

哈爾濱工業大學

人工智能实验报告

题目	搜索策略
专业	自然语言处理
学号	1172510217
学生	张景润
指导教师	李钦策
同组人员	李大鑫、李明哲、安照崇

目录

一、 问题简介.....	3
1. 问题解释.....	3
2. 形式化描述.....	3
3. 解决方案原理.....	4
二、 算法介绍.....	5
1. 算法伪代码.....	5
2. 所用方法一般介绍.....	5
三、 算法实现.....	7
1. 实验环境与问题规模.....	7
2. 数据结构.....	7
3. 问题总结果.....	8
4. 系统中间结果.....	8
四、 总结讨论.....	10
1. 实验总结.....	10
2. 实验启发.....	10
附录：A*算法证明.....	11
1. 可纳性.....	11
2. 最优性（信息性）.....	11
参考文献.....	11

一、问题简介

1. 问题解释

- (1) 要求采用且不限于课程第四章内各种搜索算法来编写一系列吃豆人程序，解决问题 1-8，问题包括到达指定位置以及有效的吃豆等
- (2) 参考 pacman.py 等对 search.py 和 searchAgents.py 进行修改以满足实验要求
- (3) 问题 1-4 需要实现面向应用的搜索算法：深度优先 DFS、广度优先 BFS、代价一致 UCS 和 A*搜索算法
- (4) 问题 5-8 是基于问题 1-4 实现算法的具体应用
 - (a) 利用 BFS 算法以找到所有的角落为目标的 **CornersProblem**
 - (b) 利用 CornersProblem 角落问题设计启发式函数 **cornersHeuristic**
 - (c) 利用 A*算法实现以吃完所有豆子为目标的启发函数 **foodHeuristic**
 - (d) 利用 UCS 实现以吃掉最近豆子为目标的 **ClosestDotSearchAgent**

2. 形式化描述

- (1) 前提：初始状态已知、问题环境已知
- (2) 推得：由前提可以得到每一步行动后的状态
- (3) 描述：搜索问题可用 5 个组成部分来形式化描述问题的输入

state 初始状态	吃豆人的初始位置(x,y)
action(state) 行动	给定状态下吃豆人的合理移动
goal(state) 目标测试	给定状态判断吃豆人是否达到目标
action(state+action) 转移模型	给定状态、行动下吃豆人的后继状态
cost(state-action-state) 路径耗散	给定状态下动作得到吃豆人的代价

- (4) 问题 1-4：搜索策略的差别不在于形式化描述不同，而在于 Open 表排序不同引起的当前状态的下一个状态 **nextState** 的不同

算法描述	algorithm(problem): actions
算法输入	上述 5 部分组成的数据结构
最终结果	初始状态到目标状态的行动序列，即吃豆人动作列表 actions
最优解	所有解中总耗散最小的，即吃豆人总代价最小的 actions

- (5) 问题 5-8：其与 1-4 形式化描述一致：问题前提、问题输入和最优解一致；下表是问题 5-8 中的关键不同之处

问题及关键不同	相关描述
5: 目标测试	是否已到达所有角落
6: 启发函数	重新设计抵达四个角落的合理启发式函数
7: 目标测试	是否吃完了所有豆子
8: 搜索策略	广度优先 BFS 或代价一致搜索 UCS

3. 解决方案原理

3.1 问题 4 启发式函数原理证明

描述	启发式函数为曼哈顿距离 $h(n) = \text{manhattanHeuristic}$
可纳性	吃豆人的动作仅包括上、下、左和右，因此节点 n 和 S_g 之间的可达距离一定有： $h^*(n) \geq a+b$ ， a 和 b 分别为横、纵向距离，可纳性得证
一致性	即单调性。问题中每一步代价均为 1 且只可横或纵向移动，因此对节点 n_i 及其后继 n_{i+1} ， $h(n_i) = h(n_{i+1}) + 1$ 或 $h(n_i) < h(n_{i+1}) + 1$ 恒成立，即满足 $h(n_i) \leq h(n_j) + c(n_i, n_j)$ ，一致性得证

