gobigger

发行版本 *0.1.0*

OpenDILab Contributors

2022 年 10 月 08 日

教程

CHAPTER 1

总览

多智能体对抗作为决策 AI 中重要的部分，也是强化学习领域的难题之一。为丰富多智能体对抗环境，OpenDILab 开源了一款多智能体对抗竞技游戏环境——Go-Bigger。同时，Go-Bigger 还可作为强化学习环境协助多智能体决策 AI 研究。与风靡全球的 Agar 等游戏类似，在 Go-Bigger 中，玩家（AI）控制地图中的一个或多个圆形球，通过吃食物球和其他比玩家球小的单位来尽可能获得更多重量，并需避免被更大的球吃掉。每个玩家开始仅有一个球，当球达到足够大时，玩家可使其分裂、吐孢子或融合，和同伴完美配合来输出博弈策略，并通过 AI 技术来操控智能体由小到大地进化，凭借对团队中多智能体的策略控制来吃掉尽可能多的敌人，从而让己方变得更强大并获得最终胜利。

GoBigger 提供了多种接口供用户方便快捷地与游戏环境进行交互。如果想快速了解游戏，可以通过实时人机对战接口在本地快速地开启一局游戏。同时 GoBigger 还提供了方便的标准 gym.Env 的接口以供研究人员学习他们的策略。用户也可以自定义游戏的初始环境，来进行更多样化的游戏对局。

CHAPTER 2

索引

2.1 安装

2.1.1 前置需求

我们已经在以下系统版本中进行过测试:

- Centos 7
- Windows 10
- MacOS

同时，我们推荐使用 Python 版本为 `3.6.8`。

2.1.2 快速安装 GoBigger

我们可以通过 PyPI 直接安装：

```
pip install gobigger
```

同样，也可以通过 conda 进行安装：

```
conda install -c opendilab gobigger
```

如果想要使用最新的版本，可以通过源码进行安装。首先，通过 Github 下载 GoBigger 源码。

```
git clone https://github.com/opendilab/GoBigger.git
```

然后，我们从源代码进行安装。

```
# install for use
# Note: use `--user` option to install the related packages in the user own␣
↪directory(e.g.: ~/.local)
pip install . --user

# install for development(if you want to modify GoBigger)
pip install -e . --user
```

2.2 快速开始

2.2.1 通过代码与游戏环境进行交互

在安装完成之后，可以用以下代码快速实现与游戏环境进行交互：

```
import random
from gobigger.envs import create_env

env = create_env('st_t2p2')
obs = env.reset()
for i in range(1000):
    actions = {0: [random.uniform(-1, 1), random.uniform(-1, 1), 0],
               1: [random.uniform(-1, 1), random.uniform(-1, 1), 0],
               2: [random.uniform(-1, 1), random.uniform(-1, 1), 0],
               3: [random.uniform(-1, 1), random.uniform(-1, 1), 0]}
    obs, rew, done, info = env.step(actions)
    print('[{}] leaderboard={}'.format(i, obs[0]['leaderboard']))
    if done:
        print('finish game!')
        break
env.close()
```

在上述代码中，首先构建了环境，然后通过 `env.step()` 完成游戏每一步的进行，并获取到对应的 `observation`，`reward`，`done`，`info` 等信息。执行之后，将会得到类似下面的输出，给出了每一帧排行榜信息。

```
[0] leaderboard={0: 3000, 1: 3100.0}
[1] leaderboard={0: 3000, 1: 3100.0}
[2] leaderboard={0: 3000, 1: 3100.0}
```

(续下页)

(接上页)

```
[3] leaderboard={0: 3000, 1: 3100.0}
[4] leaderboard={0: 3000, 1: 3100.0}
[5] leaderboard={0: 3000, 1: 3100.0}
[6] leaderboard={0: 3000, 1: 3100.0}
[7] leaderboard={0: 3000, 1: 3100.0}
[8] leaderboard={0: 3000, 1: 3100.0}
[9] leaderboard={0: 3000, 1: 3100.0}
[10] leaderboard={0: 3000, 1: 3100.0}
...
```

2.2.2 自定义游戏环境

用户也可以选择通过修改配置 cfg，并通过我们提供的 `gobigger.envs.create_env_custom` 方法来自定义游戏环境。`gobigger.envs.create_env_custom` 方法接收两个参数，第一个参数是 `type`，可选值为 `st` 或 `sp`，分别代表标准比赛模式，和独立动作比赛模式。关于两种模式的介绍具体可以看。以下我们基于标准比赛模式举几个简单的例子。

