地图生成系统

重点部分：

1. 地图地面：如何生成整张大地图
2. 数据：如何保存和更新地图数据
3. UI：如何生成和更新小地图

核心脚本：

1. MapManager.cs
2. MapChunkController.cs
3. MapGrid.cs
4. MapGenerate.cs
5. MapData.cs
6. MapConfig.cs

地图管理器 MapManager

说明：主要功能有地图地面生成、地图对象生成、AI对象生成、地图数据存储管理

初始化时生成地图：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MapInitData和地图数据存档MapData
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：（MapGenerate）<***生成地图数据***>
4. 初始化地面碰撞体网格，<***生成整个地面的mesh***>
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）
    a. 旧存档加载：
        i. 根据存档数据设置地图块加载最大数量，初始化加载进度条
        ii. 遍历所有的MapChunkIndexList，根据地图块索引<***获取单个地图块数据***>，根据下标和数据<***生成地图块管理器***>（管理器中数据和一些显示的对象，例如树木、AI等）
    b. 新存档创建：

i. <***游戏初始化时获得地图块的初始数量***>

ii. 根据玩家所在地图块计算可视范围内所有地图块索引并<***生成地图块管理器***>

7. <***更新目前可见的地图块***>
8. <***更新和关闭小地图UI***>

生成地图块管理器

1. 传入地图块索引和地图块数据（可能为空）
2. 判断地图块索引是否合法
3. 使用（地图生成器）<***在指定位置生成地图块控制器***>MapChunkController
4. 将地图块数据添加到MapChunkDict中，并将当前地图块索引添加到地图UI更新索引列表中（用于显示小地图UI）

通过世界坐标获得地图块索引

1. 索引X值：当前x值除一个地图块包含的所有网格数量`x / (mapConfig.mapChunkSize * mapConfig.cellSize)`，Z值同理

通过世界坐标获得地图块控制器

1. <***通过世界坐标获得地图块索引***>
2. 根据地图块索引坐标在MapManger中持有的MapChunkData字典里获取对应的地图块控制器

生成一个地图对象：对其他系统提供的地图对象生成方法

通过”给定的地图Id”和”位置”在”指定的地图块控制器”上生成一个地图对象

1. （MapGenerate）<***生成一个地图对象数据***>
2. （MapChunkController）<***添加一个地图对象***>，将数据绑定到
3. 在小地图上（UI_MapWindow）<***添加地图对象icon***>

在地图块刷新时生成地图对象列表

为了让MapChunkController能访问到MapGenerate方法, 因此在这里生成了同名/同功能函数

在地图块刷新时生成AI对象列表

为了让MapChunkController能访问到MapGenerate方法, 因此在这里生成了同名/同功能函数

游戏初始化时获得地图块的初始数量

1. 获得玩家在地图上的地图块索引

2. 根据玩家可视范围确定显示地图块索引，例如可视距离为1的话，那玩家可视范围则为 x → (-1, 1); y → (-1, 1) 。判断玩家可视范围是否超过地图边界，对于在边界内的地图块，需要+=到初始化数量上

更新目前可见地图块

1. 获得玩家坐标：<***通过世界坐标获得地图块索引***>
2. 移除不在视野内的地图块：遍历当前最后显示的地图块列表，如果地图块不在玩家显示范围内 `Mathf.Abs(chunkIndex.x - currChunkIndex.x) > mapConfig.viewDistance` 则将<***控制地图块显示***>设置为隐藏，并将地图块从最后显示地图块列表中移除
3. 显示在视野中出现的地图块：根据当前玩家坐标计算当前在视野范围内的地图块下标，如果对应地图块在MapChunkDict中则将其加入最后显示的地图块列表里进行缓存，如果不在则说明地图块还未初始化，需要<***生成地图块管理器***>
4. 控制地图块刷新时间：设置当前更新Chunk标志为false，当时间间隔≥UpdateVisibleChunkTime时才可<***重置更新Chunk标志***>

重置更新Chunk标志

```
canUpdateChunk = true;
```

更新目前可见地图块并更新UI坐标

1. 当玩家没有移动或者地图块更新Chunk标志为false时不刷新可见地图块和更新UI坐标
2. 记录当前玩家最后位置
3. <***更新地图UI中心点***>
4. <***更新目前可见地图块***>

