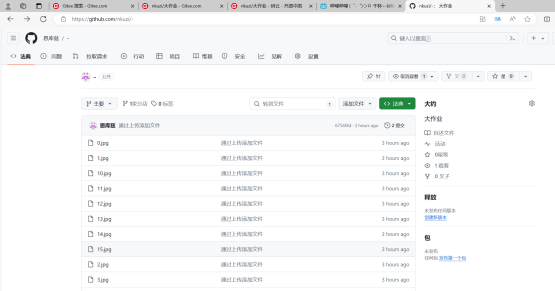
高级语言程序设计大作业实验报告

1. 作业题目

连连看之牢大重生

前言：本项目为致敬牢大，希望大家不要带节奏，也不要再把牢大和冰红茶弄在一起了0.0；

GitHub：<https://github.com/nkuzi/->



Gitee：<https://gitee.com/nkuzi/big-homework/tree/master5.0>

或再Gitee上的：<https://gitee.com/nkuzi/big-homework/branches找master5.0分支>

说明一下，GitHub是真的卡，耐心一点吧（建议）

图片和音频我都放在GitHub，代码我放在gitee，因为我不知道gitee怎么放图片和音频。

b站地址（视频介绍）：https://www.bilibili.com/video/BV1CH4y137iP/?spm\_id\_from=333.1007.0.0&vd\_source=a0120c3f9409f1156a7ddbe710bef66e

还有就是弄到代码的话，尽量不要构建项目，因为构建出来的exe文件可能因为缺少ddl文件无法运行，这个电脑或者qt配置有关，我的电脑不行。但是CTRL+R运行还是可以玩的，所以建议CTRL+R。

