**分类号：TP311.5 U D C：D10621-408-(2017)1905-0**

**密 级：公 开 编 号：2013081199**

**成都信息工程大学**

**学位论文**

**基于Unity3D的手机游戏设计与开发**

**——第一人称射击求生游戏**

|  |  |
| --- | --- |
| **论文作者姓名：** | **陈宗林** |
| **申请学位专业：** | **软件工程** |
| **申请学位类别：** | **工学学士** |
| **指导教师姓名（职称）：** | **铁菊红（讲师）** |
| **论文提交日期：** | **2016年12月4日** |

**基于Unity3D的PC游戏设计与开发**

**——第一人称射击求生游戏**

**摘要**

随着当前家庭电脑的普及，越来越多的年轻人使用电脑游戏进行放松。现在电脑游戏的种类十分的丰富，从经典简单的扫雷等休闲类到大型的网络游戏。

互动游戏和大型的故事单机游戏的角色扮演类，玩家可以自由的选择自己喜欢的游戏类型。经典的FPS(First-person Shooter)第一人称视角射击游戏是以玩家的主观视角来进行射击游戏。玩家们不再像别的游戏一样操纵屏幕中的虚拟人物来进行游戏，而是身临其境的体验游戏带来的视觉冲击，大大增强了游戏的主动性和真实感。

本游戏《第一人称射击求生游戏》基于Unity3D游戏引擎开发，以VisualStudio2015作为开发工具，主要使用C#语言开发实现。玩家可以选择和朋友一起相互控制角色竞争或相互合作生存，也可以在单人模式下，在结合了迷宫元素的场景中控制角色破坏怪物，培养自己的属性最后逃出迷宫。游戏设计上以随机为主题让玩家在相同的规则下有不同的游戏体验，提高游戏的持久可玩性。目前本游戏已经能够在电脑端正常运行。菜单导航清晰，界面清新明了用户在试听，操作等方面体验真实。

**关键词：**PC游戏；Unity3D; 第一人称射击类；局域网联机；C#；迷宫随机生成算法；迷宫A星寻路

**Design and Development of Mobile Game Based on Unity3D**

* **First Person Shooter Survival Game**

**Abstract**

With the popularity of the home computer, more and more young people to use computer games to relax.Now the kinds of computer games is very rich, from the classical simple minesweeping leisure class to large network game, etc.  
Interactive games and large story single-player role-playing game, players have the freedom to choose their own like the type of game.Classic FPS (First person Shooter) first-person Shooter game is the player's subjective perspective for shooting game.Players no longer like other games virtual characters in manipulating the screen to make games, but the immersive experience games bring visual impact, greatly enhance the initiative and sense of reality of the game.  
This game the first-person shooter survival game based on Unity3D game engine development, to VisualStudio2015 as a development tool, mainly using c # language development.Players can choose mutual control role play with my friends or survival, mutual cooperation and can also be in single mode, control role in combining the maze scene elements destroy the monster, cultivating their own properties and then the maze.In a random theme on game design for players under the same rules have a different game experience, improve the sustained playability of the game.Now the game has been able to in the normal operation of a computer terminal.Menu navigation clear, fresh and clear user interface in the audition, experience the real operation, etc.

**Key words:** PC game;Unity3D.The first person shooter;LAN;C #;Maze randomly generated algorithm;A \* pathfinding labyrinth

目录

[1 绪论 1](#_Toc468641834)

[1.1 课题背景 1](#_Toc468641835)

[1.2 单机游戏现状研究 1](#_Toc468641836)

[1.3 课题主要工作 1](#_Toc468641837)

[1.3.1 工作任务 1](#_Toc468641838)

[1.3.2 工作要求 2](#_Toc468641839)

[2 Unity3D游戏引擎相关技术研究 3](#_Toc468641840)

[2.1 、Unity编辑器特点 3](#_Toc468641841)

[2.2 Unity3D游戏引擎特点 3](#_Toc468641842)

[3 第一人称射击求生游戏系统设计 10](#_Toc468641843)

[3.1 游戏系统设计 10](#_Toc468641844)

[3.1.1 游戏结构设计 10](#_Toc468641845)

[3.1.2 游戏流程设计 10](#_Toc468641846)

[3.1.3 功能行为与控制系统设计 12](#_Toc468641847)

[3.2 游戏框架设计 13](#_Toc468641848)

[3.2.1 设计要求 13](#_Toc468641849)

[3.2.2 游戏框架介绍 13](#_Toc468641850)

[3.3 游戏界面设计 23](#_Toc468641851)

[3.3.1 画面设计 23](#_Toc468641852)

[3.3.2 主要界面设计 23](#_Toc468641853)

[3.4 数值设计 26](#_Toc468641854)

[3.4.1 单人单机模式下所需数值 26](#_Toc468641855)

[3.4.2 多人联机竞技模式下所需数值 27](#_Toc468641856)

[3.4.3 多人联机生存模式下所需数值 27](#_Toc468641857)

[3.5 资源素材 27](#_Toc468641858)

[3.5.1 模型资源 27](#_Toc468641859)

[3.5.2 声音资源 30](#_Toc468641860)

[4 第一人称射击求生游戏详细设计与实现 32](#_Toc468641861)

[4.1 设计游戏制作过程 32](#_Toc468641862)

[4.2 Prefab的简单说明 32](#_Toc468641863)

[4.3 游戏实体对象主要类类图 32](#_Toc468641864)

[4.3.1 游戏实体对象的怪物类图 32](#_Toc468641865)

[4.3.2 游戏实体对象的玩家类图 33](#_Toc468641866)

[4.4 主角的移动及开火控制 34](#_Toc468641867)

[4.4.1 主角的移动控制 34](#_Toc468641868)

[4.4.2 主角的枪及开火控制 35](#_Toc468641869)

[4.5 敌人的派生继承 36](#_Toc468641870)

[4.6 子弹功能 37](#_Toc468641871)

[4.7 奖励物品 38](#_Toc468641872)

[4.8 迷宫生成与自动寻路 38](#_Toc468641873)

[4.8.1 迷宫生成算法 38](#_Toc468641874)

[4.9 Unity旧版系统局域网联机 41](#_Toc468641875)

[4.9.1 旧版本的网络通信系统 41](#_Toc468641876)

[4.9.2 数据动画的同步的实现 41](#_Toc468641877)

[4.10 Unity新版系统局域网联机 42](#_Toc468641878)

[4.10.1 新本版的网络通信系统 42](#_Toc468641879)

[4.10.2 服务端和客户端的区分 45](#_Toc468641880)

[4.10.3 怪物的产生及AI的寻路控制 45](#_Toc468641881)

[4.10.4 数据信息的同步 46](#_Toc468641882)

[4.10.5 死亡复活功能的实现 46](#_Toc468641883)

[4.11 关键功能代码及说明 46](#_Toc468641884)

[4.11.1 单人模式下的迷宫生成代码 46](#_Toc468641885)

[5 单人模式AI系游戏运行与测试 55](#_Toc468641886)

[5.1 测试环境 55](#_Toc468641887)

[5.2 单元测试 55](#_Toc468641888)

[5.3 功能测试 55](#_Toc468641889)

[**结语** 58](#_Toc468641890)

[**参考文献** 58](#_Toc468641891)

[**致谢** 59](#_Toc468641892)

[**声明** 60](#_Toc468641893)

图目录

[图 2.1 Unity3D编辑器 3](#_Toc468643051)

[图 2.2 Unity子系统相关图 3](#_Toc468643052)

[图 2.3 Unity层次的地图 5](#_Toc468643053)

[图 2.4 GameInstantiator in the Inspector 6](#_Toc468643054)

[图 2.5 GameController in the Hierarchy 6](#_Toc468643055)

[图 2.6 MVC模式的体系结构表示 7](#_Toc468643056)

[图 2.7抽象工厂模式 8](#_Toc468643057)

[图 2.8输入子系统结构摘要 9](#_Toc468643058)

[图 2.9组成部分的图中的抽象工厂模式 9](#_Toc468643059)

[图 3.1菜单结构 10](#_Toc468643060)

[图 3.2单人单机游戏模式流程图 11](#_Toc468643061)

[图 3.3多人竞技模式游戏流程图 11](#_Toc468643062)

[图 3.4多人合作生存模式流程图 12](#_Toc468643063)

[图 3.5主菜单抽象结构类图 14](#_Toc468643064)

[图 3.6单人单机抽象结构类图 14](#_Toc468643065)

[图 3.7玩家角色抽象结构类图 15](#_Toc468643066)

[图 3.8敌人抽象结构类图 16](#_Toc468643067)

[图 3.9场景抽象结构类图 16](#_Toc468643068)

[图 3.10 UI控制抽象结构类图 17](#_Toc468643069)

[图 3.11子弹抽象结构类图 17](#_Toc468643070)

[图 3.12奖励物品抽象结构类图 17](#_Toc468643071)

[图 3.13多人竞技抽象结构类图 18](#_Toc468643072)

[图 3.14玩家角色抽象结构类图 18](#_Toc468643073)

[图 3.15客户端抽象结构类图 18](#_Toc468643074)

[图 3.16服务器抽象结构类图 19](#_Toc468643075)

[图 3.17子弹抽象结构类图 19](#_Toc468643076)

[图 3.18场景抽象结构类图 19](#_Toc468643077)

[图 3.19 UI控制抽象结构类图 20](#_Toc468643078)

[图 3.20多人合作生存抽象结构类图 20](#_Toc468643079)

[图 3.21玩家角色类图 21](#_Toc468643080)

[图 3.22网络联机抽象结构类图 22](#_Toc468643081)

[图 3.23场景抽象结构类图 22](#_Toc468643082)

[图 3.24 UI抽象结构类图 22](#_Toc468643083)

[图 3.25敌人抽象结构类图 23](#_Toc468643084)

[图 3.26主菜单UI设计选项图 23](#_Toc468643085)

[图 3.27次级的联网模式下的UI设计 24](#_Toc468643086)

[图 3.28单人单机游戏效果图 24](#_Toc468643087)

[图 3.29多人竞技模式效果图 25](#_Toc468643088)

[图 3.30多人合作求生类效果图 25](#_Toc468643089)

[图 4.1怪物类图 32](#_Toc468643090)

[图 4.2玩家类图 33](#_Toc468643091)

[图 4.3主角人物模型结构展示图 34](#_Toc468643092)

[图 4.4玩家开火效果图 35](#_Toc468643093)

[图 4.5怪物模型全体图 36](#_Toc468643094)

[图 4.6 ZombEnemy模型展示图 36](#_Toc468643095)

[图 4.7 SoulEnemy模型展示图 37](#_Toc468643096)

[图 4.8 Shooter的攻击效果图 37](#_Toc468643097)

[图 4.9子弹检测效果图 37](#_Toc468643098)

[图 4.10奖励物品效果展示图 38](#_Toc468643099)

[图 4.11 Prim算法说明图 39](#_Toc468643100)

[图 4.12本游戏的迷宫生成算法图 40](#_Toc468643101)

[图 4.13 Unity旧版本的通信系统实现方式图 42](#_Toc468643102)

[图 4.14 Network Manager的信息介绍图 43](#_Toc468643103)

[图 4.15需要动态生成物体放入图 43](#_Toc468643104)

[图 4.16系统同步动画NerworkAnimator图 44](#_Toc468643105)

[图 4.17 Navigation的渲染场景的效果图 45](#_Toc468643106)

[图 4.18位置数据两种同步方式图 46](#_Toc468643107)

表目录

[表 3‑1功能行为与控制关系表 12](#_Toc468643178)

[表 3‑2 Enemy攻击距离伤害血量数值详情 26](#_Toc468643179)

[表 3‑3 Boss文件夹素材详情 27](#_Toc468643180)

[表 3‑4 Character文件夹素材详情 28](#_Toc468643181)

[表 3‑5 SoulMonster文件夹素材详情 28](#_Toc468643182)

[表 3‑6 Envriment文件夹素材详情 29](#_Toc468643183)

[表 3‑7 SoulShooter文件夹素材详情 29](#_Toc468643184)

[表 3‑8 Cartoon Sholdier文件夹素材详情 29](#_Toc468643185)

[表 3‑9 SimpleTownPack文件夹素材详情 30](#_Toc468643186)

[表 3‑10声音资源文件详情 30](#_Toc468643187)

[表 5‑1测试环境配置信息 55](#_Toc468643188)

[表 5‑2主菜单界面测试用例 55](#_Toc468643189)

[表 5‑3返回主菜单界面测试用例 56](#_Toc468643190)

[表 5‑4复活界面测试用例 56](#_Toc468643191)

[表 5‑5重新开始有界面测试用例 56](#_Toc468643192)

[表 5‑6游戏功能测试用例 57](#_Toc468643193)

# 绪论

## 课题背景

FPS作为电脑游戏最为经典的玩法之一的游戏类型，在此基础上成就了许多经典的游戏，第九类艺术。第一人称设计游戏最开始是用于军方模拟战场使用的，到后来发展到游戏中，在设计者的玄幻或真实的场景中给人真实的体验感。在当前的游戏需求一直占有强大可客户量。

