(1)场景说明

Bootup:启动场景

Login:登录场景

Maincity:主城场景

(2)加载说明

启动—热更—登录—进入检测版号—热更—登录

启动—热更—登录—进入检测版号—loading

(3)各个模块充分解耦，功能独立，拥有自己的xxxEngine，timer。即使存在代码重复也可以理解

(4)拥有独立的资源加载和卸载逻辑，不直接供给GameObject，给实例化id，通过传入实例化id，进行操作对象

(5)shader可以先进行预加载，shader加载很快，但解析很慢，所以建议预加载，减少Keyword，去除fallback

(6)android 纹理格式etc1,etc2，对于带有透明通道的纹理，拆分成一张rgb通道，一张alpha通道贴图，去除mipmap

(7)ios 纹理格式pvrtc，去除mipmap

(8) 特效区分，具体场景特效，ui特效等，合并贴图，共用材质，减少dc<15, 允许把一张小图重复打在不同的大贴图内（也可以一个特效贴图尽量合并为一张贴图256\*256），shader中的tint color不用，依据机型做不同等级特效，面数控制<500，单个粒子吐量控制<30，同屏<200，animator可以用animation代替，关闭远处特效，或者lod,使用billboard模式，shader使用移动端，并经ta验证，渲染层级3300-3350，地裂渲染层级2501，特效所需的模型mesh render不必要的参数全部勾掉，纹理格式etc1,关mipmap，粒子大小<1m，命名规范，减少overdraw，做延时发射，分帧发射，看不见的子特效及时结束生命，依据运行时fps，动态关闭一些细节特效

(9)模型，网格在保证视觉效果的前提下，本着“够用就好”原则，依据项目镜头需求，降低网格资源的定点数和面皮数量，定点属性越多，内存占用越高，加载时间也更长，对定点属性的使用需谨慎，对于静态的网格可以关闭网格的read/write功能，大幅降低内存占用，角色面数<3000,骨骼<30,npc等不重要角色面数<500,骨骼<15,建筑初级<1500,中级<3000,高级<4500

(10)cpu性能分为两大类，自身代码和引擎模块，引擎模块主要细分为渲染，动画，物理，ui，粒子，加载和gc调用

(11)mono堆内存只增不减

(12)UI不可见组件少用，虽然不可见，但依然占据了显卡资源，降填充率(指显卡每帧能够渲染的像素数)，在每帧绘制中，一个像素被反复绘制的次数越多，占用的资源必然也会多，ui贴图使用polygon mode打图集，减少贴图留白，留白会有更多的重复渲染，尽可能静态和动态ui分离，放置在不同的canvas下，静态ui canvas下控制元素数量，一旦有元素发生变化，则会引起重新mesh，造成开销，防止aba图集叠层的问题，尽量控制mask组件的使用，不仅提高绘制的开销，同时会造成dc明显上升

(13)优秀插件，mobile move texture,性能非常高兴，仅支持ogv,ogg格式；dynamic bone,增加角色运动的真实代入感，比如头发，衣服的跟随；magic splitscreen,分屏视角；realistic effects pack,实现背景扭曲的效果，非常高效；see-through system,实现透视，线性绘制和红外线等效果；mesh baker,网格和材质合并，降dc；simpleLOD,减点减面；屏幕特效(摄像机后处理)fxpro:bloom&dof,mobile-ready和beautify；bloompro

(14)换装参考：https://github.com/zouchunyi/UnityAvater

(15)项目概述，帧率控制30+，主城面数<10w，dc<120，战斗面数<5w，dc<80，远景用lod，物理模块rigidbody控制30左右，少用碰撞（尤其ui的相互叠加），动画建议mecanim系统，mono内存控制在40m，优秀则需控制在25m，总内存200m,纹理图片etc1格式，透明通道分离，mesh资源控制在20m,贴图资源50m，控制实例化对象，使用缓存池，ui的开关使用移出入相机视角，粒子依据机型加载，及时关闭远景粒子

(16) 项目开始时，书写shader文档，项目中用到的shader一一记录

(17)视角较远，可以把贴图分辨率降低，过滤器模式改为点模式(不会渐变渲染，呈像素块状)提高亮度，线性模式为像素平滑过渡