显示地图UI

1. 检查地图UI是否初始化完成，如果地图Ui未初始化完成则优先进行<***地图UI初始化***>
2. 显示地图UI窗口

更新地图UI

1. 遍历地图管理器持有的地图UI更新索引列表
    a. 获得地图块索引并根据索引从MapChunkDict中拿到地图块控制器
    b. 如果当前地图块包含沼泽则texture设置为当前地图块MeshRenderer.material.mainTexture，否则设置为null
    c. 否则向小地图中<***添加地图块UI***>
2. 传入玩家坐标<***更新地图UI中心点***>

关闭地图UI

1. 关闭UI窗口

烘焙导航网络

```
navMeshSurface.BuildNavMesh();
```

保存地图数据

直接调用存档管理器保存地图数据方法进行保存

游戏关闭时释放场景中的资源

1. 释放小地图UI资源
2. 遍历所有的地图块控制器MapChunkDict，挨个释放地图块控制器中的资源

地图生成器 MapGenerate

生成地图数据：计算和初始化后续功能所依赖的各项数据

1. 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）
2. 使用玩家设定的随机种子<***生成柏林噪声图***>
3. （MapGrid）<***根据perlin噪声图设置全图的顶点类型和格子贴图索引数字***>
4. 预先实例化出森林和沼泽材质为后续生成地图做准备
5. 生成单个地图块Mesh
6. 设定地图物品生成随机种子
7. 计算地图物品配置总权重和计算AI对象配置总权重

生成柏林噪声图

1. 根据传入的地图宽高`mapInitData.mapSize * mapConfig.mapChunkSize`即地图中横向和纵向cell的数量申请二维数组用来存储perlin噪声地图
2. 遍历地图宽高，当前cell位置对应的柏林噪声值可由`noiseMap[x, z] = Mathf.PerlinNoise(x * lacunarity + offsetX, z * lacunarity + offsetY);`计算得来

生成单个地图块Mesh

1. 创建一个新的mesh

2. 确定mesh顶点以及由哪些顶点组成三角形，设置UV后重新计算法线

在指定位置生成地图块控制器

1. 实例化GameObject：生成地图块GameObject并挂载MapChunkController、MeshFilter组件
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<***生成地图块纹理>***，返回texture和isAllForest标记
   b. 添加MeshRenderer组件
      i. 如果当前地图块全是森林则使用共享材质，则在添加完MeshRenderer组件后设置shareMaterial为地图配置中的默认material `mapChunkObj.AddComponent<MeshRenderer>().sharedMaterial = mapConfig.mapMaterial;`
      ii. 如果当前地图块中存在沼泽则创建一个Material（材质为沼泽材质），同时设置mainTexture设置为<***生成地图块纹理>***中返回的带有沼泽的Texture。然后添加MeshRenderer组件，将material设置为刚刚的沼泽material
3. 设置MapChunkController的position和parent
4. 初始化地图块数据
   a. 如果没有存档数据则<***生成地图块数据***>然后<***添加并保存单个地图块数据***>
   b. 如果有存档数据则<***恢复地图块顶点数据>***

生成地图块纹理

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（地面默认是森林）
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. 约定好贴图都是矩形, 计算整个地图块texture的宽高（cell数 * 单个贴图的宽/高）

