**电 子 科 技 大 学**

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

**学士学位论文**

**BACHELOR THESIS**



论文题目  **基于Unity3D引擎制作的3D坦克大战游戏**

专 业 **计算机科学与技术**

学 号 **2013060110027**

作者姓名 **黄小伟**

指导教师 杨陈/罗嘉庆

摘 要

随着科技的发展，网络的普及，电子产品的普遍流行，游戏显然已经成为人们的主要娱乐方式之一。生活节奏的加快使得人们不能再像以前那样坐在电脑前玩游戏，便于携带是手机的主要特点，手游也成为当前最热门的游戏方式之一。于此同时，Unity3D 的异军突起，使手机游戏开发多了一个方向，更方便、更快捷、更加人性化的研发方式使得Unity3D 成为当前主流的游戏开发引擎。手机硬件性能越来越强大，使得 3D 游戏在手机中得以实现。近年来，越来越多的 3D 游戏出现在手机屏幕上，其中不乏优秀的作品。本次毕设在前人的基础上进行学习研究，深入了解 Unity3D 及 C#编程语言，进行 3D 格斗游戏的开发。本文列出了 3D 格斗游戏的一些功能点、关键技术，详细阐述了游戏的设计过程。

本次毕业设计的题目为《基于Unity3D引擎制作的3D坦克大战游戏》，旨在成功的设计出一 款可以运行在智能手机和各类电脑上的休闲、娱乐游戏，在毕业设计完成时，可以通过 多个平台，不同方式体验。游戏共包含若干关卡，从易到难。提供了一种放松、休闲的方式， 实现了对上班一族的工作压力缓解。实现娱乐、休闲、交流和取得虚拟成就的具有可持续性的个体性单人游戏。

**关键词：** Unity3D、坦克、UGUI、C#、组件

ABSTRACT

With the development of science and technology, the popularity of the Internet and the popularity of electronic products, the game has become one of the main entertainment modes of people. The acceleration of the pace of life makes it impossible for people to sit in front of the computer as they used to. It is the main feature of mobile phones that are easy to carry, and hand travel has become one of the most popular forms of games. At the same time, the sudden emergence of Unity3D, mobile phone game development in more than one direction, more convenient, faster and more humane way of research and development, making Unity3D become the mainstream of the game development engine. Mobile phone hardware performance is more and more powerful, make 3D game can be realized in mobile phone. In recent years, more and more 3D games appear on the mobile phone screen, many of them are excellent works. The completion of this study is based on previous studies, in-depth understanding of Unity3D and C# programming language, 3D fighting game development. This paper lists some function points and key technologies of 3D fighting game, and expounds the design process of the game in detail.

The graduation design topic is "the Unity3D engine production 3D Tankedaizhan game" based on the successful aims to design a leisure and entertainment game can run on smart mobile phone and all kinds of computer, in the graduation design is completed, through multiple platforms, different ways of experience. The game contains a number of checkpoints, from easy to difficult. It provides a way to relax and relax and relieve the working pressure of the working class. An individual, sustainable, single game that implements entertainment, leisure, communication, and virtual achievement.

**Keywords:** Unity3D, tanks, UGUI, C#, components

目 录

[摘 要 I](#_Toc483260141)

[ABSTRACT II](#_Toc483260142)

[目 录 III](#_Toc483260143)

[第一章 绪 论 1](#_Toc483260144)

[1.1 研究工作的背景与意义 1](#_Toc483260145)

[1.2 手机游戏产业的国内外研究历史与现状 2](#_Toc483260146)

[1.3 本文的主要贡献与创新 3](#_Toc483260147)

[1.4 本论文的结构安排 3](#_Toc483260148)

[第二章 Unity3D游戏引擎基础 4](#_Toc483260149)

[2.1 主界面视图 4](#_Toc483260150)

[2.2 游戏组件 6](#_Toc483260151)

[2.2.1 Gameobject(对象) 6](#_Toc483260152)

[2.2.2 Tranform（坐标） 6](#_Toc483260153)

[2.2.3 Camera(镜头) 6](#_Toc483260154)

[2.3 脚本系统 7](#_Toc483260155)

[2.3.1 C# 7](#_Toc483260156)

[2.3.2 Javascript 8](#_Toc483260157)

[2.4 物理引擎 8](#_Toc483260158)

[2.4.1 物理引擎简介 8](#_Toc483260159)

[2.4.2 物理引擎基本架构 9](#_Toc483260160)

[第三章 3Ｄ坦克游戏设计 10](#_Toc483260161)

[3.1 游戏基本介绍 10](#_Toc483260162)

[3.2 需求分析 11](#_Toc483260163)

[3.3 技术路线 11](#_Toc483260164)

[3.4 游戏框架结构设计 11](#_Toc483260165)

[第四章　3Ｄ坦克游戏实现 11](#_Toc483260166)

[4.1 图形界面实现 11](#_Toc483260167)

[4.1.1 开始界面 11](#_Toc483260168)

[4.1.2 加载界面 12](#_Toc483260169)

[4.1.3 游戏界面 13](#_Toc483260170)

[4.2 场景渲染实现 13](#_Toc483260171)

[4.3.1 天空盒 14](#_Toc483260172)

[4.3.2 地形 15](#_Toc483260173)

[4.3.3 3D模型 18](#_Toc483260174)

[4.3.3 光照和投影 19](#_Toc483260175)

[4.3.4 粒子系统 20](#_Toc483260176)

[4.3 游戏框架实现 26](#_Toc483260177)

[4.2.2 事件系统 26](#_Toc483260178)

[4.3.3 摇杆系统 28](#_Toc483260179)

[4.2.1 对象池 30](#_Toc483260180)

[4.3.3 日志系统 32](#_Toc483260181)

[4.4 游戏功能实现 34](#_Toc483260182)

[4.3.6 武器系统 34](#_Toc483260183)

[4.3.6 坦克系统 34](#_Toc483260184)

[4.3.6 寻路系统 34](#_Toc483260185)

[4.3.5 音效系统 34](#_Toc483260186)

[第五章 游戏展示与性能分析 34](#_Toc483260187)

[5.1 游戏展示 34](#_Toc483260188)

[5.2 性能分析 34](#_Toc483260189)

[5.3 优化 34](#_Toc483260190)

[第六章 项目总结与展望 34](#_Toc483260191)

[6.1 项目总结 34](#_Toc483260192)

[6.2 后续工作展望 35](#_Toc483260193)

[致 谢 36](#_Toc483260194)

[参考文献 37](#_Toc483260195)

[外文资料原文 38](#_Toc483260196)

[外文资料译文 39](#_Toc483260197)

第一章 绪 论

1.1 研究工作的背景与意义

随着计算机技术的发展，电子游戏的形式越来越丰富，用户对于电子游戏的娱乐性需求也越来越高，追求更加真实的感官体验成为电子游戏的一条发展趋势。作为虚拟现实领域的一项的综 合应用，3D 游戏已经在电脑、手机、机顶盒等多平台得到了普及，相对于传统的2D 游戏，3D 戏让用户感觉更融入，更加享受，但所依赖的计算机技术也更加复杂，计算机图形学、物理仿真等都成为 3D 游戏开发所需要的重点研究方向。我国在电子游戏领域的发展与领先国家还具有一 定的差距，为了振兴本国电子游戏行业，对这一项综合应用所涉及的技术进行整体的梳理是完全 有必要的。

国外的3D游戏起步在90年代初，他们最早完善的3D游戏引擎是美国ID公司于1993年推出的Doom引擎。国内首款自主研发的3D游戏始于2003年，到目前为止，国内在3D游戏已经有了很多的研究，但由于3D游戏开发的复杂性及经济效益的考虑，国内只有屈指可数的互联网公司拥有自主研发的3D游戏引擎，中小型公司还是使用国外较为成熟的游戏引擎和工具，如虚幻，Unity3D等。虽然近年来国内涌现一些优秀的3D网游诸如《剑侠情缘3》、《九阴真经》，但一直无法像产出世界级的3D游戏大作，大部分游戏公司还是以代理国外优秀产品为主，导致资金外流及本土文化缺失。随着移动硬件设备的日益强大，移动平台上的游戏也出现了惊人的增长，3D互动娱乐在手机领域的应用也逐渐增加。而移动领域存在诸多的操作系统，常用的有Android、Windows?Mobile、iOS?等，而在各个平台上的移植和实现对于开发商来说是不经济的，而目前国外优秀的成熟引擎已经成功解决这类需求，最为突出的要数Unity3D引擎，Unity3D中使用虚拟机技术使游戏编译结果独立于平台执行。国内对于跨平台引擎技术的研究还处于起步阶段，但熟练掌握和理解这些引擎的使用依然具有一定意义。

本次毕业设计所需要的软件为Unity3D。Unity是一款跨平台的游戏开发工具，从一开始就被设计成易于使用的产品。作为一个完全集成的专业级应用，Unity还包含了价值数百万美元的功能强大的游戏引擎。Unity3D是一个让你轻松创作的多平台的游戏开发工具，是一个全面整合的专业游戏引擎[1]

因此，利用Unity3D并且综合多重开发工具（PhotoshopCS3，Javascript等）进行的游戏开发，并利用移动平台发布会成为一个新趋势，在将来会得到更为广阔的运用。

