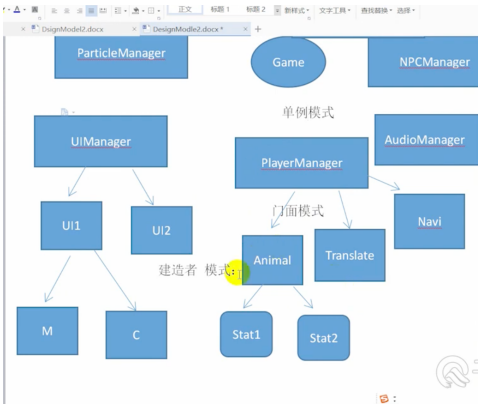
get set方法 外界可以访问 内部进行设置

用于封装数据 将对数据的访问动作和数据本身区分开

你可以在getter和setter方法中做一些转换而不是直接操作数据本身

TexturePacker 打成大图

兼容所有机型1024*1024



playerManager 单例模式

建造者模式 。小的模块（UImanager，NPCManager） 玩家playermanager--有数据，有transform，有位移，有animatoin，animation下面又有状态一 状态二。

门面模式：如果玩家要切换动画位移，同一层级

中介模式，玩家之间和怪物之间要进行伤害计算

观察者模式

1. u3d生命周期 MonoBehivior脚本的几个回调机制

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

FixedUpdata:每固定帧数绘制执行一次，和Updata不同的是渲染帧执行，如果渲染效率低下的时候fixedUpdata调用的次数会跟着下降----比较适合在物理引擎的计算，以为跟每帧渲染有关。FixedUpdate，是在固定的时间间隔执行，不受游戏帧率的影响
FixedUpdate更新的间隔可以在项目设置中更改，Edit->Project Setting->time ->Fixed Timestep。

Updata比较适合做控制

放在LateUPdate;否则会有空帧出现。

LataUpdata:每帧执行后完成调用，比较适合用于命令脚本的执行，官网是相机的跟随，都是updata执行后才跟进摄像机，不然就有可能出现摄像机已推进，但是视角里还未有角色的空帧出现

Unity 3D中的 GameObject.SetActive() 与 MonoBehaviour.OnEnable()、MonoBehaviour.OnDisable()

其实这三之前的关系很简单：

SetActive(true)，很触发MonoBehaviour.OnEnable()事件，就算对象之前本就是activeSelf==true，事件依然会发生；

SetActive(false)，很触发MonoBehaviour.OnDisable()事件,就算对象之前本就是activeSelf==false，事件依然会发生；

二、问题

在实际的开发中，对SetActive的使用是非常频繁的，所以以对已经是activeSelf==true的对象，设置SetActive(true)，和对已经是activeSelf==false的对象，设置SetActive(false)。很正常的，或者说是不可避免的。

这样问题就来了，比如，我们连续多次都调用了SetActive(true)，也会多次响应OnEnable()。就是说我们在保持对象是activeSelf==true的状态没有变化的情况下，OnEnable()方法会被多次执行。这时就它就会出问题了，当然也可能不会出问题，这个主要看人在OnEnable()实现的逻辑了。

同理，SetActive(false)，和OnDisable()与会有同样的问题。

三、解决

其实解决这个问题的办法也很简单的，就是在每次SetActive()之先检测对就的activeSelf状态。

我们可以封装一个自己的SetActive方法，来代替它：

```
static public void SetActive(GameObject go, bool state)
{
        if (go == null)
        {
                return;
        }

        if (go.activeSelf != state)
        {
                go.SetActive(state);
        }
}
```

2. prefab是什么 有什么好处

预制体 更改预制体的设置可以同时对场景的预制体实力进行更改

简述prefab的用处和环境

在游戏云顶的时候实例化，prefab相当于一个模板，对已有的素材、脚本、参数做一个默认的配置，以便以后的修改，同时prefab打包的内容简化了导出的操作，便于团队的交流

3. GC垃圾回收产生的原因，并如何避免

GC回收堆上的内存

避免：一.减少new对象的次数，在new对象时会产生内存碎片，这样会造成碎片内存无法使用

二．使用公用的对象（静态成员），也不能乱用，以为静态成员和常量在整个声明周期都存在

三．大量字符串的时候用stringbuilder。创建stringbuilder对象要设置stringbuilder的初始大小，StringBuilder sbd=new StringBuilder（size）

GC什么时候调用

堆内存不足的时候 GC会被调用

4. 动态加载资源的方式

一. Resources.Load()

