C#与c++服务器通信需要注意的事项：char的字节个数不同c++为1个字节C#为2个字节，并且c#是大端排序，c++是根据平台来区分大端排序还是小端排序，float和long根据cpu不同所占用的精度和字节不一样

动画混合树和动画的Layers的区别，动画混合树是让同一个部位不同方向的动画进行融合，Layers是不同层级的动画进行融合（如第一层控制腿部动画，第二层控制上身动画）

文件输入输出流：StreamWriter sw = new StreamWriter(fs1);//创建文件写入流，写入，然后关闭，记得关闭

字典的排序方法，先将字典转换成一个链表List<KeyValuePair<string, List<string>>> Linelist = new List<KeyValuePair<string, List<string>>>(LineBlocks);然后利用list的Sort方法进行排序

```
Linelist.Sort(
delegate(KeyValuePair<string, List<string>> s1, KeyValuePair<string,
List<string>> s2)
{//这里决定你的排序顺序是从大大小还是从小到大，我这里的排列顺序是从小到大。如果想改变排序方式，直接对换s1和s2就行了。
return s1.Value.Count .CompareTo(s2.Value.Count);
});
```

Unity开关panel的最佳方法，添加canvasgroup，然后

将其alpha设置为0，interactable和blocksRaycasts设置为false就可以隐藏，将其alpha设置为1，interactable禁用按钮和blocksRaycasts禁用射线设置为true就可以显示

引起DC的原因，顶点和材质球发生变化

机器发烫的原因 I/O过高，减少gameobject的instantitate和destory，尽量使用设置为不活跃状态来使得会被再次显示的资源，从而减少I/O

发布到真机之后黑屏，贴图过大导致无法正常显示，如苹果4就无法识别1024*1024之上的贴图

启动的时候黑屏时间过长，Resources文件夹中的资源过大

相机的移动放在lastupdate，可以有效防止抖动，因为update之后还有个动画模拟生命周期，而lastupdate是最后才执行的，而且可以防止空帧

Lua参数溢出，c#传给lua的参数不能大于2M

Lua内存溢出，大多数情况是因为lua的table有指向于C#中的对象，c#中destory之后lua没有删除引用，最终导致lua中的table内存溢出。可以使用lua中的弱引用

Setmetatble(tablename,{__mode='k'})，使得table中的参数如果引用被删除就会自动删除这个key和其对应的value

值类型和引用类型的基类：值类型的基类是System.ValueType隐式继承于object，引用类型的基类是object

FSM有限状态机，控制人物的动画跳转逻辑，减低每个动画之间的耦合性
Awake，Start和OnEnable的执行顺序和区别，顺序Awake(OnEnable(Start，OnEnable在每次激活脚本的时候都会执行，Awake和Start只会执行一次
UI多分辨率适配方案：设置UI Scale Mode为Scale With Screen Size
Reset方法，只会在编辑器中执行才会运行，但是实机上并不会被调用
四元数和欧拉角：欧拉角表示旋转的结果，四元数表示旋转的过程。欧拉角容易出现的问题1）不易在任意方向的旋转轴插值，2）会产生万向节死锁3）旋转的次序无法确定，但是四元数是表示的是旋转的过程，使用方法是（cos0.5Q,x*sin0.5Q,y*sin0.5Q,z*sin0.5Q）如若需要沿着（0,0,1）旋转60度 则为（cos30,0,0,1*sin30）
AI自动寻路，使用NavMeshAgent ，然后将地形打城static再进行烘焙，最好将组建设置为不影响物体旋转和移动，否则容易出现鬼畜现象。用法就是将NavMeshAgent 的updateRotation和updatePosition属性设置为false
相机跟随使用的API，使用SmoothDamp这个API，从而实现相机慢慢跟随过去，防止出现相机抖动造成头晕效果。
从网上得到一张图片显示在Image下，首先将图片存入Texture2D下，然后将其转换为Sprite，使用Sprite.Create(temTure,new Rect(x,y,z,w),new vector2(x,y))转换成Sprite，然后再赋值为image的sprite属性，本地加载的时候可以直接转换成Texture
脚本执行顺序设置，通过Edit(Project Settings(Script Execution Order打开MonoManager面板，或者选择任意脚本在Inspector视图中点击Execution Order按钮，通常用在框架的起始脚本上
代码动态设置父类 GameObject.transform.SetParent(),第一个参数是Transfrom，第二个字段是设置是否已世界坐标为基准，一般使用false
GameObject.Find()是采用的深度优先遍历
RigidBody常常用来实现真实的物理现象，常用的添加力的方法，给刚体添加力，给刚体一个速度，直接对Transform进行位移变化
MVC框架可以优先降低代码之间的耦合性。M为数据层 主要负责数据交互，V为视图层主要负责界面的变化 C为控制器 主要负责M和V的同时刷新
Unity网络请求，一般使用协程，迭代返回使用yield return tmpRequest.SendWebRequest()。 Get请求会将参数存在URL中，POST请求会将数据存到一个表单中，表单类似于键值对，然后传输数据。Get请求的方法UniyWebRequest.GET(url);
POST请求的方法UnityWebRequest.Post(url, form);
C#将其他类型转换为二进制的方法：普通基本类型BitConverter.GetBytes //该方法的命名空间为System 需要引用该命名空间。。string类型
System.Text.Encoding.Default.GetBytes
射线，通常用于检测是否和地面碰撞

