uMMO – Unity MMO Solution

App Version: 1.2
Document version: 1.23
Last Update: June, 23th 2014
Required Unity version: 4.5
Author: SoftRare - www.softrare.eu
Genre: Unity and Unity pro plugin
Description: This package allows you to create MMO games using native Unity networking framework. Create an MMO from any single-player game! Build MMORPG, MMOFPS, MMORTS, ….

Always find the newest version of this document here:
http://softrare.eu/user_files/unity/uMMO/Readme.pdf

See it in action here:
http://softrare.eu/user_files/unity/uMMO/web.html

Need support? http://www.softrare.eu/contact.html

Table of Contents

1. Introduction

1.1 Why MMO..

We are very proud to bring you this Unity extension. We have invested months of work to be able to present you a fully working and stable plugin. We have gone the extra mile and made sure you can use this extension with absolute ease and still keep overview.

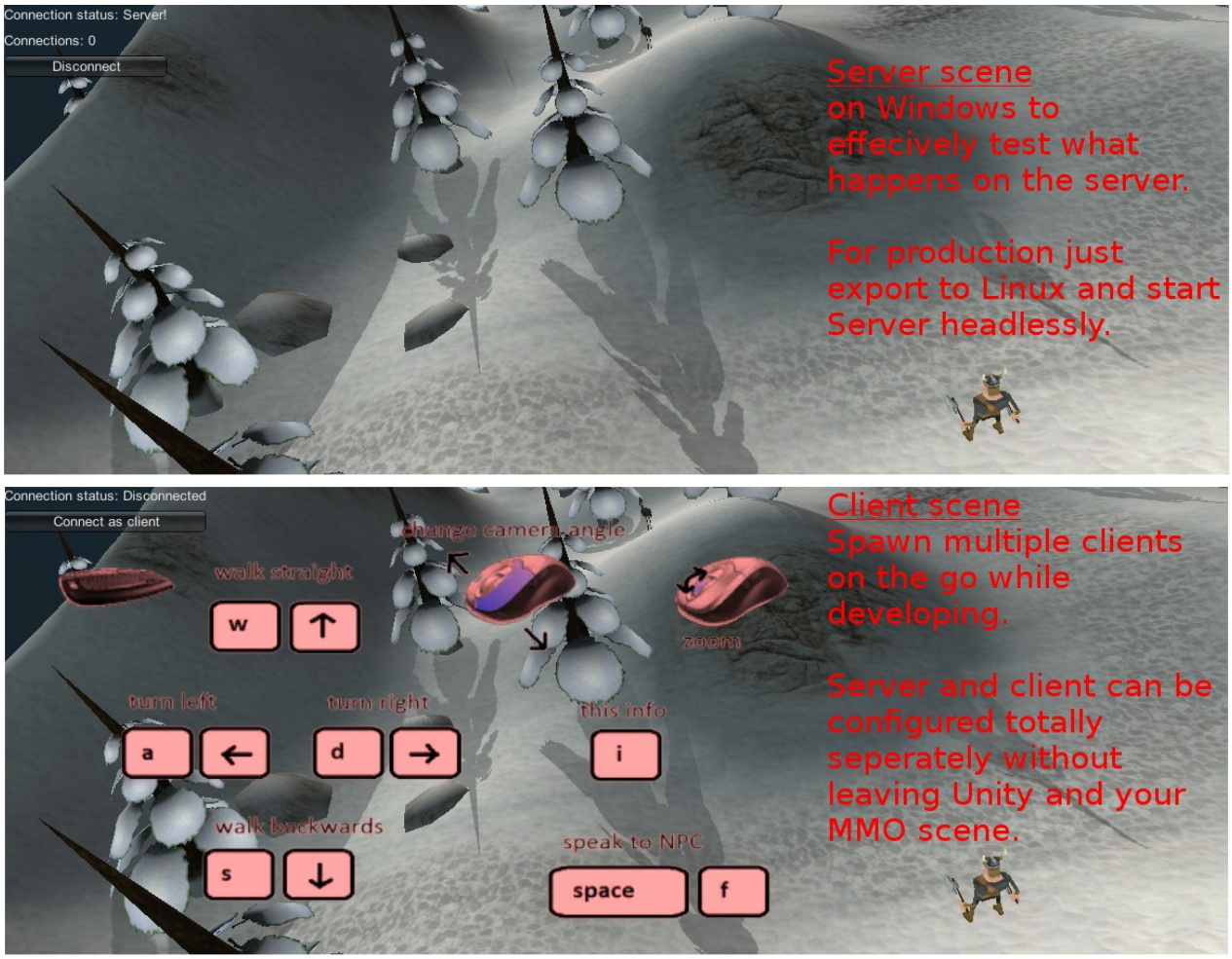
MMO (Massively Mulitplayer Online) games have gotten incredible attention, which seems to only increase. When playing MMO, the old multiplayer concepts of separate maps/hermetically separate areas which look the same every time, no matter what happened to it last match (somewhat like the movie *Groundhog Day*), feel very outdated nowadays and simultaneously frustrating. Because what you do in classical multiplayer games does not leave a mark in the virtual world of your choosing. MMO is a different concept. It's aim is consistency, and the possibility to visit different parts of virtual worlds whenever the user pleases, not when a certain map is incidentally the next one in the server queue, and at the same time, being connected to a huge number of players. MMO not only emphasizes on the amount of people being able to connect but a certain technique to do it. There is the central server being online practically always, and many clients. This idea alone creates challenges, from game-play to maintainability and to security challenges. With this extension we try to give you the right tools to succeed.

