Project 1

【Adapted from CS188】

实现值迭代和Q-learning，并将其运用到Gridworld与吃豆人上。

自动评测

- 运行以下命令可以针对所有问题进行自动评测。

`python autograder.py`

`-q` 命令可以指定测试某一问题：

`python autograder.py -q q2`

`-t` 命令可以指定具体的测试输入：

`python autograder.py -t test_cases/q2/1-bridge-grid`

代码编辑

- 需要完成的代码：
  - **qlearningAgents.py**
  - **analysis.py**
- 可编辑（见第一小题，不要求完成）的代码
  - **valueIterationAgents.py**
- 推荐浏览的代码：
  - mdp.py
  - learningAgents.py
  - util.py
  - gridworld.py
  - featureExtractors.py

NOTE：

1. 请注意**不要修改其他文件或是自己添加新的文件**，在提交代码时只需要提交**qlearningAgents.py**、**analysis.py**这两个需要编辑的文件。
2. 请**不要修改给定的函数名或者类名**，否则可能会导致autograder评分异常。

PJ的要求与提交方式

- 本次PJ可以1-2人完成
- **截止时间：5.12**
- **提交方式：elearning > 作业**
- 本此PJ分为编程题与计算题，编程题需要补全指定文件的代码，计算题需要给出计算过程或答案由来。
- **提交内容：两个需要编辑的代码文件（见上）、以及一个pdf文件。**
- Q1和Q2为计算题，其中计算部分与问答部分需要在pdf中回答，注意Q1、Q2中的粗体字。其余部分均为编程题。
- pdf报告推荐使用latex生成，选择扫描手写答案亦可，但需保证答案过程清晰。

基本操作

运行以下代码可以使用键盘方向键运行Gridworld：

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

- 蓝点表示agent，“1”、“-1”分别为两个出口

如果多次运行你会发现，**有时候agent并不会按照你给定的方向移动**，这是因为在Gridworld设定下，策略是具有一定**随机性**的。

实际上，Gridworld具有许多可以自定义的选项，使用以下命令查看：

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

例如，使用`-g`命令更改地图：

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

NOTE：

1. Gridworld的特点是，你必须首先进入一个双框方格达成**预备终止状态**，然后执行特殊的“**退出exit”动作**，才能使整个过程真正结束（**TERMINAL_STATE**为真正的终止状态，该状态在GUI中不显示）。
2. 除了图形输出之外，控制台输出给出了agent的每一次状态转换的信息。
3. agent的位置由（x，y）笛卡尔坐标表示，可以通过[x][y]索引，其中“north”是y增加的方向，“east”是x增加的方向。默认情况下，大多数状态转换将获得零奖励。

