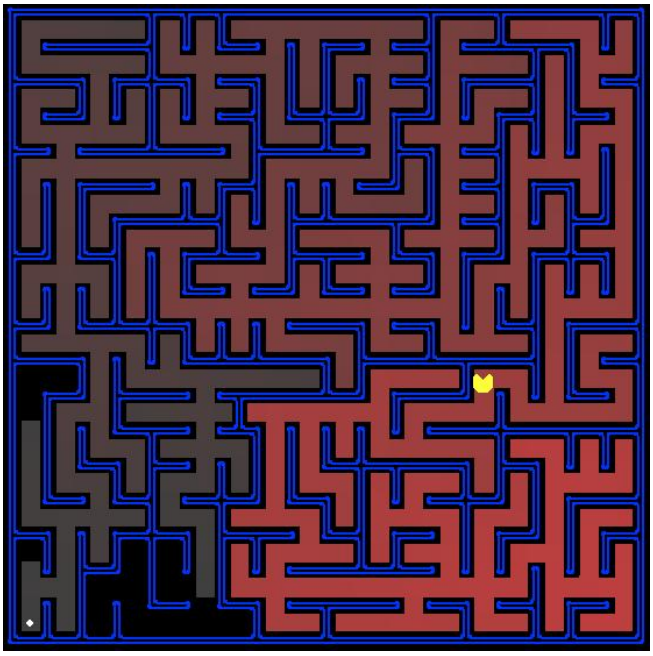
信息与计算机工程学院

人工智能导论

课程项目 1：单人搜索



1、介绍

在这个项目中，你的吃豆人 pacman 将在他的迷宫世界中找到路径，既可以到达特定位置，也可以有效地收集食物。你将构建搜索算法并将其应用于不同的场景。

与项目 0 一样，此项目包括一个自动批改程序，供你在计算机上对答案进行评分。你可以使用以下命令运行：

```
python autograder.py
```

此项目的代码由多个 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。

你需要编辑的文件	
search.py	在这里实现你的搜索算法
searchAgents.py	在这里实现所有你的搜索 Agent
你需要查看的文件	
pacman.py	执行吃豆人游戏的主要程序，描述游戏状态
game.py	吃豆人世界运行的逻辑，有许多重要的定义，例如 AgentState，Agent，Direction 和 Grid
util.py	里面有些可以帮助实现搜索算法的数据结构

剩下的文件，你基本上可以忽视。

需要编辑和提交的文件：你需要完成 search.py 和 searchAgents.py，将你修改的文件和自动批改程序（submission_autograder）产生的结果，以及项目报告一同提交。请不要修改或提交其它文件。

项目评估：你的代码会通过自动批改来判断其正确性，因此请不要修改代码中其它任何函数或者类，否则你会让自动批改程序无法正常运行。然而，你的解题思路和方法是你最终成绩的决定因素。必要的话，我们会查看你的代码来保证你得到应得的成绩。

学术造假：我们会查看你的代码和其它学生提交的代码是否雷同。禁止抄袭了他人代码，或只做简单修改后提交，一旦发现，成绩立马作废，而且会影响到你能否通过此课程。

寻求帮助：当你感到自己遇到了困难，请向你的同学和老师寻求帮助。小组合作、答疑时间、课堂讨论，这些都是用来帮助你的，请积极利用这些资源。设计这些项目的目的是让你更有效地理解和掌握课堂知识，学会如何将理论知识应用于实践，解决实际问题，而不是为考核而考核，或者有意刁难你，所以请尽你所能，完成它们。遇到困难时，向学生老师询问。

