前　言

21 世纪是计算机极其普及和迅猛发展的世纪，人们在使用计算机作为多媒体娱乐工具的时候，不可避免地接触到计算机游戏，尤其是网络游戏。作为第九艺术，专家预测 2008 年全世界计算机游戏产值的总产值将达到 350 亿美元！计算机游戏已经成为全球商业的一个非常重要的分支，并且还在迅猛地发展前进着。在我国，据艾瑞调查报告数据，网络游戏市场规模在 2007 年达到 128 亿元，游戏收入比 2006 年增长了 67%.并且，预计游戏市场将以平均每年 20%的速度增长，到 2011 年将达到 400 亿。政府也开始制定相关政策，对游戏产业逐渐重视和加大投入力度。一些高校也开始开设游戏开发专业。而且，越来越多的热血青年投入到游戏开发领域中来，无论是专业的还是业余的，他们都有着属于自己的梦想，并为之努力着。

提到游戏开发，可能还有很多人认为要使用 C++语言结合 OpenGL 或者是 DirectX 进行程序设计。没错，这的确是个不错的方法。但同时也会对程序员要求较高。如今，业界公认的最流行的、效率最高的方法是使用游戏引擎进行游戏开发。游戏引擎为我们提供了一个平台，我们在这个平台上进行我们自己工作而不用考虑这个平台和外部如何交互的。举一个可能不太恰当的例子：我们需要撰写一个报告，然后打印几份出来交给其他人传阅。这个行为可以描述为写报告——游戏编程，打印——游戏发布，传阅——玩家玩。虽然过程很简单，但是存在一个问题，用什么来写这个报告？其实您已经知道答案了，word 啊！或者其它您喜欢的文本编辑软件。这个时候，我们就可以认为这个 word 是一个引擎，您可以直接在上面进行撰写，而不用考虑字符是如何通过键盘敲打进入到您的文本中的，还有你所使用的公式、表格、图片等都是怎么放入你的文本的，所有这些都不用您去操心。您所需要做的仅仅是使用 word 给我们提供的强大功能即可实现它们。打印，电脑和打印机怎么交流的？我们不关心，我们只要按那个打印机的小图标就 OK 了。是的，我们不必为了做这样的工作，而使用什么 C++语言编写一个 word 软件，然后再在它的基础上撰写我们的文档。现在，我想您应该理解引擎的作用了吧。这就是游戏引擎的魅力所在。

本书学习使用 Torque 游戏引擎。Torque 游戏引擎是加拿大 Garagegames 公司推出的一款非常出色的商业游戏引擎之一。国内越来越的多游戏公司和个人已经开始使用它来开发自己的游戏。然而同时，国内介绍 Torque 的书却非常少，目前，仅有清华大学出版社出版的两本加拿大人芬尼撰写的《3D 游戏开发大全》和《3D 游戏开发大全（高级篇）》。这两本书很棒，本人从中受益菲浅，并且在本书中借鉴了部分内容，希望大家都能阅读一下，同时向芬尼先生致以崇高的敬意。本书是作者根据自己多年学习研究 Torque 的经验，专门为初学者撰写的一本入门级的教材，希望读者阅读之后能够增加对 Torque 游戏引擎的了解，逐步掌握使用 Torque 游戏引擎进行游戏开发的原理和方法，进而使用它制作出属于自己的游戏。

第一章 初识 Torque

近年来，随着计算机硬件和网络条件飞速发展，电脑游戏尤其是网络游戏的快速推进，以及广大人民群众娱乐方式的网络化，使得电脑游戏产业处于异常火爆的状态。加上国家政策的倾斜和扶植，越来越多的人员从事到游戏行业中来，其中许多精英成为各大公司炙手可热的人才，年薪几十万已经不再是梦想！

随着国内外民间游戏开发团队的迅速增多，专业的、业余的爱好人员已经表现出了相当高涨的游戏开发热情，因此，当务之急，是从哪里进入游戏这个行业。

我遇到许多人提出这样的问题，“ 我有一定的编程基础，非常喜欢游戏行业，想为国产游戏贡献自己的力量，但是我应该从哪里开始呢？ ”那么，现在我可以告诉您，选择 Torque！如果您获得了授权的 TorqueGameEngineSDK-1-5-2，还等什么，安装它吧。

注意：什么是 SDK？SDK 是 Software Development Kit 的缩写，中文意思就是“ 软件开发工具包 ”。这是一个覆盖面相当广泛的名词，可以这么说：辅助开发某一类软件的相关文档、范例和工具的集合都可以叫做“ SDK ”。在这里，Torque 的 SDK 为我们提供了用于开发游戏所需要的大部分工具，引擎源代码、各种插件、以及可以作为起点的非常棒的游戏 Demo。

1.1 安装 Torque SDK

本书以 TGE1.5.2 版本进行介绍。

首先，请安装 TorqueGameEngineSDK-1-5-2，这是在本书编写时候的最新版本。如图 1.1 所示：

图 1.1　Torque 引擎的安装欢迎界面

根据提示，点击 Next，直到进入路径选择界面，如图 1.2 所示：

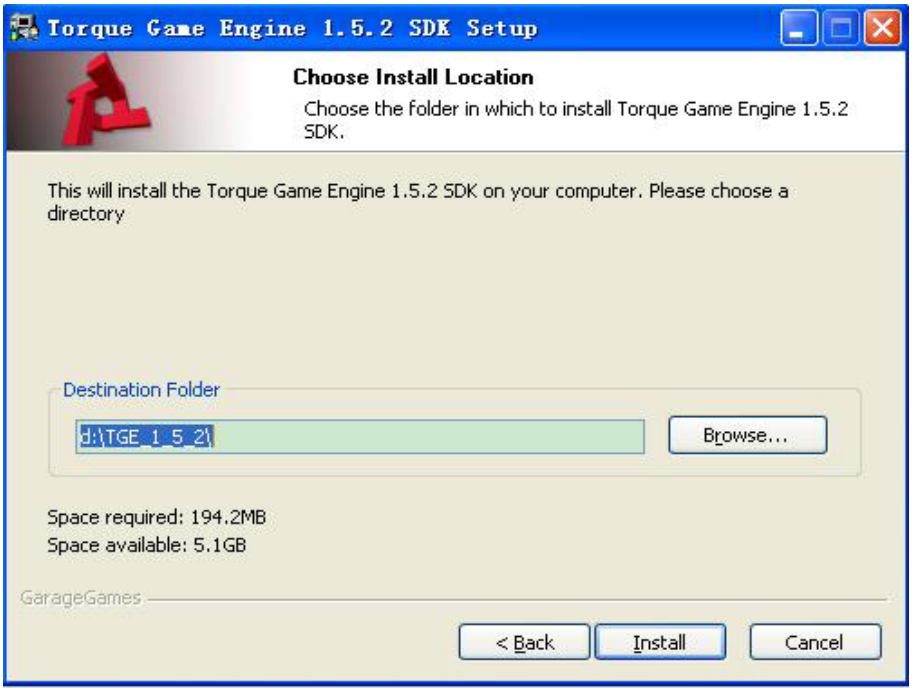


图 1.2　Torque 引擎安装的选择路径界面

在 Destination Folder 中输入你想要安装的文件夹目录，也可以点击右边的 Browse 按钮进行选择。我们推荐使用逻辑盘的根目录下，如 d、e 盘等，尽量避免安装目录中出现中文文件夹。然后点击 Install 即可开始安装 TGE1-5-2 游戏引擎，如图 1.3 所示。

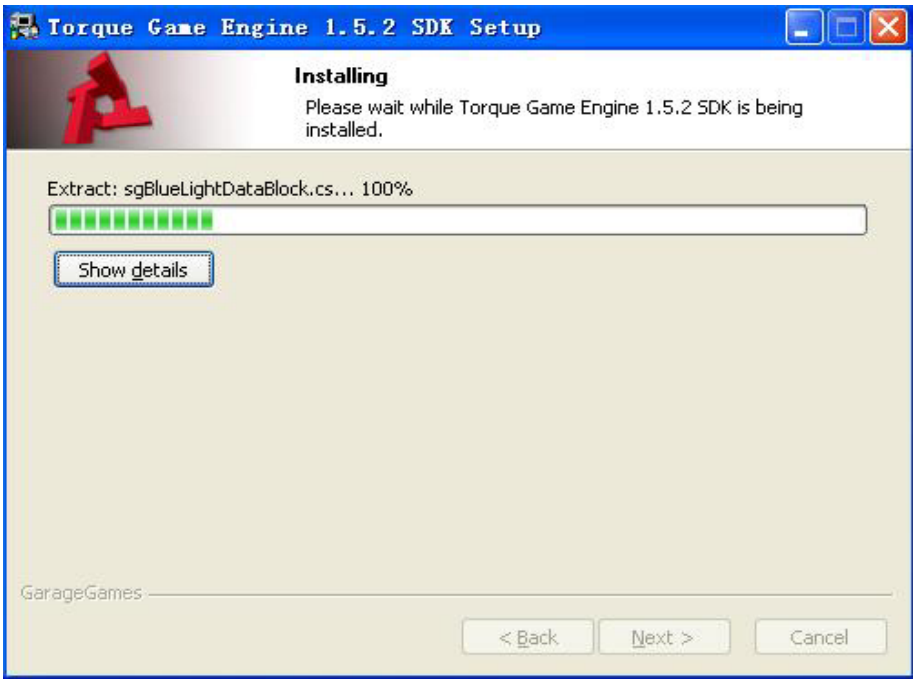


图 1.3　Torque 引擎安装过程

程序安装结束后，点击 Finish 结束安装，如图 1.4 所示。

图 1.4　Torque 引擎安装结束界面

1.2 Torque SDK 的内容

下面，我们需要了解一下安装好的程序里都有什么内容。同时了解一下在现阶段，我们进行自己的游戏制作过程中，哪些内容是基本的，必须使用的，那些

内容是高级的，是将来会使用到的。

首先，请找到您的引擎安装的位置，这里是 D:\TGE1_5_2\文件夹下。我们会看到如图 1.5 的内容：

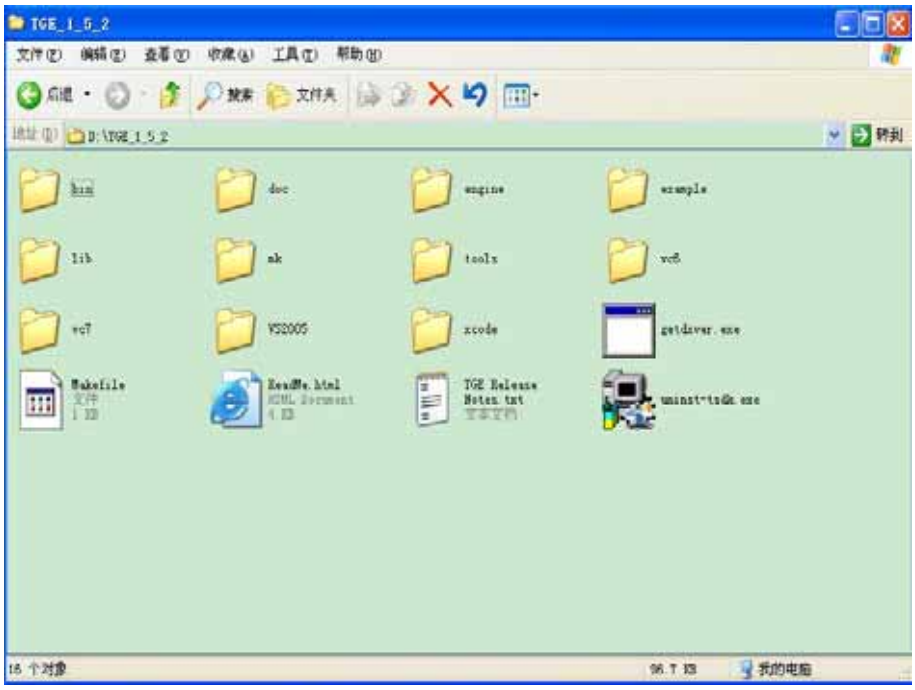


图 1.5 TGE_1_5_2 文件夹的内容

在我们初次接触 TGE 引擎时，我们最关心的是 example 文件夹中的内容，本书的所有工作全部在 example 文件夹中进行。其余内容都是与游戏开发相关的一些工具，如 engine 文件夹是所有引擎的 C++源代码，vc6、vc7 和 VS2005 文件夹中是针对 VC++6.0、VC++2003 和 VS2005 的工程文件，如果您想自己重新编译 Torque 引擎的源代码，就必须使用它们。这些内容已经超出了本书的学习范围，目前，我们将不会用到它们。

Example 文件夹中包含了 TGE 引擎的相关介绍，同时还包含了 2 个使用 TGE 游戏引擎制作的游戏，starter.fps 游戏和 starter.racing 游戏，分别是已经具备了一定的游戏因素的 FPS（第一人称射击）游戏和赛车游戏，我们今后的工作全部围绕这些例子展开。如果您现在就等不及想一睹它们的风采，可以通过“开始”→“Torque Game Engine 1.5 SDK”的方式找到它们，并且运行您想看到的游戏。

下面是 example 文件夹的内容，如图 1.6 所示。

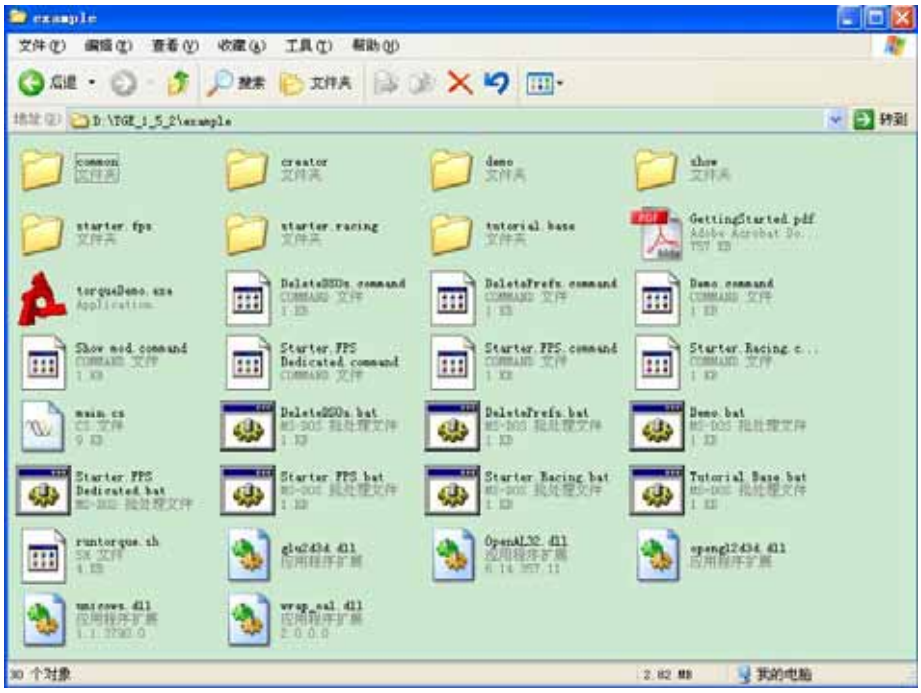


图 1.6　example 文件夹的内容

首先，我们看到的是 common 文件夹（可能你的文件顺序跟我的不一样，但是一定会找到 common 文件夹），该文件夹里包含的是有关游戏的公共代码，这些代码对于不同游戏类型和变体在本质上是相同的，其中包含所有游戏运行需要的最基本的脚本代码，如使游戏内嵌工具激活的脚本和数据。公共服务器功能和实现工具。其结构内容如图 1.7 所示：

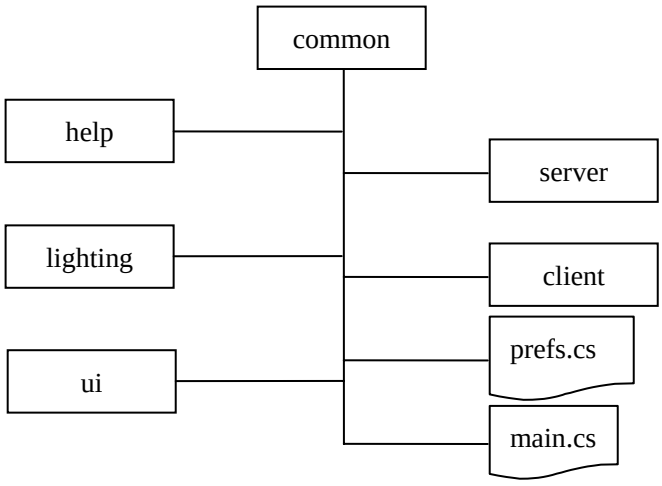


图 1.7 common 文件夹中的内容

各个文件夹包含的内容如表 1.1 所示：

表 1.1　common 文件夹中的内容

文件夹名	内容
client	客户端脚本包括客户端→服务器端通信
server	基本的服务器支持脚本，包括服务器一>客户端通信
lighting	材质的实时、动态光
help	客户端的帮助页面
ui	游戏内嵌工具使用的各种 GUI 控件和材质

creator 文件夹的内容如图 1.8 所示：

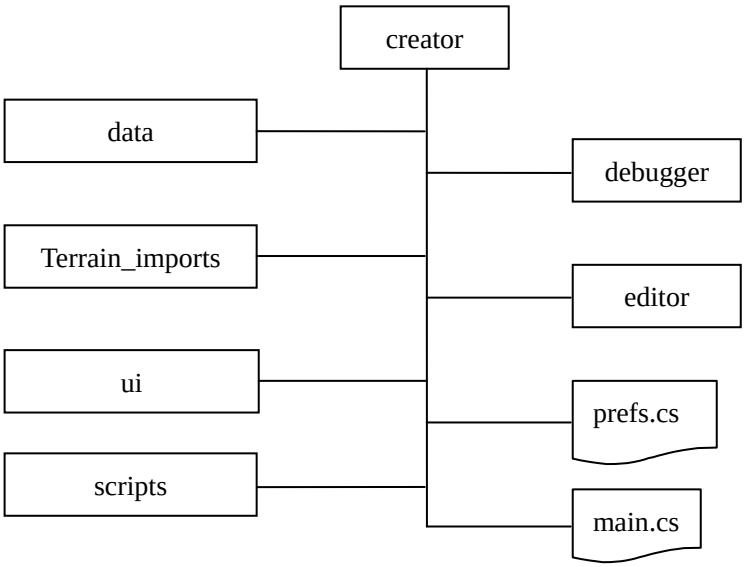


图 1.8　creator 文件夹中的内容

creator 文件夹中包含了游戏中的世界编辑器内容，它允许我们在游戏中直接进行编辑，如为游戏世界添加各种建筑物、树木等对象。

Show 文件夹中的内容，主要是在游戏编辑状态下，控制摄像头进行移动等操作。对于我们初学者来说，以后会涉及到学习这些文件夹中的内容，但基本不会涉及到对其中的内容进行修改。

Demo 文件夹中包含了游戏引擎的相关介绍，它采用浏览的方式向我们展示了该引擎的强大图像表现能力和游戏功能。如果您已经在这之前通过开始菜单的方式观看了 demo 的内容，相信您已经被它强大的图形表现能力震撼了！是的，我们就是要使用它来制作属于我们自动创作的游戏。如果您还没有观看到该演示

程序，现在是时候了，强烈建议您马上运行该演示程序。使用开始菜单的方式，选择“ 开始 ”→“ 所有程序 ”→“ Torque Game Engine 1.5.2 SDK ”→“ Torque Game Engine Demo ”。或者可以通过双击 torqueDemo.exe 文件进入，如果进入的是其它界面，请使用记事本或者你习惯的文本编辑软件打开 example 文件夹下的 main.cs 文件，将“ $defaultGame = " "; ”双引号中的内容改为 demo。然后运行 torqueDemo.exe 即可。还可以通过双击 Demo.bat 文件即可。您将会看到如图 1.9 的画面。

图 1.9　Torque Demo 的选择画面

点击“ Interactive Walkthrough ”就可以进到如图 1.10 所示的内容，请逐步点击“ > ”按钮，观看完全部的演示画面，真的很棒！

图 1.10　Welcome to Orc Town 演示画面

Starter.fps 和 starter.racing 文件夹中分别是使用 Torque 制作的具有可玩性的 FPS（First person shooting，第一人称射击）游戏和赛车游戏。将来，可以通过学习、参考和修改这两个游戏来构建属于我们自己的游戏。

Tutorial.base 文件夹是针对初学者学习使用的基本文件夹，我们开始的内容也完全从这里开始，通过对它的学习来掌握和使用 Torque。

TorqueDemo.exe 是编译好的可执行文件，双击它既可以进入我们的游戏。

GettingStarted.pdf 是 GarageGames 公司专门为 Torque 提供的入门级的开发文档，强烈建议在开始任何操作之前仔细阅读该文档，内容简单易懂，非常有用。当然，后面的章节我们将学详细习其中内容。

Main.cs 文件相当重要，当游戏运行时，引擎会首先调用该文件，进行一系列设置，但对于初学者来说，我们在很长一段时间对该文件的操作仅仅局限于其第一行代码（注释行出外），即“ $defaultGame = “ ”; ”，里面小引号中的内容是我们的游戏所在的文件夹，同时不要忘记后面的分号。

Demo.bat、starter.FPS.bat、Starter.Racing.bat、Tutorial.Base.bat 是运行相应程序的批处理文件，双击即可进入对应的游戏或程序。

Glu2d3d.dll、OpenAl32.dll、opengl2d3d.dll、unicows.dll、wrap_oal.dll 是游戏引擎需要的动态链接库文件，如声音（OpenAL32.dll）、图形（opengl2d3d.dll）

和编码（unicows.dll）等。

Console.log 文件是日志文档，用来保存游戏运行时候的相关信息，该文档可以直接用 windows 的记事本程序打开，作用很大，当游戏运行出错时，用来查看发生错误的信息是非常有帮助的。

Start.fps.Dedicated.bat 文件可以将你的电脑作为专用服务器，运行 Torque 自带的射击游戏。双击该图标运行程序，将会打开一个 Dos 命名窗口，然后进行一系列调用，最后，程序将停下来，如图 1.11 所示：

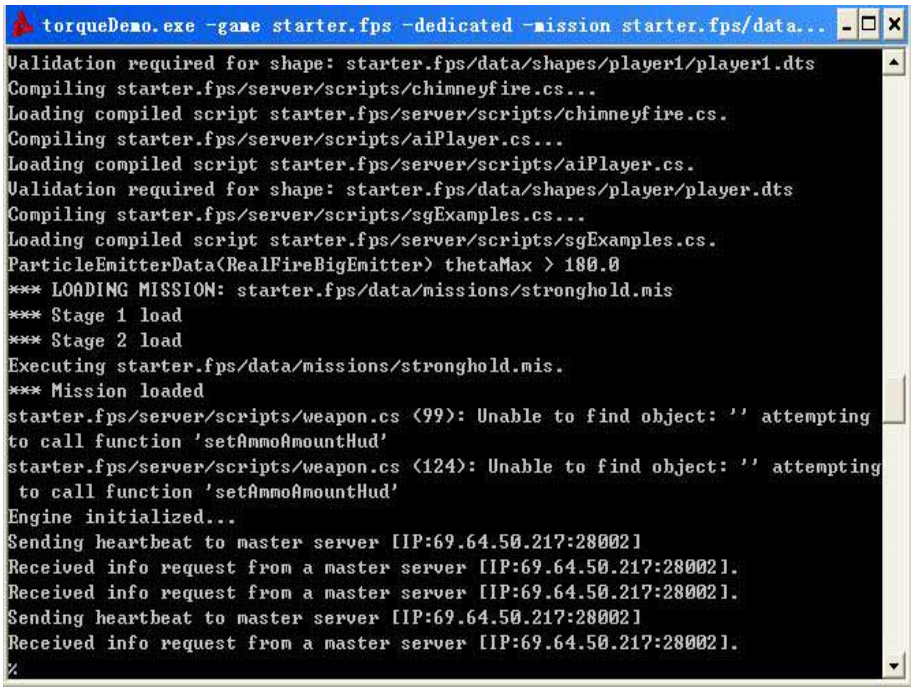


图 1.11　专用服务器运行的 Dos 命令窗口

这里，我们的主机将向 garagegames 设立的 master 服务器发送请求，该 master 的 IP 地址为 69.64.50.217，端口号为 280002。成功响应后，将会收到 master 的回复，表明你的主机已经作为专用服务器在 master 上登记成功。这样，其它游戏者可以通过查找互联网服务器而发现你提供的服务器，进而可以连接到你的主机上一起游戏。这个功能很容易理解，如果您曾经玩过任何一款的网络游戏，它们全部使用的是这样的方式：当您启动游戏客户端的时候，首先连接到的是一个主服务器，它为您提供了一个专用的服务器列表，常见的如网通、电信大区，然后是下面的很多小分区，这些分区名经常会采用相当有吸引力的名字展现给玩家，供玩家选择。您选择其中一个，就可以进入到虚拟的游戏世界中了。这样，您就了解为什么 Torque 会为我们提供 dedicated 服务器模式。

其余的 command 文件暂时与我们的学习没有太大关联。

1．3 使用 Torque 实现梦想

您已经看到，Torque Game Engine 功能强大，功能丰富，使用灵活而且易于控制。在本书中我们将要做的事情是从基础开始学习 Torque，充分使用 Torque SDK 为我们提供的各种资源的基础上，同时学习如何制作满足 Torque 的其它游戏素材，然后学习编写游戏控制脚本，最后，把这些素材联系成一个整体，完成一个属于您自己的游戏。

1．5 小结

恭喜您！您已经安装好了功能强大的 Torque 游戏引擎的 1.5.2 版本！

通过本章的介绍，相信您已经了解了 Torque 游戏引擎的相关内容，初步探索了游戏引擎的组成部分。相信随着对 Torque 引擎的深入学习，您将逐渐掌握其各部分的内容。最终，使用它作出一款属于您自己的游戏！下一章，我们将从 Getting Started 开始，使用 Torque 游戏引擎独立制作一个初具雏形的小游戏。

第二章 从 Getting Started 开始

好啦，现在准备工作已经差不多了，你是不是已经摩拳擦掌、迫不及待地要干一番了？是的，接下来让我们从 torque 的 GettingStarted.pdf 开始正式进入我们的游戏开发之旅。等一下，您说您已经浏览了该文档并且掌握了它的内容？OK，恭喜您！您具有非同寻常的超前意识和游戏开发潜质！继续努力，您必将获得成功！

对于还没有仔细学习该教程的人，现在正是时候。在本教程中，需要 2 个注意的地方，第一，当你看到类似“ Select File→Open Mission ”时，表明选择 File 菜单中的“ Open Mission ”选项。至于文件的位置，假设我们是按照前面章节安装 Torque 游戏引擎的位置，即 D:/TGE1_5_2/文件夹下。第二，本教程适用对象为 PC 平台用户，如果您使用的是苹果的操作系统，建议您使用两按键鼠标，同时，用 Option 键代替 Ctrl 键进行操作。此外，应用程序文件在 Mac 中被称为“ binaries（二进制）”文件，而这里称为“ executables（可执行）”文件。在开始学习之前，我们先了解一下游戏文件夹的组织结构要求。

2．1 游戏的组织结构

当使用 Torque 游戏引擎创建游戏时， Torque 有属于自己要求的文件夹结构模式，在此基础之上，您可以灵活使用各种自己喜欢的方式来组织它们。这里推荐使用 example 文件夹中的结构形式，如图 2.1 所示。

example 文件夹下的 main.cs 文件必须与 TorqueDemo.exe 文件放在一起，即游戏的根文件夹下。当然，example 在将来可以改成任意您喜欢的游戏的名字，比如 Fallout（辐射，相当经典的游戏）。

tutorial.base 文件夹下包括 main.cs 文件、client、data 和 server 文件夹。

Torque 设计的游戏结构非常直观、科学，其基本结构如图 2.1 所示：

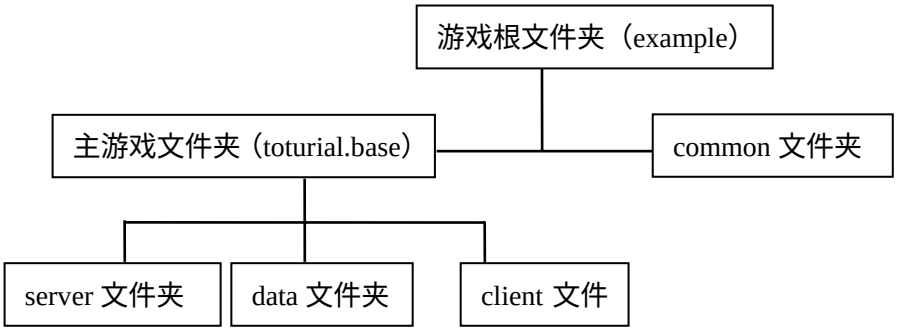


图 2.1　标准的游戏文件夹结构

对我们初学者，游戏根文件夹（example）下接触最多的是 main.cs 文件、conlose.log 文件和 TorqueDemo.exe 文件。main.cs 文件非常重要，游戏运行时首先要调用该文件。需要注意的是，在 torque 游戏文件夹树中还会出现好几个 main.cs 文件，为了便于区分，我们使用“根 main.cs”文件称呼它。conlose.log 文件是日志文件，记录了所有游戏运行时候的信息，通过它可以方便的查找游戏中的事件。直接使用记事本程序就可以查看它的内容。TorqueDemo.exe 是已经编译好的可执行文件，对于初学者，我们暂时不会进行引擎源代码的修改，也就不会重新编译该可执行文件。当然，你可以使用任何一个你喜欢的属于你自己游戏的名字来重新命名它，如 game.exe，通过“重命名”命令即可。注意，当启动 TGE 游戏时，该可执行文件会寻找和它位于同一级文件夹下的 main.cs 文件，即“根 main.cs”文件，因此，它必须与“根 main.cs”文件放在同一级文件夹下。

主游戏文件夹下存放的全部是跟我们游戏相关的内容。Server 文件夹下放置的是服务器端脚本代码。Data 文件夹下放置的是游戏中的所有资源，包括图片、声音、模型等。Client 文件夹下放置的是客户端脚本代码。

2.2 Torque 支持的文件类型

游戏是门艺术，它创造性地将其它多种艺术形式有机地融合在一起，为玩家提供了愉悦的游戏体验。因此，制作一款优秀的游戏，需要大量丰富的资源。同其它游戏引擎一样，Torque 也需要使用它支持的各种资源进行游戏制作，下面我们来了解一下 Torque 所支持的资源类型。

2.2.1 .cs 和.cs.dso 文件

TorqueScript（Torque 脚本）是 Torque 引擎中最具革命性的特性，它是一种语法类似 C++的面向对象程序设计语言，提供了对 Torque 引擎内机会所有组件的访问方法。而且它的功能非常强大，用户很少需要对引擎进行源代码级的修改，因为使用 Script 基本可以实现你的全部想法！

您可以使用任意习惯的文本编辑软件编写 Torque Script 代码，这里推荐使用 Codeweaver 编辑软件。当您的代码编写完成时，将其保存为.cs 文件类型即可。如果您的文本编辑软件没有该文件类型格式（这是很正常的，因为.cs 格式不是通用的文件格式，是 Torque 引擎自识别的格式），那么直接将文件名的后缀改为.cs 即可。如使用记事本文件编写的 test.txt 文件，改成 test.cs 即可，就这么简单。

每次运行 Torque 游戏时，Torque 会自动编译相应目录下的.cs 文件，.cs 文件经过编译成为.dso 文件，该文件为二进制文件并不可以反汇编（注，Torque 游戏的工作原理为运行游戏时候，引擎先编译同 Torque 可执行文件在同一文件夹下的 main.cs 文件，生成 main.cs.dso 文件，在继续编译其它.cs 文件，并覆盖已有的.dso 文件。如果没有发现.cs 文件，则直接运行已有的.dso 文件。这样，在游戏发布时候删掉.cs 源文件可保护自己的源代码）。

2.2.2 .gui 和.gui.dso 文件

.gui 文件是用来描述 GUI（Graphic User Interface，图形用户接口）的脚本文件。GUI 对任何一款游戏来说都是非常重要的，没有 GUI 的游戏将无法想象。而 GUI 制作的是否优秀直接影响到游戏玩家对该款游戏的第一印象，也会很大程度上影响着该款游戏的火爆程度。.gui 文件可以通过 Torque 自带的 GUI 编辑器进行可视化制作，这是相当直观而且简单的方法，推荐大家使用这种方式来制作.gui 文件。当然，您完全可以直接使用撰写代码的方式来完成该工作，只要您的代码符合规范。它们的关系就如同制作网页的时候，您是使用 Dreamweaver 这样简单易用的软件来制作，还是按照 HTML 的规范格式直接撰写代码的方式来制作。随你喜欢，呵呵。这里，也许您会觉得.gui 应该是同美工的关系更多一

些，但是，我仍然固执地认为，它应该属于程序的范畴。相信它不会造成我们之间沟通的隔阂。

同.cs 文件运行后会被编译成.cs.dso 文件一样，.gui 文件运行后会被编译成.gui.dso 文件。

2.2.3 DTS 格式文件

Torque 中使用的模型格式主要有 2 种，一种是表现普通对象的 DTS 格式，包括角色、车辆、岩石等物品；另一种是表现建筑物对象的 DIF 格式。

DTS（Dynamix Three Space）是 Torque 使用的用来显示 3D 对象的文件格式。所有可交互的 shapes（图形）如玩家角色、车辆和武器等，以及静态环境如植被、岩石等必须以 DTS 格式放入游戏场景。我们需要使用三维建模软件，如 3DS Max、Maya 和 MilkShape 等进行对象的多边形建模，然后使用 garagegames 公司提供的插件如 max2dtsExporter.dle 等将模型导出为.DTS 格式，这样才可以在我们的游戏中使用。如右图所示，为游戏自带的兽人模型。

2.2.4 DIF 格式文件

Torque 中使用术语 Interior（内景）来描述游戏中的建筑物。很遗憾，我不很清楚为什么使用该术语，而不是我们更加熟悉的 Building（建筑物）。也许它想表明的是可以实时走进里面的对象。没关系，这仅仅是术语上的问题，不会对我们的工作造成任何麻烦。

DIF（Dynamic Interior Format）格式是由 map2dif 插件生成的一种二进制文件格式，它提供了有关碰撞检测、光照检测和二叉树空间算法。该文件的源文件是.map 文件，可是使用 3DWS、Hammer 等工具进行建模，这里推荐使用 garagegames 公司推出的专门工具 Constructor，目前是 1.0.3 版本。

2.2.5 材质文件

Torque 支持多种位图文件类型：PNG、JPEG、GIF、BMP、以及用户自定

义的 BM8 格式，这是一种用于最小化纹理内存开销的颜色为 8 位的纹理格式。通常我们只使用 PNG 和 JPEG 格式，使用 photoshop 等图像编辑软件制作。

2.2.6 音乐和音效文件

Torque 支持使用.wav 和.ogg 格式的声音文件。

.wav 格式是常用的音频文件格式。

Ogg 是一种先进的有损的音频压缩技术，正式名称是 Ogg Vorbis，是一种免费的开源音频格式。OGG 编码格式远比 90 年代开发成功的 MP3 先进，它可以在相对较低的数据速率下实现比 MP3 更好的音质。此外，Ogg Vorbis 支持 VBR（可变比特率）和 ABR（平均比特率）两种编码方式， Ogg 还具有比特率缩放功能，可以不用重新编码便可调节文件的比特率。 OGG 格式可以对所有声道进行编码，支持多声道模式，而不像 MP3 只能编码双声道。多声道音乐会带来更多临场感，欣赏电影和交响乐时更有优势，这场革命性的变化是 MP3 无法支持的。在而且未来人们对音质要求不断提高， Ogg 的优势将更加明显。

2. 3 Getting Started

现在，让我们开始学习 GettingStarted.pdf 文档的全部内容。经过对该文档的详细学习，会使您对 Torque Game Engine（ 游戏引擎）有一个较全面的了解，从而知道该如何使用 Torque 游戏引擎来制作属于您自己的游戏。右图为 Torque Game Engine 的 Logo 标志。

2.3.1 起始页面

在 example 文件夹下，现在您应该非常熟悉该文件夹的内容了，教程中会使用 tutorial.base 文件夹中的所有文件。为了防止原始文件由于误操作被破坏，强烈建议您现在将 tutorial.base 文件夹进行备份。备份文件夹仍然放在 example 文件夹下，将其重命名为“ GameOne ”，用来表示我们的第一个游戏的文件夹。注意，再次强烈建议您进行任何操作前备份已有文件夹和文件，养成良好的习惯会是您的工作事半功倍。

接下来，我们要做的第一件事情，是要把默认的游戏文件夹指定为 GameOne 文件夹，还记得如何操作么？哈哈，我已经看到您露出会心的微笑了。对，根 main.cs 文件的第一行代码，就是它！将“ $defaultGame = “tutorial.base”; ”改为“ $defaultGame = “GameOne”; ”，即用 GameOne 取代 tutorial.base，然后保存文件，就这么简单 !需要注意的是，defaultGame 是区分大小写的，所以请确保 Game 和 One 都是以大写字母开头。

接下来，运行 torqueDemo 可执行文件，将会弹出一个游戏进入的窗口，如图 2.2 所示：

图 2.2　GameOne 的起始界面

屏幕上方的一共有 10 个图标按钮，这里请允许我花一点点的时间介绍它们的作用，如表 2.1 所示：

表 2.1　GameOne 起始界面图标说明

图 标	功 能
 GUI EDITOR	进入 GUI（图形用户接口）编辑器界面
 WORLD EDITOR	进入世界编辑器界面
 CONSOLE	打开 console（控制台）界面

NEWS	获得关于 Torque 的最新消息
DOCS	打开 Torque 的相关文档
FORUMS	链接到 Torque 的社区论坛
TDN	连接到 Torque Developer Network
TUTORIAL	打开 GettingStarted.pdf 文档
OPTIONS	打开选项界面
EXIT	退出 Torque

2.3.2 进入游戏世界

在弹出的窗口上部找到“ world Editor ”图标（按钮），单击一下，在一系列文件加载后（这个速度相当快），进入了我们的第一个游戏世界，如下图 2.3 所示，注意，目前处于世界编辑器状态下。

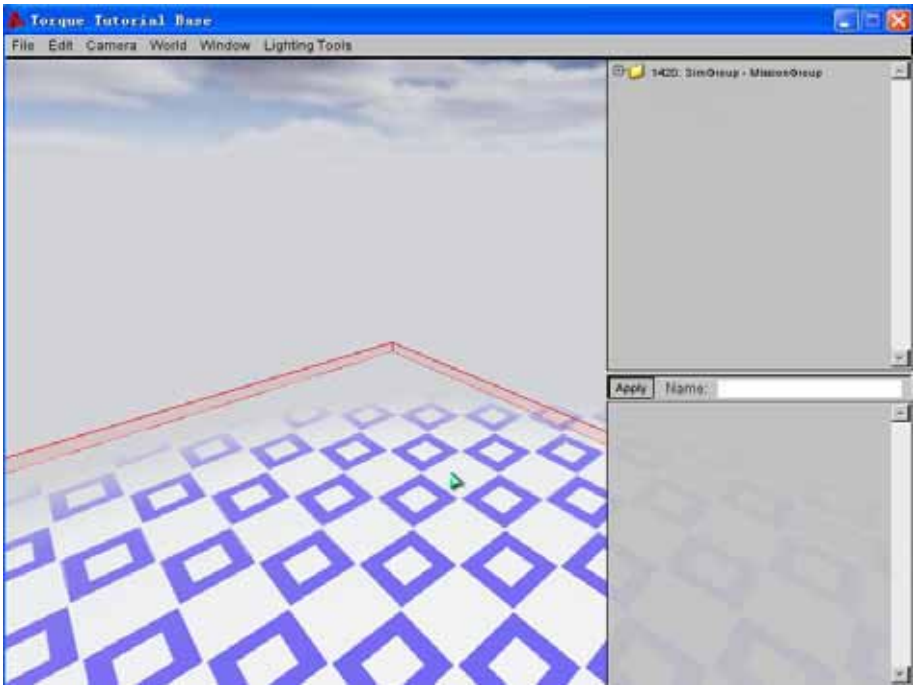


图 2.3　GameOne 的世界编辑器界面

目前，我们的游戏世界空无一物，我们将通过不断地添加物品对象来丰富它。

地面是令人讨厌的蓝、白棋盘格材质，以后我们会换成我们需要的如草地、沙地等令人满意的材质。

好，在开始丰富我们的游戏世界前，先花点时间熟悉一下 Torque 里的移动。通过 Camera→Toggle Camera（快捷键 Alt+C）转换为第一人称游戏视角。W、S、A、D 是我们相当熟悉的前、后、左、右移动的操作按键，space（空格）键为跳跃按键。按住鼠标右键左右移动鼠标可以环视游戏世界。按 Tab 键，可以切换到第三人称视角（追尾视角），好像很多的朋友更喜欢这个视角。

在编辑游戏的时候，我们更希望以“ 上帝 ”视角来观察游戏世界，便于添加物品，因此，再次使用 Camera→Toggle Camera（快捷键 Alt+C）转换到“ 上帝 ”视角，该状态下可以快速在游戏世界里飞翔。注意：天空中漂浮的灰色正方体是游戏玩家的出生点。为什么在天上？天知道。

接下来，让我们尝试着改变一下游戏世界里一马平川的地形吧。首先，确保出于“ 上帝 ”视角下，选择 Window→Terrain Editor，这个我想没有任何问题吧？现在在屏幕中移动鼠标，您会发现鼠标周围有一群绿色（或者红绿相间）的小正方形，这些小正方形覆盖的区域就是你可以进行地形调整的区域。按住鼠标左键并向上拖动，一个小山出现了，如图所示。非常棒，根据表　的内容继续熟悉地形操作，并且制作一个您满意的富有幻想的游戏场景！

注意：在您编辑地形的时候，可能会遇到如下问题：您所控制的玩家化身可能会掉到地面之下！不用担心，如果发生上述情况，请选择 Camera→Toggle Camera 回到“ 上帝 ”视角，并移动摄像头到地表以上某个位置，选择 Camera→Drop Player at Camera 选项（快捷键 Alt + W），好了，您的玩家又回到地面上了。如果你离地面过高，会有一个下落的过程。

现在，请选择 File→Save Mission As…选项，选择“ GameOne/data/missions ”文件夹目录，将刚刚我们完成的工作保存为“ GameOneMission.mis ”。记住，随时保存完成的工作也是非常好和必要的习惯。

好，现在让我们把这个讨厌的蓝、白棋盘格换掉吧，我已经受够它了，您呢？选择 Window→Terrain Texture Painter 选项，找到屏幕右侧纹理窗口中的左上角那个，就是显示棋盘格子的纹理，单击“ Change ”按钮，选择 GameOne/data/terrains 文件夹下的 sand.jpg 文件，并单击 Load 按钮。变了变了，是不是很棒？

光是沙漠就能满足我们的欲望么？当然不是。接下来再弄出点草地。单击 sand 纹理下面空文理窗口下面的“ Add… ”按钮，这里需要注意的是在 Terrain texture Painter 中添加纹理一定要按顺序进行 !这次，选择“ patchy.jpg ”然后 Load。把鼠标移动到游戏世界的地表，点击，您会发现您的画刷覆盖的地方的纹理变成了沙草混合纹理了。并且，绿草区域和临近的沙子区域自动混合好了。如下图 2.4 所示。

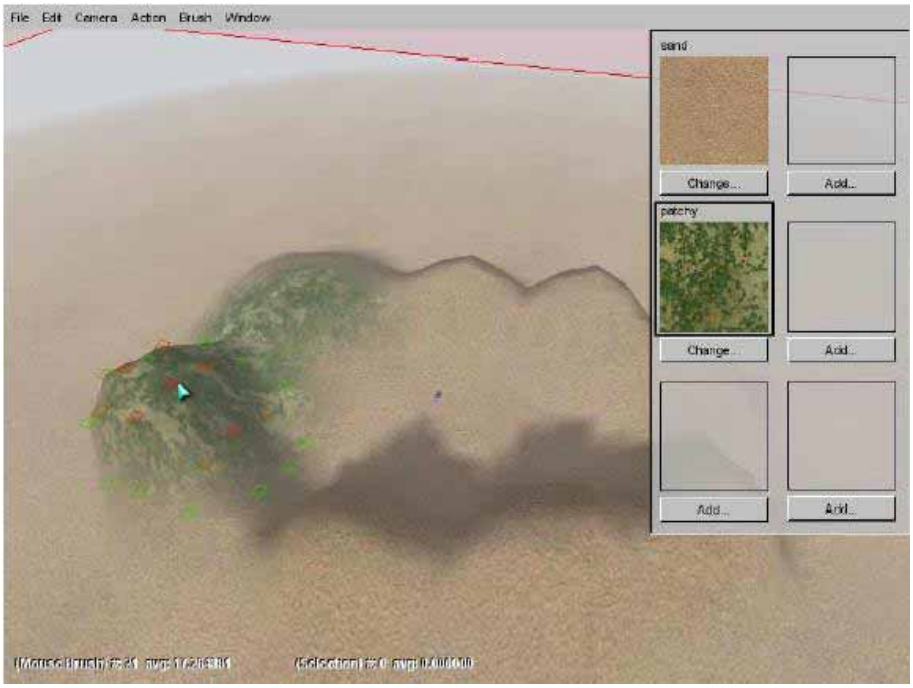


图 2.4　GameOne 中添加地貌纹理

如果您愿意，请尽情地在您创建的游戏世界里奔跑一下。嗯，您可能注意到了，当您低头看脚下附近的地方时，觉得纹理非常模糊，这绝对是会令玩家郁闷事情。幸好，Torque 允许我们使用另一种纹理—细节纹理，它是一种只在近处看得到的纹理。请选择 Window→World Editor Inspector 选项，然后选择右上方窗口中的 MissionGroup 组下面的 Terrain-TerrainBlock 对象，这时，所有关于 Terrain 的数据会出现在右下方的 Inspector 窗口中。在“ media ”区，找到“ detailTexture ”，点击右边的“ … ”按钮，选择 GameOne/data/terrains/detail1.png 文件，单击 Load。现在再来看看地面，这才是我们想要的效果！保存我们的游戏场景，接下来我们将在游戏世界中添加更多的物体。

先确保处于“ 上帝 ”视角下，选择 Window→World Editor Creator 选项，在右上窗口的内容同“ World Editor Inspector ”一样，以树状结构显示场景中的以

有对象，而下半部分“ Creator window ”显示出一系列可以放置到场景里的对象。默认的对象放置模式是“ Drop at Screen Center ”，因此调整视点对准你要放置对象的大致地方，精确的位置可以在“ Inspector ”中进行设置。

准备就绪，开始添加游戏对象。在右下方窗口，点击 Shapes 左边的“ + ”号展开树型结构，再展开 Items，这里有一个 TorqueLogoItem 对象。点击该对象，在游戏世界中创建一个 Torque 的三维 Logo 对象。在 Logo 对象周围包围着一个黄色的盒子，表明该对象处于选中状态。在它下面的数字是它的 ID 号，也就是句柄，（null）是该对象的名字，没有名字的对象不会对游戏造成任何影响，但是具有合理有效的名字不是更好么？此外，代表 X、Y、Z 坐标轴的箭头可以按住并拖动，从而改变对象在游戏世界的位置（坐标）。如图 2.5 所示：

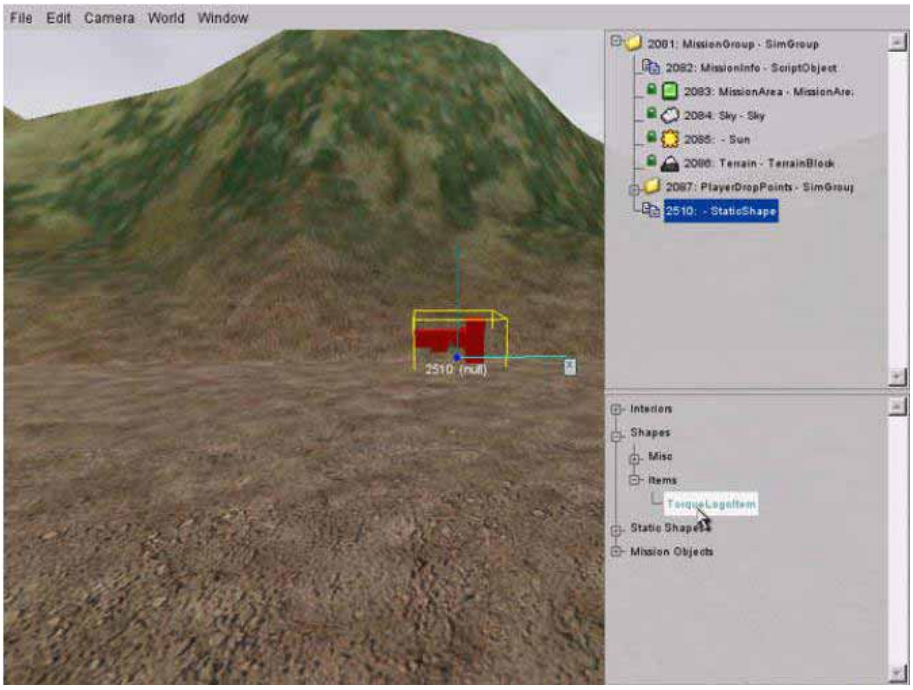


图 2.5　向游戏世界里添加 Logo 界面

拖动对象的确很方便，但是却不精确，我想您对于如何摆放您的游戏对象一定有自己的想法，并且对对象在游戏世界的坐标有相应的要求吧？好的，Torque 已经为您完成了这个工作。还记得前边提到过的 World Editor Inspector 么？什么，忘了？没关系，我们现在就来使用它！选择 Window→World Editor Inspector，在右边上面的窗口可以找到 MissionGroup，它包含了已经出现在我们游戏中的所有对象。如果没有看到它的列表内容，请点击它左边的+号。在 GameOne 文件夹（复制于 tutorial.base 文件夹）中，torque 已经为我们创建了最基本的元素，如

MissionArea（我们游戏的任务区域），Sky（游戏的天空盒子对象），Sun（太阳对象），Terrain（游戏的地形对象）以及 PlayerDropPoints（玩家的出生位置），还有我们刚刚创建的 TorqueLogoItem 对象。点选其中任何一个对象，您会在右边下部的窗口看到该对象的全部属性。

注意：如果在您打开 inspector 窗口之前已经选择了这个刚刚创建的 StaticShape 对象的话，您只需要先点选一下树型结构的其他对象，然后再点选回来就可以查看它的所有属性了，好像不太智能？呵呵，这个我也无能为力。

在 inspector 状态下，我们可以对选定的对象进行诸如位置、大小、旋转等一系列变化设置。试着把 scale 从“ 1 1 1 ”改成“ 2 2 2 ”，然后按“ Apply ”按钮确定。哈，我们的对象整整变大了 8 倍！哦，你问我为什么是 8 倍而不是 2 倍？我指的是它的体积扩大 $2^3=8$ 倍。如果您不喜欢，没问题，按 Ctrl+Z 撤销刚才的操作，请记住这个快捷组合键，它非常有用。在“ Apply ”按钮右边的文字输入框可以为该对象命名，请输入“ logo ”，按“ Apply ”确定。这时，你会发现原来没有名字的时候，在游戏中该对象上面显示的是一个数字 ID（这个数字是该对象的句柄，在游戏中是唯一的，同个它可以访问该对象并进行相关操作）和 null 字段，输入 logo 并 apply 后，null 变成了 logo 字段。注意：这个操作不会对游戏的设计有任何的影响，但是为每个对象指定一个名字也不是坏事，应该养成这个习惯，便于查找。

根据这个教程的内容，我们继续在游戏世界里放置 5～10 个 logo（如果您不嫌累的话，放更多的 logo 我没有任何意见），然后我们一起来做一个搜集 logos 的小游戏。

如果现在就开始我们的收集游戏是不是太简单了点呢？当然，我是指我们这个空旷的游戏场景，尽管我们已经在地表扑上了沙子和草坪等材质，并制作了一些小山包。好，那我们就再添加一些建筑物。Torque 中，建筑物是.dif 文件保存的，放在 data/interiors 文件夹下，为什么用 interiors 而不用 buildings，我也不太清楚，大概是想表现一种“ 大的可以直接走到其内部去 ”的建筑对象。下面就用 torque 自带的 interiors 丰富一下我们的游戏场景吧。选择 window→World Editor Creator，在右下窗口的 Interiors→GameOne→data→interiors 文件夹中，象 logos 操作一样，点击“ box ”，屏幕中央就出现了一个大的 box。

注意：这时你会发现新创建的 box 对象是黑色的，表面没有任何材质。这需要我们重新渲染一下游戏场景，可以采用 2 种方法。第一种是在屏幕下方弹出的询问框中点击“ yes ”；另一种方法是选择 lighting→Relight Scene。以后每当新创建 interior 类型对象或者是移动 interior 对象时候，都需要通过这样的操作进行重新渲染游戏世界。

创建出来这个 interior 的 box 对象后，进行位置等调整，直到您觉得满意为止，我相信您不会让 box 的一半在地面上，另一半却在地面下吧？该 box 对象有一个门，您可以直接控制您的玩家化身通过这个门进入到 box 内部，哈哈，酷吧？如果您愿意，放一些 Logo 在 box 里。

2.3.3 组织对象

在继续制作之前，请允许我占用您一点宝贵的时间来介绍一下如何组织您的游戏对象。首先让我们看一下 MissionGroup 列表，目前，添加了一些列对象的列表里面的内容稍微有些臃肿，不过还算 OK，不太难管理。但是，您的游戏不会仅仅就这些对象就搞定了吧？当您完成一个内容丰富、十分优秀的完整游戏时，想想，这个列表会有多大！我已经开始挠头了。所以，我们迫切需要类似文件夹的方式组织管理我们的游戏对象，幸好，torque 已经为我们准备好了。这就是 SimGroups 命令。

要创建 SimGroup，在 world editor creator 右下窗口里的 Mission Objects 目录下的 system 目录，这里您会找到 SimGroup 对象，点击一下，在弹出的对话框中，输入您需要的名字，这里我们先填写“ logos ”，点击 OK。在右上部的窗口中，把前面创建的所有 logo 对象拖拽到 Logos(SimGroup)目录下。还有，刚才我们创建的 interior 类型的对象还没有命名，现在给它起个满意的名字吧，如果你添加了多个 interior 对象，那么，为这些对象建立一个类似 building 的 SimGroup 不是很好么？好啦，现在您的 Mission Objects 树型结构整洁多了吧？

注意：我们完全可以先建立 SimGroup，再在里边创建我们需要的对象，这更符合我们的工作习惯。按住 Alt 键＋鼠标左键单击创建好的 SimGroup 文件夹，该文件夹会变成绿色，说明该文件夹是默认的创建对象的文件夹，即所有之后创建的对象都在该文件夹下，很方便吧。