1. 开发软件

QT 5.14.2

1. 课题要求

面向对象

单元测试

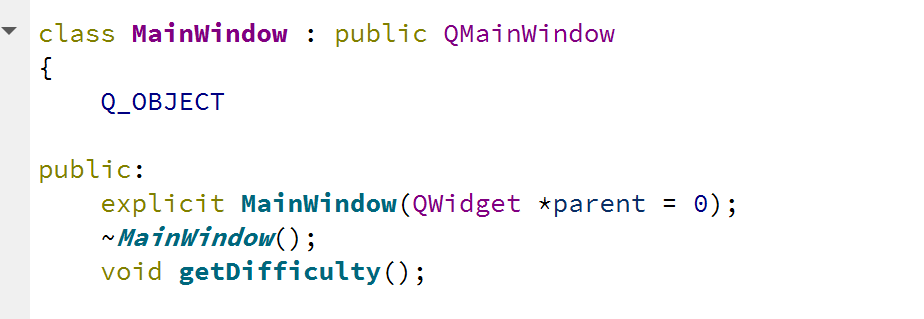
模型部分

验证

1. 主要流程
2. 整体流程

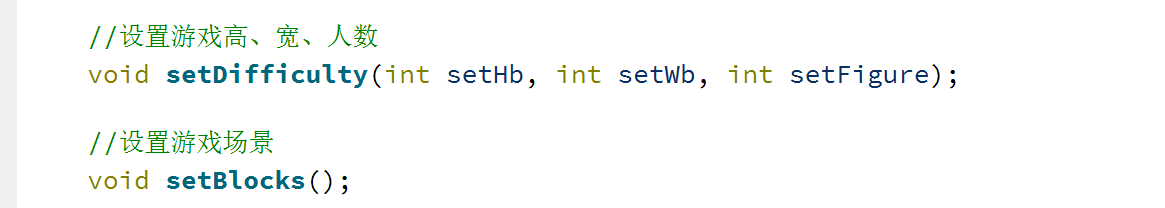
我们先建一个这个类

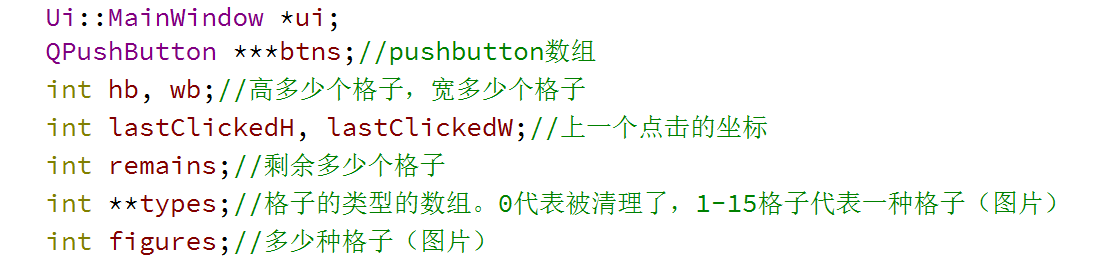
这个类连接我们的窗口，同时实现我们的游戏