2、吃豆人

下载项目代码后，你可以玩一下吃豆人的游戏。

```
python pacman.py
```

吃豆人生活在一个闪亮的蓝色世界里，有扭曲的走廊和美味的圆形美食。有效地驾驭这个世界将是吃豆人掌握他的领域的第一步。

searchAgents.py 中最简单的 Agent 称为 GoWestAgent，它总是向西。该 Agent 偶尔可以赢：

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

但是，当需要转弯时，就有点不妙了：

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

如果吃豆人卡住，你可以通过在终端中键入 CTRL-c 来退出游戏。很快，你的 Agent 不仅会解决微小的迷宫，还会解决任何迷宫。

请注意，pacman.py 支持许多选项，每个选项都可以用长方式（例如，--layout）或短方式（例如，-l）表示。你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
```

此外，此项目中显示的所有命令也显示在 command.txt 中，以便于复制和粘贴。在 UNIX/Mac OS X 中，你甚至可以使用 bash command.txt 按顺序运行所有这些命令。

新的语法

你可能没有见过这种新的语法：

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

这是注释 Python 期望的参数类型。在下面的示例中，a 应该是一个 int——整数，b 应该是 2 个整数的元组，c 应该是任何事物的列表的列表——因此一个 2D 数组，d 本质上与未注释相同，可以由任何东西，e 应该是浮点数。如果没有任何传入，则 e 会设置为 1.0，即：

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

上面的调用合乎类型注释，并且不会为 e 传递任何内容。类型注释旨在作为文档字符串的添加，以帮助你了解函数正在使用的内容。Python 本身并不强制执行这些。在编写自己的函数时，是否要注释类型由你决定；它们可能有助于保持井井有条，或者不是你想花时间做的事情。

3、项目内容

问题 1（3 分）：用深度优先算法找到一个固定的食物

在 searchAgents.py 中，你会发现一个 SearchAgent，它规划出一条穿越 Pacman 世界的路径，然后逐步执行该路径。然而，制定计划的搜索算法没有实现——这是你的工作。

首先，通过运行以下命令来测试搜索 Agent 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，应该帮助吃豆人成功地在迷宫中导航。

现在，你要编写成熟的通用搜索函数来帮助吃豆人规划路线了！搜索算法的伪代码可以在课件中找到。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。

重要说明：你的所有搜索函数都需要返回一个操作列表，这些操作将引导 Agent 从开始到目标。这些行动都必须是合法的动作（有效的方向，不能穿过墙壁）。

重要说明：请确保使用 util.py 中提供给你的堆栈、队列和优先级队列数据结构！这些数据结构实现具有与自动批改程序兼容所需的特定属性。

提示：每种算法都非常相似。DFS、BFS、UCS 和 A* 的算法仅在如何管理边缘（fringe）的细节上有所不同。因此，专注于正确处理 DFS，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，该方法配置了特定于算法的排队策略。（你的实现不需要采用此形式即可获得全部分数）。

在 search.py 的 depthFirstSearch 函数中实现深度优先搜索 （DFS） 算法。若要使算法完整，请编写 DFS 的图形搜索（graph search）版本，以避免扩展任何已访问的状态。

你的代码应快速找到以下答案：

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

吃豆人板将显示探索状态的叠加层，以及探索它们的顺序（更亮的红色表示更早的探索）。勘探顺序是你所期望的吗？吃豆人真的会在通往目标的路上去所有探索的方块吗？

提示：如果使用堆栈作为数据结构，则 DFS 算法为 mediumMaze 找到的解决方案的长度应为 130（前提是按照 getSuccessors 提供的顺序将后继者推到边缘 fringe；如果以相反的顺序推送它们，则可能会得到 246）。这是成本最低的解决方案吗？如果不是，想想深度优先搜索做错了什么。

评分：请运行以下命令以查看你的实现是否通过了所有自动批改的测试用例。

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

问题 2（3 分）：广度优先搜索

在 search.py 的 breathFirstSearch 函数中实现广度优先搜索（BFS）算法。同样，编写一个图形搜索算法，以避免扩展任何已访问的状态。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 是否找到了成本最低的解决方案？如果没有，请检查你的代码。

提示：如果吃豆人移动得太慢，请尝试选项 –frameTime 0。

注意：如果你编写了通用的搜索代码，那么你的代码应该同样适用于八数码问题（8-puzzle）搜索问题，无需进行任何更改。

```
python eightpuzzle.py
```

评分：请运行以下命令以查看你的实现是否通过了所有自动批改的测试用例。

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

问题 3（3 分）：改变成本函数

虽然 BFS 会找到一条最少行动的路径来实现目标，但我们可能希望找到在其他意义上“最佳”的路径。考虑 mediumDottedMaze 和 mediumScaryMaze。

通过改变成本函数，我们可以鼓励吃豆人找到不同的路径。例如，我们可以对幽灵猖獗地区的危险步数收取更高的费用，或者在食物丰富的地区对步数收取更少的费用，理性的吃豆人应该调整其行为作为回应。

在 search.py 年的 uniformCostSearch 函数中实现统一成本图形搜索算法。我们鼓励你在 util.py 中查找一些可能对你的实现有用的数据结构。现在，你应该在以下所有三个布局中观察到成功的行为，其中下面的 Agent 都是 UCS Agent，它们仅在使用的成本函数上有所不同（Agent 和成本函数已经写好了）：

```
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 StayEastSearchAgent 和 StayWestSearchAgent 的指数成本函数，你应该分别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）。

