

**人工智能实验报告**

题 目 搜索求解

专 业 人工智能（视听觉信息处理）

学　　 号 7203610121

姓 名 刘天瑞

同 组 人 员 无

1. **背景简介/问题描述**

**实验内容：**

实验要求采用且不限于课程第四章内各种搜索算法此编写一系列吃豆人程序解决以下列出的问题1-7 （填充search.py和searchAgents.py这两个文件中的空缺代码），包括到达指定位置以及有效的吃豆等。

**简介：**

基本代码和支持文件可以从search.zip中获取。其中，一些需要参考的文件如下：

## 需要编辑的文件：search.py和searchAgents.py。这两个文件中有一些空缺代码需要大家补充。

## 需要参考的文件：

|  |  |
| --- | --- |
| [pacman.py](https://s3-us-west-2.amazonaws.com/cs188websitecontent/projects/release/search/v1/001/docs/pacman.html) | 吃豆人游戏的程序。 文件包括一个描述”吃豆人”gamestate的类型。 |
| [game.py](https://s3-us-west-2.amazonaws.com/cs188websitecontent/projects/release/search/v1/001/docs/game.html) | 吃豆人游戏的运行逻辑. 文件包括以下类型AgentState, Agent, Direction, and Grid. |
| [util.py](https://s3-us-west-2.amazonaws.com/cs188websitecontent/projects/release/search/v1/001/docs/util.html) | 搜索策略可以用到的数据结构. |

## 可以忽略的支持性文件：graphicsDisplay.py graphicsUtils.py textDisplay.py ghostAgents.py keyboardAgents.py layout.pyautograder.py testParser.py testClasses.py test\_cases/ searchTestClasses.py

**解压缩search.zip，在此目录下，运行以下指令可打开吃豆人游戏。**

***python pacman.py***

**运行*python autograder.py*可以帮助你对自己的程序打分**。

searchAgents.py中最简单的Agent叫做GoWestAgent，一路向西，偶尔能实现目标：

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

但是其不能实现转弯：

*Python pacman.py --layout tinyMaze --pacman GoWestAgent*

如果程序卡死，可通过CTRL-c来终止。

此项目中用到的指令也都储存在commands.txt文件中，可用于复制和粘贴。

## 问题1：应用深度优先算法找到一个特定的位置的豆

首先，运行一下命令测试SearchAgent是不是正常工作：

*Python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch*

然后，完成完整的通用算法帮助吃豆人规划路线。搜索算法的伪代码见附录。注意一个搜索节点不仅包含节点的状态，而且要包含构建搜索路径所需要的信息。

注意：所有的搜索函数必须返回一个从初始状态到目标状态的操作序列。所有操作必须合法（不能翻墙）。

注意：利用util.py文件中提供的Stack, Queue 和 PriorityQueue数据结构！这是自动评分系统的兼容性要求。

你的code应该能顺利解决以下问题：

*Python pacman.py -l tinyMaze -p SearchAgent*

*Python pacman.py -l mediumMaze -p SearchAgent*

*Python pacman.py -l bigMaze -z .5 -p SearchAgent*

注意：因为不同的搜索方法的不同之处仅仅在于open表的排序不同，因此请尽量定义一个通用的搜索算法解决问题1-3。提示：问题1-3的不同之处在于用不同的数据结构对open表进行排序。

## 问题2：宽度优先算法

利用宽度优先算法实现解决以上问题。并利用以下命令测试你的code：

*Python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs*

*Python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5*

## 问题3：A\* 算法

完成A\*搜索方法(search.py文件中的aStarSearch函数)，利用曼哈顿距离作为启发函数，用以下命令测试你得code：

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

## 问题4：找到所有的角落

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

完成searchAgents.py文件中的CornersProblem搜索问题，你需要重新定义状态，使其能够表示角落是否被访问。用以下命令测试你的code：

*Python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem*

*Python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem*

提示：新的状态只包含吃豆人的位置和角落的状态。

## 问题5：角落问题（启发式）

构建合适的启发函数，完成searchAgents.py文件中的cornersHeuristic角落搜索问题。用以下命令测试你的code：

*Python pacman.py -l mediumCorners* *-p AStarCornersAgent -z 0.5*

## 问题6：吃掉所有的豆子

用尽可能少的步数吃掉所有的豆子。完成searchAgents.py文件中的FoodSearchProblem豆子搜索问题。此问题利用之前A\*算法可以很容易找到解，可用以下命令测试：

*Python pacman.py -l testSearch -p AStarFoodSearchAgent*

构建合适的启发函数，完成searchAgents.py文件中的foodHeuristic豆子搜索（启发式）问题。用以下命令测试你的code：

*Python pacman.py -l trickySearch -p AStarFoodSearchAgent*

## 问题7：次最优搜索

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。补充完成searchAgents.py文件中的AnyFoodSearchProblem目标测试函数，并完成searchAgents.py文件中的ClosestDotSearchAgent部分，在此Agent当中缺少一个关键的函数：找到最近豆子的函数。用以下命令测试你得code：

*Python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5*

1. **算法介绍**

2.1 所用算法及解题思路的一般介绍

实验环境：Windows11；Python 3.10.7

所用算法：深度优先搜索、广度优先搜索和A\*算法。

区别：每个算法所使用的数据结构不同，深度优先搜索（DFS）使用先进后出的栈结构，广度优先搜索（BFS）使用先进先出的队列结构，而A\*算法使用的是优先队列结构或者说是一种小顶堆。它们的具体行进思路如下面的流程图所示：

