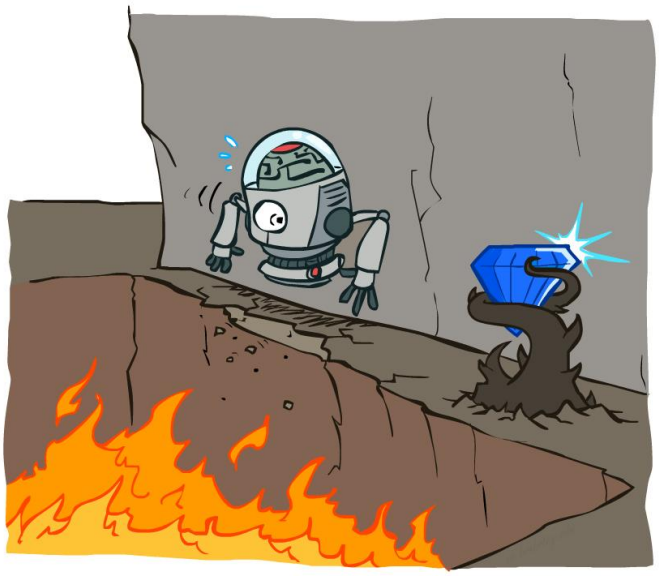
信息与计算机工程学院

人工智能导论

课程项目 3：强化学习



1、介绍

在本项目中，你将实现价值迭代和 Q 学习。你将首先在 Gridworld 上测试你的代理，然后将它们应用于模拟机器人控制器（爬行器）和吃豆人。

与以前的项目一样，此项目包括一个自动评分器，用于在计算机上对解决方案进行评分。可以使用以下命令对所有问题运行此操作：

```
python autograder.py
```

它可以针对一个特定问题运行，例如 q2，方法是：

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

它可以通过以下形式的命令为一个特定的测试运行：

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

有关使用自动批改程序的详细信息，请参阅 Project 0 中的自动批改程序教程。

此项目的代码包含以下文件，这些文件以 zip 文档形式提供。

你需要编辑的文件	
valueIterationAgents.py	用于求解已知 MDP 的值迭代代理。
qlearningAgents.py	用于 Gridworld、Crawler 和 Pacman 的 Q 学习代理。
analysis.py	用于回答项目中给出的问题。
你需要查看的文件	
mdp.py	定义常规 MDP 方法。
learningAgents.py	定义你的 Agent 将扩展的基类 ValueEstimationAgent 和 QLearningAgent。
util.py	里面有些可以帮助实现搜索算法的数据结构
gridworld.py	实现 Gridworld
featureExtractors.py	用于提取（state，action）特征的类。用于近似的 Q 学习代理（qlearningAgents.py ）。

其它的文件你可以忽视。

需要编辑和提交的文件：你需要完成 valueIterationAgents.py, qlearningAgents.py, 和 analysis.py，将你修改的文件和自动批改程序（submission_autograder）产生的结果，以及项目报告一同提交。请不要修改或提交其它文件。

项目评估：你的代码会通过自动批改来判断其正确性，因此请不要修改代码中其它任何函数或者类，否则你会让自动批改程序无法正常运行。然而，你的解题思路和方法是你最终成绩的决定因素。必要的话，我们会查看你的代码来保证你得到应得的成绩。

学术造假：我们会查看你的代码和其它学生提交的代码是否雷同。禁止抄袭了他人代码，或只做简单修改后提交，一旦发现，成绩立马作废，而且会影响到你能否通过此课程。

寻求帮助：当你感到自己遇到了困难，请向你的同学和老师寻求帮助。小组合作、答疑时间、课堂讨论，这些都是用来帮助你的，请积极利用这些资源。设计这些项目的目的是让你更有效地理解和掌握课堂知识，学会如何将理论知识应用于实践，解决实际问题，而不是为考核而考核，或者有意刁难你，所以请尽你所能，完成它们。遇到困难时，向课代表和老师提问。

2、MDPs

开始时，请在手动控制模式下运行 Gridworld，该模式使用箭头键：

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

你将看到类中的两个出口布局。蓝点是代理。请注意，当你向上按时，代理实际上只在 80% 的时间内向北移动。这就是 Gridworld（网格世界）里 agent 的生活！

