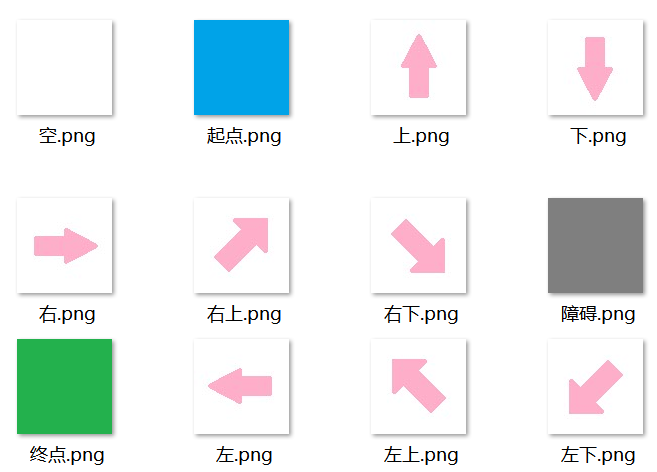
A星寻路实验报告

学院：人工智能 专业：智能科学与技术 学号：1813543 姓名：邓陈龙

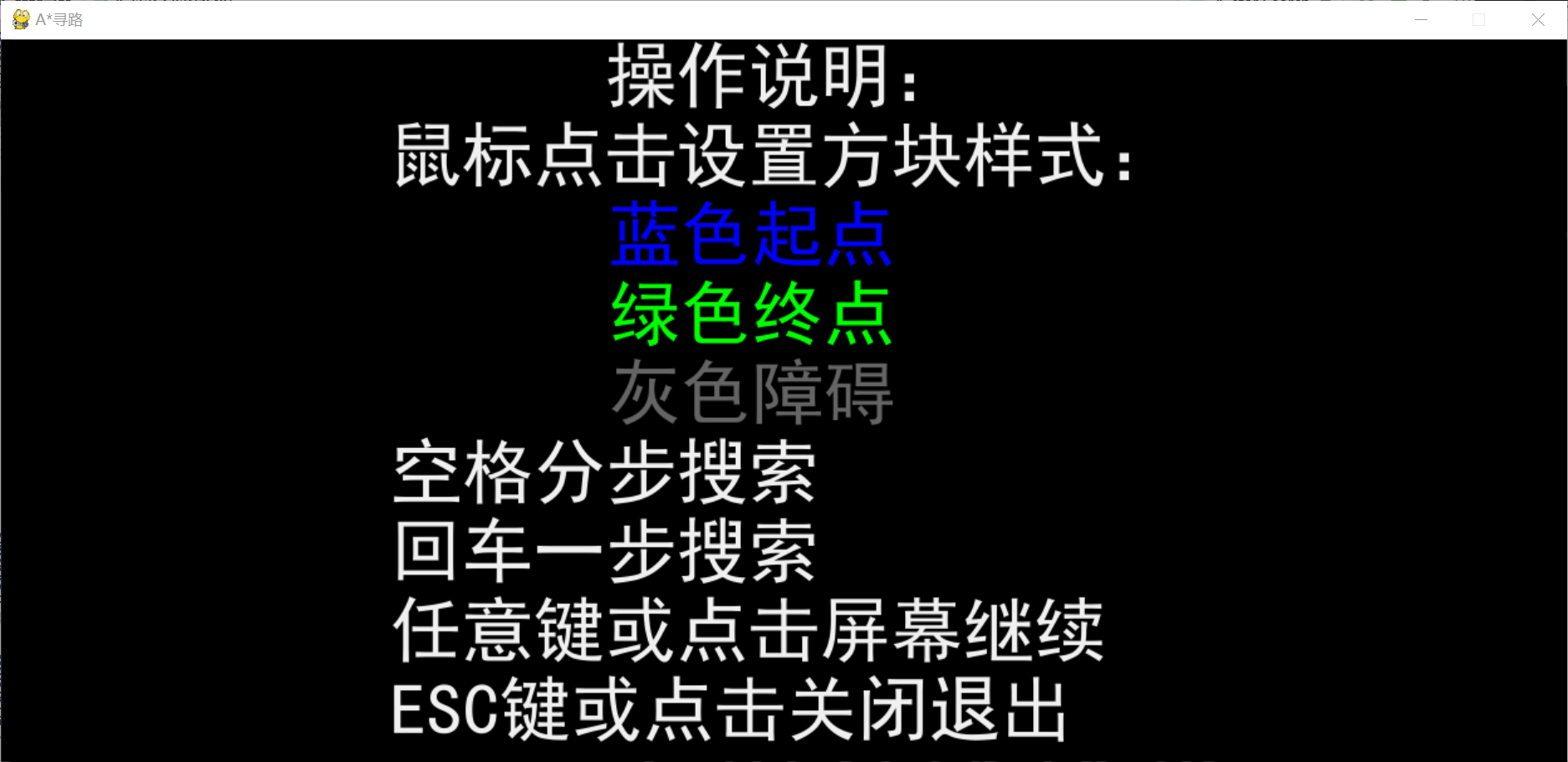
1. 实验目的
2. 掌握A星算法的代码实现；
3. 熟悉A星算法的逻辑。
4. 实验要求
5. 提交源码、可执行文件和实验报告。
6. 前期准备
7. 自学pygame。通过在网上查找资料自学pygame游戏编程；
8. 安装anaconda和pycharm，安装需要的包。
9. 编译环境

