**游 戏 存 档**

同乐技术

# 目 录

[目 录 2](#_Toc5919)

[前 言 2](#_Toc26590)

[快速开始 3](#_Toc4922)

[游戏存档基本逻辑 5](#_Toc21642)

[统计信息 5](#_Toc12765)

[相关函数、同步和异步的保存与加载 6](#_Toc13258)

[GameInstance游戏实例 7](#_Toc10054)

[检查点 10](#_Toc19519)

[删除存档文件 12](#_Toc23932)

[保存在关卡中生成的对象 12](#_Toc28151)

[正在保存的UI指示器 14](#_Toc25052)

[多存档 15](#_Toc96)

[存档逻辑 15](#_Toc14253)

[多存档的直观显示 15](#_Toc5349)

[增加新功能时不破坏旧存档 16](#_Toc210)

[C++存档与保存游戏 17](#_Toc19326)

[建议 18](#_Toc13459)

[参考文档 19](#_Toc183)

# **前 言**

此文档仅供学习使用，请不要在已成型的项目中间按照文档顺序对项目进行增加。因为增加新功能时不能破坏旧存档，如果按照文档流程，在后面实现多存档时，必然会破坏旧存档。

# 快速开始

UE5如何存储游戏数据

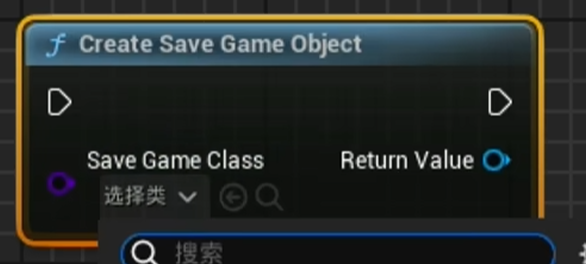
关键就是SaveGame类蓝图

先在SaveGame类中创建变量，这些变量是你要存储的所有数据

关键是，怎么把别的蓝图中的数据传递到SaveGame类中

函数：CreateSaveGameObject

新建一个空白的SaveGame对象以设置数据，然后传至SaveGameToSlot



这里选择我们创建的SaveGame类

然后就可以通过返回值得到自定义的SaveGame中的相关变量。我们可以轻易的Set这个变量。

在SaveGame中所有变量都设置好后，调用CreateSaveGameObject返回值的SaveGameToSlot函数。



将SaveGameObject的内容保存到平台专属的存档槽或文件。

Key：此操作将写出所有非临时属性，SaveGame属性标签未勾选。

Key2：如果我们要实现多存档，这里的插槽的名字应该是一个数组。

当需要读取时，使用的函数是LoadGameFromSlot

从给定插槽加载内容。他就是从本地文件中以二进制的方式读取保存的数据



使用方法是将函数的返回值CastTo\_OwnSaveGame后，就可以得到自定义SaveGame里定义的变量，然后我们就可以将这个变量用于赋值。

数据保存位置：/工程目录/Saved/SaveGames/\*

SaveGame系统能保存的数据：字符串、任何类型的数值、数值类型的数组、纯数值的结构体或向量、字符串字典

不能保存的数据：指针或引用

像Spawn、Create出的对象，他们的返回值都是指针或引用

这一套逻辑要注意的是，每一次存档的时候有没有创建一个新的存档对象。如果是的话，每次读档的时候只能读取上一次的存档对象，而不能去读取上上次的存档对象。

所以存档的第一步不应该是CreateSaveGameObject，而应该是检查这个存档是否存在，也就是“**游戏存档存在**”函数

无论是不是存在，我们都可以创建SaveData对象引用类变量（即CreateSaveGameObject的返回值类型）

# 游戏存档基本逻辑

我们需要做一个能够将保存系统扩展到多个项目中的框架，并使用良好的软件工程实践对其维护或添加新的功能。

首先创建第三人称模板，我们将使用这一模板。

我们可以给我们的地图进行一定程度的设计，设计出高度差，在角色达到某个高度的检查点的时候保存游戏

我们也可以让角色拥有一种能力，这个角色可以释放第三人称自带的蓝色物理方块。

以及实现角色的生命值数值，实现伤害和回血。我们在保存游戏的时候会保存这类数据。

同时创建一个文件夹命名为SaveGame，用于后续的使用

## 统计信息

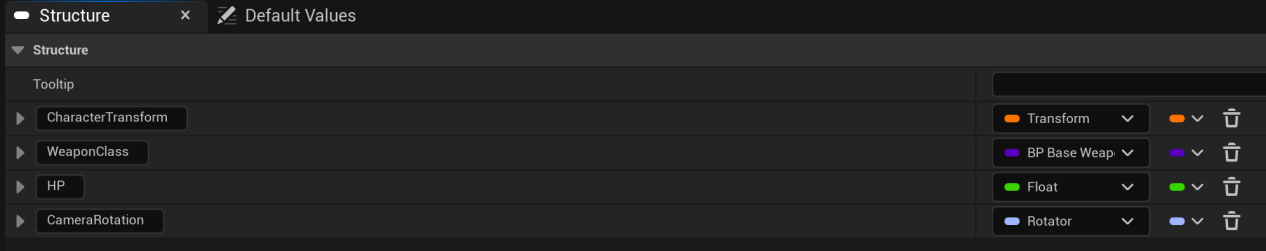
创建SaveGame类蓝图，命名为BP\_SaveGame

在其中创建变量。这些变量就是在游戏保存或者开始时我们需要保存或加载的那些数据。这些数据的类型高度依赖游戏的类型。现在我们要思考的就是在游戏关闭时我们需要保存的数据。

