贪吃蛇
反汇编代码分析报告

THINCT

January 29, 2024

SnakeGame

- SnakeGame::update
- SnakeGame::update 推测 this 的成员类型
- SnakeGame::SnakeGame 推测 this 的成员类型
- SnakeGame CE 动态监视 this 的成员类型
- SnakeGame 内存分析 this 的成员含义

SnakeGameApp

SnakeGame

SnakeGame::update

SnakeGame::update 推测 this 的成员类型

SnakeGame::SnakeGame 推测 this 的成员类型

SnakeGame CE 动态监视 this 的成员类型

SnakeGame 内存分析 this 的成员含义

SnakeGameApp

EBX 代替当前的函数栈底

```
004079C0    push    ebx
004079C1    mov     ebx, esp
004079C3    sub     esp, 8
004079C6    and     esp, -8
004079C9    add     esp, 4
004079CC    push    ebp
004079CD    mov     ebp, [ebx+4]
004079D0    mov     [esp+4], ebp
004079D4    mov     ebp, esp
```

1. 当 eip 在.text:004079C0 处，esp 所指向的是 ret addr.
2. 当 eip 在.text:004079C1 处，ebx 所指向的是 esp-4. 此时:
   - ebx+4 指向的是 ret addr
   - ebx+8 指向的是第一个参数

EBX 代替当前的函数栈底

```
                    004079C3    sub     esp, 8
                    004079C6    and     esp,
0FFFFFFF8h
                    004079C9    add     esp, 4
                    004079CC    push    ebp
```

esp 实现了向下最近的 8 的倍数取证。比如 12 取整就是 8，16 取整就是 16，18 取整就是 16. 因为是针对栈结构地址取整，所以越是往小的方向越安全，因为对于栈结构来讲，越小的地址是没有用过的地址。所以后面的 ebp,esp, ebp 只能作为局部变量的索引，而对于参数的索引，用 ebx 比较合适。

总结:

对于这个函数来讲，并不是按照套路 *ebp* 作为局部变量和函数参数的唯一参考.

operator += 传参

```
                004079FF    mov     eax,
[ebx+8]
                00407A02    mov     ecx,
[eax+4]
                00407A05    push    ecx
                00407A06    mov     edx, [eax]
                00407A08    push    edx
                00407A09    mov     eax,
[ebp-2Ch]
                00407A0C    add     eax, 28h
; '('
                00407A0F    push    eax

00407A10    call    sf::operator+=(sf::Time
&,sf::Time)
```

operator += 传参

```
                004079FF    mov     eax,
[ebx+8]
                00407A02    mov     ecx,
[eax+4]
                00407A05    push    ecx
                00407A06    mov     edx, [eax]
                00407A08    push    edx
                00407A09    mov     eax,
[ebp-2Ch]
                00407A0C    add     eax, 28h
; '('
                00407A0F    push    eax

00407A10    call    sf::operator+=(sf::Time
&,sf::Time)
```

operator += 传参

总结:

operator += 第一个参数是传地址, 第二个参数是传值, 只不过 sf::Time 的内存是 8 个字节, 所以从起始地址连续压栈 2 次. 本重载函数主要需要掌握的是: **不能根据 *push* 来判断函数的参数个数**.

函数的返回值才是第一个参数

```
                00407A19 mov         ecx,
[ebp-2Ch]
                00407A1C movss       xmm0,
ds:__real@3f800000
                00407A24 divss       xmm0, dword
ptr [ecx+24h]
                00407A29 push        ecx
                00407A2A movss       dword ptr
[esp], xmm0
                00407A2F lea         edx,
[ebp-28h]
                00407A32 push        edx
                00407A33 call
ds:__imp_?seconds@sf@@YA?AVTime@1@M@Z ;
sf::seconds(float)
```

