无极缩放大地图方案设计

时间	修改人	备注	版本号
2020/02/11	wildgrowl	2D版本	v1.0
2020/02/21	wildgrowl	3D版本	v1.1
2020/03/10	wildgrowl	增加场景数据序列化及动态场景加载	v1.2

> 项目地址：https://github.com/wildgrowl/TestSteplessZooming_3D

demo应满足以下基本目标：

1. 地图足够大
2. 在一定范围内自由缩放
3. 不同缩放比例下分辨率不同

1. 2D UI版本设计

1. 设计方法

1. 预处理

对高分辨率原图切割为M个块，再将每块做Mipmap式的处理，得到N级副本。

2. 资源加载

根据当前缩放比例，动态显示不同分辨率的副本。

3. 绘制

- 使用UGUI Scroll View作为容器，并配置M个元素，每个元素大小为图片块的分辨率
- 通过改变Scroll View的scale值改变缩放比
- 判断每个UI元素（M个块）的可见性，对于不可见的块将其贴图资源卸载

2. 实施步骤

实验采取如下步骤进行：

1. 将原始分辨率为8192 x 4096分辨率的原始地图切割为8份2048 x 2048的地图块
2. 对每个地图块设置4个LOD级别，除原始2048 x 2048分辨率，另设1024 x 1024, 512 x 512, 256 x 256分辨率副本
3. 创建Scroll View并设置8个元素，分别绑定8个地图块

3. 结果分析

在1280 x 720分辨率的屏幕下，设置Scroll View的scale区间为[0.18, 1.0]，得到如下结果：

1. 缩放达到切换阈值时会触发贴图资源的变更
2. 缩放到最小时使用最低的分辨率贴图资源，因此占用内存最低，为**267KB**

3. 缩放到最大时使用最高的分辨率贴图资源，因此占用内存高；根据当前视图内的方块数量，当最多4个时，占用的内存为**8MB**
4. 当视角锁定在中心位置，内存的占用从低到高分别为：**267KB, 1MB, 2MB, 8MB**

2. 3D版本设计

参考并复刻万国觉醒的世界结构设计。

1. 设计方法

设计主要满足以下要求：

1. 超大的世界
2. 多Lod的元素
3. 无极缩放

基于此目的，主要的手段为：

1. 使用一个超大虚拟网格表示整个大世界
2. 使用地图元素（Element类型）表示世界上的物体
3. 使用一个透视相机并借助fov，高度（y,z值同时作用）的参数调整实现缩放

1. 世界的结构

万国觉醒中，整个游戏世界分为了多个区，每个区是一个方形区域，最大值为(1200, 1200)，据初步估算，对应的unity单位是12000x12000。

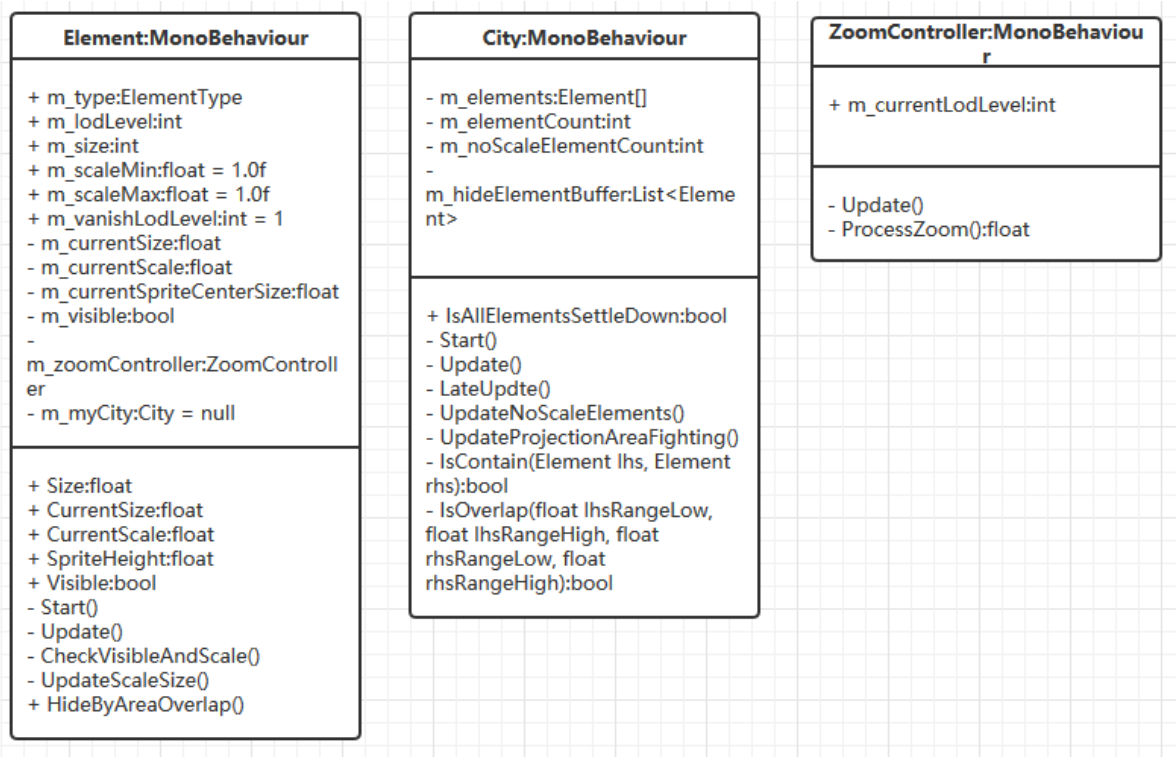
世界中分布着各种资源（可采集）、景观（3D模型的大小山脉，树木）、村落和山洞（可访问），以及最重要的国家，国家即每个玩家的城建系统。

