软件设计说明书

Contents

1 引言

1.1 目的

本文档编写的目的是为了规范软件代码的结构等方面，防止在之后的开发中出现软件结构混乱、命名冲突等问题。

1.2 命名规范

代码的命名和编写规范见 这篇关于命名规范的文档

1.3 定义

- N/A: Not Applicable，无、不适用
- 玩家: 指游玩此游戏的人
- 系统/程序: 指游戏本身
- Unity: Unity3D 游戏引擎
- mod、模组: 玩家为了改善/改变游戏体验自行向游戏添加的插件、资源等

1.4 参考资料

1. Unity Scripting Reference (https://docs.unity3d.com/ScriptReference)
2. Unity Manual (https://docs.unity3d.com/Manual)
3. Microsoft Docs (https://docs.microsoft.com/zh-cn/)

1.5 相关文档

1. 开发需求说明书 (https://github.com/01010101lzy/software-engineering-simulator/blob/master/docs/homework/spec.md)
2. 软件开发计划书 (https://github.com/01010101lzy/software-engineering-simulator/blob/master/docs/homework/dev_plan.md)

1.6 版本历史

见本文件在 GitHub 的版本历史。

2 总体结构设计

见图 1 和图 2 。

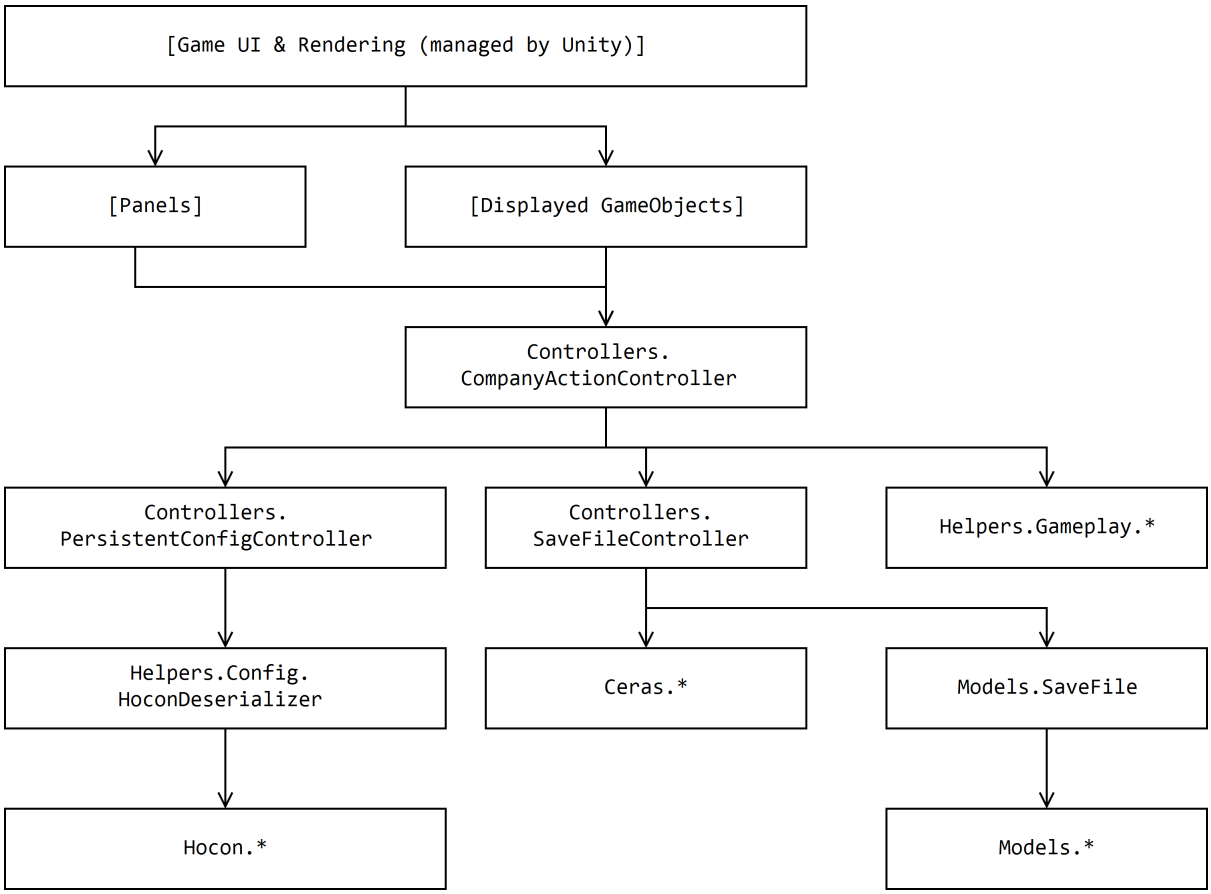

图 1: 游戏界面的结构与依赖关系

