MaCA 环境说明

中国电子科技集团公司认知与智能技术重点实验室

2019 年 1 月

目录

1. MaCA 环境简介

1.1 环境概述

MaCA（Multi-agent Combat Arena）是多智能体对抗算法研究、训练、测试和评估的环境，可支持作战场景和规模自定义，智能体数量和种类自定义，智能体特征和属性自定义，支持智能体行为回报规则和回报值自定义等。此次发布的MaCA1.0 中提供了一个电磁空间对抗的多智能体实验环境，环境中预设了两种智能体类型：探测单元和攻击单元，探测单元可模拟 L、S 波段雷达进行全向探测，支持多频点切换；攻击单元具备侦察、探测、干扰、打击等功能，可模拟 X 波段雷达进行指向性探测，模拟 L、S、X 频段干扰设备进行阻塞式和瞄准式电子干扰，支持多频点切换，攻击单元还可对对方智能体进行导弹攻击，同时具有无源侦测能力，可模拟多站无源协同定位和辐射源特征识别。

MaCA 环境为研究利用人工智能方法解决大规模多智能体分布式对抗问题提供了很好的支撑，专门面向多智能体深度强化学习开放了 RL-API 接口。环境支持使用 Python 语言进行算法实现，并可调用 Tensorflow、Pytorch 等常用深度学习框架。

1.2 环境与算法交互关系

MaCA 环境支持红蓝双方智能算法在设定地图场景中进行对抗博弈，最终进行对抗的算法可以是基于规则直接实现的，也可以是基于强化学习等方法训练后得到的模型，环境中预制了简单的基于规则实现的对抗算法。设计 MaCA 环境的主要目的是促进多智能体强化学习方法在智能对抗领域的研究与应用，因此，建议研究人员采用的多智能体强化算法来完成对抗比赛。强化学习算法与环境交互过程可以分为两个阶段：一是训练阶段，通过收集算法与环境交互的实时数据来更新模型参数；一是训练完成之后通过调用训练好的模型来与其他对手进行对抗。

在训练阶段，算法与环境之间交互的数据包括：

1）环境->算法：

- ✓ **底层观测数据(Observation raw data)**：战场态势观测数据，可用于直接使用或研究者构建适用于自己算法的观测信息；
- ✓ **回报（Reward）**：每一帧决策动作执行后的回报值，MaCA 支持回报值自定义；
- ✓ **对战是否结束（Done）**：表征一局对战是否达到终止条件。

