Time

void Update () {  
        cube.Translate (Vector3.forward\*Time.deltaTime);  
        Debug.Log ("Time.deltaTime:"+Time.deltaTime);  
    }

乘以Time.deltaTime之后，第一句话就是每秒移动一米的意思，没乘的话，就是一秒60帧，Update一秒要调用60次，这句话的意思就是一秒移动60米。Time.deltaTime的意思是完成一帧所需要的时间，一秒60帧，那么一帧需要1/60秒，第一句话的意思是每帧移动，（一米乘以1/60）米，那么一秒60帧移动60\*1\*1/60=1米，即一秒一米。

cube.Translate (Vector3.forward\*Time.deltaTime\*3);是一秒运动3米的意思。

void Update () {  
        cube.Translate (Vector3.forward\*Time.deltaTime);  
        Debug.Log ("Time.deltaTime:"+Time.deltaTime);  
        Time.timeScale = 3f;  
    }

最后一句是帧速度加快3秒的意思

float time3 = Time.realtimeSinceStartup; 是获取现在的时间

GameObject

一．

void Start () {  
        new GameObject();  
    }

取个名字new GameObject("cube"); GameObject go = new GameObject("cu");

1. 根据prefab进行实例化

GameObject go = new GameObject("cu");

可以根据prefab或者另外一个游戏物体克隆。

1. primitive创建一些基本的物体

GameObject.CreatePrimitive(PrimitiveType.Plane);  
GameObject.CreatePrimitive(PrimitiveType.Cube);

1. 对游戏物体添加一些组件

GameObject go = GameObject.CreatePrimitive(PrimitiveType.Cube);  
go.AddComponent<Rigidbody> ();  
go.AddComponent<time> (); //time是已存在的脚本

1. 物体激活状态

Debug.Log (go.activeInHierarchy);  
go.SetActive (false);  
Debug.Log (go.activeInHierarchy);  
Debug.Log (go.tag); //tag

1. 获取组件名字

Debug.Log (go.name);  
Debug.Log (go.GetComponent<Transform>()); //gameobject的名字可以通过游戏物体获取，也可以通过游戏组件获取，因为transform是没有名字的，获取出来的是他所在物体的名字

1. Object静态方法

destroy可以销毁某个物体，或者组件或者延迟几秒在销毁

Destroy,DontDestroyOnLoad(transform.gameObject)

DontDestroyOnLoad(transform.gameObject)是在场景切换的时候，本来之前场景的物体是会被销毁的，这句话就是不要销毁上个场景的传递过来的物体

Object.FindObjectOfType去搜索所有游戏物体上所有组件，返回第一个找到的type类型的组件

Object.FindObjectsOfType找到所有type组件，并返回数组

Light light = FindObjectOfType<Light> ();  
        light.enabled = false; //并禁用

Transform[] ts = FindObjectsOfType<Transform> ();//不查找未激活游戏物体  
        foreach (Transform t in ts)  
            Debug.Log (t.name); //输出了名字

还有initiate是初始化物体旋转和位置的

FindWithTag也是只返回第一个，FindObjectsWithTag返回数组。（根据标签查找比根据名字查找效率高）

    GameObject fin = GameObject.Find ("Main Camera");  
        fin.SetActive (false); //根据名字查找并禁用

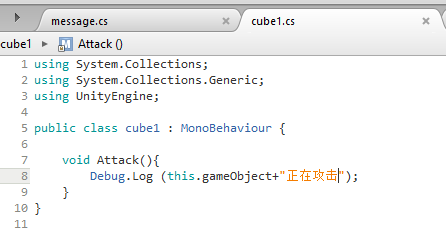
GameObject[] gos = GameObject.FindGameObjectsWithTag("Main Camera");  
        gos[0].SetActive(false); //一般要判断数组是否为空

