東南大學

课程设计(论文)报告

院　　系：仪器科学与工程

课　　程：虚拟现实技术

设计名称：3D 跑酷游戏

小组成员：陈欣

李欣怡

张亚宁

汤玉玲

指导老师：孙立博

目录

基于 Unity 平台的 3D 跑酷游戏设计

摘要：本实验利用 Visual Studio 2017 平台以及 Unity 2020 平台，从构建 3D 场景构建开始，逐步完善，最后通过，基于带物理仿真的虚拟机器人以及多元化的虚拟场景再现，实现了 3D 跑酷游戏的构建。同时，本实验还增加了运动控制模块，用户可以使用鼠标进行交互，进而操控游戏角色与场景互动。不同的道具会触发不同的效果，最终以量化形式呈现，增加用户的成就感，使得用户能够获得身临其境的游戏体验。

关键词：游戏设计；Unity 平台

一、实验目的与方法

本次实验的目的时运用在虚拟现实技术课程中所学到的理论知识，结合扩展阅读及网络检索得到的扩充资料，完成基于 Unity 的多场景 3D 跑酷游戏设计，实现相关功能。主要内容包括虚拟机器人设计、虚拟场景绘制、交互实现以及场景互动的量化呈现。本程序的设计过程锻炼了我们的程序编写与调适能力、信息检索与筛选能力、游戏设计与测试能力以及对包括 Visual Studio、Unity 在内多平台的功能掌握与协调运作能力。

在实验过程中，我们以 Windows 10 为操作系统，协同使用包括 Visual Studio 2017 以及 Unity 2020 在内的多平台，结合 EasyTouch 插件，实现了实验目标。

二、实验平台介绍

2.1 Unity 基础介绍

Unity, 也称 Unity 3D，是一个让用户轻松创建诸如三维视频游戏、建筑可视化、实时三维动画等类型互动内容的多平台的综合型开发工具，同时也是是一个全面整合的专业游戏引擎。Unity 3D 以其强大的跨平台特性与绚丽的 3D 渲染效果而闻名，因此，即便市面上有品类繁多的引擎可供使用，现在很多商业游戏及虚拟现实产品还是选择采用 Unity 3D 引擎进行开发。这样的结果与其完善的技术以及丰富的个性化功能密不可分，具体而言，可分为以下几点：

跨平台

开发者可以通过不同的平台进行开发。就游戏项目而言，一旦制作完成，无需任何修改即可直接一键发布到常用的主流平台上，包括 Windows、Linux、MacOS X、iOS、Android、Xbox360、PS3 以及 Web 等。

以往的游戏开发，往往需要开发者考虑平台之间的差异，比如屏幕尺寸、操作方式、硬件条件等。Unity 3D 很大程度上为开发者简化了这些问题，为游戏开发者节省了大量时间。

综合编辑

Unity 3D 的用户界面具备视觉化编辑、详细的属性编辑器和动态预览特性。Unity 3D 创新的可视化模式让开发者能够轻松构建互动体验，当项目运行时可以实时修改参数值，方便开发，为开发节省大量时间。

资源导入

Unity 3D 项目可以自动导入资源，并根据资源的改动自动更新。Unity 3D 支持几乎所有主流的三维格式，如 3ds Max、Maya、Blender 等，贴图材质自动转

换为 U3D 格式，并能和大部分相关应用程序协调工作。

地形编辑器

Unity 3D 内置强大的地形编辑系统，该系统可使游戏开发者实现游戏中任何复杂的地形，支持地形创建和树木与植被贴片，支持自动的地形 LOD、水面特效，尤其是低端硬件亦可流畅运行广阔茂盛的植被景观，能够方便地创建游戏场景中所用到的各种地形。

物理特效

Unity 3D 内置 NVIDIA 的 PhysX 物理引擎，其中使用了质量、速度、摩擦力和空气阻力等变量。，游戏开发者可以用高效、逼真、生动的方式复原和模拟真实世界中的物理效果，例如碰撞检测、弹簧效果、布料效果、重力效果等。

光影

Unity 3D 提供了具有柔和阴影以及高度完善的烘焙效果的光影渲染系统。

2.2 Unity 的资源工作流程

资源表示 Unity 项目中用来创建游戏或应用的任何项，可以代表项目中的视觉或音频元素，例如 3D 模型、纹理、精灵、音效或音乐。资源还可以表示更抽象的项目，例如任何用途的颜色渐变、动画遮罩或任意文本或数字数据。就来源而言，资源可能来自 Unity 外部创建的文件，例如 3D 模型、音频文件、图像。还可以在 Unity 编辑器中创建一些资源类型，例如 ProBuilder 网格 (ProBuilder Mesh)、Animator Controller、混音器 (Audio Mixer) 或渲染纹理 (Render Texture)。

