Multiplayer Games in Unity

Unity Networking Basics

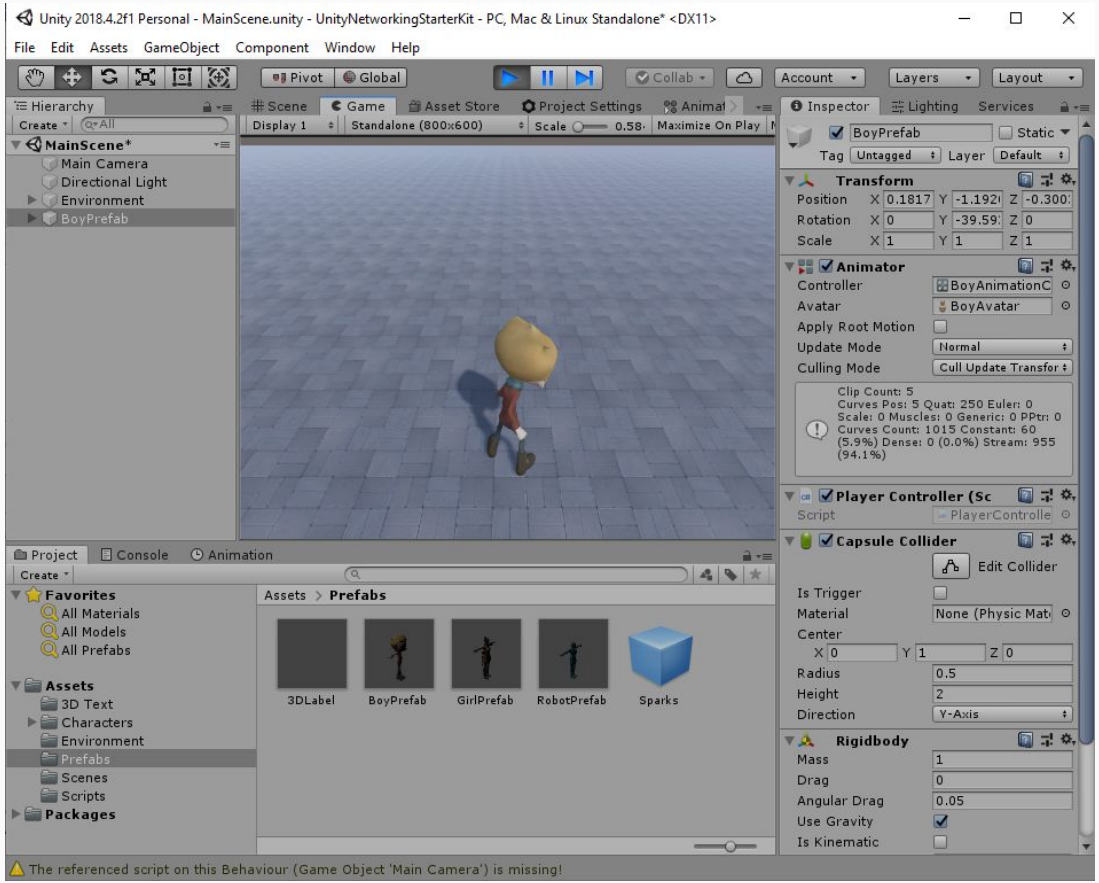
Networks and Online Games

Main components and tools

- NetworkManager
- NetworkManagerHUD
- NetworkIdentity
- NetworkStart
- NetworkTransform
- NetworkStartPosition
- ...

- SyncVars & Hooks
- Commands
- Custom NetworkManager
- Client RPCs

Download the starter kit



The NetworkManager

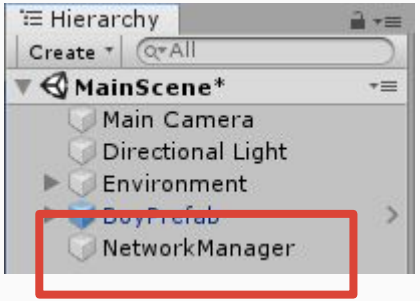
NetworkManager

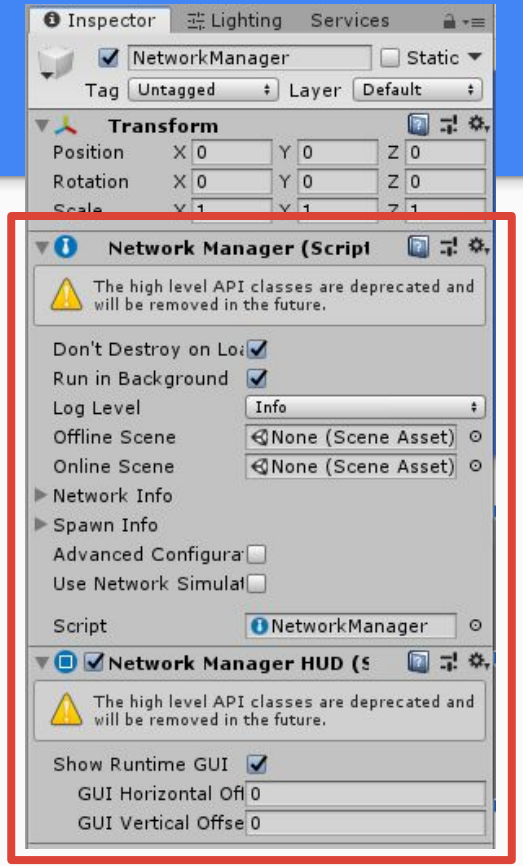
A networked application needs a **NetworkManager**