迷宫作为经典的空间解密游戏，变为3D不失为一个良好的地形题材体验和玩法目的，走出迷宫。本游戏《第一人称射击求生游戏》基于Unity3D结合了射击类的玩法和迷宫特色。在经典上做出部分的创新，并且添加上良好的界面交互UI，达到了有趣体验新奇的需求。

## 单机游戏现状研究

单机游戏一直都是游戏市场上的主流，也是很多玩家喜欢的游戏方式，因为制作精美，不管是在剧情上，还是在效果上都要高于其他的游戏方式，所以受到许多玩家的青睐，而大多数玩家都是年轻的人群，就如QQ一样当初抓住了年轻人的市场未来年轻人长大，也就是未来的大市场。

其中我国单机游戏的发展阶段尚在逐步的成熟，单机游戏主要依赖的就是版权，在过去我国对于一些游戏产品的版权的意识并没有十分的强烈，使得很多的开发商失去了对于这类游戏开发的动力。最终导致我国单机游戏自主开发比较少，但是市场十分的庞大的。

但是现在无论是腾讯，还是网易都有关注到游戏市场的潜力，在鼓励着提供着做游戏的人员的福利和在线免费课堂等。加上中国的玩家以为文化的原因心中都跟偏向的支持着我国游戏界的自主研发的崛起。

## 课题主要工作

### 工作任务

本任务主要是开发出一款第一人称的设计游戏，游戏分为三个模式：单机单人称求生，多人竞技对战模式，多人合作生存模式。

单机单人求生模式下要点：

（1） 场景（迷宫）、怪物类型地点、奖励物品类型效果地点都为随机生成。

（2） 在一个回合中记录并成长玩家角色的属性（血量，防御，攻击力）。

（3） 子弹效果和射击瞄准红外线。

多人竞技对战模式下要点：

（1） 实现局域网的下的服务器和客户端角色控制，动画的同步。

（2） 实现控制角色的伤害计算并判断游戏的结束及胜利者。

多人合作生存模式下要点：

（1） 实现局域网的下的服务器和客户端角色控制，动画的同步及怪物的信息同步（移动、攻击、死亡）。

（2） 实现客户端在服务器下的自由连接和断开操作。

（3） 实现死亡角色的复活功能。

系统下要点：

（1） 实现游戏退出功能。

（2） 实现进入各个游戏的交互UI菜单界面。

（3） 实现动态加载LOGO。

### 工作要求

在完成《第一人称射击求生游戏》游戏开发的过程当中，需按照软件工程的方法和规范，完成游戏策划、详细设计、编码以及测试。在各个工作当中认真完成其相应的文档的编写。毕业设计结束后，提交可运行，并且达到要求的源程序以及设计文档。

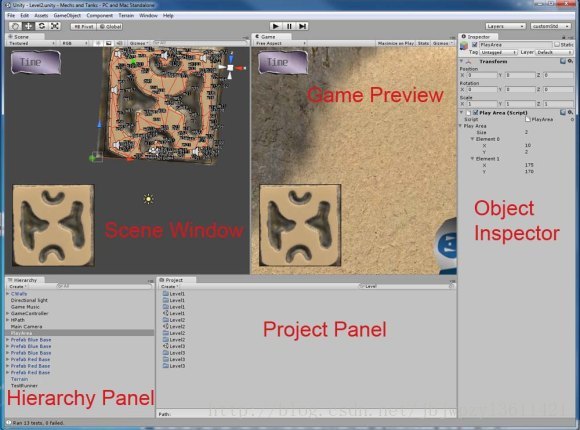
# Unity3D游戏引擎相关技术研究

## 、Unity编辑器特点

Unity3D的是一个屡获殊荣的工具，用于创建交互式3D应用程序在多个平台.Unity3D由游戏引擎和编辑器。该引擎包含的软件组件，在游戏的研究与开发中最常见的和经常性的任务。发动机所涵盖的主题包括声音，图形，物理和网络功能。该引擎支持C#，Boo，和JavaScript脚本编程。

Unity编辑窗口，作为脚本和其他组件，包含游戏场景设置和游戏的预览窗口（见图2.1）分层对象检查项目面板的集成开发环境。它还配备了几个多语言脚本编辑器和一个独特的预制装配系统。

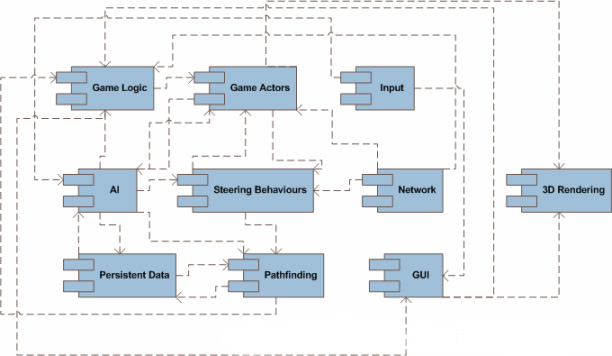
图 2.1 Unity3D编辑器



## Unity3D游戏引擎特点

游戏的建筑风格，是一个对象与数据capsules.The的下面的UML组件2.2图说明子系统及其关系网络。

图 2.2 Unity子系统相关图



**游戏逻辑**

此模块管理当前玩家和AI配置倒计时timerand当前的游戏状态（暂停，等待网络回复）。

**AI** 人工智能(Artificial Intelligence)

AI模块包含背后的逻辑单元，组和球员AI.The单位的AI寻路或障碍物避免使用不同的转向行为控制单元的状态。组AI管理组的行为和活动，如组寻路。更高的层次上管理播放机的所有组由播放器模块。  
  
人工智能机器学习保存和加载它的数据使用的持久性数据模块的接口。

**Persistant data持久性数据**

   此模块是负责数据之间不同的游戏sessions.Among其他应可用于保存和加载，存储查找表和图寻路模块和管理学习AI的accumulateddata的机器。

**Game actors游戏参与者**

  游戏参与者在游戏中的地形，单位或建筑物。他们的3D模型获得通过Unity3D的渲染管线的可视化。每场比赛的演员拥有AI模块，控制它的行为。

**Steering behaviours指导行为**

  指导行为的计算力量，影响如何以及如何快速自主游戏代理能动，应该可以用于避障，人流或简单的寻找任务。

**Pathfinding寻路**

  这模块负责创建一个pathgrid，障碍物信息收集和提供各种寻路请求aninterface的。为了获得更好的性能的一些信息保存到从磁盘中加载。

**Input输入**

此模块跟踪用户的输入，对其进行处理，并生成反馈。

**Network网络**

  网络模块是负责所有游戏演员的状态管理是保持比赛状态，在两台机器上都保持一致，以避免抖动网络单元运动网络game.Another责任。

**GUI**

 图形用户界面（GUI）显示所有按钮，菜单，在小地图和倒数计时器。它也负责为这些元素的功能和交互依赖与用于此目的的游戏的逻辑模块。

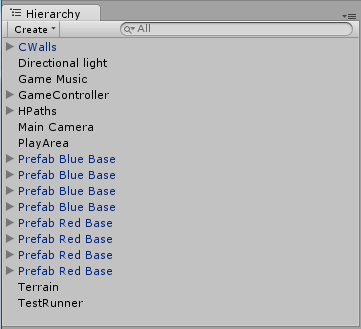
**3D渲染**

  该模块主要管理Unity3D的。场景的主摄像头确定需要渲染的对象，并把它们发送通过渲染管线。Unity3D的封装最渲染的细节，而且还提供了通过像素和顶点着色器的访问。

**Unity场景设置**

  在游戏中的每一个图表示由Unity3D的场景。下面是一个典型的场景设置在Unity层次（一）和（二）在现场窗口看起来像2.3图：

图 2.3 Unity层次的地图



**CWalls**

  这个对象包含自定义绘制墙节点和墙壁边缘。另一种是使用自定义绘制墙壁到calculatethem取决于地图的几何。

**Directional light（定向光）**

此灯仅用于计算地形光照贴图。关闭之后，由于性能的原因。

**Game Music（游戏音乐）**

持有游戏的主要音乐和播放现场启动。

**GameController(游戏控制器)**

GameController游戏物体持有并管理所有的游戏对象，管理游戏的逻辑。它包括以下对象：

**CursorController（光标控制器）：**

管理光标的外观和背后的逻辑。

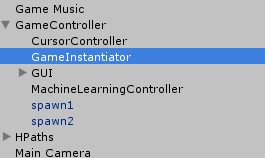
**GameInstantiator(游戏实例化)：**

   这个重要的游戏对象是负责实例化其他对象需要创建非特异性顺序。

图 2.4 GameInstantiator in the Inspector



图 2.5 GameController in the Hierarchy



GameInstantiator持有的的地图，PathCreator路径创建和管理障碍，管理球员配置和设置，是用来处理用户输入的的InputControl游戏物体，游戏场游戏物体和参考玩家游戏物体上的建筑物referenes定义了可播放的区域的地图。

 它还包含了玩家的重生点和自定义路径和墙壁的引用。

**GUI**

拥有地图的所有GUI对象。

**Machine Learning Controller**

   此游戏物体控制的所有功能，机器学习需要。

**spawn1, spawn2**

  实际玩家的重生点。重生点的标志是绿色立方体“场景视图。

**HPaths**

   自定义路径节点，在地图的边缘。自定义节点在场景编辑器中标记，并概述红线。

**Main Camera（主摄像机）**

   现场的主摄像头和音频监听。所有的3D声音是从相机的角度观察。

**PlayArea(游戏场)**

  定义实际可玩的地图区域。

Base Prefabs(基地预制)

  散落在地图上的建筑物。

**Terrain(地形)**

   Unity地形对象。

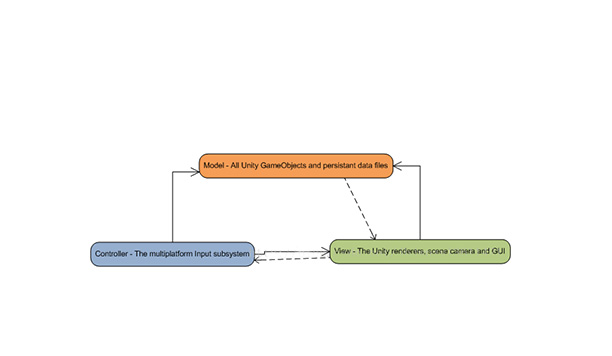
**TestRunner**

  一个物体，用于运行单元[**测试**](http://lib.csdn.net/base/softwaretest)与Unity3D的的单位testingframeworkSharpUnit的。

**MVC Pattern（MVC模式）**

  Unity引擎的设计鼓励MVC（模型 - 视图 -控制器）面向engineering.In的我的情况下，结构看起来像这样：

图 2.6 MVC模式的体系结构表示



模型包含了所有的游戏对象，组件和数据文件。游戏物体的渲染器和摄像机对象的访问。

视图呈现模型和主要管理Unity3D的引擎渲染。它需要accessthe持有模型的3D模型，纹理，材质和效果。它还具有什么输入选项的影响。

控制器接收用户输入并调用模型对象上的方法反应。这是在我的游戏中所表示的输入子系统。用户能影响与他的输入的视图。

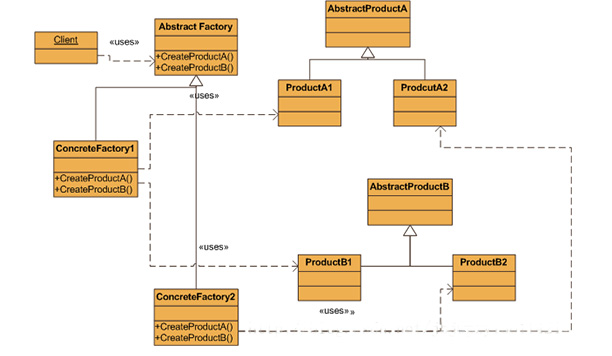
**Multiplatform Development(多平台发展)**

  Unity允许部署项目在不同的平台上有轻微的变化。演示和功能在很大程度上是保持取决于平台的capabilities.However，在某些领域，如在不同的设备上的输入机制存在重大分歧。

  抽象工厂设计模式应用于设计这些组件。

**The Abstract Factory Pattern抽象工厂模式**

图 2.7抽象工厂模式



抽象工厂模式（见图2.7）保护客户从不同的平台上实现相同的概念在不同的APC与平台是一家集的AbstractProduct类。这些类表示一个概念，是支持所有的抽象工厂platforms.An类声明创建单一产品的经营，ConcreteFactory类代表一个特定的平台。

