GYGameplayEffectDataTable

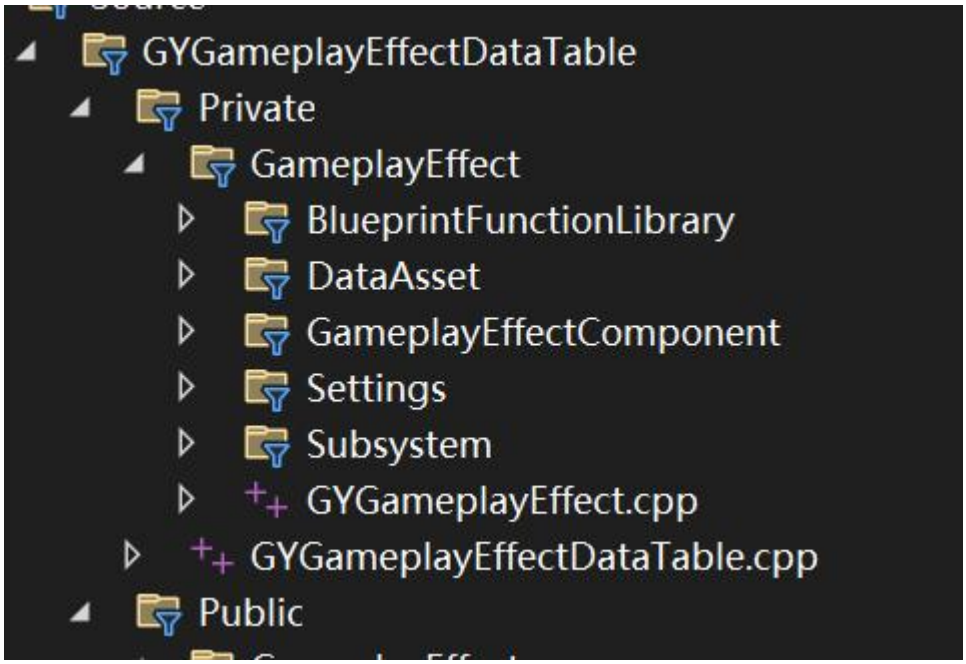
目录

序章：

GYGameplayEffectDataTable 插件可以让我们通过 DataTable 来使用 GE。同时也支持自定义 UGameplayEffect 类型，GE 组件和 DataTable 类型。通过接口可以很方便的让我们触发 GE 的效果。

这样有利于 GE 的统一管理，同时也减少了重复创建 GE 蓝图的繁琐，更加有利于项目的维护。

插件目录：



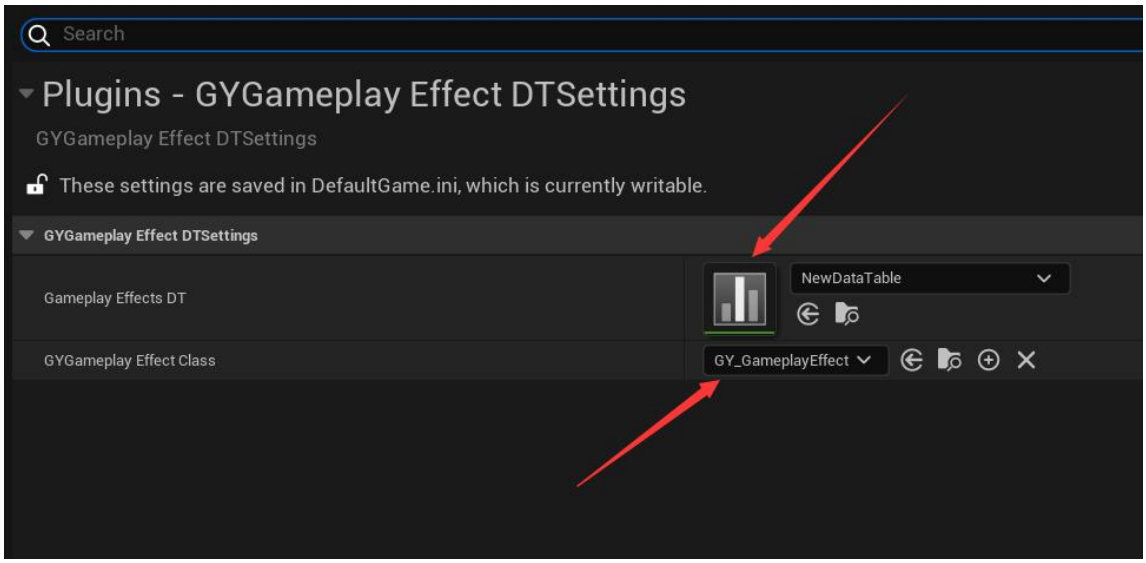
从上到下，BlueprintFunctionLibrary 里是我们的 GE 触发接口等库。DataAsset 里存放了我们 GE 的 DataTable 的结构体。GameplayEffectComponent 存放了 GE 组件的数据。Settings

