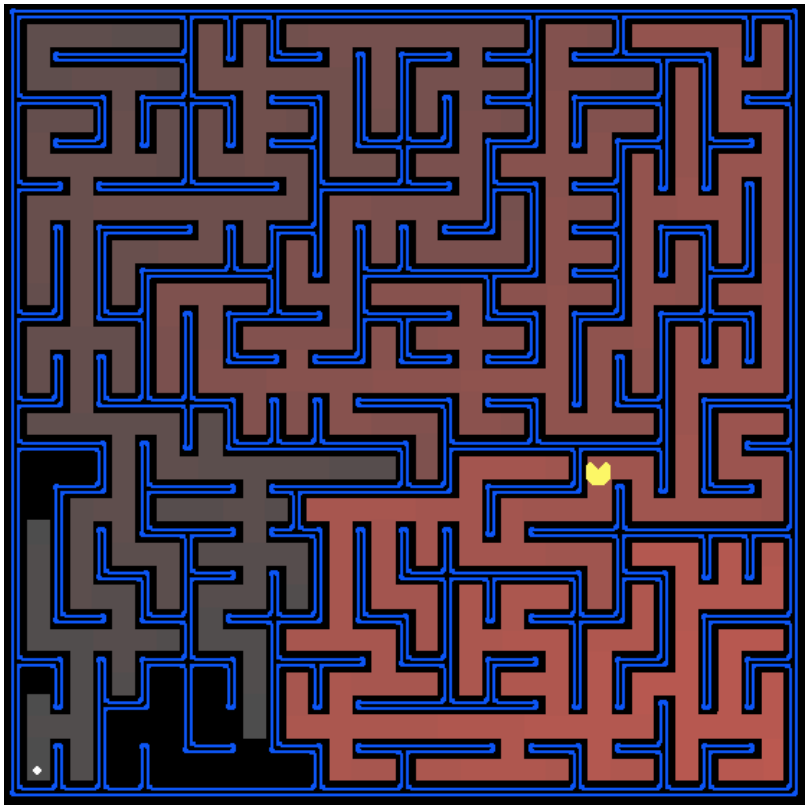
Project 1: Search
项目 1：搜索

Due: **Friday, September 12, 11:59 PM PT**.

截止日期： **9 月 12 日星期五晚上 11:59（太平洋时间）** 。



All those colored walls,

那些彩色的墙壁，

Mazes give Pacman the blues,

迷宫让吃豆人感到沮丧，

So teach him to search.

所以要教他如何寻找。

TABLE OF CONTENTS

目录

Introduction 介绍

In this project, your Pacman agent will find paths through his maze world, both to reach a particular location and to collect food efficiently. You will build general search algorithms and apply them to

Pacman scenarios.

在这个项目中，你的吃豆人智能体需要在迷宫世界中找到路径，既要到达特定地点，又要高效地收集食物。你将构建通用的搜索算法，并将其应用于吃豆人游戏场景。

As in Project 0, this project includes an autograder for you to grade your answers on your machine. This can be run with the command:

与项目 0 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/q7/food_heuristic_1
```

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 consists of several Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore. You can download all the code and supporting files as a search.zip.

本项目的代码包含多个 Python 文件，其中一些文件需要您阅读并理解才能完成作业，而另一些文件则可以忽略。您可以将所有代码和相关文件下载为 search.zip 文件。

Files you'll edit: 您将编辑的文件：	
`search.py`	Where all of your search algorithms will reside. 所有搜索算法都将位于此处。
`searchAgents.py`	Where all of your search-based agents will reside. 所有基于搜索的代理都将位于此处。
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 describes a Pacman GameState type, which you use in this project. 运行吃豆人游戏的主文件。此文件描述了一种吃豆人游戏状态类型，您将在本项目中使用它。
`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. 用于实现搜索算法的实用数据结构。
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
`searchTestClasses.py`	Project 1 specific autograding test classes 项目 1 特定自动评分测试课程

Files to Edit and Submit: You will fill in portions of `search.py` and `searchAgents.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 `search.py` 和 `searchAgents.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）。请勿更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： 系统将自动评估您的代码的技术正确性。请勿更改代码中任何提供的函数或类的名称，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比对，检查是否存在逻辑冗余。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：

- Uninformed Search: Video 2, Note 1.2-1.3

  不知情搜索： 视频 2 ， 注释 1.2-1.3
- A* Search and Heuristics: Video 3, Note 1.4-1.5

  A* 搜索和启发式算法： 视频 3 ， 注释 1.4-1.5

Welcome to Pacman
欢迎来到吃豆人世界

After downloading the code, unzipping it, and changing to the directory, you should be able to play a game of Pacman by typing the following at the command line:

下载代码、解压缩并切换到相应目录后，您应该可以通过在命令行输入以下命令来玩吃豆人游戏：

