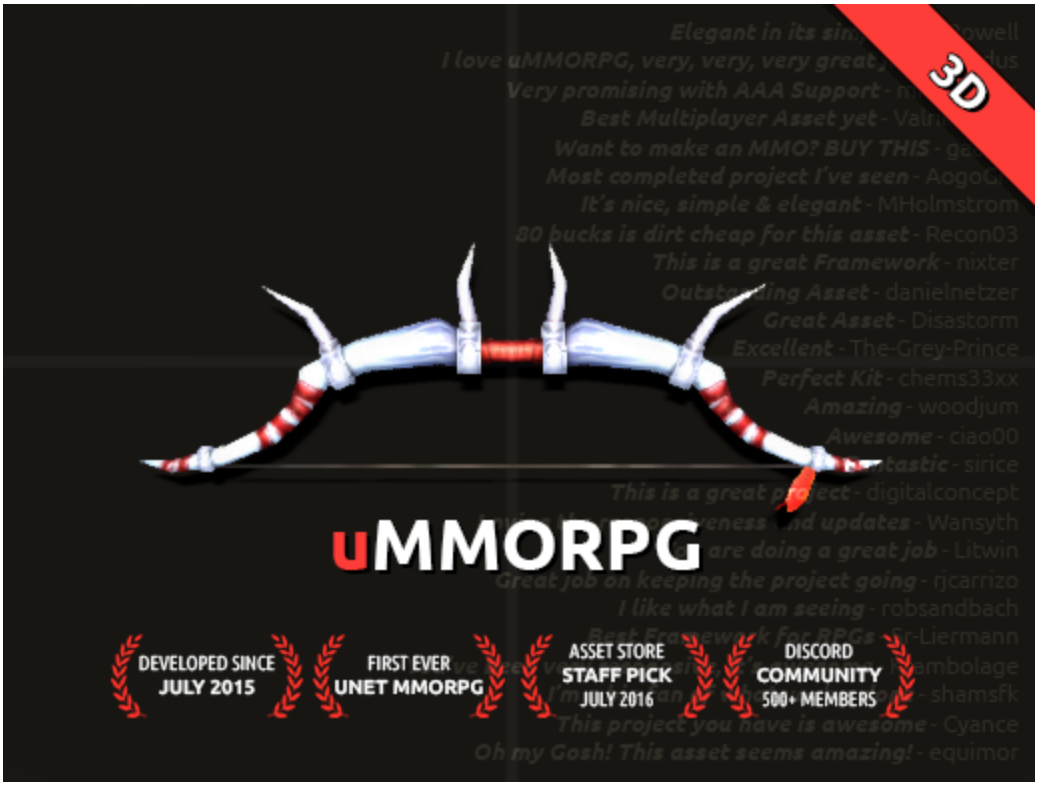
uMMORPG Documentation



(Asset Store) (uMMORPG.net) (Email) (Discord) (FAQ) (Forum) (WebGL)

Getting Started

To see **uMMORPG** in action, simply open the included scene from **uMMORPG/Scenes**, press **Play** in the Editor and select **Server & Play**. Run around a bit, kill some monsters, pick up some new items and try some quests.

Once you got a feeling for it, now it's time to get some basic project overview and learn some new stuff. Recommended first steps:

- Inspect the Prefabs/Entities/Players prefabs. Take a look at all the components to see how much you can modify easily, without writing any code.
- Look at the Resources/Items folder and play around with the items. You can modify stats and icons easily without writing any code. Duplicate the Banana and modify it to be an apple. Then select one of the Item Spawners in the Scene and assign it.
- Look at the Resources/Skills folder to see the different skill types and how to modify them.
- Inspect the Hierarchy/Scene. Add a Monster by duplicating one, then move it somewhere else. Move around some of the Environment. Go to

Window->Navigation and rebake the Navmesh afterwards. Monsters need it to move around.

- Read through the Mirror Documentation to understand the Networking system.
- Read the rest of this documentation to understand all the components.

Recommended Unity Version

Right now, the recommended Unity version is Unity 2019.4 LTS. You should not use an **older version** because Unity is not downwards compatible.

You can use **newer versions** at your own risk. uMMORPG is a big project that uses a lot of different Unity features, so whenever Unity introduces a new bug, we feel the effect very significantly. In theory, any newer Unity version should work fine if (and only if) Unity didn't introduce new bugs. That being said, it's common knowledge that Unity always introduces new bugs to new versions.

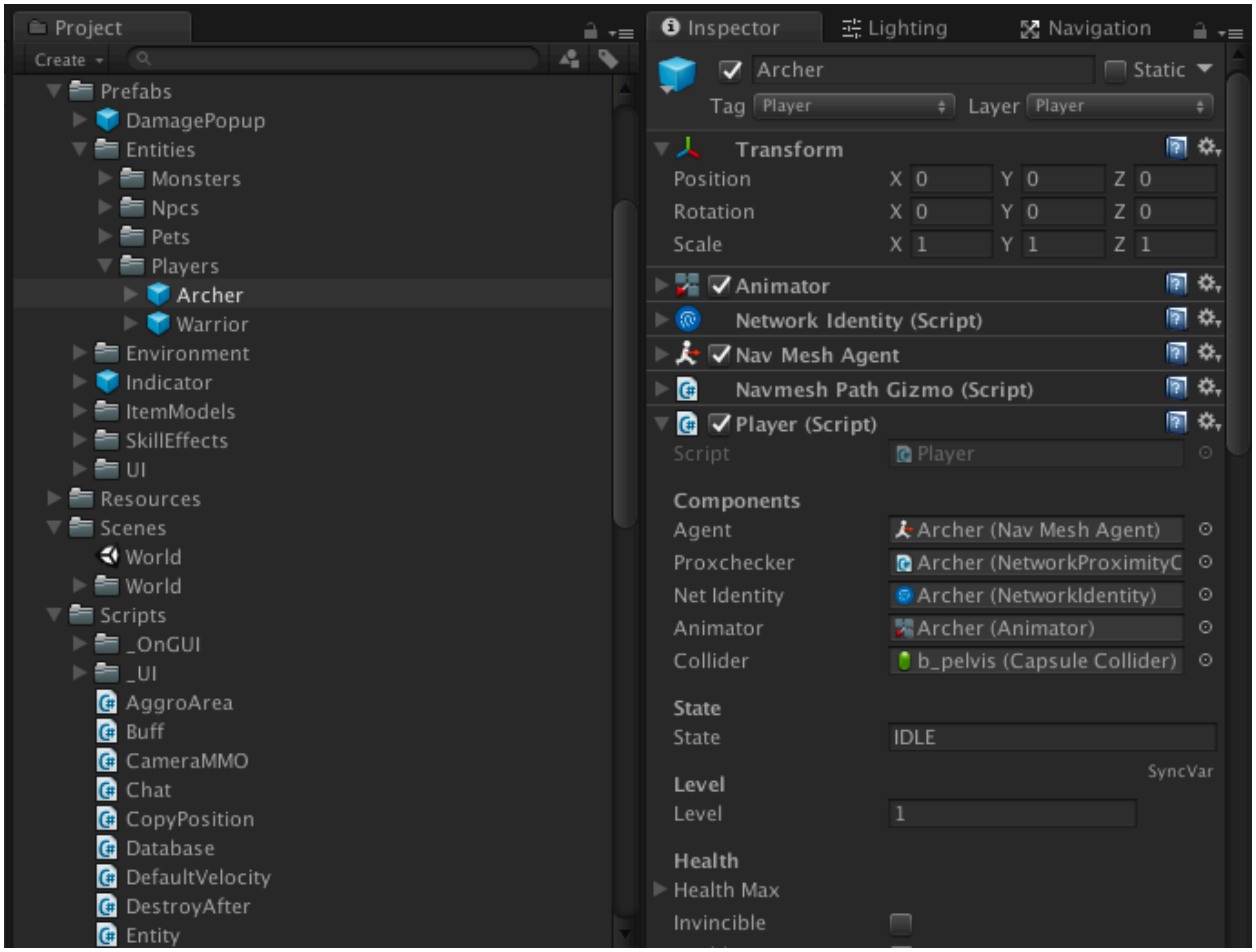
During the past few years working on uMMORPG, the process was to download Unity, encounter bugs, report them, upgrade Unity to the newer version where the bugs were fixed, only to encounter new bugs, and so on. It was a never ending cycle of upgrading headaches, with only the occasional stable version in between.

