感谢RK大神，权当探索一下汇编了。

说是从零开始，因为以前没接触过汇编、也没接触过修改器设计，逆向也只会一点点的IDA使用，所以主要是在RK的基础上去学。我希望这个帖子在记录了我的修改学习的基础上，也能为各位想要学习311修改的同学们做一个小科普。

我使用的工具主要有：

1.IDA Pro 用于逆向311的脱壳版exe（就是100多MB那个），分析汇编代码。

2.Hex WorkShop：用于改.exe。在测试阶段，直接改exe比修改内存更方便

3.Windows自带计算器：挺好用的，其中的“程序员”选项卡可以很方便地进行16进制运算，我现在还是手写机器码的状态（因为没找到好用的汇编器），算跳转指令的offset很管用

4.Intel Volume Set：英特尔官方的汇编指南，非常齐全，应有尽有，自学汇编必备工具书。大概4000多页，不过不用害怕，用到什么指令直接在目录里边定位就好了，80%以上的内容你永远不会用到。

5.Cheat Engine：喜羊羊与RK都在用的动态调试神器，可以动态查看运行时的内存。不过我暂时还没太掌握用法，因为目前的修改都比较简单。

6.RK的内存研究资料。这份资料已经涵盖了311绝大多数内存的研究，如果没有这份资料我是不可能做出来任何东西的。资料的链接会发在这层楼回复里。

## 1.去掉战法·乱战的地形限制

我们要怎么改出来这个效果呢？首先我们打开RK的内存研究资料，搜索“乱战”

amazing！居然能搜到东西！搜索结果如下：

005AE806: 乱战(15)(对应普攻与战法)

005AE800: 有效地形代号(5=森)

我们按照这个提示，进入IDA搜索相关代码，找5AE800附近的代码， 会发现代码为：

005AE7FE cmp dl, 5

005AE801 jnz short loc\_5AE814

005AE803 mov eax, [esi]

.005AE805 push 0Fh

005AE807 mov ecx, esi

005AE809 call dword ptr [eax+58h]

005AE80C test eax, eax

005AE80E jnz loc\_5AE760

这些就是与乱战有关的代码了。其中，每一行都是一条汇编指令，005AE8FE这些是指令的地址，eax ecx esi这些是寄存器，cmp jnz mov这些则是指令的名字，而short loc\_5AE814指的则是跳转的位置（也就是说，这一行会在一定条件下跳转到5AE814，相当于801和814之间的指令就被跳过了）。

看不懂我在说什么怎么办？翻Intel Volume Set啊！在研究后，我们发现这些代码讲的是：先把地形dl和5(森林)比较，如果不是森林的话直接跳出乱战的判定，而之后0Fh则是乱战的特技编号(h指16进制数，F为16进制下的5，对应RK资料里的那个15)，如果特技不是乱战，也跳出。

那么，如果要去除乱战的地形限制，我们只要把“不是森林就跳出”这个逻辑删掉就可以了。在Intel汇编中，有一个指令叫"nop"，指的是no operation（无操作）。我们把跳转指令改成nop，就相当于删掉了跳转的逻辑。

那我们怎么把这个修改写到exe里呢？我们首先要知道的是，每一条汇编语句对应1-6个字节(byte)，每个字节为8个bit，或者2个16进制数。（比如说，nop的指令为90。‘90‘就是一个byte）。在IDA中，我们可以看到每一条汇编指令对应的位置和它们的byte。比如说，我们要改的那条跳转指令的字节是0F 85 4C FF FF FF。我们只要将每一个字节都改成nop(90)就好了。

另外我们还要注意的是，在IDA（和RK的资料中）看到的5AE800，其实在EXE里边的位置是1AE800，也就是两者之间的第一位要-4，这里涉及更深入的一些知识，只需要知道这个结论就好。

所以，打开Hex Workshop，找到1AE814（可以用工具栏里边直接跳转，不需要拉滚动条），将0F 85 4C FF FF FF改为90 90 90 90 90 90。保存。Hex Workshop会提示你要不要备份原来的.exe。我的建议是在你定期备份修改成功的版本，之后如果炸了就滚回能用的版本就好。对于新手来说，把游戏改崩是常有的事情，peace。

改完之后，你可能会问了，这样一条指令不就变成6条指令了么（6个NOP）？是的，为了加快CPU运行，Intel X86汇编采用的是不定长的指令方案。也就是说，指令之间的长度是不相同的。比如简单的NOP只需要1个字节，但是jnz需要6个。因此，在未来修改的时候，也要注意指令之间的对齐。

好了，这个就是入门级别的修改了，基本属于小打小闹级别，真正创造特技还需要额外加入代码段，我们之后再讲。

## 将特技·疾驰改为“移动力大于敌方时，使地方伪报”

和上一个同属于牛刀小试，甚至比上一个还要简单，因为这里不涉及改指令，只要改数据就好了。

起手式依然是一样的，打开RK的内存研究资料，搜索“疾驰”

