， MATD3 引入的 policy\_noise 和 noise\_clip 参数主要用于目标策略平滑，用于在更新过程中添加噪声，从而减少估计误差和策略的震荡。这些参数对训练过程中策略稳定性和收敛速度有积极影响。policy\_update\_freq 也是 MATD3 的一个参数，它决定了策略更新的频率，合理设置该参数可以提升训练效率和稳定性，避免过于频繁的策略更新导致的不稳定性。

# 系统模拟

## 多智能体场景构建



图3.1

上述代码通过初始化多个环境组件来构建一个完整的仿真环境。它创建了地图矩阵、空间分辨率、偏移量等基本信息，并通过相应的类初始化了障碍物（线条、圆形、多边形）和源对象。机器人对象也被初始化，并存储在 `robot\_list` 中，同时还初始化了环境的可视化工具。最终，所有这些组件都被存储在 `self.components` 字典中，供后续仿真和操作使用。

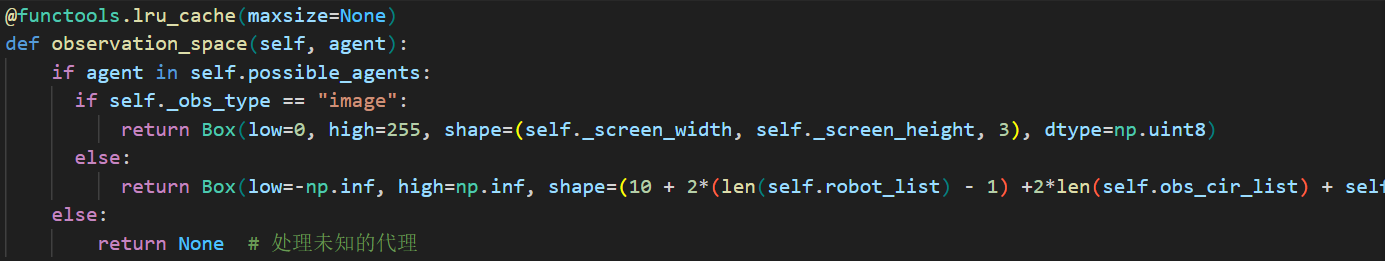


图3.2

通过定义方法 `observation\_space`，用于根据给定的代理（`agent`）返回该代理的观测空间。首先，它检查代理是否在 `possible\_agents` 中。如果代理存在且观测类型（`\_obs\_type`）为 "image"，则返回一个形状为 `(screen\_width, screen\_height, 3)` 的图像观测空间，像素值范围在 0 到 255 之间。如果观测类型不是 "image"，则返回一个浮点数类型的观测空间，形状包括 10 个常数值、与机器人数量相关的变量、与圆形障碍物数量相关的变量，以及机器人激光雷达数据的数量。若代理不在 `possible\_agents` 中，则返回 `None`。

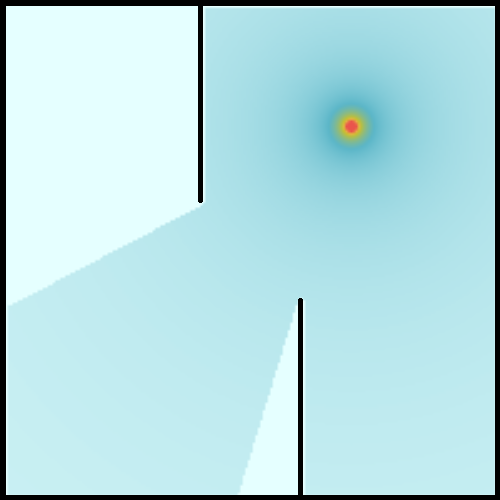


图3.3 构建的多智能体协作场景

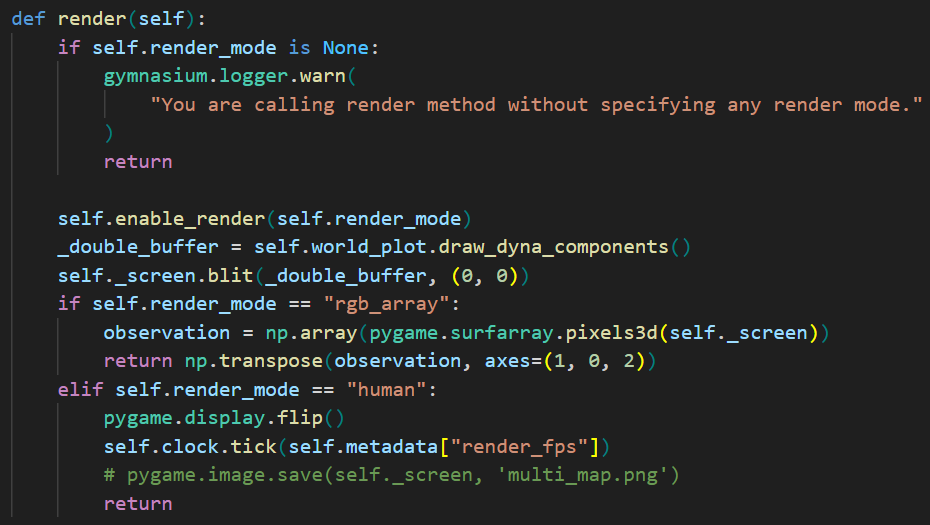


图3.4

通过 `render` 方法，根据指定的渲染模式渲染环境的可视化效果。首先，代码检查 `render\_mode` 是否为 `None`，如果是，则通过日志警告用户没有指定渲染模式并返回。接着，根据 `render\_mode` 来启用不同的渲染方式：如果渲染模式是 "rgb\_array"，则通过 `pygame.surfarray.pixels3d` 获取屏幕的像素数据，并将其转置为符合 `rgb\_array` 格式的 NumPy 数组；如果渲染模式是 "human"，则调用 `pygame.display.flip()` 刷新屏幕，并根据设置的帧率 `render\_fps` 控制渲染的速度。方法结束时，若渲染模式是 "human"，则没有返回值。

For each robot in robot\_list:

Call move\_forward with the action for the robot

Calculate the lidar range for the robot based on the components

Check if the robot is interacting with the source using source\_check

Check if the robot has collided using collision\_check

上述伪代码会遍历 `robot\_list` 中的每个机器人，并对每个机器人执行一系列操作。首先，根据给定的动作让机器人向前移动。然后，计算机器人的激光雷达（lidar）范围，这有助于确定机器人与环境中障碍物或其他物体的距离。接着，检查机器人是否与某个源（例如检测到信号或物品）发生交互，使用 `source\_check` 函数来实现。最后，检查机器人是否与环境中的任何障碍物发生碰撞，使用 `collision\_check` 函数来判断。每一步都是更新机器人状态并确保机器人在仿真环境中正常行为的关键操作。

函数 cal\_reward(robot\_list, source\_list, beta, theta, mu, gamma):

创建一个空字典 rewards 用于存储每个机器人的奖励

创建一个空字典 infos 用于存储每个机器人的额外信息

初始化全局奖励 global\_reward 为 0.0

如果 local\_ratio 不为 None:

遍历 source\_list 中的每个源 s:

对于每个机器人 a，计算其与源的距离 dists

计算所有机器人的距离和，将其累加到 global\_reward 中（对距离取负值）

创建一个空字典 agent\_reward 用于存储每个机器人的奖励

计算机器人找到源的数量 find\_num

遍历 robot\_list 中的每个机器人 robot:

计算该机器人与源之间的检测值 detector

如果 detector > max\_detector:

计算正向奖励 r\_d 为 exp(detector - max\_detector)

否则，计算负向奖励 r\_d 为 -mu \* max(max\_detector - detector, 0)

更新 max\_detector 为当前的最大值

如果机器人发现了源 (source\_flag 为 True):

增加一个额外奖励 10 \* exp(find\_num / \_robot\_number)

计算该机器人与最近障碍物的距离 min\_dis

如果 min\_dis < robot.radius \* 1.5:

计算碰撞奖励 r\_c 为 -beta \* max(lidar.range\_max - min\_dis, 0)

否则，设置碰撞奖励 r\_c 为 0

计算该机器人的总奖励 agent\_reward[robot.id] 为 r\_d + r\_c

如果 local\_ratio 不为 None:

计算机器人最终奖励 reward 为 global\_reward \* (1 - local\_ratio) + agent\_reward \* local\_ratio

否则，设置 reward 为 agent\_reward[robot.id]

将机器人的最终奖励存储到 rewards 字典中，键为 "agent\_{robot.id}"

将机器人的 source\_flag 存储到 infos 字典中，键为 "agent\_{robot.id}"

检查是否存在环境终止条件 (机器人发生碰撞):

如果环境终止，调整所有机器人的奖励，减少一个因时间步数影响的惩罚项

返回 rewards, infos, env\_termination

上述伪代码的目的是计算每个机器人在环境中的奖励，并根据不同的条件进行调整。首先，它根据是否有局部奖励比率（`local\_ratio`）来决定如何计算全局奖励。如果 `local\_ratio` 不为 `None`，它会遍历源列表，计算所有机器人与源的距离，将距离的和加入全局奖励中。然后，代码遍历所有机器人，计算每个机器人的奖励。对于每个机器人，首先计算其与源之间的检测值，并根据该值和最大检测值来决定是正向奖励还是负向奖励。机器人如果发现了源，还会获得额外的奖励。接着，计算机器人与障碍物的距离，如果距离过近，会给予碰撞惩罚。最后，根据局部奖励比率（如果有）计算最终奖励，并将每个机器人的奖励存储在 `rewards` 字典中。`infos` 字典记录机器人是否发现源的信息。最后，如果机器人发生碰撞（即环境终止），会对所有机器人的奖励进行调整。最终函数返回的是每个机器人的奖励字典、信息字典和环境是否终止的标志。

## 多智能体深度强化学习算法开发

**3.2.1 Q值网络与策略网络**

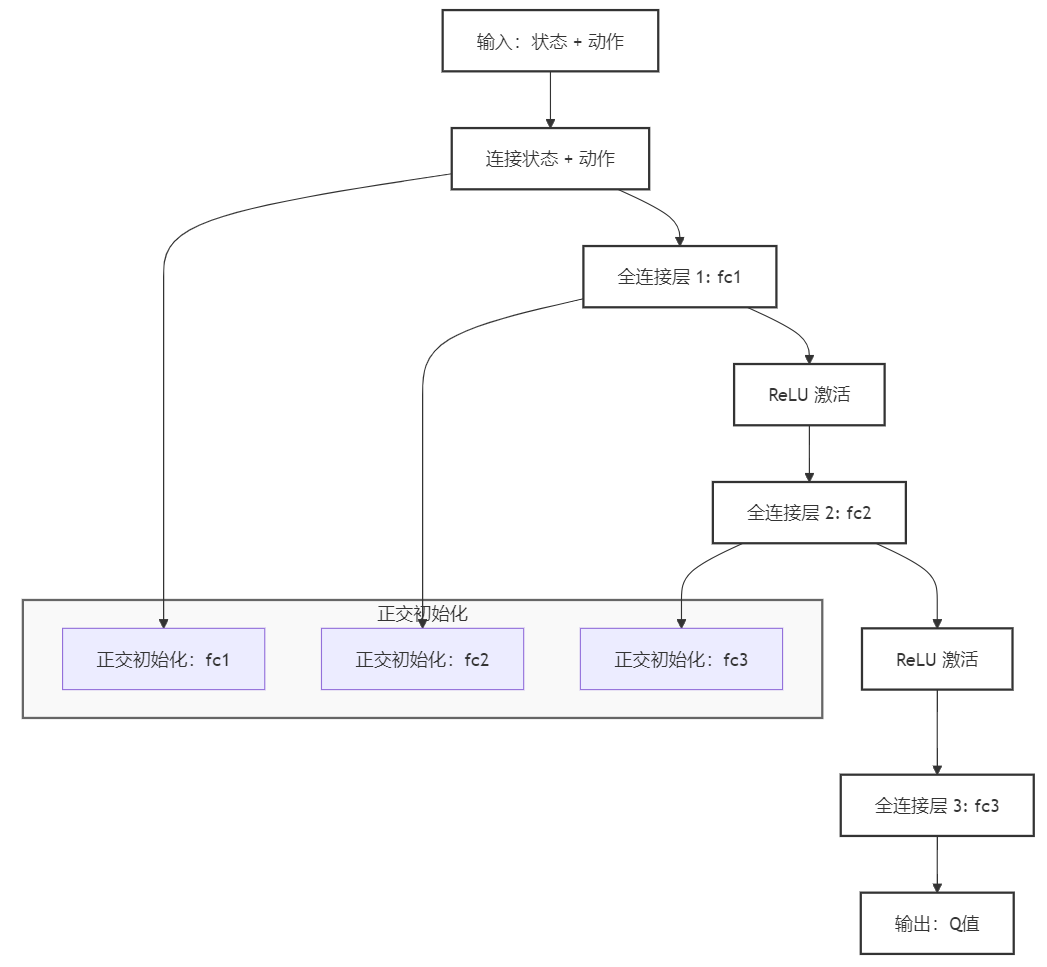


图3.5 QNet架构图

QNet作为强化学习的神经网络模型，其核心作用是估计给定状态（state）和动作（action）组合的 Q 值。网络由三层全连接层（`fc1`, `fc2`, `fc3`）构成，输入是通过拼接状态和动作向量获得的 `sa`，其中 `dim\_state` 和 `dim\_action` 分别表示状态和动作的维度，`num\_agents` 表示代理的数量。第一层将输入大小映射到隐层维度 `hidden\_dim`，第二层通过 ReLU 激活函数进行非线性映射，最后通过第三层输出一个标量表示 Q 值。

该网络的特点是使用了正交初始化（`orthogonal\_init`），在每层权重初始化时帮助网络更好地收敛，尤其适用于深度强化学习中。`forward` 方法通过拼接状态和动作，并通过多层全连接层处理，最终输出预测的 Q 值，这一 Q 值代表了某个特定状态-动作组合的预期回报，用于智能体的策略更新。

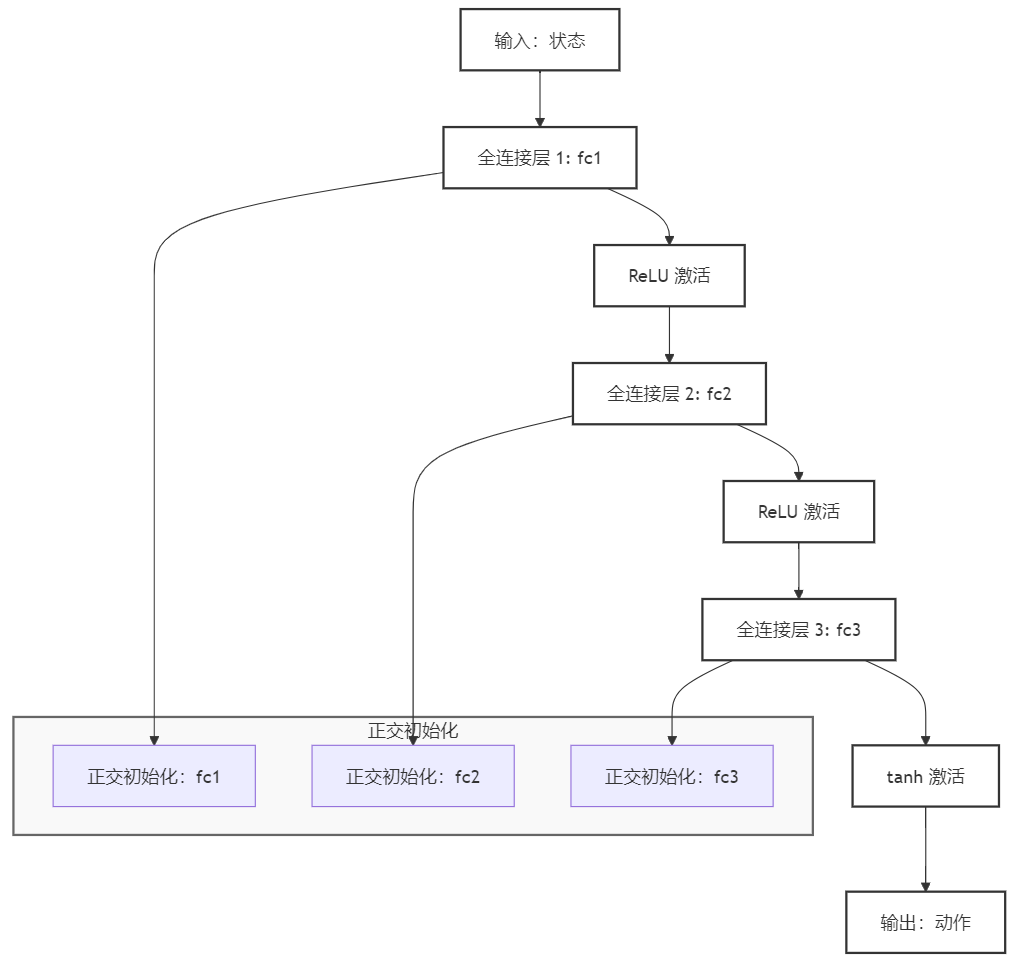


图3.6 PolicyNet架构图

PolicyNet作为深度强化学习中的策略网络，用于从给定的状态（`state`）生成对应的动作（`action`）。它由三个全连接层（`fc1`、`fc2` 和 `fc3`）组成，前两层使用 `ReLU` 激活函数，最后一层通过 `tanh` 激活函数输出动作，并且动作的大小会乘以 `max\_action` 来控制动作的范围。通过这种方式，网络可以根据输入状态生成一个合适的动作输出，适用于强化学习中的策略优化任务。

此外，网络还使用了正交初始化（`orthogonal\_init`），有助于提高训练过程中的稳定性，特别是在深层网络的训练中。正交初始化能够保持网络各层的权重矩阵的稳定性，从而促进梯度的传递，避免梯度消失或爆炸的问题。

**3.2.2 回放缓冲区**

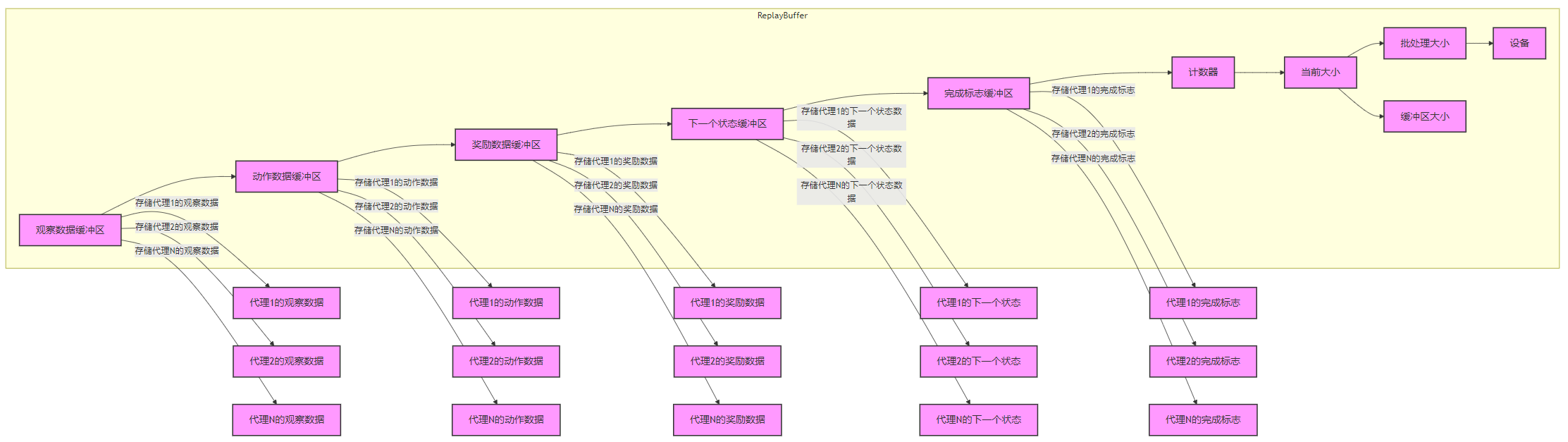


图3.7 RelayBuffer架构图

回放缓冲区用于多代理强化学习环境中的数据存储和采样。回放缓冲区的作用是存储每个代理的状态、动作、奖励、下一个状态和完成标志等信息，在强化学习训练过程中随机采样这些数据来更新模型，从而避免训练过程中数据的相关性问题。回放缓冲区使用字典来存储每个代理的历史数据，每个代理的数据（状态、动作、奖励等）被保存在对应的缓冲区中。`store\_transition`方法用于将当前的交互数据存储到缓冲区中，而`sample`方法则用于从缓冲区中随机抽取一批样本（批处理），以供训练模型时使用。

具体来说，这个回放缓冲区适用于多代理环境，其中每个代理的状态、动作和奖励等数据分别存储在对应的缓冲区中，通过设置缓冲区大小（`buffer\_size`）来限制存储的数据量，并通过批量采样（`batch\_size`）来训练模型。回放缓冲区的核心目的是通过存储历史经验并进行随机采样，来提高训练的稳定性和效率。

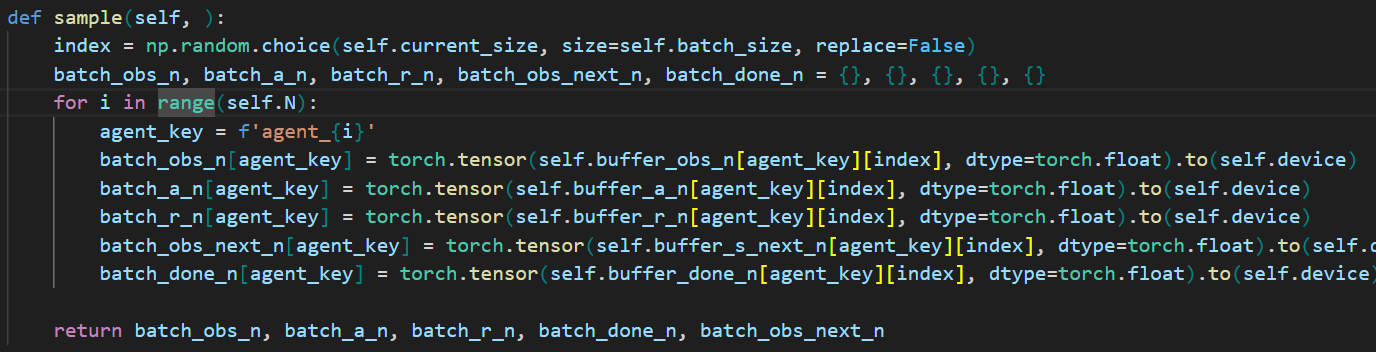


图3.8

采样过程通过`sample`方法实现，从回放缓冲区中随机选择一批数据用于训练。首先，`np.random.choice`函数根据当前缓冲区的大小（`current\_size`）和批量大小（`batch\_size`），随机选择出一组索引（`index`）。然后，针对每个代理（`agent\_0`, `agent\_1`等），从对应的缓冲区中根据这些索引提取相应的数据（状态、动作、奖励、下一个状态和完成标志），并将这些数据转换为`torch.tensor`类型，最终返回这些批量数据供模型训练使用。这一过程保证了从回放缓冲区中抽取的数据是随机的，从而避免了数据的相关性，提高了训练的稳定性和效果。

3.2.3 MASAC算法：高斯策略网络

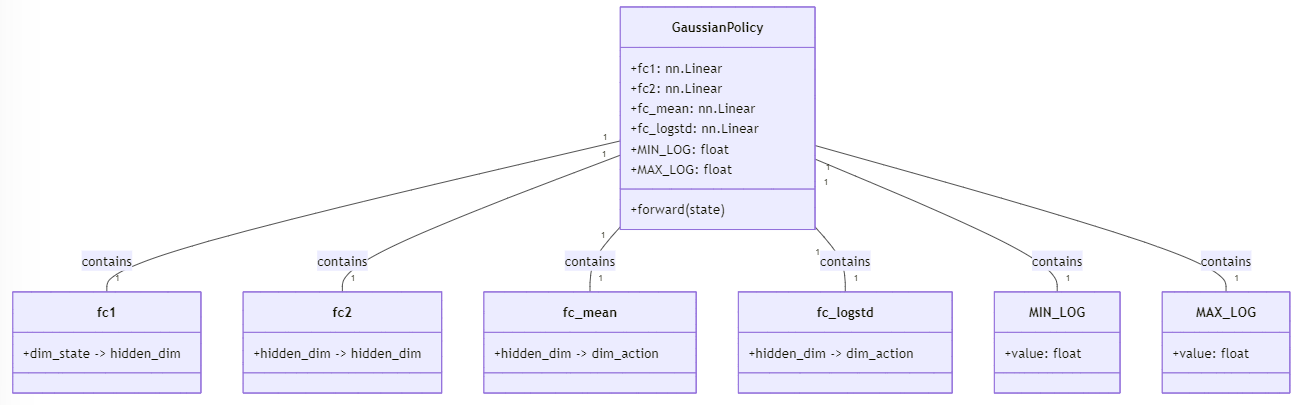


图3.9

高斯策略网络用于强化学习中的策略梯度方法。该网络通过多层全连接层（`fc1` 和 `fc2`）将状态输入（`state`）映射到一个隐藏空间，之后通过两层全连接层（`fc\_mean` 和 `fc\_logstd`）分别计算输出的均值（`mean`）和对数标准差（`log\_std`）。均值表示动作的期望值，而对数标准差则用于确定动作的随机性（标准差）。输出的 `mean` 和 `log\_std` 可以用来从高斯分布中采样动作，进一步执行在环境中的动作。

该高斯策略网络的作用是生成一个基于当前状态的高斯分布，用于选择一个动作。`log\_std` 被限制在一个预定义的范围内（`MIN\_LOG` 和 `MAX\_LOG`），以确保稳定性。该网络常用于连续动作空间的强化学习任务，尤其是在策略优化（Policy Gradient）算法中，帮助智能体根据当前策略做出决策。