1.2 ..and why uMMO?

At the same time, while there is this old concept of multiplayer games, there is also this old concept of programming MMOs. By nature, playing MMO involves different architectures and platforms. So, for every MMO you have at least one server-build and one client-build. Well, those describe the same game, only from different perspectives. While a lot of effort is put into the graphical representation on the clients machine, the server is responsible for making decisions. Still, those appearances and decisions evolve around the same data models. Data models **are** the game, and for many games the data models are incredibly complex. So, while the old style of programming MMOs means constantly synchronizing data models between architectures, in uMMO you leave this behind. Having a Unity-run server also has the benefit of incredibly powerful server-side collision detection. How? Well, its Unity, so just program it like you would program a single-player game ☺ you don't have to hire an extra programmer anymore who's sole responsibility is mapping the whole server world to some kind of data-only representation which then runs on the server and constantly calculates possible (!) impacts and the like. On the other hand, with a lot of game concepts it is absolutely mandatory to have server-side collision detection, not only to counter cheating/glitching ect.

Of course, like all plugins/programs/simulations also this plugin has a limited scope, so please check the limitations of the plugin out at the end of the document.

1.3 Server and client(s)



Above you can see the result of two test runs: One represents the server, beneath that is the client. To simplify matters, in uMMO, you can configure the editor to be your server scene and connect any standalone export as a client. Please read the Quick Start Guide in this document for detailed step for step info.

1.4 Scripts for different architectures

Above you can see a player view from our public MMO demo scene. A few elements in the screenshot have been marked to be able to discuss them here in detail:

1. This is the local player character, instantiated on the client which controls the character.
2. This is a GUI element which can only be seen by this client. It represents the amount of coins the local player has collected in the demo scene. So uMMO was specifically designed to have different GameObjects run different scripts on different architectures, and even on different clients.
3. This is a remote player characters, synchronized to this client as of i.e. movement and rotation, but it is controlled from a totally different client (player).
4. This element represents the number of coins which were selected by the player which is represented right beneath this marked element. This demonstrates once again, that, even though the remote and local player characters are effectively instantiated from the same prefab, different scripts are enabled in different situations. You can also i.e. determine scripts which will only run on the server, and not only switch scripts but also switch GameObject children of your character prefab.
5. This is a very simple example of a connection GUI. A connection GUI is a module in uMMO, which means you can replace it with your own code. The (admittedly) quite ugly connection GUI which comes with this extension can be overwritten, without changing native plugin code. This will benefit you if a new version of uMMO is being put online and you want to upgrade but don't want to risk overwriting your current game code.

1.5 In-Editor documentation

We have implemented extensive in-Editor documentation, so that you can learn about the plugin while you are using it.

In order to use uMMO, you will have to instantiate the uMMO central prefab. It contains the documentation necessary to understand the properties it contains.

Also, all uMMO modules and the uMMO_NetObject class are very extensively documented in the Unity Editor.

Once you are familiar with the module, you are free to disable the documentation to gain more overview. The property to switch off in-Editor documentation is highlighted in the screenshot shown on the left.

1.6 Demo scenes

Learning by example is probably the most effective way of getting a foot into programing. It does not matter if you are a new to the whole field of programming or you have decades of experience: when you get hands on a new plugin/IDE/programming language you will have a lot of questions. That's why we included two demo scenes in the package:

Demo scene 1: This is the main demo scene demonstrating a demo game scene (also available as a live demo on our website: http://softrare.eu/user_files/unity/uMMO/web.html).

