

1、实验目的

加深对深度优先、广度优先、代价一致搜索、A*搜索等搜索算法的理解，能独立实现这些算法完成 Pacman 中的任务。

2、实验介绍

Pac-Man（吃豆人）是1980推出的迷宫动作视频游戏，加州大学伯克利分校借鉴该游戏开发了开源的人工智能实验项目 Pacman，通过编写 AI 策略引导 Agent（智能体）走出迷宫，项目地址：

<https://inst.eecs.berkeley.edu/~cs188/fa23/projects/proj1/>。

该实验项目要求编写吃豆人搜索策略解决：找到一个指定位置的豆子、到达四个角落、吃掉所有豆子三大类问题。

2.1 Python 安装

整个项目使用 python3 开发，支持 Python 3.6 及以上版本，可运行在 Windows 和 Linux 操作系统，实验一可以任选自己熟悉的 Windows 或者 Linux 完成。

实验室电脑 Windows 系统已经安装了 Python，打开命令行窗口输入 Python 即可进入到的交互界面，同时会输出 Python 版本信息，版本信息会因安装版本不同。Linux 发行版通常自带 Python，可以直接使用。

```
I:\>python
Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>>
>>>
>>> quit()
```

如果使用的 Windows 环境没有安装 Python，可参考以下方式安装：

1. 下载对应版本安装包<https://www.python.org/downloads/>
2. 按照官方的[python 安装指引: 4.1.1 安装步骤](#)安装，再按照 4.6 节配置环境变量 path。

Python 开发 IDE 可以使用 Pycharm 或 VS Code，具体使用可查看对应官方教程。Python 基础需自主学习，课程后续实验继续使用 Python 完成。主要学习 python 安装、基本的数据类型（list, dict, tuple）、函数、类、包的导入等基础，可参考[Python 官方教程中文版](#)。

2.2 Pacman 运行

下载实验代码 [search.zip](#)，解压后，在命令行中进入到 search 目录，运行 `python pacman.py` 命令，进入交互式的吃豆人游戏，通过方向键可以控制 Agent 移动躲避 ghost，吃完所有的豆子就是胜利，结束后会在控制台输出以下信息。

```
I:\search>python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
[SearchAgent] using function tinyMazeSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:          502.0
Win Rate:        1/1 (1.00)
Record:          Win
```

```
I:\search>|
```

本实验只在非交互式模式下让智能体自动完成任务，无需方向键控制。如果运行过程中长时间卡住可按ctrl+c或者点击右上角的关闭按钮结束。

Pacman项目实现的功能比较多，不同的任务用了不同的迷宫和不同的Agent。Agent的实现在searchAgents.py文件中，最简单的Agent叫做GoWestAgent，该Agent只能一路向西移动，不能转弯，只有在最简单的测试迷宫上能吃到豆子，只作为测试用。GoWestAgent在testMaze和tinyMaze上的测试：

1. 无转弯的简单迷宫testMaze，GoWestAgent一路向西能吃到豆子。

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

2. 有转弯的迷宫tinyMaze，GoWestAgent将吃不到豆子一直运行，通过CTRL-c来终止。

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

python pacman.py 命令后面可以跟选项，每个选项的含义和取值可通过运行 python pacman.py -h 查看。主要的选项说明如下：

--layout，简写为-l，指定加载目录layouts/下的迷宫。

--pacman，简写为-p，指定agent类型，默认是KeyboardAgent即通过键盘方向控制的智能体。

--zoom，简写为-z，图形窗口的大小，默认是1，比1大则是放大，比1小则是缩小。

--agentArgs，简写为-a，传递给agent的args取值，字符串的形式，如果有多个取值以逗号分隔，比如"opt1=val1,opt2,opt3=val3"。