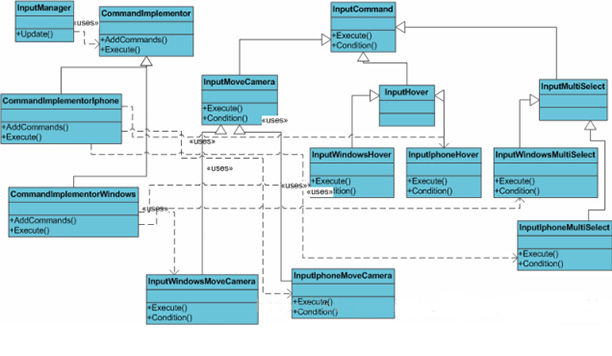
客户端只使用抽象工厂和抽象产品的方法，因此从一个平台的具体实施保护。

**Input and Persistant Data(输入和持久性数据)**

   主要有两个方面的游戏，不同的实施要求是输入机制和usageof的持久性文件的数据。Unity3D中不支持跨plattform[**数据库**](http://lib.csdn.net/base/mysql)。这就是为什么机甲和坦克使用的持久性数据在磁盘上的二进制文件。所有平台得到了他们对于自己的能力和自己的实施，支持的文件格式。

为了保持可读性和灵活性以下适应跨平台的输入处理的抽象工厂模式：

图 2.8输入子系统结构摘要

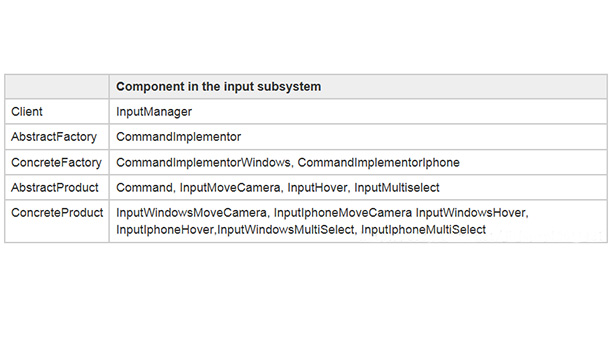


这仅仅是一个小所涉及的所有平台和命令，选择用于演示的摘录。

InputManager被附加到游戏物体在场景中的InputControl。它调用CommandImplementor的Execute方法的每一帧里面的更新功能。执行方法的CommandImplementor遍历所有添加的命令，检查他们的执行情况表示满意，并调用Execute方法，如果是这样。

**输入子系统的组成部分的图中的抽象工厂模式：**

图 2.9组成部分的图中的抽象工厂模式

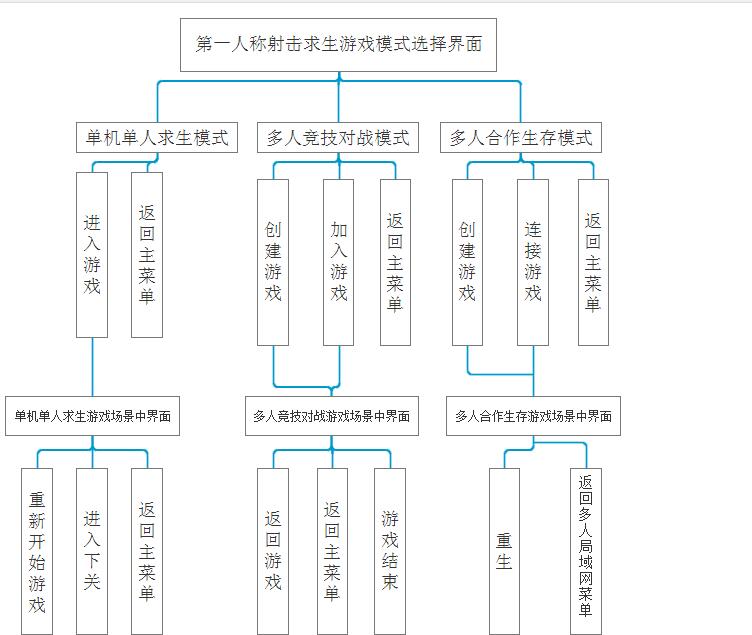


# 第一人称射击求生游戏系统设计

## 游戏系统设计

### 游戏结构设计

游戏主体上结构为一个小的射击类游戏合集，玩家可以自由的选择单人单机，多人竞技，多人合作生存模式下游戏的模式进行游戏。所以游戏的结构主要分为了三成：主菜单对游戏模式的选择，次级菜单对游戏局域网的连接的方式的选择，游戏进行界面。以下是图3.1菜单结构：

图 3.1菜单结构

### 游戏流程设计

开始打开游戏项目，进入游戏模式选择界面，如果选择了多人竞技或者多人合作生存后需要等待其他玩家的加入后才能进入下步游戏模式，选择了单人模式直接进入下步，场景加载完成同时生成了玩家的控制的第一人称角色，三个游戏的相同的目的是射击破坏点具有威胁的敌人并活下去。

其中的区别是单人单机模式下在不同的怪物的围攻下，玩家需要走出当前的迷宫，角色可以捡去各种的道具并增加自身的属性使得自己获得提升的生存能力或得到寻路的功能。

而多人模式下，如果是多人竞技，玩家的之间是竞争的关系，破坏掉对方的角色并保护好自身的角色到最后时获得最后的胜利。相对的多人合作生存下，玩家需要相互的协作下在一个密闭无限出现怪物的房间中尽可能的生存更长的时间。

图 3.2单人单机游戏模式流程图

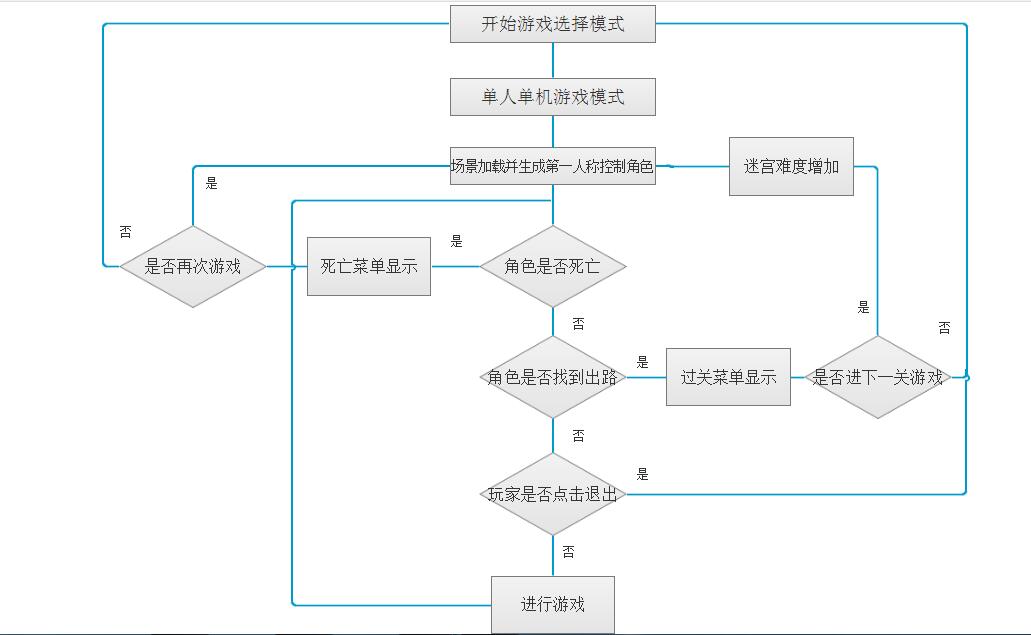


图 3.3多人竞技模式游戏流程图

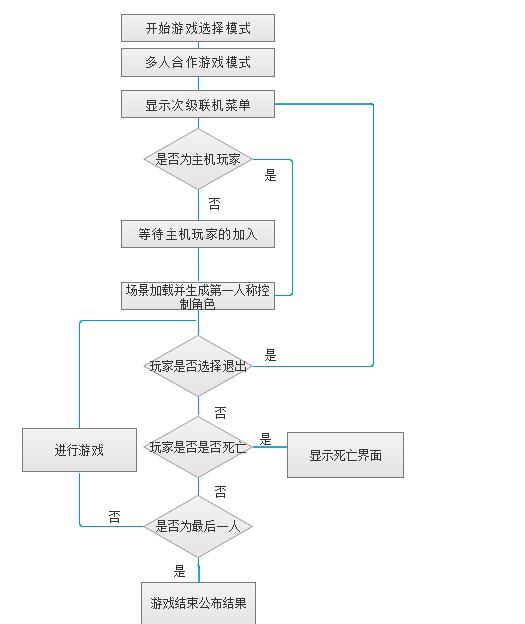
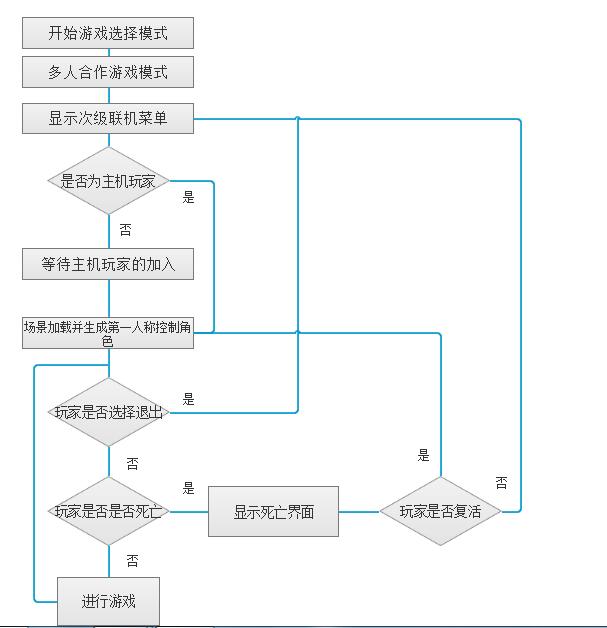


图 3.4多人合作生存模式流程图



### 功能行为与控制系统设计

游戏中有很多的功能行为，在大的导航功能菜单下面根据不同的模式有着不同的功能系统因为主体玩法是第一人称射击游戏所以主角的生成及主角的控制移动伤害死亡的判断这些功能都是相同的。

不同的是单人单机模式下场景是在开始加载时计算生成的迷宫，怪物和奖励物品的生成地点在此场景下也是随机的生成，在此基础上主角得到奖励的判断定和加成的属性的判断这些功能是单人单机特有的。

相对的多人模式下特有的功能是需要判断多人加入的联机，及其角色动画和伤害计算等的数据传输同步的功能等。

详细功能行为与控制关系如表3-1所示：

表 3‑1功能行为与控制关系表

|  |  |  |
| --- | --- | --- |
| **功能行为** | **系统控制** | **玩家控制** |
| 角色射击 | **√** | **√** |
| 角色移动 | **√** | **√** |
| 角色减血 | **√** |  |
| 角色消灭 | **√** |  |
| 迷宫生成场景加载 | **√** |  |
| 迷宫自动寻路 | **√** |  |
| 怪物生成 | **√** |  |
| 奖励物品生成 | **√** |  |
| 拾起奖励物品 | **√** |  |
| 过关判断 | **√** |  |
| 声音控制 | **√** |  |
| 玩家对战胜者判断 | **√** | **√** |
| 重新开始游戏 | **√** | **√** |
| 开启关卡 | **√** |  |
| 联机创建服务端 | **√** |  |
| 联机连接客户端 | **√** |  |
| 主角复活 |  | **√** |
| 导航菜单 | **√** | **√** |

## 游戏框架设计

### 设计要求

游戏类型 FPC是本游戏的核心，也是本游戏项目的亮点之一。单人模式为第一人称射击解密求生类型。多人模式下分为PVP（玩家和玩家进行对抗），PVE（玩家们合作与电脑进行对坑）属于第一人称射击竞技或求生类游戏。

运行平台 本游戏所有的操作及场景的计算都在Windows下P C电脑端完成，UI在不同的分辨率下能够自适应。因为第一人称射击类操控方式需要键位较多移植到移动端操作难度会提升并且真实感下降。本游戏放弃了移动端的开发。

### 游戏框架介绍

本游戏通过Unity3D引擎建模，通过脚本调用实现游戏功能。本游戏由各场景scene和脚本构成。根据游戏结构及流程分析项目的需求类型主要大体类如下列对象：

全体公共类：Menu、Player、Shoot、Bullet

单人单机类：Enemy、Gift、Migong、Wall、TrueWay、EnemyBullet

多人竞技类：Server、Client、OtherPlayer、Environment

多人合作生存类：Server、Client、Environment、OtherPlayer、Enemy

具体情况如下图：

#### 主界面

分别有 LoadManager – UI控制类、Menu -- Unity场景管理类

图 3.5主菜单抽象结构类图



#### 单人单机模式

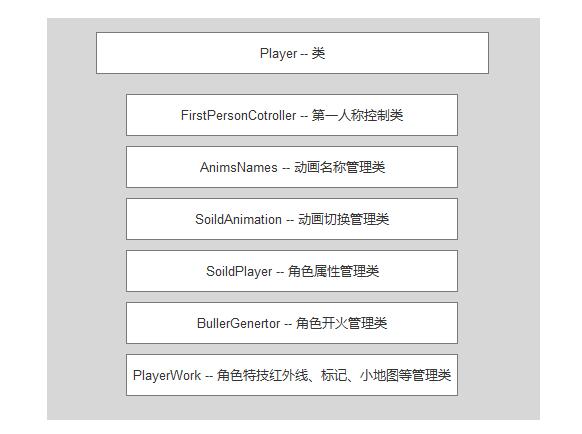
单人单机模式下分别有：Player类、Enemy类、场景类、UI控制类、Bullet类、gift类、Unity的场景分类之一migong2。

