**目 录**

[第1章 引言 1](#_Toc515791118)

[1.1研究背景及意义 1](#_Toc515791119)

[1.2 研究目的和意义 3](#_Toc515791120)

[1.3研究内容和主要章节结构 3](#_Toc515791121)

[第2章 FPS游戏需求分析 4](#_Toc515791122)

[2.1用户分析与产品定位 4](#_Toc515791123)

[2.1.1 用户分析 4](#_Toc515791124)

[2.1.2 产品定位 4](#_Toc515791125)

[2.2 游戏引擎的选型 4](#_Toc515791126)

[2.2 FPS游戏玩法分析 5](#_Toc515791127)

[2.3.1 游戏双方玩法 5](#_Toc515791128)

[2.3.2 地图场景分析 5](#_Toc515791129)

[2.4 游戏系统 6](#_Toc515791130)

[第3章 游戏系统的设计与实现 8](#_Toc515791131)

[3.1 整体系统的设计 8](#_Toc515791132)

[3.2 网络系统 8](#_Toc515791133)

[3.2.1 实例化与Spawn System 9](#_Toc515791134)

[3.2.2 玩家管理 9](#_Toc515791135)

[3.2.3 High Level API 10](#_Toc515791136)

[3.2.4 NetworkBehaviour 10](#_Toc515791137)

[3.3 控制系统 12](#_Toc515791138)

[3.3.1 移动与姿态调整 12](#_Toc515791139)

[3.3.2 瞄准与查看 13](#_Toc515791140)

[3.3.3 射击与射线检测 14](#_Toc515791141)

[3.4 武器系统 15](#_Toc515791142)

[3.4.1 弹痕效果 16](#_Toc515791143)

[3.4.2 弹道效果 18](#_Toc515791144)

[3.4.3 群体范围伤害 22](#_Toc515791145)

[3.5 工程整体结构 23](#_Toc515791146)

[第4章 系统测试与性能分析 25](#_Toc515791147)

[4.1整体结构 25](#_Toc515791148)

[4.2功能测试 25](#_Toc515791149)

[4.2.1 主界面 25](#_Toc515791150)

[4.2.2 加入游戏 26](#_Toc515791151)

[4.2.3 游戏设置 27](#_Toc515791152)

[4.2.4 选择武器 28](#_Toc515791153)

[4.2.5 得分与胜利 28](#_Toc515791154)

[4.3性能分析 29](#_Toc515791155)

[结论 30](#_Toc515791156)

[参考文献 31](#_Toc515791157)

[致谢 32](#_Toc515791158)

# 第1章 引言

1.1研究背景及意义

射击类游戏（Shooter Games，STG）是常见的游戏类型之一，它是从动作类游戏中发展出来的独立游戏类型，射击游戏拥有很明显的动作游戏特点。射击游戏风靡大街小巷，从之前几乎每台计算机上都运行过的反恐精英（Counter Strike）与最近大热的守望先锋（Overwatch）与彩虹六号（RainbowSix）都证明了这个类型游戏的受欢迎程度之深。本文主要论述一个使用Unity3D引擎开发一个可以在局域网内进行多人联机的射击游戏《Project K》以及遇到的问题，同时本文中对每一个出现的问题都提出了对应的解决方案，力求打造一个游戏性表现力均出色的多人联机游戏。

电子游戏能相关技术的发展过程：

首先是2D游戏时代，在电子游戏发展之初，由于受到计算机硬件的限制，图像技术只能用像素显示，当时的游戏从DOS时期的480\*320分辨率到Windows标准化的640\*480，再到后来的800\*600、1024\*768等高清分辨率，但是总体上游戏画面仍然是二维表现[1]。

随后是3D游戏时代，随着3D加速显卡的诞生与3D图形接口DirectX等的出现，QUAKE与《古墓丽影》游戏先后问世，3D游戏时代正式到来。这一时期的游戏画面更加惊艳，同时多出一个坐标轴的设定也使得游戏玩法开始愈发的多样化，游戏性和表现力得到了极大提高。

最后是网络游戏时代，随着科技的发展和时代的进步，单一的游戏模式开始让人逐渐厌倦，无论什么类型的游戏，无论AI（人工智能）系统设计的多么高级，即时像是沙盘游戏这样极高自由度的设定，在若干次重复游戏后，人们总会感到固定程序模式带来的一种束缚感。解决这个问题唯一的出路就是通过网络化来实现，让不同的电脑主机通过联网游戏进行对战，在这种思路的引领下网络游戏开始出现，游戏也从此进入了一个全新的时代。在此期间出现过联机对战网游时代、MMO网游启蒙时代、旧MMO网游时代与新MMO网游时代。

电子游戏能相关技术的发展过程：

首先是2D游戏时代，在电子游戏发展之初，由于受到计算机硬件的限制，图像技术只能用像素显示，当时的游戏从DOS时期的480\*320分辨率到Windows标准化的640\*480，再到后来的800\*600、1024\*768等高清分辨率，但是总体上游戏画面仍然是二维表现[1]。

随后是3D游戏时代，随着3D加速显卡的诞生与3D图形接口DirectX等的出现，QUAKE与《古墓丽影》游戏先后问世，3D游戏时代正式到来。这一时期的游戏画面更加惊艳，同时多出一个坐标轴的设定也使得游戏玩法开始愈发的多样化，游戏性和表现力得到了极大提高。

最后是网络游戏时代，随着科技的发展和时代的进步，单一的游戏模式开始让人逐渐厌倦，无论什么类型的游戏，无论AI（人工智能）系统设计的多么高级，即时像是沙盘游戏这样极高自由度的设定，在若干次重复游戏后，人们总会感到固定程序模式带来的一种束缚感。解决这个问题唯一的出路就是通过网络化来实现，让不同的电脑主机通过联网游戏进行对战，在这种思路的引领下网络游戏开始出现，游戏也从此进入了一个全新的时代。在此期间出现过联机对战网游时代、MMO网游启蒙时代、旧MMO网游时代与新MMO网游时代。

当今电脑游戏的内容基本由引擎控制，所以游戏的发展进程基本是由引擎的发展决定的。

在如今的时代，游戏引擎虽然可以拿来用作各种类型游戏的研发设计，但从世界游戏引擎发展史来看，引擎却总是伴随着FPS（第一人称射击游戏）的发展而变化，无论是第一款游戏引擎的诞生，还是次世代引擎的出现，游戏引擎往往都是依托于FPS游戏作为载体展现在世人面前，这已经成为游戏引擎发展的一条定律。

肯·希尔弗曼于1994年开发的Build引擎是一个重要的里程碑，这款引擎的前身就是家喻户晓的《毁灭公爵》（Duke Nukem 3D），之后id Software公司在1996年公布的《雷神之锤》（Quake）实现了游戏引擎的突破，让当时的人们切实感受到3D游戏画面的震慑力量。一年之后，id Software公司推出《雷神之锤2》一举确定了自己在3D引擎市场上的霸主地位。在Quake II大放异彩的时候，Epic公司开发的Unreal引擎所带来的表现力是空前震撼的，如今Unreal引擎早已从激烈的竞争中脱颖而出，成为当下主流的次世代游戏引擎，如今市面上基于PC端的大型单机游戏采用Unreal引擎的已经称为常态。

2000年开始3D引擎朝着两个不同方向发展，一是像《半条命》、《神偷》和《杀出重围》那样通过融入更多的叙事成分、角色扮演成分以及加强人工智能来提高游戏性；二是朝着纯粹的网络模式发展，在这一方面id Software再一次走到了整个行业的最前沿，在Quake II出色的图像引擎基础上加入更多的网络互动方式，推出了一款纯网络游戏《雷神之锤3竞技场》（Quake III Arena），它与Epic公司之后推出的《虚幻竞技场》（Unreal Tournament）一同成为引擎发展史上一个新的转折点。

同时德国Crytek Studios公司的CryEngine通过一款《孤岛危机》游戏惊艳四座，该引擎强大的物理模拟效果和自然景观技术一样处于顶尖水平，同时句有很多绘图、物理和动画技术的加强，包括体积云、即时动态光影、场景光线吸收、3D海洋技术、高动态光照渲染、进阶粒子效果等等。只可惜Crytek公司在2017年由于自身原因关闭了除引擎核心部门的其余工作室。

随着智能手机在世界范围的普及，手机游戏3D引擎被Unity Technologies公司所研发的Unity3D引擎所实现，以高质量的渲染系统、高级光照系统、粒子系统、动画与地形编辑系统、UI、物理引擎等占据了手游研发市场的50%以上，甚至许多大型单机游戏也逐渐开始使用Unity3D游戏引擎[3]。本论文所讨论的这个游戏即是使用Unity2017版本为基础进行开发的。

以游戏引擎为中心的集体合作时代已经到来，这就是当进游戏技术领域所称的“游戏引擎时代”。

1.2 研究目的和意义

本论文所讨论的《Project K》项目作为一款第一人称射击游戏，可以让玩家在虚拟世界中体验到日常生活中难以体验到的经历，同时得益于Unity引擎强大的底层设计，开发过程中开发者可以更加注重游戏的表现力玩法乐趣等提升游戏体验的部分。

目前市面上的第一人称射击游戏已经百花齐放，本项目在吸收和借鉴了部分游戏的玩法上对部分技术加以改进与创新，将常规游戏中只有一种类型的弹道进行了拓展，实现了多种武器拥有不同的弹道甚至不同的弹道类型。在技术层面上，对主流游戏内的基本系统全都实现，并对部分系统如武器系统实现了多样化的拓展。

1.3研究内容和主要章节结构

本课题研究的主要内容是实现一个可以局域网多人联机的FPS对战游戏，同时对游戏中的一些功能进行详细实现与整合，如对战功能，网络匹配功能，状态同步功能等进行详细的细分与实现。

整个工程采用Unity2017版本游戏引擎进行开发，同时为了进行手柄适配引入第三方动态链接库XInputInterface并通过工程内的XInput类进行拓展以满足游戏需要。

本论文对游戏内各个模块实现原理以及关键代码进行讨论，分章节讨论了地图分析、系统分析、网络系统实现、单机部分功能实现等多个方面。

# 第2章 FPS游戏需求分析

《Project K》是一款玩法简单，游戏内容丰富的局域网联机游戏，游戏支持最多4人同时联机游玩。其定位是追求简单，无需注册，同时喜欢与身边的朋友一起游戏的人群。

