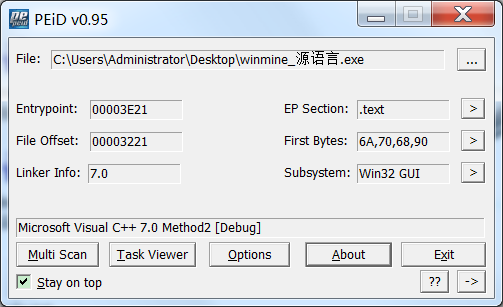
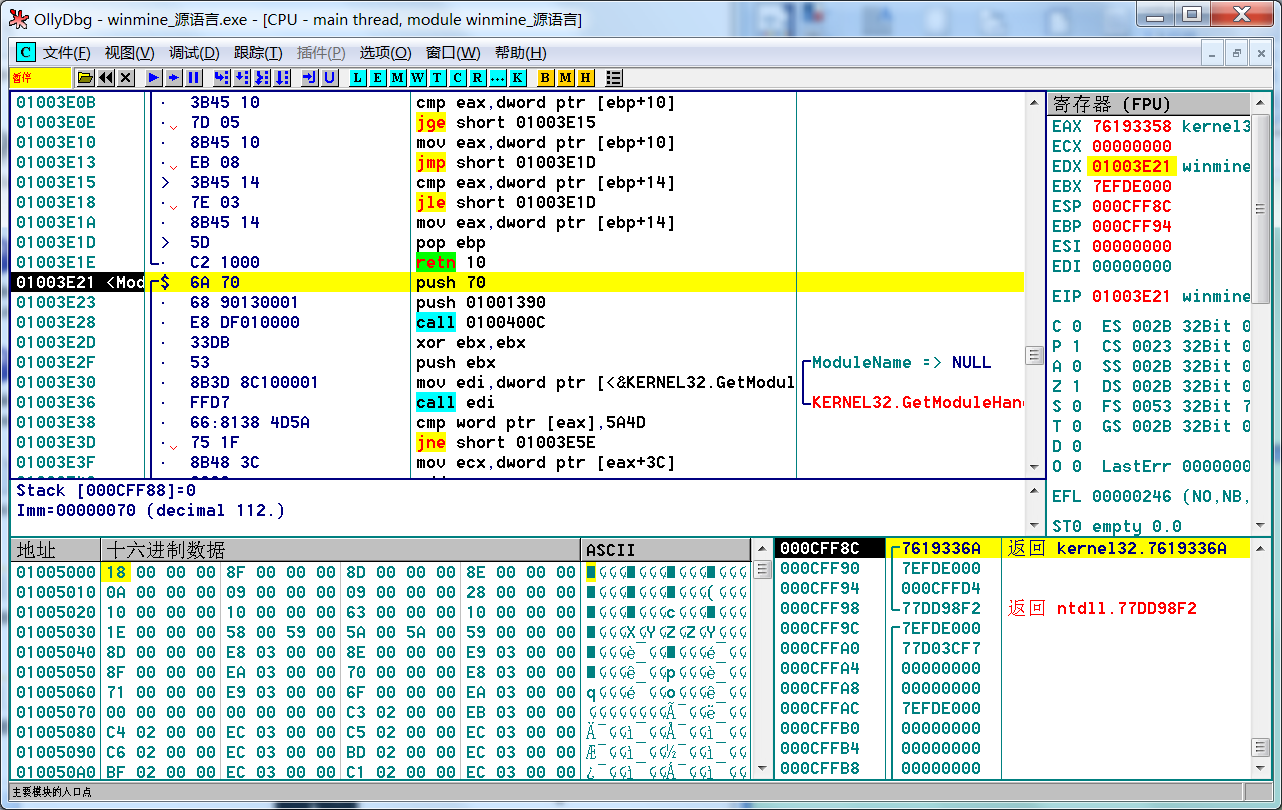
之前也写过扫雷的相关辅助，使用dll，注入，然后进行破解。但这种方式，电脑上只要装个杀软的都会一直警告。所以这次就打算直接从扫雷内部代码出发，修改扫雷内部的部分代码，达到cheat的目的。由于没有源码什么的，只能将源程序反汇编后，使用汇编加上我们要的功能。

