坐标与单位

单位

unity中有很多网格，网格的尺寸为1 unit，对应屏幕上的100px。

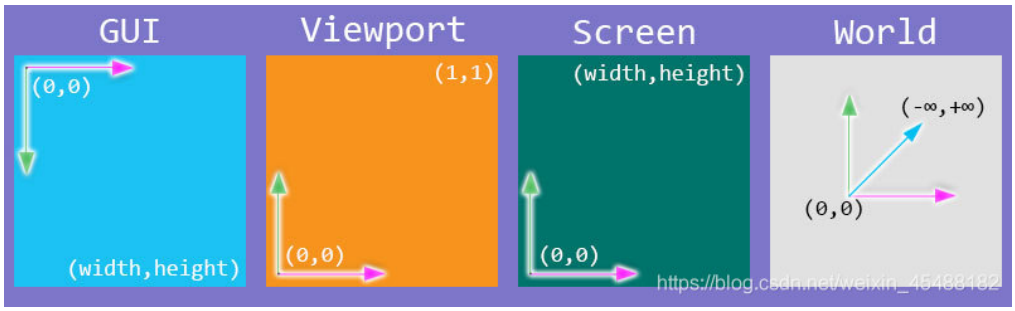
unity中，长度单位就是unit，1unit显示为100px。场景的高度为10unit。

在真实世界中1unit多大，可以自己约定。

坐标系

x朝向右，y轴向上

unit中，z轴的方向采用左手系，也就是朝里面的。



Mathf数学函数类

```
1  private float endTime = 10;
2
3     void Start()
4     {
5         // 静态变量
6         print(Mathf.Deg2Rad+",度到弧度换算常量");
7         print(Mathf.Rad2Deg+ ",弧度到度换算常量");
8         print(Mathf.Infinity+"正无穷大的表示形式");
9         print(Mathf.NegativeInfinity + "负无穷大的表示形式");
10        print(Mathf.PI);
11        // 静态函数
12        print(Mathf.Abs(-1.2f)+ ",-1.2的绝对值");
13        print(Mathf.Acos(1)+",1（以弧度为单位）的反余弦");
14        print(Mathf.Floor(2.74f)+",小于或等于2.74的最大整数");
15        print(Mathf.FloorToInt(2.74f)+",小于或等于2.74的最大整数");
16        a+(b-a)*t
17        print(Mathf.Lerp(1,2,0.5f)+",a和b按参数t进行线性插值");

18        print(Mathf.LerpUnclamped(1, 2, -0.5f) + ",a和b按参数t进
行线性插值");
```

```
    }

    void Update()
    {
        print("游戏倒计时：" + endTime);
        endTime = Mathf.MoveTowards(endTime,0,0.1f);
    }
```

bug

功能：1.占位 2.检错（可以每一步进行检测，可以很明显的知道哪一步错了）

```
Debug.Log("UnityAPI常用方法类与组件");
Debug.LogWarning("这是一个警告！");
Debug.LogError("这里有报错！");
```

Transform

查找预制体中的某个子对象

```
 对话框文本 = 对话面板.transform.GetChild(0).GetChild(0).GetComponent<TMP_Text>();
```

移动缩放和旋转

根据坐标系的来移动

```
//1.第一个参数按世界坐标系移动，第二个参数指定世界坐标系（实际情况按世界坐标系移动）
grisGo.transform.Translate(Vector2.left*moveSpeed,Space.World);

//2.第一个参数按世界坐标系移动，第二个参数指定自身坐标系（实际情况按自身坐标系移动）
grisGo.transform.Translate(Vector2.left * moveSpeed, Space.Self);

//3.第一个参数按自身坐标系移动，第二个参数指定世界坐标系（实际情况按自身坐标系移动）
grisGo.transform.Translate(-grisGo.transform.right * moveSpeed, Space.World);

//4.第一个参数按自身坐标系移动，第二个参数指定自身坐标系（实际情况按世界坐标系移动）(一般不使用)
grisGo.transform.Translate(-grisGo.transform.right * moveSpeed, Space.Self);

//旋转
grisGo.transform.Rotate(new Vector3(0,0,1));
grisGo.transform.Rotate(Vector3.forward,1);
```

成员变量

```
Debug.Log("Gris变换组件所挂载的游戏物体名字是："+grisTrans.name);
Debug.Log("Gris变换组件所挂载的游戏物体引用
是："+grisTrans.gameObject);
Debug.Log("Gris下的子对象（指Transform）的个数
是："+grisTrans.childCount);
Debug.Log("Gris世界空间中的坐标位置是："+grisTrans.position);
Debug.Log("Gris以四元数形式表示的旋转是："+grisTrans.rotation);
Debug.Log("Gris以欧拉角形式表示的旋转（以度数为单位）
是"+grisTrans.eulerAngles);
Debug.Log("Gris的父级Transform是："+grisTrans.parent);
Debug.Log("Gris相对于父对象的位置坐标是："+grisTrans.localPosition);
Debug.Log("Gris相对于父对象以四元数形式表示的旋转是：" +
grisTrans.localRotation);
Debug.Log("Gris相对于父对象以欧拉角形式表示的旋转（以度数为单位）是：" +
grisTrans.localEulerAngles);
Debug.Log("Gris相对于父对象的变换缩放是："+grisTrans.localScale);
Debug.Log("Gris的自身坐标正前方（Z轴正方向）是："+grisTrans.forward);
Debug.Log("Gris的自身坐标正右方（X轴正方向）是：" + grisTrans.right);
Debug.Log("Gris的自身坐标正上方（Y轴正方向）是：" + grisTrans.up);
```