问题

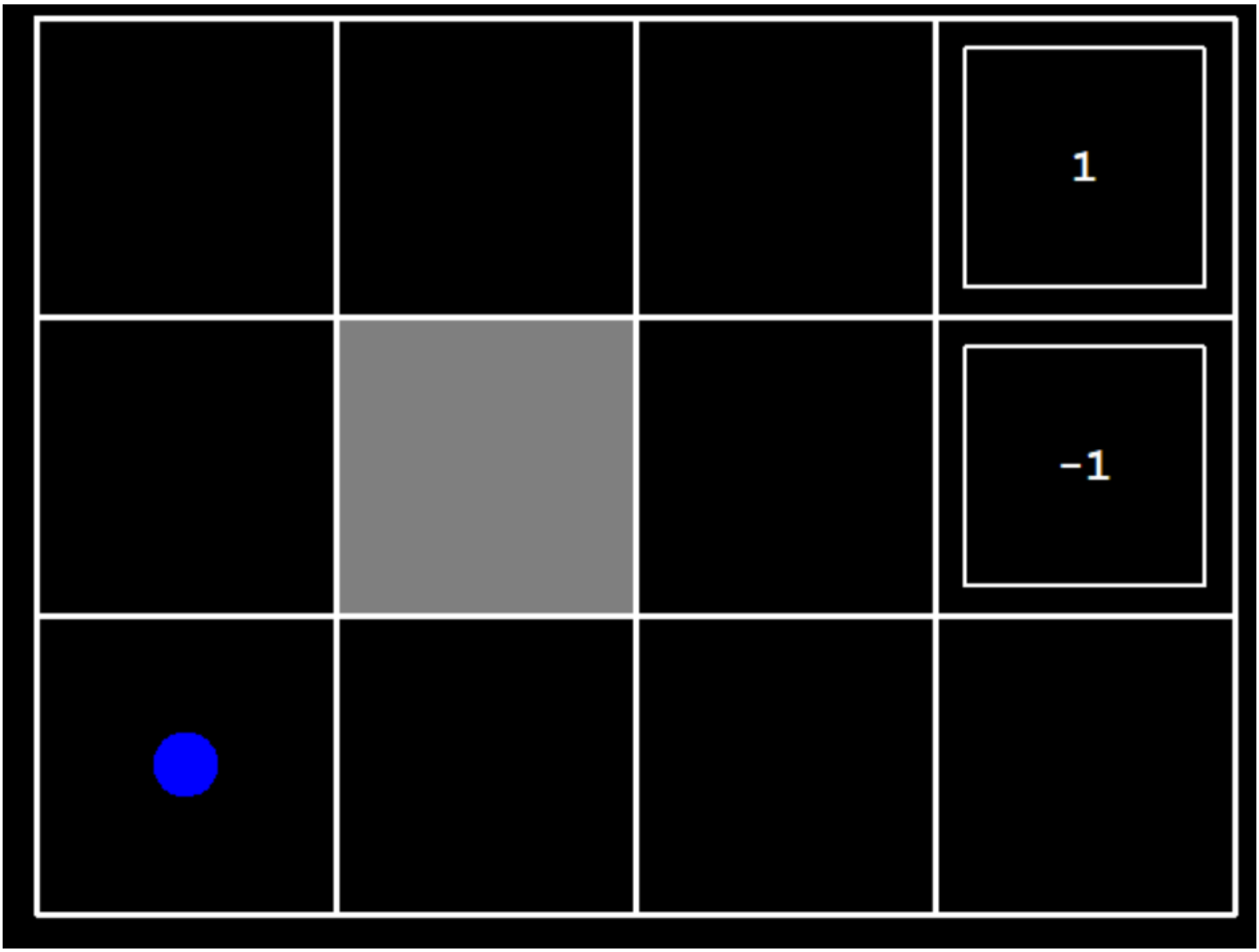
Q1：值迭代（计算题）

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

以上为值迭代的计算方法，**请根据此公式计算Gridworld默认地图中在迭代了两次之后各个状态的值。**

本题请在报告中回答。

噪音设置：所有状态初始化值为0，假设除了exit外的任何动作都有80%向指定方向移动，另外20%的概率向垂直于此方向的两个方向随机移动。撞墙或者走出边界的动作会导致agent留在原地，奖励值为0。（如动作为“up”，则80%向上走，10%向左，10%向右）



