

**毕业论文（设计）**

题目 基于Unity游戏引擎的策略游戏设计

|  |  |  |  |
| --- | --- | --- | --- |
| 学 院 | 软件学院 | 专 业 | 软件工程 |
| 学生姓名 | 林 宏 伟 | 学 号 | 173110028 |
| 指导教师 | 苏 乐 辉 | 职 称 | 讲 师 |
| 完成日期 | 2021年4月13日 | | |

**教务处制**

**基于Unity游戏引擎的策略游戏设计**

**软件学院 软件工程专业 173110028 林宏伟**

**指导教师 苏乐辉 讲师**

**摘要**

游戏是当今社会主流娱乐活动之一，结合了文学、美术、视听等内容，被人们称为是第九艺术。随着艺术，科技，教育等领域的发展，让游戏开发行业变得更加火热。Unity是目前游戏开发主流的游戏开发引擎，家喻户晓的王者荣耀，闻名海内外的原神等等，都出自于Unity，这足以证明Unity的强大能力。

本文主要以Unity2019.4为开发环境，设计并实现一个策略类游戏，画面风格偏向于卡通可爱一些。主要介绍游戏的设计与实现方法，逐步讲解一些Unity开发游戏的事项以及素材的使用和代码的实现，最终形成一个较为完整的游戏并在Windows平台上调试运行。不过这次设计的即时策略游戏还不够完善，由于时间关系，敌人和玩家可控制单位只有一种，关卡也只有一个，这些都需要后续再去进一步实现完善。

**关键词：策略游戏；Unity；游戏设计**

**Simulation game design based on Unity game engine**

**Software college Software Engineering 173110028 Lin Hongwei**

**Instructor Su Lehui Lecturer**

**Abstract**

Game is one of the mainstream entertainment activities in today's society. It combines literature, fine arts, audio-visual and other contents, and is known as the ninth art. With the development of art, science and technology, education and other fields, the game development industry has become more hot. Unity is the mainstream game development engine of the current game development. The well-known King's glory and the well-known original God at home and abroad all come from unity, which is enough to prove the powerful power of unity.

This paper mainly takes Unity 2019.4 as the development environment, designs and implements a strategy game, and the screen style tends to be more cartoon cute. This paper mainly introduces the design and implementation method of the game, and gradually explains some items of unity development game, as well as the use of material and the implementation of code. Finally, a more complete game is formed and debugged on the windows platform. However, the design of the real-time strategy game is not perfect, because of the time relationship, the enemy and players can control only one unit, and there is only one level, which need to be further improved later.

**Key words：Game Design; Simulation Game; Unity**

**目录**

[1 绪论 1](#_Toc17506)

[1.1 研究的意义和目的 1](#_Toc18325)

[1.2 国内外研究现状 1](#_Toc1034)

[1.3 主要工作 1](#_Toc16114)

[1.4 论文结构 2](#_Toc13845)

[1.5 本章小结 2](#_Toc12037)

[2 相关技术概述 3](#_Toc26456)

[2.1 运行环境 3](#_Toc23143)

[2.2 开发工具及相关技术 3](#_Toc23762)

[2.2.1 Unity引擎 3](#_Toc29625)

[2.2.2 FairyGUI与UGUI 4](#_Toc440)

[2.2.3 A\* Pathfinding Project Pro插件 4](#_Toc6163)

[2.2.4 2DSpriteShare插件 4](#_Toc10929)

[2.2.5 ScriptableObject 4](#_Toc28092)

[2.2.6 AssetBundle 4](#_Toc26)

[2.2.7 Unity中对象池的应用 4](#_Toc4591)

[2.3 本章小结 4](#_Toc12113)

[3 游戏需求分析 5](#_Toc5496)

[3.1 系统需求概述 5](#_Toc17229)

[3.2 游戏背景 5](#_Toc1890)

[3.3 系统可行性分析 5](#_Toc18196)

[3.3.1 技术可行性 5](#_Toc17949)

[3.3.2 经济可行性 5](#_Toc10448)

[3.3.3 操作可行性 6](#_Toc21183)

[3.4 系统功能需求分析 6](#_Toc19084)

[3.5 非系统功能需求分析 9](#_Toc32583)

[3.6 本章小结 10](#_Toc21989)

[4 游戏设计 11](#_Toc22544)

[4.1 游戏主菜单设计 11](#_Toc21040)

[4.2 游戏商店设计 12](#_Toc27099)

[4.3 游戏单位设计 13](#_Toc14566)

[4.4 游戏玩法设计 15](#_Toc20754)

[4.5 本章小结 19](#_Toc25172)

[5 游戏实现 20](#_Toc9265)

[5.1 游戏主菜单界面的实现 20](#_Toc13532)

[5.2 游戏商店的实现 21](#_Toc6784)

[5.3 游戏单位的实现 23](#_Toc4952)

[5.3 游戏玩法的实现 25](#_Toc6521)

[5.4 本章小结 36](#_Toc11086)

[6 测试结果和分析 37](#_Toc26260)

[6.1 游戏运行环境测试 37](#_Toc2591)

[6.2 游戏基本功能测试 37](#_Toc26711)

[6.3 游戏性能测试 39](#_Toc22615)

[6.4 本章小结 40](#_Toc30924)

[7 总结与展望 41](#_Toc21698)

[7.1 总结 41](#_Toc7377)

[7.2 展望 41](#_Toc13563)

[参考文献 42](#_Toc20202)

[致谢 43](#_Toc32646)

# 1 绪论

## 1.1 研究的意义和目的

随着智能移动终端和PC端的普及，我国早已步入了电子时代，电子信息的飞速发展，使得游戏产业蓬勃发展[1]。在现在这种快节奏的生活方式里许多人总会利用工作之余的一些闲暇时间进行娱乐玩一会游戏。游戏行业也开始大放异彩，琳琅满目的游戏，各式各样，有的游戏几百兆的游戏大小就可以让人感到暖流涌动，有的游戏只用像素的风格就可以让无数人流泪。也有为增进朋友或情侣之间感情的游戏。人们花费极少的成本就可以在游戏里体验到绝美风景，动听的音乐，动人的故事，快意恩仇，异世界拯救世界的快乐，没有任何物质纠葛的爱情等等。也诞生了职业的电竞选手。在英雄联盟中国队赢得世界冠军时中国各地高校宿舍响起的欢呼声。这些足以证明游戏的魅力和前景。人们会想花更少的成本去体验到更多东西，游戏无疑成为了一种很好的选择。

在全球，“马里奥”是当之无愧的游戏IP价值榜首，截至2018年，马里奥已经获得超过300亿美元的收益[2]。中国的游戏产业情况也相当可观。根据中国音数协游戏工委最新的统计数据，2020年国内游戏市场实际营销总额达到2786.87亿元，中国自主研发游戏的海外市场实际营销总额也达到了154.50亿美元，游戏精品化进程进一步提速，有更多游戏走出去，同时实现跨界融合，通过电竞、短视频、直播、以及周边衍生来发展产业生态[3]。庞大的市场吸引着越来越多人跻身进入游戏行业，同时也带动了周边行业的快速发展。

游戏早已是许多人生活中不可或缺的一部分，或许是需要它度过闲暇时光，或许是需要它挺过艰难岁月，或许是借助它忘却生活的不易，或许是让它成为心理上的归宿。“炉石传说”在中国有着很高的知名度，原名《炉石传说：魔兽英雄传》在2014年由暴雪娱乐公司出品的策略类卡牌游戏。游戏背景设定于暴雪的魔兽系列，共十位魔兽中的角色作为十种不同的职业。而玩家要做的，就是根据己方现有的卡牌组建合适的卡组，指挥英雄，驱动随从，施展法术，与游戏好友或素不相识的对手一决高下。2014年荣获TGA2014最佳移动掌机游戏游戏大奖， 2015年获第二届SXSW年度移动游戏游戏大奖。2017年荣获谷歌Play Awards最佳多人游戏。

## 1.2 国内外研究现状

目前网络游戏的主要基本类型有：AVG（冒险类游戏）、SLG（策略游戏）、RPG（角色扮演类）、MMOG（大型多人在线）、FPS（第一人称射击类）[4]。本次课题研究的本质是策略游戏，以策略游戏为基础做出一些创新。

策略游戏（Simulation Game），简称SLG。游戏以取得各种形式的胜利为主题。这类游戏提供给玩家一个可以动脑筋思考问题来处理较复杂事情的环境，允许玩家自由控制、管理、使用游戏中的人或事物，通过这种自由的手段以及玩家们开动脑筋想出的对抗敌人的办法来达到游戏所要求的目标。现如今大多数策略游戏以回合制为主。国外热门的策略游戏有《Sid Meier’s Civilization VI》《Crusader Kings III》《XCOM 2》等等。国内热门的策略游戏有《自由与荣耀》《率土之滨》《傲视三国》等等。他们都在策略游戏的基础上发展出了自己的特色，也许是核心玩法上的，也可能是画面风格或者题材上的。

本课题设计实现的游戏在传统策略游戏上对战斗部分做出了一些创新,以3D即时战斗的方式进行，并且融入了现在流行的卡牌元素和一些Roguelike的理念。玩家在进入关卡前需要购买道具，进入关卡后，玩家通过使用道具控制单位完成任务通过关卡，在关卡中，击杀敌人将会获得货币以支持玩家后续进行道具的购买，有能力挑战下一关卡。

## 1.3 主要工作

先构思游戏的大体思路，收集整理出游戏所需资源，分析游戏创新点，确定游戏的主要功能以及流程，细分出功能的组成部分，根据是否具有技术可行性和经济可行性以及时间安排决定对组成部分的取舍。其次根据游戏内容与未来规划对数据层进行进一步分析，选择最为合适的数据存储方式，设计出合理的本地数据存储结构，确定开发所要用到的开发插件和技术，熟悉插件的使用以及解决技术上的难题，如：最大程度上灵活配置游戏、使游戏具有较好的可扩展性。设计好系统框架方便后续开发。最后对资源、数据、系统架构整合开发，实现游戏功能模块，对完成的功能模块进行测试。然后完善论文，完成课题的研究。

## 1.4 论文结构

第一章绪论：先简单概括了该课题研究的意义和目的，阐述了该课题国内外目前的一些大体情况以及市场上现有产品和本课题的区别，最后简要描述了整个课题的主要工作。

第二章相关技术概述：主要介绍开发的环境，Unity引擎的相关知识，还有一些用到的工具，游戏引擎插件和技术点。

第三章游戏需求分析：先对游戏做一个综合性的可行性分析，再对游戏的基本功能进行分析，确认整体项目是否能够实现。

第四章游戏设计：主要介绍游戏各个模块的功能设计。

第五章游戏实现：主要介绍游戏中的UI编辑和功能实现。

第六章测试结果和分析：主要介绍游戏的测试结果，并分析结果。

第七章总结：主要总结游戏中的不足和对未来发展的展望。

## 1.5 本章小结

本章首先简单介绍了本次课题的背景及目的，其次分析了本次课题的一些国内外现状和主要工作，最后简要概括了每个章节的主要内容。

# 2 相关技术概述

## 2.1 运行环境

本次课题会将Unity项目生成Windows平台可运行应用为前提进行开发与测试。采用PlasticSCM进行项目的版本控制，提高项目容错率，可在出现问题时进行版本回滚，避免过多的损失。使用Visual Studio 2019作为C#代码编写工具，FairyGUI作为UI开发工具，Photoshop作为图形编辑工具。游戏数据则是通过ScriptableObject存储为Asset资源文件。具体的运行环境如表2-1所示。

