

基于C#与Unity3D的第一人称射击游戏

系 （院）： 软件与服务外包系

学生姓名： 沈昀

专业班级： 软件16C1

学 号： 1624051127

指导教师： 郝爱语

年 月

**声 明**

**本人所呈交的基于C#与Unity3D的第一人称射击游戏，是我在指导教师的指导和查阅相关著作下独立进行分析研究所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。**

**作者签名： 沈昀**

**日期：**

**摘要**

随着计算机技术与计算机硬件的快速发展，人们可以用计算机进行更多的应用。射击游戏具有非一般的娱乐性，能提高玩家的反应能力和双手对于距离移动的敏感度。也能在繁忙的工作之余为玩家带来放松的娱乐体验。

本次毕业设计的课题为基于C#与Unity3D的第一人称射击游戏，目的在于设计出一款具有第一人称射击游戏基本要素的简单明了的游戏。其中包括主角的移动、射击，敌人的移动、对主角的定位和攻击，以及主角与敌人的交互攻击设定。解决敌人后的分数是游戏中的“行动—收益”交互的直接表现以及最终目的。

**关键字:** 第一人称射击游戏，C#，Unity3D

With the rapid development of computer technology and computer hardware, people can use computers for more applications. Shooting games are unusually entertaining, improving the player's responsiveness and the sensitivity of the hands to distance movement. It also provides a relaxing entertainment experience for players after a busy workout.  
The subject of this graduation project is a first-person shooter based on C# and Unity3D. The goal is to design a simple and clear game with the basic elements of the first-person shooter. These include the movement of the protagonist, shooting, the movement of the enemy, the positioning and attack of the protagonist, and the interactive attack settings of the protagonist and the enemy. The score after solving the enemy is the direct expression of the "action-benefit" interaction in the game and the ultimate goal.

Keywords：FPS Game，C#，Unity3D

**目 录**

目录

[一、 引言 1](#_Toc5707215)

[（一）开发背景 1](#_Toc5707216)

[（二）研究目的与意义 1](#_Toc5707217)

[二、 系统技术及运行环境 2](#_Toc5707218)

[（一） 系统技术 2](#_Toc5707219)

[1、 Uinty3D 2](#_Toc5707220)

[2、 C#语言 2](#_Toc5707221)

[3、 Autodesk Maya 3](#_Toc5707222)

[（二） 运行环境 4](#_Toc5707223)

[三、 系统分析 5](#_Toc5707224)

[（一） 需求分析 5](#_Toc5707225)

[（二） 可行性分析 5](#_Toc5707226)

[四、 系统设计 6](#_Toc5707227)

[（一） 设计目标 6](#_Toc5707228)

[（二） 系统实现 6](#_Toc5707229)

[1、 界面设计 6](#_Toc5707230)

[2、 玩家角色设计 17](#_Toc5707231)

[3、 敌人设计 26](#_Toc5707232)

[4、 交互设计 35](#_Toc5707233)

[5、 出生点 41](#_Toc5707234)

[6、 发布游戏 43](#_Toc5707235)

[五、 总结 45](#_Toc5707236)

[六、 致谢 46](#_Toc5707237)

[七、 参考文献 47](#_Toc5707238)

[八、 附录 48](#_Toc5707239)

# 引言

## （一）开发背景

自1964年计算机诞生开始，从军事到社会科学研究到生活，人类在各种领域的生产生活活动都有了计算机的协助。而今，随着硬件系统和软件系统的不断发展，人们能够用计算机实现的东西越来越多，对计算机技术和文化产业的投入也在不断增长，以计算机为平台的娱乐发展也渐渐受到人们的了解与关注。

第一人称视角射击游戏又称FPS(First-person Shooter)游戏。FPS游戏出现时与当时市场上的其他类型游戏有很大差异，玩家通常在无法看到屏幕中的虚拟人物，以接近现实的第一人称视角进行游戏。以这种视角进行游戏不仅拉近了玩家与游戏之间的距离感，而且更加细致地、全方位地向玩家展示了游戏中想要向玩家展现的隐藏信息。将3D作为表达形式将不可避免地使玩家更接近真实的游戏体验，大大增强了游戏的主动性和真实感。其他游戏只能让玩家控制一个可见的角色，这不免使得玩家潜意识中带有违和感，而FPS游戏则带来了前所未有的代入感和视觉冲击。

1992年，《Wolfstein 3D》出现，并由id Software在DOS iPhone OS等平台上发布，这款游戏被认定为FPS游戏的开山之作。游戏中玩家需要在一个完全由纹理映射拼成的纳粹城堡里进行探索来进行游戏。

从此第一人称射击游戏便成为了11个游戏大类中不可或缺的一个部分。在FPS游戏诞生的近30年中，品质优秀、内容深刻、传播范围广、影响力大的FPS游戏层出不穷，将第九艺术的魅力发挥的淋漓尽致。

现如今第一人称射击游戏的受众范围相当大，其中著名的如CS：GO（反恐精英:全球攻势）、Rainbow Six：Siege（彩虹六号：围攻）、CrossFire（穿越火线）等成为众多游戏爱好者们热衷的第一人称射击游戏。以腾讯代理的CF为例子，2011年收入超过40亿，2012年收入近60亿，虽然近两年因以LOL（英雄联盟）为首的MOBA类游戏已经超越FPS游戏成为创收成绩更好的游戏，但这依然无法否认FPS游戏是游戏市场乃至娱乐市场中的不可忽视的一个大的方向。

## （二）研究目的与意义

本次游戏设计的目的是开发一个具有基本FPS游戏要素的小游戏，如基本的第一人称视角下的，玩家作为主角的视角移动、人物移动以及瞄准射击的要素。使自己对FPS游戏的框架和设计流程进行一个初步的认知，以及激发自己对于射击游戏开发的兴趣，同时积累对多种类开发软件的协同运用经验。

# 系统技术及运行环境

## 系统技术

### Uinty3D

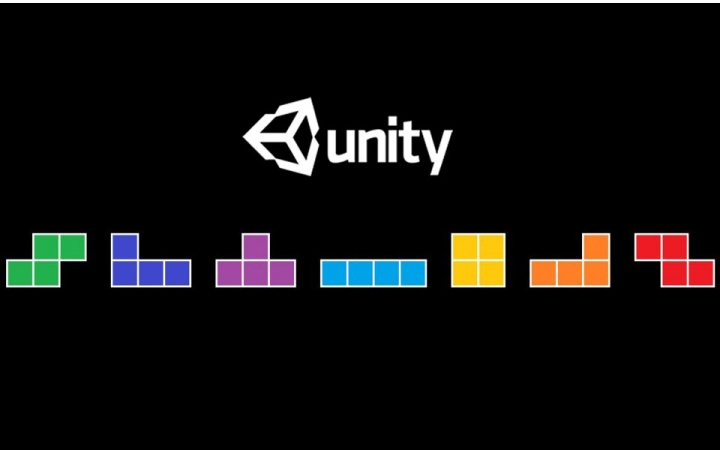


图2.1 Unity3D引擎示意图

现在制作3D游戏的主流引擎有虚幻，CryEngine 3，HeroEngine，Rage Engine，Project Anarchy等等，而本次设计我所使用的是Unity3D引擎。

Unity3D是Unity Technologies开发的技术专业游戏引擎。 它可以做很多事情，例如制作3D视频游戏，可视化建筑物，与3D动画内容交互等等。 它可以满足多平台，高度全面的用户需求。

Unity3D是游戏开发者真正负担得起的引擎，拥有众多其他引擎无法比拟的用户群体。 更重要的是，您只需支付一次，无论您的游戏有多成功，都不要担心Unity会分摊您的收入。 对于许多开发人员来说，这当然是非常有吸引力的，尤其是初创公司和新手的开发人员[1]。

介于本次设计的功能实现的要求，使用Unity3D作为开发工具进行界面与画面的开发是较为合适的选择。

### C#语言



图2.2 C#语言示意图

C＃是一种源自C和C ++的面向对象编程语言。C＃将VB的简单可视化操作与C ++的高运营效率相结合，具有以下特点：操作能力强、语法风格优雅简洁、语言创新性强，尤其在面向组件编程时使用十分便捷。这一切都使得它成为.NET开发的首选语言。

在Unity3D物理引擎中仅支持C#进行游戏脚本的编写（自2017年8月份开始，在此之前也支持UnityScript进行脚本编写），这有利于Unity引擎开发效率的提高。

以上因素决定了在本设计中，C#被用于进行游戏脚本的编写。

### https://timgsa.baidu.com/timg?image&quality=80&size=b9999_10000&sec=1552553649955&di=b1b3b477a1853ab9088e3a95bd8b0fd9&imgtype=0&src=http%3A%2F%2Fgfxhome.ws%2Fuploads%2Fposts%2F2015-04%2F1429529449_autodesk_maya_2016.jpgAutodesk Maya

图2.3 Autodesk Maya示意图

玛雅是现今游戏场景及人物建模常常会使用的软件，由Autodesk 公司出品，是世界顶级的三维动画软件[3]，应用方向是3 D动画、电影特效、游戏特效。 Maya功能齐全，灵活，具有出色的渲染效果和高生产效率。本次设计中的西部小镇场景中的建筑建模、枪械建模以及敌人建模都由玛雅进行制作。

