摘　要

模拟经营游戏是模拟游戏的一种。而模拟经营游戏就是由玩家去充当特定的角色，在这个虚拟的游戏世界进行自我发挥的经营。这个游戏世界可以是团体、公司、城市、国家、甚至是整个世界。模拟经营类的游戏游戏通常都会仿照现实中的模式，游戏的玩法跟人物的模型想法等都可以按照自己的需求定制或随机定制。一般来说，常见的模拟经营游戏的游戏主题无非是收集游戏资源，最后实现一家独大的成就；或者发展到一定程度（例如大富翁玩家资产处于第一位或遥遥领先）。模拟经营游戏最核心的是游戏玩法。值得一提的是模拟经营的游戏玩法越新颖，其受众用户群体越大，吸引的玩家也越多。一款出色的游戏想要被大众所接纳与喜爱，其本身就必须拥有闪光点，而这就要跟游戏的理念所挂勾。。

模拟经营游戏也包括仅能离线（Offline）进行的类型，这是脱离线上玩家条件下的单机模拟经营，玩家需达到最后的游戏目标，运用自身或游戏内各种能力来获取胜利。模拟经营游戏跟角色扮演游戏不一样，并不是非常注重游戏中的剧情故事、以及游戏中人物的塑造；相同的是都直接模拟出游戏中所描绘的虚拟世界。

餐厅模拟游戏便是其中的分支，玩家利用先用的资源通过自身的能力去塑造出无限的可能，玩家能在游戏中体验创业的乐趣，享受担任港式茶餐厅 boss 所带来前所未有的体验。

关键词：　Unity3D　模拟游戏　餐厅模拟

Abstract

The simulation management game is a kind of simulation game. Players play the role of managers to manage the virtual real world in the game. It can be a city, a country, a world, a company, or another group. Virtual business games generally simulate real-world models, and the characters 'ideas are randomly customized according to rules. Many simulation business games end with the word "Tycoon". The goal of common games is to completely control all resources that can be controlled in the game and achieve monopoly; or to a certain extent (for example, enough technology to lead citizens out of the earth). The core of the simulation management game is gameplay. In gameplay, players play or control a character in a realistic or fictional world. The player is responsible for playing this role and developing the role played by some actions under a structured rule. The player's success and failure in this process depends on a formal system of rules or guidelines.

Simulated business games also include types that can only be played offline (offline). Players control a person or a team to complete the game goals, and use various abilities to achieve victory. The simulation management game is different from the role-playing game. It does not pay much attention to the story in the game, the scenery of the virtual world, and the shaping of the characters played; the same is that they directly simulate the virtual world depicted in the game.

The restaurant simulation game is a branch of it. Players use the first-hand resources to create unlimited possibilities through their own capabilities. Players can experience the fun of entrepreneurship in the game and enjoy the unprecedented experience of being a Hong Kong-style tea restaurant boss.

Keywords: **Unity3D Simulation game Restaurant simulation**

目 录

第一章 绪论

1.1 开发意义及其发展前景

本课题的意义是设计并实现一个模拟经营类游戏，基于港式茶餐厅的风格及其经营方式打造出一款新颖的餐馆经营游戏，同时从现实出发，为用户呈现出当今茶餐厅的经营方式、经营理念

本课题的主要目标是设计与开发出一个基于 Unity3D 的模拟经营类游戏”TOP”，在单机游戏模式下用户将独自经营一家港式茶餐厅。以 Unity3D 作为开发游戏的工具,使用符合港式风的模型,构建符合港式茶餐厅主题的游戏场景,使用 Unity3D 自带的插件并且编写脚本代码,经历一个游戏开发周期，实现玩家可对游戏进行操作。构思独特的港式风游戏剧情,设计港式风 UI 界面,同时加入游戏外部文件等大量的游戏设计必要内容如柠檬茶、奶茶等港风理念令游戏的信息增大,并且随着游戏的进行,游戏中关键词港式茶档的理念,通过游戏内容展现出来。最终完成一款功能完善、实现了一定可玩性的主题类模拟经营游戏。

1.2 国内外的发展状况

时至今日，在国内 3D 游戏的研究已经非常普遍，甚至连许多高校都有教授游戏方向的专业。因为国内 3D 游戏开发技术引进时间比较晚，所以导致了在技术方面上我们与国外有很大的差距，其次，在游戏的开发与制作时，我国基本没有独立研发的游戏引擎，现阶段我国许多游戏公司开发时的选择是国外所开发的游戏引擎诸如寒霜引擎，同时国内开发游戏并不能排上一级游戏行列，价值也不高，这也是国内游戏一直被人诟病的主要原因。