2.1用户分析与产品定位

2.1.1 用户分析

FPS游戏起源很早，国内也很早引进了FPS这一游戏类型，而且FPS游戏属于3D游戏一种，需要快速的反应与熟练的操作技巧。游戏中的操作种类不多，但是要求连贯性与快速性。例如玩家口中的“走位瞄准射击”需要一气呵成。而且对游戏地形需要非常熟悉，所以每局游戏采用固定地图，固定规则与规定武器，使得玩家能够在游戏过程中不断熟悉与获得乐趣[2]。

2.1.2 产品定位

《Project K》的产品定位是一款局域网多人联机对战游戏，无法单人进行游戏，所有玩家需处于同一局域网内。

玩家需要利用游戏中有限的武器自由组合成自己顺手的武器套装。通过多种游戏模式来对丰富游戏可玩性。

2.2 游戏引擎的选型

开发一款好游戏的基本工作，是做好游戏需求分析。FPS游戏作为一种出现多年并在每次更新都伴随着引擎技术的更新换代的一种游戏类型来说，游戏引擎需求分析显得尤为重要。不同的开发模型对应不同分析方法。游戏中的功能与玩法会随着开发进度的推进而做出不同的改变。这些改变最终都是对游戏核心，对游戏引擎需求功能的改变。

对于游戏来讲，用户就是玩家。作为游戏开发者需要不断了解游戏玩家的需求，了解玩家想要一个怎样的游戏，了解玩家觉得好玩的游戏应当是怎样的，在本文所讨论的范围内，具体就是如何开发一个好玩的FPS游戏。首先，玩家对于好游戏最为津津乐道的一点就是创新，随着对游戏引擎的开发与掌控，逐渐产生一些新的创意点，或者因为技术本身的问题，有些之前文档中明确规划的功能无法实现[3]。这些变动都是合理的，但是迭代开发始终围绕着一个核心进行改动，对需求分析的改动同样是围绕这一个核心来进行的。这要求一开始就对游戏核心功能与一些外围功能进行清晰的设计与分析。从而使得代码与需求有着有机的结合，而非在开发过程中如同无头苍蝇一般胡乱改动[4]。

2.2 FPS游戏玩法分析

FPS游戏的核心玩法是：游戏中拥有玩家与敌人两种不同的阵营，玩家扮演单个角色，而敌人可能拥有多名。玩家与敌人均可以使用枪械或者其他兵器对另一阵营实施攻击。一般而言，游戏会对武器使用进行限制，使得人物只有有限的资源对对方进行攻击。而人物本身也属于一种资源可以被消耗，这个消耗值一般称为健康值或者生命值，这个资源属于游戏中的核心资源。当一方的核心资源被另一方消耗殆尽时，游戏结束。

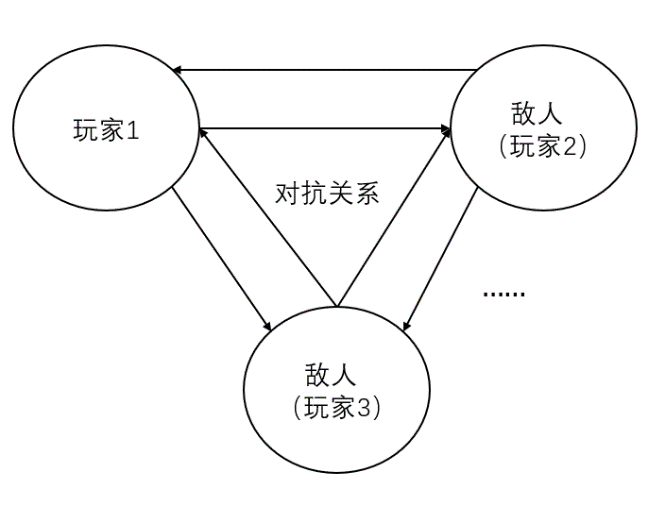


图2.1 FPS游戏核心玩法示意图

2.3.1 游戏双方玩法

FPS游戏中玩家一方由于可操作对象只有一个，其玩法也比较固定。玩家通过自己对游戏对象的移动、瞄准进行操作实现在场景中的漫游，同时通过游戏给定的武器系统中选择合适的武器来攻击敌方。玩家具有武器类型、生命值、碰撞体大小、攻击力、移动速度等属性。

而一般的对抗性游戏中，玩家与敌人的属性与操作方式完全一致，这样可以保证游戏的公平性。所以对于敌方而言，其玩法与玩家一方并无两样。而且这两个概念是互相对等的。玩家一的本身为玩家阵营，同时也是玩家二的敌方阵营。玩家二本身是玩家阵营，同时也是玩家一的敌方阵营。

2.3.2 地图场景分析

游戏中的地图是游戏非常关键的一部分，所有阵营的玩家均只能在游戏规定的区域内进行活动。对于对抗性FPS游戏而言，一张地图的质量关系到游戏公平性与乐趣。2000年发行的Counter-Strike（反恐精英）中的dust2地图，国内俗称沙漠2（如图2.2所示）就充分说明了这一点。该地图在引导、节奏控制与平衡上都相当出色，而且它还并不是一张对称性的地图。

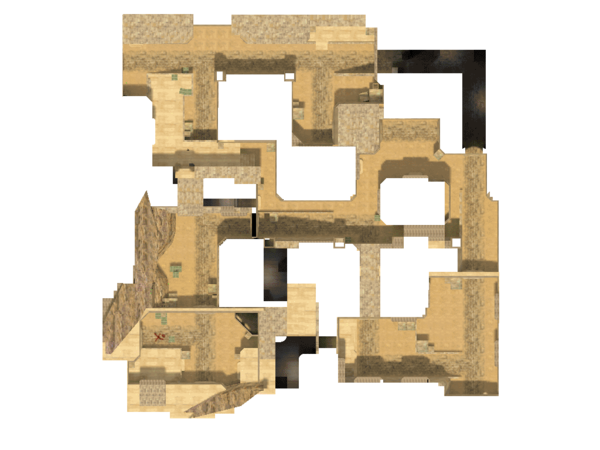


图2.2 de\_dust2地图

FPS游戏中的地图不仅作为游戏背景的载体，同时也是承担游戏各种事件的一个载体。游戏中的事件包括人物刷新，投掷物武器爆炸效果计算，人物移动检测的根据等[5]。

2.4 游戏系统

游戏系统需要满足游戏核心设计理念：局域网多人联机对抗。同时要对游戏有良好的引导方式，使得不同层次的玩家进入游戏后能够迅速了解游戏玩法与模式，迅速上手后融入虚拟世界中[6]。

（1）准备阶段

首先玩家无需进行繁杂的注册与登陆操作，只需要在主界面指定自己的游戏昵称即可搜索、加入和创建游戏。如果游戏房间内已经存在同名玩家，则后面进入的玩家无法在不更改游戏昵称的情况下继续游戏。

（2）游戏模式

游戏战斗规定只能有一方或者一名玩家获得胜利，在团队模式下，一方队伍需消灭对方所有成员才能获得胜利。在个人模式下，玩家需要获得指定的分数即可结束比赛获得胜利。

武器系统要丰富，武器系统分为即时命中武器与投掷武器。即时命中武器又分为手枪，冲锋枪、突击步枪三种类别，投掷武器分为手雷、榴弹发射器与火箭炮三种类别。每名玩家可以自由选择武器，最多可以同时携带三把武器，但是只能同时使用一种武器。

游戏场景也需富有表现力与公平性，游戏应提供两张以上不同的地图。每张地图需要包括多种元素，如平地、高台、转角等[7]。

系统性能需求要考虑游戏的加载速度和稳定性，保证玩家可以正常游戏。对于游戏画面载入，操作延迟，响应速度，地图生成缩放速度，图像重绘速度方面有一定要求。游戏启动时间，操作响应时间，策略部署保存时间，符合性能标准不大于10 秒。多人同时在线，游戏需要依然运行流畅，游戏画面载入，操作延迟，响应速度，地图生成缩放速度在10 秒之内。

本课题中的游戏项目整体功能模块结构图如图2.3所示。

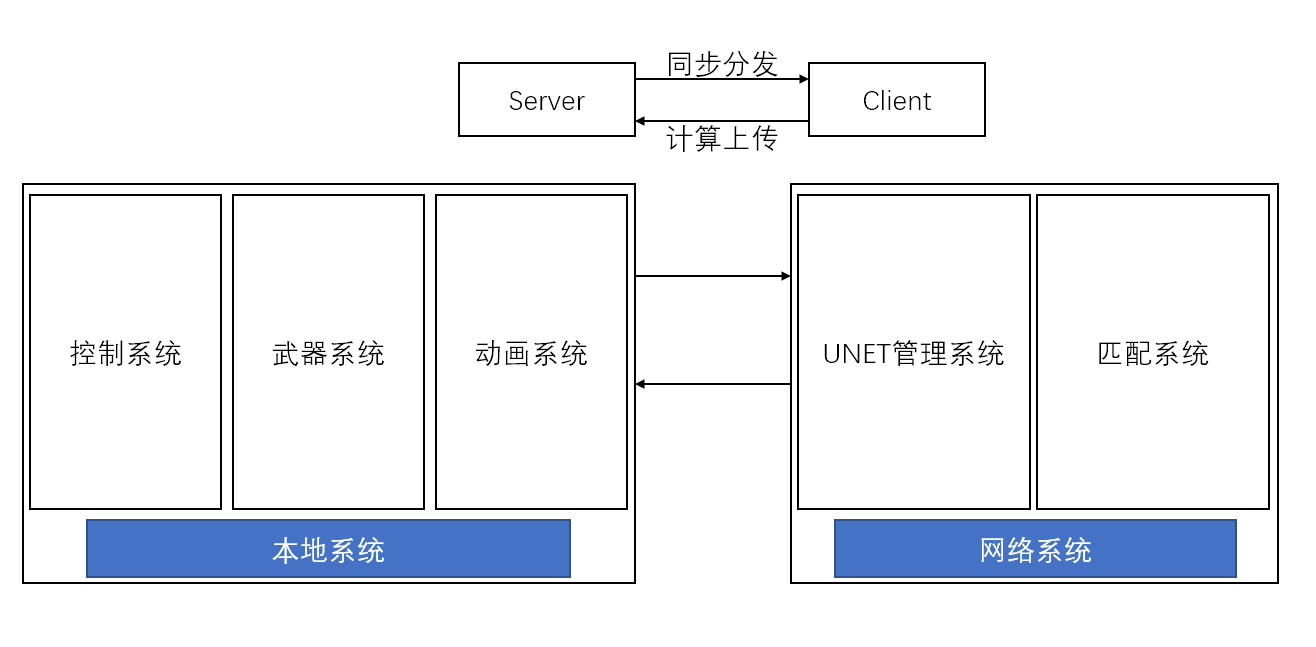


图2.3 功能模块结构图

游戏中的功能模块主要分为两个大类，其中由玩家本地进行运算的部分即为本地系统，其包含控制、武器与动画等子系统，这些子系统由开发者开发，在游戏中体现为玩家的具体操作与游戏体验。而网络系统则为管理控制整个联网机制的主要系统，其包含UNET管理系统与游戏匹配系统两个子系统。同时，本地系统与网络系统互相影响，相互交换信息与传递消息，构成了整个游戏系统。

