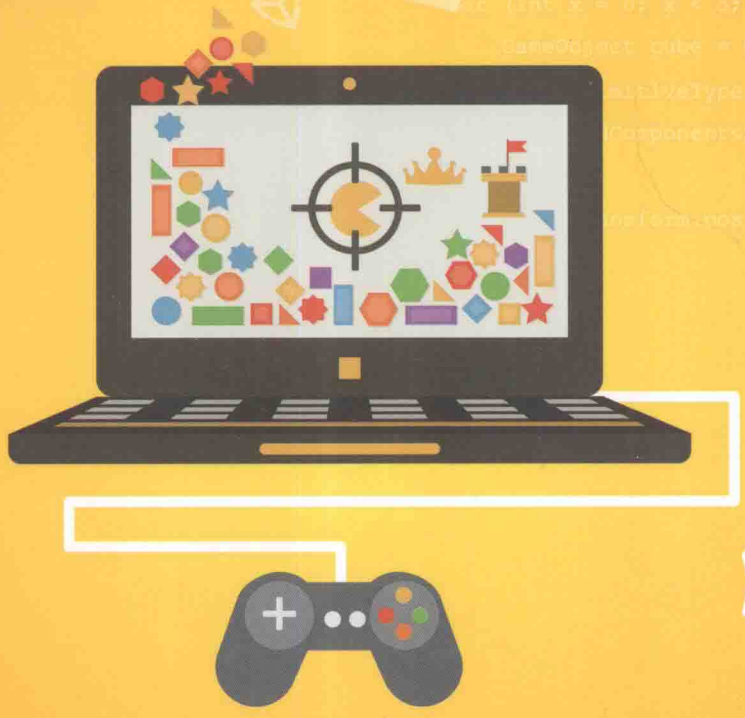
本书以Unity 3D的跨平台基础Mono，以及其游戏脚本语言C#为基础进行讲解。
全面系统地剖析了Unity 3D的跨平台原理以及游戏脚本开发的特点。

Broadview®
www.broadview.com.cn

Unity 3D 脚本编程

使用C#语言开发跨平台游戏

陈嘉栋 著



微软MVP
张善友
倾情作序

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Unity3D 脚本编程

使用C#语言开发跨平台游戏

陈嘉栋 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书以 Unity 3D 的跨平台基础 Mono，以及其游戏脚本语言 C#为基础进行讲解。全面系统地剖析了 Unity 3D 的跨平台原理以及游戏脚本开发的特点。

第 1 章主要介绍了 Unity 3D 引擎的历史以及编辑器的基本知识；第 2 章主要介绍了 Mono，以及 Unity 3D 利用 Mono 实现跨平台的原理，并且分析了 C#语言为什么更适合 Unity 3D 游戏开发的原因；第 3 章到第 10 章主要介绍了 Unity 3D 游戏脚本语言 C#在使用 Unity 3D 开发过程中的知识点，包括 Unity 3D 脚本的类型基础、数据结构，在 Unity 3D 脚本中使用泛型、使用委托和事件打造自己的消息系统、利用定制特性来拓展 Unity 3D 的编辑器、Unity 3D 协程背后的秘密——迭代器，以及可空类型和序列化在 Unity 3D 中使用的相关知识；第 11 章到第 14 章主要介绍了 Unity 3D 的资源管理，以及优化和编译的内容。

无论是初次接触 Unity 3D 脚本编程的新人，还是有一定经验的老手，相信都可以借本书来提高自己在 Unity 3D 方面的水平。

图书在版编目（CIP）数据

Unity 3D 脚本编程：使用 C#语言开发跨平台游戏 / 陈嘉栋著. —北京：电子工业出版社，2016.9
ISBN 978-7-121-29718-2

Ⅰ. ①U… Ⅱ. ①陈… Ⅲ. ①游戏程序－程序设计 Ⅳ. ①TP317.6

中国版本图书馆 CIP 数据核字(2016)第 196502 号

策划编辑：付　睿
责任编辑：徐津平
印　　刷：北京天宇星印刷厂
装　　订：北京天宇星印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×980　1/16　印张：25.25　字数：560 千字
版　　次：2016 年 9 月第 1 版
印　　次：2016 年 9 月第 1 次印刷
印　　数：3000 册　　定价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

