# Unity面试题：

1. 协程（协同程序）

在主线程运行的同时开启另一段逻辑处理，来协助当前程序的执行，协程很像多线程，但又不是多线程，unity的协程实在每帧结束之后去检测yield的条件是否满足。

1. 碰撞器：

Collider碰撞器。当is Trigger =false时，碰撞器根据物理引擎引发碰撞。当 is Trigger=true时，碰撞器变为触发器。其中碰撞器调用的函数为OnCollisionEnter/Stay/exit;触发器调用的函数为OnTriggerEnter。

1. 接上一个补充：两个物体要产生碰撞。两个物体要带有Collider，其中一个物体还带有Rigidbody刚体，好像必须是运动的物体带有刚体才能。有个名词叫做刚体休眠。处于休眠状态中的物体，不会再对其进行碰撞检测和模拟。
2. 值类型与引用类型

每一个值类型都有一个独立的内存区域保存自己的值，调用它的时候调用的是他的值，而引用类型调用的是内存中的地址。进一步研究文档，你会发现所有的结构都是抽象类型System.ValueType的直接派生类，而System.ValueType本身又是直接从System.Object派生的。根据定义 所知，所有的值类型都必须从System.ValueType派生，所有的枚举都从System.Enum抽象类派生，而后者又从System.ValueType派生。

所有的值类型都是隐式密封的（sealed），目的是防止其他任何类型从值类型进行派生。

在c#中所有的类都是引用类型，包括接口。

区别：

值类型通常被人们称为轻量级的类型，因为在大多数情况下，值类型的的实例都分配在线程栈中，因此它不受垃圾回收的控制，缓解了托管堆中的压力，减少了应用程序的垃圾回收的次数，提高性能。

所有的引用类型的实例都分配在托管堆上，c#中new操作符会返回一个内存地址指向当前的对象。所以当你在创建个一个引用类型实例的时候，你必须要考虑以下问题：

内存是在托管堆上分配的

在分配每一个对象时都会包含一些额外的成员（类型对象指针，同步块索引），这些成员必须初始化

对象中的其他字节总是设为零

在分配对象时，可能会进行一次垃圾回收操作（如果托管堆上的内存不够分配一次对象时）

值类型比引用类型更容易存取。

性能：

在设计一个应用程序时，如果都是应用类型，那么应用程序的性能将显著下降，因为这会加大托管堆的压力，增加垃圾回收的次数。

虽然值类型是一个轻量级的类型，但是如果大量的使用值类型的话，也会有损应用程序的性能（例如下面要讲的装箱和拆箱操作，传递实例较大的值类型，或者返回较大的值类型实例）。

由于值类型实例的值是自己本身，而引用类型的实例的值是一个引用，所以如果将一个值类型的变量赋值给另一个值类型的变量，会执行一次逐字段的复制，将引用类型的变量赋值给另一个引用类型的变量时，只需要复制内存地址，所以在对大对象进行赋值时要避免使用值类型。

1. 值类型的装箱与拆箱

装箱就是将值类型转化为引用类型。在托管堆中分配好内存，分配的内存量是值类型的各个字段需要的内存量加上托管堆上所有对象的两个额外成语（类型对象指针，同步快索引）需要的内存量。

拆箱就是将以装箱的引用类型实例（由值类型转换来的）转化为子类型的过程。

1. 请简述GC（垃圾回收）产生的原因，并描述如何避免？

答：GC回收堆上的内存

避免：

1）减少new产生对象的次数

2）使用公用的对象（静态成员）

3）将String换为StringBuilder

要想了解垃圾回收如何工作以及何时被触发，我们首先需要了解unity的内存管理机制。Unity主要采用自动内存管理的机制，开发时在代码中不需要详细地告诉unity如何进行内存管理，unity内部自身会进行内存管理。

unity的自动内存管理可以理解为以下几个部分：

unity内部有两个内存管理池：堆内存和堆栈内存。堆栈内存(stack)主要用来存储较小的和短暂的数据片段，堆内存(heap)主要用来存储较大的和存储时间较长的数据片段。

unity中的变量只会在堆栈或者堆内存上进行内存分配。

只要变量处于激活状态，则其占用的内存会被标记为使用状态，则该部分的内存处于被分配的状态，变量要么存储在堆栈内存上，要么处于堆内存上。

一旦变量不再激活，则其所占用的内存不再需要，该部分内存可以被回收到内存池中被再次使用，这样的操作就是内存回收。处于堆栈上的内存回收及其快速，处于堆上的内存并不是及时回收的，其对应的内存依然会被标记为使用状态。

垃圾回收主要是指堆上的内存分配和回收，unity中会定时对堆内存进行GC操作。

在了解了GC的过程后，下面详细了解堆内存和堆栈内存的分配和回收机制的差别。

#### LOD(Level of detail)多层次细节，是最常用的游戏优化技术。它按照模型的位置和重要程度决定物体渲染的资源分配，降低非重要物体的面数和细节度，从而获得高效率的渲染运算。缺点是增加了内存。

### **向量的点乘、叉乘以及归一化的意义？**

#### 1.点乘描述了两个向量的相似程度，结果越大两向量越相似，还可表示投影

#### 2.叉乘得到的向量垂直于原来的两个向量

#### 3.标准化向量：用在只关系方向，不关心大小的时候

### **什么是LightMap？**

#### LightMap:就是指在三维软件里实现打好光，然后渲染把场景各表面的光照输出到贴图上，最后又通过引擎贴到场景上，这样就使物体有了光照的感觉

### **ref参数和out参数是什么？有什么区别？**

#### ref和out参数的效果一样，都是通过关键字找到定义在主函数里面的变量的内存地址，并通过方法体内的语法改变它的大小。不同点就是输出参数必须对参数进行初始化。ref必须初始化，out 参数必须在函数里赋值。ref参数是引用，out参数为输出参数。

