1. 面对问题
   1. 64人同屏, 同步信息量大
   2. 即时对抗(炸弹人), 要求响应快速, 表现延迟低
   3. 二维格子, 碰撞简单, 地形信息简单, 位置信息简单
2. 技术点选型
   1. 帧同步(按时间帧来推动管理逻辑流程)
   2. 指令同步(只传递操作指令)
   3. 上传（UDP）+ 下发（UDP+可达性，顺序性保障逻辑）
3. 防作弊
   1. 非法/作弊操作 只能在本机进行表现, 其余客户端在运行操作指令时会有完整的检验逻辑, 会阻拦修正掉非法/作弊 操作;
   2. 服务端运行的战斗检验模块, 会对频繁出现 非法/作弊操作的客户端提出警告信息; 并同步最新状态信息, 强制矫正其非法操作带来的变化差异;
   3. 最终战斗结果以服务端跑的战斗校验结果为准, 非法/作弊 并不会改变战斗结果, 带来不正当的收益;
4. 同步流程图

时间

UDP

消息到达服务器的时间,决定进入哪一帧

UDP+

每帧到达结束时间点时,下发当前帧收集到的所有操作指令(即使没有任何玩家操作)

1. 客户处理流程图

输入(可多条)

Server

即时上传操作

Client

预表现

同步帧1; 执行

同步帧下发本帧的所有玩家操作指令

同步帧下发本帧的所有玩家操作指令

同步帧2,执行,修正预表现

同步帧下发本帧的所有玩家操作指令

同步帧3; 执行

1. 时间预估
   1. 服务器战斗同步部分, 估计 1 个月左右(不包括战斗校验模块部分)
   2. 客户端实现时间会多余服务器(主要业务逻辑实现方)
2. 异常情况预案
   1. 预表现不符

根据实际帧执行结果, “平滑”修正预表现 (技术难点之一, 需要细化讨论)

* 1. 上传消息丢失,乱序(UDP)

会带来某一次的操作失效, 只影响网络状况较差的玩家本身; 影响面较小;

出现在网络状况较差时, 是种可理解,可接受的现象;

乱序: 上传消息带上时间戳或序号 等信息, 服务器收到是进行比较, 延迟太久才到服务器的进行抛弃, 避免影响正常操作序列; 可能会稍微增加网络差的假象;

增加对应统计信息, 统计每场战斗, 每个玩家, 不同网络状况情况下 丢失/乱序带来的影响 有多大(理论上,不会太大); 根据统计信息再做后续调整优化;

* 1. 下发消息 丢失/乱序

使用 UDP 作为传输协议; 但UDP 本身不能保证 顺序和可达性；

将通过自行实现的规则 来保证 （详细说明和与TCP的比较在文档--基于帧同步的Udp方案—中进行说明）；

增加对应统计信息, 统计网络延迟情况, 为后续修正优化提供参考信息

* 1. 网络异常

上文 (3)(4) 两项都是建立在网络不通畅的情况下才会出现, 但 帧同步,指令同步 方案本身, 就会极大量的较少同步信息, 减轻网络压力, 减少网络状况不佳情况的出现;

1. 重连处理方案

两种重连恢复方式

1. 服务器下发全量帧指令集, 客户端从头快速运算帧指令--无表现, 来追上最新的帧
2. 服务器发现最近某一帧的状态全量数据, 以及后续的帧指令, 客户端在这一帧的基础上通过追帧的方式, 快速播放, 追上正常帧节奏;
3. 其他说明
   1. 战斗校验模块逻辑需要客户端实现, 最好是和客户端同一套
   2. 客户端需要做到 战斗逻辑 和 战斗表现 完全分离
   3. 预表现修改的数据 需要和真实数据分离, 真实数据只能又服务器下发的同步帧驱动
   4. 难点: 预表现 错误时 的平滑修正;
   5. 物理碰撞, 寻路 可以根据<炸弹人>这种特殊的应用场景(二维格子), 做出特殊优化, 不会太难;
   6. 需要做到屏蔽浮点数, 和服务器统一下发随机数种子