Networking Quick Start Guide

Part I - Creating a Lobby

This document is an easy start guide to unity networking API. I'll show how to create a simple game that will contain a lobby, players will be able to create and join matches. The idea is to cover the basics of the networking lobby, such as, how to create a match (using the matchmaking system), join a match, list results on the interface and more..

Results

The result of this tutorial is a lobby scene that allow players to create or join a match. Once two players have been connected to the same match, the lobby automatically transfer both to a play scene. The networked game should run on the play scene.

The created lobby

Index

Networking Quick Start Guide 0

Part I - Creating a Lobby 0

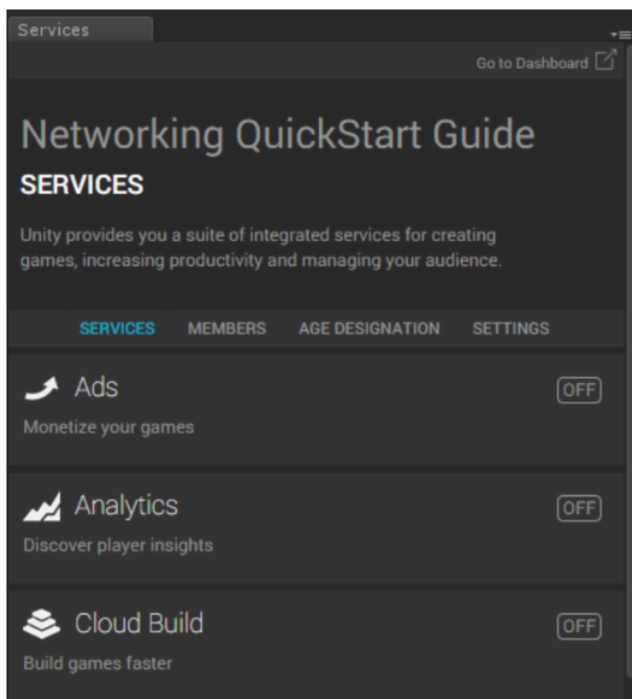
- Results 0
- Index 1
- Getting the project ready 2
- Project's Network Scheme 3
- UI Manager 6
  - Initial version of the UIManager 6
  - Representing a Match 7
  - Initializing the Lobby Manager 7
  - Creating a match 8
  - Joining a match 10
  - Refreshing the list of matches 11
- The Lobby Manager 13
  - Initial version of the LobbyManager 13
  - OnMatchCreate 15
  - OnMatchList 16
- Setting up the Lobby Manager (Inspector) 16
  - Player and LobbyPlayer 16
  - Play Scene and Lobby Scene 17
- Appendix - The code 19
  - UI Manager 19
  - Lobby Manager 24
  - Lobby Player 27
  - Player 27

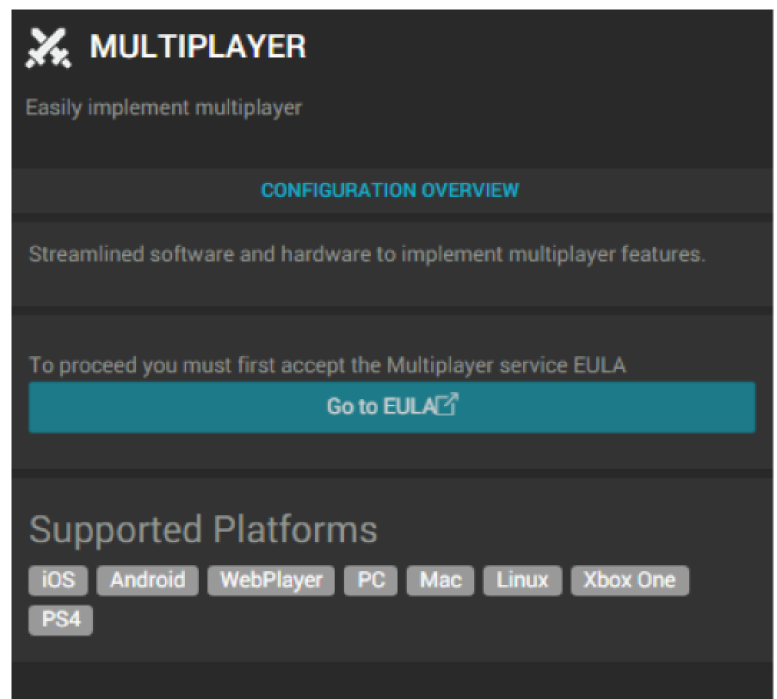
Getting the project ready

This tutorial is made using Unity3D. You will need a fresh Unity Project, I'm using for this example Unity 2017.1.0f3.

In order to use UNET, you will need a Unity account (a free account works) and an organization, you get one by default but you can also create one yourself (https://support.unity3d.com/hc/en-us/articles/210141563-What-is-an-Organization-).

On your unity project, go to the Services tab (top menu bar, Window-> Services or simply CTRL+0). Scroll down until you see Multiplayer, click on it and then click on 'Go to EULA'. You must accept it in order to be able to use Unity's networking services.

