**工作报告**

目录

[主要完成的三件事 1](#_Toc71882497)

[遇到的问题以及解决方式 3](#_Toc71882498)

[目前最大的困惑 4](#_Toc71882499)

[短期规划 5](#_Toc71882500)

[当离开莉莉丝时，希望自己不同有何不同？ 5](#_Toc71882501)

[针对目前项目最希望改变什么？如何改变？ 5](#_Toc71882502)

[过去公式值得借鉴的地方 5](#_Toc71882503)

[财务自由想做什么？ 5](#_Toc71882504)

[附件 5](#_Toc71882505)

# ****主要完成的三件事****

简要介绍一下去年11月入职以来，近半年时间自己所作的主要工作。

1. 非强制引导，一个可中断的引导系统，目前在游戏中主要用于任务系统，负责玩家点击前往任务后，引导玩家去完成任务。

设计思路，整个引导过程可以看作是一系列的**原子行为(AtomAction)**的组合(后面的Action统一指代AtomAction)。

如升级建筑的任务的行为可分解为：进入主城(切换游戏状态机)-> 选中建筑->移动相机(使得建筑处于视野中心)->显示遮罩(代理屏幕点击操作)->绑定特效(引导玩家进行的操作)->等待点击；这样几个Action组成，把这样一组Action的组合，定义为一个**Sequence**。开始一个Sequence后，过程中出现任何非策划希望的行为，都会立即结束Sequence。比如希望玩家点击“出征”按钮，而玩家点击了其他的地方，就会中断Sequence，从表现上来看就是引导的特效消失了。

显然组成Sequence的Action需要严格的按照时间线排列。同时Action也分为同步和异步两种，如前面“升级建筑”任务中的“移动相机”就是一个**异步Action**，因为需要一个镜头移动的过程，而“绑定特效”就是一个**同步Action**，因为我们需要同步加载一个特效资源，当然如果是异步加载它就变成了一个异步Action，这儿是基于实际的使用场景将其设置成了同步Action。除此之外，Action之间也有加入一段延时的需求，如打开有动画的UI后，在绑定特效的时候，由于实现上特效是挂在遮罩上面并对其需要引导的UI节点，所以针对有动画的UI需要等动画播放完，或者具有复杂布局的UI，需要等待UI relayout完成，我们才能拿到UI里面节点正确的位置，然后绑定特效。

基于以上分析，这个引导系统需要按顺序组合Action，以及有涉及延时的相关需求。自然而然，unity里面的**协程(coroutine)**完全满足该需求。所以把Action和coroutine对应上，而sequence就是一个嵌套的协程，其中包含一系列action子协程，在sequence协程内部，也能够自由的进行延时等操作。需要中断引导的时候，直接中断协程就行了。

基本的架构有了，接下来就是以比较简单的方式同步给策划，尽量减少策划的理解成本，保证一定自由度的同时简化策划后续配表的工作量。经过和策划协商，定义了一个新的id，这个id对应一个sequence的实现，即有一系列相似行为的任务(目前是任务)可以共用该id。在保证一定可配置的灵活性上，开放了一定的参数给策划配置，但是需要备注这些参数在sequence中所起的作用，如前面的“升级建筑”的sequence，策划就可以配置建筑的id。除此之外，开始sequence之前，还需要有一定的前置条件判断，这也在一定程度上开放给策划配置，当然一些特例互相约定，程序直接实现，比如升级英雄，可能的前置条件就是需要玩家至少有一个英雄。

基本思路理清之后，就是具体的程序设计和实现。简述一下在设计中的主要原则和思想。

第一点，Action设计原则是必须尽可能抽象和通用，如果是一个Sequence中特用的同步逻辑，如前面的升级建筑任务中的“选中建筑”这样一个同步行为，就直接硬编码到Sequence本身，不会设计为Action。比如我们有移动相机Action，有发送一个网络请求并结果返回的Action，有打开指定UI的Action等等。所有的Action都继承GameAtomAction，GameAtomAction实现了**IDisposable**接口并继承了**CustomYieldInstruction——unity用于实现用户自定义协程的一个基类**。使用IDisposable是为之后Action中包含非托管资源做预留，同时方便在Sequence中使用using语法糖，以自动Dispose Action。