- Like our ModuleNetworking
- Functionality client / server

Steps

- New GameObject
- NetworkManager component
- NetworkManagerHUD component

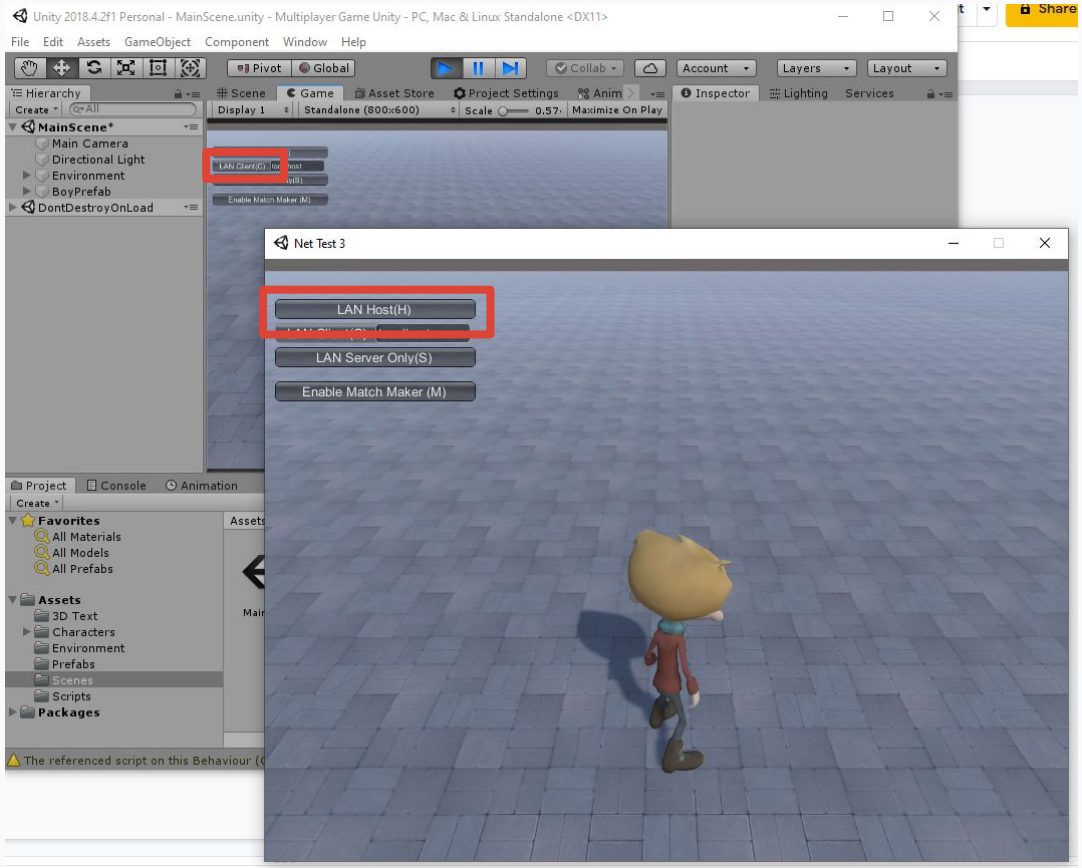




NetworkManager

Test the application

- Execute two instances
  - One creating a build: Build Settings / Build and Run
  - Another using the Play button
- Start NetworkingManager
  - One as **Host**
  - Another as **Client**
    - Connect to localhost
- Nothing happens…
  - Only the local player is visible



Adding players to the network

Adding players to the network

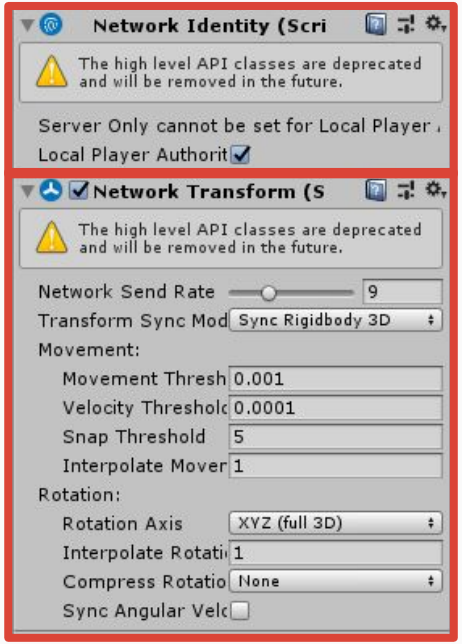
2 steps:

- Player prefab needs a network ID
- NetworkManager needs to know networked prefabs

Adding players to the network

Step 1:

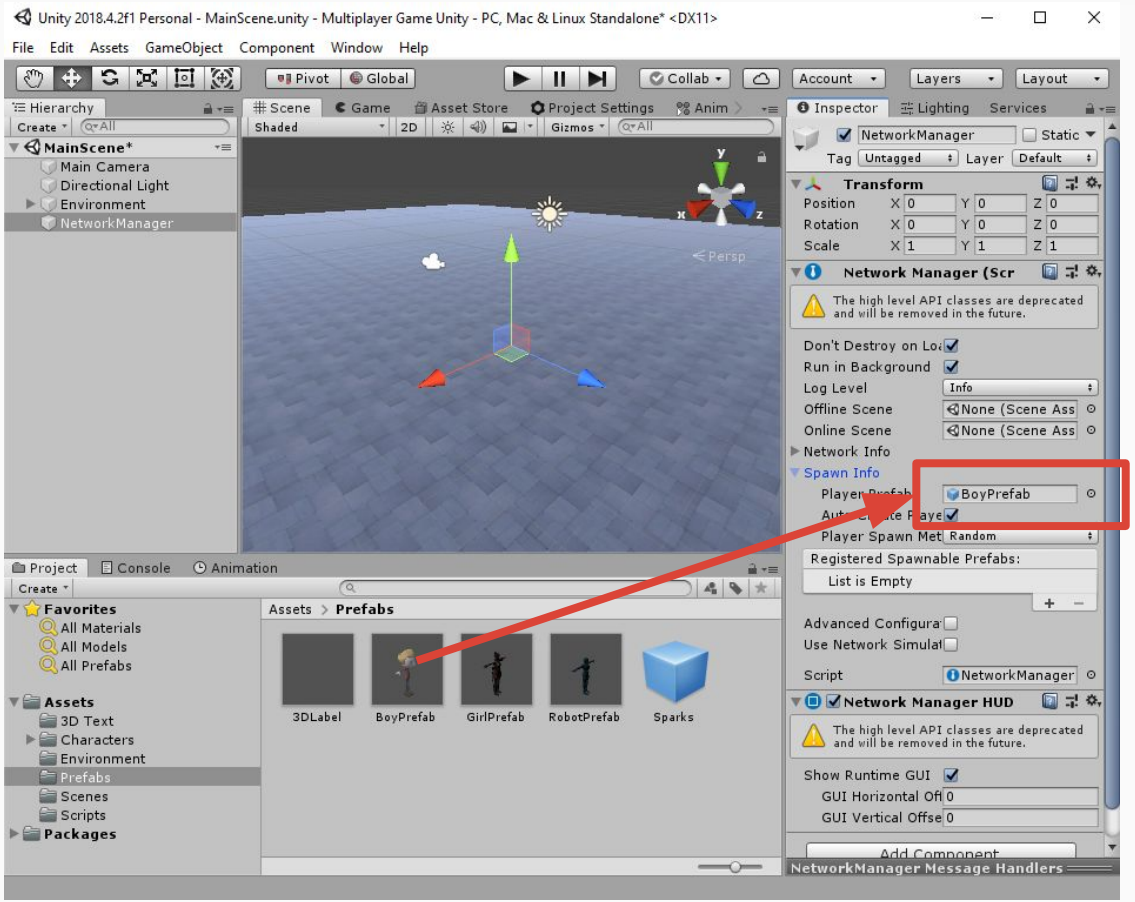
- Player prefab needs a network ID
  - **Remove the player instance from the hierarchy**
    - The server will spawn the player on connection
  - **Open the player prefab from the project window**
  - Add NetworkIdentity component
    - Set local player authority
  - Add NetworkTransform component



Adding players to the network

Step 2:

- NetworkManager needs to know the networked prefabs
  - Register the prefab into the NetworkManager
  - Drag it to the PlayerPrefab field in the NetworkManager