你可以控制模拟的许多方面。通过运行以下命令可获得完整的选项列表：

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

缺省的代理会随机移动。

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

你应该看到随机代理在网格周围反弹，直到它在碰到一个出口。对于 AI 代理来说，这不是一个最好的例子。

注意：Gridworld MDP 是这样的，你首先必须进入预终端状态（GUI 中显示的双框），然后在实际结束之前采取特殊的“退出”操作（在真正的终端状态下称为 TERMINAL_STATE，没有显示在 GUI 中）。如果你手动运行，由于折扣率（用-d 更改；默认为 0.9），你的总回报可能会低于你的预期。

查看图形输出随附的控制台输出（或使用 -t 来采用文本输出）。系统将告知代理经历的每个转换（要关闭此功能，请使用 -q）。

与吃豆人一样，位置由 （x， y） 笛卡尔坐标表示，任何数组都由 [x][y] 索引，其中“北”是增加 y 的方向，依此类推。默认情况下，大多数转换将获得零奖励，但你可以使用奖励选项 （-r） 更改此设置。

3、项目内容

问题 1（6 分）：价值迭代

回想一下值迭代状态更新公式：

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

在 ValueIterationAgent 中编写一个值迭代代理，该代理已在 valueIterationAgents.py 中为你部分指定。你的价值迭代代理是线下计划器，而不是强化学习代理，因此相关的训练选项是它在其初始规划阶段应运行的价值迭代的迭代次数（选项 -i）。ValueIterationAgent 在构造上采用 MDP，并在构造函数返回之前为指定的迭代次数运行值迭代。

值迭代计算第 k 步的最优值V_k。除了 runValueIteration ，还可以使用V_k为 ValueIterationAgent 实现下面的方法：

- computeActionFromValues（state） 根据 self.values 给出的值函数计算最佳操作。
- computeQValueFromValues（state， action） 返回由 self.values 给出的值函数给出的 （state，action） 所对应的 Q 值。

这些数量都显示在 GUI 中：值是方块中的数字，Q 值是四分之一方块里的数字，策略是从每个方块出来的箭头。

重要提示：使用值迭代的“批处理”版本，其中每个向量V_k都是从固定向量V_{k-1}计算出来的。这意味着当状态 k 的值在迭代中更新时，计算中使用的后继状态值应为迭代 k-1 中的值（即使一些继承状态已经在迭代中更新到了 k）。

注意：从深度 k 合成的策略（反映下面 k 次奖励）实际上将反映下一个 k+1 的 奖励（即你返回π_{k+1}。同样，Q 值也将反映比当前值多一个奖励（即你返回Q_{k+1}）

你应该返回合成的π_{k+1}。

提示：你可以选择使用 util.py 中的 util.Counter 类，它是一个默认值为零的字典。但是，要小心 argmax：你想要的实际 argmax 可能是计数器中没有的 key！

注意：当状态在 MDP 中没有可用操作时，请确保处理这种情况（想想这对未来的奖励意味着什么）。

若要测试你的实现，请运行自动评分器：

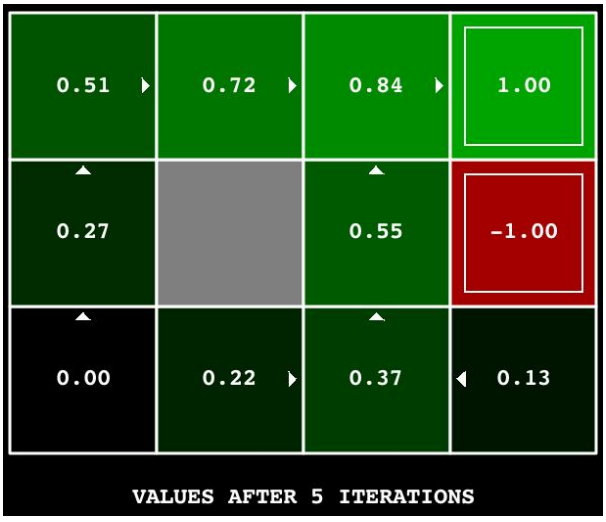
```
python autograder.py -q q1
```

以下命令加载你的 ValueIterationAgent，它将计算一个策略并执行 10 次。按键循环切换值、Q 值和模拟。你应该发现开始状态的值（V（start），你可以从 GUI 中读取）和经验结果的平均奖励（在 10 轮执行完成后打印）非常接近。

```
Python gridworld.py -a value -i 100 -k 10
```

