塔防游戏-征途设计与开发

中文摘要

本次论文主要讨论如何基于Unity3D游戏引擎，设计和开发一款休闲策略类型的2D塔防单机游戏。本文将结合国内游戏产业发展情况，讨论游戏开发需要面临的挑战。比较Unity3D游戏引擎与其他流行的游行引擎存在什么样的优势和缺点。因为本次毕业设计的作品是塔防游戏类型，所以还会着重介绍塔防游戏的玩法和目前国内外知名度比较高的塔防游戏，分析塔防游戏的本质与核心乐趣。

· 本文将分五个章节来介绍本次毕业设计作品的设计开发过程：

第一章节将介绍课题的国内外发展概况、课题背景以及课题研究的目的和意义，包括了塔防游戏目前的知名游戏以及游戏引擎在中国的占用率。

第二章节则是游戏的需求分析，将通过游戏的玩法和内容设计来具体展开本次毕业设计作品是如何策划的。

第三章节是阐述游戏开发工具制作与使用，在此章节中介绍程序中使用到的辅助插件和类库。以及详细说明如何制作游戏数据配置的相关策划使用工具。

第四章节是游戏内容具体实现，该部分也是本文的重点。合格的程序离不开健壮的框架系统，本章将结合设计模式的知识详细介绍UI框架、工厂模式、建造者模式、观察者模式等设计模式是如何在程序中运用的。在该章的第二部分和第三部分将介绍游戏中的场景管理和UI面板处理以及整体玩法逻辑是如何实现的。

第五章节总结开发过程中的学习和领悟。

关键词：Unity3D，游戏开发，塔防游戏

The design and development of Tower Defence Game – Expedition

Abstract

This paper focuses on how to design and develop a 2D tower defense stand-alone game based on the Unity3D game engine. This article will discuss the challenges of game development in light of the development of the domestic game industry. Compare the advantages and disadvantages of the Unity3D game engine with other popular parade engines. Because the work of this graduation design is the type of tower defense game, it will also introduce the gameplay of tower defense games and the towering defense games with high reputation at home and abroad, and analyze the essence and core fun of tower defense games.

· This article will introduce the design and development process of this graduation design work in five chapters:

The first chapter will introduce the development of the topic at home and abroad, the background of the topic, and the purpose and significance of the research. It includes the current well-known games of the tower defense game and the occupancy rate of the game engine in China.

The second chapter is the analysis of the needs of the game. The game design and content design will be used to specifically plan how the graduation design works are planned.

The third chapter is to explain the production and use of game development tools. In this chapter, the auxiliary plug-ins and class libraries used in the program are introduced. And detailed planning tools for how to make game data configurations.

The fourth chapter is the specific implementation of the game content, this part is also the focus of this article. Qualified procedures are inseparable from a robust framework system. This chapter will combine the knowledge of design patterns to detail how UI design, factory mode, builder mode, observer mode and other design patterns are used in the program. In the second and third sections of the chapter, we will introduce the scene management and UI panel processing in the game and how the overall gameplay logic is implemented.

The fifth chapter summarizes the learning and comprehension in the development process.

# 第一章绪论

## 1.1 课题背景

在市面上各类的游戏类型中，游戏黏着度最高的往往是内存占用小、玩法多变的休闲策略类型的游戏。玩家旨在通关整个游戏的所有玩法而不是为了竞技排名或者社交驱动，可以说这类型的游戏的设计会而更加趋近游戏的本质目的而不是用户的情感刺激需求。

目前中国的游戏产业发展迅猛，近几年的销售收入基本是呈线性增长，新的游戏层出不穷。这给游戏开发者带来了巨大的挑战，如何提高开发效率、快速推出游戏产品是挑战的核心。本次课题基于游戏黏着度和游戏开发效率考量，游戏类型选择了休闲策略型游戏中的塔防游戏，在游戏引擎上也选择了近几年来越来越热门的Unity引擎进行开发。

## 1.2国内外发展概况

### 1.2.1塔防游戏

塔防游戏指通过在地图上建造炮塔等相似的建筑物，以阻止游戏中敌人抵达堡垒或者游戏设定的终点的战略游戏。塔防类游戏的核心乐趣在于玩家如何分配有限的资源，将其用于构建有针对性的防御体系。

塔防游戏中最经典的是两款国外的塔防游戏，也就是《植物大战僵尸》（Plant vs. Zombies）和《王国保卫战》（Kingdom Rush）的系列游戏。而国内较受欢迎的游戏主要是《保卫萝卜》、《天天来塔防》等。

通过对塔防游戏定义的了解可以知道很多复杂的游戏类型核心的基础其实就是来自塔防游戏。游戏的休闲策略属性约定了塔防类游戏的操作不能太过复杂。一旦游戏操作复杂就容易成大型的即时战略游戏或者多人在线竞技游戏。前者是在建造和部队的培训中提高了复杂度，后者是单位的操作提高了复杂度。

### 1.2.2游戏引擎

游戏引擎作为游戏开发者研发游戏时的必备工具，游戏引擎选择是否合适关乎着游戏产品未来的销量，因此对引擎的选择就必须认真考量。目前国内还没有比较流行的自主研发的游戏引擎。根据2014年8月份Unity官方公布的报告中，Unity中国区在3D手机游戏市场的占有率已经达到75%，在全功能游戏引擎市场中也占到了45%的市场份额。可见Unity引擎已经成为大部分游戏开发者的选择。