During GDC 2018, Unity announced the **LTS release cycle**. LTS stands for long term support. Unity LTS versions are supported for 2 years and will only receive bug fixes, without introducing any new features (and hence new bugs).

Words can hardly express how significant LTS versions are for a multiplayer game. You should absolutely use LTS at all times, otherwise your players will suffer from bugs and all hell will break loose.

Players

The Player prefab(s) can be found in the Prefabs/Entities/Players folder:



Players have several components attached to them, with the 'Player' component being the most important one.

You can modify a whole lot without writing any code, simply browse through the Player component in the Inspector to get a feel for it.

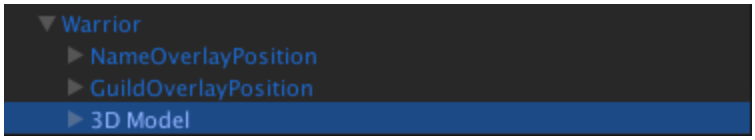
You can of course add your own components to player prefabs too.

Player Movement

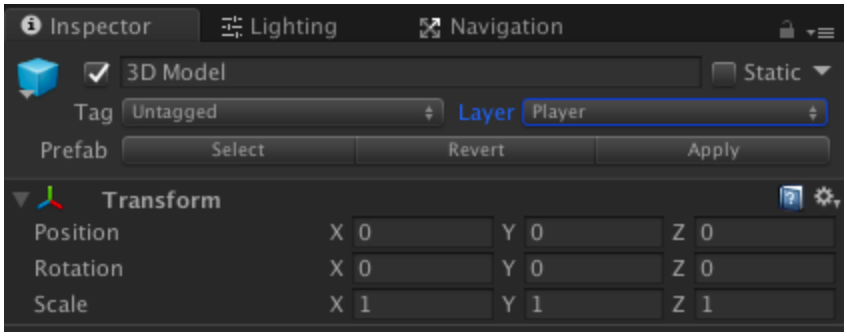
Players move on Unity's NavMesh with their NavMeshAgent component.

Modifying the Player Model

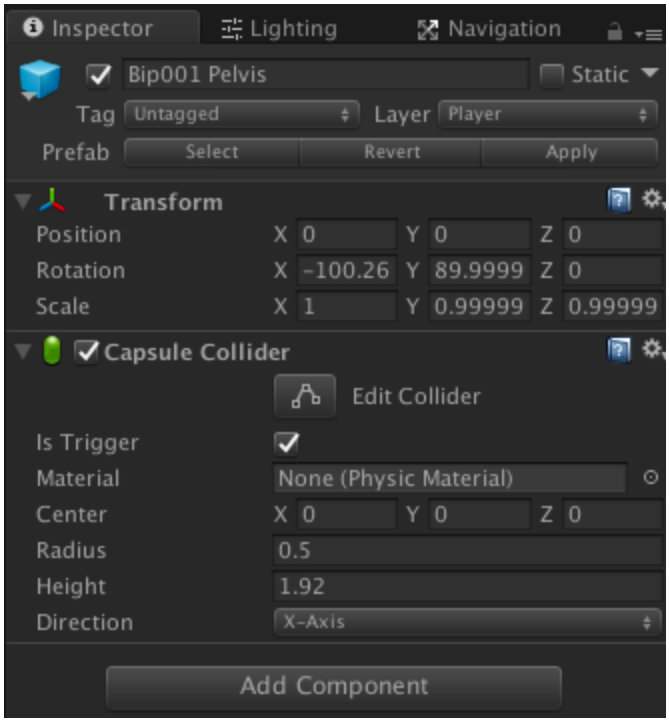
1. Drag the prefab into the Scene.
2. Replace the 3D Model child object with your new 3D Model



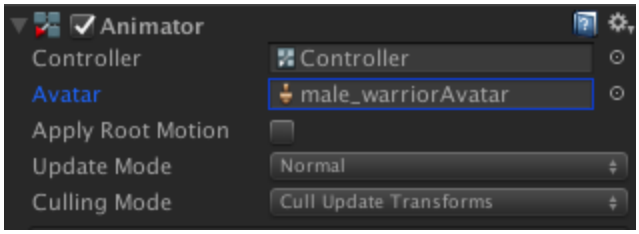
3. Set the 3D Model's Layer to Player again:



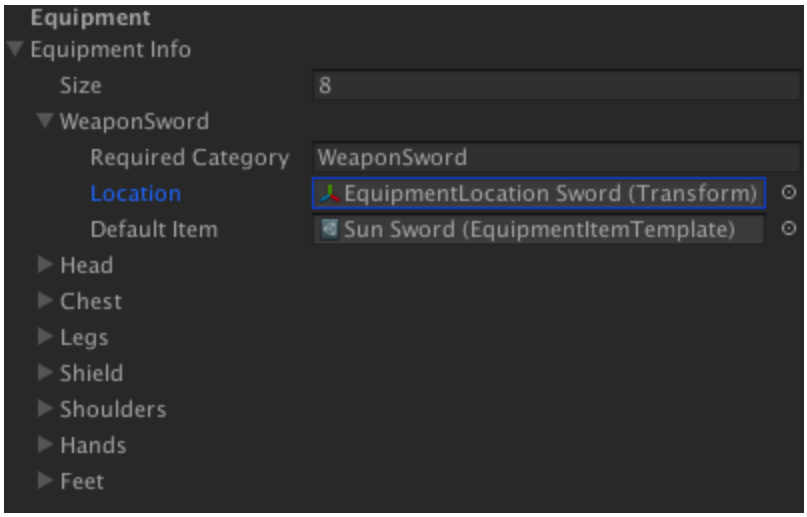
4. Add a fitting Capsule Collider to the hip bone, so that it follows the animation:



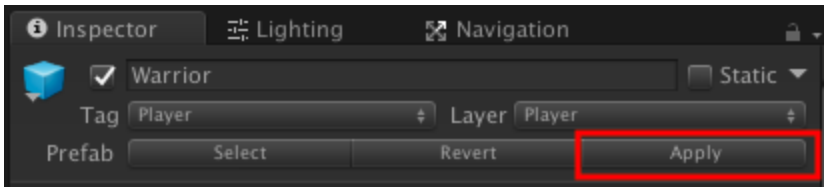
5. Assign the new model's Avatar in the Animator component:



6. Reassign Equipment Locations to the proper bones:



7. Apply the changes:



Creating a new Player Prefab

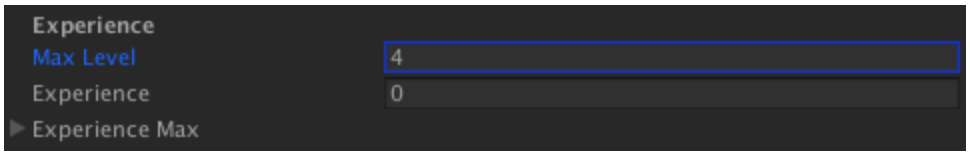
If you want to create another player type, simply drag the existing player prefab into the scene, modify it to your needs, rename it and then drag it back into the folder to save it as a different prefab.

Modifying the existing prefab step by step is always a lot easier than creating it all from scratch again.