第二点，Sequence尽可能的复用，在不同的Sequence中Action的组合出现较高频率的重复的时候，将这些Action的组合提取出来组合成一个SubSequence，减少代码重复，保证复用。如我们在打开遮罩的Action后，经常需要跟随绑定特效Action，以及一个等待点击的Action，所以会将这几个Action组合成一个SubSequence。由于协程(迭代器)能够很好的嵌套，所以修改十分方便。

第三点，在其他系统的交互上，对原有逻辑的侵入要尽可能小。首先如果是数据访问和交互，会把这些接口封装在此系统的Utility类中，同时游戏中有数据中心的思想，所以拿取数据是较为方便的。更为耦合的是在绑定引导特效位置的时候，需要拿取UI节点，就需要和UI进行事件通信，系统内部会发送一个事件并指定UI和节点名称，所以需要接收事件的UI进行相应的事件监听。所以定义了一个接口，要进行交互的UI都应该实现该接口。

第四点，不需要该系统的使用者知道系统内部的实现细节，所以只会封装和暴露必要的接口给外部。如传入SequenceId以开始一个引导，以及中止一个引导。同时有的系统可能不需要完整的跑完整个Sequence，只是希望通过系统引导玩家到达某种状态，如新手引导过程中希望通过该系统引导玩家解锁迷雾等，所以会在Sequence的生命周期内的某些节点上发送一些事件，告诉外部系统Sequence运行到哪个地方了，比如加载遮罩显示特效前会发一个ACTION\_AFTER\_MASK\_SHOW事件，外部系统可以接收该事件进行自定义操作，如中断Sequence。

以上，就是非强制引导系统的设计过程，过程中遇到的一些问题会在后面小节进行介绍。当然如果要把一些引导改成强制引导，只需要修改一个字段，对非期望的玩家操作不进行响应，而不是中止协程。

1. 集结，为加强联盟内玩家的交流，添加集结功能。使得联盟内玩家可以发起集结，同盟的其他玩家可以派遣部队去参与集结，集结主拥有操作集结部队的权力，进行一些大地图交互行为，如攻城等。同时为了丰富集结的玩法，添加只有集结部队能够攻打的资源点“宝库”。

具体实现上主要分为两部分，集结部队配置和集结部队表现。

集结部队配置，是一系列界面相关的操作，发起、浏览、加入、删除集结等，以及对集结部队进行上下阵配置，因为在进行交互的时候，最多只能有5只部队真正参与战斗，所以要对加入集结的部队进行配置。同时由于集结引入了其他玩家的英雄和士兵数据，而这些是和玩家自己的数据不同的，甚至是玩家没有的。但是玩家在集结部队面板里面是能看到这些英雄的，所以需要为集结部队维护其他玩家相关英雄和士兵的信息。

集结部队表现，主要分为在界面上的表现，以及在大地图上的表现。界面上表现和普通部队基本一致，如部队状态战力等都需要显示。所以针对这些显示需求**抽离出统一的接口**，普通部队和集结部队都需要实现该接口，最后再把显示的地方进行参数替换。当然集结部队的具体实现是按照需求而定，如集结部队的状态是以上阵部队的部队状态而定，并且按照优先级排列，如优先显示战平。然后是大地图表现，增加了一种新的移动物体(move object)类型，并给集结部队做一些显示上的表现，如特殊的行军线类型添加，特殊的行军部队排列，从之前只支持单个英雄扩展成支持排列多个英雄。

而宝库，是一种动态刷新的资源点，所以在大地图ECS架构中添加一种新的Entity，以及对应的GameObject资源缓存。

1. 试炼，为了增加玩家的日活，设计的一个每日PVE爬塔的功能。

具体实现上分为三个部分，试炼邮件，试炼场景(UI)，以及试炼部队编辑。

试炼邮件，试炼邮件的数据是单独的协议和也是每天都会清空的，所以邮件的数据会单独的维护，并且走增量逻辑。除此之外也需要对集结的邮件实现对应的红点树逻辑。

试炼场景，这是个普通的超过一屏的UI，在背景上会动态的加载策划配置的关卡，同时在关卡之间会动态的生成连线——直接对一张Image进行旋转和缩放。此外，场景界面有一定的收视需求，如左右拖动，所以需要对不同的屏幕分辨率、Canvas缩放比例进行拖动范围适配。

总的来说，试用期工作完成情况良好，按时保质的完成了相关任务。可中断引导系统主要体现了软件工程的一些思想，用协程比较巧妙的实现了一定自由度的引导系统，过程中也踩了一些协程的坑点，并解决了相关问题。集结，主要体现在已有功能基础上进行代码扩展和重构的思想和能力。试炼体现对UGUI具有一定的掌握。