Unity为什么受到游戏开发者的青睐？这是因为Unity引擎具有非常强大的跨平台能力，几乎兼容所有游戏平台。开发者社区的规模也很庞大。由于底层封装的完善，减少了很多开发者需要费心的环节，就入门学习而言较之虚幻引擎等老牌引擎更为容易。当然还有一个非常重要的原因就是Unity的费用也十分便宜。至于它的缺点由于还是十分年轻的一款游戏引擎，工具数量比较少，开发者很多时候还得研发一些工具。

## 1.3 课题研究目的和意义

本次课题主要希望通过使用现在主流的游戏引擎Unity来学习游戏制作流程，探索设计模式在游戏框架中的应用，最终开发一款可玩性高、程序设计较为健全的塔防类游戏。

本次课题的作品是塔防类游戏，同时也准备做成一款单机游戏。中国的游戏产业在单机游戏这一块非常薄弱，很大一部分原因是由于国内市场版权保护方面的制度还不够健全。然而即使不考虑国内，中国的单机游戏在海外的单机游戏市场份额也十分稀少。我们国家的网络游戏本土化都做的很好，但是在国际化的路上并不通顺。不过这并不代表我们国家没有优秀的作品，近期来说，在2018年下半年就有几款游戏突然崛起，占据Steam热门的前几名，例如《中国式家长》、《太吾绘卷》。目前看来依旧是本土化比较明显的产品，目标市场还是倾向亚洲几个文化类似的国家，但它们也展示了中国单机游戏的希望。在这次课题我也希望能够通过课题的制作加深对单机游戏设计的理解。

# 第二章 游戏需求分析

## 2.1游戏基本信息

游戏名称：塔防游戏-征途

游戏类型：2D塔防游戏

游戏背景：邪恶大军侵袭王国，国王令将军踏上征途，击退敌军和邪恶生物，保卫国土和解救国王的子民。

## 2.2游戏玩法：

### 2.2.1防御塔

将军拥有四种类型的防御塔可以用来阻止敌军行进到地图终点，每种防御塔可以升级成威力更大的同种防御塔。根据不同的关卡，防御塔的升级权限不同。

1. 防御塔基本属性：
2. 建造防御塔的价格
3. 售卖防御塔的价格
4. 防御塔的伤害类型，魔法伤害或者物理伤害
5. 防御塔的伤害值
6. 防御塔的冷却时间
7. 防御塔的攻击范围
8. 防御塔类型：
9. 弓箭塔：物理伤害，快速攻击，伤害值较低。
10. 减速塔：物理伤害，使敌人减速，升级后可以使敌人冻结，伤害值极低。
11. 魔法塔：魔法伤害，攻击速度相对弓箭塔而言更慢，但伤害大，可以无视敌人的护甲。
12. 火炮塔：物理伤害，攻击速度最慢，造成巨大的伤害。

关于防御塔，游戏的需求希望玩家初步掌握好弓箭塔和魔法塔的使用，随着游戏的难度增加让玩家逐渐尝试搭配减速塔和火炮塔进行布局。

### 2.2.2敌人

敌人如果行进到地图终点将会使战场生命值减少，不同实力的敌人对战场生命值造成的损失不同。一旦战场生命值降为零则游戏失败。

1. 敌人的属性设置：
2. 被杀死掉落金币
3. 被杀死掉落钻石
4. 敌人生命
5. 敌人速度
6. 敌人护甲等级
7. 敌人魔法抗性等级
8. 敌人到达终点会消耗的战场生命值
9. 敌人的划分

游戏的需求希望玩家更多的关注于敌人的护甲和魔法抗性，根据敌人的护甲和抗性妥善安排防御策略是整个游戏的核心。根据护甲和魔法抗性，敌人可以简单划分为三种：

1. 无护甲，无魔法抗性
2. 有护甲，无魔法抗性
3. 无护甲，有魔法抗性

第一种类型常见于生命值低的普通敌人和生命值高Boss级别的敌人，这些类型的敌人如果还赋予护甲或者魔法抗性属性，对于玩家来说游戏难度就过大，不利于体验。

第二、三种类型则是生命值适中的敌人，具有一定的威力，玩家的防御策略基本需要围绕这两种类型的敌人进行布阵。

### 2.2.3道具

玩家通过关卡获得的钻石可以购买道具，道具可以在游戏过程中为玩家增益。

总共五种道具：

（1）生命增加 为战场生命增加两点，可以用于避免游戏失败或者为使用的关卡加星。

（2）金币增加 为战场金币增加五百，玩家可以建造和升级更多的防御塔，提高通关几率。

（3）魔法加成，为战场的魔法防御塔提高攻击力为原本的150%

（4）物理加成，为战场的物理防御塔提高攻击力为原本的150%

（5）冻结敌人，冻结全场的敌人，为玩家争取攻击时间。

## 2.3游戏内容设计

### 2.3.1场景：

|  |  |
| --- | --- |
| **场景名称** | **场景介绍** |
| StartLoadScene | 刚打开游戏时的加载场景，用于过度到初始场景。 |
| BeginScene | 初始场景，在这个场景中选择是否继续进入游戏还是退出游戏。 |
| MainScene | 主场景，该场景功能最为丰富，主要展示关卡进度地图，由此场景还可以进入商店、成就系统、游戏帮助。以及处理游戏存档重置。 |
| GameLoadScene | 游戏加载场景，当玩家从主场景中选择了要进入的游戏关卡后就会进入该场景。 |
| GamePlayScene | 游戏场景，玩家体验游戏整个过程的主场景。 |

