# 第 5 章

# 虚拟漫游与交互

**学习目标**

- 理解虚拟漫游的概念。
- 了解虚拟漫游制作流程。
- 掌握虚拟场景的创建和漫游。
- 掌握虚拟场景跳转漫游与生成发布。

## 5.1 虚拟漫游概述

### 5.1.1 虚拟漫游简介

虚拟漫游，是虚拟现实技术的重要分支，在建筑、旅游、游戏、航空航天、医学等多个行业发展迅速，目前主要用于校园漫游、旅游教学、古迹复原、旅游模拟、城市规划等。由于具备可贵的“3I”特性——沉浸感、交互性和构想性，使得沿用固定漫游路径等手段的其他漫游技术和系统无法与之相比。

虚拟场景漫游是虚拟场景建立技术和虚拟漫游技术的结合。前者是基础，后者是系统运行方法。其设计与实现方法可归纳为三种：基于多边形的直接绘制法（简称直接建模法）、场景模型导入法和基于图像的绘制方法。

（1）基于多边形绘制的漫游系统。这种方法适合于场景组织不复杂、多边形数目比较少的较规范、较简单或简化的场景绘制。例如，建筑物远观或大场景的建筑点缀等。

（2）基于模型导入的漫游系统。这种方法通常利用造型软件（如 3ds Max、Maya 等）手工搭建三维模型，建立场景，因而需要耗费大量的时间，工作量很大，一般涉及测量现场、定位和数字化结构平面或者转换现存 CAD 数据；其次很难校验其结果是否精确。其漫游场景是由计算机根据一定的光照模型绘制，色彩层次没有自然景观丰富，带有明显的人工痕迹，即使采用贴图渲染也不能逼真地再现现实世界。随着建模软件的功能日益强大，设计中人们的分工日益明确，人们可以利用日益精细的建筑模型和丰富的模型库资源来加快设计，这一基于几何建模的模型导入技术已成为当今游戏设计等领域的主流技术。

(3) 基于图像的虚拟场景漫游。基于图像的虚拟场景漫游，是利用在某一固定位置所抓取的一个环境的360°全景图像，通过展现全景图像的相应部分来实现相互的调整。人们所熟悉的全景图像技术就是利用系列局部图像拼接起来的，能够进行全视野、360°全方位环视漫游的图像环境。这种技术可以避免复杂的三维建模工作，所以更适合复杂的自然风光、地景的漫游。

虚拟场景漫游的最大特点是硬件要求低、图形处理速度快、高效的网络分布式计算。在大规模场景仿真方面，它通过地块的动态调度、物体细节等级的智能变化、快速消隐等算法使任意规模的场景都可以在普通计算机平台上实时仿真。在数据管理方面，它能够与现有数据库进行有效对接，达到可视化对象和后台数据信息的有效关联，从而实现数据管理和查询。

虚拟场景漫游方案具有以下特色。

(1) 模拟场景的实时漫游。支持鸟瞰、步行、飞行等多角度对整个景区进行全方位的观赏。三维场景具有固定线路和自主漫游功能。通过键盘的简单结合或触摸屏可以方便灵活地实现漫游：前进、后退、左转、右转、左平移、右平移、上升、下降、仰视、俯视等一系列漫游操作。通过键盘上设定的按键就可以对漫游速度进行自由控制。

(2) 身临其境的感受。虚拟现实漫游系统所创造的平台，通过3D建模根据实际场景进行等比例的建模，然后通过高级材质球，打造逼真的场景还原。最后通过渲染引擎，在漫游的同时，实时渲染场景。例如，太阳光和阴影的实时更新、24h天气效果模拟、春夏秋冬效果等。

(3) 支持快捷聚焦功能。可以在固定路线和自由漫游中实时切换相机焦距，并具备焦距切换快捷键。在固定路线的游览教学中，可对游览路线进行暂停、播放和停止的操作，暂停后可以360°环视场景，并能对漫游速度进行方便调节；利用鼠标、键盘、操纵杆等通用交互设备，操作者可以在虚拟场景内若干条特定路线设定导航路径，操作简单直观，初学者即可很方便地控制速度、方向、观测角度、高度等漫游模式。也可对特定景物进行细节聚焦、360°展看，并且可对游览路线进行暂停、播放和停止的操作。

虚拟场景漫游的制作技术主要包括虚拟场景制作技术和场景交互技术。

虚拟场景制作通常包括两种方法：一种方法是利用三维建模软件(通常是3ds Max、Maya等)，根据真实场景的客观数据制作三维模型，然后将多个三维模型搭建成虚拟场景；另一种方法是利用摄像设备扫描周围空间的真实图像，再将图像拼接成全景图实现场景的虚拟再现，此技术即是全景图技术。而场景交互技术是通过引擎软件(通常是Unity或Unreal)设计场景的交互操作，例如，自主漫游、自动漫游和鼠标交互、人机交互等。本章重点介绍三维模型搭建场景的虚拟漫游的制作步骤。

### 5.1.2　虚拟漫游制作流程

虚拟场景漫游的制作流程通常分为4个步骤：场景设计、三维建模、场景搭建和人机交互，如图5.1所示。

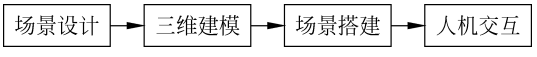


图5.1　虚拟场景漫游制作流程

场景设计是对场景的布局，设计场景中物体的大小、形状和位置等信息。

三维建模是利用3D建模软件(3ds Max、Maya等)创建场景中物体的三维模型。

场景搭建是根据场景的设计方案，利用引擎软件(Unity 或 Unreal)将三维模型拼接起来，形成相对完整的场景，并添加物理碰撞系统，模拟真实的碰撞效果。

人机交互是利用脚本控制语句(C#、C++等)，实现自由漫游、自主漫游、鼠标交互操作甚至沉浸式交互漫游等。

本章案例以 3ds Max 创建的虚拟场景为基础，讲授 Unity 3D 虚拟漫游的制作，重点讲解通过导入三维模型进行场景的搭建和人机交互的初步实现。

## 5.2 虚拟漫游与交互设计

打开 Unity 软件，新建 3D 工程文件，设置保存工程文件的路径，单击“创建”按钮。注意项目名称和工程文件保存路径不要使用中文，推荐使用 Unity Hub 新建项目。

### 5.2.1 虚拟场景的创建

#### 1. 导入标准资源包

打开新建工程，在 Project 窗口中右击，选择 Import Package 命令，选择 Unity 标准资源包 Standard Assets，单击 Import 按钮，导入 Environment 模型、Characters 模型和贴图素材包等资源，如图 5.2 所示。本节使用标准资源包创建一个简单地形效果，读者也可以搜索相关三维地形生成软件，如 World Creator、QuadSpinner Gaea 等，可以根据个人需求，快速模拟各种自然地形效果。

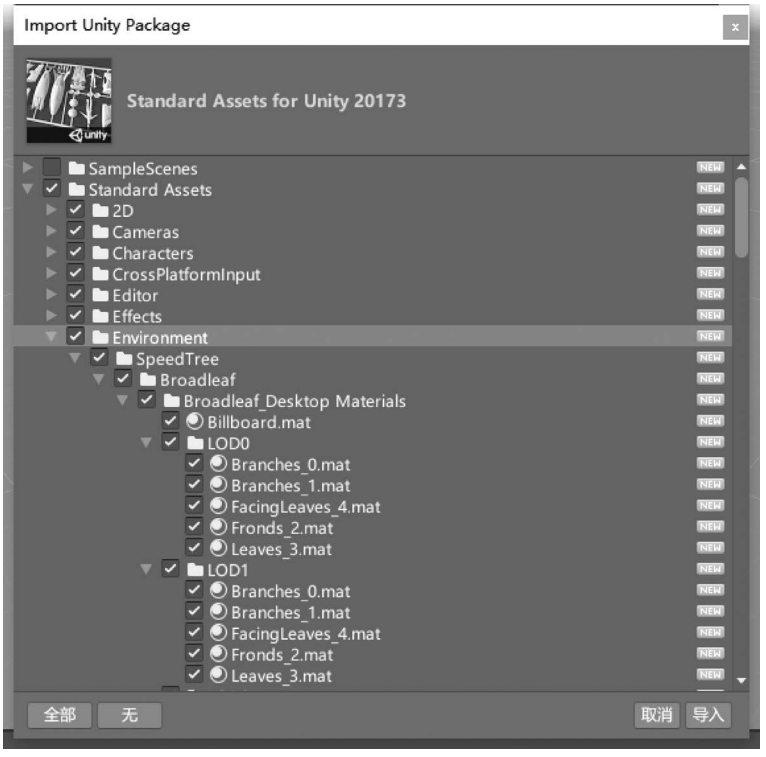


图 5.2 标准资源包

**2. 绘制地形**

在 Hierarchy 窗口空白处右击，选择 3D Object→Terrain 命令创建一个基本地形。选择 Terrain，可以在 Inspector 视图的 Terrain 组件中找到 Mesh Resolution 属性，设置 Terrain 选项，如图 5.3 所示。这里设置 Terrain Width 和 Terrain Length 均为 100。

