

# 贪吃蛇

## 反汇编代码分析报告

---



THINCT

December 14, 2023

## SnakeGame

SnakeGame::update

SnakeGame::update 推测 this 的成员类型

SnakeGame::SnakeGame 推测 this 的成员类型

## SnakeGameApp

SnakeGame

SnakeGame::update

SnakeGame::update 推测 this 的成员类型

SnakeGame::SnakeGame 推测 this 的成员类型

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)
```

- 从 0x00407A09 到 0x00407A0F 是第一个参数, 已知 [ebp-2Ch] 为 this, 所以第一个参数为 this->offset28h, 并且为 sf::Time 引用类型. 所以 *sf::Time\* this->offset28h*.

## 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)
```

- 0x004079FF 已推导出为当前函数的第一个参数, 而 0x00407A02 到 0x00407A08 是连续的内存, 从 call 得知这个连续的内存是 sf::Time 类型, 所以推导出 [ebx+8] 是 sf::Time\* 类型, 即 *sf::Time\** [ebx+8]

### 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)
```

0x00407A2F 处压栈的是第一个参数, 为局部变量 (暂存临时返回值), 0x00407A29 和 0x00407A2A 压栈第二个参数, 其中 push 只是占位作用, 0x00407A2A 才是第二个参数的值, 也就是计算出来的浮点数.

## 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
```

观察调用的函数, 分别从 0x7AC94CA3 和 0x7AC94CA9 可知: 第一个参数也是该函数的返回值, 所以可以推断出: **函数的返回值才是第一个参数**, 并且该函数其实只有一个参数, 即 0x00407A2A 处压栈的参数.



## 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::getPosition
00407B2B    push    eax ; location
00407B2C    mov     ecx, [ebp-2Ch]
00407B2F    add     ecx, 30h ; this
00407B32    call    Snake::collidesWith(sf::Vec
const &,bool)
```

0x00407B1A 处压栈参数，在 0x00407B25 call 之后并没有平栈。通过动态调试得知 0x00407B25 前后 esp 没有变化，说明该 call 是没有参数的，跟进 0x00407B25 call 直接将地址给 eax 了，直接说明了 0x00407B1A push 不是 0x00407B25 call 使用的。

## 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::getPosition
00407B2B    push    eax ; location
00407B2C    mov     ecx, [ebp-2Ch]
00407B2F    add     ecx, 30h ; this
00407B32    call    Snake::collidesWith(sf::Vec
    const &,bool)
```

分析 0x00407B32 call, 得出 0x00407B2B 是第一个参数, 上面的 0x00407B1A 是第二个参数. 是因为动态调试发现经过 0x00407B32 call 前后, esp 变化为 8, 所以直接找最近的栈的两个 push 即可.

## 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

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
00407A2F    lea        edx, [ebp-28h]
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)
00407A78    add        esp, 0Ch

```

## [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@@QAEAA
                             ; 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@@@QAEABV?$Vector2@M@2@XZ
                             ; 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)

00407C58    mov     ecx, [ebp-2Ch]
00407C5B    add     ecx, 30h          ; Snake : this->offset30h
00407C5E    call    ?length@Snake@@@QBEIXZ ; Snake::length(void)
```



[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

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    ????$deque@VDirection@V?$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@QAEABVColor@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@QAE@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@QAE@_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.

SnakeGame

SnakeGameApp

SnakeGameApp::updateCurrentState

SnakeGame

SnakeGameApp

SnakeGameApp::updateCurrentState



## 虚函数简单识别

```
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。



### 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!