**西安交通大学**

**C#开放实验-----连连看**

**韦冈松 装控41 2141805040**

**刘宗杰 机械43 2140101062**

## 项目名称

连连看游戏

## 游戏设计

1. 游戏场上有20\*8个方块，方块有五种图案。
2. 点击两个相同图案的方块，如果连接它们之间的连线在三根以内，且不经过其他的方块，则可以将它们消除。
3. 将所有方块消除即为游戏胜利。

## 程序框架

程序主体由窗体Form1和自定义的游戏管理类class llk构成。

Form1包含一个名为canvas的Label用于显示游戏场景，添加“开始”，“退出”等菜单项，同时作为主窗口管理点击（canvas1\_Click()）、重绘（canvas1\_Paint()）和开始游戏事件。

llk对象包含一个二维数组，储存场景中各个块的内容,包含各种方块图片、状态信息、算法。

两个类的成员和之间的调用关系（用箭头表示）如下（仅列出主要成员）：

class Form1 : Form {

//\*\*\*\*\*\*\*\*\*\*字段

Label canvas1; //显示画布

llk L; //游戏管理类

//\*\*\*\*\*\*\*\*\*\*方法

canvas1\_Click();

canvas1\_Paint();

开始MenuItem\_Click(); //开始

}

class llk {

//\*\*\*\*\*\*\*\*\*\*字段

int[,] container=new int[20,8]; //储存20行8列的方块的内容

Bitmap img1, img2, img3, img4, img5, decoration; //五种类型方块的图片，和被选中的方块外面加的边框

bool isFocus; //是否有方块被选中

int lastm, lastn; //上一个选中的方块的位置

Point pointToDraw; //点击的坐标

//\*\*\*\*\*\*\*\*\*\*方法

void drawcanvas(); //绘制整个画布

void drawdiv(); //点选、消除方块

bool canLink(int x1, int y1, int x2, int y2); //判断两个位置的方块是否可以连接消除

void reset(); //开始，随机填充方块

}

## 算法分析

本游戏的核心算法有：随机填充方块的算法，判断两个方块是否可以连接消除

1. 随机填充方块

容器二维数组container中，一个方块的值为0表示没有方块，1-5表示5种不同的方块。用Random类生成随机数，在某个随机位置填充特定方块，再随机出一个数，找一个没有填充过的位置填充同样的方块，直到填充完。

1. 判断两个方块是否可以连接消除

判断是否可连接是该游戏的核心算法。

可连接有以下几种情况：

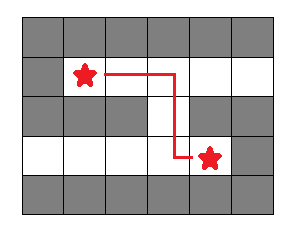
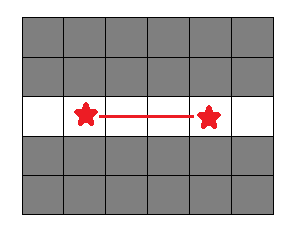
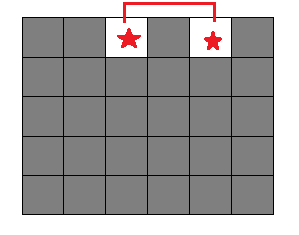
①如果两个都在边上，直接可以连接；

②如果不在边上，创建一个和容器同样大小的bool型二维数组，从方块的两边拓展，记录没有障碍的位置；如果两个方块在同一行，且扩展区域重叠，则可以连接；

③如果不在同一行，从左到右遍历两行之间的纵向连线，如果纵向连线上没有障碍，且连接两个扩展区域，则可以连接。如果都不满足，即两个方块不可连接。

如图,红色星形为点击的方块,灰色为障碍,蓝色箭头表示扩展方向,红色线条表示连线.

