目录

扫雷游戏插件的目标

1. 当鼠标放在扫雷的方格中时，会显示是否有雷。

2. 一键扫雷，快捷键是 F5

扫雷游戏的分析

1. 具备的知识

会编写 DLL

会写注入读取或者写入内存的代码
能够分析出扫雷程序中的信息
总结:需要先分析再开发

2. 需要分析的数据
   ① 扫雷数组的大小
   ② 扫雷数组的宽度、高度
   ③ 扫雷数组中雷的数量
3. 需要分析的代码
   ① 找到遍历扫雷数组的代码，或者分析出如何遍历数组
   ② 找到屏幕坐标转换数组下标的代码，或者分析出这个过程
   ③ 找到数组下标转换屏幕坐标的代码，或者分析出这个过程
4. 代码框架
   接管扫雷窗口的回调函数，处理 F5 的按键消息
   SetWindowLong

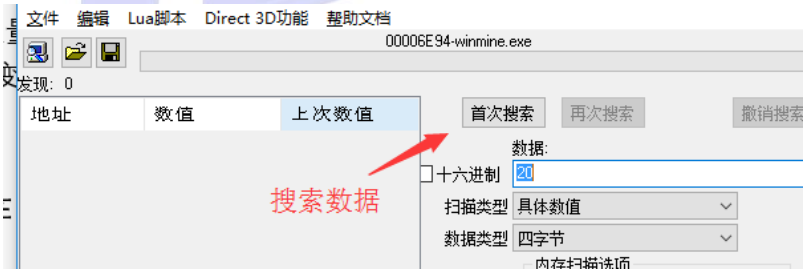
扫雷游戏-分析数据

① 扫雷数组的大小
② 扫雷数组的宽度、高度
③ 扫雷数组中雷的数量

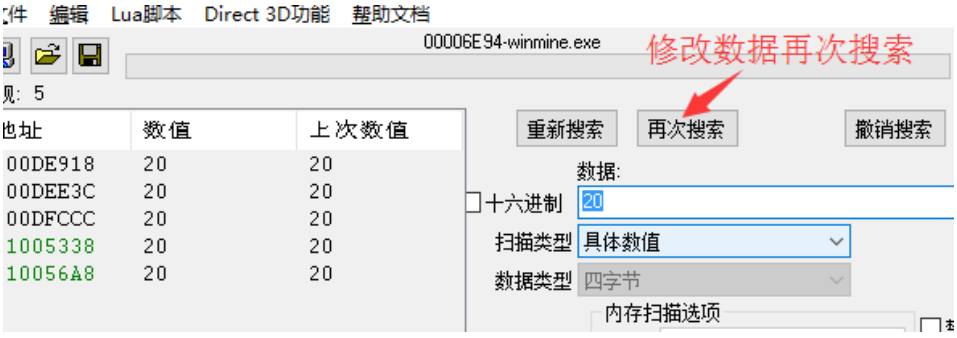
根据宽度或者高度的值的变化，使用 CE 在内存中搜索数值，从而能找到宽度的地址

CE 使用

① 修改程序数据的值，在 CE 中搜索



② 每次修改之后，再使用 CE 搜索，最后确定可能的值

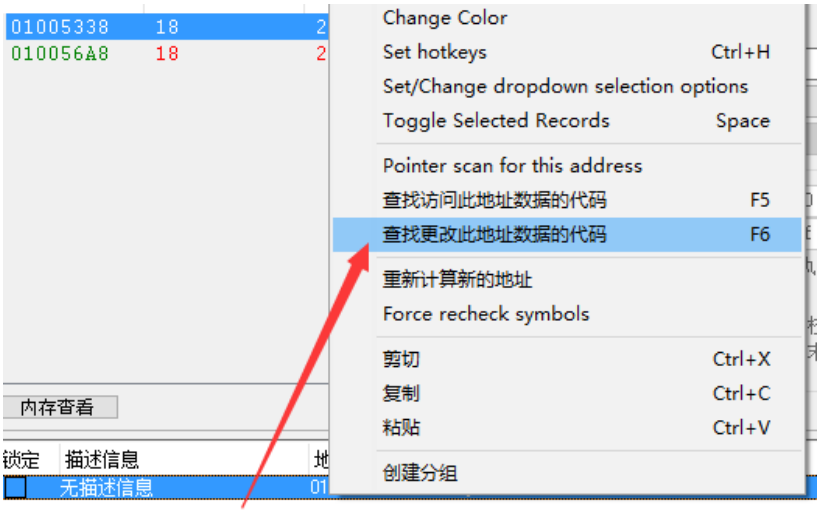


注意：绿色的地址就是可以映射到文件偏移的地址，绿色地址称为基址

```
.data:0100532D                  db    ? ;
.data:0100532E                  db    ? ;
.data:0100532F                  db    ? ;
.data:01005330 dword_1005330    dd ?
.data:01005330
.data:01005334 dword_1005334    dd ?
.data:01005334
.data:01005338 dword_1005338    dd ?
.data:01005338
.data:0100533C                  align 10h
```

