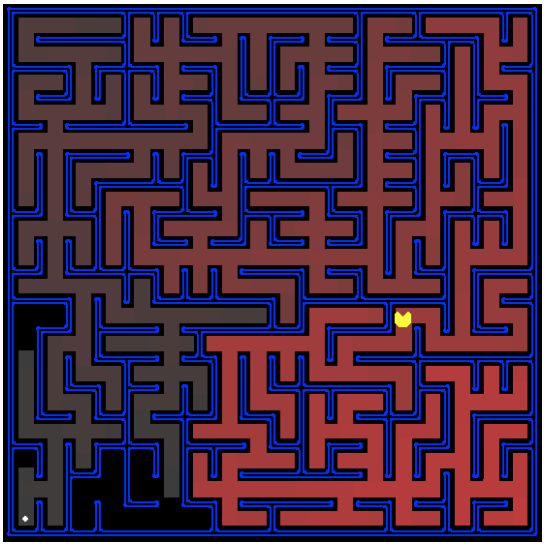
03 Search I



本节实验目标：熟悉吃豆人任务环境，练习实现无信息搜索策略

报告说明（重要）：本部分“搜索”包含两周内容，本节课为第一周内容，请留意该文档目录中标有“报告应实现任务[分值]”的部分，请在实现后及时保存文件和运行结果截图，在课程报告中提交。

介绍

在本节实验课中，你的Pacman将在他的迷宫世界中找到路径，既可以到达特定位置，又可以有效地收集食物。你将构建通用搜索算法并将其应用于 Pacman 场景。

与上节实验课一样，该项目包括一个自动评分器，供你对答案进行评分。这可以在命令终端（路径为该项目文件夹路径）使用以下命令运行：

```
python autograder.py
```

该项目的代码由几个 Python 文件组成，其中一些你需要阅读和理解才能完成作业，而其中一些你可以忽略。你可以在这个对分易-课程资源-实验课课件，或从机房上电脑桌面（当堂课上） 下载所有代码和支持文件。

你要编辑的文件:	
search.py	你的搜索算法所在地
searchAgents.py	你的搜索Agent所在地

你可能想看一下的文件:	
pacman.py	运行 Pacman 游戏的主文件。这个文件描述了你在这个项目中使用的 Pacman GameState 类型。
game.py	Pacman 世界如何运作背后的逻辑。该文件描述了几种类型，如 AgentState、Agent、Direction 和 Grid
util.py	用于实现搜索算法的有用数据结构
你可以忽略的支持文件:	
graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	Keyboard interfaces to control Pacman
layout.py	Code for reading layout files and storing their contents
autograder.py	Project autograder
testParser.py	Parses autograder test and solution files
testClasses.py	General autograding test classes
test_cases/	Directory containing the test cases for each question
searchTestClasses.py	Project-specific autograding test classes

请不要更改代码中提供的任何函数或类的名称，否则autograder很出bug。

如果你发现自己卡在某事上，请联系老师、助教或同学寻求帮助（现场、微信或邮件）。

欢迎来到Pacman世界

从对分易-课程资源-实验课课件，或从机房电脑 下载代码文件压缩包，解压缩到桌面，在PyCharm中，点击左上角 File → Open... → 查找解压的文件夹（如果是解压到机房桌面，应该在Administrator → Desktop → 对应的文件夹）并切换到目录后，你应该可以通过在命令行中键入以下内容来玩 Pacman 游戏：

```
python pacman.py
```

Pacman 生活在一个闪闪发光的蓝色世界中，这里有曲折的路和美味的小豆子美食。Pacman 掌握这个世界的第一步是进行有效地导航。

`searchAgents.py` 中最简单的agent称为 `GoWestAgent`，它总是向西走（一个简单的反射agent）。在不同的迷宫testMaze和tinyMaze运行这个agent试试，这个agent偶尔会赢：

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

但是，当需要转弯时，这个agent的情况会变得很糟糕

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

如果 Pacman 卡住了，你可以通过在终端中输入 CTRL-c 来退出游戏

很快，进行下面的任务实现后，你的agent不仅会解决`tinyMaze`，还能解决任意Maze

注意：`pacman.py` 支持许多选项，每个选项都可以用很长的方式表达 (e.g., `--layout`) 或短的方式 (e.g., `-l`). 你可以通过以下方式查看所有选项的列表及其默认值:

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

此外，该项目中出现的所有命令也出现在`commands.txt`. In UNIX/Mac OS X, 你甚至可以按顺序运行所有这些命令 `bash commands.txt`.