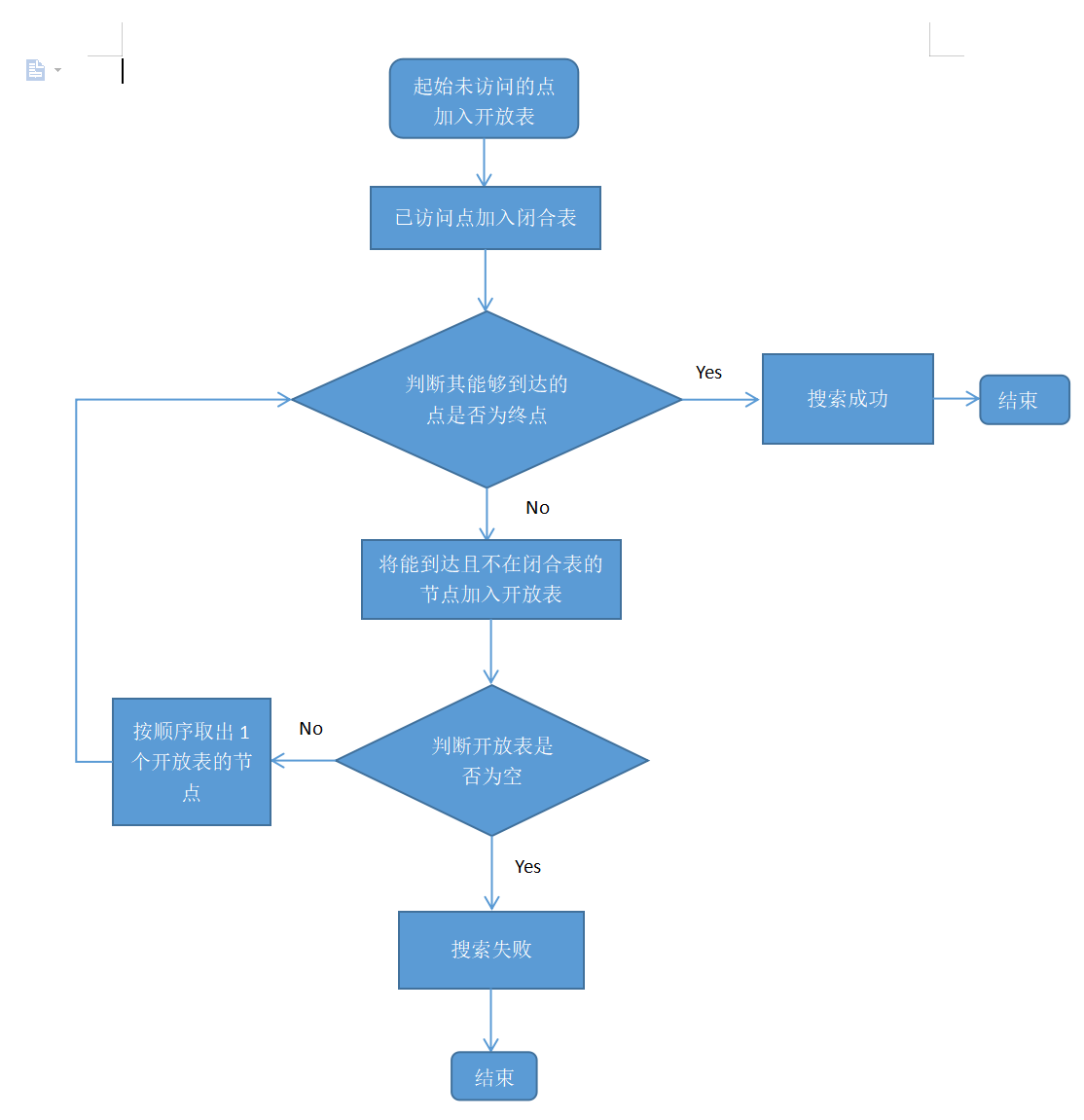


图1.3种搜索算法策略思路的流程图

1. **算法实现**

3.1 实验结果

问题1：应用深度优先算法找到一个特定的位置的豆

为实现一个图的深度优先搜索，很容易想到用栈结构，util.py中提供了栈的实现。为了让pacman在找路径的时候不走回头路，需要记录pacman访问过的节点，这里用python内置的set集合来记录访问过的节点，set使用哈希表的方法，查找效率高。利用栈的先进后出特性，可以优先从深度上访问节点。每次从栈取出一个节点，如果这个节点没有访问过，那么走到这个节点，并且把该节点可以走的子节点加入栈中，再从栈取出一个节点，以此循环，直到找到目标位置。找到目标后，返回经过的路径即可。