1.2 手机游戏产业的国内外研究历史与现状

手游即手机上的游戏。随着移动互联网时代的到来，如今，手机除了满足大家发短信、通电话的基本需求外，玩游戏早已成为手机最重要的功能之一。手游早期为简单的文字游戏，随着手机像素、运行效率的提高，一些具备了简单彩色图形的手游开始面向用户，这类游戏大多使用 JAVA 语言编写实现，不论是画面，还是玩法都比上一个代提升了许多，然后是近几年的 app 应用，例如 N-Gage, N-Gage 是诺基亚在那个时代所研发的手游平台，但是由于移动网络无法很好的支持手游下载、支付渠道也稀少而复杂，N-Gage 始终没能大红大紫。Android 和 IOS 的兴起，手机在其本质工作几乎毫无进化的几年之间却在其余几乎每个领域发生了翻天覆地的变化，最简单并且最直观的一点表现就是性能参数了，手机性能参数的提高增加了应用的安装数量及功能，在这之中，手游获得的增益很大。 门槛低，投入小，回报大，是手游的主要优势，这也使很多游戏开发商纷纷转型手机游戏的开发。随着手机游戏市场的日渐升温，手游已经开始主导着整个移动互联网的发展，千万级用户，千万级收入已经不再是空话，手机行业的发展极大推动了手游的发展。近年来，手游更是有了质的飞跃，在手机行业跨越式的发展下，手机系统再也不是当年 NOKIA的塞班系统， iPhone 的诞生，及其开创的触屏潮流，不仅革新了用户操作手机的体验，而且也使手游脱离了物理键盘的局限，有了除“上下左右”之外的新的玩法。后来谷歌研发的安卓手机系统诞生，触屏智能更加便捷，得到了很快的普及，谷歌市场也出现了各种各样的安卓游戏，随后的几年安卓手游成疯狂的趋势增持。

这几年，游戏产业己经成为中国经济市场中增长速度最快的热点，是众所关注的焦点。游戏是文化创意产业里的一部分，创意经济是知识经济的核心内容，更是其经济的重要表现形式，没有创意就没有新经济。游戏设计属于文化艺术创意和商业产品的结合，对中国文化经济发展有重大意义。有利于提升中国游戏行业整体水平，并推动创意产业快速发展，提高经济产业发展水平，提升消费者的休闲体验感受。随着科技的发展，现在手机的功能也越来越多，越来越强大。而手机游戏也远远不是我们印象中的什么“俄罗斯方块”捕鱼达人“贪吃蛇”之类画面简陋，规则简单的游戏，进而发展到了可以和掌上游戏机媲美，具有很强的娱乐性和交互性的复杂形态。

由于多媒体技术以及智能手机技术的飞速发展，各式各样的手机游戏冲击着我们的视觉。其中，像一些简单二维平面游戏已经不能够满足广大游戏爱好，他们渴望一种新的三维游戏的诞生，三维游戏不仅在视觉上能给我们带来冲击，而且还给我们一种模拟现实的感觉，使游戏者感觉自己身临其境，而智能手机，平板电脑，点读设备等终端的普及和发展，为更多平台下的手机游戏提供了新的发展机遇，更多的三维体验游戏形式进入我们的生活世界，既融入了惊险刺激与游戏之中，又满足了我们的好奇心。以智能手机，平板电脑为体验终端的新型体验类游戏已经了手机游戏中的一个十分具有发展前景的模式

1.3 本文的主要贡献与创新

本次毕业设计的题目为《基于Unity3D引擎制作的3D坦克大战游戏》，旨在成功的设计出一 款可以运行在智能手机和各类电脑上的休闲、娱乐游戏，在毕业设计完成时，可以通过 多个平台，不同方式体验。游戏共包含若干关卡，从易到难。提供了一种放松、休闲的方式， 实现了对上班一族的工作压力缓解。实现娱乐、休闲、交流和取得虚拟成就的具有可持续性的个体性单人游戏。

1.4 本论文的结构安排

本文第二章介绍了Unity3Dy游戏引擎的基础，主要有它的一些基本的简介，一些特色等等，然后对Unity3D的常用技术做一个简单的介绍；

第三章介绍了本次毕设3D坦克游戏的整体设计包括游戏的目标，游戏的框架，游戏的界面，所需使用到的各种技术；

第四章主要是讲了游戏的功能实现，包括游戏地图实现，游戏场景中树木 坦克prefab的创建，坦克AI控制以及碰撞检测的实现等等。

第五章展示了最终的游戏成果，进行了性能分析，反省做的不足的地方。

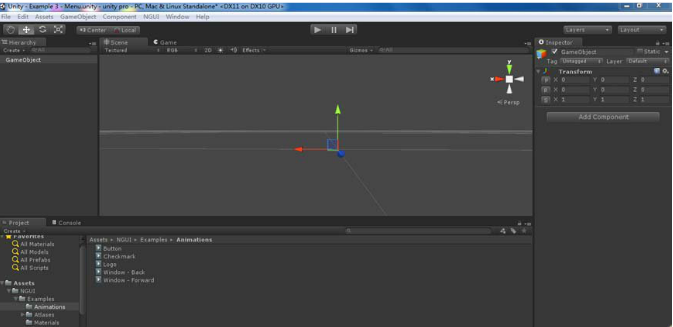
第六章对全文进行一个概括和总结。

第二章 Unity3D游戏引擎基础

Unity3D 游戏引擎 Unity3D是一款集三维视频游戏、建筑可视化、实时三维动画于一身的专业游戏引擎[1]，其多平台性表现在它可将已完成的游戏项目打包成 Windows、Mac、Wii、iPhone、Windows phone 8 和 Android 等多种平台所对应的文件格式。具有高度优化的图形渲染管道和内建的NVIDIA PhysX 物理引擎，使其能够较为真实地模拟 3D 物体的运动和碰撞。对 DirectX 和0penGL 拥有高度优化的图形渲染管道，使其能够比较真实、清晰的显示各种材质、贴图。它支持大部分主要的文件格式，并能和大部分相关应用程序协同工作，使用者不再需要使用其他转换格式的软件，用起来更加方便快捷。Unity3D 能够对编辑器、跨平台发布、地形编辑、着色器、脚本、网络、物理、版本控制等进行全面整合，优化使用界面，降低操作重复性，提高操作效率。这些都使 Unity3D 成为独一无二的游戏引擎。关于 Unity3D 的使用，需要了解以下几个方面的内容

2.1 主界面视图

主界面视图 Unity3D 界面上方为菜单栏，所有功能都可以在菜单栏中找到。菜单栏下方为具有各种功能的窗口，负责游戏项目的大部分操作，可将部分用得相对频繁的窗口单独拖出来放在外面，方便使用。Hierarchy 窗口、Project 窗口、Console 窗口、inspector 窗口、Scene窗口和 Game 窗口为主要的窗口，具体如图 2-1 所示：



一个完整的 lJnity3d 程序是由若干个场景（Scene)组合起来的，每个场景中又 包含有许多游戏对象（GameObject），每个对象可以具有若干组件（Component），其 中的继承自 MonoBehavior 的脚本组件进行初始化、更新等操作，而我们在场景所看 到的内容是由摄像头（Camera)来呈现并控制的

2.1.1 Hierarchy 窗口

　Hierarchy 窗口主要用于放置 Gameobject，可以对 Gameobject 进行选中、重命名、拖动分层、添加组件等操作，并且 Hierarchy 的所有内容都会在 scence 窗口中显示出来。

2.1.2 Project 窗口

Project 窗口为项目工程的文件入口，游戏项目的所有文件都可以在 project 中找到，进行资源替换、查找都要在这个窗口下操作。

2.1.3 Console 窗口

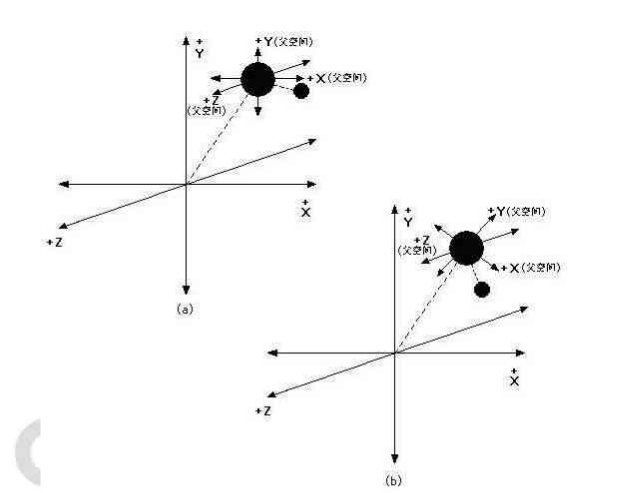
Console 窗口为一个指令提示窗口，项目运行的各种情况都会在这边以 CSharp 语言的形式显示出来，包括错误、警告、运行成功等，通过查看此窗口，可以轻松找出报错的根源。

2.1.4 Inspector 窗口

Inspector 窗口为 Gameobject 的编辑窗口，包括 Gameobject 的坐标、大小、附加控件及添加脚本等，一个 Gameobject 对应一个 inspector，默认显示被选中的 Gameobject的 inspector 窗口。

2.1.5 Scene 窗口

Scene 窗口真实地显示 Hierarchy 窗口中 Gameobject 的样式和 Tranform 位置，可对Gameobject 进行选中、拖动、旋转等操作。场景是Unity3D程序的基本组成单位，任何一个Unity3D程序都是由若干场景组合而成，程序通过脚本在这些场景之间跳转。场景通过场景图的形式组织，场景图实际上是一种树形结构，其中每个节点就是一个游戏对象。对象与对象之间存在父子关系，即当父级对象移动、旋转、缩放时，子级对象也跟着一起变换



