手游《魂斗罗：归来》同步方案分享

田亚涛

2019.5.16

关于我

来自天美J1 魂斗罗项目组，项目预研阶段开始加入，目前主要负责角色和武器技能方面的工作，除了玩法功能外，还包括联机同步及DS方面的工作。

曾参与过多款MMORPG、RTS、SLG等游戏研发。

分享的内容

- 联机同步方案选型
- 基于C/S架构的联机同步框架
- 弹道同步策略分享
- 命中判定和反外挂思考

产品特点与核心

产品特点：

- 横版射击，弹道一目了然
- 视野差别，命中感知强烈
- 技能多样化，弹道轨迹丰富
- 地形复杂，操作性强，360度各种跳跃躲闪

技术核心：

- 弹道同步（精准射击）
- 命中反馈（精准判断，减少误判）
- 弱网同步（降低对网络抖动的敏感度）
- 反外挂（公平竞技）

魂斗罗所要达成的目标

- 一款高质量的横板联机对战游戏
- 横板视角下对一致性要求更高
- 减少网络抖动带来的体验影响
- 解决命中校验和外挂问题
- 建立专有服务器(Dedicated Server)

联机同步方案选型

帧同步和状态同步

-从名称来看我们能想到什么?

状态同步实战

帧同步：

- 电影胶片
- 序列帧动画
- 游戏画面
- .....

✓指定频率(帧率)，按照顺序、逐步推进，形成一套完整的游戏状态

帧同步技术要点

- 客户端跑完整逻辑（重客户端)
- 服务器按一定频率转发并提供补帧服务(轻服务器)
- 多端同步以上报和转发操作为主（节省流量)
- 客户端逐帧推进逻辑状态(庞大的计算量)
- 客户端需逻辑与表现分离
- 断线重连需重新执行历史操作(时间取决于进度)
- 保证相同的输入+相同的时机=相同的输出(计算一致性)

状态同步实战

帧同步优势

- 逻辑开发效率高
- 高精度打击体验
- 游戏流量消耗低
- 天然支持回放
- 节省服务器资源

劣势

- 存在输入延迟
- 预表现能力有限
- 断线重连恢复时间取决于进度
- 反外挂能力弱
- 难以应对大规模多人玩法

✓**常被RTS、MOBA、动作类游戏所采用**

状态同步实战

状态同步:

- 移动
- 眩晕
- 冰冻
- 死亡
- .....

✓注重形为所产生的结果和某个时刻所处的状态，弱化过程一致性，通过角色所处状态表达游戏阶段。

状态同步技术要点

- 服务器跑完整逻辑
- 服务器按一定频率发送状态给客户端
- 多端同步以状态和RPC为主
- 客户端做预测和回滚机制
- 断线重连和中途加入仅需还原服务器状态
- 反外挂基本以服务器判定为主

状态同步实战

状态同步优势

- 安全性高
- 网络要求宽松
- 断线重连/中途加入方便
- 通过视野裁剪和优化同步量应对多人玩法
- 反外挂能力强

劣势

- 开发复杂度大
- 同步量随对象数增加
- 精准性较弱
- 回放功能弱
- 预表现受网络影响

✓**常被RPG、FPS类游戏所采用**

魂斗罗为什么选择状态同步?

	手感	命中反馈	网络容忍度	外挂风险	多人模式
状态同步	本地先执行，然后同步状态	预表现+状态纠正	对抖动不敏感	权威服务器	通过视野裁剪和优化同步量应对多人玩法
帧同步	一个有效的操作需等服务器响应才执行，存在输入延迟	存在输入延迟	网络要求高	依赖举报或多端结果上报	客户端需复盘所有玩家操作，逻辑压力大

UDP or TCP

TCP：

传输控制协议，面向连接的传输协议。基于字节流的传输层通信协议，属于可靠协议， 特点：

- 面向连接
- 可靠有序
- 自动分包
- 使用简单
- 拥塞控制

状态同步实战

TCP：

看起来很完美，但牺牲的恰恰是我们最不能忍受的：**延迟！！**

UDP：

用户数据包协议，另外一种建立在IP协议之上的协议，传输的数据是‘数据包’的形式，提供面向事务的简单不可靠信息传送服务。特点：

- 没有连接的概念 （通过心跳确定虚拟连接）
- 不可靠
- 不能保序
- 手动分包
- 无流量控制

相对于TCP而言：

优点:

- UDP协议的控制选项较少，在数据传输过程中延迟小、数据传输效率高，适合对可靠性要求不高，或者可以自行保证可靠性的程序。

缺点：

UDP报文没有可靠性保证、顺序保证和流量控制字段等，可靠性较差

状态同步实战

针对UDP中可靠性和顺序等问题：

- 可靠性问题
  - a. RUDP, 基于序号的ACK确认包+丢包重传机制
  - b. 基于冗余包机制
- 乱序问题
  - a. 按序号标识发送顺序
  - b. 队列机制, 等待缺失包+排序.

TCP的可靠和UDP的低延迟，该如何选择?