```
python pacman.py
```

Pacman lives in a shiny blue world of twisting corridors and tasty round treats. Navigating this world efficiently will be Pacman's first step in mastering his domain.

吃豆人生活在一个闪亮的蓝色世界里，那里有曲折的走廊和美味的圆形糖果。高效地穿梭于这个世界将是吃豆人征服这片领地的第一步。

The simplest agent in `searchAgents.py` is called the `GoWestAgent`, which always goes West (a trivial reflex agent). This agent can occasionally win:

`searchAgents.py` 中最简单的智能体称为 `GoWestAgent` ，它总是向西行进（一个简单的反射智能体）。这个智能体偶尔也能获胜：

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

But, things get ugly for this agent when turning is required:

但是，当需要转变立场时，这位特工的处境就变得糟糕了：

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

If Pacman gets stuck, you can exit the game by typing CTRL-c into your terminal.

如果吃豆人卡住了，你可以通过在终端输入 CTRL-c 来退出游戏。

Soon, your agent will solve not only `tinyMaze`, but any maze you want.

很快，你的智能体不仅能解决 `tinyMaze` ，还能解决你想要的任何迷宫。

Note that `pacman.py` supports a number of options that can each be expressed in a long way (e.g., `--layout`) or a short way (e.g., `-l`). You can see the list of all options and their default values via:

请注意， `pacman.py` 支持许多选项，每个选项都可以用完整格式（例如， `--layout` ）或简短格式（例如， `-l` ）表示。您可以通过以下方式查看所有选项及其默认值的列表：

```
python pacman.py -h
```

Also, all of the commands that appear in this project also appear in `commands.txt`, for easy copying and pasting. In UNIX/Mac OS X, you can even run all these commands in order with `bash commands.txt`.

此外，本项目中出现的所有命令也都列在 `commands.txt` 中，方便复制粘贴。在 UNIX/Mac OS X 系统中，您甚至可以使用 `bash commands.txt` 按顺序运行所有这些命令。

New Syntax 新语法

You may not have seen this syntax before:

你以前可能没见过这种语法：

```
def my_function(a: int, b: Tuple[int, int], c: List[List], d: Any, e: float=1.0):
```

This is annotating the type of the arguments that Python should expect for this function. In the example below, `a` should be an `int` – integer, `b` should be a `tuple` of 2 `int`s, `c` should be a `List` of `Lists` of anything – therefore a 2D array of anything, `d` is essentially the same as not annotated and can by anything, and `e` should be a `float`. `e` is also set to 1.0 if nothing is passed in for it, i.e.:

这用于指定 Python 应该为该函数传入的参数类型。在下面的示例中， `a` 应该是一个整数（ `int` ），`b` 应该是一个包含两个 `int` 的 `tuple` ， `c` 应该是一个任意类型的 `Lists` 的 `List` （即一个二维数组）， `d` 与未指定类型相同，可以是任何值， `e` 应该是一个 `float` ）。如果没有传入任何参数， `e` 值也会被设置为 1.0，例如：