# 第3章 游戏系统的设计与实现

## 3.1 整体系统的设计

游戏系统设计主要包括主菜单界面、加载场景界面、与主要游戏场景三大模块组成。其中在主菜单界面集成了包括查找游戏，创建游戏，设置与退出游戏等功能按钮；加载场景界面用于异步载入游戏场景实现场景过渡；主场景是游戏的核心场景，包括游戏主要场景与各个玩家的同步游戏，游戏中几乎所有逻辑都在此场景中实现。

游戏的主要流程图如图3.1所示：

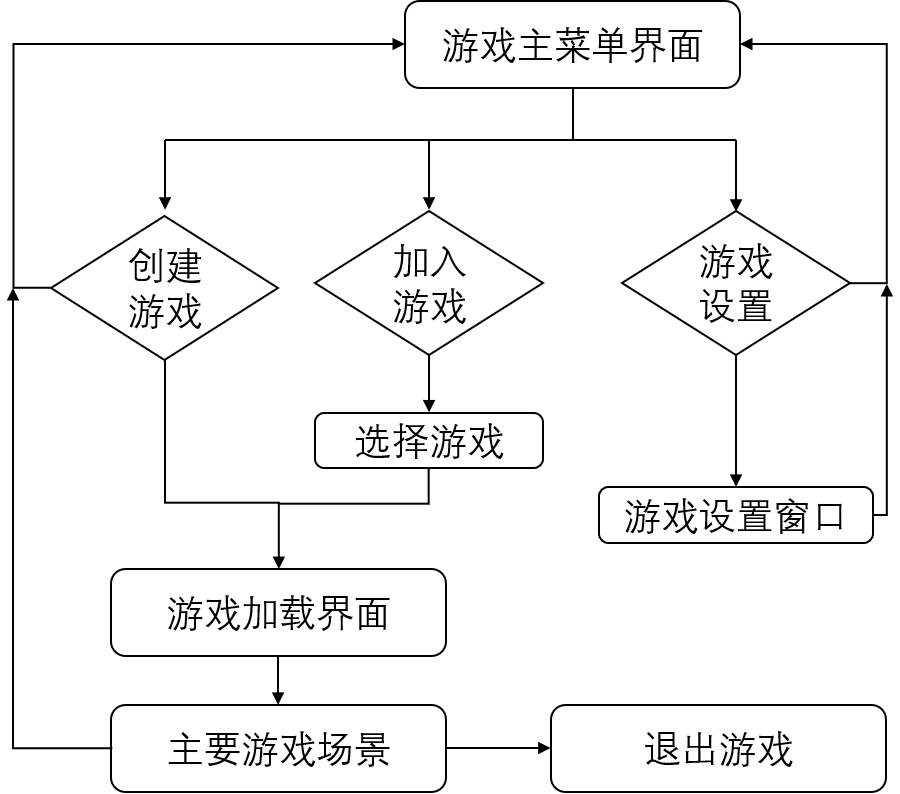


图3.1 游戏系统流程图

## 3.2 网络系统

《Project K》定位是局域网联机，无需注册的轻量化游戏，所有网络通讯都是在局域网内完成，借助与Unity5.1版本之后更新的UNet系统，局域网内的通讯得以方便快捷的实现。

在Unity的网络系统中，游戏拥有服务器与客户端之称，当没有指定的服务器的时候，将由一台客户端充当服务器，此时这台客户端成为“主机（host）”。

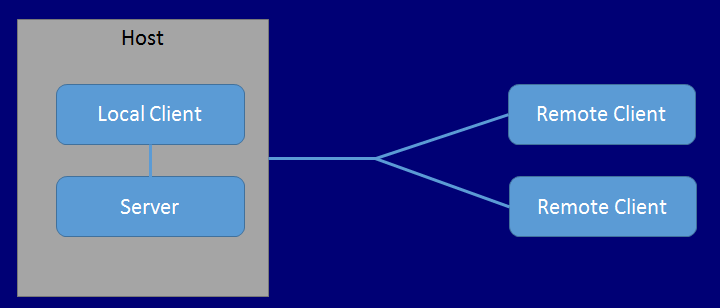


图3.2 主机

主机同时充当游戏中的服务器与客户端，主机本身的客户端称为本地客户端（LocalClient），主机上其他客户端称为模拟客户端（RemoteClient）。本地客户端通过直接函数调用与消息队列与服务器进行通讯，由于这些客户端都处于相同的进程中，实际上它们与服务器共享场景；模拟客户端通过常规网络连接与服务器通讯[8]。

3.2.1 实例化与Spawn System

通常在Unity中，GameObject.Instantiate可以创建实例化对象，但是在网络系统中，这些网络对象同时也必须在网络上被创建。这一过程只能在服务器上完成，并在与服务器连接的所有客户端上也同步创建。一旦产生了对象，Spawn System就使用分布式对象生命周期管理和状态同步原则对所有网络对象进行统一管理。

3.2.2 玩家管理

在网络系统中，“玩家”概念显得尤为重要与特殊，“玩家”对象是与其他游戏对象都有着联系的一种特殊实例化的对象。网络系统会对网络中所有对象进行轮询更新。同时，玩家只能控制属于自己的那个网络对象，无法对其他网络对象进行命令控制，如图3.3所示。

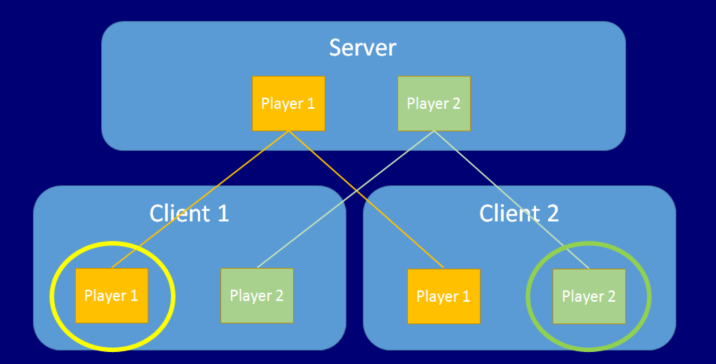


图3.3 客户端玩家

对于非玩家对象，如图3.4所示，类似与NPC、静态物体与场景物体，控制权限驻留在服务器上。



图3.4 非玩家对象权限示意图

在这种模式下，玩家所能控制的对象会被赋予“isLocalPlayer”标签。使用标签的目的是便于输入系统管理，网络系统中所有的客户端操作都只能作用在玩家所控制的对象上。本地对象还会拥有“本地权限”，本地权限的计算全部都在本地执行，这通常用于运动控制。

3.2.3 High Level API

高级API（HLAPI）是为Unity游戏构建多人游戏功能的系统。它建立在较低级别的传输实时通信层之上，并用于处理多人游戏所需的许多常见任务。虽然传输层支持任何种类的网络拓扑结构，但HLAPI是服务器授权系统; 尽管它允许其中一个参与者同时成为客户端和服务器，因此不需要专门的服务器进程。与互联网服务一起工作，这使得多人游戏可以通过互联网运行，而开发人员只需很少的工作。

HLAPI由图3.5所示的一系列层级所组成：

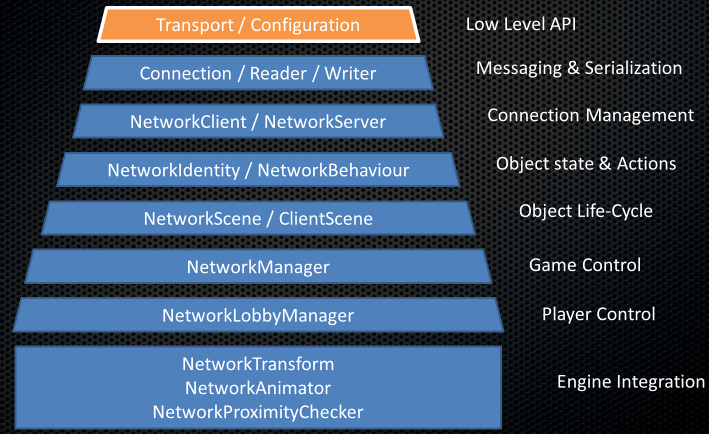


图3.5 HLAPI

HLAPI在Unity中被封装在一个专用的程序集里，并全部存在于UnityEngine.Networking的命名空间中。HLAPI专注于易用与程序迭代开发，对外提供许多高效的API与服务，如

1. 消息处理程序（Message handlers）
2. 通用的高性能序列化（General purpose high performance serialization）
3. 分布式对象管理（Distributed object management）
4. 状态同步（State synchronization）
5. 网络功能公共类，如 Server， Client， Connection等

3.2.4 NetworkBehaviour

Unity中逻辑编程采用脚本化、模块化的思路[9]。开发人员使用Unity制定的一套规则对游戏编写脚本。通常来说，需要挂载在场景中的脚本都需要统一继承MonoBehaviour类以实现序列化。在网络系统中，这一继承规则变为了NetworkBehaviour类。

NetworkBehaviour是一个特殊的与NetworkIdentity协同工作的脚本。继承此类以便执行HLAPI中的开放功能例如Commands、ClientRPCs、SyncEvents 和 SyncVars等。如图3.6所示。