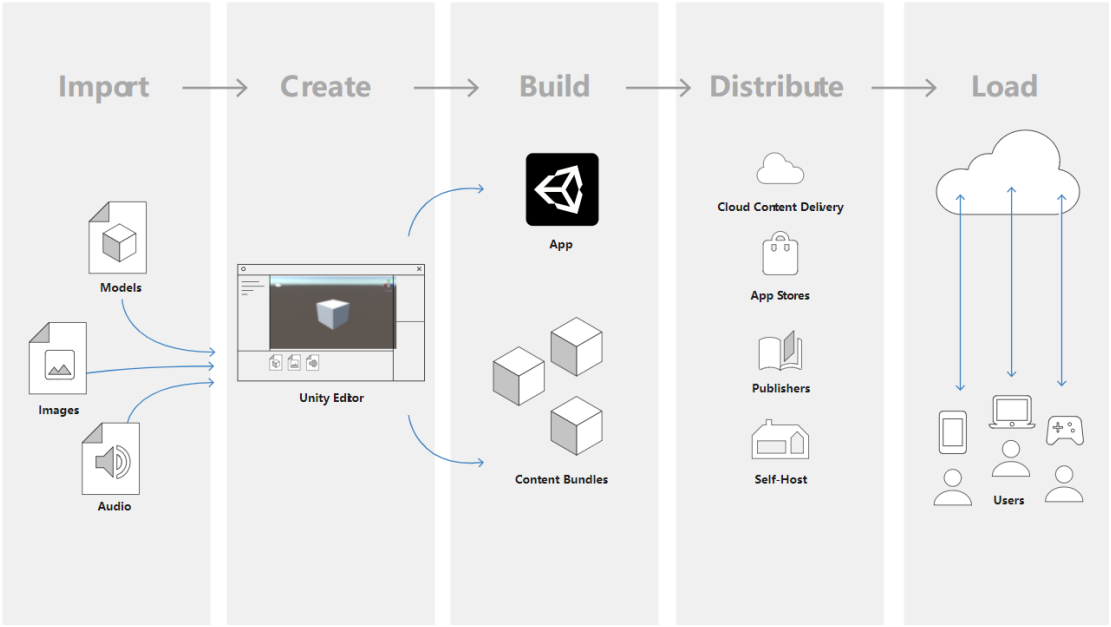

图 2.1 Unity 的资源工作流程图

根据上图，可将资源工作流程归纳如下：

1) 导入：将资源导入 Unity 编辑器以进行处理。
2) 创建：将资源作为游戏对象放置到一个或多个场景中，并添加脚本来控制用户如何与它们交互。
3) 构建：将完成的项目导出为二进制文件，以及可选的随附内容包
4) 发布：发布构建的文件，以便用户可以通过发布者或应用程序商店访问

5) 加载：根据用户的行为以及发布内容的规则和编程方式，在运行时根据需要加载进一步更新。

2.3 EasyTouch

EasyTouch 是一个 3D Unity 插件，可以实现屏幕的触摸检测和虚拟控制功能，适用于设计移动端的虚拟触摸检测或虚拟摇杆，能够解决快速移动端的输入接入问题。

就功能而言，ETC（Easy Touch Controls）控件集中有两种类型的 Joystick：

1) Static（静态的）：摇杆将会出现在预先设定好的地方。
2) Dynamic（动态的）：摇杆将会出现在触碰到屏幕的地方，也可以强制摇杆只出现在某个范围内。

每个摇杆控件由两个图片组成，一个作为背景，一个作为按键。摇杆控件的大小跟背景图片的大小相关，可以由宽度和高度参数设置；每个轴（X 和 Y 轴）返回参数大小范围为-1~1，也可以通过参数设置是否要打开对应的轴向。

