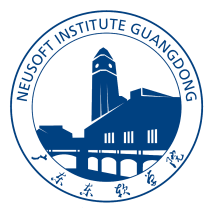
****本科毕业设计（论文）**

**基于AR增强现实技术的游戏设计与开发**

**The Design and Implementation of Game Based on AR (Augmented Reality Technology)**

|  |  |
| --- | --- |
| **院 （系）** | 计算机系 |
| **专 业** | 软件工程 |
| **班 级** | 六班 |
| **学 号** | 14210120610 |
| **学生姓名** | 陈思亮 |
| **指导教师** | 向燕飞 |
| **提交日期** | 2018年4月25日 |

**内容摘要**

增强现实是通过平台系统采集数据（一般为图片）并将其识别出来，然后提供的已准备好的虚拟信息，如三维的模型显示在用户的设备上，类似投影，但又只能通过设备才能看到的技术，通过这种途径实现了对现实视觉效果的增强。在这篇论文中，会通过增强现实技术与Unity引擎开发的游戏的结合，同时利用了移动终端方便便捷的视觉效果，实现了一款简单的动作游戏，使原本较为简单的游戏变得乐趣无穷，拥有新鲜的体验。同时无需购买任务设备，只要拥有一台手机就能体验到增强现实技术的魅力。

本论文通过介绍高通的Vuforia SDK与Unity游戏引擎，并通过两者的结合来实现一款增强现实的动作类手机游戏，因此本款游戏不只是推广增强现实技术在游戏中的应用，也具有一定的经济价值。

**关键词：**增强现实 AR技术 动作游戏

**Abstract**

AR,augmented reality,is a new technology developed on the basis of virtual reality (VR).Augmented reality is to collect data (generally pictures) and identify it through a platform system,and then provide prepared virtual information,such as a 3D model that is displayed on the user's device,similar to the projection,but can only be seen through the device,through which the visual effect is realized. Enhance.In this paper,through the combination of augmented reality technology and the game developed by the Unity engine,and using the convenient and convenient visual effects of the mobile terminal,a simple action game has been realized,making the original simpler games fun and fresh.At the same time,you can experience the charm of augmented reality technology without having to purchase task equipment.

The game design and implementation process,not only combine the latest AR technology,but also add some elements of the game,so this game not only promotes the application of the AR technology game industry,but also has a certain social and economic value.

**Key words：**augmented reality AR technology action game

**目录**

**[第一章 Unity游戏引擎与Vuforia SDK介绍](#_Toc26361)** [1](#_Toc26361)

**[1.1 Unity游戏引擎](#_Toc15580)** [1](#_Toc15580)

[1.1.1 Unity3D重要概念 1](#_Toc12959)

[1.1.2 Unity常用生命周期的函数 1](#_Toc22335)

[1.1.3 Unity的优势 2](#_Toc21271)

**[1.2 3Ds Max软件](#_Toc28695)** [2](#_Toc28695)

**[1.3 C#开发语言](#_Toc15979)** [2](#_Toc15979)

**[1.4 Vuforia SDK介绍](#_Toc7840)** [2](#_Toc7840)

[1.4.1 Vuforia概述 3](#_Toc10975)

[1.4.2 Vuforia常用组件简单介绍 3](#_Toc5178)

**[第二章 AR游戏概述与设计](#_Toc12241)** [4](#_Toc12241)

**[2.1 AR显示原理及分类](#_Toc4163)** [4](#_Toc4163)

**[2.2 AR游戏类型分析](#_Toc26301)** [4](#_Toc26301)

**[2.3 游戏框架结构](#_Toc1935)** [4](#_Toc1935)

**[2.4 游戏功能的需求](#_Toc21826)** [5](#_Toc21826)

**[2.5 外部模型的需求](#_Toc17632)** [5](#_Toc17632)

**[第三章 UI界面](#_Toc28754)** [6](#_Toc28754)

**[3.1 游戏进入界面设计](#_Toc12743)** [6](#_Toc12743)

[3.1.1 设置界面设计 7](#_Toc18941)

[3.1.2 关于游戏界面设计 7](#_Toc13261)

**[3.2 AR识别界面设计](#_Toc18036)** [8](#_Toc18036)

**[3.3 游戏战斗界面设计](#_Toc10219)** [9](#_Toc10219)

**[第四章 游戏本地化系统实现](#_Toc23091)** [11](#_Toc23091)

**[4.1本地化系统的设计思路](#_Toc5767)** [11](#_Toc5767)

**[4.2 配置中英文语言文本](#_Toc8822)** [11](#_Toc8822)

**[4.3 本地化系统的设计实现](#_Toc19605)** [12](#_Toc19605)

**[第五章 游戏增强现实的实现](#_Toc29288)** [15](#_Toc29288)

**[5.1 游戏常用工具的实现](#_Toc10895)** [15](#_Toc10895)

[5.1.1 有限状态机的实现 15](#_Toc681)

[5.1.2 游戏孵化器的实现 15](#_Toc16479)

[5.1.3 游戏消息系统的实现 18](#_Toc7708)

[5.1.4 UI管理系统的实现 19](#_Toc17783)

[5.1.5 游戏保存管理系统的实现 19](#_Toc15906)

**[5.2 游戏摇杆的实现](#_Toc13980)** [19](#_Toc13980)

**[5.3 小地图管理的实现](#_Toc22955)** [21](#_Toc22955)

**[5.4 游戏功能实现](#_Toc28476)** [29](#_Toc28476)

[5.4.1 摄像机控制 29](#_Toc1470)

[5.4.2 游戏的分层管理 29](#_Toc582)

[5.4.3 AR图片识别的实现 30](#_Toc15218)

[5.4.4 玩家与怪物的对战逻辑实现 32](#_Toc30422)

**[第六章 游戏测试](#_Toc30308)** [37](#_Toc30308)

**[6.1 功能测试](#_Toc28472)** [37](#_Toc28472)

**[6.2 内存测试](#_Toc14387)** [37](#_Toc14387)

**[第七章 总结与展望](#_Toc13582)** [38](#_Toc13582)

**[参考文献](#_Toc8330)** [3](#_Toc8330)9

**[致谢](#_Toc15261)** [4](#_Toc15261)0

1. **Unity游戏引擎与Vuforia SDK介绍**

## 1.1 Unity游戏引擎

在本章节中，我会简单地介绍Unity游戏引擎的基本概念和常用的API方法和为什么选择Unity游戏引擎来开发AR游戏的原因，也会说明和简单介绍有关Vuforia SDK的基本概念

### 1.1.1 Unity3D重要概念

Unity游戏引擎是一个能够方便发布各种平台，特别用于开发手机游戏，在安卓和苹果的移动平台上都能够很好的应用。同时高通的Vuforia的SDK也支持了Unity开发移动端平台的AR应用，这个将在后面仔细介绍。

Unity游戏引擎的开发工具可以运行在Mac0SX和Windows下，能够发布应用至多个平台，如Windows，Mac，Windows phone8，Android和iPhone，同时它也能利用unity web player插件发布网页型应用，Unity特别更新webgl，支持移动端和PC端的浏览器浏览内容的时候更加流畅。

### 1.1.2 Unity常用生命周期的函数

Unity游戏引擎不像常规的程序直接在Main函数入口运行，Unity在内部实现了自己的生命周期事件，通过对这些生命周期的事件进行写入，Unity内部就会不断地迭代这些生命周期函数。

下面按照脚本的执行顺序介绍一些游戏中常用的Unity事件函数。

Awake：当游戏对象被初始化的时候调用，无论该对象是否已被激活。

Start：在Awake事件后调用，但只有被激活的时候才能够执行。

Update：游戏中的帧事件，因为游戏大部分都是按帧率来执行逻辑的。

FixedUpdate：游戏的固定帧事件，基本同Update，但该事件可由开发者去控制执行频率。

OnEnable：在Start事件后调用，只有被激活的时候才会执行。

OnDisable：当游戏对象被禁止激活的时候会调用。

OnDestroy：当游戏对象被销毁的时候执行。

Unity3D的脚本基本上都会继承自于MonoBehavior基类。一般不继承于MonoBehavior的类通常用来写一些工具类，一般来说，它在整个程序运行的过程中都是存在的，除非自己手动GC释放内存。