2）算法->环境：

- ✓ **动作（Action）**：算法控制己方每个作战单位输出的动作。

在模型调用阶段，训练好的模型与环境之间交互的数据包括：

1）环境->模型：

- ✓ **底层观测数据(Observation raw data)**：战场态势观测数据，构成满足模型要求的观测信息图层。

2）模型->环境：

- ✓ **动作（Action）**：算法控制己方每个作战单位输出的动作。

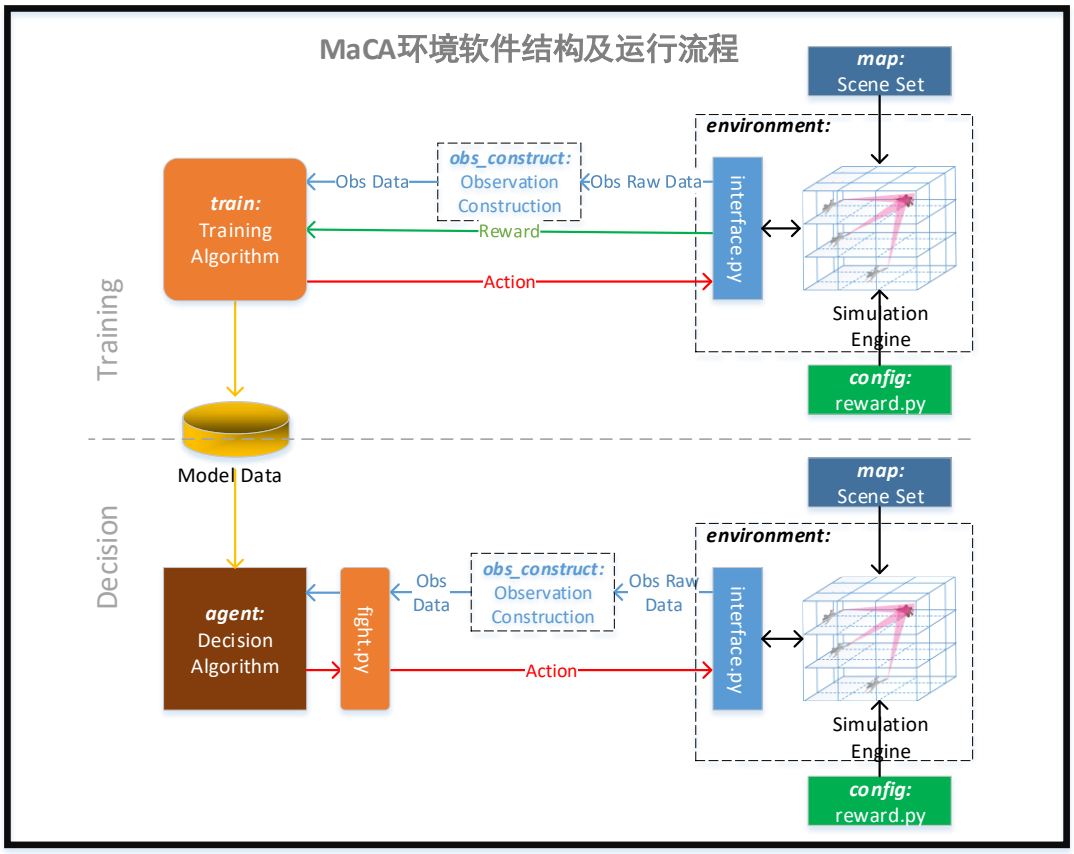


图 1 算法模型与环境交互关系

2. MaCA 环境安装

2.1 系统要求

Linux 64-bit 、Mac OS 或 windows10 x64 操作系统

Python 3.6 版本以上

numpy 1.14.2 以上 pygame 1.9.3 以上

如需深度学习框架，则需使用如下指定的版本：

框架名称	版本
TensorFlow	1.12
Keras	2.2.4
PyTorch	1.0
Mxnet	1.3.1

2.2 环境安装

```
pip install numpy pygame
git clone https://github.com/CETC-TFAI/MaCA.git
```

注：环境安装路径中请勿包含中文

1. 命令行形式配置（适用于 Linux、Mac）：

```
cd MaCA
export PYTHONPATH=$(pwd)/environment:$PYTHONPATH
```

- 在 MaCA 根目录运行所有 py 文件。

2. Pycharm 中配置（适用于 Windows、Linux、Mac）：

- 将“environment/”文件夹设置为“Sources Root”
- 运行任何 py 文件均将其“Working Directory”设置为 MaCA 根目录。

3. MaCA 环境简要使用流程

3.1 对战执行

使用 fight.py 启动对战，通过启动参数实现对战自定义。

3.1.1 fight.py 参数

参数	功能	存在性	赋值规则
--map	地图名称	可选	指定地图名称，不使用则默认“1000_1000_2_10_vs_2_10”。地图存在于“maps/”路径下的.map 文件，此参数赋值需与存在的.map 文件相对应。
--agent1	智能体 1 名称	可选	红方调用决策模型名称，不使用则默认 fix_rule。决策模型代码存在于“agent/决策模型名称”路径，此参数赋值需与存在的决策模型文件夹名称相对应
--agent2	智能体 2 名称	可选	蓝方调用智能体模型名称，不使用则默认 fix_rule。其他同—agent1
--round	对战局数	可选	输入正整数指定对战局数。若不使用此参数，则对战 1 局
--max_step	每局对战最大步长	可选	输入正整数指定每局最大步长。若不使用此参数，则最大步长默认为 5000 步
--random_pos	双方智能体初始位置是否随机	可选	无需赋值，若使用此参数则代表开启出生位置随机
--log	是否记录对战过程	可选	无需赋值，若使用此参数则代表开启日志记录功能用于回放