评分：请运行以下命令以查看你的实现是否通过了所有自动批改的测试用例。

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

问题 4（3 分）：A*搜索

在 search.py 中的空函数 aStarSearch 中实现 A* 图形搜索。A* 将启发式函数作为参数。启发式方法采用两个参数：搜索问题中的状态（主参数）和问题本身（用于参考信息）。search.py 中的 nullHeuristic 启发式函数是一个简单的例子。

你可以使用曼哈顿距离启发式（已在 searchAgents.py 年作为曼哈顿启发式实现）来查找穿过迷宫到达一个固定位置的路径问题，并以此测试你的 A* 算法实现。

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

你应该看到 A* 找到最佳解决方案的速度略快于统一成本搜索（在我们的实现中扩展了大约 549 个搜索节点与 620 个搜索节点，但优先级相同的例子可能会使你的数字略有不同）。想一想，在应用在 openMaze 上时，各种不同的搜索策略会发生什么？

评分：请运行以下命令以查看你的实现是否通过了所有自动批改的测试用例。

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

问题 5（3 分）：找到所有的角落

A* 的真正威力只有在更具挑战性的搜索问题中才会显现出来。现在，是时候制定一个新问题并为其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。我们的新搜索问题是找到穿过迷宫的最短路径，该路径触及所有四个角落（无论迷宫是否真的有食物）。请注意，对于像 tinyCorners 这样的迷宫，最短的路径并不总是先到达最近的食物！提示：通过 tinyCorners 的最短路径需要 28 步。