（RK的内存研究资料一共4个文档，建议把他们放在一起，这样搜索起来比较方便）

我们搜到的结果：

疾驰

00593B78: 疾驰(49)对应骑兵战法

00593BDE: 使敌军混乱(1) (2=伪报)

00593B98: 8a 8f c9 00 00 00 - mov cl,[edi+000000c9] 取我方攻击(参见部队内存)，cl改成ecx则可以比钱、粮、兵力等

00593B9E: 3a 8b c9 00 00 00 - cmp cl,[ebx+000000c9] 取敌方攻击(参见部队内存)，cl改成ecx则可以比钱、粮、兵力等

wow，已经把核心代码最重要的部分翻译出来，并且给了修改提示了！那我们就开始改咯

首先，我们把混乱改成伪报。RK的代码里边已经明确讲了在593BDE的位置有一个1，代表混乱，而且还讲了如果改成2就是伪报。那我们去找一找咯。在IDA的"Hex View"中按下快捷键"G"（跳转到地址），输入593BDE，发现定位出来的结果为6A 01(光标停在01前)。这就是混乱的位置了。我们改成6A 02，就可以变成伪报了。如果我们改成6A 00，则是使敌方正常... emmmm居然帮敌方镇静，怎么看都是负面特技，算了吧。

接下来我们改比较移动力。在这里，我们继续充分利用RK的资料。首先我们不知道edi + 000000c9是什么，但是RK提示了“参见部队内存”，我们在RK的资料里搜索一下“部队”

搜索结果如下：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[部队 共0xF4个字节 起始地址0736B088]

+00 4 byte (18 CC 79 00) 部队相关函数所在地址

+04 4 byte ??

+08 4 byte 类型(0=战斗,1=运输)

+0C 4 byte 主将ID

+10 4 byte 副将1ID

+14 4 byte 副将2ID

+18 2 byte 兵力

+1A 1 byte 气力

+1B 1 byte 空闲

+1C 4 byte 金钱

+20 4 byte 兵粮

+24 4 byte 状态(0=正常,1=混乱,2=伪报)

...(省略若干行，请下载资料自行查看)

+C9 1 byte 部队显示攻击力

+CA 1 byte 部队显示防御力

+CB 1 byte 部队显示建设力

+CC 4 byte 部队显示移动力

+D0 24byte 部队6个兵科适性

Amazing！按照这份资料，+C9正好就是部队的攻击力，和疾驰的技能一下就对应起来了！如果我们把C9改成别的东西，会不会管用呢？我们用Hex Workshop试着把00593B98和00593B9E里边的C9改成18，进行测试。结论是，当兵力大于敌方时，一定会眩晕，小于敌方时，一定不会眩晕。

Amazing！这不就成功了吗！我们再把C9改成CC，载入游戏，测试仍然通过，标志着这个改动成功了。

同样的，还可以设计出疾驰是气力/适性/兵粮等等高于对方时眩晕，而这些只需要相应的改掉C9就好了，是不是很神奇呢~

好啦，前两个会比较简单，讲的也比较详细，大家如果想入门的话， 可以参考我说的过程~哪里不懂的话在评论里问就好了~

## 【第三弹】特技·风水新增效果：寻宝概率提升300%（上篇）

在这一楼中，我们做一些相对更“大”的修改，而不是只满足于改改参数或者覆盖掉某一条指令。这一次我们要给风水特技新增一个效果：寻宝概率+300%。当然，跟着看完整个修改之后，你会发现，并不一定要并入风水特技，也可以单独拎出来作为新特技。同样，300%的概率也可以调整。

“上篇”的意思是，在这一弹中，我们先进行【全局性】的修改，也就是说，我们不管风水特技，而是把所有武将的寻宝概率都提升300%。在实际修改过程中，我个人也建议大家这么做，因为这样更容易验证、并且在出错的时候更容易定位到错误来源。

要注意的是，在这一弹中，RK的资料并没有直接指明修改方法，因此会更困难一些，需要自行阅读汇编代码，然后猜分析其中的逻辑。好消息是，我都替大家做完了，跟着我一步一步来就好，不用担心

虽然RK没直接告诉我们怎么改，但是我们的起手式依然是在RK的资料中搜索以“宝物”为关键词，其中的一个结果是：

005D3F1E: 发现宝物加能力经验(1)

005D3F20: 加政治(3)经验

005D3F0F: 功绩加(50)

005D3F4F: 增加技巧点的2倍(2)