提示：在默认的 BookGrid 上，运行 5 次迭代的值迭代应提供以下输出：

```
Python gridworld.py -a value -i 5
```

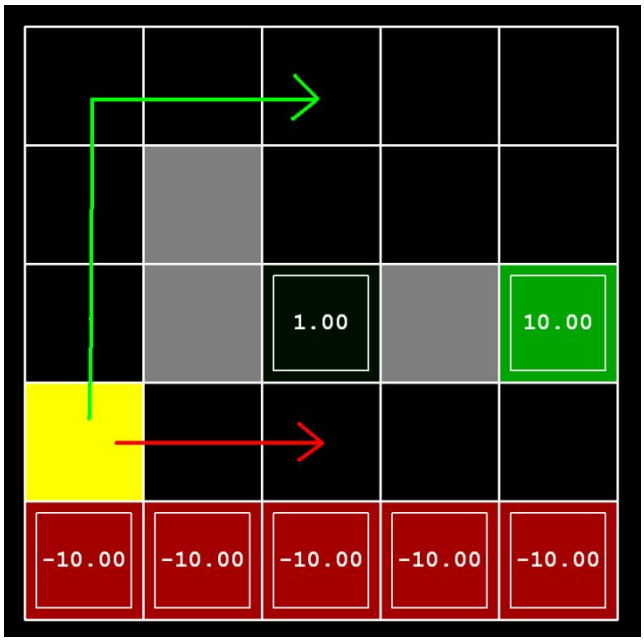


评分：你的价值迭代代理将在新网格上进行评分。我们将在固定的迭代次数和收敛（例如 100 次迭代后）检查你的值、Q 值和策略。

问题 2（5 分）：策略

考虑 DiscountGrid 布局，如下所示。这个网格有两个收益为正的终端状态（在中间行），一个收益为+1 的关闭退出和一个收益为+10 的远端退出。网格的底行由负收益的终端状态组成（以

红色显示）；这个“悬崖”地区的每个状态都有收益-10。起始状态是黄色方块。我们区分了两种类型的路径：（1）“冒险悬崖”并在网格底行附近行进的路径；这些路径较短，但有获得大量负收益的风险，如下图所示的红色箭头表示。（2）“避开悬崖”并沿网格顶部边缘行驶的路径。这些路径更长，但不太可能产生巨大的负回报。这些路径由下图中的绿色箭头表示。



在本问题中，你将通过修改 analysis.py 文件来为此 MDP 选择折扣、噪声和奖励参数的设置，以生成几种不同类型的最佳策略。每个部分的参数值设置应具有以下属性：如果你的代理遵循其最佳策略而不受任何干扰，它将表现出给定的行为。如果参数的任何设置都没有实现特定行为，则通过返回字符串“NOT POSSIBLE”来断言策略是不可能的。

以下是你应该尝试生成的最佳策略类型：

1. 更喜欢关闭出口（+1），冒着悬崖的风险（-10）
2. 更喜欢关闭出口（+1），但避免悬崖（-10）
3. 更喜欢远处的出口（+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. 避免出口和悬崖（所以运行永远不应该结束）

若要查看一组数字最终的行为，请运行以下命令以查看 GUI：

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

要检查你的答案，请运行自动评分器：

```
Python autograder.py -q q2
```

在 analysis.py 里的 question2a()到 question2e()应该分别返回一个 3 元素的元组（折扣，噪音，奖励 ）。

注意：你可以在 GUI 中检查你的策略。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，（2，1） 中的箭头应指向北。

注意：在某些计算机上，你可能看不到箭头。在这种情况下，按下键盘上的按钮切换到 qValue 显示，并通过获取每个状态的可用 qValues 的 arg 最大值来计算策略。

