《Wander In Color》

——PC/Android端2D横屏冒险闯关类

摘要

随着闯关类游戏的玩法与数目的不断增加，本文所设计的一款名为《Wander In Color》的动作冒险闯关类游戏，将从一个新的角度入手——游戏背景。该游戏相比于传统的闯关游戏来说，背景不再单一、单调，而变得色彩丰富、变化莫测，成为该游戏的核心玩法。游戏场景中的环境会随着背景颜色的变换而出现、消失，不仅使玩家眼前一亮，不会产生视觉疲劳，同时又能给玩家制造一种紧张感，不会感觉到无聊。游戏还融入了丰富的剧情、地图板块与玩法，相信你会沉浸其中，无法自拔。

关键词：动作，冒险，背景，颜色变换

Abstract

With the increasing numbers of recruit games, this action adventure game called “Wander In Color” which we designed, will start the game from a new perspective. Compared to traditional games, the background is no longer a single, monotonous element, but get colorful and polychrome which become the core gameplay of the game. The environment in the game will appear and disappear with the background color’s change. It not only make the game player themselves surprised, and avoid producing visual fatigue, but also make a sense of tension to the game player without feeling boring. The game also incorporates a rich story, map plates and gameplay. We believe that you will be addicted to this original game “Wonder In Color”.

Keywords: action, adventure, background, color switching

目录

[第一章 概述 1](#_Toc33978627)

[1.1项目名称 1](#_Toc33978628)

[1.2项目背景 1](#_Toc33978629)

[1.3工作内容和流程 1](#_Toc33978630)

[1.4核心概念 1](#_Toc33978631)

[1.5风险管理 2](#_Toc33978632)

[1.6预期目标 2](#_Toc33978633)

[第二章 游戏设计思路 2](#_Toc33978634)

[2.1游戏简介 2](#_Toc33978635)

[2.1.1游戏类型 2](#_Toc33978636)

[2.1.2游戏背景 2](#_Toc33978637)

[2.1.3核心玩法 3](#_Toc33978638)

[2.1.4游戏平台 3](#_Toc33978639)

[2.2游戏美术风格 3](#_Toc33978640)

[2.3游戏市场分析 4](#_Toc33978641)

[2.4游戏独特卖点 4](#_Toc33978642)

[第三章 游戏玩法设计 5](#_Toc33978643)

[3.1游戏机制 5](#_Toc33978644)

[3.1.1玩家能力 5](#_Toc33978645)

[3.1.2游戏胜负判定 5](#_Toc33978646)

[3.2游戏场景 5](#_Toc33978647)

[3.3游戏特殊系统 5](#_Toc33978648)

[3.4游戏多种结局 6](#_Toc33978649)

[第四章 游戏交互设计 7](#_Toc33978650)

[4.1游戏显示 7](#_Toc33978651)

[4.2操作控制 7](#_Toc33978652)

[4.3音乐音效 7](#_Toc33978653)

[第五章 游戏进程和关卡设计 7](#_Toc33978654)

[5.1游戏进程设计 7](#_Toc33978655)

[5.2 特殊元素设计 7](#_Toc33978656)

[5.3地域物品组成设计（可写入场景中的素材样式，如树桩） 8](#_Toc33978657)

[5.4目前设计的关卡内容 9](#_Toc33978658)

[5.4.1《瀑布》 9](#_Toc33978659)

[5.4.2《山洞》 9](#_Toc33978660)

[5.4.3 《溶洞》 9](#_Toc33978661)

[5.4.4《树林》 9](#_Toc33978662)

[第六章 程序设计 10](#_Toc33978663)

[6.1软件总体设计 10](#_Toc33978664)

[6.2主要脚本和算法 10](#_Toc33978665)

[6.2.1循环背景 10](#_Toc33978666)

[6.2.2背景检测控制平台的出现与消失 11](#_Toc33978667)

[6.2.3人物控制 12](#_Toc33978668)

[6.2.4自动移动平台 15](#_Toc33978669)

[6.2.5相机跟随及背景视差效果 16](#_Toc33978670)