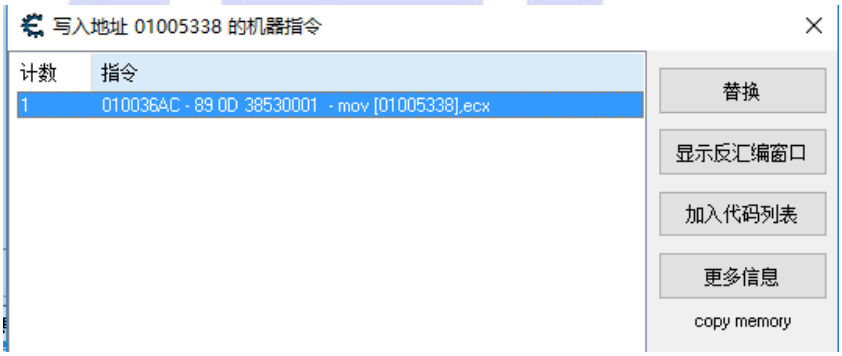
基址就是全局变量

③ 在可能的值上查找此地址数据的代码



在可能的值上查找更改此地址数据的代码
即下硬件写入断点，记录代码

再次修改数据，会有代码被记录



代码处：10036AC

高度：1005338
宽度：1005334
雷数：1005330

扫雷游戏-分析代码

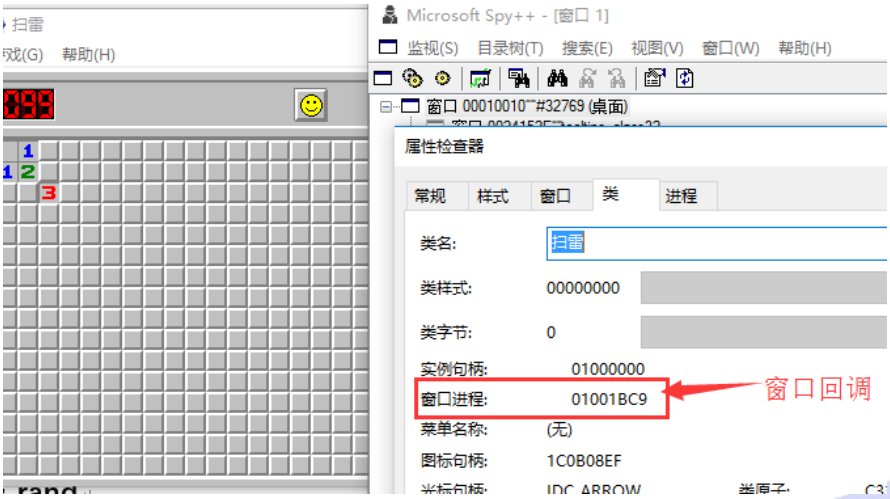
① 从数值变化的代码处开始分析
从代码处：10036AC 开始跟踪分析
② 敏感 API 下断点，再进一步分析

