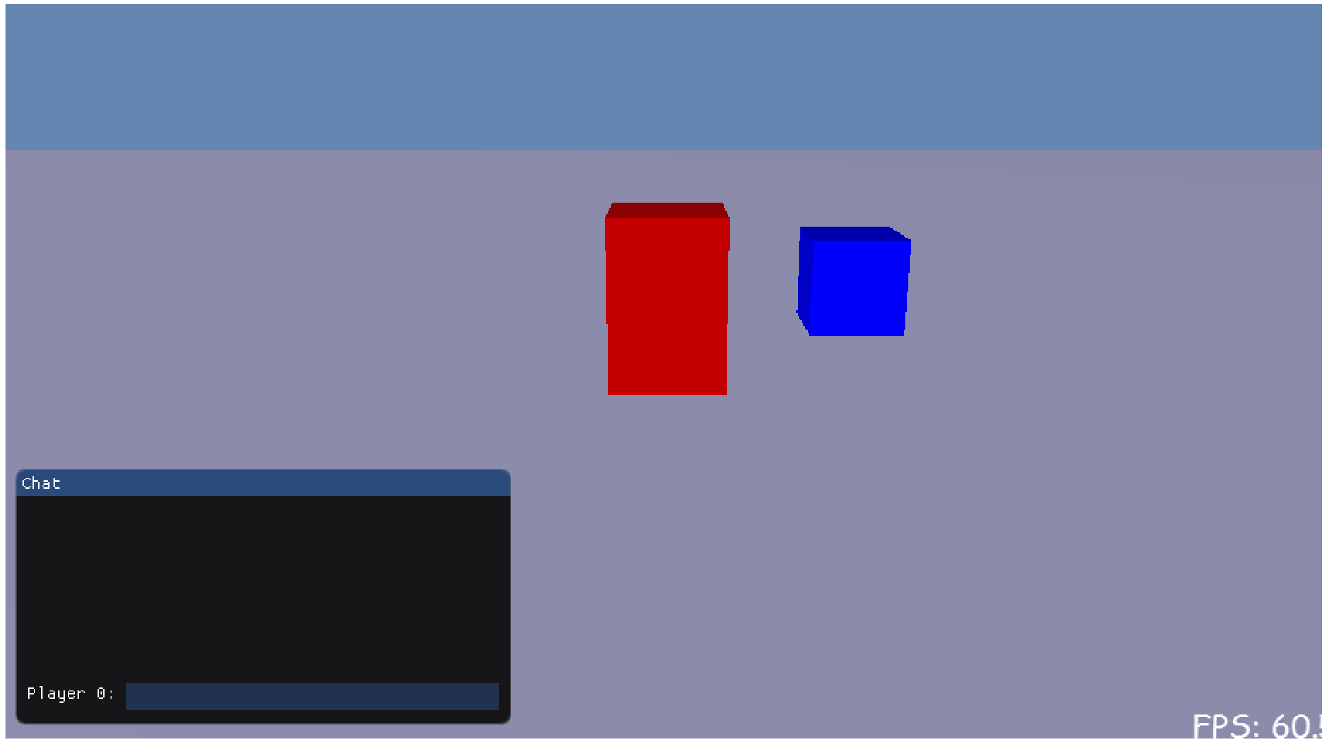
CMP303

Networking

Program Overview

- Designed with a first-person shooter in mind

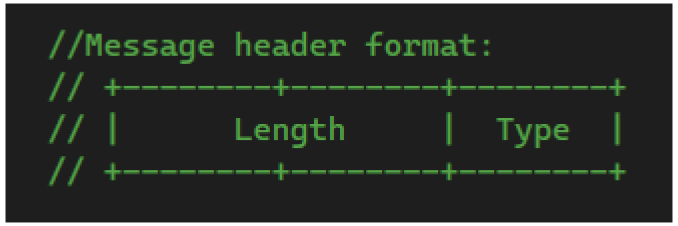


Client-server

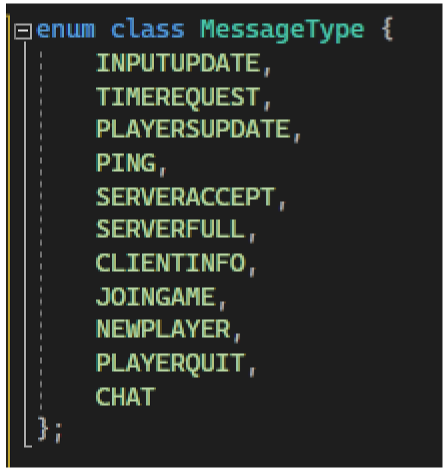
- Competitive game, so people will want to cheat
- Client server negates the effect of sending modified position updates etc, as the server always decides

Application layer protocol

- Chat feature – TCP
- Position updates – UDP
- Position packet: time stamp, velocity, rotation, bool to request jump
- Serialisation of structs



```
//Message header format:
// +--------+--------+--------+
// |      Length     |  Type  |
// +--------+--------+--------+
```



```
enum class MessageType {
    INPUTUPDATE,
    TIMEREQUEST,
    PLAYERSUPDATE,
    PING,
    SERVERACCEPT,
    SERVERFULL,
    CLIENTINFO,
    JOINGAME,
    NEWPLAYER,
    PLAYERQUIT,
    CHAT
};
```