表2-1 运行环境表

|  |  |
| --- | --- |
| 目标平台 | Windows操作系统 |
| 游戏引擎 | Unity |
| 版本控制 | PlasticSCM |
| 数据存储方式 | ScriptableObject、AssetBundle |
| 数据存储格式 | Asset资源文件 |
| 相关工具 | Visual Studio、FairyGUI、Photoshop |
| 相关插件 | A\* Pathfinding Project Pro、2DSpriteShare |

## 2.2 开发工具及相关技术

本文设计的游戏采用Unity作为游戏开发引擎，Unity支持的编程语言为C#。在Windows操作平台下进行开发。UI界面由FairyGUI与Unity自带的UGUI混合开发。3D模型与UI美术资源从Unity Asset Store平台购买获得，并使用Photoshop图形处理软件对UI美术资源进行扩展。使用A\* Pathfinding Project Pro寻路插件辅助实现人物寻路。由ScriptableObject与AssetBundle搭配使用实现本地游戏配置数据和本地玩家数据灵活存储与读取。

### 2.2.1 Unity引擎

游戏引擎是指一些已编写好的可编辑电脑游戏系统或者一些交互式实时图像应用程序的核心组件，是一组成熟的配套代码框架和配套工具[5]。游戏的快速发展使得游戏开发者和游戏公司迫切需要能够高效，快速，开源，方便二次开发的强大引擎[6]。Unity是由丹麦Over the Edge公司研发的一款跨平台游戏引擎。在2005年发布了Unity的第一个版本Unity1.0.1，并在2008年正式将公司名称更名为Unity Technologies。2012年11月Unity Technologies公司发布支持DriectX11和Mecanim动画工具的新版本，新版Unity以其跨平台开发优势，功能完整，界面简洁，迅速占领游戏引擎市场[7]。大约在2016年将Unity3D更名为Unity，并更换了官网名称与官网地址。由于Unity引擎的灵活性，使创作者能够在多个平台上发布自己的作品，这些平台包括Windows、IOS、Android、Mac OS、PS4、Switch、Xbox One等等。Unity不仅在游戏领域应用有着强大市场，同时在ATM领域应用（汽车、传输、制造）、AEC领域应用（建筑、工程、施工）和影视动画领域应用也有着不小的成就。基于Unity引擎制作的游戏有：《原神》、《王者荣耀》、《炉石传说》、《纪念碑谷》、《明日方舟》、《精灵宝可梦GO》等。Unity提供了一整套完善的软件解决方案，创作者可在Unity中统一管理3D模型、动画、音乐、程序等，并将其进行整合，在不同平台上进行内容创作。Unity还提供了AssetStore商店，让开发者能够在其中找到现成的开发资源以及便捷的开发插件，大幅缩短开发时间。

### 2.2.2 FairyGUI与UGUI

FairyGUI与UGUI都用于UI开发，FairyGUI是由广州花谷软件公司制作的一款开源跨平台UI开发工具，而UGUI是Unity引擎内置的UI组件。FairyGUI编辑器基于Unity引擎制作，使用C#和Lua编写，不仅支持Unity还支持其他游戏引擎，如：CoCos2dx、CoCos Creator、Egret、LayaAir、Cry Engine等。由于其功能齐全、更适合团队配合且界面友好容易上手，FairyGUI受到了许多游戏开发团队的青睐。UGUI是Unity后来为了解决Unity的UI开发问题内置在Unity中。内置在Unity中的UGUI与Unity和其他插件更兼容，有着更多支持，更方便使用。两者都有自身独特的优势。本次采用FairyGUI为主UGUI为辅的方式进行UI的混合开发。

### 2.2.3 A\* Pathfinding Project Pro插件

A\* Pathfinding Project Pro是在Asset Store上的一个基于A\*算法的热门寻路插件。其功能丰富、强大、性能优异且方便好用所以备受好评。本游戏将会使用A\* Pathfinding Project Pro插件的寻路算法辅助完成玩家角色与敌方AI的寻路功能。

### 2.2.4 2DSpriteShare插件

2DSpriteShare是Unity官方推出的一个图形处理插件。它可以让设计者通过调用API轻松实现各种图片的拉伸平铺变形的效果。

### 2.2.5 ScriptableObject

”This is most useful for assets which are only meant to store data.”这是官方对于这个类的描述。ScriptableObject是一个数据容器，可以用来存储大量数据，它是可序列化的，正是因为这个特性决定了它的主要用途。通过将数据存储在ScriptableObject对象中，可以避免值拷贝来减少游戏运行时的内存占用。一般我们会将它保存为本地Assets资源文件，在改变从Assets资源文件读取的ScriptableObject对象数据时Assets资源文件数据将也会一起改变。

### 2.2.6 AssetBundle

Unity项目在生成目标平台应用后可以通过AssetBundle对不包含代码的资源进行更新。这项技术使游戏的灵活性大幅提升，开发者可在不重新生成或安装的情况下对游戏的资源和数据进行更新，避免了大部分文件未改变的情况下却重新下载安装的繁琐和浪费，同时避免了重新下载安装游戏所带来的大量游戏玩家流失，导致经济严重亏损。

### 2.2.7 Unity中对象池的应用

在Unity中频繁创建和销毁游戏物体将会大量消耗CPU和内存资源，为了避免这个问题，游戏中将会有一个单例去管理产生的游戏物体。当不需要这个物体时，不真正销毁游戏物体，而是取消激活，放入对象池中，等需要这个物体时再从对象池中取出，然后重置后再激活这个物体。

## 2.3 本章小结

本章主要对本次课题的运行环境进行一个简单的介绍，并描述了引擎，工具和技术相关的一些基础知识，为后续开发提供了参考价值。

# 3 游戏需求分析

## 3.1 系统需求概述

本游戏设定为俯视角即时战斗策略类PC端游，主要玩法为控制多个人物单位移动攻击完成任务取得胜利。游戏场景有两个。一个作为UI界面背景而使用，可以看到美丽的夜空和花草树木，还有一个营火和角色作为点缀，设计偏向于美观，舒适，提高画面整体美观程度。一个作为战斗地图使用，场景背景为森林和一处营地，更偏向于追求合理，简洁，注重玩家操作方便。打开游戏最先看到游戏主菜单，游戏主菜单中有开始游戏，继续游戏，离开游戏三个按钮，点击开始游戏按钮，玩家数据应同步为游戏初始数据并跳转至游戏商店界面，点击继续游戏按钮，玩家游戏数据应与上一次游戏数据一致并跳转至游戏商店界面，点击离开游戏按钮，将会退出游戏应用，返回至电脑桌面。玩家在进入关卡前可通过金币购买或出售道具，此时可查看玩家当前金币数量，金币足够即可购买游戏商店中的道具，出售自身持有道具可获得相应数量的金币，完成道具购买后可加载战斗场景进入关卡。关卡中有AI敌人进行场地的巡逻并攻击、追击发现的玩家单位。玩家可通过使用卡牌的方式使用道具，放置人物单位，控制人物单位进行移动和攻击，击败敌人，完成关卡任务。玩家还可以在关卡中查看游戏的提示信息和关卡任务信息。完成所有任务即为关卡胜利，在没有己方单位存活且未持有道具时还未完成所有任务时即为关卡失败。在关卡胜利后将会获得金币奖励并扣除使用的道具，进入游戏商店界面，做好准备后可进入下一关卡，开始新一轮挑战，反之关卡失败后玩家游戏数据将会被重置为初始数据，玩家只能选择返回主菜单界面。

同时游戏系统还需要有非游戏功能的基础需求[8]。玩家可以关闭游戏退出至电脑桌面，界面和场景切换也有过渡画面，使界面和场景的切换更为自然。玩家可以开始新的游戏，清空所有数据，重头开始。也可以选择继续游戏，继续挑战未完成的关卡。另外玩家在战斗中也可以暂停游戏。从任何一个界面或场景都可以返回至游戏主菜单，为玩家提供最大程度的便利。

## 3.2 游戏背景

游戏发生在一个战斗任务中，玩家身为指挥官将要组织一队警务人员执行上级指派的危险任务，派往森林地区搜索和清理躲藏在森林中的邪恶势力，而该地区的敌人持有热武器，防弹衣等，设备齐全，火力十分强大，而且人数众多。好在我方装备也不差，于是玩家指挥着骁勇善战的部下与敌方激烈交战，最终在合理的指挥下歼灭了所有敌人。

## 3.3 系统可行性分析

可行性分析主要是对系统开发设计的主要内容进行调查研究和分析比较，为系统设计提供依据的一种综合性的系统分析方法[9]。在系统设计实现过程中，应考虑实际开发的各方面影响，从而评估系统是否具有可行性，下面将从技术可行性、经济可行性和操作可行性三个方面具体分析。

### 3.3.1 技术可行性

本次课题设计的游戏基于Unity游戏引擎，Unity已经帮我们管理好了大部分资源，无需去处理繁琐的细节，借助可视化编程工具，甚至不需要学习C#编程语言也可以进行开发，使用C#编程语言可以使游戏更加灵活。Unity拥有大量的插件，可帮助开发者轻松实现所需功能。Unity经过多年的更新迭代，已经趋于稳定，还有更加稳定的LTS长期支持版，帮助开发者进行游戏开发，避免工具问题导致开发难度加大。

### 3.3.2 经济可行性

首先，本次使用的Unity，Visual Studio Community 2019均为免费使用，FairyGUI不仅免费使用还完全开源，无需支付任何费用，游戏中所涉及的美术资源及插件已在打折时以低价购买无需再次付费，只需要在官方网站下载相应软件和资源，安装软件后将游戏资源导入Unity中就可以开始游戏开发。本文设计的游戏属于单机游戏，无网络功能，所以无需搭建服务器，不必产生额外费用。

其次，由于本次开发的小型游戏结构和内容相对简单且在Windows平台上运行，开发所使用到的设备仅需一台通用计算机，开发者完全可以做到独自设计，开发，测试与维护。开发中涉及的付费资源及插件已在早期通过正规渠道获取，所以在游戏开发过程中不会产生太多金钱费用。综上所述，在整个开发过程中都不会有太大的经济成本，开发者不用担心经济问题导致无法正常开发。

最后，游戏市场潜力较大且市场稳定，销售渠道众多，互联网技术发达。在游戏开发成功后可在多个平台下销售，产生较高的收益，并且游戏系统框架低耦合高内聚复用性也高，可在未来从游戏中将大部分功能剥离出来进行下一个游戏开发，极大地缩小了下一个相同类型游戏的开发成本。游戏运行不需要太好的设备，也不需要安装相应游戏运行环境，大部分电脑都能运行。玩家只需下载游戏即可运行游戏。

总而言之，本游戏开发成本低廉，没有太难的技术点，无网络通信成本，耗费资源较少，游戏维护成本低，游戏开发完成后收益高，在经济方面是可行的。

### 3.3.3 操作可行性

本文设计的游戏体积小，运行时使用内存少，计算量较少，大部分电脑设备皆可运行，对设备并没有太高的要求。游戏画面风格简约可爱，使游戏模型不需要很高的精度，大幅度减少了游戏体积和渲染时间，从根本上降低了设备压力，提高了游戏帧数。UI经过设计调整后，简洁明了，舒适大方，操作步骤简单，上手容易，玩家学习成本低，无需花费太多时间即可熟悉操作。在游戏中还有提示功能，让玩家更快地去了解游戏中的内容。提高了玩家体验。即使游戏的战斗方式是即时进行的也不需要高强度高难度的游戏操作，只需要稍微开动脑筋思考一番，简单地操作一下游戏就可以轻松获得游戏的胜利。另外游戏在战斗中还可以随时暂停游戏，可使玩家在游玩过程中暂停游戏，去处理其他要紧的事务。便利了玩家的同时也让玩家有了较好的游戏体验。