### 1.1.3 Unity的优势

综合编辑：Unity具备视觉化的编辑模式，具有层级式的开发环境，动态游戏预览特性和详细的属性编辑器。

图形引擎：使用的是OpenGL，Direct3D和Unity自身集成的APIs，因此可以写Unity自身的ShaderLab语言。

粒子系统：Unity本身自带了粒子系统，同时也可以在Unity商店中购买他人的粒子系统工具。

物理特效：Unity引擎基本上提供了很完善的物理引擎系统，也提供了许多便捷的API来供开发者使用。

真实的光影效果：提供了具有柔和阴影与lightmaps高度完善的光影渲染系统。

Unity资源服务器：支持多人并行开发，提供开发效率。

## 1.2 3Ds Max软件

3Dsmax 是游戏开发中最常用建模工具之一，是开发工业级三维作品的大多数人的选择。

这里只是简单介绍，但本游戏中的模型不使用3Ds Max来建模，因为会花费大量时间和精力，本项目会直接采用Unity商店中已建好的模型。但要导入外部的模型，为了避免出错，还是要下载该软件，有些模型格式要使用到该软件内部的执行代码。

## 1.3 C#开发语言

C#是微软公司发布的一种运行于.NET Framework之上，面向对象的高级程序设计语言。C#与Java类似，而语言只是工具。但这里不要忘记Unity是跨平台的，而C#并不是一种跨平台的语言。

但由于Mono的的重新实现，使得Unity能够使用C#来开发。因为如此，我们可以在Unity上写C#的代码。而C#的优点，语法明了，类库使用方便。是我们在Unity开发中首先考虑的语言。

## 1.4 Vuforia SDK介绍

随着对增强现实技术研究的深入和发展以及智能硬件设备的性能也在日渐提高，市场上出现了许多对增强现实技术的支持，特别是由高通开发的Vuforia SDK。因此下面着重介绍一下Vuforia SDK。

### 1.4.1 Vuforia概述

Vuforia SDK是高通开发的支持增强现实技术的工具包，分别有IOS平台，Android平台和Unity平台三个版本。在这里，理所当然是选择Unity版本的。通过在Unity编辑器中导入Vuforia SDK可以方便快捷地开发AR游戏。Vuforia SDK是通过设备摄像头对现实中的图片进行分析该图片的特征，然后识别并将虚拟的建模对象以该图片在坐标轴显示出来。Vuforia SDK与其他的AR SDK技术相比，在AR增强现实技术界之所以有如此魅力，主要体现在以下几个方面：

1. 识别物品更加迅速，准确和稳定。
2. 支持云识别技术，同时能够并行识别和处理百万数量级的识别数据。
3. 用户可以自定义识别目标。
4. 能够同时跟踪和处理多个标识物。

### 1.4.2 Vuforia常用组件简单介绍

Vuforia SDK提供了许多快捷的开发组件，包括Camera，Image Converter，Video Background Render，Tracker等。在这里主要简单介绍Camera，Image Convert和Tracker组件。

Camera：这是属于Vuforia SDK在Unity中封装了一个摄像机组件，主要用来识别Image Convert所带来的图片并反馈给用户。

Image Convert：其作用在于被Camera来识别，并将三维物体以此为坐标系，显示在此之上。

Tracker：能用一种可自定义变更的算法跟踪在摄像头得到的图片中的目标，并把对象用stateobject方式存储。跟踪器可同时追踪多个数据集，但同一时间只能有一个被激活。

# 第二章 AR游戏概述与设计

## 2.1 AR显示原理及分类

AR，即增强现实，它是在虚拟现实（VR）的基础上发展起来的一种新技术。增强现实是通过平台系统采集数据（一般为图片）并将其识别出来，然后提供的已准备好的虚拟信息，如三维的模型显示在用户的设备上，类似投影，但又只能通过设备才能看到的技术，通过这种途径实现了对现实视觉效果的增强。在这篇论文中，会通过增强现实技术与Unity引擎开发的游戏的结合，同时利用了移动终端方便便捷的视觉效果，实现了一款简单的动作游戏，使原本较为简单的游戏变得乐趣无穷，拥有新鲜的体验。同时无需购买任务设备，只要拥有一台手机就能体验到增强现实技术的魅力。

AR 系统通常会采用三个主要技术特点：

1. 是否联合现实视觉图像和虚拟图像。
2. 图形是否在虚拟的三维定位。
3. 系统是实时运行的。

## 2.2 AR游戏类型分析

目前在市场基于增强现实技术的游戏并不算多，也不够被大众所知道所了解。但仍然有许多游戏开发者投入到AR技术的潮流中，比较有特征的游戏如下：

养成类：代表作《精灵宝可梦Go》。

寻宝类：Ingress。

图片3D类：针对幼儿的幼教卡AR游戏

不过基于 AR 增强技术的动作格斗类游戏比较少见，因此本游戏设计目标为动作格斗类游戏。

## 2.3 游戏框架结构

在游戏设计界面上有游戏开始界面，AR识别主界面和战斗界面等。

当玩家运行游戏时，会进入开始界面，这此界面中可看到一系列的菜单选项等。

当点击开始游戏时，会进入AR主界面。在这里，手机的摄像头会自动识别当前画面的图片，分别会作出三种判断，分别为：可识别，不可识别和识别失败。根据这三种判断显示不同的提示信息给玩家进行操作改正（即换更有特征的图片来进行识别）

识别成功后会正式进入游戏，主要有小地图，分数面板，血量条，摇杆等界面。后面会进行详细介绍。

## 2.4 游戏功能的需求

往往一款游戏的功能需求可大可小，在这里理所当然是属于小游戏的类型。但还是要针对增强现实的技术的特点去设计能体现AR的功能需求，同时也去尽力完成一些在游戏中常看见的功能需求。因此，为了让这个AR游戏具有一定的可玩性和可观赏性性，对游戏的功能需求主要采用了以下几点：