图 3.6单人单机抽象结构类图



Player类中有：FirstPersonCotroller—第一人称控制类、AnimsNames – 动画名称管理类、SoildPlayer – 角色属性管理类、BulletGenertor – 开火管理类、PlayerWork – 角色特技管理类

图 3.7玩家角色抽象结构类图



Enemys类中有：AnimsValueNames – 动画名称管理类、Enemy – 怪物基础属性及功能管理类、ZombeEnemy – 继承Enemy之靠近主角有伤害的怪物类、SoulEnemy – 继承Enemy之有攻击动画怪物类、ShootSoulEnemy – 继承SoulEnemy之远程怪物管理类。

图 3.8敌人抽象结构类图



场景类中有：Tags – 物体类型区分管理类、RandCreatePrim – 迷宫随机生成管理类、AutoPathing – 根据主角位子自动寻路管理类、MakeProduces – 奖励与怪物生产点与产生管理类、miniMap – 小地图显示管理类。

图 3.9场景抽象结构类图



UI控制类中有：GameManager – 场景功能管理类、Blood – 主角血量等属性显示管理类、GameOverPanel – 游戏失败显示管理类、 WinGamePanel – 游戏胜利显示管理类。

图 3.10 UI控制抽象结构类图



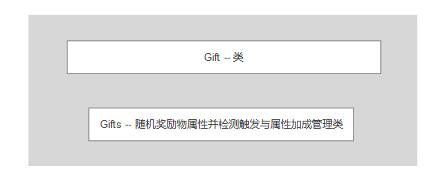
Bullet类中有：PlayerBullet – 玩家发射的子弹管理类、 EnemyBullet – 敌人发射的子弹管理类。

图 3.11子弹抽象结构类图



Gift类中有：Gifts – 随机奖励物属性并检测触发与属性加成管理类。

图 3.12奖励物品抽象结构类图



#### 多人竞技模式

单人竞技类中有：Player类、Client类、Server类、bullet类、场景类、UI控制类、Unity场景分类之一migong3。

图 3.13多人竞技抽象结构类图



Player类中有：FirstPersonCotroller – 第一人称控制类、AnimsNames – 动画名称管理类、PlayerAnimation – 动画切换管理类、 OwnerPalyer – 角色属性管理类、SCBulletGenertor – 角色开火管理类。

图 3.14玩家角色抽象结构类图



Client类：ClientPlayerCreate – 客服端玩家及主机角色管理类。

图 3.15客户端抽象结构类图



Server类中有：ServerPlayerCreate – 服务器玩家及连接玩家管理类、GameController – 伤害数据计算管理类。

图 3.16服务器抽象结构类图



Bullet类中有：PlayerBullet – 当前玩家子弹管理类、OtherPlayerBullet – 其他玩家子弹管理类。

图 3.17子弹抽象结构类图



场景类中有：ChangeAudio – 音乐切换控制类、卡通场景加载显示类。

图 3.18场景抽象结构类图



UI类中有：GameController – 局域网联机控制类、GameStop – 游戏暂停管理类、GameOver – 游戏结束管理类、OwnerPlayer – 角色属性管理类。

图 3.19 UI控制抽象结构类图



#### 多人合作生存模式

多人合作生存类中有：Player类、Network类、场景类、UI控制类、Enemy类、Unity自带场景分类：migong4Wait、migong4Play。

图 3.20多人合作生存抽象结构类图



Player类：PlayerAnimation—玩家动画管理类、PlayerDeath—玩家死亡管理类、Player\_ID – 玩家编号管理类、Player\_NetworkSetup – 玩家网络通信管理类、

Player\_Respawn – 玩家复活管理类、Player\_SyncPosition – 网络延迟角色位子信息管理类、Palyer\_SyncRotation – 网络演策角色方向信息管理、PlayerHeath – 玩家属性信息管理类、PlayerShoot – 玩家开火功能管理类。

图 3.21玩家角色类图



NetWork类中有：serverCreate – 服务器玩家属性信息管理类、ClentCreate – 客户端玩家属性信息管理类、EnemyCreate – 怪物创建管理类、NetWorkMaths – 信息传输计算管理类。

图 3.22网络联机抽象结构类图



场景类中有：EnemyGame – 怪物出现倒计时管理类、GameManager – 复活及断开连接UI显示。

图 3.23场景抽象结构类图



UI类中有：NetworkGamemanager – 局域网联机选择管理类、Spawn – 玩家复活管理类。

图 3.24 UI抽象结构类图



Enemy类中有：Zomb\_ID – 敌人编号管理类、ZombAttack – 敌人攻击管理类、ZombHealth – 敌人血量攻击力等属性管理类、Zomb\_MotionSync – 敌人移动管理类、ZombTarget – 敌人检测主角管理类。

图 3.25敌人抽象结构类图



## 游戏界面设计

### 画面设计

游戏的画面风格的设计离不开游戏风格的确定，当前的游戏为第一人称射击类游戏，如果是竞技类当然现代城市建筑更具有代入感。但当在当人单机的模式下或联网合作求生的下怪物为僵尸骷髅兵带有奇幻的色彩，所以又以恐怖为元素，增加给玩家的紧张感。迷宫配上怪物的求生无疑是紧张感很重的游戏体验，也是很有代入感的游戏体验。

### 主要界面设计

游戏部分素材使用网络上提供的美工素材，其他部分采用Unity3D自带的素材，朴素而不失趣味的素材易于获取也易于使用。

主菜单界面除了设计的LOGO显示外有 “进入单人冒险”、“进入多人对战”、“进入多人生存”、“退出游戏”等选项功能。其他UI风格类似。

图 3.26主菜单UI设计选项图



次级的联网模式下的联机菜单功能有“服务器的建立”、“客户端的连接”、“IP地址输入”、“返回主菜单”。

图 3.27次级的联网模式下的UI设计

单人单机的迷宫求生的为一个很大的地图在这里只展示部分效果图如图3.28：

图 3.28单人单机游戏效果图



关于联网竞技模式下的游戏场景是废弃的卡通风格的现代都市，增强玩家的第一人称现代战争感。

图 3.29多人竞技模式效果图



多人合作生存模式下玩家是在一个密闭的空间里争取生存更长的时间。而密闭配上恐怖的画风更给人一种紧张感。

图 3.30多人合作求生类效果图



## 数值设计

### 单人单机模式下所需数值

（1）玩家基础生命值 100 单位。

（2）玩家基础防御值 4 单位。

（3）玩家基础攻击值 11单位。

（4）每秒开枪子弹数量20颗。

（5）敌人随机出现时间间隔14s递减到6S。

（6）敌人随机生成数量2~6个单位。

（7）敌人攻击伤害值，攻击范围值。详情如表2所示：

（8）迷宫当前生成大小默认为9x9大小单位。

（9）奖励物品增加血量上限10个单位。

（10）奖励物品增加防御力20单位。

（11）奖励物品增加攻击力 10单位。

表 3‑2 Enemy攻击距离伤害血量数值详情

|  |  |  |  |
| --- | --- | --- | --- |
| **敌人名称** | **攻击距离** | **攻击伤害** | **血量** |
| Zombunny | 3 | 10 | 150 |
| ZombBear | 2 | 10 | 150 |
| Hellephant | 5 | 15 | 500 |
| SoulBoss | 3 | 30 | 250 |
| SoulMnster | 2 | 15 | 100 |
| SoulShooter | 8 | 10 | 100 |

### 多人联机竞技模式下所需数值

（1）服务端玩家角色攻击力10单位。

（2）客户端玩家角色攻击力10单位。

（3）服务端玩家角色血量100单位。

（4）客户端玩家角色血量100单位。

### 多人联机生存模式下所需数值

（1）玩家编号值 0~10号。

（2）玩家攻击力值50单位。

（3）玩家血量值100单位。

（4）怪物编号0~100号。

（5）怪物攻击值10单位。

（6）怪物血量值100单位。

（7）怪物攻击距离值1单位。

## 资源素材

### 模型资源

模型下载资源文件部分存放在项目下的Models目录下。下列列表展示了模型所在文件夹，模型名称及模型用途。

Boss文件存放的是SoulBoss的模型和动画文件。详情如表3-3所示：

表 3‑3 Boss文件夹素材详情

|  |  |  |
| --- | --- | --- |
| **文件夹** | **资源名称** | **用途** |
| Materials/NoName | NoName | 表现控制SoulBoss的显示颜色光泽等 |
| SoulBoss/SoulBossAvatar | SoulBossAvatar | 骨骼动画的支持 |
| SoulBoss/Boss | Boss | 模型文件 |
| SoulBoss@BossAttack01/ BossAttack01 | BossAttack01 | Soulboss的攻击动画1 |
| SoulBoss@BossAttack02/ BossAttack02 | BossAttack02 | Soulboss的攻击动画2 |
| SoulBoss@BossDamage01/ BossDamage01 | BossDamage01 | Soulboss的受伤动画 |
| SoulBoss@BossDeath01/ BossDeath01 | BossDeath01 | Soulboss的死亡动画 |
| SoulBoss@BossRun01/ BossRun01 | BossRun01 | Soulboss的跑动动画 |
| SoulBoss@BossStand01/ BossStand01 | BossStand01 | Soulboss的静止动画 |

Character文件存放的是其他Zomb卡通风格角色文件。详情如表3-4所示：

表 3‑4 Character文件夹素材详情

|  |  |  |
| --- | --- | --- |
| **文件夹** | **资源名称** | **用途** |
| Hellephant/Hellephant | Hellephant | Hellephant的模型 |
| Hellephant/HellephantAvatar | HellephantAvatar | 骨骼动画的支持 |
| Hellephant/Death | Death | Hellephant的死亡动画 |
| Hellephant/Hellephant/ Idle | Idle | Hellephant的静止动画 |
| Hellephant/Hellephant/ Move | Move | Hellephant的移动动画 |
| Hellephant/Hellephant/Hurt | Hurt | Soulboss的受伤动画 |
| Player/Player | Player | Player的模型 |
| Player/Gun | Gun | Gun的模型 |
| Player/Death | Death | Player的死亡动画 |
| Player/PlayerAvatar | PlayerAvatar | Player的骨骼动画的支持 |
| Player/Move | Move | Player的移动动画 |
| Player/Idle | Idle | Player的静止动画 |
| ZomBear/Zombear | Zombear | ZomBear的模型 |
| ZomBear/ZomBearAvatar | ZomBearAvatar | ZomBear的骨骼动画的支持 |
| Zombunny/Zombunny | Zombunny | Zombunny的模型的支持 |
| Zombunny/ZombunnyAvatar | ZombunnyAvatar | ZombunnyAvatar的骨骼动画的支持 |
| Zombunny/Death | Death | Zombunny的死亡动画 |
| Zombunny/Idle | Idle | Zombunny的静止动画 |
| Zombunny/Move | Move | Zombunny的移动动画 |

Monster文件夹存放的是SoulMonster的文件。详细信息如下表3-5：

表 3‑5 SoulMonster文件夹素材详情

|  |  |  |
| --- | --- | --- |
| **文件夹** | **资源名称** | **用途** |
| Materials/NoName | NoName | 表现控制SoulBoss的显示颜色光泽等 |
| SoulMonster/ SoulMonsterAvatar | SoulMonsterAvatar | 骨骼动画的支持 |
| SoulMonster/ SoulMonster | SoulMonster | 模型文件 |
| SoulMonster@ SoulMonsterAttack01/ Attack01 | Attack01 | SoulMonster的攻击动画1 |
| SoulMonster@ SoulMonsterAttack02/ Attack02 | Attack02 | SoulMonster的攻击动画2 |
| SoulMonster@ SoulMonsterDamage01/ Damage01 | Damage01 | SoulMonster的受伤动画 |
| SoulMonster@ SoulMonsterDeath01/ Death01 | Death01 | SoulMonster的死亡动画 |
| SoulMonster@ SoulMonsterRun01/ Run01 | Run01 | SoulMonster的跑动动画 |
| SoulMonster@ SoulMonsterStand01/Stand01 | Stand01 | SoulMonster的静止动画 |

