Lab 2

Continuing from the last week’s lab, first we are going to do movement and then basic sync particularly for the movement between the players.

# Create a basic movement

1. For this example, we are just going to use a very simple movement. Create a C# script called movementPlayer.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Unity.Netcode;

public class movementPlayer : NetworkBehaviour

{

public float speed = 5f;

public override void OnNetworkSpawn()

{

//this section will be triggered when a player enters / spawned into the game

}

void FixedUpdate()

{

if (Input.GetKey(KeyCode.RightArrow))

{

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

}

if (Input.GetKey(KeyCode.LeftArrow))

{

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

}

if (Input.GetKey(KeyCode.UpArrow))

{

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

}

if (Input.GetKey(KeyCode.DownArrow))

{

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

}

Debug.Log(transform.position);

}

}

This is just a basic movement script which you can control the character with the arrow button. A few important things that indicate the use of Netcode:

* You would need to include the Netcode library (using Unity.Netcode;)
* Notice the script is inherited from NetworkBehaviour class

1. When you test it out you can see that you basically control all Player characters together. So what you need to do is to indicate that only the owner can control the character. To do so you can just add a condition to check if it is the owner before checking the input key movement

if (IsOwner)

Task:

For this week you would need to decide the player mechanic on the game you are intending to create. You then need to develop the player mechanic and add these basic networking functionalities into your player mechanic

# Sync the movement – client authoritative

Unity provides 2 components to automatically sync the movement and animation

1. Click on the player prefab
2. As of this time of writing, Unity by default provides a server-authoritative **NetworkTransform** component. More details information about NetworkTransform can be found in <https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/>.
3. To test this out, you can add the NetworkTransform component to the **Player** prefab. Build and run to test the game.
4. During the test you can see that any movement on the Host (remember that the Host is technically the server + the player/client) and every movement from the Host is always sync to the client however the movement from the Client wont be sync to the server. This is an example of server-authoritative. As it is a server authoritative, any changes on the client has to be sent to the server which then will be automatically catch by the server-authoritative NetworkTransform.
5. Apart from server-authoritative NetworkTransform component, there is a ClientNetworkTransform however this functionality at the moment is not fully supported by Unity hence you would need to download it manually as indicated in the changelog of Netcode for Gameobjects (<https://docs.unity3d.com/Packages/com.unity.netcode.gameobjects@1.5/changelog/CHANGELOG.html>). So to use ClientNetworkTransform, you can download the file ClientNetworkTransform.cs from [com.unity.multiplayer.samples.coop/Packages/com.unity.multiplayer.samples.coop/Utilities/Net/ClientAuthority/ClientNetworkTransform.cs at main · Unity-Technologies/com.unity.multiplayer.samples.coop · GitHub](https://github.com/Unity-Technologies/com.unity.multiplayer.samples.coop/blob/main/Packages/com.unity.multiplayer.samples.coop/Utilities/Net/ClientAuthority/ClientNetworkTransform.cs) as indicated from the changelog.

A screenshot of a computer

Description automatically generated

1. Once you download the file, add the file to the project by dragging the file directly to the Scripts folder in your Assets
2. The ClientNetworkTransform script basically override the OnIsServerAuthoritative on the NetworkTransform from the default value of true to false
3. Add the ClientNetworkTransform as component to the Player. If you have NetworkTransport component already, you would need to delete it first before adding the ClientNetworkTransform.
4. Test it out and you should be able to see that all movements are sync.

Task:

* As there wont be any changes on the Scale and Z-axis position, you can just disable these by unticking the boxes. By reducing the information to be automatically sync, the performance / network traffic should be better
* Play around with the configuration on the ClientNetworkTransform particularly on the interpolation, Slerp Position, Quaternion Sync and Half Float Precision

# Sync the animation – client authoritative

Now we are going to replace the temporary white square as the player character with an animation and we are going to sync the animation in the network

1. So we are going to use a free asset from the assets store which you can also download directly from GCULearn. The link to download is <https://assetstore.unity.com/packages/2d/characters/sunny-land-103349>.
2. Add the downloaded assets to your project
3. Click on the player prefab on the Assets folder and replace the Sprite (in the Sprite Renderer component) to player-idle sprite (the sprite is located in folder SunnyLand Artwork -> Sprites -> Player -> idle

A screenshot of a computer

Description automatically generated

1. You can add RigidBody2D and BoxCollider2D to the Player Prefab. Adjust the Scale of the X and Y accordingly. You can freeze the Z-rotation so the player wont tip and rotate when colliding with other objects in the game
2. The free asset provides the animation for idle however not the Jump animation. So we are going to create a Jump Animation first
3. Open the Animation Window (Window -> Animation -> Animation) and then click on the Player. Click on the Player object and then on the Animation Window click on the animation name and then Choose Create New Clip and save it to folder SunnyLand Artwork -> Sprites -> Player -> Jump where the jump sprites are located. Name the new animation clip **jumpAnimation** and drag and drop the jump sprites into the timeline

