基于 U3D 的生存射击游戏的设计与实现

Design and Implementation of Survival Shooting Game Based on U3D

专　　业：计算机科学与技术（信息处理）

姓　　名：周杨博伦

指 导 教 师：

申请学位级别：学 士

论文提交日期：2018 年 6 月 4 日

学位授予单位：天津科技大学

摘　要

根据调查，现代社会，人们对于娱乐性的电子游戏的需求越来越大，其市场也越来越收到关注，随着游戏引擎的出现与发展，制作游戏的过程也变得更加简便，借助游戏引擎开发一款生存射击类游戏，既是对现有市场的呼应，又是对制作理论的实践。

本课题着力于制作一款游戏，根据目的与本质，抽象出的物理方法的编写与实现。在归类分析后，将生存射击游戏分为了关卡制作，综合控制与实时检测三部分，其中包含了地形制作，导航烘培，节点设置，“wsad”4 方向操控，360 度鼠标转向，动画切换，媒体控制，位置追踪，碰撞检测，类型检测，状态检测等部模块，结合 UI 面板进行反馈，并最终完成游戏的制作。

本游戏在制作过程中，主要依托 Unity 引擎与 C#语言编写的脚本，大量使用了引擎附带的 Asset Store 商店中的免费素材，包括 3D 模型，材质，音乐等。借助 NavMesh 导航烘培技术，实现了位置捕捉与路径追踪，physX 物理引擎完成碰撞的检测与处理。

关键字：生存射击类游戏； Unity； C#

ABSTRACT

According to the survey, in modern society, people's demand for entertainment-oriented video games is increasing, and their markets are receiving more and more attention. With the advent and development of game engines, the process of making games becomes easier with the aid of game engines. A survival shooting game is both a response to the existing market and a theoretical practice.

This project focuses on making a game, according to the purpose and nature, the abstract preparation of physical methods and implementation. After the classification analysis, the survival shooting game is divided into three levels: the level production, the integrated control and the real-time detection. It includes terrain production, navigation baking, node setting, "wsad" 4-direction control, 360-degree mouse rotation, and animation switching. , Media control, position tracking, collision detection, type detection, status detection and other modules, combined with UI panel feedback, and ultimately complete the production of the game.

In the production process, the game is mainly based on the scripts written in the Unity engine and the C# language, and the free use of the free materials in the Asset Store included with the engine includes 3D models, materials, and music. With NavMesh navigation bake technology, position capture and path tracking are achieved, and the physX physics engine completes collision detection and processing.

KEYWORDS: Survival shooting game; Unity; C#

目　　录

第一章 前言

第一节 背景调查

一、 课题来源

现代社会，我们无时无刻不与游戏（此处特指电子游戏，下同）打交道，无论是小到儿童大到成人手机里经常打开的“王者荣耀“、”荒野行动“等手游，还是运行在次世代主机上搭配有绚丽的画面，丰富的剧情，震撼的音效的大型 3A 游戏，如”神秘海域“、”塞尔达传说“等。我们比从前任何一个时候都要熟悉这些新型的娱乐产品。计算机技术发展迅猛，新型的 VR，AR 也为游戏铺展了新的道路，可以遇见，未来游戏作为人类娱乐与艺术的一门形式，发展前景是非常好的。然而游戏的实现往往比它外表看起来复杂的多，每个优秀游戏的背后，都必然充斥了大量的学科交叉知识，一个优秀的游戏部门也往往分工明确，有协调管控整体游戏品质的游戏制作人（游戏导演），有负责剧情的文本编剧，有奠定美术风格的原画和监督，有将美术具象化的建模师，地形师，有实现游戏逻辑的程序人员，有增添氛围的音乐家和音效师，甚至还有负责专门测试的测试人员。在以前，一门游戏的完成，几乎不可能一人独自完成，但随着时间的发展，游戏引擎出现了，借助游戏引擎的平台，即便是个人也可以完成游戏的制作，于是越来越多的独立游戏（Indle Game）出现了。开发者有的是计算机专业出身，有的是美术专业出身，甚至还有作家出身，但是只要对各个领域有所了解，不需要很深入的理解实现原理，我们仍然可以完成这项跨学科的工作。

有玩家就有开发者，有开发者便有了开发工具。现如今，游戏引擎经历了长足的发展后，已经形成了非常规范的商业化体系，与开发者共赢的开源免费，盈利分红的营收方式已铺展开来，较为出名的有“Unity”、“Unreal Engine”，“Cry Engine”，“cocos2d”等，其中“Unity”引擎以多平台，兼容性强，操作简单等优点在国内以及世界都占据了非常大的市场，其内置的“Asset Store”市场已经积累了大量可供直接调用的素材（收费与免费），对于想要快速将自己构想游戏实现的开发人员来说是不可多得的利刃。

二、 研究目的及意义

在这里本人想就如何基于这一专一引擎进行一款简单的生存射击游戏进行研究，着力于一款游戏的实现应该如何划分阶段与完成，根据游戏的目的与本质，抽象出物理方法的编写与实现，进行游戏机制的完善与测试。其实电子游戏作为软件的一种，其开发与普通软件也是大同小异，都要经过 PDCA 的反复完善，最大的不同便是应用程序的开发往往很容易根由现实问题得出软件的目的，而游戏则宽泛的多，玩法与目的都取决于制作人的想法，但除却剧情画面等与艺术相关

的部分，任何游戏基本都离不开“车枪球谜“，即赛车，射击，踢（打）球，解谜。通过对生存射击类游戏的设计与实现，可以了解到游戏中基数很大的类型的游戏背后，逻辑究竟是怎样实现的，而实现这些逻辑的方法和模型，对今后制作同类游戏，混合类游戏，也有着相当的参考价值，修改套用，甚至是直接移植也都是有可能的，毕竟游戏的画面可能日新月异，好玩的剧情层出不穷，但是游戏玩法背后逻辑的实现却很少变动，甚至可以说是成熟后已经没有再改变了。因此以生存射击类游戏的设计与实现为题既是对典型游戏的再现也是符合时代发展方向的课题。

三、 生存射击类游戏发展概况

（一） 国外发展状况

3D 生存射击类游戏在国外可以说是屡见不鲜，它的发展历史也是相当的久远。普遍认为，第一款该类游戏为 1993 年 12 月 10 日由 id 软件开发的 DOS 平台游戏“毁灭战士”，它既是该类游戏的先锋，也是该类游戏的代表，基于其绚丽的彩色画面，虽然只是简单的射击冒险游戏，但依然在只有黑屏白字的当年大放光彩，随后各大厂商纷纷开始在画面着手，从最初的 8bit 像素发展到现在，画面已经足够逼真，同时 VR 射击游戏也应其而生。

随后出现了注重个人英雄主义塑造的 “毁灭公爵”、注重画面与操控感的“雷神之锤”，这两款游戏都有一个共同特点，逼真的物理引擎，物理引擎的加入令该类游戏直接迈上了更高的台阶，有碰撞，有重力，似乎一切都和真实世界非常相像，你可以打碎玻璃，打碎墙，和游戏环境做互动。生存射击类游戏随后便开始了物理引擎的精细与改进，材质，摩擦，风，体积雾，光线等等，经过不断的发展，现代游戏的物理引擎已经非常强大了。

2008 年由维尔福公司发行的游戏“求生之路”在当年掀起了不小的浪潮，没有眼花缭乱的武器，没有千奇百怪的敌人，游戏用抽象过后的精简模型让玩家感受到，即便简单，乐趣丝毫不会衰减，固定的武器类型，敌人类型，目标就是从地图一端逃到另一端，游戏依靠出色的关卡设计独树一帜，为了做好关卡，开发者们甚至为该游戏编写了专门的词典，如“牢房”一词便是形容密闭空间内面对多条路线攻来的潮水般多的敌人防守一定时间的游戏部分等等，关卡设计发展到近年已逐渐演变为沙盒，多个关卡被整合在一张巨型的地图中，玩家可以自由决定先游玩哪个部分，也就是说，整张庞大的地图变味了一个巨型的关卡，而如何行进游戏流程可由玩家自行决定，可以说现代游戏的自由度变得更加高了。

（二） 国内发展概况

国内生存射击类游戏的发展相对来说较为曲折，在上世纪电子游戏发展的时代，国内第一批制作人受到海外舶来的如“魂斗罗”，“潜龙谍影”等游戏的震撼，准备制作中国人自己的射击游戏，于是，在当时和国外很火的一线游戏质量比肩的“大秦悍将”、“抗日，血战上海滩”等游戏横空出世了，玩家口碑也很好，也

十足风靡了一段时间，但是当时国内整体对于游戏产品的版权意识不强，致使盗版现象肆虐，极大的打击了国产游戏产业，许多创业公司往往推出一款产品后便草草转行，离开了这片贫瘠的市场。加至后来“传奇”等网络游戏的到来，国产生存射击类游戏进入了长久的沉默。

在这段时间里，依靠盗版吸引了大量用户的美国维尔福公司的“反恐精英“几乎占据了网吧的每一寸土地，由于国外游戏的收入不依靠国内，致使脍炙人口的优秀生存射击游戏均为外国制作，国产游戏更难有再起之力。于此同时，生存射击类游戏因其特殊的玩法和模式，在国内没有引入分级制度的环境下，确实不利于年龄过小的少儿接触，与国内文化氛围向左更导致了国产生存射击类游戏的制作雪上加霜。

进入 21 世纪后，由国内电子巨头腾讯引进的韩国射击竞技网络游戏“穿越火线”在国内大火后，国内游戏厂商似乎看到了新的希望，国内生存射击游戏便开始以一种新的姿态重生，即主打网络，对战，竞技，淡化剧情等不重要的要素，于是金山游戏的“反恐行动”，腾讯游戏的“逆战”等，均吸引了不少用户，腾讯更是依托庞大的用户群体以手机游戏“全民突击”发现了移动平台的强大活力。至 2017 年，由韩国 PUGB 公司推出的“绝地求生”游戏在中国突然火爆了起来，似乎一夜之间，许多人都热衷上了这种主打生存，社交，对战的游戏，于是国产厂商似乎抓到了新的出发点，由腾讯推出的“刺激战场”，“全军突击”，网易推出的“荒野行动”等均取得了不错的佳绩，生存射击类游戏大有再次复兴的趋势。

第二节 软件开发技术介绍

一、 Unity 引擎

Unity 是由 Unity Technologies 公司开发的专业跨平台游戏开发及虚拟现实引擎，其打造了一个完美的跨平台程序开发生态链，用户可以通过它轻松完成各种游戏创意和三维互动开发，创造出精彩的游戏和虚拟仿真内容。用户也可以通过 Unity 资源商店（Asset Store）分享和下载各种资源。

作为一款国际领先的专业游戏引擎，Unity 精简、直观的工作流程，功能强大的工具集，使得游戏开发周期大幅缩短。通过 3D 模型、图像、视频、声音等相关资源的导入，借助 Unity 相关场景构建模块，用户可以轻松实现对复杂虚拟世界的创建。

Unity 编辑器可以运行在 Windows、Mac OS X、以及 Linux 平台，其最主要的特点如下：一次开发就可以部署到时下所有主流游戏平台，目前 Unity 能够支持发布的平台有 21 个之多。用户无须二次开发和移植，就可以将产品轻松部署到相应的平台，节省了大量的开发时间和精力。在移动互联网大行其道的今天，Unity 正吸引着越来越多人的关注。

在虚拟现实应用及手机游戏大行其道的今天，Unity 正日趋受到移动开发者的青睐。在中国，游戏行业及虚拟现实行业对 Unity 引擎的关注度也正日益加深，采用 Unity 引擎开发的移动游戏、网页游戏以及端游仍在不断推出。在中国有 100 多万注册开发者，中国前 20 名游戏公司都在使用 Unity 进行项目的开发。使用 Unity 开发者遍布中国 200 多个城市，约 60%针对移动开发端，40%针对 PC 开发端。最新的 2018. 1 版本新应用的 Magic Leap 工具、可编程渲染管线（SRP）技术，Shader Graph 可视化着色器编程技术、GPU 加速的渐进式光照烘培、机器学习等技术使得引擎发展趋势被外界开发者及媒体一致看好。

二、 Asset Store

全称 Unity Asset Store，中文译名即 Unity 资源商店，是 Unity 的一个重要组成部分，它的官方网址为 http://www.assetstore.unity3d.com/。可以通过在浏览器地址栏输入网址访问，也可以在 Unity 应用程序中直接访问。

在创建游戏时，通过 Asset Store 中的资源可以节省时间、提高效率。包括人物模型、动画、粒子特效、纹理、游戏创作工具、音效特效、音乐、可视化编程解决方案、功能脚本和其他各类扩展插件全都能在这里获得。作为一个发布者，你可以在资源商店中出售或者负责提供你的资源，从而在广大 Unity 用户中建立和加强知名度并取得盈利。

值得一提的是，Asset Store 还能为用户提供技术支持服务。Unity 已经和业内一些最好的在线服务商开展了合作，用户只需下载相关插件，便可获得包括企业级分析、综合支付、增值变现服务等在内的众多解决方案。

三、 PhysX 物理系统

PhysX 是一款专用于处理实时物理动作的引擎开发组件。其最初于 2004 年被 NovodeX 开发小组写成，后被 Ageia 公司收购，并最终于 2008 年随着 Ageia 公司被 Nvidia 公司收购而被并入。

PhysX 是一个面向 Windows，OS X，Linux，PlayStation 3，Xbox 360，Wii 等主流平台的 3D 物理模拟开发套件。其支持动态刚性躯体，动态柔性躯体，布娃娃和人物控制，动态车辆，颗粒、液体体积模拟以及对服装撕裂以及加压的模拟等功能。

因其对物体物理行为出色的模拟特性，PhysX 科技被广泛的用于游戏引擎，例如 Unreal Engine，Unity，Gamebryo 等。作为最易于操控的主流物理引擎，其也被广泛用于许多广受好评的游戏，如“镜之边缘”，“极品飞车”，“爱丽丝梦游仙境”等。

四、 C#语言

C#是微软公司推出的一种基于.NET 架构的、面向对象的高级编程语言。C#以.NET 构架类库作为基础，拥有类似 Visual Basic 的快速开发能力。其被旨在设计成为一种“简单、现代、通用”，以及面向对象的程序设计语言，可以完成强

类型检查、数组维度检查、未初始化的变量引用检测、自动垃圾收集等工作，强大、持久，并具有较强的编程生产力是它的一大特点。

C#的可移植性很高，对于已熟悉 C 和 C++的程序员而言更甚，因此在 Unity 引擎由 C++编写，API 由 C#编写的前提下，C#语言作为 Unity 引擎的脚本语言之一在版本的发展中逐渐脱颖而出，而另外两种解释性语言 Boo 和 JavaScript 因与开发不相适合而分别于版本 5 和版本 2017. 1 被抛弃，C#成为了现在 Unity 唯一的脚本语言。

第二章 需求分析和概要设计

第一节 玩法分析

游戏的世界里一切都是数据。每时每刻，这些数据都在不断变化着，演绎出精彩纷呈的游戏状态。不同的游戏不同的奖励目标刺激着玩家不同的欲望，调动着玩家的游玩心情，不同的玩家喜好不同，但谁不喜欢更多呢？

游戏发展至今最原始也最基本的奖励机制就是得分，得分越高，满意度也就越高，这与人不断超越自我，发展自我的本性相关，本游戏应当能满足生存，射击两方面的要求。

生存，即在面对越来越多的敌人攻来的情况下，为自己开辟出一条生的道路，射击，即通过使用手中的枪械，向敌人倾泻你的怒火。抽象一点来看，本游戏必须能走能打，还能因受到伤害而死亡，再加上之前提到的奖励机制，通过射击消灭足够多的敌人，获得更高的分数，当然，如果分数不能保留，游戏在本次结束后也就没有了意义，所以还需要能记录分数，最好能不断的挑战自我。

第二节 系统设计

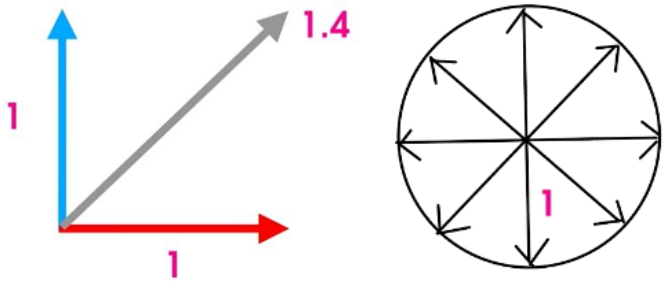
一、 行动逻辑

生存射击类游戏的本质，就在于生存，即存活下去，游戏往往随着人物的死亡而结束，而相应的，存活越久，奖励也就越加丰厚。本游戏需要有可移动的人物，可移动的怪物，而两者都有生命值，可以被消灭，而本游戏的奖励系统显然显而易见，在人物死亡前，消灭更多的敌人。所以怪物死亡时应该伴随着计分系统中分数的积累。而人物消灭怪物的手段当然是射击，而为了游戏的可玩性，怪物将不具备射击功能，相反选择寻求近距离伤害并最终消灭人物的手段。基于本游戏的需求，时下非常流行的“wsad”移动与 360 度转向是比较符合需求的。

（一） wsad 移动

标准化的 wsad 操控因为非常普遍的存在于游戏中，Unity 已经默认做好了键盘读取工作，本游戏可以简单的从标准化输入中的 Horizontal 变量与 Vertical 变量中读取到四个键值，只需要将即时读入的输入带入一个新的三维变量并在每帧游戏刷新时替代人物当前位置即可。