修改游戏中的队伍数量和玩家数量

如果用户想要在游戏中存在 6 个队伍，每个队伍中含有 2 名玩家，那么可以修改 `team_num` 和 `player_num_per_team`。

```
from gobigger.envs import create_env_custom

env = create_env_custom(type='st', cfg=dict(
    team_num=6,
    player_num_per_team=2
))
```

修改游戏时长

游戏内默认每秒 20 帧。如果用户想要将游戏时长设置为 20 分钟，即 24000 帧，可以修改 `frame_limit`。

```
from gobigger.envs import create_env_custom

env = create_env_custom(type='st', cfg=dict(
    frame_limit=24000
))
```

修改游戏地图大小

如果用户想要拥有一个更大的地图，可以修改 map_width 和 map_height。注意更大的地图可能会导致 step 速度变慢。

```
from gobigger.envs import create_env_custom

env = create_env_custom(type='st', cfg=dict(
    map_width=1000,
    map_height=1000,
))
```

2.3 GoBigger 是什么

2.3.1 总览

GoBigger 是一个类 Agar 游戏，后者是一款风靡全球的游戏。在 GoBigger 中，玩家需要在一张平面图中操控他的分身球。在地图中，还会有食物球，荆棘球，孢子球，以及其他的玩家的分身球。在有限的时间内，玩家需要通过吃掉其他的球来吸收他们的质量，并转化为自己的质量。在比赛结束的时候，质量最大的玩家将会获得胜利。为了提高游戏的对抗性，玩家还可以吃掉其他比他小的玩家来快速发育。因此，在游戏中，玩家需要兼顾快速发育和躲避风险，从而一步步获取到最多的质量以获得游戏的胜利。

为了更方便的介绍游戏规则，我们首先介绍 GoBigger 中出现的各种球。

2.3.2 游戏中的球

- 食物球

  食物球是游戏中的中立资源。他们不属于任何玩家。他们会在游戏中源源不断地被补充进来，并且不会改变位置直到被吃掉为止。如果某个玩家操控的分身球吃掉了一个食物球，那么食物球的质量将会被传递到了分身球中。地图中的食物球数量会存在上限。

- 荆棘球

  荆棘球也是游戏中的中立资源。与食物球不同在于，荆棘球一般拥有更大的尺寸，同时地图中的荆棘球数量上限会比食物球少很多。如果一个玩家的分身球吃掉了荆棘球（前提是分身球比荆棘球大），荆棘球的质量会被传递到分身球内，同时分身球会被引爆并分裂成多个小的分身球。在游戏中，分裂产生的小分身球会呈放射状发射出去，并在短时间内速度衰减。因此，虽然吃掉荆棘球是一个很好的发育方式，但是带来的分裂效果会使得玩家的分身球处在一个危险的境地中（可能会被其他玩家吃掉）。

  荆棘球的另一个特点是可以被玩家通过吐孢子的方式移动。如果某个玩家的分身球对着荆棘球进行吐孢子，那么荆棘球将会吃掉这个孢子，获得质量，并朝着孢子的移动方向前进一小

段距离。因此，玩家可以通过移动荆棘球来让更大的玩家的分身球引爆，从而寻找机会吃掉分裂出来的小球。

- 孢子球

  孢子球是由玩家的分身球通过吐孢子的技能产生的。他们会获得一个初速度并移动一小段距离，然后静止在原地。孢子球可以被比他大的玩家和荆棘球吃掉。

- 玩家的分身球

  玩家的分身球就是玩家在游戏中操控的球。玩家可以对它的运动方向进行任意的改变，还可以吃掉比它小的其他球。在吃掉其他球的瞬间，玩家的分身球会获取到被吃球的质量，并且半径变大。为了增强游戏的可操作性，每个玩家的分身球都会有以下两种技能：

  - 吐孢子

    吐孢子可以帮助玩家的分身球快速减少体积。体积越小，移动速度的上限将会越高。当某个分身球吐孢子时，孢子会有一个初速度，并在一定时间内速度衰减为零。

  - 分裂

    分裂技能可以帮助玩家分裂成相同大小的两部分。分裂之后，由于单个分身球的体积变小，对应的移动速度将会提高。分裂也是有代价的。请注意，在分裂之后，玩家的分身球会进入冷却期。在冷却期期间，分身球无法被自身的其他分身球合并。

