GAIA TRIGGER

1. **概述**

我们游戏的触发器是一个广义的触发器，狭义的触发器是就是仅仅指场景中的一个碰撞盒，而我定义的触发器是包含碰撞盒和所有的这个触发器触发后所发生的所有的事情。

根据这个思路我把触发器分为3个大方面：

* **场景触发器，**
* **物品触发器，**
* **任务触发器。**
* 场景触发器包含了两个部分：
* 一个是用来进行触发检测的碰撞盒，
* 还有一个就是触发后能够做的各个Gameplay的事情（以任务来分），
* 物品触发器： 玩家捡到什么物品触发什么事情，比如说捡到一把武器，然后触发了一个剧情或者任务或者通知其他Trigger。
* 任务触发器： 指接受到什么任务后会触发什么事件，比如玩家获得一个道具，或者其他的场景触发器触发。

好的游戏结构是通过数据来驱动，因此我在设计Trigger的时候将Trigger和Trigger所触发的事件分开来，这样做的好处：

* **触发的事件可以重复利用**
* **触发的事件具有独立性，维护方便**
* **触发的事件可控。**

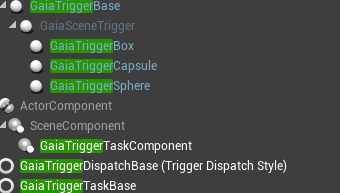
触发的事件在后期可以当成一个资源，这样可以在相同地图能做出不同的游戏内容。

就像所有的事情当有系统规划的时候，就不是一件容易的事情，它就会变得复杂，因此对理解和使用也许会有一些障碍，但是我认为这些都是值得的，非常非常值得的事情。

想要做的稳少出问题，只要事先规划好，设计好，有了好的框架后期扩展也非常容易

1. **功能介绍**
2. **基础框架介绍**

打开编辑器在触发蓝图类的时候会发现4个系列：

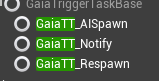


其中GaiaTriggerDispatchBase还在开发中（可以直接忽视）：它的作用是用来分发任务的

* GaiaTriggerBase:作为一个Trigger的容器，（在我们游戏里就是一个Actor）它主要的是负责对外的统一规则，比如说这个Trigger有哪些基本行为规范：比如说是不是通过碰撞后开始执行自己带的任务，还是说碰撞后执行自己的任务，还是说有其他的射线检测碰到它了它来执行。或者说一些绑定的特效什么的。

在GaiaTriggerbase下面可以看到子类GaiaSceneTrigger,并且GaiaSceneTrigger的子类GaiaTriggerBox, GaiaTriggerCapsule, GaiaTriggerSphere. 这些都是放在所有的场景中的Trigger的，是带有碰撞盒的。

* GaiaTriggerTaskComponent: 作为一个Trigger容器的部分，它也是管理这个Trigger的任务发放方式，通知子节点的方式。它不决定什么时候执行什么任务，只有当GaiaTriggeBase通知它去执行所有Task的时候它才会决定用什么顺序来执行这些任务【以后需要更新】（当前只是按顺序在同一事件内都执行了的）。它也不会主动去通知ChildNode，都是GaiaTriggerTask或者GaiaTriggerBase来通知它来按照一定的顺序去通知ChildNode.因此可以把GaiaTriggerComponent作为一个中间层，它不负责表现也不负责游戏逻辑。
* GaiaTriggerTask:这个是真正负责游戏逻辑的对象，它都是所有的游戏逻辑都分成一个又一个任务，比如说玩家打完怪开门了，那么这个里面其实是有两件事情，一个是刷怪，一个开门，那么这两个就可以拆分成两个task，如果再有触发掉落事件，那么又可以再加一个task来实现新的游戏逻辑。



在蓝图生成界面里可以看到这些Task的类。其中命名中有TT表示：Trigger Task,

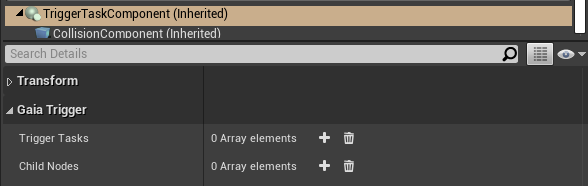
因为太长就缩减了一下。

一个Trigger可以包含多个TriggerComponent,。一个TriggerComponent又包含了若干的TriggerTask。

1. **属性配置**
2. **通用属性配置**

GaiaTriggerBase这一系列目前来说还么有属性需要配置.

