联合作战人机协同博弈挑战赛
智能体开发与训练框架使用指南

国防科大系统工程学院

中国电子科技集团公司第五十二研究所

北京机械设备研究所

二〇二五年七月

目　　录

1 概述

1.1 智能体开发与使用流程

智能体开发与训练框架封装了连接灵弈服务器、订阅并解析态势数据、数据处理、推演流程控制等大量而繁琐的底层编码，为选手提供基于灵弈平台的智能体 Python 开发环境和接口，选手可专注于战术战法实现和算法设计。

在赛前准备阶段，选手在智能体开发与训练框架内对智能体进行开发、训练和调试工作。

在比赛使用阶段，选手将智能体文件保存至灵弈客户端的指定路径下，即可在灵弈客户端界面中使用智能体进行比赛。步骤为：打开灵弈客户端-智能体调度按钮，根据界面提示导入单个智能体或者智能体配置，添加智能体控制的元素，点击运行。

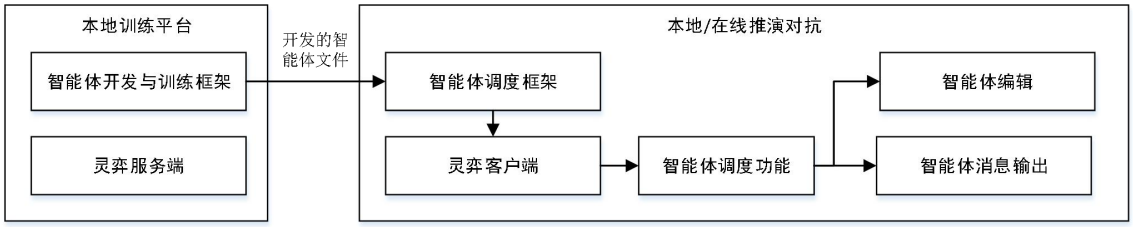

图 1 智能体开发与使用流程

1.2 项目工程介绍

在灵奕智能训练平台中可以自定义实现规则智能体和强化学习智能体的二次开发，强化学习智能体可以自定义配置它的训练参数（学习率、训练数据批次、停止条件等）、动作空间、状态空间、奖励函数等，规则智能体能够根据借口进行自定义配置其航线、目标等。项目工程文件总体内容如下图所示:

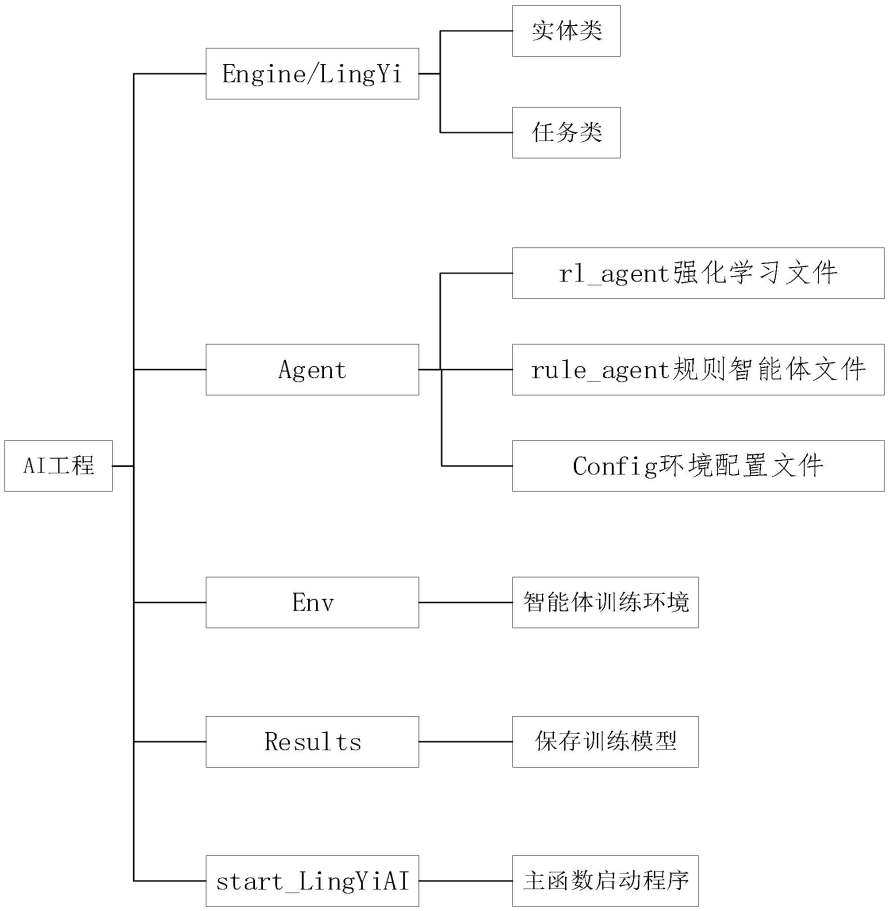

图 2 工程文件简介

Engine/LingYi 主要包括灵弈文件夹中的底层引擎设置，包括任务和实体基类等；

Agent 主要包含 rule_agent、rl_agent、config 等文件。其中 rl_agent 主要包括保存强化学习智能体类文件(状态空间、动作空间、奖励函数、指令接口等)；rule_agent:规则智能体初始化配；Config 推演环境配置文件，包含强化学习智能体/规则智能体初始化配置，服务器参数初始；

Env 主要配置智能体训练环境；

start_LingYiAI 主函数启动程序。

2 智能体开发与训练

2.1 智能体开发

2.1.1 强化学习智能体开发

2.1.1.1 强化学习智能体单元配置

选手可以配置推演方下强化学习智能体单元。

- 代码路径
  \agent\config.py 中,配置接口为类 LingYiConfig 中的 get_rl_agent 接口。
- 参数配置

```python
@classmethod
def get_rl_agent(cls):
    """
    用户需要在下方配置使用的强化学习智能体以及控制的单元，智能体类
    """
    rl_dict: dict = \
        {
            "红方": {
                "agents": {
                    'redFighterT50': {  # 强化学习智能体描述--此为战斗机，用于空战学习训练
                            "class": FighterAgent,  # 智能体类，用户基于基类RLAgentBase开发的子类，
                            "unit": ['T-50型 #1', "T-50型 #2", "T-50型 #3", "T-50型 #4"],  #
                            # 单元名列表，智能体控制的单元，强化学习会给每个单元建立一个智能体
                            "contact": [],  # 情报名列表，情报参数，可以客户端里智能体调度动态配置，示例：['空中#1', '空中#2']
                            "point": []  # 参考点名列表，参考点参数，可以客户端里智能体调度动态配置，示例：['RP-112', 'RP-113', 'RP-114']
                        },
                    # 可以继续添加红方强化学习智能体，注意：智能体越多可能越不容易奖励值收敛，不容易训练出好的效果
                },
                "enemy": "蓝方"  # 敌方，用于强化学习训练时状态输入和奖励计算
            },
            "蓝方": {
                "agents": {
                    'blueFighterF22': {  # 强化学习智能体描述
                            "class": BlueFighterAgent,  # 智能体类，用户基于基类RLAgentBase开发的子类，
                            "unit": ["F-22A型 #1", "F-22A型 #2"],  # 单元名列表，强化学习会给每个单元建立一个智能体
                            "contact": [],  # 情报名列表，情报参数，可以客户端里智能体调度动态配置
                            "point": []  # 参考点列表，参考点参数，可以客户端里智能体调度动态配置
                        },
                    'blueFighterF35': {  # 强化学习智能体描述
                        "class": BlueFighterAgent,  # 智能体类，用户基于基类RLAgentBase开发的子类，
                        "unit": ["F-35FA 型 #1", "F-35C 型 #2"],  # 单元名列表，强化学习会给每个单元建立一个智能体
                        "contact": [],  # 情报名列表，情报参数，可以客户端里智能体调度动态配置
                        "point": []  # 参考点列表，参考点参数，可以客户端里智能体调度动态配置
                    }
                },
                "enemy": "红方"  # 敌方，用于强化学习训练时状态输入和奖励计算
            },
        }
    return rl_dict
```

图 3 强化学习智能体单元配置

选手需要输入有强化学习智能体的对应推演方，agents 中包含需要强化学习智能体名称类型，以及该类型包含的强化学习智能体类、智能体名称列表、情报列表（可选）、参考点列表（可选）。

2.1.1.2 学习策略配置

选手可以配置智能体不同的学习策略。

- 代码路径

\agent\config.py 中,配置接口为类 LingYiConfig 中的 get_multiagent_config 接口。

- 策略配置

如果选手想尝试分布式学习策略/混合式学习策略/集中式策略，可配置强化学习策略来修改策略映射条件。

```
for index, agent in enumerate(agents):
    agent_id = agent.uid
    if agent.agent_type not in agent_des_list:
        policy_id: str = "policy_%d" % len(agent_des_list)  # 智能体对应的策略
        agent_des_list.append(agent.agent_type)
    else:
        policy_index = agent_des_list.index(agent.agent_type)
        policy_id: str = "policy_%d" % policy_index  # 智能体对应的策略
    module_path = cls.get_class_file_path(type(agent))
    policy_module_path[policy_id] = os.path.join(module_path, 'model', agent.agent_type)
    agent_map_policy_dict[agent_id] = policy_id
    if policy_id not in policies.keys():
        policies[policy_id] = PolicySpec(
            observation_space=agent.observation_space,
            action_space=agent.action_space,
            config={
                "model": {"custom_model": agent.model},
            }
        )
        policies[policy_id].agent_type = agent.agent_type  # 另外存储策略的算法描述

def policy_mapping_fn(_agent_id, episode, worker, **kwargs):
    return agent_map_policy_dict[_agent_id]

def is_policy_to_train(pid, batch=None):
    return pid == list(policies.keys())

multiagent_config = {
    "agent_map": agent_map,
    "policies": policies,
    "policy_mapping_fn": policy_mapping_fn,
    "policies_to_train": list(policies.keys()),  # policies_to_train,
    "policy_module_path": policy_module_path  # 用于存储策略的模型路径
}
return multiagent_config
```

图 4 策略配置

键值“agent_map”包含所有的强化学习智能体；

键值“policies”包含所有生成的强化学习策略;

键值“policy_mapping_fn”为策略映射字典，即每一个策略所控制的强化学习智能体;

键值“policies_to_train”为需要进行训练的强化学习策略。

2.1.1.3 智能体开发

- 概要

Agent 类继承自强化学习基类 RLAgentBase，RLAgentBase 类目前实现了智能体基础接口，主要实现接口包括 Act_trans、Obs_trans、Get_reward、Load_model 接口，用户可以参考 agent\rl_agent\fighter_agent\Fighter.py 中的 class Agent（RLAgentBase）进行改写。

接下来对其中的几个重要函数接口进行详细介绍:

Act_trans:该接口主要进行动作指令的转换，将决策模型的决策指令数列转换为引擎的可执行指令格式,以下是策略学习过程中主要进行的指令转换流程:

环境信息 -> 学习策略() -> 指令数组 -> 指令转换() -> 可执行指令

Obs_trans:态势空间转换接口，将引擎返回的观测信息转换为决策模型可执行的观测信息数列。

Get_reward：进行奖励函数设置来控制智能体的训练目标。

Load_model：加载神经网络模型。

- 智能体初始化配置

强化学习智能体类继承自基类 Agent，初始化过程中首先设定其的状态空间和动作空间，方便后边进行神经网络计算，防止出现维度错误；状态空间通过 gym.spaces.Box 进行定义，该数据类型用于定义连续值多维空间的数据类型，状态空间维度设置为 316；动作空间通过 gym.spaces.MultiDiscrete 进行定义，该数据类型为多维离散动作空间，离散动作空间维度设置为[3,2,2]。

在初始化配置中输入该场景下我方实体配置数量以及敌方实体配置数量，方便后边进行奖励函数的计算。同时还要指定神经网络模型，即该智能体所对应的神经网络：

```python
class Agent(RLAgentBase):
    """灵异战斗机智能体"""
    action_space: gym.Space = gym.spaces.MultiDiscrete([3, 2, 2])  # 动作空间维度---必须定义
    observation_space: gym.Space = gym.spaces.Box(low=-np.inf, high=+np.inf, shape=(316,))  # 状态空间维度---必须定义

    def __init__(self, agent_conf: dict = None):
        """配置状态空间、动作空间"""
        if agent_conf is None:
            agent_conf = {}
        super().__init__(agent_conf)
        self.aircraft_fuel = 8  # 表示已经消耗70%油了
        # self.model = None  # 智能体神经网络模型，客户端调度使用时需加载模型用于决策
        # 用户需重写load_model函数，加载生成模型，训练时在智能体脚本的model目录下会存储模型文件，例如"model_626.h"，代表第626次训练的模型
        self.action_dim = [3, 2, 2]  # 动作空间维度
        self.max_missile_num = 6  # 单架歼战斗机导弹最大挂载
        self.teammate_fighter_num = 4  # 友方战斗机最大数量
        self.teammate_bomber_num = 5  # 友方轰炸机最大数量
        self.teammate_aew_num = 2  # 友方预警机最大数量
        self.enemy_aircraft_num = 12  # 敌方战斗机最大数量
        self.enemy_ship_num = 5  # 敌方舰艇最大数量
        self.crt_select_airs = set()  # 客户端应用时，当前已编辑的单元
        self.patrol_mission = None
```

