MTEAC使用方法

本文将以基本概念-功能-实现方法-代码结构的顺序讲解如何理解和使用该项目。

在讨论概念时，我们仅讨论概念层面的东西，不会涉及文件架构和代码。我们会在后面具体阐述我们是如何将概念实现为代码，二者是如何对应的。然后我们会阐述，如何将概念上的对世界的描述的改动实现为具体的代码。

本文档意在说明MTEAC的运作逻辑，而不会涉及具体的world\_project的内容。我们会在其它文档中阐述mesh\_world的结构和实现等内容。

目录：

基本概念和结构

功能

实现方法

代码结构

自定义world\_project的方法

openAI的适配

其它说明

附录

基本概念和结构

1. 基本概念

1、

环境

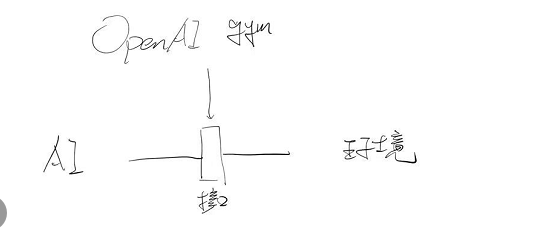
在我们的理解中环境是这样的概念 其能：

在某个时刻有一个具体的、可被描述的【状态】

以OpenAI的格式给AI定义状态和行动空间

接收来自AI的行动指令 并执行指令

随着时间变化



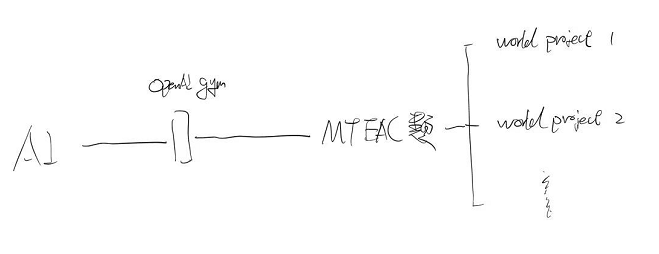
MTEAC类

即world\_env\_interface.py中的WorldEnv类。

该类适配了openAI gym 是和openAI交互的环境类。但该类本身并不是一个环境，而是一个『环境管理器』。该类会运用工厂模式，选择某一个具体的环境作为openAI gym的环境。我们称具体的环境为world project。

World project

我们称每一个具体的环境为一个world project。当MTEAC选择不同的world project时，WorldEnv会具体体现为不同的环境。



2、world project的结构

world project的主要结构包含了状态的描述、生成和状态的变化逻辑。

在MTEAC中，我们认为状态由三个基本元素描述：空间属性、实体和环境变量。

空间属性：比如一个位置（空间）的地形、地势、天气等要素。这些属性是依据空间位置定义的，所以称作空间属性。

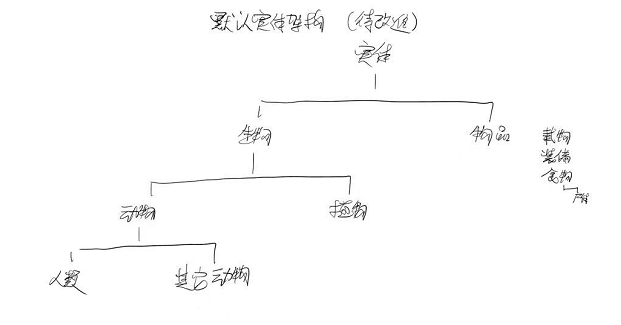
实体：所有可以被移动的『东西』。如不同的动物的物种、不同的物品等。

环境变量：那些具有全局性质的属性，比如时间、季节、昼夜等。

我们一般用简单的二维数组或者数字来表示空间属性和环境变量。对于实体则稍微复杂一些。我们通过将实体进行分类和继承的方法表示不同的实体。比如所有实体可以分为生物和非生物（物品），生物又可以分为植物和动物…我们分别定义『实体』『生物』『非生物』这些概念的抽象类，并给予他们继承关系。我们称其为【**实体架构**】。