--log_path	对战日志文件名	可选	若使用—log 参数，则用本参数可设置日志文件名，若不使用本参数默认日志名称为“红方智能体名_vs_蓝方智能体名”。日志存放于“log/日志名称”路径下

3.1.2 对战实例

异构智能体地图：

基于两个固定规则的智能体对战并记录日志到 log/default_log：

```
python fight.py --agent1 fix_rule  --agent2 fix_rule --log_path default_log  --log
```

同构智能体地图：

基于规则与基于 DQN 的智能体对战并记录日志到 log/default_log：

```
python  fight.py –map=1000_1000_fighter10v10 –agent1 fix_rule –agent2 simple --log --log_path default_log
```

3.2 对战回放

读日志 log/default_log 回放对战过程

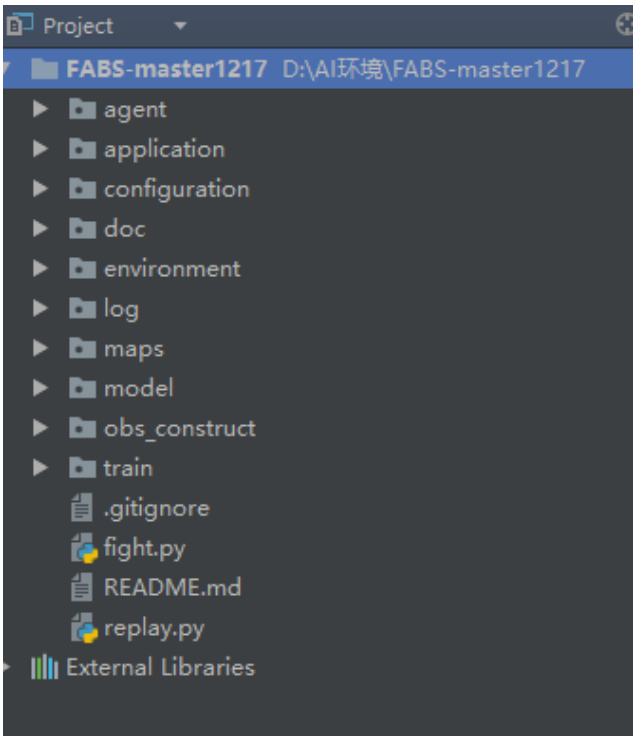
```
python replay.py default_log
```

3.3 自主决策实现

1. 编写 obs_construct 代码并存放于“obs_construct/”下的自命名文件夹中，代码规则参见 4.3
2. 编写训练代码，生成模型存放于“model/”下的自命名文件夹中，环境调用接口参考 4.2，训练流程参考 train/simple/main.py 以及 fight.py
3. 编写 agent 代码存放于“agent/”下的自命名文件夹中，编写规则参考 4.4 及 agent/simple/agent.py
4. 执行训练，生成模型文件
5. 执行 fight 进行对战验证

4. MaCA 环境详细说明

4.1 环境模块介绍



fight.py：发起对战

replay.py：回放对战录像

agent：该模块是封装好的可直接调用的智能体模型，每个参赛者可单独设立一个文件夹，将个人参赛模型按照标准封装成可直接调用的模型。

- **agent\fix_rule：基于规则的同构异构对战决策算法**
- **agent\fix_rule_no_att：基于规则的同构异构对战决策算法（无攻击能力）**
- **agent\simple：基于 DQN 的同构对战决策示例算法,需安装 pytorch**

application：用于 api 测试。

configuration：系统配置：

- **configuration\reward.py**：回报赋值规则定义
- **configuration\system.py**：部分系统仿真规则定义

doc：环境的说明文档。

log：系统日志。

maps\：地图自定义，可在地图中配置对抗单元的数量和种类。

model\：用于存放训练好的模型，例如 XX.pkl。

obs_construct\：观测值组织形式自定义，用户也可以根据算法需要自定义 obs 结构。

- **obs_construct \simple：基于 DQN 的同构对战决策示例对应的 obs 组织**

train\：编写算法进行训练。

- **train\simple：基于 DQN 的同构对战决策示例对应的训练代码**

4.2 环境调用接口说明

MaCA 环境的接口定义在 interface.py 文件的 Environment 类中，其接口函数如下

接口函数	描述
init	环境实例初始化
reset	环境重置
step	环境运行一步
get_obs	获得组合的观测信息
get_done	判断对战是否结束
get_alive_status	获取各作战单元的存活状态
get_map_size	获取地图大小信息
get_unit_num	获取各类别作战单元数目
get_unit_property_list	获取各作战单元属性信息
get_reward	获取回报

