# Unity脚本基础

## Unity3D中的协程（coroutine）和C#线程之间的区别是什么？

（1）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。

（2）除主线程之外的线程无法访问 Unity3D 的对象、组件、方法。

（3）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。

（4）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。而 LoadLevelAsync 则允许你在后台加载新资源和场景，所以再利用协同，你就可以前台用 loading 条或动画提示玩家游戏未卡死，同时后台协同处理加载的事宜

## 简述 Unity3D 支持的作为脚本的语言的名称

Unity 的脚本语言基于 Mono 的.Net 平台上运行，可以使用.NET 库，这也为 XML、数据库、正则表达式等问题提供了很好的解决方案。

Unity 里的脚本都会经过编译，他们的运行速度也很快。这三种语言实际上的功能和运行速度是一样的，区别主要体现在语言特性上。

JavaScript：和网页中常用的 JavaScript 不一样，它编译后的运行速度很快，语法方面也会有不少区别。

C#

Boo：可以看做是 Python 语言的变种，又糅合了 Ruby 和 C#的特性，它是静态类型语言

## 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

## OnEnable、Awake、Start 运行时的发生顺序？哪些可能在同一个对象周期中反复的发生？

Awake -》OnEnable-》Start

OnEnable 在同一周期中可以反复地发生

## 请简述 OnBecameVisible 及 OnBecameInvisible 的发生时机，以及这一对回调函数的意义？

当物体是否可见切换之时。可以用于只需要在物体可见时才进行的计算。

## Unity3D 如何获知场景中需要加载的数据？

Resource.Load

AssetBundle.Load

## 物体发生碰撞时，有几个阶段，分别对应的函数

三个阶段，OnCollisionEnter/Stay/Exit 函数

## u3d 中，几种施加力的方式，描述出来

rigidbody.AddForce/AddForceAtPosition，都是 rigidbody 的成员函数

## 物体自旋转使用的函数叫什么

transform.Rotate

## 物体绕某点旋转使用函数叫什么

transform.RotateAround

## u3d 提供了一个用于保存读取数据的类，（playerPrefs），请列出保存读取整形数据的函数

PlayerPrefs.SetInt 与 PlayerPrefs.GetInt

## unity3d 从唤醒到销毁有一段生命周期，请列出系统自己调用的几个重要方法。

Awake –>OnEnable –> Start –> Update –> FixedUpdate –> LateUpdate –>OnGUI –> Reset –> OnDisable –> OnDestroy

## 物理更新一般在哪个系统函数里？

FixedUpdate，每固定帧绘制时执行一次，和 update 不同的是 FixedUpdate 是渲染帧执行，如果你的渲染效率低下的时候 FixedUpdate 调用次数就会跟着下降。FixedUpdate 比较适用于物理引擎的计算，因为是跟每帧渲染有关。Update 就比较适合做控制。

## 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 update 结束后才调，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，都是在所有 update 操作完才跟进摄像机，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。

## 为什么 u3d 会出现组件上数据丢失的情况

一般是组件上绑定的物体对象被删除了

## 什么是协同程序？

在主线程运行时同时开启另一段逻辑处理，来协助当前程序的执行。换句话说，开启协程就是开启一个线程。可以用来控制运动、序列以及对象的行为。

## 反向旋转动画的方法是什么？

反转动画，讲动画的速度调到-1，碰撞时，被碰撞物体与碰撞物体有collider 组件，碰撞物体有刚体组件，或角色碰撞得包含角色组件 OR 改变 animation.speed

## 用代码实现第三角色控制器

public class Player : MonoBehaviour {

public Transform \_cameraTrans;

private Vector3 \_cameraOffset;

void Awake() {

\_cameraOffset = transform.position - \_cameraTrans.position;

}

void Update() {

\_cameraTrans.position = transform.position - \_cameraOffset;

}

}

## 实现吊机吊物体的功能

吊机吊物体需要节点挂接和坐标系转换

## 获取、增加、删除组件的命令分别是什么

获取：GetComponent

增加：AddComponent

删除：Destroy

## Animation.CrossFade 命令作用是：C

1. 动画放大 B.动画转换 C.动画的淡入为其他动画

## Application.loadLevel 命令为：A

1. 加载关卡 B.异步加载关卡 C.加载动作

## 调试记录到控制台的命令是什么？

Debug.Log();

## 编辑器类存放路径是什么？

工程目录下的 Assets/Editor 文件夹下。

## 使用原生 GUI 创建一个可以拖动的窗口命令是什么？

GUI.DragWindow();

## localPosition 与 Position 的使用区别？

localPosition：自身位置，相对于父级的变换的位置。 Position：在世界坐标 transform的位置

## 含义Mathf.Round, Mathf.Clamp, Mathf.Lerp

Mathf.Round 四舍五入

Mathf.Clamp 限制

Mathf.Lerp 插值

## 写一个计时器工具，从整点开始计时，格式为：00:00:00

private float timer = 0f;

private int h = 0;

private int m = 0;

private int s = 0;

private string timeStr = string.Empty;

void Update () {

timer += Time.deltaTime;

if (timer >= 1f) {s++; timer = 0;}

if (s >= 60) {m++;s = 0;}

if (m >= 60) {h++;m = 0;}

if (h >= 99) {h = 0;}

}

void OnGUI(){

timeStr = string.Format ("{0:D2}:{1:D2}:{2:D2}", h, m, s);

GUI.Label (new Rect (10, 10, 100, 200), timeStr);

}

## 写出 Animation 的五个方法

AddClip 添加剪辑、Blend 混合、Play 播放、Stop 停止、Sample 采样

## 用鼠标实现在场景中拖动物体，用鼠标滚轮实现缩放(用一个 Cube 即可)。

在场景中添加一个Plan，Camera，Directional Light，Cube。添加两个脚本scrollerScirpt（挂在Camera），CubeDragScript（挂在Cube上）。

1.鼠标滚轮实现缩放：将摄像机的镜头拉近或者拉远，调整摄像机的视角就可以实现，主要实现代码如下：

void Update () {

//鼠标滚轮的效果

if (Input.GetAxis("Mouse ScrollWheel") < 0) {

if (Camera.main.fieldOfView <= 100)

Camera.main.fieldOfView += 2;

if (Camera.main.orthographicSize <= 20)

Camera.main.orthographicSize += 0.5F;

}

//Zoom in

if (Input.GetAxis("Mouse ScrollWheel") > 0) {

if (Camera.main.fieldOfView > 2)

Camera.main.fieldOfView -= 2;

if (Camera.main.orthographicSize >= 1)

Camera.main.orthographicSize -= 0.5F;

}

}

2.鼠标实现在场景中拖动物体：

　　解决思路就是将世界坐标转换成屏幕坐标，然后计算物体与鼠标之间移动量，循环鼠标被按下操作，得到鼠标的当前位置，加上计算好的移动量，将新的坐标赋值给物理就行了。主要是开启一个协同程序（Corountine）来处理

主要代码如下：

// Use this for initialization

void Start () {

StartCoroutine(OnMouseDown());

}

IEnumerator OnMouseDown() {

//将物体由世界坐标系转换为屏幕坐标系

Vector3 screenSpace = Camera.main.WorldToScreenPoint(transform.position);

//完成两个步骤 1.由于鼠标的坐标系是2维，需要转换成3维的世界坐标系

//2.只有3维坐标情况下才能来计算鼠标位置与物理的距离，offset即是距离

//将鼠标屏幕坐标转为三维坐标，再算出物体位置与鼠标之间的距离

Vector3 offset = transform.position - Camera.main.ScreenToWorldPoint(new Vector3(Input.mousePosition.x, Input.mousePosition.y, screenSpace.z));

while (Input.GetMouseButton(0)) {

//得到现在鼠标的2维坐标系位置

Vector3 curScreenSpace = new Vector3(Input.mousePosition.x, Input.mousePosition.y, screenSpace.z);

