1

U3D入门

1.脚本

U3D目前支持三种语言的脚本程序，C#，Javascript，Boo，在一个游戏中，开发者可以使用一种或者同时使用多种语言来实现脚本控制。

脚本注意事项：

类名要和cs名称一致，所有类继承自MonoBehavior

项目运行过程中的修改不会保存

脚本只能依附于游戏对象或者由其他脚本调用才会运行，一个脚本可以放到多个游戏对象上并且互不干扰。

2.GameObject对象

游戏对象：GameObject，是指3D场景中所存在的所有物体，包括建筑，角色，道具等，当然除了我们在场景中可见的物体外，还存在着一些不可见的游戏对象，例如光源，音源等都属于游戏对象。

GameObject拥有Tag(标签)，Layer(层)和Name(名称)属性。场景中所有的游戏对象都是通过实例化该类来生成的。当把一个游戏资源放入场景中后，U3D就会自动通过GameObject类来生成对应的游戏对象。

任何一个游戏对象都可以同时绑定多条游戏脚本，并且这些脚本互不干扰，各自完成各自的生命周期。

查找游戏对象：

(1).GameObject.Find(string name)按名字查找

找到并返回一个名字为name的游戏物体，如果name为名字的游戏物体没有被找到则返回空。如果name中包含'/'字符，这个名字将被视作一个hierarchy中的路径名。这个函数只返回活动的物体。

⚠不要在每一帧中使用该函数，可以在开始的时候用一个成员变量来缓存结果

```
GameObject capsule;
Void start(){
capsule = GameObject.Find("Capsule");
}
```

(2).GameObject.FindWithTag(string tag)按标签查找

返回一个用tag做标识的活动的游戏对象，如果没有找到则为空。标签必须在使用之前已在标签管理器中声明。

⚠GameObject和gameobject的区别

GameObject是一个类型，所有的游戏物体都是这个类的对象

gameobject是一个对象，指的是这个脚本所附着的游戏物体

3.粒子系统

4.第一人称控制器：

U3D自带第一人称控制器，以第一人称的视角来观察周围环境，只需要导入角色包就可以使用第一人称控制器。

父级(PFSController)：

Character Controller(角色控制器)

First Person Controller(脚本)

rigidbody(刚体)

Audio Source(音频器)

子级(FirstPersonCharacter)：

Camera(摄像机)

Audio Listener(监听器)

Flare Layer(耀斑层)

5.相机

主摄像机层： 0

2

第一人称摄像机:当depth>0时，则运行时看到的为第一人称摄像机的视角(摄像机的depth越大说明越后渲染)

6.制作小地图

Render Texture:是可以被渲染的纹理。它可以用来实现基于渲染效果的图像，动态阴影，投影，反射或者监视摄像机。渲染纹理的典型用法之一是:设置它们为摄像机的"目标纹理"属性(Camera.targetTexture)。该属性将使得相机渲染到纹理而不是渲染到屏幕。

步骤:

(1).创建camera垂直照射地形(上帝视角)

(2).UI-RawImage图片:承载小地图(将渲染出的图片放在rawimage上)

(3).创建RenderTexture(渲染纹理)

(4).将RenderTexture拖到camera的TargetTexture(将相机渲染到的画面作为纹理)

(5).将RenderTexture拖到RawImage的Texture上

7.光源

Directional Light(方向光)： 类似于太阳光，最不耗费图形处理器资源的光源，也是最省内存的。

Point Light(点光源)： 从中间发光，在球形范围内有光。

Spot Light(聚光灯)： 类似于射灯，在某一方向锥形范围内有效。耗费图形处理器资源。

Area Light(区域光)： 不用于实时光照，只用于光照贴图烘焙，很少用。

8.地形

GameObject-3DObject-Terrain

+alt:四周旋转，观察

不按shift:抬高地形

+shift:降低地形

关闭烘焙(消耗资源):Window-Rendering-Lidhting Setting-取消勾选auto generate

小刷子添加纹理:要导入Unity自带的资源包(Assets-Import Packages-Environement -点击Edit Texture-add Texture导入需要的纹理)