### **移动相机动作在哪个函数里，为什么在这个函数里？**

#### LateUpdate，是在所有的update结束后才调用，比较适合用于命令脚本的执行。官网上例子是摄像机的跟随，都是所有的update操作完才进行摄像机的跟进，不然就有可能出现摄像机已经推进了，但是视角里还未有角色的空帧出现。

### **什么是渲染管道？**

#### 是指在显示器上为了显示出图像而经过的一系列必要操作。 渲染管道中的很多步骤，都要将几何物体从一个坐标系中变换到另一个坐标系中去。主要步骤有：

#### 本地坐标->视图坐标->背面裁剪->光照->裁剪->投影->视图变换->光栅化

### **请描述游戏动画有哪几种，以及其原理？**

#### 主要有关节动画、骨骼动画、单一网格模型动画(关键帧动画)。

#### 关节动画：把角色分成若干独立部分，一个部分对应一个网格模型，部分的动画连接成一个整体的动画，角色比较灵活，Quake2中使用这种动画；

#### 骨骼动画，广泛应用的动画方式，集成了以上两个方式的优点，骨骼按角色特点组成一定的层次结构，有关节相连，可做相对运动，皮肤作为单一网格蒙在骨骼之外，决定角色的外观；

#### 单一网格模型动画由一个完整的网格模型构成，在动画序列的关键帧里记录各个顶点的原位置及其改变量，然后插值运算实现动画效果，角色动画较真实。

### **写出光照计算中的diffuse的计算公式**

#### diffuse = Kd x colorLight x max(N\*L,0)；Kd 漫反射系数、colorLight 光的颜色、N 单位法线向量、L 由点指向光源的单位向量、其中N与L点乘，如果结果小于等于0，则漫反射为0。

### **下列代码在运行中会发生什么问题？如何避免？**

List<int> ls = new List<int>(new int[] { 1, 2, 3, 4, 5 });

foreach (int item in ls)

{

Console.WriteLine(item \* item);

ls.Remove(item);

}

#### 产生运行时错误，在 ls.Remove(item)这行，因为foreach是只读的。不能一边遍历一边修改。

### **Unity3D是否支持写成多线程程序？如果支持的话需要注意什么？**

#### 仅能从主线程中访问Unity3D的组件，对象和Unity3D系统调用

#### 支持：如果同时你要处理很多事情或者与Unity的对象互动小可以用thread,否则使用coroutine。

#### 注意：C#中有lock这个关键字,以确保只有一个线程可以在特定时间内访问特定的对象

### **Unity3D的协程和C#线程之间的区别是什么？**

#### 多线程程序同时运行多个线程 ，而在任一指定时刻只有一个协程在运行，并且这个正在运行的协同程序只在必要时才被挂起。除主线程之外的线程无法访问Unity3D的对象、组件、方法。

#### Unity3d没有多线程的概念，不过unity也给我们提供了StartCoroutine（协同程序）和LoadLevelAsync（异步加载关卡）后台加载场景的方法。 StartCoroutine为什么叫协同程序呢，所谓协同，就是当你在StartCoroutine的函数体里处理一段代码时，利用yield语句等待执行结果，这期间不影响主程序的继续执行，可以协同工作。

#### 存取修饰符：

#### public：存取不受限制。

#### private：只有包含该成员的类可以存取。

#### internal：只有当前工程可以存取。

#### Interface:接口，1、接口类似于抽象基类，不能直接实例化接口；接口中的方法都是抽象方法，实现接口的任何非抽象类型都必须实现接口的所有成员。2、接口不能包含常量、字段、运算符、实例构造函数、析构函数或类型、不能包含静态成员。3、接口成员是自动公开的，且不能包含任何访问修饰符。4、接口自身可从多个接口继承，类和结构可继承多个接口，但接口不能继承类。

#### protected：只有包含该成员的类以及派生类可以存取。

#### 类修饰符：

#### abstract：抽象类。指示一个类只能作为其它类的基类。

#### sealed：密封类。指示一个类不能被继承。理所当然，密封类不能同时又是抽象类，因为抽象总是希望被继承的。

#### 成员修饰符：

#### abstract：指示该方法或属性没有实现。

#### sealed：密封方法。可以防止在派生类中对该方法的override（重载）。不是类的每个成员方法都可以作为密封方法密封方法，必须对基类的虚方法进行重载，提供具体的实现方法。所以，在方法的声明中，sealed修饰符总是和override修饰符同时使用。

#### delegate：委托。用来定义一个函数指针。C#中的事件驱动是基于delegate + event的。

#### const：指定该成员的值只读不允许修改。

#### event：声明一个事件。

#### extern：指示方法在外部实现。

#### override：重写。对由基类继承成员的新实现。

#### readonly：指示一个域只能在声明时以及相同类的内部被赋值。

#### static：指示一个成员属于类型本身，而不是属于特定的对象。即在定义后可不经实例化，就可使用。

#### virtual：指示一个方法或存取器的实现可以在继承类中被覆盖。

#### new：在派生类中隐藏指定的基类成员，从而实现重写的功能。 若要隐藏继承类的成员，请使用相同名称在派生类中声明该成员，并用 new 修饰符修饰它。