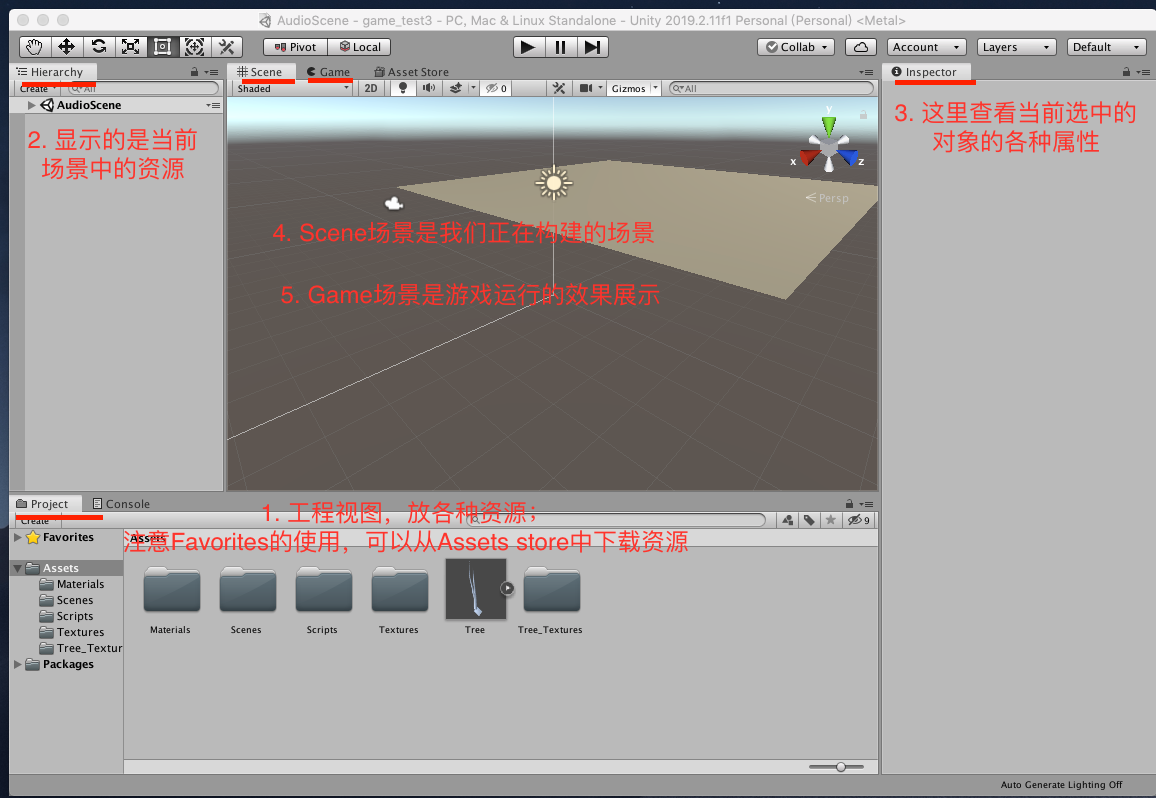
1. 简介

该笔记是在阅读《Unity游戏开发(原书第3版)》记录下来的笔记，旨在将书中的内容进行提炼归总，整理成一个单独的word文件，供后续学习回顾查看。

注：文档不是新手看的，基本建立在了解cocos creator的基础上额外进行的笔记。

1. Unity安装

这一章节主要介绍Unity安装相关的内容，不做过多笔记，我这里已经开始用2019的版本了，书中用到的是2018的版本，用的过程中看吧，差别应该不大。



这里简单的通过这个图来展示各个模块的功能，不做具体介绍。

1. Flythrough操作模式

简单来说，这种操作模式是以传统的第一人称视角来操作编辑器视角的方式，操作方式如下：

1. 按住鼠标右键 (进入Flythrough模式)；
2. 按住wasd键进行前后左右移动；
3. 按住qe键进行上下移动。
4. 总结

这一章没啥东西，就是简单的认识一下编辑器，然后会操作编辑器，不做具体记录。

1. 游戏对象

这一章节主要介绍Unity中的游戏对象，简单来说，所有的摆放在场景中的内容都是或者都是基于游戏对象的，同时，为了理解游戏对象，对Unity中的2D和3D坐标系也做了简单的归纳，下面将具体介绍。

1. 坐标系

在2D坐标系中，我们仅需要关系X轴和Y轴的值，可以通过(x, y)来描述一个物体在坐标系中的位置，即仅关心水平和垂直方向。

在3D坐标系中，我们需要额外的关心Z轴，然后通过(x, y, z)来描述一个物体在空间中的位置，即需要额外的关系深度方向。

1. 游戏对象

游戏对象作为场景中我们可以实际看到的物体，右一组组件构成 (每个游戏对象默认都有一个变换组件Transform)；对于一个3D对象来说，存在一个网格(Mesh)组件，该组件保证了我们可以看到该3D对象。

1. 总结

理解游戏对象本质上需要知道如何操作游戏对象，即如何对其做变换操作，基本的变换操作有平移、旋转、和缩放。慢慢的会掌握的。。

1. 模型、材质和纹理

这一章节需要理解相对具体一点的内容，包括模型、材质以及纹理。

1. 模型

模型，简单来说就是包含了网格信息的对象，那么什么是网格呢？具体来说，就是一系列的无缝的三角形组成的形状，我们不需要关心网格怎么实现的，Unity本身已经控制好了。

**注：**使用三角形的原因有两个：第一、当有一个三角形后，再添加一个顶点，就可以生成一个新的三角形；第二、通过三角形链接，可以对任意的3D模型进行建模。 顶点的数量会影响到绘制的速度，故我们选择三角形来作为基础进行绘制。

1.1 网格和模型的定义

网格：包含了3D对象的形状的顶点信息，即网格指的是模型的形状或形式。

模型：使用网格定义其维度，除此之外，还包含了动画、纹理、材质、着色器及其他网格。

所以，当一个模型仅包含对象的顶点数据时，我们可以称其为网格，否则就是一个模型。Unity中提供了一堆简单的3D模型，这里其实可以称之为网格，因为都是一些简单的仅包含顶点信息的对象。

1.2 使用自己制作的模型

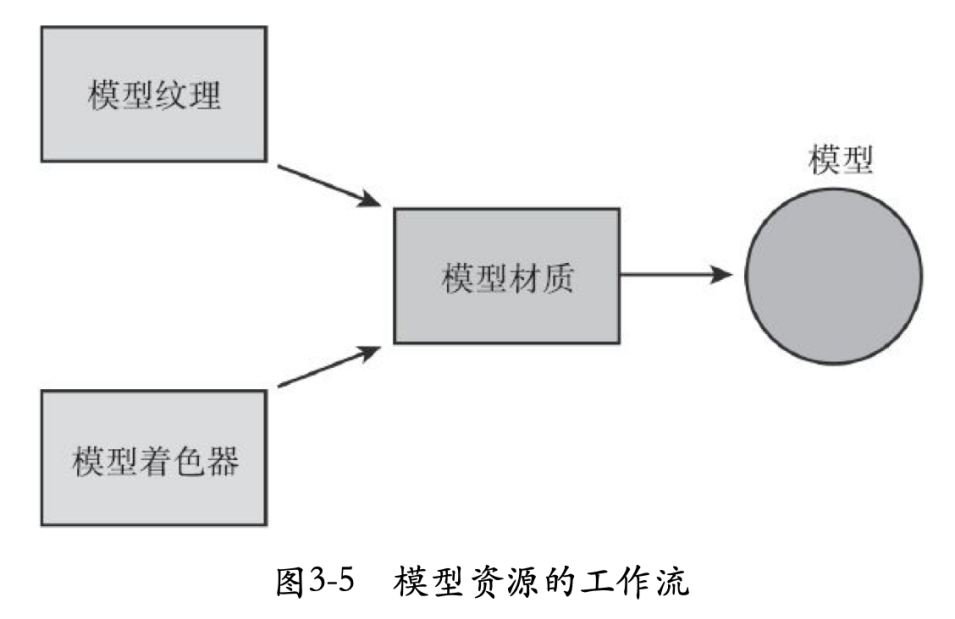
在Unity中使用自己制作的模型还是比较简单的，其支持.fbx、.dae、.3ds、.dxf、和.obj文件。