2.3.3 游戏规则

如下，有一些值得注意的规则：

1. 玩家的分身球的质量会不断减少。GoBigger 设置了一个衰减系数，玩家球每秒的实际衰减系数会在默认衰减系数的基础上乘以球本身的半径。因此衰减在玩家球质量非常大的时候体现地更为明显。
2. 如果玩家的所有球都被吃掉了，那么他会立即随机在地图中重生。
3. 玩家的视野大小是由玩家的分身球位置所决定的。我们计算玩家所有分身球的质心，并获取到它的最小外接矩形，并在此基础上进行放大来决定该玩家的视野范围。同时，我们也会指定玩家的最小视野范围。玩家的分身球的相对距离越远，所能看到的视野范围将会越大。
4. 玩家的每个分身球都会根据其半径大小存在一个速度上限。在游戏中，半径越大，移动速度将会越慢。

2.3.4 高级操作

2.4 GoBigger 引擎设计

2.4.1 总览

本节主要说明了 GoBigger 的引擎设计，其中包括各种球的详细运动逻辑设定。如果读者想要根据 GoBigger 去开发新的环境，或者想要更深入了解 GoBigger 的研发细节，可以仔细阅读本节。为了方便阐述，我们将按顺序介绍每一种球的逻辑设定，并在其中穿插和其他球的交互过程。如果你对 GoBigger 中基础的球单位还不熟悉，建议先查阅上一节（GoBigger 是什么）。注意，以下的说明均基于 `st_t4p3` 的设置。

2.4.2 所有球的共同点

1. 在 GoBigger 中，你可以看到不同的球有不同的大小。我们定义每个球都有一定的分数 `score`，并通过分数可以计算出对应的半径 `radius`。

```
import math

def score_to_radius(score):
    return math.sqrt(score / 100 * 0.042 + 0.15)
```

2. 食物球是不可移动的，但是荆棘球，孢子球，分身球都包含速度属性，也就是说他们都可以移动。
3. 如果两个球存在可以吃与被吃的关系（例如分身球之间，分身球和所有球，荆棘球和孢子球之间），那么当两个球出现圆心重合的情况时，将会通过判断二者分数来判断是否能发生吞噬。我们规定当大球的分数超过小球的 `1.3` 倍时才能发生吞噬。
4. 所有球在运动过程中圆心都不会超出地图边界。
5. 我们默认游戏内的 `FPS` 是 `20`。也就是说，一秒内游戏状态会更新 `20` 次。同时，默认情况下每次调用 `env.step` 会使得环境前进两帧，用户提供的动作会在第一帧做完，然后第二帧赋予空动作。

2.4.3 食物球

1. 食物球是游戏中的中立资源。在 `st_t4p3` 的配置中，开局时地图上的食物球数量会有 `800` 个，且数量上限为 `900` 个。每隔 `8` 帧，将会补充（数量上限-当前数量）* `0.01` 个食物球，并使得食物球的数量不会超过数量上限。
2. 食物球的初始分数是 `100`。也就是说，如果一个分身球吃掉了一个食物球，那么这个分身球的分数会增加 `100`。

2.4.4 荆棘球

1. 也称为刺球。荆棘球也是游戏中的中立资源。在 st_t4p3 的配置中，开局时地图上的荆棘球数量会有 9 个，且数量上限为 12 个。每隔 120 帧，将会补充向上取整（数量上限-当前数量）* 0.2 个荆棘球，并使得荆棘球的数量不会超过数量上限。
2. 荆棘球的初始分数会从 10000 到 15000 范围内随机选择。
3. 当玩家向荆棘球吐孢子球的时候，如果孢子球和荆棘球发生了圆心覆盖，荆棘球会吃掉孢子球，同时往孢子球的运动方向产生一个为 10 的初速度，并且该速度在 20 帧内均匀衰减到 0。

2.4.5 孢子球

1. 孢子球的大小是固定的，分数固定为 1400。
2. 孢子球被吐出的时候速度是固定的，为 30，并且该速度在 20 帧内均匀衰减到 0。

2.4.6 分身球

分身球的大小随着不断吃球而变大。分身球的初始分数为 1000。单个玩家最多拥有 16 个分身球。每个分身球的分数超过 3200 时才可以使用吐孢子 eject 技能，超过 3600 时才可以使用分裂 split 技能。

速度

分身球的速度由三部分矢量共同作用：玩家操作，多个球存在时导致的向心力，以及分裂或吃荆棘球之后带来的逐渐衰减的速度。玩家操作和向心力带来的加速度在每一帧会乘以每帧时间长度来作为速度的改变量。