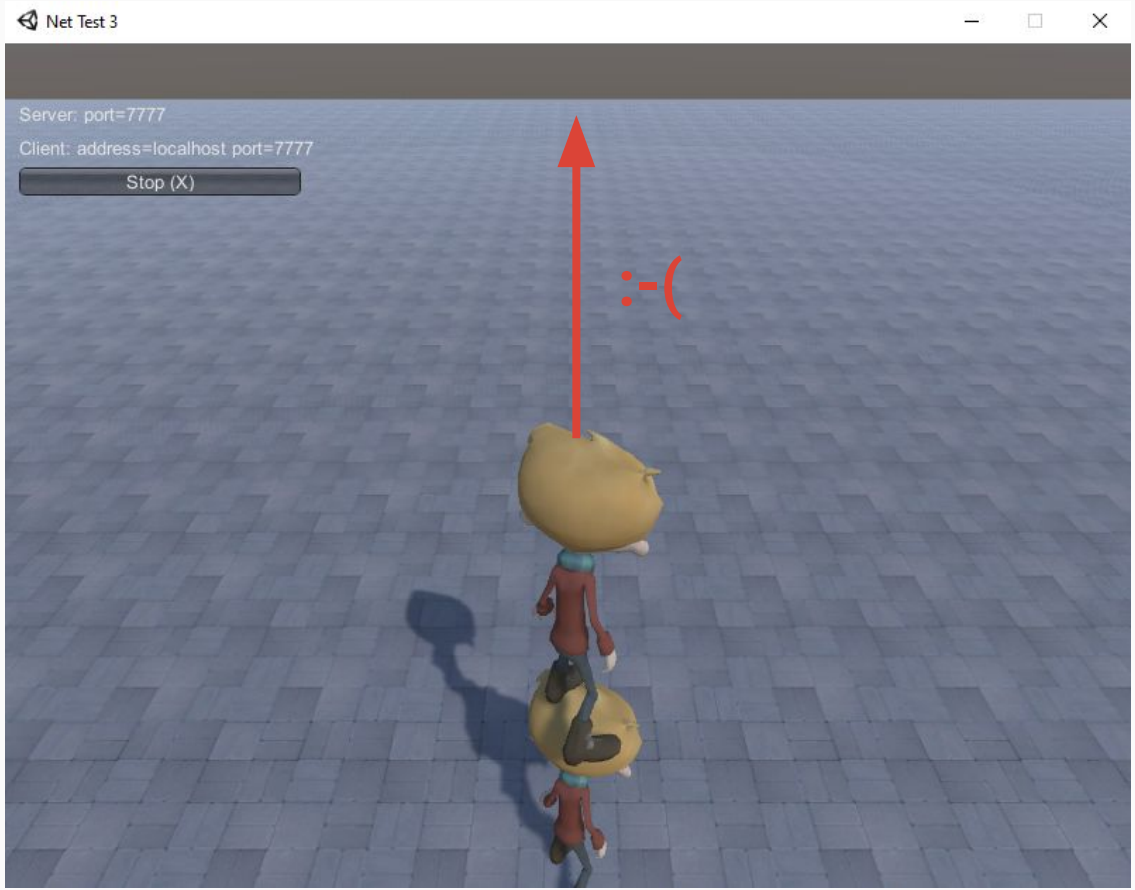
Adding players to the network

Cool!!!

- 2 players connected!!!

Not so cool...

- They appear on the same position
  - RigidBody + Colliders
  - Physics simulation
  - Rocket effect :-(
- All connected players are controlled locally
  - Script is attached to all of them