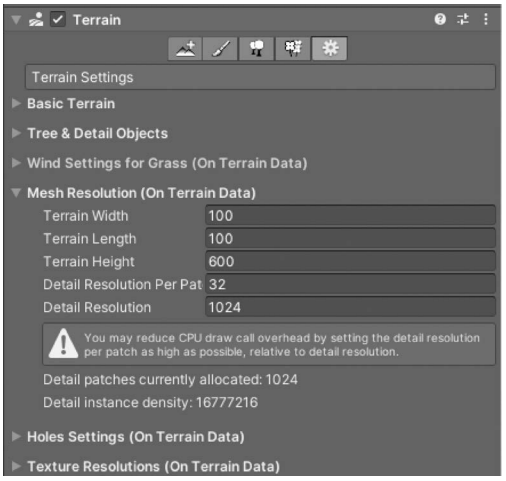


图 5.3　设置地形参数

选择 Terrain，在 Hierarchy 视图中选择主摄像机，可以在 Scene 视图中观察到地形，将其移到主摄像机适当的位置。在右边 Inspector 视图的 Terrain 组件中选择 Rains/Lower Terrain 工具，选择笔刷样式，然后在地形上刷就可以刷出突出山形，按住 Shift 键单击可以让山形重新凹平下去。

选择 Smooth Height 工具，可以平滑山形，使得地形过渡自然，不那么陡峭。可以先用大笔刷刷出大概，再用小笔刷刷细节，最后用笔刷平滑。选择不同的笔刷与贴图以及下面对笔刷大小等设置就可以刷出更复杂的地形来。选择 Paint Height 工具，然后选择“笔刷”可以输出平顶的山形，绘制地形效果如图 5.4 所示。

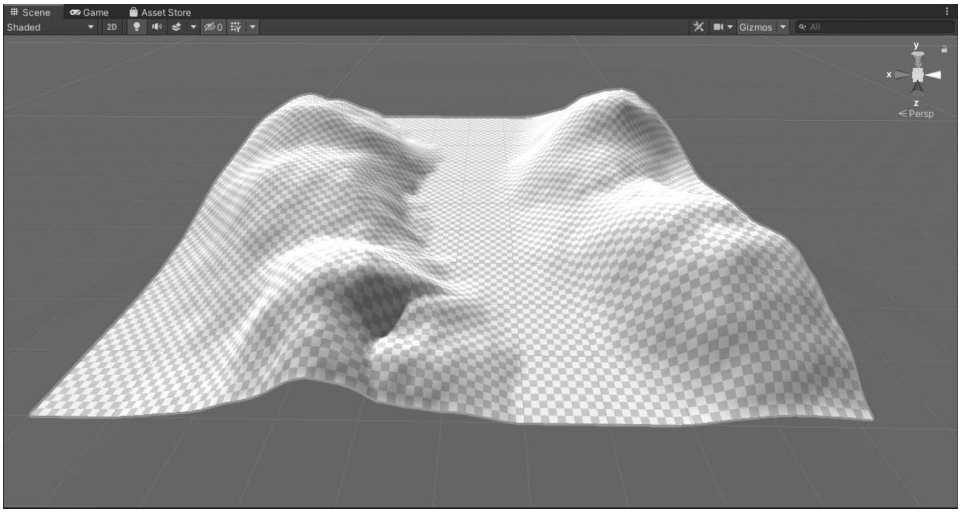


图 5.4　绘制地形效果

**3. 绘制纹理**

选择 Paint Texture 绘制纹理，然后选择 Edit Terrain Layers→Create Layer 命令添加地形图层，如图 5.5 所示，这个操作可以反复执行多次添加多个地形图层，最后在图层中选择需要的贴图，将贴图画在 Terrain 上面。注意在 Tiling Setting 中设置贴图的尺寸，并选择笔刷(Brushes)的大小、类型。

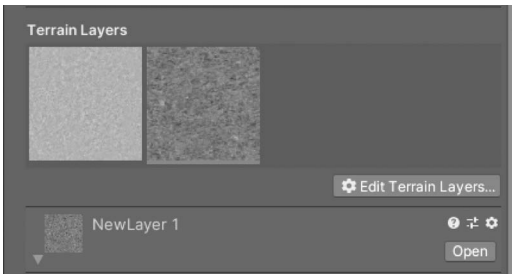


图 5.5 添加纹理图层

**4. 绘制树木**

选择 Terrain 组件中的 Paint Trees(绘制树木)工具，选择 Edit Trees→Add Tree 命令弹出对话框，选择 Tree Prafab 模型，这样操作也可以执行多次加入多个模型。在 Trees 中选择需要的模型，设置好绘制树木笔刷的大小、树的密度、高度和其他参数，然后在地形上单击鼠标即可将树木绘制到 Terrain 上面，绘制树木效果如图 5.6 所示。

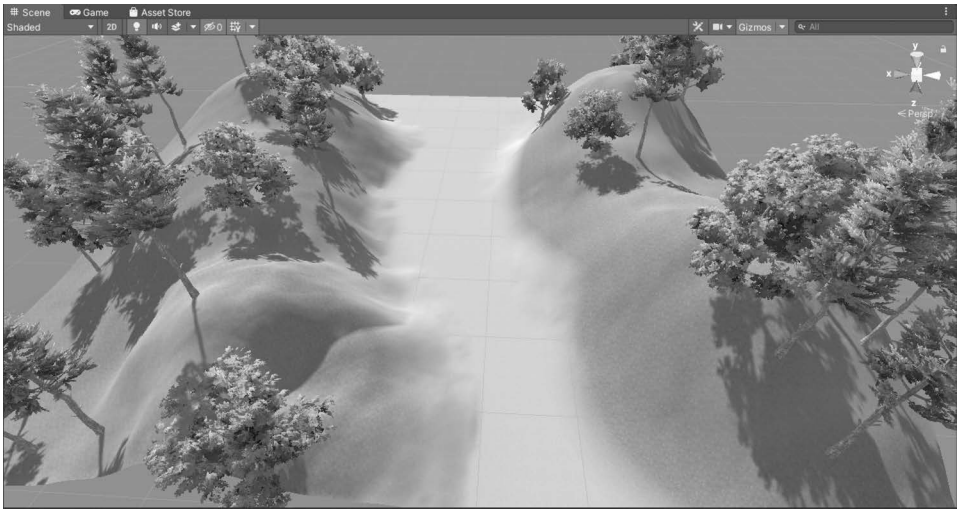


图 5.6 绘制树木效果

树木是固定的、从地形表面生长出的三维对象。当一棵树被选中时，可以在地表上用绘制纹理或高度图的方式来绘制树木，为保证好的渲染效果，Unity 3D 对远距离树木进行了优化，以保持可以接受的帧率。按住 Shift 键并单击可从区域中移除树木，按住 Ctrl 键并单击则只绘制或移除当前选中的树木。

**5. 绘制细节**

选择 Terrain 组件中的 Paint Details(绘制细节)工具，选择 Edit Details→Add Grass Texture 命令弹出对话框，选择 Detail Texture 模型设置草地细节，如图 5.7 所示。这样操作也可以执行多次加入多个模型。草地使用二维图像进行渲染来表现草丛，而其他细节从标准网格中生成，绘制草地贴图效果如图 5.8 所示。注意，该效果在远视角中不可见。

**6. 添加水特效**

找到 Water 文件夹下的 Prefab 文件夹，其中包含两种水特效的预制件，可将其直接拖

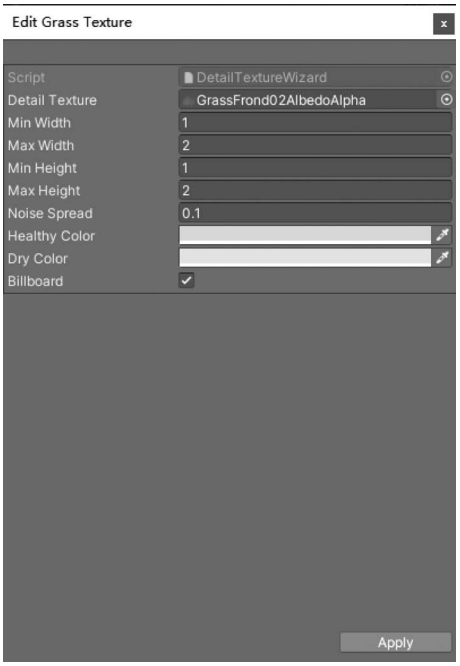


图 5.7　设置草地细节

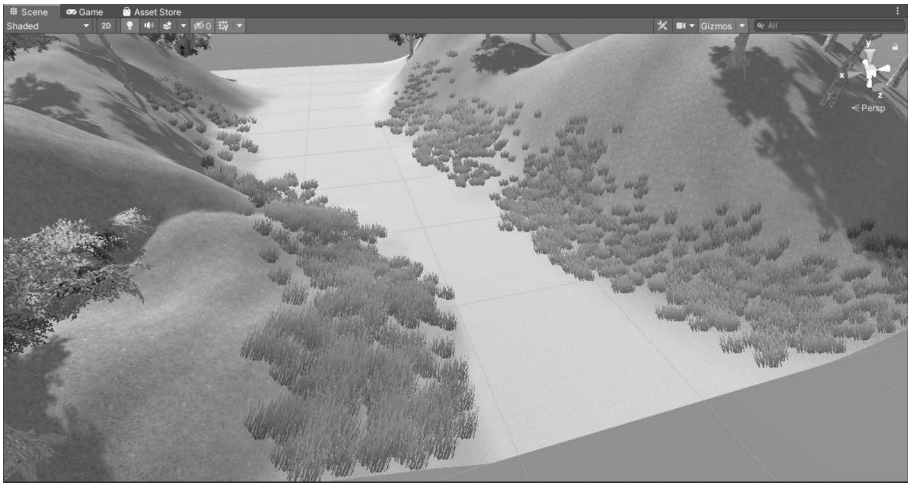


图 5.8　绘制草地贴图效果

曳到场景中，这两种水特效功能较为丰富，能够实现反射和折射效果，并且可以对其波浪大小、反射扭曲等参数进行修改。

单击 Unity 3D 编辑器上的“运行”按钮，如图 5.9 所示，可以观察水效果。水波荡漾，效果比较符合现实。

**7. 加入天空盒**

完整的漫游场景还需要使用 Skybox(天空盒)技术来实现天空的效果，Skybox 资源并不包含在标准资源包中，需要用户通过外部导入。导入的天空盒资源包可以来自第三方建模软件开发的资源，也可以是在 Unity 3D 资源商店中共享的资源。在 Project 窗口中右击，