## 3.4 系统功能需求分析

本文所设计的游戏为单机游戏，无联网功能，所以无需服务端支持，客户端应具备以下几个基础功能：

1. 客户端正常运行，玩家可以随意在不同的PC端运行游戏，并且可在设备上保存相应玩家的游戏信息数据，保证游戏运行中数据与设备本地数据同步。可以正常处理游戏中的逻辑，保证客户端正常稳定地运行。
2. 游戏主菜单。玩家可以选择开始新游戏。放弃上一个存档，重置所有玩家数据，重新开始一个新的游戏存档。从头开始挑战游戏关卡。也可以选择继续游戏，读取本地游戏数据，还原玩家上一次退出游戏时的游戏数据，使玩家可以在同样的情况下继续挑战关卡。还可以选择退出游戏，返回电脑桌面。
3. 游戏商店。玩家可以在商店查看自身拥有的道具和金币数量，点击游戏商店出售的道具或自身拥有的道具时可查看道具的详细信息。通过金币可以进行道具的购买或出售。金币不足时可以挑战关卡获得更多金币。并且可以自由选择进入关卡或是返回主菜单界面。现有一种人物卡牌道具，具体如表3-1所示。

表3-1 道具列表

|  |  |  |  |
| --- | --- | --- | --- |
| 道具ID | 道具名称 | 道具描述 | 价格 |
| 20001 | 枪手卡牌 | 手持制式手枪，威力不大，但也够用，需在指定区域放置 | 22金币 |

1. AI敌人。AI敌人具备巡逻、索敌、追击、攻击等一些基础功能，与玩家进行互动，给予玩家一定的反馈，提升游戏体验。AI敌人的用例图如图3-1。

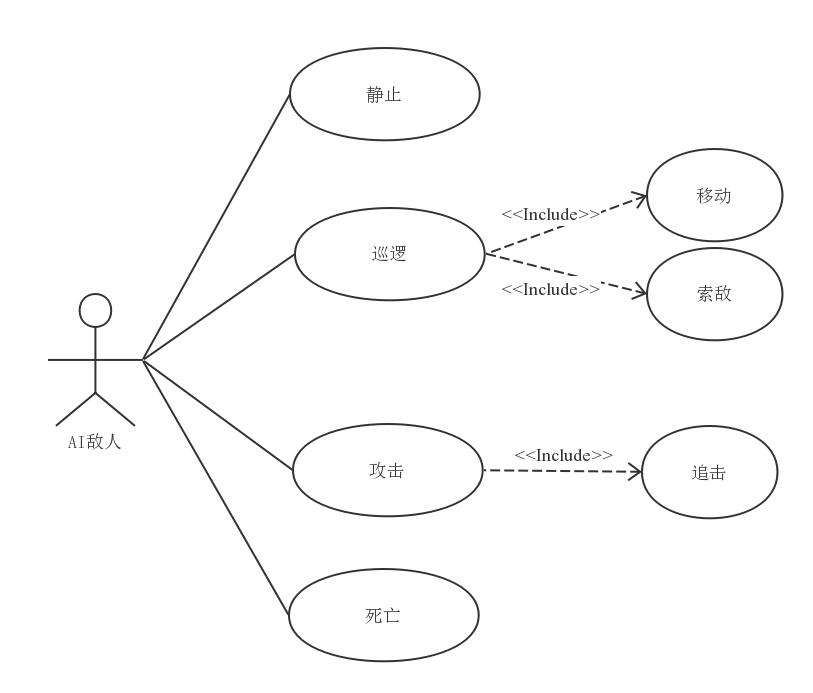


图3-1 AI敌人用例图

1. 完整正常的游戏流程。玩家在进入关卡之后，可以通过使用卡牌的方式使用道具，可在指定位置放置单位，用鼠标左键框选单位，控制单位进行移动，攻击等基础操作。现有一种关卡，具体如表3-2所示。两种游戏单位，这两种游戏单位的概括描述如表3-3所示。可通过键盘快捷键完成一些相同操作。同时可移动缩放视角查看战场其他地方的情况。玩家可与关卡内的AI敌人进行战斗。同一队之间不可互相攻击。有一个任务组作为关卡胜利的条件，一个任务组中包含一个或多个任务。在玩家完成所有任务且有己方单位存活或拥有道具时弹出胜利结算界面，可选择退出至主菜单或者进入游戏商店做好下一关卡的战斗准备，反之在玩家所有己方单位死亡且未拥有道具时依然没有完成所有任务则弹出失败结算界面，同时清空存档，重置玩家游戏数据，玩家可退出至游戏主菜单。挑战关卡的用例图如图3-2所示。

表3-2 关卡列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 关卡ID | 关卡名称 | 下一个关卡ID | 任务组 | 关卡场景名称 |
| 1 | 关卡-1 | 1 | 1001 | 2 |

表3-3 游戏单位设定表

|  |  |  |
| --- | --- | --- |
| 单位名称 | 模型预制体ID | 属性 |
| 手枪警察 | 20003 | 生命值适中，攻击力较低，攻击速度适中，  精准度适中，子弹速度适中，视野范围极大 |
| 手枪敌人 | 20004 | 生命值适中，攻击力较低，攻击速度适中，  精准度适中，子弹速度适中，视野范围适中 |

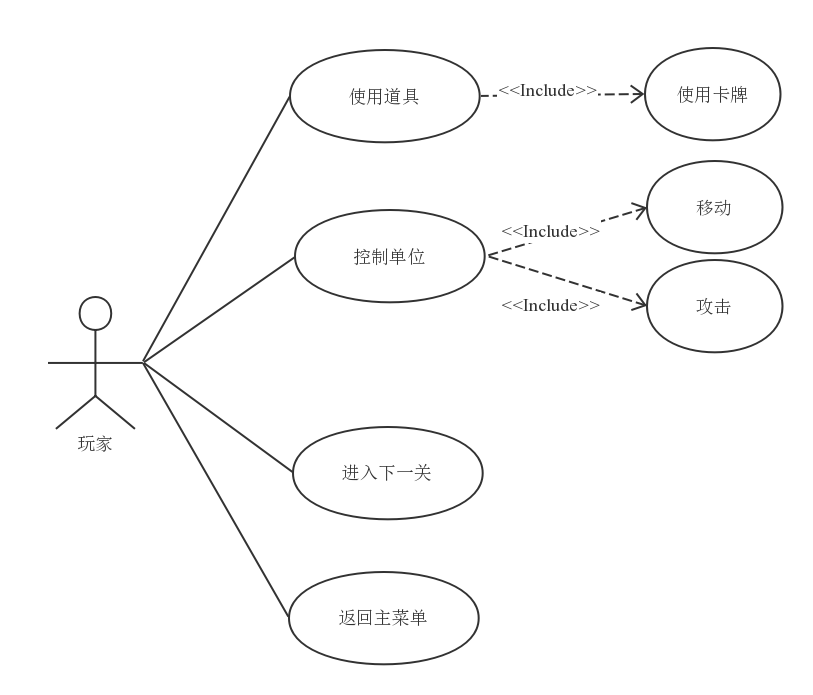


图3-2 挑战关卡用例图

1. 任务功能。每个关卡都有相对应的一个或多个任务，且玩家可查看当前关卡的任务数量以及任务说明。任务功能现有一种任务，具体如表3-4所示。

表3-4 任务列表

|  |  |  |  |
| --- | --- | --- | --- |
| 任务ID | 任务描述 | 显示优先级 | 组 |
| 1 | 消灭三个敌人 | 中 | 1001 |

1. 提示功能。玩家在进行游戏时能点击查看开发者给出的提示信息，帮助玩家更快地上手游戏，了解游戏内容。提示功能现有五种提示，具体如表3-5所示。

表3-5 提示列表

|  |  |  |
| --- | --- | --- |
| 提示ID | 提示描述 | 显示优先级 |
| 0 | 敌人受到攻击后将会反击  请小心 | 低 |
| 1 | 敌人的视野范围有限  可以巧妙利用 | 低 |
| 2 | 移动指令（快捷键Q）：  仅移动至目标地点 | 高 |
| 3 | 移动攻击指令（快捷键W）：  移动至目标地点，遇见敌人将优先攻击敌人 | 高 |
| 4 | 敌人只在一定范围内活动  打不过可以跑哦 | 中 |

1. 战斗暂停功能。玩家在战斗时可以点击暂停按钮，暂停战斗，弹出游戏暂停弹窗，并可在此窗口选择继续游戏或放弃战斗返回游戏主菜单界面。放弃战斗将不扣除此次战斗使用的道具。

## 3.5 非系统功能需求分析

游戏除了一些基本玩法功能外还应该有游戏之外的需求。首先游戏运行必须稳定。几乎所有游戏都有其客户端，用来连接服务端而为玩家服务[10]。客户端崩溃不仅影响游戏的正常运行还会影响玩家的游戏体验，甚至是给恶意玩家破坏游戏环境的机会[11]。游戏是否能够稳定运行直接影响到了玩家留存与游戏体验。其次是游戏的易修改性和可扩展性，在设计游戏整体框架时应提前准备好游戏后续的修改维护、二次开发、添加新功能或者新数据等一系列操作的可能性。所以设计的游戏框架必须灵活。还有一点需要注意的是本游戏的宗旨在于操作简单风格可爱，在玩家的交互体验上不应该有太难的操作方式，游戏的选材也应统一并且贴合主题，使游戏看起来更加整体。

## 3.6 本章小结

本章通过对游戏各方面的需求进行分析，详细描述了游戏中所需要的各种基本功能。明确了游戏的基本风格和游戏玩法。游戏需要实现的功能有：游戏正常稳定地运行，游戏主菜单，游戏商店，AI敌人的设计，玩家与AI敌人的战斗，任务功能，提示功能，暂停功能等等。介绍了每一个功能在游戏中的作用。

# 4 游戏设计

## 4.1 游戏主菜单设计

游戏主菜单是打开游戏后第一个见到的界面。决定了玩家对于游戏的第一印象。所以游戏主菜单对于游戏整体而言是非常重要的。游戏主菜单应该注重美观且布局简单明了。主菜单的主要功能是让玩家选择开始新的游戏还是继续上一次游戏或者是退出游戏至电脑桌面。根据玩家点击的按钮执行相应的操作。如图4-1所示。

按钮点击（玩家指令）

读取本地数据

开始游戏

继续游戏

离开游戏

重置玩家数据

关闭游戏

同步游戏数据

跳转至商店

跳转至商店

图4-1 游戏主菜单按钮点击处理图

游戏主菜单UI布局分为三个部分，左上角放置游戏名称标题，在右下角放置开始游戏，继续游戏，离开游戏等基础按钮，剩余部分将场景作为背景。每个按钮都添加了相应功能的触发事件，点击不同按钮会有不同的效果，开始游戏按钮添加的事件为将加载过渡画面，关闭当前UI并放入对象池中，重置玩家本地数据，从对象池中取出游戏商店UI并显示，关闭过渡画面；继续游戏按钮添加的事件为加载过渡画面，关闭当前UI并放入对象池中，从设备上读取玩家本地数据，同步至游戏中，从对象池中取出游戏商店UI并显示，关闭过渡画面；离开游戏按钮绑定的事件为关闭游戏应用，返回至电脑桌面。如图4-2所示，游戏主菜单界面布局。