- 对于实时性和通信频率不高的情况下可以采用TCP，使用简单且完善保障机制。

- 对数据的实时性要求高的情况下（FPS、格斗类），我们希望看到的是“实时”和流畅，所以，这种情况通常更倾向于会使用UDP。

涉及到的关键术语：

- Dedicated Server
- Property Replication
- 事件RPC

Dedicated Server

- 专用服务器，运行完整游戏逻辑，管理联机模式下所有战斗元素和战斗状态，并以固定频率同步状态到客户端。

- 作为权威服务器存在，用于向客户端广播最新数据及玩家行为仲裁。

- **如：命中判定、外挂检测、战斗进度…**

Property Replication

- 服务端将按照一定的频率收集属性改变列表并发送给客户端
- 客户端的属性值会被服务端的属性覆盖
- 由于频率问题可能会丢失过程量

- **如：血量、攻击力、防御、速度、状态...**

事件RPC

- (Remote Procedure Call)即远程过程调用
- 提供一种透明的机制隐藏本地和远程通信交互
- 分为同步调用和异步调用
- RPC允许Client和Server相互调用
- **如：移动事件、使用技能、命中消息、**

基于C/S架构的联机同步框架

专用服务器(Dedicated Server 简称DS)：

- 使用Unity引擎发布Linux版本
- 剔除引擎多余模块(渲染、动画、UI等等...)
- 通过目录和宏隔离前后端代码
- 核心代码逻辑与服务器共用

网络通信模块：

- 使用第三方组件
- 提供了基础的RUDP服务
- 支持Reliable/Unreliable
- 基于RPC + StateSync同步方式

消息同步：

- 事件RPC

--分可靠和不可靠，由客户端和DS之间的各种事件通知，如：移动、使用技能、命中、死亡事件等。

- 属性复制

--从DS到客户端，按一定频率收集状态变化，下发给客户端，如：角色状态、基础属性等。

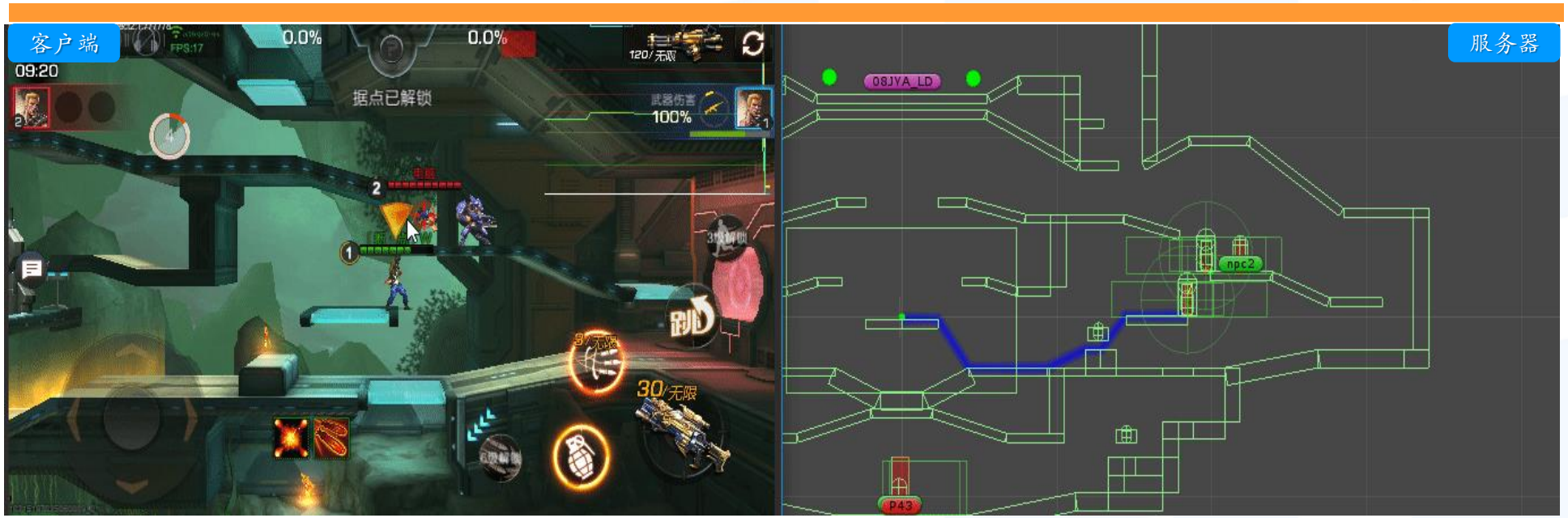
计算一致性：

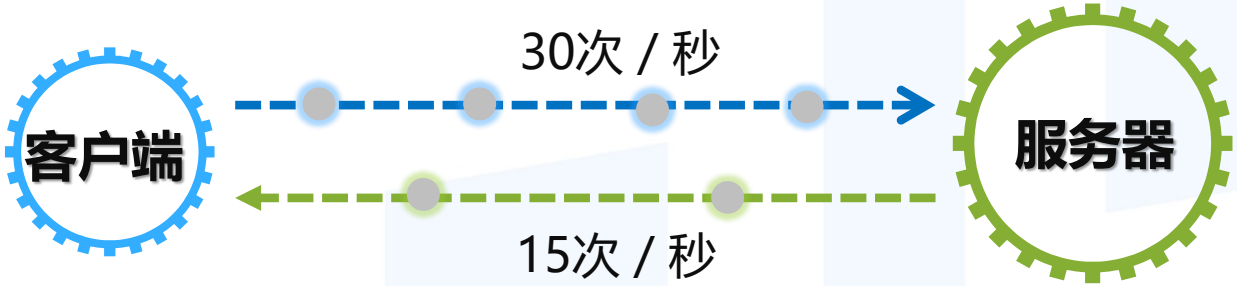
- 执行顺序的一致性
  - --逻辑更新顺序
  - --消息发送顺序
  - --消息处理顺序
  - --记录更新序号
  - --逻辑执行结果

