

代码原始出处为openai的github项目：[Multi-Agent Particle Environment](#)

在阅读此文档之前，请先阅读README.md文档，里面有对各个代码文件的系统性介绍

课程所用的环境为simple\_tag.py子环境，并进行了一定程度的修改，主要改动为是在openai的原始环境基础上，给simple\_tag环境添加了边界墙，3个固定障碍物，5个打卡点。

# 1.环境配置

## 1.1 环境所需要的依赖

根据实操后，需要以下依赖：**python3.6**及以上（在我电脑上安装的是python3.6，低于python3.6可能会报错），**gym 0.10.5**，**numpy**，**scipy**。

推荐使用anaconda创建虚拟环境，在虚拟环境中配置。

框架：**tensorflow** 或者**pytorch**。根据需要安装，因为环境文件并没有用到这些框架。

安装均可在终端用**pip**的形式安装：

```
pip install gym==0.10.5
```

```
pip install numpy
```

```
pip install scipy
```

推荐使用**ubuntu**系统，windows我没试过，不保证能成功运行环境。

## 1.2 安装环境

OpenAI将该环境封装成了库，可以直接通过**import**形式输出环境。在此之前我们需要安装该环境。

首先切换到环境**multiagent-envs-ML**主目录下（你会看到有一个 **setup.py** 文件），在此目录下打开终端，运行以下代码安装：

```
pip install -e .
```

注意 **.** 号前面有空格。

安装完成后最终会提示：

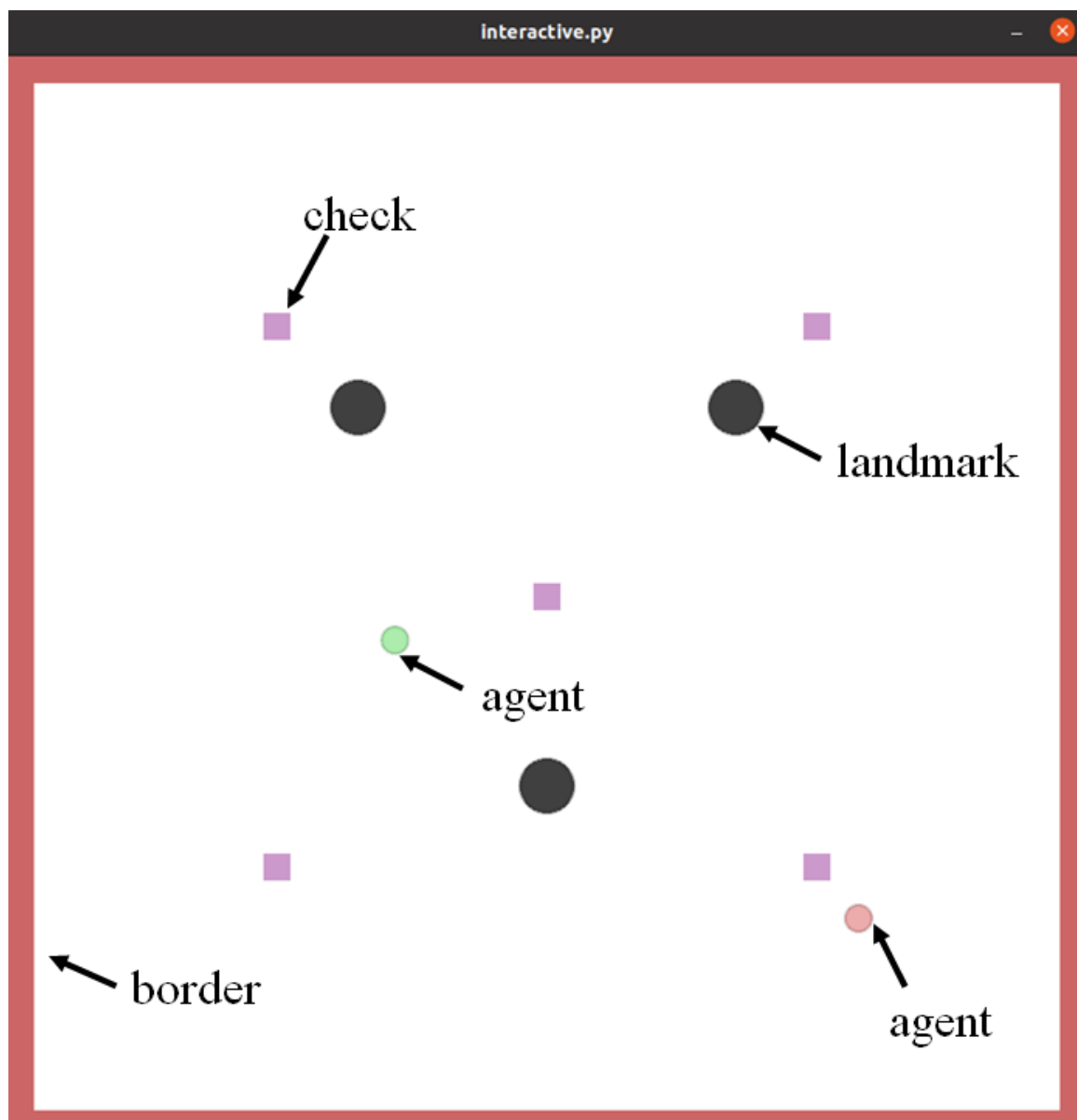
```
Running setup.py develop for multiagent
Successfully installed multiagent-0.0.1
```