首先搞连连看的话，我们先造地图，不过我这里没给他添个背景，就没那么麻烦。

然后就是放格子了。连连看就是二维数组，即搞个矩阵就好了，我这里设置了3个难度的关卡，所以3个的大小（图片数量）是不一样的。

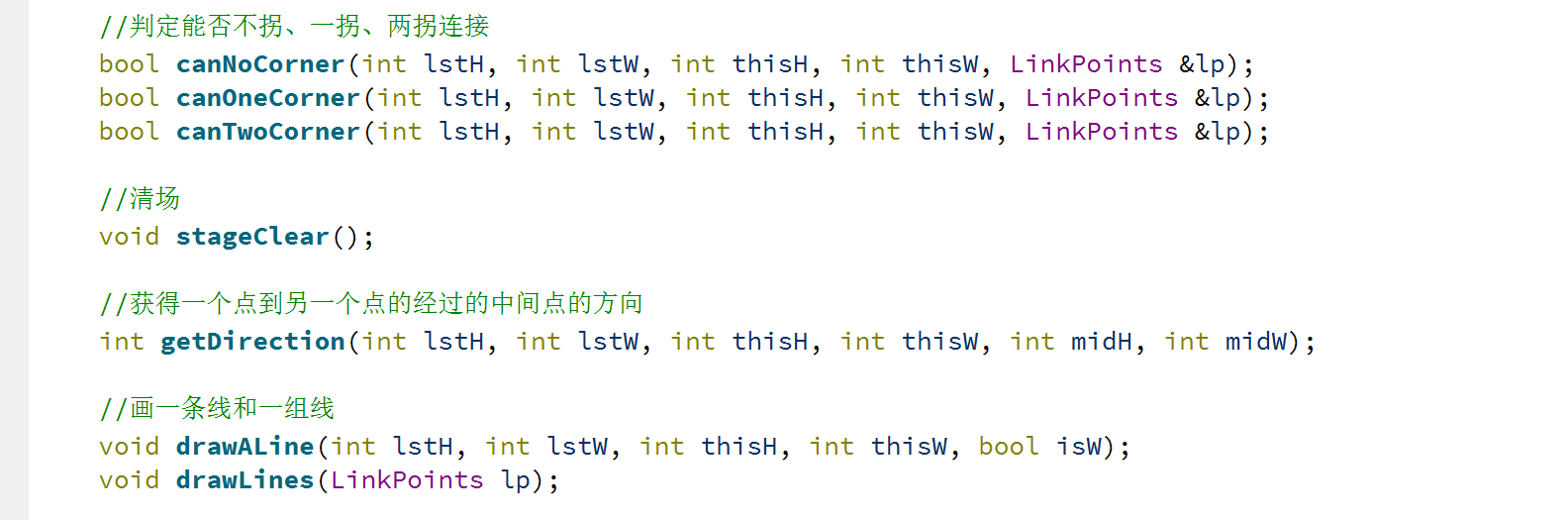




大概布局如上。

接着就是我们的算法了，这个是比较复杂的，关于能不能连的问题。

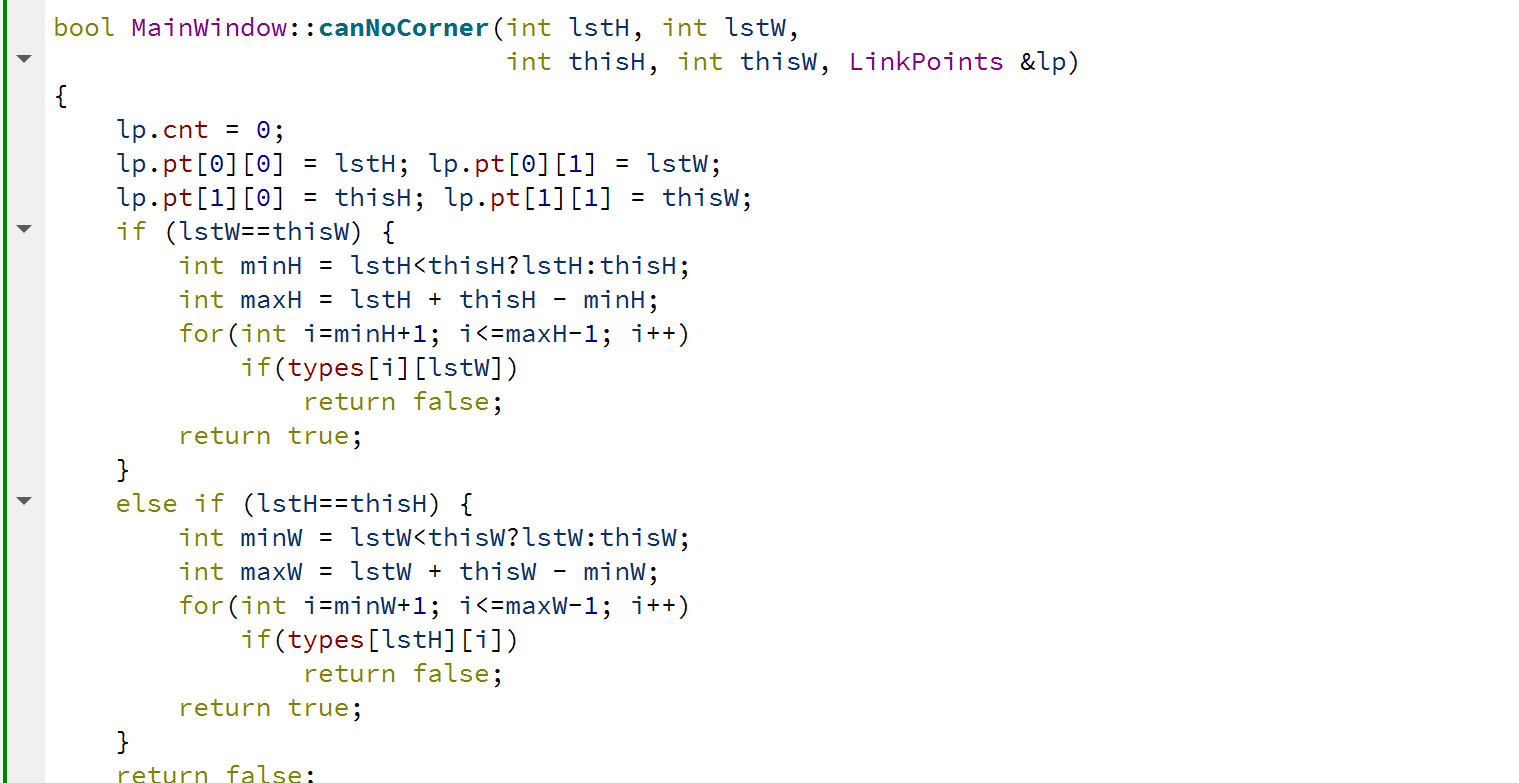