MoveStep:
更新电梯位移
更新刚体位移
更新触地检测
更新位置移动
更新玩家转向
更新技能释放
...
SendMoveToServer

- 问：跳跃->射击 和 射击->跳跃 最终同步结果会相同吗？

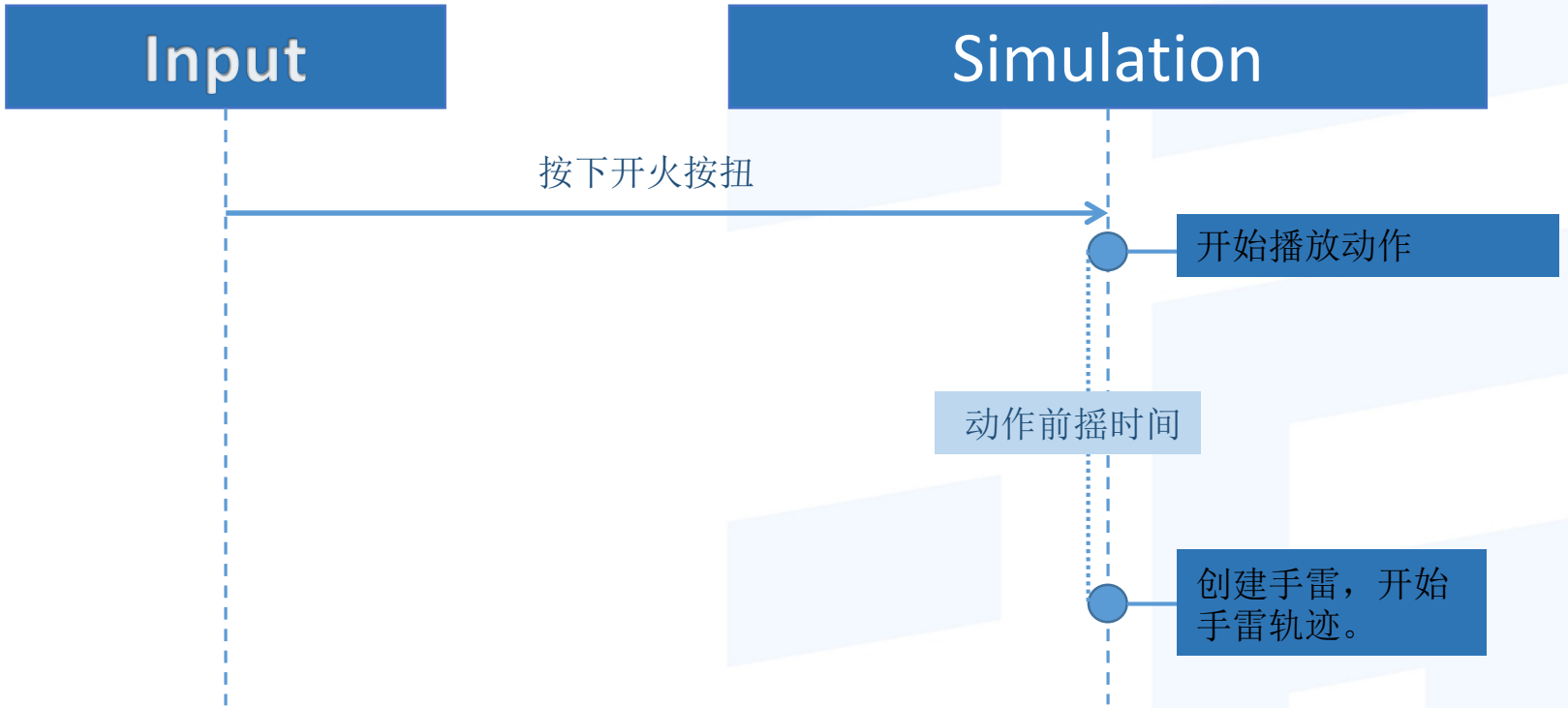
联机模式：





弹道同步策略分享

状态同步实战



展示手雷释放过程

➢ 武器特点：