评分：我们将检查每种情况下是否返回了所需的策略。

问题 3（6 分）：Q-Learning

请注意，你的价值迭代代理实际上并没有从经验中学习。相反，它会考虑其 MDP 模型，以便在与真实环境交互之前达成完整的策略。当它与环境交互时，它只是遵循预先计算的策略（例如，它成为一个反射代理）。这种区别在像 Gridword 这样的模拟环境中可能很细微，但在现实世界中非常重要，因为真正的 MDP 并不存在。

你现在将编写一个 Q 学习代理，它在起始做得很少，而是通过其更新（状态、操作、nextState、reward）方法从与环境的交互中反复试验来学习。Q-learner 的框架已经在 qlearningAgents.py 的 QLearningAgent 中定义，你可以使用选项“-a q”选择它。对于此问题，你必须实现 update、computeValueFromQValues、getQValue 和 computeActionFromQValues 几种方法。

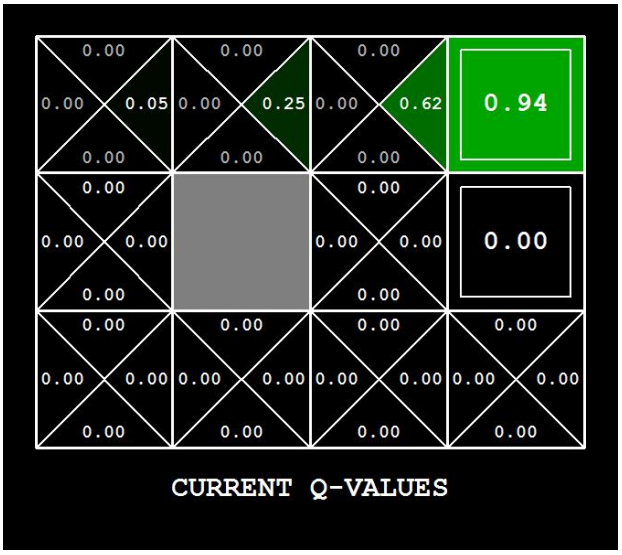
注意：对于 computeActionFromQValues，你应该随机地打破平局以获得更好的行为。对此，random.choice 函数应该会有所帮助。在某个状态下，代理以前从未见过的操作仍具有 Q 值，也就是说该 Q 值为零，如果代理之前看到的所有操作都具有负 Q 值，则从未见过的操作可能是最佳操作。

重要提示：确保在 computeValueFromQValues 和 computeActionFromQValues 函数中，只能通过调用 getQValue 来访问 Q 值。当你覆盖 getQValue 以直接使用状态-操作对而不是状态-操作对的功能时，这一抽象对问题 10 很有用。

Q 学习更新到位后，你可以使用键盘观看 Q 学习者在手动控制下学习：

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

回想一下，-k 将控制你的代理学习的集数。观察代理如何了解它刚刚所处的状态，而不是它移动到的状态，并“让学习随之而来”。提示：为了帮助调试，你可以使用 --noise 0.0 参数关闭噪声（尽管这显然使 Q 学习变得不那么有趣）。如果你手动引导吃豆人向北，然后向东，沿着最佳路径走四次，你应该看到以下 的 Q 值：



评分：我们将运行你的 Q 学习代理，并检查它是否学习到我们预期的 Q 值和策略，每个代理都会运行和我们的实现相同的算例。要对实现进行评分，请运行自动评分器：

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

问题 4（2 分）：Epsilon Greedy

通过在 getAction 中实现 epsilon 贪婪操作选择来完成你的 Q 学习代理，这意味着它会在 epsilon 的一小部分时间内选择随机操作，否则遵循其当前最佳 Q 值。请注意，选择随机操作可能会导致选择最佳操作——也就是说，你不应选择随机的次优操作，而应选择任何随机的合法操作。

你可以通过调用 random.choice 函数从列表中均匀地随机选择一个元素。你可以使用 util.flipCoin（p） 模拟具有概率 p 的二进制变量，该变量以概率 p 返回 True，以概率 1-p 返回 False。

实现 getAction 方法后，观察 GridWorld 中代理的以下行为（epsilon = 0.3）。

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

最终的 Q 值应类似于价值迭代代理的 Q 值，尤其是在行程良好的路径上。但是，由于随机操作和初始学习阶段，你的平均回报将低于 Q 值预测。

你还可以观察以下针对不同 epsilon 值的模拟。代理的行为是否符合你的预期？

```
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
```

