## Project 1: Search in Pacman

|  |  |
| --- | --- |
| 姓名 | 刘胡超 |
| 学号 | 1163710222 |
| 班号 | 1603002 |
| 电子邮件 | [1249490233@qq.com](mailto:1249490233@qq.com) |
| 手机号码 | 18846196338 |

# 实验任务简述

通过对网站下载的项目压缩包search.zip中search.py以及searchAgent.py进行内容填充，使得完成下列的8个任务：

* 深度优先搜索
* 广度优先搜索
* 一致代价搜索
* A\*搜索
* 角落问题：代表性
* 角落问题：启发式
* 吃掉所有的点：启发式
* 次优搜索

# 实验环境配置及工具：

1.python2.7.10版本

2.pycharm 2017 社区版

3.search.zip压缩包

# 实验过程(下面对每一个任务要求进行简要的叙述)

任务一：深度优先搜索

1.任务简述：

通过对压缩包之中的pacman.py,game.py以及util.py的阅读，从而获取agent的工作原理，通过编写深度优先搜索，实现让agent吃掉指定豆子的任务。

2.任务完成过程：

建立stack,用于存储当前的节点current\_state，并且对current\_state的可拓展节点进行压栈操作，并且对该节点进行递归调用。直到获得目标状态节点，返回路径直接返回

3.程序代码：

|  |
| --- |
| **def** depthFirstSearch(problem):  start\_state = problem.getStartState() path = [] visited\_state = [] **from** game **import** Actions  **def** dfs(state,action):  visited\_state.append(state)  **if** problem.isGoalState(state):  **return** True  **else** :  **for** state\_tuple **in** problem.getSuccessors(state):  **if** state\_tuple[0] **not in** visited\_state:  path.append(state\_tuple[1])  **if** dfs(state\_tuple[0],state\_tuple[1])==True :  **return** True  path.pop()  **return** False  dfs(start\_state,None) **return** path |

任务二：广度优先搜索

1.任务简述：

通过对压缩包之中的pacman.py,game.py以及util.py的阅读，从而获取agent的工作原理，通过编写广度优先搜索，实现让agent吃掉指定豆子的任务。

2.任务完成过程：

建立queue,对节点进行存储，queue之中存储的元素为自定义类node，其中包含state(用于记录位置)，list(用于存储路径),cost(用于存储代价)，通过先进先出的队列从而实现对代价相同的节点的遍历，通过循环，使得当得到目标节点时，直接返回路径。

3.程序代码：

|  |
| --- |
| **class** Node:  **def** \_\_init\_\_(self,state,list,cost=0):  self.state = state  self.list = list  self.cost = cost  **def** breadthFirstSearch(problem):start\_state = problem.getStartState()  visited\_state = []   queue = util.Queue()   start\_node = Node(start\_state, [])  queue.push(start\_node)   **def** bfs(queue):  **while not** queue.isEmpty():  state\_node = queue.pop()  state\_current = state\_node.state  list\_source = state\_node.list[:]  visited\_state.append(state\_current)    **if** problem.isGoalState(state\_current):  **return** state\_node  **else**:  **for** state\_tuple **in** problem.getSuccessors(state\_current):  **if** state\_tuple[0] **not in** visited\_state :  temp\_list = list\_source[:]  temp\_list.append(state\_tuple[1])  visited\_state.append(state\_tuple[0])  new\_node = Node(state\_tuple[0], temp\_list)  queue.push(new\_node)  **else** :  **continue** w = bfs(queue)  **return** w.list |

任务三：一致代价搜索

1.任务简述：

通过对压缩包之中的pacman.py, game.py以及util.py的阅读，从而获取agent的工作原理，通过编写一致代价搜索，实现让agent吃掉指定豆子的任务。

2.任务完成过程：

由于一致代价搜索对代价较小的节点进行拓展，因此在广度优先搜索的基础上建立Priorityqueue, 对节点进行存储，与广度优先搜索不同的是需要用到node之中的代价cost,利用已经设计好的update对queue之中的元素进行插入以及更新。保证每次拓展的节点都是代价最小的节点，并且除了goal状态以外，其他状态一旦被访问就无法再次进入队列。通过先进先出的队列从而实现对代价相同的节点的遍历，通过循环，使得当得到目标节点时，直接返回路径。