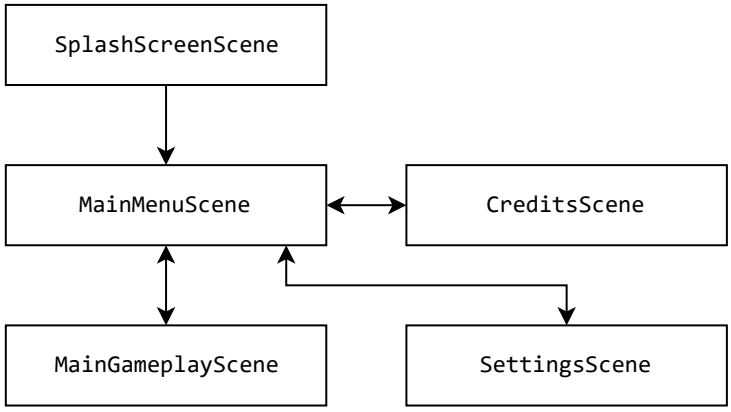

图 2: 游戏中的场景关系

3 场景和控制器设计

3.1 场景设计

场景（Scene）是 Unity 中对于游戏中不同场景的抽象。场景列表见表 1。

表 1: 场景一览表（部分未实现）

场景	用途
MainGameplayScene	游戏主界面
MainMenuScene	游戏主菜单
SplashScreenScene	加载界面
SettingsScene	设置界面
CreditsScene	工作人员列表界面

3.1.1 MainGameplayScene

MainGameplayScene 是游戏的主界面，设计图见图 3 。主要包含以下部分：

- 最左侧的面板切换菜单，用于切换信息面板所显示的内容，以及暂停游戏；
- 左侧的信息面板，用于显示人员、任务、资金、统计等方面的详细信息；
- 右侧的概览面板，用于显示玩家控制的公司的概况（如资金、声望等），以及游戏内的通知；
- 中间的公司视图，以可视化的方式展示公司内当前的状态（如员工是否工作），并允许玩家与公司内的元素互动。

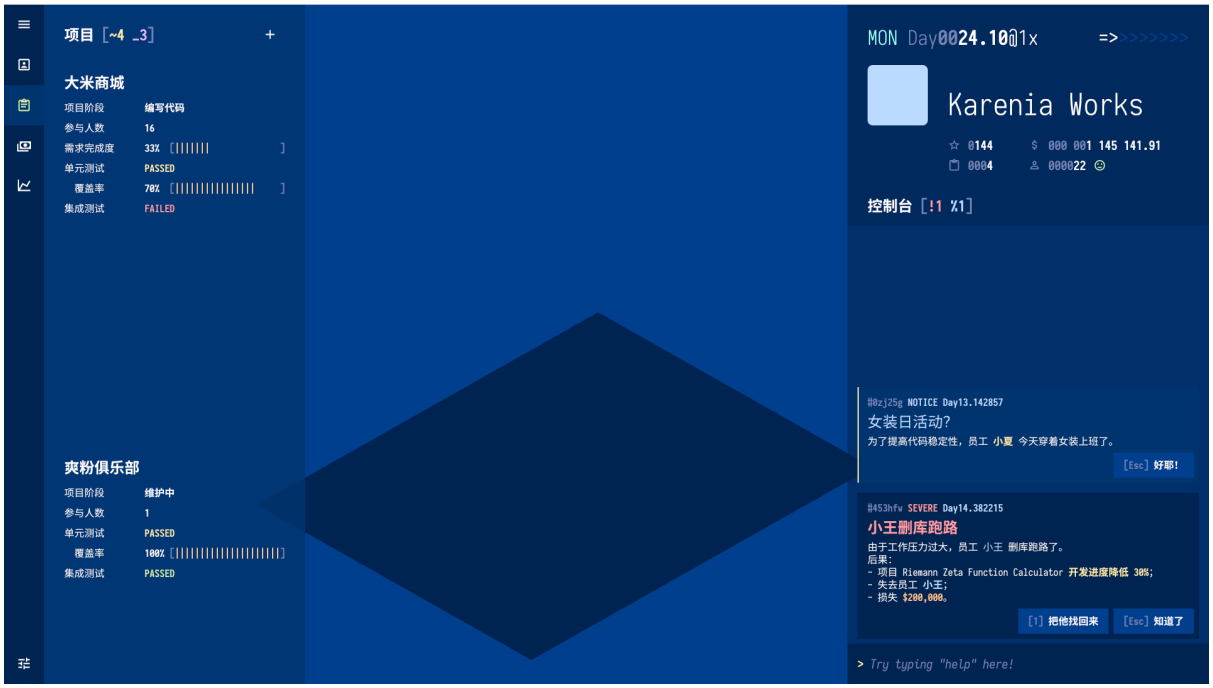

图 3: 游戏主界面（设计稿，暂定）

3.1.2 MainMenuScene

MainMenuScene 是游戏的主菜单，设计图见图 4 。用户可以在此执行以下操作：

- 新建游戏存档
- 加载已有的存档
- 修改游戏设置
- 查看制作人员名单
- 退出游戏

图 4: 游戏主菜单（设计稿，暂定）

3.1.3 SplashScreenScene

SplashScreenScene 是游戏的加载界面。在显示加载界面时游戏在后台加载游戏的资源和/或存档文件。加载界面使用一个类似命令行的界面显示加载进度条和正在加载的内容。

