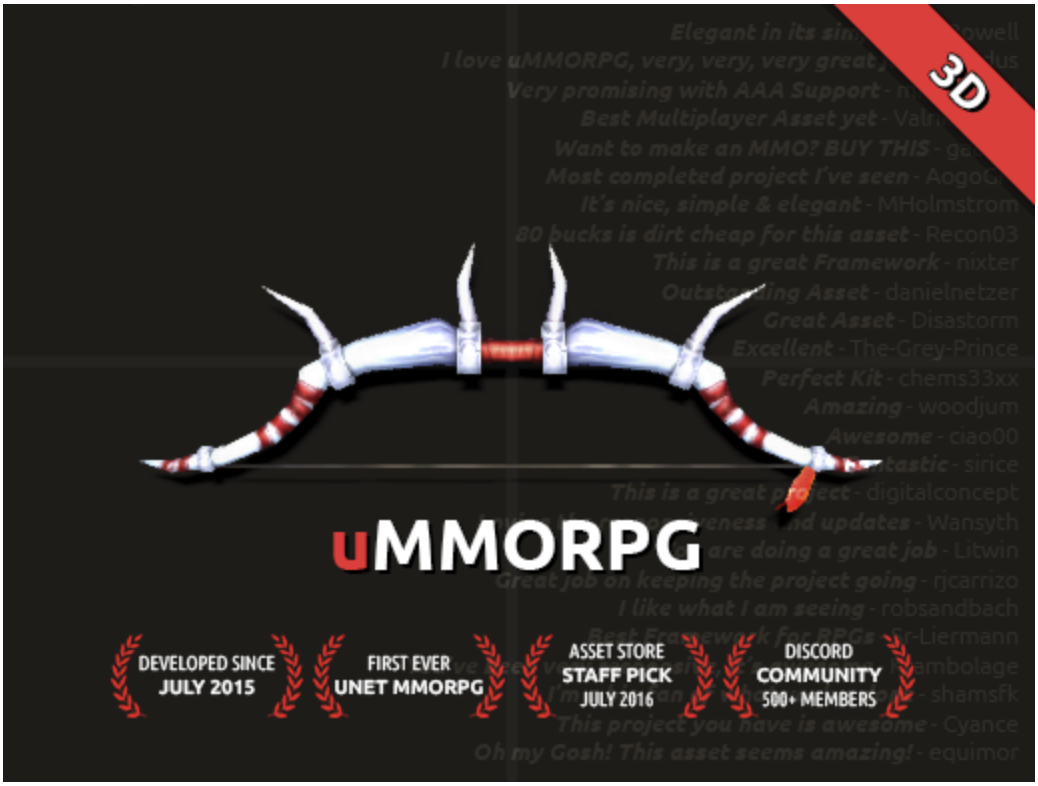
uMMORPG Documentation



(Asset Store) (Support Email) (Support Discord) (FAQ) (Forum) (WebGL)

(Online Documentation)

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 UNET manual to understand Unity's Networking system.
- Read the rest of this documentation to understand all the components.

Recommended Unity Version

Right now, the recommended Unity version is Unity 2017.4 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.