在GaiaTriggerComponent 里面【目前】是有两个属性是可以修改的，



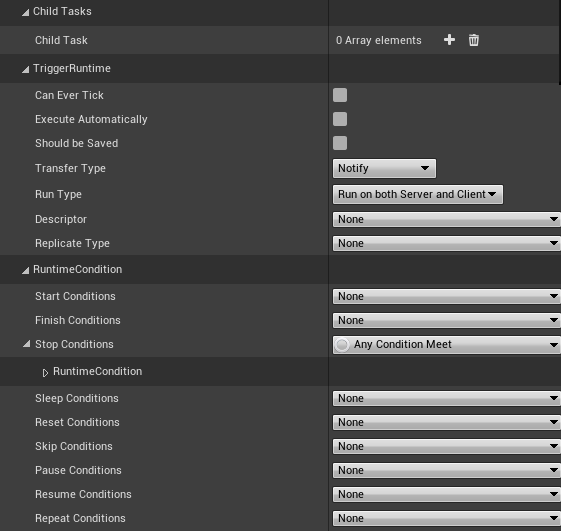
* TriggerTasks：可以让策划在这个Trigger上挂上任何需要触发的事件。
* ChildNode： 是这个触发器所关联到的Trigger。

需要注意的是这两个变量都是实列话后（也就是只能放在场景里）才可以赋值，这么做的原因是因为这两个对象都是跟当前地图的Game模块相关的。

在GaiaTriggerComponent 里的事件的自定义目前没有，只有一些暴露出可调用的函数。

TriggerTask 这个因为是以游戏为导向的，因此属性配置会变化很多，我就说一下TriggerTaskBase里面的属性，这里面的属性都是通用基础的属性：通用基础属性分为3个部分：

* ChildTasks:这个是用来配置这个task执行玩之后要执行的事情
* TriggerRuntime:这个里面是用来配置一些在运行时的属性
* RuntimeCondition:这个是用来配置在Task运行的时候自动切换Task的状态



在这个里面ChildTasks和TriggerRuntime是可以在Defalut BP Class 里面配置。

而RuntimeCondition只能在每一个Instance里面配置。

在TriggerRuntime里面有下面几个属性可以配置

* InstanceType:每一个Task都有两个基础属性，一个是Instance, 还有一个就是Template，

当为Template的时候，那么每次这个task被执行的时候都会生成一个新的实例对象来执行，当前配置的task只是作为一个模板而已。

* Use this task directly:每次这个task被执行的时候都是用自己
* Create New Instance form this task: 每次这个task被执行的时候都出穿件一个新的task来执行的
* Transfer Type: 这个枚举值用来说明当这个Task用来通知其他的task要做什么事情的时候，应该怎么做，这里面目前来说有3个值：
* Notify： 仅仅是通知其他的Task，不做额外的操作
* Move： 将会将这个Task移到另外一个Trigger。
* Copy： 将会在其他的对象上同时有一个相同的Task在运行
* Can Ever Tick: 勾选后如果这个Task能够触发那么就会一直执行Tick函数。
* Only Run On Server: 勾选后这个Task的逻辑只会在DS上执行，客户端最多只会同步数据。
* RunType: 这个是用来标记这个task是在什么机器上使用的。只有当前机器的权限和这个枚举值匹配的时候才可以被执行。而且还需要跟TriggerTaskComponent的RunType 匹配，否者都是不能被触发的
* Run on both server and client: 这个task既可以在服务器上执行，也可以在客户端执行。
* Only Run on server: 这个task 只能在服务器上执行。
* Only Run on client: 只能在客户端执行。
* Can Execute Automatically: 勾选后这个Task 会在生成后自动执行。比如如果是绑定到地图上的Trigger后，在地图加载后就会自动执行这些任务。
* Should Be Save: 这个task是否可以被trigger save system 监测保存数值的。（还属于测试阶段）

