

# Unity 4.x 2D

## 游戏开发基础教程

(内部资料)

大学霸



大学霸  
[www.daxueba.net](http://www.daxueba.net)

## 前 言

Unity 是一款综合性的游戏开发工具，也是一款全面整合的专业游戏引擎。它可以运行在 Windows 和 Mac OS X 下，并提供交互的图形化开发环境为首要操作方式。使用 Unity 开发的游戏，可以部署到所有的主流游戏平台，而无需任何修改。这些平台包括 Windows、Linux、Mac OS X、iOS、Android、Xbox 360、PS3、WiiU 和 Web 等。开发者无需过多考虑平台之间的差异，只需把精力集中到制作高质量的游戏即可，真正做到“一次开发，到处部署”。

据权威机构统计，国内 53.1% 的人使用 Unity 进行游戏开发；有 80% 的手机游戏是使用 Unity 开发的；苹果应用商店中，有超过 1500 款游戏使用 Unity 开发。

网上有为数众多的 2D 和 3D 游戏。稍微关注一下，就会发现 2D 游戏才是主流，如植物大战僵尸、愤怒的小鸟、打飞机、2048 等。而且，问问身边的人让他们印象深刻的游戏是什么，你会惊讶的发现，大部分游戏同样是 2D 的。

基于以上不可忽略的事实，本书决定着眼于讲解使用 Unity 开发 2D 游戏的基础知识，且书中包含了两个生动的 2D 游戏示例，相信读者会喜欢它们的。

### 1. 学习所需的系统和软件

- ☐ ☐ 安装 Windows 7 操作系统
- ☐ ☐ 安装 Unity 4.5.1

www.daxueba.net

# 目 录

|       |                                  |           |
|-------|----------------------------------|-----------|
| 第 1 章 | Unity 及其组成的介绍.....               | 错误!未定义书签。 |
| 1.1   | Unity 概述.....                    | 8         |
| 1.2   | 项目、资源和场景.....                    | 11        |
| 1.2.1 | 项目.....                          | 11        |
| 1.2.2 | 资源.....                          | 13        |
| 1.2.3 | 场景.....                          | 15        |
| 1.3   | 场景视图的操作.....                     | 15        |
| 1.3.1 | 使用快捷键操作场景视图.....                 | 16        |
| 1.3.2 | 使用 Gizmo 操作场景视图.....             | 18        |
| 1.4   | 游戏对象和组件.....                     | 19        |
| 1.5   | 脚本与脚本编辑器.....                    | 20        |
| 1.5.1 | 创建脚本.....                        | 20        |
| 1.5.2 | 脚本编辑器.....                       | 21        |
| 1.6   | 脚本的调试.....                       | 23        |
| 1.6.1 | 调试方法一.....                       | 24        |
| 1.6.2 | 调试方法二.....                       | 26        |
| 第 2 章 | 材质和纹理.....                       | 错误!未定义书签。 |
| 2.1   | 材质和纹理的使用.....                    | 错误!未定义书签。 |
| 2.1.1 | 使用材质.....                        | 错误!未定义书签。 |
| 2.1.2 | 不同的材料类型——着色器.....                | 错误!未定义书签。 |
| 2.1.3 | 使用纹理.....                        | 错误!未定义书签。 |
| 2.2   | 应用于 2D 游戏的材质.....                | 错误!未定义书签。 |
| 2.2.1 | 缘由.....                          | 错误!未定义书签。 |
| 2.2.2 | 技巧一：使用白色的环境光.....                | 错误!未定义书签。 |
| 2.2.3 | 技巧二：使用光不敏感着色器.....               | 错误!未定义书签。 |
| 2.3   | 纹理使用规则.....                      | 错误!未定义书签。 |
| 2.3.1 | 规则 1：分辨率是 2 的次方.....             | 错误!未定义书签。 |
| 2.3.2 | 规则 2：保证“质量”.....                 | 错误!未定义书签。 |
| 2.3.3 | 规则 3：增加阿尔法通道（Alpha Channel）..... | 错误!未定义书签。 |
| 2.4   | 导入纹理.....                        | 错误!未定义书签。 |
| 2.4.1 | 导入纹理时默认设置介绍.....                 | 错误!未定义书签。 |
| 2.4.2 | 含有透明信息的纹理.....                   | 错误!未定义书签。 |
| 第 3 章 | 着手开发一个简单的 2D 游戏.....             | 错误!未定义书签。 |
| 3.1   | 开始开发 2D 游戏.....                  | 错误!未定义书签。 |
| 3.1.1 | 导入纹理资源.....                      | 错误!未定义书签。 |
| 3.1.2 | 新建材质资源.....                      | 错误!未定义书签。 |
| 3.1.3 | 修改场景的环境光以及游戏时的屏幕尺寸.....          | 错误!未定义书签。 |
| 3.2   | 为场景添加游戏对象.....                   | 错误!未定义书签。 |

|       |                            |           |
|-------|----------------------------|-----------|
| 3.2.1 | 调整游戏对象的角度.....             | 错误!未定义书签。 |
| 3.2.2 | 改变游戏对象的位置.....             | 错误!未定义书签。 |
| 3.2.3 | 游戏对象的“碰撞”组件.....           | 错误!未定义书签。 |
| 3.3   | 让飞船动起来.....                | 错误!未定义书签。 |
| 3.4   | 让飞船发射子弹.....               | 错误!未定义书签。 |
| 3.4.1 | 在场景中添加子弹.....              | 错误!未定义书签。 |
| 3.4.2 | 游戏时，让子弹在场景中移动.....         | 错误!未定义书签。 |
| 3.4.3 | 生成子弹的预设.....               | 错误!未定义书签。 |
| 3.4.4 | 设置子弹的发射位置.....             | 错误!未定义书签。 |
| 3.4.5 | 在恰当的时机发射子弹.....            | 错误!未定义书签。 |
| 3.5   | 让外星飞船动起来.....              | 错误!未定义书签。 |
| 3.5.1 | 编写脚本.....                  | 错误!未定义书签。 |
| 3.5.2 | 设置外星飞船的触发器.....            | 错误!未定义书签。 |
| 3.5.3 | 为子弹预设添加刚体组件.....           | 错误!未定义书签。 |
| 3.6   | 为游戏添加背景.....               | 错误!未定义书签。 |
| 第 4 章 | 使用编辑器类自定义编辑器.....          | 错误!未定义书签。 |
| 4.1   | 编辑器类.....                  | 错误!未定义书签。 |
| 4.2   | 开始使用编辑器类编写工具.....          | 错误!未定义书签。 |
| 4.2.1 | 为项目添加脚本.....               | 错误!未定义书签。 |
| 4.2.2 | 创建指定名称的文件夹.....            | 错误!未定义书签。 |
| 4.3   | 把工具添加到菜单.....              | 错误!未定义书签。 |
| 4.3.1 | CreateWizard函数.....        | 错误!未定义书签。 |
| 4.3.2 | 测试脚本的实现效果.....             | 错误!未定义书签。 |
| 4.4   | 读取场景中选择的对象.....            | 错误!未定义书签。 |
| 4.4.1 | 在脚本中使用Selection类.....      | 错误!未定义书签。 |
| 4.4.2 | 测试脚本的实现效果.....             | 错误!未定义书签。 |
| 4.5   | 为工具窗口添加用户输入框.....          | 错误!未定义书签。 |
| 4.6   | 完成工具的所有功能.....             | 错误!未定义书签。 |
| 第 5 章 | 图片与几何图形对象.....             | 错误!未定义书签。 |
| 5.1   | 2D游戏常用的图片.....             | 错误!未定义书签。 |
| 5.1.1 | 精灵.....                    | 错误!未定义书签。 |
| 5.1.2 | 图块集.....                   | 错误!未定义书签。 |
| 5.1.3 | 图形绘制中的问题.....              | 错误!未定义书签。 |
| 5.1.4 | 设想.....                    | 错误!未定义书签。 |
| 5.2   | 开始编写编辑器工具.....             | 错误!未定义书签。 |
| 5.3   | 设置四边形的轴点.....              | 错误!未定义书签。 |
| 5.4   | 指定四边形资源的存放路径.....          | 错误!未定义书签。 |
| 5.5   | 生成四边形.....                 | 错误!未定义书签。 |
| 5.5.1 | 阶段一：创建构成四边形的顶点、UV和三角形..... | 错误!未定义书签。 |
| 5.5.2 | 阶段二：在资源面板中生成四边形.....       | 错误!未定义书签。 |
| 5.5.3 | 阶段三：在场景中实例化一个四边形.....      | 错误!未定义书签。 |
| 5.6   | 使用四边形生成工具.....             | 错误!未定义书签。 |
| 第 6 章 | 生成纹理图集.....                | 错误!未定义书签。 |
| 6.1   | 为什么要使用纹理图集.....            | 错误!未定义书签。 |

|       |                       |           |
|-------|-----------------------|-----------|
| 6.1.1 | 降低绘制调用的次数             | 错误!未定义书签。 |
| 6.1.2 | 便于灵活的使用纹理             | 错误!未定义书签。 |
| 6.1.3 | 便于管理纹理                | 错误!未定义书签。 |
| 6.2   | 开始编写生成纹理图集的工具         | 错误!未定义书签。 |
| 6.3   | 添加组成纹理图集的纹理           | 错误!未定义书签。 |
| 6.4   | UV对纹理图集的重要性           | 错误!未定义书签。 |
| 6.5   | 生成纹理图集                | 错误!未定义书签。 |
| 6.5.1 | 步骤一：优化输入的纹理           | 错误!未定义书签。 |
| 6.5.2 | 步骤二：构建纹理图集            | 错误!未定义书签。 |
| 6.5.3 | 步骤三：保存图集的预置           | 错误!未定义书签。 |
| 6.6   | 脚本文件TexturePacker代码汇总 | 错误!未定义书签。 |
| 6.7   | 测试工具的使用效果             | 错误!未定义书签。 |
| 第 7 章 | UV和动画                 | 错误!未定义书签。 |
| 7.1   | 生成一个可停靠的编辑器           | 错误!未定义书签。 |
| 7.2   | 编辑工具窗口的界面             | 错误!未定义书签。 |
| 7.2.1 | 添加预置资源选择区域            | 错误!未定义书签。 |
| 7.2.2 | 添加纹理选择区域              | 错误!未定义书签。 |
| 7.2.3 | 添加纹理选择的两种方式           | 错误!未定义书签。 |
| 7.2.4 | 编写用于修改网格对象UV坐标的函数     | 错误!未定义书签。 |
| 7.2.5 | 添加应用所有设置的按钮           | 错误!未定义书签。 |
| 7.3   | 工具脚本代码的汇总与使用          | 错误!未定义书签。 |
| 7.4   | 一个播放动画的平面对象           | 错误!未定义书签。 |
| 第 8 章 | 基于 2D 游戏的摄像机与场景设置     | 错误!未定义书签。 |
| 8.1   | 摄像机类型：透视与正交           | 错误!未定义书签。 |
| 8.2   | 世界单元与像素               | 错误!未定义书签。 |
| 8.3   | 世界单元与像素的转换            | 错误!未定义书签。 |
| 8.3.1 | 添加纹理和四边形对象            | 错误!未定义书签。 |
| 8.3.2 | 调整四边形与摄像机的位置          | 错误!未定义书签。 |
| 8.3.3 | 世界单元：像素 = 1: 1        | 错误!未定义书签。 |
| 8.3.4 | 对齐屏幕和场景坐标的原点          | 错误!未定义书签。 |
| 8.4   | 纹理图片的完美显示             | 错误!未定义书签。 |
| 8.5   | 其它有用的设置技巧             | 错误!未定义书签。 |
| 8.5.1 | 调节深度                  | 错误!未定义书签。 |
| 8.5.2 | 合成视图                  | 错误!未定义书签。 |
| 第 9 章 | 获取玩家对 2D 游戏的输入        | 错误!未定义书签。 |
| 9.1   | 自动检测鼠标单击事件            | 错误!未定义书签。 |
| 9.2   | 手动检测鼠标单击事件            | 错误!未定义书签。 |
| 9.2.1 | 鼠标按下的键及其位置            | 错误!未定义书签。 |
| 9.2.2 | 鼠标点击的第一个对象            | 错误!未定义书签。 |
| 9.2.3 | 鼠标点击的所有对象             | 错误!未定义书签。 |
| 9.3   | 修改游戏中的鼠标图标            | 错误!未定义书签。 |
| 9.3.1 | 准备所需的资源，并做适当设置        | 错误!未定义书签。 |
| 9.3.2 | 编写脚本                  | 错误!未定义书签。 |
| 9.3.3 | 两个坐标系导致的问题            | 错误!未定义书签。 |