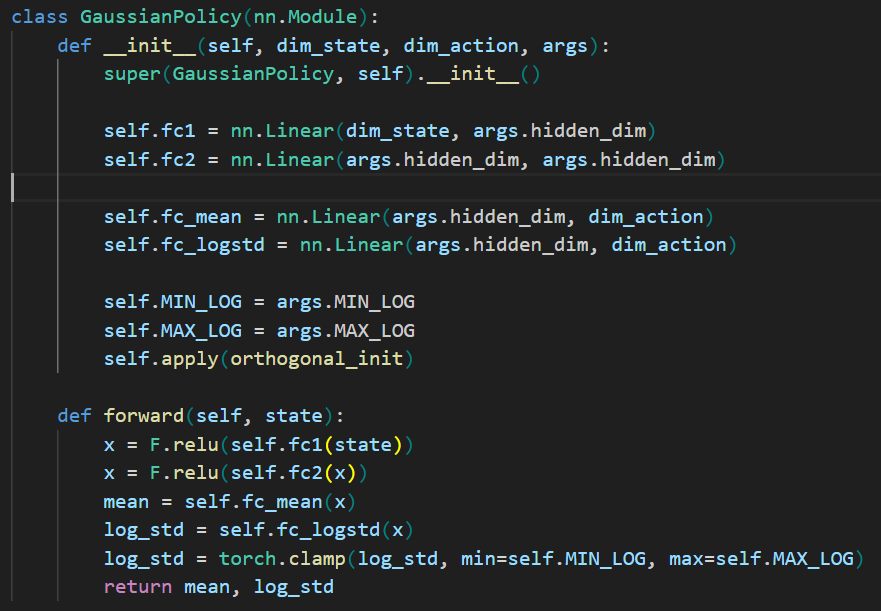


图3.9 高斯策略网络实现

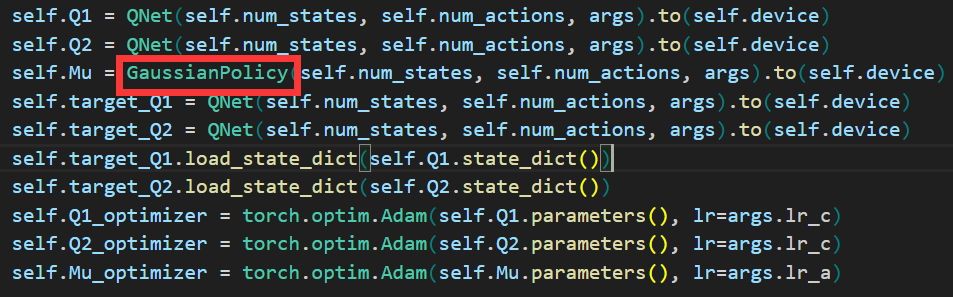


图3.10 高斯策略网络参与MASAC的构建

在MASAC模型中，高斯策略网络扮演着重要的角色，它是每个智能体的策略网络。具体来说，该网络接受当前状态作为输入，通过两层全连接层计算每个动作的均值和对数标准差。这两个输出描述了动作的概率分布，从中采样生成连续动作，用于与环境进行交互。高斯策略使得模型能够处理连续动作空间，能够输出平滑的动作而不仅限于离散的选择。

引入高斯策略的主要作用在于，首先它能够有效地建模连续动作空间，这是强化学习中处理连续控制问题的常用方法。通过对动作进行概率建模，智能体可以在策略更新时有更多的自由度来探索环境，而不是只依赖于确定性的动作选择。此外，在软演员-评论家（SAC）框架中，高斯策略通过熵正则化来鼓励策略的随机性，确保智能体能够保持足够的探索性，避免陷入局部最优解。策略的更新则通过最大化回报和调整策略的熵来优化，从而提高模型的稳定性和性能。

3.2.4 三种模型定义

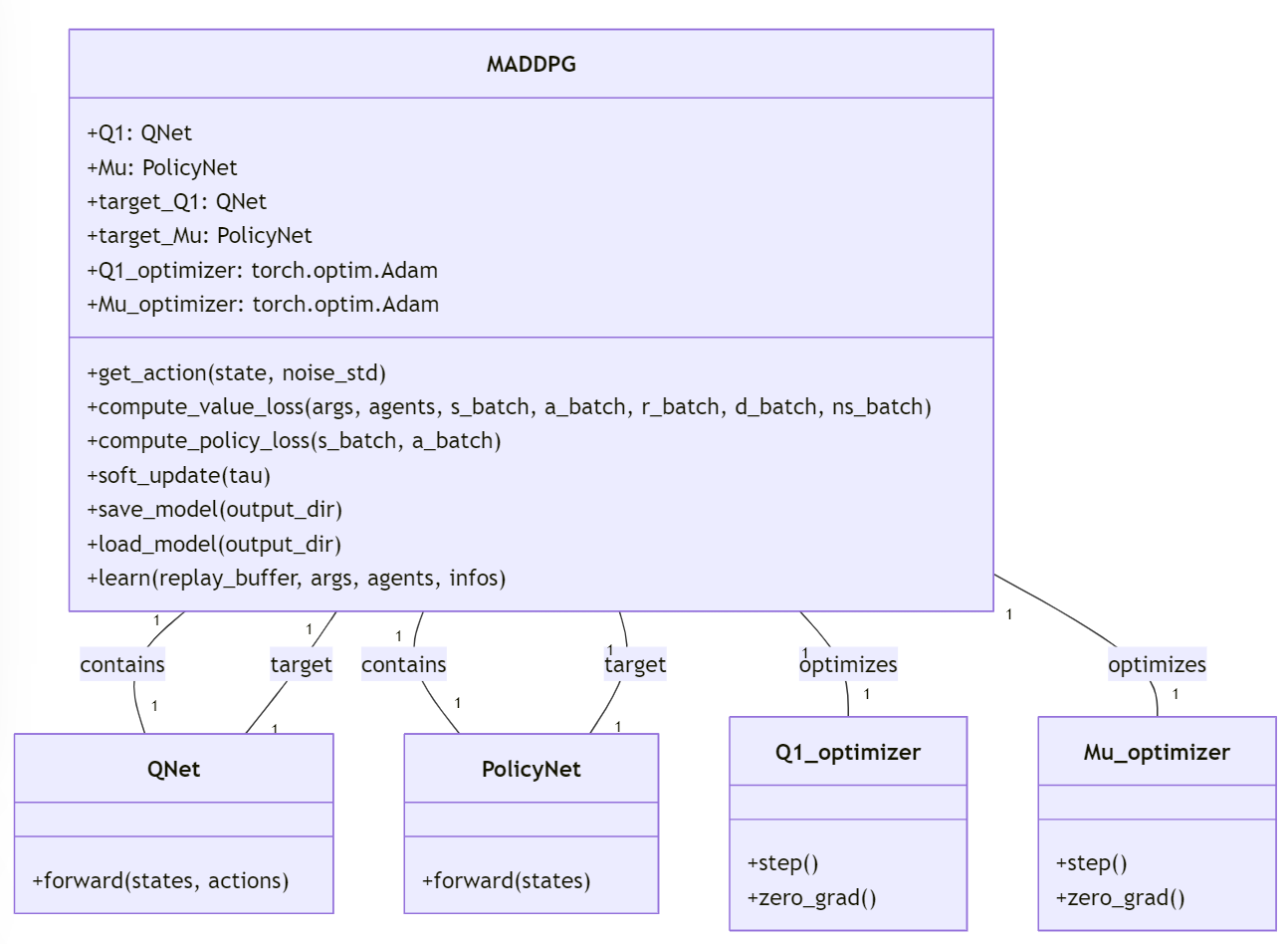


图3.11 MADDPG模型结构

MADDPG模型结合了深度确定性策略梯度（DDPG）和集中训练，分散执行（CTDE）的框架。每个智能体的结构包括一个Q网络（`QNet`）和一个策略网络（`PolicyNet`），其中 Q 网络用于评估给定状态和动作的价值，而策略网络负责生成基于当前状态的动作。模型采用了目标网络（`target\_Q1` 和 `target\_Mu`）来稳定训练过程，这些目标网络通过软更新（`soft\_update`）与主网络进行同步。

在训练过程中，智能体使用来自回放缓冲区的数据，通过优化 Q 网络来最小化价值损失（`value\_loss1`），并通过优化策略网络来最大化 Q 值（`policy\_loss`）。具体来说，智能体的策略是基于当前的状态输出一个确定性动作，并加入噪声以促进探索。该模型的优势在于通过集中的 Q 网络进行训练和更新，同时保持每个智能体的策略独立，使得多个智能体可以在多智能体环境中高效地协同工作。

相较于MATD3，MADDPG的劣势在于其没有MATD3中引入的关键技术：目标网络的平滑更新和双重Q学习。MADDPG 采用了标准的DDPG结构，其中每个智能体使用自己的确定性策略和Q函数，而在多智能体环境中，这可能导致智能体之间的行为相互依赖且容易引发不稳定或过估计问题。MATD3通过引入目标网络的平滑更新和双重Q学习，有效减轻了这些问题，减少了高方差的更新，提升了训练的稳定性和收敛速度。此外，MATD3 还使用了噪声策略来促进探索，并通过延迟更新来进一步增强训练的稳定性，这些都使得 MATD3 在多智能体强化学习任务中通常比 MADDPG 更加有效和稳定。

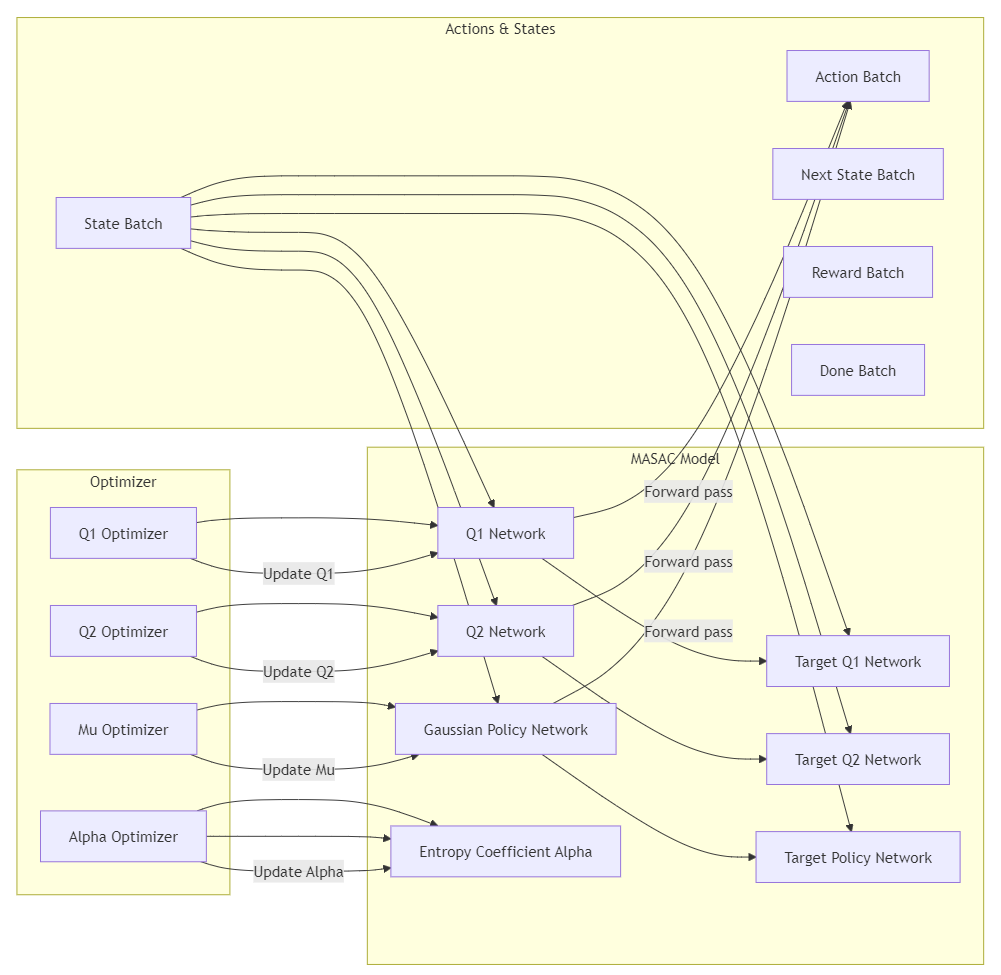


图3.12 MASAC模型结构

MASAC结合了软策略优化和深度强化学习的优势。该模型的核心结构包括三个主要组件：Q网络（Q1, Q2）、策略网络（Mu）和目标网络（target\_Q1, target\_Q2）。Q网络用于估计智能体的动作价值，策略网络通过高斯分布生成动作，并利用重参数化技巧生成连续的动作。为了确保探索性，MASAC采用了自动化的熵调节机制来增强多样性和稳定性。它还引入了目标Q网络以提高训练的稳定性和收敛速度，并使用软更新来逐步更新目标网络的参数。

与MATD3相比，MASAC的主要劣势在于它的目标值估计和策略优化之间的关系较弱。虽然MASAC利用了自动熵调节来优化策略，但这种方法将导致智能体在训练过程中依赖较高的噪声策略，而MATD3则通过双Q学习和延迟更新策略来平衡动作值的估计，减少了策略更新的过度估计问题，提升了训练的稳定性。

另外，MASAC模型的训练效率较低，因为它的策略优化和Q值优化是同步进行的，而MATD3采用了延迟更新策略，即在更新策略网络之前先更新Q网络，这样的延迟更新可以减少策略和价值网络之间的相互影响，避免训练过程中策略网络和Q网络之间的干扰。MASAC没有MATD3中的这种延迟更新机制，这将导致训练过程中更难以稳定收敛，特别是在高维动作空间和复杂的多智能体环境中。

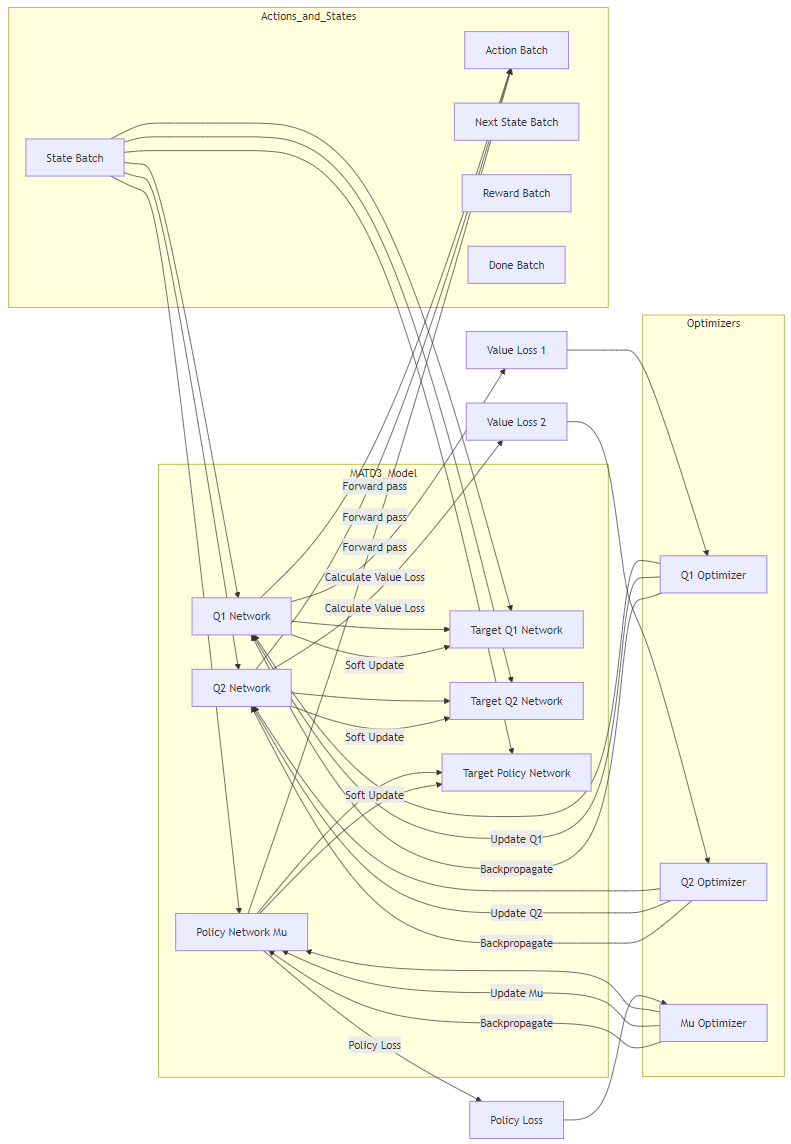


图3.13 MATD3模型结构

MATD3基于深度确定性策略梯度扩展到多智能体环境,它结合了双Q网络（Q1和Q2）和延迟更新的策略优化机制。该模型的核心架构包括Q网络用于值估计，策略网络生成连续的动作。为了稳定训练，MATD3引入了延迟更新机制，即在更新策略网络之前，先更新Q网络。这可以避免在值函数估计过度时导致策略更新不稳定。MATD3还通过目标网络（target\_Q1, target\_Q2, target\_Mu）来提高训练的稳定性，确保在多个智能体共同训练时，策略的更新不会受到其他智能体的不稳定动作的影响。

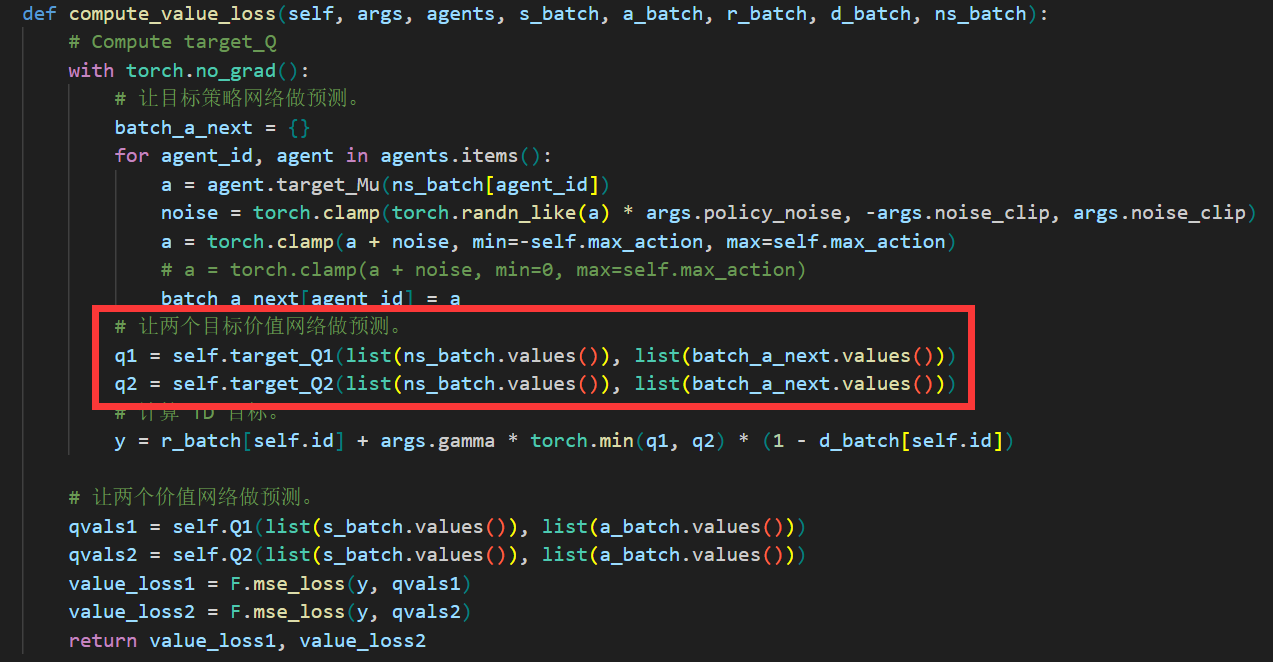


图3.14 MATD3模型的双Q网络结构

与MADDPG相比，MATD3的优势在于它的双Q网络结构，这减少了过估计问题，即在价值网络更新时可能对未来的奖励过于乐观，从而导致不稳定的策略。通过使用两个独立的Q网络并采取最小化Q值的方式，MATD3有效地减少了这种过估计，提高了训练的稳定性。此外，MATD3还采用了延迟策略更新，即每隔几个训练步骤才更新一次策略网络，这样可以在更充分地估计值函数后再进行策略优化，从而提高了策略的质量和收敛速度。

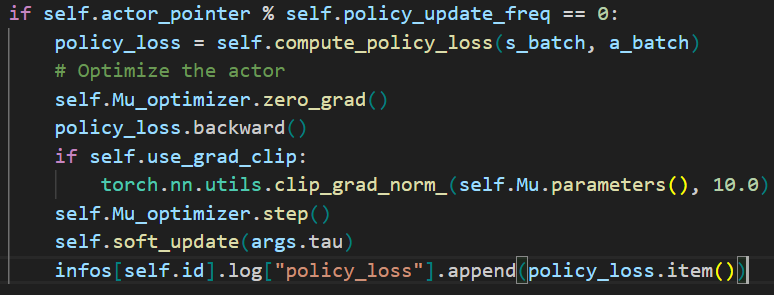


图3.15 MATD3模型的延迟策略更新

与MASAC相比，MATD3的优势在于它没有依赖于自动熵调节来调整探索性，而是通过双Q网络和延迟策略更新来确保稳定的训练过程。MASAC的自动熵调节虽然有助于探索，但也可能导致智能体在训练过程中依赖过高的噪声策略，这会增加训练的不稳定性。MATD3通过延迟更新的方式，减少了策略优化和价值函数之间的相互影响，避免了模型训练时不必要的噪声和干扰。此外，MATD3通过在计算目标Q值时加入噪声，有效地增强了探索性，同时保持了训练的稳定性。

## 模型训练

3.3.1 模型训练过程

函数 train(args, env, agents, replay\_buffer, writer):

如果 args.add\_model 为 True:

对于每个 agent 在 agents 中:

从 output\_dir 加载该 agent 的模型

初始化 step = 0

初始化 max\_reward = 300

初始化 sum\_reward = 0

初始化 episode\_data = 空列表

初始化 infos = 为每个 agent 创建一个 INFO 对象

对于 episode 从 1 到 args.episode\_num:

重置环境，获取初始 state 和 detector

在每一回合中执行：

对于每个 step:

增加 step 计数

如果 step 小于 warmup\_steps:

对于每个 agent:

随机采样动作

否则:

对于每个 agent:

根据当前状态选择动作，并加上噪声

执行动作并获取下一个状态、奖励、终止标志、截断标志

为每个 agent 计算 done 状态（终止或截断）

将当前经历（state, action, reward, next\_state, done）存入 replay\_buffer

对于每个 agent:

更新 infos，记录 done 状态、奖励和 detector 信息

更新当前状态为 next\_state

如果 step 大于等于 warmup\_steps:

对于每个 agent:

让该 agent 从 replay\_buffer 中学习（调用 learn 方法）

每隔 20 个回合打印训练信息：

将 sum\_reward 重置为 0

获取当前时间

对于每个 agent:

计算最近 20 个回合的平均奖励、价值损失和策略损失

将这些信息存入 episode\_data 列表

累加奖励到 sum\_reward

重置 agent 的 infos

记录当前 sum\_reward 到 TensorBoard

记录每个 agent 的价值损失和策略损失到 TensorBoard

每隔 100 个回合保存 episode\_data 到 CSV 文件

清空 episode\_data

如果当前 sum\_reward 大于 max\_reward:

更新 max\_reward 为 sum\_reward

保存每个 agent 的模型到 output\_dir

如上述伪代码所示，在每一回合中，智能体从环境中采集状态并根据当前策略选择动作，执行动作后通过环境返回的奖励和状态更新策略。训练过程分为热身阶段（使用随机动作）和学习阶段（从经验池中采样并更新策略）。每隔一段时间，训练信息（如奖励、损失）会被记录并输出，同时每 100 个回合将训练数据保存到 CSV 文件中。每 20 个回合会输出当前的训练进度和性能指标，如平均奖励、价值损失和策略损失。当总奖励超过历史最高记录时，智能体的模型会被保存。