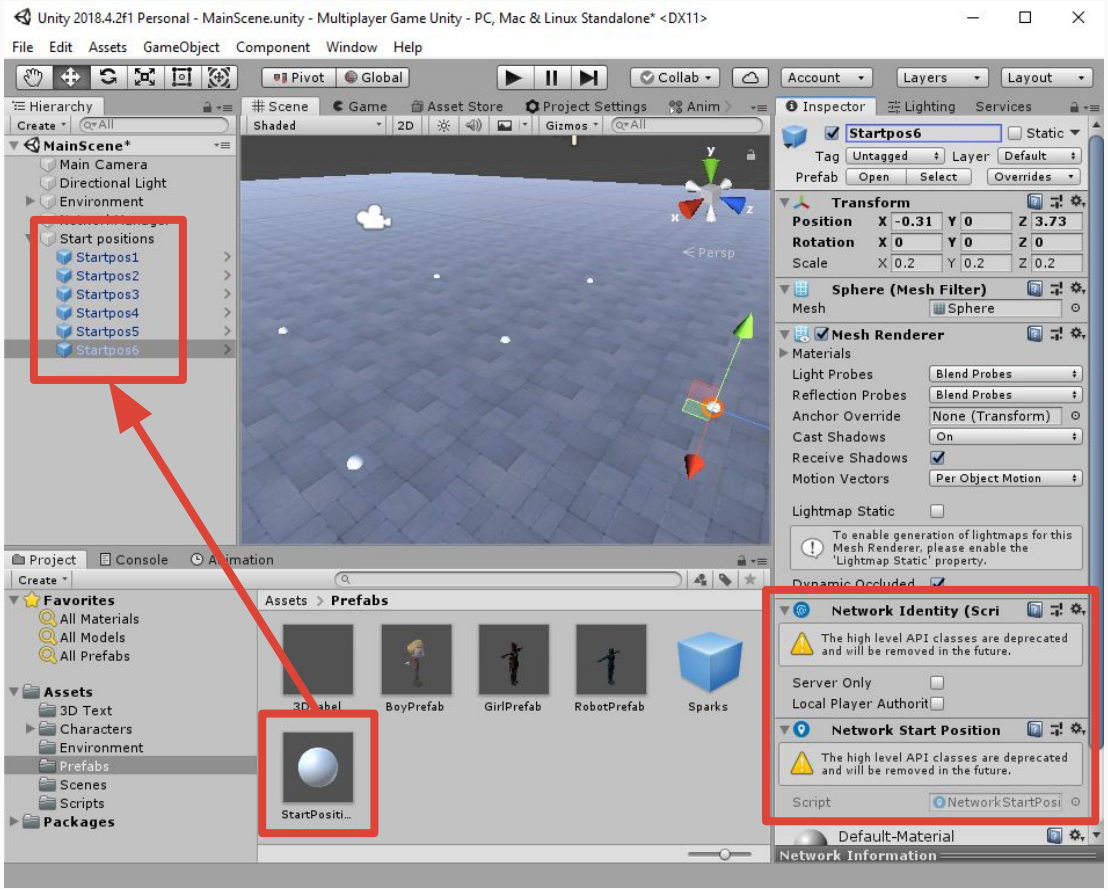


Initial player positions

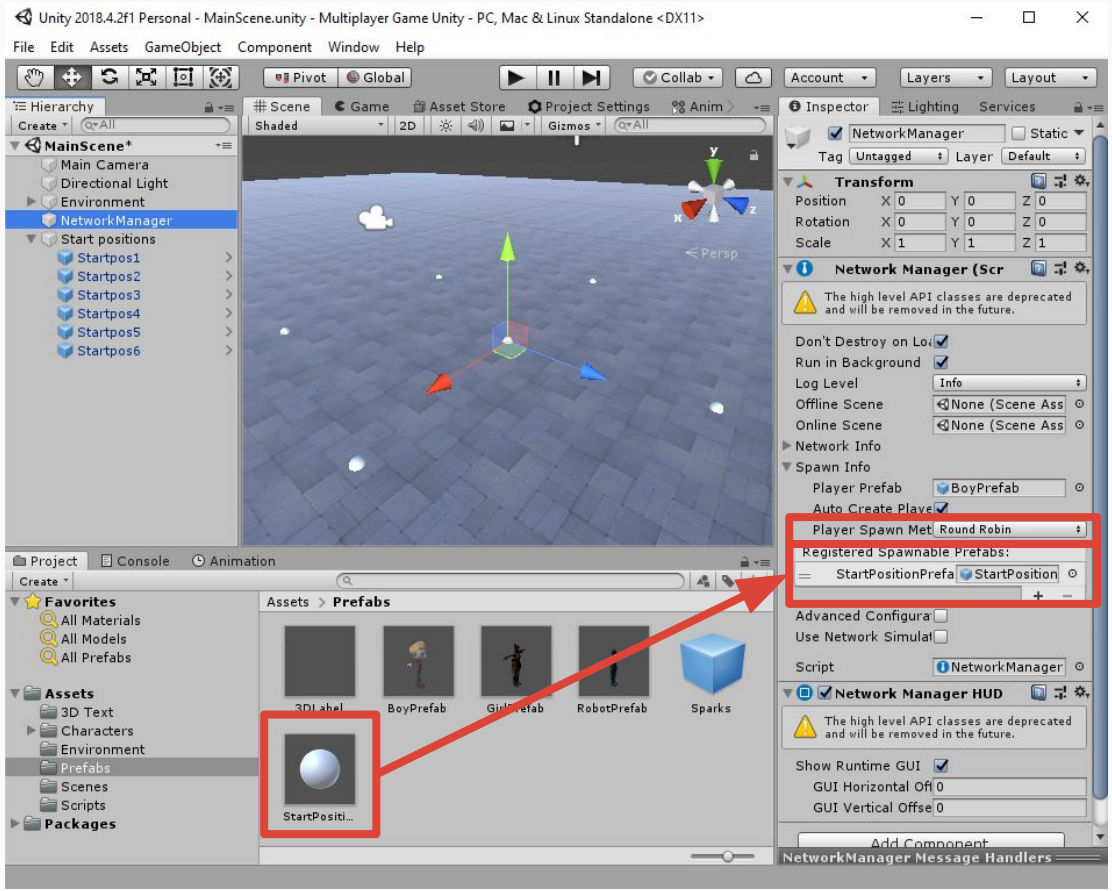
Initial player positions

- Create a new GameObject with a sphere
  - Add these components
    - **NetworkIdentity**
    - **NetworkStartPosition**
  - The sphere is for debugging (can be removed later)
- Make it a Prefab and remove from hierarchy
- In the NetworkManager
  - Add the prefab into the list of **Spawnable Prefabs**
  - Select **RoundRobin** as the Player Spawn Method
- Instantiate some Starting Positions into the scene

Initial player positions



Initial player positions



Networked scripts: isLocalPlayer

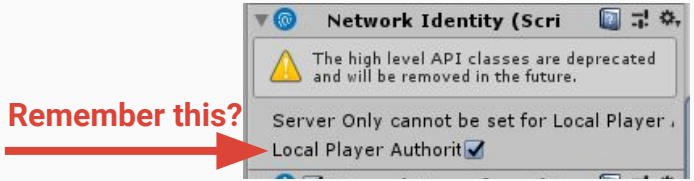
Networked scripts

We want to control/modify only our player

- Local authority

Changes in **PlayerController** script

- Import **UnityEngine.Networking**
- Inherit from **NetworkBehaviour**
- Use **isLocalPlayer** attribute




```
using UnityEngine.Networking;

public class PlayerController : NetworkBehaviour
{
    // Update is called once per frame
    void Update ()
    {
        if (isLocalPlayer)
        {
            Vector3 translation = new Vector3();
            float angle = 0.0f;

            float horizontalAxis = Input.GetAxis("Horizontal");
```

Networked scripts: Commands, SyncVars and Hooks

Commands, SyncVars, and Hooks

Command

- Function invoked by a client, but executed remotely in the server

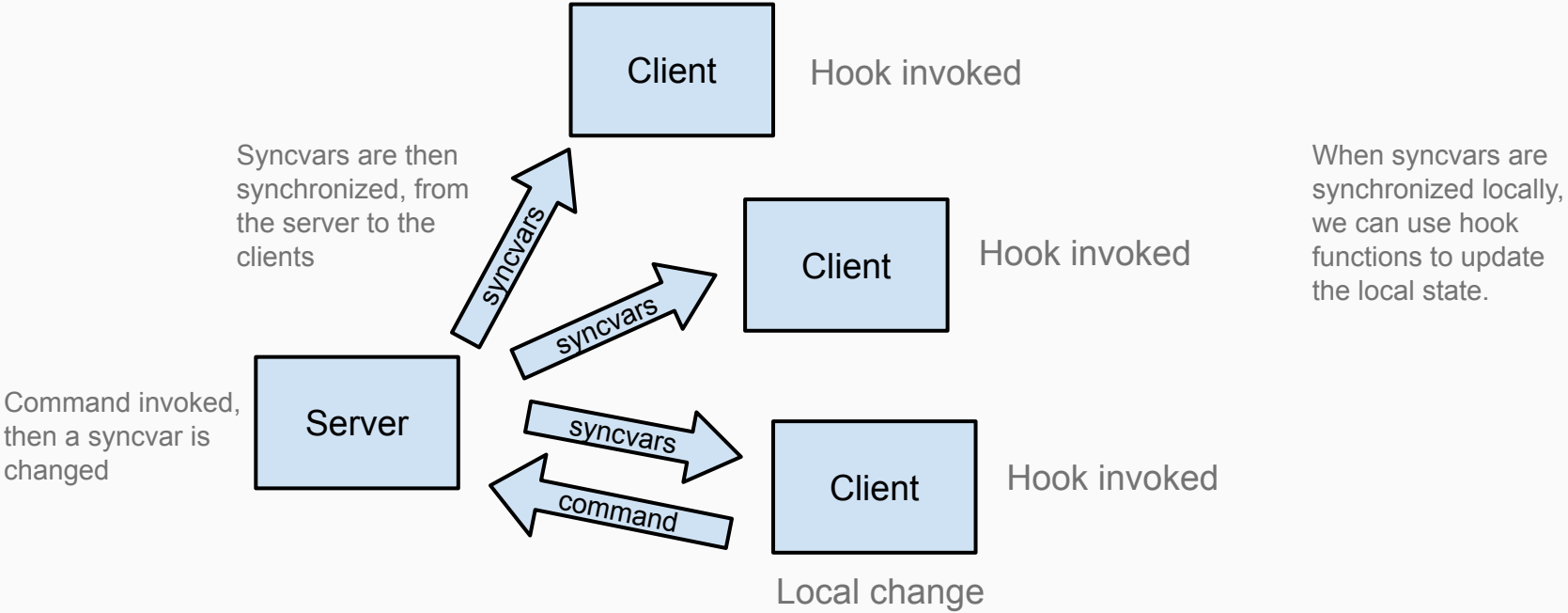
SyncVar

- Variable changed in the server, and replicated to all clients

Hook

- Client callback executed when a SyncVar changes in a client

Commands, SyncVars, and Hooks



Commands, SyncVars, and Hooks example: Changing the player name

Commands, SyncVars, and Hooks example: Changing the player name

```
// Name sync ////////////////////////////////////
[SyncVar(hook = "SyncNameChanged")]
public string playerName = "Player";

[Command]
void CmdChangeName(string name) { playerName = name; }

void SyncNameChanged(string name) { nameLabel.text = name; }


// OnGUI ////////////////////////////////////////

private void OnGUI()
{
    if (isLocalPlayer)
    {
        GUILayout.BeginArea(new Rect(Screen.width - 260, 10, 250, Screen.height - 20));

        string prevPlayerName = playerName;
        playerName = GUILayout.TextField(playerName);
        if (playerName != prevPlayerName)
        {
            CmdChangeName(playerName);
        }

        GUILayout.EndArea();
    }
}
```

Commands, SyncVars, and Hooks example: Changing the player name

```
// Name sync ////////////////////////////////////
[SyncVar(hook = "SyncNameChanged")]
public string playerName = "Player";

[Command]
void CmdChangeName(string name) { playerName = name; }

void SyncNameChanged(string name) { nameLabel.text = name; }


// OnGUI ////////////////////////////////////////

private void OnGUI()
{
    if (isLocalPlayer)
    {
        GUILayout.BeginArea(new Rect(Screen.width - 260, 10, 250, Screen.height - 20));

        string prevPlayerName = playerName;
        playerName = GUILayout.TextField(playerName);
        if (playerName != prevPlayerName)
        {
            CmdChangeName(playerName);
        }

        GUILayout.EndArea();
    }
}
```

1) A command is invoked by a client

Commands, SyncVars, and Hooks example: Changing the player name

```
// Name sync ////////////////////////////////////
[SyncVar(hook = "SyncNameChanged")]
public string playerName = "Player";

[Command]
void CmdChangeName(string name) { playerName = name; }

void SyncNameChanged(string name) { nameLabel.text = name; }


// OnGUI ////////////////////////////////////////

private void OnGUI()
{
    if (isLocalPlayer)
    {
        GUILayout.BeginArea(new Rect(Screen.width - 260, 10, 250, Screen.height - 20));

        string prevPlayerName = playerName;
        playerName = GUILayout.TextField(playerName);
        if (playerName != prevPlayerName)
        {
            CmdChangeName(playerName);
        }

        GUILayout.EndArea();
    }
}
```

2) The command is executed in the server

Commands must have the ***Cmd*** prefix!!!