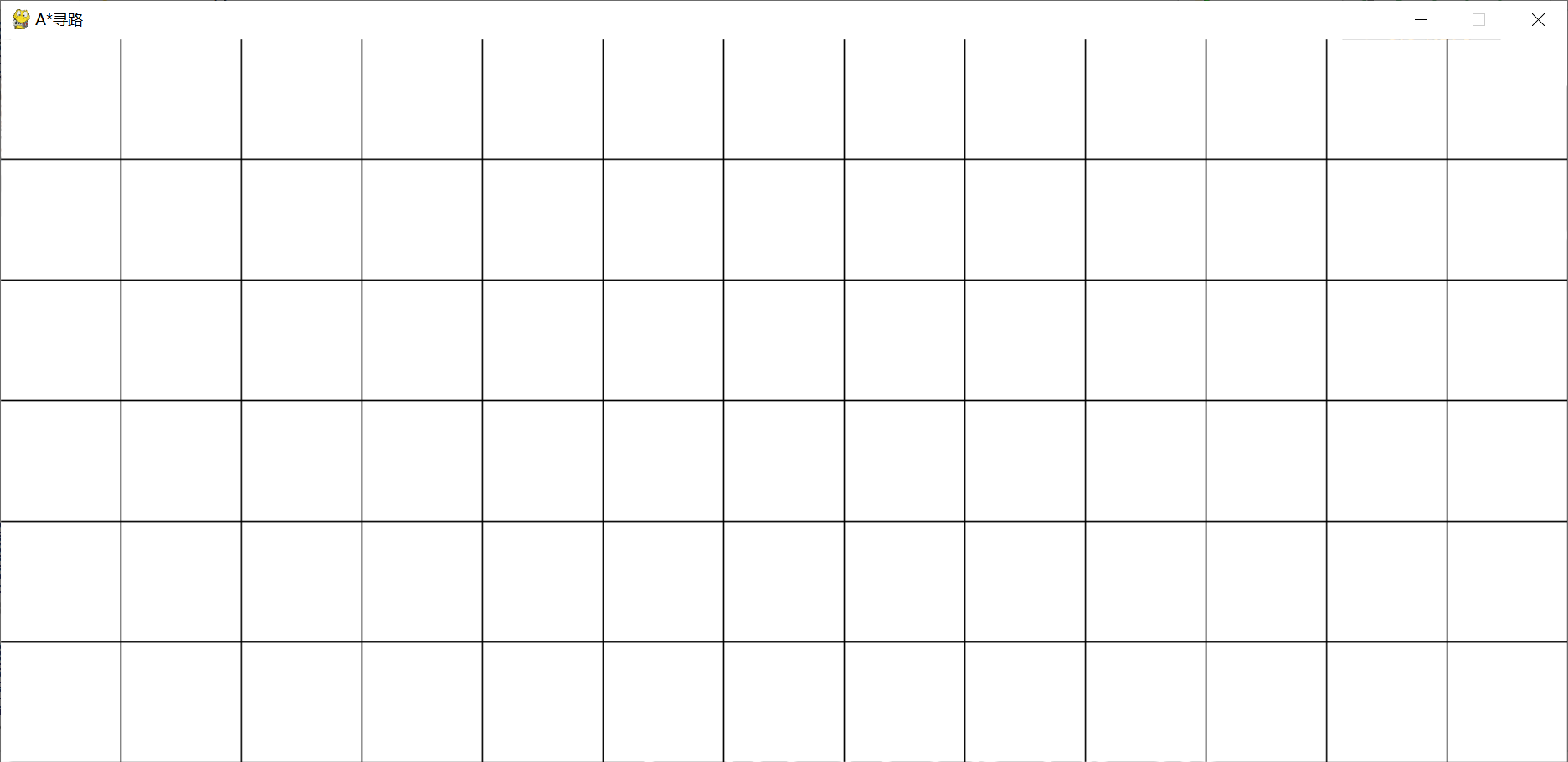
Windows10，python3.7，pycharm，anaconda。

1. 实验步骤
2. 用Windows10自带画图软件绘制需要用的位图：

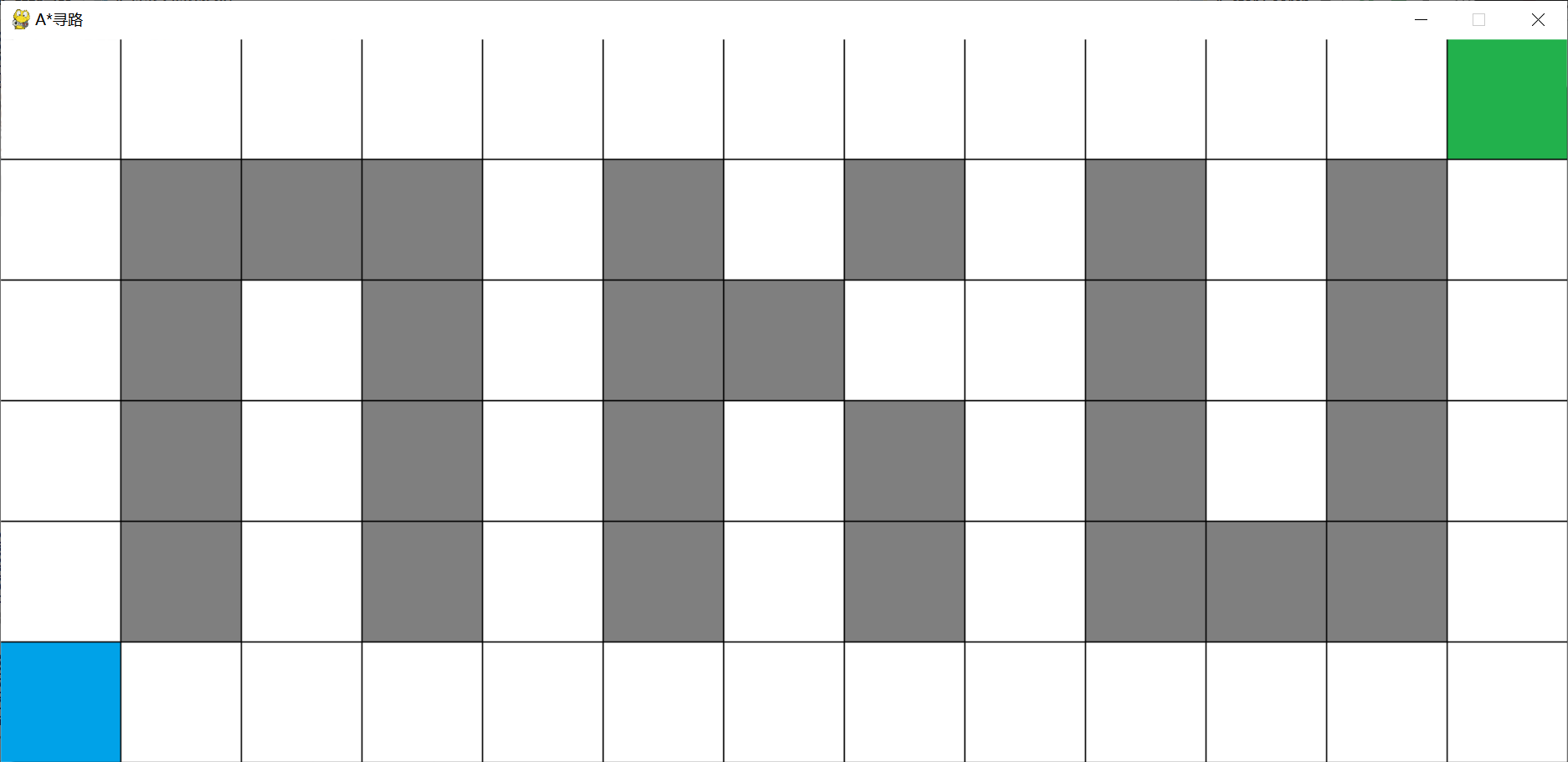


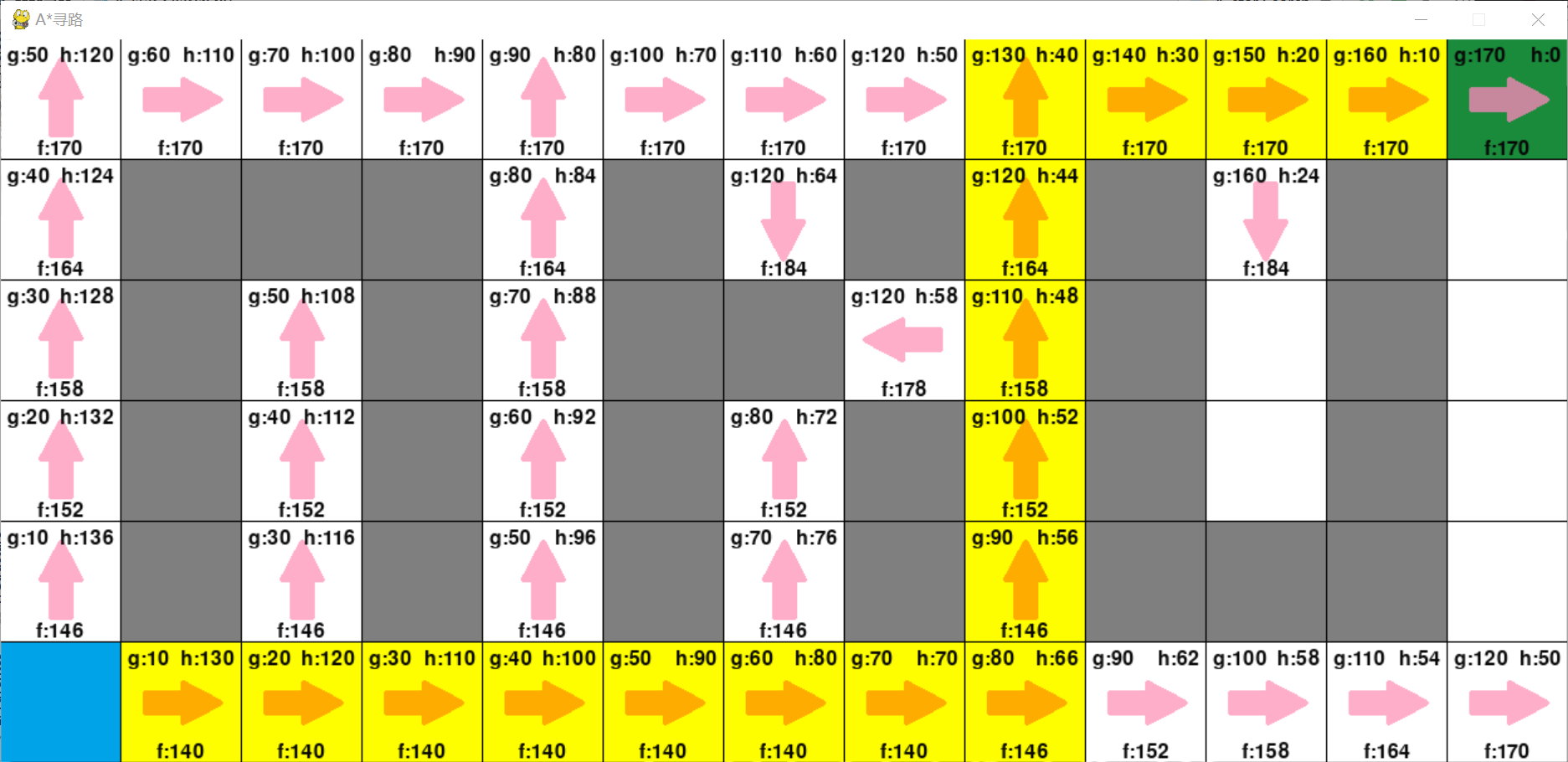
1. 基于pygame实现基础的界面：





1. 在可视化的基础上实现A\*算法





1. 代码功能
   1. 进入时的操作说明，符合老师要求；
   2. 可以手动设置起点、终点和障碍；
   3. 界面美观，全程可视化，体验极佳；
   4. 不会从墙角经过，符合老师要求；
   5. 可以一步完成也可以分步完成搜索，便于理解A\*算法；
   6. 界面尺寸、方块大小和方块个数的调节都是十分方便的；
   7. 全程面向对象，可扩展性强；
2. 问题与解决
   1. 因为每次都要从OPEN表中取估计值最小的节点，常规方法需要进行多次排序，耗时耗力。

解：我选择用堆来保存OPEN表，因为堆的性质使得添加节点后仍然可以保持有序，代码少了许多。

* 1. 使用堆的时候仍然有一点问题，比较时出现了错误，原因是自定义节点类不支持比较运算符。

解：重载<运算符。我发现python和C++重载运算符的方法不太一样，python使用\_\_lt\_\_(self,data)重载<运算符，其余同理。

* 1. 使用OPEN表和CLOSED表不好面向对象。

解：将表用布尔型成员变量代替，比如is\_closed或is\_open之类的。

1. 代码实现

import os  
from pygame import \*  
import pygame as pg  
import heapq  
*# from queue import Queue*from pygame.compat import geterror  
from typing import List, Tuple  
  