看起来这一部分讲的是发现宝物之后的操作，加功勋和经验什么的。那么如果有哪个地方调用了这个指令，那么在那附近就应该是发现宝物的逻辑了。我们进入IDA Pro，用“G”快捷键定位到005D3F1E。在汇编视图中按下快捷键“F5”，你就会发现汇编代码变成了C代码！就问你A不Amazing！这就是IDA一定要用Pro的原因。F5快捷键实在太强大了。当然，这个并不是光荣在做游戏时候的原版代码，而是根据汇编代码反向“猜”出来的。

在F5中，我们可以看到，以上的这些逻辑在一个叫做“sub\_5D3D90”的函数中。sub\_5D3D90的意思是，函数的第一条指令的首地址是5D3D90。我们右键这个函数名，在菜单中选择“Jump to XREF”，其中XREF指的就是所有调用了这个函数的地方。

我们可以看到，结果只有一个，在函数"sub\_5D5970"中。调用位置上下文的C逆向代码如下

v9 = (unsigned \_\_int8 \*)sub\_5D51D0(&v30);

signed int v10; // eax

if ( !sub\_47A630(v9) )

goto LABEL\_34;

v10 = (61 - v9[60]) / 20;

if ( v10 < 1 )

v10 = 1;

if ( !sub\_4721D0(v10) )

LABEL\_34:

v9 = 0;

if ( sub\_47A630(v9) )

{

sub\_5D3D90(v9, v3, v2); // 这个就是“发现宝物结算”的函数

}

是不是看的一脸懵逼、一头雾水没事，我们一点一点分析。首先，已知v9的数据类型是一个指针，想要进入宝物结算，必须通过sub\_47A630(v9)这个if验证。在RK的资料中搜索47A630，发现这个函数是用来判断指针是否有效的。在这里，只要v9不等于0，就是有效的。

然后我们会发现，就在上一行就是v9 = 0...这不是挑事吗！这行代码位于if ( !sub\_4721D0(v10) )所以我们要保证这个if不能生效。那么if中的条件说的是什么呢？在RK的资料中搜索4721D0，可以发现这个函数是一个[v10]%的随机函数。也就是，有[v10%]的概率会返回1，否则返回0。

我们再来看v0。上边的代码为：

v10 = (61 - v9[60]) / 20;

if ( v10 < 1 )

v10 = 1;

这就是v10，也就是的概率公式了。也就是，有v10 = ((61 - v9[60]) / 20)%的概率搜到宝物。但是还有一个问题，v9[60]是啥？我们猜想v9是一个宝物指针，然后在RK的资料里搜索宝物，发现：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[宝物 共0x54个字节 起始地址072790D4]

+00 4 byte (68 C5 79 00) 宝物相关函数所在地址

+38 4 byte 宝物类型

+3C 4 byte 宝物价值

+40 4 byte 所属武将

+44 4 byte 所在城市

+48 4 byte 宝物状态

+4C 4 byte CG

+50 4 byte ??

其中+3C，也就是10进制的+60，是宝物的价值。所以我们可以得出概率公式：v10 = (61 - 宝物价值) / 20。

了解了公式之后看一下对应的汇编代码。毕竟C代码是“猜”出来的，可能不够准：

005D5B31 call sub\_5D51D0 // 随机返回一个宝物

005D5B36 mov edi, eax // edi中存的是宝物指针

...

005D5B49~5D5B66 eax = (61-宝物价值) / 20 (算术结果小于1时设为1)

// 值得注意的是，上边的代码段用了神奇的位运算+乘法代替了除法，如果要改参数的话要仔细计算一下

// 不在意这一点点性能、或者想改掉整个公式的话，直接写一个div，然后其他地方nop掉就好了。

// 这里能看出311真是很老的一款游戏了。对于最新版本编译器来说，一般不会把这么简单的指令做成这个鬼样子的。

005D5B6C call sub\_4721D0 // 调用随机函数，有[eax]%的概率搜到宝

005D5B71 add esp, 4 // 宝物价值为1的概率:3%，2~21:2%，价值>21:1%

005D5B74 test eax, eax

005D5B76 jnz short loc\_5D5B7C // 如果没有命中以上概率，跳出到搜金币的过程

...

005D5B8C call sub\_5D3D90 // 进入宝物结算过程（加经验、功绩，把宝物指针赋值给君主指针等）

而至于修改方法，如果把5D5B76的jnz改为jmp的话，可以无条件搜宝物，也就是每一次都搜的出。如果要修改概率的话，可以修改5D5B49-5D5B66部分公式参数（比如把20改成10就是概率翻倍）也可以修改5D5B6C的随机函数（调用自定义上限的随机函数，然后改上限）。代码复杂度差不多，效果是等价的。