## 运行环境

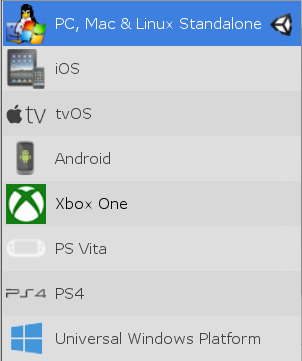
因Unity3D的强大性能，游戏可于PC，IOS，Android等众多环境中进行运行。

图2.4 多平台发布示意图

本设计中，运行环境为PC端。运行配置如下表，见表2.1：

|  |  |  |
| --- | --- | --- |
| **硬件** | **最低配置** | **推荐配置** |
| **系统** | Windows xp/ 7 / 8.1 /10 | Windows xp/ 7 / 8.1 /10 |
| **处理器** | Intel i5-6600K / AMD FX-6350 | Intel i7-4790 / AMD FX 8350 |
| **显卡** | NVIDIA GeForce GTX 660 / AMD Radeon HD7850 | NVIDIA GeForce GTX 1060 / AMD Radeon RX 480 |
| **内存** | 256MB | 256MB |
| **硬盘** | 200 MB | 200MB |

表2.1 运行配置表

# 系统分析

## 需求分析

在现今这个信息化的社会，游戏已经成为了推动经济增长的重要板块。2018年中国游戏市场实际销售收入达2144.4亿元,占全球游戏市场比例约为23.6%，其中，自主研发独立网络游戏实际收入达1643.9亿元。其中一大收入便是客户端游戏收入，总价值达619.6亿元，占中国游戏市场比例约为28.9%[2]。以此为基础的电子竞技、游戏直播、泛娱乐活动的游戏与生活的产业融合也带来了巨大的社会效益和经济效益。

根据罗切斯特大学在2009年的一份调查中发现，在游戏中向目标射击可以意外的提高玩家的视野。在调查期间，玩家们尝试了诸如“使命召唤”之类的射击游戏，并发现射击游戏玩家的“对比敏感度”显着增加。 换句话说，FPS游戏可以改善他们在夜间的视野，并帮助玩家更好地分析运动光学数据。 研究人员认为，动作游戏可以用来作为纠正视力不佳的方法。[4]

## 可行性分析

本游戏采用Visual Studio 2017与Unity 2018.2.15f1 (64-bit)开发，主要采用Unity3D框架、GameObject、Component组件等技术来实现。

# 系统设计

## 设计目标

本游戏玩法为玩家通过键盘控制角色在场景中移动，地图中会在固定地点刷新怪物，怪物会追踪玩家。玩家可以通过手中的武器对怪物进射击，对怪物造成伤害乃至消灭怪物来减少游戏中怪物的数量；通过移动操作对怪物的进攻进行躲避和逃离等操作。以上两个要素的最终能够保持玩家在游戏中的持续生存。玩家每击倒一个怪物，都会获得相应的分数。持续生存是玩家游戏的目的。

## 系统实现

### 界面设计

游戏初始页面设计

在菜单中选择File—>New Scene创建一个新场景，并将其命名为start。

创建TitleScreen.cs脚本。

using UnityEngine;

using System.Collections;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

[AddComponentMenu("Game/TitleScreen")]

public class TitleScreen : MonoBehaviour{

Button start\_button;

GameObject uicanvas = GameObject.Find("Canvas");

public void OnButtonGameLogin()

{

start\_button.gameObject.SetActive(false);

}

public void OnButtonGameStart()//响应游戏开始按钮事件

{

SceneManager.LoadScene("demo");//跳转到demo场景

}

public void OnButtonGameExit()//响应退出游戏按钮事件

{

Application.Quit();//退出

}

}

将TitleScreen.cs脚本作为组件分配给场景中的摄像机

在start场景中创建UI —>Canvas，见图4.1

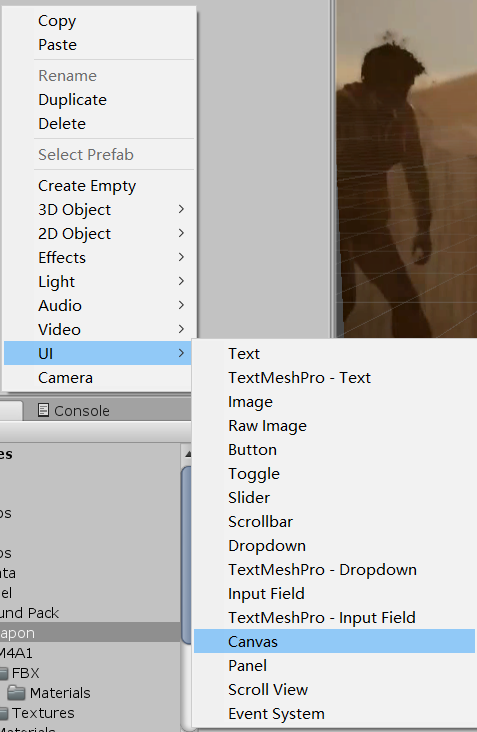


图4.1 Canvas的创建

选择UI—>Image创建图像UI

在Source Image中添加图像（Image\_background），设置图像位置和大小，见图4.2

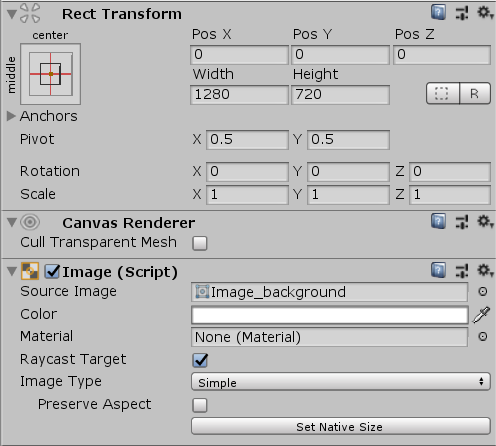


图4.2 背景UI的参数设置

同上添加按钮，UI—>Button

设置开始游戏的按钮文本和响应事件，见图4.3，图4.4

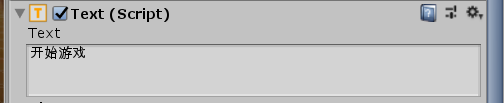


图4.3 开始游戏按钮文本编辑

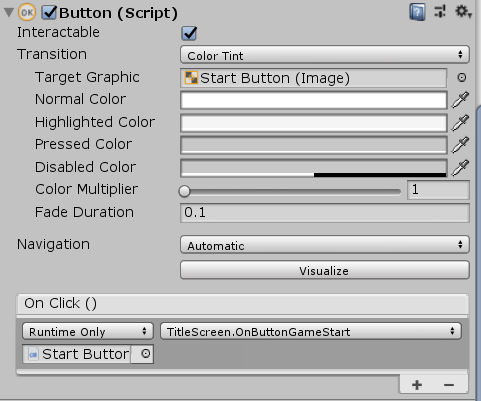


图4.4 开始游戏按钮事件绑定

设置退出的按钮文本和响应事件，见图4.5，图4.6

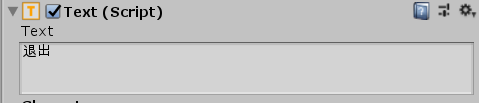


图4.5 退出按钮文本编辑

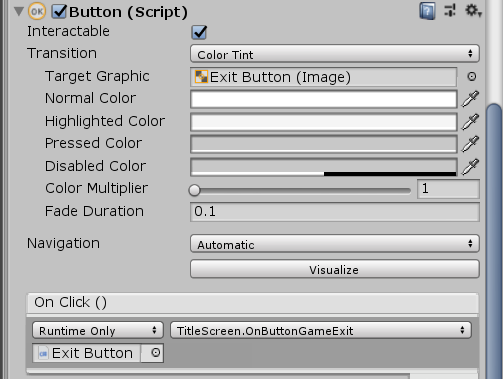


图4.6 退出按钮事件绑定

在File—>Bulid Settings中进行场景添加和排序，见图4.7，图4.8

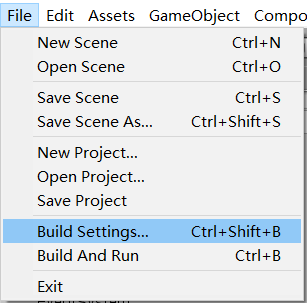


图4.7 场景设置界面



图4.8 场景排序界面

游戏场景设计

在demo中载入事先准备好的游戏场景，见图4.9



图4.9 游戏场景

然后在菜单栏中选择Component—>Physics—>Mesh Collider。我们使用这个组件为刚刚准备好的demo中的建筑添加碰撞体组件，使场景模型在可以显示的功能基础上附带有物理碰撞的特性，见图4.10。