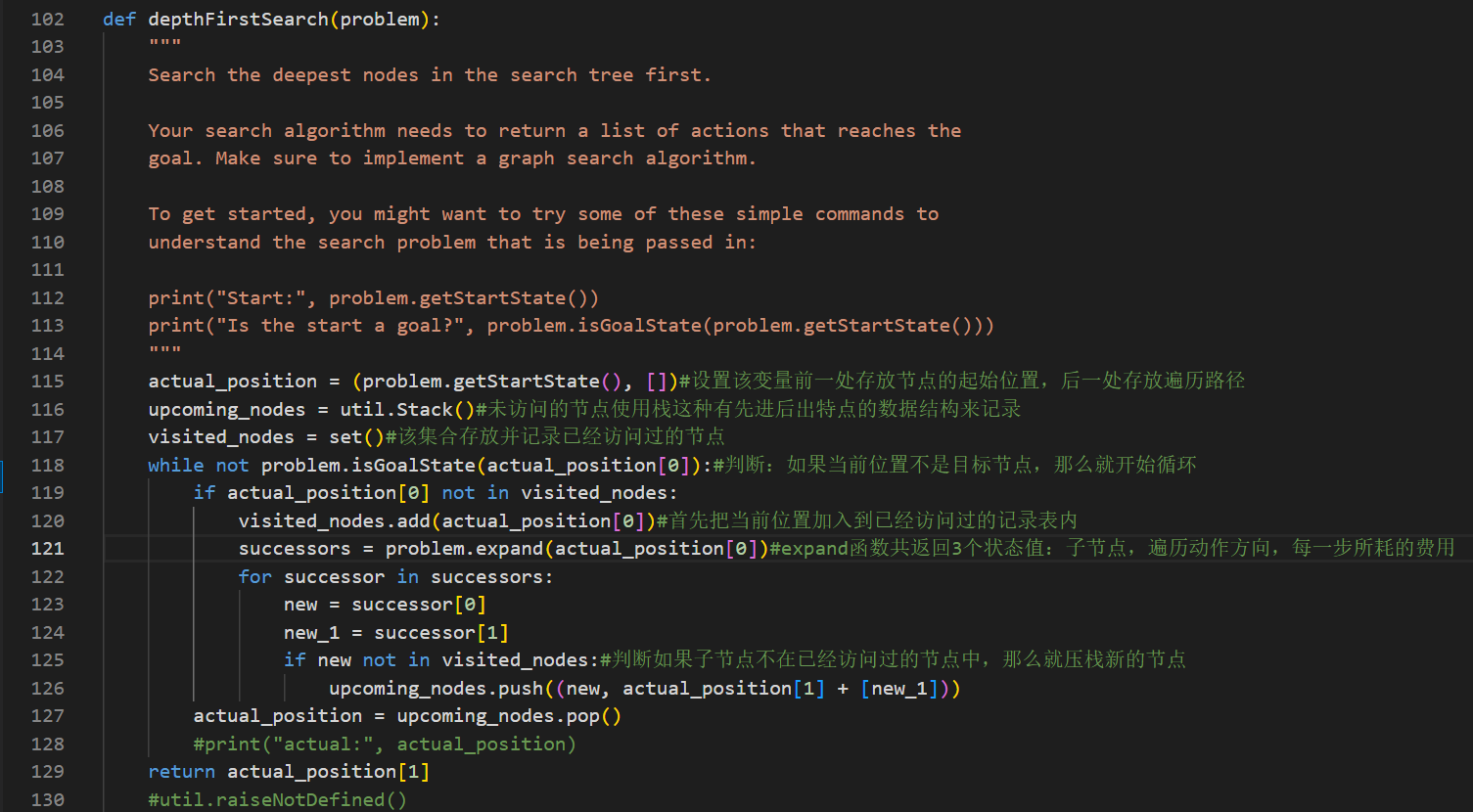


图2.问题1代码与注释

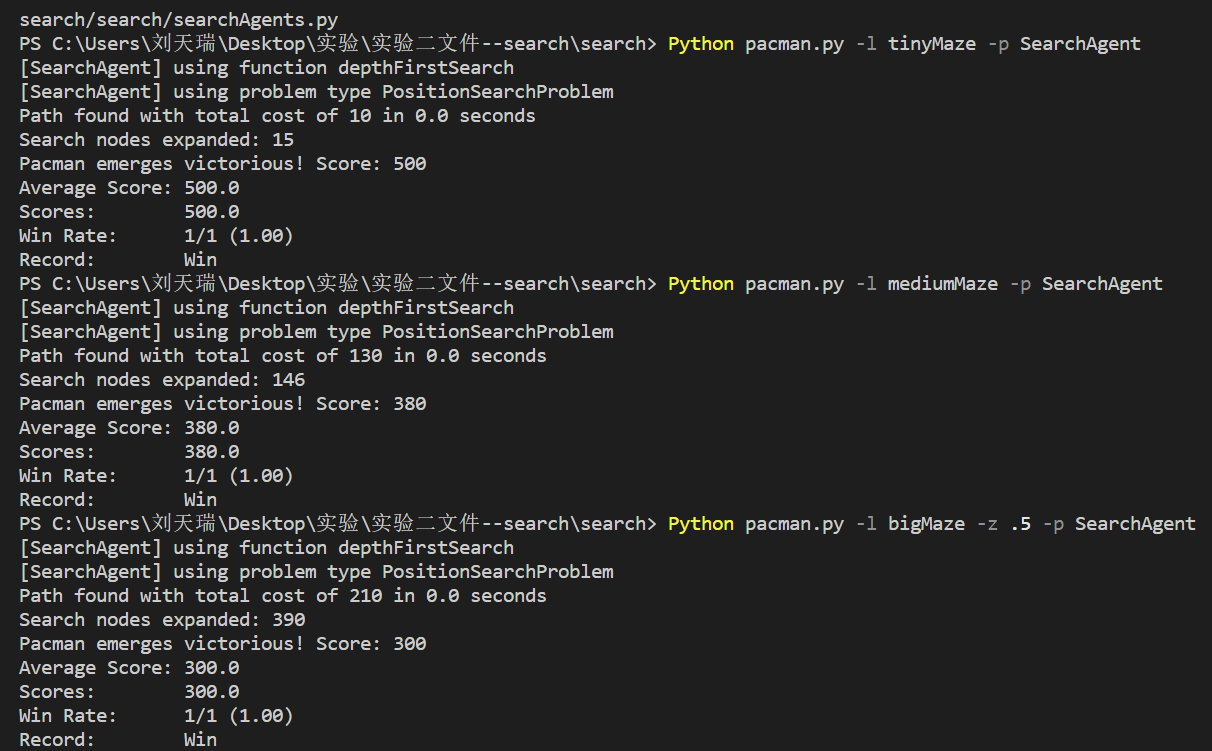


图3.问题1运行结果

问题2：宽度优先算法

广度优先搜索算法只需要把问题1中的栈改为队列即可。利用队列的先进先出特性，可以优先从广度上访问节点。每次从队列取出一个节点，如果这个节点没有访问过，那么走到这个节点，并且把该节点可以走的子节点加入栈中，再从队列取出一个节点，以此循环，直到找到目标位置。找到目标以后，返回经过的路径即可。