下面，我们需要为我们的游戏添加一些 GUI 元素。GUI（Graphic User Interface，图形用户接口）包括游戏的菜单系统以及游戏中显示在屏幕上的分数、体力、生命等状态条等对象。Torque 游戏引擎为我们提供乐一个强大的 GUI 编辑器，并且提供了很多预定义的对象方便我们使用，同时我们还可以使用脚本代码创建新的 GUI 元素。下面，我们为 GameOne 游戏添加一些 GUI 元素使它不断完善，成为一个真正的游戏。

请先回到 TGE 的主菜单界面，您可以关闭程序再进入或者选择 File→Quit 选项。在开始界面选择 GUI 编辑器，即游戏窗口上部最左边的图标，或者使用快捷键 F10。这样你会看到如下图的 GUI 编辑界面，是不是同 World Editor Inspector 很相似？

在 GUI 编辑状态下，中央上部有三个下拉菜单：New Control（新空件选择菜单）、GUI file selection（GUI 对象选择菜单）和 screen size（屏幕大小选择）菜单。New Control 菜单包括很多可以创建的 GUI 对象，如按钮、文本框等对象。GUI file selection 菜单包含所有游戏中已经创建的 GUI 对象，我们目前处在 MainMenuGui 界面中，如下图所示。建议您选择该菜单下的每个选项看看，看看显示在游戏窗口的内容，方便您理解。需要注意的是，如果这个时候选择 PlayGui 会得到一个淡绿色毫无内容的窗口，这是因为您还没有进入到游戏中，torque 还没有加载 PlayGui，希望这不会迷惑您。最后是右边的 screen size 菜单，里边有 640×480、800×600 和 1024×768 三种分辨率。不同的分辨率会得到不同的显示效果。

下面开始我们的操作。首先，为我们的游戏选项界面更换一个新的背景图片。在右边窗口的树型结构中选择 MainMenuGui，在下面的窗口中找到 Misc 项目，下面有个 bitmap 选项，通过拖动窗口左边的分界栏可以看到整个文件名。现在把您的文件名从“ background ”改为“ gameonebg ”，然后点击 Apply。看，我们的背景图片变化了，如果您没有发现变化说明您有点粗心哦。

如果您觉得窗口上部分的哪些按钮图标影响美观，您完全可以删除它们！选中一个删除一个就 OK 了。这样是不是整洁了不少？但是这个时候您需要打开 GUI 编辑器或者是世界编辑器，就不得不使用快捷键 F10 和 F11 啦。如果出现了其它无法解决的问题，您只要简单地把 tutorial.base/client/ui/MainMenuGui.gui

文件拷贝过来覆盖 GameOne/client/ui/MainMenuGui.gui 文件就可以解决问题。

下一步，我们为开始界面添加一个按钮，点击它，就进入我们的游戏世界，类似所有其它游戏的操作。在“ New Control ”中选择 GuiButtonCtrl。您可以看到在游戏界面的左上角出现一个新的按钮。拖拽它放到一个领您觉得满意的地方，比如 GameOne 标题下方屏幕居中的位置。再在右下方窗口中找到 Misc 项目的 text 选项，键入“ Start ”并点击“ Apply ”。现在按钮上显示出来的就是“ Start ”字符了。接着，我们要为该按钮添加命令脚本，这样点击它才能执行命令。在 Parent 区的“ command ”项中填入“ loadMyMission(); ”，并点击“ Apply ”确定。注意，loadMyMission()后边的“ ；（分号） ”一定不要忘记填写上。

现在，我们已经得到了一个比较满意的开始界面，在我们继续工作之前，先看看它是否适合于各种不同的分辨率格式。请您把屏幕分辨率从 640×480 改为 1024×768，观察一下，您会发现按钮的位置发生了变化！下面让我们来解决这个问题。先回到 640×480 模式，在右下方窗口中把“ horizSizing ”和“ vertSizing ”项都改成“ relative ”，然后点击“ Apply ”确定。然后在尝试着改变屏幕的分辨率，这次，按钮和屏幕的位置一致了。点击“ File ”菜单的“ Save Gui… ”，选择目录为“ GameOne/client/ui ”，然后点击“ Save ”按钮。按 F10 推出 GUI 编辑器并退出 Torque。现在重新启动 torqueDemo，在开始界面，点击“ Start “ 按钮，看，我们进入了 GameOne 游戏。

接下来我们要做的是为游戏玩家在游戏屏幕上创建一个显示分数的 GUI 对象。在游戏界面中，按 F10 打开 GUI 编辑器，当前我们处于 PlayGui 界面中，在“ GUI file selection ”的下拉菜单中已经被选中了。

现在我们来添加分数计数器对象。在“ New Control ”下拉菜单中选择 GuiTextCtrl。确定您的新空件在树状结构上处于选定状态，然后，我们准备在右下方的 Inspector 窗口中做一些修改。首先，请把这个新空件命名为‘ ScoreCounter ”（在 Apply 按钮旁边的输入框）。接着，点击 Parent 项目中的 profile 按钮，选择 GuiBigTextProfile，这是一个用来显示大文本的标准格式。然后，在 General 项目中的 text 域中填入“ Score: 0 ”，并点击 Apply 确定。这时候，我们将看到一个大的、清晰漂亮的文本文字。直接用鼠标将它拖拽到屏幕上任何一个您满意的地方，如果您希望它在不同的分辨率显示模式下保持相对的位置，那么请将

“ HorizSizing ”和“ VertSizing ”都设置为“ relative ”。再次检查您的工作，没有问题后，请选择 File > Save GUI… 选项，把它存放在“ GameOne/client/ui/playGui.gui ”文件夹下。这样当我们再回到游戏界面时，就会在屏幕上看到显示分数的文本。

注意：如果您使用的是 TGE1.5.2 版本，而在添加了 GUI 空间并且保存之后，如果退出 Torque，当再次进入游戏时候，可能会遇到鼠标转动失灵的情况，这是屏幕上会显示鼠标的图标。别慌，这是 Torque 的一个小小的 bug，可以通过以下方法进行解决。在 GameOne/client/ui/文件夹下打开 playGui.gui 文件，在 new GameTSCtrl(PlayGui)所定义的语句的底部添加如下语句“ noCursor = "1"; ”，然后保存文件，退出，重新启动 Torque，问题就解决了。由于这是引擎的一个小小的 bug，因此每次您修改游戏画面的 GUI 的时候都会出现这种情况，只要进行上面提到的操作即可解决问题。当然，如果您直接使用脚本语言的编写来实现 GUI 那就不会出现这样的问题。

2.3.4 脚本

好了，经过以上的准备，现在我们要使这个 Demo 更加象一个小的游戏，就要编写一些脚本代码了。Torque 脚本代码的一些介绍我们将在下面的章节学习。现在，我们只是快速浏览一些脚本并写一些简单的代码。如果您没有编程经验的话，这部分内容看起来可能些难度，但是没关系，只要您跟着我的思路走下去，就一定会理解我的做法。记住，目前我们只对 Torque 脚本的运行有一个大概的概念。

前面我们简单了解了 example 文件夹下的根 main.cs 文件内容，现在我们再来看看 GameOne 文件夹下的 main.cs 文件。用一个您喜欢的文本编辑器打开该文件，并拖动滚动条到文件底部，您会看到一行“ LOAD MY MISSION ”字样的标题。它下面定义了 loadMyMission 函数，这个函数是我们在 main menu 的 start 按钮的 command 域中填入的函数，还记得么？当我们点击 start 按钮的时候，这个函数里的脚本就会被执行。下面我们一起来看下这段代码，并理解一下它们的含义。

disconnect();函数的第一条语句，它的作用是如果我们目前与服务器有连接，

则断开该连接。这种操作非常标准，通常出现在程序中起着初始化清空的作用。

createServer("SinglePlayer",expandFilename("./data/missions/gameonemission.mis"));

使用 createServer 命令在本机创建一个游戏服务器，括号中的第一个参数表示创建的服务器的类型，这里是单人游戏类型；第二个参数指定在该服务器上加载的任务文件在磁盘上的位置，也就是./data/missions/文件夹下的 gameonemission.mis 文件，这个文件是我么在前面亲手创建的，还记得么？

%conn = new GameConnection(ServerConnection);然后我们使用 new GameConnection 命令为我们的客户端与服务器间创建一个新的连接对象，该对象的名字是 ServerConnection，该对象的句柄赋值给%conn 变量。前面我们已经介绍过了，Torque 是采用服务器/客户机结构设计的，即使我们要做的是一个单机的小游戏，也会同时建立服务器端和客户端，然后为服务器端和客户端建立一个连接，所有的信息通信全部通过该连接进行。注意，这个时候只是创建了一个连接，而这个连接还没有真正地连接到服务器上。

RootGroup.add(ServerConnection);使用 RootGroup 的 add()方法将创建好的连接添加到游戏的资源管理器列表中。

%conn.setConnectArgs("Player");设置连接参数为 Player。

%conn.setJoinPassword("None");设置连接密码，None 表示没有密码。

%conn.connectLocal();使用 connectLocal()方法将%conn 连接到服务器端，由于这里我们的服务器和客户端都是我们自己的电脑，因此使用的是本地连接方法。

以上就是 loadMyMission 函数的全部内容，总的来说，该函数为我们创建了一个服务器同时开始一个新游戏（连接到这个服务器上）。

注意，该函数完全为了 GettingStarted 的教程而存在的，在其它游戏 demo 的文档中您不会看到它，希望您不会因此而迷惑，把它当作游戏必须的函数。

接下来继续我们的工作。在文本编辑器中打开 GameOne/server/logoitem.cs 文件。在这里您可以看到 TorqueLogoItem 的一些数据块定义。数据块的定义包括您的游戏对象的一些特殊信息，这在网络中特别有用，因为它允许游戏对象的所有不变信息在您的 mission 载入的时候只传送一次。这部分内容我们将在下面的章节进行详细说明。

现在，我们要在 TorqueLogoItem 里添加一些功能，使得玩家化身碰到 TorqueLogoItems 模型的时候，它们会有一些适当的行为。请将下面的代码输入到 GameOne/server/logoitem.cs 文件中，并保存它，后面我会介绍它的作用。

```
function TorqueLogoItem::onCollision (%this, %obj, %col)
{
    if (%col.getClassName() $= “Player”)
    {
        %client = %col.client;
        %client.score++;
        commandToClient (%client, ‘SetScoreCounter’, %client.score);
        %obj.delete();
        %logoCount = logos.getCount();
        if (%logoCount > 0)
            return;
        //otherwise display victory screen
        commandToClient (%client, ‘Show Victory’, %client.score);
        }
    }
```

任何一个定义了数据块的对象都具有一个名为 onCollision 方法（函数），这个方法是在引擎中定义的，在这个对象和其它对象发生碰撞的时候，引擎会自动调用这个函数。即使我们没有用脚本定义这个 onCollision 方法（函数）的内容，在发生碰撞的时候，引擎也会调用该方法，只是什么都没有发生而已。调用时，引擎会传三个参数，%this 代表数据块的句柄，%obj 代表这个对象本身（的句柄，虽然我们可以将%obj 理解为对象，实际上，它保存的是对象的句柄），%col 代表和这个对象发生碰撞的对象（的句柄）。下面我们来看看以上代码的含义。

```
if (%col.getClassName() $= “Player”)
```

这行代码表示判断与 logo 对象想碰撞的对象是否是游戏的玩家化身，只要是通过 PlayerData 数据块创建的对象，其 ClassName（类名）都是 Player。

```
%client = %col.client;
```

```
%client.score++;
commandToClient (%client, 'SetScoreCounter', %client.score);
```

接着，我们定义了一个%client 变量（局部变量）来保存碰撞这个 logo 对象的玩家化身的客户端句柄，然后增加该客户端的分数，再发送一个包含分数变化信息的消息给客户端。再次提醒您，这个单人游戏仍然模拟了服务器端和客户端，分数的改变是在服务器端处理的，然后传送给客户端。

注意：Torque Script 的变量使用“ % ”或者“ $ ”标识符直接创建一个变量，它不需要事先声明这个变量及其类型（如 int、char 等），这点同 C 或者 C++语言不太一样。因此，您会经常在脚本程序中直接看到新变量的出现，这时，请提醒自己，这是个刚刚定义的变量，而不要因为从没有见过这个变量而使自己迷惑。

```
%obj.delete();
%logoCount = logos.getCount();
if (%logoCount > 0)
    return;
//otherwise display victory screen
commandToClient (%client, 'Show Victory', %client.score);
```

当碰撞发生之后，调用 logo 对象的 delete()方法将该对象删除，这样我们的感觉就是当玩家化身碰到 logo 对象时候，logo 对象就消失了，好像被我们捡走了一样。之后，我们用%logoCount 这个变量保存 logo 对象的剩余数目，这些对象放在了 logos 的 simgroup(不记得了么？我们在前面创建的名为 logos 的 simgroup)中，getCount()方法可以获得它的数量。如果我们放置的那些 logo 对象没有全部被捡起来，则返回，如果全部捡起来的话，就发消息告诉客户端已经胜利的完成任务了。

onCollision 函数总是在服务器端调用的，而不是在客户端。这就是为什么我们总是不断的发送消息给撞到了这些 logo 对象的玩家的客户端。将这样的处理放在服务器端是防止不道德的玩家在客户端作弊的最好方式。所以，类似这样的函数必须位于 server 文件夹下。

最后，我们还有一些客户端的代码来处理这些消息。

在客户端，我们需要新建立一个脚本文件。打开 GameOne/client 文件夹，创

建一个新的文本文件，名为“ clientGame.cs ”，然后输入下面的代码并且保存。

```
function clientCmdSetScoreCounter (%score)
{
    ScoreCounter.setText(“Score:” SPC %score);
}
function clientCmdShowVictory(%score)
{
    MessageBoxYesNo(“You Win!”,
        “Would you like to restart the game?”,
        “loadMyMission();”,
        “quit();”
}
```

这里，我们根据服务器端的需求定义了两个函数，一个用来在客户端的 GUI 中改变显示的分数，一个在完成游戏任务的时候显示一个对话框，询问玩家是否重新开始。

注意观察一下客户端和服务器端函数的名称，您会发现，客户端函数的名字和服务器端函数 commandToClient 的第二个参数一样，只是在这个名字前面加了一个“ ClientCmd ”前缀。这个就是标记字符串变量的用法。

好啦，我们目前的脚本工作已经接近尾声啦。为了使我们刚刚创建的 clientGame.cs 文件能够被正确载入，请打开 GameOne/main.cs 文件，找到 initClient 函数里的“ Client Scripts ”节，在这节代码的下面添加如下一行代码，自此，大功告成！

```
exec(“./client/clientGame.cs”);
```

2.3.5 游戏试玩

恭喜您！属于您的 GameOne 游戏完成了！退出 Torque（如果需要的话）并重新运行它，点击 start 进入游戏，从头到尾尽情地玩一遍，在您的游戏世界中四处游荡一下，碰碰您的 logo 对象。它们是否正常运行？如果还没有，请重新检查前面的操作，看看是哪里出现了错误。显然，当您开始正式制作您自己的游

戏的时候，就没有教材来帮您检查错误了，所以您可能会花费大量的时间在测试上，也就是说您要检查一切可能会在游戏中出错的地方并修正它们，也就是所谓的修改 bugs 了。一个优秀的游戏是不应该有 bugs 的，虽然不容易实现，不是么？

好了，GarageGames 公司为我们精心准备的 GettingStarted 教程到这里就学习完了，现在，您应该对 Torque 引擎的强大功能有了一个深刻的印象了吧。并且，用它来制作一款属于您自己或团队的并且有特色的游戏并不是一件难事，只要您相信自己，足有足够的热情和坚定不移的毅力，您和您的团队就一定会获得成功！

2.4 本章小结

在这一章中，我们一起学习 GarageGames 公司为我们精心准备的 Getting Started 教程。通过对该教程的学习，使我们基本了解了 Torque 游戏引擎的强大功能，并在教程的指导之下制作了一款虽然简单，却具备了相当的游戏雏形的一款游戏。您会发现，使用 Torque 游戏引擎制作一款自己梦寐以求的游戏，原来并不困难。下一章，我们将学习 Torque 游戏引擎中的编辑器及其功能。

第三章　Torque 的编辑器

通过上一章的学习，您已经对 Torque 的世界有了直观的认识，相信您已经非常喜欢它了。本章，我们将详细学习 torque 的编辑器，这个功能强大的编辑器将会一直陪伴着我们。Torque 主要包含两个编辑器，World Editor（世界编辑器）和 GUI Editor（GUI 编辑器）。相信您已经熟悉它们了，下面将系统地学习它们。

3.1 World Editor

Torque 的 World Editor（世界编辑器）具有良好的用户界面，窗口顶部是菜单栏，左边较大的是游戏窗口，右边上面的是游戏资源管理器，当选中不同项目时，右下边显示出选中项目的属性数据。接下来我们逐个学习各个菜单的功能。

3.1.1 File 菜单

使用 File 菜单中的选项可以进行磁盘和文件的操作，如下表 3.1 所示。这些操作包括打开、保存、导入、和导出等等。

表 3.1　File 菜单的内容

选　项	说　明
New Mission	创建一个具有默认地形和天空的新的任务
Open Mission (Ctrl O)	打开一个现有的任务
Save Mission (Ctrl S)	保存当前任务
Save Mission as	将当前任务另存为其它名称
Import Terraform Data	从现有地形文件中导入地形
Import Texture Data	从现有地形文件中导入纹理
Export Terraform Bitmap	输出当前地形地图为位图（只有在 Terrain Terraform Editor 中可用）
Toggle Map Editor (F11)	在游戏界面和世界编辑器界面切换
Quit	退出程序

3.1.2 Edit 菜单

Edit 菜单包含了许多对象编辑命令，还包括可用于获取各种编辑器设置的命令。如表 3.2 所示。

表 3.2　Edit 菜单的内容

选　项	说　明
Undo	取消最近一次的操作（注：不是所有操作都能取消）
Redo	重复上一次的撤销操作
Cut	在 World Editor 中剪切任务的对象到剪贴板
Copy	在 World Editor 中复制任务的对象到剪贴板
Paste	粘贴当前剪贴板中的内容到游戏世界中
Select All	选择 world Editor 中的所有任务对象
Select None	清除在 World Editor 和 Terrain Editor 中的当前选项
World Editor Settings	获取 World Editor 的设置对话框
Terrain Editor Settings	获取 Terrain Editor 的设置对话框

其中 World Editor Settings 内容如下图 3.1 所示，所有选项默认为选中状态。

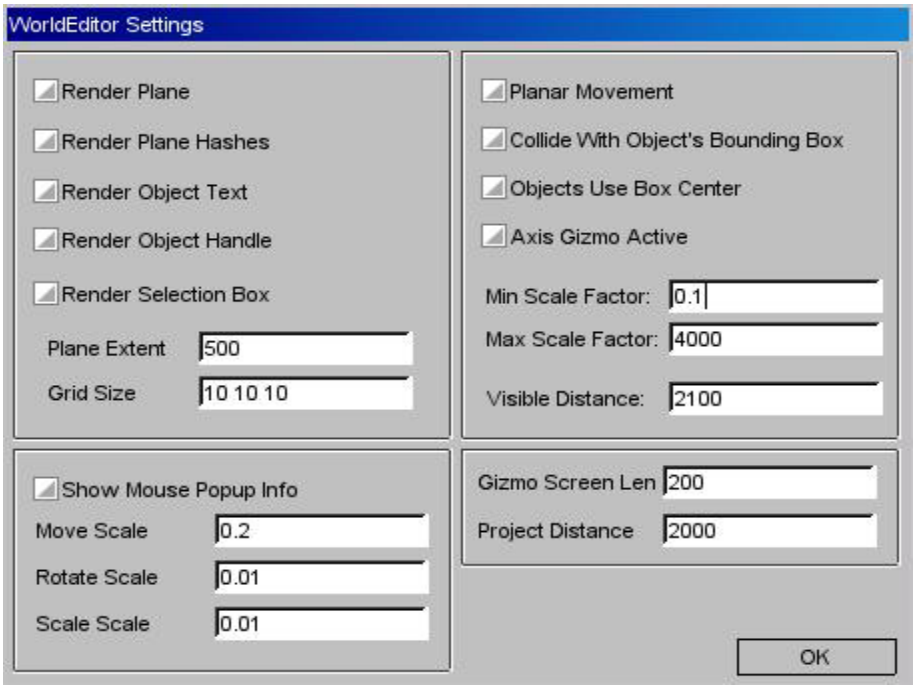


图 3.1　World Editor Settings 界面

Render Plane：选中对象时显示其句柄所在平面；

Render Plane Hashes：选中对象时显示栅格；

Render Object Text：显示对象的名字和 ID；

Render Object Handle：显示对象的句柄（红点）；

Render selection Box:显示选择包围盒（黄色）；

Plane Extent：长宽尺寸，默认 500；

Grid Size：栅格的变化单位，默认为 10 10 10；

Show Mouse Popup Info：对物体进行位移、旋转、缩放操作时显示坐标信息；

Move Scale：使用鼠标移动对象操作的灵敏度，默认为 0.2；

Rotate Scale：使用鼠标旋转对象操作的灵敏度，默认为 0.01；

Scale Scale：使用鼠标缩放对象操作的灵敏度，默认为 0.01；

Planar Movement：拖拽对象移动时，对象保持在一个平面上；

Collide With Object's Bounding Box：点击对象边界盒子任何地方既可以选中对象；

Objects Use Box Center：选中时，对象的句柄在对象的中心，否则，对象的句柄在对象边界盒子最底部；

Axis Gizmo Active：显示坐标轴；

Min Scale Factor:对象缩放的最小尺度，默认 0.1；

Max Scale Factor：对象缩放的最大尺度，默认 4000；

Visible Distance：可以看到/选中对象句柄的最大距离；

Gizmo Screen Len：屏幕上显示坐标轴的长度，默认 200 象素；

Project Distance：选择指针的投射距离，默认 2000；

OK：接受设置数据。

Terrain Editor Settings 如下图 3.2 所示：

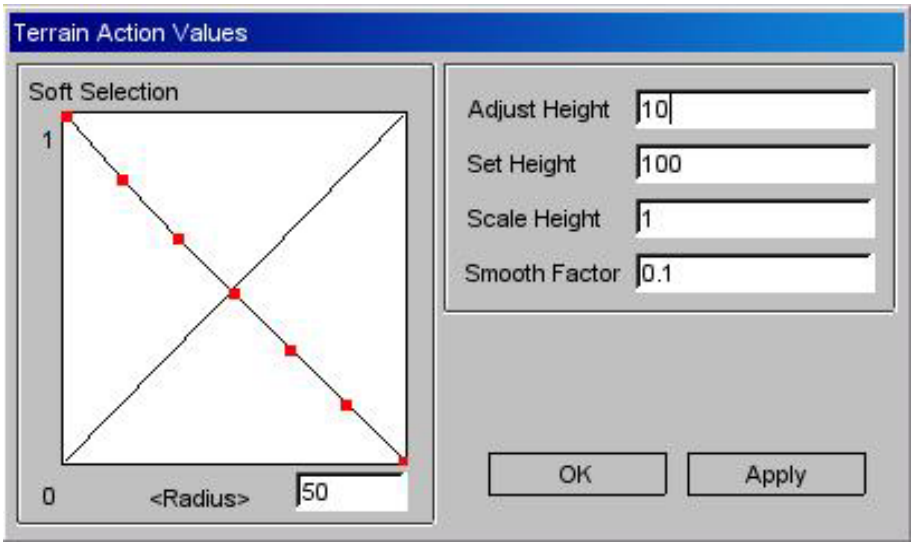


图 3.2　Terrain Editor Settings 界面

Soft Selection：使用 spline（样条线）来调节 brush（画刷）的硬度，左边控制中心，右边控制边缘；

<Radius>：选择模式下 brush（画刷）的半径，默认 50；

Adjust Height：设置高度增量，默认为 10；

Set Height：设置高度；

Scale Height：高度增量，默认为 1；

Smooth Factor：光滑因子，数值越低，越不明显，默认为 0.1。

3.1.3 Camera 菜单

Camera 菜单可以改变摄像头模式并调节其飞翔速度，如下表 3.3 所示：

表 3.3　camera 菜单的内容

选项	说明
Drop Camera at Player (Alt Q)	设置摄像头模式为飞翔模式，并移动到游戏玩家的位置
Drop Player at Camera (Alt W)	设置摄像头模式为玩家模式，并移动到游戏玩家的位置
Toggle Camera (Alt C)	在游戏玩家和飞翔模式之间切换
Slowest (Shift 1)	设置摄像头的移动速度，最慢
Very Slow (Shift 2)	设置摄像头的移动速度，很慢
Slow (Shift 3)	设置摄像头的移动速度，较慢
Medium Pace (Shift 4)	设置摄像头的移动速度，正常（默认）

Fast (Shift5)	设置摄像头的移动速度，较快
Very Fast (Shift 6)	设置摄像头的移动速度，很快
Fastest (Shift 7)	设置摄像头的移动速度，最快

3.1.4 World 菜单

World 菜单中的选项对游戏中的对象进行操作，如下表 3.4 所示：

表 3.4　World 菜单的内容

选　项	说　明
Lock Selection (Ctrl L)	锁定当前选项，使之不能从 World Editor 视图操作
Unlock Selection (Ctrl Shift L)	解除一个锁定的选项
Hide Selection (Ctrl H)	隐藏当前的选项
Show Selection (Ctrl Shift H)	显示选中的隐藏对象
Delete Selection (Delete)	删除当前选中的对象
Camera To Selection	移动摄像头至选中对象
Reset Transforms	重新设置选中对象的旋转和缩放信息
Drop Selection (Ctrl D)	放置选中对象至游戏世界中
Add Selection to Instant Group	将选中对象加入对象组
Drop at Origin	将新建对象放置在游戏世界原点（句柄位置在原点上）
Drop at Camera	将新建对象放置在摄像机所在位置
Drop at Camera w/Rot	将新建对象放置在摄像机所在位置，并与镜头同方向
Drop below Camera	将新建对象放置在摄像机下面
Drop at Screen Center	将新建对象放置在实现屏幕中心（默认）
Drop to Centroid	将新建对象放置在游戏世界原点
Drop to Groud	将新建对象放置在当前位置的地面上

3.1.5 Window 菜单

Window 菜单里包括许多专门用于构建我们游戏世界的命令，是编辑器中最重要的部分，其内容如表 3.5 所示。

表 3.5　Window 菜单的内容

选　项	说　明
World Editor (F2)	进入游戏编辑界面（暂时作用不大）
World Editor Inspector (F3)	检查和指定任务对象的属性
World Editor Creator (F4)	在游戏世界中创建新对象
Mission Area Editor (F5)	设置游戏世界中的任务区域
Terrain Editor (F6)	编辑游戏世界中的地形高度
Terrain Terraform Editor (F7)	使用数学算法生成地形高度
Terrain Texture Editor (F8)	编辑地形纹理
Terrain Texture Painter	快速添加地形纹理

下面，对每个菜单选项进行详细介绍。

World Editor：世界编辑器，该选项只是在游戏的编辑状态和非编辑状态切换，用处不是很大。

World Editor Inspector：世界编辑器指示器，用来检查和指定任务对象的属性。在 Inspector 模式下选择一个对象，其属性会显示在屏幕右下角的框架中。可以直接在其中编辑、修改对象的属性，单击 Apply 按钮接受修改。单击 Dynamic Fields 下边的 Add Field 按钮可以为对象添加动态属性，可以通过教本语言来访问，一般用于将特殊游戏属性添加至对象。

World Editor Creator：世界编辑器创建器，该创建器在屏幕右下方提供一个类似于 window 操作系统的资源管理器形式的树状文件夹目录结构，用来创建新的游戏对象，如右边的小图所示。操作非常简单，点击＋号，在目录树中找到想添加的对象的位置，点击其名字即可以在游戏当前选定的对象添加方式（见 World 菜单）下将新对象添加到游戏世界中。

Interiors 中的内容就是你的游戏文件夹下 data 文件夹中 interiors 文件夹的内容，所有.dif 文件都会出现在该选项下，可以添加到游戏世界中。

Shapes 的内容是在 Torque Script 中定义了 datablock 的对象。

Static Shapes 中的内容是游戏文件夹下 data 文件夹中 shapes 文件夹中的内容，所有.dts 文件都会出现在该选项下，可以添加到游戏世界中。

Mission Objects 内容非常有用，我们会多花一些时间来学习它。

Mission Objects 包含 3 个内容，Environment、Mission 和 System。其中，Environment 的内容全部跟游戏的环境有关，Mission 则和游戏的任务有关，而 System 的内容我们已经相当熟悉了。

Environment

Sky

Sky 用来创建天空对象，游戏中已经默认为我们创建了一个天空对象。点击 sky 将得到如图 3.3 所示的创建 Sky 对象对话框，其中：

Object Name：是你要创建的天空对象的名字，通常是 sky。

Material list：是用来描述天空盒子的材质的.mdl 文件。

Cloud0～2 Speed：用来指定三层云的移动速度，二维向量。Cloud0～2 Speed：用来指定三层云的高度。

Visible distance：可视距离，游戏世界中距离玩家眼睛的距离超过该数值的一切对象都不会进行渲染，通过设置该值可以提高游戏的渲染速度，值越小、速度越快，反之，则越慢。

Fog distance：开始产生雾化模糊的距离，从该点到 visible distance 之间会逐渐模糊，直到完全消失。

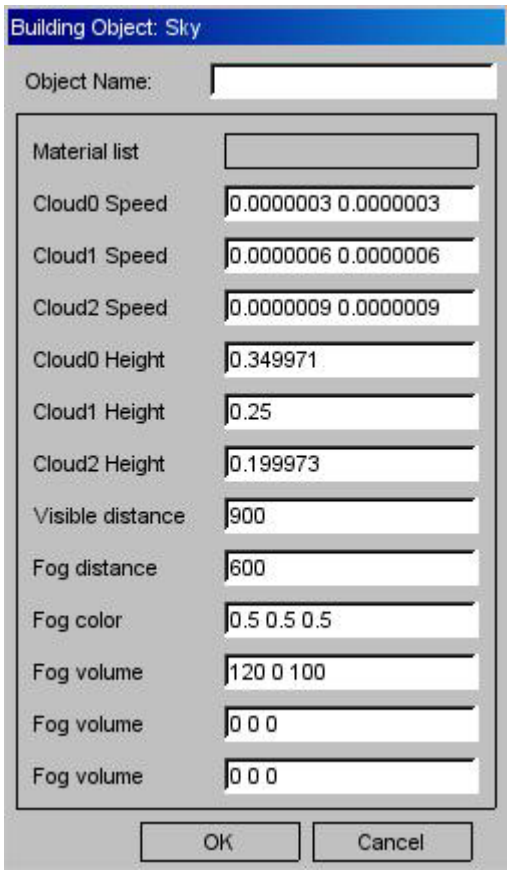


图 3.3　创建 Sky 对象对话框

注意：您应当始终确保 Visible Distance 比 fogDistance 大，否则在某些情况下会使得游戏引擎在客户端崩溃。保证 Visible Distance 比 fogDistance 大至少 50 个单位是个不错的选择。

Fog color：雾的颜色，使用的是 RGB 的百分比格式。

Fog volume：包含 3 个参数，第一个是可视距离，第二个是雾的底部，第三

个是雾的顶部。

Sun

Sun 用来创建太阳对象，游戏中已经默认为我们创建了一个太阳对象，如果您的游戏世界需要两个或者更多的太阳，可以通过它来实现。点击 sun 得到如图 3.4 所示的创建 Sun 对象对话框，其中：

Object Name:您要创建的太阳对象的名字，通常情况下为 sun。

Direction：太阳的方向矢量。

Sun color：太阳光的颜色，采用 RGB 的百分比格式。

Ambient color：环境光的颜色，采用 RGB 的百分比格式。

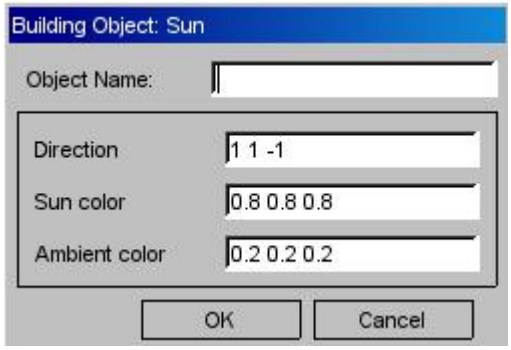


图 3.4　创建 Sun 对象对话框

Lighting

Lighting 用来创建闪电效果的对象。点击 Lighting 得到如图 3.5 所示的创建 Lighting 对话框，其中：

Object Name：您要创建的闪电对象的名字，如 Lighting1 等。

Datablock：闪电对象的数据块，默认为 DefaultStorm。如果您使用脚本创建了您需要的数据块，则可以通过旁边的按钮进行选择。

图 3.5　创建 Lighting 对象对话框

Water

Water 用来创建水（湖泊、海洋等）对象，点击 Water 得到图 3.6 所示的创建 WaterBlock 数据块对话框，其中：

Object Name：您要创建的 water 对象的名字。

Extent：覆盖的范围，第三个参数为深度。

Texture size：水材质的尺寸。

Wave Param0～3：正选波形参数，用来模拟产生波浪的效果。

Flood fill：默认为 1，表示水面下被水填满。

Seed points：种子节点。

Surface Texture：水表面使用的材质。

EnvMapTexture：水面反射周围环境的材质，通常是天空材质。

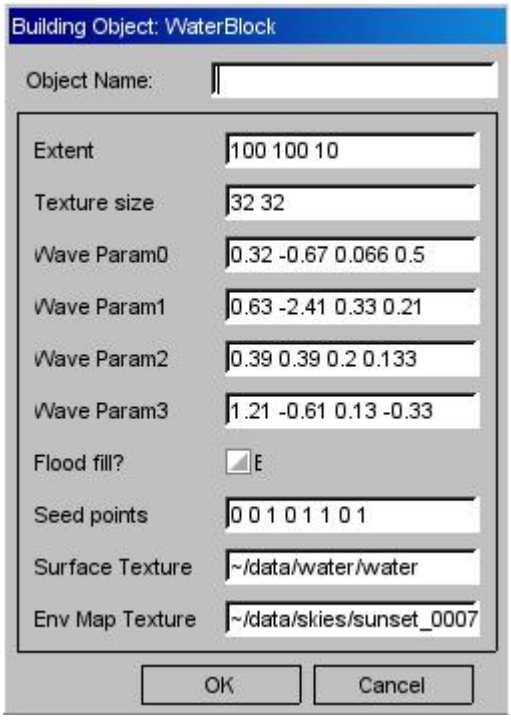


图 3.6　创建 WaterBlock 对话框

Terrain

Terrain 用来创建地形对象，游戏中已经默认为我们创建了一个地形对象。点击 Terrain，得到如图 3.7 所示的创建 TerrainBlock 数据块对话框，其中：

Object Name：地形对象的名字，通常为 Terrain。

Terrain file：使用制作好的等高图地形文件。

Square size：默认为 8，它表示覆盖 1.034 平方英里的面积。

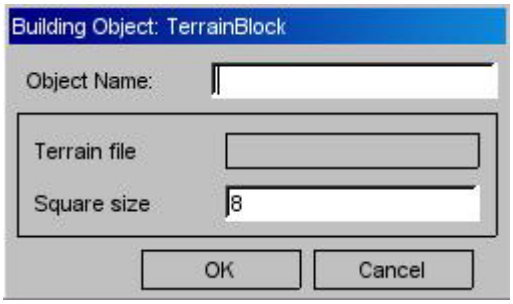


图 3.7　创建 TerrainBlock 对话框

AudioEmitter

AudioEmitter 用来创建声音的“发射器”，也就是音源。点击 AudioEmitter，得到如图 3.8 所示的创建 AudioEmitter 对象对话框，其中：

Object Name :您的游戏中需要创建的 AudioEmitter 对象的名字。

Sound Profile：需要继承的 Profile 数据块。

Sound Description ：需要继承的 Description 数据块。

Audiofile：使用的声音文件，可以是.wav 或者.ogg 文件格式。

Use profile’s desc ：是否允许屏蔽声音。

Volume：播放音频文件时候的音量 0 表示最小，1 表示最大。

Looping：是否循环播放。

is 3D sound：是否为 3D 音效，如果是，声音会随着距离的变化而变化。

Reference distance：声音的有效距离，在该距离内声音强度不会衰减。

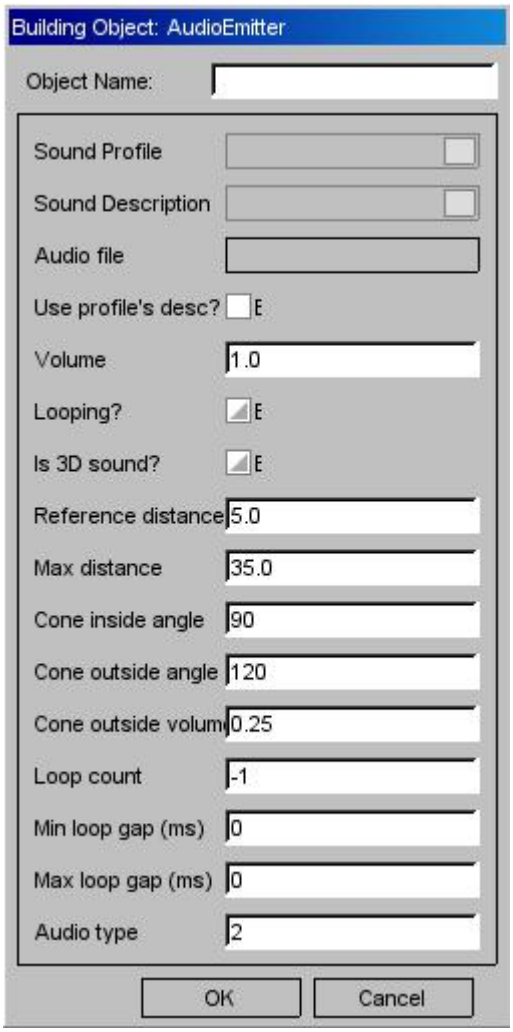


图 3.8　创建 AudioEmitter 对象对话框

Max distance：声音传播的最大距离，该距离外声音的强度会衰减到 0。

Cone inside angle：声锥的内角值。

Cone outside angle：声锥的外角值。

Cone outside volume：声锥外角之外的声音强度值。

Loop count：循环播放的次数，-1 表示无限循环。

Min loop gap：声音循环播放的最小时间间隔，单位毫秒。

Max loop gap：声音循环播放的最大时间间隔，单位毫秒。如果 Min loop gap 和 Max loop gap 的值都不为 0，引擎会在两个时间间隔中随即取得一个值作为下一次播放该声音的时间间隔。

Audio type：声音类型。

注意：关于声锥的说法可能会使您稍微迷惑，这里解释一下。声音本身是向

四面八方传播的，但是，在 Torque 中声音会有一个方向矢量，这非常符合实际。比如我们站在喇叭正前方会听到很大的声音，而如果我们站在喇叭的后面，声音就没有那么大了。图 3.9 详描述了声音对象的相关属性，相信能够对您的理解提供帮助。

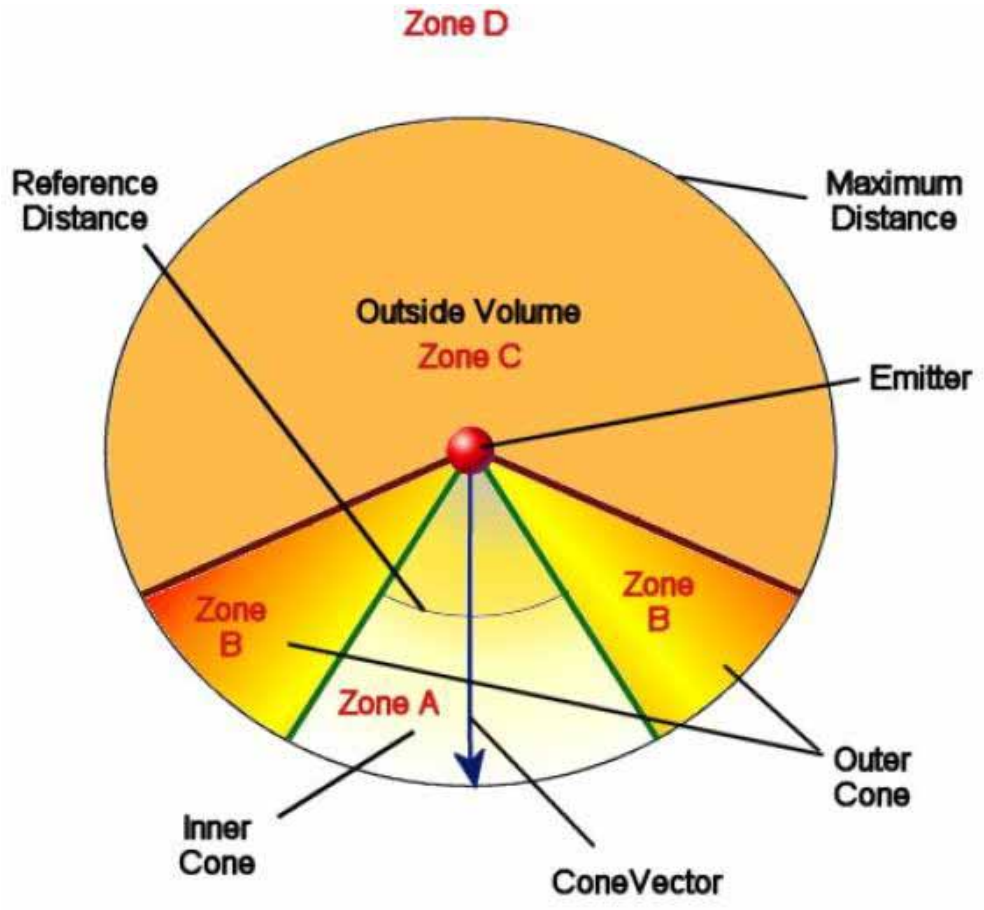


图 3.9　关于声锥的描述图

Precipitation

Precipitation 用来创建类似雨、雪等天气效果。单击 Precipitation 得到如图 3.10 所示的创建 Precipitation 对象的对话框，其中：

Object Name：您的游戏中需要创建的天气效果的对象名字。

Name：用来描述该对象的具体名字。

Precipitation data：使用的数据块。点击右边的方块可以选择可用的数据块。需要预先编写相关脚本代码。

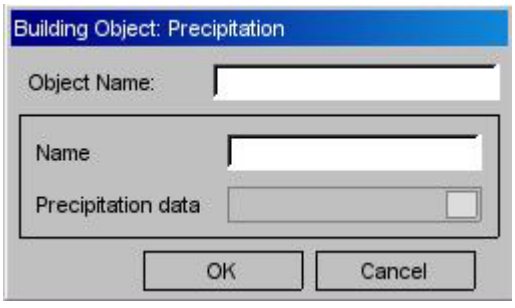


图 3.10　创建 Precipitation 对象对话框

ParticleEmitterNode

ParticleEmitterNode 用来创建粒子发射器的节点。单击 ParticleEmitterNode 得到图 3.11 所示的创建 ParticleEmitterNode 对象的对话框，其中：

Object Name：您的游戏中需要创建的对象的名字。

datablock：使用的数据块，点击右边的方块可以选择可用的数据块。需要预先编写相关脚本代码。

Particle data：从该节点发射出去的粒子所使用的数据块。点击右边的方块可以选择可用的数据块。需要预先编写相关脚本代码。

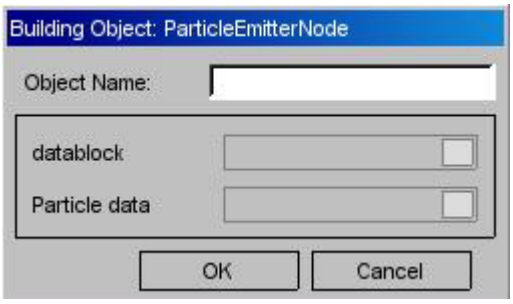


图 3.11　创建 ParticleEmitterNode 对话框

fxSunLight

fxSunLight 用来创建一些同太阳特效相关的的效果。单击 fxSunLight 得到如图 3.12 所示的创建 fxSunLight 对象的对话框，其中：

Object Name：您的游戏中需要创建的 fxSunLight 对象的名字。

图 3.12　创建 fxSunLight 对象对话框

fxShapeReplicator 和 fxFoliageReplicator

fxShapeReplicator 和 fxFoliageReplicator 都是用来在一个指定的区域面积内，使用随机的方式创建大量分布的对象的，如图 3.13 的创建对话框所示，其中：

fxShapeReplicator 用来创建大量分布的 Shapes 对象，需要为其指定 Shapes 对象；而 fxFoliageReplicator 用来创建植物分布效果的对象，只需要为其指定图片对象即可。具体采用何种方式由您自己决定。

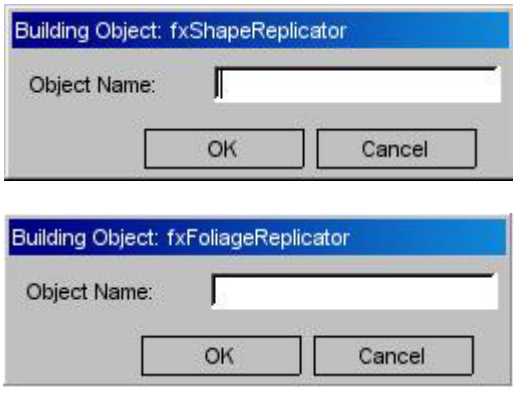


图 3.13 创建 fxShapeReplicator 和 fxFoliageReplicator 对象对话框

fxLight

fxLight 用来创建特殊光照效果。点击它得到如图 3.14 所示的创建 fxLight 对象的对话框，其中：

Object Name：您的游戏中需要创建的 fxLight 对象的名字。

fxLight Data：使用的数据块。点击右边的方块可以选择可用的数据块。需要预先编写相关脚本代码。

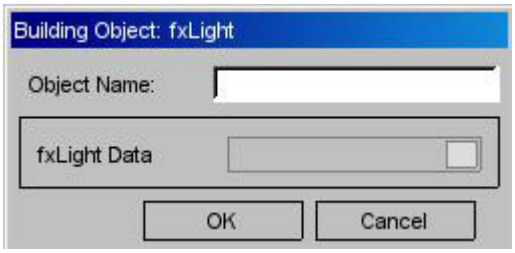


图 3.14　创建 fxLight 对象对话框

sgLightObject

sgLightObject 用来创建更多的复杂的光。点击得到如图 3.15 所示的创建 sgLightObject 对象的对话框，其中：

ObjectName：您的游戏中需要创建的 sgLightObject 对象的名字。

sgLightObjectData：使用的数据块。引擎为我们提供了许多可供使用的数据块。如 sgUniversalStaticLightData 等。

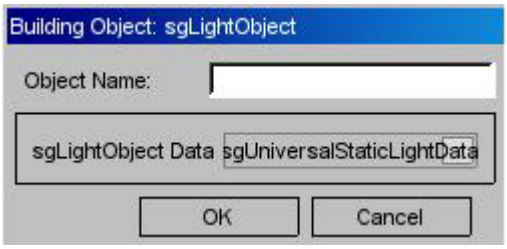


图 3.15　创建 sgLightObject 对象对话框

VolumeLight

VolumeLight 用来创建游戏中一些类似与从物体发出的光（或反光）。点击它得到如图 3.16 所示的创建 VolumeLight 对象的对话框，其中：

ObjectName：您的游戏中需要创建的 VolumeLight 对象的名字。

Data block：使用的数据块。点击右边的方块可以选择可用的数据块。需要预先编写相关脚本代码。

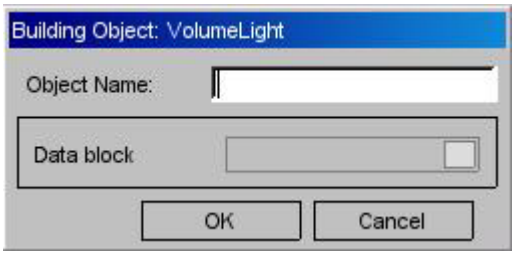


图 3.16　创建 VolumeLight 对象对话框

sgMissionLightingFilter

sgMissionLightingFilter 用来创建光线过滤器对象。该对象的创建对话框如图 3.17 所示的创建 sgMissionLightingFilter 对象的对话框，其中：

Object Name :您的游戏中需要创建的 sgMissionLightingFilter 对象的名字。

sgMissionLightingFilter：使用的数据块。点击右边的方块可以选择可用的数据块。需要预先编写相关脚本代码。

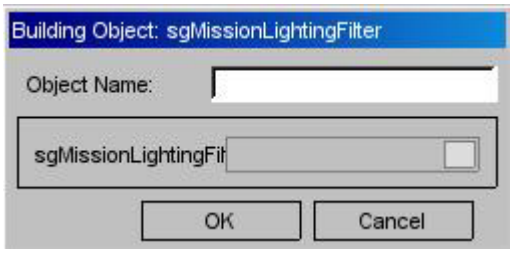


图 3.17 创建 sgMissionLightingFilter 对话框

sgDecalProjector

sgDecalProjector 用来创建投影图像的效果，如远处的山等效果。该对象的创建对话框如图 3.18 所示的创建 sgDecalProjector 对象的对话框，其中：

Object Name：您的游戏中需要创建的 sgDecalProjector 对象的名字。

DecalData Data：使用的数据块，点击右边的方块可以选择可用的数据块。需要预先编写相关脚本代码。

图 3.18 创建 sgDecalProjector 对象对话框

Mission

Mission 中的对象是同任务相关的对象，相对简单一些。其中：

MissionArea：用来创建任务区域，由于游戏已经为我们创建了默认的任务区域，我们只可以对其进行修改，不能在创建新的任务区。

Path：创建路径。

PathMarker：创建路径的标记点。

Trigger：创建触发器。

Physical zoon：创建一个物理区域。

Camera：创建一个摄像机对象。

System

System 中只有 simGroup 对象，相信您已经非常熟悉它的功能了。

Mission Area Editor

Mission Area Editor：任务区域编辑器，该状态下，您会在屏幕右上方看到如图 3.19 所示画面。

窗口中的红色矩形包围的是我们定义的游戏任务区域，在游戏世界中会用红色的矩形墙表示；另外一个呈三角装线条代码目前我们的控制的对象（玩家或飞翔的摄像机）在游戏世界中的位置，随着移动，该线条也做相应移动，需要注意的是，图中箭头指向的方向是我们视线的方向。

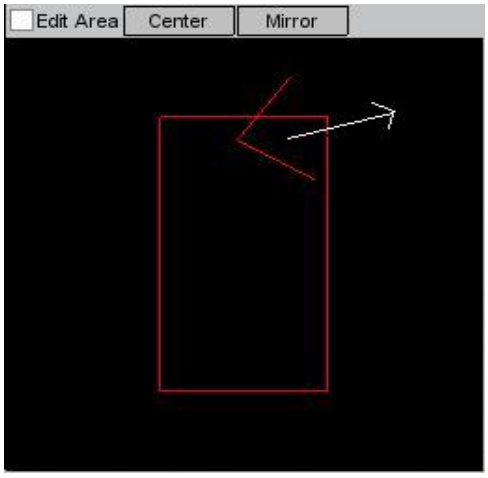


图 3.19　Mission Area Editor 界面

此外，选中 Edit Area 前面的复选框，在窗格的任务区域矩形四周将出现 8 个小方块，通过拖动它们即可以改变任务区域的大小。

Center 按钮会将你的任务区域的中心同世界的中心对齐。

Mirror 按钮允许你为你的任务区域设置对称，这对于平衡游戏有重要意义。

Terrain Editor：地形编辑器，该选项用来编辑你的游戏世界地形。同时，你会发现在菜单栏中会多出 ***Action*** 和 ***Brush*** 两个选项。下面我们先学习一下这两个选项的作用。***Action*** 菜单的内容如表 3.6 所示：

表 3.6　Action 菜单的内容

选　项	说　明
Select	选择地形上的网格点
Adjust Selection	在当前选中的网格点上，通过向上或向下拖动鼠标来升高或降低地形，配合 Select 使用
Add Dirt	在画笔的中心位置逐渐添加“泥土”，提高所影响地形的高度
Excavate	从画笔中心逐渐减少“泥土”，降低所影响地形的高度
Adjust Height	拖动鼠标升高或降低以画笔标记的区域，类似与 Adjust Selection
Flatten	设定由画笔标记的区域至一个平面高度
Smooth	使画笔标记的区域光滑
Set Height	设置画笔标记的区域至一个固定高度，使用 Terrain Editor Settings 来设置高度
Set Empty	在画笔覆盖的正方形地形上挖个洞
Clear Empty	填平画笔覆盖的正方形地形上的所有洞
Paint Material	用画笔绘制当前地形纹理材质

Brush 菜单中可用功能如下表 3.7 所示：

表 3.7　Brush 菜单的内容

选　项	说　明
Box Brush	使用一个正方形画笔
Circle Brush	使用一个圆形画笔
Soft Brush	设置画笔的硬度，绿色表示影响最弱，红色表示影响最强。 Terrain Editor Settings 的 Filter 控制
Hard Brush	设置画笔为硬画笔，所有方块都是红色
Size1×1 to 25×25	设置画笔的尺寸，单位为 Terrain Editor Settings 的<Radius>

了解上述功能后，尝试着在你的游戏世界里进行一些改变地形的操作，比如创建几座大山，是不是很有趣？

接下来让我们看看 ***Terrain Terraform Editor*** 选项。

Terrain Terraform Editor 使用数学算法来生成地形高度。高地的运算在堆栈中进行，它根据运算结果生成新的地形，通过 Apply 按钮接受新地形。请允许我使用两幅图片分开描述 ***Terrain Terraform Editor*** 的框架，如图 3.20 所示：