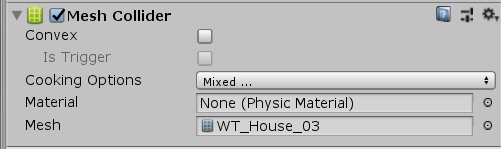


图4.10 场景碰撞体设置

游戏中UI界面设计

在demo场景中创建Canvas

创建所需要的Image、Button、Text等组件，见图4.11

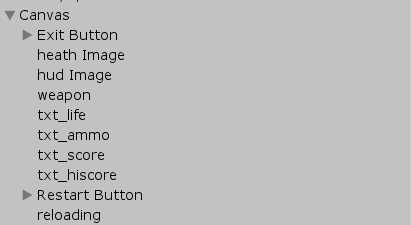


图4.11 UI组件一览

将准备好的素材代入响应的组件中，见图4.12

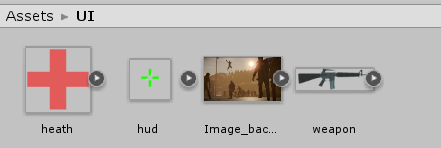


图4.12 UI素材一览

调整各组件的位置、大小等数值，见图4.13

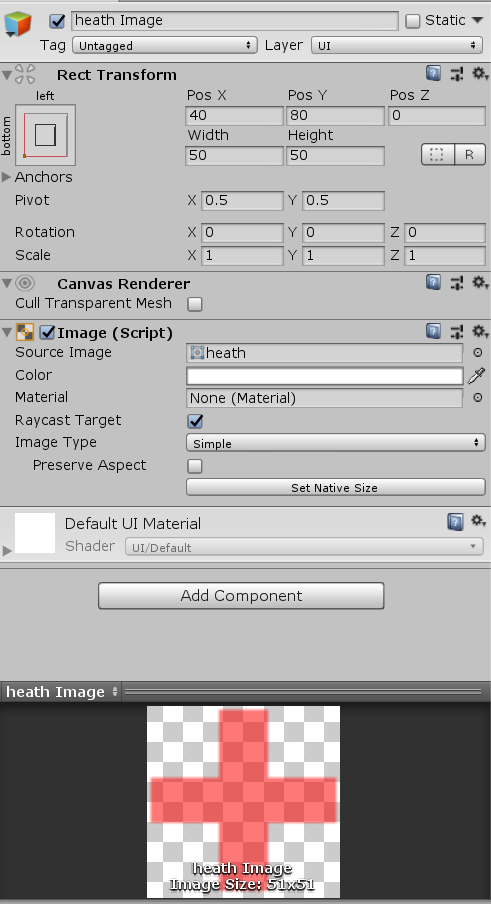


图4.13 各UI参数设置

形成一个游戏中的UI界面，见图4.14

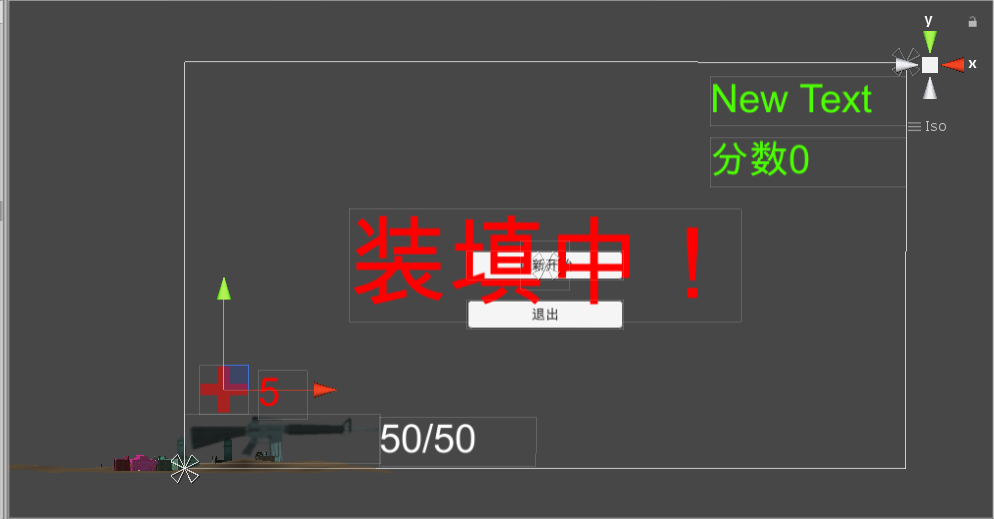


图4.14 UI界面设计

创建一个空的游戏体，将其命名为GameManager，并将其坐标设置为0，见图4.15

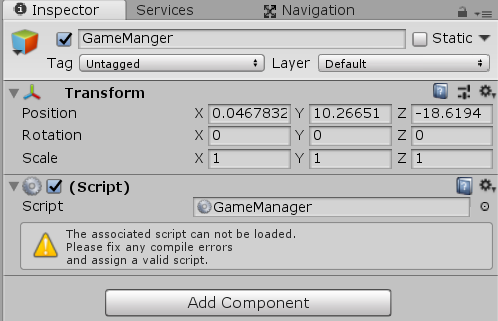


图4.15 GameManager参数设置

在VS中新建脚本，将其命名为GameManager.cs，并将其分配给相应游戏主体GameManager。

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

[AddComponentMenu("Game/GameManager")]

public class GameManager : MonoBehaviour {

public static GameManager Instance = null;

public int m\_score = 0;//游戏得分

public static int m\_hiscore = 0;//历史最高分

public int m\_ammo = 50;//弹药数量

float retime = 0;

Player m\_player;//主角

//UI文字

Text txt\_ammo;

Text txt\_hiscore;

Text txt\_life;

Text txt\_score;

Text txt\_reloading;

Button button\_restart;

Button button\_Exit;

// Use this for initialization

void Start () {

Instance = this;

m\_player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();//获取主角

//获取ui文字

GameObject uicanvas = GameObject.Find("Canvas");

foreach (Transform t in uicanvas.transform.GetComponentInChildren<Transform>())

{

if (t.name.CompareTo("txt\_ammo") == 0)

{

txt\_ammo = t.GetComponent<Text>();

}

if(t.name.CompareTo("txt\_hiscore")==0)

{

txt\_hiscore = t.GetComponent<Text>();

txt\_hiscore.text = "最高分:" + m\_hiscore;

}

if (t.name.CompareTo("txt\_life") == 0)

{

txt\_life = t.GetComponent<Text>();

}

if (t.name.CompareTo("reloading") == 0)

{

txt\_reloading= t.GetComponent<Text>();

txt\_reloading.gameObject.SetActive(false);

}

if (t.name.CompareTo("txt\_score") == 0)

{

txt\_score = t.GetComponent<Text>();

}

if (t.name.CompareTo("Restart Button") == 0)

{

button\_restart = t.GetComponent<Button>();

button\_restart.onClick.AddListener(delegate () {//设置重新开始游戏按钮事件

// 读取当前关卡

SceneManager.LoadScene(SceneManager.GetActiveScene().name);

});

button\_restart.gameObject.SetActive(false); // 游戏初期隐藏重新开始游戏按钮

}

if (t.name.CompareTo("Exit Button") == 0)

{

button\_Exit = t.GetComponent<Button>();

button\_Exit.onClick.AddListener(delegate () // 按钮事件回调

{

Application.Quit();//退出

}

);

button\_Exit.gameObject.SetActive(false); // 游戏初期隐藏重新开始游戏按钮

}

}

}

// Update is called once per frame

void Update()

{

if(!m\_player.shootflag)

{

retime += Time.deltaTime;

if(retime>=5)

{

retime = 0;

txt\_reloading.gameObject.SetActive(false);

m\_player.shootflag = true;

}

}

}

//void OnGUI()

//{

// if (m\_player.m\_life <= 0)

// {

// // 居中显示文字

// GUI.skin.label.alignment = TextAnchor.MiddleCenter;

// // 改变文字大小

// GUI.skin.label.fontSize = 40;

// // 显示Game Over

// GUI.Label(new Rect(0, 0, Screen.width, Screen.height), "Game Over");

// // 显示重新游戏按钮

// GUI.skin.label.fontSize = 30;

// if ( GUI.Button( new Rect( Screen.width\*0.5f-150,Screen.height\*0.75f,300,40),"Try again"))

// {

// Application.LoadLevel(Application.loadedLevelName);

// }

// }

//}

// 更新分数

public void SetScore(int score)

{

m\_score += score;

if (m\_score > m\_hiscore)

m\_hiscore = m\_score;

txt\_score.text = "分数 <color=yellow>" + m\_score + "</color>"; ;

txt\_hiscore.text = "最高分 " + m\_hiscore;

}

// 更新弹药

public void SetAmmo(int ammo)

{

m\_ammo -= ammo;

// 如果弹药为负数，重新填弹

if (m\_ammo <= 0)

{

m\_ammo = 50 - m\_ammo;

txt\_reloading.gameObject.SetActive(true);

m\_player.shootflag = false;

}

txt\_ammo.text = m\_ammo.ToString() + "/50";

}

// 更新生命

public void SetLife(int life)

{

txt\_life.text = life.ToString();

if (life <= 0) // 当主角生命为0时显示重新开始游戏按钮

{

button\_restart.gameObject.SetActive(true);

button\_Exit.gameObject.SetActive(true);