国家内部也分为功能性建筑以及景观性建筑，这些都是可交互的，只有城墙和城墙四周的箭楼是不可移动的，其作用是限制玩家的有效编辑区域（玩家也可以使用迁城道具移动到地图任意一处有效位置）。

万国觉醒的游戏世界可分为三级：世界-区-城市。但在我们的设计中暂时只考虑后两级，并为了方便，将二级结构称为：**区-城市。**

2. 主要数据结构

地图元素（*Element*）类型是系统中最重要的类型，此外重要的类还有表示国家的*Nation*，及控制缩放的*ZoomController*，类图如下：



*Element*挂载到世界上的各种元素上，重要的配置参数的含义为：

1. m_type：类型，世界元素/城建元素
2. m_lodLevel：元素可见的lod等级
3. m_size：元素的占地面积，单位为正方形边长网格数
4. m_scaleMin：随镜头缩放时元素的最小缩放值，默认值为1.0
5. m_scaleMax：随镜头缩放时元素的最大缩放值，默认值为1.0
6. m_vanishLodLevel：当lod等级到达该级，由可见变为不可见（如万国觉醒中的代表国家的塔变为主城的真正模型）
7. m_isBuildingPlant：当视角进入国家并所有元素都可见（缩放值到最小值），景观类元素才可见
8. m_isMainBuilding：主城比较特殊，是国家内最早显示的元素，且会随着缩放发生位移

3D的模型是个例外，没有挂载Element，而是使用了Unity自带的Lod系统，制作多个精度的模型来实现。

*Nation*挂载到国家元素的父节点上，主要执行建筑遮挡计算的逻辑。在万国觉醒中，当城建元素达到显示lod等级并且没有与其他建筑的投影面发生冲突时才予以显示，且初次显示时缩放值为m_scaleMax。比较极端的是主城在初次显示时占据了整个国家的面积，也就是说起m_scaleMax值是比较大的。

该版demo**面积遮挡计算**采用类似于分离轴算法，在这里我们的投影面积简单，只需计算x,z坐标即可，更简单。

*ZoomController*挂载到主相机的父节点，控制相机的缩放。目前只是调整fov，还可以再结合高度调整，因为fov的值不能太大，否则会导致严重的边缘拉伸现象。

此外，还有部分全局参数存储在*GlobalController*静态类中，比如：

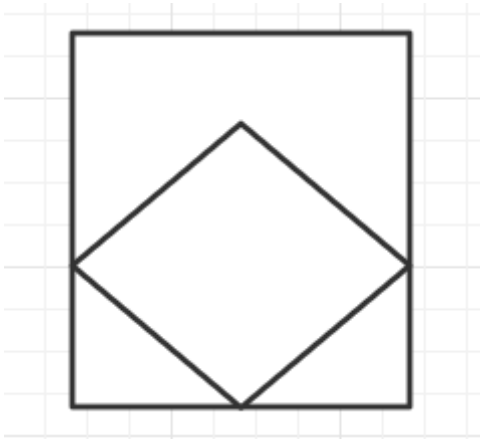
1. LodLevelCount：lod的等级数量
2. CameraFovMin：fov最小值
3. CameraFovLodRangeLow：lod的fov区间下限
4. CameraFovLodRangeHigh：lod的fov区间上限

fov的最大值等于区间上限值。

2. 实施步骤

1. Sprite资源的预处理

世界中大部分元素都是以Sprite的形式展现的，因为有投影面的概念，为了尽可能贴合，对原始图片也有要求，如图所示：



矩形区域为一个元素的投影面积，该区域是经过45°的旋转的，为了让Sprite与投影面贴合，要求主要有以下几点：

1. 图片的宽度为投影面的对角线长度，也就是:size / 2 * sin(45)
2. 图片中建筑的拐角应尽可能贴图片的下边界
3. 图片中建筑的底座拐角应为90°左右，即视角与游戏中一致

此外，在游戏运行时，需要根据当前的缩放值，计算出实际size值，并对图片的z坐标进行一定的偏移，使其始终贴合投影面的下拐角。

那么图片资源的宽度应该如何计算？我们需要明确两个比例，一个是网格与unity单位的比例，另一个是sprite导入设置中像素与unity单位的比例。后者的默认设置是100:1，我们保持该设置。我们实验测试了几种网格与unity单位的比例设置，认为2:1是比较合理的。下面是根据这两个比例，计算得到的常用尺寸与图片资源宽度的对应表：