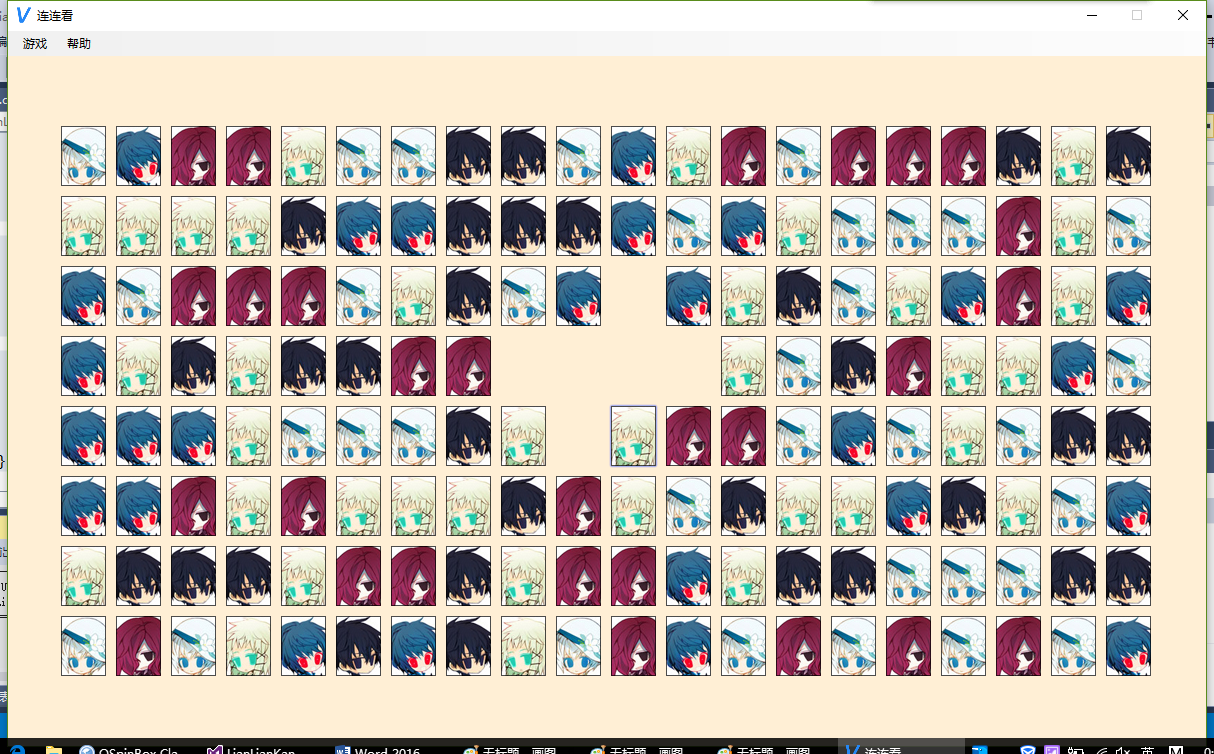
④以同样的方法进行纵向拓展，横向连线



①都在边缘 ②在同一行 ③不在同一行

**判断代码见附录代码里的bool canLink(int x1, int y1, int x2, int y2);**

## 运行效果



## 个人小结

与上课不同，开放性实验将我们从死的理论中拖出来，投入到实践中去。理论课上我们学的和练的只是按部就班的知识；而开放性实验中尽管并没有添加什么看得见摸得着的知识，但给了我们结构化的思想、设计的经验、探索的意识。

比如这一个连连看游戏，虽然是一个经典的游戏，但在自己思考设计的过程中，才体会用三条线连接两个方块不只是我们人眼看上去的那么简单，有多重的判断、临时变量的建立等等很多细节需要仔细去推敲。

除了细节上的问题，宏观上的设计也很重要。正确的将界面的交互和数据算法进行整合，对进一步的细节编写和调试、拓展有重要影响。这次的程序比较单一，如果程序变大以后，整体设计的影响将会更加大。

这一学期的课程比较仓促，做这个开放实验的时间也比较紧张，因此选择了连连看这样一个经典而又比较简单的游戏。将来我会更加努力学习C#的更深入的知识，同时注重结合实践，进一步提高自己的能力。

## 附录：全部代码

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace LianLianKan

{

public partial class Form1 : Form

{

static llk L = new llk();

class llk

{

private readonly string version = "v1.0.2";//标记类的版本号

public int[,] container = new int[20, 8];

Bitmap img1, img2, img3, img4, img5, decoration;

private int typeCount = 5;

private bool isGameStart = false;

private bool isGameComplete = false;

private bool isFocus = false;//是否被选中

private Point pointToDraw;//选中的点

private int lastm = -1;//上一个被选中的方格的坐标

private int lastn = -1;

static Random r = new Random();

public llk()

{

//加载图片

/\* 从文件夹中引用图片

\* img1 = new Bitmap("img/1.jpg");

\* img2 = new Bitmap("img/2.jpg");

\* img3 = new Bitmap("img/3.jpg");

\* img4 = new Bitmap("img/4.jpg");

\* img5 = new Bitmap("img/5.jpg");

\* decoration = new Bitmap("img/decoration.png");

\* \*/

img1 = Properties.Resources.\_1;

img2 = Properties.Resources.\_2;

img3 = Properties.Resources.\_3;

img4 = Properties.Resources.\_4;

img5 = Properties.Resources.\_5;

decoration = Properties.Resources.decoration;