函数的返回值才是第一个参数

```
7AC94C90    movss    xmm0,dword ptr [esp+8]

7AC94C96    mulss    xmm0,dword ptr
ds:[7AC982B8h]
            7AC94C9E    call     7AC9600E

7AC94CA3    mov         ecx,dword ptr
[esp+4]
            7AC94CA7    mov         dword
ptr [ecx],eax
            7AC94CA9    mov         eax,ecx
            7AC94CAB    mov         dword
ptr [ecx+4],edx
            7AC94CAE    ret
```

参数与函数可能隔了几个 call

```
                00407B1A    push        1 ;
includesHead
                00407B1C    mov         ecx,
[ebp-2Ch]
                00407B1F    add         ecx,
1408h

00407B25    call        sf::Transformable::getPo
                00407B2B    push        eax ;
location
                00407B2C    mov         ecx,
[ebp-2Ch]
                00407B2F    add         ecx,
30h ; this
```

参数与函数可能隔了几个 call

```
                00407B1A    push        1 ;
includesHead
                00407B1C    mov         ecx,
[ebp-2Ch]
                00407B1F    add         ecx,
1408h

00407B25    call        sf::Transformable::getPo
                00407B2B    push        eax ;
location
                00407B2C    mov         ecx,
[ebp-2Ch]
                00407B2F    add         ecx,
30h ; this
```

参数与函数可能隔了几个 call

总结:

1. 通过动态调试观察 esp 变化来判断参数的个数
2. 经过编译器的优化, 函数的 push 可能在其他 call 之前进行压栈的

思考:

在函数被调用前后观察 esp 的变化, 是不是就能确定函数的调用约定了呢?

SnakeGame

- SnakeGame::update
- SnakeGame::update 推测 this 的成员类型
- SnakeGame::SnakeGame 推测 this 的成员类型
- SnakeGame CE 动态监视 this 的成员类型
- SnakeGame 内存分析 this 的成员含义

SnakeGameApp

[ebp-2Ch] 存放 this, 加法偏移识别 this 的成员变量

```
00407A16    add         esp, 0Ch
00407A19    mov         ecx, [ebp-2Ch]
00407A1C    movss       xmm0, ds:__real@3f800000
00407A24    divss       xmm0, dword ptr [ecx+24h] ; float :
this->offset24h
00407A29    push        ecx
00407A2A    movss       dword ptr [esp], xmm0 ;
esp->ecx    [esp]=ds:__real@3f800000
00407A2F    lea         edx, [ebp-28h] ;
sf::seconds的返回值存放的临时变量
00407A32    push        edx
00407A33    call        ds:__imp_?seconds@sf@@YA?AVTime@1@M@Z ;
sf::seconds(float)

00407A44    mov         edx, [ebp-2Ch]
00407A47    mov         eax, [edx+2Ch]
00407A4A    push        eax
00407A4B    mov         ecx, [edx+28h] ; sf::Time : this->offset28h
00407A4E    push        ecx
00407A4F    call        ds:__imp_??Psf@@YA_NVTime@0@0@Z ;
sf::operator>=(sf::Time,sf::Time)
00407A55    add         esp, 10h

00407A6A    push        ecx
00407A6B    mov         edx, [ebp-2Ch]
00407A6E    add         edx, 28h ; sf::Time : this->offset28h
00407A71    push        edx
00407A72    call        ds:__imp_??Zsf@@YAAAVTime@0@AAV10@V10@@Z ;
sf::operator-=(sf::Time &,sf::Time)
```

