逆向-扫雷算法分析

原创

Nightsay 于 2015-05-06 20:03:21 发布 5181 收藏 3

分类专栏： 逆向分析 文章标签： 逆向 扫雷

本文链接：https://blog.csdn.net/Nightsay/article/details/45540129

逆向分析 专栏收录该内容

13 篇文章 1 订阅

订阅专栏

最近思来想去，眼看着自己就要进某厂游戏安全团队实习了，也不能整天的无所事事，所以就寻思着先找点最简单的游戏用来练练手。想到之前逆向过一些小游戏，就把之前分析的扫雷整理了一下啊，写了个外挂，发了上来。

最近实在是比较忙，快期中考试了，什么都不会，所以忙着预习去了，本文只写了关于扫雷逆向的部分，相关的外挂编写已经写差不多了，等最近有时间整理出来o(╯□╰)o

工具

分析对象：winmine/扫雷(windows xp版本)

逆向工具：ollydbg，IDA，peid，ResHacker

操作平台：windows7 旗舰版

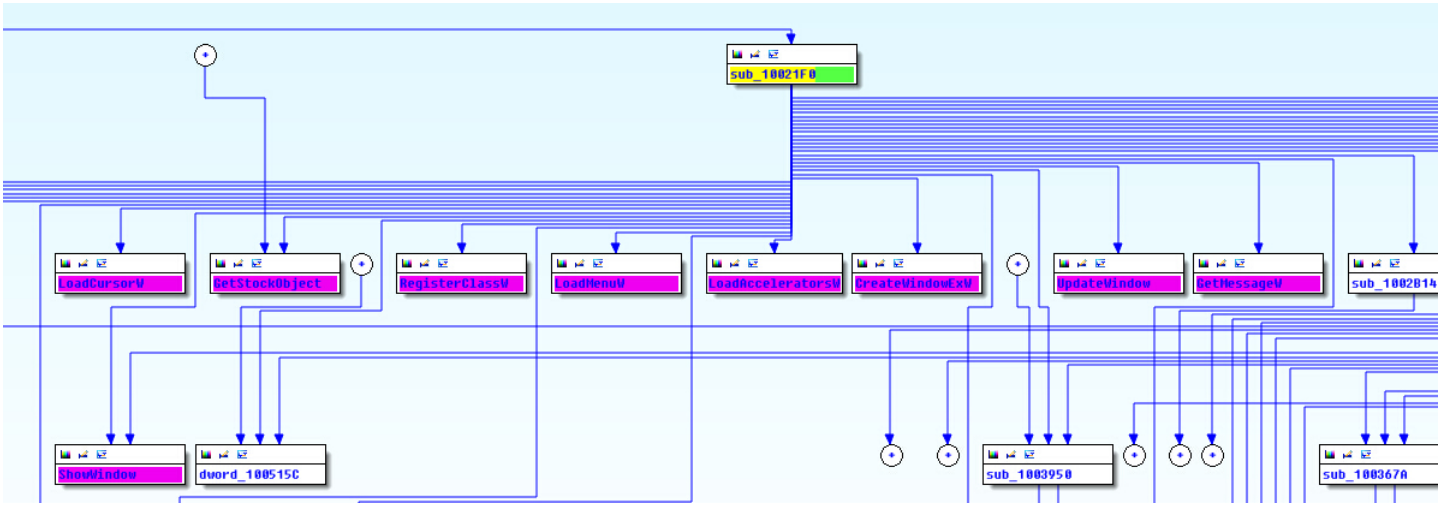
分析过程

main函数定位

peid加载winmine发现为vc编写的，没有加过壳，则可以确定加载程序的流程为：~~获取当前版本号，堆初始化，获取命令行参数，获取环境变量，分离出命令行参数，全局数据和浮点寄存器初始化~~所以在main（这里应该是winmain了）调用前，调用流程为：

GetVersion()——>_heap_init()——>GetCommandLine()——>_crtGetEnvironmentStrings()——>_setargv()——>_setenvp()——>_cinit()

