# 第3章

# 人机交互与虚拟环境

在《虚拟现实技术及其应用》一书中，阐述了虚拟现实的特征是“沉浸”“交互”“想象”三者相互影响，缺一不可。人机交互、人工智能的发展以及计算机运行性能和计算机网络通信的迅速发展，让虚拟现实技术的应用领域有了进一步的突破，如医疗、军事模拟训练、互动娱乐等领域。那么人机交互技术的运用，以及如何将人机交互技术适应虚拟环境的发展这一问题就日渐突出。根据日常应用，传统的输入输出和人的感知方式要保留，虚拟环境中的人和物如何来进行真实有效的模拟也成了人机交互技术在虚拟环境研究领域的重点。在本章中，结合 Unity 3D 进行碰撞检测的学习、视觉交互的学习、声音的可视化学习以及虚拟环境中简单的人工智能的应用，分别从视觉、听觉、虚拟触觉等角度进行人机交互的应用。

**教学的重点和难点**

- 声音交互的实现；
- 触发检测的函数、应用方法以及和屏幕坐标的转换；
- 人工智能行为在虚拟环境中的交互应用。

**学习指导建议**

- 重点掌握视觉、听觉、摄像机交互等在虚拟环境中的运用，在对碰撞检测认识的基础上，会在不同的虚拟环境中进行几种碰撞方式的选择和运用。
- 基于导航网格和群组行为等形成虚拟场景中基本的人工智能的操作，进行功能的模块化学习和扩展训练。
- 强化练习碰撞检测、触发检测、射线检测的使用，可进行各种检测方式在虚拟环境中的应用来进行练习，以达到熟练使用的程度。


视频讲解

## 3.1 视觉交互

人的感知即通过人体器官和组织进行人与外部世界的信息交流和传递认知是人们在进行日常活动时发生于头脑中的事情，它涉及思维、记忆、学习、幻想、决策、看、读、写和交谈

等，人的感知是认知的基础，认知是将感知获取的信息综合运用，认知分为经验认知和思维认知，认知过程是相互联系的，单纯的一个认知过程是非常少见的。

视觉：在黑暗而空气清新的夜晚，人们可以看到 30 英里（48000 千米）外的一只烛光（1 英里≈1.6 千米）。

听觉：在安静的环境中，人能够听到 20 英尺处的手表滴答声（1 英尺＝0.3 米）。

嗅觉：人能嗅到 1 升空气中散布的 1/1000000 毫克的人造麝香的气味。

味觉：人可尝出 9 升水中放一茶匙糖的甜味。

触觉：人可感到蜂蜜翅膀距脸颊 1 厘米处落下。

视觉是人与周围世界发生联系的最重要的感觉通道，外界 80％的信息都是通过视觉得到的，因此视觉显示是人机交互系统中用得最多的人机交互界面。视觉感知可以分为两个阶段：受到外部刺激接收信息阶段和解释信息阶段。视觉感知特点：一方面，眼睛和视觉系统的物理特性决定了人类无法看到某些事物；另一方面，视觉系统进行解释处理信息时可对不完全信息发挥一定的想象力。进行人机交互设计需要清楚这两个阶段及其影响，了解人类真正能够看到的信息。视觉活动始于光，眼睛接收光线，转换为电信号。光能够被物体反射，并在眼睛的后部成像。眼睛的神经末梢将它转换为电信号，传递给大脑。眼球结构如图 3.1 所示。

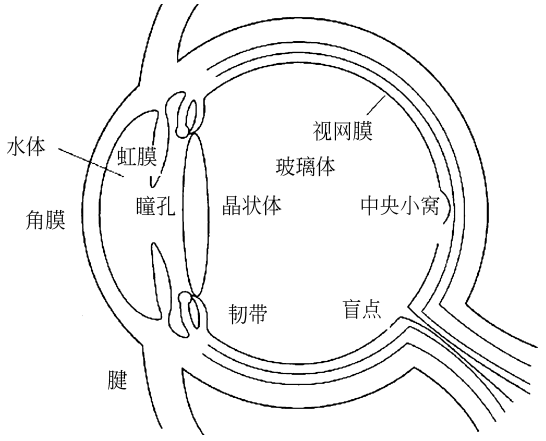


**图 3.1　眼球结构**

视网膜由视细胞组成，视细胞分为视干细胞和视锥细胞两种，它们是接收信息的主要细胞。视敏度指人眼对细节的感知能力，通常用被辨别物体最小间距所对应的视角的倒数表示。通常将能分辨出视角 1′的视敏度定为 1.0。一般人能够在 2m 的距离分辨 2～20mm 的间距，为设计人机交互作品时提供了依据。利用视觉影像中的线索，如覆盖关系（被覆盖的物体相对较远）、大小比例（一般来讲，较大的物体距离较近）、对物体的熟悉度（对非常熟悉物体，人们对物体的大小在头脑中事先有一个期望和预测，因此在判断物体距离时很容易和他看到的物体的大小联系起来）。随着亮度的增加，闪烁感也会增强。在高亮度时，光线变化低于 50Hz，视觉系统就会感到闪烁。在设计交互界面时，要考虑使用者对亮度和闪烁的感知，尽量避免使人疲劳的因素，创造一个舒适的交互环境。物体距离与物体大小的联系如图 3.2 所示，物体与观察者距离不同情况下的视角变化如图 3.3 所示，视干细胞与视锥细胞的区别如表 3.1 所示。

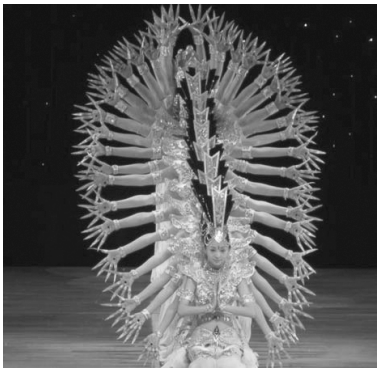

图 3.2　物体距离与物体大小的联系

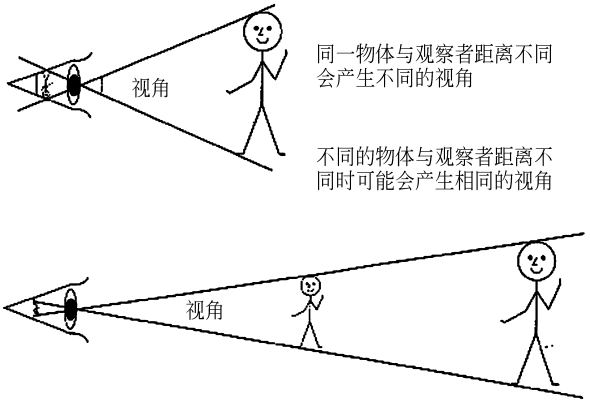


图 3.3　物体与观察者距离不同情况下的视角变化

表 3.1　视干细胞与视锥细胞的区别

| 视 干 细 胞 | 视 锥 细 胞 |
|---|---|
| 在低水平照明时(如夜间)起作用 | 在高水平照明时(如白天)起作用 |
| 区别黑白 | 区别彩色 |
| 对光谱中绿色部分最敏感,在远离视网膜中心处最多 | 对光谱中黄色部分最敏感,在视网膜中部最多 |
| 增强亮度可以提高视敏度 | 主要在识别空间位置和要求敏锐地看物体时起作用 |