4.2.1 init

参数：

序号	参数名	功能	存在性	赋值规则
1	map_path	地图路径	必选	MaCA 根目录下开始的完整地图文件路径，如“maps/1000_1000_fighter10v10.map”
2	side1_obs_ind	红方使用的 obs 构建模块	必选	使用 raw obs 则赋值为“raw”，使用其他自定义 obs 构建则输入对应构建模块所在文件夹名

3	side2_obs_ind	蓝方使用的 obs 构建模块	必选	同 side2_obs_ind
4	max_step	每局最大步长	可选	正整数值，不赋值此参数则使用默认值 5000
5	render	是否开启显示	可选	True：开启显示，False：关闭显示。不使用此参数则为默认不开启
6	render_interval	每隔几步显示一帧	可选	正整数值，不赋值此参数则使用默认每步都显示
7	random_pos	是否使用随机出生位置	可选	True：开启随机出生，False：关闭示。不使用此参数则为默认红方在左侧蓝方在右侧
8	log	日志记录开启指示	可选	False：不开启日志，其他字符串值：日志文件夹名称。不使用此参数则默认不开启
9	random_seed	指定随机种子	可选	-1：不指定随机种子，其他正整数：指定随机种子。不使用此参数则默认不指定，由环境生成。该参数主要用于回放时载入原始种子重现对战场景，fight 和 train 时不需考虑。

返回值：无

4.2.2 reset

参数：无

返回值：无

4.2.3 step

参数：无

序号	参数名	功能	存在性	赋值规则
1	side1_detector_action	红方侦查单元动作集	必选	参见 4.5.2
2	side1_fighter_action	红方干扰打击单元动作集	必选	参见 4.5.2
3	side2_detector_action	蓝方侦查单元动作集	必选	参见 4.5.2
4	side2_fighter_action	蓝方干扰打击单元动作集	必选	参见 4.5.2

返回值：无

4.2.4 get_obs

参数：无

返回值：

序号	返回值名	功能	数据形式
1	side1_obs	红方 obs	raw obs（参见 4.5.1）或自定义 obs 构建格式
2	side2_obs	蓝方 obs	同上

4.2.5 get_done

参数：无

返回值：True、False。为 True 则代表本局结束

4.2.6 get_reward

参数：无

返回值：

序号	返回值名	功能	数据形式
1	side1_detector	红方侦查单元单步回报	参见 4.5.3
2	side1_fighter	红方打击干扰单元单步回报	参见 4.5.3
3	side1_round	红方本局输赢回报	参见 4.5.3
4	side2_detector	蓝方侦查单元单步回报	参见 4.5.3
5	side2_fighter	蓝方打击干扰单元单步回报	参见 4.5.3
6	side2_round	蓝方本局输赢回报	参见 4.5.3

4.3 自组织 Observation 信息统一接口

为使得环境能够返回 Agent 对战或训练时自定义的状态信息，自组织状态信息的接口统一放在 obs_construct 下，如 simple 智能体中指定 obs_ind 为 simple，则 obs_construct 模块下构建 simple 文件夹，每一个信息组织形式文件夹下必须包含 construct.py 文件，且该文件定义 ObsConstruct 类及 obs_construct 成员函数，成员函数接口如下：

```
def obs_construct(self, obs_raw_dict): #obs_raw_dict 为 raw 信息结构.参见 1.5.1
        return obs #obs 为自定义信息结构
```