首先，使用PEid打开查看一下。

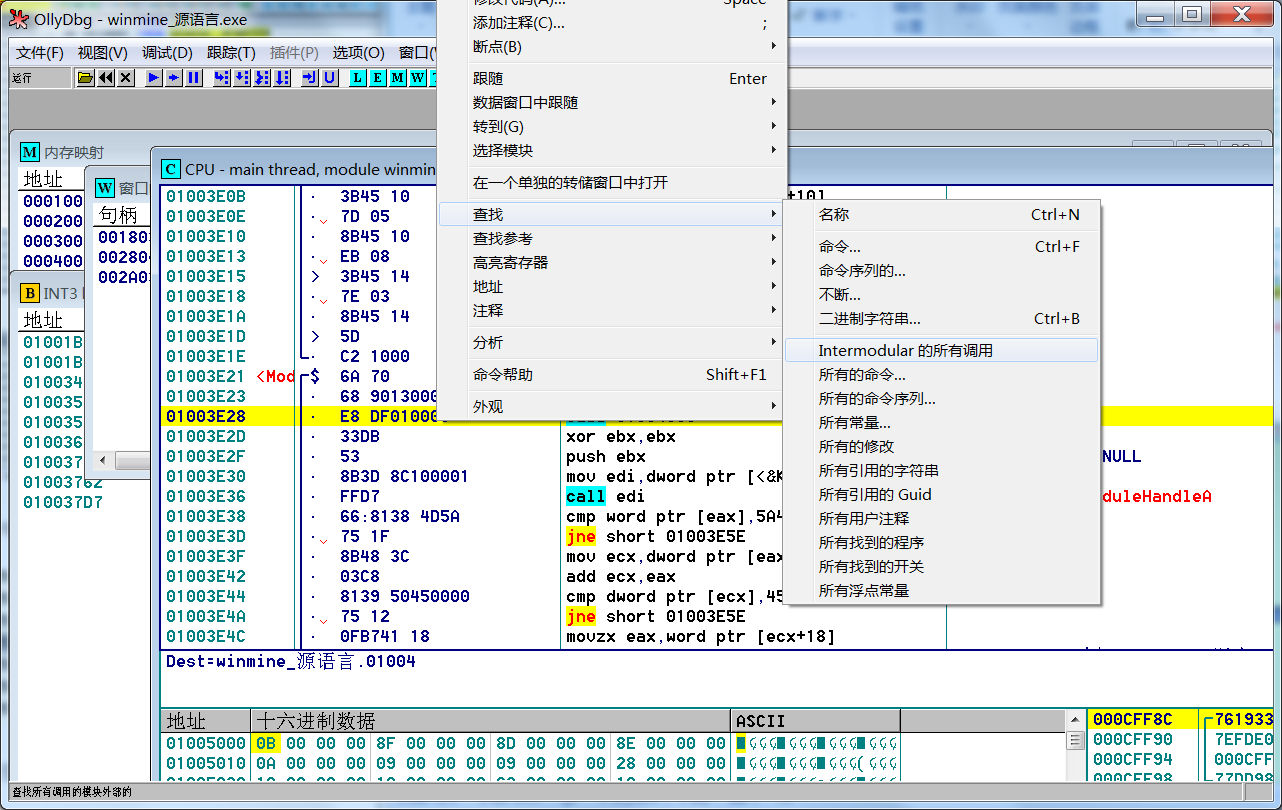


嗯，C++编写，没有壳，直接使用OllyDbg载入。

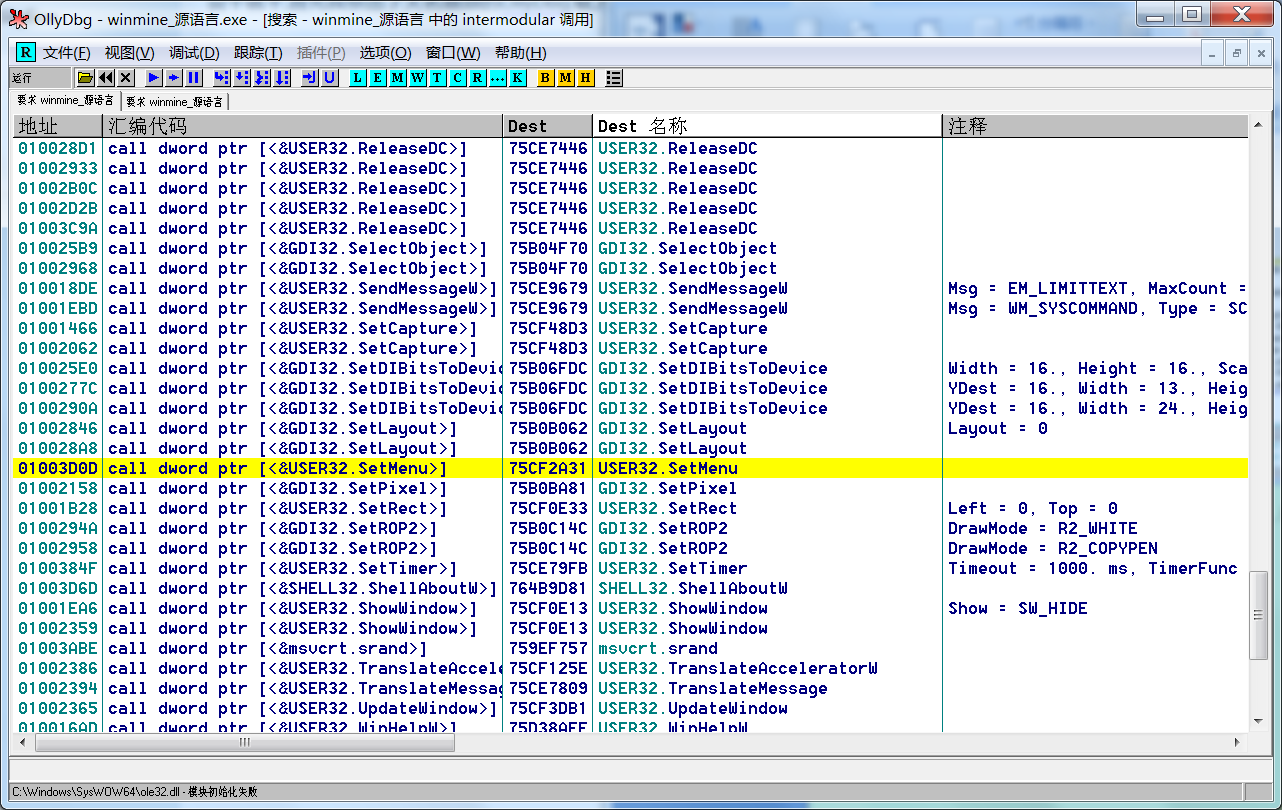


从EP处也可以明显的看出是标准的C++程序，往下浏览，也不应该有加壳加花。

点击运行，扫雷窗口打开。此时可以选择查看调用的API函数



在查找结果窗口中，可以看到一个SetMenu的函数，而不是CreateMenu之类的，也就是说，菜单通过加载相关的资源而来的，而不是新建出来的。

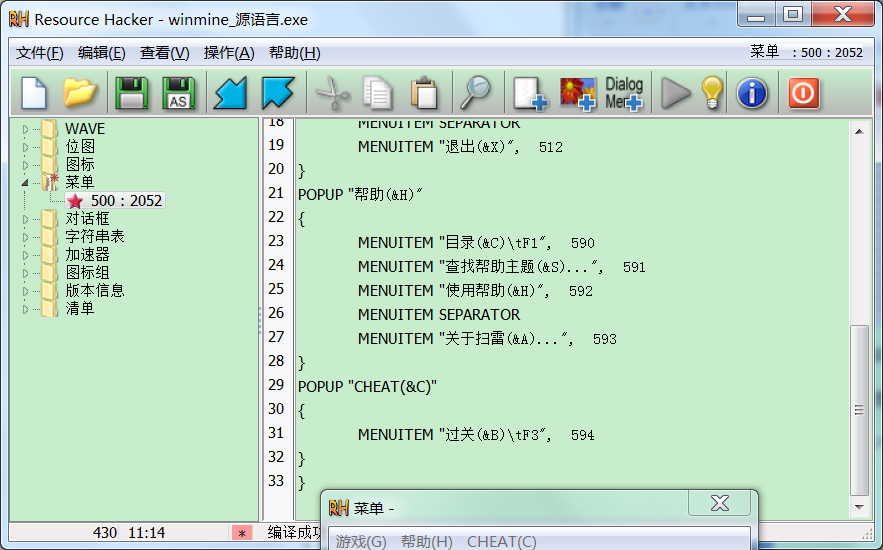




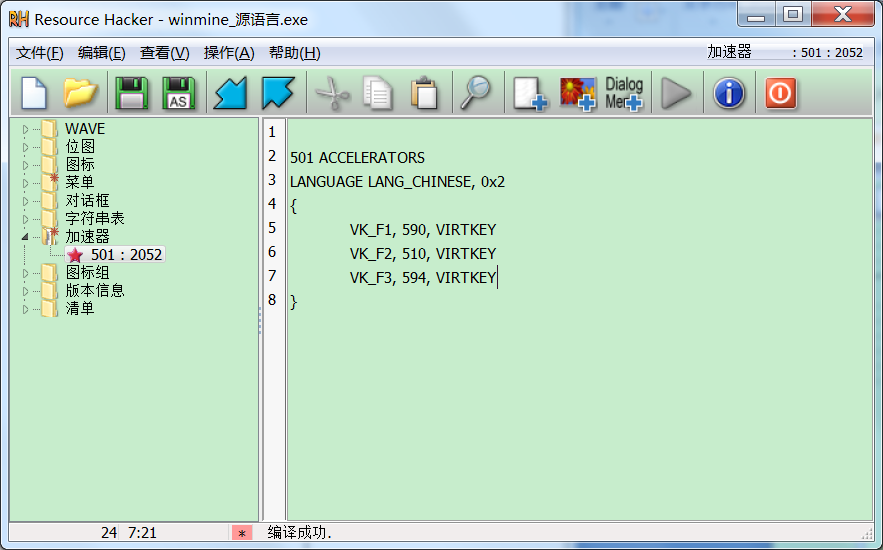
因此，我们可以考虑在菜单栏添加新的菜单项，然后我们要的功能添加进去。

这里使用一款资源修改器ResHacker，像其他的exscode都可以。

载入后，打开菜单。然后在右边添加即可。



然后再绑定一下快捷键

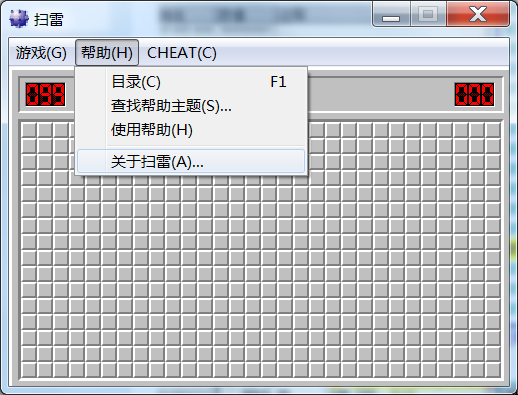


保存，重新打开，菜单项就有了，现在就只需要将功能添加进去就可以了。

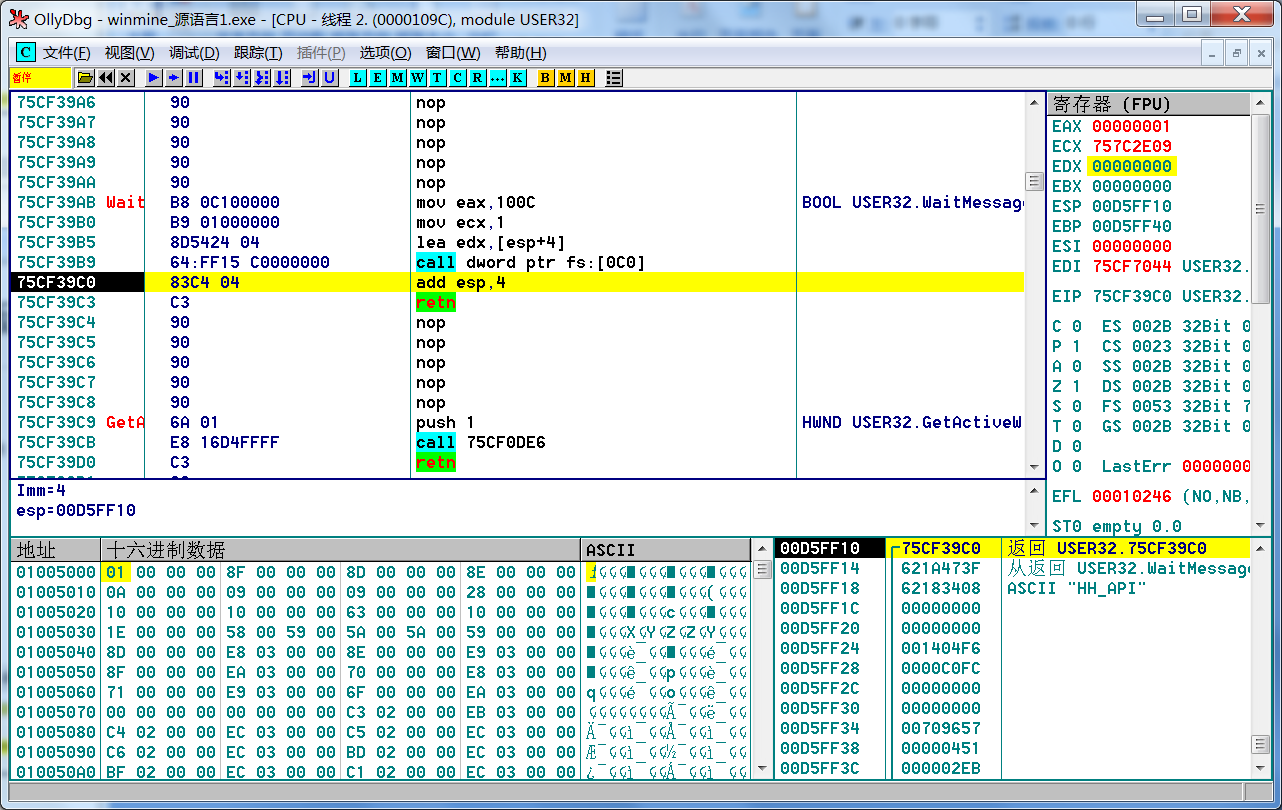
此时就需要我们找到响应菜单点击的相关代码。

以下提供几种方法供参考：

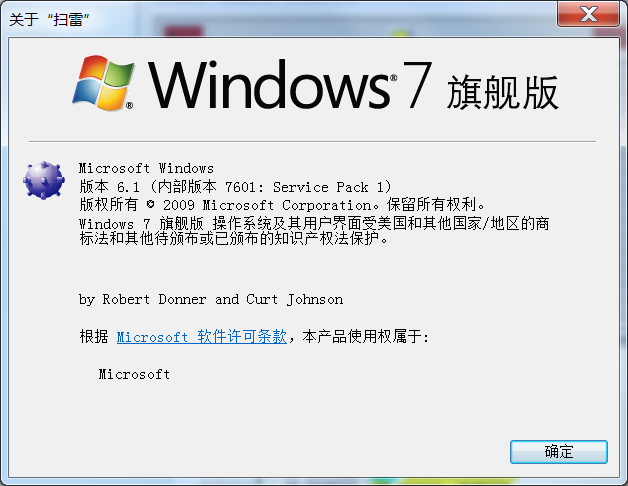
1.



