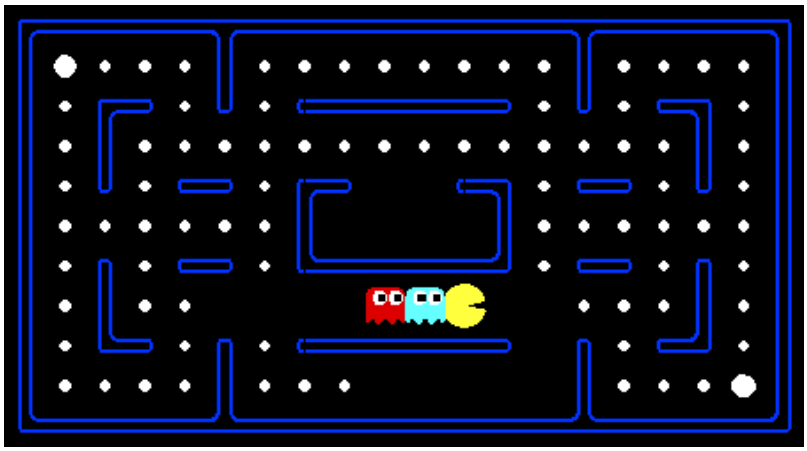
Project 2: Multi-Agent Search
项目 2：多智能体搜索

Due: **Friday, September 26, 11:59 PM PT**.

截止日期： **9 月 26 日星期五晚上 11:59（太平洋时间）** 。



Pacman, now with ghosts.

吃豆人，现在有了幽灵。

Minimax, Expectimax, 极小极大值，期望极大值，
Evaluation 评估

TABLE OF CONTENTS
目录

Introduction 介绍

In this project, you will design agents for the classic version of Pacman, including ghosts. Along the way, you will implement both minimax and expectimax search and try your hand at evaluation function design.

在这个项目中，你将为经典版吃豆人游戏设计智能体，包括幽灵智能体。在此过程中，你将实现极小极大搜索算法和期望极大搜索算法，并尝试设计评估函数。

The code base has not changed much from the previous project, but please start with a fresh installation, rather than intermingling files from project 1.

代码库与之前的项目相比变化不大，但请从全新安装开始，不要将项目 1 中的文件混用。

As in project 1, this project includes an autograder for you to grade your answers on your machine. This can be run on all questions with the command:

与项目 1 一样，本项目也包含一个自动评分器，您可以在自己的电脑上对答案进行评分。您可以使用以下命令对所有问题运行该评分器：

```
python autograder.py
```

It can be run for one particular question, such as q2, by:

可以通过以下方式针对特定问题（例如问题 2）运行该程序：

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

It can be run for one particular test by commands of the form:

可以通过如下形式的命令运行特定测试：

```
python autograder.py -t test_cases/q2/0-small-tree
```

By default, the autograder displays graphics with the `-t` option, but doesn't with the `-q` option. You can force graphics by using the `--graphics` flag, or force no graphics by using the `--no-graphics` flag.

默认情况下，自动评分器会使用 `-t` 选项显示图形，但不会使用 `-q` 选项显示图形。您可以使用 `--graphics` 标志强制显示图形，或使用 `--no-graphics` 标志强制不显示图形。

See the autograder tutorial in Project 0 for more information about using the autograder.

有关使用自动评分器的更多信息，请参阅项目 0 中的自动评分器教程。

The code for this project contains the following files, available as a zip archive.

该项目的代码包含以下文件，以 zip 压缩包的形式提供。

Files you'll edit: 您将编辑的文件：	
`multiAgents.py`	Where all of your multi-agent search agents will reside. 所有多代理搜索代理都将位于此处。
Files you might want to look at: **您可能想查看的文件：**	
`pacman.py`	The main file that runs Pacman games. This file also describes a Pacman GameState type, which you will use extensively in this project. 这是运行吃豆人游戏的主文件。该文件还描述了一种吃豆人游戏状态类型，您将在本项目中大量使用它。
`game.py`	The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.