我的改法（改掉/20的参数20）：

源代码：

005D5B45 0F B6 57 3C movzx edx, byte ptr [edi+3Ch] // edx = 宝物价值

005D5B49 B9 3D 00 00 00 mov ecx, 3Dh // ecx = 61

005D5B4E 2B CA sub ecx, edx // ecx = 61 - edx

005D5B50 B8 67 66 66 66 mov eax, 66666667h // 以下是杂技似的/20

005D5B55 F7 E9 imul ecx

005D5B57 C1 FA 03 sar edx, 3

005D5B5A 8B C2 mov eax, edx

005D5B5C C1 E8 1F shr eax, 1Fh

005D5B5F 03 C2 add eax, edx // 杂技结束，eax = (61 - edx) / 20

源代码把一个简单的除法弄得乱七八糟的，因为这里涉及了编译器的优化。我们无视这些优化，直接改：

改后：

005D5B45 0F B6 57 3C movzx edx, byte ptr [edi+3Ch] // edx = 宝物价值，不变

005D5B49 b8 3d 00 00 00 mov eax, 3Dh // eax = 61

005D5B4E 2b c2 sub eax, edx // eax = eax - edx

005D5B50 b9 01 00 00 00 mov ecx, 1h // 20改为1，搜索概率为原先20倍。本数值可调整

005D5B55 99 cdq

005D5B56 f7 f9 idiv ecx // eax = eax / 参数(1)

005D5B57- 005D5B60 90 nop

以上也可以拉出去一个函数再跳回来，但是由于这一段代码没有被引用，因此直接就地改了。可以看出，我们先把概率调高20倍（而不是3倍），这样可以很方便地看出是否有没有改成功。用Hex Workshop实装后，发现寻宝概率大约为50%，验证了修改的正确性。

接下来我们只要把20改成5，就实现了寻宝率+300%了。

至于如何设置成某个特技的专属效果，下一次继续讲。

## 【第四弹】设置特技专有效果

上一次我们已经把寻宝概率+300%设置为了全局效果，接下来我们将其设置为特技风水的转有效果。

首先，我们先要找到武将的指针位置。因为只有武将指针确定了，才能据此判定是否是指定特技。我们回到上一弹中最开始就提到的“发现宝物后结算”。由于发现宝物后要给武将加功绩、经验，因此这里是一定有武将指针的，我们只需要追踪一下它一路传进来的路径即可。

在结算中，有一行代码为调用sub\_4A70D0(a2, 3, 1, 1)。在RK的资料中查询4A70D0，可以发现第一个参数a2就是武将指针。结算代码所在函数为int sub\_5D3D90(void \*lp, void \*a2, void \*a3)，其中a2为第二个参数。我们根据XREF回到调用这个函数的地方（也是我们改动了的地方），发现函数的调用方法为sub\_5D3D90(v9, v3, v2)。

进一步的，在IDA中我们可以发现v3为esi寄存器存放的内容，也就是说，esi中就是我们要找的武将指针。事实上在很多情况下，esi存放的都是武将指针。

接下来，我们在RK的资料里搜索判定特技的代码模板，然后照葫芦画瓢。判断是否有指定特技的模板如下：

6a [\*\*] - push [\*\*] [\*\*]=待判断的特技ID

8b ce - mov ecx,esi

e8 db fa bd ff - call 004890f0

85 c0 - test eax,eax

0f 84 [X(4bytes)] - je [Y] 如果没有这个特技，那么跳转到[Y]地址，[X]=（[Y]-当前指令的下一条指令的首地址）的小端序（如，当前指令为00000000，当前指令的下一条为00000005，要跳转0000000f，那么X为0000000f-00000005=0000000A，对应的小端序为0A000000）

在我们这里，特技·风水的id为97，在16进制里是61。我们要实现的效果是：如果特技为风水，那么跳到新的公式，否则还沿用原有公式。我们写出下面的汇编码：

005D5B45 0F B6 57 3C movzx edx, byte ptr [edi+3Ch] // edx = 宝物价值，不变

005D5B49 b8 3d 00 00 00 mov eax, 3Dh // eax = 61

005D5B4E 2b c2 sub eax, edx // eax = eax – edx

// 以下四行判断是否有风水特技，有的话，标志寄存器置为1

\*005D5B50 6a 61 push 061h

\*005D5B52 8b ce mov ecx, esi

\*005D5B54 E8 ?? ?? ?? ?? call 004890f0

\*005D5B59 85 C0 test eax, eax

// 判断标志寄存器。如果有风水特技，则jne (jump if not equal to 0)成功跳转到被除数为1

\*005D5B5B 0f 85 ?? ?? ?? ?? jne Label1

\*005D5B61 b9 20 00 00 00 mov ecx, 20h

\*005D5B66 e9 ?? ?? ?? ?? jmp Label2

Label1:

\*005D5B6A b9 01 00 00 00 mov ecx, 1h // 20改为1，搜索概率为原先20倍

Label2:

\*005D5B6F 99 cdq

\*005D5B70 f7 f9 idiv ecx // eax = eax / 参数(1)