三、游戏简介与操作说明

实验程序主要实现了类似地铁跑酷的游戏功能。

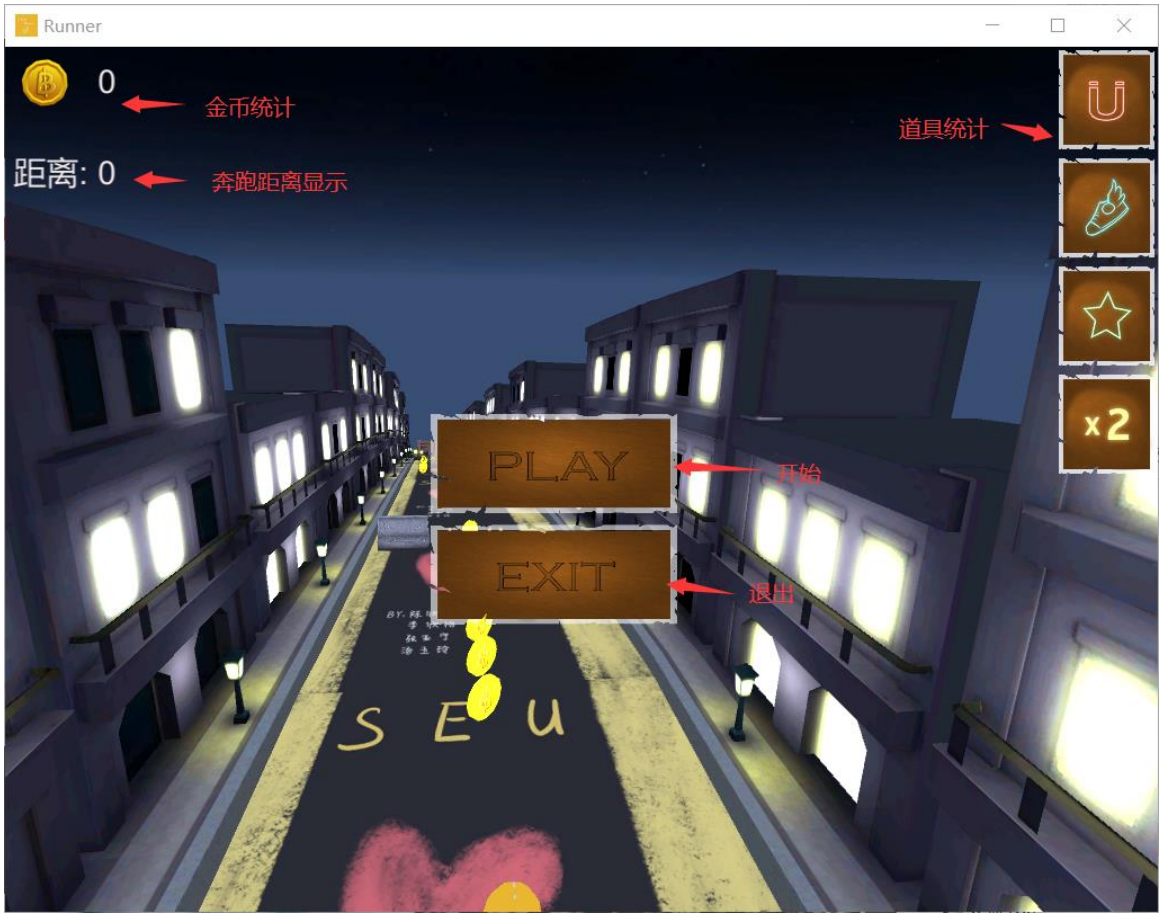

图 3.1 程序开始界面

在程序处于开始界面时，左上角显示累计金币数量，右上角显示当前拥有的道具和道具时效，右下角是游戏声音的开关，界面中心是选择开始和退出的按钮。

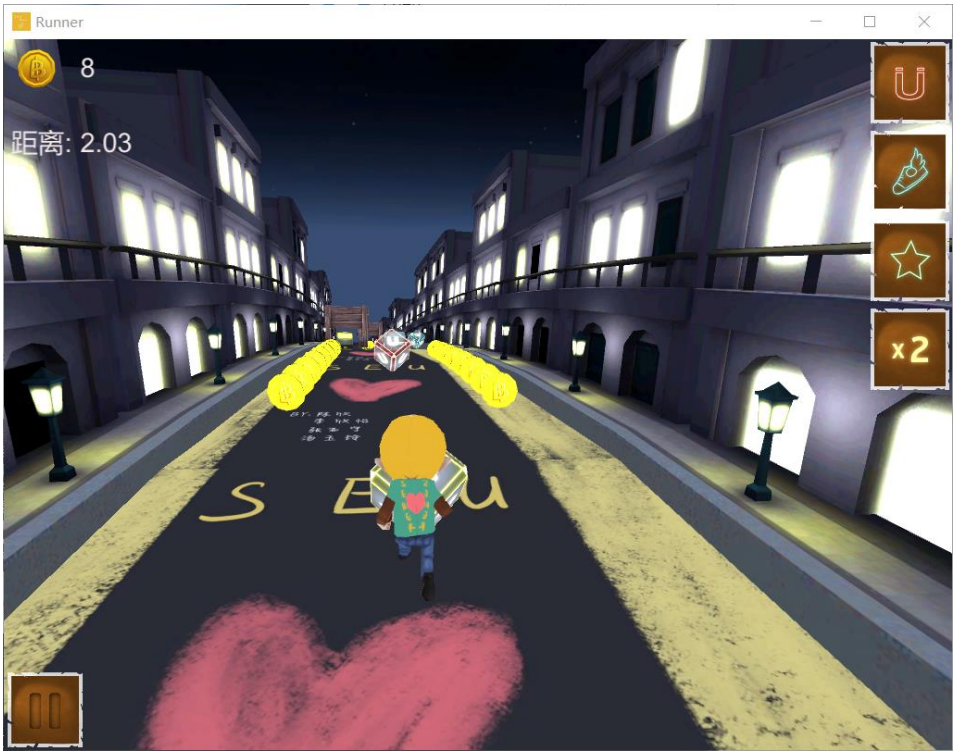

图 3.2 游戏运行界面

游戏开始后，角色进入向前运动的状态，用户可以通过鼠标左右滑动控制游戏角色的左右移动，鼠标上滑角色进行跳跃，鼠标下滑角色进行翻滚，以此来躲避游戏中的障碍物，当游戏角色撞到障碍物时，游戏结束，可以选择重新开始或者退出；用户还可以控制游戏角色接触场景中的金币，来获取金币奖励；游戏运行时左下角会有一个暂停按钮，可以用来暂停游戏。


a.吸铁石


b.双倍金币


c.跑鞋


d.暴走

图 3.3 游戏道具

游戏中有四种道具，如上图所示，分别是吸铁石、跑鞋、暴走和双倍金币，触碰到吸铁石后，角色周围的金币会自动吸附到角色身上，跑鞋可以让角色进行两段跳，暴走卡会加速游戏时间，双倍金币卡会将获取的金币数量翻倍，这些道具都有十秒的时效，在右上角的道具统计栏中会有时效显示。

四、实验过程

4.0 游戏脚本总览

本项目中所有脚本及其对应功能如下表所示：

表 4.1 游戏所用脚本及其功能

ChangeFloor	实现地板的更迭
PlayerController	实现角色的奔跑、方向的控制
CameraManager	用于控制角色的动画
AnimationManager	用于控制角色的动画
Item	用于设置碰撞体的父类
AutoDestroy	用于实现撞击后对象的消失，将其挂载到相应碰撞特效上
Obstacle	障碍物父类
PatternManager	用于管理场景中的物体
Star	用于管理暴走道具的脚本
Board	路障类（Obstacle 的子类）
GameAttribute	游戏部分重要参数的管理
GameController	用于控制游戏的进程
AudioManager	用于游戏音效的管理
Coin	硬币类（Item 的子类）
MagnetCollider	用于实现磁铁吸附金币的脚本
Magnet	磁铁类（Item 的子类）
Multiply	双倍积分类(Item 的子类)
Shoe	跑鞋类（Item 的子类）
Star	加速星星类（Item 的子类）
UIController	用于游戏 UI 的控制

4.1 角色与道路的初始化

（1）建立奔跑的角色：首先按图 4.1.1 所示方法设置场景天空盒，效果如图 4.1.2 所示。添加角色模型与道路，由于道路素材 modelRoad 的长度为 8，故设计时每隔 8 个单位增加一个 modelRoad 素材，由三个 modelRoad 拼接成 Road1（长度为 24），再由三个 modelRoad 拼接称 Road2。之后，再添加道路两旁的建筑模型，便可完成场景的基本搭建。