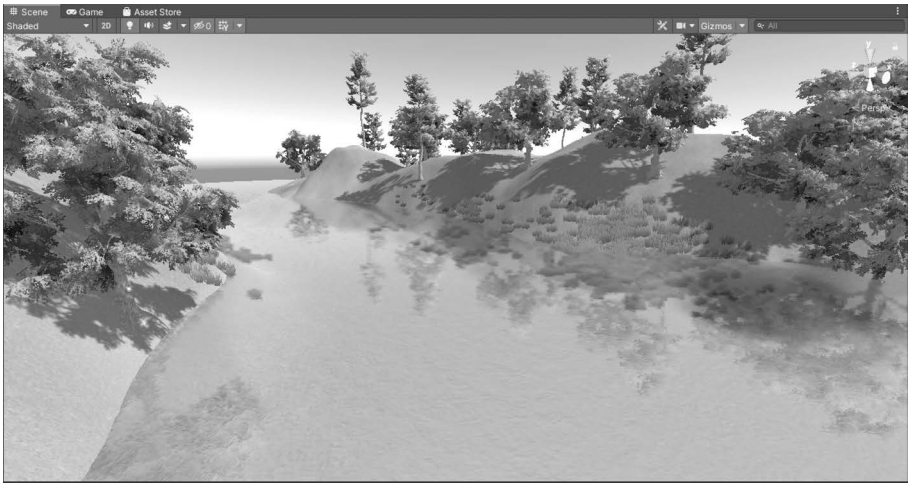


图 5.9　添加水特效

选择 Import Package→Custom Package 命令，选择要导入的 SkyBox. unitypackage 资源包。读者可以将 Project 中 SkyBox 文件夹中的天空盒直接拖曳到 Scene 中，也可以通过菜单添加。在场景中选择 Main Camera 摄像机，在菜单栏中选择 Component→Rendering→Skybox 命令为其添加 Skybox 组件，将 Clear Flags 设置为 Skybox，在 Custom Skybox 中选择 Skybox24 天空模型，如图 5.10 所示。

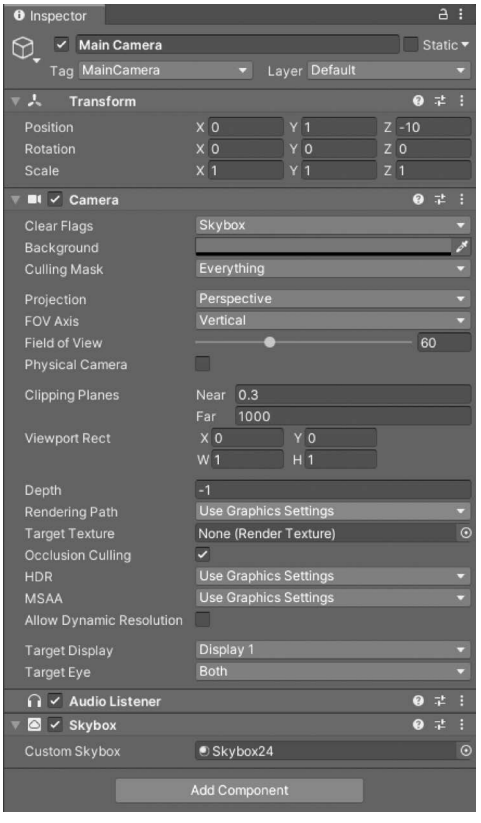


图 5.10　创建 Skybox 材质

在菜单栏中选择 Window→Lighting 命令，为场景添加 Skybox 组件，在 Skybox Material 中选择 Skybox24 天空模型，如图 5.11 所示。

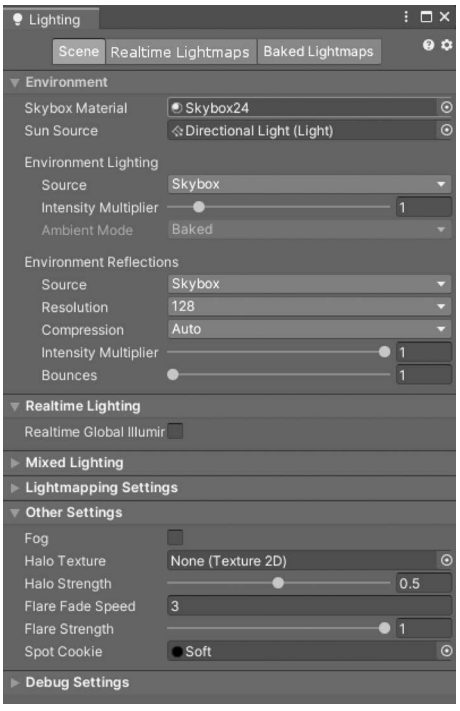


图 5.11　设置 Lighting 窗口

单击 Unity 3D 编辑器上的“运行”按钮，如图 5.12 所示，可以看到天空效果。


图 5.12　加入天空效果

**8. 导入三维模型**

在 Project 窗口中，右击，在弹出菜单中选择 Import Package 命令，分别导入利用三维建模软件创建的资源包 House、Boat 和 Simple Wood Bridge。在打开的对话框中，单击 Import 按钮，将资源导入到工程窗口中。然后，选择导入的资源的预制体 Prefab，按照场景

要求拖入 Scene 场景中，并通过位移、缩放、旋转等操作，对场景内的资源进行布局。最终场景效果如图 5.13 所示。


图 5.13　最终场景效果

**9. 添加物理碰撞**

碰撞系统是模拟物体遇到障碍物时的物理响应。众所周知，物体在没有添加碰撞时，在场景中漫游会无视所有的物体，直穿而过。例如，场景中的建筑物会发生穿过墙体的现象，不符合实际情况。所以，为了逼真地模拟现实，需要对场景中的物体添加碰撞，它是漫游系统真实性实现的方式之一。

由于地面是由自带碰撞系统的 Terrain 地形制作而成，所以不需要添加碰撞系统。而对于其他的，尤其是资源包中导入的模型，则需要添加碰撞。

选中场景中的小房子，在 Inspector 窗口中，单击 Add Component 按钮添加 Box Collider 组件，然后单击所选物体的 Inspector 中 Box Collider 组件中的 Edit Collider 按钮，如图 5.14 所示，此时包围物体的碰撞体为可编辑状态，每个碰撞面的中心都有一个可编辑点，通过拖曳可编辑点实现对碰撞范围的编辑，最终使得碰撞体紧密包围建筑物即可。

图 5.14　添加碰撞体

### 5.2.2　虚拟漫游的实现

虚拟场景中的自主漫游，比较简单且常用的方式是用 Unity Standard Assets 中导入的 Characters 包，如图 5.15 所示，里面包含第一人称控制器 FPSController，将第一人称控制器的 FPSController.prefab 预制体拖入场景中就可以用 W、S、A、D 按键实现前、后、左、右移动，控制器自带脚步音效，如图 5.16 所示，在第一视角的虚拟场景漫游中经常使用。需要注意的是，当把第一人称控制器的预制体拖入场景后，因其绑定有 Camera 组件，需要关闭或删除 Hierarchy 窗口中系统自带的 Main Camera(主摄像头)才可以避免冲突，实现自主漫游。为了便于操作，在测试运行阶段也可以将鼠标在屏幕上显示出来，通过设置 FPSController 的 Inspector 窗口中 First Person Controller(Script)组件中的 Mouse Look 属性，取消 Lock Cursor 参数即可。

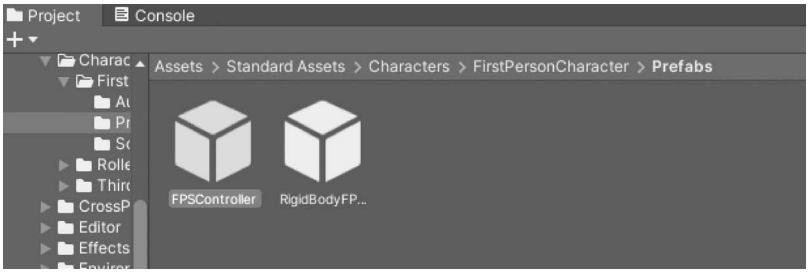


图 5.15　第一人称控制器


图 5.16　添加第一人称预制体

需要提醒的是，在加入角色控制器后，运行有时会出现视线一直在往下掉的情况，用户会看到前面加入的树和地面在往上升，这是因为重力的作用，必须将 First Person Controller 的 Gravity Multiplier 选项设置为 0，默认是 2，这样就不会发生往下掉的情况了。如图 5.17 所示为第一视角自主漫游效果。