二. 通过www从服务器上获取AssetBunlder资源，从bundle中某个object资源

AssetBuddle 优点：减小压缩包、资源更新、分开安装包和数据包、AssetBuddle加密不容易被反编译

缺点：读取麻烦 组装麻烦 、各平台打的包互相不兼容、耗内存

Resources ：

1， 不能超过 2G 。

2， 文件夹过大 启动游戏的时候非常的慢 。在给所有的文件建立索引。

3， 自动打包 。

2.如何进行内存优化？

1.压缩自带类库； 2.将暂时不用的以后还需要使用的物体隐藏起来而不是直接Destroy掉； 3.释放AssetBundle占用的资源； 4.降低模型的片面数，降低模型的骨骼数量，降低贴图的大小； 5.使用光照贴图，使用多层次细节(LOD)，使用着色器(Shader)，使用预设(Prefab)。

Lod是什么，优缺点是什么？

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。

LOD（多层次细节）

全称 Levels of Detail，进行物体不同细节层次之间的平滑过渡。 根据物体所在的环境和所处位置的重要度，决定渲染。降低不重要和远的物体的面数和细节度。一般都是视距近的物体清楚，视距远的物体模糊。 Lod：它是游戏中最常用的游戏优化技术，离摄像机越远的物体在LOD的优化下变的越模糊，非常节省不必要的资源开销。

1. 请描述Interface与抽象之间的不同。

抽象类表示该类中可能已经有一些方法的具体定义，但接口就是只能定义各个方法的界面，不能具体的实现代码在成员方法中。

类是子类用来继承的，当父类已经有实际功能的方法时该方法在子类中可以不必实现，直接引用父类的方法，子类也可以重写该父类的方法。

实现接口的时候必须要实现接口中所有的方法，不能遗漏任何一个。

委托和接口的区别

两者在功能上是一样的!

区别在于:

1.委托只是单一的方法，而接口可以封装多个方法

2.委托只约束方法的签名，而接口约束方法的名称

所以从多态的角度来将，委托可以认为是轻量级的接口。

居然有人认为没关系....

事件和委托的区别

事件就是一个狭义的委托,也就是事件是一个用于事件驱动模型的专用委托.

通俗的讲,委托你可以在客户代码中直接调用委托来激发委托指向的函数,而事件不可以,事件的触发只能由服务代码自己触发

也就是说在你的代码里委托你不但可以安排谁是它的调用函数,还可以直接调用它,而事件不能直接调用,只能通过某些操作触发

你可以理解事件就是一个或多个委托,此话应该有误的吧,事件可以有多个事件处理函数,委托同样也可以是个多播委托

2. 请简述ArrayList和List<Int>的主要区别

答：ArrayList存在不安全类型‘（ArrayList会把所有插入其中的数据都当做Object来处理）

装箱拆箱的操作（费时）

List是接口，ArrayList是一个实现了该接口的类，可以被实例化。

强类型 要检测数据安全性

14.Unity3D的协程和C#线程之间的区别是什么？

答：http://blog.csdn.net/kongbu0622/article/details/8775037

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。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asynchronous[e ɪ' s ɪŋ kr ə n ə s].synchronous同步。

43、线程是不是越多越好？如何确定最优线程数量？

Unity是单线程的，但是也还是可以自己开线程，而且5.0之后unity将变成多线程的编辑器。

协同不过是模拟多线程的而已，并非是真正意义的多线程

堆和栈的区别。

1. 使用栈就象我们去饭馆里吃饭，只管点菜（发出申请）、付钱、和吃（使用），吃饱了就走，不必理会切菜、洗菜等准备工作和洗碗、刷锅等扫尾工作，他的好处是快捷，但是自由度小。

