# HUD的使用

### 绘制函数DrawHUD

HUD类中的主要绘制函数，无法直接在蓝图中覆盖（蓝图中的绘制函数不是此函数）。绘制采用每帧均进行调用此函数的方式进行用户界面刷新（如不重新调用，则会导致屏幕绘制内容被游戏绘制覆盖，HUD绘制次序永远在最后），结合Canvas指针对象可以进行用户UI界面的基本绘制

**DrawHUD中切记在DrawHUD函数里调用父类函数**

DrawHUD函数是循环函数，不要在DrawHUD中进行复杂逻辑编写，或是死循环

### Canvas画笔指针

ClipX获取屏幕宽度

ClipY获取屏幕高度

### 绘制文字

UE字体支持

·TTF字体（TrueType Font）由微软和苹果公司共同研发的字体，中文名称全真字体。特点兼容性强，不依赖硬件（算法字体），可被存储传递，支持广泛的硬件设备（屏幕，打印机）

·OTF字体（OpenType Font）由微软和Adobe研发的一种字体，比TTF字体更加强大。支持很大的字符集，并且提供版权保护，是T1和TTF字体的超集。

优点

1. 更加强大的跨平台特性
2. 对国际字符集更好的支持
3. 支持高级印刷控制能力
4. 生成的文件尺寸更小
5. 支持在字符加入数字签名，保证文件的集成功能

基本文字绘制

成员函数DrawText即可绘制文字

参数

·绘制文字 FString

·FlinerColor 颜色

·位置X

·位置Y

·字体 默认可不填

·缩放 绘制运算缩放非字体计算缩放

·位置是否随缩放变化

如想改变字体颜色可以直接修改默认Canvas绘制颜色语法如下



此函数内部使用的是Canvas的DrawItem实现与下面的DrawItem如出一辙

Canvas绘制文字

函数Canvas->DrawText

参数

·字体指针

·绘制内容FString

·位置X

·位置Y

·X轴缩放

·Y轴缩放

·字体渲染信息FFontRenderInfo 裁切阴影

Canvas函数DrawItem

此函数可以绘制继承自FCanvasItem类的对象，如字体，形状，贴图，材质

如绘制文字需要使用FCanvasTextItem对象构建

语法



构建完成后调用Canvas->DrawItem();

FText可以完成格式化文本，语法如下



如想调整字中的默子集和大小可以使用FSlateFontInfo进行构建，使用需要引入模块Slate

"Slate", "SlateCore"

语法

FSlateFontInfo info(font, 30, "Font\_0");

FCanvasTextItem item(FVector2D(200, 50), text1, info, FLinearColor::Yellow);

Canvas->DrawItem(item);

参数

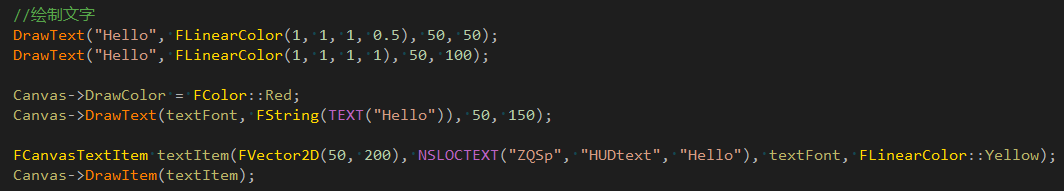
·字体指针

·字号

·字体集名称

然后再构建FCanvasItem，语法如上，将字体指针更换到FSlateFontInfo

完整语法



### FlineColor FColor

FlineColor更高级一些，支持逻辑算法，重载了基本运算符可进行颜色线性的转换

FColor基本的颜色数据存储对象，只有颜色转换逻辑

### 绘制贴图

函数Canvas->DrawTile简单绘制函数

将UTexture绘制到屏幕上

语法



参数

·材质贴图

·绘制坐标X

·绘制坐标Y

·绘制宽度

·绘制高度

·UV中的X坐标

·UV中的Y坐标

·UV的宽度（不是百分比值）

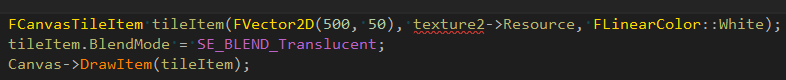
·UV的高度（不是百分比值）

·混合模式

使用FCanvasTileItem进行绘制

FCanvasTileItem有多个构造函数进行构建

示例



### 绘制材质

通过UMaterialInterface指针进行材质获取（也可以使用UMaterial）只能获取材质实例内容

使用FCanvasTileItem进行基本绘制

**注意材质的材质域必须是UserInterface**

构建函数参数5

·屏幕绘制位置FVECTOR2

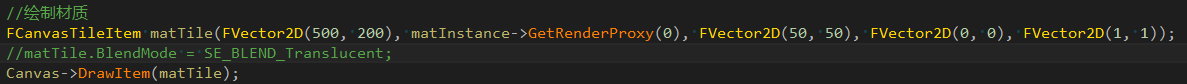
·材质绘制代理

·绘制尺寸

·绘制UV坐标

·绘制UV坐标系数，区别于宽高

使用DrawItem绘制即可



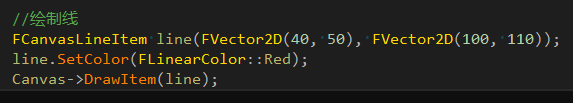
### 绘制Icon

构建FCanvasIcon（结构体在UCanvas类内部）此类包装的是UTexture对象指针，操作UV信息

### 绘制图形

###### 线

FCanvasLineItem

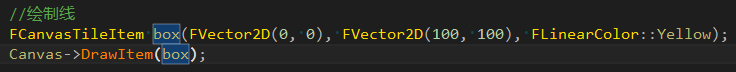


DrawLine函数

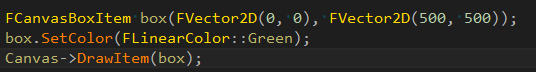
Draw3DLine 绘制3D线

Draw3DLine(FVector(0, 0, 0), FVector(500, 500, 500), FColor::Red);

###### 矩形填充



线框



# 交互

获取按键输入

通过获取PlayerController的方式，在帧函数中进行按键事件判定



获取鼠标输入



获取鼠标坐标信息



### 便捷鼠标事件交互

可以通过开启EnableInput，使用HUD自定义的点击事件进行用户输入收集

**注意如要使用回调需要保证以下操作有效**

·设置PlayerController的bEnableClickEvents为true

**设置代码**

APlayerController\* pc = GetWorld()->GetFirstPlayerController();

pc->bEnableClickEvents = true;

函数AddHitBox

参数列表

·圈定起始点

·圈定大小

·圈定标记 FName

·点击事件是否向下传递 默认flase

·层级优先级排序，默认0

放到DrawHUD中切记在DrawHUD函数里调用父类函数

RenderHitBoxes绘制碰撞框

在DrawCanvas中调用

RenderHitBoxes(Canvas->Canvas);

可接收事件

点下 NotifyHitBoxClick(FName BoxName);

抬起 NotifyHitBoxRelease(FName BoxName);

鼠标进入 NotifyHitBoxBeginCursorOver(FName BoxName);

鼠标移出 NotifyHitBoxEndCursorOver(FName BoxName);

### 在A类中响应按键图表中的绑定按键

调用A类的EnableInput函数，传入Playercontroller，即可完成。

**注意：直接绑定输入事件**

调用EnableInput函数后，将自动为类添加一个InputComponent组件，可以直接使用此组件进行输入事件绑定。如下

InputComponent->BindAction(TEXT("DoJump"), IE\_Pressed, this, &AMyActor::CallClick);