[6.2.6人物拖尾特效 18](#_Toc33978671)

[6.2.7 UI 19](#_Toc33978672)

[6.2.8 人物跟随平台移动 20](#_Toc33978673)

[6.2.9 游戏管理器Game Manager 21](#_Toc33978674)

[第七章 游戏截图 23](#_Toc33978675)

# 第一章 概述

本部分介绍项目的整体概念，便于整体把握项目。适用于开发人员、游戏委托方阅读。

## 1.1项目名称

Android端2D横屏动作冒险类闯关游戏。

## 1.2项目背景

本项目开发基于 unity3D，辅助工具为 Adobe Photo shop,Visual studio 等，在目前闯关游戏单调的环境下，经过反复的讨论，决定制作一款以色彩变幻来改变障碍物的闯关游戏。小组成员认真查阅各种资料，解决了许多难以实现的问题，耗时较多，付出了许多心血。

## 1.3工作内容和流程

1、开会讨论游戏主题和玩法；

2、设计故事情节，使游戏更加连续、完整；

3、确定游戏风格和关卡种类；

4、上网搜集游戏素材、模型；

5、搭建游戏场景与角色动画制作；

6、解决核心玩法及其他的代码难题；

7、优化游戏素材与界面UI设计；

8、收集并处理音效，与游戏过程进行搭配；

9、编写游戏设计书。

## 1.4核心概念

在科技发达的未来，负责探索宇宙的飞船发现了一颗色彩斑斓的星球。星球上的环境都在随色彩的变化消失又出现，对于这奇怪的现象，地球派出一支队伍前去调查这究竟是什么原因。

## 1.5风险管理

1、代码遇到一些技术性的问题，难以实现或达到预期的效果；

2、美工与素材资源匮乏，界面不够精美；

3、地图种类多，关卡设计量大、难度大。

## 1.6预期目标

1、游戏测试阶段只有较少的BUG或没有BUG；

2、至少完成教程关和三类地图的关卡设计，玩法不同，给人丰富感；

3、界面整体美观，各素材与场景和谐。

# 第二章 游戏设计思路

## 2.1游戏简介

### 2.1.1游戏类型

该游戏为一款2D、第三人称视角的动作冒险闯关类游戏。

### 2.1.2游戏背景

2100年，在宇宙中进行探索的无人星际旅行号发现了一颗迷人的星球——它的表面色彩斑斓，变化万千，极其神秘，旅行号立即向地球传输回了坐标。全球的科学家和宇宙观测局都为之震惊，将它命名为彩虹星，集结了一支装备精良的调查小队乘坐超光速飞船，通过虫洞迅速前去研究调查，他们在那会发现什么样的秘密？又会遇到什么样的危险呢？



当飞船在进入星球大气层时，引擎却莫名奇妙的失火被毁，坠落过程中由于与大气层摩擦，飞船受到严重损伤，有些船舱的零件也在空中掉落。幸运的是，安全地迫降在了星球表面，1000名船员都未受到性命危险。

船员们下船后发现地面都很贫瘠，缺乏生命力，不知道究竟是什么原因，最令人感到神奇的是周围世界的颜色在不停变化，而周围的物体也相应的时隐时现。他们不得不小心谨慎地慢慢向前探索，去找回飞船掉落的零件，修复飞船，并带着所调查收集到的信息，最终返回地球。（收集的过程中可能会发现一个大秘密哦）

### 2.1.3核心玩法

玩家通过手机摇杆控制角色，在色彩变化的世界中，穿过重重障碍，到达该地图的终点，并且找到隐藏在地图中的飞船零件，最后修复飞船，并带着所调查收集到的信息，最终返回地球，而在这其中也有一个隐藏情节，为游戏增加了趣味性。根据经典2D闯关类游戏的思维，一步步完成主线的各个阶段。这种融入色彩和动作的闯关游戏会更加有趣。

