Project Overview

This project is reconstruction from the basic mpe

Code story

This project uses simple_ tag as a base environment to simulate a simple fighter avoidance missile project. It can be said to be an advanced avoidance ball simulation, but the "ball" has added certain tracking and collaborative raid functions.

In this project, there is one good agent and two adversaries. As time goes by, the size of missiles will become larger and larger.

Code Modifications and Explanations

In `core.py` function

We added the `border` method:

```
# properties of border entities
class Border(Entity):
    def __init__(self):
        super(Border, self).__init__()


    def entities(self):
        return self.agents + self.landmarks + self.borders

    def border(self):
        return self.borders
```

Modifications in `environment.py` file

Drawing the boundaries:

```
pythonCopy codeif 'border' in entity.name:
    geom = rendering.make_polygon(entity.shape)  # outlining the border, which
shape is square
    geom.set_color(*entity.color)
```

Changes in `simple_tag.py` file

Modifying the number of `good_agent` and `adversary`, as well as their acceleration and size:

```
pythonCopy codedef make_world(self):
    world = World()
    # set any world properties first
    world.dim_c = 2
    num_good_agents = 1
    num_adversaries = 2
    num_agents = num_adversaries + num_good_agents
    num_landmarks = 0
    num_borders = 80
    # add agents
    world.agents = [Agent() for i in range(num_agents)]
    for i, agent in enumerate(world.agents):
        agent.name = 'agent %d' % i
        agent.collide = True
        agent.silent = True
        agent.adversary = True if i < num_adversaries else False
        agent.size = 0.05 if agent.adversary else 0.05
        agent.accel = 1.0 if agent.adversary else 1.5
        agent.max_speed = 1.2 if agent.adversary else 1.6
```

Modified the parameters of rendeng() to increase the size of the missile over time during the display process

```
def render(self, step = 1, mode='human'):
    ***



    # create rendering geometry
    #if self.render_geoms is None:
    if True:
        #counter = 0 #an counter to make sure the step == eps_len
        # import rendering only if we need it (and don't import for headless 
machines)
        #from gym.envs.classic_control import rendering
        from multiagent import rendering
        self.render_geoms = []
        self.render_geoms_xform = []
        for entity in self.world.entities:
            geom = rendering.make_circle(entity.size)
            xform = rendering.Transform()
            if 'agent' in entity.name:
                if '2' not in entity.name:
                    #counter += 1
                    #print('size Incresaed ' + entity.name + ' ' + str(counter))
                    entity.size += 0.001
                geom = rendering.make_circle(entity.size)
                geom.set_color(*entity.color, alpha=0.5)

            ***



    return results
```

Reset size after each round

```
def reset_size(self):
    # 可以在这里改变大小
    size = 0
    for agent in self.agents:
        if not agent.adversary:
            size = agent.size
    for agent in self.agents:
        agent.size = size if agent.adversary else agent.size
```

Adding new `adversary_reward` and `adversary_observation` functions:

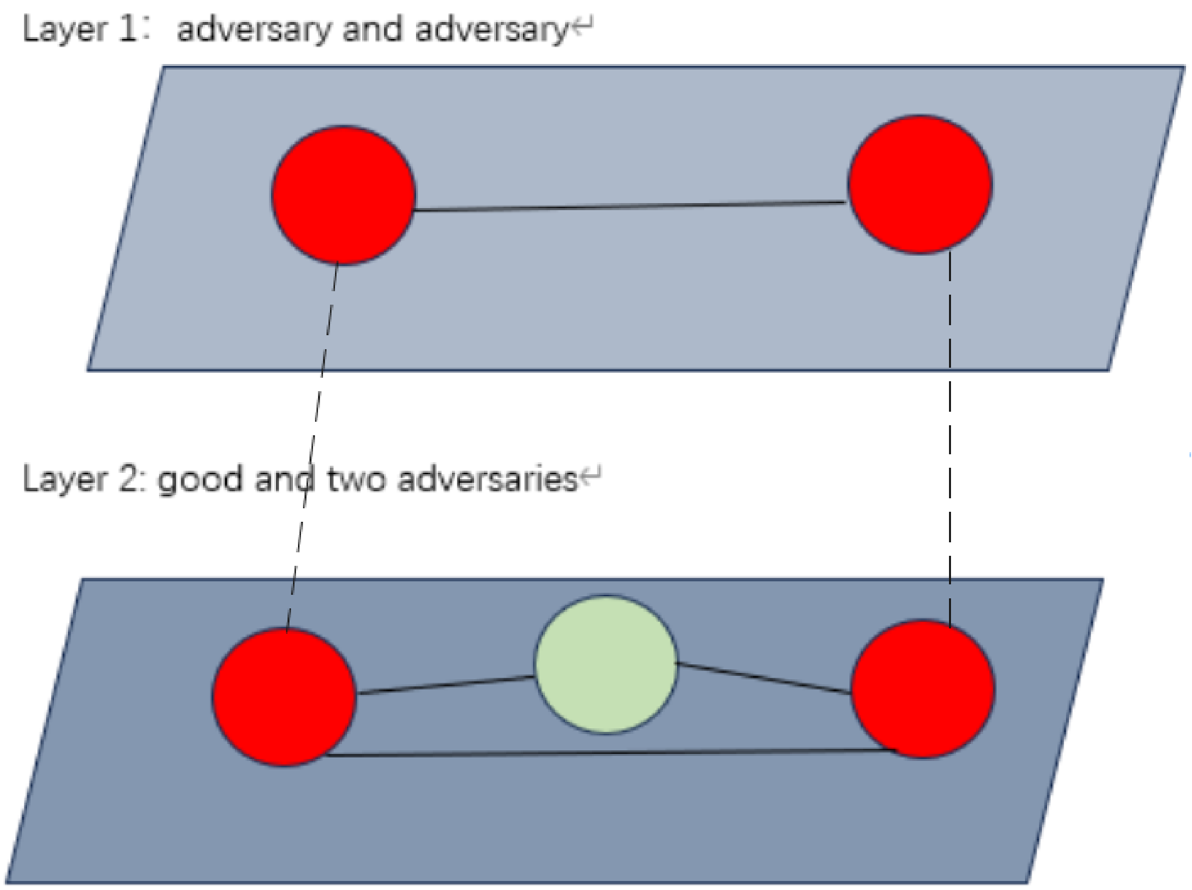
```
pythonCopy codedef adversary_reward(self, agent, world):
    # Adversaries are rewarded for collisions with agents
    rew = 0
    shape = True
    agents = self.good_agents(world)
    adversaries = self.adversaries(world)
    if shape:  # reward can optionally be shaped (decreased reward for increased
distance from agents)
        for adv in adversaries:
            rew -= 0.1 * min([np.sqrt(np.sum(np.square(a.state.p_pos -
adv.state.p_pos))) for a in agents])
    if agent.collide:
        for ag in agents:
            for adv in adversaries:
                if self.is_collision(ag, adv):
                    rew += 10
    return rew

def adversary_observation(self, agent, world):
    #print('++++++++++++++adversary_observation active++++++++++++++')  # Make
sure it's being called
    # ctl_pos = []
    # for ctl in self.controllers(world):
    # ctl_pos.append(ctl.state.p_pos - agent.state.p_pos)
    other_pos = []
    other_vel = []

    for other in world.agents:
        if other is agent:
            continue
        other_pos.append(other.state.p_pos - agent.state.p_pos)
        if not other.adversary:
            other_vel.append(other.state.p_vel)

    """if '2' not in agent.name:
        agent.size *= 1.01
        print(agent.name + ' ' + str(agent.size))"""
    return np.concatenate([agent.state.p_vel] + [agent.state.p_pos] + other_pos
+ other_vel)
```

