Development process

Most of the development time was spent on attempting to implement server authoritative lag compensation features that would be similar to something that could be used for an actual project. Unfortunately Netcode for Gameobject does not make it simple and I had to abandon all attempts (some can be found under the “legacy” folder in the player script in the project).

I also avoided the usual solution found online, which is to send transform updates every frame to trivialize prediction. The final solution only sends transform updates on input change.

In general these feature where some form of:

Client side reconciliation

With this system, any update by the server is compared to a state computed on the client. If the two values are two different, the client value is corrected.
It can be accomplished in various ways depending on the kind of variable and their rate of change (IE. it is relatively easy to reconcile a sporadically changing integer value, but hard to reconcile complex movement cleanly).
The final project does implement something with the same philosophy.

Client side prediction

This system, unusually combined with reconciliation if its precision is not perfect, is meant to spend additional computation client side to try to anticipate server values to minimize the need for reconciliation. It also allows clients to respond to input seemingly immediately.

This can be more or less subtle:
-On the easiest side, you could reflect a change client side immediately knowing that it will be confirmed by the server soon (ie a discrete color change is the example Unity give in their documentation, which is about the scope of what the API supports out of the box).
-On the more difficult (and relevant) end, having client side heuristics to anticipate the server state can be used to implement a continuous reconciliation. This is unfortunately very difficult to do on NGO while keeping server authority.

Simulation determinism

In theory, with a sufficiently deterministic simulation it is possible to keep all clients in sync by just passing inputs over the network, with some client side prediction and reconciliation for smoothness. This solution is provided by custom engines such as Photon Quantum (built on top of Unity).

Unfortunately it was very difficult to make very deterministic simulations (fixed step simulation with no physics) keep in sync using NGO. Mostly because of the tick system and the unwieldy API for reconciliation.

Game

I implemented a very simple top down shooter game.
Local player capsules can be moved with WASD and projectiles can be shot with the mouse click.

Players can take three shots before being destroyed and respawning after 3s.

Simple UI health bar and respawn countdown are also implemented.
Players don’t collide with each other.

All action, health tracking and despawning/respawning are server authoritative.

Final implementation

Syncronisation

In the end I implemented a simple reconciliation system for player movement:

-Every input change is sent to the server.
-Server applies the input and sends it to all clients with its current transform state for the relevant gameobject.
-Upon receiving the server update, clients compare the transform state with the state they had for the object **when the input was received by the server.** The difference between the two states is used as an estimation of how much the client has “drifted” from the server.
-The computed error is progressively added to transform state over the next few frames to bring the client state closer to the server state.

I initially tried to do this correction only along the current movement vector, but many edge cases caused divergences when using less optimal network simulation scenarios (if several inputs were digested on the same server tick it would cause a jump in error that negatively affected game feel). I ended up with a naive implementation that works better while in movement but does cause some drift when movement stops.

All implementation for this is self contained in Scripts/Player/PredictedPlayer.cs

Server authority

All logic is server authoritative and when logic runs on the client it is usually in a predictive way and will be overridden or reconciled with the server state.

There is not much to say about this as it mostly affects the design of network related systems.

Most server methods are designed to not actually affect the game state in a way that would propagate to the server if called on the client. The only data clients can send to transform the server game state is player input, which is processed when it is received (no server side rollback).

Optimization

The major “optimization” compared to most toy examples using this API is that server updates only happen on value change (using networkVariables) and on Input (using RPC methods).
The implementation tries to keep network traffic to the strictest minimum.

Learnings

This was generally a frustrating experience as the API seems to mostly be meant for client authoritative co-op games, and if I had the choice I would have used any of the many mature networking APIs that third parties have developed over the years.
However it was a good opportunity to read older papers and blogs about network synchronisation and it did give me a better theoretical grasp on some things like client side prediction and some server roll back implementation.