任务 6 操作步骤

1.1 消子判断

1.1.1 功能需求

1、介绍

实现游戏消子功能，玩家选中两个图片，判断是否符合消子规则，如果符合，则消掉这两个图片。否则保持不变。

图错误!文档中没有指定样式的文字。**-1 消子规则**

2、输入

选择的两个图片在游戏地图中的位置坐标、图案编号。

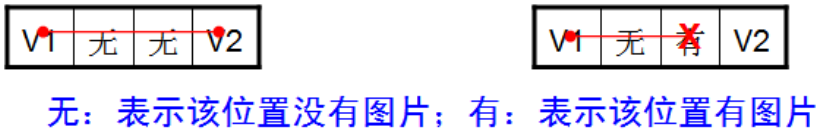
3、处理

(1) 若选中的两张图片满足下面条件，则进行连通判断。

1) 选中 2 张图片。

2) 选中的 2 张图片元素完全相同。

(2) 除选中的两个顶点外，连通线只能经过游戏地图上没有图片的空白的点。



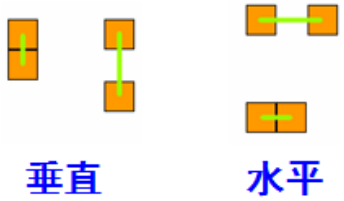
图错误!文档中没有指定样式的文字。**-2 连通判断**

(3) 如果有以下三类连通线中满足一种，则可进行消子。

1) 一条直线连通

情况一：两图片紧密相邻，中间既没有空白也没有其它种类的图案。

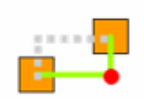
情况二：两图片并非紧密相邻，中间没有其它图案，但是有一个或者多个空白。



图错误!文档中没有指定样式的文字。**-3 一条直线连通**

2) 两条直线连通

两图片既不在同一水平线上，也不在同一垂直线上，两个图片的连通路径有两条直线组成，即有一个折点(图中红色圆点)，两条直线经过的路径必须是空白，中间只要有一个有其它图片，该路径无效，两图片无法连通。