当我们将自己的模型拖动到场景中时，该对象包含了一个网格过滤器 (mesh filter)以及一个网格渲染器 (mesh render)。这两个组件保证了我们的模型被渲染到屏幕中。

1. 纹理、着色器和材质

简单的模型是无色的，即没有任何美感可言，我们需要将图形资源应用到模型上，在3D中，就需要用到上面的东西。

纹理和着色器不能直接应用到模型上，而是需要先置于材质上，然后将材质置于模型上，完成一整套渲染操作。



2.1 纹理

纹理是应用在3D模型中的平面图像(可以反向理解，将3D模型中的外观剥离出来平铺，就形成了一个纹理)。所以我们看到的模型对应的展开贴图(unwarp)会比较奇怪，毕竟其是应用于模型的。

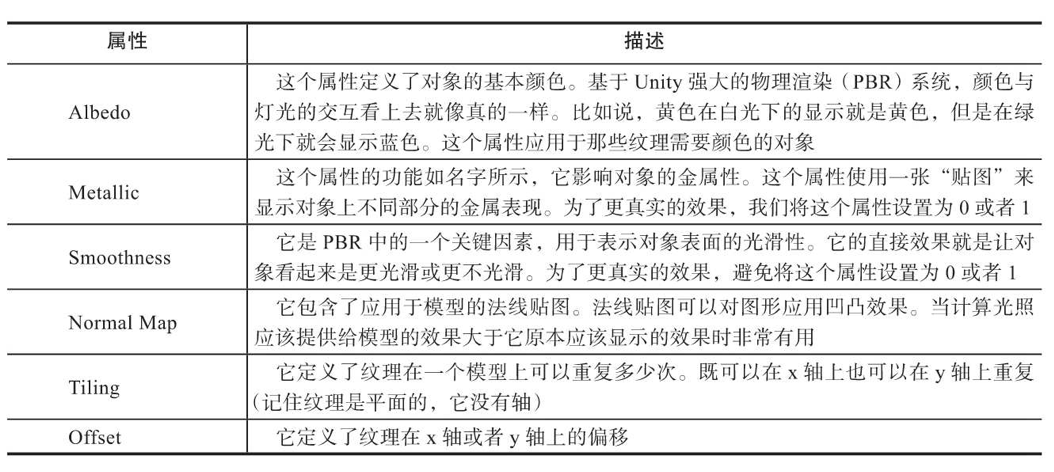
2.2 着色器

着色器的作用是确定如何绘制纹理。

2.3 材质

简单理解，材质就是将纹理和着色器应用于模型的一个容器。

2.4 Standard着色器



1. 地形和环境

这一章节将简单介绍什么是地形，如何制作宽广、辽阔的游戏地形，以及如何控制角色在地形上移动。

1. 地形

专业术语描述，地形指的是游戏中用于模拟世界外部景象的陆地地形，包括高山、平原或者潮湿的沼泽地。

Unity中的地形是可以雕刻成任何形状的平面网格，可以操控它隆起、凹陷等。但是，基本地形不能做叠层，例如洞穴或者突出物。

1.1 创建地形

通过GameObject --> 3D Object --> Terrain来创建地形。

1.2 调整地形大小

通过Inspector视图中的”设置”按钮，切换到调整地形”分辨率”的模式；

找到Mesh Resolution属性，然后修改Terrain Width及Terrain Length；

1.3 高度图

高度图是灰度图，包含了类似于地形图的海拔信息。较深的阴影可以认为较低的地方，较浅的阴影可以认为较高的地方。使用高度图可以很方便的改变地形，解析高度图的时候，我们可以通过如下公式：(灰度) / 255 \* (地形最大高度)，这样就可以看到每个位置的高度值。



1.4 高度图的使用

对现有地形使用高度图，需要如下操作：

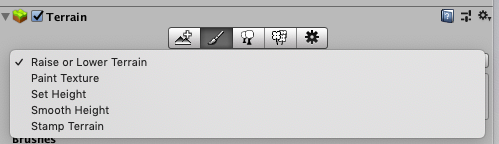
1. 选中Terrain，然后在Inspector视图中的”设置”页面找到Textures Resolution；
2. 点击Import Raw...， 将自己的高度图.raw文件导入进来；
3. 在弹窗Import Heightmap中修改数据，其中Byte Order指的是制作高度图的系统，而非我们自己使用的系统，结束后点击Import即可；
4. 我们可以在Scene视图中看到我们导入的高度图形成的新的地形。

1.5 高度图使用注意事项

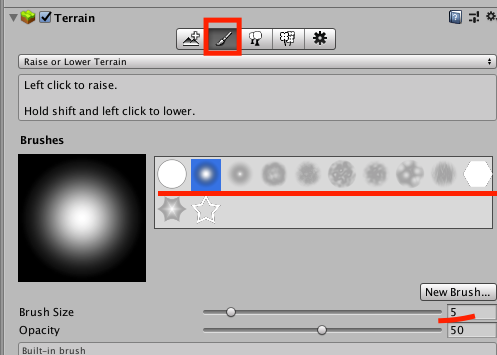
高度图必须是.raw格式的灰度图像。唯一的注意事项是，制作高度图的时候，要保证高度图和地形的长宽比一致。

1. 自己制作地形

Unity提供了一套工具来进行地形制作，下面将进行一一介绍。



2.1 Raise or Lower Terrain模式



该模式是可以随意的进行地形的隆起、凹陷操作：

1. 选中一个画刷，然后点击Scene视图，拖动鼠标，我们可以看到地形隆起了；
2. 在拖动的过程中同时按下shift键，地形降低了。

2.2 Set Height模式

该模式是将地形绘制到指定的高度；

1. 默认情况下，将地形抬高，比如设置Height为20，然后点击Flatten All；
2. 选择Brush，然后改变Height，制作不同的地形，当设置的Height > 20的话，则为高地形；当设置的值< 20时，则为低地形。

2.3 Smooth Height模式

该模式更像一个辅助模式，进行抛光操作，消除前面制作地形时候产生的锯齿。

1. 导入地形纹理

目前为止的所有地形都是无色的，没有任何美感，需要导入纹理来增加感觉，这里先导入Environment标准包。

注：Unity2018.2之后下载的时候都看不到Standard Asset了，这里直接找到2018.1的最后一个版本下载Standard Asset。

这里我通过点击Custom Package，然后从应用程序的Unity目录下Standard Assets中找到相应的包。

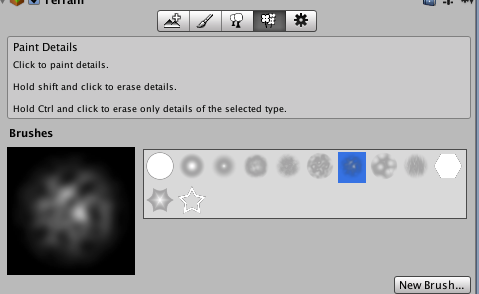
使用纹理绘制的话比较简单，切换到Brush模式，然后选择Paint Texture，点击”Edit Terrain Layers...”来创建一个Layer，选择一个图片来进行绘制即可。

1. 绘制树木



绘制树木比较简单，通过点击Edit Trees，然后选择一个树模型，画的时候直接在Scene视图中选择位置点击就好了，这个可以设置画刷的大小、树的密度以及树高，如果需要删除画好的一棵树，按住shift按键，然后点击一下要删除的树即可。

1. 绘制草地

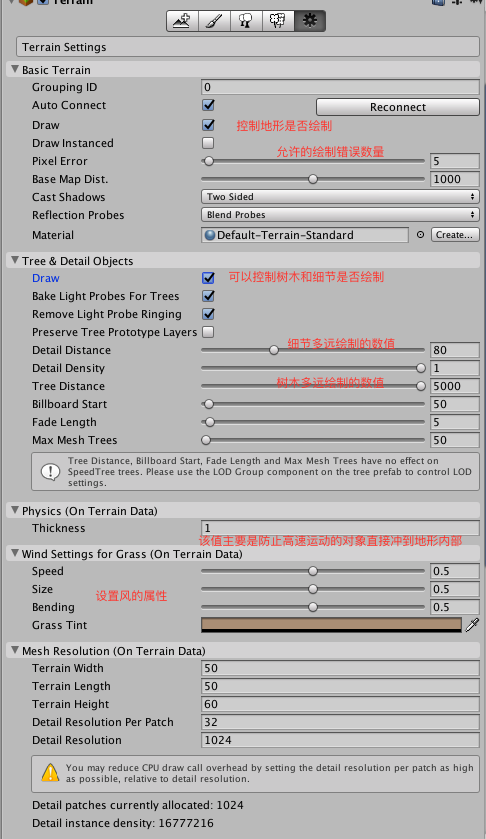


青草的绘制有单独的模式，我们将青草认为是游戏中的细节，所以绘制青草是Paint Detail。

点击Edit Details，然后添加一个Grass纹理，进行相应的添加操作，草地的可见性及细节会随着摄像机的远近而产生不同的效果，离的远的话我们只能看到一个轮廓甚至看不到，需要放大，才能看到细节。