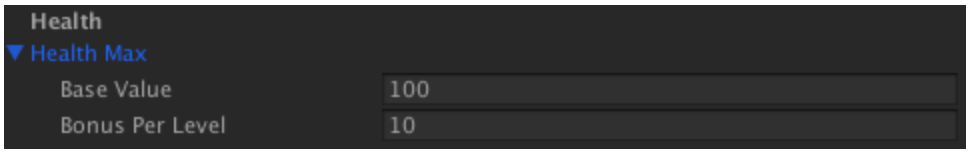
Make sure to also drag the new prefab into the NetworkManager's spawnable prefabs list. uMMORPG will then automatically display it as another option in the character creation menu.

Levels

Each player has a maximum level property that you can modify to set the level cap:



Player stats like health, mana, damage, etc. are level based. They start with a base value and add a bonus for each level:



For example, a level 1 player has 100 health. A level 2 player has 100+10=110 health. A level 3 player has 100+20 =120 health.

Level based stats are a great solution, because they scale with any level cap. If you decide to raise the level cap from 60 to 70 in your game, then all you have to do is modify the MaxLevel property. Health, mana, etc. will scale to 70 automatically.

Parties

uMMORPG comes with a party feature, which allows players to form groups and kill monsters together. Being in a party grants players an experience boost and allows them to share experience among each other.

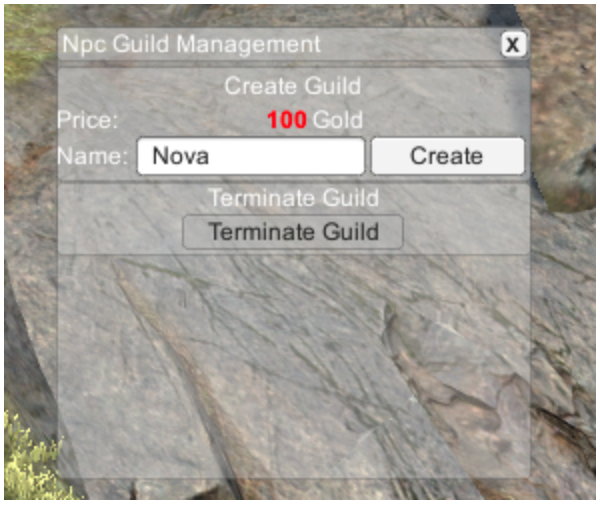
A player can invite another player to a party by standing close to the other player, targeting him and then pressing the Party Invite button in the Target UI:

The party owner can then modify the party settings in the Party UI window.

Guilds

uMMORPG comes with a guild feature.

A player can create a guild by talking to the Npc:

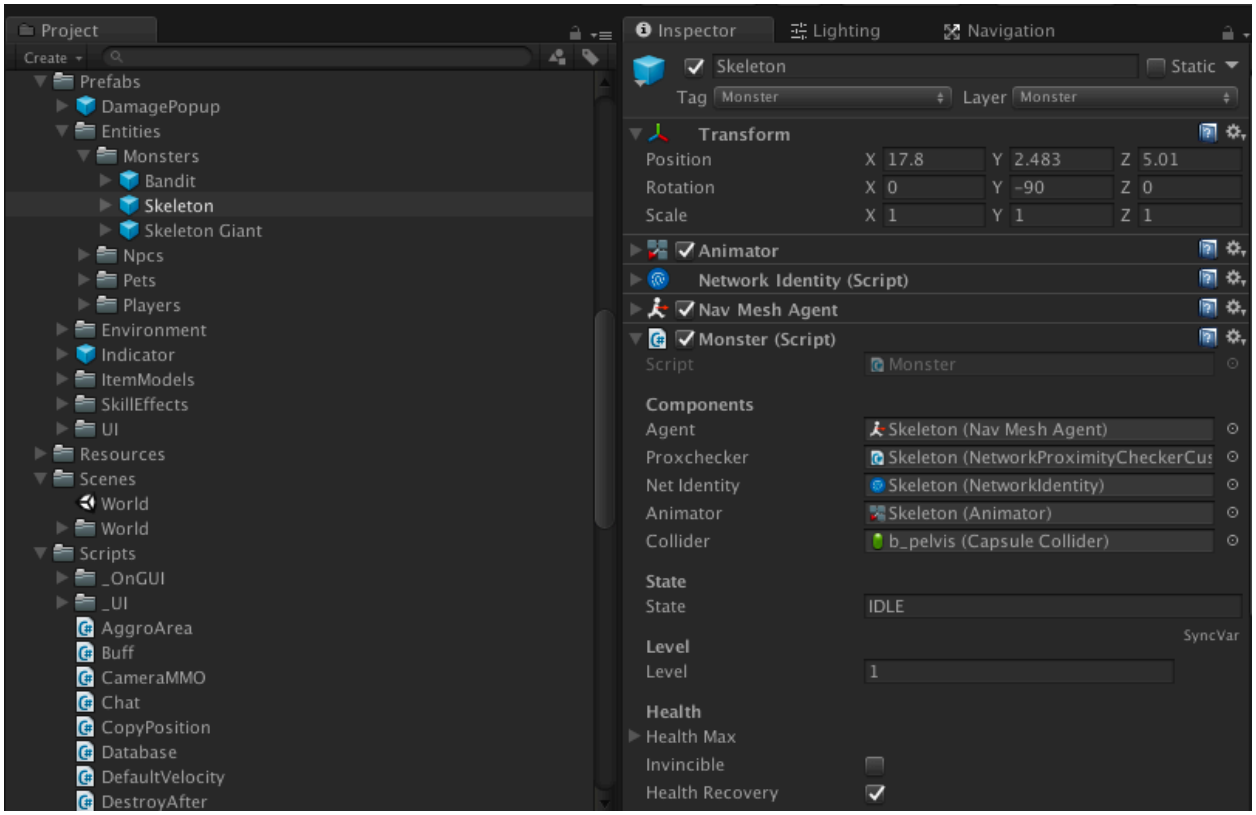


A player can invite another player to a guild by standing close to the other player, targeting him and then pressing the Guild Invite button in the Target UI:

The guild master can then modify the guild settings in the Guild UI window.

Monsters

The Monsters can be found in Prefabs/Entities/Monsters:



You can modify it to your needs, just like the Player prefab.

There is no complicated spawning system for monsters. Simply drag them into the scene and position them wherever you want them to live.

Monster Movement

Monsters have a simple AI and they need to be able to navigate around obstacles, for example to follow a player into a building. This is very difficult to do, but thanks to Unity's Navigation feature, all we have to do is set the Monster's NavMeshAgent.destination property.

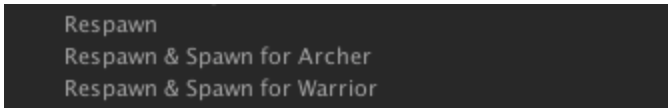
So in other words, Monsters are NavMeshAgents on Unity's Navmesh. Make sure to rebake the Navmesh whenever you modify the game world.

Monster Behaviour

Monsters are driven by a Finite State Machine(FSM) in the Monster.cs script. The behaviour is 100% code, so you'll have to modify the script if you want to change it.

Spawn Position

uMMORPG has Spawn and Respawn positions:



Spawn positions are where the player classes spawn after creating their character.

Respawn positions are where the player classes respawn after death.

You can add more Respawn positions by duplicating the existing one, afterwards move it wherever you want. A player always respawns at the closest respawn position.

Modifying the Environment & Terrain

uMMORPG uses a small and simple scene to display all of the features. You can of course add any 3D models or environment assets that you like, as well as Terrains. Simply make sure that:

- The GameObjects are marked as Static
- Rebake the NavMeshSurface component's NavMesh afterwards. You can find it on the Environment GameObject. The Monsters and Players need the Navmesh to move.