*# 检查pygame*if not pg.font:  
 print(**"Warning, fonts disabled"**)  
  
*# 定义全局变量*main\_dir = os.path.split(os.path.abspath(\_\_file\_\_))[0]  
data\_dir = os.path.join(main\_dir, **"data"**)  
have\_source = False  
have\_dest = False  
is\_searching = False  
start\_block: **'Block'**goal\_block: **'Block'**HEIGHT = 6  
WIDTH = 13  
SIZE = 96  
  
  
def load\_image(name, colorkey=None):  
 fullname = os.path.join(data\_dir, name)  
 try:  
 my\_image = pg.image.load(fullname)  
 *# image = pg.transform.smoothscale(image, (SIZE - 1, SIZE - 1))* except pg.error:  
 print(**"Cannot load image:"**, fullname)  
 raise SystemExit(str(geterror()))  
 my\_image = my\_image.convert()  
 if colorkey is not None:  
 if colorkey == -1:  
 colorkey = my\_image.get\_at((0, 0))  
 my\_image.set\_colorkey(colorkey, pg.RLEACCEL)  
 return my\_image  
  
  
*# left\_button = False  
# right\_button = False  
  
  
# 方块类  
# noinspection DuplicatedCode*class Block(pg.sprite.Sprite):  
 def \_\_init\_\_(self, n: int):  
 pg.sprite.Sprite.\_\_init\_\_(self) *# call Sprite initializer* self.location = (n % WIDTH, n // WIDTH) *# 位置* self.image = load\_image(**"空.png"**)  
 self.rect = self.image.get\_rect(topleft=(self.location[0] \* SIZE, self.location[1] \* SIZE))  
 self.type = 0 *# 0：空；1：障碍；2：起点；3：终点* self.direction\_came\_from = None *# 0：空；1：上；2：下；3：左；4：右；13：左上；14：右上；23：左下；24：右上* self.heuristic = None  
 self.cost\_so\_far = None  
 self.priority = None  
 self.came\_from\_block = None  
 self.neighbors = []  
  
 def \_\_lt\_\_(self, rhs: **"Block"**):  
 if self.location[0] < rhs.location[0]:  
 return True  
 elif self.location[0] > rhs.location[0]:  
 return False  
 else:  
 if self.location[1] < rhs.location[1]:  
 return True  
 elif self.location[1] >= rhs.location[1]:  
 return False  
  
 def update(self):  
 global have\_source  
 global have\_dest  
 if self.type == 0:  
 self.image.blit(load\_image(**"空.png"**), (0, 0))  
 elif self.type == 1:  
 self.image.blit(load\_image(**"障碍.png"**), (0, 0))  
 elif self.type == 2:  
 self.image.blit(load\_image(**"起点.png"**), (0, 0))  
 elif self.type == 3:  
 self.image.blit(load\_image(**"终点.png"**), (0, 0))  
 if self.direction\_came\_from is not None:  
 if self.direction\_came\_from == 1:  
 self.image.blit(load\_image(**"下.png"**, -1), (0, 0))  
 elif self.direction\_came\_from == 2:  
 self.image.blit(load\_image(**"上.png"**, -1), (0, 0))  
 elif self.direction\_came\_from == 3:  
 self.image.blit(load\_image(**"右.png"**, -1), (0, 0))  
 elif self.direction\_came\_from == 4:  
 self.image.blit(load\_image(**"左.png"**, -1), (0, 0))  
 elif self.direction\_came\_from == 13:  
 self.image.blit(load\_image(**"右下.png"**, -1), (0, 0))  
 elif self.direction\_came\_from == 14:  
 self.image.blit(load\_image(**"左下.png"**, -1), (0, 0))  
 elif self.direction\_came\_from == 23:  
 self.image.blit(load\_image(**"右上.png"**, -1), (0, 0))  
 elif self.direction\_came\_from == 24:  
 self.image.blit(load\_image(**"左上.png"**, -1), (0, 0))  
 if self.priority is not None:  
 my\_font = pg.font.Font(None, 24)  
 text = my\_font.render(**"f:"** + str(self.priority), True, (10, 10, 10))  
 text\_pos = text.get\_rect(midbottom=(self.rect.centerx - self.rect.left, self.rect.bottom - self.rect.top))  
 self.image.blit(text, text\_pos)  
 if self.cost\_so\_far is not None and self != start\_block:  
 my\_font = pg.font.Font(None, 24)  
 text = my\_font.render(**"g:"** + str(self.cost\_so\_far), True, (10, 10, 10))  
 text\_pos = text.get\_rect(topleft=(5, 5))  
 self.image.blit(text, text\_pos)  
 if self.heuristic is not None:  
 my\_font = pg.font.Font(None, 24)  
 text = my\_font.render(**"h:"** + str(self.heuristic), True, (10, 10, 10))  
 text\_pos = text.get\_rect(topright=(  
 self.rect.topright[0] - self.rect.topleft[0] - 5, self.rect.topright[1] - self.rect.topleft[1] + 5))  
 self.image.blit(text, text\_pos)  
  
