# 寻宝冒险

## 任务

尽可能多地躲避箭头，并快速到达终点的宝藏处。

## 环境

经过网络上浏览相关知识、询问学长，我发现环境文件中主要由dispenser确定发射箭的装置，每个装置的z坐标即为箭的z坐标，而x坐标表示箭飞到的位置。环境交互主要是通过world.py中的函数进行。其中比较重要的是跟我们判断当前状态的环境相关的函数，第一个是get\_curr\_life判断是否当前agent已经死亡，第二个是get\_curr\_pos返回agent当前的位置，包括x、y、z坐标等；第三个是get\_arrow\_pos，可以获取到当前的arrow的z坐标和x坐标。在训练过程中，每次agent的生命值减为0后，即结束该mission。

参考代码给定两个环境文件，world通过调用xml中指定的文件进行加载环境。有两种箭头环境，第一个是arrow1a文件夹中的环境，第二个是arrow6文件夹中的环境，它们分别拥有1个和3个dispenser，也就是说同时可能出现的箭头个数分别是1个和3个。

## Dodger类的属性及方法

属性包括了：

1.游戏中实际的agent

2.学习率alpha

3.折扣因子gamma

4.TD的长度1

5.随机策略概率epsilon

6.q\_table

7.起始位置start\_pos

8.分配器位置dispenser\_pos

9.Agent当前生命 life

10.作出行动后的暂停时间sleep\_time

方法主要有：

1.get\_curr\_state：传入观察字典obs，输出当前的状态state。State作为一个列表，依次包含agent当前的z坐标，箭头按各自所在的z坐标升序排序的x坐标。

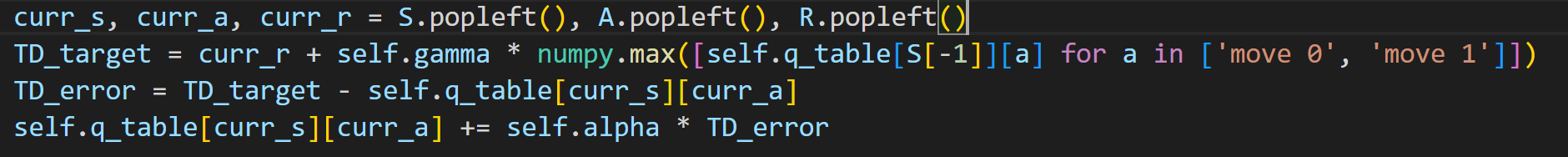
2.get\_action：传入当前的状态curr\_state和可能执行的动作列表possible\_actions，输出按照epsilon-greedy策略得到的action。该方法中需要注意，由于需要用到q表来作出决策，但可能当前的状态并未出现在q表中，这种情况下需要创建q表中这个状态的值。

3.get\_reward：传入观察字典obs和上一次执行的action，得到回报值reward，并判断是否到达终点或是否被箭射中的标志reward。按照提示，我们可以设定reward=agent的距离（\*成功或失败的奖励值）\*成功躲避了箭头的奖励扩大比例\*停下来1次的奖励缩减比例。判断被箭射中的标志有两种办法，第一是判断是否agent的生命值有所减少，第二是判断是否当前的x坐标偏离了初始的x坐标（因为正常情况下沿着z轴方向行走不会变化x坐标）。判断是否到达终点的办法有两种，第一种是通过z坐标是否到达终点的格子，第二种是判断agent能够看到的当前第一个物体是否为宝藏’chest’。判断是否停下的方法就是判断上一次行动是否是’move 0’。判断是否躲避了箭头的方法是判断agent当前所在的z坐标是否就是发射箭的z坐标。

4.update\_q\_table，传入当前已经进行到了episode的轮数、状态队列、动作队列、回报值队列等。这个部分就是Q-learning的核心部分。我们主要通过公式