添加树木和草地(草地在远处(Detail Distance)无法看到)一样的道理

+shift:去除树木和草

⚠

Opacity为0时绘制不出来纹理

Target Strength为0时，是相反绘制

9.物理引擎-刚体

刚体组建可以使游戏对象在物理系统的控制下来运动，刚体可以接受外力与阻力矩用来保证游戏对象在真实世界那样运动

添加Rigidbody组建到一个对象，这就将其置于U3D的物理引擎下使其运动，即使没有添加任何代码，刚体对象也将在重力的影响下下落，如果有一个带有碰撞器的对象与之碰撞，将作出反应。

Mass:质量

Drag:阻力

内差值，外插值:在动画中对移动位置进行计算

Freeze Position:比如物体落下来后，不会翻转，直直落下来(x,y,z固定)

```
public class Test : MonoBehaviour {
    //首先获取刚体:
    public Rigidbody rg;
    public float thrust;
```

3

```
    private void Start()
    {
        rg = GetComponent<Rigidbody>();
    }

    //当按下空格键时，给物体添加力
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) {
            rg.AddForce(transform.forward * thrust);
        }
    }
}
```

10.物理引擎-碰撞器

Collider(碰撞器):它是所有碰撞器的基类。

isTrigger如果勾选，碰撞器将是一个触发器

有关碰撞器的三个函数:

(1).void OnCollisionEnter(Collision collisionInfo)

(2).void OnCollisionStay(Collision collisionInfo)

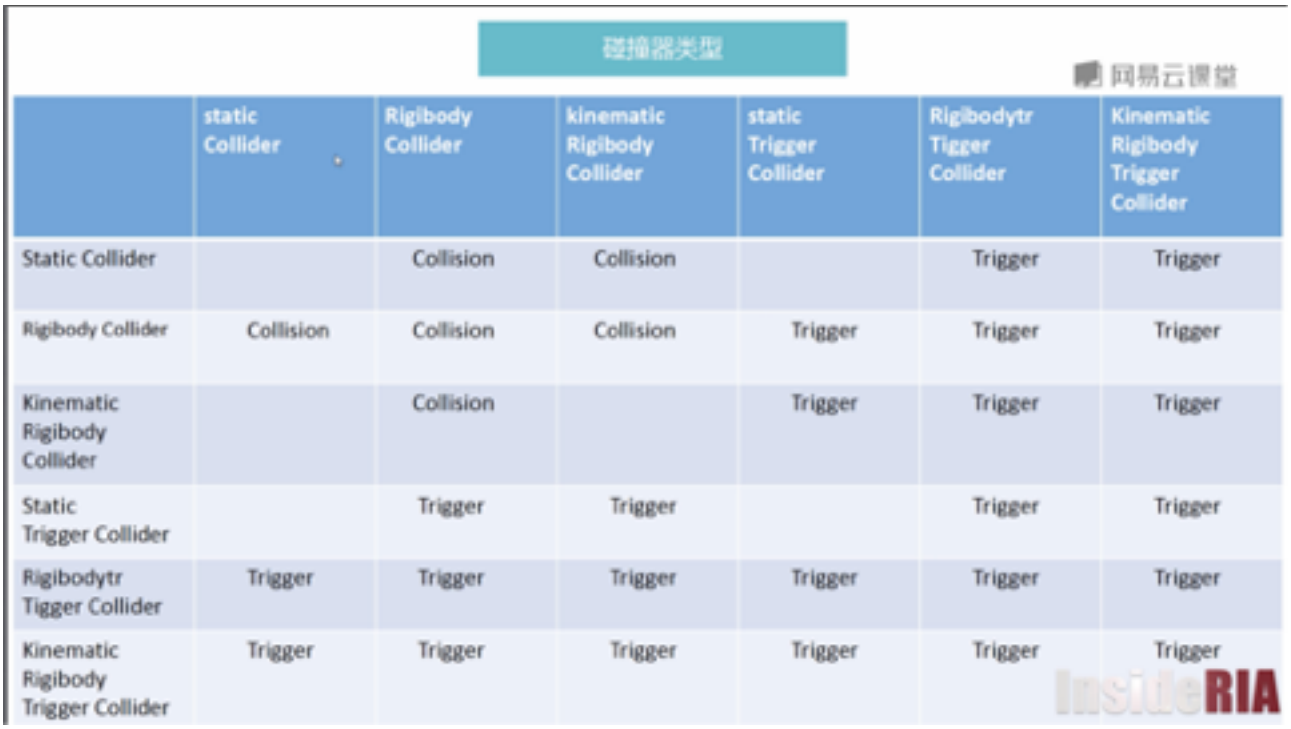
(3).void OnCollisionExit(Collision collisionInfo)