① 弹道轨迹丰富多样

② 单把武器同时允许多种子弹切换

③ 子弹有生命值，根据命中目标属性衰减

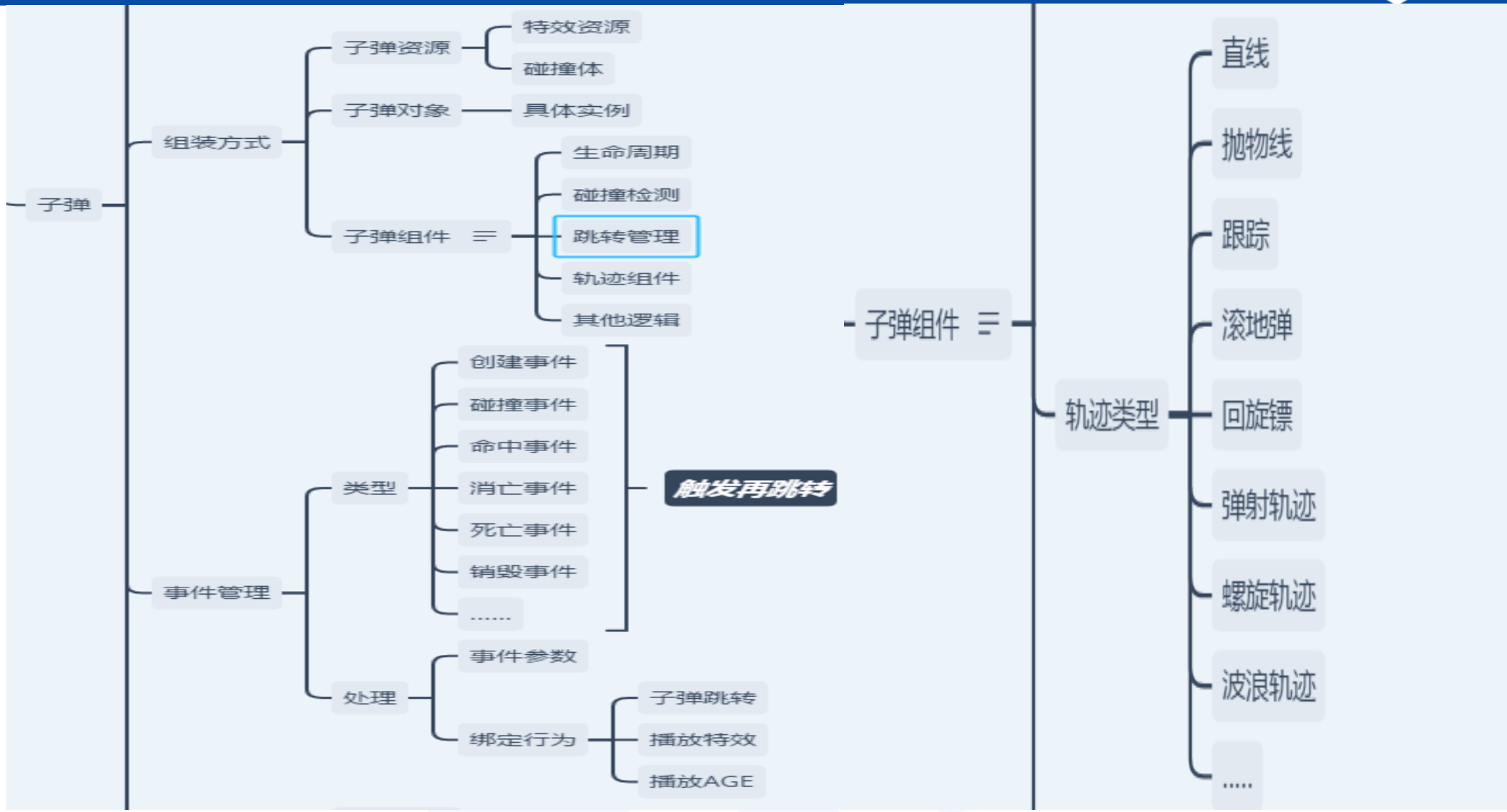
④ 个别子弹允许穿墙伤害

⑤ 发射频率随机，单位时间发射数不固定

⑥ 允许子弹轨迹随状态连环跳转

状态同步实战

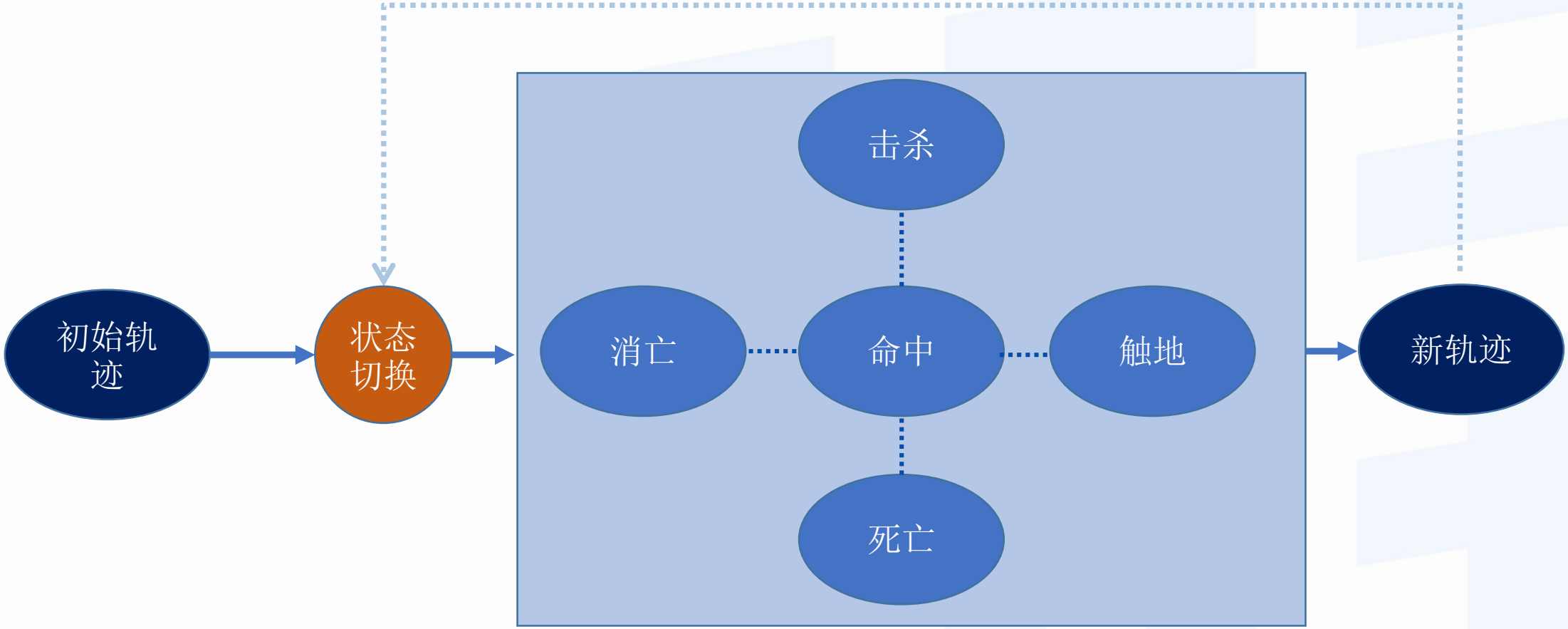
➢ 子弹结构：



