评价标准是：

本方：英雄自身偏向:输出HOTPT、生存HLIVE

本方：输出：物理伤害HATK、魔法伤害HMAG、真实伤害HHRT

本方：防御：护甲偏高HDEF、魔抗偏高HMDEF、均衡HLIVE

敌方：输出：物理伤害HATK、魔法伤害HMAG、真实伤害HHRT

敌方：防御：护甲偏高HDEF、魔抗偏高HMDEF、均衡HHP

输出结果：

物理装推荐：物理ATK、魔法MAG、穿透Decay

防御装推荐：护甲DEF、魔抗MDEF、生命值HP

装备推荐项：优先输出OTPT、优先生存LIVE

完全不需要考虑性能损失，仅来实现功能

打开UE，在CPP\_BaseCharacter上加上变量，来描述对该英雄的评价标准

对单一英雄的评价标准有2\*3+2种描述：

输出评价：物理伤害、魔法伤害、真实伤害

防御评价：护甲偏高、魔抗偏高、均衡

自身偏向：输出、生存

创建三个Enum

UENUM(BlueprintType)

enum class ELegendComment:uint8

{

    HOTPT UMETA(DisplayName = "HOTPT"),

    HLIVE UMETA(DisplayName = "HLIVE")

};

UENUM(BlueprintType)

enum class EOutputComment:uint8

{

    HATK UMETA(DisplayName = "HATK"),

    HMAG UMETA(DisplayName = "HMAG"),

    HHRT UMETA(DisplayName = "HHRT"),

};

UENUM(BlueprintType)

enum class ELiveComment:uint8

{

    HDEF UMETA(DisplayName = "HDEF"),

    HMDEF UMETA(DisplayName = "HMDEF"),

    HHP UMETA(DisplayName = "HLIVE"),

};

实现评价标准

首选选择一个标准模板

按照

HP：2000

DEF：50

MDEF：50

ATK：100

MAG：100

按照上述数据作为评价标准（这个是可优化的）

在public中，创建变量

UPROPERTY(EditAnywhere,BlueprintReadWrite,Category="CharacterComment")

ELegendComment LegendComment;

UPROPERTY(EditAnywhere,BlueprintReadWrite,Category="CharacterComment")

EOutputComment OutputComment;

UPROPERTY(EditAnywhere,BlueprintReadWrite,Category="CharacterComment")

ELiveComment LiveComment;

英雄自身偏向和英雄本身输出：

USTRUCT(BlueprintType)

struct FCharacterInfomation

{

GENERATED\_BODY()

// 结构中的每个成员变量都应当有UPROPERTY标记，否则无法在蓝图细节中展开层级进行赋值。

    UPROPERTY(EditAnywhere, BlueprintReadWrite,Category="CharacterInfo")

    ELegendComment  LenendComment;

    UPROPERTY(EditAnywhere, BlueprintReadWrite,Category="CharacterInfo")

    EOutputComment  DamageKind;

};

我们将英雄自身偏向放在CharacterInfomation结构中

UPROPERTY(EditAnywhere,BlueprintReadWrite,Category="CharacterInfo")  
FCharacterInfomation CharacterInfo;

然后获得英雄的其他属性评价

创建函数，Public：

Void SetCharacterAttributeComment（）

英雄评价标准：

实现评价标准

首选选择一个标准模板

按照

DEF：50

MDEF：50

ATK：100

MAG：100

按照上述数据作为评价标准

当英雄身上某数值高于这一标准数值时，标记为可能值

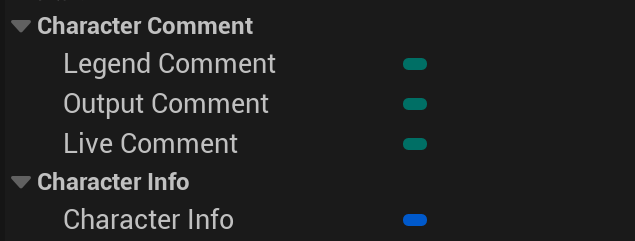
让同一类别的两种数值都高于这一标准数值时，则选择差值最高的这一种。

这个算法可以优化：

直接将两个属性值和对应的标准值做差，得到的如果都是负值则为枚举第三个变量或使用CharacterInfo变量中的DamageKind变量，否则进行评价。

这个函数的实现在蓝图中实现，而不是在CPP中实现。因为我不会。

首先进入BP\_BaseCharacter，我们可以看到



然后在C++中声明函数

UFUNCTION(BlueprintImplementableEvent,Category="AttributeComment")  
void SetCharacterAttributeComment();