Commands, SyncVars, and Hooks example: Changing the player name

```
// Name sync ////////////////////////////////////////
[SyncVar(hook = "SyncNameChanged")]
public string playerName = "Player";

[Command]
void CmdChangeName(string name) { playerName = name; }

void SyncNameChanged(string name) { nameLabel.text = name; }


// OnGUI ////////////////////////////////////////

private void OnGUI()
{
    if (isLocalPlayer)
    {
        GUILayout.BeginArea(new Rect(Screen.width - 260, 10, 250, Screen.height - 20));

        string prevPlayerName = playerName;
        playerName = GUILayout.TextField(playerName);
        if (playerName != prevPlayerName)
        {
            CmdChangeName(playerName);
        }

        GUILayout.EndArea();
    }
}
```

See? playerName is a SyncVar

3) The SyncVar playerName is changed

Commands, SyncVars, and Hooks example: Changing the player name

```
// Name sync ////////////////////////////////////
[SyncVar(hook = "SyncNameChanged")]
public string playerName = "Player";

[Command]
void CmdChangeName(string name) { playerName = name; }

void SyncNameChanged(string name) { nameLabel.text = name; }


// OnGUI ////////////////////////////////////////

private void OnGUI()
{
    if (isLocalPlayer)
    {
        GUILayout.BeginArea(new Rect(Screen.width - 260, 10, 250, Screen.height - 20));

        string prevPlayerName = playerName;
        playerName = GUILayout.TextField(playerName);
        if (playerName != prevPlayerName)
        {
            CmdChangeName(playerName);
        }

        GUILayout.EndArea();
    }
}
```

See? playerName is hooked to SyncNameChanged

4) As playerName is changed on clients, the hook SyncNameChanged is executed

Commands, SyncVars, and Hooks example: Syncing animations

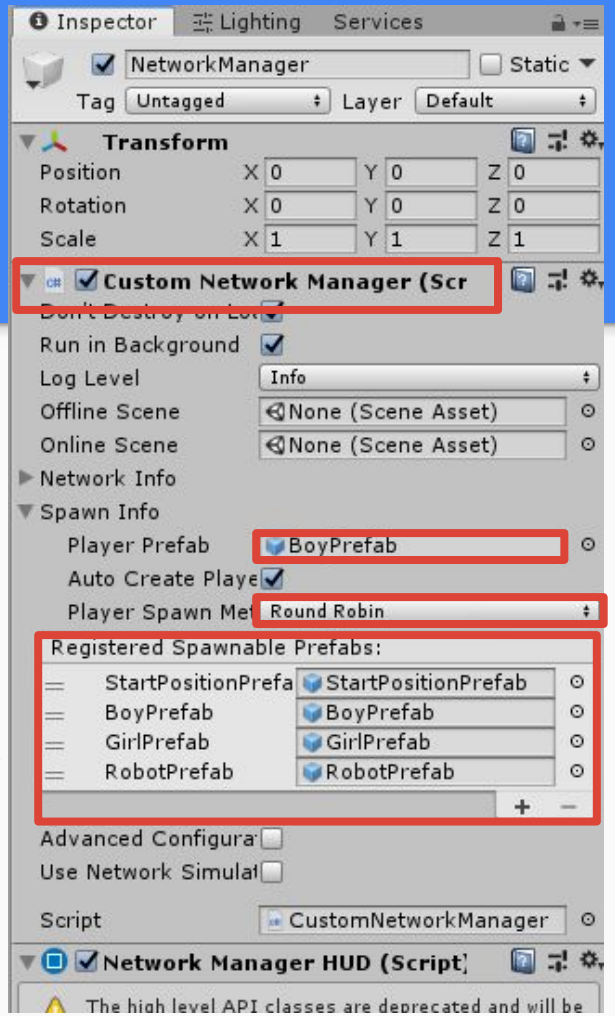
```
// Animation sync ////////////////////////////////

[SyncVar(hook = "OnSetAnimation")]
string animationName;

void setAnimation(string animName)
{
    OnSetAnimation(animName);
    CmdSetAnimation(animName);
}

[Command]
void CmdSetAnimation(string animName)
{
    animationName = animName;
}

void OnSetAnimation(string animName)
{
    if (animationName == animName) return;
    animationName = animName;

    animator.SetBool("Idling", false);
    animator.SetBool("Running", false);
    animator.SetBool("Running backwards", false);
    animator.ResetTrigger("Jumping");
    animator.ResetTrigger("Kicking");

    if (animationName == "Idling") animator.SetBool("Idling", true);
    else if (animationName == "Running") animator.SetBool("Running", true);
    else if (animationName == "Running backwards") animator.SetBool("Running backwards", true);
    else if (animationName == "Jumping") animator.SetTrigger("Jumping");
    else if (animationName == "Kicking") animator.SetTrigger("Kicking");
}
```

CustomNetworkManager: Select the player prefab before connecting

Custom NetworkManager

To change the behaviour of the NetworkManager:

- Remove prev. NetworkManager component
- Create a new script **CustomNetworkManager**
  - Import UnityEngine.Networking
  - Inherit from NetworkManager
- Drag **it** to the NetworkManager game object
  - It contains the same fields as before
  - Reconfigure them (player prefabs, start point prefab)
  - **Test everything is still ok...**