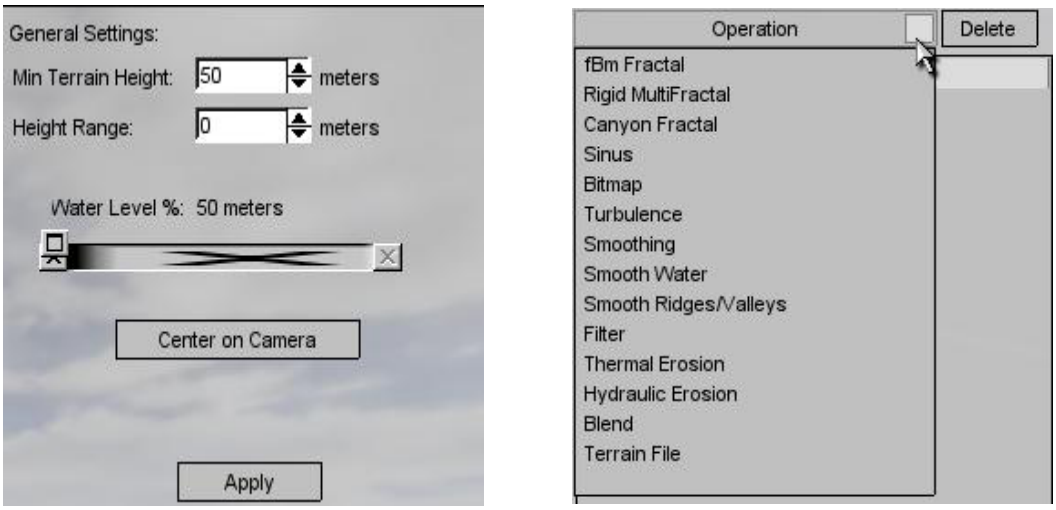


图 3.20　Terrain Terraform Editor 的内容

其中：

Min Terrain Height：定义了地形最低点的海拔高度，单位为米；

Height Range：定义了最大高度和最低高度间的距离，单位为米；

Water Level%：设置水的透明属性；

Operation 下拉菜单中的选项是一些地形算法。

注意，图片上被遮住的第一个操作总是 General 操作，并且不可删除（删除操作无效，不用担心误操作）。

下面是运算的相关说明。

表 3.8　Operation 菜单的内容

选　项	说　明
fBm Fractal	创建起伏的山脉
Rigid MultiFractal	创建山脊和绵延的山谷
Canyon Fractal	创建垂直的峡谷山脊
Sinus	创建交错的正弦波形，用于创建起伏的山脉
Bitmap	导入创建好的 256×256 象素的位图作为高度图
Turbulence	扰乱堆栈中另外一个操作的效果
Smoothing	平滑堆栈中另外一个操作的效果
Smooth Water	使水面平静
Smooth Ridges/Valleys	平滑在边缘边界的现有操作
Filter	将过滤应用到一个基于曲线的现有操作
Thermal Erosion	使用热侵蚀算法计算效果
Hydraulic Erosion	使用水侵蚀算法计算效果
Blend	根据比例因子和算术运算符将两个现有操作混合
Terrain File	加载现有地形文件至堆栈

Terraform 运算为我们提供了大量的地形效果，但是我认为在我们自己创建的游戏中基本上不会用到这些运算，除非你的游戏发生在绵延的山地！

Terrain Texture Editor

Terrain Texture Editor 使用数学方法将基于高地的地形纹理放置在 ***terraformer*** 高地堆栈的底部。通过 Add Material 按钮添加纹理。它可以在游戏文件夹目录下查找任何纹理（.png 或.jpg 格式），一般我们会把地形纹理图片放在 *data\terrain*文件夹里。选中一个纹理后，可以选择它的放置方式。如下表 3.9 所示：

表 3.9　Terrain Texture Editor 菜单的内容

选　项	说　明
Place by Fractal	基于布朗运动方式在地形上放置纹理
Place by Height	基于高程过滤器放置纹理
Place by Slope	基于坡度过滤器放置纹理
Place by Water Level	基于 terraform Editor 中的水平面参数放置纹理

上述方式提供了较方便的地形纹理操作，不过，***Terrain Texture Painter*** 将提供更灵活方便的操作。

Terrain Texture Painter：地形纹理绘制器，该地形纹理绘制器为我们提供了一个非常方便的处理游戏世界中的地形纹理的方法，它为我们提供了最多可以使用 6 种不同的纹理应用在同一个地形上，通过点击“ add… ”按钮即可添加新的地形纹理，如图 3.21 所示。新添加的纹理通过 brush 的操作可以直接加在原有的纹理之上，并覆盖它。而边缘部分，Torque 引擎会自动进行模糊处理，使它看起来过渡得非常平滑，而不会让人感到十分突兀。

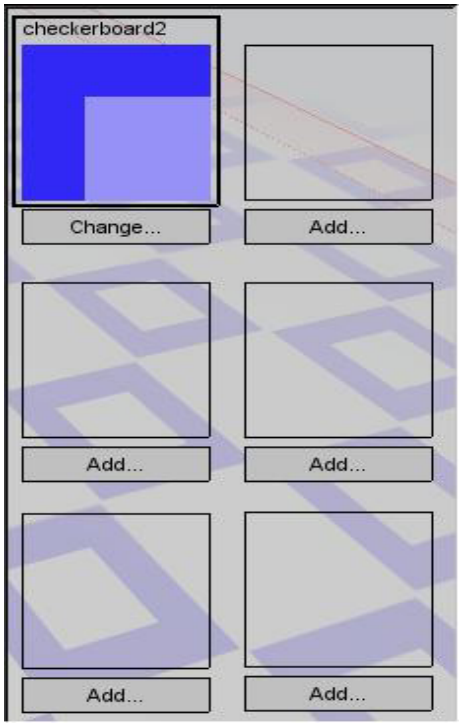


图 3.21　Terrain Texture Painter 编辑界面

尝试一下，改变现有的地形纹理（蓝白棋盘格）为 grass.jpg（在 data\terrain\），现在游戏地面变成了绿色的草地，哈，是不是漂亮多了？

Lighting Tools 菜单

Lighting Tools 菜单是 Torque1.5 及其以上版本加入的光照包，下面包含 3 个选项，如表 3.10 所示：

表 3.10　Lighting Tools 菜单的内容

选　项	说　明
Light Editor (F12)	光照编辑器，可以创建新的光照数据类
Filtered Relight (Alt F)	重新渲染游戏场景，不包括 Interior 对象
Full Relight (Alt L)	重新渲染全部场景

好了，关于游戏自带的世界编辑器的全部内容我们就都了解了，是不是一点都不难，还很有趣？熟悉了 Torque 的编辑器之后，您就可以在您的游戏世界里任意布置您的场景了，可能的话，现在就实践一下吧。

3.2 GUI Editor

GUI 是 Graphical User Interface 的简称，即图形用户接口，通常人机交互图形化用户界面设计经常读做“goo-ee”，准确来说 GUI 就是屏幕产品的视觉体验和互动操作部分，在游戏中作为用户与游戏的交互接口。相信您对这个概念并不陌生。

GUI Editor（图形用户界面编辑器）是 Torque 内建的另外一个非常重要的编辑器。它为我们提供了创建和编辑 GUI 元素的功能。在编辑的游戏过程中，您可以随时使用“ F10 ”按键进入 GUI Editor 状态。如下图 3.22 所示：

图 3.22　GUI 编辑器的界面

GUI Editor 的界面同 World Editor 的界面相似，包括菜单选项、控件选项、游戏窗口以及树型结构的资源管理器。

3.2.1 菜单选项

File 菜单

File 菜单的内容非常简单，包括 4 个选项，如表 3.11 所示：

表 3.11　File 菜单的内容

选　项	功　能
New GUI	用来创建新的 GUI 元素
Save GUI	保存当前的 GUI 元素
GUI Editor Help(F1)	打开帮助窗口
Toggle GUI Editor(F10)	在 GUI Editor 和游戏中切换

Edit 菜单

Edit 菜单包括以下 4 个选项，如表 3.12 所示：

表 3.12　Edit 菜单的内容

选　项	功　能
Cut(Ctrl+X)	剪切
Copy(Ctrl+C)	复制
Paste(Ctrl+V)	粘贴
Select all(Ctrl+A)	全选

Layout 菜单

表 3.13　Layout 菜单的内容

选　项	功　能
Align Left(Ctrl+L)	沿所有被选中控件的最左点对齐控件左边
Align Rignt(Ctrl+R)	沿所有被选中控件的最右点对齐控件右边
Align Top(Ctrl+T)	沿所有被选中控件的最顶点对齐控件顶边

Align Bottom(Ctrl+B)	沿所有被选中控件的最低点对齐控件底边
Center Horizontally	使所选控件水平对齐于绑定所有被选控件的矩形中心
Space Vertically	垂直平均分隔所有被选控件
Space Horizontally	水平平均分隔所有被选控件
Bring to Front	使所选控件置于其它控件之前
Send to Back	使所选控件置于其它控件之后
Lock Selection	锁定选中控件
Unlock Selection	解锁选定控件

Move 菜单

Move 菜单主要用来控件的移动操作，具体如下表 3.14 所示：

选　项	功　能
Nudge Left (←)	使选中的控件左移 1 个象素
Nudge Right (→)	使选中的控件右移 1 个象素
Nudge Up (↑)	使选中的控件上移 1 个象素
Nudge Down (↓)	使选中的控件下移 1 个象素
Big Nudge Left (shift+←)	使选中的控件左移 10 个象素
Big Nudge Right (shift+→)	使选中的控件左移 10 个象素
Big Nudge Up (shift+↑)	使选中的控件左移 10 个象素
Big Nudge Down (shift+↓)	使选中的控件左移 10 个象素

3.2.2 控件选项

GUI Editor 的所有控件都在这里提供。通过它们，我们可以制作出各种各样效果的 GUI 元素。通过点击 New Control 控件，我们可以看到它包括许多各种类型的控件。这里面我们将选择一些经常会使用的控件进行说明。

GuiChunkedBitmapCtrl

GuiChunkedBitmapCtrl 类控件通常用做菜单界面的大背景，该类控件的名字

来源于一个图像显示增强概念，就是把一个图像分离成若干较小图像（分片位图）以提高显示速度。下面是一个 GuiChunkedBitmapCtrl 定义的示例：

```
new GuiChunkedBitmapCtrl(MainMenuGui) {
    Profile = "GuiContentProfile";
    HorizSizing = "width";
    VertSizing = "height";
    position = "0 0";
    Extent = "640 480";
    MinExtent = "8 8";
    Visible = "1";
    bitmap = "./background";
        helpTag = "0";
// insert other controls here
};
```

该控件的名字是 MainMenuGui，它的 profile 类型为 GuiContentProfile，然后定义了 HorizSizing 和 VertSizing 属性，接着指定了位置信息，position 为该控件指定了起始位置坐标，Extent 指定了该控件的覆盖范围，MinExtent 是该控件的最小覆盖范围，它用来确定 Torque 内置的 GUI Editor 分割控件的最小尺寸。Visible 指定了该控件是否可见，1 为可见，0 为不可见。Bitmap 指定了加载的位图文件。

// insert other controls here 注释行表明 GuiChunkedBitmapCtrl 控件可以包含其它控件，这些控件都是它的子元素。

GuiControl

GuiControl 类控件是一种通用的控件容器。它通常用作选项卡型的容器，也常被其它系统称为框架（frame）。用该控件可以把其它控件集聚为一个整体，作为一起群组对其进行操作。下面是一个 GuiControl 定义的示例：

```
new GuiControl(InfoTab) {
    Profile = "GuiDefaultProfile";
    HorizSizing = "width";
```

```
    VertSizing = "height";
    position = "0 0";
    Extent = "640 480";
    MinExtent = "8 8";
    Visible = "1";
};
```

所有的属性同 GuiChunkedBitmapCtrl 中对应的属性用法和含义都相同。

GuiTextCtrl

GuiTextCtrl 类控件是一种直接而常用的控件。用户可以通过它来显示任何需要的文本，也可以放在一个无文本的界面之上，随游戏的进度再填入文本。下面是一个 GuiTextCtrl 定义的示例：

```
new GuiTextCtrl(PlayerNameLabel) {
    Profile = "GuiTextProfile";
    HorizSizing = "right";
    VertSizing = "bottom";
    position = "200 5";
    Extent = "64 16";
    MinExtent = "8 8";
    Visible = “1”;
    Text = “Player Name:”;
    maxLength = “255”;
};
```

控件的大部分属性同上面的相同。可以选择 profile 来确定文本字体和其它特征，常用的如 GuiBigTextProfile，用大字体显示文本。也可以很容易地改变文本的内容，其操作如下：

```
PlayerNameLable.text = “Your Name”;
```

其中 PlayerNameLable 就是该控件的名字，因此使用类似的方法可以任意修改控件的属性值。

maxLength 属性允许用户存储在控件内的字符数目，字符数应该尽量少以节

省内存。

GuiButtonCtrl

GuiButtonCtrl 控件从名字就可以看出来，它是一种可以单击的按钮控件。它一般用作命令界面控件，用户单击该控件类可以迅速实现动作命令请求。下面是一个 GuiButtonCtrl 定义的示例：

```
new GuiButtonCtrl() {
    Profile = "GuiButtonProfile";
    HorizSizing = "right";
    VertSizing = "top";
    position = "200 50";
    Extent = "127 31";
    MinExtent = "8 8";
    Visible = “1”;
    Command = “Canvas.getContent().Close();”;
    Text = “close:”;
groupNum = “-1”;
buttonType = “PushButton”;
};
```

最重要的属性是 command 属性，它包含一个按下按钮后即将执行的脚本语句。

另一个重要属性是 buttonType 属性。其类型如下：

ButtonTypePush

ButtonTypeCheck

ButtonTypeRadio

当 buttonType 特指为 ButtonTypeRadio 时会用到 groupNum 属性，当界面屏幕中出现 groupNum 时，单选按钮通常以独立的形式出现。只有最近按下的单选按钮才会被设为启用状态（true），群组中的其它单选按钮将会被禁用。在其它方面，单选按钮类控件与 GuiCheckBoxCtrl 类控件的作用相同，后面将会对其进行介绍。

GuiCheckBoxCtrl

GuiCheckBoxCtrl 控件是一种 GuiButtonCtrl 类控件的特定变体，用来保存当前状态值。这种类型的控件我们在游戏中会经常看到。它通常为我们提供一个选择框，通过选中或清除该选择框的“ 选中 ”状态，来实现我们的需求。下面是一个 GuiCheckBoxCtrl 定义的示例：

```
new GuiCheckBoxCtrl(IsMultiplayer) {
    Profile = "GuiCheckBoxProfile";
    HorizSizing = "right";
    VertSizing = "bottom";
    position = "200 5";
    Extent = "64 16";
    MinExtent = "8 8";
    Visible = “1”;
    Variable = “Pref::HostMultiPlayer”;
    Text = “Multiplayer”;
    maxLength = “255”;
};
```

如果用户指定 variable 属性，那么单击该控件后，所指定的变量值将会设为控件的当前状态值。如果该控件开始处于显示状态，它会根据指定变量中的数值来设置自身状态。用户需要确认所用变量包含适当的数值。对复选框应用 text 属性，可以指定复选框旁边显示的文本标签。可以看到，GuiRadioCtrl 控件与本控件作用类似，所不同的是在 GuiRadioCtrl 同一组中只能选中一个按钮。

GuiScrollCtrl

GuiTextEditCtrl 给用户提供了一个手动输入文本字符串的工具。下面是一个 GuiScrollCtrl 定义的示例：

```
new GuiScrollCtrl() {
    Profile = "GuiScollProfile";
    HorizSizing = "right";
    VertSizing = "bottom";
```

```
        position = "20 50";
        Extent = "580 200";
        MinExtent = "8 8";
        Visible = “1”;
        willFirstRespond = “1”;
        hScrollBar = “dynamic”;
        vScrollBar = “alwaysOn”;
        ConstantThumbHeight = “0”;
        childMargin = “0 0”;
        childMargin = “0 0”;
        defaultLineHeight = “15”;
        // insert listing control here
    };
```

通常我们会把滚动控件做成列表形式，这通常由 GuiTextListCtrl 控件的内容。包含列表的控件将会被添加作为本控件的子元素。

willFirstRespond 属性用来表示是本控件还是其他控件首先执行按下（控制滚动）的方先键命令。

hScrollBar 和 vScrollBar 属性分别指水平条和垂直条，它们一般设为下面几种模式：

alwaysOn 滚动条总是可见

alwaysOff 滚动条不可见

dynamic 列表超出显示空间时滚动条可见

属性 constantThumbHeight 表示，滚动条中随鼠标拖动二移动的小矩形控件——滑块，是随列表中输入数值的大小成比例缩放，或是大小恒定。该属性设为 1 表示是一个恒值，0 则表示可变。

属性 defaultLineHeight 用可见象素数目定义了每一行控件内容的高度，该数值常用来决定按下一次垂直方向键的滚动距离大小。

属性 childMargin 用来限定父控件中的可视空间，也就是控件所包含的滚动列表所占的空间。实际上，它本身就是在滚动控件内创建的一个限制滚动列表放

置的边界。第 1 个值是水平边界（左右），第 2 个值是垂直边界（上下）。

GuiTextListCtrl

GuiTextListCtrl 控件用来显示文本值的 2D 阵列。下面是一个 GuiTextListCtrl 控件的定义示例：

```
new GuiTextListCtrl(MasterServerList) {
    Profile = "GuiTextArrayProfile";
    HorizSizing = "right";
    VertSizing = "bottom";
    position = "2 5";
    Extent = "580 20";
    MinExtent = "8 8";
    Visible = “1”;
    Enumerate = “0”;
    resizeCell = “1”;
    columns = “0 30 200 240 280 400”;
    fitParentWidth = “1”;
    clipColumnText = “0”;
    noDuplicates = “false”;
};
```

属性 enumerate 表示高亮显示的文本行。

属性 resizeCell 为 1（true），表示可以使用 GUI Editor 对单元对象的大小进行变换。

阵列中的每个行或者列都有限定的空间字段，可以用 columns 属性来表明出现每个字段所在的列数，这样就使字段的显示格式化，比较规整。

属性 fitParentWidth 表示子控件本身是否放大以满足其它任一父控件的可显示空间。

通过设置 clipColumnText 属性，我们可以确定每一列的超长文本是剪切掉，或是延伸到临近的列。

把属性 noDuplicates 设为 true 可以自动防止重复显示。

GuiTextEditCtrl

GuiTextEditCtrl 控件给用户提供了一个手动输入文本字符串的工具。下面是一个 GuiTextEditCtrl 定义的示例：

```
new GuiTextEditCtrl() {
    Profile = "GuiTextEditProfile";
    HorizSizing = "right";
    VertSizing = "bottom";
    position = "20 50";
    Extent = "580 200";
    MinExtent = "8 8";
    Visible = “1”;
    Variable = “Pref::Player::Name”;
    maxLength = “255”;
    historySize = “6”;
    password = “0”;
    sinkAllKeyEvents = “0”;
    helpTag = “0”;
};
```

属性 variable 是该控件中最关键的要素。当用户在控件的编辑框中输入一串文本时，字符实际被输入到所给出的变量中。控件显示时首先显示的是输入编辑框的变量内容。

属性 historySize 非常有用，它表现为我们熟悉的历史功能，它会为我们保存所有前面输入的内容，最大值为 historySize 定义的数量，可以用按键↑来调用历史记录，或者使用按键↓来继续下步操作。

属性 password 设置为 true，就可以使用该控件来验证密码。为了保密，控件会自动将用户输入的内容用*号代替。

属性 sinkAllKeyEvents 设为 ture，该控件就会自动消除无法处理的键盘输入信息；如果设为 false，控件将完全接受输入信息。

3.3 本章小结

本章我们学习了 Torque 游戏引擎中的编辑器的内容，包括 World Editor（世界编辑器）和 GUI Editor（GUI 编辑器）。使用 World Editor 可以实现游戏场景搭建的绝大部分内容而不需要进行任何的代码编写工作。使用 GUI Editor 可以为我们的游戏画面制作非常精美的 GUI 控件。用好 Torque 游戏引擎的这两个编辑器将使您的游戏向成功迈进了一大步！下一章，我们将开始学习 Torque Script（脚本），学习 Torque 游戏引擎是如何通过程序脚本是游戏变得生动有趣起来的。

第四章　Torque Script

本章内容学习 Torque Script（脚本）。读完本章的内容并不会使你一下子成为一个优秀的 Torque Script 的精通者，但是，只要您肯花时间进行练习，就一定能够获得成功！Torque Script 为我们提供了许多工具来帮助我们设计自己的游戏，想了解更多的内容，请访问 GarageGames 网站。

4.1 Torque Script 的概念和术语

目前，成熟的游戏引擎都提供被称为 Script（脚本语言）的编程方法。那么，什么是脚本语言？用脚本语言可以做什么？为什么在现代游戏开发中越来越多地使用脚本语言？

脚本语言也是一种编程语言，它可以很好地完成工作，同时使代码的编写变得简单。如果您翻看一下编程语言的历史，您会发下，从难于阅读和编写的低级语言，到便于阅读和理解的高级语言，经历相对艰难的进化发展。例如，汇编语言在 C++语言出现之前使用了很长时间。汇编语言对绝大多数人来讲相当的困难，即使您可以同时使用 C++和汇编语言编写程序，C++通常是更好的选择，它更容易编写、测试、调试和维护。由于自顶向下的工作原理，使得 C++运行速度比汇编语言慢一些。

同样地，脚本语言同类似 C++语言相比具备同样的优点，它可以看作是编程语言的另外一种进化。实际上，许多脚本语言看起来同 C++非常相似，您可以直接进行代码的编写，而不用花时间去考虑像数据类型或者内存管理的是如何进行的等底层的原理。同时，脚本语言允许您直接修改您的代码，而不需要重新编译。这是脚本语言的主要优点之一。缺点也是显而易见的，脚本语言没有 C++运行速度快！

综上所述，很容易明白为什么现代游戏开发中更多地使用脚本语言。事实上，绝大多数游戏程序员都采纳这个观点：优先考虑使用脚本语言编写代码，除非绝对需要才使用 C++代码编写。这种观念同选择 C++语言和汇编语言一样，绝大多数情况使用 C++语言，除非绝对需要，才使用汇编语言。明白了么？

上述是总体的观点，下面将从难易度、灵活性和效率等方面介绍脚本语言的优势。

Prototyping：在您开发游戏过程中，经常会有新的想法迸出的，例如游戏增加了新特征，或者是起初的设计已经无法适应当前的情况必须作出修改甚至替换。为了提高效率，您需要迅速思考这些新的理念，并决定是否保留还是修改它们。我们称之为“ prototyping ”。使用脚本语言非常使用这种情况，因为它可以快速地编写代码、测试和修改。

Debugging：脚本语言调试非常容易。脚本语言非常容易修改，你可以找到问题，改写它们，然后重新测试而完全不用重新编译源代码。

Game customization and tweaking：游戏界面的外观和给人的感觉通常在游戏开发过程当中发生不断的变化。因此，同这些关联的代码最好好使用脚本实现。这样非常方便进行不同风格的测试。此外还有其它的好处，如允许您的玩家自定义游戏的风格和按键（当然，这完全取决于您在开发游戏到时候是否给玩家这样的权力）。

Writing non-performance-critical functionality：脚本语言可以写许多函数，虽然用它来写渲染管道不是一个好主意，但是对于 GUI 响应和绝大多数游戏功能函数是个不错的选择。

具备脚本语言是现代游戏引擎的一个强有力的特征之一，从分利用脚本语言的优势开发游戏是游戏程序员必须掌握的正确技能！

好了，现在您已经了解脚本语言非常有用了，那么如何使用脚本语言？它究竟为我们提供了多少功能？下面是一个游戏脚本语言应该具备的特征。

基本编程语言特征：脚本语言具备所有现代编程语言的基本特征，如变量类型，基本操作（加法、减法等），标准控制语句（if-then-else, for, while 等）和子程序（函数，子文件等）。

引擎结构接口：这是非常重要的特征，在游戏脚本语言的环境下，必须为其提供可以对引擎核心功能操作的接口，该脚本系统允许访问渲染、声音、物理、AI（人工智能）和 I/O（输入输出）系统，还必须可以进行创建和删除对象，以及定义新函数的操作。

其它非常好的特性：

相似和一致的语法构成：这指的是脚本语言的语法结构应该同绝大多数程序员熟悉的语言相似，如 C 或者 C++语言，同时保持相同的编写规则。

面向对象功能：面向对象编程是软件工程领域的一场革命。支持面向对象的脚本语言具有许多优点：

封装：提供对代码和数据限制访问的方法

继承：提供创建新对象的方法

多态：不用考虑对象默认的行为，无论该对象继承与引擎对象还是脚本对象。

“On-demand”加载函数：为什么不将所有代码载入内存？除了能够节省内存之外，脚本语言允许动态加载和卸载函数，这使得程序的运行十分流畅。

提高脚本运行速度：脚本代码通常不被编译，它们通常在运行时候被解释。一种许多常用脚本语言（PERL，TCL，VB Script，Java）的特征是具备将脚本语言编译程 pcode 的能力。这种 pcode 在虚拟机上执行。它的好处是文件大小和运行速度。Pcode 通常体积更小并且执行的更快。

4.2 关于 Torque Script

好了，现在回过头来看看 Torque 的脚本语言。Torque Script 是一种类似 C++语言的非常强大、灵活的脚本语言。它具有上面提到的全部优点，下面让我们具体来看看 Torque Script。

前面已经介绍过了，Torque Script 非常类似于 C++语言，它们之间只存在很小的一些差别，关于 C++语言的学习已经有很多著名的教材可供使用，因此，这里就不进行基本的句法编程介绍了，相信对于有一定编程基础的读者不会存在任何问题。如果非常不幸，您是一个初学者，那么花点时间看看 C++编程绝对不是坏事。好了，下面让我们一起来看看 Torque Script 一些特别的地方。

4.2.1 变量

Torque Script 中的变量有两种，全局变量和局部变量。局部变量在离开它的作用域后会被自动清除。比如一个函数。我们在函数中定义了一个局部变量，那么，只要该函数运行结束，这个局部变量就会马上被清除掉。这时，我们称变量

已经离开它的作用范围。全局变量则在整个程序中都有效。

Torque Script 专门为全局变量和局部变量定义了标识符，%定义局部变量，$定义了全局变量，很容易区分。与 C++代码不同，变量不被强制定义类型，而且在使用变量之前也不必预先声明。如果您在对变量赋值之前读取这个变量的值，得到的将是一个空字符串或者 0，具体情况要看把这个变量用作字符串变量还是数字变量。

您可以使用任何您喜欢的名字为变量命名，通常情况下使用英文单词或者缩写。需要注意的是，变量的命名要遵循如下规则：

不能是 Torque Script 的保留关键字

必须以字母开头；

只能由字母、数字和下划线（_）组成。

保留关键字是 Torque Script 中有特殊含义的合法标识符。表 4.1 中给出了关键字列表。出于 Torque 标识符的考虑，下划线被当作是一个字母数字标识符。下面是一些有效的标量标识符：

isClose Today x item_1 NOW

下列是不合法的标识符：

5input kilo-per-min function true +level

表 4.1 Torque Script 保留关键字

关键字	说 明
Break	中断循环的执行
Case	在 switch 表示一种选择
Continue	使循环从头开始继续执行
Default	在 switch 块中表示没有任何选择可以匹配的情况
Do	表示 do - while 类型循环块的开始
Else	表示 if 语句的另外一个执行路径
False	逻辑“假”，值=0
For	表示 for 循环的开始
Function	表示其后面的代码块是一个可调用的函数（方法）
If	表示一个条件（比较）语句的开始

New	创建一个新的对象数据块
Return	表示从一个函数返回
Switch	表示 switch 选择语句的开始
True	逻辑“真”，值=1
while	表示 while 循环的开始

4.2.2 字符串

字符串常量由单引号或者双引号包含。单引号包含的字符串被称为 tagged string（标记字符串），这是一种需要通过网络连接进行传输的特殊字符串类型。在最开始的时候，计算机会一次发送整个字符串的数据，并且每个标记字符串都由引擎自动生成一个独一无二的专用数字 ID 进行标记。接下来，无论以后任何时候需要再次使用这个字符串，计算机所发送的内容仅仅是用于标记这个字符串的标记，也就是专用数字 ID，这样会大大减少游戏对带宽的消耗。使用 detag() 命令可以得到该标记的 ID 值。

双引号字符串在 Torque Script 中称为标准字符串，这是没有加标记的。因此，无论什么时候用到字符串，对用到字符串的任何操作都必须为包含在字符串中的所有字符串分配存储空间。如果通过网络连接传送一个标准字符串，那么每一次都必须传送字符串中的所有字符。这里要说明一下消息字符串，消息字符串是被作为标准字符串进行传送的，因为这些消息在每次传送的时候基本上都会发生变化，所以，为对话的消息创建标记 ID 号就显得作用不大了。当然，如果想许多游戏那样为玩家提供一些如“大家好”等常用的对话内容，完全可以考虑使用标记字符串来提高效率。

同 C++语言类似，字符串中可以包含格式化代码，如表 4.2 所示。

表 4.2　格式化代码的内容

代　码	说　明
\n	嵌入一个新行符
\r	嵌入一个回车符

\t	嵌入一个制表符
\c0…\c9	用 n 作为索引引用由 GUIContrlProfile.fontColors 定义的颜色表中的颜色
\cr	恢复显示的色彩为默认值
\cp	将当前颜色压入颜色堆栈
\co	从颜色堆栈弹出颜色
\xhh	在 x 后嵌入表示的十六进制 ASCII 自如
\\	嵌入一个反斜杠

4.2.3 对象

对象是对象类的实例，它是一组属性和方法的集合体，这些属性和方法定义了对象的行为和特征。Torque 中的对象是对象类的实例。在 Torque 中创建对象之后，该对象就具有唯一的数字表示，即在 world editor（世界编辑器）中看到的每个对象的数字 ID，我们称之为句柄。如果两个句柄变量具有相同的数字值，则表明它们指向的是同一个对象。

当对象存在于一个含有单个服务器和多个客户机的多万家游戏环境中时，服务器和每个客户机都要为对象在内存的存储空间中分配自己的句柄。

注意：方法是通过对象来访问的函数。不同的对象类完全可以有公共的方法，同时也有属于自己的方法。实际上，不同的类可以具有名称相同的方法，但是如果使用的对象不一样，方法的行为也会完全不同。属性是属于特定对象的变量，同方法一样，可以通过对象对其进行访问。

创建对象

在创建一个新的对象时，可以在 new 语句代码块中初始化对象的各项属性，如下例所示：

```
%h = new InteriorInstance () {
    Position = “0 0 0”;
    Rotation = “1 0 0 0”;
    interiorFile = %name;
```

};

上述代码表名在世界位置（0，0，0）的地方创建一个 interior 对象，而这个建筑物对象的模型位置为%name 变量指定的文件，同时设置它的旋转为（1 0 0 0），前三个参数表明是否绕 X、Y、Z 轴旋转，1 表示是，0 表示否，第四个参数表示旋转的角度值。我们这里表示绕 X 轴旋转 0 度，其实也就不做任何旋转操作。当我们在 World Editor Creator 中为游戏添加建筑物的时候，还记得么？当你在创建列表点击一个想要添加到场景的对象时候，我们会直接在游戏窗口看到它，而这段代码就会自动添加的.mis 文件当中。唯一的区别就是在 World Editor Creator 中是直接创建对象，而没有将产生的句柄（ID 值）赋给%h。如果您改变对象的位置和旋转属性，则代码中的属性会自动改变。

当该对象创建时，新创建的 InteriorInstance 对象的句柄（ID 值）被赋值给了%h 变量。当然，您可以使用任何您喜欢的名字为该变量命名，只要这个名字是合法的并且还没有被用到，如%obj、%building 等。需要注意的是%h 变量是个局部变量，因此它仅在有限的范围内有效。一旦内存被分配给了新的对象，引擎就会按照嵌入在 new 代码块中的语句初始化对象的各项属性，没有初始化语句的属性使用对象类的默认值。一旦获得了对象的唯一句柄，如%h，就可以使用它来使用对象了。

使用对象

要使用或者控制对象，可以通过对象的句柄访问它的属性和方法（函数）。如果对象的句柄保存在局部变量%h 中，就可以使用下面的方法来访问对象的属性：

%handle.aproperty = 10;

句柄并不是用于访问对象的唯一方式。可以通过为对象赋予一个名称来访问它。对象可以用字符串、标识符以及包含字符串或标识符的变量来命名。例如，如果需要使用的对象被命名为 MyObject，那么下列 4 个代码段（A、B、C、D）的功能是完全一样的。

A

MyObject.aproperty = 10;

B

“MyObject”.aproperty =10;

C

%objName = MyObject;

%objName.aproperty = 10;

D

%objName = “MyObject”;

%objName.aproperty = 10;

以上例子说明了访问对象属性的各种方法;您可以按照同样的方式来调用对象的方法（函数）。

注意：这个对象的名称，MyObject，是一个字符串，而不是变量。在这个标识符的前面没有表明变量的%或是$前缀。

对象方法（函数）

您可以这样调用一个通过对象引用的方法（函数）：

%h.function(10, “arg1”, “arg2”);

方法（函数）function 也可以作为%h 所指的对象方法来引用。在上面的例子中，名为 function 的方法（函数）将被执行。在脚本语言中可以有多个被命名为 function 的方法（函数）存在，但是每个方法（函数）必须属于不同的的命名空间（name space）。

当对象的方法（函数）被调用的时候，传递的第一个参数是指向拥有此方法（函数）的对象的句柄。因此，在上面的例子中，function 方法（函数）的定义在参数列表中实际上有 4 个参数，其中第一个是%this 参数。而在上述书写中仅仅使用了 4 个参数中后面的 3 个。实际上当调用函数时，与%this 参数相对应的第一个参数会被自动插入。按照惯例，这个变量名通常用于在方法（函数）中表示拥有该方法（函数）的对象的句柄。这里，建议您在书写方法的时候将%this 加上，这是个不错的习惯。

Toeque 为我们提供了许多对象类的基本方法（函数），见附录，您可以直接通过对象的句柄来调用这些方法。同时，您也可以为了某种需求而通过 Torque Script 自定义其它的方法（函数）。

注意：很遗憾，对于我们初学者来说，完全掌握对象类的方法是一件非常令

人头痛的事情。很多时候，当我们需要一个对象的方法（函数）的时候，我们根本不清楚引擎是否已经为我们提供了这样的方法（函数），还是需要我们自己使用 Torque Script（脚本）去创建该方法（函数）。这里，我推荐您在游戏控制台中，使用“ 对象句柄.dump() ”命令，该命令将会为您打印出该对象所有可用的方法（函数）。这个命令相当有帮助，希望您能熟练使用它。

4.2.4 数据块

数据块是一种特殊的对象，这个对象包含一组特征，这些特征用于描述另一个对象的属性。数据块对象同时存在于服务器以及和它相连接的客户机上。无论是在服务器还是客户机上，每个特定的数据块副本都使用同样的句柄。数据块是 Torque 引擎的重要组成部分。通过数据块可以快速的定义各种游戏物品的公共数据，并且能提高游戏运行速度。Torque 引擎预先定义了一些数据块，游戏制作者可以通过他们迅速制作出许多游戏物品。当然，游戏制作者也可以创建新的数据块。

Torque Script 中的数据块使用 NameData 这种方式命名。ItemData、PlayerData 和 VehicleData 都是以这种方式命名的例子。数据块也是对象，但是为了避免在语义上和普通的对象混淆，在提到它们的时候，我们通常不显式地称它们为对象。

注意：当我们定义好了我们需要的数据块后，如 PlayerData，则我们使用类似如下方法创建我们的对象实例：

```
%vehicle = new Vehicle (vehicleName)
{
……
}
```

其中 new 命令将会按照后面的类名（这里是 vehicle）创建对象，而对象的内容全部来自于实现定义的数据块（这里比如 vehicleName），然后返回一个创建好的实例的句柄给%vehicle 变量，这样我们就可以通过%vehicle 对该对象进行一系列操作了。

一个 VehicleData 数据块包含着许多用于描述数度、质量等许多能够用于 Vehicle 对象的特征。在创建 Vehicle 对象的时候，这个对象会引用一个已经存在

的 VehicleData 数据块来进行初始化，这个数据块将告诉 Vehicle 对象如何做出各种动作。大多数对象在整个游戏过程中都会先被创建然后被删除，但是数据块一旦创建，就不会被删除。

创建数据块

数据块有其自己特有的创建语法。

```
Datablock DataBlockType (Name [:CopySound])
    {
        Category = “CategoryName”;
        [datablock_field0] = Value0;]
        …
        [datablock_fieldM = ValueM;]
        [dynamic_field0 = Value0;]
        …
        [dynamic_fieldM = ValueM;]
    };
Datablock ClassIdentifier (NameIdentifier)
{
    Initialization Statements
};
```

这条语句的值是创建出来的数据块的句柄。其中：

Databolck：关键字，告诉引擎即将创建数据块对象；

DatablockType：必须是一个已经在引擎中定义了的数据块类，如 PlayerData，继承自 GameBaseData 类或者它的子类。

Name：您要创建的数据块的名字，可以使用您任何希望的名字。必须使用有效的标识符。

CopySoure（可选项）：指定了其它某个数据块的名称，所创建的数据块将在执行下面的赋值语句之前从这个数据块中复制域值。指定的数据块必须和所创建的数据块属于同一个类。如果想创建一个和某个预先创建好的数据块几乎一样的数据块（仅有很小的变化），或者想在一个数据块中集中定义一些特征，以便

其它数据块可以反复复制其中的值，那么这种方法是有实用价值的。比如您为您的游戏创建了 enemy 数据块，然后再创建一个 boss 数据块，大部分域值复制 enemy 数据块，而提高其中一些如生命、攻击、防御等属性的域值，看，是不是很方便？

Category：关键字，告诉引擎将该对象放在 World Editor Creator 树型结构的位置。如果还没有该分类名，则会创建它。

Datablock_fieldM：可以为数据块中任何或者全部的属性赋值。

Dynamic_fieldN：我们可以为数据块对象添加引擎没有预先使用 C++代码定义的属性。注意，一点定义了 Dynamic 属性，它们都是静态的。具体内容已经超出本书的范围。

为了加深理解，我们看下面两个例子：

```
Datablock PlayerData (enemySoldier)
{
    Category = “Enemy”;
    ShapeFile = “~/data/shapes/enemy/dnemySoldier.dts”;
    Health = 100;
    Attack = 10;
    Defense = 10;
};
```

这里，我们创建了一个名为 enemySoldier 的 PlayerData 数据块，指定分类名为 enemy，这样您就会在 World Editor Creator 的树型结构中的 shapes 下找到 enemy 文件夹，里边包含了 enemySoldier 对象。点击它就能在游戏世界中添加一个 enemy 对象实例。接着通过 ShapeFile 告诉引擎该数据块对象的模型文件的位置。最后为该数据块对象添加 3 个动态属性生命（100）、攻击（10）和防御（10）。注意，PlayerData 数据块在引擎中定义了许多域名并且指定了默认值，如果我们不初始化它们，将会使用默认值。 通过上述方法，我们就得到了一个关于游戏中敌人士兵的数据块对象。

```
Datablock PlayerData (enemyBoss:enemySoldier)
{
    ShapeFile = “~/data/shapes/enemy/enemyBoss.dts”;
```

```
    Health = 300;
    Attack = 20;
    Defense = 20;
};
```

上述代码我们又创建了一个名为 enemyBoss 的 PlayerData 数据块，该数据块继承自 enemySoldier 数据块。当您再次打开 World Editor Creator 树型结构时，会在 enemy 文件夹下发现增加了 enemyBoss 对象。接着通过 ShapeFile 指定 enemyBoss 使用的模型的位置，显然 Boss 和 Soldier 的模型是不一样的对吧？最后更改了属于 enemyBoss 的属性，生命（300）、攻击（20）和防御（20），哇，强了好多！

注意：在创建新的数据块对象时候，一定不要忘记大括号后面的；（分号），这和函数不一样，因此在代码出现问题时候，首先要检查一下类似最基本的、却往往是最容易忽略的地方。

数据块将是我们在游戏编程中打交道最多的对象，接下来我们将花费较多的篇幅学习 Torque 中预先为我们定义好的数据块。

下面，让我们先学习一些基本的 Torque 中预先定义的数据块，并详细了解一下其中的成员域。

SimDataBlock

SimDataBlock 是 torque 引擎中最基本的数据块。改数据块非常简单，在脚本语言中它几乎没有定义任何数据。但它却是建立更加复杂数据块的基础。包括引擎本身，所有数据块都来源于 SimDataBlock。

例子 1：在 Toruqe 脚本中创建一个 SimDataBlock

本例中，我们将介绍创建一个新的 SimDataBlock 的语法。我们给新建立的数据命名为“ myDataBlock ”，同时再赋予它一个新的名为“ newMemberItem ”的成员。当然，你可以使用任意你喜欢的名字。

```
datablock SimDataBlock(myDataBlock) {
    newMemberItem = "Hello";
};
echo(myDataBlock.newMemberItem);
```

运行该代码，将会在 console（控制台）得到“ Hello ”。更多复杂的数据块都采用类似的定义方式。

注意：任何数据块必须至少有定义一个成员域并且赋予内容，否则数据块将不可用。成员的内容可以是引擎预先定义的，也可以是通过脚本添加的。如果我们没有定义 newMemberItem 项则上边的代码将无法正确执行。

GameBaseData

GameBaseData 是另外一个非常简单却有效的数据块。GameBaseData 仅比 SimDataBlock 复杂一点点。特别地，从任务编辑器中就能够看到 GameBaseData 将创建的数据块。下边是关于 GameBaseData 的介绍，如表 4.2。

表 4.2 GameBaseData 的属性内容

属性	默认类型	描述
Category	字符串	确认基于该数据块的物品的种类
className	字符串	确定数据块的类。内容可以通过脚本改写

例子 2：建立一个 GameBaseData 数据块

本例中，我们将创立一个新的 GameBaseData 数据块，该过程和建立 SimDataBlock 非常相似。

```
datablock GameBaseData(myGBD) {
    category = "Brand New Category";
    newItem = "Brand new field";
};
echo(myGBD.category);
echo(myGBD.newItem);
echo(myGBD.className);
```

运行结果为：

```
Brand New Category
Brand new field
GameBaseData
```

注意：我们没有明确地给出 className 域，该值由引擎给出，只要愿意，我

们可以改变 className 的值。

为 GameBase 数据块指明一个类，就相当于在任务编辑器中定义了一个新的种类。我们可以通过按 F11 键打开任务编辑器，然后通过 window→World Editor Creator，打开 Shapes 节点，你将会看到名为“ Brand New Category ”的新的类，打开该节点，会看到“ myGBD ”。

如果你只是简单的敲入上面的代码，那在变化实现之前，必须开始一个新的任务。

ShapeBaseData

从 ShapeBaseData 开始，数据块变得更加复杂和实用。ShapeBaseData 源于 GameBaseData 并提供更多新的属性。

ShapeBaseData 定义了渲染数据：可以是物品，材质效果以及一些毁坏方式。有关摄像机的数据也被定义了。此外还定义了 CRC 校验数据、物理信息数据，能量数据（可以不使用），简单的 AI 行为数据（可以不使用），和 HUD 数据（该部分目前并没有被引擎完全支持）。

Torque 引擎定义了相当多的数据块，熟悉这些数据的内容块是非常必要的。

表 4.3 ShapeBaseData 的属性内容

属性	类型	默认值	说明
GameBaseData Derived Fields			
Category	String		确认基于该数据块的物品的种类
className	String	ShapeBaseData	确定数据块的类。内容可以通过脚本改写
Rendering Data			
ShapeFile	Filename	NULL	物品的名字，要符合引擎定义的语法
cloakTexture	Filename		材质名
emap	Boolean	False	是否在游戏中渲染该物体
Destruction Effect Data			
Explosion	ExplosionDataPtr	NULL	指向 ExplosionData 类型的数据块。必须符合引擎中定义的 ExplosionData*类型。
Underwater	ExplosionDataPtr	NULL	水下爆炸效果。类型同上。

Explosion			
Debris	DebrisDataPtr	NULL	指向 DebrisData 数据块。符合引擎中 DebrisData* 定义的类型
debrisShapeName	Filename		Debris 域的物品的文件名
renderWhenDestroyed	Bollean	True	物品被破坏后是否还进行渲染
Camera Data			
cameraMaxDist	Float	0.0	第三视角下，摄像机离化身最大距离
cameraMinDist	Float	0.2	第三视角下，摄像机离化身的最小距离
caeraDefaultFOV	Float	90.0	默认视宽，以角度为单位。
cameraMinFOV	Float	5.0	允许的最小视宽（角度）
cameraMaxFOV	Float	120.0	允许的最大视宽（角度）
useEyePoint	Boolean	False	是否使用物品眼睛所在点作为摄像机观察的位置
observeThroughObject	Bolean	False	是否采用物品自身的观察方式
cameraDefaultFOV	Float	90.0	默认视宽（角度）
firstPersonOnly	Boolean	False	是否只使用第一视角
Security Data			
computeCRC	Boolean	False	指定 shapeFile CRC 是否必须同服务器传送的向匹配
Physics Data			

mass	Float	1.0	化身在游戏世界中的质量
density	Float	1.0	化身的密度，低密度可以漂浮
drag	Float	0.0	通过模拟化身的摩擦力降低速度
Damage-Specific Fields			
isInvincible	Boolean	False	指示实体是否可比遭受破坏
maxDamage	Float	1.0	实体可承受的最大伤害
destroyedLevel	Float	1.0	伤害级别，范围 0～1，可以改变
disableLevel	Float	1.0	实体丧失能力级别
repairRate	Float	0.0033	恢复率
Energy Data			
MaxEnergy	Float	0.0	最大能量，可以进行修改
inheritEnergyFromMount	Boolean	False	转移能量，可进行修改
AI Data			
aiAvoidThis	Boolean	False	该对象是否使用 AI（引擎不完全支持该项目）
HUD Data			
hudlmageName	String		引擎不完全支持该项目
hudlmageNameFriendly	String(array)		引擎不完全支持该项目
hudlmageNameEnermy	Stirng(array)		引擎不完全支持该项目
hudRenderCenter	Boolean(array)	False	引擎不完全支持该项目
hudRenderModulated	Boolean(array)	False	引擎不完全支持该项目
hudRenderA	Boolean(array)	False	引擎不完全支持该项目

lways			
hudRenderDistance	Boolean(array)	False	引擎不完全支持该项目
hudRenderName	Boolean(array)	False	引擎不完全支持该项目

创建 ShapeBaseData 方法与创建 GameBaseData 方法一样，只需要改变现有域的默认值为你想要的值，以及添加你的游戏中需要的域。

这里，我们发现 ShapeBaseData 开始定义许多有用的域，我们可以快速方便地利用他们。

PlayerData

PlayerData 数据块类是我们能够方便地定义玩家的行为特征。PlayerData 包括 ShapeBaseData 的全部域，同时增加了许多关于玩家的特征域。我们将学习 PlayerData 中的每个域，为了清晰明了，ShapeBaseData 中定义的域，这里将不在列出，如表 4.4 所示。

表 4.4　PlayerData 的属性及其内容

域名	类型	默认值	说明
Rendering Data			
renderFirstPerson	Boolean	True	在第一视角时是否渲染玩家化身
Special Effect Data			
DecalData	DecalDataPtr	NULL	如果指定，该域必须指向另一 DecalData 类型的对象，类型必须符合引擎的定义
decalOffset	Float	0.0	
footPuffEmitter	FootPuffEmitterDataPtr	NULL	如果指定，该域必须指向另一 FootPuffEmitterData 类型数据块，类型必须符合引擎的定义
footPuffNumParts	Integer	15	化身行动时产生的灰尘粒子数量

footPuffRadius	Float	0.25	灰尘在脚步边覆盖的范围
dustEmitter	ParticleEmitterDataPtr	NULL	在脚步位置发射粒子
FootSoftSound	AudioProfilePtr	NULL	脚步在软物体上的声音
FootHardSound	AudioProfilePtr	NULL	脚步在硬物体上的声音
FootMetalSound	AudioProfilePtr	NULL	脚步在金属上的声音
FootSnowSound	AudioProfilePtr	NULL	脚步在雪地上的声音
FootShallowSound	AudioProfilePtr	NULL	脚步在浅水里发出的声音
FootWandingSound	AudioProfilePtr	NULL	涉水（较深）时发出的声音
FootUnderwaterSound	AudioProfilePtr	NULL	水下发出的声音
FootBubblesSound	AudioProfilePtr	NULL	产生泡沫的声音
MovingBubblesSound	AudioProfilePtr	NULL	泡沫移动的声音
waterBreathSound	AudioProfilePtr	NULL	水下呼气的声音
impactSoftSound	AudioProfilePtr	NULL	碰撞软东西的声音
impactHardSound	AudioProfilePtr	NULL	碰撞硬东西的声音
impactMetalSound	AudioProfilePtr	NULL	碰撞金属的声音
impactSnowSound	AudioProfilePtr	NULL	碰撞雪地的声音
impactWaterEasy	AudioProfilePtr	NULL	轻轻与水碰撞的声音
impactWaterMedium	AudioProfilePtr	NULL	中度于水碰撞的声音
impactWaterHard	AudioProfilePtr	NULL	重度与水碰撞的声音
exitingWater	AudioProfilePtr	NULL	露出水面的声音
mediumSplashSoundVelocity	float	2.0	化身踏入水中溅起中等规模水花的最小速度
hardSplashSoundVelocity	float	3.0	化身踏入水中溅起大等规模水花的最小速度
exitSplashSoundVelocity	float	2.0	趟水发出声音的最小速度
Splash	SplashDtatPtr	NULL	在水中溅起水花的最小速度
splashVelocity	float	1.0	产生飞溅效果的最小速度

splashAngle	float	45	产生水花的最小角度
splashFreqMod	float	300.0	溅起水花的频率
splashVelEpsilon	float	0.25	停止移动时水花溅起的速度
bubbleEmitTime	float	0.4	持续冒泡的时间
splashEmitter	ParticleEmitterDataPtr(array)	NULL	指向 ParticleEmitterData 数据块的指针数组用以描述产生飞溅效果的粒子。
footstepSplashHeight	float	0.1	飞溅效果的最大高度
groundlmpactMinSpeed	float	10.0	产生尘土效果的最小碰撞速度
groundImpactShakeFreg	Point3F	(10,10,10)	摄像头颤动的频率
groundImpactShakeAmp	Point3F	(20,20,20)	摄像头颤动的幅度
groundImpactShakeDuration	Float	1.0	摄像头颤动持续时间
groundImpactShakeFalloff	Float	10.0	颤动振幅的削减

View Restrictions Data

minLookAngle	Float	-1.4	玩家看到的最低角度，弧度
maxLookAngle	Float	1.4	玩家看到的最高角度，弧度
maxFreelookAngle	Float	3.0	左右看到的最大角度，弧度

Player Movement Physics Data

runForce	Float	360.0	跑动的力量
maxForwardSpeed	Float	10.0	最大前进速度
maxBackwardSpeed	Float	10.0	最大后退速度
maxSideSpeed	Float	10.0	最大平移速度
maxUnderwaterForwardSpeed	Float	10.0	最大水中前进速度
maxUnderwaterBackwardSpeed	Float	10.0	最大水中后退速度
maxUnderwaterSideSpeed	Float	10.0	最大水中平移速度
maxStepHeight	Float	1.0	最大抬脚高度
runSurfaceAngle	Float	80.0	最大爬坡角度
horizMaxSpeed	Float	80.0	地面上的最大速度（使用工具）

horizResistSpeed	Float	38.0	开始产生阻力的最大速度
horizResistFactor	Float	1.0	达到 horizResistSpeed 产生的阻力
upMaxSpeed	Float	80.0	最大上升速度（使用工具）
upResistSpeed	Float	38.0	开始产生阻力的最大上升速度
horizResistFactor	Float	1.0	达到 upResistSpeed 产生的阻力
minImpactSpeed	Float	25.0	产生碰撞触发的最小速度
recoverDelay	Interger	30.0	恢复延迟
recoverRunForceScale	Float	1.0	由碰撞恢复过程的 run 力量修复值
minJumpSpeed	Float	500.0	起跳的最低速度
maxJumpSpeed	Float	1000.0	起跳的最高速度
jumpSurfaceAngle	Float	78.0	化身可跳动的斜坡最大度数
jumpDelay	Integer	78.0	两次起跳的最小间隔，毫秒
jumpForce	Float	75.0	起跳力
minJumpEnerge	Float	0.0	起跳的最小能量
jumpEnergyDrain	Float	0.0	跳跃消耗的能量
minRunEnergy	Float	0.0	奔跑的最小能量
runEnergyDrain	Float	0.0	奔跑消耗的能力
Bounding Box Data			
boundingBox	Point3F	(1,1,2.3)	指定玩家碰撞盒的宽、深和高
boxHeadPrecentage	Float	0.85	碰撞盒子底部到起始位置表明玩家的头部，85%，即从上往下的 15%被认为是头部
boxTorsoPercentage	Float	0.55	Torso 位置
boxHeadLeftPercentage	Integer	0	用来描述碰撞盒子左至右的位置，可以表示左臂（腿）被击中，推荐使用默认值
boxHeadRightPercentage	Integer	1	同上，表示身体右部分
boxHeadFrontPercentage	Integer	1	见上

boxHeadBackPercentage	Integer	0	见上
pickupRadius	Float	0.0	捡东西半径
Animation Data			
maxTimeScale	Float	1.5	化身动作动画的最大允许时间

注意：PlayerData 的 shapeFile 域必须赋予一个非空字符串，否则，Torque 将产生错误并退出。Torque 引擎没有定义任何 PlayerData 的派生数据块，下面我们将学习 ItemData，另一个 ShapeBaseData 的派生数据块。

ItemData

同 PlayerData 一样，ItemData 由 ShapeBaseData 派生并包含其全部域。ItemData 同样定义了新的域方便开发人员制作游戏世界中的基本项目。

为了清晰明了，我们仍然只列出 ItemData 中新增加的内容，如表 4.5 所示。

表 4.5 ItemData 的属性及其内容

域名	类型	默认值	说明
Lighting Data			
lightType	LightType(enum)	NoLight	指明与项目相关的光照类型。可用值为 NoLight，ConstantLight 和 PulsingLight。
lightColor	ColorF	(1.0,1.0,1.0,1.0)	光照颜色
lightRadius	Float	10.0	光照半径
lightOnlyStatic	Bollean	false	只用于静态项目，指明该项目是否用于该光照
lightTime	Integer	1000	脉冲光类型的时间间隔，毫秒
Physics Data			
elasticity	float	0.0	模拟物品的弹性物理属性，碰撞后的速度依赖该项（0～1）
friction	float	0.0	模拟摩擦力
sticky	Boolean	false	使物品具有粘性。

maxVelocity	float	-1.0	对象可以忍受的最大速度
gravityMod	float	1.0	重力加速度
Miscellaneous Fields			
PickupName	String	“an item”	当玩家拾取 item 时用于指定其名字
dynamicType	Integer	0	用来指示动 item 的动态类可用值是 $TypeMasks::DamagableItmeObjectType.