人能感觉到不同的颜色,这是眼睛接收不同波长的光的结果。颜色通常用三种属性表示:色度、强度和饱和度。色度是由光的波长决定的,正常的眼睛可感受到的光谱波长为400～700nm。图片亮度变化如图 3.4 所示。

图 3.4　图片亮度变化


视频讲解

## 3.2 基于环境交互

### 3.2.1 Unity 3D 碰撞检测交互

在现实世界中，两个物体不可能共享同一个空间区域。在虚拟环境的人机交互过程中，为了提升用户交互过程中的沉浸感，同样需要在虚拟场景中实现两个不可穿透物体间互不共享同一空间区域的体验。在本书中，若未对虚拟场景中的虚拟物体添加碰撞检测，虚拟物体之间在相互碰撞后会出现“穿越”现象，这将会带来极差的用户体验。

如果想要实现碰撞效果必须为交互对象添加刚体和碰撞器，其中刚体可以让物体在物理影响下运动，碰撞器可以检测到物体是否发生碰撞。碰撞体是一类物理组件，只有碰撞体与刚体一起添加到交互对象上才能触发碰撞。如果两个刚体相互碰撞，除非两个对象有碰撞体时物理引擎才会计算碰撞，否则无法触发，并且在物理模拟中，没有碰撞体的刚体会彼此相互穿过。当主角与其他 GameObject 发生碰撞时，需要做一些特殊的事情，例如，子弹击中敌人，敌人就得执行一系列的动作，这时，就需要检测到碰撞现象，即碰撞检测。为了完整地了解这两种方式，必须理解以下概念：碰撞器是一群组件，它包含了很多种类，如 Box Collider 和 Capsule Collider 等，这些碰撞器应用的场合不同，但都必须添加到 GameObject 上。在需要进行碰撞检测的物体上添加触发器（Is Trigger）属性，即可进行触发检测。在 Unity 3D 中，主要有以下接口函数来处理这两种碰撞检测。

触发信息检测如下。

MonoBehaviour.OnTriggerEnter(Collider other)：当进入触发器。

MonoBehaviour.OnTriggerExit(Collider other)：当退出触发器。

MonoBehaviour.OnTriggerStay(Collider other)：当逗留触发器。

碰撞信息检测如下。

MonoBehaviour.OnCollisionEnter(Collision collisionInfo)：当进入碰撞器。

MonoBehaviour.OnCollisionExit(Collision collisionInfo)：当退出碰撞器。

MonoBehaviour.OnCollisionStay(Collision collisionInfo)：当停留在碰撞器。

以上六个接口都是 MonoBehaviour 的函数，由于新建的脚本都继承 MonoBehaviour 这个类，所有的脚本里面可以复写以上六个函数。

- **碰撞检测**

仅当两个物体都带有碰撞器（Collider），并且其中一个物体还必须带有 Rigidbody 刚体属性时，才可以进行有效的碰撞检测。在 Unity 3D 引擎中，能检测碰撞发生的方式有两种，一种是利用碰撞器，另一种则是利用触发器。碰撞器包含了很多种类，如 Box Collider（盒碰撞体）、Mesh Collider（网格碰撞体）等，这些碰撞器应用的场合不同，但都必须添加到 GameObject 上。

利用触发器进行碰撞检测只需要在 Inspector 面板中的碰撞器组件中勾选 Is Trigger 属性复选框即可。如果既要检测到物体的接触又不想让碰撞检测影响物体移动或要检测一个物件是否经过空间中的某个区域时可以用到触发器。

**• 射线检测**

在开发中，尤其是跟模型交互时，都会用到射线检测。射线是3D世界中一个点向一个方向发射的一条无终点的线，在发射轨迹中与其他物体发生碰撞时，它将停止发射。射线应用范围比较广，多用于碰撞检测。

**• 相关 API**

Ray Camera.main.ScreenPointToRay(Vector3 pos)：返回一条射线 Ray 从摄像机到屏幕指定一个点。

Ray Camera.main.ViewportPointToRay(Vector3 pos)：返回一条射线 Ray 从摄像机到视口(视口之外无效)指定一个点。

**• Ray 射线类**

RaycastHit：光线投射碰撞信息。

bool Physics. Raycast ( Vector3 origin, Vector3 direction, float distance, int layerMask)：当光线投射与任何碰撞器交叉时为真，否则为假。

bool Physics.Raycast(Ray ray, Vector3 direction, RaycastHit out hit, float distance, int layerMask)：在场景中投下可与所有碰撞器碰撞的一条光线，并返回碰撞的细节信息。

bool Physics.Raycast(Ray ray, float distance, int layerMask)：当光线投射与任何碰撞器交叉时为真，否则为假。

bool Physics. Raycast ( Vector3 origin, Vector3 direction, RaycastHit outhit, float distance, int layerMask)：当光线投射与任何碰撞器交叉时为真，否则为假。

**注意**：如果从一个球形体的内部到外部用光线投射，返回为假。

**• 参数理解**

Origin：在世界坐标中射线的起始点。

direction：射线的方向。

distance：射线的长度。

hit：使用 C# 中 out 关键字传入一个空的碰撞信息类，碰撞后赋值。可以得到碰撞 transform、rigidbody、point 等信息。

layerMask：只选定 layerMask 层内的碰撞器，其他层内碰撞器忽略。选择性地碰撞。

RaycastHit[] RaycastAll(Ray ray, float distance, int layerMask)：投射一条光线并返回所有碰撞，也就是投射光线并返回一个 RaycastHit[]结构体。

以摄像机所在位置为起点，创建一条向下发射的射线，创建射线如图 3.5 所示。

应用前面所介绍的射线检测来制作一个交互功能演示原型，这样的交互功能演示原型很常见，应用也十分广泛，只要是隔空的交互效果，就可以采用这样的方式。

(1) 新建一个场景，在 Hierarchy 面板下执行 Create→3D Object→Cube 命令，创建 Cube 并布置场景，创建 Cube 如图 3.6 所示。

(2) 在该文件中主要练习射线检测对于基本交互的应用，在 3D 场景中，当抓取物体，或者是交互展馆中，人靠近文物都会有相应的提醒和提示，那么这样的一些应用如果用碰撞检测或者是触发检测就会不合乎交互规则。当人碰撞到了文物或者已经穿过才有反馈显然是不对的，这时在虚拟的环境中要应用射线检测来实现这一功能。新建一个脚本制作射线检测。