ReplicateType: 这个是用来确定这个task是应该以什么样的方式去同步属性。详情请参考Trigger属性同步文档

在RuntimeCondition里面有一些需要配置的TriggerTask自动切换状态的属性。

在Task里面如果查看TaskState的枚举值会看到有很多状态，但是在一个Task开始到Finish这种状态流转是固定的：Start->Prepare->Active->Finish->PostFinish->End.

但是可以在这个状态切换到其他的状态，比如说直接Finish的状态，就是跳过Prepare，和Active， 或者直接让这个task Stop。等等。

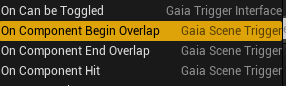
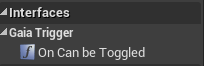
* StartConditions:这个是用来配置当这个Task开始执行的时候，需要哪些前置条件。
* FinishCondiitons:这个是用来配置在什么样的情况下，这个Task直接Finish。
* StopConditions,:这个是用来配置在什么样的情况下，这个task直接停掉。
* SleepConditions:这个是用来配置在什么样的情况下，这个task直接进入休眠状态。

在这里需要说明的是当一个Task被停用后，是永远不会再被启动了的。而当一个Task进入休眠状态的时候，他还是可以被启用的。

* ResetCondition：这个是用来配置在什么样的情况下，这个Task需要重置
* SkipCondition:这个是用来配置在什么样的情况，这个Task需要直接跳过
* PauseCondition：这个是用来配置在什么样情况下，这个Task需要暂停
* ResumeCondition:这个是用来配置在什么样情况下，这个Task需要回复

这里需要说明是Pause和Resume这两个是绝对对立的，也就是说PauseCondition起作用的时候，ResumeCondition是不起作用的，反之亦然。

* RepeatCondition:这个是用来配置在什么样的条件下是可以重复执行，这里需要注意的是，重复执行的方式必须要有一个Style来确定，这个Task是不会定义自己是要怎么重复执行的。

1. **通用函数配置**
2. **GaiaTriggerBase** 这个里面有4个函数可以自定义：
3. 

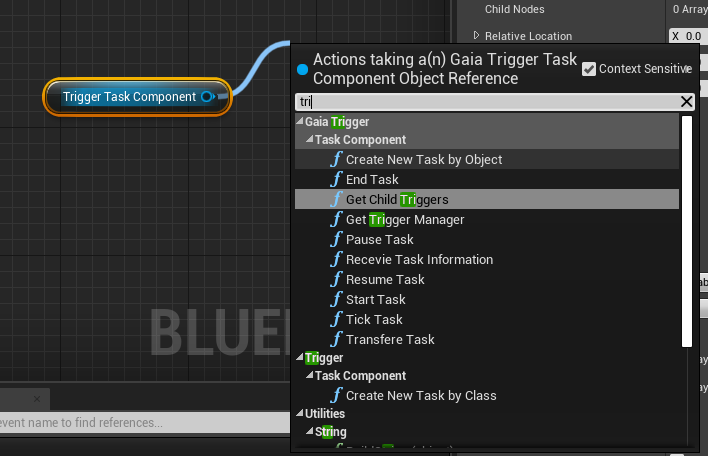
前面那张图是Interface，后面的那张图是点击函数Override按钮出现，这个里面的出现相同的函数：OnCanBeToggled是一个函数，只是显示在不同位置而已。

* On Can be Toggled :（请不要和下面的Trigger Task Base 里面相同的函数混淆），这个函数是用来自定义来确定这个Trigger是否能够被触发;比如说这个Trigger被两个AI 触碰了，那么我们只想A可以触发这个Trigger, B不能触发，那么这个函数就可以做到自定义的。

