学生学号	0121710880223	实验课成绩	

武汉理工大学

学 生 实 验 报 告 书

实验课程名称	数据结构与算法综合实验
开 课 学 院	计算机科学与技术学院
指导教师姓名	钟忺
学 生 姓 名	刘佳迎
学生专业班级	软件 1702

2018 -- 2019 学年 第 二 学期

实验课程名称： 数据结构与算法综合实验

实验项目名称	线性结构与连连看游戏综合实践			报告成绩	
实验者	刘佳迎	专业班级	软件 1702	组别	
同组者	无			完成日期	2019 年 6 月 8 日

第一部分：实验分析与设计（可加页）

一、 实验目的和要求

1.目的

(1) 调研连连看游戏，了解连连看游戏的功能和规则等。
(2) 掌握集成开发工具(Visual C++ 6.0 或 Microsoft Visual Studio 2010)。
(3) 掌握 C++的基础编程。
(4) 理解 MFC 框架，包括 MFC Dialog 应用程序和 GDI 编程。
(5) 理解线性结构，重点掌握数组和栈操作，掌握数组的遍历、消子和胜负判断等算法。
(6) 理解企业软件开发过程，理解系统需求分析和设计，应用迭代开发思路进行项目开发。
(7) 养成良好的编码习惯和培养软件工程化思维，综合应用“C++编程、MFC Dialog、算法、线性结构”等知识，开发“连连看游戏”桌面应用程序，达到掌握和应用线性结构核心知识。

2.要求

待开发连连看游戏，称为“卡通连连看”，使用二维数组来保存游戏地图中的数据，实现了连连看的核心功能。 卡通连连看游戏功能结构图如下：

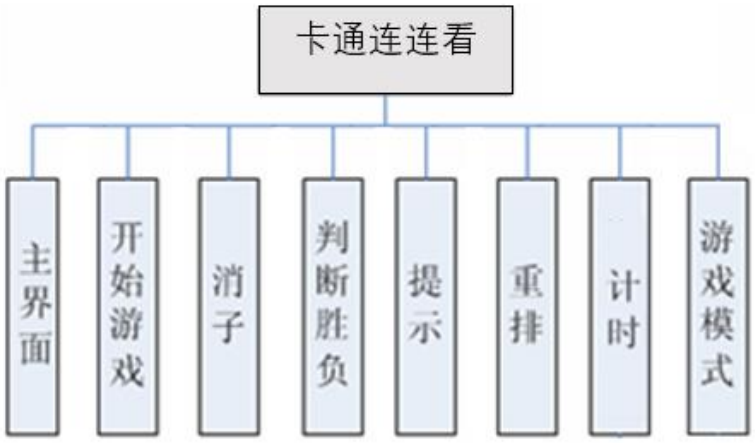


图 1-2 功能结构图

（1）主界面

为游戏主界面，进行各项操作入口。

（2）开始游戏

玩家选择游戏模式，进入游戏后，选择开始游戏，系统根据设置的主题风格生成一个图片布局(游戏地图)，以供玩家点击消除。游戏地图大小为 640*400，是一个 16 行 * 10 列矩形，分成 160 个小正方形，存放 160 张图片，每张图片大小为 40*40。

（3）消子

对玩家选中的两个图案进行判断，是否符合消除的规则。只有符合以下条件的图案对才会消失： 1）一条直线连通； 2）两条直线连通； 3）三条直线连通。如果可以消除，从游戏地图中提示连接路径，然后消除这两种图片，并计算相应的积分。如果不能消除，则保持原来的游戏地图。

（4）判断胜负

当游戏完成后，需要判断游戏的胜负。不同模式下，判断胜负的规则不同。

1）基本模式时，如果在 5 分钟内，将游戏地图中所有的图片都消除，则提示玩家获胜。

2）休闲模式时，如果游戏地图中所有的图片都被消除，则提示玩家获胜。

（5）提示

可以提示界面上能够消除的一对图片。

（6）重排

根据随机数，重新排列游戏地图上图片。

（7）定时

设定一定时间来辅助游戏是否结束。

（8）游戏模式

基本模式、休闲模式和关卡模式三种，可以根据是否定时等规则进行设置，增强趣味性。

二、 分析与设计

2.1 需求分析

2.1.1 项目简介

“连连看游戏”是给一堆图案中的相同的图案进行配对的简单游戏，在一定的规则之内对相同的图案进行消除处理，在规定时间内消除所有图案后玩家就获胜。

“连连看游戏”只要将相同的两张元素用三根以内的直线连在一起就可以消除，规则简单容易上手，游戏速度节奏快，画面清晰可爱。类似游戏界面如下图所示：

图 2.1-1 连连看游戏参考界面

2.1.2 游戏规则

(1) 一条直线消子

选择的两张图片花色相同，并且处于同一条水平线或者同一条垂直线上，并且两张图片之间没有其余的图片，则可以进行一条直线消子。

图 2.2-2 一条直线消子

(2) 两条直线消子

选择的两条图片花色相同，既不在同一水平线上，也不再同一垂直线上，两个图片的连通路径至少有两条直线组成，两条直线经过的路径必须是空白，中间只要有一个非同种类的图片，该路径无效。