```
my_function(1, (2, 3), [['a', 'b'], [None, my_class], [[]]], ('h', 1))
```

The above call fits the type annotations, and doesn't pass anything in for e. Type annotations are meant to be an adddition to the docstrings to help you know what the functions are working with. Python itself doesn't enforce these. When writing your own functions, it is up to you if you want to annotate your types; they may be helpful to keep organized or not something you want to spend

time on.

上述调用符合类型注解，并且没有为 `e` 传递任何参数。类型注解旨在作为文档字符串的补充，帮助您了解函数正在处理哪些类型的数据。Python 本身并不强制要求使用类型注解。编写自定义函数时，是否为类型添加注解完全取决于您；它们可能有助于保持代码的条理性和组织性，也可能只是您不想花费时间在这上面。

Q1 (3 pts): Finding a Fixed Food Dot using Depth First Search
Q1（3 分）：使用深度优先搜索查找固定食物点

In `searchAgents.py`, you'll find a fully implemented `SearchAgent`, which plans out a path through Pacman's world and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented – that's your job.

在 `searchAgents.py` 中，你会找到一个完全实现的 `SearchAgent` ，它会规划出一条穿越吃豆人世界的路径，然后逐步执行该路径。用于制定路径的搜索算法尚未实现——那是你的工作。

First, test that the `SearchAgent` is working correctly by running:

首先，运行以下命令测试 `SearchAgent` 是否正常工作：

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

The command above tells the `SearchAgent` to use `tinyMazeSearch` as its search algorithm, which is implemented in `search.py`. Pacman should navigate the maze successfully.

上述命令指示 `SearchAgent` 使用 `tinyMazeSearch` 作为其搜索算法，该算法在 `search.py` 中实现。吃豆人应该能够成功穿过迷宫。

Now it's time to write full-fledged generic search functions to help Pacman plan routes! Pseudocode for the search algorithms you'll write can be found in the lecture slides. Remember that a search node must contain not only a state but also the information necessary to reconstruct the path (plan) which gets to that state.

现在是时候编写完整的通用搜索函数来帮助吃豆人规划路线了！你将要编写的搜索算法的伪代码可以在讲义中找到。记住，搜索节点不仅必须包含状态，还必须包含重建到达该状态的路径（计划）所需的信息。

Important note: All of your search functions need to return a list of actions that will lead the agent from the start to the goal. These actions all have to be legal moves (valid directions, no moving through walls).

重要提示：所有搜索函数都需要返回一个操作列表，引导智能体从起点到达终点。这些操作都必须是合法的移动（有效的方向，不能穿墙）。

Important note: Make sure to use the `Stack`, `Queue` and `PriorityQueue` data structures provided to you in `util.py`! These data structure implementations have particular properties which are required for compatibility with the autograder.

重要提示：请务必使用 `util.py` 中提供的 `Stack` 、 `Queue` 和 `PriorityQueue` 数据结构！这些数据结构实现具有一些特定属性，这些属性是与自动评分器兼容所必需的。

Hint: Each algorithm is very similar. Algorithms for DFS, BFS, UCS, and A* differ only in the details of how the fringe is managed. So, concentrate on getting DFS right and the rest should be relatively straightforward. Indeed, one possible implementation requires only a single generic search method which is configured with an algorithm-specific queuing strategy. (Your implementation need not be of this form to receive full credit).

提示：每种算法都非常相似。DFS、BFS、UCS 和 A* 算法的区别仅在于边缘区域的管理细节。因此，集中精力确保 DFS 算法正确，其余算法应该相对容易实现。实际上，一种可能的实现方式只需要一个通用的搜索方法，并配置一个特定于该算法的排队策略。（你的实现不必采用这种形式也能获得满分）。

Implement the depth-first search (DFS) algorithm in the `depthFirstSearch` function in `search.py`. To make your algorithm complete, write the graph search version of DFS, which avoids expanding any already visited states.

在 `search.py` 中的 `depthFirstSearch` 函数中实现深度优先搜索 (DFS) 算法。为了完善你的算法，请编写 DFS 的图搜索版本，该版本避免扩展任何已访问过的状态。

Your code should quickly find a solution for:

你的代码应该能够快速找到以下问题的解决方案：

```
python pacman.py -l tinyMaze -p SearchAgent
python pacman.py -l mediumMaze -p SearchAgent
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration). Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal?