3.1.4 SettingsScene

SettingsScene 是游戏的设置界面。由于截至本文档的最近一次更新时，游戏的设置项内容还未确定下来，所以本场景的设计还未确定。

3.1.5 CreditsScene

CreditsScene 在游戏中展示制作人员信息。由于本场景的优先级很低，所以截至本文档的最后一次更新时其设计还未确定。

3.2 控制器设计

控制器是直接或间接控制游戏物体的对象。控制器列表见表 2 。

表 2: 主要控制器表（部分还未实现）

类名[1]	用途
MainGame.CompanyActionController	进行游戏每帧的状态更新等工作
MainGame.*PanelController [2]	控制游戏中的面板显示数据
Persistent.PersistentUIController	用于控制在场景切换时不被销毁的界面元素
Persistent.SaveController	用于管理存档文件
SplashScreen.MainMenuController	用于管理主菜单
SplashScreen.SplashScreenController	用于控制加载界面
SplashScreen.SettingsController	用于控制设置界面

3.2.1 MainGame.CompanyActionController

这个控制器负责在游戏运行的每一帧更新游戏中员工、人物的状态，产生随机事件，并控制屏幕上显示的人员状态。

3.2.2 MainGame.*PanelController

这一系列控制器控制屏幕上各个面板中的数据显示，并提供对用户在面板上操作的反馈。

3.2.3 Persistent.PersistentUIController

这个控制器控制在场景切换时不被销毁的界面元素，如控制台等。

3.2.4 Persistent.SaveController

这个控制器管理存档与快速存档，负责将游戏状态序列化（存档）和反序列化（读档），也负责获取所有存档的元数据。

3.2.5 SplashScreen.MainMenuController

这个控制器控制用户在主菜单界面的行为和导航。

3.2.6 SplashScreen.SplashScreenController

这个控制器负责在游戏加载时前台显示加载进度、后台读取以及反序列化配置文件。

[1]所有控制器都在 Sesim.Controllers 命名空间下。
[2]指以 PanelController 结尾的所有面板控制器，下同。

3.2.7 SplashScreen.SettingsController

这个控制器负责管理游戏的设置界面。

3.3 辅助模块设计

辅助模块辅助控制器进行操作，和/或提供对常用操作的抽象处理。辅助模块列表见表 **??** 。

表 3: 主要辅助模块表（部分未实现）{#tbl:helpers}

类名[3]	用途
Ceras.*Formatter	辅助部分对象到二进制的序列化和反序列化该能
Config.*	提供从 HOCON 格式反序列化到对象的功能
UI.ColorScheme	提供对提供配色方案的支持
UI.ConsoleHelper	提供模拟命令行时用到的常用方法

4 存档结构设计

SESim 的存档使用 Ceras 序列化器序列化为二进制存储。存档使用的序列化/反序列化器支持在以后的更新中添加或删除键值，所以此表的内容在开发过程中还有可能继续修改。每一个存档被存储为两个部分：元数据 SaveMetadata 和存档本身 SaveFile。存档各部分结构如表 4, 5, 6, 7, 8, 9, 10 所示:

表 4: 存档元数据 SaveMetadata 结构

变量	类型	说明
version	long	游戏版本
id	Ulid	的唯一标识符
name	string	标题
ut	int	游戏内时间
fund	decimal	资金
reputation	float	声望
employeeCount	int	员工数
contractCount	int	任务数

表 5: 存档数据 SaveFile 结构

变量	类型	说明
version	long	游戏版本
id	Ulid	唯一标识符
name	string	标题
company	Company	模拟公司信息
settings	DifficultySettings	难度设定（尚未确定）

[3]所有辅助模块均在 Sesim.Helper 命名空间下

表 6: 公司信息 Company 结构

变量	类型	说明
name	string	公司名
ut	int	游戏时间（一小时为 300 tick）
fund	decimal	资金
reputation	float	声望
avaliableContracts	List<Contract>	可接任务列表
contracts	List<Contract>	任务列表
avaliableEmployees	List<Employee>	可招聘员工列表
employees	List<Employee>	员工列表
workTimes	List<WorkPeriod>	公司作息
extraData	Dictionary<String, dynamic>	为模组预留的数据存储