BlueprintImplementableEvent表示函数没有C++实现，完全由蓝图定义逻辑。

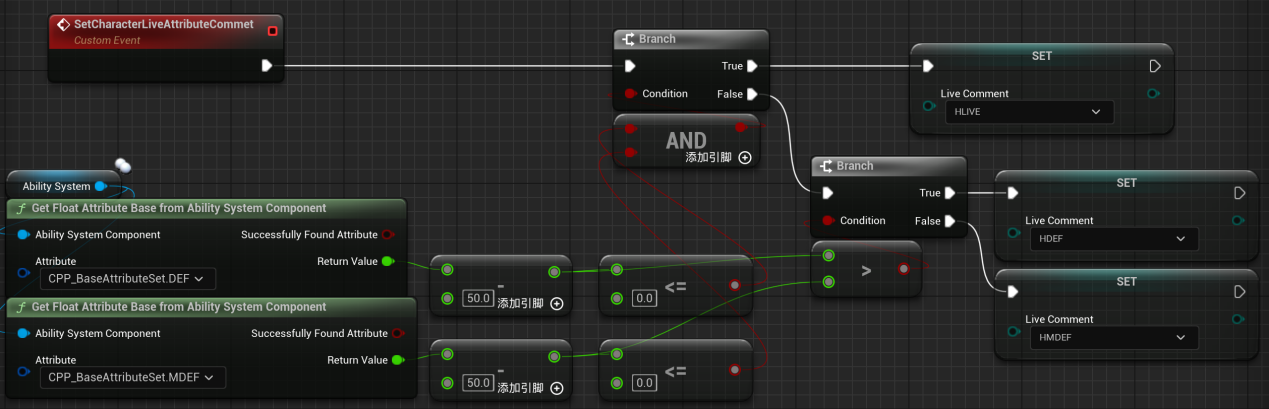
然后进入BP\_BaseCharacter，重载该函数

（绷不住了哥们，在C++里实现了函数之后，这个函数是Event类型的，而且不能在其他地方调用）

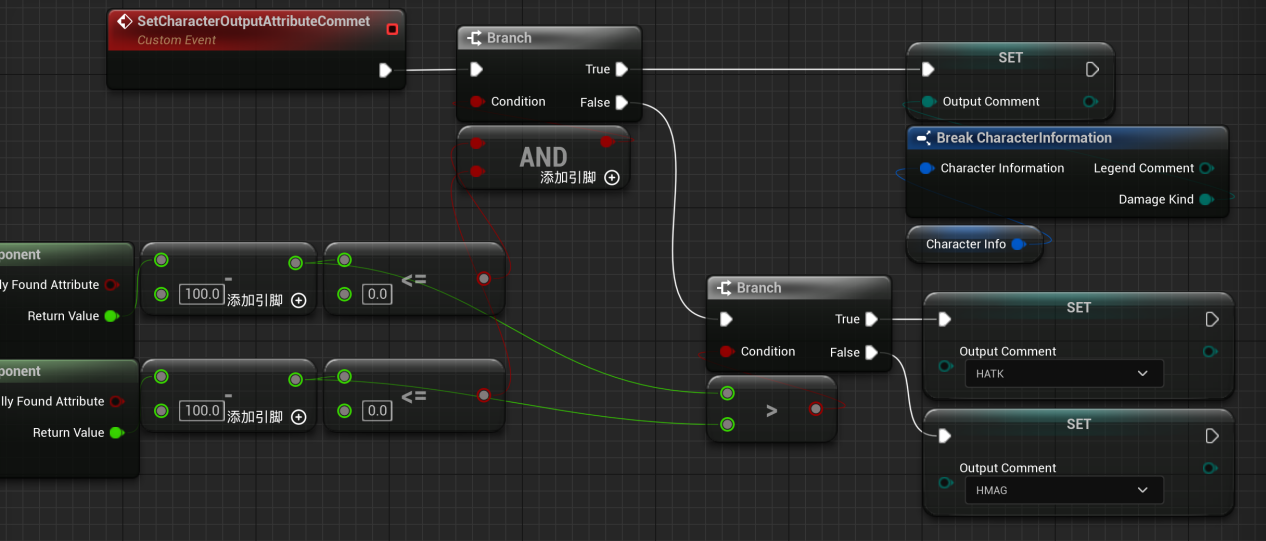
删掉C++中的函数声明，在BP\_BaseCharacter中创建和实现函数。

这个函数分成两个部分：SetOutputComment和SetLiveComment

对于LiveComment比较容易实现



对于OutputComment就需要判断角色本身的DamageKind了



行吧，现在考虑到以后的大量的CPP的工作，还是得在CPP里实现Set函数

AttributeSet中每个Attribute都是一个FGameplayAttributeData类型的数据，这个数据有两个值：CurrentValue和BaseValue，这两个值都是float型的。我们只需要获得这两个值就行。

但是GAS有独立的能力，直接给出数据。

void ACPP\_BaseCharacter::SetCharacterAttributeComment()

{

    TObjectPtr<UAbilitySystemComponent> OwnedAbilitySystemComponent=this->FindComponentByClass<UAbilitySystemComponent>();

    const UCPP\_BaseAttributeSet\* OwnedAttributeSet=OwnedAbilitySystemComponent->GetSet<UCPP\_BaseAttributeSet>();

    float CurrentATK = OwnedAttributeSet->GetATK();

}

这时就可以进行数值大小比较了。

void ACPP\_BaseCharacter::SetCharacterAttributeComment()

{

    TObjectPtr<UAbilitySystemComponent> OwnedAbilitySystemComponent=this->FindComponentByClass<UAbilitySystemComponent>();

    /\*

    if(OwnedAbilitySystemComponent)

    {

        //OwnedAbilitySystemComponent->GetAttributeSet(UCPP\_BaseAttributeSet)