1. 玩家操作：如动作空间定义的那样，玩家可以提供一个单位圆内的任意一点 (x,y) 来改变分身球的速度。具体来说，GoBigger 会首先将这一点归一化到单位圆内（如果在圆外则归一化到圆上，否则不做处理），然后乘以权重 30 来作为加速度。
2. 向心力：当玩家拥有多个分身球的时候，多个球内部会产生一个向心力，该力指向质心。实际上向心力不会直接使用，会除以半径之后作为真正的向心力，并乘以权重 10 来作为加速度。
3. 分裂或吃荆棘球之后带来的逐渐衰减的速度：分身球吃掉荆棘球分裂出来的新的分身球会拥有一个分裂后新的初始速度。这个速度的模长为和分裂后的半径有关，具体为 (260 - 20 * radius) / 7，方向背离圆心。该速度会在 14 帧内均匀衰减到 0。如果分身球是通过分裂技能得到，那这个初始速度的模长的计算公式为 (95 + 19 * radius) / 7。
4. 玩家操作和向心力带来的速度会受到速度上限的限制。速度上限和球的半径有关，并会将玩家操作和向心力中较大之一来作为 ratio 对速度上限进行调整。具体公式为 (2.35 + 5.66 / radius) * ratio。

吃荆棘球

1. 分身球每次吃掉荆棘球的时候，会分裂出不超过上限 10 个新的分身球。举个例子，如果玩家当前只有两个分身球，而其中一个分身球吃掉了一个荆棘球，那他会分裂出新的 10 个分身球，因此此时该玩家总共有 12 个分身球；如果玩家当前有 10 个分身球，而其中一个分身球吃掉了一个荆棘球，那他会分裂出新的 6 个分身球，因此此时该玩家总共有 16 个分身球。
2. 分身球吃掉荆棘球分裂出来的新球的最大分数为 5000。举个例子，分身球当前分数是 23000，他吃掉了一个分数为 10000 的荆棘球，那么他的分数就会变成 33000。与此同时，这个分身球会分裂出新的 10 个分身球，按照均匀分配，每个分身球的分数为 3000。
3. 吃荆棘球分裂之后，新增的球的位置均匀分布在周围，且总会有新增的球分身处于原始球右侧水平位置。

分裂

1. 分裂出来新的分身球会在玩家指定的方向上，如果没有指定方向，则会出现在原始球的运动方向上。
2. 分裂技能会将原始球的分数均分到两个分裂后的球上。
3. 分裂出来新的分身球也会拥有原始分身球的速度。
4. 无论是分裂后还是吃荆棘球后，玩家的分身球（包括触发分裂和吃荆棘球，以及这俩操作后新增的分身球）都会进入长度为 20 秒的冷却阶段。在冷却阶段内的分身球无法触发和自己其他的分身球合并的操作。此外，在分身球完成一次合并之后，会重新进入冷却阶段。

吐孢子

1. 分身球执行吐孢子操作后产生的孢子球会出现在玩家指定的方向上，如果没有指定方向，则会出现在分身球的运动方向上。
2. 分身球每吐一个孢子球，相当于把自身分数中割离了 1400 分到新的孢子球上。

2.5 实时游玩

GoBigger 允许用户在个人电脑中实时游玩。同时也提供了多种游戏模式供用户选择。

备注： 如果你还在使用 GoBigger v0.1.x 系列，建议升级以获得更好的体验。可以使用 `pip install --upgrade gobigger` 来快速获取最新版本的 GoBigger。

2.5.1 标准比赛模式 + 部分视野

该模式下，玩家只能看到周围一定范围内的视野。可以通过以下代码启动游戏：

```
python -m gobigger.bin.play --mode st --vision-type partial
```

在本模式中，鼠标可以用来操控玩家控制的球，三个技能分别是 Q，W。Q 技能是在移动方向上吐孢子，W 技能是将用户的球进行分裂。玩家的视野由球的相对位置决定。尝试分散玩家的球，这样可以获得更大的视野。

2.5.2 标准比赛模式 + 全局视野

可以通过以下代码启动游戏：

```
python -m gobigger.bin.play --mode st --vision-type full
```

2.5.3 独立动作比赛模式 + 部分视野

独立动作比赛模式下，玩家的每个球都单独接受一个动作。但是由于比较难操控，我们在这个模式下还是只根据鼠标和键盘接收一个动作，并将这个动作赋给玩家的所有球。可以通过以下代码来启动游戏：

```
python -m gobigger.bin.play --mode sp --vision-type partial
```