Skills

Adding a new Skill is very easy. We can right click the Resources/Skills folder in the Project Area (as seen above) and select **Create**->**uMMORPG Skill** to create one of several existing skill types. Now we rename it to something like “Strong Attack”. Afterwards we can select it in the Project Area and the modify the skill’s properties in the Inspector. It’s usually a good idea to find a similar existing skill in the same folder and then copy the properties.

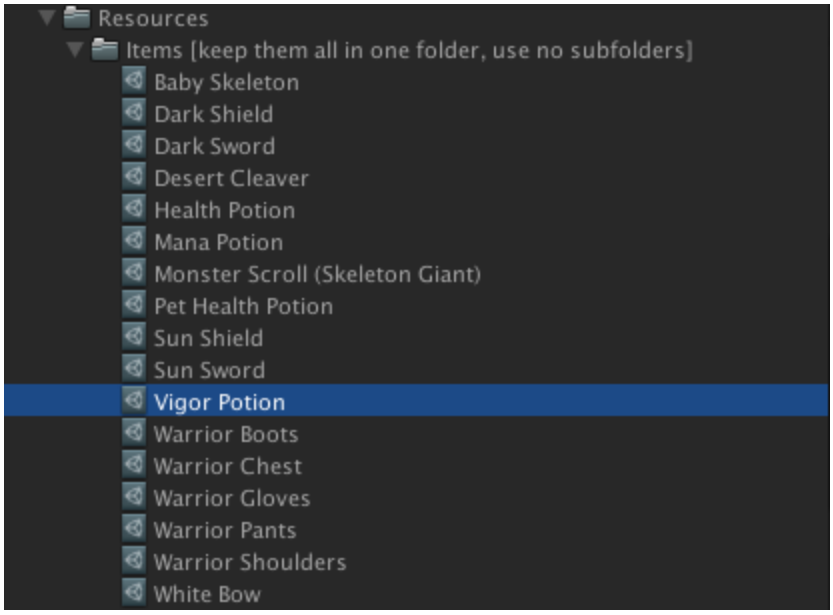
Afterwards we select the player prefab and then drag our new skill into the **Skill Templates** list to make sure that the player can learn it.

Note: uMMORPG uses scriptable skills and you can create any new skill type if necessary. Take a look at Scripts/ScriptableSkills to learn how the existing skills were implemented.

Items

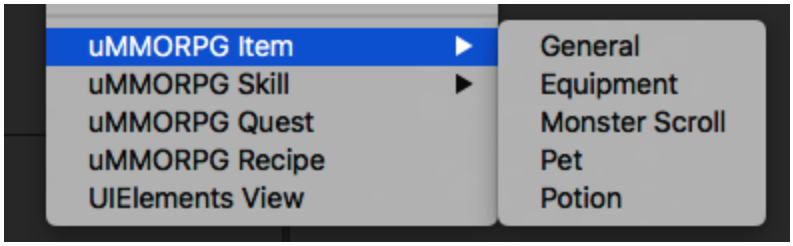
Scriptable Items

We already added the most basic items types, so you can simply duplicate existing items to make similar ones:



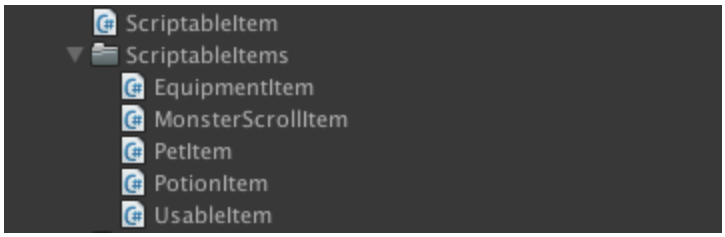
- You can duplicate and modify the Health Potion to make a Strong Health Potion.
- You can duplicate and modify the White Bow to make a Crossbow.
- You can duplicate and modify the Monster Scroll to make one that spawns a different Monster.
- Etc.

You can also create new Items by right clicking in the Project area to select Create->uMMORPG Item:



As you can see, there are already different item usage mechanisms. Potions like the Health Potion will be consumed to increase health and mana. Other items like the White Bow do

none of that, but shoot arrows instead. Every MMORPG will need all kinds of different item usage mechanisms, which is why we implemented **Scriptable Items**:



Please take look at the **Scripts/ScriptableItem.cs** file and the **Scripts/ScriptableItems** folder with the currently implemented Scriptable Item types. In most cases, you will want to inherit from UsableItem.cs. All you have to do is overwrite the **.Use()** function (and a few others depending on your needs) to add any item mechanism that you want. You could have an item that kills every Monster on the server in .Use() or levels up all guild members. The possibilities are endless.

To create a new Scriptable Item type, simply create a new Script, inherit from ItemTemplate (or UsableItem if it's supposed to be usable) and then add your logic / properties as needed. Make sure to add a menu entry like this:

```
using UnityEngine;

[CreateAssetMenu(menuName="uMMORPG Item/Monster Scroll")]
public class MyUsableItem : UsableItem
{
}
```

So that you can create an item of that type via right click.

The Item Struct

We just talked about **ScriptableItem.cs**, which is the true 'Item' class in uMMORPG. But there is also **Item.cs** - *what the hell?*

uMMORPG uses UNET's SyncListStructs for the inventory and the equipment. Those SyncListStructs only work with structs, so we can't put ItemTemplate types in there. And that's a good thing, here is why:

- ItemTemplate has huge amounts of item data like the name, icons, reload times, etc.
- Syncing all that over the Network would require large amounts of bandwidth
- Clients already know the ItemTemplates anyway - there's no point in syncing them again. All they need to know is which Scriptable Item they need to refer too.
- The Item.cs struct is just that. It contains the name (a hash to be exact) to refer to the ItemTemplate. So all we need are a couple of bytes and the clients know which ItemTemplate is in which inventory slot, etc.

Note that all 'dynamic' Item properties like the pet's level are also in Item.cs - since two Items of the same ItemTemplate type might as well have different pet levels.

ItemSlot

There is also the ItemSlot struct, which is just Item + amount. No magic here.

The Inventory and Equipment SyncListStructs work with ItemSlots.

A slot contains a valid .item if the .amount > 0. If .amount == 0 then the .item is invalid and should not be accessed.

Item Models

Oh come on.. enough with the Item classes now!

Okay, last one. If the player equips a Sword, we should see an Sword model in the player's hands. That's why we need Item Models. They are really just 3D models this time, nothing complicated.

Combining them all

Let's say you want to add a Strong Health Potion to your game. Here is the step by step guide:

1. Duplicate the HealthPotion ItemTemplate, rename it to Strong Health Potion and give it an new icon and an increased health reward.
2. Now add it to the game world, for example:
    a. To a Monster's **Drop Chances** list
    b. To an Npc's **Sale Items** list
    c. To a Player's **Default Items** list
    d. To a Player's **Default Equipment** list

Now press Play, find your item and use it!

For weapons/equipment that the player can hold in his hand, you will also have to create the ItemModel (as usual, duplicate an existing one and modify it), and then assign the ItemTemplates's modelPrefab field too.

Equipment

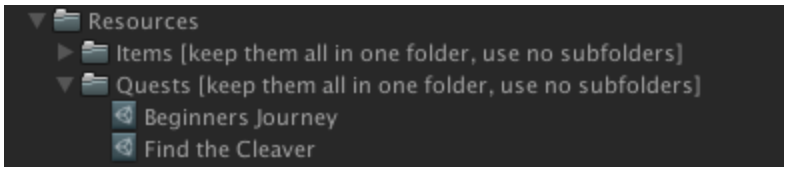
For static equipment like swords and shields, all we have to do is create the 3D model and use it in an item's model prefab.

For animated equipment like pants that should follow the character's movement, the equipment model needs to be rigged and animated.

After adding the model to the project folder in Unity it's also important to add an Animator component to the prefab. uMMORPG will then automatically copy the character's controller into the equipment prefab's Animator controller slot and pass all the same parameters to it.