 *# 改变方块类型* def change\_type(self):  
 global have\_source  
 global have\_dest  
 global start\_block  
 global goal\_block  
 if is\_searching:  
 if self.priority is None:  
 if self.type == 0 or self.type == 1:  
 self.type = (self.type + 1) % 2  
 else:  
 if not (have\_source or have\_dest):  
 self.type = (self.type + 1) % 4  
 if self.type == 2:  
 have\_source = True  
 start\_block = self  
 elif have\_source and have\_dest:  
 if self.type == 0 or self.type == 1:  
 self.type = (self.type + 1) % 2  
 elif self.type == 2:  
 self.type = 0  
 have\_source = False  
 start\_block = None  
 elif self.type == 3:  
 self.type = 0  
 have\_dest = False  
 goal\_block = None  
 elif have\_source:  
 self.type = (self.type + 1) % 4  
 if self.type == 2:  
 self.type = (self.type + 1) % 4  
 have\_dest = True  
 goal\_block = self  
 elif self.type == 3:  
 have\_source = False  
 have\_dest = True  
 start\_block = None  
 goal\_block = self  
 else:  
 self.type = (self.type + 1) % 4  
 if self.type == 3:  
 self.type = (self.type + 1) % 4  
 elif self.type == 2:  
 have\_source = True  
 start\_block = self  
 elif self.type == 0:  
 have\_dest = False  
 goal\_block = None  
 if self.type == 0:  
 self.image.blit(load\_image(**"空.png"**), (0, 0))  
 elif self.type == 1:  
 self.image.blit(load\_image(**"障碍.png"**), (0, 0))  
 elif self.type == 2:  
 self.image.blit(load\_image(**"起点.png"**), (0, 0))  
 elif self.type == 3:  
 self.image.blit(load\_image(**"终点.png"**), (0, 0))  
  
 *# 更新父节点* def came\_from(self, a: **'Block'**):  
 self.came\_from\_block = a  
 (x1, y1) = self.location  
 (x2, y2) = a.location  
 x = x2 - x1  
 y = y2 - y1  
 if x == 1 and y == 1:  
 self.direction\_came\_from = 24  
 elif x == 1 and y == 0:  
 self.direction\_came\_from = 4  
 elif x == 1 and y == -1:  
 self.direction\_came\_from = 14  
 elif x == 0 and y == 1:  
 self.direction\_came\_from = 2  
 elif x == 0 and y == -1:  
 self.direction\_came\_from = 1  
 elif x == -1 and y == 1:  
 self.direction\_came\_from = 23  
 elif x == -1 and y == 0:  
 self.direction\_came\_from = 3  
 elif x == -1 and y == -1:  
 self.direction\_came\_from = 13  
  
 def can\_walk\_neighbor(self, a: **"Block"**) -> bool:  
 if a.type == 1:  
 return False  
 *# 不能从墙角过* (x1, y1) = self.location  
 (x2, y2) = a.location  
 x = x2 - x1  
 y = y2 - y1  
 if abs(x) == 1 and abs(y) == 1:  
 if all\_blocks[self.location[0] + self.location[1] \* WIDTH + x].type == 1:  
 return False  
 if all\_blocks[self.location[0] + self.location[1] \* WIDTH + y \* WIDTH].type == 1:  
 return False  
 return True  
  
  
*# 计算估计开销*def heuristic(a: Block, b: Block) -> int:  
 (x1, y1) = a.location  
 (x2, y2) = b.location  
 x = abs(x1 - x2)  
 y = abs(y1 - y2)  
 mini = min(x, y)  
 maxi = max(x, y)  
 return mini \* 14 + (maxi - mini) \* 10  
  
  
*# 计算开销*def cost(a: Block, b: Block) -> int:  
 (x1, y1) = a.location  
 (x2, y2) = b.location  
 x = abs(x1 - x2)  
 y = abs(y1 - y2)  
 if x == 1 and y == 1:  
 return 14  
 else:  
 return 10  
  
  
class PriorityQueue:  
 def \_\_init\_\_(self):  
 self.elements: List[Tuple[int, Block]] = []  
  
 def empty(self) -> bool:  
 return len(self.elements) == 0  
  
 def put(self, item: Block, priority: int):  
 heapq.heappush(self.elements, (priority, item))  
  
 def get(self) -> Block:  
 return heapq.heappop(self.elements)[1]  
  