Camera这个api下可以将各种坐标进行转换，如屏幕坐标转换为视图坐标等，内部原理是使用矩阵的左乘

向量：向量的点乘是用来判断距离和前后，叉乘是用来判断在左边还是右边，　两个向量的法向量的计算 a X b = (l,m,n) X (o,p,q) = (mq-np,no-lq,lp-mo)

求绝对值　Mathf.ABS()　也可以使用 x = x > 0 ? x :-x;

求角度 使用Mathf.Acos()得到的值是弧度　不是角度，需要在乘以Mathf.Rad2Deg

坐标的转换是通过变换矩阵来的：旋转是4X4矩阵的左上的3X3的部分，平移是最右边的前三个，缩放是左上的3X3从左上到右下斜着的3个数。矩阵的旋转和缩放都是线性变换，平移是仿射变换

动态更换材质球：是获取Renderer组件然后使用里面的material.mainTexture属性赋值

设置UI组件的渲染顺序：使用transftom.SetSiblingIndex(Index)进行设置，实际上设置的是在父类里的位置，常用于使得点击的页面覆盖其它的页面

背包拖拽：位置变化会引起DC的增加，所以使用第三者变换图片是最省资源的

协程的关闭和开始的坑：开启协程一共有3个方法，关闭协程需要跟其一一对应，1）创建IEnumerator开启协程的话，StopCoroutine里的参数必须是IEnumerator实例,如IEnumerator x = CountTime();StartCoroutine(x);StopCoroutine(x)，2）创建Coroutine实例，StopCoroutine里的参数必须是Coroutine实例，如Coroutine x;x = StartCoroutine(CountTime());StopCoroutine(x); 3)使用字符串开关协程 StartCoroutine(“Name”); StopCoroutine(“Name”)

设置场景中的东西不会被销毁的API：DontDestroyOnLoad（gameobject）

如何读取streamingAssetsPath文件夹下的资源，PC端可以直接访问，移动端需要使用UnityWebRequest这个API来获取资源

Shader要和材质球成对出现，里面的pass为渲染通道，每出现一次就会进行一次渲染，pass里面嵌套material写材质，四种光线写在material下，Diffuse为漫反射，Specular为镜面反射，Ambient为环境光，Emission为自发光。Pass下写的lighting On开启灯光总开关，SeparateSpecular On开启镜面反射分开关。开启灯光开关之后 如果没有反射则该材质对外显示的是黑色

SetTexTure:previous前一个设置的图片，primary定点计算的结果，Texture本代码块的贴图，constant颜色常量使用constantcolor设置　可以使用加减乘和lerp

Shader中的LOD：全程Shader Level of Detail翻译为着色器的细节层次效果。在Unity3D->Project Setting->QualitySettings中的Maximum LODLevel可以设置最大LOD等级，Shader的LOD值是小于所设定的LOD值，才会被编译使用。Maximum LODLevel的等级可以设置7个级别，例如设置为1，则表示把最大LOD值设置为100，等级2，则最大LOD值为200，以此类推，若设置为0，则表示不进行LOD判断，任何LOD值的Shader都会被使用。（主要应用在多机型中的渲染，高性能机型应用高性能的shader，低性能的机型使用低性能的shader）