//将当前鼠标的2维位置转换成3维位置，再加上鼠标的移动量

Vector3 curPosition = Camera.main.ScreenToWorldPoint(curScreenSpace) + offset;

//curPosition就是物体应该的移动向量赋给transform的position属性

transform.position = curPosition;

yield return new WaitForFixedUpdate(); //这个很重要，循环执行

}

}

## NGUI Button 怎样接受用户点击并调用函数,具体方法名称是什么

1、主要是在UICamera脚本中用射线判断点击的物体并通过SendMessage调用OnClick() OnPress()等函数，可以说NGUI的按钮是通过发消息这个方式调用的。具体方法名称是OnClick()

2、void Awake () {

//获取需要监听的按钮对象

GameObject button = GameObject.Find("UI Root/Button3");

//设置这个按钮的监听，指向本类的ButtonClick方法中。

UIEventListener.Get(button).onClick = OnButton3Click;

}

private void OnButton3Click(GameObject button) {

Debug.Log("我是按钮3被点击了");

}

## <愤怒的小鸟>给予初速度以后,怎么让小鸟受到重力和空气阻力的影响而绘制抛物线轨迹,说出具体的计算方法.

Vector3 v 代表初速度 v'代表现在的速度， 假设小鸟是沿的 z 轴也就是 transform.forward方向运动的质量为 1，那么 v'=v-new Vector3(0,g\*t,f\*t)，transform.Translate(v')做的就是抛物线运动（g 为重力加速度不要用现实中的需要自己调试，f 为阻力也要自己调试设置，t 为时间）

## 当游戏中需要频繁创建一个物体时，我们需要怎样做能够节省内存？

1、使用预制物体对象 Prefab

2、使用对象池技术，不使用时关闭，使用时打开

## 碰撞检测需要物体具备什么属性？

能检测碰撞发生的方式有两种，一种是利用碰撞器，另一种则是利用触发器

【Physics.OverlapSphere 相交球检测碰撞，碰撞检测需要包围盒】

## Vector3.forward与Vector3(0,0,1)是一样的意思对吗？

对

## 下哪个函数在游戏进入新场景后会被马上调用？

MonoBehaviour.OnLevelWasLoaded

## itween 插件的作用是什么，itween 作用于世界坐标还是局部坐标，请列举出 3 个其常用方法？

iTween 是一个动画库，作者创建它的目的就是最小的投入实现最大的产出。让你做开发更轻松，用它可以轻松实现各种动画，晃动，旋转，移动，褪色，上色，控制音频等等“方法：

a、MoveTo 物体移动；

b、ColorTo：随着时间改变对象的颜色组；

c、LookTo：随时间旋转物体让其脸部朝向所提供的 Vector3 或 Transform 位置；

## U3D 中用于记录节点空间几何信息的组件名称，及其父类名称

Transform 父类是 Component

## NGUI:把子控件放在父控件里面，如何上下边距都是10。

给子控件设置上下两个锚点为10

## 如何使子控件居中,如果使用UGUI怎么实现

设置子控件锚点为中心

## 去掉敏感字的程序（手写程序）

String s = "你是坏蛋";

s.Replace("坏蛋", "\*\*");

## unity3D从唤醒到销毁有一段生命周期，列出系统自己调用的重要方法。

Awake ()

OnEnable ()

Start()

FixedUpdate()

OnTriggerXXX(Collider other)

OnCollisionXXX (Collision collisionInfo)

Update()

LateUpdate ()

OnGUI()

OnDisable ()

OnDestroy ()

## Unity3d中resources目录一般用来放些什么，打包的时候会有什么影响？

resource一般用来放置一些需要动态加载的资源，打包程序的时候会将Resource目录下的全部文件都加密压缩打进包内，这样再想使用assetbundle方式打小包的话就不行了

## 空间内一物体绕球面固定点（0、1、2）按照固定速度speed环绕运动。

public float Speed = 1;

void Update () {

transform.RotateAround (new Vector3(0,1,2),Vector3.up,Speed);

}

## 以下关于WWW.LoadFromCacheOrDownload描述正确的是： C

A．可被用于将Text Assets自动缓存到本地磁盘

B．可被用于将Resource自动缓存到本地磁盘

C．可被用于将Assets Bundles自动缓存到本地磁盘

D．可被用于将任意的Unity资源文件自动缓存到本地磁盘

## 以下哪个函数在游戏进入新场景后会被马上调用？ C

A：MonoBehaviour.OnSceneWasLoaded

B：MonoBehaviour.OnSceneEnter

C：MonoBehaviour.OnLevelWasLoaded

D：MonoBehaviour.OnLevelEnter

## 关于MonoBehavior.LateUpdate函数描述错误的是： B

A．当MonoBehavior类型应用后，每帧调用一次

B．常被用于处理RigidBody的更新

C．在所有Update函数执行后才能被调用

D．常被用于实现跟随相机效果，且目标物体的位置已经在Update函数中被更新

## 某个GameObject有一个名为MyScript的脚本，该脚本中有一个名为DoSomething的函数，则如何在该GameObject的另一个脚本中国调用该函数？ A

A．GetComponent<MyScript>().DoSomething()

B．GetComponent<Script>("MyScript").DoSomething()

C．GetComponent< MyScript >().Call("DoSomething")

D．GetComponent<Script>("MyScript"). Call("DoSomething")

## 启用MipMaps对内存的影响是？ A

A．增加约33%的内存

B．减少约33%的内存

C．增加约25%的内存

D．减少约25%的内存

## 以下关于MonoBehaviour.OnGUI()的描述的是： D

A．如果MonoBehaviour没有被启用，则OnGUI函数不会被调用

B．用于绘制和处理GUI events

C．每帧可能会被绘制多次，每次对应于一个GUI event

D．每帧被调用一次

## 采用Input.mousePosition来获取鼠标在屏幕上的位置，以下表述正确的是： B

A．左上角为原点(0, 0)，右下角为(Screen.Width, Screen.Height)

B．左下角为原点(0, 0)，右上角为(Screen.Height, Screen.Width)

C．左下角为原点(0, 0)，右上角为(Screen.Width, Screen.Height)

D．左上角为原点(0, 0)，右下角为(Screen.Height, Screen.Height)

## 正确排列出下面Unity脚本自带的函数执行顺序Start，Awake，Update，OnUpdate，OnEnable，FixedUpdate，OnGUI，LateUpdate、OnDisable、OnDestory

Awake-->OnEnable-->Start-->Update-->FixedUpdate-->LateUpdate-->OnGUI-->OnDisable -->OnDestory

## Unity中销毁GameObject的方式，简述Destroy和DestroyImmediate的区别

Destroy销毁场景中的物体，但是内存中它还是存在的，只有当内存不够时，机制才会将它销毁并释放内存。而DestroyImmediate会立即将对象销毁，并且将内存释放。

## Unity中如何派发事件（消息）

在脚本里的Update函数里调用EventDispatcher.Instance().OnTick();就可以了

## ScriptObject的作用和使用方式

ScriptObject类型经常使用于存储一些Unity本身不可以打包的一些object，比如字符串，一些类对象，用这个类型的子类型可以用BuildPipeline打包成assetbundle包共后续使用，非常方便。