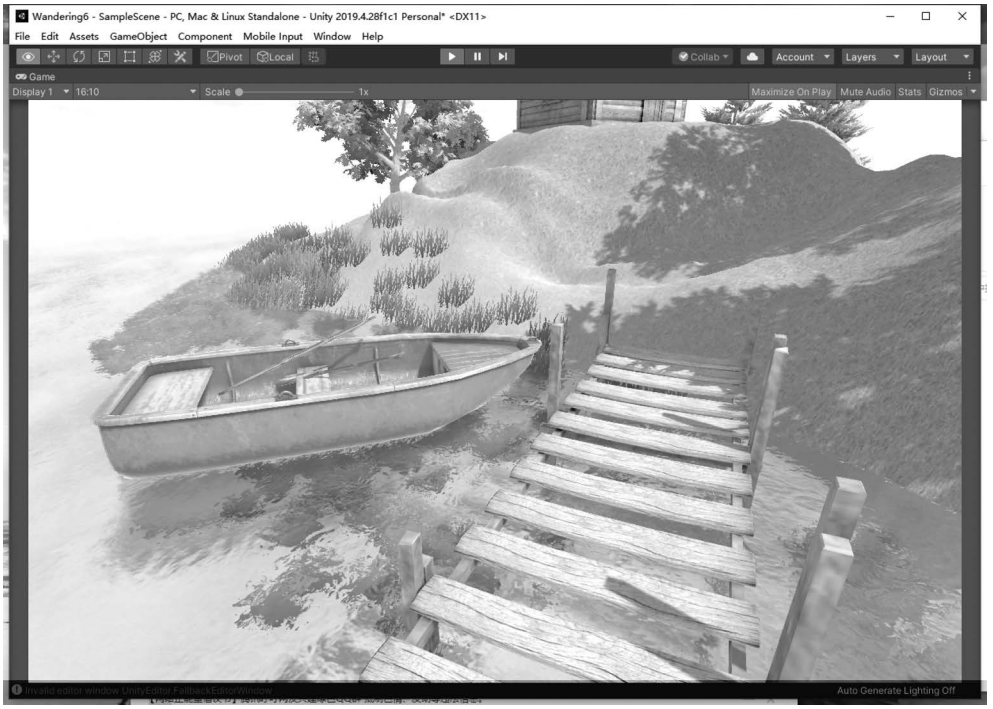


图 5.17 第一视角自主漫游效果

## 5.2.3 虚拟场景中的交互

完成虚拟场景第一视角自主漫游的功能后，接下来将通过为用户制定多种漫游方式（自动漫游、自主漫游），并通过按钮调用，来实现可视化集成虚拟场景漫游系统的所有功能。

**1. 添加跳转按钮**

（1）打开场景 Wandering，在 Hierarchy 窗口中右击，选择 UI→Button 命令新建一个按钮。选中 Button，在 Inspector 窗口中，将其名字修改为 Automatic，如图 5.18 所示。

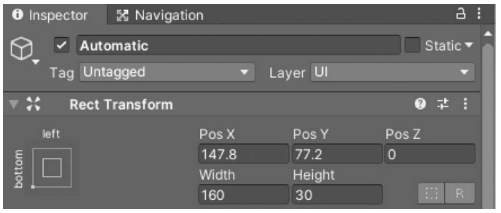


图 5.18 修改 Button 名字

（2）选中 Automatic 按钮，在 Inspector 中，通过 Rect Transform 调整其在屏幕中的位置。选择九宫格图标，按住 Shift 键和鼠标左键，设置锚点在屏幕的正中间。按住 Alt 键和鼠标左键，设置 Automatic 按钮控件在屏幕的左下方。相关操作如图 5.19 所示。

（3）利用 Photoshop 软件创建透明背景的“自动漫游”按钮图标，将按钮导入 Project 窗口中，在图标的 Inspector 中，修改 Texture Type 类型为 Sprite（2D and UI），如图 5.20 所示。

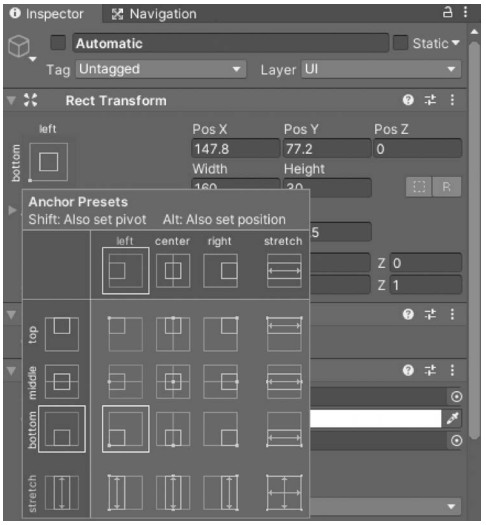


图 5.19　设置按钮控件位置

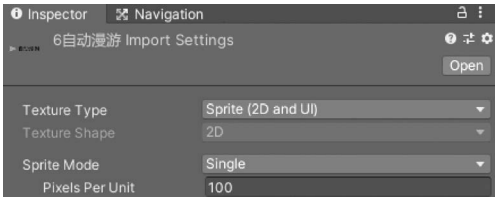


图 5.20　设置图标 Texture Type

(4) 在 Project 窗口中,选择 Automatic,删除下属子控件 Text,调整 Inspector 中 Width 和 Height 的值到合适大小,并修改 Image 组件的 Source Image 值,选择“自动漫游”图标。调整后按钮组件如图 5.21 所示。

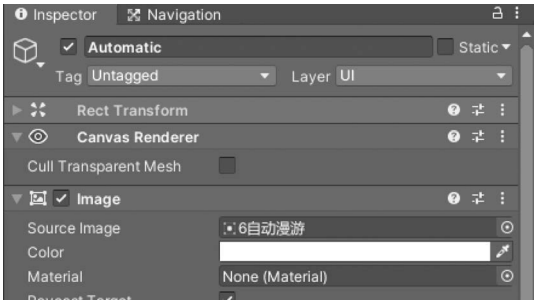


图 5.21　调整后按钮组件

(5) 以同样的方式,添加“退出”按钮,效果如图 5.22 所示。

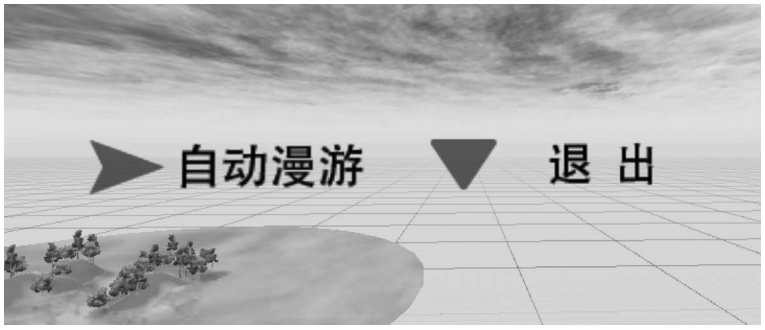


图 5.22　按钮效果

**2. 自动漫游的实现**

虚拟场景中的自主漫游,是以第一人称的角色视角浏览场景,通过 W、A、S、D 键或上、

下、左、右方向按键控制角色的位置，拖曳鼠标进行视角的旋转，空格键完成跳跃。自动漫游与自主漫游不同，是通过特定触发条件，允许角色沿着既定路线进行漫游，可以通过Animator控制摄像机运动来实现。接下来我们通过关键点和漫游动画的制作，分别实现基于按钮的自动漫游和基于导航组件的自动漫游。

1）关键点的制作

（1）执行File菜单下的Save as命令，将场景另存为AutoWandering，然后删除AutoWandering场景的Hierarchy窗口中非场景模型和灯光的其他物体，包括FPSController、Canvas等，仅保留场景模型和灯光。

（2）在AutoWandering场景的Hierarchy窗口中，右击选择新建Camera摄像机并选中，然后选择菜单Window→Animation→Animation命令，弹出Animation动画窗口，如图5.23所示，单击Create按钮，新建动画AAutoMaticAni.anim。

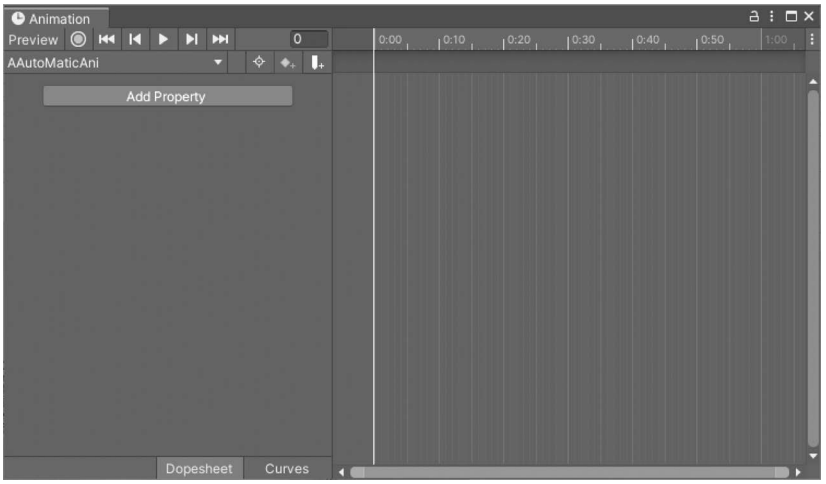


图5.23　Animation动画窗口

（3）单击Add Property按钮，选择Camera的Transform属性，为摄像机的Position和Rotation设置属性值。属性设置如图5.24所示，其中，方框内的数据为当前相机的位置和角度的数值。