3、默认实体架构

在MTEAC中我们设置了一个默认的架构（这一设置还在不断优化中）：



在不同的world project中，所有具体的实体类别都会继承默认架构。比如我们想要定义『猴子』以让我们的世界中有猴子这个东西，那么我们可以在world project中定义Monkey类，并让Monkey类继承默认架构中的动物类。

每一类不同的实体会有不同的属性和逻辑。我们用属性、逻辑、类别来描述一个具体的实体类。比如所有动物都有『移动』的方法和『移动速度』的属性，而植物没有。

我们可以用这样的描述方法描述默认实体架构下的各个概念：

实体Entity：

属性：位置position id

方法：move post\_turn\_change

（在这里我们不特意把get\_id这样的方法列出来）

解释：

抽象的实体就是一个物质存在，所有存在都会有一个位置。

在MTEAC中，为了方便管理一个世界中的实体，我们会给每个实体设定一个独一无二的ID。

post\_turn\_change指的是每个回合结束时 实体随着时间发展发生的变化 比如尸体腐烂 树长高 人变得更饿 等等

生物Creature：

属性：生命值life 甲胄carapace

方法：死亡die 被攻击be\_attack

解释：

生物就是具有【生】和【死】两种状态 生的时候可以死的东西

同时 生物还会会被其它事物伤害

生物还可以繁殖 但我们只在mesh\_world中实现了植物的繁殖 我们原计划模拟生物繁殖的基因 但是由于时间问题没能实现

物品（非生物）Obj：

属性：无

方法：无

解释：

我们还没有给默认框架下的Obj抽象类予以特殊的规定 我们本来想加一些比如『质量』『大小』『形状』之类的属性 由于时间问题我们并没能实现

动物：

属性：行为表action\_list 具体属性在具体的world\_project中定义

方法：

judge\_action\_validity action\_cost action\_interior\_outcome body\_change get\_perception

解释：

动物和植物、物品最大的区别在于其能进行【行为】 动物是能动的 因此动物特有的方法就是动作相关的方法

judge\_action\_validity 根据动作指令和动物此刻的状态判断行为的合法性

action\_cost 得到进行某行为的消耗

action\_interior\_outcome 动作进行成功后 动物状态发生的变化 比如吃东西让动物变得更饱

body\_change 用于身体状态的工具方法

get\_perception 输入一个State实例 输出动物的感知结果

植物：

属性：无

方法：无

解释：

我们暂时没有给植物设置特别的属性 因为植物的大多数逻辑和物品相似

我们没有在基本架构中实现植物的繁殖等方法

人类：

属性：无

方法：无

解释：

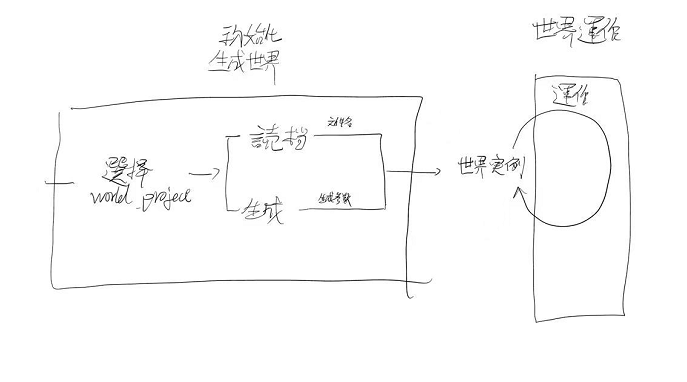
在默认实体架构中 我们没有给人类做特别的设置 但是在mesh\_world中 人类和动物大不相同 人类有许多技能 可以使用工具 可以合成物品…