游戏名称标题

开始游戏

继续游戏

离开游戏

背景场景

图4-2 游戏主菜单界面布局图

## 4.2 游戏商店设计

游戏商店的用途主要为售卖游戏道具，当玩家拥有足够的金币时可进行道具的购买并在游戏中使用，另外玩家也可售卖自身道具以换取金币。目前游戏商店售卖的道具只有一种远程类型的人物卡牌。道具价格等属性可通过游戏配置表配置。道具类实现了IJsonAttr自定义接口，可给道具添加Json格式的额外属性，道具配置表类内容如下：

[CreateAssetMenu(fileName="ItemConfig", menuName="Configs/ItemConfig")]//创建Asset文件特性

public class ItemConfig : ScriptableObject { //继承ScriptableObject类实现数据存储

public List<Item> cfs = new List<Item>(); //道具列表

[System.Serializable] //可序列化特性

public class Item : IJsonAttr {

public long id; //道具id

public bool canInBag = true; //是否可出现在玩家背包中

public ItemType itemType; //道具类型

public string icon; //道具图标

public string remark; //道具标识名称

public string name; //道具名称

public string desc; //道具描述

public int stackLimit; //道具可堆叠的堆数量

public int stackNum; //道具每个可堆叠的堆容量

public string useFunc; //对应的使用函数

[Multiline(3)] //Unity Inspector窗口多行文本特性

public string attrJsonStr; //道具属性Json字符串

[HideInInspector] //使字段在Unity Inspector窗口隐藏特性

public JsonData attrData { get; set; } //道具属性Json

}

private void OnEnable() {

for(int i = 0; i < cfs.Count; i++) {

if(!string.IsNullOrEmpty(cfs[i].attrJsonStr))cfs[i].attrData=JsonMapper.ToObject(cfs[i].att rJsonStr); //在加载道具配置表时将Json字符串转化为Json格式

}

}

public enum ItemType {

medium = 1001, // 媒介道具

currency = 1002, // 货币

card = 1003, // 卡牌

}

}

游戏商店的售卖道具均可点击查看道具的详细信息。每个售卖道具都有显示相应的购买价格，当玩家持有足够金币时可点击购买按钮进行购买，否则购买按钮将置灰不可点击，也无法购买，金币不够的时候可以通过进行游戏来赚取金币。游戏商店道具售卖列表与玩家持有道具列表为滚动容器，当道具过多不能全部显示时，可上下拖动列表来查看其他道具。游戏商店购买流程图如图4-3所示。

Y

N

N

商品购买

金币足够？

购买

结束

购买按钮置灰不可点击

图4-3 商品购买流程图

## 4.3 游戏单位设计

游戏单位是游戏中非常重要的存在，不管是玩家还是AI都可以控制各自的游戏单位。游戏单位的移动主要是通过A\* Pathfinding Project Pro插件完成，最常用到的API主要为AIBase.isStopped是否移动，AIBase.destination设置移动目标地点等等。插件将会基于A\*算法算出最优移动路径并移动游戏物体。插件主要设置如图4-4所示。



图4-4 A\* Pathfinding Project Pro插件主要设置

游戏单位的攻击方式通过使用实现了IWeapon接口的武器类，在动画事件中调用IWeapon接口的攻击方法实现游戏单位的攻击。IWeapon接口内容如下：

public interface IWeapon{

float Rate { get; set; } //武器使用速度

void StartUse(); //开始使用

void EndUse(); //结束使用

void UseOnly();} //无使用持续时间，仅执行一次

游戏中游戏单位——枪手的攻击通过武器类创建并发射带有盒型碰撞器的子弹预制体，通过碰撞器碰撞判定子弹射中的单位。如果被子弹射中的单位具有生命值，则扣除该单位的生命值。子弹预制体盒型碰撞器设置如图4-5和图4-6所示。

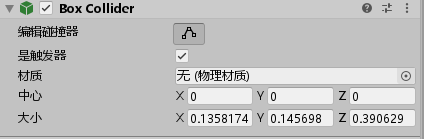


图4-5 子弹预制体盒型碰撞器设置图

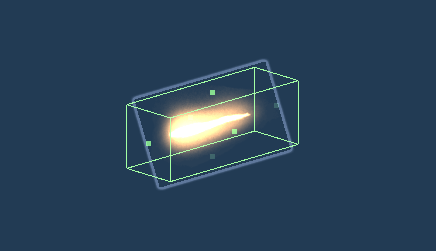


图4-6 子弹预制体盒型碰撞器效果图

## 4.4 游戏玩法设计

游戏设定的基本游戏玩法为通过使用卡牌的方式使用道具，在地图中放置人物单位，用鼠标左键框选放置的人物单位，给已框选的人物单位下达指令完成指定的任务。在挑战关卡过程中可查看当前关卡任务和游戏提示，并可以随时暂停游戏。在关卡中击杀敌人将会获得金币，但战斗过程中获得的金币只在关卡胜利后结算。

每一关都有相应的关卡任务，玩家在还有己方单位存活或拥有道具的情况下完成所有任务即为关卡胜利，继续挑战下一关，反之则为关卡失败，将要重新开始游戏。游戏的关卡挑战均有以下几个基本功能：

1. 使用卡牌，鼠标框选单位，下达移动，攻击等指令。玩家可以将在游戏商店购买的道具在关卡中以卡牌的方式使用，并可使用鼠标框选使用卡牌创建出的游戏单位。玩家可按快捷键Q切换至移动指令，在目标位置点击鼠标右键，选中的游戏单位将会执行自身移动指令，游戏中枪手的移动指令为自身移动至目标地点。按快捷键W切换至攻击指令，在目标位置点击鼠标右键，选中的游戏单位将会执行自身攻击指令，游戏中枪手的攻击指令为攻击攻击范围内距离最近的敌人，如果攻击范围内没有敌人，则自身移动至目标地点，移动过程中攻击范围出现敌人，将会暂时停止移动，攻击攻击范围内距离最近的敌人，敌人消失在攻击范围后恢复移动至目标地点。游戏单位控制流程如图4-7所示。

N

N

攻击敌人

移动至

目标地点

Y

是否有可攻击敌人

结束

使用卡牌

选择单位

移动指令

攻击指令

下达指令

移动至

目标地点

到达

目标地点

图4-7 游戏单位控制流程

1. AI敌人。游戏人工智能是为了提供游戏中的智能体，增加与玩家的交互和玩家的游戏体验，提高游戏开发的效率[12]。设计者可根据需要在关卡中任意放置若干个AI敌人，AI在游戏中有着至关重要的地位，AI的表现和给玩家的反馈直接影响到了玩家的游戏体验。游戏AI并不是越强越好，需要根据游戏需求来设定AI的强度[13]。只有合适的难度才会让玩家有挑战的动力和胜利的喜悦。本文设计的AI从视野距离，视野的角度大小，追击敌人的距离，攻击伤害，攻击精准度等多方面调整AI的难度，尽可能地让AI接近大部分玩家的水平。AI具有多种行为状态，包括：静止，巡逻，攻击，追击，死亡这几种行为状态。当AI的视野范围内没有敌人时则在地图规定范围内巡逻，在到达巡逻点时切换至静止状态，静止期间停止移动，过了设定好了静止时间则再次进入巡逻状态，如果视野范围内出现敌人则切换至攻击状态，在敌人离开攻击范围并在追击范围内则开始追击敌人，在自身位置超过指定移动范围后将切换追击状态为巡逻状态，继续巡逻，当AI血量小于等于0时，进入死亡状态。

AI静止状态持续时间，巡逻点位置与巡逻范围由继承了MonoBehaviour的PointInfo类中的字段决定。通过将PointInfo类作为Unity组件挂载在游戏空物体上，这样每个挂载了PointInfo类的游戏空物体都会带有一个PointInfo的实例，将场景中游戏物体的PointInfo实例放入AIController的PointInfo数组中，循环读取数组中的PointInfo实例决定到达巡逻顺序，巡逻点位置，到达巡逻点后静止持续时间。AIController中还有一个PointInfo类型字段，作为巡逻中心点，当超出PointInfo实例中radius字段的距离时，则返回巡逻状态，寻找下一个巡逻点。PointInfo类和AIController类内容如下：

PointInfo类：

public class PointInfo : MonoBehaviour{ //继承了MonoBehaviour类才可挂载在游戏物体上

[HideInInspector]

public Vector3 pos; //巡逻点向量

public int realTime; //到达巡逻点后静止状态持续时间

public float radius; //巡逻范围

private void Awake() {

pos = transform.position; //在游戏物体被激活时将游戏物体位置信息赋值给PointInfo

}

}

AIController类：

public class AIController : MonoBehaviour

{

public PointInfo centerPoint; //巡逻中心点

public PointInfo[] points; //将要巡逻的点

public GameObject target; //攻击目标

public float pursuitDistance; //追击距离

private UnitBase ub; //由AI控制的游戏单位

int pointsIndedx; //当前巡逻点数组索引

private float time = 0; //刷新间隔

private void Start() {}

private void Update() {}

public void setTarget(UnitBase ub) {}

}

AI追击目标通过给AI控制的游戏单位下达攻击指令，将追击的游戏单位位置作为攻击位置即可。

AI脚本通过状态判断来决定AI的执行行为，如图4-8所示。

状态判断

区域内移动

静止

巡逻

攻击

停止移动

射击

跟随敌人

追击

死亡

死亡动画

图4-8 AI状态判断

1. 任务查看。挑战关卡中的玩家可以通过任务面板查看当前关卡的任务数量和任务说明。任务可通过游戏配置表配置。任务配置表类内容如下：

[CreateAssetMenu(fileName = "TasksConfig", menuName = "Configs/TasksConfig")]

public class TasksConfig : ScriptableObject {

public List<TaskItem> cfs = new List<TaskItem>(); //总任务列表

[System.Serializable]

public class TaskItem {

public long id; //任务id

public string name; //任务标题

[TextArea]

public string desc; //任务说明

public int priority; //任务优先级

public long needItemId; //任务所需媒介道具id

public int needItemNum; //任务所需媒介道具数量

public int group; //任务所属关卡id

}

}

1. 游戏结算。关卡挑战完成后将弹出关卡结算界面，显示关卡挑战结果。胜利结算界面展示玩家在此关卡中获得的奖励，且有退出和下一关两个按钮，点击退出按钮则返回主菜单，点击下一关则进入游戏商店。而失败结算界面仅有一个退出按钮，功能与胜利结算界面的退出按钮相同。在关卡中击杀敌人获得的奖励道具和媒介道具可通过游戏配置表配置。掉落道具配置表类内容如下：

[System.Serializable]

public struct RewardItem {

public long itemId; //道具id

public int itemNum; //道具数量

}

[CreateAssetMenu(fileName = "RewardFallConfig", menuName = "Configs/RewardFallConfig")]

public class RewardFallConfig : ScriptableObject {

public List<RewardFallItem> cfs = new List<RewardFallItem>();//掉落道具列表

[System.Serializable]

public class RewardFallItem {

public long id; //掉落id

public string remark;

public List<RewardItem> rewardIds; //掉落道具信息

public long stageId; //掉落关卡

public long unitId; //掉落道具的游戏单位id

}

}

1. 暂停游戏。玩家在挑战关卡时可以随时暂停游戏并且可以随时取消暂停，在暂停时可点击退出按钮返回主界面，游戏将不计入本次关卡挑战所产生的游戏数据变化。
2. 游戏提示。玩家在挑战关卡时可点击提示按钮查看设计者给出的游戏提示。让玩家更快地熟悉游戏内容。游戏提示可通过游戏配置表配置。游戏提示配置表类内容如下：