## 如何检测物体是否被其他对象遮挡

使用射线进行检测

## 写一个角色控制器，鼠标控制屏幕晃动，鼠标控制开枪。

public class Player : MonoBehaviour {

public GameObject \_prefabBullet;

private float \_angleSpeed = 120f;

void Update() {

float eularY = Input.GetAxis("Mouse X") \* \_angleSpeed \* Time.deltaTime;

transform.Rotate(new Vector3(0, eularY, 0));

if (Input.GetMouseButtonDown(0)) {

Instantiate(\_prefabBullet, transform.position, transform.rotation);

}

}

}

## 敌人AI，有各种状态，实现各种状态之间的切换。

通过状态机来实现各种状态之间的切换

## 1.写一个角色控制器，鼠标控制屏幕晃动，鼠标控制开枪。

**2.敌人AI，有各种状态，实现各种状态之间的切换。**

**3.敌人会和主角对抗，敌人被打到之后，会闪一次红色，然后红色比例提升10%，10次攻击之后，成红色。**

**4.敌人会自动攻击主角，主角也会有颜色变化。**

**5.敌人会在范围内巡逻。**

**6.UI，左边显示8个AI的被攻击次数，右边显示AI的攻击次数排序。**

public class Player : MonoBehaviour {

public Camera \_camera;

public GameObject \_prefabBullet;

private float \_angleSpeed = 120f;

//生命值

private int \_life = 10;

//玩家的状态

private bool \_state = false;

//是否被打到

public void IsState() {

if (\_state && \_life > 0) {

\_life -= 1;

\_state = false;

} else {

\_state = true;

}

}

public void RoleRotate() {

float eularY = Input.GetAxis("Mouse X") \* \_angleSpeed \* Time.deltaTime;

}

//风发射子弹

public void RoleShoot() {

if (Input.GetMouseButtonDown(0)) {

Instantiate(\_prefabBullet, transform.position, transform.rotation);

}

}

}

public class Enemy : MonoBehaviour {

//敌人被打到状态

private bool \_state = false;

//玩家与敌人距离

private float \_distance;

//角色

public GameObject \_role;

//生命值

private int \_life = 10;

//敌人被攻击次数

private int \_timeAccack = 0;

//敌人状态

public void EnemyState() {

if (\_distance >= 10f) {

if (\_life >= 1 && \_state == true) {

\_life -= 1;

\_timeAccack++;

}

\_state = false;

}

else if (\_distance >= 0 && \_state == false) {

\_state = true;

}

}

//敌人与玩家距离

public void Distance() {

\_distance = Vector3.Distance(transform.position, \_role.transform.position);

}

void OnGUI() {

GUI.TextArea(new Rect(20, 50, 80, 30), \_timeAccack.ToString());

GUI.TextArea(new Rect(20, 90, 80, 30), \_timeAccack.ToString());

GUI.TextArea(new Rect(20, 130, 80, 30), \_timeAccack.ToString());

}

}

## 3D空间有三个cube当做点，有一条鱼的模型，要求在三点之间游动，要求转向平滑一点，控制鱼的运动朝向（用四元数和欧拉角）

使用transform.localRotation = Quaternion.Slerp(Quaternion a,Quaternion b,float c)实现物体平滑转向

# 服务器数据库等杂项

## Unity 连接数据库

需要得到 Mono.Data.Sqlite.dll 文件与 System.Data.dll 文件

## 如何与服务器交互

做游戏，基本上都避免不了与服务器端交互，与服务器端交互的方式也有几种，总结起来就是长连接模式（Socket）与短链接模式（Http）。

## 如何处理网络异常下的可玩性

1）为游戏增加单机模式：比如故事模块，网络异常时可以阅读游戏的故事；丰富的技能或卡牌，网络异常时可以了解技能和卡牌；提供单机玩法，玩家可以与AI进行游戏等。

2）为游戏提供教程模块，网络异常时可以学习游戏技巧。

## 怎样反外挂？ 对外挂的看法

游戏外挂的原理：外挂分为多种，比如模拟键盘的，鼠标的；修改数据包的；修改本地内存的。

1）对于模拟用户的鼠标键盘输入的外挂，我们可以用网页上常用的验证码的方式来对付。模拟键盘鼠标的外挂对游戏的影响比加速、修改封包、修改内存、脱机等要小得多，因此被一些人称为绿色外挂。

2）让服务器不把在正常情况下玩家看不到的东西的数据传送给客户端。

3）把玩家操作记录发到服务器进行模拟，如果和客户端的计算结果偏差较大可以认为作弊。

## 你对跨平台的了解。

跨平台就是在一个熟悉的平台上面开发的软件或者程序，直接可以在其他平台上正常的运行显示而不需要对其原始文件或者原始代码进行修改。

## 谈谈你们公司的网络编程

在我们公司基本是由服务端人员开发的，我这边只调用他们提供的客户端SDK和服务器端通信。服务器端是公司用C++自行研发的。

## 会做数据库吗？有嵌套的数据库能做么？

会。用过关系型数据库mysql。能做，只是这方面之前在公司没怎么负责这块，有些生疏，稍微熟悉一下就能做。

## 了解代码管理么？在开发中使用什么代码管理工具？使用的接口多一点还是继承多一点？

了解。使用SVN进行代码管理。接口多一点。

## 公司内部信息传递用的什么安全协议？

1、公司自定义的安全协议 或者 基于SSL/TLS协议 或者SSH协议

2、我们至少调用基于相关协议的SDK

3、SSL/TLS 协议

SSL/TLS 协议（RFC2246 RFC4346）处于 TCP/IP 协议与各种应用层协议之间，为数据通讯提供安全支持。

从协议内部的功能层面上来看，SSL/TLS 协议可分为两层：

1. SSL/TLS 记录协议（SSL/TLS Record Protocol），它建立在可靠的传输层协议（如 TCP）之上，为上层协议提供数据封装、压缩、加密等基本功能。

2. SSL/TLS 握手协议（SSL/TLS Handshake Protocol），它建立在 SSL/TLS 记录协议之上，用于在实际的数据传输开始前，通讯双方进行身份认证、协商加密算法、交换加密密钥等初始化协商功能。

SSH是英文Secure Shell的简写形式。通过使用SSH，你可以把所有传输的数据进行加密，这样”中间人”这种攻击方式就不可能实现了，而且也能够防止DNS欺骗和IP欺骗。使用SSH，还有一个额外的好处就是传输的数据是经过压缩的，所以可以加快传输的速度。SSH有很多功能，它既可以代替Telnet，又可以为FTP、Pop、甚至为PPP提供一个安全的”通道”。

