

菁英班作业第6课

对AssaultCube进行攻击，实现透视、自瞄类的作弊功能。

本作业实现了两种透视功能，一种自瞄功能。

一、搜索基本数据偏移地址

本作业所需的各项数据偏移地址如下。

```
// 结构体偏移信息  
#define entityListOffset (0x58AC04)  
  
#define viewMatrixOffset (0x57DFD0)  
  
#define playerBaseOffset (0x58AC00)  
  
#define amountOfPlayersOffset (0x0058AC0C)  
  
// 实体结构偏移信息  
#define healthOffset (0xEC)  
  
#define locationXOffset (0x28)  
  
#define locationYOffset (0x2C)  
  
#define locationZOffset (0x30)  
  
#define headXOffset (0x4)  
  
#define headYOffset (0x8)  
  
#define headZOffset (0xC)  
  
#define teamOffset (0x30c)  
  
#define yawOffset (0x34)  
  
#define pitchOffset (0x38)  
  
#define nameOffset (0x205)
```

后面只展示几个相对较难，较为重要的地址寻找过程

1、玩家基地址、基本属性寻找。

首先选择寻找的血量这一可控，已知数值。

血量为100时，搜索值为100的数值，结果如下：

结果: 2,206

地址	当前值	先前值	First	
0018FAD8	100	100	100	
0019649C	100	100	100	
001964F0	100	100	100	
0019999C	100	100	100	
00199B58	100	100	100	
00199F14	100	100	100	
0019A40C	100	100	100	
0019A4B0	100	100	100	
0019A964	100	100	100	
0019AC1C	100	100	100	
0019AD48	100	100	100	
0019ADC0	100	100	100	
0019AF40	100	100	100	
0019B6A4	100	100	100	
0019B824	100	100	100	
0019B894	100	100	100	
0019BFE8	100	100	100	
0019C140	100	100	100	

查看内存



使血量减为96，继续搜索，结果如下

结果: 1

地址	当前值	先前值	First	
0092011C	96	96	100	

查看内存



手动

搜索到唯一数值，即为结果。

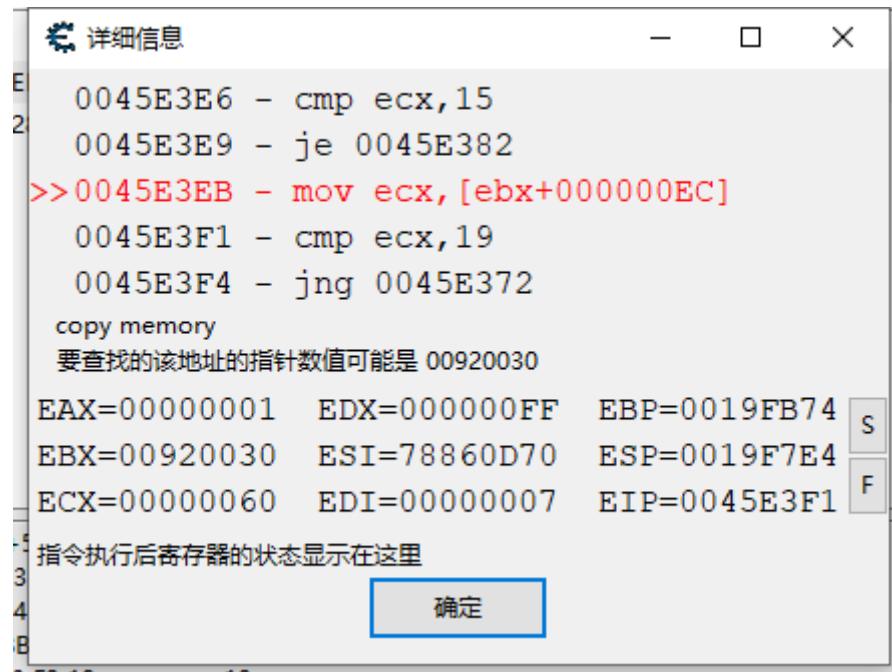
但此地址为动态加载的，每次启动时不同，需要查找其静态存储位置。

查看哪些代码访问了该地址。如下代码访问了该地址。

下列操作码访问了 0092011C

计.. 指令	替换
432 0045E3EB - 8B 8B EC000000 - mov ecx,[ebx+000000EC]	显示反汇编程序
432 00461628 - FF B7 EC000000 - push [edi+000000EC]	添加到代码表
	详细信息
	example
	关闭