吃豆人游戏界面会显示已探索过的方格，以及探索顺序（颜色越亮的红色表示探索时间越早）。这个探索顺序符合你的预期吗？吃豆人在到达终点前真的会经过所有已探索过的方格吗？

Hint: If you use a `Stack` as your data structure, the solution found by your DFS algorithm for `mediumMaze` should have a length of 130 (provided you push successors onto the fringe in the order provided by getSuccessors; you might get 246 if you push them in the reverse order). Is this a least cost solution? If not, think about what depth-first search is doing wrong.

提示：如果你使用 `Stack` 作为数据结构，那么对于 `mediumMaze` ，你的深度优先搜索算法找到的解的长度应该是 130（前提是你按照 `getSuccessors` 函数提供的顺序将后继元素压入栈边缘；如果顺序相反，长度可能会达到 246）。这是最小成本解吗？如果不是，请思考一下深度优先搜索哪里出了问题。

Grading: Please run the below command to see if your implementation passes all the autograder test cases.

评分：请运行以下命令，查看您的实现是否通过了所有自动评分测试用例。

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

Q2 (3 pts): Breadth First Search
Q2（3 分）：广度优先搜索

Implement the breadth-first search (BFS) algorithm in the `breadthFirstSearch` function in `search.py`. Again, write a graph search algorithm that avoids expanding any already visited states. Test your code the same way you did for depth-first search.

在 `search.py` 中的 `breadthFirstSearch` 函数中实现广度优先搜索 (BFS) 算法。同样，编写一个图搜索算法，避免扩展任何已访问过的状态。用与深度优先搜索相同的方法测试你的代码。

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```

Does BFS find a least cost solution? If not, check your implementation.

BFS 能找到成本最低的解决方案吗？如果不能，请检查你的实现。

Hint: If Pacman moves too slowly for you, try the option –frameTime 0.

提示：如果吃豆人移动速度太慢，请尝试使用 -frameTime 0 选项。

Note: If you've written your search code generically, your code should work equally well for the eight-puzzle search problem without any changes.

注意：如果您编写的搜索代码是通用的，那么您的代码无需任何更改即可同样适用于八谜搜索问题。

```
python eightpuzzle.py
```

Grading: Please run the below command to see if your implementation passes all the autograder test cases.

评分：请运行以下命令，查看您的实现是否通过了所有自动评分测试用例。

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

Q3 (3 pts): Varying the Cost Function

Q3（3 分）：改变成本函数

While BFS will find a fewest-actions path to the goal, we might want to find paths that are "best" in other senses. Consider `mediumDottedMaze` and `mediumScaryMaze`.

虽然广度优先搜索（BFS）可以找到到达目标所需的最少动作路径，但我们可能希望找到其他意义上的"最佳"路径。例如，可以考虑 `mediumDottedMaze` 和 `mediumScaryMaze` 。

By changing the cost function, we can encourage Pacman to find different paths. For example, we can charge more for dangerous steps in ghost-ridden areas or less for steps in food-rich areas, and a rational Pacman agent should adjust its behavior in response.

通过改变成本函数，我们可以鼓励吃豆人寻找不同的路径。例如，我们可以对在幽灵出没区域行走的危险路径收取更高的成本，而对在食物丰富的区域行走的路径收取更低的成本，理性的吃豆人智能体应该会据此调整自己的行为。

Implement the uniform-cost graph search algorithm in the `uniformCostSearch` function in `search.py`. We encourage you to look through `util.py` for some data structures that may be useful in your implementation. You should now observe successful behavior in all three of the following layouts, where the agents below are all UCS agents that differ only in the cost function they use (the agents and cost functions are written for you):

在 `search.py` 中的 `uniformCostSearch` 函数中实现统一代价图搜索算法。我们建议您查阅 `util.py` ，其中包含一些可能对您的实现有用的数据结构。现在，您应该能够在以下三种布局中观察到成功的运行结果，其中所有智能体均为 UCS 智能体，它们之间的区别仅在于使用的成本函数（智能体和成本函数已为您编写好）：

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

Note: You should get very low and very high path costs for the `StayEastSearchAgent` and `StayWestSearchAgent` respectively, due to their exponential cost functions (see `searchAgents.py` for details).