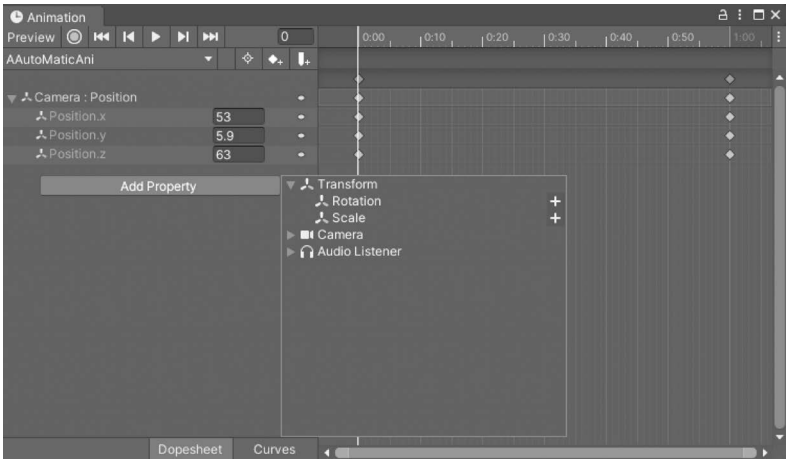


图5.24　属性设置

（4）在 Scene 场景中，按漫游设定的路线移动摄像机 Camera，确定下一个 Camera 位置，调整视角，按 Shift＋Ctrl＋F 组合键将摄像机视角与 Scene 视角同步。然后在 Animation 窗口中选择时间点，单击 Add keyframe 按钮添加关键帧。如图 5.25 所示，重复此过程，直到漫游路线的最后一个位置，即可获得摄像机的运动路径。为方便时间点选择，在 Animation 窗口中滚动鼠标滚轴可以缩放时间轴视图。

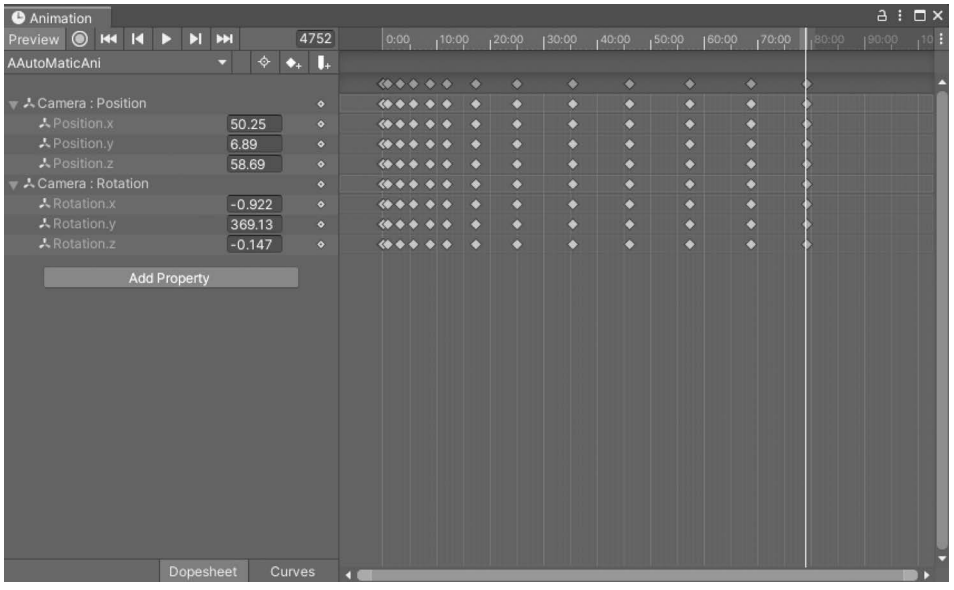


图 5.25　摄像机的运动路径

（5）单击 Animation 对话框中的“录制”按钮，完成动画的录制。

（6）关闭 Animation 对话框后，在 Project 中选择 AutoMaticAni 动画，在其 Inspector 窗口中，取消勾选 Loop Time 选项，取消后 Inspector 窗口如图 5.26 所示。

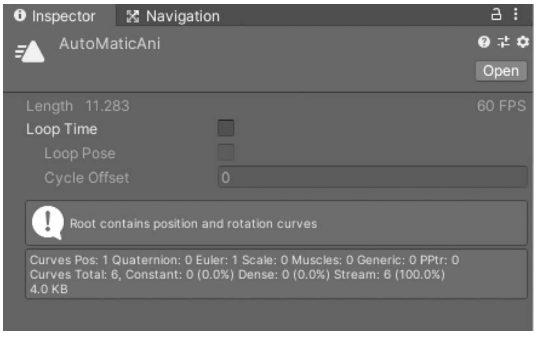


图 5.26　取消勾选 Loop Time

2）漫游动画的制作

（1）选择菜单栏 Window→Animation→Animator 命令，打开 Camera 的 Animator 窗口，右击选择 Create State→Empty 命令，新建一个空的状态 New State，新建过程如图 5.27 所示。

（2）选择 New State，右击选择 Set as Layer Default State 将其设置为默认的初始状态，设置过程如图 5.28 所示。

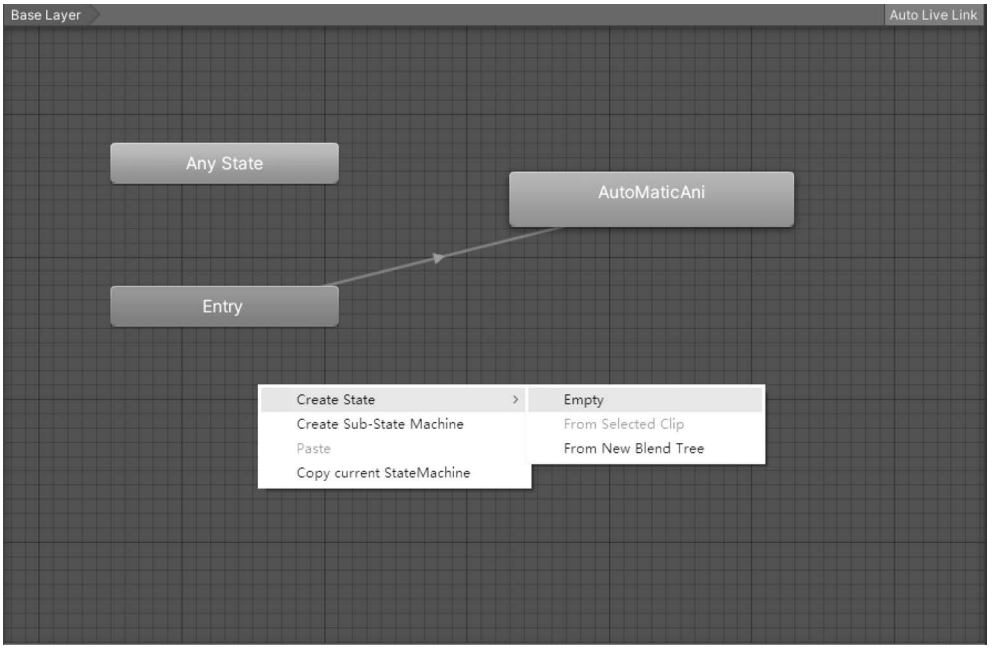


图 5.27　新建空的状态

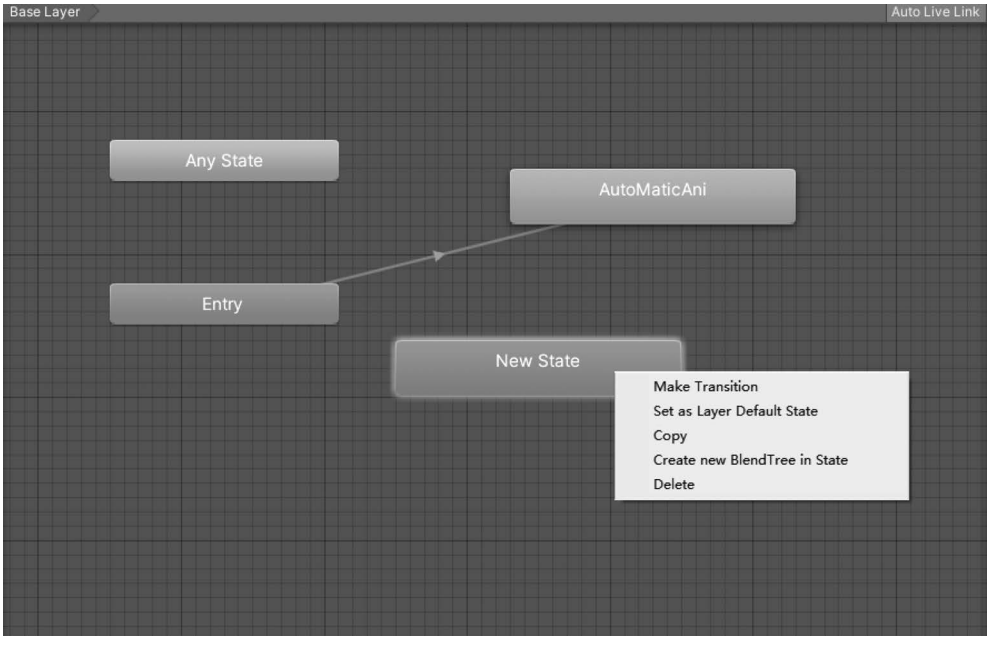


图 5.28　默认初始的状态

(3) 选择 New State,右击选择 Make Transition 命令,创建 New State 与 AutoMaticAni 之间的链接。同样,选择 AutoMaticAni,右击选择 Make Transition 命令,创建 AutoMaticAni 与 New State 之间的链接,如图 5.29 所示。

(4) 设置状态转换的条件。选择 Animator 窗口左侧的 Parameters,单击“+”(添加)按钮,在弹出的下拉菜单中选择 Bool,创建布尔类型的变量,如图 5.30 所示。在新建的框体