2.1.6 Game 窗口

Game 窗口为运行项目时所呈现出来的真实游戏效果，为映入屏幕的画面。

2.2 游戏组件

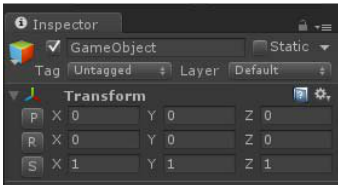
组件可以是网格（Mesh）、光源（Light）、摄像机（Camera）、 粒子系统（Particle）、物理碰撞体（Collider）、布料（Cloth）、连接体（Joint）、 声音（Audio）、动画（Animation）以及最重要的脚本（Script）。Unity3D 的设计是 以面向对象理论为基础，Unity3D 支持 C#、JavaScript Boo三种脚本语言，如果 对象继承自 MonoBehavior 脚本对象，生命周期将交给 Unity3D 来管理

2.2.1 Gameobject(对象)

Gameobject就是一个Component容器，Unity3D 中的任何组件、目标、模型都必须用 Gameobject来承载，它可以是一张图片、一段单一的文字、或是一个简单的模型，取决于你给它添加什么内容。从 Unity3D 菜单栏的“Gameobject”中可以轻松创建一个空的 Gameobject，快捷键 ctrl+shift+N 比较常用。如在一个空的 Gameobject 中添加一个 Directional light(单向灯光控件)，则此 Gameobject 就是个灯光，可在 inspector 窗口编辑。Gameobject 支持重命名功能，使开发者能一眼就看出 Gameobject 内的内容，提升工作效率，Gameobject 还可以嵌套子 Gameobject，方便进行资源的整合。

2.2.2 Tranform（坐标）

Tranform（坐标）的理解 如下图

inpector 窗口中的 Tranfrom 所示，Gameobject 的坐标包含“P”“R”“S”三个坐标，“P”即 position，为 Gameobject 的位置坐标，用于控制目标的方位信息。“R”即 rotate,为 Gameobject 的旋转坐标，调节 P 坐标将使目标作出相应转向。例如将上面提到的单向灯光的 Tranform“R”中的 X 坐标从 0 改成 90，则此灯光从横向灯变成纵向灯。“S”即 scale，为 Gameobject 的大小坐标，控制目标的大小尺寸，x、y、z 对应 Gameobject 的长宽高。一个 Gameobject 至少包含一个 tranform，有且仅有一个 tranform。

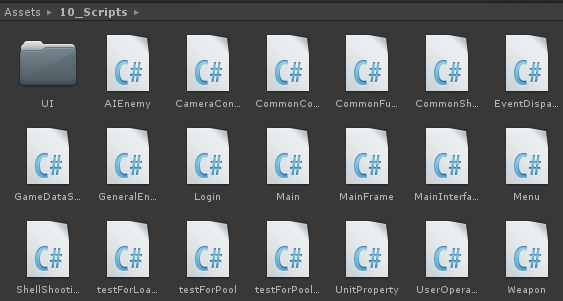
2.2.3 Camera(镜头)

正如电影导演手中的摄像机将故事呈现给观众一样，Unity3D 中的 Camera 用于将游戏画面呈现给玩家，即 Game 窗口中的内容。一个视图中至少应该放置一个镜头，否则呈现出来的画面只会是一片漆黑。镜头的放置位置也决定了游戏体验，镜头应尽量放置在能看清整个场景的位置，距离适中，太近的镜头视距太短看不到远方的路障，太远的镜头看不清前方的物体，影响游戏体验。Unity3D 中新建的工程都有提供一个 main camera 镜头。另外，可以给镜头添加一些功能，如 animation 动画，在镜头上添加具有短位移的 animation动画可以使其抖动。

2.3 脚本系统

unity3D的脚本语言在Unit3D游戏开发中占据了主角地位。它提供了三种脚本语言的支持：Javascript、C#、Boo。脚本语言的动态特性让我们可以方便的通过名称、层次结构、tags 等方式访问所有的对象。更大的好处就是脚本语言的跨平台性，绝大部分平台相关的代码都放到了引擎的内部，而游戏内容相关的代码都可以跨

本次毕设用的是 C#语言，如需要让某些 Gameobject 实现特定功能，可通过编写 C#脚本，保存起来，并将脚本拖动到 Gameobject 所对应的 inspector 窗口，这样该 Gameobject 就具有脚本所描述的功能。



2-4 Unity3D 中的C#脚本文件

2.3.1 C#

C#读作C Sharp。它是由Microsoft公司的开发团队设计的。C#是一种安全的、稳定的、简单的、优雅的，由C和C++衍生出来的面向对象的编程语言,它继承了 C 和 C++的强大功能，并简化了 C 和 C++的一些复杂特性，使其成为当前.NET 开发的首选语言。C#使得C++程序员可以高效的开发程序，且因可调用由 C/C++ 编写的本机原生函数，因此绝不损失C/C++原有的强大的功能。因为这种继承关系，C#与C/C++具有极大的相似性，熟悉类似语言的开发者可以很快的转向C#。因为它强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程，使其成为系统开发和应用开发的最佳使用语言。

C#是运行于.NET Framework之上的高级程序设计语言。并定于在微软职业开发者论坛(PDC)上登台亮相。C#是微软公司研究员Anders Hejlsberg的最新成果。C#看起来与Java有着惊人的相似；它包括了诸如单一继承、接口、与Java几乎同样的语法和编译成中间代码再运行的过程。但是C#与Java有着明显的不同，它借鉴了Delphi的一个特点，与COM（组件对象模型）是直接集成的，而且它是微软公司 .NET windows网络框架的主角。

2.3.2 Javascript

Javascript是一种基于对象和事件驱动并具有相对安全性的客户端脚本语言。同时也是一种广泛用于客户端Web开发的脚本语言，常用来给HTML网页添加动态功能，比如响应用户的各种操作。，是一种动态、弱类型、基于原型的语言，内置支

JavaScript是一种属于网络的脚本语言,已经被广泛用于Web应用开发,常用来为网页添加各式各样的动态功能,为用户提供更流畅美观的浏览效果。通常JavaScript脚本是通过嵌入在HTML中来实现自身的功能的。

（1）是一种解释性脚本语言（代码不进行预编译）。

（2）主要用来向HTML（标准通用标记语言下的一个应用）页面添加交互行为。

（3）可以直接嵌入HTML页面，但写成单独的js文件有利于结构和行为的分离。

（4）跨平台特性，在绝大多数浏览器的支持下，可以在多种平台下运行（如Windows、Linux、Mac、Android、iOS等）。

Javascript脚本语言同其他语言一样，有它自身的基本数据类型，表达式和算术运算符及程序的基本程序框架。Javascript提供了四种基本的数据类型和两种特殊数据类型用来处理数据和文字。而变量提供存放信息的地方，表达式则可以完成较复杂的信息处理，持类。

Javascript是Sun公司的注册商标。不同于服务器端脚本语言，例如PHP与ASP，Javascript是客户端脚本语言，也就是说Javascript是在用户的浏览器上运行，不需要服务器的支持而可以独立运行。

2.4 物理引擎

2.4.1 物理引擎简介

在游戏这项综合应用中中，除了华丽的画面渲染外，作为表现虚拟世界系统内在物理规律的计算是必不可少的。当被模拟的刚体的运动比较简单，可以在一定程度上通过编程或编写脚本来实现，如简单的加速和减速牛顿物理运动。但当模拟比较复杂的物体的碰撞、滚动、滑动或者弹跳的时候，通过编写底层算法会造成工程延误，这显然不符合软件工程的思想。成熟的物理引擎能够允许更复杂的物理模拟，像球形关节、轮子、气缸或者铰链，有些也支持非刚性体的物理属性，比如流体和布料。 物理引擎接受的输入是由外界调用模块传递过来的场景信息以及场景中物体的位置

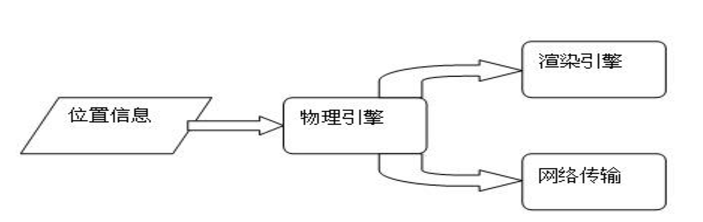


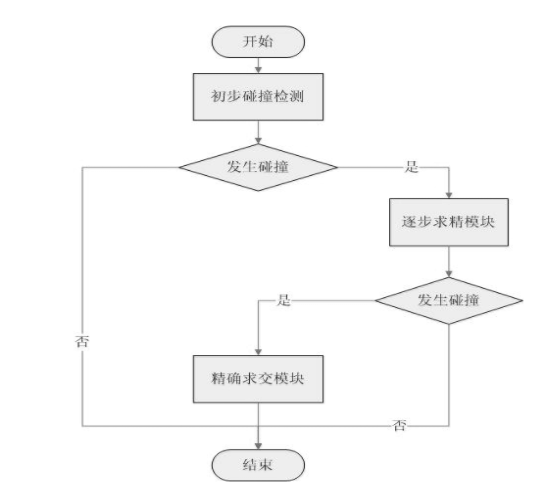
图2-4 物理引擎功能示意图

2.4.2 物理引擎基本架构

物理引擎一般分为两个重要部分，即碰撞检测和物理学世界。如图 2-4-2.1 所示。

图 2-4-2.1

碰撞检测模块是物理引擎的核心模块，建立在物理学世界的基础上的。其由初步碰撞检测、精确碰撞检测和精确求交三个基本模块组成，各模块的执行顺序。如图2-4-2.2所示。



在初始化阶段，碰撞检测模块给虚拟世界每个物体建立包围盒，比较常用的包围盒算法有AABB(Axis-aligned bounding box)、包围球、胶囊体等。逐步求精阶段将整个虚拟世界进行空间划分，主要用到数据结构有八叉树、k-d树和BSP树等。最后再精确求交模块中使用凸体算法和射线检测算法得到精确的碰撞信息。