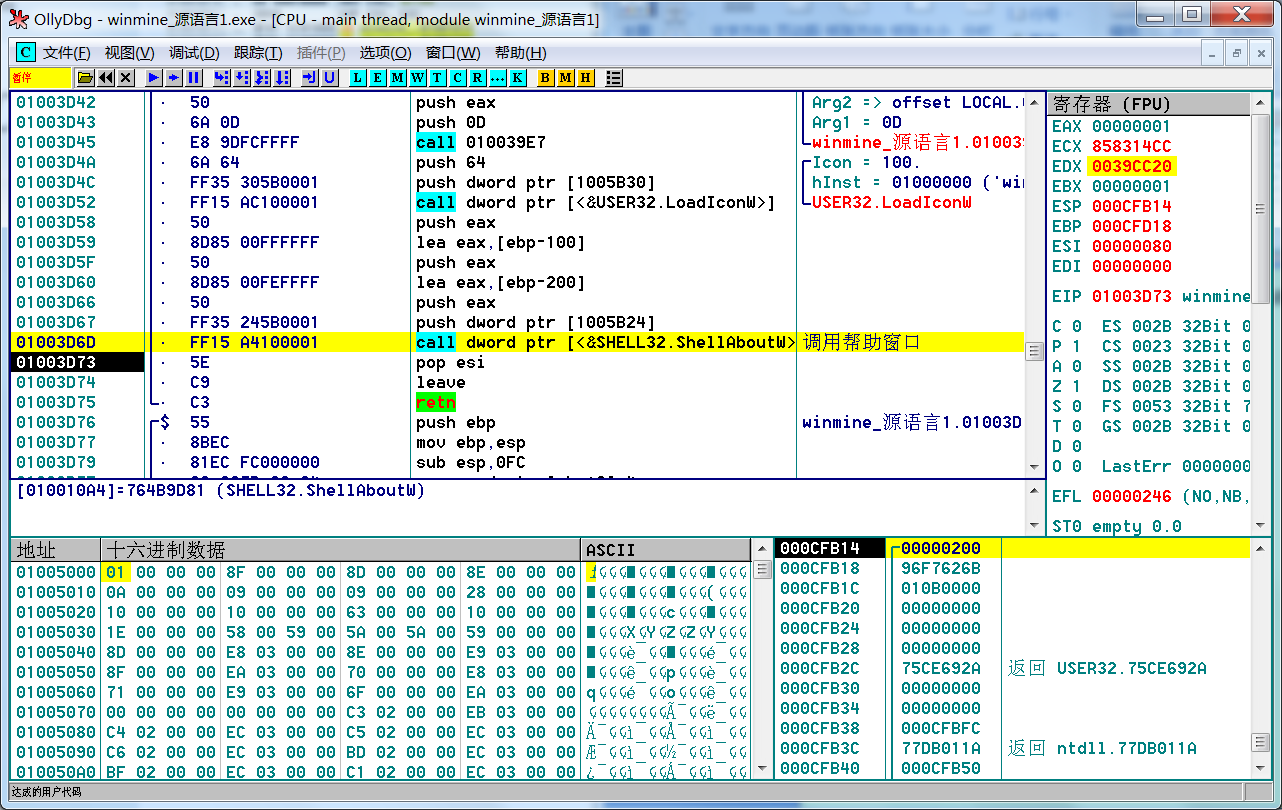
点击一个菜单相关的选项，这里以“关于扫雷”为例，点击之后，就会出现一个窗口。此时OllyDbg暂停，会发现程序停留在动态链接库的领空。