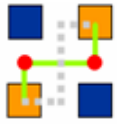


如图中的绿色实线和灰色虚线是两条直线连通的路径。

图错误!文档中没有指定样式的文字。**-4 两条直线连通**

3) 三条直线连通

连通路径有三条直线，两个折点组成(图中 2 个红色圆点)，在该直线的路径上没有图片出现，只能是空白区域。



如图中的绿色实线和灰色虚线是三条直线连通的路径。

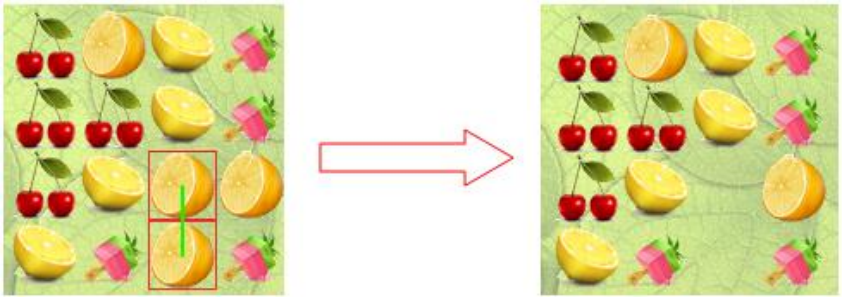
图错误!文档中没有指定样式的文字。**-5 三条直线连通**

(4) 提示框

单击图片会使得图片转变成被选中状态，给图片加一个红色的边框提示玩家。

(5) 消子

图片满足消失条件，显示连通路径(绿色线条)，然后被选中的图片立即消失，位置为空白。



图错误!文档中没有指定样式的文字。**-6 消子**

4、输出

最新的游戏地图。

1.1.2 设计思路

在“开始游戏”的基础上进行迭代开发。

1、消子流程

(1) 获得选中的两张图片的行号与列号。

(2) 判断图片的有效性。

(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。

2、消子算法与连接路径

(1) 消子算法

利用枚举法，分别枚举出三种连通方式的算法：

1) 一条直线消子算法。
2) 两条直线消子算法。
3) 三条直线消子算法。

(2) 连接路径

连接路径为两张图片连接线上的各个点，用栈结构进行存储。

3、类设计

(1) CGameLogic 类 —— 逻辑判断类。

1) 数据成员

数据成员	描述
Vertex m_avPath[4]	保存在进行连接判断时所经过的顶点
int m_nVexNum	顶点数

2) 公有成员函数

成员函数	描述
bool IsLink(int** pGameMap, Vertex v1, Vertex v2)	判断是否连通
void Clear(int** pGameMap, Vertex v1, Vertex v2)	消子
int GetVexPath(Vertex avPath[4])	得到路径，返回的是顶点数

3) 保护类型成员函数

成员函数	描述
bool LinkInRow(int** pGameMap, Vertex v1, Vertex v2)	判断横向是否连通
bool LinkInCol(int** pGameMap, Vertex v1, Vertex v2)	判断纵向是否连通
bool OneCornerLink(int** pGameMap, Vertex v1, Vertex v2)	一个拐点连通判断

bool LineY(int** pGameMap, int nRow1, int nRow2, int nCol)	直线连通 Y 轴
bool LineX(int** pGameMap, int nRow, int nCol1, int nCol2)	直线连通 X 轴
void PushVertex(Vertex v)	添加一个路径顶点
void PopVertex()	取出一个顶点
void ClearStack()	清除栈
bool TwoCornerLink(int** pGameMap, Vertex v1, Vertex v2)	三条直线消子判断

(2) CGameControl 类 —— 游戏控制类。

1) 数据成员

数据成员	描述
Vertex m_svSelFst	选中的第一个点
Vertex m_svSelSec	选中的第二个点

2) 成员函数

成员函数	描述
void SetFirstPoint(int nRow, int nCol)	设置第一个点
void SetSecPoint(int nRow, int nCol)	设置第二个点
bool Link(Vertex avPath[4], int &nVexnum)	消子判断(路径暂定为 2 个顶点，后面再对该函数进行修订)

1.1.3 编码实现

导入“开始游戏”的解决方案，在此基础上进行迭代开发，实现步骤如下：

步骤一：添加鼠标事件。

步骤二：选择图片。

步骤三：消除相同元素图片。

步骤四：一条直线消子。

步骤五：两条直线消子。

步骤六：三条直线消子。

1、添加鼠标事件

(1) 鼠标点击事件

1) 用户使用鼠标点击窗口时，当点击的位置在当前应用程序窗口所在区域，操作系统会发消息通知当前的应用程序，并且会把鼠标点击的坐标点和事件类型传入。

2) WM_LBUTTONUP 消息

当鼠标左键按钮被释放是响应。而 WM_LBUTTONDOWN 消息是在按下时响应。这个两个消息都可以代表用户的点击事件。

3) WM_LBUTTONUP 消息响应函数

函数原型：void OnLButtonUp(UINT nFlags, CPoint point);