NetworkBehaviour具有以下功能：

1. 同步变量（Synchronized Variables）
2. 网络回调（Network callbacks）
3. 服务器与客户端功能（Server and Client functions）
4. 发送指令（Sending Commands）
5. 客户端RPC调用（Client RPC Calls）
6. 网络事件（Networked Events）

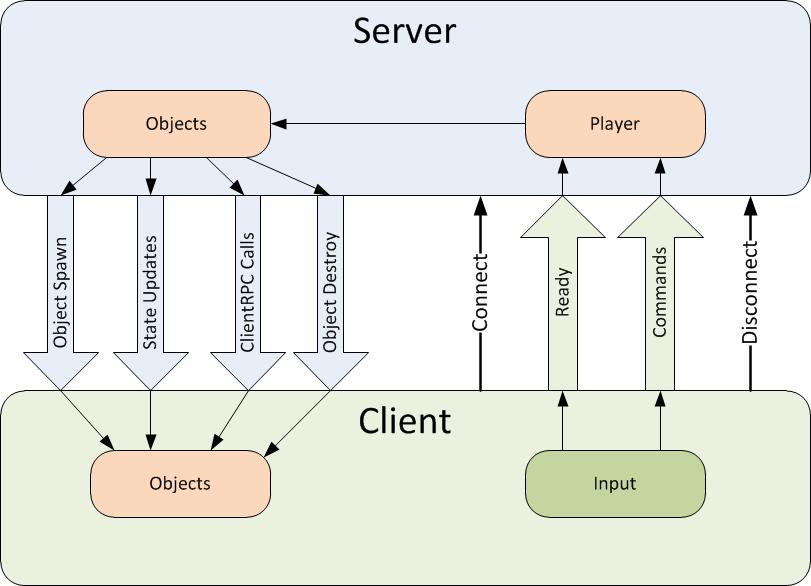


图3.6 NetworkBehaviour功能

NetworkBehaviour中的成员变量可以同步到服务端，由于服务器是整个网络系统中具有最高权限，所以同步方向只能由服务端向客户端进行而不能反过来。客户端请求的事情由Commands执行而不是在客户端本地执行。需要同步的变量需要加上SyncVar特性修饰，且变量需要是基本类型成员变量而不是class、list或者其他集合。下文给出的代码描述了这一特征。

public class Player : NetworkBehaviour

{

[SyncVar] public int health;

[SyncVar] public string playerName;

}

所有SyncVar标记的成员变量值都是保存在服务器端的，一旦这些变量值发生了变化，服务器就会将更新后的变量发送到所有客户端。生成对象的时候，这些对象会在客户端上生成，但是所有的值都会由服务器同步[10]。

## 3.3 控制系统

控制系统包括了玩家能对游戏对象进行的一切控制行为，包括以下几个方面：

3.3.1 移动与姿态调整

移动包括二维平面内的前后左右移动与跳跃，姿态调整包括人物的下蹲操作。移动操作只会对游戏对象与碰撞体的位置进行更新，而不会改变碰撞体的形状，但是姿态调整会影响玩家控制角色的碰撞体体积。

移动控制在《Project K》中是由Movement脚本所控制的，而Movement脚本所依赖的组件是CharacterController，使用CharacterController组件的原因是因为如果使用传统的刚体（rigidbody）组件与物理系统实现移动的话会造成移动控制出现错误，一个140Km/h移动速度的人物可以在瞬间停下来并进行90°转身显然是不符合实际情况的。

CharacterController组件本质上是一个胶囊形状的碰撞体，它会接收所有输入，并只需要告诉组件移动向量与速度，它便可以自行计算出移动效果与随之发生的物理碰撞。同时拥有skin width来保证对象不会被卡住[11]。

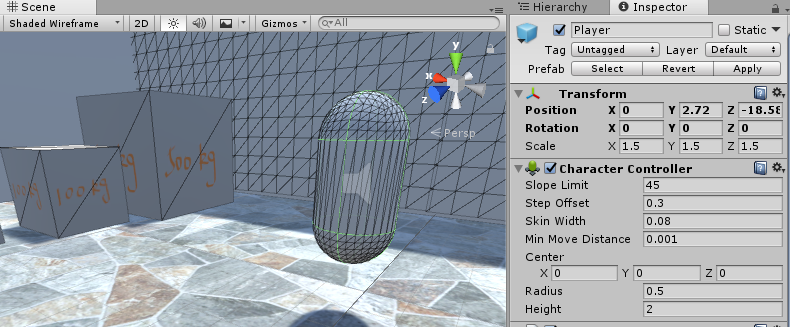


图3.7 CharacterController组件

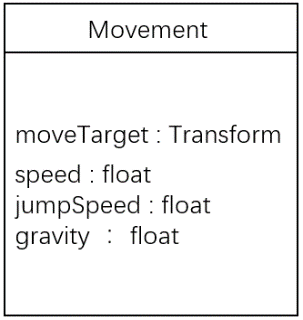
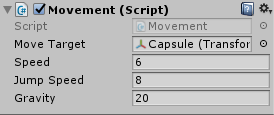
 

图3.8 Movement脚本成员与Inspector上显示效果

Movement脚本中公共变量不多，因为这个脚本仅控制玩家的移动控制而不再负责其余行为，所以在此脚本中仅包含移动对象、移动速度、跳跃速度与重力四个成员变量。

姿态调整主要是下蹲操作，这个操作会使玩家所控制的对象上的碰撞体发生形变，下蹲情况下，玩家所控制的对象会在高度上（y轴）下降，导致碰撞体由原来的1.5缩小至1.3。

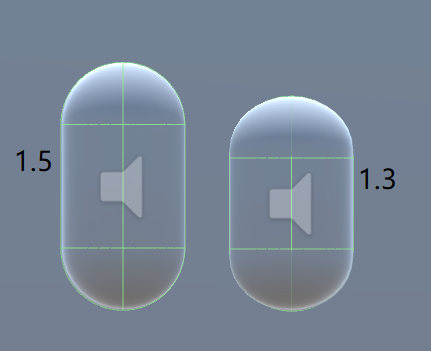


图3.9 下蹲操作前后碰撞体对比

3.3.2 瞄准与查看

瞄准与查看是FPS游戏中一个最基础的功能，360°查看与视野控制都是属于3D游戏一个非常明显的特征。视角查看主要指的是对游戏内玩家控制对象上的摄像机进行有限制的控制，使得玩家可以在场景中实现自由查看。

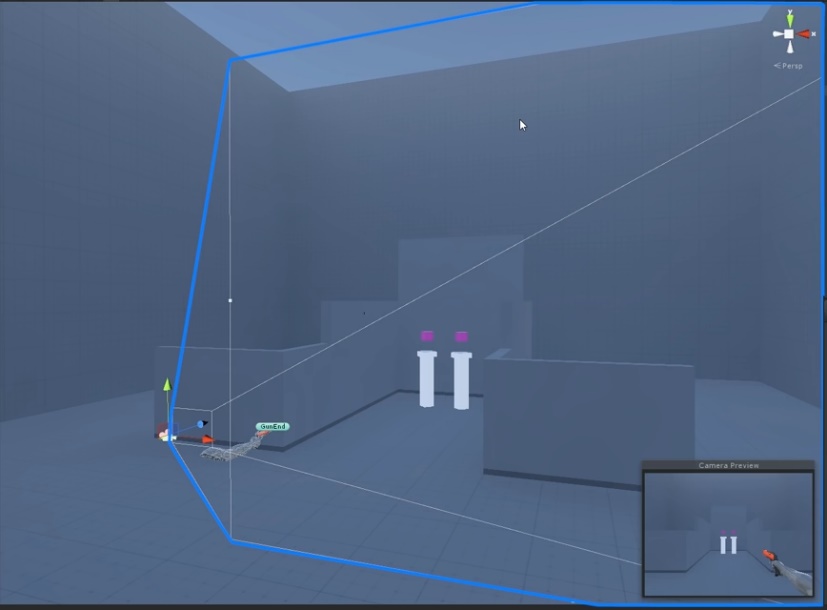


图3.10 玩家所能看见的视野范围

但是这种视野范围不是无限大的，一般而言，FPS游戏的视野范围（Field of View）限定在108会显得比较正常。当然这个值会提供在设置面板中供玩家自主调节。

在《Project K》中，所有的视角控制都是由CameraView组件来实现的，这个组件是依赖于Camera组件来工作的，内部实现的功能是接收输入信息，并根据设定的预设值来计算对Camera的控制以实现视角自由查看的功能。其成员变量图如图3.11所示。

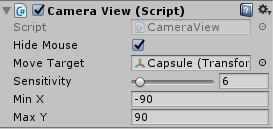
 

图3.11CameraView组件的成员变量与Inspector上显示效果

如果将minX与maxY分别设置为-90与90时，整个Camera控制范围就变成了对环绕y轴无限制而限制了环绕x轴的范围，限制范围应当是（maxY-minX），即为180°。如图3.12所示。

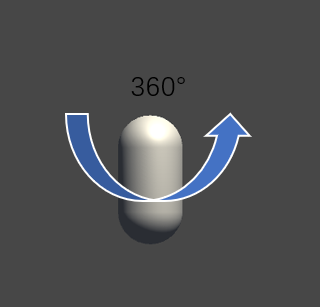
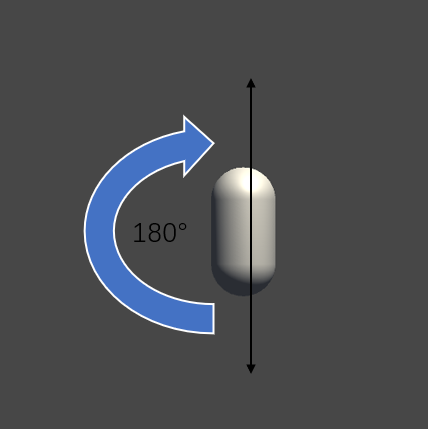


图3.12 可自由观察的视野范围

3.3.3 射击与射线检测

对于一个FPS游戏而言，射击是非常常规且核心的操作。而对于即时命中而言，游戏中的命中判定并非从枪口发射出子弹依靠子弹的碰撞体对对方进行击杀，而是在开枪的瞬间进行命中目标判定，这一判定的核心是射线检测。

射线检测是一个利用程度非常高的算法，不论是使用鼠标还是触控屏，射线检测都在其中起了重要的作用。在游戏中，射线检测也是非常常见的算法。射线检测的目的是实现一种“以某一点为射线端点发射一条射线，对射线所穿过的目标层级中的目标进行标记与返回”的效果。如图3.13，以A为起点AB为方向的一条射线，这条射线穿过了中间物体，那么应当返回中间物体作为算法的返回值。

