**游戏引擎构架分析理解报告**

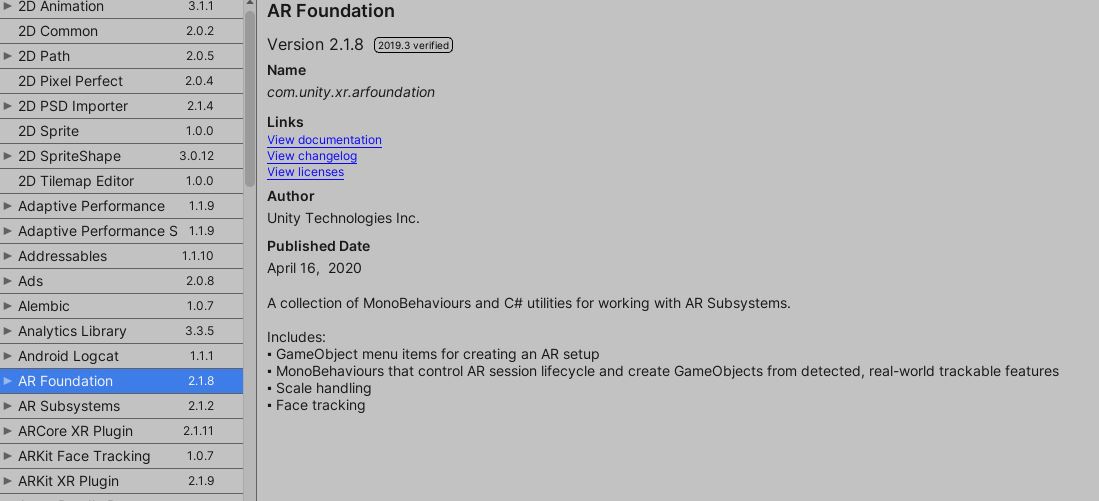
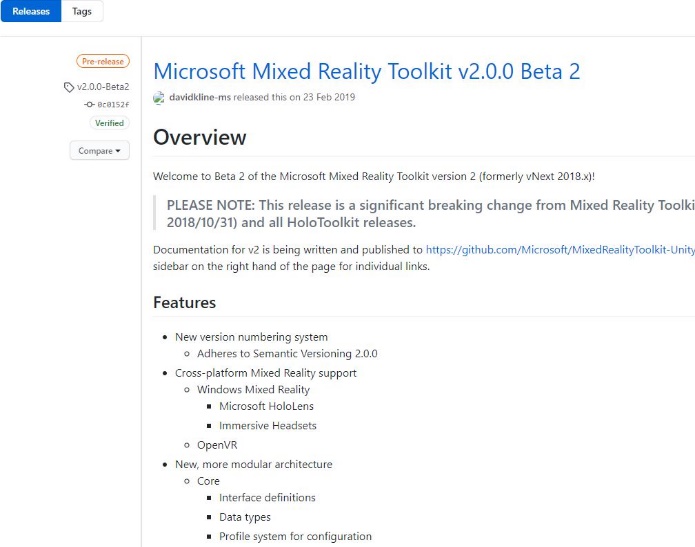
从我自己理解的说起，先是游戏引擎运行时架构，主要分为游戏专用子系统；主要引擎模块：渲染，动画，物理，网格，音效等；核心系统，资源管理系统；平台独立层，第三方软件开发工具及承包；系统、驱动、硬件层面。

**系统、驱动、硬件层面：**

目标硬件层，是将运行游戏的电脑系统或者主机。硬件驱动是比较底层的由操作系统或者硬件开发商提供的软件组件。硬件驱动管理硬件资源，把操作系统与引擎上层跟无数不同的可用的硬件细节操作隔离开来。在PC上，操作系统是一直运行着的。它指挥着多个程序在一台电脑上运行，其中就包括你的游戏。这意味着PC游戏不能假设拥有硬件的所有控制权。

在游戏主机上，操作系统通常只是一个轻量级的库，链接到游戏的执行档里。在游戏主机上，游戏一般完整的“拥有”整个硬件。

**第三方软件开发包及中间件：**

包括很多比较底层的图形显卡驱动，例如openGl，DX等，这些第三方中间件工具包会提供一些基础的API，例如图形绘制的API，物理基础碰撞交叠检测的函数等等。还有一些也属于第三方SDK的比如ARCORE，或者是Vuforia，VRTK，MRTK之类的可以很好和一些其他硬件相互对接 交互的SDK，我比较常用Arcore和Vuforia与MRTK，Arcore可以快捷的对接实现安卓端的AR功能，例如平面识别，环境光估计，人脸识别，图像识别等等，但是现在有了ARFoundtion之后就不太会直接用Arcore了，因为ARFoundtion直接提供的全面的解决方案，内部的API帮我们完成了大部分算法识别层面的种种计算，我们只需要专心做上层的应用逻辑的编写就可以。