2.3 search框架代码文件说明

pacman search项目有如下文件：

```
(base) $ ~/search »tree -F -L 1 --dirsfirst ./
./
├── layouts/
├── test_cases/
├── autograder.py
├── eightpuzzle.py
├── game.py
├── ghostAgents.py
├── grading.py
├── graphicsDisplay.py
├── graphicsUtils.py
├── keyboardAgents.py
├── layout.py
├── pacmanAgents.py
├── pacman.py
├── projectParams.py
├── searchAgents.py
├── search.py
├── searchTestClasses.py
├── testClasses.py
├── testParser.py
├── textDisplay.py
├── util.py
└── VERSION

2 directories, 20 files
```

完成实验只需要修改search.py和searchAgents.py两个文件，其他文件无需修改，需要补充代码的地方以注释 `*** YOUR CODE HERE ***` 或 `util.raiseNotDefined()` 做了提示。

需要阅读的几个文件如下：

文件名	功能
pacman.py	吃豆人游戏的主程序
game.py	吃豆人游戏的运行逻辑
util.py	搜索策略可以用到的数据结构

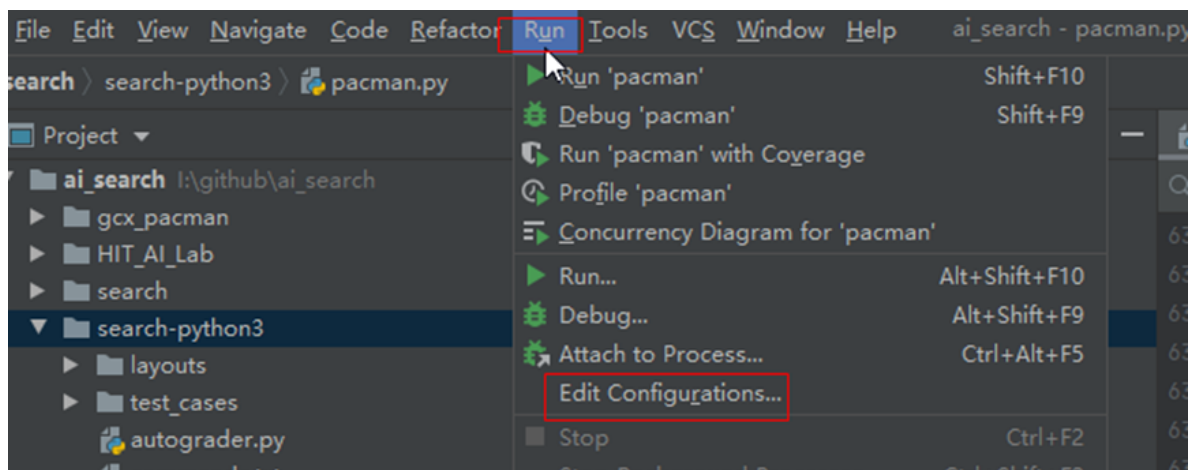
项目使用Tkinter包作为GUI开发框架，游戏交互逻辑实现较复杂，不需做了解。autograder等测试相关的文件如有兴趣可以自行了解。

2.4、pacman代码阅读

刚接手一个新的项目代码，比较快速的熟悉方式是先通过说明文件或者调试器找到函数的入口，再通过调试器单步调试查看代码的运行过程，边调试边阅读源代码。Pacman项目的入口代码是pacman.py中的runGames()。

```
if __name__ == '__main__':  
    """  
    The main function called when pacman.py is run  
    from the command line:  
  
    > python pacman.py  
  
    See the usage string for more details.  
  
    > python pacman.py --help  
    """  
    args = readCommand(sys.argv[1:]) # Get game components based on input  
    runGames(**args)  
  
    # import cProfile  
    # cProfile.run("runGames( **args )")  
    pass
```