实现角色的奔跑可以分为添加动画与角色的位移两个部分。角色的位移实际上就是实时更改角色对象的 z 轴坐标。记 z 轴正方向的单位向量为$\vec{z}$，角色奔跑的速度为speed，时间为ΔTime，则角色在ΔTime时间内的位Δz可用公式（1）表示，并使用图 4.1.3 的代码（PlayerController0）实现角色的奔跑。

$$\Delta z = \vec{z} \times speed \times \Delta Time \quad (1)$$

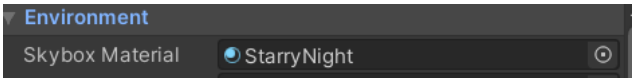

图 4.1.1 天空盒设置

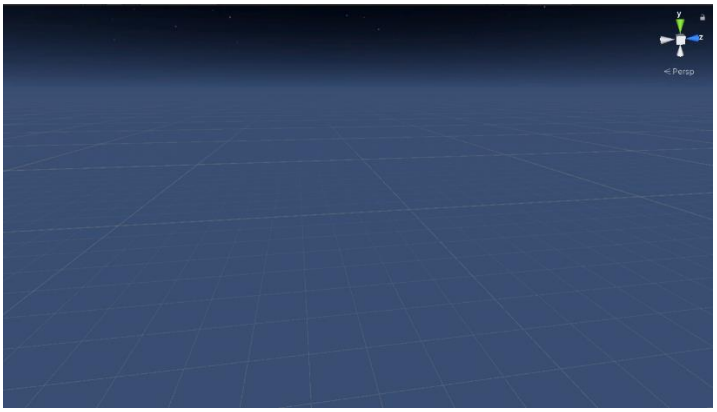

图 4.1.2 天空盒设置效果

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController0 : MonoBehaviour
{
    [SerializeField] float speed = 2;//角色的初始速度
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.Translate(Vector3.forward * speed * Time.deltaTime);//改变角色的z轴坐标
    }
}
```

图 4.1.3 代码：实现角色奔跑

（2）相机的跟随：人物在跑动时，相机应该跟随其进行运动，使用图 4.1.4 的 CameraManager 脚本使用插值对相机位置进行跟随（Target：所创建的角色）。

```
public class CameraManager0 : MonoBehaviour
{
    [SerializeField] GameObject target;//跟随目标
    [SerializeField] float height;//距离目标的高度
    [SerializeField] float distance;//距离目标的距离

    private Vector3 pos;
    // Start is called before the first frame update
    void Start()
    {
        pos = transform.position;
    }

    // Update is called once per frame
    void Update()
    {

    }

    private void LateUpdate()
    {
        //使相机的位置无限接近于角色位置
        pos.x = Mathf.Lerp(pos.x, target.transform.position.x, Time.deltaTime);
        pos.y = Mathf.Lerp(pos.y, target.transform.position.y+height, Time.deltaTime);
        pos.z = Mathf.Lerp(pos.z, target.transform.position.z-distance, Time.deltaTime);

        transform.position = pos;//更新相机位置
    }
}
```

图 4.1.4 代码：相机的跟随

（3）路面的循环：如（1）中所述，所建立的道路由 Road1 和 Road2 拼接而

成，长度均为 24，故道路总长度为 48，这显然不满足游戏设计的需求。所以，本设计建立了一个 ChangeFloor0 脚本（如图 4.1.5），来实现地板的循环。角色初始位于 Road2 上，当角色的 z 轴坐标超过 Road2 的 z 轴坐标 24 个单位以上时，说明其即将超出路面范围，这时将 Road1 沿 z 轴平移 24 个单位，替换当前地面即可。

```
public class ChangeFloor0 : MonoBehaviour
{
    [SerializeField] GameObject currentFloor;//当前的地板
    [SerializeField] GameObject theNextFloor;//下一块地板
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (transform.position.z >= currentFloor.transform.position.z + 24)
        {
            currentFloor.transform.position = new Vector3(0, 0,theNextFloor.transform.position.z + 24);
            GameObject temp = currentFloor;
            currentFloor=theNextFloor;
            theNextFloor = temp;
        }
    }
}
```

图 4.1.5 代码：地面的循环

4.2 手势判断

使用 EasyTouch 插件中的 Quick Swipe 来进行手势的判断。创建四个 Quick Swipe 方法，分别判别上、下、左、右四个方向的滑动，如图 4.2.1 所示。

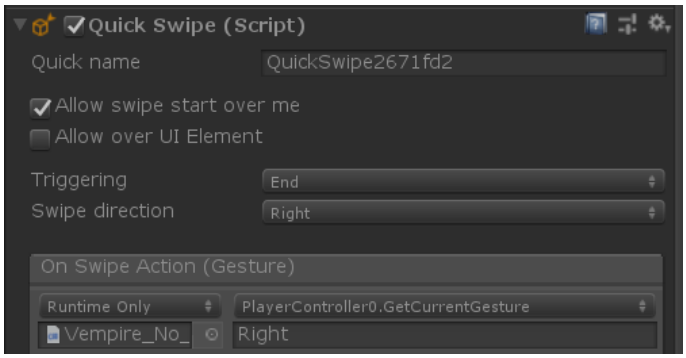

图 4.2.1 Quick Swipe 的设置（例：向右滑动）

将 Animation 设置为一个单例，使用 C#中内置的 Action 委托（本脚本中命名为 animationHandler），完成角色方向改变时相应动画的播放。并且要在每个动画播放的结尾，再将动画切换为奔跑动画。如图 4.2.2 是播放向右动画的方法。

```
public void PlayTurnRight()
{
    animation.Play(TurnRight.name);

    //如果动画播放到尾声，则自动切换到Run动画
    if (animation[TurnRight.name].normalizedTime > 0.95f)
    {
        animationHandler = PlayRun;
    }
}
```

图 4.2.2 代码：播放向右动画