所以很容易找到winmain的调用地址，跟进之后发现



发现这里便是在做一些初始化

很快找到：

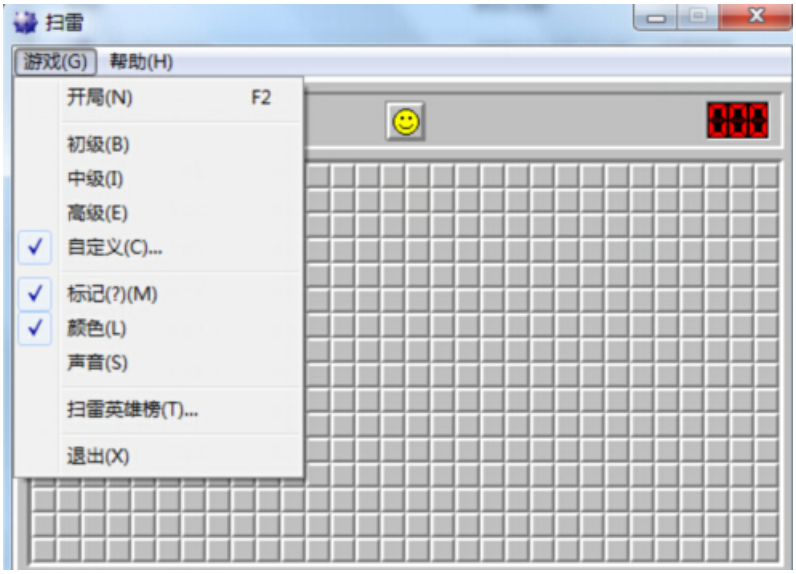
```
:010022D5                       mov     ecx, yBottom
.text:010022DB                  mov     edx, xRight
.text:010022E1                  push    edi             ; lpParam
.text:010022E2                  push    hInstance       ; hInstance
.text:010022E8                  add     ecx, eax
.text:010022EA                  push    edi             ; hMenu
.text:010022EB                  push    edi             ; hWndParent
.text:010022EC                  push    ecx             ; nHeight
.text:010022ED                  mov     ecx, dword_1005A90
.text:010022F3                  add     edx, ecx
.text:010022F5                  push    edx             ; nWidth
.text:010022F6                  mov     edx, Y
.text:010022FC                  sub     edx, eax
.text:010022FE                  mov     eax, X
.text:01002303                  push    edx             ; Y
.text:01002304                  sub     eax, ecx
.text:01002306                  push    eax             ; X
.text:01002307                  push    0CA0000h        ; dwStyle
.text:0100230C                  push    esi             ; lpWindowName
.text:0100230D                  push    esi             ; lpClassName
.text:0100230E                  push    edi             ; dwExStyle
.text:0100230F                  call    ds:CreateWindowExW
.text:01002315                  cmp     eax, edi
.text:01002317                  mov     hWnd, eax
.text:0100231C                  jnz     short loc_1002325
.text:0100231E                  push    3E8h
.text:01002323                  jmp     short loc_1002336
.text:01002325 ; ---------------------------------------------------------------------------
.text:01002325
.text:01002325 loc_1002325:                             ; CODE XREF: WinMain+12Cj
.text:01002325                  push    ebx
.text:01002326                  call    sub_1001950     ; 位置
.text:0100232B                  call    sub_1002B14     ; 装载资源
.text:01002330                  test    eax, eax
.text:01002332                  jnz     short loc_1002342
.text:01002334                  push    5
.text:01002336
.text:01002336 loc_1002336:                             ; CODE XREF: WinMain+133j
.text:01002336                  call    sub_1003950
.text:0100233B
.text:0100233B loc_100233B:                             ; CODE XREF: WinMain+ABj
.text:0100233B                  xor     eax, eax
.text:0100233D                  jmp     loc_10023C6
.text:01002342 ; ---------------------------------------------------------------------------
.text:01002342
.text:01002342 loc_1002342:                             ; CODE XREF: WinMain+142j
.text:01002342                  push    dword_10056C4
.text:01002348                  call    sub_1003CE5
.text:0100234D                  call    SetMine         ; 布置雷区
.text:01002352                  push    ebx             ; nCmdShow
.text:01002353                  push    hWnd            ; hWnd
.text:01002359                  call    ds:ShowWindow
.text:0100235F                  push    hWnd            ; hWnd
.text:01002365                  call    ds:UpdateWindow
.text:0100236B                  mov     esi, ds:GetMessageW
.text:01002371                  mov     dword_1005B38, edi
.text:01002377                  jmp     short loc_10023A4
```