## 原公司服务器怎么实现多人在线？

使用socket基于TCP协议开发的多人在线。公司有自行开发的服务端程序

## 项目问题，和美工想法冲突了怎么办？

找项目主管协调处理

## 对unity的理解

可以做VR、手游、工艺仿真、端游、页游跨平台的游戏引擎

## 面试题：VR相关问题用了哪些设备？接入了哪些SDK？根据项目问如何实现手势控制？

htc vive 和oculus， htc vive用到的是steam vr 免费插件，Oculus早期有SDK，现在Unity原生支持

## 开过什么平台？多线程的理解

PC、Android。

多线程是指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程，进而提升整体处理性能。

使用线程可以把占据时间长的程序中的任务放到后台去处理

·用户界面可以更加吸引人，这样比如用户点击了一个按钮去触发某些事件的处理，可以弹出一个进度条来显示处理的进度

·程序的运行速度可能加快

·在一些等待的任务实现上如用户输入、文件读写和网络收发数据等，线程就比较有用了。在这种情况下可以释放一些珍贵的资源如内存占用等等。

## 怎样实现植物的弯曲

1、max做骨骼动画

2、修改Tree组件的Growth Angle的值

## 用过什么游戏框架？

SimpleFramework,基于ulua的框架，支持ugui和ngui的热更新

## 模型导入Unity后怎么与数据库产生联系？类似打通程序和数据库通道

例如mysql数据库，需要导入两个文件，就可以通过sql语句往数据库写数据，数据可以是模型信息

## 用过什么传输协议，用什么语言

http，tcp(socket) 都是C# 语言编写

## 都在什么平台上开发过游戏或软件用的什么框架

Android

## 会写框架么

设计模式了解一些，但是没写过框架

## 请简述在客户端开发方面 你最擅长的模块系统，以及你觉得哪些模块会出现的问题

擅长UI模块的制作，我认为热更新模块会出现问题，比如资源依赖不合理，导致资源包过大

## 如何实现背包系统、宠物系统、装备系统的实时更新？

NGUI做界面，使用ulua做热更新。之前公司使用的是SimpleFramework这样的框架。

## 热更新的解决方案（资源以及代码）

ulua框架、simpleFramework框架 、bundle manager插件

# 数据结构算法相关

## 对数据结构了解吗？说说你常用的数据结构。

了解，数据结构是每个程序员都要会一点的。链表、列表、散列表最常用，队列和栈也经常使用。二叉树偶尔使用。

## 什么是状态机，什么是行为树

有限状态机系统：是指在不同阶段会呈现出不同的运行状态的系统，这些状态是有限的、不重叠的。这样的系统在某一时刻一定会处于其所有状态中的一个状态，此时它接收一部分允许的输入，产生一部分可能的响应，并且迁移到一部分可能的状态。

1. 基本节点是状态：他包含了一系列运行在该状态的行为以及离开这个状态的条件。

2. 状态可以任意跳转,实现简单,但是对于大的状态机很难维护.状态逻辑的重用性低.

3. 每一个状态的逻辑会随着一些新状态的增加而越来越复杂。维持状态的数量和状态逻辑复杂性是一个很大的难点。需要合理的分割以及重用状态。

4. 状态机状态的复用性很差，一旦一些因素变化导致这个环境发生变化。你只能新增一个状态，并且给这个新状态添加连接他以及其他状态的跳转逻辑。

5. 状态机的跳转条件一旦不满足，就会一直卡在某一个状态。

行为树：一个流行的AI技术，涵盖了层次状态机，事件调度，事件计划，行为等一系列技术。

1. 高度模块化状态，去掉状态中的跳转逻辑，使得状态变成一个“行为”。

2. “行为”和”行为”之间的跳转是通过父节点的类型来决定的。比如并行处理两个行为，在状态机里面无法同时处理两个状态。

3. 通过增加控制节点的类型，可以达到复用行为的目的。

4. 可视化编辑。

## 对List的理解

有序的对象列表，属于数据结构的一种：顺序结构

泛型集合类，引入System.Collections.Generic命名空间，

常用操作有，Count属性查看长度，Add()添加，Remove()去除，AddRange()添加集合，Clear()清空集合。

## 数组和List的核心区别

数组在C#中最早出现的。在内存中是连续存储的，所以它的索引速度非常快，而且赋值与修改元素也很简单。

数组存在一些不足的地方。在数组的两个数据间插入数据是很麻烦的，而且在声明数组的时候必须指定数组的长度，数组的长度过长，会造成内存浪费，过段会造成数据溢出的错误。如果在声明数组时我们不清楚数组的长度，就会变得很麻烦。

List<T>是集合，集合元素的数量可以动态变化。增加、插入、删除元素很方便。

## 数据结构的看法

具有一定关系的数据元素集合，好的数据结构有利于简化算法的编写

## 深度优化会么

会一点，比如在代码消耗上经常使用StopWatch类去优化算法，通常用IDisposable去显式的释放资源。

## 实现二分要什么条件

有序结构

## c++，socket网络编程会不会

C++上学的时候学过，工作之后一直没用过。

Socket就是一套实现双向通信的API。使用过C#的TCP（面向连接、可靠）和UDP连接（面向非连接、不可靠）

## 二叉树的所有遍历方式的原理及优缺点

前序遍历，先访问根节点在访问左节点在访问右节点。

中序遍历，先访问左节点在访问根节点在访问右节点。

后序遍历，先访问左节点在访问右节点在访问根节点。

前中后代表的是访问根节点的时序。

这一点上没有什么本质上的优缺点，要看实际需求决定采用何种遍历方式

采用递归方式和非递归方式。前者优点是直观，编写起来简单，缺点是但其开销也比较大。非递归形式开销小，但编写复杂。

## 数据结构中数组和链表各有什么特点，什么场合下应该使用数组，什么场合下应该使用链表

二者都属于一种数据结构

从逻辑结构来看

1. 数组必须事先定义固定的长度（元素个数），不能适应数据动态地增减的情况。当数据增加时，可能超出原先定义的元素个数；当数据减少时，造成内存浪费；数组可以根据下标直接存取。

2. 链表动态地进行存储分配，可以适应数据动态地增减的情况，且可以方便地插入、删除数据项。（数组中插入、删除数据项时，需要移动其它数据项，非常繁琐）链表必须根据next指针找到下一个元素

从内存存储来看

1. (静态)数组从栈中分配空间, 对于程序员方便快速,但是自由度小

2. 链表从堆中分配空间, 自由度大但是申请管理比较麻烦

从上面的比较可以看出，如果需要快速访问数据，很少或不插入和删除元素，就应该用数组；相反， 如果需要经常插入和删除元素就需要用链表数据结构了。

# 设计模式相关

## 用过哪些设计模式？谈谈自己比较熟悉的设计模式

1、工厂模式2、代理模式3、策略模式4、观察者模式6、单例模式

工厂模式：

简单工厂模式解决的问题是如何去实例化一个合适的对象。

简单工厂模式的核心思想就是：有一个专门的类来负责创建实例的过程。凡是出现了大量的产品需要创建，并且具有共同的接口时，可以通过工厂方法模式进行创建。比如说写技能是一系列类，那么就可以使用工厂模式创建。

代理模式：一个是真正的你要访问的对象(目标类)，一个是代理对象,真正对象与代理对象实现同一个接口,先访问代理类再访问真正要访问的对象。

代理模式就是多一个代理类出来，替原对象进行一些操作，比如我们在租房子的时候回去找中介，为什么呢？因为你对该地区房屋的信息掌握的不够全面，希望找一个更熟悉的人去帮你做，此处的代理就是这个意思。再如我们有的时候打官司，我们需要请律师，因为律师在法律方面有专长，可以替我们进行操作，表达我们的想法。

代理模式的应用场景：

如果已有的方法在使用的时候需要对原有的方法进行改进，此时有两种办法：

1、修改原有的方法来适应。这样违反了“对扩展开放，对修改关闭”的原则。

2、就是采用一个代理类调用原有的方法，且对产生的结果进行控制。这种方法就是代理模式。

使用代理模式，可以将功能划分的更加清晰，有助于后期维护！

策略模式:定义一系列算法，并将每个算法封装起来，使他们可以相互替换，且算法的变化不会影响到使用算法的客户。策略模式的决定权在用户，系统本身提供不同算法的实现，新增或者删除算法，对各种算法做封装。因此，策略模式多用在算法决策系统中，外部用户只需要决定用哪个算法即可。

观察者模式很好理解，类似于邮件订阅和RSS订阅，当我们浏览一些博客或wiki时，经常会看到RSS图标，就这的意思是，当你订阅了该文章，如果后续有更新，会及时通知你。其实，简单来讲就一句话：当一个对象变化时，其它依赖该对象的对象都会收到通知，并且随着变化！对象之间是一种一对多的关系。

单例对象（Singleton）是一种常用的设计模式。在C#应用中，单例对象能保证在一个CLR中，该对象只有一个实例存在。这样的模式有几个好处：

1、某些类创建比较频繁，对于一些大型的对象，这是一笔很大的系统开销。

2、省去了new操作符，降低了系统内存的使用频率，减轻GC压力。

3、有些类如交易所的核心交易引擎，控制着交易流程，如果该类可以创建多个的话，系统完全乱了。（比如一个军队出现了多个司令员同时指挥，肯定会乱成一团），所以只有使用单例模式，才能保证核心交易服务器独立控制整个流程。

## 请说出 4 种面向对象的设计原则，并分别简述它们的含义。

1. 单一职责原则 （The Single Responsiblity Principle，简称

SRP）：一个类，最好只做一件事，只有一个引起它的变化.