如今， Android 和 IOS 是两种主流的手机操作系统，成为主流的原因，是在这两个操作系统基础上开发出大量的应用。因此，安卓系统与苹果系统现在已经占据了整个手机市场，为了使软件能够在两个平台同时兼容，于是出现了能够跨平台的软件移植，但是，跨平台的应用移植会出现很多难题以及 bug 需要修复，如开发语言不同，移植代价高等，但是游戏开发的圈子，Unity3D 游戏引擎却必不会出现这些问题，在 Unity3D 中运用到虚拟机技术，独立于平台执行的游戏编译结果也是一个亮点。对于 3D 引擎的研究，我们国内还处于摸索阶段，需要足够的时间去发展，学习优秀的游戏引擎，吸收这些优秀的特点为己利用，对于我们将来无论是开发游戏还是开发游戏引擎都有很大帮助。

第二章 开发技术

2.1 Unity3D 相关介绍

Unity3D 是由 Unity Technologies 公司开发的一个专业游戏引擎。值得一提的是 Unity3D 被高占比使用的原因在于使用 Unity3D 能够轻松的开发一款游戏，Unity3D 能做到的事情非常之多，例如高帧率的三维动画、三维视频游戏、极其逼真地虚拟现实游戏等，最关键的是 Unity3D 实现了多个平台开发如安卓、PC、iOS 等，这也令其深受游戏开发人员的重用。

随着游戏产业发展越来越迅速，快速开发高质量游戏的需求已经逐渐占据重要地位，而 Unity3D 不仅可以提高开发效率，游戏渲染质量也属于中等偏上，在手游产业更是不可被取代，并且由于 Unity3D 的版权费比其他主流 3D 游戏引擎便宜很多，使得在国内越来越多游戏开发者使用该引擎进行商业化开发。

在 Unity3D 中，将支持 C#语言、JavaScript、Boo 这 3 种语言来编写游戏脚本。并支持 Shader 的编写，本设计采用的编程语言为 C#语言。

2.1.1 Unity 脚本生命周期

Unity 程序脚本有一套比较完善的生命周期，在这个从唤醒函数到销毁函数都的生命周期中，有着一定的制约，那便是在开发过程中遵从自身的 Unity MONO 生命周期规则。下列是在 Unity 的生命周期中由系统内自身调用很重要的方法：

(1) void Awake() {}：脚本初始化的启动，将脚本唤醒，其后进行初始化工作，在 Mono 库中这是系统执行的第一个方法，并且只被执行一次在脚本的生命周期中。

(2) void OnEnable() {}：启用，物体启用时被调用。

(3) void Start() {}：开始处理，开始逻辑，关于这个方法的详细讲解指是在 OnEnanle Awake()方法之后、Update()方法之前执行，而且只能够执行一次这个方法。

(4) void Update() {}：这个方法是正常更新逻辑，在游戏中的每帧都会被进行处理。而这个逻辑处理只能调用一次。

(5) void LateUpdate() {}：这个方法是推迟更新，执行时系统中会调用每一帧，但必须在 Update()方法执行完后，此方法才能由系统去调用。

(6) void FixedUpdate() {}：固定更新，和帧频无关，和时间有关系。固定更新常用于移动模型等操作。

(7) void OnGUI() {}：此方法用于绘制界面系统，任何时候都在调用绘制界面，用于绘制系统的界面，处理按钮。

(8) void OnDisable() {}：禁用时候调用，和 Active(false)，物体被禁用时调用。

(9) void OnDestroy() {}：销毁，和 OnDestroy(gameobject)调用，调用的条件是脚本被销毁时。

根据 Unity3D 脚本中的生命周期，当在调用方法时，首先初始化的顺序，其次建立出 Unity 脚本生命周期方法调用过程顺序图如图 2-1 所示。

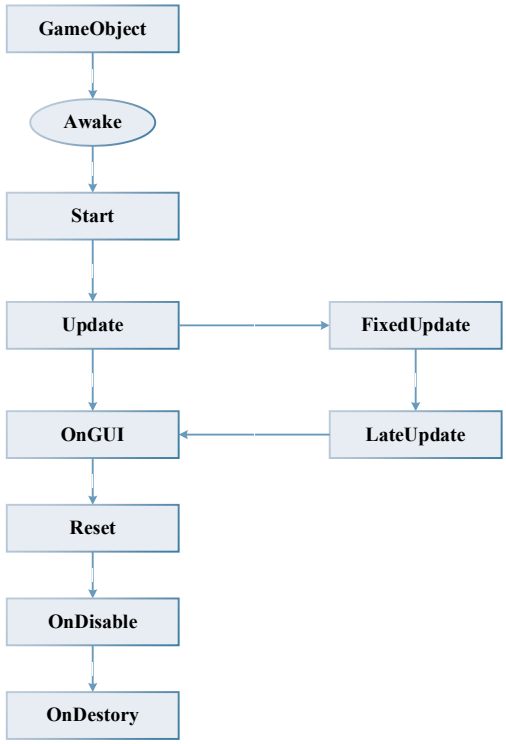

图 2-1 Unity 脚本生命周期方法调用过程顺序图

2.1.2 Unity 跨平台技术

Unity3D 几乎是市场上最流行的游戏开发引擎了，由 Unity Technology（以下简称 UT）公司开发，它可用于 Windows 和 Mac OS X 系统（Linux 系统实验版已发布）。最重要的是，它几乎可以导出到任意平台。Unity 是一个优秀的游戏引擎。Unity 中对各平台编译指令如表 2-1 Unity 跨平台技术编译指令所示：

