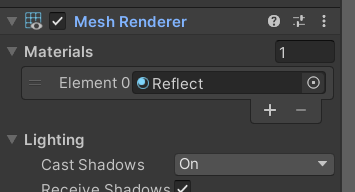
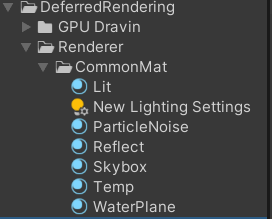
**文件结构说明**

DefferedRendering：

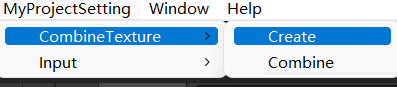
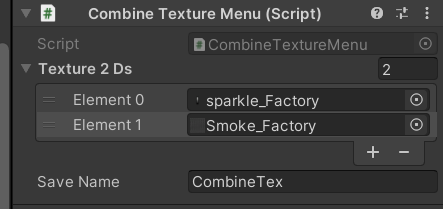
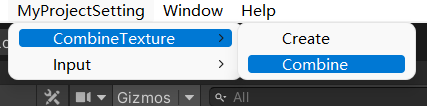
* 这个文件夹存放的是渲染管线，里面的内容不要修改，有兴趣可以看看。
* 因为这个渲染管线不再支持默认创建物体时赋予的材质，因此创建物体后找到该文件夹，选择**Lit或Reflect**替换默认材质即可。



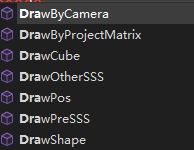
Scenes：

* 这个文件夹是用来存放所有的场景文件的，之后的所有的场景都需要放在该文件夹下。

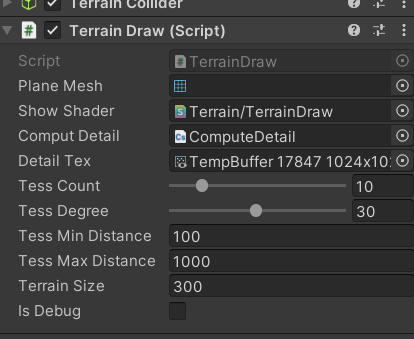
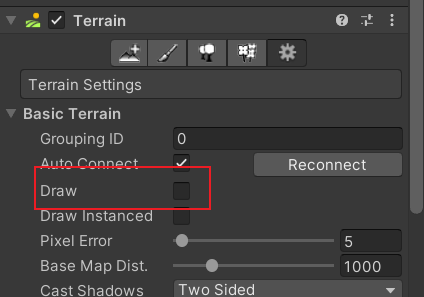
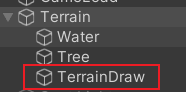
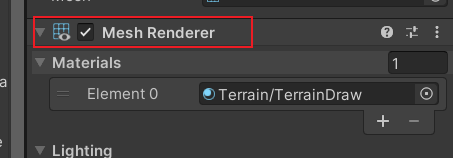
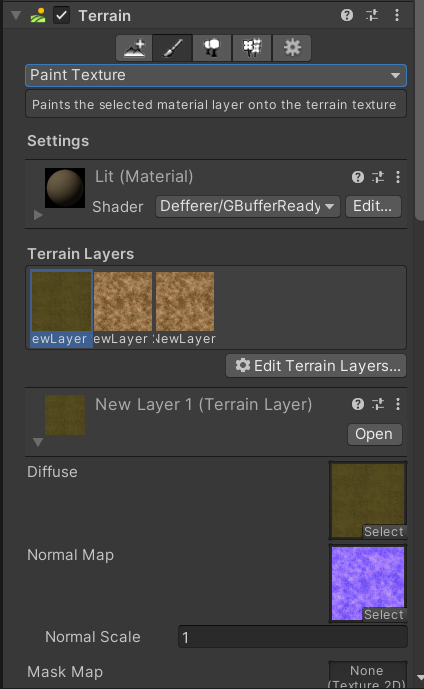
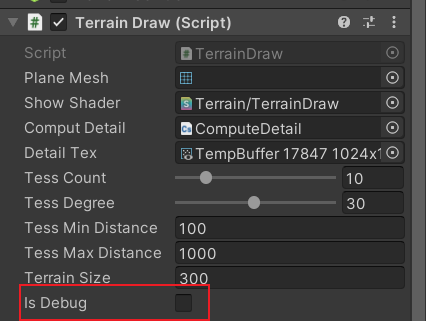
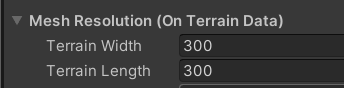
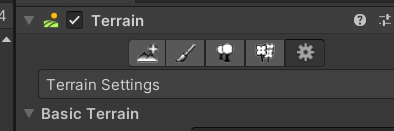
Scripts：