推荐序

Unity 3D 是由两个具有巨大吸引力而极其令人愉悦的领域混合而成的——C#语言和游戏开发。Unity 团队设计 Unity 3D 将这两者有机地结合起来。

对于 C#语言的喜爱要回溯至 2000 年，当时微软向世界推出了新的语言 C#，不仅震惊了 Windows 领域，它同时也震惊了开源世界。GNOME 项目的领导者 Meguel de Icaza 就看到了 C#语言在桌面开发的前景，着手创建了开源的.NET 跨平台实现 Mono，如今 Mono 已经用于从嵌入式系统到服务器、工业控制、移动开发和游戏的所有方面。.NET 语言不仅确保了我们不再受限于某一种当下的语言，而且确保了我们可以继续重用之前使用 C 和 C++编写的现有代码，C#使我们和我们所处的世界更加高效。随着微软成立.NET 基金会，大力发展开源跨平台的.NET，同时 Unity 公司也是.NET 基金会成员，我们有理由相信使用 C#的 Unity 3D 平台也会发展得更好。

正如 Unity Technologies 的 CEO David Helgason 先生所说：“Unity 是一个用来构建游戏的工具箱，它整合了图像、音频、物理引擎、人机交互以及网络等技术。”Unity 3D 因为它的快速开发以及跨平台能力而为人所知。Unity 3D 的快速开发和跨平台能力正是来自于它对 Mono 平台和 C#语言的依赖，使用 C/C++来编写高性能要求的引擎代码，针对开发人员采用高级的 C#、UnityScript、Boo 语言作为游戏开发的脚本。

本书作者陈嘉栋带着激情投身于 Unity 游戏开发，他在 Unity 社区也非常活跃，他也通过博客写了大量和 Unity 3D 相关的文章，如今他将这些在社区上的贡献汇集成这样一个涉及 Unity 3D 跨平台原理分析、Unity 3D 和 Mono 的结合，以及在游戏脚本编程中使用 C#语言的作品。希望无论是初次接触 Unity 3D 脚本编程的新人，还是有一定经验的老手都能对 Unity 3D 了解得更全面深刻，对 C#语言在开发游戏脚本过程中的知识点掌握得更牢固，写出更高效

的代码。

使用 C#和 Unity 3D 构建游戏是一件极佳的事情。你能够使用一种强类型的、类型安全的、垃圾回收的、具有最热门 API 的语言来开发游戏，下面开始学习陈嘉栋创作的这本佳作吧！

微软 MVP　张善友

2016.7.13 书于 深圳

前 言

2005 年 6 月 6 日，在 WWDC 大会上，Unity 3D 的第一个版本正式推出。当时的 Unity 3D 还只能在 Mac OS X 平台上运行，经过 10 年的发展，到如今 Unity 3D 已经迎来了 5.0 的时代。Unity 3D 在这十年间继承并发扬了其一贯的优势，即拥有强大的编辑器，以及便利的跨平台能力和适合移动平台的 3D 开发能力。正是基于这些优势，Unity 3D 无疑为开发者节省了大量的时间和精力，使得整个开发流程变得更加高效而且便捷。而它本身也早已成为了全功能游戏引擎市场不可忽视的一股力量。

正如 Unity Technologies 的 CEO——David Helgason 先生所说：“Unity 是一个用来构建游戏的工具箱，它整合了图像、音频、物理引擎、人机交互以及网络等技术。”的确如此，Unity 3D 因为它的快速开发以及跨平台能力而为人所知。

其中 Unity 3D 编辑器的构建和拓展，以及引擎本身强大的跨平台能力，都多多少少借助了 Mono 以及 C#语言来实现，而 C#语言更是 Unity 3D 引擎最主要的游戏脚本语言，而采用脚本语言开发游戏逻辑无疑大大提高了使用 Unity 3D 进行开发的效率。