[ebp-2Ch] 存放 this, 加法偏移识别 this 的成员变量

```
          00407A90    mov         ecx, [ebp-2Ch]
          00407A93    add         ecx, 13F0h              ;
std::deque<Direction> : this->offset13F0h

00407A99    call        ?front@?$deque@VDirection@@V?$allocator@VDirection@@@std@@@std@@
; std::deque<Direction>::front(void)

          00407ACC    push        eax                             ; right
          00407ACD    mov         ecx, [ebp-2Ch]
          00407AD0    add         ecx, 1408h

00407AD6    call        ds:__imp_?getPosition@Transformable@sf@@QBEABV?$Vector2@M@2@XZ
; sf::Transformable::getPosition(void)

                00407AFE    push        edx                             ; result
                00407AFF    mov         ecx, [ebp-2Ch]
                00407B02    add         ecx, 4Ch ; BackgroundGrid :
this->offset4Ch

00407B05    call        ?generateRandomPosition@BackgroundGrid@@QAE?AV?$Vector2@M@sf@@XZ
; BackgroundGrid::generateRandomPosition(void)

          00407B0B    mov         ecx, [ebp-2Ch]
          00407B0E    add         ecx, 1408h ; sf::Transformable :
this->offset1408h

00407B14    call        ds:__imp_?setPosition@Transformable@sf@@QAEXABV?$Vector2@M@2@@Z
; sf::Transformable::setPosition(sf::Vector2<float> const &)

          00407C00    mov         ecx, [ebp-2Ch]
          00407C03    add         ecx, 30h ; Snake : this->offset30h
          00407C06    call        ?setColor@Snake@@QAEXVColor@sf@@@Z ;
Snake::setColor(sf::Color)
```

[ebp-2Ch] 存放 this, 加法偏移识别 this 的成员变量

总结:

在同一个函数中，找到 this 变量, 将所有基于该变量偏移的片段截取出来，结合已知的调用函数原型推导变量的含义和类型.

SnakeGame

- SnakeGame::update
- SnakeGame::update 推测 this 的成员类型
- SnakeGame::SnakeGame 推测 this 的成员类型
- SnakeGame CE 动态监视 this 的成员类型
- SnakeGame 内存分析 this 的成员含义

SnakeGameApp

[ebp-14h] 存放 this, 加法偏移识别 this 的成员变量

```
004074AC mov         [ebp-14h], ecx
004074C6 mov         edx, [ebp-14h]
004074C9 mov         dword ptr [edx], offset ??_7SnakeGame@@6B@

004074D1 mov         ecx, [ebp-14h]
004074D4 add         ecx, 10h                ; this
004074D7 call        ??0Direction@@QAE@H@Z ;
Direction::Direction(int)

004074ED mov         ecx, [ebp-14h]
004074F0 add         ecx, 14h
004074F3 call        ds:__imp_??0Color@sf@@QAE@EEEE@Z ;
sf::Color::Color(uchar,uchar,uchar,uchar)

0040750D mov         ecx, [ebp-14h]
00407510 add         ecx, 18h
00407513 call        ds:__imp_??0Color@sf@@QAE@EEEE@Z ;
sf::Color::Color(uchar,uchar,uchar,uchar)

00407524 mov         ecx, [ebp-14h]
00407527 add         ecx, 1Ch
0040752A call        ds:__imp_??0Color@sf@@QAE@EEEE@Z ;
sf::Color::Color(uchar,uchar,uchar,uchar)

0040753B mov         ecx, [ebp-14h]
0040753E add         ecx, 20h ; ' '
00407541 call        ds:__imp_??0Color@sf@@QAE@EEEE@Z ;
sf::Color::Color(uchar,uchar,uchar,uchar)

00407547 mov         eax, [ebp-14h]
0040754A movss     xmm0, ds:__real@41200000
00407552 movss     dword ptr [eax+24h], xmm0
```