NOTE：

1. 请确保所有state同步更新，不能在同一轮中使用更新完成的状态的value更新其他状态。

HINT：

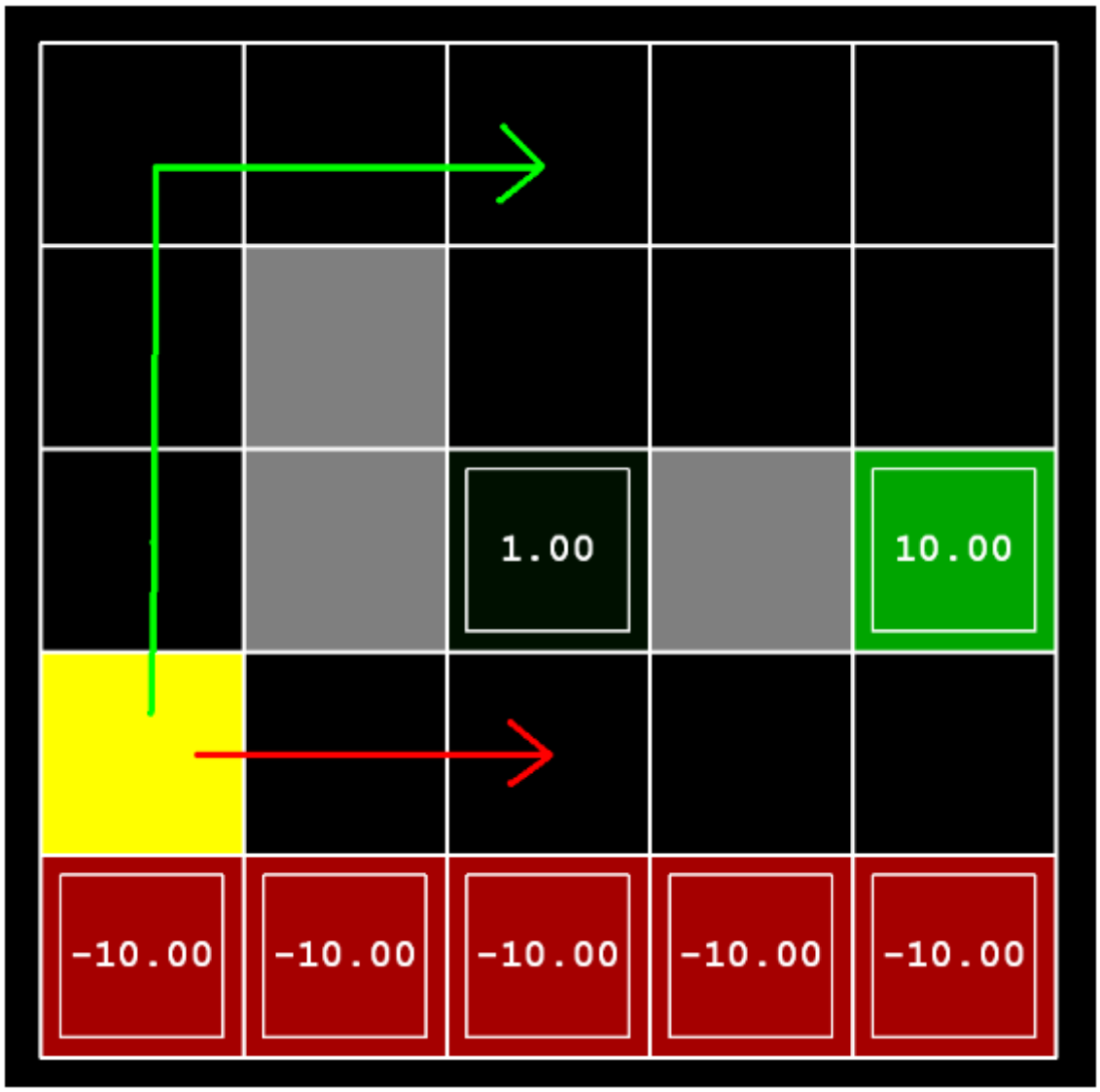
- valueIterationAgents.py中给出了值迭代agent的代码框架。如果实现正确，运行以下命令也可以得到本题的结果，可以作为答案的验证：

```
python gridworld.py -a value -i 2
```

Q2：策略构建（计算题）

考虑 `DiscountGrid` 世界。在这个世界中有两个具有正奖励的终止状态，**近的出口**奖励为1，**远的出口**为10。网格的底部行由具有负奖励的终止状态组成，该区域中每个状态都有-10的奖励。起始状态是黄色方块。我们区分两种类型的路径：(1) **冒险路线**，沿着网格底部行附近行进；这些路径更短，但可能面临巨大的负回报风险，在下图中用红色箭头表示。(2) **稳妥路线**，沿着网格的顶部边缘行进。这些路径更长，但不太可能遭受巨大的负奖励。这些路径在下图中用绿色箭头表示。

噪音设置：假设除了exit外的任何动作都有80%向指定方向移动，另外20%的概率向垂直于此方向的两个方向随机移动。撞墙或者走出边界的动作会导致agent留在原地，奖励值为0。（如动作为“up”，则80%向上走，10%向左，10%向右）



1. 假定没有噪声（即 agent 的行动是确定的），请计算 discount 和生存奖励参数的设置范围，以生成几种不同类型的特定最优策略。你需要给出计算过程，并回答指定的策略的参数满足的关系式（或是证明其不存在）。