在这个项目里，我们需要保存的有：

角色的位置、持有的武器、角色的方向、HP、我们动态创建的物理箱子。

首先创建结构S\_PlayerSave，创建成员变量



然后在蓝图类中创建结构类变量

这样每次想保存玩家信息的时候就直接弄结构体就行。

接下来要保存的就是我们动态创建的箱子。这些箱子是很多的，我们需要知道关卡中每个箱子的状态。

在蓝图类中创建Transform类数组，命名为CubeTransforms

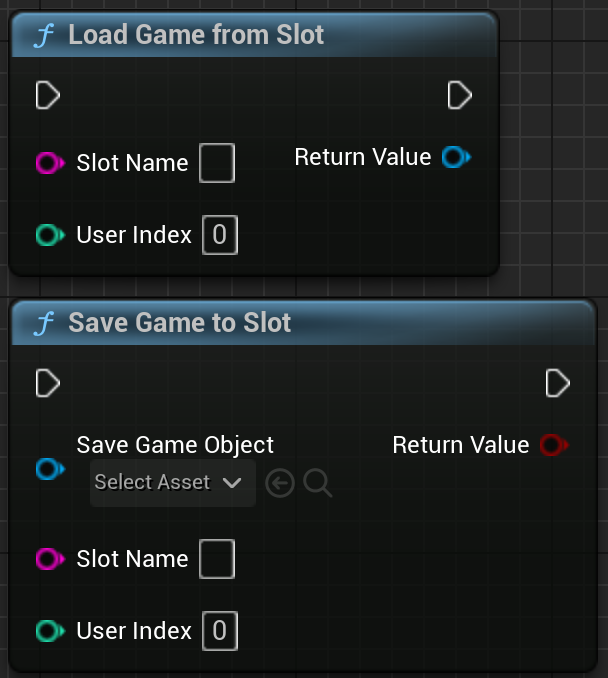
这样变量的定义就完成了。

如何保存和加载这些数据呢？

## 相关函数、同步和异步的保存与加载

通常，SaveGame类中没有事件，但是他们有函数。

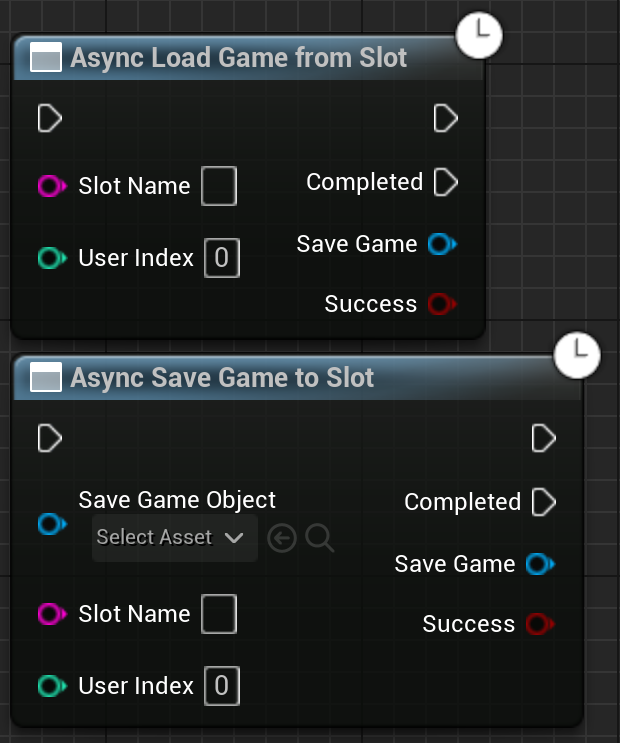
SaveGame类中有许多函数，如LoadGameFromSlot和SaveGameToSlot



SlotName通常用于多存档，UserIndex通常用于多玩家，如PlayerStation5上的多用户存档。

对于异步加载和异步存储（Async），同步加载和异步加载的区别就是，如果你有大量的数据需要处理的时候，当我们从硬盘上加载大量的数据时，同步会很慢，异步就会很快。

对于异步加载，他可以在后台运行这项操作，并且通过Complete引脚进行通知



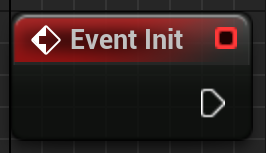
同样的，多人游戏不建议使用。

后面我们会同时支持同步和异步。同时，推荐在加载的时候使用同步的方式，在存储的时候使用异步的方式。

## GameInstance游戏实例

GameInstance是一个用来处理游戏当前状态的对象，它超越于关卡而存在。游戏实例是全局唯一的，且是最高权限的。通常，我们可以在GameInstance中实现游戏的保存和加载。在玩家切换关卡的时候加载同样的信息。

创建GameInstance类蓝图，命名为BP\_GameInstance。然后在项目设置中对游戏实例进行设置。接下来，我们要做的就是在GameInstance初始化的时候加载游戏。打开BP\_GameInstance，他有一个自己的函数Event Init



我们首先要做的就是在GI中创建一个BP\_SaveGame类变量，命名为SavedData。也创建一个String类型变量给LoadGame提供参数，变量名称为SlotName，默认值为Slot1。