注意： 由于 `StayEastSearchAgent` 和 `StayWestSearchAgent` 成本函数呈指数增长（详情请参阅 `searchAgents.py` ），因此 StayEastSearchAgent 和 StayWestSearchAgent 的路径成本应该分别非常低和非常高。

Grading: Please run the below command to see if your implementation passes all the autograder test cases.

评分：请运行以下命令，查看您的实现是否通过了所有自动评分测试用例。

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

Q4 (3 pts): A* search
第四题（3 分）：A*搜索

Implement A* graph search in the empty function `aStarSearch` in `search.py`. A* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The `nullHeuristic` heuristic function in

`search.py` is a trivial example.

在 `search.py` 文件中的空函数 `aStarSearch` 中实现 A* 图搜索。A* 算法接受一个启发式函数作为参数。启发式函数接受两个参数：搜索问题中的一个状态（主要参数）和问题本身（用于参考信息）。search.py `search.py` 中的 `nullHeuristic` 启发式函数是一个简单的示例。

You can test your A* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as `manhattanHeuristic` in `searchAgents.py`).

您可以使用曼哈顿距离启发式算法（已在 `searchAgents.py` 中实现为 `manhattanHeuristic` ）在寻找穿过迷宫到达固定位置的路径的原始问题上测试您的 A* 实现。

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```

You should see that A* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded in our implementation, but ties in priority may make your numbers differ slightly). What happens on `openMaze` for the various search strategies?

你会发现 A* 算法找到最优解的速度比均匀代价搜索略快（在我们的实现中，A* 算法大约扩展了 549 个搜索节点，而均匀代价搜索大约扩展了 620 个搜索节点，但优先级相同的情况可能会导致你的结果略有不同）。在 `openMaze` 上，各种搜索策略的表现如何?

Grading: Please run the below command to see if your implementation passes all the autograder test cases.

评分：请运行以下命令，查看您的实现是否通过了所有自动评分测试用例。

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

Q5 (3 pts): Finding All the Corners
Q5（3 分）：找出所有角点

The real power of A* will only be apparent with a more challenging search problem. Now, it's time to formulate a new problem and design a heuristic for it.

A*算法的真正威力只有在更具挑战性的搜索问题上才能体现出来。现在，是时候提出一个新的问题并为其设计一个启发式算法了。

In corner mazes, there are four dots, one in each corner. Our new search problem is to find the shortest path through the maze that touches all four corners (whether the maze actually has food there or not). Note that for some mazes like `tinyCorners`, the shortest path does not always go to the closest food first! Hint: the shortest path through `tinyCorners` takes 28 steps.

在角落迷宫中，每个角落各有一个点。我们新的搜索问题是找到一条穿过迷宫且经过所有四个角落的最短路径（无论这些角落是否有食物）。请注意，对于某些迷宫，例如 `tinyCorners` ，最短路径并非总是先到达最近的食物！提示： `tinyCorners` 最短路径需要 28 步。

Note: Make sure to complete Question 2 before working on Question 5, because Question 5 builds upon your answer for Question 2.

注意： 请务必在解答第 5 题之前完成第 2 题，因为第 5 题是建立在您对第 2 题的回答之上的。

Implement the `CornersProblem` search problem in `searchAgents.py`. You will need to choose a state representation that encodes all the information necessary to detect whether all four corners have been reached. Now, your search agent should solve:

在 `searchAgents.py` 中实现 `CornersProblem` 搜索问题。你需要选择一个状态表示，该表示能够编码所有必要信息，以便检测是否已到达所有四个角点。现在，你的搜索代理应该能够解决以下问题：

```
python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
```

To receive full credit, you need to define an abstract state representation that does not encode irrelevant information (like the position of ghosts, where extra food is, etc.). In particular, do not use a Pacman `GameState` as a search state. Your code will be very, very slow if you do (and also wrong).

要获得满分，你需要定义一个抽象的状态表示，其中不应包含无关信息（例如幽灵的位置、额外食物的位置等）。尤其要注意的是，不要使用 Pacman `GameState` 作为搜索状态。否则，你的代码运行速度会非常慢（而且也是错误的）。

An instance of the `CornersProblem` class represents an entire search problem, not a particular state. Particular states are returned by the functions you write, and your functions return a data structure of your choosing (e.g. tuple, set, etc.) that represents a state.