表 2-1 Unity 跨平台技术编译指令

名称	说明
UNITY_EDITOR	定义从游戏代码调用 Unity 编辑器脚本
UNITY_ANDROID	为 Android 平台的平台定义脚本
UNITY_IPHONE	为 Iphone 平台编译/执行的代码的平台定

	义脚本
UNITY_STANDALONE_WIN	为 Windows 独立的应用程序编译/执行代码时使用的脚本
UNITY_STANDALONE_OSX	专门用于 MacOS（包括 Universal、PPC and Intel 架构）编译或执行的代码的平台

2.2 物理引擎

在 3D 游戏这种综合类应用当中，除了逼真的模拟出真实 3D 世界的环境效果，物理引擎也是必不可少的环境功能。在真实 3D 世界当中，物体与物体之间是存在相互作用与物理规律的。而在游戏环境中，在模拟刚体的前提下，如果是简单的物理运动，例如是游戏中简单的移动、加速等，这是可以通过一般的游戏引擎然后进行编程或编写脚本去实现功能。当开发游戏需要模拟复杂的物理规律，如弹起、滑动、碰撞、爆炸等效果时，比如编译底层算法会使得工程延误，这就与软件工程的思想产生矛盾。在 3D 游戏环境中，成熟的物理引擎将支持更复杂更真实的 3D 环境物理模拟。

物理引擎的基本架构分为两个模块，碰撞检测与物理学世界。如图 2-1 所示。

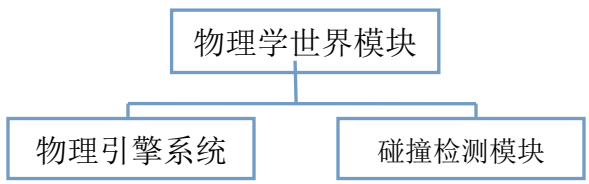

图 2-1 物理引擎结构示意图

当 3D 场景各物体的位置信息以及场景本身的各种信息通过外界调用模块，输入到物理引擎中被接受，物理引擎即产生相应回馈从而产生物理效果。因调用模块的不同，输出的计算结果也不相同，但计算的方式是相同的：碰撞发生的地点、发生碰撞后物体的地点、物体与物体和物体与场景之间是否有碰撞发生。最后将通过计算得出结果再发送给已经调用的模块。物理引擎功能示意图如图 2.3 所示。

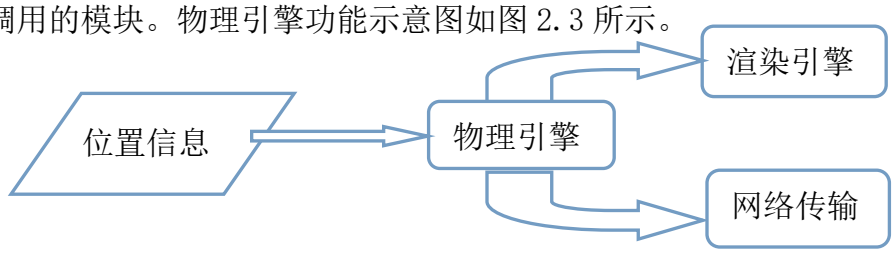

图 2-2 物理引擎功能示意图

2.2.1 Unity 概念解释

(1) Scene 场景，我们可以把它当作一部电影，储存了各种不同的资源，储存后的资源文件的后缀名都是 Unity。编写的脚本中使用 Scene 类来表示。

(2) Game Object 游戏对象，可以把其视为电影中的景物、道具、角色等等能够看见的物件。代码中使用 GameObject 类来表示。

(3) Light 灯光，可以视作电影的灯光调节器，使用其可以任意调节整个虚拟世界的明暗。代码中用 Light 类表示，Light 继承自 Component，同样其身份为游戏对象的组件。

(4) Prefab 预制件是对与自定义的一组游戏对象，开发者会预先配置，设置预制体相互位置，预制体的层级关系等等，然后把它们存储为模板形成一个整体。后续使用这个整体模板，进行的操作就是构建出一份游戏对象的拷贝。

(5) Asset 资源，Unity 中有很多资源， GameObject、Component 其实都是资源，但很多时候，我们开发者说的资源指的是：网格、动画、贴图、材质、音频片段、字体等等。

