

SHANGHAI JIAOTONG UNIVERSITY



强化学习课程设计 – Project3

Model-free Control

姓名：薛春宇

学号：518021910698

完成时间：2021/4/9

1 实验目的

分别实现Sarsa算法 (on-policy learning) 和Q-Learning算法 (off-policy learning) 来解决Model-free Control中无折扣因子的Cliff Walking问题，并通过调整不同的值来直观地感受Sarsa与Q-Learning在路径决策上的不同。

2 实验准备

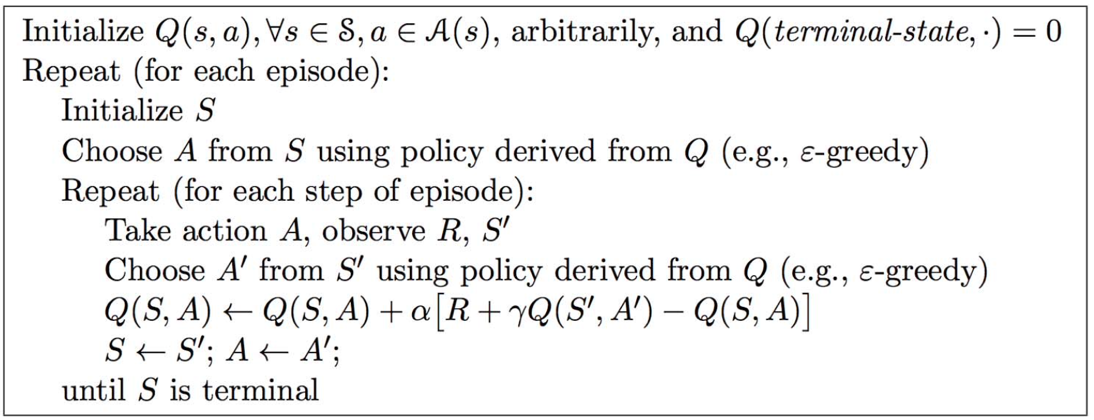
在本节中，我们将分别介绍Sarsa算法和Q-Learning算法的相关内容，以为接下来的代码实现做准备。

2.1 Sarsa算法

Sarsa算法是一种On-policy的Model-free控制算法，其主要思想是将探索和决策两个过程合并在一起，使用一个更新策略 来不断进行迭代。  
 具体来说，Sarsa算法首先随机初始化一个state（在本例中的Cliff Walking问题中，为除起、终点以及Cliff以外的其他state），在当前state下利用选择一个即将采用的action，然后开始迭代：

1. **说到做到地**采取当前action，观察下一个state’，再利用寻找下一个action’，且**说到做到地**在下一次迭代中采取该action’
2. 更新Q(state, action)，基于当前的Q(state, action)和即将要采用的Q(state’，action’)
3. 移动：state = state’，action = action’
4. 重复上述迭代过程直到到达terminal state

整个policy iteration的伪代码如下所示：



我们的python代码也是基本按照上述伪代码的思想进行实现的。

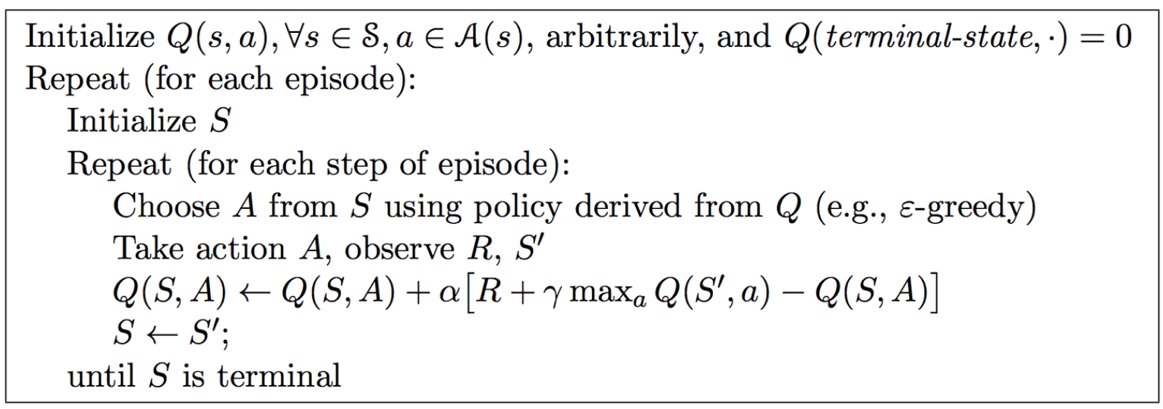
2.2 Q-Learning算法

Q-Learning算法是一种Off-policy的Model-free控制算法，其主要思想是将探索和决策两个过程分开实现，使用更新策略 来不断进行探索，使用贪心策略来进行策略改进，即决策，以各执行上述两个过程一次为一次迭代。