不过这里还有一个问题，假设横纵方向移动单位均为 1，如果同时按下，人物应当可以进行斜向移动，但是根据计算，斜向移动实际为 1. 41 左右，如图 2-1 所示。



(a) (a)

图 2-1 位移的标准化

而本游戏的实际希望是，上下左右四方向，斜向四方向均为 1，这样移动才更贴合实际，所以实际控制移动时，本游戏需要标准化位移量，这里可以使用三维变量类中的标准化方法。

另外，在本次游戏制作中，想突出平面移动的灵活性，所以跳跃以及任何 y 轴方向的移动都会被尽量避免，也就是说人物的 y 轴将会被锁定在 0。

（二） 360 度转向

本游戏可以移动，同时也需要能转向，毕竟如果只把射击方向锁定在人物面对方向，游戏体验性会不好，本游戏的人物可以倒退射击，边移动边向自己想要的方向射击。所以需要能获取想要的方向。

因为左手已经用于“wsad”移动，所以右手控制的鼠标进行转向射击是最为合适的，根据鼠标的移动进行侦测是最佳的方案，这里选用在地面放置透明遮罩进行实时检测，这里可以使用 Unity 中的焦点来完成。

游戏实际画面实际上是主摄像机的第一人称视角，当然摄像机是在空中的，所以有一个广阔的视角，本游戏可以从摄像机向鼠标输入所在位置创建焦点，两点确定一条直线，因此这个焦点会打在地面遮罩上，通过检测这个焦点的坐标，来控制人物的转向，如图 2-2 所示。

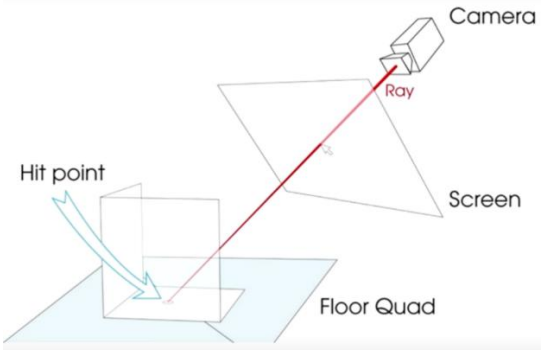


图 2-2 焦点转向的原理

（三） 动画变动

人物移动和站立时的动画肯定是不同的，不然就好像平移射击，游戏的真实

性会大大降低，所以本游戏需要有一套动画系统来控制动画的变更，根据游戏简单的逻辑，人物只需要有站立，移动，死亡三个动画即可满足基本需求，还好强大的 Unity 引擎包含了 Mecanim 动画系统，本游戏可以很轻松的完成动画的切换，本课题采用的人物素材动作骨干 Avater 及动画片段 clip 均已制作好，只需要关注变换逻辑即可，如图 2-3 所示。

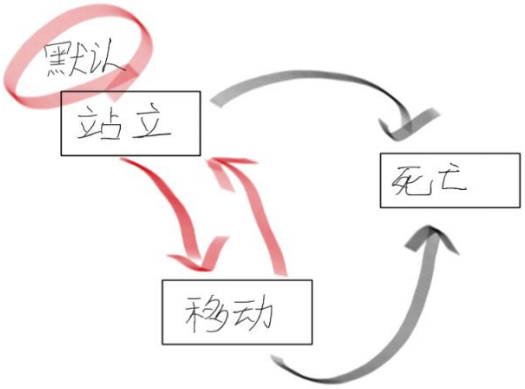


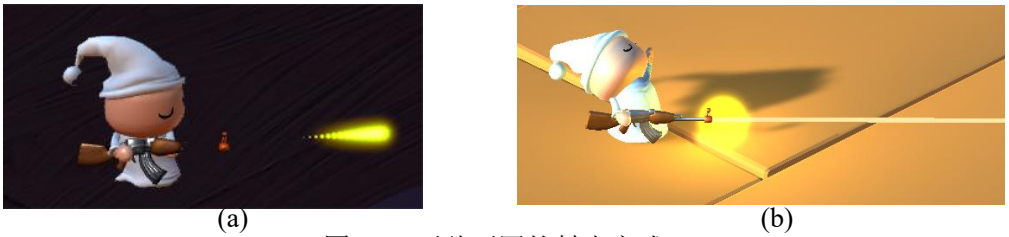
图 2-3 动画转换的控制

这里准备两个变量，一个变量用于判断人物是否在行走，如果是便从站立切换至移动，如果不是便从移动切换至站立，另一个则用于判断是否死亡，死亡后不再变换动画。

另一方面，怪物的移动与人物有着很相似的行动逻辑，可以直接借用相同的动画变动系统，Mecanim 中的控制单元可以 override 这一特点正好可以利用。

（四） 射击系统

关于射击系统，以往的游戏有两种解决思路，一种是射击与子弹分离，每次射击会创造一个子弹物体，其会向固定方向移动，有速度，有体积，可以检测碰撞，并销毁，另一种是射击一体，即指哪打哪，射击会检测面前的直线，通过第一个返回的物体的类型进行判断是否命中，并描绘出子弹轨迹，这种方法被广泛使用于对战游戏中，如图 2-4 所示。



(a) (b)

图 2-4 两种不同的射击方式

由于此次只是制作射击游戏，并不需要多种不同的射击环境，所以本游戏在这里采用第二种射击方案，同时也能减少资源消耗，让小游戏更轻量化。

关于射击，本游戏还需要考虑整个游戏环境中那些物体是可以射击的（障碍物，地图边界，敌人），那些是不可以的（粒子效果，物品），Unity 引擎内有 Layer

层的概念，可以设定多达 12 个层，在这里只需要将可被射击的物品层设为预先设定好的名字，之后即可在脚本中调用。

（五） 敌人追踪

本游戏的需求是敌人会不断的向控制的人物冲来，试图近距离消灭人物，而本游戏要做的就是让敌人实时检测人物的位置，并寻找最短路径跑向人物，并且要求在路途中可以自动绕开障碍物，而不会被卡住，这里可以使用 Unity 的 Nav mesh 导航窗格来实现。

导航窗格的原理是基于已经确定的环境，提前计算所有的路径地图，可以在计算前确定，计算半径，计算高度，步幅，安全距离，模糊度等数值，Unity 称这一动作为烘培。烘培之后，所有的 Unity 物体都可以借助 Nav mesh agent 部件调用路径，以完成自动导航（即追踪）的效果，如图 2-5 及图 2-6 所示。

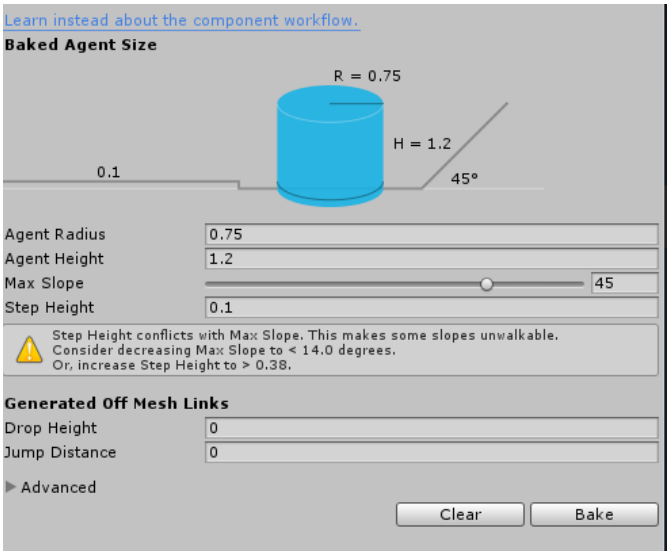


图 2-5 烘培参数界面

图 2-6 一张烘培好的地图导航遮罩

（六） 摄像机跟随

本游戏的游戏类型决定着游戏的人物一定是在不断移动的，所以摄像机要时刻跟随游戏的主角是非常重要的，在 Unity 中，摄像机有两种视角，透视型和正视型，透视型比较像现实生活中正常人的视角，即由一点视线向外发散；而正视型，则比较像上帝视角，由屏幕向一点收紧（这一点往往位于游戏世界非常远的地点），如图 2-7 所示。

前者多用于第一人称，第三人称（跨肩视角）游戏，而后者多用于战略游戏，在本次的案例中，因为射击移动均在平面内进行，所以选用正视型是比较稳妥的选择。

(a) (b)

图 2-7 两种不同的摄像机视角

二、 信息交互

绝大多数游戏都需要与玩家有一个信息交互与反馈，比如，游戏人物还有多少生命值，能承受多少伤害，人物受伤时有什么表现，敌人被击中有什么表现，玩家得到了多少分数，游戏是否已经结束等，这些信息必须得以在游戏与玩家之间频繁交互。

（一） HUD 信息面板

所有的状态都可以基于一个 UI 套组来实现，Unity 中的新 GUI 套件与游戏一般的实体不同，他会基于用户的显示画面分布数值和类型。

本游戏需要一个生命值显示栏来追踪人物当前的健康状况，放在画面的边缘显然是比较合适的。本游戏还需要得分栏，与生命值显示栏道理相同。

（二） 受伤反馈与游戏结束

本游戏中应当有 3 种反馈：人物的受伤，怪物的受伤（被击中），游戏宣告结束。恰好这三者都可以 Unity 中以不同的方式解决。

人物的受伤：大多数游戏在描绘人物受伤时都会采取屏幕闪烁红色或红色域贴图并辅以声音震动等来反馈。这里可以用 UI 绘制一个覆盖屏幕的全红遮罩，默认为全透明，在人物触发受伤方法时，调用参数修改，在极短的时间内显现并消失，并播放受伤音效达成想要的效果。

怪物的受伤：可以注意到本游戏使用的是射击的方式伤害敌人，对于物理碰撞来说，敌人的物理躯体上一定有一个命中点，只需要在哪一个点创造一个已经预制好的粒子效果即可。

游戏宣告结束：很显然，玩家控制的人物一旦死亡，本游戏便会结束，可以制作一个简单的结束动画，并在播放期间，进行记录当前分数，修改排行榜等后台操作，并在一段事件后重新开始游戏，在这里可以使用 Unity 自带的 Animation 制作自己想要的动画片段。

三、 用户菜单

游戏的主体逻辑与信息交互已经渐渐显现，但是仍称不上一个完整的游戏，本游戏需要一个菜单系统来提高一下用户的自由性，可以暂停，退出，调整音乐音效大小，查看排行榜，这就要求本游戏需要一个单独的菜单系统。

（一） 菜单暂停二合一面板

因为本游戏非常简单，显然大费周折单独设计一个带独立场景的菜单栏是不太合适的，可以将菜单与暂停面板合在一起，在这个面板上可以调整音量，调整音效，可以查看排行榜，可以开始或继续游戏，如图 2-8 所示。

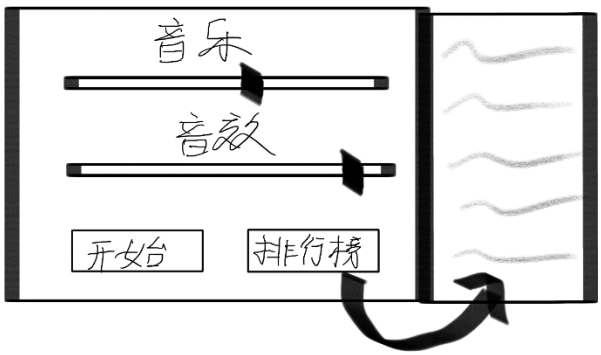


图 2-8 菜单与排行榜面板设计

（二） 排行榜面板

做一个可以记录名次的排行榜，让玩家反复挑战，增加可玩性，这也是很重要的，如果游玩一局游戏后什么都留不下，玩家很快会丧失兴趣。排行榜的 UI 制作并不困难，难点在于算法。

这里本游戏使用 Unity 中用于保存用户设置的 PlayerPrefs 类来储存记录，在新玩家第一次玩时，先检测是否有数据，如果没有就初始化几行行 0 分数据，之后在结束时遍历所有纪录分数进行比较替换并排列。

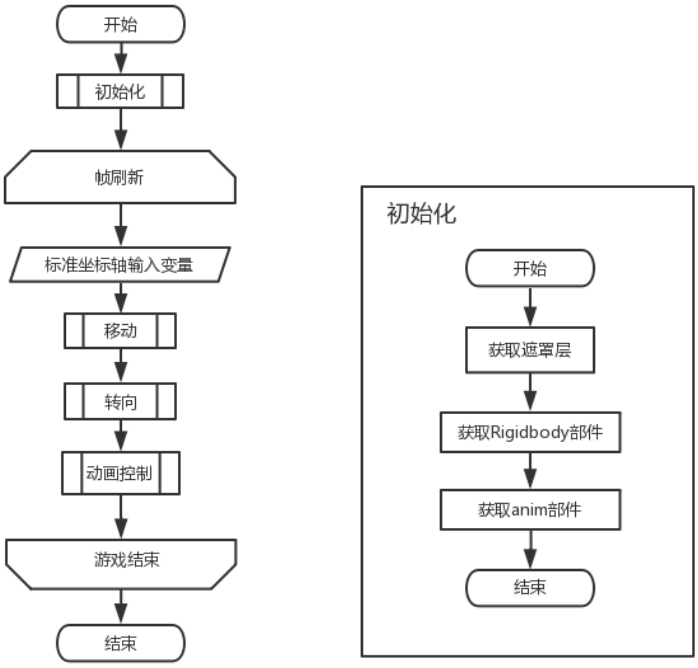
考虑到排行榜单独制作版面太费周折，可以采用折叠方式隐藏在菜单面板时，点按按钮显示或隐藏是不错的选择，如图 2-8 所示。

第三章 详细设计

第一节 人物移动

一、 总体逻辑

通过向人物挂载 Rigidbody 刚体部件作为物理操控的入口，anim 部件作为动画操控的入口，并将两个部件在人物物体初始化时做链接，在游戏帧刷新结束时通过修改参数实现实时更新，通过获取标准输入的水平与垂直输入作为参数代入以完成方法需求，主要分为三部分，位移坐标改变，转向坐标改变，动画更新检测，如图 3-1 所示。



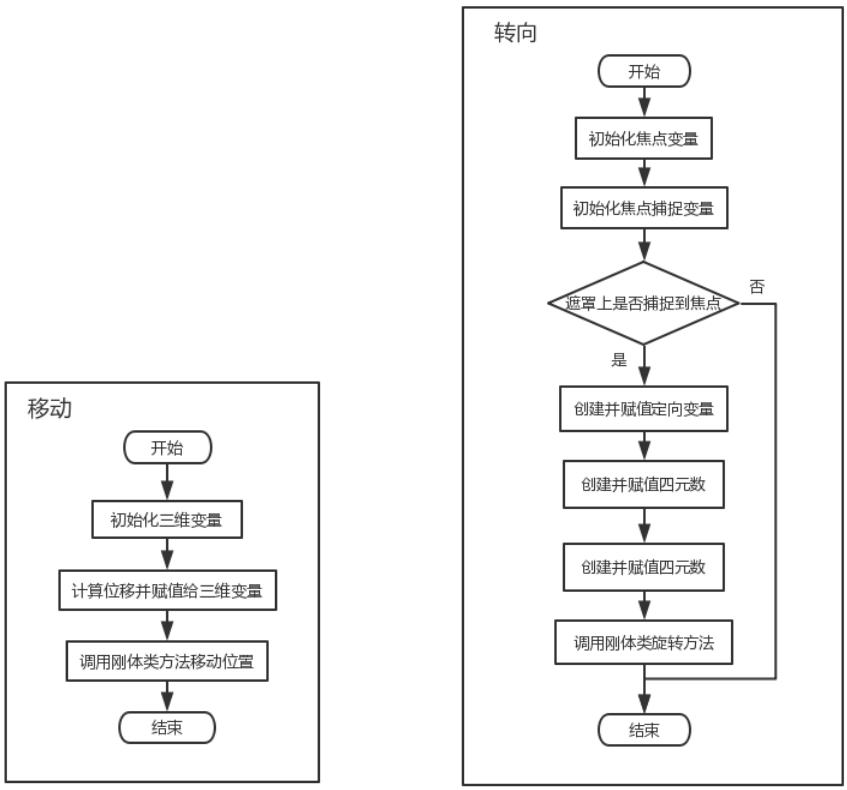
(a) (b)

图 3-1 人物移动主流程图与初始化子流程图

二、 子过程流程图及说明

（一） 移动

位移坐标改变，根据第二章设计时提到的标准化移动，首先初始化一个三维变量用来存储和标准化位移量，水平变量与垂直变量通过调用时传入的参数提供，Y 轴锁定为 0。位移量为速度与时间的乘积算得，为了方便后续调整，速度为公开的浮点型变量，时间使用实际现实世界计时模式。将位移量提交刚体类的移动方法完成移动，如图 3-2(a)所示。



(a) (b)

图 3-2 移动子流程图与转向子流程图

（二） 转向

转向坐标改变，同样根据第二章设计，首先创建一个焦点变量获取和储存玩家的鼠标指向，再新建一个焦点捕捉获取变量用于储存实际与摄像机位置交得的地面遮罩位置。通过 Physics 类方法中的捕捉焦点判断是否有转向需求，并于合适时，新建一个三维变量储存遮罩位置与人物位置之间差的之间变量用于定向，并创建一个新的四元数变量传入定向变量完成方向确认，最后将四元数提交刚体类的转向方法完成转向，如图 3-2(b)所示。