但目前国内并没有太多涉及 Unity 3D 跨平台原理分析、Unity 3D 和 Mono 的结合，以及在游戏脚本编程中使用 C#语言的资料。作者身边的很多朋友和网上认识的很多同行，由于没有一份系统剖析 Unity 3D 脚本编程的资料，而对这部分的内容没有形成一个全面的了解。

因此，本书的目标便是将 Unity 3D 和 Mono 的关系，以及使用 C#语言开发 Unity 3D 的游戏脚本的内容进行系统化地整理分析，本书不仅会分析 C#语言在 Unity 3D 脚本编程中的知识点（例如使用定制特性来拓展 Unity 3D 编辑器、使用委托事件机制打造自己的消息系统，以及 Unity 3D 中协程的实现细节等），还会带领各位读者走进更加底层的 CIL 代码层，了解 Unity 3D 借助 Mono 跨平台的原理，最后还会关注 Unity 3D 引擎本身的资源管理、项目优化以

及编译的内容。通过介绍最有代表性的两大移动平台——Android 平台和 iOS 平台，通过介绍两个平台上的项目编译进而使广大读者了解 JIT 编译方式以及 AOT 编译方式。希望能通过这些内容使国内的广大 Unity 3D 从业人员对 Unity 3D 了解得更全面深刻，对 C#语言在开发游戏脚本过程中的知识点掌握得更牢固，写出更高效的代码。

本书的作者长期关注 Unity 3D、Mono 以及 C#语言，并且在博客园、游戏蛮牛以及 InfoQ 网站以“慕容小匹夫”的笔名发表过多篇博客和文章，更是在 2015 年获得了微软最有价值专家（MVP）的称号。我在博客园写博客的期间，很荣幸地获得了电子工业出版社计算机分社付睿编辑的认可，联系之后确定了本书的选题以及主要的内容框架。在本书的写作过程中，我也得到了很多朋友的帮助和支持，在此一并感谢。

最后，还要感谢所有在我的博客浏览、评论、支持、批评、指正的朋友们，仅以本书献给所有读者，并衷心地希望和各位读者共同进步。

目 录

第 1 章 Hello Unity 3D 1

1.1 Unity 3D 游戏引擎进化史 1

1.2 Unity 3D 编辑器初印象 5

1.2.1 Project 视图 5

1.2.2 Inspector 视图 8

1.2.3 Hierarchy 视图 9

1.2.4 Game 视图 10

1.2.5 Scene 视图 12

1.2.6 绘图模式 14

1.2.7 渲染模式 16

1.2.8 场景视图控制 17

1.2.9 Effects 菜单和 Gizmos 菜单 18

1.3 Unity 3D 的组成 18

1.4 为何需要游戏脚本 20

1.5 本章总结 21

第 2 章 Mono 所搭建的脚本核心基础 22

2.1 Mono 是什么 22

2.1.1 Mono 的组成 22

2.1.2 Mono 运行时 23

2.2 Mono 如何扮演脚本的角色 24
2.2.1 Mono 和脚本 24
2.2.2 Mono 运行时的嵌入 26
2.3 Unity 3D 为何能跨平台？聊聊 CIL 38
2.3.1 Unity 3D 为何能跨平台 38
2.3.2 CIL 是什么 40
2.3.3 Unity 3D 如何使用 CIL 跨平台 44
2.4 脚本的选择，C# 或 JavaScript 48
2.4.1 最熟悉的陌生人——UnityScript 48
2.4.2 UnityScript 与 JavaScript 51
2.4.3 C#与 UnityScript 55
2.5 本章总结 57