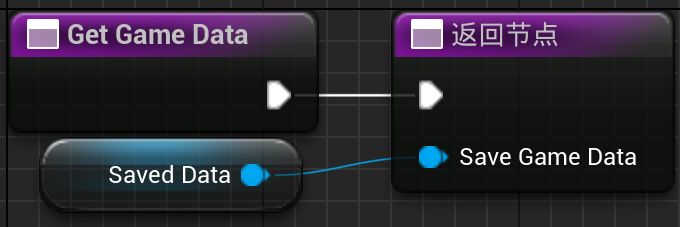
然后实现接口，完成对GameInstance中相关变脸的设置。创建BPI\_SaveGame，创建函数：

void LoadGameData（bool IsAsync）

void SaveGameData（bool IsAsync）

BP\_SaveGame对象引用 GetGameData（）

回到GI，实现函数。



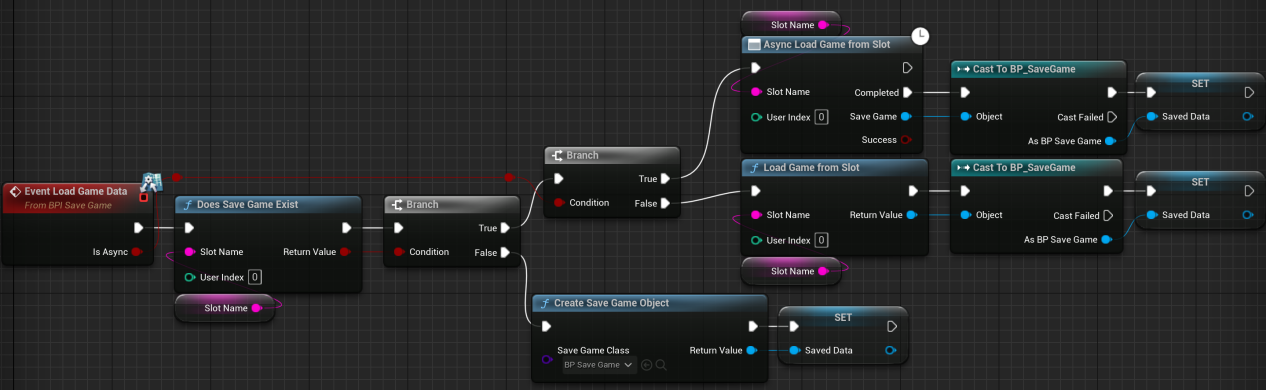


图 LoadGame接口函数的实现

对于LoadGameData中的Async，因为AsyncLoad后面的东西并不会立刻执行，如果程序的其他部分在后台等待游戏的加载，加载完成时我们需要对齐进行通知，通知他们可以使用这些数据，而不是每几秒轮询一次。最好的方法就是事件分发器。创建事件分发器OnGameLoaded。并在异步完成时进行一次通知。