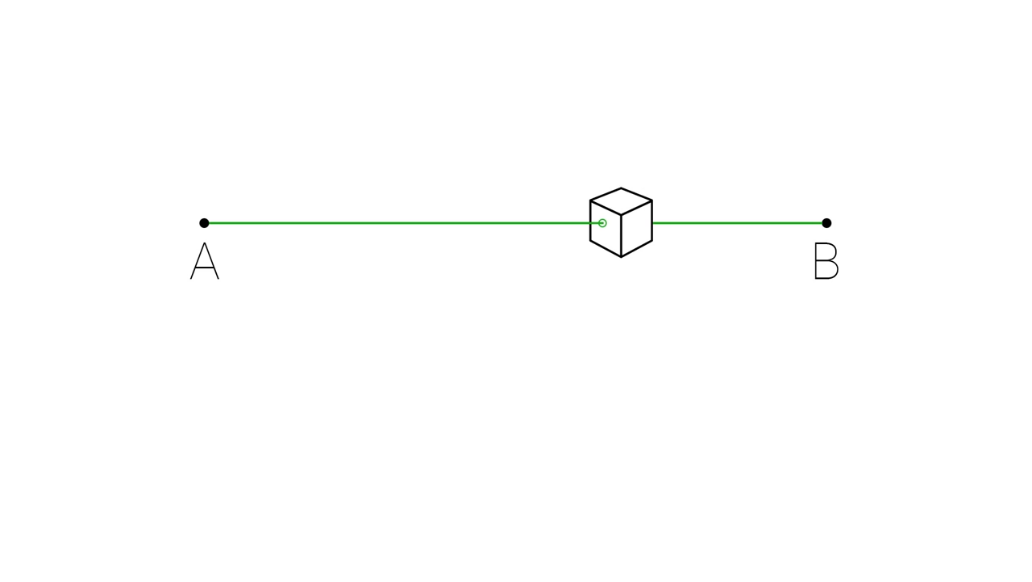


图3.13 射线检测示意图

在《Project K》中，所有的即时命中判定都是通过射线检测进行的，射线检测要确定的两个参数是起点与检测方向，起点是玩家控制的枪口方向，而方向即瞄准方向，如果有武器弹道设定，则方向应当是计算弹道之后的方向。游戏中，具体是由ShootCaster脚本对射击进行控制，Unity引擎中的API接口使用的是Physics.Raycast，这个接口的返回值是bool值，即返回是否检测到物体，若要获取到具体的物体信息，则需传入out类型变量获取该物体的碰撞体引用，通过碰撞体引用中的gameObject引用来获取具体信息。

值得注意的是为了在Sence视图中实时监测射线检测效果，需要在Sence中画出一条辅助线来对射线检测效果进行实时预览，在游戏中，也可以通过这条线来绘制子弹飞行轨迹。这里会遇到一个问题，如果使用Physics.Raycast方法，需要传入一个明确的检测距离，也可以理解为武器的射程。但是FPS游戏中，对武器射程并没有强制性规定，所以这个监测距离一般都是一个非常大的值（理解为无穷大），但是子弹设定并没有穿透效果，为了避免渲染出穿透型轨迹。《Project K》中引入了“射线深度探针”概念。

射线深度探针指的是利用两次射线检测来实现精确的距离计算，在玩家瞄准的时候，会实时计算出目前瞄准方向的第一个碰撞物体与玩家的距离d1。在玩家按下射击键的时候，使用第一步计算出的d1进行精确的射线检测距离判定与轨迹渲染。

射线深度探针的具体代码实现如下：

deepthRayOrigin = fpsCam.ViewportToWorldPoint(screenCenter);

if (Physics.Raycast(deepthRayOrigin, GetCastDirection(), out deepthHit))

castDeepth = deepthHit.distance;

Debug.DrawRay(deepthRayOrigin,fpsCam.transform.forward\*castDeepth, Color.green);

rayOrigin = fpsCam.ViewportToWorldPoint(screenCenter);

if (Physics.Raycast(rayOrigin, GetCastDirection(), out hit, castDeepth, 1 << LayerMask.NameToLayer(layerName)))

{ // TODO… }

## 3.4 武器系统

武器系统包括即时命中武器与投掷武器，这套系统是整个游戏中最能表达游戏内容的系统。

3.4.1 弹痕效果

对于FPS游戏而言，增加游戏表现力与提升射击手感非常重要。玩家控制的角色进行设计的时候应当有明确的视觉效果来提示玩家是否已经完成射击或者是否命中目标，弹痕效果是一个很明显的提示。玩家在完成射击后，所击中的墙面会产生弹痕来提升玩家的游戏体验。在目前市面上主流的游戏中这种效果是普遍存在的。

对弹痕效果的需求应当满足以下几点：

1. 即时产生弹痕
2. 不同武器产生不同弹痕

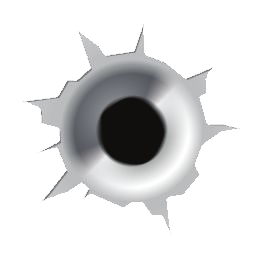
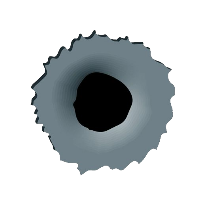


图3.14 不同武器产生的不同弹痕贴图

为了节约内存与提升游戏性能，所有弹痕效果不会永久保留而会在一段时间后消失。弹痕原理是对需要产生弹痕的表面实时生成一张弹痕贴图，并使得这张帖图与整个墙面贴图进行贴图融合。这张贴图会在开始时存放在内存中。为了节省内存，这张贴图的大小会被限定在32\*32。

在《Project K》中，弹痕的产生是通过名为ShootMarkMaker的类来计算生成的，ShootMarkMaker类是挂载在需要产生弹痕的墙面上的一个实时执行的一个脚本。其核心算法步骤如下：

1. 获取到墙面的当前贴图与弹痕贴图
2. 通过textureCoord变量获取到需要重新计算的起始坐标
3. 对弹痕贴图每个像素值进行遍历，计算出每个像素与墙面融合之后的颜色，对应坐标(x, y)的计算公式如下

其中𝑆0x\*w指需要刷新的起始坐标的x值，𝑆0y \*h指的是需要刷新的起始坐标的y值。w、h指的是墙面贴图的宽和高，由于(𝑆0x, 𝑆0y)在系统中被规范化，范围属于[0,1]，故需要与w、h相乘得到最终结果。i、j指的是遍历弹痕贴图时的迭代变量值。

1. 对墙面对应位置的像素颜色进行刷新
2. 所有需要计算的像素都刷新完成后，更新整张墙面贴图

其中重新计算的像素具体算法代码如下：

for (int i = 0; i < bulletWidth; i++)

{

for (int j = 0; j < bulletHeight; j++)

{

float w = uv.x \* wallWidth - bulletWidth / 2 + i;

float h = uv.y \* wallHeight - bulletHeight / 2 + j;

Color wallColor = newWallTexture.GetPixel((int)w, (int)h);

Color bulletColor = gun.bullueTexture.GetPixel(i, j);

newWallTexture.SetPixel((int)w, (int)h, wallColor \* bulletColor);

}

}

其中wallColor \* bulletColor即为弹痕贴图与墙面贴图色彩进行融合操作后的颜色，值得注意的是，Unity中的颜色分为(r,g,b,a)四个分量，如果贴图采用透明背景时，a=0，此时融合操作会产生(r,g,b,0)的结果，会导致墙面变为黑色，因为墙面贴图渲染模式一般是不支持透明效果的。弹痕效果在《Project K》中实现后效果如图3.15所示。



图3.15弹痕效果最终效果

弹痕的存在不是永久的，在墙面上存在一定时间后，弹痕会消失，由于弹痕不是一次性在同一时间内产生的，故一定会有一个消失的先后顺序，为了按照“先出现的先消失”的步骤进行，代码中使用一个队列对每次产生的弹痕进行存储，使用Unity内置的延时函数Invoke在规定时间后将弹痕消去。为了保存产生弹痕前墙面的颜色，在初始化时会将墙面贴图作为Texture2D方式保存。ShootMarkMaker类的成员变量如图3.16所示。

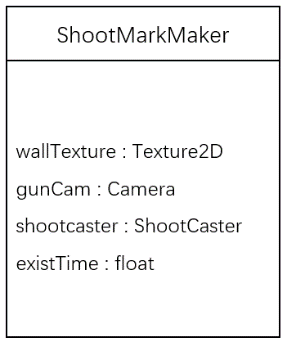
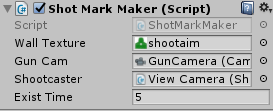
 

图3.16 ShootMarkMaker组件的成员变量与Inspector上显示效果

3.4.2 弹道效果

对于FPS游戏中的枪械来说，如果一个可以连发的枪械武器，在持续开枪中没有发生弹道散射与弹道偏移，会造成射击手感的“不适应”与游戏与现实的完全脱轨。虽然部分游戏是对未来武器的设想与对现代武器的改造，如在《守望先锋》这款游戏中，部分角色使用的武器并没有现实中的原型可以参考。但是为了增加游戏乐趣与增加游戏真实程度，一般游戏会设定弹道效果。

弹道散射与偏移一般而言被称为枪械的“后坐力”。理论上来说，一款武器的后坐力在游戏中无法完整真实的实现，但是可以通过设定弹道偏移的方式来实现仿真效果。



图3.17 守望先锋中的未来武器

由于FPS游戏视角的独特性，玩家所瞄准的焦点永远处于屏幕的正中心，这导致了玩家所瞄准的方向与武器发射子弹的方向并不一致，除非采用了机械瞄准，即“瞄准镜瞄准”[12]。所以在计算弹道的时候有两个方向，但玩家瞄准焦点是不变的，计算的方向是枪口发射的子弹方向。如图3.18所示，绿色线代表玩家瞄准的方向，而紫色线代表子弹发射方向也就是实际瞄准方向。