具体来说，Sarsa算法首先随机初始化一个state（在本例中的Cliff Walking问题中，为除起、终点以及Cliff以外的其他state），然后开始迭代：

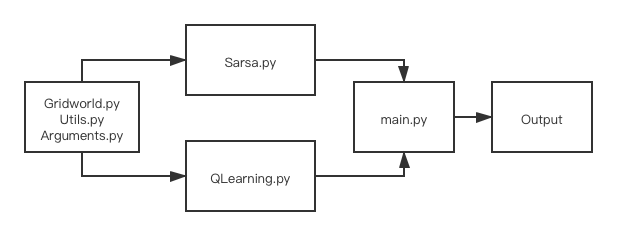
1. 当前state下利用选择一个即将采用的action
2. **说到做到地**采取这个action，观察下一个state’，再利用**普通的基于Q的贪心策略**寻找下一个action’，且**仅假设**采取该action’（实际采取哪个action’需要在每次迭代的第一步决定）
3. 更新Q(state, action)，基于当前的Q(state, action)和即将要采用的Q(state’，action’)
4. 移动：state = state’
5. 重复上述迭代过程直到到达terminal state

Q-Learning算法的伪代码如下所示：



3 实验内容

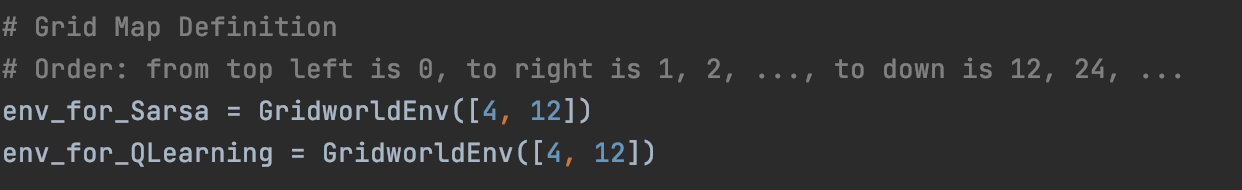
本次实验的代码及逻辑结构如下所示：



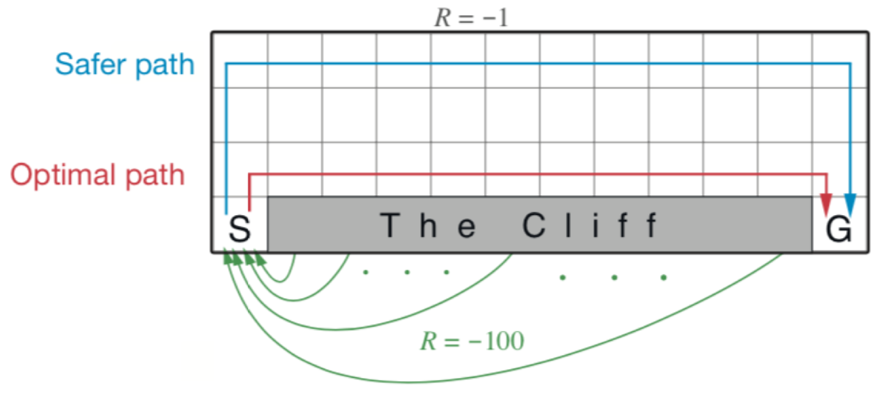
接下来，我们将对几个主要文件进行逐一分析，并分析两类控制算法的差异。相关代码位于./code目录下。