连了之后就是显示路径了，最后就是结束之类的，是否要继续玩



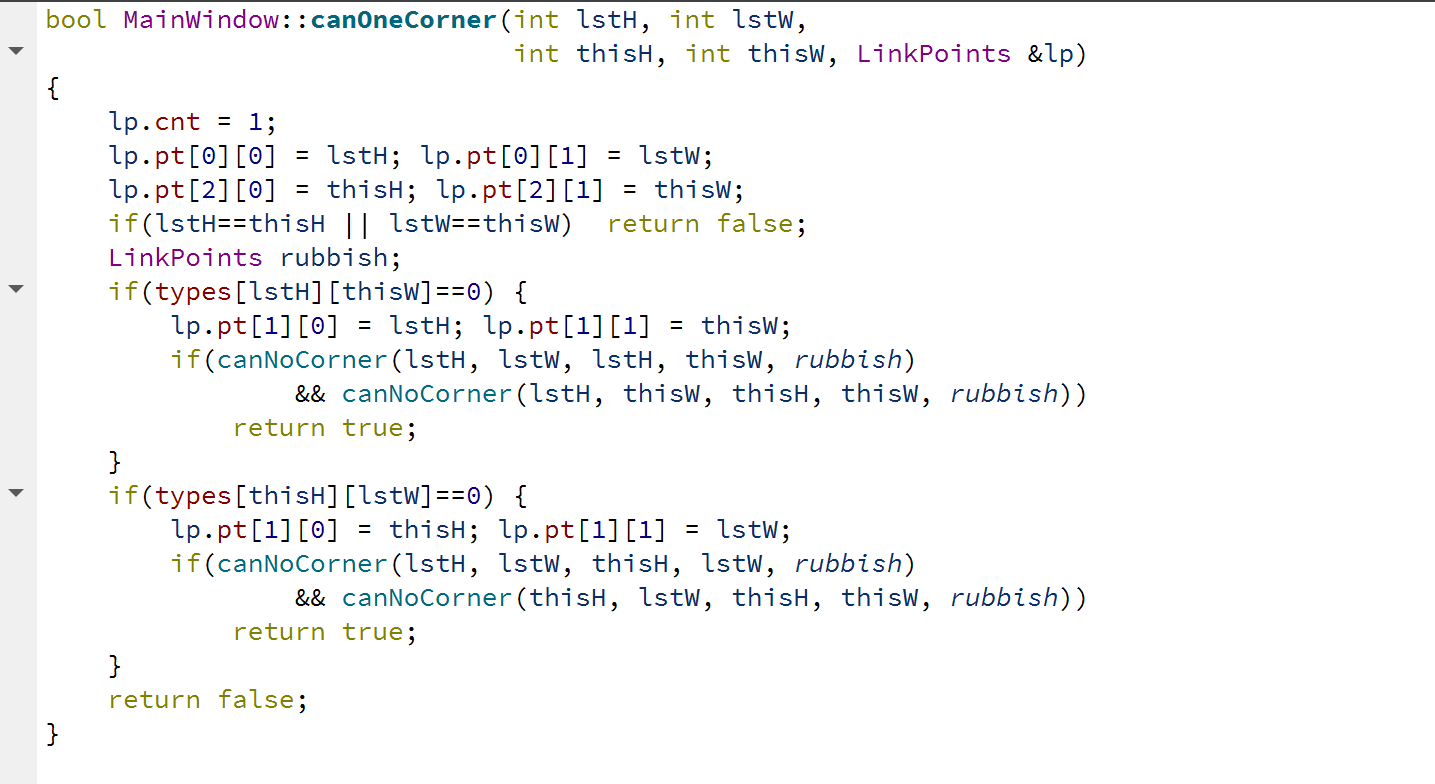


1. 算法与公式

先来重要的，能不能连：



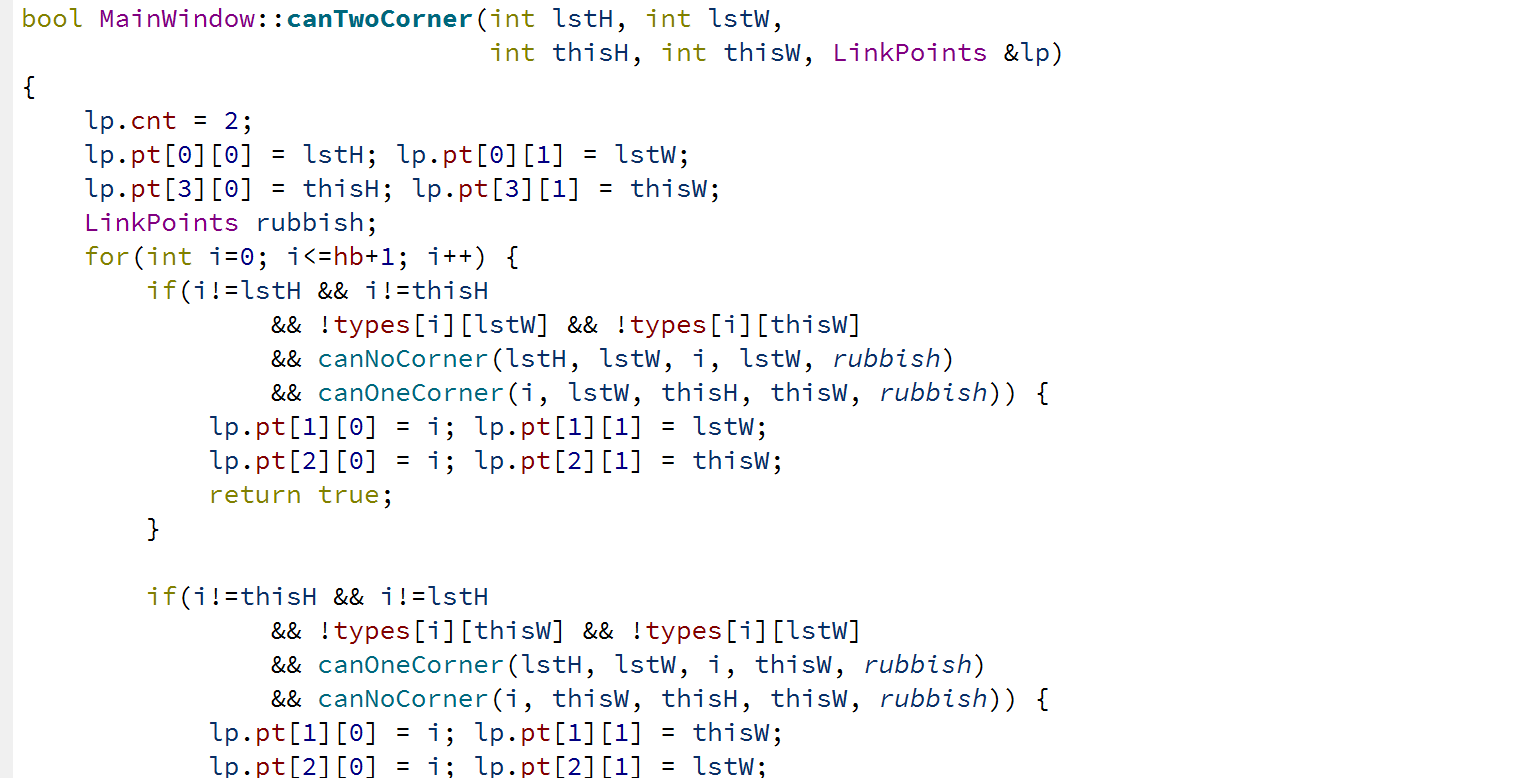
这里是分水平和垂直的，只要中间没有格子就可以连了；

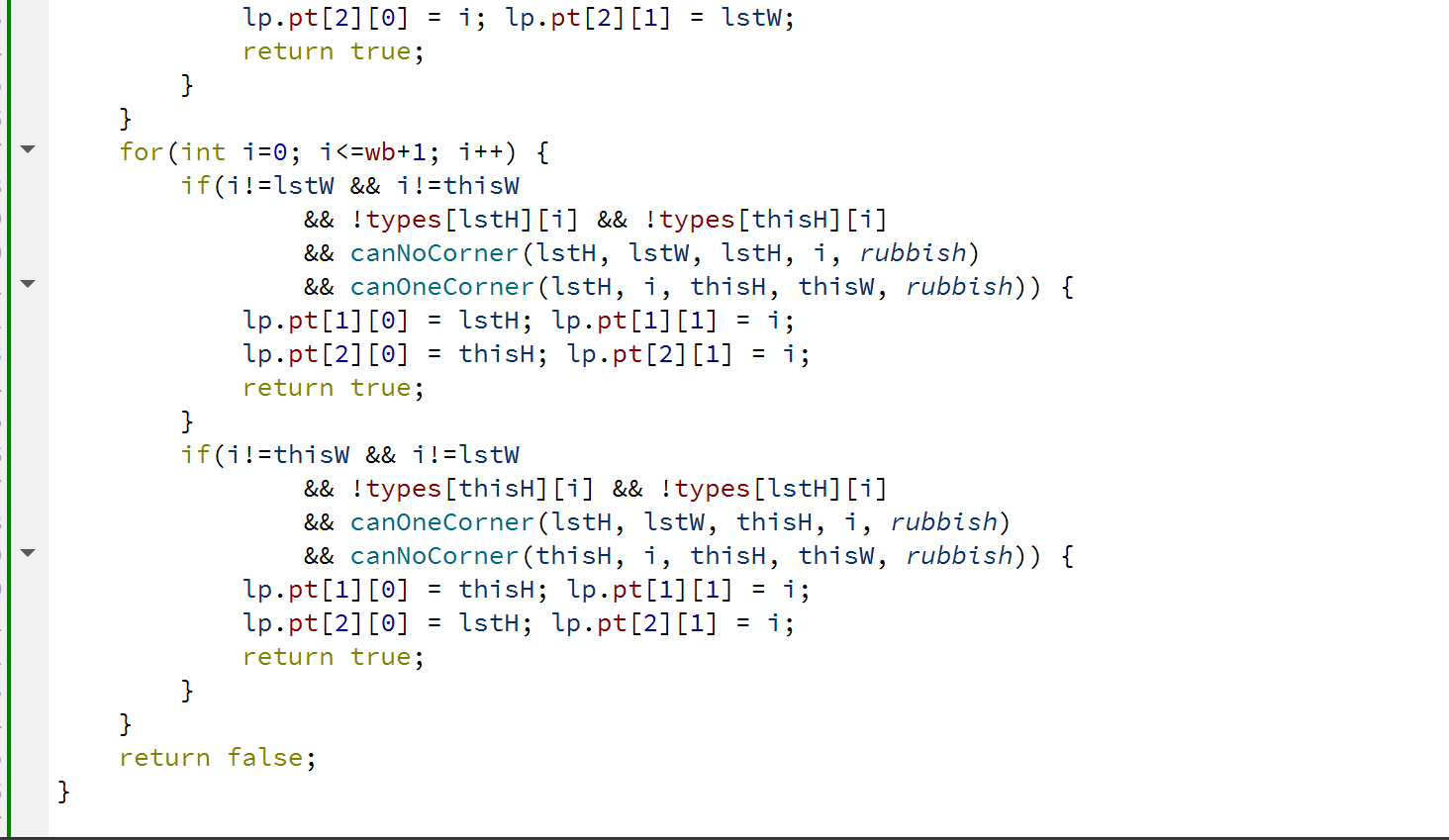