   使用堆就象是自己动手做喜欢吃的菜肴，比较麻烦，但是比较符合自己的口味，而且自由度大。 （经典！）

1、栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其

操作方式类似于数据结构中的栈。

2、堆区（heap） — 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回

收 。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。

3、全局区（静态区）（static）—，全局变量和静态变量的存储是放在一块的，初始化的

全局变量和静态变量在一块区域，　　未初始化的全局变量和未初始化的静态变量在相邻的另

一块区域。　　-　　程序结束后由系统释放。

4、文字常量区　　—常量字符串就是放在这里的。　　程序结束后由系统释放

5、程序代码区—存放函数体的二进制代码。

heap是堆 。 stack是栈

stack 栈的空间由系统分配释放 。 heap的空间是手动申请和释放

stack空间有限，heap的空间是很大的自由区

X.值类型和引用类型的区别

一.值类型存储在内存栈中，引用类型数据存储在堆上，而内存单元中存放的时堆内存放的地址

二．值类型存取快，引用类型存储慢

三．值类型表示实际的数据，引用类型表示指向存储在内存堆中的指针和引用

四．栈的内存自动释放，堆内存是由。Net中会由GC来自动释放

五．值类型继承自System.Value.Type,引用类型继承System.Object

2．如何优化内存？

有很多种方式，例如

1.压缩自带类库；

2.将暂时不用的以后还需要使用的物体隐藏起来而不是直接Destroy掉；

3.释放AssetBundle占用的资源；

4.降低模型的片面数，降低模型的骨骼数量，降低贴图的大小；

5.使用光照贴图，使用多层次细节(LOD)，使用着色器(Shader)，使用预设(Prefab)。

14.Unity3D的协程和C#线程之间的区别是什么？（待）

答：　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。

```
IEnumerator LoadScene()
{
    async = SceneManager.LoadSceneAsync("game");//异步跳转到game场景
        async.allowSceneActivation = false;
```

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asynchronous .synchronous同步。　　（加载的进度条）

8.请简述关键字Sealed用在类声明和函数声明时的作用

答：类声明时可防止其他类继承此类，在方法中声明则可防止派生类重写此方法。

9.请简述private，public，protected，internal的区别

答：

public：对任何类和成员都公开，无限制访问

private：仅对该类公开

protected：对该类和其派生类公开

internal：只能在包含该类的程序集中访问该类

protected internal：protected + internal

5.　碰撞器和触发器的区别

collider触发器有碰撞的效果，isTrigger=false，可以调用OnCollisionEnter/Stay/Exit函数

Tigger触发没有效果，isTrigger=true,可以调用OnTriggerEnter/Stay/Exit函数

6.　CharacterController和RigidBody的区别

Rigidbody具有完全真实物理的特性，而，CharacterController可以说时受限的RigidBody，具有一定物理效果，但并不是完全真实。

7. localPosition:自身位置，相对于父级的变化位置

Postion：在世界位置transform的位置

8. u3d，集中添加力的方式，描述出来

Rigbody.AddForce/AddForceAtPosition

Force：添加一个可持续力到刚体，使用它的质量。

Acceleration：添加一个可持续加速度到刚体，忽略它的质量。

Impulse：添加一个瞬间冲击力到刚体，使用它的质量。

VelocityChange ：添加一个瞬间速率变化给刚体，忽略它的质量。

9. unity提供的光源的类型

平行光：Directional Light 点光源：Point Light

聚光灯：Spot Light 区域光源：Area Light 需要烘焙

1. 如何实现游戏的暂停，加速和减速？

2. Time.timeScale=0运行 Time.timeScale=1运行

正确的应该是timeScale不会影响Update和LateUpdate的执行速度。因为FixedUpdate是根据时间来的，所以timeScale只会影响FixedUpdate的速度。

3. 请描述游戏动画有哪几种，以及其原理？

关节动画，骨骼动画，关键帧动画

a.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；

b.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；

c.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。

假如一个GameObject上有一个脚本，怎样直接删除这个脚本

this.RemoveCompoment<>();

40、如何手动释放内存？
如果有Instantiate的对象，用Destroy进行销毁。
在合适的地方调用Resources.UnloadUnusedAssets,释放已经没有引用的Asset.
如果需要立即释放内存加上GC.Collect()，否则内存未必会立即被释放，有时候可能导致内存占用过多而引发异常。

请描述为什么Unity3d中会发生在组件上出现数据丢失的情况
一般是组件上绑定的物体对象被删除了

55、游戏中你使用什么格式的音乐。

.AIFF　适用于较短的音乐文件可用作游戏打斗音效
.WAV　适用于较短的音乐文件可用作游戏打斗音效
.MP3　适用于较长的音乐文件可用作游戏背景音乐
.OGG　适用于较长的音乐文件可用作游戏背景音乐

56、向量的点乘、叉乘的区别是什么？
向量的点乘得到的是一个数值，而不是向量，在unity中表示为
Vector3.Dot（Vector3，Vector3）：Float；
两个向量差乘得到的还是一个向量，在unity中表示为
Vector3.Cross（Vector3，Vector3）：Vector3

