## 研究墨的基于tile的场景管理

### 分割地形

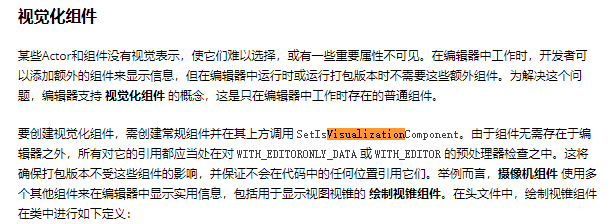
地形是由很多ULandscapteComponent组成的。

获取WorldOutliner中选中的Actor

USelection\* selection = GEditor->GetSelectedActors();

UActorComponent::SetIsVisualizationComponent

可以设置视觉化组件



#### FOpenWorldToolEditor::OnSplitTileClicked()是其插件的正式开始部分

UPackage\* mainPackage = Cast<UPackage>(world->GetOuter());

if (!mainPackage)

return;

FString const &mainPackageName = mainPackage->GetName();

FString const &mainPackageFath = FPackageName::GetLongPackagePath(mainPackageName);可以取到World的UPackage！

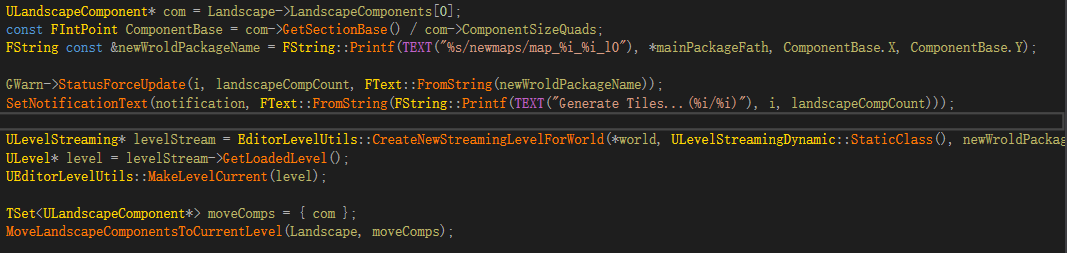
其中有调用一个GatherCurrentLevelActorsAndAABB去查找场景中的Actor并且把Actor包装一下返回回来这样。

**然后开始从LandScape中取每一块了，流程为：**

**直接拿Landscape->LandscapeComponents就可以了。。**

**创建新Level并且把那些LandScapeComponents加入新的Level中**

ULevelStreaming\* levelStream = EditorLevelUtils::CreateNewStreamingLevelForWorld(\*world, ULevelStreamingDynamic::StaticClass(), newWroldPackageName, false, nullptr);



**其中用到的几个方法**：

GatherCurrentLevelActorsAndAABB(TArray<TPair<TArray<AActor\*>,FBox>> &InOutGroupedActors)

中获取场景中的Actor并且加入TArray结构中。

TArray<AActor\*>allActors= GEditor->GetEditorWorldContext().World()->GetCurrentLevel()->Actors;

对于包组，有使用到了一个AGroupActor的东西，

AGroupActor\* inGroup = AGroupActor::GetRootForActor(actor);

找到一个Actor最上层的根，用来作为存储的key，在该函数中还取了FBox

MoveLandscapeComponentsToCurrentLevel

UObject-> GetOutermost()找到最外层的那个Outer 返回一个包

获取地形上的物品

显示策略

墨的这套插件还有个很大的弊端，就是分割是以地形的Components为基础分割的，那么每一个小地图的大小就是固定死了的。

## 使用自己的Grid管理方案进行场景的保存/加载以及地图切换

初步思考的方案：

以一个Grid为单位进行保存以及加载，将一个Grid和一个地形以及上面的多个Actor处理成一个子Level，进行统一的加载和存取（streamlevel） ， 在每一个Level的边缘，有进行判断的方格（包裹整个Grid），来判断下一个要加载的地图是那个并进行流加载即可（流加载完毕后再LoadActor等），这样就可以根据距离，视角等来一部分一部分地图的加载和释放，也可以做成一个超级大地图了。可以任意组合，甚至可以支持不同大小！

配合上之前的序列化保存场景，有搞头。

问题以及想法记录：

15X15Quads地形的每一格大小，SectionsPerComponents 是1X1 Sections，则每一格的大小为1500 X 1500，是可以完美的以格子地形覆盖的！

1、如何判断一个Actor是在该SubLevel上？

为什么要这么弄呢，我为什么，不在创建的时候，就直接把Actor放在对应的SubLevel上呢？

如果就放在Sublevel上的话，那么问题就变成了直接保存SubLevel了

那么就涉及到一个SubLevel对应的问题了

有个问题就是，StreamLevel就算加载了，上面的Actor的Outer好像还是大的那个Level！

尝试一下如果直接使用那个小地图播放，Outer对不对呢

Level->GetFullGroupName(true) 可以得到StreamLevel的Name！！ 如果是在根Level中获取，则获取的名字是LevelMap的名字！！