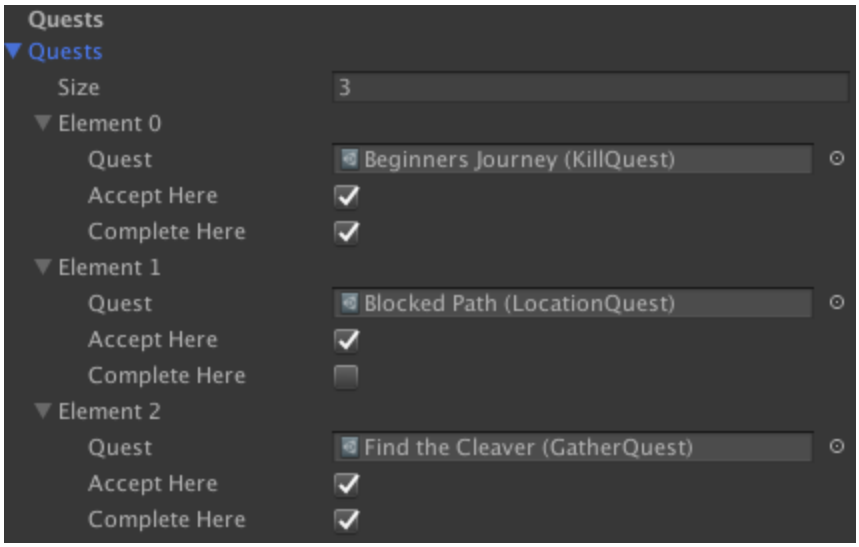
Quests

uMMORPG comes with a scriptable quest system. Quests can be found in the Resources/Quests folder:



To add more quests, simply duplicate an existing one and modify it to your needs, or inherit from ScriptableQuest to add custom requirements.

Afterwards drag it into an Npc's available quests list:

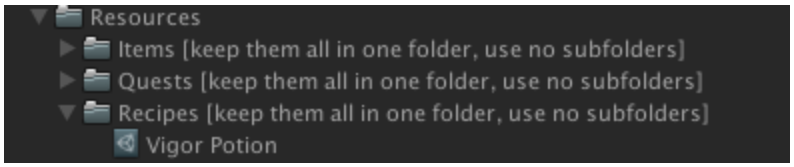


The quests list has **Accept Here** and **Complete Here** properties that should be enabled by default for the quest to be acceptable and completable at this npc.

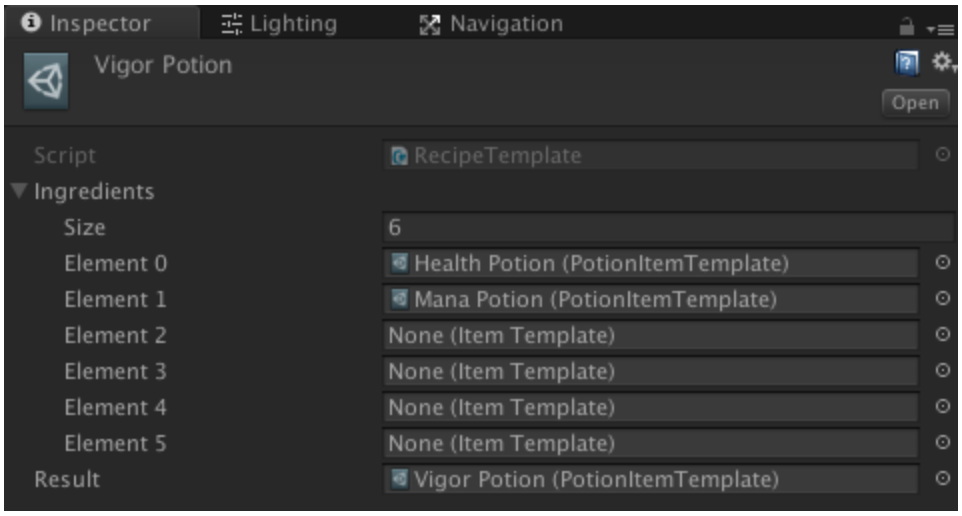
Complete Here can be disabled if the quest should be completed by talking to a different Npc, in which case the other Npc has the same quest entry but with **Complete Here** enabled and **Accept Here** disabled. For an example, please try the **Blocked Path** quest ingame.

Crafting

Crafting recipes can be found in the Resources/Recipes folder:



Recipes are very simple. They have a list with ingredients and a result item:



You can craft them ingame by dragging the ingredients into the crafting slots, afterwards the item that can be crafted will appear:

This system also allows for real recipe items. For example, the Vigor Potion could also require a secret scroll that players need to find in order to craft it. The scroll can simply be added as one of the ingredients.

Scenes

First of all, we have to understand that the game server can only handle one Scene right now, so our whole game world should be in that Scene. If you want to replace the current Scene, you can either just build on top of it or duplicate it and then modify what you want to modify.

Note: I created an experimental(!) NetworkZones addon that allows us to use one different scene per server process, with portals to move between them. The addon is pinned in our Discord server's verified channel.

The Database

About SQLITE

uMMORPG uses SQLITE for the database. SQLITE is like MySQL, but all stored in a single file. There's no need for a database server or any setup at all, uMMORPG automatically creates the Database.sqlite file when the server is started.

SQLITE and MySQL are very similar, so you could modify the Database.cs script to work with MYSQL if needed.

Note that SQLITE is more than capable though. Read the SQLITE Wikipedia entry to understand why.

How to view and modify the Database

The database can be found in the project folder, it has the name Database.sqlite.

The database structure is very simple, it contains a few tables for characters and accounts. They can be modified with just about any SQLite browser and we listed several good options in the comments of the Database script (e.g. SqliteBrowser).

Characters can be moved to a different account by simply modifying their 'account' property in the database.

A character can be deleted by setting the 'deleted' property to 1 and can be restored by setting it to 0 again. This is very useful in case someone accidentally deleted their character.

The User Interface (UI)

uSurvival uses Unity's new UI system. Please read through the UI manual first:

https://docs.unity3d.com/Manual/UISystem.html

Modifying the UI is very easy. Just modify it in the Scene in 2D view.

Most of the UI elements have UI components attached to them. There is no magic here, they usually just find the local player and display his stats in a UI element.

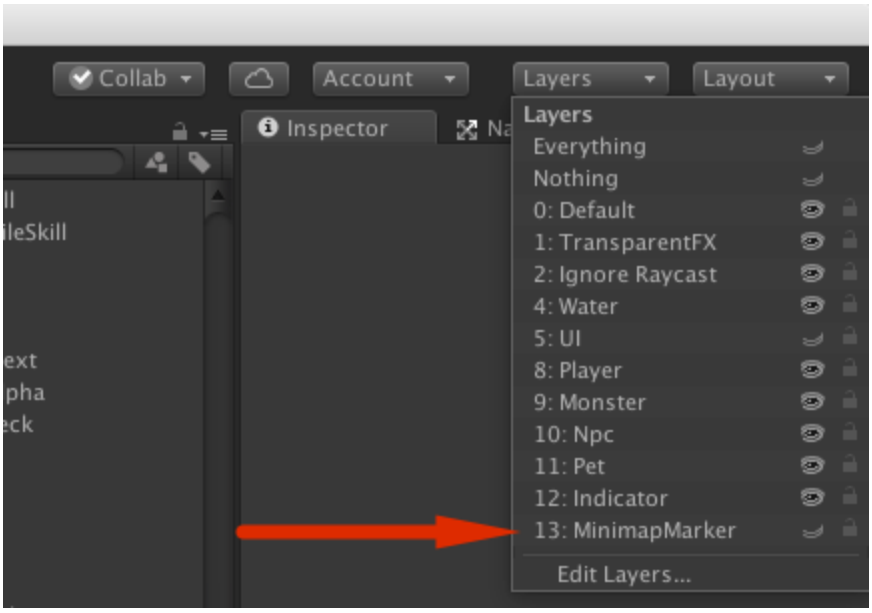
Feel free to modify all the UI to your needs.