|                               |           |
|-------------------------------|-----------|
| 9.3.4 查看游戏视图中的效果.....         | 错误!未定义书签。 |
| 9.4 使用键盘控制鼠标移动.....           | 错误!未定义书签。 |
| 9.5 对输入的抽象——输入轴.....          | 错误!未定义书签。 |
| 9.5.1 了解输入轴.....              | 错误!未定义书签。 |
| 9.5.2 输入轴在输入过程中的应用.....       | 错误!未定义书签。 |
| 9.6 来自移动设备的输入.....            | 错误!未定义书签。 |
| 9.6.1 检测移动设备上的触摸操作.....       | 错误!未定义书签。 |
| 9.6.2 把触摸操作当作鼠标操作.....        | 错误!未定义书签。 |
| 9.6.3 有选择的编译代码.....           | 错误!未定义书签。 |
| 第 10 章 2D 卡片游戏——记忆大作战.....    | 错误!未定义书签。 |
| 10.1 游戏设计文档.....              | 错误!未定义书签。 |
| 10.2 开始着手创建游戏.....            | 错误!未定义书签。 |
| 10.2.1 在资源面板创建文件夹.....        | 错误!未定义书签。 |
| 10.2.2 创建一个纹理图集.....          | 错误!未定义书签。 |
| 10.2.3 创建四边形对象.....           | 错误!未定义书签。 |
| 10.2.4 修改四边形的材质和UV.....       | 错误!未定义书签。 |
| 10.2.5 设置摄像机和游戏视图的分辨率.....    | 错误!未定义书签。 |
| 10.3 设置场景中的卡片.....            | 错误!未定义书签。 |
| 10.3.1 设置卡片的属性.....           | 错误!未定义书签。 |
| 10.3.2 定位卡片的位置.....           | 错误!未定义书签。 |
| 10.3.3 编写控制卡片行为的脚本.....       | 错误!未定义书签。 |
| 10.3.4 补全场景中其余的卡片.....        | 错误!未定义书签。 |
| 10.4 游戏管理类.....               | 错误!未定义书签。 |
| 10.4.1 重置卡片.....              | 错误!未定义书签。 |
| 10.4.2 处理玩家输入.....            | 错误!未定义书签。 |
| 10.4.3 响应玩家输入.....            | 错误!未定义书签。 |
| 10.4.4 游戏管理类代码汇总.....         | 错误!未定义书签。 |
| 10.5 完善并运行游戏.....             | 错误!未定义书签。 |
| 10.5.1 替换系统鼠标图标.....          | 错误!未定义书签。 |
| 10.5.2 游戏运行效果展示.....          | 错误!未定义书签。 |
| 第 11 章 可联机玩的游戏——记忆大作战.....    | 错误!未定义书签。 |
| 11.1 网络连接.....                | 错误!未定义书签。 |
| 11.2 建立服务器端.....              | 错误!未定义书签。 |
| 11.3 建立客户端.....               | 错误!未定义书签。 |
| 11.4 测试网络连接的功能.....           | 错误!未定义书签。 |
| 11.5 网络视图组件.....              | 错误!未定义书签。 |
| 11.6 构建授权服务器.....             | 错误!未定义书签。 |
| 11.7 建立游戏输入操作的秩序.....         | 错误!未定义书签。 |
| 11.7.1 游戏启动时，禁止输入操作.....      | 错误!未定义书签。 |
| 11.7.2 连接建立后，允许服务器端的输入操作..... | 错误!未定义书签。 |
| 11.7.3 服务器端远程调用客户端上的函数.....   | 错误!未定义书签。 |
| 11.7.4 客户端远程调用服务器端上的函数.....   | 错误!未定义书签。 |
| 11.8 修改游戏管理类脚本.....           | 错误!未定义书签。 |
| 11.9 游戏运行效果展示.....            | 错误!未定义书签。 |

|                            |           |
|----------------------------|-----------|
| 11.10 为游戏添加分数记录.....       | 错误!未定义书签。 |
| 第 12 章 优化游戏的方法.....        | 错误!未定义书签。 |
| 12.1 最优化，如你所想吗？.....       | 错误!未定义书签。 |
| 12.2 减少顶点的数目.....          | 错误!未定义书签。 |
| 12.3 减少材质.....             | 错误!未定义书签。 |
| 12.4 减少UV接缝.....           | 错误!未定义书签。 |
| 12.5 不同平台下，纹理的不同设置.....    | 错误!未定义书签。 |
| 12.6 对象缓存组件.....           | 错误!未定义书签。 |
| 12.7 避免频繁使用Update()函数..... | 错误!未定义书签。 |
| 12.8 合理使用Collider组件.....   | 错误!未定义书签。 |
| 12.9 避免使用OnGUI()和GUI类..... | 错误!未定义书签。 |
| 12.10 使用静态批处理.....         | 错误!未定义书签。 |
| 12.11 使用天空盒子.....          | 错误!未定义书签。 |

大学霸

www.daxueba.net

## 第 1 章 Unity 及其组成的介绍

本书主要讲解的是，如何使用 Unity 开发 2D 游戏。但在开始讲解之前，最好先熟悉一下 Unity 这个工具。本章会首先介绍 Unity 的下载和安装，然后会介绍 Unity 界面的各组成部分，这些知识会在本书后面的章节频繁使用，所以不要掉以轻心。

### 1.1 Unity 概述

Unity 现如今已是非常的流行，因此在开始学习 2D 游戏开发之前，本节就来简要说明下 Unity，及其下载和安装方法。

#### 1.Unity 简介

Unity 是一款跨平台的专业游戏引擎，可以使用它轻松的开发各种 2D 和 3D 游戏，然后部署到各种游戏平台上。当然也包括这些主流游戏平台：Windows、iOS、Android、Xbox 360、PS3。

#### 2.Unity 的下载

Unity 的安装包可以从它的官方网站下载到，如图 1-1 所示。

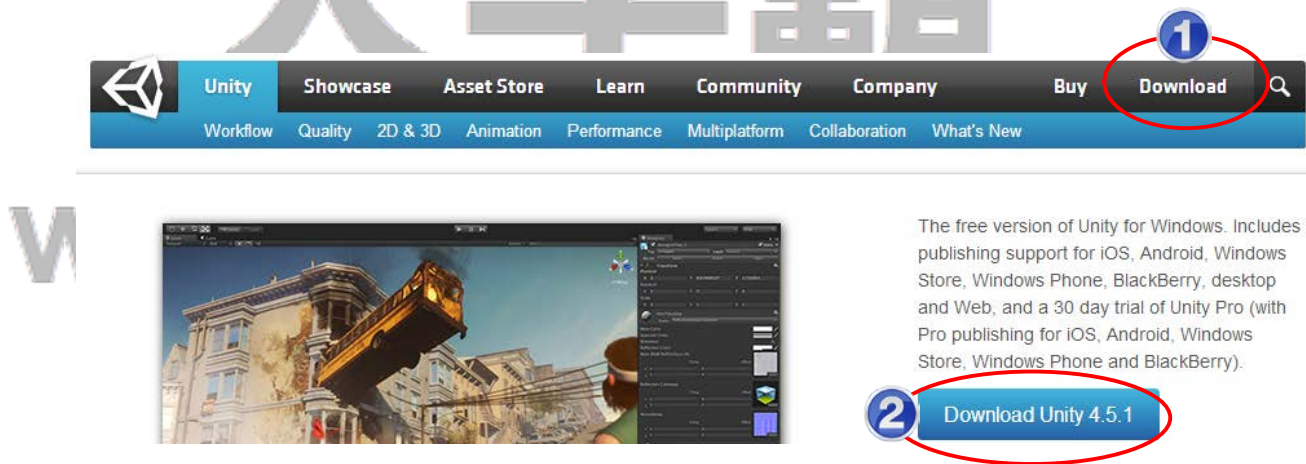


图 1-1 从官网上下载 Unity

链接是：<http://unity3d.com/unity/download>。写作本书时，可以下载到的最新版是 Unity 4.5.1，大小是 1.09G，下载到的安装包如图 1-2 所示。