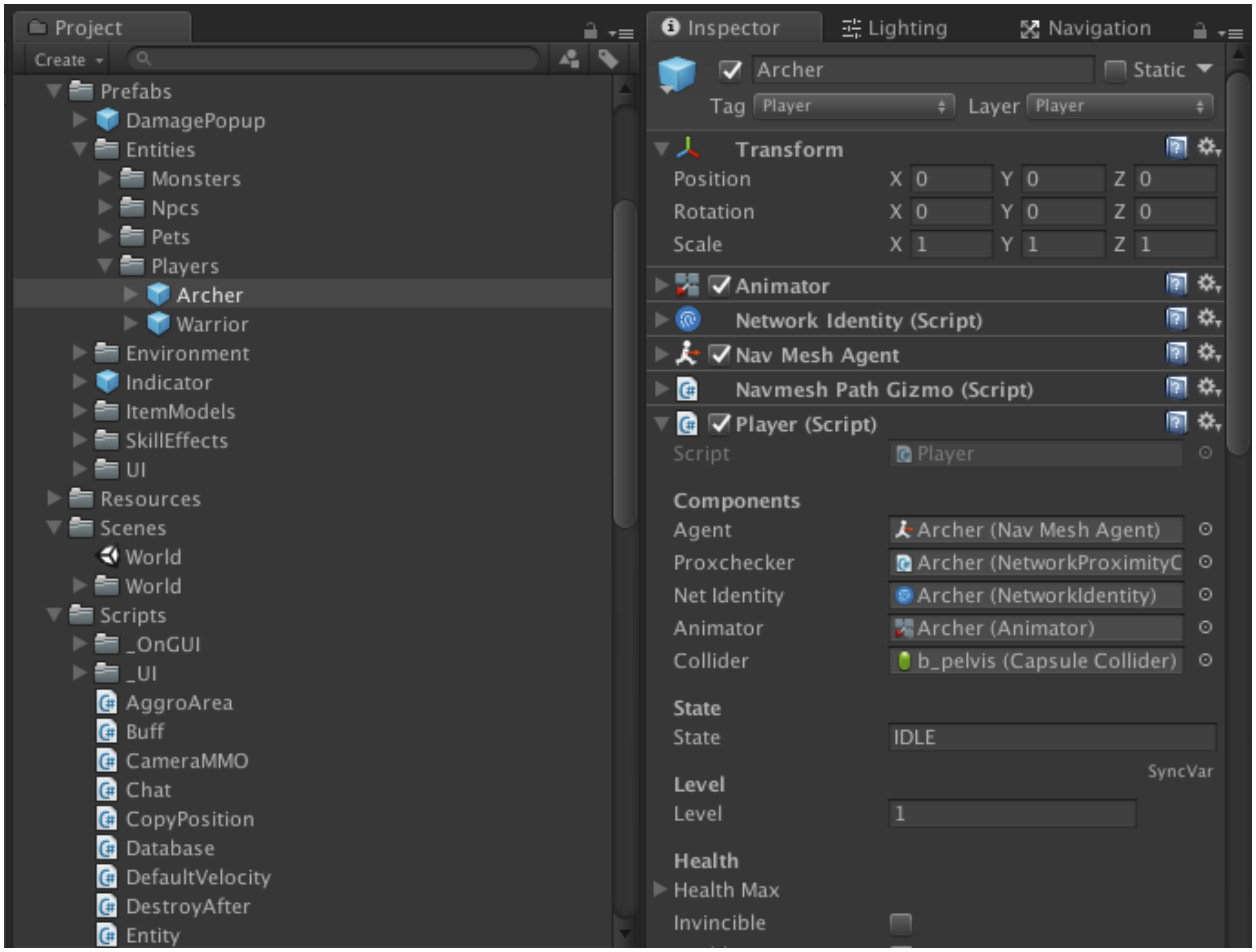
HLAPI Pro

A long time ago when starting to work on uMMORP, I encountered several critical and non critical UNET bugs. Sadly the UNET developers still haven't fixed the majority of them, so I decided to fork UNET's HLAPI and to fix them myself with HLAPI Pro.

HLAPI Pro is free and 100% compatible with the built in HLAPI - it just fixes the bugs in the background. It's a drop in replacement and while you don't have to use it for uMMORPG, you still absolutely should use it for you own sanity.

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.

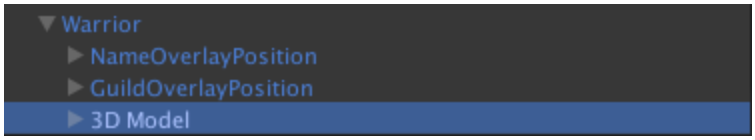
You can of course add your own components to player prefabs too.

