# 业务

## 战斗

先定个基调，什么样的战斗框架是好的战斗框架？

1. 性能好
2. 效果多样性：支持技能效果多样性，但战斗框架稳定不变
3. 策划配置自由

举个例子，《流放之路》可以自由的自合技能，技能样式可以达到“无数种“，这不可能是靠人力完成的，而是更接近技能UGC的效果，虽然游戏类型大概率不会是UGC技能，但却给策划提供了很好的自由度。

在快手时候的时期，我们的战斗框架实现了《流放之路》的技能组合效果，技能样式能达到几十万种，开发时间大概半个月完成，这依赖于底层足够抽象、颗粒度足够细

**战斗框架如果思路对了，其实非常简单，只是业务复杂且多，但并不难，基本上在我们团队呆一年的小朋友就能自己搭建战斗框架了。前期战斗框架后端投入人力1.5人，中后期0.5个人维护就够了，前端会比后端多一个人**

### 战斗框架

战斗框架由以下主要组件组成

1. 技能信息component

等级、cd、连携技信息

1. 技能执行component

技能执行流程，下面重点说

1. 属性

一级属性、二级属性、假属性、真属性、中间属性

1. Buff component
2. 状态component

保存tag，tag包括霸体、眩晕、定身、濒死、魅惑、僵直……除技能相关tag外，还包含其他模块带来的tag；

tag限制行为，行为包括是否能移动、施法、普攻……

1. 索敌component

锁定敌人，可能作为技能释放的目标

1. 选敌component

技能释放时选择目标，可以为当前锁定的敌人、面朝扇形最近的敌人、面朝扇形血量最低的敌人……按策划需求实现

1. 被动技能component

主要以事件触发为主要触发方式，样式多种，加伤害、必暴击、周期执行（通过buff）、改变属性……

1. 仇恨component

根据仇恨列表选出一个目标，强制选出嘲讽目标。玩家和怪的有差别

1. 移动component

除正常移动外，还有击退、step等

1. 冲锋component

多种冲锋样式，也调用移动component，同时自己检查碰撞

1. 子弹component

管理子弹的生命周期，子弹发射的逻辑

1. 子弹Actor

碰撞弹、追踪弹、钩子、弹弹弹

1. 虚拟体Actor

能移动的触发器

1. 区域触发器Actor

触发器，触发技能效果

1. ……其他的就不一一例举了

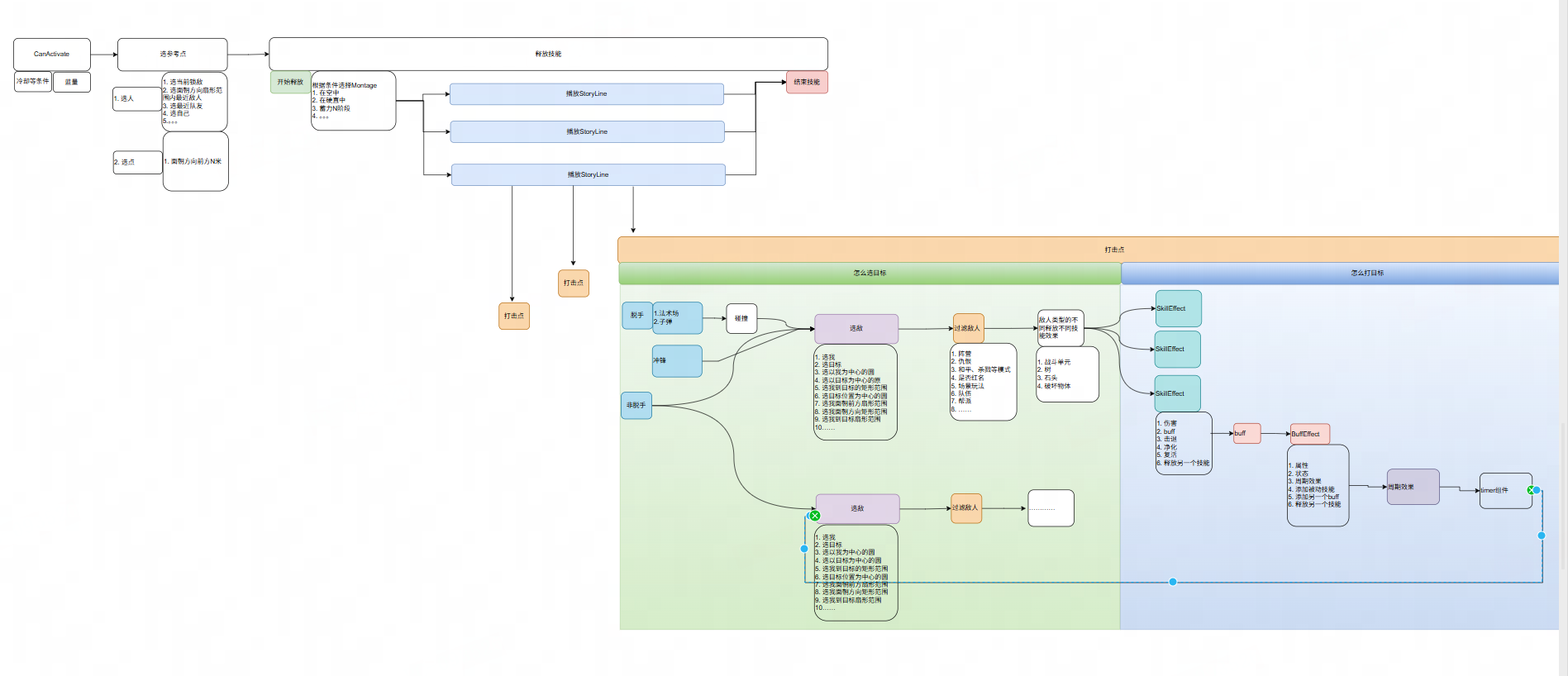