其中存在ebx+0xEC，判断血量位于0xEC偏移地址。ebx中存放玩家基址。



玩家基址为0x920030。

在内存中搜索何处存放有0x920030。

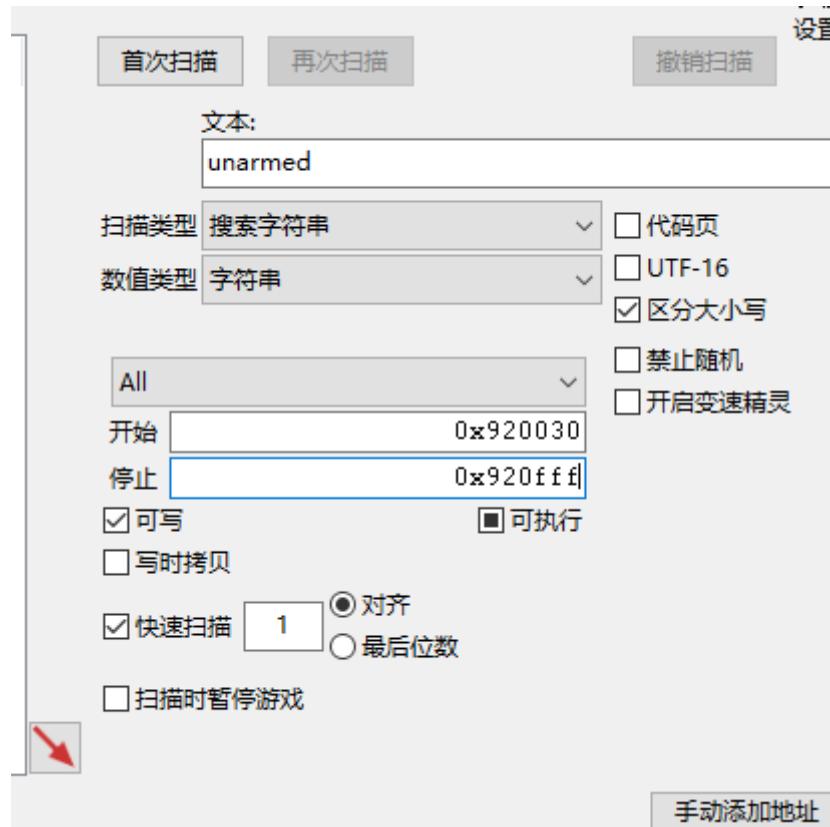
0019DBBC	00920030	00920030	00920030
0019FC44	00920030	00920030	00920030
0019FC90	EB6CC1A9	00920030	00920030
0019FD2C	00920030	00920030	00920030
0019FD34	00920030	00920030	00920030
ac_client.exe+17E0A8	00920030	00920030	00920030
ac_client.exe+17E254	00920030	00920030	00920030
ac_client.exe+17E360	00920030	00920030	00920030
ac_client.exe+18AC00	00920030	00920030	00920030
ac_client.exe+195404	00920030	00920030	00920030
00903EBO	00920030	00920030	00920030
009083F0	00920030	00920030	00920030

如下五个绿色的为静态地址。初步判断均为玩家基地址。

初步选择0x58AC00(基地址0x400000+0x18AC00)地址。

重启游戏后该地址未变。

下面查找玩家名称位置，玩家名称位于unarmed



在玩家基地址后面一部分区域内搜索该字符串。

搜索到唯一结果。

结果: 1	
地址	当前值
00920235	unarmed

其偏移为0x205。

玩家其他属性如脚的坐标，头的坐标，视角等数据可同上面方法搜索。

2、实体 (bot) 列表查找

通过姓名搜索，进而找到实体存储位置与实体列表存储位置

搜索姓名为Zeunerts的敌人

ac_client.exe+17B8C0	Zeunerts
0093A8F8	Zeunerts
0093AA06	Zeunerts
0093AA5F	Zeunerts
0093AAA0	Zeunerts
0093D8E6	Zeunerts
0093D9F4	Zeunerts
0093DA4D	Zeunerts
0093DA8E	Zeunerts
1782EAFD	Zeunerts
1FC087A6	Zeunerts
3B6D6120	Zeunerts
3B6D6E37	Zeunerts
40DDC03A	Zeunerts
40FAC58F	Zeunerts
40FACF69	Zeunerts

查看静态地址处

C0	C4	C8	CC	D0	D4	U123456789ABCDEFU1234567
0 6E75655A	73747265	00000000	00000000	65756C42	6E616D73	Zeunerts.....Bluesman
8 00000000	00000000	66657453	00006E61	00000000	00000000Stefan.....
0 626D7544	00007265	00000000	00000000	547C5058	614E6568	Dumber.....XP TheNa
8 656C656D	00007373	65646F74	72756773	0000656B	00000000	meless..todesgurke.....
0 006D7274	00000000	00000000	00000000	72626153	00000065	trm.....Sabre...
8 00000000	00000000	79646C4F	00000000	00000000	00000000Oldy.....
0 696B696B	00000000	00000000	00000000	6E303062	646E696B	Kiki.....b00nkind
8 00000000	00000000	64697053	535F7265	74756F63	00007265Spider_Scoutter..
0 77614447	00000067	00000000	00000000	656D654D	00000064	GDawg.....Memed...
8 00000000	00000000	73696857	72656C74	00000000	00000000Whistler.....

