先解决一个新问题

当SetTrees函数在CPP\_BaseCharacter的构造函数里面调用的时候，无法正常启动UE

两个方法：

1. 在CPP\_Character的beginplay里调用
2. 在BP\_Character的构造函数里调用

这里选择前者。因为后者有不可避免的问题。（TSharedPtr不能作为蓝图函数的任何可能存在）

（更正：在修复计算熵的函数之后，SetTrees函数在CPP\_BaseCharacter的构造函数里进行调用）

Predict函数很简单啊，返回值甚至都是FString类型的

调用Predict也很简单，

FString str = Predict(Trees[0],sample,attributes);

就是很简单。

这里要关注的是sample参数，它是需要我们在游戏中动态赋值的。

为了更准确的描述函数运行，创建FDataRow类型的变量，替代sample作为参数传递

UPROPERTY(BlueprintReadWrite)

FDataRow PredictComment;

这个变量需要两个参数才能进行设置：Self和TargetActor。

UPROPERTY(BlueprintReadWrite)

FDataRow PredictComment;

UFUNCTION(BlueprintCallable)

void SetPredictComment(ACPP\_BaseCharacter\* TargetActor);

首先创建一个变量来存储TargetActor

UPROPERTY(BlueprintReadWrite)

FDataRow PredictComment;

UPROPERTY(BlueprintReadWrite)

ACPP\_BaseCharacter\* AttackTargetActor;

UFUNCTION(BlueprintCallable)

void SetPredictComment(ACPP\_BaseCharacter\* TargetActor);

也就是说，我们需要一种方式能够得到TargetActor，然后将TargetActor的OutputComment和LiveComment参数传递到Self这里，给Comment进行赋值。

不要担心结构类没有实现赋值的方法。

FDataRow sample = {

    {"HOTPT","HATK","HDEF","TargetActor->OutputComment","HDEF"},

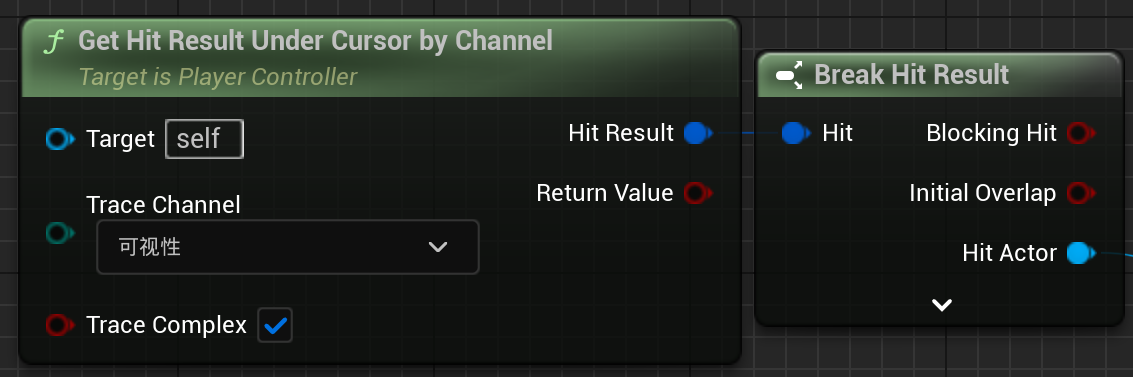
    {}

};