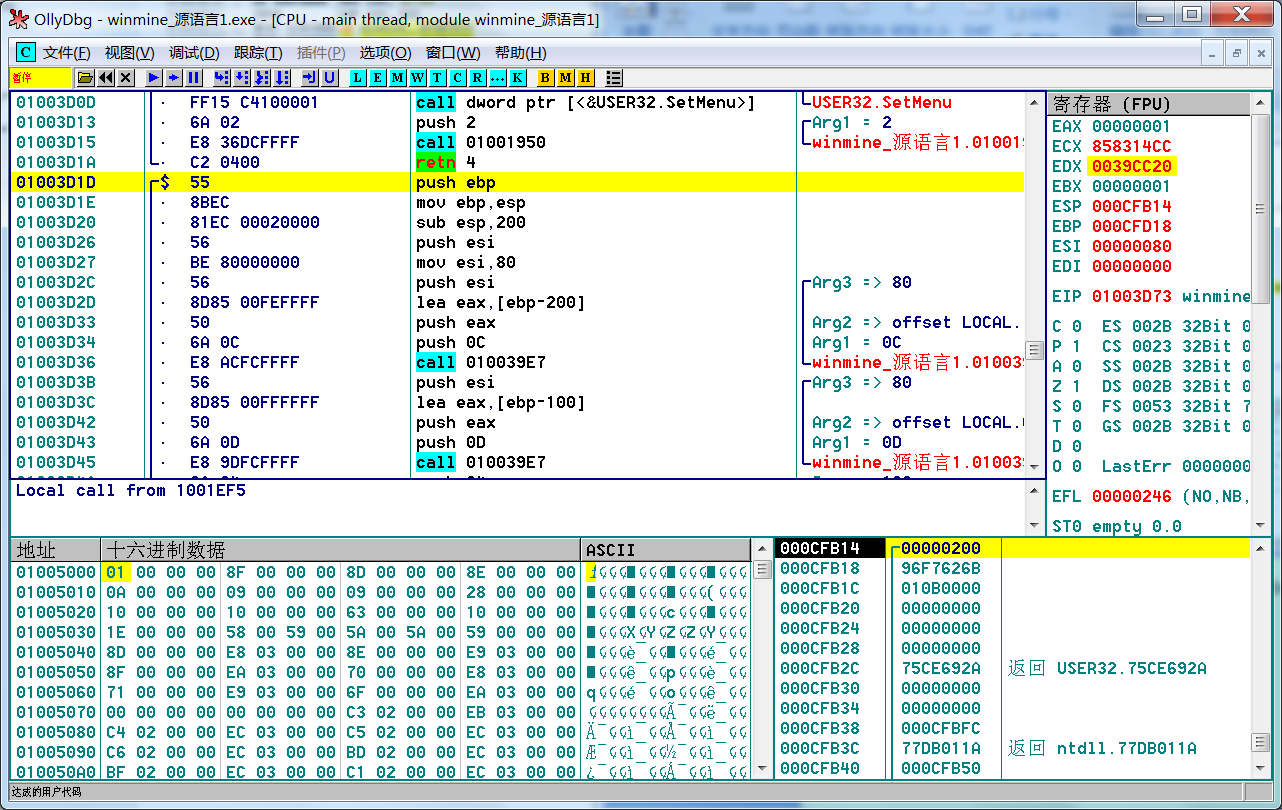
既然这样，按下Alt+F9，直到用户返回，再然后点击确定。



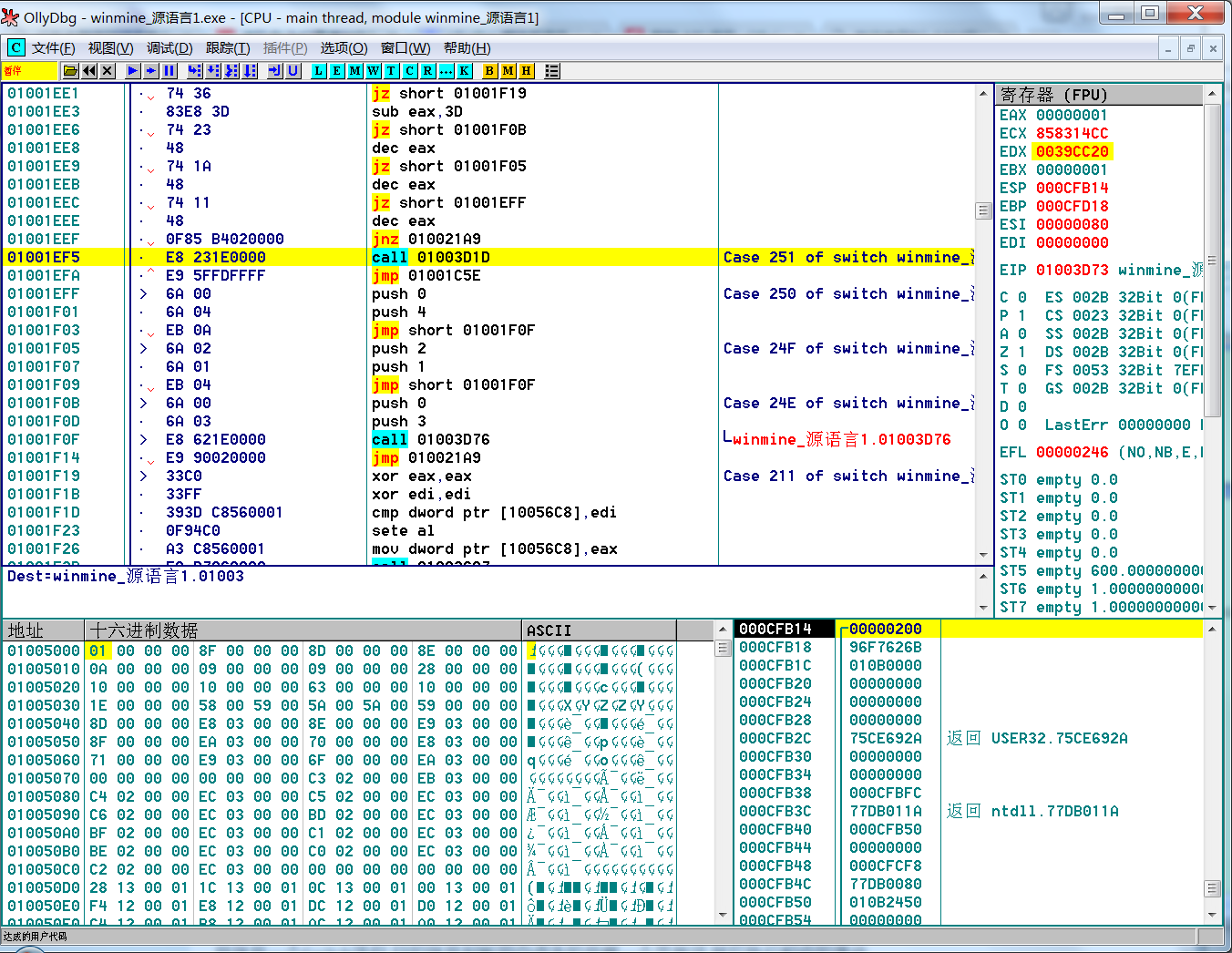
程序在01003D73处停了下来。



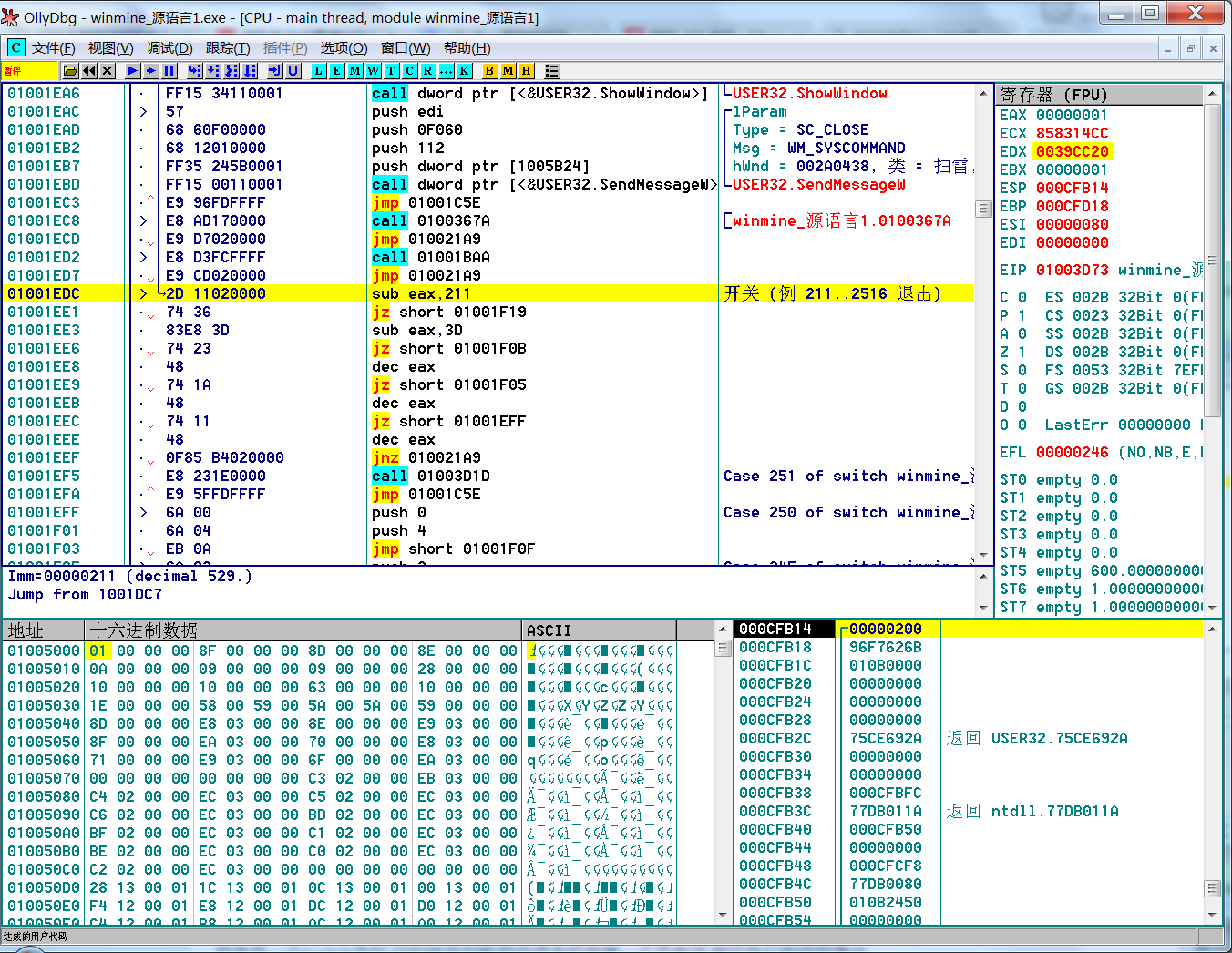
而上一行代码就是call dword ptr [<&SHELL32.ShellAboutW>],明显，这句就是调用刚刚那个窗口的语句。整体再往上看。



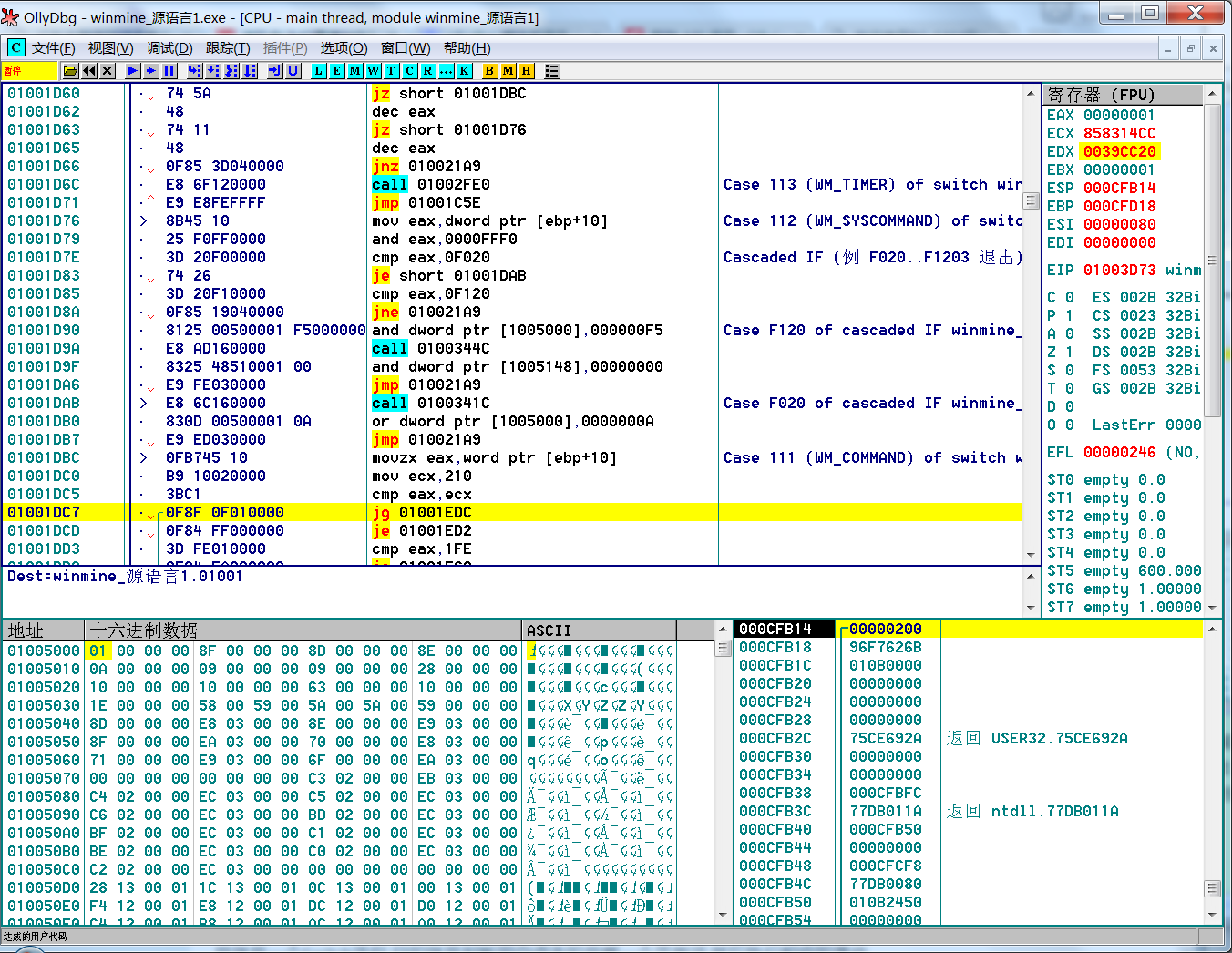
来到该函数的入口处01003D1D，此时，可以看到Local call from 1001EF5，来自这个地方的调用，ctrl+G,输入地址，然后走起，来到100EF5。



可以看到右边OllyDbg分析的结果，case 251 of switch，switch，然后251，而251的十进制又刚好是593，刚刚在resHacker中看到的正好也是593，再看一下其他的250,24F对应的也刚好是菜单中的选项ID，所以，到这里就很明显了，在用户选中菜单选项后，通过ID判断选择执行什么功能。



往上来到01001EDC,看到sub eax,211，将eax的值减到211，然后下面又有各种跳转指令，同时还有sub dec，都是在对eax进行减法运算的，不难猜出，这里应该是跟switch有关的，但是就是为什么呢？而且减来减去是为什么呢？我们可以暂时继续往上看，来到 1001DC7



来到这里，看到OllyDbg的注释WM\_TIMER WM\_SYSCOMMAND WM\_COMMAND,相信很多人应该就会明白了，一个类似于WinProc的窗口处理函数，然后switch对消息的类型进行判断然后执行相关的语句。而我们这里，就是WM\_COMMAND消息了。此处将ecx eax比较，即将eax与210比较，（210就是扫雷英雄榜的ID），是的话，下面不跳，不是则跳。到这里，这个eax很明显保存的应该就是相应菜单的id了。

回到我们刚刚的地方，sub eax,211

这里的Switch格式有点儿意思,没接触过汇编的人可能不明白,我简单解释一下.