2.6 游戏空间设计

2.6.1 标准比赛模式

标准比赛模式指的是和 agar 之类的游戏相同，玩家每帧只能提供一个动作，然后所有的玩家球都会执行这一个动作。

动作空间

由于玩家操控的每个球只能进行移动，吐孢子，分裂，停止，因此 GoBigger 的动作空间是比较简单的：

```
action = [x, y, action_type]
```

- **x，y: 是单位圆中的一个点（x，y），用来代表玩家对球的加速度的操控.**
  - GoBigger 会对（x，y）进行归一化，保证其模长不会超过 1。
  - 如果用户不提供加速度变化，可以提供（None，None）表示不对移动进行改变。
- **action_type: Int**

- 0: 代表只进行移动，实际上每次调用 env.step 都会进行移动。
- 1: 代表在给定方向上吐孢子。如果方向无指定（即为 (None, None)），则在移动方向上执行。
- 2: 代表在给定方向上进行分裂。如果方向无指定（即为 (None, None)），则在移动方向上执行。

我们希望玩家可以更灵活的使用分身球的各个技能，并希望运动方向不会对技能的选择有所限制。因此，我们允许玩家在使用技能时指定的方向和运动的方向可以不同。例如，当玩家的分身球正在往右运动时，如果想要往下吐孢子，只需要指定 action_type=1 并同时指定 (x, y)，分身球即可一边往右继续移动，一边往下吐孢子。下面给出了一个简单的例子。

另外，通过 x，y 与 action_type 的巧妙配合可以实现一些有用的操作。例如，在吐孢子的时候设置 x 和 y 均为 None，可以实现交叉吐孢子。例如：

向环境提交动作

对于多个玩家的情况，需要指定每个动作与玩家的对应关系。提交的动作应当是一个字典，每个 key-value-pair 应当是某个玩家的 id 和他在这一帧需要做的动作。因此，可以遵循如下代码来提交动作：

```
team_infos = env.get_team_infos()
actions = {}
for team_id, player_ids in team_infos.items():
    actions.update({player_id: [random.uniform(-1, 1), random.uniform(-1, 1), -1] for␣
↪player_id in player_ids)})
obs = env.step(actions)
```

状态空间

在游戏的每次 step 之后，用户可以获取到分身球视野下的游戏状态。

```
obs, reward, done, info = env.step()
global_state, player_states = obs
```

global_state 包含一些全局信息，具体如下：

```
{
    'border': [map_width, map_height], # 地图大小
    'total_frame': total_frame, # 整局游戏的总帧数
    'last_frame_count': last_frame_count,    # 当前已经过去了的帧数
    'leaderboard': { team_name: team_size } #␣
↪当前的排行榜信息，包含每个队伍的分数。队伍的分数是队内玩家分数之和
}
```

player_states 包含了每个玩家所能获得的信息，根据 player_id 来区分，具体如下：

```
{
    player_id: {
        'rectangle': [left_top_x, left_top_y, right_bottom_x, right_bottom_y], # 视野框在全局视野中的位置
        'overlap': {
            'food': [[position.x, position.y, radius, score], ...],   # 视野内食物球信息，分别是位置xy，半径，分数
            'thorns': [[position.x, position.y, radius, score, vel.x, vel.y], ...], # 视野内荆棘球信息，分别是位置xy，半径，分数，当前速度xy
            'spore': [[position.x, position.y, radius, score, vel.x, vel.y, owner], ...],  # 视野内孢子球信息，分别是位置xy，半径，分数，当前速度xy，来自玩家的id
            'clone': [[[position.x, position.y, radius, score, vel.x, vel.y,direction.x, direction.y,
                        player_id, team_id], ...], # 视野内玩家球信息，分别是位置xy，半径，分数，当前速度xy，当前方向xy，所属玩家id，所属队伍id
        },
        'team_name': team_name, # 当前玩家所属队伍id
        'score': player_score, # 当前玩家的得分
        'can_eject': bool, # 当前玩家能否执行吐孢子动作
        'can_split': bool, # 当前玩家能否执行分裂动作
    },
    ...
}
```

player_states 中的 overlap 代表的是当前玩家视野中出现的球的结构化信息。overlap 是一个简单的字典，每个键值对代表了视野中的一种球的信息。overlap 中包含了食物球，荆棘球，孢子球，分身球的结构化信息。具体来说，例如我们发现 food 字段的内容为 `[[3.0, 4.0, 2, 2], ...]`（简单起见这里只展示了列表中的第一个元素），那么其中的含义是玩家的视野中，坐标 (3.0, 4.0) 位置存在一个半径为 2 的食物球，同时这个食物球的分数是 2。