还有例如MRTK是我毕业设计开发HoloLens应用时候用到的一个能快速开发HoloLens的一个SDK，里面提供很多范例和场景供我参考，也提供了主要的交互的脚本，例GestureRecognizer,GestureSettings,InteractionSourceKind等，可以根据他的示例和API快速实现手势识别，语音识别，空间扫描等功能。

再比如leapmotion也是我本科时候做项目时候用到的手势识别的设备，在官网上也可以快速的下载到各个模块来给开发提供便利。

**平台独立层：**

平台独立层包括文件系统，网络传输，常用标准库，线程库等等，我记得unity有自己的network模块，比较方便做局域网，但是我在一个TCP服务器的项目中没有用到，因为network底层应该也是封装的socket类的一些方法，所以就直接用socket写了，多线程我记得unity在一起支持的不是很好，例如不支持让多线程来直接操作游戏物体等。

2019后又更新了一些数学库，用来做一些程序化生成的算法很好用，在以前managerpackage里还没有abc动画的支持模块，都要搞一些插件来用，18.3更新后就有了这些sdk，很多时候提升了开发效率，避免了重复造轮子。

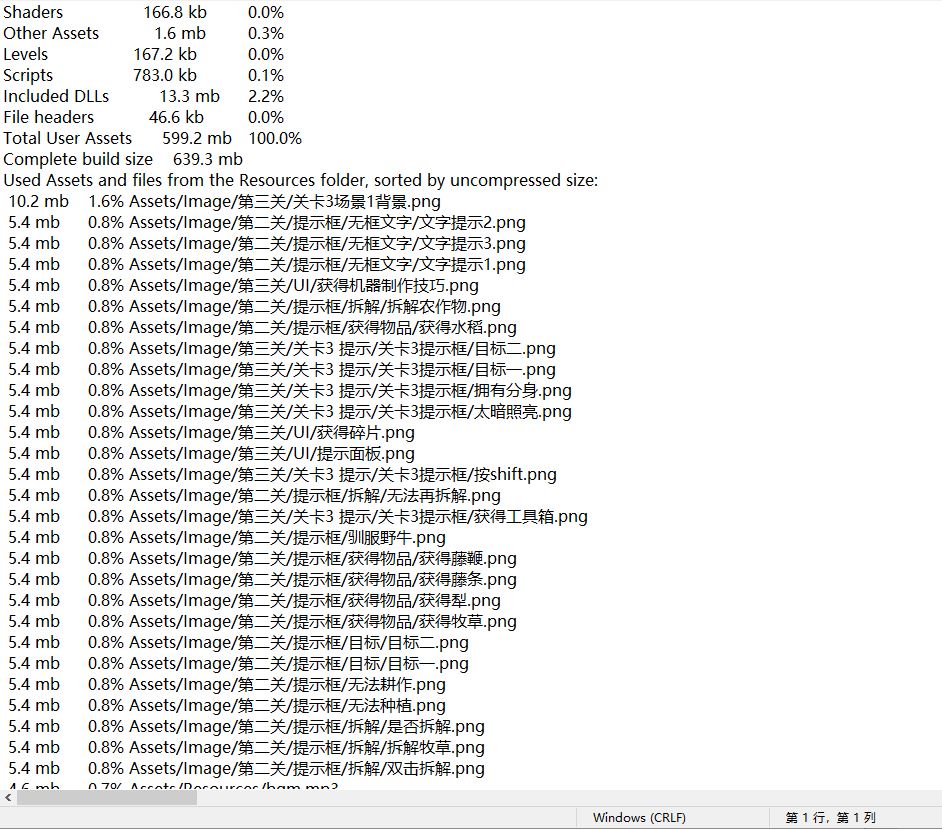
**核心系统：**

例如数学库：矢量、矩阵、四元数、三角数学、样条、数值积分，断言、调试打印及日志，反射与序列化。例如在做服务器时候解析到客户端来的消息后就可以根据字符串的函数名来调用真正的函数，序列化我的理解就是把一些public的变量的复制存储在文件中，但是很容易丢失，所以最好是动态的指定比较好。