之所以我们没有在默认架构中加这些东西是因为我们希望对人类和动物的差别的定义可以更灵活一些

这些设定依然还待优化

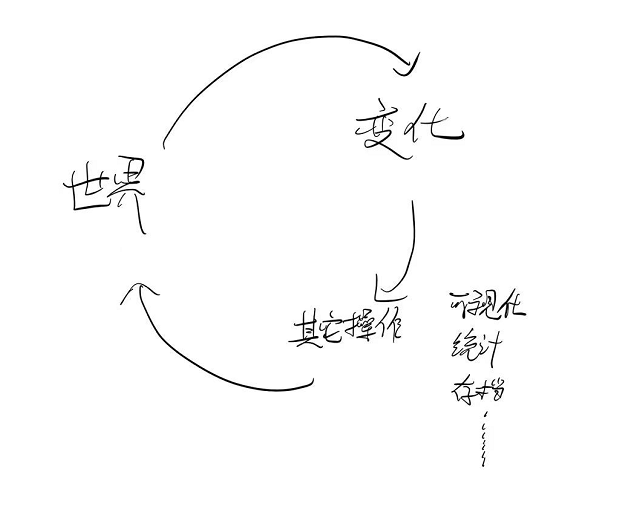
默认的实体架构仅仅为了使定义具体的架构更方便而设置的。各个world\_project都可以定义自己的实体架构，而不必继承或遵循默认架构。

4、MTEAC的运作

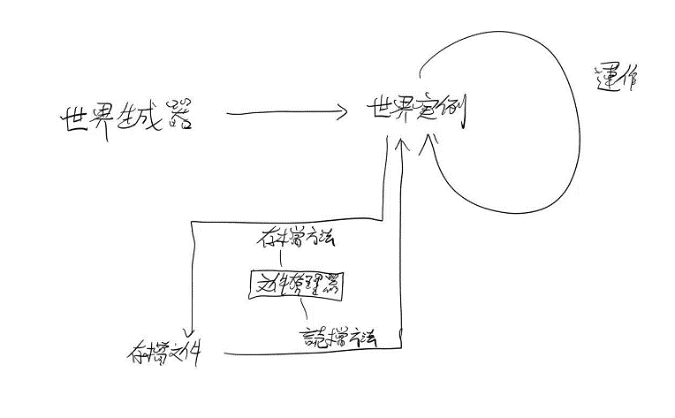


MTEAC类会首先根据所选择的world project进行初始化 生成该project的world的实例

然后根据其实例的进行运作 随着时间从一个状态变为另一个状态



在step()运作以后 可以自定义任意别的方法 比如可视化 统计 存档读档等



如在世界的实例的运作中 任意一次step()运作以后 可以调用File\_processor的存档和读档方法 对世界实例进行存档和读档

功能

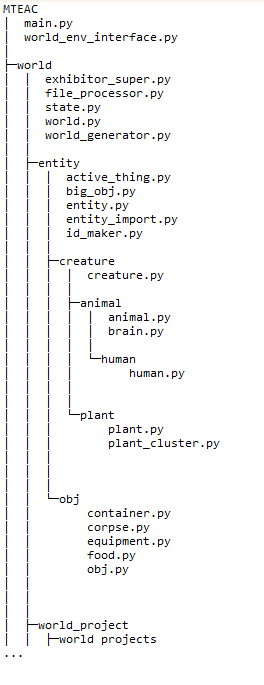
我们本来打算对以下功能内容做个图形界面进行管理 但是由于时间和能力原因没能完成 因此下面描述的功能需要通过手动修改代码的方法实现

本页仅列举能进行的操作和内涵 下一页阐述如何实现这些操作

1. world\_project管理
   1. 创建world\_project
   2. 让MTEAC选择某一个world\_project
   3. 修改/自定义world project
2. 默认存档/读档器
3. 一些不重要的功能
   1. 后台功能
   2. 统计功能