网格尺寸	图片宽度
3x3	212
5x5	354
6x6	424
7x7	495
9x9	636

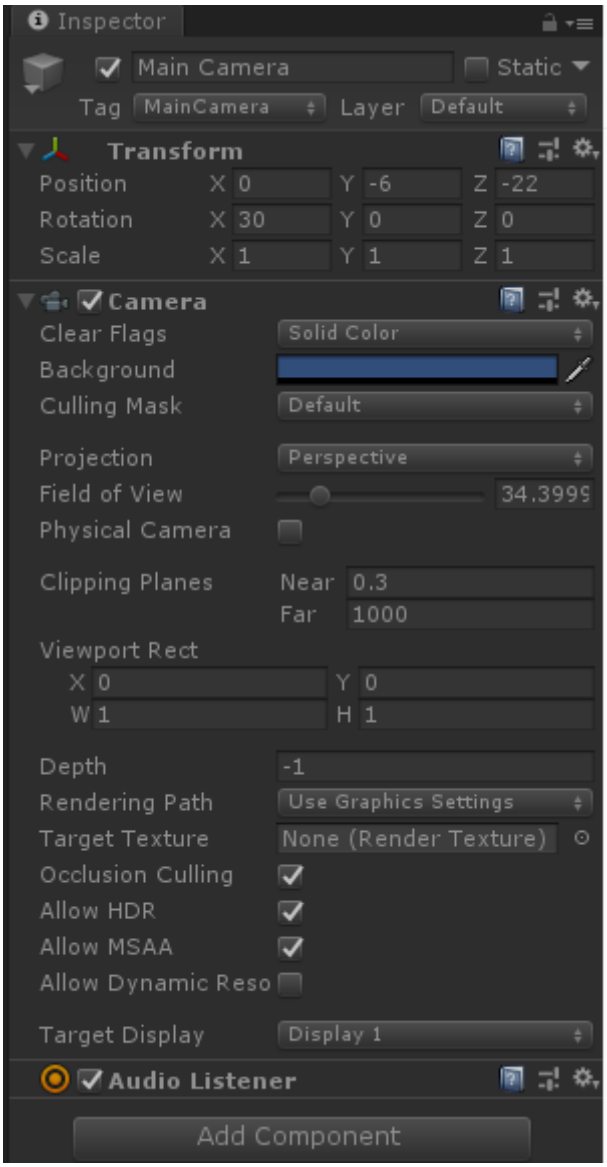
可见，使用636像素宽度的图片作为最大建筑的贴图资源，是比较合理的。

2. 模型的设置

本实验使用unity的官方插件ProBuilder简单制作了2个模型，用来表示粗糙和精细两种状态。使用unity自带的Lod系统，我们只需要为其添加LOD Group组件，并为各个级别添加模型即可。