函数参数：

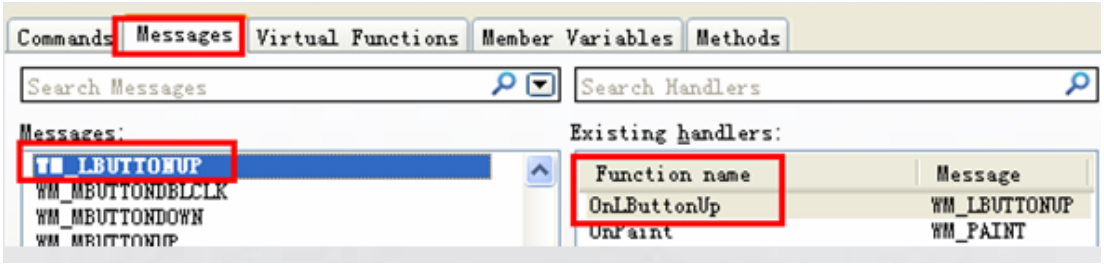
nFlags：标识在鼠标点击的同时是否有功能键，如：Ctrl、Shift 键。

point：鼠标点击的坐标点，以当前窗口客户区为坐标系。

(2) 添加鼠标点击事件

1) 选择 CGameDlg 类，打开类向导(ClassWizard)。

2) 给 CGameDlg 类添加 WM_LBUTTONUP 消息响应函数 OnLButtonUp()。



图错误!文档中没有指定样式的文字。**-7 添加响应事件**

鼠标输入消息如下：

消息	消息触发器
WM_LBUTTONDOWN	鼠标停留在窗口的客户区上，同时鼠标左键按钮被按下
WM_LBUTTONUP	鼠标左键按钮被释放
WM_RBUTTONDOWN	鼠标停留在窗口的客户区上，同时鼠标右键按钮被按下
WM_RBUTTONUP	鼠标右键按钮被释放
WM_MBUTTONDOWN	鼠标停留在窗口的客户区上，同时鼠标的中间按钮被按下
WM_MBUTTONUP	鼠标中间按钮被释放
WM_MOUSEMOVE	鼠标在窗口的客户区上移动

2、选择图片

(1) 判断点击位置是否在游戏地图中

游戏矩形区域为 m_rtGameRect，点击位置的坐标必须在游戏矩形之内。

鼠标消息响应函数 OnLButtonUp()函数的参数 CPoint point 保存了鼠标点击位置的坐标。CPoint.x 表示横坐标，CPoint.y 表示纵坐标。

在 CGameDlg::OnLButtonUp()函数中判断点击位置是否在游戏地图中。

```
void CGameDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
    // 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);
    }
}
```

(2) 计算鼠标点击位置的行号和列号

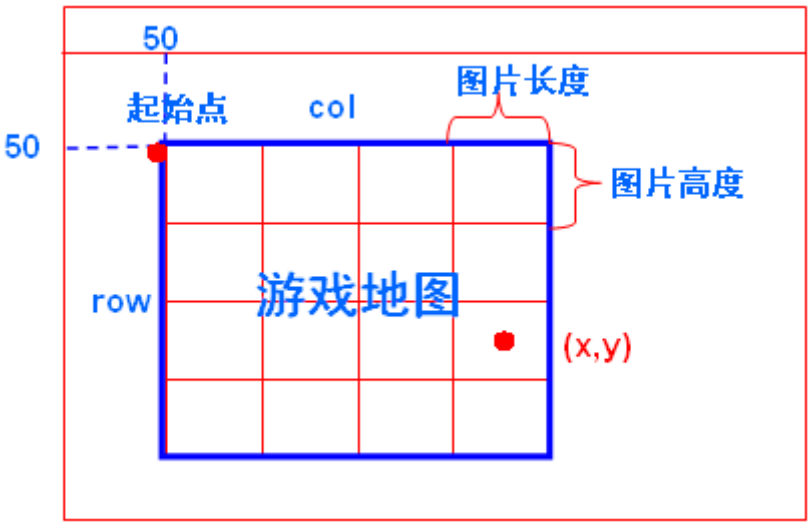
游戏地图在游戏界面对话框中的起始坐标为(50, 50)。假设点击位置的坐标为(x, y)。