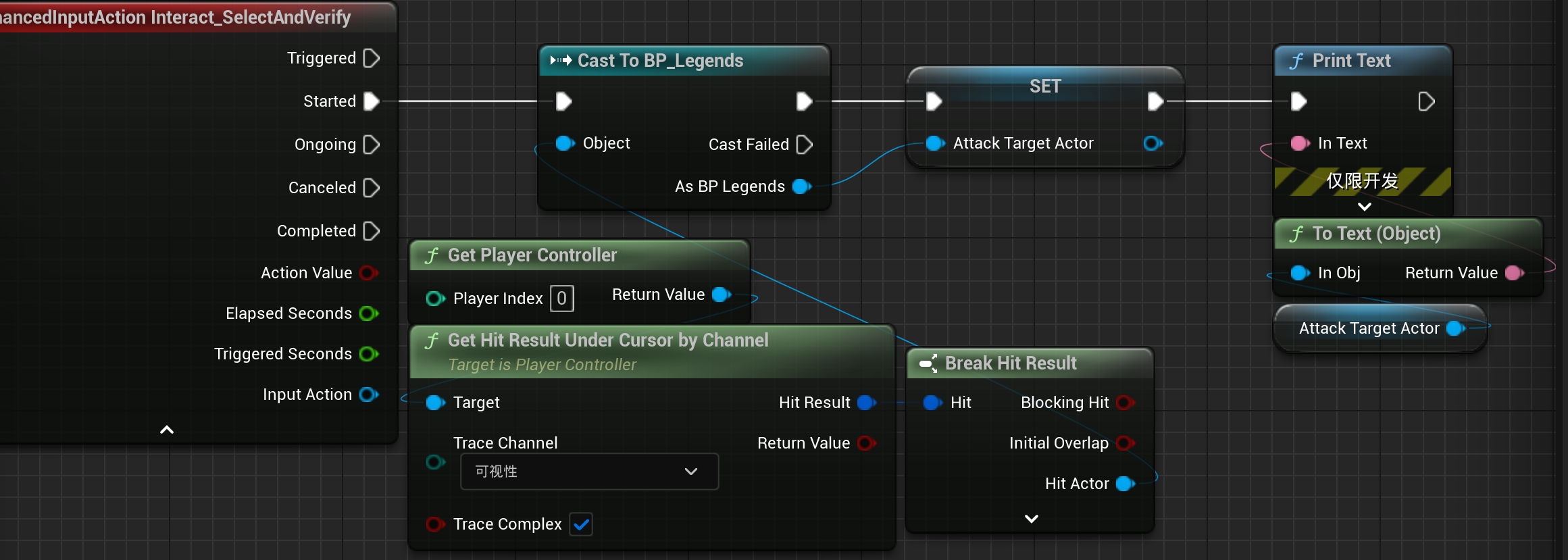
尽可能地转换就行了。

然后我们要想办法得到TargetActor。是通过鼠标指针？还是什么其他的方法？

没错，就用鼠标指针的方法



当鼠标左键点击时，我们会检测HitActor是不是BP\_Legend，如果是的话，就将这个HitActor设置成AttackTargetActor



只能说非常成功

我们在每个英雄在购买装备的时候调用SetCharacterAttributeComment函数

// 这个函数是设置这个对象本身的AttributeComment的函数

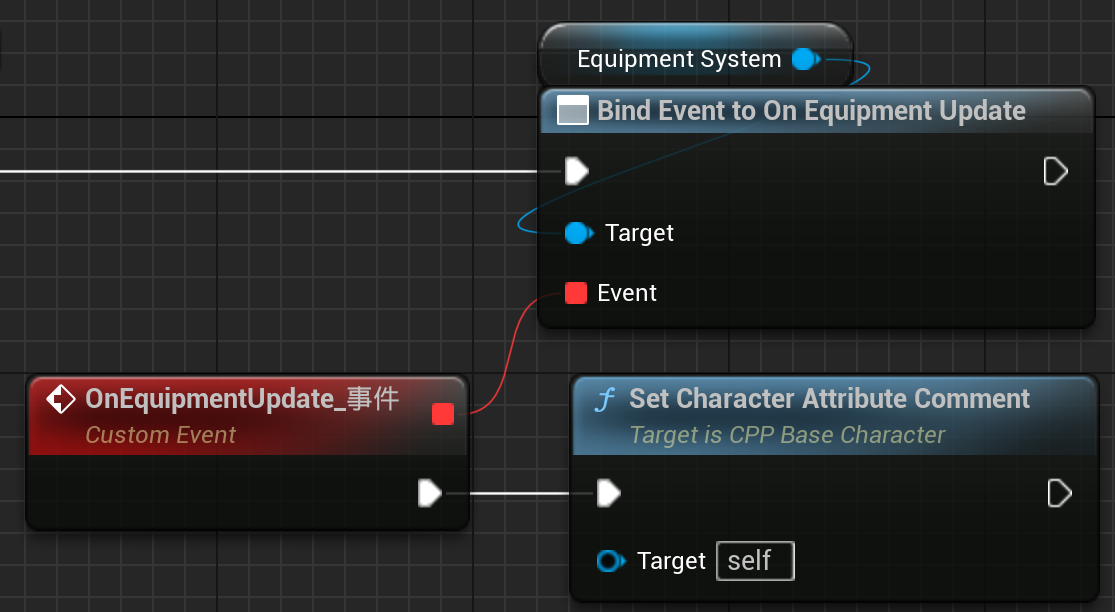
// 是用来给LegendComment、OutputComment和LiveComment赋值的函数（LegendComment是默认的，一般不会变）

UFUNCTION(BlueprintCallable)

void SetCharacterAttributeComment();

这样得到的self和TargetActor的三个评价

都是最新的



然后要做的就是

调用SetPredictComment函数

在SetPredictComment时，我们得到的是枚举值

void ACPP\_BaseCharacter::SetPredictComment(ACPP\_BaseCharacter\* TargetActor)

{

    TargetActor->OutputComment;

    TargetActor->LiveComment;

    this->OutputComment;

    this->LiveComment;

}

为了将枚举值转成FString值，需要实现函数  
inline FString EnumToString(ELiveComment EnumValue)

{

    const UEnum\* EnumPtr = StaticEnum<ELiveComment>();

    if (!EnumPtr)

    {

        return FString("Invalid Enum");

    }

    return EnumPtr->GetNameStringByValue(static\_cast<int64>(EnumValue));

}

然后实现SetPredictComment函数

void ACPP\_BaseCharacter::SetPredictComment(ACPP\_BaseCharacter\* TargetActor)

{

