第 3 章

Chapter 3

脚本开发基础

脚本是一款游戏的灵魂，Unity 引擎脚本用来界定用户在游戏中的行为，是游戏制作中不可或缺的一部分。它能实现各个文本的数据交互并监控游戏的运行状态。以往，Unity 引擎主要支持三种语言：C#、JavaScript 以及 Boo。但是选择 Boo 作为开发语言的使用者非常少，而 Unity 公司还需要投入大量资源支持它，这显然非常浪费。所以，在 Unity 5.0 后，Unity 公司放弃了对 Boo 语言的技术支持，在 Unity 2017 之后又摒弃了 JavaScript。本章主要以 C# 语言为例讲解 Unity 引擎脚本的创建、链接方法及脚本编写注意事项，为后续复杂游戏脚本开发打下基础 。

3.1 脚本概述

Unity 引擎的 C# 和微软.Net 家族中的 C# 是同一个语言，语言本身是差不多的。但 Unity 引擎的 C# 是运行在 Mono 平台上的，微软的 C# 则是运行在.Net 平台上，有一些针对 Windows 平台的专用 C# 类库可能无法在 Unity 引擎中使用。因此，编写 Unity 引擎脚本，除了要注意语言自身的语法规则外，还要注意 Unity 引擎开发环境的特性。

为了能运行脚本，最基本的要求是将脚本指定给一个 GameObject 作为它的脚本组件，最简单的方法是将脚本直接拖到 GameObject 对象的 Inspector 视图的空白位置，或者在 Hierarchy 视图中选中 GameObject 对象，将脚本拖向 Hierarchy 视图中的 GameObject 对象，即可完成脚本组件的添加。

Unity 引擎脚本有几个最重要的类，它们是 MonoBehaviour、Transform 和 Rigidbody/Rigidbody2D。MonoBehavior 是所有 Unity 脚本的基类，提供了大部分的 Unity 功能。如果脚本不是继承自 MonoBehavior，则无法将这个脚本作为组件运行。Transform 类是每个 GameObject 都包括的默认组件，它提供了位置变换、旋转、缩放、父物体连接等功能。Rigidbody(2D 版本为 Rigidbody2D)类则主要提供物理功能。

3.2 脚本编写

3.2.1 创建脚本

选择 Unity 引擎菜单栏中的 Assets→Create→C# Script 命令，创建一个空白脚本，将其命名为 Move，如图 3.1 所示。

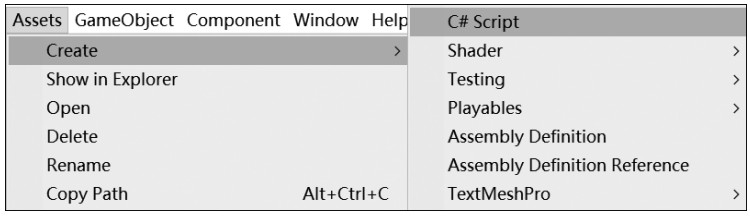

图 3.1　创建一个空白脚本

在 Project 视图中双击 Move，打开脚本，进行脚本编写。在 Update()函数中插入如下代码。函数内的每一帧代码，系统都会去执行。

```
using UnityEngine;
    using System.Collections;
    public class Move : MonoBehaviour {
      void Update () {
        transform.Translate (Input.GetAxis ("Horizontal"), 0, Input.GetAxis
        ("Vertical"));
    }
}
```

其中，Input.GetAxis()函数返回－1～1 的值，在水平轴上，左方向键对应－1，右方向键对应 1。由于目前不需要向上移动摄像机，所以 Y 轴的参数为 0。选择菜单栏中的 Edit→Project Settings→Input Manager 命令，即可修改映射到水平方向和垂直方向的对应名称和快捷键，如图 3.2 所示。

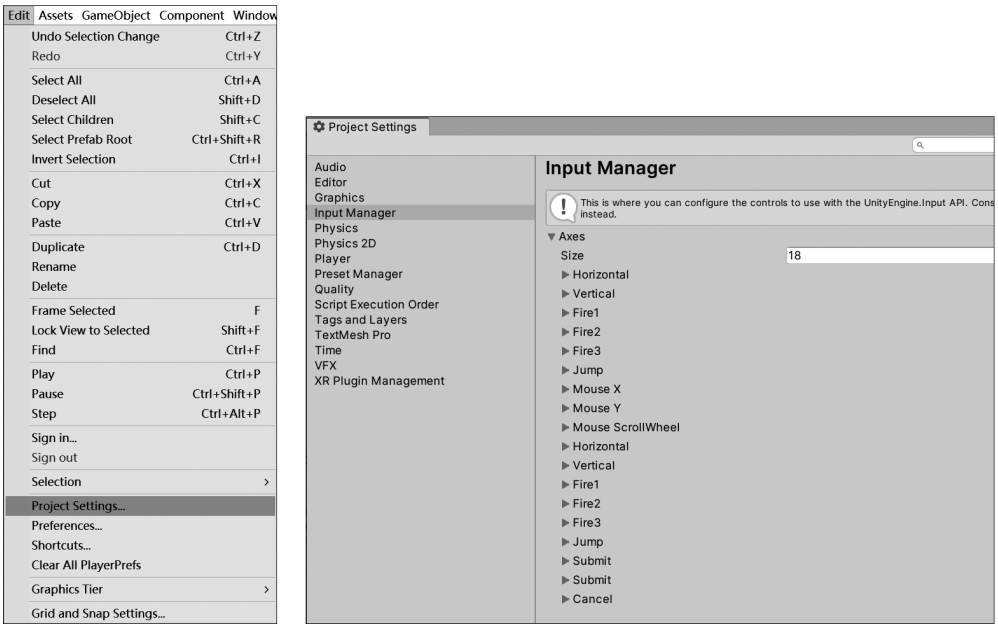

图 3.2　Input Manager 菜单

3.2.2 链接脚本

创建完脚本后,需要将其链接到游戏对象上。在 Hierarchy 视图中单击需要添加脚本的 Main Camera,然后将脚本直接拖到 Main Camera 的 Inspector 视图的空白位置,如图 3.3 所示,Move 脚本就链接到了 Main Camera 上。

