摘要

近几年来，市场上的科普游戏越来越受人们重视，而在现有的市场上，关于消防员的游戏普遍都是以 2D 或 2.5D 画面为主，并且皆是玩家简单地操作游戏人物进行灭火，科普效果并不显著。因此，开发一款以第一人称为视角并且加入训练元素的消防员角色扮演类游戏十分有意义。

本次课题通过收集消防员的日常训练资料、真实的火灾危害数据以及救援方式数据，再结合影视作品、书籍等方向进行研究探讨，从而制作具有一定真实性的游戏关卡。

本次课题研究开发的游戏是一款基于 Unity3D 游戏开发引擎开发的第一人称视角的消防员角色扮演类游戏，游戏首先以地形编辑器 Terrain 制作游戏地形，以模型制作软件 3D Studio Max 制作以及修改模型加以辅助，结合 Unity 引擎自带的粒子效果控件，创造具有真实感的火灾现场场景。在游戏中的人物操作则是通过运用 Unity3D 游戏开发引擎中自带的角色控制器 CharacterController 实现人物移动控制以及视角控制。再利用射线检测碰撞、碰撞触发器算法以及 Enum 算法等方式实现了在游戏中不同的状态拥有着不同的效果发生，提高了游戏的可玩性与真实性。最后结合设定好的流程，使玩家更好地代入消防员角色之中。

综上所述，本次课题研究开发的游戏具有完整性以及创新性，具有较高的研讨价值及运用价值和学习价值。

关键词： Unity3D 消防员 角色扮演 真实 学习

Abstract

In recent years, popular science games in the market have attracted more and more attention from people. However, in the existing market, firefighter games are generally based on 2D or 2.5d pictures, and players simply operate the game characters to put out the fire. The popular science effect is not significant. Therefore, it makes sense to develop a firefighter role-playing game with the first person as the perspective and training elements.

This project collects the daily training data of firefighters, the real data of fire hazards and the data of rescue methods, and then conducts research and discussion in combination with film and television works, books and other directions, so as to make a certain authenticity of the game level.

This topic research and development of the game is a based on Unity3D game engine development firefighters role-playing games with a first-person perspective, game first with the Terrain editor Terrain landform, model making software 3 d Studio Max assist production and modify the model, combined with the Unity engine controls own particle effect, creating realistic fire scenarios. The character manipulation in the game is to achieve the character movement control and perspective control through the use of the CharacterController built into the Unity3D game development engine. By means of ray detection collision, collision trigger algorithm and Enum algorithm, different effects are achieved in different states of the game, which further improves the playability and authenticity of the game. Finally, combined with the set process, make the player better into the fireman role.

To sum up, the game researched and developed in this project has integrity and innovation, and has high research value, application value and learning value.

Key words: Unity Firemen Role play Authentic Study

目 录

第一章 绪论

1.1 本课题的研究背景与意义

本次课题研究是运用游戏编程、Unity3D 游戏开发引擎开发、3D Studio Max 模型制作软件制作和修改模型以及计算机图形学等相关技术实现的一个“寓教于乐”的第一人称消防员角色扮演单机游戏项目。其中带有边玩游戏以及边学习知识等特点：首先玩家扮演的是一名消防员，任务包括灭火任务以及救援任务，在这方面使用第一人称能使玩家更好的代入游戏当中；其次是玩家在游戏过程中，可以通过游戏中自带的训练以及关卡设置的机关中学习到一定的日常消防知识；再者，在现有的消防题材游戏当中，分为两种：一种是网页的儿童类小游戏，简单来说就是用鼠标点击火焰图标灭火，科普性极低；第二种则是以老牌 IP“消防第六分队”“急难先锋”等策略性游戏，科普性高，但是运用的皆是 2.5D 界面，职业代入感较差。因此，本次课题研究拥有着极高的研究价值以及市场价值。

1.2 消防员游戏在国内外的研究现况

1.2.1 消防员游戏在国内的研究现况

消防员游戏目前在国内并不得到重视，在网上搜索得到的中国消防员的游戏基本皆为网页小游戏。以百度搜索引擎为例，搜索消防员游戏，出现的基本都为 4399、7K7K 等小游戏网站提供的网页小游戏，其中的游戏基本为简单的点击火焰标志即可灭火，无论什么火情都是可以用水进行灭火。操作简单，但是科普性极低，而且无论什么火情都可以用水进行灭火是一个极其严重的错误，在一些特殊的火情当中，使用水去灭火很可能会造成更加大的危险。这很容易使得我国玩家单纯的以为水可以灭万火，导致在火灾当中造成二次伤害。

以下用国内小游戏举例：奇想咕噜团-勇敢消防员。玩家使用鼠标操作角色，鼠标点击建筑中的火焰标志即可喷出水柱进行灭火。可玩性不高，科普性极低。奇想咕噜团-勇敢消防员游戏截图如下图 1-1 所示：