`CornersProblem` 类的一个实例代表一个完整的搜索问题，而不是某个特定的状态。特定的状态由你编写的函数返回，而你的函数返回的是你选择的数据结构（例如元组、集合等）来表示一个状态。

Furthermore, while a program is running, remember that many states simultaneously exist, all on the queue of the search algorithm, and they should be independent of each other. In other words, you

should not have only one state for the entire `CornersProblem` object; your class should be able to generate many different states to provide to the search algorithm.

此外，程序运行时，请记住存在多个同时存在的状态，它们都在搜索算法的队列中，并且彼此独立。换句话说，整个 `CornersProblem` 对象不应该只有一个状态；你的类应该能够生成多个不同的状态提供给搜索算法。

Hint 1: The only parts of the game state you need to reference in your implementation are the starting Pacman position and the location of the four corners.

提示 1：在你的实现中，你只需要引用游戏状态的起始吃豆人位置和四个角的位置。

Hint 2: When coding up `getSuccessors`, make sure to add children to your successors list with a cost of 1.

提示 2：在编写 `getSuccessors` 函数时，请确保将成本为 1 的子项添加到后继者列表中。

Our implementation of `breadthFirstSearch` expands just under 2000 search nodes on `mediumCorners`. However, heuristics (used with A* search) can reduce the amount of searching required.

我们实现的 `breadthFirstSearch` 在 `mediumCorners` 集上扩展了近 2000 个搜索节点。然而，启发式算法（与 A* 搜索结合使用）可以减少所需的搜索量。

Grading: Please run the below command to see if your implementation passes all the autograder test cases.

评分：请运行以下命令，查看您的实现是否通过了所有自动评分测试用例。

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

Q6 (3 pts): Corners Problem: Heuristic
Q6（3 分）：角点问题：启发式算法

Note: Make sure to complete Question 4 before working on Question 6, because Question 6 builds upon your answer for Question 4.

注意：请务必在解答第 6 题之前完成第 4 题，因为第 6 题是建立在您对第 4 题的回答之上的。

Implement a non-trivial, consistent heuristic for the `CornersProblem` in `cornersHeuristic`.

在 `cornersHeuristic` 中实现 `CornersProblem` 的非平凡、一致的启发式算法。

```
python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
```

Note: `AStarCornersAgent` is a shortcut for

注： `AStarCornersAgent` 是以下函数的快捷方式：

```
-p SearchAgent -a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic
```

Admissibility vs. Consistency: Remember, heuristics are just functions that take search states and return numbers that estimate the cost to a nearest goal. More effective heuristics will return values closer to the actual goal costs. To be *admissible*, the heuristic values must be lower bounds on the actual shortest path cost to the nearest goal (and non-negative). To be *consistent*, it must additionally hold that if an action has cost c, then taking that action can only cause a drop in heuristic of at most c.

可采纳性与一致性：请记住，启发式函数只是接受搜索状态并返回估计到达最近目标的成本的数值的函数。更有效的启发式函数会返回更接近实际目标成本的值。为了保证*可采纳性*，启发式值必须是到达最近目标的实际最短路径成本的下限（且非负）。为了保证*一致性*，它还必须满足以下条件：如果一个动作的成本为 c，那么采取该动作最多只能使启发式值下降 c。

Remember that admissibility isn't enough to guarantee correctness in graph search – you need the stronger condition of consistency. However, admissible heuristics are usually also consistent, especially if they are derived from problem relaxations. Therefore it is usually easiest to start out by brainstorming admissible heuristics. Once you have an admissible heuristic that works well, you can check whether it is indeed consistent, too. The only way to guarantee consistency is with a proof. However, inconsistency can often be detected by verifying that for each node you expand, its successor nodes are equal or higher in in f-value. Moreover, if UCS and A* ever return paths of different lengths, your heuristic is inconsistent. This stuff is tricky!