下一小节，我们将学习 MissionMarkerData、GameraData、PathCameraData 和 StaticShapeData 等 4 个数据块。由于它们全部由 ShapeBaseData 派生出来，并且几乎没有增加新的域，所以我们将它们放到一节进行学习。

MissionMarkerData 、 CameraData 、 PathCameraData 和 StaticShapeData

本小节中，我们将快速学习 MissionMarkerData, CameraData, PathCameraData, and StaticShapeData 等 4 个数据块，它们都派生于 ShapeBaseData，其中 MissionMarkerData, CameraData, PathCameraData 完全没有定义新的域，而 StaticShapeData 只定义了 2 个新域。

MissionMarkerData，正如其名，用来创建任务标记，在 torque 中主要用于建立 waypoints 和 spawn spheres。

GameraData 和 PathCameraData 都没有定义新的域名，只是简单的提供了源自 Camera 类的对象的数据。

StaticShapeData 和 StaticShape 类用于创建静态的.dts 物品。静态物品可用通过 TSStatic 类方便的创建。StaticShapeData 定义了以下 2 个新域，如表 4.6 所示。

表 4.6　StaticShpaeData 的属性及其内容

属性	类型	默认值	说明
noIndividualDamage	Boolean	False	该部分内容引擎目前不支持
dynamicType	Integer	0	用来指定 shape 的动态类型，任何类型都可用

以上是简单的数据块类型。下面几节我们将学习关于 VehicleData 数据块类。

VehicleData

VehicleData 源自 ShapeBaseData，帮助 vehicle 对象的创建。其它一些数据块如 WheeledVehicleData 和 FlyingVehicleData 等源于 VehicleData。下面是 VehicleData 定义的新属性，如表 4.7 所示。

表 4.7　VehicleData 的属性及其内容

域名	类型	默认值	描述
Special Effects Data			
softImpactSound	AudioProfilePtr	Null	描述轻微碰撞的 AudioProfile。一旦指定，必须指向一个 AudioProfile 对象。必须符合再 TGE 中定义的与 AudioProfile*关联的语意。
HardImpactSound	AudioProfilePtr		描述轻激烈碰撞的 AudioProfile。一旦指定，必须指向一个 AudioProfile 对象。必须符合再 TGE 中定义的与 AudioProfile*关联的语意。
splashFreqMod	Float	300.0	模拟车辆产生飞溅效果的频率。
splashVelEpsilon	Float	0.5	车辆停止时水花的速度
splashEmitter	ParticleEmitter DataPtr (array)	NULL	指向 ParticleEmitterData 的指针数组
exitSplashSoundVelocity	Folat	2.0	露出水面时声音的最小速度
softSplashSoundVelocity	Float	1.0	露出水面时柔软声音的最小速度
mediumSplashSoundVelocity	Float	2.0	露出水面时中度声音的最小速度
hardSplashSoundVelocity	Float	3.0	露出水面时重度声音的最小速度
exitingWater	AudioProfilePtr	NULL	露出水面的声音
impactWaterEasy	AudioProfilePtr	NULL	露出水面时轻度声音
impactWaterMedium	AudioProfilePtr	NULL	露出水面时中度声音

impactWaterHard	AudioProfilePtr	NULL	露出水面时重度声音
waterWakeSound	AudioProfilePtr	NULL	激起的水声
triggerDustHeight	Float	3.0	车辆产生尘土的最大高度
dustHeight	Float	1.0	尘土效果的高度
dustEmitter	ParticleEmitter DataPtr(array)	NULL	指向 ParticleEmitterData 数据块的指针数组
damageEmitterOffset	Point3F(array)		指向显示损坏效果的偏移量
numDmgEmitterAreas	Float	0.0	车辆上显示损失效果位置的数量
damageEmitter	ParticleEmitter DataPtr(array)	NULL	表现损伤效果发射粒子的指针数组

CameraData

cameraRoll	Bollean	True	
cameraLag	Float	0.0	车辆第三视角时摄像头滞后的距离
cameraDecay	Float	0.0	车辆第三视角时摄像头的偏移量
cameraOffset	Float	0.0	车辆视角的偏移量

Physics Data

Integration	Integer	1	每毫秒产生物理数据的离散的数字
minImpactSpeed	Float	25.0	车辆发生碰撞事件的最小速度
softImpactSpeed	Float	25.0	发出轻微碰撞声音的最小速度
hardImpactSpeed	Float	50.0	发出强烈碰撞声音的最小速度
massCenter	Point3f	(0,0,0)	车辆重心的位置
bodyRestitution	Float	1.0	碰撞时的反作用力
bodyFriction	Float	0.0	摩擦系数
contactTol	Float	0.1	触发碰撞接触的最小速度
collisionTol	Float	0.1	触发完整碰撞的最小速度
damageLevelTolerance	Float(array)	0.0	描述损伤等级的数组，表示损伤的百分比，决定显示何种损伤效果

maxSteeringAngle	Float	0.785	最大仰角，弧度
minDrag	Float	0.0	最小拉力，仅用于 FlyingVehicleData 绝对最大速度和拉力
maxDrag	Float	0.0	引擎暂时不支持该属性
jetForce	Float	500.0	车辆喷气产生的力量
jetEnergyDrain	Float	0.8	使用喷气时的能力消耗
minJetEnergy	Float	1.0	使用喷气所需的最小能能量
collDamageThresholdVel	Float	20.0	暂不可用
collDamageMultiplier	Float	0.05	暂不可用
minRollSpeed	Float	0.0	暂不可用

在下面的部分，我们讲学习 FlyingVehicleData，HoverVehicleData 和 WheeledVehicleData。

FlyingVehicleData

FlyingVehicleData 源于 VehicleData，用于创建飞行对象。它定义的新的属性如下表 4.8 所示：

表 4.8　FlyingVehicleData 的属性及其内容

域名	类型	默认值	描述
jetSound	AudioProfilePtr	Null	喷气的声音
engineSound	AudioProfilePtr	Null	发动机的声音
forwardJetEmitter	ParticleEmitterDataPtr	Null	向前喷射的粒子
backwardJetEmitter	ParticleEmitterDataPtr	Null	向后喷射的粒子
downJetEmitter	ParticleEmitterDataPtr	Null	向下喷射的粒子
trailEmitter	ParticleEmitterDataPtr	Null	产生痕迹的粒子
Physics Data			
maneuveringForce	Float	0.0	水平方向上产生的动力
horizontalSurfaceForce	Float	0.0	水平摩擦阻力
veticalSurfaceForce	Float	0.0	垂直摩擦阻力
rollForce	Float	1.0	滚动摩擦阻力

steeringForce	Float	1.0	螺旋桨动力
steeringRollForce	Float	1.0	同 rollForce 相反的力
maxAutoSpeed	Float	0.0	最大制动速度
autoAngularforce	Float	0.0	车辆纠正角度的力，速度越小，效果越明显
autoLinearForce	Float	0.0	使车辆减速的线性力
autoInputDamping	Float	1.0	速度低于 maxAutoSpeed 时候的量化因素
vertThrustMultiple	Float	1.0	产生强大 jetForce 的乘积因子
rotationalDrag	Float	0.0	与运动方向向反成角度的力
hoverHeight	Float	2.0	静止盘旋离地面的高度
createHoverHeight	Float	2.0	刚被创建时候的离地面高度
mintrailSpeed	Float	1.0	可以“拉线”的最小速度

如您所见，FlyingVehicleData 定义了一些有趣的新域名，并且使您很容易创建具有物理仿真特性的飞行器。下一小节，我们将学习 HoverVehicleData 数据块。

HoverVehicleData

HoverVehicleData 也是从 VechileData 中派生出来，用于创建盘旋的对象。HoverVehicleData 定义如下新的属性，如表 4.9 所示。

表 4.9 HoverVehicleData 的属性及其内容

域名	类型	默认值	描述
Special Effects Data			
jetSound	AudioProfilePtr	Null	喷气的声音
engineSound	AudioProfilePtr	Null	发动机的声音
floatSound	AudioProfilePtr	Null	漂浮的声音
dustTrailEmitter	ParticleEmitterDataPtr	Null	用来描述灰尘效果
dustTrailOffset	Point3F	(0,0,0)	车辆同地面产生灰尘的偏移量
dustTrailFreqMod	Float	15.0	产生灰尘的频率
triggerTrailHeight	Float	2.5	车辆保持产生灰尘的离地面最

			大高度
Physics Data			
dragForce	Float	0.0	车辆的拉力
vertFactor	Float	0.25	拉力的垂直分量
floatingThrustFactor	Float	0.15	车辆漂浮时候的前进力量
mainThrustForce	Float	0.0	向前推动力
reverseThrustForce	Float	0.0	向后推动力
strafeThrustFource	Float	0.0	转弯推动力
turboFactor	Float	1.0	喷射时候的动力因子
normalForce	Float	30.0	无
floatingGravMag	Float	1.0	漂浮时候的重力因素
brakingForce	Float	1.0	制动（刹车）产生的力
brakingActivationSpeed	Float	0.0	接受 brakingForce 的最大速度
gyroDrag	Float	10.0	产生角度的拉力
restorativeForce	Float	10.0	无
steeringForce	Float	25.0	在 X 轴操纵的力
rollforce	Float	2.5	在 X 方向旋转的力
pitchForce	Float	2.5	pitching 力
stabLenMin	Float	0.5	无
stabLenMax	Float	2.0	无
stabSpringConstant	Float	30.0	弹力值
stabDampingConstant	Float	10.0	弹力的反作用力

HoverVehicleData 提供了许对描述复杂的物理仿真的域。下一节，我们将学习 WheeledVehicleData。

WheeledVehicleData

WheeledVehicleData 派生于 VehicleData，用于创建带有轮胎的交通工具。WheeledVehicleData 定义的新属性如下表 4.10 所示：

表 4.10　WheeledVehicleData 的属性及其内容

域　名	类　型	默认值	描　述
Special Effects Data			
jetSound	AudioProfilePtr	Null	喷气的声音
engineSound	AudioProfilePtr	Null	发动机的声音
squealSound	AudioProfilePtr	Null	轮胎发出的声音
WheelImpactSound	AudioProfilePtr	Null	该属性暂无影响
tireEmitter	ParticleEmitterDataPtr	Null	指向每个轮胎的粒子发射器
Physics Data			
maxWheelSpeed	Float	40.0	车辆行使的最大速度
engineTorque	Float	1.0	扭矩
engineBrake	Float	1.0	制动

更多附加的数据在 WheeledVehicleTire 和 WheeledVehicleSpring 数据块中进行了定义，将在以后详细学习。

现在我们已经学习完所有派生于 ShapeBaseData 的数据块。下一节，我们将回过头来学习其它派生于 GameBaseData 的数据块，首先是 TriggerData。

TriggerData

我们已经详细的学习了派生于 ShapeBaseData 的数据块，现在返回 GameBaseData。TriggerData 数据块是 Torque 中经常使用的一个数据块。触发器提供了方便灵活的方式来执行响应玩家行为的动作，有关触发器的内容超出了本书的内容。

TriggerData 相当简单，它派生于 GameBaseData 并且有标准的类（category）和类名（className）域。此外，TriggerData 只定义了一个新属性，如表 4.11 所示。

表 4.11　TriggerData 的属性及其内容

属　性	类　型	默认值	描　述
tickPeriodMS	Integer	100	当触发器激活时，不断探测其行为的频率，毫秒为单位。

TirggerData 的单一域提供了创建一个行为触发器的必要数据。（当然，触发器被激发的操作执行起来也必须被提供。）更多信息参考触发器指南。

下一节，我们将学习 ProjectileData 数据块。

ProjectileData

ProjectileData 派生于 GameBaseData，用于为发射出去的对象提供渲染，行为和物理数据等。

下表中将对 ProjectileData 域名详细描述。为了节省空间，我们将省略 2 个域名（category 和 className），它们都是继承于 GameBaseData。

4.12 ProjectileData 的属性及其内容

属 性	类 型	默认值	描 述
Special Effects Data			
particleEmitter	ParticleEmitterDataPtr	Null	发射物（水面上）进入或离开水面产生的粒子
particleWaterEmitter	ParticleEmitterDataPtr	Null	发射物（水面下）进入或离开水面产生的粒子
Explosion	ExplosionDataPtr	Null	发射物水面上爆炸效果
waterExplosion	ExplosionDataPtr	Null	发射物水中爆炸
Sound	AudioProfilePtr	Null	发射物的声音
splash	SplashDataPtr	Null	发射物进入或离开水面水花的效果
decal	DecalDataPtr(array)	Null	发射物撞到地面或建筑产生的碎片
Rendering Data			
ProjectileShapeName	Filename	Null	发射物对象的名字
hasLight	Boolean	False	发射物是否有光（非水中）
hasWaterLight	Boolean	False	发射物是否有光（水中）
lightRadius	Float	1.0	发射物光的半径
lightColor	ColorF	(1,1,1)	发射物光的颜色（非水中）
waterLightColor	ColorF	(1,1,1)	发射物的光的颜色（水中）
scale	Point3F	(1,1,1)	渲染范围

Behavior Data			
lifetime	Integer	62	发射物变透明前的时间
armingDelay	Integer	0	发射物发射到爆炸的最小时间
fadeDelay	Integer	62	发射物创建出来到开始透明需要经过的时间
Physics Data			
isBallistic	Boolean	False	是否受到重力影响
velInheritFactor	Float	1.0	在脚本中起作用
muzzleVelocity	Float	50.0	初始速度（0～10,000）
bounceElasticity	Float	0.999	用来描述发射物在爆炸前碰到物体后的反弹动作，该值是反弹后速度和碰撞前速度的比值， 0.0 ～ 0.999
gravityMod	Float	1.0	如果 isBallistic 为 true，它来描述受重力影响程度

下一节，我们将学习 DebrisData，也是源于 GameBaseData。

DebrisData

DebrisData 数据块为 Debris（碎片）对象提供数据，该数据块描述的是非常简单的 shapes，不需要 ShapeBaseData 的全部功能，同时有一些独特的属性，如有限的生存周期，特效等。Debris 对象能够发射粒子、产生爆炸。DebrisData 源于 GameBaseData，表 4.13 是 DebirsData 的属性及其内容。

表 4.13 DebrisData 的属性及其内容

域名	类型	默认值	描述
Rendering Data			
Texture	String	Null	碎片对象使用的材质
shapeFile	Filename	Null	碎片对象文件名
Render2D	Boolean	False	是否使用广告牌技术
Special Effects Data			

Emitters	ParticleEmitterDataPtr(array)	Null	为碎片对象产生粒子效果
Explosion	ExplosionDataPtr	Null	碎片爆炸使用的爆炸数据块
Physics Data			
Elasticity	Float	0.3	弹力
Friction	Float	0.2	摩擦力
useRadiusMass	Boolean	False	决定 baseRadius 是否会对碎片的弹力、摩擦、旋转角度产生影响
baseRadius	Float	1.0	对碎片产生相关影响的半径
minSpinSpeed	Float	0.0	最小旋转速度
maxSpinSpeed	Float	0.0	最大旋转速度
gravModifier	Float	1.0	重力效果，1 为默认
terminalVelocity	Float	0.0	碎片飞出的最大速度
Velocity	Float	0.0	与 velocityVariance 一起决定碎片初始速度
velocityVariance	Float	0.0	与 velocity 一起决定碎片的初始速度
Lifetime	Float	3.0	碎片的生存时间
lifetimeVariance	Float	0.0	碎片的生存时间
Behavior Data			
numBounces	Integer	0.0	碎片获得的弹力最大值
bounceVariance	Integer	0	碎片可弹起的最大次数
explodeOnMaxBounce	Boolean	False	碎片是否在最大碰撞次数后爆炸
staticOnMaxBounce	Boolean	False	碎片是否在最大碰撞次数后静止
snapOnMaxBounce	Boolean	False	是否显示碎片的效果快照
ignoreWater	Boolean	True	忽视水的存在

Fade	Boolean	True	碎片开始退色（消失）

Debris 对象对那些能够跳跃或者具有周期性生存期的 shapes 非常有用。

SplashData

与 DebrisData 相似，SplashData 数据块来自于 GameBaseData，并定义了一些非常有用的域来实现特殊的功能。下面是关于 SplashData 的属性，如表 4.14 所示。

表 4.14　SplashData 的属性及其内容

域名	类型	默认值	描述
Special Effects Data			
soundProfile	AudioProfilePtr	Null	暂时无效
explosion	ExplosionDataPtr	Null	使用的爆炸数据块
emitter	ParticleEmitterDataPtr(array)	Null	飞溅对象使用的粒子发射器数据块
Rendering Effects Data			
colors	ColorF(array)	(1,1,1)	颜色数组，使用数组中指定的颜色表现颜色的变化
times	Float(array)	[0.0,1.0,…,1.0]	时间数组，使用数组中自定的时间来影响颜色数组中颜色变化的时间
Texture	Filename	0	使用的材质文件
texWrap	Float	1.0	渲染水花材质时候影响 U 坐标值
numSegments	Integer	10	产生每个飞溅粒子的段数
texFactor	Float	3.0	渲染水花材质时候影响 V 坐标值
Behavior Data			
lifetimeMS	Integer	1000	水花对象存在的最大时间，单位毫秒
lifetimeVariance	Integer	0	水花对象存在的周期，为系统会从该值和 lifetimeMS 之间随即计算出一个时间
rightLifetime	Float	1.0	由 splish 对象产生的 ring 的生存时间，单位秒。
delayMS	Integer	0	暂无作用

delayVariance	Integer	0	暂无作用
Scale	Point3F	(1,1,1)	暂无作用
Physics Data			
Velocity	Float	5.0	飞溅的速度
Acceleration	Float	0.0	飞溅的加速度
ejectionFreq	Float	5.0	产生新 splash ring 的频率
ejectionAngle	Float	45.0	新产生的 splash ring 的初射角度
startRadius	Float	1.0	Splash ring 发射的半径
Width	Float	4.0	暂无作用
Height	Float	0.0	暂无作用

SplashData 数据块使开发者具备了快速详细描述飞溅行为的能力。与 DebrisData 相似，对象可以产生各种各样的行为。

下一节，我们将学习 LightningData 数据块。

LightningData

LightningData 同样继承于 GameBaseData，发光的对象通过使用它来产生我们熟悉的光照效果。Torque 游戏引擎为我们提供的的光照效果相当出色，同时 LightningData 数据块本身却一点都不复杂，它的各种属性见表 4.15。

表 4.15 LightningData 的属性及其内容

域名	类型	默认值	描述
strikeSound	AudioProfilePtr	Null	产生闪电时候发出的声音
thunderSounds	AudioProfilePtr	Null	闪电过后的雷声
strikeTextures	String(array)	[Max Textures]	闪电使用的材质，数组

下一节，我们讲学习另外一个关于天气的数据块 PrecipitationData。

PrecipitationData

PrecipitationData 同样来自于 GameBaseData，它是另外一个关于天气的数据块。不过目前引擎的 Precipitation 代码存在一些问题，将在以后进行修正,因此，表中的部分内容不完整，请您见谅。它的内容如表 4.16 所示：

表 4.16　PrecipitationData 的属性及其内容

域　名	类　型	默认值	描　述
soundProfile	AudioProfilePtr	Null	Precipitation 对象的声音
Type	Integer	0	
maxSize	Float	1.0	
materialList	String	Null	
sizeX	Float	1.0	
sizeY	Float	1.0	
movingBoxPer	Float	(uninitialized)	
divHeightVal	Float	(uninitialized)	
sizeBigBox	Float	(uninitialized)	
topBoxSpeed	Float	(uninitialized)	
frontBoxSpeed	Float	(uninitialized)	
topBoxDrawPer	Float	(uninitialized)	
bottomDrawHeight	Float	(uninitialized)	
skipIfPer	Float	(uninitialized)	
bottomSpeedPer	Float	(uninitialized)	
frontSpeedPer	Float	(uninitialized)	
frontRadiusPer	Float	(uninitialized)	

下一节，我们来学习 ExplosionData。

ExplosionData

ExplosionData 同样来源于 GameBaseData，它可以创建惊人的爆炸效果，包括粒子、碎片、声音以及摄像机的颤抖。就像 DebrisData 和 SplashData 数据块一样，许多爆炸对象共享该数据块，并且表现出不同的效果。

下面是关于 ExplosionData 的介绍，见表 4.17 所示。

表 4.17　ExplosionData 的属性及其内容

属　性	类　型	默认值	描　述
Special Effects Data			

soundProfile	AudioProfilePtr	Null	爆炸时候的声音
particleEmitter	ParticleEmitterDataPtr	Null	爆炸产生的粒子
Emitter	ParticleEmitterDataPtr	Null	爆炸产生的粒子指针数组
Debris	DebrisDataPtr(array)	0	爆炸产生的碎片指针数组
subExplosion	ExplosionDataPtr(array)	0	描述爆炸产造成的子爆炸使用的 ExplosionData 数据块
Shockwave	ShockwaveDataPtr	Null	暂无作用
Rendering Data			
playSpeed	Float	1.0	播放爆炸 shape 动画的速度
explosionScale	Point3F		爆炸的范围
faceViewer	Boolean	False	将爆炸对准摄像头
lightStartRadius	Float	0.0	爆炸刚发生时候的光照半径
lightEndRadius	Float	0.0	爆炸结束时候的光照半径
lightStartColor	ColorF	(1,1,1)	爆炸发生时候光的颜色
lightEndColor	ColorF	(1,1,1)	爆炸结束时候光的颜色
Behavior Data			
playSpeed	Float	1.0	播放爆炸 shape 动画的速度
explosionScale	Point3F	(1,1,1)	爆炸的范围
Times	Float(array)	[0,1,…,1]	爆炸发生形变的开始时间
sizes	Point3F(array)	(1,1,1)	爆炸在各个方向的尺度
lifetimeMS	Integer	1000	爆炸的存在时间
lifetimeVariance	Integer	0	同 lifetimeMS 一起决定爆炸的存在时间
delayMS	Integer	0	爆炸延迟的时间，毫秒
delayVariance	Integer	0	同 delayMS 一起决定爆炸延迟的时间，毫秒
Offset	Float	0.0	偏移量
particleDensity	Integer	10	粒子的密度

particleRadius	Float	1.0	发射半径
debrisThetaMin	Float	0.0	碎片弹射的最小角度（x 轴）
debrisThetaMax	Float	90.0	碎片弹射的最大角度（x 轴）
debrisPhiMin	Float	0.0	碎片弹射的最小角度（z 轴）
debrisPhiMax	Float	360.0	碎片弹射的最大角度（z 轴）
debrisNum	Integer	1	爆炸弹出的碎片数量，在该值和 debrisNumVariance 之间随机产生一个值作为数量，1000 以内
debrisNumVariance	Integer	0	见 debrisNum
debrisVelocity	Float	2.0	爆炸碎片飞出的速度，该值和 debrisVelocityVariance 之间随机产生一个值作为速度
debrisVelocityVariance	Float	0.0	见 debrisVelocity
shakeCamera	Boolean	False	当摄像头在 camShakeRadius 之内时，是否会由于爆炸进行颤动
camShakeFreq	Point3F	(1,1,1)	摄像颤动的频率
camShakeAmp	Point3f	(1,1,1)	摄像头颤动的幅度
camShakeDuration	Float	1.5	摄像头颤动的时间，单位秒
camShakeRadius	Float	10.0	摄像头受爆炸影响发生颤动的最远距离
camShakeFalloff	Float	10.0	用来降低颤动的振幅
shockwaveOnTerrain	Boolean	False	暂无作用

ExplosionData 是个相对庞大和复杂的数据块，但是它为 Torque 的爆炸系统提供了非常灵活的方法，可以创造出非常酷的爆炸效果。

在下一节，我们将一起学习 torque 的粒子数据块系统。首先我们将学习数据块 ParticleEmitterNodeData 和 ParticleEmitterData，这两个数据块继承于 GameBaseData 数据块，然后学习数据块 ParticleData，该数据块继承于

SimBlockData 数据块。

ParticleEmitterNodeData 和 ParticleEmitterData

Torque 的粒子系统非常强大！粒子系统里包含 3 种数据块，ParticleEmitterNodeData 、 ParticleEmitterData 和 ParticleData 数据块。ParticleEmitterNodeData 和 ParticleEmitterData 是本节要学习的内容，ParticleData 数据块将放在下一节学习。

ParticleEmitterNodeData 和 ParticleEmitterData 都继承于 GameBaseData。ParticleEmitter 对象使用 ParticleEmitterNodeData 数据块，这意味着 ParticleEmitter 对象必须同其它对象连在一起。ParticleEmitterNode 对象提供一个或几个 hook（挂钩），在这个位置可以产生 ParticleEmitter。

ParticleEmitterNodeData 数据块相当简单，它只包含一个域，timeMultiple，如表 4.18 所示。

表 4.18　ParticleEmitterNodeData 的属性及其内容

属　性	类　型	默认值	描　述
timeMultiple	Float	1.0	粒子发射的时间

接下来是 ParticleEmitterData 数据块，如表 4.19 所示：

表 4.19　ParticleEmitterData 的属性及其内容

属　性	类　型	默认值	描　述
Particles	String	Null	定义好的 ParticleData 名字。当一个粒子由发射器发射出来后，会从定义好的 ParticleData 中随机选择一个作为该粒子的数据块
ejectionPeriodMS	Integer	100	粒子发射的周期（决定发射的频率），和 periodVarianceMS 一起决定，单位毫秒
periodVarianceMS	Integer	0	见 ejectionPeriodMS
thetaMin	Float	0.0	每次发射一个粒子，将获得一个发射的方向。由 thetaMin、thetaMax、phiReferenceVel 和 phiVariance 决

			定。引擎将会从 thetaMin 和 thetaMax 中随机产生一个角度值，该值应用于 x 轴，单位是角度，不可以大于 thetaMax
thetaMax	Float	90.0	见 thtaMin
phiReferenceVel	Float	0.0	每次发射一个粒子，将获得一个发射的方向。引擎将会从 phiReferenceVel 和 phiVariance 中随机选择一个角度，该值应用于 z 轴
phiVariance	Float	360.0	见 phiReferenceVel
ejectionOffset	Float	0.0	粒子离开发射器的偏移量
ejectionVelocity	Float	2.0	和 velocityVariance 一起决定粒子的发射速度，引擎会在这两个值之间随机选择一个速度，该值的范围在 0～655
velocityVariance	Float	1.0	见 ejectionVelocity
orientParticles	Bollean	False	指定发射的粒子是否垂直于摄像头，产生广告牌的效果
orientOnvelocity	Boolean	True	仅在 orientParticles 为 True 时有效。该值为 True 时候，如果粒子没有速度，将不会进行渲染；否则，粒子的速度大小将影响渲染的尺寸
useEmitterSizes	Boolean	True	是否使用发射器数据块的尺寸
useEmitterColors	Boolean	True	是否使用发射器数据块的颜色
overrideAdvance	Boolean	False	值为 False，新创建的粒子将立刻更新其加速度、颜色和其它设置
lifetimeMS	Integer	1000	同 lifetimeVarianceMS 一起决定粒子的存在时间。为了节省带宽，请使用 31 毫秒以内的数值

lifetimeVarianceMS	Integer	0	见 lifetimeMS

下一节我们学习 ParticleData 数据块。

ParticleData

ParticleData 用来描述粒子数据对象，其属性如表 4.20 所示。

表 4.20　ParticleData 的属性及其内容

属　性	类　型	默认值	描　述
Rendering Data			
textureName	Filename	0	粒子使用的材质文件名和位置
animTexName	FileName(array)	0	当 animateTexture 设置为 true 时，用来描述它的附加材质的文件名和位置
animateTexture	Boolean	False	指定粒子的材质是否为动画材质
framesPerSec	Integer	1	每秒播放的材质帧数，该数值必须小于 200，否则将强制为 20
useInvAlpha	Boolean	False	指定是否使用 alpha 通道
Colors	ColorF(array)	(1,1,1,1)	粒子的颜色（数组）
Sizes	Float(array)	1.0	粒子的尺寸（数组）
times	Float(array)	[0,1,2,2]	粒子的存在时间
Physics Data			
dragCoefficient	Float	0.0	模拟粒子收到的阻力，在 0～204.6 范围内有效，最小变化为 0.2
windCoefficient	Float	1.0	引擎支持风对粒子的影响
gravityCoefficient	Float	0.0	引擎支持重力对粒子的影响
inheritedVelFactor	Float	0.0	粒子创建时继承的速度
constantAcceleration	Float	0.0	粒子的基本加速度，类似其收到的拉力、风力和重力等
spinSpeed	Float	0.0	表示粒子旋转的速度，使用 spinRandomMin 和 spinRandomMax

			之间的值
spinRandomMin	Float	0.0	粒子旋转速度的最小值
spinRandomMax	Float	0.0	粒子旋转速度的最大值
Behavior Data			
lifetimeMS	Integer	1000	每个粒子的存在时间，单位毫秒，为了节约带宽，请不要设置太大
lifetimeVarianceMS	Integer	0	用来决定每个粒子的存在时间。当粒子创建时，系统将该值的正负值加到 lifetimeMS 上，然后再两个结果之间随机去一个值作为该粒子的生存时间，单位毫秒。

这样我们就学习了全部有关粒子系统的数据块。在下一节，我们将学习 WheeledVehicleTire 和 WheeledVehicleSpring 数据块。

WheeledVehicleTire 和 WheeledVehicleSpring

WheeledVehicleTire 和 WheeledVehicleSpring 数据块都继承于 SimDataBlock 数据块。首先让我们学习 WheeledVehicleTire 数据块的内容。正如它的名字那样，该数据块用来表现充气轮胎的对象。在现实中，充气轮胎具有很重要的作用，Torque 将实时地将它表现出来。下面是关于 WheeledVehicleTire 数据块的定义表。

域名	类型	默认值	描述
shapeFile	Filename		轮胎使用的模型的文件名和路径
Radius	Float	0.6	轮胎的半径。该半径由模型的碰撞盒决定，不用在脚本中设定
Mass	Float		轮胎的质量，不用在教本中指定
staticFriction	Float	1.0	轮胎表面的静态摩擦力（无牵引力）
kineticFriction	Float	0.5	轮胎表面的滑动摩擦力（无牵引力）
lateralForce	Float	10.0	使轮胎左右拐弯时需要的力
lateralDamping	Float	1.0	轮胎左右拐弯受到的阻力
lateralRelaxation	Float	1.0	轮胎能承受变形的力

longitudinalForce	Float	10.0	模拟轮胎纵向的力，可以理解为进退的牵引力
longitudinalDamping	Float	1.0	进退时候的阻力
restitution	Float	1.0	暂无作用

下表是 WheeledVehicleSpring 数据块。

域名	类型	默认值	描述
force	Float	10.0	弹力
length	Float	1.0	车辆悬空时候的高度
demping	Float	1.0	抵消弹力的阻力
antiSwayForce	Float	1.0	抵消左右摇摆的力

4.3 本章小结

本章我们学习了 Torque Script 的相关内容。学习了如何创建数据块对象。在第二章中，我们已经接触到了 tutorial.base 文件夹的大部分内容，并在其基础之上制作了一款小游戏。下一章中，我们将深入学习 tutorial.base 中的全部内容。

第五章　入门游戏 tutorial.base

通过前面第二章内容的学习，我们在 tutorial.base 文件夹的基础之上，制作了一个简单的具有基本游戏元素的小的游戏 Demo。接下来，让我们进一步学习 tutorial.base 文件中的各个文件的内容，从而从根本上掌握如何使用 Torque 构建我们自己的游戏。

tutorial.base 为我们提供了一个最基本的开发环境，里边预先定义了 sky（天空）、sun（太阳）、terrtain（地形）等元素。我们可以控制玩家在空旷的游戏世界里尽情奔跑，虽然没有更有趣的东西，但却是一个非常棒的开始。下面是 tutorial.base 文件夹同 example 文件夹的关系，如图 5.1 所示。

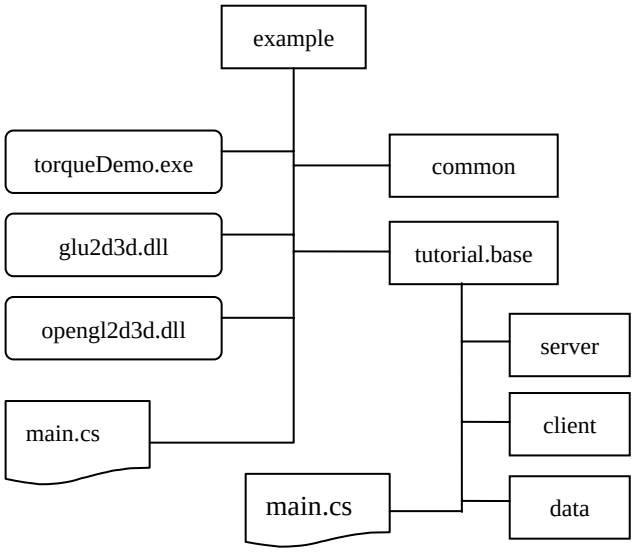


图 5.1　tutorial.base 文件夹同 example 文件夹的关系

接下来，让我们一起来学习一下 tutorial.base 中各个脚本文件中的代码，了解和掌握它们的作用。

5.1 根 main.cs

首先我们看看根 main.cs 文件。

```
//-----------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
```

```
//---------------------------------------------------------------------------
$defaultGame = "tutorial.base";
$displayHelp = false;
//---------------------------------------------------------------------------
// Support functions used to manage the mod string
function pushFront(%list, %token, %delim)
{
    if (%list !$= "")
        return %token @ %delim @ %list;
    return %token;
}
function pushBack(%list, %token, %delim)
{
    if (%list !$= "")
        return %list @ %delim @ %token;
    return %token;
}
function popFront(%list, %delim)
{
    return nextToken(%list, unused, %delim);
}
//---------------------------------------------------------------------------
// Process command line arguments
// Run the Torque Creator mod by default, it's needed for editors.
$isDedicated = false;
$modcount = 2;
$userMods = "creator;" @ $defaultGame;
for ($i = 1; $i < $Game::argc ; $i++)
{
    $arg = $Game::argv[$i];
```

```
$nextArg = $Game::argv[$i+1];
$hasNextArg = $Game::argc - $i > 1;
$logModeSpecified = false;
// Check for dedicated run
if( stricmp($arg,"-dedicated") == 0    )
{
    $userMods = $defaultGame;
    $modcount = 1;
    $isDedicated = true;
}
switch$ ($arg)
{
    //--------------------
    case "-log":
        $argUsed[$i]++;
        if ($hasNextArg)
        {
            // Turn on console logging
            if ($nextArg != 0)
            {
                // Dump existing console to logfile first.
                $nextArg += 4;
            }
            setLogMode($nextArg);
            $logModeSpecified = true;
            $argUsed[$i+1]++;
            $i++;
        }
        else
            error("Error: Missing Command Line argument. Usage: -log
```

```
<Mode: 0,1,2>");
            //--------------------
            case "-mod":
                $argUsed[$i]++;
                if ($hasNextArg)
                {
                    // Append the mod to the end of the current list
                    $userMods = strreplace($userMods, $nextArg, "");
                    $userMods = pushFront($userMods, $nextArg, ";");
                    $argUsed[$i+1]++;
                    $i++;
                    $modcount++;
                }
                else
                    error("Error: Missing Command Line argument. Usage: -mod
<mod_name>");
            //--------------------
            case "-game":
                $argUsed[$i]++;
                if ($hasNextArg)
                {
                    // Set the selected mod and creator for editor stuff.
                    if( $isDedicated )
                    {
                        $userMods = $nextArg;
                        $modcount = 1;
                    }
                    else
                    {
                        $userMods = "creator;" @ $nextArg;
```

```
                $modcount = 2;
            }
            $argUsed[$i+1]++;
            $i++;
        }
        else
            error("Error: Missing Command Line argument. Usage: -game <game_name>");

        //--------------------
        case "-show":
            // A useful shortcut for -mod show
            $userMods = strreplace($userMods, "show", "");
            $userMods = pushFront($userMods, "show", ";");
            $argUsed[$i]++;
            $modcount++;
        //--------------------
        case "-console":
            enableWinConsole(true);
            $argUsed[$i]++;
        //--------------------
        case "-jSave":
            $argUsed[$i]++;
            if ($hasNextArg)
            {
                echo("Saving event log to journal: " @ $nextArg);
                saveJournal($nextArg);
                $argUsed[$i+1]++;
                $i++;
            }
```

```
            else
                error("Error: Missing Command Line argument. Usage: -jSave <journal_name>");
            //--------------------
            case "-jPlay":
                $argUsed[$i]++;
                if ($hasNextArg)
                {
                    playJournal($nextArg,false);
                    $argUsed[$i+1]++;
                    $i++;
                }
                else
                    error("Error: Missing Command Line argument. Usage: -jPlay <journal_name>");
            //--------------------
            case "-jDebug":
                $argUsed[$i]++;
                if ($hasNextArg)
                {
                    playJournal($nextArg,true);
                    $argUsed[$i+1]++;
                    $i++;
                }
                else
                    error("Error: Missing Command Line argument. Usage: -jDebug <journal_name>");
            //-------------------
            case "-help":
                $displayHelp = true;
```

```
            $argUsed[$i]++;
        //------------------
        default:
            $argUsed[$i]++;
            if($userMods $= "")
                $userMods = $arg;
    }
}
if ($modcount == 0)
{
        $userMods = $defaultGame;
        $modcount = 1;
}

//--------------------------------------------------------------------------
// The displayHelp, onStart, onExit and parseArgs function are overriden
// by mod packages to get hooked into initialization and cleanup.
function onStart()
{
    // Default startup function
}
function onExit()
{
    // OnExit is called directly from C++ code, whereas onStart is
    // invoked at the end of this file.
}
function parseArgs()
{
    // Here for mod override, the arguments have already
    // been parsed.
```

```
}
package Help {
    function onExit() {
        // Override onExit when displaying help
    }
};
function displayHelp() {
    activatePackage(Help);
        // Notes on logmode: console logging is written to console.log.
        // -log 0 disables console logging.
        // -log 1 appends to existing logfile; it also closes the file
        // (flushing the write buffer) after every write.
        // -log 2 overwrites any existing logfile; it also only closes
        // the logfile when the application shuts down.    (default)
    error(
        "Torque Demo command line options:\n"@
        "    -log <logmode>              Logging behavior; see main.cs comments for details\n"@
        "    -game <game_name>           Reset list of mods to only contain <game_name>\n"@
        "    <game_name>                 Works like the -game argument\n"@
        "    -mod <mod_name>             Add <mod_name> to list of mods\n"@
        "    -console                    Open a separate console\n"@
        "    -show <shape>               Launch the TS show tool\n"@
        "    -jSave    <file_name>       Record a journal\n"@
        "    -jPlay    <file_name>       Play back a journal\n"@
        "    -jDebug <file_name>         Play back a journal and issue an int3 at the end\n"@
        "    -help                       Display this help message\n"
    );
```

```
}
//--------------------------------------------------------------------------
// Default to a new logfile each session.
if (!$logModeSpecified) {
    setLogMode(6);
}
// Set the mod path which dictates which directories will be visible
// to the scripts and the resource engine.
setModPaths($userMods);
// Get the first mod on the list, which will be the last to be applied... this
// does not modify the list.
nextToken($userMods, currentMod, ";");
// Execute startup scripts for each mod, starting at base and working up
function loadDir(%dir)
{
    setModPaths(pushback($userMods, %dir, ";"));
    exec(%dir @ "/main.cs");
}
echo("--------- Loading MODS ---------");
function loadMods(%modPath)
{
    %modPath = nextToken(%modPath, token, ";");
    if (%modPath !$= "")
        loadMods(%modPath);
    if(exec(%token @ "/main.cs") != true){
        error("Error: Unable to find specified mod: " @ %token );
        $modcount--;
    }
}
loadMods($userMods);
```

```
echo("");

if($modcount == 0) {
    enableWinConsole(true);
    error("Error: Unable to load any specified mods");
    quit();
}
// Parse the command line arguments
echo("--------- Parsing Arguments ---------");
parseArgs();
// Either display the help message or startup the app.
if ($displayHelp) {
    enableWinConsole(true);
    displayHelp();
    quit();
}
else {
    onStart();
    echo("Engine initialized...");
}
// Display an error message for unused arguments
for ($i = 1; $i < $Game::argc; $i++)    {
    if (!$argUsed[$i])
        error("Error: Unknown command line argument: " @ $Game::argv[$i]);
}
```

目前，对我们来说，最重要的是文件的第一行代码，即在前面章节已经再熟悉不过的“ $defaultGame="tutorial.base"; ”了。该命令指明了游戏所在的文件夹，也就是 tutorial.base，当然，以后你可以使用任何一个属于你自己游戏的文件夹名字。其它代码请不要做任何修改。下面我们解释其它代码部分的功能。

$displayHelp = false;表示不需要显示帮助文本。

三个函数 pushFront()、pushBack()和 popFront()使用来处理 Mod（包）的，作用类似与堆栈压栈、出栈的操作。

然后是处理命令行参数。如果我们直接使用开始菜单或者图标运行游戏，则命令行参数是预先定义好的。该处理是针对我们是用“开始”→“运行”命令，然后键入命令参数时候进行的相应操作。

5.2 tutorial.base 文件夹

tutorial.base 文件夹的结构及其部分内容如图 5.2 所示：

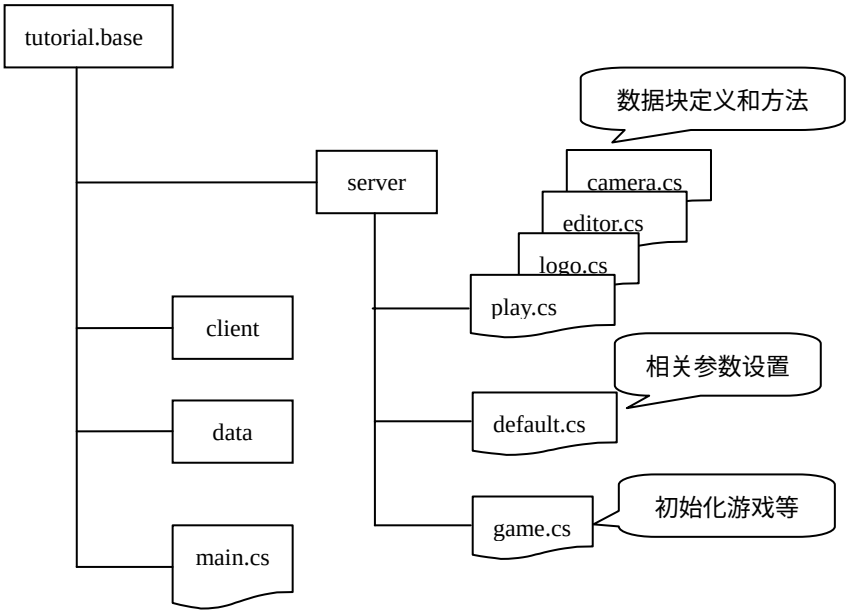


图 5.2　tutorial.base 文件夹的结构及其部分内容

注意：通常，我们在使用 C 语言编程的时候，都是将程序的内容放在 main{}之中。而在 Torque Script 中，则可以将不同的功能分别放在不同的.cs 文件之中，然后通过 exec()方法将其加载进去。我们可以将其理解为模块化设计。因此，当我们需要为游戏添加某些功能时，直接撰写相关该功能的程序，然后保存在.cs 文件中。

5.2.1 main.cs

main.cs 文件的代码如下：

```
//---------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//---------------------------------------------------------------------------
// Load up common script base
loadDir("common");
//---------------------------------------------------------------------------
// Load up default console values.
exec("./client/defaults.cs");
exec("./server/defaults.cs");
// Preferences (overide defaults)
exec("./prefs.cs");
//---------------------------------------------------------------------------
// Package overrides to initialize the game
package ttb {
function onStart()
{
    // Initialize the client and the server
    Parent::onStart();
    initServer();
    initClient();
    $Editor::newMissionOverride = "tutorial.base/data/missions/flat.mis";
}
function onExit()
{
    // Save off our current preferences for next time
    echo("Exporting prefs");
    export("$Pref::*", "./prefs.cs", False);
    Parent::onExit();
}
```

```
}; // Client package
activatePackage(ttb);
//--------------------------------------------------------------------------
function initServer()
{
   echo("\n--------- Initializing TTB: Server ---------");
   // The common module provides the basic server functionality
   initBaseServer();
   // Load up game server support scripts
   exec("./server/game.cs");
}
//--------------------------------------------------------------------------
function initClient()
{
   echo("\n--------- Initializing TTB: Client ---------");
   // The common module provides basic client functionality
   initBaseClient();
   // InitCanvas starts up the graphics system.
   // The canvas needs to be constructed before the gui scripts are
   // run because many of the controls assume the canvas exists at
   // load time.
   initCanvas("Torque Tutorial Base");
   // Load client-side Audio Profiles/Descriptions
   exec("./client/audioProfiles.cs");
   // Load up the shell and game GUIs
   exec("./client/ui/PlayGui.gui");
   exec("./client/ui/mainMenuGui.gui");
   exec("./client/ui/optionsDlg.gui");
   exec("./client/ui/loadingGui.gui");
   // Client scripts
```

```
        exec("./client/optionsDlg.cs");
        exec("./client/missionDownload.cs");
        exec("./client/serverConnection.cs");
        exec("./client/loadingGui.cs");
        exec("./client/playGui.cs");
        // Default player key bindings
        exec("./client/default.bind.cs");
        // Copy saved script prefs into C++ code.
        setShadowDetailLevel( $pref::shadows );
        setDefaultFov( $pref::Player::defaultFov );
        setZoomSpeed( $pref::Player::zoomSpeed );
        // Start up the main menu...
        Canvas.setContent(MainMenuGui);
        Canvas.setCursor("DefaultCursor");
    }
    //---------------------------------------------------------------------------
    // LOAD MY MISSION
    function loadMyMission()
    {
        // make sure we are not connected to a server already
        disconnect();
        // Create the server and load the mission
        createServer("SinglePlayer",
expandFilename("./data/missions/gameonemission.mis"));
        // Make a local connection
        %conn = new GameConnection(ServerConnection);
        RootGroup.add(ServerConnection);
        %conn.setConnectArgs("Player");
        %conn.setJoinPassword("None");
        %conn.connectLocal();
```

}

以上是游戏文件夹 tutorial.base 下的 main.cs 的内容，让我们来学习一下。

loadDir(“common”) ;加载 common 文件夹中的所有内容 ;接下来是 3 个 exec()命令，exec()命令会加载括号中指定的文件，这里加载了客户端和服务器端的 defaults.cs 文件以及 prefs.cs 文件。

Package ttb{}定义了一个包，包的名字是 ttb。里面定义了 2 个函数 onStart()和 onExit()。从名字即可以看出来，这两个函数的作用是开始和结束游戏。在 onStart()函数中，首先调用 Parent::OnStart()函数（即根 main.cs 中的 OnStart()函数）；接着调用 initServer()和 initClient(),即初始化服务器端和客户端代码；最后指定加载的任务文件的位置，即 tutorial.base/data/missions/flat.mis。onExit()函数先将属性设置保存到 tutorial.base 文件夹下的 prefs.cs 文件，然后调用 Parent::onExit()函数（即根 main.cs 文件中的 onExit()函数）。

接下来是函数 initServer()和 initClient()的具体内容。

initServer()函数负责初始化服务器端，调用了 initBaseServer()函数，该函数来自于 common 文件夹，然后使用 exec 加载 server/game.cs 文件。

initClient()函数负责初始化客户机端，首先调用 initBaseClient()函数，该函数同样来自于 common 文件夹，接下来使用 initCanvas 命令创建游戏窗口，窗口的名称为“ Torque Tutorial Base”，将来，这里你可以使用任何属于你的游戏的名字，war carft！是不是很棒？

接下来使用 exec 命令加载一系列客户端的.gui 文件，如显示玩家生命的血格等；然后加载一系列脚本文件和控制玩家的按键的文件（在 client/default.bind.cs 文件中），接下来设置了显示的细节等级、默认视角宽度以及摄像机的移动速度，这些参数来自于$pref::前缀的常量；其次，使用 Canvas.setContent 命令设置游戏窗口中显示的内容，通常是显示个动画，播放音乐和游戏选项界面，不是么？最后使用 Canvas.setCursor 命令设置鼠标状态。

最后说明一下 loadMyMission()函数，该函数是配合 GettingSearted.pdf 文档学习内容的，对我们所要制作的游戏没有任何影响。

disconnect()函数确保没有同任何服务器链接；

createServer("SinglePlayer",

expandFilename("./data/missions/gameonemission.mis"));创建一个单人游戏服务器，加载 gameonemission.mis 游戏。

%conn = new GameConnection(ServerConnection);建立一个新的连接对象，句柄保存在%conn 中。

注意，由于 Torque 是采用服务器－客户端游戏设计，所以即使是一个单机游戏，也相当于在本机上建立一个服务器同时运行一个客户端同其连接，然后进行游戏，因此需要建立一个 new GameConnection。

RootGroup.add(ServerConnection);将建立好的连接添加到 RootGroup 中。

%conn.setConnectArgs("Player");设置参数为 Player。

%conn.setJoinPassword("None");设置密码为不使用。

%conn.connectLocal();进行本地连接。

以上便是 toturial.base 的 main.cs 文件，看起来并不难以理解，是吧？好，再接再厉，我们继续！

5.2.2 server 文件夹

这里内容不多，只有 camera.cs、defaults.cs、editor.cs、game.cs、logoitem.cs、player.cs 等 6 个文件，其中 game.cs 和 player.cs 相对来说比较重要。下面是 game.cs 的内容。

game.cs

```
//-----------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------
//-----------------------------------------------------------------
//    Server and mission initialization
//-----------------------------------------------------------------
function onServerCreated()
{
    // This function is called when a server is constructed.
```

```
    // Master server information for multiplayer games
    $Server::GameType = "Torque TTB";
    $Server::MissionType = "None";
    // Load up all datablocks, objects etc.
    exec("./camera.cs");
    exec("./editor.cs");
    exec("./player.cs");
    exec("./logoitem.cs");
    exec("common/server/lightingSystem.cs");
}
function onServerDestroyed()
{
    // This function is called as part of a server shutdown.
}
//-----------------------------------------------------------------
function onMissionLoaded()
{
    // Called by loadMission() once the mission is finished loading.
}
function onMissionEnded()
{
    // Called by endMission(), right before the mission is destroyed
}
//-----------------------------------------------------------------
// Dealing with client connections
// These methods are extensions to the GameConnection class. Extending
// GameConnection make is easier to deal with some of this functionality,
// but these could also be implemented as stand-alone functions.
//-----------------------------------------------------------------
//-----------------------------------------------------------------
```

```
function GameConnection::onClientEnterGame(%this)
{
    // Every client get's a camera object.
    %this.camera = new Camera() {
        dataBlock = Observer;
    };
    MissionCleanup.add( %this.camera );
    %this.camera.scopeToClient(%this);
    // Create a player object.
    %spawnPoint = pickSpawnPoint();
    %this.createPlayer(%spawnPoint);
}
function GameConnection::onClientLeaveGame(%this)
{
    if (isObject(%this.camera))
        %this.camera.delete();
    if (isObject(%this.player))
        %this.player.delete();
}
//-----------------------------------------------------------------------
function GameConnection::createPlayer(%this, %spawnPoint)
{
    if (%this.player > 0)  {
        // The client should not have a player currently
        // assigned.  Assigning a new one could result in
        // a player ghost.
        error( "Attempting to create an angus ghost!" );
    }
    // Create the player object
    %player = new Player() {
```

```
        dataBlock = PlayerBody;
        client = %this;
    };
    MissionCleanup.add(%player);
    // Player setup...
    %player.setTransform(%spawnPoint);
    %player.setShapeName(%this.name);

    // Update the camera to start with the player
    %this.camera.setTransform(%player.getEyeTransform());
    // Give the client control of the player
    %this.player = %player;
    %this.setControlObject(%player);
}
//-----------------------------------------------------------------
function pickSpawnPoint()
{
    // Pick the first object in drop point group and use it's
    // location as a spawn point.
    %group = nameToID("MissionGroup/PlayerDropPoints");
    if (%group != -1 && %group.getCount() != 0)
        return %group.getObject(0).getTransform();
    // If no object was found, return a point near the center of the world
    error("Missing spawn point object and/or mission group " @ %groupName);
    return "0 0 300 1 0 0 0";
}
```

函数 onServerCreated()在服务器创建后调用，它负责管理服务器启动并运行之后发生的所有事情。通过$Server::GameType 和$Server::MissionType 变量设置一下服务器的信息，这里是游戏类型为“ Torque TTB ”，任务类型为“ None ”。

接下来使用我们早已熟悉的 exec 命令加载一系列文件。

函数 onServerDestroyed()在服务器关闭时候调用，这里目前没有任何代码。

函数 onMissionLoaded()在游戏任务加载结束后马上通过函数 loadMission()调用，目前没有任何代码。

函数 onMissionEnded()在游戏任务失败前，通过函数 endMission()调用，暂时没有任何内容。

接下来定义了一些 GameConnection 的方法（函数）。

方法 onClientEnterGame(%this)中，%this 表示同服务器连接的客户机的句柄，该方法首先为客户机创建一个 Observer 类型的观察摄像机，然后将该摄像机添加到任务列表，然后通过 scopeToClient(%this)命令将摄像机放置在客户机的位置。接下来创建玩家，%spawnPoint 中保存玩家将要出现的位置信息（三维坐标数据），该内容来自于函数 pickSpawnPoint()返回的内容。最后在指定的位置创建（createPlayer）玩家对象。

方法 onClientLeaveGame(%this)处理离开游戏的工作，删除玩家的摄像头和它本身。

方法 createPlayer(%this,%spawnPiont)用来创建游戏玩家化身。首先判断化身是否已经存在，存在的话将返回错误。然后通过 new Player()命令创建 client=%this 和 PlayerBody 类型的玩家化身，将返回的句柄放入%player 中；然后在任务列表中添加它，同时设置出生地点和名字，以及将摄像机的位置放置到玩家眼睛的位置上，最后通过 setControlObject()命令将控制权交给客户端玩家。

函数 pickSpawnPiont()用来选择玩家的出生位置。nameToID 命令返回括号里指定的文件夹路径，让后保存到%group 变量中，如果%group 变量不为空并且不为 0 时，返回第一个对象的位置信息，以后可以通过返回不同对象的位置信心，实现玩家出生点的变化。如果没有任何信息可以使用，则返回“ 0 0 300 1 0 0 0 ”，这是一个在游戏世界原点的位置，其中 7 个数字按顺序的含意为 x 坐标、y 坐标、z 坐标、是否绕 x 轴旋转（1 为是，0 为否）、是否绕 y 轴旋转、是否绕 z 轴旋转、旋转的角度。因此该位置信息表示在 x、y 坐标为 0、z 坐标为 300 的位置放置玩家，玩家绕 x 轴旋转 0 度，即没有旋转，而 z 坐标为 300，你会发现当你进入游戏时候，你的玩家（小蓝人）会从天上掉下来，为什么是 300 而不是 0？天