若要测试实现，请运行自动评分器：

```
Python autograder.py -q Q4
```

无需额外的代码，你现在应该能够运行 Q 学习爬虫机器人：

```
python crawler.py
```

如果这不起作用，你可能编写了一些过分针对 GridWorld 问题的代码，你应该使其对所有 MDP 都适用。

这将调用课堂上爬行机器人使用你的 Q 学习者。尝试各种学习参数，了解它们如何影响代理的策略和操作。请注意，步伐延迟是模拟的参数，而学习率和 epsilon 是学习算法的参数，折扣因子是环境的属性。

问题 5（2 分）：Q-Learning 和吃豆人

是玩吃豆人的时候了！吃豆人将分两个阶段玩游戏。在第一阶段，训练，吃豆人将开始学习姿势和动作的价值。因为即使是微小的网格也需要很长时间才能学习准确的 Q 值，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，他将进入测试模式。测试时，Pacman 的 self.epsilon 和 self.alpha 将被设置为 0.0，有效地停止 Q 学习并禁用探索，以便让 Pacman 利用他的学习策略。默认情况下，测试游戏显示在 GUI 中。在不更改任何代码的情况下，你应该能够在非常小的网格里运行 Q-learning Pacman，如下所示：

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

请注意，PacmanQAgent 已经根据你编写的 QLearningAgent 定义了。PacmanQAgent 的不同之处在于它具有默认的学习参数，这些参数对 Pacman 问题更有效（epsilon=0.05，alpha=0.2，gamma=0.8）。如果上述命令无一例外地工作，并且你的代理至少在 80% 的时间内获胜，你将获得此问题的全额学分。自动评分器将在 2000 场训练比赛后运行 100 场测试游戏。

提示：如果你的 QLearningAgent 适用于 gridworld.py 和 crawler.py，但似乎没有在 smallGrid 上学习 Pacman 的良好策略，这可能是因为你的 getAction 和/或 computeActionFromQValues 方法在某些情况下没有正确考虑看不见的操作。特别是，由于根据定义，看不见的操作的 Q 值为零，如果所有已看到的操作都具有负 Q 值，则不可见的操作可能是最优的。当心来自 util.Counter 的 argMax 函数。

要对答案进行评分，请运行：

```
Python autograder.py -q Q5
```

注意：如果要尝试学习参数，可以使用选项 -a，例如 -a epsilon=0.1，alpha=0.3，gamma=0.7。然后，这些值将在代理中以 self.epsilon、self.gamma 和 self.alpha 的形式访问。

注意：虽然总共将进行 2010 场比赛，但不会显示前 2000 场比赛，因为选项 -x 2000 指定前 2000 场比赛进行训练（无输出）。因此，你只会看到吃豆人玩这些游戏的最后 10 个。训练游戏的数量也会作为选项 numTraining 传递给你的代理。

注意：如果你想观看 10 场训练比赛以了解发生了什么，请使用以下命令：

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

在训练期间，你将看到每 100 场比赛的输出，其中包含有关 Pacman 表现的统计数据。厄普西隆在训练中是积极的，所以吃豆人即使学会了一个好的策略，也会打得很差：这是因为他偶尔会随机探索进入幽灵。作为基准，吃豆人对 100 集片段的奖励应该需要 1000 到 1400 场比赛才能变为正值，这反映了他开始赢多于输。到训练结束时，它应该保持正数并且相当高（在 100 到 350 之间）。

确保你了解这里发生了什么：MDP 状态是 Pacman 面临的确切电路板配置，现在复杂的转换描述了对该状态的整个变化。吃豆人移动但幽灵没有回复的中间游戏配置不是 MDP 状态，而是捆绑在过渡中。

一旦吃豆人完成训练，他应该在测试游戏中非常可靠地获胜（至少 90% 的时间），因为现在他正在利用他学到的策略。

但是，你会发现在看似简单的 mediumGrid 上训练相同的代理效果不佳。在我们的实现中，吃豆人的平均训练奖励在整个训练过程中保持负数。在测试时，他打得很糟糕，可能输掉了所有的测试游戏。培训也将需要很长时间，尽管它无效。