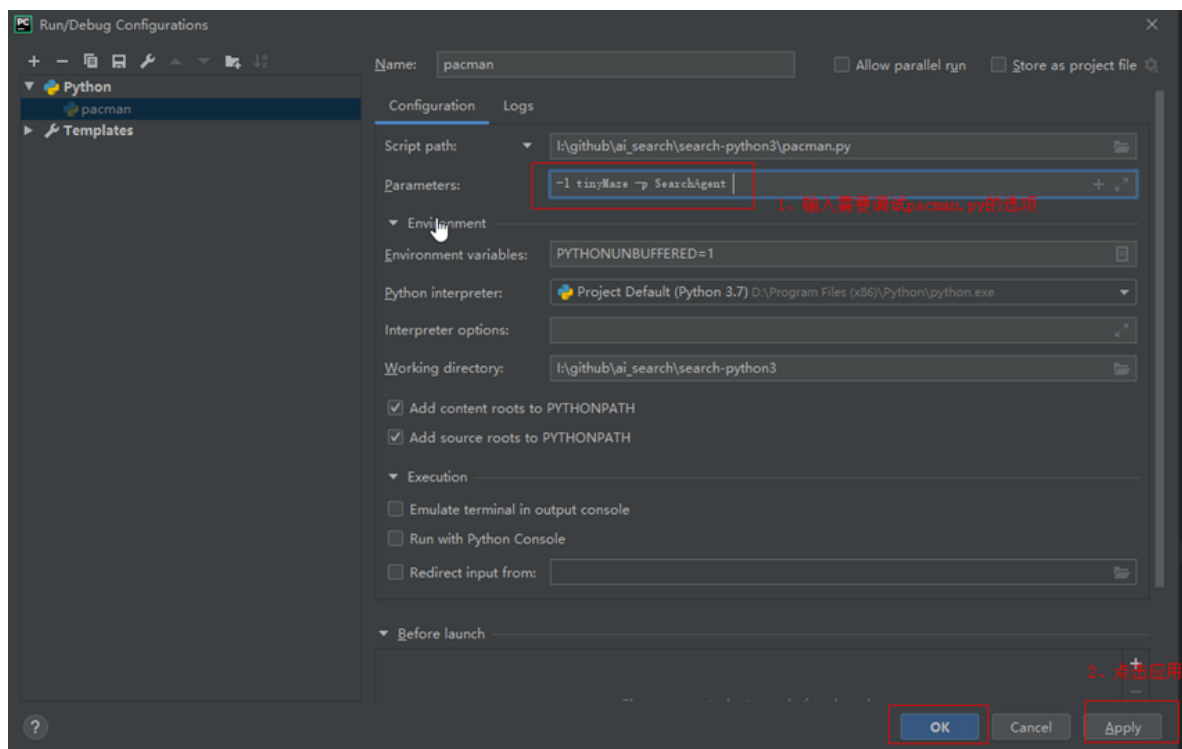
python自带调试工具pdb，类似c语言的gdb，操作比较类似。这里介绍下pycharm中可视化调试pacman ([debugging with pycharm](#))。先打开pacman.py，先点击Run工具栏中的Run pacman，运行结束之后按下图操作打开项目的配置编辑界面。



