**Walkthrough 8: Unity NetworkManager**

In this walkthrough we will be adding networking to a new version of the Rogue-Like Game.

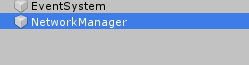
**Part 1: Network Manager & HUD**

We will need to create a network Manager object.

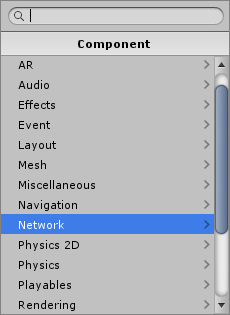
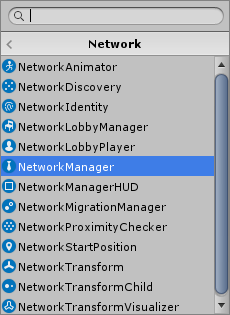
A **Network Manager** is a built in component in Unity to manage Connecting Clients to a Server.

Steps:

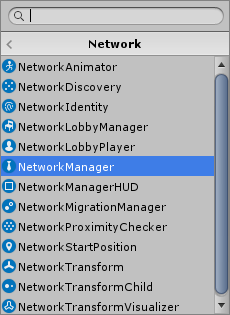
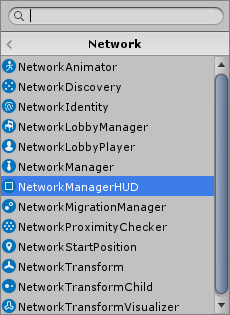
1. Download the Walkthrough 8 Project from blackboard.
2. Open the main Scene “\_Complete-Game.Unity” located in the main \_Complete-Game folder.
3. Create an empty GameObject in the scene and name it NetworkManager



1. Add a new component to the NetworkManager GameObject through the UI by selecting the Add Component button then navigating to Network > NetworkManager

1. Expand the properties of your new component on the Object and Modify the following:
   1. Network Port: 8777
2. Add in a NetworkManagerHUD component using the same add component buttons above:

This will add a brand new HUD with selectable buttons to our game so we can choose whether we are running as a Client or Host.

The NetworkManagerHUD relies on a NetworkManager component already added to the gameobject.

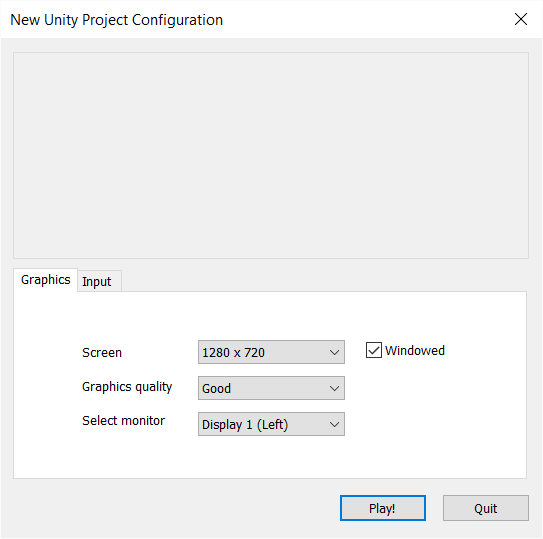
The NetworkManagerHUD will automatically handle the creation of a host setup and allow clients to connect.

1. Leave all the settings as is for this.
2. Run the Game and ensure you see the HUD display in the top left corner like so:



This interface will allow us to choose whether we are server or client and set the Network Manager settings up appropriately based off the choice.

1. The first button creates a server and sets the current user as the server client.
2. All subsequent users should select the next button to login to the server while it is running.
3. Select Build and Run.

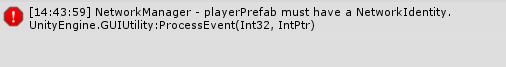


Make sure to **set the resolution o 1280x720 and check windowed.**

1. Hit play.
2. Run in the editor as well
3. Click the server button on one instance in the top left and the client button on the other. (which doesn’t matter)
4. You’ll notice you don’t see anything, BUT WE ARE CONNECTED

**Part 2: Adding Players ( Network Identity)**

Notice the errors printed in Unity when we attempt to run:



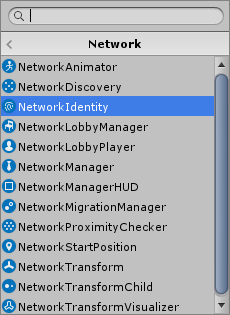
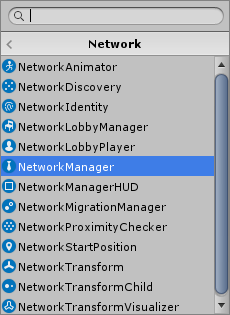
The **NetworkManager** Expects a prefab allocated for the Player so when a new Player joins the server it creates a new instance of it.