  
open\_list = PriorityQueue()  
all\_blocks: List[Block] = []  
current: Block  
  
  
def a\_star\_search\_step():  
 global open\_list  
 global is\_searching  
 global current  
  
 if not open\_list.empty():  
 current = open\_list.get()  
  
 if current == goal\_block:  
 is\_searching = False  
 return  
  
 for next\_block in [a for a in current.neighbors if current.can\_walk\_neighbor(a)]:  
 new\_cost = current.cost\_so\_far + cost(current, next\_block)  
 if next\_block.cost\_so\_far is None or new\_cost < next\_block.cost\_so\_far:  
 next\_block.cost\_so\_far = new\_cost  
 next\_block.heuristic = heuristic(next\_block, goal\_block)  
 next\_block.priority = next\_block.cost\_so\_far + next\_block.heuristic  
 next\_block.came\_from(current)  
 open\_list.put(next\_block, next\_block.priority)  
  
  
def main():  
 *# 初始化界面* global is\_searching  
 global start\_block  
 global goal\_block  
 global open\_list  
 global all\_blocks  
 global current  
 pg.init()  
 screen = pg.display.set\_mode((SIZE \* WIDTH - 1, SIZE \* HEIGHT - 1))  
 pg.display.set\_caption(**"A\*寻路"**)  
 *# 创建背景surface* background = pg.Surface(screen.get\_size())  
 background = background.convert()  
 background.fill((0, 0, 0))  
 screen.blit(background, (0, 0))  
 *# 写规则* rect\_screen = screen.get\_rect()  
 my\_font = pg.font.Font(**"C:/Windows/Fonts/simhei.ttf"**, rect\_screen.bottom // 10)  
 text = my\_font.render(**"操作说明："**, True, (240, 240, 240))  
 text\_pos = text.get\_rect(centerx=rect\_screen.centerx)  
 screen.blit(text, text\_pos)  
 text = my\_font.render(**"鼠标点击设置方块样式："**, True, (240, 240, 240))  
 text\_pos = text.get\_rect(top=rect\_screen.bottom // 9 \* 1, centerx=rect\_screen.centerx)  
 screen.blit(text, text\_pos)  
 text = my\_font.render(**" 蓝色起点 "**, True, (0, 0, 255))  
 text\_pos = text.get\_rect(top=rect\_screen.bottom // 9 \* 2, left=text\_pos.left)  
 screen.blit(text, text\_pos)  
 text = my\_font.render(**" 绿色终点 "**, True, (0, 255, 0))  
 text\_pos = text.get\_rect(top=rect\_screen.bottom // 9 \* 3, left=text\_pos.left)  
 screen.blit(text, text\_pos)  
 text = my\_font.render(**" 灰色障碍 "**, True, (100, 100, 100))  
 text\_pos = text.get\_rect(top=rect\_screen.bottom // 9 \* 4, left=text\_pos.left)  
 screen.blit(text, text\_pos)  
 text = my\_font.render(**"空格分步搜索"**, True, (240, 240, 240))  
 text\_pos = text.get\_rect(top=rect\_screen.bottom // 9 \* 5, left=text\_pos.left)  
 screen.blit(text, text\_pos)  
 text = my\_font.render(**"回车一步搜索"**, True, (240, 240, 240))  
 text\_pos = text.get\_rect(top=rect\_screen.bottom // 9 \* 6, left=text\_pos.left)  
 screen.blit(text, text\_pos)  
 text = my\_font.render(**"任意键或点击屏幕继续"**, True, (240, 240, 240))  
 text\_pos = text.get\_rect(top=rect\_screen.bottom // 9 \* 7, left=text\_pos.left)  
 screen.blit(text, text\_pos)  
 text = my\_font.render(**"ESC键或点击关闭退出"**, True, (240, 240, 240))  
 text\_pos = text.get\_rect(top=rect\_screen.bottom // 9 \* 8, left=text\_pos.left)  
 screen.blit(text, text\_pos)  
 pg.display.flip()  
 clock = pg.time.Clock()  
 loop = True  
 while loop:  
 clock.tick(60)  
 for my\_event in pg.event.get():  
 if my\_event.type == pg.QUIT:  
 pg.quit()  
 return  
 elif my\_event.type == pg.KEYDOWN and my\_event.key == pg.K\_ESCAPE:  
 pg.quit()  
 return  
 elif my\_event.type == pg.KEYDOWN:  
 loop = False  
 elif my\_event.type == pg.MOUSEBUTTONUP:  
 loop = False  
 *# 准备游戏环境* all\_blocks = []  
 for i in range(WIDTH \* HEIGHT):  
 all\_blocks.append(Block(i))  
 *# 保存相邻方块* for i in range(WIDTH \* HEIGHT):  
 if i % WIDTH > 0:  
 all\_blocks[i].neighbors.append(all\_blocks[i - 1])  
 if i % WIDTH < WIDTH - 1:  
 all\_blocks[i].neighbors.append(all\_blocks[i + 1])  
 if i >= WIDTH:  
 all\_blocks[i].neighbors.append(all\_blocks[i - WIDTH])  
 if i <= WIDTH \* (HEIGHT - 1) - 1:  
 all\_blocks[i].neighbors.append(all\_blocks[i + WIDTH])  
 if i >= WIDTH and i % WIDTH > 0:  
 all\_blocks[i].neighbors.append(all\_blocks[i - WIDTH - 1])  
 if i >= WIDTH and i % WIDTH < WIDTH - 1:  
 all\_blocks[i].neighbors.append(all\_blocks[i - WIDTH + 1])  
 if i <= WIDTH \* (HEIGHT - 1) - 1 and i % WIDTH > 0:  
 all\_blocks[i].neighbors.append(all\_blocks[i + WIDTH - 1])  
 if i <= WIDTH \* (HEIGHT - 1) - 1 and i % WIDTH < WIDTH - 1:  
 all\_blocks[i].neighbors.append(all\_blocks[i + WIDTH + 1])  
 all\_sprites = pg.sprite.RenderPlain(all\_blocks)  
 *# 主循环* entire = False *# 是否分步* loop = True  
 while loop:  
 clock.tick(60)  
  