如果出现的游戏资源类型很多，那么对应的类也不同如

表 2-2

名称	文件格式
网格：Mesh	fbx、obj 等
动画：Animation	anim 等
贴图：Texture	psd、tif、png、jpeg 等
材质：Material	mat
音频：AudioClip	ogg、wav、mp3 等
字体：Font	ttf、fnt 等

在开发过程中，UnityEngine 命名空间下方的 Object 类都会被资源类所继承，注意与 CSharp 默认空间中的 Object 不同。

第三章 游戏需求分析

3.1 系统可行性分析

3.1.1 技术可行性分析

本游戏是一款基于港式茶餐厅的风格及其经营方式的模拟经营类游戏，从现实出发，希望能够借此为用户呈现出当今茶餐厅的经营方式、经营理念。游戏平台面向是Windows，通过我自行对游戏内容的数据分析以及需求，同时我也考虑了开发环境、运行环境、设备要求，在技术上确定使用Unity3D能够开发出这款模拟经营游戏，在功能需求上可以通过现有的知识及自身需学习的知识去实现游戏功能。并且采用市面上模拟经营类的游戏大众化操作，能够大幅度降低对自身的游戏开发技术要求，让玩家在享受策略类游戏带来的乐趣 。

3.1.2 经济可行性分析

本游戏基于Unity3D开发的模拟经营游戏不需要任何经费成本，游戏中的模型场景皆是网络免费开源，因此在忽略成本的条件下能够最大限度的开发此游戏。同时本游戏所需资源皆是网络收集，游戏完成后不实现经济效益，即利用该游戏获取经济利益，所以从经济上开发此游戏是可行的。

3.1.3 法律可行性分析

本游戏纯属个人开发并没有任何商业用途、没有开展商业活动、没有获取经济效益，因此，在利用网络上的素材基础的条件下开发此款游戏没有侵犯他人利益，即不存在不合法行为。在法律上开发此游戏是可行的。

3.2 系统功能需求分析

3.2.1 游戏功能划分

根据模拟经营游戏TOP的设计需求与游戏的玩法，将整个游戏功能划分4个部分，分别是角色控制、商店功能、关卡功能、订单功能如图3-1游戏功能模块图所示。

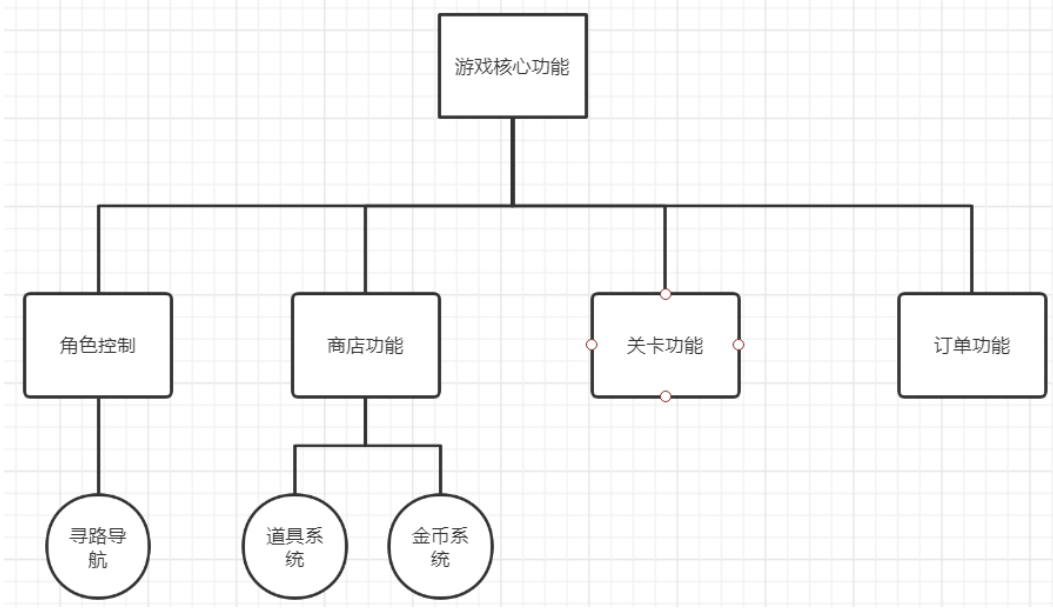

图 3-1 游戏功能模块划分

3.2.2 游戏功能描述

角色控制功能：玩家开始进入游戏后，能够使用鼠标操控服务员进行移动完成订单的出餐。使用鼠标操作简单易上手，注重玩家游戏体验。

商店功能：在游戏商店中我增添了道具系统使得游戏性大大提高，丰富了游戏的玩法，道具分别有移速加成(即加快服务员的移动速度)，订单立即完成(即随机立即完成订单)、服务员招募(即增加服务员数量)。金币系统的设置是为了商店系统能够顺利实现，在游戏中每完成一个订单，则增加 10 金币，金币可用于商店道具的购买。

