Slots程序介绍

Table of Contents

介绍 3

场景 3

Excel数据配置 3

UserData 3

Core层 4

Puzzle层 6

Common模块 8

UserData和服务端的同步 9

资源加载管理 10

热更 10

商店内购 11

第三方SDK 11

打包流程 12

Jenkins打包 13

音频资源 14

美术资源 14

# 介绍

工程目录介绍

代码以外的所有资源目录介绍

# 场景

3个scene：

（1）开始Loading场景，入口类StartLoading

（2）大厅场景，入口类MapScene

（3）机台场景，入口类GameScene

# Excel数据配置

1 在第三方插件QuickSheet上改进和定制化

2 将Excel配置导出成.asset文件，程序包使用asset文件

3 自动生成代码：定义配置文件，插件自动解析Excel格式，自动生成Data和Sheet代码

4 自动导出配置：ExcelAssetPostprocesser自动检测Excel文件的改动，然后自动导出

5 QuickSheet UML文档：Trojan/Documents/QuickSheet

# UserData

1 用插件Easy Save 2

2 各种UserData

(1) UserBasicData：用户基本数据，会传服务端保存

(2) UserMachineData：和每个机台相关的用户数据，会传服务端保存

(3) UserDeviceLocalData：用户基本数据，只存本地，不存到服务端

新加逻辑时，注意和策划讨论数据是只需要存本地还是需要存入服务端

3 千万注意保持UserData的向前和向后的兼容性，这两种case都得保证数据能正常解析和程序正常运行：

（1）新版本程序使用老版本格式UserData

（2）老版本程序使用新版本格式UserData

4 注意每增加一个字段到UserBasicData或者UserMachineData里，需要相应地增加传服务端的逻辑在UserBasicDataJSON和UserMachineDataJSON

5 用户进度上传下载功能比较重要，后面再单独说

# Core层

1 纯逻辑层，不包含任何表现层逻辑。代码上只使用了C#核心接口，没有使用任何Unity接口

2 Core和上层代码完全隔离，所以Core可以独立抽取出来单独编译成Dll，脱离于Unity做大规模数值测试。以后任何改动和加功能都需要保证Core的纯粹性

机台配置

1 CoreSetting配置

2 单线机台Excel配置规则

3 多线机台Excel配置规则

核心模块

最上层模块：CoreMachine

基本结构：CoreMachine -> CoreReel -> CoreSymbol

Spin流程

输入：CoreSpinInput

CoreMachine.Spin() -> CoreGenerator.Roll() -> CoreSpinResult.Recheck() -> CoreChecker.CheckResultWithSymbols

