# Project 3: 强化学习

参考<https://inst.eecs.berkeley.edu/~cs188/fa19/project3/>

## 引言

在这个项目中，你需要实现value-iteration和Q-learning。首先，你将在Gridworld中测试编写的agents(智能体/代理)，然后将它们应用到模拟机器人控制器(Crawler)和Pacman。

该项目包含了一个autograder，用于在你的机器上为你得到的solutions评分。可以通过下面的命令运行测试所有的问题：

python autograder.py

此外，它也可以在特定的问题上测试，例如下面的命令在q2上测试：

python autograder.py -q q2

同时，它能够在特定的测试样例上运行，形式如下：

python autograder.py -t test\_cases/q2/1-bridge-grid

这个项目的代码包含以下文件，可以获取[zip文件](https://inst.eecs.berkeley.edu/~cs188/fa19/assets/files/reinforcement.zip)。

|  |  |
| --- | --- |
| **需要修改的文件** |  |
| valueIterationAgents.py | 解决已知MDPs的价值迭代智能体 |
| qlearningAgents.py | Gridworld，Crawler和Pacman的Q-learning智能体 |
| analysis.py | 该文件用于放置你对项目中问题做出的解答 |
| 需要阅读但不需要修改的文件 |  |
| mdp.py | 在一般MDPs定义的方法 |
| learningAgents.py | 定义需要扩展的基础类ValueEstimationAgent和QLearningAgent |
| util.py | 工具类，包含在Q-learners中非常有用的util.Counter |
| gridworld.py | 实现gridworld |
| featureExtractors.py | 用于在(state, action)对上提取特征的类。用于近似 Q-learning 智能体(在 qlearningAgents.py) |
| 可以忽略的文件 |  |
| environment.py | 用于一般强化学习环境的抽象类。由gridworld.py使用 |
| graphicsGridworldDisplay.py | 网格世界的图形显示 |
| graphicsUtils.py | 图形实用程序 |
| textGridworldDisplay.py | 网格世界文本界面的插件 |
| crawler.py | 爬行者程序代码和测试工具。你将运行它，但无需编辑它 |
| graphicsCrawlerDisplay.py | 爬行者机器人的图形用户界面 |
| autograder.py | 项目自动评分器 |
| testParser.py | 分析自动评分器测试和解决方案文件 |
| testClasses.py | 常规自动评分器的测试类 |
| test\_cases/ | 包含每个问题的测试用例的目录 |
| reinforcementTestClasses.py | 项目3特定自动评分器的测试类 |

**要编辑和提交的文件：**

你将在作业期间填写 valueIterationAgents.py、qlearningAgents.py 和 analysis.py 的部分内容。请不要更改此项目中的其他文件或提交除这些文件之外的任何原始文件。

**注意：**

你只需要提交 reinforcement.token，通过运行 submission\_autograder.py 生成。它包含本地自动评分器的评估结果，以及所有代码的副本。你不需要提交任何其他文件。

**评估：**

你的代码将自动分级以确保技术正确性。请不要更改代码中任何提供的函数或类的名称，否则你将对自动评分器造成严重破坏。但是，你实施的正确性——而不是自动评分器的判断——将是你分数的最终评判者。如有必要，我们将对作业进行单独审查和评分，以确保你的工作获得应有的学分。

**学术不诚实：**

我们将根据班级中其他提交的代码检查你的代码是否存在逻辑冗余。如果你复制别人的代码并稍作修改后提交，我们就会知道。这些作弊检测器很难被骗，所以请不要尝试。我们相信你们只提交自己的作品；请不要让我们失望。如果你这样做，我们将追求我们所能得到的最强烈的后果。

**获得帮助：**

你并不孤单！如果你发现自己卡在某个地方，请联系课程工作人员寻求帮助。办公时间、专区和论坛随时为你提供支持；请使用它们。如果你无法安排我们的办公时间，请告诉我们，我们会安排更多时间。我们希望这些项目是有益的和有指导意义的，而不是令人沮丧和沮丧的。但是，除非你提出要求，否则我们不知道何时或如何提供帮助。

**讨论：**请注意不要剧透。

## MDPs

首先，在使用箭头键的手动控制模式下运行Gridworld：

python gridworld.py -m

你会看到课堂上的两个出口布局。蓝点是智能体。请注意，当您按下*向上*键时，智能体实际上只有 80% 的可能向北移动，10%向左，10%向西，0%向反方向。这就是 Gridworld 智能体的life！如果按下*向上*键，智能体一定向上走，则是一种确定Grid world，可使用A\*等搜索算法解决。