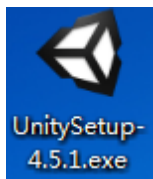


图 1-2 下载到的 Unity 安装包



### 3.Unity 的安装

双击此安装包，即可开始 Unity 的安装过程。如图 1-3、图 1-4 和图 1-5 所示。

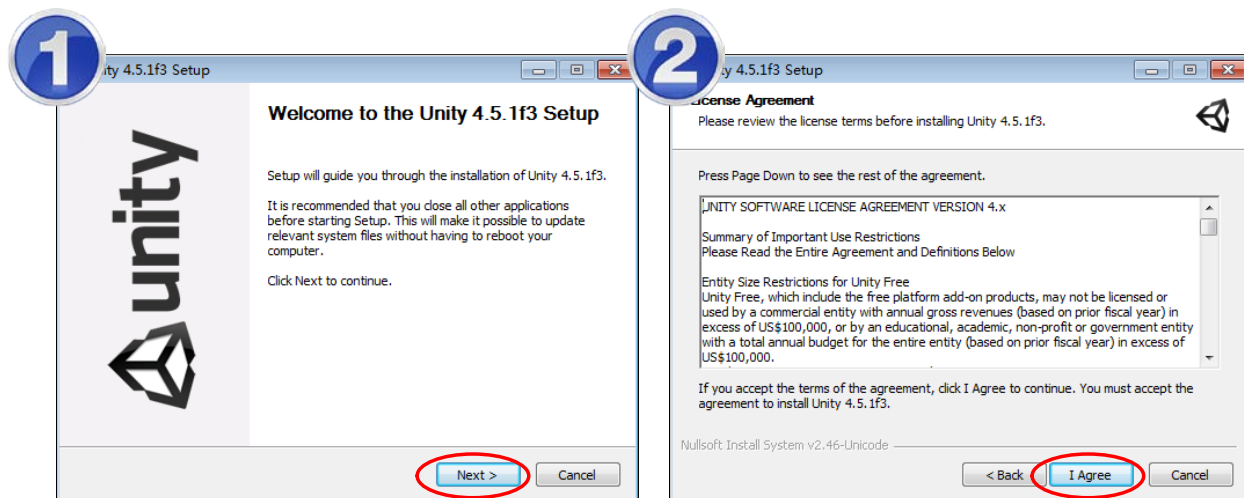


图 1-3 Unity 安装过程 1

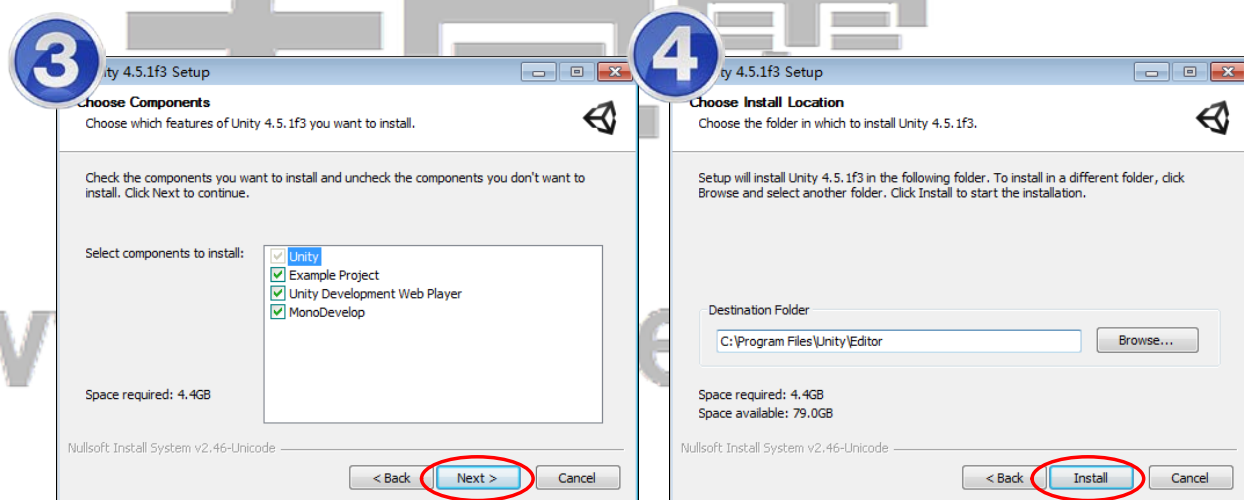


图 1-4 Unity 安装过程 2

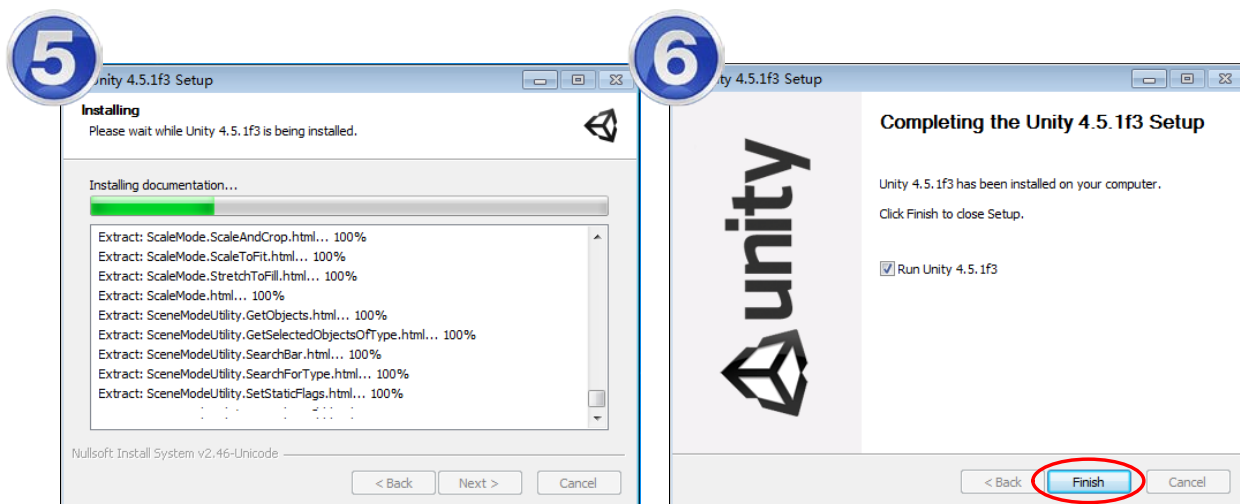


图 1-5 Unity 安装过程 3

整个安装过程十分的简单，除了指定一个软件的安装位置外，其它保留默认设置即可，大约用 15 分钟即可完成。双击安装好的 Unity 软件应用程序图标，选择免费的版本，对于初学者而言，这个版本的功能足够使用，等到以后有需求了再安装收费但是功能更强大的版本也不迟，如图 1-6 所示。

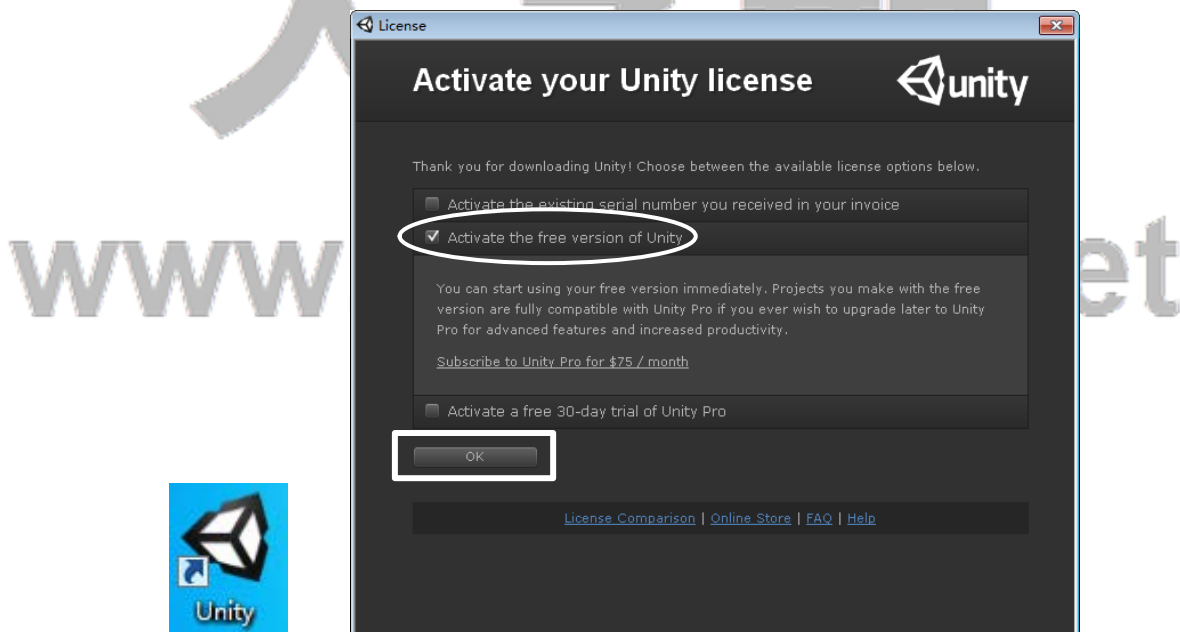


图 1-6 双击软件应用图标，选择免费版本

接下来，读者可以用不到 1 分钟的时间，使用邮箱注册一个帐号，并且在登录这个帐号以后，就可开始使用 Unity 这个软件了，如图 1-7 所示。

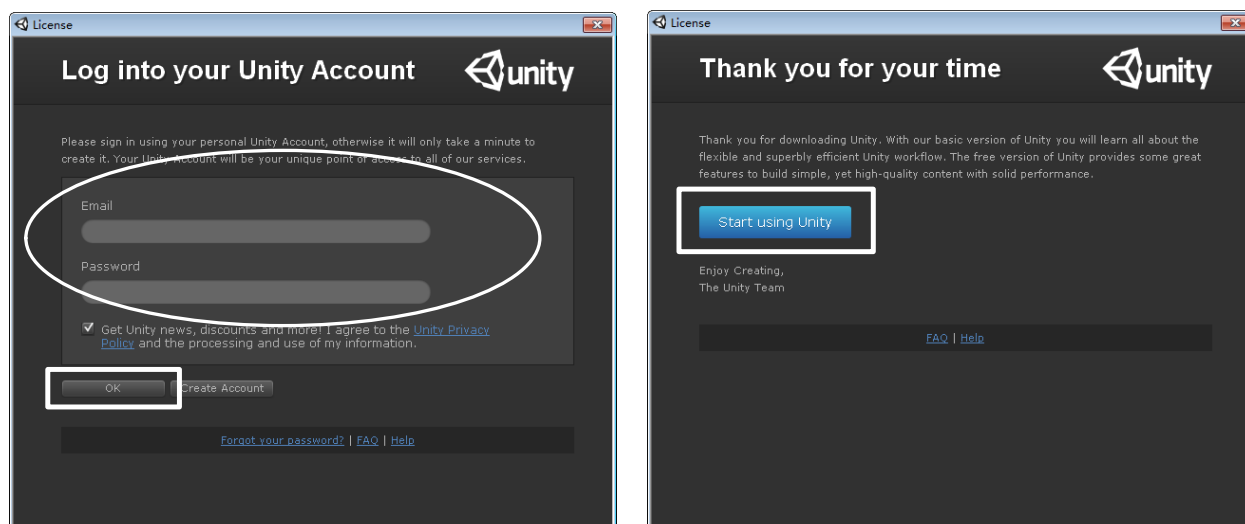


图 1-7 注册并登录帐号，然后就可以开始使用 Unity 了

提示：帐号的用途还是很多的，例如，在登录了帐号以后，可以在官网下载一些游戏示例来研究。

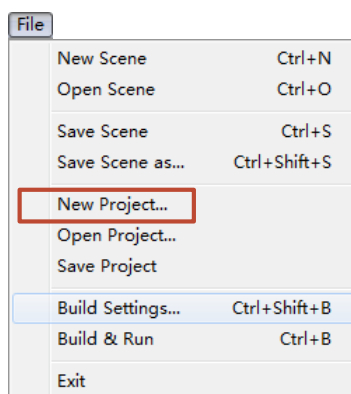
## 1.2 项目、资源和场景

如果使用 Unity 制作游戏，就一定会接触到项目（Project）、资源（Asset）和场景（Scene）。本节将依次介绍它们。

### 1.2.1 项目

Unity 是一个基于项目的应用。这意味着每开发一个新游戏，都要创建一个新项目。一个项目就代表一个游戏，不管游戏是 2D 还是 3D 的。开发人员可以把项目当做容器，它包含了开发游戏时，自动生成还有引入的所有文件。

要在 Unity 里创建一个新项目，可以单击 File|New Project 命令，如图 1-8 所示，这就像是在和 Unity 说“我想要制作一个新游戏”，于是 Unity 就会给你创建一个项目。