这里所有的?? ?? 都是跳转的偏移量。算法就是我上边写的那个公式。实际上我懒得算了，因为我们会发现，写完之后已经最后一条指令到了005D5B70，已经超出了原来的代码范围005D5B60，因此会覆盖掉原有的无辜代码。换言之，原来的内存空间已经容不下我们新增的逻辑了。因此，我们需要扩展代码范围。

好消息是，在整个exe的最后边，存在大量未使用的代码区域。SIRE的做法是把所有新增的逻辑都先跳转到8Axxxx的新增代码，再跳转回原来的代码，这样就相当于“插入”了新的代码逻辑。SIRE用了8Axxxx，我们则从920000开始使用，避免和SIRE冲突。（9200000这个代码段算是RK钦定让我使用的了，如果以后修改的人多了、比如真的有10个喜羊羊的话，可以统一分配一下各个修改者使用的代码段）

新的代码为：

005D5B45 0f b6 57 3c movzx edx, byte ptr [edi+3Ch] // edx = 宝物价值，不变

\*005D5B45 e9 b6 a4 2e 00 jmp 008C0000

\*005D5B4A-4D 90 nop

// 新增代码段，位于920000

00920000 b8 3d 00 00 00 mov eax, 3Dh // eax = 61

00920005 2b c2 sub eax, edx // eax = eax - edx

00920007 b9 14 00 00 00 mov ecx, 14h // ecx = 20

0092000C 99 cdq

0092000D f7 f9 idiv ecx // eax = eax / ecx

0092000F 50 push eax // 保护eax寄存器，因为判断特技会用到eax

00920010 6a 61 push 061h // 检验是否为风水

00920012 8b ce mov ecx, esi

00920014 E8 d7 90 B6 FF call 004890f0

00920019 85 C0 test eax, eax

0092001B 58 pop eax // 恢复eax

0092001C 0f 84 06 00 00 00 je 00920028 // 不是风水的话直接跳过乘法

00920022 6b c0 14 00 00 00 imul eax,eax,14 // \*20

00920028 e9 34 5b cb ff jmp 005D5B61 // 跳回

注意在这里，我们又做了一定的改进，原先是直接/20的参数变化，精确度不是很高，我们在这里改成先除20、再做乘法。我们这里边是在00920022里边将倍率乘以20，如果要修改成+300%，就乘以把14h改成4。

此外，我们也可以乘以浮点数，更加精确。浮点数的表示方法依靠IEEE 754标准，转换方法可以使用python的struct类库：

Python 2.7.4

>>> import struct

>>> struct.pack("<f", 238.3).encode('hex')

'cd4c6e43'

但是在这里，由于基础倍率最高才3%，乘以浮点数的意义不大，直接按整数倍翻倍即可。如果想改为更精确的乘浮点数，可以参考RK关于“能吏”等特技的修改。这一部分留做练习。（教科书既视感上线）

## 【补充说明】补充一些小技巧

1.改机器码的时候最好先在IDA的Hex View里边改一下（修改和保存的快捷键都是F2），这样可以马上看到对应的汇编和C代码，可以用于验证你的机器码有没有写错。确认机器码没写错之后再用Hex workshop或者动态改内存的工具去改。

2.把机器码抄到Hex workshop或者任意需要操作机器码的时候，都要注意不要抄错了，我已经被8和B傻傻分不清楚坑了N次了...  
3.如果保存实装后出现卡死，按一下顺序检查：①机器码抄没抄错 ②在IDA里边机器码能不能顺利反汇编成汇编码和C语言代码 ③代码逻辑有没有错，比如死循环 ④被你改掉的部分汇编指令长度有没有对齐。如果没对齐的部分一定要NOP掉 ⑤被你改掉的部分代码有没有被引用。这个在IDA里比较好判断，如果一行代码被引用，会在代码前出现一个(loc\_xxxxxx或者sub\_xxxxxx)的标签。如果有这种代码，不要直接改掉他们。

## 【第五弹】一起做SIRE 1.29吧（兼CE教学）

在这一弹中，我们一起来做SIRE的新君主技（名字我还不知道叫啥）：所属势力内政设施行动力-1。在RK的设计中，君主技和本身特技是不冲突、可以并存的，但是我们这次简单一点，就用君主本身的技能判定。我们的目标效果姑且设置成，如果君主的技能是“能吏”，则所有行动力-1。

类似的，我们先开始静态修改，也就是不管君主有没有能吏，所有的内政行动力都-1。

首先我们找和行动力有关的函数。RK的资料显示，4A1820是军团消耗行动力（势力本身也是一个军团）。搜索所有的XREF，其中有-20行动力大概有一页那么多。

我们当然可以一个一个点进去，逐个阅读每一个调用了减行动力的函数都是干什么的。然而，这实在是太复杂了，毕竟静态就意味着无法调试，只能肉眼反汇编，效率极低。

所以我们这一次引入Cheat Engine（简称CE），一个为桌面游戏设计的、基于内存的动态调试。动态调试与之前的exe修改不同的是，之前我们需要手动写机器码、然后用Hex workshop导入，虽然有一些trick可以使整个过程更加简单（比如先在IDA里边改，确保结果正确），但是毕竟还是有些复杂了。使用CE，我们直接修改内存，就算修改崩了也大不了重启一下游戏。