请注意，每一种球的信息列表的长度是不确定的。例如，在当前帧视野中一共有 20 个食物球，那么当前 food 对应的列表长度为 20。在下一帧，视野内的食物球如果变为 25，则对应的列表长度将会变成 25。此外，如果某个球只有一部分出现在玩家视野中，GoBigger 也会在 overlap 中给出该球的圆心和半径信息。

奖励

在游戏的每次 step 之后，用户可以获取到游戏默认的奖励。

```
_, reward, _, _ = env.step()
```

游戏内设置的默认奖励是非常简单的，是该玩家当前帧的总分与上一帧的总分之差。用户可以通过玩家的状态信息来设计更复杂的奖励。

其他信息

GoBigger 提供了非常丰富的统计信息，并将这些信息放在了 `info`` 中。在游戏的每次 ``step` 之后，用户可以获取到。

```
_, _, _, info = env.step()
```

具体来说，`info` 是一个字典，在 `st_t2p2` 的环境下能得到如下所示的信息：

```
info = {
    'eats': { # 每个玩家id和他对应的信息
        0: {
            'food': 382, # 整局游戏中吃了多少个食物球
            'thorns': 2, # 整局游戏中吃了多少个荆棘球
            'spore': 0,  # 整局游戏中吃了多少个孢子球
            'clone_self': 38, # 整局游戏中吃了多少个自己的玩家球
            'clone_team': 4,  # 整局游戏中吃了多少个队友的玩家球
            'clone_other': 27, # 整局游戏中吃了多少个对手的玩家球
            'eaten': 3, # 整局游戏自己被其他对手吃了多少个玩家球
        },
        1: {...},
        2: {...},
        3: {...},
    },
}
```

2.6.2 独立动作比赛模式

独立动作比赛模式指的是玩家每帧需要对他的所有玩家球提供动作。玩家的每个玩家球都可以独立执行动作。

动作空间

最小动作单元和标准比赛模式是一样的，只是在进行 `env.step(actions)` 的时候，`actions` 的格式应该如下：

```
actions = {
    player_id: {
        ball_id: [x, y, action_type],
        ...
    },
    ...
}
```

这里面的 ball_id 可以从每一帧拿到的 obs 来确定。每个 ball_id 会唯一对应到玩家的一个分身球。

状态空间

大部分和标准比赛模式是一样的，唯一不同在于 clone 球部分会增加 ball_id 的信息。这个信息可以用来告诉玩家在提供 actions 的时候 ball_id 可以从这里拿。

player_states 具体如下：

```
{
    player_id: {
        ...
        'overlap': {
            ...
            'clone': [[[position.x, position.y, radius, score, vel.x, vel.y,␣
↪direction.x, direction.y,
                        player_id, team_id, ball_id], ...], #␣
↪视野内玩家球信息，分别是位置xy，半径，当前速度xy，当前方向xy，所属玩家id，所属队伍id，球的id
        },
        ...
    }
}
```

2.7 GoBigger 环境

2.7.1 总览

本节主要说明了 GoBigger 的环境设计，其中包括给定好的环境的定义，以及如何自定义生成对应的环境。

2.7.2 已经定义好的环境

GoBigger 定义了一些基础环境，可以通过下面的代码生成这些基础环境

```
from gobigger.envs import create_env
env = create_env('st_t4p3')
```

其中，create_env() 接收的第一个参数是一个字符串，是我们已经定义好的环境的名称。类似的，除了 st_t4p3 以外，我们还有 st_t2p2，st_t3p2 可供选择。下面详细介绍一下各个环境的区别。

1. st_t4p3: 创建了一个有四支队伍，每支队伍三个玩家的游戏场景。这是我们定义的标准游戏场景，包括其中的各种球的刷新速度等参数都已经调节至一个比较健壮的水平。用户可以使用这个环境来解决一些复杂空间场景下的合作和对抗问题。

2. st_t2p2: 考虑到 st_t4p3 可能过于庞大，因此我们提供了一个更小的游戏场景。在这个场景内，只有两支队伍，每支队伍两个玩家。同样，地图尺寸，食物球和荆棘球的数量也受到了相应的削减。此外，为了缩减玩家前期发育的时间，我们在 st_t2p2 将玩家第一次出生时的分数设置成了 13000 来使得玩家可以快速进行对抗，而这个值在 st_t4p3 中是 1000。
3. st_t3p2: 这是一个间于 st_t2p2 和 st_t4p3 的中等环境，有三支队伍，每支队伍两个玩家。和 st_t2p2 一样，也设置了较高的出生分数来减少发育时间。