3.2 问题 6 启发式函数原理证明

为实现一个非平凡、可纳且一致的启发式函数，需要利用更多启发信息，在满足可纳性的情况下尽可能增大启发式函数的值。这里进行分类讨论

注意：启发式函数的最大值应不大于问题在无墙壁状态下的最小代价

- (1) 剩余 0 个豆子：问题结束，启发式函数返回值为 0
- (2) 剩余 1 个豆子 b_0 ：问题化为单目标搜索问题，选用曼哈顿启发函数
 - (a) 可纳性：参见问题 4 对于曼哈顿启发函数的可纳性证明
 - (b) 一致性：参见问题 4 对于曼哈顿启发函数的一致性证明
- (3) 剩余 2 个豆子 b_0 和 b_1 ：问题转化为两个单目标问题，启发式函数返回值为 $\min(\text{manhattanHeuristic}(b_i)) + \text{manhattanHeuristic}(b_0, b_1)$ ，其中 i 取 0, 1
 - (a) 可纳性：显然问题可分解为 2 个子问题，吃掉一个豆子并从一个豆子出发到达另外一个豆子位置。当不考虑墙壁存在时，代价下确界为 $d_1 + d_2$ 其中 d_1 为吃豆人到较近豆子的曼哈顿距离， d_2 为豆子之间的曼哈顿距离。因此选用的启发式函数是代价的一个下限，满足可纳性
 - (b) 一致性：由于该问题的 2 个子问题均选用曼哈顿函数作为代价的估计，因此结合问题 4 对于曼哈顿启发函数的一致性证明可得，满足一致性
- (4) 剩余 3 个豆子 b_0 、 b_1 和 b_2 ：关键在于如何吃第一个豆子，启发函数值为 $\text{manhattanHeuristic}(b_0) + \text{manhattanHeuristic}(b_0, b_1) + \text{manhattanHeuristic}(b_1, b_2)$
 - (a) 可纳性：3 个豆子呈现三角形，易得对任意吃豆人位置，其必须先吃非直角顶点位置豆子；当吃完该豆子后，余下的距离下界为定值（三角形的两直角边之和）。若无墙壁，代价下确界为 $d_0 + d_1 + d_2$ ，其中 d_0 为吃豆人到非直角顶点的 2 个豆子的较近曼哈顿距离， $d_1 + d_2$ 为两直角边之和
 - (b) 一致性：由于 3 个子问题选择的启发式函数值均为曼哈顿函数的值，因此结合问题 4 的一致性证明可得，满足一致性
- (5) 剩余 4 个豆子 b_0 、 b_1 、 b_2 、 b_3 ：问题转化为吃豆人到第一个豆子的代价+吃豆人在第一个豆子的位置到其余 3 个豆子的代价。启发式函数值为 $\min(\text{manhattanHeuristic}(b_i)) + 2 * (\text{width} - 1) + \text{length} - 1$
 - (a) 可纳性：当吃完一个豆子后，吃掉剩余三个豆子的代价在无墙壁情况下是一致的，因此问题关键是如何吃第一个豆子。显然选择吃最近的一个豆子。无墙壁状态下，代价下确界刚好为此启发式函数的值，因此当存在墙壁时，启发式函数的值确实是代价下界，满足可纳性

- (b) 一致性：由于 4 个子问题的启发式函数值均选用曼哈顿函数，因此结合问题 4 的一致性证明可得，满足一致性

3.3 问题 7 启发式函数原理证明

描述	启发式值为距离吃豆人当前位置实际代价最大的豆子的代价
可纳性	吃豆人吃掉所有豆子的代价下界显然不小于吃豆人吃掉最远的豆子的代价值，因此启发函数一定满足可纳性
一致性	即单调性证明。问题中每一步代价均为 1 且只可横或纵向移动，因此对于节点 n_i 及其后继 n_{i+1} ， $h(n_i) \leq h(n_{i+1}) + 1$ 恒成立，一致性得证

二、算法介绍

1. 算法伪代码

```

function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed  $\leftarrow$  an empty set
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe  $\leftarrow$  INSERT(child-node, fringe)
      end
    end
  end
end

```

2. 所用方法一般介绍

2.1 问题 1-4：通用方法介绍

不同搜索策略仅仅在于 Open 表中节点排序不同，所采用的通用算法是一致的

(1) 依据第一章第 2 节的形式化描述，介绍问题 1-4 的通用解决方案

- (a) 将问题初状态加入 open 表中：open_lst.push(startState)
- (b) 循环(c)(d)(e)直到 open 表为空：loop(not open_list.isEmpty())
- (c) 取一个 open 表节点得到其状态和动作列表
- (d) 若是目标节点则成功返回对应动作列表：isGoalState(state)
- (e) 若该节点不在 closed 表中，则将其加入并将所有后继节点加入 open 表