Player Movement

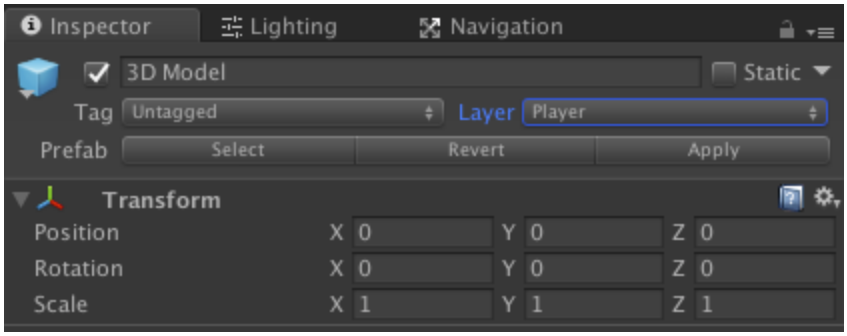
Players move on Unity's NavMesh with their NavMeshAgent component.

Modifying the Player Model

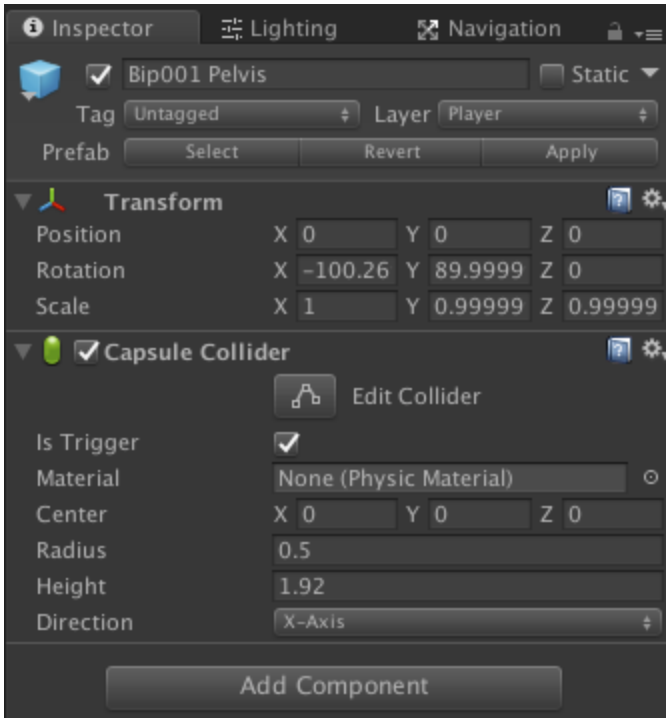
1. Drag the prefab into the Scene.
2. Replace the 3D Model child object with your new 3D Model



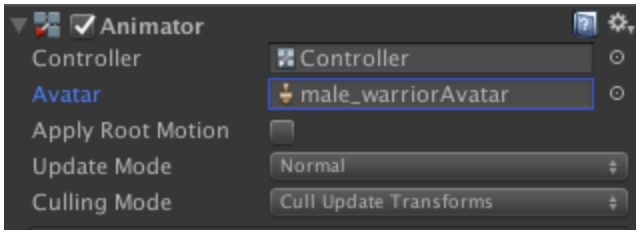
3. Set the 3D Model's Layer to Player again:



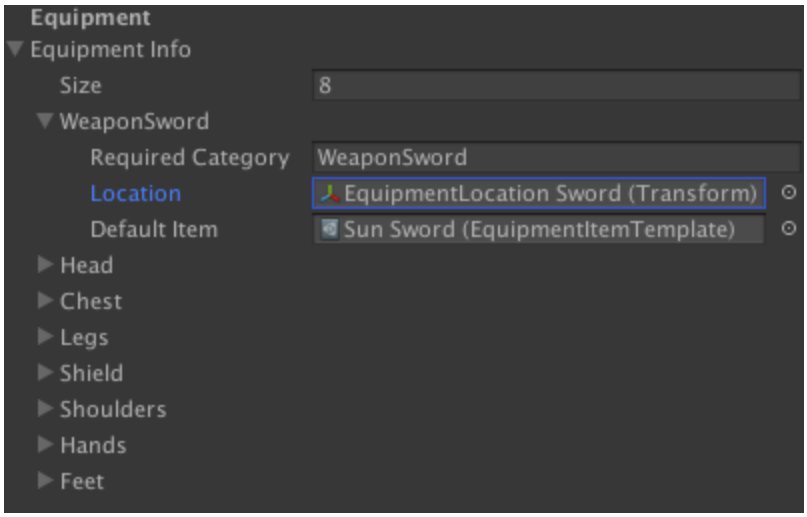
4. Add a fitting Capsule Collider to the hip bone, so that it follows the animation:



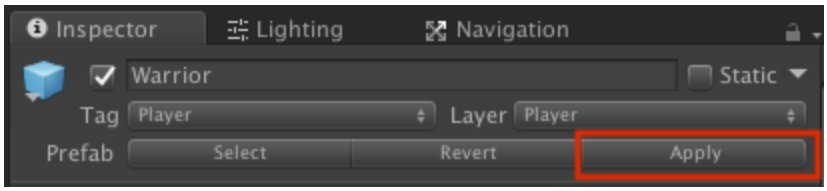
5. Assign the new model's Avatar in the Animator component:



6. Reassign Equipment Locations to the proper bones:



7. Apply the changes:



Creating a new Player Prefab

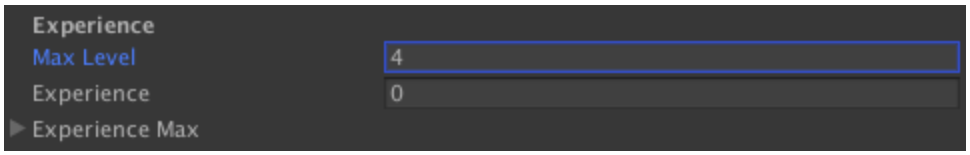
If you want to create another player type, simply drag the existing player prefab into the scene, modify it to your needs, rename it and then drag it back into the folder to save it as a different prefab.

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

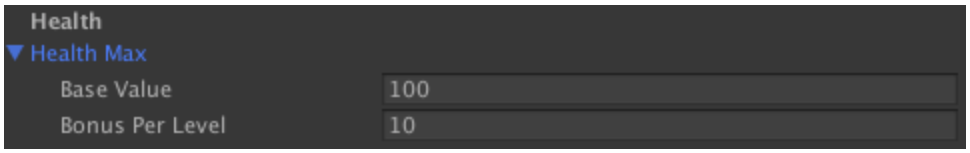
Make sure to also drag the new prefab into the NetworkManager's spawnable prefabs list. uMMORPG will then automatically display it as another option in the character creation menu.

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.

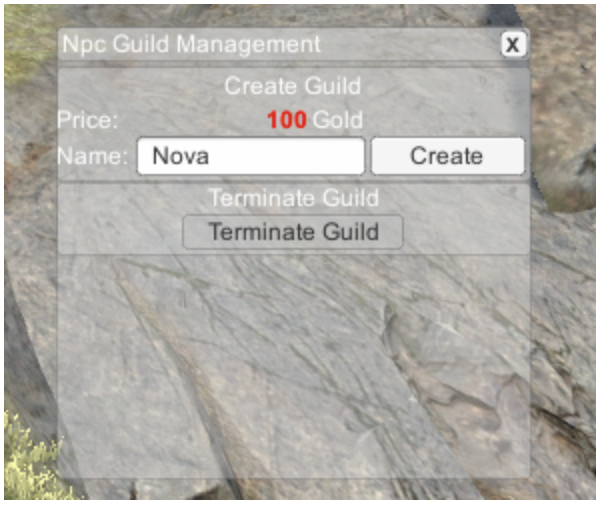
A player can invite another player to a party by standing close to the other player, targeting him and then pressing the Party Invite button in the Target UI:

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:

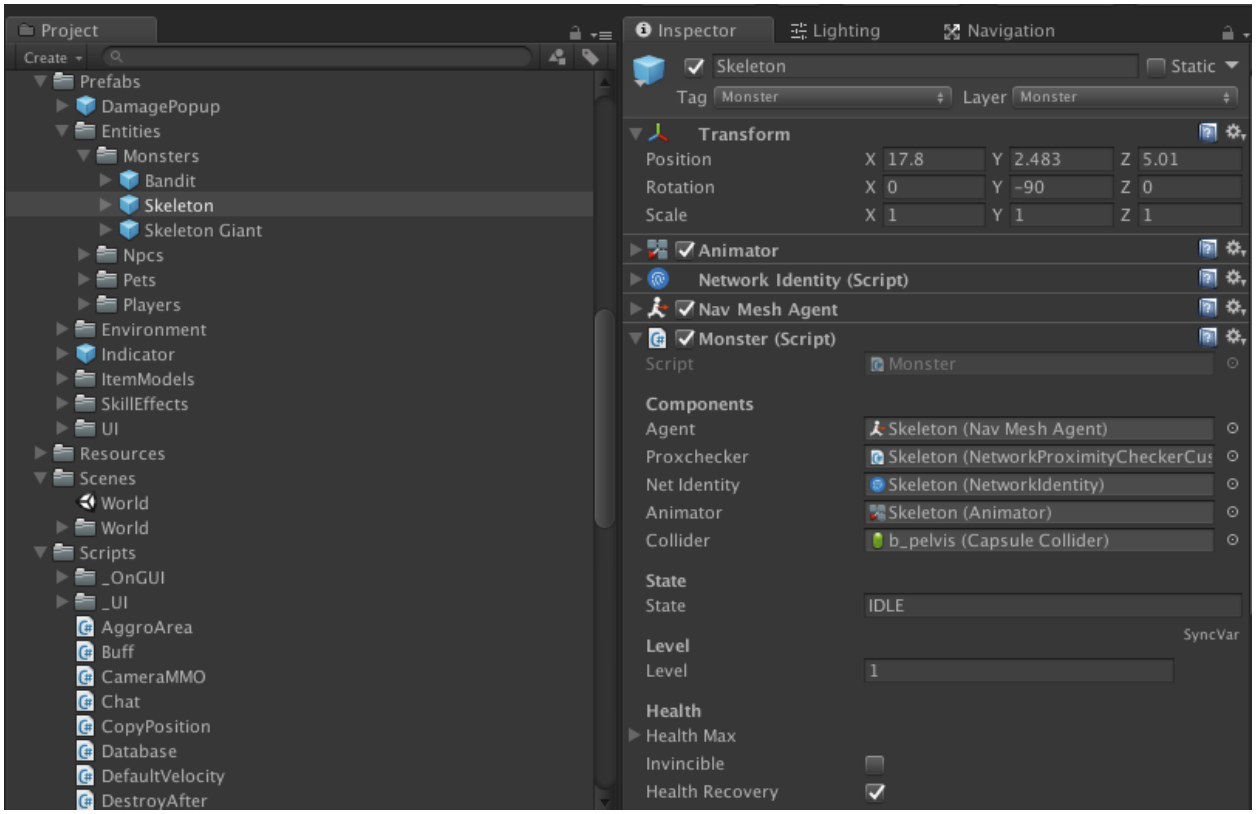


A player can invite another player to a guild by standing close to the other player, targeting him and then pressing the Guild Invite button in the Target UI:

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 zombies. Simply drag them into the scenes and position them wherever you want them to live.

Monster Movement

Monster 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.

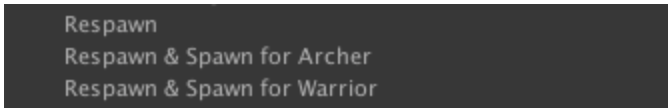
Monster Behaviour

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.

Note: there's no easier way to do it with UNET at the moment, and it does work really well too.

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
- Go to Window->Navigation and rebake the Navmesh each time you modify the environment. 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/SkillTemplates to learn how the existing skills were implemented.