### 2.3.2UI面板

|  |  |  |
| --- | --- | --- |
| **面板名称** | **介绍** | **使用场景** |
| StartLoadPanel | 该面板用于控制场景跳转，从StartLoadScene跳转到BeginScene，以及处理BeginScene需要的加载。 | StartLoadScene |
| BeginPanel | 该面板功能有：音效音乐播放处理、展示游戏关于信息、进入游戏、退出游戏。其中进入游戏是加载到MainScene场景。 | BeginScene |
| MainPanel | 显示玩家持有的钻石数量，控制进入商店、成就面板、百科全书、设置，显示关卡信息面板。 | MainScene |
| ShopPanel | 显示道具信息，显示玩家持有道具数量。 | MainScene |
| AchievementPanel | 显示玩家的成就记录，未达成的成就图片将显示为灰色。 | MainScene |
| HelpPanel | 主要是用于介绍敌人和防御塔，在面板中左边防御塔，右边敌人，点击任何选项，反方向滑动出现介绍面板。 | MainScene |
| LevelIntroducePanel | 当玩家在MainPanel中选择了某个关卡，将触发该面板显示，展示关卡的名称、介绍信息。 | MainScene |
| GameLoadPanel | 控制场景从GameLoadScene过渡到GamePlayScene，另外加载GamePlayScene场景中的面板。 | GameLoadScene |
| GamePlayPanel | 该面板显示战场金币、生命值、敌人波次。附有一个暂停按钮显示游戏暂停面板。当玩家点击防御塔或者敌人时，显示对应的敌人信息和防御塔信息。 | GamePlayScene |
| TowerSetPanel | 防御塔设置面板，在空建塔点点击时显示四种防御塔供玩家选择。当点击的是防御塔的时候，显示防御塔是否可以升级，或者售卖。 | GamePlayScene |
| GamePausePanel | 游戏暂停面板，暂停游戏时可以重新开始游戏、退出游戏回到MainScene场景、修改音效音乐的开关。 | GamePlayScene |
| GameWinPanel | 游戏胜利面板，游戏胜利后显示玩家本关卡获得的星星数，显示玩家本关卡获得的钻石数。玩家可以重新开始游戏。继续游戏将回到MainScene场景。 | GamePlayScene |
| GameOverPanel | 游戏结束面板，游戏结束后显示玩家本关卡获得的钻石数。玩家可以重新开始游戏，或者退出游戏回到MainScene场景。 | GamePlayScene |

# 第三章 游戏开发工具制作与使用

## 3.1 插件与类库

### 3.1.1DOTween插件

DOTween是一种快速、高效、类型安全的面向对象的动画引擎，基于Unity而制作的。它免费且开源，有着大量先进的功能，非常适用于C#语言的使用者。与其他Unity补间动画的插件相比，DOTween是当中效率最高，使用最方便，最流行的插件。

### 3.1.2 ExcelDataReader库

ExcelDataReader本身是一个用C#语言写的高效且轻量的操作Excel文件的类库。

使用这个类库的目的是为了读取Excel中的数据，用于作品中游戏数据的配置。游戏数据的配置一般是由游戏策划做的，有很多形式，比如JSON、XML。尽管JSON和XML这种类型用于数据配置已经足够清晰了，但是相对于Excel还是棋差一著。

在本次课题中会使用Excel进行游戏数据配置，但是Excel文件本身并不用于游戏过程中的数据读取，只是一个数据存储的中间过程。我将用ExcelDataReader读取Excel的xlsx格式文件，获取到的数据再转换为Unity自己的数据保存类ScriptableObject。至于为什么不用XML或者JSON，是因为ScriptableObject是Unity自带的数据保存类，在Unity的编辑器中可视化非常好。

## 3.2 编辑器扩展

在游戏制作开发过程中，为了工作效率的提升就需要一些工具进行辅助。像上文提及的DOTween插件，这是开发者中已经广泛使用的工具，开发人员不必再次开发，直接添加使用即可。然而每个游戏的需求都各不相同，开发者还是需要自己制作一些工具配合自己的工作，这些工具很多时候仅仅适用于一个游戏项目。

### 3.2.1地图编辑

地图是塔防游戏的灵魂，每个关卡的地图数据如何设置和保存非常重要，因此必须专门设计一个地图编辑工具进行处理。这个地图编辑工具的目的是保存每一个关卡的建塔点数据。接下来将梳理地图编辑工具的开发流程。

目标：地图实例化所有的格子，我用鼠标选择哪些格子成为建塔点，记录下来。

1.数据类的处理：

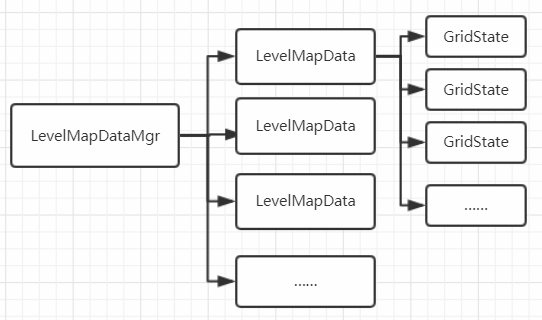
（1）每一个关卡的地图都将被平均划分为9\*12个格子，每个格子需要一个状态，记录格子自身是否是建塔点，是否已经有塔在上面了，就是GridState结构体。