图搜索伪代码

对于 q1-3 中的搜索算法，你将实现以下粗略编写的图搜索伪代码

```
Algorithm: GRAPH_SEARCH:
frontier = {startNode}
expanded = {}
while frontier is not empty:
    node = frontier.pop()
    if isGoal(node):
        return path_to_node
    if node not in expanded:
        expanded.add(node)
        for each child of node's children:
            frontier.push(child)
return failed
```

Question 1: Finding a Fixed Food Dot using Depth First Search (无需在报告中提交)

任务说明

在 `searchAgents.py`, 你会发现一个 `SearchAgent`, 它计划了一条通过 Pacman 世界的路径，然后逐步执行该路径。**制定计划的搜索算法没有实现——那是你的工作。**

首先, 测试 `SearchAgent` 是否能工作:

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

上面的命令告诉 `SearchAgent` 用 `tinyMazeSearch` 作为其搜索算法，该算法在`search.py` 。Pacman 应该能成功地在迷宫中移动。

现在是时候编写你的搜索函数来帮助 Pacman 规划路线了!你将编写的搜索算法的伪代码可以在参考教材或课件中找到。请记住，**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。**

任务提示

重要: 你的所有搜索功能都需要返回将Agent从起点引导到目标的行动列表。这些行动都必须是合法的移动（不能穿过墙壁）。

重要：确保使用 `util.py` 中的 `Stack`, `Queue` 或 `PriorityQueue` 数据结构！ 这些数据结构实现具有与autograder兼容所需的特定属性。

提示：每个算法都非常相似。 DFS、BFS、UCS 仅在边缘节点（frontier）管理方式的细节上有所不同。因此，**集中精力让 DFS 正确**，其余的应该相对简单。实际上，一种可能的实现只需要一个通用搜索方法，该方法配置有特定于算法的排队策略。

任务示例

在 `search.py` 中 `depthFirstSearch` 函数中实现深度优先搜索 (DFS) 算法。为了使你的算法完整，请编写 DFS 的图搜索版本，以避免扩展任何已访问的状态。

你的代码应该很快在以下例子中找到解决方案：

```
python pacman.py -l tinyMaze -p SearchAgent
```

```
python pacman.py -l mediumMaze -p SearchAgent
```

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

Pacman 界面将显示探索状态的叠加层，以及探索的顺序（较亮的红色表示较早的探索）。探索顺序是你所期望的吗？吃豆人真的会在前往目标的路上走遍所有探索过的网格吗？

提示: 如果你使用 `Stack` 作为数据结构，则你的 DFS 算法为 `mediumMaze` 找到的解决方案的长度应为 130（前提是你按 `expand` 提供的顺序将子节点推到边缘节点集；如果相反的顺序，你可能会得到 246）。这是成本最低的解决方案吗？如果不是，请考虑深度优先搜索做错了什么。

Question 2: Breadth First Search 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q2 的运行结果，获得评分器给的全部3分，这道题才算得它占的1分。**

任务说明

在 `search.py` 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，避免扩展任何已经访问过的状态。用和与深度优先搜索相同的方式测试代码。

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

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

BFS 是否找到成本最低的解决方案？如果没有，请检查你写的代码。

任务提示与拓展

提示: 如果 Pacman 对你来说移动得太慢，请尝试选项 `--frameTime 0`.

注意：如果你已经编写了通用的搜索代码，那么你的代码应该同样适用于 eight-puzzle 搜索问题，无需任何更改。

```
python eightpuzzle.py
```

任务测试

请运行以下命令以查看你的实现是否通过了所有autograder测试用例:

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

Question 3: Varying the Cost Function 报告应实现任务[1分]

- **说明：该小题截图只需截 python autograder.py -q q3 的运行结果，获得评分器给的全部3分，这道题才算得它占的1分。**

任务说明

虽然 BFS 会找到到达目标的最少操作路径，但我们可能希望找到其他意义上的“最佳”路径。考虑`mediumDottedMaze` 和 `mediumScaryMaze`。

通过改变cost function，我们可以鼓励 Pacman 找到不同的路径。例如，我们可以对前往鬼怪区域的危险步骤设定更高的cost，或者对食物丰富的区域的步骤设定更少的cost，并且理性的Pacman应该调整其行为作为这个设定的回应。

任务提示

在 `search.py` 的空函数 `uniformCostSearch` 中实现一致代价图搜索算法。我们鼓励你查看 `util.py` 以了解一些可能对你的代码实现有用的数据结构。实现后，你应该能在以下所有三种迷宫中观察到成功的agent行为，其中下面的agent都是 UCS agents，它们仅在它们使用的cost functions上有所不同（agent和成本函数已为你编写）：

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
```

```
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
```

```
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

注意: `StayEastSearchAgent` 和 `StayWestSearchAgent` 的路径代价应该分别非常低和非常高，因为它们是指数成本函数（有关详细信息，请参阅 `searchAgents.py`）。

任务测试

请运行以下命令以查看你的实现是否通过了所有autograder测试用例:

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

本节课实现内容保存说明 - 为第二次上机报告做准备

- **第四周上机课结束后，一并提交本部分"搜索"的报告，该报告包含本节课的Q2,Q3的代码和运行截图，请按Q2,Q3大标题下的说明进行截图，保存。**
- **本节课实现的代码文件也请保存，尤其是实现了的 search.py 文件，你将需要把其中实现的代码加到第四周的search.py中跟第四周的任务实现一并进行提交。**