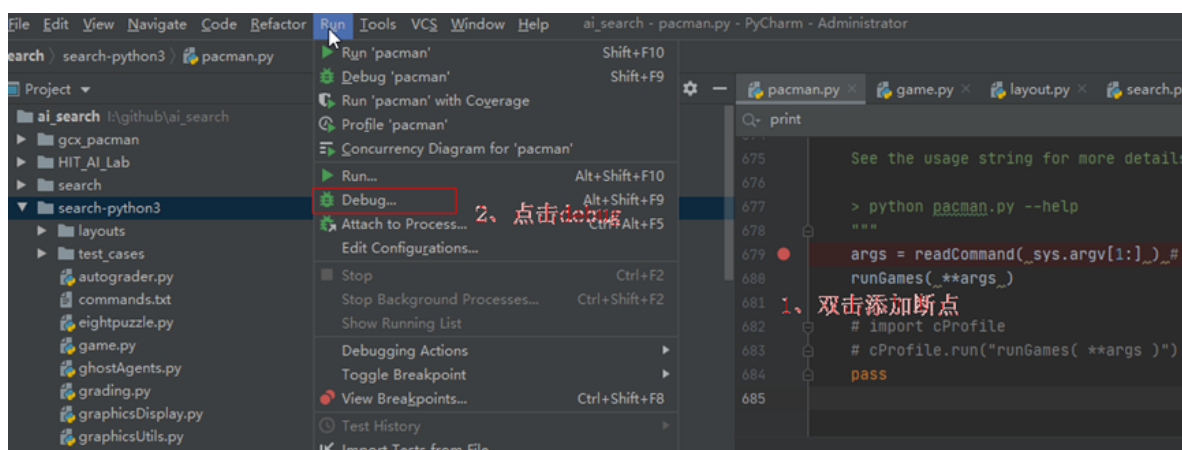
弹出如下界面，输入要调试的选项，比如要测试：

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

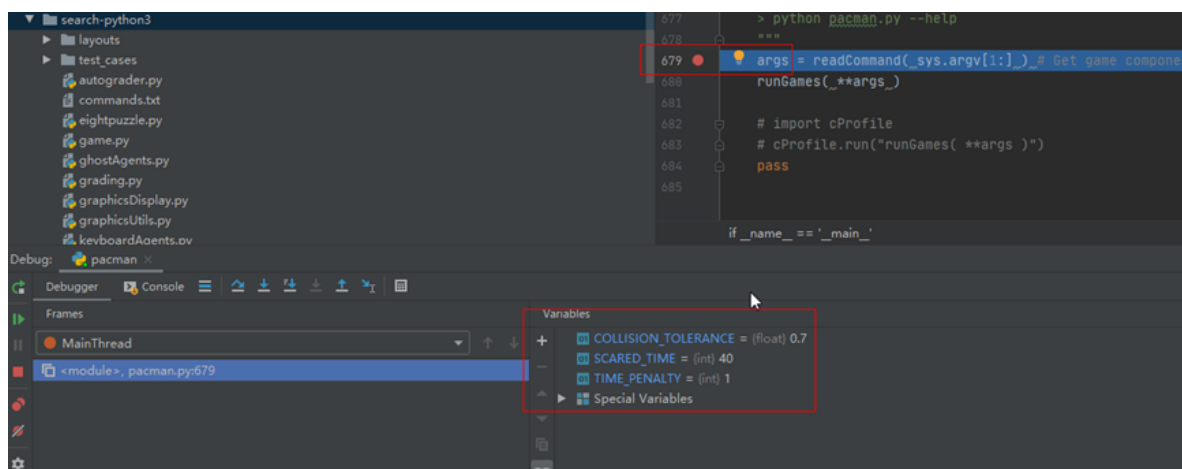
则在Parameters框中输入 `-l mediumMaze -p SearchAgent`，点击应用，再点击ok即完成配置。



在代码行数旁边双击添加断点，点击debug，即可运行代码并进入调试。



会自动在第一个断点处停止，并显示当前的变量信息，然后根据需要可以单步调试step over或step into或者其他操作。



对于要解决的问题1，使用深度优先搜索算法找到特定位置的豆子的，测试命令是：

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

pacman类型是SearchAgent, 迷宫是tinyMaze, 代码的主要运行逻辑是: `readCommand->ClassicGameRules->rules.newGame->game.run`, 即解析参数、定义游戏规则、创建游戏、运行游戏。如下图所示代码, SearchAgent默认使用depthFirstSearch算法, 解决的问题是PositionSearchProblem, PositionSearchProblem问题继承自抽象类search.SearchProblem。

```
class SearchAgent(Agent):
    """
    This very general search agent finds a path using a supplied search
    algorithm for a supplied search problem, then returns actions to follow that
    path.

    As a default, this agent runs DFS on a PositionSearchProblem to find
    location (1,1)

    Options for fn include:
    depthFirstSearch or dfs
    breadthFirstSearch or bfs

    Note: You should NOT change any code in SearchAgent
    """

    def __init__(self, fn='depthFirstSearch', prob='PositionSearchProblem', heuristic='nullHeuristic'):
        # Warning: some advanced Python magic is employed below to find the right functions and problems
```

