方框透视的原理是通过读取游戏中已知坐标数据，并使用一定算法将自己与敌人之间的距离计算出来，结合GDI绘图函数在窗体上直接绘制图形，直到现在这种外挂依然具有极强的生命力，原因就是其比较通用，算法固定并能够应用于大部分的FPS游戏中。

寻找游戏坐标数据

找本人坐标数据: 通常情况下(X,Y)坐标的浮动较大不好定位，我们可以找Z坐标因为Z坐标控制人物的高低参数比较好找。

1.首先搜索未知初始值(浮点数)，然后找到一个梯子，向上走搜索增加的数值，向下搜索减少的数值。
2.重复这个过程最后就能找到Z轴的坐标，在游戏中(X,Y,Z)坐标是紧挨着的结构(+0,+4,+8) 找到了Z坐标相应的就可以每次减4计算出(X,Y)坐标。

> 本人坐标x = server.dll+4F2FEC + 288 - 8
> 本人坐标y = server.dll+4F2FEC + 288 - 4
> 本人坐标z = server.dll+4F2FEC + 288

找敌人坐标数据: 上方我们找到了自己的XYZ坐标，接着我们可以通过自己的Z坐标定位到队友的Z坐标。

sv_cheats 1 开启创造模式 bot_stop 1 暂停机器人运动 bot_add-ct 添加反恐精英队友

1.首先打开游戏控制台输入命令，添加一个队友，然后暂停其走动，此时我们和队友站在一个高度上。
2.搜索未知初始值(浮点数)，然后紧接着使用精确扫描，扫描地址为我的坐标z的地址。
3.继续搜索，比如说我在下坡，队友在上坡，此时我的Z坐标数据肯定是队友小，那么相对的他的坐标比我大，大出来的部分就是我的坐标数据。

所以我们搜索浮点数，选择【值大于】大于的数值就是我们自己的坐标z，然后让自己比队友高，并搜索【值小于】，依次遍历最后可确定坐标数据。

> 敌人x = server.dll+4F2FFC + 288 - 8
> 敌人y = server.dll+4F2FFC + 288 - 4
> 敌人z = server.dll+4F2FFC + 288

找自己的鼠标角度: 通常FPS游戏鼠标的准心Y坐标向上抬会减少，鼠标准心向下会增加，不断的遍历(浮点数)就可以搜索到鼠标的准心Y坐标，得到了鼠标的Y坐标之后然后+4就能得到鼠标的X的坐标参数。

1.打开CE进入游戏，将鼠标放置在屏幕的中心位置，直接搜索【未知初始化数据】(浮点数)，然后将游戏鼠标向上微抬，回到CE搜索【减少的数值】多次向上抬并搜索减少的数值。
2.接着将鼠标逐步向下微压，回到CE然后搜索【增加的数值】这里要重复十几次，最后不要动鼠标直接搜索【未变动的数值】即可找到以下结果，这里都是基地址选哪一个都可以。

> 只可以读出坐标数据
> 鼠标 x = engine.dll+61D254 + 4
> 鼠标 y = engine.dll+61D254
> 鼠标 x = client.dll+4C0300 + 4
> 鼠标 y = client.dll+4C0300

> 可设置鼠标位置
> 鼠标 x = engine.dll +4622CC +4
> 鼠标y = engine.dll + 4622CC

找FOV视场角: 视场角又称FOV，视场角的大小决定了摄像机的视野范围，简单来说FOV就是屏幕与摄像机之间的夹角，我们可以通过狙击枪的狙击镜来找到游戏的视场角度，