The gameplay reassembles a very simple RPG game which lets you talk to different characters

Technically speaking it demonstrates an authoritative server setup (server-side collisions), involving NPCs (Non-player-characters) and collectables. You are invited and encouraged to roam the scene and find out more about the usage of uMMO.

Demo scene 2: This demo scene is a test scene which demonstrates a certain aspect of uMMO: data transmission filters. In this scene, the data of players/NPCs which are far away are not being shown to the clients: The server decides in configurable periodic intervals which user should receive transmission from which other user.

The range of the filter is shown graphically and is configured to be very short, so that in this demo you can watch characters getting in and out of line of sight: Characters you don't see, are not transmitting data to you and are not receiving from you. This saves a lot of bandwidth.

2. Quick Start Guide

This quick start guide assumes you have some sort of a singleplayer game scene and a singleplayer character at your disposal, which we will now together try to turn into an MMO scene.
If you don't have anything to work with, instantiate a simple plane (GameObject → Create Other → Plane) as walking ground, and download this Lerp character (originally created by the Unity Team): http://softrare.eu/user_files/unity/uMMO/lerp_test.unitypackage

1. Don't let the amount of text scare you, the steps are **very easy** to execute, at the same time a lot of the steps are repetitive:

- Import the uMMO package from the Asset Store
- Create new Scene
- Instantiate uMMO prefab (under uMMO/prefabs/INSTANTIATE THIS)

→ Enable Non Authoritative Server Setup
(NOTE: in that case all clients "own" their own player character)
→ Set Architecture To Compile to TEST_UnityEditorIsServer_OthersAreClients

- Now you have an MMO game scene, but don't start the scene just yet.

2. Now lets create an MMO character which will be instantiated once a player joins the server:

- Instantiate your player character prefab into the scene

(NOTE: all movement-/other behavior scripts should be placed in the top GameObject in the player GameObject hierarchy (so preferably don't place scripts on children of this player GameObject. If necessary replace some scripts).

- Place the script uMMO_NetObject on the prefabs top GameObject. Now your single-player character is an MMO character. Lets modify it to activate the right scripts in the right situation.
- In Editor set Object Type to Player
- Does your character have it's own Camera within its GameObject hierarchy? Drag it with your mouse from scene hierarchy to the property Camera To Activate On Local Player. In case of the demo character provided above leave this deactivated.
- A little bit beneath that there are the two options Synchronize Position and Synchronize Rotation. Enable them.
- If your character is animated also enable Synchronize Animations. A new attribute named Object Containing Animations appears. If your animations are stored on a child GameObject instead of the top one drag and drop it from scene in here. Otherwise leave it empty.
- You now told the uMMO character to synchronize transform values and animations across the network. What we didn't do is tell the engine how to synchronize those. uMMO comes with NetworkViewSerializer modules for this reason. Drag the prefab Assets/uMMO/prefabs/modules/NetObject/ModNetObject_NetworkInterpolatedTransform_NonAuthOrNPC and Assets/uMMO/prefabs/modules/NetObject/ModNetObject_SyncLegacyAnims onto NetworkViewSerializerMods one after the other. These modules contains logic to interpolate the characters position for smooth movement and synchronize animations automatically.
- Now the interesting part. We will now determine, which script should be enabled in what situation.

(NOTE: Situations we can encounter at the moment are:
→ The player character is instantiated in the game scene, which is being started as a server.
→ The player character is instantiated in the game scene, which is being started as a client.
Why? Always remember, that we design server AND client in the same game scene, to avoid having to synchronize data models and/or having the need for bridging infrastructures.)

- Enable Handle Scripts Based On Situation. Three empty lists appear: ClientLocalPlayerScripts, ClientRemotePlayerScripts, ServerPlayerScripts.
- Now, certainly your character contains a movement script. A script that picks up input from the Unity InputManager and based on which input was delivered by the user, moves/rotates the character. On the example character (link above) this would be the script ThirdPersonController. It is mostly the script where there are lines similar to this one:

```
verticalInput = Input.GetAxisRaw("Vertical");
```

→ Take this movement script (on your in-scene instantiated character) and drag it with the mouse and drop it directly on the property/list ClientLocalPlayerScripts. It will now ONLY be existing during game-play if this character is instantiated on the particular client who actively controls the character (as opposed to the server, or a client which will indeed also instantiate the character, but as a remote controlled character, meaning not "your" own).
→ Does the character GameObject contain the component Character Controller? Make sure to drop it here, too. (if not, ignore this step)
→ Does the character contain a separate script which controls which animation is being executed in which situation (On the example character this would be ThirdPersonSimpleAnimation)? Mostly a line like the following would be found in that kind of script:

```
animation.CrossFade("walk");
```