关卡功能：游戏中一共有三个关卡，越往后订单需求完成量越高难度越大，玩家在规定时间内完成任务则闯关成功可进入下一关，否则视为闯关失败。

订单功能:订单是胜负判定、奖励的关键，游戏中订单事件会随机生成（订单菜单与港式茶餐厅有关，即奶茶、柠檬茶、西多士、碟头饭四种），一个外卖员对应一个订单，服务员出餐满足订单需求并成功送给外卖员订单才算完成。

3.3 系统性能需求分析

3.3.1 精确度

保证游戏进行中的玩家数据操作精准，保证配置文件准确性与公平性，对游戏战斗结束的判定进行严谨的监控。

3.3.2 时间特性

游戏中的界面切换的响应时间在 0.1-0.3 秒内，加载页面的加载动画的响应时间在 0.5-1 秒内，点击效果与角色接触消失效果的响应时间在 0.2-0.5 秒内。

3.3.3 适用性

游戏适用于各大品牌的 PC 系统电脑，能够安全稳定运行，出现游戏闪退的几率较少。

第四章 游戏系统设计

4.1 游戏系统总体设计

本游戏系统是基于 PC 平台运行的 3D 第三人称模拟经营类游戏，游戏参照《QQ 餐厅》开发，因此，游戏实现的功能在这两款游戏基础上进行扩展。根据这种情况，经初步的需求分析得出来表 4-1 所示整个游戏系统说明。

表 4-1 游戏各系统说明

名称	说明
主菜单系统	用户对开始新游戏、游戏选项设置、查看游戏介绍、退出
商店系统	管理游戏中所有道具的生成
胜负判定系统	挑战管卡结束胜负判断，并产生奖励
奖励系统	用户可以通过完成订单的订单数量，领取相应的奖励

4.2 游戏系统模块设计

由于经营类游戏 TOP 需实现的功能较多，包括场景加载，读取和存取相关的数据，查看功能、设置等等。因此，为了提高开发该游戏的效率和方便管理游戏工程项目与游戏功能，游戏系统将采用设计原理——模块化的设计原理，即把游戏系统按照不同功能进行分块。根据游戏需求分析，在满足游戏基本功能的需求基础上，为了使游戏符合软件开发的高内聚，低耦合原则，即提高各个模块之间的独立性，各模块之间功能又密切相关。根据游戏系统用例建模，设计该游戏的目标系统，以玩家进入游戏为例。