当未开镜状态时搜索未知初始化数据(浮点数)，开镜后搜索改变的数值(浮点数)，依次遍历即可找到该游戏的视场角度，一般的FPS游戏视场角为90度的居多。

1.直接开找，打开CE和游戏，购买一把狙击枪，然后在CE中搜索【未知的初始值】，注意这里要选择浮点数搜索。
2.回到游戏，打开狙击枪的一倍狙击镜，在CE中搜索【变动的数值】，接着打开二倍狙击镜，继续搜索【变动的数值】，最后关闭狙击镜搜索【变动的数值】，该过程要重复10次左右。
3.此时狙击镜处于关闭状态，直接搜索【未变动的数值】，然后拔出你的手枪，搜索【未变动的数值】因为手枪的视野与未开镜状态下的狙击枪是一样的，这样搜索能够尽量排除干扰，从而更精确的筛选到我们所需要的数据。
4.经过了上方的遍历以后，结果已经不多了，我们可以猜测这个角度应该在【0-180度】之间，所以通过【介于两者之间】再次筛选一下结果，之后就可以看到有两个90度的角，而且是绿色的基址，一般情况下开发人员默认会将这个角设置为45，75或90度。

> FOV = client.dll+5046F0
> FOV = client.dll+504628
> FOV = client.dll+5047B8
> FOV = client.dll+50489C
> FOV = engine.dll+3C1720

取当前玩家数量: 玩家数量的查找非常简单，大部分的FPS游戏都有人物统计菜单，按下TAB键则可看到，我们可以通过查看人物数量来查找。

第一次搜索1（4字节），然后按下+号添加1个机器人搜索2，再次添加一个机器人搜索3，不断遍历即可得到玩家数量。

> 算上我自己的人机数量: server.dll+4EEFE8
> 算上我自己的人机数量: engine.dll+5D29BC

> 不算我自己的人机数量: server.dll+4EEFE0
> 不算我自己的人机数量: server.dll+588878

找敌人血量: 找敌人血量的目的是为了判断敌人是否死亡，这里找血量有两种方式，第一种是找自己的血量，由于我们知道敌人的基地址，所以找到自己血量的偏移就可以通过敌人的基地址定位到敌人血量上。

第二种找法是默认搜索100，然后打敌人一枪搜索减少的数值，然后搜索未变动的数值，再次打敌人一枪搜索减少的数值，不断的遍历最后就能找到我们想要的敌人的血量

自己血量偏移 = e4

> 自己血量: server.dll + 54B6C8 + e4
> 自己血量: server.dll + 54A82C + e4
> 自己血量: server.dll + 4F2FEC + e4

找到自己血量偏移，配合敌人基地址，即可得到敌人的血量，当血量等于1时表示敌人已经死亡。

> 敌人血量: server.dll + 4F2FFC + e4

找阵营: 首先进入游戏，按下M键可以切换阵营，通过不断地切换，然后搜索即可找到标识本人阵营的基地址与偏移地址，然后通过其他人的基地址加我们自己的偏移，即可得到其他人的阵营数据，一般阵营会用(1,2,3,4)这些数字来表示。

1.首先选择搜索未知初始值（4字节），然后搜索未变动的数值，此时切换阵营，搜索变动的数值，以此循环最终即可找到本人阵营。

> 本人阵营: server.dll+4F2FEC + 1F4
> 敌人阵营: server.dll+4F2FFC + 1F4

> 本人阵营: server.dll+54A82C + 1F4
> 本人阵营: server.dll+54B6C8 + 1F4

找相机矩阵: 找矩阵的方法就是不断移动自己相机位置，最好拿把狙击枪，然后开镜搜索变动的数值，移动身体搜索变动数值，或者是开镜移动身体搜索变动的数值，这样配合来找，最终可以锁定在2000个数值左右，然后就可以开找。