(数值全部是16进制的表示方法,不再赘述)

在这段代码中,他用菜单项的ID251(也就是存在EAX寄存器里的值),去减211,得到40,然后下面一行代码JE就是判断刚才的结果(40)是否等于0,如果等于,就执行xxx代码

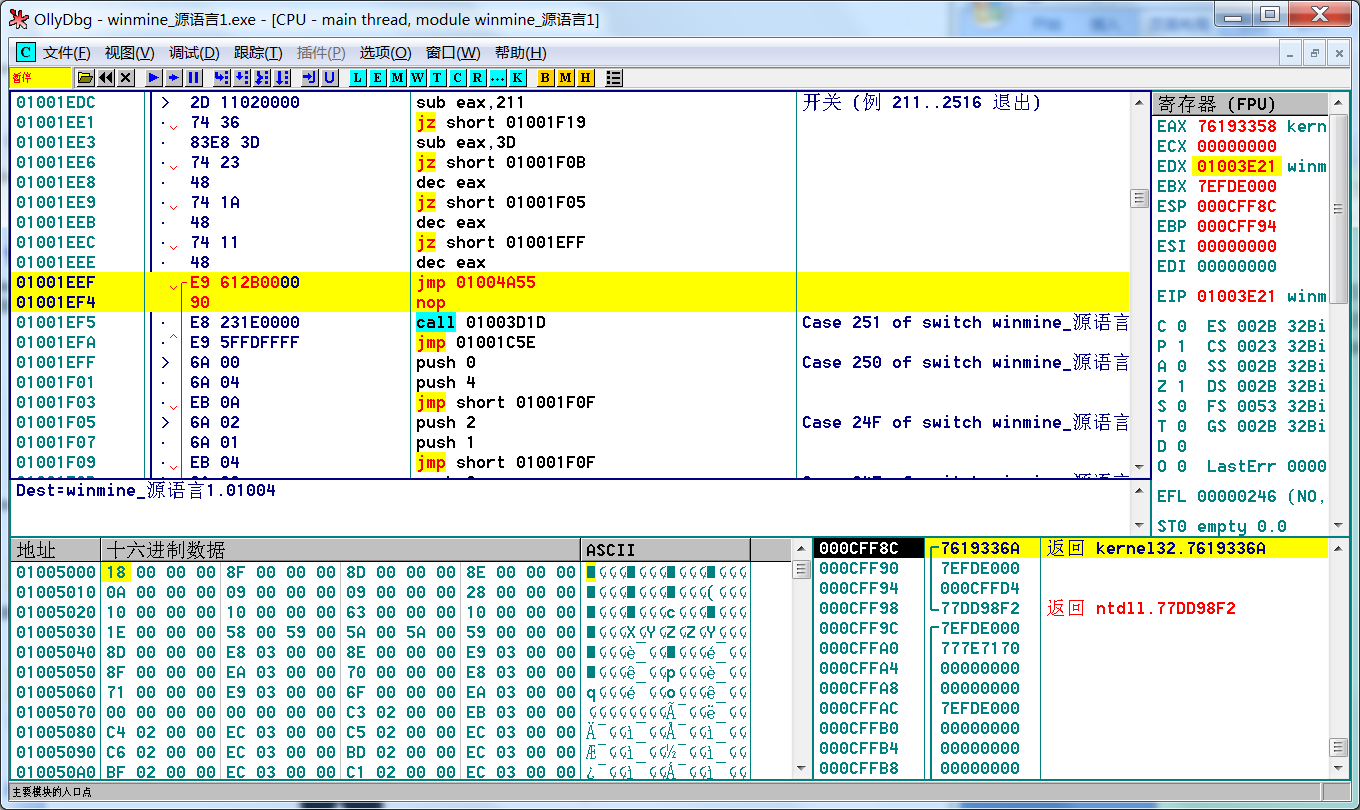
那么这句话的意思是什么呢,我们可以换一个菜单项ID来想,如果我点击的是一个ID为211的菜单,那么这里211-211,是不是就等于0了?那么下一行的JE成功跳转,执行对应的函数

这样就好理解了,他之所以用251-211,是因为有一个ID为211的菜单项,所以要判断.

而再往下的就很好理解了，sub eax,3D 再结合上面的sub，也就是eax-211-3D,以及下面的dec eax，一个菜单下，可以有很多选项，而这些选项也有对应的编号id，所以这里进行的就是这样的判断。先判断比较大的id，然后再一个一个的减下来，逐个逐个比较。

此时，如果我们要增加代码的话，应该就在这附近增加了。而这里明显没有空间让我们增加代码，考虑加一个内嵌补丁，也就是找一块空的地方，然后再写代码，让程序先跳到那里，执行完相关的指令后再跳回去。

刚才咱们增加的过关选项id为594，十六进制也就是252，那个“关于扫雷”是251，所以按照刚刚的说法，只要在他上面添加就可以了。所以来到01001EEF，将此处的指令改为jmp 01004A55,(01004A55是刚刚找到一块地方，此处都是空的，填充为00，为了页对齐。),也可以往后一些，都可以。



然后返回到 01004A5A,准备完成刚才的思路

先判断点击的菜单ID-251是否=0,若等于就跳回去(也就是上图JMP下面的CALL)

JE 01001EF5

若不等于,就再让EAX-1

DEC EAX

然后继续判断是否等于0,若等于,说明点击的是我们新添加的菜单ID,让他先跳到另一个空白处准备写代码。如果不等于，直接让他跳走。所以再来一句

JNZ 010021A9

这里的010021A9是上面switch中判断不符最后跳到的地方。此时我们就可以在下面愉快的写我们想要的指令了。

在这里，既然要过关嘛，肯定要把所有的不是雷的都点开，是雷的不去碰他，然后就过关了。

但，我们要怎么找到保存雷的地方在哪呢？找到之后，要怎么去判断，要怎么去“点开”呢？

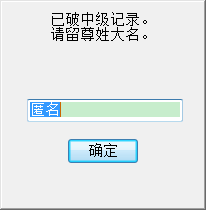
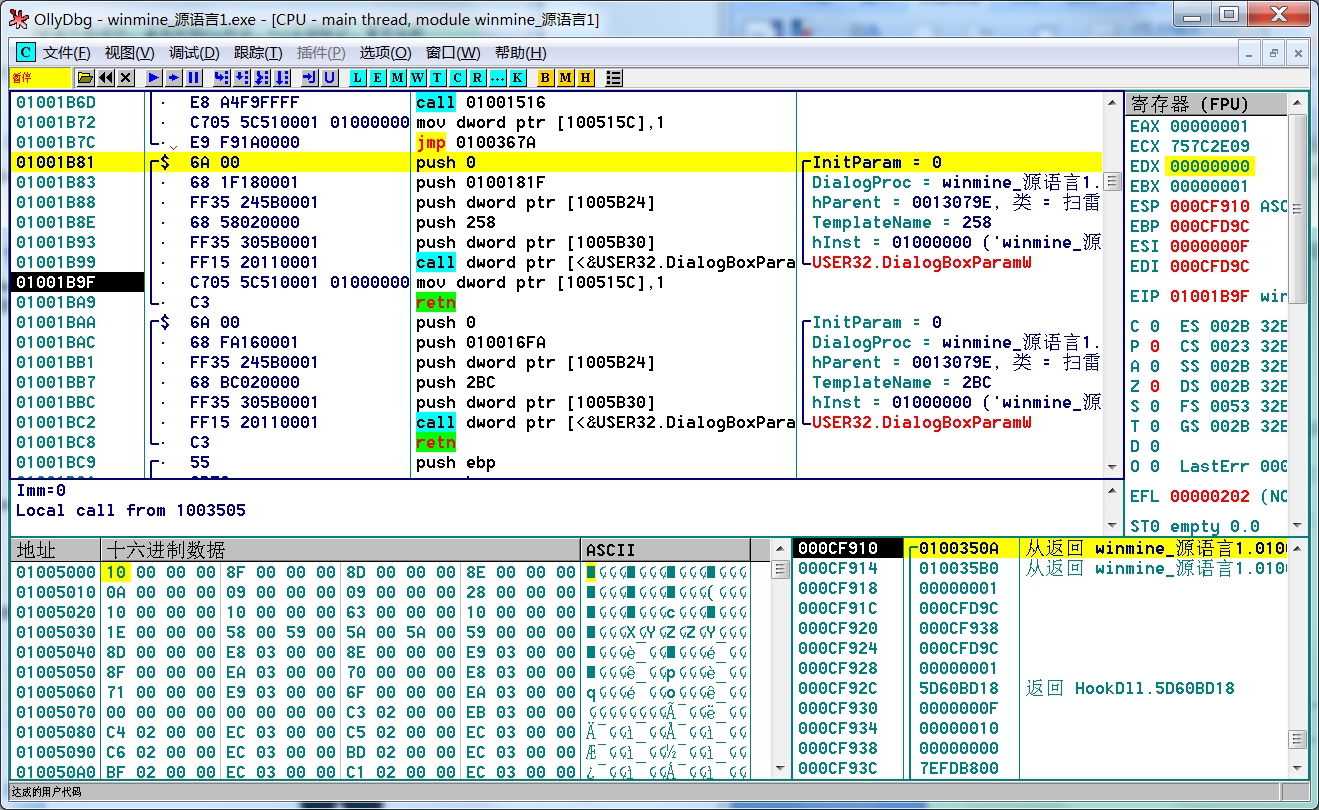
我们是不是要找到一个地方，然后下个断点，一点击格子就断下。这样的话，断点往后的代码应该就是处理的了。