图 1-1 国内消防小游戏《奇想咕噜团-勇敢消防员》游戏截图

1.2.2 消防员游戏在国外的研究现况

国外消防员题材游戏主要以老牌 IP“消防第六分队”“急难先锋”等策略性游戏为主，玩家主要是上帝视角 2.5D 控制角色进行救援任务，科普性极高，可玩性也高，但是玩家的代入性却不高。以“消防第六分队”游戏为例，玩家需要控制的是多名消防员进行救援任务，但是在游戏的过程中，很容易专注于一个任务点的救援任务而忽略某个任务点的消防队员，导致其死亡。当消防员出现危险时，游戏小地图会出现一个红色感叹号，但是在某处即将发生爆炸时，地图也是出现红色感叹号，玩家很容易将二者搞混，导致角色死亡。虽然此类游戏科普性极高，将各种火情的灭火条件都一一区分，例如水是无法熄灭油罐车的火情等等，但是如果是采用第一人称的话，玩家可以更好的代入这个职业当中，从而有更加深刻的印象。消防第六分队游戏截图如下图 1-2 所示：

图 1-2 《消防第六分队》游戏截图

1.3 本次课题的研究设计使用的开发软件

1.3.1 Unity3D

在当今电子游戏越来越得到人们重视的社会，Unity3D 也成为了绝大多数游戏开发者首选的开发 3D 引擎，虽然名字中带着 3D 二字，但是 Unity3D 也在 2D 游戏开发上有着极其优秀的表现。首先 Unity3D 采用的 ALL IN ONE 设计思路，使得开发者使用其开发游戏时极其方便，它将所有的编辑器集于一身，这可以让你对模型操作时只需要简单的选中模型、右键、弹出材质编辑器，然后对它的材质，碰撞体，结构等进行操作即可，让人觉得是在操作模型本身。其次 Unity3D 能够在编辑后立刻运行，查看效果，也可以在运行时修改游戏中的参数，可以实时看到你所作的调整，而当你停止运行时，你在游戏中所修改的参数也将失效，避免因为调整而忘记自己最开始所设置的参数，称之为“所见即所得”。再者 Unity3D 拥有独一无二的开发者商城 — Asset Stone，我们可以在这商城中购买到其他开发者开发的插件，可以用来借鉴使得我们得到更好的思路以摆脱当前所面临的困境，这是在其他游戏开发引擎中所不具备的特点之一。

1.3.2 3D Studio Max

3ds MAX 软件是由 Discreet 公司开发的用于打造三维动画渲染的软件，后来被广泛应用于游戏开发建模以及建筑设计和影视行业等领域。本次课题研究开发的游戏也将借助于 3ds MAX 强大且灵活的模型制作能力，对游戏中出现的楼房、场景中燃烧的物品以及相关的消防设备等模型进行搭建。

1.3.3 Microsoft Visual Studio

本次课题研究开发的游戏是基于 C#语言所开发的，因此利用的开发工具集是微软公司开发 Visual Studio。Visual Studio 配合 Unity 游戏引擎开发游戏的最佳搭档，能够利用其强大的调试功能来帮助使用者调试 Unity 游戏而快速发现问题。

1.4 本次课题的主要研究内容

综合前文论述，本次课题是以 Unity3D 游戏开发引擎作为开发工具，并以 3D Studio Max 模型制作软件为技术辅佐，研发的一款第一人称实现消防训练、现场灭火以及火场救出人质等工作从而完成救援使命的消防员角色扮演游戏。

第二章 游戏的需求分析与总体设计方案

2.1 消防员角色扮演游戏的需求分析

2.1.1 玩家需求分析

一个游戏的好坏，在于玩家觉得这款游戏是否好玩，而好玩二字无非体现在这款游戏能否让玩家在其中体验到：乐趣、吸引力、控制力以及挑战性，因此玩家最直观的需求也是以上几点。所以此游戏要求拥有 1.简单方便的操作模式；2.真实的火灾体验感；3.拥有挑战性的关卡。这样一来才能够让玩家有欲望玩下去，而不会半途而废。

2.1.2 功能需求分析

本游戏采用的视角为第一人称，结合消防员这一职业，需要使得玩家拥有面对灾难的真实感觉，必须紧紧围绕“真实”二字来实现游戏的整体。

首先，为了保证玩家在游戏初期训练关卡中能够快速学习游戏的控制方式，需要玩家与游戏的交互方式简单方便，并且有一定的交互反馈能让玩家体验到。

其次，由于本游戏采用的是第一人称视角，因此在游戏关卡中所设置的“火焰效果”以及“烟雾效果”必须要真实，风格不可以为卡通风或者太过于假，拥有贴近现实的“火焰效果”以及“烟雾效果”能够极大程度的提升玩家对游戏的体验感。但是粒子效果也需要“适当”，不能因为追求华丽而对游戏性能造成影响。

最后，还是围绕“真实”二字，制作氧气量的消耗。经过分析，在很多的救援案例中，百分之六十的消防员都是因为缺氧而导致伤亡，火焰燃耗会剧烈消耗氧气，再加上现场的燃烧产物有一氧化碳以及二氧化碳，这些都会造成消防员缺氧。因此打造一个氧气量消耗是十分必要的，能够极大提升游戏真实感。

综上所述，将本次课题开发的游戏的功能需求大致分为以下几个模块：

（1）人机交互：玩家通过使用键盘的 WSAD 四键来控制人物移动，空格键控制人物跳跃，再通过几个设置的按键进行触发设定好的效果。鼠标控制人物视角，左键控制激活水枪发射灭火剂、灭火器发射灭火剂等效果。