图 5 智能体初始化函数

- 动作空间设计与指令下发

强化学习智能体动作空间在 ray 框架中需设计为离散向量，其设计为 MulitDiscrete 多维离散动作空间[3,2,2]；同时也可以为单维离散动作空间[12]，对其进行解码操作可以得到对应的动作序列指令：

```python
action_space: gym.Space = gym.spaces.MultiDiscrete([3, 2, 2])  # 动作空间维度
```

图 6 离散动作空间设置

第一个维度控制机动和武器开关，第二个维度控制雷达开关，第三个维度控制电子干扰

开关；act_trans 接口接收离散动作指令转换为

智能体下一时刻的航路点以及飞行高度，武器挂载情况，雷达开关情况和电子干扰开关情况：

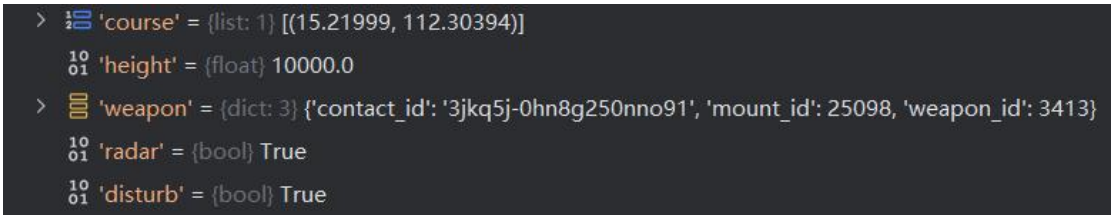

图 7 返回的动作空间字典

上面的动作空间字典会返回传输到 ray 框架中的 action_trans_dict 中，然后再输入到灵弈环境中的 deduction 中进行底层的指令下发：

```
red_step_obs[0], blue_step_obs[0] = self.env.step(agents_info=self.agents,
                                                  action_dict=action_trans_dict,
                                                  rule_agent = self.rule_agent,
                                                  rule_obs = rule_obs)
```

图 8 动作指令传入到 Dedection 中

self.env 指向的为 Deduction 类，该类主要负责灵弈环境的推理，在 Deduction 中的 step 子函数中进行智能体指令的下发：

```
    # 执行智能体动作
    player.step_rl(self.time_elapsed, player_situation, agents_info, action_dict)
```

图 9 在 Deduction 中进行指令下发

Player 指向的为本场景的红方智能体，因此通过调用红方智能体中的 step_rl 函数进行指令下发：

```python
def step_rl(self, time_elapse, situation, action_dict):
    """
    基于博弈算法框架输出的动作驱动相应装备执行相关指令
    推理应用使，调度框架会自动调用此函数
    每个强化学习描述每步决策调用一次，请在函数里实现所有多智能体的动作调用
    :param time_elapse:int, 推演仿真时间，单位：秒，取值0-推演总时间秒数
    :param situation: tuple, 单元字典和探测到的目标字典, units dict, contacts dict
    :param action_dict: dict, 智能体的单元id和动作，key:单元guid, value:动作信息字典，从act_trans函数计算而来
    :return:
    """
    # 根据智能体控制对象的单元id，获取单元对象，调用接口下发动作指令
    for agent_unit_id, agent_action in action_dict.items():
        agent_unit = self.get_unit(agent_unit_id)  # 获取到智能体对象
        if agent_unit is not None:
            # 取消智能体上一时刻的所有目标，以免影响当前攻击指令下发
            agent_unit.attack_drop_target_all()
            # 取消上一时刻的航路点，以免影响当前航路点指令下发
            agent_unit.clear_plotted_course()
            if "weapon" in agent_action:
                contact_guid = agent_action['weapon']['contact_id']
                weapon_id = agent_action['weapon']['weapon_id']
                if contact_guid and weapon_id:
                    # 2.1 机动中执行追击敌方飞机目标  打击目标发射1枚弹
                    agent_unit.attack_weapon_allocate_to_target(contact_guid, weapon_id, 1)
                elif "course" in agent_action:
                    # 2.2 机动中执行左飞、右飞
                    agent_unit.plotted_course(agent_action["course"])  # 实体航线规划
            # 2.3 传感器指令下发
            agent_unit.EMCON_Obey_doctrine(False)  # 智能体单元不用遵循电磁管控条令
            agent_unit.switch_radar(agent_action["radar"])  # 智能体单元雷达开关
```

图 10 在 step_rl 中进行底层的指令下发

可以看到在 step_rl 中根据接受到的动作指令调用接口 attack_weapon_allocate_to_target、plotted_course、switch_radar 进行智能体攻击、机动、传感器指令的下发。

- 智能体观测空间设计

  智能体的观测空间主要包括以下几个部分：

```python
# 获取智能体的自身属性信息，维度为10
agent_obs.extend(self._get_agent_attribute_info(aircraft_info=aircraft_info))
# 获取智能体的雷达信息，维度为3
# agent_obs.extend(self._get_agent_radar_info(sensors_info=aircraft_info.Sensors))
# 获取智能体电子干扰信息，维度为16
agent_obs.extend(self._get_agent_distrub_info(sensors_info=aircraft_info.Sensors))

# 获取智能体导弹信息，维度为18
agent_obs.extend(self._get_agent_missile_info(loadouts_info=aircraft_info.Loadouts))

# 获取智能体友机信息(包括战斗机和轰炸机)，维度为136
agent_obs.extend(self._get_agent_teammate_info(aircrafts_info=red_obs.aircrafts))

# 获取智能体敌方目标信息，维度为136
agent_obs.extend(self._get_agent_enemy_info(aircrafts_info=red_obs.aircrafts,
                                            contacts_info=red_obs.Contacts))
```

图 11 观测空间态势信息

分别为自身属性信息(维度为 10)，传感器信息（维度为 16），导弹信息（维度为 18），友方信息（维度为 136），敌方目标信息（维度为 136），总维度为 316。

每一个维度的信息都调用对应的接口对态势空间进行处理提取到需要的智能体观测空间信息，选手可以需要设计不同的观测空间；

```python
def _get_agent_attribute_info(self, aircraft_info):
    """
    获取飞机的属性信息
    Args:
        aircraft_info: 飞机的全部态势信息
    Returns:
        attribute_obs: 飞机的属性信息
    """
    attribute_obs: list = []
    attribute_obs.append(aircraft_info.Lon)  # 智能体经度
    attribute_obs.append(aircraft_info.Lat)  # 智能体纬度
    attribute_obs.append(aircraft_info.CurrentAltitude)  # 智能体高度
    attribute_obs.append(aircraft_info.CurrentSpeed)  # 智能体速度
    attribute_obs.append(aircraft_info.CurrentHeading)  # 智能体航向角
    attribute_obs.append(aircraft_info.Pitch)  # 智能体俯仰角
    attribute_obs.append(aircraft_info.Roll)  # 智能体翻滚角
    attribute_obs.append(aircraft_info.Status)  # 智能体存活状态
    attribute_obs.append(aircraft_info.FuelState)  # 智能体燃油状态
    attribute_obs.append(aircraft_info.FuelRate)  # 智能体燃油速率
    return attribute_obs
```

图 12 获取飞机属性信息接口

2.1.2 规则智能体开发

2.1.2.1 规则智能体单元配置

选手可以配置推演方下规则智能体单元。

- 代码路径
  \agent\config.py 中，配置接口为类 LingYiConfig 中的 get_rule_dict 接口。
- 参数配置

```python
def get_rule_dict(self):
    """
    规则智能体配置
    用户需要在下方配置使用的规则智能体以及单元参数、情报或参考点（区域），智能体类
    """
    rule_dict = {
        "红方": {
            'redmission':  # 规则智能体描述---可以用户自定义
                {
                    "class": MissionAgent,  # 智能体类，用户基于基类RuleAgentBase开发的子类
                    "unit": [],  # 单元名列表，可以客户端里智能体调度动态配置，这里此规则智能体无需单元参数
                    "contact": [],  # 情报名列表，可以客户端里智能体调度动态配置
                    "point": []  # 参考点名列表，可以客户端里智能体调度动态配置
                },
        },
        "蓝方": {
            "blueAirbat":  # 规则智能体描述---可以用户自定义
                {
                    "class": BlueAirBat,  # 智能体类，用户基于基类RuleAgentBase开发的子类
                    "unit": ["F-15K型 #1", "F-15K型 #2", "F-5F型 #1", "F-5F型 #2", "F-5F型 #3",
                             "F-5F型 #4", "F-5F型 #5"],  # 单元名列表，可以客户端里智能体调度动态配置，这里在训练框架中预先配置好
                    "contact": ['空中#1', '空中#2'],  # 情报名列表，可以客户端里智能体调度动态配置
                    "point": ['RP-01', 'RP-02', 'RP-03']  # 参考点名列表，可以客户端里智能体调度动态配置
                },
        },
    }
    return rule_dict
```

图 13 规则智能体参数配置

字典主要包含推演方设置、规则智能体描述。其中规则智能体描述中选手可以配置“unit”、“contact”、“point”的。

键值“contact”包含需要打击的敌方目标信息；

“point”包含规则智能体需要前往的路径点信息；

“unit”包含规则智能体控制的我方单元名称。

2.1.2.2 规则智能体开发

选手可以继承 RuleAgentBase 类编写自己的规则智能体类，调用需要实现的接口，可调用的接口在 operatorbase.py 中的 RuleAgentBase 类接口，可调用部分接口如下图。