图 2.1-3 两条直线消子

(3) 三条直线消子

使用一个折点的路径无法连通的两个图片，只能如图中连线所示连通，即连通路径有三条直线，在该直线的路径上没有图案出现，只能是空白区域。

图 2.1-4 三条直线消子

系统主要业务流程图如下：

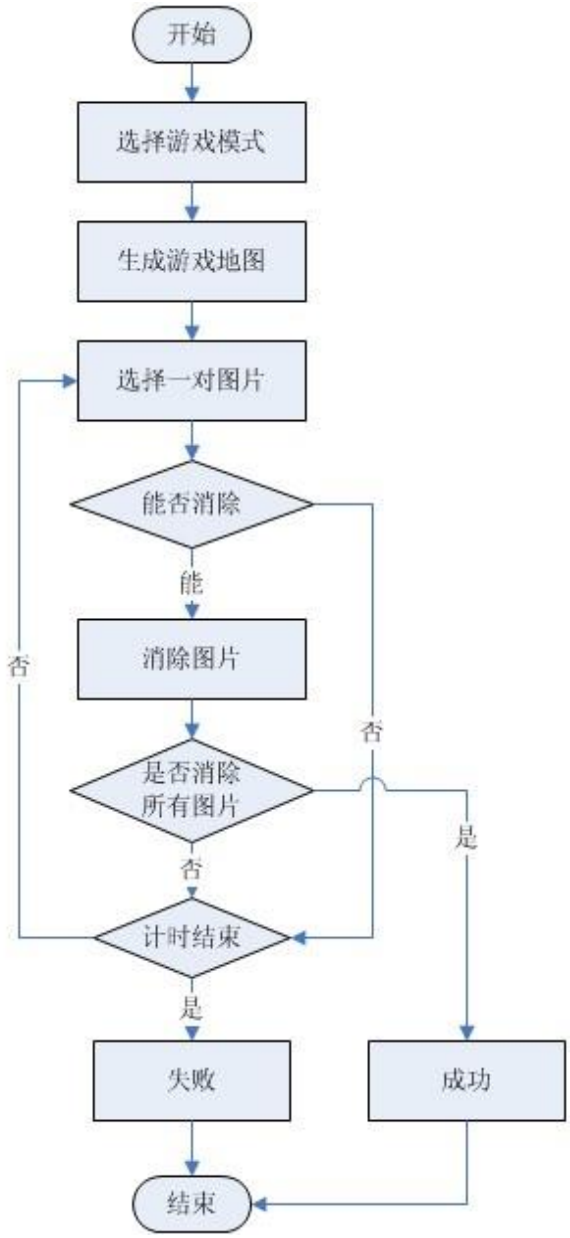


图 2.1-5 业务流程图

2.2 系统设计

2.2.1 界面设计

1、主界面

主界面为启动游戏时出现的界面，在该界面上进行游戏模式的选择、游戏的设置、查看帮助信息、关于“欢乐连连看”。主界面大小为 800*600。

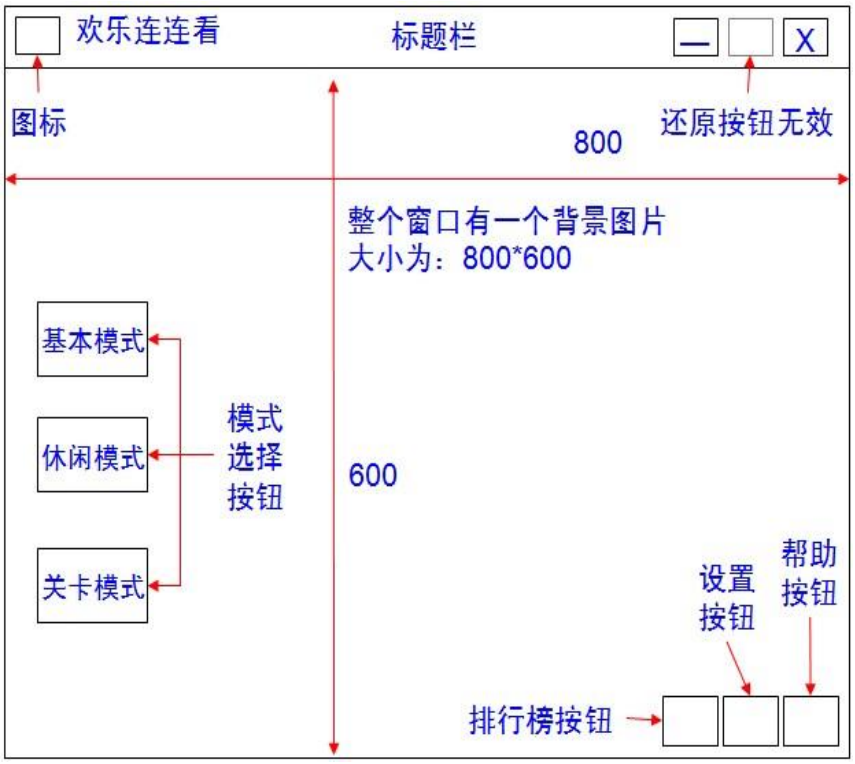


图 2.2-1 主界面设计

使用画图软件制作一张 800*600 大小的 BMP 图片，背景图片设计时需要考虑主界面上按钮位置的摆放。

图 2.2-2　主界面效果

2、游戏界面

根据设置的主题生成的游戏地图、开始新游戏按钮、暂停按钮、提示按钮、重排按钮、计时、设置按钮、帮助按钮。游戏地图像素大小为 640*400。每张图片像素大小为 40*40。

游戏地图是一个 16*10 的矩形。游戏地图有 160 张图片。图片出现的位置为随机的。

图 2.2-3 游戏界面

2.2.2 程序结构设计

1、工程结构设计

(1) 解决方案名称：Lianliankan

(2) 工程名称：LLK

(3) 工程目录结构

工程目录结构如下图所示：

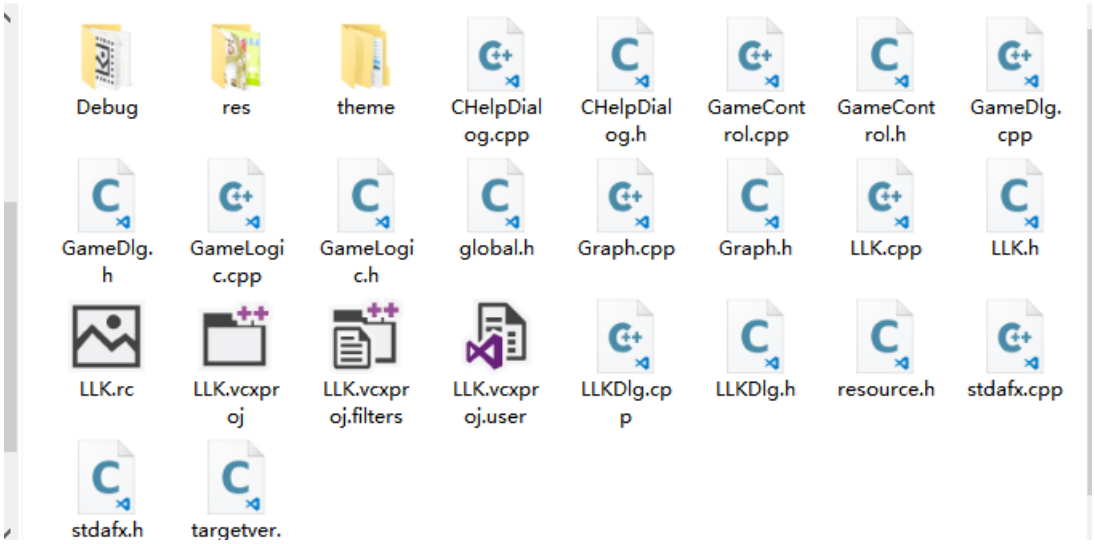


图 2.2-4 工程目录结构

1）res：程序资源文件夹，存放图标。

2）theme：主题文件夹，存放游戏的主题资源文件。

2、程序类关系设计

游戏程序按分层的思路来设计，主要分为：界面层（主窗口类 CLLKDlg、游戏窗口类 CGameDlg），游戏控制和业务逻辑层（游戏控制类 CGameControl、游戏逻辑操作类 CGameLogic）。

各层之间使用结构体（顶点信息 Vertex）来传递数据。

应用程序中公共的常量，定义在 global.h 头文件中。

2.2.3 数据结构设计

1、顶点存储结构

添加 global.h 文件，定义结构体 Vertex，用于保存游戏地图中一个点的行号、列表、值信息。

代码如下：

```
 /*结构体，保存游戏地图中每一个图片元素的行号、列号和值信息*/
typedef struct tagVertex {
   int row;        // 行号
   int col;        // 列号
   int info;       // 值信息
}Vertex;
```

2、游戏地图存储结构

使用二维数组来保存连连看游戏地图，给每种图片一个编号，并将这些编号保存在二维数组中。

用户在屏幕上选择 2 张图片，对应为数组中的两组坐标。分别实现三个消子判断算法：“一条直线消子”、“两条直线消子”、“三条直线消子”，并使用这三个算法进行消子判断。

若符合消子规则，就在屏幕上消除一对图片，并把数组对应元素清空。

(1) 游戏地图中的图片种类和重复次数与游戏的级别和难度有关。图片种类越多，重复次数越小，游戏的难度越大，反之则越容易。

(2) 因为 2 张同类的图片才能消。为保证游戏能完全消完，每种图片重复的次数一定要是偶数，即 2 的倍数。

(3) 地图的大小与图片元素种类之间的关系

地图的行数 * 地图的列数 = 图片的种类数 * 每种图片重复的次数

(4) 地图数据的存储

1) 用 int 类型动态二维数组(int** m_pGameMap)存储地图中元素图片的编号。

2) 获得某行某列对应的元素数值

2.2.4 核心算法设计

1、随机开局算法

(1) 计算游戏中元素个数：行数 * 列数。

(2) 计算每一种花色重复数：行数 * 列数 / 花色数。

1) 判断（行数 * 列数 % 花色数）是否为 0。如果不为 0，则进行异常处理。

2) 判断每一种花色重复数是否能被 2 整除，如果不能被 2 整除，则进行异常处理。

(3) 按从左到右，从上到下的顺序，将花色数填入游戏地图。

0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3
3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4
4	4	5	5	5	5	5	5	5	5	5	5	6	6	6	6
6	6	6	6	6	6	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	9	9	9	9	9	9
9	9	9	9	10	10	10	10	10	10	10	10	10	10	11	11
11	11	11	11	11	11	11	11	12	12	12	12	12	12	12	12
12	12	13	13	13	13	13	13	13	13	13	13	14	14	14	14
14	14	14	14	14	14	15	15	15	15	15	15	15	15	15	15