[CreateAssetMenu(fileName = "TipsConfig", menuName = "Configs/TipsConfig")]

public class TipsConfig : ScriptableObject {

public List<TipsItem> cfs = new List<TipsItem>(); //游戏提示列表

[System.Serializable]

public class TipsItem {

public long id; //游戏提示id

public string title; //游戏提示标题

[TextArea]

public string desc; //游戏提示描述

public int priority; //游戏提示优先级

}

}

## 4.5 本章小结

本章对客户端进行了详细地设计，包括主菜单的界面布局，各个界面的切换，游戏的一些基本操作方式和玩法以及游戏中的一些其他功能。详细介绍了每一个功能模块的设计。

# 5 游戏实现

## 5.1 游戏主菜单界面的实现

游戏主菜单的主要功能是为了玩家可以自由选择游戏开始的方式或者选择关闭游戏。玩家可以选择重新开始新的游戏，也可以选择继续之前的游戏。核心功能为玩家游戏数据的处理和UI界面的切换。在Unity中创建一个场景作为UI背景提高整体画面美观程度。使用FairyGUI编辑器来创建UI元素。如图5-1所示。

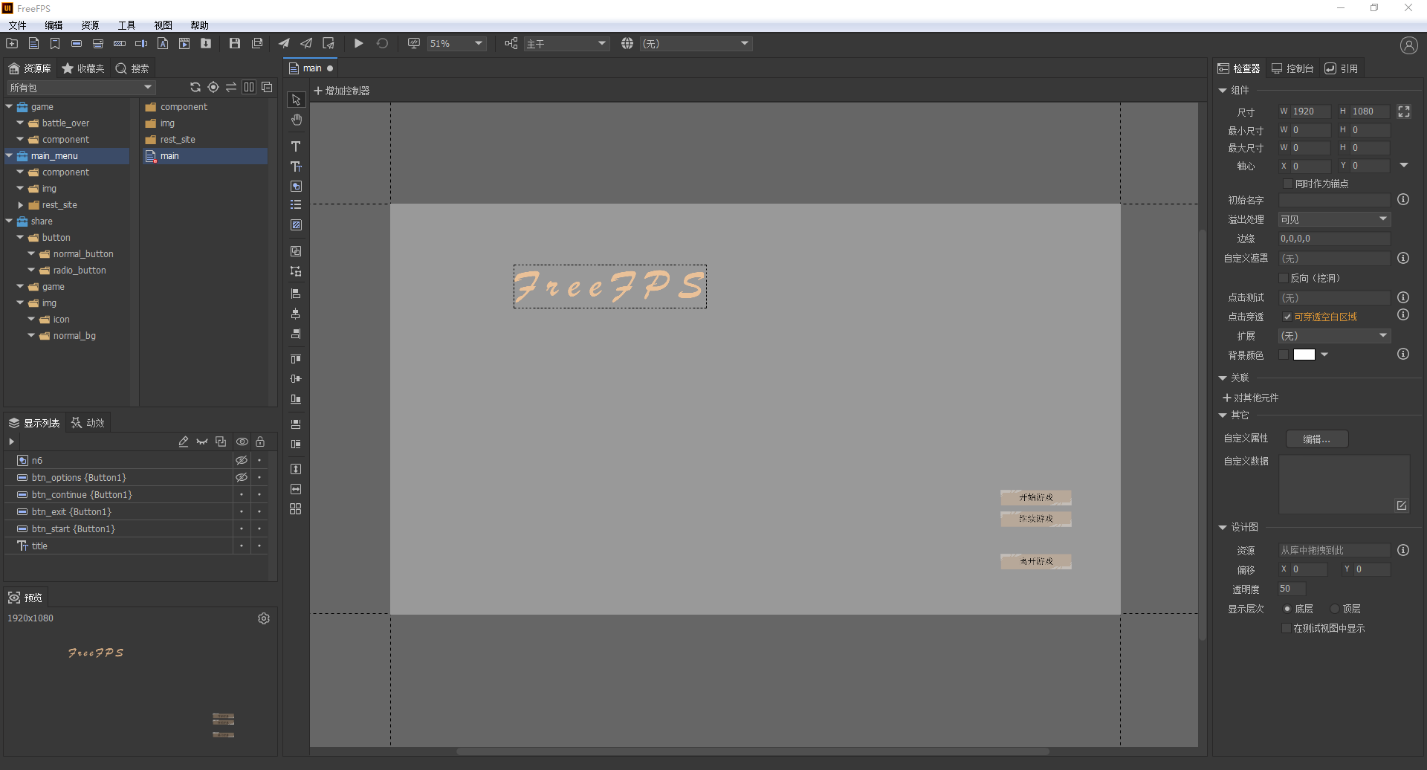


图5-1 FairyGUI编辑器界面

表5-1介绍了FairyGUI的几种常见基础UI元件。

表5-1 FairyGUI基础UI元件表

|  |  |
| --- | --- |
| 元件名称 | 元件介绍 |
| 文本控件 | 用于显示文本或输入文字 |
| 富文本控件 | 在文本控件的基础上支持了富文本 |
| 图形控件 | 可自由创建出简单的图形 |
| 列表控件 | 用于创建多个由基础元件组成的UI组件 |
| 装载器控件 | 将图片，动画等显示出来 |

在设计好UI布局后，需要添加的主要功能为：开始游戏按钮添加重新开始游戏事件；继续游戏按钮添加继续游戏事件；离开游戏按钮添加关闭游戏事件。将组件拖入UI，并在代码中给组件的onClick事件中添加重新开始游戏的函数，游戏运行时在游戏主菜单界面点击“开始游戏”即可重新开始游戏。点击开始游戏按钮进入游戏商店的部分主要代码为：

AddEvent(GetChild("btn\_start"), "onClick", (e) => {//给名为“btn\_start”的UI组件添加点击事件

PlayerModel.Instance.initPlayerData(); //重置玩家游戏数据

UIHandler. EnterRestMenu (); //打开游戏商店界面

close(); //关闭当前界面

});

继续游戏按钮、离开游戏按钮制作方法与开始游戏按钮制作方法相同，给每个按钮添加相对应功能的函数即可，就不再一一介绍了。

## 5.2 游戏商店的实现

游戏商店的主要功能是展示商店售卖的道具和玩家拥有的道具，让玩家可以查看自身金币数量，并通过金币购买或出售游戏道具。同时玩家可以点击游戏道具查看道具详情。可在此界面选择返回主菜单界面或进入关卡挑战。制作过程依然是使用FairyGUI编辑器编辑UI界面。游戏商店实际效果如图5-2所示。



图5-2 游戏商店实际运行效果图

游戏商店界面通过读取道具配置表和玩家游戏数据显示游戏商店现有售卖道具和玩家拥有的道具，游戏商店界面显示的部分主要代码为：

initPayPanel(); //初始化游戏商店道具面板

updatePayLst(LstMode.PayCharacter); //更新游戏商店道具面板

initSelfPanel(); //初始化玩家道具面板

updateSelfLst(LstMode.SelfCharacter); //更新玩家道具面板

void initPayPanel() {

var payPanel = GetChild("panel\_pay\_card"); //获得UI组件

var leftBtn = payPanel.getChild("btn\_left");

leftBtn.setText("征兵"); //设置按钮组件文本

AddEvent(leftBtn, "onClick", (e) => updatePayLst(LstMode.PayCharacter));//添加按钮点击事件

var rightBtn = payPanel.getChild("btn\_right");

rightBtn.setText("额外");

AddEvent(rightBtn, "onClick", (e) => updatePayLst(LstMode.PayProp));

}

void updatePayLst(LstMode mode) {

var gLst = GetChild("panel\_pay\_card", "list\_card").asList; //获得UI列表元件

gLst.RemoveChildrenToPool(); //清空列表

payBtnItems.Clear();

var allCard = new List<ItemConfig.Item>(); //获取道具配置表的所有道具信息

if(mode == LstMode.PayCharacter) {

//挑选相对应类型的道具信息

allCard = ConfigManager.Instance.getItemConfigs().select((x) => {

return x.getIntAttr("card\_sec\_type") == 1;

});

} else if(mode == LstMode.PayProp) {

allCard = ConfigManager.Instance.getItemConfigs().select((x) => {

return x.getIntAttr("card\_sec\_type") == 2;

});

}

for(int i = 0; i < allCard.Count; i++) { //将相应的道具信息显示在UI上

var item = gLst.AddItemFromPool().asButton;

var card = allCard[i];

var shop\_num = card.getIntAttr("shop\_num");

item.getChild("title").setText(card.name);

item.getChild("price").setText(card.getIntAttr("price") \* shop\_num);

item.setIcon(card.icon);

payBtnItems.Add(item);

AddEvent(item, "onClick", (e) => {

//记录选中的道具信息

var it = new User.bagItem() { itemId = card.id, itemNum = shop\_num };

opr = new DealOpration() { dealType = DealOpration.DealType.Purchase, item = it };

ClearSelect(); //将过去选中的按钮组件设置为未选中

item.selected = true; //将现在的按钮组件设置为选中

updateConfirmBtn(); //更新购买按钮显示

updateMainInfo(); //更新道具详细信息显示

});

}

}

游戏商店显示玩家当前金币数量，当玩家进行购买或出售操作时金币将随之变动，玩家金币数量显示的部分主要代码为：

UIHandler.UpdateCurrencyGList(GetChild("list\_currency").asList);

玩家在拥有足够金币时可点击商店道具再点击购买按钮进行道具的购买，在玩家拥有道具时也可点击玩家拥有的道具再点击出售按钮进行道具的出售。游戏道具的购买与出售的部分主要代码为：

AddEvent(GetChild("btn\_confirm"), "onClick", (e) => {

if (opr == null) return;

var cf = ConfigManager.Instance.getItemConfig(opr.item.itemId);

if(opr.dealType == DealOpration.DealType.Purchase) { //购买

if (!canPay(cf)) return; //不可购买时点击无效

var items = new Dictionary<long, int>();

items[opr.item.itemId] = cf.getIntAttr("shop\_num");

PlayerModel.Instance.addBagItem(items); //将道具添加到玩家背包中

PlayerModel.Instance.allBag.Find((p)=>p.itemId==DataConst.IT\_GOLD).itemNum-= cf.getIntAttr("price"); //减少玩家金币

}

else if(opr.dealType == DealOpration.DealType.Sell) { //出售

var addGold = new Dictionary<long, int>();

addGold.Add(DataConst.IT\_GOLD, cf.getIntAttr("price") \* opr.item.itemNum);

PlayerModel.Instance.addBagItem(addGold); //增加玩家金币

var reduceItem = new Dictionary<long, int>();

reduceItem[opr.item.itemId] = opr.item.itemNum;

PlayerModel.Instance.removeBagItem(reduceItem); //减少玩家道具

opr.dealType = DealOpration.DealType.None;

}

updateConfirmBtn(); //更新购买出售按钮

updateSelfLst(curLstMode); //更新玩家道具列表

UIHandler.UpdateCurrencyGList(GetChild("list\_currency").asList);//更新玩家拥有金币数量显示

});