在开始修改之前，要明确的是，我们修改的目标是【脱壳版三国志11威力加强版1.1】的exe，是一个修改日期为2014/06/01，文件大小约156MB的exe，如果使用未脱壳版的exe，可能出现CE跳出等问题。

下面就开始吧。CE最大的好处在于，它可以动态设置断点。达到和主流正向编译器差不多的效果。在游戏中，任何内政指令都一定会调用行动力减少的函数，在行动力减少的函数执行完毕后，会返回到之前调用的下一条指令。因此，我们只要将断点设置到行动力函数减少的函数中，然后看一下是谁调用了这个函数（专业术语：查看调用堆栈），我们就可以知道每个函数都是干什么的了。

完整步骤如下：

①打开311pk脱壳版主程序，载入游戏。【将分辨率设置为窗口模式】（否则停在断点之后还要切出去，很麻烦）。

②打开CE，点击左上角闪烁的图标，选择已经打开的311主程序（可能是乱码，因为311主程序采用的是繁体版的BIG5编码，在Windows下兼容性不好）。

③点击CE上的”Memory View”，进入内存查看区域。右键点击”jump to address”，输入已知的减行动力代码段地址：4A1820，在跳转后选中代码，使用快捷键F5设置断点。设置断点后，该行代码将变为红色。

如果我们要设置断点，就需要把CE自带的调试器连接到程序上，如果弹出“是否连接调试器”的弹窗，记得点“是”。

这一步中，我们并不关心在4A1820段发生了什么，只关心它会退出到哪里。因此，我们可以直接找到这个函数的ret指令，设置断点，这样就避免每次还需要手动完成整个函数体的执行过程。

④切回游戏。假设我们想知道巡查的函数位于哪个位置，我们就进行一次巡查。如果上述操作正确，CE的Memory View弹窗的标题会提示，现在已经停在了某个断点。我们接下来使用快捷键F8进行单步执行。（注意：F7 F8都是单步执行，区别在于，如果一行代码调用了一个函数，F7会进入这个函数，F8会直接等待函数执行完毕进入下一行。如果需要查看函数调用的子函数的执行过程，请使用F7。）

⑤在执行完ret后，函数会回到上一层调用的函数中，如果到了哪一层时，函数不一样了，就说明这一层是因行动而已的调用函数了。我们接下来就可以针对性的分析这一段代码，这样就节省了大量的精力。

贴一下我找到的各个行动指令调用行动力的位置：

军团改变行动力：004A1820

军团行动(会调用上一个函数消耗行动力)：005B9340

巡查：5CBF95

市场等内政建设：5BC4C1

商人：5CAD1B

征兵：5C3CAB

生产兵装：5C67B3

研究技巧：5D8F68

吸收合并：5D758B

在这些函数中，会先push入一个值（比如20）作为函数参数，然后执行消耗行动力的函数。我们只需要先判断君主有没有这个技能，然后针对性的给每一个-1即可。加入这些逻辑，我们需要在额外代码段设置代码，然后跳转过去。这一部分的操作方法在上一弹中已经介绍的比较详细了，留给诸位作为练习~

顺便讲一下RK的君主技是怎么实现的吧。这一部分不是我做的，是RK和我讲的。在RK的整理资料中，我们会发现每个武将实际上有0x190个字节的信息，包括了五维、性格、亲爱武将等信息，集中储存在程序7200000左右的区域。在C语言中，这实际上就是很多个struct。在这0x190个字节中，有一部分是空白的，也就是说，并没有被用上，被统一置为全0。我们可以利用这一部分空白信息加入君主技的相关信息，然后编写相关函数读写信息，完成君主技的系统设计。

至于这一部分，希望RK多多勤政，早日放出SIRE 1.30。

下一弹，我们讲多人搜索的实现（which我还没完全理清头绪，所以我会一边做一边写一边发）

## 第六弹·多人搜索（1）UI文字修改（兼CE内存搜索I）

多人搜索是PKME的亮点功能，我们这一整个的目的是把多人搜索整合进SIRE。在第一弹中，我们先进行最简单的UI替换。

首先要明确的是，所有的字符串，比如“登庸武将”、“螺旋突刺”等都储存在exe文件中的后部，会随着exe载入内存。因此，我们只要修改内存中相应的数据（而非代码），就可以改变UI中的文字。