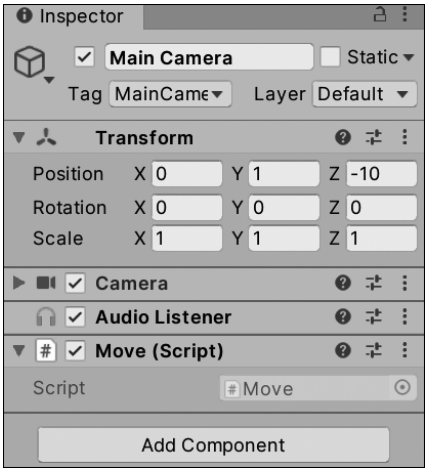

图 3.3 脚本链接

3.2.3 运行脚本

在 Game 视图中单击 Play 按钮进行测试,可以使用键盘上的方向键(水平方向、竖直方向)移动摄像机,运行效果如图 3.4 和图 3.5 所示。

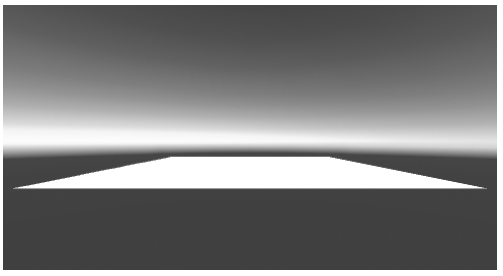

图 3.4 运行测试效果图 1

图 3.5 运行测试效果图 2

3.2.4 注意事项

1. 继承自 MonoBehaviour 类

Unity 所有挂载到游戏对象上的脚本中包含的类都继承自 MonoBehaviour 类。MonoBehaviour 类中定义了各种回调方法,例如 Start、Update 和 FixedUpdate 等。在 Unity 中创建 C# 脚本,系统模板已经包含了必要的定义,如图 3.6 所示。

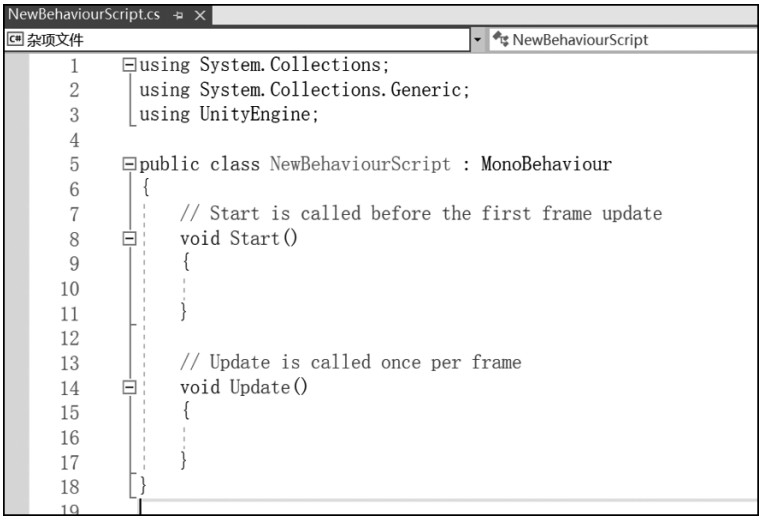

图 3.6　在 Unity 中创建 C# 脚本

2. 使用 Awake 或 Start 函数初始化

在 Unity 中，C# 中用于初始化的脚本代码必须置于 Awake() 或 Start() 方法中。Awake() 和 Start() 的不同之处在于 Awake() 方法是在加载场景时运行，Start() 方法是在第一次调用 Update() 或 FixedUpdate() 方法之前调用。

3. 类名字必须匹配文件名

在 Unity 中，C# 脚本中的类名必须和文件名相同，否则当脚本挂载到游戏对象时，控制台会报错。

4. 只有满足特定情况变量才能显示在 Inspector 视图中

在 Unity 中，C# 脚本只有公有的成员变量才能显示在 Inspector 视图中，而 private 和 protected 类型的成员变量不能显示。如果属性项要在 Inspector 视图中显示，必须是 public 类型的。

5. 尽量避免使用构造函数

在 Unity 中，C# 脚本不需要在构造函数中初始化任何变量，而是使用 Awake 或 Start 方法来实现。在单一模式下使用构造函数可能会导致严重后果，因为它把普通类构造函数封装了，这些构造函数主要用于初始化脚本和内部变量值，这些初始化有随机性，容易引发引用异常。因此，一般情况下尽量避免使用构造函数。

3.3　脚本开发实践项目

3.3.1　移动的立方体

1. 项目构思

在脚本环境测试的实践项目中，需要通过脚本的编写、编译、链接过程实现一个简单的

游戏场景中走动的效果。本项目旨在通过脚本环境编译测试结果使读者熟悉 Unity 引擎脚本开发环境,为后续程序编写打下基础。

2. 项目设计

本项目计划在 Unity 引擎内创建一个简单的 Cube 模型,通过键盘的方向按键控制 Cube 模型的上、下、左、右移动,并能与鼠标交互实现 Cube 模型复制效果,如图 3.7 所示。