3.1 Gridworld.py

该模块的实现参考了Sutton版RL提供的Gridworld类，实现了一个可实例化的格子世界，可以通过如下指令进行实例化：

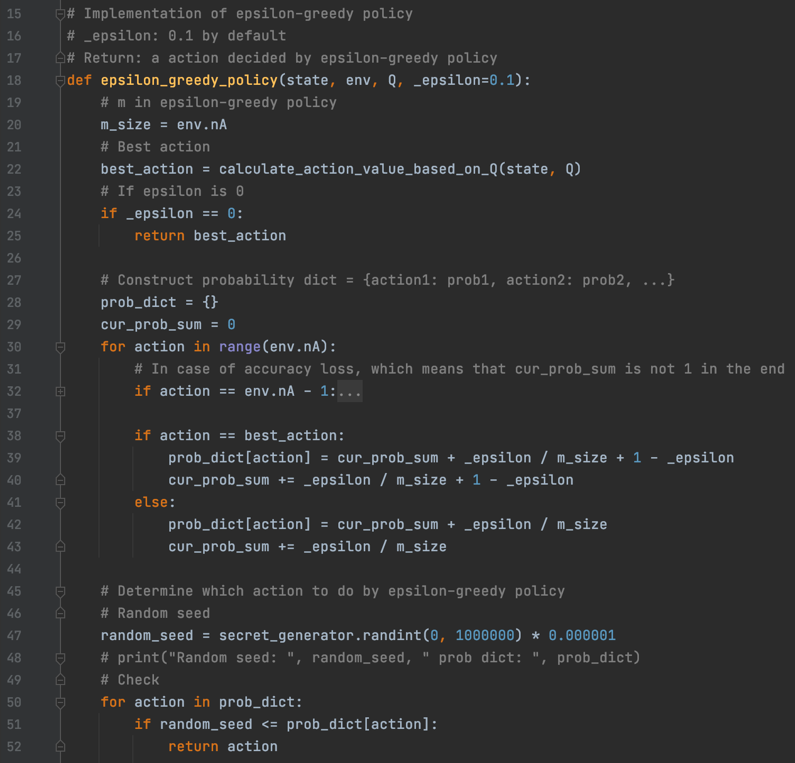


需要注意的是，本项目中的Terminal State、Action Tpye、Reward和概率转移矩阵P均需要在Gridworld.py中声明。本项目中的Gridworld需要被实例化为如下结构（其中36号grid为起点S，47号grid为终点G，37～46号grid为Cliff state）：



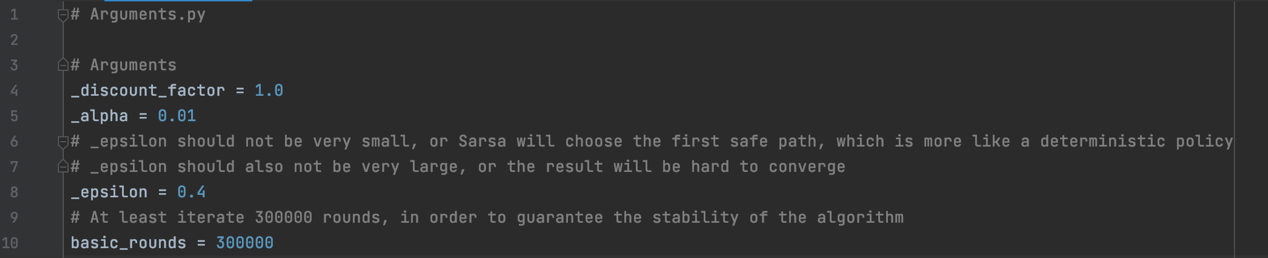
3.2 Utils.py & Arguments.py

Utils.py主要实现了 概率选择策略，通过调用random\_seed，将所有的action作为key存在一个字典中，value对应它们的概率大小，注意这些概率是对0~1区间进行划分的子区间宽度：