知道。呵呵，你完全可以将它改为你想要的数据。

player.cs

player.cs 内容如下：

```
//-----------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------
// Load dts shapes and merge animations
datablock TSShapeConstructor(PlayerDts)
{
   baseShape = "~/data/shapes/player/player.dts";
   sequence0 = "~/data/shapes/player/player_root.dsq root";
   sequence1 = "~/data/shapes/player/player_forward.dsq run";
   sequence2 = "~/data/shapes/player/player_back.dsq back";
   sequence3 = "~/data/shapes/player/player_side.dsq side";
   sequence4 = "~/data/shapes/player/player_fall.dsq fall";
   sequence5 = "~/data/shapes/player/player_land.dsq land";
   sequence6 = "~/data/shapes/player/player_jump.dsq jump";
   sequence7 = "~/data/shapes/player/player_standjump.dsq standjump";
   sequence8 = "~/data/shapes/player/player_lookde.dsq look";
   sequence9 = "~/data/shapes/player/player_head.dsq head";
   sequence10 = "~/data/shapes/player/player_headside.dsq headside";
   sequence11 = "~/data/shapes/player/player_celwave.dsq celwave";
};
datablock PlayerData(PlayerBody)
{
   renderFirstPerson = false;
   shapeFile = "~/data/shapes/player/player.dts";
};
```

```
//--------------------------------------------------------------------
// PlayerBody Datablock methods
//--------------------------------------------------------------------
function PlayerBody::onAdd(%this,%obj)
{
    // Called when the PlayerData datablock is first 'read' by the engine (executable)
}
function PlayerBody::onRemove(%this, %obj)
{
    if (%obj.client.player == %obj)
        %obj.client.player = 0;
}
function PlayerBody::onNewDataBlock(%this,%obj)
{
    // Called when this PlayerData datablock is assigned to an object
}
```

数据块 TSShapeConstructor()指定了玩家化身的 shape(形状)和 animations(动画)，其中 baseShape 指定玩家化身模型的 dts 文件，sequence * 指定了玩家化身的 dsq 动画文件，其中动画文件后面的单词，如 root、run 等，是命令字，引擎通过这些接受这些命令字，进而调用相应的动画显示在显示屏上，实现玩家化身的原地站立、跑动等动作。

数据块 PlayerData()定义了一个名为 PlayerBody 的玩家对象实例，renderFirstPerson = false 表示在第一人称视角下不渲染玩家化身的任何部位，即在第一人称视角下低头不会看到自己的脚等身体的任何部位，提高渲染效率。shapeFile 指定玩家化身模型文件在磁盘上保存的位置，即"~/data/shapes/player/player.dts"。

方法 onAdd()在引擎第一次加载 PlayerData 数据块时候调用，这里暂时没有任何内容。

方法 onRemove()移除玩家时候使用，将%obj.client.player 设置为 0。

方法 onNewDataBlock()在将该 PlayerData 数据块指派给另一对象时候调用，暂时没有任何内容。

在一下的代码里，为了节约纸张，提高效率，请允许我将部分注释的内容去掉，如果你愿意，请在引擎的代码中查看，谢谢。

camera.cs

camera.cs 内容如下：

```
//-----------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------
$Camera::movementSpeed = 40;
datablock CameraData(Observer)
{
   mode = "Observer";
};
```

太简单了，$Camera::movementSpeed 设置了摄像头移动速度，单位为引擎中默认的单位。

CaneraData 定义了一个名为 Observer 的摄像机数据块，将 mode 设置为 Observer，这个出现在前面 game.cs 中，还记得么？

default.cs

defaults.cs 文件内容如下：

```
//-----------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------
$Pref::Server::RegionMask = 2;
$pref::Master[0] = "2:master.garagegames.com:28002";
$Pref::Server::Name = "Torque TTB Server";
```

```
$Pref::Server::Info = "This is a Torque Game Engine Test Server.";
$Pref::Server::ConnectionError =
    "You do not have the correct version of the Torque Game Engine or "@
    "the related art needed to connect to this server, please contact "@
    "the server operator to obtain the latest version of this game.";
$Pref::Server::Port = 28000;
$Pref::Server::Password = "";
$Pref::Server::AdminPassword = "";
$Pref::Server::MaxPlayers = 64;
$Pref::Server::FloodProtectionEnabled = 1;
$Pref::Server::MaxChatLen = 120;
```

该文件主要内容是进行一系列游戏参数的设置，在我们目前学习阶段可以不用太关心它，放到那里就好了，当然，如果您坚持，我也相当乐意与您一起学习一下。

$Pref::Server::RegionMask = 2;好遗憾，第一个变量我就无法给你更多的解释，这里将其设置为 2，我猜想是同后面的主服务的 2 有关；

$pref::Master[0] = "2:master.garagegames.com:28002";设置主服务器信息，当我们试图开启专用服务器模式（dedicated）的时候，我们的服务器会向 garagegames 的主服务器的 28002 端口发送请求，收到应答后表明我们的专用服务器在 garagegames 的主服务器上注册成功，这样其它玩家可以通过主服务器的服务器列表找到我们的专用服务器，并连接进来。当然你也可以将地址设置为您自己架设的主服务器，不过这将超出本书的范围，这里不做介绍；

$Pref::Server::Name = "Torque TTB Server";设置服务器的名字为 Torque TTB Server；

$Pref::Server::Info = "This is a Torque Game Engine Test Server.";设置服务器的说明信息；

```
$Pref::Server::ConnectionError =
    "You do not have the correct version of the Torque Game Engine or "@
    "the related art needed to connect to this server, please contact "@
```

"the server operator to obtain the latest version of this game.";当发生错误时候，比如引擎的版本不正确，某些模型在客户端和服务器端不一致，将显示这些内容；

$Pref::Server::Port = 28000;设置我们的服务器的端口号为 28000；

$Pref::Server::Password = "";设置连接服务器的密码，这里表示没有任何密码；

$Pref::Server::AdminPassword = "";设置管理员密码，这里表示没有任何密码；

$Pref::Server::MaxPlayers = 64;设置服务器支持的最大连接玩家数；

$Pref::Server::FloodProtectionEnabled = 1;设置连接数超出保护；

$Pref::Server::MaxChatLen = 120;设置单次聊天输入的最大字符数；

以上内容都是进行服务器相关属性设置，在我们目前的所处的阶段，了解它们的含义和作用就行了，暂时不用对它们进行任何修改直到您将来需要的时候。

editor.cs

editor.cs 内容如下：

```
//---------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//---------------------------------------------------------------------
//---------------------------------------------------------------------
// This file contains shape declarations and functions used by the mission
// editor. The mission editor invokes datablock create methods when it
// wants to create an object of that type.
//---------------------------------------------------------------------
// Declare a static shape create method.  This allows DTS objects to be loaded
// using the StaticShape simulation object.
function StaticShapeData::create(%data)
{
   %obj = new StaticShape() {
```

```
        dataBlock = %data;
    };
    return %obj;
}
//--------------------------------------------------------------------
// Declare two marker objects used to place waypoints
datablock MissionMarkerData(WayPointMarker)
{
    category = "Misc";
    shapeFile = "~/data/shapes/markers/octahedron.dts";
};
datablock MissionMarkerData(SpawnSphereMarker)
{
    category = "Misc";
    shapeFile = "~/data/shapes/markers/octahedron.dts";
};
function MissionMarkerData::create(%block)
{
    switch$(%block)
    {
        case "WayPointMarker":
            %obj = new WayPoint() {
                dataBlock = %block;
            };
            return(%obj);
        case "SpawnSphereMarker":
            %obj = new SpawnSphere() {
                datablock = %block;
            };
            return(%obj);
```

```
    }
    return(-1);
}
//--------------------------------------------------------------------
// Misc. server commands used for editing
//--------------------------------------------------------------------
function serverCmdToggleCamera(%client)
{
    if ($Server::ServerType $= "SinglePlayer") {
        %control = %client.getControlObject();
        if (%control == %client.player) {
            %control = %client.camera;
            %control.mode = toggleCameraFly;
        }
        else {
            %control = %client.player;
            %control.mode = observerFly;
        }
        %client.setControlObject(%control);
    }
}
function serverCmdDropPlayerAtCamera(%client)
{
    if (!%client.player.isMounted())
        %client.player.setTransform(%client.camera.getTransform());
    %client.player.setVelocity("0 0 0");
    %client.setControlObject(%client.player);
}
function serverCmdDropCameraAtPlayer(%client)
{
```

```
    %client.camera.setTransform(%client.player.getEyeTransform());
    %client.camera.setVelocity("0 0 0");
    %client.setControlObject(%client.camera);
}
function dropFreakinCameraAtPlayer()
{
    $dropcameracount++;
    %cl = ClientGroup.getObject(0);
    if (%cl.camera) dropCameraAtPlayer(1);
    else if
($dropcameracount<100) schedule(100,0,dropFreakinCameraAtPlayer);
}
```

首先，方法 StaticShapeData::create(%this)定义了一种静态 DTS 对象的加载方式，该对象是 StaticShpae 对象。

接下来，使用 datablock 定义 2 个标记点对象，它们用来放置标记。它们都是 MissionMarkerData 类型的数据对象，名字分别为 WayPointMarker 和 SpawnSphereMarker，其中 WayPointMarker 标记用来表示路线标记，而 SpawnSphereMarker 表示玩家出生点标记。

方法 MissionMarkerData::create(%block)用来创建 MissionMarkerData 的对象实例，通过 switch 语句判断创建的具体对象实例是 WayPointMarker 类型还是 SpawnSphereMarker 类型。然后返回该对象实例的句柄。

函数 ServerCmdToggleCamera（%client）的作用就是将我们视角的位置在玩家视角和“ 上帝 ”视角之间进行切换。该函数就是 Camera 菜单中的 ToggleCamera 选项的实现。其中 ServerCmd 是服务器端命令前缀。

函数 serverCmdDropPlayerAtCamera（%client）的作用是在游戏编辑状态下，将玩家移动到摄像机所在的位置，如果在空中，则玩家一样被放置在空中，但是马上会在重力加速度作用下下落直到地面上，同时如果过高会导致生命损失。其中 if (!%client.player.isMounted())表明在您的玩家化身处在非绑定状态，也就是说您的花生没有在车辆的交通工具上。setTransform()方法会将调用该方法的对象放

置在（）中指定的坐标，getTransform()方法返回的是调用该方法的对象的当前位置坐标。

函数 serverCmdDropCameraAtPlayer(%client)的作用是将“ 上帝 ”视角的摄像机的位置设置在游戏玩家眼睛的地方，也就是玩家的第一人称视角状态，需要注意的是，该函数只是移动了摄像机的位置，而不是在“ 上帝 ”视角和玩家视角间的切换！也就是说当我们移动摄像机的时候还处于“ 上帝 ”视角状态。其中 getEyeTransform()方法将获得调用它的对象的模型虚拟点 eye 的坐标，注意，该模型必须具有 eye 虚拟点，否则将失败。至于什么是虚拟点 eye，直观的理解就是玩家化身模型的眼睛，如何制作请看考 dts 模型制作。setVelocity()方法将设置调用它的对象的速度,x、y、z 表示各坐标轴上的分量。方法 setControlObject()应该很熟悉了，将控制权交给括号中的对象。

logoitem.cs

logoitem.cs 文件的内容非常简单，如下

```
//-----------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------
datablock StaticShapeData(TorqueLogoItem)
{
    category = "Items";
    shapeFile = "~/data/shapes/3dtorquelogo/torque_logo.dts";
};
```

Server 文件夹下的最后一个文件了，看起来太简单啦，不是么？这里只定义了一个名为 TorqueLogoItem 的 StaticShapeData 类型数据块。它的 category 为 Items，这样，当我们在 world editor creator 的创建树中会看到 items 文件夹，下边包含 torque_logo.dts 文件。shapeFile 指明该对象使用的是什么位置的 3D 模型。

现在，我们已经了解了所有 server 文件夹中脚本文件的内容，我相信这些不会对您的理解造成什么障碍，好，加油，让我们一鼓作气学习一下 client 文件夹中都有什么好东西！

5.2.3 client 文件夹

client 文件夹中有 8 个.cs 脚本文件和一个 ui 文件夹。ui（User interface，用户接口）文件夹中包含的是图形用户接口相关的内容，如游戏开始时候的闪屏、可以点击的按钮等，其文件为.gui 格式。下面我们先学习一下.cs 脚本文件的内容。

audioProfiles.cs

audioProfiles.cs 文件内容如下：

```
//-----------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------
// Channel assignments (channel 0 is unused in-game).
$GuiAudioType         = 1;
$SimAudioType         = 2;
$MessageAudioType = 3;
new AudioDescription(AudioGui)
{
    volume      = 1.0;
    isLooping= false;
    is3D         = false;
    type         = $GuiAudioType;
};
new AudioDescription(AudioMessage)
{
    volume      = 1.0;
    isLooping= false;
    is3D         = false;
    type         = $MessageAudioType;
};
```

很简单，这里先定义了 3 个变量，然后定了 2 个新的 AudioDescription()对象实例，分别为 AudioGui 和 AudioMessage，其中 volue（音量）=1，isLooping=false 表示不循环播放，is3D=false 表示不是 3D 声音，即声音不随位置的变化而变化，最后，type 设置一下类型。

default.bind.cs

default.bind.cs 文件中定义了默认的游戏按键，内容如下：

```
//--------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
// Key bindings for the in-game action Map
//--------------------------------------------------------------------------
if ( isObject( moveMap ) )
    moveMap.delete();
new ActionMap(moveMap);
//--------------------------------------------------------------------------
// Misc. in-game keys
function escapeFromGame()
{
    if ( $Server::ServerType $= "SinglePlayer" )
        MessageBoxYesNo( "Quit Mission", "Exit from this Mission?", "disconnect();", "");
    else
        MessageBoxYesNo( "Disconnect", "Disconnect from the server?", "disconnect();", "");
}
moveMap.bindCmd(keyboard, "escape", "", "escapeFromGame();");
//--------------------------------------------------------------------------
```

```
// Movement Keys
$movementSpeed = 1; // m/s
function setSpeed(%speed)
{
    if(%speed)
        $movementSpeed = %speed;
}
function moveleft(%val)
{
    $mvLeftAction = %val * $movementSpeed;
}
function moveright(%val)
{
    $mvRightAction = %val * $movementSpeed;
}
function moveforward(%val)
{
    $mvForwardAction = %val * $movementSpeed;
}
function movebackward(%val)
{
    $mvBackwardAction = %val * $movementSpeed;
}
function moveup(%val)
{
    $mvUpAction = %val * $movementSpeed;
}
function movedown(%val)
{
    $mvDownAction = %val * $movementSpeed;
```

```
}
function turnLeft( %val )
{
    $mvYawRightSpeed = %val ? $Pref::Input::KeyboardTurnSpeed : 0;
}
function turnRight( %val )
{
    $mvYawLeftSpeed = %val ? $Pref::Input::KeyboardTurnSpeed : 0;
}
function panUp( %val )
{
    $mvPitchDownSpeed = %val ? $Pref::Input::KeyboardTurnSpeed : 0;
}
function panDown( %val )
{
    $mvPitchUpSpeed = %val ? $Pref::Input::KeyboardTurnSpeed : 0;
}
function getMouseAdjustAmount(%val)
{
    // based on a default camera fov of 90'
    return(%val * ($cameraFov / 90) * 0.01);
}
function yaw(%val)
{
    $mvYaw += getMouseAdjustAmount(%val);
}
function pitch(%val)
{
    $mvPitch += getMouseAdjustAmount(%val);
}
```

```
function jump(%val)
{
    $mvTriggerCount2++;
}
function mouseTrigger(%val)
{
    $mvTriggerCount0++;
}
moveMap.bind( keyboard, a, moveleft );
moveMap.bind( keyboard, d, moveright );
moveMap.bind( keyboard, w, moveforward );
moveMap.bind( keyboard, s, movebackward );
moveMap.bind( keyboard, space, jump );
moveMap.bind( mouse, xaxis, yaw );
moveMap.bind( mouse, yaxis, pitch );
moveMap.bind( mouse, button0, mouseTrigger );
//------------------------------------------------------------------------------
// Camera & View functions
function toggleFreeLook( %val )
{
    if ( %val )
        $mvFreeLook = true;
    else
        $mvFreeLook = false;
}
$firstPerson = true;
function toggleFirstPerson(%val)
{
    if (%val)
    {
```

```
        $firstPerson = !$firstPerson;
        ServerConnection.setFirstPerson($firstPerson);
    }
}
function toggleCamera(%val)
{
    if (%val)
        commandToServer('ToggleCamera');
}
moveMap.bind( keyboard, z, toggleFreeLook );
moveMap.bind(keyboard, tab, toggleFirstPerson );
moveMap.bind(keyboard, "alt c", toggleCamera);
//------------------------------------------------------------------------------
// Helper functions used by the mission editor
function dropCameraAtPlayer(%val)
{
    if (%val)
        commandToServer('dropCameraAtPlayer');
}
function dropPlayerAtCamera(%val)
{
    if (%val)
        commandToServer('DropPlayerAtCamera');
}
moveMap.bind(keyboard, "F8", dropCameraAtPlayer);
moveMap.bind(keyboard, "F7", dropPlayerAtCamera);
//------------------------------------------------------------------------------
// Key bindings for the Global action map available everywhere
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
```

```
// Misc.
GlobalActionMap.bind(keyboard, "tilde", toggleConsole);
GlobalActionMap.bindCmd(keyboard, "alt k", "cls();","");
GlobalActionMap.bindCmd(keyboard, "alt enter", "", "toggleFullScreen();");
//--------------------------------------------------------------------------
// Dubuging Functions
$MFDebugRenderMode = 0;
function cycleDebugRenderMode(%val)
{
    if (!%val)
        return;
    if (getBuildString() $= "Debug")
    {
        if($MFDebugRenderMode == 0)
        {
            // Outline mode, including fonts so no stats
            $MFDebugRenderMode = 1;
            GLEnableOutline(true);
        }
        else if ($MFDebugRenderMode == 1)
        {
            // Interior debug mode
            $MFDebugRenderMode = 2;
            GLEnableOutline(false);
            setInteriorRenderMode(7);
            showInterior();
        }
        else if ($MFDebugRenderMode == 2)
        {
            // Back to normal
```

```
                $MFDebugRenderMode = 0;
                setInteriorRenderMode(0);
                GLEnableOutline(false);
                show();
            }
        }
        else
        {
            echo("Debug render modes only available when running a Debug build.");
        }
    }
    GlobalActionMap.bind(keyboard, "F9", cycleDebugRenderMode);
```

该文件的目的是为玩家提供鼠标、按键等操作，它为我们建立一个 ActionMap 的对象实例 moveMap，因此，如果这个 moveMap 存在则必须通过 moveMap.delete()方法先将它删除。然后通过 new ActionMap(moveMap)命令建立一个新的 moveMap，这样就不会出现错误。

函数 escapeFromGame()定义了 esc 键的动作。

接下来定义了一系列移动键盘和鼠标的操作，然后使用 moveMap.bind(参数 1,参数 2,参数 3)方法将按键和动作绑定在一起。其中，参数 1 表示键盘（keyboard）或者鼠标（mouse），参数 2 表示具体的按键或者鼠标的移动以及左右键，参数 3 为对应该键的动作，使用的是前面定义的函数的名字。

接下来的函数 toggleFreeLook 和 toggleCamera 是不是很眼熟，它们就是负责切换视角的。由于进行视角切换的代码是在服务器端决定的，因此当我们想切换视角时，其实是向服务器端发送请求，然后服务器端进行了响应，这是通过从客户端使用 commandToServer()命令实现的，而括号中的 ToggleCamera 命令就是服务器端 camera.cs 中 ServercmdToggleCamera 的命令。

函数 draoCameraAtPlayer 和 dropPlayerAtCamera 类似，它们调用服务器端的 dropCameraAtPlayer 和 DropPlayerAtCamera 函数。

然后使用 GlobalActionMap.bind 绑定全局变量。

函数 cycleDebugRenderMode 是用来调试的，基本上与我们的游戏没有影响。

下面是所有按键和鼠标的绑定关系，如表 5.1 所示：

表 5.1　按键和鼠标的绑定关系

键　盘	
A	左移
D	右移
W	前进
S	后退
SPC（空格）	跳跃
Esc	弹出结束对话框
Z	自由观看
Tab（制表符）	在第一和第三人称视角之间切换
Alt + C	在玩家视角和上帝视角之间切换
Alt + enter	在全屏和窗口之间切换
`	调出控制台
F7	将玩家位置设置在摄像机所在位置
F8	将摄像机位置设置在玩家所在位置
鼠　标	
鼠标左右移动	水平方向转动视角
鼠标上下移动	垂直方向转动视角
鼠标左键	射击动作

defaults.cs

defaults.cs 文件内容如下：

```
//-----------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
```

```
//--------------------------------------------------------------------
// The master server is declared with the server defaults, which is
// loaded on both clients & dedicated servers.    If the server mod
// is not loaded on a client, then the master must be defined.
//$pref::Master[0] = "2:v12master.dyndns.org:28002";
$pref::Player::Name = "Test Guy";
$pref::Player::defaultFov = 90;
$pref::Player::zoomSpeed = 0;
$pref::Net::LagThreshold = 400;
$pref::shadows = "2";
$pref::HudMessageLogSize = 40;
$pref::ChatHudLength = 1;
$pref::Input::LinkMouseSensitivity = 1;
// DInput keyboard, mouse, and joystick prefs
$pref::Input::KeyboardEnabled = 1;
$pref::Input::MouseEnabled = 1;
$pref::Input::JoystickEnabled = 0;
$pref::Input::KeyboardTurnSpeed = 0.1;
$pref::sceneLighting::cacheSize = 20000;
$pref::sceneLighting::purgeMethod = "lastCreated";
$pref::sceneLighting::cacheLighting = 1;
$pref::sceneLighting::terrainGenerateLevel = 1;
$pref::Terrain::DynamicLights = 1;
$pref::Interior::TexturedFog = 0;
$pref::Video::displayDevice = "OpenGL";
$pref::Video::allowOpenGL = 1;
$pref::Video::allowD3D = 1;
$pref::Video::preferOpenGL = 1;
$pref::Video::appliedPref = 0;
$pref::Video::disableVerticalSync = 1;
```

```
$pref::Video::monitorNum = 0;
$pref::Video::windowedRes = "800 600";
$pref::Video::screenShotFormat = "PNG";
$pref::OpenGL::force16BitTexture = "0";
$pref::OpenGL::forcePalettedTexture = "0";
$pref::OpenGL::maxHardwareLights = 3;
$pref::VisibleDistanceMod = 1.0;
$pref::Audio::driver = "OpenAL";
$pref::Audio::forceMaxDistanceUpdate = 0;
$pref::Audio::environmentEnabled = 0;
$pref::Audio::masterVolume      = 0.8;
$pref::Audio::channelVolume1 = 0.8;
$pref::Audio::channelVolume2 = 0.8;
$pref::Audio::channelVolume3 = 0.8;
$pref::Audio::channelVolume4 = 0.8;
$pref::Audio::channelVolume5 = 0.8;
$pref::Audio::channelVolume6 = 0.8;
$pref::Audio::channelVolume7 = 0.8;
$pref::Audio::channelVolume8 = 0.8;
```

defaults.cs 文件中设置了许多相关参数。其中：

$pref::Player::Name = "Test Guy"；设置了玩家默认的名字为 Test Guy，当然您可以通过在名字输入框中输入您希望的名字，但是 Torque 游戏引擎暂时不支持中文名字。

$pref::Player::defaultFov = 90；设置了玩家的视角，即视线左右的宽度和上下的高度，这里是 90 度。

$pref::Player::zoomSpeed = 0;设置了玩家身后摄像头的聚焦速度，这里为 0。解释一下，很多游戏中，都会使用类似鼠标滚轮滚动的方式，拉近和拉远摄像头与玩家的距离，这个速度就是拉近或拉远的摄像头移动的速度。

$pref::Net::LagThreshold = 400;网络延迟的时间，单位毫秒。

$pref::shadows = "2";设置阴影等级为 2。

$pref::HudMessageLogSize = 40;

$pref::ChatHudLength = 1;

$pref::Input::LinkMouseSensitivity = 1;使鼠标感应有效，如使用鼠标点选物体等。

$pref::Input::KeyboardEnabled = 1;使键盘有效。

$pref::Input::MouseEnabled = 1;使鼠标有效。

$pref::Input::JoystickEnabled = 0;使游戏杆无效。

$pref::Input::KeyboardTurnSpeed = 0.1;键盘识别按键的速度，单位秒。

$pref::sceneLighting::cacheSize = 20000;设定场景照明的缓存大小。

$pref::sceneLighting::purgeMethod = "lastCreated";设定场景照明的清除方式为 lastCreated 模式。

$pref::sceneLighting::cacheLighting = 1;设定场景照明缓存为“真”，即有效。

$pref::sceneLighting::terrainGenerateLevel = 1;设定地形生成级别为“真”。

$pref::Terrain::DynamicLights = 1;设定地形的动态光照为“真”。

$pref::Interior::TexturedFog = 0;设定 Interior 内部雾材质为“假”，即无效。

$pref::Video::displayDevice = "OpenGL";设定显示 API 为 OpenGL。

$pref::Video::allowOpenGL = 1;使 OpenGL 有效。

$pref::Video::allowD3D = 1;使 DirectX 的 D3D 有效。

$pref::Video::preferOpenGL = 1;使优先使用 OpenGL 作为 API 有效。

$pref::Video::appliedPref = 0;设置$pref::Video::appliedPref = 0。

$pref::Video::disableVerticalSync = 1;使垂直同步有效；

$pref::Video::monitorNum = 0;设定监视器数量。

$pref::Video::windowedRes = "800 600";设定窗口尺寸，即 800×600。

$pref::Video::screenShotFormat = "PNG";设置图片默认格式为 PNG 格式。

$pref::OpenGL::force16BitTexture = "0";不使用 16 位材质。

$pref::OpenGL::forcePalettedTexture = "0";不使用调色板材质。

$pref::OpenGL::maxHardwareLights = 3;设置硬件光照等级为 3 级。

$pref::VisibleDistanceMod = 1.0;设置可是距离模式为 1.0。

$pref::Audio::driver = "OpenAL";设置声音 API 为 OpenAL。

$pref::Audio::forceMaxDistanceUpdate = 0;使最远距离更新无效。

$pref::Audio::environmentEnabled = 0;使环境音效无效。

$pref::Audio::masterVolume = 0.8;主声道音量值。

$pref::Audio::channelVolume1 = 0.8;通道 1 的音量值。

$pref::Audio::channelVolume2 = 0.8;通道 2 的音量值。

$pref::Audio::channelVolume3 = 0.8;通道 3 的音量值。

$pref::Audio::channelVolume4 = 0.8;通道 4 的音量值。

$pref::Audio::channelVolume5 = 0.8;通道 5 的音量值。

$pref::Audio::channelVolume6 = 0.8;通道 6 的音量值。

$pref::Audio::channelVolume7 = 0.8;通道 7 的音量值。

$pref::Audio::channelVolume8 = 0.8;通道 8 的音量值。

loadingGui.cs

loadingGui.cs 文件内容如下：

```
//--------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//--------------------------------------------------------------------
//--------------------------------------------------------------------
function LoadingGui::onAdd(%this)
{
   %this.qLineCount = 0;
}
//--------------------------------------------------------------------
function LoadingGui::onWake(%this)
{
   // Play sound...
   CloseMessagePopup();
}
```

```
//--------------------------------------------------------------------
function LoadingGui::onSleep(%this)
{
    // Clear the load info:
    if ( %this.qLineCount !$= "" )
    {
        for ( %line = 0; %line < %this.qLineCount; %line++ )
            %this.qLine[%line] = "";
    }
    %this.qLineCount = 0;
    LOAD_MapName.setText( "" );
    LOAD_MapDescription.setText( "" );
    LoadingProgress.setValue( 0 );
    LoadingProgressTxt.setValue( "WAITING FOR SERVER" );
    // Stop sound...
}
```

loadingGui.cs 文件中的所有方法都与 LoadingGui 有关。LoadingGui 指的是加载任务文件时候显示的图像。在 torque 中通过在游戏窗口显示不同的 gui 对象（可以显示多个）来表现游戏，如游戏界面为 playGui，开始界面为 startGui 等等。其中：

方法 LoadingGui::onAdd()十分简单，只是将%this.qLineCount 设置为 0。

方法 LoadingGui::onWake()也十分简单，直接调用 CloseMessagePopup()函数，这是在游戏窗口被唤醒的时候进行的，该函数的功能是将在游戏窗口休眠时候压入堆栈的信息出栈，以继续使用。

方法 LoadingGui::onSleep()用来处理游戏窗口进入休眠状态的工作。首先，判断一下%this.qLineCount 是否存在，如果存在，则使用一个 for 循环进行赋值,即将所有的%this.qLine[]中的内容都清空。然后设置%this.qLineCount＝0。最后使用如下命令

```
LOAD_MapName.setText( "" );
LOAD_MapDescription.setText( "" );
```

```
LoadingProgress.setValue( 0 );
    LoadingProgressTxt.setValue( "WAITING FOR SERVER" );
```

设置对应的值。

missionDownload.cs

missionDownload.cs 文件内容如下：

```
//--------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//--------------------------------------------------------------------------
// Mission Loading & Mission Info
// The mission loading server handshaking is handled by the
// common/client/missingLoading.cs.  This portion handles the interface
// with the game GUI.
//--------------------------------------------------------------------------
// Loading Phases:
// Phase 1: Download Datablocks
// Phase 2: Download Ghost Objects
// Phase 3: Scene Lighting
//--------------------------------------------------------------------------
// Phase 1
//--------------------------------------------------------------------------
function onMissionDownloadPhase1(%missionName, %musicTrack)
{
    // Close and clear the message hud (in case it's open)
    //cls();
    // Reset the loading progress controls:
    LoadingProgress.setValue(0);
    LoadingProgressTxt.setValue("LOADING DATABLOCKS");
}
```

```
function onPhase1Progress(%progress)
{
    LoadingProgress.setValue(%progress);
    Canvas.repaint();
}
function onPhase1Complete()
{
}
//---------------------------------------------------------------
// Phase 2
//---------------------------------------------------------------
function onMissionDownloadPhase2()
{
    // Reset the loading progress controls:
    LoadingProgress.setValue(0);
    LoadingProgressTxt.setValue("LOADING OBJECTS");
    Canvas.repaint();
}
function onPhase2Progress(%progress)
{
    LoadingProgress.setValue(%progress);
    Canvas.repaint();
}
function onPhase2Complete()
{
}
//---------------------------------------------------------------------------
// Phase 3
//---------------------------------------------------------------------------
function onMissionDownloadPhase3()
```

```
{
    LoadingProgress.setValue(0);
    LoadingProgressTxt.setValue("LIGHTING MISSION");
    Canvas.repaint();
}
function onPhase3Progress(%progress)
{
    LoadingProgress.setValue(%progress);
}
function onPhase3Complete()
{
    LoadingProgress.setValue( 1 );
    $lightingMission = false;
}
//--------------------------------------------------------------------------
// Mission loading done!
//--------------------------------------------------------------------------
function onMissionDownloadComplete()
{
    // Client will shortly be dropped into the game, so this is
    // good place for any last minute gui cleanup.
}
//--------------------------------------------------------------------------
// Before downloading a mission, the server transmits the mission
// information through these messages.
//--------------------------------------------------------------------------
addMessageCallback( 'MsgLoadInfo', handleLoadInfoMessage );
addMessageCallback( 'MsgLoadDescripition', handleLoadDescriptionMessage );
addMessageCallback( 'MsgLoadInfoDone', handleLoadInfoDoneMessage );
//--------------------------------------------------------------------------
```

```
function handleLoadInfoMessage( %msgType, %msgString, %mapName ) {
    // Need to pop up the loading gui to display this stuff.
    Canvas.setContent("LoadingGui");
    // Clear all of the loading info lines:
    for( %line = 0; %line < LoadingGui.qLineCount; %line++ )
        LoadingGui.qLine[%line] = "";
    LoadingGui.qLineCount = 0;
    LOAD_MapName.setText( %mapName );
}
//--------------------------------------------------------------------------
function handleLoadDescriptionMessage( %msgType, %msgString, %line )
{
    LoadingGui.qLine[LoadingGui.qLineCount] = %line;
    LoadingGui.qLineCount++;
    // Gather up all the previous lines, append the current one
    // and stuff it into the control
    %text = "<spush><font:Arial:16>";
    for( %line = 0; %line < LoadingGui.qLineCount - 1; %line++ )
        %text = %text @ LoadingGui.qLine[%line] @ " ";
    %text = %text @ LoadingGui.qLine[%line] @ "<spop>";
    LOAD_MapDescription.setText( %text );
}
//--------------------------------------------------------------------------
function handleLoadInfoDoneMessage( %msgType, %msgString )
{
    // This will get called after the last description line is sent.
}
```

该文件的内容主要用来显示任务加载过程和任务的信息。真正的任务加载是由common文件夹下的common/client/missingLoading.cs实现的，而该文件只是用

来显示客户端的图像信息，如显示加载进度条等。加载任务过程一共分3步，第一步为加载数据块，该步骤由函数onMissionDownloadPhase1()、onPhase1Progress()和onPhase1Complete()共同完成。其中，函数onMissionDownloadPhase1()的作用是初始化LoadingProgress（加载进度）的值，使用setValue(0)的方法将加载进度初始化为0。然后通过LoadingProgressTxt.setValue("LOADING DATABLOCKS");命令设置进度条上显示“ LOADING DATABLOCKS ”的字样。

函数 onPhase1Progress()用来处理加载过程中，进度条的变化。首先使用 LoadingProgress 的 setValue()方法，显示当前加载的进度，入 50%等。然后，调用 Canvase.repaint()命令进行窗口重绘，这样，您就会看到进度条的不断增加直到 100%。

函数 onPhase1Complete()是当第一步骤结束后，需要进行的工作。这里边暂时什么都不做，当然，您可以根据您的需要进行相关脚本代码的添加，如播放个声音等。

第二步为加载对象，其过程同第一步几乎完全一样，显示为“ LOADING OBJECTS ”。

第三步加载光照信息，该步骤的内容同前两步也几乎完全一样，显示为“ LIGHTING MISSION ”。只是在最后的onPhase3Complete()进行了LoadingProgress.setValue(1);和$lightingMission = false;的赋值操作。

注意：就目前来说，该文件的内容理解起来可能会有一定的难度，而且对我们目前的学习来讲影响很小，它仅仅是在我们的客户机窗口上显示一下目前游戏任务正在被加载，以及加载的进度，这是通过 Camvas.repaint()方法实现的。我们完全不用将时间浪费到深入研究其每行代码的含义，而是将精力放在对我们更有用的地方。

optionDlg.cs 文件主要用来处理游戏中“ 选项对话框 ”等相关设置的操作。内容比较多，但是并不是很复杂，相信您完全有能力看懂他，这里就不一一解释了。

playGui.cs

playGui.cs 文件内容如下：

//--

```
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
// PlayGui is the main TSControl through which the game is viewed.
//---------------------------------------------------------------------------
function PlayGui::onWake(%this)
{
    // Turn off any shell sounds...
    // alxStop( ... );
    $enableDirectInput = "1";
    activateDirectInput();
    // Activate the game's action map
    moveMap.push();
}
function PlayGui::onSleep(%this)
{
    // Pop the keymap
    moveMap.pop();
}
```

PlayGui 是我们进行游戏的主窗口，注意，这里所说的窗口是指游戏窗口的客户图形区，不包括标题栏和菜单栏。所有游戏中的对象都是通过 PlayGui 呈现给玩家。

该文件的内容是不是太简单了点？是的，就是这么简单！这里仅仅定义了 PlayGui 的两个方法 onWake()和 onSleep()，用来处理 PlayGui 的休眠和唤醒时候发生的事件，也就是类似当我们使用 Alt+Tab 键切换到其它应用程序以及返回游戏窗口时候进行的操作。onWake()方法进行游戏唤醒的操作，设置 $enableDirectInput＝1（真），使用 activateDirectInput()函数激活输入设备，然后调用 moveMap.push()方法将 moveMap 压如堆栈，这样我们就可以在游戏里使用

定义的按键进行游戏了。方法 onSleep()非常简单，直接调用 moveMap.pop()方法将 moveMap 弹出堆栈。

serverConnection.cs

serverConnection.cs 文件内容如下：

```
//-----------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------------
// Functions dealing with connecting to a server
//-----------------------------------------------------------------------------
// Server messages
//-----------------------------------------------------------------------------
function onServerMessage(%mesg)
{
   // Server message text which includes chat from other players.
}
//-----------------------------------------------------------------------------
// Server connection error
//-----------------------------------------------------------------------------
addMessageCallback( 'MsgConnectionError', handleConnectionErrorMessage );
function handleConnectionErrorMessage(%msgType, %msgString, %msgError)
{
   // On connect the server transmits a message to display if there
   // are any problems with the connection.  Most connection errors
   // are game version differences, so hopefully the server message
   // will tell us where to get the latest version of the game.
   $ServerConnectionErrorMessage = %msgError;
}
//-----------------------------------------------------------------------------
```

```
// GameConnection client callbacks
//---------------------------------------------------------------------------
function GameConnection::initialControlSet(%this)
{
    echo ("*** Initial Control Object");
    // The first control object has been set by the server
    // and we are now ready to go.
    // first check if the editor is active
    if (!Editor::checkActiveLoadDone())
    {
        if (Canvas.getContent() != PlayGui.getId())
            Canvas.setContent(PlayGui);
    }
}
function GameConnection::setLagIcon(%this, %state)
{
    if (%this.getAddress() $= "local")
        return;
    // Called when the server lag state changes state = true/false
}
function GameConnection::onConnectionAccepted(%this)
{
    // Called on the new connection object after connect() succeeds.
}
function GameConnection::onConnectionTimedOut(%this)
{
    // Called when an established connection times out
    disconnectedCleanup();
    MessageBoxOK( "TIMED OUT", "The server connection has timed out.");
}
```

```
function GameConnection::onConnectionDropped(%this, %msg)
{
    // Established connection was dropped by the server
    disconnectedCleanup();
    MessageBoxOK( "DISCONNECT", "The server has dropped the connection: " @ %msg);
}
function GameConnection::onConnectionError(%this, %msg)
{
    // General connection error, usually raised by ghosted objects
    // initialization problems, such as missing files.    We'll display
    // the server's connection error message.
    disconnectedCleanup();
    MessageBoxOK( "DISCONNECT", $ServerConnectionErrorMessage @ " (" @ %msg @ ")" );
}
//--------------------------------------------------------------------------
// Connection Failed Events
//--------------------------------------------------------------------------
function GameConnection::onConnectRequestRejected( %this, %msg )
{
    switch$(%msg)
    {
        case "CR_INVALID_PROTOCOL_VERSION":
            %error = "Incompatible protocol version: Your game version is not compatible with this server.";
        case "CR_INVALID_CONNECT_PACKET":
            %error = "Internal Error: badly formed network packet";
        case "CR_YOUAREBANNED":
            %error = "You are not allowed to play on this server.";
```

```
            case "CR_SERVERFULL":
                %error = "This server is full.";
            case "CHR_PASSWORD":
                // XXX Should put up a password-entry dialog.
                if ($Client::Password $= "")
                    MessageBoxOK( "REJECTED", "That server requires a password.");
                else {
                    $Client::Password = "";
                    MessageBoxOK( "REJECTED", "That password is incorrect.");
                }
                return;
            case "CHR_PROTOCOL":
                %error = "Incompatible protocol version: Your game version is not compatible with this server.";
            case "CHR_CLASSCRC":
                %error = "Incompatible game classes: Your game version is not compatible with this server.";
            case "CHR_INVALID_CHALLENGE_PACKET":
                %error = "Internal Error: Invalid server response packet";
            default:
                %error = "Connection error.  Please try another server.  Error code: (" @ %msg @ ")";
        }
        disconnectedCleanup();
        MessageBoxOK( "REJECTED", %error);
    }
    function GameConnection::onConnectRequestTimedOut(%this)
    {
        disconnectedCleanup();
```

```
        MessageBoxOK( "TIMED OUT", "Your connection to the server timed
out." );
    }
    //--------------------------------------------------------------------------
    // Disconnect
    //--------------------------------------------------------------------------
    function disconnect()
    {
        // Delete the connection if it's still there.
        if (isObject(ServerConnection))
            ServerConnection.delete();
        disconnectedCleanup();
        // Call destroyServer in case we're hosting
        destroyServer();
    }
    function disconnectedCleanup()
    {
        // Terminate all playing sounds
        alxStopAll();
        if (isObject(MusicPlayer))
            MusicPlayer.stop();
        // Back to the launch screen
        Canvas.setContent(MainMenuGui);
        // Dump anything we're not using
        clearTextureHolds();
        purgeResources();
    }
```

该文件的内容都是用来处理服务器连接信息的。这里重点介绍一下函数 disconnect()和 disconnectedCleanup()。先看下 disconnect()函数。

```
    function disconnect()
```

```
{
    if (isObject(ServerConnection))
        ServerConnection.delete();
    disconnectedCleanup();
    destroyServer();
}
```

if 语句判断在客户端是否还存在 ServerConnection 对象，如果是，则调用 delete()方法该对象删除。然后调用 disconnectedCleanup()函数，该函数稍后介绍。最后调用 destroyServer()函数，用来停止向服务器再发送请求。

```
function disconnectedCleanup()
{
    alxStopAll();
    if (isObject(MusicPlayer))
        MusicPlayer.stop();
    Canvas.setContent(MainMenuGui);
    clearTextureHolds();
    purgeResources();
}
```

alxStopAll()函数用来停止一切播放的声音；接下来的 if 语句判断是否存在 MusicPlayer 对象，如果是，调用 stop()方法停止播放声音。然后使用 Canvas.setContent(MainMenuGui)命令，返回到游戏主菜单界面。最后使用 clearTextureHolds()和 purgeResources()函数将还没有使用的材质、资源等倾卸掉。

注意：Canvas 代表的是我们整个游戏窗口（包括标题栏和菜单栏等）对象，使用 setContent()方法用来在主游戏窗口（不包括标题栏和菜单栏）显示括号里指定的 Gui 对象，这里是 MainMenuGui 对象，也就是游戏主菜单，因此，进入游戏场景里，就是使用 PlayGui 对象，也就是调用 Canvas.setContent(PlayGui)，理解了吧。

ui 文件夹

ui 文件夹下的内容包括 loadingGui.gui、mainMenuGui.gui、optionsDlg.gui、

playGui.gui 四个.gui 文件，几个图片文件和 buttons 文件夹。其中.gui 文件前面已经介绍过了，是保存图形用户接口元素的文件。几个图片文件都是游戏中用到的文件，其中 background.png 文件就是我们进入游戏前看到的背景图片，而 gameonebg.jpg 文件，相信您已经想起它了来了，对了，就是在 getting started 教程中，我们在制作 GameOne 游戏时候使用的背景图像，而那个 gray_bar.png 就是图标栏的灰色底，如下图 5.5 所示。Buttons 文件夹中放置的都是按钮图标文件，是的，它们是在进入 tutorial.base 之前，在游戏窗口上方显示的按钮图标， 如图 5.3 所示。

图 5.3　选项图标和灰色背景

注意：如果您仔细观察，您会发现，好像每个图标都有两个，而且几乎是一摸一样！唯一的区别就是图片命名上。是的，这两个图标代表的是按钮静止状态和鼠标放到上边的状态。尝试一下，是不是发现变化啦，呵呵。其中_h 文件表示鼠标放在按钮上边显示的图片文件，_n 文件表示静止状态的文件。

下面，我们来看看这些.gui 文件的内容，为了节省空间，这里不会粘贴全部的代码，大部分代码的内容都非常简单，前面章节已经进行了相应的介绍，这里具体针对一些重要的 gui 元素说明一下。

```
new GuiChunkedBitmapCtrl(MainMenuGui) {
    Profile = "GuiContentProfile";
    HorizSizing = "width";
    VertSizing = "height";
    position = "0 0";
    Extent = "640 480";
    MinExtent = "8 8";
    Visible = "1";
    bitmap = "./background";
    useVariable = "0";
    tile = "0";
        helpTag = "0";
```

```
new GuiChunkedBitmapCtrl() {
    Profile = "GuiDefaultProfile";
    HorizSizing = "center";
    VertSizing = "bottom";
    position = "42 0";
    Extent = "555 55";
    MinExtent = "8 2";
    Visible = "1";
    bitmap = "./gray_bar";
    useVariable = "0";
    tile = "0";
    new GuiBitmapButtonCtrl() {
        Profile = "GuiBorderButtonProfile";
        HorizSizing = "left";
        VertSizing = "top";
        position = "516 5";
        Extent = "31 43";
        MinExtent = "8 2";
        Visible = "1";
        Command = "quit();";
        tooltipprofile = "GuiToolTipProfile";
        tooltip = "Close down the engine";
        text = "Button";
        groupNum = "-1";
        buttonType = "PushButton";
        bitmap = "./buttons/exit";
    };
    ……
```

首先，定义了一个名为 MainMenuGui 的 GuiChunkedBitmapCtrl 控件，这个

控件是当我们运行游戏时候进入的一个界面，我们将其成为主菜单界面，如图 5.4 所示，其实就是该控件指定的背景图片 background。

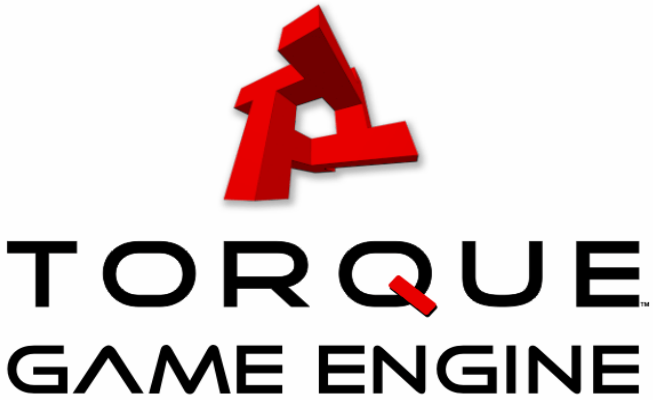


图 5.4　background 图

然后对该控件的所有属性赋值，内容相当简单明了。接下来在这个控件的内部同样又定义了一个没有名字的 GuiChunkedBitmapCtrl 控件，也就是那个灰色的线条，如图 5.5 所示。

图 5.5　灰色的线条背景

接下来，在这个灰色线条的控件里又使用 GuiBitmapButtonCtrl()定义了一系列的图标按钮文件。如 

按钮，表示退出键。其中属性 Command ＝ “quit()” 执行退出命令，tooltipprofile 指定为 GuiToolTipProfile 属性，也就是放到图标上会显示出类似帮助的说明文档的一种属性。Tooltip 的内容是具体的说明文档的内容，这里是 “ Close down the engine ” 。后面定义了一系列类似的按钮。

loadingGui.gui

```
new GuiControlProfile ("LoadingGuiContentProfile")
{
    opaque = true;
    fillColor = "200 200 200";
    border = true;
```

```
      borderColor      = "0 0 0";
};
//--- OBJECT WRITE BEGIN ---
new GuiChunkedBitmapCtrl(LoadingGui) {
      profile = "GuiContentProfile";
      horizSizing = "width";
      vertSizing = "height";
      position = "0 0";
      extent = "640 480";
      minExtent = "8 8";
      visible = "1";
      helpTag = "0";
      bitmap = "./background.jpg";
      useVariable = "0";
      tile = "0";
      new GuiControl() {
            profile = "LoadingGuiContentProfile";
            horizSizing = "center";
            vertSizing = "center";
            position = "30 110";
            extent = "455 308";
            minExtent = "8 8";
            visible = "1";
            helpTag = "0";
            new GuiTextCtrl(LOAD_MapName) {
                  profile = "GuiMediumTextProfile";
                  horizSizing = "right";
                  vertSizing = "bottom";
                  position = "7 6";
                  extent = "100 28";
```

```
      minExtent = "8 8";
      visible = "1";
      helpTag = "0";
      text = "Map Name";
      maxLength = "255";
   };
   new GuiMLTextCtrl(LOAD_MapDescription) {
      profile = "GuiMLTextProfile";
      horizSizing = "right";
      vertSizing = "bottom";
      position = "7 62";
      extent = "303 16";
      minExtent = "8 8";
      visible = "1";
      helpTag = "0";
      lineSpacing = "2";
      allowColorChars = "0";
      maxChars = "-1";
   };
   new GuiProgressCtrl(LoadingProgress) {
      profile = "GuiProgressProfile";
      horizSizing = "right";
      vertSizing = "top";
      position = "128 262";
      extent = "262 25";
      minExtent = "8 8";
      visible = "1";
      helpTag = "0";

      new GuiTextCtrl(LoadingProgressTxt) {
```

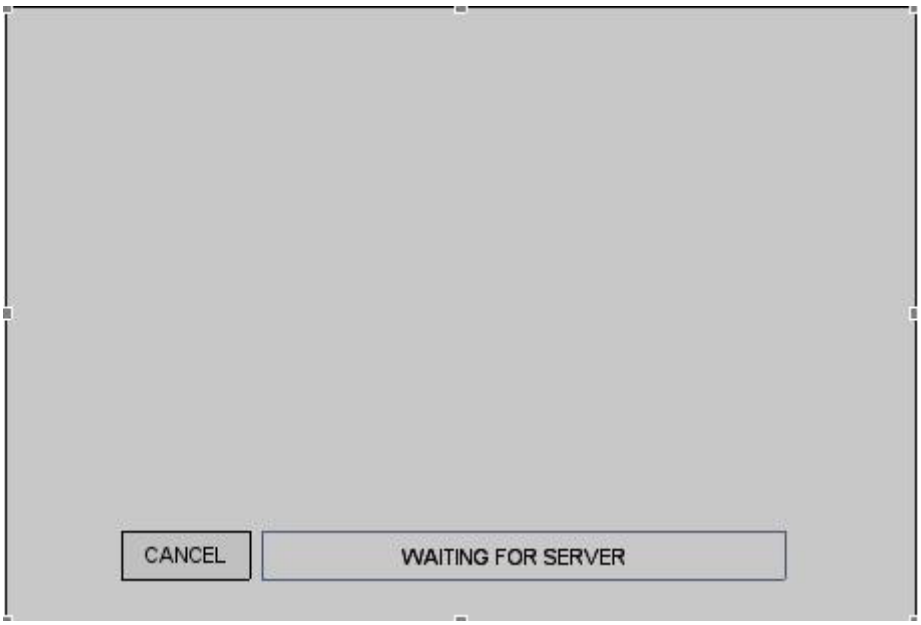
```
                profile = "GuiProgressTextProfile";
                horizSizing = "right";
                vertSizing = "bottom";
                position = "-4 3";
                extent = "262 20";
                minExtent = "8 8";
                visible = "1";
                helpTag = "0";
                text = "LOADING MISSION";
                maxLength = "255";
            };
        };
        new GuiButtonCtrl() {
            profile = "GuiButtonProfile";
            horizSizing = "right";
            vertSizing = "top";
            position = "58 262";
            extent = "65 25";
            minExtent = "20 20";
            visible = "1";
            command = "disconnect();";
            accelerator = "escape";
            helpTag = "0";
            text = "CANCEL";
        };
    };
};
//--- OBJECT WRITE END ---
```

loadingGui.gui 是在单击了WORLD EDITOR按钮后，在进入正式的游戏界面之前的

加载进度页面。

首先，使用 new 命令定义了一个新的名为 LoadingGuiContentProfile 的控件，同时进行了相关属性的赋值。

定义了 LoadingGui 控件，在该控件里面，定义里一个新的控件，而该控件的属性就说前面新定义的 LoadingGuiContentProfile 属性。加上后面定义的一些控件，构成了如图 所示的 GUI 元素。



接下来定义了两个 GuiMlTextCtrl 控件，其中一个名为 LOAD_MapName，属性是 GuiMediumTextProfile 类型，text 为“ Map Name ”，该控件的作用是显示 Map 的名字，而这里显示的实际上是后面介绍的 flat.mis 中 ScriptObject 中的名字，即“ F World ”。另一个控件名为 LOAD_MapDescription，属性是 GuiMLTextProfile 类型，该控件的作用是显示一个说明信息，内容为“ The tutorial base example mission. This mission contains a minimal set of example art and textures. ”。

也就是在 flat.mis 中的 simGroup 中的 desc0 定义的内容。如果您想将属于您自己的游戏的一些相关说明信息放在游戏的载入进度画面中，那么直接修改这里的内容就是您需要做的。

接下来，是定义了一个名为 LoadingProgress 的 GuiProgressCtrl 控件，其属性为 GuiProgressProfile，该控件的作用是提供了一个进度条，从它的名字就能看

出来，而进度条的具体执行过程是 common 文件夹中的内容，本书将不涉及到这部分内容。该控件中又包含了一个名为 LoadingProgressTxt 的 GuiTextCtrl 文本控件，其属性为 GuiProgressTextProfile 类型，该控件的作用是用来在进度条中显示字符，初始的字符为“ LOADING MISSION ”，之后会随着加载的内容变化而变化。

最后是一个 GuiButtonCtrl()按钮控件，您应该非常熟悉它了，它是用来取消连接的，通过“ command = "disconnect();"; ”来实现的。

啊，client 文件夹的内容终于学习完了，让我们放松放松，一起做个深呼吸，可能的话再来杯咖啡，呵呵。然后闭目养神一会，回想一下这些.cs 脚本文件以及.gui 文件，联系一下它们的关系，如果觉得有点乱，没关系，等准备好了再回到前面重新复习一下。

最后就剩下 data 文件夹了。正如 data 的含义那样，游戏中所需要的全部资源，包括游戏的模型、道具、材质、图片、声音等等全部都在该文件夹里。根据资源的类型，分为 interiors（建筑物）、missions（游戏的任务文件）、shapes（如岩石等小物品）、skies（天空盒子的相关信息）、sound（声音文件）、terrains（地形及地表材质信息）和 water（水资源）文件夹。这些是 tutorial.base 自带的资源，当然您可以根据您的需要组织您自己的资源文件夹结构。其基本内容如表 5.2 所示：

表 5.2　data 文件夹的内容

文件夹名	内　容
Interiors	游戏中的建筑物和其材质
Missions	游戏使用的任务文件
Shapes	游戏中的非建筑物和其材质
Skies	游戏使用的天空材质
Sound	游戏中使用的声音文件
Terrains	游戏中使用的地形文件及材质
water	游戏中使用的水的材质

这些内容都非常简单。下面我们稍微仔细的看一下 missions 文件夹。

打开 missions 文件夹，这里主要有两个文件，flat.mis 和 flat.ter 文件，如果您在这之前运行过了几次游戏，您可能还会看到类似 flat_9d4d64.ml 的一些文件，这是游戏运行时候自动生成的文件，您可以将这些文件想象成 flat.mis 文件编译后的执行文件，删除它们不会对游戏产生任何影响。