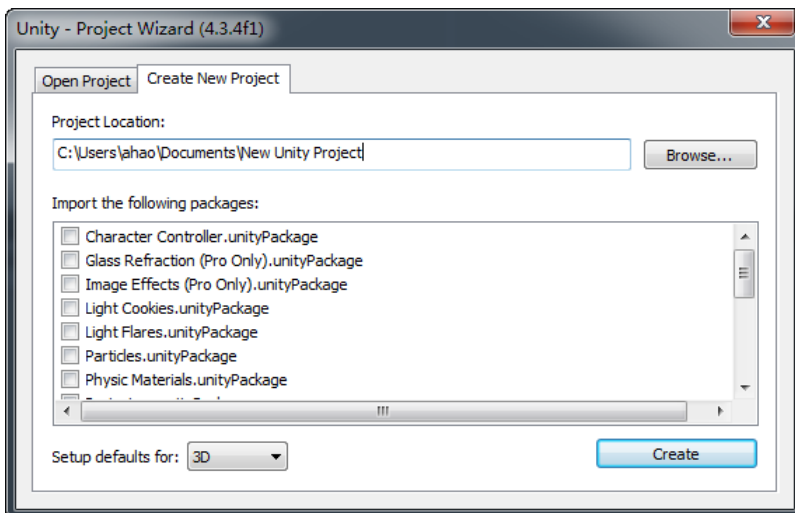


图 1-8 单击命令，创建项目

图 1-9 Project Wizard 对话框

单击命令后弹出的 **Project Wizard**（项目向导）对话框如图 1-9 所示。这个对话框允许你指定项目的存放位置，以及导入 Unity 预置的一些资源到项目。默认情况下，只指定存放位置就可以了，预置的资源可以在以后导入。如果设置好了，可以单击 **Create** 按钮，然后就生成了一个新的项目。一旦项目生成，Unity 就会显示默认的界面，如图 1-10 所示。

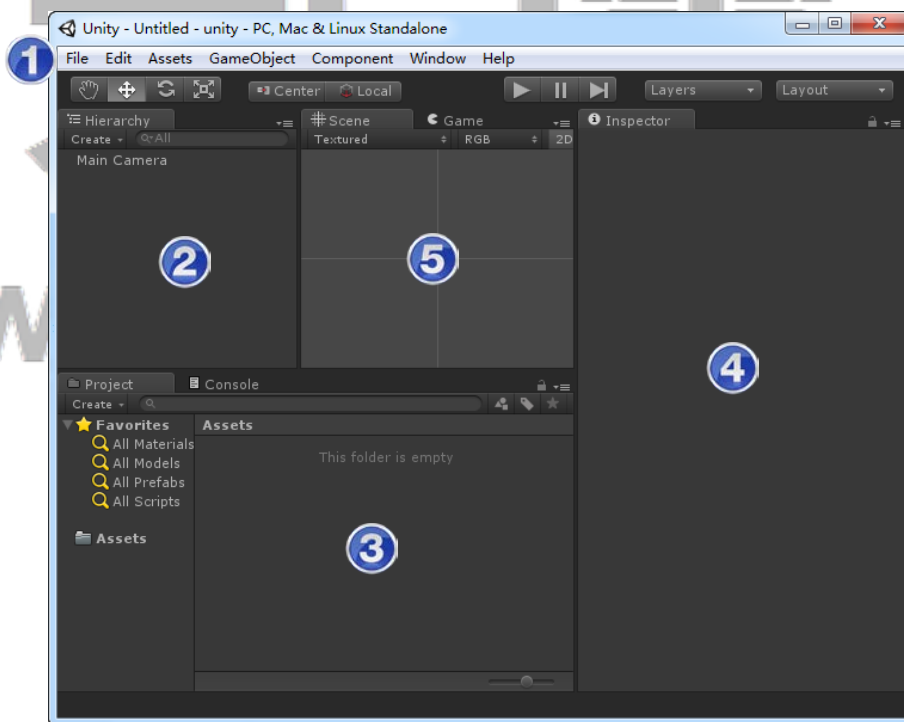


图 1-10 Unity 界面的默认布局：①是应用程序菜单②是层次面板③是资源面板④是查看器⑤是场景视图

同时，在指定的项目存放位置下，生成了 4 个子文件夹：**Library**、**Assets**、**ProjectSettings** 和 **Temp**，如图 1-11 所示。

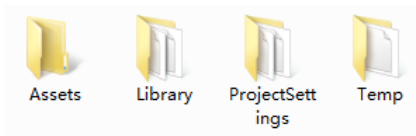


图 1-11 Unity 在项目文件夹里生成的 4 个子文件夹

## 1.2.2 资源

开发游戏一定会使用很多东西，如网格、纹理、电影、动画、声音、音乐、文本等等。这些文件都被 Unity 称为资源（Asset）。只有导入到 Unity 中的资源，才可以在游戏开发的过程中使用，所以在使用资源之前，需要把资源导入到项目中。导入资源到项目的方法有两种：

- ❑ 单击 Asset|Import New Asset 命令，在弹出的 Import New Asset 对话框中，找到存放资源的位置，选中后单击 Import 按钮即可，如图 1-12 所示。

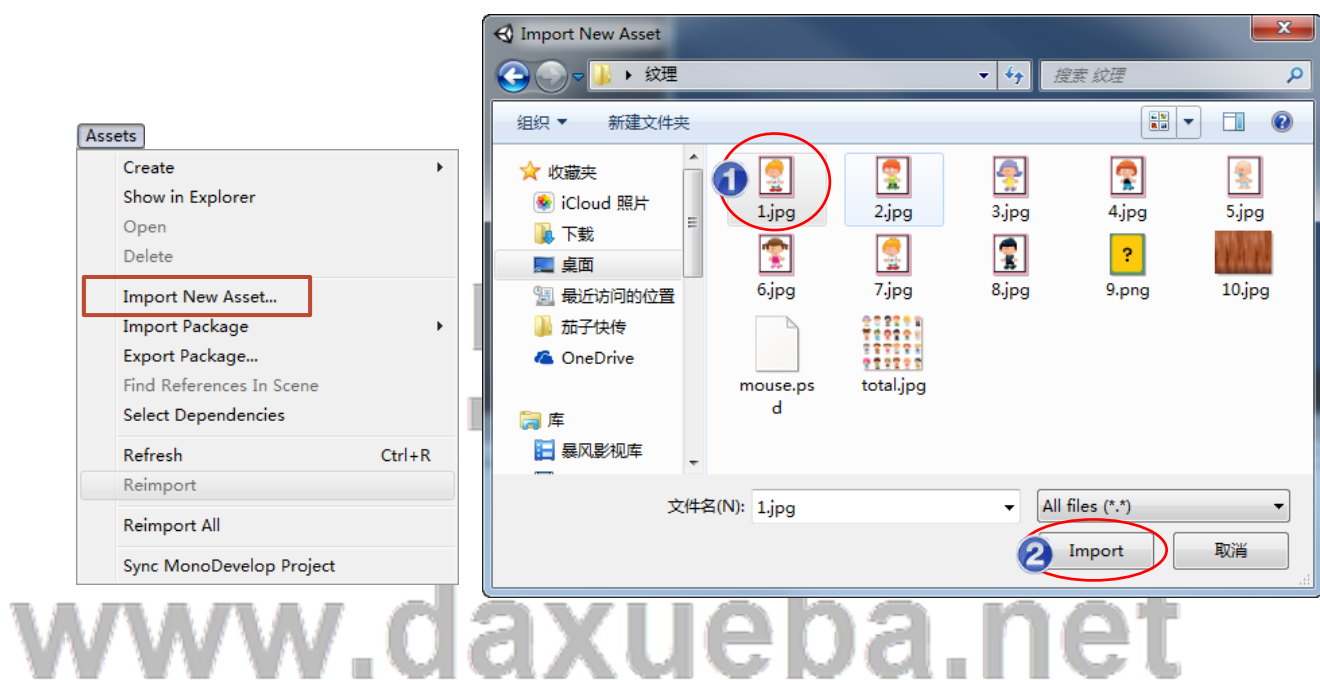


图 1-12 导入资源到项目的方法一

- ❑ 直接拖拽文件到资源面板，如图 1-13 所示。

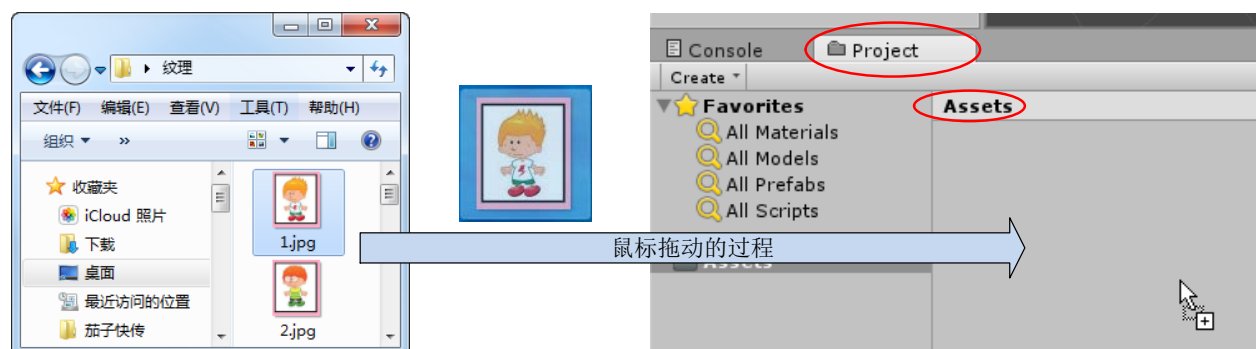


图 1-13 导入资源到项目的方法二

两种方法都可以将资源导入到项目中，而导入的资源会显示在 Unity 编辑器的资源面板中，如图 1-14 所示。



图 1-14 导入到项目中的各类资源

第一种方法，一次只能导入一个资源到项目；第二种方法，一次可以导入多个资源，只要用鼠标选中多个资源，然后拖动就可以了。

补充：Unity 可以识别的资源类型如表 1.1 所示。

表 1.1 可被Unity的文件格式

| 网格 (meshes) | 纹理 (Texture) | 音频 (Audio) | 视频 (Movie) |
|-------------|--------------|------------|------------|
| .FBX        | .PSD         | .MP3       | .MOV       |
| .MA         | .TIFF        | .OGG       | .AVI       |
| .MB         | .PNG         | .MOD       | .OGG       |
| .MAX        | .BMP         | .IT        | .ASF       |
| .BLEND      | .JPG         | .XM        | .MPG       |
| .3DS        | .TGA         | .S3M       |            |
| .DXF        | .DDS/PVR     | .WAV       |            |
| .C4D        |              |            |            |

开发游戏，一定会用到很多资源。如果全部直接放置在资源面板下，一定会非常的混乱，如果能将资源按照类别和功能放到不同的地方就好了。要做到这点，可以在资源面板中创建文件夹，然后放置资源到资源面板里对应的文件夹里即可。

要在资源面板里创建文件夹，可以在资源面板里单击鼠标右键，从弹出的快捷菜单中单击 CreateFolder 命令。如图 1-15 所示，在资源面板中创建了一个名为 Texture 的文件夹。