# ****遇到的问题以及解决方式****

总结一下自己在工作中遇到的一些问题，以及自己是如何解决这些问题的。

1. 问题：UI在Overlay下由于z值过大导致被裁剪的问题，该问题发生在做主城建筑气泡需求的时候，气泡是个小型的状态机，当达到某些状态的时候就会显示出来。而在策划测试验收的时候发现游戏时候，气泡没有显示却能够点击，但是无法稳定复现。

解决过程：让策划复现之后，保留现场找我。当我在策划电脑上查看时，首先查看GameObject的Active情况发现是Active的，同时查看Image之类的组件也是Enable

的，用Frame Debug看，是画了该组件的，也就是是给GPU发送了绘制相关图形API的，但是却看不到。这样那么情况只会是GPU在画的时候裁剪掉了，那么很明显了，虽然Overlay的UI是Unity引擎底层保证最后绘制，但是也是需要传入一个正交投影矩阵的，显然该矩阵具有一个远平面的最大深度。好，那我们再测试一下，发现Editor下其实始终能看到UI，但是Runtime时候在z值大于1000后，就会被裁剪，那么可以得出结论，Runtime的时候Overlay投影矩阵的远平面是1000。注：Unity版本2018.4.1.f1。

1. 问题：协程异常捕获和嵌套协程中止问题，这个问题是发生在可中断引导系统的开发过程中。首先前面提到整个引导的过程是一个嵌套的协程，而在这个引导过程中抛出任何非期望的异常(如因为其他模块的bug抛出一些空指针异常)，都希望能够把协程直接终止掉。所以需要在运行协程的过程中，捕获其中抛出的异常。同样，由于有终止协程的需求，可能由外部系统直接终止，也可能是发生异常导致。但是Unity是无法捕获协程抛出的异常的，同样终止嵌套的协程也需要处理一下。

解决过程：针对协程无法捕获异常的问题，本质上协程就是一个迭代器，我们将迭代器再封装一层，将MoveNext用try catch包含起来，并在捕获到异常的时候打印异常并触发异常回调。对于嵌套协程停不下来问题，我们需要对嵌套协程中的每一个协程都StopCoroutine，所以是一个递归的过程。

1. 问题：集中寻路是卡顿问题。在引导系统中，有找玩家周围一定范围内距离自己的最近的资源地，并且这个最近是真正意义上的寻路距离最近，所以需要对该范围内的资源地进行全部寻路。这儿寻路的量级是上百次，会有一定的开销，所以会在玩家第一次进入游戏的时候进行寻路，并本地缓存下来。但是还有一种情况，玩家可能改变位置(迁城)，这个时候就必须重新寻路，PC上耗时，12.9ms。但是到真机上卡顿还是比较明显的。

解决过程：和Leader商量后，确定对寻路做个分帧，每帧寻路一个点。寻路每帧耗时降到了0.04ms。同时，也做了最保守的估计，如果玩家在寻路数据还未完整更新完的情况下，就要使用该数据时，会同步的把还未寻路完成的点，全部寻路完成(虽然实际游戏过程中基本不会遇到该情况)。

# ****目前最大的困惑****

由于项目中大地图架构是用的基于Entitias的ECS，而很多数据还是绑定在Prefab上的，如Mesh，所以很多数据是依赖于加载Prefab来完成的。我们知道ECS有很多优点，如缓存优化，多线程友好。但是在Unity下绕不过去的一个优点是用ECS我们能够很好的去GameObject化，因为Unity Instantiate的效率实在太低了，其中包含Produce GameObject同时还伴随各种组件脚本的回调。用Unity DOTS测试了一些实例化25000个GameObject耗时1ms，而Instantiate耗时400ms。我想可以用资源配置文件代替Prefab，但是这样无法保证美术的所见即所？另一种方式是像DOTS那样运行时将Prefabs转换为一个对应的Entity，转化后再创建就没有Instantiate开销了，这样美术就可以在Editor像常规那样操作即可，但是运行时就不好修改了。所以想知道，目前有没有什么游戏是比较完整的去GameObject化的，以及他们是怎么做的？