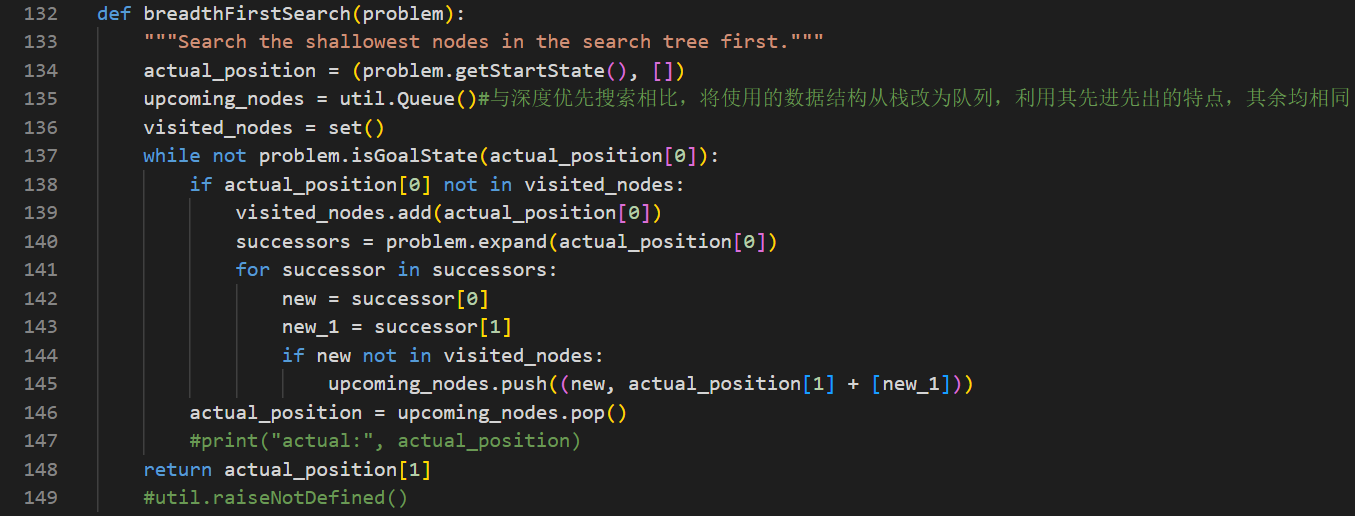


图4.问题2代码与注释

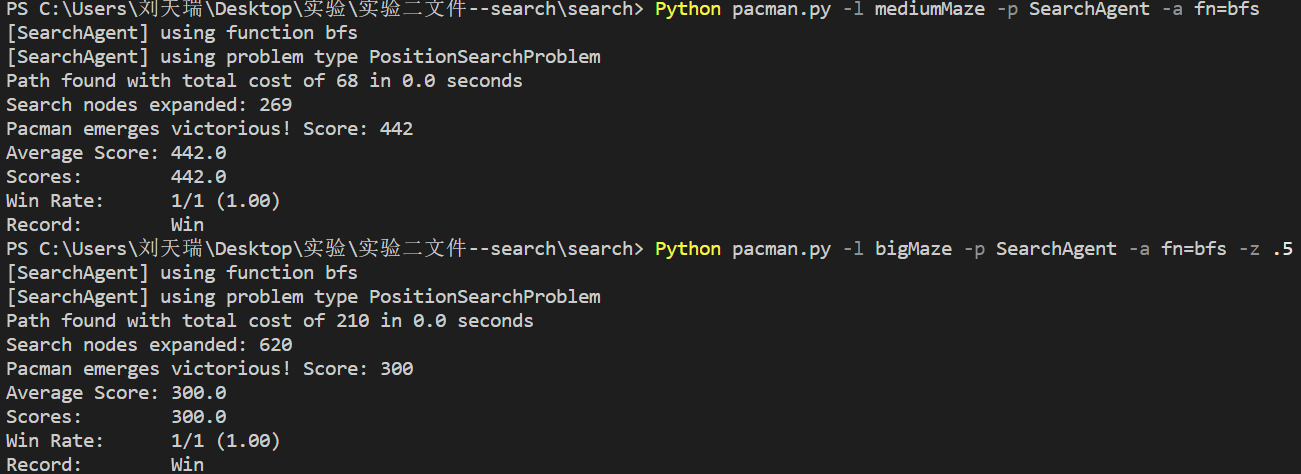


图5.问题2运行结果

问题3：A\* 算法

aStarSearch与dfs和bfs的参数不同的地方在于多了一个启发式函数heuristic。heuristic用于估计一个状态到下一个状态的代价，容易想到使用小顶堆/优先队列来实现状态的排列，堆中节点的权重是状态转换的真实费用和heuristic之和。理论上，该和越小，那么从该状态找到目标的时间、空间开销越小。利用小顶堆的特性，每次都从堆顶取费用和heuristic之和最小的状态，如果状态所在节点没访问过，那么访问该节点，并找到节点的子节点们，子节点们的位置和代价组成的特定数据结构添加到堆中，子节点加入堆时的权重为：从初始状态到父节点状态的代价+父节点状态到子节点状态的代价+heuristic(successor)。循环取堆顶状态，直到找到目标状态。

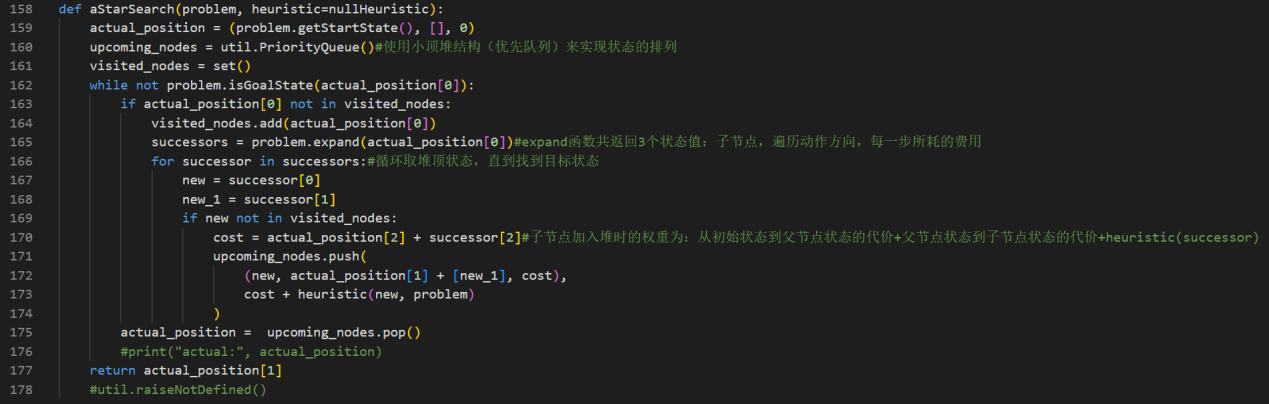


图6.问题3代码与注释

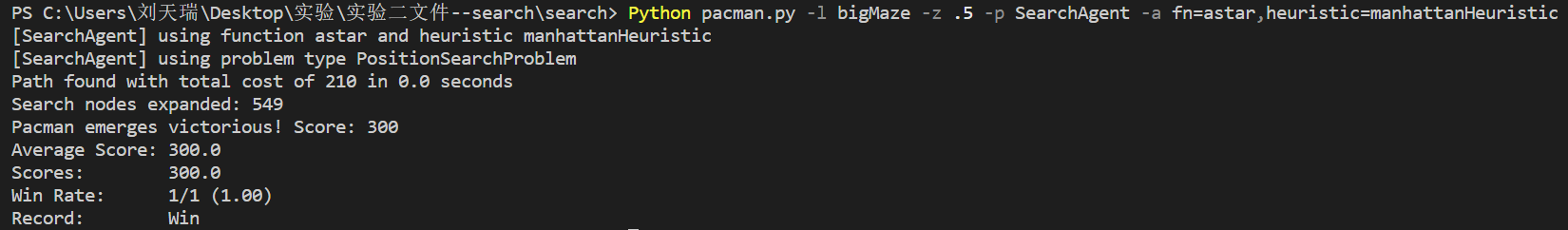
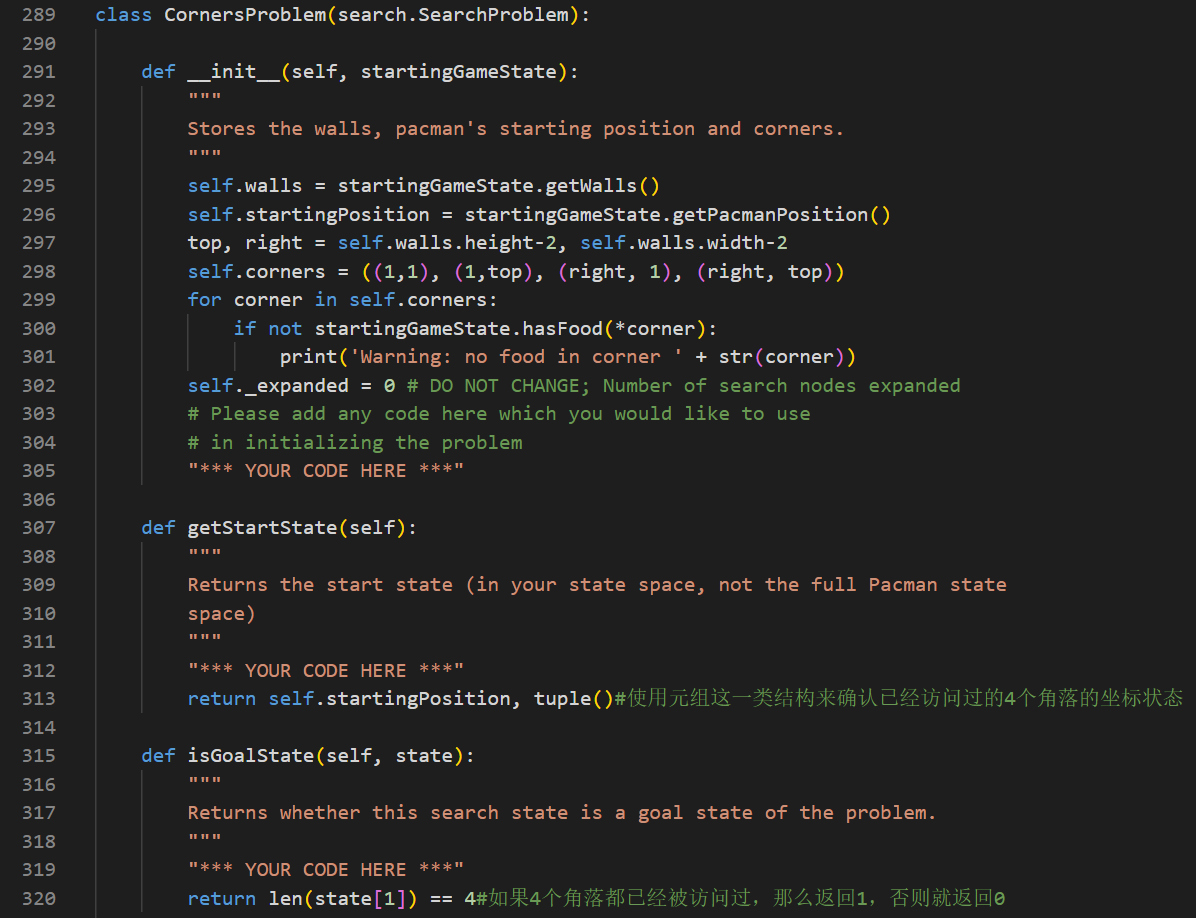