}

}

}

### 玩家角色设计

2.1角色控制器

通过从菜单中选择GameObject—>Create Empty来创建一个新的空游戏体。见图4.16

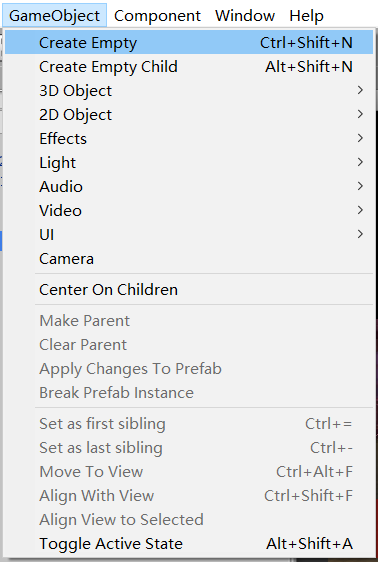


图4.16 空游戏体创建

将它的Tag设为Player，这就是我们的主角。见图4.17

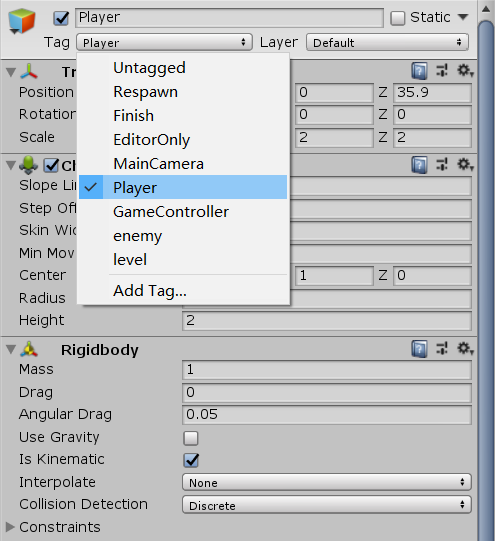


图4.17 游戏体命名及参数设置

在菜单栏中，选择Component—>Physics—>Character Controller将控制组件添加到主角色。 主角可以通过这个组件来实现对主角游戏体的碰撞控制，并且带有操控主角移动的功能。 例如，当玩家走进一个场景时，就没有穿过墙壁的虚幻现象。见图4.18

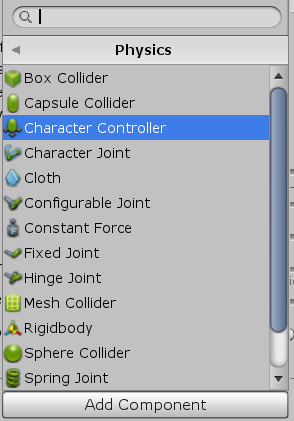


图4.18 添加角色控制组件

为主角再添加一个Rigidbody组件。见图4.19

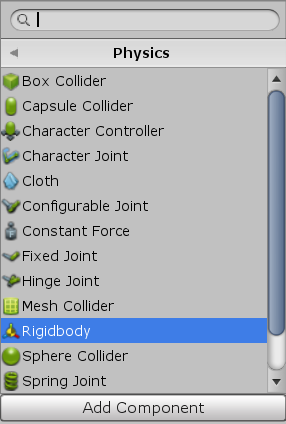


图4.19 添加Rigidbody组件

在Charatcer控制器组件中调整碰撞体的位置和大小。取消选中默认情况下选中的使用重力复选框以删除重力模拟，并选中 Is Kinematic复选框以防止它受物理计算的影响， 以便脚本可用于控制其移动[5]。见图4.20



图4.20 组件参数设置

创建Player.cs脚本，见图4.21

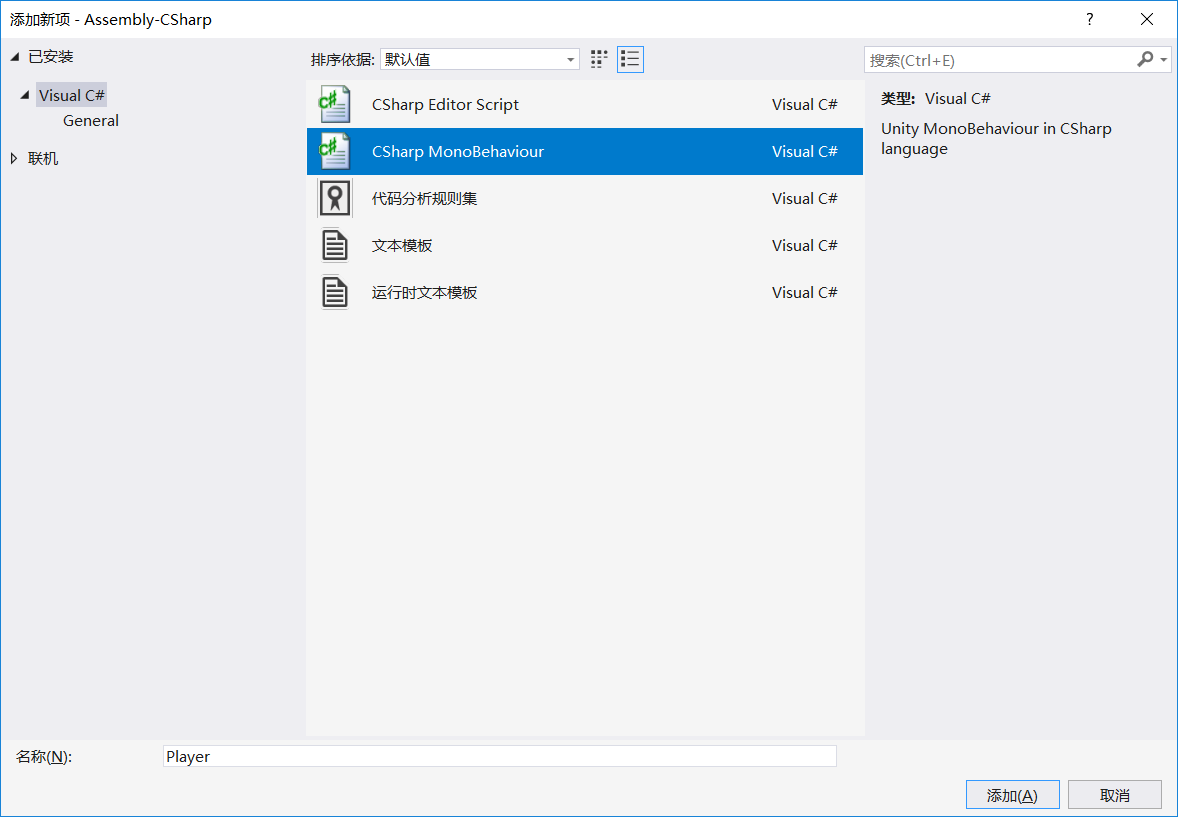


图4.21 Player脚本创建

设置一些基本角色属性并控制关键操作以执行操作。

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[AddComponentMenu("Game/Player")]

public class Player : MonoBehaviour {

public Transform m\_transform;

CharacterController m\_ch;//角色控制器控件

float m\_movspeed = 10.0f;//角色移动速度

float m\_gravity = 12.0f;//角色重力

public int m\_life = 5;//生命值

public bool shootflag = true;

// Use this for initialization

void Start () {

m\_transform = this.transform;

m\_ch = this.GetComponent<CharacterController>();//获取角色控制器控件

}

// Update is called once per frame

void Update () {

if (m\_life <= 0) return;

Control();

}

void Control()

{

float xm = 0, ym = 0, zm = 0;

ym -= m\_gravity \* Time.deltaTime;//重力

if(Input.GetKey(KeyCode.W))

{

if(Input.GetKey(KeyCode.LeftShift)) zm += 2\*m\_movspeed \* Time.deltaTime;

else zm += m\_movspeed \* Time.deltaTime;

}

else if(Input.GetKey(KeyCode.S))

{

zm -=m\_movspeed \* Time.deltaTime;

}

if (Input.GetKey(KeyCode.D))

{

xm +=m\_movspeed \* Time.deltaTime;

}

else if (Input.GetKey(KeyCode.A))

{

xm -=m\_movspeed \* Time.deltaTime;

}

if (Input.GetKeyDown(KeyCode.Space ))

{

ym += 2.0f;

}

if (Input.GetKeyDown(KeyCode.LeftControl))

{

m\_camheight = 1.0f;

}

else if (Input.GetKeyUp(KeyCode.LeftControl))

{

m\_camheight = 1.8f;

}

m\_ch.Move(m\_transform.TransformDirection(new Vector3(xm, ym, zm)));

}

private void OnDrawGizmos()

{

Gizmos.DrawIcon(this.transform.position, "Spawn.tif");

}

}

2.2摄像机

在已经存在的Player.cs脚本中进行一些添加，使玩家在操控主角移动时保证摄像机进行同步移动来使玩家接收到相应的位置变化的反馈。

添加用于控制摄像机的属性

Transform m\_camtransform;//摄像机transform

Vector3 m\_camrot;//摄像机旋转角度

float m\_camheight = 1.8f;//摄像机高度（主角身高）

修改Start函数，初始化摄像机，为固定玩家开始游戏时的视角将位置和旋转高度固定，并锁定鼠标。

void Start () {

m\_transform = this.transform;

m\_ch = this.GetComponent<CharacterController>();//获取角色控制器控件

m\_camtransform = Camera.main.transform;//获取摄像机

m\_camtransform.position = m\_transform.TransformPoint(0, m\_camheight,0);//设置相机处是位置（使用transformpoint获取play在y轴偏移一定的高度位置）

m\_camtransform.rotation = m\_transform.rotation;

m\_camrot = m\_camtransform.eulerAngles;//谁知相机旋转方向与主角一致

//Screen.lockCursor = true;//锁定鼠标

Cursor.lockState = CursorLockMode.Locked;