1. 开放－封闭原则 （The Open－Close Principle，简称

OCP）：对于扩展是开放的，对于更改是封闭的

1. Liskov 替换原则（The Liskov Substitution Principle,简称

LSP）：子类必须能够替换其基类

1. 依赖倒置原则 （The Dependency Inversion Pricinple， 简称

DIP）：依赖于抽象

1. 接口隔离原则 （The Interface Segregation Principle，简称

ISP）：使用多个小的专门的接口，而不要使用一个大的总接口。

## 设计一个状态机类型，状态值为int类型，要求：

拥有接口，获取当前状态，切换状态

外部可以监听状态切换事件，参数为切换前状态和切换后状态（使用delete和event）//定义一个状态

public abstract class IAction{

public int StateName;

public IAction(int stateName) {

this.StateName = stateName;

}

public int GetState(){

return StateName;

}

public abstract bool CanGetIn();

public abstract void GetIn();

public abstract void GetOut();

public abstract void Update(float dt);

}

//定义一个状态切换事件

public abstract class IEvent {

public int code;

public IEvent(int code) {

this.code = code;

}

public int GetCode(){

return code;

}

//当前状态事件检测

public abstract bool Check();

}

public class AvatarStateMachine{

//初始化状态机

public void InitStateMachine(){}

//注册一个状态

public void AddAction(int action) {}

//注册一个状态切换事件

public void AddEventTransition(int fromAction, int toAction, IEvent byEvent) {}

//更新当前状态，检测能否进入下一个状态

public void UpdateStateMachine(){}

//强行切换状态

public void SwitchTo(int toState) {}

}

## 如何处理unity中界面资源，界面逻辑以及功能模块三者之间耦合关系

这就是在Unity使用MVC时通常将功能模块尽量脱离脱离MonoBehavior，同一个模块内M只用来操作数据并发送更新消息，V只用来接受消息并控制界面显示跳转，使用C来处理界面与数据的频繁操作。

## 观察者模式的深入理解？

观察者模式：一对多的关系，当被观察这发生改变时会通知所有观察者。让双方都依赖于抽象，使得各自变化不会影响另一方。

## MVC模式

用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

# 图形学相关

## 简述四元数的作用，四元数对欧拉角的优点？

四元数用于表示旋转

A.四元数一般定义如下：q=w+xi+yj+zk 其中 w,x,y,z 是实数。同时，有: i\*i=-1 j\*j=-1 k\*k=-1

B.四元数也可以表示为：q=[w,v] 有多种方式可表示旋转，如 axis/angle、欧拉角(Euler angles)、矩阵(matrix)、四元组等。 相对于其它方法，四元组有其本身的优点：

a.四元数不会有欧拉角存在的 gimbal lock 问题[万向节死锁]

b.四元数由 4 个数组成，旋转矩阵需要 9 个数

c.两个四元数之间更容易插值

d.四元数、矩阵在多次运算后会积攒误差，需要分别对其做规范化(normalize)和正交化 (orthogonalize)，对四元数规范化更容易

e.与旋转矩阵类似，两个四元组相乘可表示两次旋转

## 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

## 矩阵相乘的意义及注意点

用于表示线性变换：旋转、缩放、投影、平移、仿射

注意矩阵的蠕变：误差的积累

## alpha blend 工作原理

实际显示颜色 = 前景颜色\*Alpha/255 + 背景颜色\*(255-Alpha)/255

## 写光照计算中的 diffuse 的计算公式

实际光照强度 I= 环境光(Iambient) + 漫反射光(Idiffuse) + 镜面高光(Ispecular);

环境光：Iambient= Aintensity\* Acolor; (Aintensity 表示环境光强度，Acolor 表示环境光颜色)

漫反射光：Idiffuse = Dintensity\*Dcolor\*N.L;(Dintensity 表示漫反射强度，Dcolor 表示漫反射光颜色，N 为该点的法向量，L 为光源向量)

镜面反射光：Ispecular = Sintensity\*Scolor\*(R.V)^n;(Sintensity 表示镜面光照强度，Scolor 表示镜面光颜色，R 为光的反射向量，V 为观察者向量，n 称为镜面光指数)

## 两种阴影判断的方法工作原理

阴影由两部分组成：本影与半影

本影：景物表面上那些没有被光源直接照射的区域（全黑的轮廓分明的区域）

半影：景物表面上那些被某些特定光源直接照射但并非被所有特定光源直接照射的区域（半明半暗区域）

求阴影区域的方法：做两次消隐过程

一次对每个光源进行消隐，求出对于光源而言不可见的区域 L；

一次对视点的位置进行消隐，求出对于视点而言可见的面 S；

shadow area = L ∩ S

阴影分为两种：自身阴影和投射阴影

自身阴影：因物体自身的遮挡而使光线照射不到它上面的某些可见面

工作原理：利用背面剔除的方法求出，即假设视点在点光源的位置。

投射阴影：因不透明物体遮挡光线使得场景中位于该物体后面的物体或区域受不到光照照射而形成的阴影

工作原理：从光源处向物体的所有可见面投射光线，将这些面投影到场景中得到投影面，再将这些投影面与场景中的其他平面求交得出阴影多边形，保存这些阴影多边形信息，然后再按视点位置对场景进行相应处理得到所要求的视图（利用空间换时间，每次只需依据视点位置进行一次阴影计算即可，省去了一次消隐过程）若是动态光源此方法就无效了。

## Vertex Shader 是什么？怎么计算？

顶点着色器是一段执行在 GPU 上的程序，用来取代 fixed pipeline 中的 transformation和 lighting，Vertex Shader 主要操作顶点。

Vertex Shader 对输入顶点完成了从 local space 到 homogeneous space（齐次空间）的变换过程，homogeneous space 即 projection space 的下一个 space。在这其间共有 worldtransformation, view transformation 和 projection transformation 及 lighting 几个过程。

## 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。

## 怎么判断两个平面是否相交?不能用碰撞体,说出计算方法

对于两个平面Ax+By+Cz+D=0与ax+by+cz+d=0，只要(A,B,C)与(a,b,c)不成比例，这两个平面就是相交的。

## 法线贴图 、CG 动画

A.法线贴图：是在原物体的凹凸表面的每个点上均作法线，通过 RGB 颜色通道来标记法线的方向， 你可以把它理解成与原凹凸表面平行的另一个不同的表面，但实际上它又只是一个光滑的平面。

B.CG 动画：原为 Computer Graphics 的英文缩写。随着以计算机为主要工具进行视觉设计和生产的一系列相关产业的形成,国际上习惯将利用计算机技术进行视觉设计和生产的领域通称为 CG。它既包括技术也包括艺术，几乎囊括了当今电脑时代中所有的视觉艺术创作活动，如平面印刷品的设计、网页设计、三维动画、影视特效、多媒体技术、以计算机辅助设计为主的建筑设计及工业造型设计等。