功能的实现方法

文件结构：



其中 main.py是程序的入口（测试用） world\_env\_interface.py是MTEAC类

world文件夹下 的.py文件都是对world projects的抽象类规定

所有world project的文件夹都存放在world\_project文件夹下

默认实体架构则存放在entity文件夹下

1. world\_project管理
   1. 创建world\_project

创建world\_project只需要在world\_project下建立一个文件夹即可 在这之后不必在任何地方进行修改或注册 因为MTEAC类会自动读取文件夹路径

我们提供了更方便的创建world\_project的方法：直接复制blank\_world文件夹 然后把文件夹名字中的”blank”改为自己想要的名字 然后把里面的文件中的所有blank也改为该名字即可（作为类名时首字母要大写）

如果从零开始创建新的文件夹 需要遵循以下格式（该格式待改进）：

* 文件夹名必须以『\_world』结尾

『\_world』前的部分为该world\_project的名称 该名称在构建world\_project中会用到

* 文件夹中必须有一下文件
  + exhibitor.py

该文件中必须要有Exhibitor类 而且必须继承world下的exhibitor\_super.py中的Exhibitor\_super抽象类

* + state.py

该文件中必须要有一个叫\*world\_project的名称\*+“\_state”的类 如blank\_world中的state.py中的类叫做Blank\_state

该类必须继承world下的state.py中的State抽象类

* + world.py

该文件中必须要有一个叫\*world\_project的名称\*+“\_world”的类 如blank\_world中的state.py中的类叫做Blank\_world

该类必须继承world下的world.py中的World抽象类

* + world\_generator.py

该文件中必须要有一个叫“Concrete\_world\_generator”的类

该类必须继承world下的world\_generator.py中的World\_generator抽象类

* + save文件夹（如果使用默认存档/读档器的话）

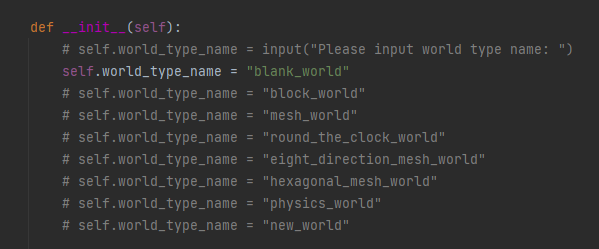
只要有一个空的名字叫save的文件夹就好了

* world.py下的类必须要有get\_openai\_action\_space\_and\_observation\_space方法来规定OpenAI所需的observation\_space和action\_space

该方法返回一个 (action\_space, observation\_space) 格式的元组 两个elements都必须为openAI gym规定的space对象

* 1. 指向world\_project

如果想让MTEAC类选择某一个world\_project 只需要修改MTEAC类（即world\_env\_interface.py下的WorldEnv类）的\_\_init\_\_()方法中的self.world\_type\_name变量为world project的文件夹的名字的字符串即可



* 1. 修改/自定义world project

见后文【自定义的方法】

1. 使用默认存档/读档器

我们只需要在任何想要读档的地方调用file\_processor.py下的archive()或load()方法即可 在默认的存档/读档器中 这两个方法都已经和具体的world project去耦合化了 所以对于所有world project都将适用 但由于其泛用性 且数据结构是json文件 所以默认的存档和读档器或许会更加低效