The Minimap

The Minimap can be found at the top right of screen:

The MinimapCamera in the scene is placed high above the world. It renders it into a separate RenderTexture in the Textures folder, which is then displayed in the Minimap Canvas element.

Minimap Icons are simply colored cylinders around the 3D objects that are only shown in the Minimap camera. The materials/colors can be changed easily.

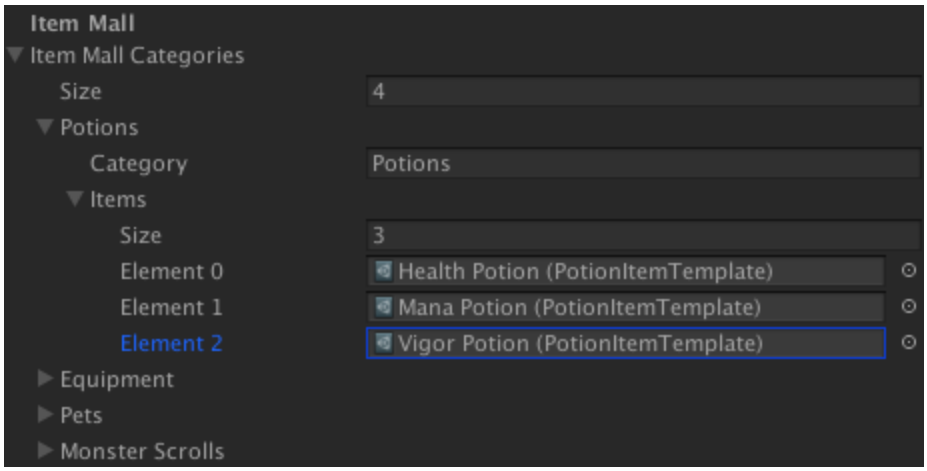


If you don't want to see those huge cylinders in the Unity Editor scene view all the time, you can hide the MinimapMarker layer in the Editor.

Item Mall

uMMORPG has a built in Item Mall that can be used to sell items for real world money. The system is very simple to use:

1. Set an item's **ItemMallPrice** to a value > 0
2. Drag it into the Player's ItemMallCategories property:



3. Select the Canvas/ItemMall window and change the **Buy Url** to the website where your users should buy item mall coins
4. Use your payment callback to add new orders to the database character_orders table. Orders are processed automatically while the player is logged in.

   Note: you can also process new orders manually and add them to the character_orders table by hand.

5. The Item Mall also has a coupons feature. Coupons are validated in Player.CmdEnterCoupon. You can use whatever algorithm you like to validate a coupon and then reward the player with item mall coins.

Instanced Dungeons

uMMORPG supports instanced dungeons, where upon entering a portal to a dungeon, the server creates an instance of that dungeon which can only be accessed by that particular party.

The Common Approach

There are several different ways to implement instanced dungeons. The **most common way** is to run **one instance per server process.**

- The **benefit** of this approach is that the load can be split between multiple server processes.
- The **downside** of this approach is complexity. It's very difficult to get right for a whole lot of reasons.
    - The server processes need to be able to communicate with each other for chat across zones, guild invites across zones, guild/party status across zones, and so on.
    - We need multiple physical servers per world. Who wants to maintain, secure and update all of them?
    - Players need to be transferred to other server processes safely.
    - Maintaining multiple server processes is way more difficult than maintaining one server process.
    - It needs to be cheat safe so that people can't log into two processes at the same time.
    - It would never work in the Unity Editor. Only in builds, because we can't spawn another editor instance and tell it which scene to run. So when using uMMORPG for the first time, you would not be able to test all the features without setting up zone servers.
    - It would require a master server to handle state and communication across all the server processes. This is not impossible to do, but uMMORPG would be a lot more difficult to use if people would have to worry about where they host their master server, how to host their master server, and if the master server is still running or not.

The Simple Approach

The most common solution is not always the best solution. Let's step back a little and think about instances from scratch. **What are instances really?**

- A bunch of players are teleported to a special place.
  - **=>** This doesn't require separate processes.
- A bunch of monsters are spawned at that place.
  - **=>** The server can handle thousands of monsters already. That doesn't require a separate process either.
- An environment(dungeon) is spawned at that place.
  - **=>** The server doesn't care about the graphics at all. All we need is an extra NavMesh at that place. That doesn't require a process either.

Separate processes make sense for World of Warcraft and AAA studios with million dollar budgets that can support master servers, separate processes and lots of dedicated servers.

For Indie MMOs, we can't afford any of that. As usual, we decided to use the most simple solution in uMMORPG: **z-stacking** instances in the main world.

Here is how it works:

1. The player walks into a portal
2. The server instantiates the Dungeon template for that party and puts it next to other instances, offset on the z-axis
3. The party proceeds through the Dungeon.
4. Upon leaving, the server destroys the instance again.

But what about performance?

We explained it in the previous chapter already, but let's make this perfectly clear:

- The server still handles the same amount of players, they are just in different locations.
- The server does not care about rendering or 3D models of the dungeon.
- The only overhead are a couple of more monsters. While the monsters on the main map have to worry about less people, which is not bad at all.

Why z-stacking and not y-stacking?

1. If we were to stack instances on top of each other then the giant stack of instances might be visible from the main world.
2. If we use z-stacking then our NetworkProximityGridChecker needs no modifications. y-stacking would require a 3D grid checker.

Creating a new Instanced Dungeon

Using the **z-stacking** approach allows for extremely easy to use instanced dungeons:

1. Place a PortalToInstance from the prefabs folder into the scene.
2. Create the Dungeon template somewhere next to the scene. Take a look at the existing Dungeon, it requires 4 main parts:
    a. A 3D model of a Dungeon
    b. InstanceSpawnPoints to let the dungeon know where to spawn the monsters. The spawn points need to be children of the Dungeon GameObject. It will detect them automatically.
    c. The Dungeon GameObject needs an Instance component
    d. The Dungeon GameObject needs a NavMeshSurface component. It has a 'Bake' button that you can use to rebake the NavMesh.

NavMeshSurfaces are part of Unity's new NavMeshComponents package on Github. They allow us to copy and move NavMeshes at runtime.

*Note: in the uMMORPG example scene, the Dungeon template lives in the Scene. It is **not a prefab** because the initial template is used to keep track of all instances of that type.*

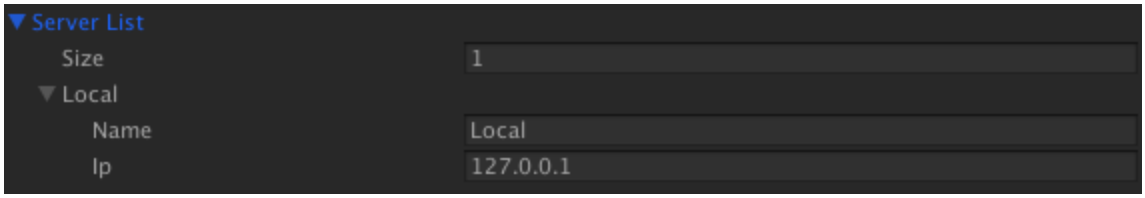
Testing a new Instanced Dungeon

To test an instanced dungeon, simply form a party and move into the portal.

Server Hosting

The Server List

uMMORPG's NetworkManager has a Server List:



The default entry is for a local server on your own computer, so you can test multiplayer easily. If you want to host uMMORPG on the internet, you can add another entry here with some name and the server's IP. Players can then select it in the Login screen:

Building the Server Binary

UNET puts the server and the client into one project by default, so any build can run as client or server as necessary. There is no special build process needed.

