Tutorial – Unity Networked Animation

This expands on the Unity Networking Tutorial to add more advanced features that you'd have in a networked game.

After finishing that tutorial you should have a multiplayer game where the capsule players can move around and shoot each other with LineRenderer beams.

Here we'll upgrade that to a game with animated character who can perform actions like a basic beam or projectile attack, and use a simple object interaction system across the network.

Adding an animated character

You'll need an animated character controller of some kind. You should have one from your Physics or Computer Graphics module. If you don't, complete the Animation tutorial in Computer Graphics to get a basic animated character.

Find your previous project and export the character prefab and associated scripts and models as a UnityPackage, so you can drop it into your Network project.

You'll then need to spend a bit of time making your character movement networkable once it's in your network project.

- Add a **NetworkTransform** to your character prefab, and tick **Client Authority**. This will also create a **NetworkIdentity** on the prefab
- Add the **MarkerMoverNetwork** and **Health** scripts from the previous tutorial. We'll come back and do the **LaserBeam** one next.
- Drag your new prefab into the **Player Prefab** member of your **NetworkManage**r in the scene
- In your character movement script, derive it from Mirror.NetworkBehaviour instead of MonoBehaviour
- In your character movement code, derive the script from NetworkBehaviour instead of MonoBehaviour. Find where you've been reading keyboard input and make sure that only happens on the local player, so put in a check or early return using the *isLocalPlayer* member like we did before.

If you've done all this, you should now see your animated character running around in the level. Have a second build join as client and you'll see both characters. However, the non-local character will probably not be animating, but just sliding around in an idle pose.

Synching animations

Our animations are driven by a number of variables in the Animator, which get set according to variables in our character controller script. You'll probably have some code like this:

```
        animator.SetBool("Jump", !m_grounded);
        animator.SetFloat("Forwards", m_movement.magnitude);
        animator.SetBool("Crouch", Input.GetKey(KeyCode.C));
```

We could try to synch this information in a number of ways:

1. Calculate these values based on how much a non-local character is moving, or whether they're grounded and so on.
2. Turn some of the variables driving the animation, such as m_grounded, m_movement, into SyncVars
3. Synchronise the variables in the animator across the network

Each method has pros and cons. Method 1 would require extra code when we add new features to the animation, for example figuring out when exactly the non-local player has crouched.

Method 2 deals with crouching, but requires maintenance.

Method 3 will automatically work for any new variables we add to the Animator, and fortunately there's a component in Mirror that does just this.

Add a **NetworkAnimator** to your prefab, give it **Client Authority**, and drag its Animator into the Animator field in the NetworkAnimator.

Try playing the game now. Both characters should animate remotely across the network as they move around.

Adding an animation-driven LaserBeam

The LaserBeam script previously worked on a keypress, which turned on a LineRenderer from the character's "head" and activated some networked code to check a raycast on the server and subtract health from characters (synched as a SyncVar).

To restore that, copy the Beam, Muzzleflash and Blood child objects from the Player prefab into your animated character. Consider placing the components on children in the animation hierarchy, like the blood particles on the chest and the laser and muzzle flash on the head.

Line them up, restore any scale issues that arise from them being children of a distorted cube, and it should be good.

Add a LaserBeam script to the character and hook up the Line Render and FireFX to reference the FX objects in the prefab, and the HitFx in its Health to the Blood particle system.

If your character has multiple colliders you may need to modify the LaserBeam code to register hits on them. If you hit the right leg collider with the raycast, then it doesn't have a Health script on it, but its parent further up the hierarchy does.

In Unity 2019, they introduced a very handy GetComponentInParent<type>() function, so use that to find the Health. You should then be able to shoot the other robot and see blood and hit point loss again across the network.

Let's extend this now by adding animation. We want an animation with a Hit event (covered in the Computer Graphics tutorials)

To do this in a single player game we'd go from:

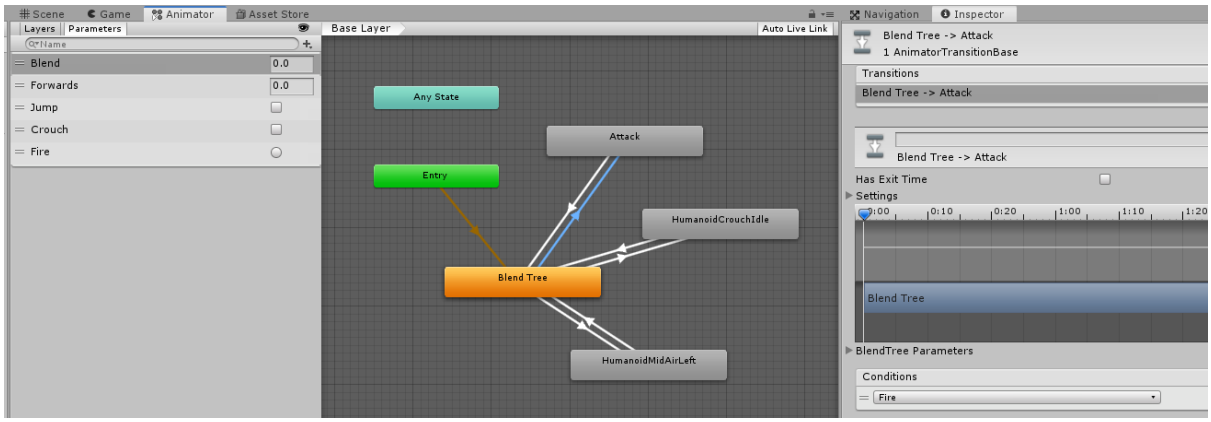
- Pressing Key starts Laserbeam function, which does visual FX and damage

To

- Pressing Key starts animation
- Animation event triggers the Laserbeam function, which does visual FX and damage

We need to find an animation (use the punch in the Computer Graphics Animation tutorials if you like) and:

- Set it to Humanoid rig
- Add a Hit event at the right place in the animation
- Add a Fire trigger in our Animator graph
- Add the animation in as a node and make Transitions to (with Has exit Time unticked and Fire trigger required) and from (Has Exit Time ticked) this node.



We've already split the visual effects (applied on the local machine) and the gameplay effects (raycast applied only on server) so we should be good to make this change for a multiplayer networked game.

The question is how do we start the animation remotely? Does the NetworkAnimation component handle that for us?

It doesn't transmit animation triggers. It is not designed to do so, due to their instantaneous nature.

We need to call the animation trigger using a [Command] and [ClientRpc] pair of fucntions like we did before for firing. The animation event response will be called on both machines locally by the animation. Your code should look like this:

```
    [Command]
    void CmdFire()
    {
        // tell all clients to do it too
        RpcFire();
    }

    [ClientRpc]
    void RpcFire()
    {
        animator.SetTrigger("Fire");
    }

    public void Hit()
    {
        // this gets called in response to animation events
        DoLaser();
    }
```