定时器创建和销毁：SetTimer KillTimer

随机函数：rand

③ 从窗口回调函数开始分析，分析鼠标按下消息

使用 spy++查找程序的窗口回调



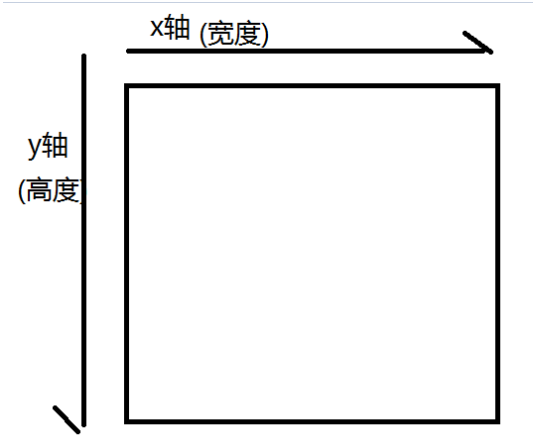
地址：1001BC9

分析 1-从数值变化的代码处开始分析

从代码处：10036AC 开始跟踪分析

```
010036A6  >  5B              POP EBX
010036A7  .  A3 34530001     MOV DWORD PTR DS:[0x1005334],EAX      修改宽度
010036AC  .  890D 3853000    MOV DWORD PTR DS:[0x1005338],ECX      修改高度
010036B2  .  E8 1EF8FFFF     CALL winmine.01002ED5
010036B7  .  A1 A4560001     MOV EAX,DWORD PTR DS:[0x10056A4]      雷的数量
010036BC  .  893D 6051000    MOV DWORD PTR DS:[0x1005160],EDI      修改雷数
010036C2  .  A3 30530001     MOV DWORD PTR DS:[0x1005330],EAX
010036C7  >  FF35 3453000    PUSH DWORD PTR DS:[0x1005334]         压入是宽度
010036CD  .  E8 6E020000     CALL <winmine.随机生成坐标>           随机生成X坐标
010036D2  .  FF35 3853000    PUSH DWORD PTR DS:[0x1005338]         压入的是高度
010036D8  .  8BF0            MOV ESI,EAX
010036DA  .  46              INC ESI
010036DB  .  E8 60020000     CALL <winmine.随机生成坐标>           随机生成Y坐标
010036E0  .  40              INC EAX
010036E1  .  8BC8            MOV ECX,EAX
010036E3  .  C1E1 05         SHL ECX,0x5
010036E6  .  F68431 40530    TEST BYTE PTR DS:[ECX+ESI+0x1005340],0x80
010036EE  .^ 75 D7           JNZ SHORT winmine.010036C7
```

修改完数据之后，应该初始化扫雷数组、随机生成雷…



随机生成雷的代码中，是一个循环，循环生成雷的 x,y 坐标，然后写入到对应缓冲区

```
010036E1  .  MOV ECX,EAX
010036E3  .  SHL ECX,0x5                          y = (y+1)*32
010036E6  .  TEST BYTE PTR DS:[ECX+ESI+0x1005340],0x80   判断(x+1+(y+1)*32+0x1005340)中的值是否为0x80
010036EE  .^ JNZ SHORT winmine.010036C7
010036F0  .  SHL EAX,0x5                          y = (y+1)*32
010036F3  .  LEA EAX,DWORD PTR DS:[EAX+ESI+0x1005340]
010036FA  .  OR BYTE PTR DS:[EAX],0x80            写入雷的标记(8F)
010036FD  .  DEC DWORD PTR DS:[0x1005330]
01003703  .^ JNZ SHORT winmine.010036C7
```

基地址：0x1005340

X 坐标计算：随机数+1

Y 坐标计算：(随机数+1)*32

根据内存情况可以分析出一些标记：

0F ： 初始化的值

8F ： 雷的标记

4X ： 周围有几颗雷就是 4X，41，42，43，44……

10 ： 边界

有了随机雷的生成，那么在雷生成前面的 CALL，应该就是初始化雷区数组的代码，经过调试查看内存，发现真的是。

```
010036A4  >  6A 06          PUSH 0x6
010036A6  >  5B             POP EBX
010036A7  .  A3 34530001    MOV DWORD PTR DS:[0x1005334],EAX     修改宽度
010036AC  .  890D 38530001  MOV DWORD PTR DS:[0x1005338],ECX     修改高度
010036B2  .  E8 1EF8FFFF    CALL <winmine.初始化雷区数组>        初始化雷区数组
010036B7  .  A1 A4560001    MOV EAX,DWORD PTR DS:[0x1005...]     雷的数量
010036BC  .  893D 60510001  MOV DWORD PTR DS:[0x1005160],EDI
010036C2  .  A3 30530001    MOV DWORD PTR DS:[0x1005330],EAX     修改雷数
010036C7  >  FF35 34530001  PUSH DWORD PTR DS:[0x1005334]        压入是宽度
```