1) 首先排除坐标(x, y)不在游戏区域的情况。

2) 根据(x, y)计算点击位置的行号和列号。

将 x>50 的区域按照图片的长度，分为 N 列，将 y>50 的区域按照图片的高度，分为 N 行。

点击位置的行号 row 和列号 col 分别为：row = (y - 50)/图片高度 和 col = (x - 50)/图片长度。



图错误!文档中没有指定样式的文字。**-8 计算图片的行和列**

根据点击位置的坐标、游戏地图中每个图片的大小，计算鼠标点击位置的行号和列号。

行号 = (点击位置 y 坐标 - 游戏地图的起始 y 坐标点)/元素的高度。

列号 = (点击位置 x 坐标 - 游戏地图的起始 x 坐标点)/元素的长度。

(3) 在鼠标选中的图中周围绘制矩形提示框。

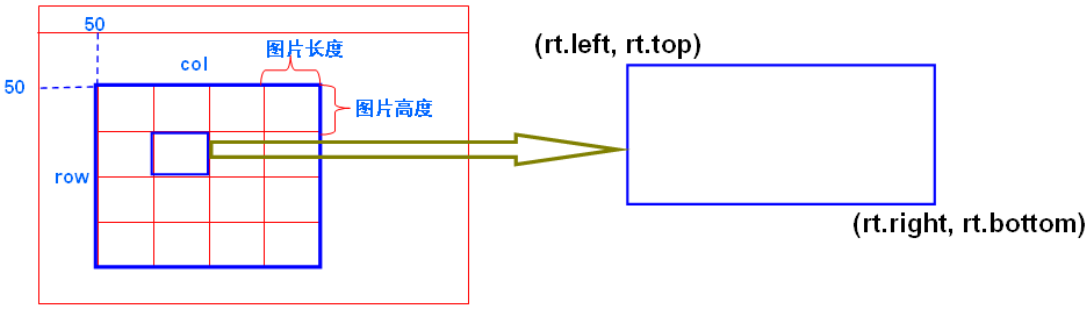
在 CGameDlg 类中添加 DrawTipFrame()函数，绘制提示框。

当鼠标选中游戏地图中的一张图片时，在该图片的四周绘制矩形提示框。矩形框的颜色为 RGB(233, 43, 43)，大小和选中的图片大小一致。

1) 在 CGameDlg 类中，添加成员函数 DrawTipFrame()，根据选择的图片的行号和列号，在选择的图片周围绘制矩形提示框。

函数定义格式为：void DrawTipFrame(int nRow, int nCol)

2) 绘制矩形时，可以根据矩形的左上角坐标和右下角坐标进行绘制。



图错误!文档中没有指定样式的文字。**-9 计算矩形框的坐标**

3、一条直线消子

(1) 添加 IsLink 函数进行连通判断

```
bool CGameLogic::IsLink(int** pGameMap, Vertex v1, Vertex v2)
{
	PushVertex(v1);

	// X 直连方式
	if(v1.row == v2.row)
	{
		//……
	}
	//Y 直连方式
	if(v1.col == v2.col)
	{
		//…..
	}
	return false;
}
```

(2) 行号相同时，判断横向是否连通。

1) 添加 CGameLogic::LinkInRow()函数

```
bool CGameLogic::LinkInRow(int** pGameMap, Vertex v1, Vertex v2)
{
	int nCol1 = v1.col;
	int nCol2 = v2.col;
	int nRow = v1.row;
	//保证 nCol1 的值小于 nCol2
	if(nCol1 > nCol2)
	{
		//数据交换
		int nTemp = nCol1;
		nCol1 = nCol2;
		nCol2 = nTemp;
	}

	//直通
	for(int i = nCol1 + 1; i <= nCol2; i++)
	{
		if(i == nCol2)		return true;
		if(pGameMap[nRow][i] != BLANK)	break;
	}
	return false;
}
```

2) 算法：从 nCol1 + 1 遍历到 nCol2，判断这条直线上是否都为空白区域。如果全为空白区域，则横向连通。

(3) 列号相同时，判断能否纵向连通。

1) 添加 CGameLogic::LinkInCol()函数

```
bool CGameLogic::LinkInCol(int** pGameMap, Vertex v1, Vertex v2)
{
	int nRow1 = v1.row;
	int nRow2 = v2.row;
	int nCol = v1.col;
	if(nRow1 > nRow2)
	{
		int nTemp = nRow1;
		nRow1 = nRow2;
		nRow2 = nTemp;
	}
	//直通
	for(int i = nRow1+1; i <= nRow2; i++)
	{
		if(i == nRow2)		return true;
		if(pGameMap[i][nCol] != BLANK) break;
	}

	return false;
}
```