图7.问题3运行结果

问题4：找到所有的角落

首先需要重新定义pacman的状态，这里使用state(position, visited\_nodes())来表示。状态的第一个参数position表示pacman在迷宫图中的位置，为一个类似(x,y)的二元坐标，第二个参数visited\_nodes是一个Python元组，其包括当前路径已经访问过的角落节点的所有坐标。再者，部分函数的实现逻辑清晰可见。所以，目标状态就是visited\_nodes中包含所有4个角落坐标。



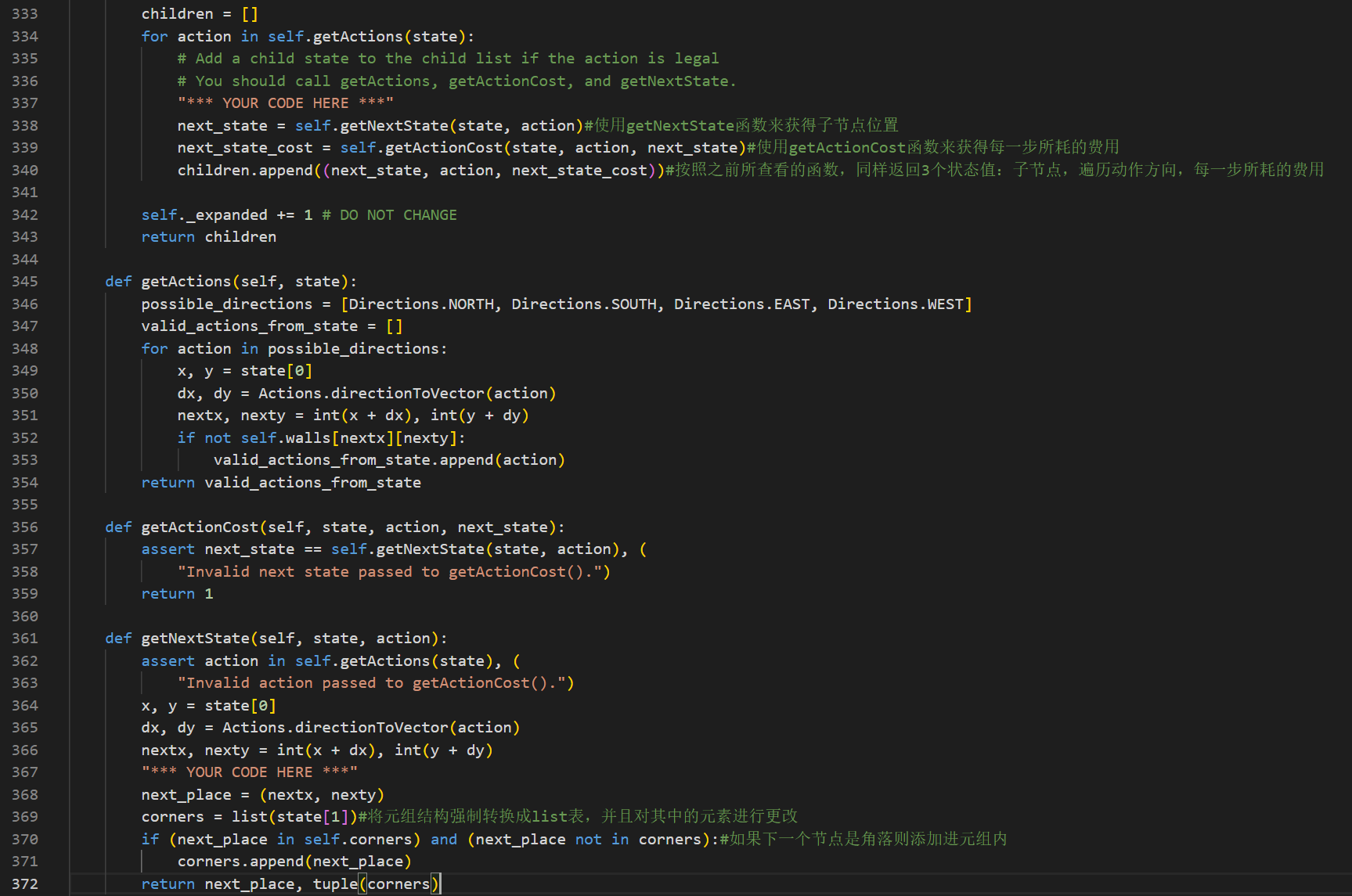


图8.问题4代码与注释

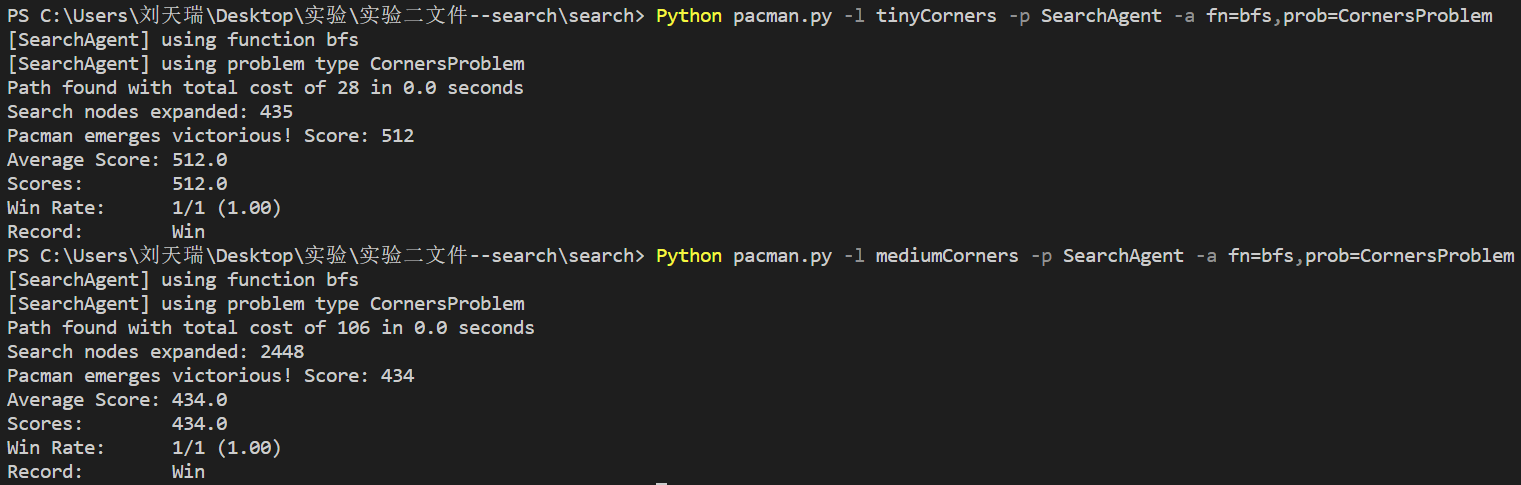


图9.问题4运行结果

问题5：角落问题（启发式）

问题4定义的problem中，当处于某个状态时，可能有多个未访问的角落，使用状态的坐标计算到各个未访问角落的曼哈顿距离，选用最大的曼哈顿距离作为启发式函数值，理由