It's obviously a bad idea to host the server on a mobile device or in WebGL of course, it should be a standalone platform like Windows/Mac/Linux with some decent hardware.

The recommended server platform is Linux. Unity can create a headless build of your game there, so that no rendering happens at all. This is great for performance.

Setting up a Server Machine

Linux is the recommended Server system. If you have no idea how to get started hosting a UNET game on a Linux system or where and which one to even rent, then please go through my UNET Server hosting tutorial for a step by step guide and then continue reading here.

If you already know your way around the Terminal, then use the following commands:

Upload Headless build to home directory:

```
scp /path/to/headless.zip root@1.2.3.4:~/headless.zip
```

Login via ssh:

```
ssh root@1.2.3.4
```

Install 32 bit support (just in case), sqlite, unzip:

```
sudo dpkg --add-architecture i386
apt-get update
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
sudo apt-get install libsqlite3-0:i386
sudo apt-get install unzip
```

Unzip the headless build:

```
unzip headless.zip
```

Run the server with Log messages shown in the terminal:

```
./uMMORPG.x86_64 -logfile /dev/stdout
```

Hiding Server code from the Client

The whole point of UNET was to have all the server and client source code in one project. This seems counter-intuitive at first, but that's the part that saves us years of work. Where we previously needed 50.000 lines of code or more, we only need 5000 lines now because the server and the client share 90% of it.

The 10% of the code that are only for the server are mostly commands. Reverse engineering the client could make this code visible to interested eyes, and some developers are concerned about that.

The first thing to keep in mind is that this does not matter as long as the server validates all the input correctly. Seeing the source code of a command doesn't make hacking that much

easier, since for most commands it's pretty obvious what the code would do anyway. For example, the Linux operating system is very secure, even though it's code is fully open source.

Furthermore it's usually a good idea for game developers to spend all of their efforts on the gameplay to make the best game possible, without worrying about DRM or code obfuscation too much. That being said, if you still want to hide some of the server's code from clients, you could wrap your [Command]s like this:

```
[Command]
void CmdTeleport(Vector3 position)
{
    #if SERVER
    ... your teleport code here
    #end
}
```

And then #define SERVER in your code before building your server.

Addon System

uMMORPG comes with a very powerful Addon system that allows you to add functionality without modifying the core files. It's also useful to share features with the community.

An example addon script can be found in the Addons folder:

The example script shows you which functions you can use and which classes you can extend.

Updating

There are a lot more features and improvements planned for uMMORPG. If you want to update your local version to the latest version, you should keep a few things in mind:

- Always make backups before updating. In fact, you should make backups at least daily so that you can go back through your changes if things go wrong.
- uMMORPG comes with 100% of the source code. Updating would be effortless if we would just ship a DLL file that you can't modify, but we decided against that. You get all the code so you can modify it if needed, but always remember: with great power comes great responsibility. If you modify core code and it stops working or it's not compatible with an update, then you have to fix it yourself.
- Try using the Addon system as much as possible, so you don't have to touch core code.
- You should not update forever. At some point in your development it's smart to stick with one version and only manually apply bug fixes afterwards. For example, Valve didn't continuously update Counter Strike 1.6 - at some point they only applied fixes while starting to work on Counter Strike Source with their newest engine.
- A local Git repository is a great tool to keep track of your own code changes and of uMMORPG code changes. Maybe Git Branches are a good idea for you.

About the next generation of uMMORPG

One of the most frequent questions is: **"I use uMMORPG 2D/Classic, should I wait for/switch to Components Edition before making my game?"**

Imho, the answer is no.

If you are already working on your game, keep working on your game. I can't stress this enough.

It's **not** about 1000 CCU now vs. 2000 CCU if you start from scratch.

It **is** about releasing your game to the world.

The only thing you should be concerned about is having a fun game with a lot of players.

There is **value** in working with established technology instead of being on the cutting edge.

In fact, for MMOs that's basically a requirement.

If you are **one Unity version behind** and one **Asset Architecture behind**, then you get all the perks of stability & free bug fixes without breaking features. This is huge.

Think about it this way: you STILL get huge architecture improvements through Mirror, in fact it's likely that by the time you release your game, through Mirror & Hardware improvements alone you will get double the CCU that you get today. That should be enough.

And when it comes to new Assets/Architectures: just use it for your next game. If you release your first game in 2 years, you can still use the 2nd gen architecture afterwards. By that time, the 3rd gen architecture is probably in beta, and the 2nd gen is in stable LTS mode, which is perfect for **your next game**.

If you are already working on a game, just keep working on it!

Networking Tests (CCU Benchmarks)

Understanding CCU Cases

Before get started, let's talk about MMORPG's CCU numbers. It's important to understand the difference between worst case and real case CCU:

- **Worst Case:** all players standing next to each other. Every player broadcasting to every other player, resulting in insane amounts of bandwidth.
- **Real Case:** all players spread across a big world. Every player broadcasting to maybe 1-20 other players, resulting in way less bandwidth and CPU usage than the worst case.

Real Case CCU should be about 2-5x of the **Worst Case** CCU.

If World of Warcraft can handle 2000 players, then that's 2000 players spread across the world. Even World of Warcraft can't handle 2000 players all standing right next to each other.

Understanding Zones

Big MMORPGs like World of Warcraft don't actually hold all 2000 players in one server process. The game world is split between smaller scenes, each scene running on a different server.

So if World of Warcraft can handle 2000 CCU, that means that i can handle maybe 500 per Server **if** the game world is split between 4 servers.

In other words, 500 CCU per server for the **real case** or 200 CCU per server for the **worst case** is a very good goal to shoot for.

Test Results

We did a **worst case** test because we really want to see how much the server can handle if it comes down to it.

In other words, multiply our CCU result with 2x-5x and you can estimate the **real case** CCU.

Date: August 25, 2018

Players (CCU): 207

Result: **Success!** No Networking errors. No packet loss. No disconnects. Some Database lag.