基本由以上component组成，component之前互相关联、互相调用

### 技能释放流程

主要在技能执行component，流程中把以上component全调动起来

一定要看知乎<https://zhuanlan.zhihu.com/p/5933153600>

从本质上看清技能流程



### 吐槽一些

自从离开了完美体系，再也没看到过成熟的战斗框架，不管是在腾讯、网易、库洛。问题的重灾区有以下几点

1. 选目标、选敌分不清楚（上面知乎中提到的），选敌阶段没抽象好，导致整个技能释放流程中的1/5的代码丑陋，导致程序和策划的工作量加大，自由度降低
2. 属性自动化和性能：现在的项目属性计算巨差，提了优化建议，还没优化；某L某C项目（使用的是GAS），优化完内存降到原来的40%。属性的性能在我们原来的项目中看不到性能热点的。**属性component有很多可设计的点，有机会再说**
3. GAS不要用。前期能快速出效果，后期优化难，本身技能释放流程存在不合理，提供模式多 但大多模式用不上，配置链路场，属性集不自动，属性计算不合理，属性内存大，配置颗粒度大……
4. 策划提需求：不懂战斗的策划提需求，懂战斗的策划提**机制**
5. 程序没理解本质（和第4条类似）：比如策划说要实现添加buff后持续性掉血，于是buff里有了dot功能，策划说持有某buff期间 打人可以回血，于是buff里有了监听打人回血功能……后来buff里的功能越积累越多。但其实dot是timer组件做的事，打人回血是被动技能做的事，buff只是桥梁，buff效果是调用各个component。这仅仅是buff的例子
6. Buff堆积太多功能（由5导致），buff仅仅互斥、共存、叠层、声明周期问题，其他的都可以封装在buff效果中去调用其他component
7. ……

**近年的几个项目中做战斗的感受就是，之前不是问题的地方，在这几个项目中全出了问题**

## 角色AI（状态机+行为树）

这个其实没什么好说的，UE本身提供了行为树，我们之前做的就是在行为树的外面套了一层状态机。目的是防止过大的树会导致性能问题、可读性变差，切状态时执行on\_leave\_state旧状态然后on\_enter\_state新状态，清理掉旧状态不用的黑板变量，只聚焦在当前状态中逻辑，lyra中也做了类似的设计，只不过比较单薄，仅仅用tag区分几个行为树