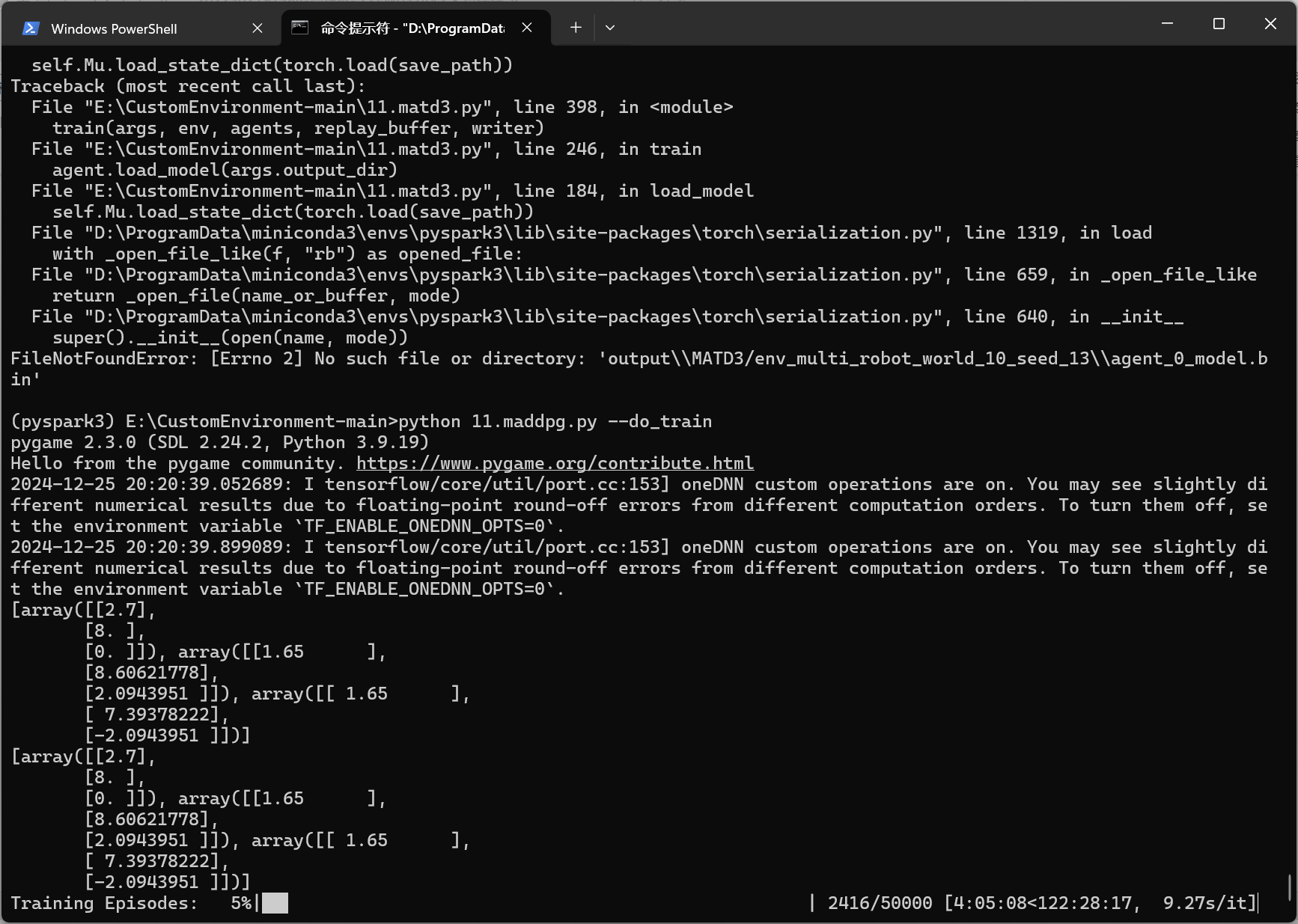


图3.16 MADDPG模型训练过程

在部署RTX 2050 GPU的机器上，训练约20.85小时后，得到MADDPG在常规场景下的模型；另需约1.37天后，得到MADDPG在复杂场景下的模型。

3.3.2 三种模型训练结果(损失曲线、奖励曲线)对比分析

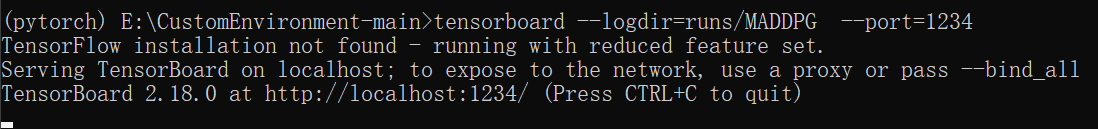


图3.17 通过tensorboard查看损失曲线、奖励曲线

1. 策略损失曲线

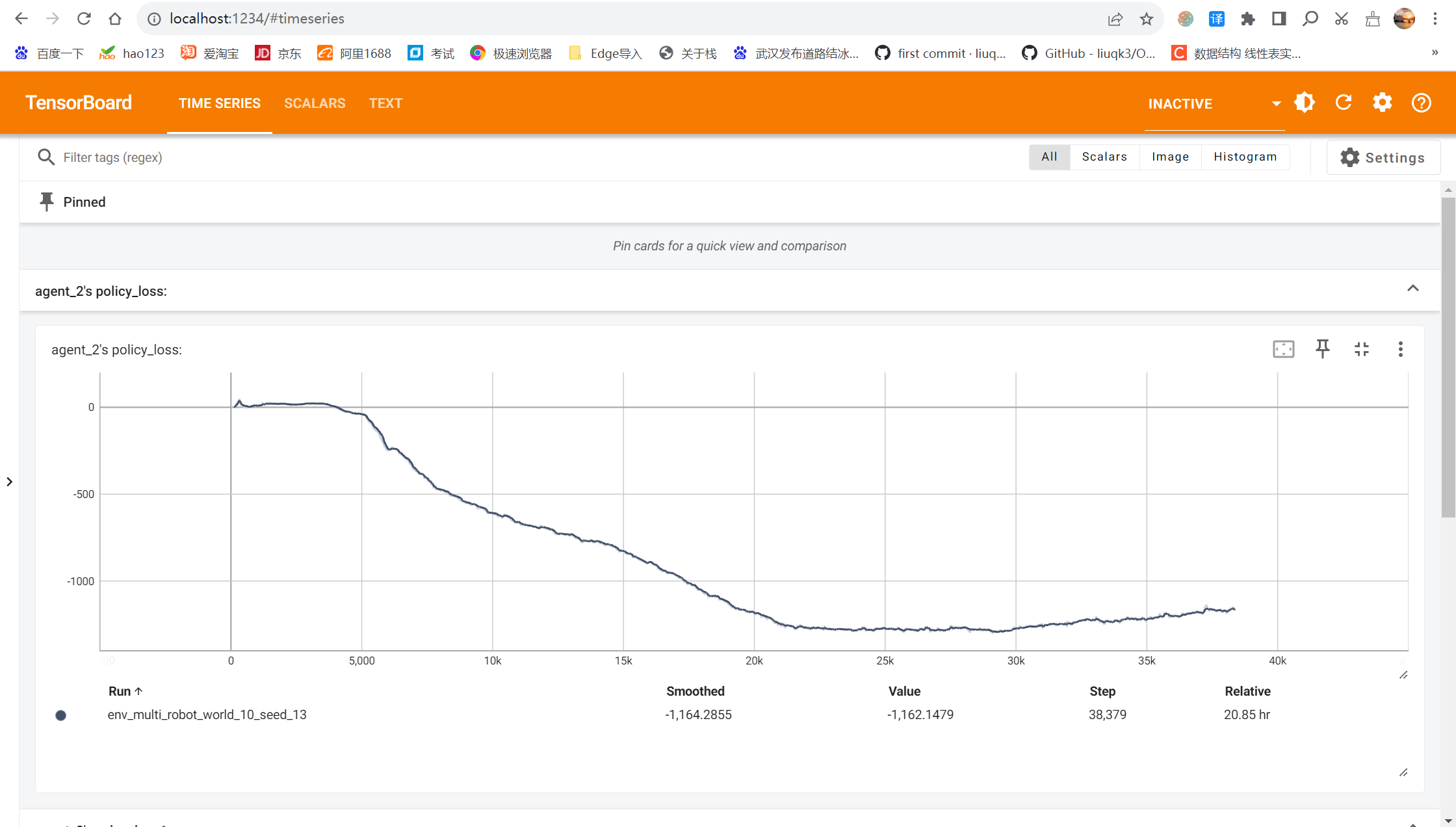


图3.18 MADDPG模型的策略损失曲线



图3.19 MATD3模型的策略损失曲线

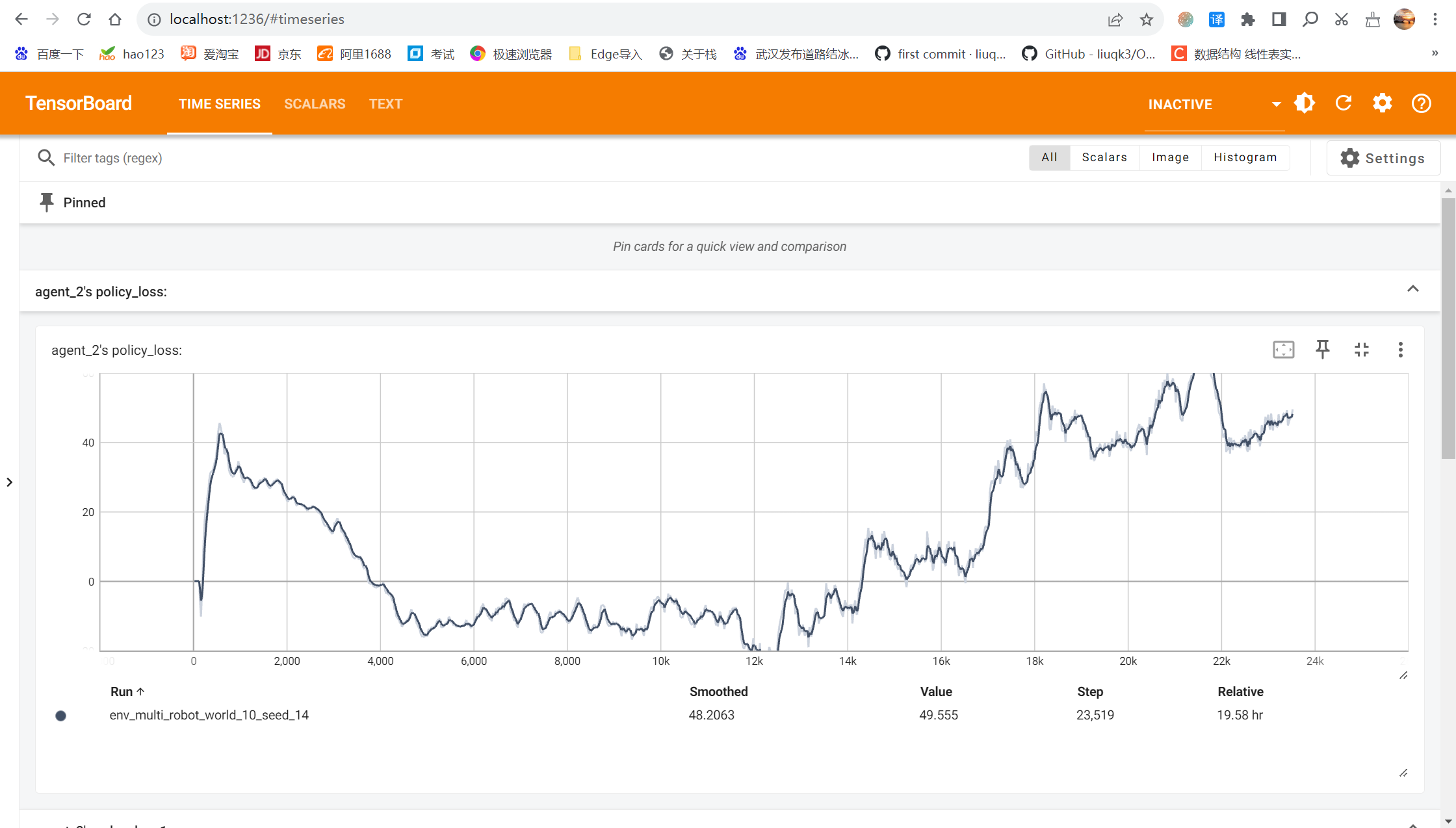


图3.20 MASAC模型的策略损失曲线

在多智能体强化学习模型的策略损失方面，MADDPG模型的表现最为出色，其策略损失最终收敛于-1162，平滑值为-1164，这一较低的损失值表明MADDPG在训练过程中能够有效地优化其策略，反映出较好的学习效果。MADDPG通过集中训练一个评论家网络来评估所有智能体的动作，这有助于它在策略优化上达到较高的效率。

MATD3的策略损失最终收敛于-128，平滑值为-129，虽然在数值上高于MADDPG，但仍然显示出较好的策略优化能力。MASAC模型的策略损失最终收敛于49，平滑值为48，是三个模型中损失值最高的，这意味着MASAC在策略学习过程中遇到了更多的挑战，其策略优化的效率相对较低。MASAC通过最大化熵来鼓励探索，这导致其在策略优化上需要更多的调整以提高效率。

1. 价值损失曲线

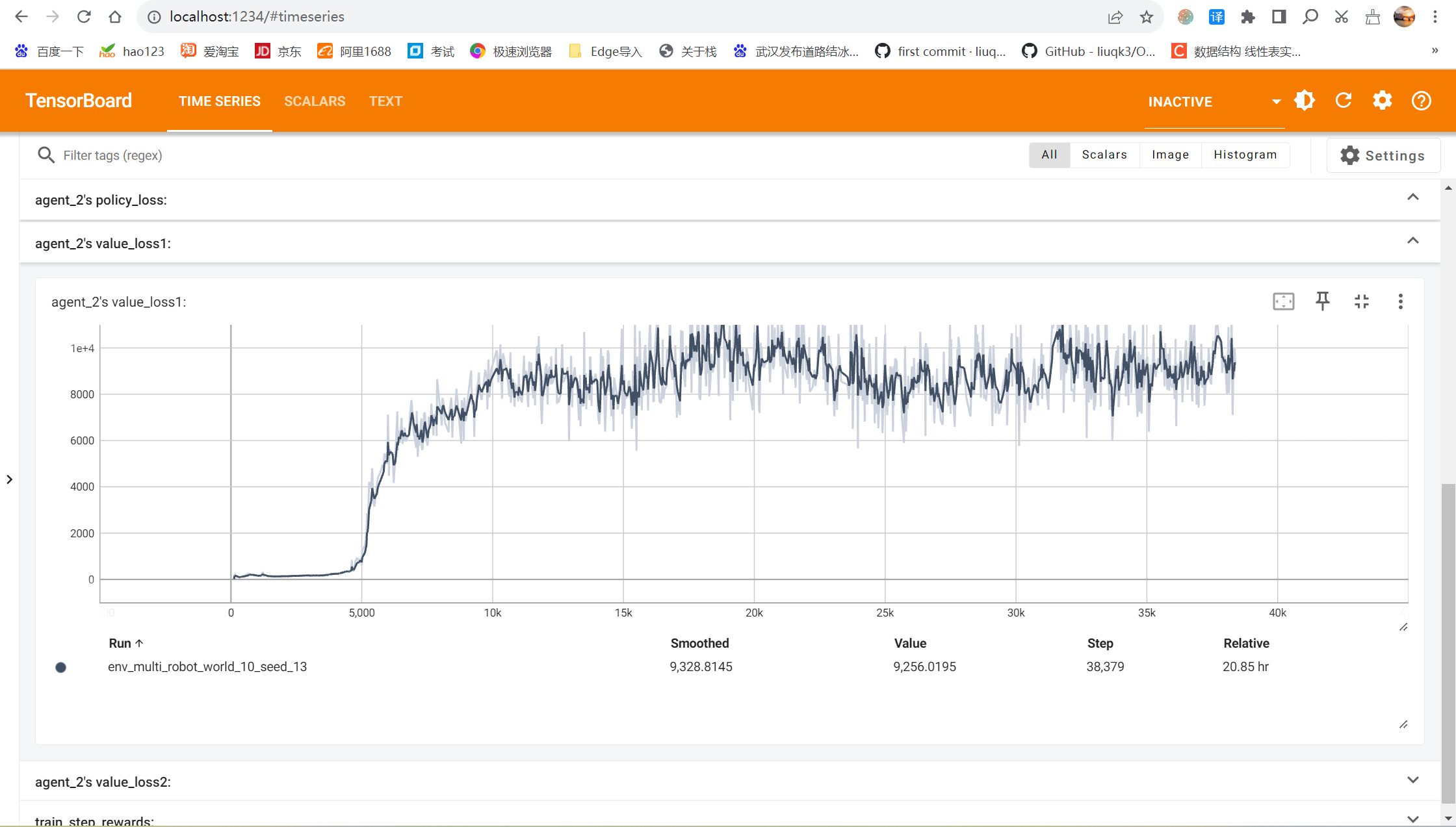


图3.21 MADDPG模型的价值损失曲线

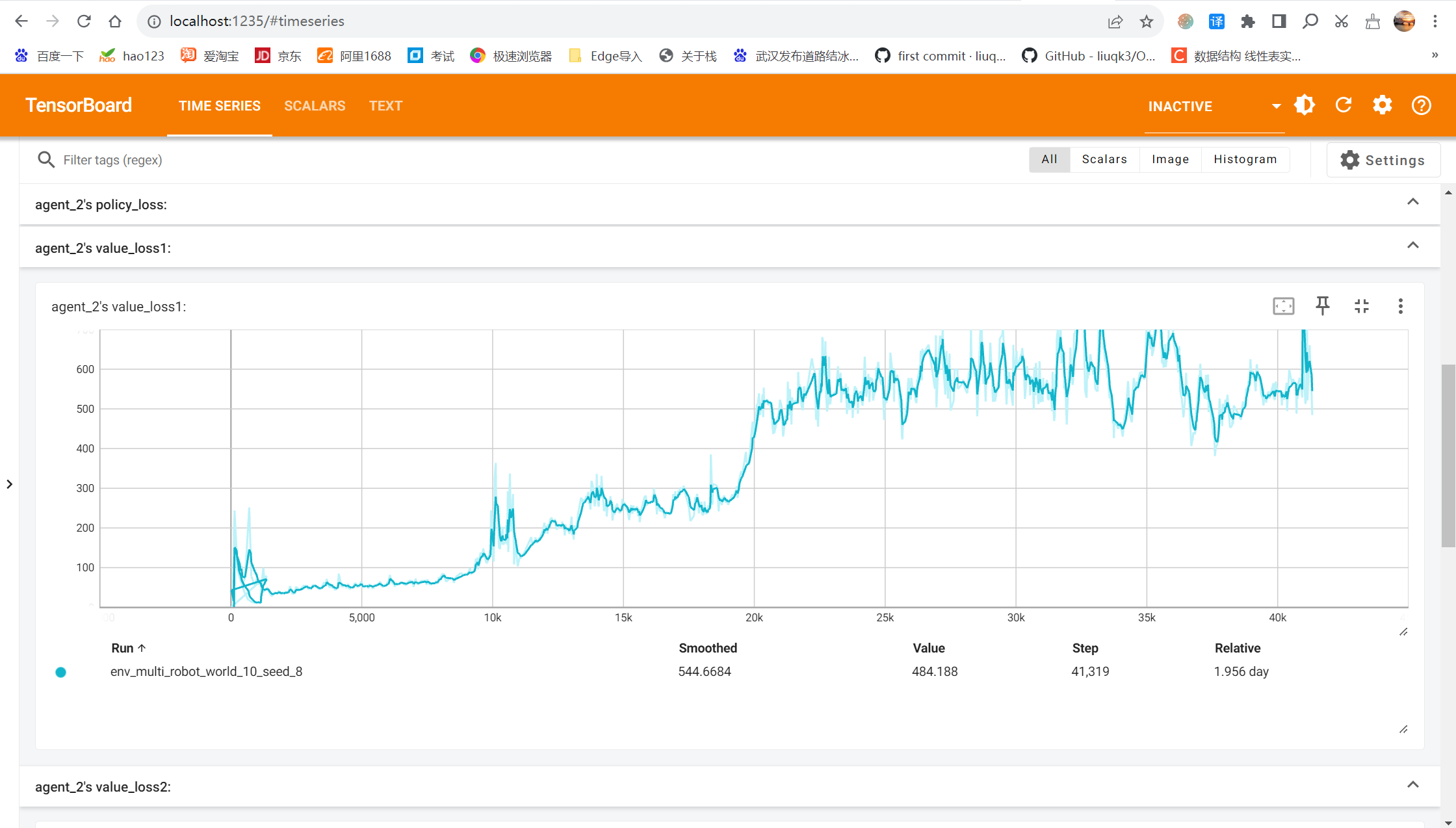


图3.22 MATD3模型的价值损失曲线



图3.23 MASAC模型的价值损失曲线

在多智能体强化学习模型的价值损失表现上，MATD3模型以最终收敛于484的价值损失和544的平滑值领先，显示出其在优化价值函数方面的显著优势。这得益于MATD3算法设计中对智能体间相互作用的有效处理，使其在评估和学习过程中更为高效。

相比之下，MADDPG模型的价值损失最终停留在9256，而MASAC模型则为2298，两者都高于MATD3。MADDPG的高价值损失指示其在策略评估和更新过程中存在一定的局限性，尤其是在处理多智能体环境中的复杂交互时。MASAC虽然在价值损失上优于MADDPG，但仍然有改进空间，其鼓励探索的策略在初期有助于发现新策略，但在后期需要更多调整以优化价值评估。这三种模型在价值损失上的表现差异，反映了它们在多智能体学习环境中的不同适应性和优化效率。

1. 奖励曲线

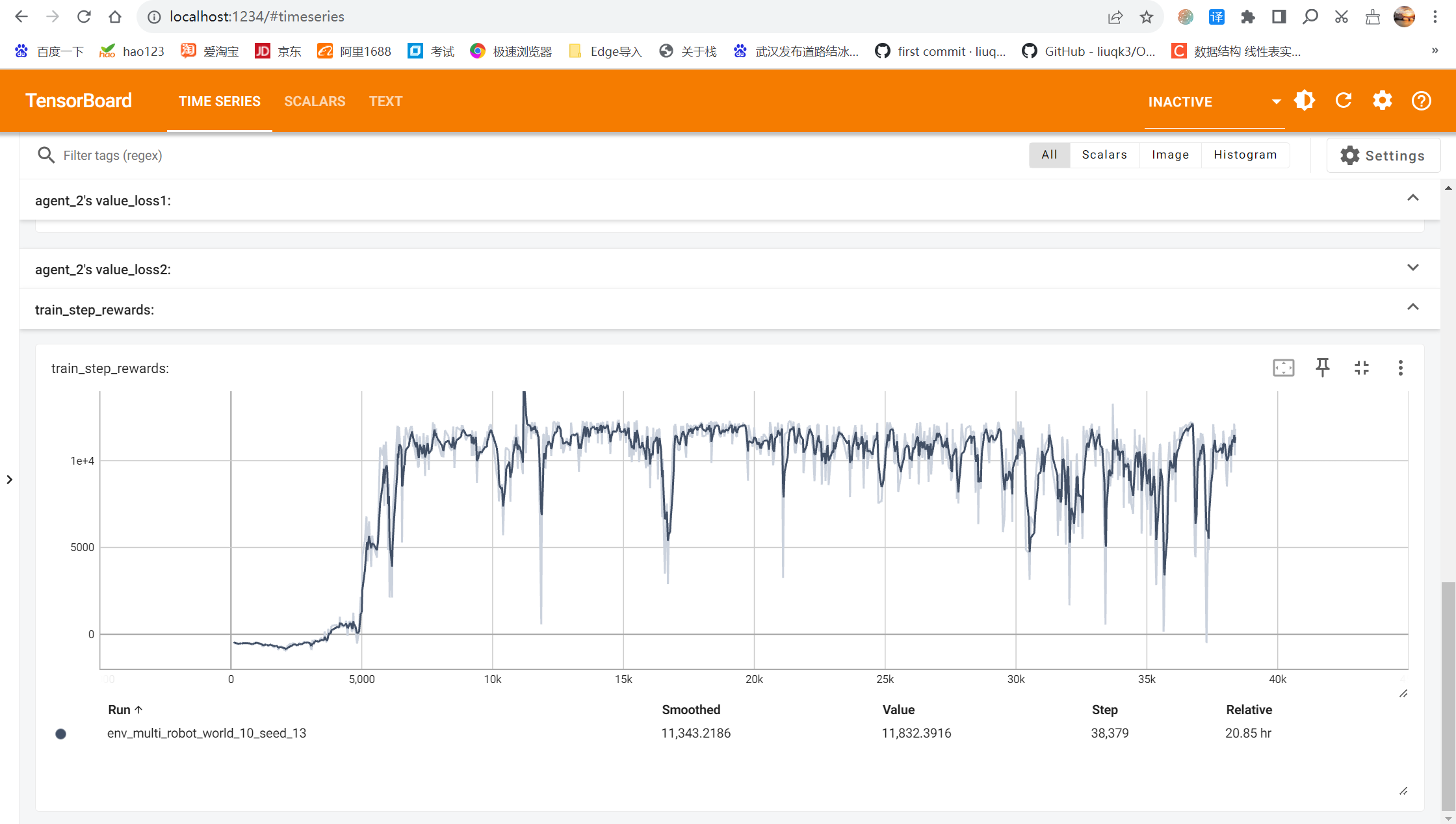


图3.24 MADDPG模型的奖励曲线

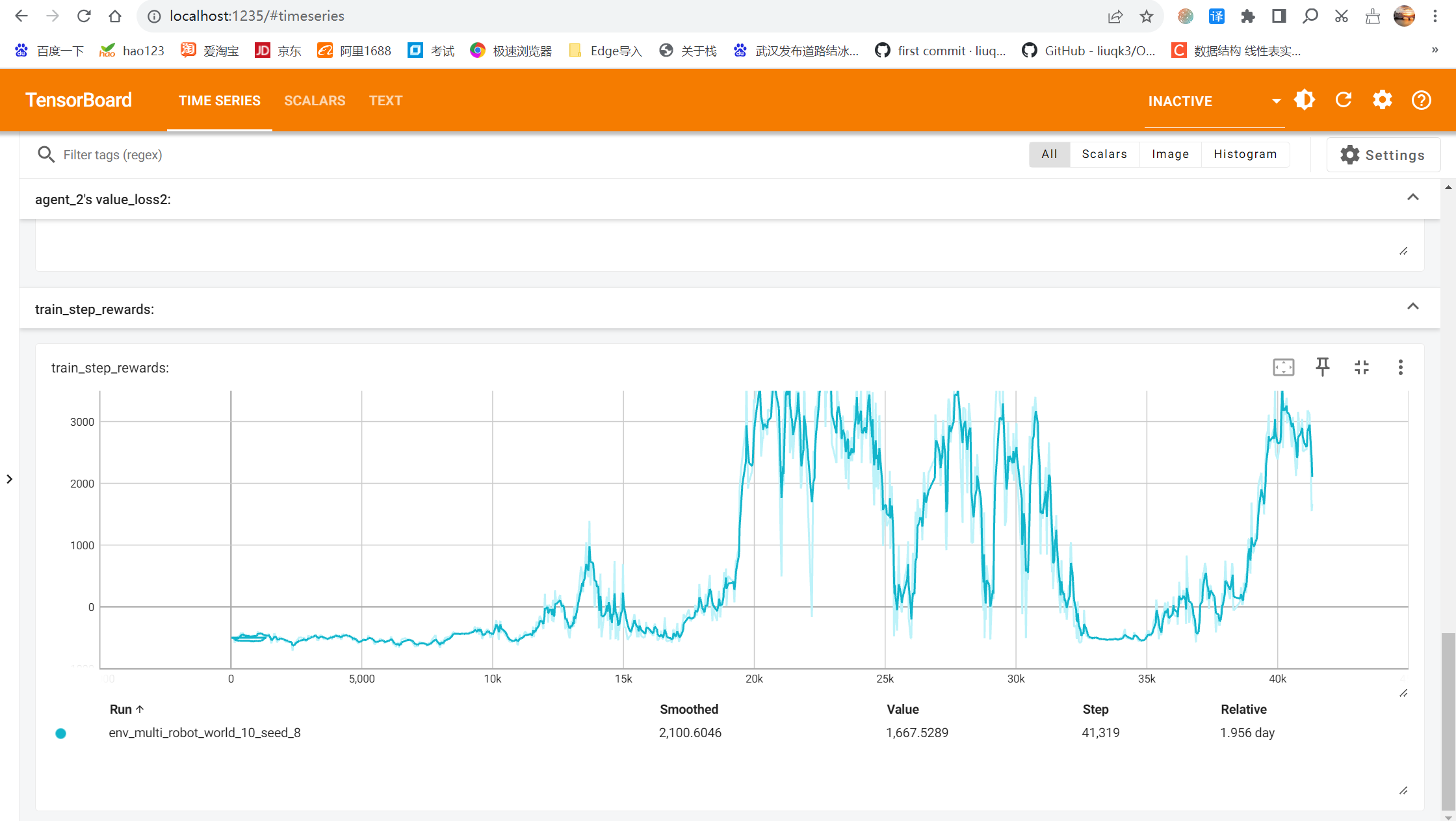


图3.25 MATD3模型的奖励曲线

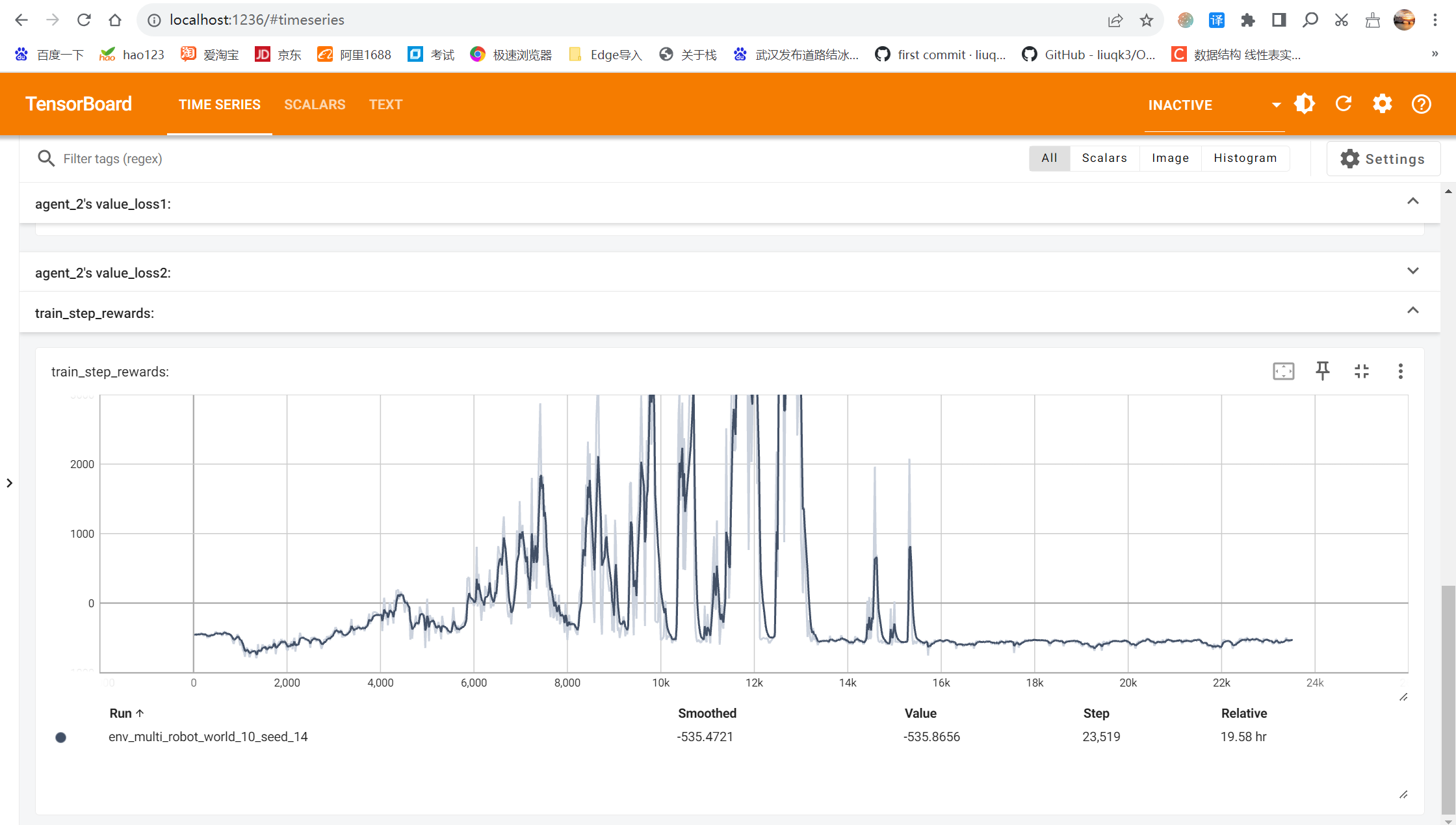


图3.26 MASAC模型的奖励曲线

在多智能体强化学习模型的奖励曲线表现上，MADDPG模型以最终收敛于11832的奖励值和11343的平滑值领先，这表明MADDPG在训练过程中能够更有效地学习并优化其策略，以获得较高的累积奖励。这种表现得益于MADDPG的集中评论家网络，它能够更好地协调智能体间的合作，从而在多智能体环境中实现更高的性能。