第 3 章 Unity 3D 脚本语言的类型系统 58

3.1 C#的类型系统 58
3.2 值类型和引用类型 65
3.3 Unity 3D 脚本语言中的引用类型 73
3.4 Unity 3D 游戏脚本中的值类型 90
3.4.1 Vector2、Vector3 以及 Vector4 90
3.4.2 其他常见的值类型 94
3.5 装箱和拆箱 95
3.6 本章总结 98

第 4 章 Unity 3D 中常用的数据结构 99

4.1 Array 数组 100
4.2 ArrayList 数组 101
4.3 List<T>数组 102
4.4 C#中的链表——LinkedList<T> 103
4.5 队列（Queue<T>）和栈（Stack<T>） 107
4.6 Hash Table（哈希表）和 Dictionary<K,T>（字典） 112
4.7 本章总结 120

第 5 章 在 Unity 3D 中使用泛型 121
5.1 为什么需要泛型机制 121
5.2 Unity 3D 中常见的泛型 124
5.3 泛型机制的基础 127
5.3.1 泛型类型和类型参数 128
5.3.2 泛型类型和继承 131
5.3.3 泛型接口和泛型委托 131
5.3.4 泛型方法 136
5.4 泛型中的类型约束和类型推断 139
5.4.1 泛型中的类型约束 139
5.4.2 类型推断 144
5.5 本章总结 146

第 6 章 在 Unity 3D 中使用委托 149
6.1 向 Unity 3D 中的 SendMessage 和 BroadcastMessage 说拜拜 150
6.2 认识回调函数机制——委托 151
6.3 委托是如何实现的 154
6.4 委托是如何调用多个方法的 160
6.5 用事件（Event）实现消息系统 164
6.6 事件是如何工作的 169
6.7 定义事件的观察者，实现观察者模式 172
6.8 委托的简化语法 177
6.8.1 不必构造委托对象 177
6.8.2 匿名方法 178
6.8.3 Lambda 表达式 196
6.9 本章总结 201

第 7 章 Unity 3D 中的定制特性 202
7.1 初识特性——Attribute 202
7.1.1 DllImport 特性 203
7.1.2 Serializable 特性 205

7.1.3 定制特性到底是谁 207
7.2 Unity 3D 中提供的常用定制特性 208
7.3 定义自己的定制特性类 213
7.4 检测定制特性 216
7.5 亲手拓展 Unity 3D 的编辑器 217
7.6 本章总结 227

第 8 章 Unity 3D 协程背后的迭代器 228

8.1 初识 Unity 3D 中的协程 228
8.1.1 使用 StartCoroutine 方法开启协程 229
8.1.2 使用 StopCoroutine 方法停止一个协程 233
8.2 使用协程实现延时效果 234
8.3 Unity 3D 协程背后的秘密——迭代器 238
8.3.1 你好，迭代器 238
8.3.2 原来是状态机 242
8.3.3. 状态管理 248
8.4 WWW 和协程 253
8.5 Unity 3D 协程代码实例 257
8.6 本章总结 259

第 9 章 在 Unity 3D 中使用可空型 260

9.1 如果没有值 260
9.2 表示空值的一些方案 261
9.2.1 使用魔值 261
9.2.2 使用标志位 261
9.2.3 借助引用类型来表示值类型的空值 265
9.3 使用可空值类型 267
9.4 可空值类型的简化语法 272
9.5 可空值类型的装箱和拆箱 278
9.6 本章总结 280

第 10 章 从序列化和反序列化看 Unity 3D 的存储机制 281
10.1 初识序列化和反序列化 281
10.2 控制类型的序列化和反序列化 290
10.2.1 如何使类型可以序列化 290
10.2.2 如何选择序列化的字段和控制反序列化的流程 292
10.2.3 序列化、反序列化中流的上下文介绍及应用 296
10.3 Unity 3D 中的序列化和反序列化 299
10.3.1 Unity 3D 的序列化概览 299
10.3.2 对 Unity 3D 游戏脚本进行序列化的注意事项 302
10.3.3 如何利用 Unity 3D 提供的序列化器对自定义类型进行序列化 305
10.4 Prefab 和实例化之谜——序列化和反序列化的过程 309
10.4.1 认识预制体 Prefab 309
10.4.2 实例化一个游戏对象 311
10.4.3 序列化和反序列化之谜 314
10.5 本章总结 317