3、实验具体内容

3.1 找到一个特定位置的豆子：问题1-4

首先, 运行以下命令测试Python环境和SearchAgent是不是正常工作。

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

这条命令的意思是用 tinyMazeSearch 算法在tinyMaze迷宫中找到特定位置的豆子。命令中fn是function的简写即搜索算法, 只能传入search.py文件中定义的函数, 提供了bfs、dfs、astar、ucs四个缩写。正常情况会弹出游戏界面, 游戏结束后会输出运行情况, 游戏分数Score无需关注。

```
I:\search>python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
[SearchAgent] using function tinyMazeSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:          502.0
Win Rate:        1/1 (1.00)
Record:          Win

I:\search>
```

tinyMazeSearch默认已实现, 该搜索算法只能在tinyMaze迷宫中成功找到豆子。接下来需要完成指定的搜索算法帮助吃豆人规划路线。

3.1.1 问题1：应用深度优先搜索找到一个特定的位置的豆

找到一个特定位置的豆子是路径规划问题, 初始状态、动作、目标函数定义如下:

• Pathing

- States: (x,y) locations
- Actions: North, South, East, West
- Transition model (getting the next state):
Update location only
- Goal test: Is (x,y)=END?

相关实现在 `searchAgents.py/PositionSearchProblem` 中, 请阅读对应代码。对于 `PositionSearchProblem`, 大家只需要实现对应的搜索算法函数即可。对于问题1, 实现深度优先算法 `searcy.py/depthFirstSearch`, 运行以下测试命令, 如果实现正确都能顺利通过:

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

弹出的游戏界面中, 对于已经搜索过的状态, 迷宫上将显示一个叠加物(overlay), 并显示出访问的顺序(红色由深到浅)。以上3个测试都通过并不代表完全正确, 需再运行以下自动化测试, -q选项是指定对应的问题。

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

若没有实现对应算法或者实现出错, 执行之后会有错误提示。下图是未实现`depthFirstSearch`时的错误信息, 会提示FAIL以及出错的原因。

```
I:\search>python autograder.py -q q1
Starting on 3-27 at 17:06:16
```

```
Question q1
```

```
=====
```

```
*** Method not implemented: depthFirstSearch at line 90 of I:\search\search.py
*** FAIL: Terminated with a string exception.
```

```
### Question q1: 0/3 ###
```

```
Finished at 17:06:16
```

```
Provisional grades
```

```
=====
```

```
Question q1: 0/3
```

```
-----
```

```
Total: 0/3
```

```
Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

```
I:\search>
```

如果运行`python autograder.py`出现以下错误, 是因为python版本不兼容的问题。可以在`grading.py`中 `import html`, 再将294行的`cgi.escape`用`html.escape`代替。


```

Traceback (most recent call last):
  File "D:\code\AI_2022\ref-search-python3\autograder.py", line 355, in <module>
    evaluate(options.generateSolutions, options.testRoot, moduleDict,
  File "D:\code\AI_2022\ref-search-python3\autograder.py", line 311, in evaluate
    grades.grade(sys.modules[__name__], bonusPic = projectParams.BONUS_PIC)
  File "D:\code\AI_2022\ref-search-python3\grading.py", line 81, in grade
    self.addExceptionMessage(q, inst, traceback)
  File "D:\code\AI_2022\ref-search-python3\grading.py", line 149, in addExceptionMessage
    self.fail('FAIL: Exception raised: %s' % inst)
  File "D:\code\AI_2022\ref-search-python3\grading.py", line 272, in fail
    self.addMessage(message, raw)
  File "D:\code\AI_2022\ref-search-python3\grading.py", line 294, in addMessage
    message = cgi.escape(message)
AttributeError: module 'cgi' has no attribute 'escape'

```

参考来源: <https://stackoverflow.com/questions/62470666/getting-this-error-with-py2-7-as-well-as-with-py3-7>)

如果结果出错, 可以使用print或者pdb单步进行调试:

1. 在适当的位置使用print输出观测信息, 比如查看depthFirstSearch函数的节点扩展过程, 与参考结果做对比。
2. 使用pycharm或者命令行pdb的单步调试, 结合出错信息, 查看变量的中间状态。

如果实现正确, 每个测试用例会显示PASS, 并显示得分。