1. 人物模型要精美；
2. 玩法够简单和有正向激励；
3. AR模式要通用，任意图片都可识别；
4. 游戏要有本地化，国际化。

## 2.5 外部模型的需求

在Unity内置商店中已经有许多模型的购买，有免费的也有收费的。在这里为了避免影响开发进度，这里直接在Unity中购买模式来进行开发。

人物模型：选择了简单的4种颜色的中世纪士兵。

环境模型：AR摄像机是显示现实坏境，因此这里没有必要使用环境模型。

# 第三章 UI界面

## 3.1 游戏进入界面设计

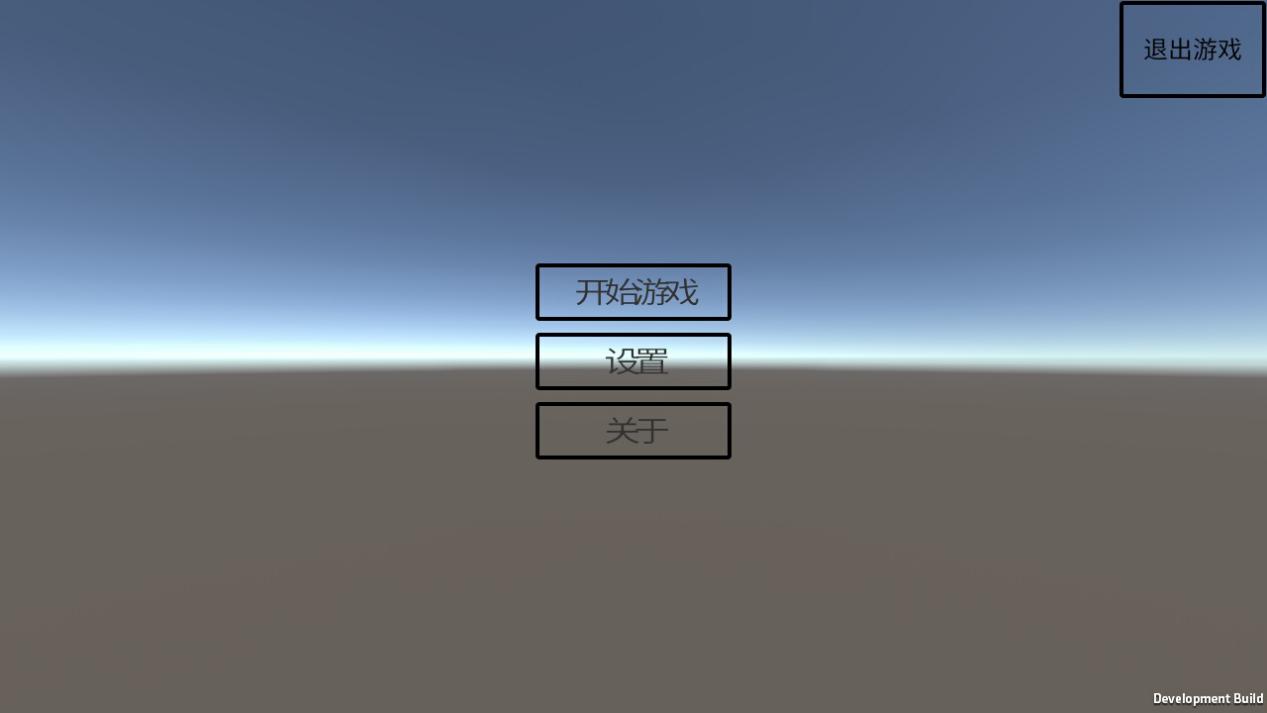
当玩家开始运行游戏的时候，首先会进入开始界面。开始界面如图3.1所示，主要有四个按钮，分别为开始游戏，设置，关于和退出游戏。

开始游戏：直接进入AR游戏的识别界面，如果识别成功会加载场景与模型，后面会有详细的介绍。

设置：分别可设置游戏的音量以及游戏的语言，语言分别有中英文。

关于：这是关于本人开发该游戏的当时的一些信息。

退出游戏：将会自动保存游戏的设置并退出游戏。



3.1 开始界面



3.2 设置界面



3.3 关于界面

### 3.1.1 设置界面设计

设置界面由图3.2可知，主要有音量设置和语言设置，本游戏采用了本地方系统，可大部分应付国内外的语言问题，增加游戏的适用人群和使用范围。有关本地化系统的设计在第4章有详细介绍。