Just drop them on the same property/list. (if not, ignore this step)
→ Have some other scripts which should only be active on the local player who actively controls the character? Just drop them on the same property/list. (if not, ignore this step)

3. The character is ready to go! Now we have it to declare to the uMMO plugin the main player character it has become. Please follow the next steps closely:

- Save the player character which is now instantiated in your game scene into project hierarchy (just drag and drop it with the mouse into it). This makes a prefab off of our player character.
- Delete it from scene hierarchy. It should of course not be instantiated when the scene starts (but only when a player connects)!
- Click on the uMMO prefab instantiated in your scene.
- Drag and drop the player character prefab we just created into the property Player Char On Non Authoritative Setup.

4. Now we are all set. Lets prepare the rest of the scene so that it is presented correctly when we test it!

- For this test the character will be spawned from the position of the uMMO prefab instantiated in your scene. Make sure it is placed (a little) above the ground of your scene and make sure the scene camera has this position in good view.
- Choose File → Build & Run. When you haven't set up an export platform and a location for the resulting file(s), do so now in the appearing dialog. For this test please export to a Standalone sceneplayer (MAC/Windows).
- Now remember that in the very beginning of this guide, we set Architecture To Compile to TEST_UnityEditorIsServer_OthersAreClients. That means all exported instances of the scene will now be clients! The Unity Editor itself will be the central server.
- First start one client. Then(!) start the scene in the Unity Editor. You can now connect the

client to the server. For this simple example the player character will be instantiated immediately on established connection. If you set up everything correctly you should now be able to control the player character from the client. It's movement is instantaneously synchronized to the server.

- Let's now start another client and connect it to the same scene: (NOTE: Don't stop the Unity Editor scene execution or the running client just yet!) Navigate to the location on your hard disk where your let Unity store the export player. Start the executable by yourself. You now started a second client, connect it to the server, and you can see the two clients magically synchronize movements to one another!

Congratulations! ☺ You now have set up an MMO example game scene! You created a central server and connected two clients to it, each with its own movement script which only gets executed on the client which actually owns the player, yet you use the same prefab for all instances of the game character, which has a lot of advantages most of which you will only learn to appreciate later in the development process of your own MMO game!

3. Limitations/Hints

- At the moment, there is no in-built database support. You can thus choose your own connector and your own way of data persistence.
- At the moment there is no user authentication implemented. That (same as with databases-support) is because there are so many possibilities to implement those features, that we feel it would possibly be more of a limitation than an asset to the package. This doesn't mean, that there is no possibility that in the future, there won't be an upgrade to add those features, if we find the time and figure out a comfortable and especially a flexible way of integrating it.
- All movement-/other behavior scripts should be placed in the top GameObject in the player GameObject hierarchy (so preferably don't place scripts on children of this player GameObject. If necessary replace some scripts). That, in order to use the uMMO_NetObject class effectively. If the custom scripts and that particular component are added to the top GameObject of your prefab hierarchy you can drag all of those scripts and drop them on the appropriate property/list to have it enabled in the right situation. Please read more about this feature in the Quick Start Guide of this document.
- At the moment, only one game scene has been actively tested with, but that does not mean that uMMO doesn't support multiple scenes, quite the opposite: Multiple scenes are a few lines away from being totally compatible with uMMO. This is sure to be added to the plugin very soon.
- The technical scope of Unity in connection with network traffic is a tricky field. In theory you can connect as many clients to the server as you like, transmitting as much data you need to, only limited by your server capabilities: uMMO does not limit you in any way in that concern but also doesn't provide explicit guarantees to what can be accomplished technically.

There is a very good answer given in this thread: http://answers.unity3d.com/questions/8597/how-many-players-does-unitys-built-in-networking-s.html

It boils down to your individual setup which makes the biggest difference. Some suggest that Unity imposes limits on how much players/traffic is handable. On the other hand all possible technical limitations concerning the Unity platform have a good chance of being fixed while the Unity Team makes a great effort with every single new version of the IDE.

UPDATE: In mai 2014 Unity announced **UNET**. UNET will revolutionize network programming. This is *GREAT* news for uMMO, and of course we will do our very best to make uMMO compatible with UNET as it promises to improve workflow and performance with native Unity networking.

Read the following blog posts:

http://blogs.unity3d.com/2014/05/12/announcing-unet-new-unity-multiplayer-technology/

http://blogs.unity3d.com/2014/05/29/unet-syncvar/

http://blogs.unity3d.com/2014/06/11/all-about-the-unity-networking-transport-layer

Need support? http://www.softrare.eu/contact.html