里是我们项目设置类。Subsystem 是我们插件的管理器类。

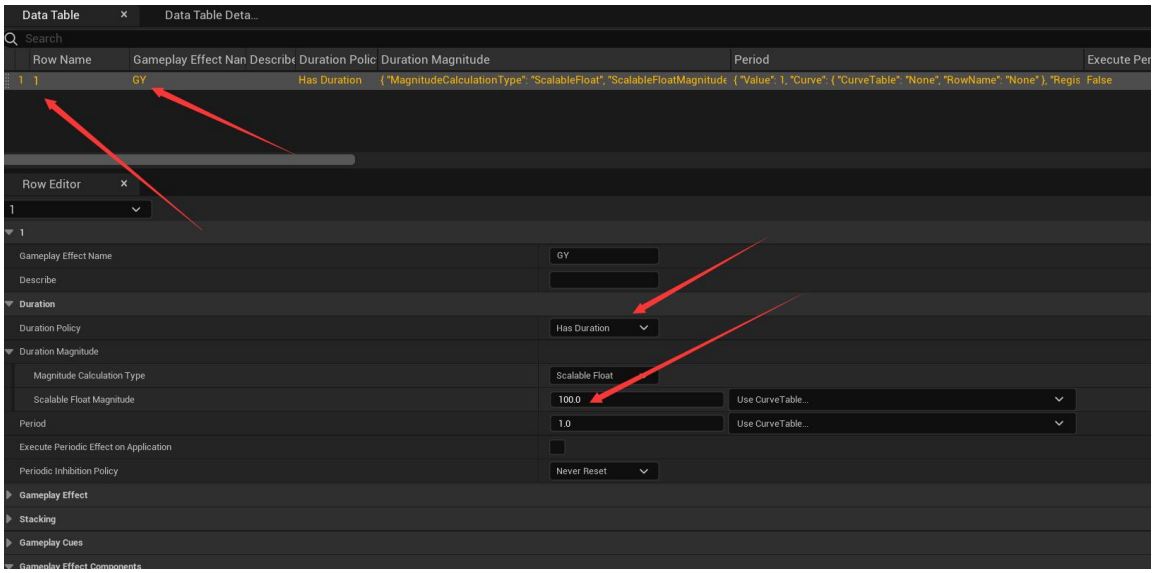
使用流程：

```
 *
 */
UCLASS()
class GY_API UGY_GameplayEffectSubsystem : public UGYGameplayEffectSubsystem
{
    GENERATED_BODY()

private:
    virtual void Initialize(FSubsystemCollectionBase& Collection) override;

    virtual void PostInitGameplayEffect(uint8* GameplayEffectsRowDT, UGYGamepla
```