	Pacman 世界运行的逻辑。此文件描述了几个支持类型，例如 AgentState、Agent、Direction 和 Grid。
`util.py`	Useful data structures for implementing search algorithms. You don't need to use these for this project, but may find other functions defined here to be useful. 用于实现搜索算法的实用数据结构。虽然本项目不需要使用这些数据结构，但您可能会发现这里定义的其他函数也很有用。

Supporting files you can ignore:

您可以忽略以下支持文件：

`graphicsDisplay.py`	Graphics for Pacman 吃豆人图形
`graphicsUtils.py`	Support for Pacman graphics 支持吃豆人图形
`textDisplay.py`	ASCII graphics for Pacman 吃豆人的 ASCII 图形
`ghostAgents.py`	Agents to control ghosts 控制鬼魂的特工
`keyboardAgents.py`	Keyboard interfaces to control Pacman 用于控制吃豆人的键盘界面
`layout.py`	Code for reading layout files and storing their contents 用于读取布局文件并存储其内容的代码
`autograder.py`	Project autograder 项目自动评分器
`testParser.py`	Parses autograder test and solution files 解析自动评分器测试和解决方案文件
`testClasses.py`	General autograding test classes 通用自动评分测试课程

`test_cases/`	Directory containing the test cases for each question 包含每个问题的测试用例的目录
`multiagentTestClasses.py`	Project 3 specific autograding test classes 项目 3 特定自动评分测试课程

Files to Edit and Submit: You will fill in portions of `multiAgents.py` during the assignment. Once you have completed the assignment, you will submit these files to Gradescope (for instance, you can upload all `.py` files in the folder). Please do not change the other files in this distribution.

需要编辑和提交的文件：在作业过程中，您需要填写 `multiAgents.py` 的部分内容。作业完成后，请将这些文件提交到 Gradescope（例如，您可以上传文件夹中的所有 `.py` 文件）。请勿修改此分发包中的其他文件。

Evaluation: Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation – not the autograder's judgements – will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

评分：系统将对您的代码进行技术正确性自动评分。请勿更改代码中提供的任何函数或类的名称，否则将导致自动评分系统出现故障。但是，最终得分将取决于您代码实现的正确性，而非自动评分系统的判断。如有必要，我们将单独审核并评分作业，以确保您获得应有的分数。

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

学术不端行为：我们会将你的代码与班级其他同学提交的代码进行逻辑冗余性比对。如果你抄袭他人的代码并稍作修改就提交，我们会发现。这些作弊检测系统很难被蒙骗，所以请不要尝试。我们相信你们都会提交自己的作品；请不要让我们失望。如果你们违反规定，我们将采取一切必要的法律措施。

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours, section, and the discussion forum are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how

to help unless you ask.

寻求帮助：你并不孤单！如果你遇到任何问题，请联系课程团队寻求帮助。我们提供办公时间、辅导课和讨论区，随时为你提供支持；请充分利用这些资源。如果你无法参加办公时间，请告知我们，我们会安排更多时间。我们希望这些项目能够让你受益匪浅，而不是令人沮丧和气馁。但是，如果你不主动寻求帮助，我们就无法知道何时以及如何提供帮助。

Discussion: Please be careful not to post spoilers.

讨论：请注意不要发布剧透内容。

Topics needed for this project:

本项目所需主题：

- Game Trees I: Video 6, Note 3.1-3.2

  博弈树 I：视频 6，注释 3.1-3.2
- Game Trees II: Video 7, Note 3.3-3.6

  博弈树 II：视频 7，注释 3.3-3.6

Welcome to Multi-Agent Pacman
欢迎来到多智能体吃豆人

First, play a game of classic Pacman by running the following command:

首先，运行以下命令来玩经典的吃豆人游戏：

```
python pacman.py
```

and using the arrow keys to move. Now, run the provided ReflexAgent in multiAgents.py

并使用方向键移动。现在，运行 multiAgents.py 中提供的 ReflexAgent。

```
python pacman.py -p ReflexAgent
```

Note that it plays quite poorly even on simple layouts:

请注意，即使在简单的布局上，它的表现也相当糟糕：

```
python pacman.py -p ReflexAgent -l testClassic
```

Inspect its code (in `multiAgents.py`) and make sure you understand what it's doing.