查找相关的成员方法

```
Debug.Log("当前脚本挂载的游戏对象下的叫Gris的子对象身上的Transform组件
是："+transform.Find("Gris"));
Debug.Log("当前脚本挂载的游戏对象下的第一个（0号索引）子对象的Transform引
用是："+transform.GetChild(0));
Debug.Log("Gris当前在此父对象同级里所在的索引位置："+
grisTrans.GetSiblingIndex());
```

静态方法

```
Transform.Destroy(grisTrans);
Transform.Destroy(grisTrans.gameObject);
Transform.FindObjectOfType();
Transform.Instantiate();
```

获取父节点

```
GameObject parent = this.transform.parent.gameObject;
Debug.Log(parent.name);
```

遍历所有子节点

```
Transform[] myTransforms = GetComponentsInChildren<Transform>();
foreach (var child in myTransforms)
{
    Debug.Log(child.name);
}
```

查找二级子物体

需要标注路径

```
Debug.Log(transform.Find("Child0/Child00"));//二级子物体
```

设置父子节点关系

注意，决定父子节点关系的是由`transform`组件控制的，所以说，在设置父子节点的时候，需要设置的是transform。

```
GameObject obj1 =  GameObject.Find("女主抱胸_1");
GameObject obj2 = GameObject.Find("Circle");
obj1.transform.SetParent(obj2.transform);
```

把节点设置为顶级节点

如果把SetParent设置为null，就会挂载到当前场景上去。

Gizmos可视化辅助

Gizmos能且只能在MonoBehaviour相关子类中，使用特定的函数调用，其中：

OnDrawGizmos() 在每帧调佣。所有在OnDrawGizmos中的渲染都是可见的。

OnDrawGizmosSelected() 仅在脚本附加的物体被选择时调用。

```
    private void OnDrawGizmos()
    {

    }

    private void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.blue;
        Gizmos.DrawCube(transform.position, Vector3.one);
    }
```

Random

```
 void Start()
    {
        // 静态变量
        print(Random.rotation+",随机出的旋转数是(以四元数形式表示)");
        print(Random.rotation.eulerAngles+",四元数转换成欧拉角");
        print(Quaternion.Euler(Random.rotation.eulerAngles)+",欧拉角转四元数");
        print(Random.value+",随机出[0,1]之间的浮点数");
        print(Random.insideUnitCircle+",在（-1，-1）~（1,1）范围内随机生成的一个vector2");
        print(Random.state+",当前随机数生成器的状态");
        // 静态函数
        print(Random.Range(0,4)+",在区间[0,4）（整形重载包含左Min,不包含右Max）产生的随机数");
        print(Random.Range(0, 4f) + ",在区间[0,4）（浮点形重载包含左Min,包含右Max）产生的随机数");
        Random.InitState(1);
        print(Random.Range(0,4f)+",设置完随机数状态之后在[0,4]区间内生成的随机数");
    }
```

MonoBehaviour基类

Behaviour与MonoBehaviour的关系：Mono继承自Behaviour,Behaviour继承自Compontent，Compontent继承自Object

```
Debug.Log("No4_MonoBehaviour组件的激活状态是："+this.enabled);
Debug.Log("No4_MonoBehaviour组件挂载的对象名称是：" + this.name);
Debug.Log("No4_MonoBehaviour组件挂载的标签名称是：" + this.tag);
Debug.Log("No4_MonoBehaviour组件是否已激活并启用Behaviour：" + this.isActiveAndEnabled);
```

Input

1.连续检测（移动）

```
连续检测（移动）
    // (-1到1)
        print("当前玩家输入的水平方向的轴值是："+Input.GetAxis("Horizontal"));
        print("当前玩家输入的垂直方向的轴值是：" + Input.GetAxis("Vertical"));
   // (-1,0,1三个值)
        print("当前玩家输入的水平方向的边界轴值是：" + Input.GetAxisRaw("Horizontal"));
        print("当前玩家输入的垂直方向的边界轴值是：" + Input.GetAxisRaw("Vertical"));
        print("当前玩家鼠标水平移动增量是："+Input.GetAxis("Mouse X"));
        print("当前玩家鼠标垂直移动增量是：" + Input.GetAxis("Mouse Y"));
```

2.连续检测（事件）

```
    //连续检测（事件）
     // bool类型
    if (Input.GetButton("Fire1"))
    {
        print("当前玩家正在使用武器1进行攻击！");
    }
    if (Input.GetButton("Fire2"))
    {
        print("当前玩家正在使用武器2进行攻击！");
    }
    if (Input.GetButton("RecoverSkill"))
    {
        print("当前玩家使用了恢复技能回血！");
    }

}
```

3.间隔检测（事件）

```
   // 间隔检测（事件）
    if (Input.GetButtonDown("Jump"))
    {
        print("当前玩家按下跳跃键");
    }
    if (Input.GetButtonUp("Squat"))
    {
        print("当前玩家松开蹲下建");
    }
    if (Input.GetKeyDown(KeyCode.Q))
    {
        print("当前玩家按下Q键");
    }
    if (Input.anyKeyDown)
    {
        print("当前玩家按下了任意一个按键，游戏开始");
    }
    if (Input.GetMouseButton(0))
```

```
    {
        print("当前玩家按住鼠标左键");
    }
    if (Input.GetMouseButtonDown(1))
    {
        print("当前玩家按下鼠标右键");
    }
    if (Input.GetMouseButtonUp(2))
    {
        print("当前玩家抬起鼠标中键（从按下状态松开滚轮）");
    }
```