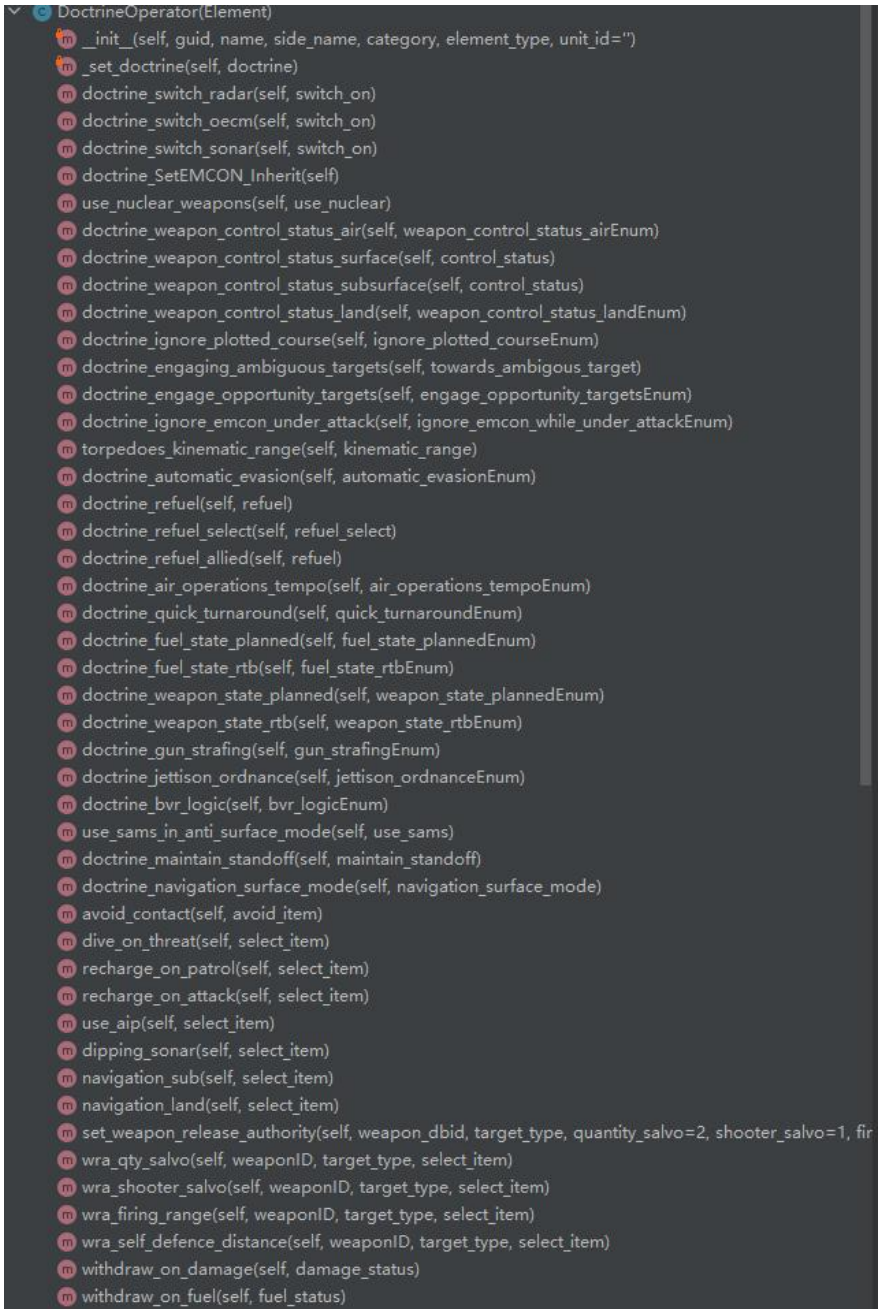

图 14 RuleAgentBase 类部分任务指令接口

用户可以根据场景需求设计规则智能体调用对应指令下发接口完成预订任务要求，下面为一个规则智能体类的典型示例：

```python
class Agent(AgentBase):
    def __init__(self, side_name, params):
        super(Agent, self).__init__(side_name, params)
        self.IsDone = False

    def initial(self, situation):
        # 北部预警
        points_1 = [(54.97, 166.77, "UserDef-1-1"), (54.97, 167.35, "UserDef-1-2"), (54.73, 167.35, "UserDef-1-3"),
                    (54.73, 166.77, "UserDef-1-4")]
        # 南部预警
        points_2 = [(54.40, 166.77, "UserDef-2-1"), (54.40, 167.35, "UserDef-2-2"), (54.16, 167.35, "UserDef-2-3"),
                    (54.16, 166.77, "UserDef-2-4")]
        # 电子侦察
        points_3 = [(54.75, 166.88, "UserDef-3-1"), (54.75, 167.80, "UserDef-3-2"), (54.42, 167.80, "UserDef-3-3"),
                    (54.42, 166.88, "UserDef-3-4")]
        # 北部制空1
        points_4 = [(55.40, 167.64, "UserDef-4-1"), (55.40, 169.11, "UserDef-4-2"), (54.72, 169.11, "UserDef-4-3"),
                    (54.72, 167.64, "UserDef-4-4")]
        # 南部制空2
        points_5 = [(54.72, 167.64, "UserDef-5-1"), (54.72, 169.11, "UserDef-5-2"), (53.98, 169.11, "UserDef-5-3"),
                    (53.98, 167.64, "UserDef-5-4")]
        """
         添加一个或多个参考点
         :param points: tuple, 或list,  参考点列表,例:(40.2, 49.6) 或 [(40, 39.0), (41, 39.0)]，其中纬度值在前，经度值在后
                                          或者 (40.2, 49.6, 'RP002') 或 [(40, 39.0, 'RP1'), (41, 39.0, 'RP2')]，已传入参考点命名
         :return:
        """
        self.reference_point_add(points_1)
        self.reference_point_add(points_2)
        self.reference_point_add(points_3)
        self.reference_point_add(points_4)
        self.reference_point_add(points_5)

        # 移动目标
        points_6 = [(54.40, 171.50, "MOVE-1"), (54.40, 169.80, "MOVE-2"), (54.30, 166.30, "MOVE-3")]

        self.reference_point_add(points_6)

        pass

    def step(self, time_elapse, situation):
        # self.set_done()
        # print("-----set_done-----")
        pass
```

图 15 规则智能体开发典型示例

Step 函数：实现推演过程中指令下发，状态更新等；

Initial 函数：实现初始化过程中推演变量初始化和状态设置等；

图中的规则智能体在初始化过程中添加部分参考点以便后边推演时规则智能体的控制单元会前往添加的目标点。

2.2 智能体训练

2.2.1 训练参数设置

- 代码路径

  \agent\config.py 文件中 LingYiConfig 类下的 get_train_config 接口。

- 参数配置

图 16 训练参数

可配置强化学习训练参数主要包括：

“alg”:为训练使用的强化学习算法，默认为 PPO 算法，可以选择的方案包括 SAC,DQN,PPO,QMIX,IMPALA,TD3,DDPG（不推荐，速度较慢）强化学习算法，选手可以根据自己的要求自定义配置算法；需要注意的是如果选手使用的为 SAC 算法时强化学习智能体的动作空间需要为 Discrete 类型，在初始化过程中动作空间需要使用 gym.spaces.Discrete 进行初始化；

“num_workers” :同时训练的进程数，数值过高容易造成系统崩溃(最大值为 4);

“num_agents” :该场景下智能体总数;

“train_batch_size” :每次进行策略更新时用于训练的样本数量；

“training_mode” :训练模式，local 为本地训练模式，可以进行 debug 调试;normal 为本地训练，但是不可调试；cluster 为集群训练模式；

“lr” :强化学习过程中策略更新的学习率;

“gamma” :折扣因子，用于均衡训练过程中未来奖励的当前价值，决定当前智能体在训练过程中即使奖励和未来奖励的重要程度

“lambda” :用于控制优势估计的偏差和方差之间的权衡；

“sgd_minibatch_size” :决定每次更新模型参数时使用的样本数量；

“store_model_interval” :保存模型的间隔;

“store_model_num” :保存模型的最大数量；

“load_pretrained” :设置是否读取预训练模型；

“stop_iter” :设置停止前的最大迭代次数;

“stop_steps”设置停止前的最大运行步数；

“stop_timesteps”设置停止前的最大时间步长；

“stop_reward“设置停止前的最大奖励值。

2.2.2 训练局数设置

- 代码路径

  envs/Lingyi/env.py 中的 LingYiEnv 类。

- 参数配置

```python
class LingYiEnv(BaseEnv):
    """灵弈智能推演系统智能体训练环境"""
    _skip_env_checking = True
    engine = None
    train_name = ""

    def __init__(self, env_config: "EnvContext"):
        super().__init__(env_config)
        self.max_step = env_config['max_step']  # 每一局最大步数
        self.log_path = os.path.join(env_config["log_path"], "LingYi")
        if not os.path.exists(self.log_path):
            os.makedirs(self.log_path)
        # 修改训练进程房间名
        if not LingYiEnv.train_name:
            LingYiEnv.train_name = env_config['game_server']['server']['room_name']
        # 依据worker编号自动生成新的房间名
        env_config['game_server']['server']['room_name'] = LingYiEnv.train_name + str(env_config.worker_index)
        self.env = Deduction(env_config)
        self.room_list = None
        self.store_model_step = env_config['store_model_interval']
        self.last_agent_obs = {}
        self.restart_period = 50  # 每间隔10局重启一次灵弈服务端
        self.timeout = 10000  # 每一局中单步等待时间阈值
        self.start_time = datetime.datetime.now().strftime("%Y-%m-%d_%H%M%S")
        self.train_data_file_name = f"{self.start_time}_train_params.json"
        # self.side = env_config['game_server']['side']  # 配置需要获取态势信息的推演方----------618
        self.cur_path = os.getcwd()
        self.env_config = env_config
        self.get_agents()  # 获取智能体
        # self.rule_obs = None  # 传入规则智能体的状态空间----------618
        self.training_data_initial()
        self.worker_index = env_config.worker_index  # worker编号
        self.worker_room_name = env_config['game_server']['server']['room_name']
        self.first_reset = True  # 环境第一次reset无效，从第2次开始
```

图 17 训练局数设置

self.timeout：每一局中等待的最大时间阈值

self.restart_period：重启灵弈服务端的局数(防止训练时间过长造成灵弈服务器卡顿)

其他参数在灵弈环境初始化过程中会自动生成，不需要选手进行单独配置。

2.2.3 训练模式设置

支持本地训练，支持单房间训练、多房间并行训练。

- 代码路径

在/agent/config.py 文件中类 LingYiConfig 下的 get_train_config 接口。

- 参数配置

```python
def get_train_config(self, args: dict) -> dict:
    """！！！选手可以在下面配置训练算法参数(可选)！！！"""
    train_config: dict = {
        # 训练参数
        "alg": "PPO",
        "engine": "LingYiWarGame",  # 引擎名，智能体存储模型目录名
        "num_workers": 1,  # 房间数量，最大上限为4(超过上限容易卡死)
        "max_step": 18,  # 每一局运行的最大步数
        "train_batch_size": 10,  # 决策多少次保存一次模型
        "training_mode": "local",  # local:本地训练, normal:本地训练(不可调试), cluster:集群训练, 需在配置文件中指定集群head节点ip,port
        "api": "tune",  # 测试分布式新增
        "framework": "tf",
        "lr": 5e-03,
        "no_done_at_end": False,
        "gamma": 0.95,
        "lambda": 0.9,
        "sgd_minibatch_size": 10,
        "store_model_interval": 1,  # 模型保存间隔
        "store_model_num": 1,
        "load_pretrained": True,  # 是否加载预训练模型
        "rule_engine": False,
        "camp": "Red",
        # 停止条件
        "stop_iters": 300,
        "stop_steps": 1000000,
        "stop_timesteps": 5000000,
        "stop_reward": 10000000,
        # 硬件资源
        "num_envs_per_worker": 1,
        "num_cpus_per_worker": 1,
        "num_gpus_per_worker": 0,
        "num_cpus": 10,
        "num_gpus": 0,
        # ckpt
        "checkpoint_config": {
            "num_to_keep": 3, "checkpoint_frequency": 1, "checkpoint_score_attribute": 'training_iteration'
        },
        "log_path": os.path.join(WORK_DIR, "log")  # 日志路径args
    }  # 训练默认设置
    if len(args['agent_map']) == 0:
        args['rollout_fragment_length'] = 1000
        train_config['train_batch_size'] = 1000
```

图 18 训练模式设置

train_mode：local 为本地训练，支持单房间训练；normal 为不可调试下的训练模式，支持多房间训练，如果需要进行多房间训练需要提前将 train_mode 修改为 normal 模式，如果在 local 模型下进行多房间训练会报错。

num_workers：训练房间数量，单方间模式下设置为 1，多房间下选手可以根据需求自定义设置房间数量，最大上限为 4

- 单/多房间训练效果

单房间训练日志刷新效果如下图。

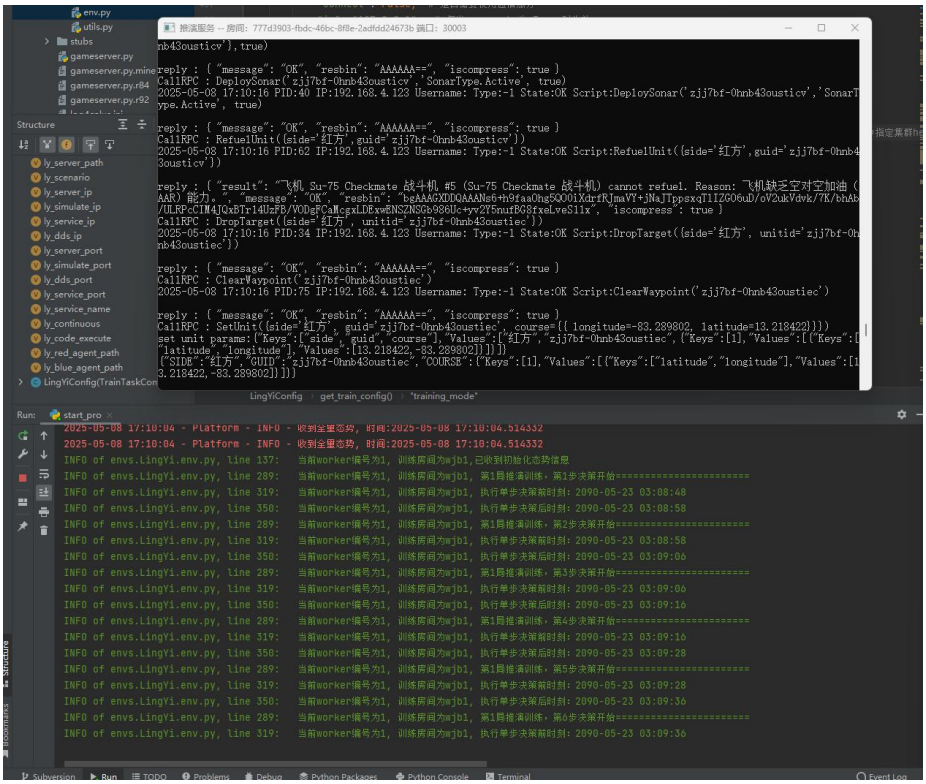

图 19 单房间训练效果

多房间训练日志刷新效果如下图。

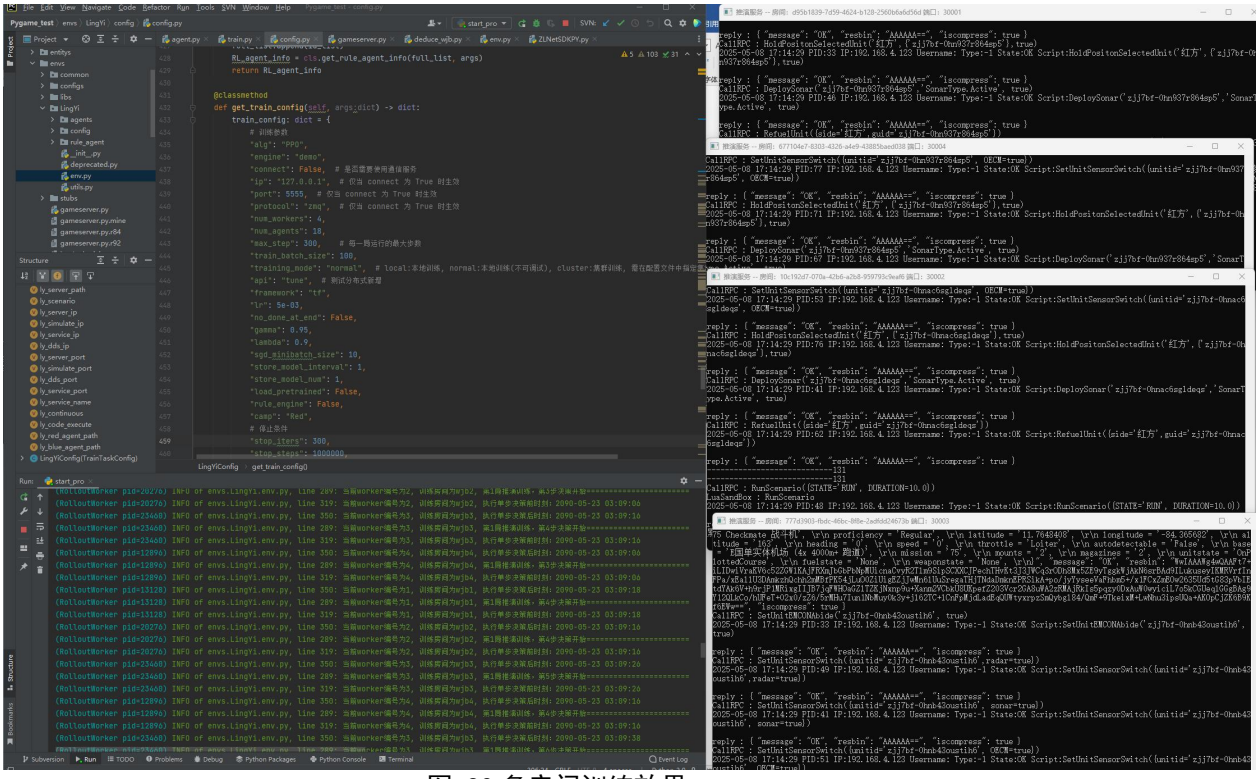

图 20 多房间训练效果

2.3 训练结果可视化

选手可以在线查看训练效果，如可以查看奖励函数变化曲线，策略熵的变化曲线等。

2.3.1 环境配置

TensorBoard 包是 TensorFlow 生态系统中的可视化工具包，主要用于机器学习实验的可视化分析和调试，支持训练过程监控。

检查 python 环境是否安装中 TensorBoard 包。若没有安装 TensorBoard 包，则输入 pip install TensorBoard 安装运行 TensorBoard 所需要的插件包。