(2) 问题 1-4 使用 util 工具类中相同的 push 和 pop 操作

- (a) 1-2 的 Stack 和 Queue 通过取出 Open 列表中不同位置的元素达到后进先出和先进先出的目的；push 函数参数为 1 个 item，这里是二元组(状态，动作列表)
- (b) 3-4 实现 push 函数传入参数一致性，使用 PriorityQueueWithFunction

```

priorityFunction = Lambda item: problem.getCostOfActions(item[1])
return general_search(problem, util.PriorityQueueWithFunction(priorityFunction))

```

lambda 表达式用于 UCS 代价计算及其通用搜索方法的调用

```
priorityFunction = lambda item: problem.getCostOfActions(item[1]) + heuristic(item[0], problem)
return general_search(problem, util.PriorityQueueWithFunction(priorityFunction))
```

lambda 表达式用于 A*代价计算及其通用搜索方法的调用

2.2 问题 5：找到所有的角落

状态描述	(位置, 4 个角落食物状态真值列表)
目标状态	4 个角落的食物是否全部已被吃掉(True, True, True, True)
得到后继	后继节点需要判断是否更新角落食物存在情况并对应更新

2.3 问题 6：启发式角落问题

启发函数值计算方法及其可纳性、一致性已在第一章第 3 节进行了详细的说明，这里主要结合代码进行说明

(1) 剩余 0 个豆子：问题结束，达到目标状态，返回启发式值为 0

```
if food_num == 0: return 0
```

(2) 剩余 1 个豆子：问题转化为状态到 1 个豆子的曼哈顿距离

```
elif food_num == 1: return util.manhattanDistance(coord, foodList[0])
```

(3) 剩余 2 个豆子：问题转化为先吃距离较近的 1 个豆子，然后再吃剩余 1 个豆子的问题，后者即为问题(2)

```
return min(util.manhattanDistance(coord, foodList[0]), util.manhattanDistance(
    coord, foodList[1])) + util.manhattanDistance(foodList[0], foodList[1])
```

(4) 剩余 3 个豆子：问题转化为先吃距离较近的 1 个非直角顶点的豆子，然后再吃剩余的 2 个豆子，后者即为问题(3)

```
man_dist_lst = min(util.manhattanDistance(coord, leftsidefood[0]),
    util.manhattanDistance(coord, leftsidefood[1]))
return man_dist_lst + top - 1 + right - 1
```

(5) 剩余 4 个豆子：问题转化为先吃距离最近的 1 个豆子，然后再吃剩余的 3 个豆子，后者即为问题(4)

```
man_dist_lst = min(util.manhattanDistance(coord, foodList[0]),
    util.manhattanDistance(coord, foodList[1]),
    util.manhattanDistance(coord, foodList[2]),
    util.manhattanDistance(coord, foodList[3]))
return man_dist_lst + 2 * (width - 1) + length - 1
```

2.4 问题 7：吃掉所有豆子

启发式函数值的计算方法及其可纳性、一致性证明已在第一章第 3 节进行了详细的说明，这里主要结合代码进行说明

(1) 目标状态：问题终止条件为剩余豆子数目为 0


```
def isGoalState(self, state):
    return state[1].count() == 0
```

- (2) 后继节点：得到后继节点的过程中需要令合法后继节点的豆子存在状态置为 False，表示已经吃掉豆子
- (3) 启发式值：选取剩余豆子中距离当前节点最大实际距离作为启发式函数值

```
for x in range(foodGrid.width):
    for y in range(foodGrid.height):
        if foodGrid.data[x][y] == True:
            dis = mazeDistance((x, y), position, problem.startingGameState)
            if ans < dis:
                ans = dis
```

2.5 问题 8：次最优搜索

- (1) 目标状态：吃到一个豆子作为目标状态（注：这里的目标状态并非整个问题的目标状态，而是 AnyFoodSearchProblem 的目标状态）

```
def isGoalState(self, state):
    
    x, y = state
    return self.food[x][y]
```