来实现。首先分别从状态、动作和回报的队列里弹出队首的值，表示当前的状态、动作和回报值，而下一状态则索引状态队列的末尾元素S[-1]。

由此更新q-table：



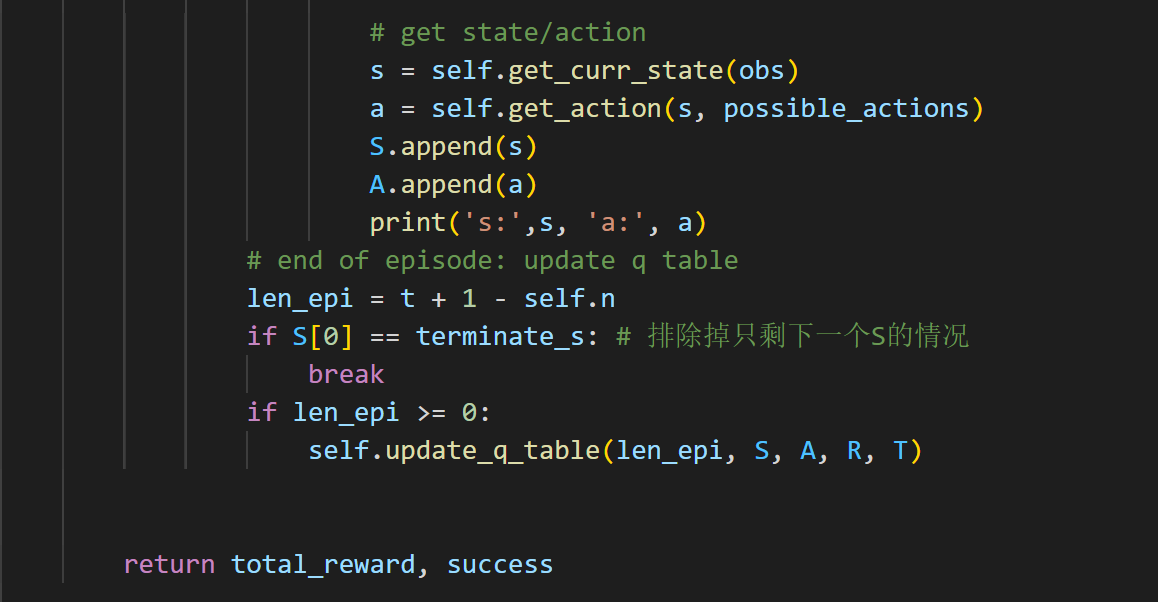