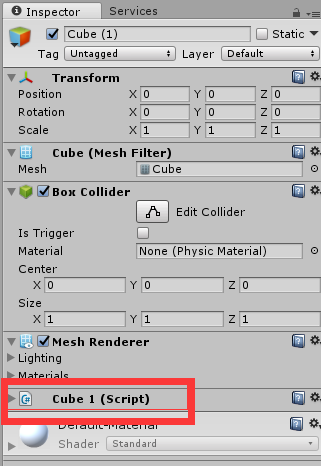
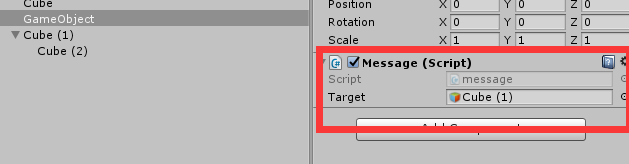
1. 物体间消息的发送和接收

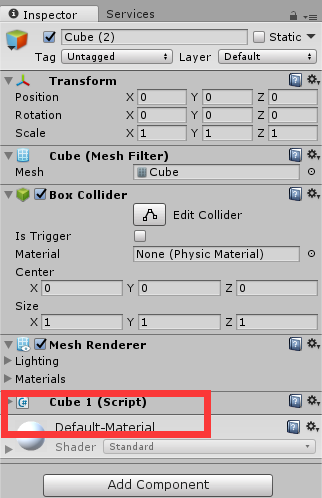
BroadcastMessage广播消息,只要游戏物体和他的子物体带有指定方法，都会被调用

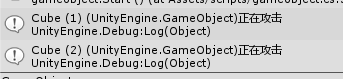
target.BroadcastMessage ("Attack",null,SendMessageOptions.DontRequireReceiver); //第三个参数意思是不一定需要接收者，就是说如果有接收者传递，如果没有接收者也不要报错











GameObject.SendMessage只调用游戏物体身上的方法，不调用他的孩子

target.SendMessage("Attack",null,SendMessageOptions.DontRequireReceiver);

GameObject.SendMessageUpwards当前物体及其父亲包括爷爷，包括爷爷的爷爷

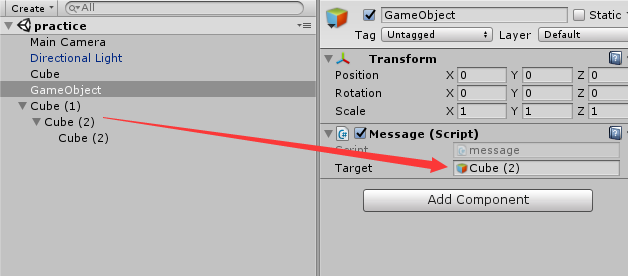


游戏才能添加组件，组件不能添加组件

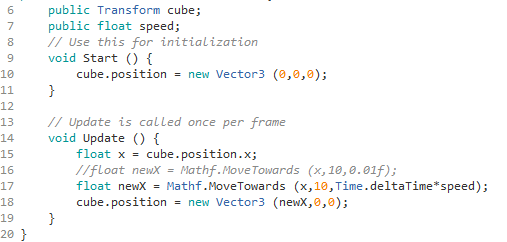
GetComponent得到游戏身上第一个要求的组件,GetComponents得到游戏物体身上所有的组件，GetComponentinChildren游戏物体和他所有的孩子的第一个要求的组件,

GetComponentsinChildren游戏物体和他所有的孩子的所有的要求的组件,





下面是匀速运动

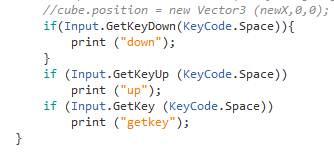


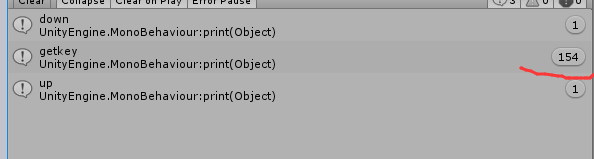
pingpong运动cube.position = new Vector3(Mathf.PingPong(Time.time\*speed,10),0,0);

其中，第一个参数是速度，第二个参数是乒砰来回运动的长度，没有起始点和重点的指定，就只有往返运动长度指定

**Input**

getkey只要按键按下就一直被触发，getkeyup,getkeydown是抬起或按下的时候被触发一次。





用GetAxis控制移动

print (Input.GetAxis("Horizontal"));  
        cube.Translate (Vector3.right\*Time.deltaTime);