（三） 动画控制

动画更新检测，初始化一个布尔型变量用于判断是否有移动方法在执行，通过传入的参数是否为 0 决定变量赋值，并根据该变量决定是否操纵 anim 部件内链接的动画控制单元 trigger 的设定与否，如图 3-3 所示。

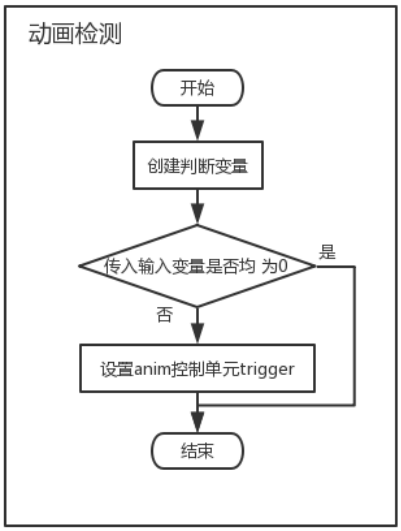


图 3-3 动画检测子流程图

第二节 人物射击

一、 总体逻辑

将空物体位置定于枪支模型的枪口部位作为实现用的平台完成与模型渲染的分离，挂载 Line Renderer 部件用于描画射击时子弹的轨迹，Partical system 部件用于模拟开枪时的粒子效果（枪口火焰），Light 部件用于模拟开枪时的火光，Audio Source 部件装载枪声以便调用，默认关闭 Line Renderer 与 Light，仅在射击方法被调用时开启并于处理完毕后结束以等待下次调用。

初始化时，将部件与做脚本链接，在游戏帧刷新时，通过检测标准输入鼠标是否有值，以判断是否调用开火方法，并做一个计时，在开火时间结束后调用一个方法来关闭所有效果以停止开火，如图 3-4 所示。

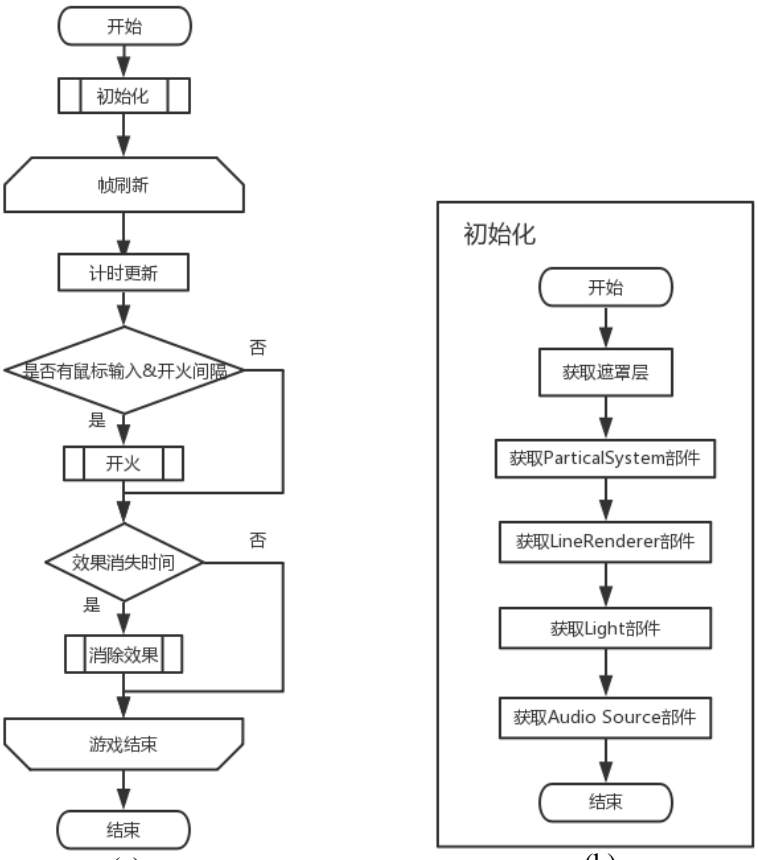


图 3-4 人物射击主流程图与初始化子流程图

二、 子过程流程图及说明

（一） 开火

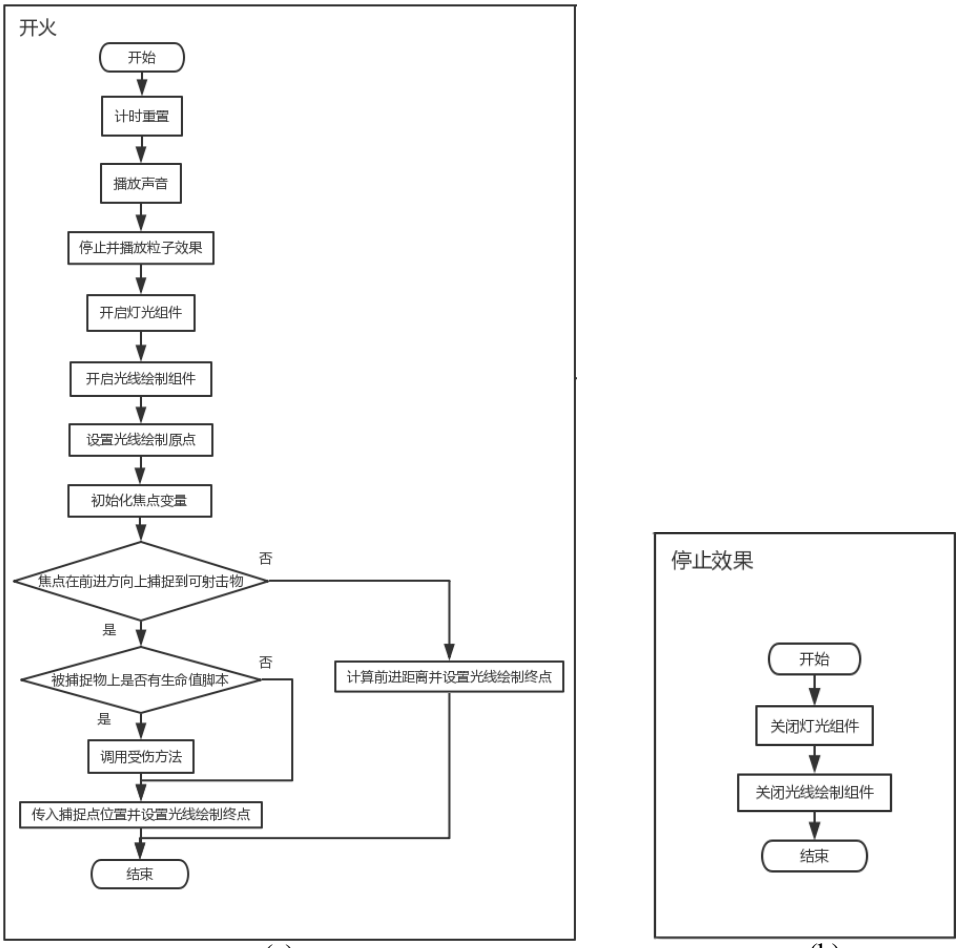
开火被触发时首先应该重置计时器，以便后续时间计算。

首先播放挂载的开火声音，因为该组件默认是不循环的，所以枪声只会短暂的播放一次。

接下来是开火的粒子效果，因为效果的播放时间有长度，为了流畅的播放，先做一次停止，再做一次播放。

接着是子弹轨迹也就是 Line Renderer 描画的光线，将该组件状态设为活跃，描画位置与轨迹需要分情况讨论，一种是射击至可被射击物，描画起点为枪口，终点即为被射击物，这里再次用焦点实现，焦点原点为枪口，方向为正前方，如果捕捉到焦点在射程内碰到了可被射击物的遮罩，则对被射击物的生命值做获取，有的话做调用敌人生命值脚本内的做伤害方法，没有的话便作罢，最后描绘光线至捕捉点以保证与射击位置一致。另一种情况是没有射击到可被射击物，这种情况简单许多，只需要向焦点方向画一条直线即可，距离为设置好的射程，由于摄像机视角有限，只要超出视角，玩家便看不到光线，所以不需要担心画面外的部分。

开火的程序详细流程图如图 3-5(a)所示。



(a) (b)

图 3-5 开火子流程图与停止效果子流程图

（二） 停止效果

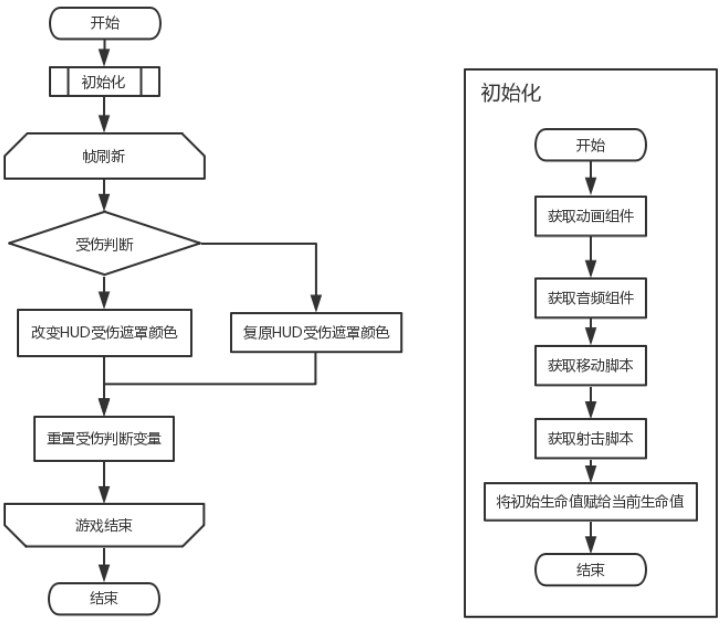
停止效果方法非常简单，它在计时合适时被执行，由开火间隔时间与效果持续变量共同控制，将光线组件与灯光组件的状态重新设为非活跃，如图 3-5(b)所示。

第三节 人物生命值

一、 总体逻辑

生命值脚本主要处理两个问题，一个是受伤，一个是死亡。

由于生命值的变动同样会影响到 HUD 信息的显示及游戏结束的动画播放所以该脚本会在初始化时链接到 HUD 中的生命栏及受伤图片遮罩，一段死亡音频，以及同物体下的音频组件，移动脚本以及射击脚本以便于在死亡方法被执行后锁定玩家的移动与射击。在游戏帧刷新时，通过受伤方法中的判断变量决定是否改变 HUD 的受伤图片遮罩颜色并重置判断变量。受伤方法由敌人攻击时脚本触发，本身不会自动执行，同时受伤脚本的最后会对人物受伤后进行计算，以决定是否执行死亡方法，如图 3-6 所示。



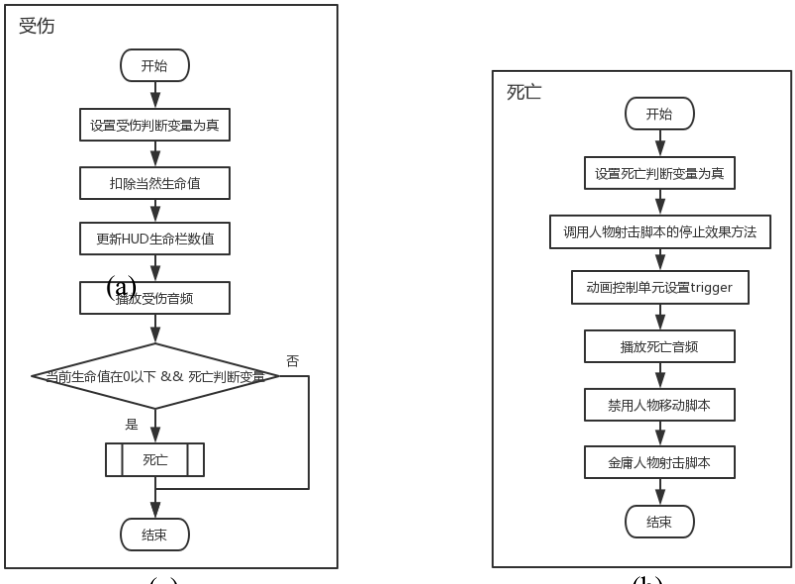
(a) (b)

图 3-6 人物生命值主流程图及初始化子流程图

二、 子过程流程图及说明

（一） 受伤

受伤方法在被调用时会先将判断变量设置表示已受伤以方便帧刷新时检测，扣除收到的伤害，更新 HUD 中生命栏内的数值，播放受伤的音频，并判断人物是否会死亡或者已经死亡，以此决定是否执行死亡方法，如图 3-7(a)所示。



(a) (b)

图 3-7 受伤子流程图与死亡子流程图

（二） 死亡

死亡方法在受伤方法调用过程中符合条件情况下被执行，首先设置死亡判断变量表明状态，调用射击脚本的停止效果方法来停止特效，同时改变动画控制单

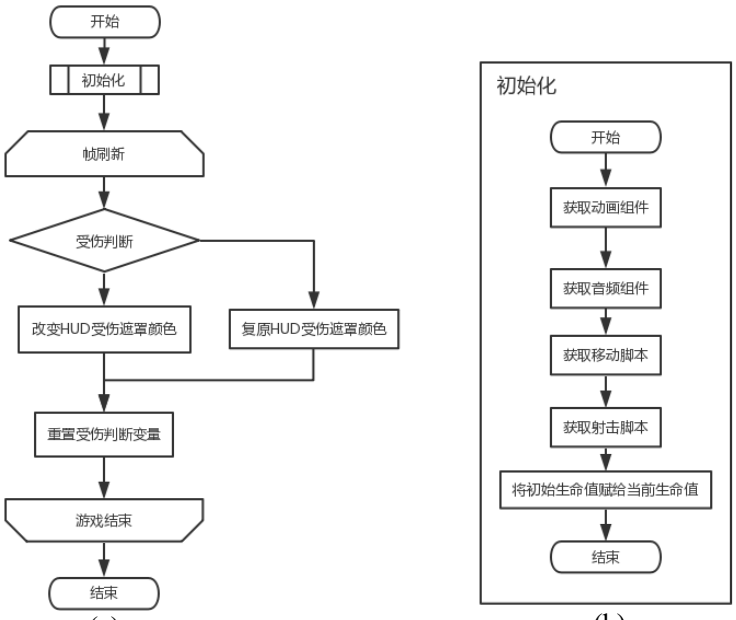
元的变量来触发死亡时人物的动画，禁用移动与射击脚本防止玩家继续操作，如图 3-7(b)所示。

第四节 敌人移动与敌人攻击

一、 敌人移动

敌人移动相对来说较为简单，由于依托的是 Nav Mesh 导航生成的计算路径，该物体需要挂载 Nav Mesh Agent 组件以调用生成好的路径，同时挂载人物生命值脚本与敌人生命值脚本以判断何时停止追踪，先通过人物独一无二的标签获取物体，再通过物体的挂载获得人物生命值脚本以保证对应。

在游戏帧刷新时，如果人物生命值与自身生命值都不为 0，便调用 agent 组件方法进行追踪，否则就关闭追踪，如图 3-8 所示。

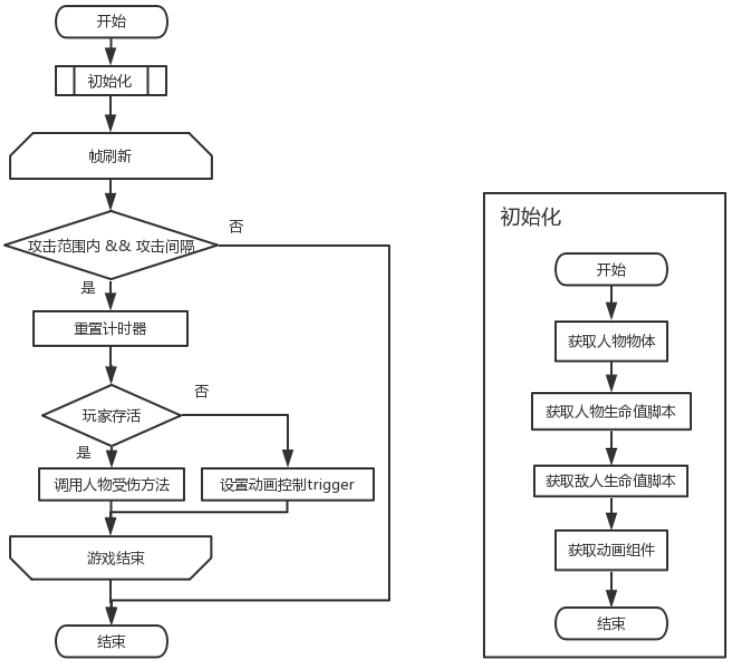


(a) (b)

图 3-8 敌人移动主流程图及初始化子流程图

二、 敌人攻击

敌人的攻击逻辑也较为简单，即足够靠近时便调用攻击方法进行攻击，当然也有一个变量用于控制时间间隔以免攻击频率过高，攻击时重置间隔并调用人物的受伤方法即可，如图 3-9 所示。



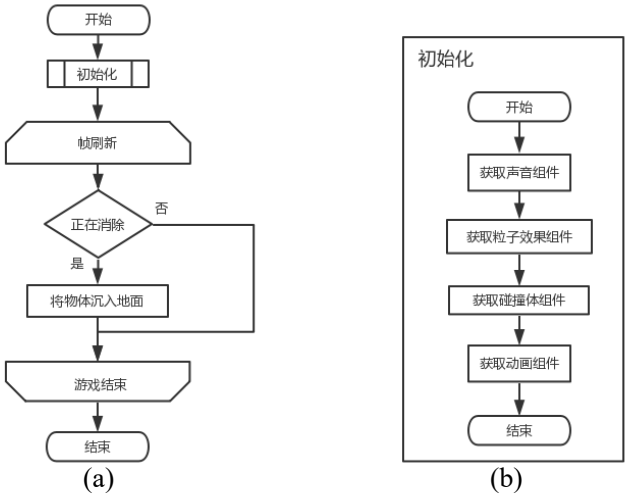
(a) (b)