（2）粒子效果：火焰粒子效果与烟雾粒子效果必须做到真实。不同的火焰需要区分开来，只有使用与之相对应的灭火器才能完成灭火操作；而烟雾则需要做到当火焰被熄灭时从燃烧物体上出现，以此提升玩家的真实感。

（3）物理效果：人物在特定场景内会慢慢消耗事先设定好的氧气量，并且在氧气量消耗殆尽时会降低身体健康值，当身体健康值降为 0 时，会造成角色死亡，任务失败。

游戏大体功能需求分析如下图 2-1 所示：

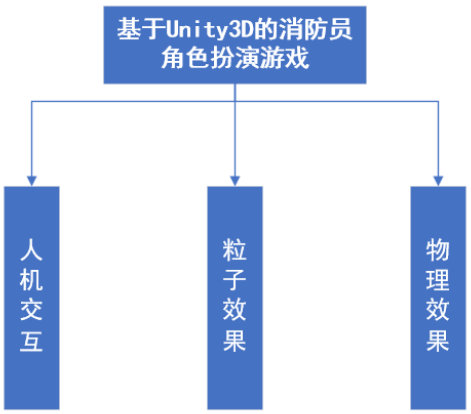

图 2-1 游戏大体功能需求分析图

2.1.2 性能需求分析

一款好的游戏，除了它的可玩性要高、内容要足够丰富以外，还需要控制它对机器性能的损耗，如果这款游戏不能够流畅的运行，那么将会极大程度的降低玩家的游戏体验。但是，在本次课题研究开发的游戏当中，制作真实的火灾现场必定会使用很多的粒子效果，而过多的粒子效果必然会损耗较多的机器性能，从而影响游戏性能。因此，在面对这些情况，必须在其它功能方面尽可能选择对机器性能损耗较低的方法实现。

2.2 游戏的总体设计方案

2.2.1 游戏结构设计方案

本次课题研究开发的游戏结构以简单方便为主，使得玩家玩家达到快速上手游戏的目的。本文游戏结构简单分为开始游戏→游戏主界面→进行训练/直接开始正式任务→任务成功/任务失败→重新开始任务/退出任务→主界面→选择关卡/退出游戏，游戏结构详情如下图 2-2 所示。