) : byte: 48 word: 48 integer: 9568304 int64: 1948918032786128944 float:0.00 double: 0.00

其附近地址处均为姓名字符串，判断此处存储机器人姓名，无作用。

查看其他地址有哪些代码访问了

下列操作码访问了 1782EAFD

计..	指令
192	00481B02 - 3A 11 - cmp dl,[ecx]
128	00481A67 - 8A 10 - mov dl,[eax]
128	00481AA6 - 8A 10 - mov dl,[eax]
768	00481B00 - 8A 10 - mov dl,[eax]
128	00481B35 - 80 7D 00 00 - cmp byte ptr [ebp+00],00
128	0041B45D - 0FBE 0F - movsx ecx,byte ptr [edi]
768	0041B472 - 0FBE 0E - movsx ecx,byte ptr [esi]
64	0046DDEE - 38 45 00 - cmp [ebp+00],al
512	0046DE10 - 0FBE 0C 2F - movsx ecx,byte ptr [edi+ebp]
448	0046DE75 - 80 3C 2F 00 - cmp byte ptr [edi+ebp],00

X

替换

显示反汇编程序

添加到代码表

详细信息

example

关闭

0x1782EAFD。

选择某一处汇编指令，如481B02附加的汇编指令

3C	je	ac_client.exe+81B29
4C 24 14	cmp	[esp + 14],ecx
36	je	ac_client.exe+81B29
14 24 10	mov	eax,[esp + 10]
C1 05020000	add	ecx,00000205
F 00	nop	dword ptr [eax]
10	mov	dl,[eax]
11	cmp	dl,[ecx]
1A	jne	ac_client.exe+81B20
02	test	dl,dl

ecx即为字符串存放地址。

add ecx, 0x205。

初步判断ecx一开始存放有实体基地址，205为名称的偏移。

实体基地址为0x1782e8f8。

而玩家的名称偏移恰好也为0x205，也可判断玩家与实体具有相同的结构。

8B 1D 04AC5800	mov	ebx,[ac_client.exe+18AC04]
8B 0C B3	mov	ecx,[ebx + esi*4]
85 C9	test	ecx,ecx
74 3C	je	ac_client.exe+81B29

上面一部分代码，有esi * 4。

类似于指针数组的遍历，初步判断[ebx]为实体列表的基地址。

实体列表指针存放地址为0x58AC04。其值为0x1B0BF2A0。

地址0x1B0BF2A0处数据如下。

地址	A0	A4	A8	AC	B0	B4
1B0BF2A0	00000000	1F32EAA8	169CF5F0	169C1088	0DD0CA58	1782E8F8
1B0BF2B8	0E02FB80	3B727FF8	75EFB45B	88001702	55C592D4	55C6A01C
1B0BF2D0	00000002	00000001	00000001	1675B55C	00000000	1B0BF098
1B0BF2E8	0003003E	00000096	00000090	0004003D	0000002F	00000097
1B0BF300	00000081	0004003D	A3183A41	08FADA33	16AB2790	169CFA10
1B0BF318	00000009	F0E0D0C0	6DDC3933	0000000C	1B0BF32C	FFFFF000
1B0BF330	759CB465	80000000	0045004B	004E0052	004C0045	00320033
1B0BF348	0044002E	004C004C	00000000	00000000	7591B468	80000100
1B0BF360	0045004B	004E0052	004C0045	00320033	0044002E	004C004C
1B0BF378	00000000	00000000	7596B473	80000200	0045004B	004E0052