图 3-9 敌人攻击主流程图及初始化子流程图

第五节 敌人生命值

一、 总体逻辑

敌人生命值脚本的逻辑与人物的有些相似但是作为被攻击方，需要进行射击反馈，即被射击方位有粒子效果产生，所以需要挂载 Particle System 组件。除此以外为了防止敌人尸体的过量堆积，需要敌人在死亡清算后消失，这里使用的是比较普遍的沉入地中再删除物体的方法，unity 会帮忙检测碰撞并自动执行方法，方法具体之后重写覆盖，如图 3-10 所示。

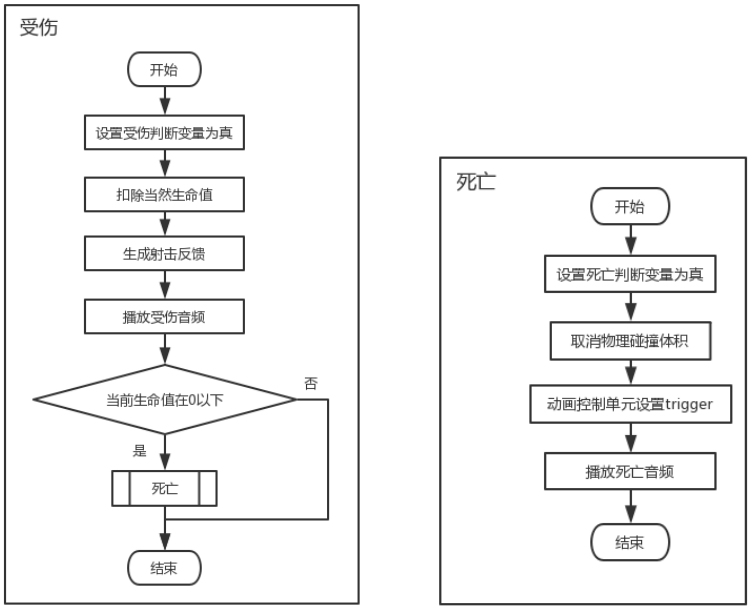


(a) (b)

图 3-10 敌人生命值主流程图及初始化子流程图

二、 子过程流程图及说明

受伤与死亡与人物生命值脚本的同名函数很相像，但是敌人不需要实时更新生命栏的信息，也不需要锁定人物的操控，不过对应的，会多了一条生成射击反馈的动作，如图 3-11 所示。



(a) (b)

图 3-11 受伤子流程图与死亡子流程图

第六节 菜单操作及管理相关脚本

一、 总体逻辑

菜单管理系统中存在着大量的方法，例如，暂停，音量调节，计分，排行榜，退出等，大多数方法并不会随着程序运行固定执行，而是随着玩家与游戏内设定好的 UI 固件互动时即时执行，执行时刻往往受到很大的不确定性限制，判断条件与执行子程序结果如图 3-12 所示。

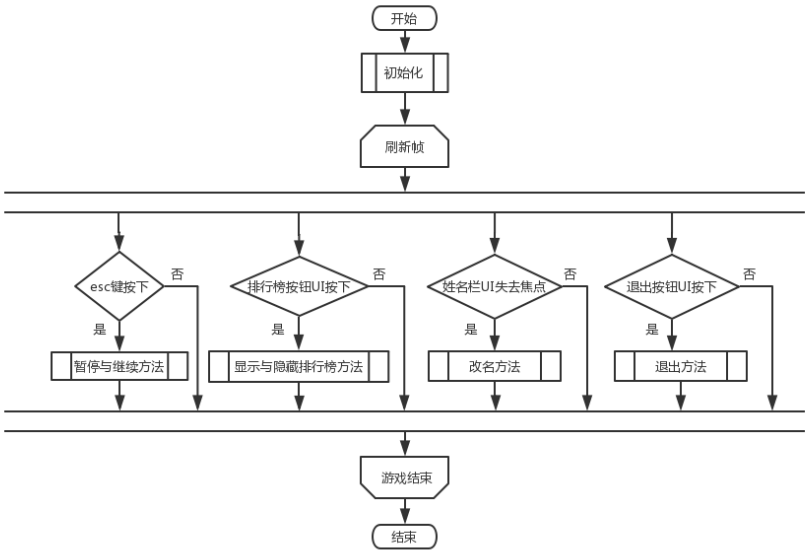


图 3-12 菜单脚本主流程图

二、 暂停与继续

虽然菜单可以调用的方法功能很多，但大多逻辑简单，只需要执行一句语言或修改一两个变量即可，故在此不展示流程图，在下一章实现时会做更详细的说明，对于实现较为复杂的暂停与继续功能在此进行说明。

暂停继续游戏是现代游戏的基本功能，需要在游戏帧刷新时检测是否有 esc 键按下，然后通过停止游戏时间来暂停游戏，并弹出菜单面板，隐藏状态栏，如图 3-13 所示。

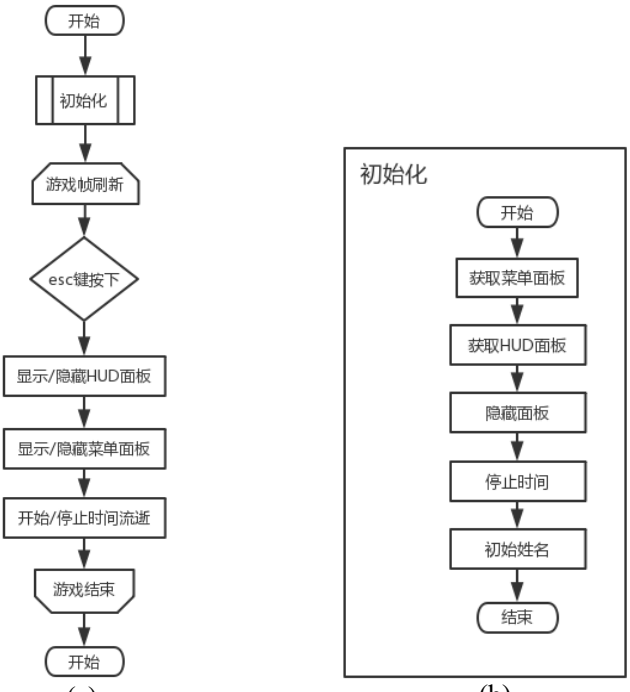


图 3-13 暂停与继续主流程图及初始化子流程图

第四章 游戏实现

第一节 人物移动转向与射击

一、 整体架构

在对人物制作完成后，部件架构如下图所示，由可拆分的三部分组成：Gun 枪的模型；GunBarrelEnd 枪口部位，用于绘制开枪时的灯光，粒子效果，以及开枪时的处理；Player 人物的模型，三者共同构成了人物模块，关系如图 4-1 及 4-2 所示。因为 Gun 与 Player 两部分只是绘制的模型，所以在此处并不详细说明。

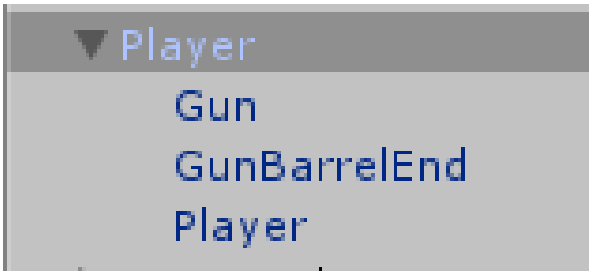


图 4-1 Player 结构

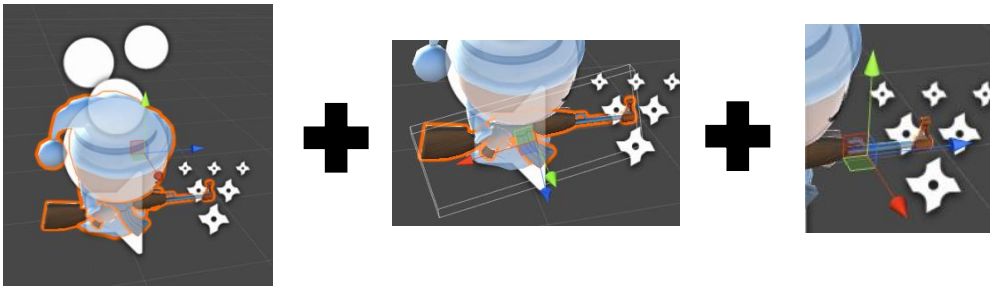


图 4-2 Player 的物体组合关系

二、 人物模块部分

（一） 挂载构造

整体模块挂载部件如图 4-3 所示，默认的 transform 部件表示其在游戏场景内的位置及旋转缩放等信息，由挂载的名为 PlayerAC 的动画控制单元负责调控动画变换，刚体与胶囊型碰撞块负责物理交互，Audio Source 中则存储了人物受伤时的音效，以便脚本调用。Player Movement 脚本负责移动相关的操作，Player Health 脚本负责与生命值相关的操作。

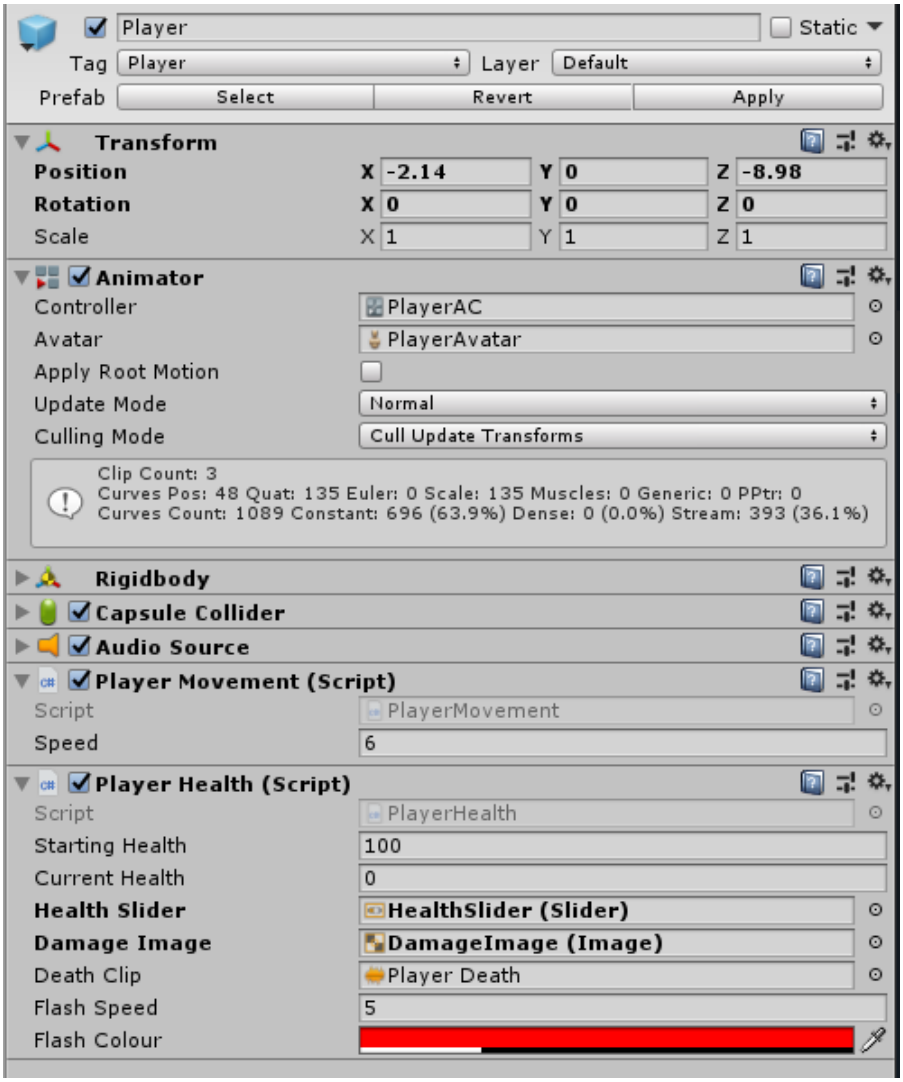


图 4-3 Player 部件

（二） 物理碰撞部分

根据之前设计好的移动与转向方式，游戏需要人物能做到即走即停，没有惯性，所以刚体部件的 Drag 阻力与 Angular Drag 角阻力应为无限大以确保输出停止时人物马上会停止行动。同时应该冻结 Y 轴位置变动，以确保人物始终在地面行走，同时保证射击的转向正常运行，基于 Y 轴 360 度旋转即可，所以同时冻结 X 轴与 Z 轴的转向。

设置刚体的同时，也要为其设置实际的碰撞体积以便后续的检测，本次添加的是胶囊形碰撞块，比较贴合人物的模型，在对位置、半径、高度进行微调后，人物的物理条件已经设置完毕，如图 4-4 所示。

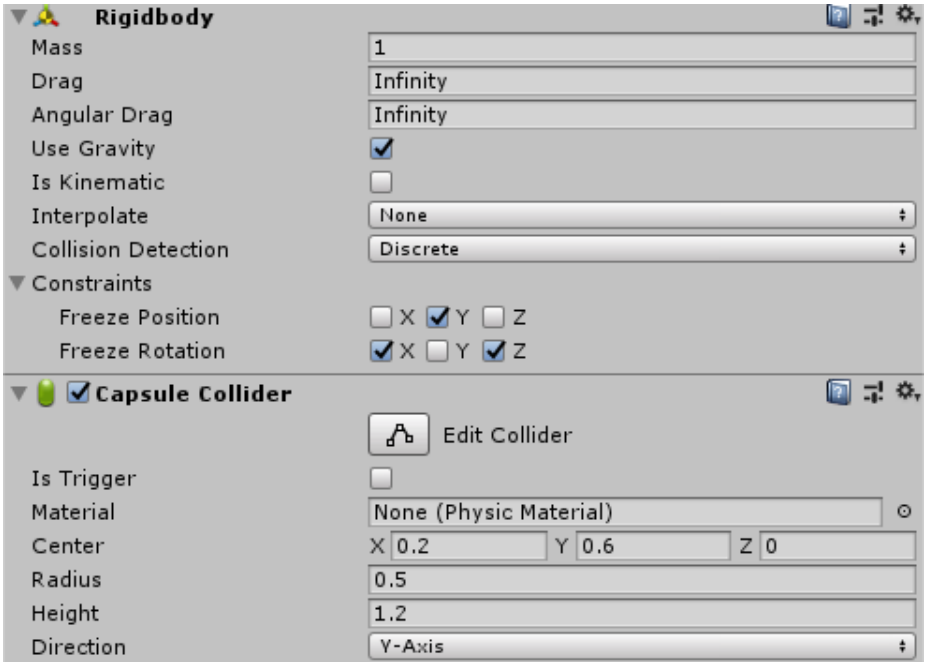


图 4-4 物理部件

（三） 移动脚本

人物移动的脚本代码的核心部分如下所示。

```
void Move(float h, float v)
{
    movement.Set(h, 0f, v);
    movement = movement.normalized * speed * Time.deltaTime;
    playerRigidbody.MovePosition(transform.position + movement);
}
void Turning()
{
    Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit floorHit;
    if(Physics.Raycast(camRay, out floorHit, camRayLength, floorMask))
    {
        Vector3 playerToMouse = floorHit.point - transform.position;
        playerToMouse.y = 0f;
        Quaternion newRotation =
        Quaternion.LookRotation(playerToMouse);
        playerRigidbody.MoveRotation(newRotation);
    }
}
void Animating(float h, float v)
{
    bool walking = h != 0f || v != 0f;
    anim.SetBool("IsWalking", walking);
}
```

（四） 生命值脚本

与生命值相关的脚本代码的核心部分如下所示。

```
public void TakeDamage (int amount)
{
    damaged = true;
    currentHealth -= amount;
    healthSlider.value = currentHealth;
    playerAudio.Play ();
    if(currentHealth <= 0 && !isDead)
    {
        Death ();
    }
}
void Death ()
{
    isDead = true;
    playerShooting.DisableEffects();
    anim.SetTrigger ("Die");
    playerAudio.clip = deathClip;
    playerAudio.Play ();
    playerMovement.enabled = false;
    playerShooting.enabled = false;
}
```

（五） 动画控制单元

人物的动画控制单元相对来说较为简单，设置了两个参数，分别为 bool 型的 IsWalking，用于检测人物是否在行走，另一个为 trigger 型的 Die，用于触发人物死亡动画，如图 4-5 所示。