第三章 3Ｄ坦克游戏设计

3.1 游戏基本介绍

开始游戏后，随机生成若干辆敌方坦克，可以使用虚拟摇杆控制玩家坦克在场景中移动，点击炮弹图标并向前发射子弹，子弹碰撞到敌方坦克则会爆炸并消灭敌方坦克，敌方坦克会向玩家坦克移动，并在攻击距离内向玩家发射炮弹。玩家被敌方坦克命中后扣除血量。血量降到0则游戏结束。

3.2 需求分析

3.3 技术路线

3.4 游戏框架结构设计

第四章　3Ｄ坦克游戏实现

4.1 图形界面实现

在本次游戏中采用UGUI来实现游戏的用户界面，UGUI即UnityGUI是官方的UI实现方式，后来因为效率和可视化程度低，逐渐被NGUI和EzGUI等第三方插件替代，但Unity4.6之后，又推出了新的UGUI系统，并在不断完善。

所有的UI元素必须是画布canvas的子元素，画布铺满整个屏幕空间，UI元素至于屏幕的最上层，简单来说看看到的就是一个2D的样子。

4.1.1 开始界面

进入游戏后首先进入开始界面（图4-1-1 所示），



开始界面有两个按钮，“开始游戏”和“退出游戏”按钮，背景图为坦克大战图， 开始界面相对其他界面来说较为简单，新建两个UIButton，将button 的text的文字改成“开始游戏” 和 “退出游戏”，调整 Tranform大小。功能实现点击“进入游戏”按钮，进入加载游戏界面，实现代码如下：

public void onBtnEvent(GameObject btn, UserOperate.btnState btnState)

{

if (btnState == UserOperate.btnState.btnStateUp && btn.name == "begin")

{

SceneManager.LoadScene("6\_Scenes/MainScene");

}

else if (btnState == UserOperate.btnState.btnStateUp && btn.name == "quit")

{

Application.Quit();

}

}

4.1.2 加载界面

玩很多游戏，在加载切换场景的时候往往需要载入很多的资源，因此需要加载比较长的时间，为了有更好的玩家交互，让玩家可以在等待加载的同时也不会觉得无聊，就需要一个好的缓冲过程了。这时候我们用到的就是异步加载。在加载的时候我们可以通过各种各样的方式给玩家一种“假象”，让玩家可以知道游戏的加载进度。本次游戏中实现了一个异步加载场景的方法。

加载界面需新建一个UI->Slider滑动条来表示加载进度，在场景中挂上脚本LoadManager.cs来控制加载，利用核心代码:  
在上述代码中,游戏开始时在start函数中 启动协程 进行异步加载资源，在Update函数中每帧更新加载进度条。

public class LoadManager : MonoBehaviour

{

public Slider m\_slider;

int progress = 0; //加载进度//

AsyncOperation async; //异步对象//

private const float WAIT\_TIME = 1.5f; //间隔时间1.5秒//

private float \_lasttime = 0.1f;

void Start()

{

StartCoroutine(loadScene());

}

void Update()

{

progress = (int)(async.progress \* 100);

m\_slider.value = async.progress;

if (Time.time - \_lasttime >= WAIT\_TIME)

{

\_lasttime = Time.time;

}

}

IEnumerator loadScene()

{

async = SceneManager.LoadSceneAsync(Global.loadSceneName);

yield return async;

}

}

加载界面效果图如下:



4.1.3 游戏界面

此界面为游戏战斗界面，UI元素较多，主要负责控制坦克、发射炮弹、显示玩家信息等，

从左到右依次为：玩家头像、玩家得分、玩家血量条、虚拟摇杆(控制方向位移)、视角切换(第三人称和第一人称视角)、跟踪导弹按钮、普通导弹按钮、炮杆角度和旋转摇杆:



4.2 场景渲染实现

考虑到游戏的需求， 整个场景主要包括几类渲染对象， 如天空盒、 水面、 地形、外部导入的 3D 模型、 投影、 粒子系统等，

图4-3游戏场景中渲染对象

下面主要介绍一些重要渲染模块的具体实现：

4.3.1 天空盒

所谓天空盒， 就是用一个作为远景贴图的立方体。 在立方体的前后左右上五个面贴上纹理（ 需要时底面也可以帖）。 在简单的天空盒实现中， 只需要填写好六个面的24 个顶点， 并设置立方体随着视点的移动而移动， 这样就会给玩家带来天空是在极远处的错觉。 但在较为复杂的应用中， 还需要使用天空盒的纹理来生成 Cube Map，并用之来做水面倒影、 云影等特效的贴图。



图 4-2 天空盒由六个面映射而成

在 Unity3D 中对天空盒有很好的支持， 开发人员 只需要几个简单的步骤：

1. 创建一个材质；

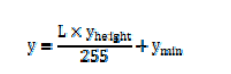
2. 并设置 Shader 为 Unity3D 自 带的 RenderFX-&gt;Skybox；

3. 为天空的各个面选择贴图；

 4. 选中场景中的摄像机对象，然后点选菜单栏中的“Component”——>“Rendering”——>“Skybox”为摄像机添加一个天空盒。添加成功之后，您就能在摄像机的属性面板中找到刚才添加的天空盒了，在 Render Setting 中设置天空材质为前面创建的材质。 图 4-3 为实现的效果图：

图 4-3 实现的天空效果

4.3.2 地形

 在 Unity3D 中通过高度图来实现地形， 高度图实际上是一张矩形的灰度图， 在灰度模式中， 颜色通过 0~255 来表示， 0 代表黑色， 255 代表白色， 像素越亮代表地势越高， 255 的像素代表此次顶点最高， 反之亦然， 所以像素灰度值和顶点的高度可以用公式 4-3 表示，

其中 L 为像素的灰度值，、yheight ymin分别为 Unity3D 中定义的地形高度和地形最低点， 求得网格顶点的高度 y。 公式 4-3 程序在需要渲染地形时通过读取这张图片， 并按照灰度生成顶点高度不一的矩形网格。 这样大大减少了 游戏模型体积。 Unity3D 中的地形模块不但可以导入高度图， 还可以用笔刷为绘制地形纹理、 树、草和岩石， 所见即所得， 大大减少了 开发人员 的工作量。 图 4-3-2 为小岛的效果图：

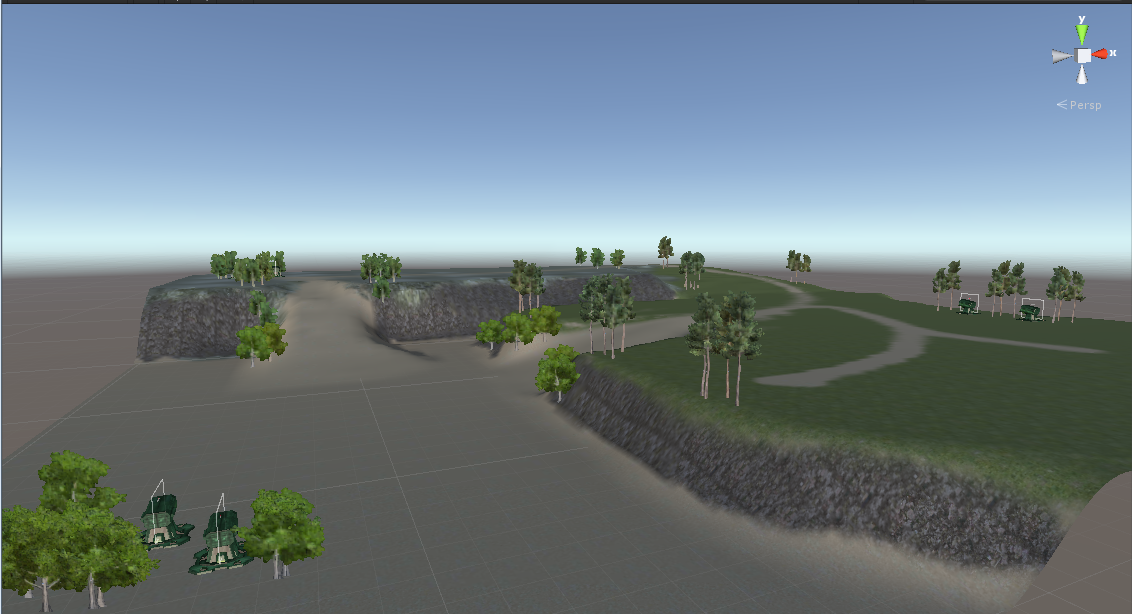
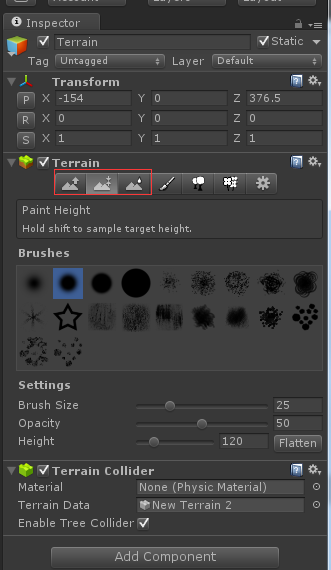


图 4-3-2 小岛地形效果图

地形创建步骤如下:



地形绘制：首先创建一个地形Terrain->Create Terrain；可以在Terrain菜单下设置地形大小、分辨率、导入高度等。

点击“Game Object”——>“Create Other” ——>“Camera”，在场景中创建一个摄像机，这时您就可以在Game【动画面板】中观看到摄像机所观察到的景象了。

在地形的属性（Inspector）这一层中，可以看到很多绘制地形的工具。第一个工具是抬高工具，可以用来绘制山脉或者是一些河道的效果。第二个工具是绘制高度，可以用来踏平一个地形。第三个工具是柔滑高度，就是使地面变得比较光滑。