## 场景AI（关卡逻辑）

场景AI为什么不用行为树？

答：角色AI更多关注自己，而场景AI更关注多个角色

行为树中更多的主动tick自己，而场景tick所有玩家的话，会导致性能问题。场景AI更适合被触发，所以很早之前就诞生了**war3触发器**。

在快手期间实现过一套类似的，就像行为树sequence、select、action、conditon节点一样，通过非常简单的规则就实现丰富的逻辑

场景AI的原则就两个：

1. 计时触发
2. 计数触发

计时器及时一个timer，计数器就是外部事件触发变量的数值变化，这些通过逻辑单元组合起来……

详见知乎：<https://zhuanlan.zhihu.com/p/596598633>

## 多层级AOI

按体型实现不同的广播距离，体型大的怪更容易被看到，广播更远，可广播9宫格、25宫格、49宫格……无限层宫格

AOI高度上也有划分

增加体验、提升性能

# 性能

## 广播消息过滤（千人同屏）

不一定非要做千人同屏玩法，这只是性能优化的思路。

1. 定时刷新可见Actor，Actor按各种因素、权重排除优先级
2. 消息分级，可过滤效率、不可过滤消息
3. 多线程分担广播遍历和序列化压力

以上在上次已经聊过了

# 体验增强

以下内容之前也聊过，简单复述一下

## 无缝地图

1. 场景cell管理，动态规划
2. 边缘防抖，场景cell边缘几个格子分别为广播格子、共享格子、跳转格子，共享格子区域可防抖
3. 人群重心预测，防止一次切场景过大导致卡顿

## 服务器物理动画

知乎：<https://zhuanlan.zhihu.com/p/533180304>

这个功能应用范围比较小，但是通过这个功能掌握了动画和物理的技术。理论可实现更多花样和性能提升，例如

1. 动画可不可以只tick某个骨骼。例如服务器验证是否子弹命中了敌人头部，怎么用最小的性能开销验证。传统做法从根骨骼一层一层播放到头，那么可不可以直接播放头？
2. Physix性能怎么调优？什么规模和参数会导致性能问题？
3. 可不可以实现多个场景共用一个物理世界？
4. 利用Simulate和FetchResult之间的间隙做其他的事情，提高线程利用率。我是在看从物理引擎的时候知道的，后来发现UE中封装了这个tick阶段，但很有使用UE的人并不知道，其他引擎的人就更难知道了（至少我这几年所在的几个大公司的周围同事不清楚）
5. Phtsix双buffer机制，和我们DBServie中的机制很像

## recastnavigation寻路

源码解析：<https://zhuanlan.zhihu.com/p/583887024>

源码注释：<https://github.com/wcqdong/recastnavigation_comment>

除了源码优化以外，还实现了很多需求

1. 用navmesh描述水体，实现游泳等水中判定，性能比物理好
2. 多体型寻路。UE的方式是为每一种体型生成一个navmesh，内存大。可以生成多种半径的边缘，用不同的area标识
3. 不同的怪可以走到水的不同深度。自动生成水的navmesh边缘，减少人工，实现新玩法

## 体素

知乎：<https://zhuanlan.zhihu.com/p/455768430>

极致内存优化，比大厂内的某几个内存优化要好，而且相对于物理、navmesh，体素作为最简单的数据结构，有着天然的性能优势（除了内存），可应用于多中场景，不仅用于寻路

1. 音频体素：走路、爬墙、碰撞的声音，代替物理。从Actor和landscape中获取材质数据，播放不同的声音
2. 封闭空间：减少封闭空间外的加载，优化性能（其实这本应该是PVS遮挡剔除该做的功能，用体素实现并不适合，但既然有体素，也可以做为次级代替方案）

物理、navmesh、体素除了完成它们的本职工作外，还起到了感知周围空间环境的作用

在渲染层面上可以通过肉眼看到游戏中的环境，但在代码层面上，只能通过这三个系统看到周围环境。（还有避障）