```
private void PlayAnimation()
    {
        if (direction != null)
        {
            switch (direction)
            {
                case "Down":
                    AnimationManager.instance.animationHandler = AnimationManager.instance.PlayRoll;
                    break;
                case "Left":
                    AnimationManager.instance.animationHandler = AnimationManager.instance.TurnLeft;
                    break;
                case "Right":
                    AnimationManager.instance.animationHandler = AnimationManager.instance.TurnRight;
                    break;
                case "Up":
                    AnimationManager.instance.animationHandler = AnimationManager.instance.PlayJumpUp;
                    break;

                default:
                    break;
            }
        }

    }
```

图 4.2.3 代码：实现手势检测并播放相应动画

4.3 角色的左右平移实现

用枚举 Position 来定义角色左、中、右的三个位置，添加角色控制器（Character Controller）来控制左右的移动。角色在向左或者向右移动时，仍然要保持向前奔跑，故此时角色在ΔTime 时间的位移应该为Δx +Δz，其中Δz仍按照公式（1）计算，按照公式（2）计算。其中$\vec{x}$为沿 x 轴正方向的单位向量；xDis 为用于增大角色平移幅度的参数，通过实验发现，当 xDis 为 4 时，比较合适。

$$\Delta x = \begin{cases} \vec{x} \times xDis \times \Delta Time(\text{右移}) \\ -\vec{x} \times xDis \times \Delta Time(\text{左移}) \end{cases} \qquad (2)$$

以左移为例，向左移动时，首先要停止当前角色的动画，并且播放向左移动的动画，再将人物位移设置为向 x 轴负方向移动，并且改变当时角色的 position，如图 4.3.1。

```
    /// <summary>
    /// 往左移动
    /// </summary>
    private void MoveLeft()
    {
        if (position != Position.Left)
        {
            GetComponent<Animation>().Stop();
            AnimationManager.instance.animationHandler = AnimationManager.instance.PlayTurnLeft;

            xDirection = Vector3.left;
            if (position == Position.Middle)
            {
                position = Position.Left;
                formPosition = Position.Middle;
            }
            else if (position == Position.Right)
            {
                position = Position.Middle;
                formPosition = Position.Right;
            }
        }

    }
```

图 4.3.1 代码：角色向左移动方法

使用以上方法便可完成角色的左右移动，但是角色会跑出道路以外，故使用图 4.3.2 的方法来对人物的 x 轴位置进行限制，当其 x 轴位置达到一定数值时就不再改变，停止位置如表 4.3.1 所示。

```
/// <summary>
    /// 左右位置偏移的限制
    /// </summary>
    private void MoveLeftRight()
    {
        if (position == Position.Middle)
        {
            if (formPosition == Position.Left) //从左边往中间移动
            {
                if (transform.position.x > 0)
                {
                    xDirection = Vector3.zero;
                    transform.position = new Vector3(0, transform.position.y, transform.position.z);
                }
            }
            if (formPosition == Position.Right) //从右边往中间移动
            {
                if (transform.position.x < 0)
                {
                    xDirection = Vector3.zero;
                    transform.position = new Vector3(0, transform.position.y, transform.position.z);
                }
            }

        }
        if (position == Position.Left)
        {
            if (transform.position.x <= -1.7)//主角已经到达最左边
            {
                xDirection = Vector3.zero;
                transform.position = new Vector3(-1.7f, transform.position.y, transform.position.z);
            }
        }
        if (position == Position.Right)
        {
            if (transform.position.x >= 1.7)//主角已经到达最右边
            {
                xDirection = Vector3.zero;
                transform.position = new Vector3(1.7f, transform.position.y, transform.position.z);
            }
        }
    }
```

图 4.3.2 代码：角色 x 轴位置限制方法

移动方向 / 起始位置	左	右
左	-	0
中	-1.7	1.7
右	0	-

表 4.3.1 人物移动停止 x 轴坐标

4.4 角色的跳跃、翻滚动作实现

当检测到用户向上或向下的手势时，应根据其在地面和不在地面的两种状态，给出不同的响应，其逻辑如图 4.4.1 所示。

图 4.4.1 角色跳跃、翻滚动作控制逻辑

角色应实时受到重力的影响，使得其在弹跳后可以正常回落到地面上。故$\Delta Time$时间内，角色在 y 轴方向上恒有一个位移Δy，计算方式如公式（3）。其中 gravity 为重力大小，本实验中设置为 10。如此，角色在空中可以自然落地。但

同时，又要保证角色在地面上时不会落到地面以下，实验中使用的是给路面添加网格碰撞器（Mesh Collider）的方法，如图 4.3.2 所示。

$$\Delta y = -\vec{y} \times gravity \times \Delta Time \quad (3)$$

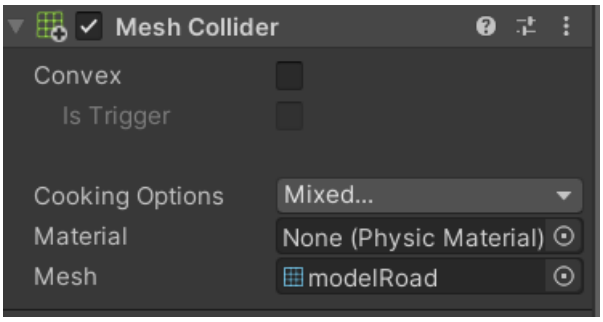

图 4.3.2 给路面添加网格碰撞器

按照图 4.4.1 的逻辑，设计如图 4.3.3 的 MoveForward 方法，当检测到向上的手势时，根据角色的不同状态，有不同的结果，并实现当角色处于空中时，下滑触发快速落地。

```
/// <summary>
/// 往前以及上下移动
/// </summary>
void MoveForward()
{
    if (characterController.isGrounded)
    {
        //如果当前不在播放Roll、TurnLeft、TurnRight动画
        if (AnimationManager0.instance.animationHandler != AnimationManager0.instance.PlayRoll &&
            AnimationManager0.instance.animationHandler != AnimationManager0.instance.PlayTurnLeft &&
            AnimationManager0.instance.animationHandler != AnimationManager0.instance.PlayTurnRight)
        {
            //那么就要播放run动画
            AnimationManager0.instance.animationHandler = AnimationManager0.instance.PlayRun;
        }