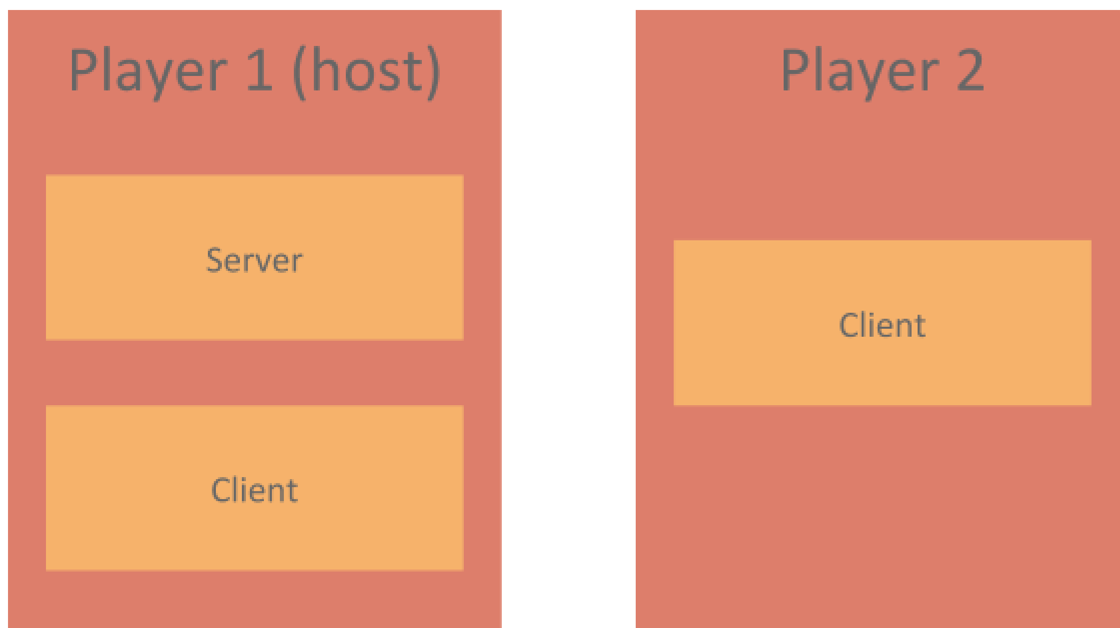




Multiplayer Services Screen

Project's Network Scheme

There are several ways that you can create a network setup, each have its advantages and disadvantages. For the sake of this tutorial I will be working with the following setup.

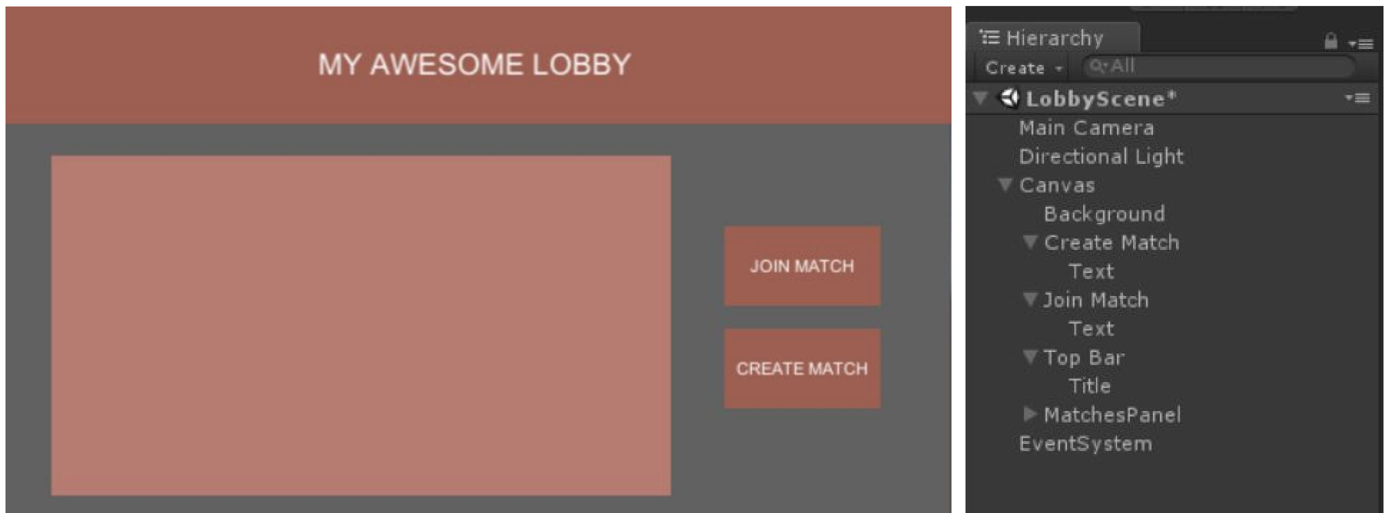


The network scheme

This setup is pretty simple and works great on 1v1 games, like Chess or Hearthstone. It wouldn't work so great for Counter-Strike or Battlefield. On this arrangement we must remember that the Player 1 has latency 0, it will always be in a certain advantage if you game relies on timely response from players. Another thing to consider is that whenever the Player 1 decides to quit the match Player 2 is automatically disconnected.