有关触发器的三个函数:

(1).void OnTriggerEnter(Collision collisionInfo)

(2).void OnTriggerStay(Collision collisionInfo)

(3).void OnTriggerExit(Collision collisionInfo)



	static Collider	Rigibody Collider	kinematic Rigibody Collider	static Trigger Collider	Rigibodytr Tigger Collider	Kinematic Rigibody Trigger Collider
Static Collider		Collision	Collision		Trigger	Trigger
Rigibody Collider	Collision	Collision	Collision	Trigger	Trigger	Trigger
Kinematic Rigibody Collider		Collision		Trigger	Trigger	Trigger
Static Trigger Collider		Trigger	Trigger		Trigger	Trigger
Rigibodytr Tigger Collider	Trigger	Trigger	Trigger	Trigger	Trigger	Trigger
Kinematic Rigibody Trigger Collider	Trigger	Trigger	Trigger	Trigger	Trigger	Trigger

eg:碰撞(Collision):发生碰撞

触发(Collider):穿过物体

4

进入碰撞(触发)和退出碰撞(触发):调用一次
逗留碰撞(触发):调用多次

创建两个Cube，添加碰撞盒，刚体

```
public class Test : MonoBehaviour {
    void OnCollisionEnter(Collision collision)
    {
        Debug.Log("进入碰撞");
    }

    void OnTriggerEnter(Collider collider)
    {
        Debug.Log("进入触发");
    }
}
```

11.物理引擎-射线(多用于检测碰撞)
射线(Ray):射线是3D世界中一个点像一个方向发射的一条无终点的线，在发射过程中与其他物体发生碰撞时，它将停止发射
用途:射线应用较广，多用于碰撞检测(eg:子弹飞行后是否击中目标)，角色移动等
eg:射线的一端从摄像机发出，另一端随着鼠标的移动而移动

```
public class RawTest : MonoBehaviour {
    private void Update()
    {
        //屏幕一点转化为射线， 参数是要射到哪里的位置
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        //将射线显示出来，射线的起始点(Camera的位置)和方向
        Debug.DrawRay(transform.position, ray.direction, Color.red);
    }
}
```