1. 地形设置

前面的小节中我们已经使用过了Terrain setting中的内容，例如分辨率、高度图的使用，分别对应Mesh Resolution及Texture Resolution，下面我们具体介绍。



1. 角色控制器

Unity提供了两种方式添加一个角色控制器，用户无须做太多的工作就可以轻松的进入场景中。当场景中存在一个控制器后，我们就可以四下移动了。

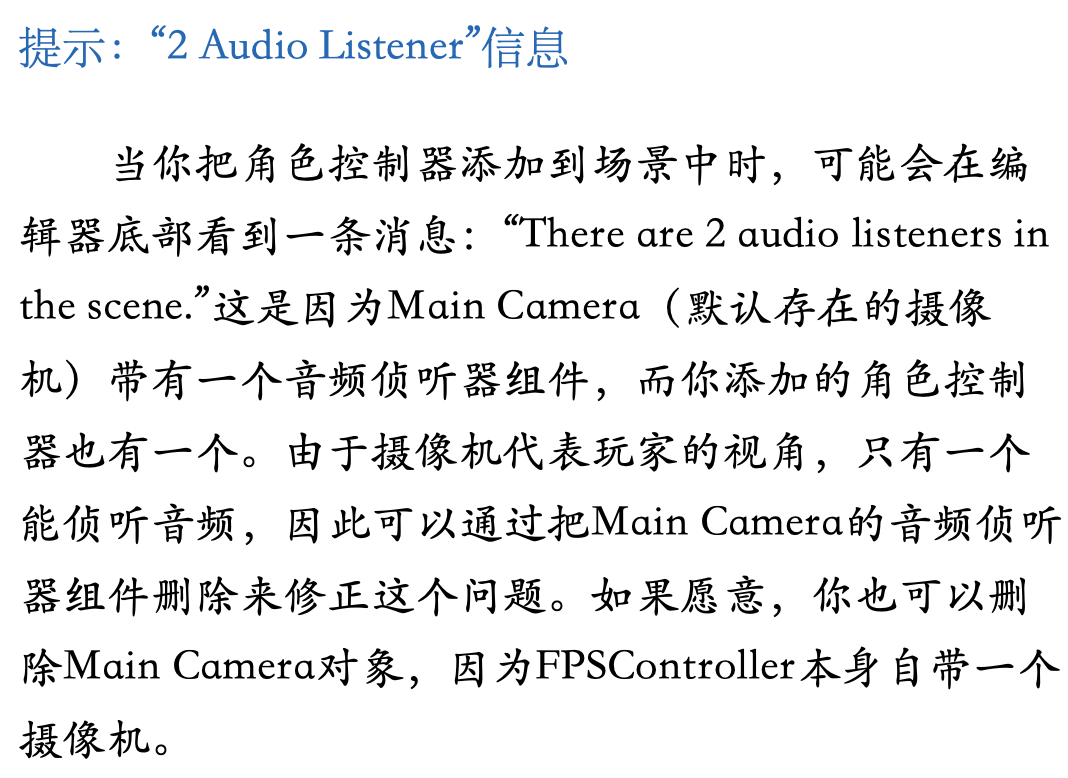
7.1 添加角色控制器

点击Import package，然后选择Characters，将该包体导入到项目中，选择Assets/Standard Assets/Characters/FirstPersonCharacter/Prefabs/FPSController.prefab，将该prefab拖动到Scene视图中。

然后点击运行，即可通过鼠标控制旋转，通过wasd来控制移动，通过空格来控制旋转。

注：

报的两个audio listeners问题。



如果角色开始移动时直接下落到地面之下了，考虑将FPSController的位置放到地面上面。

1. 总结

这一章节主要涉及到游戏中地形如何创建，包括简单到通过外部创建到高度图创建、以及通过Unity提供的一套绘制工具、树木、草地等的绘制。同时，导入了一个第一人称视角的prefab，用于感受玩家在自己创建的地形上随意运动的感觉。

1. 灯光

灯光的添加可以让游戏世界更加拟合真实世界，不同的灯光会给同一个场景造成不同的视觉效果。

在Unity中，灯光本身并非对象，而是一个Light组件，我们添加到游戏中的灯光都是带有Light组件的对象。

对于灯光的操作，有两种方式：烘焙灯光和实时灯光。不做具体介绍，看了也不懂。

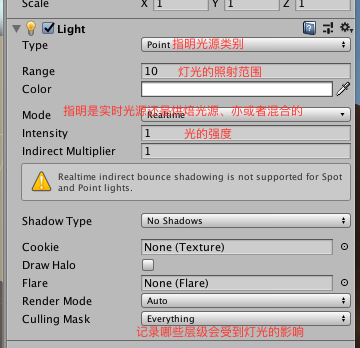
1. 灯光类别

这一章节会介绍各种类别的灯光，下面开始具体介绍。

1.1 点光源

简单理解，点光源类似于一个灯泡，从一个中心点开始向四周照明。

通过Create object --> Light --> Point Light来创建。



1.2 聚光灯 (SpotLight)

类似于车灯或手电筒，灯光照射出去的方向向前。

其有一个属性Spot Angle，改变该值可以修改照射出去的灯光的范围。

1.3 定向光

定向光比较通用，默认情况下新建的场景会自带一个定向光，用于照亮整个场景。

1. 光晕

光晕的话主要是增加感官，通过勾选Light组件中的Draw Halo属性即可实现。

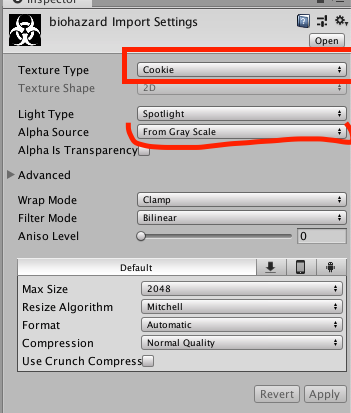
通过Window --> Rendering --> Lighting Settings来对光晕进行修改。

1. Cookie的使用

Cookie的作用类似于，在墙壁及灯光之间放一个物体，然后物体的影子照到墙壁上的效果。

Cookie的效果就是将这个图案直接放到墙上，具体操作方式如下。

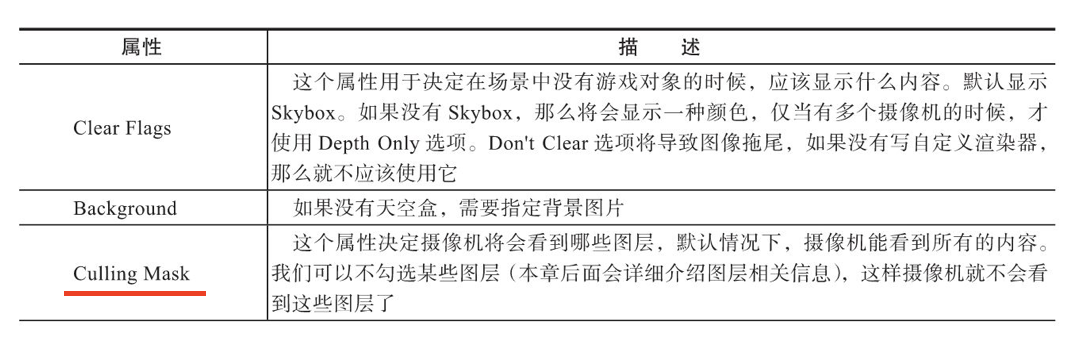
1. 选择一个符合2的N方的图片；
2. 选中图片，然后在Inspector视图中修改Texture Type及Alpha Source，点击Apply应用；
3. 将图片拖到Light组件的Cookie中，灯光选择Sport，然后就会看到图片映射到墙上了。