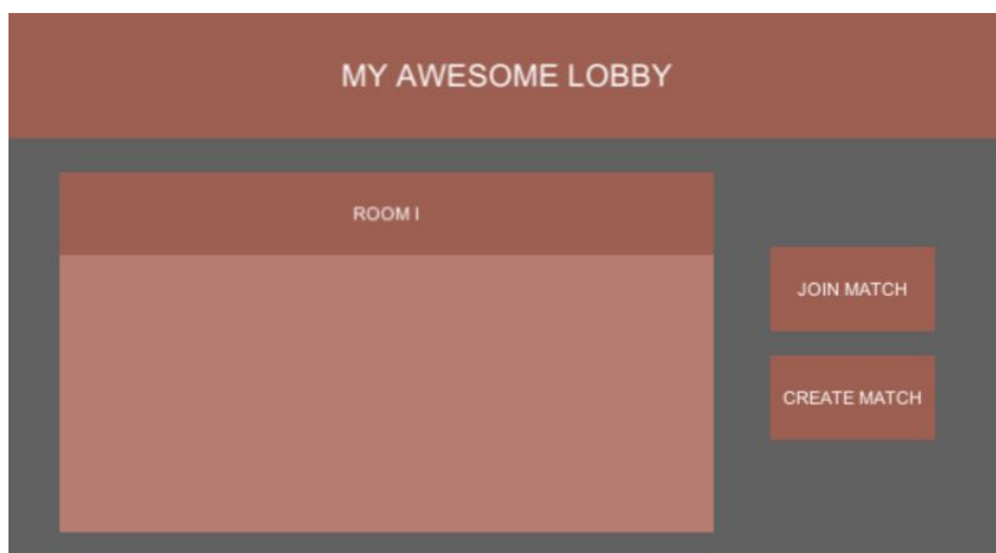
Setting the lobby UI

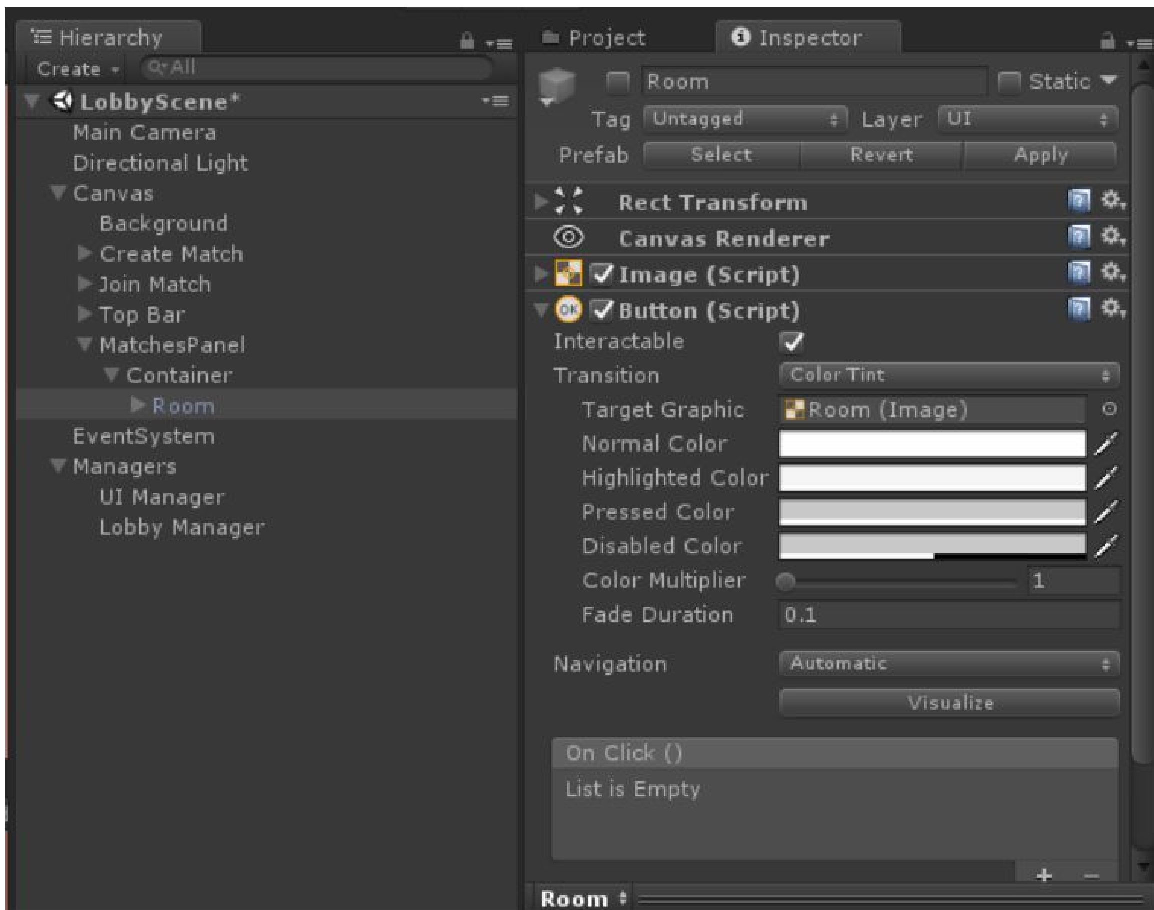
Let’s start by creating the a very simple UI. Like the one below, you need two buttons and a panel to list the matches. The picture shows how it’s made on my project.



Hiearchy and Lobby

It’s needed an UI to represent the match other players will see on the panel. This should be a button so players can click to join. I’ll call it ‘Room’ like in the image below. Once it’s done create a folder called “Prefabs” on your hierarchy and drag it there making it a prefab, we will use it in the future.





Room container hiearchy

Under the MatchesPanel I created a 'Container' object with a 'VerticalLayoutGroup' component on it. So when Rooms are spawned they are properly organized in a list. If you are having trouble creating this UI, refer to the project available at (link).

In order to keep the scene organized, create a game object called 'Managers'. This will hold the managers we will have on this scene. Add two child game objects, 'UI Manager' and 'Lobby Manager'. Each will have its respective manager script.

UI Manager

The UI Manager script will hold the events and methods for controlling the user interface. Create a folder named ‘Scripts’ and a new C# script called UIManager. Initialize it with the code below.

Initial version of the UIManager

```
public class UIManager : MonoBehaviour
{
    public static UIManager Instance;
    public GameObject RoomPrefab;
    /// <summary>
    /// Awake method transforms this class into a singleton.
    void Awake()
    {
        // Performing a Singleton.
        if (UIManager.Instance != null)
        {
            Destroy(this.gameObject);
            Destroy(this);
            return;
        }
        else
            Instance = this;
    }

    // Use this for initialization
    void Start () { }

    // Update is called once per frame
    void Update () { }

    /// <summary>
    /// Fill the list of matches on the UI.
    /// </summary>
    public void ListMatches(List<MatchInfoSnapshot> matchList){}

    /// <summary>
    /// Creates a room and waits for player.
    /// </summary>
    public void OnClickCreateRoomButton(){}

    /// <summary>
    /// When the player click to join the room.
    /// </summary>
    public void OnClickJoinRoomButton(NetworkID networkID){}

}
```