## 5.3 游戏单位的实现

玩家和AI都可以控制各自的游戏单位，将游戏单位做成预制体，再通过游戏单位的预制体生成到所需场景中。游戏单位预制体实际效果如图5-3所示。

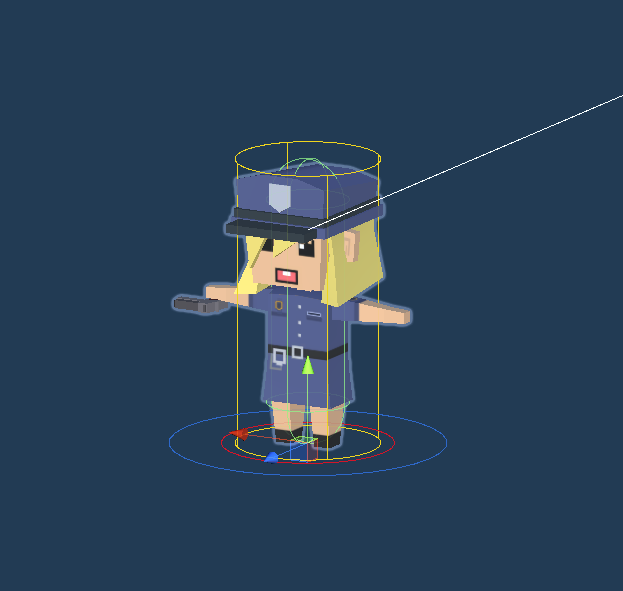


图5-3 游戏单位预制体效果图

游戏单位的移动主要是通过A\* Pathfinding Project Pro插件完成，而目前现有游戏单位的攻击方式则是通过使用实现了IWeapon接口的WeaponAK类去创建并发射子弹实现游戏单位的攻击，并为子弹设置了发射角度的随机偏移，让游戏更加生动。游戏单位移动和攻击的部分主要代码为：

游戏单位的移动：

public virtual void MoveToDestination() {

…

ap.isStopped = false; //开启移动

ap.destination = currentOpration.pos; //设置移动坐标

…

SetMoveFromSecType(); //设置移动速度和移动动画

}

游戏单位的攻击：

public class WeaponAK : IWeapon{ //继承IWeapon接口

UnitBase owner; //武器所属于的游戏单位

int bulletId; //子弹预制体id

public float rate;

public float Rate {get {return rate;}set {rate = value;}} //攻击的速度

float force; //子弹速度

float harm; //子弹伤害

LayerMask targetMask; //目标碰撞层级

LayerMask obstacleMask; //障碍物碰撞层级

Vector2 offset1; //子弹发射角度随机偏移最小值

Vector2 offset2; //子弹发射角度随机偏移最大值

public void UseOnly() {

var \_pos = owner.bulletSlot.position; //子弹初始位置

//子弹方向

Vector3 direction = owner.minTarget.transform.position - owner.bulletSlot.transform.position;

var \_dir = new Vector3(direction.x, owner.transform.forward.y, direction.z).normalized;

Vector3 offset = new Vector3(Random.Range(offset1.x, offset2.x), Random.Range(offset1.y, offset2.y), 0); //随机子弹发射角度

GameObject bul = PoolManager.Instance.AddGOFromPool(bulletId, \_pos, Quaternion.Euler(offset)); //生成子弹游戏物体

bul.transform.forward = Quaternion.Euler(offset) \* \_dir.normalized;

Bullet bs = bul.GetComponent<Bullet>(); //获取子弹游戏物体的Bullet子弹类实例

//设置子弹

bs.owner = owner;

bs.harm = harm;

bs.targetMask = targetMask;

bs.obstacleMask = obstacleMask;

Rigidbody rigi = bul.GetComponent<Rigidbody>();

if (rigi != null) {

rigi.velocity = Vector3.zero;

//为子弹添加移动向量

rigi.AddForce(Quaternion.Euler(offset) \* \_dir.normalized \* force, ForceMode.Impulse);

}

}

}

## 5.3 游戏玩法的实现

1. 使用卡牌，鼠标框选单位，下达移动，攻击等指令。玩家可通过使用卡牌在指定位置放置人物单位，并可框选已放置的己方人物单位下达移动，攻击等指令，实际效果如图5-4和图5-5所示。



图5-4 使用卡牌效果图

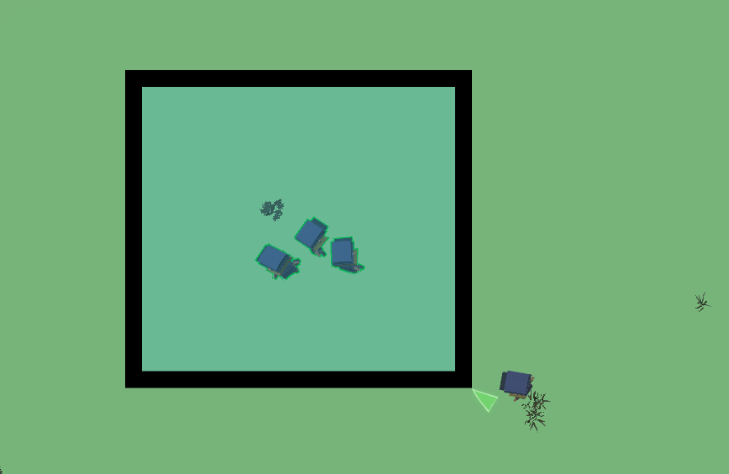


图5-5 框选单位效果图

卡牌使用时箭头指向的UI效果是通过配合Unity插件2DSpriteShare根据鼠标起始位置和鼠标当前位置平铺变形图片实现。使用卡牌是通过道具配置表中字符串类型的useFunc字段反射相对应的函数，执行反射的函数，实现可配置的卡牌的使用。放置人物单位本质上是在指定位置生成指定的物体，反射和人物单位放置的部分主要代码为：

反射的代码：

public bool ExeCmdByStr(string cmds) {

Type ins = Type.GetType("DynScript.DynamicScript"); //反射函数的命名空间和类名

BindingFlags flag = BindingFlags.Public | BindingFlags.Instance; //反射函数的类型

var cmdsArr = cmds.Split(','); //拆分字符串

Regex rgx = new Regex("\\s+");

if (cmdsArr.Length <= 1 && string.IsNullOrEmpty(cmdsArr[0])) cmdsArr = new string[0];

int results = 0;

for(int i = 0; i < cmdsArr.Length; i++) { //执行多个字符串方法

var rawCmd = rgx.Replace(cmdsArr[i], "");

var cmd = Regex.Match(rawCmd, @"\w+(?=\()").ToString(); //从字符串中取出方法名

//从字符串中取出参数

var paras = Regex.Match(rawCmd, @"(?<=\()\w+(?=\))").ToString().Split(',');

if (paras.Length <= 1 && string.IsNullOrEmpty(paras[0])) paras = new string[0];

Type[] pattern = new Type[paras.Length];

for(int j = 0; j < paras.Length; j++) { //解析字符串方法参数个数

pattern[i] = typeof(string);

}

var method = ins.GetMethod(cmd, pattern); //反射对应函数

results += (int)method.Invoke(Instance, flag, Type.DefaultBinder, paras, null);//执行反射函数

}

if (results == cmdsArr.Length) return true; //返回执行结果

return false;

}

放置人物单位的代码：

public int CreatObj(string objId, string isLimit) {

var pos = InputManager.Instance.mousePos; //读取鼠标位置

var cam = CameraControl.Instance.cam;

Ray ray = cam.ScreenPointToRay(pos); //通过屏幕鼠标位置生成跟屏幕垂直的射线

if (!Physics.Raycast(ray, out RaycastHit hitInfo, cam.farClipPlane)) return 0;

if (isLimit.boolSafe(true) && !hitInfo.collider.gameObject.CompareTag("LayPanel")) return 0;

UnitManager.Instance.addUnit(objId.longSafe(), hitInfo.point, Quaternion.identity);//指定位置生成物体

return 1;

}

鼠标框选单位是通过将场景中己方单位在世界空间的坐标转换为屏幕空间坐标，通过对比己方单位的屏幕空间坐标和鼠标在屏幕空间的坐标确定被玩家框选的己方单位。而UI的显示部分通过变换一个FairyGUI图形元件的位置与大小实现选择框的显示。鼠标框选单位的部分主要代码为：

public void BeginSelectUnit() { //按下鼠标左键的事件

//获取己方单位在屏幕空间的位置

playerUnits=UnitManager.Instance.getUnitsByUnitType<GameObject>(UnitType.PlayerSceneUnit);

unitsToScreen = new Vector3[playerUnits.Count];

startPos = mousePos; //记录按下鼠标左键的起始位置

selectUnits.Clear(); //取消选中已选中的单位

isdDragging = true;

}

void SelectUnit(){

if (left\_click && isdDragging){ //按住鼠标左键并且游戏摄像机可移动

endPos = mousePos; //记录鼠标的位置

for (int i = 0; i < playerUnits.Count; i++){ //筛选鼠标框选范围内的己方单位

unitsToScreen[i] = cam.WorldToScreenPoint(playerUnits[i].transform.position);

var unit = unitsToScreen[i];

if (playerUnits[i].activeSelf == false) continue;

if (unit.x < Mathf.Min(startPos.x, endPos.x) || unit.x > Mathf.Max(startPos.x, endPos.x)) continue;

if (unit.y < Mathf.Min(startPos.y, endPos.y) || unit.y > Mathf.Max(startPos.y, endPos.y)) continue;

selectUnits.Add(playerUnits[i]);

}

var ie = playerUnits.GetEnumerator();

while (ie.MoveNext()) {

var go = ie.Current;

Outline outLine = go.GetComponent<Outline>();

if (outLine == null) continue;

outLine.enabled = selectUnits.Contains(go);//给被框选的己方单位加轮廓描边效果

}

}

}

public void EndSelectUnit() { //松开鼠标左键的事件

isdDragging = false;

}

给人物单位下达移动，攻击等指令则是通过创建一个游戏单位基类UnitBase，拥有update方法和移动，攻击等虚方法，update方法由UnitManager类单例调用，用于更新当前状态和执行当前状态相应的行为，再创建一个RifleTroop类重写移动，攻击等虚方法再将脚本附加至游戏物体上，通过调用UnitBase基类的SetcurrentOpration方法切换当前状态实现给人物下达移动，攻击等指令。寻路移动则是通过调用A\* Pathfinding Project Pro插件的API接口实现。UnitBase类和RifleTroop类部分主要代码为：

UnitBase类部分主要代码：

public virtual void update() {

BehaviorTree(); //执行当前单位行为树

}

void BehaviorTree() {

if (CurrentHealth <= 0) return; //死亡时则不执行任何行为

switch (currentOpration.oprationType) { //根据当前状态执行不同的行为

case OprationType.None:

NoneBehavior();break; //静止

case OprationType.OnlyMove:

MoveToDestination();break; //移动

case OprationType.MoveAttack:

MoveAttackToDestination();break; //攻击

…

}

}

public virtual void NoneBehavior() {

AnimaNone(); //停止动画

ap.isStopped = true; //关闭寻路

minTarget = null; //取消目标

}

public virtual void MoveToDestination() {

minTarget = null;

ap.isStopped = true;

ap.isStopped = false; //开启寻路

ap.destination = currentOpration.pos; //切换状态

…

SetMoveFromSecType(); //移动的方式是走还是跑

}

public virtual void MoveAttackToDestination() {

minTarget = FindShortTarget(); //寻找目标

AttackTarget(); //攻击目标

}

RifleTroop类部分主要代码：

public override void AttackTarget() {

…

if (animator.GetCurrentAnimatorStateInfo(0).IsName("Attack")) {

animator.speed = weapon.Rate; //让动画的播放速度适配人物的攻击速度

}

bool isAttack = minTarget != null; //如果有目标则进入攻击状态

ap.isStopped = isAttack; //攻击时不可移动

animator.SetBool("isFire", isAttack); //在攻击时播放攻击动画

if (weaponSlot != null) weaponSlot.SetActive(isAttack);//在攻击时显示武器

if (!isAttack) {

if (col != null) col.isTrigger = true;

ap.destination = currentOpration.pos; //在无目标时则移动至目标地点

SetMoveFromSecType();