。。。。。。。。。。。。。。。。。。。。。

cube.Translate (Vector3.right\*Time.deltaTime\*Input.GetAxis("Horizontal")); 当不按键的时候，GetAxis("Horizontal"是0，不移动，但是是加速效果

cube.Translate(Vector3.right\*Time.deltaTime\*Input.GetAxisRaw("Horizontal"));

只要按下，GetAxisRaw("Horizontal")值为1，不按为0，不会对速度造成影响

if (Input.anyKeyDown)  
            print ("any key down"); 如果有鼠标或者键盘按键，

anyKeyDown返回true

print (Input.mousePosition); 获取鼠标坐标，坐标以屏幕左下角为（0，0），以像素为单位

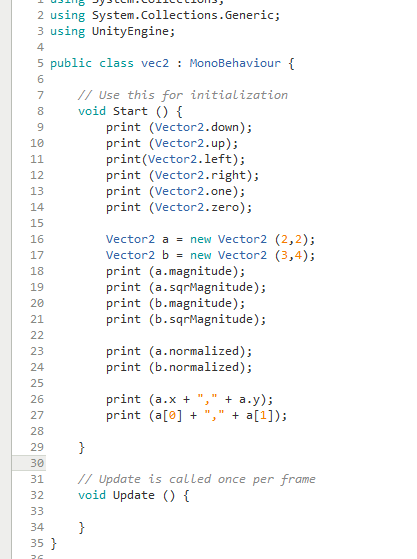
Vector2

Normalized，把向量单位化，a.normalized 返回a单位化之后的向量，但是对a没有影响

magnitude=sqrt(x^2+y^2).

sqrMagnitude=x^2+y^2,如果是比较两个向量的长度大小用sqrMagnitude就可以了，算平方耗费性能

Vector2的用法



如果只修改一个坐标的值，不能写成transform.position.x = 10，因为position是结构体

要写成

Vector2 pos = transform.position;

pos.x = 10;

Transform.position = pos;

向量是结构体，是值类型，要整体赋值

Vector2的Equals是判断两个向量是否x,y值都一样，返回布尔值

Normalize把向量单位化产生向量不返回值

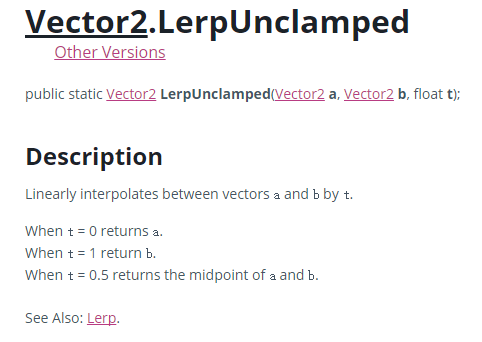
Angle两个向量的夹角

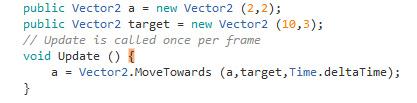
ClampMagnitude（Vector2 vector,float maxLength）如果vector小于最大长度，则返回原向量，否则返回最大长度的向量

Distance,a-b得到的向量的长度，a,b为向量

Lerp,对a.x,b.x进行插值，再对a.y,b.y插值

Max返回最大长度的向量MoveTowards,当前向量到目标向量移动，根据传递的速度参数



匀速运动

Vector3

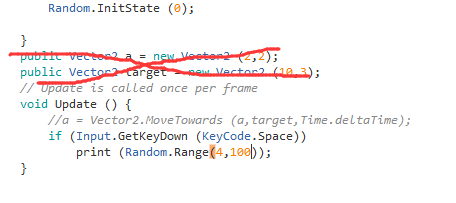
可以代表点和向量，x代表左右，y代表上下，z代表前后.one(1,1,1),zero(0,0,0)

Cross,左手法则。Slerp向量插值

把二维转三维坐标，z坐标为0 ，反之，z舍弃

Random

Random.InitState(seed); Random.Range(a,b);

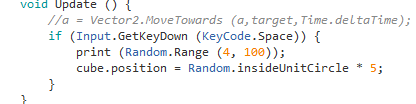
生成随机数，要在start里面给一个InitState的种子，然后再生成。但是种子一样，每次运行生成的随机数都是一样的，只要种子一样，每次生成的随机数都是一样的

为了每次运行生成随机数不一样，用Random.InitState ((int)System.DateTime.Now.Ticks);