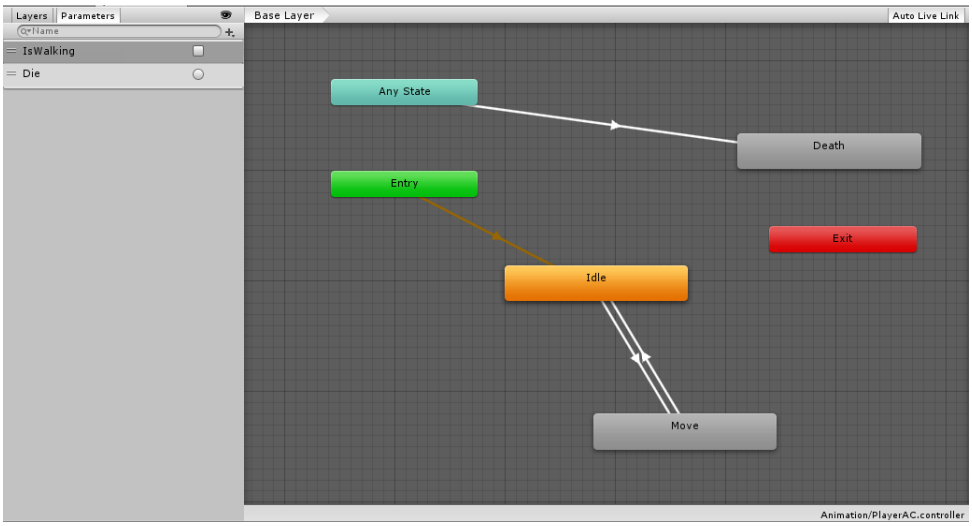


图 4-5 PlayerAC 动画控制单元

三、 枪口部分

（一） 挂载构造

枪口部分控制着与人物射击相关的一切动作，通常游玩射击游戏时，开枪都会伴随着枪声，枪口的闪光，以及子弹的出现。由于在第二章的设计中，本游戏采用了射击一体的方式，为了帮助判断射击到底打在了哪里，可以用 Line Renderer 组件画一条线来帮助判断，同时 Light 组件创造一个点状光来反馈开枪时子弹的轨迹，设置完后默认关闭这两个组件，在脚本中调用开枪方法时再打开以便模拟真实的开枪射击，枪口部分的整体挂载如图 4-6 所示。

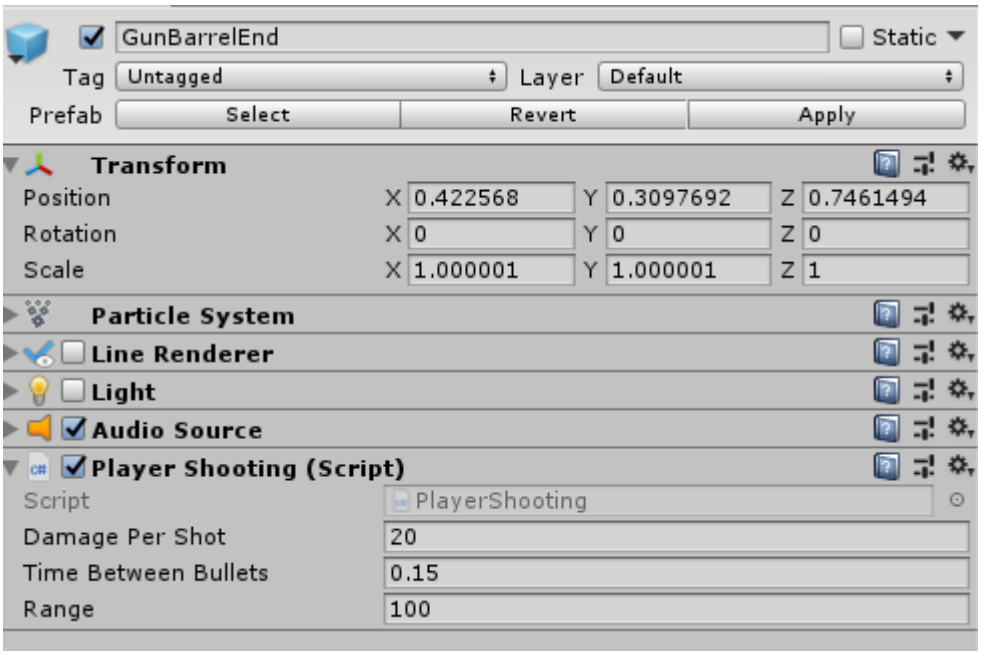


图 4-6 GunBarrelEnd 部件

（二） 射击脚本

有关射击脚本的代码核心部分如下所示。

```
        void Update()
        {
            timer += Time.deltaTime;
            if (Input.GetButton("Fire1") && timer >= timeBetweenBullets && Time.timeScale != 0)
            Shoot();
            if (timer >= timeBetweenBullets * effectsDisplayTime)
            DisableEffects();
        }
```

```
public void DisableEffects()
{
    gunLine.enabled = false;
    gunLight.enabled = false;
}
void Shoot()
{
    timer = 0f;
    gunAudio.Play();
    gunLight.enabled = true;
    gunParticles.Stop();
    gunParticles.Play();
    gunLine.enabled = true;
    gunLine.SetPosition(0, transform.position);
    shootRay.origin = transform.position;
    shootRay.direction = transform.forward;
    if (Physics.Raycast(shootRay, out shootHit, range, shootableMask))
    {
        EnemyHealth enemyHealth =
        shootHit.collider.GetComponent<EnemyHealth>();
        if (enemyHealth != null)
        enemyHealth.TakeDamage(damagePerShot, shootHit.point);
        gunLine.SetPosition(1, shootHit.point);
    }
    else
    {
        gunLine.SetPosition(1, shootRay.origin + shootRay.direction * range);
    }
}
```

第二节 敌人的追踪与生成

一、 整体架构

正如设计游戏时的初衷，本游戏的敌人将会不断生成，如潮水般涌来，而游戏的目的就是尽量存活足够长并消灭足够多的敌人以获取更高的分数，所以使用一个空物体 Enemy Manager 来挂载脚本控制刷怪，并在地图放置多个刷怪点以便生成的怪物获取初始位置，同时制作了怪物模块以便多次使用，如图 4-7 所示。

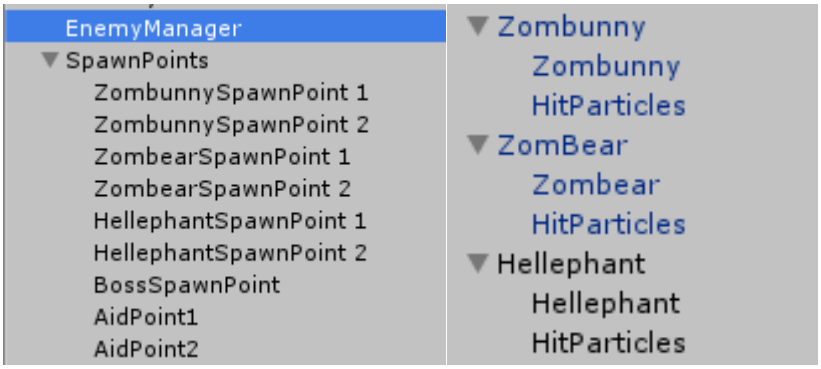


图 4-7 敌人与点的结构

正如上图看到，本游戏为每个怪物都设计了两个刷怪点，位于地图上不同的位置，每次脚本调用生成时选取的位置都是随机的，同时还有两个急救包生成点，为了节省资源考虑，在这里急救包的生成与初始化共用了 EnemyManager 脚本。

另一方面，也可以看到三种怪物实际上构成都非常相像，由一个本体模型与一个击中的粒子效果组成，由于模型并不是这次课题的讨论重点，所以本次关于模型部分也不予论述。

二、 怪物模块部分

由于三种怪物除了动画、模型、体积不同外，行动逻辑均为一致，所以此处以 ZomBunny 为例说明。

（一） 挂载构造

除了默认控制物体在游戏世界的位置的 Transform 部件外，由动画控制单元 EnemyAC 控制动画的变换，刚体、胶囊型碰撞体、球型碰撞体共同构成了物理交互，Nav Mesh 调控烘焙后地形上自动追踪的参数，挂载的 Enemy Movement 负责敌人的移动操作，Enemy Attack 脚本负责敌人的攻击操作，Enemy Health 脚本负责敌人生命值相关的操作，Audio Source 则储存了敌人受伤时的音效，如图 4-8 所示。

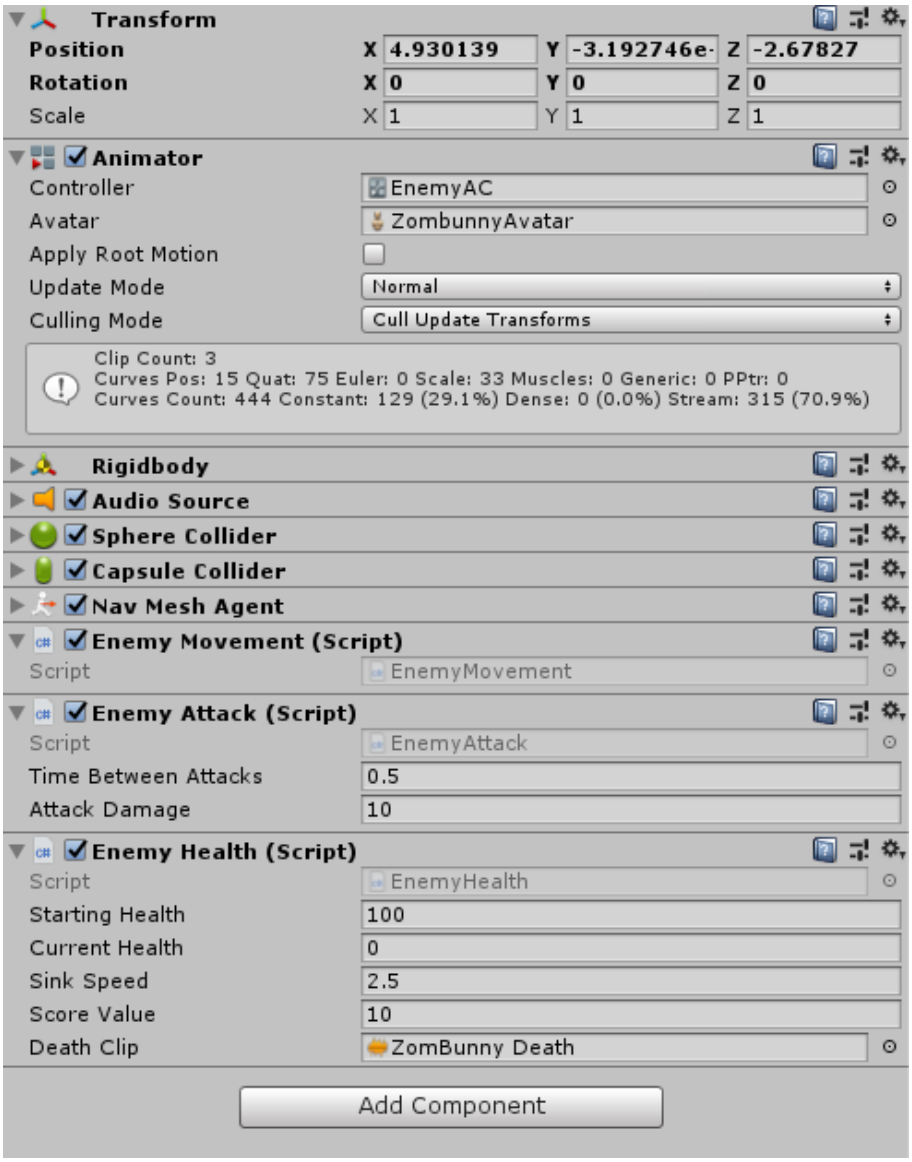


图 4-8 ZomBunny 的部件

（二） 物理碰撞部分

作为敌人的物理部分与人物很相像，本游戏同样需要他们即走即停，只在平面行走，Y 轴转向，所以刚体部分其实参数是一致的。

胶囊型碰撞体作为碰撞互动部分，对于不同的敌人设置的参数有所不同，比如 Hellephant 体积较大较胖，游戏处理时，半径就要相应的加大，以确保能够准确的完成物理反馈。

与人物不同，怪物模块还配有一个球型碰撞体，但这个碰撞体并不是用于模拟物体的物理互动，而是用于脚本检测，在 Unity 中，可以设置碰撞体为 Trigger，以便代码判定并调用，使用比敌人本身稍大一点的球型判定碰撞体检测是否与人物相碰，检测由代码实现，通过检测物体的 Tag 标签是否为 player 决定。相应的，作为判定器，球型碰撞体在与别的碰撞体交互时并没有实体，如图 4-9 及 4-10

所示。

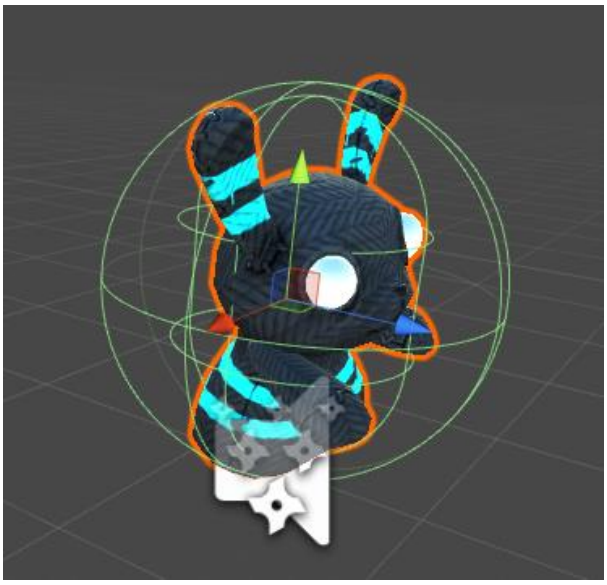


图 4-9 ZomBunny 的两个碰撞体

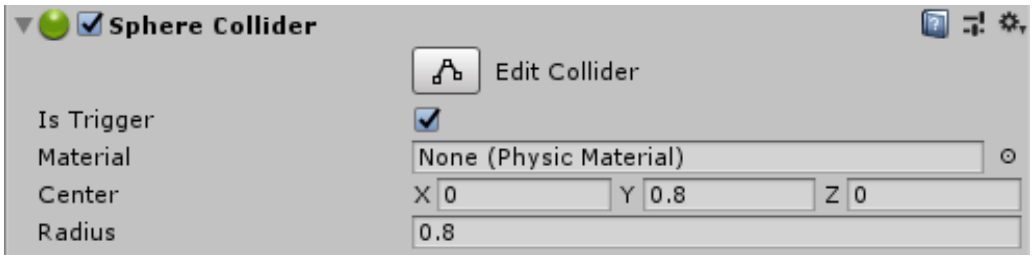


图 4-10 ZomBunny 的球型碰撞体

（三） 移动脚本

敌人的移动正如第三章设计的那样，采用 Nav Mesh 技术计算导航路径并追踪完成，所以可以看到敌人均挂载了 Nav Mesh Agent 部件，针对于每个敌人的具体大小定义安全距离后，在脚本的初始化方法中，获取同模块下的 Nav Mesh Agent 部件以调用数据，代码核心部分如下所示。

```
void Awake ()
{
    player = GameObject.FindGameObjectWithTag ("Player").transform;
    playerHealth = player.GetComponent <PlayerHealth> ();
    enemyHealth = GetComponent <EnemyHealth> ();
    nav = GetComponent <UnityEngine.AI.NavMeshAgent> ();
}
void Update ()
{
    if (enemyHealth.currentHealth > 0 && playerHealth.currentHealth > 0)
    {
        nav.SetDestination (player.position);
    }
    else
    {
        nav.enabled = false;
    }
}
```

（四） 攻击脚本

在本节第二小节的物理碰撞中知道，怪物有专门用于检测是否碰到人物的碰撞体，本游戏只需要在初始化时挂载球型判断碰撞体部件，并调用继承自父类的 OnTriggerEnter/Exit 方法即可，具体实现代码核心部分如下所示。

```
void Update ()
{
    timer += Time.deltaTime;
    if(timer >= timeBetweenAttacks && playerInRange &&
    enemyHealth.currentHealth > 0)
    {
        Attack ();
    }
    if(playerHealth.currentHealth <= 0)
    {
        anim.SetTrigger ("PlayerDead");
    }
}
void Attack ()
{
    timer = 0f;
    if(playerHealth.currentHealth > 0)
    {
        playerHealth.TakeDamage (attackDamage);
    }
}
```

（五） 生命值脚本

该脚本在游戏帧刷新时会不断检测物体的下沉判断变量，该变量在下沉方法被执行时设置为真，如果物体在下沉，那么就修改物体的位置使其移动至地面以下，下沉方法会在设置好的延时之后销毁物体。

下沉方法触发的条件便是死亡方法中修改碰撞体为检测类型，一旦碰撞体变为该类型，其便不再保有碰撞体积，与任何有碰撞体的接触都会导致下沉方法的自动执行。而死亡方法在受伤方法的最后被判断，即承受不住伤害而触发。

伤害方法会先判断人物是否死亡以免出现冲突，随后播放受伤声音，减去生命值，根据传入的射击点设定射击反馈生成位置并生成射击反馈，最后判断生命值是否能承受并据此判断死亡方法，代码核心部分如下所示。

```
void Update ()
{
    if(isSinking)
    transform.Translate (-Vector3.up * sinkSpeed * Time.deltaTime);
}
public void TakeDamage (int amount, Vector3 hitPoint)
{
    if(isDead)
    return;
    enemyAudio.Play ();
    currentHealth -= amount;
    hitParticles.transform.position = hitPoint;
    hitParticles.Play();
    if(currentHealth <= 0)
    Death ();
}
void Death ()
{
    isDead = true;
    capsuleCollider.isTrigger = true;
    anim.SetTrigger ("Dead");
    enemyAudio.clip = deathClip;
    enemyAudio.Play ();
}
```

（六） 动画控制单元

怪物的动画和人物几乎完全相同，他们都有移动，站立，死亡三种动画，但是执行逻辑与人物不同，人物默认动画是站立，当移动时才切换为移动动画，死亡时切换为死亡动画，而怪物默认则是移动，因为他们被生成后马上就会扑向人物，而只有当人物死亡时，他们才会由移动切换为站立，并且不会也没有理由再次切换为移动动画。除此以外，他们还会在他们死亡时切换为死亡动画，所以这