如果是GaiaSceneTrigger 那么可有额外的几个事件可以自定义：

* OnComponentHit是当Trace的时候碰到用到的。
* OnComponentBeginOverlap 是当有其他的碰撞体开始碰到这个Trigger的时候调用。
* OnComponentEndOverlap 是当有其他的碰撞体离开这个Trigger的时候调用。

1. GaiaTriggerTaskComponent 这个里面只有可调用的函数（非可重载自定义的函数）（这个后期会有改动，函数名字，函数分类可能都要改动的）

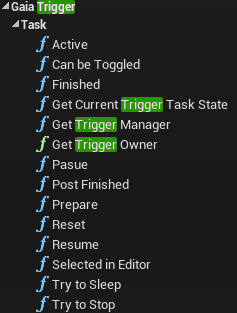
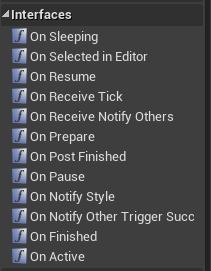


* Create New Task by Object: 创建一个Task 通过已有的Task Object 来生成一个TriggerTask
* Create New Task by Class: 创建一个Task 通过已有的Task class来生成一个TriggerTask
* Get Child Triggers: 获得所有的子节点，（子节点就是指上面提到的ChildNode里面的值）
* GetTriggerManager: 获得所有的Trigger的控制器。
* Pause Task: 暂停正在执行的task.
* Receive Task Information: 当有其他的Component 发送消息的时候这个函数会处理这些消息
* Resume Task: 当我们暂停任务的时候，我们可以通过这个来恢复任务的
* Start Task: 这个是用来开始执行这个component里面包含的所有的任务。
* Tick Task: 这个是会执行所有的Tick的功能。
* Transfere Task: 这个函数用来主动传递消息给其他的模块。而这个函数传递消息给其他trigger的时候对方Trigger可以通过Receive Task Information来接收消息，

这个函数一般和Receive Task Information和下面GaiaTriggerTask里是统一使用的。

当Task 或者Trigger通过Component来调用TransferTask 来发送消息的时候，那么在目标Trigger 的TaskComponent 通过ReceiveTaskInformation 来接收消息，它再把消息发给它所管理的Task，然后这些Task再通过ReceiveInfosFormOthers 来处理这些消息。当Task处理通过了的就可以通过了那么原来的发消息的Task可以通过ReceiveTaskSuccessed 来作为一个回调函数。

1. TriggerTaskBase: 这个类里面既有可调用的BP函数，也有可以自定义的事件。



左边是可以自定义的事件，右边是可以调用的函数。

Task 主要有7个处理状态：Prepare, Active, Tick, Finish, PostFinished, Pause, Sleeping.

这里面Prepare一般用来作为这个Task初始化的地方，这个时候这个Task一般来说是还没有走GamePlay的逻辑。

Active也就是说 这个Task开始执行GamePlay的逻辑，也有可能是是再碰撞后触发这个任务。

Tick 这个是每一帧都会处理的函数，但是它的激活需要通过勾选上面的CanEverTick这个变量。

Finish一般是用来通知说这个任务结束了

PostFinished 一般是用来说这个任务结束后，我们Task要做的一些后处理的事情，比如说删除零时状态，通知其他Trigger等等（具体做的事情会根据不同的task有不同的表现）