是如果因为到达某个状态导致到某个角落的距离更远，那么先访问该状态的权重应该更低，代价应该更大，使得该状态在小顶堆中的位置尽可能深，这样该状态被优先访问的概率降低，也许能够先搜索到更优路线。

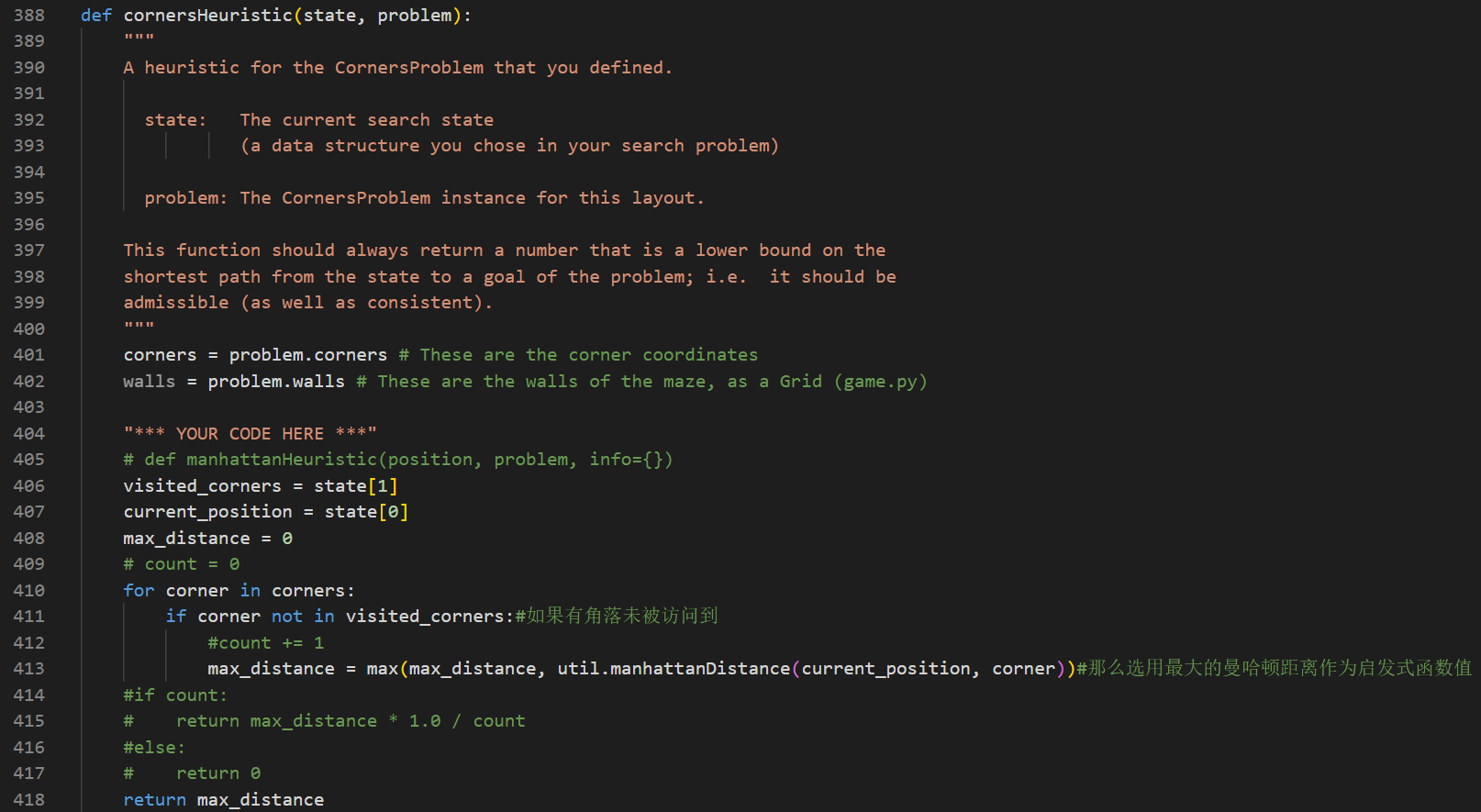


图10.问题5代码与注释

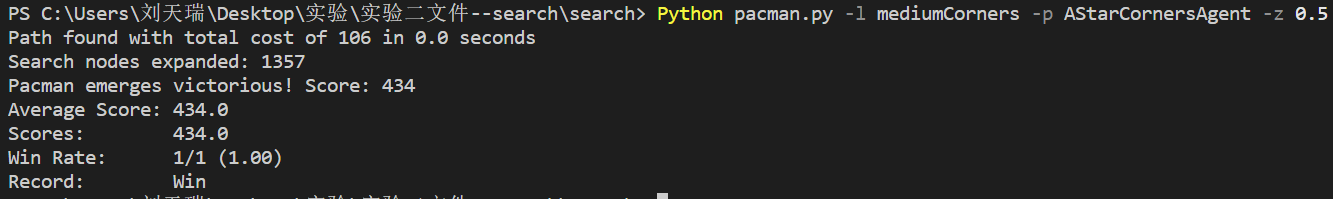


图11.问题5运行结果

问题6：吃掉所有的豆子

当处于某个状态时，此时可能有多个未吃的food存在，遍历所有的food位置，计算food坐标到当前状态坐标的mazedistance（利用已有函数mazeDistance），选取最大的mazedistance作为启发式函数的计算值。理由与问题5相似。如果因为到达某个状态导致到某个距离最远的food的距离更远，那么先访问该状态的权重应该更低，代价应该更大，使得该状态在小顶堆中的位置尽可能深，这样该状态被优先访问的概率降低，也许能够先搜索到更优路线。