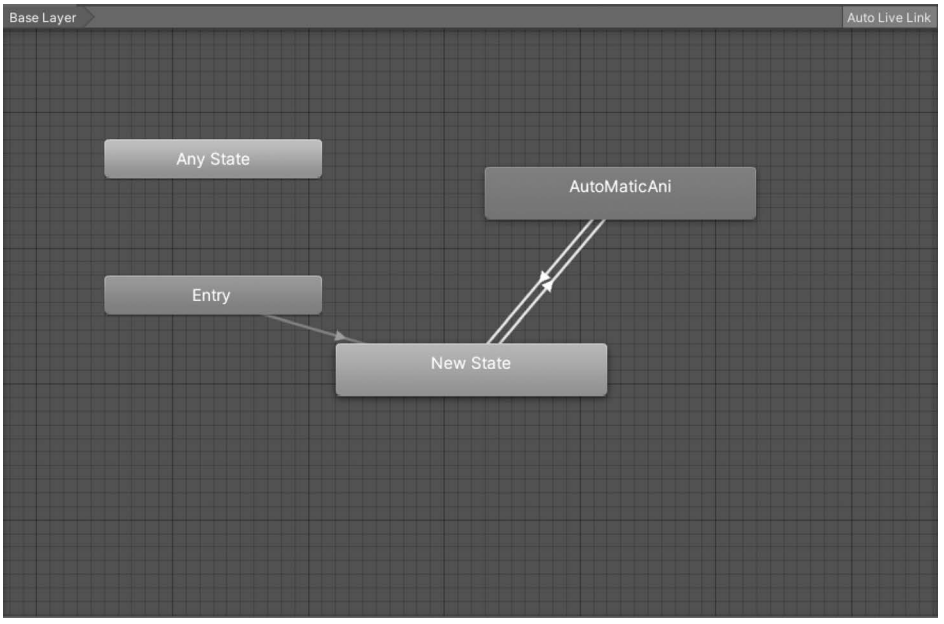


图 5.29　创建状态之间的链接

中输入变量的名字“IsRun”，名字后面的复选框，如果处于选中状态，表示变量的初始值为 True；如果处于未选中状态，表示变量的初始值为 False。本案例中 IsRun 的初始值为 Flase，即不勾选复选框。

（5）使用变量 IsRun 对 Animator 中的连线设置条件。通过鼠标单击选择 New State 到 AutoMaticAni 的连线，在 Inspector 窗口中，取消 Has Exit Time 的选择，单击“＋”按钮添加变量 IsRun，设置值为 True，如图 5.31 所示。

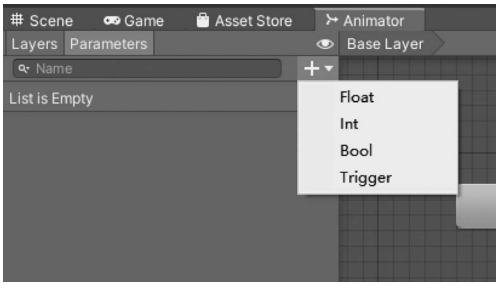


图 5.30　创建布尔类型的变量

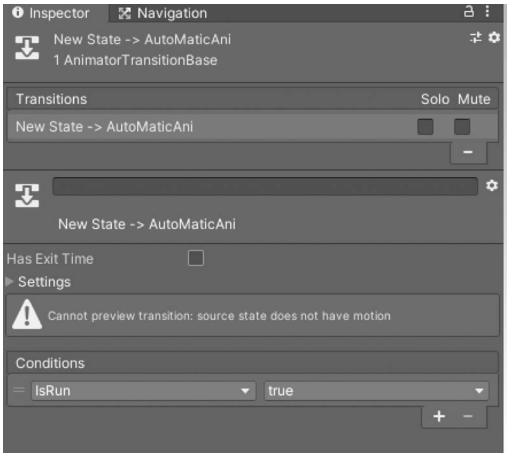


图 5.31　设置 New State 到 AutoMaticAni 的条件

（6）同样的方法，在 Animator 窗口中选择 AutoMaticAni 到 New State 的连线，单击“＋”按钮添加变量 IsRun，设置值为 False，如图 5.32 所示。

（7）在 AutoWandering 场景中添加“返回”按钮，作为中断自动漫游返回主页面的快捷方式，效果如图 5.33 所示。创建的添加和定位方法类似“自动漫游”按钮，这里就不再赘述。

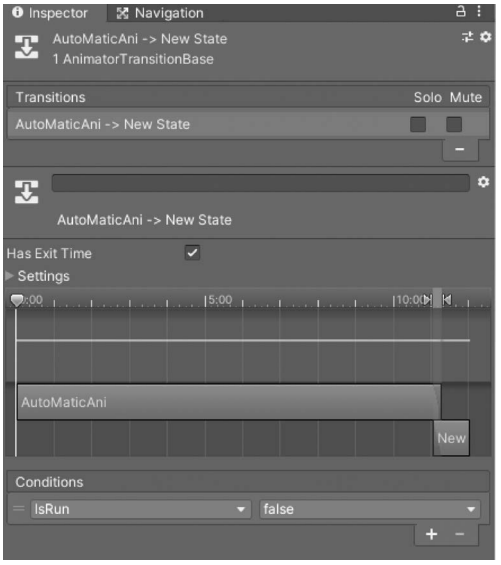


图 5.32 设置 AutoMaticAni 到 New State 的条件


图 5.33 “返回”按钮

3）漫游动画的控制

（1）打开 Wandering 场景，为了实现单击“自动漫游”按钮进行场景跳转的功能，需要为“自动漫游”按钮添加场景跳转控制代码。

（2）在 Project 窗口中，右击新建 C# Script，重命名为 AutoWanderingScript.cs。然后双击打开，在程序编辑器中输入以下代码。这里需要注意的是，因为要操作按钮进行场景跳转，所以需要引入类 UnityEngine.UI 和 SceneManagement。最后，将 AutoWanderingScript 脚本文件拖曳到 Hierarchy 窗口中的 Automatic“自动漫游”按钮上。AutoWanderingScript.cs 代码如下。

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

```
using UnityEngine.SceneManagement;
public class AutoWanderingScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        GetComponent<Button>().onClick.AddListener(OnClick);   //监听按钮
    }
     void OnClick()
    {
        SceneManager.LoadScene("AutoWandering");                //场景跳转
    }
}
```

(3) 为了使自动漫游能够自动运行,还需要用代码控制 AutoWandering 场景中的 Camera 的动画控制器 AutoMaticAni。新建 C# Script,重命名为 RunScript.cs,将其挂载到 Camera 摄像机上。RunScript.cs 代码如下。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class RunScript : MonoBehaviour
{
    private Animator ani;
    // Start is called before the first frame update
    void Start()
    {
        ani = GetComponent<Animator>();   //找到动画控制器
        ani.SetBool("IsRun",true);        //设置动画控制器变量的值
    }
    // Update is called once per frame
    void Update()
    {
    }
}
```

4) 基于导航组件实现自动漫游

"自动漫游"可以理解为通过构造一系列关键点组成的行走路线,程序控制按照规定好的路线来漫游场景。Unity 中自带的导航系统(Navigation)是实现动态物体自动寻路的一种技术,根据开发者所编辑的场景内容,自动地生成用于导航的网格。实际导航时,只需要给导航物体挂载导航组件,导航物体便会自行根据目标点来寻找符合条件的路线,并沿着路线行进到目的地。因此,自动漫游也可以借助 Navigation 系统来实现。具体过程如下。

(1) 前期静态导航设置。将 AutoWandering 场景中的所有几何对象选中,然后在 Inspector 视图中,在 Static 下拉列表中勾选 Navigation Static,将所有对象标记为 Navigation Static(静态导航)。

(2) 烘焙导航网格。选择菜单栏中的 Window→AI→Navigation 命令,如图 5.34 所示,在弹出的 Navigation 窗口中,单击 Bake 标签,设置 Bake 选项卡中的各项参数,例如,Agent Radius 设置为 0.3,Agent Height 设置为 1,Step Height 设置为 0.5 等。然后单击右下角

的 Bake 按钮，烘焙生成导航网格效果如图 5.35 所示。

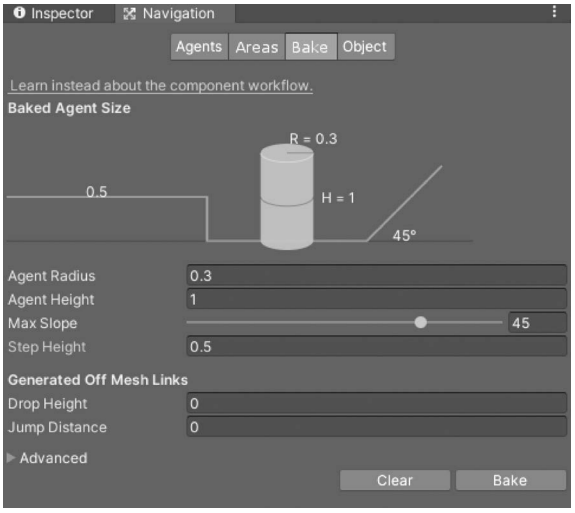


图 5.34　Navigation 窗口


图 5.35　烘焙生成导航网格

（3）创建导航代理。新建一个 Capsule 对象并命名为 Player，设置其 Scale 为(0.5.1.0.5)，然后选择菜单栏中的 Component→Navigation→NavMesh Agent 命令，为 Player 对象添加导航代理组件。

（4）设置漫游路径的关键点。添加一个空物体，命名为 Nav_Patrolling。然后，创建多个空物体作为 Nav_Patrolling 的子物体，放置在场景的关键位置，分别以漫游的顺序序号命名，代表着漫游路径中的关键点，关键点名称如图 5.36 所示。

（5）代码控制实现自动漫游。在 Project 窗口中创建 C＃ Script 代码，命名为 Nav_Patrolling.cs，然后将代码绑定到移动对象 Player 中，并将 Nav_Patrolling 组件中的路径设为代码中的漫游关键点 patrolWayPoints。Nav_Patrolling.cs 代码如下。

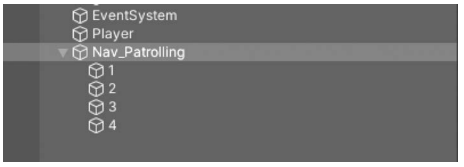


图 5.36　关键点

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class Nav_Patrolling : MonoBehaviour
{
    public float patrolSpeed = 2f;                  //漫游的速度
    public float patrolWaitTime = 1f;               //每个关键点等待的时间为 1s
    public Transform patrolWayPoints;               //漫游点
    private NavMeshAgent agent;                     //智能体
    private float patrolTimer;                      //每个关键点的停留时间
    private int wayPointIndex;                      //漫游关键点的编号
    // Start is called before the first frame update
    void Start()
    {
        agent = GetComponent<NavMeshAgent>();       //找到漫游的智能体
    }
    // Update is called once per frame
    void Update()
    {
        Patrolling();
    }
    void Patrolling()
    {
        agent.speed = patrolSpeed;
        if (!agent.pathPending && agent.remainingDistance <= agent.stoppingDistance)//剩余
//距离小于 stoppingDistance 则计算持续时间,否则持续时间为 0
        {
            patrolTimer += Time.deltaTime;          //持续时间
            if (wayPointIndex == patrolWayPoints.childCount - 1)//如果是最后一个结点,则
//编号为 0,否则++,持续时间改为 0
            {
                wayPointIndex = 0;
            }
            else
            {
                wayPointIndex++;
            }
            patrolTimer = 0;
        }
       else{
       patrolTimer = 0;
            }
        agent.destination = patrolWayPoints.GetChild(wayPointIndex).position; //制定下一
//个目标点的位置
    }
}
```