Representing a Match

On the UIManager we will need a class to store information about matches (I'll call it 'Room'). This class will contain a name, a reference to itself and a *NetworkID*. The id is given by the matchmaking system when the match is created. Add the code below to the *UIManager* script.

```
/// <summary>
/// A structure to hold information regarding a room.
/// </summary>
private class Room
{
    public string name;
    public GameObject instance;
    public NetworkID networkId;

    /// <summary>
    /// Constructor for the RoomInformation.
    /// </summary>
    /// <param name="Name"> The name of the room. </param>
    /// <param name="Instance"> The GameObject that represents the room on the list.</param>
    public Room(string Name, GameObject Instance, NetworkID id)
    {
        name = Name;
        instance = Instance;
        networkId = id;
    }
}
```

Each match instantiated in the UI will have a *Room* class representing it on the *UIManager*. Therefore there will be a list of *Room* on the UIManager. Add on the *UIManager*:

```
private List<Room> roomsAvailable;
```

Initializing the Lobby Manager

In order to perform any networking operation, we must initialize the *LobbyManager*. Add the following code to the *Start* method of the *UIManager.* This will start the matchmaking engine.

```
// Use this for initialization
void Start()
{
   // Start the matchmaking
   LobbyManager.Instance.StartMatchMaker();
}
```

Creating a match

The 'Create Match' button will allow a player to create a match. In order to create a match we must call the following method on the lobby manager. Add this method to the *UIManager.* On the Unity Inspector, hook this to the onClick event of the 'Create Button'.

```
/// <summary>
/// Creates a room and waits for player.
/// </summary>
public void OnClickCreateRoomButton()
{
     /* Creates the match. */
     LobbyManager.Instance.matchMaker.CreateMatch(
          "New Room",
          (uint)LobbyManager.Instance.maxPlayers,
          true,
          "", "", "", 0, 0,
          LobbyManager.Instance.OnMatchCreate);
}
```

The LobbyManage's CreateMatch method takes as argument:

1. The name of the new match.
2. The max number of players on that match.
3. If it should advertise the match for other players to see it.
4. The password if there is one.
5. The public client address (leave it blank for standart)
6. The private client address (leave it blank for standart)
7. Elo score for match (leave it 0 for standart)
8. The request domain, this should be the same throughout your application!
9. A method to execute when the match is created.

Once the match is **succesfully created** by the *LobbyManager*, it needs to be instantiate on the UI. In order to do so, let's add another method to the UIManager. The *CreateRoomOnUI*, this will be called by on the *OnMatchCreate* callback we just set. It will receive a NetworkID which is an unique identifier for the match.

Add the *CreateRoomOnUI* to the *UIManager* script:

```
/// <summary>
/// Creates a room on the user interface.
/// </summary>
public void CreateRoomOnUI(NetworkID networkID)
{
    GameObject parent = GameObject.Find("MatchesPanel/Container");
```

```
     /* Creates the room and set it under the container object.*/
    GameObject room = GameObject.Instantiate(RoomPrefab, parent.transform);
    roomsCreated++;
    room.GetComponentInChildren<Text>().text = "Room " + roomsCreated;
    room.GetComponent<Button>().onClick.RemoveAllListeners();
    room.GetComponent<Button>().onClick.AddListener(delegate {
ChangeSelectedRoom(networkID); });

    roomsAvailable.Add(new Room("Room " + roomsCreated, room, networkID));

}
```

Note that at the same time we are adding the room to the UI we are setting it's 'onClick' event with the *ChangeSelectedRoom* method. This method will be responsible for setting the current selected match on the *UIManager* script so it knows which room to join when clicking on the 'Join Match' button.

```
/// <summary>
/// Change the selected room on the UI
/// </summary>
/// <param name="selection"></param>
public void ChangeSelectedRoom(NetworkID selection)
{
    this.roomSelected = (ulong)selection;
}
```

A method to remove a match from the UI will also be needed, the following code finds and remove a match from the UI.

```
/// <summary>
/// Creates a room on the user interface.
/// </summary>
public void RemoveRoomOnUI(NetworkID networkID)
{
    Room elementToBeDeleted = roomsAvailable.Find(R => R.networkId == networkID);
    if (elementToBeDeleted != null)
    {
        roomsAvailable.Remove(elementToBeDeleted);
        GameObject.Destroy(elementToBeDeleted.instance.gameObject);
    }
    else
        Debug.LogError("<color=orange>UIManager:</color>Could not find the element to
destroy.");
}
```

Add the variables *roomSelected* (the networkingID of the selected match) and *roomsCreated* (just a variable to help naming Rooms) to the *UIManager* script.

```
private ulong roomSelected = (ulong)NetworkID.Invalid;

private int roomsCreated = 0;
```

Joining a match

In order to join a match we must find the NetworkID of the selected room and call the JoinMatch method.

```
/// <summary>
/// When the player click to join the room.
/// </summary>
public void OnClickJoinRoomButton()
{
    if (roomsAvailable[roomSelected] == null)
    {
        Debug.Log("<color=orange>UIManager</color> There seems to be an error the selected
room doesn't exist.");
        return;
    }

    NetworkID selected = roomsAvailable[roomSelected].networkId;

    /* Call the method that tries to login */
    Debug.Log("Attempt to join.");
    LobbyManager.Instance.matchMaker.JoinMatch(selected, "", "", "", 0, 0,
LobbyManager.Instance.OnMatchJoined);
}
```

The JoinMatch method takes as argument:

1. The id of the match we want to join.
2. The password if there is one.
3. The public client address (leave it blank for standart)
4. The private client address (leave it blank for standart)
5. Elo score for match (leave it 0 for standart)
6. The request domain, this should be the same throughout your application!
7. A method to execute when the match is joined.

Refreshing the list of matches

Now we want to get a list of matches from the server and display it on the UI. To get it from the server let's create the method.

```
private void RefreshMatchList()
{

    LobbyManager.Instance.matchMaker.ListMatches(
            0,
            6,
           "",
         true,
            0,
            0,
            LobbyManager.Instance.OnMatchList);
}
```

The *ListMatches* takes as arguments:

1. Start page number (we want the first page).
2. Number os rooms per page (let's set it to 6).
3. A string to filter page names (let's leave it empty).
4. A bool if we want to filter private matches (let's say true).
5. Elo score for match (leave it 0 for standart)
6. The request domain, this should be the same throughout your application!
7. A callback to do something when completed.