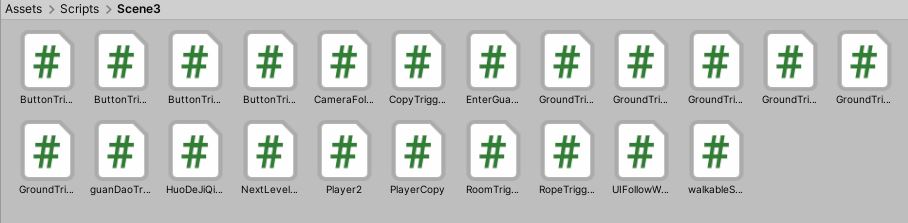
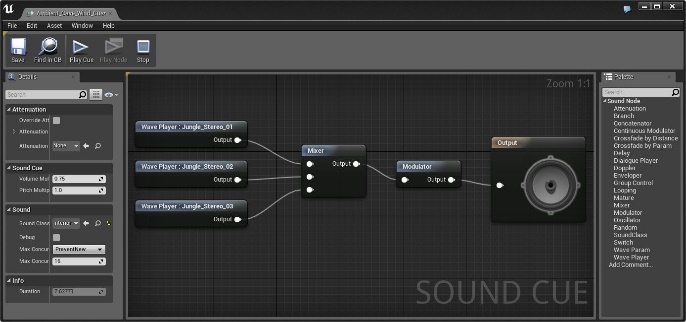
像类似数学库，mathf里就比较全面，还有特殊的Quaternion类里的插值，平滑插值，lookat之类的API也是开发时候经常用到的。

日志的输出我一般只有在项目打包的时候会仔细观看，当打包完成，我会根据编辑器的log来看到底这次打包所有的资源都占了多大的空间，然后有针对性的去优化和调整。

或者是frameDubug，可以输出整个渲染画面的过程，或者是unity的性能分析工具profile等等。



**资源管理：**

管理游戏资源与生命周期，例如模型资源，纹理资源，材质资源，字体资源，动画资源，参数与配置。每个引擎都有某种形式的资源管理器，它提供一个统一的接口(或者几个接口)来存取任何及所有的游戏资源及其它引擎输入数据。例如一个unity项目中不可避免的要包括很多的图片，模型，音效，动画控制器，预制体，脚本，插件，材质等等的资源，上图是我最近在做的项目的图片和脚本资源截图，都需要分门别类的把它们放在对应的文件夹中，有时候还需要将一些资源打包成AssetsBundle来整合压缩资源，unity给我们的就是对这些多种类型的资源的支持和调整，例如在ue4中我可以直接的编辑模型的碰撞体，或者是调整音频的混响，或者是对图像进行饱和度，明度，srgb等参数的调整。

**主要引擎模块：**

**首先是渲染引擎：**

分层架构，在引擎使用时候这些东西大部分都是设置好的编写好的，只能进行参数的设置调整，但是有些模块也可以自己进行改动，例入屏幕的后处理效果这些都可以通过OnRenderImage(RenderTexture src, RenderTexture dest)来通过回调函数会把当前渲染好的屏幕画面传输到src参数里，然后经过我们处理再设置会dest里，unity再把dest输出到屏幕上，把scr的图像传给材质，这个函数会Graphics.Blit(src, dest, postMaterial)材质会利用自己的shader来计算后处理效果，然后返回到dest中。还有一些渲染相关的算法也可以写在可编程渲染管线里，也就是shader中，在shader中可以自定义光照模型，或者是直接对顶面和片元进行操作和剪裁，以此来达到各种高级材质的效果，例如3s材质，毛发材质等等。

对于想粒子模块或者贴花模块之类的虽然不能很自定义的去修改里面的代码，但是可以利用computershader和脚本直接自己手写一套粒子系统。

UI等界面显示unity的UGUI一些基本的小项目还是够用的，以前会有一些NGUI或者什么的，或者是根据项目需求自己写一些UI管理的框架，对视频的支持unity会提供一些组件来实现视频的简单播放，但是当打包h5或者一些其他平台的时候可能会不支持。

**渲染管线：**

Unity的渲染管线大致是支持前项渲染，延迟渲染，或者两者结合的，现在也支持一些高清管线HDRP等。Unity目前支持Fixed function shader 属于固定渲染管线Shader, 基本用于高级Shader在老显卡无法显示时的备用Shader。