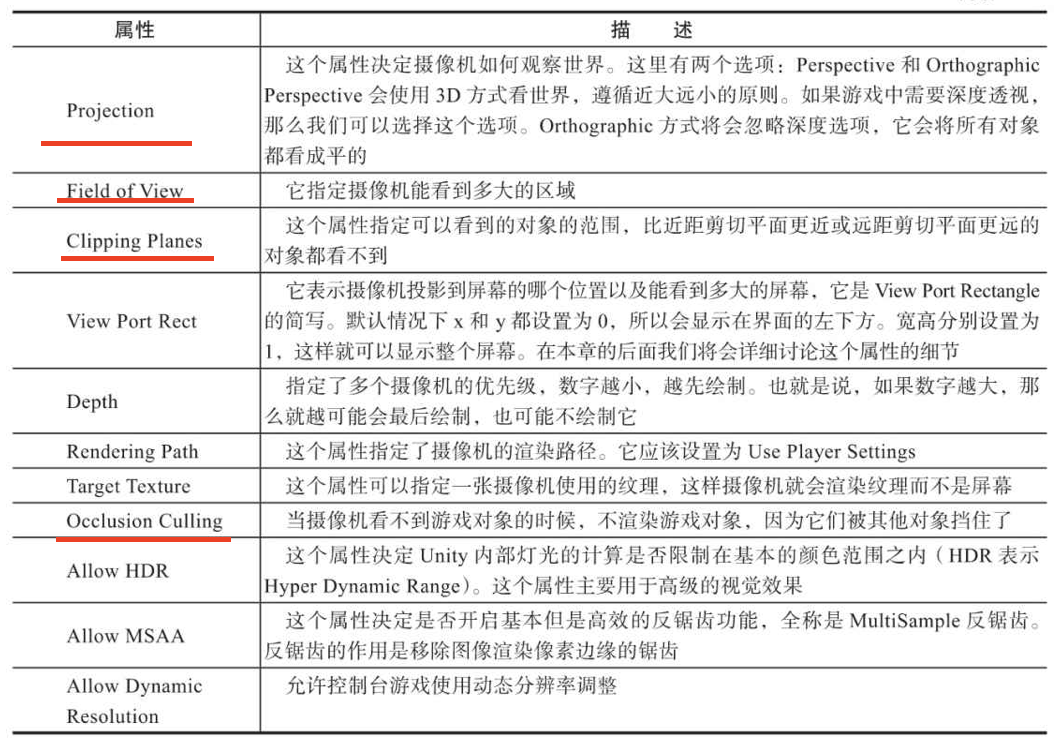


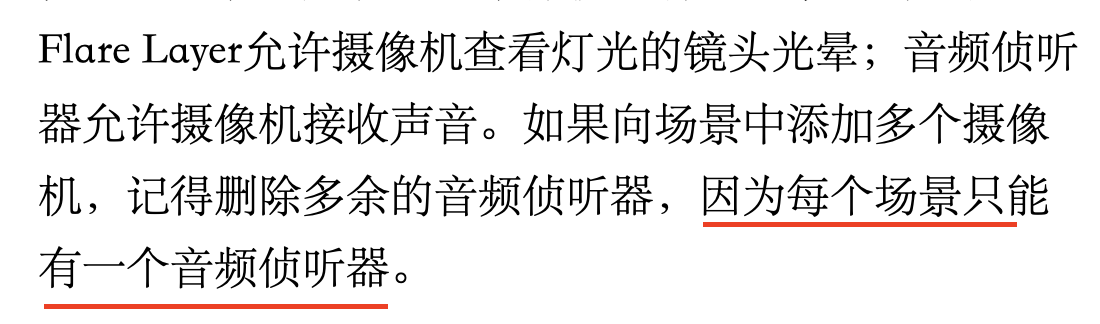
1. 摄像机

摄像机是我们用于观察游戏场景的必要游戏对象，对于一个新建的场景，必然有一个Main Camera。

摄像机有如下一堆属性，需要特别关注的已经标红。







1. 多摄像机实现小地图

小地图的本质是通过一个额外的相机将内容显示在固定的位置，简单的操作的话可以通过修改相机Camera的Viewport Rect属性。



上述参数的意思是指定相机绘制的起点为0.75，0.75的位置，然后相机本身的视口大小为0.2 \* 0.2。

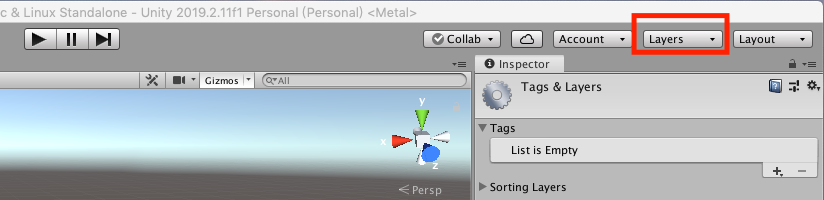
1. 图层

所有的游戏对象都属于某一个图层Layer，默认情况下，3D对象属于Default图层，2D的对象基本都属于UI图层。我们还可以通过Add Layer... 来添加额外的图层来进行自己控制。这里的图层的概念，可以理解为Creator中的分组的概念，为创建的游戏对象指定不同的分组。

6.1 图层的作用

图层的作用是分组，那么我们可以用于以下内容：

1. 在Layer管理器中控制该层的情况；



1. 选择Light中的Culling Mask，控制灯光照射的层；
2. 用于碰撞管理，指明哪些之间可以发生碰撞；

注：通过Edit --> Project Settings... --> Physics来修改碰撞分组情况。

1. 总结

这一章节主要介绍了灯光，以及摄像机的作用，比较简短的介绍，可以大致了解灯光的类别，摄像机如何操作。

1. Amazing Racer游戏制作
2. 游戏玩法

Amazing Racer游戏是一款比较简单的竞速游戏，从游戏的一端跑到另一端，在地形中设置了各种障碍，包括水坑、树木等内容，玩家需要从出生点跑到结束区域，以尽可能短的时间。

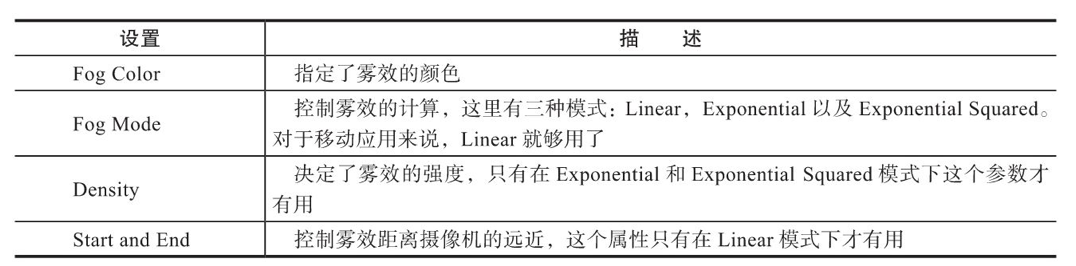
1. 游戏中的一些重点记录

这里记录了游戏中的一些重点内容。

2.1 雾效

雾效主要是为了增加游戏的真实感，制造一些特殊天气，如大雾天气等，操作方式如下：

1. 打开window --> rendering --> lighting setting；
2. 在other settings属性中，勾选Fog属性；
3. Fog有一些属性记录如下：



2.2 天空盒 (skybox)

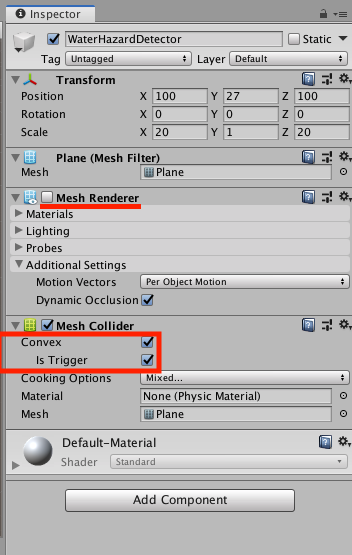
天空盒是一个包裹着整个游戏场景的巨大的盒子，可以通过如下方式创建：

1. 在Project视图中，右击Create --> Material；
2. 点击Material，在Inspector视图中修改Shader属性为Skybox - xxxx;
3. 点击window --> rendering --> lighting settings 中修改default skybox；