Envrioment文件夹中存放的是恐怖风格密室所需元素。详细信息如下表3-6：

表 3‑6 Envriment文件夹素材详情

|  |  |  |
| --- | --- | --- |
| **文件夹** | **资源名称** | **用途** |
| Arches | Arches | 暗系场景玩具1 |
| Bat | Bat | 暗系场景划桨 |
| Blox | Blox | 暗系场景玩具2 |
| Clock | Clock | 暗系闹钟 |
| DollArm | DollArm | 暗系玩具手臂 |
| Dollhouse | Dollhouse | 暗系场景房子 |
| Drawers | Drawers | 暗系场景抽屉 |
| Firetruck | Firetruck | 暗系火车 |
| Floor | Floor | 暗系场景地板 |
| Hearse | Hearse | 暗系反倒的汽车 |
| Robot | Robot | 暗系场景惊吓盒子 |
| SpinningTop | SpinningTop | 暗系转轮 |
| Stool | Stool | 暗系场景坐凳 |
| Train | Train | 暗系火车 |
| Wall | Wall | 暗系墙壁 |

SoulShooter是根据Soul系列的Player改编得到的，具体文件信息如表3-7：

表 3‑7 SoulShooter文件夹素材详情

|  |  |  |
| --- | --- | --- |
| **文件夹** | **资源名称** | **用途** |
| Materials/NoName | NoName | 表现控制SoulBoss的显示颜色光泽等 |
| SoulShooter | SoulShooterAvatar | SoulShooter骨骼动画的支持 |
| SoulShooter / SoulShooter | SoulShooter | 模型文件 |
| SoulShooter r@ SoulShooter / Shoot | Shoot | SoulShooter 的攻击动画1 |
| SoulShooter @ SoulShooter Damage01/ Damage01 | Damage01 | SoulShooter的受伤动画 |
| SoulShooter @ SoulShooter Death01/ Death01 | Death01 | SoulShooter的死亡动画 |
| SoulShooter @ SoulShooterRun01/ Run01 | Run01 | SoulShooter的跑动动画 |
| SoulShooter @ SoulShooterStand01/Stand01 | Stand01 | SoulShooter的静止动画 |

Cartoon Sholdier中是第一人称控制的士兵角色文件，具体使用到的部分文件信息如下表3-8：

表 3‑8 Cartoon Sholdier文件夹素材详情

|  |  |  |
| --- | --- | --- |
| **文件夹** | **资源名称** | **用途** |
| 3D/soldier/gun | gun | 主角枪模型 |
| 3D/soldier/soldier | soldier | 主角人物模型 |
| Animations/soldier | soldierCrouch | 主角走动动画 |
| Animations/soldierCrouchRun | soldierCrouchRun | 主角跑动动画 |
| Animations/SoldierSpinLeft | SoldierSpinLeft | 主角向左走动动画 |
| Animations/ SoldierSpinRight | SoldierSpinRight | 主角向右走动动画 |
| Animations/soldierDieBack | soldierDieBack | 主角向后死亡动画 |
| Animations/soldierFalling | soldierFalling | 主角下落动画 |
| Materials/bullerHoles/bulletHoleHeat | bulletHoleHeat | 主角子弹击中墙面特效 |
| Materials/laserSight/LaserDust | LaserDust | 红外线瞄准位置特效 |
| Materials/laserSight/LaserLine | LaserLine | 红外线线上特效 |
| Materials/laserSight/LaserPoint | LaserPoint | 红外线发射点特效 |
| Materals/muzzleFlash/muzzleFlash1 | muzzleFlash1 | 开火特效1 |
| Materals/muzzleFlash/muzzleFlash2 | muzzleFlash2 | 开火特效2 |
| Materals/muzzleFlash/muzzleFlash3 | muzzleFlash3 | 开火特效3 |
| Materals/muzzleFlash/muzzleFlash4 | muzzleFlash4 | 开火特效4 |
| PhysicMaterials/sentryGunPhysics | sentryGunPhysics | 枪的物理特性 |
| PhysicMaterials/soldierPhysics | soldierPhysics | 角色的物理特性 |

SimpleTownPack中有许多文件但都只是用于场景布置，在此列举部分文件内容如下表3-9：

表 3‑9 SimpleTownPack文件夹素材详情

|  |  |  |
| --- | --- | --- |
| **文件夹** | **资源名称** | **用途** |
| ambo\_mesh | ambo\_mesh | 场地汽车模型 |
| ambo\_seperate\_mesh | ambo\_seperate\_mesh | 场地汽车模型 |
| apartment\_large\_mesh | apartment\_large\_mesh | 场地建筑模型 |
| apartment\_small\_mesh | apartment\_small\_mesh | 场地建筑模型 |
| ariel\_mesh | ariel\_mesh | 场地建筑模型 |
| billboard\_mesh | billboard\_mesh | 场地建筑模型 |
| bin\_mesh | bin\_mesh | 场地建筑模型 |
| bus\_mesh | bus\_mesh | 场地汽车模型 |
| bus\_seperate\_mesh | bus\_seperate\_mesh | 场地汽车模型 |
| bush\_large\_mesh | bush\_large\_mesh | 场地建筑模型 |
| bush\_small\_mesh | bush\_small\_mesh | 场地建筑模型 |
| car\_mesh | car\_mesh | 场地汽车模型 |
| cop\_mesh | cop\_mesh | 场地汽车模型 |
| dish\_mesh | dish\_mesh | 场地建筑模型 |
| dumpster\_mesh | dumpster\_mesh | 场地建筑模型 |
| fence\_long\_mesh | fence\_long\_mesh | 场地建筑模型 |
| fire\_truck\_mesh | fire\_truck\_mesh | 场地汽车模型 |
| house\_mesh | house\_mesh | 场地建筑模型 |

### 声音资源

声音资源文件全部放在项目下的Assets/Audio或Audios文件夹中。声音资源文件详情如表11所示：

表 3‑10声音资源文件详情

|  |  |
| --- | --- |
| **资源名称** | **用途** |
| bg\_menu.mp3 | 游戏菜单选择时背景音乐 |
| bg\_play.mp3 | 游戏进行时背景音乐 |
| Shoot.mp3 | 角色开枪时音效 |
| Background Music.mp3 | 恐怖风格下的音乐背景 |
| HellphantDeath.mp3 | 僵尸大象死亡音效 |
| HellePhantHurt.mp3 | 僵尸大象受伤音效 |
| PlayerDeath.mp3 | 玩家死亡音效 |
| PlayerGunShot.mp3 | 玩家开枪音效（暗系卡通主角） |
| Player Hurt.mp3 | 玩家受伤音效 |
| ZomBear Death.mp3 | 僵尸熊死亡音效 |
| ZomBear Hurt.mp3 | 僵尸熊受伤音效 |
| ZomBunny Death.mp3 | 僵尸兔子死亡音效 |
| ZomBunnyHurt.mp3 | 僵尸兔子受伤音效 |

# 第一人称射击求生游戏详细设计与实现

## 设计游戏制作过程

首先本游戏项目分为三个环节，第一环节为单人单机模式下加入迷宫模式的求生模式设计。第二环节为双人联机对战模式。第三环节为多人联机末日求生。因为通过Unity3Dy引擎制作，首先制作游戏所需元素制作Prefabs并通过脚本计算生成场景和其他元素。

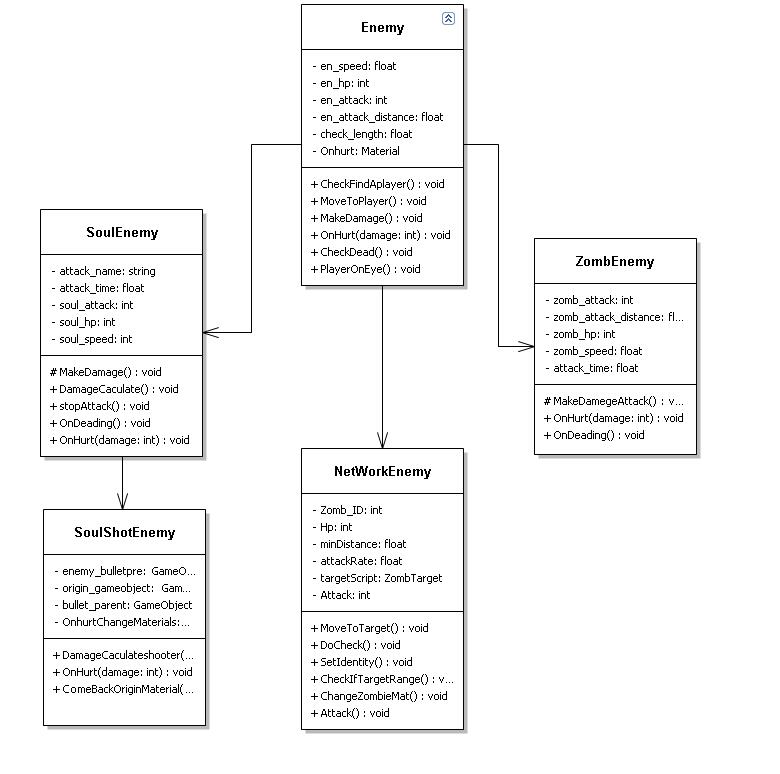
## Prefab的简单说明

Prefab中的模型运用于制作属性相同的重复模型（可修改位置方向等信息），同时Prefabs模型上添加了脚本文件后在拖拽到场景中时是通用的，在场景中对脚本的文件的属性也可进行修改使得模型有不同的表现。

## 游戏实体对象主要类类图

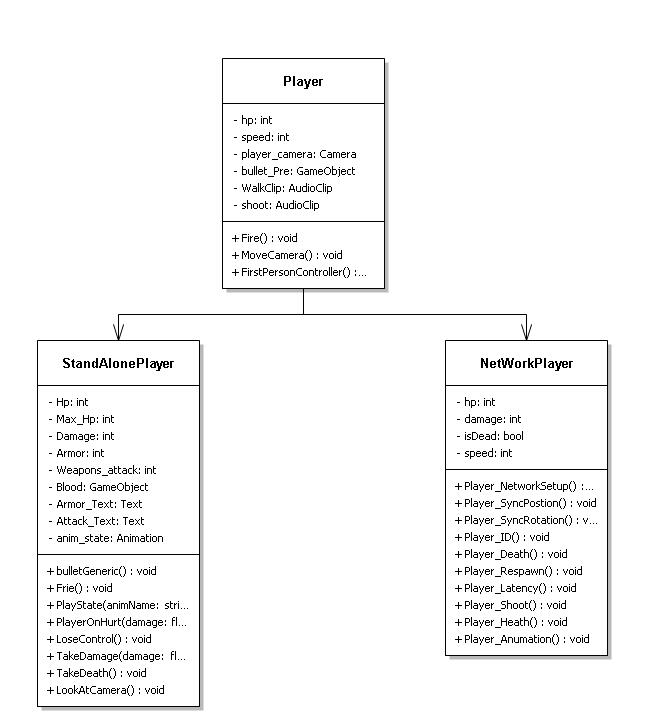
### 游戏实体对象的怪物类图

图 4.1怪物类图



### 游戏实体对象的玩家类图

图 4.2玩家类图

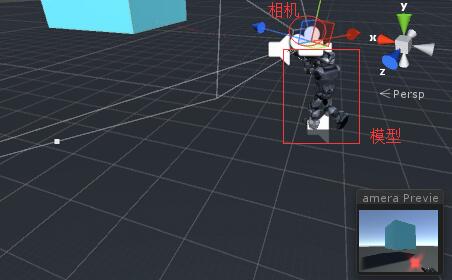


## 主角的移动及开火控制

### 主角的移动控制

主角是通过模型和第一人称相机组合形成的一个物体。其中的模型主要控制玩家指令动作的动画的播放和切换如跑动、跳跃、受伤、死亡等。相机控制的是第一人称的视角，并实现跑动中的摇晃增强真实感。总体的物体类控制玩家的位置的改变。

图 4.3主角人物模型结构展示图



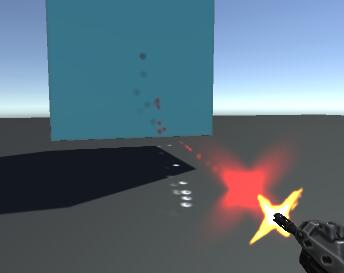
### 主角的枪及开火控制

主角的枪是带有红外线瞄准功能同时设计时主角移动中枪的准星是不稳定的，稳定性：跑动<走动<站立。

红外线的检测原理是和静止站立时子弹发射检测原理相同。通过当前摄像机的位置然后面向屏幕正中位置发射射线检测，在角色的枪上提供红外线的发生点并枪口提供子弹的发生点。

玩家在跑动过程中是子弹则是根据枪的模型朝向进行检测飞行。当子弹碰撞到物体是进行判断检测当前物体如果为墙面地面则生成伤痕。如果为怪物则怪物收到伤害。

图 4.4玩家开火效果图



## 敌人的派生继承

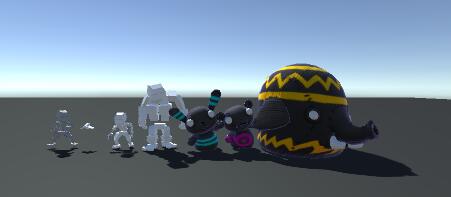
Enemy的AI和攻击方式等不同的敌人是有区别的，但是大体上大部分的功能是相同的所具有的的属性也是大部分相同只是参数不同。所有这种类型的模型代码控制很适合使用派生继承类实现。