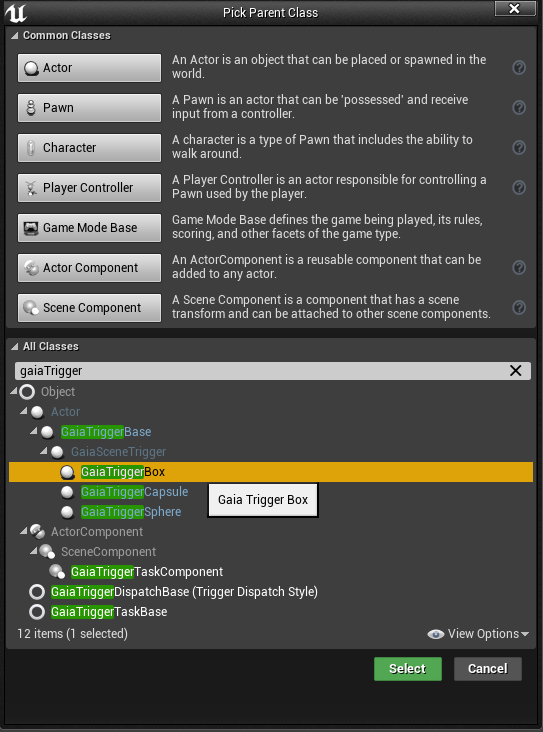
这里有两个特殊的状态：Pause 和 Sleeping， 这两状态都表示这个task不会修改当前GamePlay 的状态，他们的区别是Pause 不修改当前GamePlay状态，但是这个Task还是处理激活状态，当他激活的时候只需要通过调用Resume 来从当前状态继续走，而Sleeping状态是说这个Task处于沉睡状态，如果要激活这个Task那么就需要再重新走一边流程（Prepare->Active->Tick->Finished->PostFinished），所有的激活的状态都是冲头开始的新状态。其实就类似于重新生成一个Task，但是Sleeping 激活不同的就是不需要重新生成一个Object（开销会小一些）。

再上面看到左边的自定义的函数再又变有名字长不多相同的函数，只是没有[On]（比如OnActive 🡺 Active）,这说明这些自定一的事件的触发都是再右边相对应的函数里触发的。

1. **简单的使用范例：**

就比如说我们要通过一个Trigger在刷新AI后，所有AI死亡后能够通知一个门的Trigger去开门。

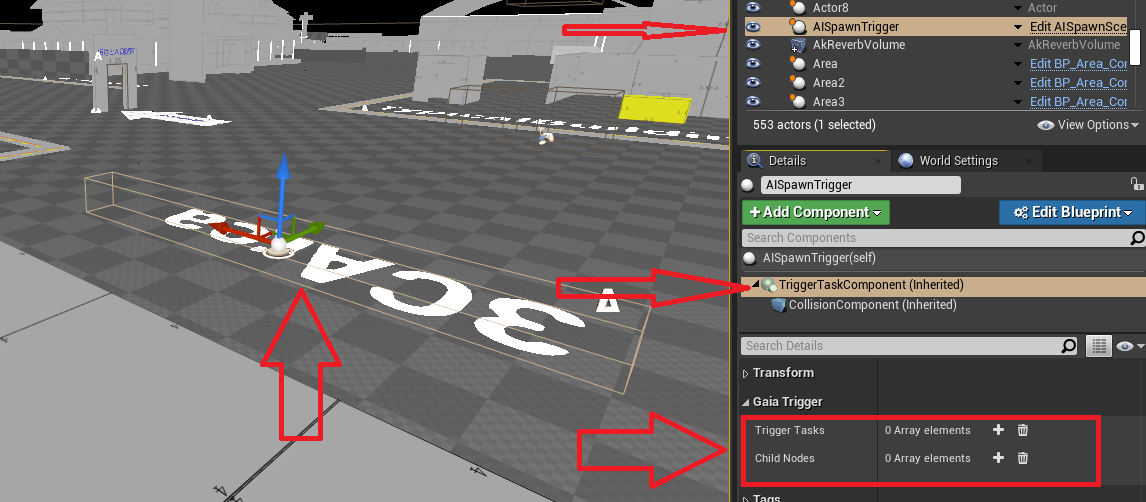
1. 打开BPClass 创建界面



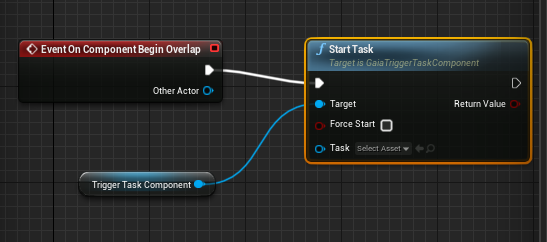
1. 选中GaiaiTriggerBox，然后选中资源管理中的Asset，重命名为AISpawnTrigger,。



