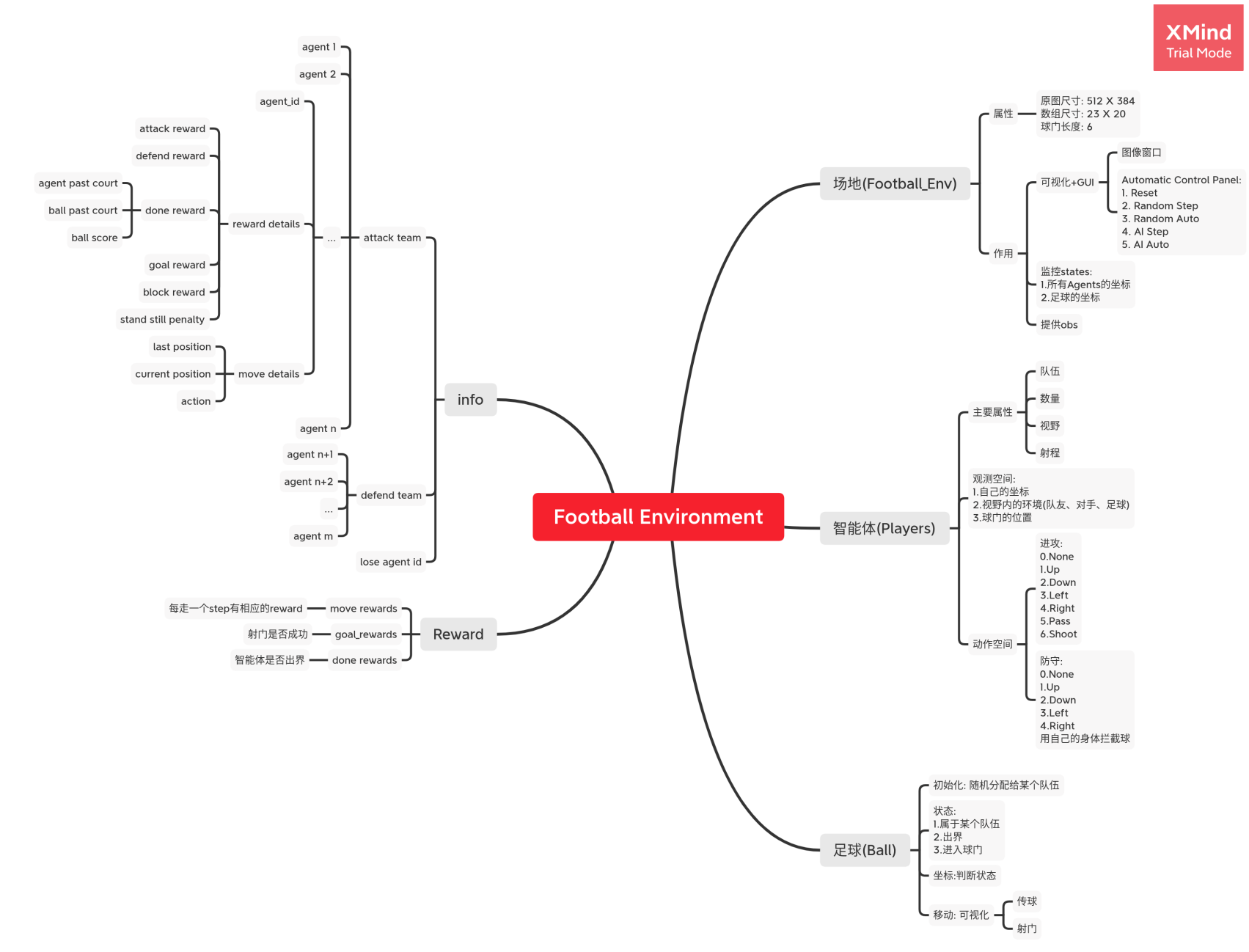
多智能体足球游戏环境

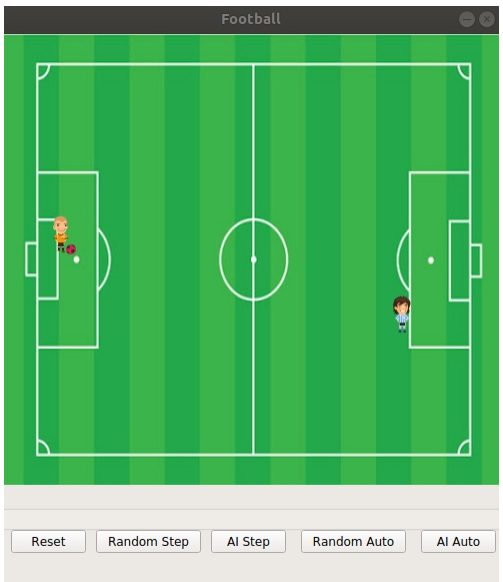
一、简介

根据实验室的需要并参考现有的一些开源多智能体强化学习环境，我自己写了这样一个基于网格世界的多智能体足球游戏。该环境简单易懂、容易操作，所有代码都是用python完成的并且提供了一个UI界面可以实现一些简单的交互功能。