分析初始化雷区数组代码，发现分为三步

1. 初始化全部缓冲区为 0x0F
2. 初始化行标记为 0x10
3. 初始化列标记位 0x10

```
01002ED5  $  MOV EAX,0x360                          数组大小 0x360
01002EDA  >  DEC EAX
01002EDB  .  MOV BYTE PTR DS:[EAX+0x1005340],0xF    将数组内存全部填充为 0F
01002EE2  .^ JNZ SHORT winmine.01002EDA
01002EE4  .  MOV ECX,DWORD PTR DS:[0x1005334]       宽度
01002EEA  .  MOV EDX,DWORD PTR DS:[0x1005338]       高度
01002EF0  .  LEA EAX,DWORD PTR DS:[ECX+0x2]         宽度+2
01002EF3  .  TEST EAX,EAX
01002EF5  .  PUSH ESI
01002EF6  .v JE SHORT winmine.01002F11
01002EF8  .  MOV ESI,EDX                            esi=edx=高度
01002EFA  .  SHL ESI,0x5                            esi=高度*32
01002EFD  .  LEA ESI,DWORD PTR DS:[ESI+0x1005360]   esi=高度*32+基地址+32
01002F03  >  DEC EAX                                递减 x坐标(宽度)
01002F04  .  MOV BYTE PTR DS:[EAX+0x1005340],0x1    第一行初始化为10
01002F0B  .  MOV BYTE PTR DS:[ESI+EAX],0x10         最后一行初始化为10
01002F0F  .^ JNZ SHORT winmine.01002F03
01002F11  >  LEA ESI,DWORD PTR DS:[EDX+0x2]         esi=高度+2
01002F14  .  TEST ESI,ESI
01002F16  .v JE SHORT winmine.01002F39
01002F18  .  MOV EAX,ESI
01002F1A  .  SHL EAX,0x5                            eax=(高度+2)*32
01002F1D  .  LEA EDX,DWORD PTR DS:[EAX+0x1005340]   edx=(高度+2)*32+基地址
01002F23  .  LEA EAX,DWORD PTR DS:[EAX+ECX+0x1005   eax=(高度+2)*32+基地址+宽度
01002F2A  >  SUB EDX,0x20
01002F2D  .  SUB EAX,0x20
01002F30  .  DEC ESI                                递减y坐标(高度)
01002F31  .  MOV BYTE PTR DS:[EDX],0x10             初始化每行的列首
01002F34  .  MOV BYTE PTR DS:[EAX],0x10             初始化每行的列尾
01002F37  .^ JNZ SHORT winmine.01002F2A
01002F39  >  POP ESI
```

搭建代码框架

第一步，定义一些值和函数

```
WNDPROC g_OldProc = NULL;
HWND g_hWnd = NULL;

LRESULT WINAPI WindowProc (
    _In_ HWND hWnd,
    _In_ UINT Msg,
    _In_ WPARAM wParam,
    _In_ LPARAM lParam)
{
    if (Msg == WM_KEYDOWN && wParam == VK_F5)
    {
        CString strString;
        strString.Format(L"wParam = %p", wParam);
        OutputDebugString(strString.GetBuffer());

        return DefWindowProc(hWnd, Msg, wParam, lParam);
```

```
    }

    return CallWindowProc(g_OldProc, hWnd, Msg, wParam, lParam);
}
```

第二步，修改回调函数

```
    // 修改扫雷窗口的回调函数
    // 1. 找到扫雷窗口
    g_hWnd = FindWindow(NULL, L"扫雷");
    // 2. 修改窗口回调
    g_OldProc = (WNDPROC)SetWindowLong(g_hWnd, GWL_WNDPROC, (LONG)WindowProc);

    // 恢复窗口回调
    SetWindowLong(g_hWnd, GWL_WNDPROC, (LONG)g_OldProc);
```

