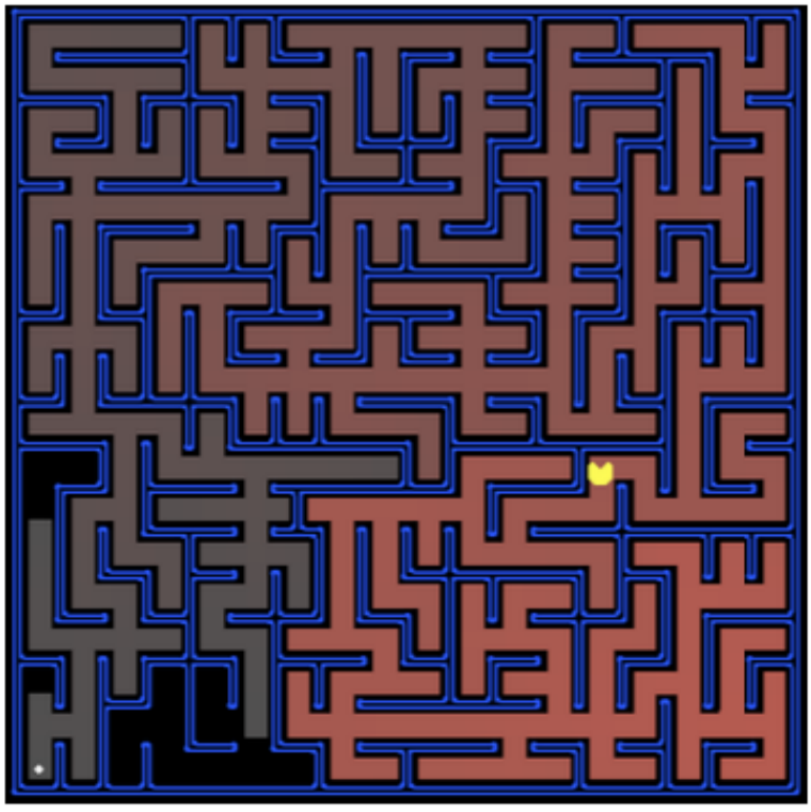
Project1 在吃豆人游戏中进行搜索

改编自最初由 John DeNero 和 Dan Klein 创建的伯克利吃豆人作业。

目录



All those colored walls
Mazes give Pacman the blues,
So teach him to search

介绍

在这个项目中，你的吃豆人代理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。您将构建通用搜索算法并将其应用 于 Pacman 场景。该项目的代码由几个 Python 文件组成，其中一些您需要阅读和理解才能完成作业，而其中一些您可以忽略。

您将编辑的文件：

search.py 您的所有搜索算法都将在此文件中。

searchAgents.py 您的所有基于搜索的智能体都在此文件中。

您可能想要查看的文件：

search.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。该文件描述了几种支持类型，如 AgentState、Agent、 Direction 和 Grid。

util.py 用于实现搜索算法的有用数据结构。

您可以忽略的支持文件：

graphicsDisplay.py 吃豆人的图形

graphicsUtils.py 支持吃豆人图形

textDisplay.py Pacman 的 ASCII 图形

ghostAgents.py 控制敌人的智能体

keyboardAgents.py 控制 Pacman 的键盘接口

layout.py 用于读取布局文件并存储其内容的代码

autograder.py 项目自动评分器（利用它来获得更高的分数）

testParser.py 解析自动评分器测试和解决方案文件

testClasses.py 通用自动评分测试类

test_cases/ 包含每个问题的测试用例的目录

searchTestClasses.py 项目 1 特定的自动评分测试类

要编辑和提交的文件：在任务期间，您将填写 *search.py* 和 *searchAgents.py* 的部分内容。您应该将这些文件连同您的代码和注释一起提交。请不要更改其他文件或提交除这些文件之 外的任何我们的原始文件。

评分：您的代码将自动评分以确保技术正确性。请不要更改代码中提供的任何函数或类的名称，否则会对自动评分器造成严重破坏。但是同时请注意，你的代码实现的正确性将决定你最终的分数， 而不是自动评分器的判断 。如有必要，我们将单独审查和评分作业，以确保您的工作获得应有的分数。