2.3.2 训练结果保存

交互环境中，输入 Tensorboard --logdir=“训练结果保存路径”进行训练结果按路径保存，一般路径配置为 results\LingYi\train_results\，如下图所示：

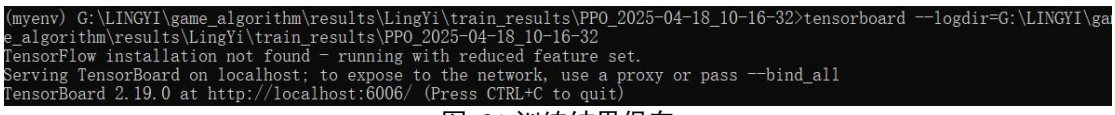

图 21 训练结果保存

2.3.2 训练结果查看

在浏览器打开显示的本地端口号(例如 http://localhost:6006/)，在页面上可以看到训练结果，如奖励函数变化曲线，策略熵的变化曲线等，如下图。

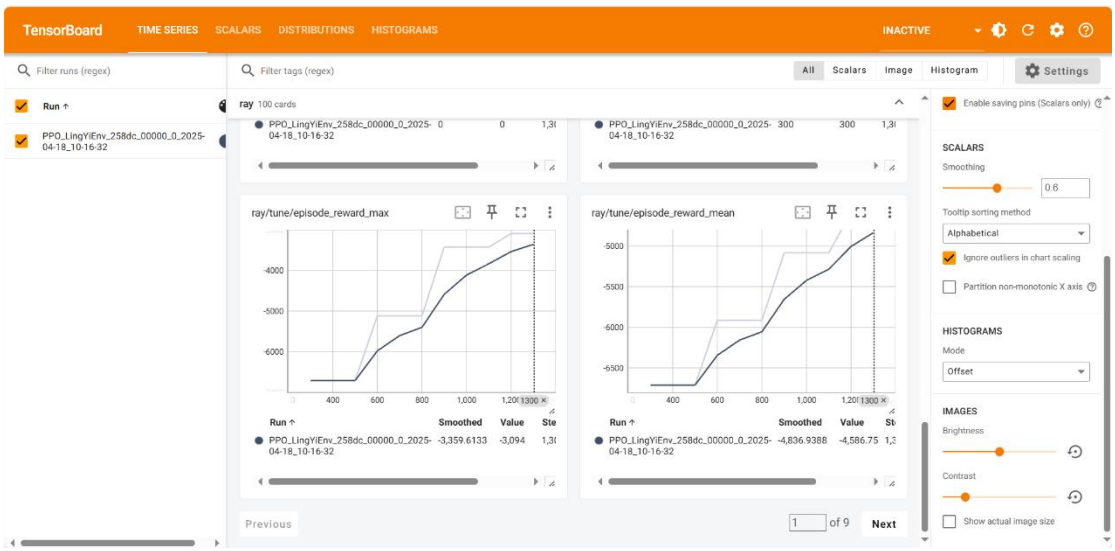

图 22 训练结果

3 智能体使用-灵弈智能体调度

3.1 智能体调度功能介绍

当选手完成智能体开发与调试后，在比赛过程中通过灵弈客户端智能体调度功能实现智能体的使用，智能体调度主要包括智能体编辑和智能体消息输出两部分功能。

图 23 智能体调度功能

3.1.1 智能体编辑

智能体编辑可以实现对于智能体的管理功能，主要包括智能体导入、智能体运行、智能体元素添加（实体单元、目标、参考点）、智能体元素展示、智能体配置保存等，智能体调度界面如下图所示。

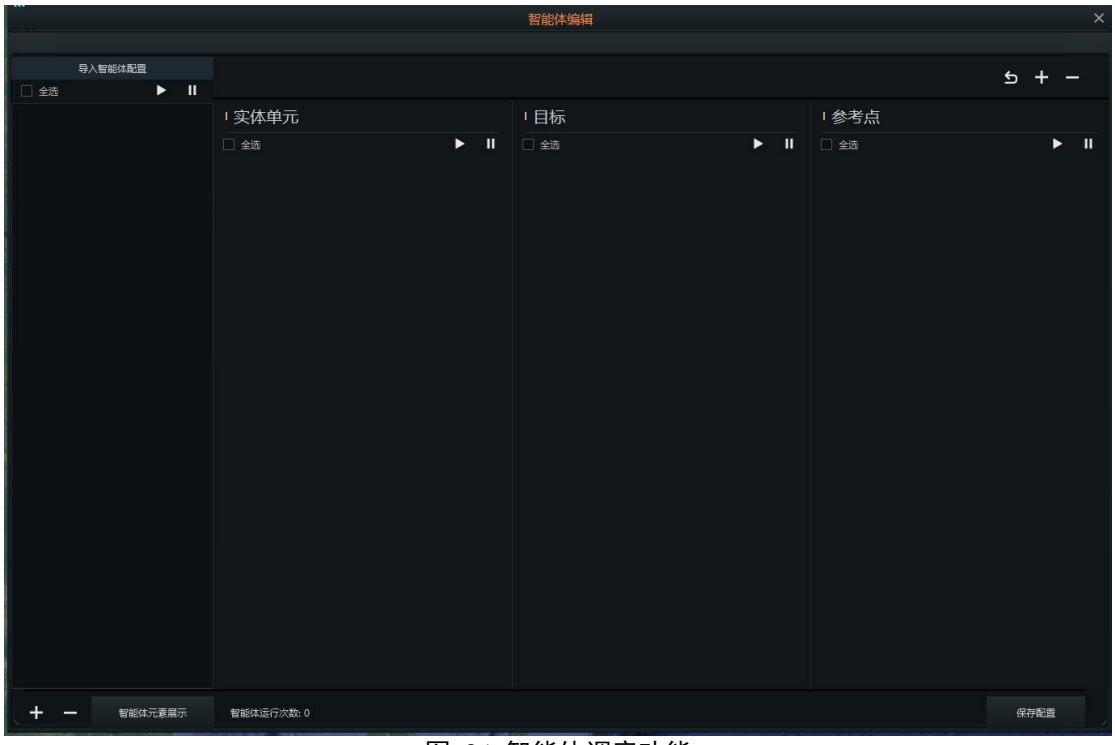

图 24 智能体调度功能

在智能体编辑界面的左侧为智能体的功能界面，可以实现添加、启动、删除智能体。添加智能体有两种方式：导入单个智能体和导入智能体配置。导入单个智能体通过点击智能体编辑界面左下角的“＋”实现，点击“＋”后选择需要导入的智能体文件即可。