### 3.1.2 关于游戏界面设计

关于界面的设计只是为了简单地向玩家展示游戏开发的相关信息，在此不作多介绍。

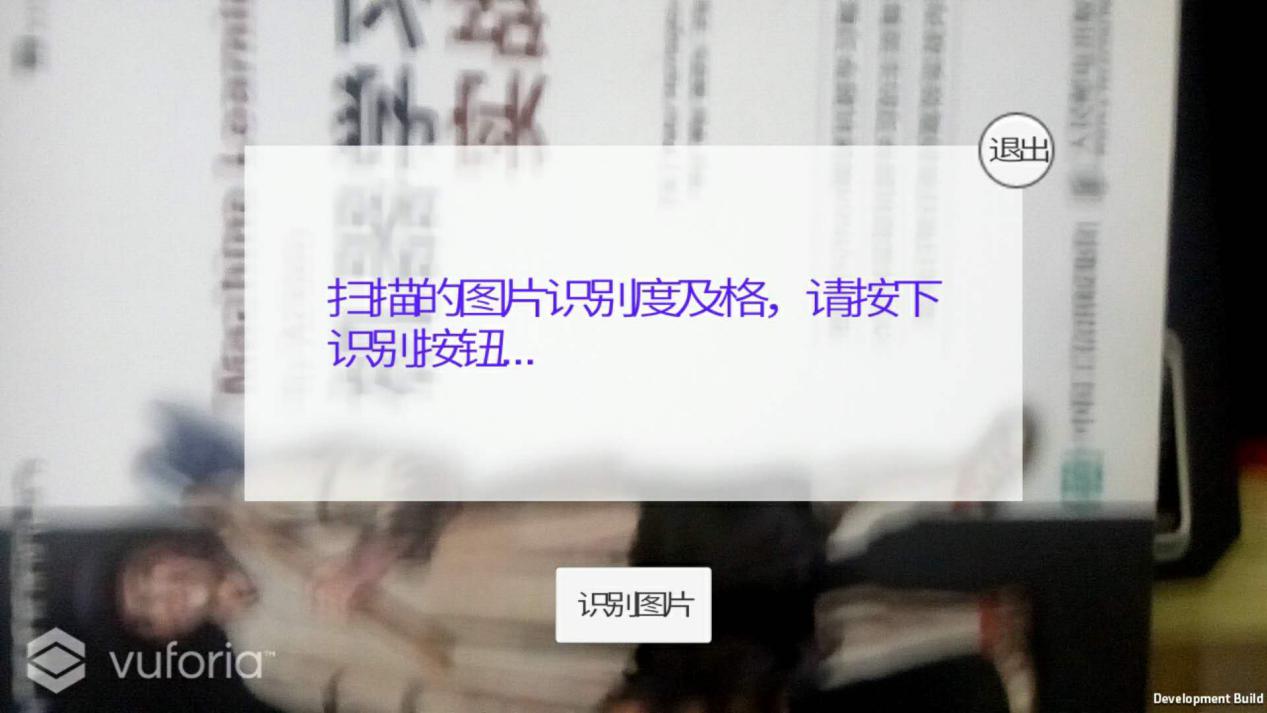
## 3.2 AR识别界面设计

AR识别界面的设计主要有三种：

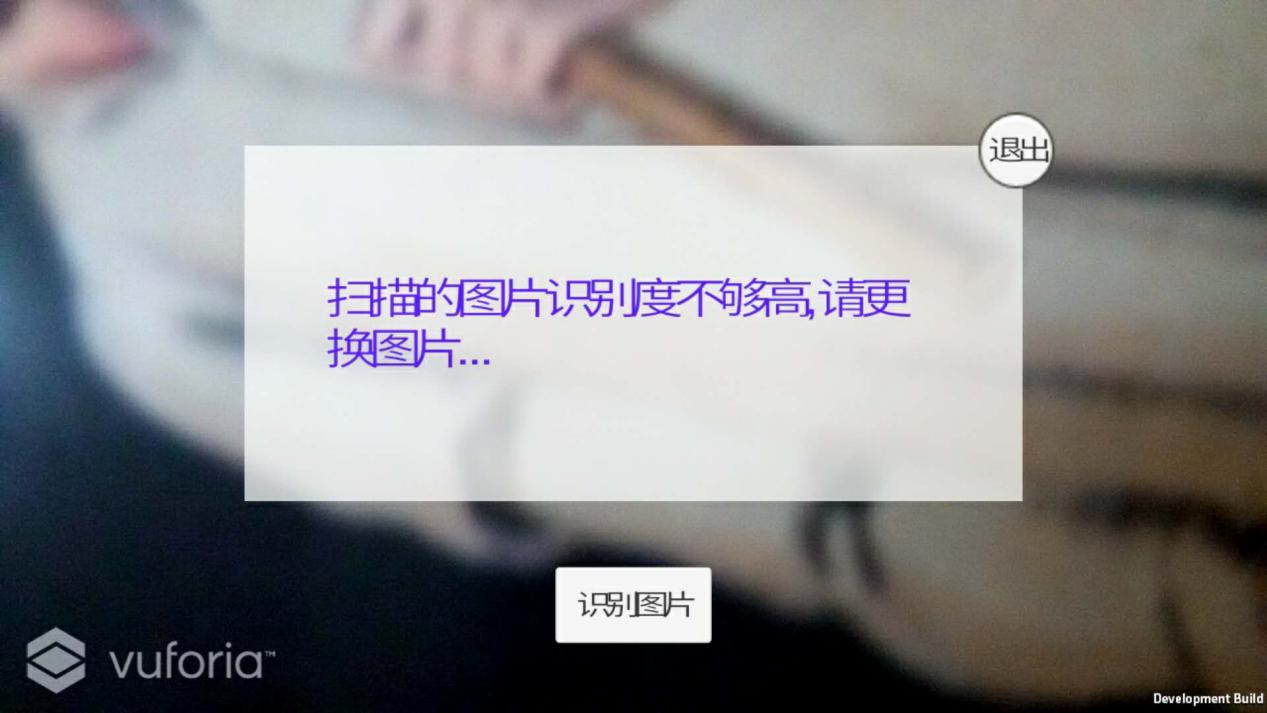
检查可识别：若可识别会提示识别及格信息。

检查不可识别：若不可识别，也就是有可能图片采样点不够（没有很多特征点，图片提供的信息不够丰富），提示不可识别。

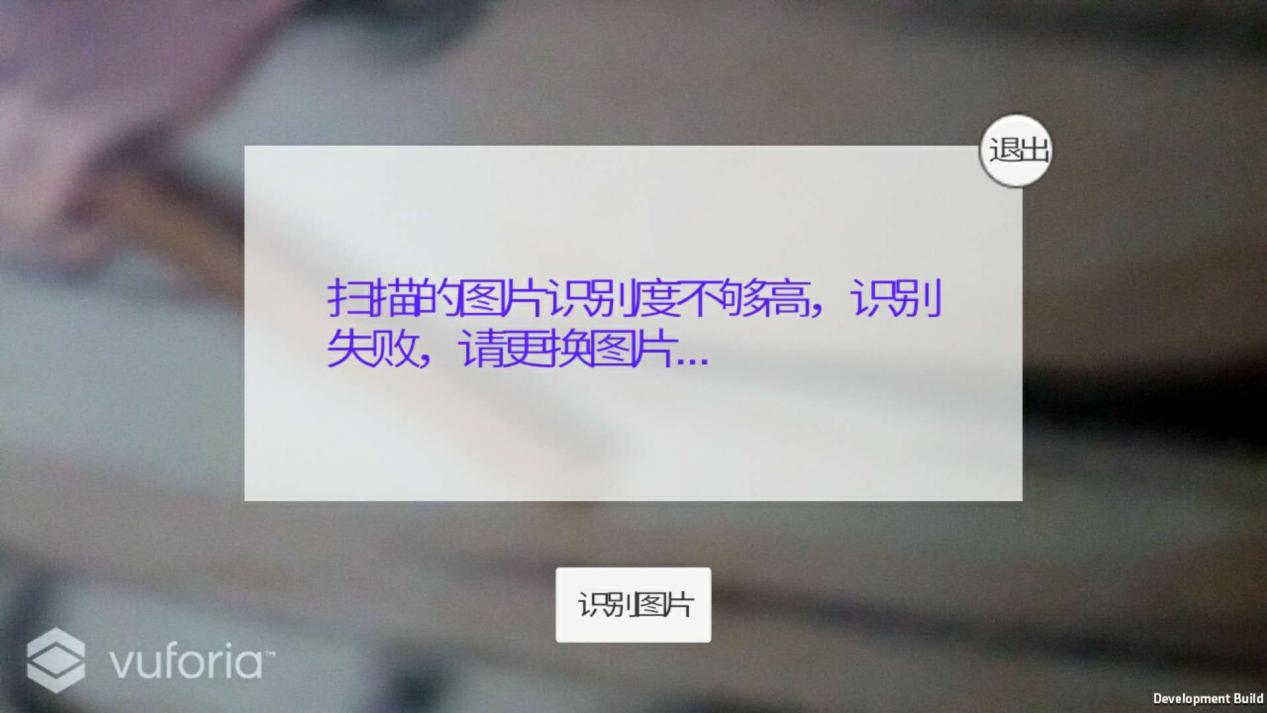
识别失败：在点击识别图片按钮后，若成功则直接进入游戏战斗场景，失败提示失败。



3.4 图片可识别出来



3.5 图片不可识别出来



3.6 图片识别失败

## 3.3 游戏战斗界面设计

游戏战斗界面主要有两个界面：



3.7 战斗界面



3.8 游戏暂停界面

战斗界面主要包含了玩家生命值，玩家分数，敌人血条，暂停按钮，小地图（右上角），摇杆和攻击按钮。小地图系统会在后面进行具体的介绍。

暂停界面是由战斗界面的暂定按钮点击所切换，主要用来暂停游戏，也可点击继续游戏来恢复游戏的流程。

# 第四章 游戏本地化系统实现

## 4.1本地化系统的设计思路

在游戏中，一个游戏的本地化是可以包含很多东西，不只是简单的语言切换，但因本项目时间有限，在这里只是简单地实现了语言的本地化。但仍要详细地介绍一下游戏该有的本地化是如何的。

游戏本地化的根基理所当然是语言的本地化，也就是翻译。毕竟只有用户能够看明白游戏给予的信息后，他们才能更容易去接受和理解一款游戏，否则他们很难吸收到游戏的故事情节和对游戏的体验也会大打折扣。然后根据当地文化的习惯而修改游戏中UI显示，例如根据阅读习惯，对文字信息进行排列顺序，姓名的排序，或者根据当地的宗教文化例如印度教把牛当作圣物等等。最后就是运营和推广，根据当地的节日来做活动。

在这里只是来简单地实现语言的本地化。设计思路如下：

对于本地方得支持方式都是类似得，第一步要设计好本地化所语言的资源文件。我这里采用普通的文本格式，然后指定用key-value的形式来读取数据，然后通过读取配置文件来管理语言的更新，并嵌入游戏的消息系统，可达到每当玩家在游戏设置更改语言时可使注册消息的文本直接更改语言，方便快捷。

## 4.2 配置中英文语言文本

这里简单展示本地化的资源文件。

English.txt文件：

Begin Game=Begin Game

Options=Options

About=About

Exit=Exit

Return=Return

Volume=Volume

Language=Language

Save=Save

game is loading=game is loading

Chinese.txt文件：

Begin Game=开始游戏

Options=设置

About=关于

Exit=退出游戏

Return=返回

Volume=游戏音量

Language=语言选择

Save=保存

game is loading=游戏正在加载中

## 4.3 本地化系统的设计实现

从上面的文件可知是根据英文为key，value为对应的本地化的语言。

接下来就是要写一个类来读取本地化的配置文件并解析，简单代码如下：

public class LocalizationManager

{

private static LocalizationManager \_instance;

public static LocalizationManager Instance

{

get

{

if (\_instance == null)

{

\_instance = new LocalizationManager();

}

return \_instance;

}

}

//可选语言

public const string Chinese = "Localization/Chinese";

public const string English = "Localization/English";

//已选语言

private string \_language = Chinese;

public string Language

{

get { return \_language; }

set

{

if (\_language != value)

{

\_language = value;

UpdateLanguageToDict();

}

}

}

//存储语言映射

private Dictionary<string, string> dict = new Dictionary<string, string>();

private LocalizationManager()

{

UpdateLanguageToDict();

}

private void UpdateLanguageToDict()

{

dict.Clear();

TextAsset ta = Resources.Load<TextAsset>(Language);

string[] lines = ta.text.Split('\n');

foreach (string line in lines)

{

if (string.IsNullOrEmpty(line) == false)

{

string[] keyvalues = line.Split('=');

dict.Add(keyvalues[0], keyvalues[1]);

}

}

}

public string GetValue(string key)

{

string value;

dict.TryGetValue(key, out value);

return value;