相比之下，MATD3模型的奖励曲线最终收敛于1667，平滑值为2100，虽然在数值上低于MADDPG，但仍然显示出较好的学习效果。MATD3通过双延迟机制来减少智能体间的相互影响，这有助于它在某些情况下实现稳定的学习进度。

MASAC模型的奖励曲线表现则相对较差，其最终收敛值和平滑值均为-535，这表明MASAC在策略学习过程中遇到了更多的挑战，其策略优化的效率相对较低。MASAC的高价值损失指示其在策略评估和更新过程中存在一定的局限性，尤其是在处理多智能体环境中的复杂交互时。

## 模型测试与结果评估

函数 eval(args, env, agents):

初始化 episode\_data 为一个空列表

初始化 detector\_info 为一个空列表

初始化 infos 为每个 agent 创建一个 INFO 对象

创建 gif 存储目录 gif\_dir

获取 gif 目录下当前已存在的 gif 数量 gif\_num

对于每个 agent 在 agents 中:

从 output\_dir 加载该 agent 的模型

对于每个 episode 从 1 到 args.evaluate\_times:

重置环境，获取初始 state 和 detector

对于每个 agent:

初始化该 agent 的 infos

初始化 frame\_list 为一个空列表，用于保存 gif 动画的每一帧

在每一回合中执行:

对于每个 agent，基于当前 state 选择一个动作，并加入噪声

执行动作，获取下一个状态、奖励、终止标志、截断标志和 detector 信息

如果 render\_mode 为 "rgb\_array" 且 evaluate\_times 小于 30:

将当前帧保存到 frame\_list 列表

计算 done 状态（终止或截断）

更新每个 agent 的 infos，记录当前的 done 状态、奖励和 detector 信息

更新当前状态为 next\_state

计算每个 episode 的总奖励 sum\_reward:

计算每个 agent 的平均奖励并累加到 sum\_reward

记录当前 episode 的 detector 信息到 detector\_info 列表

如果 render\_mode 为 "rgb\_array" 且 evaluate\_times 小于 30:

将 frame\_list 中的帧保存为 gif 文件

将所有 detector 信息保存到 CSV 文件 detector\_info.csv

上述伪代码描述了三种多智能体强化学习模型的测试过程。首先，加载每个智能体的训练模型，并初始化评估所需的变量和数据结构。然后，在指定次数的评估回合中，环境被重置，每个智能体基于当前状态选择动作并执行，记录状态转移、奖励和是否终止的标志。如果处于指定条件下，会将每个回合的渲染帧保存为GIF文件。在每个回合结束时，计算并记录每个智能体的平均奖励以及环境中的其他信息，并将这些数据保存到CSV文件中，最终输出评估结果。

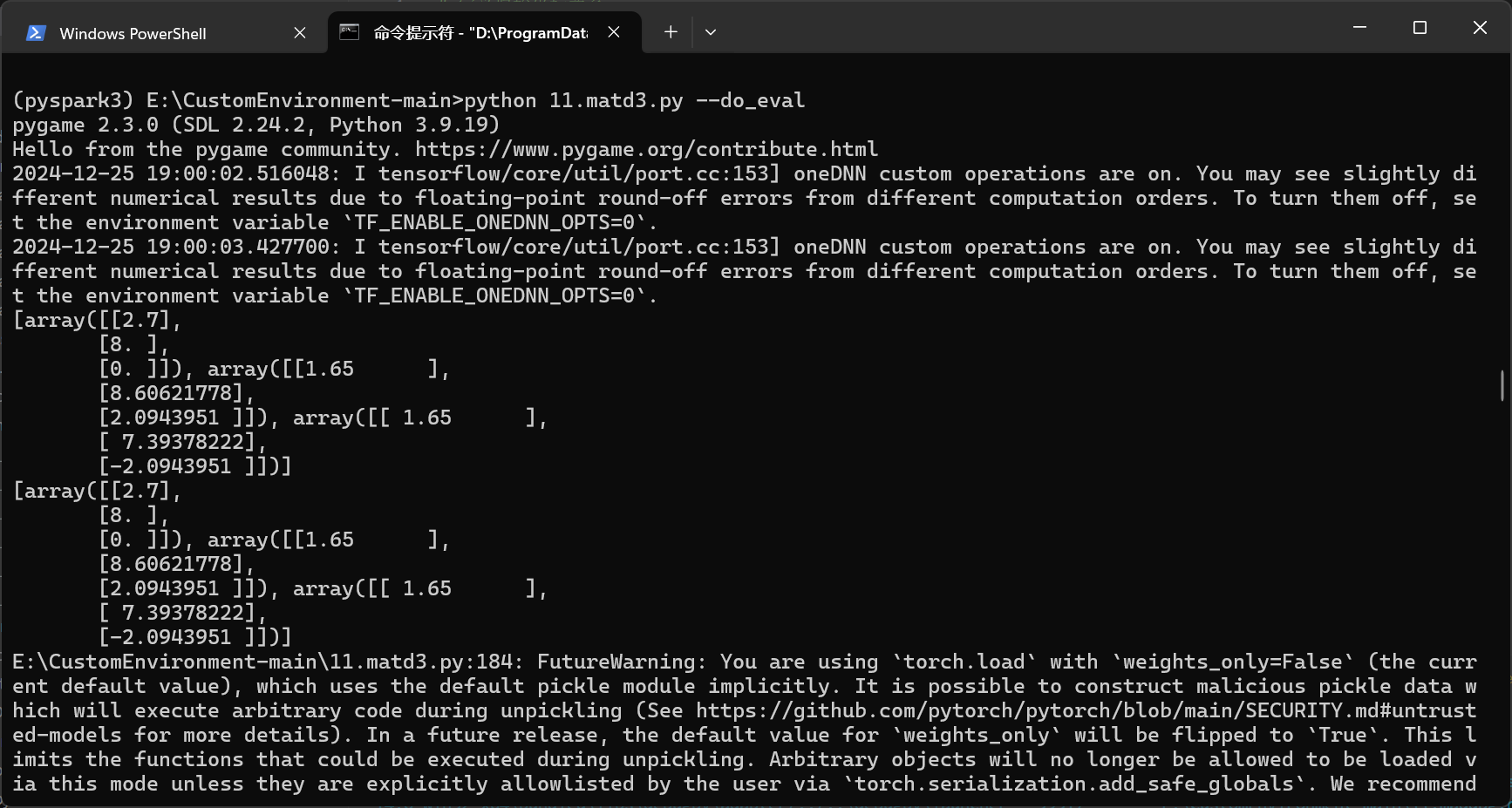


图3.27 MATD3模型的测试过程

在命令行中输入测试指令后，将调用已训练好的模型进行测试，测试结果为动态图gif.

3.4.1 常规场景下三种模型性能对比与分析

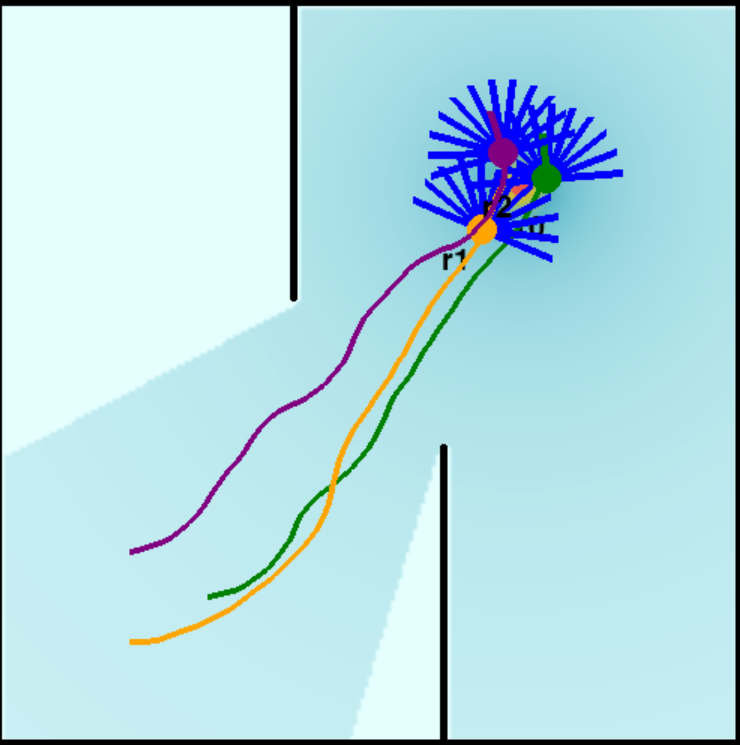
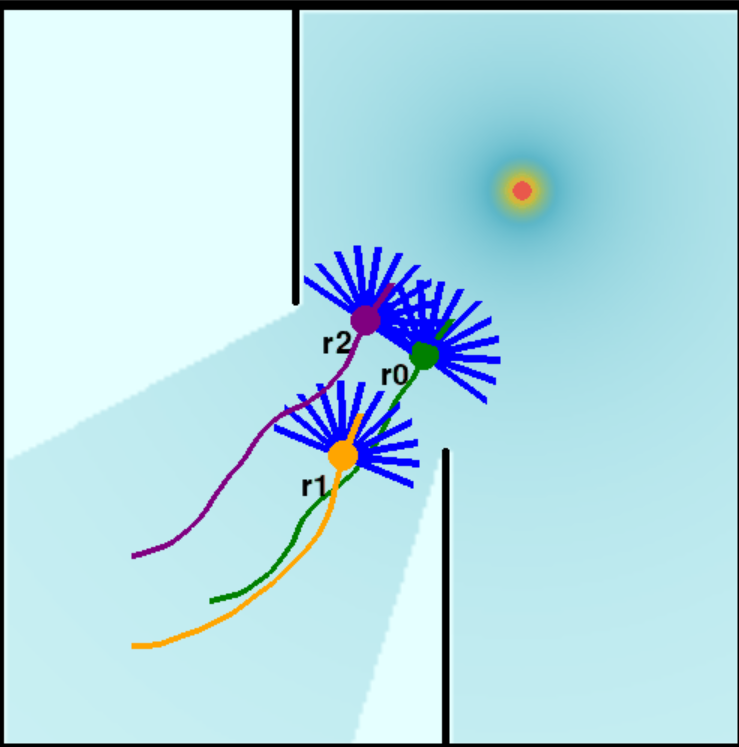
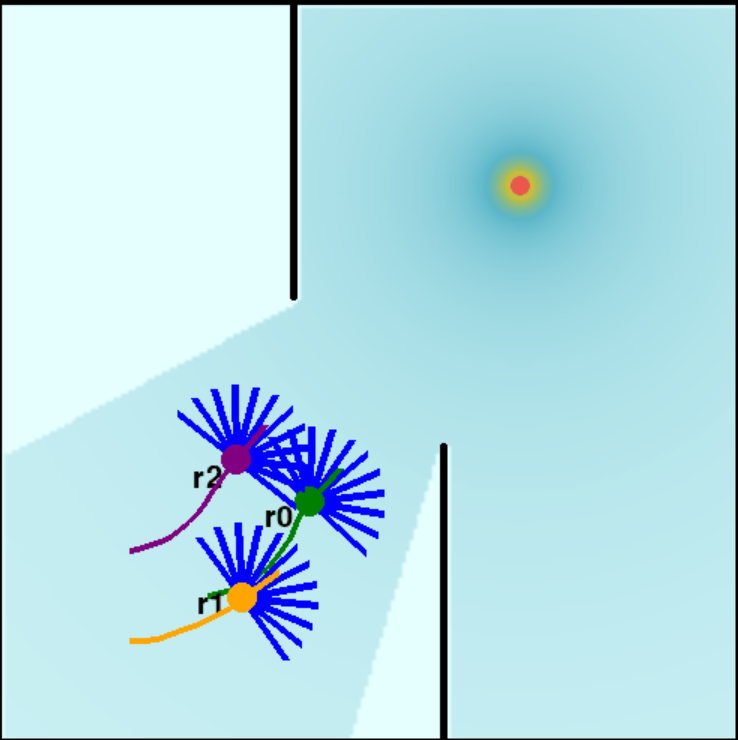


图3.28 MADDPG常规场景下测试结果

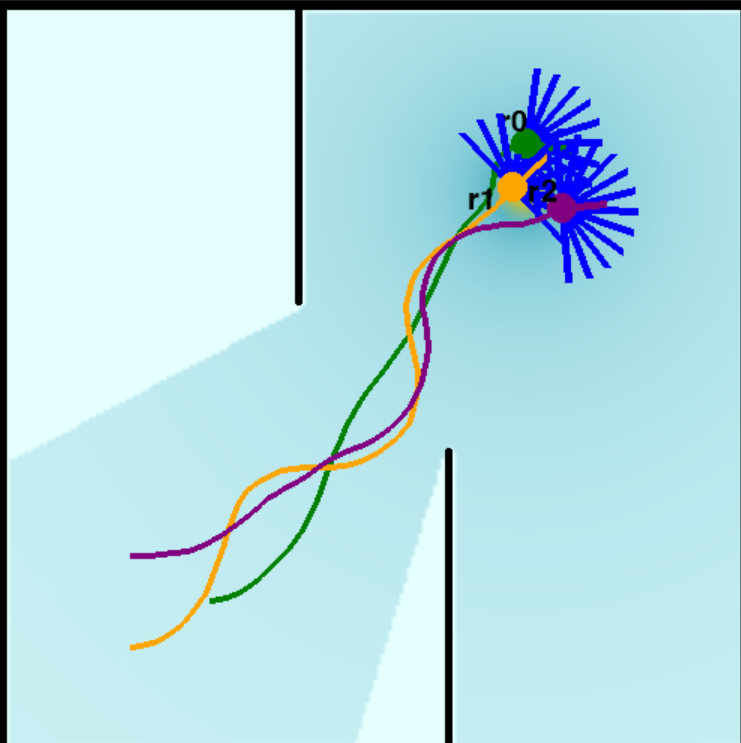
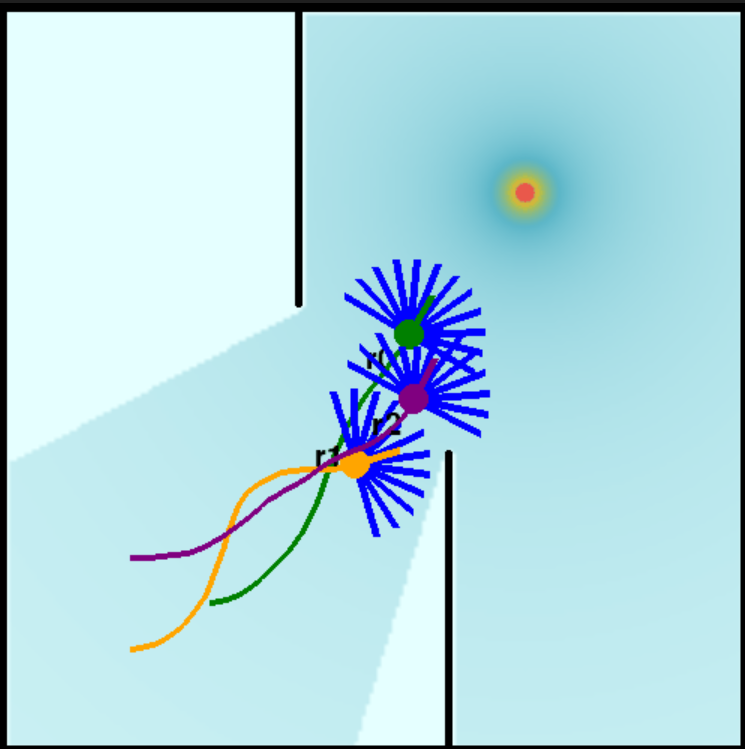
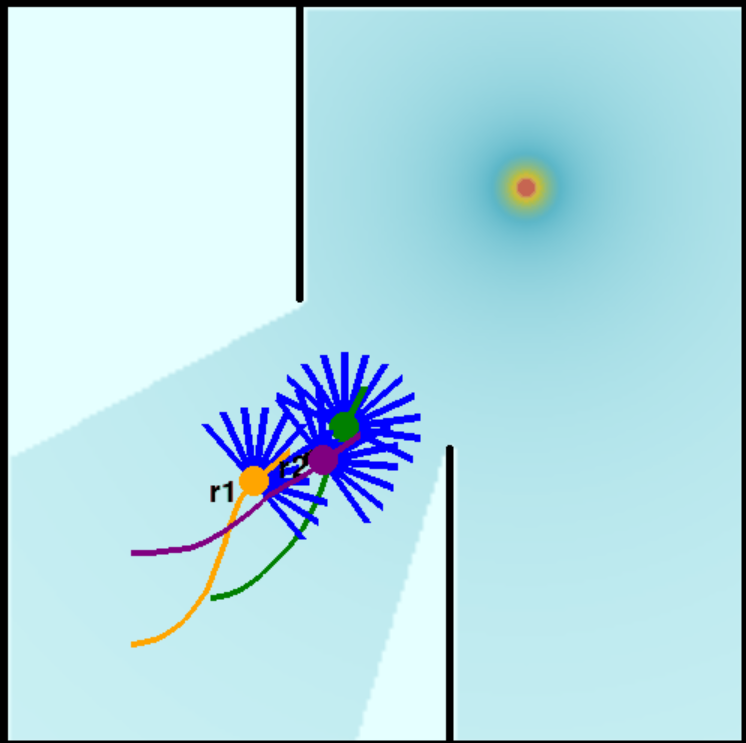


图3.29 MASAC常规场景下测试结果

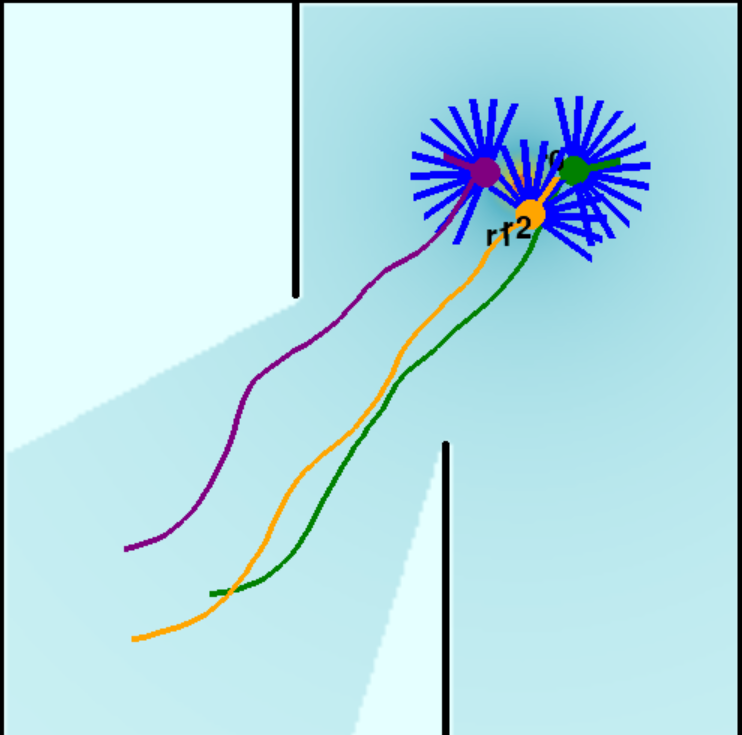
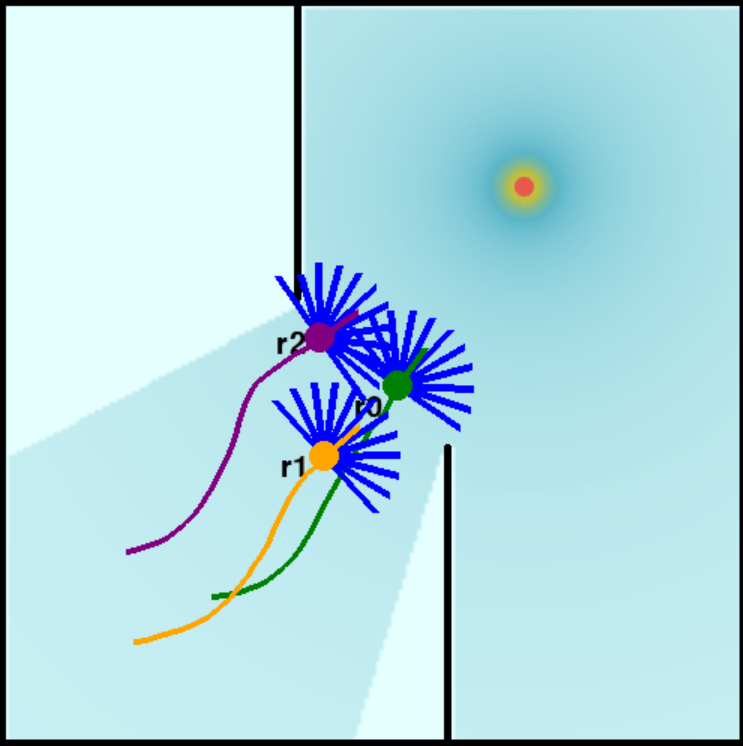


图3.30 MATD3常规场景下测试结果

在常规场景下的测试结果来看，MADDPG、MATD3、MASAC三种多智能体强化学习算法均能够有效地完成辐射源定位任务，表现出智能体在环境中能够准确地找到目标位置并避免碰撞。然而，在这三种算法的测试结果中，MASAC模型的轨迹表现出一些异常现象，尤其是智能体之间的相互交叠问题。

具体来说，虽然MASAC算法能够完成任务，但在多智能体协作过程中，智能体的路径规划没有充分考虑到彼此间的空间布局和相互间的影响，导致智能体在某些时刻相互靠近并发生交叠现象。这种现象表现为多个智能体在执行任务时会在同一空间位置聚集，缺乏必要的分散策略，导致多个智能体重复探索同一区域或相互干扰，从而影响任务的效率和性能。

该现象的根本原因与MASAC算法的策略优化机制有关。在MASAC中，智能体通常依赖于软策略来平衡探索和利用。然而，这种策略有时会导致智能体选择较为相似的行动，从而在执行过程中没有有效的空间分配策略，造成智能体之间的轨迹重叠。相比之下，MADDPG和MATD3由于其不同的更新机制和策略优化方法，能够更好地处理多智能体协作和避免相互干扰。