- (2) 吃掉最近豆子：为提高搜索速度，选择最近豆子作为第一个目标不失为一个好办法。由于 UCS 在拓展节点的过程中，总是可以拓展到最近的一个豆子，因此可以在 AnyFoodSearchProblem 问题中选用 UCS 搜索策略不断搜索最近的一个豆子。特殊地，当行动代价均为 1 时，UCS 等价于 BFS

```
problem = AnyFoodSearchProblem(gameState)
return search.ucs(problem)
```

三、算法实现

1. 实验环境与问题规模

问题规模 全状态空间	(S,F,G): S 为所有初状态的集合; G 为所有目标状态的集合; F 为所有操作的集合 其中 S 和 G 的大小为整个搜索地图, F 为上下左右移动
开发环境	PyCharm 2019.3 Community Edition
操作系统	Windows 10
Python	2.7

2. 数据结构

- (1) 问题 1 到 4 中：list 顺序列表用于 Closed 表结构，保存已拓展节点
- (2) 问题 5 和 7 中：Successors 数据结构为三元组(State, 动作, 代价)

问题 1	util.Stack 用于 Open 表结构，先进后出
问题 2	util.Queue 用于 Open 表结构，先进先出
问题 3	util.PriorityQueueWithFunction 用于 Open 表结构，取出代价小者

问题 4	util. PriorityQueueWithFunction 用于 Open 表结构, 取出代价小者
问题 5	State 数据结构: 二元组(位置, [True,False,True,True])
问题 6	food_lst 数据结构: 角落食物列表, 用于保存存在食物的角落
问题 7	State 数据结构: 二元组(位置, 食物列表)
问题 8	State 数据结构: 二元组(位置, 食物列表)

3. 问题总结结果

(1) 实验结果: 如下图, 拿到基础的 25 分, 同时拿到了问题 7 中的附加分

Provisional grades =====	Question q5: 3/3
Question q1: 3/3	Question q6: 3/3
Question q2: 3/3	Question q7: 5/4
Question q3: 3/3	Question q8: 3/3
Question q4: 3/3	-----
	Total: 26/25

(2) 问题 6 和 7 分析: 得益于设计优秀的启发函数, 拓展节点为: 741 远小于 1200 个, 5137 远小于 7000 个; 得分分别为 3/3 和 5/4

Number of nodes expanded	Grade	Number of nodes expanded	Grade
more than 2000	0/3	more than 15000	1/4
at most 2000	1/3	at most 15000	2/4
at most 1600	2/3	at most 12000	3/4
at most 1200	3/3	at most 9000	4/4 (full credit; medium)
		at most 7000	5/4 (optional extra credit; hard)

4. 系统中间结果

4.1 问题 1-4

(1) 测试结果: 下面是问题 1-4 在 mediumMaze 下的路径代价以及拓展的节点数

问题 (mediumMaze)	1	2	3	4
代价/拓展节点数	130/146	68/269	68/269	68/221

(ai_lab) C:\Users\86185\Desktop\lab2>python pacman.py [SearchAgent] using function dfs [SearchAgent] using problem type PositionSearchProblem Path found with total cost of 130 in 0.0 seconds Search nodes expanded: 146	(ai_lab) C:\Users\86185\Desktop\lab2>python pacman.py [SearchAgent] using function bfs [SearchAgent] using problem type PositionSearchProblem Path found with total cost of 68 in 0.0 seconds Search nodes expanded: 269
---	--

(ai_lab) C:\Users\86185\Desktop\lab2>python pacman.py [SearchAgent] using function ucs [SearchAgent] using problem type PositionSearchProblem Path found with total cost of 68 in 0.0 seconds Search nodes expanded: 269	(ai_lab) C:\Users\86185\Desktop\lab2>python pacman.py [SearchAgent] using function astar and heuristic manhattan [SearchAgent] using problem type PositionSearchProblem Path found with total cost of 68 in 0.0 seconds Search nodes expanded: 221
--	--

左上为 DFS, 右上边为 BFS, 左下为 UCS, 右下为 A*

(2) 结果分析 (以 mediumMaze 为例)

- (a) **DFS:** 由 DFS 的通用性及可纳性, 问题 1 有解, 其路径代价为 130。但非最优解, 对应代价 68; 拓展节点数较少, 为 146, 小于 BFS 的 269
- (b) **BFS:** 由 BFS 的可纳性, 问题 2 可找到最优解, 其路径代价为 68, 但拓展节点数目为 269, 大于 DFS 的拓展节点数