这里我们可以模仿刚刚的做法，先玩一次，破个记录先（为了往上跟踪代码）。

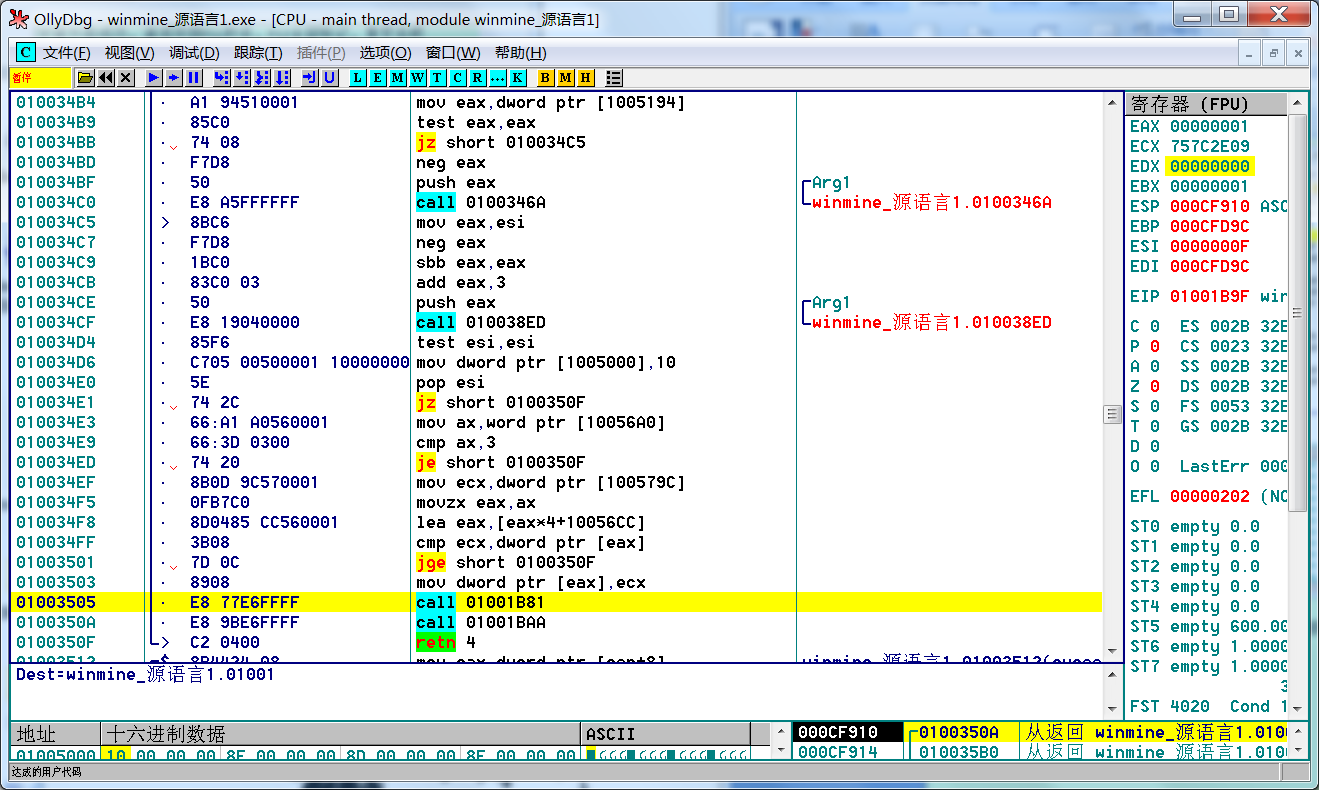
如果我写代码我会这样

破纪录->是否破纪录->游戏是否结束->点开了一个格->点击了一个格

所以我们就这样往上试试。

破记录后，出现了一个对话框，让你输入名称，然后确定。此处继续暂停，alt+f9,确定，然后就在01001B9F断下了。

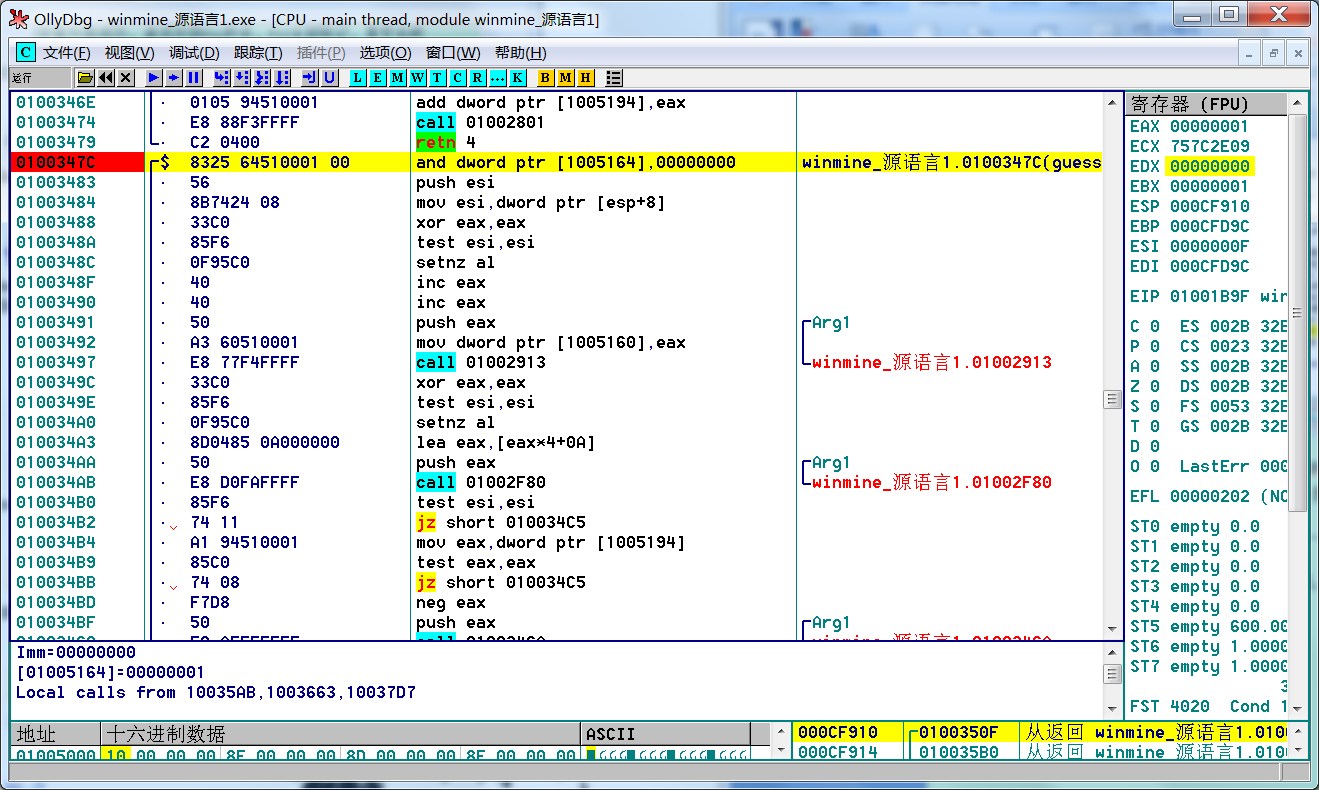
上一句语句call出了一个Dialog,也就是刚刚的那个窗口。往上看，01001B81处，跳转来自1003505，嗯，走起。