检查其代码（在 `multiAgents.py` 中），并确保您理解它正在做什么。

Q1 (4 pts): Reflex Agent

Q1（4 分）：反射剂

Improve the `ReflexAgent` in `multiAgents.py` to play respectably. The provided reflex agent code provides some helpful examples of methods that query the `GameState` for information. A capable reflex agent will have to consider both food locations and ghost locations to perform well. Your agent should easily and reliably clear the `testClassic` layout:

改进 `multiAgents.py` 中的 `ReflexAgent` ，使其能够正常运行。提供的 ReflexAgent 代码包含一些查询 `GameState` 信息的方法示例。一个优秀的 ReflexAgent 需要同时考虑食物和幽灵的位置才能表现出色。你的代理应该能够轻松可靠地清除 `testClassic` 布局：

```
python pacman.py -p ReflexAgent -l testClassic
```

Try out your reflex agent on the default `mediumClassic` layout with one ghost or two (and animation off to speed up the display):

在默认的 `mediumClassic` 布局上，使用一到两个幽灵（并关闭动画以加快显示速度）来测试你的反射代理：

```
python pacman.py --frameTime 0 -p ReflexAgent -k 1
```

```
python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

How does your agent fare? It will likely often die with 2 ghosts on the default board, unless your evaluation function is quite good.

你的智能体表现如何？除非你的评估函数非常出色，否则在默认棋盘上遇到两个幽灵棋子时，它很可能会经常失败。

Note: Remember that `newFood` has the function `asList()`

注意：请记住 `newFood` 具有 `asList()` 函数。

Note: As features, try the reciprocal of important values (such as distance to food) rather than just the values themselves.

注意：作为特征，请尝试使用重要值的倒数（例如到食物的距离），而不是仅仅使用值本身。

Note: The evaluation function you're writing is evaluating state-action pairs; in later parts of the project, you'll be evaluating states.

注意：你正在编写的评估函数是评估状态-动作对；在项目的后续部分，你将评估状态。

Note: You may find it useful to view the internal contents of various objects for debugging. You can do this by printing the objects' string representations. For example, you can print `newGhostStates` with `print(newGhostStates)`.

注意：为了进行调试，查看各种对象的内部内容可能很有用。您可以通过打印对象的字符串表示形式来实现这一点。例如，您可以使用 `print(newGhostStates)` 打印 `newGhostStates` 。

Options: Default ghosts are random; you can also play for fun with slightly smarter directional ghosts using `-g DirectionalGhost`. If the randomness is preventing you from telling whether your agent is improving, you can use `-f` to run with a fixed random seed (same random choices every game). You can also play multiple games in a row with `-n`. Turn off graphics with `-q` to run lots of games quickly.

选项：默认幽灵是随机的；您也可以使用 `-g DirectionalGhost` ，让幽灵的移动方向更智能一些，从而获得更好的游戏体验。如果随机性让您无法判断智能体是否有所改进，可以使用 `-f` 使用固定的随机种子运行游戏（每局游戏都使用相同的随机选择）。您还可以使用 `-n` 参数连续进行多局游戏。使用 `-q` 参数关闭图形显示，可以快速运行大量游戏。

Grading: We will run your agent on the `openClassic` layout 10 times. You will receive 0 points if your agent times out, or never wins. You will receive 1 point if your agent wins at least 5 times, or 2 points if your agent wins all 10 games. You will receive an additional 1 point if your agent's average score is greater than 500, or 2 points if it is greater than 1000. You can try your agent out under these conditions with

评分：我们将使用 `openClassic` 布局运行您的代理 10 次。如果您的代理超时或从未获胜，您将获得 0 分。如果您的代理至少获胜 5 次，您将获得 1 分；如果您的代理赢得全部 10 场比赛，您将获得 2 分。如果您的代理的平均得分高于 500 分，您将额外获得 1 分；如果高于 1000 分，您将额外获得 2 分。您可以在这些条件下测试您的代理。

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

To run it without graphics, use:

要以不显示图形的方式运行程序，请使用：