### 2.1.4游戏平台

游戏平台为 PC端平台，也可在Android平台运行，开启EasyTouch插件即可。

## 2.2游戏美术风格

因颜色变化，该游戏的整体美术风格背景采用简约的剪影风格，清新脱俗又不失画面感。

## 2.3游戏市场分析

游戏的目标市场定位为大众市场。

因为该游戏操作简单，玩法独特，剧情丰富，所以会有许多对这种新鲜的游戏模式感兴趣的玩家来尝试。游戏过程中，下一秒前方可踩踏的平台可能就会消失，也有可能出现一个陷阱，这种未知增加了游戏的神秘感，还有丰富的关卡与地图模式，我相信大多数人都会喜欢，所以定位为大众市场。

## 2.4游戏独特卖点

1、颜色随时间不断变化的游戏背景以及场景中不断变化的物体和障碍物，这区别于目前市面上很多背景单调的冒险闯关游戏。

2、精巧有趣的多样化地图、关卡设计，引人入胜。

3、关卡以星球地图板块的形式，类似下面图片的效果，到达的显示，未到达的用阴影表现，给玩家一种神秘感。



4、简单易上手的操作，玩家更容易明白游戏原理。

5、关卡中有一些小怪物，可通过踩踏头部击杀。

6、游戏具有完整的主线，还有一些具有暗示性的线索，并且结局多样化，趣味性大大增加。

7、剧情采用静态画与文字结合的方式，让玩家体验到这不仅仅只是个闯关游戏。

8、收藏系统，每幅解锁后的静态画都存放在这里，玩家可回顾之前的插画剧情；收集到的隐藏物品也存放在这，给予玩家一种满足感，并且可看到还有多少未找到的隐藏物品。

# 第三章 游戏玩法设计

## 3.1游戏机制

### 3.1.1玩家能力

玩家在游戏中饰演一名银河探索小分队的一员，躲过各种陷阱，越过各种障碍物，收集飞船零件与星球上的神秘外星物品。

游戏中，玩家可以任意水平方向移动，调整视角，二次跳跃，触碰拾取物品。

### 3.1.2游戏胜负判定

游戏胜利条件很简单，穿越整个关卡，到达地图的另一侧，但是在这之间玩家需要注意场景中一些可放置隐藏物品的小细节的地方，还要躲过地刺、喷泉、岩浆、落石这些自然障碍物的攻击。

## 3.2游戏场景

游戏场景包含多种地域分格，可分为：枯树林、山谷、山洞、火山、雪山、瀑布、河流、沼泽。

在某些特殊的地域中，还会有一些你意想不到的效果。比如：火山，通关时间过久会被热死；在雪山，地面会变得滑，不能立刻停下来；在沼泽区域，移动速度会降低，跳跃高度也会降低；每个场景最后一关还会加设类似逃亡模式，比如火灾、岩浆、雪崩。

## 3.3游戏特殊系统

1、某些关卡中会放置飞船零件或一些隐藏外星物品，以可见或不可见的方式，通过触碰获得。就像是一个成就系统，但它不止是如此，它们还将引出这个星球背后的一个惊天秘密。

2、某些关卡中有时会有可以降低难度的一次性道具，辅助玩家过关。比如：时间变慢、背景色变白可落脚点均出现、喷气式背包到达无法抵达的地方。

3、在发现外星人之后解锁武器系统，玩家可获得几种不同攻击模式的武器，对付强大的外星BOSS。外星BOSS处在与自己颜色接近的背景下时，能力会大幅削弱。

## 3.4游戏多种结局

游戏的结局可分为三种：

1、触发条件：玩家在闯关过程中，显示目前的死亡次数，也就是船员死亡的数量。当死亡次数达到1000次（可根据游戏难易程度更改）时，也就代表着去探索的小队全军覆没。

触发事件：出现一幅荒芜的飞船残骸和死亡的船员代表游戏结束。

2、触发条件：玩家通关所有关卡（不包含隐藏关卡），但并未集齐之前关卡的隐藏外星物品。

触发事件：进入剧情，飞船上剩余的船员乘坐修复的飞船带着有关这个星球的资料返回地球。就在他们到达地球时，却发现地球也变得像先前自己前去探索调查的星球一样色彩斑斓，而且生物开始死亡，环境慢慢恶劣。