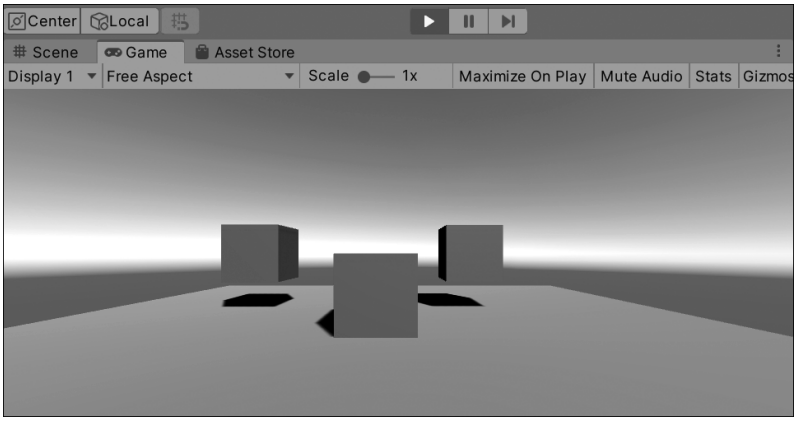

图 3.7 简单场景搭建

3. 项目实施

第 1 步:双击 Unity Hub 图标,并设置其名称以及存储路径,单击右上角“新建”按钮即生成一个新项目,如图 3.8 所示。

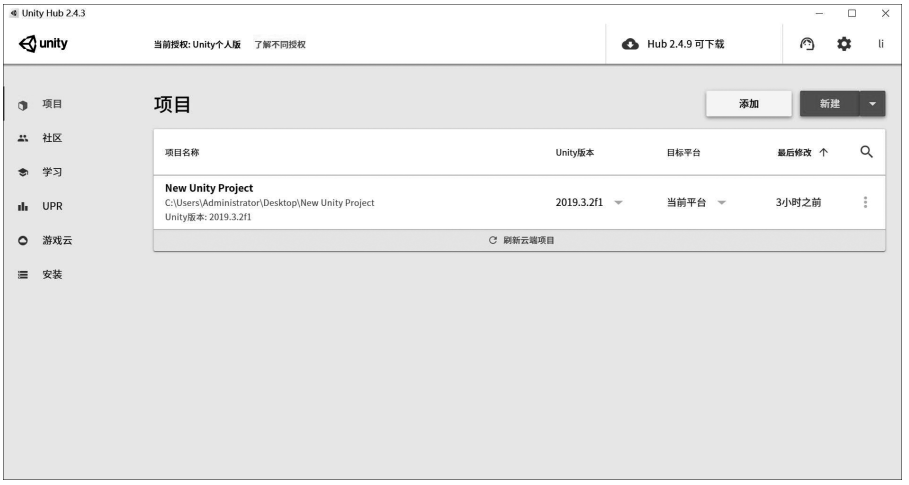

图 3.8 新建项目

第 2 步:选择菜单栏中的 GameObject→3D Object→Plane 命令,在游戏场景中创建一个 Plane 作为地面,如图 3.9 所示。

第 3 步:选择菜单栏中的 GameObject→3D Object→Cube 命令,将它放置在 plane 的中心位置,如图 3.10 所示。

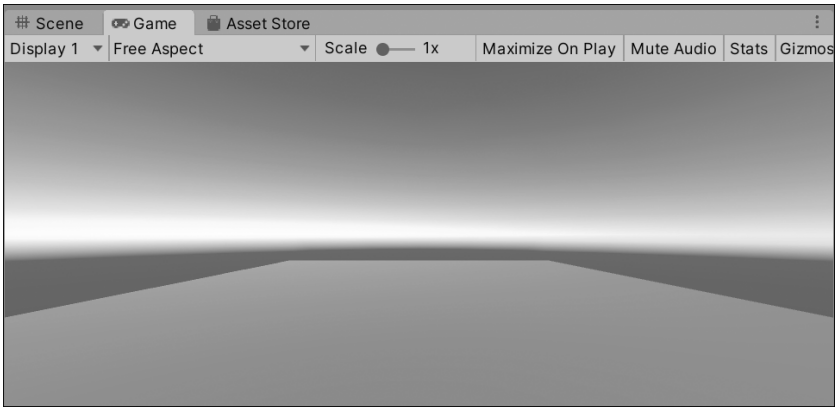

图 3.9　创建 Plane

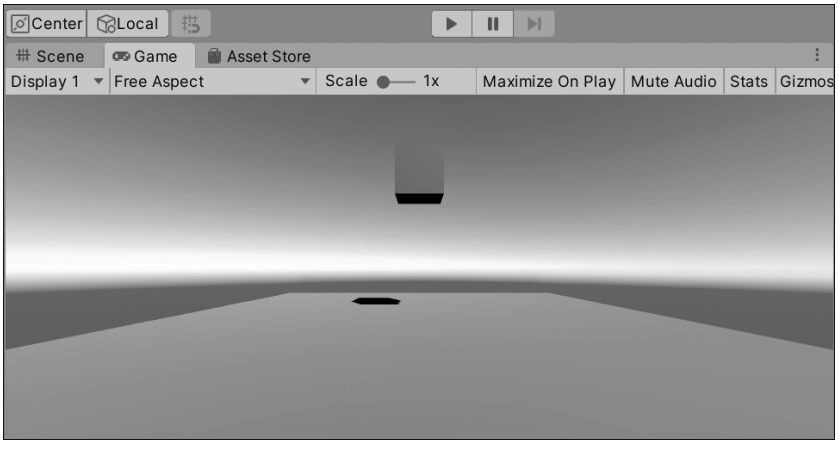

图 3.10　创建 Cube

第 4 步：接下来创建一个空脚本，选择菜单中的 Assets→Create→C# Script 命令，并在项目视图中重命名为 Move。

第 5 步：双击 Move 脚本，输入如下代码。

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

public class Move : MonoBehaviour
{
void Update()
    {
        transform.Translate(Input.GetAxis("Horizontal"), 0, Input.GetAxis
        ("Vertical"));
    }
}
```

Update()函数在渲染一帧之前被调用,这里是大部分游戏行为代码被调用的地方。在脚本中,为了移动一个游戏对象,需要用 transform 来更改它的位置,Translate 函数有 x、y 和 z 共 3 个参数。

第 6 步:保存脚本(快捷键为 Ctrl+S)。

第 7 步:将脚本与主摄像机相连,即将脚本拖到 Hierarchy 视图中的 Main Camera 对象上,这时脚本与场景中的摄像机产生了关联。

第 8 步:单击 Play 按钮测试,发现通过键盘方向键可以在场景中移动摄像机,但是速度稍快,并且速度不能改变。

第 9 步:更新代码,内容如下。

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

public class Move : MonoBehaviour
{
    public float speed=5.0f;
    void Update()
    {
        float x=Input.GetAxis("Horizontal") * Time.deltaTime * speed;
        float z=Input.GetAxis("Vertical") * Time.deltaTime * speed;
        transform.Translate(x, 0, z);
    }
}
```

位于 Update()函数上面的这个速度变量 speed 是一个 public 变量,它会显示在 Inspector 视图中,可以调整它的值,便于测试。

第 10 步:增加新的功能,实现单击时在摄像机当前位置创建 Cube 游戏对象,创建 C# 脚本,将其命名为 Create,并将脚本链接到 Main Camera 上,如图 3.11 所示。