Shader的Tag：主要使用Queue,1）background最先渲染1000，2）geometry渲染不透明的物体，实体物体 2000，3）alphatest 要经过α测试的队列 2450，4）transparent 透明的半透明的物体3000，5）overlay 最后渲染的4000。使用的话还以在加上一个数，从而控制是在什么时候进行渲染的

Shader报错的话可以使用FallBack返回调用的shader文件

渲染流水线：顶点着色器-->光栅化-->片段着色器（openGL叫片元着色器）-->alpha测试-->模板测试-->深度测试-->blend-->gbuffer-->framebuffer-->frontbuffer。

顶点着色器的用处：计算顶点的位置和颜色。 片段着色器进行的逻辑：1）计算像素的颜色 2）跟灯光进行计算 3）跟贴图进行计算

Lerp插值公式：Lerp(a,b,t) a*t+b*(1-t)

图片的纹理寻址的方式：point是根据对应的百分比获取，bilinear是根据百分比对应的像素及附近的其他四个像素的平均值，trilinear会自动生成更多的图，然后根据符合大小的最近的两张图片所对应的像素及附近的其他四个像素共计十个像素的平均值。是设置图片的Filter Mode参数

Shader三大测试。Alpha测试根据设定的alpha值淘汰不符合条件的像素alphaTest comparison alpahvalue。深度测试ztest是将已经渲染的和即将渲染做比较，zwrite是控制是否写入深度缓冲区。Cull front | back | off 不渲染前面或者不渲染后面，off代表全都需要渲染 默认是只渲染前面。模板测试：if（referenceValue&readMask comparisonFunction stencilBufferValue&readMask） 通过像素 else 抛弃像素

渲染顺序：渲染顺序为是底层固定的，法线可以通过叉乘求得，正方面是根据顶点的顺序顺时针的方向是正面

Blend融合：基本格式Blend SrcFactor DstFactor；最常用的几种1）Blend SrcAlpha OneMinusSrcAlpha即将渲染的a 值* 即将渲染像素的RGBA + （ 1-即将渲染的像素的A值）*已经渲染像素的RGBA 2）Blend One One // Additive 1* 即将渲染像素的RGBA + 1*已经渲染像素的RGBA

C#宏指令，语法是 #if #endif 中间的代码会在符合条件的情况下执行，一般是区分android和ios的代码分别进行执行。

Shader2.0需要定义两个入口函数 #pragma vertex vert 顶点着色器 #pargma fragment frag 片段着色器

C#中获取Render下的material然后调用setfloat方法修改shader中的参数

渲染通道：U3D中的所有物体的渲染次数大于等于1次。Forward一共渲染两次，光线追踪算法；Legacy Deferred Lighting 做次时代会用到（现在被淘汰了）会渲染3次，效率最低；Pefered 一共渲染两次效果比上一个好，效率也提高了；Legacy Vertex只渲染一次，在一次渲染中结合所有的灯光在顶点着色器里面计算，效率最高效果最差。

Surface Shader #pragma surface surf Custom vertex:vert finalcolor:myColor；surface surf是定义了一个叫surf的surface表面着色器的入口函数，用来设置顶点的颜色。Vertex：vert是定义了一个叫vert的顶点着色器，custom定义的是灯光入口函数，finalcolor：myColor是定义了一个叫myColor的最后的颜色入口方法；；执行顺序是先执行顶点入口 ，然后执行surface表面着色器的入口函数，再执行custom定义的是灯光入口函数，最后执行Finalcolor:myColor入口函数。顶点着色器的参数信息

Phone着色器 ：高光 跟入射角和观察角有关 高光= Spec * 灯光的颜色 Spec=Dot（反射角，观察角）；简化版本Spec = Dot（H，N） H=入射角-观察角，N为法线；漫反射公式：跟入射角有关但是跟观察角无关 漫反射=Diffuse*灯光的颜色，Diffuse = Dot（法线，入射角）