画线系统

地形系统

1. 在Hierarchy面板中，右键3D object，选择Terrain
2. 在Terrain组件中选择Paint Terrain，可以来绘制地形
3. 其中有一个下拉框，可以绘制很多不同的东西

TileMap

用于绘制像素画

1. 在hierarchy中右键——2D Object——TileMap——rectanglar

   这个是创建画布的，画板上的内容可以直接画到这个上面

2. window——2D——palette

   创建画板，这个就是RPGMV左边那个画板。

3. 把sprite拖入到platee中去，此时会出现一个保存文件夹，建议把tile的资源单独保存

4. 要是画布上的图片有空白，就去sprite中，看看pixels per unit和图片大小是否一样

可以添加TileMap collider 2d来进行图块碰撞器的添加

还可以添加composite collider 2D来进行碰撞器的合并，这样可以提高性能，同时也可以防止多个碰撞器之间的缝隙，以免玩家卡进去.

如果不想被碰撞，那么需要去**tile保存的那个文件夹**，把对应的tile设置为none

自动寻路

1. 选择window——AI——navigation
2. 在Hierarchy中选择想要寻路的地图，在navigation中选择bake
3. 如果某一个物体不想被烘焙，可以点击在navigation中的Object的navigation area选择not walkable
4. 3D自动寻路的代码

   agent把玩家自身拖过来就行

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class Character : MonoBehaviour
{
    public NavMeshAgent agent;


    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

        if (Input.GetMouseButtonDown(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit raycastHit;
            if (Physics.Raycast(ray, out raycastHit))
            {
                print(raycastHit.point);
                agent.SetDestination(raycastHit.point);
            }
        }
```

```
30
31      }
32  }
33
```

1. 对需要寻路的物体添加static
2. 在window-ai下面找到navigation面板
3. 在bake下面进行烘焙
4. 如果是普通的gameobject，可能会出现可以通过的情况，此时选中物体，在navigation面板中的object中，可以选择Navigation Area是否通过。
5. 代码

```
1  using UnityEngine;
2  using UnityEngine.AI;
3
4  public class Character : MonoBehaviour
5  {
6      public NavMeshAgent agent;
7
8
9      void Start()
10     {
11
12     }
13
14     // Update is called once per frame
15     void Update()
16     {
17
18         if (Input.GetMouseButtonDown(0))
19         {
20             Ray ray =
   Camera.main.ScreenPointToRay(Input.mousePosition);
21             RaycastHit raycastHit;
22             if (Physics.Raycast(ray, out raycastHit))
23             {
24                 print(raycastHit.point);
25                 agent.SetDestination(raycastHit.point);
26             }
27         }
28
29     }
30 }
```

动画系统

Timeline

Cinemachine

2D相机

1. 在Hierarchy中创建Vitural Cinema
2. 把主角挂载到 CM vcam中
3. Body 选择Framing Transposer

- Dead zoom：当不为0时，会出现一个死区，玩家在其中运动的时候，镜头不会跟随，进入蓝色区域后才会跟随。
- Damping：阻尼，也就是镜头跟随玩家的速度

- 白区：跟随对象在白区内，摄像机不跟随。白区大小受Dead zoom影响。
- 蓝区： 跟随对象移动至蓝区中，摄像机就会跟随。速度会受Damping影响。
- 红区： 跟随对象高速移动时，会由于摄像机没有及时跟随上目标而丢失目标不在跟随。此时红区的作用就是，当跟随目标触及红线时，摄像机会立即跟随目标而不是缓慢跟随。

  红区的大小受到Soft Zone影响。

增加镜头移动边缘：

1. 在add Extension中选择confiner 2D，
2. 创建一个多边形碰撞器
3. 将碰撞器拖入到confiner 2D中
4. 要注意碰撞器的大小要比镜头大才有效果

灯光系统

2D光照

1. 安装Scriptable Build Pipeline
2. 安装Universal RP

四种灯光

制作天空盒

1. 在网上找一个天空盒的素材
2. 拖到unity中
3. texture type改成cube
4. 拖到画面上
5. 此时会自动生成一个material
6. 可以在material的inspector面板修改相关属性

影响光照的因素

- 天空盒

  在lighting界面中，可以设置

- Directional Light

  这个是模拟平行光的光影

- Environment Light

  在lighting界面设置

Gamma空间和Linear空间

全局光照

全局光照由直接光照和间接光照组成。

全局光照可以使暗面没有那么暗

如果想要开启实时全局光照，需要在lighting中的scene开启realtime Global illumination

灯光烘焙

选择烘焙的时候，只有baked模式下的物体会有灯光

UGUI系统

自动排列，滚动框

Scroll Rect组件：

Text Mesh Pro

获取

```
using TMPro;
dialogText = GameObject.Find("对话框").GetComponent<TMP_Text>();
```

解决无中文问题

1. 将一个字体拖入unity
2. 右键-creat-text mesh pro-for asset

支持的转义字符