Ideally we would add our Player prefab by selecting the **NetworkManager** and dragging from the project view, but if you try this is will reject your prefab.

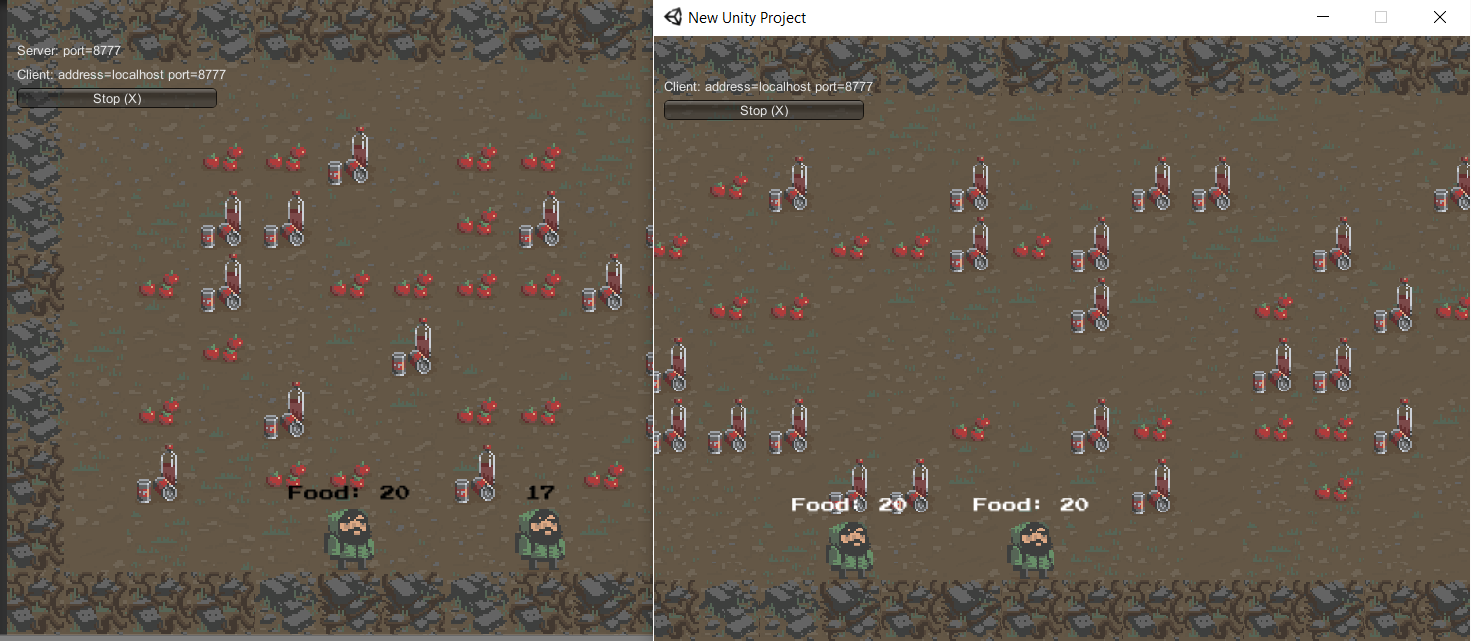
To fix this we need to modify our player prefab to contain a **NetworkIdentity**.

The **NetworkIdentity**  allows each object to have it’s own unique ID for tracking who owns which object in the game.

1. Open up the prefab for the player and add the following component:



1. We should now be able to apply our player Prefab to the networkManager’s Playerprefab slot, so go ahead and do that now.
2. Run the game and select host. You should now see a playerPrefabs loaded.
3. Move your character on the first instance so it is offset from the starting location and then run an external game by building and running from the file menu.

Notice that there are now two players added to the scene, but it’s not quite right yet.