吃豆人无法在更大的布局上获胜，因为每个板配置都是具有单独 Q 值的独立状态。

问题 6（4 分）：近似 Q-Learning

实现一个近似的 Q 学习代理，该代理学习状态特征的权重，其中许多状态可能共享相同的特征。在 qlearningAgents.py 的 ApproximateQAgent 类中编写你的实现，它是 PacmanQAgent 的一个子类。

注意：近似 Q 学习假设存在定义在状态和动作对上的特征函数$f(s,a)$，产生一个向量的特征值$[f_1(s,a),\ldots,f_i(s,a),\ldots,f_n(s,a)]$。我们在 featureExtractors.py 里为你提供了特性函数。特征向量是 util.Counter（计数器，一个类似字典的数据结构）包含非零的特征和值；所有省略的要素的值均为零。

近似 Q 函数采用以下形式：

$$Q(s,a) = \sum_{i=1}^{n} f_i(s,a) w_i$$

每一个权重w_i都和特征$f_i(s,a)$相对应。在你的代码里，应将权重向量实现为将特征（特征提取器返回的）映射到权重值的字典。你将更新你的权重向量，与更新 Q 值的方式类似：

$$w_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s,a)$$

$$\text{difference} = \left(r + \gamma \max_{a'} Q(s',a')\right) - Q(s,a)$$

请注意，difference 项和正常的 Q-Learning 是一样的，而r是经历的回报。

默认情况下，ApproximateQAgent 使用 IdentityExtractor，它将单个特征分配给每个（state，action）对。有了这个特征提取器，你的近似 Q 学习代理应该与 PacmanQAgent 一样工作。你可以使用以下命令对此进行测试：

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

重要提示：NearQAgent 是 QLearningAgent 的一个子类，因此它共享几种方法，如 getAction。确保 QLearningAgent 中的方法调用 getQValue 而不是直接访问 Q 值，以便在近似代理中覆盖 getQValue 时，新的近似 q 值用于计算操作。

一旦你确信你的近似学习器可以正确使用身份特征，请使用我们的自定义特征提取器运行你的近似 Q 学习代理，该代理可以轻松学习获胜：

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

对于你的 EquiQAgent 来说，即使是更大的布局也应该没有问题（警告：这可能需要几分钟来训练）：

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

如果你没有错误，你的近似 Q 学习代理应该几乎每次都用这些简单的功能获胜，即使只有 50 场训练游戏。

评分：我们将运行你的近似 Q 学习代理，并检查它是否学习与我们的参考实现相同的 Q 值和特征权重，当每个代理都提供相同的示例集时。要对实现进行评分，请运行自动评分器：

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

祝贺你！你现在有了一个会学习的吃豆人！

4、项目报告

简要清晰地描述完成项目时遇到的困难，采用的解决方法，提出改进意见，和总结每个小组成员的贡献。

5、提交

在提交你的解答之前，你需要通过执行 submission_autograder.pyc 来产生几个文件。在运行这个程序之前，你必须确认所有与 autograde 有关的文件都处在原始状态，没有做过任何的改动。

假如你编辑过任何 autograde 的文件，请重新下载一份项目代码，仅仅替换你作解答的文件，否则运行 submission_autograder.pyc 将无法通过。

此外，submission_autograder.pyc 要在 Python 3.6（准确的说是 3.6.13，你可以用 Anaconda 来安装正确的 Python 版本）下执行，否则会报错。

最后，submission_autograder.pyc 需要用 rsa 库来给你的成绩加密，假如你没有的话，请用下面的命令安装 rsa 库。

```
conda install -c conda-forge rsa
```

或者用下面的命令，假如你没有 conda。

```
pip install rsa
```

进到你的 reinforcement 文件夹里，执行以下命令：

```
python submission_autograder.pyc
```

成功执行后，该命令会输出你的各个题目的得分和最后总分，并在 grade 文件夹里会生成一个 log 文件和一个 token 文件。确认该分数和你自己运行 autograder.py 得到的分数相同后，将整个 grade 文件夹和你修改过的文件（其它没有修改过的，例如 autograder.py，不需要）以及项目报告，放在一个以你的组号命名的文件夹里，生成一个 zip 文件，一并提交上来。