4.4 Agent 统一接口

为方便统一对战调用形式，限定 Agent 必须包含如下成员、函数接口及固定格式的返回值。

```
from agent.base_agent import BaseAgent
class Agent(BaseAgent):
```

```
def __init__(self):#初始化接口
    BaseAgent.__init__(self)#从 BaseAgent 继承
    self.obs_ind = 'raw'#状态信息形式，’raw’表示原始信息形式，具体参见 1.6.1
    #注：参赛者可基于根据原始信息自行组织新的信息,可参考 simple

def set_map_info(self, size_x, size_y, detector_num, fighter_num):#读取地图信息
    pass#根据需要自行选择函数实现形式

def get_action(self, obs_dict, step_cnt):
#obs_dict 为状态,step_cnt 为当前步数(从 1 开始)
    return detector_action, fighter_action
    #返回动作结构参见 1.5.2
```

4.5 主要数据格式定义

4.5.1 raw obs 信息结构

环境底层原始观测数据 Raw data Observation 结构为字典，包含三部分内容分别为 detector_obs_list、fighter_obs_list、join_obs_dict。

detector_obs_list 表示探测单元信息，结构为 detector 数目的 list。每一个元素为字典结构，表示当前 detector 的 obs 信息，具体内容如下：

Key	Value
id	探测单元 id 编号,从 1 开始
alive	攻击单元存活状态
pos_x	横向坐标（位置坐标体系圆点为左上角，水平向右为 x，垂直向下为 y）
pos_y	纵向坐标（位置坐标体系圆点为左上角，水平向右为 x，垂直向下为 y）
course	航向(0-359)，水平向右为 0 度，顺时针旋转
r_iswork	雷达开关状态
r_fre_point	雷达频点
r_visible_list	雷达观测到的敌方单位列表，每一元素字典，表示敌方单位信息，结构为 {‘id’:编号,’type’:类型(0:探测单元，1：攻击单元) ,‘pos_x’:横向坐标,’pos_y’:纵向坐标}
last_reward	上一个状态采用动作的回报
last_action	上一个状态采用的动作(字典结构) {‘course’:,’r_iswork’:,’r_fre_point’: }

fighter_obs_list 表示攻击单元信息，结构为 detector 数目的 list。每一个元素为字典结构，表示当前 detector 的 obs 信息，具体内容如下：

Key	Value
id	攻击单元 id 编号,编号从探测单元数目开始
alive	攻击单元存活状态
pos_x	横向坐标（位置坐标体系圆点为左上角，水平向右为 x，垂直向下为 y）
pos_y	纵向坐标（位置坐标体系圆点为左上角，水平向右为 x，垂直向下为 y）
course	航向(0-359)，水平向右为 0 度，顺时针旋转
r_iswork	雷达开关状态
r_fre_point	雷达频点
r_visible_list	雷达观测到的敌方单位列表，每一元素字典，表示敌方单位信息，结构为{‘id’:编号,’type’:类型(0:探测单元，1：攻击单元），‘pos_x’:横向坐标,’pos_y’:纵向坐标}
j_iswork	干扰雷达开关
j_fre_point	干扰雷达频点
j_recv_list	雷达被动观测的敌方单位列表，每一元素字典，表示敌方单位信息，结构为 {‘id’:编号,’type’: 类型(0:探测单元，1：攻击单元），’direction’:目标相对自身的方向,’r_fp’:频点}
l_missle_left	远程导弹剩余数目
s_missle_left	短程导弹剩余数目
striking_list	打击列表[{'attacker_id': 攻击者编号, 'target_id': 目标编号, 'missile_type': 导弹类型(1：远程，2：中程), information_source_type: 信息来源类型（0：L 探测单元，1:S 探测单元，2：战机，3：被动侦测） step_count: step 计数, op_count: 观测点计数, del_ind: 删除指示(默认 false)}]
striking_dict_list	攻击列表[{‘target_id’:目标编号，‘pos_x’:横向坐标,’pos_y’:纵向坐标，‘type’: 类型(0:探测单元，1：攻击单元) }]
last_reward	上一个状态采用动作的回报
last_action	上一个状态采用的动作(字典结构) {‘course’:,’r_iswork’:,’r_fre_point’:,’j_iswork’:,’j_fre_point’;,’hit_target’:,’missle_type’:}

joint_obs_list 为表示全局信息的字典{‘strike_list’: 打击列表，’passive_detection_enemy_list’:被动侦测列表}。strike_list 为每一个作战单元 striking_list 的并集，passive_detection_enemy_list：[{‘id’:敌方编号,’type’: 类型(0:探测单元，1：攻击单元)，‘pos_x’:横向坐标,’pos_y’:纵向坐标}]