MessagePack

Loomis (2019) cppack. Available at: https://github.com/mikeloomisgg/cppack/tree/master (Accessed: 5 January 2024)

Protocol continued - Handshake

- Server accepts connection
- If full sends ‘server full’ message and closes connection
- Else sends accept message
- Client receives this and sends back its UDP port number so server will accept its UDP packets. Also starts synchronising (more later).
- Server receives this and tells client to join, assigns a player ID and tells client what other player IDs are active. Tells all other clients a new player joined

Time synchronisation

- Synchronises the clock when player joins, trusts both machines' clocks to be accurate
- Client sends 15 time stamped messages 300ms apart.
- Server sends them back with its own time stamp.
- Client works out the latency from this and sets its clock = server time + latency, using the lowest latency measured
- If the client doesn’t hear from the server it keeps retrying

Network API

- Winsock – lower level than some alternatives, so I felt I would learn more

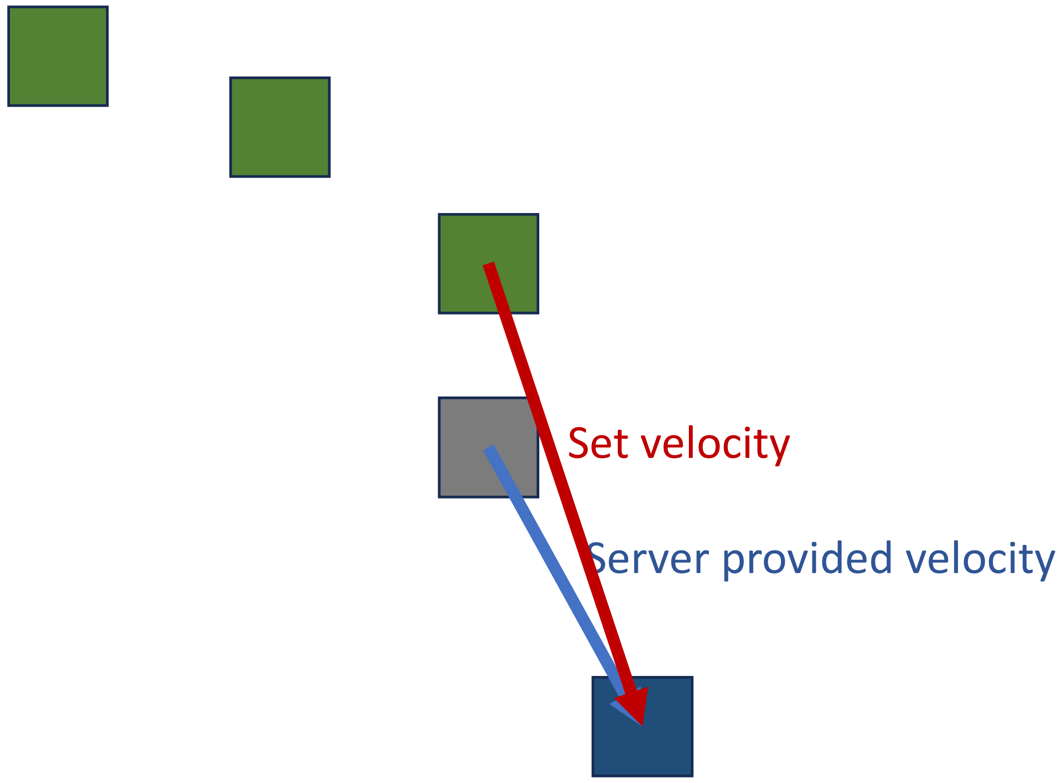
Network code

- Event based, asynchronous IO
- Two separate threads, one for TCP and one for UDP, wait efficiently with infinite time-out value (except server UDP thread)
- When the application wants to send a message, it’s added to a queue till it can be sent. The rest of the program continues (not blocking)