这里Enemy为所有怪物的原始类型。Enemy中有着所有怪物所必需的的属性及其功能的实现如

属性上有：血量、攻击力、攻击范围、自动的寻路、移动速度等。

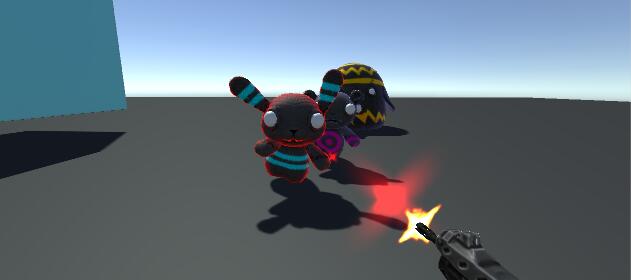
功能上有：对玩家出现的监听、对玩家的锁定追击、攻击、受到伤害、死亡等。

图 4.5怪物模型全体图



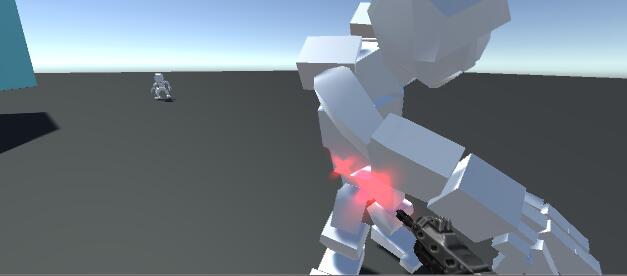
ZombEnemy为Enemy的继承类之一。Zomb类为僵尸型怪物类，这类怪物的特点是血厚移动快但是攻击距离短。同时小的怪物Zomb是能被玩家攻击击退的。而大的怪物boss不能击退并且移动更快给玩家一种挑战。

图 4.6 ZombEnemy模型展示图



SoulEnemy为Enemy的另一大继承类。Soul类为骷髅兵形怪物类，这类的怪物特点是血量少，攻击距离适中，同时攻击时有攻击出手动作玩家可闪避。攻击力较高，收到其一次攻击对主角的伤害较高。同时小怪移动速度快速大怪类型移动缓慢。

图 4.7 SoulEnemy模型展示图



SoulShooterEnemy是SoulEnemy的下级继承类。SoulShooter的特别之处是这类怪物可以使用枪械对主角进行射击。攻击距离远攻击力中等但是血量较少。

图 4.8 Shooter的攻击效果图



## 子弹功能

子弹制作时主要注意的功能是向前的快速移动、定时自身的销毁、和碰撞物体的检测。

这里要注意的是子弹的向前移动的速度很快，检测时可能上一个监测点到下一个监测点移动的距离很大，中间经过了很多物体而没法检测。所以设计时是通过下个监测点向之前的检测点发射射线。通过这一段距离的射线进行检测有无碰撞物体然后进行判断响应。

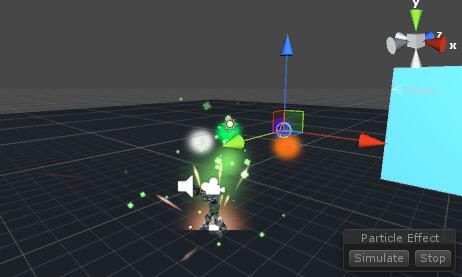
图 4.9子弹检测效果图



## 奖励物品

奖励物品的模型是根据粒子器制作的，同时不同奖励物替换了不同的色彩看上去很绚丽。当玩家捡起不同的物品时对应的属性加成有不同的效果粒子特效生成并伴有文字的显示给玩家清楚明白的提示。奖励物品的属性根据生成时随机确定，奖励效果固定加成什么便有相应的奖励效果。

图 4.10奖励物品效果展示图



## 迷宫生成与自动寻路

### 迷宫生成算法

#### Prim算法迷宫实现原理

这里的迷宫生成算法是借鉴的Prim迷宫生成算法基础上的独创生成算法。

普里姆算法（Prim算法），图论中的一种算法，可在加权连通图里搜索最小生成树。意即由此算法搜索到的边子集所构成的树中，不但包括了连通图里的所有顶点（英语：Vertex (graph theory)），且其所有边的权值之和亦为最小。该算法于1930年由捷克数学家沃伊捷赫•亚尔尼克（英语：Vojtěch Jarník）发现；并在1957年由美国计算机科学家罗伯特•普里姆（英语：Robert C. Prim）独立发现；1959年，艾兹格•迪科斯彻再次发现了该算法。因此，在某些场合，普里姆算法又被称为DJP算法、亚尔尼克算法或普里姆－亚尔尼克算法。——来自百度百科

当我们将Prim算法用于迷宫生成时，情况有些不同，维基百科中给出了随机Prim迷宫生成算法的解释及实现过程：

1. 让迷宫全都是墙.。

2. 选一个格，作为迷宫的通路，然后把它的邻墙放入列表.。

3. 当列表里还有墙时:

1. 从列表里随机选一个墙，如果它对面的格子不是迷宫的通路:

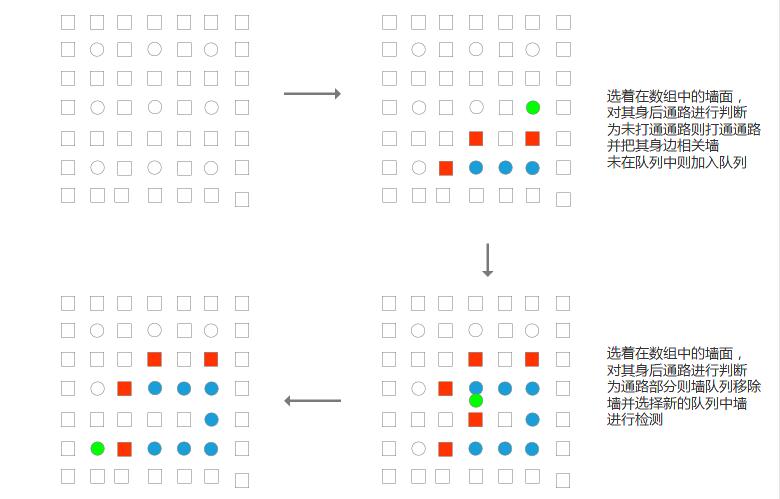
(a) 把墙打通，让对面的格子成为迷宫的通路.。

(b) 把那个格子的邻墙加入列表。

2. 如果对面的格子已经是通路了，那就从列表里移除这面墙。

及其核心是把随机迷宫中的通路使用墙完全的封住把隔断未连同的通路的墙放入一个数组。然后对数组中的墙后的通路进行判断是否打通墙面并对墙从数组中移除。

图 4.11 Prim算法说明图



#### 基于Prim迷宫算法的改进后的迷宫算法

本游戏的迷宫算法与Prim迷宫算法的相同点是开始时也是使用墙把迷宫的全部通路都通过墙面进行隔离。不同点是该算法的判断点核心是未连同点的数目为零时说明迷宫生成连同成功。

生成开始时迷宫中从出发点到终点随机朝向的移动形成一条通路，然后随机选着剩下未打通的点进行随机连通到主路经上。标记点未打通的节点总个数为0时迷宫自动生成成功。

该算法的注意点，连通两点的路径方式源于自动寻路的公式只有一个路径并且走过的不能再次行走，当进入死路时后退一步重新选择方向进行自动寻路操作。

相对于之前的Prim下的迷宫生成算法的缺点：

生成迷宫时除了记录未连同点的数列外（prim算法中为墙的数列），在每次生成连同路径时需要额外的数列存储连同时走过的路径点，以防止当前走进思路后需要退回到上一节点进行重新寻路操作。

相对于之前的Prim下的迷宫生成算法的优点：

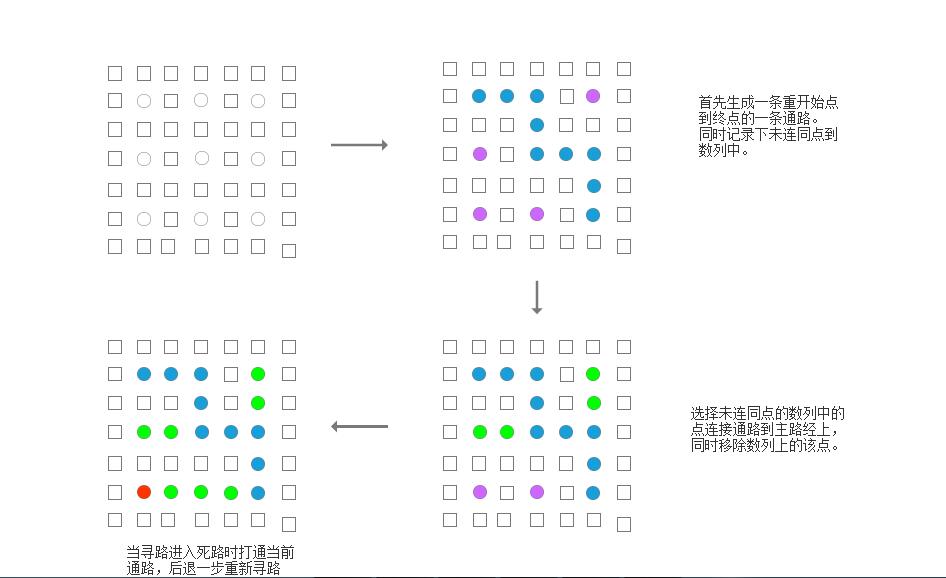
1、生成迷宫的同时，可以知道起点到终点的正确路径。

2、生成迷宫的同时知道死路的最终点的个数和每个点的位置。（死路点为三面墙都没有打通的点）一般都是随机选择的支路的出发点，和寻路时进入的死路需后退的点。

这些点在本游戏中用于固定生成怪物和奖励物品以增强玩家探索迷宫的兴趣度同时前期为玩家走错迷宫时做出补偿和挑战使得单一的走迷宫游戏变得不那么无聊。

3、除此之外如果需要同时可以记录下支路寻路的路径，当需要开始点或终点到之路点的路径可以快速得到。

图 4.12本游戏的迷宫生成算法图



## Unity旧版系统局域网联机

### 旧版本的网络通信系统

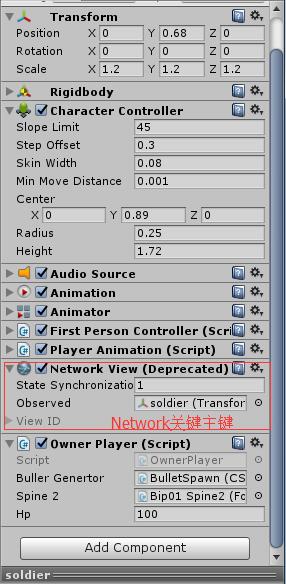
Unity之前的网络通信系统是通过在需要在服务端和客户端交互的物体上添加NetWorkView关键主键。当只有服务器时该物体只会创建一个模型在服务器端。但是当有客户端连接进入时。首先服务端会自动生成有NetWorkView主键的物件作为客户端的物体。同时在客户端连接时，同时生成两个带有NetWorkView的该模型物体，一个作为客户端本生控制的物体，一个作为服务器端的物体参照。

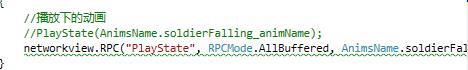
### 数据动画的同步的实现

在老版本的系统下NetWorkView物体在服务器和客户端的动画同步需要通过调用NetWorkView功能主键进行通信。并传输相应的动画名称进行通信实现同步处理。

而数据的计算时，是在物体本身的代码控制上提供好对应的方法再通过NetWorkView主键下的方法如RPC进行远程的调用实现数据的计算。如客户端对服务器端造成伤害并进行计算时，首先服务器端角色物体上的代码需要有受到伤害计算的方法类或函数，然后是客户端通过RPC的方法进行远程的调用实现。

图 4.13 Unity旧版本的通信系统实现方式图





## Unity新版系统局域网联机

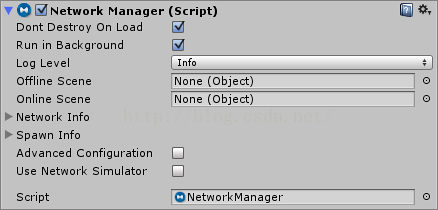
### 新本版的网络通信系统

#### NetWorkManager介绍

NetworkManager 网络管理器是 管理网络状态的多人游戏的一个组成部分。它实际上被实现完全使用 HLAPI，所以它做的一切都是可被其他forms的开发人员使用的。然而网络管理器包装了很多有用的功能，使得创建、 运行和调试尽可能简单的多人游戏。