分析可疑函数

在createwindow，分别调用了4个函数，然后就showwindow了，那么这4个函数肯定完成了在扫雷前用户的选择，资源的加载，雷区的绘制.
跟进第一个函数**sub_1001950**,发现主要调用了GetMenuItemRect和MoveWindow函数将窗口待会创建到制定位置，这里跟我们关心的算法没有太大关系，所以不做分析。
跟进第二个函数**sub_1002B14**，发现主要在调用LoadResource进行加载资源
跟进第三个函数**sub_1003CE5**：

```
.text:01003CE5 sub_1003CE5     proc near               ; CODE XREF: sub_1001BC9:loc_1001D00p
.text:01003CE5                                         ; WinMain+158p
.text:01003CE5
.text:01003CE5 arg_0           = dword ptr  4
.text:01003CE5
.text:01003CE5                 mov     eax, [esp+arg_0]
.text:01003CE9                 mov     dword_10056C4, eax
.text:01003CEE                 call    sub_1001516     ; 菜单选项
.text:01003CF3                 mov     eax, dword_10056C4
.text:01003CF8                 and     al, 1
.text:01003CFA                 neg     al
.text:01003CFC                 sbb     eax, eax
.text:01003CFE                 not     eax
.text:01003D00                 and     eax, hMenu
.text:01003D06                 push    eax             ; hMenu
.text:01003D07                 push    hWnd            ; hWnd
.text:01003D0D                 call    ds:SetMenu
.text:01003D13                 push    2
.text:01003D15                 call    sub_1001950
.text:01003D1A                 retn    4
.text:01003D1A sub_1003CE5     endp
```

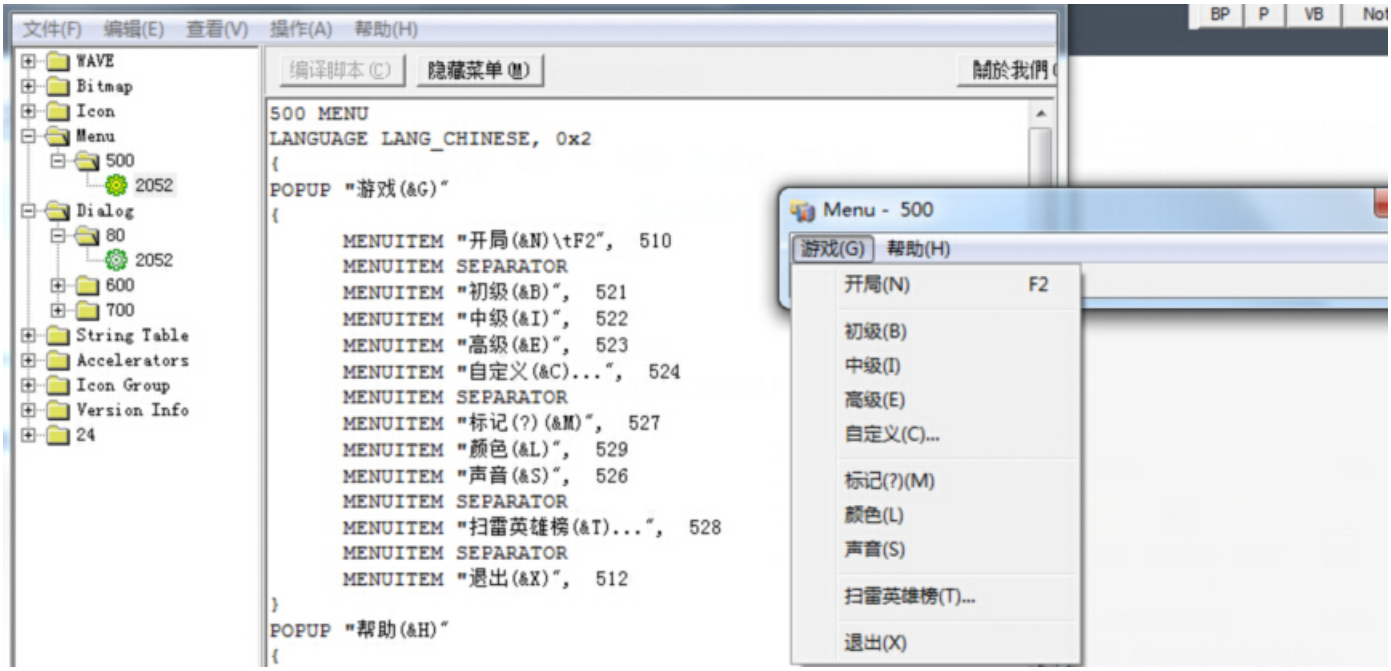
发现在调用**sub_1001516**之后调用了SerMenu，之后再次调用了第一个函数**sub_1001950**,应该就是处理用户选择不同难度而对窗口进行的重新布置吧