表 7: 任务信息 Contract 结构

变量	类型	说明
id	Ulid	唯一标识符
name	string	名称
status	ContractStatus	状态
description	string	描述
contractor	string	甲方名称
difficulty	float	难度
hasExtendedMaintenancePeriod	bool	是否有维护期
startTime	int	开始时间
liveTime	int	在任务表中存活时间
timeLimit	int	结束时间
extendedTimeLimit	int	维护期结束时间
members	List<Employee>	负责任务的员工
techStackPreference	Dictionary<string, float>	偏好的技术栈
techStack	string	玩家选择的技术栈
totalWorkload	double	总工作量
completedWork	double	已完成工作量
depositReward	ContractReward	接单奖励
completeReward	ContractReward	完成奖励
maintenanceMonthlyReward	ContractReward	维护期每月奖励
breakContractPunishment	ContractReward	违约惩罚
completeCondition	ICompleteCondition	完成条件
extraData	Dictionary<String, dynamic>	为模组预留的数据存储

表 8: 员工信息 Employee 结构

变量	类型	说明
id	Ulid	唯一标识符
name	string	姓名
experience	float	经验
baseEfficiency	float	基础效率

变量	类型	说明
abilities	Dictionary<string, float>	不同技术栈的技能
salary	decimal	工资
health	float	健康度
pressure	float	压力大小
lastWorkTime	int	上次工作开始时间
isWorking	bool	是否在工作
position	Vector3	在公司视图中的位置
rotation	Quaternion	在公司视图中的旋转
efficiencyTimeCurve	AnimationCurve	效率-工作时间关系曲线
efficiencyHealthCurve	AnimationCurve	效率-健康度关系曲线
efficiencyPressureCurve	AnimationCurve	效率-压力大小关系曲线
extraData	Dictionary<string, dynamic>	为模组预留的数据存储

表 9: 公司作息时间 WorkPeriod 结构

变量	类型	说明
start	int	工作开始时间
end	int	工作结束时间

表 10: 任务奖励 ContractReward 结构

变量	类型	说明
fund	decimal	资金增加量
reputation	float	声望增加量

5 配置文件设计

在游戏设计稳定之后，玩家可以使用配置文件自定义游戏的行为。配置文件应当放在游戏根目录 GameData 文件夹下，以 UTF-8 字符集、.conf 扩展名的纯文本文件存储，并采用 HOCON 格式编写。每一个配置文件均会被解释成对单个对象的配置。对于 HOCON 格式的介绍说明见其作者在 GitHub 发布的文档。由于截至本文档最后一次编辑时，配置文件的设计还没有定稿，所以以下内容均为提案。

下面列出了两份配置文件的示例：

```
// 这是一份定义订单生成器的配置文件（节选）
// 配置文件类型标注
$type: contractFactory
// 配置的键-值对
name: androidMarketApplication
category: applicationContract
title: "Make a market app in Android for SomeCompany"
// -- snip --
difficulty-multiplier: {
  // 配置可以按照定义嵌套
```

```
  method: exponential
  fund: 1.08
  reputation: 1.21
}

// 这是一份查找并修改符合要求的配置的文件示例
$find: {
  // 定义查找条件
  $type: contractFactory
  category: applicationContract
  duration: {
    // select those shorter than 30 days
    easy: { $lt: 30 }
  }
  limit: 1
}
$do: {
  duration: {
    // 对配置进行修改
    easy: { $inc: 15 }
  }
  modules: {
    // 查找、修改的操作可以嵌套
    $find: { name: exampleModule }
    $do: { exampleValue: 1 }
  }
}
```

对于在一个文件内定义多个配置的情况，用户可以在配置文件的最底层使用数组语法定义多个对象，如下：

```
[
  {
    $type: something
    // -- config 1 --
  }
  {
    $type: somethingElse
    // -- config 2 --
  }
]
```

6 其他设计

6.1 接口设计

为了加强游戏的可扩展性，游戏中各种事件发生处都将预留事件 (Event)，方便玩家在为了改变游戏行为在此处增加回调函数调用玩家自己的插件。

另见上方的配置文件设计。

7 错误处理机制

如果游戏某项功能在运行过程中遇到错误，错误将被 Unity 的记录系统记录到日志中，并在屏幕上显示相应错误记录。如果错误十分严重、会阻碍游戏继续运行，游戏将会弹窗显示错误并立即退出。

8 测试计划

本项目的单元测试和集成测试代码均位于 `Assets/Test` 目录下。

截至本次文档更新时，开发组已在 Travis CI 上设置了测试流水线。流水线在每一次代码提交之后都会对当前提交的版本进行单元测试和集成测试，并自动返回测试结果。Travis CI 上的测试结果公开于 这个链接。