B5 - Advanced C++

B-CPP-501

Advanced R-Type

Annexes for Advanced Topics

4.0-wip

EPITECH.

Annexes

Advanced Server

Lag Compensation

Client-side prediction/simulation

To reduce the perception of lag, a smart things to do is what you might have already started to implement up to this point: the client is not just a "dumb" terminal only doing rendering and input handling, but also computes the game logic part. This technique is a form of "client-side prediction".

The general idea: both the client and the server runs the same game simulation (or possibly a simplified one on the client, only focusing on entities movements). The client immediately apply its local inputs to its local simulation and render the results before sending them to the server. Such system allows to not have to wait for the server for each game updates, making things smoother.

If you implement the full game logic on the client, beware of your RNGs ("Random Number Generators") ! They need to behave the same on the client and the server…

Server reconciliation

The previous technique does not come without issues: at some point in time, the following events could happen:

- Game state desynchronization: maybe the server computed the player died, although the client assume the player have just moved because he does not have all information the server already have.
- Timing issues: we receive a "past state" already processed locally AND we already applied new local inputs the server does not have yet.
- CPU timers issues: they are not being the same between machines, leading a time delta to grow progressively between the client and server.
- Unwanted bugs: for example, RNG not giving the same results on client and server, differences in floating point computation between machines, etc.

Hence, there is the need to implement some form of synchronization with the server, to fix the entity state to the real state as computed by the server. This is called "reconciliation" or "blending".

To reduce the occurrence of desynchronizations, you might want to add some small "input delay" (a couple of frames are processed before inputs are effectively applied). This helps a bit as it give some time to the packets to reach servers and other clients, but come with a big drawback: if there is too much delay, the game will feel unresponsive.

Alternatively, you might want to "rewind" or "rollback" client game state to the actual server state when received, and "replay" the game logic up to your client current time (possibly with local input information the server does not have yet).

Entity State Interpolation

In case of a prediction error, some large differences could be observed in positions between predicted state and server state, leading to potential "hops/jumps" of positions to their corrected position. Interpolation might be useful in this case: the idea is to smooth these "hops" by interpolating state between predicted and corrected state amongst several frames.

Finally, even though your client runs the game logic, it cannot decide what the others players will do. It is maybe possible to implement some form of remote entities' prediction, or extrapolation of their behavior based on their characteristics (position, velocity, etc.) or previous behavior (last inputs, etc.).

Further Readings

Networking development in video games is a really vast topic. For the curious, here is a link to a curated list of well-done articles, talks, libraries and tools:

- `https://github.com/ThusSpokeNomad/GameNetworkingResources`
- A small (online) application allowing you to better understand the issue of lag (client A, server, client B have different views of the world at time T): `http://ernestwong.nz/crystalorb/demo/`

In particular the following articles give some nice introduction to the various problems and techniques:

- Gaffer On Games introduction to game networking
- Game Networking Demystified, Part I: State vs. Input
- Explaining how fighting games use delay-based and rollback netcode