成 绩 _____

北京航空航天大學
BEIHANG UNIVERSITY

模式识别与智能技术
实验报告

院（系）名称	高等理工学院
专 业 名 称	自动化
学 号	18375305
姓 名	吕书礼

2021 年 6 月 3 日

一、实验目的

1、使学生加深对图搜索技术的理解

2、掌握图搜索基本编程方法

3、运用图搜索技术解决一些应用问题

二、实验要求

1、用启发式搜索算法实现路径规划问题。

2、有明确的状态空间表达，规则集以及估计函数。

3、程序运行时，应能清晰直观演示搜索过程。

三、实验内容

机器人路径规划问题：左上角为坐标原点，水平向右为 x 轴方向，竖直向下为 y 轴方向。白色为自由栅格，黑色为障碍栅格，机器人只能在自由栅格中运动，并躲避障碍。每个栅格由唯一的坐标（x,y）表示。机器人一般有八个可移动方向。给出由初始位置（3，3）到目标位置（9，9）的最佳路线。

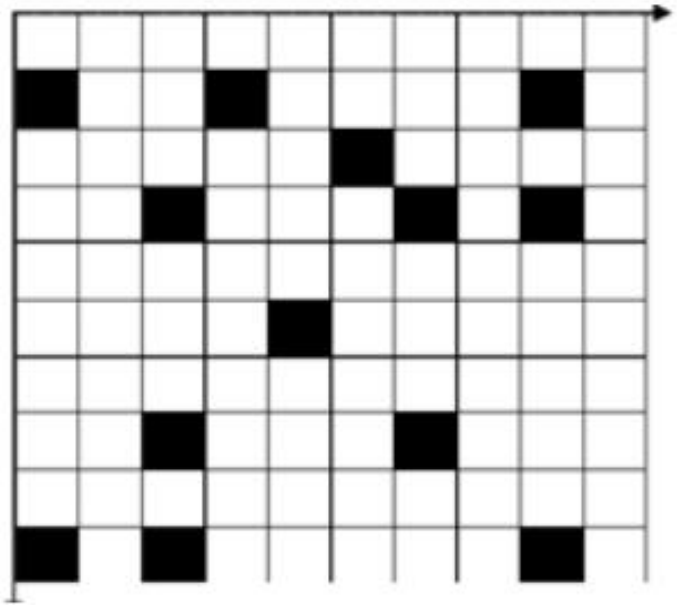

图 1.实验内容图

四、算法原理

A*算法：

g*(n)：从 s 到 n 的最优路组的实际代价。

h*(n)：从 n 到 g 的最优路组的实际代价。

f*(n)=g*(n)+h*(n)：从 s 经过 n 到 g 的最优路组的实际代价。

g(n)、h(n)、f(n)分别是 g*(n)、h*(n)、f*(n)的估计值。

g(n)通常为从 S 到到 n 返段路组的实际代价，则有 $g(n) \geq g^*(n)$。

h(n):是从节点 n 到目标节点 Sg 的最优路组的估计代价. 它的选择依赖于有关问题领域的启发信息，又被叫做启发函数。

要求：$h(n) \leq h^*(n)$。

在图搜索的一般算法中，在搜索的每一步都利用估价函数 f(n)= g(n)+h(n)对 Open 表中的节点进行排序表中的节点进行排序, 找出一个最有希望的节点作为下一次扩展的节点。

1.把起始节点 S 放到 OPEN 表中，计算 f(S)，并把其值与节点 S 联系起来。

2.如果 OPEN 表是个空表，则失败退出，无解。

3.从 OPEN 表中选择一个 f 值最小的节点 i。结果有几个节点合格，当其中有一个为目标节点时，则选择此目标节点，否则就选择其中任一个节点作为节点 i。

4.把节点 i 从 OPEN 表中移出, 把它放入 CLOSED 的扩展节点表中。

5.如果 i 是个目标节点, 则成功退出, 求得一个解。

6.扩展节点 i, 生成其全部后继节点. 对于 i 的每一个后继节点 j:

a)计算 f(j)。

b)如果 j 既不在 OPEN 表中, 也不在在 CLOSED 表中, 则用估价函数 f 把它添入 OPEN 表. 从 j 加一指向父辈节点 i 的指针。

c)如果 j 已在 OPEN 表或 CLOSED 表上, 则比较刚刚对 j 计算过的 f 值和前面计算过的该节点在表中的 f 值. 如果新的 f 值较小, 则：

I. 以此新值替代旧值。

II. 从 j 指向 i, 而不是指向它的父辈节点。

III. 如果节点 j 在 CLOSED 表中, 则把它移回 OPEN 表 7 转向 2,即

GOTO 2。

7.转向 2,即 GOTO 2。

五、实验设计

（1）设计问题的状态表示方法；

定义状态图是为（S,F,G），状态 S 是点的坐标（x,y），初始状态任意给定，F 是转换规则，这里使用八个移动方向表示八种规则，G 是目标状态，在搜索之前给定。

设计状态表示为节点 x,y，父节点与当前节点状态（障碍物，已加入 open 表，未加入 open 表，路径上点）对应颜色为黑色，蓝色，白色，红色。

（2）定义该问题的启发式函数，判断该定义是否满足 A*算法？

由于问题中机器人移动可以斜向移动，为满足 A*算法定义，故不能使用曼哈顿距离作为启发函数，本实验采用欧式距离作为启发函数，因为两点间距离小于实际代价，所以满足 A*算法。

同时本实验对采用曼哈顿距离与欧式距离进行了对比，观察 A 算法与 A*算法区别。

对欧式距离，即定义：

$$h(n)=\sqrt{(x_{current}-x_{final})^2+(y_{current}-y_{final})^2}$$

因为两点之间直线最短，对无障碍情况下的题目要求（可以斜角寻路），欧式距离满足实际最优特殊情况下与最小值相等（即目标节点刚好可以通过斜角寻路得到），其他情况下欧式距离均小于实际代价，所以显然满足 A*算法要求 h(n)≤h*(n)。符合 A*算法标准。

对曼哈顿距离：

$$h(n)=|x_{current}-x_{final}|+|y_{current}-y_{final}|$$

在可以斜角寻路的情况下，无法满足 h(n)≤h*(n)，因此仅为 A 算法。本文使用两种启发函数进行对比分析。

（3）设计数据结构

每个节点由类组成，包含父节点，当前代价，总代价估计值，所在坐标，点

的状态。

（4）设计规则集

1.每次取出的点都预估代价为最小。

2.每格距离设置为 10，斜着走时距离设置为 14。

3.每次遍历 8 个方向。

4.若超出边界则不继续此次循环直接跳过。

5.若搜索点在 open 表中，则比较新代价与原代价，并根据结果进行更新。

6.若搜索点不在 open 表中，将其加入 open 表中，并设置其父节点以及相关参数。

六、实验过程

本实验采用已走过路程作为 g(n)，即定义斜向走一步为 14，正向走一步为 10，采用欧式距离作为 A*算法的启发函数 h(n)，并进行可视化寻找。算法流程具体设计如下：