记住，在图搜索中，可采纳性并不足以保证正确性——你还需要更强的一致性条件。然而，可采纳的启发式算法通常也是一致的，尤其是在它们源自问题松弛的情况下。因此，最简单的方法通常是先集思广益，提出一些可采纳的启发式算法。一旦你找到了一个效果良好的可采纳启发式算法，你就可以检查它是否也具有一致性。保证一致性的唯一方法是给出证明。但是，通常可以通过验证以下情况来检测不一致性：对于你扩展的每个节点，其后继节点的 f 值是否等于或大于该节点。此外，如果 UCS 和 A* 算法返回的路径长度不同，则你的启发式算法是不一致的。这方面的内容很复杂！

Non-Trivial Heuristics: The trivial heuristics are the ones that return zero everywhere (UCS) and the heuristic which computes the true completion cost. The former won't save you any time, while the latter will timeout the autograder. You want a heuristic which reduces total compute time, though for this assignment the autograder will only check node counts (aside from enforcing a reasonable time limit).

非平凡启发式算法：平凡启发式算法是指处处返回零的算法（UCS）以及计算真实完成成本的启发式算法。前者不会节省任何时间，而后者会导致自动评分器超时。你需要的是一个能够减少总计算时间的启发式算法，尽管对于本次作业，自动评分器只会检查节点计数（除了强制执行合理的时间限制之外）。

Grading: Your heuristic must be a non-trivial non-negative consistent heuristic to receive any points. Make sure that your heuristic returns 0 at every goal state and never returns a negative value. Depending on how few nodes your heuristic expands, you'll be graded:

评分：你的启发式算法必须是非平凡、非负且一致的启发式算法才能得分。请确保你的启发式算法在每个目标状态下都返回 0，并且永远不会返回负值。根据你的启发式算法扩展的节点数，你将获得相应的分数：

Number of nodes expanded **扩展节点数**	**Grade 年级**
more than 2000 超过2000	0/3
at most 2000 最多2000	1/3
at most 1600 最多1600	2/3
at most 1200 最多 1200	3/3

Remember: If your heuristic is inconsistent, you will receive no credit, so be careful!

记住：如果你的启发式方法前后不一致，你将得不到任何分数，所以要小心！

Grading: Please run the below command to see if your implementation passes all the autograder test cases.

评分：请运行以下命令，查看您的实现是否通过了所有自动评分测试用例。

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

Q7 (4 pts): Eating All The Dots
Q7（4 分）：吃掉所有的点

Now we'll solve a hard search problem: eating all the Pacman food in as few steps as possible. For this, we'll need a new search problem definition which formalizes the food-clearing problem: `FoodSearchProblem` in `searchAgents.py` (implemented for you). A solution is defined to be a path that collects all of the food in the Pacman world. For the present project, solutions do not take into account any ghosts or power pellets; solutions only depend on the placement of walls, regular food and Pacman. (Of course ghosts can ruin the execution of a solution! We'll get to that in the next project.) If you have written your general search methods correctly, A* with a null heuristic (equivalent to uniform-cost search) should quickly find an optimal solution to `testSearch` with no code change on your part (total cost of 7).

现在我们将解决一个难题：用尽可能少的步数吃掉吃豆人世界里的所有食物。为此，我们需要一个新的搜索问题定义，它将清除食物的问题形式化： `searchAgents.py` 中的 `FoodSearchProblem` （已为您实现）。解决方案被定义为收集吃豆人世界中所有食物的路径。在本项目中，解决方案不考虑任何幽灵或能量豆；解决方案仅取决于墙壁、普通食物和吃豆人的位置。（当然，幽灵可能会破坏解决方案的执行！我们将在下一个项目中讨论这个问题。）如果您正确编写了通用搜索方法，那么使用零启发式函数的 A* 算法（等价于均匀成本搜索）应该可以快速找到 `testSearch` 的最优解，而无需您修改任何代码（总成本为 7）。

```
python pacman.py -l testSearch -p AStarFoodSearchAgent
```

Note: `AStarFoodSearchAgent` is a shortcut for

注： `AStarFoodSearchAgent` 是以下函数的快捷方式：

```
-p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic
```

You should find that UCS starts to slow down even for the seemingly simple `tinySearch`. As a reference, our implementation takes 2.5 seconds to find a path of length 27 after expanding 5057 search nodes.