往上分析。上面有一个mov，还有一个jge，还有一个cmp，作用大致是，将此次通关时间与最高纪录时间比较，然后判断是不是破纪录了，是不跳，不是跳。

继续往上看。来到0100347C，在这里下一个断点，然后分析一下。

继续运行程序，新游戏。随便点击一个格。没有任何反应，再点，还是没反应，再点，还是没反应，所以此处应该就还不是关键的地方。



此处跳转来自三个地方，分别下断点，然后跟踪看看。

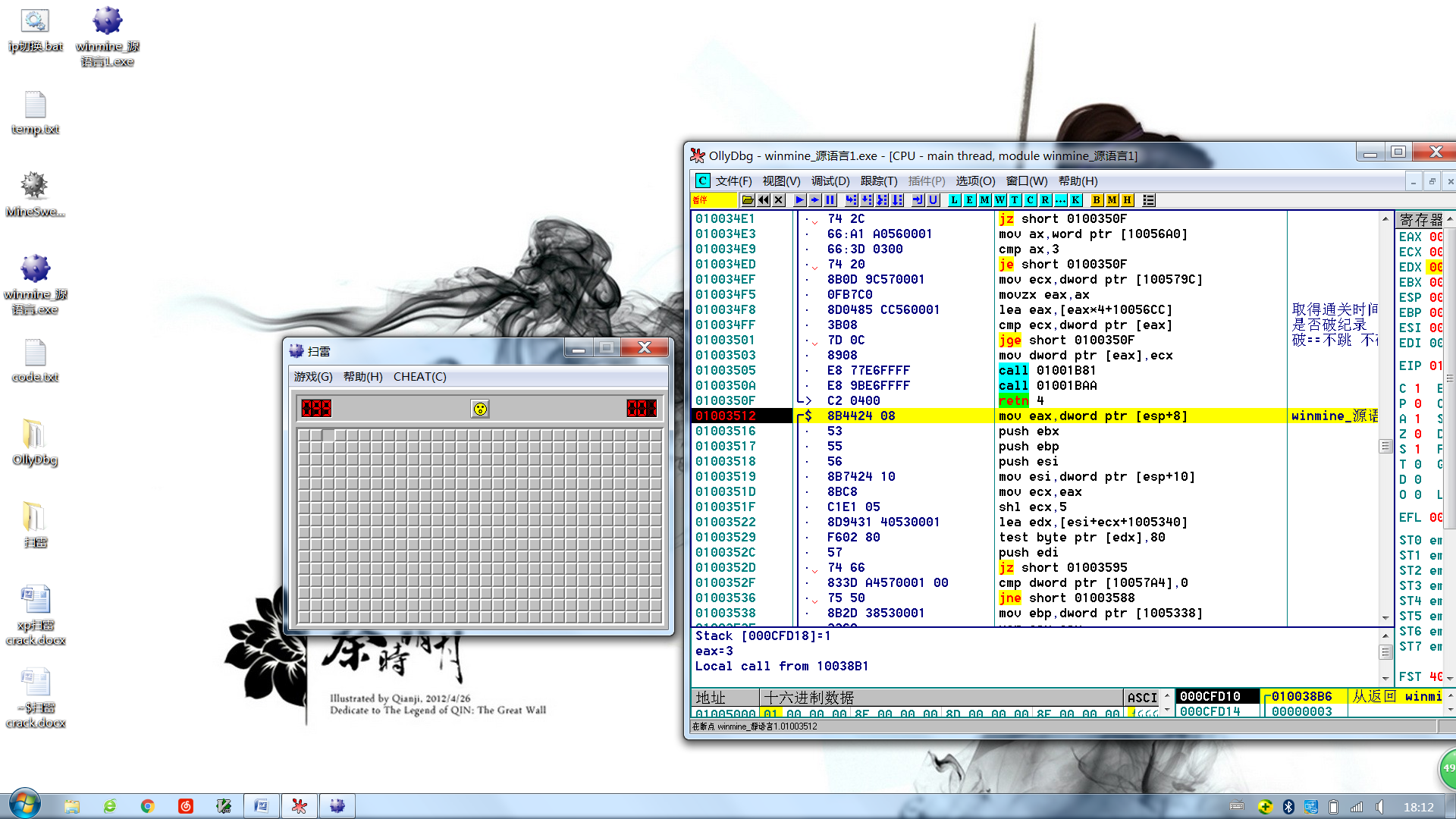
分别来到这三个地方，都有一个共通点

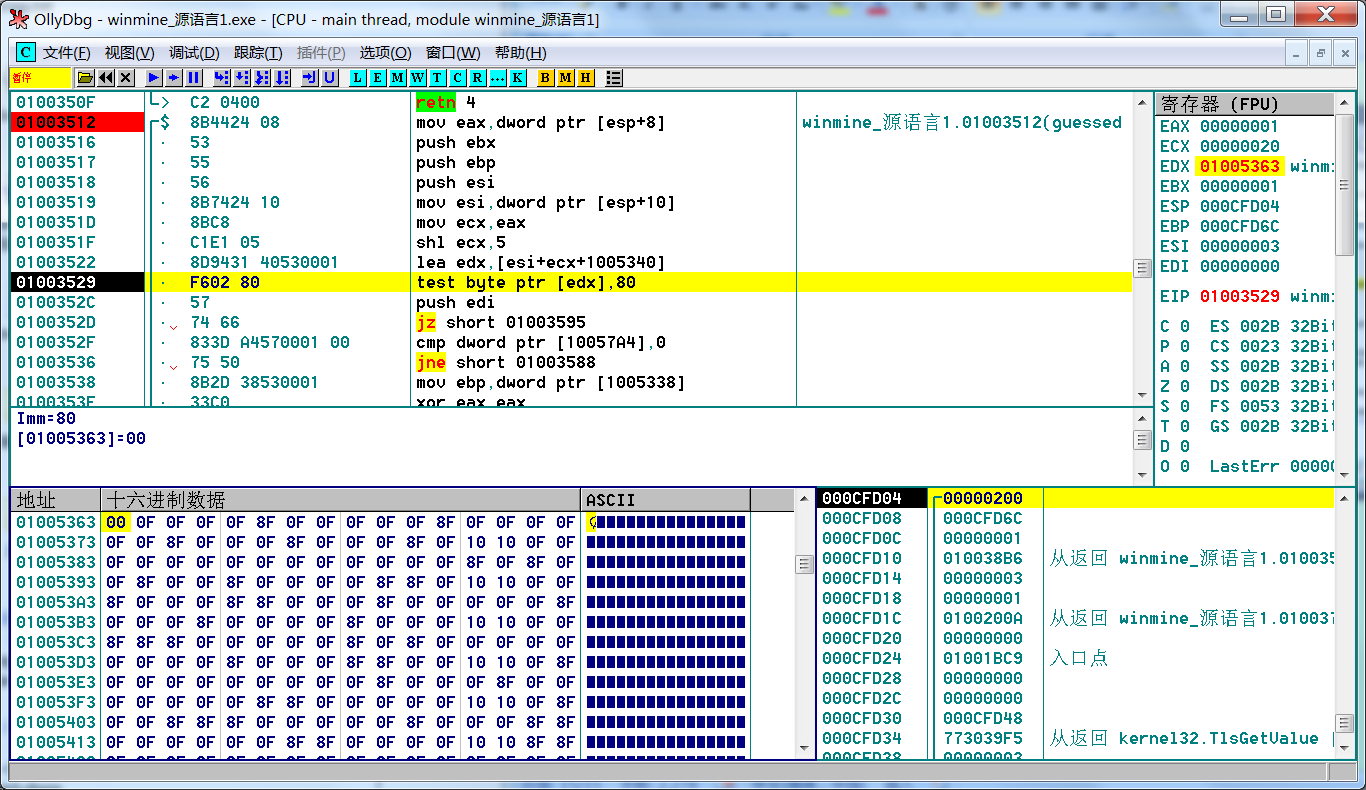
Jge….

Push 1

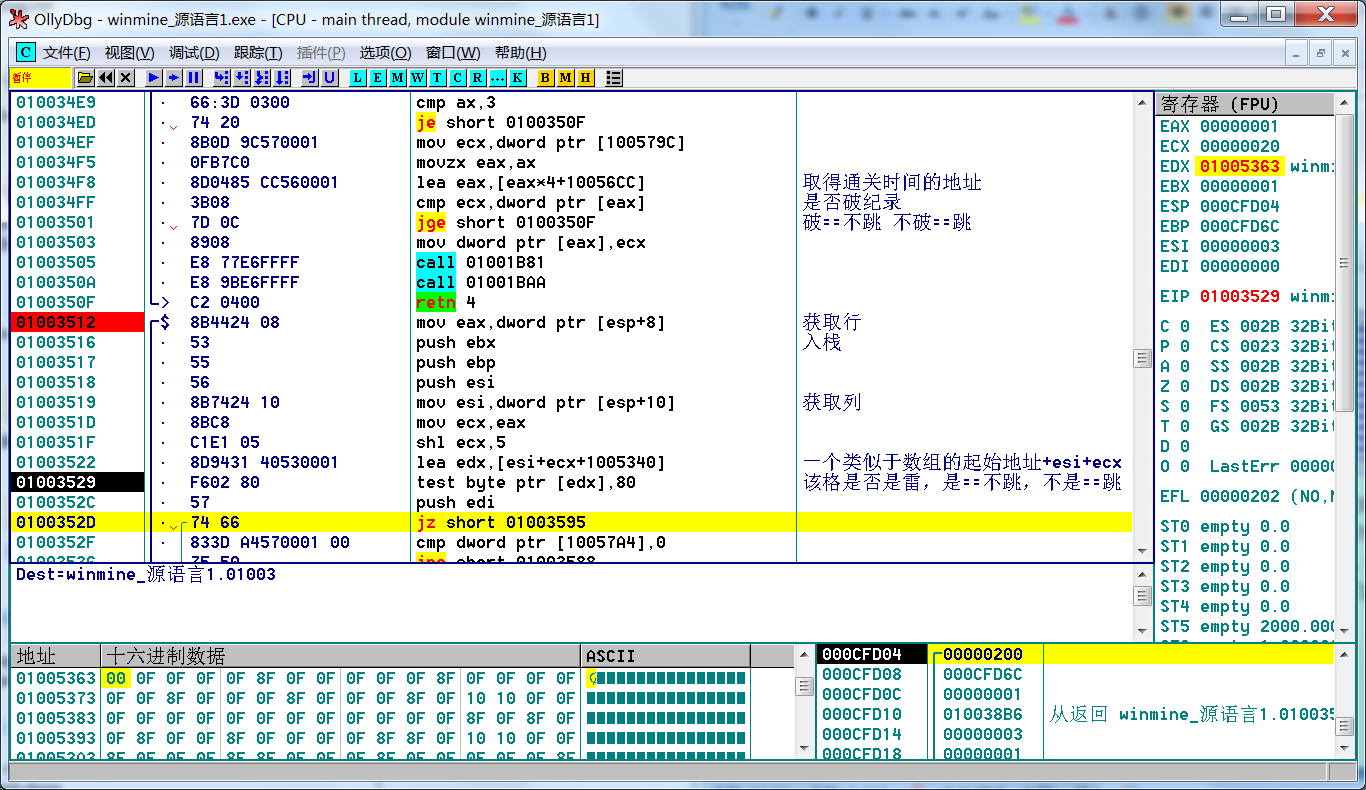
Call 0100347c

明显，刚刚那里应该就是一个函数，进行一些结束游戏的工作，判断是否破纪录等等。在这三个地方分别往上，直到最上面，然后分别下断点。0100374F 010035B7 01003512。然后再次点击，发现在01003512处断下，取消其他的断点，一步一步跟进分析。