下面用你习惯的文本编辑软件打开 flat.mis 文件，你会看到如下内容，这里我用的是未进行任何修改的 tutorial.base 文件夹下的内容：

```
//--- OBJECT WRITE BEGIN ---
new SimGroup(MissionGroup) {
    new ScriptObject(MissionInfo) {
            desc0 = "The tutorial base example mission. This mission contains a minimal set of example art and textures.";
            name = "F World";
    };
    new MissionArea(MissionArea) {
        Area = "-360 -648 720 1296";
        flightCeiling = "300";
        flightCeilingRange = "20";
            locked = "true";
    };
    new Sky(Sky) {
        position = "336 136 0";
        rotation = "1 0 0 0";
        scale = "1 1 1";
        materialList = "~/data/skies/sky_day.dml";
        cloudHeightPer[0] = "0.349971";
        cloudHeightPer[1] = "0.25";
        cloudHeightPer[2] = "0.199973";
        cloudSpeed1 = "0.0001";
        cloudSpeed2 = "0.0002";
        cloudSpeed3 = "0.0003";
```

```
    visibleDistance = "500";
    fogDistance = "300";
    fogColor = "0.82 0.828 0.844 1";
    fogStorm1 = "0";
    fogStorm2 = "0";
    fogStorm3 = "0";
    fogVolume1 = "300 0 71";
    fogVolume2 = "0 0 0";
    fogVolume3 = "0 0 0";
    fogVolumeColor1 = "128 128 128 -2.22768e+038";
    fogVolumeColor2 = "128 128 128 0";
    fogVolumeColor3 = "128 128 128 -1.70699e+038";
    windVelocity = "1 1 0";
    windEffectPrecipitation = "1";
    SkySolidColor = "0.547 0.641 0.789 0";
    useSkyTextures = "1";
    renderBottomTexture = "0";
    noRenderBans = "0";
        locked = "true";
};
new Sun() {
    azimuth = "0";
    elevation = "35";
    color = "0.988 0.985 0.98 1";
    ambient = "0.5 0.5 0.5 1";
        rotation = "1 0 0 0";
        scale = "1 1 1";
        direction = "0.57735 0.57735 -0.57735";
        position = "0 0 0";
        locked = "true";
```

```
    };
    new TerrainBlock(Terrain) {
        rotation = "1 0 0 0";
        scale = "1 1 1";
        terrainFile = "./flat.ter";
        squareSize = "8";
        emptySquares = "99744 443522 443778 444034";
        bumpScale = "1";
        bumpOffset = "0.01";
        zeroBumpScale = "8";
        tile = "1";
            position = "-1024 -1024 0";
            locked = "true";
    };
    new SimGroup(PlayerDropPoints) {
        new SpawnSphere() {
            position = "23.1063 -410.857 166.036";
            rotation = "0 0 1 130.062";
            scale = "1 1 1";
            dataBlock = "SpawnSphereMarker";
            radius = "100";
            sphereWeight = "100";
            indoorWeight = "100";
            outdoorWeight = "100";
                lockCount = "0";
                locked = "false";
                homingCount = "0";
        };
    };
};
```

//--- OBJECT WRITE END ---

这里边先创建一个名为 MissionGroup 的 SimGroup 对象，是不是很熟悉了？对，在 world editor inspector 中我们看到的带+号的根文件夹就是这个！而且是通过 SimGroup 对象进行组织所有游戏对象的。之后大括号中的内容都包含在该 MissionGroup 对象里。接下来定义了一个名叫 MissionInfo 的 ScriptObject 对象，这个只是用来描述一下任务信息，没有什么特殊的地方，您可以直接修改 desc0 后面的内容为您的内容。

接着创建了名为 MissionArea 的 MissionArea 对象实例，虽然名字是一样的，但是表示的意义完全不同。

然后是创建名为 Sky 的 Sky 对象实例，里面定义了该对象实例的成员变量。具体内容参考数据块那部分内容。

继续定义了 Sun、Terrain 对象实例，最后为文家出生点创建了一个 SimGroup，您可以按照 SpawnSphere()中的内容添加更多的出生点，这样每次进入游戏就会出现在不同的地方（这是随机选择的），是不是有些游戏的味道了？

关于 flat.ter 文件没有太多可说的内容，它是专门用来保存地形信息的，当你在游戏世界中设置好了你的地形，使用 File→Save Mission as 命令会自动保存地形信息到 flat.ter 文件中。很遗憾，该文件不能用普通的文本编辑软件查看，说实话，我们根本不用去关心里边到底是什么。

5.3 本章小结

好了，到目前为止，我们已经将 tutorial.base 文件夹的内容全部学习了一遍。下一章，我们将继续学习进阶游戏 starter.fps 的内容。

第六章 进阶游戏 starter.fps

经过前面对 tutorial.base 文件夹的学习，我们已经知道如何使用 Torque 搭建自己的游戏了。下面让我们来学习一下内容更多的 starter.fps 游戏（它已经可以认为是一个非常不错的游戏了）中的内容，这个 demo 包含了大量 fps 游戏中具备的因素，掌握它的内容对制作属于我们自己的游戏非常有帮助。

首先，让我们看一下 starter.fps 文件夹同 example 文件夹的关系（省去了一些脚本文件），如图 6.1 所示：

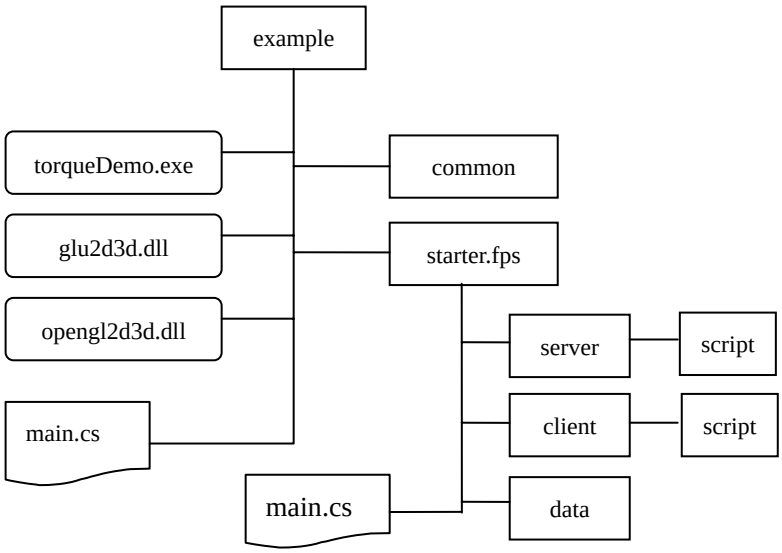


图 6.1　starter.fps 文件夹同 example 文件夹的关系

基本结构同 tutorial.base 一致，只是在 server 和 client 文件夹下进行了一个小的变体，新建立了一个 scripts 文件夹用来统一保存所有的脚本文件，这是完全符合要求的，只要您在使用 exec 加载这些脚本文件的时候将文件夹的路径指定正确即可，并且显得文件组织结构更加整洁。

下面开始我们的学习。

6.1 starter.fps 文件夹

首先要改变一下根 main.cs 文件中第一行的代码为“ $defaultGame = "starter.fps"; ”，相信您现在已经非常熟悉它了。然后打开 starter.fps 文件夹下的

main.cs 文件。

6.1.1 main.cs 文件

您会发现这个文件的内容同 tutorial.base 文件夹下的 main.cs 文件几乎一样！这里需要说明的是 FpsStarterKit 包。这个包同 tutorial.base 文件夹下的 main.cs 文件里的 ttb 包的作用一样，只是名字不同，同时 FpsStarterKit 包中定义了更多的内容，以下是它的内容：

```
package FpsStarterKit {
function displayHelp() {
    Parent::displayHelp();
    error(
        "FPS Mod options:\n"@
        "    -dedicated                 Start as dedicated server\n"@
        "    -connect <address>         For non-dedicated: Connect to a game at
<address>\n" @
        "    -mission <filename>        For dedicated: Load the mission\n"
    );
}
function parseArgs()
{
    Parent::parseArgs();
    // Arguments, which override everything else.
    for (%i = 1; %i < $Game::argc ; %i++)
    {
        %arg = $Game::argv[%i];
        %nextArg = $Game::argv[%i+1];
        %hasNextArg = $Game::argc - %i > 1;
        switch$ (%arg)
        {
```

```
                //--------------------
                case "-dedicated":
                    $Server::Dedicated = true;
                    enableWinConsole(true);
                    $argUsed[%i]++;
                //--------------------
                case "-mission":
                    $argUsed[%i]++;
                    if (%hasNextArg) {
                        $missionArg = %nextArg;
                        $argUsed[%i+1]++;
                        %i++;
                    }
                    else
                        error("Error: Missing Command Line argument. Usage: -mission <filename>");
        //--------------------
                case "-connect":
                    $argUsed[%i]++;
                    if (%hasNextArg) {
                        $JoinGameAddress = %nextArg;
                        $argUsed[%i+1]++;
                        %i++;
                    }
                    else
                        error("Error: Missing Command Line argument. Usage: -connect <ip_address>");
                }
            }
        }
```

```
function onStart()
{
    Parent::onStart();
    echo("\n--------- Initializing MOD: FPS Starter Kit ---------");
// Load the scripts that start it all...
    exec("./client/init.cs");
    exec("./server/init.cs");
    exec("./data/init.cs");
// Server gets loaded for all sessions, since clients
// can host in-game servers.
    initServer();
// Start up in either client, or dedicated server mode
    if ($Server::Dedicated)
        initDedicated();
    else
        initClient();
}
function onExit()
{
    echo("Exporting client prefs");
    export("$pref::*", "./client/prefs.cs", False);
    echo("Exporting client config");
    if (isObject(moveMap))
        moveMap.save("./client/config.cs", false);
    echo("Exporting server prefs");
    export("$Pref::Server::*", "./server/prefs.cs", False);
    BanList::Export("./server/banlist.cs");
    Parent::onExit();
}
}; // Client package
```

activatePackage(FpsStarterKit);

函数 displayHelp()是在游戏无法正常运行时候，在 console.log 文件中输出错误的帮助信息。

函数 parseArgs()用来逐个检查命令参数，并通过用户提供的命令参数执行任务。这里主要参数为-dedicated、–mission 和-connect。这个需要解释一下，当我们正常双击 torqueDemo.exe 可执行文件的时候，命令参数是已经设置好的（即“ torqueDemo.exe –game starter.fps ”，只是我们没有看到这个命令行而已），直接就可以进入游戏。

注意：其实我们是完全可以看到这个命令行的调用的，只要您在 example 文件夹下运行相应的.bat 文件，如 starter.fps.bat 文件，将会看到 dos 命令提示符下的运行命令行参数的情况，如图 6.2 所示：

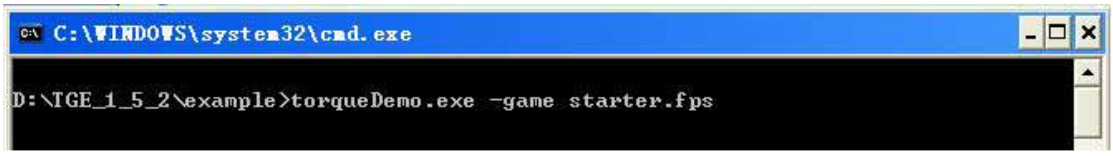


图 6.2　TGE 运行调用的命令行参数

而上述命令是需要我们是使用 XP 系统下的开始→运行，在运行的窗口下我们可以输入类似“ torqueDemo.exe -dedicated -mission *\data\missions*.mis ”命令，这样 Torque 就会根据命令参数为先建立一个专用服务器，然后加载-mission 参数后面的路径指定的任务文件。如果您对这些代码感到有点乱，是因为您对程序还不够熟悉，可以先把它放在一遍，只要知道这段代码是实现这个功能就好了。随着以后不断练习、不断熟练，您就会慢慢掌握它。

函数 onStart()开始加载一系列脚本文件，然后启动服务器端，这里同 totorial.base 不一样的地方就是，如果$Server::Dedicated 变量为真，就开启专用服务器模式，否则就开启客户端模式。而$Server::Dedicated 是在前面的 case “-dedicated”语句成立时候赋值的。

函数 onExit()在游戏结束的时候进行一系列的保存操作。

activatePackage(FpsStarterKit);语句用来激活 FpsStarterKit 包。

6.1.2 server 文件夹

接着，我们一切学习一下 server 文件夹的内容。如果您是在访问 starter.fps\server 文件夹之前没有运行过该游戏 demo，那么文件夹下只有 init.cs、defaults.cs 文件和 scripts 文件夹。如果您运行过该游戏 demo，那么您除了得到许多.dso 文件之外，还会发现新增加了两个脚本文件，banlist.cs 和 sprefs.cs 文件。这两个文件都是游戏运行后，在结束游戏的时候使用脚本命令创建的，而它们的内容并不难以理解，都是游戏中的一些全局变量，相信您通过观察那些变量的命名就可以猜出它们代表的含义了。

init.cs

下面我们重点看一下 init.cs 文件，init.cs 文件内容如下：

```
function initServer()
{
    echo("\n--------- Initializing MOD: FPS Starter Kit: Server ---------");
    // Server::Status is returned in the Game Info Query and represents the
    // current status of the server. This string sould be very short.
    $Server::Status = "Unknown";
    // Turn on testing/debug script functions
    $Server::TestCheats = false;
    // Specify where the mission files are.
    $Server::MissionFileSpec = "*/missions/*.mis";
    // The common module provides the basic server functionality
    initBaseServer();
    // Load up game server support scripts
    exec("./scripts/commands.cs");
    exec("./scripts/centerPrint.cs");
    exec("./scripts/game.cs");
}
//------------------------------------------------------------------------
```

```
function initDedicated()
{
    enableWinConsole(true);
    echo("\n--------- Starting Dedicated Server ---------");
    // Make sure this variable reflects the correct state.
    $Server::Dedicated = true;
    // The server isn't started unless a mission has been specified.
    if ($missionArg !$= "") {
        createServer("MultiPlayer", $missionArg);
    }
    else
        echo("No mission specified (use -mission filename)");
}
```

函数 initServer()用来初始化服务器。开始进行几个全局变量的初始化，$Server::MissionFileSpec = "*/missions/*.mis";命令指定了需要加载的任务文件的磁盘路径信息，因此如果您在 missions 文件夹下有多个.mis 文件，当您进入游戏时，会在游戏列表中看到相应多的任务信息。initBaseServer()函数运行 common 文件夹下的该函数，然后加载一系列脚本文件。

函数 initDedicated()开启专用服务器模式。说到这里，您可能会非常疑惑，并问，“ Torque 不就是设计成服务器/客户机运行的模式么？我们在运行游戏的时候不就建立了一个服务器么？这个专用服务器又是什么意思呢？”很好，起初我也同您一样存在疑问。其实，这里所说的专用服务器是指仅运行游戏的服务器端程序而不必运行游戏的图形界面。这样我们就可以用一台旧的、没有 3D 图形加速卡的机器作为游戏服务器，服务器仅仅用来完成与各个客户机的信息传递，因此这类低档次的机器也完全可以胜任。了解了么？

scripts 文件夹

有了前面的基础，这些内容很容易吧，下面就是 scripts 文件夹了。如果您在此之前已经运行过该游戏 demo，您会发现在 script 文件夹下有很多.dso 文件，这就是游戏自动生成的二进制文件，如果您觉得它们有点妨碍您的工作，没关系，

请运行 example 文件夹下的 DeleteDSOs.bat 文件，屏幕会弹出一个 dos 命令窗口，稍微闪动几下，就会关闭，好了，您会发现所有的.dso 文件全都删除了。请放心，这不会影响您的任何工作，当您再次运行游戏时候它们又会自动生成出来。

Script 文件夹下共有 21 个文件！看来我们的工作量不小，这一次，我将不会全部分析它们，一些简单的应该留给您自己去消化、理解，我相信这将对您的进步非常有帮助。

game.cs

首先来看 game.cs 文件，它是最先调用的，该文件的内容同 toturial.base 下的 game.cs 内容相仿，但是该文件中增加了一些新的内容，提供更多的功能。

```
//-----------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------------
$Game::Duration = 20 * 60;
$Game::EndGameScore = 30;
$Game::EndGamePause = 10;
//    Functions that implement game-play
function onServerCreated()
{
    $Server::GameType = "FPS Starter Kit";
    $Server::MissionType = "Deathmatch";
    $Game::StartTime = 0;
    exec("./audioProfiles.cs");
    exec("./envAudioProfiles.cs");
    exec("./camera.cs");
    exec("./markers.cs");
    exec("./triggers.cs");
    exec("./inventory.cs");
```

```
    exec("./shapeBase.cs");
    exec("./item.cs");
    exec("./environment.cs");
    exec("./health.cs");
    exec("./staticShape.cs");
    exec("./weapon.cs");
    exec("./radiusDamage.cs");
    exec("./crossbow.cs");
    exec("./environment.cs");
    exec("common/server/lightingSystem.cs");
    exec("./player.cs");
    exec("./chimneyfire.cs");
    exec("./aiPlayer.cs");
    exec("./sgExamples.cs");
    // Keep track of when the game started
    $Game::StartTime = $Sim::Time;
}
function onServerDestroyed()
{
    // This function is called as part of a server shutdown.
}
function onMissionLoaded()
{
    startGame();
}
function onMissionEnded()
{
    cancel($Game::Schedule);
    $Game::Running = false;
    $Game::Cycling = false;
```

```
}
function startGame()
{
    if ($Game::Running) {
        error("startGame: End the game first!");
        return;
    }
    for( %clientIndex = 0; %clientIndex < ClientGroup.getCount();
%clientIndex++ ) {
        %cl = ClientGroup.getObject( %clientIndex );
        commandToClient(%cl, 'GameStart');
        %cl.score = 0;
    }
    if ($Game::Duration)
        $Game::Schedule = schedule($Game::Duration * 1000, 0,
"onGameDurationEnd" );
    $Game::Running = true;
    new ScriptObject(AIManager) {};
    MissionCleanup.add(AIManager);
    AIManager.think();
}
function endGame()
{
    if (!$Game::Running)  {
        error("endGame: No game running!");
        return;
    }
    AIManager.delete();
    cancel($Game::Schedule);
    for( %clientIndex = 0; %clientIndex < ClientGroup.getCount();
```

```
%clientIndex++ ) {
            %cl = ClientGroup.getObject( %clientIndex );
            commandToClient(%cl, 'GameEnd');
        }
        resetMission();
        $Game::Running = false;
    }
    function onGameDurationEnd()
    {
        if ($Game::Duration && !isObject(EditorGui))
            cycleGame();
    }
    function cycleGame()
    {
        if (!$Game::Cycling) {
            $Game::Cycling = true;
            $Game::Schedule = schedule(0, 0, "onCycleExec");
        }
    }
    function onCycleExec()
    {
        endGame();
        $Game::Schedule = schedule($Game::EndGamePause * 1000, 0,
"onCyclePauseEnd");
    }
    function onCyclePauseEnd()
    {
        $Game::Cycling = false;
        %search = $Server::MissionFileSpec;
        for (%file = findFirstFile(%search); %file !$= ""; %file =
```

```
findNextFile(%search)) {
            if (%file $= $Server::MissionFile) {
                // Get the next one, back to the first if there
                // is no next.
                %file = findNextFile(%search);
                if (%file $= "")
                   %file = findFirstFile(%search);
                break;
            }
        }
        loadMission(%file);
    }
    function GameConnection::onClientEnterGame(%this)
    {
        commandToClient(%this, 'SyncClock', $Sim::Time - $Game::StartTime);
        %this.camera = new Camera() {
            dataBlock = Observer;
        };
        MissionCleanup.add( %this.camera );
        %this.camera.scopeToClient(%this);
        %this.score = 0;
        %this.spawnPlayer();
    }
    function GameConnection::onClientLeaveGame(%this)
    {
        if (isObject(%this.camera))
            %this.camera.delete();
        if (isObject(%this.player))
            %this.player.delete();
    }
```

```
function GameConnection::onLeaveMissionArea(%this)
{
}
function GameConnection::onEnterMissionArea(%this)
{
}
function GameConnection::onDeath(%this, %sourceObject, %sourceClient, %damageType, %damLoc)
{
    %this.player.setShapeName("");
    if (isObject(%this.camera) && isObject(%this.player)) {
        %this.camera.setMode("Corpse",%this.player);
        %this.setControlObject(%this.camera);
    }
    %this.player = 0;
    // Doll out points and display an appropriate message
    if (%damageType $= "Suicide" || %sourceClient == %this) {
        %this.incScore(-1);
        messageAll('MsgClientKilled','%1 takes his own life!',%this.name);
    }
    else {
        %sourceClient.incScore(1);
        messageAll('MsgClientKilled','%1 gets nailed by %2!',%this.name,%sourceClient.name);
        if (%sourceClient.score >= $Game::EndGameScore)
            cycleGame();
    }
}
function GameConnection::spawnPlayer(%this)
{
```

```
    // Combination create player and drop him somewhere
    %spawnPoint = pickSpawnPoint();
    %this.createPlayer(%spawnPoint);
}
function GameConnection::createPlayer(%this, %spawnPoint)
{
    if (%this.player > 0)  {
        error( "Attempting to create an angus ghost!" );
    }
    %player = new Player() {
        dataBlock = PlayerBody;
        client = %this;
    };
    MissionCleanup.add(%player);
    %player.setTransform(%spawnPoint);
    %player.setShapeName(%this.name);
    %player.setInventory(Crossbow,1);
    %player.setInventory(CrossbowAmmo,10);
    %player.mountImage(CrossbowImage,0);
    %this.camera.setTransform(%player.getEyeTransform());
    %this.player = %player;
    %this.setControlObject(%player);
}
function pickSpawnPoint()
{
    %groupName = "MissionGroup/PlayerDropPoints";
    %group = nameToID(%groupName);
    if (%group != -1) {
        %count = %group.getCount();
        if (%count != 0) {
```

```
            %index = getRandom(%count-1);
            %spawn = %group.getObject(%index);
            return %spawn.getTransform();
        }
        else
            error("No spawn points found in " @ %groupName);
    }
    else
        error("Missing spawn points group " @ %groupName);
    // Could be no spawn points, in which case we'll stick the
    // player at the center of the world.
    return "0 0 300 1 0 0 0";
}
```

$Game::Duration 变量保存满足结束游戏条件需要经过的时间，单位为秒，20×60 说明游戏运行 20 分钟将自动结束，同时询问是否进行新的游戏。使用 20×60 而不是直接使用 1200，主要是因为直观，一目了然。

$Game::EndGameScore 变量保存满足结束游戏条件需要的分数。在 starter.fps 游戏里，每杀死一个敌人将得到 1 分，得到 30 分游戏会自动结束，询问是否进行新的游戏。

$Game::EndGamePause = 10;

函数 OnserverCreated()，在服务器创建后调用，它负责管理服务器启动并运行之后发生的所有事情。

$Server::GameType 变量定义了游戏的类型，这里是“ FPS Starter Kit ”类型，该变量会在服务器创建后发送给主服务器，用来说明您的服务器提供的游戏类型。

$Server::MissionType 变量定义了游戏的任务类型，这里是“ Deathmatch ”类型，表明是一种死亡竞技游戏。

$Game::StartTime 变量定义了游戏开始的时间，它的作用是用来计时的，计算游戏运行经过时间，一旦这个时间同前面$Game::Duration 定义的时间相同，

则游戏结束。

接下来使用 exec 命令加载很多的脚本文件，相信您对这样的操作已经相当熟悉了。

函数 onServerDestroyed()在游戏服务器关闭时候调用，这里暂时没有任何内容。

函数 onMissionLoaded()在任务文件加载完成后由 loadMission()函数调用，它的内容非常简单，仅仅调用 startGame()函数。

函数 onMissionEnded()在任务结束或失败时候由 endMission()函数调用，进行一些变量设置。

函数 startGame()安排了在每次启动游戏时候需要的操作。首先判断游戏是否已经运行，如果是则需要关闭游戏。然后循环访问保存在服务器端的客户端的句柄，使用它来通知每个客户端开始游戏，这里使用的是 commandToClient()命令，并将每个客户端的分数清零。接下来的 if 语句用来判断游戏是否有规定的结束时间$Game::Duration，没有则继续游戏，游戏不会因为时间因素而结束。如果有，则使用 Schedule 函数启动游戏的定时器。

Schedule 是一个相当重要的函数，我们要多花一点时间看看它。调用该函数的语法是：

%event = Schedule(time, reference, command, <param1… paramN>)

该函数将在 time 毫秒内对一个事件进行时间安排，并执行带参数的 command。如果 reference 不是 0，那么需要确保 reference 被赋值为一个有效的对象句柄。如果这个对象被删除了，如果它还没有被触发的话，这个安排好的事件就会被丢弃。Schedule 函数返回一个事件的 ID 号，可以使用这个 ID 跟踪安排好的事件，或者在该事件发生前用这个 ID 来取消它。

由于我们的游戏中有一个 20 分钟的限制，那么该函数生成的事件就会在 20 分钟后调用 onGameDurationEnd 函数来结束游戏，这就是为什么游戏运行一段时间会结束的原因，懂了么？

最后，创建了一个名为 AIManager 的 ScriptObject 对象，然后使用 MissionCleanup 的 add 方法将该对象添加进去，最后调用 think()方法。

以上就是函数 OnserverCreated()的全部工作。

函数 endGame()负责处理游戏结束时的工作。调用 delete()方法停止 AIManager 对象，使用 cancle 命令取消游戏计时器。接着，通知所有客户机调用 GameEnd 函数。释放所有临时的任务对象，$Game::Running 变量设置为 false。

函数 onGameDurationEnd()用来在满足一定条件时候循环游戏，调用下面的 cycleGame()函数。

函数 cycleGame()在满足条件下调用接下来 onCycleExec 函数。

函数 onCycleExec()函数首先调用 endGame()函数停止游戏，然后显示一个类似 CS 的游戏得分的屏显，持续的时间为$Game::EndGamePause×1000，单位为毫秒，然后调用 onCyclePauseEnd()函数。

函数 onCyclePauseEnd()的作用时加载新的游戏任务文件，使用 loadMission()函数。

下面是一系列重要的 GameConnection 的方法，大部分我们在 tutorial.base 中已经接触到了。

方法 onClientEnterGame()将游戏玩家的化身添加到游戏中。

方法 onClientLeaveGame()处理客户端离开服务器的工作。

方法 onLeaveMissionArea()和 onEnterMissionArea 处理玩家离开和进入游戏任务区发生的事件。这里什么都不做，对于我们的游戏来说，大部分情况下，应该保证玩家一直呆在任务区内，因为离开任务区将导致一些行为失效。使用栅栏、围墙或高山等限制玩家的活动区域在任务区内是个不错的方法。

方法 onDeath()在玩家死亡时候调用，它需要传递 5 个参数，分别是%this，GameConnection 对象的句柄； %sourceObject，杀死玩家的工具（如武器）句柄；%sourceClient，杀死玩家的玩家客户端句柄；%damageType，杀死玩家的伤害类型；%damLoc，伤害值。该方法的作用是当玩家所受到的伤害超过一定的数量，那么将从玩家的 Damage 处理器中调用 GameConnection::onDeath()方法。首先将玩家化身的名字清空，然后将客户机切换到死亡视角并将控制权交给摄像机，这样就同玩家的化身尸体脱钩。然后将客户端的玩家数量设置为 0，知道它决定再次加入游戏。接下来判断一下玩家的伤害类型和造成该伤害的输出客户端，如果伤害类型为“ 自杀 ”或者伤害输出客户端是玩家自己，则将玩家客户端的分数减 1，同时通知所有玩家“ * takes his own life ”，其中*表示玩家的名字，来自

%this.name 变量。否则将杀死该玩家化身的玩家客户端分数加 1，然后通知所有玩家“ * get nailed by ** ”，其中*来自于第一个变量%this.name，而**来自于第二个变量%sourceClient.name。最后判断得分的这个客户端的分数是否达到分数上限，如果是就调用函数 cycyleGame()循环游戏。

方法 SpawnPlayer(%this)用来在指定的玩家化身出生点处创建玩家。

方法 createPlayer(%this, %spawnPoint)中的大部分代码我们在 tutorial.base 中已经见过，相信不会对您什么迷惑。稍微不同的是这里使用 setInventory()方法为游戏玩家化身初始化了一些装备。%player.setInventory(Crossbow,1);初始化玩家携带的 Crossbow（弩）的数量为 1。

%player.setInventory(CrossbowAmmo,10);初始化玩家携带的 CrossbowAmmo（弩箭）的数量为 10。

然后使用%player.mountImage(CrossbowImage,0);绑定 CrossbowImage(弩的图片)，该操作的结果是在屏幕上显示我们的武器（弩）的图片，很多 FPS 游戏都是这样的，不是么？

最后一个函数 pickSpawnPoint()同 tutorial.base 中的该函数没有任何区别！

Player.cs

Player.cs 定义了所有同玩家角色化身有关的信息。为了节省空间，今后将不在粘贴全部的脚本代码，同时一些简单意义的代码也将留给读者自己去思考和理解，您自己读懂代码将比听我唠叨增加更多的经验值，这对以后您自己独立思考和制作游戏大有帮助！下面让我们一起看下我认为需要讲解的代码。

$CorpseTimeoutValue 变量用来保存玩家的尸体趴在地上直到消失的时间，单位毫秒。

$PlayerDeathAnim::*定义了 11 种玩家角色化身死亡的方式，蛮多的，呵呵。

接下来定义了相当多的 AudioProfile 声音数据块对象，用来处理游戏中各种同玩家化身相关的声音，语法相当简单，相信已经难不到您了。

然后，又定义了相当多的 ParticleData 粒子数据块和相应的 ParticleEmmiterData 粒子发射器数据块，用来表现玩家化身在游戏中运动时候产生的效果，如进入水里溅起的水花等等。如下面的代码所示：

datablock ParticleData(PlayerSplashMist)

```
{
   dragCoefficient          = 2.0;
   gravityCoefficient     = -0.05;
   inheritedVelFactor     = 0.0;
   constantAcceleration = 0.0;
   lifetimeMS                  = 400;
   lifetimeVarianceMS     = 100;
   useInvAlpha                = false;
   spinRandomMin             = -90.0;
   spinRandomMax             = 500.0;
   textureName                = "~/data/shapes/player/splash";
   colors[0]        = "0.7 0.8 1.0 1.0";
   colors[1]        = "0.7 0.8 1.0 0.5";
   colors[2]        = "0.7 0.8 1.0 0.0";
   sizes[0]         = 0.5;
   sizes[1]         = 0.5;
   sizes[2]         = 0.8;
   times[0]         = 0.0;
   times[1]         = 0.5;
   times[2]         = 1.0;
};
datablock ParticleEmitterData(PlayerSplashMistEmitter)
{
   ejectionPeriodMS = 5;
   periodVarianceMS = 0;
   ejectionVelocity = 3.0;
   velocityVariance = 2.0;
   ejectionOffset     = 0.0;
   thetaMin              = 85;
   thetaMax              = 85;
```

```
    phiReferenceVel    = 0;
    phiVariance        = 360;
    overrideAdvance = false;
    lifetimeMS         = 250;
    particles = "PlayerSplashMist";
};
```

这些数据块只要按照数据块的属性进行相应的定义就行了，没有太多的难度，就是数量比较多，占用了很大篇幅。

数据块 PlayerData（playerBody）定义了我们游戏中玩家化身的数据块。内容如下：

```
datablock PlayerData(PlayerBody)
{
    renderFirstPerson = false;
    emap = true;

    className = Armor;
    shapeFile = "~/data/shapes/player/player.dts";
    cameraMaxDist = 3;
    computeCRC = true;

    canObserve = true;
    cmdCategory = "Clients";

    cameraDefaultFov = 90.0;
    cameraMinFov = 5.0;
    cameraMaxFov = 120.0;

    debrisShapeName = "~/data/shapes/player/debris_player.dts";
    debris = playerDebris;
```

```
aiAvoidThis = true;

minLookAngle = -1.4;
maxLookAngle = 1.4;
maxFreelookAngle = 3.0;

mass = 90;
drag = 0.3;
maxdrag = 0.4;
density = 10;
maxDamage = 100;
maxEnergy =    60;
repairRate = 0.33;
energyPerDamagePoint = 75.0;

rechargeRate = 0.256;

runForce = 48 * 90;
runEnergyDrain = 0;
minRunEnergy = 0;
maxForwardSpeed = 14;
maxBackwardSpeed = 13;
maxSideSpeed = 13;

maxUnderwaterForwardSpeed = 8.4;
maxUnderwaterBackwardSpeed = 7.8;
maxUnderwaterSideSpeed = 7.8;

jumpForce = 8.3 * 90;
jumpEnergyDrain = 0;
```

```
minJumpEnergy = 0;
jumpDelay = 15;

recoverDelay = 9;
recoverRunForceScale = 1.2;

minImpactSpeed = 45;
speedDamageScale = 0.4;

boundingBox = "1.2 1.2 2.3";
pickupRadius = 0.75;

// Damage location details
boxNormalHeadPercentage       = 0.83;
boxNormalTorsoPercentage      = 0.49;
boxHeadLeftPercentage         = 0;
boxHeadRightPercentage        = 1;
boxHeadBackPercentage         = 0;
boxHeadFrontPercentage        = 1;

// Foot Prints
decalData   = PlayerFootprint;
decalOffset = 0.25;

footPuffEmitter = LightPuffEmitter;
footPuffNumParts = 10;
footPuffRadius = 0.25;

dustEmitter = LiftoffDustEmitter;
```

```
splash = PlayerSplash;
splashVelocity = 4.0;
splashAngle = 67.0;
splashFreqMod = 300.0;
splashVelEpsilon = 0.60;
bubbleEmitTime = 0.4;
splashEmitter[0] = PlayerFoamDropletsEmitter;
splashEmitter[1] = PlayerFoamEmitter;
splashEmitter[2] = PlayerBubbleEmitter;
mediumSplashSoundVelocity = 10.0;
hardSplashSoundVelocity = 20.0;
exitSplashSoundVelocity = 5.0;

// Controls over slope of runnable/jumpable surfaces
runSurfaceAngle    = 70;
jumpSurfaceAngle = 80;

minJumpSpeed = 20;
maxJumpSpeed = 30;

horizMaxSpeed = 68;
horizResistSpeed = 33;
horizResistFactor = 0.35;

upMaxSpeed = 80;
upResistSpeed = 25;
upResistFactor = 0.3;

footstepSplashHeight = 0.35;
```

```
//NOTE:    some sounds commented out until wav's are available

// Footstep Sounds
FootSoftSound             = FootLightSoftSound;
FootHardSound              = FootLightHardSound;
FootMetalSound            = FootLightMetalSound;
FootSnowSound              = FootLightSnowSound;
FootShallowSound         = FootLightShallowSplashSound;
FootWadingSound           = FootLightWadingSound;
FootUnderwaterSound    = FootLightUnderwaterSound;

//FootBubblesSound         = FootLightBubblesSound;
//movingBubblesSound      = ArmorMoveBubblesSound;
//waterBreathSound         = WaterBreathMaleSound;

//impactSoftSound          = ImpactLightSoftSound;
//impactHardSound           = ImpactLightHardSound;
//impactMetalSound         = ImpactLightMetalSound;
//impactSnowSound           = ImpactLightSnowSound;

//impactWaterEasy          = ImpactLightWaterEasySound;
//impactWaterMedium        = ImpactLightWaterMediumSound;
//impactWaterHard          = ImpactLightWaterHardSound;

groundImpactMinSpeed       = 10.0;
groundImpactShakeFreq      = "4.0 4.0 4.0";
groundImpactShakeAmp       = "1.0 1.0 1.0";
groundImpactShakeDuration = 0.8;
groundImpactShakeFalloff = 10.0;
```

```
    //exitingWater                = ExitingWaterLightSound;

    observeParameters = "0.5 4.5 4.5";

    // Allowable Inventory Items
    maxInv[BulletAmmo] = 20;
    maxInv[HealthKit] = 1;
    maxInv[RifleAmmo] = 100;
    maxInv[CrossbowAmmo] = 50;
    maxInv[Crossbow] = 1;
    maxInv[Rifle] = 1;
};
```

其中：

renderFirstPerson = fale 表明在第一人称视角下无论如何转动鼠标都不渲染玩家化身，也就是说玩家将看不到自己身体的任何一个部分，如手、脚等。

Emap = true 表面允许把玩家化身模型映射到环境贴图上，就是产生影子。

className 是对象的类名，这里是 Armor。

shapeFile 指定了玩家化身模型在磁盘中的位置。

cameraMaxDist 定义了第三人称视角下追为的摄像机远离玩家身体的距离。

computerCRC 指定 CRC 循环校验。

剩下的是 Player 类的属性定义。

最后面的 MaxInv[*]中的内容是动态域，定义了玩家化身可以捡起的物品的最大数量，可以直接修改它们变成您希望的值，或者添加更多的新的动态域，来表示其它物品。

其它所有属性都是在 PlayerData 中有定义的，这里做了一个初始化而已。可以参照着数据块那章看。

定义完游戏玩家化身的数据块，就开始为这个数据块定义一系列方法（函数）。

```
function Armor::onAdd(%this,%obj)
{
```

```
    // Vehicle timeout
    %obj.mountVehicle = true;
    // Default dynamic armor stats
    %obj.setRechargeRate(%this.rechargeRate);
    %obj.setRepairRate(0);
//%light = new volumeLight() {
    //     dataBlock = "sgMountLight";
    //              rotation = "-0.357694 0.933839 9.9834e-009 180";
    //              scale = "1 1 1";
    //              dataBlock = "sgMountLight";
    //              Enable = "1";
    //              IconSize = "1";
    //              ParticleColorAttenuation = "1";
    //              Texture = "common/lighting/lightFalloffMono.png";
    //              lpDistance = "0.35";
    //              ShootDistance = "5";
    //              Xextent = "0.6";
    //              Yextent = "0.6";
    //              SubdivideU = "4";
    //              SubdivideV = "4";
    //              FootColour = "1.000000 1.000000 1.000000 0.182000";
    //              TailColour = "0.000000 0.000000 0.000000 0.000000";
    //};

    //%light.attachtoobject(%obj);
    //%obj.light = %light;
}
```

方法 Armor::onAdd(%this, %obj)在向游戏添加新的化身实例的时候调用，%this 是指向 PlayerBody 数据块的句柄，%obj 是这个刚刚创建的化身对象的句柄。

然后进行相关的初始化设置，如设置玩家对象的车辆属性状态，这里表示玩家没有绑定在一个车辆上。接着，设置了玩家的能量恢复率，这个在您进入游戏的时候会发现玩家的能量（蓝色状态条）在不断增加直到全满为止。同时设置了玩家生命的恢复率，这里是 0，也就是说当玩家的生命受到损伤时，不会自动恢复生命。这与我们通常玩的游戏不太符合，没关系，如果您愿意，完全可以修改它。最后是定义了一个 volumeLight 的光对象，它来自 sgMountLight 数据块，这里使用//将其屏蔽了，也就是说在游戏中它没有任何作用。但实际上，这在游戏中是个相当有用的功能。它会使玩家身上发出耀眼或者美丽的光芒。目前流行的游戏，大都会通过人物身上（准确的说应该是衣服，武器装备等）发出的光芒，来证明这是一个与众不同的“ 高手 ”，这真实个相当棒的创意啊！

```
function Armor::onRemove(%this, %obj)
{
    if (%obj.client.player == %obj)
        %obj.client.player = 0;
    //if(isObject(%obj.light))
    //    %obj.light.delete();
}
```

方法 Armor::onRemove()在移除玩家化身时候调用，参数同上。

```
function Armor::onMount(%this,%obj,%vehicle,%node)
{
    if (%node == 0)  {
        %obj.setTransform("0 0 0 0 0 1 0");

%obj.setActionThread(%vehicle.getDatablock().mountPose[%node],true,true);
        %obj.lastWeapon = %obj.getMountedImage($WeaponSlot);

        %obj.unmountImage($WeaponSlot);
        %obj.setControlObject(%vehicle);
        %obj.client.setObjectActiveImage(%vehicle, 2);
    }
```

}

方法 Armor::onMount()在玩家化身进入交通工具时候调用，%this 是指向 Armor 数据块的句柄，%obj 是这个刚刚创建的化身对象的句柄，%vehicle 是玩家化身进入的交通工具的句柄，%node 是装配的节点。这里是当%node=0（0 是车辆默认的司机的坐位）的时候进行“ 装配 ”操作，第一件事情是将玩家化身以标准的方向设置成一个空变换，剩下的游戏玩家化身的变换信息将由游戏引擎处理，使得玩家化身绑定在车辆上，车辆到哪里，玩家化身就跟到哪里。但是如果只是简单地绑定玩家化身到车辆上，您会发现玩家和车辆非常不和谐，因为玩家直挺挺地戳在那里！因此需要调用 setActionThread()方法（该方法超出了本书的范围），它会调用玩家化身对象预先定义的针对车辆装配的时候为 mount0 定义的动画序列，这个序列只有一帧，所以玩家的化身只是坐在那里，即在汽车里。也许我们在创建汽车模型的时候只创建了外部模型，而根本没有为玩家化身制作坐位，不过没关系，因为玩家的化身“ 坐 ”在了车里边，由于被车辆的模型遮挡就好像真的进入车辆内部，又由于它跟着车辆移动，所以给我们的感觉好像真的进入车里，而即使它在车里面的真实造型是“ 扎马步 ”很难受，就不是我们操心的事情啦。之后，将玩家上车前拿在手上的武器信息保存，然后不再显示玩家的武器，将键盘控制权交给车辆，激活车辆的画面，即您看到的不再是您端着的武器，而坐在车上应该看到的图像。

```
function Armor::onUnmount( %this, %obj, %vehicle, %node )
{
    if (%node == 0)
        %obj.mountImage(%obj.lastWeapon, $WeaponSlot);
}
```

方法 Armor::onUnmount()处理玩家化身一旦离开车辆之后触发的动作，这里是恢复我们最后使用的武器。参数同上。

```
function Armor::doDismount(%this, %obj, %forced)
{
    // This function is called by player.cc when the jump trigger
    // is true while mounted
    if (!%obj.isMounted())
```

```
    return;
// Position above dismount point
%pos       = getWords(%obj.getTransform(), 0, 2);
%oldPos = %pos;
%vec[0] = " 0    0    1";
%vec[1] = " 0    0    1";
%vec[2] = " 0    0 -1";
%vec[3] = " 1    0    0";
%vec[4] = "-1    0    0";
%impulseVec    = "0 0 0";
%vec[0] = MatrixMulVector( %obj.getTransform(), %vec[0]);
// Make sure the point is valid
%pos = "0 0 0";
%numAttempts = 5;
%success         = -1;
for (%i = 0; %i < %numAttempts; %i++) {
    %pos = VectorAdd(%oldPos, VectorScale(%vec[%i], 3));
    if (%obj.checkDismountPoint(%oldPos, %pos)) {
        %success = %i;
        %impulseVec = %vec[%i];
        break;
    }
}
if (%forced && %success == -1)
    %pos = %oldPos;
%obj.mountVehicle = false;
%obj.schedule(4000, "mountVehicles", true);
// Position above dismount point
%obj.setTransform(%pos);
%obj.applyImpulse(%pos,                    VectorScale(%impulseVec,
```

```
%obj.getDataBlock().mass));
        %obj.setPilot(false);
        %obj.vehicleTurret = "";
    }
```

方法 Armor::doDismount()是当玩家驾驶车辆时，按下跳跃（空格）键后出发的动作，这是在引擎源代码 player.cc 中设置好的。如果您有兴趣可以去查看一下引擎源代码，不过本书的内容不设计任何引擎源代码级的操作，脚本编程对我们目前来说已经足够了。上述代码的作用是确定在将游戏玩家化身从车辆中移出之后，所选的游戏玩家化身的落脚点是否安全和合理。我们从设置方向矢量开始，应用这个矢量至我们的游戏玩家以便提前计算出刚刚拆卸的游戏玩家化身要落脚的地方，然后使用 checkDismountPoint 方法确认一切正常。如果出现问题，算法将继续四处移动矢量直到找到合适的地方为止。

```
    function Armor::onCollision(%this,%obj,%col)
    {
        if (%obj.getState() $= "Dead")
            return;
        // Try and pickup all items
        if (%col.getClassName() $= "Item")
            %obj.pickup(%col);
        // Mount vehicles
        %this = %col.getDataBlock();
        if ((%this.className $= WheeledVehicleData) && %obj.mountVehicle &&
                %obj.getState() $= "Move" && %col.mountable) {
            // Only mount drivers for now.
            %node = 0;
            %col.mountObject(%obj,%node);
            %obj.mVehicle = %col;
        }
    }
```

方法 Armor::onCollision(%this,%obj,%col)在玩家化身同那些能够产生碰撞的对象发生碰撞时又引擎自动调用，该方法只检测 ShapeBase 类的对象之间发生的碰撞，比如 Item（物品）、Player（玩家化身）、vehicle（交通工具）和 weapon（武器）。参数%this，PlayerBody 数据块的句柄；%obj，玩家化身的句柄；%col，同玩家化身产生碰撞的对象句柄。发生碰撞时，首先检查玩家的状态是不是“死亡”，是则返回。不是，则判断碰到的对象是否是“Item”对象，是，则触发捡起动作。接着通方法%col.getDataBlock()可以获得%col 对象所属的数据块对象的句柄，如果该对象是交通工具（WheeledVehicleData）对象，并且该车辆可以使用、玩家状态正在移动以及玩家的化身可以驾驶车辆，则使用%col.mountObject(%obj, %node)方法完成上车动作。

```
function Armor::onImpact(%this, %obj, %collidedObject, %vec, %vecLen)
{
    %obj.damage(0, VectorAdd(%obj.getPosition(),%vec),
        %vecLen * %this.speedDamageScale, "Impact");
}
```

方法 Armor::onImpact(%this, %obj, %collidedObject, %vec, %vecLen)同上一个方法有相似的地方，但是不同，该方法会检测任何碰撞，包括地形和 Interior（内景、建筑物）之间的碰撞。参数%this，PlayerBody 数据块对象的句柄；%obj，玩家化身的句柄；%collidedObject，击中玩家化身的对象（或玩家化身击中的对象）的句柄；%vec，玩家化身和被击中对象的相对速度；%vecLen，被击中对象的绝对速度。这里我们使用这些参数调用%obj.damage()方法。

```
function Armor::damage
(%this, %obj, %sourceObject, %position, %damage, %damageType)
{
    if (%obj.getState() $= "Dead")
        return;
    %obj.applyDamage(%damage);
    %location = "Body";
    %client = %obj.client;
    %sourceClient = %sourceObject ? %sourceObject.client : 0;
```

```
    if (%obj.getState() $= "Dead")
        %client.onDeath(%sourceObject, %sourceClient, %damageType, %location);
}
```

方法 rmor::damage()确定伤害对玩家化身产生的影响。如果玩家处于“ 死亡 ”状态，则退出。否则调用%obj.applyDamage()方法来接受伤害，如生命值减少等。如果玩家化身接受本次伤害后死亡，则调用%client.onDeath()方法进行处理。

```
function Armor::onDamage(%this, %obj, %delta)
{
    if (%delta > 0 && %obj.getState() !$= "Dead") {
        %flash = %obj.getDamageFlash() + ((%delta / %this.maxDamage) * 2);
        if (%flash > 0.75)
            %flash = 0.75;
        %obj.setDamageFlash(%flash);
        if (%delta > 10)
            %obj.playPain();
    }
}
```

方法 Armor::onDamage()在玩家对象的伤害值发生变化时引擎将调用这个方法。当有伤害发生时，如果需要为玩家产生一些特殊的效果，如使屏幕颤动或者播放某些音效，我们就会用到这个方法。这里我们使屏幕颤动，并开始缓慢的减去以因伤害而耗损的能量。如果伤害非常严重，将使用%ojb.playPain()方法播放玩家化身痛苦的哀嚎。

```
function Armor::onDisabled(%this,%obj,%state)
{
    %obj.playDeathCry();
    %obj.playDeathAnimation();
    %obj.setDamageFlash(0.75);
    %obj.setImageTrigger(0,false);
    %obj.schedule($CorpseTimeoutValue - 1000, "startFade", 1000, 0, true);
```

```
    %obj.schedule($CorpseTimeoutValue, "delete");
}
```

方法 rmor::onDisabled()在玩家化身的伤害超过 maxDamage 属性值的时候调用。参数%this,玩家化身的数据块对象句柄；%obj，玩家化身的句柄；%state，玩家化身的状态。接下拉到操作是发出死亡的喊叫，播放死亡动画（前面定义过的 11 种死亡动画），颤动屏幕。冻结武器，最后处理玩家化身的尸体，保留一段时间后慢慢消失，然后删除该尸体对象。

其它的方法都不难理解，这里就留给读者自己思考一下吧。

Inventory.cs

Inventory.cs 文件里保存的代码是关于游戏中玩家对象携带物品的库存管理的。下面我们将学习它的内容。初次接触这些内容，肯定会或多或少令您异常困惑，请打起十二分的精神来！希望我的讲解能够尽可能地解决您的困惑。

```
function serverCmdUse(%client,%data)
{
    %client.getControlObject().use(%data);
}
```

函数 serverCmdUse()执行“ 使用 ”动作，参数%client，客户端句柄；%data，将使用的物品的句柄。serverCmd 为标记字符串的前缀，是服务器端执行客户机端请求的命令。这里只做一件事，“ 使用 ”（武器）。

```
function ShapeBase::use(%this,%data)
{
    // Use an object in the inventory.
    if (%this.getInventory(%data) > 0)
        return %data.onUse(%this);
    return false;
}
```

方法 ShapeBase::use()是基于 ShapeBase 类的方法，参数同上。它完成的操作是，如果请求使用的物品的数量（库存）>0，则“ 使用 ”它，否则失败。

```
function ShapeBase::throw(%this,%data,%amount)
```

```
{
    if (%this.getInventory(%data) > 0) {
        %obj = %data.onThrow(%this,%amount);
        if (%obj) {
            %this.throwObject(%obj);
            return true;
        }
    }
    return false;
}
```

方法 ShapeBase::throw()完成丢弃操作，参数%amount 是丢弃的数量。如果库存物品的数量>0，则进行丢弃操作。具体过程可以这样理解：首先判断物品的的库存数量，大于 0，则新建一个%obj 局部变量，调用 onThrow()方法返回一个丢弃物品的句柄，这个句柄保存在%obj 中（这个时候要丢弃的物品还没有出现在游戏场景中），如果%obj 为真，则调用 ShapeBase 数据块的 throwObject()方法将该物品丢弃到游戏世界中。

```
function ShapeBase::pickup(%this,%obj,%amount)
{
    %data = %obj.getDatablock();
    if (%amount $= "")
        %amount = %this.maxInventory(%data) - %this.getInventory(%data);
    if (%amount < 0)
        %amount = 0;
    if (%amount)
        return %data.onPickup(%obj,%this,%amount);
    return false;
}
```

方法 ShapeBase::pickup()进行捡起物品操作。参数%this，ShapeBase 数据块对象的句柄；%obj，被拾取的对象的句柄；%amount，被拾取对象包含的（弹药

等）数量。首先，将%obj 对象所属的数据块对象的句柄保存在%data 中，注意，这里保存的内容同%this 不同。如果实现没有为物品设置其包含的数量，则将其设置为（最大值－现有值）。如果%amount<0，则令%amount=0。如果不为 0，即（最大值－现有值）不为 0，或者预先设定的%amount 不为 0，就返回%data.onPickup(%obj,%this,%amount);的结果，即进行捡起物品，同时添加物品库存的操作。

```
function ShapeBase::maxInventory(%this,%data)
{
    return %this.getDatablock().maxInv[%data.getName()];
}
```

方法 ShapeBase::maxInventory 返回物品的最大库存量（不是库存中现有物品的数量），参数%this，ShapeBase 数据块对象的句柄；%data，物品的句柄。刚刚看到这个函数的时候，您可能会感到非常迷惑。请这样理解，我们有一个物品弹夹 a（a 是该弹夹的句柄），而 a 是属于 ShapeBase 数据块的对象，该数据块的句柄为 b，则%this = b，%data = a。a 的名字假设为 Ammo，则 a.getName()=Ammo。而 maxInv[Ammo]是在初始化 PlayerBody 数据块对象时候定义的静态域（忘记了？翻到前面复习一下，一定会对您的理解相当有帮助），它表示的是 a 中最大容量，比如等于 10，即弹夹 a 的最大容量是 10 发子弹，这样的解释您是否清楚了？

```
function ShapeBase::incInventory(%this,%data,%amount)
{
    %max = %this.maxInventory(%data);
    %total = %this.inv[%data.getName()];
    if (%total < %max) {
        if (%total + %amount > %max)
            %amount = %max - %total;
        %this.setInventory(%data,%total + %amount);
        return %amount;
    }
    return 0;
```

```
}
```

方法 ShapeBase::incInventory()进行物品库存的增加操作。参数同上。首先用%max 保存该物品的最大库存数量，将捡起物品所携带的数量保存在%total 中，接下来进行相关判断，确保增加的数量同库存中已有数量的和不要超过最大库存的数量。然后使用%this.setInventory()方法将增加的数量添加到库存中。最后，该方法返回的是实际增加的数值。

```
function ShapeBase::decInventory(%this,%data,%amount)
{
    %total = %this.inv[%data.getName()];
    if (%total > 0) {
        if (%total < %amount)
            %amount = %total;
        %this.setInventory(%data,%total - %amount);
        return %amount;
    }
    return 0;
}
```

方法 ShapeBase::decInventory()同方法 ShapeBase::IncInventory()相似，只是操作相反，返回的是实际减少的数值。

```
function ShapeBase::getInventory(%this,%data)
{
    return %this.inv[%data.getName()];
}
```

方法 ShapeBase::getInventory()返回物品库存里当前剩余的数量。

```
function ShapeBase::setInventory(%this,%data,%value)
{
    if (%value < 0)
        %value = 0;
    else {
        %max = %this.maxInventory(%data);
```

```
        if (%value > %max)
            %value = %max;
    }
    %name = %data.getName();
    if (%this.inv[%name] != %value)
    {
        %this.inv[%name] = %value;
        %data.onInventory(%this,%value);
        %this.getDataBlock().onInventory(%data,%value);
    }
    return %value;
}
```

方法 ShapeBase::setInventory()用来设置物品库存里的数量。参数%this,ShapeBase 数据块对象的句柄；%data，需要进行设置的物品对象的句柄；%value，设置的数量。首先判断一下%value 的特殊情况，小于 0 则令其等于 0，大于最大值则令其等于最大值。然后判断物品库存现有数量%this.inv[%name]的值是否等于要设置给它的值%value，如果不相等，则设置其值＝%valued 的值，而接下来两次调用的 onInventory()方法在本例子中为空函数，即什么都没有做。最后返回本次操作要设置的值。

```
function ShapeBase::clearInventory(%this)
{
    // To be filled in...
}
```

方法 function ShapeBase::clearInventory()，暂时没有内容，从字面意思理解是清空库存的意思，如果您在您的游戏中需要这样的操作，可以自己尝试这在里边编写您的代码，应该不会难倒您的。