注意：请确保在处理问题 5 之前完成问题 2，因为问题 5 基于你对问题 2 的回答。

在 searchAgents.py 中实现 CornerProblem 搜索问题。你需要选择一个状态表示形式，该表示形式对检测是否已到达所有四个角所需的所有信息进行编码。现在，你的搜索代理应该解决：

```
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，你需要定义一个抽象的状态表示，该表示不会对不相关的信息进行编码（例如鬼魂的位置，额外的食物在哪里等）。特别是，不要使用 Pacman GameState 作为搜索状态。如果你这样做，你的代码会非常非常慢（而且也是错误的）。

提示 1：在实现中需要引用的游戏状态的唯一部分是起始吃豆人位置和四个角的位置。

提示 2：在编写 getSuccessors 时，请将成本为 1 的孩子（children）添加到后续者（successor）的列表中。

我们对 widethFirstSearch 的实现在 mediumCorner 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。

评分：请运行以下命令以查看你的实现是否通过了所有自动批改的测试用例。

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

问题 6（3 分）：角落问题，启发函数

注意：请确保在处理问题 6 之前完成问题 4，因为问题 6 基于你对问题 4 的回答。

在角落启发式中为 CornersProblem 实现一个非直白的、具有一贯性的启发式方法。

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

注意：AStarCornersAgent 是下面式子的一个简洁写法：

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

可接受性与一贯性（Admissibility and consistency）：请记住，启发值只是采用搜索状态并返回估计最接近目标的成本的数字的函数。更有效的启发式方法将返回更接近实际目标成本的值。要达到可接受（admissible），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）。为了保持一贯性（consistency），它还必须满座，如果一个动作有成本 c，那么采取该行动只会导致多于 c 的启发值下降。

请记住，可接受性不足以保证图形搜索的正确性 —— 你需要更强的一贯性条件。然而，可接受的启发式方法通常也是一贯的，特别是如果它们来自问题放松。因此，通常最简单的方法是通过头脑风暴（brainstorm），思考可接受的启发式方法开始。一旦你有一个效果良好的可接受启发式方法，你就可以检查它是否确实是一贯的。保证一贯性的唯一方法是使用证明。但是，通常可以通过验证对于你扩展的每个节点，其后续节点的 f 值是否相等或更高来检测一贯性。此外，如果 UCS 和 A* 返回不同长度的路径，则你的启发式是不一贯的。这个问题通常比较棘手！

非直白的启发式：直白的启发式是随处返回零的启发式（UCS）和计算真实完成成本的启发式。前者不会为你节省任何时间，而后者会使自动批改超时。你需要一种更好的启发式方法，减少总的计算时间，但在我们的题目里，自动批改程序将只检查节点计数（除了强制执行合理的时间限制）。

评分：你的启发式方法必须是非直白的非负一贯启发式方法才能获得任何分数。确保启发式方法在每个目标状态都返回 0，并且永远不会返回负值。我们将根据启发式扩展的节点数量，进行评分：

扩张的节点数	超过 2000	最多 2000	最多 1600	最多 1200
分数	0/3	1/3	2/3	3/3

请记住：如果你的启发式不一贯，你将不会获得任何分数，所以要小心！

评分：请运行以下命令以查看你的实现是否通过了所有自动批改的测试用例。

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

问题 7（4 分）：吃掉所有的食物

现在我们将解决一个困难的搜索问题：尽可能用最少的步子来吃掉所有的食物。为此，我们需要一个新的搜索问题来定义食物清理的问题：searchAgents.py 中的 FoodSearchProblem（已经为你实现了）。解决方案被定义为一条可以收集吃豆人世界中所有食物的路径。对于本项目，解决方案不考虑任何幽灵或动力颗粒；解决方案仅取决于墙壁，常规食物和吃豆人的位置。（当然，鬼魂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。如果你正确编写了常规搜索方法，则无需更改代码，具有空启发式（相当于统一成本搜索）的 A*方法应该可以快速找到最佳解决方案（总成本为 7）。

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

注意：AStarFoodSearchAgent 是下面的简洁表达方式：

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

你应该发现 UCS 开始变慢，即使对于看似简单的 tinySearch 也是如此。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：请确保在处理问题 7 之前完成问题 4，因为问题 7 基于你对问题 4 的回答。

在 searchAgents.py 中填写 foodHeuristic，为 FoodSearchProblem 提供一个一贯的启发式。在 trickySearch 上尝试你的方法：

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

我们的 UCS Agent 在大约 13 秒内找到最佳解决方案，探索超过 16,000 个节点。

任何非直白的非负一贯启发式都将获得 1 分。确保启发式方法在每个目标状态都返回 0，并且永远不会返回负值。根据启发式扩展的节点数量，你将获得额外的分数：

扩展节点数	超过 15000	最多 15000	最多 12000	最多 9000	最多 7000
分数	1/4	2/4	3/4	4/4	5/4

请记住：如果你的启发式不一贯，你将不会获得任何分，所以要小心！你能在短时间内解决中等搜索吗？如果是这样，我们要么非常厉害，要么你的启发式不一贯。

评分：请运行以下命令以查看你的实现是否通过了所有自动批改的测试用例。

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

问题 8（3 分）：次优搜索

有时，即使使用 A* 和良好的启发式方法，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，你将编写一个吃豆人，它总是贪婪地吃掉最近的点。ClosestDotSearchAgent 在 searchAgents.py 中已经为你实现，但它缺少一个查找到最近点的路径的关键函数。

在 searchAgents.py 中实现函数 findPathToClosestDot。我们的吃豆人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 AnyFoodSearchProblem，它缺少目标测试。然后，使用适当的搜索功能来解决这个问题。解决方案应该很短！

你的 ClosestDotSearchAgent 并不总是能找到通过迷宫的最短路径。为了确保你明白为什么，你可以尝试想出一个小例子，重复去最近的点不会导致找到吃掉所有点的最短路径。

评分：请运行以下命令以查看你的实现是否通过了所有自动批改的测试用例。

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

4、项目报告

简要清晰地描述完成项目时遇到的困难，采用的解决方法，提出改进意见，和总结每个小组成员的贡献。

5、提交

在提交你的解答之前，你需要通过执行 submission_autograder.pyc 来产生几个文件。在运行这个程序之前，你必须确认所有与 autograde 有关的文件都处在原始状态，没有做过任何的改动。假如你编辑过任何 autograde 的文件，请重新下载一份项目代码，仅仅替换你作解答的文件，否则运行 submission_autograder.pyc 将无法通过。

此外，submission_autograder.pyc 要在 Python 3.6（准确的说是 3.6.13，你可以用 Anaconda 来安装正确的 Python 版本）下执行，否则会报错。

最后，submission_autograder.pyc 需要用 rsa 库来给你的成绩加密，假如你没有的话，请用下面的命令安装 rsa 库。

```
conda install -c conda-forge rsa
```

或者用下面的命令，假如你没有 conda。

```
pip install rsa
```

进到你的 search 文件夹里，执行以下命令：

```
python submission_autograder.pyc
```

成功执行后，该命令会输出你的各个题目的得分和最后总分，并在 grade 文件夹里会生成一个 log 文件和一个 token 文件。确认该分数和你自己运行 autograder.py 得到的分数相同后，将整个 grade 文件夹和你修改过的文件（其它没有修改过的，例如 autograder.py，不需要）以及项目报告，打包生成一个以你的组号命名的 zip 文件，一并提交上来。