Tutorial – Unity Networked Events

This expands on the **Unity Networked Animation** Tutorial to add even more networked feature.

Here we'll allow our animated characters to use switches in the level to make things happen on both client and server, such as opening and closing a door. The end result is very basic, but we'll set up a framework that can be re-used for many things.

Adding a single player switch system

We'll quickly put together a system for a single player to be able to use switches. It will have the following components:

Interactable – and object that can be interacted with, on every switch type object. It has UnityEvent that is Invoked when its clicked on.

User – a component on the player that allows them to click on Interactable with a raycast.

The Interactable script for now can just hold a UnityEvent and have a public function to Invoke it.

```
using UnityEngine;
using UnityEngine.Events;
using Mirror;

public class Interactable : MonoBehaviour
{
    public UnityEvent onUsed;

    public void Use()
    {
        onUsed.Invoke();
    }
}
```

The User script for the player just needs to do a raycast check when you click the mouse, and see if the object under the mouse is an Interactable. If so, call the event on it.

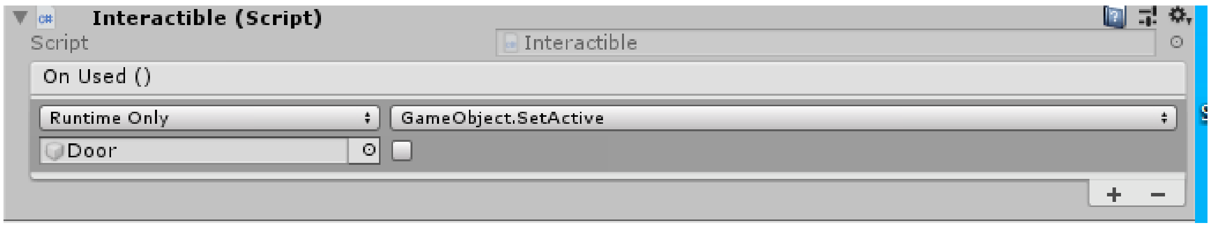
```
using UnityEngine;
using Mirror;

public class User : NetworkBehaviour
{
void Update()
    {
        // check for clicking on nearby objects
        if (Input.GetMouseButtonDown(0))
        {
            RaycastHit hit;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out hit))
            {
                // make sure we're within 2m
                if (Vector3.Distance(hit.point, transform.position) < 2)
                {
                    Interactible target =
                        hit.transform.GetComponent<Interactible>();
                    if (target)
                        target.Use();
                }
            }
        }
    }

...
```

Make a Capsule in your level, call it Switch, and put an Interactable on it.

Add a Cube and stretch it out, call it Door, and in the UnityEvent for the switch, set it up to make Door call GameObject.SetActive(false), so that clicking on the switch turns off the door.



Interactible (Script)

Script: Interactible

On Used ()

Runtime Only | GameObject.SetActive

Door | ☐

Put a User script on the Player prefab, and test quickly just as the Host. You'll see that walking up to the switch and clicking on it will make the door vanish.

If you run this in multiplayer though, the door only vanishes locally of course, for either host or client.

The Interactable itself can't help us out, because it won't have any authority on the client to give commands to the server. Fortunately, the User does, since it's part of the player prefab. So we'll make the User derive from Mirror.NetworkBehaviour and set it up to work across the network.

As before, the client issues a Command to the server, who then calls a ClientRpc function to execute it on every client.

A first attempt might look like this:

```
...
    [Command]
    public void CmdUse(Interactable target)
    {
        RpcUse(target);
    }

    [ClientRpc]
    public void RpcUse(Interactable target)
    {
        target.Use();
    }

}
```

This will not work.

We can't pass a reference through as a function argument.
Recall from lectures and the C++ tutorials that any data we pass across the network has to be encoded as Plain Old Data, ie floats, chars, ints and so on.

To pass an object through, we'd want a system where every object has an ID, and we can pass that ID through and the receiving programme can turn it back into a reference from that ID. **We don't need to write this ourselves**, because that's exactly what the NetworkIdentity component does.

This does allow us to pass GameObject refs through the network, **as long as they have a NetworkIdentity attached.**

(Each GameObject has a one-to-one correspondence with a NetworkIdentity, whereas Components do not, since a GameObject could have two components of the same type on it.)

(For more details on what you can pass through networked commands, go to https://mirror-networking.com/docs/Guides/DataTypes.html)

Fix up the above calls so they're passing a GameObject through, and get the Interactable using GetComponent so we can Use() it.

Note that Interactable is not a NetworkBehaviour, but does require a NetworkIdentity to work properly. It might be a good idea to give it a RequiresComponent tag above the class.

Consolidating this system

This provides a powerful framework for any event which is instigated by the player but needs to propagate over the network. You may want to investigate how you can set this up further to provide UnityEvent based activity in a networked game.

Put together a PlayerTrigger class that fires off a UnityEvent when a User enters or exits the trigger, and figure out how to get it network-enabled in a similar manner. Look up the [Server] tag in the Mirror documentation to help you with this.