OpenAI Gym介绍

注意，先前没有观察

All development of Gym has been moved to Gymnasium, a new package in the Farama Foundation that's maintained by the same team of developers who have maintained Gym for the past 18 months. If you're already using the latest release of Gym (v0.26.2), then you can switch to v0.27.0 of Gymnasium by simply replacing import gym with import gymnasium as gym with no additional steps. Gym will not be receiving any future updates or bug fixes, and no further changes will be made to the core API in Gymnasium.

[Read more about the Farama Foundation and the backstory of the transition from Gym to Gymnasium](https://farama.org/Announcing-The-Farama-Foundation)

重点内容： gym以后将不会更新，所有gym的开发转移到了Gymnasium /dʒɪmˈneɪziəm/

对于导入，我们使用 import gymnasium as gym

目前gymnasium版本为27.0只比gym26.2多出0.8个版本，跨度并不大

1. Gym(健身房，训练房)简介

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like pong or pinball.

We provide the environment; you provide the algorithm. You can write your agent using you existing numerical computation library, such as TensorFlow or Theano.

Link: <https://www.gymlibrary.dev/>

旧版介绍总结：

作用：开发强化学习算法，提供工具，环境支持

契合框架：TensorFlow，Theano

目前主要有几个框架，Tensorflow(Keras), Jax, Pytorch, Mxnet, Caffe, Theano, Paddle Paddle

新版简介：

作用仍然没有变化，契合框架根据查询

Tensorflow，pytorch，Jax，Mxnet都契合

有看到存在他人编辑的 openai-Mxnet库

1. Gym简单案例(实例)

# 导入gym库

import gym

# 初始化环境，一般是利用gym.make()方法，其中render\_mode, 以及make()并未在文档中写明\

但render\_mode 表示可视化模式，human表示人们能理解的图像模式，具体参考gym.env.render()文档

env = gym.make("LunarLander-v2", render\_mode="human")

# gym.reset重置环境到最初始的状态

返还最初始观察状态，以及观察状态的信息

observation, info = env.reset(seed=42)

# policy需要人为定义

for \_ in range(1000):

# 根据给定的状态观察observation，策略返还动作

action = policy(observation) # User-defined policy function

# 然后环境根据返还的动作进行更新，返还观察状态，奖励，是否到达终止状态，是否满足截断条件，就是可以提前结束

Info 就是辅助诊断信息

observation, reward, terminated, truncated, info = env.step(action)

# 如果到达终止状态 或 提前截断，那么重新设置环境

if terminated or truncated:

observation, info = env.reset()

# 在以上过程中，我们可以尝试记录多条MDP状态动作奖励序列，并且根据需要获取回报 Total payoff

# 主动关闭环境，垃圾回收。当然环境自己也会自动关闭，当程序退出或者自动垃圾回收

env.close()

1. 部分 api 方法讲解
2. env.render()选择渲染模式，观察文档，感觉并没有返回值

只是提供可视化选择，并且会导致运行的速度变慢

因此一般在训练的时候，是不需要可视化展示的

如果自己编辑一个环境，选择的render(), 往往渲染的直接是直接列表状态

1. env.reset() ,重置环境，回到最初状态

返还初始initial state以及环境信息info

1. env.step()

gym.Env.step(self, action: ActType) → Tuple[ObsType, float, bool, bool, dict]

输入仅有动作，输出则是一个元组

observation (object) reward (float)  terminated (bool)

truncated (bool) info (dictionary)

1. Action

目前对于每个环境的动作，似乎是可以直接从env的属性观察

而且在具体某个环境介绍，github 相关source file，注释中有标识

1. 其他人编写的环境 env

尝试去pip 别人已经写好的环境与对应的动作

1. 如何查看所有环境以及对应环境的action\_space, observation \_space

from gym import envs

Print(envs.registry)

Envv = gym.make(“CartPole-v0”)

Print(Envv.action\_space)