学术诚信：我们将根据课堂上其他提交的代码检查您的代码是否存在逻辑冗余。如果您复制别人的代码并进行细微更改提交，我们会知道。我们相信你们都只提交自己的代码，请不要让我们失望！！！如果您执意这么做，将受到最严重的后果。

欢迎来到吃豆人

下载材料（pj1-search.zip）后，解压并切换到目录（search），你应该可以通过在命令行输入以下内容来玩 Pacman 游戏：

```
python pacman.py
```

Pacman 生活在一个闪闪发光的蓝色世界中，这里有曲折的走廊和美味的圆形美食。有效地导航这个世界将是 Pacman 掌握他的领域 的第一步。

searchAgents.py 中最简单的智能体称为 GoWestAgent，它总是向西走（一个简单的反射智能体）。这个智能体偶尔会赢：

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

但是，当需要转弯时，这个智能体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，您的智能体不仅会需要解决tinyMaze，还会解决您想要的任何迷宫。

请注意， *pacman.py* 支持许多参数形式，每个参数都可以用长式（例如， --layout）或短式（例如， -l）表示。您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都出现在commands.txt 中，便于复制和粘贴。在 UNIX/Mac OS X 中，您甚至可以使用

```
bash commands.txt
```

按顺序运行所有这些命令。

Q1（2 分）：使用深度优先搜索找到固定的食物点

在 *searchAgents.py* 中，您会找到一个完全实现的 SearchAgent，它会规划一条穿过 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。 在处理以下问题时，您可能会发现参考对象词汇表（上面导航栏中倒数第二个选项卡）很有用。

首先，通过运行测试SearchAgent是否正常工作：

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

上面的命令告诉 SearchAgent 使用 tinyMazeSearch 作为其搜索算法，该算法在 *search.py* 中实现。 Pacman 应该成功地在迷宫中导航。

现在是时候编写成熟的通用搜索函数来帮助 Pacman 规划路线了！ 您将编写的搜索算法的伪代码可以在课堂幻灯片中找到。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。

重要提示：您的所有搜索功能都需要返回将智能体从起点引导到目标的动作列表。这些动作都必须是合法的移动（有效的方向，不能穿过墙壁）。确保使用 *util.py* 中提供给您的 Stack、 Queue 和 PriorityQueue 数据结构！这些数据结构实现具有与自动评分器兼容所需的特定属性。

提示：每个算法都非常相似。 DFS、BFS、UCS 和 A* 的算法仅在如何管理边缘的细节上有所不同。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，该方法配置有特定算法的队列策略。 （您的实现无需采用这种形式即可获得全部分数）

请在 *search.py* 的 depthFirstSearch 函数中实现深度优先搜索 (DFS) 算法。为了使您的算法完整，请编写 DFS 的图搜索版本，它避免扩展任何已访问的状态。

您的代码运行以下命令，应该能很快找到解：

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

python pacman.py -l mediumMaze -p SearchAgent

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

Pacman 可视化界面将显示探索叠加状态，以及探索的顺序（较亮的红色表示较早的探索）。探索顺序与你预期是否相同？吃豆人真的会在前往目标的路上走遍所有探索过的方块吗？

提示：如果你使用Stack作为数据结构，你的 DFS 算法为 mediumMaze 找到的解决方案的长度应该是 130（假设你按照 getSuccessors 提供的顺序将后继者推到边缘；如果以相反的顺序推出这些节点，将会得到246）。这是成本最低的解决方案吗？

Q2（2 分）：广度优先搜索

请在 *search.py* 的 breadthFirstSearch 函数中实现广度优先搜索 (BFS) 算法。再次，请编写一个图搜索算法，避免扩展任何已经访问过的状态。以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，请检查您的实现方式。

提示：如果 Pacman 对您来说移动太慢，请尝试选项 --frameTime 0 。

注意：如果您已经编写了通用的搜索代码，那么您的代码应该同样适用于八数码搜索问题，无需任何更改。

```
python eightpuzzle.py
```

Q3 (2分)：改变代价函数

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。

通过改变成本函数，我们可以鼓励 Pacman 找到不同的路径。例如，我们可以对鬼怪地区的危险步骤收取更高的费用，或者对食物丰富的地区的步骤收取更少的费用，并且理性的吃豆人智能体应该调整其行为作为回应。