```
表情<sprite=1>

字体大小<size=30>大小

字体颜色<color=#FFFFFF>白色

超链接<link=123>显示的文本
```

动态创建按钮

```
private void OnGUI()
{
    if (GUI.Button(new Rect(10, 20, 100, 40), "这是一个按钮"))
    {
        print("点击");
    }
}
```

单选框

1. 给父组件添加`Toggle Group`组件
2. 创建ui-toggle的子组件
3. 把父组件拖到Group里面就行了

组件

Outline：描边

Shadow：阴影

创建滚动条

富文本

代码	类型	
`<b>不</b>`	粗体	
`<color=green>羡慕</color>`	颜色	
`<size=50>大部分</size>都未受到影响`	字体大小	

Canvas

Canvas组件自带有三个组件，分别是Canvas、Canvas Scaler、Graphic Raycaster组件

Canvas组件

Screen Space-Overlay —— 屏幕空间覆盖模式

表示不管有没有相机去渲染场景，Canvas下的所有UI永远位于屏幕的前面，覆盖掉渲染场景显示的元素。

属性	功能
Pixel Perfect	使UI元素像素对应，效果就是边缘清晰不模糊
Sort Order	多个Canvas时，数值越大越后渲染。值大的 画布，会挡住值小的
Target Display	目标显示器，如果有多个屏幕的话可以选择
Addtional Shader Channels	附加着色通道，决定Shader可以读取哪些相关数据，比如法线、 切线 等数据。

Screen Space-Camera —— 相机模式

这种渲染模式 适用于场景模型太多太大，在调整UI的时候挡住UI，让UI和渲染的相机移动到比较远的位置，就可以避免遮挡。并且Canvas 和 摄像机之间有一定的距离，可以在摄像机和 Canvas之间放置一些模型或粒子特效。

属性	功能
Pixel Perfect	使UI元素像素对应，效果就是边缘清晰不模糊
Render Camera	渲染的相机
Plane Distance	Canvas与相机之间的距离
Sorting Layer	画布的深度,指定了相机的渲染顺序
Order In Layer	值越大，该UI越显示在前面

World Space —— 世界模式

属性	功能
Event Camera	响应事件的相机
Sorting Layer	画布的深度,指定了相机的渲染顺序
Order in Layer	值越大，该UI越显示在前面
Addtional Shader Channels	附加着色通道，决定Shader可以读取哪些相关数据，比如法线、 切线 等数据

Canvas Scaler组件

控制UI画布的放大缩放的比例

Constant Pixer Size —— 恒定像素

这种模式下 UI以像素为大小，同样的像素在不同的分辨率下尺寸不一样

属性	功能
Scale Factor	缩放因子
Reference Pixels Per Uit	单位面积像素数量

Scale With Screen Size —— 屏幕尺寸比例

这种缩放模式下的UI位置是根据屏幕的分辨率和设置的宽高比来调整UI的位置的，通常做屏幕UI自适应的时候都需要调整到这个缩放模式下。

属性	功能
Referencee Resolution	预设屏幕大小
Screen Match Mode	缩放模式
Match	宽高比

Constant Physical Size —— 恒定尺寸

属性	功能
Physical Unit	使用单位
Fallback Screen DPI	备用屏幕的DPI
Default Sprite DPI	默认图片的DPI
Reference Pixels Per Uit	单位面积像素数量

Graphic Raycaster组件

控制是否让UI响应射线点击

属性	功能
Ignore Reversed Graphic	忽略反转的UI，UI反转后点击无效。
Blocking Objects	阻挡点击物体，当UI前有物体时，点击前面的物体射线会被阻挡。
Blocking Mask	阻挡层级，当UI前有设置的层级时，点击前面的物体射线会被阻挡。

EventSystem

Canvas一同创建的还有一个EventSystem，这是一个基于Input的事件系统，可以对键盘、触摸、鼠标、自定义输入进行处理。

Event System组件

Event System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Event System组件。

属性	介绍
First Selected	首选对象
Send Navigation Events	发送导航事件
Drag Threshold	拖动阈值

Standalone Input Module组件

处理输入的鼠标或触摸事件，进行事件的分发。

属性	介绍
Horizontal Axis	横轴
Vertical Axis	纵轴
Submit Button	提交按钮
Canvel Button	取消按钮
Input Actions Per Second	每秒输入动作
Repeat Delay	重复延迟
Force Module Active	力模块激活

Button组件

Transition 过渡

点击按钮时候的变化

Color Tint —— 颜色过渡

属性	介绍
Interactable	是否启动按钮的响应
Transition	按钮的过渡动画类型，有Color Tint颜色过渡、Sprite Swap图片过渡、Animation动画过渡
Target Graphic	目标图形
Normal Color	普通状态下的颜色
Highlighted Color	鼠标悬停时状态下的颜色
Pressed Color	点击状态的颜色
Disabled Color	禁用状态的颜色
Color Multiplier	颜色乘数
Fade Duration	效果消失的时间
Navigation	导航类型
OnClick	点击事件列表

事件绑定

1. 直接在inspector面板绑定
2. 监听器绑定