网络管理器可以使用完全不用脚本。它有在编辑器中有 inspector 控件，允许配置其所有功能，和 NetworkManagerHUD 提供简单、默认的用户界面，可以在运行时允许被用户控制的网络游戏。为高级的使用，开发人员可以从网络管理器派生出类 并通过重写虚函数，为它提供的任何自定义其行为。

图 4.14 Network Manager的信息介绍图

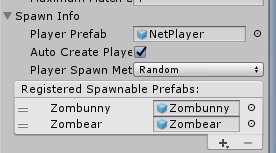


#### Spawning Management主角和怪物的加载生成

网络管理器可以用于管理网络对象从预置的实例化 。大多数游戏都有 prefab 用作player 的对象，因此网络管理器有一个 槽（字段），拖动player prefab进行赋值。当一个player prefab 设置时，player 对象会自动产生从该prefab， 为每个用户在游戏中创建。这适用于在本地player托管hosted服务器上，和远程客户端上的远程players 参与者。请注意player prefab 必须有 NetworkIdentity 组件。

除了player prefab，其他对象的预置将动态地被生成了但是必须是ClientScene登 记的。

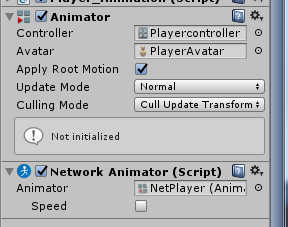
图 4.15需要动态生成物体放入图



#### 动画同步系统NetWorkAnimator

NetWorkAnimator用来跨网络同步动画，需要指明同步的动画的参数后，系统自动同步。

图 4.16系统同步动画NerworkAnimator图



#### NetworkBehaviour 添加的脚本父类

NetworkBehaviours 是特别脚本，用于处理对象上的 NetworkIdentity 组件。这些脚本都能够执行 HLAPI ，像Commands、 ClientRPCs、 SyncEvents 和 SyncVars 等功能。

与Unity Network System服务器authoritative 权威系统，网络对象的 NetworkIdentities 必须是“spawned”，由服务器使用 NetworkServer.Spawn()。这会导致他们被分配一个NetworkInstanceId，在连接到服务器的客户端上创建。

以下是一些相关判断的参数变量：

isLocalPlayer 如果对象是本地玩家的对象。

isServer 如果对象在服务器运行，并已经实例化。

isClient 如果对象在客户端运行。

hasAuthority 如果对象为当前权威版本的对象。 如果既不在server也不在Client则称作 localPlayerAuthority.

assetId 对象在 NetworkIdentity上的 assetId。.

netId 对象在NetworkIdentity上的netId。

playerControllerId 对象在NetworkIdentity上的 PlayerControllerId。

connectionToServer NetworkConnection 对象向服务器发送消息命令。

connectionToClient

NetworkConnection 对象向客户端发送消息命令。

NetworkBehaviours 具有以下的特点，介绍如下。

• Synchronized Variables 同步变量

• Network callbacks 网络回调

• Server and Client functions服务器和客户端功能

• Sending Commands发送命令

• Client RPC Calls 客户端 RPC 调用

• Networked Events 网络事件

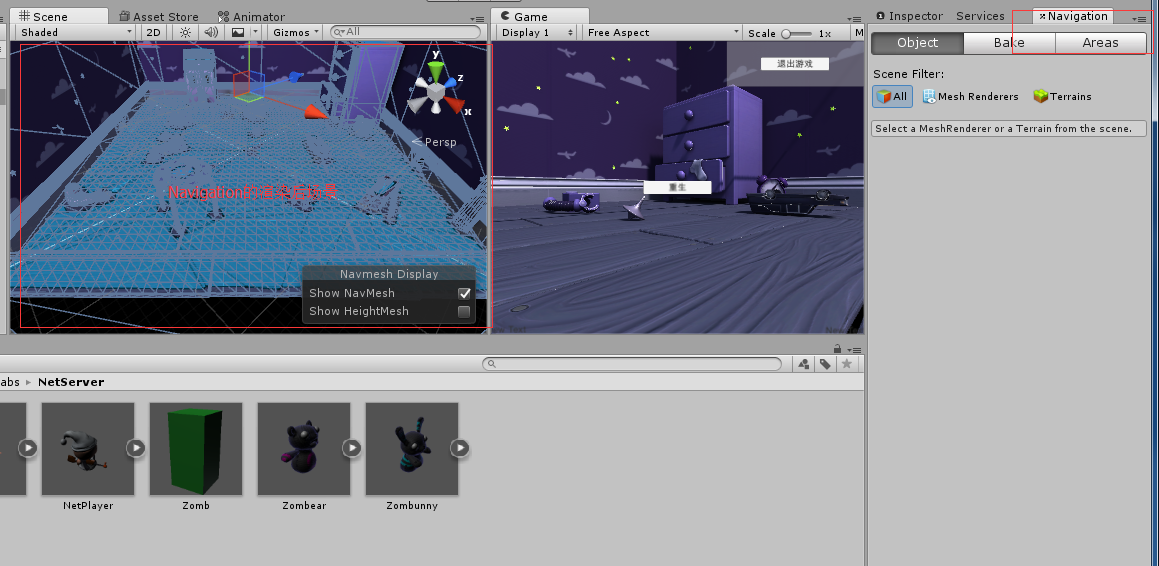
### 服务端和客户端的区分

服务器与客户端的物体模型是NetWorkMamnager自动创建创建的，同时在创建的的时候会有对应网络上的ID编号，一般在具体选择判断某一个具体模型的时候通过这个唯一标识的ID编号变可以。同时如果是服务器和客户端的具体区分的话，需要的是通过在执行继承NetworkBehaviour的代码中判断islocalPlayer、Client、Command等特殊命令进行代码的区分进行。

### 怪物的产生及AI的寻路控制

怪物的产生是在服务器端产生的如上通过编号进行具体的区分，同时客户端得到相关数据信息。在AI寻路方式上，因为场景不在是动态加载所以这里使用的是Unity的NAV自带的寻路方式。首先对场景进行渲染成Nav层。当指示怪物到某个点的移动时怪物通过Nav的函数会自动的移动前往。所以这里主要找到判断某一个玩家的位置并传输给怪物的脚本代码执行便能自动寻路。这里查找时，使用的是发射射线的方式进行查找。Unity提供了场景中自定义物体所在层的区分。可以把玩家模型单独放置在一层中。然后只对当前玩家所在层进行检测查找，这样就不必在意场景中物体的阻挡而无法找到玩家对象。

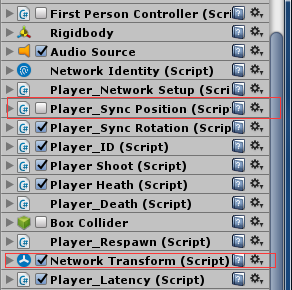
图 4.17 Navigation的渲染场景的效果图



### 数据信息的同步

数据同步同步是有通过脚本实现或NetworkManager相关主键的添加两种实现方式。NetworkManager提供的主键只是提供了最基本的同步的数据操作信息但是面对有网络延迟问题是没有更好的解决方案的。如位置的同步时数据可以通过添加的NetworkTranform进行同步，也可以自定义继承NetworkBehaviour脚本进行自己的编写。在面对延迟问题时，脚本可以记录一段时间的位置信息在数组中一次性进行传输，对应接收端再分开解析实现对延迟影响的消除。

图 4.18位置数据两种同步方式图



### 死亡复活功能的实现

当多人模式下坚持活下去是游戏的目的但是如果玩家死亡后能够复活继续战斗更能增加游戏的乐趣，这里死亡复活的事件是需要监听的而本游戏使用的方式是通过DieDelegate事件托管在需要的时候进行调用便可以。同时复活UI的显示在玩家正在游戏时应该隐藏起来。

## 关键功能代码及说明

### 单人模式下的迷宫生成代码

单人模式下最大的亮点在于随机性，在相同的规则下面玩家能有不同的体验。而一切随机的基础也是本来自身最大的随机变数便是迷宫，迷宫的生成随机生成的情况。

//创建实体迷宫

//创建实体迷宫

private void CreatePrefabMap()

{

//检测保证输入的值是否有效

if (widthLimit < 1)

{

widthLimit = 1;

}

if (heightLimit < 1)

{

heightLimit = 1;

}

if (startXY[0] < 1 || startXY[0] > widthLimit)

{

startXY[0] = UnityEngine.Random.Range(1, widthLimit); //输出的为整数

}

if (startXY[1] < 1 || startXY[1] > heightLimit)

{

startXY[1] = UnityEngine.Random.Range(1, heightLimit);

}

//生成全是墙封闭的墙面的格子地图

//N个格子需要打通N个墙面(连同所有的格子)

//初始点是未格子中的任意一个格子

//终点是连通格子（第一次默认是开始位置到终点）

node[,] map = MakeMazemap(startXY[0], startXY[1], widthLimit, heightLimit, widthLimit, heightLimit, CreateEmptyMazemap(widthLimit, heightLimit));

//初始墙体 和 结束位子墙体

map[widthLimit \* 2 - 1, heightLimit \* 2] = node.pass;

//循环二维数组XY坐标查看具体迷宫信息并实例话

for (int i = 0; i < widthLimit \* 2 + 1; i++)

{

for (int j = 0; j < heightLimit \* 2 + 1; j++)

{

if (map[i, j] == node.wall)

{

//生成实体迷宫同时放入一个父物体下方便管理操作比如整体的放大操作。

GameObject obj\_node = Instantiate(wallPrefabs, new Vector3((float)i, 0, (float)j), Quaternion.identity) as GameObject;

obj\_node.transform.parent = parentobj.transform;

}

//生成随机物

if (map[i, j] == node.truewayproduce || map[i, j] == node.produce)

{

//根据创建迷宫知道的信息特点生成怪物和奖励物品

CreateProduce(i, j);

CreateGifts(i, j);

}

//生成地面

GameObject obj\_ground = Instantiate(groundPrefabs, new Vector3((float)i, -1, (float)j), Quaternion.identity) as GameObject;

obj\_ground.transform.parent = parentobj.transform;

//保存地图信息

map\_dict.Add(i + "," + j, map[i, j]);

}

}

//迷宫放大十倍

parentobj.transform.localScale = new Vector3(10, 10, 10);

}

//生成全是墙封闭的墙面的格子地图

private node[,] CreateEmptyMazemap(int widthLimit, int heitLimit)

{

//形成外围墙

node[,] mazemap = new node[widthLimit \* 2 + 1, heitLimit \* 2 + 1];

//个数扩张一倍作为墙体添加

for (int column = 0; column < widthLimit \* 2; column ++) //列

{

for (int row = 0; row < heitLimit \* 2; row++) //行

{

//分配墙还是通路

if (column % 2 == 0 || row % 2 == 0)

{

mazemap[row, column] = node.wall;

}

else

{

mazemap[row, column] = node.unpass;

}

}

}

//补全围墙

for (int column = 0; column <= widthLimit \* 2; column++)

{

mazemap[widthLimit \* 2, column] = node.wall;

}

for (int row = 0; row <= heitLimit \* 2; row++)

{

mazemap[row, heitLimit \* 2] = node.wall;

}

return mazemap;

}

//打通连通的墙壁

private node[,] MakeMazemap(int startX, int startY, int endX, int endY, int widthLimit, int heightLimit, node[,] empty\_mazemap)

{

//存放未连通格子

List<string> unpassnode = new List<string>();

for (int row = 0; row < heightLimit; row ++)

{

//记录未打通通道的位置信息和个数在数列中

for (int column = 0; column < widthLimit; column++)

{

string nodexy = (row \* 2 + 1).ToString() + "," + (column \* 2 + 1).ToString();

unpassnode.Add(nodexy);

}

}

//尝试打通一条通路的中间过程记录，防止走进死路需要后退

List<string> way = new List<string>();

//换算新的开始点

int targetX = startX \* 2 - 1;

int targetY = startY \* 2 - 1;

//未打通点数列中出去第一个点

unpassnode.Remove(targetX+","+targetY);

unpassnode.Remove((endX \* 2 - 1) + "," + (endY\*2 -1));

//改变已打通点属性

empty\_mazemap[targetX, targetY] = node.pass;

int count = 4; //方向选择判断次数

int dection = RandomInt(); //方向

while (targetX != endX \* 2 - 1 || targetY != endY \* 2 - 1)

{

//打通到迷宫出口的一条路

if (dection == 0) //上

{

//判断选中点如果为不能打通点着跳过

if (targetX - 2 < 1 || empty\_mazemap[targetX - 2, targetY] == node.pass || empty\_mazemap[targetX - 2, targetY] == node.truewayproduce)

{

dection++;

count--;

}

else

{

targetX -= 2;

empty\_mazemap[targetX, targetY] = node.pass;

empty\_mazemap[targetX + 1, targetY] = node.pass;

way.Add(targetX + "," + targetY);

unpassnode.Remove(targetX + "," + targetY);

count = 4;

dection = RandomInt(); //方向

}

}

if (dection == 3) //右

{

if (targetY + 1 >= widthLimit \* 2 || empty\_mazemap[targetX, targetY + 2] == node.pass || empty\_mazemap[targetX, targetY + 2] == node.truewayproduce)