在 *search.py* 的 uniformCostSearch 函数中实现统一成本图搜索算法。我们鼓励您通过 *util.py* 查看一些可能对您的实现有用的数据结构。您现在应该在以下所有三种布局中观察到成功的行为，其中下面的智能体都是 UCS 智能体，它们仅在它们使用的代价函数上有所不同（智能体和代价函数函数是由您编写的）：

```
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：由于它们的指数成本函数，您应该分别为 StayEastSearchAgent 和 StayWestSearchAgent 获得非常低和非常高的路径成本（有关详细信息，请参阅 *searchAgents.py* ）。

Q4（3 分）：A* 搜索

在 *search.py* 中的空函数 *aStarSearch* 中实现A*图搜索。 A* 将启发式函数作为参数。启发式采用两个参数：搜索问题中的状态（主要参数）和问题本身（用于参考信息）。

search.py 中的 *nullHeuristic* 启发式函数就是一个简单的例子。

您可以使用曼哈顿距离启发式（已经在 *searchAgents.py* 中作为 manhattanHeuristic实现）来测试您的 A* 实现，以解决通过迷宫找到通往固定位置的路径的原始问题。

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

您应该看到 A* 找到最佳解决方案的速度比代价一致搜索算法稍快（在我们的实现中扩展了大约 549 vs. 620 个搜索节点，但优先 级的关系可能会使您的数字略有不同）。各种搜索策略在 openMaze 上会发生什么？

Q5（2 分）：找到所有角落

只有在更具挑战性的搜索问题中，A* 的真正威力才会显现出来。现在，是时候制定一个新问题并为它设计一个启发式方法了。

在角落迷宫中，有四个点，每个角落一个。我们新的搜索问题是找到穿过迷宫并触及所有四个角的最短路径（无论迷宫那里是否真的有食物）。请注意，对于一些像 tinyCorners 这样的迷宫，最短路径并不总是首先到达最近的食物！提示：通过tinyCorners的最短路径需要 28 步。

注意：确保在处理问题 5 之前完成问题 2，因为问题 5 建立在您对问题 2 的回答之上。

在 *searchAgents.py* 中实现 CornersProblem 搜索问题。您将需要选择一个状态表示来编码检测是否已到达所有四个角所需的所有信息。现在，您的搜索智能体应该解决：

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

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

要获得满分，您需要定义一个不编码无关信息的抽象状态表示（如幽灵的位置、额外食物的位置等）。特别是，不要使用 Pacman 的 GameState 作为搜索状态。如果您这样做（并且也是错误的,您的代码将非常非常慢。

提示：您在实现中需要参考的游戏状态的唯一部分是 Pacman 的起始位置和四个角的位置。

我们的广度优先搜索实现在 mediumCorners 上扩展了不到 2000 个搜索节点。但是，启发式（与 A* 搜索一起使用）可以减少所需的搜索量。

Q6（2 分）：转角问题：启发式

注意：确保在处理问题 6 之前完成问题 4，因为问题 6 建立在您对问题 4 的回答之上。

在 cornersHeuristic 中为 CornersProblem 实现一个重要的、一致的启发式。

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

注意： AStarCornersAgent是参数

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

的缩写。

可采纳性与一致性：请记住，启发式算法只是获取搜索状态并返回估计最近目标成本的数字的函数。更有效的启发式方法将返回更接近实际目标成本的值。为了拥有可采纳性，启发式值必须是到最近目标的实际最短路径

成本的下限（并且非负）。为了保持一致性，它必须另外满足，如果一个动作的成本为c，那么采取该动作只能导致最多降低 c 的启发值。

请记住，可采纳性不足以保证图搜索的正确性 您需要更强的一致性条件。然而，可采纳的启发式通常也是一致的，特别是如果它们来自问题松弛。因此，通常最容易从头脑风暴可接受的启发式方法开始。一旦你有一个运行良好的可采纳启发式，你也可以检查它是否确实是一致的。保证一致性的唯一方法是使用证明。但是，通常可以通过验证对于您扩展的每个节点，其后继节点的 f 值是否相等或更高来检测不一致性。

此外，如果 UCS 和 A* 曾经返回不同长度的路径，则您的启发式方法是不一致的。 这东西很棘手！