2.3 如何创建一个有碰撞，但是看不到的平面

我们需要一个可以用于检测碰撞的平面，但是视觉上不希望看到它，我们可以如下操作：

1. 右击，3D Object --> Plane；
2. 勾选Mesh Collider中的Convex以及Is Trigger；
3. 取消Mesh Renderer的选中，这样的话就看不到该平面了。



1. 总结

这一章节主要介绍了一个简单的Amazing Racer游戏如何制造的重点，具体该怎么做没有记录，主要是对前面几章内容的简单总结。

1. 脚本

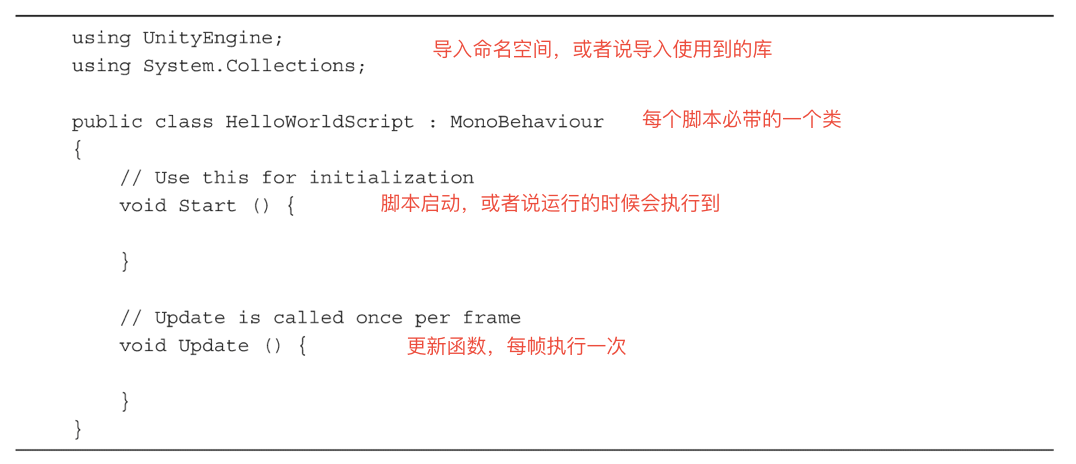
脚本的作用是串联游戏对象，以及将我们的想法通过代码实现起来，同时在合适的时机来操作游戏对象。

1. 脚本名称

C#中的脚本名称必须和当前文件对应的类名保持一致。

1. 脚本最简单的结构解析

这一小节将简单的介绍一个最简单的脚本结构，同时介绍每一部分的内容及作用。



1. 输入控制

正常的操作来说，我们是监听特定的按键是否按下来决定响应什么事件，但是这种方式操作比较局限，Unity引入了一个轴的概念，可以让我们针对一个轴进行某个操作。

3.1 Input Manager中提到的轴

通过Edit --> Project Setting... -> Input打开目前已有的18个轴的输入情况。

使用方式如下：

float hVal = Input.GetAxis(“Horizontal”);

如果按下了左键或者A键，则是一个负值；

如果按下了右键或者D键，则是一个正值。

以上是当我们仅关心输入轴的情况下，不需要具体的关注是哪个按键。

3.2 特定的按键监听

对于需要明确的监听特定按键的需求，我们通过GetKey来达到目的。

bool isKeyDown = Input.GetKey(KeyCode.K)；

上述监听就可以实现监听是否按下了K键的目的，且当没有释放的时候，会一直为true。

Bool isKeyDown = Input.GetKeyDown(KeyCode.K);

该接口是在按键第一次被按下的时候响应一次。

同时，还有一个方式为GetKeyUp。

3.3 鼠标输入的按键监听

鼠标输入的作用主要针对主机游戏，Input.GetMouseButtonDown可以满足需求。参数为0/1/2，分别对应左键/右键/中间滚轴。

1. 组件的控制

在脚本中我们可以获取指定游戏对象中包含的组件。

GetComponent<Light>()；获取当前节点的Light组件；

而对于Transform来说，则使用起来很方便，因为其使用频率比较高。

transform.Translate()；/ transform.Rotate(); / transform.localScale 三种操作。

1. 对象的查找

比较简单的操作是我们定义一个变量，然后直接通过编辑器拖动，当不想拖动的话，可以通过以下几种方法来操作：

1. 通过节点名字查找：GameObject.Find(“Cube”)，会找到第一个匹配的名为Cube的对象，不是特别好；
2. 通过节点的tag来查找：首先通过tag manager来添加自己指定的tag名，然后调用方法来查找：GameObject.FindWithTag(“MyTag”)；
3. 总结

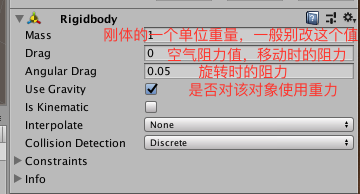
这一节前半部分主要是对脚本进行了简单的归纳，很多内容没有笔记，因为属于基础的语法向，后半部分主要是介绍了如何监听输入，以及如何通过脚本来控制组件、查找对象。

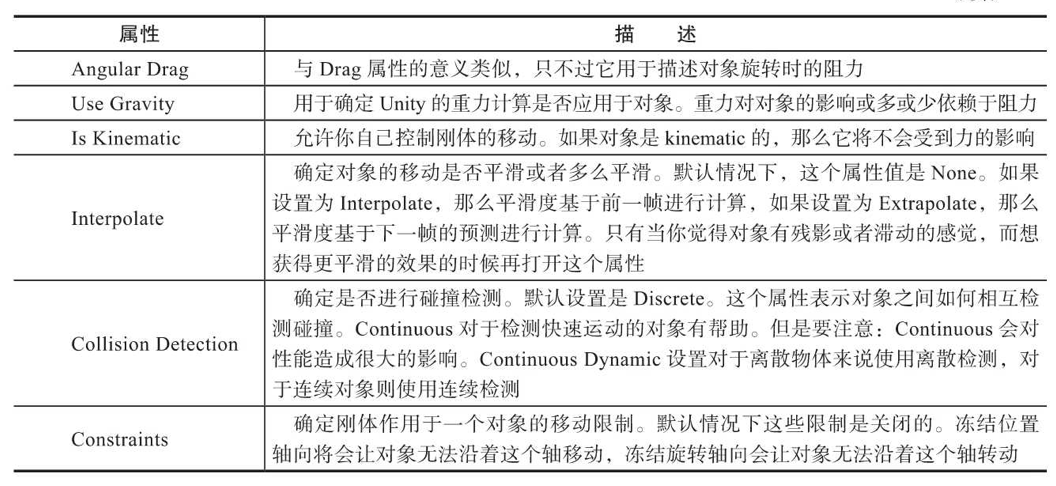
1. 碰撞

碰撞是我们产生游戏效果必须的操作，理解碰撞，我们需要知道刚体的概念。了解Unity提供了两种物理引擎：Box2D和PhysX。利用触发器检测碰撞，以及学习使用光线投射来检测碰撞。

1. 刚体

刚体的使用是为了配合物理引擎的生效，即当我们需要一个对象表现出真实世界的效果的时候，需要为该对象添加Rigidbody组件，Unity中的刚体组件属性如下。





表现：对于一个立方体，当不存在刚体时，运行游戏，发现立方体不会动，然后添加刚体，并且勾选Use Gravity，发现刚体向下移动。

1. 碰撞体

碰撞体的存在，使得我们可以在两个对象发生碰撞的时候进行某种操作，即监听碰撞的发生，并处理相应的操作。

默认的我们创建的3D对象都会加上一个碰撞组件，例如Plane默认带有Mesh Collider；Cube默认带有Box Collider等等。

碰撞体有一些基础属性，我们可以调整碰撞体的大小，偏移，以及改变碰撞组件的材质。

当我们不勾选Is Trigger的话，则起作用的是物理碰撞器，该物理碰撞器自发的产生碰撞处理效果。

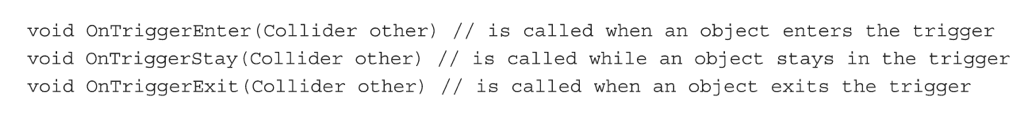
2.1 物理材质

物理材质本质上决定了物体发生碰撞后的表现，比如反弹、平移、弹起跳动几下等。Standard Assets中提供了一些内置的物理材质，导入Charactor包，然后在PhysicsMaterials中找到几种物理材质：Bouncy(无损弹跳)、Rubber(正常的碰撞反弹效果)。