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

   b. 遍历所有格子并绘制格子中的每个像素点，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。需要注意沼泽贴图中像素点是透明的情况也需要绘制forestTexture同位置的像素颜色

```
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

通过地图权重配置得到一个地图物体id

1. 根据顶点类型获取对应地图物体生成权重配置
2. 确定地图对象生成权重总和并得到(1, 权重总和)之间的一个随机值作为阈值
3. 遍历当前地图顶点类型对应的地图物体生成权重配置
    a. 如果当前地图对象生成概率大于阈值则返回当前地图对象配置Id

通过地图权重配置得到一个AI物体id

1. 根据顶点类型获取对应AI物体生成配置权重

生成一个地图对象数据

说明：地图对象数据包含id、configId、destroyDay、position信息

1. 根据传入的地图对象configId、位置、销毁天数设置地图对象数据
2. 将MapData中currentId设置为地图对象id，设置后currentId += 1

生成一个地图块AI对象数据

生成一个AI对象数据

注意：一个地图块内生成AI物体有数量限制，同时如果当前地图块森林/沼泽顶点的数量不够也不能生成AI物体，这是为了限制AI物体生成数量

1. 检查当前地图块数据中AI物体是否达到数量上限
2. 检查当前地图块包含的森林/沼泽顶点类型数量是否超过阈值
3.

生成地图块数据

说明：生成MapChunkData中mapObjectDataDict、AIDataDict、forestVertexList、marshVertexList即可

1. 初始化需要生成的MapChunkData中各种成员变量
2. 生成地图对象数据 mapObjectDataDict 和地图块顶点列表 forestVertexList、marshVertexList
    a. 遍历整个地图块，根据传入的地图块索引计算cell偏移位置
        i. 根据实际位置<***获取顶点数据***>，将对应的顶点类型加入到地图块顶点列表中
        ii. <***通过地图权重配置得到一个地图物体id***>并根据地图物体id得到对应配置
        iii. 设置地图物体位置为当前cell顶点前后上下的某处随机位置
        iv. 记录当前cell顶点生成的地图对象id
        v. <***生成一个地图对象数据***>并将其加入到 mapObjectDataDict 中
3. 生成AI对象数据 AIDataDict

a.

地图网格 MapGrid

注意：地图网格主要用来计算地图系统中顶点数据（顶点位置、顶点类型、顶点上的地图物体）和格子数据（格子位置、格子纹理索引）。游戏地图为矩形，长宽相等且长为`mapInitData.mapSize * mapConfig.mapChunkSize`

地图网格初始化

1. 保存当前地图宽高和cell大小
2. 遍历整个地图cell，依次<***添加顶点数据***>和<***添加格子数据***>

添加顶点数据

1. 根据传入的顶点坐标直接向顶点字典vertexDict中添加顶点数据，其中顶点类型和顶点上地图物体暂不赋值

获取顶点数据

1. 根据传入的顶点索引查看顶点字典中是否包含顶点数据，如果有则返回对应数据没有返回null

添加格子数据

1. 根据传入的顶点坐标直接向格子数据字典cellDict中添加格子数据

获取格子数据

1. 根据传入的顶点索引查看格子数据字典cellDict中是否包含格子数据，如果有则返回对应数据没有返回null

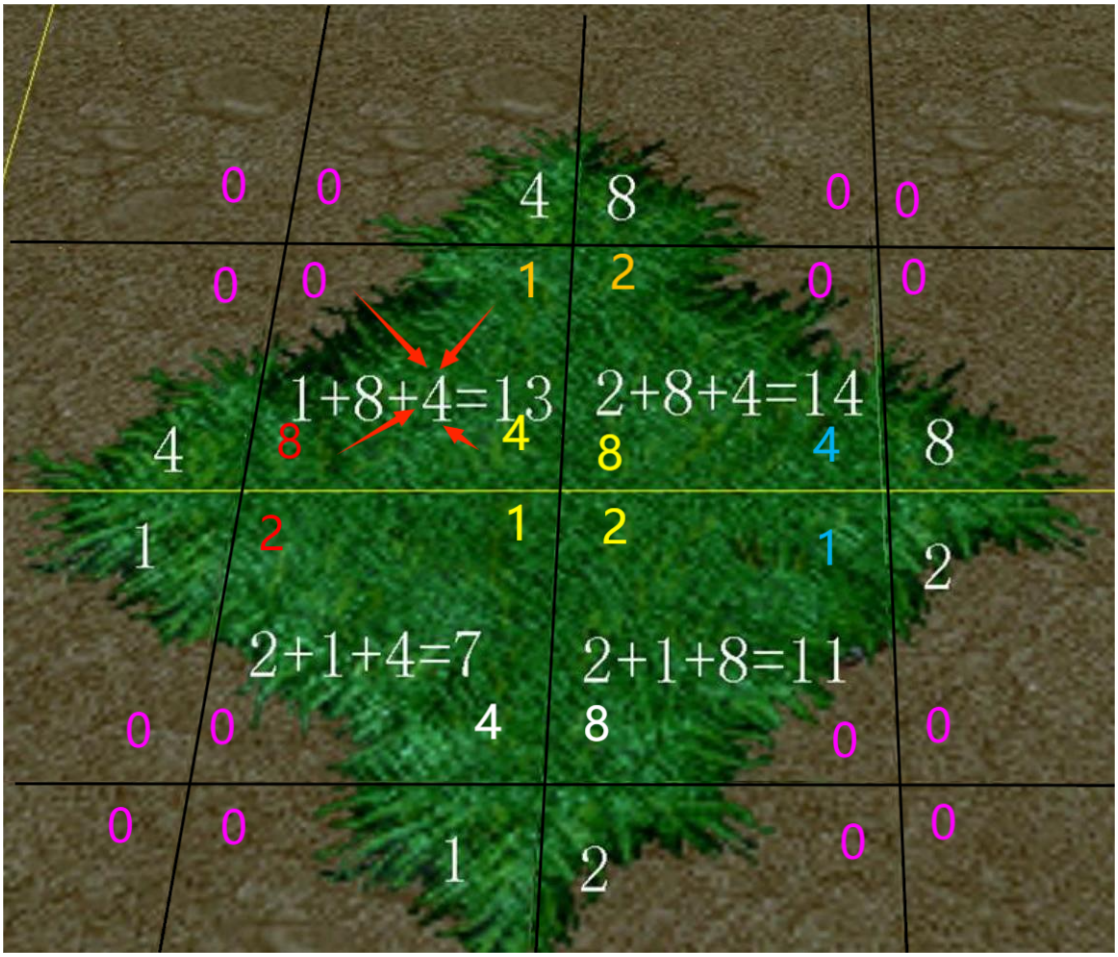
获取左上、右上、左下、右下格子数据

此4个函数功能相似，例如获取左上格子数据`GetCell(vertexIndex.x, vertexIndex.y + 1)`其他函数类似

设置顶点类型和计算格子贴图纹理索引

1. 根据传入的顶点位置和顶点类型设置当前顶点数据
2. 如果传入的顶点类型和当前顶点数据中类型不一致
    a. 更新当前顶点数据中顶点类型
    b. 重新计算顶点附近贴图
        i. 如果是顶点类型是森林则不需要计算

ii. 如果顶点类型是沼泽则需要根据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



具体计算过程：获得左下、右下、左上、右上贴图数字，直接求和得到当前格子数据沼泽纹理索引

根据perlin噪声图设置全图的顶点类型和格子贴图索引数字

1. 遍历二维perlin噪声图结果
    a. 如果当前位置噪声图结果≥阈值则将对于的cell顶点类型设置为沼泽，否则设置为森林，详情见<***设置顶点类型和计算格子贴图纹理索引***>