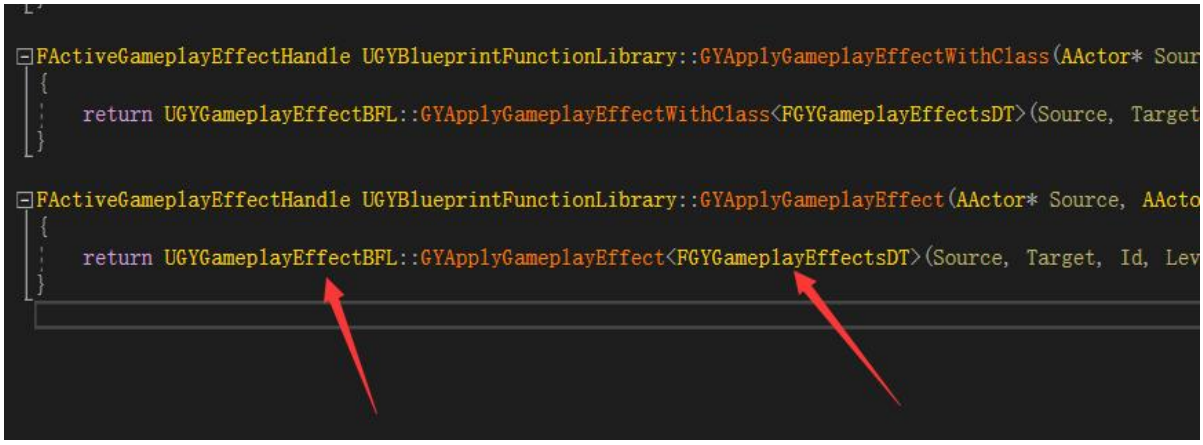
首先我们在自己的项目中继承 UGYGameplayEffectSubsystem，生成一个 GE 的子系统。然后打开编辑器创建一个 FGY_GameplayEffectsDT 的 DataTable。



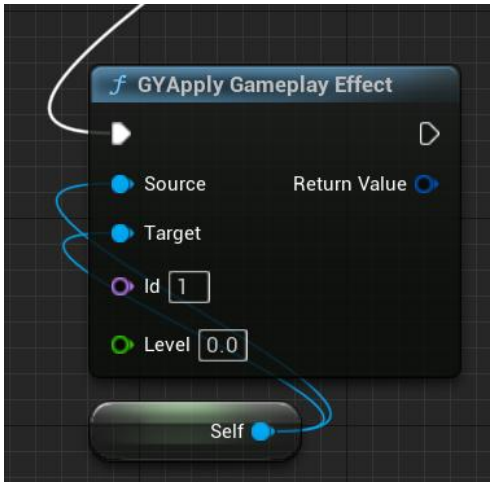
把 DataTable 配置到插件的项目设置中。



打开 DataTable 按图中箭头配置一下，RowName 为 1。RowName 是我们触发这个 GE 的标识符。



在自己的项目库中去封装一下 UGYGameplayEffectBFL::GYApplyGameplayEffect 静态模板函数。注意 FGYGameplayEffectsDT 得是我们使用的 DataTable 类型。



然后我们就可以这样简单的通过 id 触发我们的 GE 了。

自定义 DataTable:

直接把 FGY_GameplayEffectsDT 中的成员变量全部复制粘贴到一个新的 DataTable 类中。

这里不用继承的原因是为了保证变量在 DataTable 中的顺序一致，同时保证以后导出 Excel 的变量顺序一致。当然如果你不考虑变量属性美观，也可以直接继承 FGY_GameplayEffectsDT。

这里不封装成#define 宏的原因是，虚幻引擎的 UHT 会比 C＋＋的预处理还要早，会导致 UPROPERTY 等虚幻引擎的宏失效。

```
USTRUCT(BlueprintType)
struct FGYGameplayEffectsDT : public FTableRowBase
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY(BlueprintReadWrite, EditDefaultsOnly)
        FName GameplayEffectName;

    UPROPERTY(BlueprintReadWrite, EditAnywhere)
        FString Describe;
```

```
27      UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "GameplayEffectComponents")
28          TArray<FGY_TargetTagRequirementsGECInfor> GETargetTagRequirementsGameplayEffectCompone
29
30      UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "GameplayEffectComponents")
31          TArray<FGY_TargetTagsGameplayECInfor> GETargetTagsGameplayEffectComponents;
32
33
34      UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "GameplayEffectComponents")
35          TArray<FGY_VelocityGameplayEffectComponentInfor> GEVelocityGameplayEffectComponents;
36  };
37
```