We want the room list UI to run as soon as the scene is loaded and refresh every second so we always know which rooms are available to join. In order to do so I'll create a coroutine and run it at the start of the *UIManager.*

```
private IEnumerator RefreshLobby()
{
    while (true)
    {
        RefreshMatchList();
        yield return new WaitForSeconds(1f);
    }
}
```

Add it to the *Start* function of the *UIManager* to guarantee that it will start running as soon as the script starts. The *Start* function should look like this.

```
// Use this for initialization
void Start()
{
    roomsAvailable = new List<RoomInformation>();
    // Start the matchmaking
    LobbyManager.Instance.StartMatchMaker();
    StopAllCoroutines();
    StartCoroutine(RefreshLobby());
}
```

At this point we are able to request to the server which rooms are available online, every 1 second. However there isn't a method to update the list match list on the user interface, removing old rooms and adding new ones.

The method below is able to use the *CreateRoomOnUI* and *RemoveRoomOnUI* methods that we developed and update the UI.

```
public void UpdateMatchList(List<MatchInfoSnapshot> matchList)
{
    /* Case there is no match. */
    if (matchList.Count == 0)
    {
        int elementIndex = 0;
        while (roomsAvailable.Count != 0)
        {
            RemoveRoomOnUI(roomsAvailable[elementIndex].networkId);
        }
        return;
    }

    /* Check every room online to see if it exist on the UI if not create it.*/
    for (int i = 0; i < matchList.Count; ++i)
    {
        /* If room exist online but not on UI create it.*/
        if (roomsAvailable.Find(R => R.networkId == matchList[i].networkId) == null)
            CreateRoomOnUI(matchList[i].networkId);
    }

    /* Check every room on the UI to see if it exists online.*/
    List<Room> roomListCopy = new List<Room>(roomsAvailable);
    foreach (Room roomy in roomListCopy)
    {
        MatchInfoSnapshot matchInfoSnapshot = matchList.Find(N => N.networkId ==
roomy.networkId);
        /* If doesn't exist remove from UI. */
        if (matchInfoSnapshot == null)
            RemoveRoomOnUI(roomy.networkId);
    }
```

```
}
```

The Lobby Manager

This class will inherit from the *NetworkLobbyManager* class (UnityEngine.Networking package). Notice that the *NetworkLobbyManager*, is a child of the *NetworkManager* class. In that way, **this script will be have access to the callback methods as the network manager runs**. We will be looking into 5 callbacks at this point.

1. OnMatchList
2. OnMatchCreate
3. OnDestroyMatch
4. OnServerAddPlayer
5. OnCreateGamePlayerfromLobbyPlayer

Initial version of the LobbyManager

```
/// <summary>
/// Class that hold the Lobby methods for creating and managing rooms.
/// </summary>
public class LobbyManager : NetworkLobbyManager
{
    #region Properties

    /// <summary>
    /// Variable to save memory address reference to this singleton.
    /// </summary>
    public static LobbyManager Instance;

    public MatchInfo CurrentMatch;

    private UIManager uiManagerReference;

    // ------------------------------------------------------------------------

    /// <summary>
    /// Awake method transforms this class into a singleton.
    /// To access this singleton from another class, see the public variable:
    /// <see cref="Instance"/>
    /// </summary>
    void Awake()
    {
        // Performing a Singleton.
        if (LobbyManager.Instance != null)
        {
```

```
            Destroy(this.gameObject);
            Destroy(this);
            return;
        }
        else
            Instance = this;
    }

    // ------------------------------------------------------------------------

    /// <summary>
    /// Callback that is called whenever the matches are listed from the server.
    /// </summary>
    public override void OnMatchList(bool success, string extendedInfo, List<MatchInfoSnapshot>
matchList)
    {
        base.OnMatchList(success, extendedInfo, matchList)
    }

    // ------------------------------------------------------------------------

    /// <summary>
    /// Event that runs when a match is created.
    /// </summary>
    public override void OnMatchCreate(bool success, string extendedInfo, MatchInfo matchInfo)
    {
        base.OnMatchCreate(success, extendedInfo, matchInfo);
    }

    // ------------------------------------------------------------------------

    /// <summary>
    /// Callback that is called whenever a match is destroyed.
    /// </summary>
    public override void OnDestroyMatch(bool success, string extendedInfo)
    {
        base.OnDestroyMatch(success, extendedInfo);

    }

    // ------------------------------------------------------------------------

    public override void OnServerAddPlayer(NetworkConnection conn, short playerControllerId)
    {
        base.OnServerAddPlayer(conn, playerControllerId);
    }

    // ------------------------------------------------------------------------
```

```
    public void OnCreateGamePlayerfromLobbyPlayer(LobbyPlayer lp)
    {
        base.OnCreateGamePlayerfromLobbyPlayer(lp);
    }
}
```

OnMatchCreate

Now let's add some extra functionality to those callbacks. For the *OnMatchCreate*, we would like to notify the *UIManager* that a match was successfully created so it can add a reference to it in the UI.

```
/// <summary>
/// Event that runs when a match is created.
/// </summary>
/// <param name="success">Indicates if the request succeeded.</param>
/// <param name="extendedInfo">A text description of the failure if success is
false.</param>
/// <param name="matchInfo">Information about the match created.</param>
public override void OnMatchCreate(bool success, string extendedInfo, MatchInfo matchInfo)
{
    base.OnMatchCreate(success, extendedInfo, matchInfo);

    /* If there is no reference to the UI Manager */
    if (uiManagerReference == null)
        uiManagerReference = GameObject.Find("Managers/UI
Manager").GetComponent<UIManager>();

    /* Check if recovered. */
    if (uiManagerReference == null)
        Debug.Log("<color=red>Lobby Manager:</color> Can't find the local area UI. Are you
sure this is being called on the correct scene?");
    else
        uiManagerReference.CreateRoomOnUI(matchInfo.networkId);

    /* Save a reference to the current match.. (server only)*/
    CurrentMatch = matchInfo;
}
```

OnMatchList

After listing matches we want to update the UI, removing rooms that don't exist anymore and adding rooms that were created. This can be done by calling *UpdateMatchList* on the UIManager passing as reference the list of *MatchInfoSnapshots* on the server.