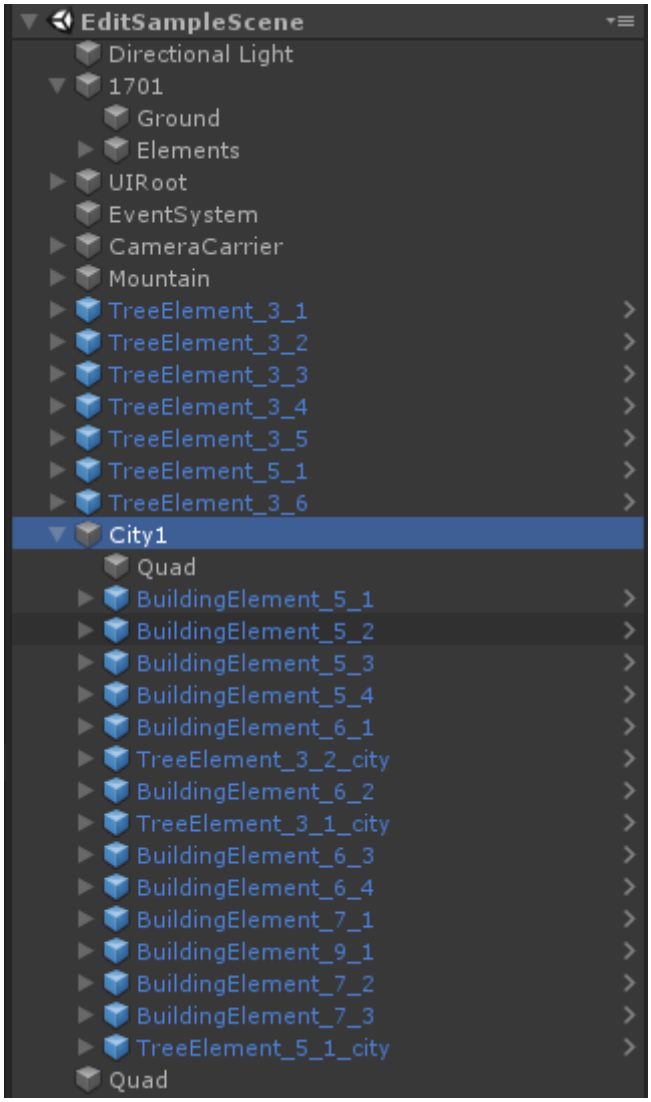
3. 相机的设置

我们对相机适当抬高，并绕x轴旋转了30度，并把它挂载一个空节点下，通过调整这个空节点的transform实现滑动位移。



4. 场景结构

如图所示，场景中布置了地面，山，树和一个城市及其内部建筑元素。