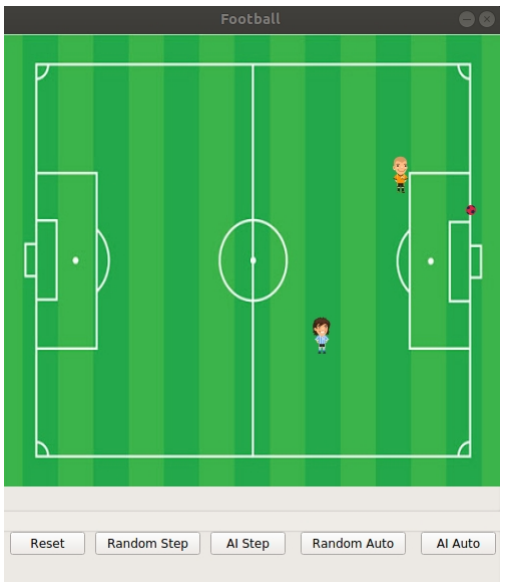


二、场景描述

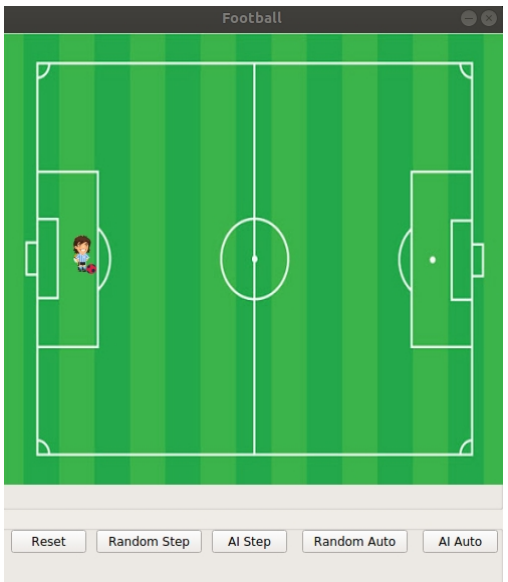
1、基本规则



图一



图二



图三

以现实的足球场景为基础，该环境是全场游戏分为黄、蓝两支队伍，一方进攻另一方防守如图所示。与真实足球场景不同的是该环境做了许多简化：

（1）没有开球、点球、边线发球、加时赛等环节；不考虑禁区；当使用多智能体环境时智能体传球不考虑越位；没有裁判智能体

（2）智能体带球跑动时球也跟随移动不会出现丢球等现象

（3）智能体射门需要满足特定的条件：射程足够、角度合适

（4）该场景进攻方的目标是射门得分，防守方的目标是阻止进攻方射门。

图一为一对一场景的初始化；图二为进攻方得分；图三为防守方和进攻方发生了碰撞，试做防守方完成了抢断。

2、UI界面

UI界面包括五个功能：

Reset：初始化环境

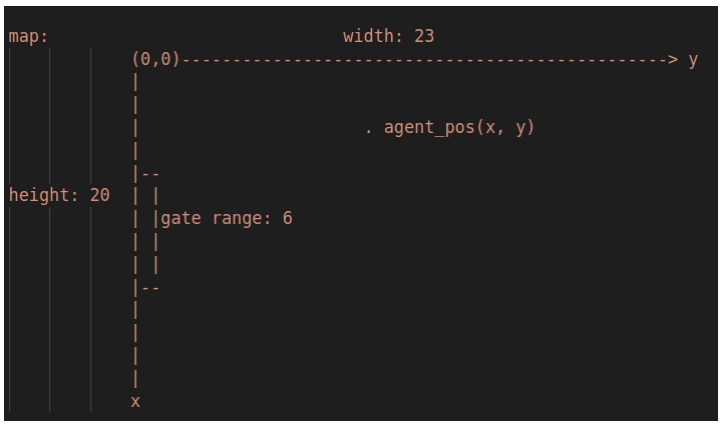
Random Step：每个智能体都采取随机策略、单步执行

AI Step：已经训练过的智能体根据网络输出动作，其他智能体使用随机策略、单步执行

Random Auto：每个智能体都采取随机策略并且连续执行

AI Auto：已经训练过的智能体根据网络输出动作，其他智能体使用随机策略并且连续执行

三、坐标系



由于本环境是基于网格世界的，所以整个足球场使用一个二维numpy array来存储，为了方便索引重新定义了坐标系方向，如上图所示。

四、初始化

```
 # initialize map
self.reset_map()

# initializer team
self.reset_agents_team()

# intialize players postion
for agent in self.agents.values():
    self.reset_agent_position(agent)

    # give ball to the attack team