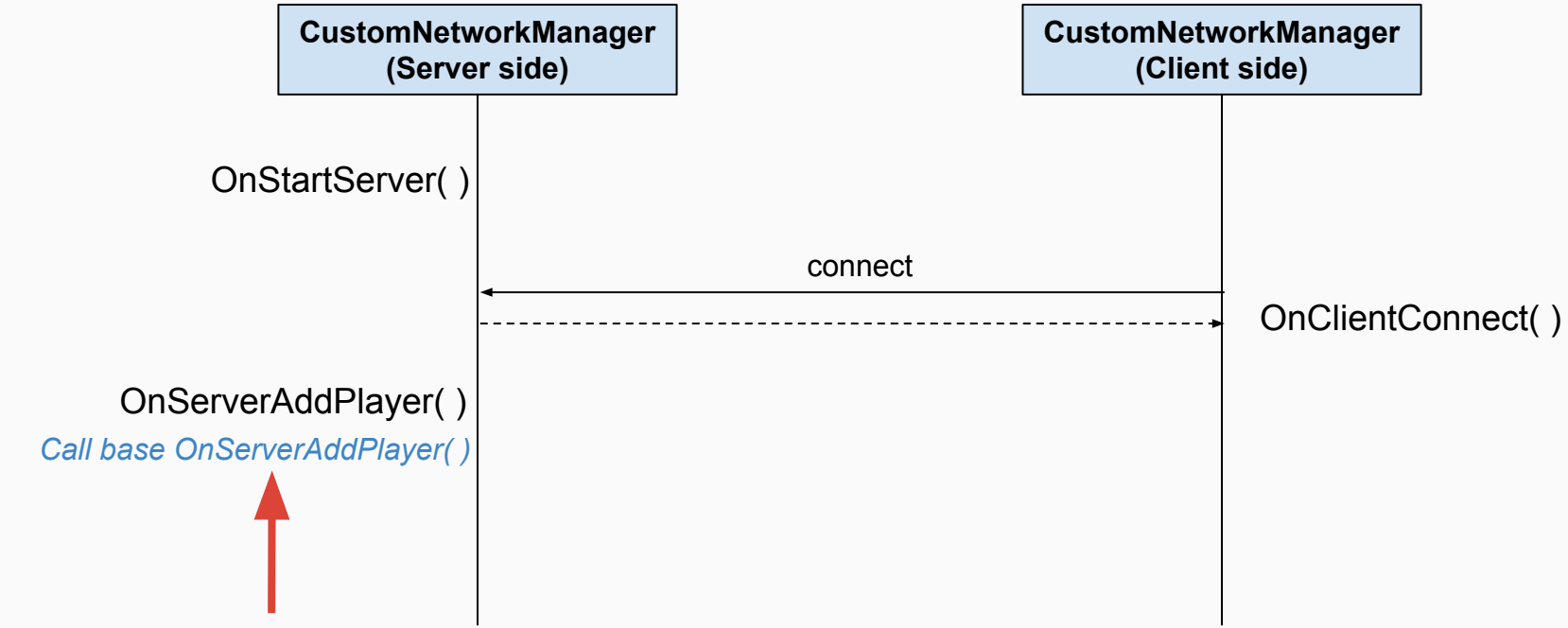
Custom NetworkManager - Selecting the player before connecting

```
using UnityEngine.Networking;

public class CustomNetworkManager : NetworkManager
{
    // 1) Executed in the server
    public override void OnStartServer()
    {
        base.OnStartServer();
    }

    // 2) Executed in the client
    public override void OnClientConnect(NetworkConnection conn)
    {
        base.OnClientConnect(conn);
    }

    // 3) Executed in the server
    public override void OnServerAddPlayer(NetworkConnection conn, short playerControllerId)
    {
        base.OnServerAddPlayer(conn, playerControllerId);
    }
}
```

Calling the base class implementations (as in this example) results in the default behaviour.