（2）由于加载不同的关卡会布置不同的建塔点，我们需要一个关卡类LevelMapData，记录这个关卡所有的地图数据，也就是对应关卡所有的GridState。LevelMapData还需要一个编号，以便我们加载不同关卡时找到对应的数据。

（3）所有的关卡数据需要一个地方存放在一起，就是LevelMapDataMgr类，继承自ScriptableObject。

|  |  |  |
| --- | --- | --- |
| 数据类 | 成员 | 备注 |
| GridState(结构体)  每个格子的状态 | int id | 所在格子的编号 |
| bool isTowerPoint | 是否是建塔点 |
| bool hasTower | 是否有塔 |
| int towerID | 持有什么塔 |
| LevelMapData  每个关卡的地图数据 | int levelID | 关卡编号 |
| List<GridState> gridStateList | 该关卡所有的建塔点 |
| LevelMapDataMgr  记录所有关卡的地图数据，继承Scriptable Object | List<LevelMapData>levelMapDataList | 记录所有关卡地图数据 |

上述三个类由上到下是被包含的关系。



2.功能类

1. GridPoint

继承Unity的MonoBehaviour类，挂载到格子物体上。捕捉鼠标点击事件，实现通过鼠标点击选择哪些格子成为建塔点的功能。持有数据成员GridState，在格子被选为建塔点后就修改GridState数据。

1. MapMaker

继承Unity的MonoBehaviour类，负责加载所有的格子，在游戏时加载对应的地图数据。

1. MapTool

继承Unity的Editor类，也就是编辑器扩展部分，对MapMaker进行扩展，赋予MapMaker组件几个功能按钮。主要有保存当前关卡数据、读取指定关卡数据、搜集当前关卡数据、重置操作。

### 3.2.2游戏数据

除了上文中的地图数据，本次课题中的大部分游戏数据都是在Excel中配置的，每一种数据都需要转化为Unity的ScriptableObject数据保存格式。这部分的实现也比较简单，就是对应一个数据写一个数据处理类。比如敌人的数据EnemyInfo，跟上文的LevelMapData一样，也需要一个EnemyInfoMgr来存储每一条敌人数据EnemyInfo。EnemyInfoMgr所有的数据最后就是转化存储到Unity的一个asset文件。因此，这部分编辑器扩展的工作其实就是生成这一个asset文件。流程如下：

1. 读取每一行Excel数据
2. 每一行Excel数据生成一个EnemyInfo对象，将行数据对应EnemyInfo的成员变量
3. 将上一步的EnemyInfo对象添加到EnemyInfoMgr的List<EnemyInfo>中。
4. 将所有的EnemyInfo对象添加完毕。
5. 写一个EnemyInfoEditor类，继承Editor，使得能够在Unity菜单中添加一个子项，当点击这个菜单项时，就实现上面的流程，并且将EnemyInfoMgr生成对应的asset文件。

在本次课题项目中，以下的数据类都是这么完成的：

|  |  |
| --- | --- |
| 数据类 | 作用 |
| AchievementInfo | 成就系统的数据 |
| EnemyInfo | 敌人的数据 |
| TowerInfo | 防御塔的数据 |
| ItemInfo | 道具的数据 |
| RoundData | 回合的数据 |
| LevelInfo | 关卡的数据 |
| UIPanelInfo | UI面板的数据 |

### 3.2.3 UI面板代码生成

在第二章中提及的每个UI面板，为了便于管理，面板的变量赋值都是通过代码赋值的，而不是采用Unity拖拽的方式来赋值。因此对应面板的脚本除了需要敲写对应的变量，还要写这些变量的赋值代码。这些代码形式类似，又非常繁琐。例如GamePlayPanel中的UI变量就多达二十个。而本次课题的项目中涉及的UI面板就有十余个，每次重复这样的工作非常浪费时间。看似本次课题的项目UI面板并不多，但是如果遇到的项目非常大，类似《王者荣耀》这类游戏，需要制作的面板可以达到上百个，效率就会被严重拖低。因此我希望能够有一个代码生成器，帮助我生成对应的UI面板代码，生成变量和对变量进行赋值的代码。

代码思路流程如下：

1. 获取选中的UI面板中所有UI物体的名称；
2. 根据命名的规范，提取出需要变量的物体；
3. 编写文件字符，组合整个类的字符结构，根据代码格式添加对应的变量字符；
4. 将字符写入到文件中，生成为.cs文件。

关于命名规范，我的约定是，只有UI物体的名称为Txt\_、Img\_、Btn\_、Trans\_、Raw\_开头的物体才会创造它们的变量。最终效果要做到在Unity的hierarchy界面中选中需要生成代码的面板，点击右键菜单中的GameObject->UITool->CreateScripts就可以生成对应的面板代码。

首先建立一个UIScriptsGenerator类，继承自Editor，这样才能在菜单中创建菜单项。建立一个菜单项点击方法名为CreateScript，方法中先要获取到当前鼠标选中的物体，UnityEditor中提供了Selection.gameObjects可以实现这个要求。这是一个数组，我的需求不考虑多选物体生成代码的情况，因此获得数组第一个对象即可。