44、如何实现小地图？
1.通过插件实现
2.场景中放置两个摄像机，一个是场景中的跟随人物的（摄像机a），一个是用于呈现地图的摄像机（摄像机b）。创建一个render texture，把贴图拖到摄像机b的Target texture，b照射的物体将在这个render texture中显示。然后建一个材质球，把render texture贴到材质球上，然后在场景中添加Quad对象，把材质球拖到这个对象上，quad就能展示b相机照射的地图了。之后用遮罩shader可以调节quad的展示形状，做成一个任意的小地图形状。

Lua原理 为什么lua不报错

Lua代码是一个内存块 ， 系统不知道这个内存块里有什么东西，

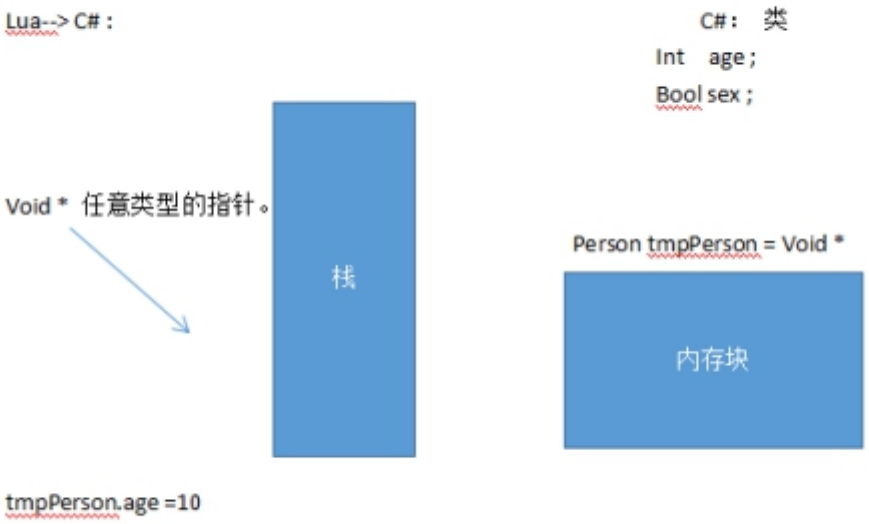
Lua把void的任意类型的指针 强制转换成所需要的类型

不加Local 默认覆盖全局的变量

点和冒号的区别 点静态方法 冒号是动态方法

Lua的table #查数组的长度 用for循环代替

C#区别和lua报错之后 后面别的代码就不会执行了



EventSystem组件主要负责处理输入、射线投射以及发送事件。一个场景中只能有一个EventSystem组件，并且需要BaseInputModule类型组件的协助才能工作。EventSystem在一开始的时候会把自己所属对象下的BaseInputModule类型组件加到一个内部列表，并且在每个Update周期通过接口UpdateModules接口调用这些基本输入模块的UpdateModule接口，然后BaseInputModule会在UpdateModule接口中将自己的状态修改成'Updated'，之后BaseInputModule的Process接口才会被调用。

42.物体发生碰撞的必要条件

物体A必须带有（collider+rigidbody）或者CharacterController，另一个物体也必须至少带有collider

有相对运动

关于Unity中RectTransform和Transform

以前一直以为在Inspector面板上的是Transform，后来才发现原来2D是RectTransform，3D是Transform

3D面板上显示的是位置坐标组件Transform，2D面板上显示的是位置坐标组件RectTransform

RectTransform是Transform的子类
可以这样强制转换，反过来则不行
RectTransform r_trans = (RectTransform)this.transform;

结构体和类的区别
结构体是值类型，类是引用类型
值类型用于存储数据，引用类型用于储存对实际数据的引用
在这个例子中，最显著区别就是关键字——“struct”而不是“class”。其他区别包括：

- 结构体不能从基类继承，但类可以
- 结构体不能有无参构造函数
- 在构造函数结束之前，所有的结构体域都必须被赋值
- 结构体是传值，而类的实例是传引用

6.MipMap是什么？作用？
在三维计算机图形的贴图渲染中有一个常用的技术被称为Mipmapping。为了加快渲染速度和减少图像锯齿，贴图被处理成由一系列被预先计算和优化过的图片组成的文件,这样的贴图被称为 MIP map 或者 mipmap。