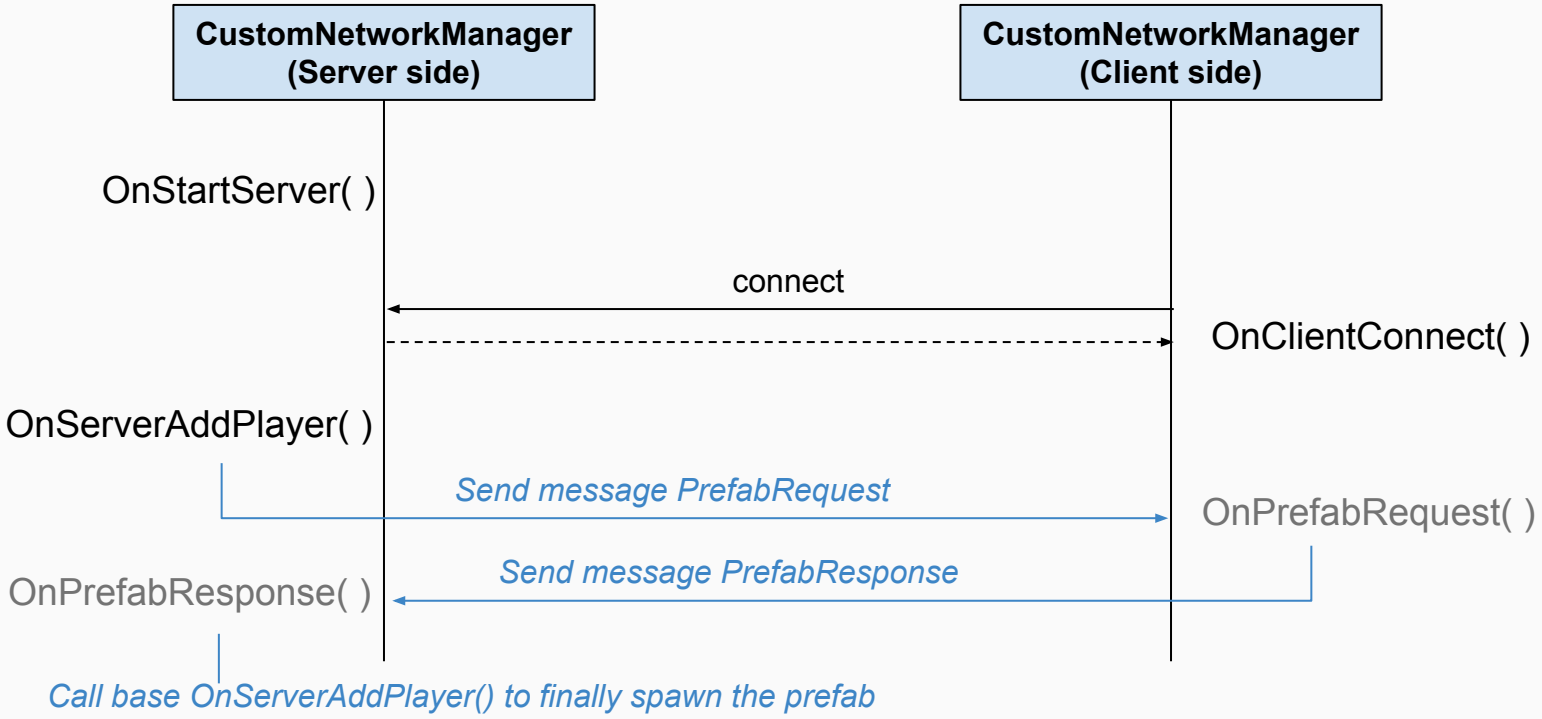
This spawns the default player prefab

Custom NetworkManager - Selecting the player before connecting

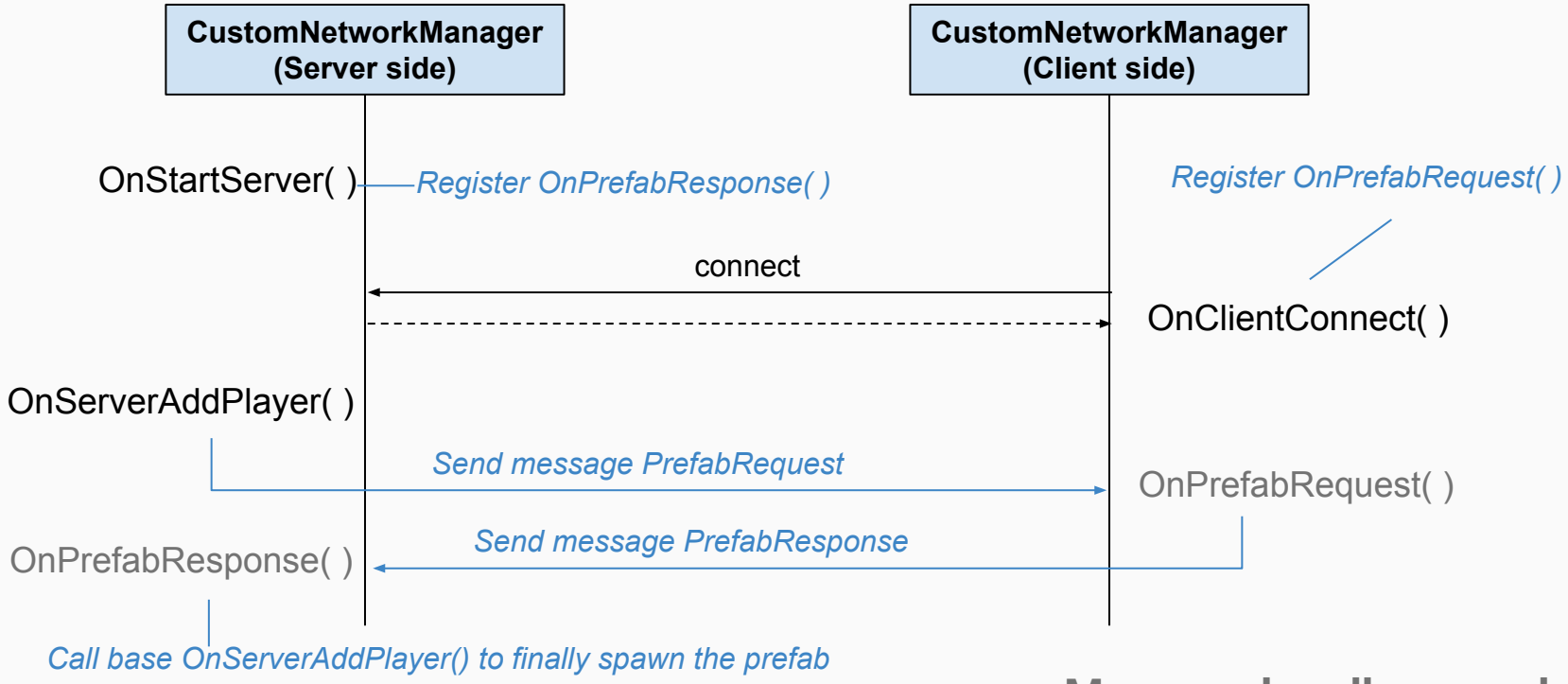


This spawns the default player prefab

Custom NetworkManager - Selecting the player before connecting



Custom NetworkManager - Selecting the player before connecting



Message handlers need to be registered

Custom NetworkManager - Selecting the player before connecting

This is the class of the message with the player prefab request/response. Define it at the top of the file with the implementation of the CustomNetworkManager

```
public class MsgTypes
{
    public const short PlayerPrefabSelect = MsgType.Highest + 1;
    public class PlayerPrefabMsg : MessageBase
    {
        public short controllerId;
        public short prefabIndex;
    }
}
```

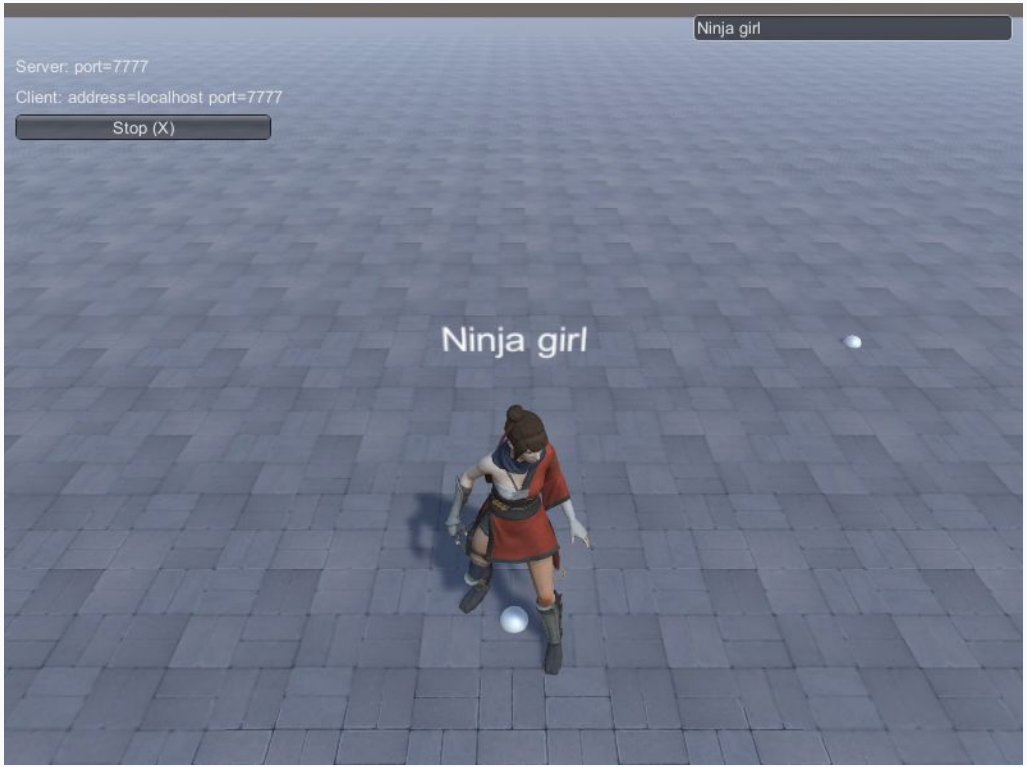
Copy this into CustomNetworkManager to make the selection more interactive

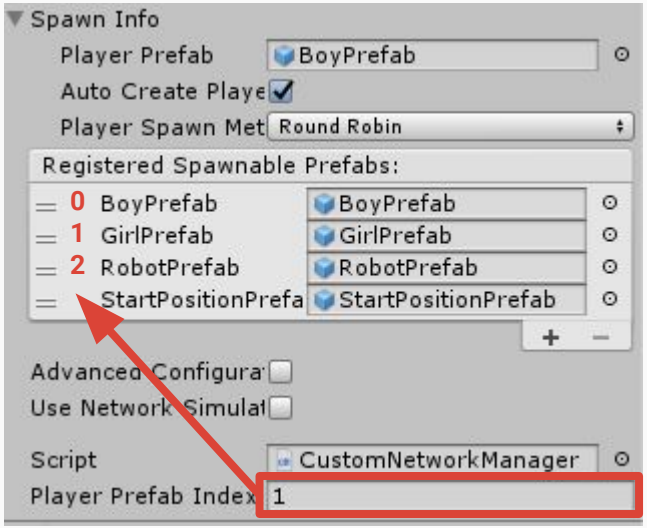
```
public string[] playerNames = new string[] { "Boy", "Girl", "Robot" };

private void OnGUI()
{
    if (!isNetworkActive)
    {
        playerPrefabIndex = (short)GUI.SelectionGrid(
            new Rect(Screen.width - 200, 10, 200, 50),
            playerPrefabIndex,
            playerNames,
            3);
    }
}
```

Our CustomNetworkManager

```
public class CustomNetworkManager : NetworkManager
{
    public short playerPrefabIndex;

    // 1) Executed in the server
    public override void OnStartServer()
    {
        NetworkServer.RegisterHandler(MsgTypes.PlayerPrefabSelect, OnPrefabResponse);
        base.OnStartServer();
    }

    // 2) Executed in the client
    public override void OnClientConnect(NetworkConnection conn)
    {
        client.RegisterHandler(MsgTypes.PlayerPrefabSelect, OnPrefabRequest);
        base.OnClientConnect(conn);
    }

    // 3) Executed in the server
    public override void OnServerAddPlayer(NetworkConnection conn, short playerControllerId)
    {
        MsgTypes.PlayerPrefabMsg msg = new MsgTypes.PlayerPrefabMsg();
        msg.controllerId = playerControllerId;
        NetworkServer.SendToClient(conn.connectionId, MsgTypes.PlayerPrefabSelect, msg);
    }

    // 4) Prefab requested in the client
    private void OnPrefabRequest(NetworkMessage netMsg)
    {
        MsgTypes.PlayerPrefabMsg msg = netMsg.ReadMessage<MsgTypes.PlayerPrefabMsg>();
        msg.prefabIndex = playerPrefabIndex;
        client.Send(MsgTypes.PlayerPrefabSelect, msg);
    }

    // 5) Prefab communicated to the server
    private void OnPrefabResponse(NetworkMessage netMsg)
    {
        MsgTypes.PlayerPrefabMsg msg = netMsg.ReadMessage<MsgTypes.PlayerPrefabMsg>();
        playerPrefab = spawnPrefabs[msg.prefabIndex];
        base.OnServerAddPlayer(netMsg.conn, msg.controllerId);
    }

}
```