1. 将这个TT\_AISpawn拖到场景里去，选中这个Trigger,然后选中Details里面的TriggerTaskComponent 组件会看到TriggerTasks 和ChildNodes的选项的。

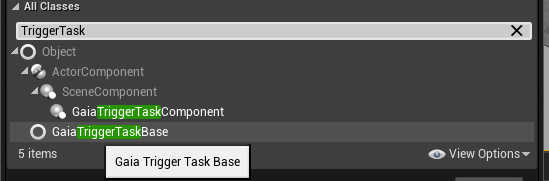


1. 在TriggerTask里面点击+号选中GaiaTT\_AISpawn 这个任务，然后填充这些值
2. 打开Trigger的蓝图，在蓝图里添加Begin overlap事件。

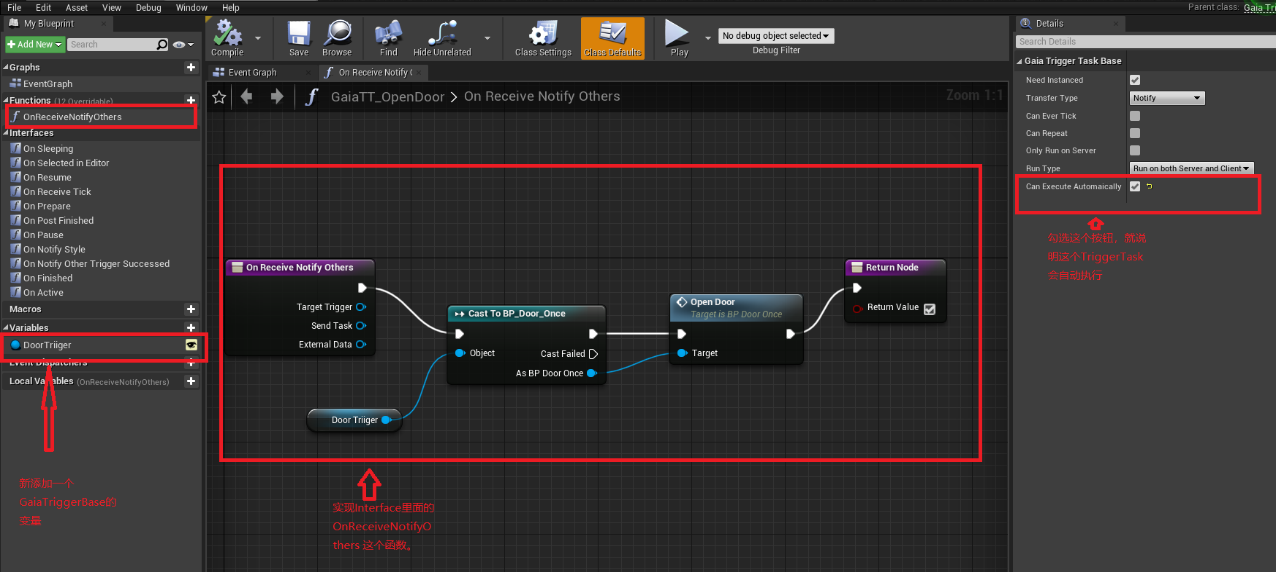


这个就说明当这个Trigger被碰撞后就会执行它所包含的所有的Task.

1. 通过BP class 来生成一个继承GaiaTriggerTaskBase的函数



1. 打开资源然管理，然后重命名为GaiaTT\_OpenDoor
2. 打开这个Task,编辑



1. 然后把这个TriggerTask赋值到一个BP\_OnceDoor这个Trigger里面的TriggerTasks里面，就可以了的
2. **程序模块**
3. **总结**