其中，calculate\_action\_value\_based\_on\_Q()为自行实现的获取当前state基于贪心的最佳action的函数。

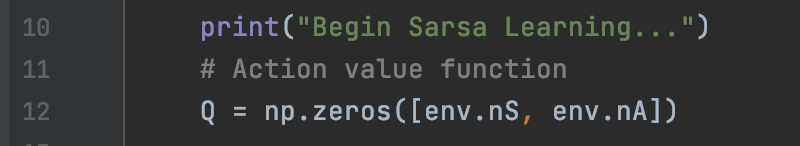
此外，Arguments.py主要定义了一些模型相关的参数，包括和迭代轮数：

**

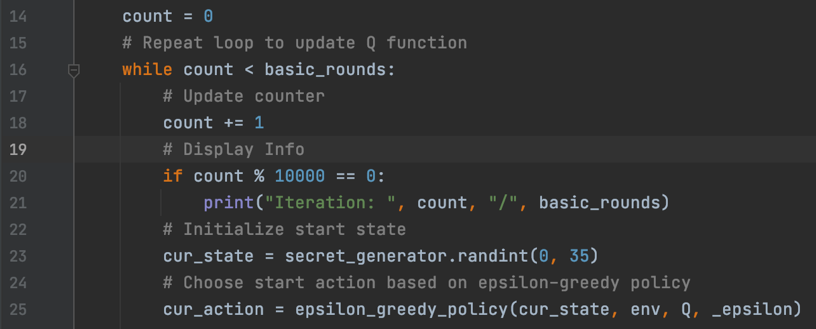
3.3 Sarsa.py

该模块的实现基本遵循了2.1节中给出的伪代码的思想，并将整体内容设置成一个可调用的函数Sarsa\_learning()，供用户在main.py中调用。

在函数的开始部分，我们根据已知的参数大小（env.nS为Gridworld的全部状态数，env.nA为Gridworld的全部行为数）来创建行为值函数Q的容器：



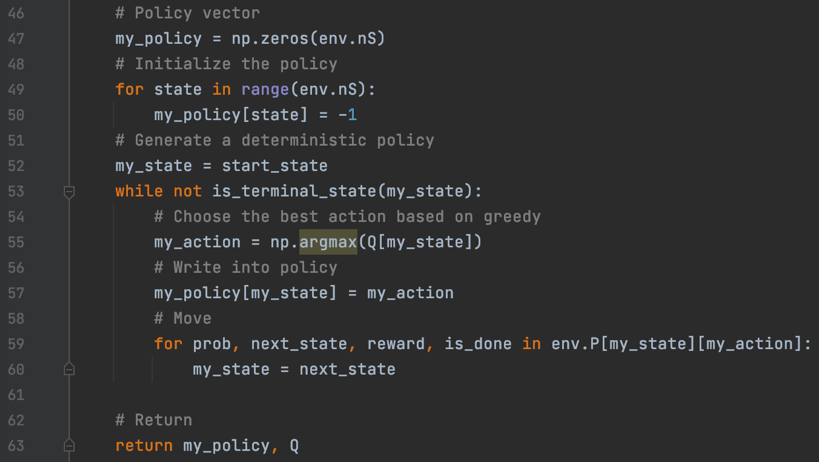
按照2.1节中伪代码的思想，我们用一个while循环来表示不断在Gridworld中探索,并首先随机初始化一个合适的state，根据策略选择一个初始action。