Spawn出来的Actor，都是已当前的主Level作为Outer所以其最外层的Outer是Level所在的Map，而一开始就在streamLevel中创建出来的那些，则是最外层Outer为StreamLevel。

**所以，保存时的StreamLevel与动态生成的Actor的保存关系就在于，地面Grid这种直接由设计者从编辑器中拖入的Obj来进行绑定的。**

OpenLevel最大的问题在于，open之后的World变化了，必须要使用Level中的参数和值来重新启动。

那么加载的步骤就应该是：

由存档界面点击开始加载Level地图->加载完毕后由Level的蓝图Start调用SaveGameSystem的AfterLoad->读取当前应该显示的StreamLevel ->加载序列化数据

FLatentActionInfo加载StreamLevel时，用来进行回调的处理的结构

FLatentActionInfo info;

info.CallbackTarget=this;

info.ExecutionFunction="test";

info.UUID=12345;

#### StreamLevels

无论StreamLevel是否加载，通过World的GetStreamingLevels方法都可以获取到添加到EditorLevel中的那些level ，

UWorld\* CurWorld = GEngine->GetWorldFromContextObject(WorldContextObject, EGetWorldErrorMode::LogAndReturnNull);

TArray<ULevelStreaming\*> LevelStreamings = CurWorld->GetStreamingLevels();

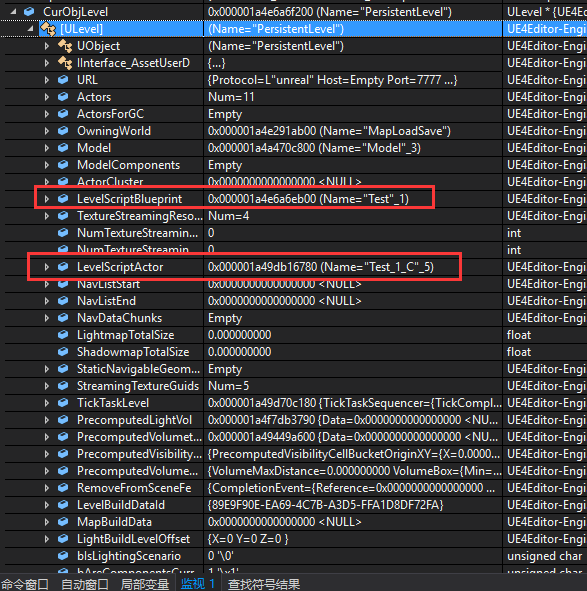
只要通过ULevelStreaming::ECurrentState::LoadedVisible Type判断了是否已经加载了以后，

ULevelStreaming::ECurrentState CurStreamLevelState = LevelStreamings[LevelCount]->GetCurrentState();

if (CurStreamLevelState == ULevelStreaming::ECurrentState::LoadedVisible)

可以通过CurObjLevel = LevelStreamings[LevelCount]->GetLoadedLevel();

方法获取到当前已加载的Level，虽然该Level显示的名字是最上层的level，但是由其内部可以看到



该Level确实是加载的StreamLevel。那么用该Level做Outer生成的Actor，在通过Outer递归最终获取到的Uleve，虽然也是显示的最上层Level的名字，但是其实就是StreamLevel，并可以通过GetFullGroupName(true);

获取其真正的StreamLevel的名字。

#### 自定义一个LevelStreamVolume 体积Actor

使用继承自class ENGINE\_API AVolume : public ABrush 的AVolume , ABrush继承自AActor

首先明确一下需求：

需要一个可在Editor界面中显示放置的Volume，该Volume会响应Overlap在运行运行时，并且触发对应的函数，并且可以判断Overlap的是对应的哪一面；

需要可以对四个面分别设置加载的StreamLevel，理想状态：可以直接下拉选择StreamLevel不需要手动输入名字；四个边上绘制4个箭头一样的标识，直接点击箭头就可以设置四个边碰撞时触发的不同参数，箭头只在Editor显示；

ABrush上用来显示线框的部分，应该是

UPROPERTY(Category = Collision, VisibleAnywhere, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))

class UBrushComponent\* BrushComponent;

UBrushComponent继承自 PrivateComponent ，PrivateComponent 自身是带有

BeginComponentOverlap方法和EndComponentOverlap方法的，满足条件后会调用

this->OnComponentBeginOverlap.Add(OverlapBeginDele);

this->OnComponentEndOverlap.Add(OverlapEndDele);

这两个委托进行回调，参数的最后一个 FOverlapInfo 是含有一个HitResult类型的变量的,因此可以比较方便的得到碰撞的位置。