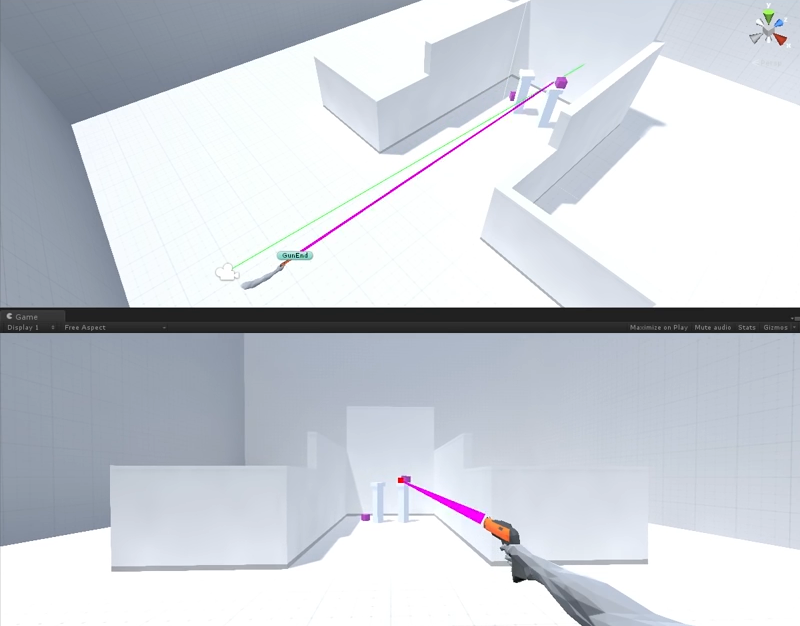


图3.18 瞄准方向

主流游戏中的弹道一般分为固定弹道与随机弹道两种，《Project K》中，武器的弹道设定是属于固定弹道。固定弹道的特点是每种武器的弹道都是事先确定的，也就是特定的武器每次进行射击时都会按照程序固定好的弹道曲线来计算弹道，这样会使得每次射击都会产生相同的偏移。由于这种偏移是固定的，在大量的练习之后，玩家会逐渐熟悉常用武器的弹道，从而实现“压枪”的操作。

固定弹道的原理是事先在程序中实现弹道的偏移位置，从而计算出每发子弹的偏移量。为达到此效果，《Project K》中定义了两个结构体GunValues和GunProperties。GunValues结构体定义了枪械的基本数值，如是否属于全自动武器、射击速率、伤害量、弹夹容量、换弹速度等。GunProperties结构体则定义了枪械的各种属性。其中包括了弹道系统的核心属性偏移曲线。

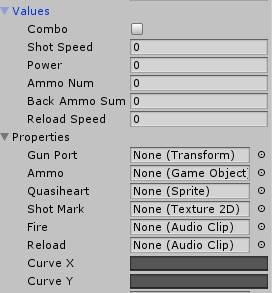
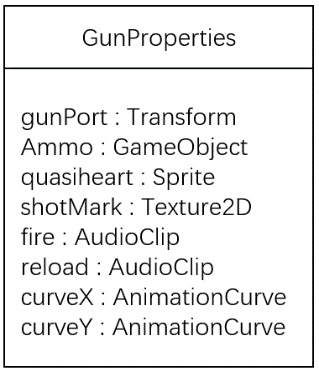
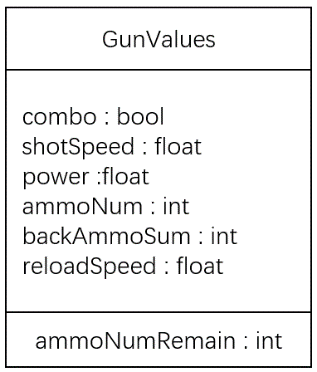


图3.19 GunValues结构体与GunProperties结构体与Inspector上显示效果

其中AnimationCurve是Unity内置的结构，其本质是存储了一系列能够随时间计算的关键帧。从另一个方面来说可以理解为存储了一个一元二次函数。Unity编辑器提供了Curve可视化编辑器，当然也可以通过代码动态改变。

《Project K》中的弹道计算是通过两条曲线叠加计算完成的，这是由于弹道偏移考虑了两个方向（即水平x轴方向与竖直y轴方向）导致的。核心Shoot方法返回值是IEnumerator，使用IEnumerator的方法可以用于协程。协程是Unity中实现在同一线程中无需考虑同步与锁的一种可挂起等待执行的解决方案。

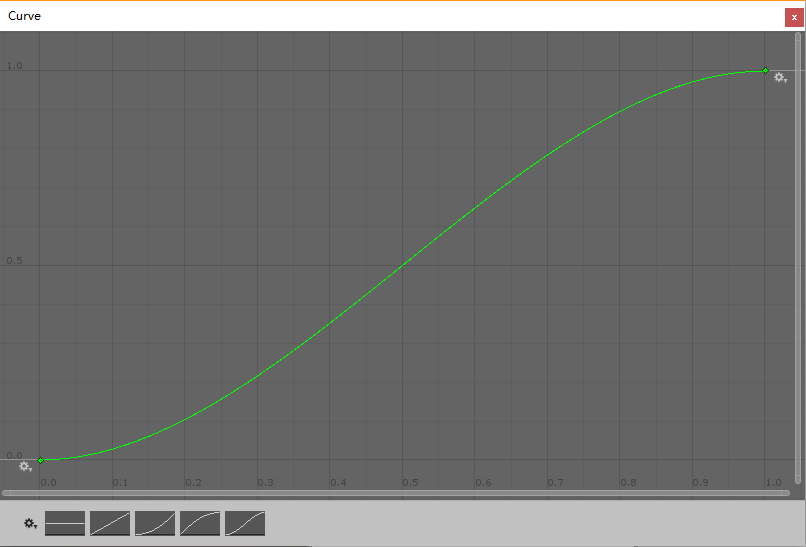


图3.20 Unity内置的Curve编辑器

核心计算方法如下：

1. 计算出当前发射子弹的弹道进度，即计算出x轴的数值
2. 分别计算出x、y轴的偏移量Mx、My
3. 以当前枪口位置为基础添加偏移量Mx、My
4. 得到最终结果（Mx，My），x轴数值计算公式如下：

其中tp指弹道进度，具体即一次射击过程中的第几颗需要计算弹道的子弹，as指的是弹夹数量。

（Mx，My）的具体计算代码如下：

private Vector3 GetCastDirection()

{

if (gun.values.combo)

{

trajectoryProcess = (gun.values.ammoNum / 1.0f - tarjectoryCount / 1.0f) / gun.values.ammoNum / 1.0f;

float x = gun.properties.curveX.Evaluate(trajectoryProcess);

float y = gun.properties.curveY.Evaluate(trajectoryProcess);

Vector3 castDirection = fpsCam.transform.forward;

castDirection.x += x;

castDirection.y += y;

return castDirection;

}

else return fpsCam.transform.forward;

}

《Project K》中的冲锋枪Uzi-Gold弹道曲线如图3.21所示，其最终弹道效果如图3.22所示。目前市面上主流FPS游戏中采用固定弹道的游戏有《Counter-Strike: Global Offensive》、《Tom Clancy's Rainbow Six® Siege》等。

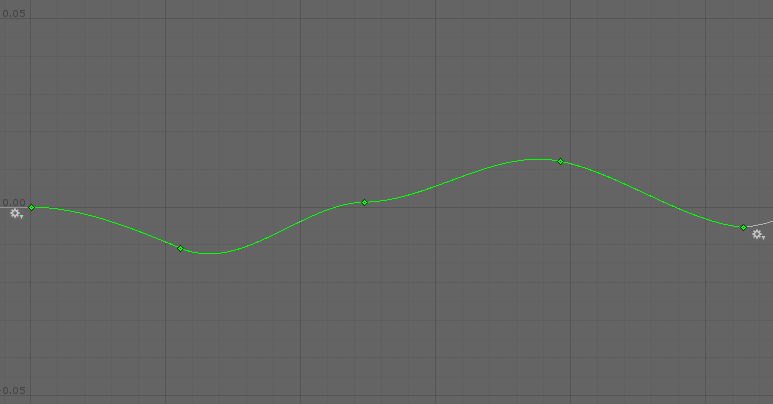
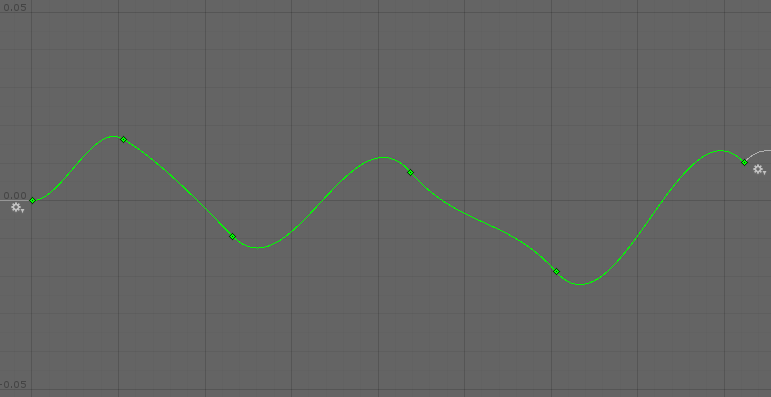
 

图3.21 游戏中的弹道曲线（左X偏移曲线，右Y偏移曲线）

另外一种方式就是随机弹道，随机弹道指的是事先没有设定好的偏移量，而只需要确定一个散射范围r，在射击过程中，每发子弹的偏移量会在[0,r]内随机取值。使用随机弹道会使得游戏更加真实，但是就无法对枪械进行逆向操作（也即是压枪操作）。目前市面上的主流FPS游戏中采用随机弹道处理方式的有《Overwatch》、《Call of duty系列》等等。



图3.22 Uzi-Gold弹道轨迹图（相隔13米）

其核心代码与固定弹道非常相似，只是偏移量x，y变成了电脑生成的随机数。代码如下：

private Vector3 GetCastDirection()

{

if (gun.values.combo)

{

float x = Random.range(0,r);

float y = Random.range(0,r);

Vector3 castDirection = fpsCam.transform.forward;

castDirection.x += x;

castDirection.y += y;

return castDirection;

}

else return fpsCam.transform.forward;