这里就是拐角了一次的，大概思路是这样的：

这个拐角的x要么和起点（鼠标点的第一个）一样，y和终点（鼠标点的第二个）一样，

要么就反过来，所以我们又回到第一种（水平或者垂直的问题了）；



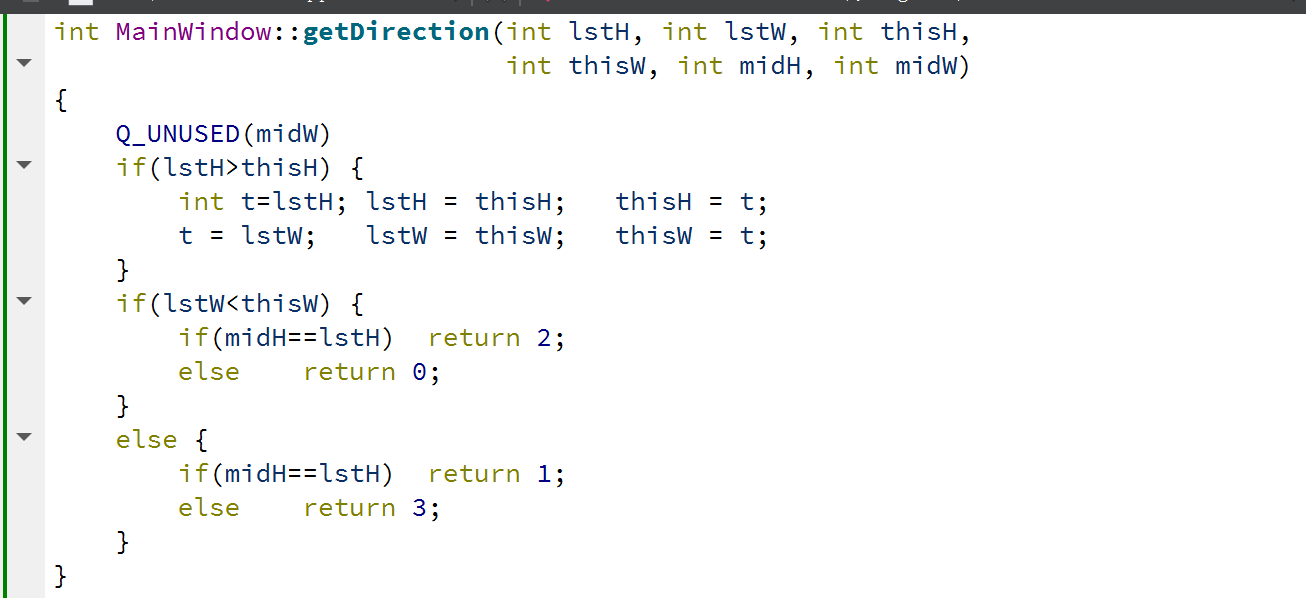


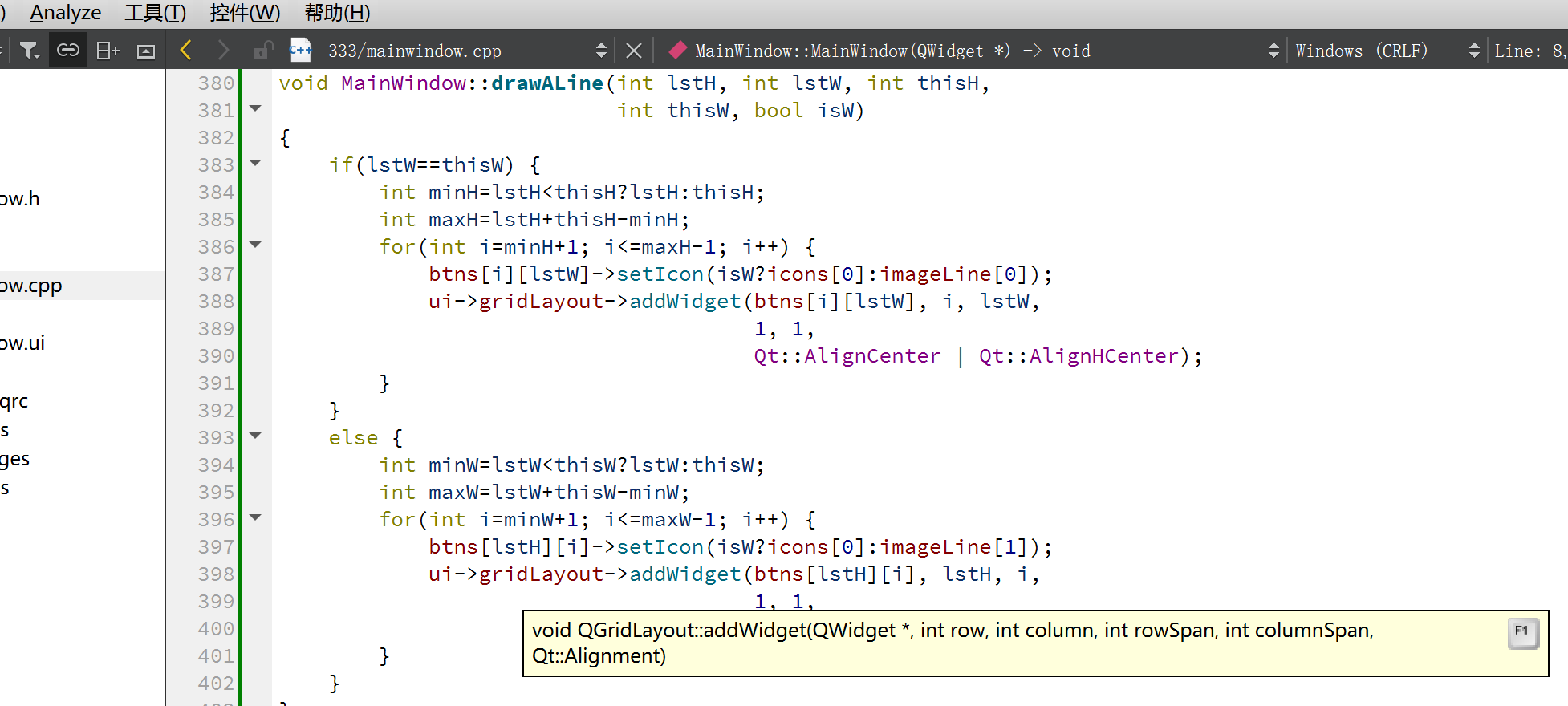
这个拐角两次就会难很多，我也是想了好久，然后在网上找了一些算法

