课程作业 5：强化学习（Reinforcement Learning）

任务概述

在本次课程作业中，你将实现价值迭代和 Q 学习，首先在格子世界中测试你所实现的智能体，然后将它们应用于仿真机器人控制器（Crawler）和吃豆人。

首先，可以尝试在手动控制模式下运行格子世界，该模式使用箭头键控制：

```
python gridworld.py -m
```

正如在课上所介绍的，格子世界包含两个出口，蓝点代表智能体，当按下向上键时，智能体实际上仅有 80%的概率向北移动。

运行以下命令可以获得完整的仿真选项列表：

```
python gridworld.py -h
```

默认智能体随机移动：

```
python gridworld.py -g MazeGrid
```

可以看到智能体在格子上随机移动，直到碰巧到达一个出口。

作业的程序包位于 QQ 群文件：课程作业\reinforcement.zip。

所有你自己的算法实现都位于 `valueIterationAgents.py`、`qlearningAgents.py` 和 `analysis.py` 中相应任务的类和函数下面。

作业内容

任务 1：价值迭代（Value Iteration）

根据价值迭代状态更新方程：

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

在 `valueIterationAgents.py` 中的 `ValueIterationAgent` 类中实现一个价值迭代智能体。 这里的价值迭代智能体是一个离线规划器，而不是强化学习智能体，因此相关的训练选项是它在初始规划阶段应该运行的价值迭代的迭代次数（选项 `-i`）。`ValueIterationAgent` 在构造时采用一个 MDP，并在构造函数返回之前会运行指定迭代次数的价值迭代。

价值迭代计算最优值的 k 步估计 V_k。 除了 `runValueIteration` 之外，还使用 V_k 为 `ValueIterationAgent` 实现以下方法：

- `computeActionFromValues(state)`根据 `self.values` 给出的价值函数计算最佳动作；
- `computeQValueFromValues(state, action)`返回由 `self.values` 给出的价值函

数给出的`(state, action)`对的 Q 值。

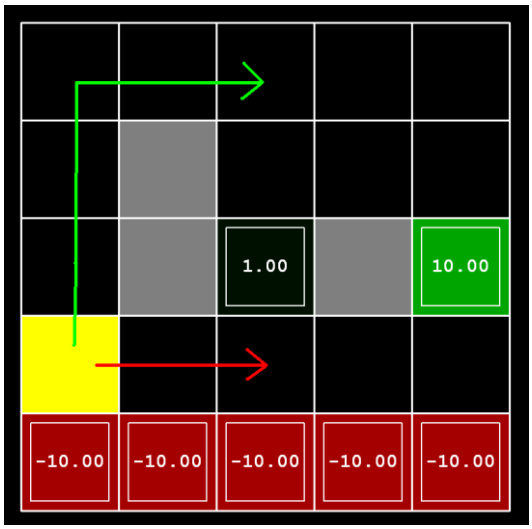
这些数值会显示在 GUI 中：价值是正方形中的数字，Q 值是正方形四分之一中的数字，策略是每个正方形中向外的箭头。

为了测试你的代码，运行：

```
python autograder.py -q q1
```

任务 2：策略（Policies）

考虑如下的 `DiscountGrid` 布局：该网格有两个具有正收益的终止状态（中间行），一个收益为+1 的近距离出口和一个收益为+10 的远距离出口。网格的底部由具有负收益的终止状态组成（红色方块）； 这个“悬崖”区域的每个状态的收益为-10。 起始状态是黄色方块。 我们有两种可能的路径：（1）靠近网格底部的路径。这些路径较短，但有可能获得较大的负收益，如下图中的红色箭头所示。(2)沿着网格顶部边缘行进的路径。 这些路径更长，但产生较大负收益的可能性较小，如下图中的绿色箭头表示。



该任务需要在 `analysis.py` 中为 MDP 对折扣、噪声和生存奖励参数进行设置，以生成几种不同类型的最优策略。每个部分设置的参数值应该使得：如果智能体遵循其最佳策略而不受任何噪音影响，它将表现出给定的行为。如果任何参数设置都未实现特定行为，则通过返回字符串`'NOT POSSIBLE'`来断言该策略是不可能的。

要查看一组设置所产生的行为，运行以下命令来查看 GUI：

```
python gridworld.py -g DiscountGrid -a value --discount [YOUR_DISCOUNT]
--noise [YOUR_NOISE] --livingReward [YOUR_LIVING_REWARD]
```

为了检查你的答案，运行：

```
python autograder.py -q q2
```

任务 3：Q 学习（Q-Learning）

价值迭代智能体实际上并不从经验中学习，而是根据其 MDP 模型制计算完整的策略。与环境交互时，它简单的遵循预先计算的策略。

该任务需要在 `qlearningAgents.py` 的 `QLearningAgent` 类中实现一个 Q 学习智能体，通过 `update(state, action, nextState, reward)`方法从与环境交互中的反复试错来学习。可以使用选项`'-a q'`选择 Q 学习智能体。需要实现的方法包括：`update`, `computeValueFromQValues`, `getQValue` 和 `computeActionFromQValues`。

