

# 地图系统

---

## 地图生成系统整体流程：

先找到数据 → 初始化地图内容（碰撞体、AI导航网格、根据存档恢复地图对象、AI对象、更新可见地图块）  
→ 初始化UI

1. 找到地图存档：找到玩家设定的**地图初始化参数存档**和**地图数据存档**
2. 确定地图对象和AI对象生成配置：根据地图/AI对象生成配置生成一个临时字典，字典Key为顶点类型，Value为配置id
3. 初始化地图生成器：生成地图数据（GenerateMapData）
  - a. 生成网格顶点数据：设置地图真实长宽以及cell的大小，默认mapSize=20（chunk的个数），mapChunk=5（cell个数），cellSize=2（unity标准格子的个数）
  - b. 使用玩家设定的随机种子生成柏林噪声图
  - c. 确定各个顶点的类型以及计算周围网格贴图的索引数字
  - d. 生成地面Mesh
  - e. 设定地图物品生成随机种子
  - f. 计算地图物品配置总权重和计算AI对象配置总权重
4. 初始化地面碰撞体网格
5. 烘焙导航网格：后续AI巡逻需要使用到
6. 根据存档情况判断是否需要加载之前的地图（生成地图块数据）
7. 更新目前可见的地图块
8. 更新和关闭小地图UI（刷新一次）