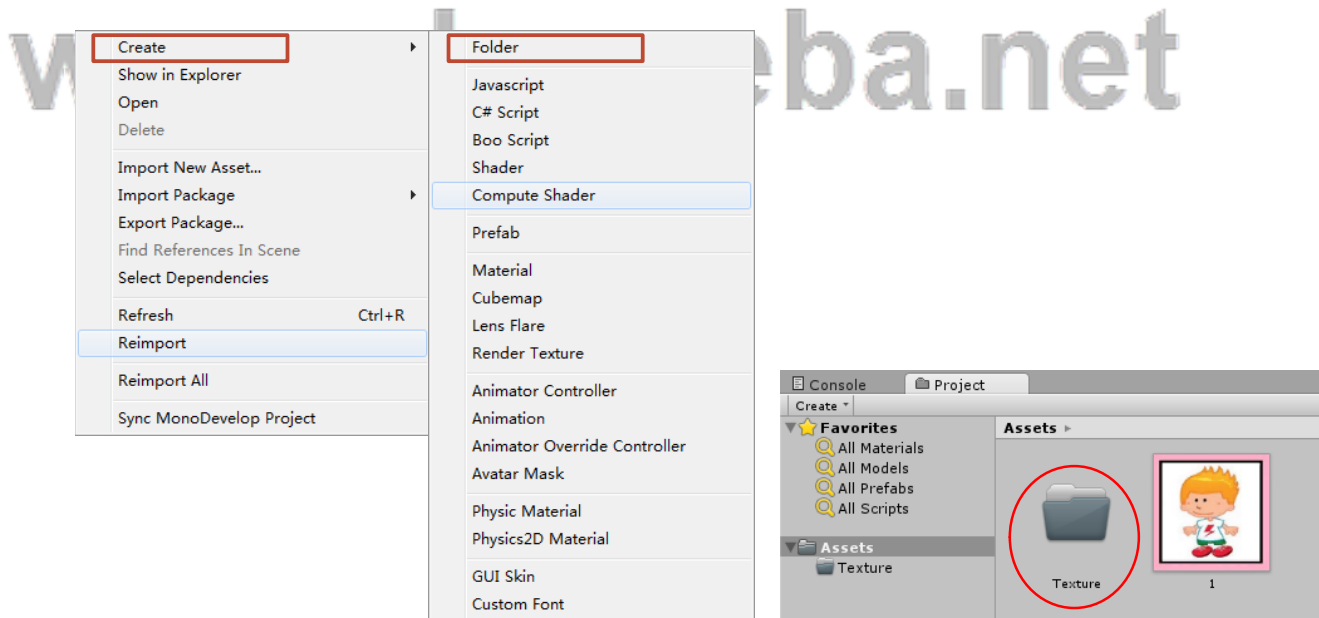


图 1-15 在资源面板中创建文件夹

然后选中对应的资源，拖动到文件夹中，就完成了资源的整理。



### 1.2.3 场景

在 Unity 中，场景（Scene）就是游戏开发者制作游戏时，所使用的游戏场景。它是一个三维空间，对应的三维坐标轴分别是 X 轴、Y 轴和 Z 轴。

要创建一个新的场景，只需单击 File|New Scene 命令，或者按下快捷键 Ctrl+N，如图 1-16 所示。

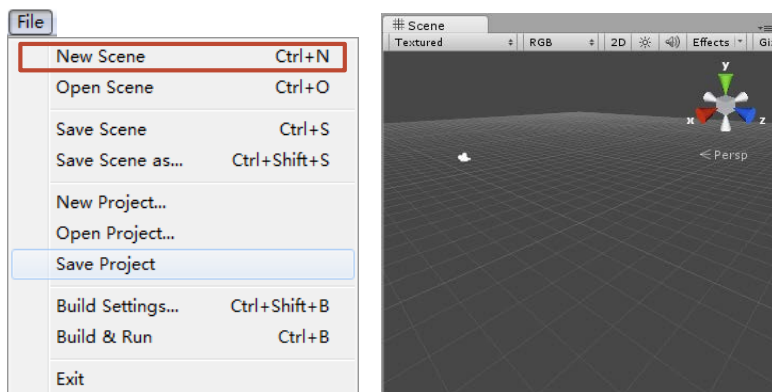


图 1-16 创建程序的命令，以及场景

默认情况下，新创建游戏项目的同时，也新创建了游戏的场景，只不过还没有保存罢了。使用快捷键 Ctrl+S 即可保存场景，保存后的场景会被作为一种资源而显示在资源面板中，如图 1-17 所示，保存了名为 envr 的场景。

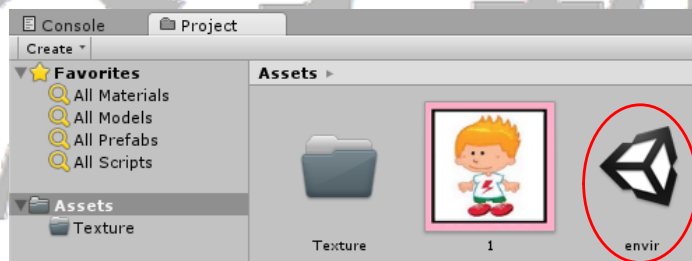
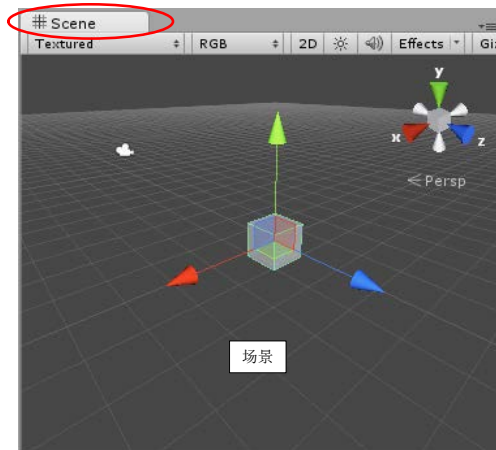


图 1-17 被保存的场景，作为资源显示在了资源面板中

## 1.3 场景视图的操作

只要在资源面板里双击场景资源，就可以在场景视图中查看这个场景了，如图 1-18 所示。为了在场景视图中多角度的查看游戏中的各种元素，掌握一些场景视图的操作方法是很有必要的，本节主要介绍了两种方法：快捷键和工具。



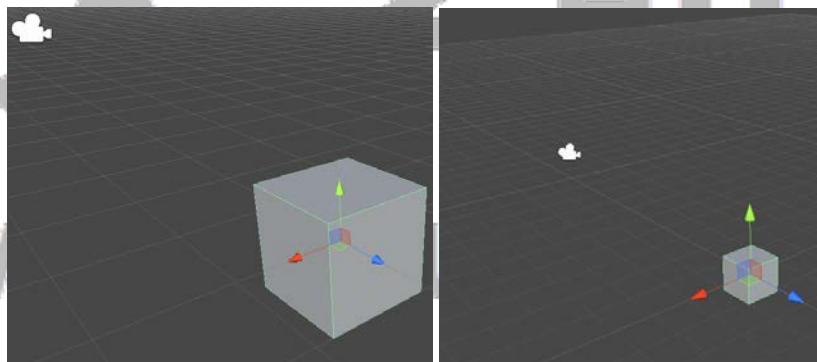
场景视图

图 1-18 在场景视图中查看场景

### 1.3.1 使用快捷键操作场景视图

使用快捷键操作场景视图，常用的操作方式有：

- 使用鼠标的滚轮，可以控制游戏视图与场景中各元素距离的远近，如图 1-19 所示，向上滚动靠近，向下滚动远离。



向上滚动

向下滚动

图 1-19 使用鼠标的滚轮，控制游戏视图与场景中各元素距离的远近

- 使用鼠标双击层次视图上的对象名，场景视图会移动，直到对应的对象处于场景视图的中间，如图 1-20 所示。当开发这在场景中找到对应的对象时，可以使用这种方法快速找到。

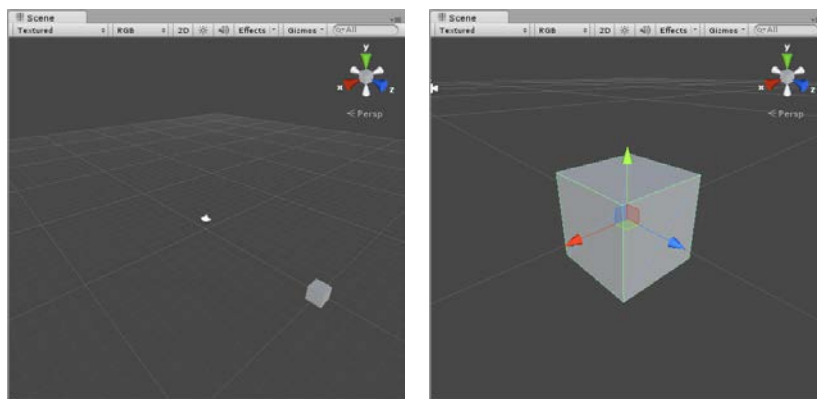




图 1-20 鼠标双击对象名，场景视图速度定位到对应对象，且被显示在场景视图的中央

- 在场景视图里，按下鼠标中间的按钮（或滚轮），鼠标变成了 ，然后移动鼠标，可以任意移动场景，如图 1-21 所示。
- 在场景视图里，按下键盘上的 Alt 键，鼠标就会变成 ，再按下鼠标的左键拖动，可围绕指定的游戏对象移动，如图 1-22 所示。

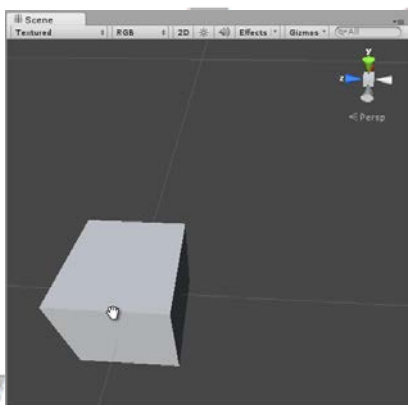


图 1-21 任意移动场景视图中的场景

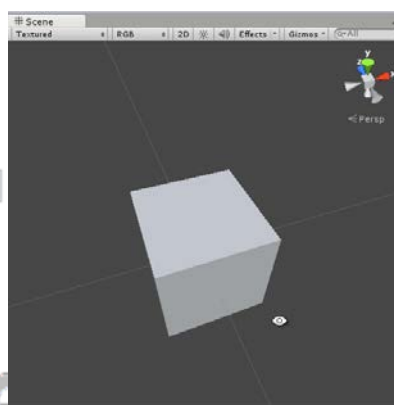



图 1-22 绕着场景视图中的对象移动

- 在场景视图里，按下鼠标的右键，鼠标就会变成 ，然后再按下键盘上的 W、S、A、D 键，就可以模拟第一人称在场景中向前、后、左和右移动，移动鼠标相当于转动人物的头部。如图 1-23 所示。