Vertex and Fragment Shader属于可编程渲染管线。使用的是CG/HLSL语言。

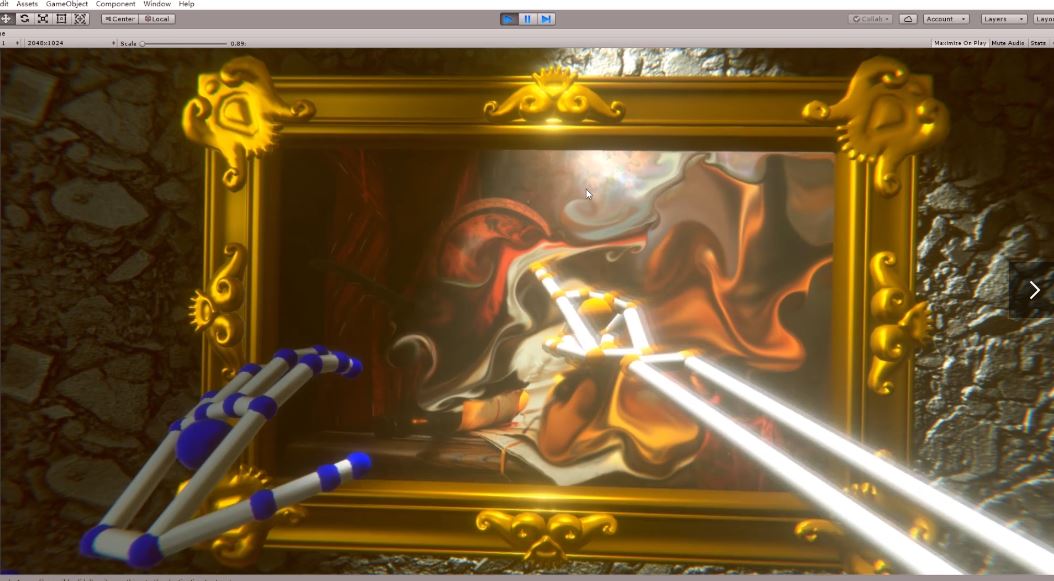
Surface ShaderUnity3d推荐的Shader类型。它是一个代码生成器，帮我们将重复的代码省去了，使得编写Shader更为容易。使用的也是CG/HLSL语言。Compute Shader则是是微软DirectX 11 API新加入的特性，在Compute Shader的帮助下，程序员可直接将GPU作为并行处理器加以利用，GPU将不仅具有3D渲染能力，也具有其他的运算能力。

**碰撞和物理：**

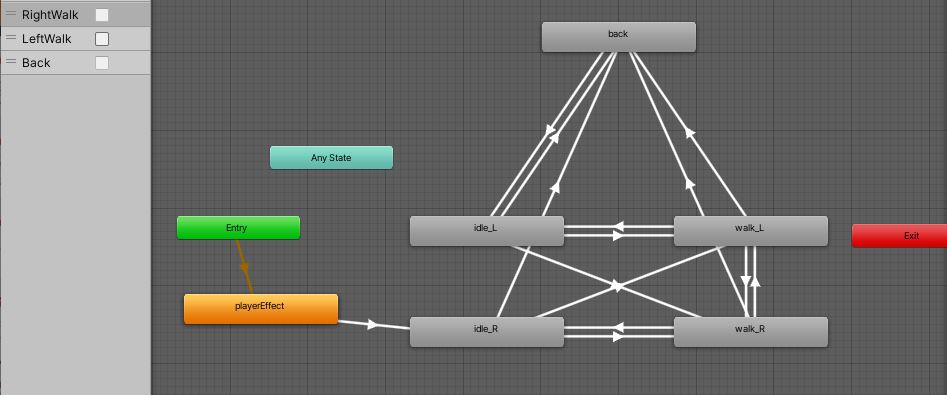
这个模块unity应该是PhysX，有刚体组件，各种类型的碰撞体，但是不如ue4做的那么完善，可以直接对模型凸包分解，或者是根据建模时候特定名称的模型作为碰撞体。还有一些较为复杂的动力学例如流体模拟，布料模拟，烟雾模拟等，这些可能都需要其他的一些插件或者自己手写一些模拟算法。