你需要考虑以下策略：

1. 倾向于选择 +1 出口，冒险路线；
2. 倾向于选择 +10 出口，冒险路线；

3. 倾向于选择 +1 出口，稳妥路线；
4. 倾向于选择 +10 出口，稳妥路线；
5. 避免走到出口（+1、+10、-10），因此整个过程不会结束；
6. 放弃游戏，直接进入 -10 出口。

本小问请在报告中回答。

2. 请给出 noise、discount 和生存奖励的特定设置，来分别生成第 1 小问中的前五种最优策略，即：假如 agent 的每次行动都幸运地免受噪声干扰，则遵循其最优策略时它将表现出给定的行为。你可以利用第 1 小问的计算结果来回答，但如果第一部分中不存在生成某种策略的参数，那么你就需要继续探索 noise>0 的情况。

本小问的结果请填写在 analysis.py 中。如果不存在这样的参数组合，请返回"NOT POSSIBLE"。

3. （附加题）令 noise=0.3，当 discount=0.2、0.4、0.6、0.8、1.0 时，不同的生存奖励参数会产生哪些最优策略？这里只考虑 agent 从起始状态开始的行为序列（假定每次行动都幸运地免受噪声干扰），因此如果两种策略下 agent 从起始状态开始的行为相同，即使一些无关格子的最优方向不同，也认为两种策略是等价的。你不需要求出解析解，只需要给出生存奖励参数的数值范围和对应的最优策略，并定性地分析最优策略为什么是这样的。（例如，当 discount=0.4 而生存奖励为 -4 的时候，最优策略是什么？为什么？）

你可能需要计算很多种生存奖励参数对应的最优策略才能覆盖所有的情况。可以利用以下定理减少计算量：对于固定策略（不一定是最优策略），其效用值随生存奖励参数线性变化。（使用这个定理前请先证明它。）

本小问请在报告中回答。

Q3：Q-Learning

值迭代agent实际上并没有通过经验进行学习。相反，它在与真实环境进行交互之前就已经完善了其策略。当它与环境进行交互时，它只是按照预先计算的策略行动。而这在现实世界中并不总是可行，因为我们常常难以获取真实MDP。

现在，你将编写一个Q-learning agent，它通过 `update(state, action, nextState, reward)` 方法在与环境交互的过程中学习。 `qlearningAgents.py` 实现了该agent的基本框架（ `QLearningAgent` ），并且可以使用参数 `-a q` 来选择它。

你需要实现 `update` 、 `computeValueFromQValues` 、 `getQValue` 和 `computeActionFromQValues` 方法。

NOTE：

1. 对于 `computeActionFromQValues` ，请注意Q值相同的情况，可以使用 `random.choice()` 函数。agent未见过的动作仍有一个Q值。

2. 确保在 `computeValueFromQValues` 和 `computeActionFromQValues` 中只通过调用 `getQValue` 来访问Q值。

测试

你可以通过以下命令观察你实现的Q-learning agent的表现（需要手动操控）：

```
python gridworld.py -a q -k 5 -m
```

- 可以通过设置参数 `--noise 0.0` 来消除噪音

运行以下命令测试结果：

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

Q4：ε-greedy

通过在 `getAction` 方法中实现epsilon-greedy来完成你的Q-learning agent，这使得它有一定概率选择随机动作，而不是选择其当前最大Q值对应的动作。**请注意，选择随机动作不应该最佳动作。**

通过调用 `random.choice()` 从列表中均匀地随机选择一个元素。通过使用 `util.flipCoin(p)` 来模拟具有成功概率p的随机变量，该函数以概率p返回True，并以概率1-p返回False。

测试

在实现了 `getAction` 方法之后，可以观察agent在GridWorld中的以下行为（使用epsilon = 0.3）。

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

- Q-value的结果应当与值迭代的结果相近，平均回报会比预计的略低一些因为动作有一定随机性。

改变epsilon进行观察：

```
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
```

运行以下命令测试结果：

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

在实现了Q-learning agent之后，以下命令应当能正确运行：

```
python crawler.py
```

如果程序结果有误，说明你的部分代码只适用于Gridworld，而没有抽象到MDP整体上。

Q5：Q-Learning与吃豆人

吃豆人将分为两个阶段。在第一个阶段，即训练阶段，吃豆人将学习有关位置和动作价值的信息。由于即使对于小网格世界来说，学习准确的Q值也需要很长时间，因此吃豆人的训练过程将默认以安静模式运行，没有GUI（或控制台）显示。一旦吃豆人的训练完成，他将进入测试模式。在测试阶段，吃豆人的 `self.epsilon` 和 `self.alpha` 将设置为0.0，停止Q学习并禁用探索，以允许吃豆人利用他