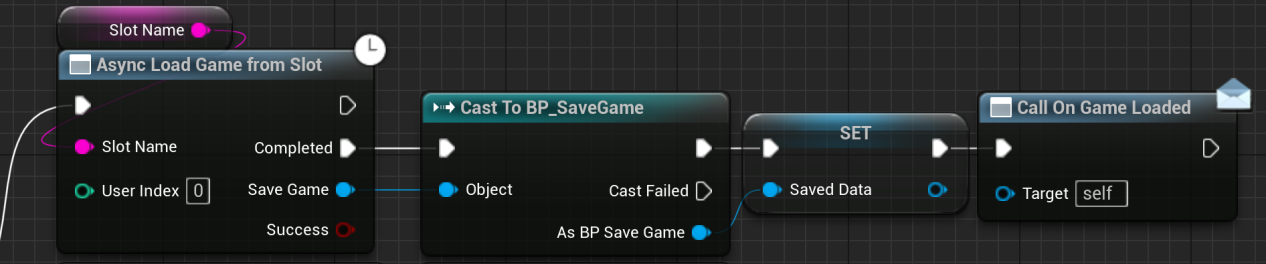


图 LoadGame接口函数的完成

这样其他类就会监听这个事件有没有被调用，一旦被调用就能访问加载的数据。

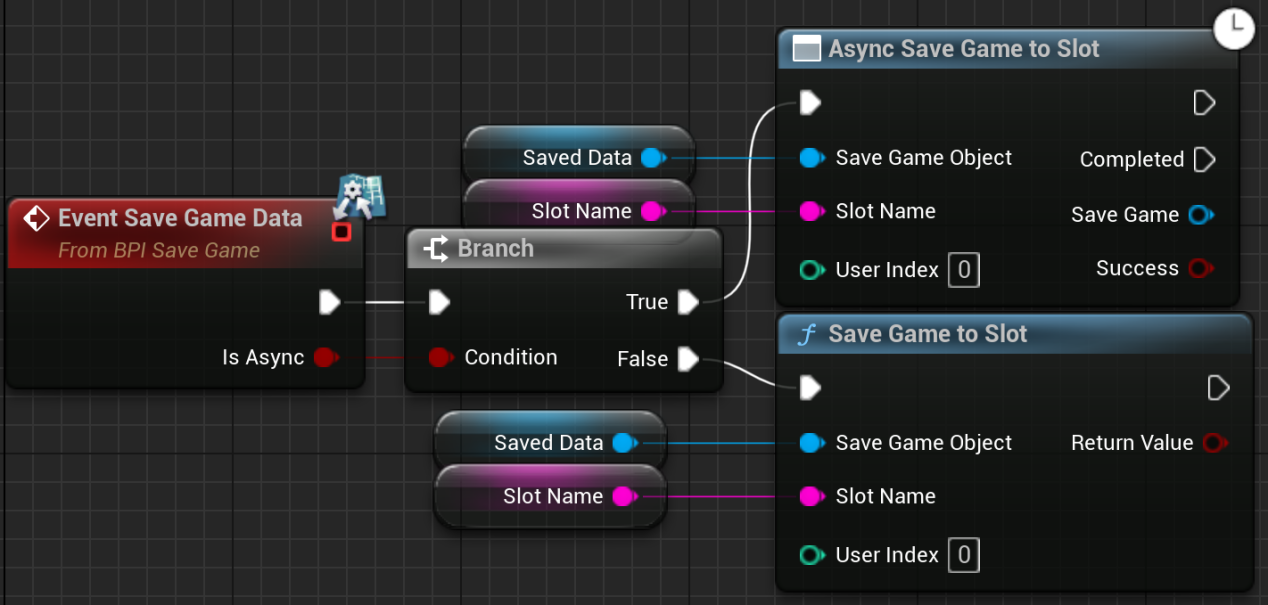


图 SaveGame接口函数的完成

接下来就是GameInstance的Init事件

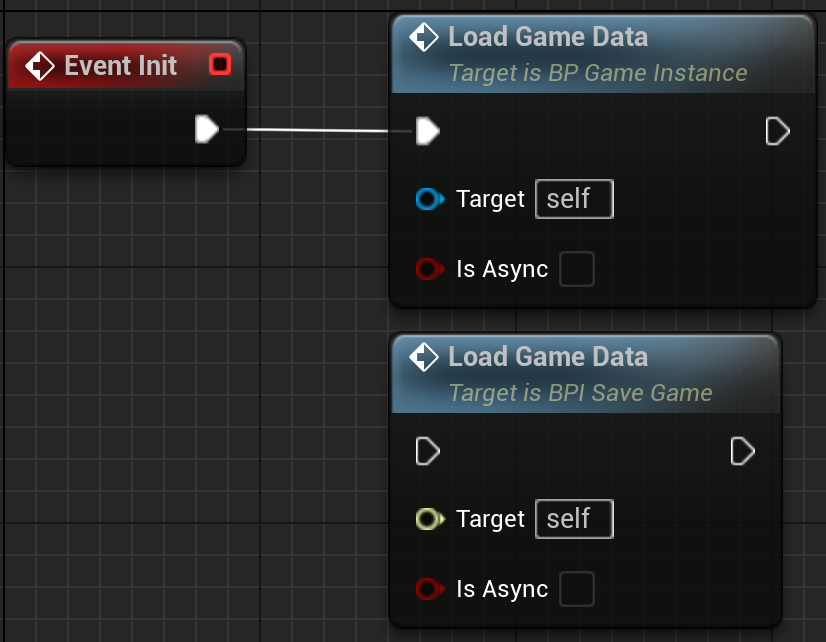


图 注意用上面的函数

确保调用的是同步的版本，这是因为在GI中，游戏还什么都没有加载，我们要做的第一件事就是加载这些基础的游戏数据，我们希望可以同步的加载，来阻断程序的执行。只有在加载完成基础数据后才能加载关卡和角色。因为我们需要知道我们要把角色生成在何处。

当然，异步加载可以实现加载界面。

## 检查点

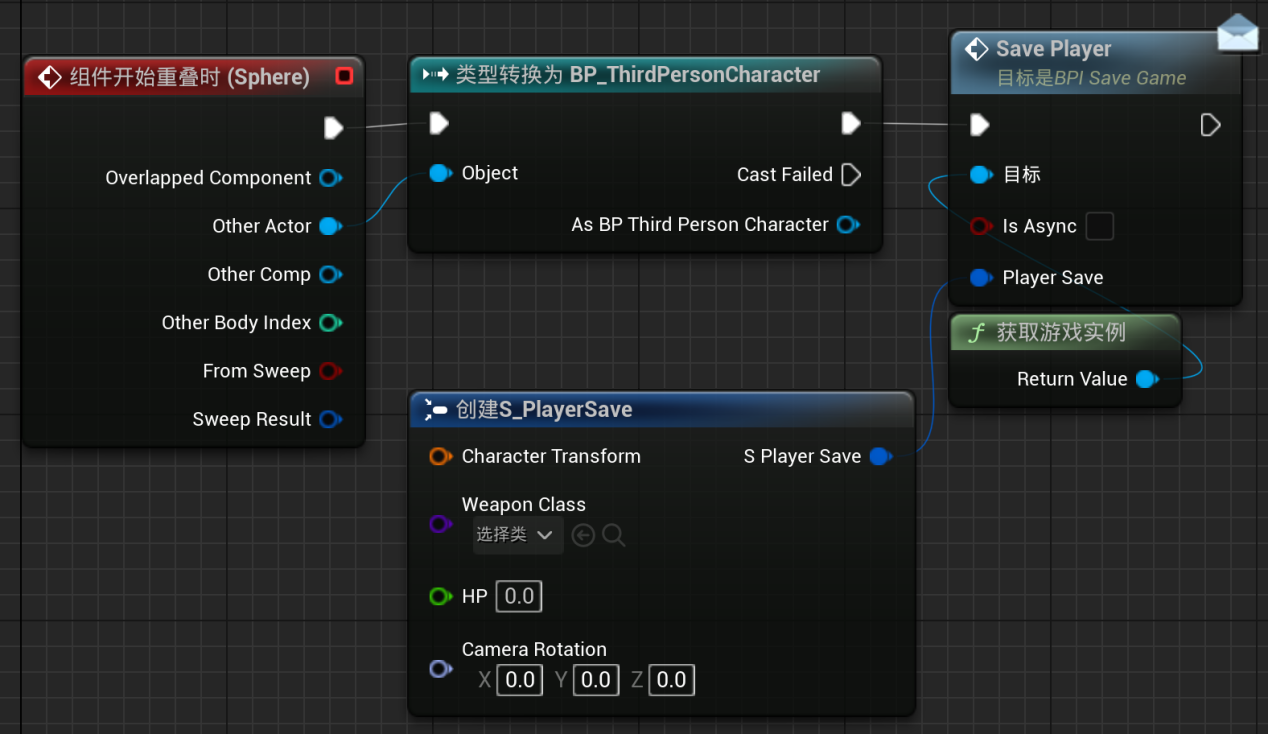
我们希望角色在达到某个高度后可以通过检查点来保存。我们创建一个检查点Actor，命名为BP\_CheckPoint。很简单，就是一个球体碰撞。OnComponentBeginOverlap时，我们需要实现SavePlayerData的功能。