[ebp-14h] 存放 this, 加法偏移识别 this 的成员变量

```
00407557 mov         ecx, [ebp-14h]
0040755A add         ecx, 28h ; '('
0040755D call        ds:__imp_??0Time@sf@@QAE@XZ ;
sf::Time::Time(void)

0040778D mov         ecx, [ebp-14h]
00407790 add         ecx, 13F0h             ; this
00407796
call        ??0?$deque@VDirection@@V?$allocator@VDirection@@@std@@@std@@QAE@XZ
; std::deque<Direction>::deque<Direction>(void)

004077E0 mov         ecx, [ebp-14h]
004077E3 add         ecx, 1404h
004077E9
call        ds:__imp_?setFillColor@Shape@sf@@QAEXABVColor@2@@Z ;
sf::Shape::setFillColor(sf::Color const &)
004077EF

004077F6 mov         ecx, [ebp-14h]
004077F9 add         ecx, 4Ch ; 'L'     ; this
004077FC
call        ?generateRandomPosition@BackgroundGrid@@QAE?AV?$Vector2@M@sf@@XZ ;
BackgroundGrid::generateRandomPosition(void)

                00407811 push        1
                00407813 mov         ecx, [ebp-14h]
                00407816 add         ecx, 1408h
                0040781C
call        ds:__imp_?getPosition@Transformable@sf@@QBEABV?$Vector2@M@2@XZ ;
sf::Transformable::getPosition(void)

00407822 push        eax                            ; location
00407823 mov         ecx, [ebp-14h]
00407826 add         ecx, 30h ; '0'     ; this
00407829
call        ?collidesWith@Snake@@QBE_NABV?$Vector2@M@sf@@_N@Z ;
Snake::collidesWith(sf::Vector2<float> const &,bool)
```

[ebp-14h] 存放 this, 加法偏移识别 this 的成员变量

总结:

一般构造函数会对所有成员变量进行初始化的, 所以从构造函数来分析 this 的成员变量是一个不错的选择.

上面 this 偏移分析的结构梳理

Table 1: SnakeGame Parse

Class members by offset	
+10h	m_Direction_1
+14h	m_sf__Time_1
+18h	m_sf__Color_1
+1Ch	m_sf__Color_2
+20h	m_sf__Color_3
+24h	m_float_second_1
+28h	m_sf__Time_2
+30h	m_object_Snake_1
+4Ch	m_BackgroundGrid
+13F0h	m_std__deque_Direction_1
+1404h	m_sf__Shape
+1408h	m_sf__Transformable_position

SnakeGame

- SnakeGame::update
- SnakeGame::update 推测 this 的成员类型
- SnakeGame::SnakeGame 推测 this 的成员类型
- SnakeGame CE 动态监视 this 的成员类型
- SnakeGame 内存分析 this 的成员含义

SnakeGameApp

CE 平坦式内存

Memory Viewer

File Search View Debug Tools Kernel tools

PrjSnakeGame.WinMainCRTStartup

Address	Bytes	Opcode	Comment
PrjSnakeGame.WinMainCRTStartup()			
PrjSnakeGame.Win	E8 1C040000	**call**	PrjSnakeGame.__security_init_cookie
PrjSnakeGame.Win	E9 7AFEFFFF	**jmp**	PrjSnakeGame.__scrt_common_main_se
PrjSnakeGame.__raise_securityfailure()			
PrjSnakeGame.__rai	55	**push**	ebp
PrjSnakeGame.__rai	8B EC	**mov**	ebp,esp
PrjSnakeGame.__rai	6A 00	**push**	00 0
PrjSnakeGame.__rai	FF 15 04B04000	**call**	dword ptr [PrjSnakeGame._imp__SetUnl->KERNEL32.SetUnhandledExceptionFilter
PrjSnakeGame.__rai	FF 75 08	**push**	[ebp+08]
PrjSnakeGame. _rai	FF 15 30B04000	**call**	dword ptr [PrjSnakeGame._imp__Unhan->KERNEL32.UnhandledExceptionFilter