地形贴图：选择第四个工具之后，点击Edit Texture来添加纹理。在弹出的对话框下选择地形纹理后，点击Add按钮即可。添加之后选中草地的材质，在地形中进行绘制，绘制前还是可以设置笔刷的大小、强度和融合度。

植物的种植：第五个工具是用来在地形上种植树木的。选中第五个工具，并添加要种植的树的模型。选中树，并对其参数值进行设定后，就可以在场景中进行种植树了。种植树时，直接点击左键即可。可以根据滚动条设置笔刷的大小、树的密度、树的颜色的随机性、树的高度、高度的随机性、树的宽的、宽的随机性。

草地绘制:第六个工具用来绘制细节。选中第六个工具用来绘制草地。添加一个草地的材质。在弹出的对话框中选择相应的草地，同时可以根据自己的需要修改下面的参数值。也可以在绘制完后再进行调整。点击Add按钮即可。

细节的实现：第六个工具是有关地形的细节的设置。具体的参数有网格优化的容差，Map的距离，光晕图的设置等。

4.3.3 3D模型

在3DS Max中，点击File【文件】——>export【导出】，将模型文件导出为FBX文件，保存到 “Turret”文件夹中的“Assets”【资源】文件夹下面。

打开3dMax软件制作一个坦克模型，分为上下两部分，导出格式为fbx，再将制作好的模型保存在“Turret”文件夹中的“Assets”文件夹下边。来到Unity环境中，找一个合适的位置，将我们的炮塔拖拽上去。在这里可以在它的属性面板中找到“Scale Factor ”，来改变它的大小。 将其制作成prefab 保存为资源文件，供程序使用

4.3.3 光照和投影

在一般 3D 应用程序中，光照和投影是各自独立的计算过程，光照既可以使用固定管线、 也可以再 Shader 自 定光照模型实现。 投影则是一个双过程的技术：

1. 首先， 场景以光源的位置为视点被渲染。 每个渲染图像的像素的深度值被记录在一个“深度纹理” 中， 被称为阴影贴图。

2. 然后， 场景从眼睛的位置渲染， 但是用标准的投影纹理把阴影贴图从灯的位置通过投影矩阵运算混合到场景中。 在每个像素， 深度采样值 （ 从被投影的深度纹理中）与片段到灯的距离进行比较。 如果后者小， 这意味着这个片段不是阴影， 不需要和投影纹理进行混合。

图 4-3-3 举例说明了 阴影的深度比较。 在图的左边， 正要被着色的点 P 在阴影中，因为这个点的深度值（ ZB） 比记录的阴影贴图的深度值（ ZA） 大。 相反， 在图的右边显示了 点 P 的深度值与在阴影贴图中记录的值相同的情况。 这意味着在在 P 和灯源之间没有任何物体， 因此 P 不再阴影中。

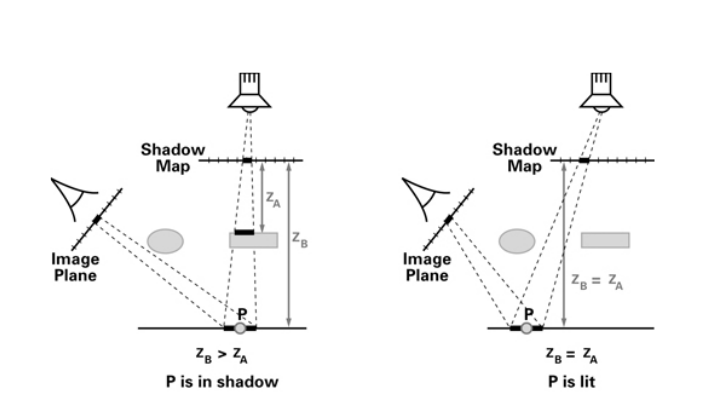


图 4-3-3 投影的深度比较

对于静态物体的投影渲染的第一过程计算， 可以在游戏开发过程中进行。 Unity内置的光照贴图烘培工具是 Illuminate Labs 的 Beast， 其主要原理是对场景中所有静态物体的光照和投影进行计算， 并利用特殊的. exr 光照贴图格式用以记录光照和投影两种信息， 在游戏运行时用光照贴图对模型原有材质进行混合渲染， 即可得到正确的光照和投影效果。 这样大大减少了 游戏运行过程中计算量， 提高游戏效率。 如图4-10 所示， 进行一些参数设置（ 实际上是设置光照模型和投影过程二的阴影纹理），选择“Bake Scene” 即可为场景生成光照贴图。



图 4-10 光照贴图烘焙窗口

创建光照：点击“GameObject”——>“Create Other”——>“Directional Light”创建一个太阳光。

太阳光和自身的位置没有多大的关系，只与自身的角度有关，这和我们平时生活中的太阳光是一致的。所以为了让场景变得更亮，我们就需要用到场景调整工具中的旋转按钮来对太阳光进行旋转。当然您也可以在属性面板（Rotation）中对它进行调节。根据我们生活的常识，当太阳光90°直射地面的时候，光线最强。

4.3.4 粒子系统

粒子基本上是在三维空间中渲染的二维图像。它们主要用于诸如烟、火、水滴、或树叶等效果。一个粒子系统是由三个独立部分组成：粒子发射器、粒子动画器、和粒子渲染器。如果你想要一个静态粒子系统，你可以用一个粒子发射器和渲染器来完成。粒子动画器将在不同的方向移动粒子和改变粒子的颜色。你还可以通过脚本使用粒子系统中的每一个粒子，因此你可以创建自己独特的（粒子）行为。本次研究利用Unity3D的粒子系统实现坦克 及炮弹的爆炸特效，下面简单说明下例子面板中一些主要模块:

1. 粒子系统初始化模块：



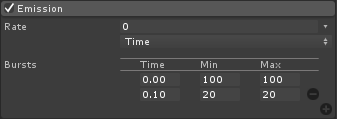
图4-3-4 粒子系统初始化模块

表4-3-4 粒子系统模块参数说明:

|  |  |
| --- | --- |
| 参数 | 作用 |
| 持续时间 (Duration) | 粒子系统 (Particle System) 发射粒子的持续时间。 |
| 循环 (Looping) | 粒子系统 (Particle System) 是否循环 |
| 预热 (Prewarm) | 只可预热循环系统，这意味着，粒子系统 (Particle System) 在游戏一开始时就发射粒子，就像已发射了一个周期的粒子。 |
| 初始延迟 (Start Delay) | 粒子系统(Particle System) 发射粒子之前等待的延迟，以秒为单位。请注意，预热的循环系统不能使用初始延迟。 |
| 初始生命 (Start Lifetime) | 粒子存活时间，以秒为单位 |
| 初始速度 (Start Speed) | 粒子发射时的速度 |
| 初始大小 (Start Size) | 粒子发射时的大小 |
| 初始旋转 (Start Rotation) | 粒子发射时的旋转 |
| 初始颜色 (Start Color) | 粒子发射时的颜色 |
| 重力修改器 (Gravity Modifier) | 粒子在存活期间受到的重力影响 |
| 继承速度 (Inherit Velocity) | 控制粒子速率的因素应继承粒子系统 (Particle System) 的转换（对于移动中的粒子系统） |
| 模拟空间 (Simulation Space) | 模拟本地坐标系或世界坐标系中的粒子系统 (Particle System) |
| 唤醒时播放 (Play On Awake) | 如果启用该项，粒子系统 (Particle System) 会在其创建时自动开始播放 |
| 最大粒子数 (Max Particles) | 粒子系统 (Particle System) 可发射的最大粒子数量 |

（2） 发射模块:

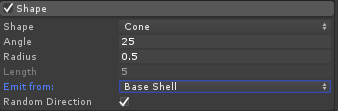
控制粒子发射的速率，允许在某个时刻生成大量粒子（粒子系统 (Particle System) 存续期间）。在爆炸时非常有用，那时需要一次创建大量粒子。



|  |  |
| --- | --- |
| 参数 | 作用 |
| 速率 (Rate) | 单位时间 (Time)（每秒）或单位距离 (Distance)（每米）发射的粒子数量 |
| 突发 (Bursts)（仅限时间 (Time) 选项下） | 添加粒子系统 (Particle System) 存续期间发生的粒子爆发 |
| 时间和粒子数量 (Time and Number of Particles) | 指定发射规定数量粒子的时间（在存续期内，以秒为单位）。用“+”和“-”调节爆发数量 |