接下来，我们开始探索以及策略改进，通过不停调用策略来更新行为值函数Q以及探索下一个状态和行为，并移动agent（该while循环包含在上面的while内部）：

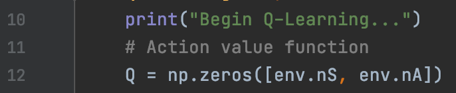


在达到最大迭代次数后，跳出最外层的while循环，开始构造确定性策略。注意，在构造确定性策略时，我们直接使用贪心算法进行最优路径的选择：

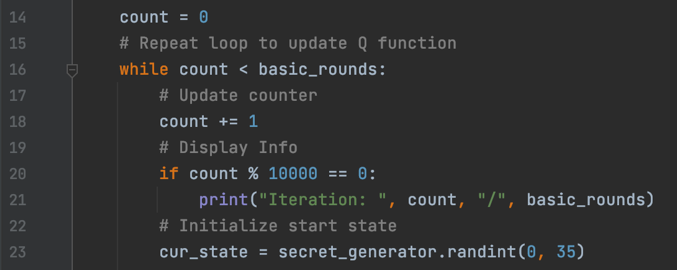


3.4 QLearning.py

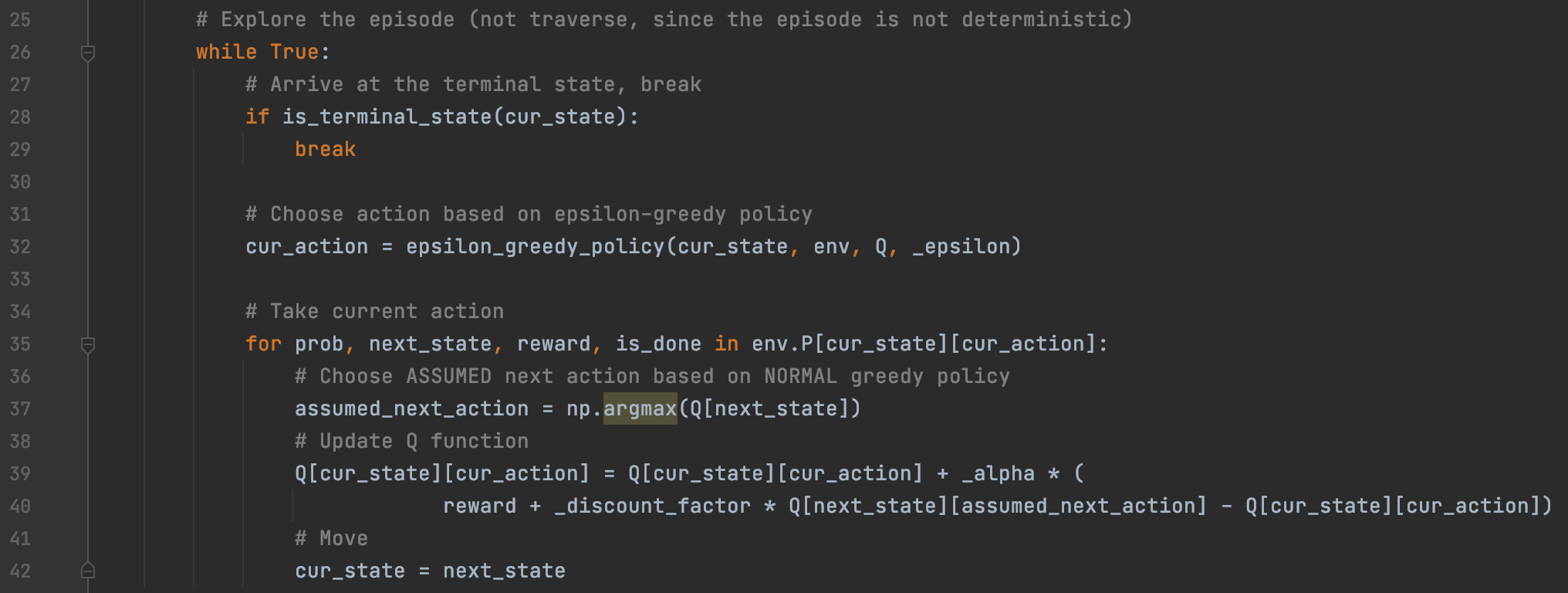
该模块的实现基本遵循了2.2节中给出的伪代码的思想，并将整体内容设置成一个可调用的函数Q\_learning()，供用户在main.py中调用。在函数的开始部分，我们根据已知的参数大小（env.nS为Gridworld的全部状态数，env.nA为Gridworld的全部行为数）来创建行为值函数Q的容器：



按照2.1节中伪代码的思想，我们用一个while循环来表示不断在Gridworld中探索,并首先随机初始化一个合适的state。

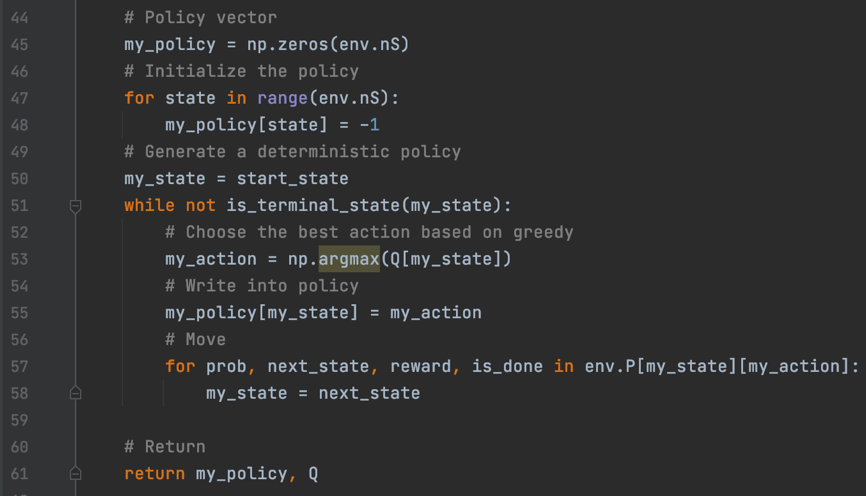


接下来，我们开始探索以及策略改进，首先调用策略选择当前state下应该采取的action，再用贪心算法**假设**下一个state’应该选择的action’，并更新Q及状态移动：