4.5.2 动作 Action 结构

以 2 个探测单元，10 个攻击单元为例，则：

detector_action 为 2*2 的数组，每一行表示每一个探测单元的两个动作

	意义	取值范围
动作 1	航向	[0-359]离散值

动作 2	雷达开关及频点	频点[0,频点总数],0 表示雷达关，非 0 表示开机对应频点

fighter_action 为 10*4 的数组，每一行表示每一个攻击单元的四个动作：

	意义	取值范围
动作 1	航向	[0-359]离散值
动作 2	雷达开关及频点	频点[0,频点总数],0 表示关，非 0 表示开机对应频点
动作 3	干扰设备开关机频点	频点[0,频点总数+1],0 表示关, [1,频点总数]表示开机对应频点 频点总数+1：阻塞干扰
动作 4	是否发射导弹及攻击目标的 id	[0,25] 0：表示不发射导弹 1-12: 表示远程导弹攻击敌方目标 id 13-24: 表示中程导弹攻击敌方目标 id+12

4.5.3 回报 Reward 结构

回报信息可以通过 Environment 接口中的 get_reward 获得，具体结构如下：

#回报格式说明，以 2 个探测单元，10 个攻击单元为例，则

- detector_reward 为(2,)的 numpy 数组，每一个元素表示每一个探测单元的动作回报。
- fighter_reward 为(10,)的 numpy 数组，每一个元素表示每一个攻击单元的动作回报。
- round_reward 为游戏输赢的回报，此回报只有当游戏结束才有效。

4.6 环境配置

4.6.1 回报值自定义

回报值配置定义在“configuration/reward.py”中，提供若干回报生成检查点回报值设定，用户可根据自己算法训练实际要求自行修改回报值。若目前提供回报检查点不满足算法训练需求，可通过获取 obs 自行实现

个体回报定义

序号	单元类型	分类	意义	变量名
1	攻击单元	攻击合理性回报	攻击动作合法	reward_strike_act_valid
2			攻击动作非法	reward_strike_act_invalid

3		探测回报	探测到探测单元	reward_radar_fighter_detector
4			探测到攻击单元	reward_radar_fighter_fighter
5		攻击结果回报	打击探测单元成功	reward_strike_detector_success
6			打击探测单元失败	reward_strike_detector_fail
7			打击攻击单元成功	reward_strike_fighter_success
8			打击攻击单元失败	reward_strike_fighter_fail
9		被击毁回报	攻击单元被击毁	reward_fighter_destroyed
10	探测单元	被击毁回报	探测单元被击毁	reward_detector_destroyed
11		探测回报	探测到探测单元	reward_radar_detector_detector
12			探测到攻击单元	reward_radar_detector_fighter
13	通用	存活回报	每步存活	reward_keep_alive_step

全局回报定义

序号	**意义**	**变量名**
1	完胜	reward_totally_win
2	完败	reward_totally_lose
3	胜利	reward_win
4	失败	reward_lose
5	平局	reward_draw

4.6.1 系统配置

系统配置功能在“configuration/system.py”中，提供一些仿真规则自定义功能。具体配置信息如下：

- attack_effect_delay：攻击是实时生效还是具有导弹飞行延迟，True 为具有延迟，False 为实时生效
- hit_prob_enable：攻击效果是否考虑击中概率，True 为使用击中概率计算，False 代表每次都击中。

注：本次竞赛使用默认值开展，此处配置在比赛中不可修改，但用户可尝试修改用于简化训练。

4.7 胜负判定规则

当一轮对战符合结束规则，则本轮结束并进行胜负判定。当对战结束时get_done 函数将返回 True，胜负结果则由 get_reward 返回值中的 round_reward 变量表征。