```

(base) $ ~/search »python autograder.py -q q1
Starting on 3-27 at 17:08:07

```

```

Question q1
=====
*** PASS: test_cases/q1/graph_backtrack.test
***   solution:          ['1:A->C', '0:C->G']
***   expanded_states:   ['A', 'D', 'C']
*** PASS: test_cases/q1/graph_bfs_vs_dfs.test
***   solution:          ['2:A->D', '0:D->G']
***   expanded_states:   ['A', 'D']
*** PASS: test_cases/q1/graph_infinite.test
***   solution:          ['0:A->B', '1:B->C', '1:C->G']
***   expanded_states:   ['A', 'B', 'C']
*** PASS: test_cases/q1/graph_manypaths.test
***   solution:          ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
***   expanded_states:   ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases/q1/pacman_1.test
***   pacman layout:    mediumMaze
***   solution length:  130
***   nodes expanded:    146

### Question q1: 3/3 ###

```

Finished at 17:08:07

Provisional grades

=====

Question q1: 3/3

Total: 3/3

Your grades are NOT yet registered. To register your grades, make sure to follow your instructor's guidelines to receive credit on your project.

```

(base) $ ~/search »

```

depthFirstSearch 返回的是一个从初始状态到目标状态的操作 (actions) 序列, 即移动的方向序列, 具体取值可以参考 `search.py/tinyMazeSearch` 函数的返回结果, 每一步的actions从 `SearchProblem.getSuccessors` 获取, 所有操作必须合法 (不能翻墙)。

说明:

1. 从 `util.py` 中选择合适的数据结构实现, 否则无法兼容autograder测试将不通过。
2. 搜索算法不只是到了目标节点就返回, 还需返回到目标节点的路径。对于 `PositionSearchProblem`, 目标节点goal是固定的 (1,1)。

3. 压入到open表的数据可以自定义，可以是tuple、list或者两者的嵌套等，除了状态本身，还可以附加其他信息。

3.1.2 问题2：应用广度优先搜索找到一个特定的位置的豆

实现广度优先搜索算法 `searcy.py/breadthFirstSearch`，并分别通过以下测试：

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
python autograder.py -q q2
```

如果已经正确实现了深度优先搜索，广度优先搜索的实现则很容易。实际上，深搜（DFS）、广搜（BFS）、代价一致搜索（UCS）、A*搜索可以用一个通用的图搜索框架实现。以上四个问题可以用通用的图搜索算法实现，伪代码如下，reached对应closed表，frontier对应open表，不同的搜索方法仅仅在于open表内元素的排序不同。

```
function GRAPH-SEARCH(problem, frontier) return a solution or failure
    reached ← an empty set
    frontier ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), frontier)
    while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        if problem.IS-GOAL(node.STATE) then return node
        end if
        if node.STATE is not in reached then
            add node.STATE in reached
            for each child-node in EXPAND(problem, node) do
                frontier ← INSERT(child-node, frontier)
            end for
        end if
    end while
    return failure
```

3.1.3 问题3：应用代价一致搜索找到一个特定的位置的豆

深搜和广搜中，默认每一步的代价都是1。现在考虑更复杂一点的情况，通过修改代价函数 `costFn`，使得每一步的代价并不相同，Pacman从而可以发现不同路径。例如，有ghost的区域，增加每步的代价，而在食物丰富的区域减少每步的代价，一个理性的Pacman应该相应地调整它的行为，躲开ghost吃到更多的食物。实现代价一致搜索算法 `searcy.py/uniformCostSearch`，并分别通过以下测试：