3.4.2 复杂场景下三种模型性能对比与分析

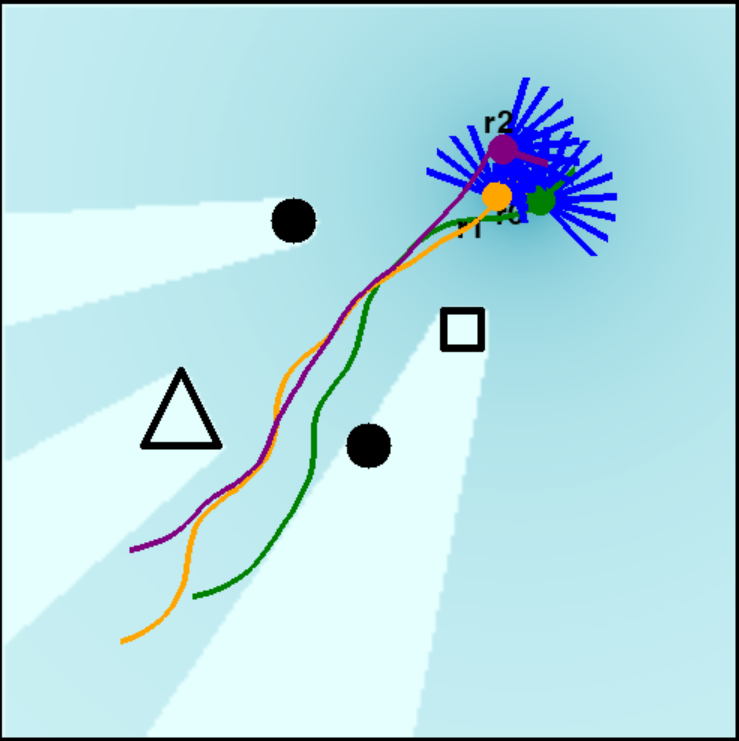
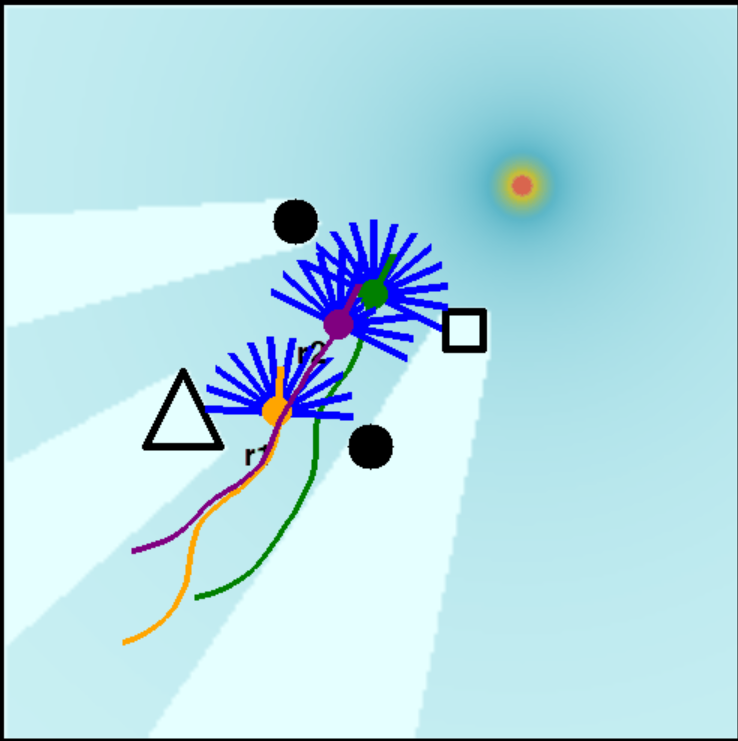
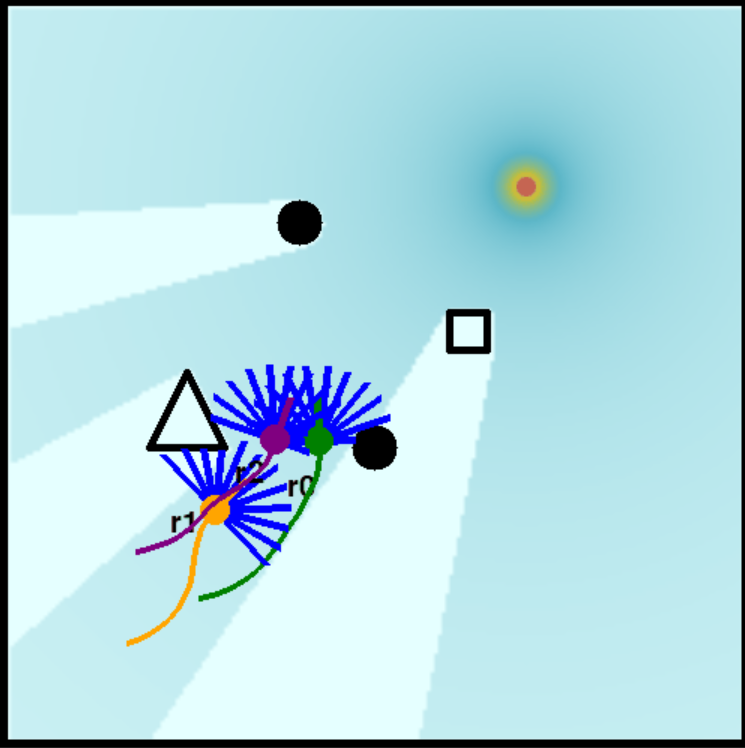


图3.31 MADDPG复杂场景下测试结果

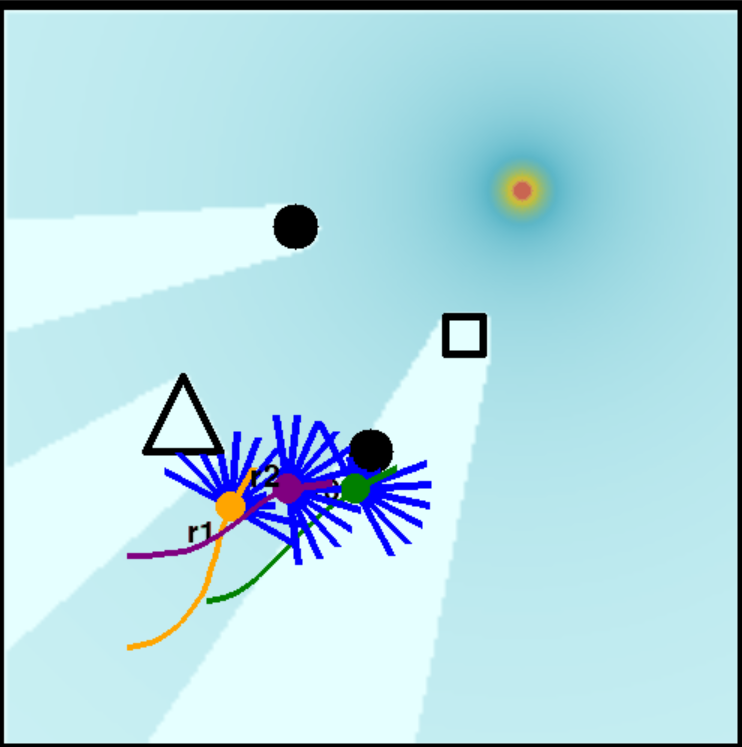


图3.32 MASAC复杂场景下测试结果

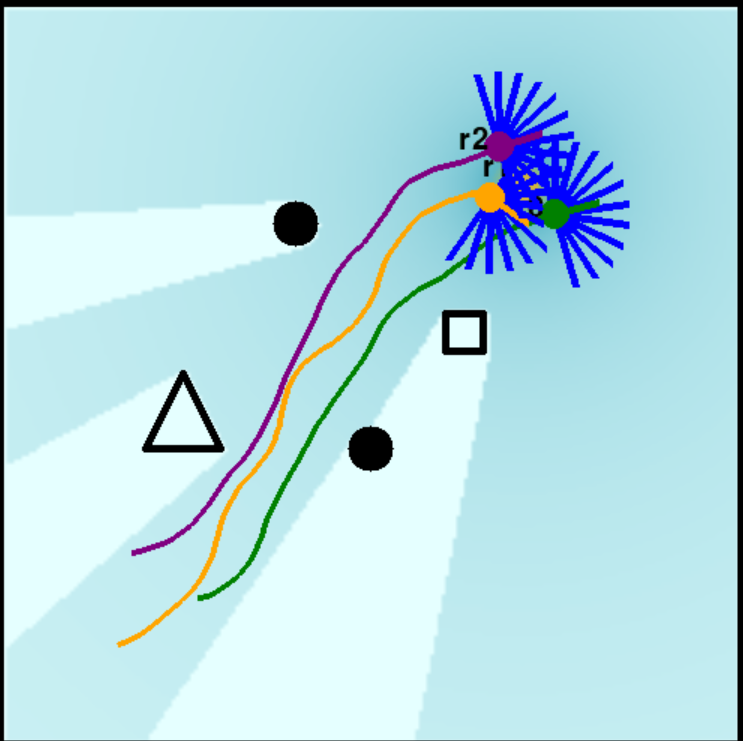
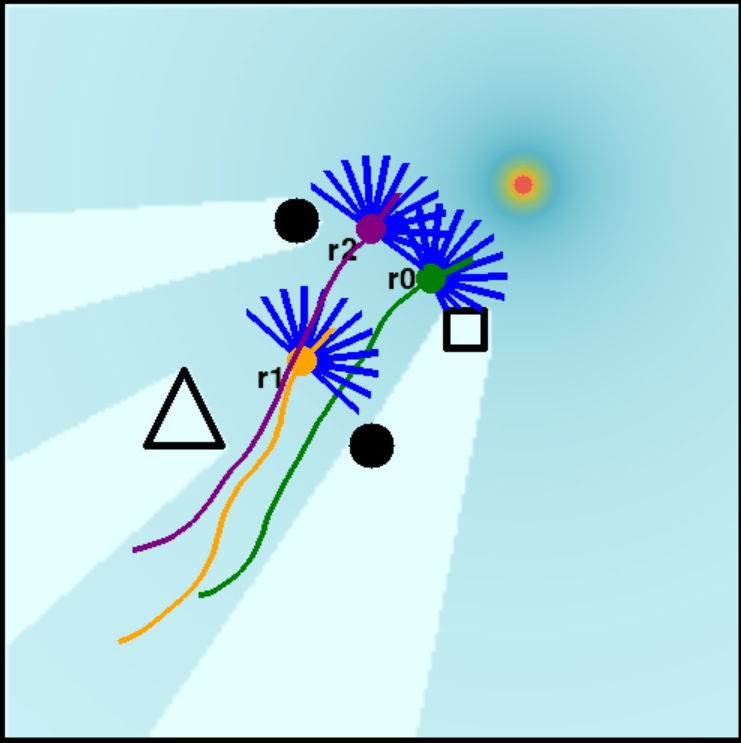
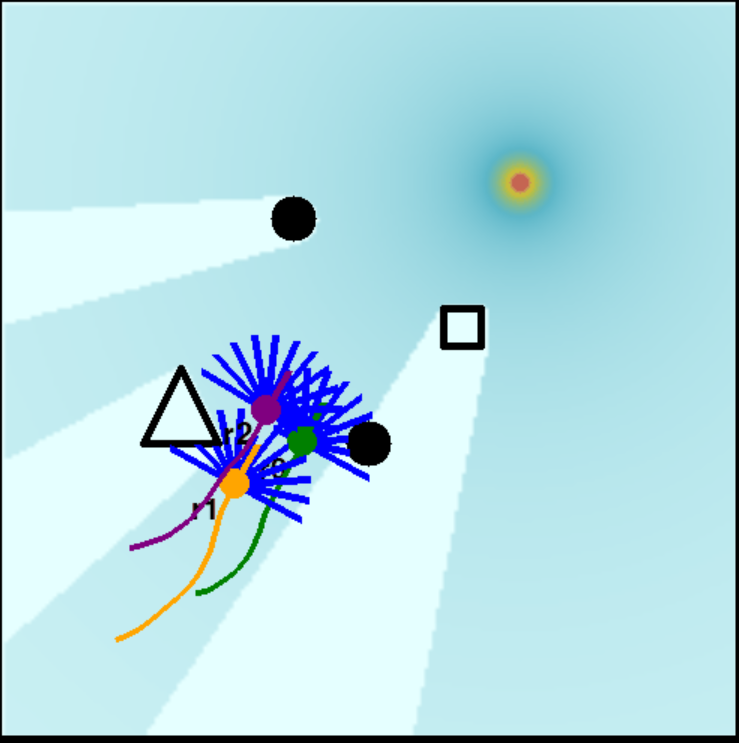


图3.33 MATD3复杂场景下测试结果

在复杂场景下的测试结果表明，MADDPG和MATD3算法能够有效地定位辐射源，显示出它们在多智能体协同任务中的有效性。然而，MASAC模型下的智能体在执行任务时与障碍物发生碰撞，导致任务失败。这是因为MASAC算法在处理复杂环境中的动态障碍物和多智能体间的交互时，缺乏有效的碰撞避免机制。此外，MADDPG算法虽然能够定位辐射源，但在智能体的轨迹规划上存在交叠现象，这是由于算法在优化过程中未能充分考虑智能体间的相互影响和环境约束，导致在实际执行中智能体行动的冲突。

相比之下，MATD3算法在复杂场景下展现了卓越的性能，它不仅避免了与障碍物的碰撞，还解决了轨迹交叠的问题。MATD3算法的优势源于其设计中对多智能体协同和个体行动的精细控制，以及对环境动态的敏感响应。这种算法通过双延迟更新机制，减少了智能体在决策时的价值函数高估问题，从而在复杂环境中保持了更好的稳定性和效率。此外，MATD3算法的这种稳定性和效率也得到了其他研究的支持，表明其在多无人机协同路径规划等任务中具有较好的表现。

# 结论

在这次实验中，我专注于开发和验证了三种多智能体深度强化学习算法——MADDPG、MASAC和MATD3，以实现避障无人车的放射源定位。我的目标是通过这些算法，使无人车能够在复杂环境中自主探索、智能避障、高效协作，并精准定位放射源。我设计了一系列实验，旨在比较这些算法在定位精度、避障能力、任务完成时间和动态复杂环境中的稳定性及鲁棒性。

实验结果显示，MATD3算法在复杂环境中表现最为出色，它不仅避免了与障碍物的碰撞，还解决了轨迹交叠问题。相比之下，MADDPG算法虽然能够定位辐射源，但在轨迹规划上存在交叠现象；而MASAC算法在避障方面表现不佳，导致任务失败。这些发现对于理解不同算法在实际应用中的适用性和局限性至关重要。

在实验过程中，我对每种算法的核心结构和参数进行了深入优化，包括奖励函数设计、策略网络架构和更新频率等。我还探索了多智能体系统中的通信机制、协作方式和环境感知能力对算法效果的影响，这些因素对于实现无人车多智能体系统在复杂环境中的高效协作至关重要。

最终，我的目标是构建一个高效、稳定、具有强鲁棒性和适应性的无人车多智能体系统。这个系统不仅能够在复杂的动态环境中完成放射源的精准定位任务，还能为无人车技术在紧急救援、环境监测和智能交通等领域的应用提供可靠的算法基础和实践经验。通过全面对比分析三种算法的表现，我为多智能体强化学习算法的选择与优化提供了实用的决策依据，为无人车技术的发展和应用贡献了自己的一份力量。

参考文献

[1]简泽民,申国伟,刘莉,等.基于多智能体强化学习的防空编队部署方法[J].电光与控制,2024,31(12):91-97.

[2]陈东旭,李岩松,许智光,等.基于BNN-IKAE双层多智能体深度强化学习的大电网母线电压智能自动调整方法[J].电工电能新技术,2024,43(11):68-79.

[3]禹鑫燚,刘飞,欧林林.基于图神经网络的多智能体路径规划方法[J].高技术通讯,2024,34(10):1081-1090.

[4]蔡玉,官铮,王增文,等.基于多智能体深度强化学习的车联网区分业务资源分配算法[J].计算机工程与科学,2024,46(10):1757-1764.

[5]李方昱,刘金溢,黄琰婷,等.基于深度强化学习的多智能体分布式事件触发优化控制[J].中国科学:技术科学,2024,54(10):1991-2002.

[6]高冠中,姚建国,严嘉豪,等.基于多智能体深度强化学习的配-微网协同优化调度研究[J].智慧电力,2024,52(09):80-87.

[7]钟孙健,张德平.基于EC-MAVEN算法的多智能体多约束作战任务分配方法[J].火力与指挥控制,2024,49(09):58-63.

[8]欧嘉俊,曾伟良,李谕锋,等.基于多智能体注意力机制的自动巡检路线强化学习模型[J].广东工业大学学报,2024,41(05):39-47+71.

[9]马刚,马健,颜云松,等.基于多智能体强化学习的配电网电压分散控制[J].综合智慧能源,2024,46(10):32-39.

[10]严地宝,文红,侯文静,等.面向任务卸载的鲁棒多智能体联邦强化学习[J].通信技术,2024,57(08):850-854.

[11]张海龙,赵永娟,张鹏飞,等.基于多智能体强化学习的重载运输车队队列控制[J].兵器装备工程学报,2024,45(08):45-50+66.

[12]刘全,施眉龙,黄志刚,等.基于双视角建模的多智能体协作强化学习方法[J].计算机学报,2024,47(07):1582-1594.

[13]钟意,杨勇,姜学涛,等.基于多智能体深度Q网络交互的板壳加强筋生长式设计[J].中国机械工程,2024,35(08):1397-1404.

[14]缪刘洋,朱其新,朱永红.基于多智能体强化学习的滑模控制器参数整定[J].机床与液压,2024,52(11):160-166.

[15]高玉钊,聂一鸣.基于值函数分解的多智能体深度强化学习方法研究综述[J].计算机科学,2024,51(S1):34-42.

[16]方馨,秦子健,高新平,等.基于多智能体深度强化学习的智慧医疗网络计算卸载方法[J].移动通信,2024,48(06):86-90.

[17]Rao J ,Wang C ,Liu M , et al.ISFORS-MIX: Multi-agent reinforcement learning with Importance-Sampling-Free Off-policy learning and Regularized-Softmax Mixing network[J].Knowledge-Based Systems,2025,309112881-112881.

[18]Talihati B ,Fu S ,Zhang B , et al.Community shared ES-PV system for managing electric vehicle loads via multi-agent reinforcement learning[J].Applied Energy,2025,380125039-125039.

[19]Zhang X ,He Z ,Zhu Y , et al.Coupled vehicle-signal control based on Stackelberg Game Enabled Multi-agent Reinforcement Learning in mixed traffic environment[J].Physica A: Statistical Mechanics and its Applications,2025,658130289-130289.

[20]Alagha A ,Mizouni R ,Singh S , et al.Adaptive target localization under uncertainty using Multi-Agent Deep Reinforcement Learning with knowledge transfer[J].Internet of Things,2025,29101447-101447.

[21]Chernyavskiy A ,Skrynnik A ,Panov A .Applying opponent and environment modelling in decentralised multi-agent reinforcement learning[J].Cognitive Systems Research,2025,89101306-101306.

[22]Jing Y ,Guo B ,Li N , et al.Scalable order dispatching through Federated Multi-Agent Deep Reinforcement Learning[J].Expert Systems With Applications,2025,264125792-125792.

[23]Zhou J ,Xiang Y ,Zhang X , et al.Optimal self-consumption scheduling of highway electric vehicle charging station based on multi-agent deep reinforcement learning[J].Renewable Energy,2025,238121982-121982.

[24]Haupt A ,Christoffersen P ,Damani M , et al.Formal contracts mitigate social dilemmas in multi-agent reinforcement learning[J].Autonomous Agents and Multi-Agent Systems,2024,38(2):51-51.

[25]Song Z ,Wang X ,Wu Q , et al.A Task Offloading Strategy Based on Multi-Agent Deep Reinforcement Learning for Offshore Wind Farm Scenarios[J].Computers, Materials & Continua,2024,81(1):985-1008.

[26]Wen G ,Niu B .Optimized distributed formation control using identifier-critic-actor reinforcement learning for a class of stochastic nonlinear multi-agent systems.[J].ISA transactions,2024,1551-10.

[27]Wang J ,Zhang M ,Yin Q , et al.Multi-agent reinforcement learning for task offloading with hybrid decision space in multi-access edge computing[J].Ad Hoc Networks,2025,166103671-103671.

[28]Ejaz A M ,Wu G ,Ahmed A , et al.Utility-Driven End-to-End Network Slicing for Diverse IoT Users in MEC: A Multi-Agent Deep Reinforcement Learning Approach[J].Sensors, 2024, 24(17):

5558-5558.

[29]Tovey S ,Lohrmann C ,Holm C .Emergence of chemotactic strategies with multi-agent reinforcement learning[J].Machine Learning: Science and Technology,2024,5(3):035054-035054.

[30]Bildik E ,Tsourdos A ,Perrusquía A , et al.Decoys Deployment for Missile Interception: A Multi-Agent Reinforcement Learning Approach[J].Aerospace,2024,11(8):684-684.

[31]Song G ,Ifaei P ,Ha J , et al.The AI circular hydrogen economist: Hydrogen supply chain design via hierarchical deep multi-agent reinforcement learning[J].Chemical Engineering Journal,2024,497154464-154464.

[32]Bhuyan M ,Sarma K K ,Misra D D , et al.Adaptive Hybrid Beamforming Codebook Design Using Multi-Agent Reinforcement Learning for Multiuser Multiple-Input–Multiple-Output Systems[J].Applied Sciences,2024,14(16):7109-7109.

[33]Gadiraju S D ,Karmakar P ,Shah K V , et al.GLIDE: Multi-Agent Deep Reinforcement Learning for Coordinated UAV Control in Dynamic Military Environments[J].Information,2024,15(8):477-477.

[34]Wang C ,Wang Y ,Yuan Y , et al.Joint computation offloading and resource allocation for end-edge collaboration in internet of vehicles via multi-agent reinforcement learning.[J].Neural networks : the official journal of the International Neural Network Society,2024,179106621.

[35]Xi J ,Garcia A ,Chen C Y , et al.Multi-agent reinforcement learning for multi-area power exchange[J].Electric Power Systems Research,2024,235110711-110711.

[36]Baccour E ,Erbad A ,Mohamed A , et al.Multi-agent reinforcement learning for privacy-aware distributed CNN in heterogeneous IoT surveillance systems[J].Journal of Network and Computer Applications,2024,230103933-103933.