就是还是需要你把起点和终点的x和y方向遍历，然后在同一个方向上你可以找到与这个方向垂直的一个方向，差不多找一条线，然后这个路径没有阻碍；

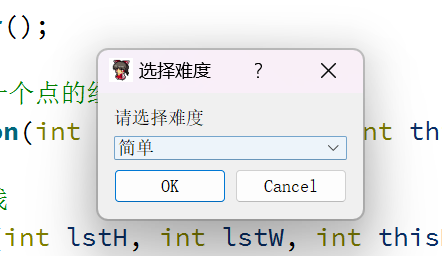
其实剩下的都是次要的，能不能连是最关键的；

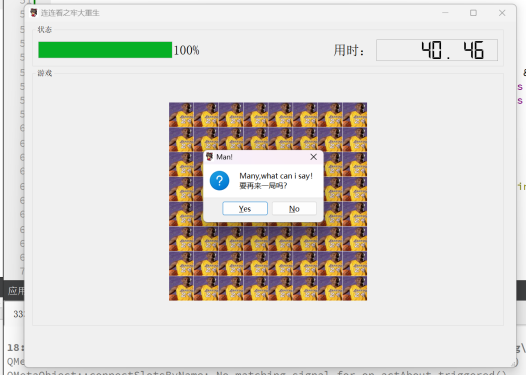
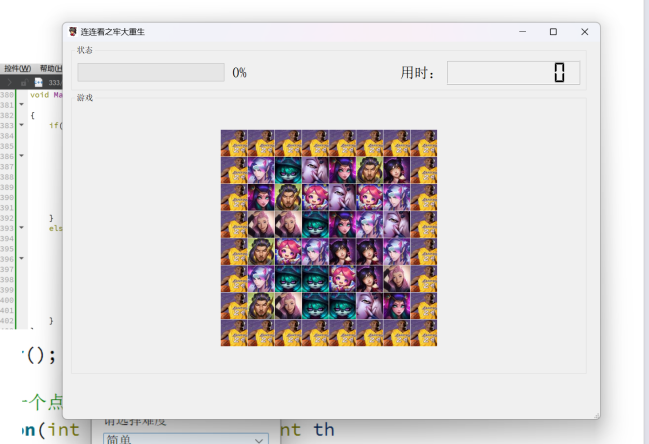