```
using UnityEngine;
using UnityEngine.UI;

public class ButtonTest : MonoBehaviour
{
    public Button m_Button;
    public Text m_Text;
    void Start()
    {

 m_Button.onClick.AddListener(ButtonOnClickEvent);
    }
    public void ButtonOnClickEvent()
    {
        m_Text.text = "鼠标点击";
    }
}

```

3. 射线检测监听

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class ButtonTest : MonoBehaviour
{
    public Text m_Text;

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            if (OnePointColliderObject() != null)
            {
                if (OnePointColliderObject().name == "Button" || OnePointColliderObject().name == "Text")
                {
                    ButtonOnClickEvent();
                }
            }
        }
    }

    //点击对象获取到对象的名字
    public GameObject OnePointColliderObject()
    {
        //存有鼠标或者触摸数据的对象
        PointerEventData eventDataCurrentPosition = new PointerEventData(EventSystem.current);
        //当前指针位置
        eventDataCurrentPosition.position = new Vector2(Input.mousePosition.x, Input.mousePosition.y);
        //射线命中之后的反馈数据
        List<RaycastResult> results = new List<RaycastResult>();
        //投射一条光线并返回所有碰撞

        EventSystem.current.RaycastAll(eventDataCurrentPosition, results);
        //返回点击到的物体
        if (results.Count > 0)
            return results[0].gameObject;
        else
            return null;
    }

    public void ButtonOnClickEvent()
    {
        m_Text.text = "鼠标点击";
    }
}
```

4. 通过 EventTrigger 实现按钮点击事件

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

[RequireComponent(typeof(EventTrigger))]
public class ButtonTest : MonoBehaviour
{
    public Text m_Text;

    void Start()
    {
        Button btn = transform.GetComponent<Button>();
        EventTrigger trigger =
btn.gameObject.GetComponent<EventTrigger>();
        EventTrigger.Entry entry = new
EventTrigger.Entry
        {
            // 鼠标点击事件
            eventID = EventTriggerType.PointerClick,
            // 鼠标进入事件 entry.eventID =
EventTriggerType.PointerEnter;
            // 鼠标滑出事件 entry.eventID =
EventTriggerType.PointerExit;
            callback = new EventTrigger.TriggerEvent()
        };
        entry.callback.AddListener(ButtonOnClickEvent);
        // entry.callback.AddListener (OnMouseEnter);
        trigger.triggers.Add(entry);
    }

    public void ButtonOnClickEvent(BaseEventData
pointData)
    {
        m_Text.text = "鼠标点击";
    }
}

```

5. 通过通用类 UIEventListener 来处理Button响应事件

```
using UnityEngine;
using UnityEngine.EventSystems;

public class UIEventListener : MonoBehaviour,
IPointerClickHandler
{
    // 定义事件代理
    public delegate void UIEventProxy();
    // 鼠标点击事件
    public event UIEventProxy OnClick;

    public void OnPointerClick(PointerEventData
eventData)
    {
        if (OnClick != null)
            OnClick();
    }
```

```
}

```

从

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

[RequireComponent(typeof(EventTrigger))]
public class ButtonTest : MonoBehaviour
{
    public Text m_Text;

    void Start()
    {
        Button btn = this.GetComponent<Button>();
        UIEventListener btnListener =
btn.gameObject.AddComponent<UIEventListener>();

        btnListener.OnClick += delegate () {
            ButtonOnClickEvent();
        };
    }

    public void ButtonOnClickEvent()
    {
        m_Text.text = "鼠标点击";
    }
}

```

Text组件

属性	说明
Text	用于显示的文本
Font	文本的字体
Font Style	文本的样式（正常、加粗、斜线）
Font Size	字体的大小
Line Spacing	文本行之间的间距
Rich Text	是否支持富文本，富文本是带有标记标签的文本，增强文本的显示效果
Alignment	文本的水平和垂直对齐方式
Align By Geometry	是否以当前所显示的文字中获得的最大长宽（而不是字体的长宽）进行对齐。
Horizontal Overflow	文字横向溢出处理方式，可以选择Warp隐藏或者Overflow溢出
Vertical Overflow	文本纵向溢出的处理方式，可以选择Truncate截断或者Overflow溢出
Best Fit	忽略Font Size设置的文字大小，自适应改变文字大小以适应文本框的大小
Color	文本的颜色
Material	用来渲染文本的材质，可以通过设置材质，让文本拥有更加炫酷的效果。
Raycast Target	是否可以被射线检测，通常情况下可以关闭，因为文本最好只用来显示。

Toggle

接口

接口	说明
IPointerEnterHandler - OnPointerEnter	当指针进入对象时调用
IPointerExitHandler - OnPointerExit	当指针退出对象时调用
IPointerDownHandler - OnPointerDown	当指针压在对象上时调用
IPointerUpHandler - OnPointerUp	当指针被释放时调用(在原始按下的对象上调用)
IPointerClickHandler - OnPointerClick	当在同一对象上按下和释放指针时调用
IInitializePotentialDragHandler - OnInitializePotentialDrag	在找到拖动目标时调用，可用于初始化值
IBeginDragHandler - OnBeginDrag	当拖动即将开始时，在拖动对象上调用
IDragHandler - OnDrag	当发生拖动时在拖动对象上调用
IEndDragHandler - OnEndDrag	当拖动完成时在拖动对象上调用
IDropHandler - OnDrop	在拖动完成时对对象调用
IScrollHandler - OnScroll	当鼠标滚轮滚动时调用
IUpdateSelectedHandler - OnUpdateSelected	在选定的对象上调用
ISelectHandler - OnSelect	当对象变成选定对象时调用
IDeselectHandler - OnDeselect	被选中的对象被取消选中
IMoveHandler - OnMove	当移动事件发生时调用(左、右、上、下等)
ISubmitHandler - OnSubmit	在按下提交按钮时调用
ICancelHandler - OnCancel	按下取消按钮时调用