您可以控制模拟的许多方面。运行以下命令可以获得完整的选项列表：

python gridworld.py -h

默认智能体随机移动：

python gridworld.py -g MazeGrid

您应该看到随机智能体在网格周围反弹，直到它发生在退出时。显然，这不是 AI 智能体结束的最佳时机。

*注意：*Gridworld MDP 的运行流程是：您首先必须进入预终端状态（GUI 中显示的双框），然后事件实际结束之前采取特殊的“退出”操作（在真正终端状态中称为 TERMINAL\_STATE，这未显示在 GUI 中）。如果您手动运行事件，由于折扣率(discount rate)，您的总回报可能会低于您的预期。(折扣率可通过-d 更改，默认为 0.9)。

查看伴随图形输出的控制台输出（或使用-t获得所有文本）。您将获得智能体经历的每次转换（要关闭此功能，请使用 -q）。

与 Pacman 一样，位置由 (x,y) 笛卡尔坐标表示，任何数组由 [x][y] 索引，'north' 是增加 y 的方向，等等。默认情况下，大多数转换都会收到奖励为零，但您可以使用living reward选项 (-r) 更改此设置。

## 问题1：价值迭代(Value iteration)

回忆一下值迭代状态更新方程：

其中，T表示在状态s下取动作a转移到状态的概率（可以理解为环境模型或者动态环境，已知），R表示在s状态下取动作a转移到的奖励，是折扣率（折扣因子）。

在 valueIterationAgents.py 中，已经为你部分指定了 ValueIterationAgent 类，你需要在其中编写一个价值迭代智能体。你的价值迭代智能体是一个离线规划器，而不是一个强化学习智能体，因此在其初始规划阶段，相关的训练选项是价值迭代应该运行的迭代次数（选项 -i）。在构造函数返回之前，ValueIterationAgent 在构造时接收一个 MDP 并运行指定次数的价值迭代。

值迭代进行k步计算来近似最优值。除了运行值迭代外，ValueIterationAgent还使用实现以下方法：

computeActionFromValues(state)：根据self.values给出的值函数计算最佳行动

computeQValueFromValues(state, action)：返回根据self.values给出的值函数给出的（状态, 动作）对的Q值

这些数值都显示在 GUI 中：价值是正方形中的数字，Q值是正方形划分为四份中的数字，策略是每个正方形中的箭头。

重要事项：使用“批处理”版本的值迭代，其中每个向量都是从一个固定的向量计算得来，而不是像“在线”版本一样更新一个合适的单一的权重向量。这意味着，当一个状态的值在迭代k中根据其后续状态的值更新时，值更新计算中使用的后续状态值应该是从迭代k-1中更新的值（即使一些后续状态已经在迭代k中更新）。这些差异在第91页第4.1章的[Sutton&Barto](https://inst.eecs.berkeley.edu/~cs188/fa19/assets/files/SuttonBartoIPRLBook2ndEd.pdf)中讨论。

注意：由深度k值（反应了接下来k个奖励）合成的策略实际上将反应下一个k+1的奖励（也就是返回）。同样地，Q值也将会反映比价值多一个奖励（即返回）。

你应当返回合成策略。

提示：你可以选择使用util.py中的计数器类util.Counter，它是默认值为零的字典。但是，注意argMax：你想要的实际上的argmax可能是一个不在计数器中的关键词。

注意：当一个状态在MDP中没有可用的操作时，一定要处理这个情况（想想这对未来的奖励意味着什么）。

要测试你的实现代码，请运行自动评分器：

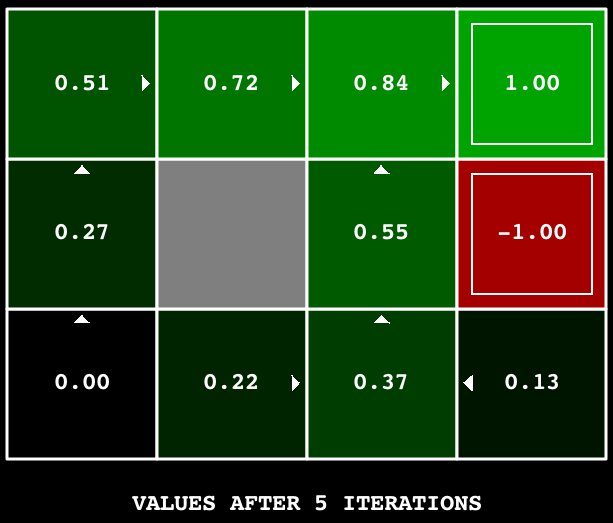
python autograder.py -q q1

下面的命令会加载你的ValueIterationAgent，它将计算一个策略并执行10次。按一个键来使用价值、Q值和模拟进行循环。你应该发现，初始状态的值（V(start)，你可以在GUI中读出）和经验结果的平均奖励（在执行完10轮后打印）是相当接近的。