数组第一个值为0，空值，符合设计习惯。

查看第五个指针0x1782E8F8，恰好为前面找到的名为Zeunerts的实体的基地址。故此判断正确。

因此寻找到存放实体列表地址的指针位于0x58AC04处。访问其地址即可找到实体列表，遍历该列表中的指针，即可找到所有实体对象。

3、寻找玩家的矩阵信息

后面透视需要用到该矩阵信息。

矩阵具有一些特征，可以依据这些特征进行搜索，其特征略。下面只叙述查找方法。

首先搜索-1到1的浮点数据。

搜索到的结果较多，接下来通过改变位置，矩阵改变。位置姿态不变，矩阵不变，进行继续搜索筛选

地址	当前值	先前值	First
ac_client.exe+16BA98	0.9954281449	0.9954281449	0.987798512
ac_client.exe+17DF90	-0.2311792076	-0.2311792076	0.5339778662
ac_client.exe+17DF98	-1.919896704E-7	-1.919896704E-7	7.450079238E-7
ac_client.exe+17DF9C	-5.091929367E-8	-5.091929367E-8	2.669472143E-10
ac_client.exe+17DFA0	0.1094393879	0.1094393879	0.06128234416
ac_client.exe+17DFA4	-0.02600395121	-0.02600395121	-0.03869942203
ac_client.exe+17DFA8	0.7415165901	0.7415165901	0.7464900017
ac_client.exe+17DFAC	-2.15289675E-9	-2.15289675E-9	2.072027883E-8
ac_client.exe+17E098	1.875002384	1.875002384	0.4999988079
ac_client.exe+17E260	0.00001097987297	0.00001097987297	1.568705122E-19
ac_client.exe+17E264	1.669109224E-7	1.669109224E-7	0.00004272460137
ac_client.exe+17E268	2.63938E-6	2.63938E-6	6.555769261E-10
ac_client.exe+19206C	6.730612278	6.730612278	-0.6806271076
00D8AEC4	2.403540757E-39	2.395636033E-39	2.116465149E-39
00D8AED4	1.342443929E-42	1.062184236E-42	1.209320575E-42
.....

最终剩下约183个数据。

而矩阵一般位于固定地址区域内，因此对绿色的固定地址区域进行搜索。

0057DFD0	0.92	0.04	-0.39	-0.39
0057DFE0	-0.39	0.09	-0.92	-0.92
0057DFF0	0.00	1.33	0.07	0.07
0057E000	-86.83	-20.38	144.23	144.51

该4X4矩阵满足以下三个特征：

1. 跳动时，3个数据变动
2. 晃动鼠标时，一个数据保持0不动
3. 存在3对两两相同

故该地址0x57DFD0为矩阵存放地址。

4、其余信息搜索

同上述方法即可找到所有所需信息。

二、通过矩阵实现透视

1、更新所有实体的信息与矩阵

注入的线程中，每次循环更新所有实体的信息

```
void refreshEntityList()
{
    // 获取实体列表地址
    DWORD entityListBase = *(DWORD)*(DWORD*)entityListOffset;
    // 获取实体数量
    DWORD amountOfPlayers = *(DWORD*)(amountOfPlayersOffset);
    // 更新实体列表
    for (int i = 1; i < amountOfPlayers; i++)
    {
        DWORD entityBase = *(DWORD*)(entityListBase + 0x4 * i);
        if (entityBase != NULL)
        {
            entityList[i].x = *(float*)(entityBase + locationXOffset);
            entityList[i].y = *(float*)(entityBase + locationYOffset);
            entityList[i].z = *(float*)(entityBase + locationZOffset);
            entityList[i].headX = *(float*)(entityBase + headXOffset);
            entityList[i].headY = *(float*)(entityBase + headYOffset);
            entityList[i].headZ = *(float*)(entityBase + headZOffset);
            entityList[i].health = *(DWORD*)(entityBase + healthOffset);
            entityList[i].team = *(DWORD*)(entityBase + teamOffset);
            strcpy_s(entityList[i].name, (char*)(entityBase + nameOffset));
            refreshEntityTowards(player, entityList[i]);
        }
    }
    return;
}
```

更新矩阵的信息

```
// 获取视图矩阵
memcpy(&Matrix, (PBYTE*)(viewMatrixOffset), sizeof(Matrix));
```

2、通过矩阵3D坐标转2D坐标

利用如下函数将实体的头部坐标，脚坐标转为视野中的二维坐标

```

// 将世界坐标转换为屏幕坐标
bool WorldToScreen(Vec3 pos, Vec2 &screen, float matrix[16]) // 3D to 2D
{
    // 通过矩阵计算将世界坐标3D转换为剪辑坐标
    Vec4 clipCoords;
    clipCoords.x = pos.x * matrix[0] + pos.y * matrix[4] + pos.z * matrix[8] + matrix[12];
    clipCoords.y = pos.x * matrix[1] + pos.y * matrix[5] + pos.z * matrix[9] + matrix[13];
    clipCoords.z = pos.x * matrix[2] + pos.y * matrix[6] + pos.z * matrix[10] + matrix[14];
    clipCoords.w = pos.x * matrix[3] + pos.y * matrix[7] + pos.z * matrix[11] + matrix[15];

    if (clipCoords.w < 0.1f)
        return false;

    // 通过对剪辑坐标运算获取NDC坐标
    Vec3 NDC;
    NDC.x = clipCoords.x / clipCoords.w;
    NDC.y = clipCoords.y / clipCoords.w;
    NDC.z = clipCoords.z / clipCoords.w;

    // 将NDC坐标与分辨率运算获取到对应的屏幕2D坐标
    screen.x = (windowWidth / 2 * NDC.x) + (NDC.x + windowHeight / 2);
    screen.y = -(windowHeight / 2 * NDC.y) + (NDC.y + windowHeight / 2);
    return true;
}