里使用两个 Trigger 参数，一个用于判断怪物自身是否死亡，一个用于判断人物是否死亡，如图 4-11 所示。

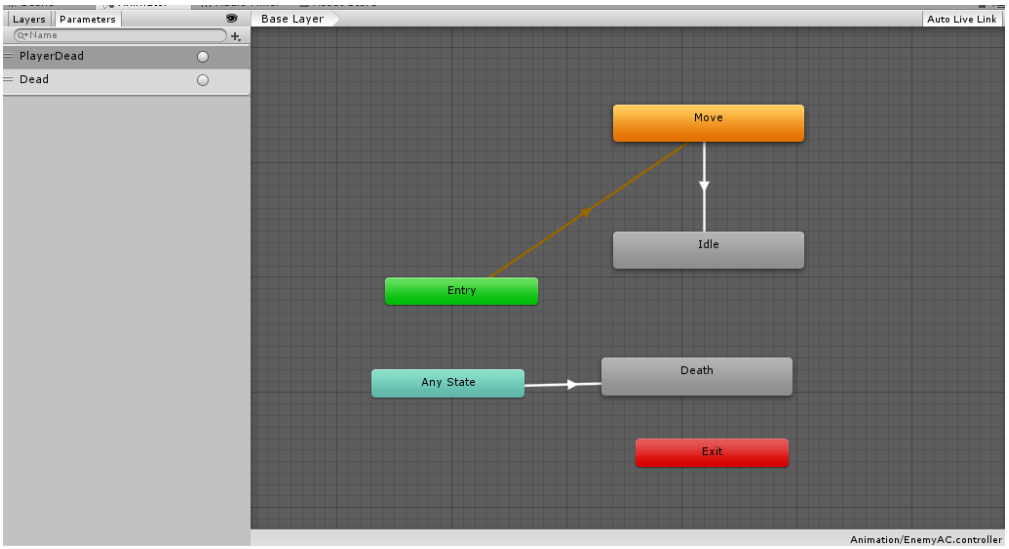


图 4-11 EnemyAC 动画控制单元

三、怪物的生成与管理

为了确保怪物的生成速度较为圆滑，为怪物的生成脚本 EnemyManager 规定了生成时间，可以针对不同的怪物设定不同的数值，比如熊和兔子因为较弱，所以生成时间较短，5 秒左右较为适宜，而大象因为较强，所以生成时间较长，15 秒左右较为稳妥，当然可以根据需求后续进行修改。

另一方面，为了游戏的可玩性与挑战性，怪物刷新速度应当越来越快，所以在脚本中设置了每 30 秒，所有怪物减少 1 秒刷新时间，直至 1 秒位置，以提高难度，同时，当有一种怪物为 1 秒刷新时，通过对场景灯光，天空盒，背景音乐的调整，营造另一种氛围的变换，以求增加可玩性。

同样，为了提高玩家的容错率，游戏内同时有可拾取道具医疗包，医疗包的生成与管理与怪物共用一个脚本。

生成怪物的代码的核心部分如下所示。

```
        void Start ()
        {
            InvokeRepeating ("Spawn", spawnTime, spawnTime);
        }
        void Spawn ()
        {
            if (playerHealth.currentHealth <= 0f)
            {
                return;
            }
            countTime += Time.deltaTime;
            int spawnPointIndex = Random.Range(0, spawnPoints.Length);
            Instantiate(enemy, spawnPoints[spawnPointIndex].position,
spawnPoints[spawnPointIndex].rotation);
            if ((countTime-0.1) > 0 && spawnTime != 1)
            {
                spawnTime = spawnTime - 1;
                countTime = 0;
            }
            if((spawnTime == 1) && bgchange)
            {
GameObject.Find("BackgroundMusic").GetComponent<MusicManager>().Change();
                bgchange = false;
            }
        }
```

该脚本与其他脚本略有不同，并没有通过使用刷新帧方法，而是在初始化方法中调用了一个重复方法，该方法会根据给定的时间参数与方法参数，间断性不断执行一个方法。

生成方法会先进行人物生命值判断，如果已经死亡便不再生成敌人以节省内存，并给计时器增加，随后从给定的刷新点数组中随机抽取一个点生成一个敌人，并做判断，如果满足一定时间且生成敌人的时间不是 1 秒，则调低刷新时间并清空计时器，至 1 为止。

同时如果任意敌人刷新时间到达底线，则通过查找背景音乐物体取得其下的音乐管理脚本来执行环境变换方法，该方法只需执行一次，所以用一个判断变量控制执行条件，执行后该变脸被设为假并不再使用。

该脚本在游戏内物体内被实例了多个对象，分别对应于不同的生成敌人或物品，对应的传入参数生成地点也不相同，对于游戏内脚本的配置如图 4-12 所示。

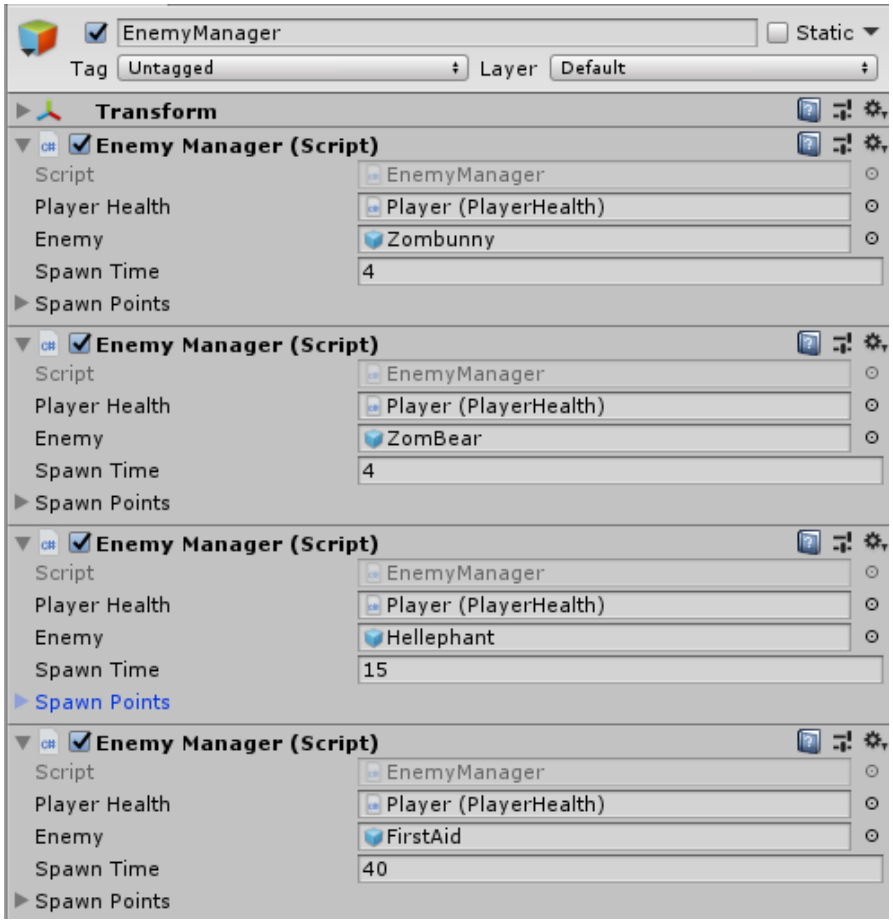


图 4-12 EnemyManager 部件

第三节 HUD 信息交互

一、 整体架构及挂载构造

HUD 信息面板的整体构造如图 4-13 所示，由生命栏，得分栏，受伤闪烁遮罩，游戏结束淡入遮罩，游戏结束的文字栏组成。由于 HUD 主要用于向用户传递信息，为了防止游戏过程中的意外，HUD 整体设置为了不可互动，即用户无法依靠输入设备改变或选择到 HUD 面板，挂载构造如图 4-14 所示。

图 4-13 HUD 面板结构

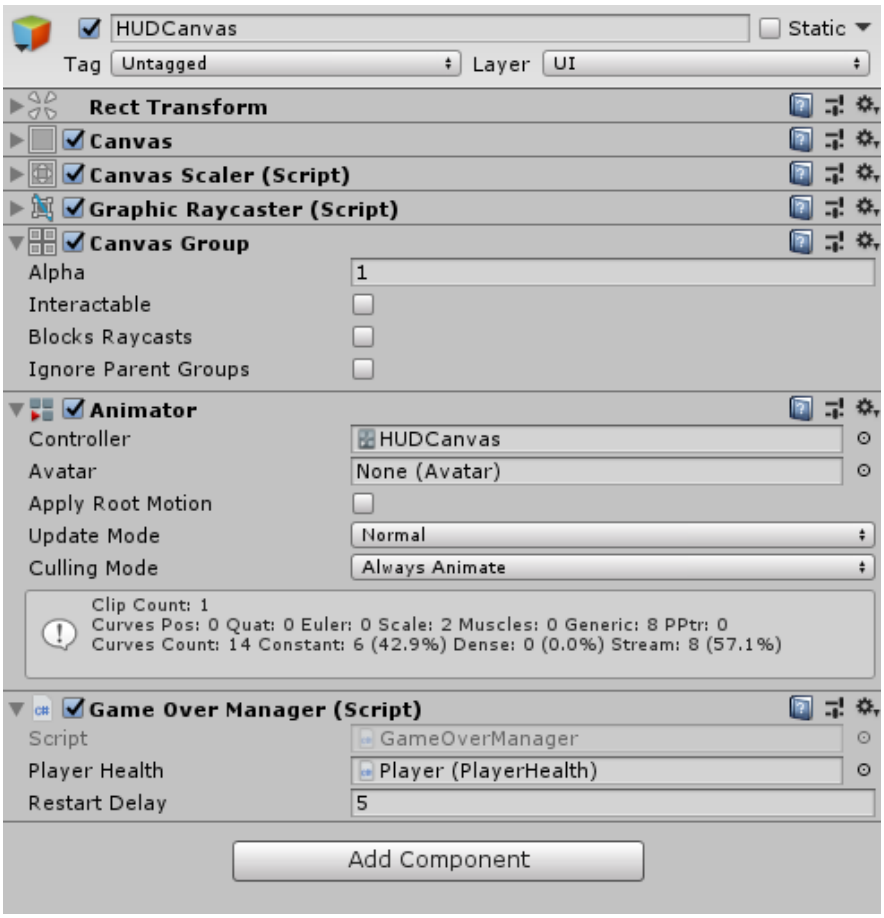


图 4-14 HUDCanvas 部件

因为本次游戏制作中不涉及画中画，多屏显示的等问题，Canvas，Canvas Scaler，Graphic Raycaster 等组件保持创建时自带的参数即可，本游戏要在该画布上创建多个 UI 控件，所以添加了一个 Canvas Group 用于管理，本游戏不希望 HUD 被用户操作到，所以 Interactable 被取消，名为 HUDCanvas 的动画控制单元负责 HUD 面板的动画变换（实际只操控游戏结束画面的切换），Game Over Manager 脚本用于检测人物是否死亡以便结束游戏。

二、 生命栏

生命栏由一个心形图标与一个去掉按钮的条形选择器组成，选择器的最高值为 100，即为生命值，在人物使用受到伤害方法时，条形选择器的数值会做相应减少，用户可以籍由背景与填充颜色背景的不同辨认当前的健康状况，生命栏如图 4-15 所示。

图 4-15 生命栏

生命栏修改数值的脚本主要为 Player Health 与 First Aid 控制，均为直接调用，生命栏 UI 本身并未挂载脚本，只负责显示。

三、 结束画面

游戏结束淡入遮罩 ScreenFader 和游戏结束文字栏 GameOverText 共同构成了游戏结束的画面，由于 unity 的渲染顺序是从上至下，所以当他们被显示时会遮挡除了得分栏以外所有的屏幕，以模拟结束场景，默认两者的 alpha 透明值均为 0，在动画效果中会在 1 秒内完全恢复，由 HUD 的动画控制单元操作。实际结束画面效果如图 4-16 所示。

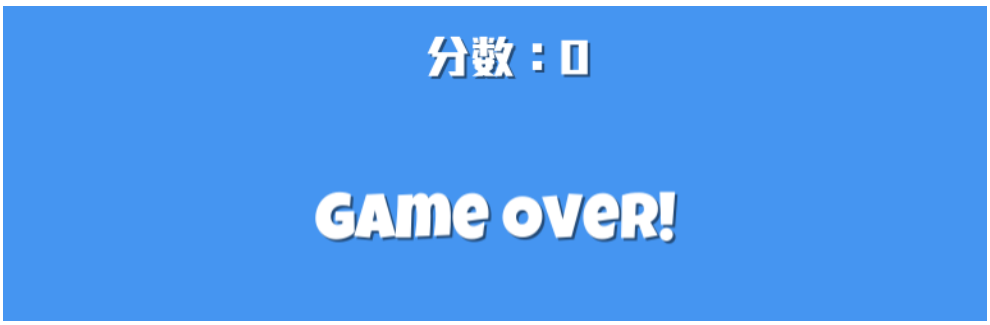


图 4-16 结束画面

四、 得分栏

得分栏会自始至终位于上方，并随着得分而增加，得分由其自身挂载的 Score Manager 脚本控制，挂载状况如图 4-17 所示。

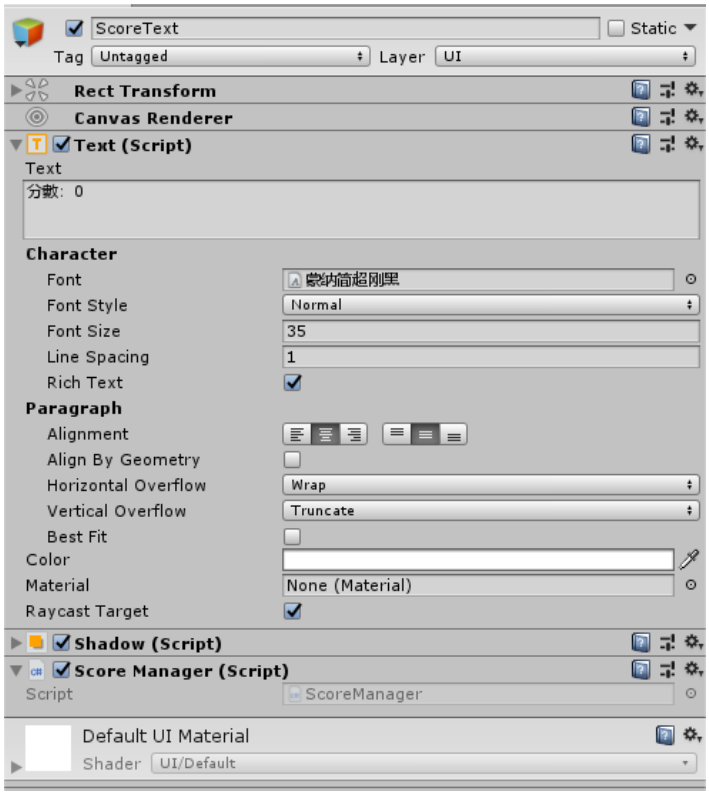


图 4-17 ScoreText 部件

Text 用于显示文本及其设置，游戏开始后挂载的脚本会实时修改 Text 部件中 Text 的值，Shadow 阴影部件用于增强显示效果，Score Manager 的代码核心部分如下所示。

```
void Awake ()
{
    text = GetComponent <Text> ();
    score = 0;
}
void Update ()
{
    text.text = "分数: " + score;
}
```

由于游戏较为简单，且分数在全部游戏中只有 1 个，所以分数变量设为了静态变量，方便其他脚本直接调用，初始化时获取 Text 部件并将得分清零，每帧实时将得分填入 Text 部件的文本中。

五、 动画控制部件

HUD 面板的动画操控非常的简单，因为只有游戏结束一个动画，本游戏只需要在游戏结束时切换至该动画即可，所以只需要一个 trigger 型的变量控制即可，如图 4-18 所示。

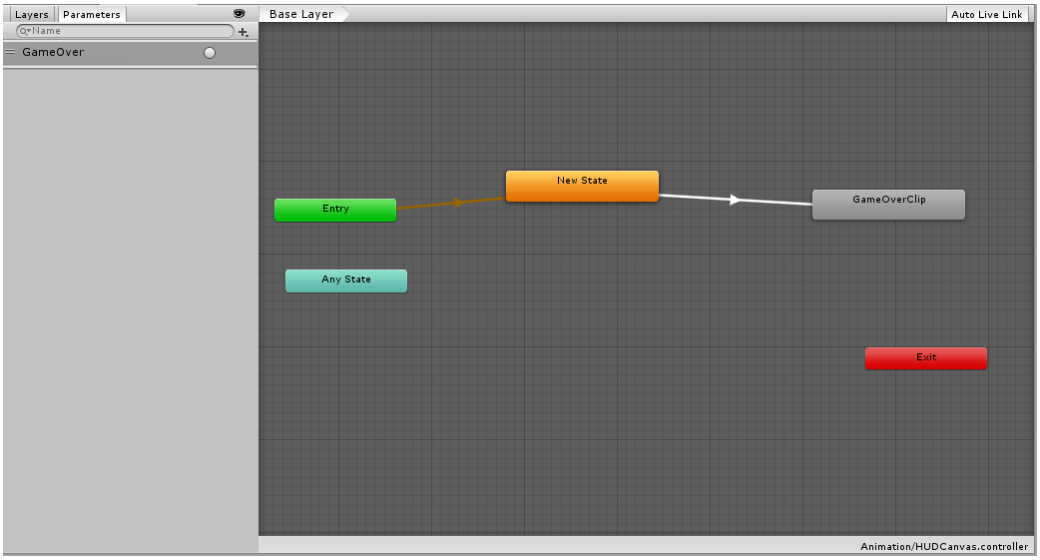


图 4-18 HUDCanvas 动画控制单元

六、 游戏结束检测脚本

游戏控制脚本的工作同样非常简单，每帧检测人物的血量是否在 0 以下，是的话便进行动画变换，并将当前的分数比对并添加进排行榜，同时在设定的事件后重新开始游戏，代码核心部分如下所示。

```
void Update()
{
    if (playerHealth.currentHealth <= 0)
        {
        anim.SetTrigger("GameOver");
        restartTimer += Time.deltaTime;
        if (!AddScoreTrigger)
        {
            playername =
            GameObject.Find("MenuCanvas").GetComponent<MenuManager>().pl
            ayername;
            GameObject.Find("MenuCanvas").GetComponent<MenuManager>().A
            ddScore(playername, ScoreManager.score);
            AddScoreTrigger = true;
        }
        if (restartTimer >= restartDelay)
        {
            AddScoreTrigger = false;
            SceneManager.LoadScene(0);
        }
    }
}
```