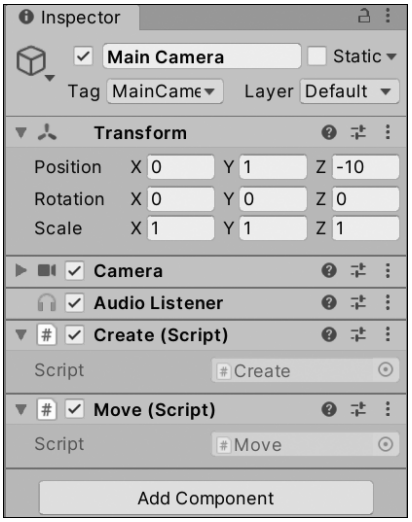

图 3.11 脚本链接

第 11 步：输入代码，如下所示。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class Create : MonoBehaviour
{
    public Transform newObject;
    void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            Instantiate(newObject, transform.position, transform.rotation);
        }
    }
}
```

第 12 步：创建预制体。

Unity 2019 新版本采用直接的预制体创建方式，直接将游戏对象从 Hierarchy 视图拖到 Project 视图，即可完成预制体的创建。具体实现时，将 Hierarchy 视图中制作完成的立方体 Cube 拖到 Project 视图中，重命名为 MyCube 即可完成预制体内容的制作，如图 3.12 所示。

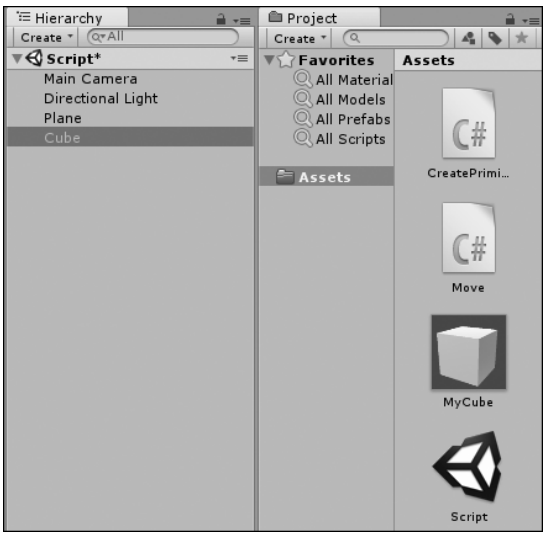

图 3.12　关联预制体

第 13 步：调试。调试是发现和修正代码中人为错误的技巧，Unity 提供了 Debug 类，Log()函数允许用户发送信息到 Unity 的控制台。当用户单击时，发送一个消息到 Unity 控制台，修改脚本如下所示。

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;

public class Create : MonoBehaviour
{
    public Transform newObject;
    void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            Instantiate(newObject, transform.position, transform.rotation);
            Debug.Log("Cube created");
        }
    }
}
```

第 14 步：将 MyCube 拖入 Inspector 视图中对 Create 脚本赋值，如图 3.13 所示。

4. 项目测试

运行游戏并单击，创建一个新的 Cube 实例，控制台会出现 Cube created 字样，如图 3.14 所示。同时场景中创建了新的 Cube，如图 3.15 所示。

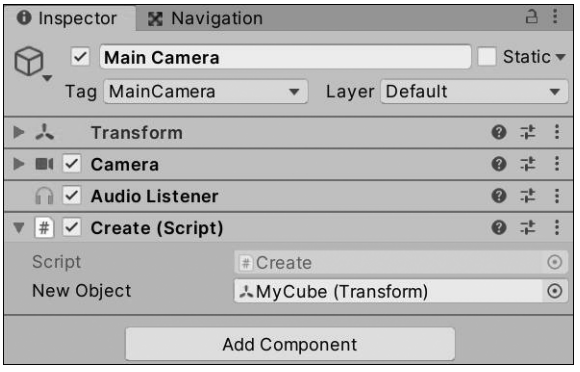

图 3.13　对 Create 脚本赋值

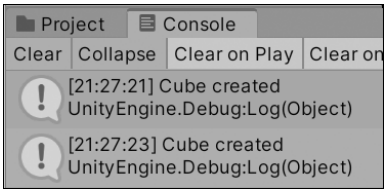

图 3.14　控制台测试效果

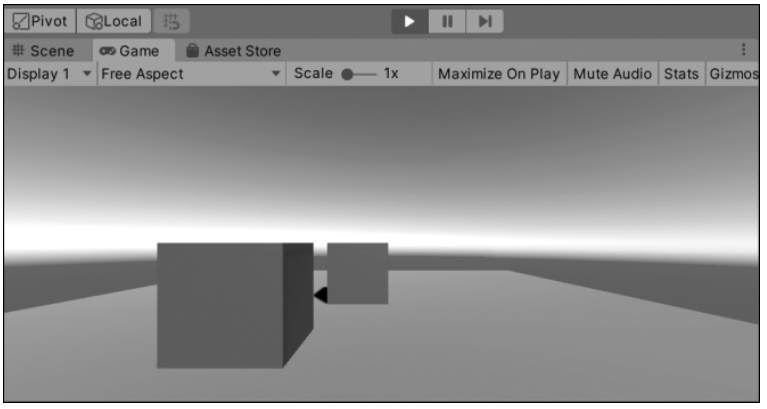

图 3.15　运行测试效果

3.3.2 创建游戏对象

1. 项目构思

游戏场景中对象间的交互都可以通过程序脚本控制并实现。创建游戏对象的方法有3种：第1种是将物体模型资源由Project视图直接拖到Hierarchy视图中；第2种是在GameObject下拉菜单中创建Unity自带的游戏对象；第3种是利用脚本编程动态地创建或删除游戏对象。本项目计划采用第3种方法，即利用脚本编程动态地创建游戏对象。该方法又分为2种：使用CreatePrimitive方法创建Unity系统自带的基本游戏对象和使用Instantiate实例方法对预制体进行实例化操作。

调用Instantiate方法实例化游戏对象与调用CreatePrimative方法创建游戏对象的最终结果是完全一样的，实例化游戏对象会将对象的脚本及所有继承关系实例化到游戏场景中。相较于创建物体的CreatePrimative方法，Instantiate实例化方法的执行效率要高很多。在开发过程中，通常会使用Instantiate方法执行实例化物体。调用该方法时，一般与预制体Prefab结合使用。

2. 项目设计

本项目计划通过C#脚本在Unity引擎内创建一个简单的Cube模型和Sphere模型。单击屏幕左上方的按钮创建Cube和Sphere模型，如图3.16所示。

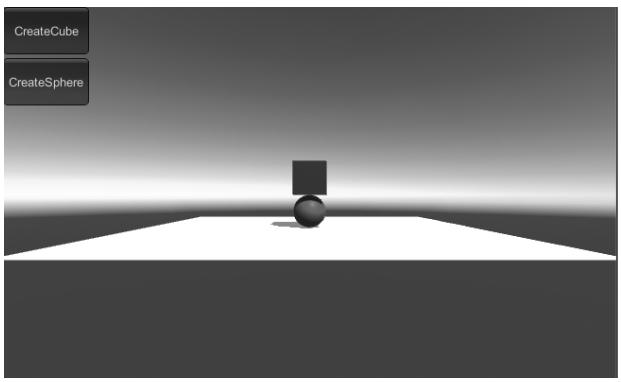

图3.16 创建Cube和Sphere模型

3. 项目实施

第1步：选择菜单栏中的File→New Scene命令，新建场景，创建平面，搭建简单场景，如图3.17所示。

第2步：使用CreatePrimitive方法创建游戏对象，创建C#脚本，将其命名为CreatePrimiteve，输入代码如下。

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

public class CreatePrimiteve : MonoBehaviour
{
```

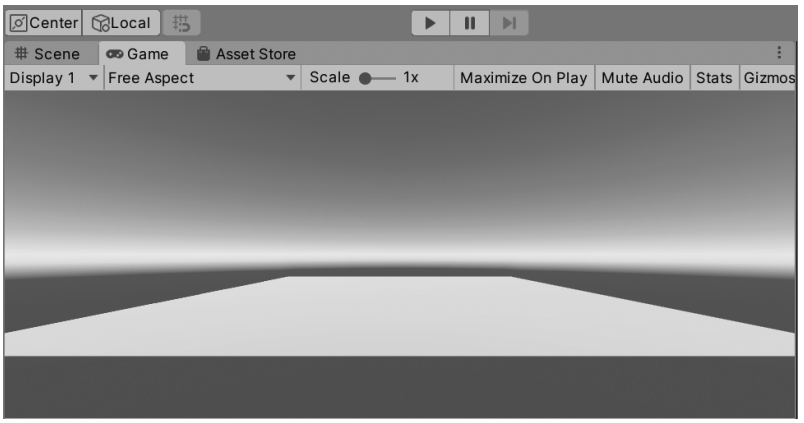

图 3.17　创建平面

