**基本流程**

单独做一个服务器，服务器比较简单，先等待客户端连接，连接后点击开始按钮，开始同步。期间可以点击结束按钮，结束同步。

4

3

2

1

点击开始按钮后：

（1）服务器进入第1帧，发送操作列表（是一个空列表）给客户端，客户端收到后执行第1帧，执行完成后发送帧结束消息；服务器在收到所有客户端操作后进入下一帧。

（2）服务器进入第2帧，将第1帧的结果发给所有客户端；客户端收到服务器消息后，执行第2帧，执行完成后把结果发送给服务器。服务器在收到所有客户端操作后进入下一帧。

（3）服务器进入第3帧，将第2帧的结果发给所有客户端；客户端收到服务器消息后，执行第3帧，执行完成后把结果发送给服务器。服务器在收到所有客户端操作后进入下一帧。

（4）重复上面的过程。

由于在等待所有客户端都收到后才进入下一帧，等待的时间有时就会比较长。如果服务器固定30fps，那么客户端固定帧率30fps，真正执行的帧肯定会低于30fps，测试只有23fps。

这时候可以提高服务器和客户端的fps，比如都提高到40fps，这样就可以获得30fps的实际帧率了。

要真正解决上面的问题，应该让服务器已固定帧率运行，比如30fps，如果一帧开始时没有收到客户端操作，则认为客户端客户端这一帧没操作，后面再收到时，忽略掉玩家的这个操作。也就是说，只有玩家操作的帧，是服务的当前帧，这个操作才有效。另一方面，如果玩家比服务器帧要落后，则本次会一次性返回所有落后的帧，客户端快速播放落后的帧，直到跟上的服务器的节奏。比如客户端处于100帧，服务器处于110帧，则一次性返回101-110帧，客户端加速播放完10帧，追赶上服务器。这种模式下，客户端帧基本稳定在30帧上下。

采用TCP和UDP都可以，暂不考虑掉线的判断。为了简单处理。UPD有速度优势，但是容易丢包。

**服务器**

如上所述。

（1）客户端发起连接请求，连接上后给客户端发送玩家列表

（2）点击开始按钮，服务器给客户端发送第一帧的操作列表。

（3）点击结束按钮后，服务器给客户端发送结束消息。这时候当前这一帧就作废了，客户端的回复也作废。

**客户端**

（1）给服务器发送请求连接请求，收到玩家列表后待命。

（2）收到第一帧消息后，开始同步

采用最简单的设计，几个box具有在空间中移动的行为，可以玩家操作进行前后左右移动，同时AI会自由活动。

**消息通信**

连接请求 & 连接成功回复

开始消息 & 结束消息

客户端发送操作给服务器 & 服务器发送操作给玩家