回到BPI，创建函数：void SavePlayer（bool IsAsync , S\_PlayerSave PlayerSave）

回到GI实现函数

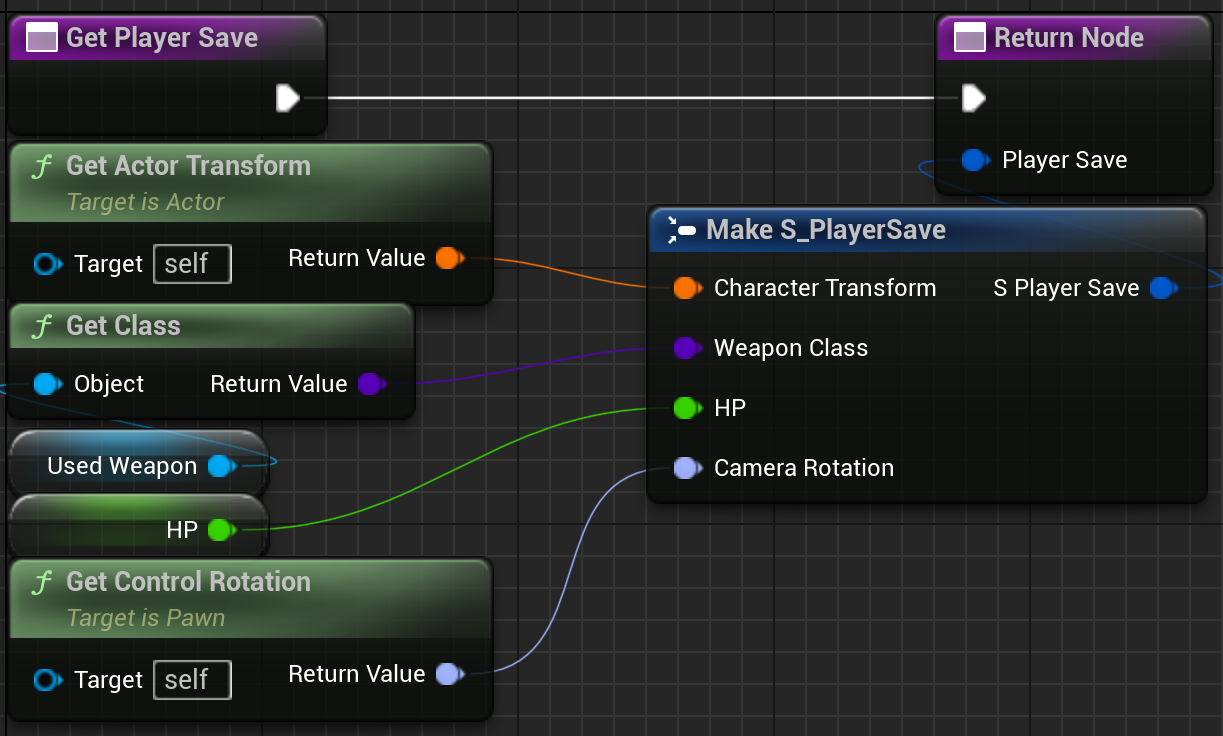


然后回到BP\_CheckPoint

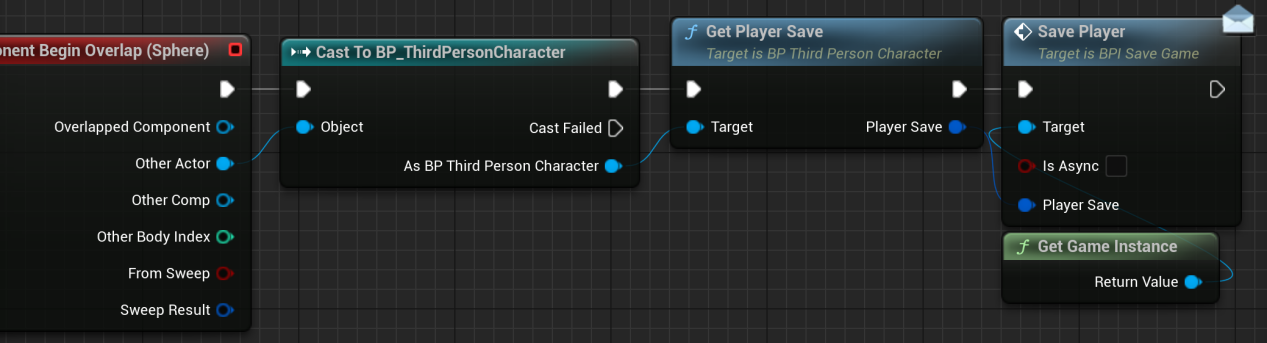


这些参数是BP\_ThirdPerson的相关变量

在BP\_ThirePerson中，实现函数GetPlayerSave



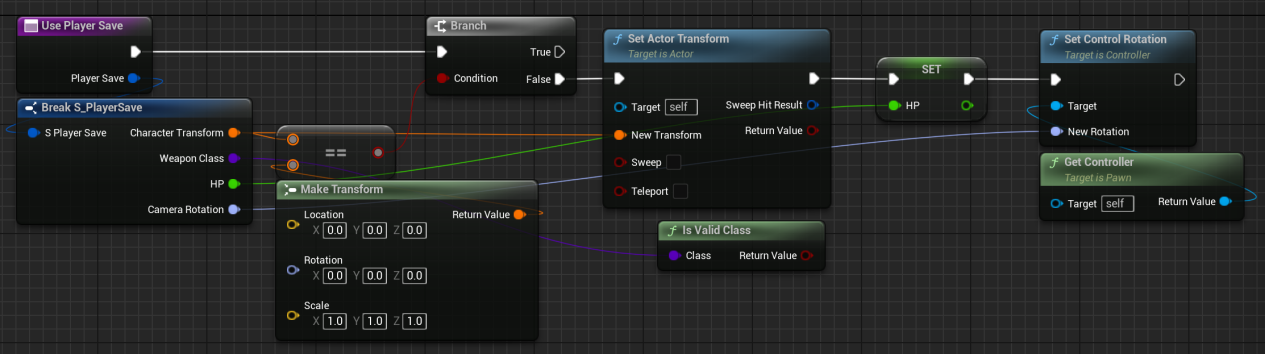
回到BP\_CheckPoint



这样就完成了保存逻辑。此时我们只是把他们存到了文件里，然后加载到GameInstance里。但是我们没有在Player中应用这些逻辑。

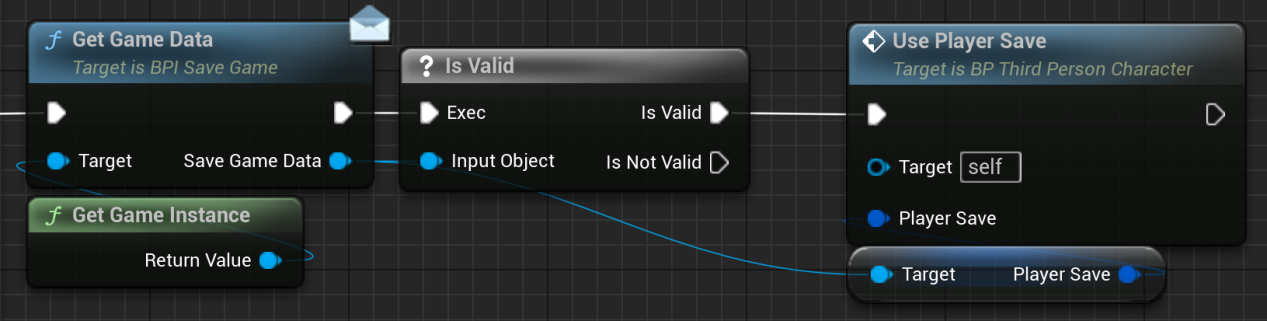