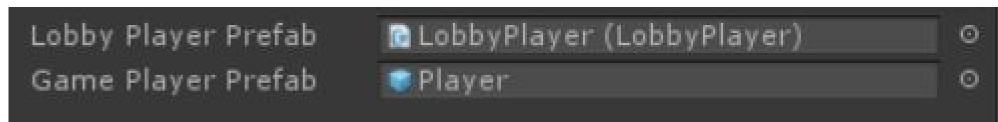
```
/// <summary>
```

```
/// Callback that is called whenever the matches are listed from the server.
/// </summary>
/// <param name="matchList">Information about the match created.</param>
public override void OnMatchList(bool success, string extendedInfo, List<MatchInfoSnapshot>
matchList)
{
    base.OnMatchList(success, extendedInfo, matchList);
    uiManagerReference.UpdateMatchList(matchList);
}
```

Setting up the Lobby Manager (Inspector)

Player and LobbyPlayer

Unity's networking system uses a prefab to represent a player on the lobby (LobbyPlayer), and a prefab for player in game. It's needed to set those on the inspector (LobbyManager).



Create a script called 'Player', set it as a child of *NetworkBehaviour* (instead of *MonoBehaviour*). At this point this script can be blank. On the ***Networking Quick Start Part II*** this will be further developed, allowing information to be exchanged between game instances.

Finally attach this script to a GameObject, make it a prefab and attached to the 'Game Player Prefab' field on the Lobby Manager script.

Now create a script called 'LobbyPlayer', set is as a child of *NetworkLobbyPlayer*. On the *Start* method add the following.

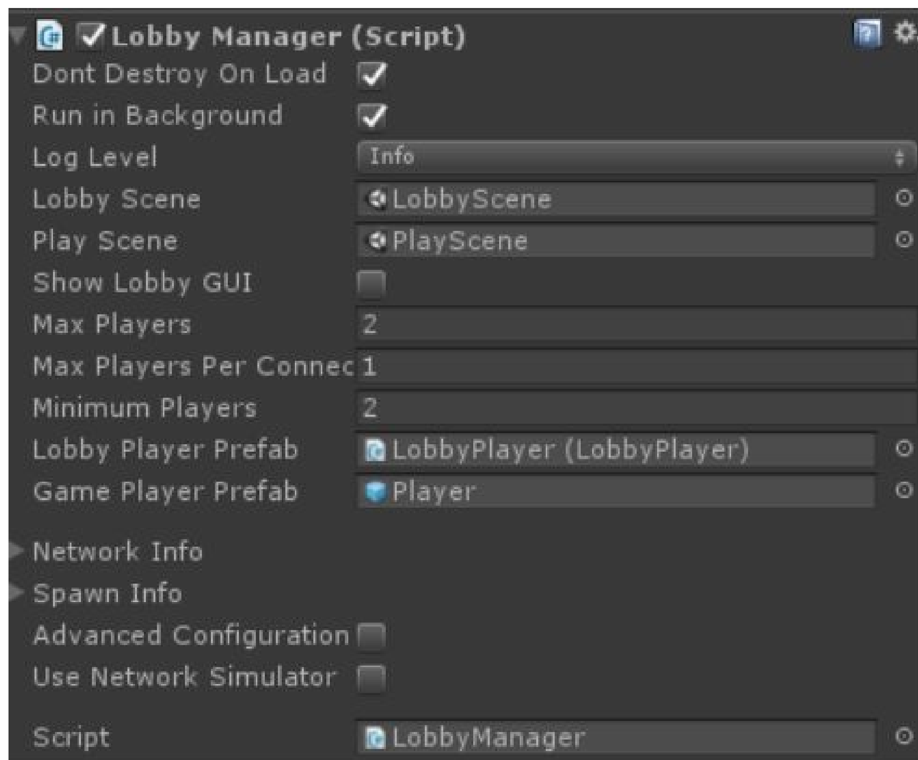
```
if (isLocalPlayer)
            this.SendReadyToBeginMessage();
```

This line tells the lobby manager that this player is all set to start playing the match. Once all the players are ready the game will automatically starts. Create a new game object called 'Lobby Player' attach this script to it, make it a prefab and attach it to the 'Lobby Player Prefab' field.

Play Scene and Lobby Scene

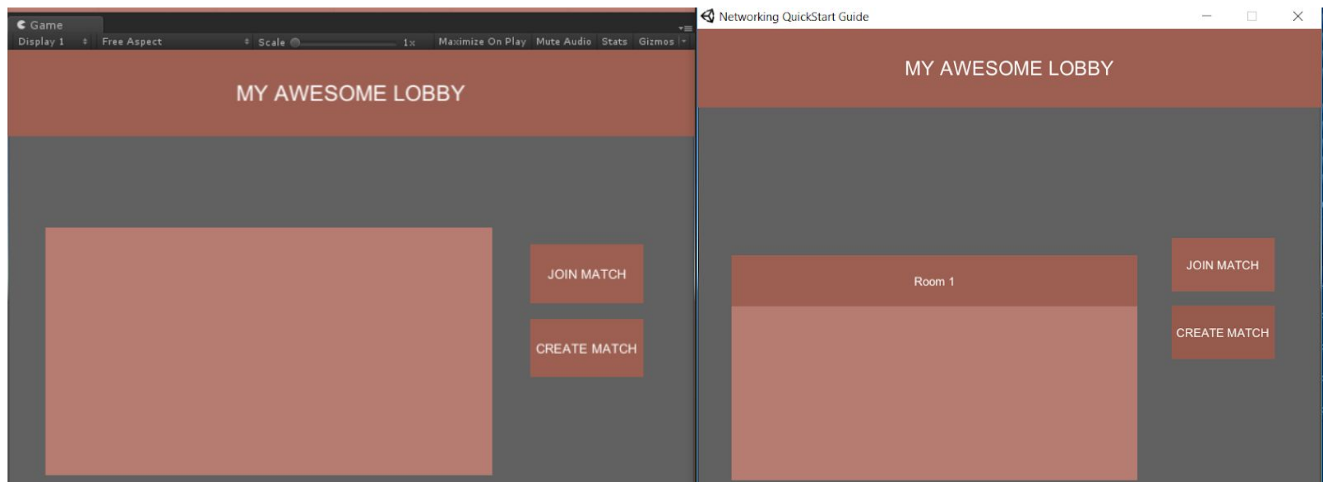
The networking system also uses two scenes, one for the lobby and other for the game. Once the match is ready to be set it will start the game scene. In that way, set two scenes the current one as the lobby scene and another empty scene as the game scene.