粒子系统

属性	说明
Duration	粒子运行时间，若勾起Looping则没有效果
Looping	是否循环播放
Prewarm	是否立即播放。选了这个就不能选Start Delay
Start Delay	粒子的播放延迟时间
Start Lifetime	粒子的生命周期，可以理解为单个粒子的存在时间
Start Speed	粒子的速度，就是单个粒子发射速度
3D Start Size	可以从xyz来调整粒子大小
Start Size	等比例缩放粒子大小

3D Start Rotation:3D旋转，勾选后可以用XYZ调节粒子的旋转
Start Rotation：开始的旋转，勾选后可从整体调整粒子旋转。
Flip Rotation:镜像翻转。
Start Color:开始时的颜色，可以调节
Gravity Modifier:受重力影响的大小，越大粒子下落越快，负数则粒子上升越快。可以用做喷泉等。
Simulation Space：粒子系统在自身坐标系还是世界坐标系。如果是粒子之间的互动，用局部空间。如果是物体来影响粒子，用世界空间。
Simulation Speed:模拟速度，根据Update模拟的速度。
Delta Time:主要用于暂停菜单的粒子系统。
Scaling Mode:缩放比例，三个选项：Hierarachy:当前粒子大小会受到上一级对象的缩放影响、Local:只跟自身大小有关、Shape:跟发射器有关系。
Play On Awake:点击Play时是否运行。
Emitter Velocity:发射器速度
Max Particles:最大粒子数，就是游戏内存在的最大粒子数量
Auto Random Seed:粒子随机，启用后每次播放都会有不同。
Stop Action:当属于系统的所有粒子都已完成时，可使系统执行某种操作。当一个系统的所有粒子都已死亡，并且系统存活时间已超过 Duration 设定的值时，判定该系统已停止。对于循环系统，只有在通过脚本停止系统时才会发生这种情况。
Disable:禁用游戏对象
Destory：销毁游戏对象
Callback：将 OnParticleSystemStopped 回调发送给附加到游戏对象的任何脚本。

场景系统

同步加载场景

异步加载场景

```
private AsyncOperation ao;
    void Start()
    {
        SceneManager.LoadScene(1);
        SceneManager.LoadScene("TriggerTest");
        SceneManager.LoadScene(2);
        SceneManager.LoadSceneAsync(2);
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            SceneManager.LoadSceneAsync(2);
            StartCoroutine(LoadNextAsyncScene());
        }
```

```
        if (Input.anyKeyDown&&ao.progress>=0.9f)
        {
            ao.allowSceneActivation = true;
        }
    }

    IEnumerator LoadNextAsyncScene()
    {
        ao= SceneManager.LoadSceneAsync(2);
        ao.allowSceneActivation = false;
        while (ao.progress<0.9f)
        {
         // 当前场景加载进度小于0.9
         // 当前场景挂起，一直加载，直到加载基本完成
            yield return null;
        }
        Debug.Log("按下任意键继续游戏");
    }
```

物理系统

Player settings：Queries Hit Triggers，取消鼠标和触发器的碰撞

射线检测

```
 private Collider2D collider;
void Start()
{
    collider = GetComponent<Collider2D>();
}
void Update()
{

            collider.enabled = false;
            RaycastHit2D hitTest = Physics2D.Linecast(start位置,end位置);
            collider.enabled = true;
}
```

```
 RaycastHit2D hit = Physics2D.Raycast(rigidbody2d.position+

Vector2.up*0.2f,lookDirection,1.5f,LayerMask.GetMask("NPC"));
```

2D碰撞器

box collider 2D

产生碰撞的条件：

双方都挂载碰撞器，运动的一方挂载刚体

旋转与抖动问题

使用默认的组件只会会发生抖动与旋转的问题

1. 在rigidbody 2d中把constrain的z轴冻结
2. 会发生抖动是因为自己写的代码直接控制了 transform.position
   但是这个位置会直接移动进入刚体内部，因此会被引擎拉出去。
   因此为了防止这个bug，可以直接操纵刚体的位置

```
rigidbody2D.position = position;
```

触发器和碰撞器的区别

触发器只会进行触发而不会进行物理碰撞

1. 双方挂载collider
2. 一方挂载刚体
3. 一方把is Trigger打开
4. 写代码

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.GetComponent<RubyController>
()!=null)
        collision.GetComponent<RubyController>
().changeHP(-1);
    }
```

碰撞器会进行物理碰撞

1. **两物都有Collider**
2. **至少有一个带有RigidBody**
3. 写代码

```
 void OnCollisionEnter2D(Collision2D collision)
    {
        Debug.Log("OnCollisionEnter2D:" +
collision.transform.name);
    }
```

防止人物和子弹之间碰撞

1. 在右边设置层级
2. 自定义一个层级，子弹和人物分别占据不同的层级
3. 在设置中找到physics 2d，拉到最下面，把对应的勾取消就行

注意，碰撞器是Collision2D而触发器是Collider2D

Unity 3D 中的碰撞体和触发器的区别在于：碰撞体是触发器的载体，而触发器只是碰撞体的一个属性。

碰撞和触发检测