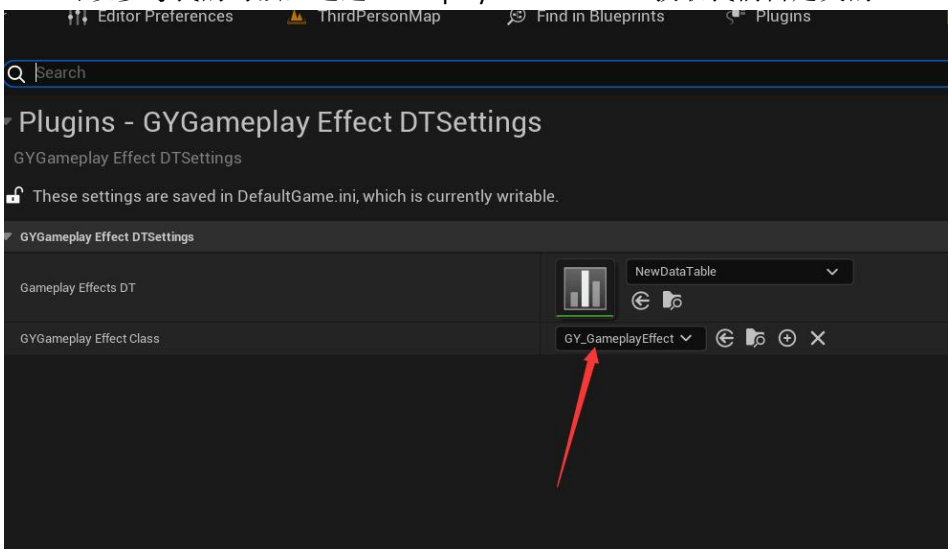
FGYGameplayEffectsDT 就是我复制粘贴 FGY_GameplayEffectsDT 自定义的 DataTable，同时向里面增加了自己的自定义 GE 组件 GEVelocityGameplayEffectComponents。

```
10  /**
11   *
12   */
13  UCLASS()
14  class GY_API UGY_GameplayEffectSubsystem : public UGYGameplayEffectSubsystem
15  {
16      GENERATED_BODY()
17
18  private:
19      virtual void Initialize(FSubsystemCollectionBase& Collection) override;
20
21      virtual void PostInitGameplayEffect(uint8* GameplayEffectsRowDT, UGYGameplayEffect* Out
22
23      void Add_GEVelocityGameplayEffectComponents(TArray<FGY_VelocityGameplayEffectComponentI
24
25  };
```

通过重写 UGYGameplayEffectSubsystem 中的 PostInitGameplayEffect 虚函数，我们可以处理我们自己的 DataTable 中自定义的数据。

```
void UGY_GameplayEffectSubsystem::PostInitGameplayEffect(uint8* GameplayEffectsRowDT, UGYGameplayEffect* OutGameplayEffect, FName RowID)
{
    Super::PostInitGameplayEffect(GameplayEffectsRowDT, OutGameplayEffect, RowID);

    if (GameplayEffectsRowDT)
    {
        FGYGameplayEffectsDT* GameplayEffectsDTRow = UGYGameplayEffectBFL::CastFStruct<FGYGameplayEffectsDT>(GameplayEffectsRowDT);
        if (!GameplayEffectsDTRow)
        {
            return;
        }

        Add_GEVelocityGameplayEffectComponents(GameplayEffectsDTRow->GEVelocityGameplayEffectComponents, OutGameplayEffect);
    }
}
```

可以参考我的写法，通过 GameplayEffectsRowDT 获取我们自定义的 DataTable 数据。



通过继承 UGY_GameplayEffect 可以自定义我们的 UGYGameplayEffect 类型，然后配置到上图。UGYGameplayEffectSubsystem 中的 PostInitGameplayEffect 虚函数中的传入参数 OutGameplayEffect 就是我们自定义的 UGYGameplayEffect。简单的 Cast 就可以实现自定义逻辑处理。

总结：

插件完全免费，欢迎大家使用和提意见。