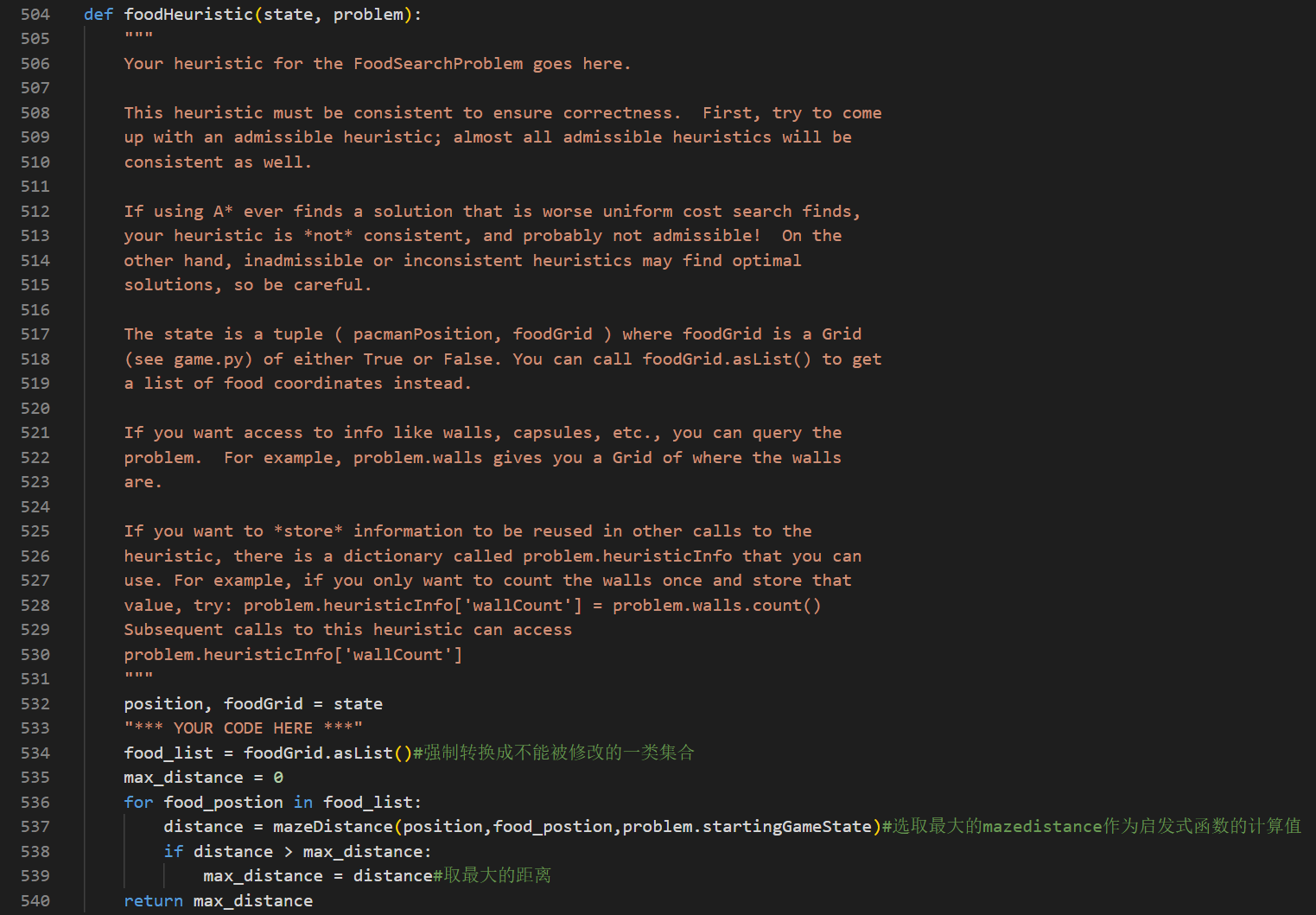


图12.问题6代码与注释

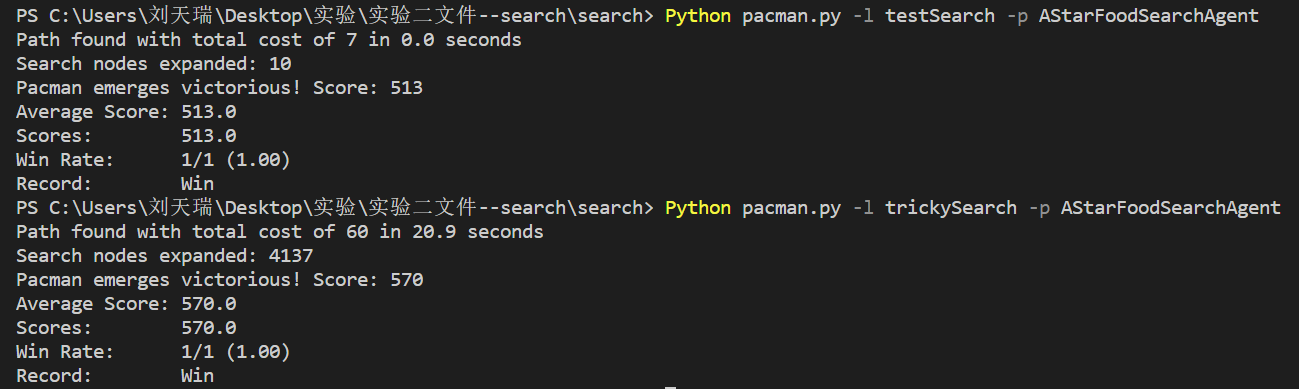
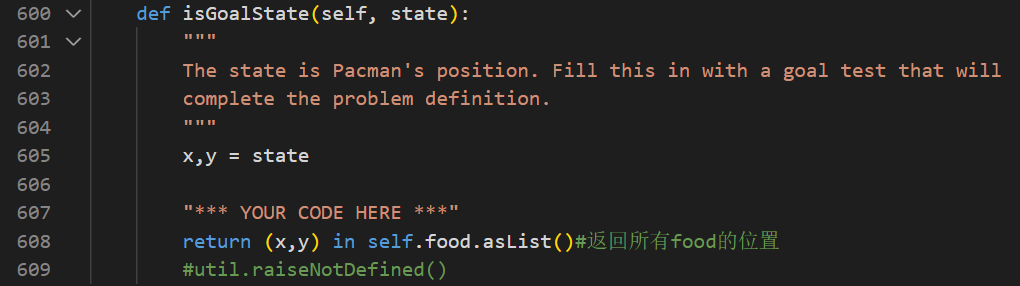