/\*

鼠标锁定

Cursor.lockState = CursorLockMode.Locked;

鼠标解锁

Cursor.lockState = CursorLockMode.None;

\*/

}

在Control函数中实现摄像机即玩家视角的旋转，并在玩家移动主角视角角度时与主角同步进行移动，通过鼠标的移动来控制摄像机的旋转方向，使主角跟随摄像机进行视角的旋转，在玩家移动主角位置时，摄像机也随主角的移动进行同步的移动，使游戏实现在场景中的位置移动与视角的实时变化。

void Control()

{

float xm = 0, ym = 0, zm = 0;

float rh = Input.GetAxis("Mouse X");

float rv = Input.GetAxis("Mouse Y");//获取鼠标移动距离

m\_camrot.x -= rv;

m\_camrot.y += rh;

m\_camtransform.eulerAngles = m\_camrot;//旋转相机

Vector3 camrot = m\_camtransform.eulerAngles;

camrot.x = 0;camrot.z = 0;

m\_transform.eulerAngles = camrot;//让主角的面向与摄像机一致

ym -= m\_gravity \* Time.deltaTime;//重力

if(Input.GetKey(KeyCode.W))

{

if(Input.GetKey(KeyCode.LeftShift)) zm += 2\*m\_movspeed \* Time.deltaTime;

else zm += m\_movspeed \* Time.deltaTime;

}

else if(Input.GetKey(KeyCode.S))

{

zm -=m\_movspeed \* Time.deltaTime;

}

if (Input.GetKey(KeyCode.D))

{

xm +=m\_movspeed \* Time.deltaTime;

}

else if (Input.GetKey(KeyCode.A))

{

xm -=m\_movspeed \* Time.deltaTime;

}

if (Input.GetKeyDown(KeyCode.Space ))

{

ym += 2.0f;

}

if (Input.GetKeyDown(KeyCode.LeftControl))

{

m\_camheight = 1.0f;

}

else if (Input.GetKeyUp(KeyCode.LeftControl))

{

m\_camheight = 1.8f;

}

m\_ch.Move(m\_transform.TransformDirection(new Vector3(xm, ym, zm)));

m\_camtransform.position = m\_transform.TransformPoint(0, m\_camheight,0);

}

2.3武器

在Transform中将相机的位置和旋转角度初始化为0，并将相机的Clipping Planes / Near设置为0.1。 使玩家能够在游戏场景中看到更近距离的物体并改善游戏的沉浸感。见图4.22

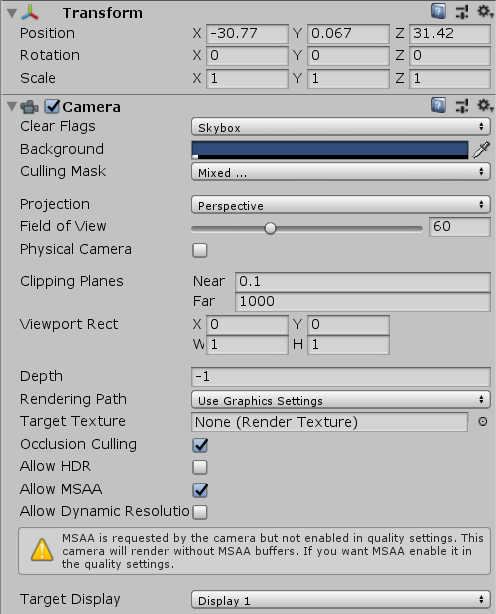
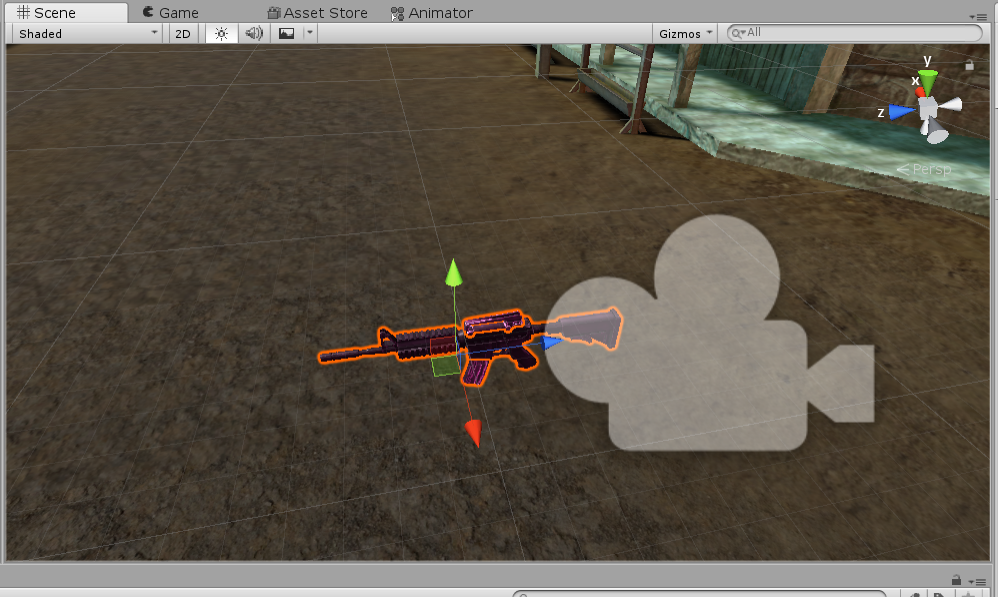
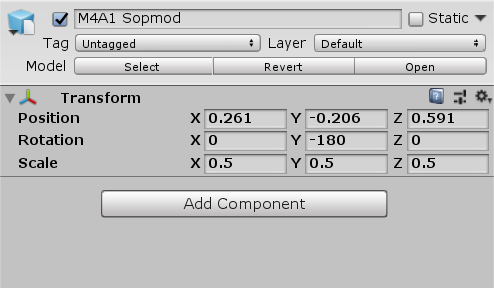


图4.22 摄像机参数设置

找到预先准备好的武器枪支模型，将其导入工程中，再拖入场景。



在Hierarchy窗口中选择武器的模型（文件为M16或M4A1 Sopmod），设置位置和旋转角度均为0。



将武器模型文件置于主摄像机分层的下方，使其变成主摄像机的子单位。



然后在场景中找到模型位置，并调整它的位置和角度，使其在第一人称视角中看上去较为自然。

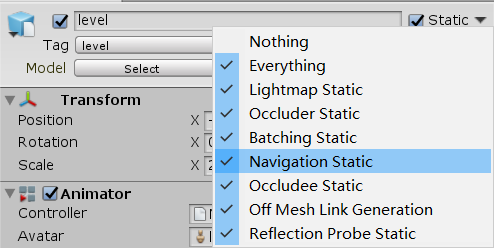




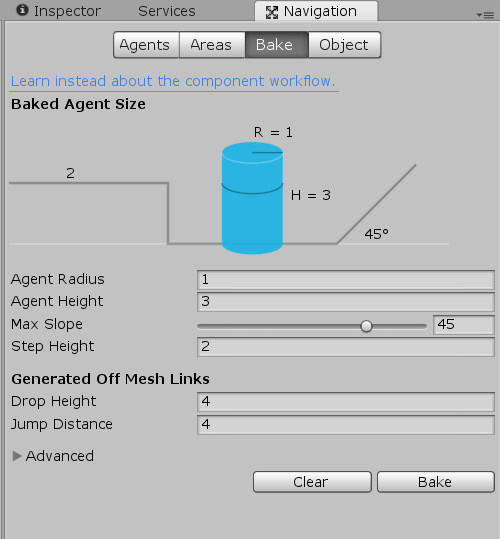
### 敌人设计

3.1寻路

选择场景模型，然后单击Inspecter窗口Static旁边的小三角形按钮以显示下拉列表，确认选择了Navigation Static，Unity将完成路径查找计算使其作用于场景模型中。



在菜单中选择Window—>Navigation，打开Navigation窗口。



Bake窗口主要是定义地形对寻路的影响。

* Agent Radius和Height可以理解为寻路者的半径和高度。
* Max slope是最大斜率，探路者无法通过超过这个设定好斜率的路径。
* Step Height是楼梯的最大高度，探路者无法通过超过这个设定好高度的路径。
* Drop Height表示探路者可以跳跃的高度限制
* Jump Distance表示探路者的跳跃极限。

