Lab 3

We have tried Client Authoritative Network Transform and Animator to sync basic movement and animation. Now we are going to do Server Authoritative Network Transform and Animator so you have the comparison and this also indicates that to do a functionality you have the option of choosing whether to use server or client / owner authoritative method.

RPC is one of the method to communicate between server and client which normally used for syncronisation. RPC consists of 2 main functions ServerRPC and ClientRPC. ServerRPC is basically only run a function on the server which can be used to send data from client to server. ClientRPC is used for otherwise which allow the server to “message” all clients. ClientRPC is normally called from the server.

A diagram of a process

Description automatically generated

Client can invoke a server RPC on a Network Object. The RPC will be placed in the local queue and then sent to the server, where it will be executed on the server version of the same Network Object.

At a high level, when calling an RPC client side, the SDK will take a note of the object, component, method and any parameters for that RPC and send that information over the network. The server will receive that information, find the specified object, find the specified method and call it on the specified object with the received parameters.

When calling an RPC, you call a method remotely on an object that can be anywhere in the world. They're "events" you can trigger when needed.

If you call an RPC method on your side, it will execute on a different machine.

Netcode has two variations of RPCs to execute logic on either server-side or client-side: ServerRpc and ClientRpc.

A diagram of a software company

Description automatically generated with medium confidence

Server can invoke a client RPC on a Network Object. The RPC will be placed in the local queue and then sent to a selection of clients (by default this selection is "all clients"). When received by a client, RPC will be executed on the client's version of the same Network Object.

To use RPCs, make sure

* [ClientRpc] or [ServerRpc] attributes are on your method
* Your method name ends with ClientRpc or ServerRpc (ex: DoSomethingServerRpc())
* your method is declared in a class that inherits from NetworkBehaviour
* your GameObject has a NetworkObject component attached
* Make sure to call your RPC method server side or client side (using isClient or isServer)
* Only accept value types as parameters

1. Just for the purpose of this exercise and for you to compare on the performances, you can just create a duplicate of the Player prefab. You can name the new Player prefab *PlayerServerAuthoritative* (you can duplicate it by drag the Player prefab into the scene to become a gameobject and then drag the gameobject back to the Assets folder)
2. Reassign the Player Prefab on the NetworkManager

A screenshot of a computer

Description automatically generated

1. Replace the ClientNetworkTransform with NetworkTransform component on the Player prefab and for now delete the movementPlayer component as we are going to create a new script for the server authoritative using RPC
2. Through RPC we can send a message to the server to notify that there is a change in state in one of the player. The server will then notify all clients the change that happen. We are going to revise the code on the FixedUpdate to call a ServerRpc method called *HandleMovementServerRpc* and in this case we will pass an integer that would indicate the direction of the movement (1 = move right, 2 = left, 3 = up, 4 = down) and the id of the client that does the movement. It is worth noting here that any ServerRpc method should have suffix *-ServerRpc* as noted at the start of this document.

if (IsOwner)

{

if (Input.GetKey(KeyCode.RightArrow))

{

HandleMovementServerRpc(1,this.NetworkObjectId);

// transform.position += new Vector3(speed \* Time.deltaTime, 0f, 0f);

}

if (Input.GetKey(KeyCode.LeftArrow))

{

HandleMovementServerRpc(2, this.NetworkObjectId);

// transform.position -= new Vector3(speed \* Time.deltaTime, 0f, 0f);

}

if (Input.GetKey(KeyCode.UpArrow))

{

HandleMovementServerRpc(3, this.NetworkObjectId);

// transform.position += new Vector3(0f, speed \* Time.deltaTime, 0f);

// movementAnimator.SetBool("isJump", true);

}

if (Input.GetKey(KeyCode.DownArrow))

{

HandleMovementServerRpc(4, this.NetworkObjectId);

// transform.position -= new Vector3(0f, speed \* Time.deltaTime, 0f);

}

}

1. The server then will notify all clients to update the position of the client character who moves and will invoke a method called HandleMovementClientRpc that is accessible by all clients

[ServerRpc]

void HandleMovementServerRpc(int movementdirection, ulong theIDOftheCharacterThatMoves)

{

Debug.Log("the player " + theIDOftheCharacterThatMoves + " just moves from position " + NetworkManager.Singleton.ConnectedClients[0].PlayerObject.transform.position);

HandleMovementClientRpc(movementdirection);

}

1. HandleMovementClientRpc itself controls the movement according to the movement direction

switch (movementdirection)

{

case 1:

transform.position += new Vector3(speed \* Time.deltaTime, 0f, 0f);

break;

case 2:

transform.position -= new Vector3(speed \* Time.deltaTime, 0f, 0f);

break;

case 3:

transform.position += new Vector3(0f, speed \* Time.deltaTime, 0f);

movementAnimator.SetBool("isJump", true);

break;

case 4:

transform.position -= new Vector3(0f, speed \* Time.deltaTime, 0f);

break;

}

Task:

* Compare the performance of this approach to the client-authoritative approach we did last week.
* Continue to add run animation to the player