下面实现读取逻辑

在BP\_ThirdPerson中，创建函数UsePlayerSave（S\_PlayerSave PlayerSave）



然后我们就要考虑在什么地方调用这个函数了

我们可以在ThirdPersonChar的BeginPlay时调用



这个时候进入游戏就会发现可以有存档点保存角色位置和生命值了

## 删除存档文件

在编辑器下，存档文件在/Game/Saved/SaveGames下，会有以SlotName为名称的一个sav文件。删掉他就行。

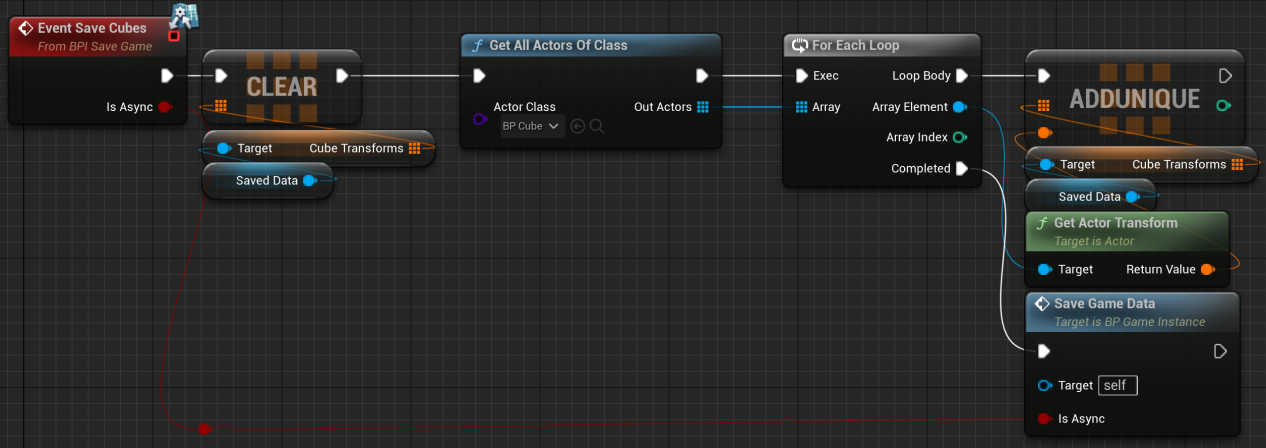
## 保存在关卡中生成的对象

我们通过SaveGame类中的Transform数组保存生成的箱子的位置。我们要做的就是在箱子的生成或销毁时、在箱子的位置更新时，把每个箱子的Transform保存在数组中。