在Navigation窗口中设置好选项后，点击Bake按钮对地形展开计算，单击Clear按钮会清楚计算数据。

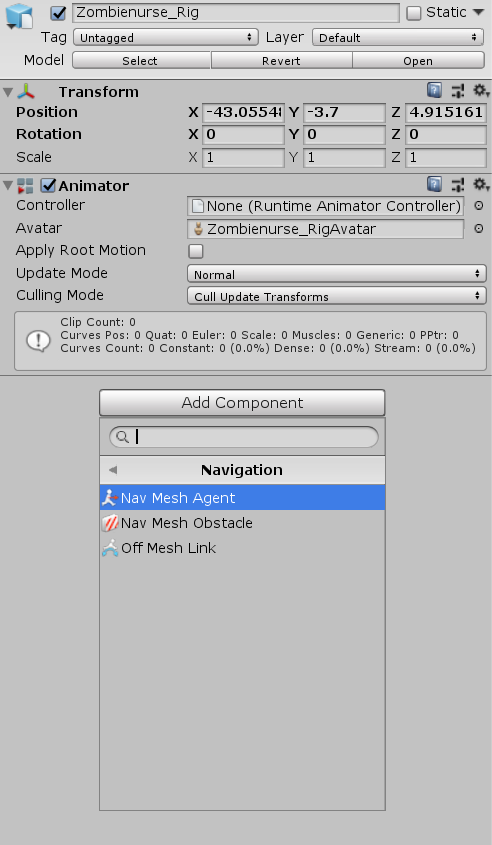
来自Bake的寻路数据将保存为名为NavMesh.asset的文件。

接下来，对探路者进行初步设置，也就是游戏中与玩家交互的主体，游戏中的敌人。

将敌人模型（Zombie）拖入场景，这是一个僵尸模型，将成为游戏中的敌人。



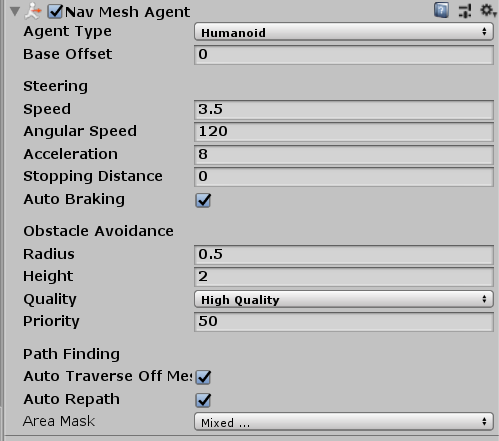
从菜单栏中选择Add Component-] Nav Mesh Agent，将寻路组件分配给敌人。



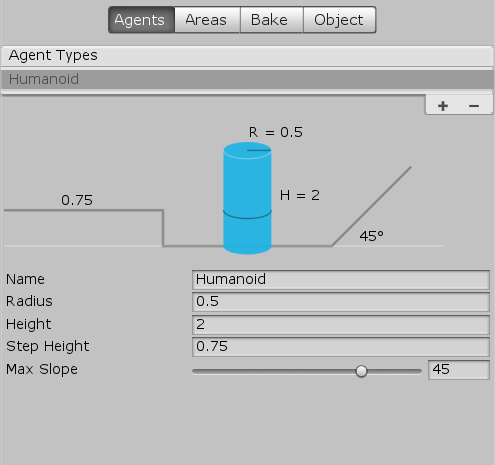
然后在Inspecter窗口中进行进一步的设置。

Speed是最大运动速度

Angular Speed是最大旋转速度。



在Agent Type中选择Open Agent Settings，可以打开Navigation的Agents窗口，设置Radius和Height，表示寻路者的半径和高度。



创建脚本Enemy.cs，在Start函数中获取NavMeshAgent组件，调用SetDsetion函数设置目标点，然后自动追逐主角。添加代码如下：

using System.Collections;

using System.Collections.Generic;

using UnityEngine.AI;

using UnityEngine;

public class Enemy : MonoBehaviour {

Transform m\_transform;

Player m\_player;//主角

NavMeshAgent m\_agent;//寻路件组

public float m\_movspeed = 2.5f;//移动速度

void Start () {

m\_transform = this.transform;

m\_ani = this.GetComponent<Animator>();// 获得动画播放器

m\_player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();//获得寻路件组

m\_agent = GetComponent<NavMeshAgent>();// 获得寻路组件

m\_agent.speed = m\_movspeed; //设置寻路器的行走速度

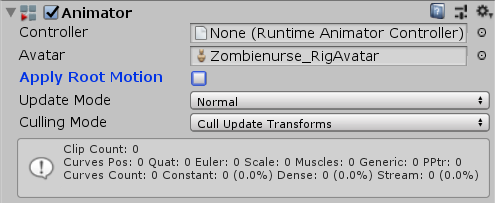
m\_agent.SetDestination(m\_player.m\_transform.position);//获得寻路目标

}

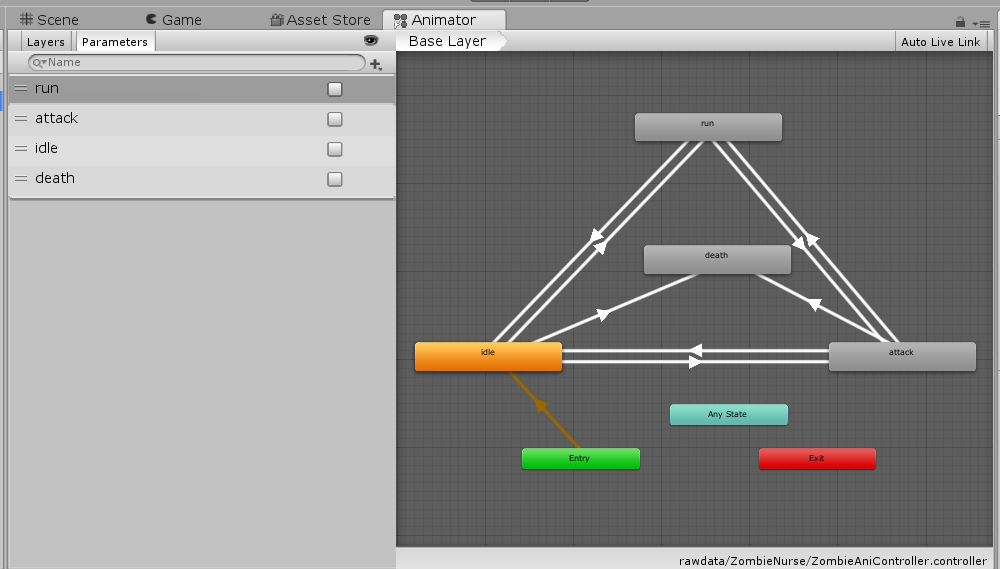
}

3.2设置动画

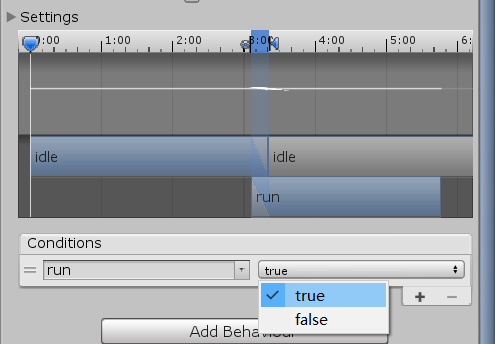
在场景中选择一个敌人。它默认有一个Animator组件，且在Controller中提前预置了一个Animator Controller。取消选中Appley Root Motion复选框以强制脚本控制游戏主体的位置，而不是依靠动画。



在菜单栏中选择Window—>Animator，打开Animator窗口，Animaotr Controller的信息都显示在这里。单击Parameters边的“+” ，创建四个Bool类型数值，名称分别为idle，run，attack，death，以将它们与动画关联并在脚本中控制它们。



在Conditions中将动画过度条件设为run，当Bool值run为true时即从待机动画转换为运动动画。



3.3行为

在Enemy.cs脚本中，添加动画组件等属性

Transform m\_transform;

NavMeshAgent m\_agent;//寻路件组

Animator m\_ani;//动画件组

float m\_rotspeed = 5.0f;//旋转速度

float m\_timer = 2;//计时器

int m\_life = 15;//生命值

protected EnemySpawn m\_spawn;//出生点

void Start () {

m\_transform = this.transform;

m\_ani = this.GetComponent<Animator>();// 获得动画播放器

m\_player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();//获得寻路件组

m\_agent = GetComponent<NavMeshAgent>();// 获得寻路组件

m\_agent.speed = m\_movspeed; //设置寻路器的行走速度

m\_agent.SetDestination(m\_player.m\_transform.position);//获得寻路目标

}

添加RotateTo函数，使敌人始终旋转到面向主角的角度：

void RotateTo(){

// 获取目标（Player）方向

Vector3 targetdir = m\_player.m\_transform.position - m\_transform.position;

// 计算出新方向

Vector3 newDir = Vector3.RotateTowards(transform.forward, targetdir, m\_rotspeed \* Time.deltaTime, 0.0f);

// 旋转至新方向

m\_transform.rotation = Quaternion.LookRotation(newDir);

}

修改Update函数：

void Update () {

if (m\_player.m\_life <= 0) return;//主角死亡什么都不做

if (m\_life <= 0)// 如果生命值为0播放死亡动画

{

m\_ani.SetBool("attack", false);

m\_ani.SetBool("run", false);

m\_ani.SetBool("death", true);

m\_agent.ResetPath();// 停止寻路

}

m\_timer -= Time.deltaTime;//更新计时器

AnimatorStateInfo stateInfo = m\_ani.GetCurrentAnimatorStateInfo(0);//获取当前动画

if(stateInfo.fullPathHash==Animator.StringToHash("Base Layer.idle") &&!m\_ani.IsInTransition(0))//如果处于待机状态而且不是过渡状态