python gridworld.py -a value -i 100 -k 10

提示：在默认的BookGrid上，运行5次迭代的值迭代应该给你这个输出：

python gridworld.py -a value -i 5

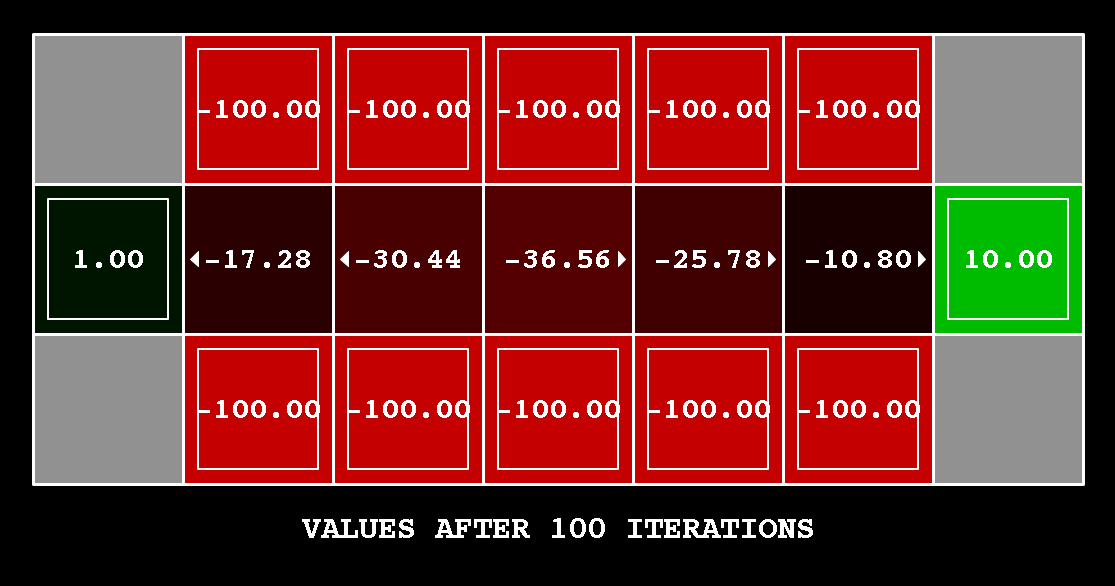


评分：你的值迭代智能体将在一个新的网格上评分。我们将在固定迭代次数并收敛时检查你的价值、Q值和策略(例如，经过100次迭代)。

## 问题2：过桥分析(Bridge Crossing Analysis)

BridgeGrid 是一个网格世界地图，低回报终端状态和高回报终端状态被一条狭窄的“桥”隔开，桥的两边都是高负回报的鸿沟。智能体从接近低回报状态开始。使用默认折扣 0.9 和默认噪声 0.2，最优策略不会过桥。仅更改折扣和噪声参数中的一个，以便最优策略导致智能体尝试过桥。将你的答案放在 analysis.py 的 question2() 中。 （噪音是指智能体在执行操作时以意外的后继状态结束的频率。）默认值对应于：