call procedure

Protect:Read/Write AllocationBase=07BA0000 Base=0801B000 Size=B4000

```
address  C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF 0123456789ABCDEF0123456789ABCDEF
0801B5C0 AC B9 40 00 C0 B5 01 08 90 01 00 00 B4 01 00 00 03 00 00 00 00 FF 20 FF B2 B5 BA FF 4D 1A 6B FF   @.  ..  ... ........      M.k
0801B5E0 4D 1A 5A FF 00 00 20 41 3F 37 00 00 00 00 00 00 8C B9 40 00 10 CB 01 08 04 00 00 00 03 00 00 00 M.Z .. A?7.....  @.. ..........
0801B600 00 00 70 41 00 00 80 41 00 FF 94 FF D8 B2 40 00 C0 CD 09 08 50 D6 0C 08 84 E2 0C 08 00 00 C8 43 ..pA.. A.     @.  ..P ..  .... C
0801B620 00 00 DA 43 00 00 00 00 71 02 00 00 CA C1 2A 8B E9 7B 0A 76 8A 0A CA 71 DA 8B 27 F2 A1 4D 60 12 .. C....q...  *  {.v . q  '  M`.
0801B640 8A C9 F0 D4 13 D4 98 2D 86 D6 4E 97 1C 46 83 70 7A 2E 1B CB C7 16 E7 FE 5F DF B6 8B BD E4 39 54      .  -  N .F pz..  .  _     9T
0801B660 39 DA 63 C7 F0 22 C4 89 89 4B B3 5D B8 94 E0 5B 0E AE 40 8C CE 16 FD C4 F4 B3 A8 44 BD 1C 90 A6 9 c  "   K ]   [. @  .    D .
0801B680 70 8E 76 33 46 22 10 E5 50 72 97 0D A8 E9 1C 49 C6 A0 12 01 38 64 CD 51 98 0B 3A FA 43 87 59 7E p v3F". Pr .  .I  ..8d Q .: C Y~
0801B6A0 27 AF 82 4D 1C 70 54 56 90 C0 1A 8C BA 1B 9A 73 E8 03 C1 BA 74 C5 A4 CC 16 97 EF 84 08 4F EC 3B '  M.pTV  .  . s .  t   .  .O ;
0801B6C0 4D 76 BE A3 51 F4 9D 17 1C BD 47 B3 FE AA 54 F3 FA DA D3 B8 02 1C 8C E8 90 95 4A EA 2B AE 7D 74 Mv  Q  .. G   T     ..    J + }t
0801B6E0 BF 30 F0 B2 BF 5F 03 34 8A FD CA 85 D8 CE 7B B9 33 46 81 0F 51 FD F6 BB F2 5C 48 47 13 B7 EF E7  0   _.4      { 3F .Q    \HG.
0801B700 85 C9 FA 8B 79 C1 7B E6 59 9F B3 EB BA 08 9B 71 00 85 F3 EA 69 15 57 EB 0D 2D B7 C1 C2 44 C3 97     y { Y   . q.   i.W .-   D
0801B720 FD DF 14 41 AA 3A FE 48 B6 A8 5F 98 44 E3 F1 DB 44 AA 00 64 7B 1A 45 93 00 33 BD DA 73 BB 06 36   .A : H  _ D   D .d{.E .3  s .6
0801B740 A4 7F 1D 8F C4 32 7D C5 CA 87 02 42 5F 36 38 C0 95 AE C8 37 13 9E AD 09 CA 88 9C 11 FE 11 7C AD  ▯.  2}   .B_68   7.  .   . .|
0801B760 B9 F4 76 A5 15 A1 03 86 62 DD 44 EC 95 40 4B 0B 1A 38 53 5A F9 95 98 18 90 6C 44 82 EE F7 16 93   v . . b D  @K..8SZ   . lD  .
0801B780 71 1C 31 AA B5 C4 BE B8 8A 8B 8A 8F 00 D5 0C F7 88 A5 23 6D 67 A0 65 75 99 DE B8 1E B9 B3 3E B0 q.1          . .   #mg eu   .  >
0801B7A0 24 FC 08 E5 C1 5A 8D 84 4E 2A AC B0 5C 5C A3 0E AD AC 64 89 6D C8 EA A1 2E 7B ED D5 25 AE D7 D7 $ .  Z  N*  \\ .  d m  .{  %
0801B7C0 63 0B 54 5F 10 F0 ED 3D B7 46 C8 14 9B D5 DC 83 C6 26 9E 97 BE 2F 80 33 61 84 51 61 4C 9A A0 23 c.T_.  = F .     &   / 3a QaL  #
0801B7E0 69 8C 03 49 76 0D 62 6B 62 FF 24 69 7F BD F9 07 8C BA 69 8E 78 98 6E C6 FA FB 98 88 4C 21 FB 46 i .Iv.bkb $i▯  .  i x n    L! F
0801B800 D6 58 01 CB 7F 09 E5 19 92 B6 EA 9F 48 17 3D 71 46 41 8F CD B3 AD 55 A8 50 40 98 AF D6 42 BB 1B  X. ▯. .    H.=qFA   U P@   B .
0801B820 EB E4 7F 43 D0 8A 5C 0D 8F 14 7C 57 86 1A D6 62 C4 B6 5B 40 3B 89 61 94 00 A5 79 89 4E 2C 5A B1   ▯C  \. .|W . b  [@; a . y N,Z
```