输出：CoreSpinResult

每次Spin过程中的数据：CoreSpinData，每种小游戏都有对应的一个CoreSpinData子类

单线机台生成算法：Win，NearHit，Loss

多线机台生成算法：重点是Excel的PayoutDist和RewardResult表

随机数种子算法

1 Utility.Random：LCG随机数生成算法。Core层不调用C#系统Random函数

2 每个机台初始种子一旦确定，后面的结果就都确定了。所以我们通过控制初始种子值控制新玩家的前期体验

小游戏模块

PlayModule：各种和机台高度耦合的小游戏，如FreeSpin，FixWild

PlayModule系统是一个状态机

1 CoreMachine里保存当前PlayModule的引用，将一些关键操作如Spin转发给当前PlayModule具体实现

2 默认状态是PlayModuleNormal，其他都是小游戏PlayModule。不同PlayModule之间的跳转逻辑封装在各自当前的PlayModule里

Jackpot：Jackpot大奖玩法

IndieGame：各种完全独立于机台玩法的小游戏，如TapBox，Wheel

CoreLuckyManager：管理Lucky逻辑

Utility：

Core层工具类

ListUtility：基于List的各种操作，包括：

1 几个函数式语言里常用的操作：map，fold，filter

2 List的创建，查询，一些Predicate的判定，计数等

3 两个List之间的求交集，相减

注意：

1 函数的参数类型尽量用接口IList，这样能同时处理Array和List

2 用ListUtility能替代绝大部分for循环做的事情，建议多用这些工具函数来简化代码

3 C#有提供Linq来完成ListUtility能做的事情，但那个以较差的效率为代价增强了代码的简洁性，原则上我们项目里禁用Linq

SimpleSingleton：

一个singleton的简单实现，另一个singleton的实现在CitrusFramework下。区别是：

SimpleSingleton没有依赖于UnityEngine，所以没有DontDestroyOnLoad这种操作。所以Core层只能用这个类

CitrusFramework的Singleton依赖于UnityEngine，有DontDestroyOnLoad调用和考虑到游戏退出时的逻辑

StringUtility：

字符串常用操作，比如在数字的千位百万位加逗号

TypeUtility：

类型相关的逻辑，比如从字符串转成一个枚举类型

自动测试MachineTest：

只需要在Core层上加很少的逻辑就能模拟玩家一直Spin的场景

种子生成器

1 每个机台有一组待选种子配置，这些种子都满足一定的筛选条件，能给玩家带来较好的前期体验

2 代码分两部分：

编辑器：Editor/MachineSeedGenerator/

运行期：Scripts/Common/MachineSeed/

注意：一个种子是好是坏，依赖于Core层的SpinResult生成算法。如果以后对算法进行改动的话，比如中间增加了某个步骤多了一个Rand操作，那么之前的好种子就可能变成坏种子，这种情况下需要对所有机台的种子重新生成一遍。程序在改Core层算法时需要注意这点

多线机台导出工具

代码目录：Assets/Editor/MultiLineExport，这个是给策划在制作多线机台时用的一个辅助工具，用于生成一些数据，最终这个数据会填到多线机台Excel文件的RewardResult表里

编译Dll工程

Core层可以独立编译成Dll文件，编译工程在svn的CoreDll/下。编译的Dll可以脱离Unity环境，用NUnit单独做大规模数值测试。但目前已有很长一段时间没有维护，以后需要的话可以用起来

保持Core层代码的高质量

由于以后会持续新增机台，扩展玩法，而且Core层要同时支持运行期，自动测试，以及以后可能的Dll测试，所以保持Core层代码的高质量非常重要。包括：

1 代码应该符合所有优秀代码的品质：清晰，简洁，易读，没有重复等

2 由于一直在新增机台，所以代码扩展性非常重要，写的时候尽量考虑到未来可能的需求变更

3 加任何功能都以最好的方式加代码，避免任何形式的补丁。有时候加一个补丁能很快实现一个功能，但为了保持良好的结构，可能需要花更多时间调整已有的代码结构。我们应该选择后者

4 把类，函数的粒度保持足够小，足够单一。工具性质的函数不写到主要的类里，而是分出来写到Helper或者Utility性质的工具类里

5 小游戏玩法之间完全独立没有耦合。理想情况下，如果一个机台里需要支持很多个小游戏，只要修改下配置，程序就能完美运行

6 独立小游戏（IndieGame）也应该严格区分Core层和Puzzle层，原因是Core层的逻辑需要加到自动测试里，在Core和Puzzle区分很好的情况下，自动测试只需要加很少量的代码就能支持小游戏，参考CoreTapBox

一个设计良好的例子

CoreGenerator的职责很单一：负责生成一个SpinResult。所以这个类虽然有近2000行代码，但重要的public函数只有一个：Roll，另外两个public函数是构造函数和一个用于Debug 模式下调试的函数。那么在修改和扩展这个类时，就不应该添加其他任何public接口以破坏它的单一性

# Puzzle层

1 负责机台玩法的表现层逻辑

2 很多模块和Core层一一对应，PuzzleMachine，PuzzleReel，PuzzleSymbol，Puzzle层的IndieGame

3 Puzzle层的模块划分不像Core层那么细，有很大重构空间，比如PlayModule在Puzzle层的逻辑都在PuzzleMachine里，可以分离出去

配置

PuzzleConfig，PuzzleReelSpinConfig

这两个配置可供美术和策划调整，比如特效会调整卷轴的旋转节奏

PuzzleMachine：

1 状态机MachineState：

Idle，Spinning，SmallGameFront，PostSpin，SmallGameBehind，PreSpin

（1） Idle：玩家可以点Spin开始卷轴旋转

（2） Spinning：卷轴正在旋转

（3） SmallGameFront，SmallGameBehind：小游戏前端和后端，都是用来展示小游戏逻辑的，只是时机不同，一个在PostSpin前，一个在后。目前只有万圣节翻箱子位于SmallGameBehind，其余小游戏都在SmallGameFront。