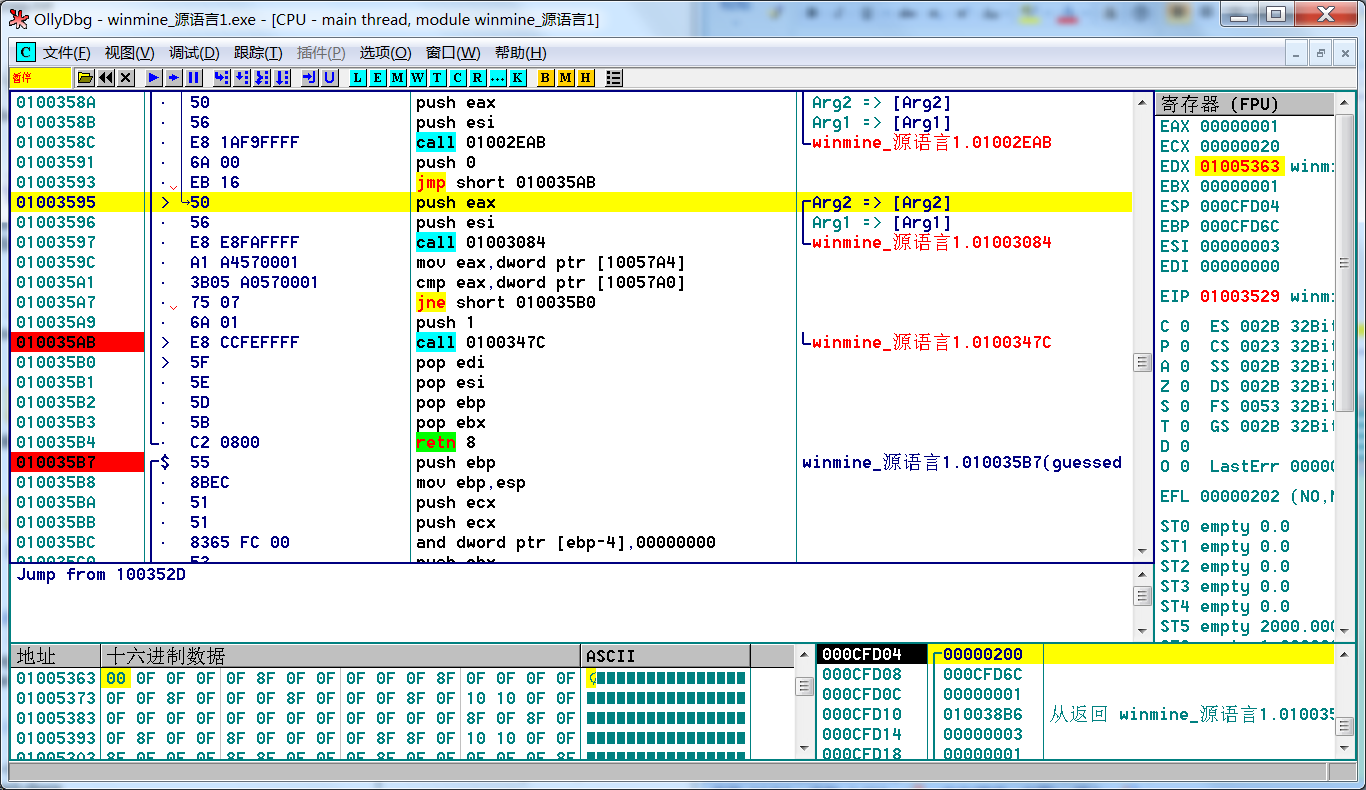




来到01003529，然后查询edx数据，发现在往后的地方都是很有规律的，00 0F 8F，明显，这些地方储存的是雷。右边寄存器eax的值为1，esi的值为3，刚好是刚才点击的行与列，也就是说，而mov ecx,eax shl ecx,5将行的值左移5位。行优先。将该格的数据与80比较，判断是否是雷。（8x表示是雷，可以自己点开后分析4x表示已经点开了这个格）。如果是雷，往下执行，不是则跳。

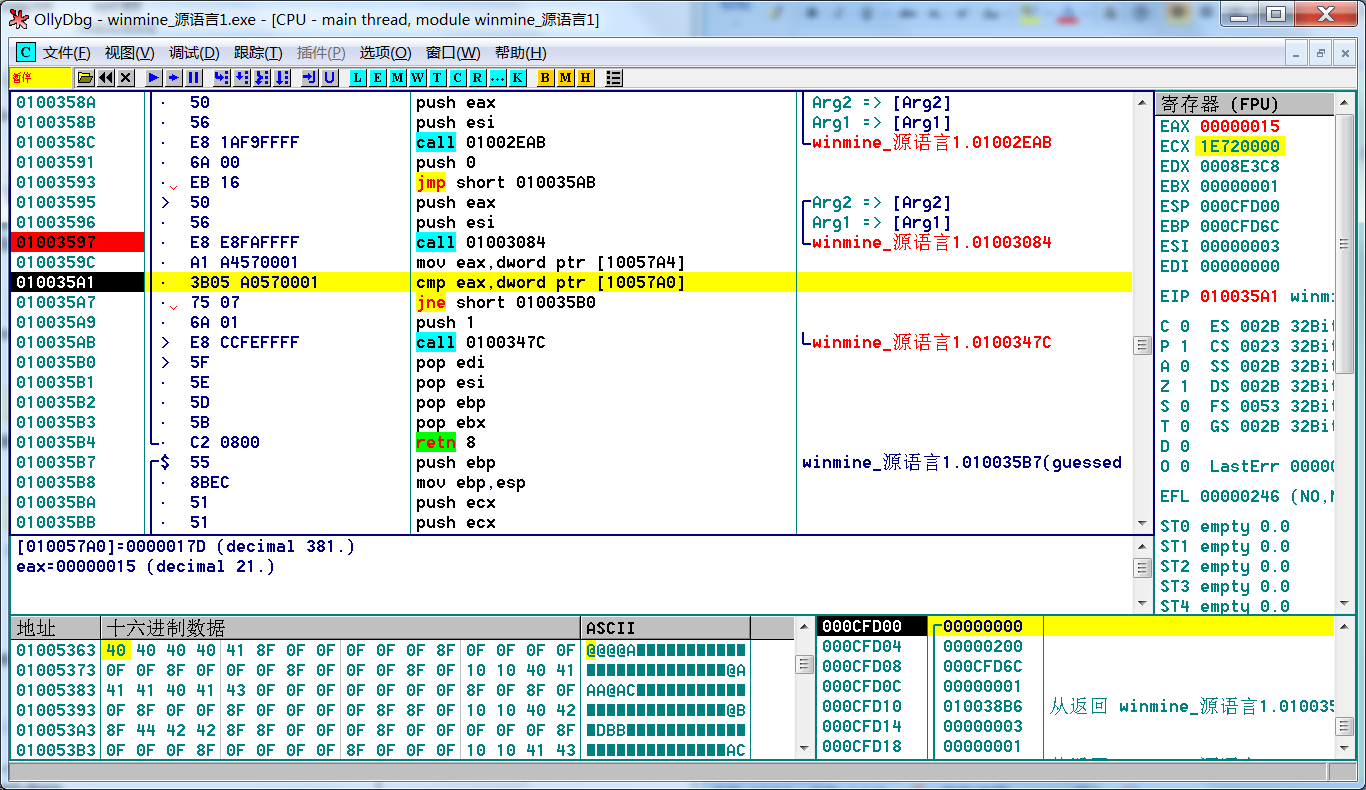


往下跟，来到不是雷的地方，



将行与列都压入栈，当作函数01003084的参数。下断点与这个call，发现这个call的作用就是判断点下这个格后，是否需要展开周围的格，需要则展开，不需要则不展开。

下断点于010035A1，发现这里的比较是比较eax（点开了的个数）与不是雷的个数比较，判断是否结束游戏。





15是当前点开了格数，下面170（十六进制）是总的不是雷的数目。

如果相等，则结束游戏---🡪push 1 call 0100347C

不想等的话，弹出恢复数据，然后返回。

至此，分析就差不多了，可以开始写代码了。假设有nxm格，可以这样子有，将每一个都遍历，然后如果是雷，则跳，不点开，进入下一个格，如果已经点开了格（4x），则跳进入下一个格。如果该行结束，则进入下一行。如果已经结束了，则进行结束工作。

在数据段里面找到两个可以储存数据的地方，存储一下当前的行，当前的列

这里使用[1005BE0]与[1005BF0].

以下代码只需要将loop，loop1，loop2，改为相应的地址即可。没有采取伪指令。

jz 01001EF5

dec eax

jnz 010021A9

//初始化工作

pushad 将所有的寄存器压入栈，保存数据

mov dword ptr [1005BE0],1 将行初始化为1

mov dword ptr [1005BF0],1 将列初始化为1

//开始

mov eax,dword ptr [1005BE0] loop,将当前行赋给eax

mov esi, dword ptr [1005BF0] 将当前列赋给esi

mov ecx,eax

shl ecx,5

lea edx, [esi+ecx+1005340]

test byte ptr [edx],40 判断是否已经打开了当前的这个格

jne loop1 是的话，跳到loop1

test byte ptr [edx],80 判断是否是雷

jne loop1 是的话也跳到loop1

push eax

push esi

call 01003084 对点开这个格进行操作，是否打开周围的格

mov eax,dword ptr [10057A4]

cmp eax,dword ptr [10057A0] 是否过关

jne loop1 不是跳到loop1

push 1 是，调用相关的函数

call 0100347C

popad 恢复刚刚的各个寄存器的数据

jmp 010021A9 跳回刚才点击了菜单选项的switch的结束的地方

cmp esi,dword ptr [1005334] loop1，判断是否进入下一行，【1005334】保存了当前难度下有多少列

je loop2 是，进入下一行，来到loop2

inc dword ptr [1005BF0] 不是，将列数加1，继续遍历

jmp loop 继续遍历

inc dword ptr [1005BE0] loop2，将行数加1

mov dword ptr [1005BF0],1 将列数重置为1

jmp loop 继续遍历

至于如何寻找当前难度下有多少列，在这里偷懒，直接使用ce查找，得到两个地址，取其中一个。

有兴趣使用ollydbg跟踪的可以看一下01002C18附近的代码。

下面贴上一张改好的代码的截图，以及源程序与改好的程序。

