性能优化分为 渲染优化和内存优化

渲染优化主要是减少DrawCall，DrawCall是cpu像gpu发送到渲染指令，需要cpu收集渲染信息。

场景之中主要是采用合批技术减少DrawCall，静态合批还有动态合批。

静态合批通过勾选Inspector面板中的static使用，是收集材质相同而且transform信息不会发生改变的物体的网格信息，合并成一个大的网格，储存起来，由CPU统一发送，然后减少了DrawCall的发送次数，但是也是有限制的，单词合批最多是64000左右的顶点数，所以需要合批的物体网格处于激活状态，而且网格也要开启。缺点是运行时占用内存增大，因为需要开辟额外的内存来储存合并的mesh网格信息。

然后是动态合批，是收集相同材质物体的mesh信息，合并起来，由cpu统一发送，动态合批的限制是 必须材质相同，单个网格最多支持225个顶点数，需要注意的是延迟渲染不支持动态合批，还有就是材质使用的shader的代码中不能有多个pase。

两个合批方式都会有一个合并网格的操作，会使性能开销变大，但是这个消耗是可以接受的。

还有一种是GPU Instancine，这个也是unity提供的一种方案，它的本质也是使用一个DrawCall渲染多个相同的物体，比如场景中的 花草树木等。GPU Instancine和动静合批的区别是 他不会去合并网格信息

然后ui优化这一部分，因为Canvas是他的最基本的渲染单位，主要是为了解决这个Canvas重绘问题，采用了一个Canvas分层的方案，因为 如果一个Canvas的ui发生变化只会该Canvas重新渲染，不会影响其他的Canvas，Canvas分层主要有，背景层，面板层，tips层，3D交互层还有新手引导层，然后就是按照使用频率区分面板，频率高比如聊天面板，小地图等，频率低比如充值，设置面板等，区分面板放在不同的Canvas层。

避免频繁删除/增加UI对象，UI层次发生变化会引起，Canvas的重绘，避免UI元素数目过多和层次结构过于复杂影响Batch更新速度。

然后还可以采用合批技术，ui的合批主要是材质相同，如果精灵图不同，也会打断合批，所以要打图集，使用同一图集，就能进行合批，尽量不要使用Mask（内部使用了末班缓冲，至少会增加两个DrawCall）