游戏用例图如 图 4-1

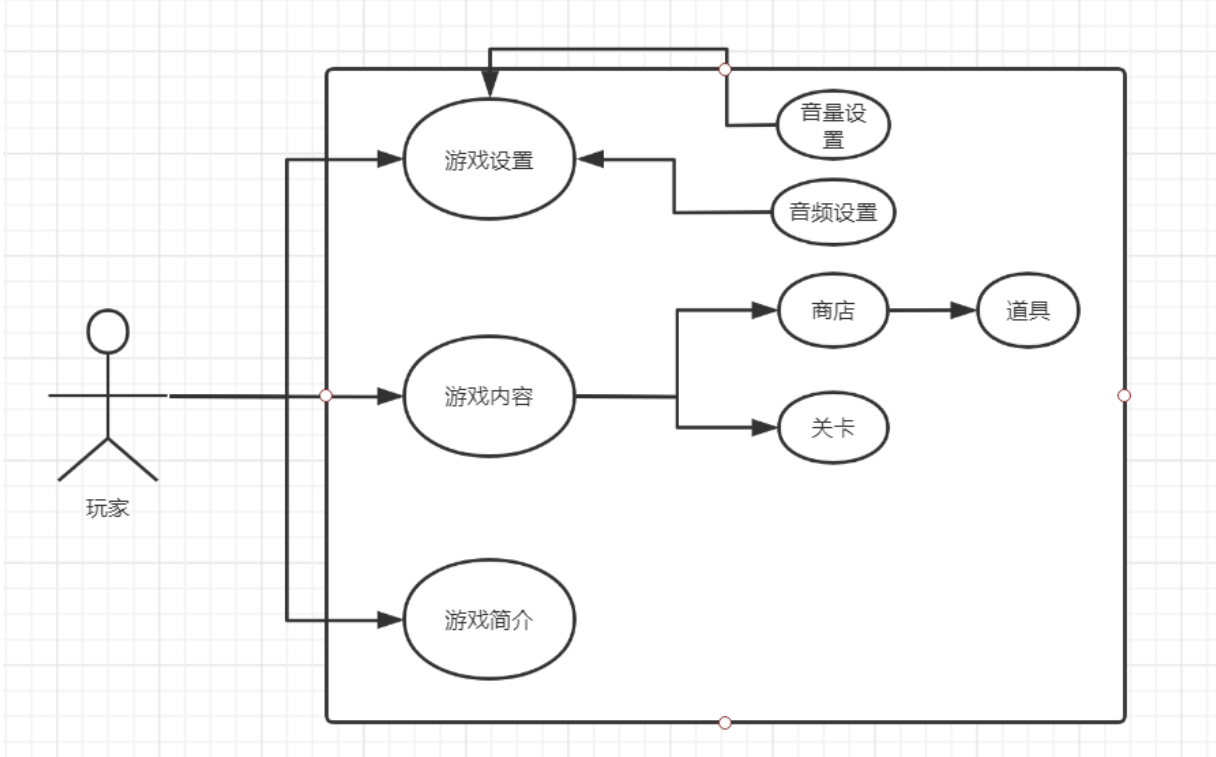

图 4-1 游戏用例图

4.3 数据模型设计

4.3.1 人物订单

人物订单分析，当这一关需要的数据超过我们想要的数据时候，就将其通过这关人物完成。这样设计可以将所有的订单数据集合在我们的核心脚本中，可以判断这关的所有情况。

表 4-2 任务订单表

数据类型	数据名字	实现的操作
float	LevelTime	//这一关用到的时间;
float	LevelNeedPeopleNum	//这一关需要完成的数量;
float	LevelproducePeopleTime	//产生人物的时间。
float	LevelSHOPTime	//商品时间

4.3.2 关卡数据类型。

关卡数据总共有三关每关根据不同的数量，进行不同的派单，让其有更多的丰富性。可以自行组装。这样设计可以将所有的关卡内需要的数据集合脚本中，判断这关的所有情况。

表 4-3 关卡数据表

数据类型	数据名字	实现的操作
float	CurrentTime	//这一关用到的时间;
float	NeedPeopleNum	//这一关需要完成的数量;
float	CurrentproducePeopleTime	//产生人物的时间。

4.3.3 商品数据类型。

在 Core 中，存储有核心的 MHaveItem 结构体，可以将他的所有商品加载在内存中。商品价格填写在脚本中，可以进行购买。做成类型核心控制的脚本，可以将所有的关键核心词放在我们的脚本中，方便操作。

表 4-4 商品数据表

数据类型	数据名字	实现的操作
int	CurrentGold	//当前金币
int	ItemSpeed	//加速道具
int	ItemOver	//立即完成订单道具
int	ItemAddNum	//增加服务员道具
int	CurrentNpcNum	//当前 NPC 的数量

第五章 游戏系统实现

5.1 UI 设计模块实现

5.1.1 UI 设计简介

UI 设计是指对在与软件进行交互的界面的整体设计。在人机交互、按钮界面的 UI 设计是重点关注。当我们制作一个游戏，需要游戏独特、有辨别性，游戏的操作能够符合用户群体的需求，满足其需求的条件下同时充分体现一个游戏的定位和特点，就需要优秀的 UI 设计。

关于 UI，Unity 中自带 UGUI 可用于 UI 的设计，但根据游戏系统用例建模分析游戏功能模块涉及到 UI 较多，而且拥有完整的 UI 动画设计特效。

本游戏在界面设计上采用了层次化的思想，把整个游戏中的全部界面独立开来，利用 UGUI 进行可视化界面设计，可即时查看到界面设计，对整个界面及时进行调整，不需要在程序调试时才可以看到游戏界面，大大提高了开发效率和维护成本。

美观符合游戏主题的界面能够吸引用户，并且当使用 UI 实现了游戏操作方便，更容易上手体验。同时还能够避免用户在体验时出现太大的不适感，视觉上的效果越好便越利于用户的理解和使用。通常情况下，UI 界面设计时能够反映用户的需求，并不是开发人员特意为之。所有的操作的设计都是以用户体验为出发点，因为所有的用户都拥有自己的理解和使用方法。界面整洁美观，游戏整体风格与产品级别便越高。

5.1.2 开始界面 UI 设计

创建 UGUI 中 2D UI 组件，在 Hierarchy 视图下 UI Root 中 Camera 创建子对象面板 Canvas、Canvas，Canvas 面板用于存储开始游戏按钮，声音调节按钮和设置按钮；Panel2 面板为玩家信息操作面板，在此面板上，玩家可以对玩家信息进行读取、删除、新建操作。点击开始游戏按钮，触发按钮事件，从 Panel 面板切换至 Panel2 面板，代码实现如下：

```
    public StartUI m_StartUI;
[System.Serializable]
public struct StartUI
{
public Button DesGameBtn;
public Button StartGameBtn;
public Button QuitGameBtn;

public GameObject DesObj;
```

```
public Button DesQuitBtn;
}

private void Start()
{
m_StartUI. DesGameBtn. onClick. AddListener(delegate
{ m_StartUI. DesObj. gameObject. SetActive(true); });
m_StartUI. DesQuitBtn. onClick. AddListener(delegate
{ m_StartUI. DesObj. gameObject. SetActive(false); });

if (Core. Instance)
{
m_StartUI. StartGameBtn. onClick. AddListener(delegate
{ Core. Instance. LoadScene("02"); });
m_StartUI. QuitGameBtn. onClick. AddListener(delegate { Application. Quit(); });
}
}
}
```