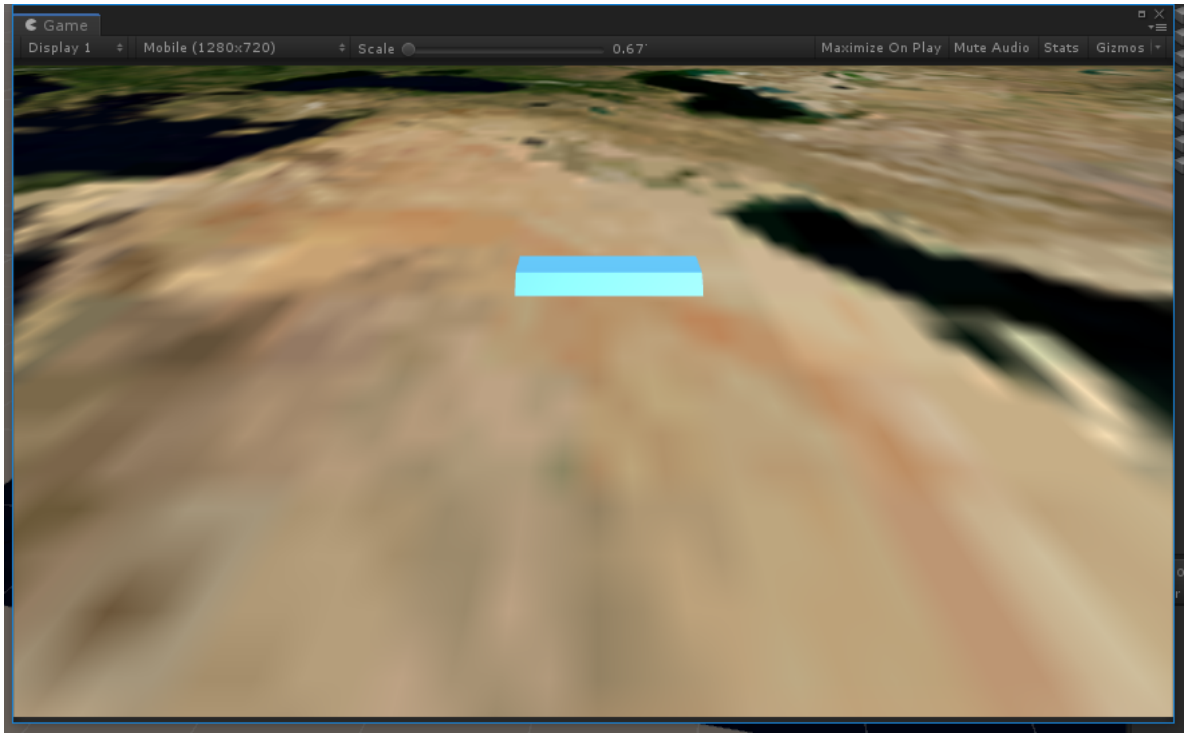


5. 元素的设置

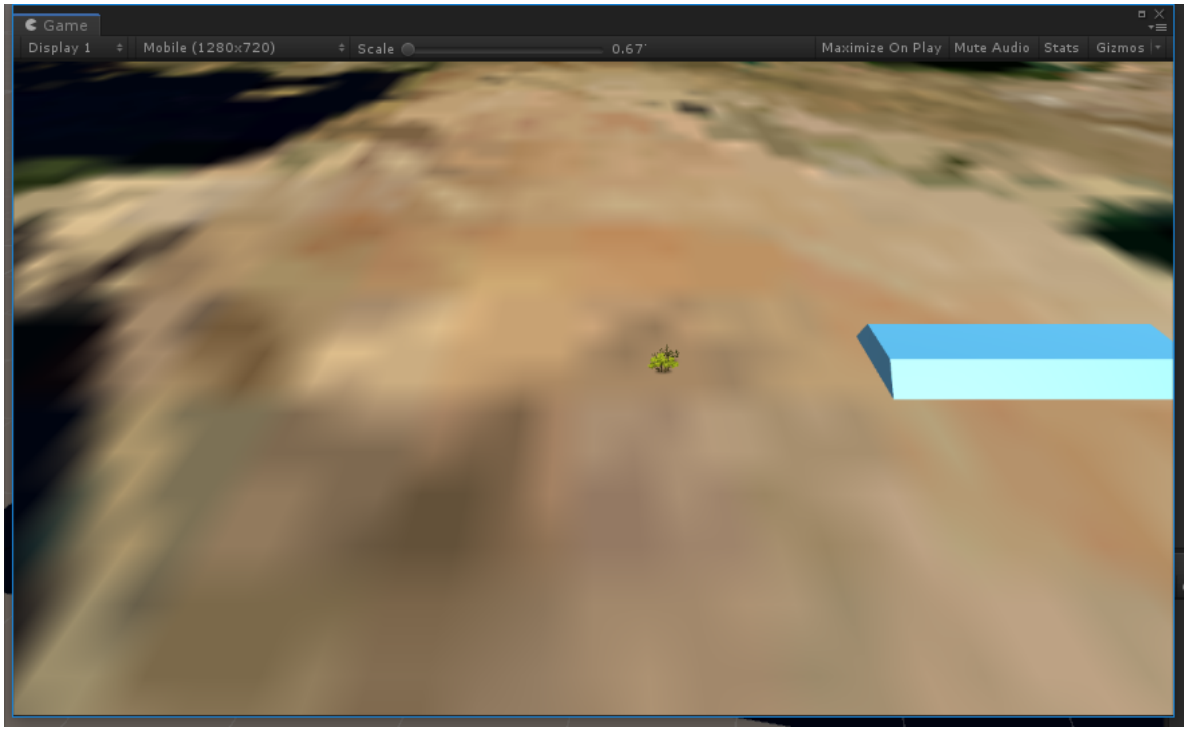
下图是一个建筑的元素设置。

3. 结果分析

下面是试验得到的部分效果截图：



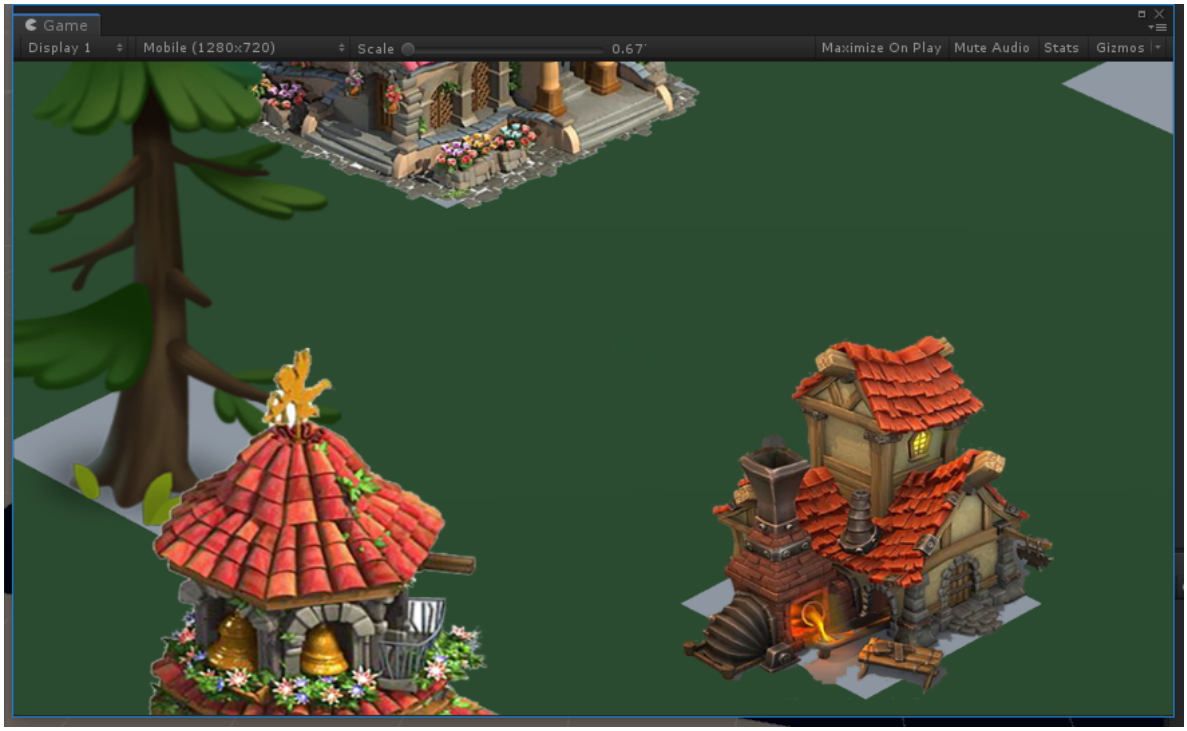
Lod为7，只能看到粗糙的“山”