可是这并不是个美好的我们希望的结局，给予玩家第二次机会，回到选关界面，这时存在未收集到的隐藏物品的关卡会有提示，玩家可重新进入寻找并获得隐藏物品。之后进入结局3。

3、触发条件：玩家通关所有关卡（不包含隐藏关卡），并且集齐所有隐藏物品。

触发事件：船员发现这些不同寻常奇怪的物品，指向了一个外星文明的存在，并且种种迹象表明，这个星球原本并不是这个样子，而是由于某种人为因素造成的。他们通过这些物品给出的线索，在一个岩浆山洞里找到了这些外星人与他们的飞船，原来这些外星人就是把这个星球变成毫无生命力的罪魁祸首。在这看似使星球变得漂亮的表面下，其实是这些生物为其他物体赋予不属于它的颜色，并在这个过程中掠夺吸收属于这个物体的构成元素。

更令人感到无比害怕的是，他们的飞船屏幕上是地球的景象。他们下个目标就是地球，船员们决定阻止他们，与他们进行了一场激烈的战斗。开启隐藏关卡，解锁武器系统。(通关后)船员们击败了这些肆意给星球染色的外星人，并高兴的带着资料返回了地球。

# 第四章 游戏交互设计

## 4.1游戏显示

进入游戏后可看到开始游戏，退出游戏，成就系统，游戏设置，操作说明。

## 4.2操作控制

屏幕点击控制按钮控制移动、跳跃、二次跳跃。

## 4.3音乐音效

背景音乐来源于网络Energy Drink。

# 第五章 游戏进程和关卡设计

## 5.1游戏进程设计

玩家按照关卡顺序依次解锁进行游戏，第一关主角将在山川瀑布中漫步，比较简单，可以熟悉操作，了解游戏的特性；第二关主角会进入到山洞，体验不一样的地域风格，解锁更多新玩法；第三关主角将深入山洞深处，体验岩浆在脚下刺激的感觉，当完成本关即可开启下一关。当三关都已完成，玩家可以任意选择重新体验之前已完成的关卡。

## 5.2 特殊元素设计

|  |  |
| --- | --- |
| 元素 | 功能 |
| 彩色/黑色突刺 | 触碰到刺死 |
| 水平/竖直移动平台（可随地域变化外形） | 自动移动，可踩踏 |
| 易碎平台（悬空） | 踩踏一段时间后，会碎掉 |
| 暗坑（地面上） | 踩上去后过一段时间会崩塌 |
| 轻质平台 | 踩踏时会慢慢下坠 |
| 彩色藤蔓 | 悬挂在空中，可以拽住 |
| 爬虫 | 彩色藤蔓上的虫子，可踩死 |
| 荆棘（树林） | 同地刺 |
| 落石 | 上方下落，会被砸死 |
| 钟乳石（山洞） | 上方下落，会被砸死 |
| 小叶子 | 可弹跳 |
| 滚石（斜面有点难以实现） | 滚动，会被压死 |
| 石块 | 可推动 |
| 水柱 | 冲击到，会被淹死 |
| 岩浆柱 | 冲击到，会被烧死 |
| 上涨的岩浆 | 触碰到烧死 |
| 下落的一大堆雪块 | 在雪山中，间隔下落，由于地面滑，碰到会被砸死 |
| 彩色传送门 | 传送到颜色相同的传送门位置 |
| 变色机关（背景色固定的情形下） | 将整个背景变为该色 |
| 彩色吐刺花 | 向前方射出彩色尖刺，无法消灭，触碰死亡 |
| 爬行怪 | 触碰死亡，可踩击头部致死 |
| 彩色流动船只 | 可踩踏 |
| 潜水服/氧气瓶（暂不考虑） | 加入水下关卡，可水下呼吸 |
| 漩涡（暂不考虑） | 卷入死亡 |
| 逃亡 | 如森林火灾，随机障碍物着火，只能从下方通过，背景上多做几种动物 一起逃亡 换BGM和融入脚步声， 玩家被火烧，加速限制时间，可以进入水潭扑灭，同时减速加大游戏难度。  如山洞崩塌，画面震动。  山谷滚石追赶像滑雪大冒险那样。 |
| 隐藏关 | 如超级玛丽树洞下去触发，比如奖励关，或者隐藏剧情 |
| 地面突然裂开(一次性) | 在玩家前方裂开，需跳过去 |
| 魔幻菇 | 吃了后，镜头上下颠倒。一段时间后 恢复 |