图 4-1 开始界面菜单

点击开始游戏，通过异步加载，从场景 Start 加载到场景 Menu。加载菜单 Menu 场景代码实现如下：

```
Application.LoadLevel("Menu");
```

5.1.3 菜单界面 UI 设计

菜单界面各功能 UI 设计可以归为两类：1. 游戏简介，开始游戏和退出游戏，这三个系统的显示都需要按序按位置排列，并调用配置文件对每个对象属性进行赋值；2. 关卡系统与其他功能，关卡系统设计方式是将所有关卡按钮名称统一格式命名，在点击某个按钮时，传入按钮名称，触发对应按钮事件。选择商店系统中 UI 商品物品 Item 和关卡系统选关 UI 加以详细描述，其余就不一赘述了。

UGUI 实现等大等距排序显示运用到的组件：Scroll View 组件和 Grid 组件。Scroll View 组件作用是设置一块可视区域，在 Grid 组件下的子对象超过其可视区域就无法在界面中显示，并可以实现子对象从左到右排列。考虑后续维护和拓展的简便，将 UI 商品物品 Item 展示模型制作成预制体，通过代码加载到 Grid 组件下作为子对象，并加载配置文件数据。如图 4-1

菜单面板设计如图 4-2。

数据管理加载如图 4-3。

```
public class StartPanel : MonoBehaviour
{
    public StartUI m_StartUI;

    [System.Serializable]
    1 个引用
    public struct StartUI
    {
        public Button DesGameBtn;
        public Button StartGameBtn;
        public Button QuitGameBtn;

        public GameObject DesObj;
        public Button DesQuitBtn;
    }

    0 个引用
    private void Start()
    {
        m_StartUI.DesGameBtn.onClick.AddListener(call: delegate { m_StartUI.DesObj.gameObject.SetActive(true); });
        m_StartUI.DesQuitBtn.onClick.AddListener(call: delegate { m_StartUI.DesObj.gameObject.SetActive(false); });

        if (Core.Instance)
        {
            m_StartUI.StartGameBtn.onClick.AddListener(call: delegate { Core.Instance.LoadScene("1"); });
            m_StartUI.QuitGameBtn.onClick.AddListener(call: delegate { Application.Quit(); });
        }
    }

    // Update is called once per frame
    0 个引用
    private void Update()
    {
    }
}
```

图 4-1 菜单界面

```
Assembly-CSharp                                                    MenuPanel

    [System.Serializable]

    public struct MenuPanelUI
    {
        public Button MenuBtn;
        public GameObject MenuObj;
        public GameObject DesPanelObj;

        public Button CloseMenuBtn;
        public Button DesBtn;
        public Button QuitBtn;
        public Button CloseDesBtn;

        public Slider MusicSlider1;
        public Slider MusicSlider2;

        public Button ChangeMusicBtn;
        public Text MusicName;
    }


    private void Start()
    {
        m_MenuPanelUI.MenuBtn.onClick.AddListener(call:delegate { m_MenuPanelUI.MenuObj.gameObject.SetActive(true); });
        m_MenuPanelUI.DesBtn.onClick.AddListener(call:delegate { m_MenuPanelUI.DesPanelObj.gameObject.SetActive(true); });
        m_MenuPanelUI.CloseMenuBtn.onClick.AddListener(call:delegate { m_MenuPanelUI.MenuObj.gameObject.SetActive(false); });
        m_MenuPanelUI.CloseDesBtn.onClick.AddListener(call:delegate { m_MenuPanelUI.DesPanelObj.gameObject.SetActive(false); });
        m_MenuPanelUI.QuitBtn.onClick.AddListener(call:delegate { Application.Quit(); });

        if (Core.Instance)
        {
            m_MenuPanelUI.MusicSlider1.onValueChanged.AddListener(call:delegate (float value) { MusicManager.Instance.SetAudioSourceVOLUME1(value); });
            m_MenuPanelUI.MusicSlider2.onValueChanged.AddListener(call:delegate (float value) { MusicManager2.Instance.SetAudioSourceVOLUME1(value); });
            m_MenuPanelUI.ChangeMusicBtn.onClick.AddListener(call:delegate () { m_MenuPanelUI.MusicName.text = MusicManager.Instance.SetAudioSourceClip(); });
            m_MenuPanelUI.MusicName.text = MusicManager.Instance.ReturnMusicName();

            m_MenuPanelUI.MusicSlider1.value = MusicManager.Instance.AudioSource.volume;
            m_MenuPanelUI.MusicSlider2.value = MusicManager2.Instance.AudioSource.volume;
        }
    }
```