获取系统时间，这个系统时间，每次程序运行是不一样的。

Random.insideUnitCircle在一个指定半径的圆内生成随机位置

Transform.position = Random.insideUnitCircle\*5;//在半径为5米的圆内。。。。



Random.insideUnitSphere在一个球体内随机生成位置

**Quaternion四元数**

完成旋转。X,Z轴rotation改变是围绕x,z旋转，而y不是，y是围绕世界坐标的y轴旋转。欧拉角代表物体在x,y,z分别旋转多少度。四元数也能表示旋转。欧拉角方便观察，四元数方便计算。

上面是欧拉角，下面是世界坐标四元数，如果是localrotation,就是局部坐标下的四元数

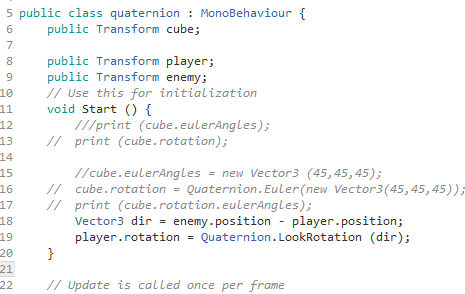
不能cube.rotation = new Vector3(10,0,0);因为前面是四元数，后面是欧拉角

应该cube.eulerAngles = new Vector3(10,0,0);

cube.rotation = Quaternion.Euler(new Vector3(45,45,45));

将欧拉角转换为四元数

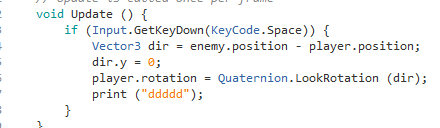
print (cube.rotation.eulerAngles); 这样是把四元数转换为欧拉角



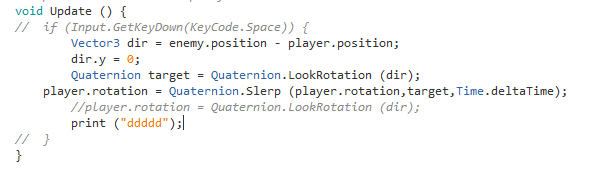
将玩家望向敌人，但是，可能会有弯腰动作，因为两者高度不一致

    Vector3 dir = enemy.position - player.position;  
        dir.y = 0;  
        player.rotation = Quaternion.LookRotation (dir);

这样就没有高度差了



Quaternion.Slerp是在做朝向旋转是推荐用的，与角度有关的都推荐s开头的

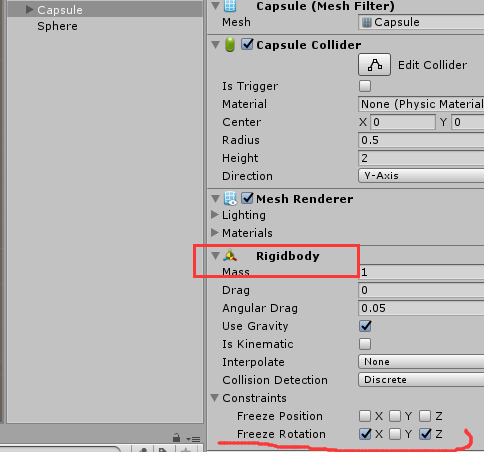
缓慢旋转，在deltatime后面乘一个数，还要放慢速度

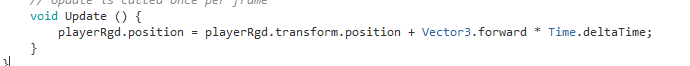
RigidBody

CenterOfMass求物体重心，也有potision,rotation,也可以通过刚体组件设置这些值

Rigidbody.position比transform.position快，性能好

用Rigidbody.position移动时，这个物体要加rigidbody,

并且冻结x,z轴，不然胶囊会旋转



但是移动还是推荐用MovePosition,是用插值的，运动看起来平滑

playerRgd.MovePosition(playerRgd.transform.position + Vector3.forward\*Time.deltaTime\*10);