```

首先通过矩阵乘法获取剪辑坐标

剪辑坐标 $x = a_0x + a_4y + a_8z + a_{12}w$
 剪辑坐标 $y = a_1x + a_5y + a_9z + a_{13}w$
 剪辑坐标 $z = a_2x + a_6y + a_{10}z + a_{14}w$
 剪辑坐标 $w = a_3x + a_7y + a_{11}z + a_{15}w$

再通过剪辑坐标运算得到NDC坐标。

矩阵的设计中w 是可以让剪辑坐标范围到-1和1的 也就成了NDC坐标

所以NDC坐标很好理解,就是 -1到1的平面坐标系 中心点为0,0

NDC.x = 剪辑坐标 x/剪辑坐标 w
 NDC.y = 剪辑坐标 y/剪辑坐标 w
 NDC.z = 剪辑坐标 z/剪辑坐标 w

在将NDC坐标转换为屏幕中的二维坐标。

若实体不在视野内，该函数返回false。

3、通过2D坐标绘制矩形方块与血量

```

// 头部高度
float head = vHead.y - vScreen.y;
// 宽度
float width = head / 2;
// 中心点
float center = width / -2;
// 头部上方额外区域
float extra = head / -6;
// 设置画刷颜色
Brush = CreateSolidBrush(RGB(158, 66, 244));
// 画人物框
DrawBorderBox(vScreen.x + center, vScreen.y, width, head - extra, 1);
DeleteObject(Brush);
// 将人物血量转换为字符串
char healthChar[255];
sprintf_s(healthChar, sizeof(healthChar), "%d", (int)(entity.health));
// 标记人物血量
DrawString(vScreen.x, vScreen.y, TextCOLOR, healthChar);
// 画线
DrawLine(vScreen.x, vScreen.y);

```

绘制人物方框、血量和线。

三、不通过矩阵实现透视

1、更新玩家与所有实体信息

```

void refreshPlayer()
{
    // 获取玩家基本信息
    player.x = *(float*)((DWORD)((DWORD*)playerBaseOffset) + locationXOffset);
    player.y = *(float*)((DWORD)((DWORD*)playerBaseOffset) + locationYOffset);
    player.z = *(float*)((DWORD)((DWORD*)playerBaseOffset) + locationZOffset);
    player.headX = *(float*)((DWORD)((DWORD*)playerBaseOffset) + headXOffset);
    player.headY = *(float*)((DWORD)((DWORD*)playerBaseOffset) + headYOffset);
    player.headZ = *(float*)((DWORD)((DWORD*)playerBaseOffset) + headZOffset);
    player.health = *(DWORD)((DWORD)((DWORD*)playerBaseOffset) + healthOffset);
    player.team = *(DWORD)((DWORD)((DWORD*)playerBaseOffset) + teamOffset);
    player.yaw = *(float*)((DWORD)((DWORD*)playerBaseOffset) + yawOffset);
    player.pitch = *(float*)((DWORD)((DWORD*)playerBaseOffset) + pitchOffset);
    strcpy_s(player.name, (char*)((DWORD)((DWORD*)playerBaseOffset) + nameOffset));
    return;
}

```

```

void refreshEntityList()
{
    // 获取实体列表地址
    DWORD entityListBase = *(DWORD)(*(DWORD *)entityListOffset);
    // 获取实体数量
    DWORD amountOfPlayers = *(DWORD*)(amountOfPlayersOffset);
    // 更新实体列表
    for (int i = 1; i < amountOfPlayers; i++)
    {
        DWORD entityBase = *(DWORD*)(entityListBase + 0x4 * i);
        if (entityBase != NULL)
        {
            entityList[i].x = *(float*)(entityBase + locationXOffset);
            entityList[i].y = *(float*)(entityBase + locationYOffset);
            entityList[i].z = *(float*)(entityBase + locationZOffset);
            entityList[i].headX = *(float*)(entityBase + headXOffset);
            entityList[i].headY = *(float*)(entityBase + headYOffset);
            entityList[i].headZ = *(float*)(entityBase + headZOffset);
            entityList[i].health = *(DWORD*)(entityBase + healthOffset);
            entityList[i].team = *(DWORD*)(entityBase + teamOffset);
            strcpy_s(entityList[i].name, (char*)(entityBase + nameOffset));
            refreshEntityTowards(player, entityList[i]);
        }
    }
    return;
}