5

eg：视窗(Game窗口)转射线

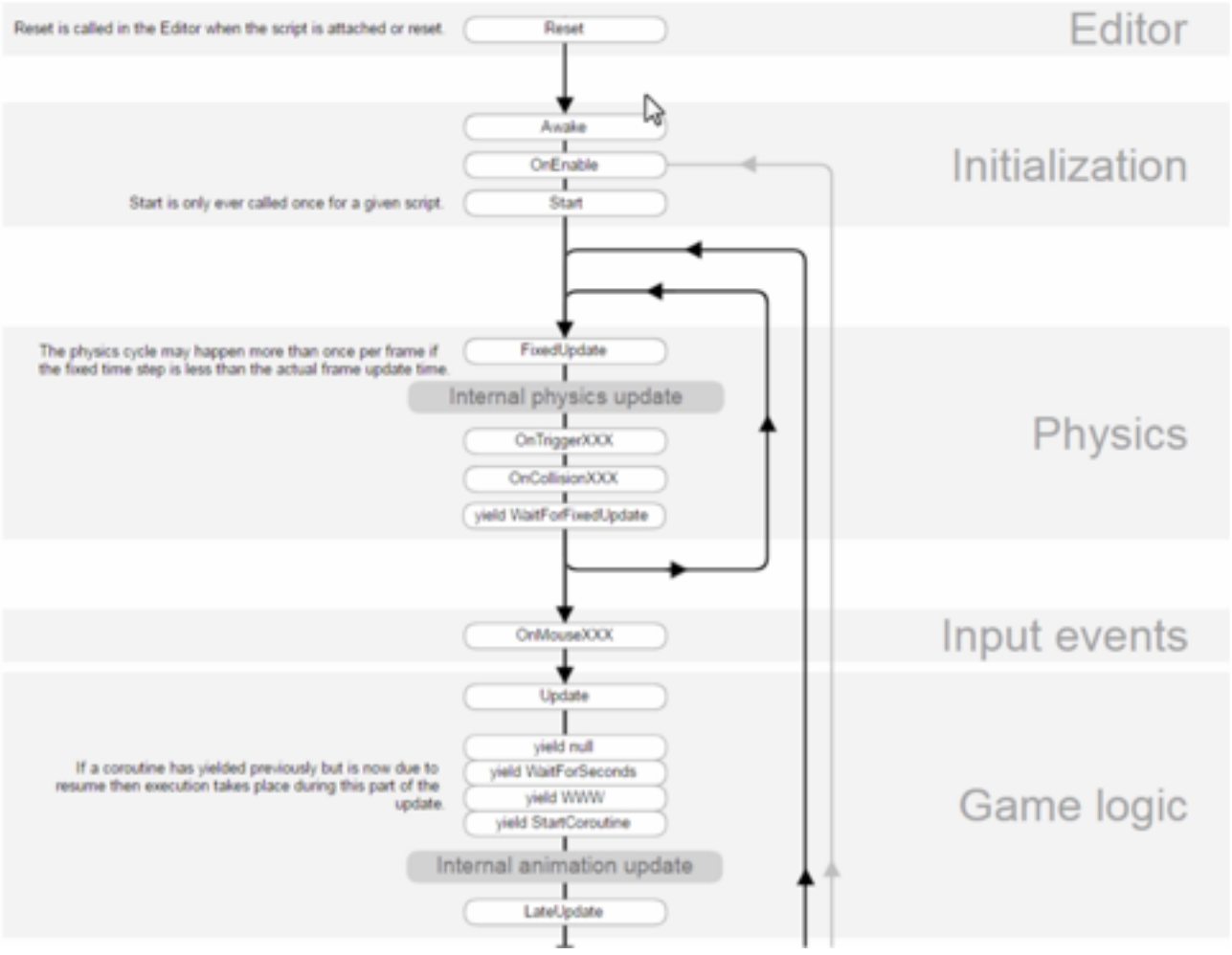
如果视窗看到了对象；就返回对象的名字

```
public class RawTest : MonoBehaviour {
    private void Update()
    {
        Ray ray = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
        Debug.DrawRay(transform.position, ray.direction, Color.red);
        //如果检测到了对象
        //hit指所碰撞到的对象(RaycastHit类型)
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit)) {
            Debug.Log("我们看到了:" + hit.transform.name);
        }
        else {
            Debug.Log("没有看到任何东西");
        }
    }
}
```

12.脚本生命周期(Start, Update方法对客户可见)

每个游戏组件的脚本有一个生命周期。

从开始实例化，直到结束实例被销毁，在这期间，他们有时候处于激活状态，有时候处于非激活状态，对于激活状态，对用户有时候可见，有时候不可见。



6

常用顺序：

Reset

Awake

OnEnable

Start

FixedUpdate

Update

LateUpdate

OnDisable

OnDestroy

Reset：只有在编辑模式下才能够被调用

Awake：当脚本实力被载入时(初始化变量或者初始化游戏状态)，Awake方法被调用。只调用一次。

如果每个对象上都挂有脚本而且每个脚本都有Awake方法，则Awake调用顺序是随机的。

```
public class ScriptOrder : MonoBehaviour {
    void Reset()
    {
        print("Reset方法被调用");
    }

    void Awake()
    {
        print("Awake方法被调用");
    }

    void Start() {
        print("Start方法被调用");
    }
}
```