```
python autograder.py -q q1 --no-graphics
```

Don't spend too much time on this question, though, as the meat of the project lies ahead.

不过，不要在这个问题上花费太多时间，因为项目的真正重点还在后面。

Q2 (5 pts): Minimax

Q2（5 分）：极小极大值

Now you will write an adversarial search agent in the provided `MinimaxAgent` class stub in `multiAgents.py`. Your minimax agent should work with any number of ghosts, so you'll have to write an algorithm that is slightly more general than what you've previously seen in lecture. In particular, your minimax tree will have multiple min layers (one for each ghost) for every max layer.

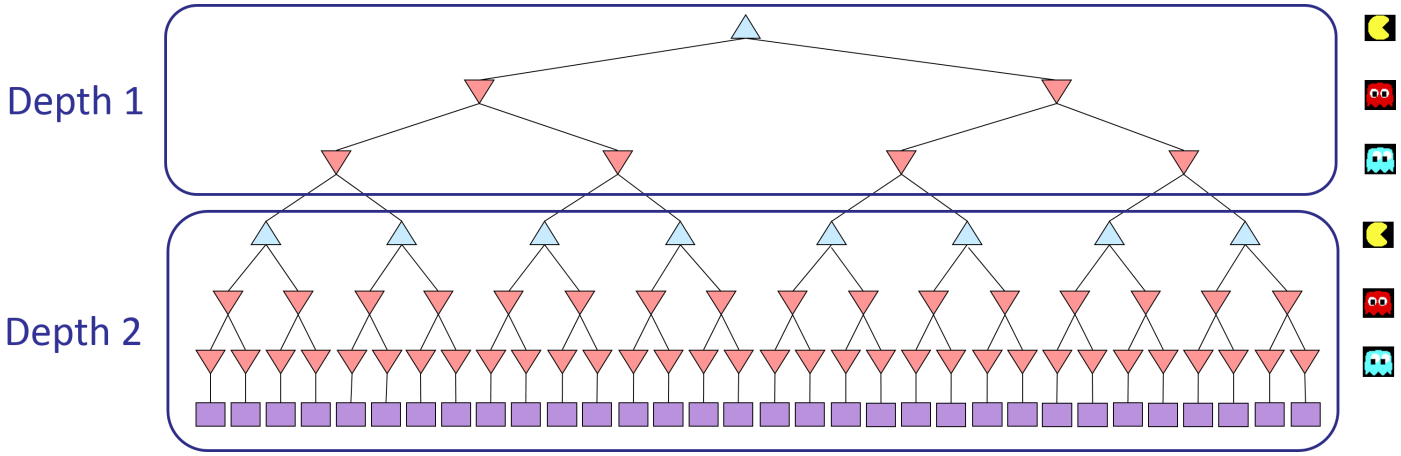
现在，你需要在 `multiAgents.py` 文件中提供的 `MinimaxAgent` 类存根中编写一个对抗搜索代理。你的 minimax 代理应该能够处理任意数量的幽灵，因此你需要编写一个比之前课堂上看到的算法更通用的算法。具体来说，你的 minimax 树对于每个 max 层都将有多个 min 层（每个幽灵对应一个 min 层）。

Your code should also expand the game tree to an arbitrary depth. Score the leaves of your minimax tree with the supplied `self.evaluationFunction`, which defaults to `scoreEvaluationFunction`. `MinimaxAgent` extends `MultiAgentSearchAgent`, which gives access to `self.depth` and `self.evaluationFunction`. Make sure your minimax code makes reference to these two variables where appropriate as these variables are populated in response to command line options. (Do not harcode calls to `scoreEvaluationFunction` since we will be writing a better evaluation function later in the project.)

你的代码还应该将博弈树扩展到任意深度。使用提供的 `self.evaluationFunction` 对你的极小极大树的叶子节点进行评分，该函数默认为 `scoreEvaluationFunction` 继承自 `MinimaxAgent` ，因此可以访问 `self.depth` 和 `self.evaluationFunction` 。确保你的极小极大 `MultiAgentSearchAgent` 在适当的地方引用这两个变量，因为这些变量会根据命令行选项进行填充。（不要硬编码 `scoreEvaluationFunction` 的调用，因为我们将在项目的后续部分编写一个更好的评估函数。）

Important: A single search ply is considered to be one Pacman move and all the ghosts' responses, so depth 2 search will involve Pacman and each ghost moving two times (see diagram below).

重要提示：一次搜索操作是指吃豆人移动一次以及所有幽灵的响应，因此深度 2 搜索将涉及吃豆人和每个幽灵移动两次（见下图）。



Grading: We will be checking your code to determine whether it explores the correct number of game states. This is the only reliable way to detect some very subtle bugs in implementations of minimax. As a result, the autograder will be very picky about how many times you call `GameState.generateSuccessor`. If you call it any more or less than necessary, the autograder will complain. To test and debug your code, run

评分：我们将检查您的代码，以确定它是否探索了正确数量的游戏状态。这是检测极小极大算法实现中一些非常细微的错误的唯一可靠方法。因此，自动评分器会非常严格地检查您调用 `GameState.generateSuccessor` 次数。如果您调用次数过多或过少，自动评分器都会发出警告。要测试和调试您的代码，请运行

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

This will show what your algorithm does on a number of small trees, as well as a pacman game. To run it without graphics, use:

这将展示你的算法在一些小型树以及一个吃豆人游戏上的表现。要以不显示图形的方式运行，请使用：