获取到选中物体后，需要查找到这个物体下所有的子物体，使用GameObject.GetComponenetsInChildren<Transform>()可以获得一个包含所有子物体的Transform数组。紧接着获取子物体的名称，根据name.Split('\_')可以获得物体名称的前缀。用一个List<string>存下所有的物体名称，同时也要根据前缀存下它们的类型枚举到一个List<UIItemType>中。

public enum UIItemType

{

Txt,

Img,

Btn,

Raw,

Trans,

}

物体的数据收集完就看可以开始拼代码的字符了。

using UnityEngine;

using UnityEngine.UI;

public class 面板名称：BaseUI

{

遍历List<string>、List<UIItemType> 添加对应的变量

public void Init()

{

遍历List<string>、List<UIItemType> 添加对应的变量赋值

}

……

其他常用的方法

}

## 3.3 调试信息输出工具

游戏在制作过程中需要进行调试测试。在编辑器模式下进行测试，遇到问题的时候通过调试信息基本能够比较直观的确定问题所在。但是游戏的测试不仅仅只有在编辑器模式的时候测试，更重要的是真机测试，也就是将游戏打包出来进行测试。然而在真机测试的时候，游戏一旦出现BUG往往只能凭借经验猜测问题可能存在在什么地方。而修复BUG的时候又得回到编辑器模式进行处理，有些BUG仅仅出现在真机测试的情况，编辑器模式难以模拟复现，也就导致查找问题所在变得非常困难也很消耗时间精力。如何在真机测试的时候准确定位到代码的问题所在就显得尤为重要。

我们在查找BUG的时候，最常见的两种方式就是断点调试和通过调试信息定位。在真机测试的情况下，显然我们无法使用断点调试的方式，因此重点就落下调试信息上。调试信息通常是我们为了确保某些代码是否执行而在代码中添加的一些输出信息，也包括代码中的报错。调试信息基本是在控制台输出的，我需要做的就是在真机测试的时候也能查看到调试信息。简单说就是在游戏界面中做一个类似UnityConsole的界面。

首先，获取Unity的调试信息。基于最基本的编程常识，Unity一定有开放调试信息的接口。因为很多游戏有日志保存功能，日志保存的内容其中一部分就是调试信息。因此通过对Unity官方文档的搜索，可以找到调试信息接口:Application.logMessageReceived。这是一个委托事件。它会传三个参数，第一个是调试信息，第二个是调试信息的堆栈信息，第三个就是调试信息的类型。我建了一个对应的结构体来接收这三个信息：

struct Log

{

public string message;

public string stackTrace;

public LogType type;

}

接着开始模拟UnityConsole绘制到游戏界面中。除此之外还有三个重要功能需要模拟，第一是显示堆栈信息，第二是折叠重复信息，第三清空调试信息。

整个工具用一个类来实现即可，建立一个TestConsole类，继承自MonoBehaviour。绘制界面的部分需要借由MonoBehaviour中的OnGUI方法来实现。再者用HandLog方法来接收调试信息，再将每条信息暂存到List<Log>中用于后续绘制时使用。

在OnGUI中先绘制一个GUILayout.Window，在其中再绘制工具栏、Log列表、堆栈信息标签。

工具栏包括一个Clear按钮和一个Collapse勾选。这两个工具实现清空调试信息和折叠重复信息。

Log列表中的每一行有两个组件，一个是显示调试信息的标签，另一个是显示堆栈的按钮stackTrace。点击某一行的stackTrace按钮，堆栈信息标签就会显示该行调试信息的堆栈内容，便于我跟踪代码。

最后需要一个控制显示的布尔变量，我不希望这个界面时时存在与游戏运行过程，指定某个按键来控制界面的显示与隐藏，平时默认为隐藏，在测试出现问题的时候再手动显示这个界面查看调试信息。

# 第四章 游戏内容具体实现

## 4.1 游戏框架

以下是游戏框架图，纵向的箭头表示继承关系，横向箭头表示成员控制关系。

GameRoot是继承MonoBehaivour的，在GameRoot的Awake()方法中对UIMgr等成员进行初始化操作，UIMgr中的Update()方法则在GameRoot.Update()中调用。

### 4.1.1 UI框架

UI框架对于游戏来说十分关键，因为程序的流程基本就是通过UI框架控制的UI面板进行的。下图是UI框架的脑图。

UI框架中最主要实现功能的部分在UIFacade中，但UIFacade是由UIMgr调用的，一般外部无法直接访问UIFacade。UIMgr继承单例模板。其次所有第二章提及的面板都是继承BasePanel。

比如要显示一个MainPanel面板的时候，UIMgr.Instance.Show(UIPanelName.MainPanel)，执行这一步后，如果是第一次执行，UIFacade会向UIBusiness获取MainPanel类，然后获取MainPanel的UIPanelInfo信息，得知预制体的保存地址和UI面板的层级，再加载预制体并将预制体与MainPanel类进行关联。显示的面板会添加到panelShowDict字典中，被隐藏时则会从字典中移除，只有在字典中的面板的BasePanel.Update()才会被调用。