注意：写代码要记得释放和恢复，写代码要做好 debug 的准备

经过调整编写之后，遍历扫雷的数组终于写完

```
// 基地址：0x1005340
// 高度：1005338
// 宽度：1005334
// 雷数：1005330
byte** g_MineArray = (byte**)0x1005340;
int* g_nHeight = (int*)0x1005338;
int* g_nWidth = (int*)0x1005334;
int* g_nMineCount = (int*)0x1005330;

LRESULT WINAPI WindowProc (
    _In_ HWND hWnd,
    _In_ UINT Msg,
    _In_ WPARAM wParam,
    _In_ LPARAM lParam)
{
    if (Msg == WM_KEYDOWN && wParam == VK_F5)
    {
        CString strString;
        strString.Format(L"saolei wParam = %p", wParam);
        OutputDebugString(strString.GetBuffer());

        // 遍历扫雷数组
        int nHeight = *g_nHeight; // 高度y
        int nWidth = *g_nWidth;   // 宽度x
        int nCount = *g_nMineCount;
```

```
        int nCurrentCount = 0;
        for (int j = 1; j < nHeight+1; j++)
        {
            CString strString1;
            strString1.Format(L"saolei 行：%d ", j);
            // 行遍历时，需要注意去掉边界
            for (int i = 1; i < nWidth + 2 - 1; i++)
            {
                byte byCode = *(byte*)((int)g_MineArray+i+j*32);
                if (byCode == (byte)0x8F)
                {
                    nCurrentCount += 1;
                }
                CString strCode;
                strCode.Format(L" %02x ", byCode);
                strString1 += strCode;

            }
            strString1 += L"\r\n";
            OutputDebugString(strString1.GetBuffer());

        }

        CString strString2;
        strString2.Format(L"saolei 雷数 = %d", nCurrentCount);
        OutputDebugString(strString2.GetBuffer());

        return DefWindowProc(hWnd, Msg, wParam, lParam);
    }

    return CallWindowProc(g_OldProc, hWnd, Msg, wParam, lParam);
}
```

分析-屏幕坐标转数组下标

分析思路：
① 窗口回调下消息断点，分析鼠标按下弹起消息的流程
② 静态分析窗口回调，使用 IDA 找到对应的消息处理流程

窗口回调地址：1001BC9
在地址处 下消息断点 WM_LBUTTONDOWN
在这个消息的 lParam 参数里是 x,y 坐标

```
01002099  .  8B45 14    MOV EAX,[ARG.4]
0100209C  .  C1E8 10    SHR EAX,0x10                    eax = arg4 >> 16 = y
0100209F  .  83E8 27    SUB EAX,0x27                    eax = (arg4 >> 16)-0x27
010020A2  .  C1F8 04    SAR EAX,0x4                     y =((高16位arg4 >> 16)-0x27)>>4
010020A5  .  50         PUSH EAX                        y
010020A6  .  0FB745 1   MOVZX EAX,WORD PTR SS:[EBP+0x14] 获取arg4 低16位
010020AA  .  83C0 04    ADD EAX,0x4                     x = (arg4+4)
010020AD  .  C1F8 04    SAR EAX,0x4                     x = (低16位arg4+4)>>4
010020B0  .  50         PUSH EAX                        x
010020B1  > -E8 1F110   CALL winmine.010031D4
```

将汇编代码翻译为 C 代码

```
WORD y = HIWORD(lParam);
y = (y - 0x27) >> 4;
WORD x = LOWORD(lParam);
x = (x + 4) >> 4;

byte byCode = *(byte*)((int)g_MineArray + x + y * 32);
if (byCode == (byte)0x8F)
{
    SetWindowText(hWnd,L"扫雷 - 友情提示：你踩到雷了！");
}
else {
    SetWindowText(hWnd, L"扫雷");
}
```

分析-数组下标转屏幕坐标

将上面的代码反转

```
// 发送一个鼠标按下弹起的消息
WORD y = j;
y = (y << 4) + 0x27; //y 高16位
WORD x = i;
x = (x << 4) - 4; // x 低16位

SendMessage(hWnd, WM_LBUTTONDOWN, 0, MAKELPARAM(x,y));
SendMessage(hWnd, WM_LBUTTONUP, 0, MAKELPARAM(x, y));
```

DLL 调试

1. 如果有源码，可以使用 VS 进行调试，只需要设置启动 exe 即可
2. 如果没有源码，使用 OD 调试，一般选择在调试的代码处加入特征汇编指令

比如：

Mov eax, eax

Mov eax,eax

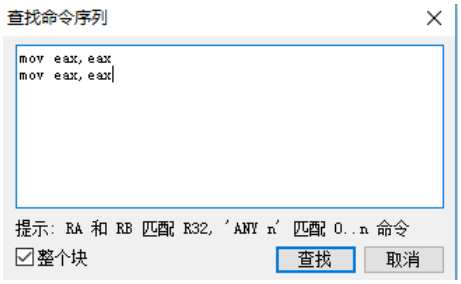
```
if (Msg == WM_KEYDOWN && wParam == VK_F5)
{
    _asm {                                    插入特征指令
        mov eax, eax
        mov eax, eax
    }
```

在注入 DLL 之后，可以搜索指令

① 选择 DLL 模块（在模块列表中找到对应模块，右键跟随入口）

② 选择 DLL 模块的代码基址(进入模块代码之后，拖到最前面)

③ 使用 Ctrl+S,输入指令，搜索



3. 在 OD 中注入 DLL