## 5.3地域物品组成设计（可写入场景中的素材样式，如树桩）

通用（几乎所有地域风格都可用到）

元素：突刺、轻质平台、易碎平台、爬行怪、彩色传送门、变色机关

5.3.1树林

场景物体：树桩、朽木

元素：小叶子、吐刺花、藤蔓、爬虫、荆棘、魔幻菇

5.3.2瀑布

场景物体：

元素：小叶子

5.3.3山谷

场景物体：

元素：滚石、落石

5.3.4山洞

场景物体：

元素：钟乳石

5.3.5火山

场景物体：

元素：岩浆柱、岩浆、地面突然裂开(一次性)

5.3.6熔洞

场景物体：

元素：上涨的岩浆、岩浆柱

5.3.7雪山

场景物体：冰面

元素：下落的一大堆雪块

5.3.8河流

场景物体：

元素：水柱、彩色流动船只、小叶子

5.3.9沼泽

场景物体：

元素：

5.3.10枯树林

场景物体：树干、

元素：小叶子、魔幻菇、荆棘

5.3.11沙漠

场景物体：

元素：

5.3.12海底

场景物体：珊瑚、暗礁

元素：氧气瓶、漩涡

## 5.4目前设计的关卡内容

### 5.4.1《瀑布》

元素：突刺、可水平自动移动的平台，瀑布

第一关难度简单，包括一些跨越山谷、躲避尖刺、攀爬瀑布系列刺激体验。

### 5.4.2《山洞》

元素：小叶子、突刺、石块、爬行怪

第二关玩法多样新奇，包括越过沟壑、体验弹跳草、遭遇爬行怪。

### 5.4.3 《溶洞》

元素：轻质平台，落石，岩浆

第三关惊险刺激，脚下就是岩浆，头顶还有落石，踩踏的平台还在慢慢下坠。

### 5.4.4《树林》

元素：突刺、爬行怪、吐刺花、藤蔓

# 第六章 程序设计

## 6.1软件总体设计

本游戏为横版闯关冒险游戏，主要代码有：

具有本游戏特色的背景颜色变化及颜色平台消失脚本，人物移动及跳跃脚本，镜头跟随脚本，与物体交互脚本等。

## 6.2主要脚本和算法

### 6.2.1循环背景

**表3**

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class BGHorizontal : MonoBehaviour {  //public Color xcolor;//自定义一个颜色  private float wait=0.5f;  private float rush = 0.5f;  public float HorzDis;  void Start () {  StartCoroutine(move());//启动辅助函数  }  IEnumerator move()//协助函数  {  while (true)  {  GetComponent<Rigidbody2D>().velocity = new Vector2(7, 0);  yield return new WaitForSeconds(wait);  GetComponent<Rigidbody2D>().velocity = new Vector2(-4, 0);  yield return new WaitForSeconds(wait);  GetComponent<Rigidbody2D>().velocity = new Vector2(40, 0);  yield return new WaitForSeconds(rush);  GetComponent<Rigidbody2D>().velocity = new Vector2(0, 0);  }  }  void Update()  {    }  void OnTriggerEnter2D(Collider2D collider)  {  if (collider.tag == "HorizontalTrigger")  {  this.transform.Translate(new Vector3(HorzDis, 0, 0));  }  }  } |