Lod为5，能看到粗糙的“山”和一棵树

Lod为3，能看到细节丰富的“山”和许多树

Lod为0，能看到部分未被遮挡的建筑，图中按照规则隐藏了建筑，但为观察效果显示了投影面



Lod为0，且拉到最近，能看到全部建筑和景观类元素

注：图中灰色为测试用绘制的投影面，按照标准使用正式资源可以保证贴合。

4. 后续工作

1. 制定编辑和存储规范
2. 分析内存和CPU表现

3. 数据持久化

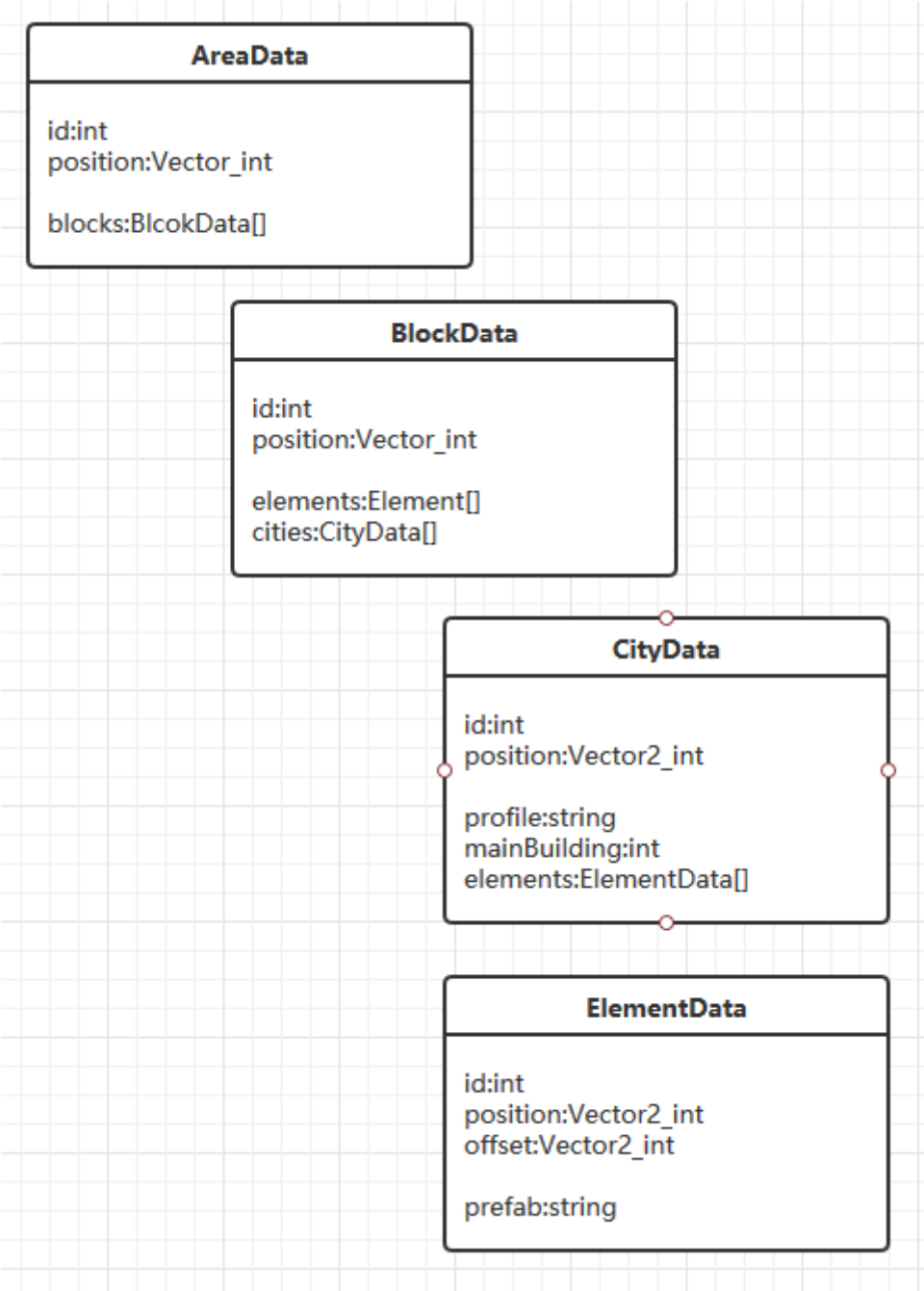
1. 设计方法

1. 将场景元素存储为预制体
2. 在特定的编辑场景下由地编人员通过刷预制体的方式制作场景
3. 遍历场景创建ScriptableObject对象，并导出asset文件
4. 运行时读取asset文件反序列化加载场景