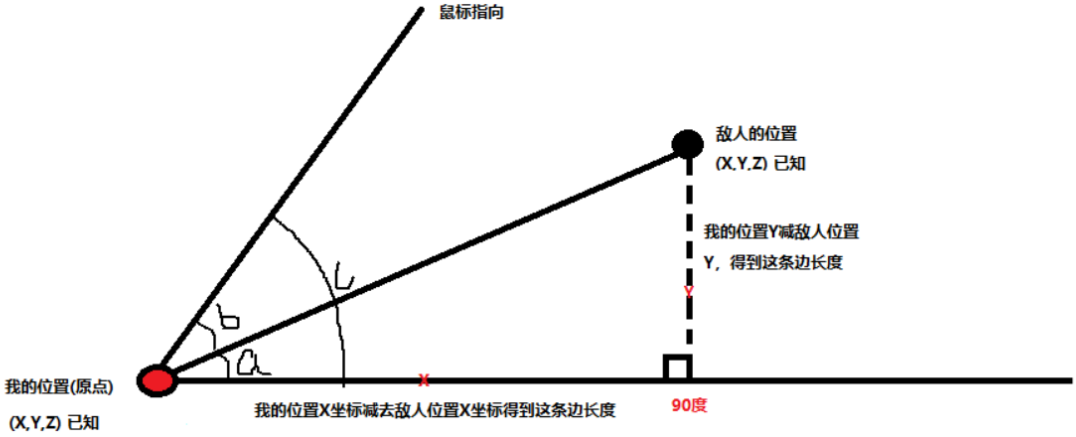
```
----------------------------------------------------------------------
【4x4 竖矩阵】

0.74      -0.09         0.00        -124.05
0.07       0.54         1.22         -1924. 48
0.11       0. 91       -0. 41       -2700. 54
0.11       0.91        -0. 41       -2692. 87

【4x4  横矩阵】

0.74         -0.09             0. 00              0.91
0.07          0.54              1.22             -0.41
0.11          0.91             -0.41             -0.41
-124.05    -1924. 48    -2700. 54       -2692. 87

----------------------------------------------------------------------
【3x4  横矩阵】

0.74          -0.09           0.00            0.0
0.07           0.54            1.22            0.0
0.11           0.91           -0.41           0.0
-124.05     -1924.48    -2700.54     1.00

【3x4  竖矩阵】

0.74          -0.09          0.00          -124.05
0.07          0.54           1.22           -1924.48
0.11          0.91           -0.41          -2700.54
0.0            0.0              0.0             1.0
----------------------------------------------------------------------
```

找敌人之间的数组偏移： 在前面我们已经找到了第一个敌人的数据【server_css.dll+3D24E4】指向的就是第一个敌人的地址，通过与偏移【15B8】相加就能得到X坐标，在此基础上加4就能得到Y坐标，显然该游戏并不会将玩家数据放到偏移中，很有可能每个敌人分别占用一个地址，我们可以通过使用内存遍历工具，找到第二个敌人的地址，然后用第2个敌人的地址减去第1个敌人的地址就能得到敌人与敌人之间的差值。

方框透视算法分析

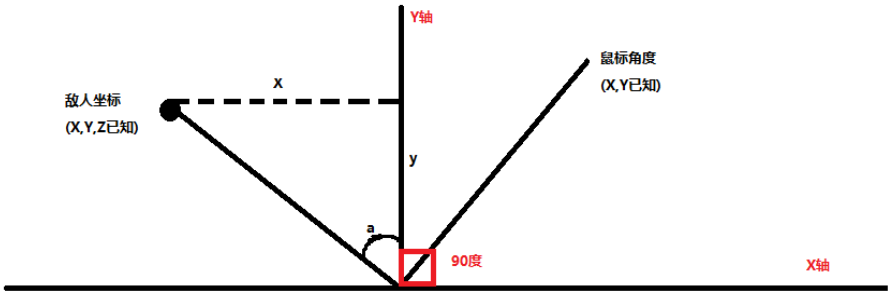
在前面的教程中我们已经手动找到了【FOV视场角】【本人坐标数据】【本人鼠标角度】【敌人坐标数据】【玩家数量】【玩家是否死亡】【敌人之间的数组偏移】这些基址数据，多数情况下类FPS游戏找坐标手法都大同小异，接下来我们将具体分析计算方框的思路，以及实现这些方框绘制算法。

第一象限求角： 假设敌人在第一象限，求鼠标指向与敌人之间的夹角b，可以使用反正切求导。



我们知道自己与敌人的相对(X,Y)距离，可以使用反正切公式求出a角的度数。而我们最终的目的是要求出我们的鼠标指向与敌人之间的夹角b，此时我们可以通过已知的鼠标角度C减去a既可得到b的角度。