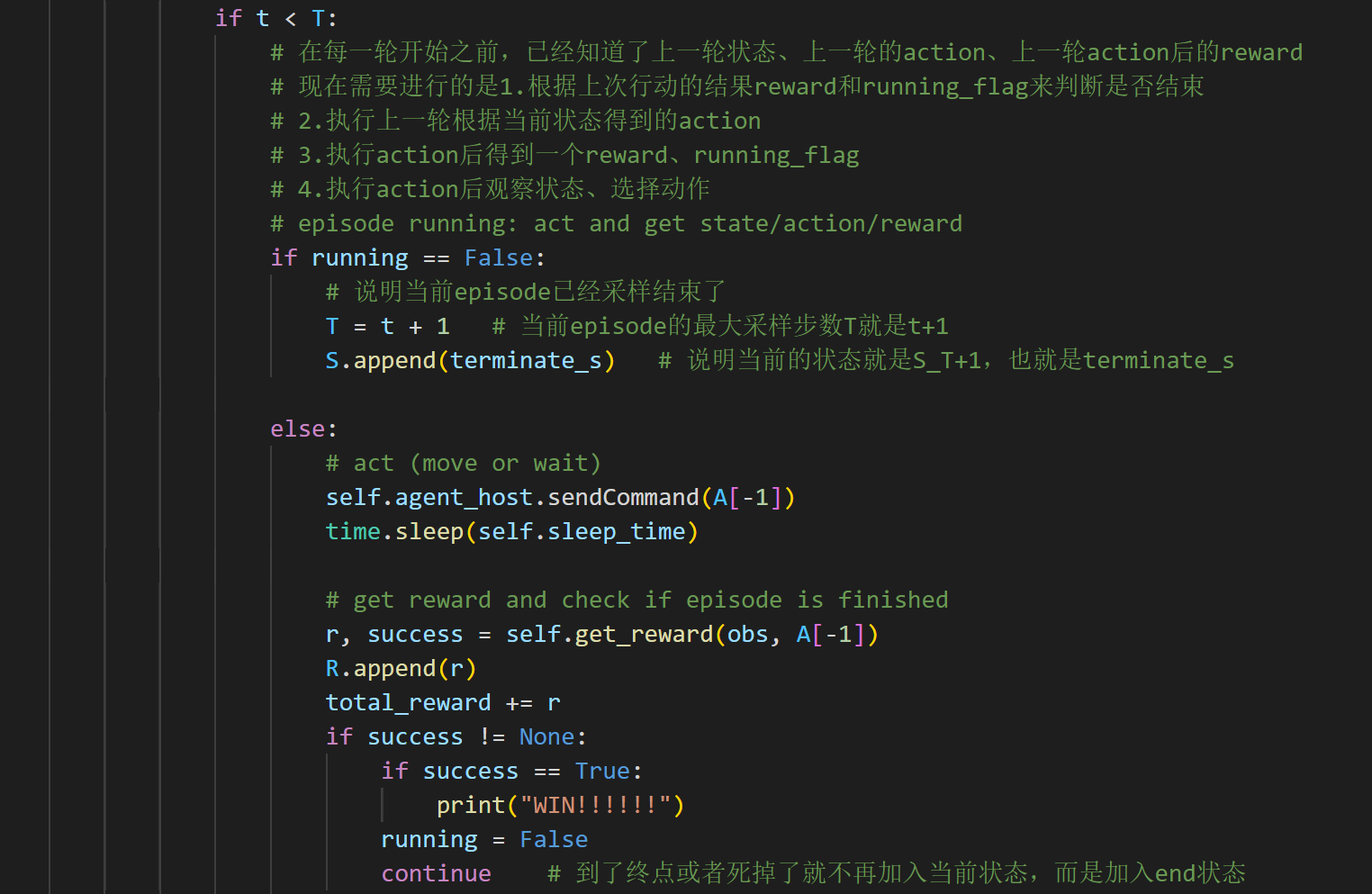
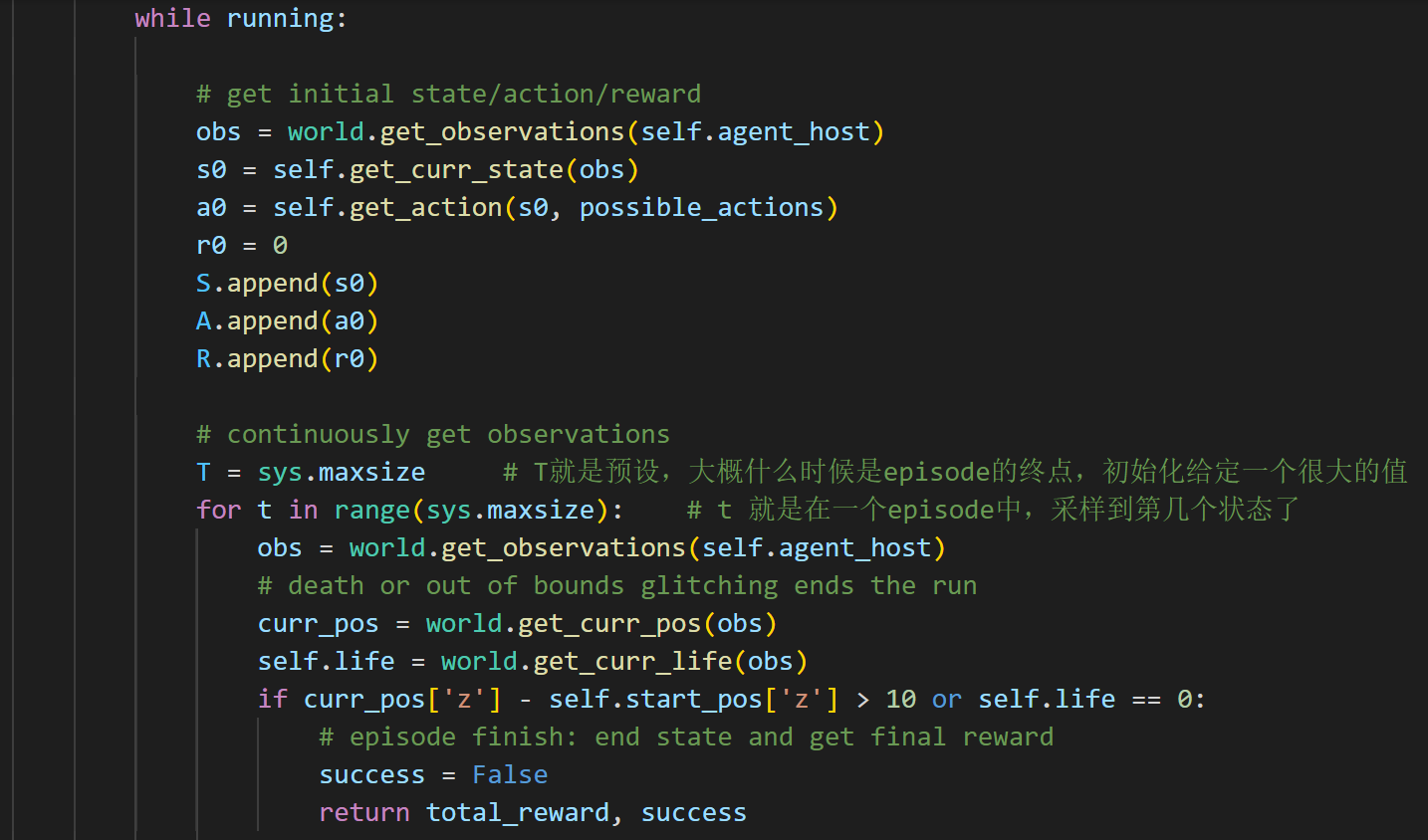
## Q-learning整体控制流（dodger.py）