When testing if everything worked, both players should be redirected to the game scene. And their Player object should be instantiated. The picture below shows how the lobby manager should be set (Remember to uncheck Show Lobby GUI).



Testing

In order to test the network game, it's necessary to make a build. Once the build is done execute it and create a match. On the Unity Editor run the game, you should see the match created on the list. Upon clicking on it and clicking 'Join'. Both players should be taken to the game scene.



Conclusion

This document was made to get anyone with minimum Unity experience into making a simple lobby system. Working with networked games is really different from working with single player games. An important lesson that I learned during this process is to set as many log messages as you can.

Each callback can be overwritten to have a debug message added, in that way you can know exactly how to code is running. Don't save on log messages! Thanks for reading :-)

Appendix - The code

UI Manager

The final code *UIManager.cs* that you should have by the end of this tutorial can be found below.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking.Match;
using UnityEngine.Networking.Types;
using UnityEngine.UI;

public class UIManager : MonoBehaviour
{
    public static UIManager Instance;

    public GameObject RoomPrefab;

    private ulong roomSelected = (ulong)NetworkID.Invalid;

    private int roomsCreated = 0;

    private List<Room> roomsAvailable;

    /// <summary>
    /// Awake method transforms this class into a singleton.
    /// To access this singleton from another class, see the public variable:
    void Awake()
    {
        // Performing a Singleton.
        if (UIManager.Instance != null)
        {
            Destroy(this.gameObject);
            Destroy(this);
            return;
        }
        else
        {
            Instance = this;
        }
    }

    /// <summary>
    /// A structure to hold information regarding a room.
    /// </summary>
```

```
    private class Room
    {
        public string name;
        public GameObject instance;
        public NetworkID networkId;

        /// <summary>
        /// Constructor for the RoomInformation.
        /// </summary>
        /// <param name="Name"> The name of the room. </param>
        /// <param name="Instance"> The GameObject that represents the room on the list.
</param>
        public Room(string Name, GameObject Instance, NetworkID id)
        {
            name = Name;
            instance = Instance;
            networkId = id;
        }
    }

    // Use this for initialization
    void Start()
    {
        roomsAvailable = new List<Room>();
        // Start the matchmaking
        LobbyManager.Instance.StartMatchMaker();
        StopAllCoroutines();
        StartCoroutine(RefreshLobby());
    }

    // Update is called once per frame
    void Update() { }

    /// <summary>
    /// Fill the list of matches on the UI.
    /// </summary>
    /// <param name="matchList">The list of matches available online.</param>
    public void ListMatches(List<MatchInfoSnapshot> matchList)
    {

    }
    /// <summary>
    /// Creates a room and waits for player.
    /// </summary>
    public void OnClickCreateRoomButton()
    {
        /* Creates the match. */
        LobbyManager.Instance.matchMaker.CreateMatch(
            "New Room",
```

```
            (uint)LobbyManager.Instance.maxPlayers,
            true,
            "", "", "", 0, 0,
            LobbyManager.Instance.OnMatchCreate);

    }
    /// <summary>
    /// When the player click to join the room.
    /// </summary>
    public void OnClickJoinRoomButton()
    {
        if (roomSelected == (ulong)NetworkID.Invalid)
        {
            Debug.Log("<color=orange>UIManager:</color> There seems to be an error the
selected room doesn't exist.");
            return;
        }

        NetworkID selected = roomsAvailable.Find(R => R.networkId ==
(NetworkID)roomSelected).networkId;

        /* Call the method that tries to login */
        Debug.Log("Attempt to join.");
        LobbyManager.Instance.matchMaker.JoinMatch(selected, "", "", "", 0, 0,
LobbyManager.Instance.OnMatchJoined);
    }

    /// <summary>
    /// Creates a room on the user interface.
    /// </summary>
    public void CreateRoomOnUI(NetworkID networkID)
    {
        GameObject parent = GameObject.Find("MatchesPanel/Container");

        /* Creates the room and set it under the container object.*/
        GameObject room = GameObject.Instantiate(RoomPrefab, parent.transform);
        roomsCreated++;
        room.GetComponentInChildren<Text>().text = "Room " + roomsCreated;
        room.GetComponent<Button>().onClick.RemoveAllListeners();
        room.GetComponent<Button>().onClick.AddListener(delegate {
ChangeSelectedRoom(networkID); });

        roomsAvailable.Add(new Room("Room " + roomsCreated, room, networkID));

    }

    /// <summary>
    /// Creates a room on the user interface.
    /// </summary>
```

```
    public void RemoveRoomOnUI(NetworkID networkID)
    {
        Room elementToBeDeleted = roomsAvailable.Find(R => R.networkId == networkID);
        if (elementToBeDeleted != null)
        {
            roomsAvailable.Remove(elementToBeDeleted);
            GameObject.Destroy(elementToBeDeleted.instance.gameObject);
        }
        else
            Debug.LogError("<color=orange>UIManager:</color>Could not find the element to
destroy.");
    }

    public void UpdateMatchList(List<MatchInfoSnapshot> matchList)
    {
        /* Case there is no match. */
        if (matchList.Count == 0)
        {
            int elementIndex = 0;
            while (roomsAvailable.Count != 0)
            {
                RemoveRoomOnUI(roomsAvailable[elementIndex].networkId);
            }
            return;
        }

