**一、实验题目**

用递归或非递归方式实现迷宫路径求解

**二、实验目的：**

1、掌握递归的基本概念。

2、熟悉掌握递归的使用方法。

**三、实验设备与环境**

微型计算机、macOS 系列操作系统 、Pycharm系列软件

**四、实验内容**

利用归或非递归方式实现迷宫路径求解。

**五、概要设计（思路、算法、步骤等）**

**思路**

使用深度优先搜索（DFS）算法来搜索迷宫的出路。在搜索的过程中，按照一定的优先级顺序来探索周围的路径：首先向右走到底，如果右边的路可以走且没有走过，就继续往右边走。如果右边的路不能走，就依次尝试下边、左边、上边是否可以走，如果可以走就往这个方向走。如果四个方向都不能走了，就回溯到上一个经过的位置，重新探索四个方向，直到找到出口。

在深度优先搜索算法中，每次搜索时都会沿着一个路径走到底，直到到达一个死胡同，然后回溯到上一个经过的位置，继续探索其他路径。这种方法会一直尝试探索所有可能的路径，直到找到出口或者确定没有出路为止。

深度优先搜索算法的时间复杂度较高，因为它可能会搜索整个迷宫，所以在实际应用中，通常需要对算法进行优化，如剪枝等，以提高搜索效率。

**算法**

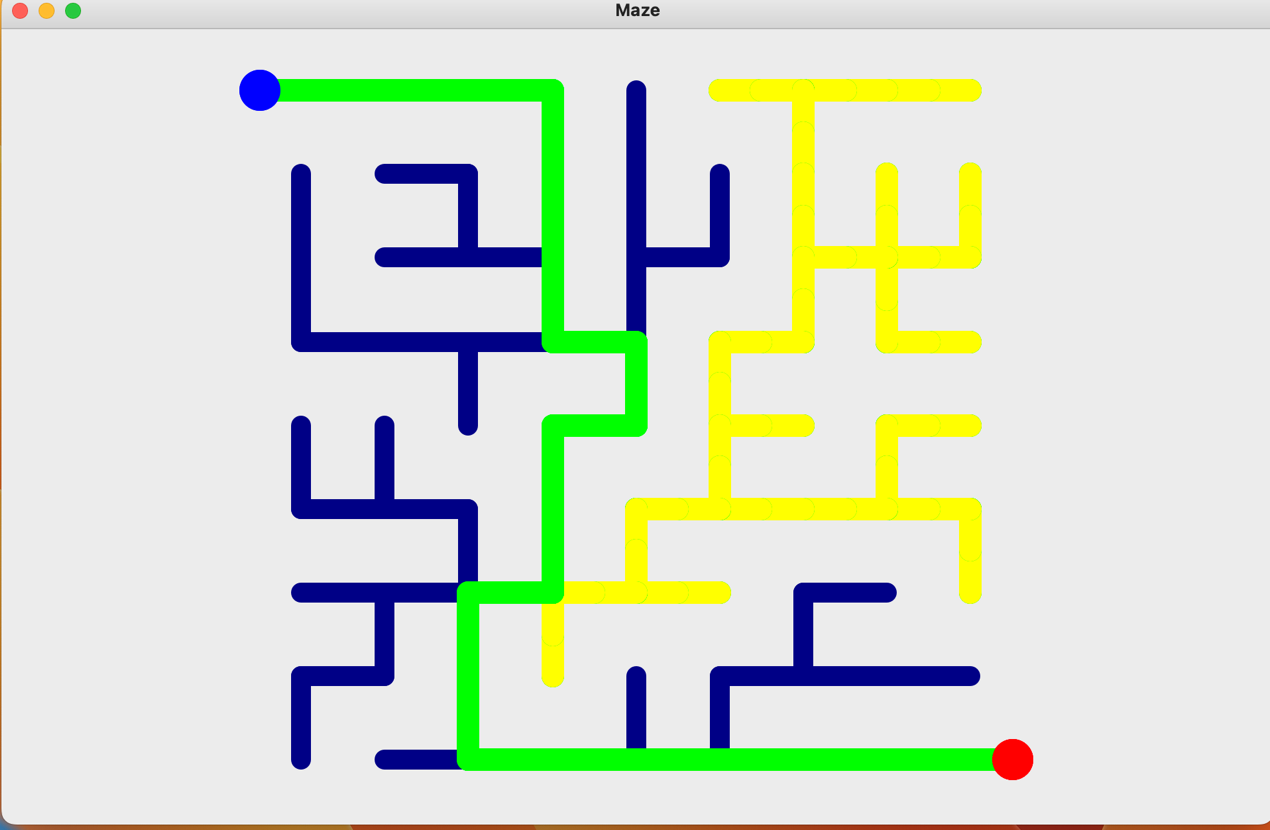