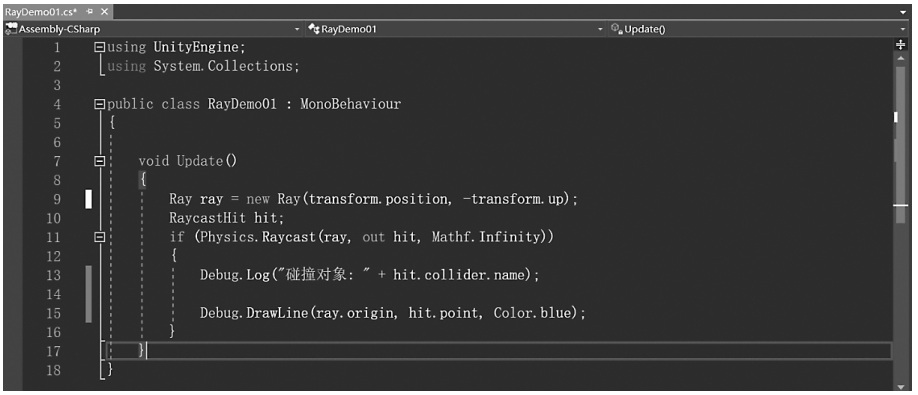


图 3.5　创建射线

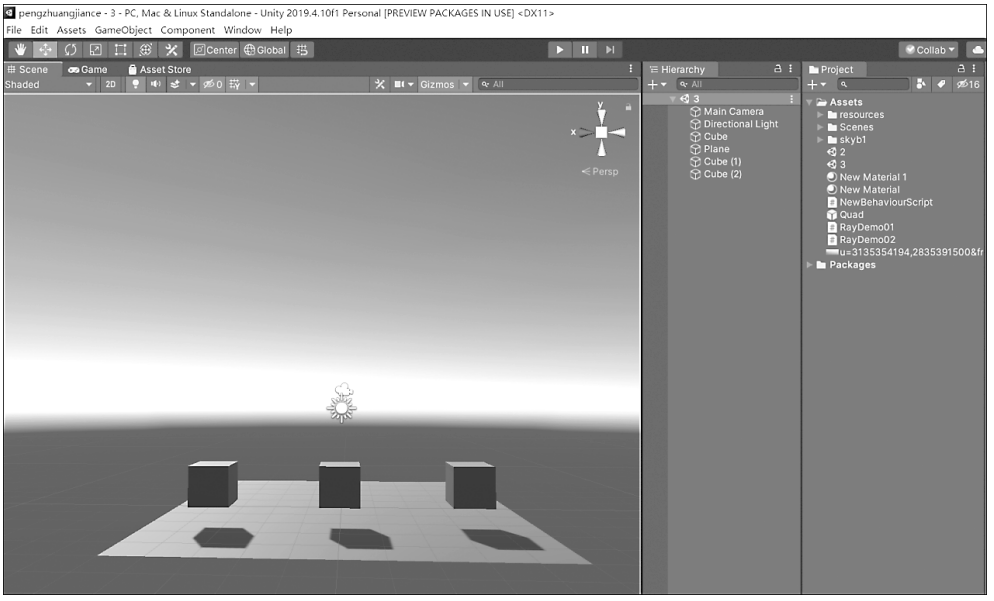


图 3.6　创建 Cube

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class NewBehaviourScript : MonoBehaviour {
    public GameObject prefab;
    private object danhen;
    void Update () {
        if (Input.GetMouseButton(0))
        {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;                    //射线信息
            if(Physics.Raycast(ray,out hit, 100f))
            {
                Vector3 weizhi = hit.point;    //射线接触物体的位置点
                //实例化子弹
```

```
                GameObject danhen = GameObject.Instantiate(prefab, weizhi,
Quaternion.identity) as GameObject;
                //让子弹弹痕朝向屏幕,广告版模式
                danhen.transform.LookAt(hit.point - hit.normal);
                danhen.transform.Translate(Vector3.back * 0.01f);
            }
        }
    }
}
```

(3) 将脚本拖动到摄像机 Main Camera 上,运行文件,将弹痕的 prefab 指定到实例化对象位置。实例化弹痕如图 3.7 所示。

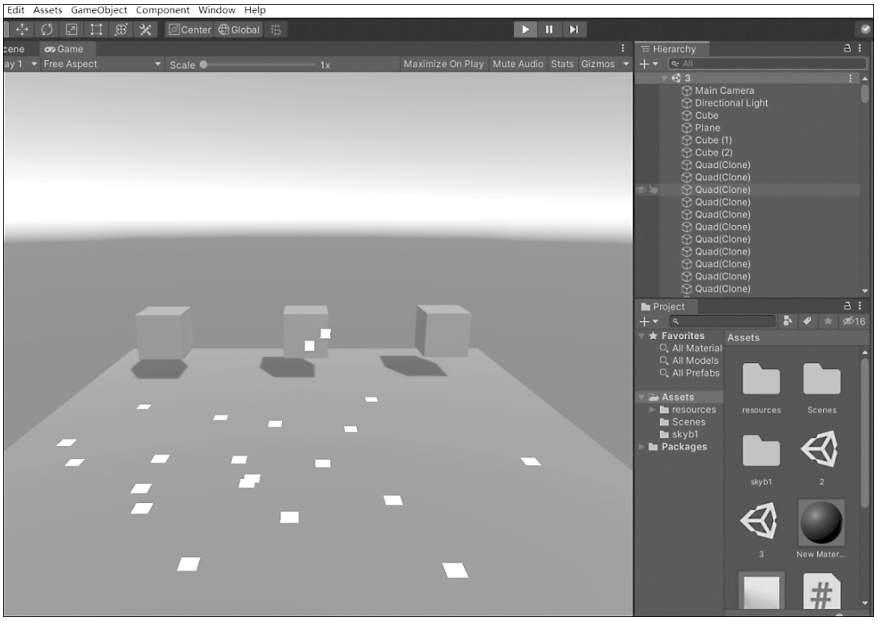


**图 3.7　实例化弹痕**

(4) 鼠标单击任意场景,可以出现预先设定好的弹痕特效,这种交互反馈可以替换成任何想要设计的效果,但是射线检测的方法是不变的,需要注意以下几点。

实例化子弹:

```
GameObject danhen=GameObject.Instantiate(prefab, weizhi, Quaternion.identity)
as GameObject;
```

让子弹弹痕朝向屏幕:

```
danhen.transform.LookAt(hit.point - hit.normal);
```

射线的运用:

```
Ray ray =Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hitInfo;
```

### 3.2.2 Unity 3D 声音可视化交互

听觉感知传递的信息仅次于视觉，可人们一般都低估了这些信息。人类的听觉可以感知大量的信息，但被视觉关注掩盖了许多。听觉所涉及的问题和视觉一样，即接受刺激，把它的特性转换为神经兴奋，并对信息进行加工，然后传递到大脑。声波在空气中的振动传播的特性、音调与声波的频率有关，低频能产生低调的声音，高频能产生高调的声音。响度指在频率一定的情况下声波的振幅。音色与发声的材料有关，不同的乐器可以产生相同频率和振幅的声波，但音色不同。人类能够听到频率为 20Hz～20kHz 的声音，其中在 1000Hz～4000Hz 范围内听觉的感受性最高。500Hz 以下和 5000Hz 以上的声音，强度很大时才能被听到。响度超过 140dB(分贝)时，所引起的不再是听觉而是痛觉。人类可以辨认的语音频率范围是 260Hz～5600Hz。听觉系统把输入分为如下三类：

噪声和可以忽略的不重要的声音；

被赋予意义的非语言声音，如动物的叫声；

用来组成语言的有意义的声音。

听觉系统就像视觉系统一样，利用以前的经验来解释输入。Lindsay PH 和 Norman DA 的“材料-驱动”(Data-Driven)和“概念-驱动”(Conceptually-Driven)过程。材料-驱动指的是对言语材料在感知水平上进行的加工过程，它是由下而上的分析过程。概念-驱动则是在理解水平上进行的加工过程，它是由上而下(从最高的结构概念开始)的分析过程。人类听觉系统对声音的解释可帮助设计人机交互系统中的声音的合理运用，在接下来的案例中，根据 Unity 3D 中声音的运用来具体分析其在虚拟环境中的体现。

**• 支持格式**

在虚拟环境中，一般存在两种音乐，一种是时间较长的背景音乐，另一种是时间较短的音效(如按钮单击、开枪音效等)。

Unity 3D 支持如下几种音乐格式。

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。

**• Unity 3D 中播放音乐**

Unity 3D 对声音进行了封装，以下三个组件是播放声音的基本组件。

首先是 Audio Listener 组件。一般创建场景时在主摄像机上就会带有这个组件，该组件只有一个功能，就是监听当前场景下的所有音效的播放并将这些音效输出，如果没有这个组件，则不会发出任何声音。但是不需要创建多个该组件，一般场景中只需要在任意的 GameObject 上添加一个该组件就可以了，但是要保证这个 GameObject 不被销毁，所以一般按照 Unity 3D 的做法，在主摄像机中添加即可。

其次是 Audio Source 组件。控制一个指定音乐播放的组件，可以通过属性设置来控制音乐的一些效果，具体设置可以查看官方的文档(官方文档的网址详见前言二维码)。

下面列出一些常用的属性。

AudioClip：声音片段，还可以在代码中去动态地截取音乐文件。

Mute：是否静音。

Bypass Effects：是否打开音频特效。

Play On Awake：开机自动播放。

Loop：循环播放。

Volume：声音大小，取值范围为 0.0～1.0。

Pitch：播放速度，取值范围为－3～3，设置 1 为正常播放，小于 1 为减慢播放，大于 1 为加速播放。

最后一个是 AudioClip 组件。当把一个音乐导入到 Unity 3D 中，这个音乐文件就会变成一个 AudioClip 对象，既可以直接将其拖动到 AudioSource 的 AudioClip 属性中，也可以通过 Resources 或 AssetBundle 进行加载，加载出来的对象类型就是 AudioClip。

**• 播放声音的简单案例**

不需要一行代码，即可加载声音，并且循环播放。新建一个场景，给 Main Camera 添加一个 Audio Source 组件，在 Inspector 面板上单击 Add Component 按钮输入 Audio Source 并选择，将准备好的音乐文件拖动到 AudioClip 属性上，勾选 Loop 使其可以进行循环播放。按照图 3.8 所示，单击▶按钮运行程序即可实现加载的声音循环播放。

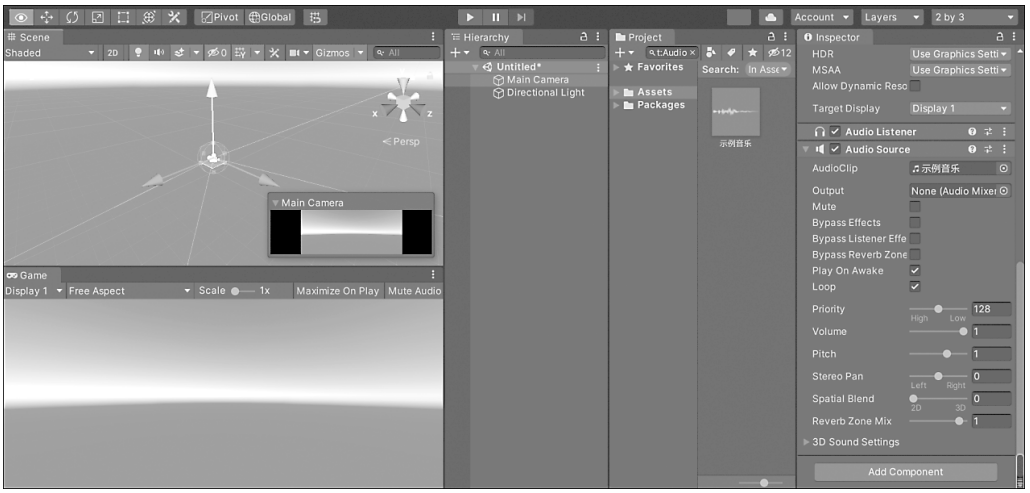


**图 3.8　播放声音组件**

**• 音效效果**

Unity 3D 中把声音分为三个方法进行管理，可以实现 3D 音效效果。在满足基本的交互情况下，可以更好地提升用户体验。将 Audio Listener 看作一双耳朵的话就可以很好地理解什么是 3D 音效效果了，Unity 3D 会根据 Audio Listener 对象所在的 GameObject 和 Audio Source 对象所在的 GameObject 判断距离和位置来模拟真实世界中的音量近大远小的效果。首先，找到导入的音乐文件，必须设置为 3D 音乐，默认就是。当然如果是 2D 音乐就不会有近大远小的效果了，音频文件设置如图 3.9 所示。

在很多情况下，为了使交互作品更加完善还会为作品添加立体音效，3D 声音加载只需要以下的几步操作。

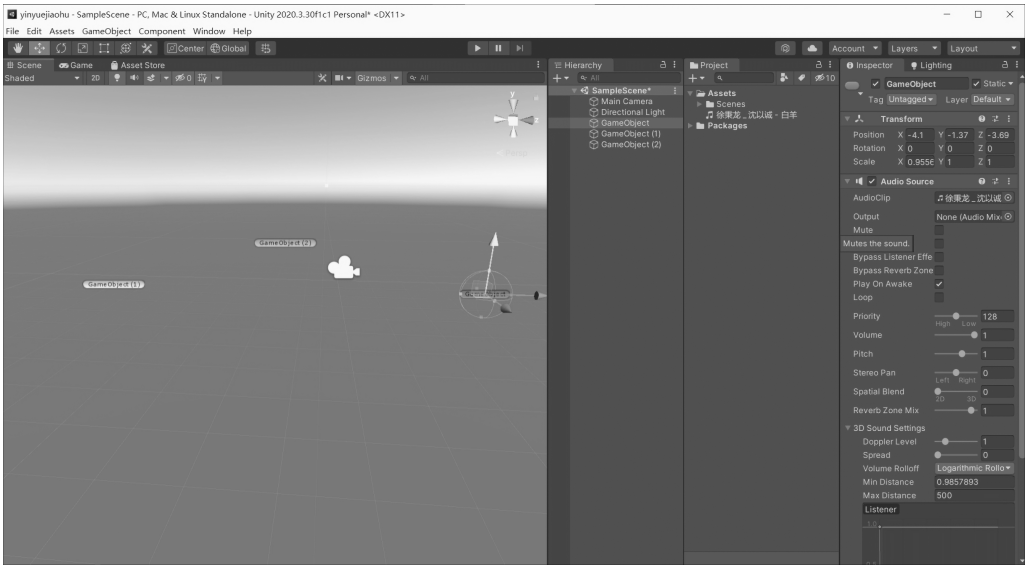


**图 3.9　音频文件设置**

（1）创建一个新的场景，在场景添加三个 GameObject，在 Hierarchy 面板下执行 Create→Create Empty 命令，摆放位置，创建空对象（GameObject）如图 3.10 所示。

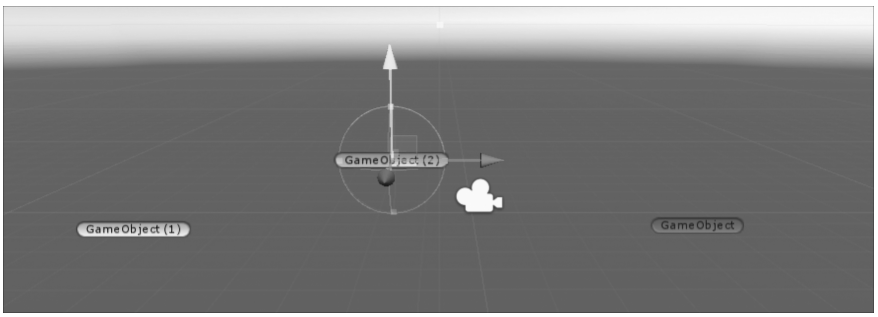


**图 3.10　创建空对象（GameObject）**

（2）单击 Audio Listener 组件右上角的设置按钮选择 Remove Component 选项，移除主摄像机上的 Audio Listener 组件。在 GameObject 的 Inspector 面板上单击 Add Component 按钮，输入 Audio Listener 和 Audio Source，给 GameObject 添加一个 Audio Listener 组件，其他两个添加 Audio Source 组件。

（3）将下载好的 3D 音乐拖动到 GameObject（1）和 GameObject（2）的 Audio Source 上，单击▶按钮，可以感受到两个音响之间移动的效果。

- **通过代码控制声音的播放和控制**