        /* Check every room online to see if it exist on the UI if not create it.*/
        for (int i = 0; i < matchList.Count; ++i)
        {
            /* If room exist online but not on UI create it.*/
            if (roomsAvailable.Find(R => R.networkId == matchList[i].networkId) == null)
                CreateRoomOnUI(matchList[i].networkId);
        }

        /* Check every room on the UI to see if it exists online.*/
        List<Room> roomListCopy = new List<Room>(roomsAvailable);
        foreach (Room roomy in roomListCopy)
        {
            MatchInfoSnapshot matchInfoSnapshot = matchList.Find(N => N.networkId ==
roomy.networkId);
            /* If doesn't exist remove from UI. */
            if (matchInfoSnapshot == null)
                RemoveRoomOnUI(roomy.networkId);
        }
    }

    /// <summary>
    /// Change the selected room on the UI
    /// </summary>
```

```
    /// <param name="selection"></param>
    public void ChangeSelectedRoom(NetworkID selection)
    {
        this.roomSelected = (ulong)selection;
    }

    /// <summary>
    /// Method that refreshes the lobby interface.
    /// </summary>
    private void RefreshMatchList()
    {
        // Set the callback to do something once the matchmaker have started
        if (LobbyManager.Instance == null)
        {
            Debug.Log("Lobby manager seems to be destroyed..");
            return;
        }

        if (LobbyManager.Instance.matchMaker == null)
        {
            LobbyManager.Instance.StartMatchMaker();
        }

        LobbyManager.Instance.matchMaker.ListMatches(0, 6, "", true, 0, 0,
LobbyManager.Instance.OnMatchList);
    }

    /// <summary>
    /// Coroutine that calls RefreshMatchList every 1 second.
    /// </summary>
    /// <returns></returns>
    private IEnumerator RefreshLobby()
    {
        while (true)
        {
            RefreshMatchList();
            yield return new WaitForSeconds(1f);
        }
    }

}
```

Lobby Manager

The final code *LobbyManager.cs* that you should have by the end of this tutorial can be found below.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.Networking.Match;
using UnityEngine.SceneManagement;

/// <summary>
/// Class that hold the Lobby methods for creating and managing rooms.
/// </summary>
public class LobbyManager : NetworkLobbyManager
{
    #region Properties

    /// <summary>
    /// Variable to save memory address reference to this singleton.
    /// </summary>
    public static LobbyManager Instance;

    public MatchInfo CurrentMatch;

    #endregion

    #region Attributes

    private UIManager uiManagerReference;

    #endregion

    #region Methods

    #region Unity Methods

    // ------------------------------------------------------------------------

    /// <summary>
    /// Awake method transforms this class into a singleton.
    /// To access this singleton from another class, see the public variable:
    /// <see cref="Instance"/>
    /// </summary>
    void Awake()
    {
        // Performing a Singleton.
        if (LobbyManager.Instance != null)
```

```
        {
            Destroy(this.gameObject);
            Destroy(this);
            return;
        }
        else
        {
            Instance = this;
        }
    }

    // -----------------------------------------------------------------------

    // Use this for initialization
    void Start()
    {
        if (uiManagerReference == null)
            uiManagerReference = GameObject.Find("Managers/UI
Manager").GetComponent<UIManager>();

        // Start the matchmaking
        LobbyManager.Instance.StartMatchMaker();
    }

    // -----------------------------------------------------------------------

    #endregion

    #region Public Methods

    // -----------------------------------------------------------------------

    /// <summary>
    /// Callback that is called whenever the matches are listed from the server.
    /// </summary>
    /// <param name="success">Indicates if the request succeeded.</param>
    /// <param name="extendedInfo">A text description of the failure if success is
false.</param>
    /// <param name="matchList">Information about the match created.</param>
    public override void OnMatchList(bool success, string extendedInfo,
List<MatchInfoSnapshot> matchList)
    {
        base.OnMatchList(success, extendedInfo, matchList);
        uiManagerReference.UpdateMatchList(matchList);
    }

    // -----------------------------------------------------------------------

    /// <summary>
```

```
    /// Event that runs when a match is created.
    /// </summary>
    /// <param name="success">Indicates if the request succeeded.</param>
    /// <param name="extendedInfo">A text description of the failure if success is
false.</param>
    /// <param name="matchInfo">Information about the match created.</param>
    public override void OnMatchCreate(bool success, string extendedInfo, MatchInfo
matchInfo)
    {
        base.OnMatchCreate(success, extendedInfo, matchInfo);

        /* If there is no reference to the UI Manager */
        if (uiManagerReference == null)
            uiManagerReference = GameObject.Find("Managers/UI
Manager").GetComponent<UIManager>();

        /* Check if recovered. */
        if (uiManagerReference == null)
            Debug.Log("<color=red>Lobby Manager:</color> Can't find the local area UI. Are
you sure this is being called on the correct scene?");
        else
            uiManagerReference.CreateRoomOnUI(matchInfo.networkId);

        /* Save a reference to the current match.. (server only)*/
        CurrentMatch = matchInfo;
    }
    #endregion
    #endregion
}
```

Lobby Player

The final code *LobbyPlayer.cs* that you should have by the end of this tutorial can be found below.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class LobbyPlayer : NetworkLobbyPlayer
{

    // Use this for initialization
    void Start ()
    {
        if (isLocalPlayer)
            this.SendReadyToBeginMessage();
    }
}
```

Player

The final code *Player.cs* that you should have by the end of this tutorial can be found below.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Networking;

public class Player : NetworkBehaviour
{

       // Use this for initialization
       void Start () {}

       // Update is called once per frame
       void Update () {}
}
```