完胜：一方被全部击毁

胜利：

- 双方导弹存量为 0，则存活作战单元数量多的获胜
- 当达到最大 step，则剩余作战单元数量多的获胜

平局：

- 双方作战单元全都被击毁
- 当双方导弹存量为 0 且双方存活作战单元数量相等
- 当达到最大 step 且双方存活作战单元数量相等

4.8 地图信息

4.8.1 同构智能体地图

同构地图对战双方各拥有 10 个相同的类型属性的攻击单元，攻击单元具备侦察、探测、干扰和打击等功能，具体信息如下：

地图：

地图名称	1000_1000_fighter10v10.map
地图尺寸	1000*1000

波段信息：

波段	标识
L	0
S	1
X	2

攻击单元类型属性：

速度	雷达波段	频点数目	干扰波段	干扰频点数目	近程导弹数量	远程导弹数量	类型数量
2	2	10	2	10	4	2	10

4.8.2 异构智能体地图

同构地图对战双方各拥有 12 个不同类型属性的攻击单元和探测单元，探测单元具备侦察和探测功能，攻击单元具备侦察、探测、干扰和打击等功能，具体信息如下：

地图：

地图名称	1000_1000_2_10_vs_2_10.map
地图尺寸	1000*1000

波段信息：

波段	标识
L	0
S	1
X	2

探测单元类型 1 属性：

速度	雷达波段	频点数目	类型数量
1	0	20	1

探测单元类型 2 属性：

速度	雷达波段	频点数目	类型数量
1	1	20	1

攻击单元类型 1 属性：

速度	雷达波段	频点数目	干扰波段	干扰频点数目	近程导弹数量	远程导弹数量	类型数量
3	2	10	2	10	4	2	8

攻击单元类型 2 属性：

速度	雷达波段	频点数目	干扰波段	干扰频点数目	近程导弹数量	远程导弹数量	类型数量
3	2	10	0	10	4	2	1

攻击单元类型 3 属性：

速度	雷达波段	频点数目	干扰波段	干扰频点数目	近程导弹数量	远程导弹数量	类型数量
3	2	10	1	20	4	2	1

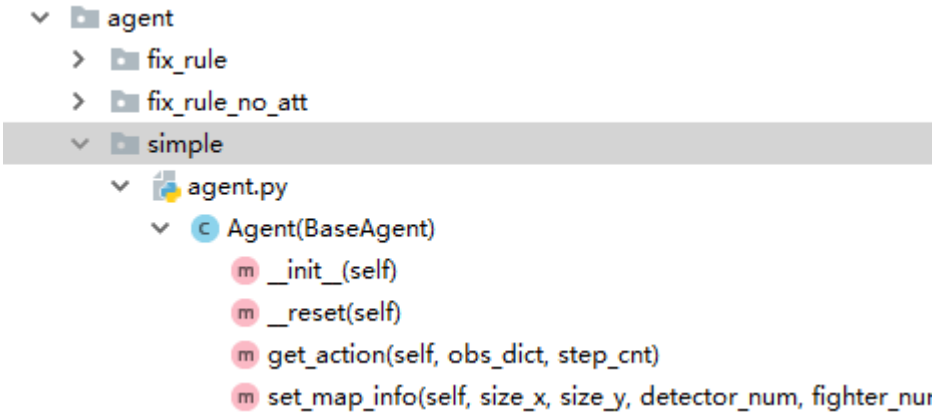
5. 参赛作品提交要求

参赛者需要提供如下文件：

1. Agent 算法设计说明文档包括算法整体思路、状态信息组织方式、智能训练算法介绍等，若 Agent 为基于规则的模式则需详细介绍规则模型。