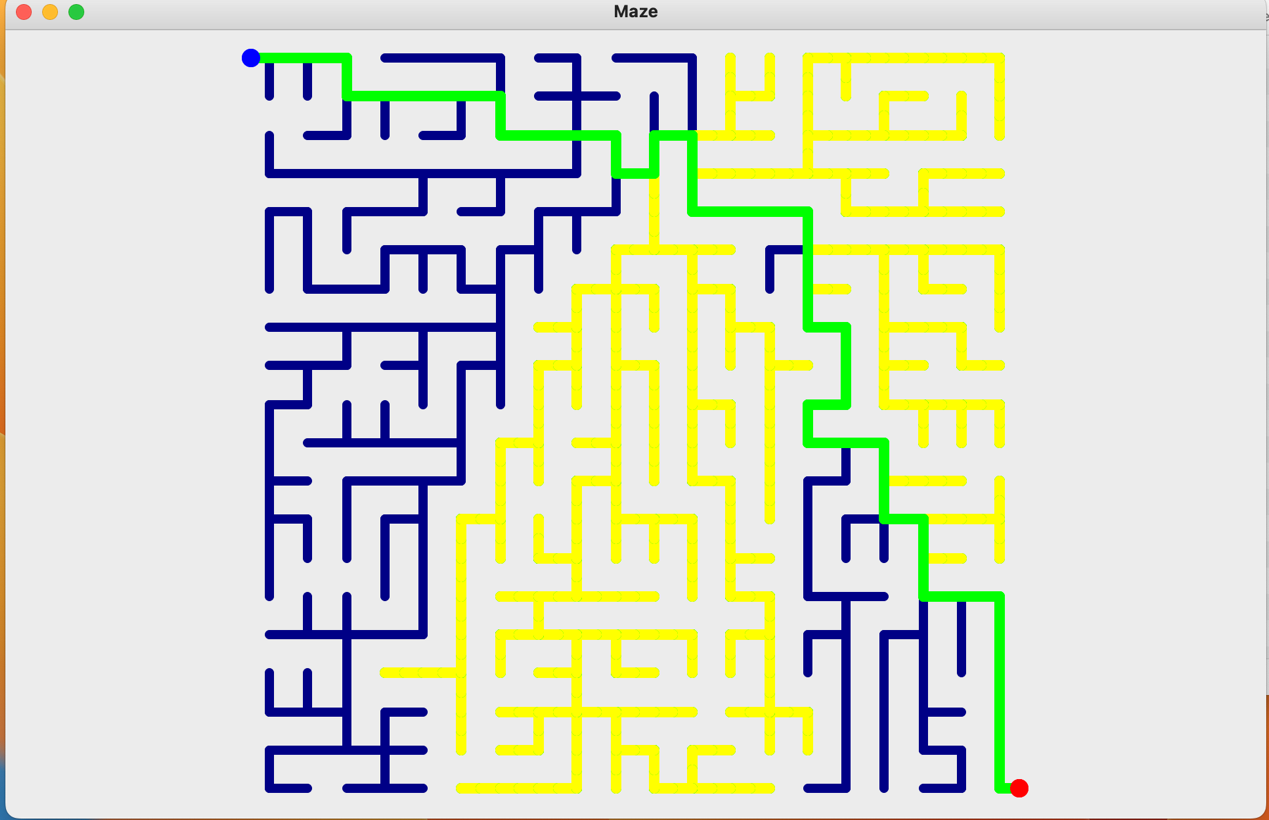
为了寻找迷宫的出路，我使用了**递归算法**实现深度优先搜索。在搜索的过程中，我使用**栈**来存储已经访问过的点的坐标，以便于回溯。每次访问一个新的点时，我们将它的坐标入栈，并标记该点已经访问过。如果该点是出口，说明我们已经找到了出路，就可以结束搜索并输出结果。如果该点不是出口，则继续搜索它周围的点，如果有可到达的点，就递归调用自己来继续搜索。如果周围的点都不能到达，就需要回溯到上一个点。从栈中弹出最近访问的点的坐标，并把它和上一个点的坐标用黄线相连，表示我们走了一条回头路。然后，我们再弹出上一个点的坐标，并调用自己来继续搜索。

在递归函数 find\_path(self, x, y) 中，首先，将当前点的坐标压入栈中，并标记该点已经访问过。然后，检查该点是否是出口。如果是出口，就结束搜索并输出结果。如果不是出口，搜索该点周围的可到达点。如果找到可到达点，递归调用 find\_path 函数来搜索这些点。如果周围没有可到达点，需要回溯。从栈中弹出最近访问的点的坐标，并把它和上一个点的坐标用黄线相连。然后，再弹出上一个点的坐标，并调用自己来继续搜索。