你会发现，即使是看似简单的 `tinySearch` UCS 的性能也会开始下降。作为参考，我们的实现扩展了 5057 个搜索节点后，需要 2.5 秒才能找到长度为 27 的路径。

Note: Make sure to complete Question 4 before working on Question 7, because Question 7 builds upon your answer for Question 4.

注意：请务必在解答第 7 题之前完成第 4 题，因为第 7 题是建立在您对第 4 题的回答之上的。

Fill in `foodHeuristic` in `searchAgents.py` with a *consistent* heuristic for the `FoodSearchProblem`. Try your agent on the `trickySearch` board:

在 `searchAgents.py` 中，为 `FoodSearchProblem` 问题填写 `foodHeuristic` ，并使其*具有一致的*启发式算法。然后，在 `trickySearch` 板上测试你的智能体：

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
```

Our UCS agent finds the optimal solution in about 13 seconds, exploring over 16,000 nodes.

我们的 UCS 代理在大约 13 秒内找到了最优解，探索了超过 16,000 个节点。

Any non-trivial non-negative consistent heuristic will receive 1 point. Make sure that your heuristic returns 0 at every goal state and never returns a negative value. Depending on how few nodes your heuristic expands, you'll get additional points:

任何非平凡、非负且一致的启发式算法都将获得 1 分。请确保您的启发式算法在每个目标状态下都返回 0，并且永远不会返回负值。根据启发式算法扩展的节点数，您将获得额外分数：

Number of nodes expanded **扩展节点数**	**Grade 年级**
more than 15000 超过15000	1/4
at most 15000 最多 15000	2/4
at most 12000 最多 12000	3/4
at most 9000 最多9000	4/4 (full credit; medium) 4/4（满分；中等难度）
at most 7000 最多7000	5/4 (optional extra credit; hard) 5/4（可选加分；难度较高）

Remember: If your heuristic is inconsistent, you will receive no credit, so be careful! Can you solve `mediumSearch` in a short time? If so, we're either very, very impressed, or your heuristic is inconsistent.

记住：如果你的启发式方法前后不一致，你将得不到任何分数，所以务必小心！你能在短时间内解决 `mediumSearch` 吗？如果可以，我们要么非常非常佩服你，要么你的启发式方法前后不一致。

Grading: Please run the below command to see if your implementation passes all the autograder test cases.

评分：请运行以下命令，查看您的实现是否通过了所有自动评分测试用例。

```
python autograder.py -q q7
```

Q8 (3 pts): Suboptimal Search
Q8（3 分）：次优搜索

Sometimes, even with A* and a good heuristic, finding the optimal path through all the dots is hard. In these cases, we'd still like to find a reasonably good path, quickly. In this section, you'll write an agent that always greedily eats the closest dot. `ClosestDotSearchAgent` is implemented for you in `searchAgents.py`, but it's missing a key function that finds a path to the closest dot.

有时，即使使用 A* 算法和良好的启发式函数，找到穿过所有点的最优路径也很困难。在这种情况下，我们仍然希望快速找到一条相当不错的路径。在本节中，您将编写一个始终贪婪地吞噬最近点的 `ClosestDotSearchAgent` 体。`closestDotSearchAgent` 已在 `searchAgents.py` 中实现，但它缺少一个用于找到通往最近点的路径的关键函数。

Implement the function `findPathToClosestDot` in `searchAgents.py`. Our agent solves this maze (suboptimally!) in under a second with a path cost of 350:

```
python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
```

Hint: The quickest way to complete `findPathToClosestDot` is to fill in the `AnyFoodSearchProblem`, which is missing its goal test. Then, solve that problem with an appropriate search function. The solution should be very short!

Your `ClosestDotSearchAgent` won't always find the shortest possible path through the maze. Make sure you understand why and try to come up with a small example where repeatedly going to the closest

dot does not result in finding the shortest path for eating all the dots.

Grading: Please run the below command to see if your implementation passes all the autograder test cases.

```
python autograder.py -q q8
```

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.