在BPI中实现函数SaveCubes（bool IsAsync）

回到GameInstance中，我们在游戏实例中保存所有功能

首先实现接口函数

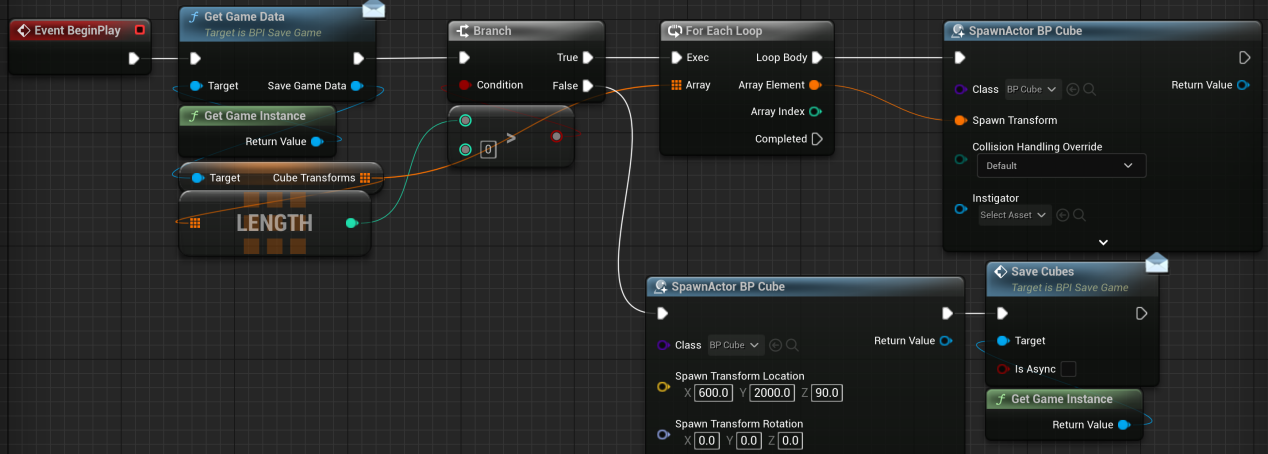


Key：很显然，当你的关卡中有成千上万的Actor时，我们并不需要对他们全部进行更新。有一个更新时，对他进行标记，我们只更新发生变化的那一部分即可。所以这个方法并不是最好的。但是，经过测试，2000个箱子也并不能在一定程度上影响性能，所以就还是用图上提供的这个方法。

下面，当箱子发生变化时，更新他们的Transform信息

首先一点：在ThirdPersonChar中，我们通过一个按键来SpawnActor，通过另一个按键对SphereTraceByChannel范围内的箱子进行Destroy。在关卡蓝图中，我们也通过SpawnActor生成了几个箱子。

在关卡蓝图中，实现保存功能。



当我们通过角色动态SpawnActor时，或删除Actor时，我们也需要更新数据。

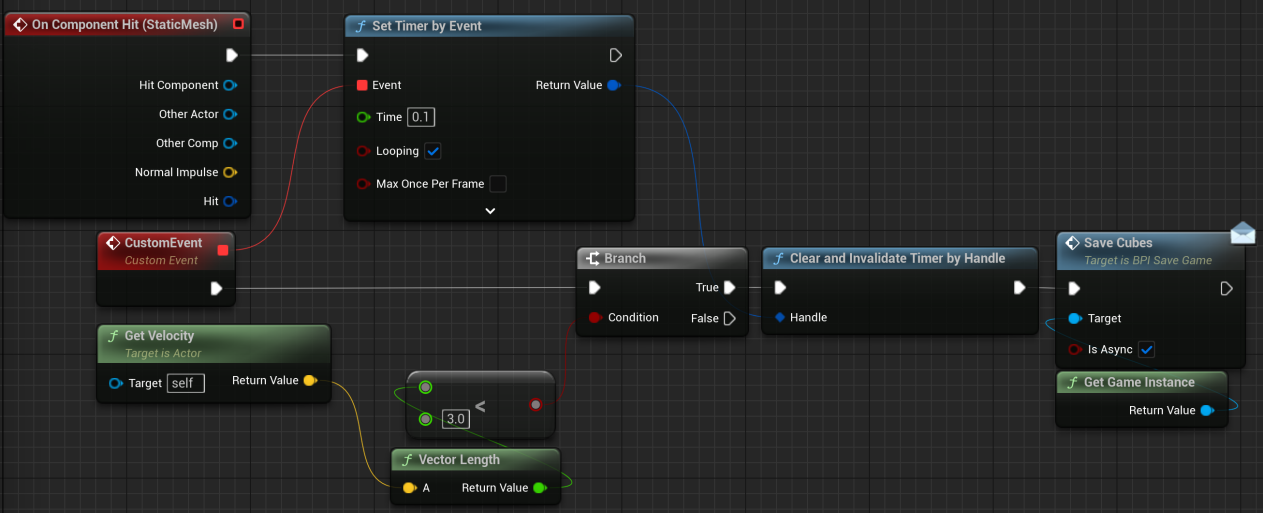
就是在角色类中，在SpawnActor或DestroyActor后面调用GameInstance的SaveCubes函数。