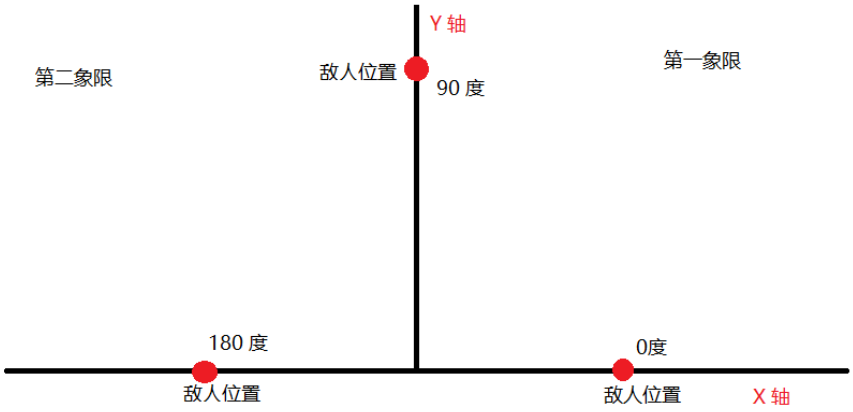
第二象限求角： 假设敌人在第二象限，而我们的鼠标依然指向在第一象限，求敌人与X轴之间的夹角度数。



如上图：由于(X,Y)(黑色)是已知条件，我们可以通过X比Y求反正切，即可得到a角的度数，然后与90度相加，即可求出敌人当前坐标位置与X轴之间的夹角度数。

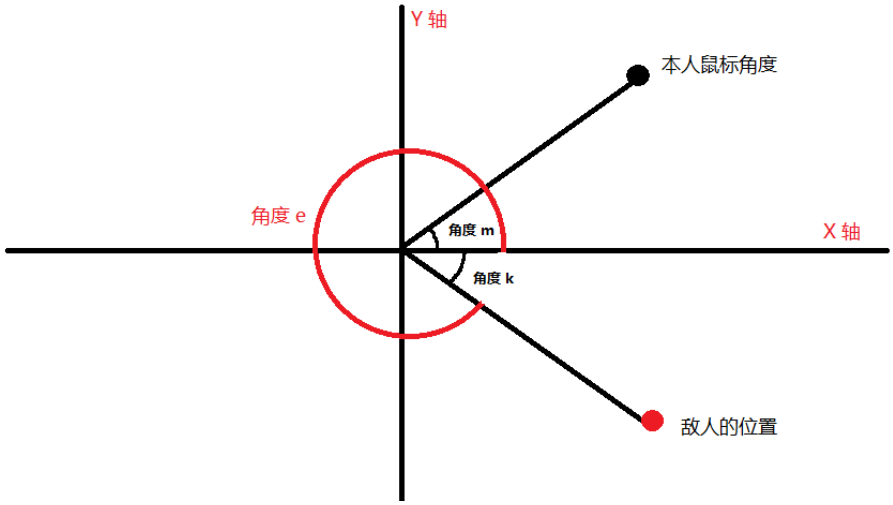
第三四象限： 敌人在第三与第四象限与上图差不多，最终目的就是求敌人的位置与X轴之间的夹角，第三象限应该加180度，第四象限加上270度数。这里就不罗嗦了，很简单的东西。

另外4种特殊情况： 如果敌人在第一象限且与X轴重合，那么敌人与X轴为之间的夹角度数必然为零度，同理如果与Y轴重合的话，那么X轴与敌人之间的夹角度数为90度，以此类推就是这四种特殊情况。

