智能计算技术大作业

实验报告

Intelligent computing technology

姓名：李广通

学号：919106840421

实验：基于 A*算法的无人车路径规划

专业：计算机科学与技术专业

学院：计算机科学与工程学院

南京理工大学
NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

目录

1.实验概述

1.1 实验目的

通过实验掌握 A*智能搜素算法及其在无人车路径规划中的应用。提升使用智能搜索算法解决实际问题的能力。

1.2 实验要求

C/C++或 Python

1.3 实验内容

生成一个 NxN 的二维网格，随机指定一些格子为障碍， 并设定左上角有辆无人车（占一个格子）要去右下角，使用 A*算法为该无人车计算起点到终点的不撞到障碍的最优路径。

2.实验源码

使用 python 语言实现，所有实验代码已经做好注释说明，源文件打包在压缩包内，可执行验证

```
import random
from heapq import heappop,heappush

def manhattanDistance(a, b):
    # 曼哈顿距离
    return abs(a[0] - b[0]) + abs(a[1] - b[1])
def astar_1(grid, start, goal):
    # 创建优先队列来存储下一个要访问的节点
    queue = []
    heappush(queue, (0, start))
    # 创建字典来存储结点访问开销
    cost = {start: 0}
    # 创建字典存储节点的前驱
    parent = {start: None}
    while queue:
        # 取出访问开销最低的结点
        current = heappop(queue)[1]
        # 检查是否到达目的地
        if current == goal:
            # 达到目的地就可以整合路径了
            path = []
            while current != start:
                path.append(current)
                current = parent[current]
            path.append(start)
            return path[::-1]
        # 检查当前结点的所有邻居（只考虑了前后左右）
        # v1.0 只考虑前后左右四个方向
        # v2.0 考虑前后左右+左上右上左下右下共计 8 个个方向
        for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
            x, y = current
            neighbor = x + dx, y + dy
```

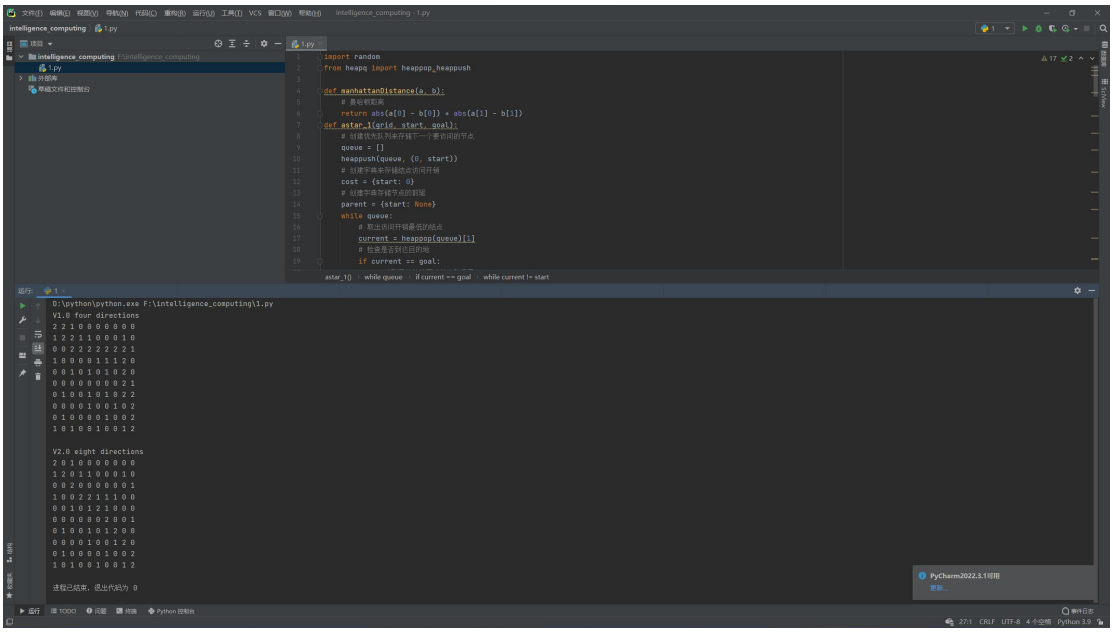
```
                    if 0 <= x + dx < N and 0 <= y + dy < N:
                        if grid[x + dx][y + dy] != 1:
                            if neighbor not in cost or cost[current] + 1 < cost[neighbor]:
                                cost[neighbor] = cost[current] + 1
                                priority = cost[neighbor] + manhattanDistance(goal, neighbor)
                                heappush(queue, (priority, neighbor))
                                parent[neighbor] = current
        return None
    def astar_2(grid, start, goal):
        # 创建优先队列来存储下一个要访问的节点
        queue = []
        heappush(queue, (0, start))
        # 创建字典来存储结点访问开销
        cost = {start: 0}
        # 创建字典存储节点的前驱
        parent = {start: None}
        while queue:
            # 取出访问开销最低的结点
            current = heappop(queue)[1]
            # 检查是否到达目的地
            if current == goal:
                # 达到目的地就可以整合路径了
                path = []
                while current != start:
                    path.append(current)
                    current = parent[current]
                path.append(start)
                return path[::-1]
            # 检查当前结点的所有邻居（只考虑了前后左右）
            # v2.0 考虑前后左右+左上右上左下右下共计 8 个个方向
            for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (-1, -1),  (1, -1),  (-1, 1)]:
                x, y = current
                neighbor = x + dx, y + dy
                if 0 <= x + dx < N and 0 <= y + dy < N:
                    if grid[x + dx][y + dy] != 1:
```

```
                        if neighbor not in cost or cost[current] + 1 <
cost[neighbor]:
                            cost[neighbor] = cost[current] + 1
                            priority = cost[neighbor] + manhattanDistance(goal,
neighbor)
                            heappush(queue, (priority, neighbor))
                            parent[neighbor] = current
        return None

    # 初始化迷宫，0 为无障碍，1 为障碍物
    N = 10
    grid = [[0 for x in range(N)] for y in range(N)]

    # 随机添加障碍
    for i in range(N):
        for j in range(N):
            if random.random() < 0.2:
                grid[i][j] = 1

    # 使用 A*算法对路径进行标注
    path_1 = astar_1(grid, (0,0), (N-1,N-1))
    for cell in path_1:
        x, y = cell
        grid[x][y] = 2

    print("V1.0 four directions")
    # Print the final grid
    for i in range(N):
        for j in range(N):
            print(grid[i][j], end=' ')
        print()

    for i in range(N):
        for j in range(N):
            if grid[i][j] == 2:
                grid[i][j] = 0

    path_2 = astar_2(grid, (0,0), (N-1,N-1))
```

```
for cell in path_2:
    x, y = cell
    grid[x][y] = 2

print("\nV2.0 eight directions")
# Print the final grid
for i in range(N):
    for j in range(N):
        print(grid[i][j], end=' ')
    print()
```