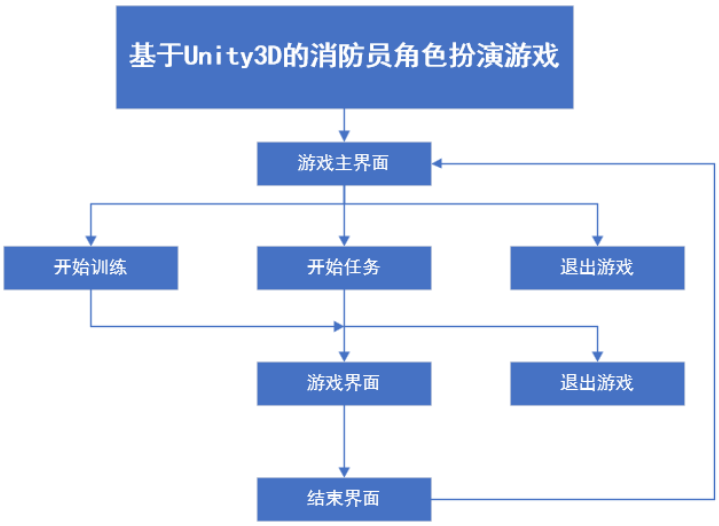

图 2-2 游戏结构图

2.2.2 总体结构设计方案

结合上文所述，可以得出本次课题研究开发的游戏总体模块的架构如下图所示：

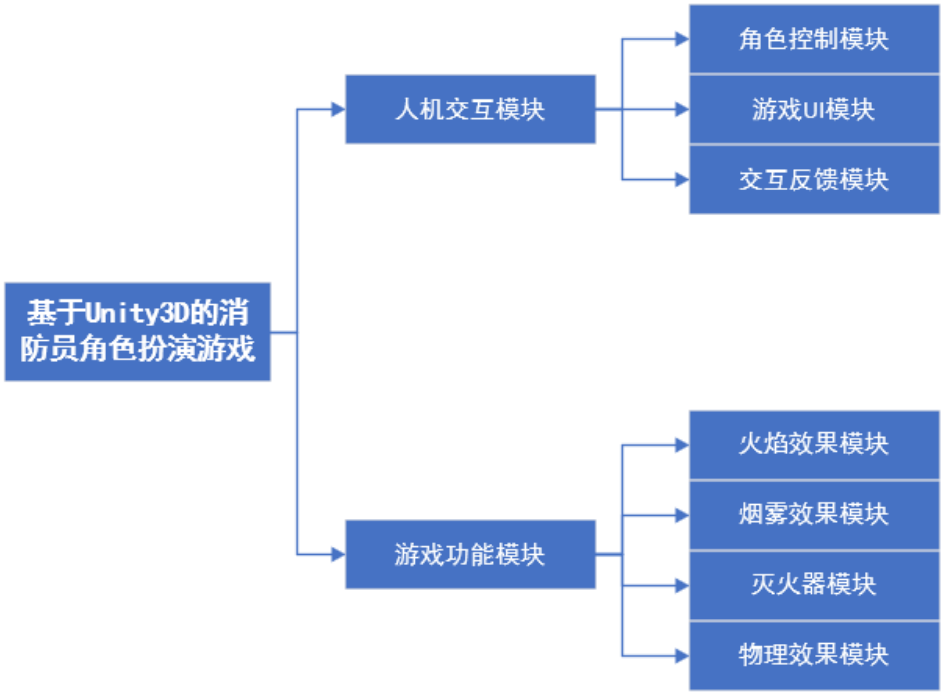

图 2-3 游戏总体模块架构图

对于本次课题研究开发的游戏中各个模块的具体分析如下：

（1）人机交互模块：游戏中的主界面、开始游戏界面以及游戏进行时的 UI 界面均由 Unity3D 游戏开发引擎中自带的 UGUI 模块来设计，使用 C#脚本来监听识别玩家的操作，再通过 UI 的变化告知玩家此操作对游戏带来的影响，以此完成一系列的人机交互。在交互反馈方面，使用 C#脚本编辑模块监听识别玩家的键盘输入或者鼠标点击事件，通过 UI 的颜色变化或者放大缩小又或者 UI 颜色递减等方式与玩家进行交互，例如当玩家点击开始任务按钮可以通过按钮的变化来告知玩家是否已经点击到按钮；而在游戏训练关卡中，会以一部分字牌方式来提醒玩家。

（2）游戏功能模块：游戏场景将使用由 3ds MAX 制作或修改的游戏建筑模型，配合 Unity 游戏开发引擎自带的 Terrain 地形编辑组件搭建完成。在救援现场建筑模块中，建筑模型的表面将配合火焰粒子效果以及烟雾粒子效果而制成真实的灾难现场效果。在建筑内部模块中，同样会添加火焰粒子效果以及烟雾粒子效果，同时再加入 Unity 游戏开发引擎中自带的 Fog Light 设定来增加烟雾降低玩家可见度的效果。当玩家进入特定区域之后，氧气剩余量或者身体健康值会发生递减变化来告知玩家在此区域呆久可能会有生命危险；当玩家使用消防器材进

行灭火时会以粒子效果变化来告知玩家此操作的是否有效，以达到提高玩家的游戏真实体验。

第三章 游戏详细设计与实现

3.1 角色控制模块

3.1.1 角色移动控制

在本次课题研究开发的游戏当中，角色操作模式与一般的第一人称射击类游戏操作方式大同小异，均是以键盘操作以及鼠标操作作为输入方式来对游戏事件进行触发以及操作。因此在本游戏中，人物移动是以 WSAD 控制前进、后退、向左以及向右，空格控制跳跃。

在基于 Unity3D 游戏开发引擎开发的游戏当中，常见的控制角色移动方法有几种。第一种方法是直接修改角色的 transform.position 属性，可以利用 if 语句检测玩家的键盘输入来修改相对应方向的 transform.position 属性，以达到角色移动的目的，但这种方法有个缺点，就是角色移动起来会让人有一种卡顿的感觉；第二种是使用刚体 Rigidbody 组件，通过在相对应的方向施加力的作用来带动角色的移动，然而这种方法是具备物理特性的，意味着当你松开按键的时候，角色不会立即停下，而是会继续滑动一部分距离；第三种是使用 Unity3D 游戏开发引擎自带的角色控制器 CharacterController，调用 Move 方法就可以实现角色的移动以及跳跃。在本次课题研究开发的游戏当中，为了减少代码开发量，我们选择使用第三种方法来实现角色移动。角色移动控制脚本核心代码如下图 3-1 所示：

```
public class Move : MonoBehaviour
{

    public float speed = 6.0F;
    public float jumpSpeed = 8.0F;
    public float gravity = 20.0F;
    private Vector3 moveDirection = Vector3.zero;

    0 个引用
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
    }

    0 个引用
    void Update()
    {

        CharacterController controller = GetComponent<CharacterController>();
        if (controller.isGrounded)
        {
            moveDirection = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
            moveDirection = transform.TransformDirection(moveDirection);
            moveDirection *= speed;
            if (Input.GetButton("Jump"))
                moveDirection.y = jumpSpeed;
        }
        moveDirection.y -= gravity * Time.deltaTime;
        controller.Move(moveDirection * Time.deltaTime);

    }
}
```

图 3-1 角色移动控制脚本

3.1.2 角色视角控制

在本游戏中，角色的的视觉控制完全由鼠标完成，使用鼠标控制玩家前进的方向以及瞄准火源等操作。同时，与上文 3.1.1 同理，本游戏开发使用的视角控制代码也是 Unity3D 游戏开发引擎自带的 MouseLook 脚本。角色视角控制脚本核心代码如下图 3-2 所示：

```
    public enum RotationAxes
    {
        MouseXandY = 0,
        MouseX = 1,
        MouseY = 2
    }

    public RotationAxes axes = RotationAxes.MouseXandY;
    public float sensitivityHor = 9.0f;
    public float sensivityVert = 9.0f;
    public float minimumVert = -90.0f;
    public float maximumVert = 90.0f;
    public bool mouselock = true;

    public float _rotationX = 0;

    0 个引用
    void Update()
    {
        if (axes == RotationAxes.MouseX)
        {
            transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityHor, 0);
        }
        else if (axes == RotationAxes.MouseY)
        {
            _rotationX -= Input.GetAxis("Mouse Y") * sensivityVert;
            _rotationX = Mathf.Clamp(_rotationX, minimumVert, maximumVert);

            float rotationY = transform.localEulerAngles.y;

            transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
        }
        else
        {
            _rotationX -= Input.GetAxis("Mouse Y") * sensivityVert;
            _rotationX = Mathf.Clamp(_rotationX, minimumVert, maximumVert);

            float delta = Input.GetAxis("Mouse X") * sensitivityHor;
            float rotationY = transform.localEulerAngles.y + delta;

            transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
        }
```