其中Envv后方的可查看参数，自主Tap选择

重点：在了解Gym toolkit后，关键是如何编写自己的环境

对于很多实际问题，我们需要学会如何使用gym来编写属于自己的环境

进行agent交互，训练agent

1. 文件结构管理

目前已知的有两种方法创建环境

第一种：首先，直接将自定义的环境文件，放置在在gym 库文件目录的envs下

比如：D:\Anaconda3\envs\hp\Lib\site-packages\gym\envs\classic\_control

**如何查看具体某个库在环境中的文件路径**

**到该环境下，输入 pip show gym 然后回车，即可**

然后打开该文件路径下的\_\_init\_\_.py文件，其实就是python的包与类创建，注册过程

写入以下语句：from gym.envs.classic\_control.grid\_map import GridEnv

其中grip\_map就是文件名，GridEnv 就是类名

最后在总环境下注册，因为调用环境是一个一个文件夹深入调用，而gym则是直接来到envs

下的\_\_init\_\_.py 位置调用，因此需要在该文件中注册自定义环境的位置与名称

在结尾写入以下：

register(

id = “GridWord-v0” # 该环境名称自定义

entry\_point = ”gym.envs.classic\_control:GridEnv” # 导入的节点，前者是文件路径，后者是类名

reward\_threshold = 20, # 奖励上限，看了一下官方文档，似乎是非必须

# 还可以设置更多，有没有都能调用环境，比如最大迭代次数

max\_episode\_steps = 200

)

第二种：自定义所有环境文件路径，根据第一种特点

首先假定你有一个文件夹在main.py 同一路径下，其实不同也可以，后面可以测试

该文件夹名称及结构为 OwnGyms/ # 可以理解为自己建立了一个环境库

\_\_init\_\_py # 用于注册各种环境，参考register()，必须有

README.md # ReadMe帮助你了解文件内容写的什么，可有可无

envsClass1Name/ # 第一大类环境的名字，比如卡普空的游戏

\_\_init\_\_.py # 对于该类环境下的包的类注册，必须有

Game1Env.py # 卡普空类环境的某个具体环境，如生化危机7

Game2Env.py # 同样某个具体环境，比如生化危机6

envsClass2Name/ #第二大类环境名字

\_\_init\_\_py # 同样该类环境下的注册文件

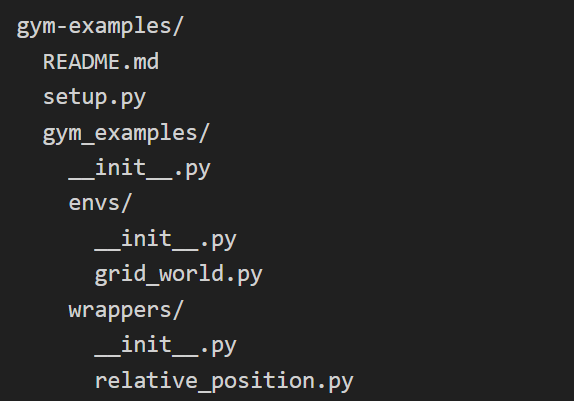
Game3Env.py # 某个具体环境，如辐射4

此时由于是在main.py同一路径下注册的，因此只需要直接 gym.make(“环境名”)就可以直接调用/导入环境

如果OwnGyms在另一个文件路径下(与main.py在同一路径下)，比如Folder/OwnGyms, 此时就是

gym.make(“Floder/环境名”), 以此类推，如果与main.py 独立的路径下，就是

gym.make(“D:/Floder1/Floder/环境名”)