通过事件类型，可以再配置触发不同的新子弹（轨迹）

状态同步实战

跳转机制：



- 例：向前方发射一枚直线飞行子弹
- 击杀：变为追踪轨迹，飞向附近目标　命中：比较穿透值,确认是否可穿透目标　消亡：原地生成爆炸框

同步策略拆分：

- 瞬发型
- 投射型
- 组合型

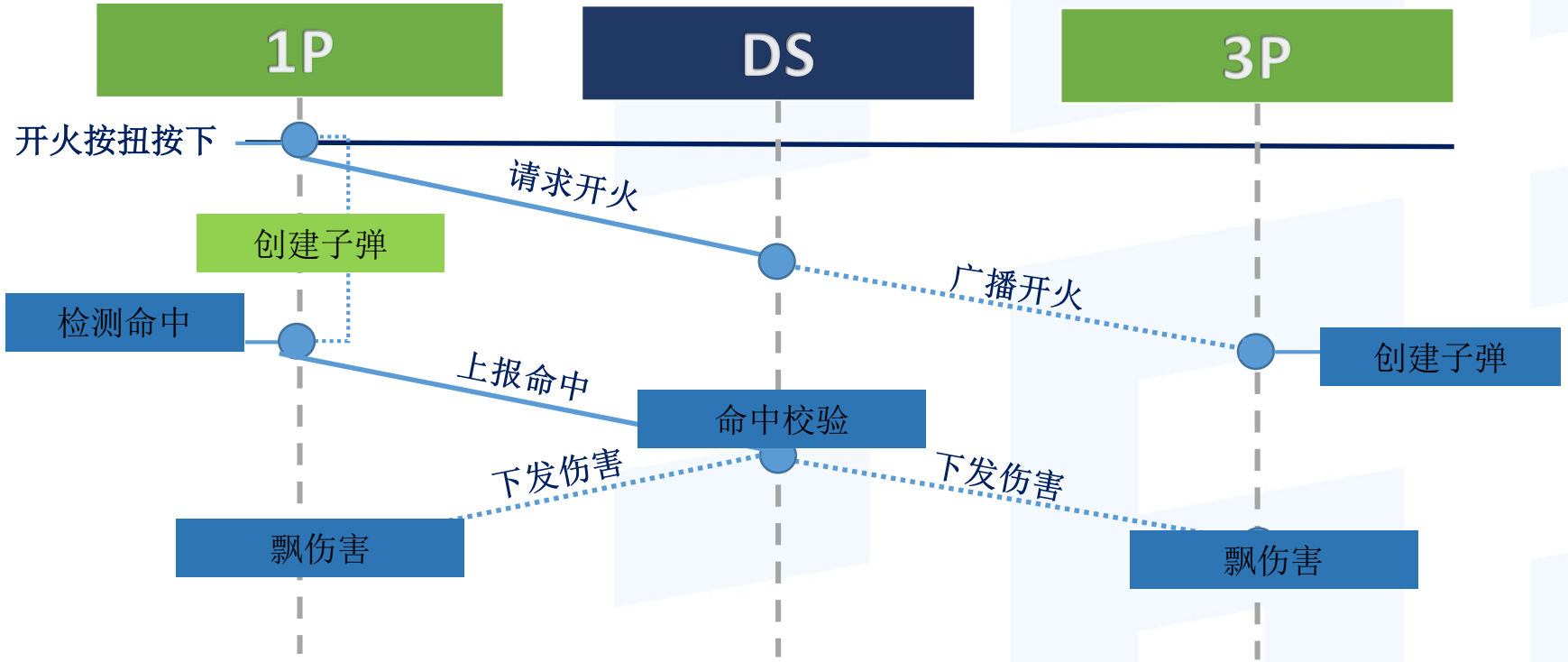
状态同步实战

瞬发型:

- 特点
  - 弹道单一
  - 速度快、频率高
  - 无复杂状态跳转
  - 如： AK、狙击

状态同步实战

同步过程：



优势：本地预表现，响应及时，体验顺畅

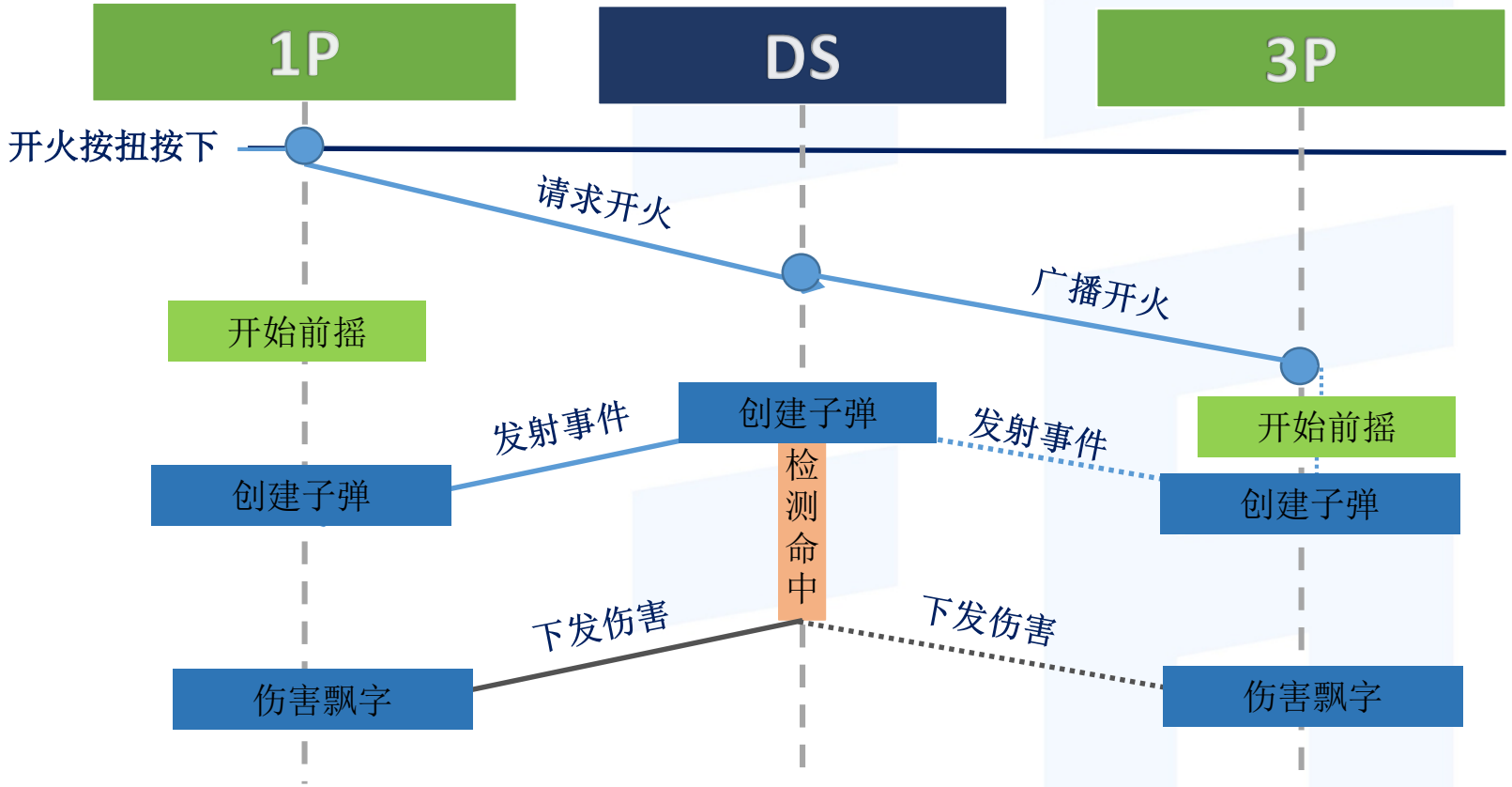
难点：命中校验、反外挂

投射型：

- 特点
  - 弹道复杂
  - 以AOE为主
  - 速度慢、轨迹多变
  - 各种复杂状态跳转
  - 如：RPG、跟踪弹、手雷等

状态同步实战

同步过程：



优势：与3P保持弹道一致，公平性优先

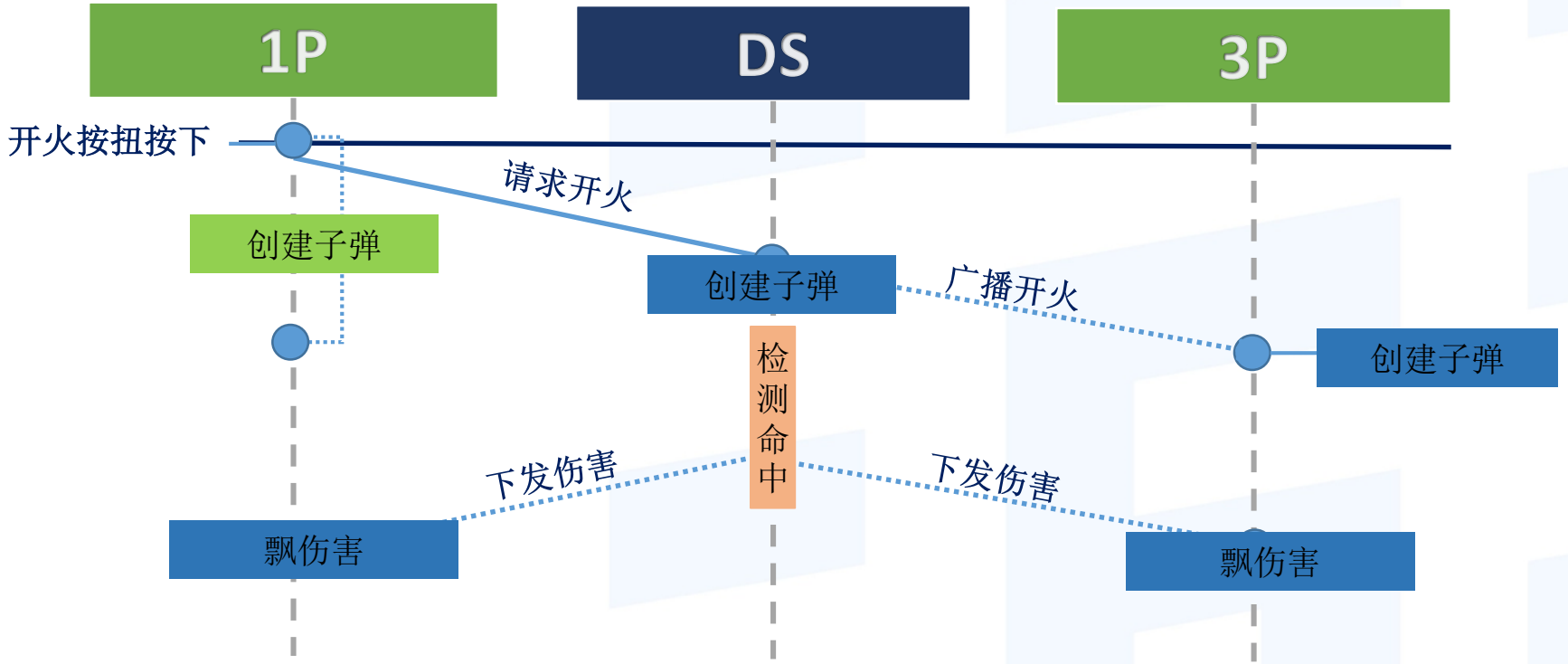
缺点：牺牲局部实时性，弹道创建存在延迟

组合型：

- 特点
  - 弹道复杂
  - 速度较快
  - 各种复杂状态跳转
  - 如：武器：镭射步枪、比尔：爆破飞弹

状态同步实战

同步过程：



优势：结合前两种同步，针对速度快且跳转状态多变时，由DS公平角度检测命中。

一、网络延迟带来的问题

优化方案：

- **延迟补偿 + 平滑插值**
  - -- 获取相对于DS的延迟时间
  - -- 计算具体延迟帧
  - -- 设置最大补偿帧数
  - -- 根据延迟距离进行平滑插值