17.请简述向量的点乘，向量的叉乘以及向量归一化的几何意义？
点乘的几何意义是：计算两个向量之间的夹角，以及在某一方向上的投影；
叉乘的几何意义是：创建垂直于平面，三角形，或者多边形的向量；
20.请简述如何在不同分辨率下保持UI的一致性
NGUI很好的解决了这一点，屏幕分辨率的自适应性，原理就是计算出屏幕的宽高比跟原来的预设的屏幕分辨率求出一个对比值，然后修改摄像机的size。
原生GUI http://unity3d.9ria.com/?p=2587
NGUI http://blog.csdn.net/mfc11/article/details/17681429
28.如何安全的在不同工程间安全地迁移asset数据？三种方法

答：

将Assets目录和Library目录一起迁移

导出包

用unity自带的assets Server功能

RawImage和Image的区别

RawImage可以是任何类型的Texture

Image只能是精灵(Sprite)

texture更底层 2D 3D都能用 sprite主要还是用于2D 且默认支持自动打包贴图

```
            Input.multiTouchEnabled  =  true;//开启多点触碰
  (Input.touchCount  >  1)//多点触碰
(touch.phase  ==  TouchPhase.Moved)  //手指在移动
```

原理

对象池原理

遥感

```
//在圆内
if (deltaDistance<=radius)
{
    frontImage.transform.position = eventData.position;
}
else
{
    //世界位置=相对位置+圆心的位置
    frontImage.transform.position = deltaPos.normalized* radius+orignPos;

}

//圆内圆外都要算
//求出弧度
float tmpAngle = Mathf.Atan2(deltaPos.y, deltaPos.x);

tmpAngle = Mathf.Rad2Deg * tmpAngle;

//改变z值，使其旋转
Vector3 tmpEuler = arrowImage.transform.localEulerAngles;

tmpEuler.z = tmpAngle;

arrowImage.transform.localEulerAngles = tmpEuler;

//计算 箭头的位置
arrowImage.transform.position= deltaPos.normalized * (radius+selfSize)*0.5f + orignPos;
if (PlayerCtrl .Instance. playerM.BloodCount > 0)

//控制玩家的旋转
PlayerCtrl.Instance.RotatePlayer(90-tmpAngle);
}
```

```
0 个引用
public void OnPointerClick(PointerEventData eventData)
{
    if (PointerClick != null) PointerClick(eventData);
}

7 个引用
public void OnPointerDown(PointerEventData eventData)
{
    if (PointerDown != null) PointerDown(eventData);
}

7 个引用
public void OnPointerUp(PointerEventData eventData)
```

技能冷却

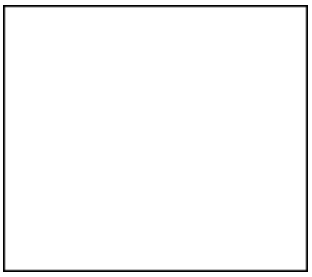
```
public IEnumerator MaskCircle()
{
    isFinish = false;
    float timeCount = 0;
    while (timeCount<totalTime)
    {
        timeCount += Time.deltaTime;

        maskImage.fillAmount = 1 - timeCount / totalTime;
        yield return new WaitForEndOfFrame();
    }
    isFinish = true;
}
}
```

小地图原理

获取地形的包围盒的宽 和高

1, 相对地形的位置= 人物的世界位置 -- 地形的世界位置。

2, 比例= 相对位置 / 地形的长和宽



8

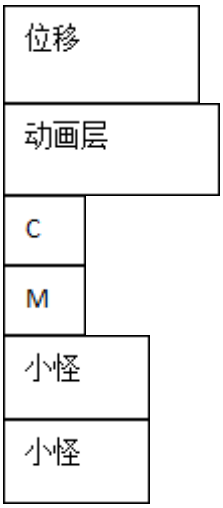
相对小地图的位置 = 小地图的长和宽 * 比例

```
public void CaculatePos()
{
    //相对位置
    Vector3 deltaPos=PlayerCtrl.Instance.transform.position - terrian.position;
    BoxCollider box = terrian.GetComponent<BoxCollider>();

   float xRate=  deltaPos.x / box.size.x;

    float zRate = deltaPos.z / box.size.z;

    Vector2 result = Vector2.zero;
    RectTransform parentRect = (RectTransform)head.transform.parent;

    result.x = parentRect.sizeDelta.x * xRate;
    result.y= parentRect.sizeDelta.y * zRate;


    head.rectTransform.anchoredPosition = result;
}
```

Manager

NPC:　小怪 。

怪物管理器：


位移
动画层
C
M
小怪
小怪