图13.问题6运行结果

问题7：次最优搜索

首先需要重新定义problem，如何每次都先去找到距离最近的food：使用广度优先搜索，因为广度优先搜索从状态所在位置逐层扩散产生路径，距离最近的food一定最先被找到。于是修改AnyFoodSearchProblem中的isGoalState函数,当访问到的坐标(x,y)位于food列表中某一个时，都达到目标状态，然后用广度优先搜索依次去找到目标状态（即找到最近food）。



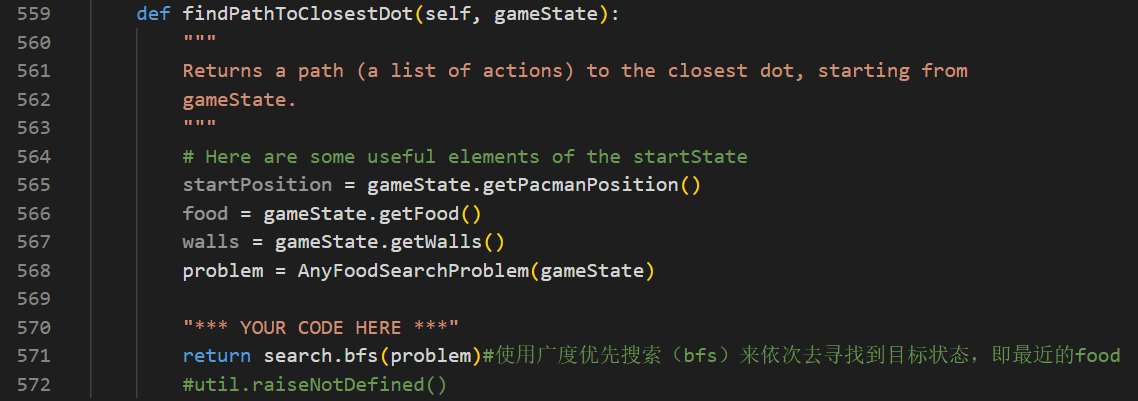


图14.问题7代码与注释

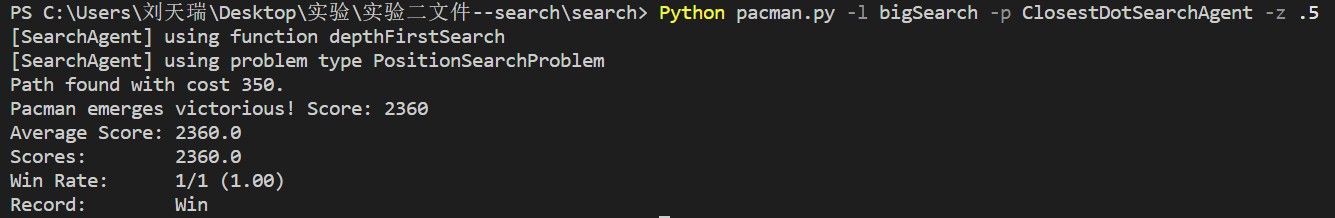


图15.问题7运行结果

运行autograde，使用一些测试案例测试上述7个问题的代码：

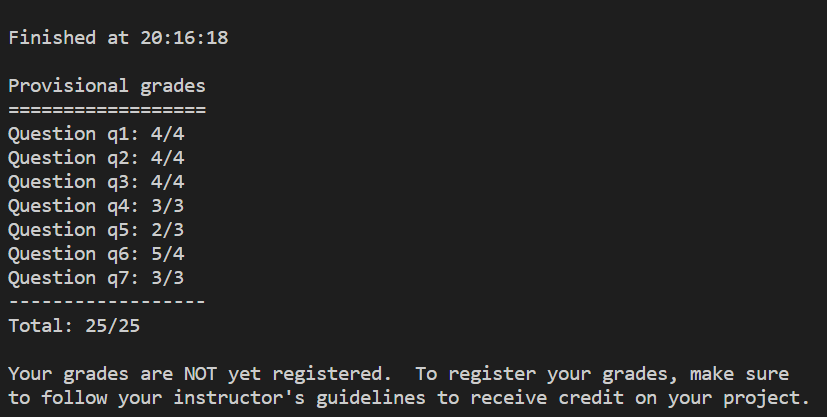


图16.7个问题的测试结果

发现问题5的test3我并未通过，原因是expand的节点个数为1357，已经超出所要求的1200以下。

1. **讨论及结论**

4.1 讨论与结论

总结：DFS在搜索中使用栈返回，IDA\*是DFS的延伸并且依旧用到栈结构，而BFS使用队列扩展，A\*使用堆来取出评估最好的值，理解这三个数据结构，才能较好理解这三种算法，无非就是：如果这点符合条件的就扩展，然后用不同的数据结构扩展，前两者不管数据如何，看到就放入，后者用堆，实现大小排序，可以选择更好的路径。

几种搜索不同的条件下返回，DFS是走到底，A\*跟BFS是队列为空，IDA\*是到达目标深度或者通过剪枝。

对于这次的吃豆人迷宫问题，我认为dfs并不能找到最短的搜索路径，但它时间复杂度和空间复杂度均较小。bfs则一定能找到最短的搜索路径，其搜索策略保证每条路径都是同步地扩大，但正因如此才会导致时间和空间复杂度均较高。而A\*算法综合二者优点，但其启发性能取决于评判函数，需要更多时空资源来计算估值从而有效排序。除此之外，在不同的程序之间，参数规格各不相同，保证参数正确传递的接口更为兼容稳定与高效，协调复杂的程序是写代码时关注的重中之重。

**参考文献**

1. [人工智能原理及其应用（第4版）](https://baike.baidu.com/reference/49764044/dad4lODc3k3H5p4ctpSN3ySkuaRZmNm1KgUQLv0jZcQcQas6PMHgHWpwagHjXUGtN1EEgrSzUmjlHhjguABwQVyrDNuTBX88J-HwOT2k5vng5JAtStxbiObVfDKPq7g" \t "https://baike.baidu.com/item/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD%E5%8E%9F%E7%90%86%E5%8F%8A%E5%85%B6%E5%BA%94%E7%94%A8%EF%BC%88%E7%AC%AC4%E7%89%88%EF%BC%89/_blank).电子工业出版社
2. <https://blog.csdn.net/ljt735029684/article/details/78945098?ops_request_>