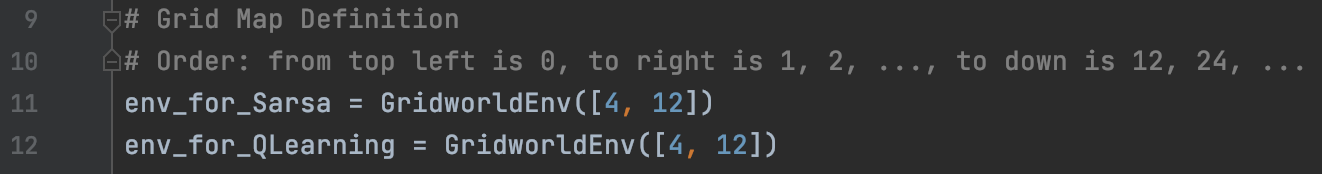
注意，上述的while循环包含在前面的while循环内部。

在达到最大迭代次数后，跳出最外层的while循环，开始构造确定性策略。注意，在构造确定性策略时，我们直接使用贪心算法进行最优路径的选择：

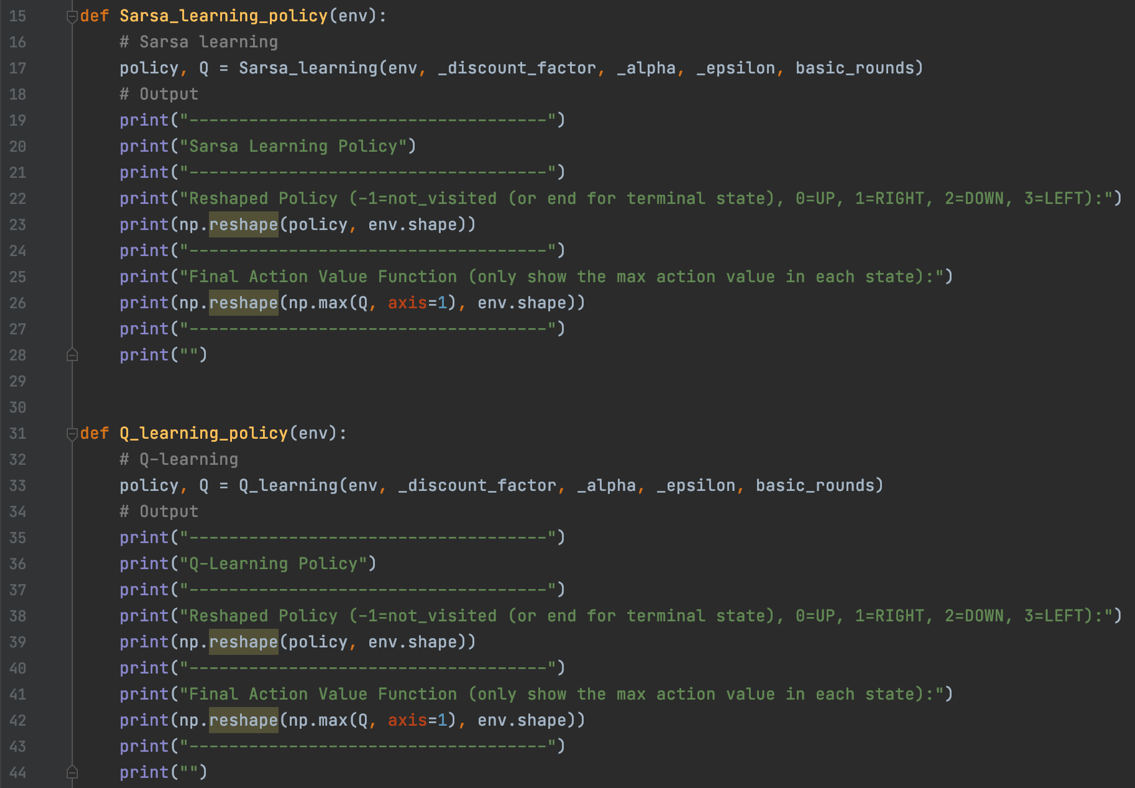


3.5 main.py

主函数中，我们首先进行了Cliff gridworld的初始化定义：



然后，分别设置两个封装函数，对上述两种算法的调用进行封装，以及输出的格式化：