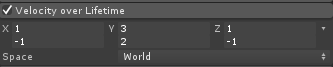
(3) 形状模块

定义发射器的形状：球体 (Sphere)、半球体 (Hemishpere)、圆锥 (Cone)、立方体 (Box) 和网格 (Mesh)。可沿着表面法线或随机方向施加初始力。选择不同的形状，下面各个参数不一样：



|  |  |
| --- | --- |
| **参数** | **作用** |
| **球体((Sphere)** |  |
| 半径 (Radius) | 球体的半径。（也可以在场景视图 (Scene View) 中用手柄操作）。 |
| 从外壳发射 (Emit from Shell) | 从球体外壳发射。如果禁用此项，粒子将从球体内部发射。 |
| 随机方向 (Random Direction) | 粒子是在随机方向还是沿着球体表面法线方向发射？ |
| **半球体 (Hemisphere)** |  |
| 半径 (Radius) | 半球体的半径。（也可以在场景视图 (Scene View) 中用手柄操作）。 |
| 从外壳发射 (Emit from Shell) | 从半球体外壳发射。如果禁用此项，粒子将从半球体内部发射。 |
| 随机方向 (Random Direction) | 粒子是在随机方向还是沿着半球体表面法线方向发射？ |
| **圆锥 (Cone)** |  |
| 角度 (Angle) | 圆锥的角度。如果角度为 0，粒子将在一个方向发射。（也可以在场景视图 (Scene View) 中用手柄操作）。 |
| 半径 (Radius) | 发射点的半径。如果值接近零，则将从一点发射。如果值超过 0，将创建一个帽子状的圆锥，粒子从一个圆盘而非一个点发射。 |
| 长度 (Length) | 发射量的长度。仅从内部 (Volume) 或内部外壳 (Volume Shell) 发出时可用。（也可以在场景视图 (Scene View) 中用手柄操作） |
| 发射位置 (Emit From) | 确定从哪里发射出。可能的值有底部 (Base)、底部外壳 (Base Shell)、内部 (Volume) 和内部外壳 (Volume Shell)。 |
| 随机方向 (Random Direction) | 粒子是在随机方向还是沿着圆锥方向发射？ |
| **立方体 (Box)** |  |
| 立方体 X 轴 (Box X) | 立方体 X 轴的缩放 |
| 立方体 Y 轴 (Box Y) | 立方体 Y 轴的缩放。 |
| 立方体 Z 轴 (Box Z) | 立方体 Z 轴的缩放。（也可以在场景视图 (Scene View) 中用手柄操作）。 |
| 随机方向 (Random Direction) | 粒子是在随机方向还是沿着立方体 Z 轴方向发射？ |
| **网格 (Mesh)** |  |
| 类型 (Type) | 粒子可从顶点 (Vertex)、边 (Edge) 或面 (Triangle) 发射。 |
| 网格 (Mesh) | 选择网格 (Mesh) ? 作为发射形状。 |
| 随机方向 (Random Direction) | 粒子是在随机方向还是沿着网格表面方向发射？ |

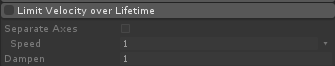
（4）存活时间的速度模块:

 直接动画化粒子的速率。主要用于具有复杂物理特性的粒子，不过只演示简单的视觉行为（例如飘荡的烟雾和温度降低），与物理世界的互动很少。

|  |  |
| --- | --- |
| 参数 | 作用 |
| XYZ | 使用曲线常量值或曲线间的随机值来控制粒子的运动。 |
| 空间 (Space) | 本地 (Local)/ 世界 (World)：速度值是本地坐标系还是世界坐标系中的值？ |

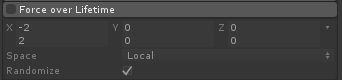
（5）存活时间的限制速度模块:

基本上用于模拟阻力。如果超过某些阈值，就会抑制或固定速率。可以按每个轴或每个向量长度配置。



|  |  |
| --- | --- |
| 参数 | 作用 |
| 分离轴 (Separate Axis) | 用于设置每个轴控制。 |
| 速度 (Speed) | 指定量级为常数或由限制所有轴速率的曲线指定量级。 |
| 阻尼 (Dampen) | (0-1) 值，控制应减慢的超过速率的幅度。例如，值为 0.5，则将超过的速率减慢 50%。 |

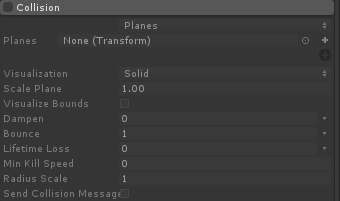
（6）存活时间的受力模块



|  |  |
| --- | --- |
| 参数 | 作用 |
| XYZ | 使用曲线常量值或曲线间的随机值来控制作用于粒子上的力 |
| 空间 (Space) | 本地 (Local)/世界 (World)：速度值是本地坐标系还是世界坐标系中的值 |
| 随机化 (Randomize) | 随机化每帧作用于粒子上的力。 |

（7）**碰撞模块**

为该粒子系统 (Particle System) 中的粒子设置碰撞。现在支持世界碰撞和平面碰撞。平面碰撞对简单碰撞检测而言非常有效。通过引用场景中的现有变换或为此创建新的空游戏对象 (GameObject) 来设置平面。平面碰撞的另一个优势是具有碰撞平面的粒子系统可设为预设。世界碰撞采用光线投射，所以必须小心使用，以确保良好性能。然而，在近似碰撞可接受的情况下，低 (Low) 或中等 (Medium) 质量的世界碰撞非常有效。



任何碰撞模块 (Collision Module) 的常见属性

|  |  |
| --- | --- |
| 参数 | 作用 |
| 平面/世界 (Planes/World) | 指定碰撞类型：平面 (Planes) 指平面碰撞，世界 (World) 指世界碰撞。 |
| 阻尼 (Dampen) | (0-1) 粒子碰撞时，其将保持这一速度比例。除非设置为 1.0，否则粒子在碰撞后变慢 |
| 反弹 (Bounce) | (0-1) 粒子碰撞时，其会保持这一速度分量比例，该分量与碰撞平面垂直。 |
| 生命减弱 (Lifetime Loss) | (0-1) 每次碰撞后初始生命周期 (Start Lifetime) 减弱比例。生命周期达到 0 后，粒子死亡。例如，如果粒子应该在第一次碰撞时死亡，则将该属性设为 1.0。 |
| 最小杀死速度 (Min Kill Speed) | 粒子在被杀死前的最小速度。 |

只在平面模式 (Planes Mode) 中可用的属性

|  |  |
| --- | --- |
| 参数 | 作用 |
| 平面 (Planes) | 平面通过向变换分配引用来定义。该变换可为场景中的任何变换，且可动画化。可以使用多平面。注意：Y 轴用作平面法线。 |
| 可视化 (Visualization) | 仅用于将平面可视化：网格 (Grid) 或实体 (Solid)。 |
| 反弹 (Bounce) | (0-1) 粒子碰撞时，其会保持这一速度分量比例，该分量与碰撞平面垂直。 |
| 网格 (Grid) | 渲染为小图示，能快速指出在世界中的位置和方向。 |
| 实体 (Solid) | 在场景中渲染平面，用于精确定位平面。 |
| 缩放平面 (Scale Plane) | 调整可视化平面大小。 |
| 粒子半径 (Particle Radius) | 因碰撞目的假设的粒子半径。（因此，粒子视为球体。） |

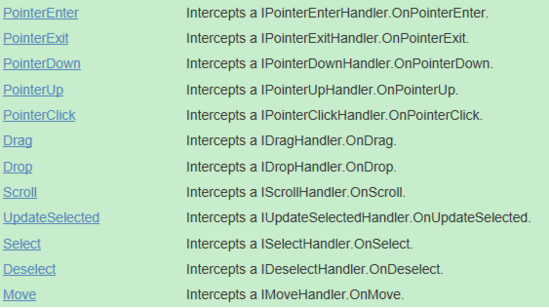
只在世界模式 (World Mode) 中可用的属性

|  |  |
| --- | --- |
| 参数 | 作用 |
| 碰撞对象 (Collides With) | 世界碰撞的质量。 |
| 高质量 (High) | 每帧的所有粒子进行场景光线投射。注意：这是 CPU 密集型应用，只能使用 1000 个（场景宽度）或更少的同步粒子。 |
| 中等质量 (Medium) | 在每帧中，粒子系统接收一部分全局设置粒子光线投射计划 (Particle Raycast Budget)（请参阅粒子光线投射计划）。粒子以循环方式更新，其中在给定帧中不接收光线投射的粒子将查找和使用高速缓存中存储的较旧碰撞物。注意：这类碰撞属于近似碰撞，会遗漏一些粒子，特别是在角落处。 |
| 低质量 (Low) | 除了粒子系统每四帧获得一部分粒子光线投射计划 (Particle Raycast Budget)之外，其他与中等质量 (Medium) 相同。 |
| 体素大小 (Voxel Size) | 用于缓存中等 (Medium) 和低 (Low) 质量设置中所用交集的体素密度。体素大小以场景单位为单位给出。通常会使用 0.5 ? 1.0（假设使用公制单位）。 |

4.3 游戏框架实现

4.2.2 事件系统

Unity3D的uGUI系统的将UI可能触发的事件分为12个类型，即EventTriggerType枚举的12个值。如下图所示：



在本游戏中，统一用事件注册回调 来管理UI的响应事件，实现一个事件调度类:

public class EventDispatcher : MonoBehaviour

{

public delegate void ui\_event\_btn(GameObject btn, UserOperate.btnState btnState);

public delegate void ui\_event\_camera();

public delegate void ui\_event\_drop(GameObject dragObj, UserOperate.dragState dragState, PointerEventData eventData);

public delegate void ui\_event\_key();

public delegate void ui\_event\_mouse();

public static event ui\_event\_btn btnEvent;

public static void onBtnEvent(GameObject btn, UserOperate.btnState btnState)

{

btnEvent(btn, btnState);

}

public static event ui\_event\_drop dropEvent;

public static void onDropEvent(GameObject dragObj, UserOperate.dragState dragState,PointerEventData eventData)

{

dropEvent(dragObj, dragState,eventData);

}

void Start()

{

//进行初始化，避免空监听时报错

btnEvent += (o, s) => {};

dropEvent += (o, s,e) => {};

}

}

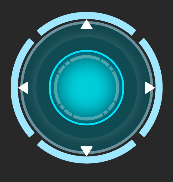
在每个需要UI响应的组件上挂在一个 UserOperate.cs脚本，此脚本中 重写了Unity3D自带的各个UI响应函数，注册到EventDispatcher中。在使用时 只需 EventDispatcher.btnEvent += onBtnEvent 就完成了按钮事件的处理函数的添加，然后实现具体的按钮处理函数 onBtnEvent。