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
python autograder.py -q q3
```

3.1.4 问题4：应用A* 算法找到一个特定的位置的豆

实现A*搜索算法 `searcy.py/aStarSearch`，利用曼哈顿距离作为启发函数，并分别通过以下测试：

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic
python autograder.py -q q4
```

3.2 找到所有的角落：问题5-6

在角落迷宫的四个角上面有四个豆，该搜索问题要求找到一条访问所有四个角落的最短的路径。

3.2.1 问题5：基于BFS的角落问题

相比于问题1-4，问题5-6的状态空间不同，需要先完成 `searchAgents.py/CornersProblem` 类，相比于 `PositionSearchProblem`，要初始状态、后继函数、目标函数都不同。新的状态只包含吃豆人的位置和角落的状态，使其能够表示角落是否被访问过。实现 `CornersProblem` 之后分别进行以下测试：

```
python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
python autograder.py -q q5
```

说明：

1. 状态的定义只需包含必要的信息，Agent位置和4个角落的状态，数据类型不限。
2. 后继函数 `getSuccessors` 中，子节点的 `cost` 都是1。

3.2.2 问题6：基于A*的角落问题

在问题5完成的 `CornersProblem` 基础上，使用A*算法，构建合适的启发函数，找到一条访问迷宫所有四个角落的最短路径。实现 `searchAgents.py/cornersHeuristic` 函数，并通过以下测试：

```
python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
python autograder.py -q q6
```

其中 `AStarCornersAgent` 等同于以下选项

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

不同的启发函数扩展的节点不一样得分也会不一样，要求启发函数是非平凡非负一致的，一致性即单调性，平凡的启发函数是处处为0，非平凡的启发函数会返回非0值。 `autograder.py` 会跟根据扩展的节点数给分，标准如下。

Number of nodes expanded	Grade
more than 2000	0/3
at most 2000	1/3
at most 1600	2/3
at most 1200	3/3

3.3 吃掉所有的豆子：问题7-8

3.3.1 问题7：吃掉所有的豆子（FoodSearchProblem）

构造恰当的启发式函数，利用 A* 算法，以尽可能少的步数吃掉迷宫中存在的所有豆子。FoodSearchProblem已经实现好了不需修改，问题的解定义为一条收集到世界中所有食物的路径。如果你已经正确地完成了前面的搜索算法，不需要改代码，使用null heuristic 的A*（等价于代价一致算法）将很快求得testSearch问题的最优解，可用以下命令测试：

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

根据自己定义的启发函数修改 searchAgents.py/foodHeuristic 函数，用以下命令测试：

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
python autograder.py -q q7
```

不同的启发函数扩展的节点会不一样，要求启发函数是非平凡非负一致的，autograder.py会跟根据扩展的节点数给分，标准如下。

Number of nodes expanded	Grade
more than 15000	1/4
at most 15000	2/4
at most 12000	3/4
at most 9000	4/4 (full credit; medium)
at most 7000	5/4 (optional extra credit; hard)

3.3.2 问题8：次最优搜索

有的时候，即使使用 A* 加上好的启发函数，找到吃掉所有豆子的最优路径也比较耗时。定义一个优先吃最近的豆子函数是提高搜索速度的一个好的办法。在本节中，使用 searchAgents.py/ClosestDotSearchAgent 进行次最优搜索，ClosestDotSearchAgent总是贪心地吃掉最近的豆子。但关键函数 findPathToClosestDot 需要补充实现，该函数搜索到最近豆的路径。完成 findPathToClosestDot 之前先实现 AnyFoodSearchProblem.isGoalState() 函数，再使用合适的搜索算法来求解问题，解会非常短。

实现之后用以下命令测试：

```
python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
python autograder.py -q q8
```

4、实验报告与提交说明

代码只需要提交修改后的search.py、searchAgents.py文件，报告内容见模板。