archive()读入三个参数 state, world\_type\_name, file\_name

state即为当前world的state属性 代表了需要被存档的状态

world\_type\_name即为world project的名称的字符串

file\_name为存档文件的命名名称

调用该方法后 对应的world project的文件夹的save文件夹下将会出现一个以file\_name为名称的json文件 该文件即为存档文件

load()方法读入两个参数 world\_type\_name, file\_name

world\_type\_name和archive()中的相同 为world project的名称的字符串

file\_name为存档文件的命名名称 load()方法将读取该文件 生成一个相应的State实例

在我们的项目中 由于openAI所要求的reset()方法我们还没有实现 因此我们暂时使用读档的方法代替之 这是另一个待优化的点

1. 一些不重要的功能
   1. 后台功能
   2. 统计功能

我们实现了简单的后台和统计功能 但是效果并不好 因此在这里不多做介绍 我们会在优化完毕以后再做详细介绍

代码结构

概念和代码的对应

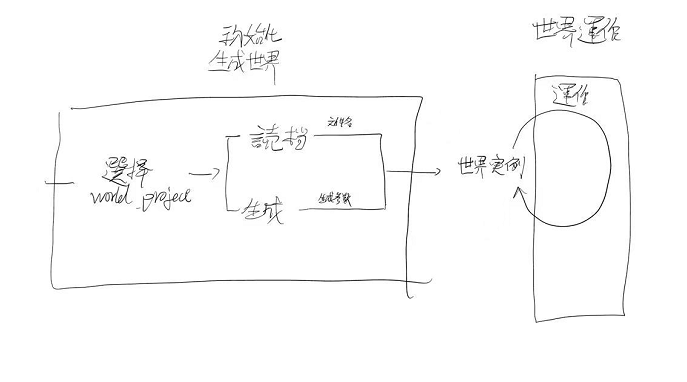
在world文件夹下的world.py state.py是规定世界和状态的抽象类

这两个文件下的注释说明了构建这二者所需的方法和各个方法的含义

执行顺序 以main.py中openai的例子为例

初始化： env = WorldEnv()

运作: env.step()



当调用step时:

World\_env\_interface.py中的step()将会被调用

然后在step()中 对应的world project中的world.py的take\_action()和evolution()方法将被调用

该方法读入一个行动指令和id

Id用于指定执行指令的主体是哪个实体

take\_action()使得指定的实体执行一次指令

evolution()则使得环境随时间变化一次 以及其它实体自行运作一次

自定义world project的方法

我们将自定义world project的一般过程分为如下步骤：

1 具体地描述world的状态 并确定：

这一步骤是在概念的层次确定对world的描述

这个world有哪些属性 这些属性分别用什么样的数据结构表示

各个属性之间的相互作用

属性如何变化

这些属性是随着时间变化 还是依据行为变化 还是依据别的属性变化

初始化世界时如何生成这些属性

2 构思如何实现这些属性 并实现之

找到概念对应的文件的位置

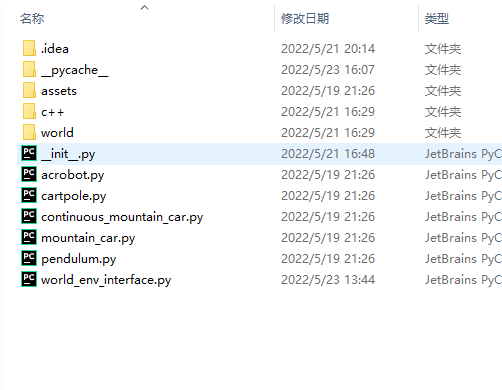
添加属性和逻辑

openAI gym的适配

如果要将MTEAC适配进openAI gym只需要遵循一下步骤：

1. 将MTEAC目录下的world文件夹和world\_env\_interface.py文件夹拖入site-packages\gym\envs\classic\_control文件夹下

注：mesh\_world使用了一些C++代码 放在c++文件夹中 如果要使用mesh\_world 也需要将c++文件夹拖入该文件夹中



在 gym\envs 目录下的\_init\_.py 文件中加入如下的代码：

register(

id="MTEAC-v0",

entry\_point="gym.envs.classic\_control:WorldEnv",

max\_episode\_steps=200,

reward\_threshold=100,

)

在 gym\envs\classic\_control 目录下的\_\_init\_\_.py 文件中加入下面这行代码：

from gym.envs.classic\_control.world\_env\_interface import WorldEnv

完成以上步骤之后，就可以调用我们自行创建的环境了

新建一个 main.py，然后写入 以下代码，来调用环境

import gym  
  
# program entry  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 ai\_mode = 1  
 ai\_num = 2  
  
 env = gym.make("MTEAC-v0")  
 env.set\_ai\_num(ai\_num)  
 env.reset()  
  