4.3.3 摇杆系统

**摇杆界面实现：**

虚拟摇杆有一张固定的2D贴图(背景层)和一张可拖动的2D贴图(控制层)构成，默认情况下控制层在背景层的中心，我们称这个位置为初始位置，当移动控制层后移动层的位置会发生变化，此时控制层的当前位置和初始位置两点间可以计算出一个2D向量，通过这个向量我们就可以判断虚拟摇杆的移动方向。在经典的八方向摇杆导航中摇杆中可移动方向被分成了上、左上、右上、下、左下、右下、左、右共8个方向。我们知道根据三角函数可以非常容易地计算出这个2D向量的角度并由此判定摇杆是在向着这8个方向中的哪一个方向移动。在本次游戏中，我们不需要考虑这8个方向，因为我们可以向任何一个方向进行移动。

首先在场景中创建两个Image组件和一个空的游戏体，然后将这两个Image组件拖拽到这个空的游戏体下使它们称为其子节点。这里需要注意的是这两个Image的层级关系。制作完成后如图:



**摇杆控制实现:**

现在我们来编写脚本，这个脚本将被添加到控制层物体上，脚本JoyStickUGUI.cs中

void Update()

{

//当虚拟摇杆移动到最大半径时摇杆无法拖动

//为了确保被控制物体可以继续移动

//在这里手动触发OnJoyStickTouchMove事件

if (isTouched && touchedAxis.magnitude >= JoyStickRadius)

{

if (this.OnJoyStickTouchMove != null)

this.OnJoyStickTouchMove(TouchedAxis);

}

//松开虚拟摇杆后让虚拟摇杆回到默认位置

if (selfTransform.anchoredPosition.magnitude > originPosition.magnitude)

selfTransform.anchoredPosition -= TouchedAxis \* Time.deltaTime \* JoyStickResetSpeed;

}

/// <summary>

/// 返回虚拟摇杆的偏移量

/// </summary>

/// <returns>The joy stick axis.</returns>

/// <param name="eventData">Event data.</param>

private Vector2 GetJoyStickAxis(PointerEventData eventData)

{

//获取手指位置的世界坐标

Vector3 worldPosition;

if (RectTransformUtility.ScreenPointToWorldPointInRectangle(selfTransform,

eventData.position, eventData.pressEventCamera, out worldPosition))

selfTransform.position = worldPosition;

//获取摇杆的偏移量

Vector2 touchAxis = selfTransform.anchoredPosition - originPosition;

//摇杆偏移量限制

if (touchAxis.magnitude >= JoyStickRadius)

{

touchAxis = touchAxis.normalized \* JoyStickRadius;

selfTransform.anchoredPosition = touchAxis;

}

return touchAxis;

}

Update()每帧执行，刷新和控制摇杆的位置，包括摇杆移动到最大半径时摇杆无法拖动和松开虚拟摇杆时 摇杆的Touch按钮能回到默认中心位置。

GetJoyStickAxis 函数用于获取虚拟摇杆的偏移量，用返回的二维向量来确定控制物体的位移和方向。

在需要用摇杆控制的物体上加上摇杆控制脚本JoyStickUGUICon.cs，在本游戏中使用摇杆来控制玩家的坦克的移动和方向，核心代码如下:

private JoyStickUGUI js;

void Start()

{

js = GameObject.FindObjectOfType<JoyStickUGUI>();

js.OnJoyStickTouchBegin += OnJoyStickBegin;

js.OnJoyStickTouchMove += OnJoyStickMove;

js.OnJoyStickTouchEnd += OnJoyStickEnd;

}

void OnJoyStickMove(Vector2 vec)

{

float fRs = (vec.x) / Screen.width \* 100000 \* Time.deltaTime;

float fMs = (vec.y) / Screen.height \* 10000 \* Time.deltaTime;

//设置角色朝向

transform.Rotate(Vector3.up \* fRs);

//移动角色

transform.Translate(Vector3.forward \* fMs);

}

在上述代码中，start函数将注册回调函数，包括摇杆开始移动、摇杆移动中、摇杆结束移动函数。OnJoyStickMove为摇杆持续移动的处理函数。

4.2.1 对象池

移动端与PC端相比内存稀缺，我们都希望游戏能够更加稳定，而不能有效的管理内存，此时大量的内存碎片是致命的。 内存碎片的意思是内存被分成一个一个的小块而不是整个大块，所有内存小块的大小可能很大但并不能使用，比如你想分配16byte的内存，此时如果有 20byte的空间就可以分配成功，但是如果这20byte是内存碎片，为两个10byte就会分配失败。所以，如果存在大量内存碎片，理论上有足够的可用内存，也会分配失败。

很多游戏公司的游戏都会进行浸泡测试，让一个游戏跑好几天，查看是否崩溃来检测内存泄露等等，因为内存碎片产生毁灭性的结果是一个缓慢的过程

（1）对象池作用: 提升性能和内存的使用

（2）对象池原理：

将对象存储在一个池子中，当需要时在再次使用，而不是每次都实例化一个新的对象。池的最重要的特性，也就是对象池设计模式的本质是允许我们获取一个“新的”对象而不管它真的是一个新的对象还是循环使用的对象

（3）对象池实现:

我们需要把生成的对象储存在池中，来取出增加回收，有下面几种可以选择

①.数组，此处不是ArrayList就是普通的Object[]这样的数组，在内存中是连续存储的，索引速度非常快，使用起来比较。但是这种数组不能动态扩充，也就是生成对象的数量是不变的，不小心超出了这个范围还会产造成数据溢出，而且只能存储一种类型的对象。

②.ArrayList，动态数组，可以动态扩充，也可以存储不同类型对象，但是在操作不同类型数据时需要装箱拆箱（要做一个强制转换），带来性能耗损，并且不是类型安全。

③.List<T>，泛型，可动态扩充，但是不能存储不同类型数据，需要制定存储数据类型T。安全类型，不存在装箱拆箱。

除此三种基本的之外还可以用堆，栈，哈希，Dictionary，如果需要根据key来查找具体的对象可以用哈希或Dictionary。

下面以代码用C#的 List<T>来示范:

public class EnemyPool : MonoBehaviour

{

public GameObject Hero;

public GameObject perfab;

List<GameObject> enemy = new List<GameObject>();

int Max\_Amount = 10;

// Use this for initialization

void Start()

{

InvokeRepeating("setEnemy", 1, 10);

for (int i = 0; i < Max\_Amount; i++)

{

enemy.Add(Instantiate(perfab, Vector3.left\*i\*2, Quaternion.identity) as GameObject);

}

}

void addEnemy()

{

enemy.Add(Instantiate(perfab, Vector3.zero, Quaternion.identity) as GameObject);

++Max\_Amount;

}

}

在本游戏中此次采用一个功能非常强大的对象池插件——PoolManager。这是目前比较成熟的Unity3D缓冲池插件，借助此插件 完成游戏中坦克对象、炮弹对象的管理，写一个通用的方法 生成对象池：

public static SpawnPool CreatePoolWithPrefab(PoolInfo pi)

{

SpawnPool pool;

pool = PoolManager.Pools.Create(pi.sName);

//池子建在本MAIN物体的子物体

pool.group.parent = pi.tParent;

//先将PREFABS存在PRE池里

PrefabPool prefabPool = new PrefabPool(pi.tPrefab.transform);

prefabPool.preloadAmount = pi.iPreloadAmount; //池子最大PF数量

prefabPool.cullDespawned = pi.bAutoDespawn;//池子过大自动清理，深度化化时使用。

prefabPool.cullAbove = pi.iCullAbove;

prefabPool.cullDelay = pi.iCullDelay;

prefabPool.limitInstances = pi.bLimitInstances;

prefabPool.limitAmount = pi.iLimitAmount;

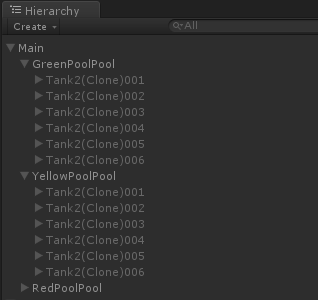
prefabPool.limitFIFO = pi.bLimitFIFO;

pool.CreatePrefabPool(prefabPool);//将上述N个实例创建。

return pool;

}

在游戏中生成各种坦克的对象池：



4.3.3 日志系统

在开发中，往往会觉得Unity3D自带的日志系统不是那么好用，比如，不能筛选日志，不能打印在屏幕上，想生成自己的日志文件，等等等等的需求，这时候就需要自己来扩展编日志的相关功能了。在本次游戏中专门写了一个日志类 用来输出日志文件和管理日志的各种输出，方便程序调试。

public class Log

{

public delegate void LogFunc(object obj);

public static LogFunc Error = UnityEngine.Debug.LogError;

#if UNITY\_EDITOR

public static LogFunc Debug = UnityEngine.Debug.Log;

public static LogFunc Warning = UnityEngine.Debug.LogWarning;

static int line = 0;

public static void Print(string info)

{

line++;

string path = Application.dataPath + "/LogFile.txt";

StreamWriter sw;

if (line == 1)

{

sw = new StreamWriter(path, false);

string fileTitle = "日志文件创建的时间 " + System.DateTime.Now.ToString();

sw.WriteLine(fileTitle);

}

else

{

sw = new StreamWriter(path, true);

}

string lineInfo = line + "\t" + "时刻 " + Time.time + ": ";

sw.Write(lineInfo);

sw.WriteLine(info);

sw.Flush();

sw.Close();

}

#else

public static void Debug(object obj) { }

public static void Warning(object obj) { }

public static void Print(string info) { }

#endif

}

此日志系统有两个新功能 (1)在编辑器模式下才输出日志内容，其他运行模式下执行空函数，不输出实际内容。使用Unity的预处理#if UNITY\_EDITOR 来实现