图 3-2 角色视角控制脚本

3.1.3 鼠标锁定与解锁

考虑到玩家有时候需要使用鼠标对游戏的设置进行操作，而鼠标又是角色视角的控制器，这时候就需要增加一个操作可以将鼠标解锁出来，使得玩家可以进行点击画面的设置 UI，进入设置界面进行操作。

首先定义一个 Bool 值为鼠标的默认状态，再使用 if 语句检测玩家的鼠标解锁操作，改变这个 Bool 值，使鼠标显示出来，当玩家操作完成之后，再将此 Bool 值改回默认值，隐藏鼠标。

3.2 火焰效果模块

3.2.1 火焰部署

在游戏中，我们需要布置大量的火焰粒子，用以呈现火灾的效果，但并不只是简单将火焰粒子拖放至游戏场景当中，这样子我们只能看到粒子效果而无法检测到粒子效果，因此我们需要使用一些方法使其能够被检测到。在本次课题研究开发的游戏当中，我们先创建一个 Cube，然后根据我们所需部署火焰的场景位置改变它的形状大小，然后将我们所需要使用的火焰粒子添加至 Cube 当中作为其子类，再将 Cube 的 Layer 层修改为我们相对应灭火器检测的层，例如是用水灭的火焰层就选择成“WaterColl”，为下文 3.3.1 中灭火功能实现检测火焰作为铺垫，然后取消勾选 Cube 的 Mesh Renderer 用以实现隐藏，最后在父类上添加下文 3.3.1 中灭火功能用以实现“灭火”效果的代码即可。这样一来，我们能够被检测到的火焰就完成了，再拖放至场景中部署即可。部署效果如下图 3-3 所示

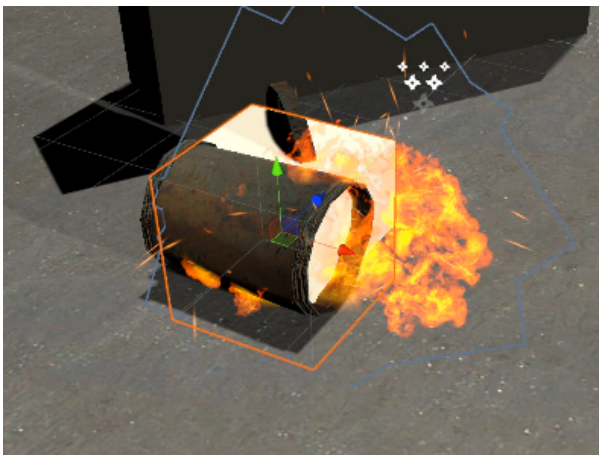

图 3-3 将 Cube 部署在场景中

3.2.2 灭火功能实现

在现实生活中，消防员实现熄灭火焰，无非就是使用能够灭火的物质，将其浇到火焰之上使之熄灭。同理，在游戏当中亦是如此，我们使用水粒子效果“浇”到火焰粒子效果上使火焰粒子效果慢慢消失从而达到灭火效果的实现。在这里实现灭火效果功能的方法有两种，第一种是粒子碰撞器，使用水粒子去触碰火粒子，再通过 C#脚本编辑模块调用 C#脚本去控制火焰粒子的生命周期或其他参数慢慢减小至 0 而实现灭火效果。第二种是通过射线碰撞检测玩家点击的物体是否为火焰粒子，如果是则通过 C#脚本编辑模块调用 C#脚本去控制火焰粒子的其中一个属性参数慢慢减小至 0 而实现灭火效果，如果玩家点击的不是火焰粒子，则不做任何操作。在第一种方法当中，使用粒子碰撞器会造成机器性能的大大损耗，而第二种方法则可以避免这种情况的发生。在本次课题研究开发的游戏当中，从游戏项目性能优化方面考虑，我们将选择第二种方法，以此来减小游戏对机器性能的损耗。

在游戏中玩家实现灭火效果，需要将鼠标准心指向火焰，其次是按下鼠标左键发射灭火剂击中火焰，最后灭火剂“浇灭”火焰。因此在本次课题研究开发的游戏中实现灭火的功能设计如下：