```

需要获取玩家的坐标，水平视角，垂直视角。

还需要获取实体的坐标，头部坐标，相对于玩家的水平视角，水平视角差，垂直视角，垂直视角差。

2、通过坐标将3D坐标转换为2D坐标

利用如下函数将实体的头部坐标，脚坐标转为视野中的二维坐标

```

bool WorldToScreenWithoutMatrix(Vec2& screen, float yawDiff, float pitchDiff)
{
    // 计算可视角度
    float visibleAngle = (float)((double)atan2((double)windowHeight, (double)windowWidth) * 180.0 / PI);
    // 若不可见则返回false
    if (fabs(yawDiff) > 45 || fabs(pitchDiff) > visibleAngle)
        return false;
    // 计算水平差
    int diff1 = (int)(tan(yawDiff * PI / 180) * (windowWidth / 2));
    // 计算窗口中x坐标
    screen.x = (float)(windowWidth / 2 + diff1);
    // 计算垂直差
    int diff2 = (int)(tan(pitchDiff * PI / 180) * (windowHeight / 2));
    // 计算窗口中y坐标
    screen.y = (float)(windowHeight / 2 + diff2);
    return true;
}

```

若不在可视范围内则直接返回false。

3、通过2D坐标绘制矩形方块与血量

```

// 头部高度
float head = vHead.y - vScreen.y;
// 宽度
float width = head / 2;
// 中心点
float center = width / -2;
// 1/3宽度
float extra = head / -6;
// 设置画刷颜色
Brush = CreateSolidBrush(RGB(158, 66, 244));
// 画人物框
DrawBorderBox(vScreen.x + center, vScreen.y, width, head - extra, 1);
DeleteObject(Brush);
// 将人物血量转换为字符串
char healthChar[255];
sprintf_s(healthChar, sizeof(healthChar), " %d", (int)(entity.health));
// 标记人物血量
DrawString(vScreen.x, vScreen.y, TextCOLOR, healthChar);
// 画线
DrawLine(vScreen.x, vScreen.y);

```

由于获取了头部2D坐标与躯干2D坐标，同上方法绘制矩形，血量与线。

四、实现自瞄

1、更新玩家与实体信息

同上。

2、计算离玩家鼠标最近的敌人。

通过遍历实体列表，计算在画面中距离玩家视角水平方向上最近的敌人。

```

// 获取实体列表地址
DWORD entityListBase = *(DWORD)*(DWORD*)entityListOffset;
// 获取实体数量
DWORD amountOfPlayers = *(DWORD*)(amountOfPlayersOffset);
// 自动瞄准
Entity *closestEntity = NULL;
for (int i = 1; i < amountOfPlayers; i++)
{
    DWORD entityBase = *(DWORD*)(entityListBase + 0x4 * i);
    if (entityBase != NULL)
    {
        // 计算准星角度差
        Entity *entityPtr = &entityList[i];
        float difference = fabs(entityPtr->HAngularDifference.x);
        if (closestEntity != nullptr)
        {
            if (entityPtr->team != player.team && entityPtr->health > 0 && entityPtr->health < 100 && difference < fabs(closestEntity->HAngularDifference.x))
                closestEntity = entityPtr;
        }
        else
        {
            if (entityPtr->team != player.team && entityPtr->health > 0 && entityPtr->health < 100)
                closestEntity = entityPtr;
        }
    }
}

```

3、调整玩家视角

若最近的敌人存活，则将玩家视角强制修改为敌人方向，实现自瞄

函数如下：

```

if (closestEntity != nullptr)
{
    // 设置玩家yaw
    *(float*)(DWORD)(*(DWORD*)playerBaseOffset) + yawOffset) = closestEntity->HAngular.x;
    // 设置玩家pitch
    *(float*)(DWORD)(*(DWORD *)playerBaseOffset) + pitchOffset) = closestEntity->HAngular.y;
}