python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.2

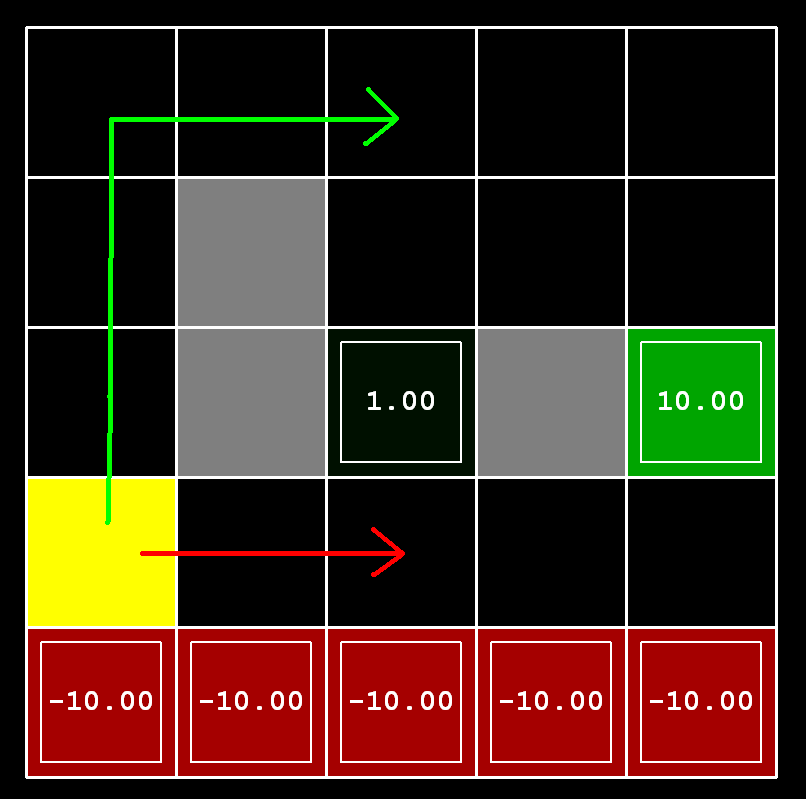


评分：我们将检查您是否仅更改了一个给定参数，并且通过此更改，正确的值迭代智能体应该通过桥。要检查您的答案，请运行 autograder：

python autograder.py -q q2

## 问题三：策略(Policies)

考虑 DiscountGrid 布局，如下所示。该网格有两个具有正收益的终端状态（在中间一行），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。网格的底行由具有负收益的终端状态组成（以红色显示）；这个“悬崖”区域中的每个状态的收益为 -10。起始状态是黄色方块。我们区分两种类型的路径：（1）“冒险悬崖”并在网格底部行附近行进的路径；这些路径较短，但有可能获得大量负收益，在下图中用红色箭头表示。 (2) “避开悬崖”并沿着网格顶部边缘行进的路径。这些路径更长，但不太可能招致巨大的负回报。这些路径在下图中用绿色箭头表示。



在这个问题中，您将为这个 MDP 选择折扣、噪音和living rewards参数的设置，以产生几种不同类型的最优策略。您为每个部分设置的参数值应该具有以下属性：如果您的智能体遵循其最优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没有实现特定行为，则通过返回字符串“NOT POSSIBLE”来断言该策略是不可能的。

以下是您应该尝试制定的最佳策略类型：

1. 更喜欢较近出口 (+1)，冒着悬崖的风险 (-10)
2. 更喜欢较近出口 (+1)，但避免悬崖 (-10)
3. 更喜欢远方的出口 (+10)，冒着悬崖的风险 (-10)
4. 更喜欢远处的出口 (+10)，避开悬崖 (-10)
5. 避开出口和悬崖（这样单个迭代事件就永远不会结束）

要检查您的答案，请运行 autograder：

python autograder.py -q q3

question3a() 到 question3e() 都应该在 analysis.py 中返回一个 3元组（折扣、噪音、living reward）。

注意：您可以在 GUI 中检查您的策略。例如，使用3(a)的正确答案，(0,1)中的箭头应指向东，(1,1)中的箭头也应指向东，而(2,1)中的箭头应指向北。

注意：在某些机器上您可能看不到箭头。在这种情况下，按下键盘上的按钮切换到 qValue 显示，并通过为每个状态获取可用 qValues 的 argmax 来在心里计算策略。

评分：我们将检查每种情况下是否返回了所需的策略。

## 问题四：异步值迭代

在 AsynchronousValueIterationAgent 中编写一个值迭代智能体，在 valueIterationAgents.py 中已经为您部分指定。您的价值迭代智能体是离线规划器，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迭代次数（选项 -i）。 AsynchronousValueIterationAgent 在构造时采用 MDP，并在构造函数返回之前针对指定的迭代次数运行循环值迭代（在下一段中描述）。请注意，所有这些值迭代代码都应放在构造函数（\_\_init\_\_ 方法）中。

这个类被称为 AsynchronousValueIterationAgent 的原因是因为我们将在每次迭代中只更新一个状态，而不是进行批处理式更新。这是循环值迭代的工作原理。在第一次迭代中，只更新状态列表中第一个状态的值。在第二次迭代中，只更新第二次的值。继续下去，直到您更新了每个状态的值一次，然后从第一个状态开始返回以进行后续迭代。如果选择更新的状态是终端状态，则该迭代中不会发生任何事情。您可以将其实现为对代码框架中定义的状态变量的索引。