（1）粒子效果的控制：在游戏中实现灭火，其实就相当于让这个火焰粒子效果“消失”，而做到让这个粒子效果消失，我们可以修改粒子效果中的一些属性参数，例如“Start Lifetime”、“Simulation Speed”以及“Max Particles”，而在本游戏中，我们选择修改的粒子效果属性是“Max Particles”。首先创建一个 C#脚本命名为 FirePoint，用以控制游戏中部署的火焰粒子，使用 ParticleSystem[]方法定义一个数组 ParticalArr 用以管理粒子数组，其次再定义一个 int 值 ReduceRate 用以控制粒子减小的速率，最后使用结合时间差值与粒子减小速率的乘积——Time.time * ReduceRate 来控制粒子效果中“Max Particles”的变化。具体效果实现代码 FirePoint 如下：

```
//获取粒子数组
public ParticleSystem[] ParticalArr;
//火焰粒子减小速率
public int ReduceRate = 3;

private void Start(){}
private void Update(){}
```

```
//当准心指向火焰时，火焰开始慢慢减小
public void ReduceScale(){
ParticalArr[0].maxParticles -= (int)(Time.time * ReduceRate);
}
```

（2）使用射线碰撞检测+LayerMask 调用 FirePoint 脚本控制火焰粒子：我们发射一条射线，当射线碰撞到的物体的 Layer 层是我们设定好的目标 Layer 层时，调用一系列 C#脚本进行操作。在本游戏当中，我们是这样实现的：首先我们已经对部署的火焰设定好了相对应的 Layer 层，我们使用 Camera.main.ScreenPointToRay 方法从屏幕中心往鼠标指向的方向发射一条射线，用来进行碰撞检测；再定义一个 int 变量 layerMask，用来过滤我们不需要的 Layer 层；当以上准备完成后，我们就可以使用 Physics.Raycast 方法去获取碰撞物体，返回值设定为 bool 类型，当射线投射与任何碰撞器交叉时为真，否则为假。具体效果实现代码如下：

```
//过滤不必要的物体，只获取需要灭火的物体发生碰撞
int layerMask = 1<< LayerMask.NameToLayer("WaterColl");
//从屏幕中心发射射线
 Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
//获取从屏幕中心射出的射线是否碰到东西
if (Physics.Raycast(ray,out hit,30f,layerMask)){
    //如果点中了火焰，就获取 FirePoint 组件
    if (!hit.collider.gameObject.GetComponent<FirePoint>()){
        //如果找不到就去他的父类找
        CurFirePoint =
hit.collider.transform.parent.gameObject.GetComponent<FirePoint>();
        }
    //如果有就直接获取 FirePoint
    else{
        CurFirePoint =
hit.collider.gameObject.GetComponent<FirePoint>();
         }
        //找到组件之后开始调用 FirePoint 的 ReduceScale 函数进行灭火
         CurFirePoint.ReduceScale();
         }
      //查看射线方向
      Debug.DrawRay(ray.origin, ray.direction);
```

3.3 烟雾效果模块

在本次课题研究开发的游戏当中，烟雾效果主要有两种，第一种烟雾效果是着火的物体熄灭之后会冒出一定的烟雾；第二种烟雾效果是模拟在空气流通性差的空间内充斥大量因燃烧而产生的烟雾。前者我们可以使用粒子效果实现，而后者我们则可以使用 Unity3D 游戏开发引擎中自带的 FogLight 来实现。

第一种烟雾实现比较简单，使用 if 语句判断当前火焰粒子的 Max Particles 是否为 0 即可，假定检测到 Max Particles 值已经减小至 0，那么就开启烟雾效果。当 Max Particles 值不为 0 时，则烟雾粒子效果默认关闭。实现代码如下：

```
foreach (var item in ParticalArr){
          item.maxParticles -= (int)(Time.time * ReduceRate);
          //当 maxParticles 等于 0 的时候开启烟雾粒子
           if (item.maxParticles == 0){
               f (Fog != null) ;
               Fog.SetActive(true);
             }
}
```

第二种烟雾效果实现过程则比较繁琐，首先我们实现这个效果需要一个碰撞检测，检测当玩家身处于一个特定的区域时，应该开启烟雾效果，用以降低玩家可见度，提高游戏的真实性。

碰撞检测我们可以使用 OnTriggerEnter（）方法与 OnTriggerExit（）方法实现，实现该方法有三个前提条件，第一个条件是两个物体都必须有碰撞器 Collider 组件；第二个条件是其中一个物体的碰撞器 Collider 组件的 isTrigger 属性必须勾上；第三个条件也是最重要的一个条件，其中一个物体必须带有刚体 Rigidbody 组件，而且这个物体还是发生碰撞的主体。不满足这三个条件，就无法触发这两个方法。得出具体实现方法如下：

（1）首先创建一个 Cube，改变其大小使之与我们要进入的建筑差不多大小，其次关闭 Cube 的 Mesh Renderer，勾选其 isTrigger 属性。因为玩家身上已经拥有了刚体 Rigidbody 组件，且玩家是发生碰撞的主体，因此该 Cube 不需要添加刚体 Rigidbody 组件。该 Cube 属性如下图 3-4 所示：