3.实验结果演示及说明

使用 pycharm 运行结果如下所示



```
D:\python\python.exe F:\intelligence_computing\1.py
V1.0 four directions
2 2 1 0 0 0 0 0 0 0
1 2 2 1 1 0 0 0 1 0
0 0 2 2 2 2 2 2 2 1
1 0 0 0 0 1 1 1 2 0
0 0 1 0 1 0 1 0 2 0
0 0 0 0 0 0 0 0 2 1
0 1 0 0 1 0 1 0 2 2
0 0 0 0 1 0 0 1 0 2
0 1 0 0 0 0 1 0 0 2
1 0 1 0 0 1 0 0 1 2

V2.0 eight directions
2 0 1 0 0 0 0 0 0 0
1 2 0 1 1 0 0 0 1 0
0 0 2 0 0 0 0 0 0 1
1 0 0 2 2 1 1 1 0 0
0 0 1 0 1 2 1 0 0 0
0 0 0 0 0 0 2 0 0 1
0 1 0 0 1 0 1 2 0 0
0 0 0 0 1 0 0 1 2 0
0 1 0 0 0 0 1 0 0 2
1 0 1 0 0 1 0 0 1 2

进程已结束，退出代码为 0
```

其中，0 为无障碍，1 为障碍物，2 为路径

V1.0（只考虑前后左右四个方向）

```
D:\python\python.exe F:\intelligence_computing\1.py
V1.0 four directions
2 2 1 0 0 0 0 0 0 0
1 2 2 1 1 0 0 0 1 0
0 0 2 2 2 2 2 2 2 1
1 0 0 0 0 1 1 1 2 0
0 0 1 0 1 0 1 0 2 0
0 0 0 0 0 0 0 0 2 1
0 1 0 0 1 0 1 0 2 2
0 0 0 0 1 0 0 1 0 2
0 1 0 0 0 0 1 0 0 2
1 0 1 0 0 1 0 0 1 2
```

V2.0（考虑前后左右+左上右上左下右下共计 8 个个方向）

```
V2.0 eight directions
2 0 1 0 0 0 0 0 0 0
1 2 0 1 1 0 0 0 1 0
0 0 2 0 0 0 0 0 0 1
1 0 0 2 2 1 1 1 0 0
0 0 1 0 1 2 1 0 0 0
0 0 0 0 0 0 2 0 0 1
0 1 0 0 1 0 1 2 0 0
0 0 0 0 1 0 0 1 2 0
0 1 0 0 0 0 1 0 0 2
1 0 1 0 0 1 0 0 1 2
```