```
python autograder.py -q q2 --no-graphics
```

Hints and Observations

提示和观察

- Implement the algorithm recursively using helper function(s).

  使用辅助函数递归地实现该算法。

- The correct implementation of minimax will lead to Pacman losing the game in some tests. This is not a problem: as it is correct behaviour, it will pass the tests.

  正确实现极小极大算法会导致吃豆人在某些测试中输掉游戏。这并非问题：因为这是游戏的正确行为，所以测试仍然会通过。

- The evaluation function for the Pacman test in this part is already written (`self.evaluationFunction`). You shouldn't change this function, but recognize that now we're evaluating states rather than actions, as we were for the reflex agent. Look-ahead agents evaluate future states whereas reflex agents evaluate actions from the current state.

  本部分中 Pacman 测试的评估函数已经编写完成（`self.evaluationFunction`）。您不应该修改此函数，但请注意，现在我们评估的是状态而不是动作，这与反射型智能体的情况不同。前瞻型智能体评估的是未来的状态，而反射型智能体则根据当前状态评估动作。

- The minimax values of the initial state in the `minimaxClassic` layout are 9, 8, 7, -492 for depths 1, 2, 3 and 4 respectively. Note that your minimax agent will often win (665/1000 games for us) despite the dire prediction of depth 4 minimax.

  在 `minimaxClassic` 布局中，初始状态的极小极大值在深度分别为 1、2、3 和 4 时分别为 9、8、7 和 -492。值得注意的是，尽管深度为 4 的极小极大预测结果非常糟糕，但你的极小极大智能体通常仍能获胜（我们每 1000 局游戏中赢了 665 局）。

  ```
  python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
  ```

- Pacman is always agent 0, and the agents move in order of increasing agent index.

  吃豆人始终是 0 号智能体，智能体按智能体索引递增的顺序移动。

- All states in minimax should be `GameStates`, either passed in to `getAction` or generated via `GameState.generateSuccessor`. In this project, you will not be abstracting to simplified states.

  极小极大算法中的所有状态都应该是 `GameStates` ，要么通过 `getAction` 传递，要么通过 `GameState.generateSuccessor` 生成。在这个项目中，你不会将其抽象为简化状态。

- On larger boards such as `openClassic` and `mediumClassic` (the default), you'll find Pacman to be good at not dying, but quite bad at winning. He'll often thrash around without making progress. He might even thrash around right next to a dot without eating it because he doesn't know where he'd go after eating that dot. Don't worry if you see this behavior, question 5 will clean up

all of these issues.

在像 `openClassic` 和 `mediumClassic` （默认设置）这样的大型棋盘上，你会发现吃豆人虽然不容易死，但却很难赢。他经常会原地打转却毫无进展。他甚至可能在某个豆子旁边转来转去却不吃掉它，因为他不知道吃掉那个豆子之后该往哪里走。如果你遇到这种情况不用担心，第五题会解决所有这些问题。

- When Pacman believes that his death is unavoidable, he will try to end the game as soon as possible because of the constant penalty for living. Sometimes, this is the wrong thing to do with random ghosts, but minimax agents always assume the worst:

  当吃豆人认为自己必死无疑时，他会尽快结束游戏，因为活着会受到持续的惩罚。有时，对付随机出现的幽灵时，这样做是错误的，但极小极大算法总是会假设最坏的情况：