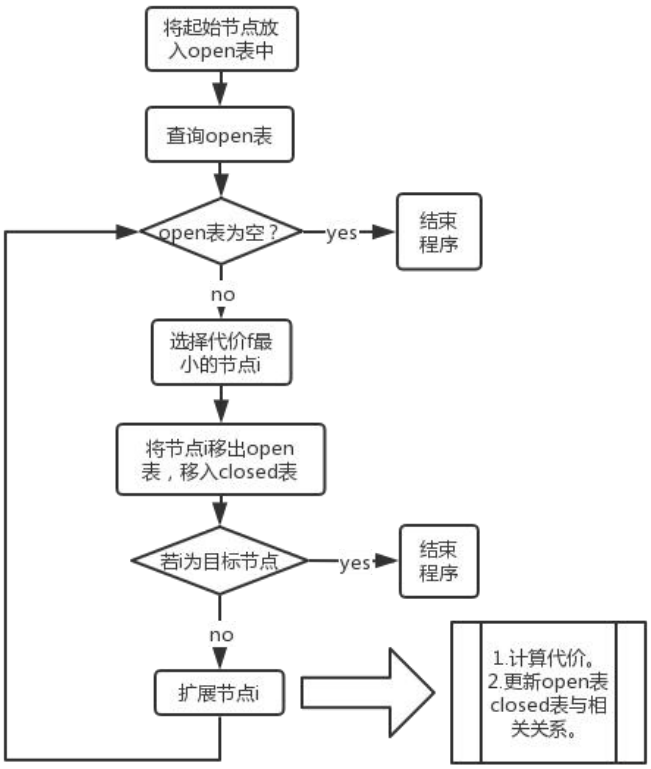

图 2.算法流程图

对于实验的可视化方法，采用 python 的 GUI 库 Tkinter 库进行绘制。将鼠标左键与障碍设置绑定，鼠标右键与开始寻路绑定，鼠标中键与重新开始绑定。实现了手动任意设置障碍，后台可以任意改变寻路起点与中点，以及后台任意改变地图大小的功能。并对不同节点状态进行的不同颜色的可视化，以表示未使用，曾经扩展，寻路路径三种状态。

七、实验结果

1.设置实验参数与实验内容要求相同。

```
star = AstarMap(10, 10, 40, (2, 2), (8, 8), 0.02)
```

参数 1 为行数，参数 2 为列数，参数 3 为方格大小，参数 3 为起始点，参数 4 为终止点，参数 5 为延迟时间。

使用欧式距离作为启发函数可视化结果如下：

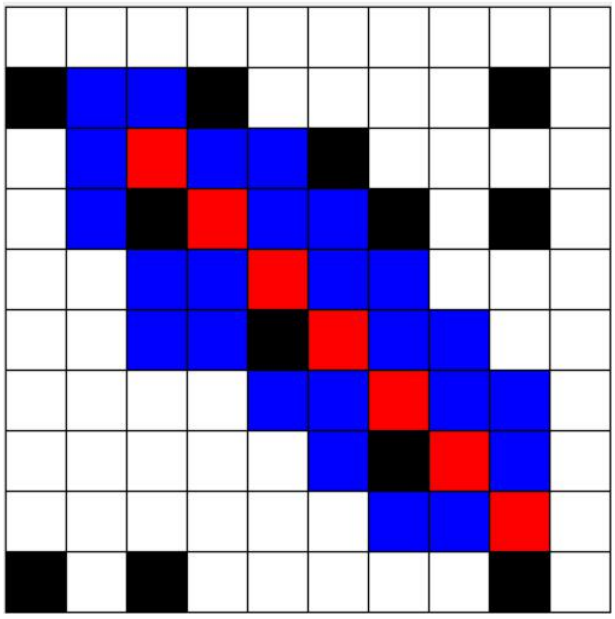

图 3.欧式距离可视化结果

使用曼哈顿距离作为启发函数可视化结果如下所示：

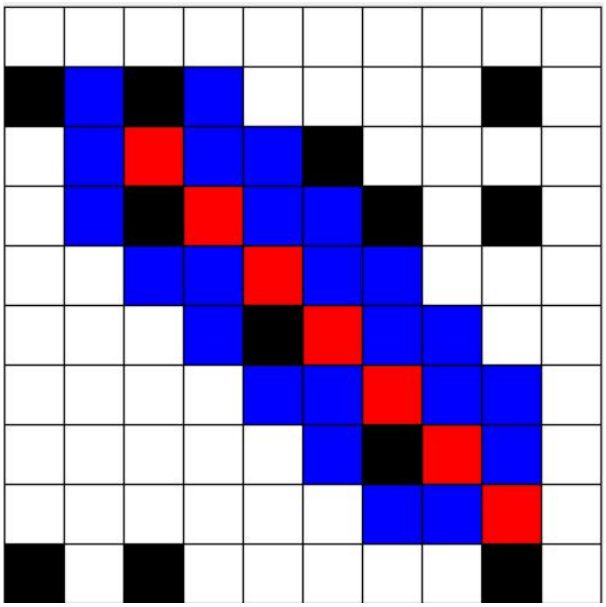

图 4.曼哈顿距离可视化结果

2.任意更改实验参数。

选用本人学号后三位数字 305 作为障碍，参数设定为：

```
star = AstarMap(20, 30, 30, (2, 18), (28, 3), 0.02)
```