实现代码如下：

```
/* 初始化游戏地图 */
int ** CGameLogic::InitMap()
{
    // 获取地图大小和花色
    int nRows = CGameControl::s_nRows;
    int nCols = CGameControl::s_nCols;
    int nPicNum = CGameControl::s_nPicNum;

    // 游戏地图开辟内存空间
    int** pGameMap = new int*[nRows];
    if(NULL == pGameMap)
    {
```

```
        throw new CGameException(_T("内存操作异常！"));
    }
    else
    {
        for (int i = 0; i < nRows; i++)
        {
            pGameMap[i] = new int[nCols];
            if(NULL == pGameMap)
            {
                throw new CGameException(_T("内存操作异常！"));
            }
            memset(pGameMap[i], NULL, sizeof(int) * nCols);
        }
    }

    // 多少花色，根据花色的种类计算出每种花色的图片的平均个数，依次给数组赋值。
    if ((nRows * nCols) % (nPicNum * 2) != 0)
    {
        ReleaseMap(pGameMap);
        throw new CGameException(_T("游戏花色与游戏地图大小不匹配！"));
    }
    int nRepeatNum = nRows * nCols / nPicNum;
    int nCount = 0;
    for(int i = 0; i < nPicNum; i++)
    {
        for(int j = 0; j < nRepeatNum; j++)  // 重复数
        {
            pGameMap[nCount / nCols][nCount % nCols] = i;
```

```
            nCount++;
        }
    }

    /* 随机找到两个位置的图片，进行交换 */

    srand((int)time(NULL));   // 设置种子

    // 随机任意交换两个数字
    int nVertexNum = nRows * nCols;
    for(int i = 0; i < nVertexNum; i++)
    {
        // 随机得到两个坐标
        int nIndex1 = rand() % nVertexNum;
        int nIndex2 = rand() % nVertexNum;

        // 交换两个数值
        int nTmp = pGameMap[nIndex1 / nCols][nIndex1 % nCols];
        pGameMap[nIndex1 / nCols][nIndex1 % nCols] = pGameMap[nIndex2 / nCols][nIndex2 %
nCols];
        pGameMap[nIndex2 / nCols][nIndex2 % nCols] = nTmp;
    }

    return pGameMap;
}
```

(4) 由于生成的地图是规则的，因此，需要将地图中的花色打乱。实现思路是，随机选择两个元素，将其值对调。重复若干次（当前游戏重复了元素总数次）。

2、消子判断的流程

(1) 获得选中的两张图片的行号与列号。
(2) 判断选中的图片是否同色，不同色，则不能相消。判断选中的图片是否为同一个图片，如果为同一个图片，不能相消。
(3) 判断连通性，如以下三种情况均不满足，则结束。
    1) 首先判断能否一条直线连通。
    2) 如果不能一条直线连通，则判断能否两条直线连通。
    3) 如果不能两条直线连通，则判断能否三条直线连通。
(4) 获得连通路径，绘制连通线。
(5) 消除图片。
(6) 更新游戏地图。

3、一条直线消子算法

(1) 判断两个顶点，行是否相同，若相同，则判断两个顶点在 X 方向是否连通。

在 CGameLogic 类中定义 LinkInRow()函数实现 X 方向连通判断。依次判断在 X 方向两个顶点间每一个顶点，是否都为空，全为空，表示可以连通，否则不能连通。

实现伪代码如下：

```
bool CGameLogic::LinkInRow(int** pGameMap, Vertex v1,
Vertex v2) {
    1：将顶点的列进行调整，使 nCol1 的值小于 nCol2
    2：使用 for 循环，判断两个顶点间是否不为空的图片
 for(int i = nCol1 + 1; i <= nCol2; i++)
    {
        if(i == nCol2)      return true;
    if(pGameMap[nRow][i] != BLANK)  break;  // BLANK 表示空
    }
  return false;
}
```

(2) 判断两个顶点，列是否相同，若相同，则判断两个顶点在 Y 方向是否连接。

在 CGameLogic 类中定义 LinkInCol()函数实现 Y 方向连通判断。依次判断在 Y 方向两个顶点间每一个顶点，是否都为空，全为空，表示可以连通，否则不能连通。

4、两条直线消子算法

若一条直线无法连通，则判断二条直线的情况。在 CGameLogic 类中定义 OneCornerLink()函数判断两点是否能两条直线连通。

先判断两个顶点的 X 和 Y 方向的直线相交的两个顶点，是否为空。若能构成两条直线连通，那么这个相交的顶点必须为空才行。

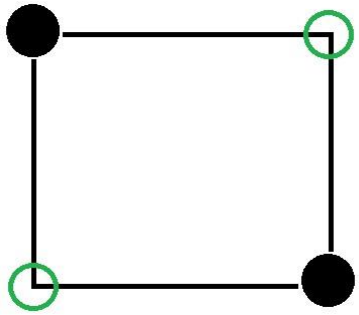


图 2.2-5 两条直线消子算法示意图

若顶点有一个可以相消，则判断该顶点与两个顶点，横向或纵向一条直线是否连通。若都连通，则表示两条直线消子成功。否则不能相消。

实现伪代码如下：

```
  bool CGameLogic::OneCornerLink(int** pGameMap, Vertex v1, Vertex v2)
{
  1：判断相交的顶点是否为空
  if (pGameMap[v1.row][v2.col] == BLANK)
    {
        2：判断两个同行的顶点是否一条直线连通
        3：判断两个同列的顶点是否一条直线连通          if(纵向连通判
断(pGameMap, v1.row, v2.row, v2.col)
        && 横向连通判断(pGameMap, v1.row, v1.col, v2.col))
        {
          return true;
        }
    }

  if(pGameMap[v2.row][v1.col] == BLANK)
    {
     if(纵向连通判断(pGameMap, v1.row, v2.row, v1.col)
          && 横向连通判断(pGameMap, v2.row, v1.col, v2.col))
        {
```

```
            return true;
        }
    }
  return false;
}
```

5、三条直线消子算法

若二条直线无法连通，则判断三条直线的情况。在 CGameLogic 类中定义 TwoCornerLink()函数判断两点是否能三条直线连通。

三条直线消子时，假设选择的两个图片的位置为(nRow1, nCol1)和(nRow2, nCol2)，则先寻找与 Y 轴平行的连通线段。

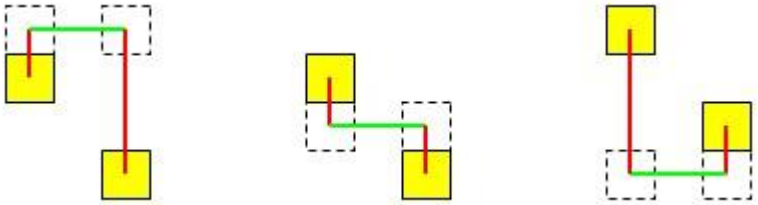


图 2.2-6 寻找与 Y 轴平行的连通线段

如果 Y 轴没有找到可以连通的三条直线，则寻找与 X 轴平行的连通线段。

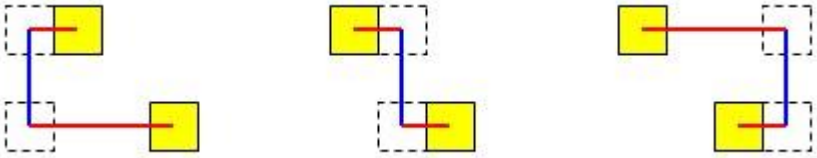


图 2.2-7 寻找与 X 轴平行的连通线段

(1) 搜索关键路径

如何找到这样的一条关键路径呢？以搜索水平方向上的关键路径为例。

假设玩家选择的两个顶点为 v0(row0, col0)，v3(row3, col3)。

图 2.2-8 搜索关键路径

步骤如下：

第一步，从地图的第一行开始扫描，当前扫描到 nRow 行；

第二步，设置拐点：V1(nRow, col0)，V2(nRow, col3)；

第三步，判断顶点 V1 和 V2 是否为空；

第四步，判断顶点 V1 和 V2 是否水平方向上连通，如果连通，则 V1 到 V2 之间的连线即为关键路径。如果不连通则接着扫描下一行，重复②③④的步骤。

(2) 判断三条直线连通

采用枚举法判断三条直线连通，假设玩家选择的两个顶点为 V0 和 V3，判断三条直线连通具体实现步骤如下：

1) 找到其中一条关键路径 V1，V2；
2) 判断 V1 和 V0 是否连通；
3) 判断 V2 和 V3 是否连通；
4) 如果同时满足 V1 和 V0 连通，V2 和 V3 连通，则 V0 和 V3 满足三条直线连通；否则，在此关键路径下 V0 和 V3 不连通，找到下一条关键路径，重复步骤②③④，直到判断出 V0 和 V3 是否连通。