```
private AudioSource _audioSource;
```

首先需要声明加载的 Audio Source 组件，声明完成后在 start 方法中进行如下脚本的编写即可。

```
_audioSource = this.gameObject.AddComponent<AudioSource>();
AudioClip audioClip = Resources.Load<AudioClip>("bgm");
_audioSource.loop = true;
_audioSource.clip = audioClip;
_audioSource.Play();
```

**• 声音交互高级案例——声乐可视化交互**

声音能够增加虚拟作品中的真实感，无论在虚拟现实作品还是人机交互系统中，都是在和无形的环境、人物打交道，声音能够增加作品的趣味性，频率是声音的物理特性，而音调则是频率的主观反映。一般地，音调的高低与频率的高低一致。频率不变，强度的变化对音调稍有影响，强度增大时，低频率音调显得更低，而高频率音调显得更高。例如，8192Hz 的声音在 100dB 强度下所产生的音调要比 80dB 时高，而 128Hz 的声音在 100dB 时所产生的音调要比 70dB 时低。音调的单位为美(Mel)。频率 1000Hz 听阈以上 40dB(感觉级)的纯音所产生的音调为 1000Mel，音调比它高 1 倍为 2000Mel(大致相当于 3000Hz 纯音的音调)。当频率增加 1 倍时称为 1 个倍频程(Octave)，相当于音乐中音调增高一个八度音阶。在 Unity 3D 中，可以将声音进行可视化的显示。

沉浸，就是让用户专注在由设计者营造的当前目标情境下感到愉悦和满足，而忘记真实世界的情境。音频是人机交互的重要组成部分，音频能够营造更好的沉浸感。在 Unity 3D 开发中也是不可或缺的元素，是构成人机交互项目背景音乐、特效音乐等内容必需的资源。音乐不仅能渲染出用户参与作品时的氛围，还能增加用户对作品的认知度。不同的音乐，可以根据其特点形成不同的可视化图形。具体实现步骤如下。

(1) 创建空的场景，执行 Assets→Import Package→Custom Package 命令，导入 Koreographer 插件和音乐素材，搭建场景，为场景中的 Ball 添加刚体组件，在 Ball 的 Inspector 面板上单击 Add Component 按钮输入 Rigidbody 添加刚体组件。在 Assets 的 Project 面板中的空白处右击，在弹出的快捷菜单中执行 Create→Folder 命令，新建一个 MyDemo 文件夹，在 MyDemo 文件夹下的 Project 面板中的空白处右击，在弹出的快捷菜单中执行 Create→Folder 命令，再创建 Koreographer、Scenes、Scripts、Materials 文件夹分别存放音轨、场景、脚本和材质。导入素材如图 3.11 所示。

(2) 双击打开 MyDemo 文件夹，右击 MyDemo 文件夹面板空白处，在弹出的快捷菜单中执行 Create→Koreographer 命令，创建 Koreography，右击 Koreography，在弹出的快捷菜单中选择 Rename 选项更改名称为 MyKoreography，在 MyDemo 文件夹面板空白处右击，在弹出的快捷菜单中执行 Create→KoreographerTrack 命令，创建 KoreographyTrack，右击 KoreographyTrack，在弹出的快捷菜单中选择 Rename 选项更改名称为 BallTrack，并在 KoreographyTrack 的 Inspector 面板上把 Event ID 修改为 BallTrack，与前面的 KoreographerTrack 修改完成的名称保持一致。创建 Koreography 如图 3.12 所示。

(3) 将素材音乐拖动到 MyKoreography 的 Inspector 面板中的 M Source Clip 上，把 BallTrack 拖动到 MyKoreography 的 Inspector 面板中的 M Tracks 上，单击 Open In Koreography Editor 按钮，打开音乐编辑器。Inspector 面板如图 3.13 所示。

(4) 在音轨上根据音乐节奏，使用 Select、Draw、Clone 等方法添加事件，也可以直接在节拍处双击，添加事件。

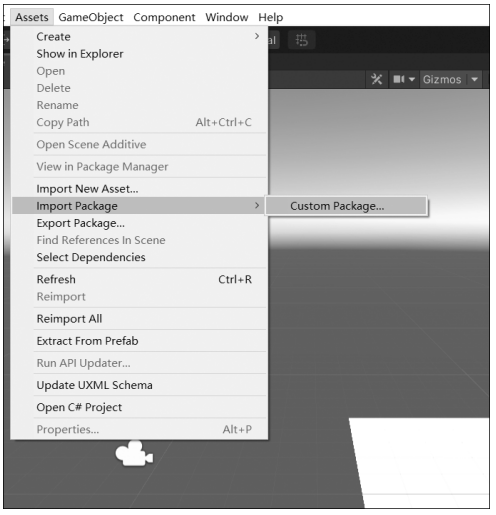


图 3.11　导入素材

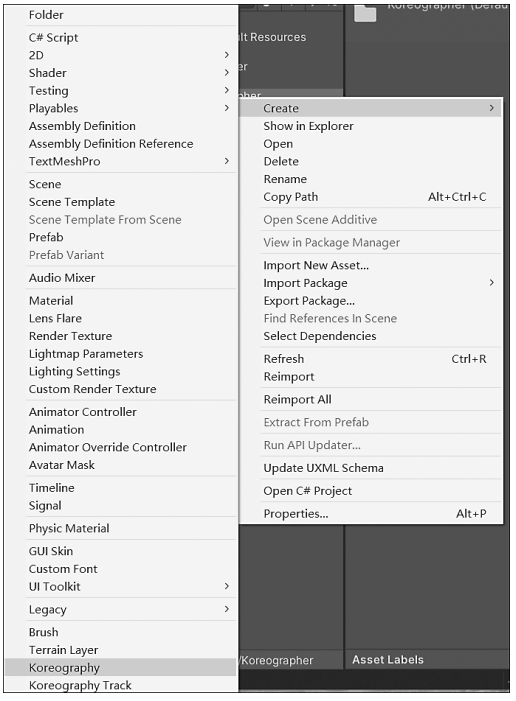


图 3.12　创建 Koreography

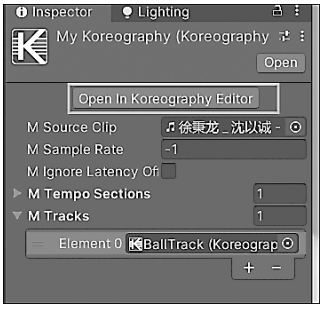


图 3.13　Inspector 面板

快捷键如下。

A：Select 事件。

S：Draw 事件。

D：Clone 事件。

E：播放时添加事件。

Space：播放或暂停。

(5) 创建脚本 BallTrack，实现小球根据音乐节奏跳动，引入命名空间 SonicBloom.Koreo。在小球的 Inspector 面板上单击 Add Component 按钮分别输入 Sphere Collider 和 BallTrack，给小球添加 Sphere Collider 组件和 BallTrack 脚本。然后将 BallTrack 的 ID 名称拖动到 Event ID 并设置小球的跳动速度。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using SonicBloom.Koreo;
public class BallTest : MonoBehaviour
{
    private Rigidbody rigidbodyCom;
    public string eventID;
    public float jumpSpeed;
    void Start()
    {
        rigidbodyCom = GetComponent<Rigidbody>();
        Koreographer.Instance.RegisterForEvents(eventID, BallJump);
    }
    private void BallJump(KoreographyEvent koreographyEvent)
    {
        Vector3 vel = rigidbodyCom.velocity;
        vel.y = jumpSpeed;
        rigidbodyCom.velocity = vel;
    }
}
```