## 核心脚本：

MapManager

MapChunkController

MapGrid

MapGenerate

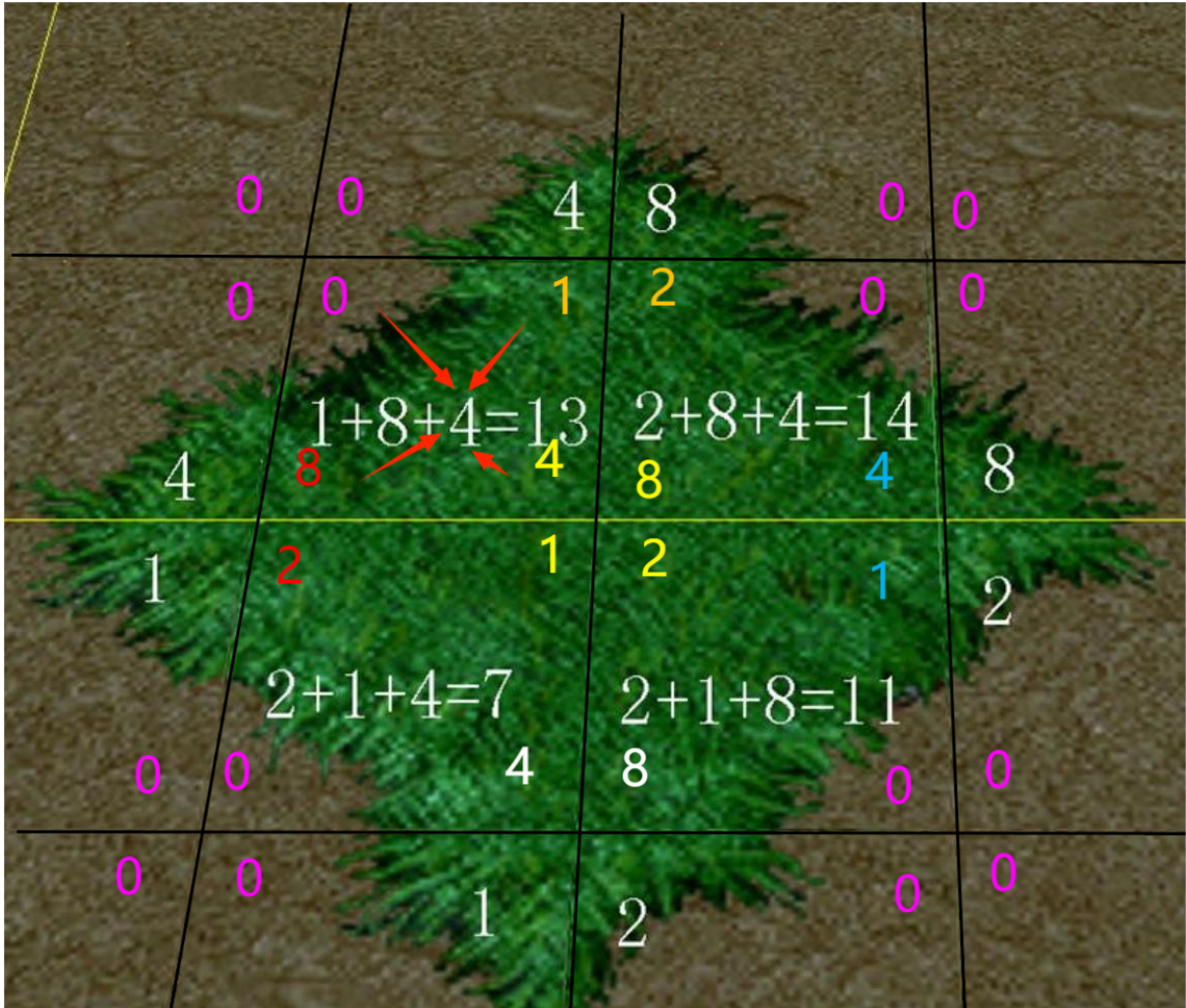
---

## 地图生成器：MapGenerate

生成通用地图块数据 `GenerateMapData()`

1. 统一生成整张地图网格/顶点数据

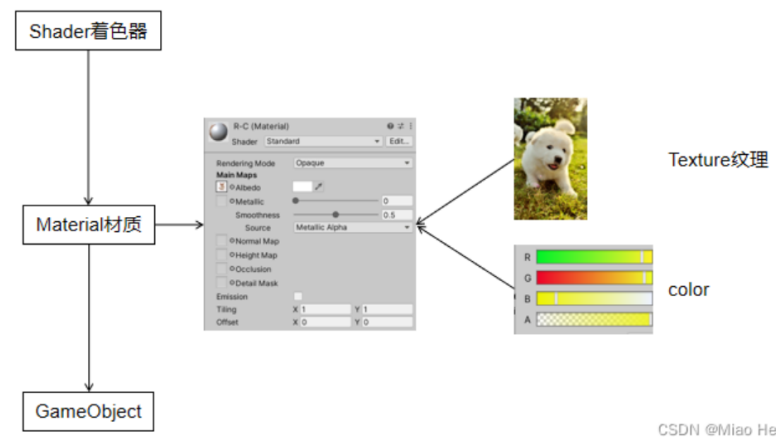
2. 使用随机种子生成随机柏林噪声图 (Unity.Random.InitState)
3. 确定各个顶点的类型以及计算周围网格贴图的索引数字
  - a. 通过魔兽争霸地形贴图拼接算法生成更合理的地形  
<https://blog.csdn.net/twopointfive/article/details/104255705>
  - b.



4. 根据顶点类型实例化森林和沼泽材质
  - a. 需要注意Material / Texture / Shader的关系

### 纹理、着色器与材质的关系：

Shader着色器的属性值配置可以在Material实现，而Material材质在Mesh Renderer中设置，且Mesh Renderer在GameObject物体中。三者关系如下图：



在指定位置生成地图块数据 `MapChunkController GenerateMapChunk`

1. 生成地图块GameObject并挂载MapChunkController、MeshFilter组件

```
// 生成地图块物体
GameObject mapChunkObj = new GameObject("Chunk_" + chunkIndex.ToString());
MapChunkController mapChunk = mapChunkObj.AddComponent<MapChunkController>();
// 生成mesh
mapChunkObj.AddComponent<MeshFilter>().mesh = mapChunkMesh;
```

2. 生成地图块贴图，使用协程分帧执行提高效率
  - a. 渲染当前地图贴图，详情见下方<地图块生成流程-贴图>
  - b. 设置MapChunk的position和parent
  - c. 初始化地图块数据：如果没有存档数据则生成场景物体数据并保存，否则则通过序列化的方式读出数据加载到内存里，详情见下方<地图块生成流程-数据>

## 地图块生成流程

贴图：

1. 计算当前地图块的偏移量，找到这个地图块每块具体的格子位置并统计出所有格子是否都为森林
2. 如果都为森林则不对地面进行渲染
3. 如果包含沼泽则需要遍历MapChunk中包含的所有格子单独进行渲染
  - a. 计算整个MapChunk所需的Texture大小后创建MapChunkTexture

```
int textureCellSize = mapConfig.forestTexture.width;
int textureSize = mapConfig.mapChunkSize * textureCellSize;
mapTexture = new Texture2D(textureSize, textureSize, TextureFormat.RGB24, false);
```

- b. 从左下到右上依次绘制每个Cell中的像素，通过Texture2D中的SetPixel在指定位置设置像素点

```
Color color = mapConfig.forestTexture.GetPixel(x1, z1);
mapTexture.SetPixel(x1 + pixelOffsetX, z1 + pixelOffsetZ, color);
```

- c. 最后设置一下Texture2D的filterMode和wrapMode然后Apply修改后的Texture2D

数据：

1. 需要生成的数据

```
MapChunkData mapChunkData = new MapChunkData();
mapChunkData.mapObjectDataDict = new Serialization_Dict<ulong, MapObjectData>();
mapChunkData.AIDataDict = new Serialization_Dict<ulong, MapObjectData>();
mapChunkData.forestVertexList = new List<MapVertex>();
mapChunkData.marshVertexList = new List<MapVertex>();
```