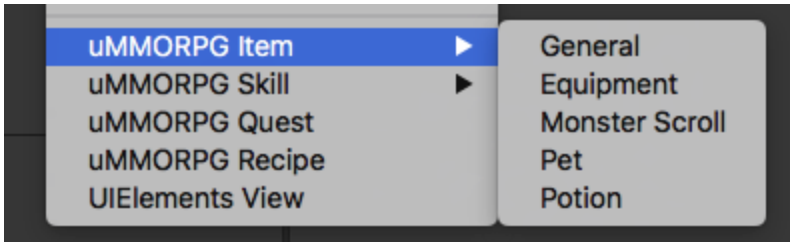
Items

Scriptable Items

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

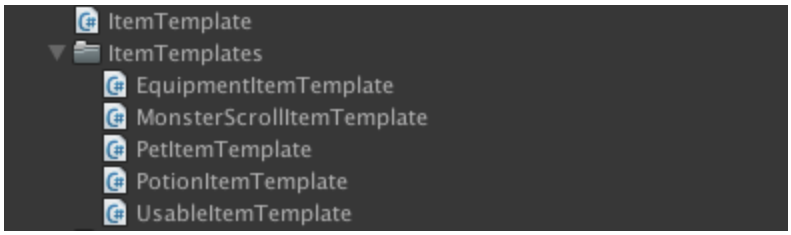
- You can duplicate and modify the Health Potion to make a Strong Health Potion.
- You can duplicate and modify the White Box to make a Crossbow.
- You can duplicate and modify the Monster Scroll to make one that spawns a different Monster.
- Etc.

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** (aka ItemTemplates):



Please take look at the **Scripts/ItemTemplate.cs** file and the **Scripts/ItemTemplates** folder with the currently implemented Scriptable Item types. In most cases, you will want to inherit from UsableItemTemplate.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 UsableItemTemplate 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 MonsterScrollItemTemplate : UsableItemTemplate {