这个变量和Core层的SmallGameMomentType对应，当MomentType里的Front和Behind对应了这里的SmallGameFront和SmallGameBehind

（4） PostSpin：win后数字往上涨的阶段，同时在播放BigWin，EpicWin

（5） PreSpin：从 schedule spin到开始spin之间的时间段，在这个状态下玩家不能做任何操作

当\_state == MachineState.SmallGame时，进入下一层的小游戏状态机PuzzleSmallGameHandlers

2 PuzzleMachine包含了很多小游戏的逻辑，其实这些逻辑可以拆分成更多的文件，就像Core层一样，这是一个可以重构的点

PuzzleReel：

1 使用DoTween播放Symbol移动动画

2 SpinOneUnit：移动 Symbol一格，动画运行时的回调函数不直接执行，而是放到SimpleMessageQueue里，然后在Update函数里去处理这个Queue。否则如果直接在Dotween回调里写具体逻辑，然后发生崩溃，会导致看不到正确的堆栈信息

3 重要并且微妙的\_areaSymbolList机制

PuzzleSymbol：

Symbol旋转时模糊的机制和shader实现

注意：

1 为了保证运行时的性能，symbol的模糊图片是在一开始进机台时就用模糊shader 渲染好了，用的SpinBlur.shader，运行时只是用普通的shader去渲染前面生成的模糊图片

2 symbol上的收集物也需要有模糊效果，但方式不同于symbol，是直接在运行时实时渲染模糊效果的，用的SpinBlurRealTime.shader。因为这部分消耗并不高所以实时渲染没问题。但如果想改成和symbol一致，也是可以的

PuzzleEffect：

整个机台的整体特效，如果是每个卷轴的各自特效，则在PuzzleReel里

GameData：

存储了玩家数据，比如当前credit。这个文件不位于Puzzle目录，但其实可以看做Puzzle层的东西

Machine Asset机台资源

1 策划，美术，程序都需要区分下载机台和非下载机台

2 编辑器模式下，编辑器有个按钮Tools/Load Downloaded MachineAssets。

若勾选，对于下载机台，会走设备上一样的流程，先下载AssetBundle资源，再从中加载资源进机台

若不勾选，则直接加载本地资源

3 具体逻辑和流程参考后面的热更章节及svn上的文档Documents/LiveUpdate

新做机台的流程

1 编辑器下有个按钮Toos/Gen Preset Assets，是一个用于新做机台的工具，根据配置生成一堆机台运行起来前需要的文件，比如大厅机台，一组特效，背景prefab等，代码位于Editor/PresetAssetsGenerator/

2 理论上，如果新机台不包含任何新的逻辑，则完全不需要程序的参与。只需要：

（1）策划先做好配置

（2）UI做好机台内的symbol图

（3）程序运行下Gen Preset Assets工具

机台应该就能跑起来。如果发现要改一些代码，则考虑下是不是可以做成配置。理想目标是没有新逻辑的机台就不需要程序的任何工作

3 美术经常会漏掉设置资源的AssetBundle名称，程序做之前需要检查下。以后可以考虑写一个脚本，自动扫描某些特定目录，把下面的文件都设置一遍AssetBundle名称

4 机台制作规范参考svn文档：Documents/Design/ Slots项目机台制作流程规范.docx

机台玩法的Excel数值加密

1 目的是为了防止破解

2 Excels目录下Core和Game下的所有Excel导出的.asset文件都不会直接打到包里，而是先分别打成AssetBundle，然后对二进制做对称加密，打到包里。运行期会先加载文件，解密，再生成AssetBundle

3 一个重要的配置文件：Scripts/Common/AssetConfig，两个标记为分别指定了是否要把asset文件打成AssetBundle，及是否对打好的AssetBundle做加密。这两个标记会同时影响编辑器的打包代码和运行期的加载解析代码。所以只要改这一个地方即可开关这个功能。目前是都打开的