2. 顶点数据：可以直接根据世界坐标从地图数据中获取

3. MapChunk中包含的地图对象数据 `mapObjectDataDict`

- a. 遍历MapChunk中包含的Cell

- i. MapCell顶点类型 == 空 不生成地图对象
- ii. MapCell顶点类型 != 空 则根据MapCell顶点类型和地图对象随机生成权重去生成对应结果，其中MapObject Id由MapData统一管理，每次创建新对象Id + 1

```
private MapObjectData GenerateMapObjectData(int mapObjectConfigId, Vector3 position, int destoryDay) {
    MapObjectData mapObjectData = PoolManager.Instance.GetObject<MapObjectData>();
    mapObjectData.id = mapData.currentId;
    mapData.currentId += 1;
    mapObjectData.configId = mapObjectConfigId;
    mapObjectData.position = position;
    mapObjectData.destoryDay = destoryDay;
    return mapObjectData;
}
```

4. MapChunk中包含的AI对象数据 `AIDataDict`

- a. 首先判断当前MapChunk中包含的顶点类型数量是否超过阈值，例如沼泽顶点数量少的话意味着沼泽很小不适合在该地形中放置AI物体

## 地图顶点/网格数据：MapGrid

注意：MapGrid是构建整个地图的坐标关系，MapChunkIndex，MapCellIndex，WorldPosition是三种不同的坐标，他们之间有关联，目的是通过真实的世界坐标能够找到具体的chunk、cell即可

主要功能：

1. 添加地图顶点（Vertex）：`Dictionary<Vector2Int, MapVertex> vertexDict`
  - a. 其中Key是MapCell坐标体系中的具体坐标（该坐标应该可以通过MapChunk去索引到），Value中position真是世界坐标
  - b. 顶点坐标中包含世界坐标position、顶点类型vertexType、当前Cell中地图对象Id mapObjectId
2. 添加地图网格（Cell）：`Dictionary<Vector2Int, MapCell> cellDict`
  - a. 其中Key是MapCell坐标体系中的具体坐标（该坐标应该可以通过MapChunk去索引到），Value中position真是世界坐标左下角的一处坐标 `new MapCell() { position = new Vector3(x * cellSize - offset, 0, z * cellSize - offset) }` 这样做的是贴图算法具体执行逻辑导致的
  - b. 地图网格中包含贴图位置position和贴图素材textureIndex
3. 计算格子贴图的索引数字：通过noiseMap的值设定Cell地图顶点类型

## 地图系统数据结构

玩家设定的地图初始化参数存档：

1. `public class MapInitData`：
  - a. 地图大小(`mapSize`)
  - b. 地图种子(`mapSeed`)
  - c. 沼泽范围(`marshLimit`)
  - d. 生成种子(`spawnSeed`)


地图数据存档：按照地图上包含对象的层级可以分成 MapData → MapChunkData → MapObjectData

1. `public class MapData` 地图数据
  - a. `public ulong currentId` 管理整个地图上的id序列
  - b. `public List<Serialization_Vector2> MapChunkIndexList` 当前地图包含哪些地图块，使用地图块顶点作为key来管理地图块
2. `MapChunkData` 地图块数据
  - a. `public Serialization_Dict<ulong, MapObjectData> mapObjectDataDict` 地图对象字典，用来检索地图对象
  - b. `public Serialization_Dict<ulong, MapObjectData> AIDataDict` AI对象字典，用来检索AI对象
  - c. `[NonSerialized]public List<MapVertex> forestVertexList` 用来记录当前森林顶点数量
  - d. `[NonSerialized]public List<MapVertex> marshVertexList` 用来记录当前沼泽顶点数量
3. `MapObjectData` 地图上的物体数据

- `public ulong id` 地图对象id, 唯一标识
- `public int configId` 地图对象对应配置文件
- `public int destoryDay` 地图对象销毁天数
- `private Serialization_Vector3 sv_position` 地图对象位置

地图对象配置：

Key	Value
0	森林空 (Map Object Config)
1	树1 (Map Object Config)
2	树2 (Map Object Config)
3	树3 (Map Object Config)
4	树4 (Map Object Config)
5	树5 (Map Object Config)
6	石头1 (Map Object Config)
7	小石头 (Map Object Config)
8	石头2 (Map Object Config)
9	石头3 (Map Object Config)



树1 (Map Object Config)

Open

Script

空的(不生成物品)

地图顶点类型

预制体

物体icon

物体icon尺寸

描述

生成概率(百分比类型)

腐烂天数

MapObjectConfig

Forest

tree\_01

Tree

1

3

-1

AI对象配置

Key	Value
0	森林空 (AI Config)
1	野猪1 (AI Config)
2	野猪2 (AI Config)
3	野猪3 (AI Config)
4	蜘蛛5 (AI Config)
5	沼泽空 (AI Config)

Script	AIConfig
空的(不生成物品)	
地图顶点类型	Forest
预制体	Bora_1
生成概率(百分比类型)	10
腐烂天数	-1