2. agent 调用、训练、模型参数、观测信息组织四个模块文件夹

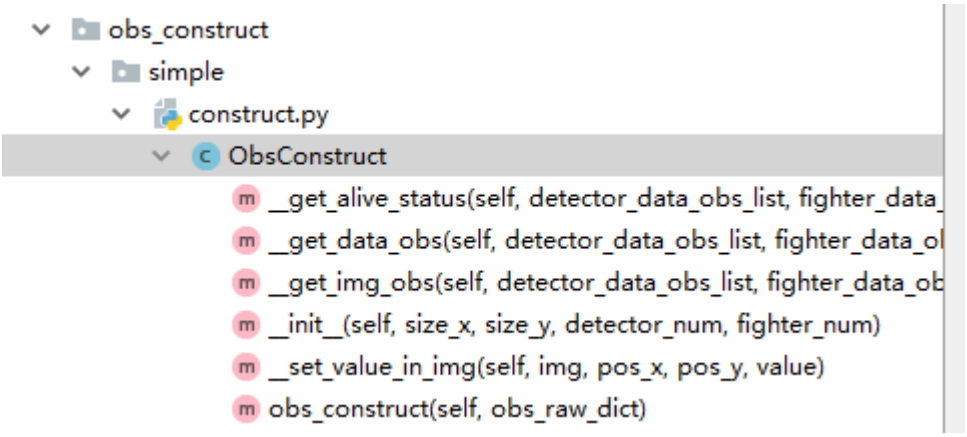
（1）agent 文件夹

agent 文件夹包含以参赛 agent 的名字命名的子文件夹，子文件夹中必须包括 agent.py 且 agent.py 中必须包含 1.6 中定义的接口,保证该模块能够被 fight.py 调用。以名字为”simple”的 agent 为例，如下图所示



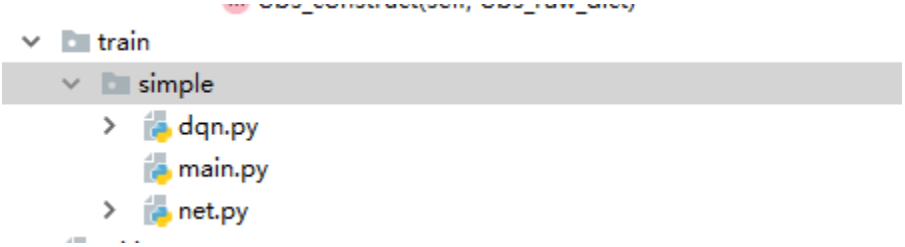
（2）obs_construct 文件夹(可选)

若参赛者选择自己组织状态信息，则需要提供 obs_construct 文件夹，obs_construct 文件夹中包含以参赛 agent 的名字命名的子文件夹，子文件夹中存放 construct.py 且 construct.py 中必须包含 1.5 中定义的接口。以 agent 名字为”simple”为例，如下图所示



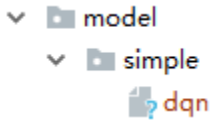
（3）train 文件夹(可选)

若参赛者算法为基于训练得到的模型，则需要提供参数训练源代码，即提供 train 文件夹, train 文件夹包含以参赛 agent 的名字命名的子文件夹，子文件夹中存放模型参数训练源代码，以名字为”simple”的 agent 为例，如下图所示



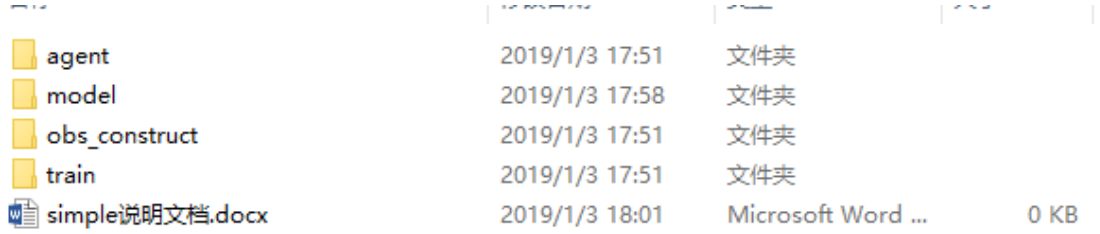
（4）model 文件夹(可选)

若参赛者算法为基于训练得到的模型，则需要提供训练好的模型参数文件，model 文件夹包含以参赛 agent 的名字命名的子文件夹，子文件夹中存放模型参数，以名字为”simple”的 agent 为例，如下图所示



注：

参赛者最终需要提交四个文件夹和一个说明文档即



并其打包成 rar 压缩文件，并以“子竞赛名称+参赛智能体名称+参赛者昵称“ 命名。

子竞赛名称可选同构或异构

参赛者用户名为共享社区注册时填写的昵称

以昵称为 TQA 的用户，参加同构智能体竞赛，智能体名字为 simple 为例，压缩文件名称为

同构+simple+TQA.rar