Documentation

Content

Meta

Register
Log in
Entries RSS
Comments RSS
WordPress.org

Search & hit enter..

Quick start guide

1. Install and open Unity 5.5 or newer.
2. Import the Project from Unity's Asset Store and open the World scene file.
3. Build & Run it for your operating system via **File**->**Build Settings**
   Note: you can also press Ctrl+B to immediately build and run it afterwards.
4. Select "Server & Play" in the Editor to start the Server and play on it (with the account entered in the Login Mask)
   Note: a Dedicated Server lets you start a Server without playing on it at the same time..
5. Enter a different account (any one is accepted right now) and press "Login" in the build to see the first two players in your MMORPG.

Architecture Overview

uMMORPG was designed for you to modify. Here is a quick overview for the code and architecture:

- The **NetworkManagerMMO** lives in the Hierarchy. It's used for login, character selection/creation and to start/stop the server.
- The Canvas holds all **UI** elements. Each element has a script that pulls in the local player's state for health, mana, etc.
- The **Main Camera** follows the local player.
- Players, Monsters, Npcs all inherit from **Entity**. Entity holds common properties like health, mana, damage, defense.
    - **NPCs** don't do much except for standing around and offering quests and items.
    - **Monsters** use a finite state machine for IDLE/MOVE/CASTING/DEAD etc.

    - **Players** use a finite state machine for IDLE/MOVE/CASTING/DEAD etc too. There is also a localPlayer check. Local players react to input and send commands to the server.
    - There are **Prefabs** for all players/monsters /npcs. Use the prefabs to modify them.
- Item**Template**/Quest**Template**/Skill**Template** are what's usually stored in a database. Thanks to Unity ScriptableObjects we can edit them in the Inspector though. Templates can be found and modified in the Resources folder.
- **Item**/**Quest**/**Skill** structs are the actual instances that live in the player/monster/npc. They pull their static properties from the templates. They can have dynamic properties like cooldown, amount, etc.
- There is also **NetworkTime** which has to be in the Hierarchy for time synchronization.

General usage advice

If you are a developer then you can either use this project to build your own customized MMORPG on top of it, you can use it as a reference to learn the best way to implement specific MMORPG features or you can even take components out of it and add them to your own game.

We used lots of comments throughout the code, so if you want to learn more, simply take a look at the implementation of the function that you are interested in.

HLAPI-Pro

HLAPI-Pro is a drop-in replacement of UNET's high level networking API (HLAPI). HLAPI-Pro works exactly the same, but is a lot more stable and has a lot of bugfixes that are really important for MMORPGs.

We highly recommend using HLAPI-Pro along with uMMORPG.

A Beginner's Exercise

A simple exercise to get your feet wet without writing any code: try adding a new 'Large Health Potion' item that restores twice as much health as the default potion. Give it a nice description, adjust all the properties and add an icon too.

When you are done, give players a way to obtain it in the game world, e.g. by adding it to a monster's drops.
Note: the Tutorial section on items could be useful.