此外，上述环境默认都是每两帧做一个动作（一秒 20 帧）。如果想要更长的动作间隔，可以通过下面代码设置：

```
from gobigger.envs import create_env
env = create_env('st_t4p3', step_mul=10)
```

此外，我们还有更多已经定义好的环境，如下：

Name	Agents Size	Map Size	Food	Thorn	Init Size	Limited Frame
st_t1p1	1x1	32x32	[65,75] [130,150]	[1,2]	1000	3600(3min)
st_t1p2	1x2	48x48 48x48	[130,150]	[2,3] [2,3]	1000	3600(3min)
st_t2p1	2x1	64x64 88x88	[260,300]	[3,4] [5,6]	1000	3600(3min)
st_t2p2	2x2	128x128	[500,560]	[9,12]	13000	3600(3min)
st_t3p2	3x2	128x128	[800,900]	[10,12]	13000	3600(3min)
st_t4p3	4x3	144x144	[900,1000]	[10,12]	1000	14400(12min)
st_t5p3	5x3	144x144	[900,1000]	[11,13]	1000	14400(12min)
st_t5p4	5x4		[1000,1100]		1000	14400(12min)
st_t6p4	6x4				1000	14400(12min)

2.7.3 自定义生成对应的环境

GoBigger 丰富的配置文件设计使得用户可以很方便地设置游戏内的每一个细节。

如果想要在某个我们已经定义好的环境的基础上进行简单修改，例如在 st_t2p2 修改一局时长，可以如下：

```
from gobigger.envs import create_env
env = create_env('st_t2p2', dict(
    frame_limit=10*60*20,
))
```

这样，开启的新环境就会在 st_t2p2 的基础上变成了 10 分钟一局。

2.8 回放系统

GoBigger 的回放系统支持三种选择，可以通过环境的配置文件来进行选择。涉及到的配置项如下：

```
config = dict(
    ...
    playback_settings=dict(
        playback_type='none', # ['none', 'by_video', 'by_frame']
        by_video=dict(
            save_video=False,
            save_fps=10,
            save_resolution=552,
            save_all=True,
            save_partial=False,
            save_dir='.',
            save_name_prefix='test',
        ),
        by_frame=dict(
            save_frame=False,
            save_all=True,
            save_partial=False,
            save_dir='.',
            save_name_prefix='test',
        ),
        by_action=dict(
            save_action=False,
            save_dir='.',
            save_name_prefix='test',
        ),
    ),
    ...
)
```

`playback_type` 可以是 `['none', 'by_video', 'by_frame']` 中的其中一种。其中，

- `none`: 代表不需要存回放
- `by_video`: 代表直接保存录像，保存文件后缀是 `.mp4`。一般来说，`st_t4p3` 环境存下来的录像在 80M 左右。
- `by_frame`: 代表存每一帧的变化量，保存文件后缀是 `.pb`。一般来说，`st_t4p3` 环境存下来文件在 25M 左右。

2.8.1 直接保存录像

如果选择 playback_type='by_video'，具体的配置项可以像下面这样：

```
env = create_env('st_t4p3', dict(
    playback_settings=dict(
        playback_type='by_video',
        by_video=dict(
            save_video=True,
            save_dir='.', # 需要保存录像的目录位置
            save_name_prefix='test', # 保存录像名字的前缀
        ),
    ),
))
```

2.8.2 直接保存 pb 文件

如果选择 playback_type='by_frame'，具体的配置项可以像下面这样：

```
env = create_env('st_t4p3', dict(
    playback_settings=dict(
        playback_type='by_frame',
        by_frame=dict(
            save_frame=True,
            save_dir='.', # 需要保存录像的目录位置
            save_name_prefix='test', # 保存录像名字的前缀
        )
    ),
))
```

得到保存后的 .pb 文件之后，需要通过我们给定的播放器来查看。在命令行中执行下面的命令来打开播放器。

```
python -m gobigger.bin.replayer
```

打开播放器之后，需要选择你想要查看的 .pb 文件。然后就可以开始看了。播放器支持倍速播放，包括 2 倍，4 倍，8 倍（通过点击左下角的按钮）。同时支持拖动进度条。

2.9 游戏配置介绍

2.9.1 总览

为了让玩家更进一步了解游戏机制的实现，我们开放了部分参数供玩家进行调整。我们希望玩家可以通过修改对应的参数，来实现各种不同的环境。同时，也可以设计自己的代理环境，用于对算法的快速验证。