Custom NetworkManager - Selecting the player before connecting



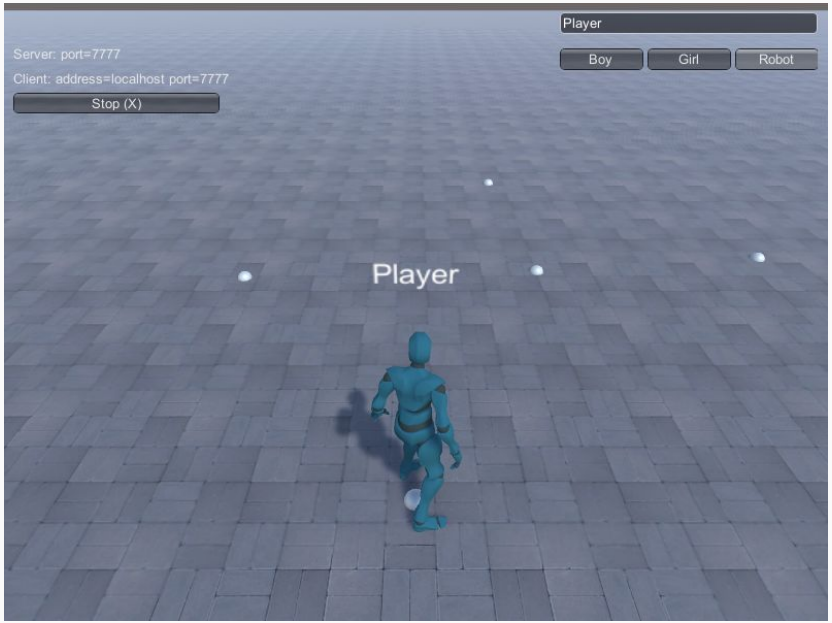


We are sending the index in the array of spawnable prefabs

Changing the player prefab after connecting

Changing the player prefab after connecting

1. New player selection from **local PlayerController**
2. Command executed in **server PlayerController**
3. New player instantiation in **server CustomNetworkManager**



Changing the player prefab after connecting

1. New player selection from **local PlayerController**

In void Start( )

```
NetworkManager mng = NetworkManager.singleton;
networkManager = mng.GetComponent<CustomNetworkManager>();
```

In void OnGUI( )

```
short newIndex = (short)GUILayout.SelectionGrid(
    networkManager.playerPrefabIndex, networkManager.playerNames, 3);
if (newIndex != networkManager.playerPrefabIndex)
{
    networkManager.playerPrefabIndex = newIndex;
    CmdChangePlayerPrefab(newIndex);
}
```

Changing the player prefab after connecting

2. Command executed in **server PlayerController**

Add this new command

```
[Command]
void CmdChangePlayerPrefab(int prefabIndex)
{
    networkManager.ChangePlayerPrefab(this, prefabIndex);
}
```

Changing the player prefab after connecting

3. New player instantiation in **server CustomNetworkManager**

Add this new function

```
public void ChangePlayerPrefab(PlayerController currentPlayer, int prefabIndex)
{
    // Instantiate a new GameObject where the previous one was
    GameObject newPlayer = Instantiate(spawnPrefabs[prefabIndex],
        currentPlayer.gameObject.transform.position,
        currentPlayer.gameObject.transform.rotation);

    // Destroy the previous player GameObject
    NetworkServer.Destroy(currentPlayer.gameObject);

    // Replace the connected player GameObject
    NetworkServer.ReplacePlayerForConnection(
        currentPlayer.connectionToClient, newPlayer, 0);
}
```

Spawn other networked objects

How to spawn more objects?

- Create a **spawnable prefab**
  - Needs a **NetworkIdentity** component (set local authority)
  - **Register** the prefab in the NetworkManager (list of **spawnable objects**)
- In the **PlayerController** script
  - **Add a command** to communicate the spawning action to the server-side NetworkManager
- In the **CustomNetworkManager**
  - **Add a method** to instantiate a prefab
    - The index is the position in the list of registered spawnable prefabs
    - Initial transform of the prefab to spawn

How to spawn more objects?

Command in PlayerController

```
[Command]
public void CmdAddPumpkin()
{
        networkManager.AddObject(5, this.transform);
}
```

Method in CustomNetworkManager

```
public void AddObject(int objIndex, Transform t)
{
    GameObject newObject = Instantiate<GameObject>(
        spawnPrefabs[objIndex],
        t.position,
        Quaternion.identity);

    NetworkServer.Spawn(newObject);
}
```

Client RPCs:

Remote Procedure Calls on clients

Client RPCs

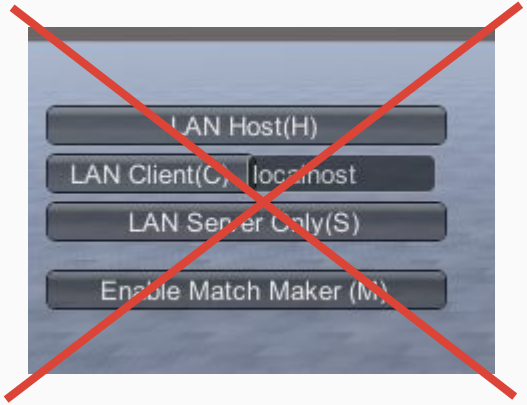
Remote procedure calls on clients (client RPCs):

- Invoked in the server
- Executed in all clients
- Available in **NetworkBehaviour** scripts
- Function names prefixed with **Rpc**
- Similarities with other methods
  - Call convention similar to server commands
  - Alternative to SyncVars + Hooks

```
[ClientRpc]
void RpcRunThisFunctionOnClient()
{
  // do something in all clients
}
```

More about the NetworkManager

Getting rid of the NetworkManagerHUD



- The NetworkManager is a singleton
  - Only one instance
  - Methods to start / stop its activity

```
NetworkManager.singleton.StartHost();

NetworkManager.singleton.StartClient();

NetworkManager.singleton.StopHost();
```

Capturing network events

In our CustomNetworkManager, we can capture events

- Override some NetworkManager callbacks
- For instance, clients being notified about disconnection:

```
public override void OnStopClient()...
```

- All possible NetworkManager callbacks:
  - https://docs.unity3d.com/Manual/NetworkManagerCallbacks.html

Create a nice character selection screen!

Official documentation pages

- Multiplayer and Networking index
  https://docs.unity3d.com/Manual/UNet.html
- Converting a single-player game to Unity Multiplayer
  https://docs.unity3d.com/Manual/UNetConverting.html
- Spawning Game Objects
  https://docs.unity3d.com/Manual/UNetSpawning.html
- State synchronization (SyncVars and SyncLists)
  https://docs.unity3d.com/Manual/UNetStateSync.htm

Official documentation pages

- Remote actions (Commands and ClientRCPs)
  https://docs.unity3d.com/Manual/UNetActions.html
- NetworkManager Callbacks
  https://docs.unity3d.com/Manual/NetworkManagerCallbacks.html
- NetworkBehaviour Callbacks
  https://docs.unity3d.com/Manual/NetworkBehaviourCallbacks.html
- Network Messages
  https://docs.unity3d.com/Manual/UNetMessages.html