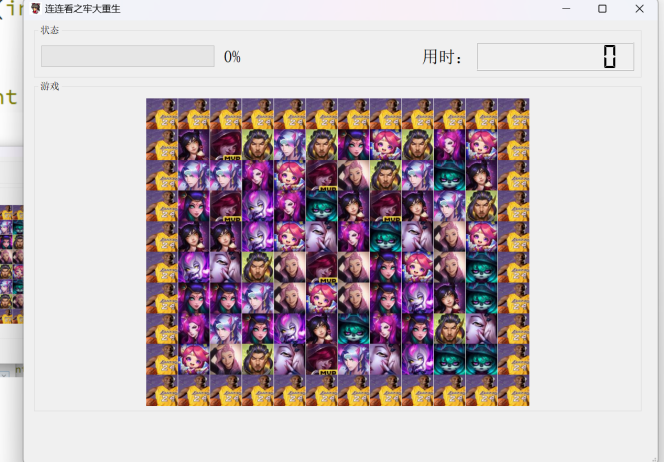
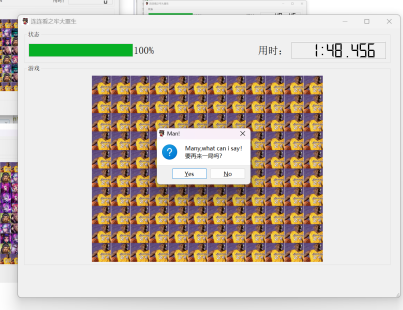
划线的话，就是跟着上一个连的路径走即可drawline函数完成。

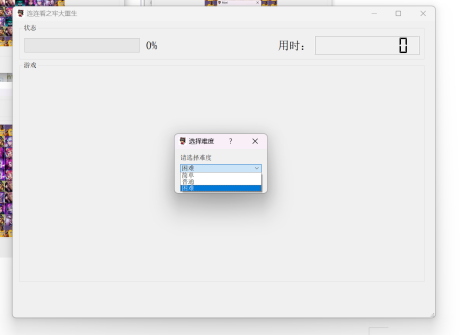
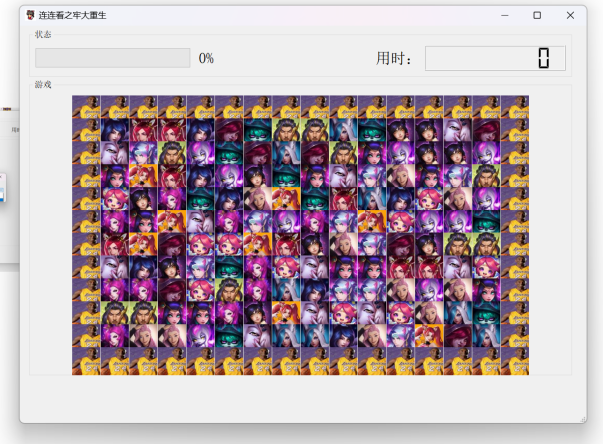
Getdirection函数帮我完成找连接的路径，中间有没有拐点之类的，给他连上；