跟进**sub_1001516**
为了之后分析方便，将sub_1001516改名为MenuChoose，sub_1001950改名为SetWindow

```
/*注：本代码中CheckMenu为自己分析后自己起的函数名*/
.text:01001516 MenuChoose      proc near               ; CODE XREF: sub_1001B49+24p
.text:01001516                                         ; sub_1003CE5+9p
.text:01001516                 xor     eax, eax
.text:01001518                 cmp     word ptr dword_10056A0, ax
.text:0100151F                 setz    al
.text:01001522                 push    eax
.text:01001523                 push    209h            ; 521
.text:01001528                 call    CheckMenu
.text:0100152D                 xor     eax, eax
.text:0100152F                 cmp     word ptr dword_10056A0, 1
.text:01001537                 setz    al
.text:0100153A                 push    eax
.text:0100153B                 push    20Ah            ; 522
.text:01001540                 call    CheckMenu
.text:01001545                 xor     eax, eax
.text:01001547                 cmp     word ptr dword_10056A0, 2
.text:0100154F                 setz    al
.text:01001552                 push    eax
.text:01001553                 push    20Bh            ; 523
.text:01001558                 call    CheckMenu
.text:0100155D                 xor     eax, eax
.text:0100155F                 cmp     word ptr dword_10056A0, 3
.text:01001567                 setz    al
.text:0100156A                 push    eax
.text:0100156B                 push    20Ch            ; 524
.text:01001570                 call    CheckMenu
.text:01001575                 push    dword_10056C8
.text:0100157B                 push    211h            ; 529
.text:01001580                 call    CheckMenu
.text:01001585                 push    Data
.text:0100158B                 push    20Fh            ; 527
.text:01001590                 call    CheckMenu
.text:01001595                 push    dword_10056B8
.text:0100159B                 push    20Eh
.text:010015A0                 call    CheckMenu
.text:010015A5                 retn
.text:010015A5 MenuChoose      endp
```