  ```
  python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
  ```

  Make sure you understand why Pacman rushes the closest ghost in this case.

  务必理解为什么吃豆人会在这种情况下冲向最近的幽灵。

Q3 (5 pts): Alpha-Beta Pruning
Q3（5 分）：Alpha-Beta 剪枝

Make a new agent that uses alpha-beta pruning to more efficiently explore the minimax tree, in `AlphaBetaAgent`. Again, your algorithm will be slightly more general than the pseudocode from lecture, so part of the challenge is to extend the alpha-beta pruning logic appropriately to multiple minimizer agents.

在 `AlphaBetaAgent` 中创建一个新的智能体，它使用 alpha-beta 剪枝来更有效地探索极小极大树。同样，你的算法会比课堂上的伪代码更通用一些，因此部分挑战在于如何将 alpha-beta 剪枝逻辑适当地扩展到多个极小化智能体。

You should see a speed-up (perhaps depth 3 alpha-beta will run as fast as depth 2 minimax). Ideally, depth 3 on `smallClassic` should run in just a few seconds per move or faster.

你应该会看到速度提升（或许深度 3 的 alpha-beta 算法运行速度能达到深度 2 的 minimax 算法的水平）。理想情况下，在 `smallClassic` 地图上，深度 3 的算法每步应该只需几秒钟甚至更快。

```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

The `AlphaBetaAgent` minimax values should be identical to the `MinimaxAgent` minimax values, although the actions it selects can vary because of different tie-breaking behavior. Again, the minimax values of the initial state in the `minimaxClassic` layout are 9, 8, 7 and -492 for depths 1, 2, 3 and 4 respectively.

`AlphaBetaAgent` 极小极大值应该与 `MinimaxAgent` 极小极大值相同，但由于平局打破机制不同，它选择的操作可能会有所不同。同样，在 `minimaxClassic` 布局中，初始状态的极小极大值在深度 1、2、3 和 4 时分别为 9、8、7 和 -492。

Grading: Because we check your code to determine whether it explores the correct number of states, it is important that you perform alpha-beta pruning without reordering children. In other words, successor states should always be processed in the order returned by `GameState.getLegalActions`. Again, do not call `GameState.generateSuccessor` more than necessary.

评分： 由于我们会检查您的代码以确定其是否探索了正确数量的状态，因此请务必在执行 alpha-beta 剪枝时不要对子状态进行重新排序。换句话说，后继状态应始终按照 `GameState.getLegalActions` 返回的顺序进行处理。此外，请勿过度调用 `GameState.generateSuccessor` 。

You must not prune on equality in order to match the set of states explored by our autograder. (Indeed, alternatively, but incompatible with our autograder, would be to also allow for pruning on equality and invoke alpha-beta once on each child of the root node, but this will not match the autograder.)

为了与我们的自动评分器探索的状态集相匹配，您不得基于相等性进行剪枝。 （实际上，另一种方法是允许基于相等性进行剪枝，并对根节点的每个子节点调用一次 alpha-beta 算法，但这与我们的自动评分器不兼容。）

The pseudo-code below represents the algorithm you should implement for this question.

下面这段伪代码表示了你应该为这个问题实现的算法。

Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v > β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v < α return v
        β = min(β, v)
    return v
```

To test and debug your code, run

要测试和调试您的代码，请运行

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

This will show what your algorithm does on a number of small trees, as well as a pacman game. To run it without graphics, use:

这将展示你的算法在一些小型树以及一个吃豆人游戏上的表现。要以不显示图形的方式运行，请使用：

```
python autograder.py -q q3 --no-graphics
```

The correct implementation of alpha-beta pruning will lead to Pacman losing some of the tests. This is not a problem: as it is correct behaviour, it will pass the tests.

正确实现 alpha-beta 剪枝会导致 Pacman 程序在某些测试中失败。但这并非问题：因为这是程序的正常行为，所以它最终会通过所有测试。

Q4 (5 pts): Expectimax
第四题（5 分）：期望最大值

Minimax and alpha-beta are great, but they both assume that you are playing against an adversary who makes optimal decisions. As anyone who has ever won tic-tac-toe can tell you, this is not always the case. In this question you will implement the `ExpectimaxAgent`, which is useful for modeling probabilistic behavior of agents who may make suboptimal choices.

极小极大算法和 alpha-beta 算法都很棒，但它们都假设你的对手会做出最优决策。任何玩过井字棋的人都知道，这种情况并非总是如此。本题中，你将实现 `ExpectimaxAgent` ，它适用于模拟可能做出次优选择的智能体的概率行为。

As with the search and problems yet to be covered in this class, the beauty of these algorithms is their general applicability. To expedite your own development, we've supplied some test cases based on generic trees. You can debug your implementation on small the game trees using the command:

与本课程中尚未涉及的搜索和问题一样，这些算法的魅力在于它们的通用性。为了加快您的开发速度，我们提供了一些基于通用树的测试用例。您可以使用以下命令在小型博弈树上调试您的实现：

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

Debugging on these small and manageable test cases is recommended and will help you to find bugs quickly.

建议在这些小型且易于管理的测试用例上进行调试，这将有助于您快速找到错误。

Once your algorithm is working on small trees, you can observe its success in Pacman. Random ghosts are of course not optimal minimax agents, and so modeling them with minimax search may not be appropriate. `ExpectimaxAgent` will no longer take the min over all ghost actions, but the expectation according to your agent's model of how the ghosts act. To simplify your code, assume you will only be running against an adversary which chooses amongst their `getLegalActions` uniformly at random.

一旦你的算法能够处理小树，你就可以在吃豆人游戏中观察到它的效果。随机生成 `ExpectimaxAgent` 幽灵当然不是最优的极小极大智能体，因此用极小极大搜索来建模它们可能并不合适。ExpectimaxAgent 不再取所有幽灵动作的最小值，而是根据你的智能体对幽灵行为的模型来计算期望值。为了简化你的代码，假设你只会遇到一个对手，该对手会从其 `getLegalActions` 中随机均匀地选择一个动作。

To see how the ExpectimaxAgent behaves in Pacman, run:

要查看 ExpectimaxAgent 在 Pacman 中的行为，请运行：