三个Update区别：

(1).Update：和机器性能有关，时快时慢

(2).FixedUpdate：0.02秒(固定帧更新)，用于处理物理行为，比如给刚体添加力，则需要每0.02秒添加一个力给物体。

(3).FixedUpdate之后会执行碰撞触发等方法(Update之前)。

OnEnable：对象被激活

OnDisable：对象被禁用时(eg：禁用一个物体)

OnDestroy：预先已经被激活的物体，即将被销毁时

13.输入管理器

Edit-Project Setting-Input

```
public class Test : MonoBehaviour {
    //通过输入管理使得物体移动
    //获取键，执行相应操作太麻烦，通过输入管理类获得水平轴和垂直轴
    void Update()
    {
```

```
        //horizontal返回float
        float h = Input.GetAxis("Horizontal");
        float v = Input.GetAxis("Vertical");

        //使物体运动
        this.transform.Translate(h, 0, v);
    }
}
```

14.Time类

Time.time:从游戏开始后开始计时，表示截止目前运行的游戏时间

Time.fixedTime:FixedUpdate()方法中固定消耗的时间总和

Time.deltaTime:获取Update()方法中完成上一帧所消耗的时间

Time.fixedDeltaTime:获取FixedUpdate()方法中完成上一帧所消耗的时间

eg: 添加速度时，经常用速度*deltaTime，这样可以让速度更平滑，不会有卡顿的现象

用UI现实Update和FixedUpdate一帧大概需要的时间:

```
using UnityEngine.UI;

public class TimeTest : MonoBehaviour {

    public Text text;
    //通过输入管理使得物体移动
    //获取键，执行相应操作太麻烦，通过输入管理类获得水平轴和垂直轴
    void Update()
    {
        //先获取text组件，再改变text文本信息，要把float转string
        text.GetComponent<Text>().text = "游戏运行时间:" + Time.time.ToString();
    }
}
```

⚠:不仅要把脚本挂在GameObject上，还要把UI挂在GameObject上

U3D进阶

1.寻路系统

加入AI的好处是提高游戏的可玩性，刺激玩家的挑战欲望，增加游戏的友好体验。

自动寻路是所有游戏的一个难点，属于AI(人工智能范畴)其算法非常复杂，U3D中提供了一套非常成熟的组件来为我们解决这一难题。

NavMesh(导航网格)是3D游戏世界中用于实现动态物体自动寻路的技术，将游戏中复杂的结构组织关系简化为带有一定信息的网络，在这些网络基础上通过一系列的计算实现自动寻路。

步骤:

(1).选择场景中需要生成寻路系统的几何体-可行走表面和障碍物。

(2).在NavMesh面板中选择需要烘焙寻路的物体，检测是否勾选Navigation Static。

(3).根据需要的agent调整bake面板的设置

Agent Radius: agent可以距离墙体，窗户或者边缘多近

Agent Height: agent可以通过的最低的空间高度

Max Slop: agent可以直接行走上去的最小坡度

Step Height: agent可以踩上(走上)的障碍物的最高高度

(4).点击bake按钮烘焙NacMesh

2.分层寻路

高级设置：

最小区域面积(Min Region Area)

Min Region Area允许你剔除掉小的非连接NavMesh区域，当NavMesh区域小于指定值时将被剔除

⚠有些区域可能无法剔除，尽管有Min Region Area的设置，原因是NavMesh的构建是一个个的网格。当一个区域连跨两个网格时将不会被移除，因为区域修建过程中无法获取到周围的网格。

Voxel Size立体像素尺寸

Manual Size:允许你改变烘焙操作过程中的精确性

Navigation在构建寻路网格过程中，第一遍会把场景光栅化为像素，然后提取可行走区域，最后可行走区域会被烘焙成网格。因此Voxel Size决定了寻路网格烘焙的精确性。