这里差不多就应该是通过资源ID选择了，用ResHacker分析一下得证



然后分析最后一个函数SetMine(*原函数名为sun_xxxxxx，通过分析已经修改函数名*)

```
.text:010036A4
.text:010036A4 loc_10036A4:                            ; CODE XREF: SetMine+1Cj
.text:010036A4                                         ; SetMine+24j
.text:010036A4                 push    6
.text:010036A6
.text:010036A6 loc_10036A6:                            ; CODE XREF: SetMine+28j
.text:010036A6                 pop     ebx
.text:010036A7                 mov     dword_1005334, eax
.text:010036AC                 mov     dword_1005338, ecx
.text:010036B2                 call    PaintMine
.text:010036B7                 mov     eax, dword_10056A4
.text:010036BC                 mov     dword_1005160, edi
.text:010036C2                 mov     dword_1005330, eax
.text:010036C7
.text:010036C7 loc_10036C7:                            ; CODE XREF: SetMine+74j
.text:010036C7                                         ; SetMine+89j
.text:010036C7                 push    dword_1005334
.text:010036CD                 call    Rand
.text:010036D2                 push    dword_1005338
.text:010036D8                 mov     esi, eax
.text:010036DA                 inc     esi
.text:010036DB                 call    Rand
.text:010036E0                 inc     eax
.text:010036E1                 mov     ecx, eax
.text:010036E3                 shl     ecx, 5
.text:010036E6                 test    byte_1005340[ecx+esi], 80h
.text:010036EE                 jnz     short loc_10036C7
```

绘制雷区函数分析

发现期中一个函数（**PaintMine**）便是在绘制雷区
大体F5把握一下大致的意思

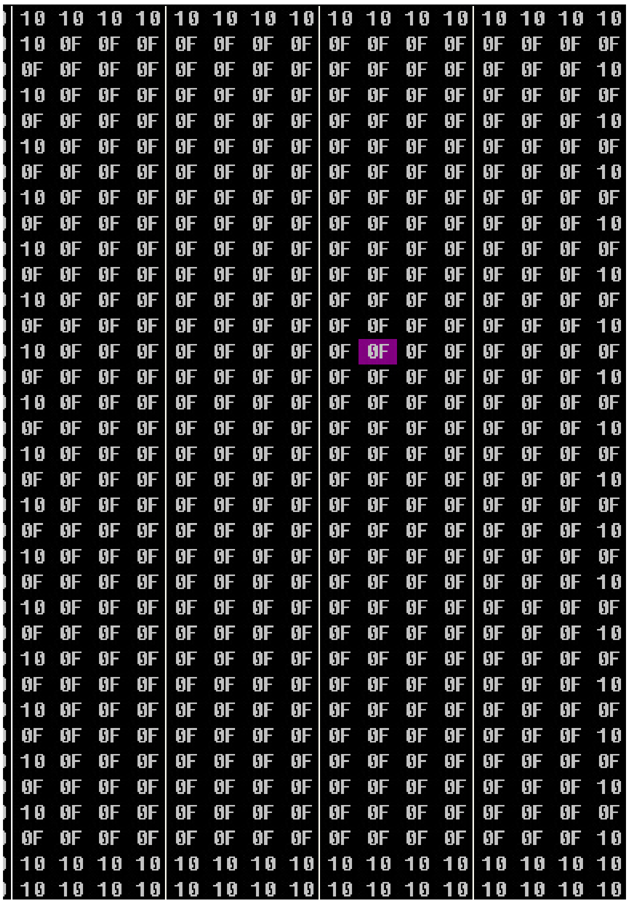
```
int __cdecl PaintMine()
{
  signed int v0; // eax@1
  int v1; // ecx@3
  int v2; // edx@3
  int result; // eax@3
  int v4; // esi@5
  char *v5; // edx@6

  v0 = 864;
  do
  {
    --v0;
    byte_1005340[v0] = 15;
  }
  while ( v0 );
  v1 = dword_1005334;
  v2 = dword_1005338;
  result = dword_1005334 + 2;
  if ( dword_1005334 != -2 )
  {
    do
    {
      --result;
      byte_1005340[result] = 16;
      *(&byte_1005360[32 * v2] + result) = 16;
    }
    while ( result );
  }
  v4 = v2 + 2;
  if ( v2 != -2 )
  {
    v5 = &byte_1005340[32 * v4];
    result = (int)((char *)&unk_1005341 + 32 * v4 + v1);
    do
    {
      v5 -= 32;
      result -= 32;
      --v4;
      *v5 = 16;
      *(_BYTE *)result = 16;
    }
    while ( v4 );
  }
  return result;
}
```