}

public void reset()

{

//清空游戏数据

for (int i = 0; i <= 19; i++)

{

for (int j = 0; j <= 7; j++)

{

container[i, j] = 0;

}

}

//给二维数组随机赋值

for (int i = 1; i <= 160; i++)

{

int line\_now = (i - 1) / 20;

//(i / 20 != 8) ? ((i - 1) / 20) : 7

int row\_now = ((i % 20 - 1) != -1) ? (i % 20 - 1) : 19;

if (container[row\_now, line\_now] != 0 && i <= 160)

{

//若检测到该位置已被赋值,则跳过

}

else

{

//检测到该位置的值为0(初始值),则给找一个位置与该位置赋相同值

int type = r.Next(1, typeCount + 1);//type=1-typeCount,为该位置的方块类型

int m = r.Next(0, 20);

int n = r.Next(line\_now, 8);//随机一个位置(跳过已经被赋值的行)

while (container[m, n] != 0 || (m == row\_now && n == line\_now))

{

//若检测到该位置已被赋值则重新随机位置,直到随机到的位置未被赋值

m = r.Next(0, 20);

n = r.Next(line\_now, 8);//随机一个位置(跳过已经被赋值的行)

}

//将相同的值赋给两个位置

container[row\_now, line\_now] = type;

container[m, n] = type;

}

}

}

public void drawcanvas(Graphics canvas)

{

for (int m = 0; m <= 19; m++)

{

for (int n = 0; n <= 7; n++)

{

//判断该位置的图片类型绘制相对应的图片

switch (container[m, n])

{

case 0:

break;

case 1:

canvas.DrawImage(img1, 55 \* m + 55, 70 \* n + 70);

break;

case 2:

canvas.DrawImage(img2, 55 \* m + 55, 70 \* n + 70);

break;

case 3:

canvas.DrawImage(img3, 55 \* m + 55, 70 \* n + 70);

break;

case 4:

canvas.DrawImage(img4, 55 \* m + 55, 70 \* n + 70);

break;

case 5:

canvas.DrawImage(img5, 55 \* m + 55, 70 \* n + 70);

break;

}

}

}

}

public void drawdiv(Graphics canvas)

{

//选中方格时，边缘高亮

int Px = pointToDraw.X;

int Py = pointToDraw.Y;

int m = ((Px - 55 \* (Px / 55)) <= 45) ? Px / 55 - 1 : -1;

int n = ((Py - 70 \* (Py / 70)) <= 60) ? Py / 70 - 1 : -1;

if (m >= 0 && n >= 0 && m<= 19 && n<= 7)

{

if (canLink(lastm, lastn, m, n))

{

//上一个被选中的位置与本次选中的位置类型相同时,消除方块,将lastType重置为0,取消焦点

container[m, n] = 0;

container[lastm, lastn] = 0;

lastm = -1;

lastn = -1;

isFocus = false;

}

else if (container[m, n] != 0)

{

//上一个被选中的位置与本次选中的位置类型不同或不存在上一个被选中的位置时,(重新)设定上一个位置的信息,并显示选中当前位置

canvas.DrawImage(decoration, 55 \* m + 55 - 5, 70 \* n + 70 - 5);

lastm = m;

lastn = n;

}

else

{

lastm = -1;

lastn = -1;

}

}

}

public bool IsGameStart

{

get { return isGameStart; }

set { isGameStart = value; }

}

public bool IsGameComplete

{

get { return isGameComplete; }

set { isGameComplete = value; }

}

public bool IsFocus

{

get { return isFocus; }

set { isFocus = value; }

}

public Point PointToDraw

{

get { return pointToDraw; }

set { pointToDraw = value; }

}

private bool canLink(int x1, int y1, int x2, int y2)

{

if (x1 < 0) return false;

if (x1 > 19) return false;

if (x2 < 0) return false;

if (x2 > 19) return false;

if (y1 < 0) return false;

if (y1 > 7) return false;

if (y2 < 0) return false;

if (y2 > 7) return false;

if (x1 == x2 && y1 == y2) return false;

if (container[x1, y1] == container[x2, y2])

{

if (x1 == 0 && x2 == 0) return true;//当两个位置在同一边时返回真

if (x1 == 19 && x2 == 19) return true;

if (y1 == 0 && y2 == 0) return true;

if (y1 == 7 && y2 == 7) return true;

bool[,] link1 = new bool[20, 8];