第四节 菜单与排行榜组合面板

一、 整体架构

菜单面板与排长榜分别绘制在两个不同的画布上，菜单面板包含标题栏，排行榜按钮，开始/继续按钮，音乐音量条形选择器，音效音量条形选择器。其中排行榜按钮同时包含一个输入框，可用于输入玩家自己的名字以便排行榜辨认。排行榜面板则包含了 7 条记录，每条记录都由一个名字与一个分数组成，共计 14 个 UI 组件，如图 4-19 所示。

在此处两个面板之所以分为两个不同的画布的原因有两点，第一，Unity 为了保证渲染顺序的正确，要求任何 UI 部件必须画在画布下，即 Canvas 中，所以将排行榜面板放置在菜单画布下逻辑上是说不通的，第二，菜单面板是可交互的，而排行榜面板只负责显示，无法交互，从这一点做区分的话，两者是不可以共用一个画布的。

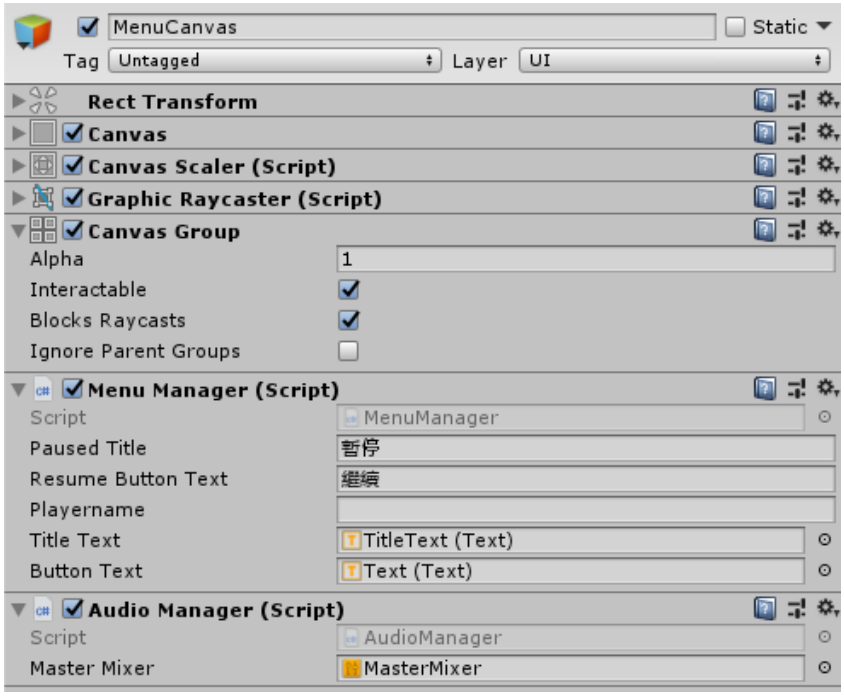
图 4-19 菜单栏与排行榜组合面板整体挂载

二、 菜单面板

（一） 挂载构造

与 HUD 面板类似的，本游戏同样不需要对 Canvas、Canvas Scaler、Graphic Raycaster 组件进行修改，不同的是，菜单面板需要与用户做交互，所以 Interactable 应该被勾选，如图 4-20 所示。

同时该面板也挂载了 Menu Manager 脚本用于操控与游戏进度相关的操作，Audio Manager 用于操纵用户修改音量时游戏内混音等音量方面的改变。



4-20 MenuCanvas 部件

（二） 排行榜按钮

排行榜按钮实际由一个按钮与一个文字输入框组成，如图 4-21 所示。

图 4-21 排行榜控件

按钮与文字分别挂载了方法，在按钮被点击时，会调用 Menu Manager 的 Rank 方法打开或关闭排行榜面板，文字输入栏中当输入完毕时会调用 Menu Manager 的 ChangeName 方法，并将其 Text 子部件作为参数传入以进行名称修改，如图 4-22 及图 4-23 所示。

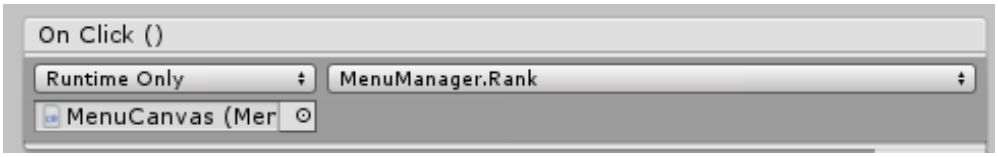


图 4-22 按钮执行方法

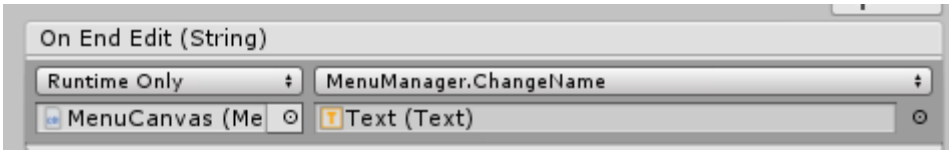


图 4-23 文本编辑执行方法

（三） 开始/暂停按钮

该按钮实现相对简单，在游戏开始时，按钮上的字为开始，游戏开始后再次呼出菜单面板时，按钮上为继续两字，由 Menu Manager 实时修改，该按钮在被点击时会调用 Menu Manager 脚本的 Pause 方法，如图 4-24 所示。

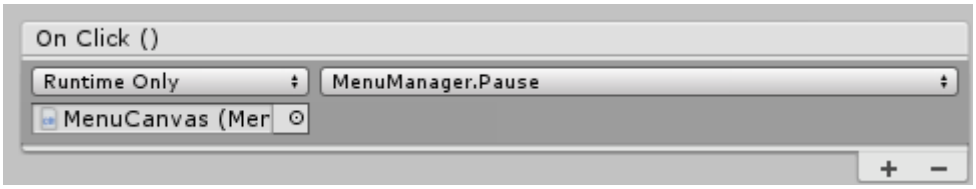


图 4-24 按钮执行方法

（四） 退出按钮

退出按钮位于菜单栏右上方，点按即可退出，实现原理是调用 Menu Manager 的 GameExit 方法，如图 4-25 所示。

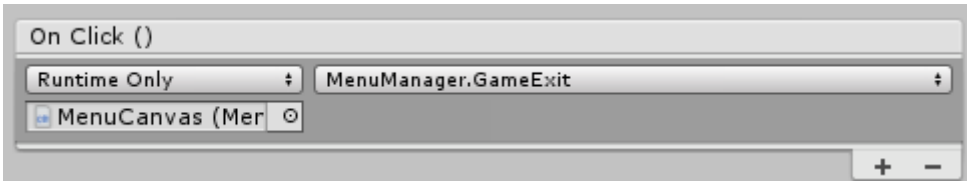


图 4-25 按钮执行方法

（五） 音量调节器

负责调节音量的两个控件均为条形选择器，玩家可以通过他们调节音量，在控件数值变化时，会实时调用 Audio Manager 中的 SetMusicLvl 方法或 SetSfxLvl 方法，如图 4-26 与 4-27 所示。

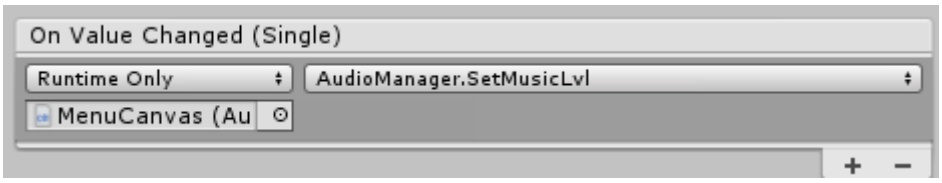


图 4-26 音乐调节器执行方法

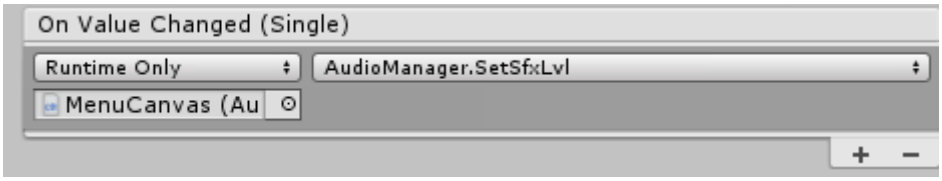


图 4-27 音效调节器执行方法

Audio Manager 的代码核心部分如下所示，在初始时将混音器的数值调制预设值，在方法被调用时，修改混音器中两个参数的值，两个参数均已分别绑定于两个播放器的音量上。

```
public void SetSfxLvl(float sfxLvl)
{
    masterMixer.SetFloat("sfxVol", sfxLvl);
}
public void SetMusicLvl(float musicLvl)
{
    masterMixer.SetFloat("musicVol", musicLvl);
}
```

（六）菜单管理脚本

菜单管理脚本实现的功能和可调用的方法众多，代码核心部分如下所示。

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        gameHasStarted = true;
        Pause();
    }
}
public void Rank()
{
    rankCanvas.enabled = !rankCanvas.enabled;
    GameObject.Find("RankCanvas").GetComponent<HighscoresManager>().Upd
ateHScore();
}
public void Pause()
{
    menuCanvas.enabled = !menuCanvas.enabled;
    hudCanvas.enabled = !hudCanvas.enabled;
    if (rankCanvas.enabled)
    {
        rankCanvas.enabled = !rankCanvas.enabled;
    }
    Time.timeScale = Time.timeScale == 0 ? 1 : 0;
    if (gameHasStarted)
    {
        titleText.text = pausedTitle;
        buttonText.text = resumeButtonText;
    }
}
```

管理脚本在初始化时会获取对象，并隐藏 HUD 信息面板与排行榜面板，并暂停时间，初始化排行榜分数，初始化玩家姓名。之后在帧更新时检测是否有键盘的 esc 键按下，有的话便会调用 Pause 方法，该方法会打开或关闭菜单面板，同时暂停或继续时间，顺便检测状态并修改按钮文字。

Rank 方法用于显示或隐藏折叠的排行榜面板，该方法首先会将排行榜的活跃值设为相反，即显示或关闭，随后通过寻找排行榜画布物体，取得其下的最高分脚本执行其中的刷新分数方法。

ChangeName 方法用于改变玩家的姓名。其参数传入 Text 类实体，并将实体内的字符串带入玩家姓名变量

AddScore 方法会先检测玩家配置文件中是否有记录，没有的话则先初始化，而后将分数比对添加并遍历替换排序排行榜。

GameExit 方法用于退出游戏，执行的是 Application 类下的关闭方法。

三、 排行榜面板

（一） 挂载构造

排行榜的挂载如图 4-28 所示，与 HUD 信息现实面板很像，因为都是显示为主，所以设置为无法操作，而面板的展开由菜单面板控制所以不需要多做什么额外的方法，挂载的 Highscores Manager 脚本用于将用户配置文件中的分数记录更新到 UI 控件中以显示给玩家。

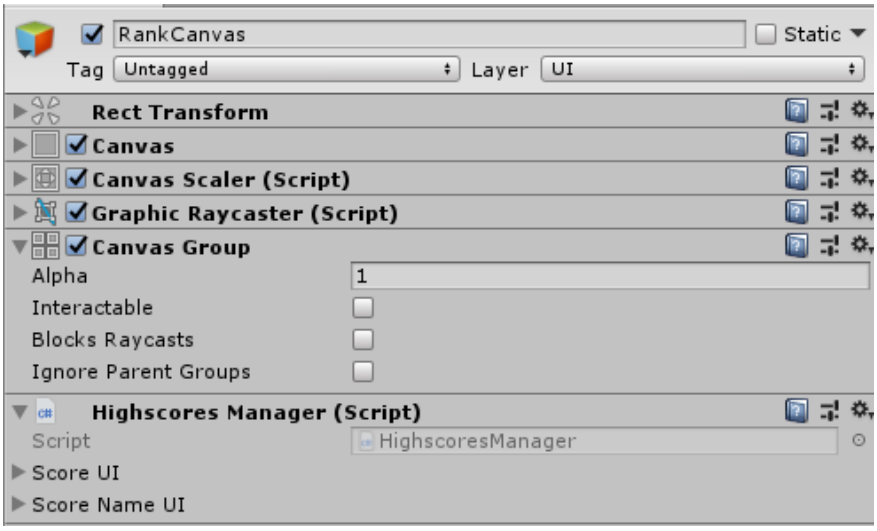


图 4-28 RankCanvas 部件

（二） 排行榜管理脚本

排行榜管理脚本如下所示，整型数组存储分数，字符型数组存储姓名，两个 Text 型数组即排行榜面板下的控件，通过循环遍历并填充数据，以在排行榜 UI 中进行显示，代码的核心部分如下。

```
public void UpdateHScore()
{
    for(int i = 0; i < 7; i++)
    {
        score[i] = PlayerPrefs.GetInt(i + "HScore");
        scoreName[i] = PlayerPrefs.GetString(i + "HScoreName");
        scoreUI[i].text = score[i].ToString();
        scoreNameUI[i].text = scoreName[i];
    }
}
```

第五节 游戏环境的设置

一、 地图环境

本游戏通过 Asset Store 中获得的模型与素材，为游戏搭建了简易的游戏场

景，如图所示，各个障碍物均有其自己的碰撞体积，挂载的碰撞体组件因为与前面相似且不是终点，在此就不一一列出了，游戏场景四周均围上了不可见的有碰撞体积的墙以防玩家掉出游戏世界，实际游戏环境还有天空盒作为背景，在该预览界面未显示，绿色线体描绘的即是碰撞体积，如图 4-29 所示。

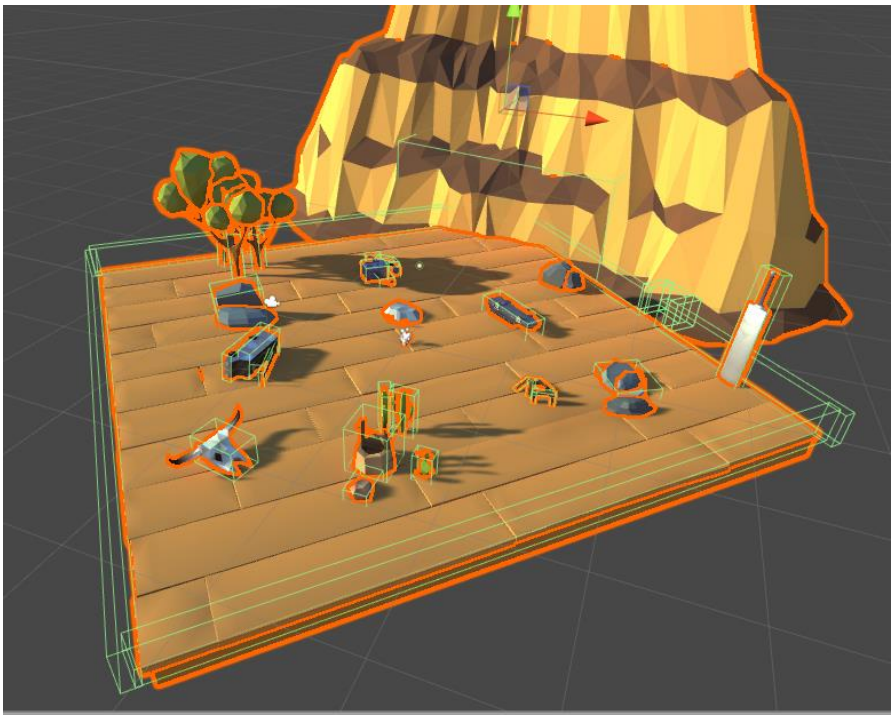


图 4-29 地图环境

二、 混音控制

（一） 环境切换

游戏环境内设有空物体用于处理音乐变换时的动作，如图所示。挂载 Music Manager 插件，作为参数提供了了两首不同的背景音乐与一个替换的黑夜天空盒，用于适当时机替换时做相关操作，如图 4-30 所示。

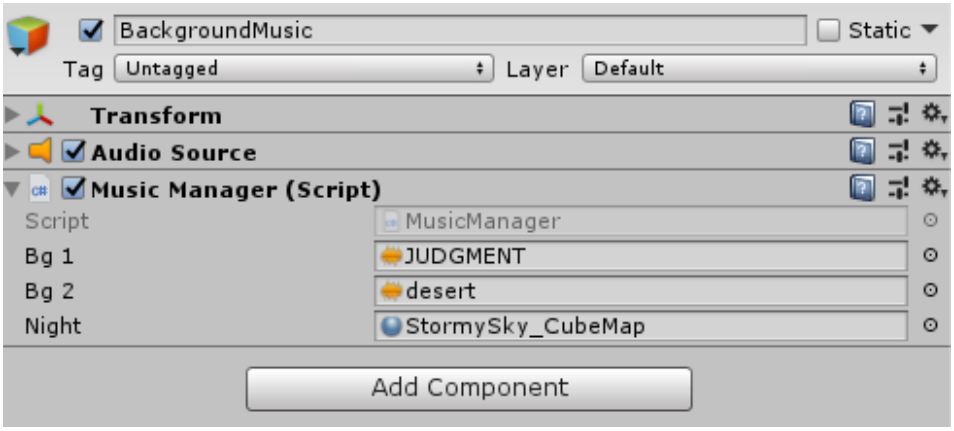


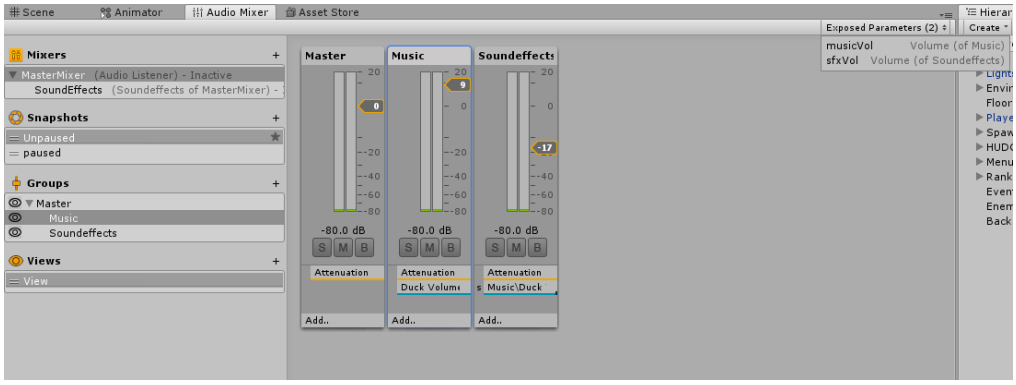
图 4-30 BackgroudMusic 部件

切换音乐控制的代码核心部分如下。