Tutorials

How to add or modify a Player/Monster/Npc Type in 2D

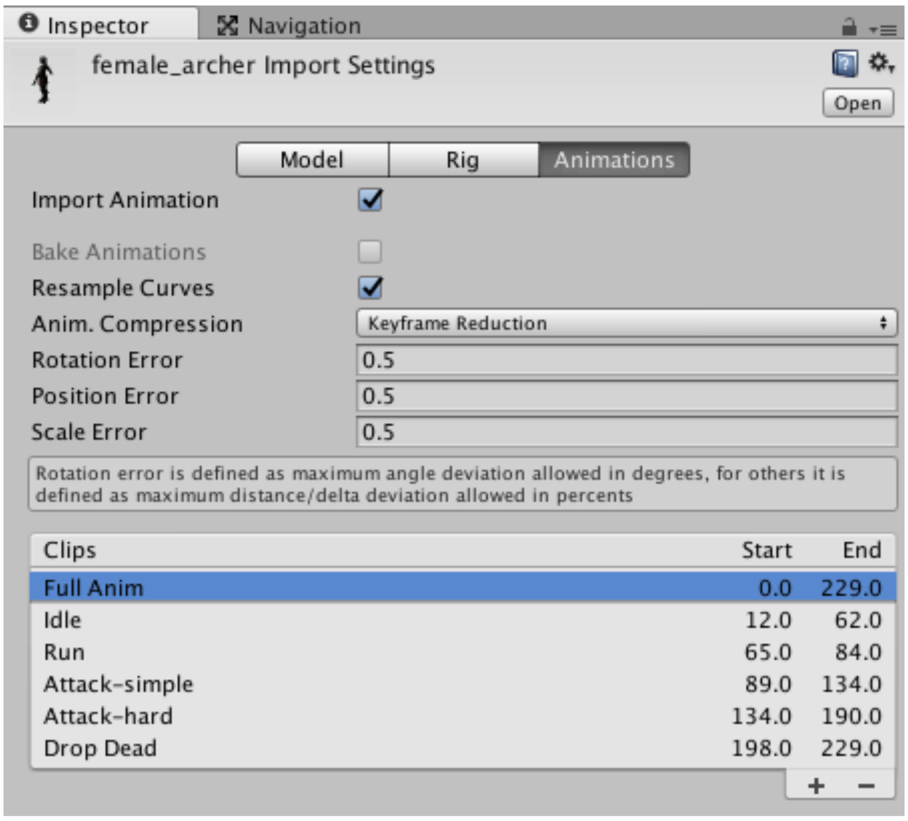
Adding or changing entities in 2D is very easy, but there are a few things to keep in mind. Here is an example on how to create a new player class:

1. Duplicate the Warrior's sprite sheet (CTRL+D) so that you don't have to slice it yourself. Then modify the sprite sheet with your graphics tool of choice and put the new player class into it.
2. Select all the IDLE sprites, drag them into the world to create the IDLE animation. Repeat for all the other animations like ATTACK and RUN.
3. Duplicate the Warrior prefab (CTRL+D to duplicate), rename it to something like "Archer".
4. Modify the Sprite Renderer's Sprite slot to your new sprite
5. Modify the Animator's Controller slot to your new controller
6. Modify the Player components to your needs if you want to change health, mana, damage etc.
7. Drag the new prefab into the NetworkManager's Registered Spawnable Prefabs list. uMMORPG will detect it automatically.

How to add or modify a Player/Monster/Npc Type in 3D

Adding or changing entities is not very difficult, but there are a few things to keep in mind. We will go through the process that we used to create the archer from our warrior, one step after another.

First of all, make sure that your 3D model is correctly imported with all the animations:



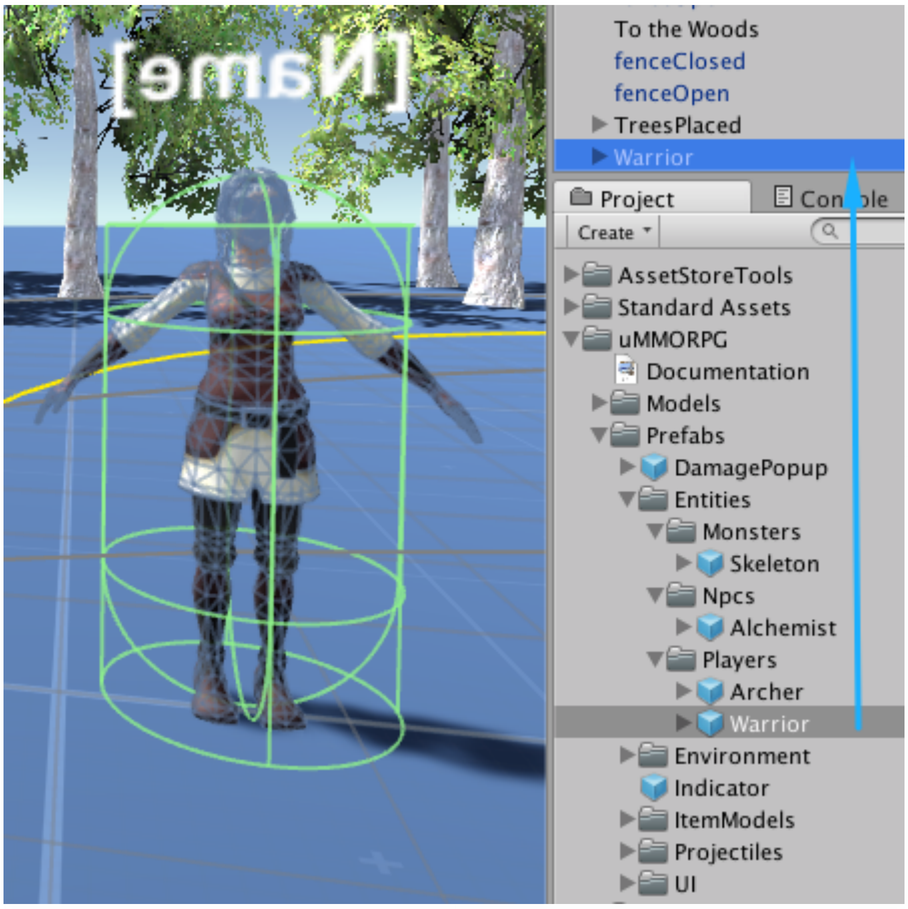
female_archer Import Settings

Import Animation	✓
Bake Animations	
Resample Curves	✓
Anim. Compression	Keyframe Reduction
Rotation Error	0.5
Position Error	0.5
Scale Error	0.5

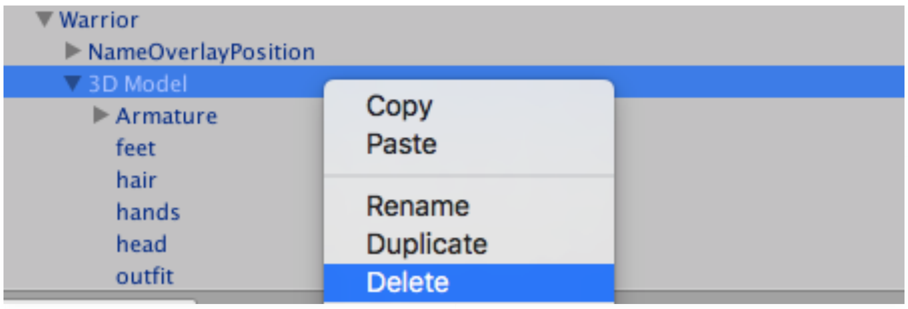
Rotation error is defined as maximum angle deviation allowed in degrees, for others it is defined as maximum distance/delta deviation allowed in percents

Clips	Start	End
Full Anim	0.0	229.0
Idle	12.0	62.0
Run	65.0	84.0
Attack-simple	89.0	134.0
Attack-hard	134.0	190.0
Drop Dead	198.0	229.0