第一，初始化状态、动作和回报的队列，实例化为collections.deque()类。

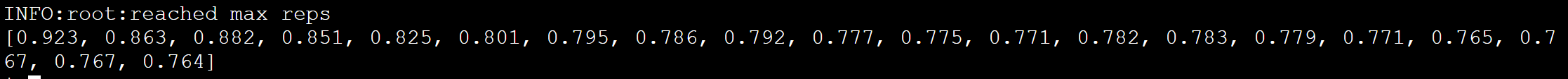
第二，初始化possible\_actions、总回报值、任务成功标志success。

第三，设定running表示一直运行，直到判断success为True/False时停下。再初始化当前状态、动作和回报。

第四，开始一边采样一边学习。设定当前采样步数为t。每一步采样中，首先，获得当前的观察情况；其次，判断当前生命值为0或者行走的路程越界了；第三，执行前一步按照epsilon-greedy策略得到的动作，并获取执行动作后的回报、成功标志；第四，判断是否到达终点或者被箭射中，如果不属于这两种情况就获取当前状态和动作。最后，更新q表。



Q-learning结果

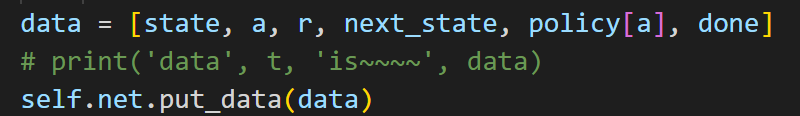


## PPO算法：

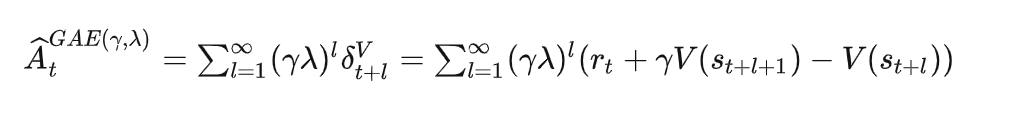
在PPO文件中定义了ppo模型的类

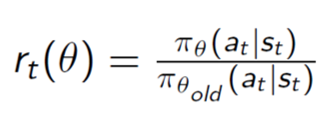
首先初始化了折扣因子gamma，学习率，lambda，和剪枝的值epsilon\_clip。并初始化data列表，装每一个状态、动作和回报对。初始化第一层全连接层用于提取特征，初始化第二层全连接层用于分别输出pi和输出v。

每个mission运行的时候，首先按照初始化的policy随机选择动作，然后得到新的状态和回报，并将状态、动作、回报、动作对应的概率、完成标志分别加入data列表中。

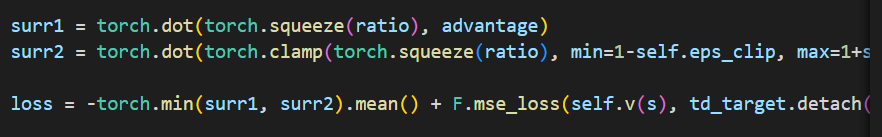


每一步后，对模型进行训练。训练时，首先通过GAE算法获得advantage

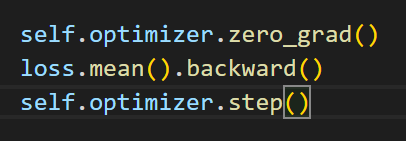


然后计算r\_t(θ)，即，表示更新策略的幅度大小。

之后，再计算epsilon剪枝后的actor loss以及值函数预测的critic loss。

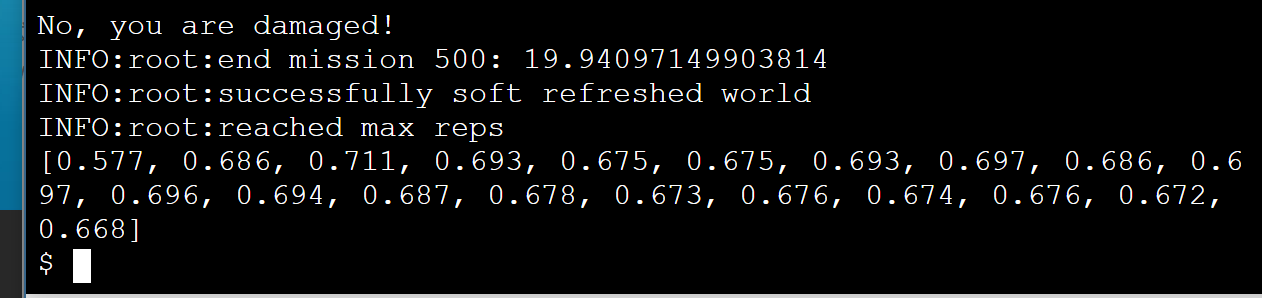


最后，反向传播计算loss，并更新policy参数θ和价值评估的权重参数w。



PPO结果：

随着轮数增加，每25轮结束后的前序所有轮数的正确率为下表



可见，最后500后训练收敛的正确率仅有66.8%