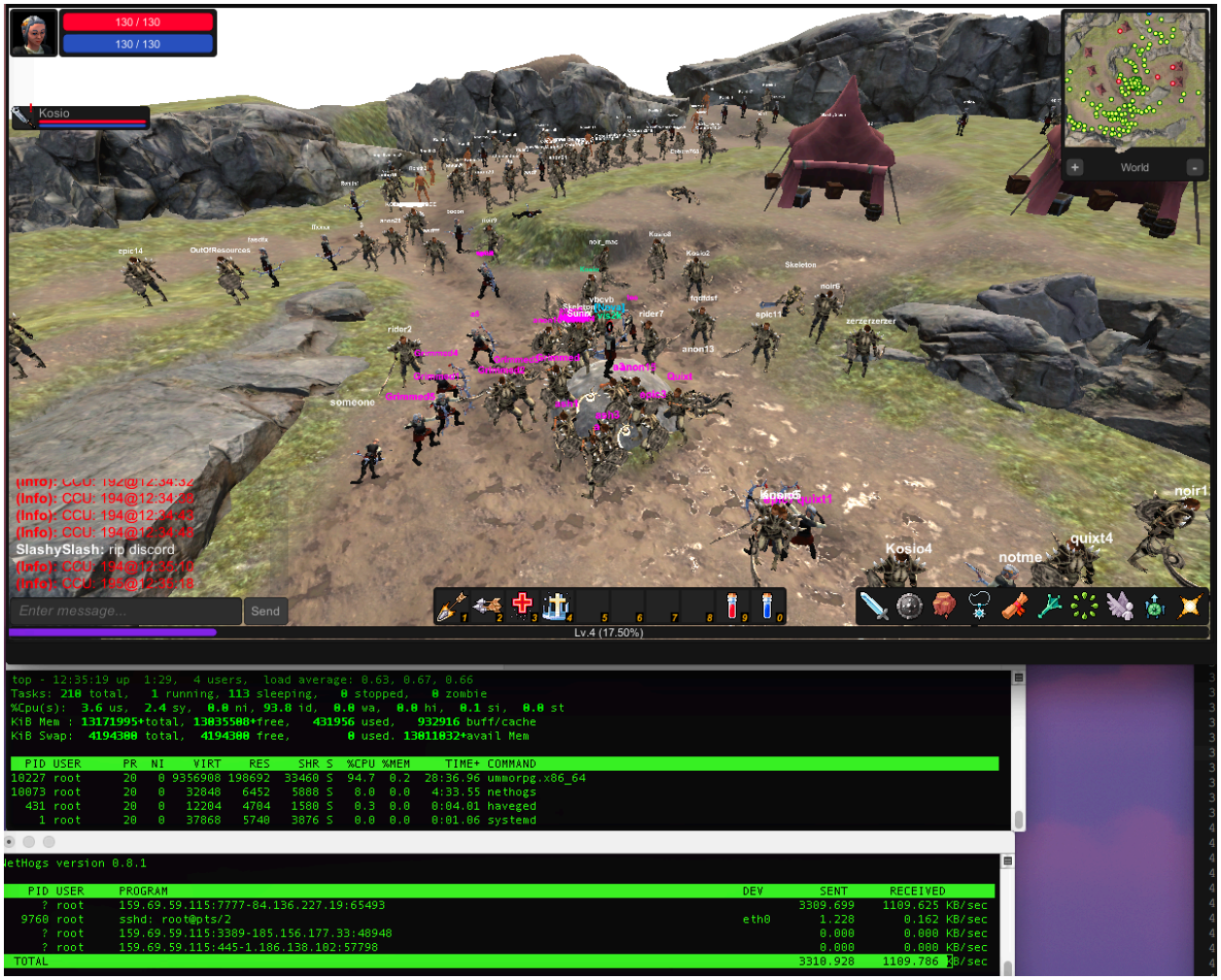
Notes: we forgot to modify the server's database save interval. It was still set to 30 seconds from the demo and for 200 players, it did cause some lag every 30 seconds then. In a future test we will set it to something more realistic like 5 minutes, and improve database performance even further (e.g. by using MySQL instead of SQLite).

Video: https://youtu.be/Utl-dYShJAY

Screenshots: *(right click and open images in new Tab for details)*

201 Players ingame:

195 Players CPU, RAM, Bandwidth usage:



The Server Machine

We rented the PX92 dedicated server from Hetzner for 105€/month with the following specs:

CPU	RAM	HARD DRIVE	OPERATING SYSTEM
Xeon W-2145 8 Cores x 3.7 GHz	128 GB DDR3	240 GB SSD, 6GB/s	Ubuntu 16.04 LTS x86_64

Many MMORPGs run on $400+/month servers. We will test on one of those next time.

The Software Stack

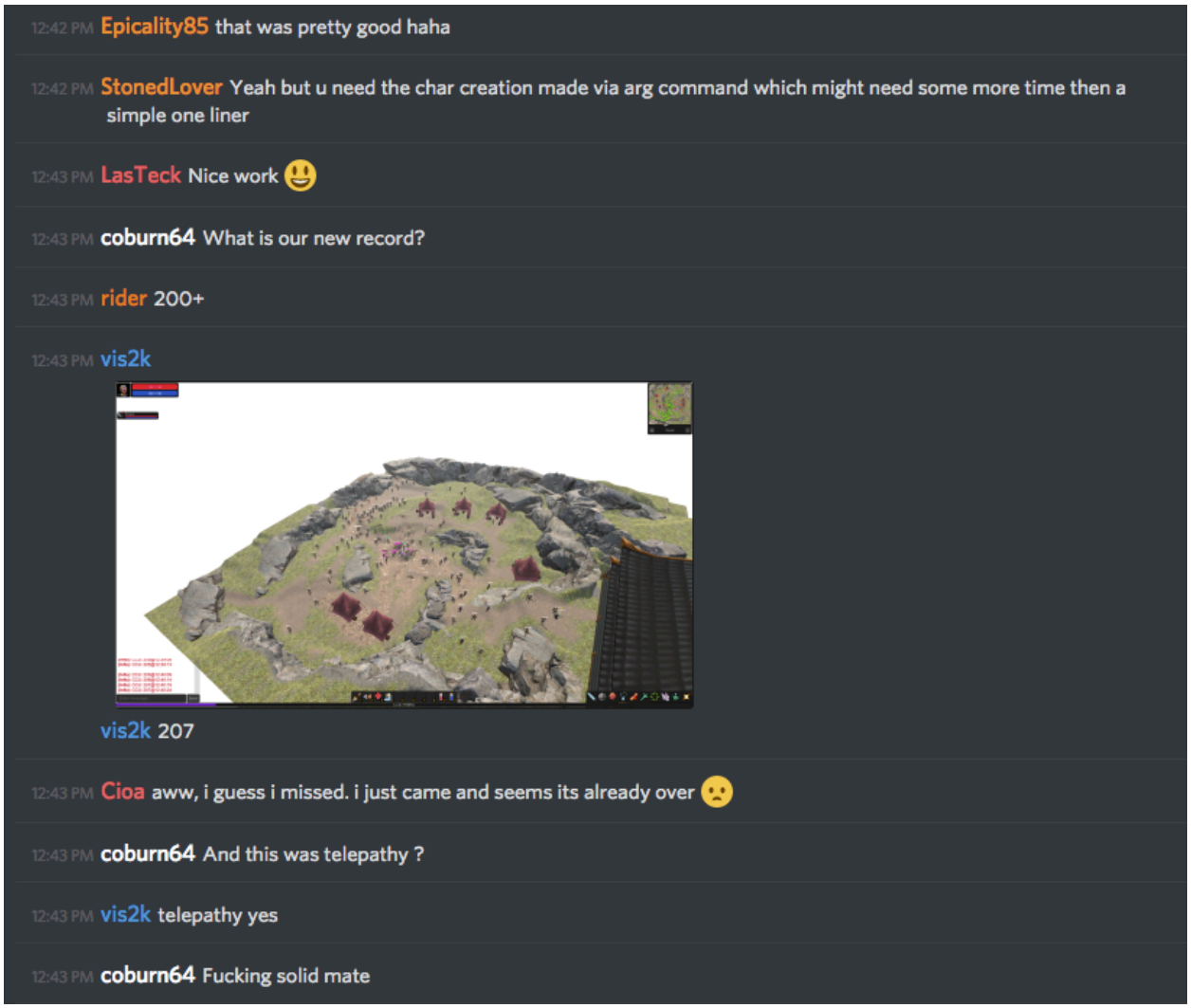
uMMORPG	Unity Version	High Level Networking	Low Level Networking
V1.129	2017.4.7f1 LTS	HLAPI CE	Telepathy

Witnesses

We always test CCU with our community, so that there are witnesses for the test results.

If you don't believe the test results, feel free to come by our Discord server and ask any of the participants:

`vis2k, DavidDeMo, rider, Ronith, coburn64, Kosio, Epicality85, Sunix, StonedLover, Isaac, LasTeck ...`



Test Summary

We were able to run 207 CCU worst case on one zone without any networking problems.

The server was located in the EU, so there was definitely some latency for our non European testers, which is expected.

As mentioned above, we forgot to set the database saving delay from 30s to something more realistic like 5 minutes. In our test, database saving was the only real bottleneck and it always caused some lag while saving.

We are very happy about the result. The networking ran very solid, without any disconnects or packet loss. There is no doubt that a real case CCU of 500 players should be doable now. With NetworkZones, 1000 CCU should be no problem.

The next (and final) Test

We still want to do one final 500 CCU real world test to remove any last doubts. For this test we will:

- Create a way bigger map where not all Players broadcast to each other
- Use MYSQL as the database backend for maximum performance
- Gather at least 100 people that can run 5 clients each. We will probably need a holiday for that.