```

4、细节调整

此功能可设置为玩家左键设计时再锁定敌人头部，实现更优良的体验。

五、注入DLL

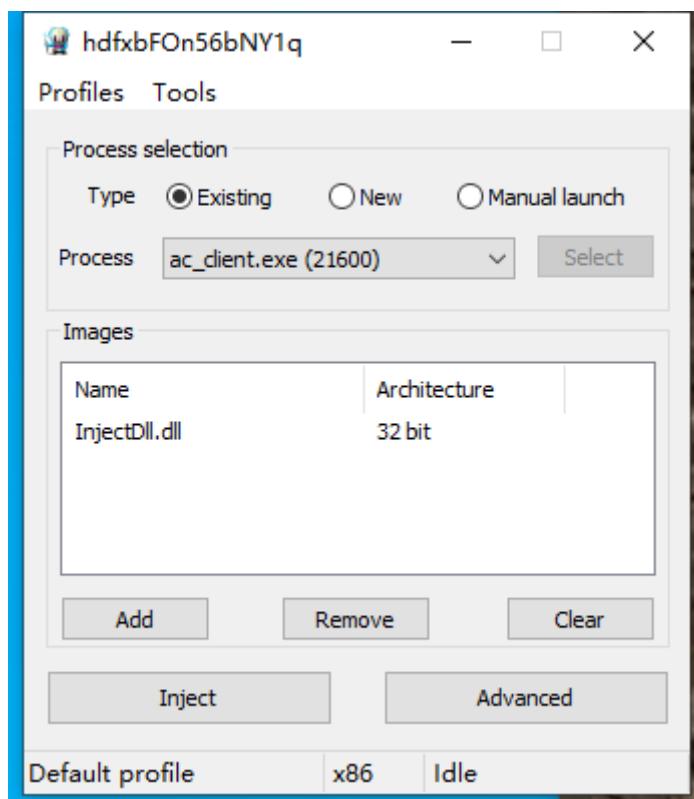
1、编译DLL

```

1>InjectDll.vcxproj -> C:\Users\22057\Documents\Study\Class\GameSecurity\Homework7\src\InjectDll\Debug\InjectDll.dll
1>已完成生成项目“InjectDll.vcxproj”的操作。
===== 版本: 1 成功, 0 失败, 0 更新, 0 跳过 =====
===== 占用时间 00:02.679 =====

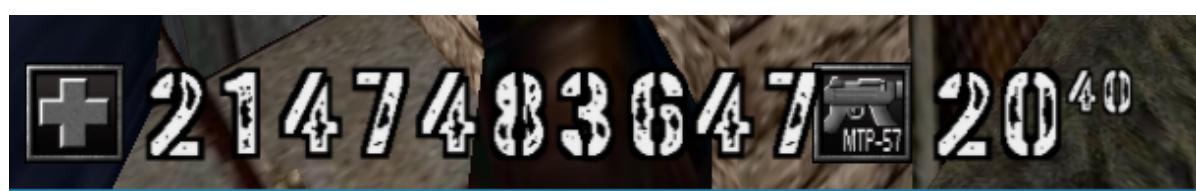
```