剩下的一些方法基本上仅有方法名，而在目前的游戏中没有更多的内容，您可以在需要的时候自己添加相应的代码。

Crossbow.cs

Crossbow.cs 文件是一个专用的武器模块，用来描述 crossbow（弩）这种武器。其中包括数据块、方法（函数）、粒子效果定义（如果是独一无二的武器的话）以及其它相关的东西。如果您计划在您的游戏里出现多种武器，就需要为每一种武器创建一个这样的模块，好像挺复杂，其实都是模版式的，用起来并不复杂。下面来看看它的内容。

```
datablock AudioProfile(CrossbowFireSound)
{
filename = "~/data/sound/relbow_mono_01.ogg";
description = AudioClose3d;
preload = true;
};
```

标准的声音数据块对象定义，CrossbowFireSound 是它的名字，表示弩开火时发出的声音。Filename 指出使用磁盘中的哪个声音文件，这里是~/data/sound/explosion_mono_01.ogg。description 描述声音的类型，这里是 AudioDefault3d，即默认的 3D 音效，它会随着距离远近而不同。Preload 表明该声音是否需要预先加载，true 表示是，这样会占用一些内存，但是调用的时候会非常快。其它的声音数据块对象的定义类似，就不多说了。

接下来是一系列 ParticleData 数据块对象和 ParticleEmitterData 数据块对象的定义，完全都是标准的数据块对象定义，由于初始化的属性相对较多，看起来好像挺复杂，其实是非常简单的工作，具体属性的含义请参考粒子数据块那部分，这里就不多费口舌了。

其它的一些如 DebrisData（碎片）、ExplosionData（爆炸）和 ProjectileData（发射物）数据块对象都是使用标准方法初始化的，没有任何技术深度。

下面我们来看看有关数据块的方法，这是我们感兴趣的。

```
functionCrossbowProjectile::onCollision(%this,%obj,%col,%fade,%pos,%normal)
{
    if (%col.getType() & $TypeMasks::ShapeBaseObjectType)
        %col.damage(%obj,%pos,%this.directDamage,"CrossbowBolt");
    radiusDamage(%obj, %pos, %this.damageRadius, %this.radiusDamage,
"Radius", %this.areaImpulse);
```

}

方法 CrossbowProjectile::onCollision()是最关键的方法之一。参数%this，CrossbowProjectile 数据块对象的句柄；%obj，弩箭对象的句柄；%col，被击中对象的句柄；%fade，暂时不清楚它表示的含义；%pos，被击中对象的位置信息；%normal，弩箭对象发射出去的方向向量。

该方法是在弩箭已经击中一个对象的时候马上调用，而不是弩箭在空中飞行时候调用。在击中了对象后，该方法先进行一个判断，因为弩箭可能直接击中的玩家的身体，但是更多的可能击中的是房屋、树木等对象以及地面！因此，通过%col 的类型（%col.getType()方法获得）同 ShapeBaseObjectType 的掩码进行“ 与 ”操作，如果为真，则表明击中的是 Shape 类型的对象，则调用 damage()方法来将伤害直接反映到被击中的对象上。即使这个对象不是玩家的化身，比如说击中的是一辆汽车或者是其它有“ 生命值 ”的对象，那么它们可能在这样的打击下“ 爆炸 ”，这在游戏中太平常不过了，不是么？最后调用 radiusDamage()函数把伤害反映到爆炸半径内的对象上。该函数在 radiusDamage.cs 文件中定义。

接下来是对武器和弹药数据块对象的定义。

```
datablock ItemData(CrossbowAmmo)
{
    category = "Ammo";
    className = "Ammo";
    shapeFile = "~/data/shapes/crossbow/ammo.dts";
    mass = 1;
    elasticity = 0.2;
    friction = 0.6;
    pickUpName = "crossbow bolts";
    maxInventory = 20;
};
```

定义了名为 CrossbowAmmo 的 ItemData 数据块对象，这是一个表示弩箭弹夹物品的对象，同时进行相关属性的初始化。其中，category 是该对象的分类名，前面已经详细解释过；className 是该对象的类的名字；shapeFile 指定了对象模

型在磁盘的位置；mass 是质量，elasticity 是弹力，friction 是摩擦力；然后定义了 2 个动态属性，pickUpName 是该对象被捡起的时候显示在对话框中的名字，maxInventory 是最大库存数量，即弹夹里弹药的数量。

```
datablock ItemData(Crossbow)
{
    category = "Weapon";
    className = "Weapon";
    shapeFile = "~/data/shapes/crossbow/weapon.dts";
    mass = 1;
    elasticity = 0.2;
    friction = 0.6;
    emap = true;
    pickUpName = "a crossbow";
    image = CrossbowImage;
};
```

定义了名为 Crossbow 的 ItemData 数据块对象。这是一个表示弩枪物品的对象，同时进行了相关的属性初始化，大致内容同上。需要说明的是它的动态属性 image，这个表示在第一人称视角下，游戏界面上显示端着的弩枪的图片。如果您足够细心，会发现弩枪没有定义弹药的数量，也就是说，如果您在游戏中捡起了一把弩枪是不会增加您的弹药数量的，如果您是一个 CS 的老万家，肯定会觉得这与我们通常的观念不太一致，也许老外认为枪就是枪，弹夹就是弹夹，哈哈。

```
datablock ShapeBaseImageData(CrossbowImage)
{
    shapeFile = "~/data/shapes/crossbow/weapon.dts";
    emap = true;
    mountPoint = 0;
    eyeOffset = "0.1 0.4 -0.6";
    correctMuzzleVector = false;
    className = "WeaponImage";
    item = Crossbow;
```

```
ammo = CrossbowAmmo;
projectile = CrossbowProjectile;
projectileType = Projectile;
stateName[0]                    = "Preactivate";
stateTransitionOnLoaded[0]      = "Activate";
stateTransitionOnNoAmmo[0]      = "NoAmmo";
stateName[1]                    = "Activate";
stateTransitionOnTimeout[1]     = "Ready";
stateTimeoutValue[1]            = 0.6;
stateSequence[1]                = "Activate";
stateName[2]                    = "Ready";
stateTransitionOnNoAmmo[2]      = "NoAmmo";
stateTransitionOnTriggerDown[2] = "Fire";
stateName[3]                    = "Fire";
stateTransitionOnTimeout[3]     = "Reload";
stateTimeoutValue[3]            = 0.2;
stateFire[3]                    = true;
stateRecoil[3]                  = LightRecoil;
stateAllowImageChange[3]        = false;
stateSequence[3]                = "Fire";
stateScript[3]                  = "onFire";
stateSound[3]                   = CrossbowFireSound;
stateName[4]                    = "Reload";
stateTransitionOnNoAmmo[4]      = "NoAmmo";
stateTransitionOnTimeout[4]     = "Ready";
stateTimeoutValue[4]            = 0.8;
stateAllowImageChange[4]        = false;
stateSequence[4]                = "Reload";
stateEjectShell[4]              = true;
stateSound[4]                   = CrossbowReloadSound;
```

```
    stateName[5]                    = "NoAmmo";
    stateTransitionOnAmmo[5]        = "Reload";
    stateSequence[5]                = "NoAmmo";
    stateTransitionOnTriggerDown[5] = "DryFire";
    stateName[6]                    = "DryFire";
    stateTimeoutValue[6]            = 1.0;
    stateTransitionOnTimeout[6]     = "NoAmmo";
    stateSound[6]                   = CrossbowFireEmptySound;
};
```

定义了一个名为 CrossbowImage 的 ShapeBaseImageData 数据块对象。同时进行了相关的属性初始化操作。其中 shapeFile 是模型文件在磁盘中的位置，emap = true 表明渲染该对象。mountPoint 是该武器绑定在玩家化身的位置，通常是用 0，是化身的右手的位置出，eyeOffset 是眼睛的偏移量。接下来是一些相关的定义，不难理解。最后定义了 6 组弩枪的状态描述。用来描述“ 准备－开火－添装 ”状态变化以及“ 没有弹药 ”的状态变化。

```
function CrossbowImage::onFire(%this, %obj, %slot)
{
    %projectile = %this.projectile;
    %obj.decInventory(%this.ammo,1);
    %muzzleVector = %obj.getMuzzleVector(%slot);
    %objectVelocity = %obj.getVelocity();
    %muzzleVelocity = VectorAdd(
        VectorScale(%muzzleVector, %projectile.muzzleVelocity),
        VectorScale(%objectVelocity, %projectile.velInheritFactor));
    %p = new (%this.projectileType)() {
        dataBlock        = %projectile;
        initialVelocity  = %muzzleVelocity;
        initialPosition  = %obj.getMuzzlePoint(%slot);
        sourceObject     = %obj;
        sourceSlot       = %slot;
```

```
        client                  = %obj.client;
    };
    MissionCleanup.add(%p);
    return %p;
}
```

方法 CrossbowImage::onFire()的工作是创建一个箭矢对象并把它发射出去。首先，将箭矢从装备库存中删除，使用的是 decInventory()方法，然后获取箭矢即将射出的矢量，计算出箭矢将射出去的速度，这个速度由弩枪发射箭矢的速度和弩枪的移动速度（玩家的移动速度）加合而得到。然后创建一个新的 projectile 对象，初始化它的属性信息，然后将这个新建的对象添加到游戏任务列表中，它就立即按照设置好的属性信息飞了出去。最后返回该对象的句柄。

Weapon.cs

Weapon.cs 文件的内容包含了用于 Weapon 和 Ammo 类的命名空间帮助方法，这两个类定义了一组方法，这些方法是动态命名空间的组成部分。所有 ShapeBase 类的都被放置在其的 8 个插槽上的一个，该武器系统假设所有的主武器都被绑定在专门的插槽上，即 Slot0 上。下面来看下它的内容。

首先使用 datablock 的方法定义了几个 AudioProfile 声音数据块对象，用来描述武器的使用、捡起等声音。

```
function Weapon::onUse(%data,%obj)
{
    if (%obj.getMountedImage($WeaponSlot) != %data.image.getId()) {
        serverPlay3D(WeaponUseSound,%obj.getTransform());
        %obj.mountImage(%data.image, $WeaponSlot);
        if (%obj.client)
            messageClient(%obj.client, 'MsgWeaponUsed', '\c0Weapon
selected');
        }
}
```

方法 Weapon::onUse()用来处理“使用”武器时候的行为，这里所说的“使

用”不是指开火，而是类似于选择武器的操作，如从手枪切换到步枪的时候，步枪处于“ onUse ”状态，或者是手里没有武器，然后捡起一件武器时，这个武器成为当前使用武器，处于“ onUse ”状态。

```
function Weapon::onPickup(%this, %obj, %shape, %amount)
{
    if (Parent::onPickup(%this, %obj, %shape, %amount)) {
        serverPlay3D(WeaponPickupSound,%obj.getTransform());
        if (%shape.getClassName() $= "Player" &&
                %shape.getMountedImage($WeaponSlot) == 0)    {
            %shape.use(%this);
        }
    }
}
function Weapon::onInventory(%this,%obj,%amount)
{
    if (!%amount && (%slot = %obj.getMountSlot(%this.image)) != -1)
        %obj.unmountImage(%slot);
}
function WeaponImage::onMount(%this,%obj,%slot)
{
    if (%obj.getInventory(%this.ammo))
        %obj.setImageAmmo(%slot,true);
}
function Ammo::onPickup(%this, %obj, %shape, %amount)
{
    if (Parent::onPickup(%this, %obj, %shape, %amount)) {
```

```
            serverPlay3D(AmmoPickupSound,%obj.getTransform());
        }
    }
    function Ammo::onInventory(%this,%obj,%amount)
    {
        for (%i = 0; %i < 8; %i++) {
            if ((%image = %obj.getMountedImage(%i)) > 0)
                if (isObject(%image.ammo) && %image.ammo.getId() ==
%this.getId())
                    %obj.setImageAmmo(%i,%amount != 0);
        }
    }
```

item.cs

item.cs 文件用来描述 item 对象，及其相应的方法。

$Item::RespawnTime 表示游戏中的可以“ 自产生 ”的物品，从它被捡起（这是它消失了）到它自动再次出现之间经过的时间，单位为毫秒，这里的值相当于 20 秒，任何“ static ”物品都是“ 自产生 ”物品。这种现象经常出现在一些第一人称射击游戏之中。

$Item::PopTime 表示游戏中的被丢弃的物品在它被删掉或者被再次捡起前，可以在游戏世界保存的时间。

```
    function Item::respawn(%this)
    {
        %this.startFade(0, 0, true);
        %this.setHidden(true);
        %this.schedule($Item::RespawnTime, "setHidden", false);
        %this.schedule($Item::RespawnTime + 100, "startFade", 1000, 0, false);
```

```
}
```

方法 Item::respawn()用来为“ static ”物品添加新的副本，当它们被捡起之后。第一条语句使得对象迅速又连续的消失掉，然后隐藏对象，最后，安排在以后的没个时间把对象重新显示出来，先使对象不再隐藏，然后在不长的时间内迅速而又连续地显示出来。

```
function Item::schedulePop(%this)
{
    %this.schedule($Item::PopTime - 1000, "startFade", 1000, 0, true);
    %this.schedule($Item::PopTime, "delete");
}
```

方法 Item::schedulePop()用来将一些物品在经过一段时间之后删除，这些物品都是被丢弃的物品，删掉它们能够保证我们的游戏世界中不会充满这些“ 不需要 ”的东西，减少资源占用。先进行隐藏操作，再进行删除操作。

```
function ItemData::onThrow(%this,%user,%amount)
{
    if (%amount $= "")
        %amount = 1;
    if (%this.maxInventory !$= "")
        if (%amount > %this.maxInventory)
            %amount = %this.maxInventory;
    if (!%amount)
        return 0;
    %user.decInventory(%this,%amount);
    %obj = new Item() {
        datablock = %this;
        rotation = "0 0 1 " @ (getRandom() * 360);
        count = %amount;
    };
    MissionGroup.add(%obj);
    %obj.schedulePop();
```

```
    return %obj;
}
```

方法 ItemData::onThrow()丢弃物品并在一段时间后清除它们。参数%this，ItemData 数据块对象的句柄；%user，使用者的句柄；%amount，数量值。结果几个条件语句的判断，最终减少库存中相应的值。然后为丢弃的物品创建一个新的对象实例，并把它添加到游戏世界中去。然后启动前面看到的 schedulePop()方法，最后返回这个被丢弃的物品的句柄。

```
function ItemData::onPickup(%this,%obj,%user,%amount)
{
    %count = %obj.count;
    if (%count $= "")
        if (%this.maxInventory !$= "") {
            if (!(%count = %this.maxInventory))
                return;
        }
        else
            %count = 1;
    %user.incInventory(%this,%count);
    if (%user.client)
        messageClient(%user.client, 'MsgItemPickup', '\c0You picked up %1', %this.pickupName);
    if (%obj.isStatic())
        %obj.respawn();
    else
        %obj.delete();
    return true;
}
```

方法 ItemData::onPickup()是所有物品对象都会使用到的方法之一。参数%this，ItmeData 数据块对象的句柄；%obj，被捡起对象的句柄；%user，捡起者

对象的句柄；%amount，数量值。它把物品添加到库存中，然后向客户机发送一条消息表明对象已经被捡起来了。如果对象是“ static ”的，将安排它在以后的某个时间重现创建其副本对象。否则，对象实例将被删除，我们不会再次看到它。

```
function ItemData::create(%data)
{
    %obj = new Item() {
        dataBlock = %data;
        static = true;
        rotate = true;
    };
    return %obj;
}
```

方法 ItemData::create()是万能的物品创建方法。还记得 Mission Editor Creator 么？对了，在那里通过点击树型结构的对象来创建物品的时候，其实就是调用的这个方法。从 Mission Editor Creator 中创建的对象全部是“ static ”的“ 自产生 ”可以旋转的对象。这是因为 static 和 rotate 的属性都设置为 true 的缘故。

staticShape.cs

```
function StaticShapeData::create(%data)
{
    %obj = new StaticShape() {
        dataBlock = %data;
    };
    return %obj;
}
```

该文件的内容相当简单，只定义了 StaticShape 对象的创建方法。这个是 Mission Editor Creator 中树型结构下 static object 中对象的创建方式。在前面 tutorial.base 中 editor.cs 中也有该方法，它们是完全一样的。

shapeBase.cs

shapeBase.cs 文件包含了 ShapeBase 对象的方法，所有继承它的子类都可以

使用该方法。

radiusDamage.cs

radiusDamage.cs 文件只有一个函数，用来处理爆炸对周围对象产生的作用。内容如下：

```
function radiusDamage(%sourceObject, %position, %radius, %damage, %damageType, %impulse)
{
InitContainerRadiusSearch(%position, %radius, TypeMasks::ShapeBaseObjectType);
   %halfRadius = %radius / 2;
   while ((%targetObject = containerSearchNext()) != 0) {
      %coverage = calcExplosionCoverage(%position, %targetObject,
         $TypeMasks::InteriorObjectType | $TypeMasks::TerrainObjectType |
         $TypeMasks::ForceFieldObjectType | $TypeMasks::VehicleObjectType);
      if (%coverage == 0)
         continue;
      %dist = containerSearchCurrRadiusDist();
      %distScale = (%dist < %halfRadius)? 1.0:
         1.0 - ((%dist - %halfRadius) / %halfRadius);
      %targetObject.damage(%sourceObject, %position,
         %damage * %coverage * %distScale, %damageType);
      if (%impulse)
      {
         %impulseVec = VectorSub(%targetObject.getWorldBoxCenter(), %position);
         %impulseVec = VectorNormalize(%impulseVec);
         %impulseVec = VectorScale(%impulseVec, %impulse * %distScale);
         %targetObject.applyImpulse(%position, %impulseVec);
```

```
            }
        }
    }
```

其中，参数%sourceObject，产生爆炸的对象，如努箭；%position，爆炸点的位置坐标信息；%radius，爆炸影响的半径；%damage，伤害的程度；%damageType，上海的类型；%impulse，爆炸产生的冲击力。首先调用了InitContainerRadiusSearch()函数，它的工作是为我们准备出一个以给定的中心坐标(%position)和半径（%radius）所构成的的空间范围，它为后面的containerSearchNext()函数做好准备，用来查找所有符合$TypeMasks::ShapeBaseObjectType类型的对象。然后设置%halfRadius值等%radiu的一半，开始循环搜索该指定爆炸影响的范围内满足条件的对象。只要containerSearchNext()的值不等于 0，就证明还有对象，将该对象的句柄保存在%targetObject 内，继续循环。每找到一个对象，就计算爆炸对该对象产生的伤害，但是在把计算结果反映到对象上时还要考虑到爆炸是否被某些类型的对象挡住了，如建筑物、车辆等。如果有这样的对象挡住了爆炸，那么爆炸的效果就就是0。这里的%coverage 是计算受到爆炸冲击的对象“暴露”在爆炸下的“面积”的百分比，如果为 0，就是完全被遮挡，不会受到伤害；如果为 1，就表面完全“暴露”，会受到较大的伤害。然后用 containerSearchCurrRadiusDist()函数得到被影响对象的近似半径，并从对象中心到爆炸中心的距离中减去这个值，从而获得对象离爆炸中心最近的一个表面间的距离，如果该距离小于爆炸半径的一半，则受到完全的爆炸伤害，否则，按照距离的一定比例得到相应的伤害。最后，如果存在“冲击波”，将会把对象推离爆炸中心！记住，调用了这个函数会产生很多的动作。

health.cs

health.cs 文件包含的是游戏使用的类似于急救包的物品。这里边一共定义了两种急救包，HealthKit 和 HealthPatch，它们的作用都是恢复玩家化身的生命值，区别是：一种是可以携带的，通过键盘命令才能“使用”的；另一种是我们玩家的化身一碰到它就自动“使用”的。

datablock AudioProfile(HealthUseSound)

```
{
    filename = "~/data/sound/health_mono_01.ogg";
    description = AudioClose3d;
    preload = true;
};
```

标准的 AudioProfile 数据块对象创建方式，描述的是玩家化身“使用”急救包恢复生命的时候发出的声音。

```
datablock ItemData(HealthKit)
{
    category = "Health";
    shapeFile = "~/data/shapes/items/healthKit.dts";
    mass = 1;
    friction = 1;
    elasticity = 0.3;
    emap = true;
    pickupName = "a health kit";
    repairAmount = 50;
};
```

标准的 ItemData 数据块对象创建方法，相信已经不会有任何疑问了。repairAmount 表示“使用”该急救包可以恢复的生命数值。

```
function HealthKit::onUse(%this,%user)
{
    if (%user.getDamageLevel() != 0) {
        %user.decInventory(%this,1);
        %user.applyRepair(%this.repairAmount);
        if (%user.client)
            messageClient(%user.client, 'MsgHealthKitUsed', '\c2Health Kit Applied');
        }
}
```

方法 HealthKit::onUse()在“使用”急救包时候调用，参数%this，HealthKit 数据块对象的句柄；%user，使用者对象的句柄。方法先判断玩家化身的伤害程度是否为 0，如果不是，则将玩家化身上急救包库存数量减 1，然后调用 applyRepair()方法实现恢复生命，该方法在 ShapeBase 类中定义，这里是继承该方法。然后判断一下是否是使用该急救包的客户机，如果是，则发送一条信息，显示\c2 指定的颜色的“Health Kit Applied”字样。

注意：如果玩家的生命值没有损失，则即使按下“使用”键，也不会“使用”急救包

```
datablock ItemData(HealthPatch)
{
    category = "Health";
    shapeFile = "~/data/shapes/items/healthPatch.dts";
    mass = 1;
    friction = 1;
    elasticity = 0.3;
    emap = true;
    repairAmount = 20;
    maxInventory = 0; // No pickup or throw
};
```

标准的 ItemData 数据块对象创建方式，非携带式的急救包。其中属性 maxInventory＝0 表明无法拾起或者丢弃。

```
function HealthPatch::onCollision(%this,%obj,%col)
{
    if (%col.getDamageLevel() != 0 && %col.getState() !$= "Dead" ) {
        %col.applyRepair(%this.repairAmount);
        %obj.respawn();
        if (%col.client)
            messageClient(%col.client, 'MsgHealthPatchUsed', '\c2Health Patch Applied');
            serverPlay3D(HealthUseSound,%obj.getTransform());
```

```
    }
}
```

由于这种急救包无法携带，是“即捡即用”型的，所以在它的方法上命名就变成了 HealthPatch::onCollision()。参数%this，数据块对象的句柄；%obj，急救包对象的句柄；%col，玩家化身的句柄。当玩家化身具有一定的生命伤害并且还没有死亡时，使用 applyRepair()方法恢复玩家生命，然后调用%obj.respawn()方法使该物品过一段时间在显示出来。最后，向“使用”它的客户机发送一条“Health Patch Applied”字样的消息，并且在这个位置播放一个使用了急救包的声音。

Chimneyfire.cs 文件中定义了很多的有关烟火的粒子信息，用来表现游戏中的烟火，内容非常简单，留给读者思考吧。

aiPlayer.cs

aiPlayer.cs 文件中定义了关于游戏中出现的 NPC 兽人（按照固定路线奔跑的那个）的相关内容。

```
datablock PlayerData(DemoPlayer : PlayerBody)
{
    shootingDelay = 2000;
};
```

定义了兽人 NPC 的数据块对象，名字叫 DemoPlayer，继承于 PlayerBody 数据块对象，添加了一个属性 shootingDelay。

```
function DemoPlayer::onReachDestination(%this,%obj)
{
    if (%obj.path !$= "") {
        if (%obj.currentNode == %obj.targetNode)
            %this.onEndOfPath(%obj,%obj.path);
        else
            %obj.moveToNextNode();
    }
}
```

方法 DemoPlayer::onReachDestination()用来是 NPC 兽人跑向为它指定的标

记点。参数%this，DemoPlayer 数据块对象的句柄；%obj，兽人 NPC 的句柄。如果该兽人 NPC 当前的节点等于目标节点，则停止奔跑，调用 onEndOfPath()方法，否则就跑向下一个标记节点的位置。

```
function DemoPlayer::onEndOfPath(%this,%obj,%path)
{
    %obj.nextTask();
}
```

方法 DemoPlayer::onEndOfPath()使兽人 NPC 执行下一个任务动作。

```
function DemoPlayer::onEndSequence(%this,%obj,%slot)
{
    echo("Sequence Done!");
    %obj.stopThread(%slot);
    %obj.nextTask();
}
```

方法 onEndSequence()调用了 stopTread()方法后开始执行下一个任务动作。

接下来创建了一系列 AIPlayer 类的方法，用来配合 NPC 兽人的动作，代码都不难理解，相信读者自己既可以理解它们。

6.2.3 client 文件夹

相比 server 文件夹的内容，client 文件夹的内容简单许对，大部分同 tutorial.base 文件夹中的内容几乎一致，这里就不再一一介绍了，剩下来的时间我将有选择地介绍相关内容。

注意：需要记住的是，毕竟游戏演示的 starter.fps 比用来入门的 tutorial.base 的内容多了很多，游戏世界中的对象也复杂了很多，尽管这些内容繁多，但是并不困难，希望读者能够花些时间将他们全部学习完。

Init.cs

```
function initClient()
{
    echo("\n--------- Initializing MOD: FPS Starter Kit: Client ---------");
```

```
$Server::Dedicated = false;
$Client::GameTypeQuery = "FPS Starter Kit";
$Client::MissionTypeQuery = "Any";
exec("./ui/customProfiles.cs"); // override the base profiles if necessary
initBaseClient();
initCanvas("Torque Game Engine", true);
if (!isObject(Canvas))
    return;
exec("./scripts/audioProfiles.cs");
exec("./ui/defaultGameProfiles.cs");
exec("./ui/PlayGui.gui");
exec("./ui/ChatHud.gui");
exec("./ui/playerList.gui");
exec("./ui/mainMenuGui.gui");
exec("./ui/aboutDlg.gui");
exec("./ui/startMissionGui.gui");
exec("./ui/joinServerGui.gui");
exec("./ui/loadingGui.gui");
exec("./ui/endGameGui.gui");
exec("./ui/optionsDlg.gui");
exec("./ui/remapDlg.gui");
exec("./ui/StartupGui.gui");
exec("./scripts/client.cs");
exec("./scripts/game.cs");
exec("./scripts/missionDownload.cs");
exec("./scripts/serverConnection.cs");
exec("./scripts/playerList.cs");
exec("./scripts/loadingGui.cs");
exec("./scripts/optionsDlg.cs");
exec("./scripts/chatHud.cs");
```

```
        exec("./scripts/messageHud.cs");
        exec("./scripts/playGui.cs");
        exec("./scripts/centerPrint.cs");
        exec("./scripts/default.bind.cs");
        exec("./config.cs");
        setNetPort(0);
        setShadowDetailLevel( $pref::shadows );
        setDefaultFov( $pref::Player::defaultFov );
        setZoomSpeed( $pref::Player::zoomSpeed );
        if ($JoinGameAddress !$= "") {
            loadMainMenu();
            connect($JoinGameAddress, "", $Pref::Player::Name);
        }
        else {
            Canvas.setCursor("DefaultCursor");
            loadStartup();
        }
    }
    function loadMainMenu()
    {
        Canvas.setContent( MainMenuGui );
        if($Audio::initFailed) {
            MessageBoxOK("Audio Initialization Failed",
                "The OpenAL audio system failed to initialize.    " @
                "You can get the most recent OpenAL drivers
<a:www.garagegames.com/docs/torque/gstarted/openal.html>here</a>.");
        }
        Canvas.setCursor("DefaultCursor");
    }
```

第一个函数就是 initClient()，用来启动客户端。先进行全局变量设置，将$Server::Dedicated 设为 false，表明我们启动的是个客户端而不是专用服务器端。

$Client::GameTypeQuery="FPS Starter Kit"将搜索的游戏类型通知服务器。

$Client::MissionTypeQuery = "Any";将搜索的任务类型通知服务器，Any 表示所有任务类型。

使用 initCanvas()启动客户端的窗口。

然后进行一系列文件加载。

SetNetPort(0);这条语句非常关键，尽管我们永远不会用到 0 端口，但是调用此语句很必要，它能保证 Torque 中的 TCP/IP 代码正确运行。以后，在其它模块中我们将根据所做的操作设置合适的端口。

最后，如果$JoinGameAddress 预先设置了需要连接的服务器的 IP 地址，则直接进行连接操作；否则调用 Startup()函数进入闪屏。

```
function loadMainMenu()
{
    Canvas.setContent( MainMenuGui );
    if($Audio::initFailed) {
        MessageBoxOK("Audio Initialization Failed",
            "The OpenAL audio system failed to initialize.    " @
            "You can get the most recent OpenAL drivers <a:www.garagegames.com/docs/torque/gstarted/openal.html>here</a>.");
        }
    Canvas.setCursor("DefaultCursor");
}
```

函数 loadMainMenu()使得的游戏客户端进入主菜单画面。如果声音初始化问题，则显示一个警告对话框，最后设置鼠标状态为“显示”状态。

defaules.cs 和 prefs.cs 文件的内容几乎同 tutorial.base 文件夹下的相应文件一摸一样，这里就不再介绍了。

此外，scripts 文件夹中的内容同 tutorial.base 文件夹下相应的文件内容绝大部分一致，而针对于 starter.fps 这个游戏增加了一些同游戏内容相关的脚本文件，

这些文件的内容都不是很难，就交给读者自己完成理解吧。

好了，我们使用了相当多的经历学习了 tutorial.base 和 starter.fps 文件夹下的代码，相信您已经对如何组织 Torque 的游戏文件夹结构，以及其中脚本文件的内容了如指掌了吧。您可能会说，还剩下 starter.racing 文件夹的内容没有学习呢。说实话，我希望您自己独立去研究 starter.racing 文件夹中的内容。如果您仔细研究一下，您会发现，该文件夹下的内容几乎和 starter.fps 没什么区别，除了有个关于车辆的 car.cs 脚本文件之外，甚至还少了好多内容。也就是说，其实您已经基本掌握了它的全部内容！在后面的章节中，我们将使用引擎 SDK 为我们提供的现成的各种资源，将它们组合在一起，做一个内容更加丰富的游戏。

注意：直接使用 starter.pfs 和 starter.racing 游戏 demo 对于我们初学者来搭建自己的游戏框架是个非常不错的起点！事实上，garagegames 公司也是鼓励我们这样做的。在它的基础上，我们可以迅速地制作出一款非常出色的 FPS 游戏，这对于提高您的能力，增强您的信心，鼓舞您的士气都是相当有帮助的。好的开始是成功的一半，加油干吧。

6.2 本章小结

通过本章的学习，我们了解了 starter.fps 游戏文件夹的绝大部分内容，基本掌握了该射击游戏的工作原理，到此，本书中所有关于代码的部分就结束了。下一章，我们将学习如何是用 GarageGames 公司官方指定的 DIF 文件专用制作工具 Constructor，通过熟练掌握和使用该工具，您可以为您的游戏制作出效果非凡的 Interior 建筑物，为您的游戏锦上添花！

第七章　制作.DIF 文件

本章，我们将学习如何为 Torque 创建 Interior 对象。

前面我们已经介绍过了，Torque 支持一种被称为 DIF 型的对象，我们不能直接创建 DIF 文件，但是我们可以先创建.map 文件，然后使用 garagegames 公司为我们提供的插件 Map2Dif Exporter 自动转换成我们需要的 DIF 文件。而制作.map 文件的工具非常多了，常见的有 Hammer、3D World Studio、QuArk 等等。这里我推荐您使用 garagegames 公司自己开发的专用工具——Constructor。Constructor 的界面同目前主流三维建模软件的界面风格一致，制作简单，输出的.DIF 基本没有问题，制作效果上丝毫不逊色于那些知名的软件。目前，constructor 已经出到了 1.0.3 版本，如果您购买了 Torque1.5 独立版的 SDK 授权，您将免费获得 Constructor 的使用授权，还等什么？赶快下载 Constructor1.0.3，然后开始我们新的旅程。

Constructor 系统需求

Macintosh（苹果）

Mac OS X Version 10.2 或者更高

G4，G5 或者 Intel Mac

512MB 内存（推荐 1G）

OpenGL 兼容的 3D 图形加速卡

Windows

Windows 2000, XP 或者 Vista

Pentium III 500

512MB 内存（推荐 1G）

OpenGL 或者 DirectX 兼容的 3D 图形加速卡

注意：在使用 Map2Dif Exporter 插件时，当输出路径包括非英文字符时可能失败；当使用 map2dif 导出时卡巴斯基网络安全 6.0 会提示安全警告；静态网格只支态光照。

7.1 Constructor 的界面

首先让我们通过熟悉 Constructor 默认的工作界面开始学习，Constructor 的工作界面如图 7.1 所示：

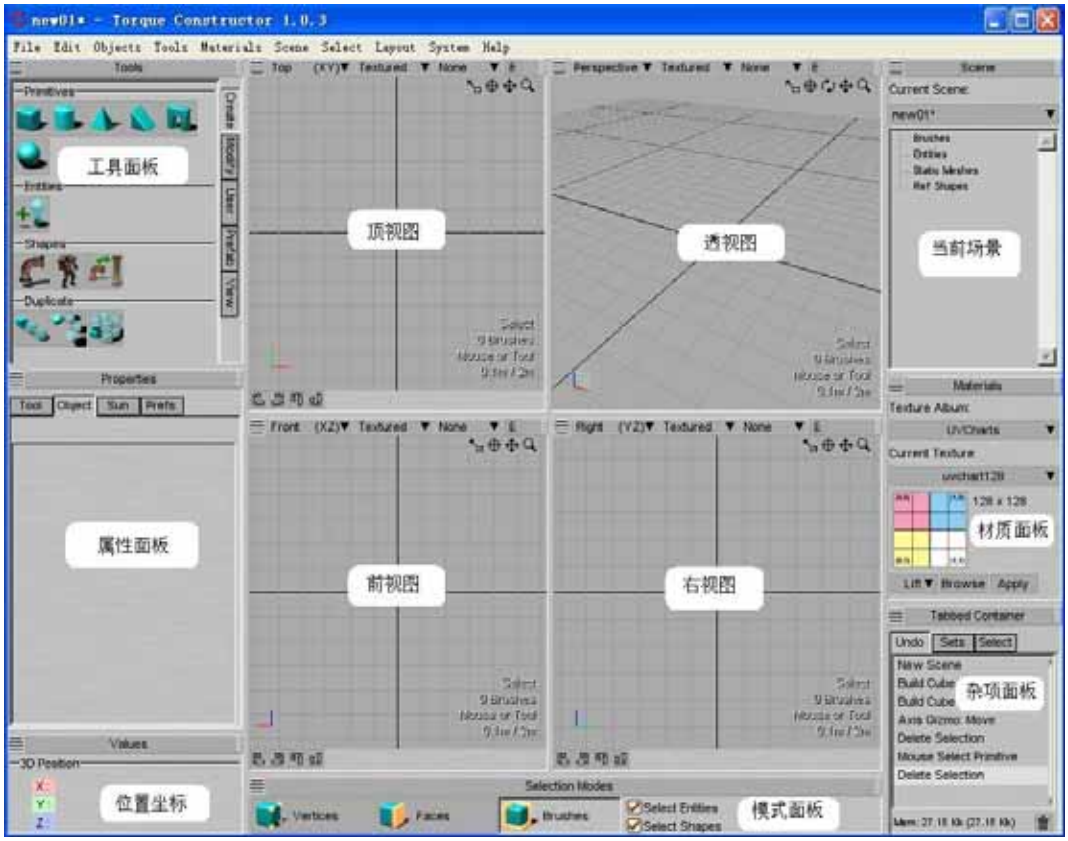


图 7.1　Constructor 的工作界面

Constructor 的工作界面同大多数三维建模软件的工作界面风格相似，这种设计非常体贴，使我们更加容易熟悉和使用它。从整体来看，Constructor 的工作界面主要分为左、中、右三个部分。左边部分包括工具面板、属性面板和位置坐标面板；中间部分包括四种视图面板和模式面板；右边部分则是当前场景面板、材质面板和杂项面板。其中：

工具面板：包括所有用来创建和修改您的 brushes（画刷）和静态网格几何体的功能。也包括高级视图控制选项。

属性面板：显示与当前工具或者对象相关联的属性。所有关于实体对象的数字参数都在这里现实。同时包括高级材质对齐工具（类似于 max 中的 UV 坐标），这必须在 face（面）模式下。

位置坐标面板：显示您的鼠标所在 3D 空间的位置坐标。

视图面板：默认情况下为 4 视图，顶视、透视、前视和右视。

模式面板：允许您选择选定几何体的方式，点、面和画刷方式。

当前场景面板：列出当前场景内所有的 brushes 和其他对象。

材质面板：允许您从材质列表或者当前对象上选择材质，或者是打开指定位置的材质。您创建的对象将默认使用您当前指定的材质。

杂项面板：包含显示历史操作列表，创建和删除组以及类似资源管理器的功能。

7.2 创建模型

接下来，我们来学习如何创建模型。在 Constructor 中，构成对象的基本单元是 brush（画刷）。Brush（画刷）是封闭的凸型的 3D shape（形体）。这些 shapes 是 Constructor 的基本建模工具。它们提供了一种纯多边形模型，该模型都经过了代码最优化的处理，并且使用的是许多游戏引擎都支持的标准 interior 格式。

7.2.1 工具面板

工具面板包括 Create、Modify、User、Prefab 和 View 五个选项卡。

Create（创建）选项卡

在 Primitives 中，constructor 为我们提供了可以创建的基本 brush（画刷）形状，分别是 cube（立方体）、tube（管状体）、cone（锥体）、ramp（斜面）、arch（拱门）和 sphere（球体）。如表 7.1 所示：

表 7.1　Constructor 中的基本 brush 创建工具

图标	名称	功能
	Cube（立方体）	创建一个立方体模型 brush
	Tube（管状体）	创建一个管状体（圆柱）模型 brush
	Cone（锥体）	创建一个锥体模型 brush
	Ramp（斜面）	创建一个斜面模型 brush

	Arch（拱门）	创建一个拱门模型 brush
	Sphere（球体）	创建一个球体模型 brush

通过这些基本的 brush，采用直接创建和间接拼凑的方法，我们可以创建出绝大部分游行中需要的建筑对象。

在 Entities 中，提供了光源实体，可以通过它创建各种类型的光源，为我们的场景提供色彩斑斓的光照效果。

在 Shapes 中，包含了插入静态 shape、插入动态 shape 以及替换静态 shape 的操作，这样，我们就可以将我们制作好的 shape 模型导入到当前场景中，这对于处理比例大小、位置关系都是非常有帮助的。

Duplicate 中包括三种复制操作，分别是 Linear Clone（直线复制）、Radial Clone（旋转复制）和 Array Clone（克隆复制）。它可以方便地为我们复制出多个相同的 brushes。

Modify（修改）选项卡

在 Transforms 中，包括移动、旋转、平面放缩、整体放缩等基操作和一些辅助操作。如 Center1D 操作可以使您的对象同 x、y、z 轴中的任何一个对齐；Reset on ground 操作可以将您的对象底部同地平面对齐；Quantize 可以量化您的画刷。具体内容如表 7.2 所示：

表 7.2　Transforms 的具体内容

图 标	内　容	说　明
	移动　W	通过鼠标拖拽来移动一个或多个 brushes
	旋转　E	通过鼠标拖拽沿着一个轴旋转一个或多个 brushes
	拉伸面 R	通过选中一个面，使用鼠标拖拽拉伸这个面
	缩放　T	通过鼠标拖拽缩放被选中的 brush
	居中	使选中的 brushes 的中心对齐选定的坐标中心
	落地	使选中的 brushes 的底部与地面对齐

	量化	将选中的 brushes 量化处理

在 CSG 中，是关于 brush 的复杂操作。CSG Subtract 可以进行 brush 减操作。CSG Intersection 进行 brush 的交集操作。Hollow Brush 进行 brush 的挖空操作。Knife、Slice 和 Clip 都可以进行切割操作，Knife 提供一条线，按照线来切割物体；Slice 提供一个平面切割物体，Clip 提供一个三角面来切割物体。具体内容如表 7.3 所示：

表 7.3　CSG 中的具体操作

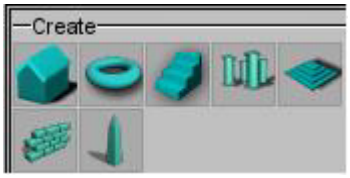
图 标	内　容	说　明
	减运算	用一个 brush 减去同它相交的 brush 的部分，得到剩下的部分
	交运算	得到两个相交的 bursh 的相交部分的 brush
	挖空运算	将一个 brush 中间挖空
	割运算	在一个 brush 上进行割操作，类似 3DS Max 的 cut 操作
	切片运算	在一个 brush 上进行割操作，类似 3DS Max 的 slice 操作
	切运算	在一个 brush 上进行割操作，类似 3DS Max 的 clip 操作

在 Brush Pivot Point 中定义 brush 的支点位置，可以选择为顶点或者是中心点。具体内容如表 7.4 所示：

表 7.4　Brush Pivot Point 的具体内容

图 标	内　容	说　明
	设置 brush 的轴点	将 brush 的创建方式设置为左上角的定点
	重置 brush 的轴点	将 brush 的创建方式设置为中心点

User（用户）选项卡



如果您没有安装 Constructor1.03 的插件，在 Create 中没有任何内容，否则，将会出现如右边小图所示的一些图标按钮，它们也是用来创建 brush 的。这里提供了 House（房屋）、Torus（圆环体）、Stairs（楼梯）、Column（柱体）、Ziggurat（金字塔）、BlockStatic（堆栈块）和 Obelisks（方尖塔）等模型的创建，这大大方便了我们创建一些类似形状的对象。具体内容如表 7.5 所示：

表 7.5　Create 中的具体内容

图 标	名　称	内　容
	House	创建一个房屋形状的 brush
	Torus	创建一个圆环形状的 brush
	Stairs	创建一个楼梯形状的 brush
	Columns	创建一个圆柱形状的 brush
	Ziggurat	创建一个金字塔形状的 brush
	BlockStatic	创建数多 brushes 构成一面墙
	Obelisks	创建一个尖方塔形状的 brush

Edit 中包含 Extrude Face（挤压面）的操作。Information 中，可以获得选定对象的信息。Build 为我们提供了两种创建对象的方法，一个是创建大量相同对象的方法，一个是创建一个简单的 box 的方法。General 中的“扳手”是干什么的？它就是弹出一个“hello world”的对话框，没有更多的内容了。

Prefab（预置）选项卡

这里边的内容非常非常有趣，并且我认为你可以直接使用它为我们做好的建筑在您的游戏当中！尝试着点击其中任何一个图标，然后在视图窗口中单击鼠标左键，您会看到在您的视图窗口中创建了新的对象，由于没有确定，这些对象都是线框结构的，点击左边属性面板下的“Make”按钮，啊哈，多么漂亮的建筑

出现了！可能的话，用在您的游戏场景里吧！

Veiw（察看）选项卡

Workplane 用来调整工作平面的位置，各个功能如表 7.6 所示：

表 7.6　Workplace 中各图标的说明

图 标	快捷键	功　能
	END	默认的工作平面
	HOME	将工作平面同选定的面所在平面对齐
	Shift+HOME	将工作平面同选定的面的材质坐标对齐
	无	将工作平面的中心移动到选定对象的中心
	Ctrl+HOME	将工作平面的中心与选定的对象的中心对齐

Scene workplanes 中可以添加新的工作平面和管理它们。

7.2.2 属性面板

属性面板中包含 4 个选项卡，Tool（工具）、（Object）、Sun（太阳）和 prefs（参数）。

Tool 选项卡

在没有创建任何对象之前，Tool 中的内容是空的。当您点击要创建物体的图标时，tool 中会显示出相关对象的相应的参数信息。在第一次创建时，brush 四周是紫色的边框。这表明该 brush 处于工作状态，它的参数没有最终完成。您可以通过鼠标拖拽进行修改，也可以直接修改它的各个参数。当你调整好你的 brush 后，必须“ make ”它，然后才能继续使用其它工具。使用“ Make&Continue” 按钮可以连续创建多个一模一样的 brushs，但是您需要通过移动将它们分开。

Object 选项卡

当您的对象创建完成后，可以在 Obejct 选项卡下设置它的类型。其中 Structural 是标准的结构体类型。Detail 是细节类型；Collision 是碰撞类型；Portal 是入口类型，Trigger 是触发器类型；Vehicle Collision 车辆碰撞类型；Other 其它

类型。Group，如果您为您的对象分组的话，可以在这里选择创建的对象属于哪个组，这对于对象的管理是个不错的方法。

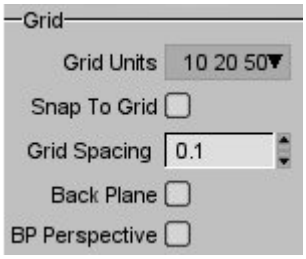
Sun 选项卡

Azimth 表示太阳的方位角，默认为 180 度。Elevation 为太阳的高度，默认为 35。Color 是太阳光的颜色，采用 RGB 值，默认为 255，255，255，白色。环境颜色，RGB 值，默认 64，64，64，深褐色。

Prefs 选项卡

Grid

该部分用来控制 grid（栅格）属性。

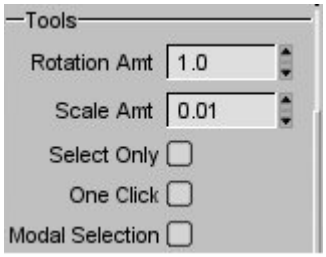


Grid units：Grid（栅格）单位。

Snap to Grid Spacing

选中该项目，将使所有操作对齐栅格，即创建、移动、改变等所有修改 brush 的操作，都会按照栅格的度量变化，它适用于所有 bursh、face 和 vertices 模式。

如果选择了 Snap To Grid，数字为变化的最小度量。



Rotation Amt：旋转时每变化一个象素的角度增量值。

Scale Amt：缩放时每变化一个象素的增量值。

Select only：只对选定的项目有效；

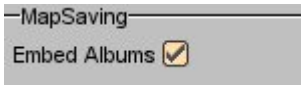
One Click：鼠标弹起执行“ make ”事件；

Modal Selection：无任何 tool 可用时 item selection 才可以执行。

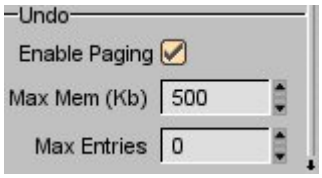
Enhanced Light：OpenGL 光照模式有效。

Inner Shell：Enable drawing a shell for a light's inner falloff

Outer Shell：



Embed Albums：保存文件时候，将使用的材质文件同生成的.map 文件放到一起；



Enable Paging：选中该选项，可以记录 undo 系统的操作。

Max Mem（Kb）：系统为 Undo 系统提供的最大内存量，默认 4096Kb，即 512K 字节。

Max Entries：Undo 堆栈的最大尺寸，0 表示无穷大。

位置坐标

用来显示鼠标在场景中的位置的坐标。

7.2.3 视图

视图是我们用来观看我们制作的模型效果的窗口，它们占据了整个应用程序窗口的大部分空间。这些视图中包含了不少命令，下面我们以下图 7.2（透视图）为例进行说明。

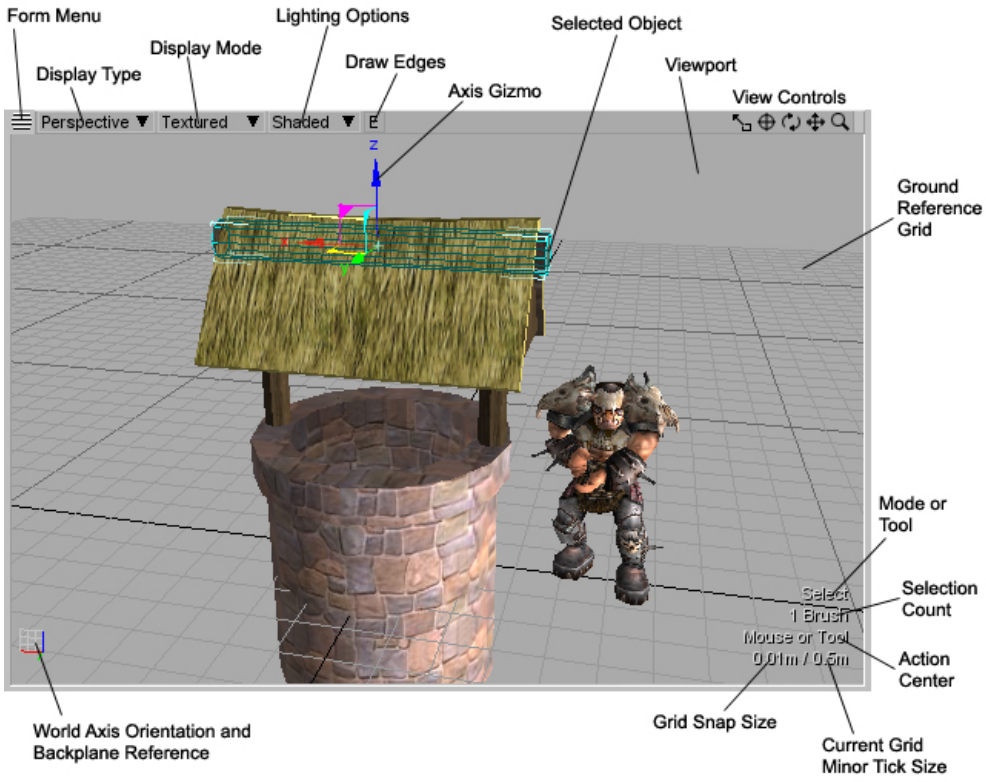


图 7.2　透视图的相关说明

请允许我按照顺时针的顺序来说明这些内容。

Form Memu：

全部都是一些选项操作，只要使用它的默认设置即可。

Display Type:

视图的显示模式：通过下拉菜单可以进行顶视、底视、左视、右视、前视、后视、透视以及材质 8 个视图之间的切换。

Display Mode：

显示模式选择，可选的模式如下表 7.7 所示：

表 7.7　Display 模式下的选项内容

模　式	内　容
Wireframe	所有对象采用线框显示
Solid	所有对象采用固体显示（无材质），并且接受关照影响
Textured	所有对象采用材质显示，接受光照影响
By brush	所有 brush 类型对象使用不同颜色显示，静态 mesh 采用固体显示，所有对象接受光照影响

By face	所有 brush 类似对象的每个面用随机颜色显示，静态 mesh 采用固体显示，所有对象接受光照影响
Export	该命令只渲染通过 Scene->Generate Export Preview 命令产生的 brush 对象，与 Shaded 合用，每个面用随机颜色显示，与 None 合用，显示为材质无 shading（阴影）效果

Light Options：

光照模式选择，可选的模式如下表 7.8 所示：

表 7.8　Light Options 的具体内容

选 项	内　容
None	无任何特殊的光照效果，只使用白光
Shade	使用 Shade 模式渲染场景
Mapped	使用 Mapped 模式渲染场景
Dynamic	使用 Dynamic 模式渲染场景

Draw Edges：

显示和关闭选中对象周围的包围框。

Axis Gizmo：

选中对象时显示的坐标轴向量。

Selected Object：

当前选中的对象。

Viewport：

视图。

View Controls：

视图控制选项，包括放大视图，指定对象、旋转、移动对象等操作。

Ground Reference Grid：

显示栅格的地平面。

Mode or Tool：

表明当前操作处于创建对象还是修改选择对象的状态。

Selection Count：

表明当前选中的对象数目。

Action Center：

鼠标操作；

Current Grid Minor Tick Size：

表示当前显示的栅格的单位，随着场景远近而变化。

Grid Snap Size：

拾取栅格的大小，将会使创建的对象与栅格自动对齐。

World Axis Orientation and Backplane Reference：

世界坐标轴和背景参考平面。

此外，每个视图窗口的左下方有 4 个小图标，用来对齐所选中的多个对象的，使用起来非常方便。

7.2.4 模式面板

选择模式面板为我们提供了选取对象的三种不同方式。

 选择点模式；  选择面模式；  选择对象模式。

Select Entities 允许使用鼠标选择 Entities 对象；Select Shapes 允许使用鼠标选择 Shapes 对象，该两个选项对于避免误选不需要的对象很有帮助。

7.2.5 当前场景面板

当前场景面板采用类似资源管理器的形式列出了所有当前场景创建的对象，包括 brushes（画刷）、Entities（实体）、Static Meshes（静态网格）和 Ref Shapes（关联形状）。通过它也个以快速的选择和查找对象。

材质面板

Constructor 中的材质通过材质面板访问和设置。

Texture Albums

上边第一项是材质 album 下拉菜单，所有的材质可以放在 albums 中。Album 以外的材质无法使用，constructor 预先为你准备了几个 albums。

Current Textures

第二个下拉菜单允许你从选定的 album 中选择并激活材质。激活的材质可以被指定给对象，这是唯一的方法。

图 7.3　constructor 的材质面板

接下来是一个选定的材质的预览，下方是该材质的尺寸，如 128×128。你可以选择使用象素或者米作为单位，通过左上角的菜单进行设置。

对于 torque 中使用的材质，必须是 jpg 或者 png 格式，并且必须是 2 的幂次大小。

Life 和 apply

当你选中一个 brush 或者 face 时，点击 lift 按钮可以展开一个可用材质的列表，选择其中一个作为当前材质。当你创建 brush 时，它的所有面都会指定为当前材质，如果你需要将看不见面指定为 null，可以先选择 face，然后选定面，选择 null 材质，点击 apply。