{

m\_ani.SetBool("idle", false);

if (m\_timer > 0) return;//待机一段时间

if(Vector3.Distance(m\_transform.position, m\_player.m\_transform.position) < 3.0f)//如果距离小于1.5进入攻击动画状态

{

m\_agent.ResetPath();//停止寻路

m\_ani.SetBool("attack", true);//攻击动画

}

else

{

if (m\_life <= 0)// 如果生命值为0播放死亡动画

{

m\_ani.SetBool("death", true);

m\_agent.ResetPath();// 停止寻路

}

m\_timer = 1;//重置定时器

m\_agent.SetDestination(m\_player.m\_transform.position);//设置自动寻路点

m\_ani.SetBool("run", true);

}

}

if (stateInfo.fullPathHash == Animator.StringToHash("Base Layer.run") && !m\_ani.IsInTransition(0)) //如果处于奔跑切不是过渡

{

m\_ani.SetBool("run", false);

if(m\_timer<0)

{

m\_agent.SetDestination(m\_player.m\_transform.position);

m\_timer = 1;

}//没隔1s重新定位主角位置

if (m\_life <= 0)// 如果生命值为0播放死亡动画

{

m\_ani.SetBool("run", false);

m\_ani.SetBool("death", true);

m\_agent.ResetPath();// 停止寻路

}

if (Vector3.Distance(m\_transform.position, m\_player.m\_transform.position) <= 3.0f)// 如果距离主角小于1.5米，向主角攻击

{

// 停止寻路

m\_agent.ResetPath();

// 进入攻击状态

m\_ani.SetBool("attack", true);

}

}

if (stateInfo.fullPathHash == Animator.StringToHash("Base Layer.attack") && !m\_ani.IsInTransition(0)) // 如果处于攻击且不是过渡状态

{

RotateTo();// 面向主角

m\_ani.SetBool("attack", false);

if (stateInfo.normalizedTime >= 1.0f) // 如果动画播完，重新进入待机状态

{

m\_ani.SetBool("idle", true);

m\_timer = 2; // 重置计时器待机2秒

m\_player.OnDamage(1); // 攻击

}

}

if (stateInfo.fullPathHash == Animator.StringToHash("Base Layer.death") && !m\_ani.IsInTransition(0)) // 如果处于死亡且不是过渡状态

{

m\_ani.SetBool("death", false);

if (stateInfo.normalizedTime >= 1.0f)// 当播放完成死亡动画

{

m\_spawn.m\_enemyCount--;//更新敌人计数

GameManager.Instance.SetScore(100);// 加分

Destroy(this.gameObject);// 销毁自身

}

}

}

### 交互设计

4.1主角的射击

在脚本Player.cs中，添加OnDamage函数，该函数用于在主角受伤时降低血量值，并在UI界面中将其显示为信息以提供玩家知晓。当生命值在0以下时，判定角色死亡，取消鼠标在游戏中的锁定状态。

public void OnDamage(int damage)

{

m\_life -= damage;

// 更新UI

GameManager.Instance.SetLife(m\_life);

// 取消锁定鼠标光标

if (m\_life <= 0)

Cursor.lockState = CursorLockMode.None;

}

为主角添加几个新属性

Transform m\_muzzlepoint;// 枪口transform

public LayerMask m\_layer;// 射击时，射线能射到的碰撞层

public Transform m\_fx; // 射中目标后的粒子效果

public AudioClip m\_audio;// 射击音效

float m\_shootTimer = 0;// 射击间隔时间计时器

将代码添加到Start函数中以获取枪口位置。

m\_muzzlepoint = m\_camtransform.Find("M16/weapon/muzzlepoint").transform;

将代码添加到Update中以实现显示射击功能。使用Physics.Raycast射出一条射线，在游戏中由玩家通过准心UI进行瞄准，如果射出的射线击中了敌人，判定与敌人产生碰撞，对敌人造成设定好的伤害量，并依据量减少敌人一定的生命值。

m\_shootTimer -= Time.deltaTime;

// 鼠标左键射击

if (Input.GetMouseButton(0) && m\_shootTimer <= 0)

{

m\_shootTimer = 0.1f;

this.GetComponent<AudioSource>().PlayOneShot(m\_audio);

// 减少弹药，更新弹药UI

GameManager.Instance.SetAmmo(1);

// RaycastHit用来保存射线的探测结果

RaycastHit info;

// 从muzzlepoint的位置，向摄像机面向的正方向射出一根射线

// 射线只能与m\_layer所指定的层碰撞

bool hit = Physics.Raycast(m\_muzzlepoint.position, m\_camtransform.TransformDirection(Vector3.forward), out info, 100, m\_layer);

if (hit)

{

// 如果射中了Tag为enemy的游戏体

if (info.transform.tag.CompareTo("enemy") == 0)

{

Enemy enemy = info.transform.GetComponent<Enemy>();

// 敌人减少生命

enemy.OnDamage(1);

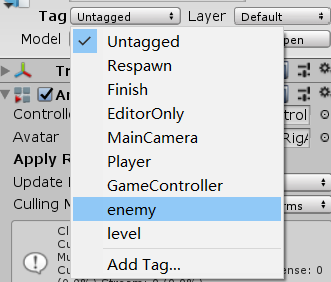
}

// 在射中的地方释放一个粒子效果

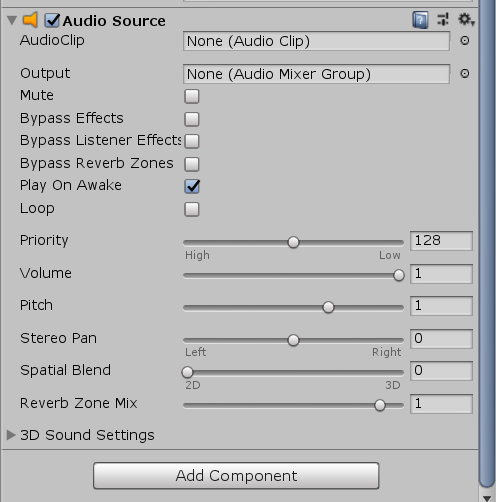
Instantiate(m\_fx, info.point, info.transform.rotation);

}

添加两个碰撞层：enemy和level。将enmey碰撞层分配给敌人，level碰撞层分配给场景模型，创建一个Tag属性，命名为enemy，分配给敌人。



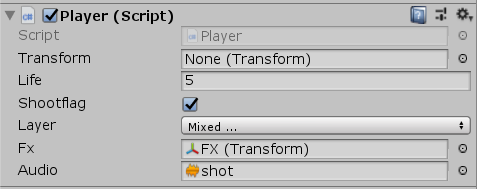
在场景中选择游戏的主体，并将Audio Source组件添加到其中。



将Player脚本组件中的Layer设为enemy和level，以便主角在开火时射击的射线可以击中敌人和场景，使游戏看起来更逼真。



在素材中找到预先准备好的特效素材，分别为射击时的击中目标的时得到反馈的特效（FX.Prefab）与射击时的音效（shot.WAV）



创建一个新的脚本AutoDestory.cs，将其分配给击中目标特效（FX.Prefab），一段时间内这将自动摧毁游戏体。

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[AddComponentMenu("Game/AutoDestroy")]

public class AutoDestroy : MonoBehaviour{

public float m\_timer = 1.0f;

// Use this for initialization

void Start () {

Destroy(this.gameObject, m\_timer);//通常可以采用缓存方式避免在游戏中使用ｉｎｓｔａｎｔｉａｔｅ和ｄｅｓｔｒｏｙ

}

// Update is called once per frame

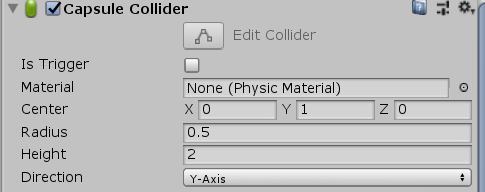
void Update () {

}

}

4.2敌人的进攻与死亡

选择敌人，在菜单栏中选择Component—>Physics—>Capsule Collider，为敌人添加碰撞体。



将OnDamage函数添加到enemy.cs脚本以更新敌人的伤害，当敌人的血量数值小于等于0时，则判定状况为死亡。

public void OnDamage(int damage)

{

m\_life -= damage;

if (m\_life <= 0)// 如果生命值为0播放死亡动画

{

//m\_ani.SetBool("attack", false);

//m\_ani.SetBool("run", false);

m\_ani.SetBool("death", true);

m\_agent.ResetPath();// 停止寻路

}

}

在Update函数中添加死亡状态，当敌人的死亡动画播完后，更新UI界面并销毁自身。

if (stateInfo.fullPathHash == Animator.StringToHash("Base Layer.death") && !m\_ani.IsInTransition(0)) // 如果处于死亡且不是过渡状态{

m\_ani.SetBool("death", false);

if (stateInfo.normalizedTime >= 1.0f)// 当播放完成死亡动画

{

m\_spawn.m\_enemyCount--;//更新敌人计数

GameManager.Instance.SetScore(100);// 加分

Destroy(this.gameObject);// 销毁自身

}

}

修改Update函数中的攻击状态，并执行主角的OnDamage函数以更新主角的生命值。

if (stateInfo.fullPathHash == Animator.StringToHash("Base Layer.attack") && !m\_ani.IsInTransition(0)) // 如果处于攻击且不是过渡状态