设置关键点后，Nav_Patrolling 组件如图 5.37 所示。

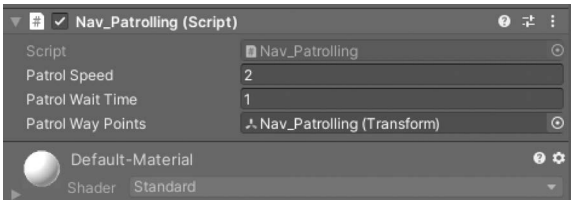


图 5.37　Nav_Patrolling 组件

(6) 相机跟随 Player，提升沉浸体验效果。采用固定摄像机的方法，将摄像机放置在主角头部上方靠后的位置，在主角移动的过程中，摄像机也随着移动。在 Project 窗口中创建 C# Script 代码，命名为 CameraFollow.cs，代码挂载在 Camera 上，并将移动对象 Player 与代码中的 follow 相连。CameraFollow.cs 代码如下。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CameraFollow : MonoBehaviour
{
    public float disAway = 1.7f;
    public float disUp = 1.3f;
    public float smooth = 2f;
    private Vector3 m_TargetPosition;
    public Transform follow;
    // Start is called before the first frame update
    void Start()
    {
    }
        // Update is called once per frame
    void Update()
    {
         m_TargetPosition = follow.position + Vector3.up * disUp - follow.forward * disAway;
          transform.position = Vector3.Lerp(transform.position, m_TargetPosition, Time.deltaTime * smooth);
        transform.LookAt(follow);
    }
}
```

在运行的过程中，如果不想看到第三人称对象 Player，则可以选择 Player 的 Inspector 窗口下的 Mesh Renderer，取消勾选即可。

**3. 返回与退出**

“返回”按钮的功能是从自动漫游的场景跳转到初始场景，“退出”按钮的功能是结束程序，退出漫游场景。

1)“返回”按钮的实现

从自动漫游的场景返回，首先需要将自动漫游场景中的自动漫游关闭，然后再跳转到初始场景。“返回”按钮的代码为 BackScript.cs，编辑完成后将 BackScript.cs 文件直接拖放到 AutoWandering 场景的 Hierarchy 窗口中的 Return 按钮上。BackScript.cs 代码如下。

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

public class BackScript : MonoBehaviour
{
    private Animator ani;
    // Start is called before the first frame update
    void Start()
    {
        GetComponent<Button>().onClick.AddListener(OnClick);
        ani = GameObject.Find("Camera").GetComponent<Animator>();
}

    void OnClick()
    {
        ani.SetBool("IsRun",false);                    //动画关闭,设置变量 IsRun 为 False
        SceneManager.LoadScene("Wandering");
    }
}
```

2）“退出”按钮的实现

“退出”不是退出自动漫游状态，而是退出整个虚拟漫游系统，结束虚拟漫游程序，所以通常出现在最顶层的界面或漫游窗口中。“退出”按钮的代码是 ExitScript.cs，编辑完成后将 ExitScript.cs 直接拖入 Wandering 场景的 Hierarchy 窗口中的 Exit 按钮上。ExitScript.cs 代码如下。

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

public class ExitScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        GetComponent<Button>().onClick.AddListener(OnClick);
    }
    // Update is called once per frame
    void OnClick()
    {
        Application.Quit();
    }
}
```

至此，自主漫游和自动漫游功能均已经实现，选择 File→Build Setting 命令，在弹出的对话框中，分别将 Wandering 和 AutoWandering 场景添加到生成场景中，即可通过按钮实

现两种不同漫游方式的切换。场景漫游最终效果如图 5.38 所示。


图 5.38 自主漫游与自动漫游场景效果

## 5.3 虚拟场景的跳转

### 5.3.1 场景的创建与管理

为丰富场景漫游的内容，可以利用外部资源导入的方式构建新的场景。在 Project 窗口中右击，选择 Import package→Custom package 命令，在弹出的窗口中选择 POLYGON City Pack. unitypackage。这是由 Synty Studios 呈现的一个包含角色、建筑物、道具、车辆和环境资产的低多边形资产包，用于创建基于城市的多边形风格游戏。

导入资源后，在 Project 中会出现 POLYGON city pack 文件夹，找到并双击打开 scene 文件夹下的 DemoScene. unity，可以看到如图 5.39 所示场景效果。

选择 Standard Assets→Characters→FirstPersonCharacter→Prefabs→FPSController. prefab 文件，将其拖曳到 DemoScene 场景中，调整到合适的位置，单击“运行”按钮，即可实现第一人称视角的自主漫游。注意：为了防止边缘跌落，建议创建一个 Cube 并调整合适宽度和高度，摆放在碰撞无法通过的位置，取消勾选网格碰撞器 Mesh Collider 即可形成一堵隐形的碰撞墙。

### 5.3.2 虚拟漫游的界面设计

虚拟漫游的界面设计通常采用 UI 元素实现，本例通过三个按钮分别实现两个场景的


图 5.39　DemoScene 场景效果

漫游进入和退出漫游系统功能，步骤如下。

(1) 添加新 Scene，在 Hierarchy 窗口中右击添加 UI→Image，并将 Image 修改为 background。选择 background 对象，单击 Inspector 窗口，按住键盘上的 Alt 键，同时单击 Inspector 窗口中的 stretch 图标，此时可以将 background 对象铺满整个 UI 界面背景。

(2) 导入 bg.jpg 图片并选中，在 Inspector 中修改纹理类型为 Sprite(2D 和 UI)，单击右下角的 Apply 按钮，然后将其拖曳到 background 对象 Image 组件 SourceImage 中，也可以直接拖曳到 Hierarchy 窗口的 background 上。

(3) 给 background 对象添加子元素 Button，修改 Button 名称为 Nature，删除 Button 子元素 Text，通过精灵图片替换 Inspector 窗口中按钮的 Source Image，得到“漫游自然”按钮。采用类似方法分别添加“漫游城市”和“退出”按钮，效果如图 5.40 所示。


图 5.40　漫游系统界面

(4) 保存场景名为 Start. unity,并将其加入到 Build Setting 窗口 Scenes In Build 选框中,并确保 Start 场景编号为 0。

截至目前,在 Project 中一共添加了 4 个场景,分别为 Start、AutoWandering、Wandering 和 DemoScene,打开 File 菜单下的 Build Setting 窗口,可以利用鼠标直接将其他三个场景从 Project 窗口拖曳添加到 Scenes In Build 选框中,此时四个场景均有一个编号。Build Setting 设置如图 5.41 所示。

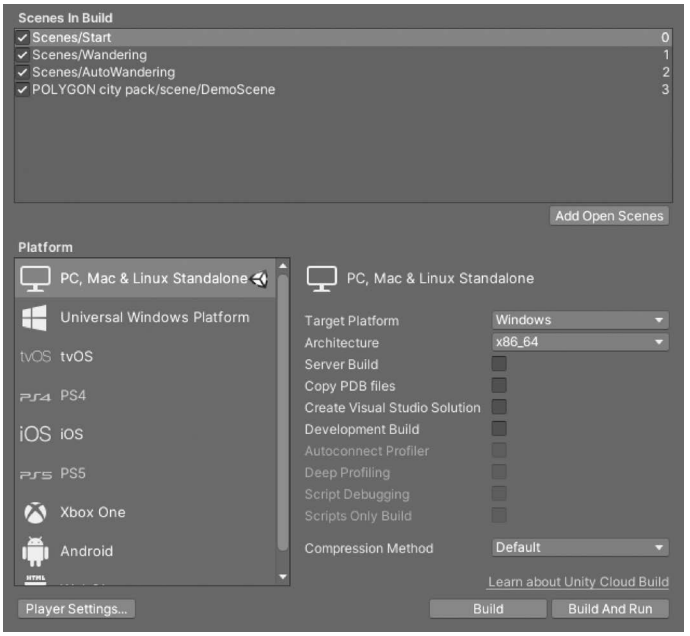


图 5.41 Build Setting 设置

### 5.3.3 场景跳转漫游

本节想实现的效果,是通过一个 Start 场景的 UI 界面上的按钮,分别触发前面完成的 Wandering 和 City 两个场景的漫游。其中,Wandering 除了可以自主漫游外,还保留“自动漫游”按钮和返回功能,在 Wandering 场景窗口中的“退出”按钮可以返回到 Start 起始场景 UI。不管是 Wandering 和 City 任一场景的漫游过程中,只要按了 Esc 键即可终止漫游,返回到 Start 起始场景 UI,UI 界面中的“退出”按钮可以终止漫游程序,退出虚拟场景。实现过程主要从两个方面考虑,一是单击 UI 界面按钮的进入环节,二是漫游过程中的返回和退出环节。

**1. UI 界面按钮实现场景跳转**

在 Project 窗口中右击,新建 C# Script,保存名称为 Control. cs,代码如下。

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