    }
    else
    {
        //如果当前不在播放JumpUp、Roll、DoubleJump动画
        if (AnimationManager0.instance.animationHandler != AnimationManager0.instance.PlayJumpUp
            && AnimationManager0.instance.animationHandler != AnimationManager0.instance.PlayRoll
            && AnimationManager0.instance.animationHandler != AnimationManager0.instance.PlayDoubleJump)
        {
            //则播放JumpLoop动画
            AnimationManager0.instance.animationHandler = AnimationManager0.instance.PlayJumpLoop;
        }
    }

    if (dir == "Down")   //向下滑动
    {
        QuickGround();  //快速落地
    }
    if (doubleJump)
    {
        if (dir == "Up") //向上滑动
        {
            JumpDouble();   //双连跳
            doubleJump = false;     //退出双连跳状态
        }
    }

}
```

4.3.3 代码：MoveForward 方法

4.5 碰撞检测与金币设置

设置 Item 父类，用于检测碰撞，编写脚本，为游戏中的对象添加撞击后的特效，如图 4.5.1。设置金币为刚体，为其添加网格碰撞，设置其为触发器，当撞击到金币时，触发金币累加事件。金币数Coin的计算方法如式(4)所示，其中$Coin_0$为原始金币数，multiply 用于记录角色是否拾取到双倍积分道具，未拾取到时为 1，拾取到时为 2。

$$\text{Coin} = Coin_0 + multiply \times 1 \tag{4}$$

```
public class Item : MonoBehaviour
{
    public float rotateSpeed = 1;

    public GameObject hitEffect;

    public void Update()
    {
        //转动
        transform.Rotate(0,rotateSpeed * Time.deltaTime,0);
    }
    /// <summary>
    /// 撞击后的处理
    /// </summary>
    public virtual void HitItem()
    {
        PlayHitAudio();
        //生成撞击特效
        GameObject effect = Instantiate(hitEffect);

        //设置 父物体为主角
        effect.transform.parent = PlayerController.instance.gameObject.transform;

        //设置局部坐标
        effect.transform.localPosition = new Vector3(0, 0.5f, 0);

        //销毁自己
        Destroy(gameObject);
    }

    public virtual void OnTriggerEnter(Collider other)
    {
        //撞击的是主角
        if (other.tag == "Player")
        {
            HitItem();
        }
    }
    public virtual void PlayHitAudio()
    {
        AudioManager.instance.PlayGetitemAudio();
    }

}
```

图 4.5.1 代码：用于检测碰撞的父类 Item

4.6 障碍物设置

（1）路障

将路障设置为刚体，并为其设置碰撞盒。当角色与碰撞盒相撞时，角色的生命值减一（初始生命值为 1）。此时，对生命值进行判断，如果生命值为 0，角色死亡，游戏结束，如图 4.6.1。但是这样角色即使向下翻滚，也会与碰撞盒进行碰撞。故设置变量 isRool 来标记角色是否处于翻滚状态，只有当角色处于非翻滚状态时，才进行碰撞处理，如图 4.6.2。

```
public virtual void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            CameraManager.instance.CameraShake();
            GameAttribute.instance.life -= hurtValue;
            moveSpeed = 0;
            AudioManager.instance.PlayHitAudio();
        }

    }
```

图 4.6.1 代码：障碍物碰撞检测

```
public override void OnTriggerEnter(Collider other)
    {
        if (!PlayerController.instance.isRool)
        {
            base.OnTriggerEnter(other);
        }

    }
```

图 4.6.2 代码：路障的碰撞处理

（2）斜面

在斜面上创建一个倾斜的空物体，并为其设置碰撞盒，如图 4.6.3。如此，角色在遇到斜面时，就可以走上斜面。

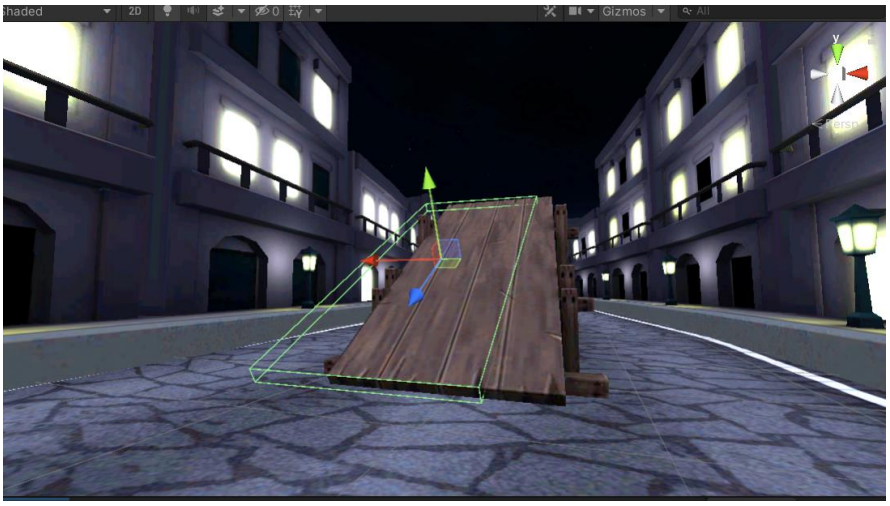

图 4.6.3 斜面碰撞盒的设置

（3）公交车、小汽车、隧道、桥梁等障碍物

公交车的正面与角色碰撞时，会触发碰撞，而在其顶部、左边和右边，只需要为其设置碰撞盒，让角色无法穿过即可，如图 4.6.4 是右侧的例子。并为公交车设置运动，其原理类似于实现角色的奔跑，只需改变运动的方向即可。小汽车、隧道、桥梁等障碍物的设计原理都与公交车相同，即在不同位置设置碰撞盒，并挂上 Obstacle 脚本。

图 4.6.4 公交车侧面碰撞盒的设置

4.7 场景的随机切换

建立 PatternManager 类来对场景中的物体进行管理，其包含场景物体集合的类 PatternItem 以及管理单个场景物体的 Pattern 类，如图 4.7.1。对这两个类均进行可序列化，以在 Unity 中可视化。在进入每一段 Road1 或者 Road2 时，都先对所有道具和障碍物进行清空，再用图 4.7.2 的方法，用随机数下标来产生随机道具。

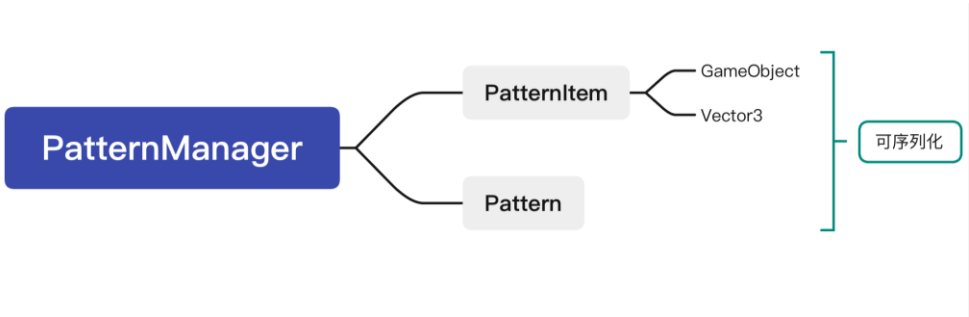

图 4.7.1 PatternManager 类的结构