{

RotateTo();// 面向主角

m\_ani.SetBool("attack", false);

if (stateInfo.normalizedTime >= 1.0f) // 如果动画播完，重新进入待机状态

{

m\_ani.SetBool("idle", true);

m\_timer = 2; // 重置计时器待机2秒

m\_player.OnDamage(1); // 攻击

}

}

### 出生点

为了让敌人不断出现，我们需要定期在场景中以不同的间隔设置一定数量的敌人。

创建脚本 EnemySpawn.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[AddComponentMenu("Game/EnemySpawn")]

public class EnemySpawn : MonoBehaviour {

public Transform m\_enemy; // 敌人的Prefab

public int m\_enemyCount = 0; //生成的敌人数量

public int m\_maxEnemy = 3;// 敌人的最大生成数量

public float m\_timer = 0;// 生成敌人的时间间隔

protected Transform m\_transform;

// Use this for initialization

void Start () {

m\_transform = this.transform;

}

// Update is called once per frame

void Update () {

if (m\_enemyCount >= m\_maxEnemy)

return;

m\_timer -= Time.deltaTime;// 每间隔一定时间

if (m\_timer <= 0)

{

m\_timer = Random.value \* 15.0f;

if (m\_timer < 5)

m\_timer = 5;

Transform obj = (Transform)Instantiate(m\_enemy, m\_transform.position, Quaternion.identity);// 生成敌人

Enemy enemy = obj.GetComponent<Enemy>();// 获取敌人的角本

enemy.Init(this);// 初始化敌人

}

}

void OnDrawGizmos()

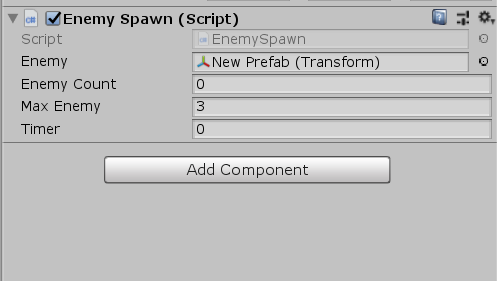
{

Gizmos.DrawIcon(transform.position, "item.png", true);

}

}

创建一个空游戏体，指定EnemySpawn.cs脚本，并关联敌人

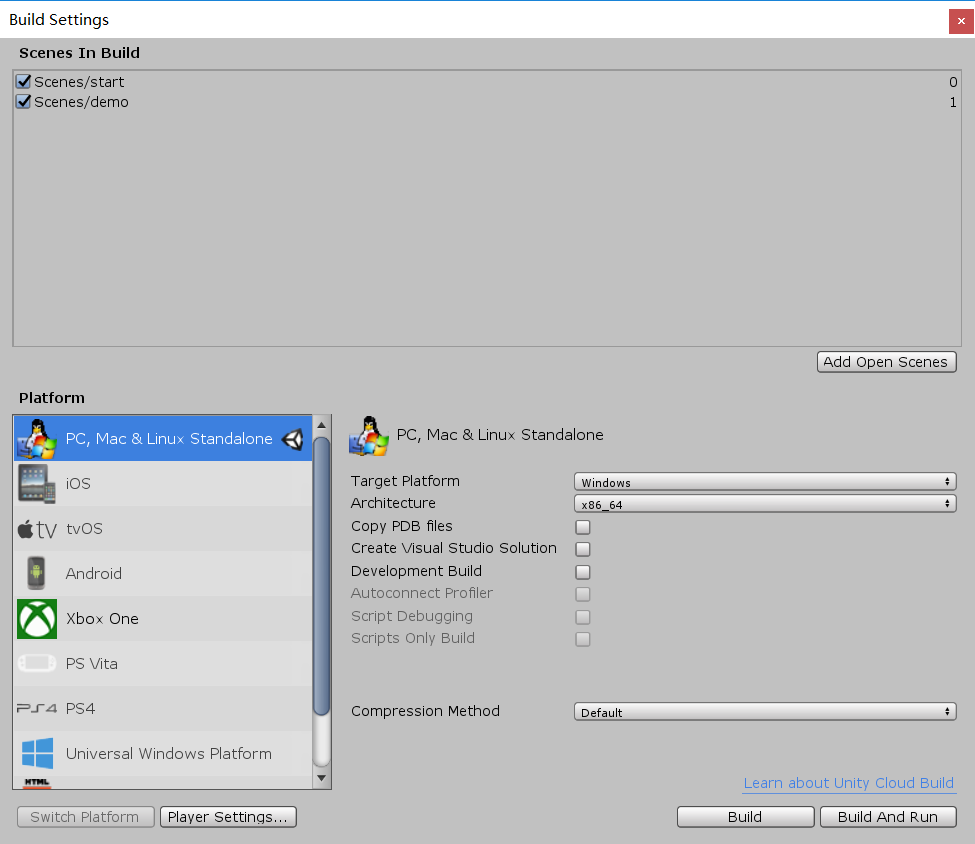


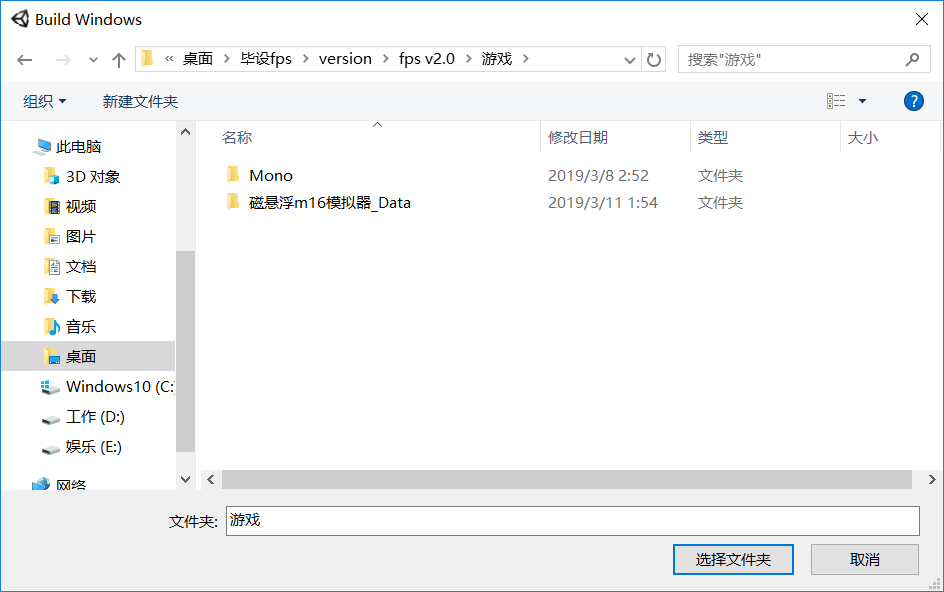
在地图中用同样的方式设置多个重生点



### 发布游戏

打开File—>Bulid Settings，选择PC，执行Build，选择要保存的地址。





# 总结

本设计是根据我的自身需求来进行构建的，它主要实现了第一人称游戏的基本要素，使用了在图形方面较为成熟的Unity3D和在代码编写中较为简洁方便的Visual Studio 2017进行脚本的编写。

在设计的进行过程中，由于自身水平的限制，使设计存在着不足之处：

1. 场景中的光影效果十分诡异并且不自然。
2. 在交互射击的实现中存在一直无法解决的问题。
3. 玩家无法以自己的意愿去进行换弹操作以及换弹中只有“装填中”三字的僵硬过渡。
4. 游戏场景建模的不合理性以及模型缺乏精细度导致游戏质量差。

以上总结的几点是本设计在之后需要不断完善的地方，还需要很长时间的调试和改进。

# 致谢

在相当大的时间跨度中，我完成了本设计的雏形与我的毕业论文。从前期的简单的需求到设计流程的不断清晰到最后的发布与测试中，我都要感谢我的同学与指导老师。在实现毕业设计的过程中，他们给予了我非常大的帮助，伴随着我一点点一步步完成整个设计。对于即将毕业的我来说，这段时间的磨炼与协作，是我走上工作岗位之前非常宝贵的经验。同样我也感谢我自己，兴趣的风帆推动着我，成为我完成设计的动力和决心。我相信在将来的工作中，兴趣依旧是驱使我不断前进的重要力量。

最后我还要感谢苏州工业职业技术学院的安静无干扰的环境，让我有时间和空间来静心完成我的设计，实现不一样的毕业成果。

# 参考文献

[1] 十大游戏开发引擎优缺点对比. <https://blog.csdn.net/English0523/article/details/72820183>

[2] CNG中新游戏研究（伽马数据）. 2018年中国游戏产业报告[M].中国书籍出版社，出版年份：2018，12-20。

[3] 关于maya软件. <https://zhidao.baidu.com/question/158380671.html>

[4] 游戏设计给人类带来的7大益处. <https://wenku.baidu.com/view/bd002c716edb6f1aff001f90.html>

[5] 《Unity3D手机游戏开发》.65

# 附录