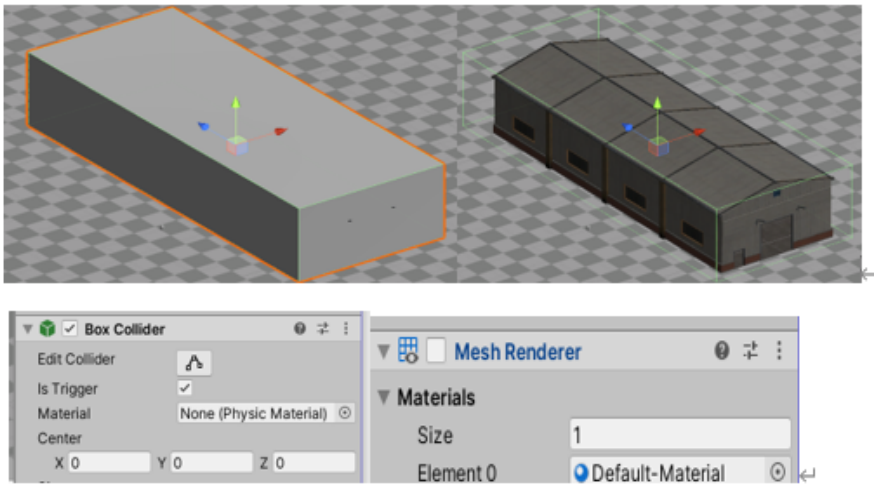

图 3-4 Cube 的属性

（2）在 Cube 上有一个 Box Collider 碰撞器组件，在玩家身上有 Character Controller 碰撞器组件，如果检测到玩家身上的碰撞器进入了该 Cube 的碰撞器，则调用 OnTriggerEnter()方法，如果检测到玩家身上的碰撞器停止接触该 Cube 的碰撞器，则调用 OnTriggerExit()方法。在这两个方法中使用 if 语句判断玩家接触的触发器是否为我们提前设定好的触发器，如果是，则开启 FogLight，如果不是则不做反应。当玩家停止接触触发器，则关闭 FogLight。具体实现代码如下：

```
//玩家进入触发器
private void OnTriggerEnter(Collider hit){
  //碰撞的是设定好的 Fog 触发器
  if (hit.gameObject == fog){
      RenderSettings.fog = true;
      }
}
//玩家停止接触触发器
private void OnTriggerExit(Collider hit){
   if (hit.gameObject == fog){
       RenderSettings.fog = false;
       }
}
```

FogLight 效果开启后如下图 3-5 所示：

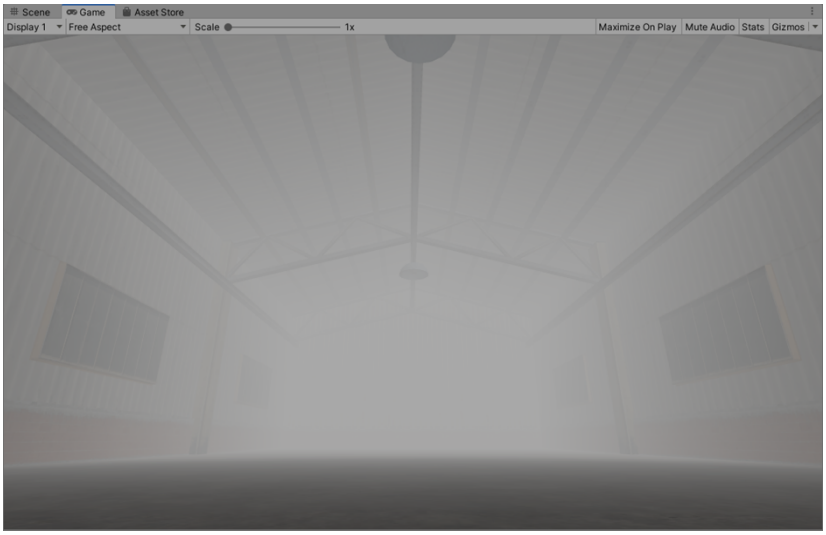

图 3-5 开启 FogLight

（3）在 Unity3D 游戏开发引擎中自带的 FogLight 中，Fog 雾有三种模式，分别是 Linear 线性模式、Exponential 指数模式以及 Exponential Squared 指数平方模式。在经过三种模式比较测试之后，最终选择了使用 Exponential 指数模式，该模式有 Density 参数表示雾的浓度，数值越大表示雾越浓。经过多次测试之后，最终选定 Fog 的 Density 参数值为 0.05，Fog 颜色选择 134/134/134/255。关于 Fog 的参数设置如下图 3-6 所示：