{

dection = 0;

count--;

}

else

{

targetY += 2;

empty\_mazemap[targetX, targetY] = node.pass;

empty\_mazemap[targetX, targetY - 1] = node.pass;

way.Add(targetX + "," + targetY);

unpassnode.Remove(targetX + "," + targetY);

count = 4;

dection = RandomInt(); //方向

}

}

if (dection == 2) //下

{

if (targetX + 1 >= heightLimit \* 2 || empty\_mazemap[targetX + 2, targetY] == node.pass || empty\_mazemap[targetX + 2, targetY] == node.truewayproduce)

{

dection++;

count--;

}

else

{

targetX += 2;

empty\_mazemap[targetX, targetY] = node.pass;

empty\_mazemap[targetX - 1, targetY] = node.pass;

way.Add(targetX + "," + targetY);

unpassnode.Remove(targetX + "," + targetY);

count = 4;

dection = RandomInt(); //方向

}

}

if (dection == 1) //左

{

if (targetY - 1 <= 0 || empty\_mazemap[targetX, targetY - 2] == node.pass || empty\_mazemap[targetX, targetY - 2] == node.truewayproduce)

{

dection ++;

count--;

}

else

{

targetY -= 2;

empty\_mazemap[targetX, targetY] = node.pass;

empty\_mazemap[targetX, targetY + 1] = node.pass;

way.Add(targetX + "," + targetY);

unpassnode.Remove(targetX + "," + targetY);

count = 4;

dection = RandomInt(); //方向

}

}

/////四个方向都检查完毕后

if (count < 0)

{

if (way.Count == 0)

{

Debug.Log("一定是出错了，进入死路但是已经无路可退了");

break;

}

//判断后退节点是否为死路点

CheckPoint(targetX, targetY, ref empty\_mazemap);

//Debug.Log("trueway produce point" + targetX + targetY + empty\_mazemap[targetX, targetY]);

//倒退到前一步

way.RemoveAt(way.Count - 1);

string nodeXY = way[way.Count - 1];

int nodeindex = nodeXY.IndexOf(",", 0);

targetX = int.Parse(nodeXY.Substring(0, nodeindex));

targetY = int.Parse(nodeXY.Substring(nodeindex + 1));

count = 4;

}

}

empty\_mazemap[(endX \* 2 - 1), (endY \* 2 - 1)] = node.pass;

//打通其余墙面

MakeThroughtMap(unpassnode, ref empty\_mazemap);

//返回处理后的墙面信息。

return empty\_mazemap;

}

4.13.2、网络第一人称的区分

在局域网络下不同的玩家控制的第一人称的模型Prefab是相同的但是同一个场景中应该只有一个模型玩家真正控制的模型对玩家的按键进行响应，其他的模型应该忽略该事件命令。

//当客户端模型创建执行脚本

public override void PreStartClient()

{

healthScript = GetComponent<PlayerHeath>();

healthScript.EventRespawn += EnablePlayer;

}

//当服务端模型创建执行脚本

public override void OnStartLocalPlayer()

{

crossHairImage = GameObject.Find("CrossHarImage").GetComponent<Image>();

SetRespawnButton();

}

//当网络断开时执行脚本

public override void OnNetworkDestroy()

{

healthScript.EventRespawn -= EnablePlayer;

base.OnNetworkDestroy();

}

# 单人模式AI系游戏运行与测试

## 测试环境

本游戏是在PC端的游戏测试也是基于PC机上进行测试，首先Unity的编辑平台自带有测试的执行界面。所以游戏第一步需要在Unity的编辑平台上正式跑通后，再进行发布版本的测试。测试环境配置信息如表5-1所示：

表 5‑1测试环境配置信息

|  |  |
| --- | --- |
| **平台**  **属性** | **Mac OSX** |
| CPU | AMD A8-4500.19GHb |
| 内存 | 8G |
| 操作系统 | Windows 7 |
| 存储 | 256G |
| 显卡 | Intel Iris Pro Graphics |
| 屏幕 | 1366 \* 768 |

## 单元测试

单元测试（unit testing），是指对某一个软件中的最小可测试单元进行检验。游戏开发中单元的种类很多如音乐播放中的音乐，模型中的每一个动画，最平常的代码中的摸一个具体的代码功能实现函数。使用Unity操作平台的优点就在这里。在制作编辑器中开发者可以随时进行运行测试。基本上没完成一个小的单元都只需要运行下便可以进行单元测试。而运行的结果反馈又会影响到下一步是回去修改运行表现存在的bug还是执行下一步的运行进行。

## 功能测试

功能测试对游戏而言是必不可少的，而且需要大量的玩家反馈。当收到反馈的时候某些功能可能并不适用需要去掉，某些功能玩家很需要却没有。这些功能的添加删除都和玩家的对游戏功能测试提出有很大的影响。毕竟游戏的最大的优势是用户的体验融入其中，功能对玩家的特和是最大的要求。这里在好友测试下实现了当前游戏的大概功能提供需求完成并对功能完整性进行了测试。

关于界面测试用例，主要测试返回主菜单功能是否正常。关于界面测试用例如表5-2所示：

表 5‑2主菜单界面测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 主菜单界面 | | | | |
| 测试功能 | 测试前提 | 输入 | 期望结果 | 测试结果 |
| 单人冒险模式 | 游戏在主菜单界面 | 1. 点击” 单人冒险模式” | 1. 进入单人模式第一关场景 | 通过 |
| 联机竞技模式 | 游戏在主菜单界面 | 1. 点击” 联机竞技模式” | 1. 进入连机竞技界面 | 通过 |
| 联机合作生存模式 | 游戏在主菜单界面 | 1. 点击” 联机合作生存模式” | 1. 进入新本连机界面 | 通过 |
| 退出 | 游戏在主菜单界面 | 1）点击” 联机合作 | 1）退出游戏 | 通过 |

关于界面测试用例，主要测试返回主菜单功能是否正常。关于界面测试用例如表5-3所示：

表 5‑3返回主菜单界面测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 关于界面 | | | | |
| 测试功能 | 测试前提 | 输入 | 期望结果 | 实际结果 |
| 返回主菜单 | 游戏处于对相应不同模式下界面 | 1. 点击“返回主菜单”按钮 | 1. 返回主菜单 | 通过 |

关于界面测试用例，主要测试返回多人生存模式下复活功能是否正常。关于界面测试用例如表5-4所示：

表 5‑4复活界面测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 关于界面 | | | | |
| 测试功能 | 测试前提 | 输入 | 期望结果 | 实际结果 |
| 复活玩家 | 游戏玩家角色处于死亡 | 1）点击“复活”按钮 | 角色复活 | 通过 |

关于界面测试用例，主要测试单人冒险模式下功能是否正常。关于界面测试用例如表5-5所示：

表 5‑5重新开始有界面测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 关于界面 | | | | |
| 测试功能 | 测试前提 | 输入 | 期望结果 | 实际结果 |
| 重新开始游戏 | 游戏处于运行中 | 1）点击“ESC”按钮，后点击重新开始游戏 | 场景重新加载 | 通过 |

游戏中功能测试用例，主要测试了迷宫生成、迷宫寻路、子弹发射、子弹击中检测、移动动画、AI行为、怪物寻路、目标减血、死亡判断、过关判断、红外线检测、双人竞技模式产生胜者等功能。游戏主界面测试用例如表5-6所示：

表 5‑6游戏功能测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 游戏功能 | | | | |
| 测试功能 | 测试前提 | 输入 | 期望结果 | 实际结果 |
| 迷宫生成 | 单人冒险模式下 | 进入游戏 | 生成迷宫大地图 | 生成迷宫成功 |
| 迷宫寻路 | 单人冒险模式下捡去相应道具 | 第一次捡起道具或点击“Tab”键 | 根据主角当前位置生成寻路导航 | 生成导航成功 |
| 子弹发射 | 单人冒险模式下 | 点击鼠标右键 | 生成子弹并子弹快速向前飞行 | 生成子弹成功 |
| 子弹击中检测 | 子弹存在并在飞行过程中 | 碰撞到物体 | 输出物体的Tag标识 | 输出物体的Tag标识成功 |
| 移动动画 | 创建第一人称控制模型 | 点击WASD控制键 | 控制的角色进行移动 | 控制的角色进行移动成功 |
| AI行为 | 存在怪物在场景中 | 是否发现目标并距离目标的距离 | 没有发现主角或攻击目标静止不动，发现后进行追击，到一定攻击距离进行攻击 | AI表现成功 |
| 怪物寻路 | 单人冒险模式下怪物发现目标追击 | 目标坐标位置 | 如果目标相对怪物在迷宫中只是转了一弯继续追击。否则放弃。 | 追击表现成功。 |
| 目标减血 | 怪物或主角存在血量角色 | 收到伤害 | 血量值减少响应伤害值 | 血量减少成功 |
| 死亡判断 | 怪物或主角 | 血量少于0 | 模型播放死亡动画并摧毁 | 死亡表现成功 |
| 过关判断 | 单人冒险模式 | 找到迷宫出口 | 显示闯关成功UI | 显示成功 |
| 红外线检测 | 单人冒险模式下主角生成 | 红外线击中墙面 | 显示红外线特效 | 显示成功 |
| 双人竞技模式产生胜者 | 双人竞技模式下 | 一方被击败 | 双方都显示游戏结束界面 | 显示成功 |

通过上列测试用例的测试，证明游戏已经能够响应用户的操作并做出正确的处理，说明游戏已经能稳定运行。

**结语**

本游戏的设计和开发是在2016年11月开始的，经过一个月的忙碌完成了当前版本的射击游戏，虽然大体功能已经实现并完善但是如果要发布到市场上还需要一定的功底打磨。一年来的使用Unity的开发让我找到了人生前进的目标，单机游戏的开发。虽然目前国内的单机游戏的发展市场一直比较低迷但是它正在变好，而游戏在我看来是艺术的一种形式。好的游戏完全可以作为第九的艺术的存在。而我一生的追求可能就是这样的第九的艺术。接触的好游戏越多，越是发现现在的自己的不足。希望未来我能做出我和大家都认可的游戏。

毕业设计，也许是大学生涯中最后一份作业。这份作业却让我学到了四年中最丰富、最特别的知识：从对游戏引擎的一窍不通，到现在能够开发简单游戏并部署；从对Office的陌生，到现在能够编写规范的文档；从对英文阅读感到力不从心，到现在基本能够独立查阅英文文献。整个过程经历过失败，经历过浮躁，也经历过成功的喜悦。这最后一份作业，是我对自己大学生涯交上的一份满意的答卷。

**参考文献**

[1] unity官网解释什么是Unity ：<https://unity3d.com/cn/unity>

[2] 迷宫中的算法实践]迷宫生成算法——Prim算法<http://www.cnblogs.com/JiaoWoWeiZai/p/5890379.html>

[3] Unity3d 引擎原理详细介绍： <http://blog.csdn.net/jbjwpzyl3611421/article/details/10441681>

**致谢**

本游戏《第一人称射击游戏》的设计与开发，是在成都信息工程大学软件工程学院铁菊红老师的细心指导和严格要求下完成的，从课题需求分析，概要设计到详细设计与开发实现，所有工作的顺利进行都离不开铁菊红老师的关心和指导。感谢铁菊红老师给予本人以课外技术完成毕业设计的机会。

本毕业设计的顺利完成，还离不开软件工程学院的各位老师在平时授课过程中的认真负责，让本人拥有足够的软件基础知识和充分的自学能力。同时也感谢为自己提供游戏见解、游戏策划、技术指导、文献翻译以及游戏测试的所有同学和朋友，他们为本作《第一人称射击游戏》的完成做出了不容忽视的贡献。

最后，向各位不辞辛劳，认真负责的论文评阅老师致以衷心的感谢！

作者简介：

姓 名：陈宗林 性别：男

出生年月：1994年10月28日民族：汉

E-mail:874944424@qq.com

**声明**

本论文的工作是2016年11月至2016年12月在成都信息工程大学软件工程

学院完成的。文中除了特别加以标注地方外，不包含他人已经发表或撰写过的研究成果，也不包含为获得成都信息工程大学或其他教学机构的学位或证书而使用过的材料。

关于学位论文使用权和研究成果知识产权的说明：

本人完全了解成都信息工程大学有关保管使用学位论文的规定，其中包括：

（1）学校有权保管并向有关部门递交学位论文的原件与复印件。

（2）学校可以采用影印、缩印或其他复制方式保存学位论文。

（3）学校可以学术交流为目的复制、赠送和交换学位论文。

（4）学校可允许学位论文被查阅或借阅。

（5）学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

除非另有科研合同和其他法律文书的制约，本论文的科研成果属于成都信息工程学院。

特此声明！

作者签名：

年月日