```
        public void Change()
        {
            GameObject.Find("Scene Lighting").GetComponent<Light>().enabled =
            false;
            RenderSettings.skybox = night;
            this.GetComponent<AudioSource>().clip = bg2;
            this.GetComponent<AudioSource>().Play();
        }
    }
```

（二） 音量调控

游戏的音乐与音量应当有一个相适宜的调节过程，在混音中本游戏将音乐与音效分开，均挂在主混音中，音效包括很多种，开枪声，受伤声，拾取声，死亡声均包括在其中，本游戏通过将两个参数绑定在音乐与音效上配合 Audio Manager 脚本控制音量，如图 4-31 所示，我们有两个混音器，其中生效混音是主混音的子集，而背景音乐与它同级，通过将音效与音量从主混音中分离开来，我们可以实现音量的分离调控，本章上一节第五小节中音量调节器代码就是通过调节我们设置好的参数来更改数值，而这两个参数分别与音乐音量与音效音量的强度相关联，除此以外，如果之后想要增加调节主音量的功能，只需新建一个关联主混音强度的变量并在音量调节器中扩充即可。



4-31 混音器

另外，通过使用回避音量功能，使用音量阈值的控制，我们可以实现游戏音量的动态控制。本游戏先检测音效的音量大小，当其超过一定值时程序会适当调小背景音乐以便使游戏反馈更加明了，而没有声效发出，或生效衰减低于阈值时，背景音乐会加大，如图 4-32 所示。

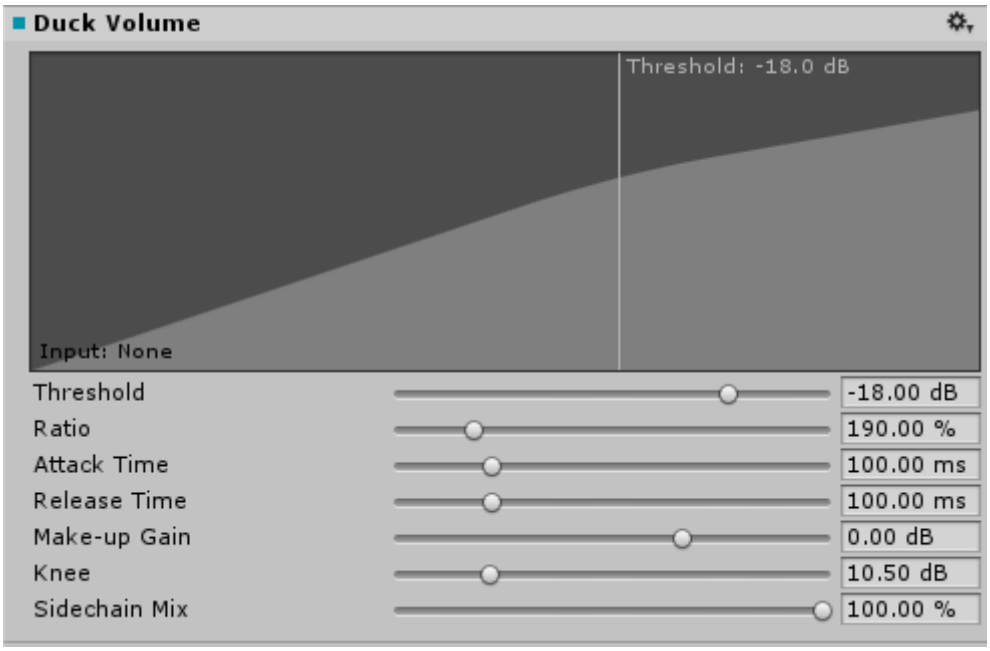
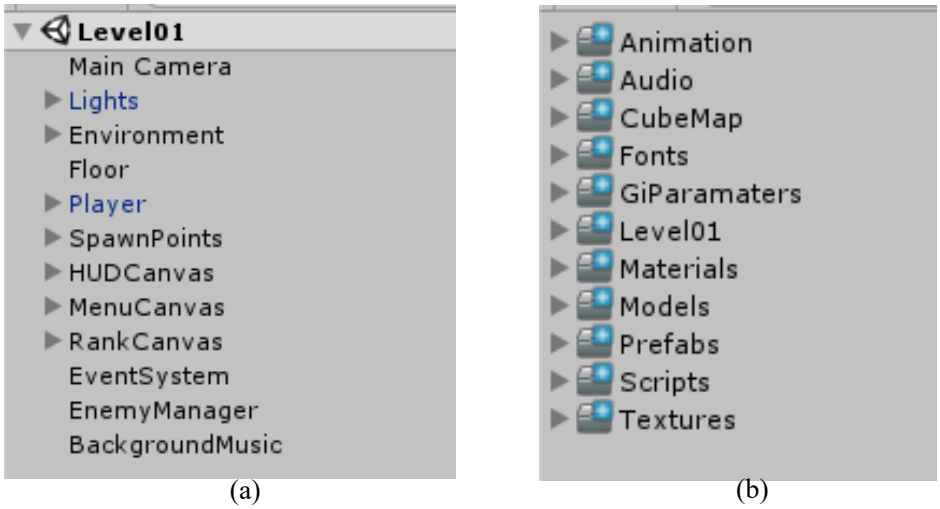


图 4-32 Duck 阈值控制

第六节 资源配置与组建

一、 资源配置

在完成了游戏的所有模块设计，对于游戏的资源配置，需要重新做一个规整，Unity 中非常重要的一个部分就是预制体，通过预制体，可以轻松保存一个配置好的物体，以便之后调用与再修改，所以在制作完怪物与人物后，可以将他们规整为预制体，并清整环境，文件结构与环境物体如图 4-33 所示。



(a) (b)

图 4-33 文件结构与环境结构

其中 Animation 文件夹中存储着所有动画控制单元，Audio 中存储音频文件及混音器，CubeMap 中为编辑好的天空盒，Fonts 中存储着字体，GiParamaters 为

Unity 生成的图片信息，Level01 中存储着场景 Level01 烘培好的导航窗格，Materials 中存储着材质，Models 中存储模型，Prefabs 中存储预制体，Scripts 中存储脚本，Textures 中存储材质图片。

二、 游戏组建

Unity 的游戏组件十分便捷，只需要设置目标平台的处理器架构，渲染的设置，以及质量，此处选择的目标平台为 64 位 x86windows 系统，导出后就可以获得经过压缩后的资源文件夹 Data，以及游戏可执行文件 exe 及配置文件等, 如图 4-34 所示。

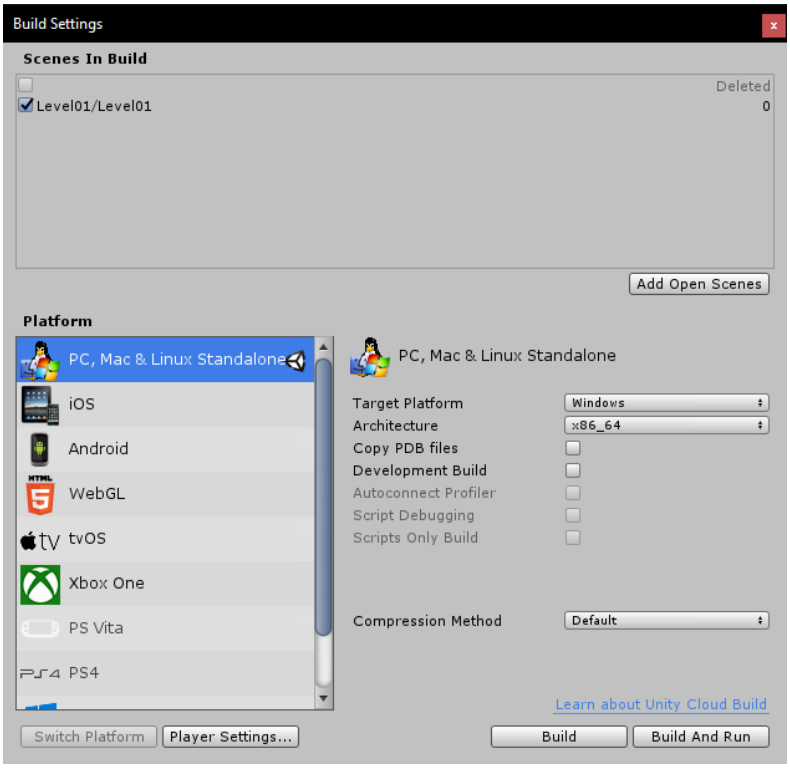


图 4-34 组建设置

结　论

基于 Unity 引擎和 C#语言的 3D 生存射击游戏的设计与实现制作了一个供玩家轻松畅快游玩并带有一定挑战性的单人游戏，本游戏针对玩家的兴趣点与游玩需求，做了多方面的考量与设计。按照需求分析，依次进行了玩法分析与系统设计，抽象了游戏的玩法与功能。经过了 Unity 引擎，C#语言的学习，各个功能的设计，游戏地图、界面的绘制，实际模块与功能的实现，玩法的多样性丰富，实际测试与完善等步骤。

依据对生存射击类游戏发展至今的玩法与系统模式的考察与分析，本生存射击类游戏分为关卡控制，综合控制与实时监测三个模块，其中游戏地形及障碍物的环境制作，敌人追击的导航烘培，敌人生成的节点设置，标准化“wsad”4 方向操控，单 Y 轴 360 度鼠标转向，脚本可控的动画切换系统，混音音量媒体控制，人物位置实时追踪，物理碰撞检测，游戏物体类型检测，状态检测等各个模块均在设计与实现中一一完成，通过精美的画面，将游戏最终呈现了出来。

此次基于 Unity 引擎与 C#语言制作的生存射击类游戏，用户可以在面对怪物的围追堵截中，通过障碍物进行紧张的周旋射击，也能在越来越多的敌人面前感到紧迫感，同时，位于游戏画面中的生命栏与得分栏可以让玩家随时确认自己的状况，游戏适时的环境切换也给游戏增添了不少切合的氛围与乐趣。另一方面。菜单栏的加入让玩家有了更大的自由性，玩家可以根据自己的需求随时暂停，或调节音乐音效的音量，排行榜的加入也让玩家可以不断挑战自我，玩家可以通过修改自己的名字并在结束时将分数计入排行榜，即便邀请朋友游玩也可以根据名字很方便的区分。

基于 U3D 的生存射击类游戏的设计与实现以玩家为中心，设计并实现了所有的核心玩法与功能，本游戏可以体现出很多生存设计类游戏的基本特性与方法，比如奖励性与挑战性，人物与敌人的行动逻辑等，通过对这些特性的研究与改进，本游戏不仅可以满足越来越挑剔的玩家需求，也能在同类游戏的制作中提供绝佳的模型参考。

参考文献

[1]张海藩，牟永敏. 软件工程导论[M]. 清华大学出版社，2013 年 8 月

[2]Unity Technologies. Unity 4.X 从入门到精通[M]. 中国铁道出版社，2013 年 11 月第一版

[3]Unity Technologies. Unity 5.X 从入门到精通[M]. 中国铁道出版社，2016 年 1 月 8 日

[4]Unity Technologies. Survival Shooter tutorial[Z]. YouTube，2014 年 10 月

[5]姚磊、陈帼鸾、陈洪. 游戏软件开发基础[M]. 清华大学出版社，2010 年 1 月第一版

[6]杨鑫. 基于 Unity3D 在 PC 端的 TPS 游戏的开发与设计[D]. 南京信息工程大学滨江学院，2017 年 10 月 25 日

[7]赵海峰. 基于 unity3d 的游戏开发及设计[D]. 山东科技大学，2014 年 6 月

[8]bylin. 取悦的工序：如何理解游戏[Z]. 简书，2014 年 7 月

[9]Unity Technologies. Roll-a-ball tutorial[Z]. YouTube，2015 年 4 月

[10]Ben Tristem，Mike Geig. Unity Game Development in 24 Hours, Sams Teach[M]. 株式会社ボーンデジタル，2016 年 9 月

[11]Jack Purdum. Beginning Object-Oriented Programming with C#[M]. wrox，2013 年 3 月 29 日

[12]Dawn Griffiths. Head First C#[M]. O'Reilly，2013 年 11 月 16 日

[13]Jane McGoniga. Reality is Broken:Why Games Make Us Better and How They Can Change the World[M].Penguin Books，2011 年 12 月 27 日

[14]Survival Shooter Extended[Z]. TwiiK.net，2017 年 1 月 9 日

[15]Chris Watters. The Gamer's Bucket List: The 50 Video Games to Play Before You Die[M].Key Lime Press，2015 年 11 月 4 日

[16]Tommaso Lintrami. Unity 2017 Game Development Essentials[M].Packt Publishing，2018 年 1 月 31 日

[17]Jesse Schell. The Art of Game Design: A Book of Lenses[M].CRC Press，2014 年 11 月 8 日

[18]Robert Nystrom. Game Programming Patterns[M].Genever Benning，2014 年 12 月 2 日

[19]Eric Lengyel. Foundations of Game Engine Development[M].Terathon Software LLC，2016 年 11 月 11 日

[20]Scott Rogers. Level Up! The Guide to Great Video Game Design[M].Wiley，2014 年 4 月 28 日

致　谢

此次毕业设计的选题、设计、实现等都是在赵奇老师的细心教导和真切关怀下完成的。从最初毕业设计的选题开始，本来非常害怕游戏这个题材作为计算机科学与技术这一专业的课题会不会太牵强，毕竟这是一门跨很多学科的方向，但是赵老师爽快，明朗的同意让我信心倍增，在老师的帮助下，我确定了选定的题目与大方向；在软件的分析和设计阶段，赵老师在百忙之中也不忘帮我对游戏的模块进行分析，对我出现纰漏与错误的地方及时指正，使我的游戏更加完善，而在实战阶段，赵奇导师更是给我极大的激励与肯定.每周的汇总，不管赵老师多忙，一定会到场帮我分章指导，纠正。在老师认真负责并的专业指导下我平稳顺利的完成了此次毕业设计，而老师严谨专业的教学风范和和蔼可亲的为人性格也深深地影响了我，让我在学业上、人格上都得到了提升。在此，我向赵奇老师致以由衷的感谢与敬佩之情。

另外，在此次完成毕业设计的过程中也离不开同学们的鼓励和帮助。他们在帮我测试游戏时，也提供了许多宝贵的改进意见，我很庆幸我身边有一群这样亲密的同学、朋友，在大学的最后阶段，大家还能够聚在一起学习技术、探讨问题、分享经验。自己独立完成一个项目绝不是一个简单的事情，我经常会在技术上遇到各种各样的问题，有时也会被一个问题卡住，花费掉整个下午，但是有同学有老师，他们的帮助着实让我的内心感到了温暖。在项目进行的过程中我也变得越发自信，更加珍视自己所学了四年的专业，更加珍惜友情和师生情。向我的朋友们、老师们、辅导员们送去祝福，谢谢大家让我在大学即将结束的时光里能有这样美好的经历和回忆。

在制作的过程中，我也收到了来自 stack overflow 上热心朋友 Oliver 的帮助，在他的帮助下我解决了游戏内排行榜的存储，在此对他的热心帮忙致以衷心的感谢。

最后，由衷地感谢大学期间的老师们给予我的知识与技术，希望老师们能为我的论文提出宝贵的意见和建议。谢谢！