```
private void OnCollisionEnter2D(Collision2D collision)
  {
      // 碰撞到的游戏物体名字
      Debug.Log(collision.gameObject.name);
  }

  private void OnCollisionStay2D(Collision2D collision)
  {
      Debug.Log("在碰撞器里");
  }

  private void OnCollisionExit2D(Collision2D collision)
  {
      Debug.Log("从碰撞器里移出");
  }

  private void OnTriggerEnter2D(Collider2D collision)
  {
      // 碰撞到的游戏物体名字
     Debug.Log(collision.gameObject.name);
  }

  rivate void OnTriggerStay2D(Collider2D collision)
  {
      Debug.Log("在触发器里");
  }

  private void OnTriggerExit2D(Collider2D collision)
  {
      Debug.Log("从触发器里移出");
  }
```

时间系统

暂停游戏

Time.timescale = 0

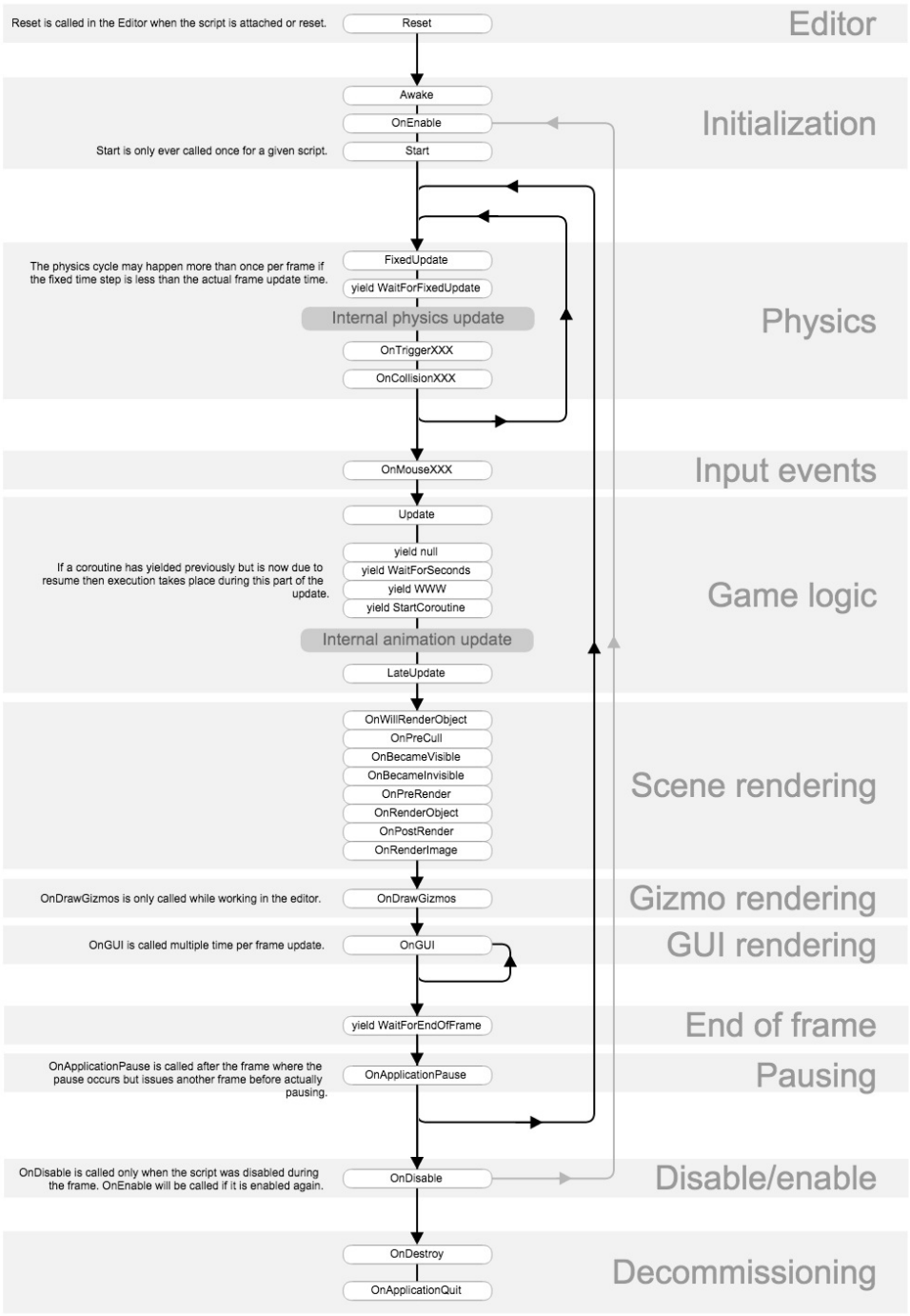
Time.timescale = 1

unity脚本的生命周期

生命周期图

unity脚本是挂在在游戏物体上的，物体有生命周期，脚本当然也有了。

unity脚本的生命周期是unity开发中必须掌握的重要内容。


Reset is called in the Editor when the script is attached or reset.
Reset
Editor
Awake
OnEnable
Start is only ever called once for a given script.
Start
Initialization
The physics cycle may happen more than once per frame if the fixed time step is less than the actual frame update time.
FixedUpdate
yield WaitForFixedUpdate
Internal physics update
OnTriggerXXX
OnCollisionXXX
Physics
OnMouseXXX
Input events
Update
If a coroutine has yielded previously but is now due to resume then execution takes place during this part of the update.
yield null
yield WaitForSeconds
yield WWW
yield StartCoroutine
Internal animation update
LateUpdate
Game logic
OnWillRenderObject
OnPreCull
OnBecameVisible
OnBecameInvisible
OnPreRender
OnRenderObject
OnPostRender
OnRenderImage
Scene rendering
OnDrawGizmos is only called while working in the editor.
OnDrawGizmos
Gizmo rendering
OnGUI is called multiple time per frame update.
OnGUI
GUI rendering
yield WaitForEndOfFrame
End of frame
OnApplicationPause is called after the frame where the pause occurs but issues another frame before actually pausing.
OnApplicationPause
Pausing
OnDisable is called only when the script was disabled during the frame. OnEnable will be called if it is enabled again.
OnDisable
Disable/enable
OnDestroy
OnApplicationQuit
Decommissioning

生命周期函数

函数名	调用时机
Reset	当Scripts第一次绑定 到物体上或者点击Reset按钮的时候会触发， 且只在Editor的模式下触发，游戏打包的时候并不会触发
Awake	当脚本实例在游戏运行被载入的时候运行，一般为了初始化游戏变量和游戏状态，注意，无论函数是否被激活，Awake都会执行!!
OnEnable	每当启用脚本时调用
Start	游戏对象调用脚本时执行
FixedUpdate	它是物理循环中最先执行的函数，它每隔固定时间执行一次。我们通常与把物理模拟有关的代码，写在FixedUpdate 函数中，比如给游戏对象添加力。默认0.02s执行一次。
OnTrigger	
OnCollision	
OnMouseXXX	有关鼠标的函数就在此刻执行。
Update	它在每帧执行一次，该函数主要处理游戏对象在游戏世界的行为逻辑，例如游戏角色的控制和游戏状态的控制.
LateUpdate	它也是每帧执行一次，在 Update 函数后执行。 在实际开发过程中 Update 函数与 LateUpdate 函数通常共同使用。一般我们在 Update 函数中处理玩家角色的移动，在 LateUpdate 函数中处理摄像机跟随玩家，这样能防止摄像机出现抖动现象。因为，如果我们把这两个都写在Update中，有可能造成摄像机先跟随，玩家后移动的逻辑bug。
OnBecameVisible	
OnBecameInvisible	当物体在任何相机中可见/不可见时调用。注意：Scene视图的相机也需要考虑进去。
OnGUI	一帧会调用多次来响应GUI事件。
OnDestroy	脚本或者脚本挂载的游戏对象销毁时，在对象存在的最后一帧调用。
OnApplicationQuit	应用退出时所有的游戏物体将会调用此函数

Awake和Start区别

各个脚本的awake乱序执行，而start

变量初始化的顺序问题

顺序如下，首先是初始化时赋值，然后再外部编辑器，之后以此类推