### 6.2.2背景检测控制平台的出现与消失

**表4**

|  |  |
| --- | --- |
| using UnityEngine;  using System.Collections;  public class ColorJudge : MonoBehaviour  {  public GameObject Block;  public GameObject BG;  public Color Xcolor;  // Use this for initialization  void Start()  {  }  void Awake()  {  Block.GetComponent<Renderer>().material.color=Xcolor;  BG.GetComponent<Renderer> ().material.color = Xcolor;  }  // Update is called once per frame  void Update()  {    }  //颜色相同时一旦触碰，使平台消失  void OnTriggerEnter2D(Collider2D collider)  {    if (collider.tag == "BackColor")  {  if (collider.GetComponent<Renderer> ().material.color == Block.GetComponent<Renderer> ().material.color) {  Block.SetActive (false);  //Block.GetComponent<Collider2D> ().enabled = false;  }  }  }  //颜色不同时，平台出现  void OnTriggerExit2D(Collider2D collider)  {  if (collider.tag == "BackColor") {  if (collider.GetComponent<Renderer> ().material.color == Block.GetComponent<Renderer> ().material.color) {  Block.SetActive (true);  }  }  }  } |  |

### 6.2.3人物控制

**表5**

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class PlayerControl : MonoBehaviour  {  private bool IsGrounded;    //动画控制  private Animator AnimatorController;  //音效  public AudioClip jump;  public AudioClip dead;  //移动速度与跳跃力度  public float jumpSpeed;  public float crashSpeed;  //粒子系统  public ParticleSystem JumpParticles\_Floor;  public ParticleSystem JumpParticles\_doubleJump;  public ParticleSystem Particles\_DeathBoom;  void Start()  {  AnimatorController = GetComponent<Animator>();  //move.SetBool("walk", false);  }  // Update is called once per frame  void Update()  {  if (GameManager.getInstance().GAMESTATE == GameManager.getInstance().PLAYING)  {//二连跳判断  if (GameManager.getInstance().jumpFlag && GameManager.getInstance().jumptime < 2)  {  if (Input.GetKeyDown(KeyCode.Space))  {  GameManager.getInstance().jumptime++;  AudioSource.PlayClipAtPoint(jump,this.transform.position);  if (GameManager.getInstance().jumptime == 1)  JumpParticles\_Floor.Emit(20);//起跳粒子  if (GameManager.getInstance().jumptime == 2)  {  JumpParticles\_doubleJump.Emit(10);  GameManager.getInstance().jumpFlag = false;  }  GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D>().velocity.x, jumpSpeed);  }  }  //左右移动  if (Input.GetKey(KeyCode.RightArrow))  {  //GetComponent<Rigidbody2D> ().AddForce (Vector2.right \* 2000f \* Time.deltaTime);  GetComponent<Rigidbody2D>().velocity = new Vector2(crashSpeed, GetComponent<Rigidbody2D>().velocity.y);  }  if (Input.GetKeyUp(KeyCode.RightArrow))  {  GetComponent<Rigidbody2D>().velocity = Vector2.zero;  }  if (Input.GetKey(KeyCode.LeftArrow))  {  GetComponent<Rigidbody2D>().velocity = new Vector2(-crashSpeed, GetComponent<Rigidbody2D>().velocity.y);  }  if (Input.GetKeyUp(KeyCode.LeftArrow))  {  GetComponent<Rigidbody2D>().velocity = Vector2.zero;  }  //转向  if (Input.GetKey(KeyCode.LeftArrow))  {  transform.rotation = Quaternion.AngleAxis(180, Vector3.up);  if (Input.GetKey(KeyCode.RightArrow))  {  transform.rotation = Quaternion.AngleAxis(0, Vector3.up);  }  }  if (Input.GetKey(KeyCode.RightArrow))  {  transform.rotation = Quaternion.AngleAxis(0, Vector3.up);  if (Input.GetKey(KeyCode.LeftArrow))  {  transform.rotation = Quaternion.AngleAxis(180, Vector3.up);  }  }  }  }  void FixedUpdate()  {  AnimatorController.SetFloat("HorizontalSpeed", this.GetComponent<Rigidbody2D>().velocity.x \* this.GetComponent<Rigidbody2D>().velocity.x);  AnimatorController.SetFloat("VerticalSpeed", this.GetComponent<Rigidbody2D>().velocity.y);  }  void OnCollisionStay2D(Collision2D other)  {  if (other.gameObject.tag == "Dead")  {  this.GetComponent<Rigidbody2D>().velocity = Vector2.zero;  if (GameManager.getInstance().GAMESTATE == GameManager.getInstance().PLAYING)  AudioSource.PlayClipAtPoint(dead, this.transform.position);  GameManager.getInstance().GAMESTATE = GameManager.getInstance().GAMEOVER;  }  }  void OnTriggerExit2D(Collider2D collider)  {  if (collider.tag == "Ground" && IsGrounded == true) {  IsGrounded = false;  }  }  } |