2. 实施步骤

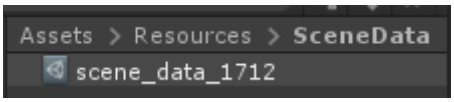
1. ScriptableObject数据结构

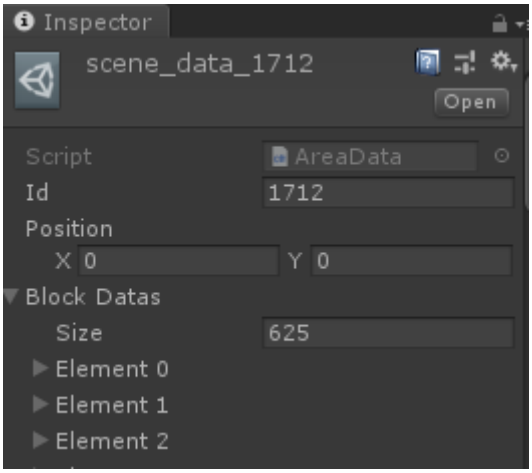
ScriptableObject对象的数据结构如下图所示：

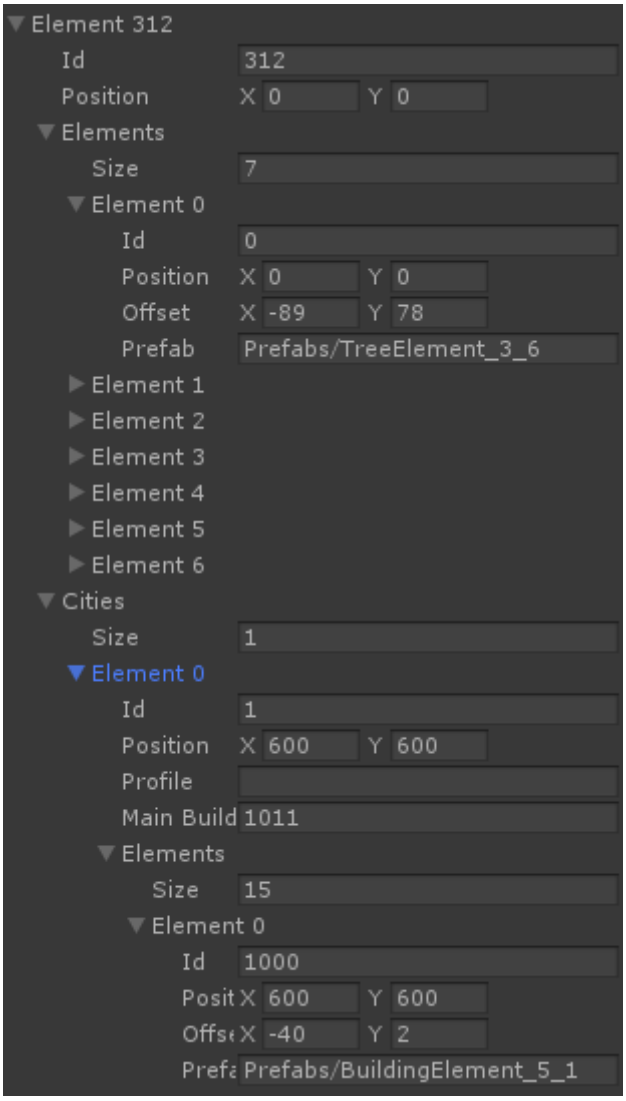


2. 场景数据导出

在编辑场景完成编辑后，使用自定义的菜单项导出ScriptableObject对象，如图示：

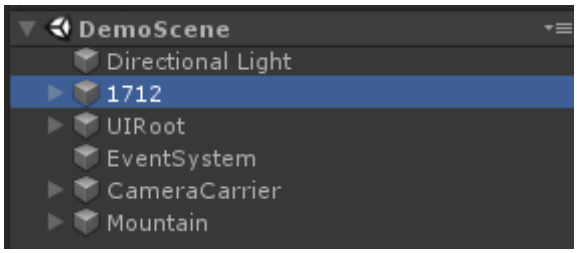


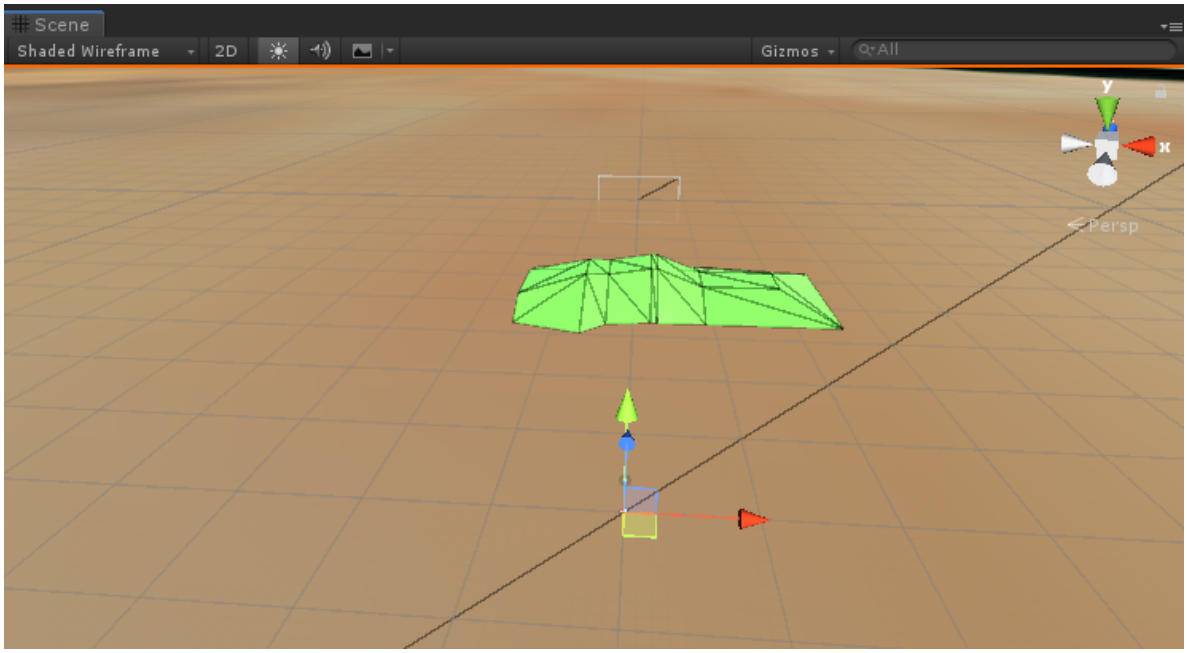