附 录

**11.maddpg.py:**

1. import argparse
2. import torch
3. import torch.nn as nn
4. import torch.nn.functional as F
5. import numpy as np
6. import multirobotenv
7. from torch.utils.tensorboard import SummaryWriter
8. from collections import defaultdict
9. import os
10. import matplotlib.pyplot as plt
11. from PIL import Image
12. import pandas as pd
13. from datetime import datetime
14. from tqdm import tqdm
15. def orthogonal\_init(layer, gain=1.0):
16. for name, param in layer.named\_parameters():
17. if 'bias' in name:
18. nn.init.constant\_(param, 0)
19. elif 'weight' in name:
20. nn.init.orthogonal\_(param, gain=gain)
21. class QNet(nn.Module):
22. def \_\_init\_\_(self, dim\_state, dim\_action, args):
23. super().\_\_init\_\_()
24. self.fc1 = nn.Linear(args.num\_agents \* (dim\_state + dim\_action), args.hidden\_dim)
25. self.fc2 = nn.Linear(args.hidden\_dim, args.hidden\_dim)
26. self.fc3 = nn.Linear(args.hidden\_dim, 1)
27. if args.use\_orthogonal\_init:
28. orthogonal\_init(self.fc1)
29. orthogonal\_init(self.fc2)
30. orthogonal\_init(self.fc3)
31. def forward(self, state, action):
32. sa = torch.cat(state + action, 1)
33. x = F.relu(self.fc1(sa))
34. x = F.relu(self.fc2(x))
35. x = self.fc3(x)
36. return x
37. class PolicyNet(nn.Module):
38. def \_\_init\_\_(self, dim\_state, dim\_action, args):
39. super().\_\_init\_\_()
40. self.max\_action = args.max\_action
41. self.fc1 = nn.Linear(dim\_state, args.hidden\_dim)
42. self.fc2 = nn.Linear(args.hidden\_dim, args.hidden\_dim)
43. self.fc3 = nn.Linear(args.hidden\_dim, dim\_action)
44. if args.use\_orthogonal\_init:
45. orthogonal\_init(self.fc1)
46. orthogonal\_init(self.fc2)
47. orthogonal\_init(self.fc3)
48. def forward(self, state):
49. x = F.relu(self.fc1(state))
50. x = F.relu(self.fc2(x))
51. x = self.max\_action \* torch.tanh(self.fc3(x))
52. return x
54. class ReplayBuffer(object):
55. def \_\_init\_\_(self, num\_agents, obs\_dim\_n, action\_dim\_n, args):
56. self.N = num\_agents  *# The number of agents*
57. self.buffer\_size = args.buffer\_size
58. self.batch\_size = args.batch\_size
59. self.count = 0
60. self.current\_size = 0
61. self.device = args.device
62. self.buffer\_obs\_n = {f'agent\_{i}': np.empty((self.buffer\_size, obs\_dim\_n)) for i in range(self.N)}
63. self.buffer\_a\_n = {f'agent\_{i}': np.empty((self.buffer\_size, action\_dim\_n)) for i in range(self.N)}
64. self.buffer\_r\_n = {f'agent\_{i}': np.empty((self.buffer\_size, 1)) for i in range(self.N)}
65. self.buffer\_s\_next\_n = {f'agent\_{i}': np.empty((self.buffer\_size, obs\_dim\_n)) for i in range(self.N)}
66. self.buffer\_done\_n = {f'agent\_{i}': np.empty((self.buffer\_size, 1)) for i in range(self.N)}
67. def store\_transition(self, obs\_n, a\_n, r\_n, obs\_next\_n, done\_n):
68. for i in range(self.N):
69. agent\_key = f'agent\_{i}'
70. self.buffer\_obs\_n[agent\_key][self.count] = obs\_n[agent\_key]
71. self.buffer\_a\_n[agent\_key][self.count] = a\_n[agent\_key]
72. self.buffer\_r\_n[agent\_key][self.count] = r\_n[agent\_key]
73. self.buffer\_s\_next\_n[agent\_key][self.count] = obs\_next\_n[agent\_key]
74. self.buffer\_done\_n[agent\_key][self.count] = done\_n[agent\_key]
75. self.count = (self.count + 1) % self.buffer\_size
76. self.current\_size = min(self.current\_size + 1, self.buffer\_size)
77. def sample(self, ):
78. index = np.random.choice(self.current\_size, size=self.batch\_size, replace=False)
79. batch\_obs\_n, batch\_a\_n, batch\_r\_n, batch\_obs\_next\_n, batch\_done\_n = {}, {}, {}, {}, {}
80. for i in range(self.N):
81. agent\_key = f'agent\_{i}'
82. batch\_obs\_n[agent\_key] = torch.tensor(self.buffer\_obs\_n[agent\_key][index], dtype=torch.float).to(self.device)
83. batch\_a\_n[agent\_key] = torch.tensor(self.buffer\_a\_n[agent\_key][index], dtype=torch.float).to(self.device)
84. batch\_r\_n[agent\_key] = torch.tensor(self.buffer\_r\_n[agent\_key][index], dtype=torch.float).to(self.device)
85. batch\_obs\_next\_n[agent\_key] = torch.tensor(self.buffer\_s\_next\_n[agent\_key][index], dtype=torch.float).to(self.device)
86. batch\_done\_n[agent\_key] = torch.tensor(self.buffer\_done\_n[agent\_key][index], dtype=torch.float).to(self.device)
87. return batch\_obs\_n, batch\_a\_n, batch\_r\_n, batch\_done\_n, batch\_obs\_next\_n
88. class MADDPG(nn.Module):
89. def \_\_init\_\_(
90. self,
91. max\_action=1,
92. num\_states=6,
93. num\_actions=5,
94. args=None,
95. agent\_id=0,
96. ):
97. super().\_\_init\_\_()
98. self.id = agent\_id
99. self.max\_action = max\_action
100. self.num\_states = num\_states
101. self.num\_actions = num\_actions
102. self.device = args.device
103. self.use\_grad\_clip = args.use\_grad\_clip
104. self.Q1 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
105. self.Mu = PolicyNet(self.num\_states, self.num\_actions, args).to(self.device)
106. self.target\_Q1 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
107. self.target\_Mu = PolicyNet(self.num\_states, self.num\_actions, args).to(self.device)
108. self.target\_Q1.load\_state\_dict(self.Q1.state\_dict())
109. self.target\_Mu.load\_state\_dict(self.Mu.state\_dict())
110. self.Q1\_optimizer = torch.optim.Adam(self.Q1.parameters(), lr=args.lr\_c)
111. self.Mu\_optimizer = torch.optim.Adam(self.Mu.parameters(), lr=args.lr\_a)
112. def get\_action(self, state, noise\_std=0):
113. state = torch.from\_numpy(state).unsqueeze(0).float().to(self.device)
114. action = self.Mu(state).cpu().data.numpy().flatten()
115. noise =  np.random.normal(0, noise\_std, size=self.num\_actions)
116. action = (action + noise).clip(-self.max\_action, self.max\_action)
117. return action.astype(np.float32)
118. def compute\_value\_loss(self, args, agents, s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch):
119. *# Compute target\_Q*
120. with torch.no\_grad():
121. *# 让目标策略网络做预测。*
122. batch\_a\_next = {agent\_id: agents[agent\_id].target\_Mu(ns\_batch[agent\_id]) for agent\_id in agents}
123. *# 让两个目标价值网络做预测。*
124. q1 = self.target\_Q1(list(ns\_batch.values()), list(batch\_a\_next.values()))
125. *# 计算 TD 目标。*
126. y = r\_batch[self.id] + args.gamma \* q1 \* (1 - d\_batch[self.id])
127. *# 让两个价值网络做预测。*
128. qvals1 = self.Q1(list(s\_batch.values()), list(a\_batch.values()))
129. value\_loss1 = F.mse\_loss(y, qvals1)
130. return value\_loss1
131. def compute\_policy\_loss(self, s\_batch, a\_batch):
132. a\_batch[self.id] = self.Mu(s\_batch[self.id])
133. policy\_loss = -self.Q1(list(s\_batch.values()), list(a\_batch.values())).mean()
134. return policy\_loss
135. def soft\_update(self, tau=0.01):
136. def soft\_update\_(target, source, tau\_=0.01):
137. for target\_param, param in zip(target.parameters(), source.parameters()):
138. target\_param.data.copy\_(target\_param.data \* (1.0 - tau\_) + param.data \* tau\_)
139. soft\_update\_(self.target\_Q1, self.Q1, tau)
140. soft\_update\_(self.target\_Mu, self.Mu, tau)
141. def save\_model(self, output\_dir):
142. save\_path = os.path.join(output\_dir, f"{self.id}\_model.bin")
143. torch.save(self.Mu.state\_dict(), save\_path)
144. def load\_model(self, output\_dir):
145. save\_path = os.path.join(output\_dir, f"{self.id}\_model.bin")
146. self.Mu.load\_state\_dict(torch.load(save\_path))
147. def learn(self, replay\_buffer: ReplayBuffer, args, agents: dict, infos: dict):
148. s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch = replay\_buffer.sample()
149. *# Optimize the critic*
150. value\_loss1 = self.compute\_value\_loss(args, agents, s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch)
151. infos[self.id].log["value\_loss1"].append(value\_loss1.item())
152. self.Q1\_optimizer.zero\_grad()
153. value\_loss1.backward()
154. if self.use\_grad\_clip:
155. torch.nn.utils.clip\_grad\_norm\_(self.Q1.parameters(), 10.0)
156. self.Q1\_optimizer.step()
157. *# Optimize the actor*
158. policy\_loss = self.compute\_policy\_loss(s\_batch, a\_batch)
159. *# Optimize the actor*
160. self.Mu\_optimizer.zero\_grad()
161. policy\_loss.backward()
162. if self.use\_grad\_clip:
163. torch.nn.utils.clip\_grad\_norm\_(self.Mu.parameters(), 10.0)
164. self.Mu\_optimizer.step()
165. self.soft\_update(args.tau)
166. infos[self.id].log["policy\_loss"].append(policy\_loss.item())
167. class INFO:
168. def \_\_init\_\_(self):
169. self.log = defaultdict(list)
170. self.episode\_length = 0
171. self.episode\_reward = 0
172. self.max\_episode\_reward = -float("inf")
173. def put(self, done, reward, detector):
174. if done is True:
175. self.episode\_length += 1
176. self.episode\_reward += reward
177. self.log["episode\_length"].append(self.episode\_length)
178. self.log["episode\_reward"].append(self.episode\_reward)
179. if self.episode\_reward > self.max\_episode\_reward:
180. self.max\_episode\_reward = self.episode\_reward
181. self.episode\_length = 0
182. self.episode\_reward = 0
183. else:
184. self.episode\_length += 1
185. self.episode\_reward += reward
186. self.log["detector"].append(detector)
188. def reset(self):
189. self.log = defaultdict(list)
190. def train(args, env, agents: dict, replay\_buffer: ReplayBuffer, writer: SummaryWriter):
191. if args.add\_model:
192. for agent in agents.values():
193. agent.load\_model(args.output\_dir)
194. step = 0  *# global step counter*
195. max\_reward = 300
196. sum\_reward = 0
197. episode\_data = []
198. infos = {agent\_id: INFO() for agent\_id in env.possible\_agents}
199. for episode in tqdm(range(args.episode\_num), desc="Training Episodes"):
200. state, \_, detector = env.reset()
201. for agent\_id in env.possible\_agents:
202. infos[agent\_id].put(False, 0, detector[agent\_id])
203. while env.agents:
204. step += 1
205. if step < args.warmup\_steps:
206. *# action = {agent: env.components['robots'].robot\_list[i].cal\_des\_vel().reshape(2,) for i, agent in enumerate(env.agents)}*
207. action = {agent\_id: env.action\_space(agent\_id).sample() for agent\_id in env.agents}
208. else:
209. action = {agent\_id:agents[agent\_id].get\_action(state[agent\_id], noise\_std=args.noise\_std) for agent\_id in env.agents}
210. next\_state, reward, terminated, truncated, \_, detector = env.step(action)
211. done = {agent: terminated[agent] or truncated[agent] for agent in terminated}
212. replay\_buffer.store\_transition(state, action, reward, next\_state, done)
213. for agent\_id in env.possible\_agents:
214. infos[agent\_id].put(done[agent\_id], reward[agent\_id], detector[agent\_id])
215. state = next\_state
216. if step >= args.warmup\_steps:
217. for agent\_id in env.possible\_agents:
218. agents[agent\_id].learn(replay\_buffer, args, agents, infos)
219. *# episode finishes*
220. if (episode + 1) % 20 == 0:  *# print info every 20 episodes*
221. sum\_reward = 0
222. current\_time = datetime.now()
223. for agent\_id, info in infos.items():  *# record reward*
224. avg\_reward = np.mean(info.log["episode\_reward"][-20:])
225. value\_loss1 = np.mean(info.log["value\_loss1"][-20:]) if len(info.log["value\_loss1"]) > 0 else 0
226. value\_loss2 = np.mean(info.log["value\_loss2"][-20:]) if len(info.log["value\_loss2"]) > 0 else 0
227. policy\_loss = np.mean(info.log["policy\_loss"][-20:]) if len(info.log["policy\_loss"]) > 0 else 0
228. episode\_data.append({"time": current\_time.strftime("%Y-%m-%d %H:%M:%S"),"episode": episode + 1,"agent\_id": agent\_id, "reward": avg\_reward, "value\_loss1": value\_loss1, "value\_loss2": value\_loss2, "policy\_loss": policy\_loss})
229. *# print(f"episode={episode+1:d}, {agent\_id} reward={avg\_reward:.2f}, value loss1={value\_loss1:.4f}, value loss2={value\_loss2:.4f}, policy loss={policy\_loss:.4f}")*
230. sum\_reward +=avg\_reward
231. infos[agent\_id].reset()
232. writer.add\_scalar('train\_step\_rewards:',sum\_reward, global\_step=episode)
233. writer.add\_scalar(f'{agent\_id}\'s value\_loss1:',value\_loss1, global\_step=episode)
234. writer.add\_scalar(f'{agent\_id}\'s value\_loss2:',value\_loss2, global\_step=episode)
235. writer.add\_scalar(f'{agent\_id}\'s policy\_loss:',policy\_loss, global\_step=episode)
237. *# 将信息保存到DataFrame*
238. if (episode + 1) % 100 == 0:
239. df = pd.DataFrame(episode\_data)
240. csv\_file\_path = f"{args.output\_dir}/episode\_info.csv"
241. df.to\_csv(csv\_file\_path, mode='a', header=not os.path.exists(csv\_file\_path), index=False)
242. episode\_data.clear()
244. if sum\_reward > max\_reward:
245. max\_reward = sum\_reward
246. for agent\_id in env.possible\_agents:
247. agents[agent\_id].save\_model(args.output\_dir)
249. def eval(args, env, agents: dict, writer: SummaryWriter):
250. episode\_data = []
251. detector\_info = []
252. infos = {agent\_id: INFO() for agent\_id in env.possible\_agents}
253. gif\_dir = os.path.join(args.output\_dir, 'gif' + str(args.seed))
254. os.makedirs(gif\_dir, exist\_ok=True)
255. gif\_num = len([file for file in os.listdir(gif\_dir)])  *# current number of gif*
256. for agent in agents.values():
257. agent.load\_model(args.output\_dir)
258. for episode in tqdm(range(args.evaluate\_times), desc="Evaluating Episodes"):
259. state, \_, detector = env.reset()
260. for agent\_id in env.possible\_agents:
261. infos[agent\_id].put(False, 0, detector[agent\_id])
262. frame\_list = []  *# used to save gif*
263. while env.agents:
264. action = {agent\_id:agents[agent\_id].get\_action(state[agent\_id], noise\_std=0) for agent\_id in env.agents}
265. next\_state, reward, terminated, truncated, \_, detector = env.step(action)
266. if env.render\_mode == "rgb\_array" and args.evaluate\_times < 30:
267. frame\_list.append(Image.fromarray(env.render()))
268. done = {agent: terminated[agent] or truncated[agent] for agent in terminated}
269. for agent\_id in env.possible\_agents:
270. infos[agent\_id].put(done[agent\_id], reward[agent\_id], detector[agent\_id])
271. state = next\_state
272. *# episode finishes*
273. sum\_reward = 0
274. current\_time = datetime.now()
275. for i in range(env.steps):
276. detector\_info.append({"steps": i+1, "agent\_0":infos["agent\_0"].log["detector"][i], "agent\_1":infos["agent\_1"].log["detector"][i], "agent\_2":infos["agent\_2"].log["detector"][i]})
277. detector\_info.append({"steps": "avg\_reward", "agent\_0":infos["agent\_0"].log["episode\_reward"][-1], "agent\_1":infos["agent\_1"].log["episode\_reward"][-1], "agent\_2":infos["agent\_2"].log["episode\_reward"][-1]})
278. for agent\_id, info in infos.items():  *# record reward*
279. avg\_reward = info.log["episode\_reward"][-1]
280. *# print(f"episode={episode+1:d} steps={env.steps}, {agent\_id} reward={avg\_reward:.2f}")*
281. sum\_reward += avg\_reward
282. episode\_data.append({"time": current\_time.strftime("%Y-%m-%d %H:%M:%S"),"episode":episode+1, "steps": env.steps, "sum\_reward": sum\_reward})
283. if env.render\_mode == "rgb\_array" and args.evaluate\_times < 30:
284. frame\_list[0].save(os.path.join(gif\_dir, f'out{gif\_num + episode + 1}.gif'),
285. save\_all=True, append\_images=frame\_list[1:], duration=1, loop=0)
286. *# writer.add\_scalar('evaluate\_step\_rewards:',sum\_reward, global\_step=episode)*
287. *# 将信息保存到DataFrame*
288. df = pd.DataFrame(detector\_info)
289. csv\_file\_path = f"{args.output\_dir}/detector\_info.csv"
290. df.to\_csv(csv\_file\_path, mode='a', header=not os.path.exists(csv\_file\_path), index=False)
291. episode\_data.clear()
292. if \_\_name\_\_ == '\_\_main\_\_':
293. parser = argparse.ArgumentParser("Hyperparameters Setting for MADDPG and MATD3 in MPE environment")
294. parser.add\_argument("--env", default="multi\_robot\_world\_10\_3", type=str, help="Environment name.")
295. parser.add\_argument("--max\_action", default=1.0, type=float, help="Action scale, [-max, max].")
296. parser.add\_argument('--episode\_num', type=int, default=50000, help='total episode num during training procedure')
297. parser.add\_argument("--evaluate\_freq", type=float, default=20, help="Evaluate the policy every 'evaluate\_freq' steps")
298. parser.add\_argument("--evaluate\_times", type=float, default=1, help="Evaluate times")
299. parser.add\_argument("--warmup\_steps", default=10000, type=int, help="Warmup steps without training.")
300. parser.add\_argument("--algorithm", type=str, default="MADDPG", help="MATD3 or MATD3\_LSTM or MADDPG")
301. parser.add\_argument("--buffer\_size", type=int, default=int(1e6), help="The capacity of the replay buffer")
302. parser.add\_argument("--batch\_size", type=int, default=1024, help="Batch size")
303. parser.add\_argument("--hidden\_dim", type=int, default=64, help="The number of neurons in hidden layers of the neural network")
304. parser.add\_argument("--noise\_std", type=float, default=0.1, help="The std of Gaussian noise for exploration")
305. parser.add\_argument("--lr\_a", type=float, default=1e-3, help="Learning rate of actor")
306. parser.add\_argument("--lr\_c", type=float, default=1e-3, help="Learning rate of critic")
307. parser.add\_argument("--gamma", type=float, default=0.95, help="Discount factor")
308. parser.add\_argument("--seed", default=14, type=int, help="Random seed.")
309. parser.add\_argument("--tau", type=float, default=0.01, help="Softly update the target network")
310. parser.add\_argument("--use\_orthogonal\_init", type=bool, default=True, help="Orthogonal initialization")
311. parser.add\_argument("--use\_grad\_clip", type=bool, default=True, help="Gradient clip")
312. parser.add\_argument("--output\_dir", default="output", type=str, help="Output directory.")
313. parser.add\_argument("--do\_train", action="store\_true", help="Train policy.")
314. parser.add\_argument("--do\_eval", action="store\_true", help="Evaluate policy.")
315. parser.add\_argument("--add\_model", type=bool, default=False, help="Loading model")
316. args = parser.parse\_args()
317. args.device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
318. args.output\_dir = os.path.join(args.output\_dir, '{}/env\_{}\_seed\_{}'.format(args.algorithm, args.env, args.seed))
319. os.makedirs(args.output\_dir, exist\_ok=True)
320. torch.manual\_seed(args.seed)
321. np.random.seed(args.seed)
322. world\_name = args.env + '.yaml'
323. env = multirobotenv.parallel\_env(render\_mode="rgb\_array", world\_name=world\_name)
324. env\_eval = multirobotenv.parallel\_env(render\_mode="human", world\_name=world\_name)
325. args.num\_agents = len(env.possible\_agents)
326. num\_actions = env.action\_space(env.possible\_agents[0]).shape[0]
327. args.max\_action = env.action\_space(env.possible\_agents[0]).high[0]
328. observation\_size = env.observation\_space(env.possible\_agents[0]).shape[0]
329. agents = {agent\_id:MADDPG(max\_action=args.max\_action, num\_states=observation\_size, num\_actions=num\_actions, args=args, agent\_id=agent\_id) for agent\_id in env.possible\_agents}
330. replay\_buffer = ReplayBuffer(args.num\_agents, observation\_size, num\_actions, args)
332. if args.do\_train:
333. writer = SummaryWriter(log\_dir='runs/{}/env\_{}\_seed\_{}'.format(args.algorithm, args.env, args.seed))
334. writer.add\_text("args", str(args))
335. train(args, env, agents, replay\_buffer, writer)
336. writer.close()
337. if args.do\_eval:
338. eval(args, env\_eval, agents, None)