非平凡启发式：平凡启发式（trivial heuristics）是处处返回零（UCS）和计算真实完成成本的启发式。前者不会为您节省任何时间，而后者会使自动评分器超时。您需要一种减少总计算时间的启发式方法，但对于此分配，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）。

评分：您的启发式必须是非平凡的非负一致启发式才能获得分数。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。 根据您的启发式扩展的节点数，您将被评分：

节点数扩展等级	分数
超过 1600	0/2
最多1600	1/2
最多1200	2/2

请记住：如果您的启发式不一致，您将不会获得任何分数，所以要小心！不鼓励使用过多的内存和计算资源，我们希望您的启发式函数是可解释的和优雅的。

Q7（3 分）：吃掉所有的点

现在我们要解决一个困难的搜索问题：用尽可能少的步骤吃掉所有的 Pacman 食物。 为此，我们需要一个新的搜索问题定义，它将食物清理问题形式化： *searchAgents.py* 中的 FoodSearchProblem （为您实现好了）。 解决方案被定义为收集 Pacman 世界中所有食物的路径。对于目前的项目，解决方案不考虑任何鬼怪或能量颗粒；解决方案仅取决于墙壁、常规食物和吃豆子的位置。 如果您正确编写了一般搜索方法，则带有空启发式（相当于代价一致搜索）的A*应该很快无需更改代码即可找到 testSearch 的最佳解决方案（总成本为 7）。

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

注意： AStarCornersAgent是参数

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

的缩写。

您应该会发现，即使对于看似简单的 tinySearch ，UCS也开始变慢。作为参考，我们的实现在扩展 5057 个搜索节点后需要 2.5 秒才能找到长度为 27 的路径。

注意：确保在处理问题 7 之前完成问题 4，因为问题 7 建立在您对问题 4 的回答之上。

在 *searchAgents.py* 中填充 foodHeuristic 使得完成该 FoodSearchProblem 的启发式是一致的。在 trickySearch 面板上尝试你的智能体：

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

我们的 UCS 智能体在大约 13 秒内找到了最佳解决方案，探索了超过 16,000 个节点。

任何非平凡非负一致启发式将获得 1 分。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。根据您的启发式扩展的节点数，您将获得额外的分数：

扩展节点数	分数
超过 15000	1/3
最多 12000	2/3
最多7000	3/3（难）
解决 mediumSearch	不做要求

请记住：如果您的启发式不一致，您将不会获得任何分数，所以要小心！

对于mediumSearch，它非常难，你不需要解决它。但是你可以试一试！在没有并行计算的情况下，您的解决方案时间应该少于 60 秒。

问题 8（2 分）：次优搜索

有时，即使使用 A* 和良好的启发式算法，也很难找到通过所有点的最佳路径。在这些情况下，我们仍然希望快速找到一条相当好的路径。在本节中， 您将编写一个总是贪婪地吃掉最近的点的智能体。在 *searchAgents.py* 中为您实现了 ClosestDotSearchAgent，但它缺少一个找到最近点的路径的关键函数。

在 *searchAgents.py* 中实现函数 findPathToClosestDot 。我们的智能体在一秒钟内解决了这个迷宫（次优!），路径成本为 350：

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

提示：完成 findPathToClosestDot 的最快方法是填写缺少目标测试的 AnyFoodSearchProblem 。然后，使用适当的搜索功能解决该问题。解决方案应该很短！

您的 ClosestDotSearchAgent 不会总能找到穿过迷宫的最短路径。确保您了解原因并尝试提出一个小例子，即重复前往最近的点并不会导致找到吃掉所有点的最短路径。

提交 （**重要的事情!!!!请仔细阅读**）

- 提交您的代码（只有两个文件：*search.py* 和 *searchAgents.py*）和一份简单的报告（最多4页），将它们打包为一个命名为 "***StudentID_YourName_PJ1.zip***" 的 zip 文件，并通过**学习通**平台提交。该报告将概述程序实现中使用的一般方法和启发式方法，以及您认为在阅读您的程序时需要了解的其他设计细节。报告还应包含一些搜索算法的分析，例如启发式函数的可采纳性和一致性等。
- 截止日期为：**2023年10月10日23:59**
- 有问题欢迎咨询：杜梦飞 mfdu22@m.fudan.edu.cn