(6) 在 Hierarchy 中执行 Create→UI→EventSystem 命令，创建 EventSystem，并执行 Create→Create Empty 命令，创建一个空物体，给空物体更改名称为 MusicPlayer。选中 MusicPlayer 选项，在其 Inspector 面板上单击 Add Component 按钮添加 Koreographer 组件、Audio Source 组件和 Simple Music Player 组件，把 MyKoreography 指定拖动给 Simple Music Player 组件中的 Koreography 卡槽。然后运行，就可以实现小球随音乐节奏跳动。音乐管理如图 3.14 所示。

(7) 执行 Assets→Koreographer→Dermos→Prefabs→Particle System 命令，拖入场景面板。在 MyDemo 的 Koreographer 文件夹下右击，在弹出的快捷菜单中执行 Create→KoregrophyTrack 命令，新建一个 KoregrophyTrack，更改名称为 ParticleTrack，同样修改 Event ID 为 ParticleTrack，与其名称保持一致。将新创建的 ParticleTrack 拖动到 MyKoreography 的 Tracks 上，单击 Open In Koreography Editor 按钮，打开编辑器。MyKoreography 设置

如图 3.15 所示。


图 3.14　音乐管理

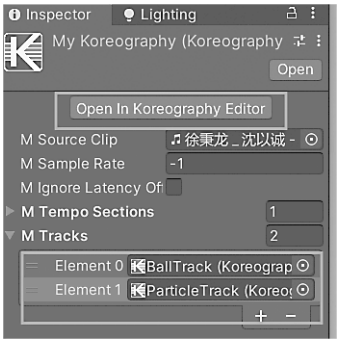


图 3.15　MyKoreography 设置

（8）在 Koreography Editor 中，将 Track to Edit 修改为这次要更改的粒子系统音轨 ParticleTrack，在合适的节拍处添加事件。Koreography Editor 如图 3.16 所示。