4 代码分两部分：

编辑器：Editor/BuildAssetBundles/BuildExcelAssetBundles，打包入口类PerformBuild会调用这里

运行期：Scripts/Common/AssetManager

Excel分表功能

Android下读取Excels/目录下的asset

iOS下优先读取ExcelsIOS/目录下的asset，若没有则读取Excels/下的asset

# Common模块

GameManager管理

StartLoading初始化流程：

（1）先热更

（2）热更结束后再调用GameManager初始化各种功能管理器

注意：热更必须在各种Manager以及UserData的初始化之前，这样就能保证党任何模块初始化的时候，用到的数据一定是经过热更后的最新数据

AudioManager：声音播放管理

UIManager：管理各种UI加载

PlatformManager：获取平台和渠道信息

ResolutionAdapt：分辨率适配，主要适配iPhone X和iPad

1 CameraAdaptor：调整摄像机的viewport参数

2 CanvasScalerAdaptor：调整Canvas Scaler的宽高适配比例

3 WideResolutionHelper：适配iPhone X的宽屏时，计算黑边宽度

VersionHelper：管理版本升级时的逻辑

ServerConfig：所有服务端的URL配置

UserGroup：用户分群

ScenesController：场景管理

# UserData和服务端的同步

1 账号分Device和Facebook两种，两种账号都可以同步进度

2 用户登陆和从服务器下载UserData的流程，逻辑入口是

StartLoading.LogInWithTourists -> UserDataHelper.UserSocialLogin

大致流程是：

（1）传Device ID或者Facebook ID给服务端注册获取UDID

（2）服务端返回是否是第一次注册，也就是是否为新玩家

（a）若是，则把本地UserData上传一份给服务端，流程结束

（b）若不是，则从服务端请求下载UserData

（3）下载UserData后，若是安装游戏后第一次玩，则直接使用下载的UserData作为本地进度。若不是，则弹出进度选择框，让玩家选择使用服务端进度或者本地进度

注意：

（1）除了注册UDID外，客户端发给服务端的所有请求都必须带UDID字段，否则被服务端视为无效请求

（2）服务端有两张数据库表，分别记录

（a）Device ID或Facebook ID到UDID的映射

（b）UDID到UserData数据的映射

所以服务端存UserData是用UDID作为Key的

3 上传UserData到服务端，由发送SaveUserDataToServerEvent事件触发，目前就两个地方：

（1）进入大厅场景时

（2）内购成功并获取商品后

4 Device ID在玩家卸载重装游戏后保持不变，具有设备唯一性。Android和iOS的实现机制不同：

（1）Android直接调用Unity接口获取ID

（2）iOS用到了Objective C的原生代码，和iOS的Keychain机制，iOS原生代码是Assets/Plugins/iOS/SvUDIDTools.m

5 UserData同步的逻辑本身非常重要，但这部分代码一开始就写得很乱，而且以前出现过几次用户丢进度的事故，所以处理这部分逻辑要非常小心

# 资源加载管理

AssetManager，CoreAssetManager

1 加载Excel配置数据：AssetManager.LoadExcelAsset

2 加载机台资源：AssetManager.LoadMachineAsset

3 其他普通资源：AssetManager.LoadAsset

注意：上层加载资源不要直接调用Resources.Load，而是通过这些接口去调用，这样才能保证该资源支持热更，AB测试等

加载Excel导出的配置资源的流程：

1 尝试从热更包里加载

2 若热更里没有，则从前面编好的AssetBundle里加载，该AssetBundle是加密的

3 读取时Android直接读Excels/目录下的asset，iOS优先度ExcelsIOS/目录下的asset

4 若都加载不到，直接调用Resources.Load加载

5 Core层代码在Dll模式下，直接读Excel原文件而不是导出的asset文件

# 热更

热更分3种：

1 普通热更

2 AB测试

3 机台资源下载

这3个功能的共同点：

1 都要打AssetBundle，所以都基于一些公用的底层模块逻辑，比如BuildAssetBundles和BuildAssetBundleHelper

2 都有相应的Editor工具代码和运行期代码

Editor代码：Assets/Editor/BuildAssetBundles/

运行期代码：Assets/Scripts/Common/LiveUpdate/

3 都集成到了Jenkins工程里，打包和上传服务器尽量的全自动化  
热更模块有很具体的程序文档和ppt，在svn上：Documents/LiveUpdate/