延迟时间 = 消息传输时间 + (当前时间 – 消息接收时间)

追帧数 = Min（最大补偿帧, 延迟时间 / 帧间隔)

一、网络延迟带来的问题

二、网络抖动带来的问题：

- **消息积压和延迟加大**

DS：

瞬间收到N个操作&命中事件，导致DS明显波动，命中反馈对受击者来讲，无法接受.

Client：

瞬间收到N个子弹发射或命中等事件，客户端帧率急速下降。

负载增加的同时，加剧消息延迟继续扩大.

状态同步实战

网络抖动影响优化：

- **DS：**
  a. 消息环形队列, 加速执行+丢弃机制

  b. 命中事件, 超时丢弃

- **Client：**
  a. 缓冲队列，存活时间检查（超过，直接移除）

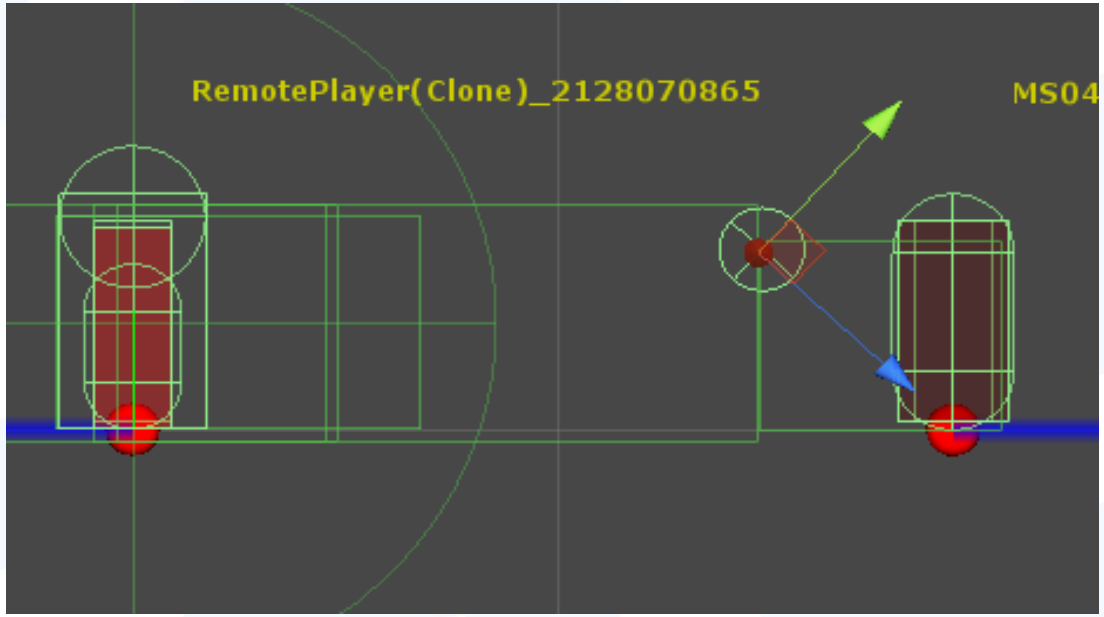
  b. 限制每帧最大创建数，子弹加速追赶DS位置

  c. 命中事件 效果合并，选择性丢弃

命中判定和反外挂思考

案例：

命中校验：投射型&组合型



S2C_OnRecvDamageInfo:
-- 飘字
-- 受击动作
-- 伤害效果
-- ....

BradcastSkillDamage(
AttackType _attackType,
int _decHp,
int _hitActionId,
...)