（2）将日志内容 输出到指定文件中 方便查看，如 写下如下代码：

则在项目当前文件夹 LogFile.Txt文本中可看到：

Log.Print("这是第一次日志输出");

Log.Print("这是第二次日志输出");

Log.Print("这是第三次日志输出");

4.4 游戏功能实现

日志文件创建的时间 5/21/2017 8:38:01 PM

1 时刻 0: 这是第一次日志输出

2 时刻 0: 这是第二次日志输出

3 时刻 0: 这是第三次日志输出

4.3.6 武器系统

4.3.6 坦克系统

4.3.6 寻路系统

4.3.5 音效系统

第五章 游戏展示与性能分析

5.1 游戏展示

5.2 性能分析

5.3 优化

由上面性能分析可知，此游戏性能还存在很大优化空间，在本次研究试验中总结了一些优化经验，使用过后，效果甚好:  
（1）如上所述，最好不要频繁使用GetComponent，尤其是在循环中

（2）善于使用OnBecameVisible()和OnBecameVisible(),来控制物体的update()函数的执行以减少开销。不要直接访问gameobject的tag属性。比如if (go.tag == “human”)最好换成if (go.CompareTag (“human”))。因为访问物体的tag属性会在堆上额外的分配空间。如果在循环中这么处理，留下的垃圾就可想而知了。

（3）使用缓存池：despawn并不销毁对象，而是暂移走，再创建时，他自动再将他调回来，但是！！！不执行start，所以私有一些初始变量要用专有函数处理，并在创建时调用赋值。

（4）相同的模型尽量打包在一起，他们会公用一套资源文件。不相同的模型尽量分开打包，相同模型具有不同的脚本、组件的话把他们放在不同的Prefab中，最后把这些Prefab一起打包在一个Assetbundle中。如下图所示，现在Project视图中选择需要打包的Prefab，然后在导航菜单栏中选择Create Assetbundles Main表示分别打包、Create AssetBundles All表示将他们打包在一起。

第六章 项目总结与展望

6.1 项目总结

本次基于Unity3D的设计，最后发布的游戏原型，实现了基本的游戏效果。3D的地形效果比较真实，游戏中的天空、光照效果也基本实现。坦克360°旋转行走，可以发射炮弹，并且有爆炸效果，游戏界面中有炮弹状态的图形和文字效果。不过由于自身水平有限，加上时间也不够充足，设计存在很多不足之处：

1. 游戏原型比较简陋，功能较少。

2. 炮塔填装效果未实现，不够完善。

3. 模型设计不够美观

6.2 后续工作展望

时域积分方程方法的研究近几年发展迅速，在本文研究工作的基础上，仍有以下方向值得进一步研究：

……

致 谢

本论文的工作是在我的导师 杨陈 罗嘉庆 的亲切关怀与细心指导下完成的。从课题的选择到论文的最终完成，两位老师始终都给予了细心的指导和不懈的支持。在进度和效果方面，老师不断的给我提示和意见，让我能更好的完成毕业设计。

还要感谢长期以来给我诸多帮助的同学们，设计中的遇到问题都能及时给予我帮助。你们的友情将是我一生最值得珍惜的财富和最值得怀念的情感。

转眼间,四年的大学生活即将结束，感谢我的父母、老师、同学以及所有帮助过我的人，祝每个人开心快乐。

……

参考文献

1. 中国版协游戏工委. 2012年中国游戏产业调查报告[R].2012.
2. 魏强.3D棒球游戏的设计与实现[D].厦门:厦门大学.2008.
3. Dave Shreiner 等著,徐波等译.OpenGL编程指南(原书第五版) [M].北京:机械工业出版社.2006.6-8.
4. 王树军.三维游戏引擎中物理关键技术的研究[D].天津:天津大学.2007.
5. Luke Ludwig.John Haurykiewicz. Collision checking anlaysis tool:discovering dynamic collissions in a modeling and simulation environment [J]. Int J Interact DesManuf(2007):135-141.
6. 谭同德，许绘香.，赵红领，余晓霞.一种基于Shadow Mapping的阴影生成改进算法[J].计算机共产与应用.2008,(32):165-168
7. 朱柱.基于Unity3D的虚拟实验系统设计与应用研究[D].武汉:华中师范大学.2012.
8. Unity.Unity3D参考手册.Unity,2010:94-121
9. 孔令德.计算机图形学基础教程.北京:清华大学出版社,2008:26-35
10. 微软公司;《Visual C# 2005》[M], 高等教育出版社2007年版，全书

外文资料原文

Animation workflow

Unity‟s animation system is based on the concept of Animation Clips, which contain information about how certain objects should change their position, rotation, or other properties over time. Each clip can be thought of as a single linear recording. Animation clips from external sources are created by artists or animators with 3rd party tools such as Max or Maya, or come from motion capture studios or other sources.

Animation Clips are then organised into a structured flowchart-like system called an Animator Controller. The Animator Controller acts as a “State Machine” which keeps track of which clip should currently be playing, and when the animations should change or blend together.

A very simple Animator Controller might only contain one or two clips, for example to control a powerup spinning and bouncing, or to animate a door opening and closing at the correct time. A more advanced Animator Controller might contain dozens of humanoid animations for all the main character‟s actions, and might blend between multiple clips at the same time to provide a fluid motion as the player moves around the scene.

Unity‟s Animation system also has numerous special features for handling humanoid characters which give you the ability

to retargethumanoid animation from any source (Eg. motion capture, the asset store, or some other third-party animation library) to your own character model, as well as adjusting muscle definitions. These special features are enabled by Unity‟s Avatar system, where humanoid characters are mapped to a common internal format.

Each of these pieces - the Animation Clips, the Animator

Controller, and the Avatar, are brought together on a GameObject via theAnimator Component. This component has a reference to an

Animator Controller, and (if required) the Avatar for this model. The Animator Controller, in turn, contains the references to the Animation Clips it uses.The above diagram shows the following:

Animation clips are imported from an external source or created within Unity. In this example, they are imported motion captured humanoid animations.

The animation clips are placed and arranged in an Animator Controller. This shows a view of an Animator Controller in the Animator window. The States (which may represent animations or nested sub-state machines) appear as nodes connected by lines. This Animator Controller exists as an asset in the Project window.

The rigged character model (in this case, the astronaut “Astrella”) has a specific configuration of bones which are mapped to Unity‟s common Avatar format. This mapping is stored as an Avatar asset as part of the imported character model, and also appears in the Project window as shown.

When animating the character model, it has an Animator component attached. In the Inspector view shown above, you can see the Animator Component which has both the Animator

Controller and the Avatar assigned. The animator uses these together to animate the model. The Avatar reference is only necessary when animating a humanoid character. For other types of animation, only an Animator Controller is required.

外文资料译文

Unity动画工作流程

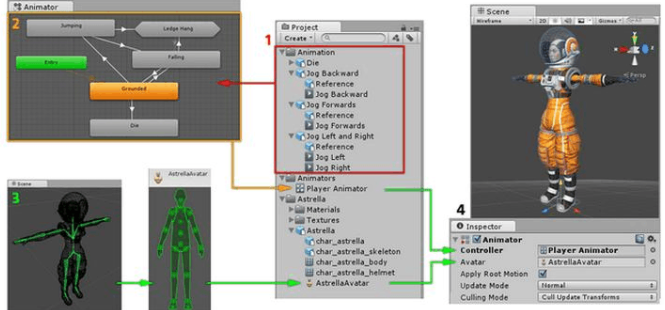
动画系统是基于统一的概念动画短片包含关于特定对象信息应该改变他们的位置,旋转,或其他属性。 每个片段可以被认为是一个线性记录。动画剪辑从外部来源是由艺术家和动画师等第三方工具Max或玛雅,或来自动作捕捉工作室或其他来源完成。

动画剪辑然后组织成一个结构化流程图一样的动画叫系统控制器。 动画控制器作为一个“状态机”跟踪剪辑现在应该改变或混合在一起。

一个非常简单的动画控制器可能只包含一个或两个片段,例如控制powerup旋转和跳跃,或有生命的一扇门打开和关闭在正确的时间。 更先进的动画控制器可能包含几十个人形动画的主要人物的行动,和可能之间的混合多个剪辑同时提供流体运动的球员围绕现场。

 统一的动画系统也有许多特色处理人形角色给你的能gdp8 %从任何来源(如人形动画。 动作捕捉,资产存储或其他第三方动画库)自己的角色模型,以及调整肌肉的定义。这些特点使统一的Avatar系统,人形角色被映射到一个通用的内部格式。

这些产品动画短片,动画控制器,Avatar GameObject通过,都聚集在一起动画组件。 此组件有一个引用一个动画控制器,如果需要Avatar这个模型。 反过来,动画控制器包含引用动画短片它使用.



上面的图显示如下:

动画短片中从外部源进口或创建团结。在这个例子中,他们是进口运动捕获的人形动画。

动画剪辑放置和安排在一个动画师控制器。 这显示了一个动画师的视图控制器在动画窗口中。 (这可能代表动画或嵌套的子状态机),由线连接节点。 这个动画控制器存在作为一种资产在项目窗口。

操纵角色模型(在这种情况下,宇航员“Astrella”)有一个特定的配置映射到统一的共同的骨头Avatar格式。 这种映射是存储为一个Avatar资产作为导入的角色模型的一部分,也出现在项目窗口如图所示。

当动画角色模型,它有一个动画师组件连接。 在Inspector视图所示,可以看到动画组件这既有动画控制器和Avatar分配。动画师使用这些动画模型。Avatar的引用只在动画时需要一个人形的性格。对于其他类型的动画,只需要一个动画控制器。

统一的动画系统(称为“Mecanim”)提供了大量的概念和术语。如果在任何时候,你需要去我们的动画术语表找出一些手段。