# 商店内购

1 StoreManager实现了IStoreListener接口，负责初始化内购系统，监听购买成功和失败的系统回调

2 StoreManager里，渠道购买成功后，调用ProcessVerifyPurchase开始验证流程

3 StoreNetWorkServer.VerifyPurchaseCoroutine里，若验证时访问网络失败，就判定是不是联网状态（连着wifi即可），如果是就判定验证成功。这么做是为了保证在我们服务器出现意外情况，比如负载过大，网络阻塞时，不会因此导致玩家拿不到商品

4 StoreNetWorkServer.IsRevalidating代表是否在进行重验证。重验证发生在正常验证流程走了一半，进程被杀重新进游戏的时候。这是个很重要的case，需要处理好

5 最终获得商品加金币的逻辑在SaveBuyItem.SaveBuyItemToData里

6 StoreController负责商品购买的显示层逻辑，比如成功后的弹窗。重点函数是BuySuccess

注意：

1 由于商城代码质量比较差，各种商城的代码量比较大，而且这是个很重要的功能，所以改动时务必小心，多测试。项目之前出现过很严重的丢单bug，需要杜绝

2 有些结构不合理的代码在加新功能前可以先重构，不要在烂代码上添加更多烂代码来解决问题，之前在写商城逻辑时出现过这种情况

# 第三方SDK

Facebook

无

Sensor

AnalysisManager：

1 事件打点用于给策划分析线上数据，是很重要的。写完每个打点逻辑后，建议用设备测试下，然后去神策后台查看下数据是否发送成功

2 注意事件字段的类型，字符串数据应该发送到strXXX字段，整型数据应该发送到integerXXX字段。一个字段的类型是由服务端第一次接受到的该字段数据的类型决定的，一旦定下来后，后面就不能更改，否则神策SDK就报错。比如上个版本发送的是字符串类型的数据到abc字段，如果下个版本改为发整型数据到abc字段，神策SDK就报错，说拒绝类型的变更。写代码时尤其要注意这点

Fabric

1 每个版本上线后，需要检查Fabric上是否有严重的崩溃。在版本上线前的QA阶段，也建议周期性检查下Fabric，出现问题及时修复

2 FabricManager里把玩家的device ID和UDID都上传给了Fabric后台。所以我们能在Fabric上网页上通过搜索device ID或者UDID查到该玩家的所有崩溃信息

3 崩溃分为non fatal和crash。Crash直接闪退，高优先级处理。non fatal可能会导致卡死或者其他奇怪的问题，需要结合代码评估问题的严重性

如何分析线上玩家的bug

1 策划会搜集玩家反馈邮件，然后在神策上查询玩家的device ID拉取事件列表，做初步分析定位bug发生的时间点，然后和程序沟通

2 程序分析bug发生时的事件序列，由于很多bug是崩溃导致的，所以在Fabric后台搜索玩家的device ID或UDID，及对比Fabric记录的崩溃时间和事件序列的时间

Adjust（替代了之前使用过的AppsFlyer）

无

广告相关，Admob，AppLovin，Vungle

广告有一个线程问题，有时候广告SDK的回调函数不是在主线程里，而Unity的大部分接口只能在主线程里调用，所以在广告回调函数里，就只能简单记录事件，push到一个事件队列里，主线程Update函数里从队列里拉取事件，然后做具体的处理逻辑。注意线程同步加锁

# 打包流程

PerformBuild：

本地打包和Jenkins打包入口函数不一样，比如Jenkins打包函数有CommandLineBuildAndroid，包含一些逻辑：

1 解析命令行参数，然后做一些操作，比如设置DEBUG，RELEASE宏等

2 设置VersionCode

注意：VersionCode会在每次Jenkins打包时自增1，这对Release版非常重要，不管是App Store还是GP，Release版每次提交的包的VersionCode都必须大于之前包的VersionCode。所以用于提交的包始终得用某个特定的Jenkins工程来打包，这样就能保证这点，比如iOS上始终用Trojan\_iOS\_Release打提交包

3 在构建前和构建后执行一些操作，比如：

（1）构建前把StreamingAssets下的.svn隐藏文件夹移出去

（2）前面提过的编译Excel AssetBundle，以及打包前会把Excel导出的原.asset文件都移出去，防止打到包里

iOS打包流程

XcodePostProcess

OnPostProcessBuild函数处理当xcode工程生成后，需要对该工程做的修改，需要注意的逻辑：

1 这个类非常特殊，其他代码里常用的DEBUG，RELEASE宏在这个类里是不起作用的（即使在PerformBuild里有设置这些宏），原因未知。所以我们用的方式是通过文件通信，在PerformBuild里把需要的信息写入到txt文件里，然后这里读取文件解析。目前用到了两个文件：