}

3.4.3 群体范围伤害

群体范围伤害指的是一种武器或技能不是对单一敌人造成伤害，而是对某一个范围内的目标都会触发效果。《Project K》中，即时命中武器只能造成单一目标伤害，而范围武器可以造成群体范围伤害，也就是俗称的AOE(Area of Effect)。

|  |  |
| --- | --- |
| 方法原型 | 检测区域 |
| public static bool BoxCast(Vector3 center, Vector3 halfExtents, Vector3 direction, Quaternion orientation,float maxDistance,int layerMask,QueryTriggerInteraction queryTriggerInteraction); | 盒状区域 |
| public static RaycastHit[] CapsuleCastAll(Vector3 point1, Vector3 point2, float radius, Vector3 direction, float maxDistance, int layermask, QueryTriggerInteraction queryTriggerInteraction); | 胶囊状区域 |
| public static RaycastHit[] SphereCastAll(Vector3 origin, float radius, Vector3 direction, float maxDistance, int layerMask, QueryTriggerInteraction queryTriggerInteraction); | 球形区域 |
| public static RaycastHit[] RaycastAll(Ray ray, float maxDistance, int layerMask, QueryTriggerInteraction queryTriggerInteraction); | 射线 |

表3.1 部分不同的检测区域函数原型

群体范围伤害的核心思路也是通过射线检测来实现的，只不过对于AOE而言，一条射线已经不能满足需要了。所以一般是多条射线进行检测，同时也不再像即时命中射击时那样需要射线深度探针，而是对物体进行穿透检测，返回一个命中的物体碰撞体数组。

在Unity中实现这样的效果，需要使用一个特殊的API，也即Physics.SphereCastAll，此API会返回RaycastHit数组。通过RaycastHit中的碰撞体成员变量即可获取对球形范围内检测到的物体引用。

这种检测方案通常用于火箭弹、导弹、炸弹、榴弹等的设计与实现，一般来说，设定这些武器的关键参数为检测区域、检测半径、威力与是否可以被遮挡等。其中是否可以被遮挡可以用物理检测层级掩码来表明[13]。

当然，有些时候不一定需要一个球形的检测区域，应当根据实际情况运用不同的检测范围，Unity3D引擎采用的物理系统内封装了许多检测函数[14]，表3.1列出了一些不同区域的射线检测API原型。

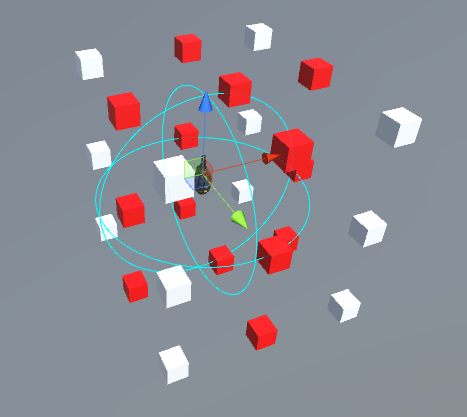


图3.23 手雷的范围检测效果（Debug模式）

如图3.23，手雷的检测范围为球形，在Debug模式下，当场景中有满足检测的物体时，物体会被渲染成红色（默认为白色）。其中青色线框代表手雷爆炸检测范围。

## 3.5 工程整体结构

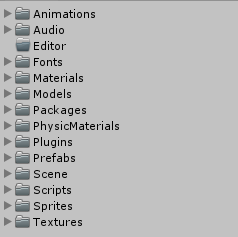


图3.24 工程目录结构图

《Project K》采用Unity2017.3.1f1 Personal (64 bit)版本开发，脚本采用C#语言编写，其工程目录遵循Unity一般规则，将同类资源或脚本放在同一文件夹中，不同资源放在不同子文件夹中。其文件目录结构图如3.24所示。

其中各个文件夹功能如表3.2所示。

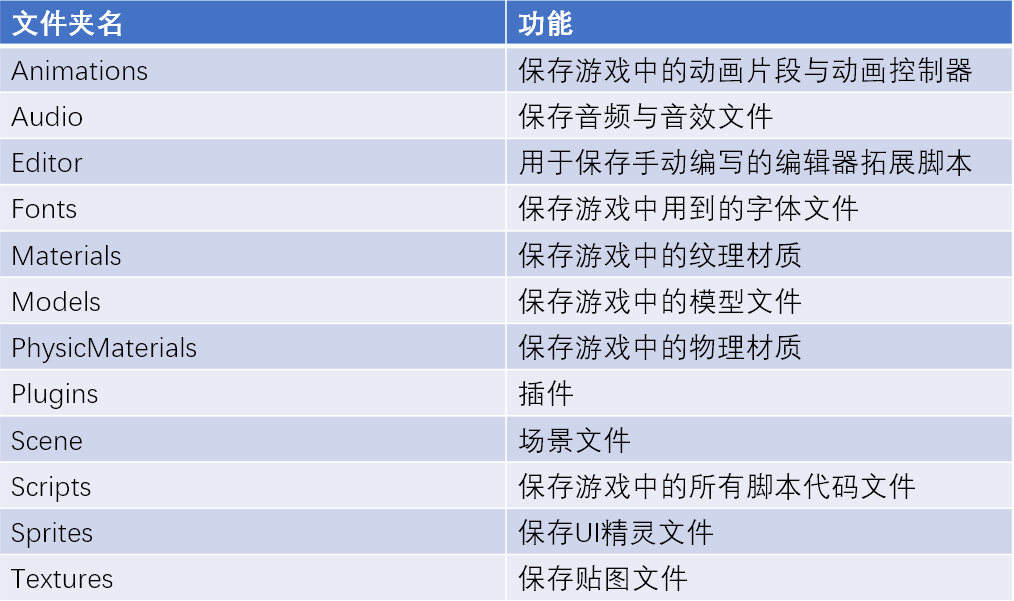


表3.2 各个文件夹功能

游戏中的全部脚本较多，负责整个游戏的逻辑与相关计算处理。其Scripts文件夹下的部分内容结构如图3.25所示。各个子文件下的脚本分别处理各个文件夹所命名的模块功能，

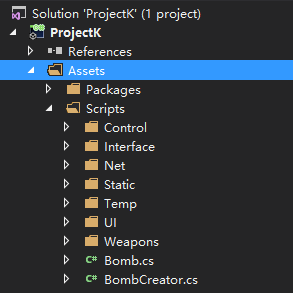


图3.25 Scripts目录结构

# 第4章 系统测试与性能分析

4.1整体结构

本项目采用Unity推荐的模块化编程思路，将各个功能模块进行单独编写后进行耦合，使得游戏系统具有相当高的可靠性以及可拓展性。游戏系统整体结构图如图4.1所示。



图4.1 整体结构图

4.2功能测试

本部分对《Project K》所有功能进行展示与测试。

4.2.1 主界面

游戏的主界面并没有过多设计，只保留了游戏中的一个地图用作背景以及必要的UI界面。如图4.2所示。



图4.2 游戏主界面

其中“新建游戏”按钮负责在本地开放一个游戏房间对局域网开放。此时本地主机充当服务器。



图4.3创建游戏界面

创建游戏主要的工作是确定一个房间的网络参数，其网络参数主要已经由UI显示在创建面板上。用户确定好参数后，点击创建按钮即可创建一个局域网游戏房间等待其他用户登陆进行游戏。

4.2.2 加入游戏

用户在主界面选择加入游戏后，会显示当前网络中已经存在的游戏。

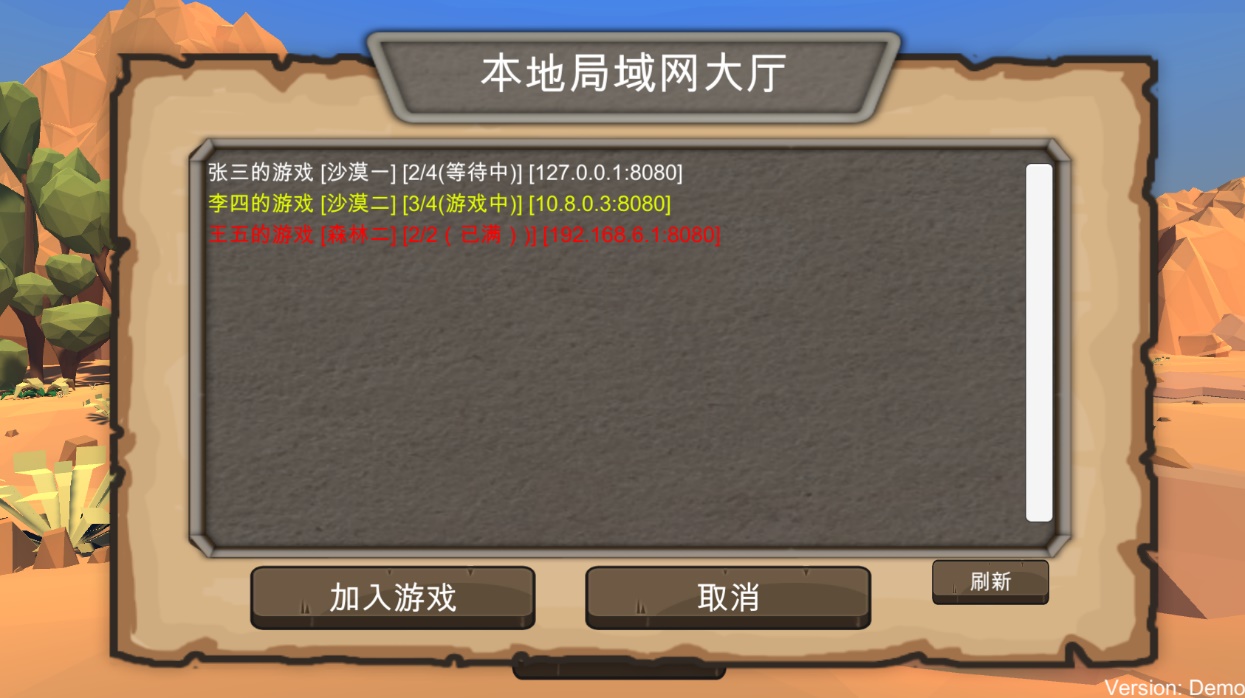


图4.4 当前存在的游戏

其中界面中心黑色部分会用于显示当前网络中已经存在的游戏，如果游戏房间还在等待玩家未开始游戏，则会显示白色字体，并给出当前房间内已经存在的游戏。如果游戏已经开始，则会以黄色字体显示房间信息，但是此时玩家仍然可以选择加入游戏。如果游戏房间已经人满，则会以红色字体显示房间信息，此时不论该房间内是否开始游戏，玩家都不可加入。玩家如果不是自行创建的游戏房间，加入房间后一律充当客户端。