### 6.2.4自动移动平台

**表6**

|  |  |
| --- | --- |
| using UnityEngine;  using System.Collections;  public class AutoPlatformMove : MonoBehaviour {  public float speed;  //记录平台的位置  public Transform End;  //记录关键位置  private Vector3 StartPosition;  private Vector3 EndPosition;  private bool OnTheMove;  void Start()  {  //存储起点和终点的位置  StartPosition = this.transform.position;  EndPosition = End.position;  }  void FixedUpdate()  {  float step = speed \* Time.deltaTime;  if (OnTheMove == false)  {  this.transform.position = Vector3.MoveTowards(this.transform.position, EndPosition, step);  }  else  {  this.transform.position = Vector3.MoveTowards(this.transform.position, StartPosition, step);  }  //当到达终点时转向  if (this.transform.position.x == EndPosition.x && this.transform.position.y == EndPosition.y && OnTheMove == false)  {  OnTheMove = true;  }  else if (this.transform.position.x == StartPosition.x && this.transform.position.y == StartPosition.y && OnTheMove == true)  {  OnTheMove = false;  }  }  } |  |

### 6.2.5相机跟随及背景视差效果

**表7**

|  |  |
| --- | --- |
| using UnityEngine;  using System.Collections;  public class CameraFollow : MonoBehaviour  {  public GameObject FollowTargetOBJ;  public GameObject[] Background;  public float FollowSpeed;  // Use this for initialization  void Start()  {  }  // Update is called once per frame  void Update()  {  }  void FixedUpdate()  {  //镜头跟随小人  Vector3 PositionBefore = this.transform.position;  Vector3 NewPosition = Vector3.Lerp(this.transform.position, FollowTargetOBJ.transform.position, FollowSpeed \* Time.deltaTime);//Lerp是一个物体跟随另一个物体的写法  //通过相机的位置判断是否到达瀑布  if (this.transform.position.x <= 85f)  {  if (this.transform.position.y < 30)  {  //在水平地面上移动时，镜头水平移动  this.transform.position = new Vector3(NewPosition.x, 1.5f, this.transform.position.z);  }  //到达通关地面  else  {  this.transform.position = new Vector3(85, NewPosition.y, this.transform.position.z);  }  //背景视差效果  Vector3 CameraMovementAmount = PositionBefore - this.transform.position;  Background[0].transform.Translate(-CameraMovementAmount \* 0.8f);  Background[1].transform.Translate(-CameraMovementAmount \* 0.7f);  Background[2].transform.Translate(-CameraMovementAmount \* 0.6f);  Background[3].transform.Translate(-CameraMovementAmount \* 0.5f);  Background[4].transform.Translate(-CameraMovementAmount \* 0.4f);  Background[5].transform.Translate(-CameraMovementAmount \* 0.3f);  Background[6].transform.Translate(-CameraMovementAmount \* 0.2f);  }  else  {  if (NewPosition.y >= 1.6f)  {  //当到达瀑布跳跃时镜头可以上下左右移动  this.transform.position = new Vector3(NewPosition.x, NewPosition.y, this.transform.position.z);  }  else  this.transform.position = new Vector3(NewPosition.x, 1.5f, this.transform.position.z);  }  }  } |  |