3.程序代码：

|  |
| --- |
| **class** Node:  **def** \_\_init\_\_(self,state,list,cost=0):  self.state = state  self.list = list  self.cost = cost  **def** uniformCostSearch(problem): **from** game **import** Actions  **from** game **import** Directions  start\_state = problem.getStartState()  visited\_state = []  queue = util.PriorityQueue()  start\_node = Node(start\_state, [], 0)  queue.push(start\_node,0)  **def** ufs(queue):  **while not** queue.isEmpty():  state\_node = queue.pop()  state\_current = state\_node.state  pre\_cost = state\_node.cost  list\_source = state\_node.list[:]  visited\_state.append(state\_current)  **if** problem.isGoalState(state\_current):  **return** state\_node  **for** state\_tuple **in** problem.getSuccessors(state\_current):  **if** state\_tuple[0] **not in** visited\_state :  temp\_list = list\_source[:]  temp\_list.append(state\_tuple[1])  **if not** problem.isGoalState(state\_tuple[0]):  visited\_state.append(state\_tuple[0])  new\_node = Node(state\_tuple[0],temp\_list,pre\_cost + state\_tuple[2])  queue.update(new\_node,new\_node.cost)   w = ufs(queue)  **return** w.list |

任务四：A\*搜索

1.任务简述：

通过对压缩包之中的pacman.py, game.py以及util.py的阅读，从而获取agent的工作原理，通过编写A\*搜素算法，实现让agent吃掉指定豆子的任务。

2.任务完成过程：

由于已经完成了一致代价搜索，而两者都是属于best-first算法，最为重要的是A\*算法多了一个启发式为h(x),因此完成十分简单，只需要对原先cost的计算进行处理即可。

3.程序代码：

|  |
| --- |
| **class** Node:  **def** \_\_init\_\_(self,state,list,cost=0):  self.state = state  self.list = list  self.cost = cost  **def** aStarSearch(problem, heuristic=nullHeuristic):start\_state = problem.getStartState()  visited\_state = []  queue = util.PriorityQueue()   start\_node = Node(start\_state, [],0)  queue.push(start\_node, heuristic(start\_state,problem))   **def** ass(queue):  **while not** queue.isEmpty():  state\_node = queue.pop()  state\_current = state\_node.state  pre\_cost = state\_node.cost  list\_source = state\_node.list[:]  visited\_state.append(state\_current)  **if** problem.isGoalState(state\_current):  **return** state\_node  **for** state\_tuple **in** problem.getSuccessors(state\_current):  **if** state\_tuple[0] **not in** visited\_state :  temp\_list = list\_source[:]  temp\_list.append(state\_tuple[1])  **if not** problem.isGoalState(state\_tuple[0]):  visited\_state.append(state\_tuple[0])  new\_node = Node(state\_tuple[0], temp\_list, pre\_cost + state\_tuple[2])  cost = new\_node.cost+heuristic(state\_tuple[0],problem)  queue.update(new\_node,cost)   w = ass(queue)  **return** w.list |

任务五：角落问题：代表性

1.任务简述：

通过对压缩包之中的pacman.py, game.py以及util.py的阅读，从而获取agent的工作原理，编写问题的基本结构，从而让agent能够访问迷宫的四个不同的角落。

2.任务完成过程：

前所未见的程序要求，通过对程序的阅读，发现其实与之前的吃豆子问题相比，需要修改的无外乎两处。1.对所剩角落的记录，针对这一点，我使用的是对state进行包装改造，使得它能够同时存储state信息以及corn\_list信息2.对终止状态的判断，正如正如之前所说的，如果state能够记录信息的话，那么只需要在角落时将state的corner\_list进行删除元素操作即可，

3.程序代码：

|  |
| --- |
| **def** \_\_init\_\_(self, startingGameState):  listx = list(self.corners[:]) **if** self.startingPosition **in** listx:  listx.remove(self.startingPosition) self.startState = (self.startingPosition,listx)  **def** getStartState(self):**return** self.startState  **def** getSuccessors(self, state):  **for** action **in** [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:x,y = state[0]  dx, dy = Actions.directionToVector(action)  nextx, nexty = int(x + dx), int(y + dy)  **if not** self.walls[nextx][nexty]:  nextState = (nextx, nexty)  cost = self.costFn(nextState)  corner\_list = state[1][:]  **if**(nextState **in** corner\_list):  corner\_list.remove(nextState)  nextState = (nextState,corner\_list)  successors.append( ( nextState, action, cost) )   self.\_expanded += 1 *# DO NOT CHANGE*  **return** successors |

任务六：角落问题：启发式

1.任务简述：

通过对压缩包之中的pacman.py, game.py以及util.py的阅读，从而获取agent的工作原理，编写启发式，从而让agent能够通过A\*算法访问四个角落并且路径最短