```
    void OnGUI()
    {
        if (GUILayout.Button("CreateCube", GUILayout.Height(50)))
        {
            GameObject m_cube=GameObject.CreatePrimitive(PrimitiveType.Cube);
            m_cube.AddComponent<Rigidbody>();
            m_cube.GetComponent<Renderer>().material.color=Color.blue;
            m_cube.transform.position=new Vector3(0, 10, 0);
        }
        if (GUILayout.Button("CreateSphere", GUILayout.Height(50)))
        {
            GameObject m_cube=GameObject.CreatePrimitive(PrimitiveType.Sphere);
            m_cube.AddComponent<Rigidbody>();
            m_cube.GetComponent<Renderer>().material.color=Color.red;
            m_cube.transform.position=new Vector3(0, 10, 0);
        }
    }
}
```

第 3 步：将 CreatePrimitive 脚本链接到 Main Camera 对象上，如图 3.18 所示。

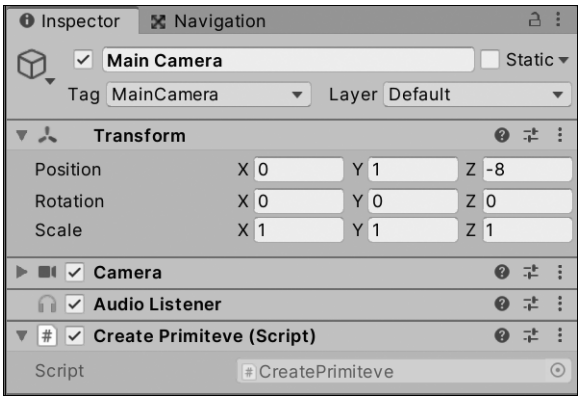

图 3.18　脚本链接

4. 项目测试

单击 Play 按钮进行测试。可以看到，在 Game 视图中单击 CreateCube 按钮或 CreateSphere 按钮后，将分别调用 CreatePrimitive()方法，从而创建 Cube 和 Sphere 游戏对象，运行效果如图 3.19 和图 3.20 所示。

图 3.19　运行测试前

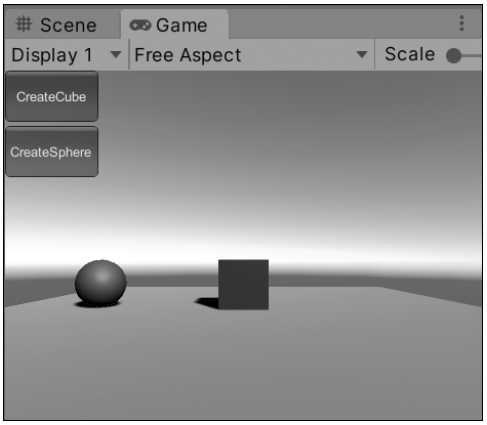

图 3.20　运行测试后

3.3.3　变换的立方体

1. 项目构思

移动、旋转、缩放功能在脚本编写中经常遇到，可以使用 transform.Translate()、transform.Rotate()等方法实现。本项目通过一个立方体讲解脚本编译中的移动、旋转、缩放函数的编写以及与 OnGUI()函数交互功能的实现。

2. 项目设计

本项目计划通过 C＃脚本在 Unity 引擎内创建一个简单的 Cube 模型，采用 OnGUI()方法写 3 个交互按钮，实现与 Cube 模型进行移动、旋转、缩放的交互功能，如图 3.21～图 3.24 所示。

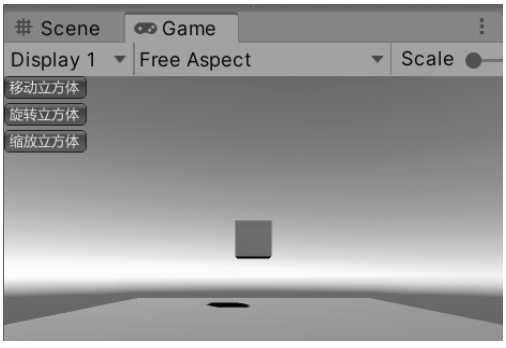

图 3.21　初始场景效果

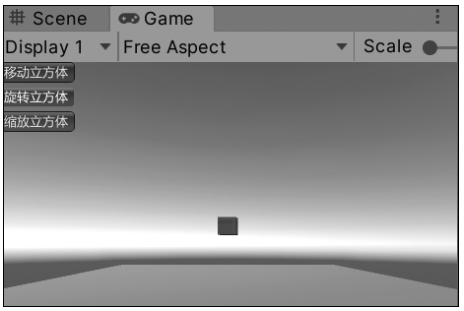

图 3.22　移动立方体效果

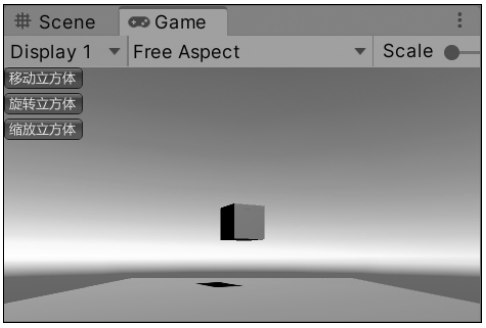

图 3.23　旋转立方体效果

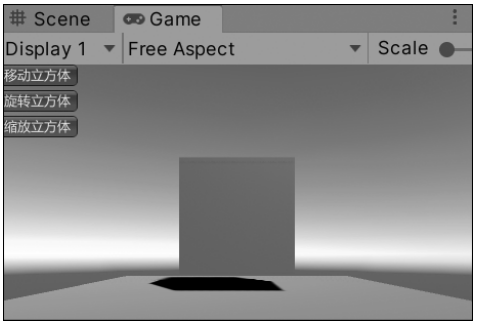

图 3.24　缩放立方体效果

3. 项目实施

第 1 步：选择菜单栏中的 File→New Scene 命令，新建场景。

第 2 步：选择菜单栏中的 GameObject→3D Object→Plane 命令，创建平面，搭建简单场景，如图 3.25 所示。

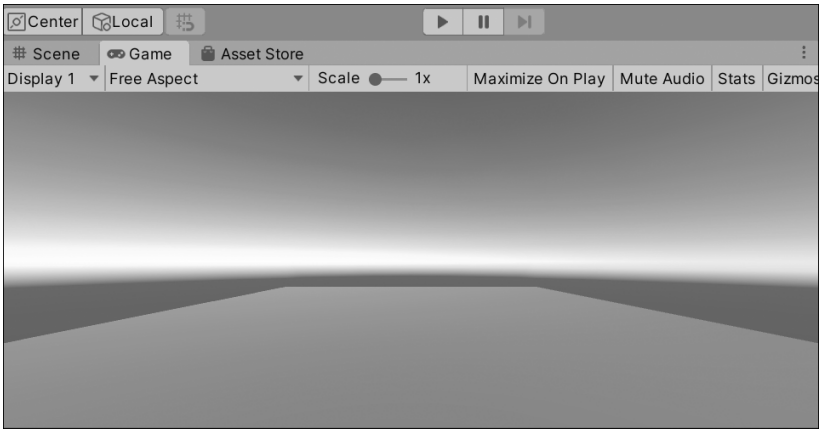

图 3.25　创建平面

第 3 步：选择菜单栏中的 GameObject→3D Object→Cube 命令，创建一个盒子，如图 3.26 所示。

第 4 步：在 Project 视图中新建一个 C# 脚本，将其命名为 MyScript，打开此脚本并添

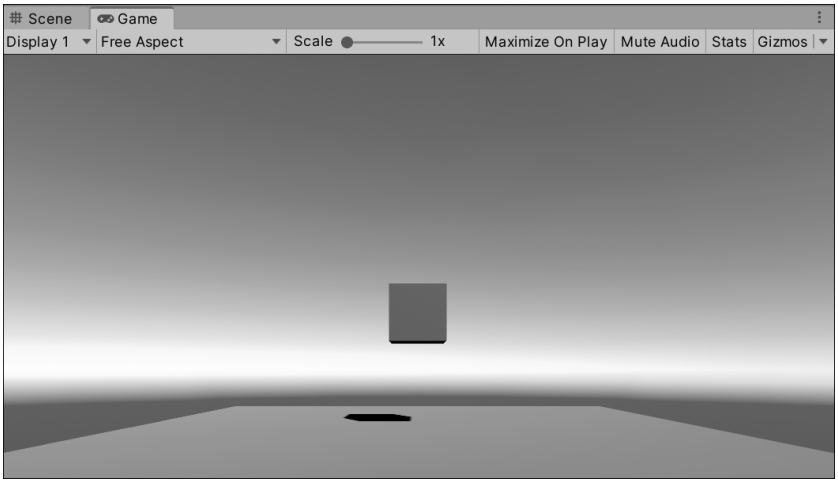

图 3.26 游戏物体场景摆放图

加代码,如下所示。