```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

You should now observe a more cavalier approach in close quarters with ghosts. In particular, if Pacman perceives that he could be trapped but might escape to grab a few more pieces of food, he'll at least try. Investigate the results of these two scenarios:

现在，你应该观察到吃豆人在近距离面对幽灵时采取更加随意的态度。特别是，如果吃豆人意识到自己可能被困住，但或许有机会逃脱去多吃几块食物，他至少会尝试一下。请调查以下两种情况的结果：

```
python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
```

```
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
```

You should find that your `ExpectimaxAgent` wins about half the time, while your `AlphaBetaAgent` always loses. Make sure you understand why the behavior here differs from the minimax case.

你会发现， `ExpectimaxAgent` 大约有一半的概率获胜，而 `AlphaBetaAgent` 则总是失败。务必理解为什么这里的行为与 minimax 的情况不同。

The correct implementation of expectimax will lead to Pacman losing some of the tests. This is not a problem: as it is correct behaviour, it will pass the tests.

正确实现 expectimax 函数会导致 Pacman 在某些测试中失败。但这并非问题：因为这是程序的正确行为，所以它最终会通过所有测试。

Q5 (6 pts): Evaluation Function

Q5（6 分）：评估函数

Write a better evaluation function for Pacman in the provided function `betterEvaluationFunction`. The evaluation function should evaluate states, rather than actions like your reflex agent evaluation function did. With depth 2 search, your evaluation function should clear the `smallClassic` layout with one random ghost more than half the time and still run at a reasonable rate (to get full credit,

Pacman should be averaging around 1000 points when he's winning).

请在提供的 `betterEvaluationFunction` 函数中为 Pacman 编写一个更好的评估函数。该评估函数应该评估状态，而不是像你的反射代理评估函数那样评估动作。使用深度为 2 的搜索，你的评估函数应该在超过一半的时间内清除包含一个随机幽灵的 `smallClassic` 布局，并且仍然保持合理的运行速度（为了获得满分，Pacman 在获胜时平均得分应在 1000 分左右）。

Grading: the autograder will run your agent on the smallClassic layout 10 times. We will assign points to your evaluation function in the following way:

评分：自动评分器将使用 smallClassic 布局运行您的代理 10 次。我们将按以下方式为您的评估函数分配分数：

- If you win at least once without timing out the autograder, you receive 1 points. Any agent not satisfying these criteria will receive 0 points.
- +1 for winning at least 5 times, +2 for winning all 10 times
- +1 for an average score of at least 500, +2 for an average score of at least 1000 (including scores on lost games)
- +1 if your games take on average less than 30 seconds on the autograder machine, when run with `--no-graphics`.
- The additional points for average score and computation time will only be awarded if you win at least 5 times.
- Please do not copy any files from Project 1, as it will not pass the autograder on Gradescope.

You can try your agent out under these conditions with

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

To run it without graphics, use:

```
python autograder.py -q q5 --no-graphics
```

Submission

In order to submit your project upload the Python files you edited. For instance, use Gradescope's upload on all `.py` files in the project folder.