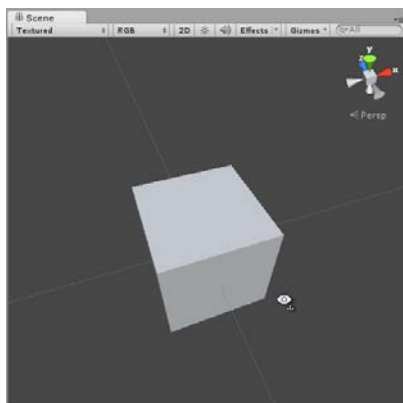


图 1-23 模拟第一人称，在场景中移动

### 1.3.2 使用 Gizmo 操作场景视图

场景视图的右上角有个 Gizmo 工具，如图 1-24 所示。

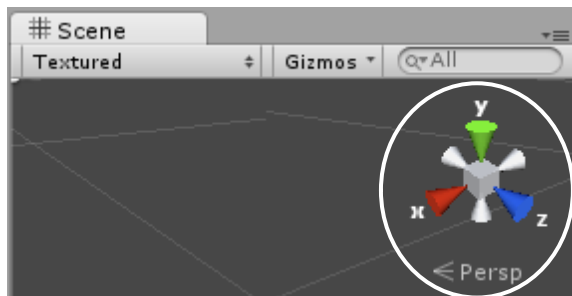
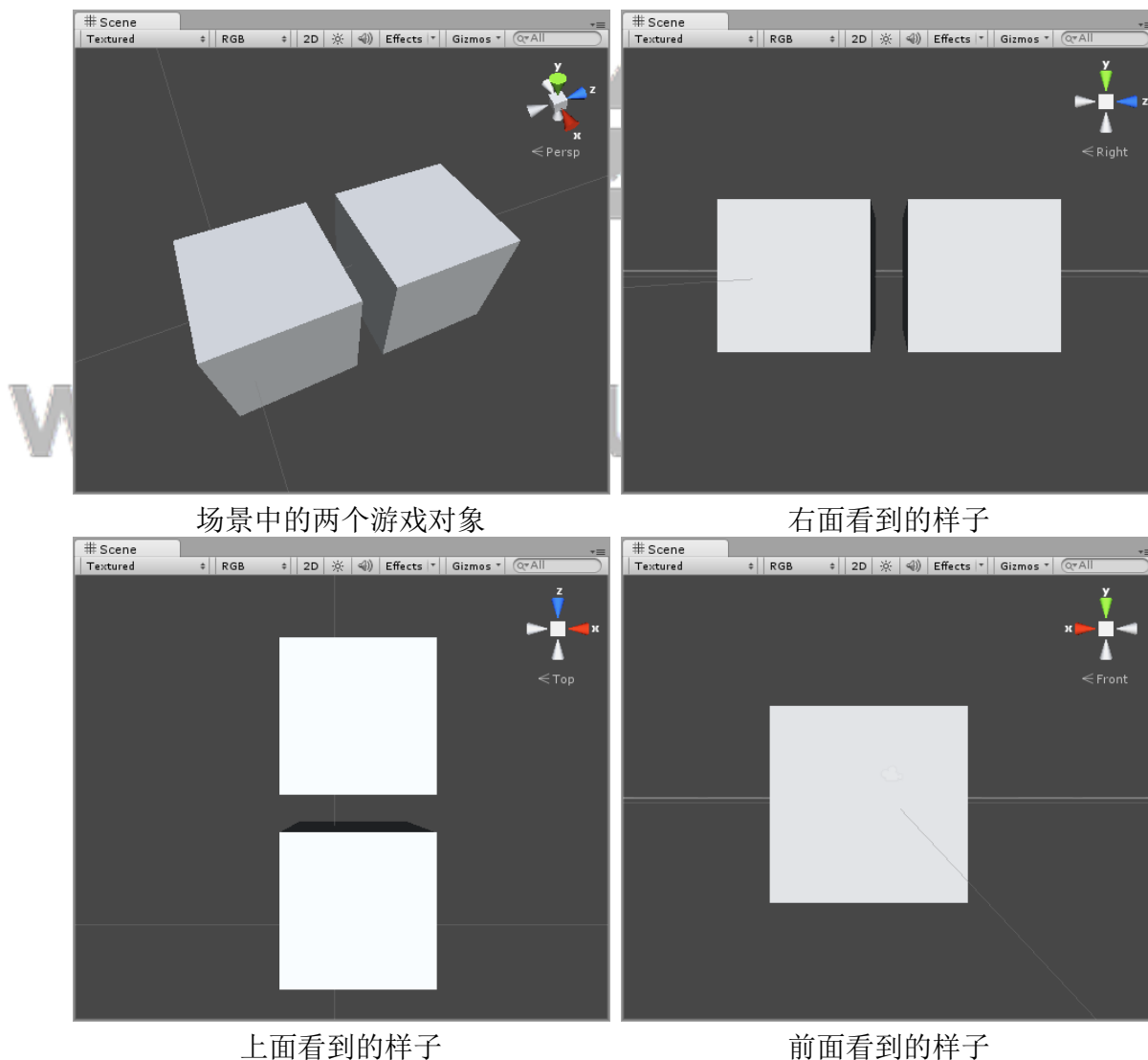


图 1-24 场景视图中的 Gizmo 工具

对于选中的游戏对象，使用它可以迅速切换不同的观察视角，只需要单击 Gizmo 的轴即可。如图 1-25，切换了三个视角来查看场景中的游戏对象。



场景中的两个游戏对象

右面看到的样子

上面看到的样子

前面看到的样子

图 1-25 使用不同的视角查看视图中的游戏对象

## 1.4 游戏对象和组件

游戏场景中的任何元素都属于游戏对象（GameObject），而且所有的游戏对象的名字还会被列在层次面板中，如图 1-26 所示。

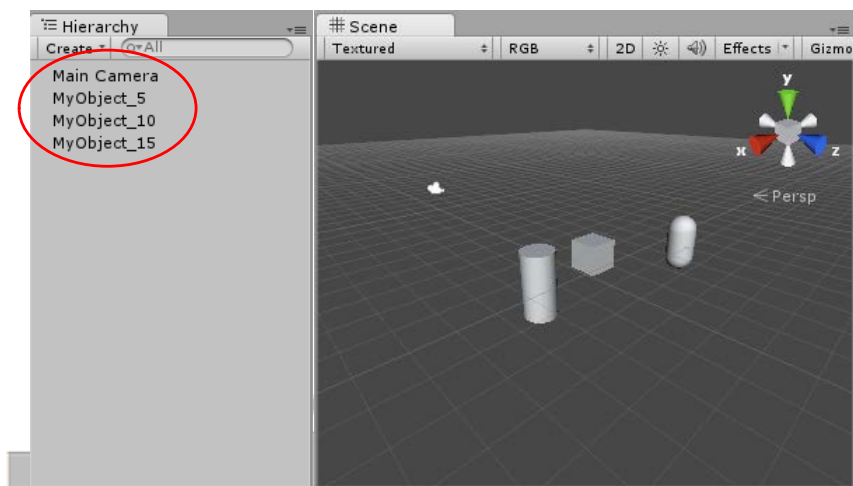


图 1-26 层次面板中列出了游戏场景中的所有游戏对象名

游戏对象并非场景的最小组成部分，每个游戏对象都是由组件（Component）构成的。当一个游戏对象在场景中被鼠标选中的时候，构成这个对象的组件就可以在查看器中看到。如图 1-27 所示，选中场景中的立方体，然后在查看器中看到它的构成组件是 Transform 组件、Mesh Filter 组件、Box Collider 组件、Mesh Renderer 组件。每个组件都有自己的属性，同样可以在查看器里看到，需要单击组件名左边的 ▶ 按钮即可。

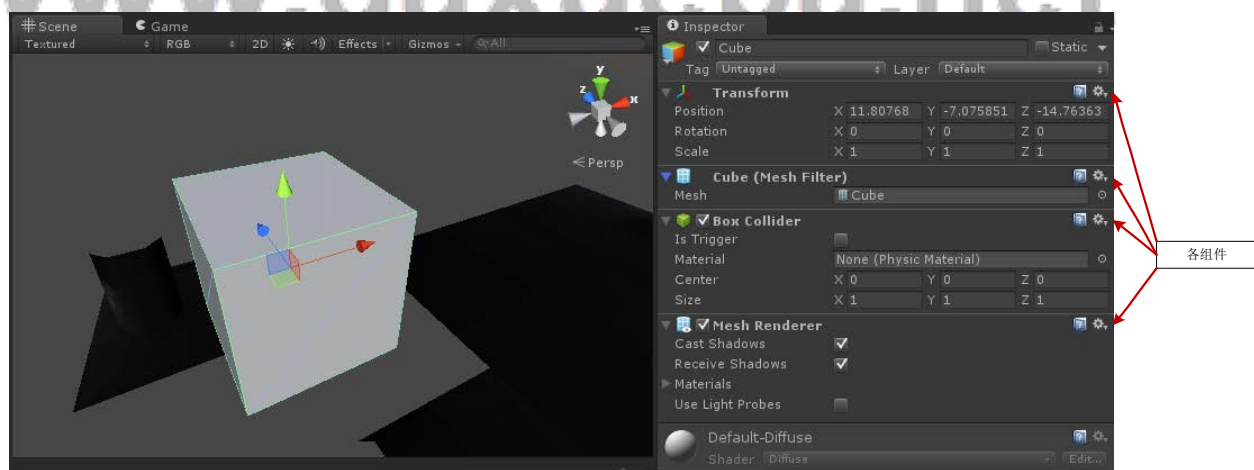


图 1-27 即使是最简单的立方体对象也是由组件构成的

游戏对象上的所有组件相互影响，并最终决定了对象的行为。各游戏对象也由于组件的不同而不同。游戏对象都有一个共同的组件，就是 Transform 组件，如图 1-28 所示。

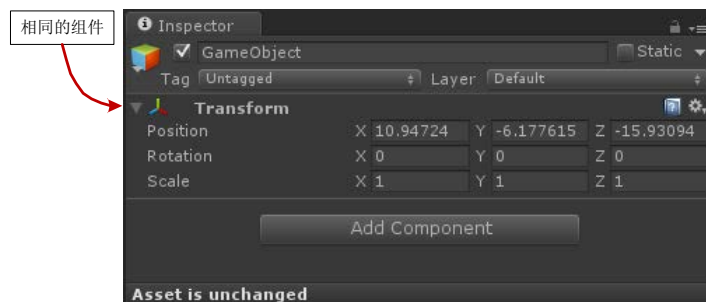


图 1-28 Transform 组件定义了坐标空间中 GameObject 的位置、旋转和大小

Transform 组件的作用是，修改游戏对象的位置、朝向和大小。它有三个属性：Position、Rotation 和 Scale。修改这些属性，就意味着修改了游戏对象的位置、朝向和大小。除此以外，还可以使用 Unity 工具栏上的对应按钮修改，如图 1-29 所示。✚ 负责位置、↺ 负责朝向、📏 负责大小。依次对应于键盘上的快捷键 W、E 和 R。

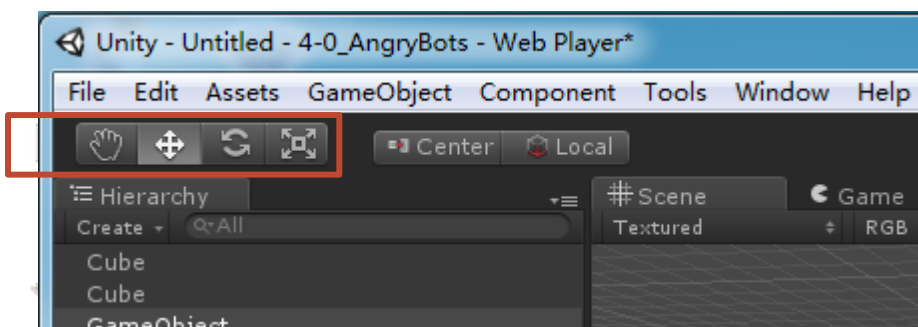


图 1-29 Unity 的工具栏

## 1.5 脚本与脚本编辑器

要使用 Unity 做到更多的事情，就必须熟悉脚本的编写。脚本可以使用三种语言编写，它们是 C#、JavaScript 和 Boo。推荐你选择其中的一种，然后将其应用于游戏项目中的脚本。