```
void AddItem(GameObject floor)
    {
        Transform item = floor.transform.Find("Item");
        if (item != null)
        {
            var patternManager =   PatternManager.instance;

            if (patternManager != null && patternManager.patterns != null && patternManager.patterns.Count > 0)
            {
                var pattern = patternManager.patterns[Random.Range(0, patternManager.patterns.Count)];

                if (pattern != null && pattern.patternItems != null && pattern.patternItems.Count > 0)
                {
                    foreach (var patternItem in pattern.patternItems)
                    {
                        var newObj = Instantiate(patternItem.gameobject);
                        newObj.transform.parent = item;
                        newObj.transform.localPosition = patternItem.position;
                    }
                }
            }

        }
    }
```

图 4.7.2 代码：道具的随机产生

4.8 游戏道具设置

对于所有的道具，都是用模型 ModelTheCube 加上对应的贴图实现的。同样将每个道具设置为刚体，并设置碰撞盒。对于每个道具，创建一个 Text 来显示剩余时间。暴走道具只需在检测碰撞后增加角色向前移动的速度 speed 即可；二级跳道具只需将用于标记用户是否能够进行二级跳的标记置为 1（二级跳的功能在 4.4 节已经实现）；双倍积分道具只需将式(4)中的 multiply 置为 2 即可。下面详细介绍磁铁道具的设计过程。

不同于其他道具的是，磁铁道具并不能仅仅通过简单地改变一些标记或参数来实现，其还要实现一个磁铁吸附金币的效果。据此，在角色周围建立一个较大的球形碰撞体，如图 4.8.1 所示。当用户在触发吸铁石的碰撞体以后，该球形碰撞体开始检测与金币的碰撞。当与金币发生碰撞后，金币的位置就向主角的位置进行插值（MagnetCollider 脚本中），如图 4.8.2。

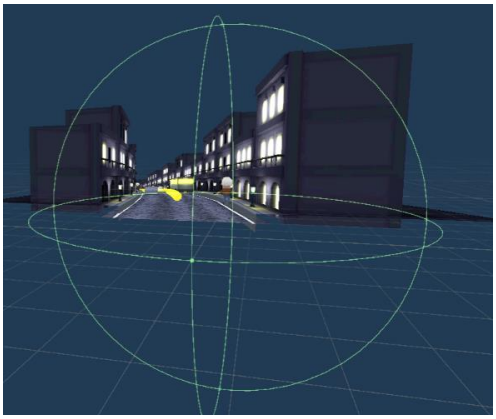

图 4.8.1 用于检测金币的球形碰撞体

```
IEnumerator HitCoin(GameObject coin)
{
    bool isLoop = true;
    while (isLoop)
    {
        if (coin == null)
        {
            isLoop = false;
            continue;
        }
        coin.transform.position = Vector3.Lerp(coin.transform.position, PlayerController.instance.gameObject.transform.position,
        //金币的位置向主角的位置进行插值