 *# 处理输入事件* for my\_event in pg.event.get():  
 if my\_event.type == pg.QUIT:  
 loop = False  
 elif my\_event.type == pg.KEYDOWN and my\_event.key == pg.K\_ESCAPE:  
 loop = False  
 elif my\_event.type == pg.MOUSEBUTTONUP:  
 pos = pg.mouse.get\_pos()  
 all\_blocks[pos[0] // SIZE + pos[1] // SIZE \* WIDTH].change\_type()  
 elif my\_event.type == pg.KEYDOWN and my\_event.key == pg.K\_SPACE:  
 if start\_block is not None and goal\_block is not None:  
 is\_searching = True  
 loop = False  
 open\_list.put(start\_block, 0)  
 start\_block.came\_from = None  
 start\_block.cost\_so\_far = 0  
 a\_star\_search\_step()  
 elif my\_event.type == pg.KEYDOWN and my\_event.key == pg.K\_RETURN:  
 if start\_block is not None and goal\_block is not None:  
 is\_searching = True  
 loop = False  
 open\_list.put(start\_block, 0)  
 start\_block.came\_from = None  
 start\_block.cost\_so\_far = 0  
 entire = True  
  
 all\_sprites.update()  
  
 *# 绘制界面* screen.blit(background, (0, 0))  
 all\_sprites.draw(screen)  
 pg.display.update()  
  
 while is\_searching:  
 clock.tick(60)  
 if entire:  
 a\_star\_search\_step()  
 else:  
 *# 处理输入事件* for my\_event in pg.event.get():  
 if my\_event.type == pg.QUIT:  
 is\_searching = False  
 elif my\_event.type == pg.KEYDOWN and my\_event.key == pg.K\_ESCAPE:  
 is\_searching = False  
 elif my\_event.type == pg.MOUSEBUTTONUP:  
 pos = pg.mouse.get\_pos()  
 all\_blocks[pos[0] // SIZE + pos[1] // SIZE \* WIDTH].change\_type()  
 elif my\_event.type == pg.KEYDOWN and my\_event.key == pg.K\_SPACE:  
 a\_star\_search\_step()  
 elif my\_event.type == pg.KEYDOWN and my\_event.key == pg.K\_RETURN:  
 entire = True  
  
 all\_sprites.update()  
  
 *# 绘制界面* screen.blit(background, (0, 0))  
 current.image.fill((200, 200, 200), special\_flags=BLEND\_MULT)  
 all\_sprites.draw(screen)  
 pg.display.update()  
  
 temp: Block = goal\_block.came\_from\_block  
 *# 绘制结果* while temp != start\_block:  
 temp.image.fill((255, 255, 0), special\_flags=BLEND\_MULT)  
 temp = temp.came\_from\_block  
 screen.blit(background, (0, 0))  
 all\_sprites.draw(screen)  
 pg.display.update()  
  
 loop = True  
 while loop:  
 clock.tick(60)  
 *# 处理输入事件* for my\_event in pg.event.get():  
 if my\_event.type == pg.QUIT:  
 loop = False  
 elif my\_event.type == KEYDOWN:  
 loop = False  
 elif my\_event.type == MOUSEBUTTONUP:  
 loop = False  
  
 pg.quit()  
  
  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()