当游戏程序更换场景时，旧场景的面板资源就要被释放，就会BasePanel.OnDestory()。

### 4.1.2资源加载与工厂

一般大型的游戏对资源的管理是十分看重的，因为大型游戏对内存的占用非常大，如何合理的运用内存资源就很关键。一旦超出内存导致游戏卡死，或者程序频繁调用垃圾回收机制（GC）引起游戏卡顿，这就会导致玩家体验感极差。本次课题的游戏是小游戏，内存问题一般不会存在，但是管理好内存是一个游戏开发者应该做的。

下图是关于资源加载部分的脑图：

其中FactoryMgr负责外部调用的，比如成就系统列表要加载一个成就的UI，则需要这样调用：FactoryMgr.Instance.GetUI(uiPath)。回收这个UI的时候就是FactoryMgr.Instance.PushUI(uiPath,uiGO)。回收的时候就是将UI加入到对象池中。

在脑图中的UIFactory和GameFactory均继承自BaseFactory，BaseFactory管理需要重复生成的游戏物体的对象池。这里引入了对象池的概念，对象池就是意图通过重用固定池中的对象而不是单独分配和释放它们来提高性能和内存使。用UIFactory管理的比如MainPanel中的关卡按钮一般最多有三十个，可以用对象池管理。GameFactory就比如防御塔、子弹、敌人等。

其他的SpriteFactory、AudioFactory则是继承BaseRescourceFactory，类似Sprite、AudioClip这类资源加载一次即可在游戏进程中一直使用，因此只需要用字典来进行记录，保证不重复加载即可。

### 4.1.3消息机制

一个健壮的程序系统一般要求低耦合、高内聚。我们通常不希望系统中的模块之间有太多的依赖，希望每个模块都尽可能独立完成自己的功能。但是模块之间也不可能毫无交互，甚至有时候一个模块的某个操作需要跟不止一个模块进行处理。为了保证模块的独立性，我们一般把它们的交互交给一个稳定的第三方来处理。设计模式中有一种模式适用于上述提及的情况，那就是观察者模式。在程序中一般用来搭建消息机制，使用一个消息处理中心来达到解耦的目的。

本次课题的消息机制一共就三个文件，EventCenter、EventType、CallBack。

（1）CallBack很好理解，就是泛型的委托，定义了从无参数到五个参数六种泛型委托。

（2）EventType，这是一个枚举。消息号就是EventType里面定义的枚举，每次需要新的消息类型，就在EventType中定义即可。

（3）EventCenter，消息机制处理中心。有三个核心的公共静态方法提供给程序模块使用：

AddListener(eventType,callBack)：注册监听事件；

RemoveListener(eventType,callBack)：移除监听事件；

Broadcast(eventType,……)：消息广播。

在EventCenter中使用字典来记录事件，key值是EventType，value值则是Delegate。Delegate本身就像一个链表，可以通过+-的符号进行叠加和删减。在Broadcast的时候也是直接使用Delegate，不需要进行遍历就可以将全部方法都调用。

### 4.1.4建造者模式

建造者模式也是一种设计模式，目的是将一个复杂的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。建造者模式经常跟工厂模式配合使用，工厂产生零件，建造者负责组装。本次课题里面实现的建造者模式形式十分简单，EnemyBuilder、BullectBuilder、TowerBuilder都继承了一个IBuilder<T>接口，生成对应的方法进行实现。

主要是四个方法：

GetProduct()：返回GameObject，在这个方法中调用FactoryMgr获得游戏物体。

GetProductClass(GameObject gameObjct)：返回T，在这个方法中获取游戏物体的组件进行配置。

GetData(T productClassGo)：GetProductClass返回的T传入该方法进行数据配置。

GetOtherResource(T productClassGo)：个别特殊的实现在这里进行。

在本次课题中如果需要生成一个Enemy，就会通过EnemyBuilder调用GetProduct()，返回的GameObject就是配置好的了，可以直接使用。GetProduct()中的逻辑大致如下：

Public GameObject GetProduct()

{

GameObject go=FactoryMgr.Instance.GetGame(敌人预制体地址);

BaseEnemy enemy=GetProductClass(go);//获得敌人身上的BaseEnemy组件

GetData(enemy);//在GetData方法中对BaseEnemy组件进行配置

}

### 4.1.5玩家数据存档与读取

关于玩家数据的存档有很多种方式，我采取的是二进制文件。二进制文件基本不具备可读性，玩家不能自行修改数据，对开发者来说就可以避免作弊的情况发生。而且二进制文件的读写是最快的。而JSON和XML都是可读性比较强的文件，玩家可以轻易读懂数据的保存形式，这就不是开发者所希望的。实现玩家数据存档和读取需要两个类：

PlayerData：玩家数据类

PlayerDataOperator：玩家数据操作类

PlayerData记录的内容也很简单，主要是完成关卡的情况、成就达成情况、玩家持有的道具、玩家持有的钻石数量。

PlayerDataOperator有三个公共方法：

LoadPlayerData()，加载玩家数据，在游戏启动时调用。

SavePlayerData()，在各种需要存档的时候调用，如游戏结束、购买道具等。

ResetPlayerData()，玩家删除存档的时候调用。

## 4.2 游戏场景与UI面板

### 4.2.1 场景切换

为了场景切换时视觉上的过渡自然，我在场景退出前会先显示一块黑色的遮罩，进入新的场景时又将这块黑色遮罩隐藏。遮罩的出现和隐藏也要尽可能自然，所以使用的DOTween的To方法，在一定时间内通过修改图片的颜色Alpha值达到渐隐或者渐现的效果。