雾，根据shader的自定义最后一次颜色加算函数中使用lerp插值，根据和相机的距离进行插值 如 color = lerp（color，_fogColor,IN.fogDensity）。使用UnityObjectToViewPos将v.vertex.xyz从世界坐标转换为带Z值的视图坐标，然后利用length来设计视图坐标长度，求得的模即为这个物体跟相机的距离

法线贴图，将法线存在贴图中，具体方法是使用 （xyz+1）*0.5，将贴图转换成法线则为（xyz-0.5）*2；因为法线z值对应的是B，而且法线大多Z值比较大所以法线贴图是蓝色的

红警一样的小地图，相机视野：思路 1）从相机的四个顶点发出一条射线，然后检测射线的碰撞 2）将碰撞的四个点获取到，然后跟大地图做一个比例计算 3）将比例尺计算到小地图上，然后根据比例尺在小地图上进行绘制

Shader边缘检测：利用法线和像素到相机的向量点乘来判断是否处于边缘，边缘是垂直

主要是应用了边缘检测，只有在接近垂直的时候赋予颜色，随着是否垂直强度越来越大

进程、线程、协程：进程是每个应用程序所开辟的内存空间每一个应用程序都是一个进程；线程一个进程包含多个线程；协程是协助主线程运行的程序

UDP和TCP的区别：TCP的特点是整个过程只寻址一次，中间不会发生丢包，而且每个包的顺序是固定的，通过数据流的形式进行传输；UDP的特点是每次发送都会寻址，但是在发送的过程中有可能发生丢包，而且其顺序会因为网络堵塞乱序，数据的传输形式是数据报

网络延迟的解决办法：一般常用方法1）影子追踪法，物体的包围盒跟在物体之后 2）减去路由时间差的方法：计算时间的时候减去路由中消耗的时间，实际上计算的是发送数据的时刻 3）现在大多数是采用等待一个固定时间收集所有的消息之后在进行计算，而不是收到一个就进行一次操作，在固定的时间内没有发送过来消息的视为没有操作

TCP沾包现象：原因是网络延迟导致一次接收了多条数据，解决思想：给数据加一个包体 分为包头和包体，包头为固定长度有body的长度，UUID等信息，根据包头的信息然后取包头中的body长度的位即可

使用UDP传输视频：采集摄像头的每一帧图片，然后将图片给Rander的mainTexture，然后利用相同点的像素颜色生成一个新的texture2D，然后扩展为EncodeToJPG转换成二进制数组进

行传输，那边接收之后在使用texture2D的loadImage将二进制转换成图片
Socket句柄和文件流句柄的区别：socket是内存到网卡的一个映射，使得内存可以获取到网卡中的数据；文件流句柄是内存到硬盘的一个映射，使得内存可以获取到硬盘的数据
背包商城之类的操作都会发生什么：获取物品ID，用户名(传入服务器(服务器计算结果(返回客户端(协程接收(传入主线程（使用委托的方式 有可能发生死锁 需要使用LOCK）(解析数据(分发模块(分发界面(更新数据并且刷新界面
Assetbundle包需要先给资源打上标记，然后再打包
矩阵变换的时候需要注意会发生矩阵的蠕变，因为精度的限制有可能 sinθ^2 + cosθ^2 !=1 从而导致大量矩阵变换的时候会出现较大的误差
装箱拆箱：装箱是将值类型转换成引用类型，拆箱是把引用类型转换成值类型
合批：静态合批就是将固定的模型之类的打成static从而减少DC，动态合批需要有同样的材质球，合批最好有很多顶点重合在一起，否则效果不是很明显
Assetbundle的注意事项：1）场景文件和普通文件不能打成一个包，2）同一个包中不能存在名字和后缀都相同的内容； 打包注意是使用BuildPipeline.BuildAssetBundles这个API 第一个参数选择路径，第二个参数大多选为0，第三个参数是选择平台一般使用EditorUserBuildSettings.activeBuildTarget自动选择当前选择的平台；
Assetbundle加载的内存：主要会使用3块内存，1）将assetbundle包加载的时候会加载到内存中；释放AssetBunlde.Unload(false) 2）将内存中的assetbundle包解压会占用第二块内存 释放gameobject Resources.unloadAssets(assets)释放飞gameobject的东西如贴图之类的Resources.unloadunusedAssets() 3）将解压过的assetbundle中的资源实例化的时候占用第三块内存 释放Destroy(GameObject)
编写Editor插件：首先需要继承Editor ，然后再方法前写上[MenuItem(“ITools/AssetBundles/Bulid AssetBundles”,false,100)];第一个参数是工具栏的路径；第二个参数如果设置为true则会在其他同名的菜单选项中先执行该函数然后在执行点击的（个人理解为所有同名函数调用的时候都会调用一次该函数，这个函数也叫做验证函数，验证函数打为true的 时候有返回值，返回值类型为bool），所以平时最好打成false；第三个参数是优先级，每50个分割
反射角的计算方法：入射角-2*法线*dot（入射角，法线）
接入SDK，在Plugins文件夹下添加Android文件夹，然后将jar包放入该文件夹下，调用方法
```
private AndroidJavaObject jo;  AndroidJavaClass jc = new AndroidJavaClass("com.unity3d.player.UnityPlayer");  jo = jc.GetStatic<AndroidJavaObject>("currentActivity");    jo.Call();
```
博客：https://blog.csdn.net/ithinking110/article/details/103239125
动态合批的思路：1）获取所有的meshRender中的sharedMaterial 2）然后获取所有的MeshFilter 再使用CombineInstance结构体获取MeshFilter中的mesh和transfrom，然后将