note:

This report only shows the code used, and some debugging is not shown in above

Multi-layer structure display

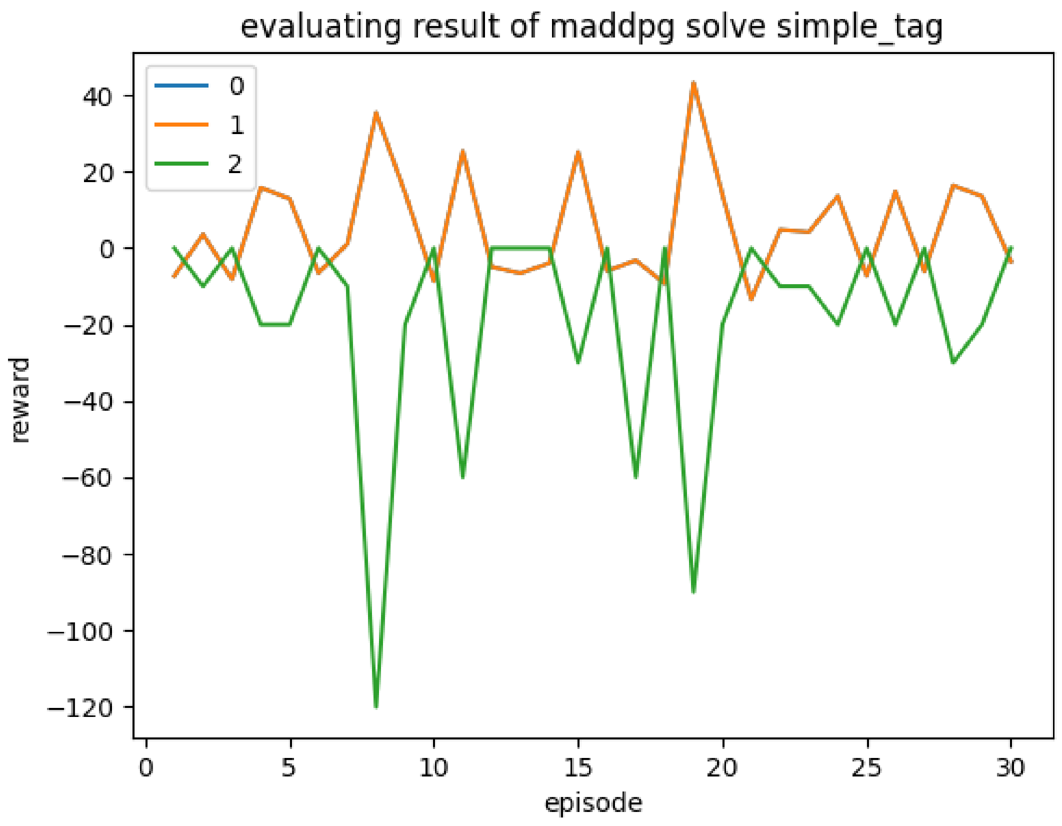
In the first layer here, two adversaries can be accessed through scenario.adversary_ observation communicates with each other, and in the second layer, good communicates with two other adversaries through scenario. observation


Layer 1: adversary and adversary
Layer 2: good and two adversaries

Training result of maddpg

Evaluate Result

Analysis: It can be seen that there is a certain fluctuation in the reward, which is caused by the random refresh point of the agent, resulting in some unpredictable deductions. However, observing the process of model display, the overall results are in line with expectations.


evaluating result of maddpg solve simple_tag
0
1
2
reward
episode
40
20
0
−20
−40
−60
−80
−100
−120
0
5
10
15
20
25
30