Figure 1: 平坦式内存

CE 层级内存

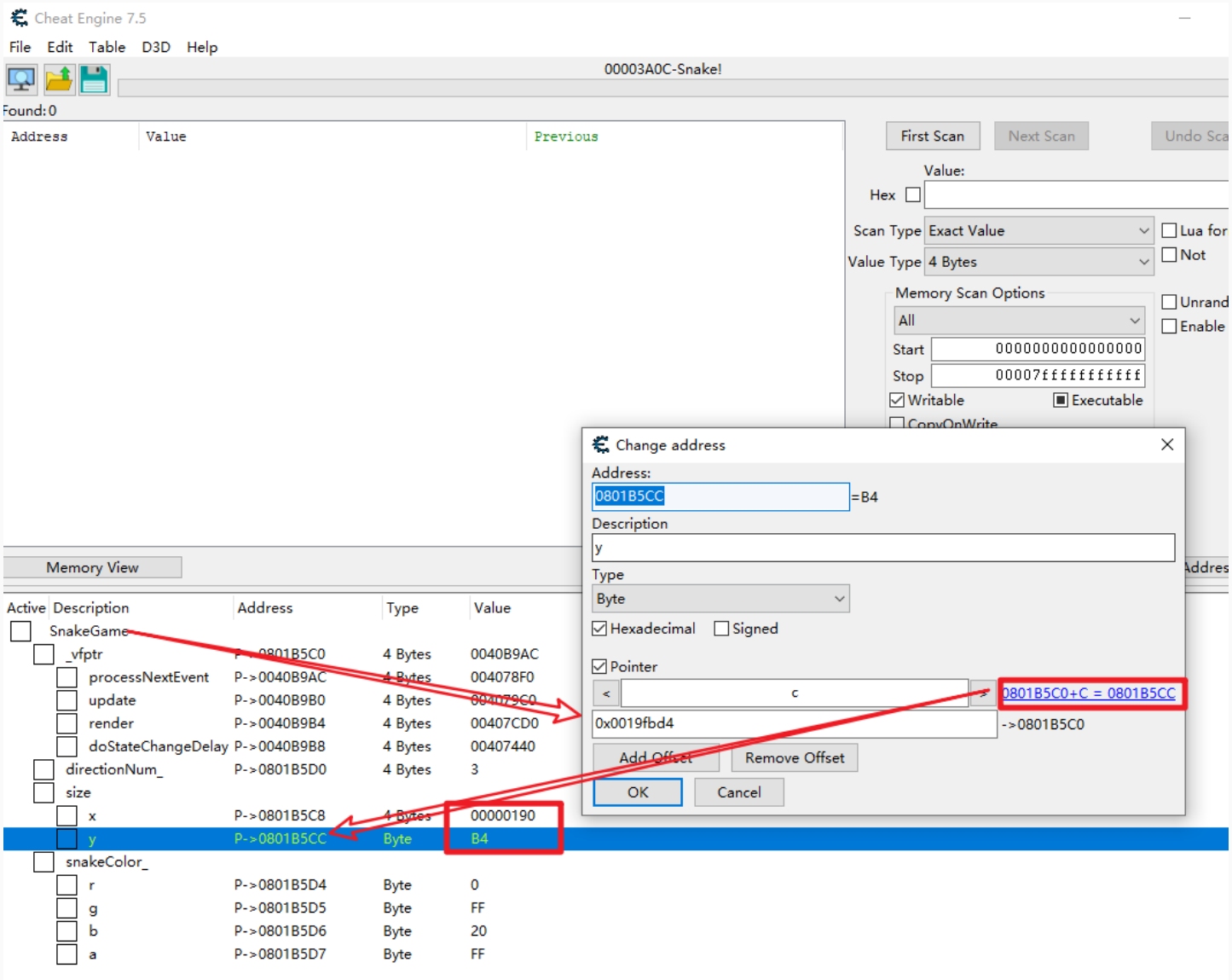

Figure 2: 层级内存

总结:

1. 通过平坦式内存全局观察所有的内存单元变化. 比如 SnakeGame 的内存偏移就说明是线性存储了数据.
2. 通过对象字段存储的地址继续进一步观察，不断得到下一级数据, 从而得到纵向的存储数据.

纵横内存分析，即可描绘出完整的嵌套式数据结构.

SnakeGame

- SnakeGame::update
- SnakeGame::update 推测 this 的成员类型
- SnakeGame::SnakeGame 推测 this 的成员类型
- SnakeGame CE 动态监视 this 的成员类型
- SnakeGame 内存分析 this 的成员含义

SnakeGameApp

时间旅行反汇编 AddrFlowEasy.asm

Table 2: SnakeGame Object Memory Parse

Class members by offset base on memory line	
行 65:	;[edx+0x4]=[0x08A66894]=0x08A66890
行 68:	;[edx+0x8]=[0x08A66898]=0x00000190
行 71:	;[edx+0xC]=[0x08A6689C]=0x00000150
行 88:	;[edx]=[0x08A66890]=0x00406708
行 93:	;[edx+0x10]=[0x08A668A0]=0x00000003
行 141:	;[esi+0x24]=[0x08A668B4]=0x41200000
行 212:	;[eax]=[0x08A668C0]=0x004066E8
行 230:	;[ecx+0x4]=[0x08A668C4]=0x07836E98
行 237:	;[ecx+0xC]=[0x08A668CC]=0x00000003
行 242:	;[ecx+0x10]=[0x08A668D0]=0x41700000
行 245:	;[ecx+0x14]=[0x08A668D4]=0x41800000
行 251:	;[eax]=[0x08A668D8]=0xFF94FF00
行 356:	;[ecx+0x8]=[0x08A668C8]=0x00000001
行 469:	;[esi]=[0x08A67C80]=0x08A36778

对象内存分析

```
/*0x00403691*/         mov edx, ecx
/*0x00403693*/         mov dword ptr ss:[ebp-0x20], edx
;[ebp-0x20]=[0x0019FBF8]=0x00000001
;[ebp-0x20]=[0x0019FBF8]=0x08A66890     <-- Modify
```

通过 ecx 推断可知 0x08A66890 是 Object 对象的首地址. 根据上面的表格的地址计算 0x08A66890 的偏移, 并结合赋值推测变量的类型和含义。