⚠在烘焙场景中Voxel Size增加会造成4x倍的内存消耗和4x倍的时间消耗。因此通常不要自己去设置Voxel Size。

eg:简易搭建两个区域A，B

有三座桥，颜色分别为黄，蓝，红

A区域有三个cube，颜色分别为黄，蓝，红

B区域有一个目标对象Target

使得A区域的三个对象分别按照自身颜色的桥来到达目的地点

(1).搭建相应物体场景

(2).将可以走的路进行烘焙(选择Bridge下的Cube,Cube(1),RR,BR,YR)(选择Navigation Static,打开Navigation窗口，进行烘焙)

(3)设置层，在Arears中添加red,yellow,blue三个层

(4)给每一个桥在Object中设置对应的层

(5)给每一个对象方块添加Nav Mesh Agent,将Nav Mesh Agent中的Area Mask设置为自己能走的颜色，eg:红色方块去掉黄色和蓝色

(6).要想要物体向Target移动，则要给每个物体添加脚本

```
using UnityEngine.AI;
public class moveTest : MonoBehaviour {
    //定义目标
    GameObject target;
    private NavMeshAgent nav;
    // Use this for initialization
    void Start () {
        target = GameObject.Find("Target");
        nav = gameObject.GetComponent<NavMeshAgent>();
        nav.SetDestination(target.transform.position);
    }
    // Update is called once per frame
    void Update () {
    }
}
```

3.粒子系统

创建meterial，将meterial调节成激活的粒子系统，图片必须是png格式。将图片添加到粒子系统的渲染选项里面

4.UGUI

UI:User Interface

UGUI:是Unity推出的最新UI系统，被集成在Unity的编辑器中。优点:灵活，快速，可视化

Canvas:是Unity新UI系统里面的游戏对象，名叫"画布"，UI系统中所有的UI控件都应该被绘制在它的上面，也就是说所有的UI控件都是它的子对象。

*自适应功能(Rect Transform):当父对象大小发生变化时，锚点(Anchors)会令子对象自动地完成自适应的操作。eg:当调整panel大小时(Image锚点的位置会变化)，但Image的锚点的四个三角形距离Image四个角的距离是不变的。

*Render Mode:

(1).Overlay:canvas下的游戏组件绘制到游戏场景中的最上层

(2).Camera:最上层，有指定相机来保持和UI有固定的距离，即canvas下的所有UI控件都是由该camera进行绘制

(3).World Space:UI控件被绘制在了游戏场景中，是游戏场景中的一个对象，可以在游戏场景中看到。

*UI的绘制顺序:

canvas上面的会被先绘制也就是说靠后的有可能遮盖前面的，所以要调整顺序。

*在脚本中调节绘制顺序

SetAsFirstSibling():设置为同级第一

SetAsLastSibling():设置为同级最后

SetAsSiblingIndex():设置同级索引(用索引确定第几个被渲染)

5.UGUI控件_1

6.UGUI控件_2

7.UGUI控件_3

```
using UnityEngine.UI;

public class dropOptions : MonoBehaviour {

    public InputField name;
    public InputField paw;
    public InputField repa;
    public Toggle male;
    public Toggle formale;

    void Start() {

    }

    private void Update() {

    }

    bool IsSame() {
        string pawtext = paw.transform.FindChild("Text").GetComponent<Text>().te
xt;
        string repawtext = repaw.transform.FindChild("Text").GetComponent<Text>(
).text;
        if (paw.Equals(repawtext)) {
```

```
            return true;
        }
        else {
            return false;
        }
    }

    //按钮按下触发的方法(要在面板中设置)
    public void Submit()
    {
        if (IsSame()) {
            //打印出用户信息(用户名，密码，性别)
            //哪个isOn勾选上，说明哪一个选项被勾选
            string nameText = name.transform.FindChild("Text").GetComponent<Text
>().text;
            string pawtext = paw.transform.FindChild("Text").GetComponent<Text>(
).text;
            if (male.GetComponent<Toggle>().isOn) {
                Debug.Log(nameText + pawtext + male.GetComponentInChildren<Text>
().text);
            }
            else {
                Debug.Log(nameText + pawtext + formale.GetComponentInChildren<Te
xt>().text);
            }
        }
        else {
            Debug.Log("密码错误，请重试");
        }
    }
}
```

