NetworkingAssignment

Elin Gammelli

17th january 2025

1 Initial Idea

Initially I wanted to implement client-side prediction, and only send client input somehow. As I was pondering how this would be implemented, I realized that this was a very complex task, and that if I were to do this, there would be a high probability that I could not meet the deadline.

The issues that were of concern is. The input packet has to be built manually, to have smooth gameplay, the game would still need to store data of all the input every frame, then pack this into an input packet that is sent out not too frequently. Then the server would simply reconstruct and validate these, and perhaps give an OK or a new position for the player.

The above is not too difficult, the difficulty would lay in the synchronization. The server would need to inform all other clients to move these empty shells of player cubes about the movements of each player. Which again, should not be too difficult. However, as soon as synchronization comes into play it becomes very difficult, because a real time line does not exist, and any player with a RTT that varies would be very difficult to ascertain the difference in timeline between players and server. This could cause issues in hit detection, and the while the server may decide on some hit detection, it may look very wrong on the clients.

While the above was interesting to think about, the implementation was too much, and I decided to do the bare minimum, or thereabouts. Additionally, If I were to make a game with networking beyond e.g. card games, I would use Netcode for Entities.

2 Implementation

I created a player which had a NetworkTransform, which makes synchronization of player position trivial, and I could merely move the cube locally, and it'd move on the server and the server would let everyone else know about the position. This is client authoratitive movement, and of course lets players cheat however much they want.

The rules for shooting is also handeled clientside (again, because of synchronization difficulties, and ensuring smooth gameplay for players with a poor connection). However, only the server spawns and moves bullets (by the Server-Manager). The bullets move on their own, with a NetworkTransform, and so the synchronization is trivial.

The hit detection of bullets are done by the bullets themselves, but only on the server instance, checking a physics sphere for the players in the player layer. If it detects a hit, the bullet informs the ServerManager (only exists on the server) that at hit was found, and so the Servermanager despawns the bullet and spawns an BulletExplodeSync, which is a NetworkObject that merely exist to inform all players, even those that joined after the hit, that a hit has occured, with infromation of where.

As players get the the information of the BulletExplodeSync, they will spawn a local particle effect to display the explosion. Later, the bullets on the Server will clean up the BulletExpodeSync NetworkObjects by themselves, and the local explosions will too clean themselves up.

3 Difficulties and optimizations

Since much of the networking was handeled by Unity itself, the networking was not too difficult. I started with spawning the players themselves, which gave a better idea of how this was done. There were some difficulties in knowing what e.g. IsServer or !IsServer would actually mean. For instance, one would expect the client host to be !IsServer, as one would imagine the Server to exist in another domain, but that is not the case.

Network Variables were somewhat tricky, in that you'd expect values to change very quickly, so that if they were set right after spawn, other clients would get those values in the same information packet. But, that was not the case. Initially I solved the issue of desync due to network variables being updated some time after the spawn, by subscribing to the OnValueChange to notify non-networked objects that data had changed (BulletExplosion, to know where it should explode). Though, later on, I optimized this by simply not spawning the local explosion particle system unless this network variable data had changed.

Since the bullets were synced with NetworkTransform, the interpolation made the updates very slow, which made bullets suddenly get deleted even though they were far away from hitting a target, for clients. To solve this, I simply removed the interpolation on their NetworkTransform, and added slerp position, to improve this appearance, at least for low ping clients.

4 Conclusion

To make real improvements to sychronization and player experience, client side prediction would have to be implemented. The use case for Netcode for GameObjects appears very narrow, only a fit for highly casual or turnbased games. Implementing client side prediction to Netcode for GameObjects would also seem a questionable decision, as it's in general slow, and one may wish to handle the transportation of packets in some other way than rpc. So, in conclusion, for any non hyper-casual or turn based games, I believe one should learn Netcode for Entities instead (Or use e.g. Mirror), for as far as I know, this package already has clientside prediction and rollback.