地图生成器 MapGenerate旧内容

恢复地图块顶点数据

由于地图块中其他数据已经通过存档管理器存入了mapChunkData中，仅顶点数据无法序列化因而未存储到mapChunkData中，所以需要单独恢复

1. 根据传入的地图索引得到世界坐标

2. 遍历当前地图块中所有cell的顶点类型，添加到mapChunkData中

地图块控制器 MapChunkController

注意：

1. 地图块控制器是在地图块GameObject对象实例化的时候挂载到其身上的，因此控制器只用来存储当前地图块数据、动态生成和回收地图上的物体、控制当前地图块上的物体显示/隐藏等功能，并不提供初始化地图块GameObject的方法
2. 控制器会持有当前地图块上的所有地图对象/AI对象，地图对象/AI对象也会持有地图块控制器的引用

初始化

1. 根据传入的地图块索引、地图块数据初始化控制器中的地图数据，例如地图/AI对象实例和地图/AI对象数据。根据mapObjectDataDict生成后续可能需要销毁的地图对象字典，例如地面上的武器当到达销毁天数时需要在地图上移除该对象
2. 添加<***地图块刷新事件***>

控制地图块显示

1. 如果当前地图块显示变量与传入变量不一致，则设置地图块显示变量为当前变量（取反）并设置gameObject.SetActive(地图块显示变量)
2. 如果当前为显示状态
   a. 遍历当前地图块数据中的地图对象数据字典传入数据并<***实例化地图对象***>
   b. 遍历当前地图块数据中的AI对象数据字典传入数据并<***实例化AI对象***>
3. 如果当前为隐藏状态
   a. 遍历当前地图块数据中的地图对象字典，将地图对象放入对象池中
   b. 遍历当前地图块数据中的AI对象字典，将AI对象放入对象池中
   c. 清空当前地图对象和AI对象字典

实例化地图对象

1. 根据传入的地图对象数据拿到对应id并获取到配置信息
2. 从对象池中实例化一个地图对象
3. 设置地图对象的位置为传入数据中的位置
4. 调用`MapObjectBase`初始化方法声明当前地图对象id和对应的地图块控制器
5. 将地图对象加入到地图对象字典中

实例化AI对象

1. 根据传入的AI对象数据拿到对应id并获取到配置信息
2. 从对象池中实例化一个地图对象
3. 如果AI对象坐标未初始化则<***获取AI可以到达的随机坐标***>
4. 调用`AIBase`初始化方法<***初始化AI对象***>，例如设置位置、血量、初始状态(Idle)
5. 将AI对象加入到AI对象字典中

移除一个地图对象

1. 数据层面：根据传入的id从地图对象数据字典（MapObjectDataDict）中移除对应数据，并将out出的数据放入对象池
2. 显示层面：1. 根据传入的id从地图对象字典（MapObjectDict）中移除对象，并将out出的MapObjectBase放入对象池；2. 在小地图UI上直接<***移除地图对象icon***>

移除一个AI对象：游戏物体、存档数据都需要删除

1. 数据层面：根据传入的id从AI对象数据字典（AIDataDict）中移除对应数据，并将out出的数据放入对象池
2. 显示层面：1. 根据传入的id从AI对象字典（AIObjectDict）中移除对象；2.调用AI对象<***销毁方法***>，将AI对象放入对象池，停止当前AI状态

移除一个AI对象：物体迁移，当AI从一个地图块移动到另一个地图块时触发该方法，由于物体迁移不需要删除游戏对象，因此只需要原MapChunkController中删除AI数据并且在新的MapChunkController中添加AI数据即可，可查看<添加一个AI对象：物体迁移>

1. 根据传入的id从AI对象数据字典（AIDataDict）和AI对象字典（AIObjectDict）中移除

添加一个地图对象

1. 数据层面：根据传入的数据，直接在地图对象数据字典中添加该数据，如果该地图对象数据存在销毁天数则将该地图对象加入待销毁地图字典中
2. 显示层面：如果当前地图块已显示（isActive==true）则需要<实例化地图对象>

添加一个AI对象

1. 数据层面：根据传入的数据，直接在AI对象数据字典中添加该数据
2. 显示层面：如果当前地图块已显示（isActive==true）则需要<实例化AI对象>

添加一个AI对象：物体迁移，当AI从一个地图块移动到另一个地图块时触发该方法

1. 将传入的AI对象添加到地图块AI对象字典中
2. 将传入的AI对象数据添加到地图块AI对象数据字典中
3. 设置AI对象的transform.parent为当前mapChunkController.transform
4. 调用AI对象<迁移时的初始化方法>记录当前AI对象所在的地图块控制器

获取AI可以到达的随机坐标

1. 根据传入的AI所对应的Cell顶点类型获取到当前地图块所包含的所有顶点数据
2. 随机选择一份顶点数据，使用NavMesh.SamplePosition去判断能否到达，如果能到达则返回对应的position作为随机坐标