}

}

# 第五章 游戏增强现实的实现

## 5.1 游戏常用工具的实现

游戏开发有许多常用的辅助工具，这些工具是必不可少的，它们能大大加快我们的开发进度，也能更好地帮助游戏设计良好的框架以及实现一些常用的功能。

### 5.1.1 有限状态机的实现

FSM，中文名称为有限状态机，同时又称有限状态自动机，简称状态机或者FSM，表示在有限个状态下，以及在这些状态之间的转移切换动作等行为的数学模型。

有限状态机的主要是可以解决大部分比较复杂的游戏逻辑。因为游戏逻辑有时涉及到大量的判断条件，但基本的语言设计只提供了简单的if语句，switch语句，这是绝不够用的。不过，状态机也不是万能的，当游戏逻辑复杂到状态机都不能解决或者也很难去理清游戏的逻辑时，可考虑使用行为树。不过本游戏中的逻辑较为简单，因此使用状态机已足够。

状态机可归纳为4个要素，即现态，条件，动作，次态。“现态”和“条件”是因，“动作”和“次态”是果。详解如下：

1. 现态：是指当前所处的状态。
2. 条件：又称为“事件”。当一个条件被满足，将会触发一个动作，或者执行一次状态的迁移。
3. 动作：条件满足后执行的动作。动作执行完毕后，可以迁移到新的状态，也可以仍旧保持原状态。动作不是必需的，当条件满足后，也可以不执行任何动作，直接迁移到新状态。
4. 次态：条件满足后要迁往的新状态。“次态”是相对于“现态”而言的，“次态”一旦被激活，就转变成新的“现态”了。

状态机代码实现请查阅FSM文件夹。

### 5.1.2 游戏孵化器的实现

游戏孵化器的定义为：可通过对象池技术来可以不断自动地实例化出无限的游戏对象，但游戏的内存可以保持在一个稳定的数量级别。

在游戏中要创建许多的重复的，相同的游戏对象，比如粒子效果，游戏敌人，游戏主角等。游戏孵化器的意义就在于去管理这些对象的创建和回收，以及更加自动化。

这里有一个关键的技术点时对象池，对象池的使用在各个领域都用得很多，因此这里不作多介绍。简单来说是将重复的对象内存使用完后不将它释放掉，而是存储起来等待下次重新使用，避免重复创建重复释放的过程。

下面介绍部分游戏孵化器的实现：

public class Spawner : MonoBehaviour

{

public GameObject[] enemies;

public Transform[] spawnPostions;

public Transform spawnParent;

public float spawnWait;

public float spawnMostWait;

public float spawnLeasWait;

public int startWait;

public bool stop = false;

public int maxCount = 5;

public static int currentCount = 0;

int randEnemy;

int randPosition;

private void Start()

{

currentCount = 0;

for (int i = 0; i < enemies.Length; i++)

{

PoolManager.Instance.Create(enemies[i].name,enemies[i], spawnParent);

}

StartCoroutine(WaitSpawner());

}

private void Update()

{

spawnWait = Random.Range(spawnLeasWait, spawnMostWait);

}

IEnumerator WaitSpawner()

{

yield return new WaitForSeconds(startWait);

while (!stop)

{

if (currentCount < maxCount)

{

randEnemy = Random.Range(0, enemies.Length);

randPosition = Random.Range(0, spawnPostions.Length);

PoolManager.Instance.GetPool(enemies[randEnemy].name).Allocate(spawnPostions[randPosition].position, Quaternion.identity);

currentCount++;

yield return new WaitForSeconds(spawnWait);

}

yield return 0;

}

}

public void StartSpawn()

{

StartCoroutine(WaitSpawner());

}

private void OnDestroy()

{

for (int i = 0; i < enemies.Length; i++)

{

PoolManager.Instance.GetPool(enemies[i].name).Clear();

}

PoolManager.Instance.RemoveAll();

}

}

有关对象池的技术，因为网上也有许多教程，这里不作多详细介绍。本游戏中使用的对象池的技术代码文件在Spawner/Pool目录下，请自行查询阅读即可。

### 5.1.3 游戏消息系统的实现

在本游戏中，消息系统实质上是采用了设计模式中的订阅发布模式来实现的。订阅发布模式定义了一种一对多的依赖关系，能够让多个订阅者对象同时监听某一个自定义的对象。这个对象在自身状态发生变化时，会通知所有已订阅的对象，使它们能够自动执行绑定时的委托函数，从而达到一种收到通知并执行的效果。

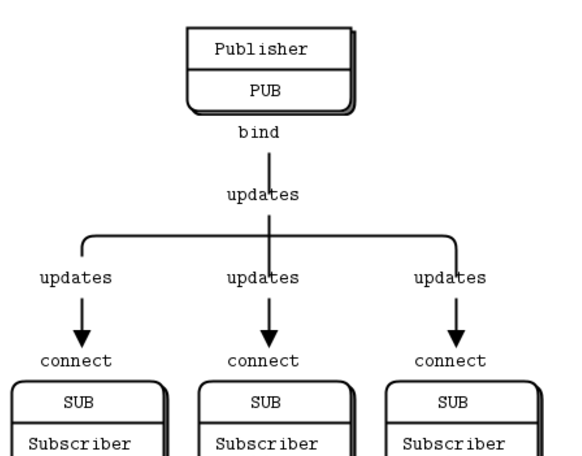
而消息系统同理，主要为了解耦，但也要合理运用，避免到处出现消息系统。

消息系统主要有如下特点：

一个发布者，多个订阅者的关系，1：N

当发布者数据变化时发布数据，所有订阅者均能够接收到数据并处理。

如图：



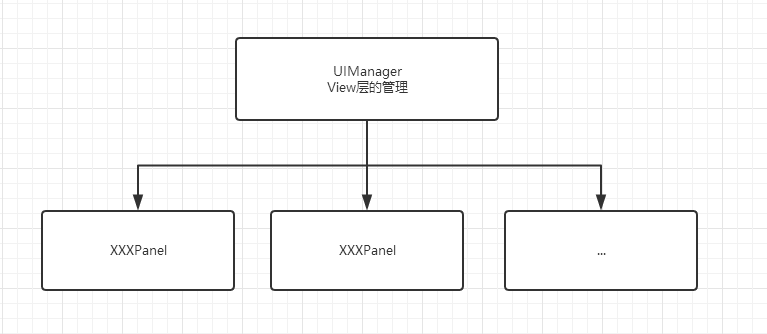
5.1 消息系统流程

### 5.1.4 UI管理系统的实现

UI管理系统是为了更方便，更有效和更系统地管理游戏显示界面。

在这里，我主要实现了BasePanel的页面基础类与UIManager页面管理类。

主要通过UIManager中的一个栈去管理显示的UI，通过压栈和出栈的操作，可使得UI显示获得4个生命周期，分别为显示，暂停，恢复和关闭。



5.2 UI管理框架

主要代码文件请查阅UIManager.cs文件。

### 5.1.5 游戏保存管理系统的实现

游戏的时长现在变得越来越长，有些较大游戏已经超过了 100 个小时的内容，环境的限制不可能让玩家一次性就玩完整个游戏。为了避免中断的情况，允许玩家保存游戏是所有游戏中最基本的一个功能——哪怕只是保存玩家的得分记录或者是保存一些玩家喜欢的设置。

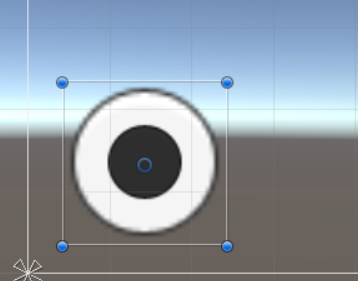
一般来说，可以直接使用Unity引擎自带的PlayerPrefs来实现游戏保存的功能，通过加上基本的封装，足够来做一些本地的存档了。但要是涉及到将玩家的存档放在互联网上，可使玩家可以在任何的手机平台上都可以通过账号来读取自己的存档也是一件很有意义的挑战。因此这里我借用了SaveGameFree这个插件来实现Json存档数据的实现，但因为本游戏不涉及联网，因为想做网上联网的话实现方式主要有将游戏数据上传到游戏的服务器的数据库中即可。