## 什么是局部坐标，什么是世界坐标？

世界坐标是不会变的，一直以世界坐标轴的 XYZ 为标准。 局部坐标其实就是自身的坐标，会随着物体的旋转而变化的。

## 请描述 MeshRender 中 material 和 shader 的区别？

Shader（着色器）实际上就是一小段程序，它负责将输入的 Mesh（网格）以指定的方式和输入的贴图或者颜色等组合作用，然后输出。绘图单元可以依据这个输出来将图像绘制到屏幕上。输入的贴图或者颜色等，加上对应的 Shader，以及对 Shader 的特定的参数设置，将这些内容（Shader 及输入参数）打包存储在一起，得到的就是一个 Material（材质）Shader 大体上可以分为两类:表面着色器（Surface Shader） 、片段着色器（Fragment Shader）

## 什么是矢量图

矢量图：计算机中显示的图形一般可以分为两大类——矢量图和位图。矢量图使用直线和曲线来描述图形，这些图形的元素是一些点、线、矩形、多边形、圆和弧线等等，它们都是通过数学公式计算获得的。例如一幅花的矢量图形实际上是由线段形成外框轮廓， 由外框的颜色以及外框所封闭的颜色决定花显示出的颜色。由于矢量图形可通过公式计算获得，所以矢量图形文件体积一般较小。矢量图形最大的优点是无论放大、缩小或旋转等不会失真；最大的缺点是难以表现色彩层次丰富的逼真图像效果。

## 四元组是什么?

所谓四元数，就是把 4 个实数组合起来的东西。4 个元素中，一个是实部，其余 3 个是虚部。

## 你对你的数学知识感觉如何？比如空间向量，图形学

个人感觉还是不错的，空间向量和图形学之前有自学过。只是在实际工作中直接使用的机会比较少，底层的一些操作有些生疏了。

## 会写shader么

了解一些，但是写的不多

## 3D基础相关

2 1 0 1 0 1

1 0 1 × 0 0 0

2 1 1 1 1 1

上述两个矩阵相乘的结果

2 0 2

2 1 2

3 1 3

## 什么是投影矩阵

投影矩阵是一个典型的缩放和透视矩阵。投影变换将视锥变换成一个直平行六面体的形状。因为视锥的近处比远处小，这样就会对靠近摄像机的对象起到放大的作用，也就将透视应用到了场景当中

## 什么是UV

UV坐标是指所有的图象文件都是二维的一个平面。水平方向是U，垂直方向是V，通过这个平面的，二维的UV坐标系。我们可以定位图象上的任意一个象素。

## Cullback，cullfront，culloff区别

剔除背面、剔除前面、不剔除

## 什么是顶点程序和片段程序

顶点着色器是一组指令代码，这组指令代码在顶点被渲染时执行。

片段着色器也是在 GPU 上运行的小程序。它们负责输出每个呈现的三角形像素的最终像素颜色。基本而言，它的工作原理如下：片段着色器以输入的形式收到顶点着色器通过管道传递的所有这些片段。

## 如何实现以下人物在树丛中部分透明效果？

Shader "Custom/PlayerDiffuse" {

Properties {

\_NotVisibleColor ("NotVisibleColor (RGB) ", Color) = (0.3,0.3,0.3,1)

\_MainTex ("Base (RGB) ", 2D) = "white"{}

}

SubShader {

Tags { "Queue"= "Geometry+500""RenderType"="Opaque"}

LOD 200

Pass {

ZTest Greater

Lighting Off

ZWrite Off

//Color [\_NotVisibleColor]

Blend SrcAlpha OneMinusSrcAlpha

SetTexture [\_MainTex] { ConstantColor [\_NotVisibleColor] combine constant \* texture }

}

Pass {

ZTest LEqual

Material {

Diffuse (1,1,1,1)

Ambient (1,1,1,1)

}

Lighting Off

SetTexture [\_MainTex] { combine texture }

}

}

FallBack "Diffuse"

}

## 简述lightmap的使用

(此为帖子全文,请自己简述)

Unity 完全集成了光照贴图，可以通过编辑器创建完整的光照贴图，你完全不用担心，所有材质会自动获得光照贴图。光照贴图的意思是，所有灯光的特性将被直接映射到Beast lightmapper并烘培到纹理，以此获得更好的性能。UnityPro版扩展了全局光照功能，可以烘焙出真实而漂亮的光照，当然这就不能同时使用实时光照。此外，Unity专业版带给你天光和发光材质，为你增加更有趣的场景照明。

在此页中，你会发现更深入的描述，可以找到在Lightmapping窗口的所有属性。从菜单中Window – Lightmapping打开Lightmapping窗口。

物体

物体的烘培设置：灯光、网格渲染和地形 - 取决于当前的选择。

网格渲染器和地形：

·Static 静态

可渲染网格和地形必须标记为静态才能被烘培。

·Scale In Lightmap 光照图比率

（只作用于可渲染网格）特别大的数值将分配给可渲染网格更大的分辨率。最终分辨率比例（光照图缩放）\*（物体世界坐标空间所占面积）\*全局分辨率烘培设置）如果设置为0物体将不被烘培。（但是它依旧对其他的物体有影响）

·Lightmap Size 光照图大小

（只作用于地形）光照贴图尺寸是地形实例独有的属性，地形将不和其他物体共用图集。地形光照贴图将存贮在一个单独的文件中。

·Atlas 图集

图集信息-如果Lock Atlas（锁定图集）选项没有开启那么这些参数将自动更新。如果Lock Atlas（锁定图集）选项开启，这些参数将不会自动编辑。但是你可以手动设置他们。

·Lightmap Index 光照图索引

光照贴图序列中的索引。

·Tiling 平铺

（只作用于可渲染网格）物体光照贴图UVs平铺。

·Offset 偏移

（只作用于可渲染网格）物体UVs的偏移。

Lights 灯光：

·Lightmapping 光照贴图

光照图模式：仅实时模式，自动模式和仅烘培 模式。查看下面Dual Lightmaps的描述。

·Color 颜色

灯光颜色。一些属性只作用于实时光照。

·Intensity 光强度

灯光照明强度。一些属性只作用于实时光照。

·Bounce Intensity 反弹强度

特定光源间接光照强度的倍增值。

·Baked Shadows 烘焙阴影

控制当前灯光是否产生阴影。（当选择自动选项时同时影响实施阴影的产生）

·Shadow Radius 阴影半径

（只作用于点光源和聚光灯）增加这个值将得到一个较柔和的阴影。增加这个值的大小影响阴影覆盖面积范围的计算（不会影响到灯光照射范围）。

·Shadow Angle 阴影角度

（只影响平行光）增加这个值将会得到一个比较柔和的阴影-增加灯光阴影覆盖角度范围的计算。（不会影响到灯光照射范围）

·Shadow Samples 阴影采样

如果你将阴影半径或者角度设置为大于0的值，增加阴影采样值能得到一个比较好的结果。高采样值将消除半影中中的噪点。

Bake 烘焙

全局烘焙设置。

·Mode 模式

控制离线烘培和实时光照模式。在Dual Lightmaps 模式下近端和远端光照图都将被烘培；只有延迟光照渲染方式式才能更好的支持双重光照图。Single Lightmaps 模式仅烘培远端光照图。在延迟渲染方式下也可以强制使用单光照图模式。

