**MonoBehavior生存周期**

Awake→OnEnable→Start [→FixedUpdate→yield WaitForFixedUpdate] →Update→yield null/WaitForSeconds→LateUpdate→yield WaitForEndOfFrame→OnDisable→OnDestroy

生命周期函数通过反射调用，有一定开销，尽量避免继承MonoBehavior。

**3C**

Camera、Character、Control。

**Unity资源管理**

**Resources**

将Asset资源放在一个或多个名为Resources的文件夹中，并且在运行时利用相关API来加载或卸载。Resources索引信息用二叉平衡树来存储，其时间复杂度为O(nlog(n))，当资源太多时花费的build时间会长很多。

**AssetBundle**

资源打包，在游戏过程中加载，可用来减小包体，做资源更新

**Adressable Asset System**

基于AssetBundle，囊括了资源管理（资源依赖，自动计算引用）

**Canvas渲染模式**

Screen Space-overlay : 将UI渲染到屏幕空间，是所有物体的最前面，可设置sortingOrder。

Screen Space-camera : 将UI渲染到视距空间，可用于渲染3d模型或者粒子特效。

World Space : 将UI作为3D对象，放在世界坐标里，通常用来作为人物名称或血条等UI。

**UGUI屏幕适配**

Canvas Scaler设置UI Scale Mode为Scale With Screen Size。

设置Match：

手机端竖屏为0（宽度）：720\*1280，一般更高，即宽高比更小。

电脑端/手机端横屏为1（高度）：1920\*1080，一般更宽，即宽高比更大。

**锚点Anchor/轴点Pivot**

锚点（Anchor）用于判断UI相对于父物体的位置，由四个点组成。当四个点重合时，用轴点（Pivot）到锚点的水平距离和垂直距离来定位，用宽度和高度来表示范围；如果某一方向上锚点不重合，则会根据到两锚点的距离来判断位置和范围（left、right、top、bottom）。

**Image和Raw Image**

Image只支持Sprite类型的图片，Raw Image支持任何一种贴图模式，利用的是其Texture。Image有Image Type，可进行Simple、Sliced、Tiled、Filled四种设置；Raw Image有UV Rect，可以简单地裁剪图片。

**Unity特殊文件夹**

Editor：脚本只在编辑器环境下使用（可放子目录）

Gizmos：存放在场景中绘制的辅助标记文件。

Plugins：插件文件夹，放第三方的库插件。

Resources：采用Resources加载资源时将资源放在该文件夹下。（可放子目录）

StreamingAssets：采用AssetBundle加载资源时资源包位置。

AA：采用Addressable加载资源时资源包的位置。

**Unity实现截图**

Unity自带截图API：Application.CaptureScreenshot。

使用Texture2D的ReadPixels实现。

对于指定相机可以使用RenderTexture。

**状态机与行为树**

**状态机**：包括状态集、事件/条件（实现状态迁移的条件）、动作（条件满足时进行的动作）、转换（进行状态迁移）。简单的可以通过枚举+switch case/事件来实现，复杂的可以通过状态模式（多态和虚函数实现）。

**行为树**：由行为节点组成的树状结构，叶子节点都是可执行的行为。

Sequence：序列节点按从左到右的顺序执行其子节点，当其中一个子节点执行失败时序列节点也将停止运行；

Selector：选择器节点按从左到右的顺序执行其子节点，当其中一个子节点执行成功时选择器节点将停止执行；

Parallel：并行节点执行所有孩子节点，直到至少M个孩子节点返回成功状态或所有孩子节点返回失败状态；

Decorator：装饰器节点可以以自定义形式修改孩子节点行为（Invert反转、Repeat重复等）。

状态机每次只能存在一个状态，行为树可以通过并行实现；状态机对添加状态不太友好，可能需要更改大量状态转换，而行为树容易更改。

**Unity性能优化**

1. 程序/Gameplay部分：

减少继承MonoBehaviour，一方面设计模式方面解耦引擎，一方面脚本生命周期函数通过反射调用具有性能开销。

减少update的调用，改用协程或事件驱动。

减少对大规模对象的查询，例如FindObjectsOfType等，性能较低，改用其他方式获取。

避免频繁创建和销毁物体，改用对象池。

减少GC，利用缓存方式对调用堆的库函数结果进行缓存。避免拆箱装箱。

2. 美术资源/渲染部分：

减少模型复杂度，减少顶点数量，使用**LOD**（层次细节），根据相机距离优化模型顶点数。

**静态合批**：将static的静态物体合并网格，减少batch送往GPU处理，会消耗额外内存。

**动态合批**：如果动态物体共用着相同的材质，那么Unity会自动进行批处理，进行顶点组合，CPU有进行顶点转换的开销。

使用**遮挡剔除**，提前剔除看不到的模型，减少绘制模型数量。（增加CPU压力和内存消耗）

对于UI使用**图集**（Sprite Atlas），从而减少Draw Call。

利用**烘焙**技术，用烘焙光代替实时光照。

3. GPU部分：

Shader进行简化，避免使用if（GPU中会对所有分支进行处理）。

使用mipmap进行纹理优化。

光照多时可以考虑使用延迟渲染。

**Draw Call、Batch和SetPass Call**

DrawCall：一次图形渲染接口的调用。

Batch：一个渲染批次，设置渲染状态，并且调用一次或多次DrawCall。

SetPass Call：Shader中Pass被切换的次数，每个渲染批次都会设置一个Pass，Pass可能不变。

**正面剔除和背面剔除**

正面剔除：应用程序阶段，距离剔除、视椎体剔除、遮挡剔除（在CPU中为大遮挡体构造低分辨率的Z-buffer，查询其他物体是否被遮挡）。

背面剔除：在光栅化阶段，可对一个模型背面的面进行剔除，通过计算法线和视线方向的夹角实现。

**Shader中的uniform、attribute、varying**

uniform：应用程序（CPU）、vertex shader、fragment shader所共享的全局变量， 逻辑语言传递给着色器， CPU可读写，GPU只读。

attribute：只能在（应用程序和）vertex shader中使用的变量。

varying：在vertex shader和fragment shader之间传递的变量。

**Builtin和URP**

Builtin：内置渲染管线。

URP：通用渲染管线，属于可编程渲染管线。相比于Builtin性能更好，易用性高，高度可定制化，持续维护。

**Unity和UE的区别**

Unity提供的工具基于社区，UE提供的功能比较全，不易魔改，难以优化。

UE提供了Gameplay框架：GameInstance、GameMode(PlayerController/GameState等)、UObject/(Actor/Component)/Pawn/Character、SaveGame等。

Unity默认前向渲染，UE默认延迟渲染。

**柏林噪声**

随机生成排列表和梯度表，对某一点，找出其参考点（一维为两个，二维为四个），根据参考点的方向和梯度进行插值得到最终的值。

**ECS/DOTS**

面向数据编程，Entity：实体、Component：组件/数据、System：系统（处理状态切换的逻辑）。

优点：快，基于数据，缓存的cache hit频率比较高；能够摆脱面向对象中的一些耦合的问题。

缺点：要解决内存管理和编码规范的问题。