有关SaveGameFree这个插件可参考官方网站，这里不作多介绍。

## 5.2 游戏摇杆的实现

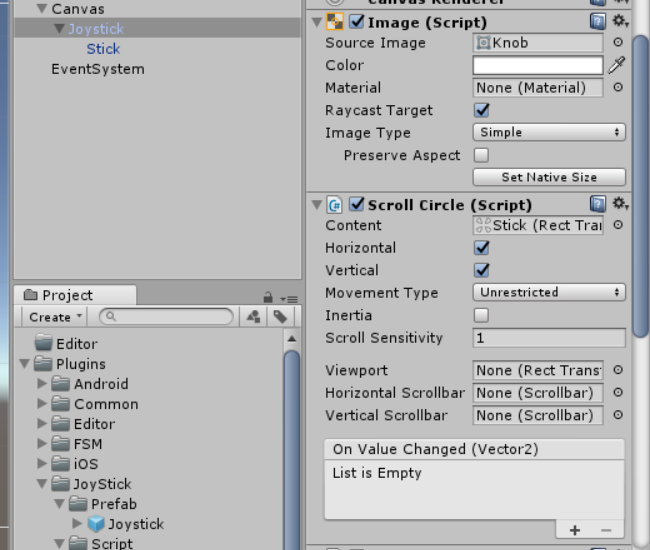
目前在手机游戏市场上，游戏中出现虚拟摇杆是一件非常常见的事，无论是个格斗游戏，MOBA游戏等，虚拟摇杆都是很常见的一种技术。下面特别介绍怎么利用Unity自身的组件特性来实现自己的虚拟摇杆，而不用去找插件，同时能理解大多数的摇杆是如果开发的。

使用Unity的UGUI可是很简单地实现自己的游戏摇杆。首先可以确定摇杆的基本图形，如图5.3



5.3 虚拟摇杆

后面白色的圆形实质上是Image和ScrollRect的结合。Image理所当然是为了显示背景图片，而ScrollRect的关键作用在于让黑色圆圈有一个来回拉伸的效果。但这里ScrollRect我们要继承重写一遍，如图5.4



5.4 虚拟摇杆

ScrollCircle继承ScrollRect的意义在于可以实现拉伸效果并扩展摇杆的拖动事件，实现对摇杆的值的计算和处理。

这里有关摇杆的实现代码在JoyStick文件夹下，请自行阅读。

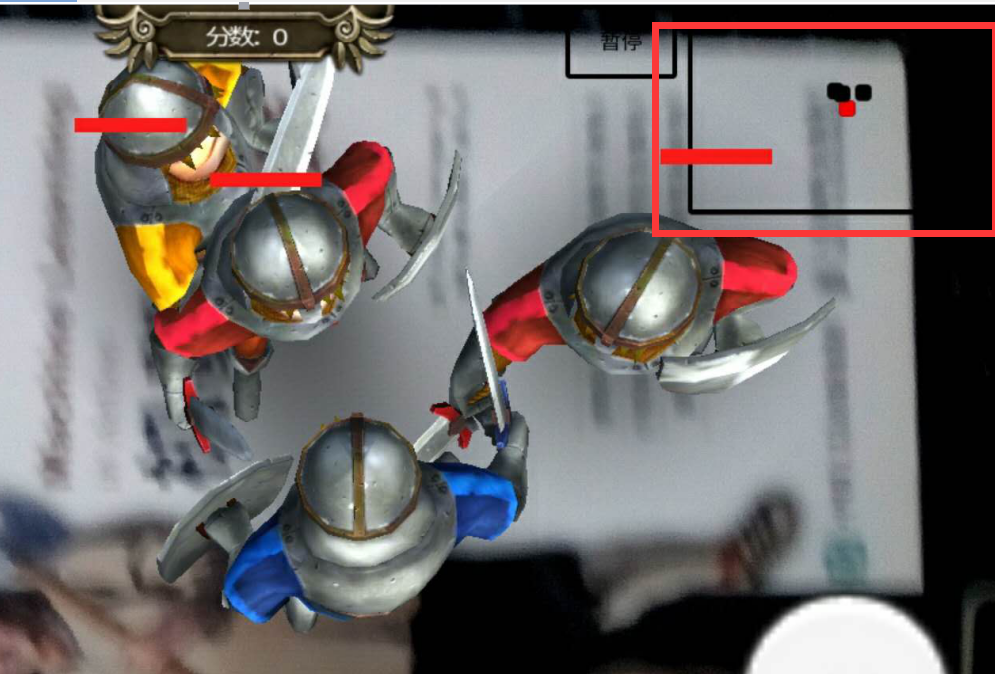
## 5.3 小地图管理的实现

众所周知，在场景较大的游戏中，小地图往往是不可避免的。而AR的场景是在现实世界中，可以说是非常大。因此很有必要要提供小地图或雷达来提示玩家周围的环境信息。

首先，在这个项目中，设定小地图的中心是以玩家操纵的主角为中心。其次，小地图也会使用其他的标识和有一定的等比例的距离来表示敌人和各方的距离。

Unity制作小地图主要有两种方法，第一种方法最为简单，就是添加多一个Camare，设置垂直投影，然后将投影得到的画面赋值Textrue，然后显示在UI上面。因此这里不作多介绍。

接下来主要介绍第二种小地图实现的方法，通过计算得出。如图5.5



5.5 小地图

说明：右上角为小地图，每个有颜色的方点代表一个坦克，正中心的红点是玩家。

原理：就是先设定代表玩家的方点，再通过计算玩家与敌人距离，按比例绘制其他圆点在小地图上，并根据角度计算出正确的方位。

代码如下：

MapIcon类：用于描述小地图的图标

namespace TMoonMapManager

{

public enum MapIconType

{

Player,

Monster,

Map

}

public class MapIcon : MonoBehaviour

{

/// <summary>

/// 图标存储在MapManager的唯一标识

/// </summary>

private int \_id;

public int Id

{

get { return \_id; }

set { \_id = value; }

}

/// <summary>

/// 图标类型

/// </summary>

public MapIconType type;

/// <summary>

/// 图标对应的真实物体

/// </summary>

public Transform target;

/// <summary>

/// 若超出地图范围是否保持在地图边缘

/// </summary>

public bool keedInBounds = true;

/// <summary>

/// 是否锁定图标的大小，若锁定则minScale设置会失效

/// </summary>

public bool lockScale = false;

/// <summary>

/// 是否锁定旋转，如果锁定，真实物体旋转而图标不会旋转

/// </summary>

public bool lockRotation = false;

/// <summary>

/// 图标最小的显示大小

/// </summary>

public float minScale = 1f;

RectTransform myRectTransform;

private void Start()

{

myRectTransform = GetComponent<RectTransform>();

}

private void LateUpdate()

{

Vector2 newPosition = MapManager.Instance.TransformPosition(target.position);

if (keedInBounds)

{

newPosition = MapManager.Instance.MoveInside(newPosition);

}

if (!lockScale)

{

float scale = Mathf.Max(minScale, MapManager.Instance.zoomLevel);

myRectTransform.localScale = new Vector3(scale, scale, 1f);

}

if (!lockRotation)

{

myRectTransform.localEulerAngles = MapManager.Instance.TransformRotation(target.eulerAngles);

}

myRectTransform.localPosition = newPosition;

}

}

}

MapMananger类：用于管理小地图的控制

namespace TMoonMapManager

{

public class MapManager : MonoBehaviour

{

/// <summary>

/// MapManager单例

/// </summary>

public static MapManager Instance;

/// <summary>

/// 实例化在小地图上的图标

/// </summary>

public GameObject IconPrefab;

/// <summary>

/// 小地图对应的环境

/// </summary>

public Transform target;

/// <summary>

/// 小地图缩放的比例

/// </summary>

public float zoomLevel = 10f;

/// <summary>

/// 小地图是否锁定旋转，如果锁定旋转就不会受到target三维目标旋转而导致icon旋转

/// </summary>

public bool lockRotation = false;

/// <summary>

/// 这个对象是否被销毁

/// </summary>

public bool isDestroy = false;

Vector2 xRotation = Vector2.right;