```
// 确保AI可以到达该位置
if (NavMesh.SamplePosition(mapVertexList[index].position, out NavMeshHit hitInfo, 1, NavMesh.AllAreas)) {
    return hitInfo.position;
}
```

3. 如果无法到达则继续递归搜索AI可以到达的随机坐标

地图块刷新事件：移除满足天数的地图物体，并根据当前时间更新AI物体

1. 移除地图块上的物体：
    a. 遍历所有可能需要销毁的地图对象，更新剩余时间
    b. 遍历所有剩余时间为0的地图对象，直接<***移除一个地图对象***>
2. 在地图块上刷新物体：
    a. 每天<***在地图块刷新时生成地图对象***>，<***添加一个地图对象***>到地图块控制器中
    b. 每隔3天<***在的地图块刷新时生成AI对象***>，<***添加一个AI对象***>到地图块控制器中

保存地图数据

1. 当整个地图块数据被销毁时(例如关闭游戏), 需要将当前新的地图块数据保存到磁盘上，详情见（存档管理器）<***保存地图块数据***>

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

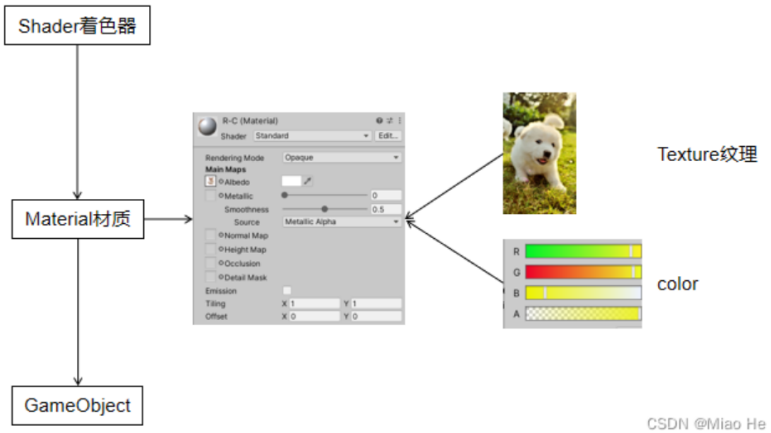
a. 通过魔兽争霸地形贴图拼接算法生成更合理的地形
https://blog.csdn.net/twopointfive/article/details/104255705

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：



地图块生成流程

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

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vertextType、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：

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. 地图数据`MapData`
2. 地图块数据`MapChunkData`
3. 地图上的物体数据`MapObjectData`