(3) 保存连通路径

使用栈来保存连通路径中的关键点：起始点 V0、拐点 V1，拐点 V2 和终点 V3。

保存连通路径的步骤如下：

1) 保存起始点 V0；
2) 判断是否存在能够满足三条直线消子的关键路径 V1，V2；
3) 如果存在，保存顶点 V1，V2，V3；如果不存在，删除起始点 V0。

6、胜负判断算法

一种方案是：当有元素被消掉后，进行胜负判断，遍历地图中所有元素的值，当所有的元素都为空时，表示获胜，游戏结束，否则继续游戏。但是，这种方案每次消除后都要遍历一次二维数组，效率很低，时间复杂度为 $O((m*n)^2)$会使游戏不流畅，因此采用了一种时间复杂度仅为 O(m*n)的算法。就是在每次消除图片后技术其加 2，并且和游戏图片元素总数比较，若相等则获胜，否则游戏继续。

7、提示算法

在 CGameDlg 类中添加提示按钮的响应事件 CGameDlg::OnBnClickedBtnPrompt()，调用 CGameControl 类的 Help () 函数，查找可消子的图片对。若有一对元素可连通，则在界面对应元素区绘制矩形框进行提示。1 秒后，重绘界面，矩形框消失。 在 CGameDlg 类中添加 DrawTipLine()函数，绘制提示框。 在 CGameControl 类中添加 Help()函数，先判断游戏是否为空，不为空，则调用 CGameLogic 类中的 SearchValidPath()函数来查找可消子的图片对。 在 CGameLogic 类中添加 SearchValidPath()函数，搜寻有效可消子的路径，算法实现的 过程，从左到右，从上到下，依次判断地图中同色元素是否可以连通。

8、重排算法

在 CGameDlg 类中添加重排按钮的响应事件 CGameDlg:: OnBnClickedBtnRerank()，调用 CGameControl 类的 Rerank() 函数进行重提成，调用 UpdateMap()函数更新界面。 在 CGameControl 类中添加 Rerank()函数，调用 CGameLogic 类中的 RerankGraph()函数来对地图数据重排。 在 CGameLogic 类中添加 RerankGraph()函数，随机任选地图中两个顶点，将元素进行交换，这样进行 100 次。

3.核心算法实现

（1） 更新游戏地图

```
void CGameDlg::UpdateMap()
{
    UpdateGameRect();   //重绘矩形游戏区域

    // 获取地图行数、列数和图片数
    int nRows = CGameControl::s_nRows;
    int nCols = CGameControl::s_nCols;
    int nPicNum = CGameControl::s_nPicNum;

    // 计算图片的顶点坐标与图片大小
    int nLeft = m_ptGameTop.x, nTop = m_ptGameTop.y;      //游戏区起始顶点坐标
    int nElemW = m_sizeElem.cx, nElemH = m_sizeElem.cy;   //图片高度和宽度像素

    CClientDC dcGame(this);
```

```
    for (int i = 0; i < nRows; i++) {
        for (int j = 0; j < nCols; j++) {
            // 得到图片编号的值
            int nElemVal = m_GameC.GetElement(i, j);
            if (nElemVal != -1)    //如果已被置为BLANK(-1)，则跳过
            {
                // 将背景与掩码相或，边保留，图像区域为 1
                //m_dcMem.BitBlt(nLeft + j * nElemW, nTop + i * nElemH, nElemW, nElemH,
&m_dcMask, 0, nElemVal * nElemH, SRCPAINT);
                dcGame.StretchBlt(nLeft + j * nElemW, nTop + i * nElemH, nElemW, nElemH,
&m_dcMask, 0, nElemVal * nElemH, nElemW, nElemH, SRCPAINT);

                // 与元素图片相与，边保留，图像区域为元素图片
                //m_dcMem.BitBlt(nLeft + j * nElemW, nTop + i * nElemH , nElemW, nElemH,
&m_dcElement, 0, nElemVal * nElemH, SRCAND);
                dcGame.StretchBlt(nLeft + j * nElemW, nTop + i * nElemH, nElemW, nElemH,
&m_dcElement, 0, nElemVal * nElemH, nElemW, nElemH, SRCAND);
            }
        }
    }
}
```

（2）鼠标左击事件响应

```
void CGameDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
    if (!m_bPlaying)    //如果游戏未在进行，不响应鼠标事件
        return;
    // 判断鼠标点击的区域
    if(point.y < m_rtGameRect.top || point.y > m_rtGameRect.bottom || point.x <
m_rtGameRect.left || point.x > m_rtGameRect.right)
    {
        return CDialogEx::OnLButtonUp(nFlags, point);
    }
    int nRow = (int)(point.y - m_rtGameRect.top) / m_sizeElem.cy;    //点击位置所在行号
    int nCol = (int)(point.x - m_rtGameRect.left) / m_sizeElem.cx;   //点击位置所在列号

    if (m_GameC.GetElement(nRow, nCol) != BLANK)   // 若未被消除则选中
    {
        if (m_bFirstPoint)  // 第一个点
        {
            DrawTipFrame(nRow, nCol);     //绘制第一个矩形提示框
            m_GameC.SetFirstPoint(nRow, nCol);   //设置选中的第一个点的记录
            m_bFirstPoint = false;
        }
        else               // 第二个点
        {
            DrawTipFrame(nRow, nCol);     //绘制第二个矩形提示框
            m_GameC.SetSecPoint(nRow, nCol);   //设置选中的第二个点的记录

            // 连子判断
            Vertex avPath[PathLen];
            int nVexnum = 0;
            bool bSuc = m_GameC.Link(avPath, nVexnum);
            if (bSuc == true)
            {
                DrawTipLine(avPath, nVexnum);    // 画提示线
                Sleep(200);     //暂停200ms，避免一闪而过
                //PlaySound(_T("res\\music\\cut2.wav"), NULL, SND_FILENAME | SND_ASYNC );   //
播放消除声音
                if (m_GameC.IsWin())
                {
                    CButton *pBtn = (CButton*)GetDlgItem(IDC_BTN_START);   //按钮指针指向
IDC_BTN_START这个按钮
                        pBtn->EnableWindow(TRUE);    // "开始游戏"按钮设为可用
                        MessageBox(_T("You are the winner!!!"), _T("Successful"), MB_OK);   //
弹窗提醒玩家胜利!
                }
            }
            UpdateMap();      // 更新地图
            m_bFirstPoint = !m_bFirstPoint;   //重置
        }
    }
```

```
}
```

（3）进度条定时器事件处理

```
void CGameDlg::OnTimer(UINT_PTR nIDEvent)
{
    int m_nStep = 1;
    int nPrePos = pProgCtrl->StepIt();      //取得更新前位置,即当前剩余时间
    pProgCtrl->SetPos(nPrePos - m_nStep);    //更新进度位置
    CString timeStr;
    timeStr.Format(_T("剩余时间:%ds"), nPrePos);
    pProgCtrl->SetWindowTextW(timeStr);
    pStaticTime->SetWindowTextW(timeStr);

    CDialogEx::OnTimer(nIDEvent);
}
```

