扫雷问题第三次研究

2008 年的时候，我研究了一段时间的 Win32 汇编语言，写了《Windows 扫雷游戏》。原文在 http://www.aogosoft.com/downpage.asp?mode=viewtext&id=212，其中给出了一个扫雷游戏的作弊器。
2009 年的时候，又写了一篇《扫雷问题再探究》。原文在 http://www.aogosoft.com/downpage.asp?mode=viewtext&id=242，通过反编译解释了为什么第一次点击永远不会被地雷炸到。

前几天，我在网上发现了一份号称是 MS 的扫雷程序源代码的程序，于是又做了一番研究。

再问为什么第一次点击不会被炸到？

查看 Source Code 之后发现，原来是：左键点下才开始游戏，然后根据你点中的位置再开始布雷的，布雷过程中特别绕开了你点的位置。之前我反编译结果是：扫雷程序自动判断是否为第一次按下，如果是，那么将下面的地雷挪开。不过这套代码看起来是先让游戏者点击，然后再进行布雷，在布雷过程中绕开点击位置。这套 code 应该是属于比较老的扫雷游戏的代码，不同版本之间处理方式可能会有所不同。

MineWnd.cpp

```
//布雷
LayMines(m_pOldMine->uRow, m_pOldMine->uCol);            // lay all the mines down

//埋下随机种子
    srand( (unsigned)time( NULL ) );
    UINT i, j;

    for(UINT index = 0; index < m_uMineNum;)
    {
        //取随即数
        i = rand() % m_uYNum;
        j = rand() % m_uXNum;

        if (i == row && j == col) continue;    <----这里绕开了
        if(m_pMines[i][j].uAttrib != ATTRIB_MINE)
        {
            m_pMines[i][j].uAttrib = ATTRIB_MINE;//修改属性为雷
```

```
                index++;
            }
        }
}
```

第二个问题：能否做一个永远无法解开的扫雷游戏？

通过阅读代码，找到在每次点击之后进行判断的位置，在其中插入我的代码。第一步是加入判断条件，如果条件为真，则进行第二步；第二步是从其他位置取一个雷出来，然后将这个雷尝试插入到点击位置；第三步是检测这样替换之后是否满足当前的局面指示，如果不满足就继续第二步，从其他位置取雷进行实验。如果如果所有雷都不满足条件则退出循环，不进行插入雷的动作。

关键位置：

```
                if (m_pOldMine->uOldState == STATE_NORMAL)
                {//当该雷区域为正常未作标记才打开
                    //ZivDebug_Start
                    if (!IsMine(m_pOldMine->uRow, m_pOldMine->uCol))
                    {   //可以使用下面的随机数的方法
                        //srand(m_uMineNum);
                        //if ((m_uMineNum - m_nLeaveNum)>rand())
                        //也可以使用下面这样的很凶悍的方法：标记第一颗雷之后必死
                        if ((m_uMineNum - m_nLeaveNum)>=1)
                        {
                            PutAMine(m_pOldMine->uRow, m_pOldMine->uCol);
                        }
                    }
                    //ZivDebug_End
                    // first judge if the special MINEWND is a mine
                    //如果该区域为雷，则死亡
                    if (IsMine(m_pOldMine->uRow, m_pOldMine->uCol))
                    {
                        Dead(m_pOldMine->uRow, m_pOldMine->uCol);
                        ReleaseCapture();
                        return;
                    }
```

移动雷的过程

```
//ZivDebug_Start
//
//   找个雷放到 row col 的位置，目前已近保证 row col 处没有雷
//
```

```
UINT CMineWnd::PutAMine(UINT row, UINT col)
{
	UINT i, j,x,y;
	UINT flag;

	//首先测试  row col  上能否放下雷

	//1.先放上一个雷
	m_pMines[row][col].uAttrib = ATTRIB_MINE;

	//2.在其余地方找到一个雷
	for (i = 0; i < m_uYNum; i++)
	{
		for (j = 0; j < m_uXNum; j++)
		{
			flag=0;
			//3.找个有雷的位置
			if ((m_pMines[i][j].uAttrib == ATTRIB_EMPTY)) { flag=1; continue; }
			if ((i==row)&&(j==col)) { flag=1; continue; }

			//4.将这个雷移除然后测试能否满足盘面
			m_pMines[i][j].uAttrib = ATTRIB_EMPTY;
			//  只要测试移除雷移除位置周围的 8 个位置即可
			for ( x=(i == 0) ? 0 : i - 1; x< i+2; x++)
				{
					for ( y=(j == 0) ? 0 : j - 1; y< j+2; y++)
					{
						if ((m_pMines[x][y].uState>=STATE_NUM8) &&
(m_pMines[x][y].uState<=STATE_NUM1))
						{
							if (m_pMines[x][y].uState!=15-GetAroundNum(x, y)) {flag=1;
goto Next1;}
						}
					}
				}

Next1:
			//  标志表示不符合情况，恢复雷，跳出本轮循环
			if	(flag==1) {	m_pMines[i][j].uAttrib = ATTRIB_MINE; continue;}

			//  再测试放置雷位置周围的 8 个位置即可
			for ( x=(row == 0) ? 0 : row - 1; x< row+2; x++)
				{
					for ( y=(col == 0) ? 0 : col - 1; y< col+2; y++)
```

```
                    {
                        if ((m_pMines[x][y].uState>=STATE_NUM8) && (m_pMines[x][y].uState<=STATE_NUM1))
                        {
                            if (m_pMines[x][y].uState!=15-GetAroundNum(x, y)) {flag=1; goto Next2;}
                        }
                    }
                }

Next2:
            // 标志表示不符合情况，跳出本轮循环
            if    (flag==1) {m_pMines[i][j].uAttrib = ATTRIB_MINE; continue;}
            else { goto Found;}
        }
    }
Found:
    //如果查找过一遍都没有合适位置，那么需要复原
    if (flag==1) {  m_pMines[row][col].uAttrib = ATTRIB_EMPTY;}
    return 0;
}
//ZivDebug_End
```

其实我更倾向于找到一个好的随机数的方法，然后完全可以用来和那些“扫雷高手”来进行打赌……☺

程序还可以做如下改进：

1. 因为每次测试都是移动一个雷，因此可能会漏掉一些实际上能够在翻开位置埋下雷的情况；
2. 针对当前使用的版本，可以编写一个直接在内存中修改雷分布的外挂，更隐蔽，更不容易被别人发现。

之前网上取得的扫雷源程序在

OrgMine.zip

修改之后的源程序在

NerverS.zip 中

修改内容在下面两个文件中

MineWnd.cpp

MineWnd.h

需要注意的是，编译和运行过程中可能需要一些 Lib 和 DLL（估计是因为源程序是很老的）位于 runtime.zip 中。

Zoologist

2012-9-5