项目目录结构 2

一.配置表工具使用说明 4

1.1配置表格式 4

1.2导出配置表 4

1.3创建对应数据类 4

1.3配置表的使用 4

1.3.1加载配置表 4

1.3.2调用数据 5

二.事件驱动系统说明 6

2.1文件说明 6

2.2使用 6

2.1.1 添加新事件 6

2.1.2 发送事件 6

2.1.3 接收事件 6

三.数据存储 8

3.1文件说明 8

3.1使用 8

四.场景道具及场景数据 9

4.1文件结构 9

4.2增加道具 9

4.3载入和保存关卡 9

五.场景道具及场景数据 10

项目目录结构

3rd party：第三方插件 如NGUI

code：游戏内相关代码

Base：定义了游戏内一些常用基础类

Config：游戏配置表管理器，负责配置表的读取和访问

Editor：编辑器相关代码

Event：游戏事件管理器

GameData：游戏数据相关

GameManager：游戏管理相关

InGame：游戏内相关

AnimationManager：游戏角色动画管理器

InGameEvent：游戏内事件管理器

InGameUI：游戏内UI

MapObject：游戏中的物体（场景道具，敌人，主角等）

ObjAI：AI

Net：这里定义游戏与数据层或服务器的相关协议已经请求和响应

Server：这里管理游戏的请求去向

Test：一些临时测试类

Tools：一些常用代码、定义等

AH：加密的数据类型

JoyStick:摇杆类

Stream：数据流

ExcelTools：Excel导出工具

Plugins:第三方库

Resources：游戏资源

Config:游戏配置表文件

LevelData：关卡数据文件

Models：美术模型文件

Prefabs：游戏中的预制体

scenes：游戏场景

一.配置表工具使用说明

## 1.1配置表格式

游戏配置表第一行为字段注释，第二行为字段名称，第三行开始为内容。

一个excel文件中，每一个工作表会导出为一个配置文件，文件名为工作表名称。

## 1.2导出配置表

1.将excel文件放到 工程目录->Assets-> ExcelTools-> xlsx中。

2.在菜单栏中选择 配置表工具/生成配置文件。在文件列表中选择要导出的配置文件即可。

3.生成的配置文件保存在/工程目录->Resources/Config/中。

配置表文件经过了GZIP压缩存储于二进制文件中。

## 1.3创建对应数据类

1.在ConfigClass脚本中创建对应的类，类名应该与表名保持一致。

2.该类需要继承自ConfigClass，并且使用[Serializable]对该类进行标记。

3.类中需要实现父类中的GetId()方法，该方法返回主键，即存储到字典中的key。

使用中需要根据此id获取对用数据

## 1.3配置表的使用

### 1.3.1加载配置表

ConfigManager需要挂载到游戏物体上，并且不能被销毁。

为了尽量适应较为复杂的情况，配置表管理器存储数据的结构为：

Dictionary<string, Dictionary<string, List<ConfigClass>>>；

红色部分封装到类ConfigClassDic中。

第一层key为表名，第二层key为类中GetId方法指定的id，一个id可以有多条数据，存储在list中。

### 1.3.2调用数据

ConfigManager 中已封装获取配置表数据的方法

Example:

id唯一: ConfigManager.instance. GetConfByName ("MapObjectConf ")["id"][0];

返回ConfigClass。

id不唯一: ConfigManager.instance. GetConfByName ("MapObjectConf ")["id"];

返回List<ConfigClass>。

获取到数据后需要自行强转一下类型。

二.事件驱动系统说明

## 2.1文件说明

EventData :游戏中的所有事件继承自此类。暂时将所有事件定义在EventData脚本中。

EventID:为游戏中每一个事件分配一个唯一id。

EventManager:事件管理器。

EventObserver:监听者接口，所有需要监听事件的类要实现此接口。

## 2.2使用

### 2.1.1 添加新事件

1.在EventID中为新事件分配一个id

2.在EventData中定义实现新事件的数据结构，并在新事件的构造函数中赋予eventid。

### 2.1.2 发送事件

实例化一个具体的事件对象:

EventEntryGame eve = new EventEntryGame();

为实例属性赋值：

eve.uname = "aa";

eve.password = "123456";

发送:

eve.Send();

### 2.1.3 接收事件

一个类想要接收事件必须要实现EventObserver中的HandleEvent方法。

并且注册到消息管理器中，注册时需要指定要监听的事件id。

三.数据存储

## 3.1文件说明

所有用户数据由UserDataManager类进行管理。外部访问或操作数据必须经过此类。

UserData类用于在游戏过程中存储在内存中，UserDataManager即对此类进行读取操作。此类中的数据类型均为已加密类型。

UserDataMode用于存储文件使用，UserDataManager对数据操作之后需要存储到文件，需用根据UserData来生成UserDataMode进行存储。

## 3.2使用

UserDataManager.instance.Start();//初始化，游戏启动时调用

每一个数据都要在UserDataManager中有相应的获取和修改方法。每次修改之后都要调用SaveData方法对数据进行存储。

例:UserDataManager.instance.GetUserid();

四.场景道具及场景数据

## 4.1文件结构

此部分内容设计的脚本都在Code/Game/InGame中。

Map:关卡数据存储，管理，游戏内关卡控制器等都要放在这里。

MapObject:场景中物体的基础类，以及所有物体放在这里。游戏中所有互动物体，包括地面，道具，障碍，敌人，主角，都在这里实现，并且都要继承自BaseMapObject。

场景中的物体尽量都创建自己独立的脚本，不要直接使用BaseMapObject。

## 4.2增加道具

1.在GameConst脚本中增加对应的枚举类型enMSObjID作为道具id。

2.创建对应脚本，实现相应功能，实现序列化和反序列化功能

3. 制作该道具的预制体，

4. 在关卡道具配置表中增加对应的道具，id要与第一步中增加的枚举类型一致。

## 4.3载入和保存关卡

1 保存：

MapData类中包涵2部分内容，MapProperty为关卡基本信息，包括ID，名称，使用的背景图片等内容。objlist为场景内所有物体的列表。

在保存关卡前，需要先整理好MapData的全部数据，然后直接调用存储方法即可：

MapEditor.SaveLevelData(md);

2载入:

直接调用载入方法即可: MapData md = MapEditor.LoadLevel("test1");

此方法会返回MapData数据，包涵关卡的全部信息，此时道具列表中的道具已经全部添加到场景内了。

五.其他说明

游戏中创建的所有unity类需要继承BaseUnityObject或其派生类。

游戏中创建的非unity类需要继承BaseGameObject或其派生类