} else {

Vector3 direction = minTarget.transform.position - transform.position;

transform.forward = new Vector3(direction.x, transform.forward.y, direction.z).normalized;

}

…

}

1. AI敌人。AI本质上也是游戏单位，不同的是AI不由玩家操控而是遵循开发者设计的程序行动。通过更改游戏配置表，程序再读取游戏配置表中的数据，实现配置AI的攻击伤害和攻击精准度等属性。AI的视野角度大小，视野距离可通过游戏物体中的UnitBase类序列化公开参数viewAngle和findEnemyDistance进行调整，实际效果如图5-6和图5-7所示。



图5-6 调整视野角度及视野距离参数效果图



图5-7 调整视野范围及角度后的实际视野效果图

通过视野寻找目标，实际上是对比目标和人物自身的距离和角度判定目标是否在视野范围内。UnitBase类寻找最近距离目标的部分主要代码为：

public virtual GameObject FindShortTarget() {

Transform shortTarget=null;

Collider[]targetsInViewRadius=Physics.OverlapSphere(transform.position,findEnemyDistance, targetMask); //获得距离内所有目标

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

if (targetsInViewRadius[i].GetComponent<IHealth>().CurrentHealth <= 0) continue;//已死亡

Transform target = targetsInViewRadius[i].transform;

Vector3 dirToTarget=(target.position-transform.position).normalized;//判断目标和自身的角度

if (Vector3.Angle(transform.forward, dirToTarget) > viewAngle / 2) continue;//不在视野范围

float dstTotarget = Vector3.Distance(transform.position, target.position);

if (Physics.Raycast(transform.position + (Vector3.up \* (ap.height - ap.radius)), dirToTarget, dstTotarget, obstacleMask)) continue; //有障碍物

if(shortTarget==null || dstTotarget < Vector3.Distance(shortTarget.position, transform.position)){

shortTarget = target; //选择距离最近的目标

}

}

if (shortTarget != null) return shortTarget.gameObject; //返回寻找到的最近目标

return null;

}

创建AIController类附加至人物的游戏物体中，通过AIController类控制人物的状态和行为完成AI的制作。AIController类的部分主要代码为：

private void Update() {

if(target != null) {

//在自身处于活动范围外或目标死亡则回到巡逻状态

if (Vector3.Distance(transform.position, centerPoint.pos) > centerPoint.radius) target = null;

if (!target.activeSelf || target.GetComponent<IHealth>().CurrentHealth <= 0) target = null;

}

var destination = Vector3.zero;

SecOprationType secOpr;

GameObject g = null;

if (time > ub.findEnemyInterval) {

ub.FindShortTarget(); //寻找目标

time -= ub.findEnemyInterval;

}

if (target == null && g != null && target != g) target = g;

//如果目标在离开攻击范围但在追击的范围内则追击目标

if(target!=null&&Vector3.Distance(target.transform.position, transform.position) < pursuitDistance){

destination = target.transform.position;

secOpr = SecOprationType.Run; //追击目标的移动方式用跑的

} else {

//否则处于巡逻状态

target = null;

destination = points[pointsIndedx].pos;

secOpr = SecOprationType.Walk; //巡逻的移动方式是用走的

}

OprationData od = new OprationData() {

oprationType = OprationType.MoveAttack,

secOprationType = secOpr,

pos = destination

};

ub.SetcurrentOpration(od); //切换至相应的状态

if (ub.ap.reachedEndOfPath) { //到达巡逻点静止一段时间

var t = ub.timer.Find((timer) => { return timer.name == "stopDelay"; });

t.interval = points[pointsIndedx].realTime;

t.rTime += Time.deltaTime;

if (t.rTime > t.interval) { //静止时间结束后前往下一个巡逻点

t.rTime = 0;

pointsIndedx++;

if (pointsIndedx >= points.Length) pointsIndedx = 0;

}

}}

1. 任务查看。只有玩家明白当前任务是什么才能更好地做出决策，这也是策略游戏的要素之一。玩家可以在关卡挑战时点击任务按钮查看当前关卡任务，再次点击任务按钮则隐藏当前关卡任务。实际效果如图5-8所示。

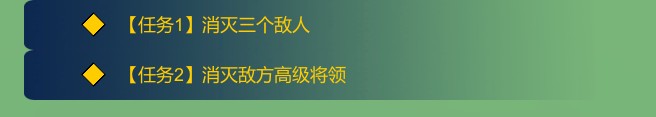


图5-8 关卡任务效果图

关卡任务的显示是先读取游戏配置表数据再通过FairyGUI的列表元件显示出来，然后给任务按钮添加移动列表元件位置的事件，将列表元件移入或移出屏幕显示范围达到显示和隐藏的效果。显示关卡任务的部分主要代码为：

void InitTasksGList() {

…

//根据当前关卡ID获取相应的游戏任务配置表数据

var tasks = ConfigManager.Instance.getTasksByStageId(ScenesManager.Instance.curStageId);

tasks.Sort((a, b) => -a.priority.CompareTo(b.priority));

for(int i = 0; i < tasks.Count; i++) { //将每个任务添加至列表元件中

var task = tasks[i];

var item = glst.AddItemFromPool();

item.getChild("title").setText(string.Format("{0}{1}", task.name, task.desc));

}

glst.ResizeToFit(); //调整列表元件显示大小

var lstHeight = glst.actualHeight;

AddEvent(GetChild("btn\_task"), "onClick", (e) => { //添加任务按钮点击事件

showTasks = !showTasks;

if (showTasks) {

//移动列表元件的位置至屏幕显示范围内

glst.TweenMoveY(lstHeight, tasksGListAnimaTime).SetEase(EaseType.SineOut);

} else {

//否则就将列表元件的位置移动至屏幕显示范围外

glst.TweenMoveY(0, tasksGListAnimaTime).SetEase(EaseType.SineOut);

}

});

}

1. 游戏结算。关卡挑战胜利或失败后更新玩家游戏数据并显示游戏结算界面，显示当前关卡挑战结果，游戏结算界面还有一个退出按钮，能够退出至游戏主菜单界面，胜利结算界面额外显示当前关卡获得的奖励并且还额外多一个下一关按钮，玩家能够直接点击下一关按钮进入游戏商店界面进行关卡挑战前的准备。实际效果如图5-9和图5-10所示。



图5-9 胜利结算界面



图5-10 失败结算界面

击杀敌人将会掉落金币和任务媒介道具，如果掉落的任务媒介道具数量大于等于当前关卡任务完成所需的任务媒介道具数量则算任务完成，当玩家还有己方单位存活或拥有道具的同时完成了所有任务则关卡胜利，更新玩家数据，显示胜利结算界面，反之则关卡失败，重置玩家数据，显示失败结算界面。游戏结算的部分主要代码为：

public void tryEndBattle() {

//失败

if (UnitManager.Instance.getUnitsByUnitType<GameObject>(UnitType.PlayerSceneUnit).Count <= 0 && BattleView.allCard.Count <= 0) { //如果场上没有有效玩家单位并且未拥有道具

ScenesManager.Instance.isWin = false; //记录关卡挑战结果

PlayerModel.Instance.initPlayerData(); //重置玩家数据

UIHandler.OpenUI(DataConst.UIT\_BATTLE\_OVER, null);//弹出游戏结算界面

enabled = false; //停止关卡

CameraControl.Instance.CloseDragCam(); //停止摄像机移动

}

//胜利

int finishTaskCount = 0; //完成的任务数量

var rewardDic = BattleView.rewardFallDic; //敌人掉落的金币和任务媒介道具

for (int i = 0; i < winTasks.Count; i++) { //遍历所有任务

var task = winTasks[i];

if(rewardDic.ContainsKey(task.needItemId)&&rewardDic[task.needItemId]>=task.needItemNum) finishTaskCount += 1; //掉落媒介道具大于任务所需媒介道具则任务完成

}

if (finishTaskCount >= winTasks.Count) { //当所有任务都完成了则为胜利

PlayerModel.Instance.addBagItem(rewardDic.select((x)=>{returnConfigManager.Instance.getItemConfig(x.Key).canInBag; })); //添加关卡奖励至玩家背包

var allCard = BattleView.allCard.select((p) => p.itemNum > 0);

PlayerModel.Instance.removeAllCard(); //扣除玩家关卡中使用的道具

var items = new Dictionary<long, int>();

for(int i = 0;i< allCard.Count; i++) {

if (items.ContainsKey(allCard[i].itemId)) {

items[allCard[i].itemId] += allCard[i].itemNum;

} else {

items[allCard[i].itemId] = allCard[i].itemNum;

}

}

PlayerModel.Instance.addBagItem(items);

ScenesManager.Instance.finishStage(); //关卡变化

ScenesManager.Instance.isWin = true; //根据记录的关卡挑战结果显示相应的结算界面

UIGame.Instance.UpdateCurrency(); //更新显示当前持有的金币数量

UIHandler.OpenUI(DataConst.UIT\_BATTLE\_OVER, null);

enabled = false;

CameraControl.Instance.CloseDragCam();

}

}

1. 暂停游戏。关卡挑战中可点击暂停按钮暂停游戏并弹出游戏暂停界面。玩家可在游戏暂停界面选择点击退出按钮返回主菜单，也可选择点击继续按钮，继续挑战关卡。实际效果如图5-11所示。



图5-11 游戏暂停界面

通过给暂停按钮添加点击事件，在点击时将游戏时间速度设置为0，从而达到游戏被暂停的效果，同时通过切换FairyGUI组件的控制器索引将给继续按钮和退出按钮添加了相应点击事件的游戏暂停界面显示出来，就完成了暂停游戏的功能。暂停游戏部分主要代码为：

AddEvent(GetChild("btn\_pause"), "onClick", (e) => { //添加暂停按钮点击事件

Time.timeScale = 0; //将游戏时间速度设置为0

GetCtrl("pause").SetSelectedIndex(1); //切换FairyGUI组件控制器索引

});

AddEvent(GetChild("btn\_continue"), "onClick", (e) => { //添加继续按钮点击事件

GetCtrl("pause").SetSelectedIndex(0); //恢复FairyGUI组件控制器索引

Time.timeScale = 1; //恢复游戏时间速度

});

AddEvent(GetChild("btn\_exit"), "onClick", (e) => { //添加退出按钮点击事件

GetCtrl("pause").SetSelectedIndex(0);

Time.timeScale = 1;

Launcher.Instance.MainMenu(); //切换到主菜单界面

});