为了测试你的实现，运行：

```
python autograder.py -q q3
```

任务 4：epsilon 贪心策略（Epsilon Greedy）

通过在 `getAction` 中实现 epsilon 贪心动作选择来完成 Q 学习智能体，即，以 epsilon 概率选择随机动作，否则选择其当前的最佳 Q 值动作。需要注意的是，选择随机动作可能会导致选择最佳动作。也就是说，不应选择随机的次优操作，而应选择任何随机的合法动作。

可以通过调用 `random.choice` 函数从列表中以均匀概率分布随机选择一个元素。可以使用 `util.flipCoin(p)`模拟成功概率为 `p` 的二值变量，其返回 `True` 的概率为 `p`，返回 `False` 的概率为 `1-p`。

实现 `getAction` 方法后，可以运行以下命令来观察格子世界中智能体的以下行为（epsilon = 0.3）：

```
python gridworld.py -a q -k 100
```

最终的 Q 值应该类似于价值迭代智能体的 Q 值，特别是沿着经常通过的路径。尽管如此，由于随机动作和初始学习阶段，平均收益将低于 Q 值预测。

也可以运行使用不同 epsilon 值的仿真来观察智能体的行为是否符合预期：

```
python gridworld.py -a q -k 100 --noise 0.0 -e 0.1
python gridworld.py -a q -k 100 --noise 0.0 -e 0.9
```

为了测试你的实现，运行：

```
python autograder.py -q q4
```

现在应该可以运行 Q 学习爬行机器人（Crawler）：

```
python crawler.py
```

尝试各种学习参数来看看它们如何影响智能体的策略和行动。请注意，步长延迟是仿真参数，而学习率和 epsilon 是学习算法的参数，折扣因子是环境的属性。

任务 5：Q 学习与吃豆人（Q-Learning and Pacman）

吃豆人将分两个阶段进行游戏。第一阶段为训练阶段，吃豆人关于位置和行动的值进行学习。 因为即使对于非常小的网格来说，学习准确的 Q 值也需要花费很长时间，所以吃豆人的训练游戏默认以安静模式运行，没有 GUI（或控制台）显示。吃豆人的训练完

成后将进入测试模式。测试时，吃豆人的 `self.epsilon` 和 `self.alpha` 将被设为 0.0，从而停止 Q 学习并禁用探索，以便让吃豆人能够利用所学到的策略。默认情况下，游戏测试会在 GUI 中显示。通过执行如下命令，可以在非常小的网格上运行 Q 学习吃豆人：

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

需要注意的是，`PacmanQAgent` 已被所实现的 `QLearningAgent` 定义。`PacmanQAgent` 的唯一不同之处在于它具有对吃豆人问题更有效的默认学习参数（`epsilon=0.05, alpha=0.2, gamma=0.8`）。 **正常情况下，所实现的智能体应该在至少 80%的情况下获胜。**autograder 将在 2000 轮训练后运行 100 轮测试。

为了测试你的实现，运行：

```
python autograder.py -q q5
```

任务 6：近似 Q 学习（Approximate Q-Learning）

该任务需要实现一个近似 Q 学习智能体，用于学习状态特征的权重，其中多个状态可能共享相同的特征。在 `qlearningAgents.py` 的 `ApproximateQAgent` 类中完成代码实现。该类是 `PacmanQAgent` 的一个子类。

默认情况下，`ApproximateQAgent` 使用 `IdentityExtractor` 来将单个特征分配给每个 `(state,action)`对。使用该特征提取器，近似 Q 学习智能体应该与 `PacmanQAgent` 的运行方式是一致的。可以执行以下命令对此进行测试：

```
python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid
```

确保使用 `IdentityExtractor` 特征提取器可以正确运行后，执行以下命令来使用自定义特征提取器运行所实现近似 Q 学习智能体，通过学习应该可以轻松获胜：

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x
50 -n 60 -l mediumGrid
```

对于所实现的 `ApproximateQAgent`，即使是更大的网格也应该不在话下（可能需要花费数分钟来训练）：

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x
50 -n 60 -l mediumClassic
```

如果实现无误，即使只进行 50 轮训练，所实现的近似 Q 学习智能体也应该能够凭借这些简单的特征保证在测试中几乎每轮都能获胜。

为了测试你的实现，运行：

```
python autograder.py -q q6
```

各个任务的更详细描述可以参考：
https://inst.eecs.berkeley.edu/~cs188/sp23/projects/proj4/

作业报告

本次作业需要提交报告和代码。对于以上任务，**报告需分别详细介绍代码的实现并分析实验结果**。使用 QQ 群文件中的报告模版（课程作业\报告模版.doc）撰写实验报告。

作业提交

将作业报告存储为 PDF 文件，用学号命名，例如 221900001.pdf，并与相应的源码打包为学号命名的.zip 文件，例如 221900001.zip。

上传到百度网盘：
https://pan.baidu.com/disk/main#/transfer/send?surl=AAIAAAAAABGvSg
注意：与之前的课程作业是不同的网址。

提交截止日期：12 月 28 日 23:59:59

学术诚信

允许同学之间的相互讨论，但是**署你名字的工作必须由你自己独立完成**。

如果发现作业之间高度相似将被判定为互相抄袭行为，**抄袭和被抄袭双方的成绩都将被取消**。

应项目开发者的要求，**严禁将作业答案发布在网上**。