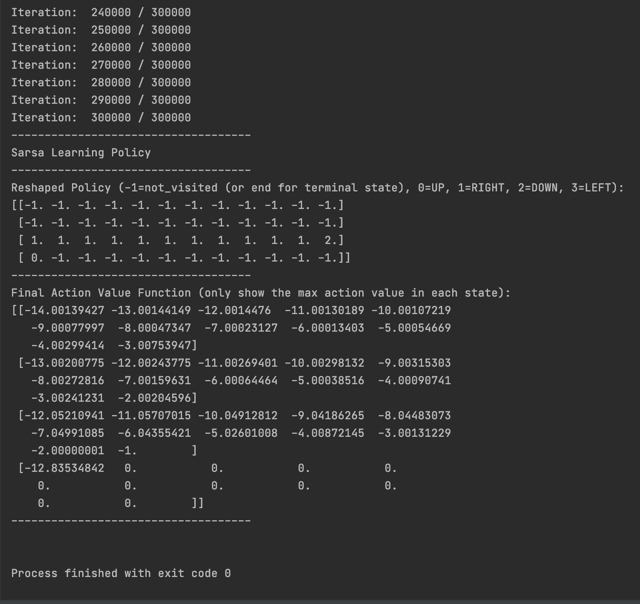
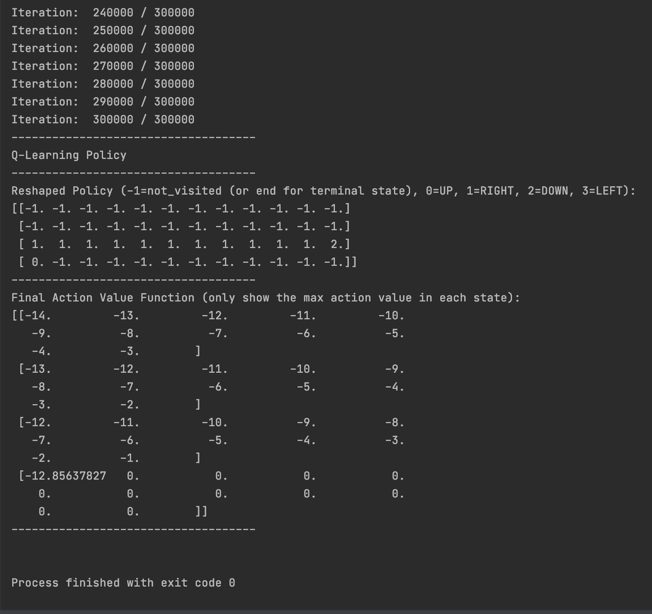


最后，我们在main()函数中调用上述两个封装函数，即可完成算法的调用及结果输出。

4 实验结果

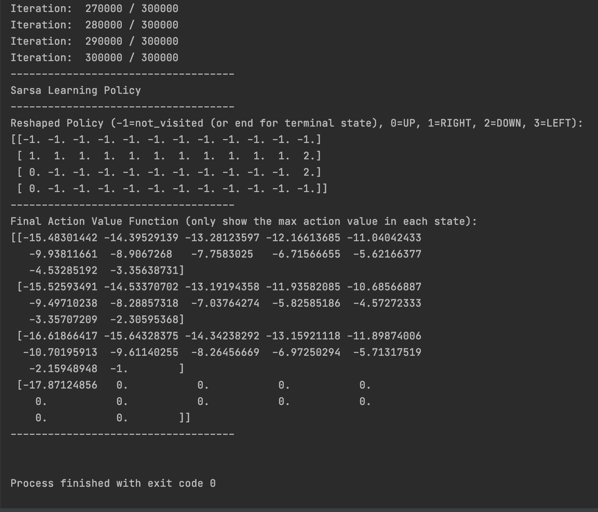
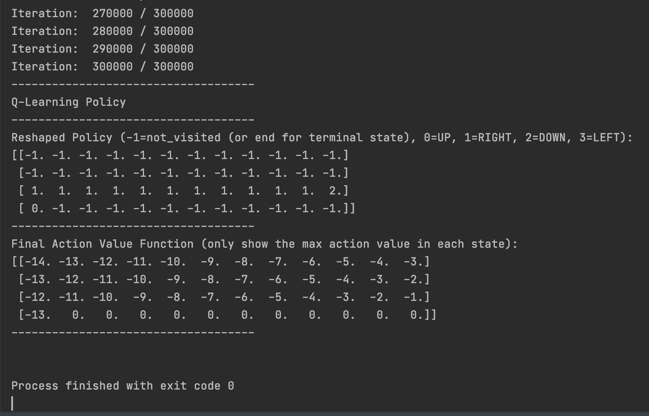
本节中，我们将基于不同值，分别使用Sarsa算法和Q-Learning算法进行对比实验，讨论两种算法在路径决策上的差异性。