* 这个文件夹存储了全部的脚本文件，也是目前的框架，之后根据需求情况进行拓展。
* Common：
  + 这个文件夹存储了一些封装好的方法，用来对一些常用方法进行封装。
  + AsyncLoad是一个多线程类，通过委托将需要进行多线程处理的方法插入，依次处理。一般委托不会这么用，但是为了方便管理使用了链表的方式存储。
  + SceneObjectMap是一个场景加载类，通过标签将需要进行控制的物体用名称控制起来，之后有更好的对象控制方法可以直接换掉。
  + SustainCoroutine是协程类，用法和多线程一样，但是这个比较常用，因为对于GameObject只能在主线程中修改，协程是运行在主线程中的。这个类会在每一帧运行一次所有入栈的方法，如果这个方法放回true，就会移除这个类。这个类使用了池化技术，会比多线程快一点。
  + HandlerList：定义了一些委托类型，方便统一数据类型。
  + CommonFunction：定义了一些可能用到的方法，这个类是感觉有了SceneObjectMap就没必要了，需要查找时直接使用map映射更快。
  + FileReadAndWrite：文件读取写入类，读取字符串和写只服从到文件中
* Control： 控制文件夹，之后需要对主角、场景等进行控制的脚本都放在这个文件夹中
  + PlayControl：主角控制，目前只添加了简单的输入输出，控制主角移动等。
  + SceneChangeControl：场景切换类，先放在这里，以后用得上，但是这个类目前是直接切换，理论上是要进入到中间场景再切换是最合理的。
  + SceneObjectControl：场景物体控制类，用于存放一些节点的控制器，比如敌人、子弹、终点…
  + GameLoad：场景加载，之后的一些单例、工厂、设置等类需要优先加载时调用的类，之后可能需要定义一些加载模式，进行不同场景不同的初始化。
* Info： //存储角色数据的类
  + CharacterInfo：主角所有的根据数据都会在这个类中，以后敌人等信息也会存储在这里，以后估计会在其继承的子类上进行文件加载，毕竟要保存进度，而且不同的敌人需要不同的数据。
* Move： //运动脚本存储位置，这部分没有完善，目前就是能跑而已，可以的话最好搞成多态，让控制器只需要知道一个移动方法就能控制不同类型的物体移动，不过先这样先。
  + OribitCamera：第三人称的摄像机跟随主角脚本
  + FirstPersonCameraControl：第一人称的主角摄像机脚本
  + RigiBodyMotor：主角移动脚本
* ProjectSet：这是拓展的项目设置，以后可能进行编译器拓展时的脚本存放位置
  + MyInput：一个封装的Input类，本质上就是Unity提供的输入输出，之前想要在游戏运行时修改输入设置，没找到就写了这个类，其用法和Input类是一样的，不过将更新时间换为了FixUpdate，之后有需要拓展输入方式时可以更新该类。
* StreamingAssets：这个是一个存放文件的文件夹
  + 这个文件夹不会在Unity发表项目时打包、加密，所以可以保证在游戏发布后都能正常读取，因此以后的数值，剧情等数据都可以放在这里。
* ParticleNoiseFactory类
  + 这个类是粒子工厂类，根据传入的参数以及设置的图片可以渲染在指定位置渲染出粒子。
  + 现在提供了三种方式，一种是直接在初始位置释放，一种是在初始位置周围的一个矩形中释放，矩形的大小是cubeOffset，也就是一个Vector3代表xyz的偏移大小。球形的需要参数是radius和radian，也就是根据参数方程在一个球面上生成粒子。
  + 这个类可以指定普通的图片，也就是这个图片不需要变化，这种时候需要在我的Resources->ParticleFactory下的文件出指定一下其的行与列数，指定为1和1。之所以要这样是因为我需要确定这张图片是否本身就是一个图集，默认设置的第2张图片，这个图片本身就是一个图集，因此需要指定行数以及列数为8和8，表示总共有64个图片，渲染的粒子要一张张切换。
  + 图集图片的生成方式很简单，首先要保证图片都是一致的大小，不一致可以扔到PS中导出一张一致的图片，目前设的是256\*256，粒子没必要很精细。在点击MyProjectSetting->CombineTexture->Create：  
    会出现一个物体(或者给该物体加上组件，物体名称：CombineTextureMenu)，之后可以在其的设置面板上添加图片数组，注意图片的编号，之后传入参数的图片编号要和这个数组的编号一致，然后指定下名称就行了：
  + 继续之前的流程，点击下面的Combine，就会打包出一个图集：
* 粒子生成

在函数内引用ParticleNoiseFactory.Instance实例，内置多种渲染方法



参数内容为ParticleDrawData结构体

* 地形系统
  + 首先，在地形创建后加入脚本TerrainDraw，设置为以下参数：
  + 
  + 其中的DetailTex不用管，这个是我用来Debug用的，实时生成。
  + 在渲染时需要将Draw给关掉，不用Unity来绘制高度。
  + 不过由于我渲染时的数据都是实时生成的，因此在绘制高度时可以关闭我的渲染，在Terrain的子物体处：处的，关闭后就不会渲染了，然后开启Terrain的Draw，在这个模式下绘制高度会方便一点。
  + 绘制纹理时就需要关闭Terrain的Draw模式，改用我的渲染模型。
  + 在Terrain的Paint Texture设置完纹理后就可以使用了，但是需要额外设置以下值，我将这个值定位了我纹理的主颜色。
  + 在TerrainDraw上开启is Debug后，会实时生成笔刷上绘制的数据，但是这个损耗很高，绘制时用一下，不要在游戏发布时开启。
  + 具体贴图绘制就没什么区别了，也是设置金属度和光滑程度即可。
  + Terrain Size是地形大小，和Unity的  
      
    相对应，不过这里我强制要求为正方形而已。

# 需求汇总：

1. 机制-终点会根据玩家得分提供不同表现（是否达成通关条件）
2. 美术-到达终点时播放粒子特效
3. 机制-角色的下蹲、闪现、突进
4. 美术-更多的场景…
5. UI-显示角色的生命值与得分
6. UI-设置界面，调整玩家的fov，视角高度，鼠标灵敏度