例如下图是我假期在家利用NS方程模拟的二维平面的流体效果，里面加入了leapmotion作为互动。

碰撞检测对每个游戏来说都是很重要的。没有它，物体之间就会相互渗透，不可能跟虚拟的3D世界进行任何合理的交互。有的游戏包含真实或半真实的动力学模拟。



**动画系统：**

动画系统也是很常用的模块，例如精灵/纹理动画，刚体层次结构动画，顶点动画，骨骼动画，状态树及层，IK，混合，播放，动画压缩，变形目标动画等。

例如这是我的一个项目里的动画状态机，只是简单的控制了一个2d角色的左右移动之类，混合树一般用在动画的混合中，例如直走和拐弯的动画需要一个融合，ik，fx是正向和反向动力学，在maya或者3dmax里做动画这个是个很常见的概念，用于动画骨骼的控制，骨骼就相当于变形器，影响周围的点来运动，而ikfk就是相当于控制骨骼动画的方式，骨骼动画让动画师使用相对简单的骨头系统去设定精细三维角色网格的姿势。骨骼网络渲染组件是连接的渲染器跟动画系统的桥梁。动画系统为骨骼中的每一根骨头计算出一个位置，然后将位置作为矩阵系列(palette of matrices)之一传递给渲染引擎。渲染器对每个顶点用矩阵系统中的一个或多个矩阵进行坐标变换，作出一个最终的混合的顶点位置。这个过程被称之为蒙皮(skinning)。

至于点动画或者是一些通过脚本和shader来控制的动画一般就用不了动画状态机了，例如在shader里对顶点做偏移，或者在c#里获取顶点数组进行操作完后再设置回去，再或者是houdini中烘焙出来的点级别动画，需要特定的脚本来读取动画信息。

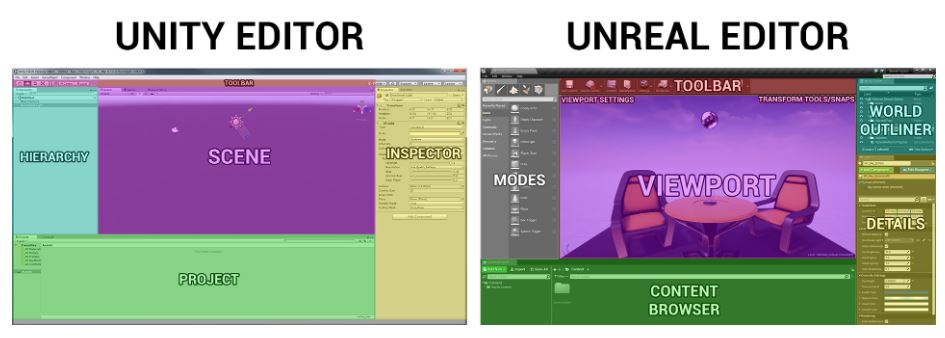
人体接口设备：

一般来说包括不限于鼠标和键盘，专用控制器和传感器，触摸屏，游戏手柄等。例如leapmtion，kinect，HoloLens，HTC VR等。

**多人在线/网络：**

多人在线网络游戏一般会涉及到tcp，udp或者是类似photo之类的第三方服务，一般也会涉及帧同步和状态同步。我在开发的时候选择用unity作为客户端，直接用vsc#的socket来编写服务器逻辑。

**总结：**

总的来说，引擎的构架包括很多方面，unity和ue4作为出色的游戏引擎受到了市场的认可，他们之间有很多相似和不同之处。

例如源文件在 Unity 中，大家习惯将 C# 的源文件放在资产目录中。

虚幻 4 的工作机制有点不同。假如项目包含 C++ 代码，项目目录中有一个 Source 子目录，其中包含各种文件，包括 C++ 源文件（.cpp）和头文件（.h），以及一些编译用的脚本（.Build.cs，.Target.cs）。

在虚幻 4 中使用 C++ 的最简单做法就是通过编辑器菜单 为项目添加代码

在 Unity 中， 基本上所有的东西都是GameObject ，可以放置在地图中的 "东西"。虚幻 4 中对应的概念是 Actor。在虚幻编辑器中，可以从放置面板直接拖一个空的 Actor 放置到场景中。

在 Unity 中，可以通过为 GameObject 添加组件来赋予其特定的功能。

在虚幻 4 中，可以为 Actor 添加组件。在关卡中放置一个空 Actor 后，点击"添加组件"按钮。

我个人用unity会多一些在平时，虽然我是美术出身，但是写了2年代码，shader，算法之后，一般的要求不高的独立游戏也可以独立开发，但是我也学习过ue4的蓝图和c++，感觉ue还是会重型一些，不像unity那么小巧，一切都是脚本和组件来的那么直接，开发相对比较难上手。