A screenshot of a video game

Description automatically generated

1. Click on the animation clip **jumpAnimation** you created and disable the Loop Time

A screenshot of a computer

Description automatically generated

1. Next we are going to put everything together into an Animation Controller which will control which action to show when a button is pressed. To do so, right click on the Assets folder (or any folder you want to store the controller) and create an Animator Controller (Create -> Animator Controller) called *playerAnimation*
2. Double click and open *playerAnimation* Animator Controller. Drag and drop the idle animation clip (SunnyLand Artwork -> Sprites -> Player -> idle) to the Animator Window. You should then see an arrow automatically forming from Entry state to Idle-Animation state. This basically indicate that the animator will start from the Idle animation clip.
3. Add the jump animation clip (SunnyLand Artwork -> Sprites -> Player -> Jump) to the Animator and jumpAnimation state will be created automatically
4. We then need to create a condition which will trigger transition between the states. In this case you need a variable to control the transition which in this case the condition would be “transition from idle animation to jump animation if the player press a jump button and return to idle animation afterwards”. So first lets create a variable called isJump with a type of Boolean and the default value of “false”. To do so, select the tab “Parameter” on the Animator Window and then click on the “+” button and choose Bool. Rename the variable to *isJump*

A screenshot of a computer

Description automatically generated

1. Right click on the *Idle-Animation* state and choose *Make Transition* which would automatically create an arrow you will point to the *jumpAnimation* state. This indicates that there will be a transition from idle to jump animation.
2. Click on the arrow between the idle to jump animation and customise the transition. In this case you can disable the Exit Time and Fixed Duration and manually customise how your animation would be in the graph. You also need to assign the value of the variable to the condition to trigger the transition which in this case *isJump* should be True

A screenshot of a computer

Description automatically generated

1. Right click on the *jumpAnimation* state and choose *Make Transition* which would automatically create an arrow you will point to the *Idle-Animation* state. This indicates that there will be a transition from jump to idle animation.
2. Click on the arrow between the jump to idle animation and customise the transition. In this case you can disable the Exit Time and Fixed Duration and manually customise how your animation would be in the graph. You also need to assign the value of the variable to the condition to trigger the transition which in this case *isJump* should be False

A screenshot of a computer

Description automatically generated

1. Add an Animator component to the Player prefab and assign the Animation Controller with *PlayerAnimation* animation controller we created earlier

A screenshot of a computer

Description automatically generated

1. Now we are going to use this Animator in the movementPlayer script. First we need to create a variable for the animator and then on the NetworkSpawn we want to capture the animator of the spawned object

private Animator movementAnimator;

public override void OnNetworkSpawn()

{

movementAnimator = this.GetComponent<Animator>();

}

1. Activate the jump animation when a button jump (up button)

if (Input.GetKey(KeyCode.UpArrow))

{

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

movementAnimator.SetBool("isJump", true);

}

1. Create a new object tag called *Grounds* and Disable the jump animation when the player is on the ground

void OnCollisionEnter2D(Collision2D target)

{

if (target.gameObject.tag.Equals("Grounds") == true)

{

movementAnimator.SetBool("isJump", false);

}

}

1. To sync the animator, you need to add NetworkAnimator component. In Netcode, there are 2 types of NetworkAnimator based on the type (<https://docs-multiplayer.unity3d.com/netcode/current/components/networkanimator/index.html>). For now we will create a client / owner authoritative network animator.
2. Create a client authoritative network animation script called *OwnerNetworkAnimator* and paste the code below

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Unity.Netcode;

using Unity.Netcode.Components;

public class OwnerNetworkAnimator : NetworkAnimator

{

protected override bool OnIsServerAuthoritative()

{

return false;

}

}

1. Similar to the ClientNetworkTransform script, the OwnerNetworkAnimator script basically override the OnIsServerAuthoritative from the default value of true to false on the NetworkAnimator
2. Attach *OwnerNetworkAnimator* script to the Player prefab and build and test the project you should be able to see now all client animation should sync.

Notes:

A diagram of a company's timing

Description automatically generated

In the above diagram, it shows that the owner client has an Animator state change that's detected by the NetworkAnimator ( OwnerNetworkAnimator) which automatically synchronizes the server with the changed state. The server applies the change(s) locally and then broadcasts this state change to all non-owner clients.

**Owner authoritative model benefits:**

* The owner is provided instant visual feedback of Animator state changes, which does offer a smoother experience for the local player.

**Owner authoritative model drawbacks:**

* Non-owner clients lag behind the owner client's animation by roughly one full RTT.
* A host lags behind the owner client's animation by roughly half RTT.

Task: Create a walking animation and sync it