第 11 章 移动平台动态读取外部文件 318
11.1 假如我想在编辑器里动态读取文件 318
11.2 移动平台的资源路径问题 320
11.3 移动平台读取外部文件的方法 323
11.4 使用 Resources 类加载资源 330
11.5 使用 WWW 类加载资源 332
11.5.1 利用 WWW 类的构造函数实现资源下载 332
11.5.2 利用 WWW.LoadFromCacheOrDownload 方法实现资源下载 333
11.5.3 利用 WWWForm 类实现 POST 请求 335
11.6 本章总结 335

第 12 章 在 Unity 3D 中使用 AssetBundle 336
12.1 初识 AssetBundle 336
12.2 使用 AssetBundle 的工作流程 337

12.2.1 开发阶段 337
12.2.2 运行阶段 340
12.3 如何使用本地磁盘中的 AssetBundle 文件 344
12.4 AssetBundle 文件的平台兼容性 345
12.5 AssetBundle 如何识别资源 345
12.6 本章总结 346

第 13 章 Unity 3D 优化 347

13.1 看看 Unity 3D 优化需要从哪里着手 347
13.2 CPU 方面的优化 348
13.2.1 对 DrawCall 的优化 348
13.2.2 对物理组件的优化 354
13.2.3 处理内存，却让 CPU 受伤的 GC 355
13.2.4 对代码质量的优化 356
13.3 对 GPU 的优化 357
13.3.1 减少绘制的数目 358
13.3.2 优化显存带宽 358
13.4 内存的优化 359
13.4.1 Unity 3D 的内部内存 359
13.4.2 Mono 的托管内存 360
13.5 本章总结 363

第 14 章 Unity 3D 的脚本编译 365

14.1 Unity 3D 脚本编译流程概览 365
14.2 JIT 即时编译 368
14.2.1 使用编译器将游戏脚本编译为托管模块 368
14.2.2 托管模块和程序集 369
14.2.3 使用 JIT 编译执行程序集的代码 370
14.2.4 使用 JIT 即时编译的优势 371
14.3 AOT 提前编译 372
14.3.1 在 Unity 3D 中使用 AOT 编译 372

14.3.2 iOS 平台和 Full-AOT 编译 373
14.3.3 AOT 编译的优势 374
14.4 谁偷了我的热更新？Mono、JIT 还是 iOS 374
14.4.1 从一个常见的报错说起 375
14.4.2 美丽的 JIT 377
14.4.3 模拟 JIT 的过程 378
14.4.4 iOS 平台的自我保护 381
14.5 Unity 3D 项目的编译与发布 382
14.5.1 选择游戏场景和目标平台 382
14.5.2 Unity 3D 发布项目的内部过程 384
14.5.3 Unity 3D 部署到 Android 平台 384
14.5.4 Unity 3D 部署到 iOS 平台 386
14.6 本章总结 389

第 1 章
Hello Unity 3D

本章会带领大家走进 Unity 3D 游戏引擎的发展历史，并介绍 Unity 3D 引擎的基本结构，最后引出搭建了 Unity 3D 引擎脚本核心基础的 Mono 平台。

1.1 Unity 3D 游戏引擎进化史

正如 Unity Technologies 的 CEO——David Helgason 先生所说：“Unity 是一个用来构建游戏的工具箱，它整合了图像、音频、物理引擎、人机交互以及网络等技术。”的确如此，Unity 3D 因为它的快速开发，以及跨平台能力而为人所知。那么它究竟是如何“横空出世”的呢？