GoBigger 将可配置参数统一放在 `gobigger/server/server_default_config.py`。其中对可配置参数进行详细介绍。

2.10 碰撞检测算法

2.10.1 总览

为了在游戏的每一帧中检测球体的碰撞，以便更新球体的状态，我们需要设计高效的碰撞检测算法。因此，我们设计了四种碰撞检测算法，并将它们封装成以下四类。下面仅介绍算法相关结果，更多细节请查看源码。

2.10.2 算法介绍

四种算法的介绍和理论时间复杂度如下：

ExhaustiveCollisionDetection:

$O(n * m)$

暴力解法，其中 n 表示球的总数，m 表示要询问的球数。

PrecisionCollisionDetection:

$O(n * log(n) + \Sigma r * logn + p)$

卡精度解法。其中 n 表示球的总数，m 表示要询问的球数，k 表示我们设置的精度，p 表示实际碰撞的球体数量。

RebuildQuadTreeCollisionDetection:

$O(n * log(n) + m * log(n) + p)$

重建四叉树解法。其中 n 表示球的总数，m 表示要询问的球数，p 表示实际碰撞的球体数量。

RemoveQuadTreeCollisionDetection:

$O(r * log(n) + m * log(n) + p)$

优化后的重建四叉树解法，增加了对四叉树的删除和维护。其中 n 表示球的总数，m 表示要询问的球数，r 表示位置状态发生变化的球体数量，p 表示实际碰撞的球体数量。

为了测试这些算法的效率，我们修改了球总数、查询数、改变球数、迭代轮数等参数来设置测试场景。下表中的数据来自最具代表性的场景：

- T：地图中所有球的数量
- Q：查询球的数量，通常表示地图中移动的球
- C：需要修改的球的数量，即碰撞次数

	Exhaustive	Precision	Rebuild QuadTree	Remove QuadTree
T=3000 Q=300 C=600	688ms	14ms	47ms	48ms
T=3000 Q=300 C=1500	1067ms	16ms	50ms	178ms
T=10000 Q=1000 C=2000	8384ms	61ms	339ms	497ms
T=10000 Q=2000 C=5000	12426ms	86ms	586ms	2460ms
T=30000 Q=6000 C=3000	127000ms	403ms	5691ms	8419ms

为了更直观的看到每种算法的优劣，我们整合了测试数据，绘制了四种算法和各种参数的图表如下：

根据结果，我们可以认为 **PrecisionCollisionDetection** 算法在效率和稳定性方面都远远优于其他算法。

2.11 FAQ

2.11.1 Q1: 如何保存对局录像？

A1

创建 env 的时候传入 `playback_settings` 相关参数，如下图所示。这样在 ** 一局结束 ** 之后会在 `save_dir` 目录下保存 `test.pb` 文件。通过 GoBigger 提供播放器可以看这局比赛。

```
env = create_env('st_t3p2', dict(
    playback_settings=dict(
        playback_type='by_frame',
        by_frame=dict(
```

(续下页)

(接上页)

```
            save_frame=True,
            save_dir='.',
            save_name_prefix='test',
        ),
    ),
))
```

2.11.2 Q2: 比赛最后的获胜条件是什么？

A2

通过计算比赛结束时每个队伍下所有玩家的得分和来进行排序。

2.11.3 Q3: 局部视野的大小有范围限制吗？

A3

玩家的局部视野的大小由其分身球的相对位置决定。我们设置玩家的最小视野是 36*36 的一个矩阵。随着分身球的分散，玩家的最大视野可以达到全局的程度。

2.11.4 Q4: conda 环境（使用的推荐的 python3.6.8）下安装了 gobigger，实际运行的时候出现 libGL error failed to open iris 该怎么办

A4

是 glibc 版本过低却安装了高版本的 libgl 导致的。这里有个相似的问题可以看看 https://askubuntu.com/questions/1352158/libgl-error-failed-to-load-drivers-iris-and-swrast-in-ubuntu-20-04

2.11.5 Q5: 该环境中，智能体能否执行出类似于人类玩家的中吐行为？周围自己的小球把孢子吐给中间的球？

A5

在执行吐孢子操作的时候，可以指定方向。我们把和方向有关的参数（x, y）设置为（0, 0），则玩家球会逐渐减速，同时方向会慢慢转向质心。之后再执行吐孢子的动作，则会实现中吐。

```
action1 = [0, 0, -1] # 停止
action2 = [0, 0, 2] # 吐孢子
```

2.11.6 Q6: 吃荆棘球，自己体积会变大吗？还是说只分裂？

A6

体积也会变大。