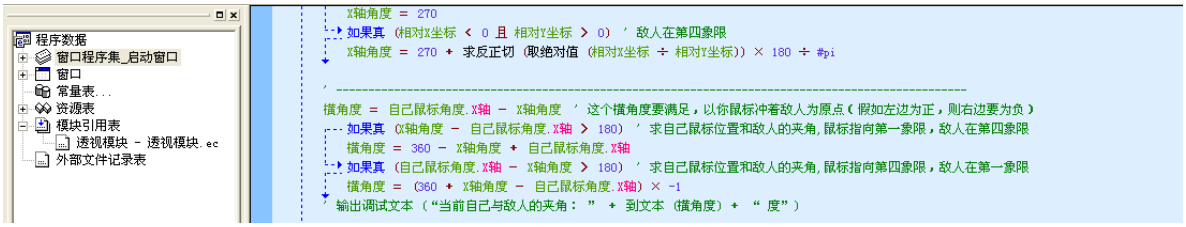


上方的4条象限与特殊情况，如果展开的话一共是8种不同的情况，如下代码就是这八种不同情况，调试下面的这段代码会发现一个缺陷，那就是当我们绕着敌人转圈时，偶尔会出现一个大于180度的角度，这又是两种非常特殊的情况。

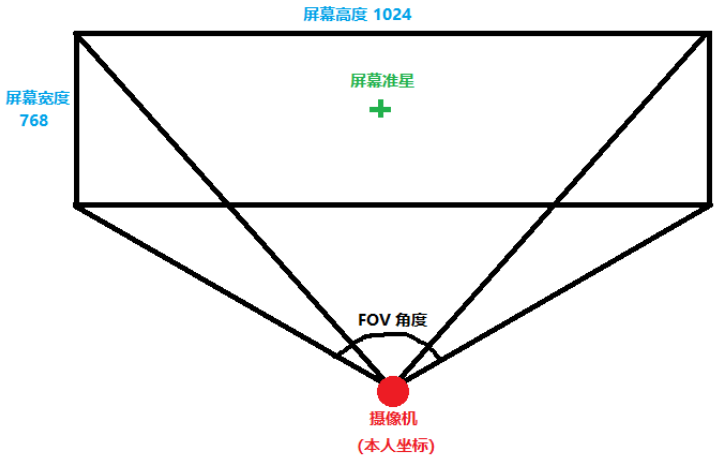
特殊情况： 当敌人在第四象限且鼠标角度依然在指向第一象限的情况下，则会出现大于180度的角。



如上图：我们的目标是求鼠标角度与敌人之间的夹角度数，而此时的鼠标指向第一象限，而敌人却在第四象限上，我们用360度减去e角度（e = 敌人坐标与x轴之间的夹角度数），即可得到K角度，用K角度加上M角度，即可得到鼠标与敌人之间的夹角度数，另一种特殊情况敌人与鼠标角度调换位置求角，最终代码如下：