# ****短期规划****

简短的描述一些自己一年内的短期规划。

1. 好记性不如烂笔头。养成做笔记，写东西的习惯。将自己读书学到的东西，自己的思考记录下来。将自己阅读源码，阅读Paper、Slide的相关总结思考记录下来。之前很多时候由于自己懒，读完东西有一定的感受，但是没有记录，导致后面抓不到当时的点了。所以希望能较好的养成写东西的习惯。
2. 基于自己的职业规划是向图形程序方向发展。所以会继续这方面的学习和积累。
3. 要开发一个渲染引擎，所以需要详细设计和规划自己的开发路线，希望年末时能有一定的成果。

# ****当离开莉莉丝时，希望自己不同有何不同？****

1. 项目上，希望做出受玩家喜爱的作品，同时也希望游戏能打出自己的名声(和别人聊天时好装X)。
2. 技术上，积累更多的经验，不断的提升自己，希望自己变得更有竞争力。

# ****针对目前项目最希望改变什么？如何改变？****

从个人角度提出一些希望改变的地方。

1. Review返工问题。项目功能开发流程大致是，策划案->UI->程序开发->验收->**Review->修改**->提测。一般情况，Review阶段功能大致都确定了，只会进行细节上的调整。但有时候，Review可能会进行相对较多的调整，甚至Review、修改这个过程会循环好几次。当然可以理解有些东西就是要看到效果才能知道好不好，而有些方案可能前期想象一下，多讨论一下，就会发现不合理。当然，不是说个人不想改，从程序的角度来说，很多时候程序的修改是不大的。但是对于一些修改，可能需要美术或UI走一遍，重新设计，出资源，做UI等流程，个人觉得有点浪费人力了。希望大家都能有一个意识，方案再越前期确定，那么修改成本越小。当然，解决方案就是前期定方案的时候，大家都从自己角度多思考，提出自己的看法。针对策划，因为他是撰写案子的人，要有主人翁意识，前期多准备，多思考。
2. 目前项目组中没有专业的UGUI。由于之前该岗位的人离职了，所以目前的UI是美术或原画同学拼的，他们拼的UI能实现对应的功能。但是UI的效率就无法保证了， Relayout、Rebuild、Batch上的考虑。解决方式，可以直接招个对应岗位的人，或者内部提供一些培训的机会。

# ****过去公司值得借鉴的地方****

由于自己是客户端程序，所以从客户端的角度提出可以参考的地方，当然目前项目已经到中后期，很多方案不好用于当前项目，但是如果开新项目之类的可以参考一下。

1. 配置类生成，对于配置来说，根据每个表生成对应的配置类，需要注意一下配置中的复合类型(json对象)，表头会直接对应配置类中的属性。所以访问的时候直接拿到对应的配置类实例，访问属性即可。同时，如果配置中如果有重复度很高的字符串，可以将这些字符串提取出来，放在二进制表文件头或者其他地方，通过Id之类的索引。降低配置文件大小。
2. 单例生命周期，游戏逻辑中避免不了使用的是单例，但是很多单例是实现都是Lazy Load即用到才初始化，所以如果单例有依赖，顺序不好保证。所以在不同的GameState中可以明确的初始化单例。
3. 限制MonoBehaviour 的使用，首先不同的MonoBehaviour 的执行时顺序控制起来太麻烦，如果有前后顺序依赖容易出问题。其次，MonoBehaviour 是从Native C++那边驱动的，所以会有非托管代码到托管代码的Interop开销。推荐的做法是游戏中仅保留必要的Mono脚本，如一个脚本负责把整个游戏拉起来，驱动游戏运行，一个脚本处理协程相关需求。
4. 寻路问题，针对地图较大的游戏，推荐使用网格寻路，用统一大小的格子，OpenList会过大。而地图较小的时候，地形较为开阔的时候，可以使用Jump Point Serach，可以跳过中间的很多点以减小OpenList。

# ****财务自由想做什么？****

这个还是取决于自由的程度，一般程度的话，会做独立游戏，可能是那种艺术性比较强的游戏，如奥日。如果很自由，因为个人很喜欢Disney和Pixar的动画，所以可能会做动画电影之类的。

# ****附件****

包含一些试用期自己做的读代码的分析和记录吧。是用到Entitas，所以阅读了对应的源码；做动效需求的时候，用到DoTween，阅读了DoTween的代码；最后是做UI的时候有些东西有点忘记了，所以重新浏览了一遍UGUI C#源码。

[1] Enttias笔记，

[2] DOTween笔记，

[3] UGUI笔记，