### 1.5.1 创建脚本

本书的示例将只使用 C# 编写的脚本。为了在你的游戏项目中创建一个新的脚本，可以在资源面板上单击鼠标的右键，在弹出的快捷菜单中，单击 Create|C# Script、Javascript 或者 Boo Script 命令，创建使用不用语言编写的脚本，如图 1-30 所示，生成的 C# 脚本文件，默认命名为 NewBehaviourScript。



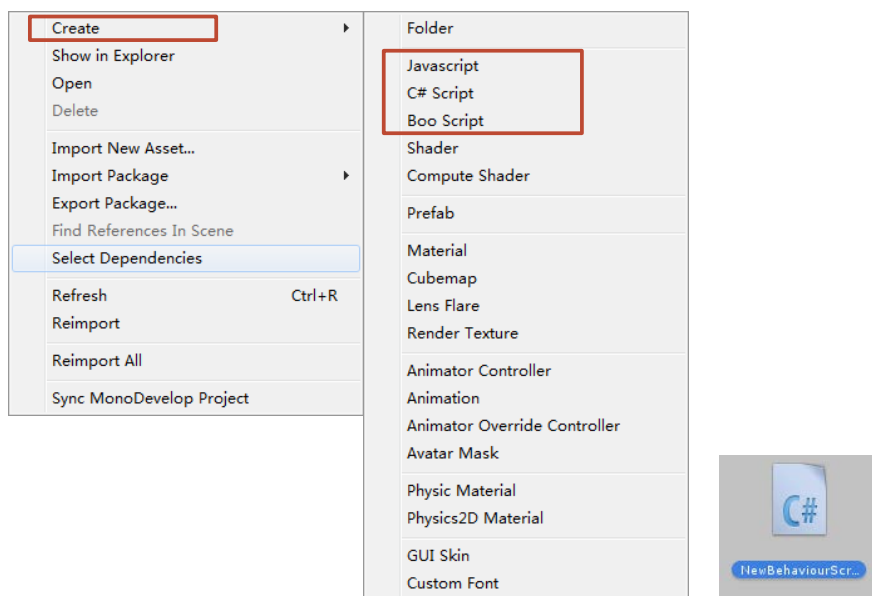


图 1-30 为游戏项目添加脚本文件

提示：同一游戏项目，可以包含不同种的脚本文件。也就是既可以有 C#编写的脚本，又可以有 JavaScript 编写的脚本。但建议如果读者是一个人在开发游戏的话，最好只使用最熟悉的一种语言编写脚本。

## 1.5.2 脚本编辑器

安装 Unity 时，也一并安装了一个脚本编辑器，名为“MonoDevelop”。当鼠标双击资源面板中的脚本文件时，脚本文件默认由 MonoDevelop 打开，如图 1-31 所示。

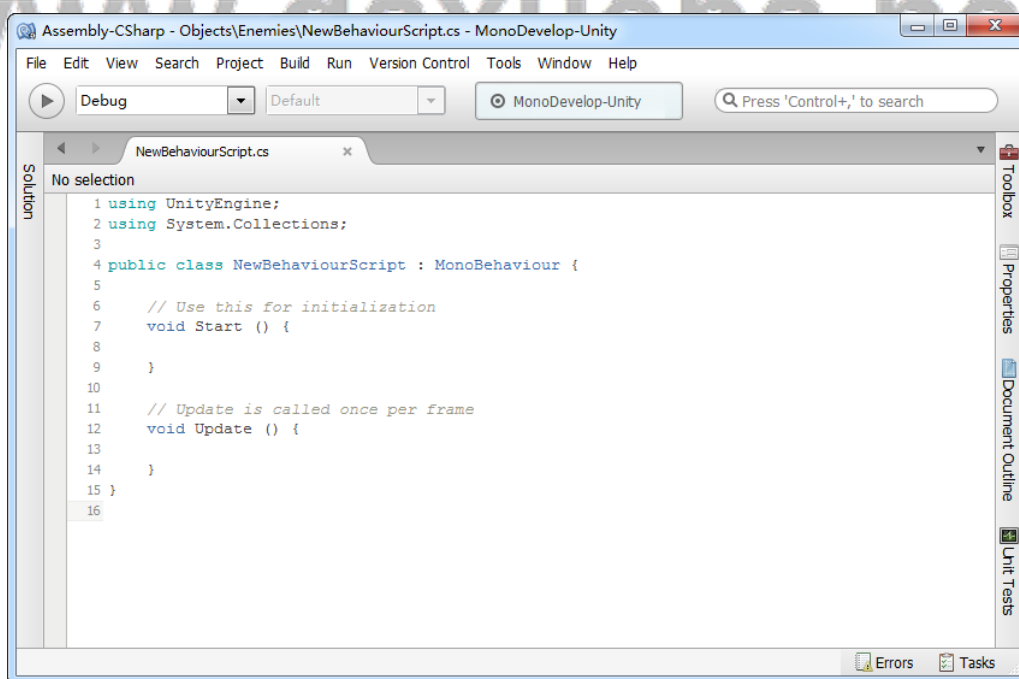


图 1-31 脚本编辑器 MonoDevelop

在 MonoDevelop 中编写了脚本代码以后，需要保存。当你的 Unity 编辑器处于活动状态时，它将自动检测文件的改变，然后编译脚本文件。如果有任何错误和警告信息，将会被列在 Unity 的控制台窗口，如图 1-32 所示。警告信息以黄色字体显示，错误信息以红色字体显示。