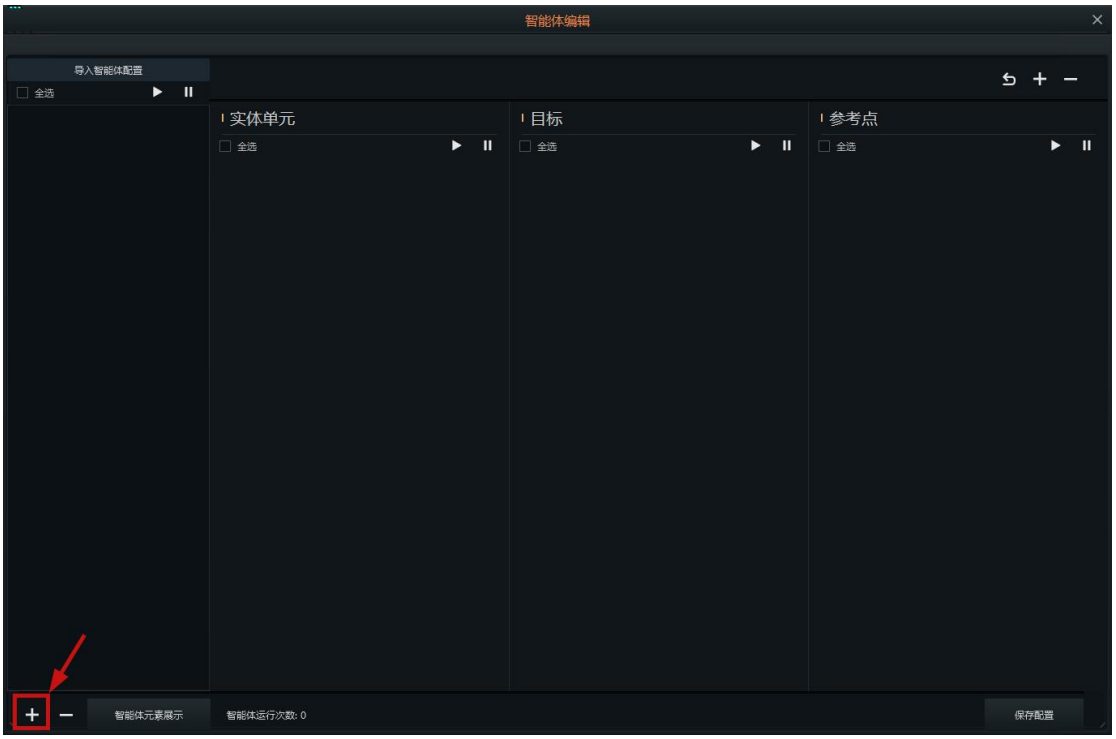

图 25 导入单个智能体功能

智能体编辑功能支持对于当前的智能体方案保存（保存当前智能体和当前智能体元素），点击“保存配置”即可。

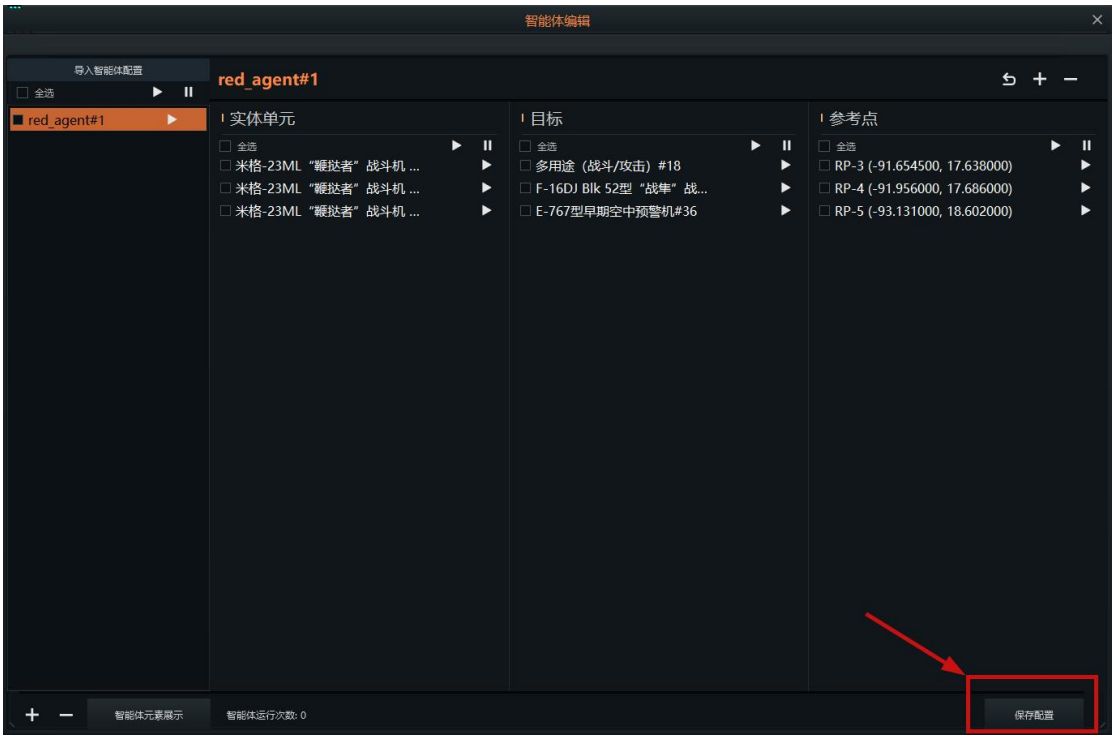

图 26 保存智能体配置功能

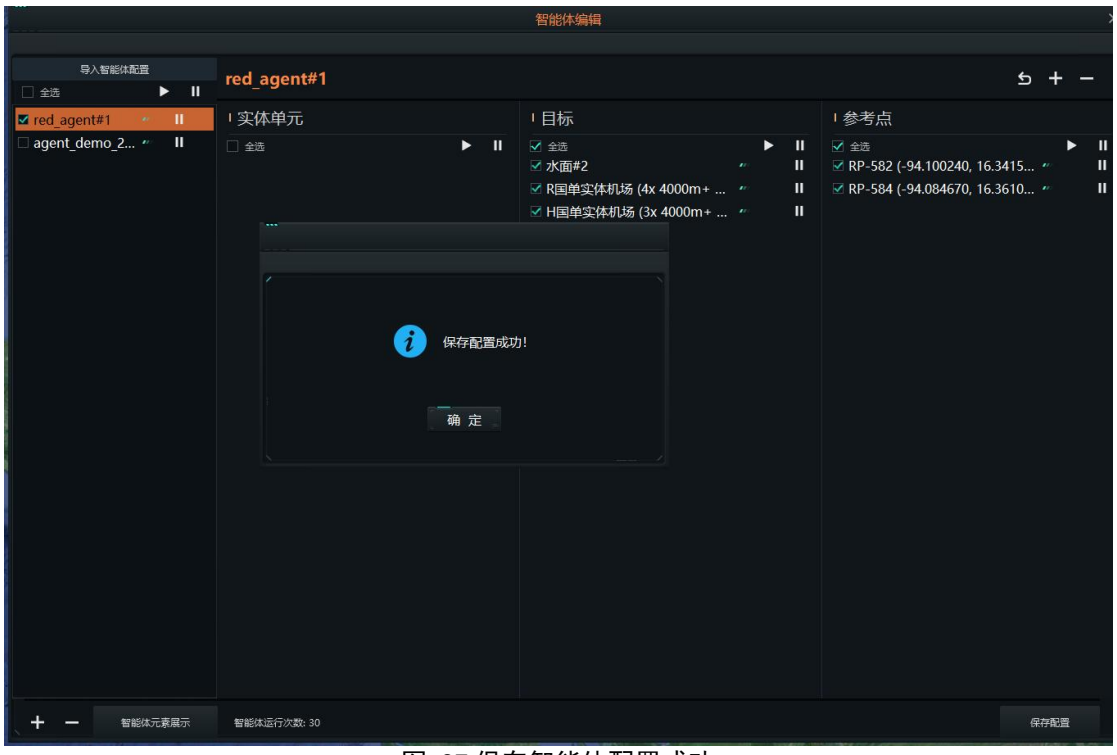

图 27 保存智能体配置成功

当保存智能体配置后，选手可以通过点击“导入智能体配置”功能一键导入之前保存的智能体配置方案。

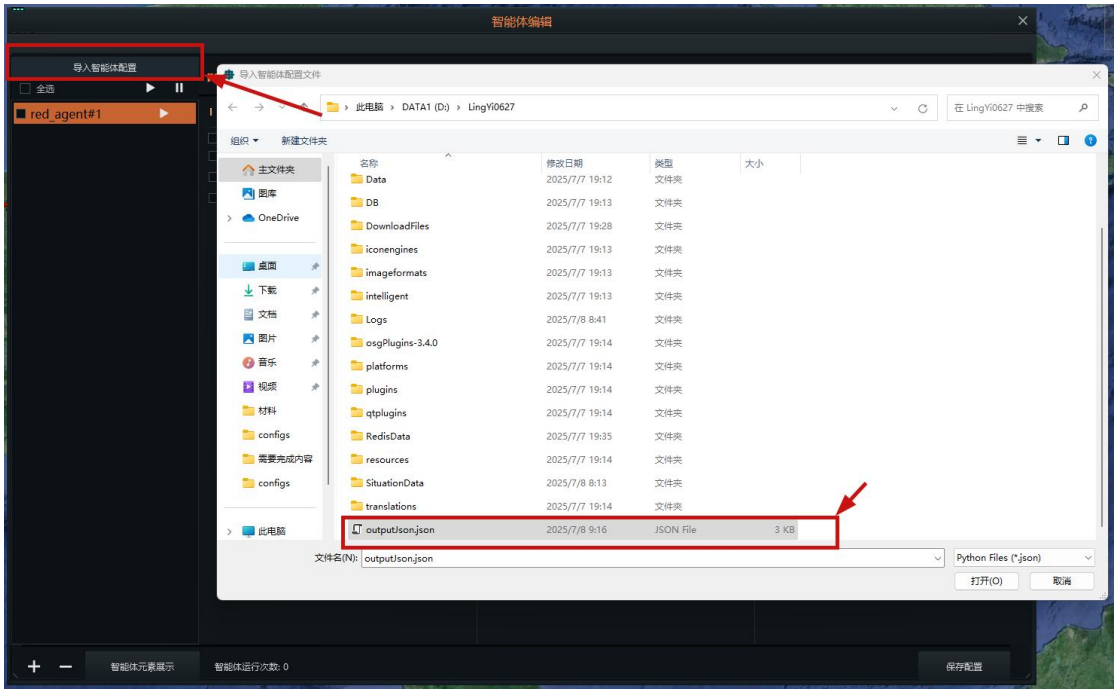

图 28 导入智能体配置功能

3.1.2 智能体消息输出

智能体消息输出功能可以实现 5 种消息展示（纯文本、重要情报、圆形区域、箭头、多边形区域），在智能体调度中我们为选手提供了智能体消息输出接口，选手在开发完智能体后可以通过智能体消息输出接口实现 5 种类型消息的展示。

表 1 智能体消息输出接口表

类型	作用	调用接口（Agent 类中）	输入参数
纯文本	将文本信息展示在智能体消息输出界面	self.agent_message_text(text='纯文本显示输出')	text：文本信息
重要情报	对重要情报进行可视化标注，重要情报等级分为三级，分别用红色、橙色、白色展示	self.agent_message_intelligence(level, contact_id, text)	level：重要情报等级（ level=1：一级情报，核心情报信息，显示为红色 level=2：二级情报，重要情报信息，显示为橙色 level=3：三级情报，一般情报信息，显示为白色） contact_id：重要情报单元 id text：文本信息
圆形区域	可视化表示圆形区域，多用于展示敌我方探测或打击范围。	self.agent_message_unit_range(unit_id,dbrange,color_type, line_type, text)	unit_id：圆形区域跟随的单元 id，可以跟随己方单元或者是敌方单元 dbrange：显示区域半径 km color_type：颜色类型，1：白色 2：橙色 3：红色 4：蓝色 5：紫色 6：黄色 7：绿色 line_type：线段类型，0：实线 1：虚线 text：文本信息
箭头	用于展示敌方的进攻意图	self.agent_message_arrow(arrow_name,color_type,line_type,points_list, text)	arrow_name：箭头名称 color_type：颜色类型，1：白色 2：橙色 3：红色 4：蓝色 5：紫色 6：黄色 7：绿色 line_type：线段类型，0：实线 1：虚线 points_list ： 形成箭头的点[(lon1,lat1),(lon2,lat2)...]，列表最后一组经纬度为箭头指向 text：文本信息
多边形区域	用于可视化展示区域，多用于自定义封锁区、威胁区、预警区等。	self.agent_message_area(area_name, points_list)	area_name：多边形区域名称 points_list 形成多边形区域的点[(lon1,lat1),(lon2,lat2) ,(lon3,lat3)...] text：文本信息

智能体如果调用消息输出接口时，会将智能体消息输出文本消息展示到灵弈客户端界

面，如下图所示。

图 29 智能体消息输出功能

3.1.3 智能体决策间隔设置

智能体 Agent 类中支持对于智能体决策间隔的设置，可以调用 Agent 的父类 Player 中 self.set_agent_decision_time()函数实现，输入参数为智能体决策间隔，单位为 s，决策间隔时间为推演时间。

