**A\*算法——寻路算法**

**代码：**

**from** random **import** randint  
  
  
**class** SearchEntry():  
 **def** \_\_init\_\_(self, x, y, g\_cost, f\_cost=0, pre\_entry=**None**):  
 self.x = x  
 self.y = y  
 *# cost move form start entry to this entry* self.g\_cost = g\_cost  
 self.f\_cost = f\_cost  
 self.pre\_entry = pre\_entry  
  
 **def** getPos(self):  
 **return** (self.x, self.y)  
  
  
**class** Map():  
 **def** \_\_init\_\_(self, width, height):  
 self.width = width  
 self.height = height  
 self.map = [[0 **for** x **in** range(self.width)] **for** y **in** range(self.height)]  
  
 **def** createBlock(self, block\_num):  
 **for** i **in** range(block\_num):  
 x, y = (randint(0, self.width - 1), randint(0, self.height - 1))  
 self.map[y][x] = 1  
  
 **def** generatePos(self, rangeX, rangeY):  
 x, y = (randint(rangeX[0], rangeX[1]), randint(rangeY[0], rangeY[1]))  
 **while** self.map[y][x] == 1:  
 x, y = (randint(rangeX[0], rangeX[1]), randint(rangeY[0], rangeY[1]))  
 **return** (x, y)  
  
 **def** showMap(self):  
 print(**"+"** \* (3 \* self.width + 2))  
  
 **for** row **in** self.map:  
 s = **'+'  
 for** entry **in** row:  
 s += **' '** + str(entry) + **' '** s += **'+'** print(s)  
  
 print(**"+"** \* (3 \* self.width + 2))  
  
  
**def** AStarSearch(map, source, dest):  
 **def** getNewPosition(map, locatioin, offset):  
 x, y = (location.x + offset[0], location.y + offset[1])  
 **if** x < 0 **or** x >= map.width **or** y < 0 **or** y >= map.height **or** map.map[y][x] == 1:  
 **return None  
 return** (x, y)  
  
 **def** getPositions(map, location):  
 *# use four ways or eight ways to move* offsets = [(-1, 0), (0, -1), (1, 0), (0, 1)]  
 *# offsets = [(-1,0), (0, -1), (1, 0), (0, 1), (-1,-1), (1, -1), (-1, 1), (1, 1)]* poslist = []  
 **for** offset **in** offsets:  
 pos = getNewPosition(map, location, offset)  
 **if** pos **is not None**:  
 poslist.append(pos)  
 **return** poslist  
  
 *# imporve the heuristic distance more precisely in future* **def** calHeuristic(pos, dest):  
 **return** abs(dest.x - pos[0]) + abs(dest.y - pos[1])  
  
 **def** getMoveCost(location, pos):  
 **if** location.x != pos[0] **and** location.y != pos[1]:  
 **return** 1.4  
 **else**:  
 **return** 1  
  
 *# check if the position is in list* **def** isInList(list, pos):  
 **if** pos **in** list:  
 **return** list[pos]  
 **return None** *# add available adjacent positions* **def** addAdjacentPositions(map, location, dest, openlist, closedlist):  
 poslist = getPositions(map, location)  
 **for** pos **in** poslist:  
 *# if position is already in closedlist, do nothing* **if** isInList(closedlist, pos) **is None**:  
 findEntry = isInList(openlist, pos)  
 h\_cost = calHeuristic(pos, dest)  
 g\_cost = location.g\_cost + getMoveCost(location, pos)  
 **if** findEntry **is None**:  
 *# if position is not in openlist, add it to openlist* openlist[pos] = SearchEntry(pos[0], pos[1], g\_cost, g\_cost + h\_cost, location)  
 **elif** findEntry.g\_cost > g\_cost:  
 *# if position is in openlist and cost is larger than current one,  
 # then update cost and previous position* findEntry.g\_cost = g\_cost  
 findEntry.f\_cost = g\_cost + h\_cost  
 findEntry.pre\_entry = location  
  
 *# find a least cost position in openlist, return None if openlist is empty* **def** getFastPosition(openlist):  
 fast = **None  
 for** entry **in** openlist.values():  
 **if** fast **is None**:  
 fast = entry  
 **elif** fast.f\_cost > entry.f\_cost:  
 fast = entry  
 **return** fast  
  
 openlist = {}  
 closedlist = {}  
 location = SearchEntry(source[0], source[1], 0.0)  
 dest = SearchEntry(dest[0], dest[1], 0.0)  
 openlist[source] = location  
 **while True**:  
 location = getFastPosition(openlist)  
 **if** location **is None**:  
 *# not found valid path* print(**"can't find valid path"**)  
 **break**;  
  
 **if** location.x == dest.x **and** location.y == dest.y:  
 **break** closedlist[location.getPos()] = location  
 openlist.pop(location.getPos())  
 addAdjacentPositions(map, location, dest, openlist, closedlist)  
  
 *# mark the found path at the map* **while** location **is not None**:  
 map.map[location.y][location.x] = 2  
 location = location.pre\_entry  
  
  
WIDTH = 10  
HEIGHT = 10  
BLOCK\_NUM = 15  
map = Map(WIDTH, HEIGHT)  
map.createBlock(BLOCK\_NUM)  
map.showMap()  
  
source = map.generatePos((0, WIDTH // 3), (0, HEIGHT // 3))  
dest = map.generatePos((WIDTH // 2, WIDTH - 1), (HEIGHT // 2, HEIGHT - 1))  
print(**"source:"**, source)  
print(**"dest:"**, dest)  
AStarSearch(map, source, dest)  
map.showMap()

程序运行结果：