Browse

除了用上面提到的下拉方式，也可以使用材质浏览器窗口进行选择。点击 browse，出现如下图 7.4 所示：

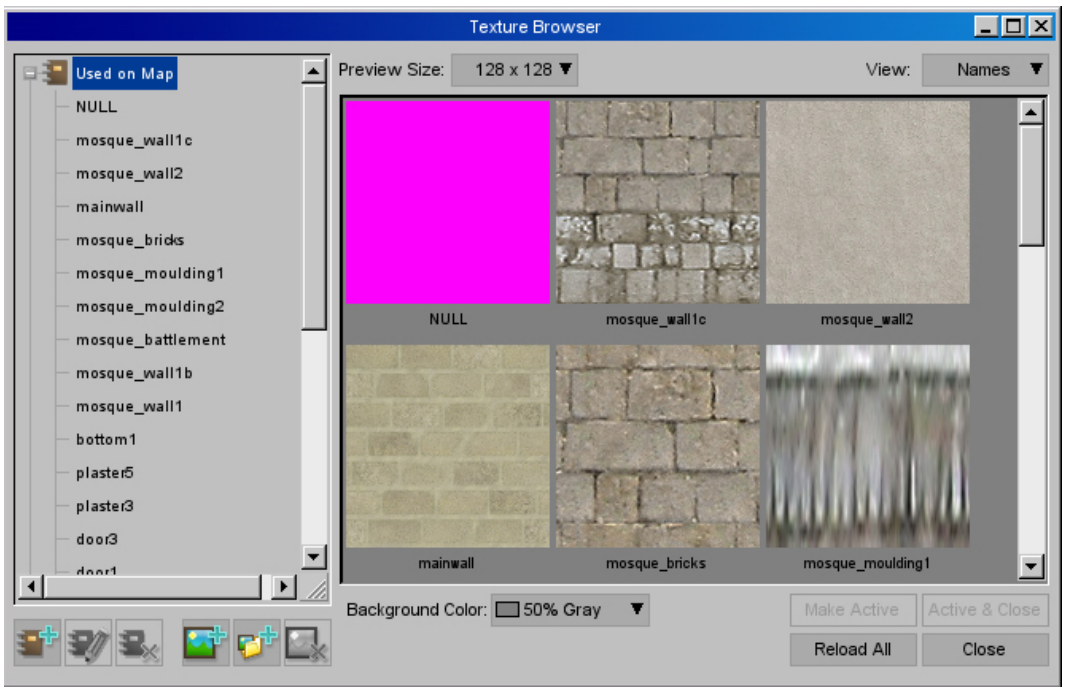


图 7.4　材质浏览器页面

The Texture Browser

左边是 albums 和 textures 列表。顶部的 Used on Map 是一个特殊的不能被删掉的 album，它显示当前场景中使用过的所有材质，但并不保证及时更新所有的变化，可以通过 Materials 菜单的 Used on Map 命令进行手动更新。其余的 albums 可以删除，并可以创建属于自己的 album。下面的按钮为：

Add a New Album：增加一个新的 album。

Rename the Selected Album：重新命名选定 album。

Delete the Selected Album：删除选定的 album。删除前会询问确认。

Add Texture to Selected Album：将硬盘中的 texture 添加到当前 album。

注意：如果是 Windows 操作系统，强烈推荐 textures 放在 C 盘根目录下，或者是 Constructor 的按照目录下，因为 Constructor 在跨分区操作存在问题。我们建议统一放到 Constructor\texture 文件夹下。

Add all textures in a directory to the selected Album：将一个文件夹下的所有 textures 放入选定的 album。有时并不起作用，估计是该文件夹下的文件必须符合 2 的幂次的尺寸。

 Delete the selected texture：删除选定的 texture。有确认询问。

当选定一个 album，可通过上方的 preview size 和 view 进行 textures 的筛选，被选中的 texture 将会出现在左边的列表里同时在图片格式显示出来。

当选中一个 texture 时，上方会出现一些按钮，

Back：返回 album；

Previous：转到预览界面；

Next：转到下一个 texture；

Copy to Album：复制到另外一个 album；

右下方的按钮

Make Active：激活当前选定的 texture；

Active and Close：激活当前选定的 texture 同时关闭材质浏览器；

Reload All：

Close：关闭材质浏览器。

7.2.6 杂项面板

杂项面板里主要包含 Undo（撤销）、Sets（成套）和 Select（选择）3 个选项卡。

Undo（撤销）窗格里是 Undo 堆栈，显示所有历史操作，选择其中的选择可用回到相应的操作状态。

Sets（成套）允许我们为自己创作的对象分组，这是一个好的管理项目的方法。

Select（选择）允许我们选择所有场景里的对象，但真正有用的是 Concave（凹面）选项，通过它可以找到表面为凹面的 brush，然后进行修改，因为具有凹面对象的文件在输出.DIF 文件是会出现错误。

7.3 制作.DIF 文件

根据实际操作积累的经验，在制作 Interior 对象时，推荐使用搭“积木”的方式，即为墙壁创建 brush（画刷，以一种纯数学描述定点位置信息的建模方式），如一个型 box 的简单房屋，为其创建前、后、左、右、上等 5 个 brushes，而“下”bursh 可以视实际情况决定是否创建（可以减少 brush 使用量，提高效率）。

注意：在 Constructor 中，支持先创建一个 box 的 brush（画刷），然后采用“挖空（hole）”命令，在弹出的对话框中选择挖空后留下的墙壁厚度，输入一个希望的值确定，即可形成中空的 box。但是该操作实际上是将 box 进行了“布尔减”运算，相当于对 box 进行了切割操作，会导致产生许新的 brush，当场景中的对象多次使用该方法，会大大降低游戏速度，因此，强烈建议不要使用这种方法建模。

使用 Constructor 进行 Interior 对象制作的时候，在模型制作的同时，会自定为模型添加上材质编辑栏中默认的材质。这样，需要我们在制作模型之前为其做好相应的材质，然后直接使用。

在 Constructor 的安装文件夹下，所有的材质都放置在 texture 文件夹中。里面已经有 auxiliary、colors、starter、tgedemo 和 uvchart 五个材质文件夹。其中 starter 和 tgedemo 文件夹中有许多材质，请您留意一下，可能在您的游戏中就可以使用。

在开始制作 interior 模型需要的材质文件时候，更好的方法是我们自己创建一个属于我们自己的材质文件夹。假设我们的材质文件夹的名字是 neu（名字没什么特别，是作者所在大学的英文 Northeastern University 的缩写），那么，请先在 texture 文件夹下创建一个新文件夹命名为 neu 或者是您希望的名字，然后先将 auxiliary 文件夹下的 NULL.png 和 TRIGGER.png 文件复制到 neu 文件夹下。

注意：NULL.png 和 TRIGGER.png 是 torque 的保留识别材质，它们在游戏中是不会进行渲染的，因此能够节约内存空间，提高显示速度。比如，那些在游戏中任何角度都看不到的地方（面），完全可以指定为 NULL 材质,该材质在 Constructor 中显示为紫色，用以区分其它材质。TRIGGER 材质是当你创建触发器类型的对象时候使用的材质，该材质在 Constructor 中显示为淡绿色，用以区分其它材质。很可能您暂时不会用到它。复制这两个材质的目的是在您制作模型时，就不需要为了指定这两个材质而频繁地在不同的材质文件夹中切换，提高工

作效率。

然后，请运行 constructor，点击材质工具栏中的 browse 按钮，弹出如图 7.4 的材质浏览器对话框，点击  按钮，在 New Album Name 中输入 neu 或者您希望的名字，然后确定。这样就会在您的材质浏览器里添加了一个 neu 的文件夹，但是目前的文件夹中的内容是空的，您可以通过下面的其它材质添加按钮加入您的材质。

接下来我们了解一下 Constructor 中的单位。Constructor 中，默认的 1 单位＝1 米（游戏中的），同时，1 单位＝32 象素＝1 米，即创建一个 32×32 象素的正方形材质，相当于游戏中 1 平方米的大小。

当准备工作都做好之后，我没就可以进行模型制作了。由于本书的作者并不是一名专业的美术人员，因此制作不出来更加精美的模型。这里只是为您提供一些思路上的帮助，相信您通过自己的努力，一定能作出令人满意的模型来。为了提高效率，我们尽量使用 Constructor 为我们提供的模版。

1.打开 Constructor 的编辑界面，找到 Tools 面板下的 Prefab 选项卡，单击  按钮，在任意 4 个视图中点击一下，即可以得到一个现成的类似城墙的建筑，如图 7.5 所示：

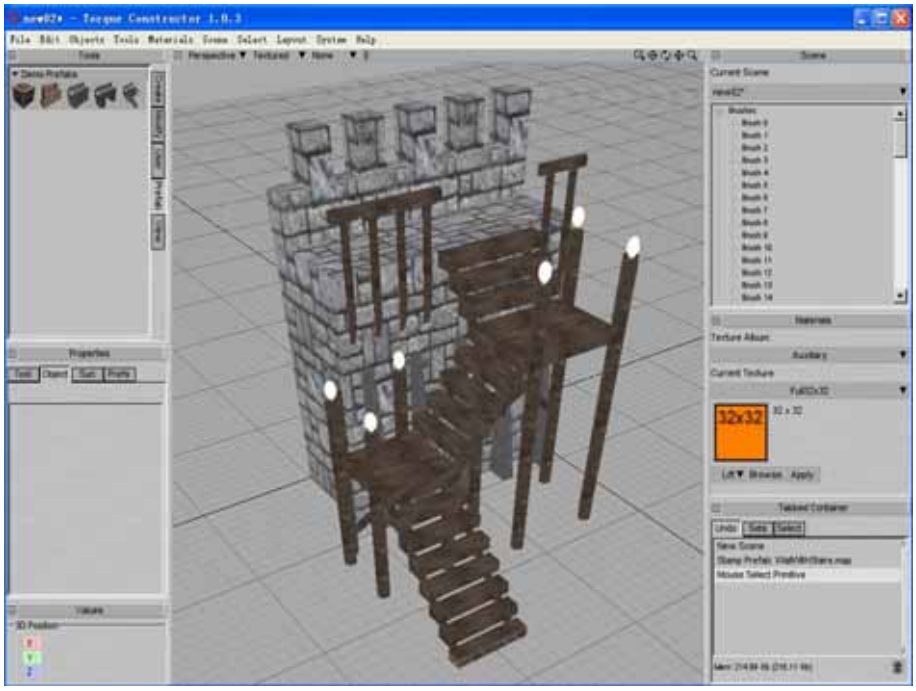


图 7.5　使用 Prefab 的内容创建模型

2．使用 Prefeb 中的其它的几个图标，创建出城门和城墙，移动它们的位置直到您满意为止。如图 7.6 所示：

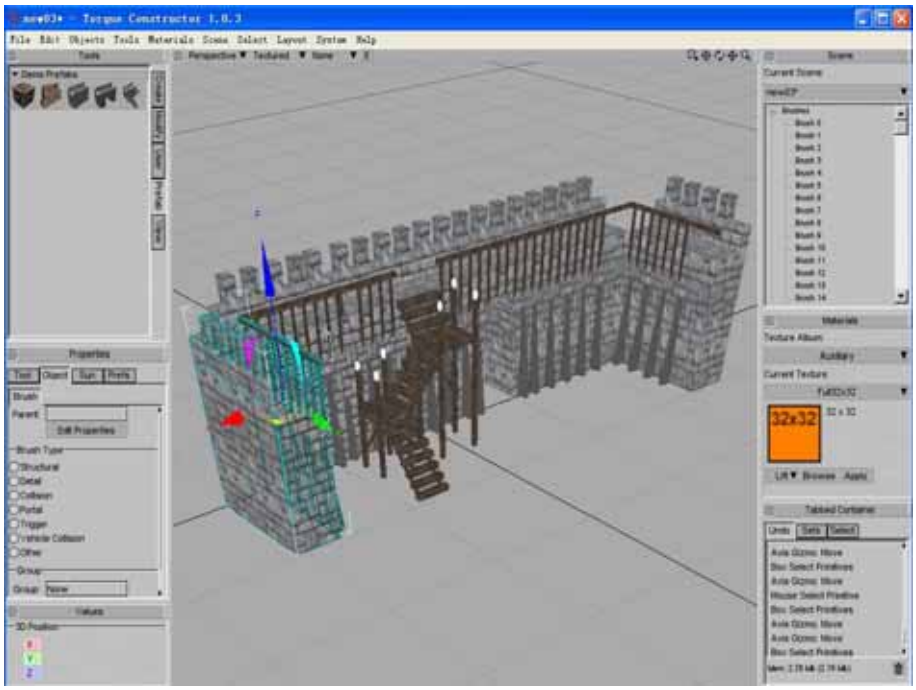


图 7.6　逐步创建您的城堡

3．在此基础之上，添加更多的 burshes，一步一步地制作出令您满意的城堡模型。

注意：考虑到游戏的渲染速度，本着速度至上的理念，在制作该城堡的时候可以使用下面的一些技巧。由于城堡的底部的所有面在游戏中是看不到的，因此，将它们的贴图全部指定为 Null.png 贴图，同时，将所有靠在一起的 brush 的靠在一起的面也全都指定为 Null.png 贴图（将一些多余的小 bursh 修饰物品，如墙侧面的三角物，扶手等，完全可以删掉来降低 brush 的数量）。

4．制作完毕后，点击 File→Save，将做好的模型保存为.map 文件，名字只要你喜欢就行。然后在点击 File→Export As DIF 命令，在弹出的窗口输入相应的名字，保存成.dif 文件，该文件格式是 Torque 中使用的格式。

注意 如果您使用的 Torque 游戏引擎的版本在 1.5 之下，请使用 File→Export As Legacy DIF 命令，否则输出的文件可能不被 1.5 版本之下的引擎所支持。

5．将您的.dif 文件和相应的材质文件复制到您的游戏文件夹中的 data\Interior 目录下，打开 Torque 游戏引擎，在 World Editor Creator 中找到您的城堡（相信现在这样的操作已经完全难不到您了），将它添加到游戏中，是不是非常酷啊！

控制您的角色在里面转转，楼上楼下地跑跑，是不是相当兴奋？

注意：如果您愿意，完全可以在 Inteiror 目录下为每个您的建筑物继续创建文件夹，如 fort 文件夹来保存您的城堡。而且进行文件夹管理是个相当不错的选择。需要说明的是，Torque 引擎运行时候会在以 Interior 文件夹为根文件夹（其实应该是 data 文件夹为根文件夹），进行材质搜索，也就是说您模型的.dif 文件可以放到 *data\interior\fort*下，而相应的材质文件完全可以放在 *data\interior*文件夹下！这样做的好处是，如果您的多个建筑物模型使用同一个材质，这种情况是非常可能发生的，那么可以节约相当的内存空间。推荐使用这种方式存储文件和材质。

7.4 DIF 的 LOD

7.4.1 为什么使用 LOD

LOD 是 Level of Details 的缩写，可以理解为“精细等级”。不可否认，模型的精细度越高，表现的效果就越好，但占用的资源也越多，运行起来的效率就越低。相信大家都有过玩游戏非常“卡”的经历。当游戏的画面运行非常不流畅的时候，再精美的画面对玩家的意义会远远小于速度上的要求。试想一下，当您的角色漫步在高楼大厦林立的现代化都市之中，每前进一步都要耗时几秒钟，您能忍受得了么？要是我肯定会退出该程序，给我的电脑放个小假！因此，LOD 就显得非常重要。

LOD 技术将我们的 DIF 模型制作成几个不同精细等级的模型，当我们的玩家角色距离模型很近的时候（这个判断是通过模型在游戏窗口上占用的象素高度值判断的），Torque 引擎将使用精细等级最高（通常为 0 等级）的模型进行渲染，而随着玩家逐渐远离该模型，Torque 引擎将自动替换该精细度最高的模型为精细度较低的模型，这样将大大提高显卡的渲染效率。

7.4.2 如何制作 LOD

每个等级 DIF 都有一个细节数字和最小象素值，当它在屏幕上显示的高度的像素值达到最小象素值时，Torque 引擎将自动使用对用的 DIF 等级来显示。不

幸的是，DIF 物体的 LOD 只能手工制作，没有 max 中类似 Multires 修改器这样的捷径。

每个 LOD 等级必须有对应其的.map 文件，并且这些.map 要遵循适当的命名规则，该命名规则仅仅需要在名字末尾添加“ _#.map ”，起始为 0。例如，一个具有 3 级 LOD 的 DIF 对象的 map 文件命名如下：

Anyobject_0.map，

Anyobject_1.map，

Anyobject_2.map。

后缀为 0 的是细节度最高的，而后缀为 2 的是最低的。

注意：将这些文件放在同一个文件夹下是非常重要的。

现在需要编译全部的.map 文件为一个.DIF 文件，操作很简单，只要运行 map2dif 来编译 anyobject_0.map 文件即可。操作方式为：将*_0.map 文件直接拖拽到 map2dif 文件上，将会自动进行转换操作。由于该文件有_0 后缀，map2dif 会自动在同一文件夹下寻找其后续 maps（_1、_2 等）文件，直到全部，同时将它们添加到编译的 DIF 文件内。

注意：加入 LOD 的.dif 文件会比没有加入 LOD 的.dif 文件占用的空间多一些，多出的部分由 LOD 的级别所决定。相比使用 LOD 后能够成倍的提高游戏显示速度，这个买卖是相当合算的。

在制作.dif 文件前，需要知道如何操作这些.map 文件才能正确实现 LOD。需要对 map 属性部分进行简单编辑。主要在于“ 细节数字 ”和“ 最小象素 ”。这两个值必须恰当设置才能保证.map 正常工作。

细节数字

“ 细节数字 ”非常简单，在保存 map 文件时候添加相应的数字即可。例如，你保存的文件名为“ mymap_0.map ”，然后使用任何一款您喜欢的编辑软件，打开该.map 文件，您会看到如下内容：

{

"classname" "worldspawn"

"classname" "worldspawn"

"detail number" "0"

"min_pixels" "250"

"geometry_scale" "32.0"

"light_geometry_scale" "32.0"

"ambient_color" "0 0 0"

"emergency_ambient_color" "0 0 0"

"mapversion" "220"

"mapversion" "220"

……

其中的细节数字“ detail ”值就应该是 0。如果是保存的文件名是 1，则细节数字要为 1，依此类推。

最小象素值

最小象素值的大小决定何时在两个细节间转换。当对象在屏幕上显示的尺寸高度小于最小象素值时，将会转换到下一级别。每个 map 文件里都有这个最小象素值，因此，离一个对象越近，这个对象越复杂，则这个最小值应该越大。min_pixels表示的是这个最小象素值，这里表示250象素。

注意：您要制作的建筑物的实际尺寸决定了该最小值的大小，在最终作出决定时（制作多少个LOD）需要花费一定的时间进行尝试。

7.5 本章小结

本章，我们学习了如何使用 Constructor 为 Torque 游戏引擎制作.DIF 格式文件，由于使用其它软件制作的文件在导出时经常会发生问题，所以再次建议您使用 Constructor 制作游戏中的建筑物模型。下一章，我们将学习如何使用 3D Studio Max 7.0 为 Torque 游戏引擎制作.DTS 格式文件。

第八章 制作.DTS 文件

8.1 什么是 DTS

美工艺术品是一款游戏的重要组成部分。优秀的美工作品是一款游戏吸引玩家的必要条件。在进行游戏艺术品创作之前，您不仅要学会如何为Torque游戏引擎制作专用的模型，更要学会如何制作更加高效率的模型，尤其是这种实时渲染的游戏。

同3D的动漫影片不同，游戏引擎必须具备实时、高效率地渲染数量巨大的几何体的能力。因此，在这个渲染的过程中，我们就需要进行许多巧妙的设计使之最优化，提高渲染效率。在Torque游戏引擎中，我们使用DTS这种模型格式，它同时具有灵活和最优化的效果。

以下是在Torque游戏引擎中需要是用DTS格式来制作的模型。

●小的复杂的模型：包括从岩石到树木等一些独立的对象。它们可以绑定到玩家或者车辆模型上，如武器、包裹、旗帜等。所有的DTS对象都可以包含动画。

●人物模型 :这是DTS格式真正强大所在。人物可以使用骨骼动画，配合script的使用，人物的可以通过组合简单的动作实现复杂的动作（姿势）。

●车辆：同人物模型相似，使用DTS表现车辆效果非常棒！玩家模型和武器可以绑定到车辆上，同时它们可以被玩家控制。

一些具备特点的对象：如旋转的雷达塔，坦克的炮塔以及其它能动的对象。这类的DTS对象在尺寸上应该小于建筑物。

下面列出的是一些不能使用 DTS 格式表现的物体。如果您试图使用 DTS 格式来制作这些物体，您会发现它们通常不是像你设想的那样正常工作。在创建一个模型之前，请仔细考虑好您需要它在游戏中做什么？在您并不具备足够的编程能力之前，尽量避免完全改变您为它初衷设计的作用。

- 建筑物和其它巨大的物体：.DIF 格式的文件更适用于这些物体，它们提供了更多、更高效的裁减算法。对那些巨大的可以使角色进入的建筑物，使用 DIF 格式使非常棒的选择。

- 需要具备复杂和精确碰撞的物体：使用.DIF格式的对象更适用于这样的物体。游戏引擎最复杂的功能是处理碰撞，对于复杂的碰撞，DTS 的模型不够优化。它只在小范围内检测简单的碰撞时做的不错，而在一些特殊的情况下则无能为力，比如当玩家角色在一个很大的物体上行走时。
- 一些占据几个屏幕长的物体：DTS 格式的对象最好小于一个屏幕的大小，如玩家角色，车辆或者树木等等。当一个 DTS 对象的部分非常长，那么，当这个部分的中心离开屏幕距离较远的时候，这个部分将会被裁减掉。通过使用 DIF 格式的对象就可以解决这个问题，或者是将大的 DTS 对象制作成几个小一些的对象。

8.2 Max2DTS Exporter

本节我们来学习如何使用 max2dtsExporter 将 3D Studio Max7.0 制作的模型转换成 Torque 游戏引擎可以使用的 DTS 文件格式。在此之前，请从 GarageGames 的官方网站上下载其为我们提供的教学包 torque_max_filepack.zip 压缩文件，大小约为 1.8MB。该包中的内容主要包括一下几个部分：

dtsGlobal.cfg 文件

/player folder:包括玩家角色输出需要的文件：

player.cfg: 玩家角色模型输出时默认使用的 configuration 文件。

player.cs: 玩家角色相应的脚本文件。

player.png: 玩家角色相应的材质。

player.max: 玩家角色模型的 max 文件。

/simpleshape folder: 包括 simpleshape 的全部文件：

SimpleShape.max (1～6):6 个 max 格式的 SimpleShape 文件.

SimpleTexture.png, SimpleTexture4.png: SimpleShape 相应的材质文件。

adjustLODs.mcr: 用于使用 MultiRes 修改器进行 LOD 设置的脚本文件。

意：虽然到目前位置，Autodesk 公司的 3D Stuidio Max 已经发行了 9.0 版本，但是由于兼容性等问题，该插件在 8.0 及其以上版本进行格式转变输出时会发生错误，而在 3D Stuido Max 7.0 版本则没有问题，因此，下面的所有内容全部都是针对 3D Stuidio Max 7.0 版本进行说明的。

8.2.1 安装 DTS Exporter

将“ max2dtsExporter.dle ”文件复制到您的 max 安装目录的“ Plugins ”文件夹下，打开 3D Stuidio Max 7.0 应用程序，在右上方找到 utility（工具）按钮，即  图标，下面您会看到 10 个默认的按钮，点击“ Sets ”按钮右边的图标  ，在弹出的对话框中，如图 8.1 所示，找到 DTS Exporter Utility 按钮，使用鼠标左键点住该按钮，然后拖拽到右边的任何一个按钮的位置上，这时候，该按钮将会替换原来的按钮，点击“ OK ”确定。这样，在我们以后需要将制作好的模型导出为 DTS 格式文件的时候，非常方便地调出 DTS Exporter Utility 面板。

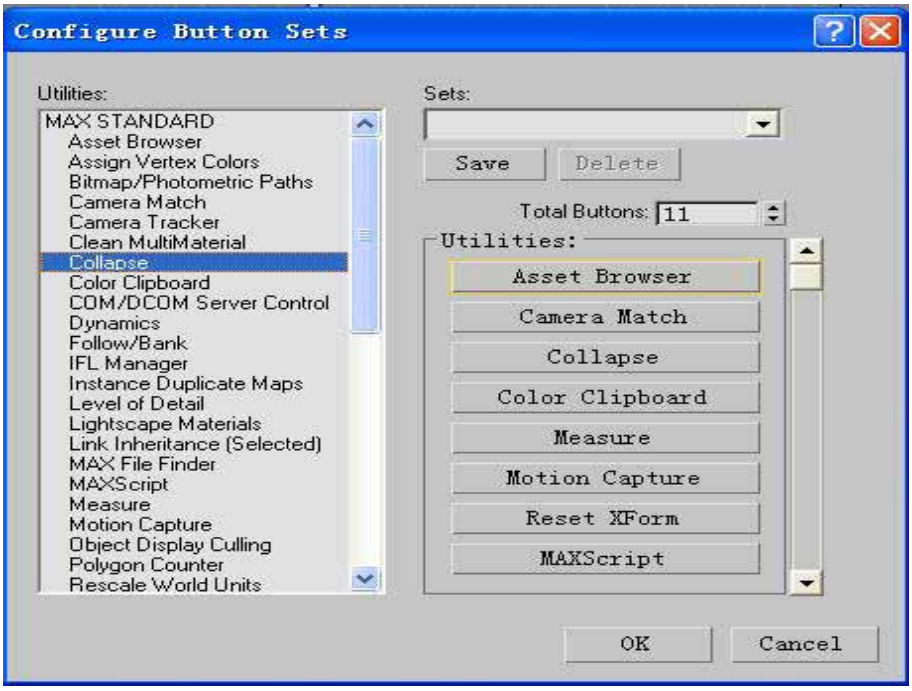


图 8.1　Configure 按钮设置面板

所有的准备工作完成之后，我们开始尝试创建我们的模型并将其导出为 DTS 格式，最终可以被我们的 Torque 游戏引擎使用。

1．创建模型：

首先，我们将会使用 3D Studio Max 创建一个简单的球体模型。

1) 在任何一个视图中，使用创建面板下的 sphere 按钮，创建一个 16 sides 的球体（sphere）。确保创建的球体的坐标轴同世界坐标轴对齐，这在输出时候是非常重要的。

2) 确保该模型的“ mapping coordinates ”选项（位于创建面板的底部）处于选中状态。不使用 mapping coordinates 的 Meshes（网格）在游戏引擎中将不能正常工作。再次确认您想要输出的 meshes（网格）上具备 UV 坐标。

3) 将球体的中心同世界坐标的原点对齐，可以使用移动工具完成操作，右键单机移动图标  （类似“ ＋ ”的图标），在探出的坐标对话框中，手工输入 0，使得该球体的位置中心同原点重合。

4) 将该球体命名为“ SimpleShape ”。

注意：到目前为止我们并没有做好输出 DTS 格式的全部工作。

2．添加材质

1) 复制 SimpleTexture.bmp 文件到目标文件夹。为了使其正常工作，材质和模型必须处于同一个文件夹下。

2) 在 3D Studio Max 中打开材质编辑器，并选择一个材质球。打开“ maps ”面板，在 diffuse（漫反射）中，点击后面的按钮，找到我们的 SimpleTexture.bmp 文件，选择它。

3) 点击“ Show map in Viewport ”按钮（类似一个魔方立方体的图标）。

注意：到目前为止我们并没有做好输出 DTS 格式的全部工作。

3．创建 Bounding Box

1) 创建一个新的 box，用它将所有的 shape 包含进去。

2) 使用上面的方法将其中心对齐世界原点。

3) 将这个 box 命名为“ bounds ”。

注意：到目前为止我们并没有做好输出 DTS 格式的全部工作。

4．添加细节等级数

1) 选中 SimpleShape 球体。

2) 在“ Utility（工具）”选项卡下找到我们前面添加的“ DTS Exporter Utility ”按钮，单击该按钮，在其下方看到一系列展开的选项内容。在 Utilities 下的 General 栏目中，找到“ Renumber Selection ”按钮，该按钮为 General 项目下的第一个按钮，由于显示的问题，您可能看到的是“ number selection ”。不要紧，点击该“ Renumber Selection ”按钮，得到一个名为

“ Renumber “ 的对话框。

3) 在 Renumber 对话框中，在 New number 后面的文本框中输入“ 2 ”，然后点击 OK。该操作将会为你的模型添加一个数字后缀（它用来进行细节处理）。

4) 确保 SimpleShape（不是 SimpleShape2）处于选中状态，然后在 General 项目下，点击“ Embed Shape ”按钮（上数第二个按钮）。该按钮将为场景中的模型自动创建层级链接关系。点击工具栏上的  按钮，打开层级面板，得到如图 8.1 所示内容：

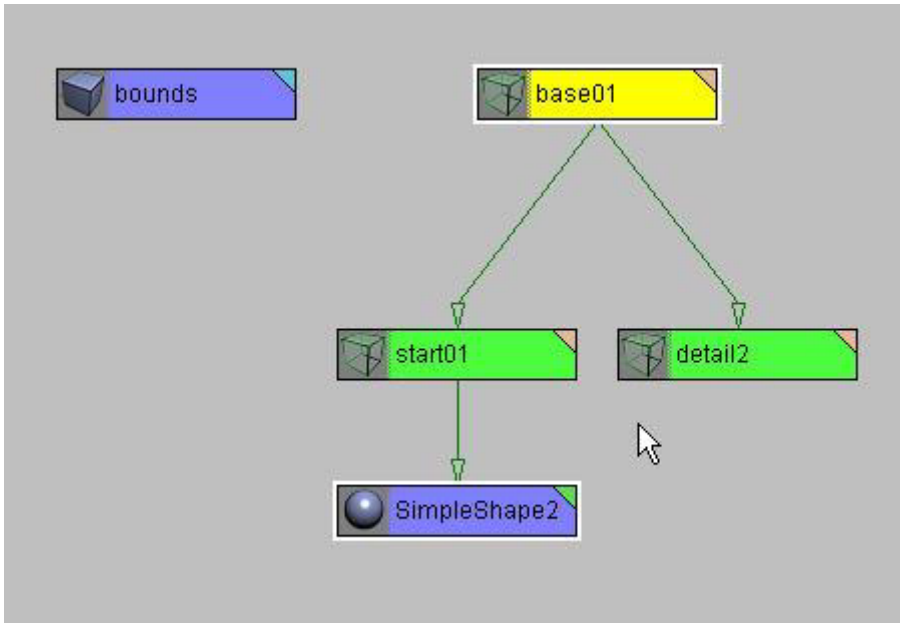


图 8.1　层次关系图

Embed Shape 按钮会正确创建出需要的层次关系，使得您的模型可以顺利导出。同时也会创建一些 dummy（虚拟点）对象，并且为它们自动地正确命名。该过程同样可以通过手工操作实现，使用 Max 的 dummy（虚拟点）对象，创建 base、start 以及 detail 标记点。

现在已经为输出 shape 准备好了。

5．输出

1) 打开 Utility（工具）面板，选择“ DTS Exporter Utility ”。

2) 使用 Export 下的“ Whole Shape ”进行模型导出。

3) 在弹出的窗口，选择您将要保存的目标文件夹，命名为 SimpleShape.dts 或者是任何您喜欢的名字。

注意：请确保材质文件同输出的模型文件位于相同的文件夹下，否则，在游戏中您的模型将显示为白色。

请在 Show Tool 中观察 SimpleShape.dts 文件。一切就是这么简单。

6．创建细节等级

这里我们将为 Simple Shape 创建细节等级，它的作用是，当我们在游戏中远离该物体时，降低该物体的模型面数，也就是说，用另一个面数相对较低的模型代替该模型（模型的大小不变，只是面数减少）。如果处理得好，这种细节的变化对于玩家来说是察觉不到的。随着物体与玩家眼睛距离的变大，引擎会在满足条件的时候自动替换细节较高的模型为细节较低的模型。而判断的标准是，该对象的高度在显示器上占用的象素值。

这里我们将为 Simple Shape 创建细节等级。

1) 选择 SimpleShape2。
2) 将其重命名为 SimpleShape128。
3) 复制该模型，使用 Edit →Clone，确保选中 copy 选项，而不是 reference 或者 instance 选项。
4) 将这个新复制出来的模型重命名为 SimpleShape64。打开 modifier（修改）面板，将“ segement（分段）”改为 8。这表明当该对象在屏幕上的高度介于 64～128 象素之间的时候，使用 8 分段的模型进行渲染。
5) 继续重复上述操作，这次命名为 SimpleShape2，并且将分段数设为 4。这表明当该对象在屏幕上的高度介于 2～64 象素之间的时候，使用 4 分段的模型进行渲染。
6) 断开 SimpleShape64 和 SimpleShape2 同 Start01 的链接。
7) 选中 Base01，在 DTS exporter 面板上点击“ register details ”按钮，将会自动为您创建细节标记。现在，您可以直接输出.dts 文件，覆盖原来的即可。

在 Show Tool 对刚刚创建的模型进行细节等级查看，通过按住 alt 键和鼠标滚轮可以快速在不同的细节等级上转换。

7．添加碰撞检测盒

在游戏中，玩家和车辆等物体之间会经常发生碰撞，从而会产生一些交互。

那么，游戏程序是如何判断碰撞是否发生呢？在 Torque 中，我们使用碰撞检测盒子。简单的说，就是在玩家角色或者其它物品的模型外部再创建一个将它们包围起来的“ 盒子 ”，我们称之为“ 碰撞检测盒 ”，当两个物体之间的碰撞检测盒子在游戏中发生了碰撞时，引擎就会知道了，同时如果相应的程序，引擎会自动调用这些程序，比如玩家遇到了武器，会捡起来，遇到了车辆，会“ 钻进去 ”，遇到了垃圾桶，会撞飞它等等。

可见，为这些物品创建碰撞检测盒子是非常重要的。

要想为一个物体创建碰撞检测盒子，需要创建两种类型的对象。

1) 实体 mesh 对象。这些实体 mesh 用来包围您的模型，创建方式类似 bounds 对象。命名标准为“ Col-*n* ”，*n* 表示 1～9。
2) 虚拟体对象。这些对象使用 dummy 进行创建，命名标准为“ Collision-*n* ”，*n* 表示 1～9，同前面的实体对象编号相对应。

注意：如果您只为您的模型创建 1 个碰撞检测盒子，通常是一个 box，那么只需要创建一个名为“ Col-1 ”的 box 和一个名为“ Collision-1 ”的 dummy（虚拟点）对象就 OK 了。若是您需要为您的模型建立更细致的碰撞检测，比如玩家角色的头部、左右手臂、左右腿等的检测，那么就需要在相应的位置创建相应的碰撞检测盒子，因此，Torque 为我们提供了最多 9 个可以创建的碰撞检测。此外，确保您的碰撞检测盒子是凸状而非凹状结构，否则在输出的时候会出错。

创建碰撞盒子的步骤：

1) 打开前面创建的 SimpleShape.max 文件。
2) 复制细节标记（detail-1），将其命名为“ Collision-1 ”。当然，您也可以直接添加一个 dummy 对象，然后命名为“ Collision-1 ”，然后将其链接到“ base01 ”。
3) 复制 SimpleShape128，将其 segments 改为 6，然后命名为“ col-1 ”。

注意：复制并不是一个好的创建碰撞检测盒的方法。这里使用这种方式只是为大家提供一种方法。对于非常复杂的物体，如玩家角色、车辆等，为它们创建一个简单的碰撞检测盒是非常重要的，通常我们手工创建一个 box 是最好的选择。并且，任何比 cube 或者 sphere 更复杂的碰撞检测盒基本上无效。

也就是说推荐大家使用下面步骤 4 的的方法而不是步骤 3 的方法制作碰撞盒子。

4）请按照该步骤创建碰撞盒子，而不是上面的 3）步骤。复制 bounds，然后命名为“ col-1 ”，如果您愿意，最好将该 box 的尺寸缩小一些，保证它比 sphere 大一些而比 bounds 的 box 小一些就行了。然后，在层级面板中，将 col-1 链接到 start01 下面，最终结果如图 8.2 所示。

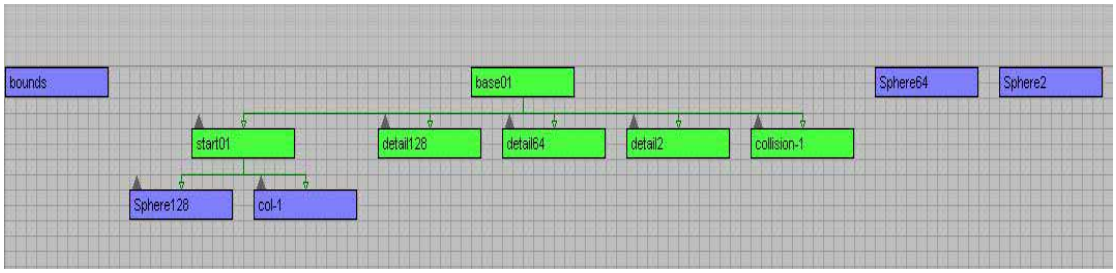


图 8.2　添加碰撞盒子有的层级关系

8.3 材质

本节我们将学习为 Torque 引擎制作材质。在学习之前，我们需要了解一下有关材质的信息。

1．Torque 不支持颜色贴图和程序贴图，支持 bitmap（位图）贴图。

2．Torque 支持多级子贴图，贴图必须是 bitmap（位图）贴图。

3．材质贴图的尺寸必须采用 2 的幂次，如 32、64、128、256、512 象素，Torque 支持的最大尺寸是 512 象素。材质不一定是方形的，即宽和高可以不相同，只要满足 2 的幂次。

4．Bitmap（位图）的格式必须是.PNG 或者是.JPG 格式。

5．所有的面必须使用 UV 坐标贴图保证正常输出。

6．Torque 支持透明材质。要想使一个物体具有透明效果，其材质必须具备 alpha 通道。透明的程度通过 alpha 通道控制。在材质编辑器中，在 diffuse 中载入需要的材质，选中“ Opacity ”钩选框，您不需要在 Opacity 中载入一个材质，关于 Opacity 的信息会从 alpha 通道获得。

7．Torque 支持双面材质。在材质编辑器中选中“ 2-sided ”就可以制作双面材质。默认情况下，该选项是未被选中的，如果需要，还要在 DTS esporter

中将其选中。

8.4 创建 Sequence 对象

Sequence（序列）是用来保存输出动画的。Sequence 会告诉 exporter（输出器）如何将 3D Stuido Max 中制作好的一段时间的动画输出成 Sequence（序列）文件。在 3D Stuido Max 中，sequence（序列）对象可以通过创建 helper（帮助）对象的方法进行创建。在“ Create（创建）”面板下选择“ Helps（帮助）”，在下拉菜单中选择“ General DTS Objects ”。如图 8.3 所示：

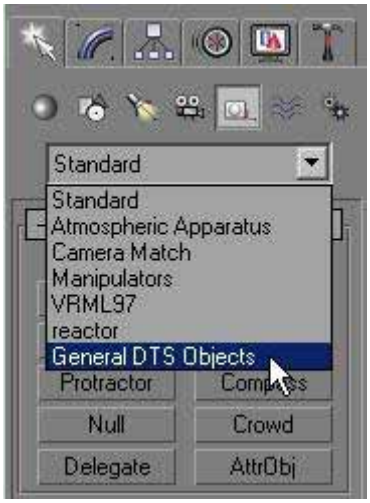


图 8.3　选择创建 DTS 对象

注意：您在场景中创建的 Sequence（序列）对象不需要链接任何对象。

Sequence（序列）对象通常用来定义一个动画序列的开始和结束。我们可以通过添加关键帧到 *Sequence Begin/End* 轨迹进行参数设置。这是一种开/关类型的轨迹。只有将其设置为 on（开）状态时，动画信息才能够正常输出。如图 8.3 所示：

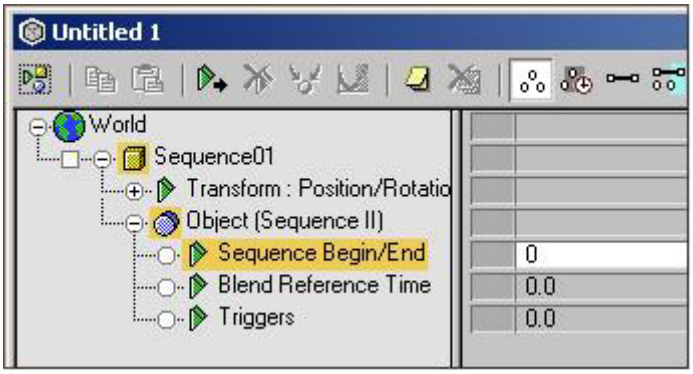


图 8.3　Sequence 控制面板

默认情况下，您添加的第一帧为动画的开始，第二帧为动画的结束。

为动画 Shape 添加 Sequence 对象

1、打开 SimpleShape5.max 文件。该 shape 已经具备了动画。只有细节等级最高的 shape 需要进行动画设置，而其它细节等级较低的则不需要。这适用于所有具有动画的对象，低级别的 shape 会自动继承最高级别 shape 的动画效果。
2、在 create（创建）面板下的 helper（序列帮助器）页面下，在下来菜单中选择“ General DTS Objects ”，然后在其下面点击“ sequence ”按钮，接着，将鼠标移动到任一视图之中，当鼠标显示为“ 十字架 ”形状的时候，单击鼠标左键拖拽出一个 sequence 对象，默认情况下，它会呗自动命名为 sequence01，请将其命名为 SimpleAnimation 或者是任何您喜欢的名字。
3、打开 trackview（轨迹查看器），找到 SimpleAnimation 轨迹。使用“ 创建帧 ”按钮，在 Sequence Begin/End 上在 0 帧处添加一个关键帧。在 40 帧处添加另外一个关键帧。这样就为 sequence（序列）创建了开始和结束点。关闭该面板，点击“ Whole Shape ”按钮，将其保存 Tutorial.base 下的/data/shapes/3dtorquelogo/文件夹下，名为 TestSequence.dts。

完成上述操作。输出该 shape 并在 ShowTool 中察看一下效果。当你载入它的时候，它应该处于动画的状态。滑动一下细节滑块，您会发现，所有细节等级的对象都会继承使用最高细节等级对象的动画。

注意：如果刚刚创建的带动画的模型文件不会在 torque 游戏引擎中自动播放，您还需要为其添加适当的代码才可以实现。具体方法如下：

首先，为该模型创建一个数据块，请输入一下脚本：

```
datablock StaticShapeData(TestSecquence)
{
   category = "demo";
```

```
    shapeFile = "~/data/shapes/3dtorquelogo/.dts";
};
```

然后运行torque，在场景中使用世界编辑器创建器，将其从shape中（注意，不是staticshape）添加到场景中。然后使用“ `”键，调出控制台窗口，输入Id.playthread(0, “SimpleAnimation”);命令，然后确定，就可以看到动画了。其中，Id为该对象的句柄。

8.4 本章小结

经过本章的学习，我们已经可以使用 3D Studio Max7.0 制作人物的模型，并使用插件将其输出为 Torque 可以使用的.DTS 格式文件，从而极大地丰富了您的游戏世界。

现在，关于 Torque 游戏引擎的基本知识您已经全部掌握了，相信您的脑海里已经勾勒出您的游戏的蓝图了，放手去做吧，一切皆有可能。

附　录

表 1　Torque Script 对象类型掩码

掩码标识符	号　码	掩码位位置
DefaultObjectType	0	0
StaticObjectType	1	1
EnvironmentObjectType	2	2
TerrainObjectType	4	3
InteriorObjectType	8	4
WaterObjectType	16	5
TriggerObjectType	32	6
MarkerObjectType	64	7
Unassigned	128	8
unassigned	256	9
decalManagerObjectType	512	10
GameBaseObjectType	1024	11
ShapeBaseObjectType	2048	12
cameraObjectType	4096	13
StaticShapeObjectType	8192	14
PlayerObjectType	16384	15
ItemObjectType	32768	16
VehicleObjectType	65536	17
VehicleBlockerObjectType	131072	18
ProjectileObjectType	262144	19
ExplosionObjectType	524288	20
Unassigned	1048576	21
CorpseObjectType	2097152	22

DebrisObjectType	4194304	23
PhysicalZoneObjectType	8388608	24
Unassigned	16777216	25
StaticTSObjectType	33554432	26
StaticRenderdObjectType	67108864	27
Unassigned	134217728	28
Unassigned	268435456	29
Unassigned	536870912	30
unassigned	1073741824	31

表 2　Torque 对象方法

对象类	方法
AIPlayer	.moveForward() .walk() .run() .stop() .setMoveSpeed(float) .setTargetObject(object) .getTargetObject() .targetInSignt() .aimAt(point) .getAimLocation()
BanList	.add(id, TA, banTime) .addAbsolute(id, TA, banTime) .removeBan(id, TA) .isBanned(id, TA) .export(filename)

Camera	.getPosition() .setOrbitMode(obj, xform, min-dist, max-dist, cur-dist) .setFlyMode()
DbgBreakPointView	.set(file, line,,,) .remove(file,line) .condition(clear, passct, condition) .isSet(file, line) .clear()
DbgCallStackView	.add(file, line, function) .getFrame() .clear()
DbgFileView	.setBreak(line) .removeBreak(line) .findString(Text)
DbgWatchView	.set(expression) .update(queryId) .edit(newValue) .remove() .queryAll() .clear()
Debris	.init(position, velocity)
DebugView	.addLine(startPt, endPt, color) .clearLines() .SetText(line, text,[colorF]) .ClearText()
EditTSCtrl	.renderSphere(pos, radius, =) .renderCircle(pos, normal, radius, =) .renderTriangle(pnt, pnt, pnt) .renderLine(start, end)

FileObject	.openForRead(filename) .openForWrite(filename) .openForAppend(filename) .writeLine(text) .isEOF() .readLine() .close()
FlyingVehicle	.setCreateHeight(bool)
GameBase	.getDataBlock() .getDataBlock(DataBlock)
GameConnection	.chaseCam(size) .setControlCameraFov(fov) .getControlCameraFov() .transmitDataBlocks(seq) .activateGhosting() .resetGhosting() .setControlObject(%obj) .getControlObject() .isAIControlled() .play2D(AudioProfile) .play3D(AudioProfile, Transform) .isScopingCommanderMap() .scopeCommanderMap(bool) .listenEnabled() .getListenState(clientId) .canListen(clientId) .listenToAll() .listenToNone() .setVoiceChannels(0-3)

	.setVoiceDecodingMask(mask) .setVoiceEncodingLevel(codecLevel) .setBlackOut(fadeTOBlackBool, timeMS) .setMissionCRC(cre)
GuiBitmapCtrl	.setBitmap(blah) .setValue(xAxis, yAxis)
GuiCanvas	.renderFront(bool) .setContent(ctrl) .getContent() .pushDialog(ctrl) .popDialog() .popLayer() .cursorOn() .cursorOff() .setCursor(cursor) .hideCursor() .showCursor() .repaint() .reset() .isCursorOn() .getCursorPos() .setCursorPos(pos)
GuiControl	.getPosition() .getExtent() .getMinExtent() .resize(x, y, w, h) .setValue(value) .getValue() .setActive(value)

	.isActive() .setVisible(value) .isVisible() .isAwake() .setProfile(profileI) .makeFirstResponder(value)
GuiEditCtrl	.addNewCtrl(ctrl) .select(ctrl) .setRoot(root) .setCurrentAddSet(ctrl) .toggle() .justify(mode) .bringToFront() .pushToBack() .deleteSelection()
GuiFilterCtrl	.getValue() .setValue(f1, f2, …) .identity()
GuiFrameSetCtrl	.frameBorder(index, enable) .frameMovable(index, enable) .frameMinExtent(index, w, h)
GuiInspector	.inspect(obj) .apply(newName)
GuiMessageVectorCtrl	.attach(MessageVectorId) .detach()
GuiPopUpMenuCtrl	.sort() .add(name, idNum, {scheme}) .addScheme(id, fontColor, fontColorHL, fontColorSEL) .getText()

	.setText(text) .getValue() .setValue(text) .clear() .forceOnAction() .forceClose() .getSelected() .setSelected(id) .getTextById(id) .setEnumContent(class, enum) .findText(text) .size() .replaceText(bool)
GuiSliderCtrl	.getValue()
GuiTerrPreviewCtrl	.reset() .setRoot() .getRoot() .setOrigin(x, y) .getOrigin() .getValue() .getValue(t)
GuiTextListCtrl	.getSelectedId() .setSelectedById(id) .setSelectedRow(index) .clearSelection() .clear() .addRow(id, text, index) .setRow(id, text) .getRowId(index)

	.removeRowById(id) .getRowTextById(id) .getRowNumById(id) .getRowText(index) .removeRow(index) .rowCount() .scrollVisible(index) .sort(colId, {increasing}) .findText(text) .setRowActive(id) .isRowActive(id)
GuiTreeViewCtrl	.open(obj)
HTTPObject	.get(addr, request-uri) .post(addr, request-uri, query, post)
InteriorInstance	.activateLight() .deactivateLight() .echoTriggerableLights() .getNumDetailLevels() .setDetailLevels(level)
Item	.isStatic() .isRotating() .setCollisionTimeout(object) .getLastStickyPos() .getLastStickyNormal()
Lightning	.warningFlashes() .strikeRandomPoint() .strikeObject(id)
MessageVector	.deleteLine(DeletePos) .clear()

	.dump(filename{, header})
	.getNumLines()
	.getLineText(Line)
	.getLineTag(Line)
	.getLineTextByTag(Tag)
	.getLineIndexByTag(Tag)
PhysicalZone	.activate()
	.deactivate()
Player	.setActionThread(sequenceName)
	.setControlObject(obj)
	.getControlObject()
	.clearControlObject()
	.getDamageLocation(pos)
Precipitation	.setPercentage(percentage <1.0 ~ 0.0>)
	.stormPrecipitation(percentage <0 ~ 1>, Time)
SceneObject	.getScale()
	.getWorldBox()
	.getWorldBoxCenter()
	.getObjectBox()
	.getForwardVector()
ShapeBase	.setShapeName(tag)
	.getShapeName()
	.playAudio(slot, AudioProfile)
	.playAudio(slot)
	.playThread(thread)
	.setThreadDir(thread, bool)
	.stopThread(thread)
	.pauseThread(thread)
	.mountObject(object, node)

	.unmountObject(object)
	.unmount()
	.isMounted()
	.getObjectMount()
	.getMountedObjectCount()
	.getMountedObjectNode(index)
	.getMountedObject(index)
	.getMountNodeObject(node)
	.mountImage(DataBlock, slot, [loaded = true], [skinTag])
	.unmountImage(slot)
	.getMountedImage(slot)
	.getPendingImage(slot)
	.isImageFiring(slot)
	.isImageMounted(DataBlock)
	.getMountSlot(DataBlock)
	.getImageSkinTag(slot)
	.getImageState(slot)
	.getImageTrigger(slot)
	.setImageTrigger(slot, bool)
	.getImageAmmo(slot)
	.setImageAmmo(slot, bool)
	.getImageTarget(slot)
	.setImageTarget(slot, bool)
	.getImageLoaded(slot)
	.setImageLoaded(slot, bool)
	.getMuzzleVector(slot)
	.getMuzzlePoint(slot)
	.getSlotTransform(slot)
	.getAIRepairPoint()

	.getVelocity() .setVelocity(Vector) .applyImpulse(Pos, Vector) .getEyeVector() .getEyeTransform() .setEnergyLevel(value) .getEnergyLevel() .getEnergyPercent() .setDamageLevel(value) .getDamageLevel() .getDamagePercent() .setDamageState(state) .getDemageState() .isDestroyed() .isDisabled() .isEnabled() .applyDamage(value) .applyRepairRate(value) .getRechargeRate() .setRechargeRate(value) .getRepairrate() .getControllingClient() .getControllingObject() .canCloak() .setCloaked(true|false) .isCloaked() .setDamageFlash(flash level) .getDamageFlash() .setWhiteOut(flash level)

	.getWhiteOut() .setInvincibleMode(time, speed) .getCameraFov() .setCameraFov(fov) .hide(bool) .isHidden() .isHidden() .startFade(U32, U32, bool) .setDamageVector(vec)
ShapeBaseData	.checkDeployPos(xform) .getDeployTransform(pos, normal)
SimpleNetObject	.setMessage(msg)
Sky	sky.stormCloudsOn(0 or 1, Time) sky.stormFogOn(Percentage <0 ~ 1>, Time) sky.realFog(0 or 1, max, min, speed) sky.getWindVelocity() sky.setWindVelocity(x, y, z) sky.stormCloudsShow(bool) sky.stormFogShow(bool)
StaticShape	.setPoweredState(bool) .getPoweredState(bool)
TCPObject	.listen(port) .send(string, …) .connect(addr) .disconnect()
Terraformer	.canyon(dst, freq, turb, seed) .preview(dst_gui, src) .previewScaled(dst_gui, src) .clearRegister(r)

	.fBm(r, freq, 0.0~1.0{roughness}, detail, seed) .smoothRidges(src, dst, 0.0~1.0, iterations) .setTerrain(r)
Trigger	.getNumObjects() .getNumObjects(Object Index)
TriggerData	.enterTrigger(Trigger, ObjectId) .leaveTrigger(Trigger, ObjectId) .tickTrigger(Trigger)
WaterBlock	.toggleWireFrame()
WorldEditor	.redirectConsole(objID)

TorqueScript 运算符

符　号	含　义
+	加法运算符
-	减法运算符
*	乘法运算符
/	除法运算符
%	求模运算符
++	递增 1
--	递减 1
+=	递加赋值符
-=	递减赋值符
*=	递乘赋值符
/=	递除赋值符
%=	求模赋值符
@	字符串连接符

()	圆括号—提升运算优先级
[]	方括号—数组索引界定符
{}	花括号—指示代码块的开始与结束
SPC	空格符
TAB	制表符
NL	换行符
~	取反
|	按位“或”运算符
&	按位“与”运算符
^	按位“异或”运算符
<<	左移动运算符
>>	右移动运算符
|=	利用赋值给第一个操作数的结果进行按位或运算
&=	利用赋值给第一个操作数的结果进行按位与运算
^=	利用赋值给第一个操作数的结果进行按位异或运算
<<=	利用赋给第一个操作数的结果进行左移运算
>>=	利用赋给第一个操作数的结果进行右移运算
!	指定值的相反值
&&	当两个值都为真的时候，计算的结果为真
||	当两个值有一个为真的时候，计算的结果为真
==	左边的值等于右边的值
!=	左边的值不等于右边的值
<	左边的值小于右边的值
>	左边的值大于右边的值
<=	左边的值小于或等于右边的值
>=	左边的值大于或等于右边的值
$=	左边的字符串等于右边的字符串
!$=	左边的字符串不等于右边的字符串

//	注释符
;	语句结束符
.	对象、数据块的方法或者属性的定界符

Torque 数据块

数据块	父数据块
AudioDescription	SimDataBlock
AudioEnvironment	SimDataBlock
AudioProfile	SimDataBlock
AudioSampleEnvironment	SimDataBlock
CameraData	ShapeBaseData
DebrisDta	GameBaseData
DecalData	SimDataBlock
ExplosionData	GameBaseData
FlyingVehicleData	VehicleData
fxLightData	GameBaseData
GameBaseData	SimDataBlock
HoverVehicleData	VehicleData
ItemData	ShapeBaseData
LightningData	GameBaseData
MissionMarkerData	ShapeBaseData
ParticleData	SimDataBlock
ParticleEmitterNodeData	GameBaseData
PathCameraData	ShapeBaseData
PathedInteriorData	GameBaseData
PlayerData	ShapeBaseData
PrecipitationData	GameBaseData
ProjectileData	GameBaseData

ShapeBaseData	GameBaseData
ShapeBaseImageData	GameBaseData
SimDataBlock	None
SplashData	GameBaseData
StaticShapeData	ShapeBaseData
TSShapeConstructor	SimDataBlock
TriggerData	GameBaseData
VehicleData	ShapeBaseData
WheeledVehicleData	VehicleData
WheeledVehicleSpring	SimDataBlock
WheeledVehicleTire	SimDataBlock