3.2 智能体调度使用

当通过智能体开发与训练框架完成智能体的开发后，选手通过灵弈客户端界面的智能体调度功能进行对于智能体的调度使用。智能体调度使用流程如下：

1、首先，将智能体文件由智能体开发与训练框架 agent 文件夹内文件拷贝至灵弈客户端\intelligent\ai\agent 内。

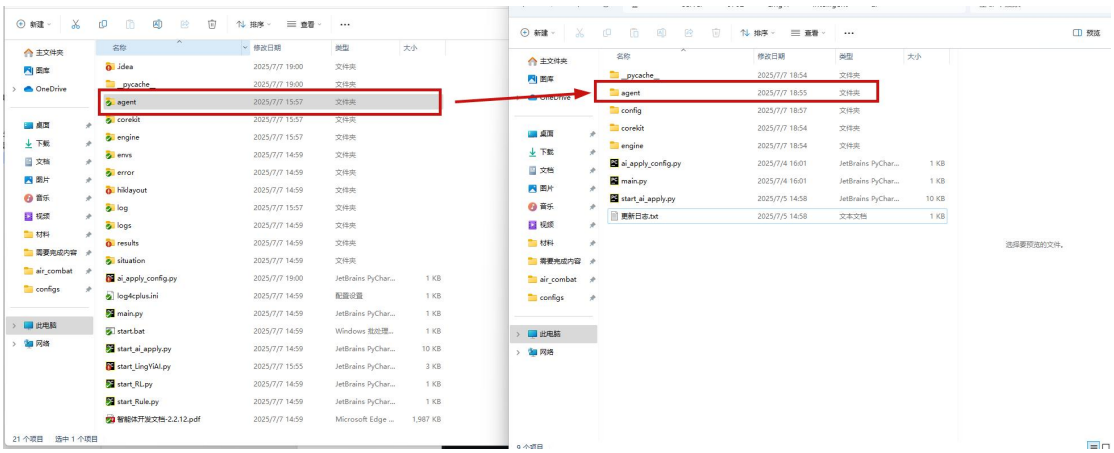

图 30 智能体文件迁移

2、打开“智能体调度-智能体编辑”界面，通过点击左下角“＋”或者点击“导入智能体配置”按钮导入智能体。

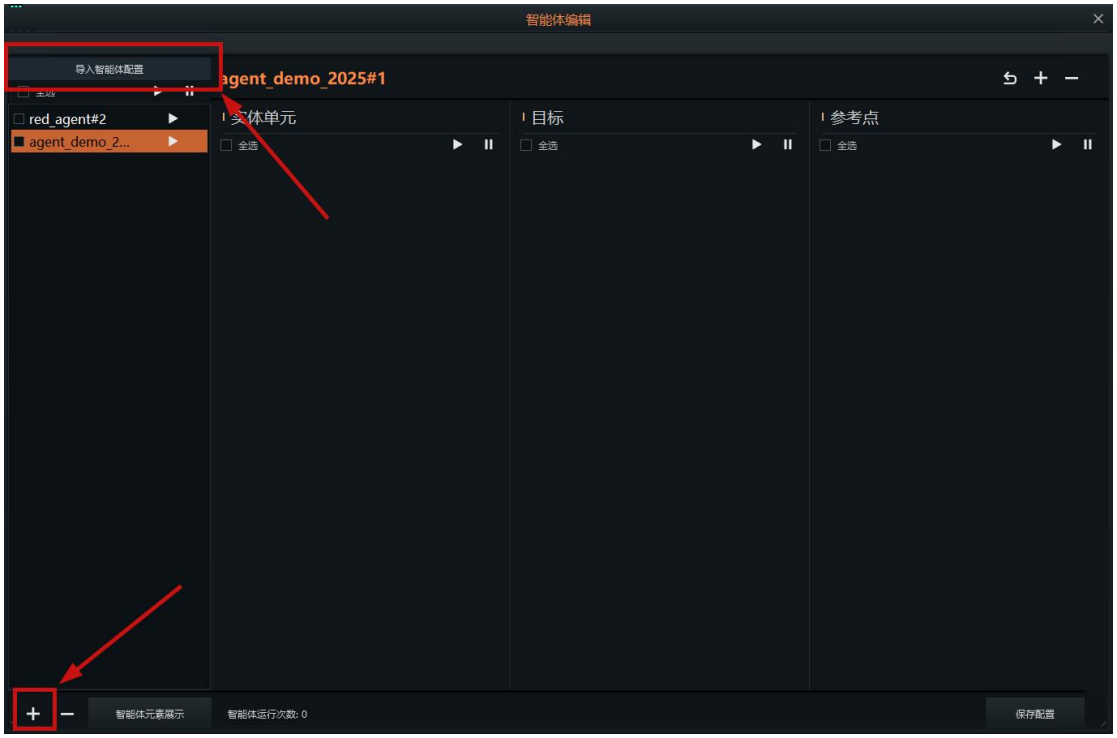

图 31 导入智能体

3、导入智能体后，可以对智能体中的元素进行添加或删除操作。智能体元素包括实体单元、目标、参考点。

图 32 智能体元素编辑

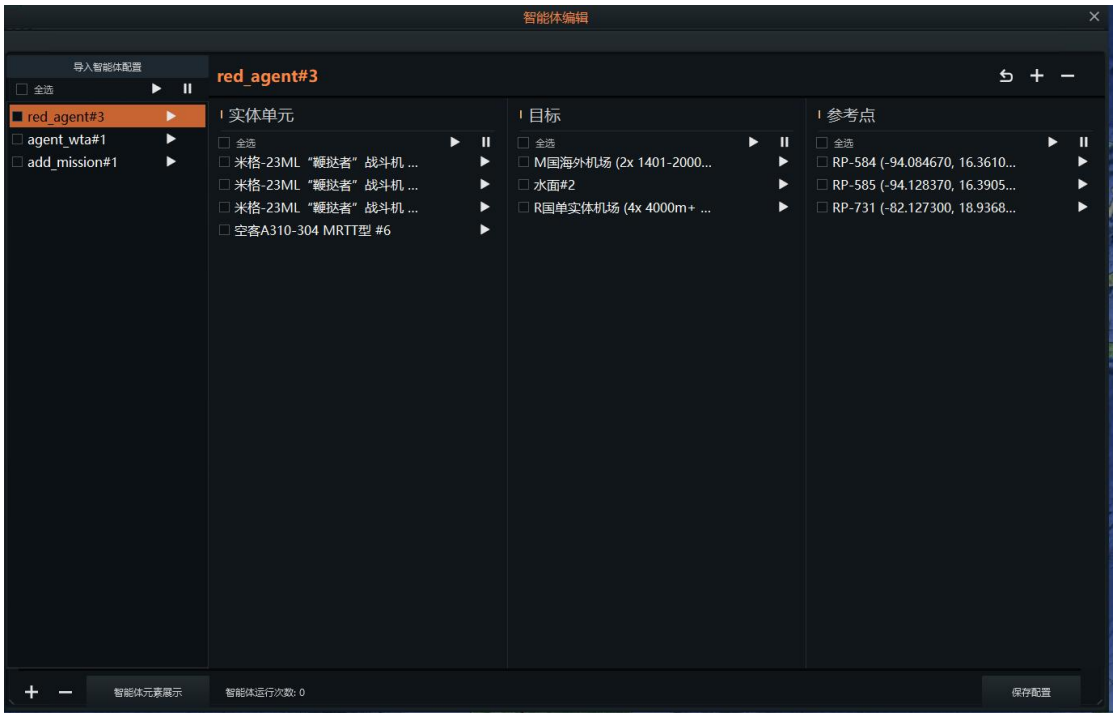

图 33 完成智能体元素编辑

4、完成智能体元素的编辑后，点击智能体和智能体内元素的按钮开始运行智能体。需要注意智能体元素在运行状态下智能体才能够控制该元素；如果智能体中某个元素是停止状态时，智能体不对该元素进行控制。

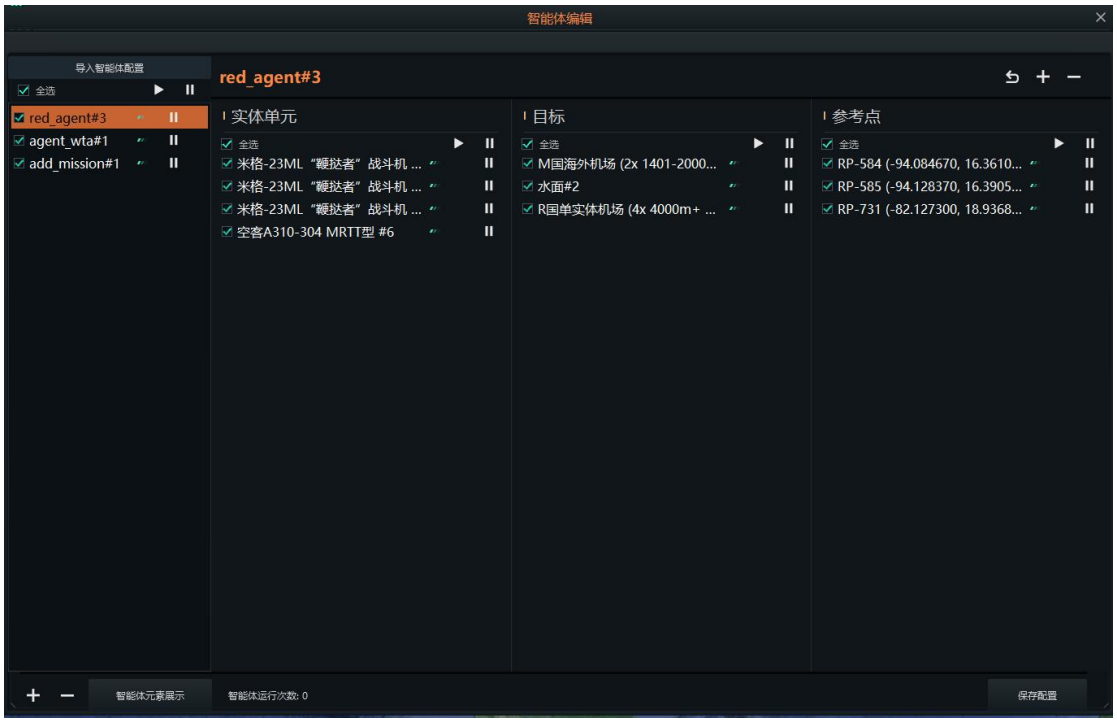

图 34 运行智能体和智能体元素

5、运行过程中智能体编辑界面会记录智能体的运行次数，智能体运行次数即 Agent 类

中 self.step 方法的调用次数：

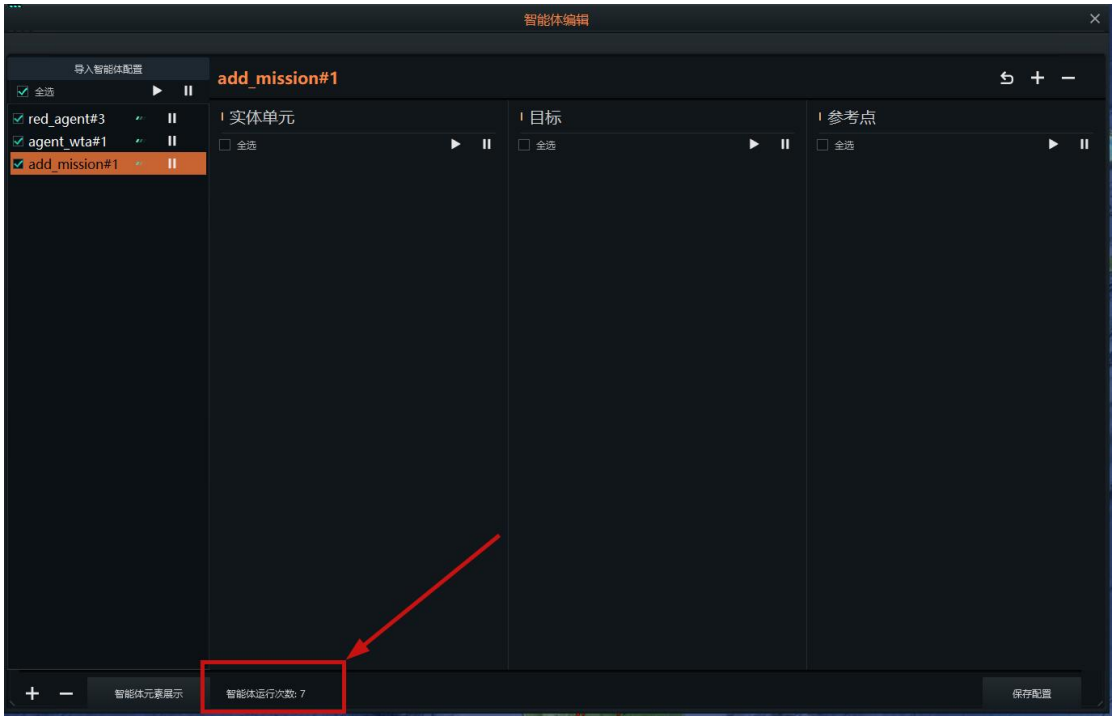

图 35 智能体运行次数显示

6、智能体运行过程中智能体消息输出界面会展示选手调用智能体消息输出接口的文本信息：

图 36 智能体消息输出显示

7、智能体运行的执行效果如下：

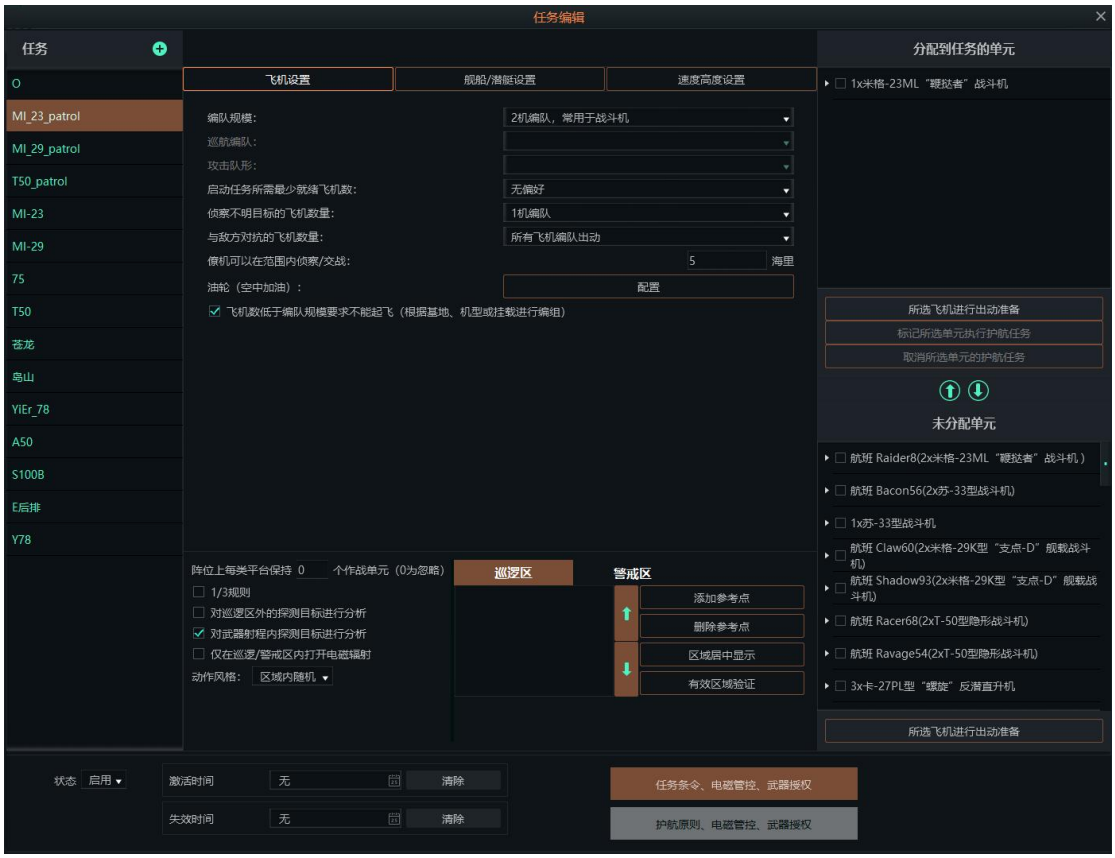

图 37 智能体运行效果-任务添加

3.3 智能体调度调试方法

选手在使用智能体调度功能时可能存在调试智能体的情况，例如智能体异常退出、智能体运行效果异常、调试智能体消息输出和智能体决策间隔设置等。本节提供选手在智能体调度使用时调试智能体的方法。智能体调度调试前需要生成运行所需的配置文件，配置文件由灵弈客户端生成。智能体调度调试方法如下：