```
using UnityEngine;
  using System.Collections;
  public class MyScript : MonoBehaviour
{//声明 4 个变量
  public GameObject myCube;
    public int transSpeed=100;
    public float rotaSpeed=10.5f;
    public float scale=3;
    void OnGUI()
    { if(GUILayout.Button ("移动立方体"))
    {
    myCube.transform.Translate (Vector3.forward * transSpeed * Time.deltaTime,
Space.World);
    }
if(GUILayout.Button ("旋转立方体"))
     {  myCube.transform.Rotate (Vector3.up * rotaSpeed,Space.World); }
    if(GUILayout.Button ("缩放立方体"))
    { myCube.transform.localScale=new Vector3(scale,scale,scale);}
      }
  }
```

脚本的第 5 行到第 8 行一共声明了 4 个变量,且都使用 public 修饰,所以它们可以作为属性出现在 Inspector 视图中。脚本的第 9 行到第 17 行是 OnGUI()函数,用于在界面中显示按钮,通过单击按钮实现与立方体的交互功能。

第 5 步:将脚本 MyScript 链接到 Main Camera 上,并将 Cube 拖入 Inspector 视图中,如图 3.27 所示。

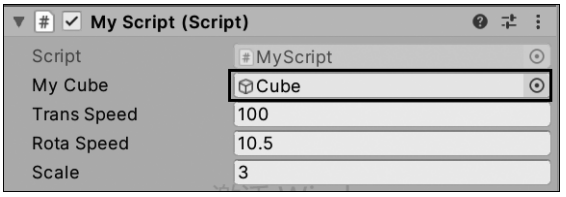

图 3.27　脚本链接

4. 项目测试

单击 Play 按钮进行测试，可以看到 Game 视图的左上角会出现 3 个按钮：移动立方体、旋转立方体和缩放立方体。单击相应的按钮，即可完成对立方体对象的指定操作，如图 3.28 所示。

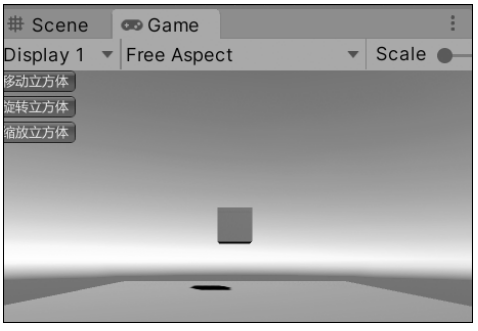

图 3.28　运行测试效果图

3.4　脚本开发综合项目

1. 项目构思

虚拟漫游可以提升游戏玩家的沉浸感。Unity 引擎中提供了第一人称及第三人称虚拟漫游的组件。本项目通过脚本实现第一人称虚拟漫游功能，使读者深入掌握 Unity 脚本编写的方法。

2. 项目设计

本项目计划在场景内摆放一些基本几何体，构建简单 3D 场景，采用 C＃脚本开发第一人称虚拟漫游功能。即通过键盘的 W、S、A、D 键在场景内自由行走，通过鼠标实现观察者视角的旋转功能，如图 3.29 所示。

3. 项目实施