2、使用注射器注入DLL



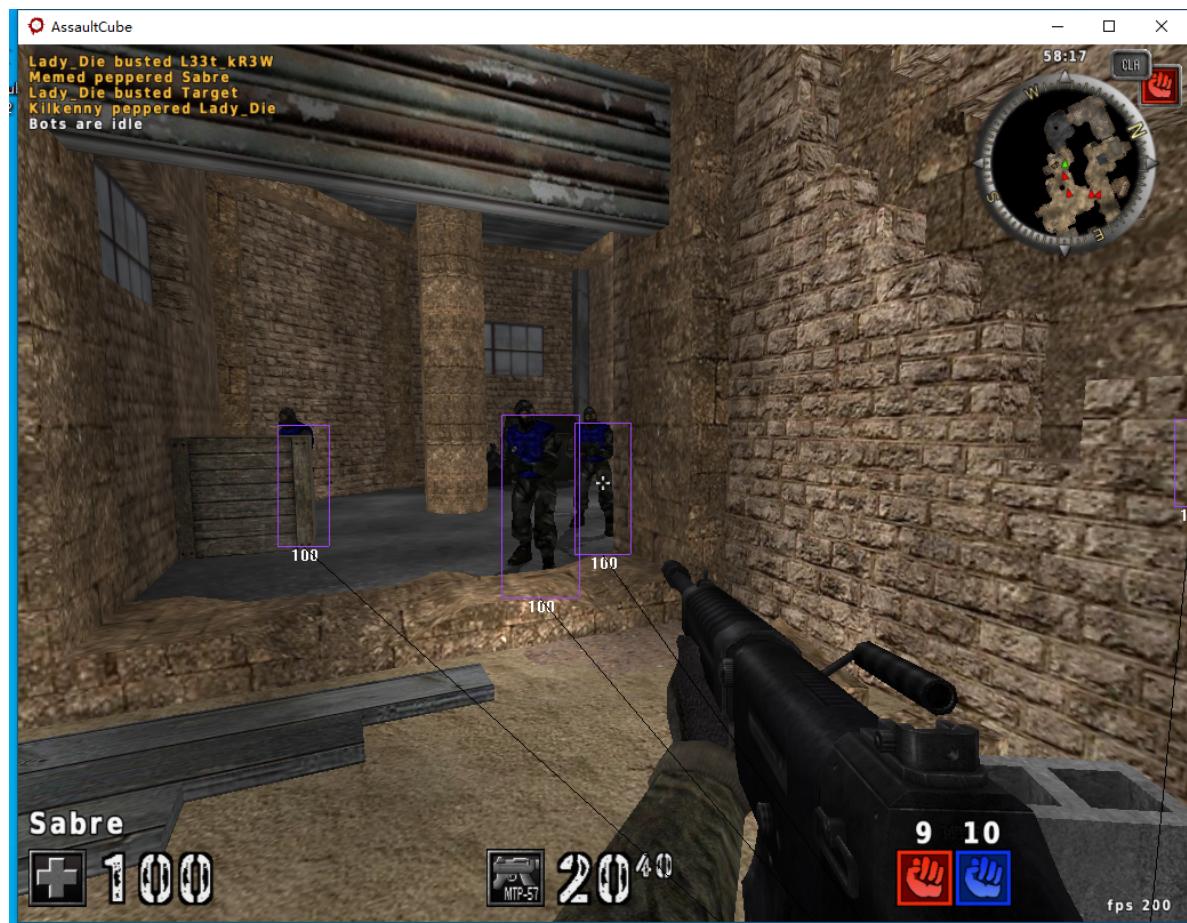
六、测试

1、F1:锁血

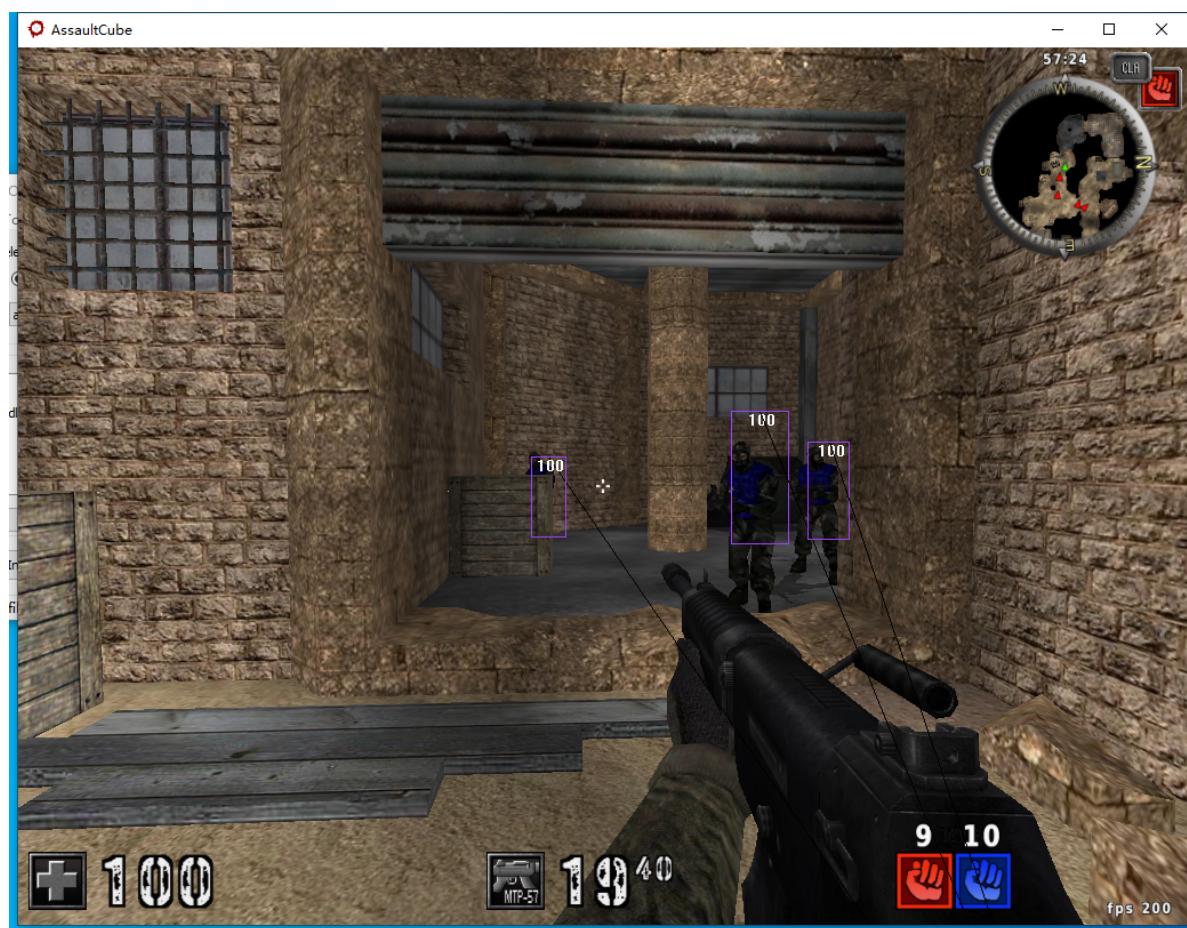


按下F1后，血量锁定并修改

2、F2:透视(矩阵)



3、F3:透视(非矩阵)



效果较差

4、F4自瞄



自动瞄准头部