    FString OwnLegendComment=EnumToString(this->LegendComment);

    FString OwnOutputComment=EnumToString(this->OutputComment);

    FString OwnLiveComment=EnumToString(this->LiveComment);

    FString TargetOutputComment=EnumToString(TargetActor->OutputComment);

    FString TargetLiveComment=EnumToString(TargetActor->LiveComment);

    this->PredictComment = {

        {OwnLegendComment,OwnOutputComment,OwnLiveComment,TargetOutputComment,TargetLiveComment},

        {}

    };

}

最后一步了

我们创建函数，来调用Predict函数，并返回预测的结果

创建函数CallPredictAndGetAdvice

这一步甚至相当简单

FString ACPP\_BaseCharacter::CallPredictAndGetAdvice(int32 TreeIndex, FDataRow TwoLegendComment)

{

    return Predict(Trees[TreeIndex],TwoLegendComment,attributes);

}

最后就是要考虑，CallPredict函数和SetPredictComment函数各自在什么时候使用。

我们可以在CallPredict里面调用一次SetPredictComment。

这样就需要加上一个TargetActor参数，而且性能可能较差。

但是捏，我才不管你性能，哈哈哈哈哈哈哈哈哈

好吧，还是怂了

将SetPredictComment函数放在CallPredict函数里面进行调用

没办法，想得太多，太牛逼了这Lizzy

FString ACPP\_BaseCharacter::CallPredictAndGetAdvice(int32 TreeIndex, ACPP\_BaseCharacter\* TargetActor,FDataRow TwoLegendComment)

{

    SetPredictComment(TargetActor);

    return Predict(Trees[TreeIndex],TwoLegendComment,attributes);

}

得到建议结果Very Easy，