## 2.代码运行

安装成功后，正常情况下代码运行应该不会报错。运行步骤如下：

cd到 **/multiagent-envs-ML/bin** 文件夹下，直接在终端运行

即可。你会看到有两个一样的环境窗口，这是因为有两个智能体，你可以通过用键盘的上下左右键分别控制各自窗口的一个智能体。**interactive.py**可以把它当作主函数。



### 3.环境代码部分介绍

1. 该部分可以参考这篇博客的内容，是对openai环境及其算法的介绍：[深度解析OPENAI-MADDPG](#)
2. 修改的环境部分主要是增加了边界类Border，打卡点类Check。
3. 最重要的环境文件是 `core.py`，`simple_tag.py`，`environment.py` 这三个文件。这三个文件的大致调用关系是：`core`被`simple_tag`调用，`simple_tag`被`environment`调用。
4. `core`文件中主要声明了环境中出现的各个实体：`agent`，`border`，`landmark`，`check`等。
5. `simple_tag`文件主要设置实体的参数（初始位置、加速度等），`reward`设置等。
6. `environment`文件是强化学习算法中的经典环境接口（`step`,`reset`，`render`等）。你可以按照给定任务重写 `environment.py` 文件中的 `self._set_action` 函数。

## 4. 一些问题说明

- 控制量action均为离散量，范围为 `[0,1]`，是个  $2 * 5$  维的向量，对应两个智能体。每个智能体都是由  $1 * 5$  维的action控制。其中，第1维并没有用到，用到的是后面四维。（官方给的代码默认设置是这样的）
- `policy.py` 文件是使用键盘控制写的文件，只是为了演示(`interactive.py`文件中使用)。
- 整个画面的实际尺度为 `[-1,1] * [-1,1]`，屏幕正中心为原点。画面宽高为800\*800像素。
- 在 `/multiagent-envs-ML/bin` 文件夹下的 `interactive.py` 文件，有初步获取图像信息的方式，至于如何利用这个信息来提取目标信息，还得同学们自己实现。

```
50         # step environment
51         obs_n, reward_n, done_n, _ = env.step(act_n, target)
52
53         # # render all agent views
54         image = env.render("rgb_array") # 使用这种方法读取图片信息
55
56         # print(image)
57         # print(np.shape(image)) # [2,height, width, channels (3=RGB)]
```

上一届的同学已经通过这种方式获取图像信息，再通过图像处理的方式(CenterNet算法)获得所有智能体、目标点、障碍物的位置信息。这也是老师所提出的第一个任务要求：通过图像处理获取各个物体的位置信息。

另外，在 `simple_tag.py` 文件中有直接获取位置等信息的代码：

```
environment.py  simple_tag.py x
multiagent-envs-ML > multiagent > scenarios > simple_tag.py > Scenario > make_world
216         return rew
217
218     def observation(self, agent, world):
219         # get positions of all entities in this agent's reference frame
220         entity_pos = []
221         for entity in world.landmarks:
222             if not entity.boundary:
223                 entity_pos.append(entity.state.p_pos - agent.state.p_pos)
224         # communication of all other agents
225         comm = []
226         other_pos = []
227         other_vel = []
228         check_pos = []
229         check_pos.append(agent.state.p_pos - world.check[0].state.p_pos)
230         for other in world.agents:
231             if other is agent:
232                 continue
233             comm.append(other.state.c)
234             other_pos.append(other.state.p_pos - agent.state.p_pos)
235             # if not other.adversary:
236             other_vel.append(other.state.p_vel)
237             other_vel.append(other.state.p_vel) # 增加
238             dists = np.sqrt(np.sum(np.square(agent.state.p_pos - other_pos)))
239         # return np.concatenate([agent.state.p_pos] + [agent.state.p_vel] + dists) # 知道自己的位置、速度和
240         return np.concatenate([agent.state.p_pos] + other_pos + check_pos + entity_pos + [agent.state.p_
```

但是这种方式是不符合老师所提出的要求的，在这里提个醒（老师是想让同学们用图像处理等知识获取状态信息）。