所有的子物体设置为不显示 ，然后给父物体创建新的Mesh然后利用CombineMeshes赋予新的网格　最后赋予Mesh新的材质

在UGui上使用粒子特效：创建一个新的正交Camera，并且该相机只渲染UI层的东西，将画布放入该相机没然后将粒子特效放入该画布，粒子特效的层级也要选为UI

UI层面的UNITY自动合批：会将渲染层级相同的且材质球相同的自动合批，此处的渲染层级是指是否发生覆盖，从0开始计算

灯光的终级公式Ambient（物体自身材质 环境光设置） * Lighting Window’s Ambient Intensity setting + (Light Color * Diffuse + Light Color * Specular) + Emission 既环境光*设置的环境光颜色+灯光颜色*漫反射+灯光颜色*高亮反射+自发光

Shader中的texcoord简单来说texcoord就是存在顶点里的一组数据，我们可以通过这组数据在渲染的时候进行贴图采样，比如我们常用的第一套uv作为基础纹理，通常基础纹理我们可以根据需求进行一些区域的uv重用（比如左右脸贴图一样，可以映射到统一贴图区域），第二套uv经常用于光照贴图，光照贴图要求是uv不可以重复，所以通常不能用第一套uv，第三套uv用于更加奇特的需求，以此类推

Lua调用方法的.和:的区别，.方法是静态方法 :方法是对象方法

Lua使用设置元表的__index元方法来实现面向对象

Lua中this和self的区别，this是自己定义的一个参数，self是调用本方法的对象

Lua 的代码执行顺序为顺序执行，如果执行报错则跳过继续 执行下面的文件

Lua多行字符串使用[[]]

程序设计模式：工厂设计模式，单例模式，建造者模式，原型模式，观察者模式等等

顶点数数量限制：手机5W 流畅3—4W； NPC一般为700 主角为2000-3000

根据TAG对比的时候不要使用gameObject.tag==“”，而应该使用 gameObject.CompareTag("XXX")，因为.tag会遍历所有的对象分配的标签

对象池技术可以避免频繁的实例化和销毁

音频上ios使用mp3格式，安卓采用vorbis格式，音频采样率手机一般8K

对小的频繁使用的声音最好打成Decompress On Load，这样就可以在加载的时候解压缩到内存，从而避免多次解压缩造成I/O过高，导致手机发烫等

打包后shader丢失：设置always include shaders里包含这个shader

每隔一段时间通过代码执行一次GC回收，减少内存堆的压力，一般设置为5s

UI优化，1）动静分离 2）降低刷新频率 3）打成一张大图，从而使得unity自动合批

Dc的数量范围 120以下 最大180 一般都在50以上