2) 算法：从 nRow1 + 1 遍历到 nRow2，判断这条直线上是否全为空白区域。如果全为空白区域，则纵向连通。

4、两条直线消子

如果满足下列 2 个条件中的一个，则可以两条直线消子：

条件 1：(nRow1, nCol1)到(nRow1, nCol2)，(nRow1, nCol2)到(nRow2, nCol2)可以连通，并且(nRow1, nCol2)位置的图片为空。

条件 2：(nRow1, nCol1)到(nRow2, nCol1)，(nRow2, nCol1)到(nRow2, nCol2)可以连通，并且(nRow2, nCol1)位置的图片为空。

(1) 判断横向、纵向的线段是否能够连通

1) 判断(nRow1, nCol1)到(nRow1, nCol2)能否连通。

在 CGameDlg::LineX()函数中，判断(nRow1, nCol1)到(nRow1, nCol2)能否连通。

LineX()定义格式为：bool LineX(int** pGameMap, int nRow, int nCol1, int nCol2)

2) 判断(nRow1, nCol2)到(nRow2, nCol2)能否连通

在 CGameDlg::LineY()函数中，判断(nRow1, nCol2)到(nRow2, nCol2)能否连通。

LineY()定义格式为：bool LineY(int** pGameMap, int nRow1, int nRow2, int nCol)

(2) 判断(nRow1, nCol1)到(nRow2, nCol2)能否连通。

在 CGameLogic 类中定义数组 Vertex m_avPath[4]，使用栈来保存连通路径中的关键点：起始点 V0、拐点 V1 和终点 V2。

在 CGameLogic::OneCornerLink()函数中，判断(nRow1, nCol1)到(nRow2, nCol2)能否连通。

OneCornerLink()定义格式为：bool OneCornerLink(int anMap[][4], Vertex v1, Vertex v2)

```
bool CGameLogic::OneCornerLink(int** pGameMap, Vertex v1, Vertex v2)
{
    // 直角能够消子，那么顶点一定在与两个点的行和列相交的点，只有这两个点为空，才有可能实现二条直线消子

    if (pGameMap[v1.row][v2.col] == BLANK)
    {
        if(LineY(pGameMap, v1.row, v2.row, v2.col) && LineX(pGameMap, v1.row, v1.col, v2.col))
        {
            Vertex v = {v1.row, v2.col, BLANK};
            PushVertex(v);
            return true;
        }
    }

    if(pGameMap[v2.row][v1.col] == BLANK)
    {
        if(LineY(pGameMap, v1.row, v2.row, v1.col) && LineX(pGameMap, v2.row, v1.col, v2.col))
        {
            Vertex v = {v2.row, v1.col, BLANK};
            PushVertex(v);
            return true;
        }

    }

    return false;
}
```

(3) 在 CGameLogic::IsLink()中调用 CGameDlg::OneCornerLink()，判断能否两条直线消子。

5、三条直线消子

在 CGameLogic::TwoCornerLink()函数中，判断是否能进行三条直线消子。

```
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;
```

```
}
```

6、消除相同元素图片

(1) 判断选择的图片是否是同一种图片。

1) 如何判断连续两次选的图片，是否是要判断消除的一对图片？

要判断连续两次选择的图片是否是一组图片，需要设置一个标志位，CGameDlg 类的属性 bool m_bFirstPoint 用于标识是否是第一个点，true 表示是第一次点击的，false 表示是第二次点击的。

2) 如果选择的一对图片是同一种的图片，那么它们在表示地图的二维数组中的数值相等。

定义 CGameControl::Link()函数，在该函数中添加代码，先这一对图片是否是同一张，然后判断是否是同一种图片。

```
bool CGameControl::Link(Vertex avPath[4], 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_s
vSelSec.col])
    {
        return false;
    }
    //……
}
```

(2) 对选中的两个图片进行连通判断。

1) 三种连通方式：

① 一条直线消子

② 两条直线消子

③ 三条直线消子

2) 添加 CGameLogic::IsLink()函数来判断选中的两点是否能够消子，并记录关键点。

IsLink()定义格式为：bool IsLink(int** pGameMap, Vertex v1, Vertex v2)