1. 触发碰撞器

Unity提供了区别于物理碰撞器的额外的一套碰撞器：触发碰撞器。当勾选了Collider组件的Is Trigger选项的话，则开启了触发碰撞器。

触发碰撞器可以检测到碰撞发生，但是不会对碰撞进行处理，需要自己来写处理函数。



other指的是与当前对象发生碰撞的实体，通过other.gameObject可以获取到当前发生碰撞的对象。

1. 光线投射

简单理解，光线投射就是以一个点为起点，向着某个方向检测，看是否有与物体发生碰撞。



origin：光线投射的起点；

direction: 光线检测的方向；

distance：光线投射检测的距离；

mask：用于决定光线能够撞上哪些层；



这个重载方法中有一个参数为hit，表明光线是否发生了碰撞，同时引入了关键字out，表明这个参数是一个输出参数。hit的类别为RaycastHit。

该hit可以获取到碰撞的对象：hit.collider.gameObject。

1. 总结

这一章节介绍了几种碰撞的方式：通过刚体来响应物理世界；通过勾选collider的is trigger，使用触发碰撞器来产生效果；通过光线照射检测。

在实际中，我们可以选择适合自己的方式进行控制操作。

1. ChaosBall游戏制作
2. 介绍

ChaosBall游戏是一个弹力球游戏，玩家通过控制球将其打入指定区域，同时，又不要碰撞到屏幕中到混乱球。

1. 重点记录

这一小节将记录该游戏制作过程中的一些知识点。

2.1 将物理材质应用于整个游戏中的所有对象

打开Edit --> Project Settings --> Physics；

将Default Meterials设置为你自己创建的物理材质。

2.2 为刚体设置初速度

通过GetComponent<RigitBody>来获取当前节点上的刚体组件，然后指定它的velocity属性，即指定其三个方向的速度 (x、y、z轴)。

1. 2D游戏

使用Unity制作2D游戏也是可以的，而且即使我们做的3D游戏，我们可能也需要一些2D的界面，所以我们需要了解2D相关的内容，下面将对其中的一些重点进行记录。

1. Sprite

在2D中，我们需要用到图形来进行渲染。

当我们创建的是2D项目时，默认图片的Texture Type属性为”Sprite(2D and UI)”。但是，如果是创建的3D项目的话，我们需要手动修改一下，将Texture Type改为这个值。

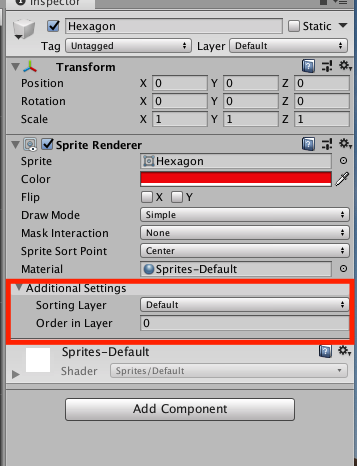
1. SpriteSheet的使用

精灵表单，简单点说就是精灵的合图，通常用于制作序列帧的时候。

如何将一个单图切割然后用于动作制作呢？

1. 将单图拖到Unity Editor中，然后点击图片，打开Inspector视图；
2. 修改图片的Sprite Mode为Multiple，然后点击应用；
3. 点击Sprite Editor，打开精灵编辑器，如果提示没有的话，则点击Window --> Package Manager，然后搜索Sprite，进行安装；
4. 打开Sprite Editor，然后点击Slice，按照自己想要的分割方式进行分割，主要是调整Type，然后确定的话点击Slice，我们可以看到其是如何分割的；
5. 完成后点击Apply，可以在Project视图中展开图片，看到整张大图被分割了。
6. Sprite的绘制顺序

为了确定哪些图层在上面，哪些在下面，Unity提供了sorting layer和order in layer来进行控制。



点击Sorting Layer --> add sorting layer...；

然后点击+，添加sorting layers；

按照添加进去的顺序排列，排在最下边的最后渲染，可以遮挡其他的；

同时，在同一层中，还可以调整Order In Layer来调整绘制层级，注意这个问题哈。

这个层级控制，类似于cocos creator中的zIndex，不过这里是提前定义好统一的几种层级关系。比如和我们设置场景层 (地图层、中间层、顶层、以及弹窗层)。

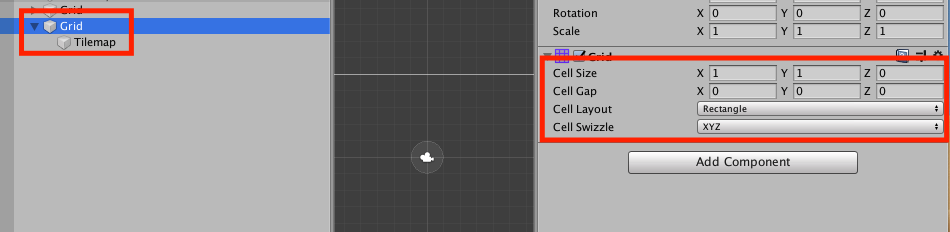
1. 瓦片地图（TileMap）

这一小节介绍TileMap的内容。TileMap经常用于2D游戏中地图的生成操作，相对来说还是比较重要的。

4.1 Unity中的瓦片地图操作

在Unity Editor中，我们可以直接创建一个TileMap对象。2D Object --> TileMap来创建好该对象。

一般来说，我们在创建一个地图时，一般会设置好前景、中景和背景。那么可以通过设置TileMap Render的sorting Layer来达到这个目的。



当我们创建一个TileMap的时候，会默认变成上图的结构，即会自动创建一个Grid作为地图的父节点，Grid中指定了格子的大小，如果我们所有的地图的格子大小都一样的话，则可以放在一个Grid下面，否则要用多个Grid来控制。

4.2

1. 画布

画布作为2D游戏的组织者，所有的2D元素(按钮、图片、文字)都位于画布(Canvas)之下，而当我们创建一个Canvas之后，同时会创建一个EventSystem出来，这个对象作为接收用户和UI界面的交互监听对象，必须存在。

画布重构：当我们对UI进行改变后，整个画布会进行重构，所以可以将经常变化的放在一个Canvas下面，而不动的放在另一个Canvas下面。

1. UI元素
2. Captain Blaster游戏制作

这一章节记录了飞机游戏的内容，这里将记录一些基本的上面没有提及的信息。

1. 将一个图片切割的sprite表单组装成一个动画帧
2. 将拖入到工程中的图片用sprite editor进行切割；
3. 选中所有的动作帧(选中第一个，按住shift，选中最后一个)；
4. 拖入到场景中即可。
5. 如果图片反向了，可以修改scale的x或者y为负数，进行镜像操作。
6. 碰撞边界销毁

在场景的边界，上下分别放一个空节点，然后挂载一个box collider，开启is trigger，监听onTriggerEnter2D函数，当发生的时候，执行Destroy(collision.gameObject)的操作。

1. 创建运动的物体

陨石和子弹都是在创建的时候移动的物体，那么简单来说就是预制体绑定刚体和碰撞体组件，然后在创建对象的时候，为刚体指明velocity。

一个预制体需要绑定场景中的对象的时候，使用GameObject.FindObjectOfType<Type>()来找到GameManager，即Type指定的东西。

1. 子弹和陨石的碰撞

两个碰撞体之间发生碰撞，然后会进入到OnCollisionEnter2D中，然后在这个里面进行逻辑处理，比如说刚体的移除，然后分数的累加等操作。

1. 总结

这一小节完成了一个简单的飞机碰撞陨石，发射子弹销毁陨石的操作。涉及到了动画帧、碰撞、触发、文本刷新等内容。

同时，注意，当修改Time.timeScale = 0的时候会将停止。

1. 粒子系统

模块很多，不细学了，需要时候再学习吧。

1. 动画

动画系统对标的就是cocos creator中的简单的动作系统，包括操作对象的旋转、缩放、移动等，这些操作都可以通过动作系统来实现，同时动作帧也属于动画的范畴。

1. 动作帧

动作帧的制作比较简单，我们将一个图片拖到项目中，如果是3D项目，则修改sprite type为texture类别，然后修改type为multi类别，打开sprite editor，进行切割。