### 6.2.6人物拖尾特效

**表8**

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class HunterState : MonoBehaviour  {  public GameObject destinationTarget;//追击目标  public NavMeshAgent agent;//追击者  private Vector3 o\_position;//上帧坐标  private Vector3 n\_position;//当前坐标  private Vector3 c;//上帧与当前坐标差  private Vector3 c\_target;//当前坐标与目标坐标差  private float dis;//当前坐标与目标坐标差的向量长度  private bool flag;  private bool trigger;//人物进入到一定范围内才可以开始被追击,trigger为true时进行追击  private float speed = 5f;//追击速度  void Start()  {  trigger = false;  }  void OnTriggerEnter(Collider other)  {  if (other.tag == "Player")  {  trigger = true;  }  }  void OnTriggerExist(Collider other)  {  if (other.tag == "Player")  {  trigger = false;  }  }  void Update()  {  if (trigger)  {  agent.destination = destinationTarget.transform.position;//target目标设为追击目标  c\_target = agent.destination - transform.position;  n\_position = transform.position;  if (o\_position == Vector3.zero)//第一帧时候仅记录旧坐标  {  }  else  {  c = n\_position - o\_position;  }  dis = Time.deltaTime \* speed;  o\_position = n\_position;  }  else  {  agent.destination = transform.position;//没有不追击时候令其追击自己坐标，即不移动  }  }  } |

### 6.2.7 UI

**表9**

|  |
| --- |
| using UnityEngine;  using System.Collections;  using UnityEngine.SceneManagement;  using UnityEngine.UI;  public class UI : MonoBehaviour {  public GameObject panel;//结束界面  public GameObject teach;//教程界面  void Awake()  {    }  void Start () {    }    // Update is called once per frame  void Update () {  if (GameManager.getInstance().IsFirst == false)  {  teach.SetActive(false);  }  if (GameManager.getInstance().GAMESTATE == GameManager.getInstance().GAMEOVER)  {  panel.SetActive(true);  }  }  public void Restart()//重新开始  {  panel.SetActive(false);  GameManager.getInstance().GAMESTATE = GameManager.getInstance().PLAYING;  SceneManager.LoadScene("Game");  //复活到初始位置  }  public void Exit()  {  Application.Quit();//结束游戏  }  } |

### 6.2.8 人物跟随平台移动

**表10**

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class PlayerFollowMoveBlock : MonoBehaviour {  // Use this for initialization  void Start () {    }    // Update is called once per frame  void Update () {    }  void OnCollisionStay2D(Collision2D other)  {  other.gameObject.transform.parent = transform;  }  void OnCollisionExit2D(Collision2D other)  {  other.gameObject.transform.parent = null;  }  } |

### 6.2.9 游戏管理器Game Manager

**表11**

|  |
| --- |
| using UnityEngine;  using System.Collections;  public class GameManager : MonoBehaviour {  public GameObject player;  public GameObject BGM;  public bool IsFirst = true;  //检测游戏状态  public int GAMESTATE;  public int MENU = 0;  public int PAUSE = 1;  public int PLAYING = 2;  public int PREPARING = 3;  public int GAMEOVER = 4;  float waitTime = 4f;  //跳跃次数判断  public bool jumpFlag;  public int jumptime;  //Singleton 单例  private static GameManager instance;  private static GameObject container;  public static GameManager getInstance()  {  if (!instance)  {  container = new GameObject();  container.name = "Logger";  instance = container.AddComponent(typeof(GameManager)) as GameManager;  }  return instance;  }  void Awake()  {  GAMESTATE = PREPARING;  jumpFlag = true;  jumptime = 0;  }  // Use this for initialization  void Start () {  DontDestroyOnLoad(gameObject);//每次加载场景时，不破坏GameManager  DontDestroyOnLoad(BGM);// 每次加载场景时，不破坏音乐  StartCoroutine(wait());  }  IEnumerator wait()//协助函数  {  yield return new WaitForSeconds(waitTime);  IsFirst = false;  GAMESTATE = PLAYING;  }  // Update is called once per frame  void Update () {  }  } |

# 第七章 游戏截图