//做横向匹配

link1[x1, y1] = true;

link1[x2, y2] = true;

int pos;

if (x1 != 19)

{

pos = x1 + 1;

while (container[pos, y1] == 0)

{

link1[pos, y1] = true;

if (pos == 19) break;//pos等于19的时候跳出循环

pos++;

}

}

if (x1 != 0)

{

pos = x1 - 1;

while (container[pos, y1] == 0)

{

link1[pos, y1] = true;

if (pos == 0) break;//pos等于0的时候跳出循环

pos--;

}

}

if (x2 != 19)

{

pos = x2 + 1;

while (container[pos, y2] == 0)

{

link1[pos, y2] = true;

if (pos == 19) break;//pos等于19的时候跳出循环

pos++;

}

}

if (x2 != 0)

{

pos = x2 - 1;

while (container[pos, y2] == 0)

{

link1[pos, y2] = true;

if (pos == 0) break;//pos等于0的时候跳出循环

pos--;

}

}

if (y1 == y2)

{

for (int i = Math.Min(x1, x2); i <= Math.Max(x1, x2); i++)

{

if (link1[i, y1]) break;

if (i == Math.Max(x1, x2)) return true;

}

}

else

{

for (int i = 0; i <= 19; i++)

{

if (link1[i, y1] == true && link1[i, y2] == true)

{

if (i == 0 || i == 19) return true;//同时延展到边上时返回能消除

for (int j = Math.Min(y1, y2); j <= Math.Max(y1, y2); j++)

{

if (container[i, j] != 0 && j != Math.Max(y1, y2) && j != Math.Min(y1, y2)) break;

if (j == Math.Max(y1, y2)) return true;

}

}

}

}

bool[,] link2 = new bool[20, 8];

//做纵向匹配

link2[x1, y1] = true;

link2[x2, y2] = true;

if (y1 != 7)

{

pos = y1 + 1;

while (container[x1, pos] == 0)

{

link2[x1, pos] = true;

if (pos == 7) break;//pos等于7的时候跳出循环

pos++;

}

}

if (y1 != 0)

{

pos = y1 - 1;

while (container[x1, pos] == 0)

{

link2[x1, pos] = true;

if (pos == 0) break;//pos等于0的时候跳出循环

pos--;

}

}

if (y2 != 7)

{

pos = y2 + 1;

while (container[x2, pos] == 0)

{

link2[x2, pos] = true;

if (pos == 7) break;//pos等于19的时候跳出循环

pos++;

}

}

if (y2 != 0)

{

pos = y2 - 1;

while (container[x2, pos] == 0)

{

link2[x2, pos] = true;

if (pos == 0) break;//pos等于0的时候跳出循环

pos--;

}

}

if (x1 == x2)

{

for (int i = Math.Min(y1, y2); i <= Math.Max(y1, y2); i++)

{

if (link1[x1, i]) break;

if (i == Math.Max(y1, y2)) return true;

}

}

else

{

for (int i = 0; i <= 7; i++)

{

if (link2[x1, i] == true && link2[x2, i] == true)

{

if (i == 0 || i == 7) return true;//同时延展到边上时返回能消除

for (int j = Math.Min(x1, x2); j <= Math.Max(x1, x2); j++)

{

if (container[j, i] != 0 && j != Math.Max(x1, x2) && j != Math.Min(x1, x2)) break;

if (j == Math.Max(x1, x2)) return true;

}

}

}

}

}

return false;

}

}

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

InitializeComponent();

}

private void canvas1\_Click(object sender, EventArgs e)

{

//左上角的坐标为(0,25)

Point formPoint = this.PointToClient(Control.MousePosition);

formPoint.Y -= 25;//修复获取到的坐标

L.IsFocus = true;//打开选中方格的开关

L.PointToDraw = formPoint;//输入点击的点

this.Refresh();//触发canvas1的重绘事件

}

private void canvas1\_Paint(object sender, PaintEventArgs e)

{

Graphics g = e.Graphics;

if (L.IsFocus)

{

L.drawdiv(g);

}

if (L.IsGameStart)

{

L.drawcanvas(g);

}

}

private void 开始ToolStripMenuItem\_Click(object sender, EventArgs e)

{

L.reset();//随机生成游戏

L.IsGameStart = true;//打开游戏开始开关

this.Refresh();//触发canvas1的重绘事件

}

private void 退出ToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Close();

}

private void 关于ToolStripMenuItem\_Click(object sender, EventArgs e)

{

MessageBox.Show("Copyright ©2015 LianLianKan Team. All Rights Reserved.\nPowered by 韦冈松，刘宗杰.", "关于");

}

}

}