（4）消子判断

```
bool CGameControl::Link(Vertex *avPath, int &nVexnum)
{
    // 判断是否同一张图片
    if(m_svSelFst.row == m_svSelSec.row && m_svSelFst.col == m_svSelSec.col)
    {
        return false;
    }
    // 判断图片是否相同
    if(m_pGameMap[m_svSelFst.row][m_svSelFst.col]!=m_pGameMap[m_svSelSec.row][m_svSelSec.col])
    {
        return false;
    }

    // 判断是否连通
    if (m_GameLogic.IsLink(m_pGameMap, m_svSelFst, m_svSelSec))
    {
        // 消子实现
        m_GameLogic.Clear(m_pGameMap, m_svSelFst, m_svSelSec);
        clearPic += 2;       //以此消除2张图片，计数器加2
        // 返回路径顶点数
        nVexnum = m_GameLogic.GetVexPath(avPath);
        return true;
    }
    return false;
}
```

（5）初始化游戏地图

```
int** CGameLogic::InitMap()
{
    // 获取地图大小和花色
    int nRows = CGameControl::s_nRows;
    int nCols = CGameControl::s_nCols;
    int nPicNum = CGameControl::s_nPicNum;
```

```
    // 游戏地图开辟内存空间
    int** pGameMap = new int*[nRows];
    if(NULL == pGameMap)
    {
        throw new CGameException(_T("内存操作异常！"));
    }
    else
    {
        for (int i = 0; i < nRows; i++)
        {
            pGameMap[i] = new int[nCols];
            if(NULL == pGameMap)
            {
                throw new CGameException(_T("内存操作异常！"));
            }
            memset(pGameMap[i], NULL, sizeof(int) * nCols);
        }
    }

    // 多少花色，根据花色的种类计算出每种花色的图片的平均个数，依次给数组赋值。
    if ((nRows * nCols) % (nPicNum * 2) != 0)
    {
        ReleaseMap(pGameMap);
        throw new CGameException(_T("游戏花色与游戏地图大小不匹配！"));
    }
    int nRepeatNum = nRows * nCols / nPicNum;
    int nCount = 0;
    for(int i = 0; i < nPicNum; i++)
    {
        for(int j = 0; j < nRepeatNum; j++)  // 重复数
        {
            pGameMap[nCount / nCols][nCount % nCols] = i;
            nCount++;
        }
    }

    /* 随机找到两个位置的图片，进行交换 */

    srand((int)time(NULL));   // 设置种子

    // 随机任意交换两个数字
    int nVertexNum = nRows * nCols;
    for(int i = 0; i < nVertexNum; i++)
    {
        // 随机得到两个坐标
        int nIndex1 = rand() % nVertexNum;
        int nIndex2 = rand() % nVertexNum;

        // 交换两个数值
        int nTmp = pGameMap[nIndex1 / nCols][nIndex1 % nCols];
        pGameMap[nIndex1 / nCols][nIndex1 % nCols] = pGameMap[nIndex2 / nCols][nIndex2 %
```

```
nCols];
        pGameMap[nIndex2 / nCols][nIndex2 % nCols] = nTmp;
    }

    return pGameMap;
}
```

（6）2个拐点，3条线消子判断

```
bool CGameLogic::TwoCornerLink(int** pGameMap, Vertex v1, Vertex v2)
{
    for (int nCol = 0; nCol < CGameControl::s_nCols; nCol++)
    {
        // 找到一条与 Y 轴平行的连通线段
        if (pGameMap[v1.row][nCol] == BLANK && pGameMap[v2.row][nCol] == BLANK)
        {
            if (LineY(pGameMap, v1.row, v2.row, nCol))
            {
                if (LineX(pGameMap, v1.row, v1.col, nCol) && LineX(pGameMap, v2.row, v2.col,
nCol))
                {
                    // 保存节点
                    Vertex vx1 = { v1.row, nCol, BLANK };
                    Vertex vx2 = { v2.row, nCol, BLANK };
                    PushVertex(vx1);
                    PushVertex(vx2);
                    return true;
                }
            }
        }
    }
    for (int nRow = 0; nRow < CGameControl::s_nRows; nRow++)
    {
        // 找到一条与 X 轴平行的连通线段
        if(pGameMap[nRow][v1.col] == BLANK && pGameMap[nRow][v2.col] == BLANK)
        {
            if(LineX(pGameMap, nRow, v1.col, v2.col))
            {
                if(LineY(pGameMap, nRow, v1.row, v1.col) && LineY(pGameMap, nRow, v2.row,
v2.col))
                {
                    // 保存节点
                    Vertex vx1 = { nRow, v1.col, BLANK };
                    Vertex vx2 = { nRow, v2.col, BLANK };
                    PushVertex(vx1);
                    PushVertex(vx2);
                    return true;
                }
            }
        }
    }
    return false;
```

}

4.测试用例设计

4.1 黑盒测试

(1) 测试游戏主窗体各个功能按钮的实现和窗体的最小化、关闭、移动、关于等菜单项。

(2) 进入“基本模式”，开始游戏，首先进行简单的相邻图片的消除测试，再进行边沿图片的消除测试，寻找一拐点 2 直线、2 拐点 3 直线的情况进行消除测试；2 拐点 3 直线有两种情况：“U 型”和“Z 型”，即 2 直线在另一直线同侧或异侧。

(3) 观察进度条的工作情况，包括剩余时间提示、进度条移动等，待进度条超时后测试游戏结束的情况。

(4) 进行一次完整的游戏，在预定时间内取胜，测试游戏成功后的弹窗提示和新一轮游戏的初始化工作完成情况，紧接着下一轮的游戏测试。

(5) 声音和显示测试：进入游戏时背景音乐正常播放，图片消失时，有背景音乐播放；游戏退出时背景音乐停止播放。

4.2 白盒测试

（1）构造合适的输出结构，检查游戏地图数组的初始化情况。

（2）检查消子判断的各个函数的算法逻辑正确性，从 Link()到 isLink()，再到 LinkInRow()、LinkInCol()、OneCornerLink()、TwoCornerLink()逐层检查其逻辑。

（3）测试各个窗体的初始化函数和图像绘制函数。

（4）测试消子路径生成函数。

三、主要仪器设备及耗材

1. 安装了 Windows 7 或 Windows 10 或其它版本 Windows 操作系统的 PC 机 1 台

2. PC 机系统上安装了 Microsoft Visual Studio 2010 以上开发环境

3. 4 张游戏背景图 800*600、20 张游戏元素图 40*40

4.画图、Photoshop CC2017

第二部分：实验过程和结果（可加页）

一、 实现说明

1、创建解决方案

(1) 选择“开始 -> 程序 -> Miscrosoft Visual Studio 2015 -> Miscrosoft Visual Studio 2015 ”，打开 VS2015。

(2) 在 VS2015 开发工具中选“File -> New -> Project”菜单,出现新建对话框。

(3) 在新建对话框中，选择解决方案类型为“Other Project Type -> Visual Studio Soluctions -> Blank Soluction”,解决方案名为“Lianliankan”，保存路径。

(4) 创建完成后，解决方案保存路径中，生成解决方案文件夹，在解决方案文件夹中，生成解决方案文件(.sln 后缀)。

2、创建工程

(1) 创建解决方案之后，选择“File -> New -> Project”，显示新建对话框。

(2) 选择工程类型为“Visual C++ -> MFC -> MFC Application”，输入工程名称 LLK，选择“Soluction” 为 “Add to solution”，点击“OK”，进入应用程序向导。

(3) 选择应用程序类型 在应用程序向导的“Application Type”中，选择应用程序类型为“Dialog based”。然后点击 “Next”进入下一步。

(4) 在“User Interface Features”中，勾选“Minisize box”，给对话框窗口添加一个最小化按钮。 然后点击“Finish”完成工程的创建。

3、修改主界面对话框属性

(1) 打开主界面对话框资源，方法一：选择主界面对话框类 CLLKDlg，右键选择“Go To Dialog”,打开主界面对话框资源。方法二：在资源视图，双击主界面对话栏资源。打开对话框资源。