Table 3: SnakeGame Object Memory Parse

Class members by offset base on memory line		
行 65:	;[edx+0x4]=[0x08A66894]=0x08A66890	+Offset 0x4
行 68:	;[edx+0x8]=[0x08A66898]=0x00000190	+Offset 0x8
行 71:	;[edx+0xC]=[0x08A6689C]=0x00000150	+Offset 0xc
行 88:	;[edx]=[0x08A66890]=0x00406708	+Offset 0x0
行 93:	;[edx+0x10]=[0x08A668A0]=0x00000003	+Offset 0x10
行 141:	;[esi+0x24]=[0x08A668B4]=0x41200000	+Offset 0x24
行 212:	;[eax]=[0x08A668C0]=0x004066E8	+Offset 0x30
行 230:	;[ecx+0x4]=[0x08A668C4]=0x07836E98	+Offset 0x34
行 237:	;[ecx+0xC]=[0x08A668CC]=0x00000003	+Offset 0x3c
行 242:	;[ecx+0x10]=[0x08A668D0]=0x41700000	+Offset 0x40
行 245:	;[ecx+0x14]=[0x08A668D4]=0x41800000	+Offset 0x44
行 251:	;[eax]=[0x08A668D8]=0xFF94FF00	+Offset 0x48
行 356:	;[ecx+0x8]=[0x08A668C8]=0x00000001	+Offset 0x38
行 469:	;[esi]=[0x08A67C80]=0x08A36778	+Offset 0x13f0

对象内存分析

Table 4: SnakeGame Object Memory Parse

Class members by offset base on memory line		
行 65:	;[edx+0x4]=[0x08A66894]=0x08A66890	猜测系统 DLL 的函数
行 68:	;[edx+0x8]=[0x08A66898]=0x00000190	SizeX
行 71:	;[edx+0xC]=[0x08A6689C]=0x00000150	SizeY
行 88:	;[edx]=[0x08A66890]=0x00406708	猜测本模块的函数地址, 虚表?
行 93:	;[edx+0x10]=[0x08A668A0]=0x00000003	枚举值
行 141:	;[esi+0x24]=[0x08A668B4]=0x41200000	浮点数
行 212:	;[eax]=[0x08A668C0]=0x004066E8	猜测本模块的函数地址, 虚表?
行 230:	;[ecx+0x4]=[0x08A668C4]=0x07836E98	猜测系统 DLL 的函数
行 237:	;[ecx+0xC]=[0x08A668CC]=0x00000003	枚举值
行 242:	;[ecx+0x10]=[0x08A668D0]=0x41700000	浮点数:15.0
行 245:	;[ecx+0x14]=[0x08A668D4]=0x41800000	浮点数:16.0
行 251:	;[eax]=[0x08A668D8]=0xFF94FF00	???
行 356:	;[ecx+0x8]=[0x08A668C8]=0x00000001	枚举值或者 bool 值
行 469:	;[esi]=[0x08A67C80]=0x08A36778	猜测系统 DLL 的函数

SnakeGame

SnakeGameApp

SnakeGameApp::updateCurrentState

SnakeGame

SnakeGameApp

SnakeGameApp::updateCurrentState

```
0040906B    mov     edx,dword ptr [eax+4]
0040906E    call    edx
```

0040906E call 调用目标是寄存器, 说明 edx 存放的是函数指针, 从 0040906B 处 edx 通过 eax+4 解引用得到的。初步确认 edx 存放的是虚函数指针, 而 eax 是虚表的首地址.eax 进一步在内存中确认:

0x0040B9AC f0 78 40 00

0x0040B9B0 c0 79 40 00

0x0040B9B4 d0 7c 40 00

0x0040B9B8 40 74 40 00

0x0040B9BC 60 78 40 00

从上面的存放内容, 基本就可以确认是虚表内容了. 如果某个对象存放了这个 eax, 那么那个引用地址就是虚表的所属的对象了。

虚函数简单识别

总结:
call 后面的寄存器是由一个地址加偏移的结果解引用得到的, 那么就大胆的猜测它就是虚函数吧!