}
```

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

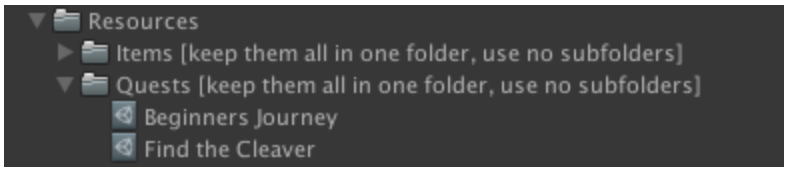
For static equipment like swords and shields, all we have to do is create the 3D model and use it in an item's model prefab.

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

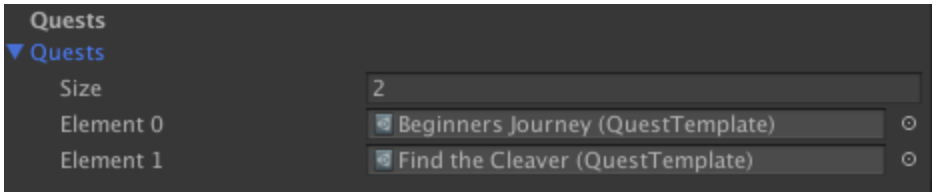
After adding the model to the project folder in Unity it's also important to add an Animator component to the prefab. uMMORPG will then automatically copy the character's controller into the equipment prefab's Animator controller slot and pass all the same parameters to it.

Quests

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

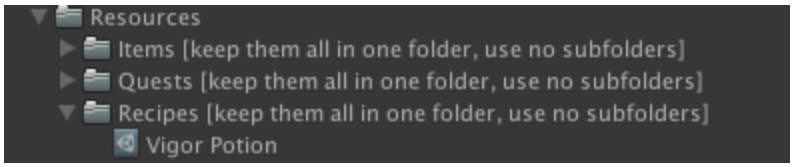


To add more quests, simply duplicate an existing one and modify it to your needs. Afterwards drag it into an Npc's available quests list:

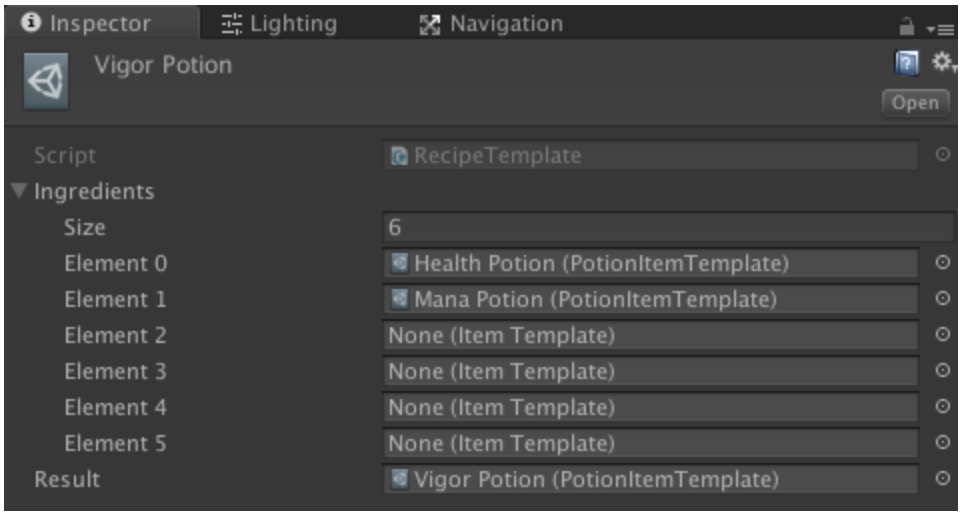


Crafting

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. Unity doesn't have a Duplicate option for Scenes, but we can open the project folder with a file manager, duplicate the scene file and then rename it.

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.

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.

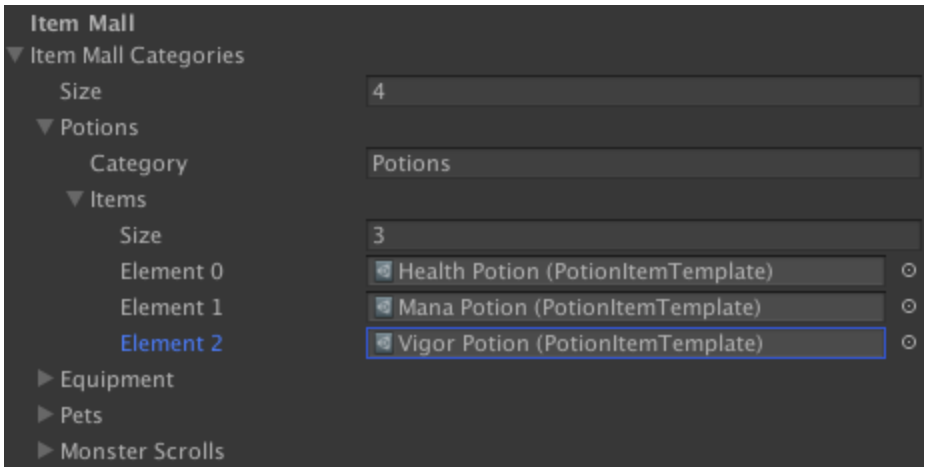
Most of the UI elements have UI components attached to them. There is no magic here, they usually just find the local player and display his stats in a UI element.

Feel free to modify all the UI to your needs.

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.

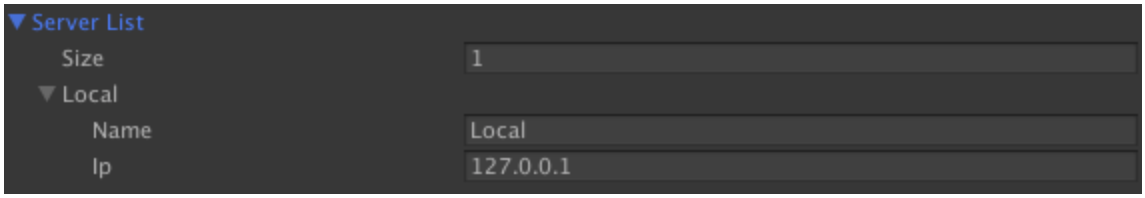
   Note: you can also process new orders manually and add them to the character_orders table by hand.

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.

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.

In short: pick Ubuntu, upload the server binary and then run it.

If you want to see log messages in the terminal, then run it with this command:

```
./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.