**Part 3: Server Commands (Player Movement 1)**

Both players are controlling the same character, we are not changing our functionality based off the Network ID.

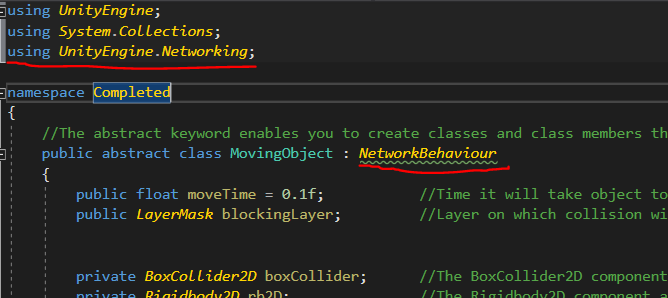
Notice in our Unity Editor instance that we have two Player(Clone) entries.



When a new user logs into the game Unity creates a new clone for that player.

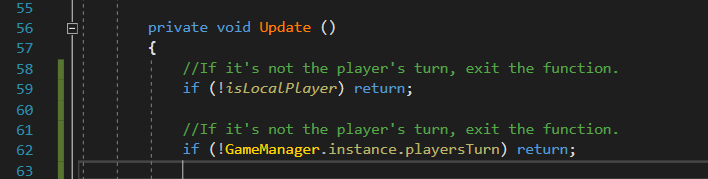
The Processing only effects the first object added.

1. Open up your Player.cs script and notice it is based off the MoveableObject Script.
   1. This is used for both the player and the zombies.
2. We will need to modify the MoveableObject script to inherit from NetworkBehaviour instead of MonoBehaviour:



**NetworkBehaviour** itself is just a beefed up MonoBehaviour object. It **inherits** **from** **MonoBehaviour** and adds on **extra functionality and properties used for Networking**.

1. Add an early return to our player.cs file:



Here we are checking a variable “**isLocalPlayer**”. This variable is given to us in the player object thanks to the **NetworkBehaviour** we previously changed our inheritance from.

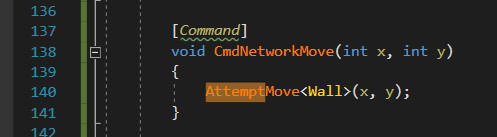
Previously our update function went on to map out the horizontal and vertical controls to the first object.

When further objects were added, they couldn’t apply the horizontal and vertical to both objects so only the first player character was updated for both.

With this early return added, we no longer update the players the client does not own.

We have to update the Server from the client of all of our movement.

1. **Add using UnityEngine.Networking;** to the top of your player.cs file.
2. Add the following function to your player.cs file directly above the AttemptMove function:

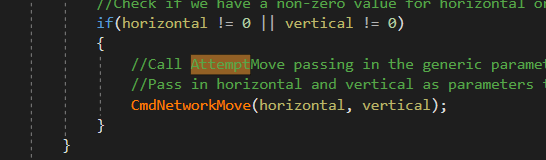


Note the **[Command] Tag** above the new function.

This again is a part of the **UnityEngine.Networking** components. Essentially this tag **marks a function as a network function**.   
Network functions can be called from any client, however, the function itself **is only processed on the server.**

Commands in Unity **require** **Cmd** **as** **the first three characters** **of the functions Identifier** (Name)

1. Replace the call above to AttemptMove at the bottom of the update function with your new Command.



1. Run the game in both the editor and the external build and observe moving both characters.
   1. (Ensure you move the server character a bit first before the second clients connects)

Notice that the server now updates, and the client who sent the Command to move does not.

**Part 4: Player Spawn Position**

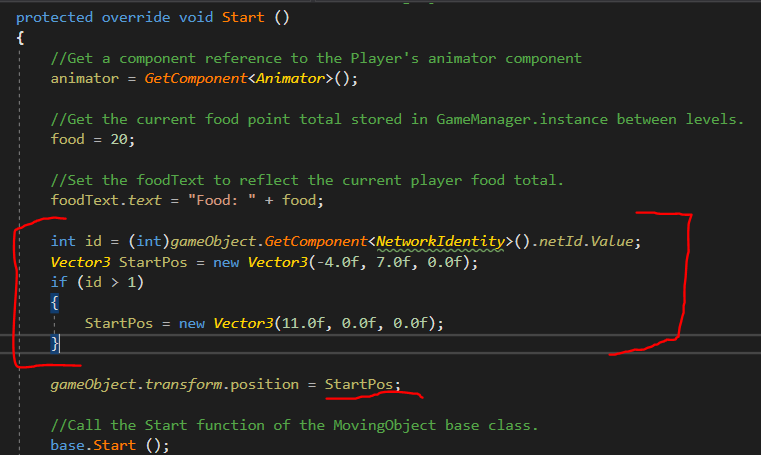
Presently we are modifying the position before we spawn in a second character, this is not ideal.