搜索结果如下所示：

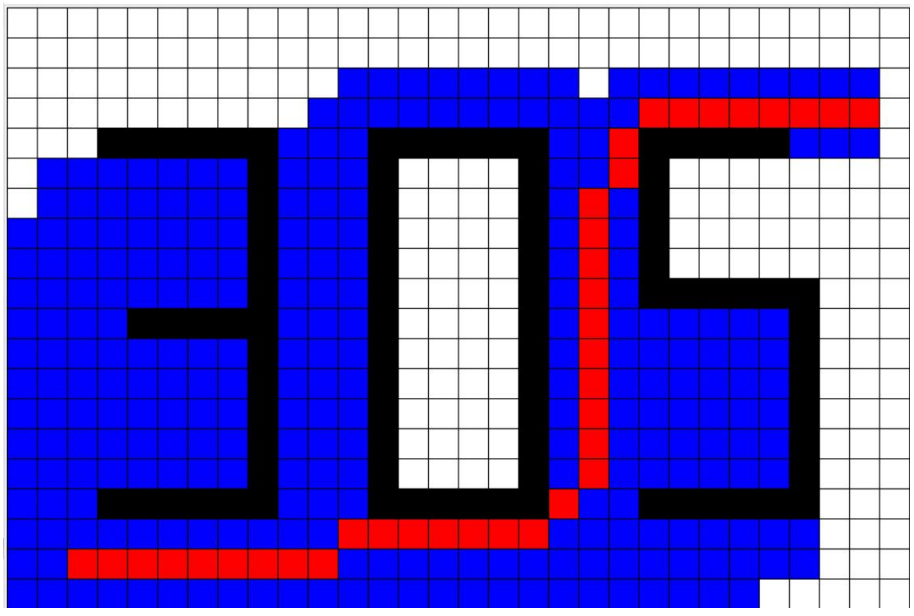

图 5.欧式距离可视化结果

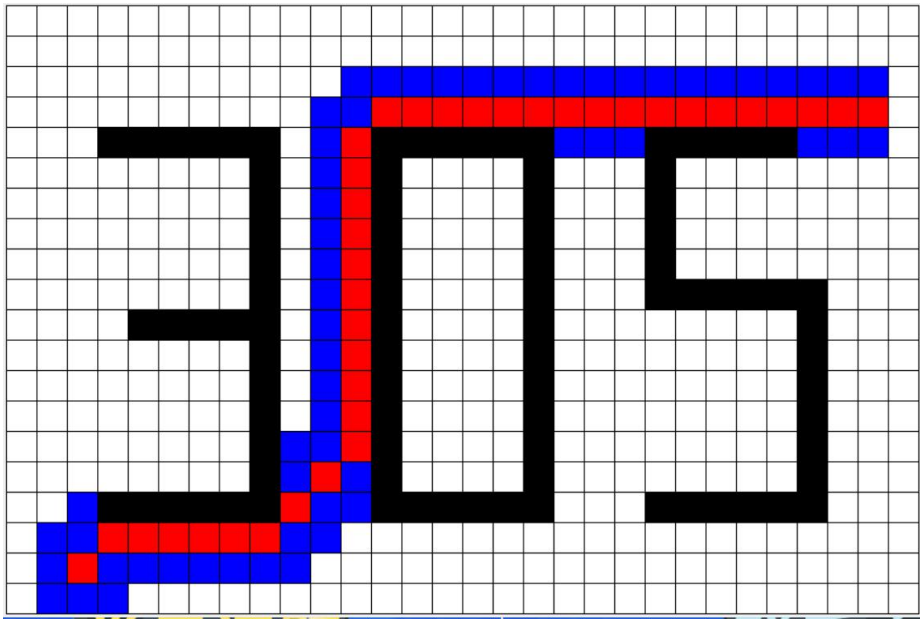

图 6.曼哈顿距离可视化结果

构造复杂搜素障碍：

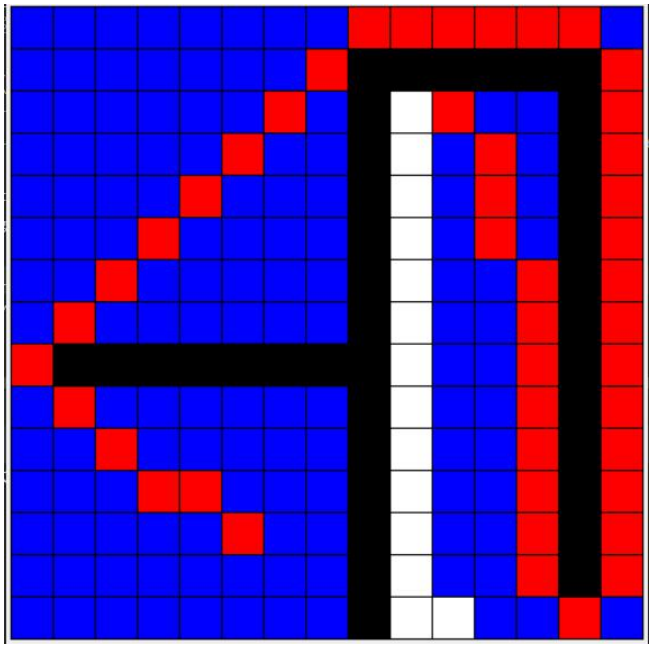

图 7.欧式距离可视化结果

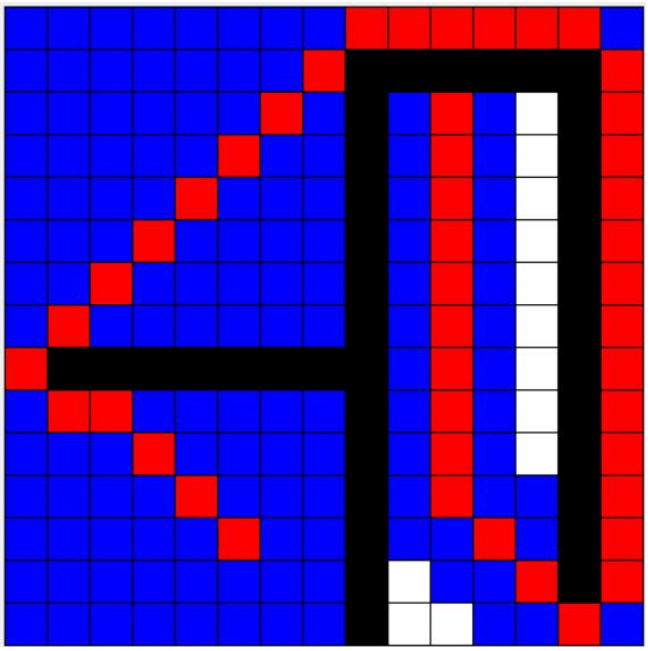

图 8.曼哈顿距离可视化结果

对于无法寻找到结果的寻路过程：