    self.reset_ball_possesion()

    # update map
    self.update_map()
```

初始化环境主要有以下几个流程：

1、初始化整个地图，将智能体id全部清零

2、初始化智能体的进攻方与防守方

3、分配球权

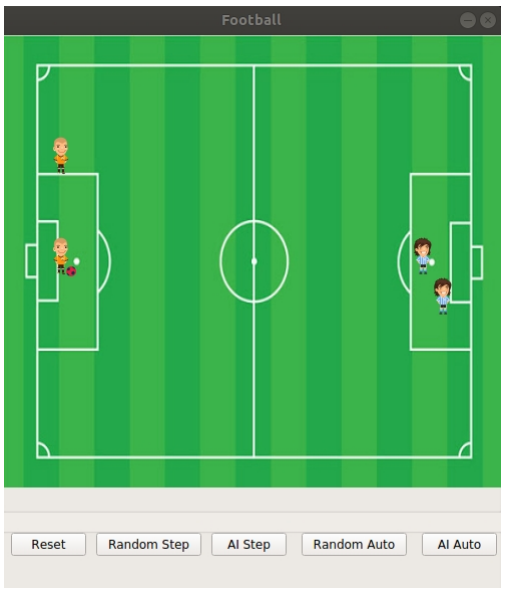
4、初始化智能体的坐标

5、更新整个地图

6、返回整个observation

智能体的个数可以自由添加，最大步数、移动收益的权重、场地的参数都可以才创建环境实例的时候调整。

```
env = Football_Env(agents_left=[1, 2], agents_right=[3, 4],
    max_episode_steps=500, move_reward_weight=1.0,
    court_width=23, court_height=20, gate_width=6)
env.reset()
```



五、智能体

虽然该环境的智能体分为进攻方与防守方，但是基本上大同小异。

进攻方的动作空间为：

0：不动

1：上

2：下

3：左

4：右

以上五个动作不会导致球权的变化

5：传球

智能体根据自己的视野判断是否有能够传球的队友，如果有并且选择了传球这个动作则直接将球传过去，传球轨迹为一条线段。

6：射门

智能体根据自己的射程和射门角度以及球门位置判断是否满足射门条件，如果满足并且自己拥有球权则直接射门，射门轨迹也是直线。在本环境中向球门中心射门的成功率最高，球门中心两侧方向成功率依次对称递减。

防守方的动作空间为：

0：不动

1：上

2：下

3：左

4：右

防守方有一定的防守范围，如果球的运动轨迹经过了防守方的防守范围则视为防守成功。

每个智能体的观测空间为自己的绝对坐标、自己视野范围内是否有其他智能体、球门的位置。

每当进攻方射门成功后，进攻方得到一个正的reward，防守方相应得到负的reward；当进攻方的射门被防住了或者被防守方抢断了则防守方得到正的reward，进攻方得到负的reward。