MonoBehaviour是Component下的部件

Debug.Log输出可以在任意地方调用，但是print只能在MonoBehaviour里面调用，因为它是MonoBehaviour里面的方法

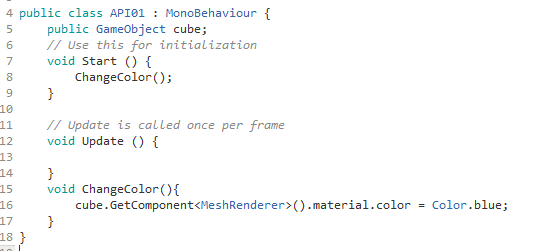
Invoke：CancelInvoke,Invoke,InvokeRepeating,IsInvoking



协程

协程方法不会阻塞主方法的执行，还自己可以暂停

下面不是协程，是顺序



改变物体颜色，颜色是在MeshRenderer组件里面的.cube通过MeshRenderer组件控制渲染的。通过material访问到身上的材质，通过color访问到材质的颜色.

下面是协程

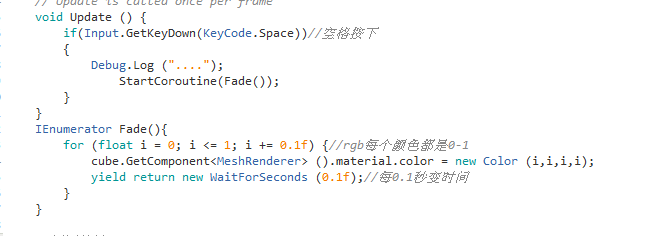


每次暂停都要用yield

yield return new WaitForSeconds (3);

协同程序内部暂停3秒

cube渐变颜色



下面是颜色渐变到红色，注意理解

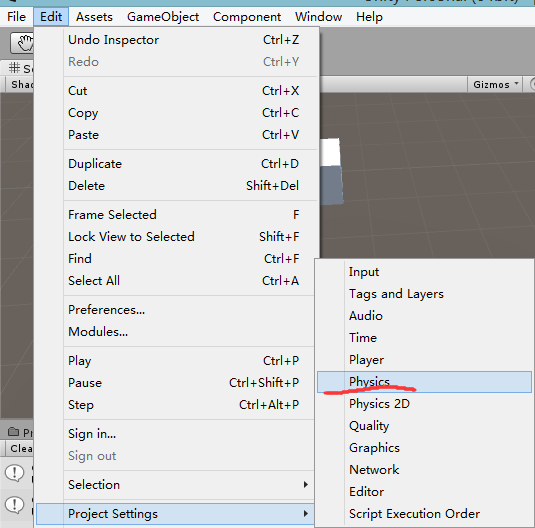


把每次颜色变化之后的颜色赋值给color,再改变颜色，并把新颜色赋值给newColor,用newColor作为物体新颜色，新的循环，把物体此时的颜色又赋值给color,这个循环中颜色每次变化0.02f,每隔0.02秒变化一次，判断新颜色和红色的g值如果小于等于0.01，则退出循环。Color.Lerp是一个差值运算

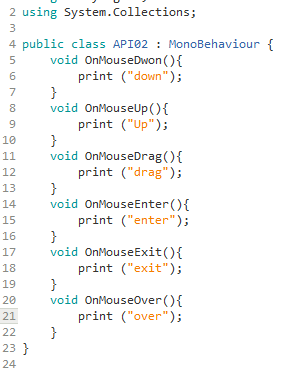


协程的开启与关闭



鼠标检测1，必须加collider,2.勾选3.

射线检测勾选



鼠标在哪个物体点击触发，拖拽之后无论在哪放，都是原物体触发up

void OnMouseUpAsButton() 当点击和抬起在同一物体上时，触发一次

Mathf

Mathf.Deg2Rad,Mathf.Rad2Deg,Mathf.Infinity（无限的数）,Mathf.NegativeInfinity（无限小的数，是一个负数）,

Mathf.PI,Mathf.Epsilon//最后这个指一个非常小的数

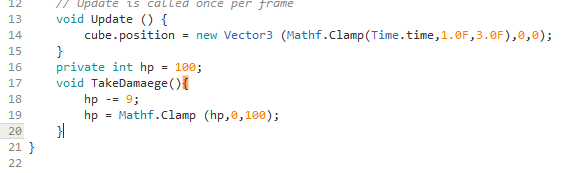
Mathf.Ceil向上取整，返回float Abs,取绝对值

Mathf.Floor向下取整 Mathf.FloorToInt

Mathf.CeilToInt向上取整，返回int

Mathf.Clamp(float value,float min,float max),value小于min返回min,大于max,返回max,在中间返回value