学到的策略。测试游戏默认在GUI中显示。在先前代码正确的情况下，你应该能够按如下方式运行Q-learning吃豆人agent：

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

请注意，`PacmanQAgent` 已经按照你实现 `QLearningAgent` 的方式为你定义好了。PacmanQAgent的唯一区别在于它具有更适合吃豆人问题的默认学习参数（epsilon=0.05，alpha=0.2，gamma=0.8）。请确保上面的命令没有异常并且你的agent基本能获得胜利（>80%）。

提示：如果你的QLearningAgent适用于 `gridworld.py` 和 `crawler.py`，但不能在smallGrid上为Pacman学到一个好的策略，可能是因为你的 `getAction` 与 `computeActionFromQValues` 方法在某些情况下没有正确考虑未见过的动作。**特别地，由于未见过的动作的Q值为零，如果所有已见过的动作都有负的Q值，那么未见过的动作可能会是最优的**。请留意 `util.Counter` 中的 `argMax` 函数，确保它能正确输出结果。

测试

运行以下命令测试结果：

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

NOTE：

1. 若想测试不同参数的影响，可以使用 `-a` 更改参数。例如 `-a epsilon=0.1,alpha=0.3,gamma=0.7`
2. 使用如下命令可以观察10次训练过程：

```
python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
```

在训练过程中，每100局会有一个关于吃豆人表现的统计数据。在训练期间，由于存在随机探索，因此即使学会了一个好的策略，吃豆人也会表现不佳。通常需要1000到1400局游戏，吃豆人的周期奖励才会变为正值。而到训练结束时，奖励应该保持为正并且相当高（在100到350之间）。

然而，对看似简单的mediumGrid进行相同的训练效果并不好。吃豆人的平均训练奖励在整个训练过程中恒为负，且训练需要很长时间。吃豆人在更大的世界里无法取胜，这是因为在当前设置下，不同的鬼魂位置，豆子剩余，吃豆人位置都会造成状态的不同，因而具有单独的Q值。吃豆人agent无法将撞到鬼魂这一对所有位置都会被视为不利的行为进行概括。

此外，你可以手动操控吃豆人完成游戏：

```
python pacman.py
```

Q6：值函数近似Q-Learning

你需要在 `qlearningAgents.py` 中实现值函数近似 Q-learning agent（`ApproximateQAgent` 类）。

NOTE：

1. 值函数近似Q-learning假设存在特征函数 $f(s,a)$，并根据状态-动作对输出一个特征向量。

2. 特征函数在 `featureExtractors.py` 中已经定义好了，特征向量均为 `util.Counter` 对象，所有无关特征值为0。

值函数近似Q函数为如下形式：$Q(s,a)=\sum_{i=0}^{n} f_i(s,a)w_i$ ， w_i 为每个特征对应的权重。在实现时，你需要将权重向量当作dict，将特征函数返回的向量（ `featureExtractors.py` ）转化为实际的Q值。以下为更新方式：

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$
$$difference = (r + \gamma \max_{a'} Q(s',a')) - Q(s,a)$$

在默认情况下 `ApproximateQAgent` 使用 `IdentityExtractor` ，此时你的agent应该和 `PacmanQAgent` 表现一致：

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

重要： `ApproximateQAgent` 继承自 `QLearningAgent` ，因此它的部分方法与后者相同（如 `getAction` ）。请确保在 `QLearningAgent` 获取Q值时，是通过 `getQValue` 而不是直接获取的。

在正确实现值函数近似Q-learning之后，使用一个更好的特征提取器可以轻松取胜：

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

以下地图比较大，训练可能需要几分钟，但是吃豆人应仍能取得胜利：

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

甚至于只需要50次训练即可基本保证胜利。

测试

运行以下命令测试结果：

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