2.任务完成过程：

前所未见的程序要求，但是由于题目有提示，每次都走最近的点是得不到最优结果的，而结合启发式的要求，h(n)小于真实的代价，这就使得边长width成为一个很好的帮助，因此，每次选择最近的，而启发式则是与corner\_list之中元素的数量相关，剩余三个顶点，则最少需要2条边的代价，具体的表示在下面的代码有显示。

3.程序代码：

|  |
| --- |
| **def** cornersHeuristic(state, problem):corners = problem.corners *# These are the corner coordinates* walls = problem.walls *# These are the walls of the maze, as a Grid (game.py)* min = 1000  min\_state = None  **def** distance(state,state\_corner):  **return** abs(state[0][0]-state\_corner[0]) + abs((state\_corner[1] - state[0][1]))   **for** state\_corner **in** state[1]:  **if** distance(state,state\_corner)<min:  min\_state = state\_corner  min = distance(state,state\_corner)  **if** min ==1000:  min = 0  **if** len(state[1])==0:  **return** min + (len(state[1]))\*(problem.walls.width-5)  **return** min + (len(state[1])-1)\*(problem.walls.width-5)*# Default to trivial solution* |

任务7：

未完成，因此不进行叙述

任务8：次优搜索

1.任务简述：

通过对压缩包之中的pacman.py, game.py以及util.py的阅读，从而获取agent的工作原理，通过对函数的调用，从而使得程序实现次优搜索。

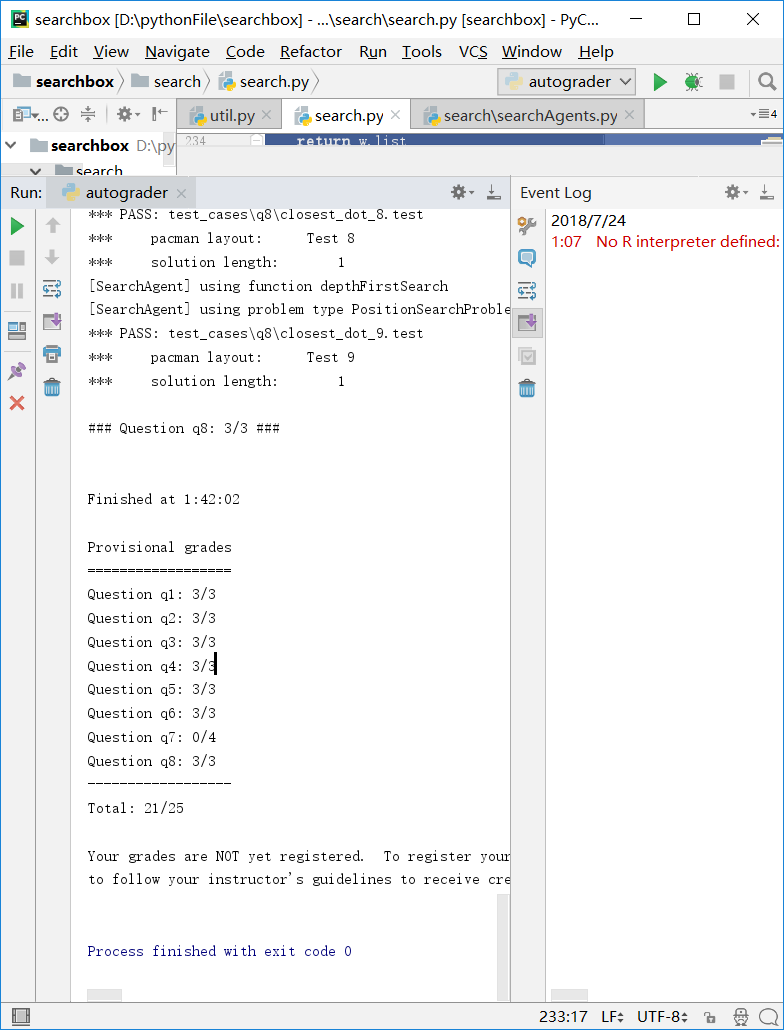
2.任务完成过程：

感觉题目要求说的十分玄学，其实只需要对特定的函数极限调用就可以实现要求，下面展示程序的代码

3.程序代码：

|  |
| --- |
| **def** findPathToClosestDot(self, gameState):  **return** search.breadthFirstSearch(problem)  **def** isGoalState(self, state):  **return** state **in** self.food.asList() |

# 实验结果



# 实验感想：

真的很难，主要是读程序花了很多时间，但是自己阅读能力不够的硬伤也暴露出来，很高兴能够写这一次的实验，使我对python的使用更加熟悉。