这里的保存可以使用异步保存，因为我们得在保存的时候继续游戏。同步的话就不能继续游戏（除非保存的东西很少，体感延时很短）。

最后我们更新箱子的位置。我们需要在箱子的位置更新的时候进行保存。

由于箱子是一个物理物品，它受外力影响其Transform，所以我们在BP\_Cube的OnComponentHit事件触发的时候调用SaveCubes函数。

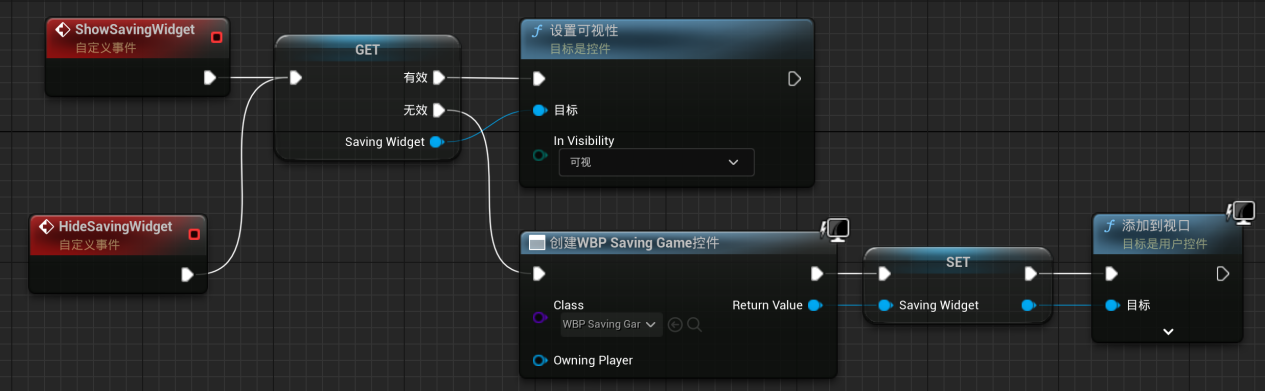
当事件触发时，我们要检测箱子的速度，一旦箱子停止，立刻保存位置。



## 正在保存的UI指示器

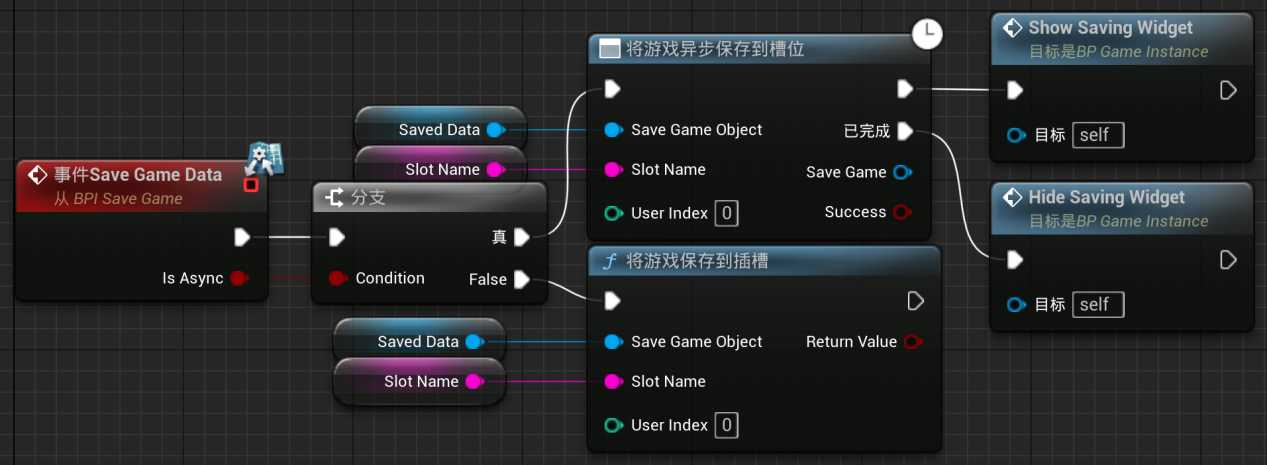
关键就是在异步保存的时候CreateWidget并AddToViewport，在结束后RemoveFromParent。

在GameInstance中实现相关功能。创建两个函数。



这个Widget我们只会在异步的时候使用。

在BPI的SaveGameData函数的实现中，在异步这里调用两个函数



# 多存档

多存档的一个核心就是，我们需要把存档保存到一个位置，逻辑是在SaveGame类中有多个相同的需要存储的数据。

所以我们将所有需要存储的数据集合成为一个结构，然后创建结构类的数组变量。

这样的好处是，不用创建多个存档Slot，存档文件也并不会很多。

## 存档逻辑

如果想创建新存档，就是对结构数组的简单的Add

如果想修改当前存档，就是尽可能地找到结构的Index

读档时，就是对Index的赋值

## 多存档的直观显示

首先，我们的多存档应该会有一个WBP来显示我们的存档

我们可以创建新的WBP，或者就使用WBP中的控件切换器。

无论哪种方法，我们都是要让它能显示读档的多档，且是通过按钮来显示。

对于多个档，我们可以创建滚动框（包裹覆层），在滚动框中加上自定义的WBP\_SingleSaveDocument

在WBP\_SingleSaveDocument中，层级为“尺寸框（宽度重载800）--水平框--文本（好几个）、按钮等”（当然可以把整个尺寸框包裹在按钮里面）

文本中，显示存档日期、游戏时长、等级等相关参数

首先我们需要在WBP的EventConstruct时获取游戏存档，通过CreateWidget（WBP\_SingleSaveDocument）并通过结构体初始化后，将这个Widget添加进滚动框。

此时，删除存档也变得非常简单，将数组中的某个索引删除即可。

# 增加新功能时不破坏旧存档

核心内容就是尽可能不要修改SaveGame类，即使修改也只做增量，不做减量和修改。变量的顺序、名称最好都不要修改。

下面给出一种设计思想。

我们可以创建多种存档：战斗存档、战绩存档、游戏存档（全局性质）。

在游戏存档中，把游戏相关的所有全局的内容都放到这里，如系统设置、音乐音量float、音效音量float、累计击杀、成就等。



还有战斗存档和战绩存档。

战斗存档：一般是用于继续游戏用。当我们需要在游戏中间进入游戏时，保存的对应数据。当一场战斗打完后，可以将这个存档删掉，将其保存到战绩存档里面。

战绩存档：首先，我们如果打完一局，将这一局的信息保存为一个文件，那么当局数多的时候，文件数量会非常多，不利于下载上传。所以战绩存档最好就一个文件。

其实关键的是，游戏存档这个东西，一旦确定好了之后，尽量只做增量更新，尽量不修改旧的。（int32改成int64是没问题的，但还是尽量别改吧）无论是SaveGame，还是结构体，都不建议修改，可能会出现打包失败的现象。

# C++存档与保存游戏

参考文档：

<https://www.bilibili.com/video/BV1Skx4eJE29/?spm_id_from=333.1387.favlist.content.click&vd_source=79fbe818ff96aae3677e123f0374bd91>

# 建议

在玩家第一次玩的时候，建议给一个初始化存档，用于初始化设置等。

# 参考文档

<https://www.bilibili.com/video/BV1jzBeYfEBg/>

<https://www.youtube.com/watch?v=H6rqJbwjRIk>