时间回溯到 2002 年 5 月 21 日下午 1 点 47 分。一个叫作 Nicholas Francis 的丹麦程序员在网上发出了一个寻找合作伙伴的帖子，帖子的内容就是协助他，并为他的游戏引擎共同开发一套 Shader 系统（Shader 即着色器，是一个能够针对 3D 对象进行操作，并被 GPU 所执行的程序。通过这些程序，程序员就能够获得绝大部分想要的 3D 图形效果）。不久之后，有一位叫作 Joachim Ante 的程序员响应了这个帖子，并决定和 Nicholas Francis 共同开发这套 Shader 系统。而作为最初的第三位开发者，也是后来成为 CEO 的 David Helgason 先生，在听说了这个项目之后，也决定加入这个项目。

两年之后，他们成立了一个叫作 Over the Edge Entertainment（OTEE）的公司，David Helgason 成为 CEO。他们当时决定开发一种独立开发者也有能力使用的游戏引擎。

而日后 Unity 3D 游戏引擎之所以获得成功的一大原因，就是得益于对无力承担游戏引擎高额许可费用的独立开发者的支持。

又经过两年没日没夜的工作，Unity 3D 游戏引擎最初的版本已经初具雏形，如图 1-1 所示为早期 Unity 3D 版本的截图。但是考虑到游戏开发者在没有看到基于 Unity 3D 游戏引擎开发的成功案例之前，很难说服他们使用 Unity 3D。因此，Unity 3D 团队认识到，必须要使用他们的新引擎开发一套完整的商业游戏。这不仅可以用来检测和证明他们引擎的能力，同时开发出的这款商业游戏，也可以补贴后续开发的费用。