1、启动灵弈服务端和客户端，点击客户端智能体调度：

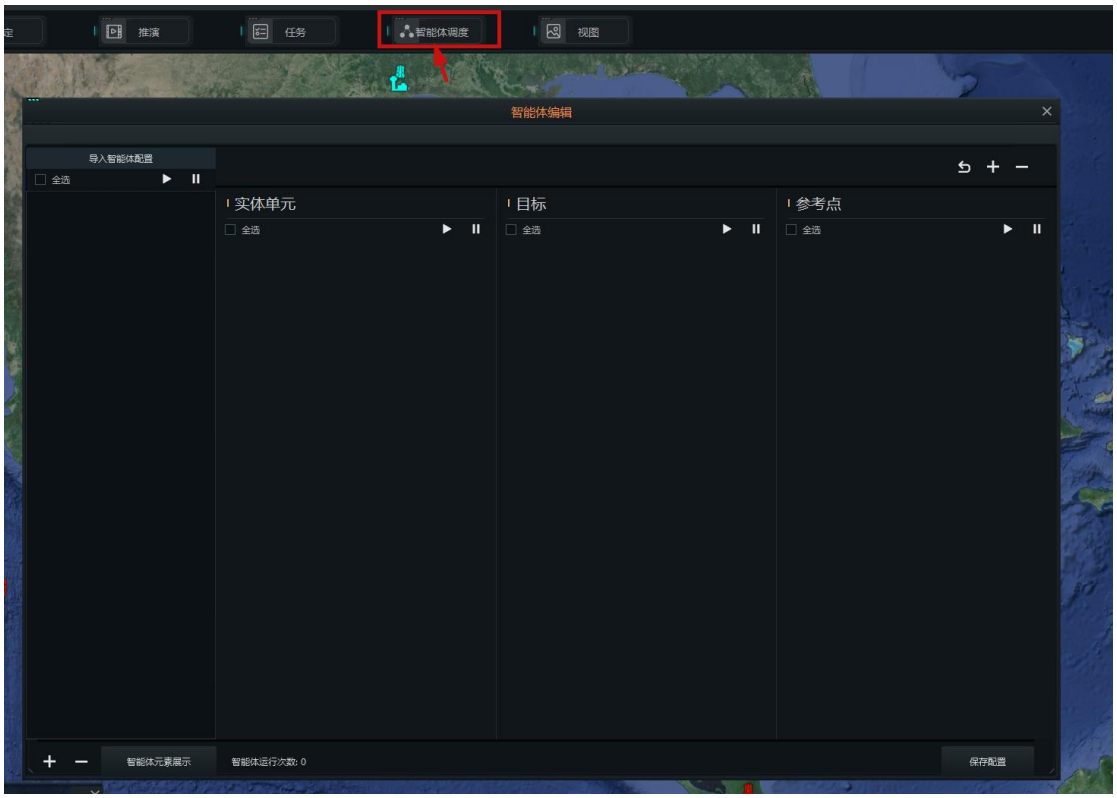

图 38 点击智能体编辑

2、导入需要调试的智能体算法，并选择智能体元素（实体单元、目标、参考点）。

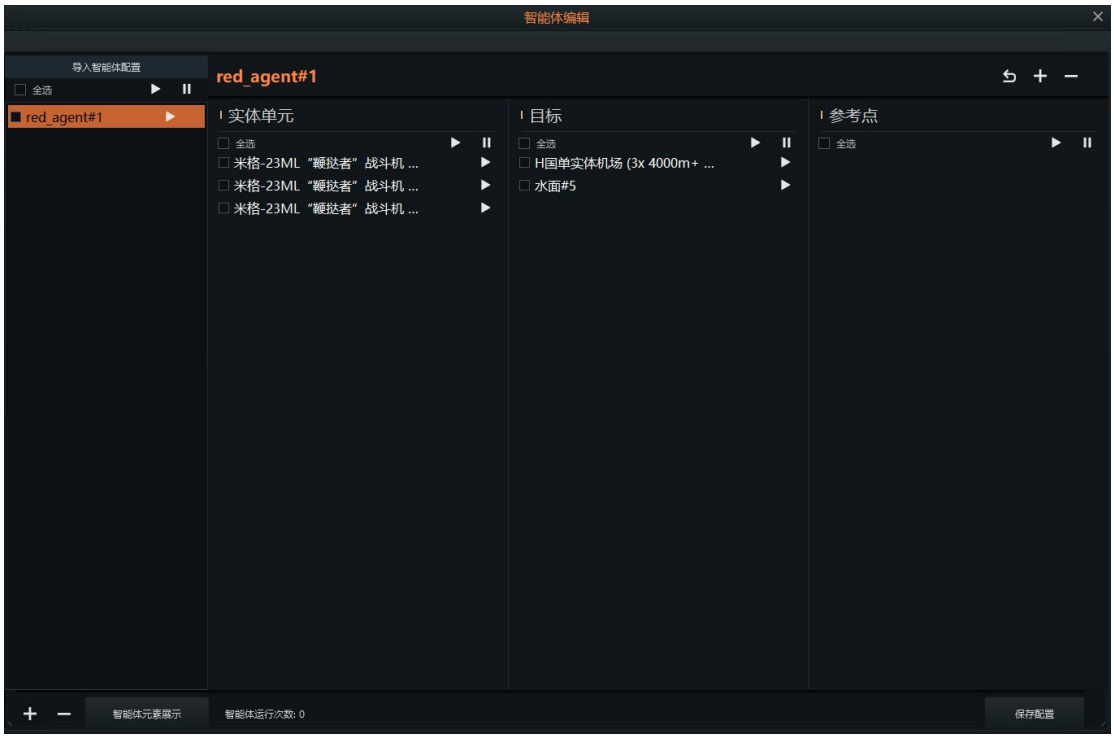

图 39 导入需要调试的智能体

3、点击启动智能体，随后停止智能体，在客户端 intelligent\ai\config 路径下生成了一个

与智能体名称相同的文件：

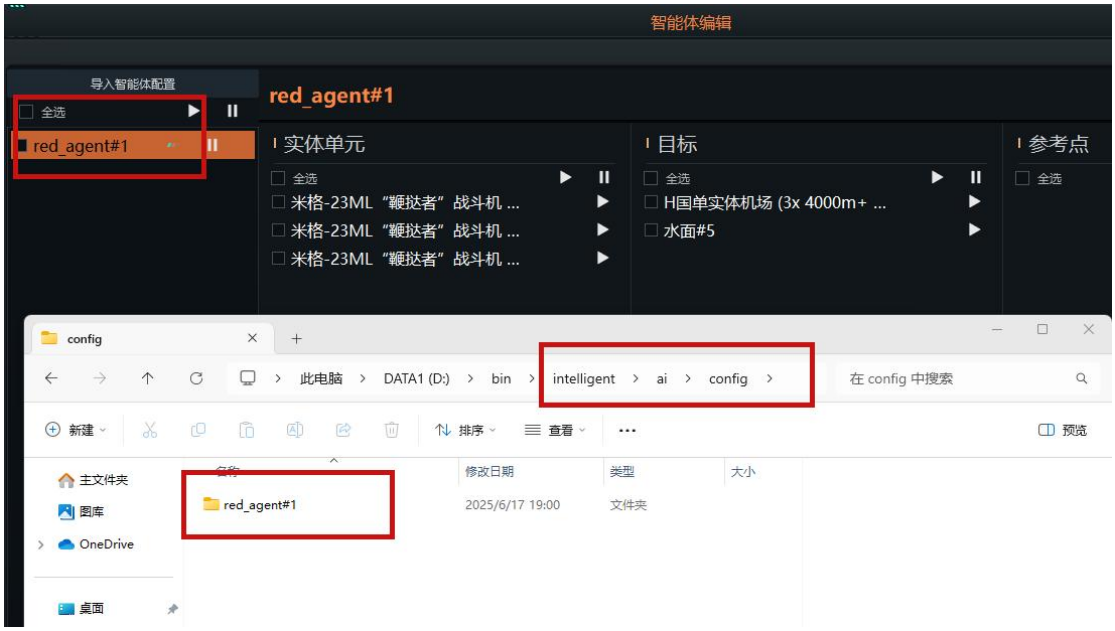

图 40 智能体运行配置文件

4、进入此文件夹，复制文件夹路径：

图 41 智能体运行配置文件所在路径

5、将此路径复制在 ai_applyconfig.py 内的 GLOBAL_CONFIG_PATH 变量中

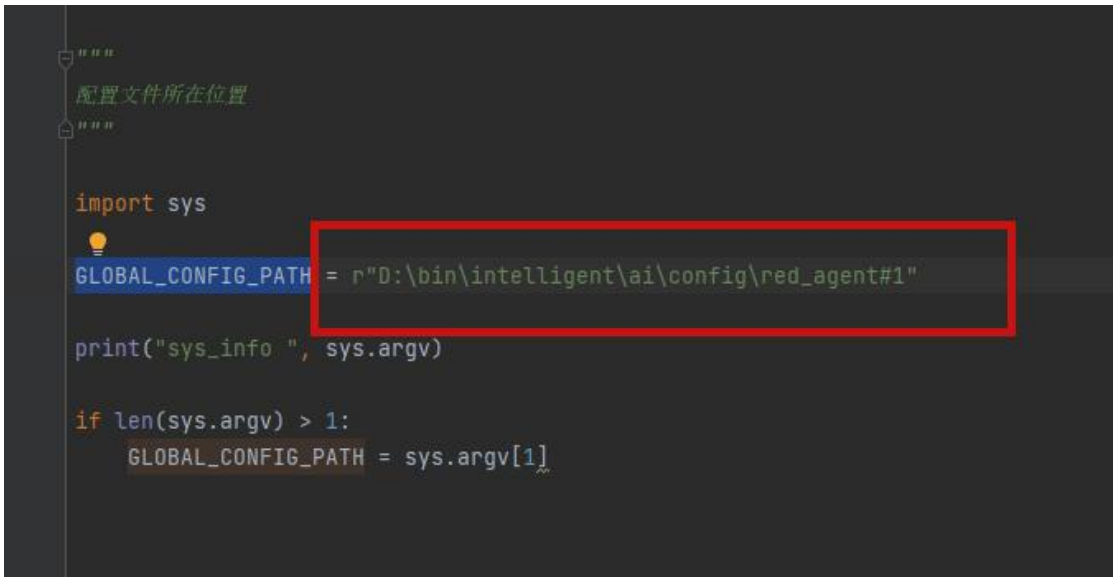

图 42 修改 GLOBAL_CONFIG_PATH 变量

6、最后，使用 debug 运行 main.py 即可进行调试：

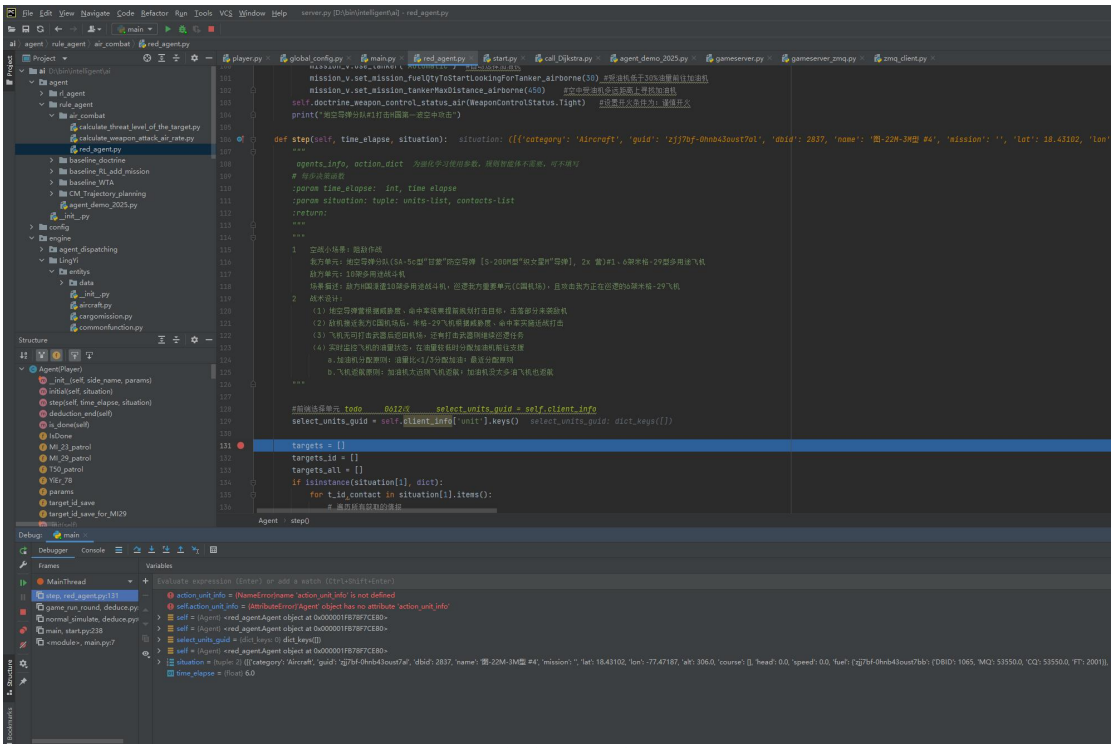

图 43 智能体调度调试

附件：

附件 1：智能体控制接口

序号	接口	类名	说明
1	aircraft.py	飞机类	1、获取飞机挂架和挂载上的武器信息 2、获取精简信息, 提炼信息进行决策 3、获取飞机当前油量 4、获取飞机当前飞行状态 5、获取飞机状态 6、设置飞机的武器挂载---需考虑基地武器是否支持飞机挂载，有挂载耗时； 7、飞机快速出动与否 8、飞机中止出动 9、飞机燃油计算 10、设置当前单元（飞机）的飞行高度跟随地形 11、卸货，飞机等卸掉物资 12、卸载选定物资 13、设置飞机的飞行高度层
2	cargomission.py	货运任务类	1、设置任务的出航油门 2、设置任务的阵位油门 3、设置任务的出航高度 4、设置任务的阵位高度 5、设置卸载货物 DBID 6、设置卸载货物数量 7、货运任务区域按顺序排列的参考点名
3	commonfunction.py	通用工具函数集合	通用工具函数集合，包含数据处理、文件操作、时间转换等功能 示例: 任务 GUID 解析、JSON 文件操作、想定时间处理等
4	element.py	不常用	元素类 飞机，舰船等作战单元，武器、传感器、天气实体，触发器对象等的基类
5	facility.py	地面设施类	地面设施类 完成武器库存管理（全武器获取/挂架状态监控）、作战信息提炼（关键数据提取）、阵位保持控制三大核心能力建设 1、获取地面设施弹药库的所有武器 2、获取单元挂架武器信息, 以列表形式返回 3、获取精简信息, 提炼信息进行决策