- (c) **UCS**: 当路径代价为单位代价时, UCS 等价于 BFS。因此 UCS 在此条件下, 找到了最优解且拓展节点数较多
- (d) **A***: A*算法一定可找到最优解 (可纳性); 启发式函数的一致性决定 A*算法的单调性且曼哈顿函数是一致的, 因此问题 4 拓展节点较少; 启发函数值越接近真实值, 拓展节点数越少, 效率越高 (信息性)

算法	可找到解	得到最优解	拓展节点数目
DFS	是	不一定	实验表现为最少
BFS	是	一定	数目>DFS
UCS	是	一定	数目>DFS
A*	是	一定	DFS<数目<BFS, UCS

4.2 问题 5-6

- (1) **测试结果**: 下面是问题 5-6 在 mediumCorners 下的路径代价及拓展的节点数

问题 (mediumCorners)	5	6	其他尝试
代价/拓展节点数	106/1966	106/741	106/1136

- (2) **结果分析** (以 mediumCorners 为例)

- (a) 问题 5 选用 BFS 算法, 一定可找到最优解, 但拓展节点数目较多
- (b) 问题 6 选用 A*算法, 且已在第一章 3.2 节中证明选择的启发函数满足可纳性和一致性, 因此找到了最优解且拓展节点较少

- (3) **其他满足一致性的启发式函数尝试**

- (a) **描述**: 尝试选用新的启发式函数进行测试, 关键为选择剩余豆子中位置距离当前位置的曼哈顿距离最大者作为启发式函数值
- (b) **可纳性与一致性**: 同曼哈顿函数证明, 可知该函数满足可纳性与一致性
- (c) **效果**: 在找到最优解的情况下, 拓展节点数为 1136 个, 少于 BFS 的拓展节点数, 但是多于选择原启发式函数得到拓展节点数
- (d) **原因分析**: 由于 A*算法具有最优性或者说信息性, 即选择的启发式函数在满足可纳性的条件下值越大, 拓展节点就会越少, 利用的启发性信息越多。因此, 原启发式函数效果优于所尝试的启发式函数

4.3 问题 7

- (1) **测试结果**: 下面是问题 7 在 trickySearch 下的路径代价及拓展的节点数

问题 (trickySearch)	7	其他尝试
代价/拓展节点数	60/4137	60/9551

- (2) **结果分析** (以 trickySearch 为例): 该问题选用已在第一章 3.3 节证明的满足一致性和可采纳性的启发式函数的 A*算法, 所以一定得到最优路径且拓展的节点数为 4137 个, 达到附加分要求 (拓展数目<7000)

- (3) **其他满足一致性的启发式函数尝试**

- (a) **描述**: 尝试选用新的启发式函数进行测试, 选择所有豆子中距离当前位置的曼哈顿距离最大者作为启发式函数值
- (b) **可纳性与一致性**: 同曼哈顿函数证明, 可知该函数满足可纳性与一致性
- (c) **效果**: 找到最优解但拓展节点数为 9551, 大于 4137 (原方法)

- (d) **原因分析**: 由于 A*算法具有信息性, 即选择的启发式函数在满足可纳性的条件时值越大, 所拓展的节点就会相对越少, 利用的启发性信息越多。因此, 我们组的原启发式函数明显优于现在所尝试的启发式函数

4.4 问题 8

- (1) **测试结果**: 在 bigSearch 下路径代价为 350
- (2) **结果分析** (以 bigSearch 为例): 在第二章 3.7 节中已对实现方法进行了详细的介绍。由于 UCS 在拓展节点的过程中, 总是可以拓展到最近的一个豆子, 因此 ClosestDotSearchAgent 中选用 UCS 一定可以得到一个解 (若路径代价均为单位代价, 则可选用 BFS), 且解代价并不是特别高, 为 350

```
(ai_lab) C:\Users\86185\Desktop\lab2>python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 350.
```