图 4-2 菜单面板设计

```
    32 个引用
public class Core : MonoBehaviour
{
    public static Core Instance;

    public HaveItem MHaveItem;//当前有的题目数量

    [System.Serializable]
    1 个引用
    public struct HaveItem
    {
        public int CurrentGold;//当前金币
        public int ItemSpeed;//加速道具
        public int ItemOver;//立即完成订单道具
        public int ItemAddNum;//增加服务员道具
        public int CurrentNpcNum;//当前NPC的数量
    }

    0 个引用
```

图 4-3 数据管理加载

在场景加载中，存储着这些人物属性道具，以及道具数据等等。

5.2 数据管理管理模块实现

在模拟经营游戏中，数据的读取与存储是最为重要的，游戏中数据包括静态数据和

动态数据，静态数据即为游戏中固定的某些数值，如地图关卡数据，服务员属性数据，外卖员属性数据等；动态数据也称为玩家数据，包含玩家花钱购买的游戏币（金币），关卡的解锁和玩家获得的星级等。

5.2.1 静态数据管理

关于静态数据，Unity3D 工程中使用配置文件一般使用 Txt、Xml、Json、Db(数据库)，四种格式文件对比如下:

Txt: 开发人员必须设定格式、封装读取的方式。设定与封装因人而异，重点是只能够读取。

Xml:标签语言，已规定格式，系统中提供读取和修改等相关 API,操作方便。

Json:采用键值对的方式去进行数据的存储，可在任何平台下使用，普遍被客户端与服务端通讯采用，存储相等数据信息下文件要比 Xml 文件小，还可以只进行值存储（不需要键），更加缩小了文件的体积，适合在移动端进行联网游戏的通信操作。

db: 适合复杂数据和敏感数据，增删该查非常方便。

根据本设计的需求，选用了 Xml 格式作为配置文件格式，方便信息数据和地图数据的加载。Xml 格式的配置文件一般放在由 Untiy 自主管理模式的 Resources 文件夹中，在此文件夹下当需要读取资源可通过 Resources. Load()读取。通常所有资源都可读取。Unity 读取 Xml 文件的步骤如下：

(1) 新建一个 TextAsset 类型的成员 MapXml，调用 Resources. Load()方法从 Resources 文件夹下读取相应的 Xml 文件的数据。例如：

```
MapXml = Resources. Load<TextAsset>("XML/Map");
```

(2) 创建 XmlDocument 新对象 xmlDoc，关联并解析 MapXml. text。

(3) 获取列表数值,获取子节点。例如：

```
mapList = xmlDoc. SelectSingleNode("MapConfig/Map"). ChildNodes;
```

如图 4-4

```
18 个引用
public class FightManager : MonoBehaviour
{
    public static FightManager Instance;

    public Level m_Level;

    [System.Serializable]
    1 个引用
    public struct Level
    {
        public float LevelTime;//这一关用到的时间;
        public float LevelNeedPeopleNum;//这一关需要完成的数量;
        public float LevelproducePeopleTime;//产生人物的时间。
        public float LevelSHOPTime;//商品时间
    }

    [HideInInspector]
    public CurrentLevel m_CurrentLevel;

    [System.Serializable]
    1 个引用
    public struct CurrentLevel
    {
        public float CurrentTime;//这一关用到的时间;
        public float NeedPeopleNum;//这一关已经完成的数量;
        public float CurrentproducePeopleTime;//产生人物的时间。
    }

    public Transform[] ClientPostion;
```

图 4-4 数据管理

5.3 服务员寻路模块实现

5.3.1 寻路网格设置

1. NavMesh(导航网格)是 Unity3D 自带的一种自动寻路技术，可以在开发的虚拟游戏世界中实现动态物体自动寻路，该寻论技术能够将游戏中复杂的结构组织关系转化成带有一定信息的网格，在这些网格的基础上系统计算后得出最优路线，以此来实现自动寻路。简单来说就是为需要的物体添加 NavMeshAgent 组件，并设置适当的参数。一般需要设置物体的半径、高度、速度、加速度、距离目标的停止距离。当需要这个物体进行导航移动时，我们需要给导航物体添加一个条件，即挂载导航组件，这个导航物体将会自动寻路他计算出最合适的路线，并沿该线路到达我们设定的目标点。

2. 点击进行寻路的物体，并添加在 Unity3D 右侧的导航栏选中 Component——Navigation——Nav Mesh A gent 组件，然后在导航栏打开 Window——Navigation 窗口，在 Navigation 窗口选择 Bake，选择后我们需要再次选择右下角的 Bake。

3. 在每一个服务员身上挂载 NavMeshAgent 进行烘培场景，让 AI 可以自动寻路。如图 4-8 所示：