就这样，Unity 3D 开发团队使用他们的新引擎，花费了 5 个月的时间开发了一款商业游戏——《GooBall》，如图 1-2 所示。

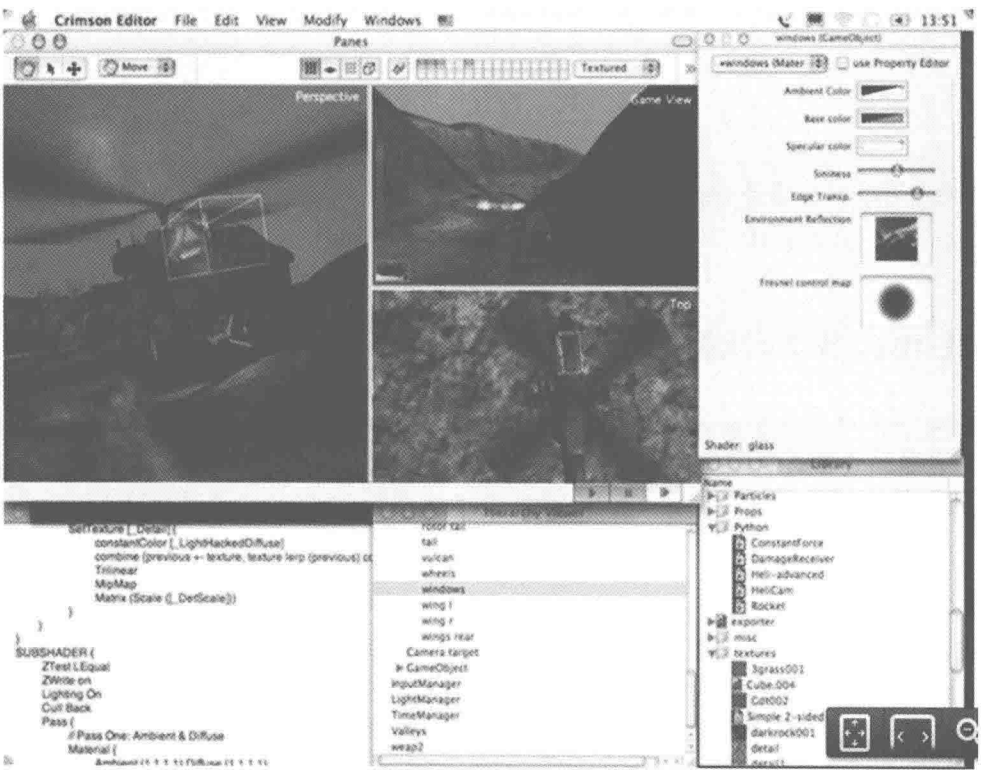


图 1-1 早期 Unity 3D 版本（version 0.2b）的截图

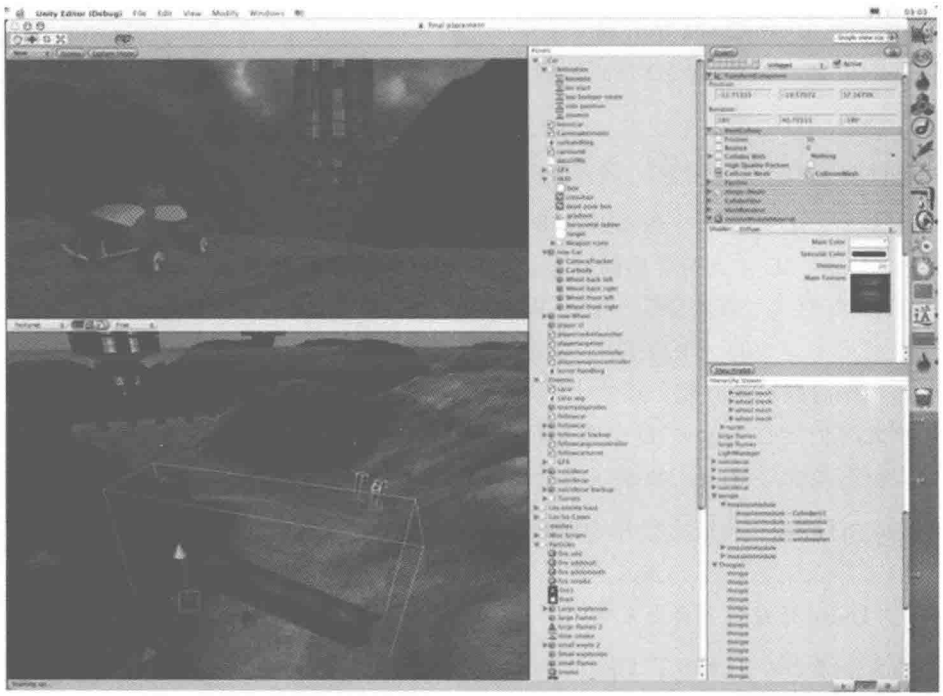


图 1-1　早期 Unity 3D 版本（version 0.2b）的截图（续）

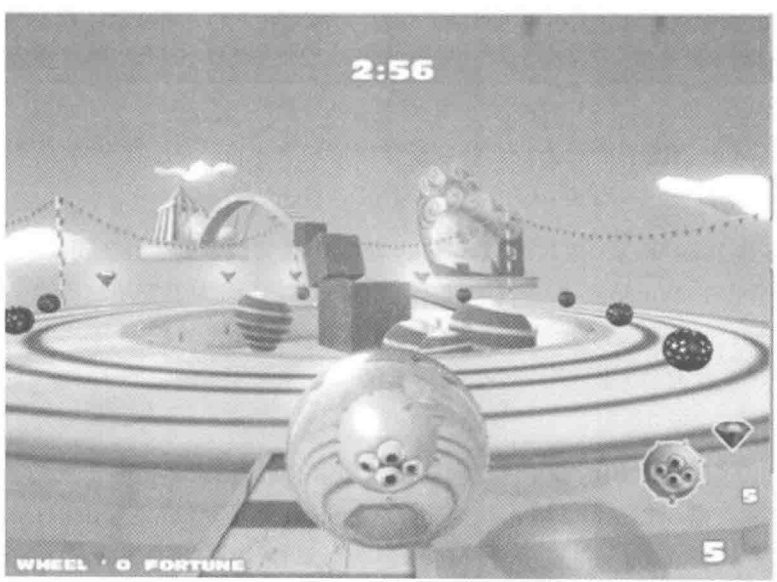


图 1-2　GooBall 截图