Vector2 yRotation = Vector2.up;

private void Awake()

{

Instance = this;

isDestroy = false;

}

private void LateUpdate()

{

if (!lockRotation)

{

xRotation = new Vector2(target.right.x, -target.right.z);

yRotation = new Vector2(-target.forward.x, target.forward.z);

}

}

private void OnDestroy()

{

isDestroy = true;

}

/// <summary>

/// 地图标识字典，存储已创建的标识

/// </summary>

private Dictionary<int, MapIcon> mapIcons = new Dictionary<int, MapIcon>();

/// <summary>

/// go要已被实例出来，或者被对象池创建出来

///

/// 根据传入的go在地图上创建对应的go图标

/// </summary>

/// <param name="go">图标对应的对象</param>

/// <param name="typeName">图标的资源名字</param>

/// <param name="type">图标类型</param>

/// <param name="minScale">图标在地图上显示的最小大小</param>

/// <param name="LockScale">是否锁定图标原有的大小，设置为true则minScale无效</param>

/// <param name="KeedInBounds">若超出地图范围是否保持在地图边缘</param>

/// <param name="LockRotation">是否锁定旋转，如果锁定，真实物体旋转而图标不会旋转</param>

public void AddMapIconByGO(GameObject go, MapIconType type)

{

MapIcon icon = GameObject.Instantiate<GameObject>(IconPrefab, this.transform).GetComponent<MapIcon>();

icon.target = go.transform;

icon.Id = go.GetInstanceID();

icon.type = type;

Image image = icon.GetComponent<Image>();

switch (type)

{

case MapIconType.Player:

image.color = Color.red;

break;

case MapIconType.Monster:

image.color = Color.black;

break;

}

mapIcons.Add(icon.Id, icon);

}

public void RemoveMapIconById(int id)

{

Destroy(mapIcons[id].gameObject);

mapIcons.Remove(id);

}

public MapIcon GetMapIconById(int id)

{

return mapIcons[id];

}

/// <summary>

/// 将3d物体坐标转换为2d图标的坐标

/// </summary>

/// <param name="position"></param>

/// <returns></returns>

public Vector2 TransformPosition(Vector3 position)

{

Vector3 offset = position - target.position;

Vector2 newPosition = offset.x \* xRotation;

newPosition += offset.z \* yRotation;

newPosition \*= zoomLevel;

return newPosition;

}

/// <summary>

/// 将3d物体旋转转换为2d图标的旋转

/// </summary>

/// <param name="rotation"></param>

/// <returns></returns>

public Vector3 TransformRotation(Vector3 rotation)

{

if (!lockRotation)

{

return new Vector3(0, 0, -rotation.y);

}

else

{

return new Vector3(0, 0, target.eulerAngles.y - rotation.y);

}

}

/// <summary>

/// 限制icon移出地图，如果移出地图，则会卡在地图边缘

/// </summary>

/// <param name="point"></param>

/// <returns></returns>

public Vector2 MoveInside(Vector2 point)

{

Rect mapRect = GetComponent<RectTransform>().rect;

point = Vector2.Max(point, mapRect.min);

point = Vector2.Min(point, mapRect.max);

return point;

}

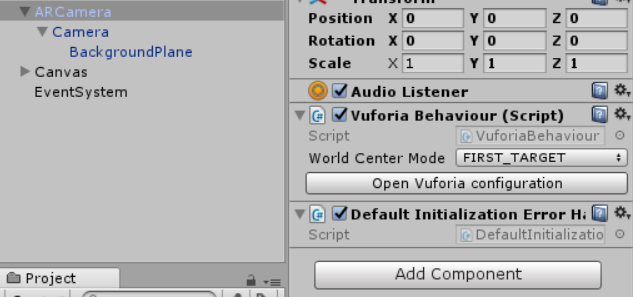
}

}

## 5.4 游戏功能实现

### 5.4.1 摄像机控制

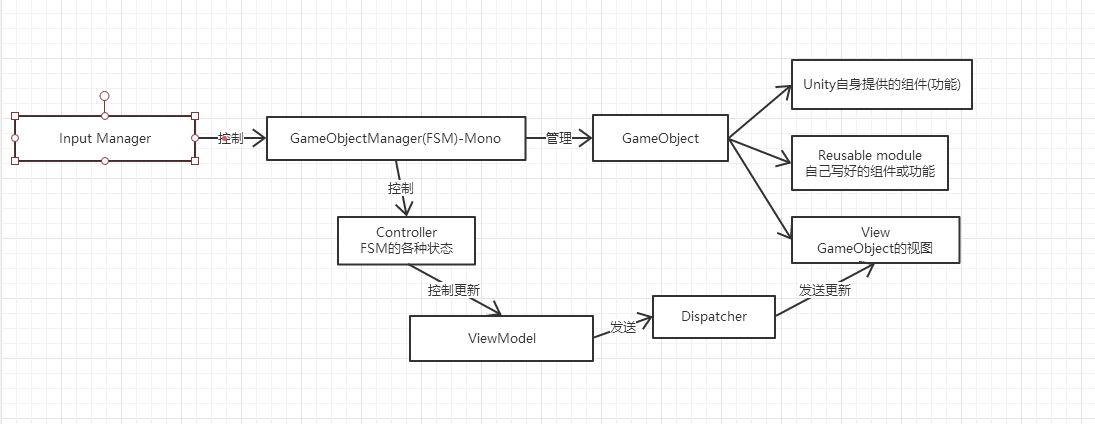
因为该游戏为AR游戏，所以摄像机的控制为玩家的相机，玩家可通过摆动手机的摄像头来转移游戏的视角，实现对游戏人物的多方位观察。如图5.6



5.6 AR摄像机配置

### 5.4.2 游戏的分层管理

在游戏中，我采用了类似MVC的思想来实现了游戏的基本框架，框架结构图5.7



5.7 游戏逻辑框架

Controller：主要让有限状态机来实现，通过有限状态机的状态来分裂出多个Controller，即每个状态都是一个Controller，主要用来解耦部分逻辑并实现对数据层的操作。

View：主要让Unity中原本的GameObject来代理，基本上只挂载一些基本的显示的组件和绑定相应的事件去获取对应的数据层的更改的值。

Model：实质上是ViewModel，在Controller中控制Model数据层的更新，Model数据层更改时通过发送相应的事件消息即可。

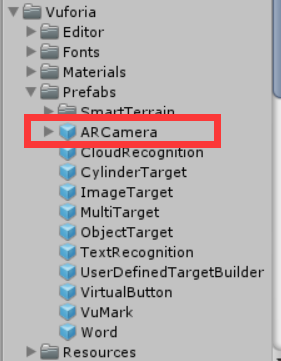
### 5.4.3 AR图片识别的实现

首先添加一个ARCamera，位于Vuforia SDK导入的文件夹中，这是一个特殊的相机类型，由Vuforia来实现，主要用于支持手持设备和数码眼镜的增强现实应用。

主要步骤：

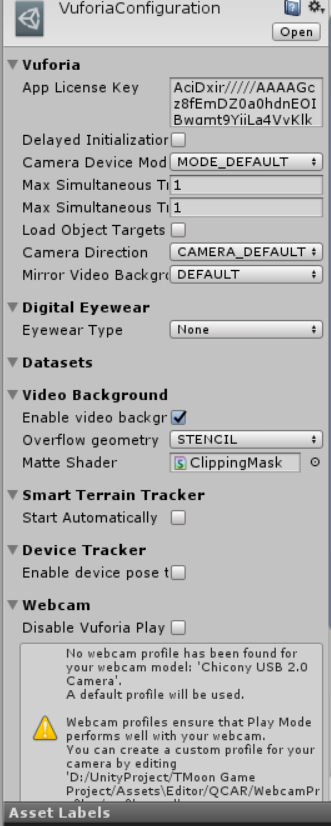
1. 添加ARCamera

添加一个ARCamera。删除主摄像头



5.8 ARCamera

1. 打开全局Vuforia配置检查器



5.9 Vuforial配置

1. 激活目标数据库

加载并激活要使用的目标数据库，如果是刚刚使用Vuforia并希望评估所有的目标类型，建议加载并激活所有目标数据库。

1. 添加目标到您的场景

你可以通过在Vuforia菜单中选择相关联的游戏对象，将Vuforia目标组件添加到场景中。目标游戏对象将添加到您的场景层次结构中，并在您的场景中显示。

可以在检查器中配置每个目标对象。选择要使用的目标的数据库和目标名称。

### 5.4.4 玩家与怪物的对战逻辑实现

游戏种玩家和怪物的逻辑分别是由状态机来编写的，采用状态机的设计模式来编写可达到即简单又清晰的效果。玩家和怪物会根据各自的状态采用不同的行为模式，进而写入数据层，达到血量值的降低和死亡的效果。

逻辑管理设计图如下：

EnemyManager

UI

PlayerModel

PlayerManager(FSM)

WalkState

...