(2) 修改对话框标题为“卡通连连看”。 1) 在对话对话编辑器中，删除对话框资源中默认产生的控件。 2) 在对话框资源上右键，选择“Properties”，打开对话框属性编辑器。 3) 在对话框属性编辑器中修改对话框标题栏为“欢乐连连看”。 4) 编译并运行程序。

(3) 修改对话框图标。 1) 在工程目录 res 文件夹中，找到对话框图标“LLK.ico”。将需要设置为对话框图片的 ico 文 件命名为“LLK.ico”，替换工程目录 res 中默认的 LLK.ico 文件。 2) 编译并运行程序。 由于修改了资源文件，必须要先把原来编译的文件清除后，全部重新编译才行。否则 VS 中默认是增量编译的，已编译的内容不会重新编译。只替换了图标的文件，工具并不会重新编译图标。

4、进行项目的具体设计实现

(1) 主界面设计：背景载入，按钮布局和按钮事件响应。

(2) 开始游戏：游戏窗体的布局，背景载入，游戏图片加载。

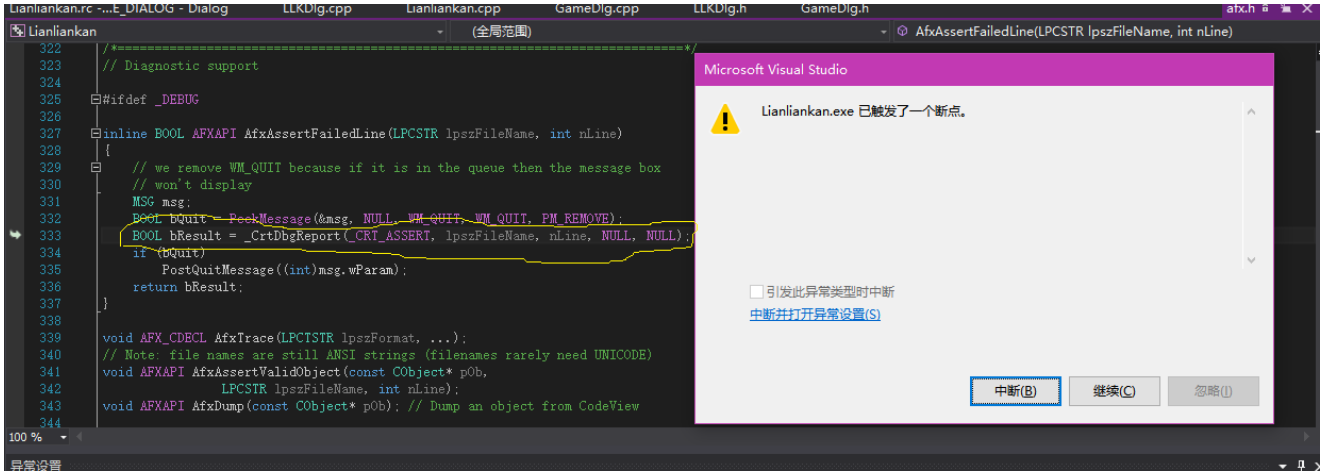
(3) 游戏控制和逻辑类的设计实现，逻辑的链接。

(4) 游戏图片的消除判断。

(5) 游戏胜负的判断，以及游戏结束后的下一轮准备工作。

二、 调试说明（调试手段、过程及结果分析）

1. 关于 MFC 对话框最小化后从任务栏还原就出发中断的问题



程序最小化后不能还原，一般原因是程序中至少存在一个 Popup 类型的窗口引起的，因为 Popup 类型的子窗口即使由于父窗口的隐藏而隐藏，其 WS_VISIBLE 属性仍然是可见的，当用户再次点击任务栏的程序图标时，Popup 窗口会拦截系统（还原）消息，使主程序框架无法接收到系统消息，从而导致主程序无法正常还原。如果将其修改为 Child 类型的窗口，那么主程序的最小化和还原的功能就可以正常了。不过在实际项目中，往往就需要一个 Popup 类型的窗口作为子窗口（Popup 类型的窗口也可以有父窗口），那么这又如何解决程序最小化后不能还原的问题呢？根据以上分析的原理，只要在主程序最小化时，相应也隐藏掉 Popup 窗口（ShowWindow(SW_HIDE)），这样系统消息就能够正确传递了；当主程序还原时，再将隐藏的 Popup 窗口显示出来，这样就既不影响程序的显示效果，又能解决问题了！具体方法如下：

首先需要在主程序（如 MainFrame）中拦截系统消息（响应最大化，最小化，还原，关闭等消息的地方）。其消息为 WM_SYSCOMMAND.如在 MainFrame.h 头文件中加入 afx_msg void OnSyscommand(UIND nID,LPARAM lParam);在 MainFrame.cpp 的 BEGIN_MAP 与 END_MAP 之间加入 ON_WM_SYSCOMMAND，响应函数为
void CMainFrame::OnSyscommand(UIND nID,LPARAM lParam){}。

其次根据系统消息对 Popup 窗口进行隐藏与显示操作，代码如下：

```
CWnd* m_pPopupWnd;/// Popup 类型的窗口指针
void CMainFrame::OnSyscommand(UIND nID,LPARAM lParam)
{
        static BOOL s_bDialogVisible = FALSE;
        /// 如果是最小化消息
        if(SC_MINIMIZE == nID)
        {
              if(NULL != m_pPopupWnd && ::IsWindow(m_pPopupWnd->m_hWnd))
              {
```

```
                    if(::IsWindowVisible(m_pPopupWnd->m_hWnd))
                    {
                          s_bDialogVisible = TRUE;
                          /// 隐藏 Popup 类型窗口
                          m_pPopupWnd->ShowWindow(SW_HIDE);
                    }
              }
        }
        else
        {
              if(NULL != m_pPopupWnd && ::IsWindow(m_pPopupWnd->m_hWnd))
              {
                    if(TRUE == s_bDialogVisible)
                    {
                          s_bDialogVisible = FALSE;
                          /// 显示 Popup 类型窗口
                          m_pPopupWnd->ShowWindow(SW_SHOW);
                    }
              }
        }
        CWnd::OnSyscommand(nID,lParam);
}
```

方法二：拦截系统的还原消息，对其进行自定义的操作，如先设置为活动窗口，然后继续执行还原操作。

```
      BOOL PreTranslateMessage(MSG* pMsg)
      {
                ASSERT(pMsg);
                  /// 如果是激活窗口消息
                if(pMsg->message == WS_APPACTIVE)
                    {
                          /// 如果是按下左键
                          if(pMsg->wParam == VK_LBUTTON)
                          {
                                ASSERT(AfxGetMainFrame());
                                /// 激活主窗口
                                SetActiveWindow(AfxGetMainFrame()->m_hWnd);
                          }
                    }
              /// 可继续向基类传递消息
              return     C**APP::PreTranslateMessage(pMsg);
      }
```

***问题所在：**

问题出在背景图加载函数 InitBackground()上，将位图资源加载进 dc 内存后直接绘制图像，导致最小化窗口还原后无法重绘，要想正常重绘，必须将绘制图像的函数放进

OnPaint()函数；包括 CGameDlg 控制的游戏窗体类同理，加载游戏背景和游戏地图元素的逻辑，都是要先将位图加载进相应的 CDC 位图内存，然后执行各自的绘制或重绘，并且要保证游戏地图元素在游戏进行状态还原窗口后也能实现重绘，这就需要在 onPaint()中加入一个判断语句：

```
if (m_bPlaying)     //如果游戏处于开始状态，则需要重绘游戏地图，主要是为了窗口最小化还原后可以自动重绘元素
    UpdateMap();     //如果是刚进入游戏界面，还没有点击“开始游戏”，则不需要加载游戏地图
```