 """  
 openAI mode  
 """  
 while True:  
 # Take a random action  
 action = [env.action\_space.sample() for num in range(ai\_num)]  
 obs, reward, done, info = env.step(action)  
  
 if done[0] == 1:  
 print("pac man win")  
 break  
 elif done[0] == -1:  
 print("pac man lost")  
 break  
  
 # Render the game  
 env.render("ai")  
  
 if done is True:  
 break  
  
 env.close()

其它说明

一、关于注释

我们在MTEAC的抽象类文件中添加了注释以说明方法和类对应的概念。

二、关于Demo

我们会在根目录下放置一个视频demo以演示对一个简单的world project的实现

三、关于openAI gym的指令的格式

由于对multi-agent的支持。WorldEnv的step()方法读入的将需要是一个存有许多条指令的数组，而非单一的一条指令。

指令在数组中的下标对应实体的ID。

例如env.step([“eat”, “run”])指的是id为1的实体进行eat的动作，而id为2的物体进行run的动作。

四、main.py中的ai\_mode和game mode

ai\_mode决定了main.py中的if分支的走向

当ai\_mode=1时 main会以openAI的模式运行WorldEnv

当ai\_mode=2是 main会以game mode运行WorldEnv

game mode指的是用户使用键盘操控实体 而非由ai操控实体的模式

需要注意的是game mode不是所有world project都有的 如果想要给world project添加game mode的功能 需要在world project的world类中设定类属性play\_mode = True（默认为False） 然后在exhibitor中具体写你打算如何让world读取玩家的命令

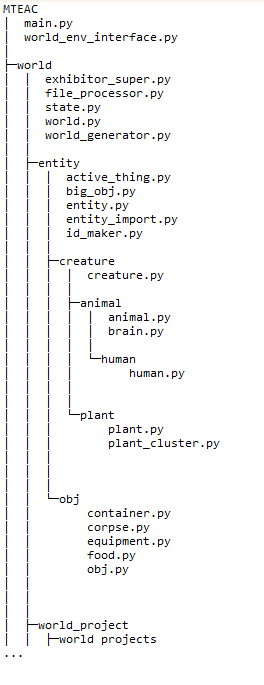
可以参照block\_world 该world project实现了最简的推箱子的游戏

五、未完成的计划

该项目由于时间原因 还存在大量待改进的地方，我们会在之后对这些地方加以修正。（我们会在之后具体列出有哪些地方还待改进）

附录：

文件架构



World\_project的文件要求

如果从零开始创建新的文件夹 需要遵循以下格式（该格式待改进）：

* 文件夹名必须以『\_world』结尾

『\_world』前的部分为该world\_project的名称 该名称在构建world\_project中会用到

* 文件夹中必须有一下文件
  + exhibitor.py

该文件中必须要有Exhibitor类 而且必须继承world下的exhibitor\_super.py中的Exhibitor\_super抽象类

* + state.py

该文件中必须要有一个叫\*world\_project的名称\*+“\_state”的类 如blank\_world中的state.py中的类叫做Blank\_state

该类必须继承world下的state.py中的State抽象类

* + world.py

该文件中必须要有一个叫\*world\_project的名称\*+“\_world”的类 如blank\_world中的state.py中的类叫做Blank\_world

该类必须继承world下的world.py中的World抽象类

* + world\_generator.py

该文件中必须要有一个叫“Concrete\_world\_generator”的类

该类必须继承world下的world\_generator.py中的World\_generator抽象类

* + save文件夹（如果使用默认存档/读档器的话）

只要有一个空的名字叫save的文件夹就好了

* world.py下的类必须要有get\_openai\_action\_space\_and\_observation\_space方法来规定OpenAI所需的observation\_space和action\_space

该方法返回一个 (action\_space, observation\_space) 格式的元组 两个elements都必须为openAI gym规定的space对象

程序的运行顺序