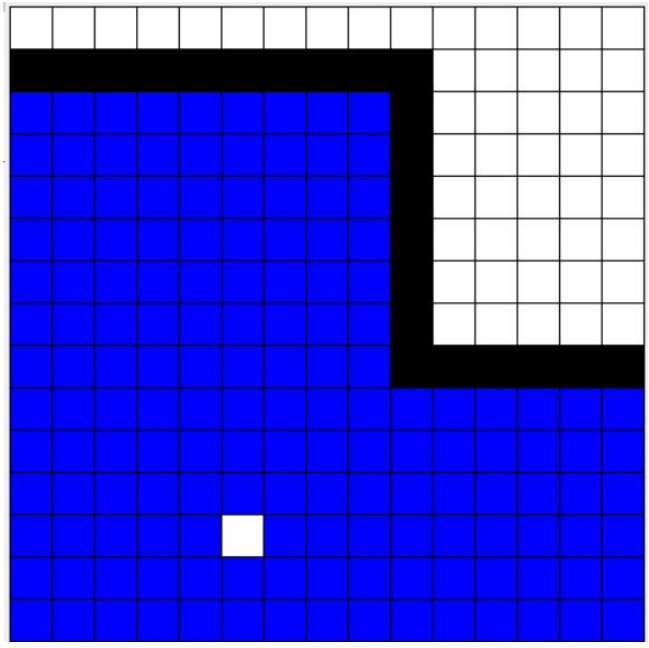

图 9.欧式距离可视化结果

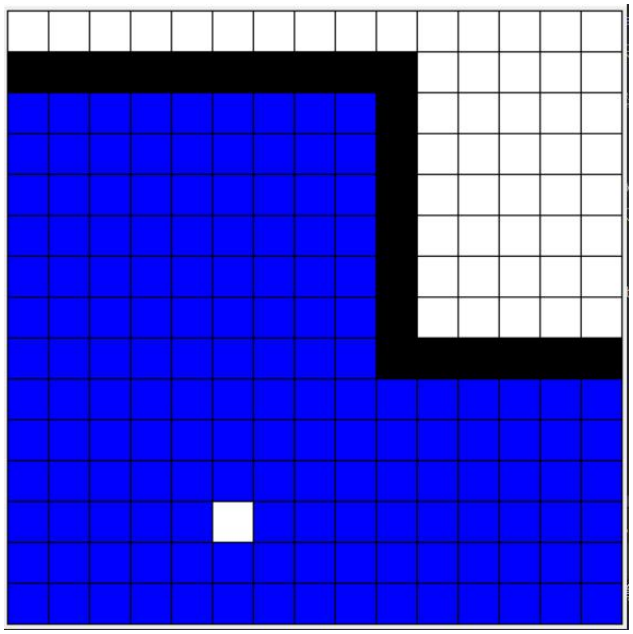

图 10.曼哈顿距离可视化结果

八、实验结论

对比观察曼哈顿距离与欧式距离作为启发函数，即对应 A 算法与 A*算法进行比较，可以发现 A*算法遍历的点数多于 A 算法。可以得出结论：

保证找到最短路径（最优解的）条件，关键在于估价函数 f(n)的选取（或者说 h(n)的选取）。

以 h(n)表达状态 n 到目标状态估计的距离，那么 h(n)的选取大致有如下三种情况：

如果 h(n)< h*(n)，这种情况下，搜索的点数多，搜索范围大，效率低。但能得到最优解。

如果 h(n)=h*(n)，此时的搜索效率是最高的。

如果 h(n)>h*(n)，搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。

本文选取的 h(n)，欧式距离对应 h(n)< h*(n)，曼哈顿距离对应 h(n)>h*(n)。即欧式距离对应搜索点数多，范围大，时间长，而曼哈顿距离则反之。与实验时所观察现象一致。