2. 游戏图片元素的组织问题

一开始没能理解老师给的操作步骤原理，看网上别人写的连连看都是给图片编号，根据游戏地图数组存储的随机编号调用显示相关图片。经过分析，原来课件中的思想是，把所有游戏图片元素组合在一起，相当于一个一维图片组，根据游戏地图中的图片编号确定图片在元素图片组和掩码图片组中的位置，然后利用 BitBit()函数将其提取并做位运算处理，之后显示在游戏地图对应位置。

3. 游戏相关的 C++类的组合问题：CGameDlg、CGameControl、CGameLogic、CGameException

CGameDlg 类负责游戏界面的交互和消息事件响应，其中包含 CGameControl 类的对象，用于实现游戏的控制，包括初始化游戏地图、设置选中点的信息、消子判断等；CGameControl 类中创建了 CGameLogic 类的对象，用于实现游戏的逻辑控制，包括随机生成游戏地图、游戏图片的连通判断、连通路径的记录和消子等，逻辑性强、算法最复杂。CGameException 类贯穿各个类，用于处理游戏中的一些异常事件。

4. 游戏胜负判断的算法优化

老师给的游戏胜负判断的参考算法是在 m_GameLogic 对象中用 IsBank(int** pGameMap)来遍历游戏地图二维数组，判断其中的元素是否全部置为空，然后在 m_GameC 类中用 IsWin()调用 IsBank(pGameMap)函数，判断胜负。在每次选中两张图片并判断可以消除后，都要调用一次 IsWin()来判断胜负，也就是要每次都遍历一次 10*16 的二维数组，效率非常低。因此，可以在 CGameControl 类中定义一个 int 变量 clearPic 用于记录消除的图片数，每次消除后 clearPic 自加 2，与图片总数比较，若相等则说明所有图片消除完毕，玩家胜利，这样就避免了遍历二维数组带来的时空效率的浪费。

```
/*根据消除的图片数判定胜负*/
bool CGameControl::IsWin(void)
{
    /*如果消除的图片数与原有图片数相等，则判定玩家取胜；优化了每次遍历二维地图数组带来的时空复杂度*/
    if (clearPic == s_nRows*s_nCols)
    {
        clearPic = 0;   //重置计数器，为下一轮做准备
```

```
            return true;
        }
        else
            return false;
    }
```

5. 游戏地图元素数据利用两个随机数重排使游戏进程阻塞的问题

CGameLogic 类中的 RerankGraph()函数中使用了如下代码随机生成两个坐标：

```
        do {
            // 随机得到第一个坐标
            int nIndex1 = rand() % nVertexNum;
            x1 = nIndex1 / nCols; y1 = nIndex1 % nCols;
        } while (pGameMap[x1][y1] != BLANK);   //直到第一个元素数据不为空

        do {
            // 随机得到第二个坐标
            int nIndex2 = rand() % nVertexNum;
            x2 = nIndex2 / nCols; y2 = nIndex2 % nCols;
        } while (pGameMap[x2][y2] != BLANK);    //直到第二个元素数据不为空
```

出现的问题是每当点击“重排”按钮后都会很长时间无响应，不会真正实现重排，而且导致了游戏其他进程阻塞。开始我以为是两个随机数生成的筛选条件可能有点苛刻，导致长时间无法生成 2 个符合要求的随机数才导致运行变慢。然而开始游戏后，直接点击“重排”，进度条本来正在加载阶段，之后直接停在中间不动了。于是，我把其中一个坐标只用一次随机数生成，另一个任然用 while 循环生成地图数据不为空的坐标，游戏开始后 10s，进度条加载完毕，正常计时，这时点击“重排”，可以实现重排。如果游戏一开始就重排，仍然会导致阻塞。进一步测试，如果点击“重排”后，值生成 2 个简单随机数，不进行复杂的筛选，仍然会在进度条加载阶段卡死。所以问你题在于进度条的加载导致阻塞，而 2 个严格的随机数筛选也是游戏运行变慢、有效地图坐标命中率低下无法实现重排的因素。那么就要解决进度条加载的问题。

6. 边缘图片的消除和内部图片区域外引线消除情况的统一解决方案

游戏地图数组外层加一“圈”，并初始化为 BLANK（-1），需要同步修改的其余部分有：加载游戏图片是需要从编号为 1 的行列开始，遍历数组寻找通路时扩展到 0 至 nRows+1 和 0 至 nCols+1；其余部分可直接复用。

```
    // 游戏地图开辟内存空间
    int** pGameMap = new int*[nRows + 2];
    if(NULL == pGameMap)
    {
        throw new CGameException(_T("内存操作异常！"));
    }
    else
    {
```

```
        for (int i = 0; i < nRows + 2; i++)
        {
            pGameMap[i] = new int[nCols + 2];
            if(NULL == pGameMap)
            {
                throw new CGameException(_T("内存操作异常！"));
            }
            memset(pGameMap[i], BLANK, sizeof(int) * (nCols + 2));  //初始化数组为
            BLANK(-1)
        }
    }
```

三、 软件测试（测试效果.界面、综合分析和结论）

1. 测试效果界面

通过黑盒测试，一步步运行游戏得到如下测试截图：图 1 展示了游戏主界面图；图 2 为游戏基本模式-开始游戏；图 3 为图片消除实现；图 4 为进度条及游戏将获胜；图 5 为退出游戏弹窗确认；图 6 为边缘可消除提示线；图 7：帮助对话框；图 8、9：关卡模式和休闲模式。

图 1：游戏主界面图

图 2：游戏基本模式-开始游戏

图 3：图片消除实现

图 4:进度条及游戏将获胜

图 5：帮助菜单

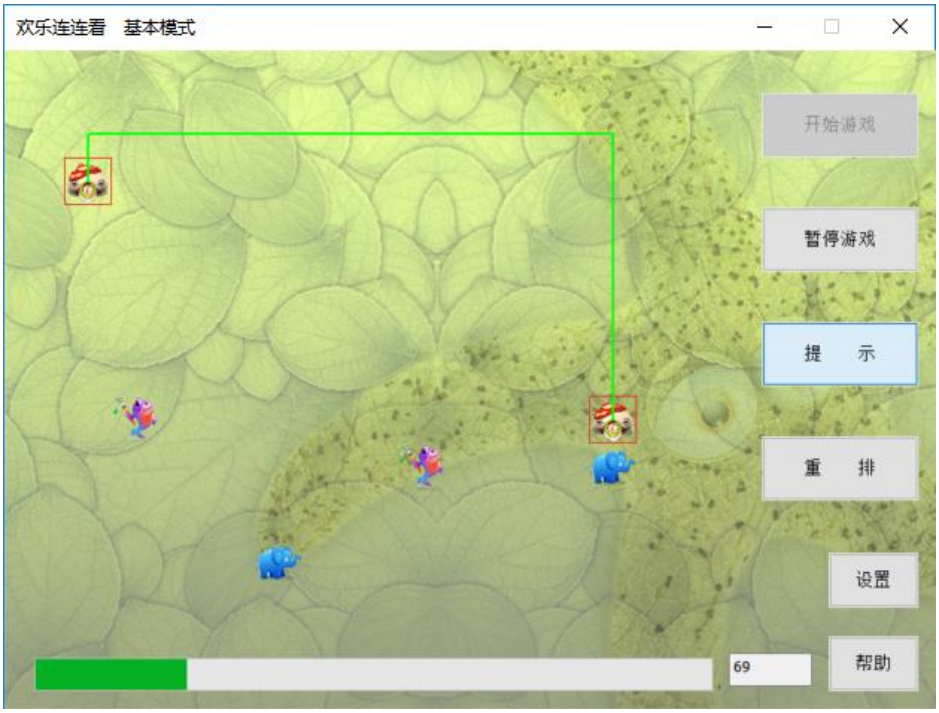


图 6：提示线

2. 综合分析和结论

（1）根据玩家在使用连连看时会涉及到的相关功能，我在整个项目中用几个类分版块的实现。首先设计出对话框，若有按钮则在相应的类中根据其 ID 添加与之相应的响应函数，因为主对话框需要绘制棋盘，则还应添加于绘图相关的响应函数。