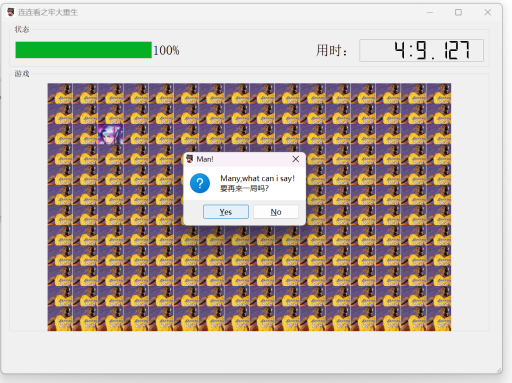
1. 单元测试：

五、测试三个难度这是简单的



这是普通的

这是困难的



现在三个关卡都测试完了。

注意：游戏会有bug，比如说不相同的方块也一起消除了，还有没点完的也结束了。不过游戏也是有限制的，就是能不能连只有4个方式（如果前2个归为一个的话，就算3个）：水平连、垂直连、一个拐角的连、两个拐角的连。实际上，我们玩的时候会发现3个拐角也可以连，不过是我们人为的一种做法，我现在计算机水平还不能达到这种跟我想的一样的。也算是一个游戏规则吧。

上述为测试结果

六、收获

1.无疑是那个能不能连的算法

从简单的水平或垂直连，到一个拐角连，两个拐角；

这个算法从逻辑上来说是层层递进的，就是后面一个算法可以用到前面的算法

bool MainWindow::**canNoCorner**(int lstH, int lstW,

int thisH, int thisW, LinkPoints &lp)

{

lp.cnt = 0;

lp.pt[0][0] = lstH; lp.pt[0][1] = lstW;

lp.pt[1][0] = thisH; lp.pt[1][1] = thisW;

if (lstW==thisW) {

int minH = lstH<thisH?lstH:thisH;

int maxH = lstH + thisH - minH;

for(int i=minH+1; i<=maxH-1; i++)

if(types[i][lstW])

return false;

return true;

}

else if (lstH==thisH) {

int minW = lstW<thisW?lstW:thisW;

int maxW = lstW + thisW - minW;

for(int i=minW+1; i<=maxW-1; i++)

if(types[lstH][i])

return false;

return true;

}

return false;

}

bool MainWindow::**canOneCorner**(int lstH, int lstW,

int thisH, int thisW, LinkPoints &lp)

{

lp.cnt = 1;

lp.pt[0][0] = lstH; lp.pt[0][1] = lstW;

lp.pt[2][0] = thisH; lp.pt[2][1] = thisW;

if(lstH==thisH || lstW==thisW) return false;

LinkPoints rubbish;

if(types[lstH][thisW]==0) {

lp.pt[1][0] = lstH; lp.pt[1][1] = thisW;

if(canNoCorner(lstH, lstW, lstH, thisW, *rubbish*)

&& canNoCorner(lstH, thisW, thisH, thisW, *rubbish*))

return true;

}

if(types[thisH][lstW]==0) {

lp.pt[1][0] = thisH; lp.pt[1][1] = lstW;

if(canNoCorner(lstH, lstW, thisH, lstW, *rubbish*)

&& canNoCorner(thisH, lstW, thisH, thisW, *rubbish*))

return true;

}

return false;

}

bool MainWindow::**canTwoCorner**(int lstH, int lstW,

int thisH, int thisW, LinkPoints &lp)

{

lp.cnt = 2;

lp.pt[0][0] = lstH; lp.pt[0][1] = lstW;

lp.pt[3][0] = thisH; lp.pt[3][1] = thisW;

LinkPoints rubbish;

for(int i=0; i<=hb+1; i++) {

if(i!=lstH && i!=thisH

&& !types[i][lstW] && !types[i][thisW]

&& canNoCorner(lstH, lstW, i, lstW, *rubbish*)

&& canOneCorner(i, lstW, thisH, thisW, *rubbish*)) {

lp.pt[1][0] = i; lp.pt[1][1] = lstW;

lp.pt[2][0] = i; lp.pt[2][1] = thisW;

return true;

}

if(i!=thisH && i!=lstH

&& !types[i][thisW] && !types[i][lstW]

&& canOneCorner(lstH, lstW, i, thisW, *rubbish*)

&& canNoCorner(i, thisW, thisH, thisW, *rubbish*)) {

lp.pt[1][0] = i; lp.pt[1][1] = thisW;

lp.pt[2][0] = i; lp.pt[2][1] = lstW;

return true;

}

}

for(int i=0; i<=wb+1; i++) {

if(i!=lstW && i!=thisW

&& !types[lstH][i] && !types[thisH][i]

&& canNoCorner(lstH, lstW, lstH, i, *rubbish*)

&& canOneCorner(lstH, i, thisH, thisW, *rubbish*)) {

lp.pt[1][0] = lstH; lp.pt[1][1] = i;

lp.pt[2][0] = thisH; lp.pt[2][1] = i;

return true;

}

if(i!=thisW && i!=lstW

&& !types[thisH][i] && !types[lstH][i]

&& canOneCorner(lstH, lstW, thisH, i, *rubbish*)

&& canNoCorner(thisH, i, thisH, thisW, *rubbish*)) {

lp.pt[1][0] = thisH; lp.pt[1][1] = i;

lp.pt[2][0] = lstH; lp.pt[2][1] = i;

return true;

}

}

return false;

}