8.UGUI高级

```
using UnityEngine.UI;

public class dropOptions : MonoBehaviour {

    Dropdown dropdownItem;
    List<string> tempNames;

    void Awake() {
        //直接获取本对象的下拉菜单,所以要把脚本放在下拉菜单对象(Dropdown)上
        dropdownItem = GetComponent<Dropdown>();
        tempNames = new List<string>();
    }
```

```
// Use this for initialization
void Start () {
    AddNames();
    DropDownView(tempNames);
}

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

}

//自定义名称
void AddNames() {
    string str1 = "水蜜桃";
    string str2 = "苹果";
    string str3 = "榴莲";
    string str4 = "草莓";
    string str5 = "猕猴桃";
    string str6 = "山竹";

    tempNames.Add(str1);
    tempNames.Add(str2);
    tempNames.Add(str3);
    tempNames.Add(str4);
    tempNames.Add(str5);
    tempNames.Add(str6);
}

//将自定义的名称添加到列表中
void DropDownView(List<string> showNames) {
    //清除原先选项
    dropdownItem.options.Clear();
    //声明选项类型
    Dropdown.OptionData tempdata;
    for (int i = 0; i < showNames.Count(); ++i) {
        tempdata = new Dropdown.OptionData();
        tempdata.text = showNames[i];
        dropdownItem.options.Add(tempdata);
    }
    //一开始默认显示第一个选项
    dropdownItem.captionText.text = showNames[0];
}

//当value值发生变化时就会调用该方法(要在面板中进行设置)
```

```
    public void TextA() {
        Debug.Log("A");
    }
}
```

9.网格布局器

10.事件处理系统

11.2D游戏_1

png图片带有透明通道，也就是可以自动忽略背景。2D图片也是精灵图片，对于多个图片放在一张大图中，要进行切割。点击图片右边小三角，会出现切割完成的图片。之后可以利用这些图片制作动画。

保存后会出现动画状态机(Breath_0)。

12.2D游戏_2

Unity每次在准备数据并通知GPU渲染的过程称为一次Draw call。

一般情况下，渲染一次拥有一个网格并携带一种材质的物体便会使用一次Draw Call。

对于渲染场景中的这些物体，在每一次Draw Call中除了在通知GPU的渲染上比较耗时之外，切换材质与shader也是非常耗时的操作。

Draw Call的次数是决定性能比较重要的指标。

制作图集:精灵打包(节省时间和内存)

步骤:

(1).Edit->Projet Setting->editor->里面把Sprite Packer打开

(2).设置精灵图片的图集标签

(3).打开Sprite Packer窗口，点击Pack

13.2D刚体(类似于之前的3D刚体)

IsKinematic:控制刚体是否变为运动学物体，就是刚体不受物理引擎控制，而受transform或者是动画等等的控制。

Sleeping Mode:睡眠，unity会临时吧处于静止状态的物理模拟去除，这样主要是节省计算资源

(1).Never sleep:永远不睡眠

(2).Start awake:awake时就进入睡眠

(3).Start asleep:立即进入睡眠

CollisionDetection:碰撞发现

(1).Discrete:正常状态下的检测

(2).COntinue:连续性检测

2D碰撞器:

Edge Collider2D和Polygon Collider2D(可以编辑多边形碰撞器的外形)是3D里面没有的

刚体属性介绍:

Material:定义刚体表面的一些物理特性，例如摩擦力，柔软程度等，由此决定了对其他物体的反弹能力

Is Trigger:勾选则变成触发器

13.动画系统_1

获取动画的方式:自己创建，直接导入美工的动画

window->Animation->Animation->create->AddProperty->Transform/BoxCollider/MeshRenderer

双击创建关键帧，在左侧面板中设定位置

14.动画系统_2

点击动画状态->Read/Write Enabled:在动画运行过程中，可以改变网格(比如墙被子弹穿透，会留下一个子弹的印儿)

