Lab 6

This lab sheet will show you an alternative way to do the UI and scoring. Compared to last week when the clients can just have a look at the objects that literally exist on the built, this week we will use only the server can write the relevant information.

This lab is written so that you can put together with the last week’s lab materials so instead of trying to use the same object to interact with, a new object (with a new tag) will be created and used on this lab sheet.

1. Create a 2d square object (2D objects -> Sprites -> Square) and add RigidBody2D with Static type and a Box Collider 2D so this object can collide with the player
2. Assuming there could be many square objects you want to have later on, we will create a custom tag for the square objects. In this case lets name the custom tag to “Mushrooms”
3. A UI then needs to be setup and this time its using a UnityEngine.UI. So right click on the Hierarchy and choose UI -> Canvas. This will create a canvas for the UI and lets name this canvas “gameUI”

A screenshot of a computer

Description automatically generated

1. Create a script for the gameUI object and lets the call the script gameUI.cs and paste the code below

public class gameUI : MonoBehaviour

{

List<char> UIName;

Dictionary<ulong, GameObject> scoreGOs;

// Start is called before the first frame update

void Start()

{

UIName = new List<char>();

scoreGOs = new Dictionary<ulong, GameObject>();

}

public void AddScoreUI(ulong playerid) {

int x = 0;

if (!UIName.Contains((char)playerid)) {

GameObject newGO = new GameObject(playerid.ToString());

Text myText = newGO.AddComponent<Text>();

myText.text = playerid + " has score of 0";

Font ArialFont = (Font)Resources.GetBuiltinResource(typeof(Font), "Arial.ttf");

myText.font = ArialFont;

myText.material = ArialFont.material;

myText.fontSize = 14;

newGO.transform.localPosition = new Vector3(120, 50 + (10 \* scoreGOs.Count), 0);

newGO.transform.SetParent(this.transform);

newGO.GetComponent<RectTransform>().sizeDelta = new Vector2(150, Screen.height);

UIName.Add((char)playerid);

x++;

scoreGOs.Add(playerid, newGO);

}

}

}

1. To detect collision between the player and the mushrooms and to update the score accordingly, we are going to update the movement codes (which if you follow the lab sheet your current active movement script is *movementPlayerRpc*. First we are going to create a Network Variable with a type of Integer called theScoreForEachPlayer. We are also going to give an initial value of 0 and most importantly we are going to set read permission to allow everybody (server and client) and write permission to Owner only.

private NetworkVariable<ushort> theScoreForEachPlayer = new NetworkVariable<ushort>(default,NetworkVariableReadPermission.Everyone, NetworkVariableWritePermission.Server);

if you recall last week the networkvariablewritepermission was on the owner while this time really only the server can change the network variable.

1. We are then going to check the collision so lets create an OnCollisionEnter2D to check if the player collides with the object with a tag “Mushrooms”.
2. When a new player joins and spawned into the game, a subscription will be created to monitor any change in the network variable theScoreForEachPlayer. On the script below, it basically will call a function *scorechanged* when the value of the network variable theScoreForEachPlayer is changed

public override void OnNetworkSpawn()

{

theScoreForEachPlayer.OnValueChanged += scoreChanged;

}

1. If the player collide the Mushrooms, only increase the networkvariable called theScoreForEachPlayer for the particular network object that collides with the mushroom is the owner. So create a new script called MushroomsScript that would handle the collisions with objects (alternatively you could also just add the code into the onCOllissionEnter2D on the player to check if it is colliding with the mushrooms)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Unity.Netcode;

public class MushroomsScript : NetworkBehaviour

{

private void OnCollisionEnter2D(Collision2D collision)

{

if (!collision.gameObject.CompareTag("Player")) return;

if (!NetworkManager.Singleton.IsServer) return;

if (collision.gameObject.TryGetComponent(out movementPlayerRPC updatenetscoring))

{

updatenetscoring.addScore();

}

}

}

1. So then on the method addScore() in the movementRPC script you can add the value of the network variable

public void addScore()

{

theScoreForEachPlayer.Value += 1;

}

1. When method *addscore*() is called to increase the network variable, it will automatically call the method *scoreChanged*() because its already subscribed. The score then will be updated on the UI. So in the *scoreChanged* method:

public void scoreChanged(ushort oldValue, ushort newValue)

{

theScore[OwnerClientId] = theScoreForEachPlayer.Value;

GameObject.Find("gameUI").GetComponent<gameUI>().modifyScoreUI(NetworkObjectId, newValue);

}

1. In the gameUI.cs, add a new method modifyScoreUI to update the score:

public void modifyScoreUI(ulong playerid, ushort updatedScore)

{

scoreGOs[playerid].GetComponent<Text>().text = playerid.ToString() + " has score of " + updatedScore;

}

To allow late joining:

1. Create a dictionary on the local player / client

public Dictionary<ulong, ushort> theScore;

1. On NetworkSpawn, initiate the variable theScore

theScore = new Dictionary<ulong, ushort>();

1. On the NetworkSpawn, call a serverRPC to notify that there is another player logged into the game

if (IsClient)

sendDataToScoreUIServerRpc(this.NetworkObjectId);

1. On the ServerRPC, it simply will notify the id of the newly joined client

[ServerRpc(RequireOwnership = false)]

void sendDataToScoreUIServerRpc(ulong newlyJoinedClient)

{

sendDataToScoreUIClientRpc(newlyJoinedClient);

}

1. On the ClientRPC,which basically just update the Dictionary to be displayed in the gameUI

[ClientRpc]

void sendDataToScoreUIClientRpc(ulong clientID)

{

theScore[clientID] = theScoreForEachPlayer.Value;

GameObject.Find("gameUI").GetComponent<gameUI>().AddScoreUI(clientID);

Debug.Log("clientID : " + clientID + " while OwnerClientId = " + OwnerClientId + " and NetworkObjectId = "+ NetworkObjectId);

if(theScoreForEachPlayer.Value > 0)

{

GameObject.Find("gameUI").GetComponent<gameUI>().modifyScoreUI(NetworkObjectId, theScoreForEachPlayer.Value);

} else {

GameObject.Find("gameUI").GetComponent<gameUI>().modifyScoreUI(NetworkObjectId, 0);

}

}

1. On the gameUI object we will add modifyScoreUI method that would be called by the clientRPC

public void modifyScoreUI(ulong playerid, ushort updatedScore)

{

scoreGOs[playerid].GetComponent<Text>().text = playerid.ToString() + " has score of " + updatedScore;

}