提醒一下，这是值迭代状态更新方程：

正如课堂上讨论的那样，值迭代对不动点方程进行迭代。也可以以不同的方式更新状态值，例如以随机顺序（即随机选择一个状态，更新其值，然后重复）或以批处理方式（如 Q1）。在这个问题中，我们将探索另一种技术。

AsynchronousValueIterationAgent 继承自 Q1 的 ValueIterationAgent，因此您唯一需要实现的方法是 runValueIteration。由于超类构造函数调用 runValueIteration，覆盖它足以根据需要更改智能体的行为。

注意：确保处理状态在 MDP 中没有可用操作的情况（想想这对未来的奖励意味着什么）。

要测试您的实现，请运行 autograder。运行时间应该不到一秒钟。如果需要更长的时间，您可能会在项目的后期遇到问题，所以现在就让您的实施更有效率。

python autograder.py -q q4

以下命令将您的 AsynchronousValueIterationAgent 加载到 Gridworld 中，它将计算一个策略并执行 10 次。按一个键循环显示价值、Q 值和模拟。您应该会发现开始状态的值（V(start)，您可以从 GUI 中读取它）和经验得到的平均奖励（在 10 轮执行完成后打印）非常接近。

python gridworld.py -a asynchvalue -i 1000 -k 10

评分：您的价值迭代智能体将在新的网格上评分。我们将在固定次数的迭代后和收敛时（例如，在 1000 次迭代后）检查您的价值、Q 值和策略。

## 问题五：优先级扫描价值迭代(Prioritized Sweeping Value Iteration)

您现在将实现 PrioritizedSweepingValueIterationAgent，它已在 valueIterationAgents.py 中为您部分指定。请注意，此类派生自 AsynchronousValueIterationAgent，因此唯一需要更改的方法是 runValueIteration，它实际上运行值迭代。

优先扫描尝试以可能改变策略的方式集中更新状态值。

对于此项目，您将实施标准优先清除算法的简化版本，[相关论文](http://papers.nips.cc/paper/651-memory-based-reinforcement-learning-efficient-computation-with-prioritized-sweeping.pdf)对此进行了描述。我们已针对我们的设置调整了该算法。首先，我们将状态 s 的前驱定义为通过采取某个动作 a 到达 s 的概率非零的所有状态。此外，作为参数传入的 theta 将表示我们在决定是否更新状态值时对错误的容忍度。这是您在实施中应遵循的算法。

* 计算所有状态的predecessors
* 初始化一个空的优先级队列
* 对于每个非终点状态s，进行以下操作（注意：要使自动评分器对此问题有效，必须按照self.mdp.getStates()返回的顺序迭代状态）：
  + 找到s在self.values中的当前值与s中所有可能动作的最高Q值之差的绝对值（这表示值的应当的取值）；称此数字为diff。不要在此步骤中更新self.values[s]。
  + 将s以-diff的优先级进入优先级队列（注意这是一个负数）。因为优先级队列是一个最小堆，所以我们使用负数，但我们希望优先更新错误较高的状态。
* Iteration在0, 1, 2, ...., self.iterations - 1中，执行以下操作：
  + 如果优先队列为空，则终止
  + 状态s出队列
  + 在self.values中更新状态s（如果不是最终状态）。
  + 对于s的每个前身p，执行：
    - 找到p在self.values中的当前值和p的所有可能动作的最高Q值之差的绝对值（这表示值的应当的取值）。称此数为diff。不要在此步骤中更新self.values[p]。
    - 如果diff > theta，将p以优先级-diff置入优先级队列中（注意这是个负数），只要它不是已经以相等或较低的优先级存在于优先级队列中。 和以前一样，我们使用了一个负数，因为优先级队列是一个最小堆，但是我们优先更新错误较高的状态。

关于执行的几个重要说明：

* 在计算状态的predecessors时，确保将它们存储在集合中，而不是列表中，以避免重复。
* 请在你的实现中使用util.PriorityQueue。 该类中的更新方法可能很有用；请查看它的文档。

要测试你的实现，请运行自动评分器。 应该不到一秒钟就能跑完。 如果需要更长的时间，你可能会在项目的后续阶段遇到问题，因此现在使你的实现更加高效。

python autograder.py -q q5

你可以使用以下命令在Gridworld中运行PrioritizedSweepingValueIterationAgen。

python gridworld.py -a priosweepvalue -i 1000

评分：你的优先扫描值迭代代理将在一个新的网格上评分。 我们将在固定迭代次数并收敛时（例如，在1000次迭代之后）检查你的价值、Q值和策略。

根据CS188课程的项目信息，后续还有相关的问题，本次实验不做探讨。