We can take advantage of the **NetworkIdentity** attached to each player to alter their positional data.

**NetworkIdentity** assigns a **unique incremental ID** to each new client that joins.

We can fix the issue of the users spawning over top of each other in the same location by changing the players starting location based off which **NetworkIdentity** it has.

1. Inside of our Player.cs file we can get the NetworkIdentity component and check what netID was assigned to it and modify the starting position based on this

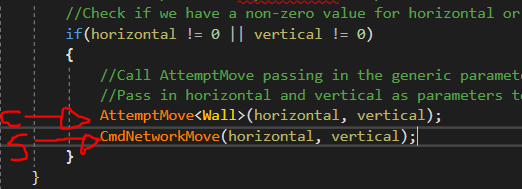


**Part 5: Player Movement Replication (RPC’s)**

Presently the client is telling the server about its actions, but only the server then executes them.

In order to resolve this issue, we need to do both. We need to process it locally on the client and update the call on the server.

1. We can achieve this quickly by calling both the local call in addition to the server Command:



1. Run both games and ensure that both the server and the client update the position of the characters.

Notice that when the server moves the client is not notified.

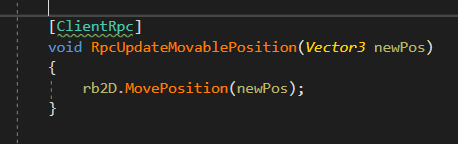
We need to ensure that the server also replicates it’s changes down to all the clients

To do this we can use what’s called a **ClientRPC**. This is a function that only the server can trigger which then calls that function on each and every client. The Function itself is not run on the server so it does not affect the servers state at all, it strictly calls the function to run on all the clients.

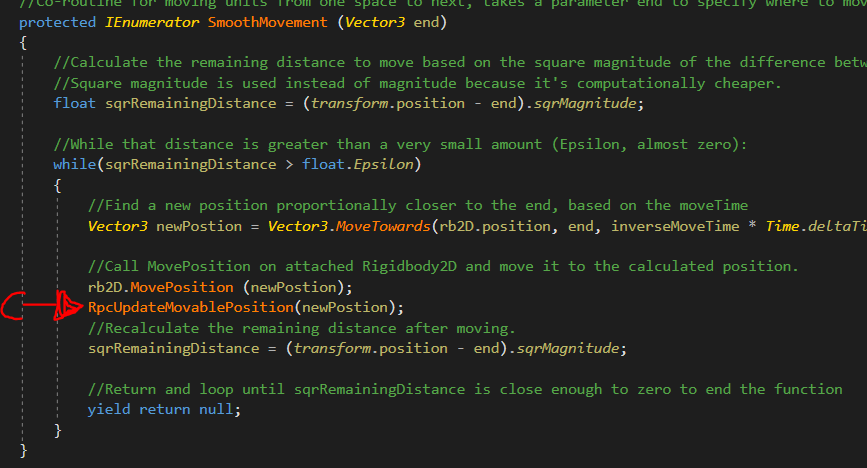
Similar to our [Command] tag we added for a function the client can call on the server, the **[ClientRpc]** tag can be added above a function to be called on the clients.

Again, similar to [**Command]** needing **Cmd**, [**ClientRpc]** needs the first three letters of the identifier to be **Rpc**

1. Inside of our MovingObject.cs script we can create a new ClientRPC to update the position of the character:



1. We can then call the Client RPC by invoking the function when a movement occurs:



RUN THE GAME, YOU SHOULD BE ABLE TO HAVE BOTH THE CLIENT AND SERVER CONNECT AND MOVE AROUND THE SCREEN AND BOTH SHOULD BE UPDATED OF EACH OTHERS CHARACTER POSITIONS.

\*\*\* SIGN OFF WITH THE INSTRUCTOR\*\*\*