3) CGameControl::Link()中调用 CGameLogic::IsLink()函数进行连通判断。

(3) 获取连接路径

1) 在 CGameLogic 中，添加成员函数 GetVexPath()，获取连接路径。通过参数返回路径，通过返回值返回数组中实际元素个数。

```
int CGameLogic::GetVexPath(Vertex avPath[4])
{
    for(int i = 0; i < m_nVexNum; i++)
```

```
    {
        avPath[i] = m_avPath[i];
    }
    return m_nVexNum;
}
```

2) 在 CGameControl::Link() 函数中，判断一对图片是否能够连通，调用 CGameLogic::GetVexPath()函数获取连接路径。

```
bool CGameControl::Link(Vertex avPath[4], int &nVexnum)
{
    //……
    // 判断是否连通
    if(m_GameLogic.IsLink(m_pGameMap, m_svSelFst, m_svSelSec))
    {
        // 消子
        m_GameLogic.Clear(m_pGameMap, m_svSelFst, m_svSelSec);

        // 返回路径顶点
        nVexnum = m_GameLogic.GetVexPath(avPath);

        return true;
    }
    return false;
}
```

3) 在 CGameDlg:: OnLButtonUp()函数中调用 Link()函数，进行判断是否为相同的图片，判断是否能够连通，以及消子操作。

```
void CGameDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
    //……
    if(m_bFirstPoint)                    // 第一个点
    {
        //……
    }
    else                                 // 第二个点
    {
        ……
        // 连子判断
        bool bSuc = m_GameC.Link(avPath, nVexnum);
        if(bSuc == true)
        {
            // 画提示线
            DrawTipLine(avPath, nVexnum);
            // 更新地图
```

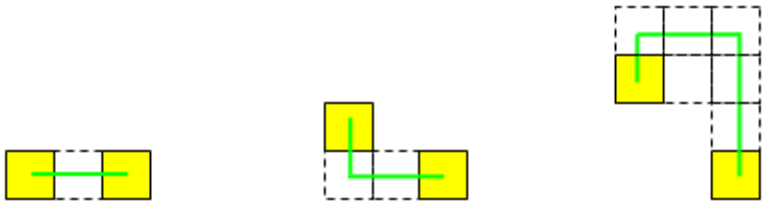
```
                UpdateMap();
            }
            //……
        }

        m_bFirstPoint = !m_bFirstPoint;
}
```

(4) 绘制连接线

当选中的两张图片是同种图片时，根据顶点绘制直线在选中的两张图片间绘制一条直线。起始点为第一次选中的图片的中心位置，终点为第二次选中图片的中心位置。

连接线颜色为 RGB(0, 255, 0)。



图错误!文档中没有指定样式的文字。**-10 绘制连线**

1) 在 CGameDlg 类中添加成员函数 DrawTipLine()。

函数定义格式为：void DrawTipLine(void)

2) 绘制连接线时，调用 CDC::MoveTo()函数和 CDC::LineTo()函数。

3) 连接线的起点为第一次点击的图片的中心位置，终点为第二次点击图片的中心位置。

```
void CGameDlg::DrawTipLine(Vertex asvPath[4], int nVexnum)
{
    CClientDC dc(this);
    CPen penLine(PS_SOLID, 2, RGB(0, 255, 0));
    CPen* pOldPen = dc.SelectObject(&penLine);

    dc.MoveTo(m_ptGameTop.x + asvPath[0].col * m_sizeElem.cx + m_sizeElem.cx / 2,
        m_ptGameTop.y + asvPath[0].row * m_sizeElem.cy + m_sizeElem.cy / 2);

    for(int i = 1; i < nVexnum; i++)
    {
        dc.LineTo(m_ptGameTop.x + asvPath[i].col * m_sizeElem.cx + m_sizeElem.cx / 2,
            m_ptGameTop.y + asvPath[i].row * m_sizeElem.cy + m_sizeElem.cy / 2);
    }

    dc.SelectObject(pOldPen);
}
```

4) 为了方便清除提示线，将提示线直接绘制到视频内存 DC 中，而不要绘制到 m_dcMem 中。