因为遮罩只在场景切换时使用，我将遮罩的管理也放在SceneStateMgr中。当SceneStateMgr被初始化时，遮罩也会被初始化。在SceneStateMgr中带有两个私有方法，一个是ShowMask()，另一个是HideMask()，分别用来控制遮罩的显示和隐藏。

我为每个场景都建立一个场景状态类，这些类都继承BaseSceneState类。它们只有两个公共方法需要实现。一个是EnterScene()，另一个是ExitScene()。在进入场景和退出场景需要做什么工作都在这两个方法内调用。

场景状态类一共五个：StartLoadSceneState、BeginSceneState、MainSceneState、GameLoadSceneState、GamePlaySceneState。

如果要从StartLoadSceneState切换到BeginSceneState，就要执行SceneStateMgr.Instance.ChangeSceneState(new BeginSceneState)。在场景管理类中首先会执行ShowMask()先显示遮罩，当遮罩完全显示时会执行一个回调，我在这个回调中处理场景StartLoadSceneState.ExitScene()和BeginSceneState.EnterScene()。这样一个场景的切换所涉及的工作就全部完成了。

### 4.2.2 UI面板

本次课题一共有13个UI面板，这边主要介绍几个功能比较多、处理时比较困难的两个面板，其他面板的处理方式基本在这两个面板中都能体现。

1. **MainPanel**

MainPanel涉及到ShopPanel、AchievementPanel、HelpPanel、LevelIntroducePanel的显示。与第四个关卡介绍面板LevelIntroducePanel关联的关卡按钮的管理比较复杂。

我在MainPanel类中建立了一个内部类是LevelButton，继承BaseUIListItem。MainPanel在初始化的时候就会实例所有的LevelButton。每个LevelButton在构造的时候会根据自己的编号获取关卡信息和星级信息并显示，然后注册按钮点击事件。ShopPanel、AchievementPanel、HelpPanel中关于道具按钮、成就项、敌人按钮、防御塔按钮都是依照这种方式进行的。

MainPanel中有一个横版地图，所有的关卡按钮会显示在地图上，玩家通过拖拽的方式来查看地图的全貌。“拖拽”如何实现，是一个困扰我的问题。常见的实现拖拽方式都是继承几个Unity的接口，一般是IBeginDragHandler、IDragHandler、IEndDragHandler这三个接口。这三个接口非常方便，但是必须继承MonoBehaviour。我们的面板类并没有继承MonoBehaviour。当然我可以直接建一个MonoBehaviour的子类挂载在横版地图上，然后继承这三个接口，但这不利于程序管理，尤其是迁移项目的时候，挂载在物体上的类容易丢失。

后来通过Unity官方文档，我还是找到了一个方法，那就是EventTrigger组件。实现方法如下：

**EventTrigger** mImg\_MapEventTrigger;

**EventTrigger.Entry** onDragEntry;

onDragEntry = new EventTrigger.Entry();

onDragEntry.eventID = EventTriggerType.Drag;

onDragEntry.callback.AddListener((data) => { OnDrag((PointerEventData)data); });

mImg\_MapEventTrigger.triggers.Add(onDragEntry);

1. **GamePlayPanel**

这个面板主要功能就是显示战场的金币、波次、生命值、暂停按钮。点击敌人或者防御塔的时候还负责显示敌人或防御塔的部分信息。这部分的工作其实最主要的就是消息机制的运用。敌人的死亡或者到达终点都会影响面板的金币和生命值显示。最直接的办法就是每个敌人都持有面板的引用，然后面板将金币和生命值的变量设置为公共变量，提供给敌人修改。但这不是我希望的，也不符合模块独立的要求。

为了解决金币值等更新的问题，就需要使用消息机制。面板显示的时候就注册金币更新的消息。当敌人死亡的时候广播更新金币，并传递增加的金币数。面板接收到消息后就自动更新金币值。

当我们点击某个敌人或者防御塔，要查看它们的信息时也是通过消息机制来处理。敌人被点击时触发OnMouseDown()方法，我在该方法中进行广播敌人被点击的消息，并传递敌人的信息。面板接收到后，显示敌人信息的面板并更新面板上的信息即可。

在该面板中还有关于道具使用的部分，这部分加载道具的逻辑与MainPanel中的LevelButton相同。道具也是按钮。道具被点击后发送消息和传递自身的编号，根据编号接收消息方执行相应的操作。如金币道具，就会让GamePlayPanel更新金币数值。物理加成道具就会广播全部敌人提高物理伤害。

## 4.3 游戏玩法逻辑

### 4.3.1防御塔

关于防御塔有两个基类，一个是BaseTower，还有一个是TowerProperty。

1. BaseTower

BaseTower继承MonoBehaviour，主要功能很简单，就是确认范围内的敌人目标。

在敌人身上挂载了碰撞器之后，就可以通过OnTriggerEnter、OnTriggerStay、OnTriggerExit来获取进入范围内的敌人。我用一个链表保存所有在范围内的敌人，敌人进入时添加到链表，离开时从链表移除。当敌人死亡或者到达终点时，还在范围内的敌人也要从链表中移除。