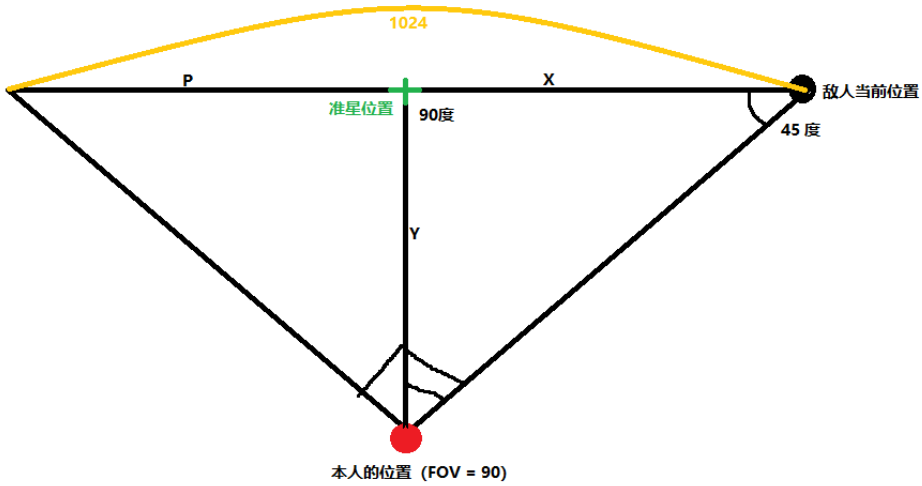


FOV视场角度： 摄像机的作用就是，移动游戏中的场景，并将其投影到二维平面，显示给玩家。

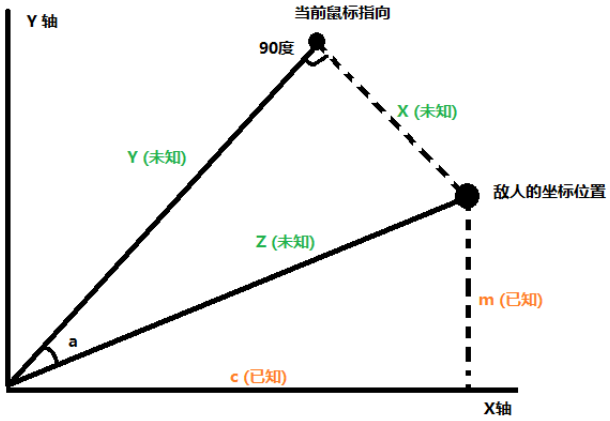


如上图：摄像机与屏幕之间的夹角统称为视场角，游戏中的准星位置到屏幕的边缘是FOV的一半，以屏幕分辨率1024x768为例，当FOV为90度时，则准心与屏幕的垂线构成45度等腰直角三角形，此时的摄像机到屏幕的距离就是一半屏幕长度（1024/2 = 512）的大小。

三维横坐标转屏幕X坐标： 将三维矩阵中的敌人坐标数据，转换为屏幕的X坐标。

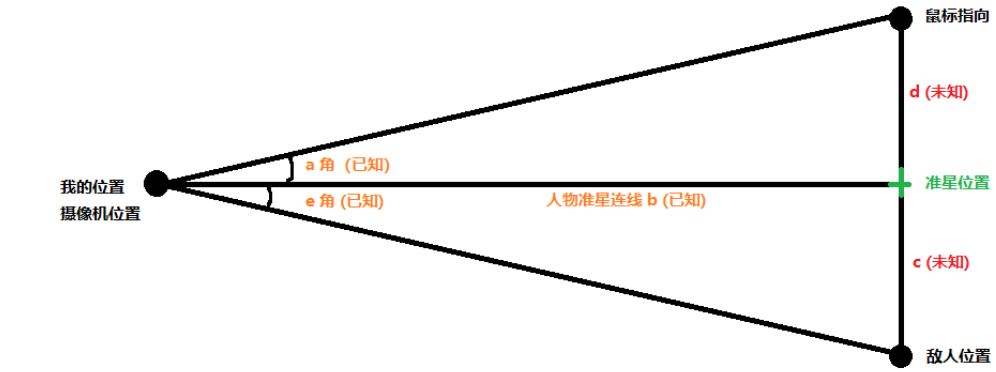


如上图：我们需要求出敌人位置的坐标数据，可以使用 `(x/y) x (1024/2)` 最后还需要加上P的长度，由于窗口的总长度是1024那么我们可以直接除以2得到另一半的长度(512)，将敌人位置与另一半长度相加就是敌人投射在屏幕上的X坐标，但是此时我们并不知道(X,Y)的长度，所以需要先求出(X,Y) 如下图所示。



上图中：我们需要求出(X,Y)的距离，此时我们已经知道了M和C的长度，则此时我们可以直接使用勾股定理 `M的平方 + C的平方 （开方）= Z`，得到Z之后，通过 `sin a = (x/z) => sin a * z = X` 此时我们已经得到的X的长度，接着 `cos a =(y/z) => cos a * z = Y` 此时我们也得到了Y的长度，最后 `(x/y) x 512 + 512` 即可得到敌人位置，投射到屏幕上的X坐标。

三维纵坐标转屏幕Y坐标： 三维横坐标搞懂了，这个纵坐标就更简单了，如下图：



tan a = d/b => tan a * b = d
tan e = c/b => tan e * b = c
坐标Y：(tan a + tan e) * b + 屏幕高度一半

上图中：通过tan公式即可推导出d与c的距离，然后将d与c的长度相加，即可得到鼠标指向与敌人位置之间的距离，然后再加上屏幕高度的一半，本游戏屏幕高度为768，所以要加上384即可。

最终屏幕横坐标与纵坐标的转换算法如下所示