第 1 步：双击 Unity Hub 图标，启动 Unity 引擎，建立一个空项目。

第 2 步：在游戏场景中选择菜单栏中的 GameObject→3D→Plane 命令，创建一个平面，如图 3.30 所示。

第 3 步：选择菜单栏中的 GameObject→Create Empty 命令，创建空物体，并将其标签设为 Player。

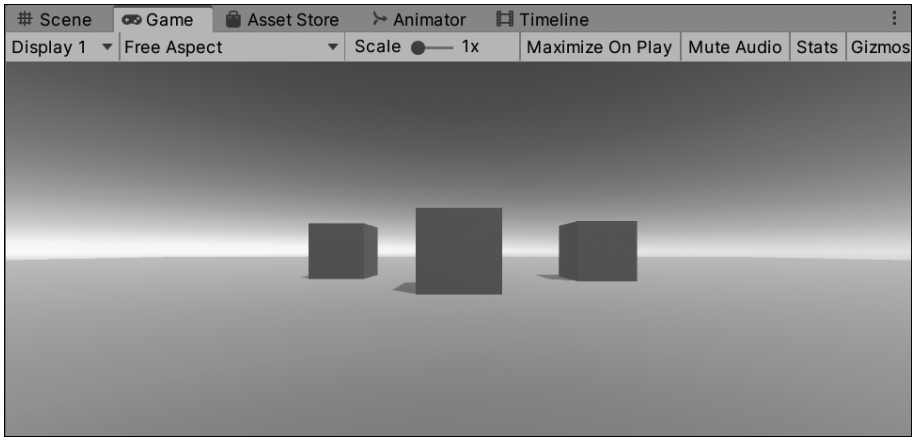

图 3.29　第一人称虚拟漫游测试效果

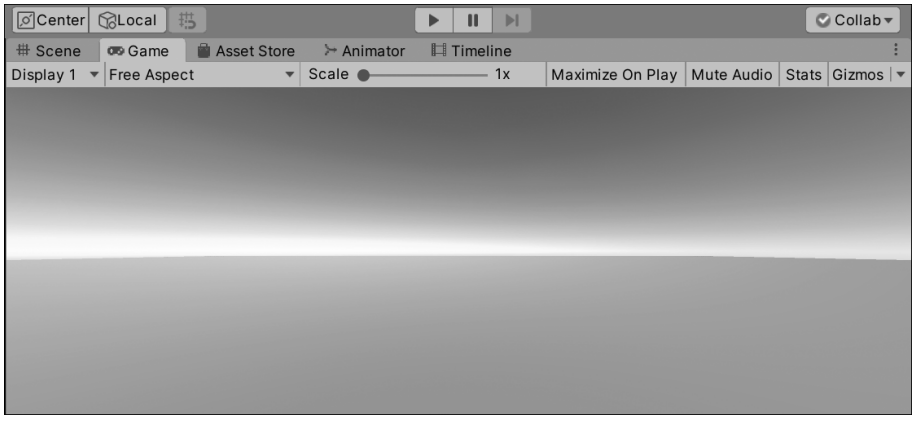

图 3.30　创建平面

一个标签是用来索引一个或一组游戏对象的词。标签是为了编程而对游戏对象的标注，可以使用标签来书写脚本代码，通过搜索找到包含想要的标签的对象。添加标签的方法很简单，即选中 Inspector 视图右上方的 Tag，单击 Add Tag，打开标签管理器，然后在其中输入 Player，如图 3.31 所示。然后再次选择空物体，在 Tag 的下拉菜单中找到 Player 标签，完成标签添加，如图 3.32 所示。

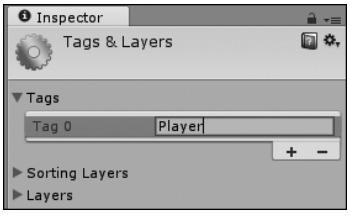

图 3.31　打开标签管理器

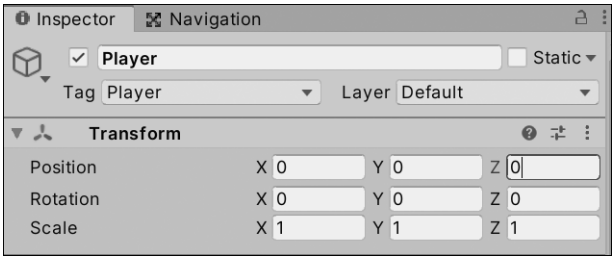

图 3.32　添加标签

第 4 步：在 Hierarchy 视图中选中 Player，然后选择菜单栏中的 Component→Physics→Character Controller 命令，为主角 Player 添加 Character Controller 组件，如图 3.33 所示。Character Controller 主要用于第三人称或第一人称游戏主角控制，并不使用刚体物理效果。

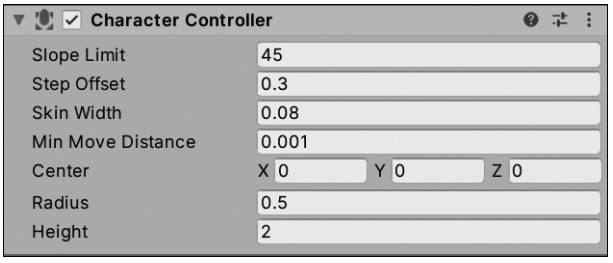

图 3.33　添加 Character Controller 组件

第 5 步：在 Hierarchy 视图中选中 Player，然后选择菜单栏中的 Component→Physics→Rigidbody 命令，为主角 Player 添加 Rigidbody 组件。在 Rigidbody 组件属性中取消 Use Gravity，选中 Is Kinematic，使其不受物理影响，而受脚本控制，如图 3.34 所示。

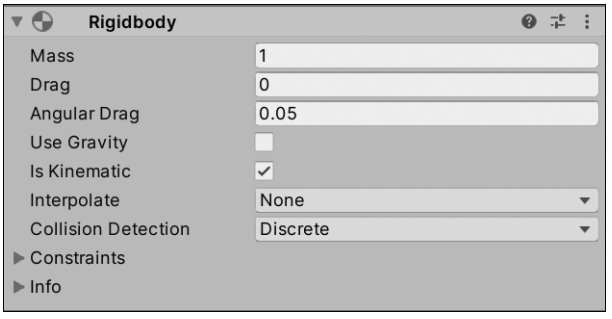

图 3.34　添加 Rigidbody 组件

第 6 步：在 Scene 视图中调整 Character Controller 的位置和大小，使其置于平面之上。

第 7 步：在 Project 视图的空白处右击，在弹出的快捷菜单中选择 Create→C＃命令，创建一个 C＃脚本，将脚本命名为 Player。

第 8 步：输入代码，如下所示。