BaseTower在初始化的时候会注册敌人移除的消息事件。当敌人死亡或到达终点时会向所有防御塔广播。接收到消息的防御塔就会从链表移除这个敌人。

1. TowerProperty

TowerProperty负责攻击、冷却计时、塔的转动。

因为本次课题的游戏有四种防御塔，不同的防御塔有不同的需求。我通常是再建立一个类继承TowerProperty，重写里面的部分方法来实现的。比如慢速塔和法师塔并不需要防御塔转动瞄准，重写的时候就把防御塔转动的方法置空就可以了。

只要防御塔攻击目标不为空且不在冷却期间，防御塔就会执行攻击方法。

private void Update()

{

if (isBeginCD)

{

if (timeVal >= baseTower.towerInfo.CD)

{

timeVal = 0;

isBeginCD = false;

}

else

timeVal += Time.deltaTime;

}

if (target == null || GameController.Instance.isPause == true)

return;

TransformRotate();

if (isBeginCD == false)

Attack();

}

### 4.3.2敌人

在本次课题，敌人就一个基类BaseEnemy，没有其他的子类。对于敌人的最大需求就是按照指定的路径行走到终点为止。敌人移动的实现是使用插值运算，已知敌人当前位置和目标位置，调用Vector3.Lerp来实现，它有三个参数，头两个是起始位置和目标位置，第三个参数是移动的比例。在使用插值运算方法时，如果移动比例设置为一个固定值，敌人越接近目标点，移动的速度就会越慢。因为每次移动都是按照当前距离的一个比例来移动。而距离是缩短的，这就导致每次移动的距离也在缩短。

为了达到移动的匀速进行，我就把移动距离单位化，使得每次移动都按照单位距离移动。

position = Vector3.Lerp(起点, 终点,1/Vector3.Distance(起点, 终点) \* Time.deltaTime \* enemySpeed);

解决了移动的问题之后，第二问题就是敌人的转向。这个主要是根据当前位置与目标位置的XY差值来处理。2D角色的转向表现是通过播放不同的动画实现的。因此根据差值的正负判断播放对应的动画即可。

if (xOffset < 0)

animator.Play("Right");

else if (xOffset > 0)

animator.Play("Left");

else if (yOffset > 0)

animator.Play("Front");

else if (yOffset < 0)

animator.Play("Back");

### 4.3.3子弹

子弹基类为Bullect当防御塔执行攻击方法时就会调用子弹的建造者产生子弹在防御塔位置，然后将攻击目标赋给子弹。

子弹移动方式并不是物理的方式，跟敌人移动是一个逻辑，不再赘述。

当子弹与敌人的碰撞器接触时，子弹就会向敌人发送伤害的消息。这边使用的不是上文提及的消息机制。而是Unity自己的SendMessage方法。因为游戏框架中的消息机制是广播性质的，没有针对个体的，当然这是可以实现的，但是既然Unity已经有了，就直接拿来用。火炮塔和弓箭塔的子弹基本就是走上述逻辑，慢速塔和魔法塔则又不同。

慢速塔的子弹严格来说不是“子弹”，只是一个圆形动画，在防御塔基础上播放，因此没有移动的逻辑。慢速塔的攻击是范围攻击，也就是遍历攻击前面防御塔提到的敌人链表中的每个敌人。慢速塔也会发送攻击消息，但同时还会再发送一个减速消息。

魔法塔的子弹是线性闪电，所以要重写移动的方法。线性闪电有两种实现方式，一种是使用LineRender组件，还有一种是在目标点和起始点中间生成与距离等长的闪电。我采取的是第二种方式。

# 第五章 总结

为了本次毕业设计作品的完成，我在此之前进行大量的程序设计相关的学习。《设计模式与游戏完美开发》和《游戏编程模式》在游戏框架设计上给了我很大的启发。我也初步通过这次的作品搭建了自己的游戏框架和类库。

在整个制作过程中，我也深深的体会到了游戏制作的不易。在一个游戏开发团队中，游戏策划、游戏美工等其他成员也至关重要。我作品中的美术资源都来自于网络，但是由于资源的问题，很多我还需要再次处理。比如角色动画，我只有敌人动画图集，没有分割图集的文件。因此每一帧动画都是我在PS中一个个裁剪出来的，仔细算来大约裁剪了一千多帧。游戏数据方面，这部分工作一般是由游戏策划来做的。因为我是独立开发，所以我自己配数据，三十关的数据一共有三百五十条左右的数据需要手动配置。自己切身体验过后，我相信未来团队工作会更加顺利。

开发过程中我也遇到了很多困难，好在大部分努力查找资料后基本都得到了解决。我在这段时间的进步是巨大的，我的写程序的思考方法也比过去更加严谨和妥善。

# 参考文献

# 致谢

非常感谢王宝土老师一直以来的指导，我开始接触Unity开发就是受到宝土老师的启发。在宝土老师的实验室中学习了很多基本的Unity知识，为我后续的开发打下了坚实的基础。同时也很感谢实验室里的王栋学长，他给我布置学习任务，在我遇到难以解决的问题的时候给我答疑解惑，为我提供思路。

同时我也非常感谢我之前的实习单位——福州市马尾区朱雀网络信息技术有限公司，团队里的同事对我都非常照顾，悉心带领我熟悉游戏开发工作，给我推荐了很多学习资料和学习方法，使我的业务能力在实习期间得到了迅速的提升。

四年的大学生活转瞬即逝。感谢福州大学对我的培养，感谢专业老师的一直以来的帮助和提点，我将永远铭记在心！