完成之后，选中所有的图片，拖动到scene视图中，则创建好了一个动作帧。

1. 动画系统

通过Window --> Animation --> Animation来打开动作系统；

选中要操作的节点，然后点击create，则创建了一个animation；

通过add property来将需要修改的属性添加到动作中，然后进行操作。

1. 记录模式(record)

记录模式的作用主要是简化制作过程，即当我们打开记录模式之后，我们对节点的所有属性的修改都会被记录在当前帧，然后这个修改就会被应用到节点上，从而产生动作效果。



1. Curves Editor模式使用

该模式主要是对动作进行细致的调整，包括动作曲线等。

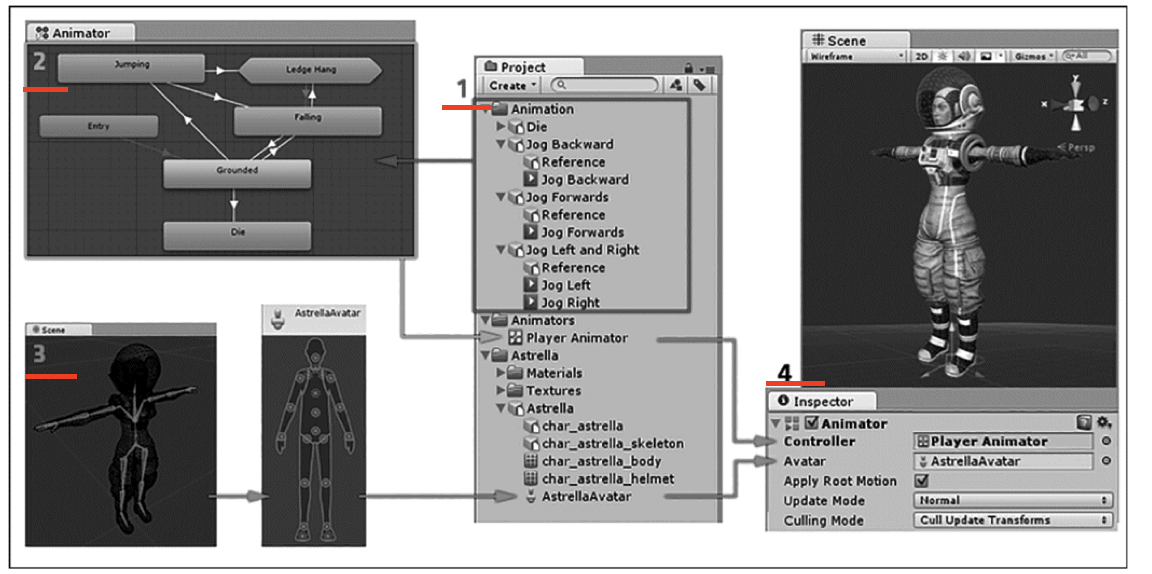
4.1 调整某个属性

1. 打开Curves，选择Animation窗口中下方的视口，进行切换；
2. 选中某个属性，然后右侧的曲线会亮起来，如果没有填充满，可以鼠标选中这条线，然后按F，则会放大；
3. 然后进行动作曲线调整。
4. 总结

本章记录了动画系统相关的内容，这个章节需要自己多多联系来掌握，基本可以参考cocos creator中的动作系统来看，两个系统还是比较像的。

1. Animator

Unity中所有的动画都始于Animator组件。Unity的动画系统(Mecanim)主要由三部分组成：动画片段、Animator控制器以及Animator组件。如下图所示：



动画片段简单说是需要表现动作的GameObject，上图中的1；

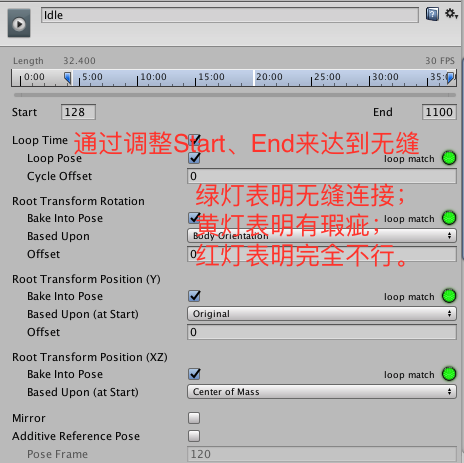
模型指的是3；

1. Unity中对动画的修改

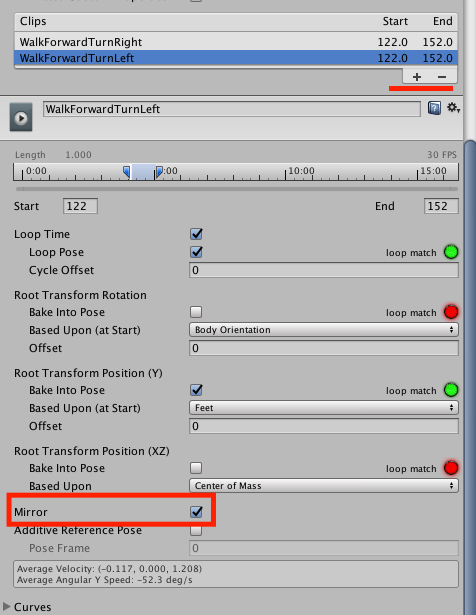
当我们将一个模型(.fbx)拖入到项目中后，可以通过右侧的检视(Inspector)视图对其参数进行修改：

1. 通过Rig --> Animation Type，可以修改为Humanold、Legacy、Generic、None等等；
2. 通过Animation --> 下方的内容，可以修改动画的执行方式，是否循环，以及开始结束帧。

注：Unity通过指示灯的形式对当前动作的执行是否有缝进行了标识。



c. 当我们制作了一个动作后，可以直接制作动画的镜像，例如基于左转计算右转，如下两步操作，两个动作的参数完全一致，唯一的区别是Mirror的勾选：

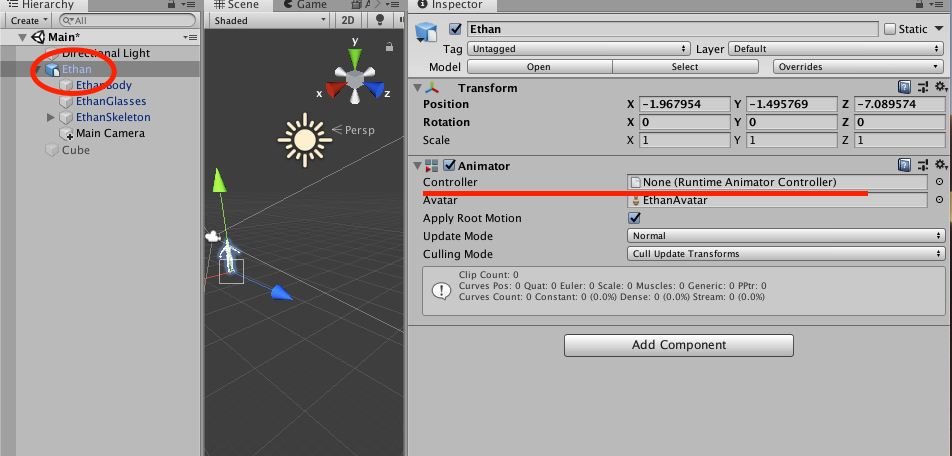


1. 创建一个Animator

首先明确，Animator是作为一个资源为存在的，那么意味着我们创建并设置好一个Animator之后，我们完全可以反复的使用它在不同的项目中。

下面记录如何创建一个Animator：

1. 在Project视图中创建一个文件夹；
2. 文件夹右击选择Create --> Animator Controller；
3. 将一个模型拖动到场景中，比如Assets/Standard Assets/ Characters/ ThirdPersonCharacter/Models/Ethan.fbx该模型；
4. 选中该模型，然后将刚才创建到Animator拖到属性中；