```
public class Control : MonoBehaviour
{
    public GameObject UIbg;

    // Start is called before the first frame update
    void Start()
    {
        UIbg.SetActive(true);
    }

    public void Return() {
        SceneManager.LoadScene(0);
        UIbg.SetActive(true);
    }
    public void Quit() {
        Application.Quit();
    }
    public void LoadScene1(){
        SceneManager.LoadScene(1);
    }
    public void LoadScene3()
    {
        SceneManager.LoadScene(3);
    }
}
```

将完成后的 Control.cs 代码挂载到 Hierarchy 窗口中的 Canvas 画布上，并通过鼠标拖曳设置 Control 组件的 UIbg 为 background，此时 Control 组件设置如图 5.42 所示。

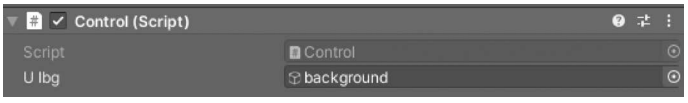


图 5.42　Control 组件设置

单击 Exit 按钮，在 Inspector 窗口中设置 Button 组件的 OnClick()事件，添加 Canvas 对象，并选择 Control.Quit 事件方法。此时 OnClick()属性设置如图 5.43 所示。

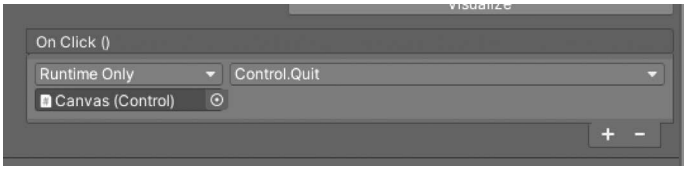


图 5.43　属性设置

同样方法，分别为 Nature 按钮设置 Button 中的 OnClick()事件，添加 Canvas 对象并选择 Control.LoadScene1 事件方法，为 City 按钮设置 Button 中的 OnClick()事件添加 Canvas 对象并选择 Control.LoadScene3 事件方法，最后保存场景 Start。

**2. 漫游过程中按 Esc 键返回**

在 Project 窗口中右击，新建 C# Script，保存名称为 Getout.cs，代码如下。

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

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Getout : MonoBehaviour
{
    // Update is called once per frame
    void Update()
    {
    if(Input.GetKeyDown(KeyCode.Escape)→→Input.GetKeyDown(KeyCode.Home))
        {
            SceneManager.LoadScene(0);
        }
    }
}
```

打开 Wandering 场景，在 Hierarchy 窗口中选中 FPSControl 对象，将代码 Getout.cs 挂载到第一视角 FPSControl 对象身上。注意，如果弹出 Cannot restructure Prefab Instance 警告窗口，单击 Open Prefab 按钮即可。

打开 DemoScene 场景，在 Hierarchy 窗口中选中 FPSControl 对象，同样方法确保将代码 Getout.cs 挂载到第一视角 FPSControl 对象身上。这样就可以实现在漫游过程中，可以随时通过按 Esc 键返回到 Start 场景 UI 界面。

**注意**：测试运行的虚拟漫游很多功能无法呈现，尤其是按钮的触发效果，只有发布出来才可以完整运行。同时，在跳转到其他场景漫游时，跳转后的场景会出现光照烘焙的问题，缺乏光照效果。解决的办法是提前进行光照渲染。

要为多个场景同时烘焙光照贴图，只需要打开所有需要操作的场景，在 Window→Rendering→Lighting 窗口中关闭 Auto Genarate 选项，然后单击 Genarate Lighting 按钮即可。

## 5.4 虚拟漫游系统的设置与发布

选择 File→Build Setting 命令，可以打开发布设置窗口，如图 5.44 所示，其中包含多个发布选项。Unity 开发的项目支持发布到多个平台，Web 和 PC 版是免费发布的，其他平台的发布则需要付费并安装相应的发布插件。发布出来的虚拟场景漫游系统在任何一台不联网的对应类型 PC 上都可以运行。

Scenes In Build 一栏是一个场景列表，默认是空的。如果在场景列表为空的时候发布，则只会将当前正在编辑的场景发布出去并启动场景。这有利于我们快速做一些独立场景的测试工作。正式发布的系统往往包含多个场景，有两种方式添加场景：一是单击 Add Open Scenes 按钮，将打开正在编辑的场景加入栏目；二是直接将保存的场景文件从 Project 窗口拖动到发布窗口栏中。

拖动场景到列表中后，每个场景都被分配了一个数字，其中，0 是漫游开始后第一个被自动加载的场景。如果已经添加多个场景文件并且希望重新排列它们的顺序，只需要在列表中选中并拖动以交换场景。选中一个场景并按 Delete 键就可以删除某个已经添加的场景。

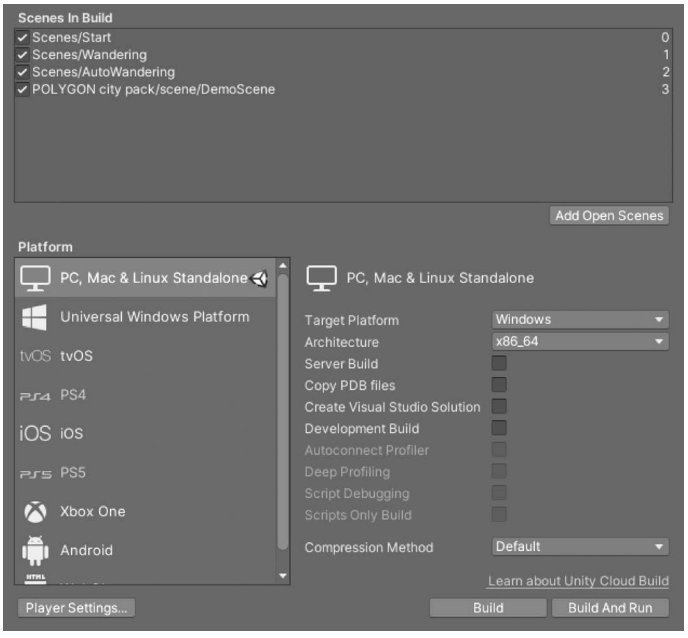


图 5.44　Build Setting 对话框

PC 版本虚拟场景漫游系统在运行时，可以发布选择 Windows 复选框，可以选择窗口运行，可以设置窗口大小；或者选择全屏运行。发布的可执行文件如图 5.45 所示。发布时建议新建一个文件夹，单独存放发布出来的可执行文件及其配套文件，文件之间的相对路径保持不变。

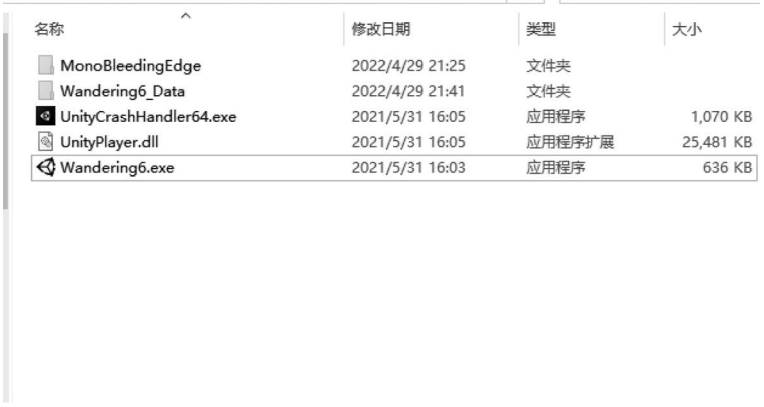


图 5.45　可执行文件

## 小结

本章主要介绍了虚拟漫游的概念，了解虚拟漫游制作流程，并利用一个完整实例逐步讲解了虚拟场景的创建和漫游实现，最后掌握虚拟场景跳转漫游与生成发布，发布出来的虚拟漫游系统可独立运行。

# 习题

## 一、填空题

1. 虚拟场景漫游设计与实现方法可归纳为三种：_____、_____和_____。

2. 虚拟场景漫游的制作技术主要包括_____和_____。

3. 虚拟场景漫游的制作过程通常分为 4 个步骤：_____、_____、_____和_____。

4. 虚拟场景制作通常包括两种方法：一种是_____；另一种是_____。

## 二、简答题

1. 虚拟现实场景漫游方案具有哪些特色？

2. 请简述目前主流三维地形生成软件有哪些，各有什么特点。

3. 请简述虚拟漫游系统制作流程。