```
// 地图初始化数据
[Serializable]
public class MapInitData {
    public int mapSize;         // 地图大小(地图块数量)
    public int mapSeed;         // 地图随机种子
    public int spawnSeed;       // 地图对象随机种子
    public float marshLimit;    // 沼泽生成变量
}

// 地图数据
[Serializable]
public class MapData {
    // 当前地图对象id取值
    public ulong currentId = 1;             // 整个地图当前生成对象Id
    // 地图块索引列表: 已经生成过的所有地图块
    public List<Serialization_Vector2> MapChunkIndexList = new List<Serialization_Vector2>();   // 地图上包含的地图块索引
}

// 地图块对象数据
[Serializable]
public class MapObjectData {
    public ulong id;                                // 地图中地图对象id
    public int configId;                            // 地图物体配置id
    public int destoryDay;                          // 地图对象销毁天数, -1代表无效
    private Serialization_Vector3 sv_position;      // 坐标: sv_postion存档用, position外部调用用
    public Vector3 position {
        get => sv_position.ConverToVector3();
        set => sv_position = value.ConverToSVector3();
    }
}

// 地图块数据
[Serializable]
public class MapChunkData {
    // 当前地图上的所有地图对象
    public Serialization_Dict<ulong, MapObjectData> mapObjectDataDict;  // 地图对象字典, 使用Id检索地图对象
    public Serialization_Dict<ulong, MapObjectData> AIDataDict;         // AI对象字典, 使用Id检索AI对象
    // 记录当前地图块顶点数量
    [NonSerialized]
    public List<MapVertex> forestVertexList;            // 记录当前森林顶点数量
    [NonSerialized]
    public List<MapVertex> marshVertexList;             // 记录当前沼泽顶点数量
}
```

```
// 顶点类型
public enum MapVertexType {
    None,   // 默认类型
    Forest, // 森林
    Marsh,  // 沼泽
}

// 地图顶点: 每个cell有一个顶点, 每个顶点上都有一个地图对象
public class MapVertex
{
    public Vector3 position;
    public MapVertexType vertexType;
    public ulong mapObjectId;           // 当前地图cell上的物体对象, 0代表为空
}

// 地图格子
public class MapCell
{
```

```
    public Vector3 position;
    public int textureIndex;
}
```

地图系统配置

地图配置：

```
public class MapConfig : ConfigBase {
    [LabelText("地图块包含网格数")]
    public int mapChunkSize;                // 地图块大小
    [LabelText("地图块网格大小")]
    public float cellSize;                  // 网格大小
    [LabelText("噪声图采样间隔大小")]
    public float noiseLacunarity;           // 噪声图采样间隔大小
    [LabelText("森林贴图")]
    public Texture2D forestTexture;         // 森林贴图
    [LabelText("沼泽贴图")]
    public Texture2D[] marshTextures;       // 沼泽贴图
    [LabelText("地图材质")]
    public Material mapMaterial;            // 地图材质
    [LabelText("玩家可视距离")]
    public int viewDistance;                // 玩家可视距离, 单位为ChunkSize
    [LabelText("地图块刷新概率")]
    public int mapChunkRefreshProbability;  // 每天早晨地图块刷新地图物品的概率

    [Header("AI")]
    [LabelText("地图块AI数量限制")]
    public int maxAIOnChunk;
    [LabelText("森林/沼泽地图块生成AI最小顶点数")]
    public int generateAIMinVertexCount;
}
```