（1）buildSymbol.txt：存储Debug，Release等

（2）buildChannel.txt：存储渠道信息Normal，IW等

2 CopySplashImages把一组splash图片从Unity工程拷到xcode下。做这个操作的原因是这些图片会直接打包到ipa里，所以它们对包体大小影响很大。于是我们就用TinyPNG工具压缩过的图片替换之前尺寸很大图片。另外，由于我们游戏限于横屏，所以竖屏的splash图片都用纯色的，仅起占位符作用。做这个操作后，ipa尺寸能缩小10M左右

3 CopyIcons也对一组icon做了类似的事情

4 拷贝KeychainAccessGroups.plist到xcode工程，这里包含了两个权限：

（1）Keychain权限，用来实现device ID功能，程序生成的device ID就存储在设备的keychain里

（2）Associated groups权限，用来实现兑换码功能，这个plist里记录了允许跳转的 universal links

5 更改info.plist，注意更改的时候如果是要添加字段，那么要确保不要覆盖掉原先已有的内容

# Jenkins打包

打游戏包：

1 参数化构建里包含了当前版本号，每编译一次，Build Number会加1，Build Number会作为versionCode传入到Unity打包命令参数里。游戏里的Setting界面下方显示的是版本号（1.20.0）+VersionCode（260）

2 每次出版本后，需要提升版本号，改两个地方：

（1）BuildUtility里的两个version变量

（2）每个游戏包Jenkins工程的参数化构建版本号

3 Trojan\_iOS\_Release参数化构建里还包括两个选项：

（1）isExportAppStore：是否导出AppStore的包，该包仅用于提交，QA测试期间不勾选，打提交包时勾选

（2）isUploadTestFlight：是否通过脚本把包上传至TestFlight。脚本写好了，但上传不上去，所以至今未使用过，不要勾选。上传包用Xcode/Applications目录下的Application Loader手动上传

4 打好的包会连同editor.log上传公司FTP，log很有用，一旦打包出问题，一般都可以通过查log查到问题

5 有一些Jenkins的Tag工程比如Trojan\_iOS\_Tag，它们仅用来打Tag的debug包。Release提交包仍用Trojan\_iOS\_Release打包，才能保证VersionCode的连续性

热更相关AssetBundle打包：

1 LiveUpdate和ABTest工程打包都打出来zip文件上传至FTP，需要策划或者程序手动上传zip至GM网页

2 MachineAssets工程打出的包会通过js脚本自动上传Amazon S3，脚本位于目录Support/UploaderTool/

3 自动上传如果失败，就再次开始打包流程，勾选onlyUpload选项，勾选这个后会跳过打包流程，直接开始上传。如果打包机一直上传失败，就把FTP上打好的文件放到本地目录，然后本地电脑连4G，在编辑器的机台资源界面里上传

# 音频资源

几个best practice：

1 最重要的原则是在保证听觉效果的前提下，尽量压缩音频资源的尺寸

2 较长的背景音乐建议用单声道，可以省不少空间，方法是在Inspector里勾选Force to Mono（不用改源文件）。有些音效也没有立体声效果，也可以用单声道

3 一般的音乐音效用正常采样率，48K，44.1K。有些没有丰富频段信息的音效可以采样率低一点，比如爆破声也许可以用32K。人声的频谱很窄，可以用低采样率，比如16K甚至8K

4 音效师做出的音频文件里，有时候会有多余的空白波段，用音频编辑软件检查一下，剪切掉无用的片段

5 声音播放代码里会做一个逻辑：不同时播放同一个音效多次。否则，如果同时播放一个音效两次，就会导致波形叠加音量太大。不仅不能同时，甚至间隔很短时间再播放同一个文件也会被拒绝

# 美术资源

1 注意设置每个图片的压缩格式，具体参考svn文档：

Documents/Design/美术/ Slots项目图片压缩事项（第二版）.doc

2 目前很多UI没有统一的Tag管理。原则上一个UI的很多图片可以设置为一个Tag，这样可以减少draw call。这是以后可以优化的一个地方