```
using UnityEngine;
using System.Collections;
public class Player : MonoBehaviour {
    public Transform m_transform;
    //角色控制器组件
    CharacterController m_ch;
    float m_movSpeed=3.0f;          //角色移动速度
float m_gravity=2.0f;               //重力
      void Start ()
    {   m_transform=this.transform;
      m_ch=this.GetComponent<CharacterController>();    //获取角色控制器组件
```

```
    }
    void Update ()
    {Control();}
    void Control()
    {  float xm=0, ym=0, zm=0;                              //定义 3 个值控制移动
        ym -=m_gravity * Time.deltaTime;                    //重力运动
            //上下左右移动
        if (Input.GetKey(KeyCode.W)){
            zm+=m_movSpeed * Time.deltaTime;  }
        else if (Input.GetKey(KeyCode.S)){
            zm -=m_movSpeed * Time.deltaTime; }
        if (Input.GetKey(KeyCode.A)){
            xm -=m_movSpeed * Time.deltaTime; }
        else if (Input.GetKey(KeyCode.D)){
            xm+=m_movSpeed * Time.deltaTime; }
        //使用角色控制器提供的 Move 函数进行移动
        m_ch.Move(m_transform.TransformDirection (new Vector3(xm, ym, zm)));
    }
}
```

上述代码主要是控制主角前、后、左、右移动。在 Start ( ) 函数中，首先获取 CharacterController 组件，然后在 Control 函数中通过键盘操作获得 X 和 Y 方向上的移动距离，最后使用 CharacterController 组件提供的 Move 移动主角。使用 CharacterController 提供的功能移动，在移动的同时会自动计算移动体与场景之间的碰撞。

第 9 步：在 Hierarchy 视图中选中 Player 游戏对象，将写好的 Player 脚本链接到 Player 游戏对象上，如图 3.35 所示。

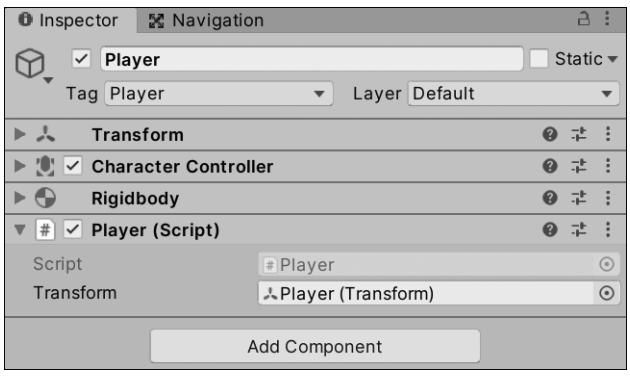

图 3.35　Player 脚本链接

第 10 步：此时运行测试，按 W、S、A、D 键可以控制主角前、后、左、右移动。但是，在 Game 视图中却观察不到主角在场景中移动的效果，这是因为摄像机还没有与主角的游戏对象关联起来。此时需要添加摄像机代码，打开 Player.cs 继续添加代码，如下所示。

```
using UnityEngine;
using System.Collections;
```

```
public class Player : MonoBehaviour {
    public Transform m_transform;
    CharacterController m_ch;
    float m_movSpeed=3.0f;
    float m_gravity=2.0f;
    Transform m_camTransform;                                   //摄像机 Transform
    Vector3 m_camRot;                                           //摄像机旋转角度
    float m_camHeight=1.4f;                                     //摄像机高度
    //修改 Start()函数,初始化摄像机的位置和旋转角度
void Start ()
{   m_transform=this.transform;
    m_ch=this.GetComponent<CharacterController>();              //获取角色控制器组件
    m_camTransform=Camera.main.transform;                       //获取摄像机
    Vector3 pos=m_transform.position;
    pos.y+=m_camHeight;
    m_camTransform.position=pos;
    m_camTransform.rotation=m_transform.rotation;               //设置摄像机的旋转方向与主
                                                                  角一致
    m_camRot=m_camTransform.eulerAngles;
    Screen.lockCursor=true;                                     //锁定鼠标
}
void Update ()
{ Control();}
void Control()
{   //获取鼠标移动距离
    float rh=Input.GetAxis("Mouse X");
    float rv=Input.GetAxis("Mouse Y");
    //旋转摄像机
    m_camRot.x -=rv;
    m_camRot.y+=rh;
    m_camTransform.eulerAngles=m_camRot;
    //使主角的面向方向与摄像机一致
    Vector3 camrot=m_camTransform.eulerAngles;
    camrot.x=0; camrot.z=0;
    m_transform.eulerAngles=camrot;
    float xm=0, ym=0, zm=0;
    ym -=m_gravity*Time.deltaTime;
    if (Input.GetKey(KeyCode.W)){
        zm+=m_movSpeed * Time.deltaTime;}
    else if (Input.GetKey(KeyCode.S)){
        zm -=m_movSpeed * Time.deltaTime;}
    if (Input.GetKey(KeyCode.A)){
        xm -=m_movSpeed * Time.deltaTime;}
    else if (Input.GetKey(KeyCode.D)){
        xm+=m_movSpeed * Time.deltaTime;}
    m_ch.Move(m_transform.TransformDirection (new Vector3(xm, ym, zm)));
    //使摄像机位置与主角一致
```

```
        Vector3 pos=m_transform.position;
        pos.y+=m_camHeight;
        m_camTransform.position=pos;}
}
```

上述代码通过控制鼠标旋转摄像机方向，使主角跟随摄像机的 Y 轴旋转方向，移动主角时，使摄像机跟随主角运动。

4. 项目测试

单击 Play 按钮进行测试，效果如图 3.36 和图 3.37 所示。通过鼠标可以在场景中旋转视角，通过 W、S、A、D 键可以在场景中移动主角向前、向后、向左、向右移动。

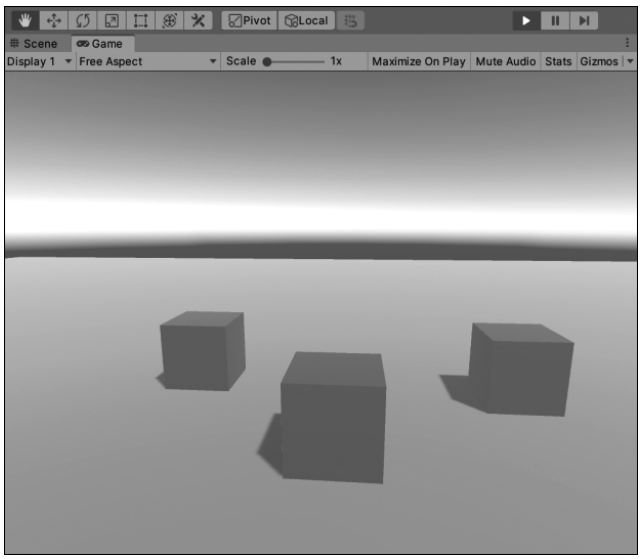

图 3.36　用鼠标控制旋转视角

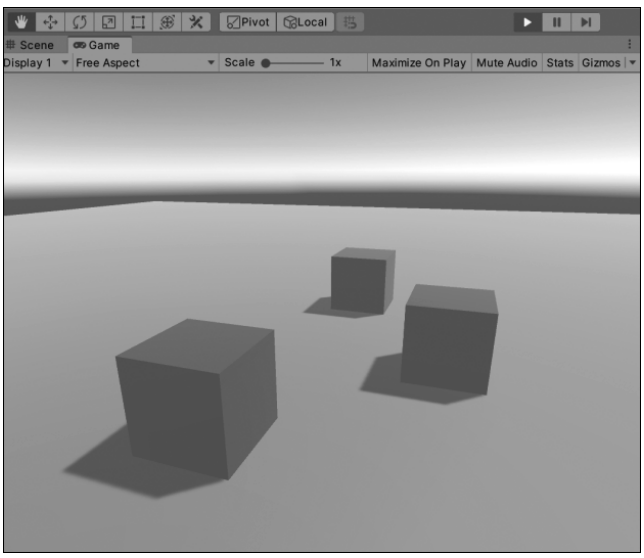

图 3.37　用键盘控制移动方向