·Use in forward rendering 使用正向渲染

(Dual lightmaps only) 在正向渲染中启用dual lightmaps。要注意的是这需要你创建自己的shader来达到目的。

·Quality 质量

高质量（比较好的效果）和低质量（计算速度快）的预设值。它们将影响最终聚集光线数量和对比度的数值以及其他有关最终聚集和抗锯齿的设置。

·Bounces 反弹

控制全局光照模拟中光线反弹的次数。如果想得到一个比较柔和的相对真实的光照至少需要将此数值设置为1.如果设置为0那么只会计算直接光照。

·Sky Light Color 天光颜色

天光是模拟从天球向所有方向发射光线照射场景-在烘培室外场景的时候开启天光将得到一个非常好的效果。

·Sky Light Intensity 天光强度

天光的强度 - 0代表禁用天光。

·Bounce Boost 反弹增强

增强间接光， 可用来增加场景中渲染太快没有烘焙出灯光的反弹量。

·Bounce Intensity 反弹强度

间接光照的强度倍增器。

·Final Gather Rays 最终聚集光线

控制从最终聚集点发射的光线数量-较高的数值能得到更好的效果 。

·Contrast Threshold

对比度阈值

颜色对比度根据采样运算规则控制哪些最终聚集点被创建。较高的数值可以使Beast处理物体表面灯光照明时有很大的改善，从而得到一张平滑的光照贴图但是会牺牲一些照明细节。较低的数值需要最终聚集光线数量要高于对比度阈值这样被过滤掉的最终聚集点才能被创建出来。

·Interpolation 插值

控制最终聚集点颜色的插值。设置0为线性插值，设置1为基于梯度的高级插值。多数情况下推荐使用后者。

·Interpolation Points 插值点

设置最终聚集点颜色的插值的参考点。高数值可以获得比较平滑的结果，但是会牺牲一些光照细节。

·Ambient Occlusion

环境光遮挡

烘培光照图时产生一定数量的环境阻光。环境阻光计算物体每一点被一定距离内的其他物体或者一定距离内自身物体的遮挡程度（用来模拟物体表面环境光及阴影覆盖的比例，达到全局光照的效果），所以和所有的灯光设置没有太大关系。

·Max Distance 最大距离

Beyond this distance a ray is considered to be unoccluded. 0 stands for infinitely long rays.

超出这个距离的光线将不被遮挡。0表示无限长的光线。

·Contrast 对比度

控制完全阻光和不阻光之间的过度。

·Lock Atlas 锁定图集

当锁定图集被打开。自动图集功能将不执行，物体光照图索引，平铺和偏移将不能编辑。

·Resolution 分辨率

每世界单位中光照图分辨率的大小。因此当设置分辨率为50的一个10单位乘以10单位的平面将产生一张500\*500像素的光照贴图。

Maps 贴图所有光照贴图的可编辑数组。

·Compressed 压缩

控制是否压缩所有当前场景中的光照贴图。

·Array Size 数组大小

光照贴图数组的大小（0～254）。

·Lightmaps Array 光照贴图数组

当前场景被烘培的所有光照图的可编辑序列。没有被指定的通道将被处理为黑色光照图。索引相当于网格渲染和地形中所使用的光照图编号。除非锁定图集被开启否则序列将自动初始化并分配给你烘培的贴图。

Lightmap Display 光照贴图显示

在编辑器中控制光照图如何显示的工具。光照图显示是在场景视口中的一个子窗口。可见只要光照图窗口可见。

·Use Lightmaps 使用光照贴图

是否渲染光照图。

·Shadow Distance 阴影距离

根据过度到只使用远端光照图时哪些设置为自动模式的灯光被关闭的距离。这里的设置覆盖其他面板中相关设置。但是不能覆盖质量设置中的阴影距离设置。

·Show Resolution 显示分辨率

切换在场景中是否显示光照图分辨率。开启这项设置你可以在你的场景中直接预览静态物体光照图的分辨率大小。

Details 细节

Dual Lightmaps 双重光照贴图

双光照贴图是Unity中的光照贴图可以同高光贴图，法线贴图和适当混合的实时阴影一起渲染的工具。同时它也可以让你的光照贴图看起来效果更好，即使设置的光照分辨率较低。

双光照贴图默认情况下只能用于延迟光照渲染路径。但通过编写自定义着色器就可能让双光照贴图用于前向渲染路径中（使用dualforward表面着色指令）。

双光照贴图即意为使用两套光照贴图：

·远 ：包含全部照明

·近 ：包含标记为自动模式的灯光所产生的的间接照明，和标记为仅烘焙模式的灯光，发光材质和天光的所有照明。

设置为仅实时的灯光永远不会被烘培。近端光照图作用于质量设置中的相对摄像机最小阴影距离范围内。

在这个距离范围内，自动模式的灯光将实时渲染高光，凹凸和阴影，（它们的阴影与实时模式的等光的阴影融为一体），间接光则来自近端光照贴图。 若超出阴影距离自动模式灯光就不再进行实时渲染，而全部照明都由光照贴图产生（仅实时模式灯光仍然产生照明，但不产生阴影）。

下面的场景包含一个默认设置为为自动模式的平行光，一些烘培完光照贴图的静态物体（建筑物，障碍物，其他静态物体）和一些动态的移动或可移动物体（持枪的假人，桶）。 该场景在双重光照贴图模式中烘焙和渲染：阴影距离以外的建筑物完全由光照贴图产生照明，而两个假人由实时灯光产生动态光照，但是不投射阴影; 阴影距离范围内的建筑物和地面被即时光照亮并投下实时阴影，但柔和的间接光是来自于近端光照贴图产生。

## Shader的代码实现？大概写一下

Shader "Custom/NewSurfaceShader"{

Properties {

\_Color ("Color", Color) = (1,1,1,1)

\_MainTex ("Albedo (RGB) ", 2D) = "white" {}

\_Glossiness ("Smoothness", Range(0,1)) = 0.5

\_Metallic ("Metallic", Range(0,1)) = 0.0

}

SubShader {

Tags { "RenderType"="Opaque" }

LOD 200

CGPROGRAM

// Physically based Standard lighting model, and enable shadows on all light types

#pragma surface surf Standard fullforwardshadows

// Use shader model 3.0 target, to get nicer looking lighting

#pragma target 3.0

sampler2D \_MainTex;

struct Input {

float2 uv\_MainTex;

};

half \_Glossiness;

half \_Metallic;

fixed4 \_Color;

void surf (Input IN, inout SurfaceOutputStandard o) {

// Albedo comes from a texture tinted by color

fixed4 c = tex2D (\_MainTex, IN.uv\_MainTex) \* \_Color;

o.Albedo = c.rgb;

// Metallic and smoothness come from slider variables

o.Metallic = \_Metallic;

o.Smoothness = \_Glossiness;

o.Alpha = c.a;

}

ENDCG

}

FallBack "Diffuse"

}

## 游戏中要怎么实现矩阵相乘？

//矩阵相乘

public static float[][] Mul(float[][] a, float[][] b) {

//确保矩阵a的列数和b的行数相等

if(a[0].length != b.length) {

return null;

}

//用来存放结果的矩阵，axb的结果为a的行数和b的列数

float[][] result = new float[a.length][b[0].length];

//对a的每行进行遍历

for(int i=0; i<a.length; i++) {

//对b的每列进行遍历

for(int j=0;j<b[0].length; j++) {

//c为每一个点的值

float c = 0;

//第i行j列的值为a的第i行上的n个数和b的第j列上的n个数对应相乘之和，其中n为a的列数，也是b的行数，a的列数和b的行数相等

for(int k=0; k<a[0].length; k++) {

c += (a[i][k]\*b[k][j]);