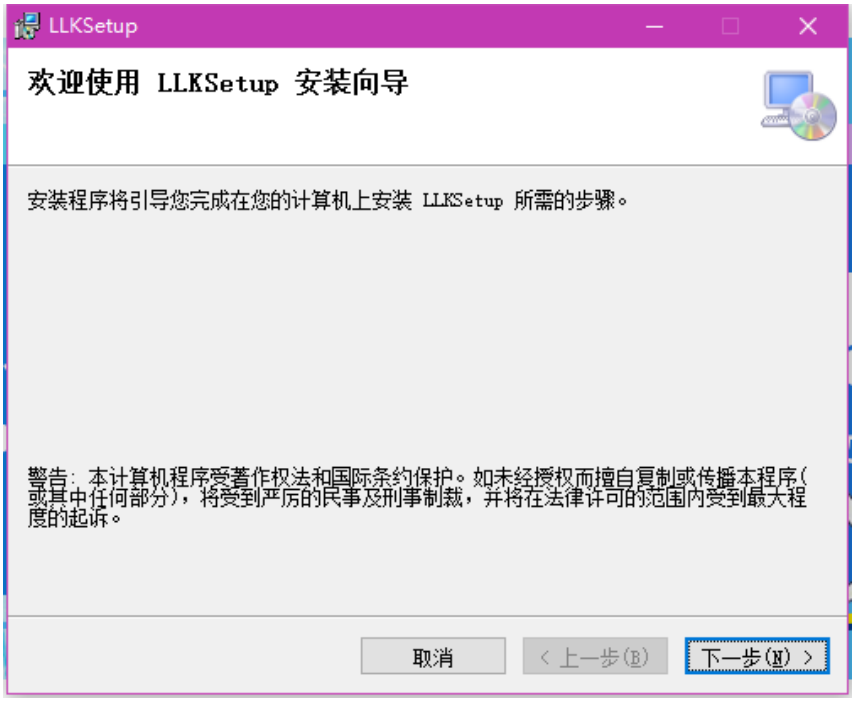
对于一些游戏功能的实现还要额外的添加成员函数，根据实际体验连连看游戏，了解到，一般的游戏都配有相应的音乐，以及英雄榜等。同时，为了增加难度，一般的游戏都会设置时间限制，为了满足这些功能，我查阅了相关资料后，根据游戏过程流程图设计出了这些功能代码。

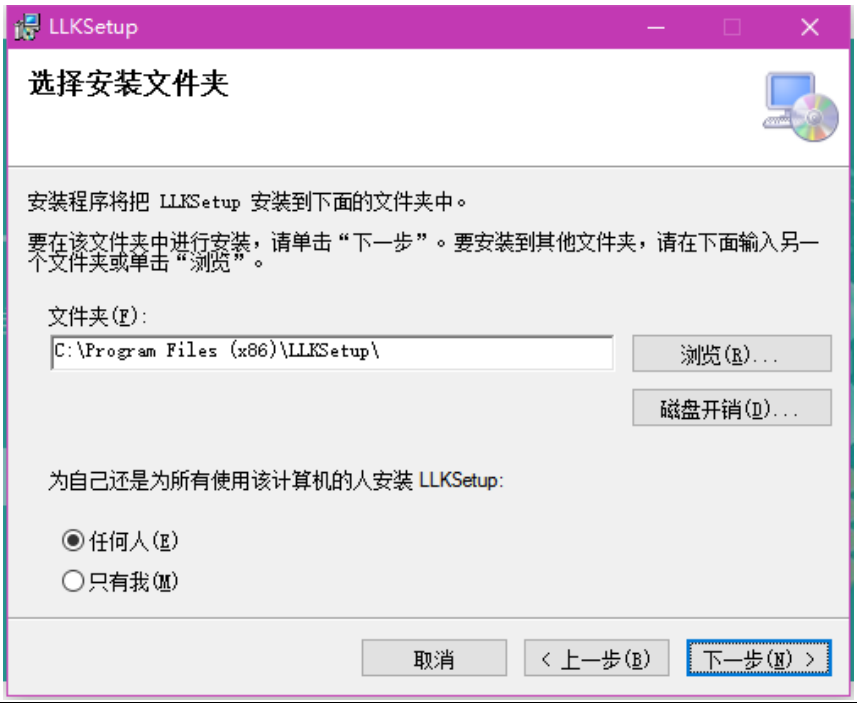
（2）本次连连看的消子算法设计主要在参考算法的基础上做了改进，比如对于边沿图片的消除，参考算法没有考虑到，对于边沿同一行货同一列的图片可以直接通过在游戏区域外引直线使其连通实现消除。如果只要达到这个目的，只需要在判断直通的函数中加一个边沿图片的判断就可以了：

```
//如果两元素处在边缘列，则可以直接消除，无需判断是否可以直通
    if (nCol == 0 || nCol == CGameControl::s_nCols-1)
        return true;
```

然而，还有另一种情况，如果边沿内部的两个图片可以通过在游戏区域外引 3 条直线 2 个拐点连通的话也是应该可以消除的，但是在游戏区域内可能是无法连通的。综合考虑这两种情况，最好的解决方案就是在游戏地图数组外再加一圈 BLANK 元素，专门用于边沿图片和内部图片外部连通的情况下进行连线和消除。

（4）在将 Lianliankan.exe 程序直接拷贝到别的计算机上运行时，除了缺少 mfc140ud.dll 和 vcruntime140d.dll 动态链接库的问题外，程序动态加载的背景资源图目录也要放在同一目录下才能正常加载，这样就比较麻烦，用户体验也不好。在网上查阅资料的得知建一个 Setup 安装项目生成一个安装包可以解决这些问题。于是自己新建了一个 LLKSetup 项目，将 LLK 项目调试生成的文件打包成安装包，并且将资源打包进去，经测试可以在所有 x86 的 Windows 系统上运行。安装过成如图所示：





第三部分：实验小结、收获与体会

游戏设计与实践是一项复杂而庞大的工作，在仔细思考了连连看游戏的需求分析和具体设计，我才意识到过程的艰难，因为以前从来都没有接触过游戏设计，更是对 MFC 望而生畏，刚开始有点迷茫和彷徨。以前自学过 MFC，当时就感觉很难，慢慢就放弃了，但是实验还要做，连连看又那么有吸引力，值得我去好好研究。后来通过翻阅书籍和在网上查阅资料，逐渐找到了一些感觉。

本次设计让我初步懂得了电子游戏涉及到的有关技术、方法，包括电子游戏选题、构思、设计步骤等。并实现一些可演示的游戏软件，其中有很多应用了学习的相关技术，并且做到了界面、声音都能实际演示。此次设计过程中印象最深的收获有：1、学到了很多新知识，并且对 C++知识进行了回顾。经过长时间的学习，更进一步熟悉了 MFC 编程、通过不断上机实验，调试程序，总结经验，从对课题的不理解到能够开始动手去做，提出新问题并自己想办法去解决问题，自己多实践，所以增强了动手能力。2、提高了中、英文资料的检索能力。这次专业设计过程中我查阅了多资料，包括一些期刊、杂志，还有网络中的电子文档、电子书籍、网页及下载的视频教学课程，尤其是大神们的博客；不但有中文资料还有英文资料。这些资料，使我的眼界更开阔，对课题的认识更加深刻，编写程序的时候思路更加清楚，少走了很多弯路。

回顾此次设计过程，我学到了许多书本上没有学到的知识。通过这次自己制作的软件，丰富了自己的实践技能，扩张了本专业的知识面，使我受益匪浅，同时也体验到了搞软件开发的难度。在这次设计的同时，由于我对这样的软件开发还只是一个开始，了解的不多，这其中或许还有很多的不足，有些模块做得不是很好，有些功能还不能够完全的实现，如窗体过度移动会出发中断，这个问题从一开始一直困扰到最后也没有解决，因为 MFC 类库中封装的东西太多，有些函数它底层的具体实现可能还没有真正的理解。