**六、详细设计（核心代码、算法流程图等）**

import os  
  
from PyQt5.QtGui import QMouseEvent  
from PyQt5.QtWidgets import QApplication  
import sys  
from utils.maze\_gui import MazeGui  
from PyQt5.QtCore import Qt  
from stack import Stack  
  
class DfsQMaze(MazeGui):  
 def \_\_init\_\_(self, a=5, b=20):  
 super(DfsQMaze, self).\_\_init\_\_(a, b) *# 父类初始化  
 # print("start:{},end:{}".format(self.ENTRANCE, self.EXIT))  
 # print(self.MAZE)* def paintEvent(self, paintEvent):  
 *"""  
 【注】QWidigets类的方法，所有画图的操作放在这里。实例化时，会自动调用这个方法。  
 """* super(DfsQMaze, self).paintEvent(paintEvent) *# 父类绘画界面初始化* self.paintDfs()  
  
 def paintDfs(self):  
 *"""将递归的过程画出来，必须通过`paintEvent()`方法调用  
 【注】如果在`dfs()`中使用了相关画图、画线等函数，即可在窗口中绘制出来  
 """* self.dfs()  
 self.draw\_point(self.ENTRANCE, self.pen, self.painter, \_color=Qt.blue)  
 self.draw\_point(self.EXIT, self.pen, self.painter, \_color=Qt.red)  
 del self.pen, self.painter  
  
 def dfs(self):  
 self.xl = len(self.MAZE) *# 迷宫纵向总长度* self.yl = len(self.MAZE[0]) *# 迷宫横向总长度* self.mtemp = [] *# 用来存放到过的位置，若来过该点则为 1* [self.xend, self.yend] = self.EXIT *# 迷宫出口坐标* for x in range(0, self.xl):  
 list = []  
 for y in range(0, self.yl):  
 list.append(0)  
 self.mtemp.append(list) *# 将mtemp列表全部置零* [x, y] = self.ENTRANCE *# 迷宫入口坐标* mystack.push(x) *# 将迷宫第一个位置压入栈* mystack.push(y)  
 self.find\_path(x, y) *# 开始寻找出口* def get\_xy(self): *# 返回栈顶元素x，y为当前处理的坐标* y = mystack.top()  
 mystack.pop()  
 x = mystack.top()  
 mystack.push(y) *# 把y放回去* return x, y  
  
 def find\_path(self, x, y): *# 递归实现* x0, y0 = self.get\_xy() *# 获取栈顶元素坐标（上一个经过位置）* mystack.push(x) *# 将现在处理的坐标压入栈中* mystack.push(y) *# 将现在处理的坐标和栈顶元素坐标用绿线相连，说明走过该条路* self.drawLine([x0, y0], [x, y], color=Qt.green, width\_scale=0.55)  
 self.mtemp[x][y] = 1 *# 将走过的点赋值为1* if x == self.xend and y == self.yend: *# 若现在处理的点是出口坐标，则函数结束，找到出口* return  
 else:  
 *# 顺序：右下左上  
 # 若右边坐标未超出迷宫，且右边不是墙，且右边没有走到过，则往右边走（调用本函数，递归）* if y + 1 < self.yl and self.MAZE[x][y + 1] != 1 and self.mtemp[x][y + 1] == 0:  
 self.find\_path(x, y + 1)  
 *# 若右边不行，则往下走* elif x + 1 < self.xl and self.MAZE[x + 1][y] != 1 and self.mtemp[x + 1][y] == 0:  
 self.find\_path(x + 1, y)  
 *# 若右边、下边不行，则往左边走* elif y - 1 >= 0 and self.MAZE[x][y-1] != 1 and self.mtemp[x][y-1] == 0:  
 self.find\_path(x, y - 1)  
 *# 若右边、下边，左边都不行，则往上边走* elif x - 1 >= 0 and self.MAZE[x - 1][y] != 1 and self.mtemp[x - 1][y] == 0:  
 self.find\_path(x - 1, y)  
 *# 若四个位置都不能走，则说明近死胡同了，这条路不能走路，返回上一个位置* else: *# 死胡同* x0, y0 = self.get\_xy() *# 获取现在这个点的位置* mystack.pop() *# 把上一个位置从栈中弹出* mystack.pop()  
 y = mystack.top() *# 获取上一个点的位置* mystack.pop()  
 x = mystack.top()  
 mystack.pop()  
 *# 将上一个点到死胡同的路标黄，作为记录* self.drawLine([x0, y0], [x, y], color=Qt.yellow, width\_scale=0.55)  
 self.find\_path(x, y) *# 到上一个点寻找出路* def mouseDoubleClickEvent(self, a0: QMouseEvent): *# 鼠标双击事件，重新启动，达到刷新效果* python = sys.executable  
 os.execl(python, python, \*sys.argv)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 mystack = Stack()  
 application = QApplication(sys.argv)  
 solver = DfsQMaze()  
 sys.exit(application.exec\_())

**七、测试结果及分析：**



**八、总结**

当在迷宫中寻找通路时，我使用了递归算法，遵循深度优先搜索的策略，即“走一条路走到黑，再往上回溯”。虽然这种方法可能会走许多没有用的路，而且找到的通路也不一定是最短路径，但其优点也十分明显，代码量较小，使用递归方法可以处理相对复杂的问题。这次实验深入了解递归算法，对我理解该算法的原理和应用有了更深入的认识。

除了深度优先搜索算法，还有广度优先搜索算法等多种搜索算法。每种算法都有自己的优点和适用场景。深度优先搜索虽然在寻找最短路径方面不如其他算法，但是在处理一些复杂的非最短路径问题上，具有明显的优势。在实际应用中，根据实际需求选择适合的搜索算法非常重要。