}

result[i][j] = c;

}

}

return result;

}

# 逻辑题

## 也许是因为我太善良，我上的当是要比别人多一些。D

1. 非常同意 B.比较同意 C.比较不同意 D.非常不同意

## 现在生活中老实人总是吃亏。D

1. 非常正确 B.比较正确 C.比较不正确 D.非常不正确

## 社会上很多不法分子逍遥法外 D

A.非常正确 B.比较正确 C.比较不正确 D.非常不正确

## 如果我家很小，如带客人来家我会有伤自尊 D

A.非常同意 B.比较同意 C.比较不同意 D.非常不同意

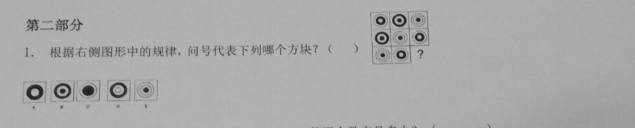
## 我多数娱乐活动都是和学生一起进行的 A

A.非常同意 B.比较同意 C.比较不同意 D.非常不同意

## 如果有件小事（如头痛或家务事），使我感到烦恼时，我会向别人倾诉 C

A.非常同意 B.比较同意 C.比较不同意 D.非常不同意

## 根据如下图形中的规律，问号代表下列哪个方块？ A



## 观察数字：54321,43215,32154，（），15432，第四个数字是多少？

21543

## 在下列分数中，选出不同类的一项： C 可以约分

A.3/5 B.3/7 C.3/9

## 小赵的店里来了一位顾客，挑了20元的货，顾客拿出50元，小赵没有零钱找不开，就到隔壁小韩的店里把这50元换成零钱，回来给顾客找了30元零钱，过了一会儿，小韩来找小赵，说刚才的是假钱，小赵马上给小韩换成了一张真钱，问：在这一过程中小赵赔了多少钱？ D

A.50元 B.60元 C.70元 D.80元

## 观察3、3、8、8这一组数字，不改变数字顺序，加入运算符号和括号，将这些数字组成一个算式，使结果等于27

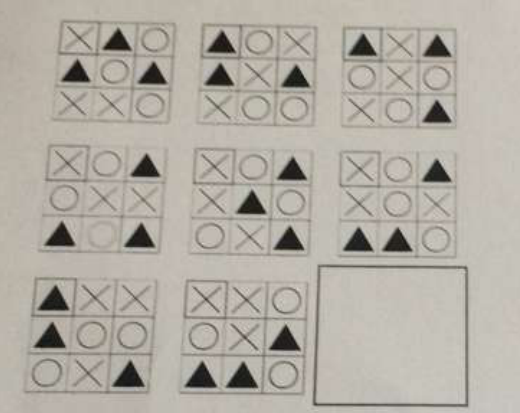
答案：(3+3/8)\*8=27

## 图形共有2000个，按照下面的规律排列： △□□□△□□□△□□□……第1995个图形是什么？

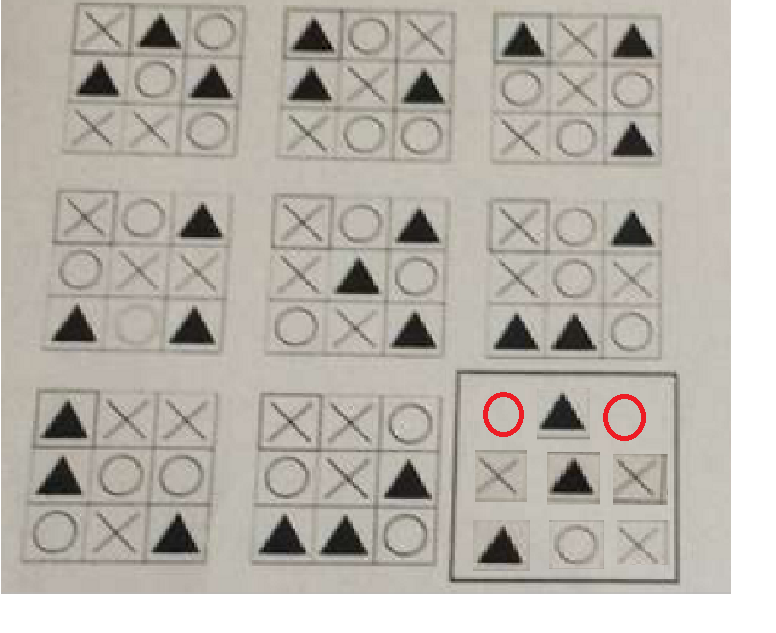
1995 % 4 = 3 所以是方块

## 用三根火柴要摆成一个最小的数（不许把火柴折断或者弯曲），这个数是什么？

-11



根据逻辑关系在方框内填出空缺处图形。



## 说一个岛上有100个人，其中5个红眼睛，95个蓝眼睛。这个岛上有三个奇怪的宗教规则。

他们不能照镜子，不能看自己眼睛的颜色。

他们不能互相交流眼睛颜色的问题，也不能告诉别人对方的眼睛是什么颜色。

一旦有人知道了自己眼睛的颜色，他就必须在当天夜里自杀。

注：虽然题设了5个红眼睛，但岛民是不知道具体数字的。

岛民一直过着宁静的生活，相安无事。某天，有个旅行者到了这个岛上，由于不知道这里的规矩，所以一起狂欢的时候，不留神就说了一句：你们这里有红眼睛的人。

最后的问题是：

1）假设这个岛上的人够聪明，每个人都可以做出缜密的逻辑推理。请问这个岛上将会发生什么？

2）为什么旅行者看似无意义的一句话改变了岛上的格局？

" 因为自从那个人说了岛上有红眼人后他们开始思考自己眼睛的颜色，假如有一个红眼人，那么他看到99个蓝眼人，他第一天会自杀，如果有两个红眼人，他看到98蓝1红，如果他发现那个红眼人第一天没自杀那么他将意识到他也是红眼人，故第二天两个红眼人都将自杀，同理可推知，第五天五个红眼人将自杀，五个红眼人自杀之后，通过推理，剩下的都是蓝眼人了，所以第六天，剩下的蓝眼人都自杀了

## 有4位好友，其中一个人干了一件好事，不留名，记者来查此事，问4位，他们的回答如下：

A：不是我

B.是C

C.是D

D.他胡说

已知3个人说真话，一个人说假话，现在根据以上信息找出做了好事的人(并编程实现)

1.假设A讲假话,其馀三人讲真话,则B和C出现矛盾,所以假设不成立,A讲真话.

2.假设B讲假话,其馀三人讲真话,则C和D的

说法出现矛盾,所以假设不成立,B讲真话.

3.假设C讲假话,其馀三人讲真话,则A、B、D的证供都成立,所以假设成立,C讲假话.

4.假设D讲假话,其馀三人讲真话,则B和C出现矛盾,所以假设不成立,D讲真话.

推出,C是做好事的人.

提示：

第一步：被测者从1—4

第二步：将被测者代入表达式

第三步：判断3个表达式为真吗?是的则打印结果程序结束。不是,判断是否全测完，不是转第一步，是打印未找到，程序结束。完成下面的程序。

# include <stdio.h>

main(){

int k = 0,sum = 0,g = 0;

char man = ' ';

for(k = 1; k <= 4 && g==10;k++) {

man=64+k;

sum=(man != 'A') + (man == 'C') + (men == 'D') + (men != 'D');

if(sum==3) {

clrscr();

printf("\nman=%c\n", 64 + k);

g = 1;

}

}

if(g != 1)

printf("can't found! ");

getch();

}