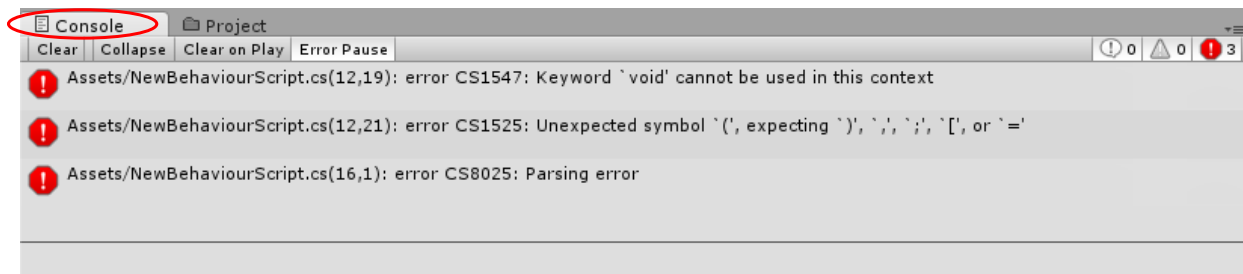


图 1-32 Unity 编辑器中的控制台窗口

每个新创建的 C#脚本文件里都会有模版代码，如下：

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class NewBehaviourScript : MonoBehaviour
05 {
06     // 在脚本初始化时，调用此函数
07     void Start ()
08     {
09
10     }
11     //游戏运行时，每一帧都调用此函数
12     void Update ()
13     {
14
15     }
16 }
```

代码 04 行，脚本定义了一个类，类名和脚本的文件名一致，继承自 **MonoBehaviour**。**MonoBehaviour** 是 UnityAPI 里预先定义的类。**MonoBehaviour** 几乎是所有组件的基类。

脚本也是一种组件，可以被赋予到游戏对象上，方法是拖动资源面板上的脚本文件到场景中的游戏对象上，或者拖动到层次面板中游戏对象的对象名上。一旦被赋予到游戏对象上，脚本就会成为游戏对象上的组件，被显示到了查看器中，如图 1-33 所示。

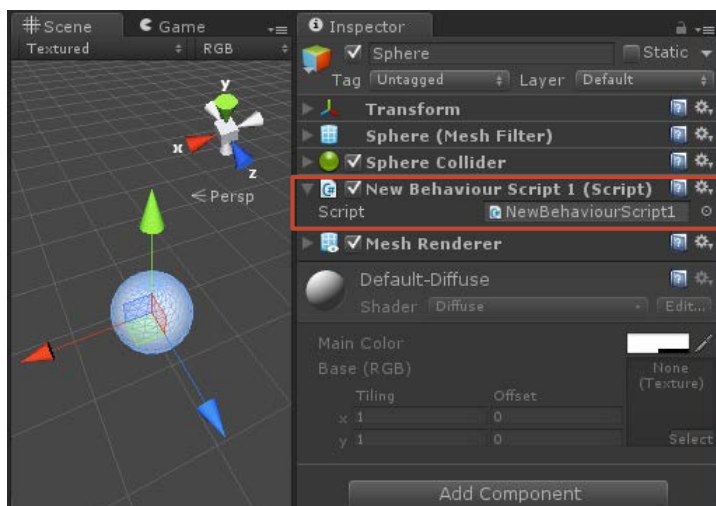


图 1-33 脚本可以被赋予到游戏对象上，然后成为后者的组件

如果不习惯使用 **MonoDevelop** 编写脚本文件，可以修改 Unity 默认打开脚本时所使用的脚本编辑器。以设置默认脚本编辑器为 **Microsoft Visual Studio 2010** 为例，修改方法是，单击 **Edit|Preferences** 命令，在弹出的 **Unity Preferences** 窗口中，选中左侧的 **External Tools**，然后修改右侧的 **External Script Editor** 为 **Microsoft Visual Studio 2010** 即可，如图 1-34 所示。

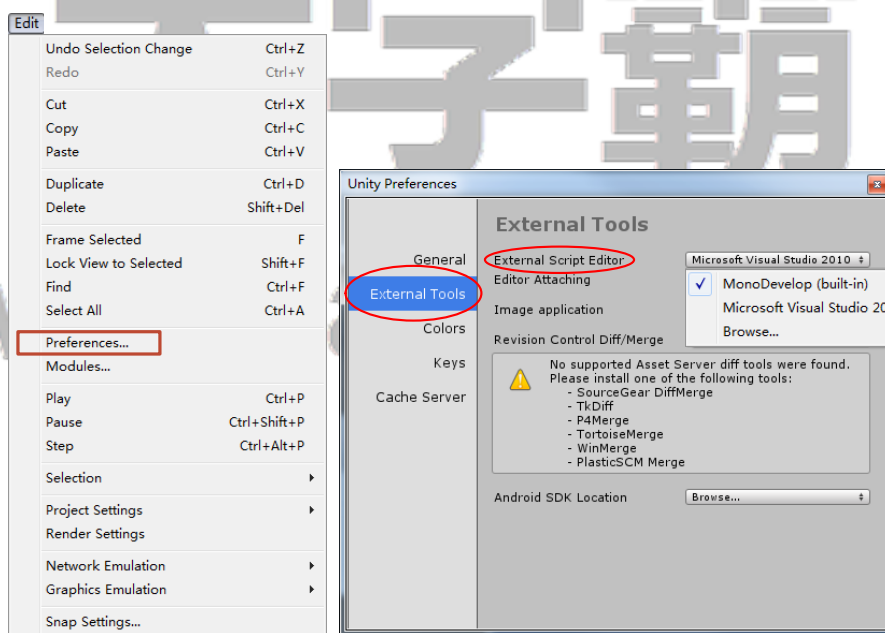


图 1-34 修改 Unity 设置的默认脚本编辑器

提示：要设置的脚本编辑器，应该提前安装在操作系统中。本书所使用的脚本代码都将由 **MonoDevelop** 编写。

## 1.6 脚本的调试

如果脚本中编写的代码，排除了语法上的错误后，游戏依然没有按照预期那样运行，就

肯定是逻辑上出了错。而逻辑上的问题只能通过代码的调试才能解决。本节将介绍两种代码调试的方法，相信在未来，一定会有所帮助的。

注意：本节讲解的是，在 MonoDevelop 中调试代码的方法。因此，需要设置 Unity 的默认脚本编辑器为 MonoDevelop，设置方法在本章前面介绍过了，这里不再重复。

### 1.6.1 调试方法一

打开 MonoDevelop，单击 Tools|Options 命令，在弹出的 Options 对话框中，选择对话框左侧的 Unity|Debugger，确保对话框右侧的 Editor Location 属性被设置成了 Unity.exe，且下面的两个复选框也被选中，如图 1-35 所示。

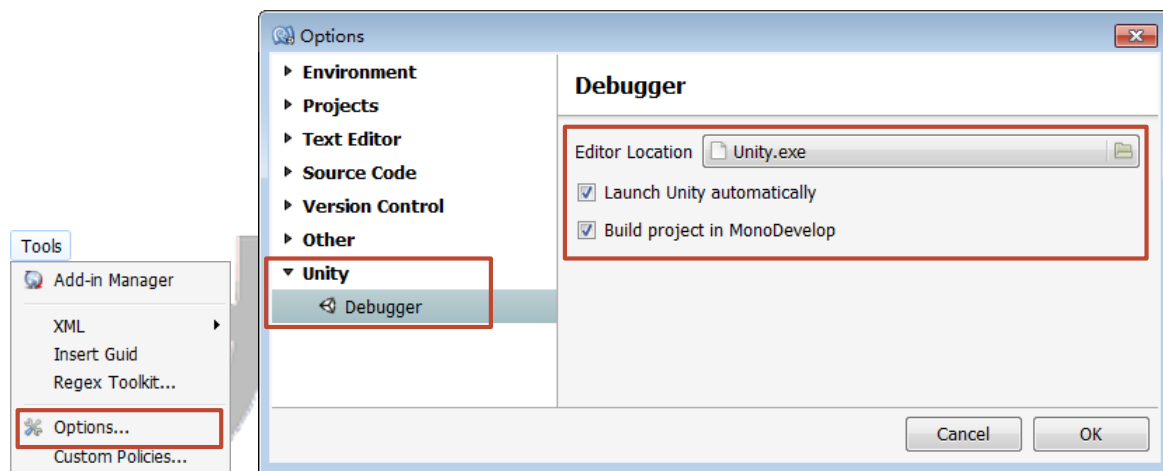


图 1-35 对 MonoDevelop 做的简要设置

在 Unity 的资源面板中新建一个 C#脚本文件，命名为 Count，然后在 MonoDevelop 中为其添加如图 1-36 所示的代码，并添加断点（在代码行号前鼠标单击即可），确保当前为 Debug 模式。

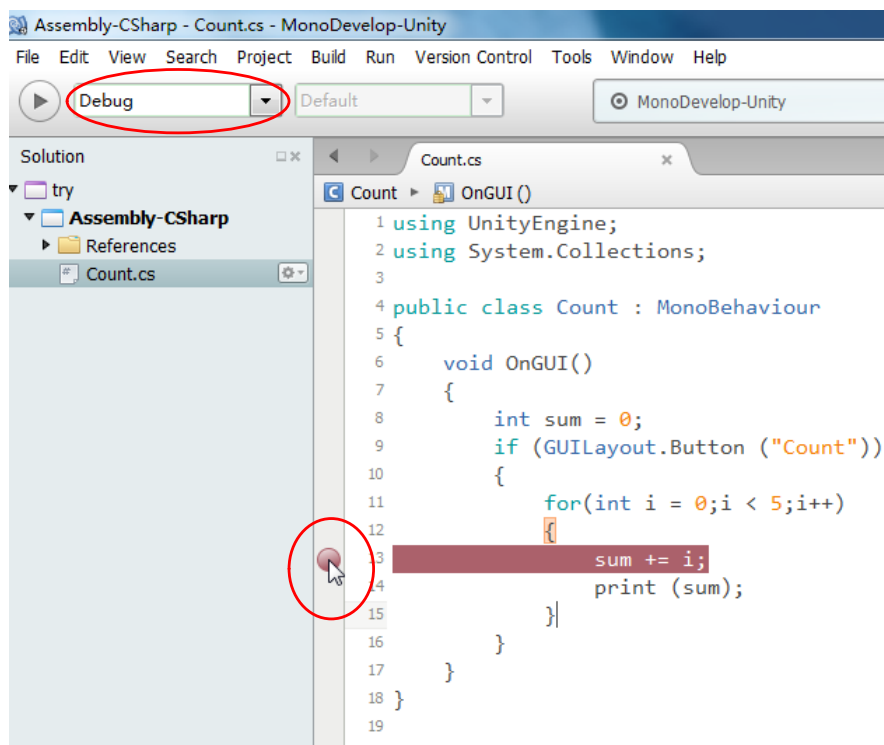


图 1-36 脚本中添加代码和断点


保存脚本代码后，将此脚本拖动到场景中的 Main Camera 对象上，成为后者的组件。然后，关闭 Unity 编辑器。再在 MonoDevelop 中，按下快捷键 F5（开始调试），会自动重新启动 Unity 编辑器。打开这个脚本所在的游戏项目后，单击工具栏上的  按钮，运行游戏。单击游戏视图上的 Count 按钮，即可进入到代码中的断点位置（黄色的箭头指到断点的位置），如图 1-37 所示。



图 1-37 运行游戏，进入断点的位置

而各个变量的值会显示在 Local 窗口中，如图 1-38 所示。

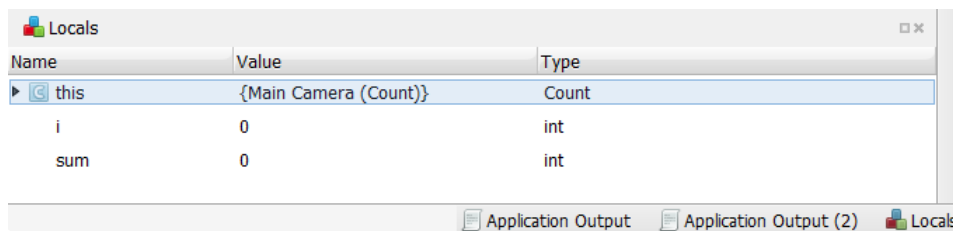


图 1-38 调试代码时，在 Local 窗口中实时查看变量的值

可以选择 MonoDevelop 工具栏上的按钮，如图 1-39 所示，决定采用何种调试方式。



图 1-39 MonoDevelop 工具栏上的调试按钮

4 个按钮，从右到左依次是 Continue Execution、Step Over、Step Into 和 Step Out。

- ☐ Continue Execution: 继续运行游戏，直到遇到下一个断点，或者游戏结束。
  - ☐ Step Over: 按照代码的顺序，一行一行调试断点后面会依次运行的代码。
  - ☐ Step Into: 当箭头所指的代码有函数调用的时候，会进入这个函数继续调试。
  - ☐ Step Out: 当箭头所指的代码在某一函数内，会返回函数被调用的地方。
- 要终止调试可以按下 Ctrl+F5 快捷键。

## 1.6.2 调试方法二

使用上一小节介绍的调试方法，就意味着每次调试都需要先关闭 Unity 编辑器。如果在实际的游戏开发中，不方便关闭 Unity 编辑器的话，可以考虑使用本节介绍的第二种调试方法。

在 MonoDevelop 的窗口中，单击 Run|Attach to Process 命令，会弹出 Attach to Process 对话框。选中当前运行的 Unity 编辑器进程名，然后单击 Attach 按钮，即可在 Unity Editor 与 MonoDevelop 中建立连接，如图 1-40 所示。



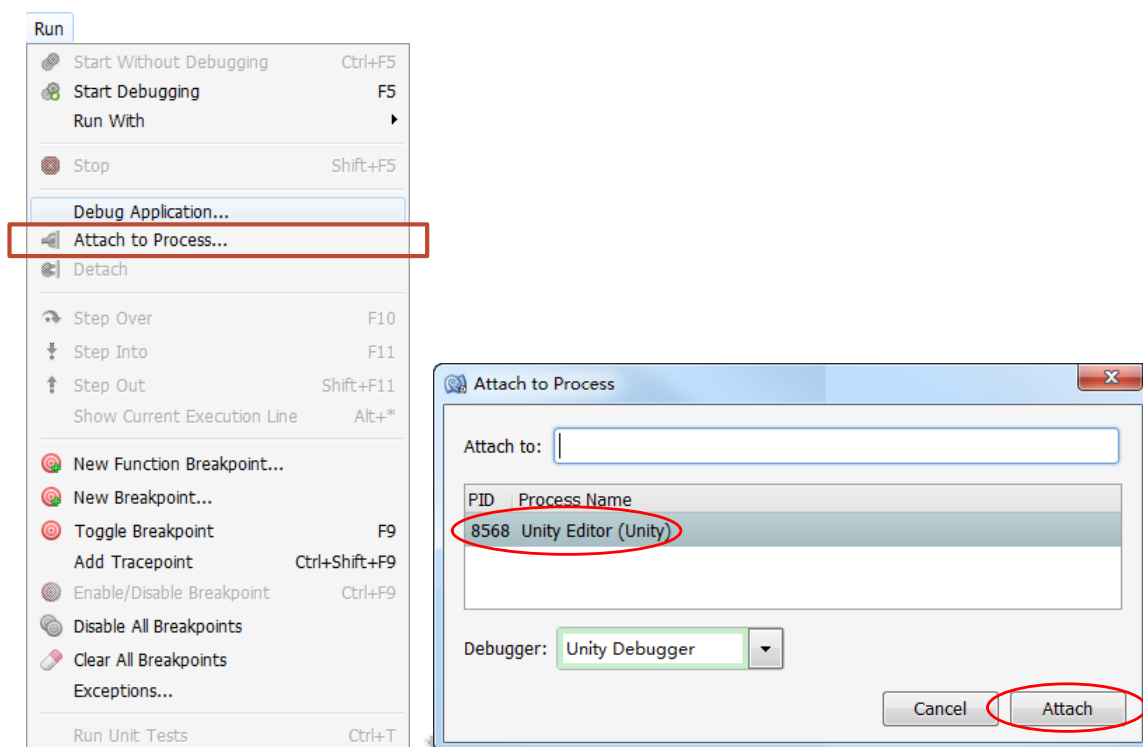


图 1-40 在 Unity Editor 与 MonoDevelop 间建立连接

建立了连接以后，就可以在 MonoDevelop 窗口中按下 F5，调试程序了。其余的步骤与上一小节的一致。