Mathf.Clamp01把一个值限定到0-1之间



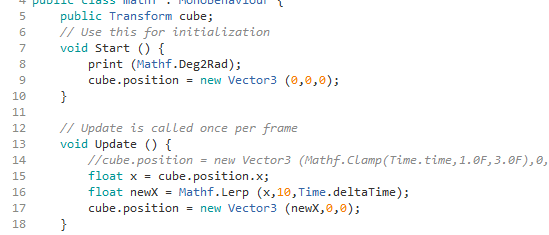
Update是改变物体位置，TakeDamaege是用于游戏血量改变

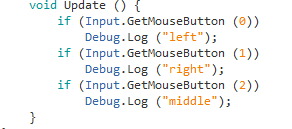
Mathf.DeltaAngle两个角度最短距离

Mathf.Pow(float f,float p),f的p次方

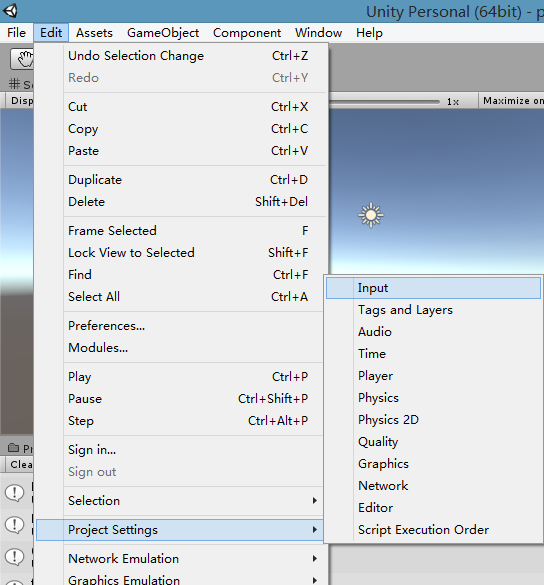
Mathf.ClosestPowerOfTwo(n)取n到2的次方里面最近的数，eg，2，4，6，8，16，32

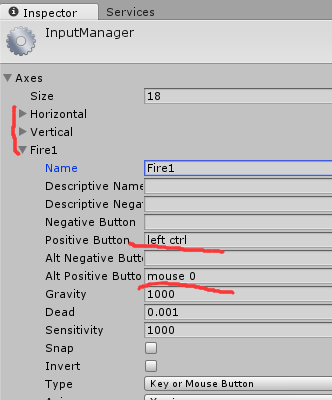
Mathf.Lerp(float a,float b,float t);a到b中间的值，t代表插的比例



鼠标点击事件

虚拟键





if (Input.GetButtonDown ("Fire1"))  
            print ("fire");

Vector3 dir = enemy.position - playerRgd.position;

Dir.y = 0;

Quaternion target = Quaternion.LookRotation(dir);

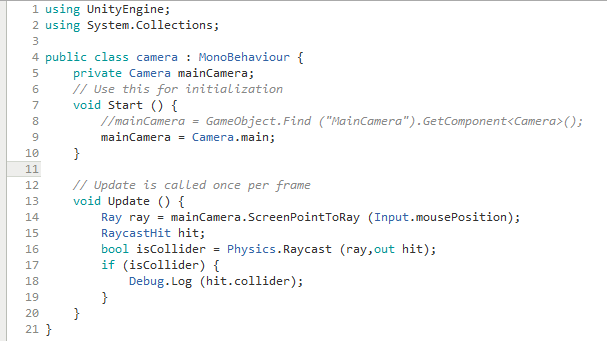
playerRgd.MoveRotation(Quaternion.Lerp(playerRgd.rotation,target,Time.deltaTime));//旋转用MoveRotation

Addforce : playerRgd.AddForce (Vector3.forward\*force);

查找main camera有两种方式



其中第二种条件是maincamera所在的tag必须是



这是通过相机，把鼠标变成射线，检测碰撞的。

把相机的一条射线画出来，10米，这个点的方向

Application.datePath