Rig->Legacy:旧动画系统

Generic:非人类骨骼

humanoid:类人骨骼

Animation->Clips(Start, End):动画剪辑

点击动画物体，设置默认动画

```
public class AnimationTest : MonoBehaviour {
    Animation ani;

    // Use this for initialization
    void Start () {
        ani = GetComponent<Animation>();
        ani.Play("jump");
    }

    // Update is called once per frame
    void Update ()
    {
        if (Input.GetKeyDown(KeyCode.S)) {
            ani.Stop();
        }
        if (Input.GetKeyDown(KeyCode.R)) {
            //淡入淡出，两个动画切换
            ani.CrossFade("run");
        }
    }
}
```

15.动画系统_3 Animator VS Animation

一定不要忘记更改旧动画模式

Animator是管理很多Amation的管理器，将动画的状态，跳转的条件进行处理

创建Animator->Animatior Controller:动画状态机

所以要创建动画状态机:Create->Animation Controller

16.动画系统_4 动画过渡

也就是简单的动画过渡控制，在Animator中能比较方便直观地修改。Transition就是所谓的过渡

Settings能调整过渡的时间和衔接

Conditions就是所谓的过渡条件

⚠️:如果一个动画有Conditions建议取消Has Exit Time，否则会出现无法及时触发的问题

Has Exit Time就是必须过渡完自身动画，才能过渡到下一个动画。

动画状态机橘黄色框:表示下一个要制作的动画

右键Make Transition

在Paramaters上右键会出现Float, Int, Bool, Trigger。

创建一个Bool条件(isDeath)

14

```
public class AnimationTest : MonoBehaviour {

    Animation ani;

    // Use this for initialization
    void Start () {
        ani = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update ()
    {
        if (Input.GetKeyDown(KeyCode.Space)) {
            //ani.SetBool(« isDeath », true);
            ani.SetTrigger("isDeath 0");
        }

        if (Input.GetButtonDown("Fire1")) {
            ani.SetFloat("IsRun", 1f);
        }

        if (Input.GetButtonUp("Fire1"))
        {
            ani.SetFloat("IsRun", 0.01f);
        }
    }
}
```

U3D高级

1.协程

普通程序调用协同程序的执行顺序：

开始协同程序->执行协同程序->中断协同程序->返回上层继续执行->中断时间结束后->执行协同程序剩下的内容

2.进程VS线程VS协程

进程:是正在运行的程序的实例

线程:进程内一个相对的，可调度的执行单元(每下载的一个任务都是一个线程)

协程:协作程序，一个既定任务

⚠:协程不是多线程

3.协程的作用:

(1).延时

(2).等待

4.一个类A实现了IEnumerator(枚举数)，也就是实现了Current属性，MoveNext方法，Reset方法，只要实现了这些方法，这个类A就可以用foreach这种语法了。

5.yield关键字用于遍历循环中

```
yield return
yield return 0
```

yield return null:"中断协同程序->等待下一帧从这里开始继续执行"

yield break用于终止循环遍历

```
6.void start(){
    foreach(int i in GetLessFive2){
        print(i);
    }
}

public static IEnumber<int> GetLessFive2(){
    foeach(int i in list){
        if(i < 5)
            yield return i;
        else
            tield break;
    }
}
7.void start(){
    StartCoroutine("SayHelloFiveTimes");
    StopCoroutine("SayHelloFiveTimes");
}

IEnumerator SayHelloFiveTimes(){
    while(true){
        Debug.Log("Hello");
        yield return 0;
    }
}
```

//结果是只打印了一次Hello

8.协同程序开启另一协同程序很特殊

```
public class AnimationTest : MonoBehaviour {

    Animation ani;

    void Start()
    {
        StartCoroutine("SaySomethings");
        Debug.Log("Start fini");
    }

    IEnumerator SaySomethings()
    {
        Debug.Log("开始协程");
        yield return StartCoroutine(Wait("1"));
        Debug.Log("S1");
        yield return StartCoroutine(Wait("2"));
```

```
        Debug.Log("S2");
    }

    IEnumerator Wait(string number)
    {
        yield return new WaitForSeconds(5);
        Debug.Log("W" + number);
    }
}
```

//输出顺序：

开始协程

```
Start fini
W1
S1
W2
S1
```