大体意思就是先将最大可能的雷区内数据全部设置为0xF，然后在根据用户选择菜单中的等级，绘制雷区的边界

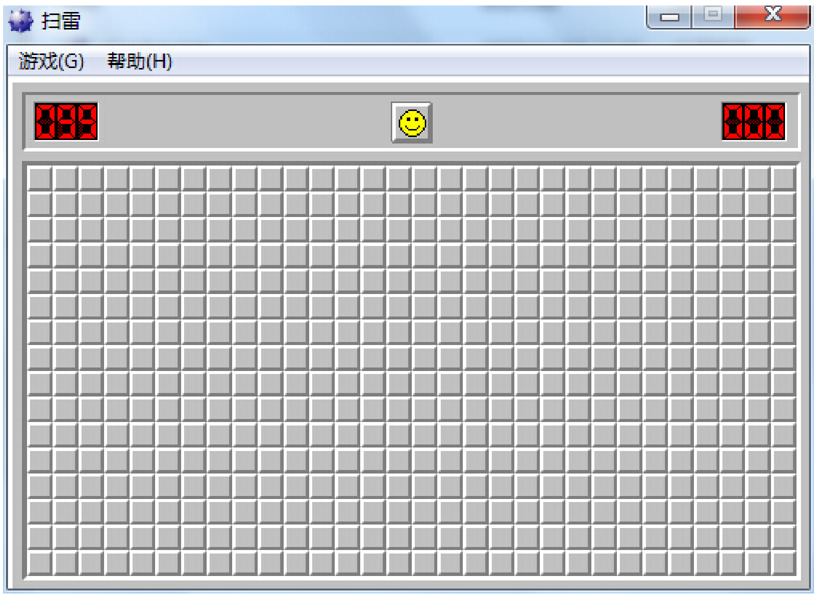
OD动态跟踪：

```
01002ED5  /$  B8 60030000   mov eax,0x360                              ;  算法如下，最大面积吧
01002EDA  |>  48           /dec eax                                   ;  循环0x360次
01002EDB  |.  C680 40530001>|mov byte ptr ds:[eax+0x1005340],0xF     ;  全部设置成0xf
01002EE2  |.^ 75 F6         \jnz Xwinmine.01002EDA
01002EE4  |.  8B0D 34530001 mov ecx,dword ptr ds:[0x1005334]         ;  长度
01002EEA  |.  8B15 38530001 mov edx,dword ptr ds:[0x1005338]         ;  宽度
01002EF0  |.  8D41 02       lea eax,dword ptr ds:[ecx+0x2]           ;  长度+2
01002EF3  |.  85C0          test eax,eax
01002EF5  |.  56            push esi
01002EF6  |.  74 19         je Xwinmine.01002F11
01002EF8  |.  8BF2          mov esi,edx
01002EFA  |.  C1E6 05       shl esi,0x5                              ;  长度左移5位
01002EFD  |.  8DB6 60530001 lea esi,dword ptr ds:[esi+0x1005360]     ;  为定位到边界值
01002F03  |>  48           /dec eax
01002F04  |.  C680 40530001>|mov byte ptr ds:[eax+0x1005340],0x10    ;  雷区的上边界设置为0x10
01002F0B  |.  C60406 10     |mov byte ptr ds:[esi+eax],0x10          ;  设置下边界的标志位0x10
01002F0F  |.^ 75 F2         \jnz Xwinmine.01002F03
01002F11  |>  8D72 02       lea esi,dword ptr ds:[edx+0x2]
01002F14  |.  85F6          test esi,esi
01002F16  |.  74 21         je Xwinmine.01002F39
01002F18  |.  8BC6          mov eax,esi
01002F1A  |.  C1E0 05       shl eax,0x5                              ;  宽度移位
01002F1D  |.  8D90 40530001 lea edx,dword ptr ds:[eax+0x1005340]
01002F23  |.  8D8408 415300>lea eax,dword ptr ds:[eax+ecx+0x1005341] ;  定位到左边界
01002F2A  |>  83EA 20       /sub edx,0x20                            ;  一行的长度
01002F2D  |.  83E8 20       |sub eax,0x20
01002F30  |.  4E            |dec esi
01002F31  |.  C602 10       |mov byte ptr ds:[edx],0x10              ;  设置左右边界
01002F34  |.  C600 10       |mov byte ptr ds:[eax],0x10
01002F37  |.^ 75 F1         \jnz Xwinmine.01002F2A
01002F39  |>  5E            pop esi                                  ;  winmine.01005AA0
```