六、重要类

Players

重要属性：

```
self.id = agent_id # 编号从1开始，地图上智能体所在位置值为对应的编号，没有智能体的位置值为0
self.court_id = court_id # 左半场或右半场
self.team = team # 进攻方或防守方
self.court_width = court_width
self.court_height = court_height
self.gate_width = gate_width
self._map = _map
self.gate_pos = self.get_gate_pos() # 球门坐标，是一个范围
self.success_rate = self.get_success_rate() # 球门范围内不同坐标对应的射门成功率
self.posses_ball = False # 是否有球权，有球权的智能体对应的坐标就是球的坐标
```

重要方法：

```
_get_obs(_map) # 返回智能体的obs

sample_action(_map, agents) # 从动作空间里选择一个动作

see_around(_map) # 观察自己九宫格或更大范围内的地图

get_gate_pos() # 获取球门的坐标范围

get_success_rate() # 计算不同球门位置的射门成功率

can_shoot() # 判断是否能够射门

can_pass(_map, agents) # 判断是否能够传球

simulate_move(action, _map, ball, agents) # 模拟一下当前动作

after_step(action, _map, ball, agents) # 返回执行step之后的一系列变化
```

Ball

重要属性：

```
self.blocked = False # 判断球在运动过程中是否被拦截
```

重要方法：

```
give_ball_possession(pos) # 分配球权，确定球的坐标

move(source, destination, _map) # 执行球的运动

check_ball_score(team, court_id, court_width, court_height, gate_width=6) # 根据球的坐标判断是否得分
```

Football_Env

重要属性：

```
self.agents_left = agents_left # 左半场智能体的id列表
self.agents_right = agents_right # 右半场智能体的id列表
self.max_episode_steps = max_episode_steps # 最大步数
self.move_reward_weight = move_reward_weight
self.elapsed_steps = 0 # 已经走过的步数
self.GOAL_REWARD = 100.0 # 进攻方射门成功的reward
self.tackle_reward = 0.0 # 防守抢断的reward
self.score_defend_penalty = 0.0 # 进攻方射门成功时防守方的reward
self.defender_block_reward = 0.0 # 防守拦阻拦传球或射门的reward
self.tackle_winner = None # 抢断成功的智能体
self.agents_conflict = False # 是否抢断成功
self.previous_map = None # 更新之前的地图
self._map = None # 当前的地图
self.n_agents = len(agents_left) + len(agents_right) # 智能体总数
self.agents = {} # 智能体字典
self.ball = Ball()
self.Done = False # 一轮游戏是否结束
self.attack_action_space_n = 7
self.defend_action_space_n = 5
self.blocked = False
self.winner = [] # 一轮游戏的胜利者
self.lose_attack_agent_id = None # 导致进攻方失败的智能体id
```

重要方法：

```
reset() # 初始化整个环境

reset_map()

reset_agent_position(agent)

reset_agents_team()

reset_ball_possesion()

update_map()

get_obs()

sample_actions()

```

```
agents_past_court(id) # 判断智能体是否出界

step(actions) # 返回 next_state, rward, done, info

get_goal_rewards(agent, action)

get_done_rewards(agent, action)

get_move_rewards(agent, action)

get_all_pos()
```

七、算法

首先写了一个随机策略来检查环境的公平性，测试进攻方和防守方的胜率。

我首先采用了最简单的Q-Learning方法试了一下这个环境，由于观测空间是用的绝对坐标而不是相对方位，所以状态空间比较大，Q-Learning、SARSA这样的表格型方法并不适用。

又采用了DQN算法，效果还不错。再加上环境本身比较简单，训练速度也很快。

八、一些问题

1、其实多智能体的环境状态最好还是设计成方位和距离，因为绝对坐标这样的信息是一个额外定义的信息，不利于智能体学到一个普适的规则。

2、在设计传球这个动作的时候，设计成每两个智能体之间的传球都单独给一个动作可能会更好，这样能保证每个动作是平等的，否则可能训练不出有效的传球策略。尽管这样设计会导致动作空间随着智能体数量增加急剧增加。

3、reward设置是一个很微妙的东西，需要好好调试。