玩家加入游戏后也需要进行必要的设定才能开始战斗，其中最重要的就是玩家需要指定自己在游戏中的昵称，游戏昵称是玩家唯一标识符，此昵称在同一房间内不可重复。



图4.5 加入游戏设置

4.2.3 游戏设置

游戏设置面板中对游戏控制参数进行调节，可调节的参数如图4.6所示。其中游戏选项卡中的分辨率可对游戏分辨率进行调节，全屏按钮可以确定是否全屏，画面选项调节游戏画面的渲染等级。控制选项卡中的灵敏度调节游戏中武器视角灵敏度，FOV选项调节游戏摄像机视野。



图4.6 设置界面

4.2.4 选择武器

游戏中目前总共提供三种武器，分别称为“M16”突击步枪，“Uzi-Gold”冲锋枪和“小手枪”。这三种武器各有特点。

M16突击步枪射速较慢，但是单发伤害较高，可以连发，每个弹夹拥有30发子弹。弹道偏移较大；Uzi-Gold冲锋枪射速很快，单发伤害低，可以连发，每个弹夹拥有40发子弹，换弹时间为三把武器中最快，弹道偏移较小，是一把很平衡的武器；小手枪不可连发，只能单次激发，射速由玩家手速决定，单发伤害非常高，每个弹夹拥有7发子弹。

游戏中所有武器均没有爆头双倍伤害效果。

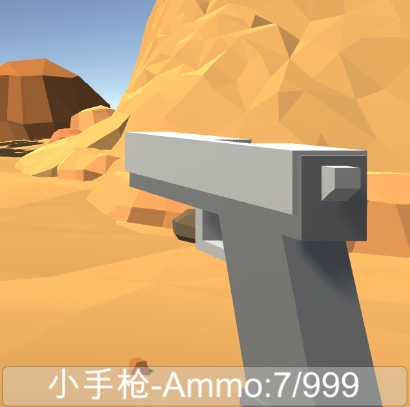


图4.7 游戏中的不同武器

4.2.5 得分与胜利

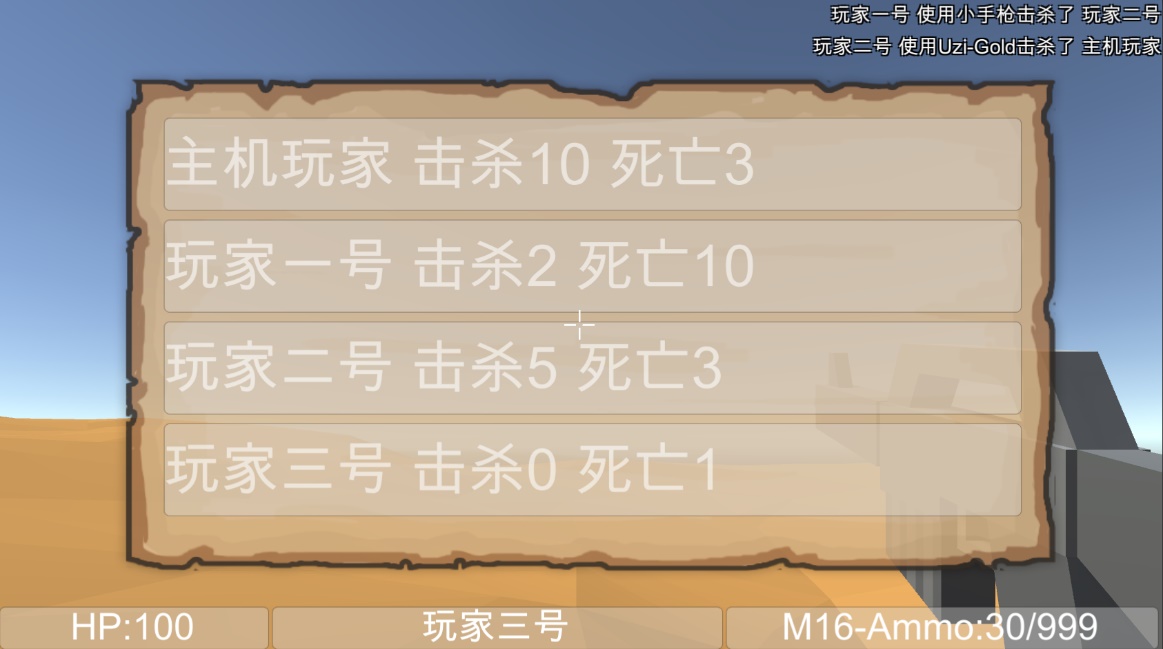


图4.8 实时积分榜

游戏目前只有一种模式，即个人竞技模式，在该模式下，玩家与其余所有玩家均为敌对关系，每名玩家拥有100点初始生命值，玩家通过自身的操作与枪械射击对其他玩家进行攻击。当某玩家生命值归零时，由最后一名造成伤害的玩家获得1分。最先获得20分的玩家赢得比赛。

玩家在游戏过程中可以通过Tab键来实时查看得分榜，得分榜如图4.8所示。其中击杀信息会实时更新在游戏画面右上角，游戏画面下方为三段式显示玩家信息，左下方为当前生命值，中部为玩家昵称，右侧为玩家当前使用的武器以及剩余弹药量。

4.3性能分析

本游戏项目的性能要求并不高，由于采用了低面数3D模型渲染，对CPU与GPU性能消耗并不多[15]。图4.9显示了主界面的性能数据。由下图可以看出，游戏在主界面场景的时候能够稳定60fps，开发电脑硬件配置为CPUi7-6700HQ，GPU NIVDIA GTX1060 3G，内存为2\*8双通道2400Hz，显示器刷新率60Hz。

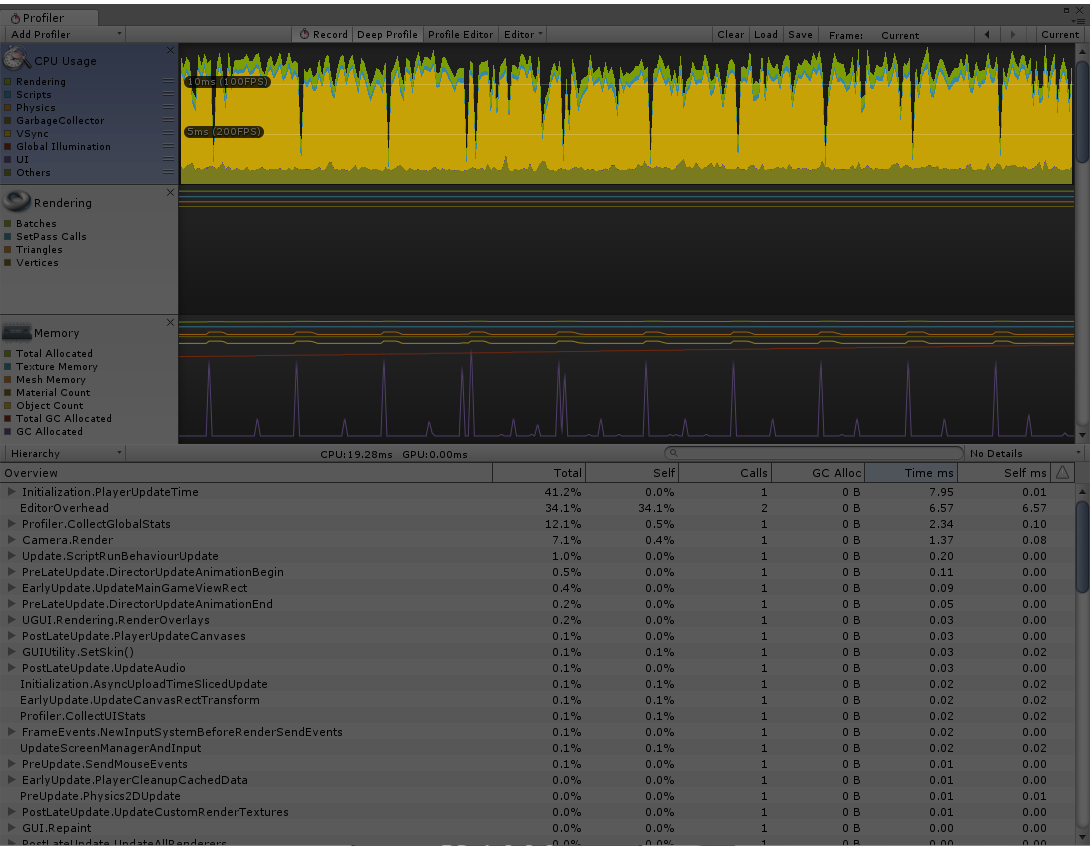


图4.9 主界面场景性能分析

在游戏场景中，性能消耗也并不高，在一个沙漠一地图，拥有两个玩家的游戏房间内游戏时，游戏可以稳定在120fps以上，主线程执行一次时间为7ms，内存占用在100MB上下。这样的数据对于一个局域网3D联机游戏而言已经是非常不错的了。

# 结 论

本文研究了局域网内联机第一人称射击游戏《Project K》的设计与实现，其最终实现了包括UI引导、地图池、枪械系统（包括弹道弹痕以及AOE武器）、FPS控制器、局域网联机与匹配、状态与动画同步等功能。而本文的重点在于将这些系统全部一一实现后进行统一为一个整体。

基于上述思路，本项目底层采用成熟的商业化引擎Unity来实现，而开发者只需要专注于游戏功能与逻辑以及控制系统的编写与优化。最终实现了一个完整的局域网联机游戏《Project K》。本游戏亮点与优点主要体现在几个方面：首先，游戏内容算是比较丰富，拥有四张不同的地图与三种不同类型的武器，同时游戏可以实现多人联机，大大增强了游戏乐趣；其次，虽然游戏内容不少，但是游戏的安装包很小，仅仅只有100M左右，玩家只需要花费很少的时间就可以下载完成整个游戏，而且游戏打包为绿色版，无需安装即可进行游戏；此外，游戏的性能消耗并不高，能够确保低配置电脑渲染出整个游戏场景仍然保持稳定60帧的流畅体验。而且游戏内部设计较好，方便维护与更新，具有很大的开发前景与游戏乐趣。本系统也能为其他类似游戏提供参考。