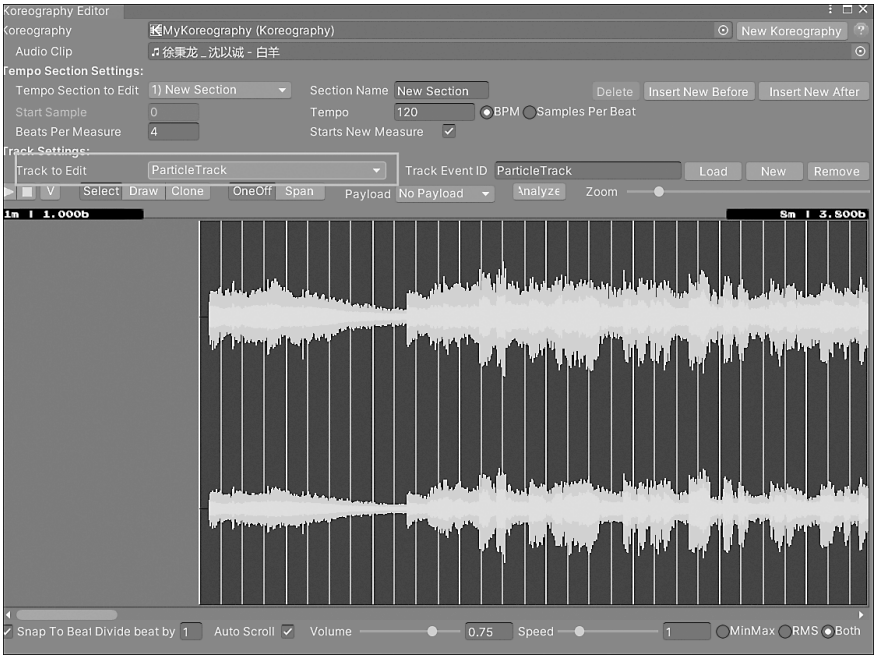


图 3.16　Koreography Editor

（9）创建脚本 ParticleTrack，控制粒子的喷射数量，同样引入命名空间 SonicBloom.Koreo。单击 Add Component 按钮输入 ParticleTrack，添加 ParticleTrack 脚本。然后将 ParticleTrack 的 ID 名称拖动到 Event ID 并设置粒子的喷射速度。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using SonicBloom.Koreo;
public class ParticleTrack : MonoBehaviour
{
    public string eventID;
    public float particlePerBeat = 100;
    private ParticleSystem particleSystemCom;
    void Start()
    {
        particleSystemCom = GetComponent<ParticleSystem>();
        Koreographer.Instance.RegisterForEvents(eventID, CreatParticle);
    }
    private void CreatParticle(KoreographyEvent KoreographyEvent)
    {
        int particleCount = (int)(Koreographer.GetBeatTimeDelta() * particlePerBeat);
        particleSystemCom.Emit(particleCount);
    }
}
```

以上，通过 Koreographer 组件，实现了根据音乐节拍添加游戏事件。

### 3.2.3　Unity 3D 摄像机交互

在很多大屏幕的互动项目中，会用到摄像机的交互，用摄像机调取用户的输入，和场景中的物体发生互动。在 Unity 3D 中创建一个摄像机后，除了默认带有一个 Transform 组件外，还会附带 Flare Layer、GUI Layer、Audio Listener 等组件。

Unity 3D 摄像机包含的参数如下。

Clear Flags：清除标记。决定屏幕的哪部分将被清除。一般用户使用多台摄像机来描绘不同游戏对象的情况，有 3 种模式选择。

Skybox：天空盒。默认模式。在屏幕中的空白部分将显示当前摄像机的天空盒。如果当前摄像机没有设置天空盒，会默认用 Background 颜色。

Solid Color：纯色。选择该模式，屏幕上的空白部分将显示当前摄像机的 Background 颜色。

Depth only：仅深度。该模式用于游戏对象不希望被裁剪的情况。

Dont Clear：不清除。该模式不清除任何颜色或深度缓存。其结果是，每一帧渲染的结果叠加在下一帧之上。一般与自定义的 Shader 配合使用。

Background：背景。设置背景颜色。在镜头中的所有元素渲染完成且没有指定天空盒的情况下，将设置的颜色应用到屏幕的空白处。

Culling Mask：剔除遮罩，选择所要显示的 Layer。

Projection：投射方式。

Perspective：透视。摄像机将用透视的方式来渲染游戏对象。

Field of view：视野范围。用于控制摄像机的视角宽度以及纵向的角度尺寸。

Orthographic：正交。摄像机将用无透视的方式来渲染游戏对象。

Size：大小。用于控制正交模式摄像机的视口大小。

Clipping Planes：剪裁平面。摄像机开始渲染与停止渲染之间的距离。

Near：近点。摄像机开始渲染的最近的点。

Far：远点。摄像机开始渲染的最远的点。

ViewportRect：标准视图矩形。用四个数值来控制摄像机的视图将绘制其在屏幕上的位置和大小，使用的是屏幕坐标系，数值范围为0～1。坐标系原点在左下角。

Depth：深度。用于控制摄像机的渲染顺序，较大值的摄像机将被渲染在较小值的摄像机之上。

Rendering Path：渲染路径。用于指定摄像机的渲染方法。

Use Player Settings：使用 Project Settings→Player 中的设置。

Vertex Lit：顶点光照。摄像机将所有的游戏对象作为顶点光照对象来渲染。

Forward：快速渲染。摄像机将所有游戏对象将按每种材质一个通道的方式来渲染。

Deferred Lighting：延迟光照。摄像机先对所有游戏对象进行一次无光照渲染，用屏幕空间大小的缓冲器保存几何体的深度、法线以及高光强度，生成的缓冲器将用于计算光照，同时生成一张新的光照信息缓冲器。最后所有的游戏对象会被再次渲染，渲染时叠加光照信息 Buffer 的内容。

Target Texture：目标纹理。用于将摄像机视图输出并渲染到屏幕上。一般用于制作导航图或者画中画等效果。

HDR：高动态光照渲染。用于启用摄像机的高动态范围渲染功能。

从上述参数列举可知 Unity 3D 摄像机的参数很多，所以在运用的过程中，根据不同的案例进行不一样参数的选取，例如，在调取摄像机案例中设置如下。

（1）创建工程文件，布置场景，一个 Camera，一个 Cube，一个 Canvas，在 Hierarchy 面板执行 Create→Camera 命令，新建 Camera，执行 Create→3D→3D Object 命令，新建 Cube，执行 Create→UI→Canvas 命令，新建 Canvas，同时创建测试的按钮和 UI，右击 Canvas，在弹出的快捷菜单中执行 Create→UI→Button 命令，右击 Canvas，在弹出的快捷菜单中执行 Create→UI→Image 命令，自行根据需要进行创建即可，搭建场景如图 3.17 所示。

（2）由于 Unity 3D 内置了摄像机的很多方法，所以实现摄像机的调取并不是很难。只要找到相应的摄像机及其组件即可调取摄像机并修改其属性。执行 Create→C# Script 命令，创建脚本 CamController，在脚本中定义以下变量。

```
public GameObject cam1;                          //第一个摄像机
public GameObject cam2;                          //第二个摄像机
public Button btn;                               //切换摄像机的按钮
```

（3）在 Start 方法中进行变量的初始化。

```
void Start () {
    cam1.SetActive(true);
    cam2.SetActive(false);                       //隐藏第二个摄像机
    btn.onClick.AddListener(changeCamera);       //监听切换按钮
}
```

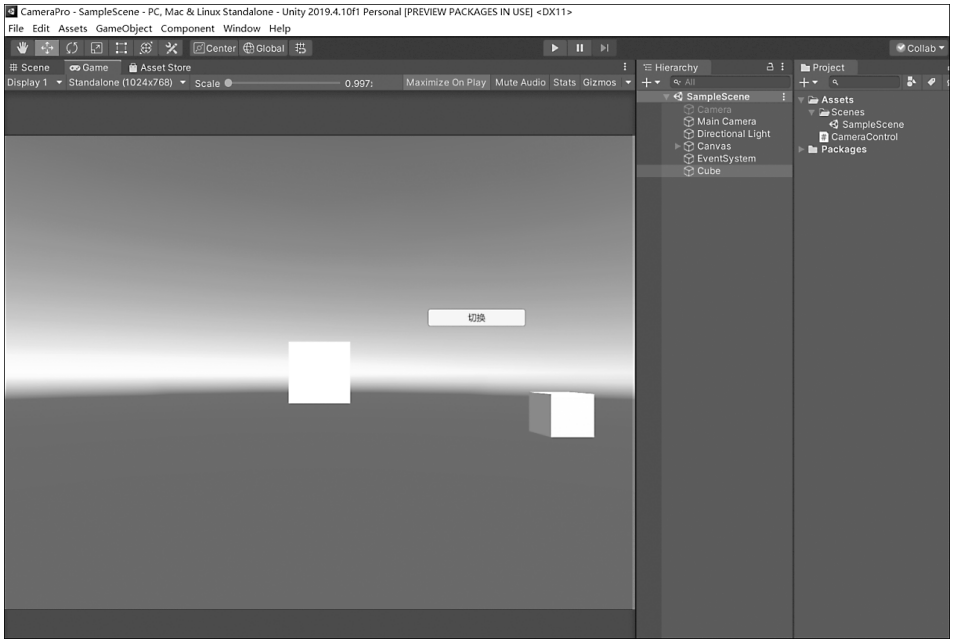


**图 3.17　搭建场景**

(4) 创建 changeCamera()函数,在其中判断摄像机的显隐状态并实现摄像机的切换。

```
void changeCamera()
{
    if (cam1.activeSelf == true)
    {
        cam1.SetActive(false);
        cam2.SetActive(true);
    }
    else
    {
        cam1.SetActive(true);
        cam2.SetActive(false);
    }
}
```

(5) 在 Update()函数中使用 Input()函数调用键盘按键,改变摄像机的视野范围。

```
void Update () {
    if (Input.GetKey(KeyCode.S))              //S 键被按下的状态
    {
        if (cam1.activeSelf == true)
        {
            cam1.GetComponent<Camera>().fieldOfView += Time.deltaTime * 10;
        }
        else
        {
```