- Server maintains a list of connections each with their own queue

Prediction and Interpolation

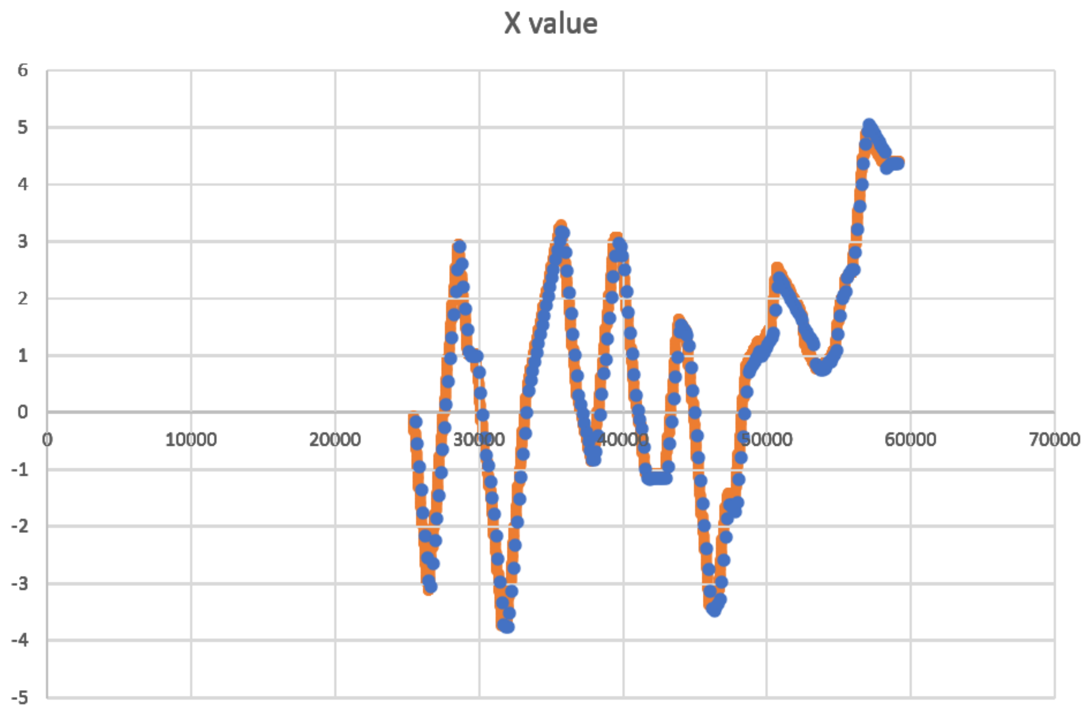
- Linear prediction – PhysX will just continue simulation using last sent velocity. No friction so will continue forever.
- Interpolation:



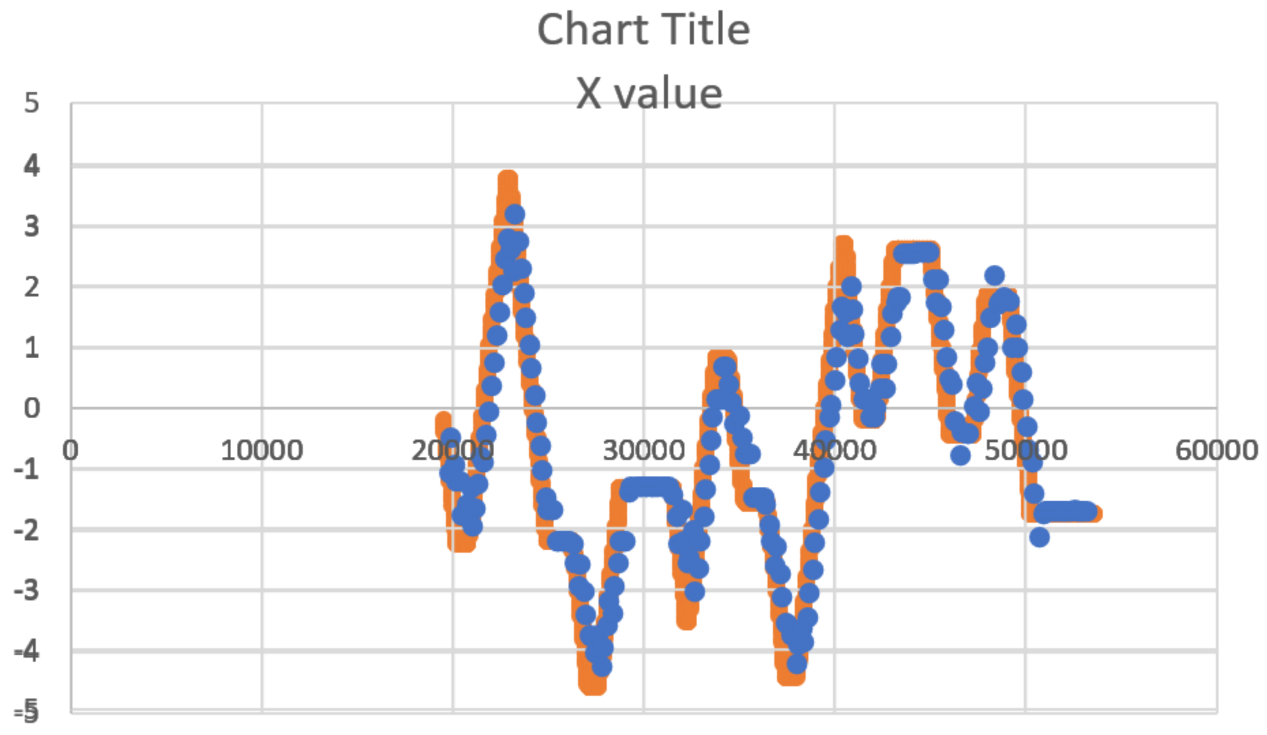
Other

- Client sends input update whenever player moves. Had to limit to one send every 2ms, otherwise network becomes clogged.

Effectiveness and testing



0 ping



300ms ping

Thank you