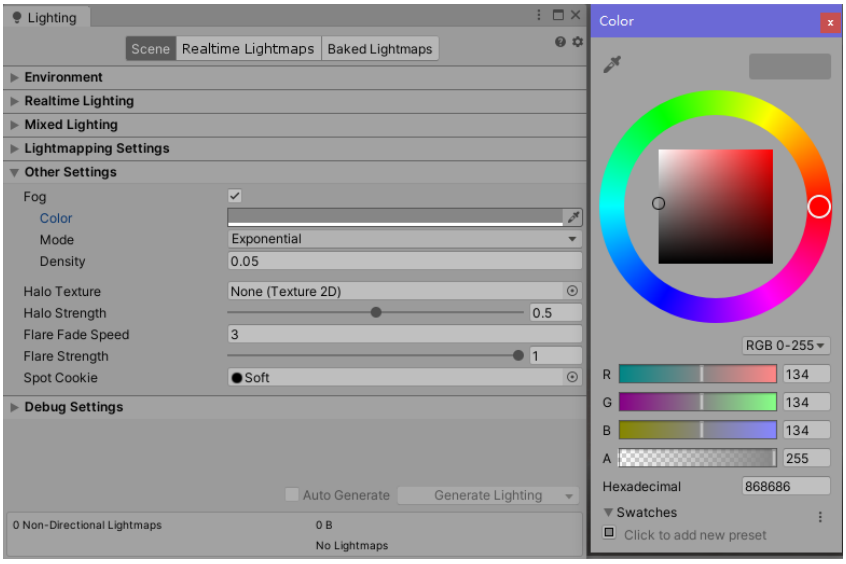

图 3-6 Fog 参数设置

3.4 灭火器模块

3.4.1 灭火器模型设置

在本次课题研究开发的游戏当中，玩家接触的灭火器总共有 4 种，分别为水基灭火器、干粉灭火器、泡沫灭火器以及二氧化碳灭火器。将使用 3ds MAX 软件做好的灭火器模型导入 Unity 中后，我们需要为其添加刚体 Rigidbody 组件与碰撞体 Mesh Collider 组件，使其存在物理效果，其次是将 4 种灭火器的 Tag 修改为不同灭火器的专属 Tag，以能够实现下文 3.4.4 中的功能。二氧化碳灭火器属性图如下图 3-6 所示：

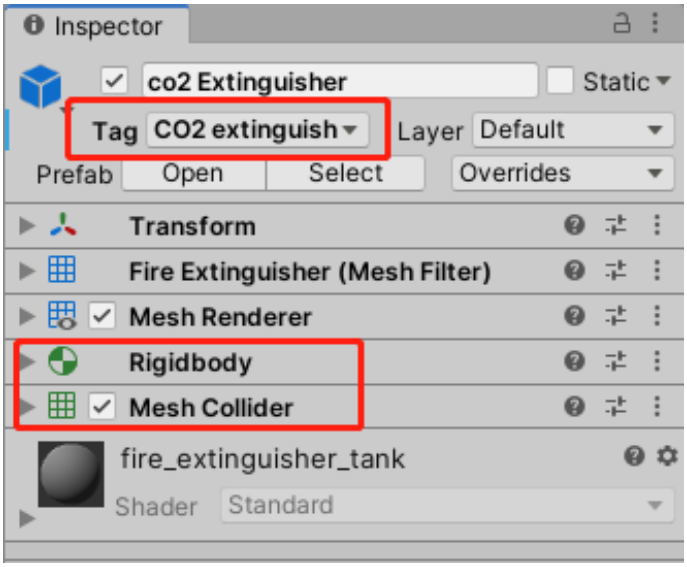

图 3-6 二氧化碳灭火器属性

3.4.2 灭火器发射灭火剂功能实现

在现实生活里，灭火器在正常情况都不会自己发射灭火剂，需要人为的进行一系列操作：拉掉保险丝，对准火源，按下压把，发射灭火剂。我们在游戏中也要实现这点，不能说拿起灭火器它就会自动发射灭火剂，那样子违背了现实，也不符合第二章游戏需求分析中所追求的真实感。因此，在本次课题研究开发的游戏当中，我们使用鼠标来模仿人的手，点击和松开鼠标左键来模仿人按下消防器压把和松开消防器压把。

首先我们创建一个用以实现开火的脚本，将其挂至我们在场景中放置的灭火剂粒子效果当中，在 Start（）方法中，默认关闭当前粒子效果，其次在 Update

（）方法当中，使用 if 语句检测鼠标的输入，当检测到按下鼠标左键时，使用ParticleSystem中的GetComponent<ParticleSystem>()方法打开粒子效果；当检测到松开鼠标左键时，同理使用该方法关闭粒子效果。具体实现代码如下：

```
void Start(){
    //默认关闭
    Water.gameObject.GetComponent<ParticleSystem>().Stop();
}

void Update(){
    //按下鼠标左键
    if (Input.GetMouseButton(0)){
        Water.gameObject.GetComponent<ParticleSystem>().Play();
    }
    //松开鼠标左键
    if (Input.GetMouseButtonUp(0)){
        Water.gameObject.GetComponent<ParticleSystem>().Stop();
    }
}
```

3.4.3 多种类灭火器问题

在现实生活中我们都知道灭火器不止一种，很多时候眼前的灭火器并不一定适用灭眼前的火情，不同种类的火源需要用到不同种类的灭火器，因此为了提高玩家对游戏真实性的感受，我们必须在游戏中体现出这一点：不一样的火情需要用相对应的灭火器才能解决。

在本次课题研究开发的游戏当中，灭火器的种类有 4 种，在进行拾取操作的时候很可能会因为灭火器参数定位错误而导致无法拾取，又或者在灭火的时候因为灭火器参数定位错误导致无法灭火。因此，我们在这里选择使用 Unity3D 游戏开发引擎中自带的枚举类型 enum 进行操作。当一个游戏拥有多种可能状态的时候，枚举 enum 是一种对我们十分有帮助的工具，它可以增强 C#脚本的准确性，以此来减少游戏的运算损耗。

首先声明一个 enum 类，用来存放所有的灭火器种类，再定义一个值用以存放灭火器种类。在游戏开始时，玩家手中是没有灭火器的，所以灭火器的默认种类定义为 None，此时是无法进行发射灭火剂操作，也无法进行灭火操作。