```
            cam2.GetComponent<Camera>().fieldOfView +=Time.deltaTime * 10;
        }
    }else if (Input.GetKey(KeyCode.W))
    {
        if (cam1.activeSelf == true)
        {
            cam1.GetComponent<Camera>().fieldOfView -=Time.deltaTime * 10;
        }
        else
        {
            cam2.GetComponent<Camera>().fieldOfView -=Time.deltaTime * 10;
        }
    }
}
```

（6）运行程序，可通过 Button 按钮切换摄像机，也可通过按 W、S 键调节视野范围。Button 功能和视野范围如图 3.18 所示。

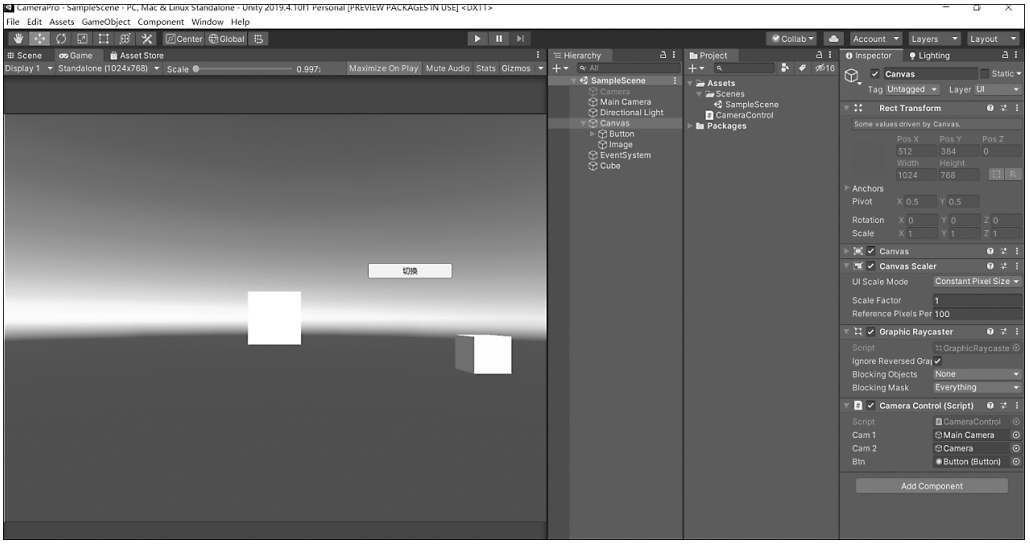


**图 3.18　Button 功能和视野范围**


## 3.3　高级环境交互

### 3.3.1　Unity 3D 自动寻路 Navmesh 之入门

Unity 3D 中的导航网格 Navmesh 广泛应用于动态物体实现自动寻路的功能，属于人工智能的一种，通过此功能可以使智能 AI 自行绕过障碍或翻越墙体等，最终到达目标地点或找到目标对象，是一种既方便，又简单，同时还很实用的功能。根据下面的案例，来简单地了解一下 Navmesh。

(1) 首先导入场景和人物素材,搭建一个场景,并选中所有地面和建筑,在 Inspector 面板中选择静态(Static)下拉选项中的 Navigation Static 选项,Navigation Static 静态类型如图 3.19 所示。

(2) 在 AI 人物的身上单击 Add Component 按钮添加 Nav Mesh Agent 寻路组件和控制脚本。Nav Mesh Agent 组件如图 3.20 所示。

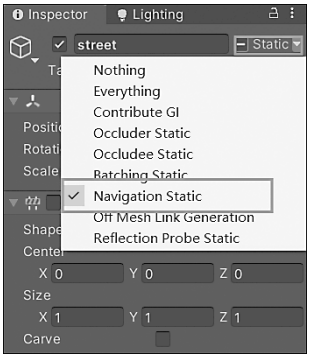


图 3.19　Navigation Static 静态类型

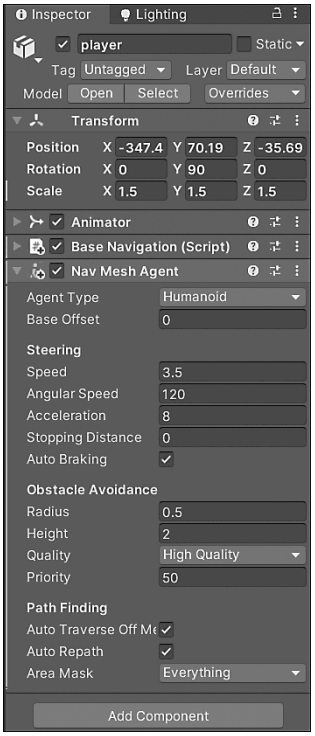


图 3.20　Nav Mesh Agent 组件

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class BaseNavigation : MonoBehaviour
{
    public Transform target;
    private NavMeshAgent nav;
    private void Start()
    {
        nav = this.GetComponent<NavMeshAgent>();
    }
    private void Update()
    {
        if(target&&nav)
        {
            nav.SetDestination(target.transform.position);
```

```
            }
        }
    }
```

（3）创建目标物体，并拖动到 BaseNavigation 脚本中 target 的卡槽中。

（4）在顶层菜单栏中执行 Windows→AI→Navigation 命令，打开 Navigation 面板，单击 Bake 按钮，开始烘焙导航网格。运行整体项目，实现 AI 人物的导航网格自动寻路。烘焙导航网格如图 3.21 所示。

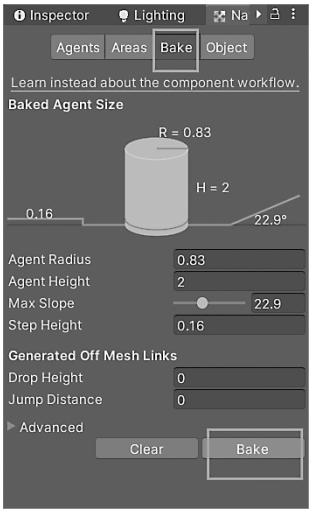


**图 3.21　烘焙导航网格**

（5）新建场景二，将目标物体的位置设置在高处，在场景中制作斜坡。

（6）选中所有地面和建筑，在 Inspector 面板中选中静态(Static)下拉选项中的 Navigation Static 选项，然后单击 Add Compenont 按钮添加 Nav Mesh Agent 寻路组件和控制脚本。单击 Bake 按钮重新烘焙导航网格。运行后，AI 人物可以通过斜坡到达目标位置。重新烘焙效果如图 3.22 所示。

（7）菜单栏上执行 File→New Scene 命令，新建场景三，场景中有红色、蓝色两个 AI 人物，并标记出红色、蓝色两条不同的道路。

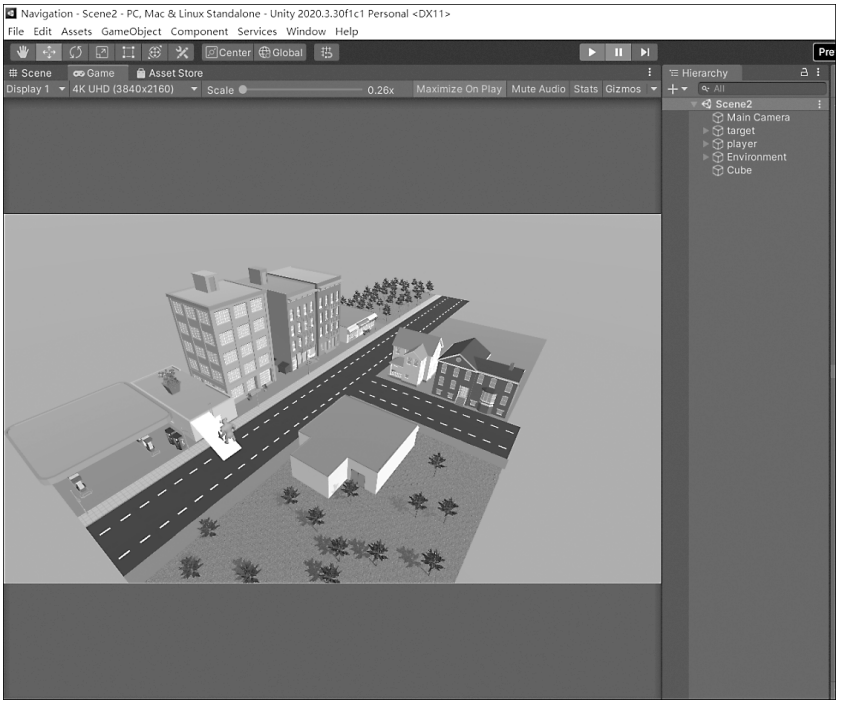


**图 3.22　重新烘焙效果**

（8）在新建场景三内，选中所有地面和建筑，在 Inspector 面板中选中静态(Static)下拉选项中的 Navigation Static 选项，在 Navigation 面板中的 Areas 条件中的卡槽内添加红色