**11.masac.py:**

1. import argparse
2. import torch
3. import torch.nn as nn
4. import torch.nn.functional as F
5. import numpy as np
6. import multirobotenv
7. from pettingzoo.mpe import simple\_spread\_v3
8. from torch.utils.tensorboard import SummaryWriter
9. from collections import defaultdict
10. import os
11. import matplotlib.pyplot as plt
12. from PIL import Image
13. import pandas as pd
14. from datetime import datetime
15. from tqdm import tqdm
16. def orthogonal\_init(layer, gain=1.0):
17. if type(layer) == nn.Linear:
18. for name, param in layer.named\_parameters():
19. if 'bias' in name:
20. nn.init.constant\_(param, 0)
21. elif 'weight' in name:
22. *# nn.init.orthogonal\_(param, gain=gain)*
23. nn.init.xavier\_uniform\_(param, gain=gain)
24. class QNet(nn.Module):
25. def \_\_init\_\_(self, dim\_state, dim\_action, args):
26. super().\_\_init\_\_()
27. self.fc1 = nn.Linear(args.num\_agents \* (dim\_state + dim\_action), args.hidden\_dim)
28. self.fc2 = nn.Linear(args.hidden\_dim, args.hidden\_dim)
29. self.fc3 = nn.Linear(args.hidden\_dim, 1)
30. if args.use\_orthogonal\_init:
31. orthogonal\_init(self.fc1)
32. orthogonal\_init(self.fc2)
33. orthogonal\_init(self.fc3)
34. def forward(self, state, action):
35. sa = torch.cat(state + action, 1)
36. x = F.relu(self.fc1(sa))
37. x = F.relu(self.fc2(x))
38. x = self.fc3(x)
39. return x
41. class GaussianPolicy(nn.Module):
42. def \_\_init\_\_(self, dim\_state, dim\_action, args):
43. super(GaussianPolicy, self).\_\_init\_\_()
44. self.fc1 = nn.Linear(dim\_state, args.hidden\_dim)
45. self.fc2 = nn.Linear(args.hidden\_dim, args.hidden\_dim)
46. self.fc\_mean = nn.Linear(args.hidden\_dim, dim\_action)
47. self.fc\_logstd = nn.Linear(args.hidden\_dim, dim\_action)
48. self.MIN\_LOG = args.MIN\_LOG
49. self.MAX\_LOG = args.MAX\_LOG
50. self.apply(orthogonal\_init)
51. def forward(self, state):
52. x = F.relu(self.fc1(state))
53. x = F.relu(self.fc2(x))
54. mean = self.fc\_mean(x)
55. log\_std = self.fc\_logstd(x)
56. log\_std = torch.clamp(log\_std, min=self.MIN\_LOG, max=self.MAX\_LOG)
57. return mean, log\_std
58. class ReplayBuffer(object):
59. def \_\_init\_\_(self, num\_agents, obs\_dim\_n, action\_dim\_n, args):
60. self.N = num\_agents  *# The number of agents*
61. self.buffer\_size = args.buffer\_size
62. self.batch\_size = args.batch\_size
63. self.count = 0
64. self.current\_size = 0
65. self.device = args.device
66. self.buffer\_obs\_n = {f'agent\_{i}': np.empty((self.buffer\_size, obs\_dim\_n)) for i in range(self.N)}
67. self.buffer\_a\_n = {f'agent\_{i}': np.empty((self.buffer\_size, action\_dim\_n)) for i in range(self.N)}
68. self.buffer\_r\_n = {f'agent\_{i}': np.empty((self.buffer\_size, 1)) for i in range(self.N)}
69. self.buffer\_s\_next\_n = {f'agent\_{i}': np.empty((self.buffer\_size, obs\_dim\_n)) for i in range(self.N)}
70. self.buffer\_done\_n = {f'agent\_{i}': np.empty((self.buffer\_size, 1)) for i in range(self.N)}
71. def store\_transition(self, obs\_n, a\_n, r\_n, obs\_next\_n, done\_n):
72. for i in range(self.N):
73. agent\_key = f'agent\_{i}'
74. self.buffer\_obs\_n[agent\_key][self.count] = obs\_n[agent\_key]
75. self.buffer\_a\_n[agent\_key][self.count] = a\_n[agent\_key]
76. self.buffer\_r\_n[agent\_key][self.count] = r\_n[agent\_key]
77. self.buffer\_s\_next\_n[agent\_key][self.count] = obs\_next\_n[agent\_key]
78. self.buffer\_done\_n[agent\_key][self.count] = done\_n[agent\_key]
79. self.count = (self.count + 1) % self.buffer\_size
80. self.current\_size = min(self.current\_size + 1, self.buffer\_size)
81. def sample(self, ):
82. index = np.random.choice(self.current\_size, size=self.batch\_size, replace=False)
83. batch\_obs\_n, batch\_a\_n, batch\_r\_n, batch\_obs\_next\_n, batch\_done\_n = {}, {}, {}, {}, {}
84. for i in range(self.N):
85. agent\_key = f'agent\_{i}'
86. batch\_obs\_n[agent\_key] = torch.tensor(self.buffer\_obs\_n[agent\_key][index], dtype=torch.float).to(self.device)
87. batch\_a\_n[agent\_key] = torch.tensor(self.buffer\_a\_n[agent\_key][index], dtype=torch.float).to(self.device)
88. batch\_r\_n[agent\_key] = torch.tensor(self.buffer\_r\_n[agent\_key][index], dtype=torch.float).to(self.device)
89. batch\_obs\_next\_n[agent\_key] = torch.tensor(self.buffer\_s\_next\_n[agent\_key][index], dtype=torch.float).to(self.device)
90. batch\_done\_n[agent\_key] = torch.tensor(self.buffer\_done\_n[agent\_key][index], dtype=torch.float).to(self.device)
91. return batch\_obs\_n, batch\_a\_n, batch\_r\_n, batch\_done\_n, batch\_obs\_next\_n
92. class MASAC(nn.Module):
93. def \_\_init\_\_(
94. self,
95. max\_action=1,
96. num\_states=6,
97. num\_actions=5,
98. args=None,
99. agent\_id=0,
100. ):
101. super().\_\_init\_\_()
102. self.id = agent\_id
103. self.max\_action = max\_action
104. self.num\_states = num\_states
105. self.num\_actions = num\_actions
106. self.device = args.device
107. self.automatic\_entropy\_tuning = args.automatic\_entropy\_tuning
108. self.policy\_update\_freq = args.policy\_update\_freq
109. self.actor\_pointer = 0
110. self.use\_grad\_clip = args.use\_grad\_clip
111. self.alpha = args.alpha
112. *# action rescaling*
113. self.action\_scale = torch.FloatTensor([(args.max\_action - args.min\_action) / 2.]).to(self.device)
114. self.action\_bias = torch.FloatTensor([(args.max\_action + args.min\_action) / 2.]).to(self.device)
115. self.Q1 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
116. self.Q2 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
117. self.Mu = GaussianPolicy(self.num\_states, self.num\_actions, args).to(self.device)
118. self.target\_Q1 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
119. self.target\_Q2 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
120. self.target\_Q1.load\_state\_dict(self.Q1.state\_dict())
121. self.target\_Q2.load\_state\_dict(self.Q2.state\_dict())
122. self.Q1\_optimizer = torch.optim.Adam(self.Q1.parameters(), lr=args.lr\_c)
123. self.Q2\_optimizer = torch.optim.Adam(self.Q2.parameters(), lr=args.lr\_c)
124. self.Mu\_optimizer = torch.optim.Adam(self.Mu.parameters(), lr=args.lr\_a)
125. *# automatic entropy tuning*
126. if self.automatic\_entropy\_tuning:
127. self.target\_entropy = -torch.prod(torch.Tensor((self.num\_actions,)).to(self.device)).item()
128. self.log\_alpha = torch.zeros(1, requires\_grad=True, device=self.device)
129. self.alpha\_optim = torch.optim.Adam([self.log\_alpha], lr=args.alpha\_lr)
130. def sample\_action(self, state):
131. mean, log\_std = self.Mu(state)
132. std = log\_std.exp()
133. normal = torch.distributions.Normal(mean, std)
134. x\_t = normal.rsample()  *# for reparameterization trick (mean + std \* N(0,1))*
135. y\_t = torch.tanh(x\_t)
136. action = y\_t \* self.action\_scale + self.action\_bias
137. log\_prob = normal.log\_prob(x\_t)
138. *# Enforcing Action Bound*
139. log\_prob -= torch.log(self.action\_scale \* (1 - y\_t.pow(2)) + 1e-6)
140. log\_prob = log\_prob.sum(1, keepdim=True)
141. mean = torch.tanh(mean) \* self.action\_scale + self.action\_bias
142. return action, log\_prob, mean
144. def get\_action(self, state, is\_evaluate=False):
145. state = torch.from\_numpy(state).unsqueeze(0).float().to(self.device)
146. if is\_evaluate is False:
147. action, \_, \_ = self.sample\_action(state)
148. else:
149. \_, \_, action = self.sample\_action(state)
150. return action.detach().cpu().numpy()[0]
152. def compute\_value\_loss(self, args, agents, s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch):
153. *# Compute Q*
154. with torch.no\_grad():
155. *# Select the target smoothing regularized action according to policy*
156. batch\_a\_next, batch\_log\_pi\_next = {}, {}
157. for agent\_id, agent in agents.items():
158. action, log\_prob, \_ = agent.sample\_action(ns\_batch[agent\_id])
159. batch\_a\_next[agent\_id] = action
160. batch\_log\_pi\_next[agent\_id] = log\_prob
161. *# Compute the target Q-value*
162. q1 = self.target\_Q1(list(ns\_batch.values()), list(batch\_a\_next.values()))
163. q2 = self.target\_Q2(list(ns\_batch.values()), list(batch\_a\_next.values()))
164. *# 计算 TD 目标。*
165. min\_qf\_next\_target = torch.min(q1, q2) - self.alpha \* batch\_log\_pi\_next[self.id]
166. y = r\_batch[self.id] + args.gamma \* min\_qf\_next\_target \* (1 - d\_batch[self.id])
168. *# Get the current Q-value estimates*
169. qvals1 = self.Q1(list(s\_batch.values()), list(a\_batch.values()))
170. qvals2 = self.Q2(list(s\_batch.values()), list(a\_batch.values()))
171. value\_loss1 = F.mse\_loss(y, qvals1)
172. value\_loss2 = F.mse\_loss(y, qvals2)
173. return value\_loss1, value\_loss2
174. def compute\_policy\_loss(self, s\_batch, a\_batch):
175. a\_batch[self.id], log\_prob, \_ = self.sample\_action(s\_batch[self.id])
176. q1\_pi = self.Q1(list(s\_batch.values()), list(a\_batch.values()))
177. q2\_pi = self.Q1(list(s\_batch.values()), list(a\_batch.values()))
178. min\_q\_pi = torch.min(q1\_pi,q2\_pi)
179. policy\_loss = ((self.alpha \* log\_prob) - min\_q\_pi).mean()
180. return policy\_loss, log\_prob
181. def soft\_update(self, tau=0.01):
182. def soft\_update\_(target, source, tau\_=0.01):
183. for target\_param, param in zip(target.parameters(), source.parameters()):
184. target\_param.data.copy\_(target\_param.data \* (1.0 - tau\_) + param.data \* tau\_)
185. soft\_update\_(self.target\_Q1, self.Q1, tau)
186. soft\_update\_(self.target\_Q2, self.Q2, tau)
187. def save\_model(self, output\_dir):
188. save\_path = os.path.join(output\_dir, f"{self.id}\_model.bin")
189. torch.save(self.Mu.state\_dict(), save\_path)
190. def load\_model(self, output\_dir):
191. save\_path = os.path.join(output\_dir, f"{self.id}\_model.bin")
192. self.Mu.load\_state\_dict(torch.load(save\_path))
193. def learn(self, replay\_buffer: ReplayBuffer, args, agents: dict, infos: dict):
194. self.actor\_pointer += 1
195. s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch = replay\_buffer.sample()
196. value\_loss1, value\_loss2 = self.compute\_value\_loss(args, agents, s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch)
197. infos[self.id].log["value\_loss1"].append(value\_loss1.item())
198. infos[self.id].log["value\_loss2"].append(value\_loss2.item())
199. self.Q1\_optimizer.zero\_grad()
200. value\_loss1.backward(retain\_graph=True)
201. if self.use\_grad\_clip:
202. torch.nn.utils.clip\_grad\_norm\_(self.Q1.parameters(), 10.0)
203. self.Q1\_optimizer.step()
204. self.Q2\_optimizer.zero\_grad()
205. value\_loss2.backward()
206. if self.use\_grad\_clip:
207. torch.nn.utils.clip\_grad\_norm\_(self.Q2.parameters(), 10.0)
208. self.Q2\_optimizer.step()
209. *# TD 3 Delayed update support*
210. if self.actor\_pointer % self.policy\_update\_freq == 0:
211. *# Compute policy loss*
212. policy\_loss, log\_prob = self.compute\_policy\_loss(s\_batch, a\_batch)
213. *# Optimize the actor*
214. self.Mu\_optimizer.zero\_grad()
215. policy\_loss.backward()
216. if self.use\_grad\_clip:
217. torch.nn.utils.clip\_grad\_norm\_(self.Mu.parameters(), 10.0)
218. self.Mu\_optimizer.step()
219. infos[self.id].log["policy\_loss"].append(policy\_loss.item())
220. *# Tune the temperature coefficient*
221. if self.automatic\_entropy\_tuning:
222. alpha\_loss = -(self.log\_alpha \* (log\_prob + self.target\_entropy).detach()).mean()
223. self.alpha\_optim.zero\_grad()
224. alpha\_loss.backward()
225. self.alpha\_optim.step()
226. self.alpha = self.log\_alpha.exp()
227. self.soft\_update(args.tau)
228. class INFO:
229. def \_\_init\_\_(self):
230. self.log = defaultdict(list)
231. self.episode\_length = 0
232. self.episode\_reward = 0
233. self.max\_episode\_reward = -float("inf")
234. def put(self, done, reward, detector):
235. if done is True:
236. self.episode\_length += 1
237. self.episode\_reward += reward
238. self.log["episode\_length"].append(self.episode\_length)
239. self.log["episode\_reward"].append(self.episode\_reward)
240. if self.episode\_reward > self.max\_episode\_reward:
241. self.max\_episode\_reward = self.episode\_reward
242. self.episode\_length = 0
243. self.episode\_reward = 0
244. else:
245. self.episode\_length += 1
246. self.episode\_reward += reward
247. self.log["detector"].append(detector)
249. def reset(self):
250. self.log = defaultdict(list)
251. def train(args, env, agents: dict, replay\_buffer: ReplayBuffer, writer: SummaryWriter):
252. if args.add\_model:
253. for agent in agents.values():
254. agent.load\_model(args.output\_dir)
255. step = 0  *# global step counter*
256. max\_reward = 300
257. sum\_reward = 0
258. episode\_data = []
259. infos = {agent\_id: INFO() for agent\_id in env.possible\_agents}
260. for episode in tqdm(range(args.episode\_num), desc="Training Episodes"):
261. state, \_, detector = env.reset()
262. while env.agents:
263. step += 1
264. if step < args.warmup\_steps:
265. *# action = {agent: env.components['robots'].robot\_list[i].cal\_des\_vel().reshape(2,) for i, agent in enumerate(env.agents)}*
266. action = {agent\_id: env.action\_space(agent\_id).sample() for agent\_id in env.agents}
267. else:
268. action = {agent\_id:agents[agent\_id].get\_action(state[agent\_id]) for agent\_id in env.agents}
269. next\_state, reward, terminated, truncated, \_ , detector= env.step(action)
270. done = {agent: terminated[agent] or truncated[agent] for agent in terminated}
271. replay\_buffer.store\_transition(state, action, reward, next\_state, done)
272. for agent\_id in env.possible\_agents:
273. infos[agent\_id].put(done[agent\_id], reward[agent\_id],detector[agent\_id])
274. state = next\_state
275. if step >= args.warmup\_steps:
276. for agent\_id in env.possible\_agents:
277. agents[agent\_id].learn(replay\_buffer, args, agents, infos)
278. *# episode finishes*
279. if (episode + 1) % 20 == 0:  *# print info every 20 episodes*
280. sum\_reward = 0
281. current\_time = datetime.now()
282. for agent\_id, info in infos.items():  *# record reward*
283. avg\_reward = np.mean(info.log["episode\_reward"][-20:])
284. value\_loss1 = np.mean(info.log["value\_loss1"][-20:]) if len(info.log["value\_loss1"]) > 0 else 0
285. value\_loss2 = np.mean(info.log["value\_loss2"][-20:]) if len(info.log["value\_loss2"]) > 0 else 0
286. policy\_loss = np.mean(info.log["policy\_loss"][-20:]) if len(info.log["policy\_loss"]) > 0 else 0
287. episode\_data.append({"time": current\_time.strftime("%Y-%m-%d %H:%M:%S"),"episode": episode + 1,"agent\_id": agent\_id, "reward": avg\_reward, "value\_loss1": value\_loss1, "value\_loss2": value\_loss2, "policy\_loss": policy\_loss})
288. *# print(f"episode={episode+1:d}, {agent\_id} reward={avg\_reward:.2f}, value loss1={value\_loss1:.4f}, value loss2={value\_loss2:.4f}, policy loss={policy\_loss:.4f}")*
289. sum\_reward +=avg\_reward
290. infos[agent\_id].reset()
291. writer.add\_scalar('train\_step\_rewards:',sum\_reward, global\_step=episode)
292. writer.add\_scalar(f'{agent\_id}\'s value\_loss1:',value\_loss1, global\_step=episode)
293. writer.add\_scalar(f'{agent\_id}\'s value\_loss2:',value\_loss2, global\_step=episode)
294. writer.add\_scalar(f'{agent\_id}\'s policy\_loss:',policy\_loss, global\_step=episode)
296. *# 将信息保存到DataFrame*
297. if (episode + 1) % 100 == 0:
298. df = pd.DataFrame(episode\_data)
299. csv\_file\_path = f"{args.output\_dir}/episode\_info.csv"
300. df.to\_csv(csv\_file\_path, mode='a', header=not os.path.exists(csv\_file\_path), index=False)
301. episode\_data.clear()
303. if sum\_reward > max\_reward:
304. max\_reward = sum\_reward
305. for agent\_id in env.possible\_agents:
306. agents[agent\_id].save\_model(args.output\_dir)
308. def eval(args, env, agents: dict, writer: SummaryWriter):
309. episode\_data = []
310. detector\_info = []
311. infos = {agent\_id: INFO() for agent\_id in env.possible\_agents}
312. gif\_dir = os.path.join(args.output\_dir, 'gif' + str(args.seed))
313. os.makedirs(gif\_dir, exist\_ok=True)
314. gif\_num = len([file for file in os.listdir(gif\_dir)])  *# current number of gif*
315. for agent in agents.values():
316. agent.load\_model(args.output\_dir)
317. for episode in tqdm(range(args.evaluate\_times), desc="Evaluating Episodes"):
318. state, \_, detector = env.reset()
319. for agent\_id in env.possible\_agents:
320. infos[agent\_id].put(False, 0, detector[agent\_id])
321. frame\_list = []  *# used to save gif*
322. while env.agents:
323. action = {agent\_id:agents[agent\_id].get\_action(state[agent\_id], is\_evaluate=True) for agent\_id in env.agents}
324. next\_state, reward, terminated, truncated, \_, detector = env.step(action)
325. if env.render\_mode == "rgb\_array" and args.evaluate\_times < 30:
326. frame\_list.append(Image.fromarray(env.render()))
327. done = {agent: terminated[agent] or truncated[agent] for agent in terminated}
328. for agent\_id in env.possible\_agents:
329. infos[agent\_id].put(done[agent\_id], reward[agent\_id], detector[agent\_id])
330. state = next\_state
331. *# episode finishes*
332. sum\_reward = 0
333. current\_time = datetime.now()
334. for i in range(env.steps):
335. detector\_info.append({"steps": i+1, "agent\_0":infos["agent\_0"].log["detector"][i], "agent\_1":infos["agent\_1"].log["detector"][i], "agent\_2":infos["agent\_2"].log["detector"][i]})
336. detector\_info.append({"steps": "avg\_reward", "agent\_0":infos["agent\_0"].log["episode\_reward"][-1], "agent\_1":infos["agent\_1"].log["episode\_reward"][-1], "agent\_2":infos["agent\_2"].log["episode\_reward"][-1]})
337. for agent\_id, info in infos.items():  *# record reward*
338. avg\_reward = info.log["episode\_reward"][-1]
339. *# print(f"episode={episode+1:d} steps={env.steps}, {agent\_id} reward={avg\_reward:.2f}")*
340. sum\_reward += avg\_reward
341. episode\_data.append({"time": current\_time.strftime("%Y-%m-%d %H:%M:%S"),"episode":episode+1, "steps": env.steps, "sum\_reward": sum\_reward})
342. if env.render\_mode == "rgb\_array" and args.evaluate\_times < 30:
343. frame\_list[0].save(os.path.join(gif\_dir, f'out{gif\_num + episode + 1}.gif'),
344. save\_all=True, append\_images=frame\_list[1:], duration=1, loop=0)
345. *# writer.add\_scalar('evaluate\_step\_rewards:',sum\_reward, global\_step=episode)*
346. *# 将信息保存到DataFrame*
347. df = pd.DataFrame(detector\_info)
348. csv\_file\_path = f"{args.output\_dir}/detector\_info.csv"
349. df.to\_csv(csv\_file\_path, mode='a', header=not os.path.exists(csv\_file\_path), index=False)
350. detector\_info.clear()
351. if \_\_name\_\_ == '\_\_main\_\_':
352. parser = argparse.ArgumentParser("Hyperparameters Setting for MADDPG and MATD3 in MPE environment")
353. parser.add\_argument("--env", default="multi\_robot\_world\_10", type=str, help="Environment name.")
354. parser.add\_argument("--max\_action", default=1.0, type=float, help="Action scale, [-max, max].")
355. parser.add\_argument('--episode\_num', type=int, default=50000, help='total episode num during training procedure')
356. parser.add\_argument("--evaluate\_freq", type=float, default=20, help="Evaluate the policy every 'evaluate\_freq' steps")
357. parser.add\_argument("--evaluate\_times", type=float, default=1, help="Evaluate times")
358. parser.add\_argument("--warmup\_steps", default=10000, type=int, help="Warmup steps without training.")
359. parser.add\_argument("--algorithm", type=str, default="MASAC", help="MATD3 or MATD3\_LSTM or MADDPG or MASAC")
360. parser.add\_argument("--buffer\_size", type=int, default=int(1e6), help="The capacity of the replay buffer")
361. parser.add\_argument("--batch\_size", type=int, default=1024, help="Batch size")
362. parser.add\_argument("--hidden\_dim", type=int, default=64, help="The number of neurons in hidden layers of the neural network")
363. parser.add\_argument("--lr\_a", type=float, default=1e-3, help="Learning rate of actor")
364. parser.add\_argument("--lr\_c", type=float, default=1e-3, help="Learning rate of critic")
365. parser.add\_argument("--gamma", type=float, default=0.95, help="Discount factor")
366. parser.add\_argument("--seed", default=15, type=int, help="Random seed.")
367. parser.add\_argument("--tau", type=float, default=0.01, help="Softly update the target network")
368. parser.add\_argument("--use\_orthogonal\_init", type=bool, default=True, help="Orthogonal initialization")
369. parser.add\_argument("--use\_grad\_clip", type=bool, default=True, help="Gradient clip")
370. parser.add\_argument("--output\_dir", default="output", type=str, help="Output directory.")
371. parser.add\_argument("--do\_train", action="store\_true", help="Train policy.")
372. parser.add\_argument("--do\_eval", action="store\_true", help="Evaluate policy.")
373. parser.add\_argument("--add\_model", type=bool, default=False, help="Loading model")
374. *# --------------------------------------SAC--------------------------------------------------------------------*
375. parser.add\_argument('--alpha', type=float, default=0.2, help='Entropy regularization coefficient (controls the trade-off between exploration and exploitation)')
376. parser.add\_argument('--alpha\_lr', type=float, default=3e-4)
377. parser.add\_argument('--MIN\_LOG', type=float, default=-20, help='Minimum log value')
378. parser.add\_argument('--MAX\_LOG', type=float, default=2, help='Maximum log value')
379. parser.add\_argument('--automatic\_entropy\_tuning', type=bool, default=True, help='Enable automatic entropy tuning')
380. parser.add\_argument("--policy\_update\_freq", type=int, default=2, help="The frequency of policy updates")
381. args = parser.parse\_args()
382. args.device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
383. args.output\_dir = os.path.join(args.output\_dir, '{}/env\_{}\_seed\_{}'.format(args.algorithm, args.env, args.seed))
384. os.makedirs(args.output\_dir, exist\_ok=True)
385. torch.manual\_seed(args.seed)
386. np.random.seed(args.seed)
387. world\_name = args.env + '.yaml'
388. env = multirobotenv.parallel\_env(render\_mode="rgb\_array", world\_name=world\_name)
389. env\_eval = multirobotenv.parallel\_env(render\_mode="rgb\_array", world\_name=world\_name)
390. args.num\_agents = len(env.possible\_agents)
391. num\_actions = env.action\_space(env.possible\_agents[0]).shape[0]
392. args.max\_action = env.action\_space(env.possible\_agents[0]).high[0]
393. args.min\_action = env.action\_space(env.possible\_agents[0]).low[0]
394. observation\_size = env.observation\_space(env.possible\_agents[0]).shape[0]
395. agents = {agent\_id:MASAC(max\_action=args.max\_action, num\_states=observation\_size, num\_actions=num\_actions, args=args, agent\_id=agent\_id) for agent\_id in env.possible\_agents}
396. replay\_buffer = ReplayBuffer(args.num\_agents, observation\_size, num\_actions, args)
398. if args.do\_train:
399. writer = SummaryWriter(log\_dir='runs/{}/env\_{}\_seed\_{}'.format(args.algorithm, args.env, args.seed))
400. writer.add\_text("args", str(args))
401. train(args, env, agents, replay\_buffer, writer)
402. writer.close()
403. if args.do\_eval:
404. eval(args, env\_eval, agents, None)

**11.matd3.py:**

1. import argparse
2. import torch
3. import torch.nn as nn
4. import torch.nn.functional as F
5. import numpy as np
6. import multirobotenv
7. from pettingzoo.mpe import simple\_spread\_v3
8. from torch.utils.tensorboard import SummaryWriter
9. from collections import defaultdict
10. import os
11. import matplotlib.pyplot as plt
12. from PIL import Image
13. import pandas as pd
14. from datetime import datetime
15. from tqdm import tqdm
16. def orthogonal\_init(layer, gain=1.0):
17. for name, param in layer.named\_parameters():
18. if 'bias' in name:
19. nn.init.constant\_(param, 0)
20. elif 'weight' in name:
21. nn.init.orthogonal\_(param, gain=gain)
22. class QNet(nn.Module):
23. def \_\_init\_\_(self, dim\_state, dim\_action, args):
24. super().\_\_init\_\_()
25. self.fc1 = nn.Linear(args.num\_agents \* (dim\_state + dim\_action), args.hidden\_dim)
26. self.fc2 = nn.Linear(args.hidden\_dim, args.hidden\_dim)
27. self.fc3 = nn.Linear(args.hidden\_dim, 1)
28. if args.use\_orthogonal\_init:
29. orthogonal\_init(self.fc1)
30. orthogonal\_init(self.fc2)
31. orthogonal\_init(self.fc3)
32. def forward(self, state, action):
33. sa = torch.cat(state + action, 1)
34. x = F.relu(self.fc1(sa))
35. x = F.relu(self.fc2(x))
36. x = self.fc3(x)
37. return x
38. class PolicyNet(nn.Module):
39. def \_\_init\_\_(self, dim\_state, dim\_action, args):
40. super().\_\_init\_\_()
41. self.max\_action = args.max\_action
42. self.fc1 = nn.Linear(dim\_state, args.hidden\_dim)
43. self.fc2 = nn.Linear(args.hidden\_dim, args.hidden\_dim)
44. self.fc3 = nn.Linear(args.hidden\_dim, dim\_action)
45. if args.use\_orthogonal\_init:
46. orthogonal\_init(self.fc1)
47. orthogonal\_init(self.fc2)
48. orthogonal\_init(self.fc3)
49. def forward(self, state):
50. x = F.relu(self.fc1(state))
51. x = F.relu(self.fc2(x))
52. x = self.max\_action \* torch.tanh(self.fc3(x))
53. return x
55. class ReplayBuffer(object):
56. def \_\_init\_\_(self, num\_agents, obs\_dim\_n, action\_dim\_n, args):
57. self.N = num\_agents  *# The number of agents*
58. self.buffer\_size = args.buffer\_size
59. self.batch\_size = args.batch\_size
60. self.count = 0
61. self.current\_size = 0
62. self.device = args.device
63. self.buffer\_obs\_n = {f'agent\_{i}': np.empty((self.buffer\_size, obs\_dim\_n)) for i in range(self.N)}
64. self.buffer\_a\_n = {f'agent\_{i}': np.empty((self.buffer\_size, action\_dim\_n)) for i in range(self.N)}
65. self.buffer\_r\_n = {f'agent\_{i}': np.empty((self.buffer\_size, 1)) for i in range(self.N)}
66. self.buffer\_s\_next\_n = {f'agent\_{i}': np.empty((self.buffer\_size, obs\_dim\_n)) for i in range(self.N)}
67. self.buffer\_done\_n = {f'agent\_{i}': np.empty((self.buffer\_size, 1)) for i in range(self.N)}
68. def store\_transition(self, obs\_n, a\_n, r\_n, obs\_next\_n, done\_n):
69. for i in range(self.N):
70. agent\_key = f'agent\_{i}'
71. self.buffer\_obs\_n[agent\_key][self.count] = obs\_n[agent\_key]
72. self.buffer\_a\_n[agent\_key][self.count] = a\_n[agent\_key]
73. self.buffer\_r\_n[agent\_key][self.count] = r\_n[agent\_key]
74. self.buffer\_s\_next\_n[agent\_key][self.count] = obs\_next\_n[agent\_key]
75. self.buffer\_done\_n[agent\_key][self.count] = done\_n[agent\_key]
76. self.count = (self.count + 1) % self.buffer\_size
77. self.current\_size = min(self.current\_size + 1, self.buffer\_size)
78. def sample(self, ):
79. index = np.random.choice(self.current\_size, size=self.batch\_size, replace=False)
80. batch\_obs\_n, batch\_a\_n, batch\_r\_n, batch\_obs\_next\_n, batch\_done\_n = {}, {}, {}, {}, {}
81. for i in range(self.N):
82. agent\_key = f'agent\_{i}'
83. batch\_obs\_n[agent\_key] = torch.tensor(self.buffer\_obs\_n[agent\_key][index], dtype=torch.float).to(self.device)
84. batch\_a\_n[agent\_key] = torch.tensor(self.buffer\_a\_n[agent\_key][index], dtype=torch.float).to(self.device)
85. batch\_r\_n[agent\_key] = torch.tensor(self.buffer\_r\_n[agent\_key][index], dtype=torch.float).to(self.device)
86. batch\_obs\_next\_n[agent\_key] = torch.tensor(self.buffer\_s\_next\_n[agent\_key][index], dtype=torch.float).to(self.device)
87. batch\_done\_n[agent\_key] = torch.tensor(self.buffer\_done\_n[agent\_key][index], dtype=torch.float).to(self.device)
88. return batch\_obs\_n, batch\_a\_n, batch\_r\_n, batch\_done\_n, batch\_obs\_next\_n
89. class MATD3(nn.Module):
90. def \_\_init\_\_(
91. self,
92. max\_action=1,
93. num\_states=6,
94. num\_actions=5,
95. args=None,
96. agent\_id=0,
97. ):
98. super().\_\_init\_\_()
99. self.id = agent\_id
100. self.max\_action = max\_action
101. self.num\_states = num\_states
102. self.num\_actions = num\_actions
103. self.device = args.device
104. self.policy\_update\_freq = args.policy\_update\_freq
105. self.actor\_pointer = 0
106. self.use\_grad\_clip = args.use\_grad\_clip
107. self.Q1 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
108. self.Q2 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
109. self.Mu = PolicyNet(self.num\_states, self.num\_actions, args).to(self.device)
110. self.target\_Q1 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
111. self.target\_Q2 = QNet(self.num\_states, self.num\_actions, args).to(self.device)
112. self.target\_Mu = PolicyNet(self.num\_states, self.num\_actions, args).to(self.device)
113. self.target\_Q1.load\_state\_dict(self.Q1.state\_dict())
114. self.target\_Q2.load\_state\_dict(self.Q2.state\_dict())
115. self.target\_Mu.load\_state\_dict(self.Mu.state\_dict())
116. self.Q1\_optimizer = torch.optim.Adam(self.Q1.parameters(), lr=args.lr\_c)
117. self.Q2\_optimizer = torch.optim.Adam(self.Q2.parameters(), lr=args.lr\_c)
118. self.Mu\_optimizer = torch.optim.Adam(self.Mu.parameters(), lr=args.lr\_a)
119. def get\_action(self, state, noise\_std=0):
120. state = torch.from\_numpy(state).unsqueeze(0).float().to(self.device)
121. action = self.Mu(state).cpu().data.numpy().flatten()
122. noise =  np.random.normal(0, noise\_std, size=self.num\_actions)
123. action = (action + noise).clip(-self.max\_action, self.max\_action)
124. return action.astype(np.float32)
125. def compute\_value\_loss(self, args, agents, s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch):
126. *# Compute target\_Q*
127. with torch.no\_grad():
128. *# 让目标策略网络做预测。*
129. batch\_a\_next = {}
130. for agent\_id, agent in agents.items():
131. a = agent.target\_Mu(ns\_batch[agent\_id])
132. noise = torch.clamp(torch.randn\_like(a) \* args.policy\_noise, -args.noise\_clip, args.noise\_clip)
133. a = torch.clamp(a + noise, min=-self.max\_action, max=self.max\_action)
134. *# a = torch.clamp(a + noise, min=0, max=self.max\_action)*
135. batch\_a\_next[agent\_id] = a
136. *# 让两个目标价值网络做预测。*
137. q1 = self.target\_Q1(list(ns\_batch.values()), list(batch\_a\_next.values()))
138. q2 = self.target\_Q2(list(ns\_batch.values()), list(batch\_a\_next.values()))
139. *# 计算 TD 目标。*
140. y = r\_batch[self.id] + args.gamma \* torch.min(q1, q2) \* (1 - d\_batch[self.id])
141. *# 让两个价值网络做预测。*
142. qvals1 = self.Q1(list(s\_batch.values()), list(a\_batch.values()))
143. qvals2 = self.Q2(list(s\_batch.values()), list(a\_batch.values()))
144. value\_loss1 = F.mse\_loss(y, qvals1)
145. value\_loss2 = F.mse\_loss(y, qvals2)
146. return value\_loss1, value\_loss2
147. def compute\_policy\_loss(self, s\_batch, a\_batch):
148. a\_batch[self.id] = self.Mu(s\_batch[self.id])
149. policy\_loss = -self.Q1(list(s\_batch.values()), list(a\_batch.values())).mean()
150. return policy\_loss
151. def soft\_update(self, tau=0.01):
152. def soft\_update\_(target, source, tau\_=0.01):
153. for target\_param, param in zip(target.parameters(), source.parameters()):
154. target\_param.data.copy\_(target\_param.data \* (1.0 - tau\_) + param.data \* tau\_)
155. soft\_update\_(self.target\_Q1, self.Q1, tau)
156. soft\_update\_(self.target\_Q2, self.Q2, tau)
157. soft\_update\_(self.target\_Mu, self.Mu, tau)
158. def save\_model(self, output\_dir):
159. save\_path = os.path.join(output\_dir, f"{self.id}\_model.bin")
160. torch.save(self.Mu.state\_dict(), save\_path)
161. def load\_model(self, output\_dir):
162. save\_path = os.path.join(output\_dir, f"{self.id}\_model.bin")
163. self.Mu.load\_state\_dict(torch.load(save\_path))
164. def learn(self, replay\_buffer: ReplayBuffer, args, agents: dict, infos: dict):
165. self.actor\_pointer += 1
166. s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch = replay\_buffer.sample()
167. value\_loss1, value\_loss2 = self.compute\_value\_loss(args, agents, s\_batch, a\_batch, r\_batch, d\_batch, ns\_batch)
168. infos[self.id].log["value\_loss1"].append(value\_loss1.item())
169. infos[self.id].log["value\_loss2"].append(value\_loss2.item())
170. self.Q1\_optimizer.zero\_grad()
171. value\_loss1.backward(retain\_graph=True)
172. if self.use\_grad\_clip:
173. torch.nn.utils.clip\_grad\_norm\_(self.Q1.parameters(), 10.0)
174. self.Q1\_optimizer.step()
175. self.Q2\_optimizer.zero\_grad()
176. value\_loss2.backward()
177. if self.use\_grad\_clip:
178. torch.nn.utils.clip\_grad\_norm\_(self.Q2.parameters(), 10.0)
179. self.Q2\_optimizer.step()
180. *# Trick 3:delayed policy updates*
181. if self.actor\_pointer % self.policy\_update\_freq == 0:
182. policy\_loss = self.compute\_policy\_loss(s\_batch, a\_batch)
183. *# Optimize the actor*
184. self.Mu\_optimizer.zero\_grad()
185. policy\_loss.backward()
186. if self.use\_grad\_clip:
187. torch.nn.utils.clip\_grad\_norm\_(self.Mu.parameters(), 10.0)
188. self.Mu\_optimizer.step()
189. self.soft\_update(args.tau)
190. infos[self.id].log["policy\_loss"].append(policy\_loss.item())
191. class INFO:
192. def \_\_init\_\_(self):
193. self.log = defaultdict(list)
194. self.episode\_length = 0
195. self.episode\_reward = 0
196. self.max\_episode\_reward = -float("inf")
197. def put(self, done, reward, detector):
198. if done is True:
199. self.episode\_length += 1
200. self.episode\_reward += reward
201. self.log["episode\_length"].append(self.episode\_length)
202. self.log["episode\_reward"].append(self.episode\_reward)
203. if self.episode\_reward > self.max\_episode\_reward:
204. self.max\_episode\_reward = self.episode\_reward
205. self.episode\_length = 0
206. self.episode\_reward = 0
207. else:
208. self.episode\_length += 1
209. self.episode\_reward += reward
210. self.log["detector"].append(detector)
212. def reset(self):
213. self.log = defaultdict(list)
214. def train(args, env, agents: dict, replay\_buffer: ReplayBuffer, writer: SummaryWriter):
215. if args.add\_model:
216. for agent in agents.values():
217. agent.load\_model(args.output\_dir)
218. step = 0  *# global step counter*
219. max\_reward = 300
220. sum\_reward = 0
221. episode\_data = []
222. infos = {agent\_id: INFO() for agent\_id in env.possible\_agents}
223. for episode in tqdm(range(args.episode\_num), desc="Training Episodes"):
224. state, \_, detector = env.reset()
225. while env.agents:
226. step += 1
227. if step < args.warmup\_steps:
228. *# action = {agent: env.components['robots'].robot\_list[i].cal\_des\_vel().reshape(2,) for i, agent in enumerate(env.agents)}*
229. action = {agent\_id: env.action\_space(agent\_id).sample() for agent\_id in env.agents}
230. else:
231. action = {agent\_id:agents[agent\_id].get\_action(state[agent\_id], noise\_std=args.noise\_std) for agent\_id in env.agents}
232. next\_state, reward, terminated, truncated, \_, detector = env.step(action)
233. done = {agent: terminated[agent] or truncated[agent] for agent in terminated}
234. replay\_buffer.store\_transition(state, action, reward, next\_state, done)
235. for agent\_id in env.possible\_agents:
236. infos[agent\_id].put(done[agent\_id], reward[agent\_id], detector[agent\_id])
237. state = next\_state
238. if step >= args.warmup\_steps:
239. for agent\_id in env.possible\_agents:
240. agents[agent\_id].learn(replay\_buffer, args, agents, infos)
241. *# episode finishes*
242. if (episode + 1) % 20 == 0:  *# print info every 20 episodes*
243. sum\_reward = 0
244. current\_time = datetime.now()
245. for agent\_id, info in infos.items():  *# record reward*
246. avg\_reward = np.mean(info.log["episode\_reward"][-20:])
247. value\_loss1 = np.mean(info.log["value\_loss1"][-20:]) if len(info.log["value\_loss1"]) > 0 else 0
248. value\_loss2 = np.mean(info.log["value\_loss2"][-20:]) if len(info.log["value\_loss2"]) > 0 else 0
249. policy\_loss = np.mean(info.log["policy\_loss"][-20:]) if len(info.log["policy\_loss"]) > 0 else 0
250. episode\_data.append({"time": current\_time.strftime("%Y-%m-%d %H:%M:%S"),"episode": episode + 1,"agent\_id": agent\_id, "reward": avg\_reward, "value\_loss1": value\_loss1, "value\_loss2": value\_loss2, "policy\_loss": policy\_loss})
251. *# print(f"episode={episode+1:d}, {agent\_id} reward={avg\_reward:.2f}, value loss1={value\_loss1:.4f}, value loss2={value\_loss2:.4f}, policy loss={policy\_loss:.4f}")*
252. sum\_reward +=avg\_reward
253. infos[agent\_id].reset()
254. writer.add\_scalar('train\_step\_rewards:',sum\_reward, global\_step=episode)
255. writer.add\_scalar(f'{agent\_id}\'s value\_loss1:',value\_loss1, global\_step=episode)
256. writer.add\_scalar(f'{agent\_id}\'s value\_loss2:',value\_loss2, global\_step=episode)
257. writer.add\_scalar(f'{agent\_id}\'s policy\_loss:',policy\_loss, global\_step=episode)
259. *# 将信息保存到DataFrame*
260. if (episode + 1) % 100 == 0:
261. df = pd.DataFrame(episode\_data)
262. csv\_file\_path = f"{args.output\_dir}/episode\_info.csv"
263. df.to\_csv(csv\_file\_path, mode='a', header=not os.path.exists(csv\_file\_path), index=False)
264. episode\_data.clear()
266. if sum\_reward > max\_reward:
267. max\_reward = sum\_reward
268. for agent\_id in env.possible\_agents:
269. agents[agent\_id].save\_model(args.output\_dir)
271. def eval(args, env, agents: dict):
272. episode\_data = []
273. detector\_info = []
274. infos = {agent\_id: INFO() for agent\_id in env.possible\_agents}
275. gif\_dir = os.path.join(args.output\_dir, 'gif' + str(args.seed))
276. os.makedirs(gif\_dir, exist\_ok=True)
277. gif\_num = len([file for file in os.listdir(gif\_dir)])  *# current number of gif*
278. for agent in agents.values():
279. agent.load\_model(args.output\_dir)
280. for episode in tqdm(range(args.evaluate\_times), desc="Evaluating Episodes"):
281. state, \_, detector = env.reset()
282. for agent\_id in env.possible\_agents:
283. infos[agent\_id].put(False, 0, detector[agent\_id])
284. frame\_list = []  *# used to save gif*
285. while env.agents:
286. action = {agent\_id:agents[agent\_id].get\_action(state[agent\_id], noise\_std=0) for agent\_id in env.agents}
287. next\_state, reward, terminated, truncated, \_, detector = env.step(action)
288. if env.render\_mode == "rgb\_array" and args.evaluate\_times < 30:
289. frame\_list.append(Image.fromarray(env.render()))
290. done = {agent: terminated[agent] or truncated[agent] for agent in terminated}
291. for agent\_id in env.possible\_agents:
292. infos[agent\_id].put(done[agent\_id], reward[agent\_id], detector[agent\_id])
293. state = next\_state
294. *# episode finishes*
295. sum\_reward = 0
296. current\_time = datetime.now()
297. for i in range(env.steps):
298. detector\_info.append({"steps": i+1, "agent\_0":infos["agent\_0"].log["detector"][i], "agent\_1":infos["agent\_1"].log["detector"][i], "agent\_2":infos["agent\_2"].log["detector"][i]})
299. detector\_info.append({"steps": "avg\_reward", "agent\_0":infos["agent\_0"].log["episode\_reward"][-1], "agent\_1":infos["agent\_1"].log["episode\_reward"][-1], "agent\_2":infos["agent\_2"].log["episode\_reward"][-1]})
300. for agent\_id, info in infos.items():  *# record reward*
301. avg\_reward = info.log["episode\_reward"][-1]
302. detector = info.log["detector"]
303. sum\_reward += avg\_reward
304. *# episode\_data.append({"time": current\_time.strftime("%Y-%m-%d %H:%M:%S"),"episode":episode+1, "steps": env.steps, "sum\_reward": sum\_reward})*
305. if env.render\_mode == "rgb\_array" and args.evaluate\_times < 30:
306. frame\_list[0].save(os.path.join(gif\_dir, f'out{gif\_num + episode + 1}.gif'),
307. save\_all=True, append\_images=frame\_list[1:], duration=1, loop=0)
308. *# writer.add\_scalar('evaluate\_step\_rewards:',sum\_reward, global\_step=episode)*
309. *# 将信息保存到DataFrame*
310. df = pd.DataFrame(detector\_info)
311. csv\_file\_path = f"{args.output\_dir}/detector\_info.csv"
312. df.to\_csv(csv\_file\_path, mode='a', header=not os.path.exists(csv\_file\_path), index=False)
313. detector\_info.clear()
314. if \_\_name\_\_ == '\_\_main\_\_':
315. parser = argparse.ArgumentParser("Hyperparameters Setting for MADDPG and MATD3 in MPE environment")
316. parser.add\_argument("--env", default="multi\_robot\_world\_10", type=str, help="Environment name.")
317. parser.add\_argument("--max\_action", default=1.0, type=float, help="Action scale, [-max, max].")
318. parser.add\_argument('--episode\_num', type=int, default=50000, help='total episode num during training procedure')
319. parser.add\_argument("--evaluate\_freq", type=float, default=20, help="Evaluate the policy every 'evaluate\_freq' steps")
320. parser.add\_argument("--evaluate\_times", type=float, default=1, help="Evaluate times")
321. parser.add\_argument("--warmup\_steps", default=10000, type=int, help="Warmup steps without training.")
322. parser.add\_argument("--algorithm", type=str, default="MATD3", help="MATD3 or MATD3\_LSTM")
323. parser.add\_argument("--buffer\_size", type=int, default=int(1e6), help="The capacity of the replay buffer")
324. parser.add\_argument("--batch\_size", type=int, default=1024, help="Batch size")
325. parser.add\_argument("--hidden\_dim", type=int, default=64, help="The number of neurons in hidden layers of the neural network")
326. parser.add\_argument("--noise\_std", type=float, default=0.1, help="The std of Gaussian noise for exploration")
327. parser.add\_argument("--lr\_a", type=float, default=1e-3, help="Learning rate of actor")
328. parser.add\_argument("--lr\_c", type=float, default=1e-3, help="Learning rate of critic")
329. parser.add\_argument("--gamma", type=float, default=0.95, help="Discount factor")
330. parser.add\_argument("--seed", default=13, type=int, help="Random seed.")
331. parser.add\_argument("--tau", type=float, default=0.01, help="Softly update the target network")
332. parser.add\_argument("--use\_orthogonal\_init", type=bool, default=True, help="Orthogonal initialization")
333. parser.add\_argument("--use\_grad\_clip", type=bool, default=True, help="Gradient clip")
334. parser.add\_argument("--output\_dir", default="output", type=str, help="Output directory.")
335. parser.add\_argument("--do\_train", action="store\_true", help="Train policy.")
336. parser.add\_argument("--do\_eval", action="store\_true", help="Evaluate policy.")
337. parser.add\_argument("--add\_model", type=bool, default=True, help="Loading model")
338. *# --------------------------------------MATD3--------------------------------------------------------------------*
339. parser.add\_argument("--policy\_noise", type=float, default=0.1, help="Target policy smoothing")
340. parser.add\_argument("--noise\_clip", type=float, default=0.25, help="Clip noise")
341. parser.add\_argument("--policy\_update\_freq", type=int, default=2, help="The frequency of policy updates")
342. args = parser.parse\_args()
343. args.device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
344. args.output\_dir = os.path.join(args.output\_dir, '{}/env\_{}\_seed\_{}'.format(args.algorithm, args.env, args.seed))
345. os.makedirs(args.output\_dir, exist\_ok=True)
346. torch.manual\_seed(args.seed)
347. np.random.seed(args.seed)
348. world\_name = args.env + '.yaml'
349. env = multirobotenv.parallel\_env(render\_mode="rgb\_array", world\_name=world\_name)
350. env\_eval = multirobotenv.parallel\_env(render\_mode="rgb\_array", world\_name=world\_name)
351. args.num\_agents = len(env.possible\_agents)
352. num\_actions = env.action\_space(env.possible\_agents[0]).shape[0]
353. args.max\_action = env.action\_space(env.possible\_agents[0]).high[0]
354. observation\_size = env.observation\_space(env.possible\_agents[0]).shape[0]
355. agents = {agent\_id:MATD3(max\_action=args.max\_action, num\_states=observation\_size, num\_actions=num\_actions, args=args, agent\_id=agent\_id) for agent\_id in env.possible\_agents}
356. replay\_buffer = ReplayBuffer(args.num\_agents, observation\_size, num\_actions, args)
358. if args.do\_train:
359. writer = SummaryWriter(log\_dir='runs/{}/env\_{}\_seed\_{}'.format(args.algorithm, args.env, args.seed))
360. writer.add\_text("args", str(args))
361. train(args, env, agents, replay\_buffer, writer)
362. writer.close()
363. if args.do\_eval:
364. eval(args, env\_eval, agents)