Client

DS

命中判定流程

- **本地预表现 + 服务器检测命中**

  -- 本地子弹仅碰撞模拟

  -- 命中后播放命中反馈（爆炸、命中效果等）

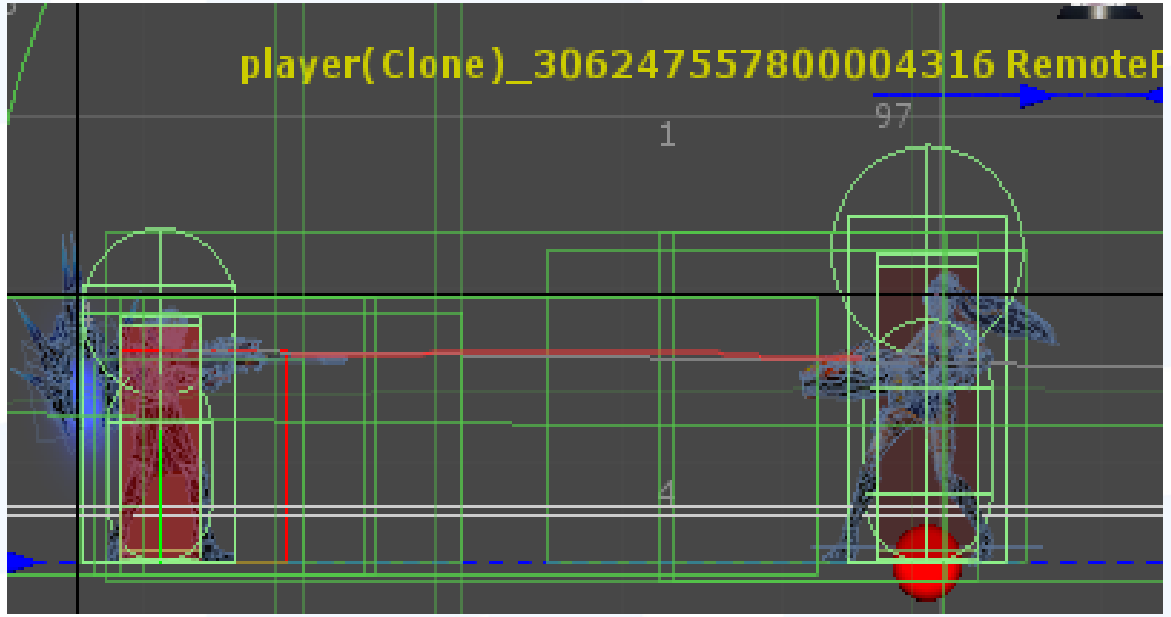
  -- **DS执行命中检测**

  -- 判定命中后广播结果给所有玩家

  -- 客户端收到命中结果播放飘字、扣血、执行伤害效果

- ✓ **DS作为权威，以公平为目的**

命中校验：瞬发型



SendHitToServer(
seqNo,
angle,
targetId,
startPos,
……)

Client

DS

C2S_OnHit :
-verifyPosition()
-verifyTarget()
-verifyRange()
-verifyBullet()
……

命中判定流程

- **本地命中上报 + 服务器校验**

  -- 本地子弹执行碰撞检测

  -- 命中后播放命中反馈（爆炸、命中效果等）

  -- **客户端上报命中消息**

  -- **DS执行命中校验并将命中结果广播给所有玩家**

  -- 客户端收到命中结果播放飘字、扣血、执行伤害效果

- ✓ **满足射击的人， 多数情况下射击是会成功命中的**

常见的作弊方式有哪些?

- 秒杀
- 穿透
- 瞬移
- 无敌
- 无限弹
- .....

单人副本(非联机模式)

- 内存加密
- 伤害采样
- 数值校验
- 数据上报
- 特征识别
- .....

✓ **依靠客户端关键数据上报，由安全组通过一些策略检测外挂**

组队&PVP(联机模式)

- 数据在服务器
- 关键逻辑在服务器
- 怪物AI在服务器
- **命中校验**
- 行为分析
- **位置验证**
- 持续迭代

✓ **依靠DS做为权威，不断升级防外挂策略**

命中校验

- **难点：**

  -- 大量信息依赖客户端（移动、瞄准、转向、命中）

  -- 子弹状态存在实时变化，DS仅靠命中上报命中信息，已无法精准判断作弊.

  问题：DS命中校验，如何有效防止外挂问题?

命中校验

- 射击行为监控， DS创建影子快照，采集2s内发射信息（不做物理检测）
- 记录历史玩家状态，支持按序号或时间回滚，加入超时机制
- 基于以上信息做坐标、朝向、发射频率、HitBox、射程、弹量等策略匹配
- 根据命中目标管理子弹生命周期
- 支持增加和策略配置

校验规则组合

- 一级规则：
  - 超时检查
  - 角色状态检查
  - 位置检查
  - 撞体检查
  - 技能检查
  - 子弹检查
  - 角色检查
  - 穿墙检查
  - 蓄能检查
- 二级规则：
  - 轨迹检查
  - 射程检查
  - 包围盒校验
  - 射击方向检查
  - 射击点检查
  - 弹药检查
  - 穿透检查
  - 频率检查
  - 跳转检查
  - 特殊武器检查

✓ **基于射击行为快照+角色状态快照 = 为外挂检测提供数值支撑**

新项目未来尝试方向

- 基于DOTS的游戏架构
- 基于帧驱动的状态同步
- 基于DS帧状态外挂检测机制
- 强化同步质量评估体系

腾讯游戏学院
TENCENT INSTITUTE OF GAMES

THANK YOU