三国志11中用的是【BIG5】编码储存繁体汉字的。BIG5转换器在网上都有，请大家自行寻找。

下面我们讲以下如何用CE来搜索替换字符串。首先，假设我们要改变“探索人才”为“多人探索”。首先我们要写出“探索人才”的【繁体字】，这个最蠢的办法就是在word里转换，其他方法大家自行探索。然后在BIG5码查询器中查出对应的BIG5码，按照【小端序】排列。小端序是什么意思呢？就是把所有的字节(2位16进制)按倒序排列就好了。比如，“探索”的小端序为C1AFB4B1，“人才”为F7A748A4。

接下来，打开CE，在右侧点选“Hex”框（因为是16进制），然后选择4 byte（8位16进制 = 4 byte），点击first scan。如果你的操作没有错误，左侧的选框会出现大量匹配项。其中的某一个就是我们要找的。在这里有一个小技巧：在这些选项中，有一些是固定的数据区，不会随着操作改变；还有一些只是被加载到内存里的数据。因此，我们可以在游戏里随便干点别的事情，然后再回去，点击“Next Scan”，在刚刚已经筛出的结果中再筛一次，这样就可以筛掉被加载到内存中的部分，留下固定的数据区。

剩下的中，有一个是UI中的“探索”或者“人才”字符串，我们需要找到它。如果数据不多的话可以一个一个改，然后看看游戏中有没有相应的改变。如果比较多的话可以采用二分搜索的办法，每次改一半，这样的算法复杂度下降为logN。

最终我们会发现，“探索”位于007EB684，“人才”位于007EB688，将这两个数据用CE改成“多人”与“探索”的BIG5编码的小端序，即可完成UI中的替换。

在这里我没有太过深入的去讲解CE的操作方法，因为类似的教程在网上有很多，甚至CE还自带了一个类似于闯关游戏的教程，推荐大家玩一玩（在菜单Help – Cheat Engine Tutorial中）

下一弹我们讲如何增加菜单项（在召唤移动搜索登庸褒赏授予没收的基础上，增加一个新的选单）

## 第六弹·多人搜索（1）界面修改（兼CE内存搜索II）

这一弹中我们增加选单数。这里的逻辑是怎么实现的呢？在RK的指点下，我们可以去研究一下计略·落雷的显示逻辑——如果部队中有鬼门特技，则落雷和妖术会显现，如果有妖术特技，单独的妖术会显现，而如果都没有，则两种计略连显示都不会显示（而非灰色）。

我们推测，应该是有一个专门的显示函数，函数的一个参数就是选项的数量，然后一个一个去调用文字、显示。如果我们能改掉传入的参数数量，那么就可以起到增加/减少选单的作用了。（为什么要减少选项呢？比如说我们可以删掉吸收合并的UI显示。）

我们采用CE的Next Scan模式来定位传入的选项数量参数。首先，我们点击城市 – 都市，会出现开发、征兵等等选单。维持鼠标位置不要动，直接用Alt + Tab切到CE，设置搜索长度2 byte，搜索7，因为都市下有7个选项。搜索后，我们得到了一堆结果。切回311，打开军事选单，训练出征输送选项出现后，维持鼠标位置，切回CE，搜索3，点击Next Scan。再切回311，点开人才，再切回CE搜索Next Scan...几次之后，我们发现只有三个结果了，它们会随着我们选择的菜单而改变选项数。分别位于0019F318, 0019F334, 0019F42C。0019xxxx是属于游戏运行时的堆栈区域，并不是我们的代码，因此没有办法静态分析，必须动态分析它们是如何压进去的。

这三个数据中只有最早确定的那个是有用的，因为后两个参数很可能是UI“复制”了第一个参数用于显示。这里LZ有一个重要的小技巧，要特意讲一下。在汇编语言中，堆栈的push顺序是从高到低的。换言之，假如第一个数据在10000，我们再push一个数据，它的位置是9999，之后是9998，以此类推。换言之，如果几个数据同时存在于栈中，【那么地址最高的数据一定是最先入栈的】。因此，0019F42C是最早入栈、最根本的数据。这个判断方法需要保证这几个数据在栈中一直存在，不能消失。在大多数情况下，都是好用的。当然，这个方法只适用于堆栈区域，如果是静态代码就没有用了。

如果实在不放心这个trick，或者遇到了诡异的情况，一种方法就是查看三个数据的调用堆栈顺序，最顶层的调用者所对应的数据是最根本的数据。如果我们使用这种办法，会发现调用0019F318的堆栈为4DFE26<-4DC47A<-4DC8EA<-59EE9A<-59F204<-59F4C9<-618687<-6187B1（右侧调用左侧），而最右边的59Fxxx正是0019F42C对应的修改代码段，也印证了0019F42C是最根源的数据。