**multirobotenv.py（环境相关代码）:**

1. import functools
2. import numpy as np
3. import random
4. import pygame
5. import yaml
6. import sys
7. from gym.spaces import Discrete, Box
8. from pettingzoo import ParallelEnv
9. from pettingzoo.utils import parallel\_to\_aec, wrappers
10. *# init*
11. from world import env\_plot, mobile\_robot, obs\_circle, obs\_polygon
12. import cv2
13. from env.env\_robot import env\_robot
14. from env.env\_obs\_cir import env\_obs\_cir
15. from env.env\_obs\_line import env\_obs\_line
16. from env.env\_source import env\_source
17. from env.env\_obs\_poly import env\_obs\_poly
18. from scipy.interpolate import RectBivariateSpline
19. import gymnasium
20. from gymnasium.utils import seeding
21. import math
22. *# Define constants*
23. REWARD\_FOUND\_ITEM = 300
24. class parallel\_env(ParallelEnv):
25. metadata = {"render\_modes": ["human, rgb\_array"], "name": "multi\_robot\_env", "render\_fps": 30}
26. def \_\_init\_\_(self, render\_mode="rgb\_array", obs\_type="coords", world\_name=None, max\_cycles=200, local\_ratio=1, \*\*kwargs):
27. self.render\_mode = render\_mode
28. self.\_obs\_type = obs\_type
29. world\_name = sys.path[0] + '/' + world\_name
30. self.local\_ratio = local\_ratio
31. with open(world\_name) as file:
32. com\_list = yaml.load(file, Loader=yaml.FullLoader)
33. world\_args = com\_list['world']
34. self.\_world\_height = world\_args.get('world\_height', 1000)
35. self.\_world\_width = world\_args.get('world\_width', 1000)
36. self.\_offset\_x = world\_args.get('offset\_x', 0)
37. self.\_offset\_y = world\_args.get('offset\_y', 0)
38. self.\_step\_time = world\_args.get('step\_time', 0.1)
39. self.\_world\_map =  world\_args.get('world\_map', None)
40. self.\_xy\_reso = world\_args.get('xy\_resolution', 1)
41. self.\_yaw\_reso = world\_args.get('yaw\_resolution', 5)
42. self.\_offset = np.array([self.\_offset\_x, self.\_offset\_y])
43. self.robots\_args = com\_list.get('robots', dict())
44. self.\_robot\_number = kwargs.get('robot\_number', self.robots\_args.get('robot\_number', 0) )
46. *# obs\_cir*
47. self.obs\_cirs\_args = com\_list.get('obs\_cirs', dict())
48. self.obs\_cir\_number = self.obs\_cirs\_args.get('number', 0)
49. self.obs\_step\_mode = self.obs\_cirs\_args.get('obs\_step\_mode', 0)
50. *# obs line*
51. self.obs\_lines\_args = com\_list.get('obs\_lines', dict())
52. *# obs polygons*
53. self.obs\_polygons\_args = com\_list.get('obs\_polygons', dict())
54. self.vertexes\_list = self.obs\_polygons\_args.get('vertexes\_list', [])
55. self.obs\_poly\_num = self.obs\_polygons\_args.get('number', 0)
56. *# source*
57. self.source\_args = com\_list.get('source', dict())
58. *# world*
59. self.\_screen\_height = int(self.\_world\_height / self.\_xy\_reso)
60. self.\_screen\_width = int(self.\_world\_width / self.\_xy\_reso)
61. pygame.init()
62. self.renderOn = False
63. self.max\_cycles = max\_cycles
64. self.\_screen = pygame.Surface([self.\_screen\_width, self.\_screen\_height])
66. self.possible\_agents = [f"agent\_{i}" for i in range(self.\_robot\_number)]
67. self.agent\_name\_mapping = dict(zip(self.possible\_agents, range(self.\_robot\_number)))
68. *# init*
69. self.components = dict()
70. self.init\_environment(\*\*kwargs)
71. def init\_environment(self, robot\_class=mobile\_robot, obs\_cir\_class=obs\_circle, obs\_polygon\_class=obs\_polygon,  \*\*kwargs):
72. if self.\_world\_map != None:
73. world\_map\_path = sys.path[0] + '/' + self.\_world\_map
74. map\_img = cv2.imread(world\_map\_path, cv2.IMREAD\_GRAYSCALE)
75. map\_matrix = cv2.resize(map\_img, (self.\_screen\_width, self.\_screen\_height), interpolation=cv2.INTER\_LINEAR)
76. *# cv2.imshow('Image', map\_matrix)*
77. *# cv2.waitKey(0)*
78. map\_matrix[map\_matrix>255/2] = 255
79. map\_matrix[map\_matrix<=255/2] = 0
80. self.map\_matrix = map\_matrix.T
81. else:
82. self.map\_matrix = None
83. self.components['map\_matrix'] = self.map\_matrix
84. self.components['xy\_reso'] = self.\_xy\_reso
85. self.components['offset'] = np.array([self.\_offset\_x, self.\_offset\_y])
86. self.components['obs\_lines'] = env\_obs\_line(\*\*{\*\*self.obs\_lines\_args, \*\*kwargs})
87. self.obs\_line\_states=self.components['obs\_lines'].obs\_line\_states
88. self.components['sources'] = env\_source(components=self.components, \*\*{\*\*self.source\_args, \*\*kwargs})
89. self.source\_list = self.components['sources'].source\_list
91. self.components['obs\_circles'] = env\_obs\_cir(obs\_cir\_class=obs\_cir\_class, obs\_cir\_num=self.obs\_cir\_number, step\_time=self.\_step\_time, components=self.components, \*\*{\*\*self.obs\_cirs\_args, \*\*kwargs})
92. self.obs\_cir\_list = self.components['obs\_circles'].obs\_cir\_list
93. self.components['obs\_polygons'] = env\_obs\_poly(obs\_poly\_class=obs\_polygon\_class, vertex\_list=self.vertexes\_list, obs\_poly\_num=self.obs\_poly\_num, \*\*{\*\*self.obs\_polygons\_args, \*\*kwargs})
94. self.obs\_poly\_list = self.components['obs\_polygons'].obs\_poly\_list
95. self.components['robots'] = env\_robot(robot\_class=robot\_class, step\_time=self.\_step\_time, components=self.components, \*\*{\*\*self.robots\_args, \*\*kwargs})
96. self.robot\_list = self.components['robots'].robot\_list
97. self.world\_plot = env\_plot(self.\_world\_width, self.\_world\_height, self.\_xy\_reso, self.components,map\_matrix=self.map\_matrix ,offset\_x=self.\_offset\_x, offset\_y=self.\_offset\_y, \*\*kwargs)
98. if self.\_robot\_number > 0:
99. self.robot = self.components['robots'].robot\_list[0]
100. self.components['robots'].init\_transforms()
101. @functools.lru\_cache(maxsize=None)
102. def observation\_space(self, agent):
103. if agent in self.possible\_agents:
104. if self.\_obs\_type == "image":
105. return Box(low=0, high=255, shape=(self.\_screen\_width, self.\_screen\_height, 3), dtype=np.uint8)
106. else:
107. return Box(low=-np.inf, high=np.inf, shape=(10 + 2\*(len(self.robot\_list) - 1) +2\*len(self.obs\_cir\_list) + self.robot\_list[0].lidar.data\_num, ), dtype=np.float32)
108. else:
109. return None  *# 处理未知的代理*
111. @functools.lru\_cache(maxsize=None)
112. def action\_space(self, agent):
113. if agent in self.possible\_agents:
114. return Box(low=-self.robot\_list[0].vel\_max[0, 0], high=self.robot\_list[0].vel\_max[1, 0], shape=(2,), dtype=np.float32)  *# 速度和角速度的动作空间*
115. else:
116. return None  *# 处理未知的代理*
118. def render(self):
119. if self.render\_mode is None:
120. gymnasium.logger.warn(
121. "You are calling render method without specifying any render mode."
122. )
123. return
124. self.enable\_render(self.render\_mode)
125. \_double\_buffer = self.world\_plot.draw\_dyna\_components()
126. self.\_screen.blit(\_double\_buffer, (0, 0))
127. if self.render\_mode == "rgb\_array":
128. observation = np.array(pygame.surfarray.pixels3d(self.\_screen))
129. return np.transpose(observation, axes=(1, 0, 2))
130. elif self.render\_mode == "human":
131. pygame.display.flip()
132. self.clock.tick(self.metadata["render\_fps"])
133. *# pygame.image.save(self.\_screen, 'multi\_map.png')*
134. return
136. def enable\_render(self, mode="human"):
137. if not self.renderOn and mode == "human":
138. self.\_screen = pygame.display.set\_mode(self.\_screen.get\_size())
139. self.clock = pygame.time.Clock()
140. self.renderOn = True
141. def render\_coordinate\_array(self, robot, source):
142. old\_detector = source.source\_detector(robot.previous\_state, robot.radius, self.components)
143. detector = source.source\_detector(robot.state, robot.radius, self.components)
144. r\_d = detector - self.max\_detector
145. min\_dis, min\_angle = robot.lidar.nearest\_obstacle(robot.state)
146. entity\_pos = []
147. for obs\_cir in self.obs\_cir\_list:  *# world.entities:*
148. entity\_pos.append(obs\_cir.state[0:2,0] - robot.state[0:2,0])
149. source\_pos = []
150. source\_pos.append(source.source\_state[0:2,0] - robot.state[0:2,0])
151. other\_pos = []
152. for other in self.robot\_list:
153. if other.id == robot.id:
154. continue
155. other\_pos.append(other.state[0:2,0] - robot.state[0:2,0])
156. observation = np.concatenate([\*source\_pos, [-r\_d], \*entity\_pos, \*other\_pos, robot.lidar.range\_data.flatten(), [robot.lidar.range\_max - min\_dis], [min\_angle], [robot.radius\_collision], robot.vel\_diff.flatten(), robot.state[0:2, 0]])
157. return observation
159. def close(self):
160. if self.render\_mode == "human":
161. pygame.quit()  *# 关闭 Pygame 窗口*
162. def reset(self, seed=None, options=None, reset\_mode=4):
163. self.agents = self.possible\_agents[:]
164. self.components['sources'].sources\_reset(reset\_mode=0, include\_robot=True)
165. self.components['robots'].robots\_reset(reset\_mode=reset\_mode, include\_robot=True, search\_source=True)
166. self.world\_plot.robot\_trajectories = {agent: [] for agent in range(self.\_robot\_number)}
167. for robot in self.robot\_list:
168. robot.cal\_lidar\_range(self.components)
169. self.steps = 0
170. self.wait = 0
171. self.max\_detector = self.source\_list[0].source\_detector(self.robot\_list[0].state, self.robot\_list[0].radius, self.components)
172. observations = {agent: self.render\_coordinate\_array(self.robot\_list[i], self.source\_list[0]) for i, agent in enumerate(self.agents)}
173. self.state = observations
174. infos = {agent: {} for agent in self.agents}
175. detector = {agent:math.exp(self.source\_list[0].source\_detector(self.robot\_list[i].state, self.robot\_list[i].radius, self.components)) - 1 for i, agent in enumerate(self.agents)}
176. return observations, infos, detector
177. def step(self, actions):
178. self.steps += 1
179. for i, (agent, action) in enumerate(actions.items()):
180. self.robot\_list[i].move\_forward(action)
181. self.robot\_list[i].cal\_lidar\_range(self.components)
182. self.source\_list[0].source\_check(self.robot\_list[i],self.components)
183. self.robot\_list[i].collision\_check(self.components)
185. rewards, infos, env\_termination = self.cal\_reward(self.robot\_list, self.source\_list)
186. *# 提前终止、游戏胜利和终止都应该env\_termination = True而env\_truncation表示截断*
187. if min(infos.values()) == True and not env\_termination:
188. self.wait += 1
189. if self.wait > self.max\_cycles\*0.05:
190. rewards = {key: value + (self.max\_cycles - self.steps)\*value for key, value in rewards.items()}
191. env\_termination = True
192. *# pygame.image.save(self.\_screen, 'multi\_map.png')*
193. *# 一荣俱荣、一损俱损*
194. terminations = {agent: env\_termination for agent in self.agents}
195. env\_truncation = self.steps >= self.max\_cycles
196. truncations = {agent: env\_truncation for agent in self.agents}
197. observations = {agent: self.render\_coordinate\_array(self.robot\_list[i], self.source\_list[0]) for i, agent in enumerate(self.agents)}
198. self.state = observations
200. detector = {agent:math.exp(self.source\_list[0].source\_detector(self.robot\_list[i].state, self.robot\_list[i].radius, self.components)) - 1 for i, agent in enumerate(self.agents)}
201. if env\_truncation or env\_termination:
202. if env\_truncation and min(infos.values()) == False:
203. rewards = {key: value - self.max\_cycles \* math.exp(-self.steps/self.max\_cycles) for key, value in rewards.items()}
204. self.agents = []
205. if self.render\_mode == "human":
206. self.render()
207. return observations, rewards, terminations, truncations, infos, detector
208. def cal\_reward(self, robot\_list, source\_list, beta=1, theta=1, mu=1, gamma=1):
209. rewards = {}
210. infos = {}
211. global\_reward = 0.0
212. if self.local\_ratio is not None:
213. for s in source\_list:
214. dists = [
215. np.sqrt(np.sum(np.square(a.state[0:2,0] - s.source\_state[0:2,0])))
216. for a in robot\_list
217. ]
218. global\_reward -= sum(dists)
219. agent\_reward = {}
220. find\_num = sum(robot.source\_flag for robot in robot\_list)
221. for robot in robot\_list:
222. detector = source\_list[0].source\_detector(robot.state, robot.radius, self.components)
223. *# 不能取等*
224. if detector - self.max\_detector > 0.0:
225. r\_d = math.exp(detector - self.max\_detector)
226. else:
227. r\_d = - mu\*max(self.max\_detector - detector, 0)
228. self.max\_detector = detector if self.max\_detector < detector else self.max\_detector
230. if robot.source\_flag:
231. r\_d += 10\*math.exp(find\_num/self.\_robot\_number)
232. min\_dis, \_ = robot.lidar.nearest\_obstacle(robot.state)
233. if min\_dis < robot.radius\*1.5:
234. r\_c = - beta \* max(robot.lidar.range\_max - min\_dis, 0)
235. else:
236. r\_c = 0.0
237. agent\_reward[robot.id] = sum([r\_d, r\_c])
238. if self.local\_ratio is not None:
239. reward = (
240. global\_reward \* (1 - self.local\_ratio)
241. + agent\_reward[robot.id] \* self.local\_ratio
242. )
243. else:
244. reward = agent\_reward[robot.id]
245. rewards[f"agent\_{robot.id}"] = reward
246. infos[f"agent\_{robot.id}"] = robot.source\_flag
247. env\_termination = any(robot.collision\_flag for robot in robot\_list)
248. if env\_termination:
249. rewards = {key: value - self.max\_cycles \* math.exp(-self.steps/self.max\_cycles) for key, value in rewards.items()}
250. return rewards, infos, env\_termination
252. def env(render\_mode=None):
253. internal\_render\_mode = render\_mode if render\_mode != "ansi" else "human"
254. env = raw\_env(render\_mode=internal\_render\_mode)
255. if render\_mode == "ansi":
256. env = wrappers.CaptureStdoutWrapper(env)
257. env = wrappers.AssertOutOfBoundsWrapper(env)
258. env = wrappers.OrderEnforcingWrapper(env)
259. return env
260. def raw\_env(render\_mode=None):
261. env = parallel\_env(render\_mode=render\_mode)
262. env = parallel\_to\_aec(env)
263. return env