```
public int a = 0;

//外部编辑器赋值，也就是inspector面板

private void Awake()
{
    a = 1;
}

private void OnEnable()
{
    a = 2;
}

void Start()
{
    a = 3;
}

```

OnMouseEventFunction鼠标回调事件

```
 private void OnMouseDown()
    {
        print("在Gris身上按下了鼠标");
    }

    private void OnMouseUp()
    {
        print("在Gris身上按下的鼠标抬起了");
    }

    private void OnMouseDrag()
    {
        print("在Gris身上用鼠标进行了拖拽操作");
    }

    private void OnMouseEnter()
    {
        print("鼠标移入了Gris");
    }

    private void OnMouseExit()
    {
        print("鼠标移出了Gris");
    }
```

```
25
26      private void OnMouseOver()
27      {
28          print("鼠标悬停在了Gris上方");
29      }
30
31      private void OnMouseUpAsButton()
32      {
33          print("鼠标在Gris身上松开了");
34      }
```

Unity特性

修改inspector

Header

```
1  [Header("与物体交互时的信息")]
2  public string[] text;
```

###

修改editor

代码	代码	特性描述
[SerializeField]		序列化一个类，实际用处是把数据存储到硬盘上，表面用处可以把私有的在检视面上显示出来
[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.AfterSceneLoad)]		无需创建空物体就能执行代码
[HideInInspector]		隐藏公有变量
		在Inspector中重命名一个变量
[Range(minnum,maxNum)]		给一个数值变量添加滑块
[TextArea（最小行数，最大行数）]		文本框扩大， 超过最大行数会出现滚动条
[ContextMenu("执行函数")]		为挂载脚本的物体的Inspector界面脚本添加一个脚本右键选项,点击执行方法内的逻辑

##

增加标题

```
[Header("xxxxx")]
```

人工智能

群组行为

模拟鸟群飞行或者人群行走过程称之为群组行为

分离 队列 聚集

小技巧

scriptableobject类

浅拷贝来避免开销

使用TryGetComponent

对于

使用本地函数localFunction

对于一些只在函数内部调用的函数，可以写成本地函数。

```
public void Test()
{
    var attack = RandamNum();
    var defense = RandamNum();
    int RandamNum()
    {
       return Mathf.Rnage(10,20);
    }
}
```

多个Awake之间的顺序

方法一：手动调整Scripts的执行顺序

在Unity菜单中，选择Edit>Project Settings>Script Execution Order,可以添加并且调整Scripts的手动顺序。可以参考官方文档。

方法二：使用 [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]

建立程序集

Create Assembly Defintion，然后跟他同一个文件夹的脚本都会成为该程序集。

多用try

- 对齐摄像机到视口

选中摄像机，GameObject选择algin with view

-