1. 游戏提示。在玩家点击提示按钮的时候将会显示游戏提示界面，可左右切换提示信息，可再次点击提示按钮隐藏游戏提示界面。实际效果如图5-12所示。

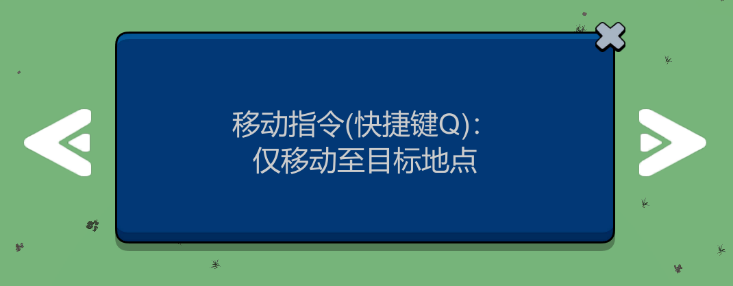


图5-12 游戏提示界面

显示和隐藏的实现方式跟游戏暂停界面相同，同样是切换FairyGUI的组件控制器索引达到显示和隐藏游戏提示界面的效果。游戏提示的内容则是在进入关卡时读取游戏配置表中的数据显示在FairyGUI的列表上，通过列表的滚动功能切换提示内容。游戏提示部分主要代码为：

void InitTipsGList() {

var tipCtrl = GetCtrl("show\_tip");

AddEvent(GetChild("btn\_tip"), "onClick", (e) => { //添加提示按钮点击事件

showTips = !showTips;

tipCtrl.SetSelectedIndex(showTips ? 1 : 0); //切换FairyGUI组件控制器索引

});

tips.Sort((a, b) => { return -a.priority.CompareTo(b.priority); }); //将提示信息按配置权重排序

var glst = GetChild("tip\_list").asList;

glst.SetVirtualAndLoop(); //设置为循环列表，可使提示信息头尾相连

glst.itemRenderer = TipsItemRenderer; //渲染提示信息内容

glst.numItems = tips.Count; //设置提示信息数量

glst.scrollPane.scrollStep = glst.actualWidth; //设置列表滚动距离

AddEvent(GetChild("tip\_next"), "onClick", (e) => { glst.scrollPane.ScrollRight(); });//向右滚动

AddEvent(GetChild("tip\_pre"), "onClick", (e) => { glst.scrollPane.ScrollLeft(); }); //向左滚动

AddEvent(GetChild("btn\_tip\_close"), "onClick", (e) => {//添加关闭按钮点击事件

showTips = false;

tipCtrl.SetSelectedIndex(0);

});

}

## 5.4 本章小结

本章主要介绍了客户端的一些主要功能的具体实现和UI的制作过程。如游戏主菜单界面，游戏商店界面，AI敌人，暂停游戏，游戏提示等功能的实现，还有游戏基本玩法，判断关卡挑战胜利或者失败等一些游戏内容的实现方式。

# 6 测试结果和分析

## 6.1 游戏运行环境测试

本文游戏开发的主要环境配置包括硬件配置和软件配置。

硬件开发环境：处理器Intel(R)Core(TM)i7-9750H @CPU 2.60GHz 2.60Hz，已安装内存(RAM)：16GB，

显示适配器NVIDIA GeForce GTX 1650，三星SM951 512G固态硬盘。

软件开发环境：64位Windows10教育中文版操作系统，Visual Studio 2019，Unity引擎2017.4.0f1。

将编译完成的游戏运行在不同的环境下，测试游戏是否能够正常运行。测试结果如表6-1所示。

表6-1 环境测试表

|  |  |  |
| --- | --- | --- |
| 系统环境 | 硬件 | 测试结果 |
| Windows7 | Intel(R)i5-4460处理器，8GB内存，显示适配器：GT730，512GB硬盘 | 正常运行 |
| Windows8 | Intel(R)i7-8750H处理器，16GB内存，显示适配器：GTX1060，256GB固态硬盘 | 正常运行 |
| Windows10 | Intel(R)i7-9750H处理器，16GB内存，显示适配器：GTX1650，512GB固态硬盘 | 正常运行 |

## 6.2 游戏基本功能测试

游戏功能测试对于游戏测试来说是最为重要的一个环节，测试的结果直接影响到了游戏是否能够正常上线。本小节将会根据测试用例判断游戏功能是否达到预期标准。游戏功能测试结果如表6-2所示。

表6-2 游戏功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 功能 | 测试 | 预期 | 测试结果 |
| 启动客户端 | 在不同的系统环境下启动客户端 | 正常运行 | Windows7，Windows8，  Windows10系统下皆可正常运行 |
| 游戏主菜单 | 启动游戏后进入游戏主菜单界面，并测试各个按钮功能 | 正常显示游戏主菜单，各按钮功能正常 | 游戏主菜单正常显示，点击开始游戏，继续游戏，离开游戏按钮功能正常 |
| 游戏商店 | 购买或出售游戏道具，查看游戏道具详情，查看玩家金币数量 | 金币数量足够时可购买游戏商店道具，持有游戏道具时可出售自身拥有的游戏道具，点击游戏道具可显示道具详情，显示当前剩余玩家金币数量 | 游戏商店道具和玩家持有道具显示正常，玩家剩余金币显示正常，点击游戏道具正常显示道具详情，玩家金币足够时可正常购买道具，金币不足时购买按钮置灰不可点击，出售自身拥有的游戏道具获得了相应的金币数量 |
| 挑战关卡 | 进入关卡，使用卡牌，鼠标左键框选玩家人物单位，控制已选中的玩家人物单位 | 玩家可自由使用卡牌，鼠标左键可框选有效玩家人物单位，已选中的玩家人物单位准确执行玩家下达的指令 | 可以正常体验游戏，正常使用卡牌，鼠标左键框选玩家人物单位时，人物单位在选框中且存活时则被选中，反之则不被选中，选中人物正常执行下达的移动，攻击指令 |

表6-2 游戏功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 功能 | 测试 | 预期 | 测试结果 |
| 关卡结算 | 进入关卡，在各种情况下结束战斗 | 有玩家单位存活或拥有道具时，全部任务完成，则关卡胜利，当没有玩家单位存活或未拥有道具时，任务没有全部完成，则关卡失败 | 在有玩家单位存活或拥有道具时，任务全部完成，正常结算游戏奖励，正常显示关卡胜利界面，关卡胜利界面功能正常，在未完成全部任务，玩家单位全部死亡且未拥有道具时，正常显示关卡失败界面，关卡失败界面功能正常 |
| 任务和提示显示 | 进入关卡，查看关卡任务和游戏提示 | 点击任务按钮弹出关卡任务信息界面，点击提示按钮弹出游戏提示界面 | 点击任务按钮正常弹出关卡任务信息界面并显示正常，再次点击任务按钮可正常关闭任务信息界面，点击提示按钮，正常弹出游戏提示界面，游戏提示界面各功能正常，再次点击提示按钮，正常关闭游戏提示界面 |
| 战斗暂停 | 进入关卡，暂停战斗 | 点击暂停按钮，战斗暂停，弹出游戏暂停界面，点击继续按钮可恢复战斗，点击退出按钮回到游戏主菜单界面 | 点击暂停按钮战斗正常暂停并且弹出游戏暂停界面，游戏暂停界面显示正常，点击游戏暂停界面的继续按钮战斗恢复正常，点击游戏暂停界面的退出按钮成功退出至游戏主菜单界面 |

## 6.3 游戏性能测试

游戏性能测试是对游戏执行效率进行的测试。本文所设计的游戏采用了Unity作为游戏引擎，经过了十几年的考验已经相当成熟，各种算法优化已经达到较好的水平。游戏中的程序部分经过优化也已经有了较大的性能改善。美术方面采用了简单又可爱的美术资源，把在美术方面对硬件的要求降到了最低。经过测试表明，在主流的不同平台及硬件下极大部分情况游戏帧率均能保持在每秒60帧以上。对玩家的经济条件要求较低，玩家能够有流畅的游戏体验。因此本文设计实现的游戏性能通过测试。

## 6.4 本章小结

本章主要在不同的硬件和系统环境下对游戏的运行及游戏功能进行测试。已修复所有已知问题。游戏在不同的硬件和操作系统条件下都能正常流畅地运行游戏。游戏符合预期设计的标准。想要上线还需要一些后续的优化调整和功能添加。

# 7 总结与展望

## 7.1 总结

本文主要介绍了目前市面上一款较为成熟且功能丰富强大的游戏引擎——Unity，并设计开发了一款策略游戏。介绍了游戏开发的基本流程。Unity因为其功能丰富强大以及编程语言相对简单等一些原因使Unity成为了目前市面上最流行的游戏开发引擎之一，大幅度提高了游戏开发的效率。通过这款游戏的制作，让我对游戏行业以及游戏开发多了一些认识和了解，提升了自身实际编程开发的能力，也让我更坚定了进入游戏开发行业的想法。现在游戏的品质越来越好，也用上了更为先进的技术，游戏的画质，内容也有了质的提升，技术的更新换代让游戏策划许多天马行空的想法得到实现，让游戏的玩法更加有趣。本文设计开发的策略游戏，拥有完整的游戏流程，但还有许多不足需要我去学习改进。

## 7.2 展望

本文设计实现的策略游戏虽然功能完整，但是想让玩家有个满意的游戏体验还有许多工作要做，游戏关卡和游戏道具单一，容易枯燥乏味，应该添加更多的道具和关卡，让玩家体验更久的游戏。还有虽然游戏数据对外开放但是没有一个可以让普通玩家去改变游戏，创建游戏模组，在原有游戏的基础上自定义游戏功能的编辑器，让玩家去丰富玩家的游戏体验，同时也能节省开发成本，制作游戏模组的编辑器也是很有需要的。游戏中的细节也不够完善需要去改进，给玩家一个更好的游戏体验。游戏中个性有趣的功能较少，需要更多的创意和能力去填补。游戏的玩法略显单一，需要在以后的开发维护工作中进一步完善。

# 参考文献

[1]杜渐.我国网络游戏产业研究[D].对外经济贸易大学,2016.

[2][3]田小军.游戏产业提速催生版权保护新模式[N/OL].中国新闻出版广电报,2021(7).

[4]石民勇.游戏概论[M].第1版.北京:北京传媒大学出版社,2016.

[5]王龙,李韬伟,杨振发.游戏引擎研究与分析[J].软件导刊,2018,17(2).

[6]王瑞雪.基于DirectX9.0的多玩家FPS游戏引擎的研究与应用[D].太原理工大学,2017.

[7]Wu Tyan-Yu.Tsao Chueh-Yung,Sie Cian-Yu,Unity enhances product aesthetics and emotion[J].International Journal Ergonomics,2017,59:92-99.

[8]龚珏.基于Unity3D的移动游戏客户端框架设计与应用[D].华中科技大学,2018.

[9]蒲冬梅.软件可行性分析评审的要点[J].电子技术与软件工程,2018,12(22):54-55.

[10]何謇.基于Unity3D引擎第三人称射击手机游戏场景的设计与实现[D].华南理工大学,2016.

[11]白羽.网络游戏中个体持续使用行为意向影响因素理论模型与实证研究[D].陕西师范大学,2017.

[12]李博.游戏人工智能关键技术的研究[D].上海交通大学,2016.

[13]刘灵均.基于人工智能角色的3D游戏开发设计与实现[D].中国科学院大学,2018.

# 致谢

到此本文也进入尾声，同时我的大学生活也进入了倒计时。还记得大学初来乍到时对各种新奇的事物感到好奇，时光荏苒，转眼即将离开校园的怀抱走入社会。

感谢我的大学校园给了我专注而又轻松的学习环境，能让我快乐地学习。也感谢校园中的每个人，每件事，给我留下了美好的回忆。

感谢每一位教导过我的教师，感谢你们细心的栽培，才有我学业上的不断成长。感谢指导了我的毕业论文的每一位老师，正是你们耐心的批改，才使我的论文更加精致完善。

感谢支持我的亲人，朋友，是你们无私的奉献与陪伴，让我心无旁骛，安心地完成了我的学业。