对以上的推理给出怀疑，似乎只能在main同一路径下

因为main中开头要import Floder, 跟python的类打包使用类似

官方给出的实例，在/Folder（也就是gym-examples）中

1. 存在setup.py

通过查阅资料了解，它是一个打包工具或者脚本

能够同时打包需要的库，自己编写的库，以及一些exe文件一同

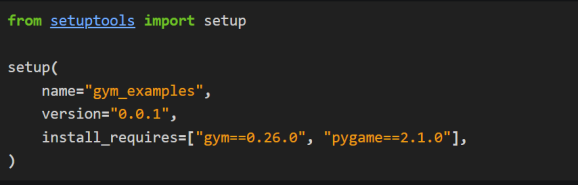
通过 pip setup.py install

这样就能减少打包的多步骤复杂性，直接通过setup.py完成

那么很多github上的库，在git到本地后，就可以直接在该路径下

pip setup.py install

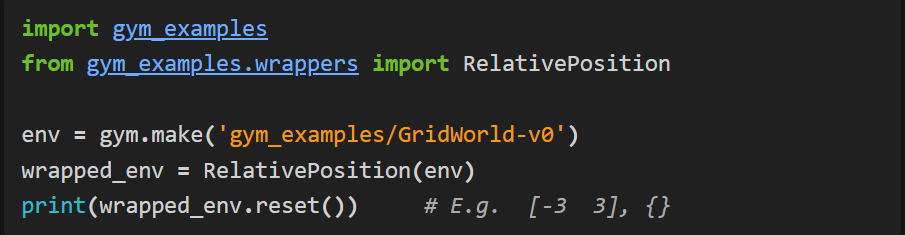
从而减少了自己寻找安装环境的复杂步骤



这里只放出setup的示例文件内容图片，如需了解如何打包，百度

1. 存在装饰器文件夹 /wrappers/

目前官方文档尚未查阅到 装饰器的tutorial，但给出了使用wrapper的示例



2. 具体的环境文件（例如GameEnv.py）编写

根据OpenAI Gym官方文档的tutorial，书写一个环境子类 subclassing gym.env

名为GridWorldEnv，文件具体位置为gym-examples/gym\_examples/envs/grid\_world.py

也就是某个环境类下的具体游戏环境

1. 游戏环境说明

该环境是二维的，有一个终点，以及随机初始化的agent位置

在每一个时间点，agent能够移动上下左右四个方位

Agent的目标为每次随机初始化都要抵达终点

具体分类解释：

1. Observations provide the location of the target and agent.

该环境的观察会返回目标位置与agent位置

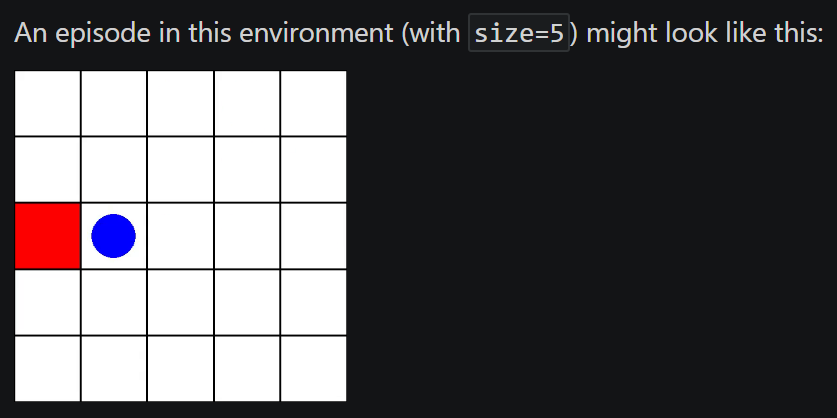
1. There are 4 actions in our environment, corresponding to the movements “right”, “up”, “left”, and “down”. 在该环境中有4个动作可执行，分别为上下左右
2. A done signal is issued as soon as the agent has navigated to the grid cell where the target is located.

如果Agent抵达了终点，那么将会有一个Done信号被提出

1. Rewards are binary and sparse, meaning that the immediate reward is always zero, unless the agent has reached the target, then it is 1.

Reward是离散的，只有两个值，终点是1，其余位置则全部都是0

该reward设置有待考虑，可以尝试将其他位置设置为-0.1，逼迫agent快速找到终点



where the blue dot is the agent and the red square represents the target.

1. 具体环境创建编写

Tutorial中提到需要对metadata 元数据进行定义，其作用是规定render\_mode(渲染/可视化模式)的选择

metadata = {"render\_modes": ["human", "rgb\_array"], "render\_fps": 4}

并且有提到，官方环境render\_mode一般有三种选择(e.g. "human", "rgb\_array", "ansi")

以上则是给出了两种选择，后面的render\_fps属性提及是 以4fps渲染

在类的\_\_init\_\_初始化方法中

需要表明游戏表格的size大小

同时必须要定义self.observation\_space 与 self.action\_space

还需要设定关于render渲染或者说可视化的variables

根据1. 环境介绍中提到的要求，observation\_space有两个要点，agent位置与target位置

例如两个关键字agent与target的dict {"agent": array([1, 0]), "target": array([0, 3])}

同时action\_space 需要包含四个动作，Tutorial中说使用一个数据结构 Discrete(4)

具体内容为 (“right”, “up”, “left”, “down”)

目前环境编写完成，且测试成功，最终与OwnEnvs同一目录的 setup.py 打包文件没有测试