首先，我们将值设置为0.001，运行算法：

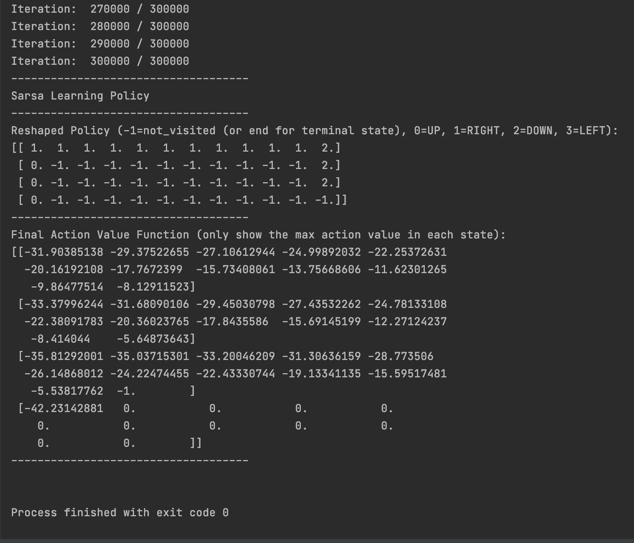
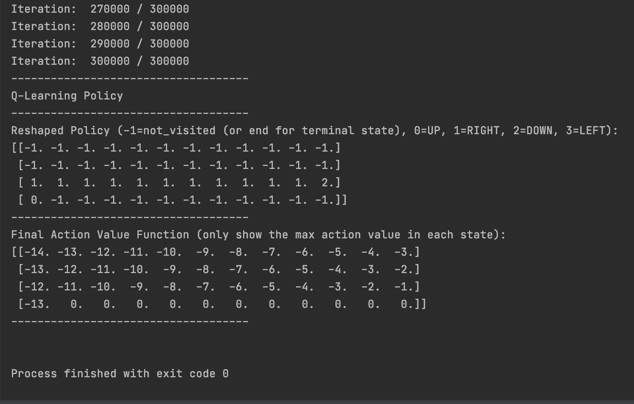
Sarsa算法() Q-Learning算法()

可以看到，两种算法均选择了cost相对最小的最优路径。接着，我们将值设置为0.1，运行算法：

Sarsa算法() Q-Learning算法()

当值取到0.1时，Sarsa算法选择了一个相对更安全，但并非cost最优的路径；而Q-Learning算法仍选择cost最优的路径。最后，我们将值设置为0.4，运行算法：

Sarsa算法() Q-Learning算法()

当值取到0.4时，Sarsa算法选择了一个所有路径选择中最安全的路径，但并非cost最优的路径；而Q-Learning算法仍选择cost最优的路径。

结合上述实验结果，分别分析Sarsa和Q-Learning算法的特性可以发现：

1. Sarsa是说到做到型, 所以我们也叫他On-policy, 在线学习, 学着自己在做的事情
2. Q-Learning 是说到但并不一定做到, 所以它也叫作Off-policy, 离线学习

而因为有了贪心选择max\_Q，Q-Learning也是一个特别勇敢的算法。因为Q-Learning 机器人永远都会选择最近的一条通往成功的道路, 不管这条路会有多危险。而Sarsa则是相当保守, 他会选择离危险远远的, 拿到宝藏是次要的, 保住自己的小命才是王道。这就是使用Sarsa方法和使用Q-Learning方法的不同之处。

5 实验心得

在本次实验的过程中，我首先在课堂授课内容的基础上，系统性地学习了Sarsa算法和Q-Learning算法的相关知识，并掌握了其在代码层面上的实现方法。在实现的过程中，由于从伪代码到python代码的差别还是比较大的，我遇到了一些困难，其中就包括了在实现策略选择的时候，由于当值取的足够小时，python会出现一些精度丢失，导致算法出现bug。在发现这个bug之后，我进行了一定的代码优化，解决了这个bug。

整个过程约花费半天时间，在本次实验中，我不仅了解了两种Model-free控制方案的基本知识，掌握了从伪代码到可运行代码的复现方法，更是提高了自身发现bug，解决bug的能力。希望在接下来的实验中也能收获满满！