接下来，我们选中0019F42C的数据，右键点击Find out what writes to this address，监视所有改变这个值的代码。接着我们回到游戏，再随便点一个别的选单，就会发现在监视窗口中，出现了两行代码。第一行代码只被调用了一次，在IDA中反汇编这行代码，发现是将这个值清零。这个是一个初始化清零的指令，我们不用管。第二行代码(0059F281)被调用了n次（n=选项数量），在IDA或CE中查看，会发现这是一行自增指令。

因此，代码逻辑很清晰了。每次点击新的菜单时，先将显示的选项数（称为x）清零，接着，进行n次x++操作，最终达到 x = n的效果，其中n是菜单数量。因此，只要我们修改自增指令的调用次数n，就能相应修改显示的菜单数。

在CE中观察4DFE26的调用堆栈，理论上，我们要找的东西为一个带break的while循环。Break条件即为循环指定次数。我们沿着上边的调用堆栈一路找过去，发现6185C0函数就是我们要找的：在函数中，出现了这样的语句：

while ( 1 )

{

// call some functions,

if ( ++v7 > v11 )

{

v5 = a2;

goto LABEL\_11; // 这一行等同于break

}

}

用CE设断点调试，我们发现eax与edi寄存器中储存的是【从总菜单的第一条选项（即内政·开发）到当前菜单最后一条的距离】，edx初始存的是“当前菜单栏中第一条的序号”（比如“君主”菜单的第一条·评定的序号为32），每次自增edx，等edx == eax后跳出。最终的效果就是，显示当前菜单从第一条到最后一条的全部内容。而具体eax和edx中的序号和距离，合理推测是从内存中某一段固定数据中提取出来的。

分析上下文汇编代码，发现eax与edi中的【总距离】是调用了一个函数后确定的，但是只用IDA无法分析出调用了哪个函数。用CE动态调试，发现函数位于617890，点开之后发现就是非常明了的一个switch-case语句，写死了每一个菜单（如内政，军事，人才等）中选项卡的个数。我们直接改掉这部分的内存、给人才选项卡多一个选项就ok了。

4DFE26<-4DC47A<-4DC8EA<-59EE9A<-59F204<-59F4C9<-618687<-6187B1<-4D5C7A<-4D5752<-4D5902<-4D5ACA<-4773D1<-

【禪讓】(BIG5:FDC549C1)

刘岱君主：0720A840

修改指令：00481584

**修改君主调用堆栈：**

481584->

**4B7AB2->**

**4B917C(4B9080)**->***很可能是我们要找的函数！***

4ACDDA(4ACBE0)-> 斩首后的操作

**4B2C47**(4B2820)->

4BEFE6(4BE790)->灭队之后的操作

4BFA4E(4BFA30)->

5B1D6C(部队攻击结算)->

1C6BB5->1C85C6->1B9F40->1B9CD3->1B2BE0->61B0C3

**君主死亡调用堆栈：**

SetPersonStatus->SetPersonStatus\_WithTest)->**4B79EE(4B78F0)->4B917C(4B9080)**->**4ACDDA(4ACBE0)->4B2C47**(4B2820)

**一般死亡的调用堆栈：**

489902->4BCC24(4BCA30)->4BCE31(4BCCD0)->4A5E3C->**4ACC7F(4ACBE0)->4B2C47**(4B2820)

**掉兵力的调用堆栈：**

496243->4AE4F2->5B1C66->586BB5->

**结论：**

**只有斩首后才进入的函数：4ACBE0**

**修改方式：**

**57555C:**

**mov ecx, esi**

**call 4866F0 // 取城市指针，eax<-城市首地址**

**mov ecx, eax**

**mov eax, [eax]**

**call dword ptr [eax+40] // 通过城市指针取势力ID**

**push eax // eax<-势力编号**

**mov ecx, 7201958**

**call 490AA0 // 通过势力ID获得势力指针，存于eax**

**push 0 // int**

**push 0 // void \***

**push eax // push势力指针**

**mov ecx, 755895C**

**call 4B9080**

**jmp 576232**

刚研究了一波UI，有几个结论，抄送一下羊羊@txz\_mk 羊羊

1. 在函数选取正确的情况下，UI<-->窗口<-->逻辑的桥梁可以完全打通，没有太大技术难度。然而，如果需要自定义新的窗口，需要吃透UI的大量函数，比较费时间，建议新功能在已有菜单上改。
2. 测试了一下最简单的新增选项·禅让，没有出现bug，当然这得益于改变君主是一个已经写好可以随意调用的函数。
3. 目前的问题是，游戏中所有的string是紧密排放的（这是编译器代码·数据分离思想指导下的结果），采用base+offset的方式寻址。这使得加入新的菜单比较困难，目前只能做到替换功能，替换后原有功能消失。比如禅让改写了吸收合并的函数。这一部分我和RK的初步共识是可以通过在寻址时新增代码解决，但是还处于嘴炮状态（逃