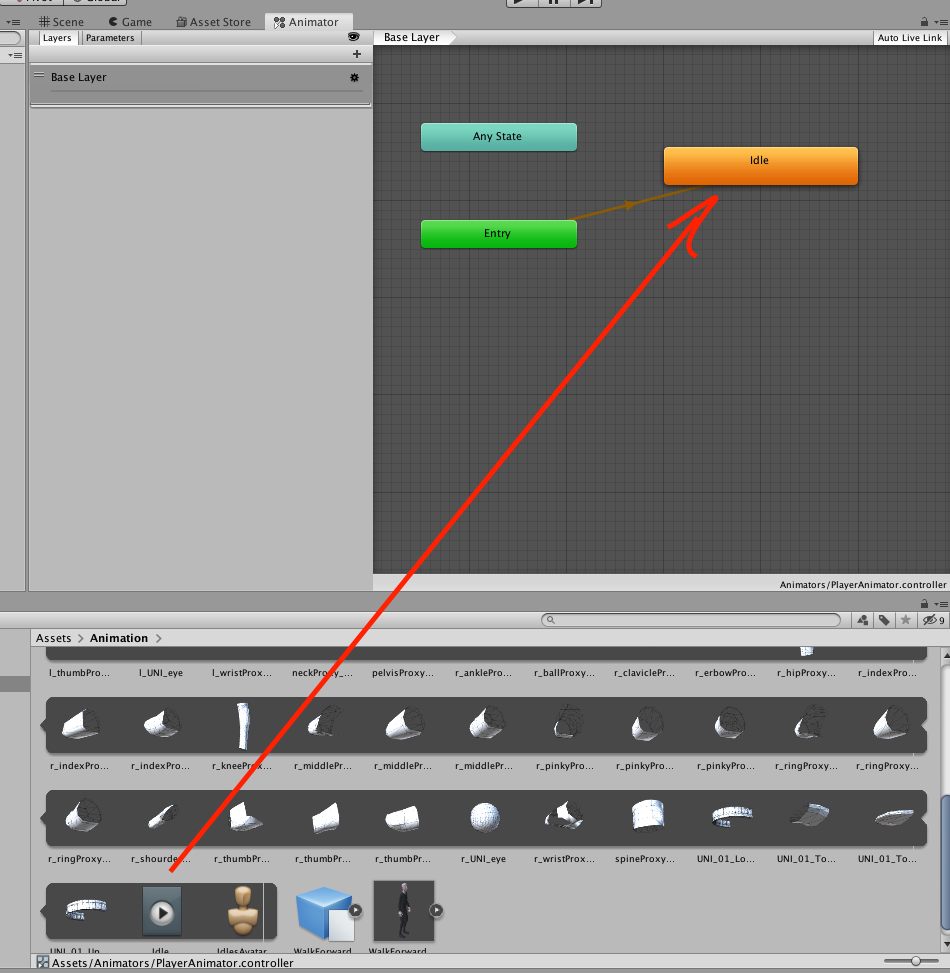


1. 操作Animator视图

双击一个Animator则可打开Animator视图。

3.1 将动画应用到Animator视图

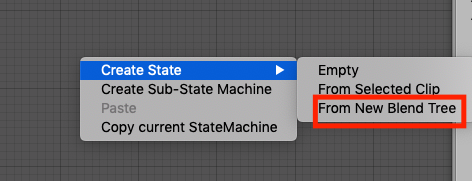
找到一个动作，然后将其拖到Animator视图中，直接拖入的形式。



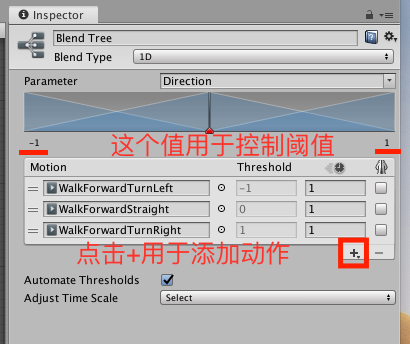
3.2 创建混合树 (Blend Tree)

当某个动作属于一类的话，例如行走动作，包括直行、左转、右转。那么就需要用到混合树了。

1. 右击Create State --> From New Blend Tree；



1. 双击新建的Blend Tree，然后选中它，观察右侧的Inspector视图；

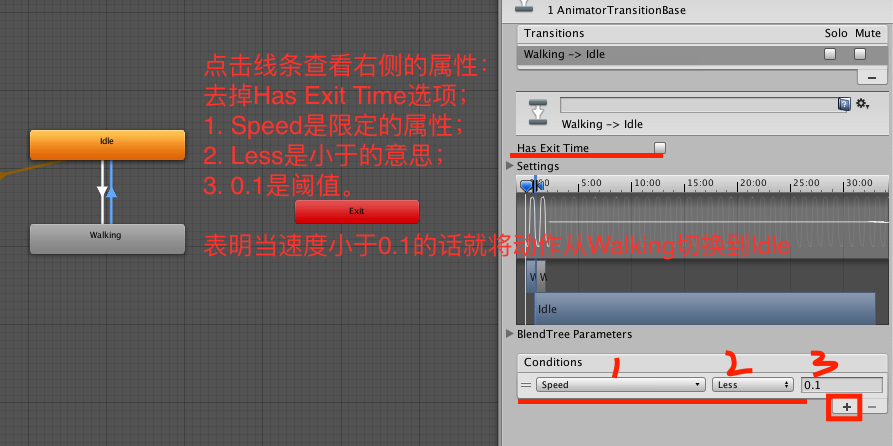


c. 操作Blend Tree不同动作的方式是修改Paramter的值，当值为0.5的时候，执行Turn Left，当为0的时候，执行Straight，当为1的时候，执行Turn Right，其他时候按百分比混合两个动作。

3.3 如何在不同的动画之间过渡

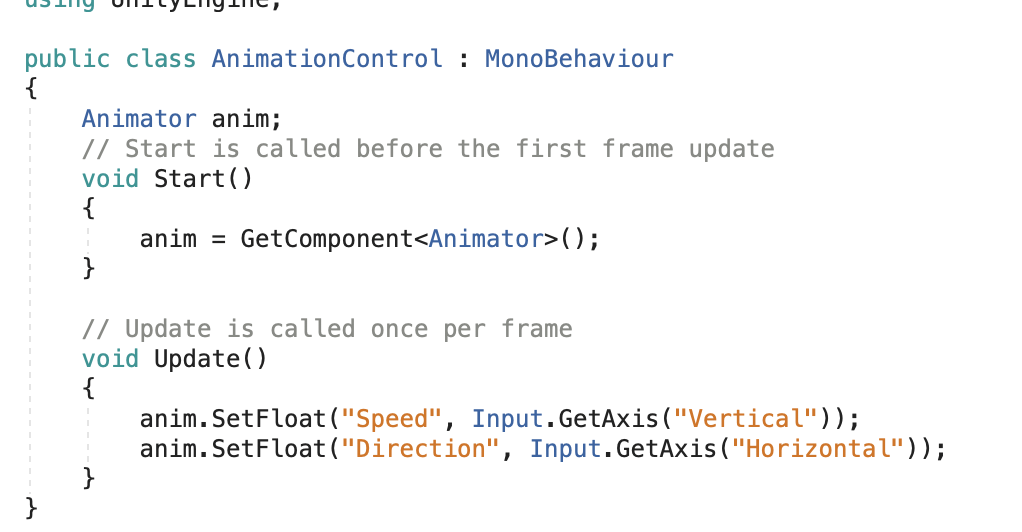
不同的动作之间需要进行切换，Animator提供了过渡方式：

1. 右击创建的一个State，选择Make Transition，然后再点击需要切换到的状态，完成创建；
2. 这样只是完成了单向切换，如果需要反向过渡的话，则再需要一条额外的线；
3. 单击刚才创建的线，然后进行编辑；
4. 编辑的本质是设置两个动作的切换条件：



1. 脚本控制

脚本控制简单来说就是控制参数的修改，主要通过调用SetXXX方法，该方法的对象是Animator。



左右是水平方向，控制转向；

垂直为移动，控制Speed。

1. 总结

这一章节了解的简单的Animator控制器，该控制器帮助我们方便的操作对象的动作，包括移动的混合树、不同动作之间的切换等等，内容比较多，需要慢慢消化使用来增强了解记忆。