        const UCPP\_BaseAttributeSet\* OwnedAttributeSet=OwnedAbilitySystemComponent->GetSet<UCPP\_BaseAttributeSet>();

    }

    \*/

    const UCPP\_BaseAttributeSet\* OwnedAttributeSet=OwnedAbilitySystemComponent->GetSet<UCPP\_BaseAttributeSet>();

    float CurrentATK = OwnedAttributeSet->GetATK();

    float CurrentMAG = OwnedAttributeSet->GetMAG();

    float CurrentDEF = OwnedAttributeSet->GetDEF();

    float CurrentMDEF = OwnedAttributeSet->GetMDEF();

    // Live part

    float SubDEF=CurrentDEF-50;

    float SubMDEF=CurrentMDEF-50;

    if(SubDEF<=0 && SubMDEF<=0)

    {

        LiveComment=ELiveComment::HHP;

    }

    else

    {

        if(SubDEF>SubMDEF)  LiveComment=ELiveComment::HDEF;

        else LiveComment=ELiveComment::HMDEF;

    }

    // Output part

    float SubATK=CurrentATK-100;

    float SubMAG=CurrentMAG-100;

    if(SubATK<=0 && SubMAG<=0)

    {

        OutputComment=CharacterInfo.DamageKind;

    }

    else

    {

        if(SubATK>SubMAG)   OutputComment=EOutputComment::HATK;

        else OutputComment=EOutputComment::HMAG;

    }

}

这样就完成了属性评价。

然后就要考虑到，什么时候对该变量进行设置。

不考虑性能的来说，直接在EventTick上调用，真TM爽。

吊毛，放Tick上还不行，有bug

那就在BP\_BaseCharacter在OnEquipmentUpdate的时候