最后在内存中画成的雷区的样子：（0x10为边界值）



而我们打开游戏界面如图，刚好与之对应



生成地雷函数分析

之后回到SetMine函数中，在调用**PaintMine**之后绘制雷区之后通过调用**Rand**随机在雷区中生成雷。
直接F5看一下SetMine

```
do
  {
    do
    {
      v1 = Rand(dword_1005334) + 1;
      v2 = Rand(dword_1005338) + 1;
    }
    while ( *(&byte_1005340[32 * v2] + v1) & 0x80 );
    *(&byte_1005340[32 * v2] + v1) |= 0x80u;
    --Num;
  }
  while ( Num );
```

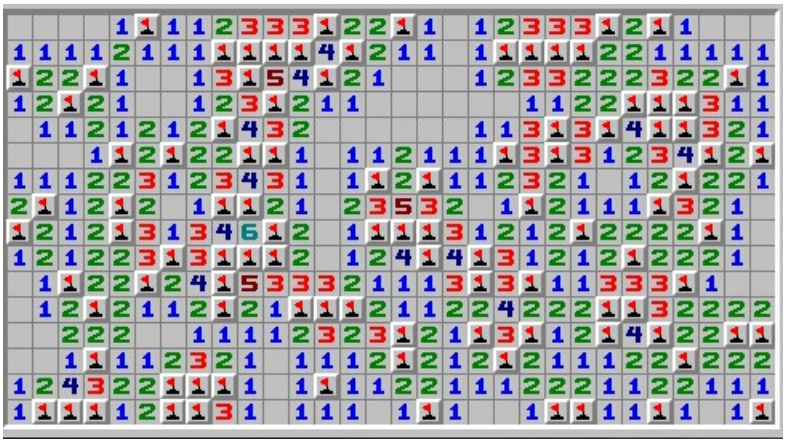
通过分析，可以明白这里的v2即是纵坐标，v1为横坐标，然后判断此处是否有地雷，没有在设置地雷（0x8F）

OD动态分析证明上述分析：

```
010036C7  > FF35 3453000 push dword ptr ds:[0x1005334]
010036CD  .  E8 6E020000 call winmine.01003940          Rand生成地雷横坐标
010036D2  .  FF35 3853000 push dword ptr ds:[0x1005338]
010036D8  .  8BF0         mov esi,eax
010036DA  .  46           inc esi
010036DB  .  E8 60020000  call winmine.01003940         Rand生成地雷纵坐标
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   判断此处是否已经有雷
010036EE  .^ 75 D7        jnz Xwinmine.010036C7
010036F0  .  C1E0 05      shl eax,0x5
010036F3  .  8D8430 40530 lea eax,dword ptr ds:[eax+esi+0x1005340]   通过偏移计算得到地址
010036FA  .  8008 80      or byte ptr ds:[eax],0x80                  将这个地方设置为雷
010036FD  .  FF0D 3053000 dec dword ptr ds:[0x1005330]               雷的数量-1
01003703  .^ 75 C2        jnz Xwinmine.010036C7
```

最后生成完雷就是这个样子：

按照这个自然就很好扫雷咯



后

关于扫雷 游戏的核心算法大体就分析这么多了，因为主要是为了下一篇的扫雷外挂的编写，所以相关的鼠标点击算法就没有记录下来了。

未完待续，大牛勿喷！