实验代码上传在 github 中：https://github.com/lyushuli/A-star

附录

```
import enum
import numpy as np
import heapq
import time
import _thread
from tkinter import *

class AstarMap:
    class Point:
        def __init__(self, x, y, f, g, father, state, rectangle):
            self.x = x
            self.y = y
            self.f = f
            self.g = g
            self.father = father
            self.state = state
            self.rectangle = rectangle

        def __lt__(self, other):
            if self.f < other.f:
                return True
            else:
                return False

    def __init__(self, *args):
        self.row = args[0]
        self.col = args[1]
        self.size = args[2]
        self.start = args[3]
        self.end = args[4]
        self.delay = args[5]

        self.root = Tk()
        self.root.title("Astar")
        self.canva = Canvas(self.root, width=self.col * self.size + 3, height=self.row * self.size + 3)
        self.points = self.init_points()
        self.init_mesh()

        self.canva.bind("<Button-1>", self.init_barrier)
        self.canva.bind("<Button-2>", self.clear_map)
```

```
        self.canva.bind("<Button-3>", self.navigation)

        self.canva.pack(side=TOP, expand=YES, fill=BOTH)
        self.root.resizable(0, 0)
        self.root.mainloop()

    def init_points(self):
        points = [[self.Point(x, y, 0, 0, None, PointState.UNUSED.value,
                              self.canva.create_rectangle((x * self.size + 3, y * self.size + 3),
                              ((x + 1) * self.size + 3, (y + 1) * self.size + 3),
                              fill=PointState.UNUSED.value)) \
                   for y in range(self.row)] for x in range(self.col)]
        return points

    def init_mesh(self):
        for i in range(self.row + 1):
            self.canva.create_line((3, i * self.size + 3), (self.col * self.size + 3, i * self.size + 3))
        for i in range(self.col + 1):
            self.canva.create_line((i * self.size + 3, 3), (i * self.size + 3, self.row * self.size + 3))

    def init_barrier(self, event):
        x = int((event.x + 3) / self.size)
        y = int((event.y + 3) / self.size)
        if x < self.col and y < self.row:
            if self.points[x][y].state == PointState.BARRIER.value:
                self.points[x][y].state = PointState.UNUSED.value
                self.canva.itemconfig(self.points[x][y].rectangle, fill=self.points[x][y].state)
            else:
                self.points[x][y].state = PointState.BARRIER.value
                self.canva.itemconfig(self.points[x][y].rectangle, fill=self.points[x][y].state)

    def clear_map(self, event):
        for i in range(self.col):
            for j in range(self.row):
                if (self.points[i][j].state != PointState.BARRIER.value):
                    self.points[i][j].state = PointState.UNUSED.value
                    self.canva.itemconfig(self.points[i][j].rectangle, fill=self.points[i][j].state)

    def navigation(self, event):
        _thread.start_new_thread(self.find_path, (self.start, self.end))

    def find_path(self, start, end):
        x1 = start[0]
        y1 = start[1]
```

```
            x2 = end[0]
            y2 = end[1]

            openlist = []
            closeset = set()
            openset = set()
            heapq.heappush(openlist, self.points[x1][y1])
            openset.add((x1, y1))
            while 1:
                p_min = heapq.heappop(openlist)
                openset.remove((p_min.x, p_min.y))
                closeset.add((p_min.x, p_min.y))

                for i in range(-1, 2):
                    for j in range(-1, 2):
                        if i == 0 and j == 0:
                            continue
                        x_new = p_min.x + i
                        y_new = p_min.y + j
                        if x_new >= self.col or x_new < 0 or y_new >= self.row or y_new < 0:
                            continue
                        p_new = self.points[x_new][y_new]
                        oblique = i != 0 and j != 0
                        if (x_new, y_new) not in closeset and self.points[x_new][y_new].state !=
PointState.BARRIER.value:
                            if (x_new, y_new) in openset:
                                if ((14 if oblique else 10) + p_min.g) < p_new.g:
                                    p_new.g = p_min.g + (14 if oblique else 10)
                                    # p_new.f = p_new.g + 10 * (abs(x2 - x_new) + abs(y2 - y_new)) #  曼哈
顿距离
                                    p_new.f = p_new.g + 10 * round(np.linalg.norm(np.array([x2, y2]) -
np.array([x_new, y_new])))  #  欧式距离
                                    p_new.father = p_min
                            else:
                                p_new.g = p_min.g + (14 if oblique else 10)
                                # p_new.f = p_new.g + 10 * (abs(x2 - x_new) + abs(y2 - y_new)) #  曼哈顿距
离
                                p_new.f = p_new.g + 10 * round(np.linalg.norm(np.array([x2, y2]) -
np.array([x_new, y_new])))  #  欧式距离
                                p_new.father = p_min
                                p_new.state = PointState.TRAVERSED.value
                                self.canva.itemconfig(p_new.rectangle, fill=PointState.TRAVERSED.value)
                                heapq.heappush(openlist, p_new)
                                openset.add((x_new, y_new))
```

```
                if (x2, y2) in openset:
                    p_next = self.points[x2][y2]
                    p_next.state = PointState.PATH.value
                    self.canva.itemconfig(p_next.rectangle, fill=PointState.PATH.value)
                    while p_next.father:
                        p_next = p_next.father
                        self.points[p_next.x][p_next.y].state = PointState.PATH.value
                        self.canva.itemconfig(self.points[p_next.x][p_next.y].rectangle,
fill=PointState.PATH.value)
                    break
                if len(openlist) == 0:
                    print('No path!')
                    break
                time.sleep(self.delay)

class PointState(enum.Enum):
    BARRIER = 'black'
    UNUSED = 'white'
    TRAVERSED = 'blue'
    PATH = 'red'

if __name__ == "__main__":
    star = AstarMap(15, 15, 30, (5, 12), (10, 2), 0.02)
```