			4、保持阵位-所选单元
6	ferrymission.py	转场任务类	1、设置转场规则 2、设置任务编队规模 3、是否飞机数低于编队规模不允许起飞 4、设置任务所需最少飞机数 5、飞机转场油门 6、飞机转场高度 7、潜艇转场深度 8、设置转场地形跟随 9、舰艇转场油门 10、潜艇转场油门
7	gameunit.py	不常用	元素类
8	geo.py	地理信息库	1、求地面两点的水平距离 2、获取三维直线距离 3、获取直线距离 4、角度调整为 0-360 度以内 5、获取角度差 6、获取 point1 指向 point2 的方位角 7、WGS84 坐标系(x, y, z) 转 大地坐标系(lat, lon, alt) 8、大地坐标系(lat, lon, alt) 转 WGS84 坐标系(x, y, z) 9、从地球海拔水平上，选一角度出发一定距离后，获取新的点 10、获取地球上两点的差值点 11、获取多边形的中心点 12、判断一个点是否在该多边形内
9	global_util	不常用	元素类
10	group.py	编组类	1、获取精简信息, 提炼信息进行决策 2、设置编组单元方位 3、设置编队领队
11	mineclearmission	排雷任务类	1、三分之一规则 2、设置任务编队规模 3、是否飞机数低于编队规模不允许起飞 4、设置任务舰船/潜艇编队规模 5、是否舰船/潜艇数低于编队规模不允许出发 6、设置清雷任务所需最少飞机数 7、设置任务的出航油门 8、设置任务的阵位油门 9、设置任务的出航高度 10、设置任务的阵位高度 11、设置出航地形跟随 12、阵地地形跟随 13、设置排雷任务的区域，按顺序排列的参考点名

12	minemissio n	部雷任务类	1、三分之一规则 2、水雷接触保险延迟 3、设置任务编队规模 4、是否飞机数低于编队规模不允许起飞 5、设置任务舰船/潜艇编队规模 6、是否舰船/潜艇数低于编队规模不允许出发 7、设置任务所需最少飞机数 8、设置任务的出航油门 9、设置任务的阵位油门 10、设置任务的出航高度 11、设置任务的阵位高度 12、设置出航地形跟随 13、设置阵位地形跟随 14、设置部雷任务的区域 15、设置出航地形跟随 16、阵地地形跟随
13	mission.py	基础任务类	1、设置任务 2、是否启用任务 3、设置、删除任务开始时间 4、设置任务：删除任务结束时间 5、设置任务雷达是否打开 6、设置任务将实体分配到任务中来 7、返回任务已分配的单 8、任务取消某单元的任务分配 9、任务设置出航航线或者阵位航线，或者设置返航航线 10、修改任务名
14	mission_co mmon.py	任务中包含加油机相关设置的任务基类	1、设置任务加油/补给 2、设置任务加油规划-选择加油机设置 3、任务规划细节：设置没有油轮执行发射任务(加油机没法到位的情况下启动任务） 4、设置作为加油源的要使用的任务数组 5、任务规划细节：设置所需油轮最低数量 6、任务规划细节：设置空中油轮最低数量 7、任务规划细节：就位油轮最低数量 8、任务执行细节：每个油罐车排队最大接受器数

			9、任务执行细节：接受者开始寻找油轮燃料百分比 10、任务执行细节：加油机遵循受油机的飞行计划 11、空中受油机可以距离油轮在
15	operatorbase.py	条令类	1、设置条令 2、条令中，电磁管控设置，设置雷达开关机 3、条令中，电磁管控设置，设置电子干扰传感器开关机 4、条令中，电磁管控设置，设置声呐开关机 5、设置条令中电磁管控与上级一致，推演方条令无法设置为 Inherit 6、条令设置，是否使用核武器； 推演方条令无法设置为 Inherit 7、武器控制状态，对空；推演方条令无法设置为 Inherit 8、武器控制状态，对水面；推演方条令无法设置为 Inherit 9、武器控制状态，对潜；推演方条令无法设置为 Inherit 10、武器控制状态，对地；推演方条令无法设置为 Inherit 11、攻击时忽略计划航线设置；推演方条令无法设置为 Inherit 12、接战模糊位置目标；推演方条令无法设置为 Inherit 13、接战临机出现目标；推演方条令无法设置为 Inherit 14、受到攻击忽略电磁管控；推演方条令无法设置为 Inherit 15、鱼雷射程；推演方条令无法设置为 Inherit 16、自动规避；推演方条令无法设置为 Inherit 17、加油/途中补给；推演方条令无法设置为 Inherit 18、加油/途中补给；推演方条令无法设置为 Inherit 19、给盟军加油/途中补给；推演方条令无法设置为 Inherit 20、空战节奏；推演方条令无法设置为 Inherit 21、快速周转；推演方条令无法设置为 Inherit 22、燃油状态，预先规划；推演方条令无法设置为 Inherit 23、燃油状态，返航；推演方条令无法设置为 Inherit 24、武器状态，预先规划；推演方条令无法设置为 Inherit 25、武器状态-返航；推演方条令无法设置为 Inherit 26、空对地扫射；推演方条令无法设置为 Inherit 27、丢弃武器；推演方条令无法设置为 Inherit 28、超视距交战；推演方条令无法设置为 Inherit 29、反水面战模式下使用防空导弹；推演方条令无法设置为 Inherit 30、反水面作战行动 与目标保持距离；推演方条令无法设置为 Inherit

			31、反水面作战行动 导航；推演方条令无法设置为 Inherit 32、反潜作战行动，规避搜索；推演方条令无法设置为 Inherit 33、反潜作战行动，探测到威胁进行下潜；推演方条令无法设置为 Inherit 34、反潜作战行动，充电电池 运输/站点；推演方条令无法设置为 Inherit 35、反潜作战行动，充电电池 进攻/防守；推演方条令无法设置为 Inherit 36、反潜作战行动，使用 AIP 技术；推演方条令无法设置为 Inherit 37、反潜作战行动， 吊放声呐；推演方条令无法设置为 Inherit 38、反潜作战行动， 导航；推演方条令无法设置为 Inherit 39、陆战导航；推演方条令无法设置为 Inherit 40、设置条令的武器使用规则 41、武器使用规则--齐射武器数 42、武器使用规则--齐射发射架数 43、武器使用规则 - -自动开火距离 44、武器使用规则--自动防御距离 45、满足如下条件时撤退 - -毁伤程度大于；推演方条令无法设置为 Inherit 46、满足如下条件时撤退--燃油少于；推演方条令无法设置为 Inherit 47、满足如下条件时撤退--主要攻击攻击武器至少处于；推演方条令无法设置为 Inherit 48、满足如下条件时撤退--主要防御武器至少；推演方条令无法设置为 Inherit 49、满足如下条件时重新部署--毁伤程度小于；推演方条令无法设置为 Inherit 50、满足如下条件时重新部署--燃油至少处于；推演方条令无法设置为 Inherit 51、满足如下条件时重新部署--主要攻击武器处于；推演方条令无法设置为 Inherit 52、满足如下条件时重新部署--主要防御武器处于；推演方条令无法设置为 Inherit 53、获取武器使用规则
16	patrolmissio	巡逻任务类	1、设置任务加油/补给

	n.py		2、设置任务加油规划-选择加油机设置 3、任务规划细节：设置没有油轮执行发射任务(加油机没法到位的情况下启动任务） 4、设置作为加油源的要使用的任务数组 5、任务规划细节：设置所需油轮最低数量 6、任务规划细节：设置空中油轮最低数量 7、任务规划细节：就位油轮最低数量 8、任务执行细节：每个油罐车排队最大接受器数 9、任务执行细节：接受者开始寻找油轮燃料百分比 10、任务执行细节：加油机遵循受油机的飞行计划 11、空中受油机可以距离油轮在
17	player.py	玩家类	1、通过任务名获取任务对象 2、获取实体 3、获取情报信息 4、获取当前天气 5、获取某点（纬经度）高程 6、添加一个或多个参考点 7、移动参考点的位置 8、删除参考点 9、批量增加多个参考点 10、分配多个单元到任务 11、清除单元所有航路点 12、创建巡逻任务 13、增加巡逻任务的警戒区 14、创建打击任务 15、创建支援任务 16、创建转场任务 17、创建布雷任务 18、创建扫雷任务 19、创建货运任务 20、删除任务 21、给任务分配单元 22、将若干单元取消分配任务 23、将某单元取消分配任务 24、删除禁航区或封锁区域

			25、添加禁航区 26、添加封锁区 27、添加禁航区或封锁区 28、修改禁航区和封锁区 29、将多个单元作为一个编队 30、分离编组单元 31、将单元移除出编队 32、编队添加一个单元 33、飞机单机出动 34、飞机编组出动 35、飞机准备/备战，飞机挂载武器 36、船舶/潜艇单独出航 37、船舶/潜艇编队出航 38、船舶/潜艇中止出航 39、保持所有单元阵位 40、单元选择新基地/新港口 41、实体返航 42、批量自动攻击某目标 43、开火条件检查 44、情报标识重命名 45、情报标识标记位置 46、标识情报阵营 47、断开若干情报标识 48、修改单元名称 49、战前规划时间，可设置飞机挂载 50、战前规划时间，对战开始之前，设置本单元经纬度位置或高度或深度或航向 51、智能体消息输出接口——纯文本 52、智能体消息输出接口——重要情报（202507 版本未实现） 53、智能体消息输出接口——圆形区域显示（跟随单元）（202507 版本未实现） 54、智能体消息输出接口——箭头（202507 版本未实现） 55、智能体消息输出接口——多边形区域（202507 版本未实现）
18	satellite.py	卫星类	1、获取精简信息，提炼信息进行决策
19	ship.py	舰船类	1、获取单元武器信息，以列表形式返回 2、获取精简信息，提炼信息进行决策

20	strikemission.py	打击任务类	任务中包含加油机相关设置的任务基类 任务中：打击任务，巡逻任务、支援任务、布雷、清雷，转场任务：油轮配置 1、设置任务加油/补给 2、设置任务加油规划-选择加油机设置 3、任务规划细节：设置没有油轮执行发射任务(加油机没法到位的情况下启动任务） 4、设置作为加油源的要使用的任务数组 5、任务规划细节：设置所需油轮最低数量 6、任务规划细节：设置空中油轮最低数量 7、任务规划细节：就位油轮最低数量 8、任务执行细节：每个油罐车排队最大接受器数 9、任务执行细节：接受者开始寻找油轮燃料百分比 10、任务执行细节：加油机遵循受油机的飞行计划 11、空中受油机可以距离油轮在
21	submarine.py	潜艇类	1、设置潜艇的期望深度
22	supportmission.py	支援任务	1、三分之一规则 2、阵位上每类平台保持几个 3、仅一次 4、仅在阵位上打开电磁辐射 5、导航类型 6、编队规模 7、是否飞机数低于编队规模不允许起飞 8、设置任务的飞机出航油门 9、设置任务的飞机阵位油门 10、设置任务的飞机出航高度 11、设置任务的阵位高度 12、支援任务区域按顺序排列的参考点名 13、支援任务设置一个加油周期后，加油队列为空时，加油机返回起降机场 14、支援任务设置加油机可给飞机加油数 15、设置任务所需最少飞机数 16、设置任务舰船/潜艇编队规模 17、是否舰船/潜艇数低于编队规模不允许出发 18、设置飞机出航地形跟随

			19、设置飞机阵地地形跟随 20、设置潜艇出航油门 21、设置潜艇出航潜深 22、设置潜艇阵位潜深 23、设置潜艇阵位油门 24、设置水面舰艇出航油门 25、设置水面舰艇阵位油门
23	unit.py	单元类	1、获取挂架上的武器信息 2、获取挂架统计的武器信息 3、获取单元航路点列表 4、获取挂架上武器的挂架 ID 和武器数量 5、获取当前单元目标列表 6、实体传感器面板， 实体是否遵循电磁管控条令 7、实体传感器面板， 实体雷达开关机 8、实体传感器面板， 实体声呐开关机 9、实体传感器面板， 实体电子干扰开关机 10、自动攻击目标 11、查询手动攻击信息 12、手动攻击，指定武器挂架、武器 ID 和武器数量打击某目标 13、手动攻击，指定武器 ID 和武器数量打击某目标 14、纯方位攻击 15、取消武器攻击，需运行推演后调用 16、单元释放一个箔条弹 17、箔条连续爆破 18、实体放弃某个之前设定的目标 19、实体放弃所有目标，脱离接战 20、反潜作战行动 21、设置单元的期望速度 22、设置实体油门 23、设置飞机或者潜艇手动控制高度 24、设置飞机单元的期望高度 25、设置当前单元（飞机）的飞行高度跟随地形 26、实体航线规划 27、单元清除航路点 28、给单元航路点设置速度高度

			29、单元的航路点设置传感器 30、设定参考点相对于本单元固定方位距离或者相对本单元航向固定旋转，或取消关联 31、实体返航 32、实体选择新基地/新港口 33、加油/补给 34、保持阵位-所选单元 35、分配加入到任务中 36、将单元分配为某打击任务的护航任务 37、将单元取消分配任务 38、设置单元优先挂载武器 39、空中行动、船舶出动业务装卸物资 40、获取某单元的航路点对象, 航路点对象可调用条令接口设置航路点条令 41、手动攻击里设置、取消武器攻击弹道 42、武器航线设置"
24	waypoint.py	航路类 不常用	1、获取航路点精简信息
25	weapon.py	武器类	1、获取武器打击目标 2、获取精简信息, 提炼信息进行决策