四、总结讨论

1. 实验总结

- (1) **知识收获**: 本次搜索策略实验给了我极大的触动, 尤其是问题 1-4
- (a) 问题 1-4 使我通过自己的实践认识到不同的搜索策略的差别仅仅在于 Open 表排列方式的不同, 算法的整体实现完全相同;
 - (b) 在问题 5-8 中, 通过设计启发式函数, 并通过动画 GUI 展示, 使我充分认识到不同启发函数在拓展节点和寻找最优路径上的差异。
- (2) **代码心得**: 代码复用、类继承与实现、函数设计
- (a) 得益于 util 工具类, 问题 1-4 代码具有高度的复用性, 值得学习;
 - (b) 整个实验的源代码对我来说, 是一份极好的学习模板。代码通过继承与实现等关系, 可以很好的处理各种不同的问题和启发式函数。

2. 实验启发

- (1) 理解一个搜索策略, 可以从算法的形式化描述和 Open 表排序的不同入手;
- (2) 对于 A*算法来说, 一个良好的满足可纳性与一致性的启发式函数, 可以极大地减少问题拓展节点数和搜索时间。因此在设计启发式函数时, 要在真实代价下界中, 尽可能地利用更多的启发信息, 增大启发式函数值。

附录：A*算法证明

1. 可纳性

描述	A*算法是可采纳的，即若存在从初始节点 S_0 到目标节点 S_g 的路径，则 A*算法必定可以结束在最优路径上
说明	问题 4 目标节点只有 1 个且为有限图，所以不用考虑无限图或多目标情况。因此算法证明简化为 3 部分： 算法必然结束、成功结束和路径最优
定理	在 A*算法结束前的任意时刻，Open 表中总存在节点 n' ，它是从初始节点到目标节点的最佳路径上的一个节点，且满足 $f(n') \leq f^*(S_0)$ （证明略）
证明	<p>必然结束：有限图若找到解则成功结束；否则必然有 open 表变空而结束</p> <p>成功结束：至少存在一条从初始节点到目标节点的路径 $S_0=n_0, n_1, \dots, n_k=S_g$；算法开始时，节点 S_0 在 Open 表中，且路径中任意节点 n_i 离开 Open 表后，其后继 n_{i+1} 必然会加入 Open 表，因此在 Open 表变为空之前，目标节点必然会出现在 Open 表中，则算法一定成功结束</p> <p>路径最优：假设算法未终结在最佳路径上，而是终结在目标节点 t 处，则可得到 $f(t)=g(t)>f^*(S_0)$。但由定理可知，A*算法结束前必有最佳路径上的一个节点 n' 在 Open 表中，且 $f(n') \leq f^*(S_0) < f(t)$，这时 A*算法一定会选择 n' 来拓展，而不可能选择结点 t，从而也不会去测试目标节点 t。得到结论与假设矛盾，因此，A*算法只终止在最佳路径上</p>

2. 最优性（信息性）

描述	在满足 $h(n) \leq h(n^*)$ 的情况下， $h(n)$ 的值越大越好。当 $h(n)$ 越大时，其携带的启发信息就越多，A*算法搜索的拓展节点就越少，搜索效率越高
定理	<p>表述：A1*: $f_1(n)=g_1(n)+h_1(n)$; A2*: $f_2(n)=g_2(n)+h_2(n)$</p> <p>若 $h_1^* < h_2^*$，则在搜索过程中，被 A2*拓展的节点必然会被 A1*拓展</p>
证明	对于深度为 0 节点，若其为目标节点，则 A1*和 A2*均不拓展 n ；否则都要拓展该节点；假设对于 A2*搜索树中深度为 k 的节点 n 结论成立，即 A1*也拓展了这些节点；证明 A2*搜索树中深度为 $k+1$ 的任意节点 n ，也要由 A1*拓展（由反证易得，这里略）

参考文献

- [1]王万森.人工智能原理及其应用[M]. 北京:电子工业出版社, 2018. 115-132.
- [2]张仁平, 周庆忠, 熊伟, et al. A*算法改进算法及其应用[J]. 计算机系统应用, 2009, 18(9):98-100.
- [3]龚道雄, 刘翔. 迷宫搜索算法的比较研究[J]. 计算机应用研究, 2011, 28(12):4433-4436.
- [4]王士同. 建立启发式评价函数 $h(n)$ 的一般方法[J]. 计算机工程与设计, 1987(06):38-44.
- [5]贾森浩. 游戏人工智能中 A*算法的应用研究[D].杭州电子科技大学,2017.