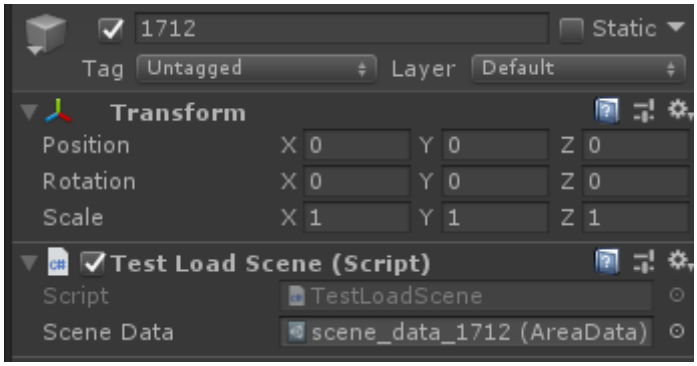
3. 场景数据加载

开启一个测试demo场景，只有一张低分辨率底图和静态的山：

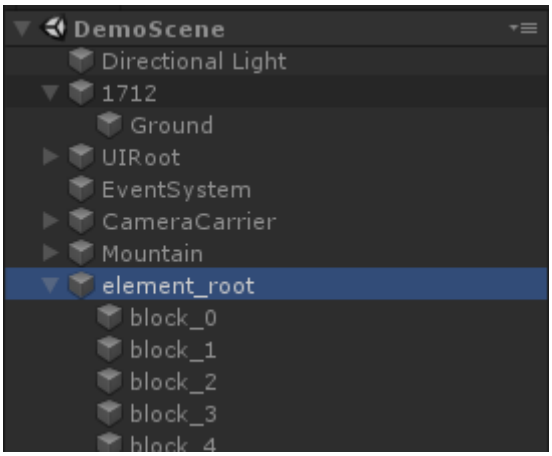


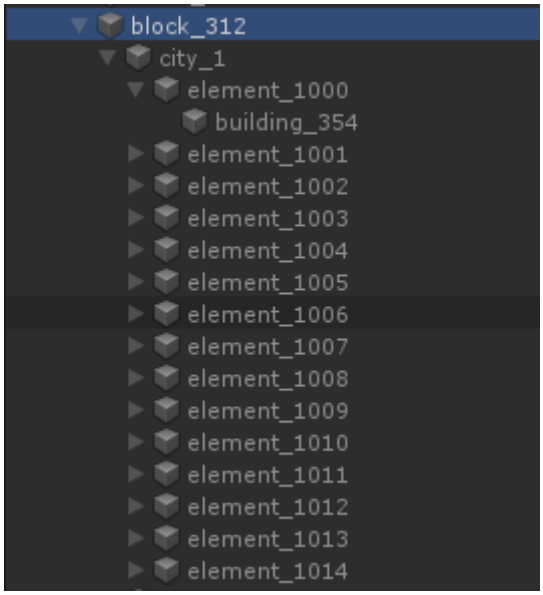


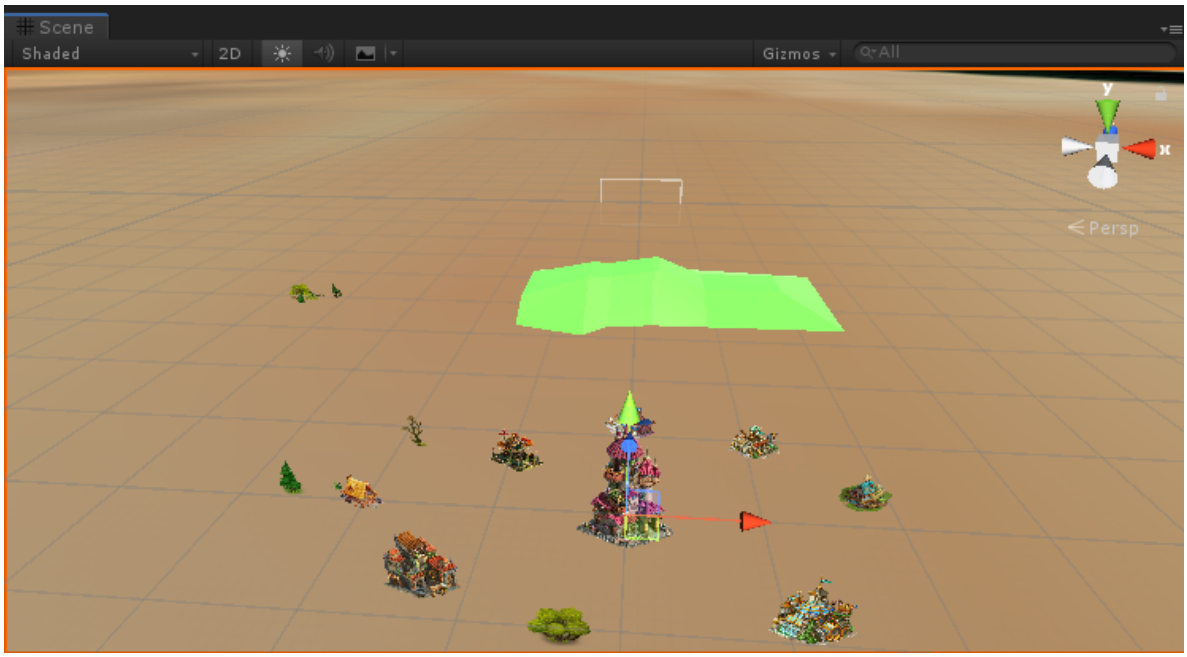
在1712区域节点上挂载了一个测试加载场景的脚本：



运行后经过序列化过的场景元素得到了加载：







3. 后续工作

1. 根据视野分区域动态加载
2. 进行压力测试