        if (Vector3.Distance(coin.transform.position, PlayerController.instance.gameObject.transform.position) < 0.5)
        {
            coin.GetComponent<Coin>().HitItem();
            isLoop = false;
        }
        yield return null;
    }
}
```

图 4.8.2 代码：用于实现磁铁吸附金币的方法

4.9 UI 制作

创建五个按钮，分别为 PLAY、EXIT、RESTART、RESUME 以及暂停按钮，并为每个按钮增添相应事件。在不需要相应 UI 显示在画面中时，将对应按钮移动到画面外，而在需要相应的按钮时，将按钮移动到画布内。在设计时，获取屏幕的宽度与高度，这样即使屏幕的分辨率发生改变，按键也能显示在正确的位置。如图 4.9.1 为设置好的 UI 界面。

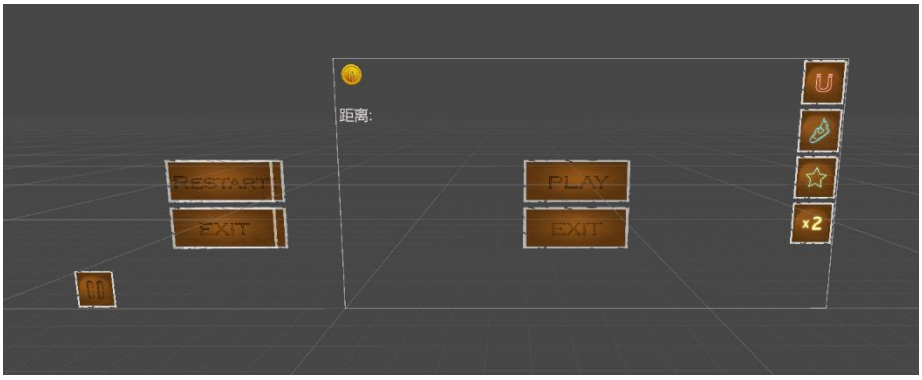

图 4.9.1 创建好的 UI 界面

4.10 SEU 元素设计

本项目还设计了一些 SEU 的元素，将其贴图到不同的游戏对象中。例如软件的图标、桥面以及路面的图案。

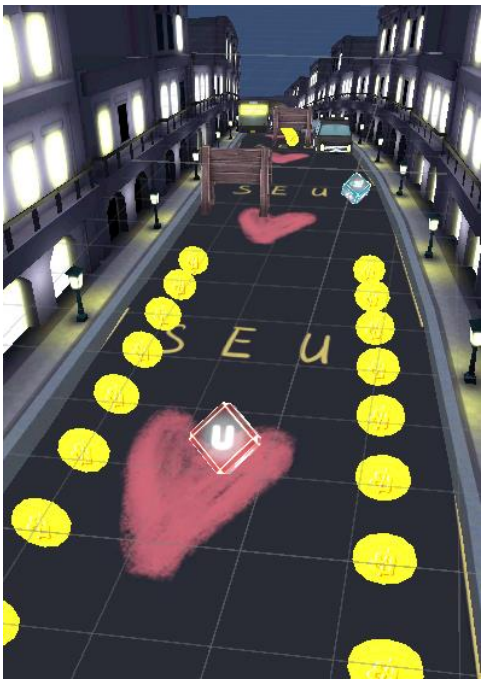

图 4.10.1 带有 SEU 元素的路面

4.10.2 带有 SEU 元素的桥面

4.10.3 软件图标

五、心得体会

陈欣：本次的游戏设计让我第一次 Unity 项目的开发，实验过程中我深刻谈体会到了使用 Unity 进行场景搭建的便捷性。这也是我第一次接触到 C#，但是在脚本编写过程中，感觉到和 C++的区别不是很大，在本项目中主要就是用到了 C#中的委托，所以上手也不是很困难。在 Unity 中，每一个 GameObject 就像是一个容器，可以为其添加不同的脚本、碰撞盒、刚体等等各种组件，这些组件共同决定了这个游戏对象的表现，刚开始不太熟悉，但逐渐地就熟悉了这一过程，并且找到了其中的乐趣。在设计过程中遇到了很多问题，但是通过和队友的交流，或是搜集资料，大部分都得以解决，提高了我解决问题的能力以及团队合作能力。

张亚宁：在本次大作业中，我和小组成员互相合作、合理分工完成了任务。主要完成了使用 EasyTouch 插件进行手势判断，以及控制人物动作的任务，这次大作业让我对 Unity3D 有了一定的了解；同时，在完成任务的过程中，发现问题，和组员及时沟通、通力协作解决问题的过程，也让我充分意识到良好的沟通和合作的重要性。

汤玉玲：本次任务让我认识到了用底层的 opengl 搭建场景和用引擎来搭建场景的不同之处。相比而言，用引擎搭建场景更加便捷，更适合游戏场景，而 opengl 有更大的灵活性，更适用于其它场景的搭建。另外，在学习探索的过程中我们小组互帮互助，体会到了小组协作的力量和重要性。

李欣怡：在本次大作业中，在组长的合理分工下，我与同学互相合作，最终完成了任务。这次任务的主要内容是基于 Unity 3D 的虚拟现实设计，我们使用 EasyTouch 插件和 Unity 以及 Visual Studio 设计了 3D 跑酷游戏。这次大作业让我对 Unity3D 和虚拟现实技术有了一定的了解，并且锻炼了我的小组合作能力，让我充分意识到良好的沟通和合作的重要性。在将来的学习实践中，我会将所学所感应用其中，努力进一步充实自己。

六、参考文献

[1] 董涛,张瑛. 基于 Unity3D 的第三视角射击类手游设计与实现[J]. 通讯世界,2019,26(11):98-99. DOI:10.3969/j.issn.1006-4222.2019.11.065.
[2] 张贝贝. 基于 Unity3D 的音乐游戏制作[J]. 现代信息科技,2021,5(8):112-114,118. DOI:10.19850/j.cnki.2096-4706.2021.08.031.
[3]李敏,张世遨,廖成林. 基于 Unity3D 的 VR 跑酷游戏设计与实现[J]. 信息与电脑,2020,32(12):119-122. DOI:10.3969/j.issn.1003-9767.2020.12.041.
[4] Unity 3D 技术手册：http://c.biancheng.net/unity3d/
[5] Unity 用户手册 2020.3(LTS)：
https://docs.unity.cn/cn/current/Manual/AssetWorkflow.html
[6]unity 插件 easytouch5 讲解：https://www.bilibili.com/video/av36809434/
[7]unity5.3+Easytouch4.3——EasyTouch 及摇杆控件介绍：
https://blog.csdn.net/xueyedie1234/article/details/51303494
[8]代码着色网站：https://tool.oschina.net/highlight

[9]EasyTouch 的使用教程：
https://blog.csdn.net/dingxiaowei2013/article/details/19967041
[10]Unity3D 中的线性插值函数 Lerp()解析：
https://www.cnblogs.com/unity3ds/p/5737152.html
[11]视频教程及原始资源来源：
https://edu.manew.com/goods/show/221?targetId=353&preview=0

附录：小组分工

汤玉玲：场景构建、循环，相机跟随、震动以及音效添加
张亚宁：使用 EasyTouch 插件对手势进行判断，人物的动作及效果控制
陈欣：碰撞检测，道具以及障碍物的设计以及相应脚本编写
李欣怡：UI 制作以及 UI 动画，游戏的暂停、恢复

对组员的评价：小组成员都很积极，大家都很好地完成了自己的任务，并互相讨论，帮助解决问题，这才使得我们的项目圆满完成。