AttackState

IdleState

可读可写

只读

只读

5.10 Player管理框架

玩家状态机管理类代码如下：

public class PlayerManager : MachineBehaviour

{

/// <summary>

/// 怀疑是Vuforial SDK的问题，当每一次进入这个脚本时所有的资源都被重置或者释放了，放在正常场景中(非AR场景)则不会产生这个问题

/// 因为在这个脚本第一次实例化的时候就用一个静态变量存储起来，所有用到这个脚本的都直接使用这个静态变量

/// </summary>

public static PlayerManager Instance;

[HideInInspector]

public Animator anim;

[HideInInspector]

public CharacterController cc;

[HideInInspector]

public Vector3 direction;

[HideInInspector]

public bool isCombos = false;

[HideInInspector]

public bool canCombos = false;

[HideInInspector]

public bool isAttackClick = false;

/// <summary>

/// 玩家移动速率

/// </summary>

public float moveSpeed = 3f;

/// <summary>

/// 玩家旋转速率

/// </summary>

public float rotateSpeed = 2f;

/// <summary>

/// 玩家对敌人造成的伤害

/// </summary>

public int damage = 10;

public bool isAttack = false;

private void Awake()

{

Instance = this;

anim = GetComponent<Animator>();

cc = GetComponent<CharacterController>();

}

public override void AddStates()

{

AddState<PlayerIdleState>();

AddState<PlayerWalkState>();

AddState<PlayerDieState>();

AddState<PlayerAttackState>();

AddState<PlayerGetHitState>();

SetCurrentState<PlayerIdleState>();

}

public override void Start()

{

base.Start();

MapManager.Instance.AddMapIconByGO(this.gameObject, MapIconType.Player);

}

public override void Update()

{

currentState = Instance.currentState;

states = Instance.states;

base.Update();

}

/\*\* Input Handler \*\*/

public void OnBeginControl()

{

}

public void OnControlling(Vector3 pos)

{

Instance.direction = new Vector3(pos.x, 0, pos.y);

}

public void OnEndControl()

{

Instance.direction = Vector3.zero;

}

public void OnAttackBtnClick()

{

if (Instance.IsCurrentState<PlayerDieState>())

{

return;

}

if (Instance.isCombos)

{

Instance.canCombos = true;

}

Instance.isAttackClick = true;

}

public void Combos()

{

Instance.isCombos = true;

}

public void DisCombos()

{

Instance.isCombos = false;

}

public void Attacking()

{

Instance.isAttack = true;

}

/\*\* Input Handler \*\*/

}

部分状态机的代码可查阅Player/PlayerStates文件，这里涉及到的逻辑较多，因此不展示代码，但可通过管理类以及上面提及的FSM有限状态机的介绍，仔细阅读便可理解其中的运行顺序。

同时，有关怪物行为逻辑的同理，项目中也是通过FSM去管理怪物的逻辑，与PlayerManager类似，可查阅Enemy/EnemyManager和EnemyStates文件。

# 第六章 游戏测试

在游戏测试方式中除了基本的软件测试，还要思考游戏本身的特点，如对游戏的可玩性，合理性和平衡性等方面进行游戏测试，保证游戏能在正常人的思维的操作来达到通畅可运用的效果。而且，为了平衡游戏平台间硬件使用率存在差异的特点，也可以从内存资源占用，操作响应延迟等多方面对游戏性能进行测试。

综上所述，本游戏的测试首要从多方面游戏的功能性、可玩性、逻辑性和内存占用率。

## 6.1 功能测试

本游戏的测试方式采用黑盒测试方式对游戏功能来进行覆盖性测试。主要测试内容包括游戏UI，战斗逻辑，地图逻辑，数据存储/读取模块等功能。

主菜单界面：开始游戏/设置界面/玩法介绍/退出游戏 测试通过

AR识别界面：不可识别/可识别/识别失败 测试通过

战斗逻辑：敌人伤害/敌人追踪/敌人死亡 测试通过

地图逻辑：小地图正确显示 测试通过

保存/读取：测试通过。

## 6.2 内存测试

测试情况如下：

测试设备：魅蓝2

内存占用：275M

CPU占用率：10%

GPU占用率：5%

通过测试软件数据显示，各方面数据也均在正常范围之内。

**第七章 总结与展望**

此游戏的开发是采用单机模式设计来实现的，部署的操作系统平台用的是安卓系统4.2以上版本，配置人物模型是4个，能实现任意图片识别。作为AR游戏开发，经过调研、设计和测试等几个过程，最终达到游戏简单，有趣和拥有一定的可玩性的基本要求，在此同时也展示了AR的趣味性。

在本文这些设计的基础上，能够对各个功能模块进行了实现，同时在一些核心的问题和工具库进行详细全面的介绍，包括小地图系统，游戏分层管理，本地化系统等多个系统。本文到此也差不多结束了，在本文中，主要介绍了游戏开发中的一些常用的技术和问题的解决方法，同时也针对增强现实的游戏进行了大胆地想象和实践。在未来，我相信随着科学技术的发展，在增强现实的技术上也会得到突发猛进的发展，然后不仅可以在游戏市场上增加新的色彩，也可以在我们的日常生活中大放异彩。

**参考文献**

[1]加藤政树.Unity游戏设计与实现[M].人民邮电出版社,2017年03月

[2]李晔.Unity AR 增强现实完全自学教程[M].电子工业出版社,2017年09月

[3]Francesco Sapio.Unity UI Cookbook[M].Packt Publishing,2015年12月29日

[4]Clifford Peters.Unity AI Game Programming[M].Packt Publishing Limited,2015年9月24日

[5]吴志达.一个基于Unity3d游戏引擎的游戏研究与实现[D].中山大学,2012年

[6]严朝军，周其力，张宁.游戏中的程序设计技术[J].电脑开发与应用,2000(05)

[7]周洋.增强现实技术（AR）在游戏上的运用研究[j].无线互联科技,2016年

[8]许轶超.虚拟现实交互设计课程教学改革探索与实践[j].福建电脑,2017年

[9]张利强.基于AR+Unity3D的廉洁游戏——急速廉球[j].电脑知识与技术,2017年

[10]马春江,张明禎.Unity3d游戏开发中的面向组件思想和MVC框架[j].信息与电脑(理论版),2017年

**致谢**

本论文在撰写与本项目在开发中，得到了向燕飞老师认真耐心的指导和支持，每次在项目过程中遇到的问题或论文撰写中不明白的地方，都能得到向老师的详细的回复和耐心的解答。不仅如此，向老师还将自身丰富的教学经验，实践经验和开发思想一一提示与扩展，给予了论文和项目极大的思路，对项目的开发工作起到了决定性的指导作用。同时，在此过程中，也体会到了向老师这种严谨的工作态度，这种严谨的态度也为即将从事的软件研究工作中的我指引了方向。这段经历将使我受益终身。因为在这里，本人衷心地感谢我的指导教师向燕飞，以及感谢在我身边一直关心和支持我的同学，也感谢关心和耐心帮助过我的校领导。向他们表示深深的谢意。