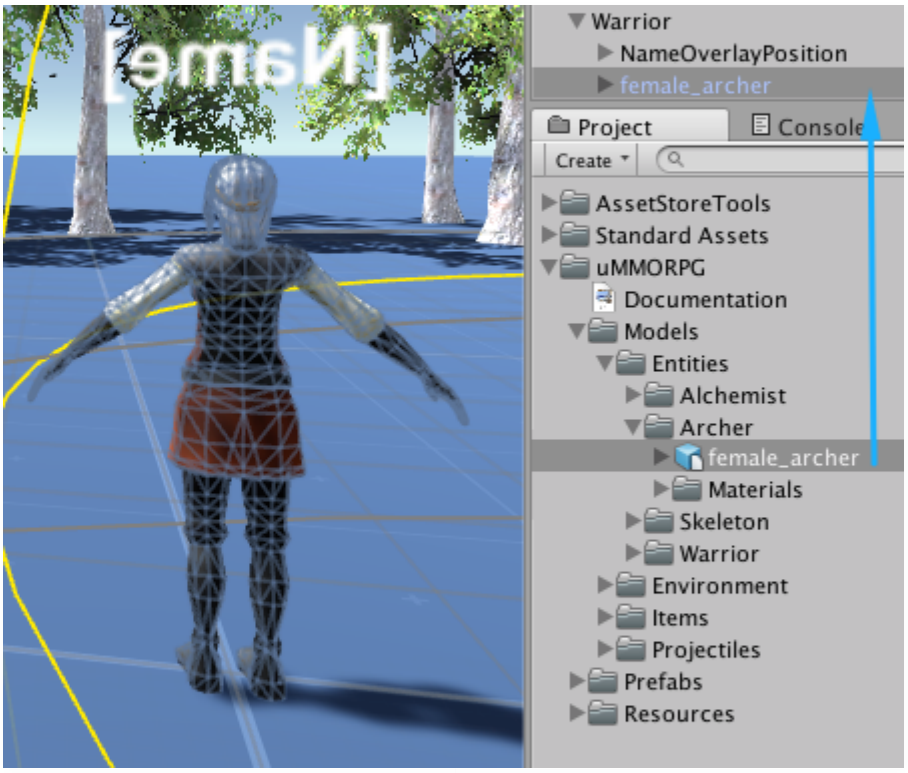
Drag the Warrior Prefab into the Hierarchy:



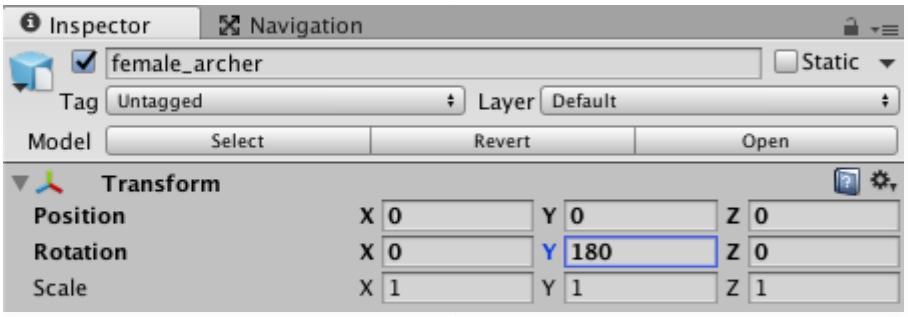
Then remove the 3D Model child object in the Hierarchy:



Now drag the new mesh from your model file *(FBX etc.)* into it:

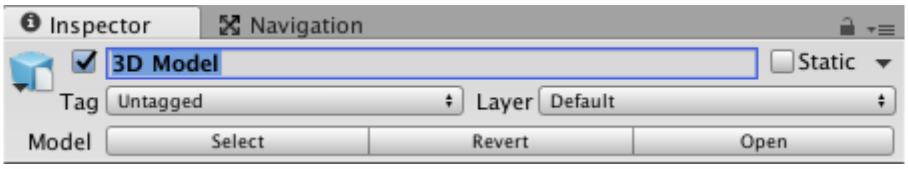


If the model is facing into a weird direction, then rotate the model part:



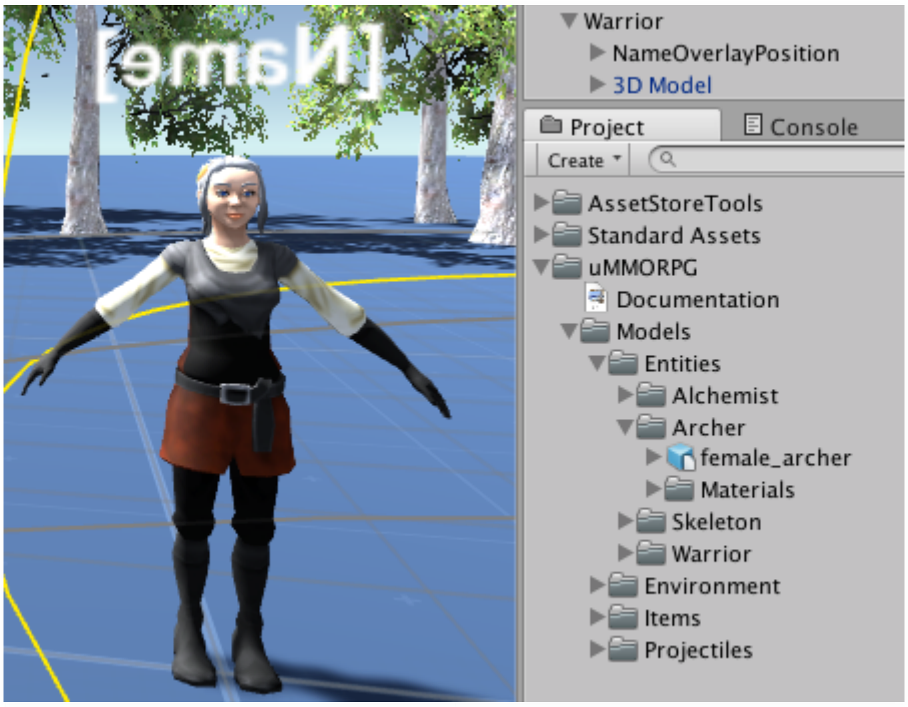
female_archer
Tag Untagged Layer Default
Model
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 180 Z 0
Scale X 1 Y 1 Z 1

We also rename it to '3D Model' for consistency:

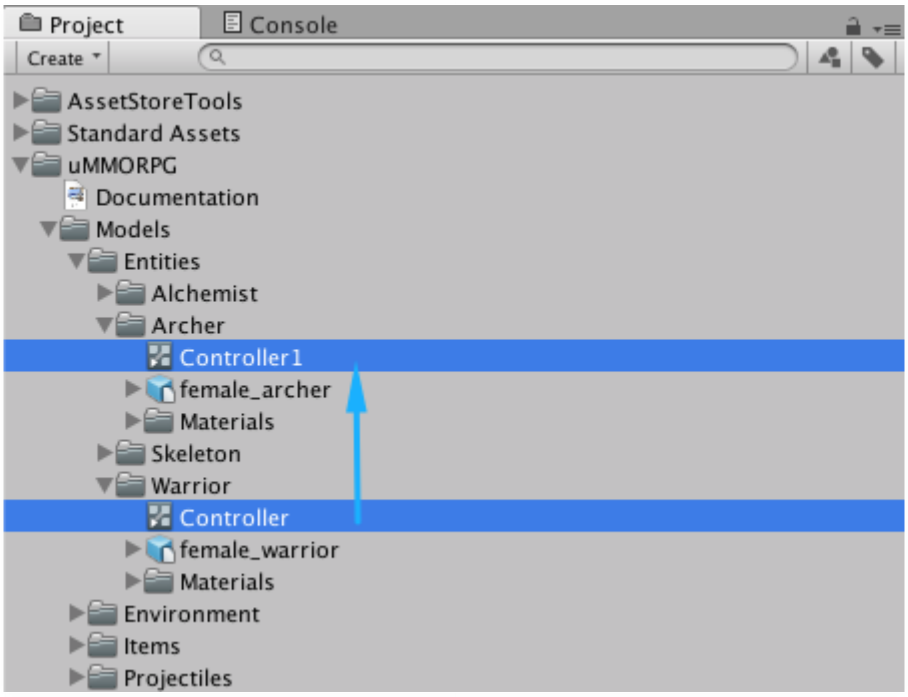


3D Model
Tag Untagged Layer Default
Model

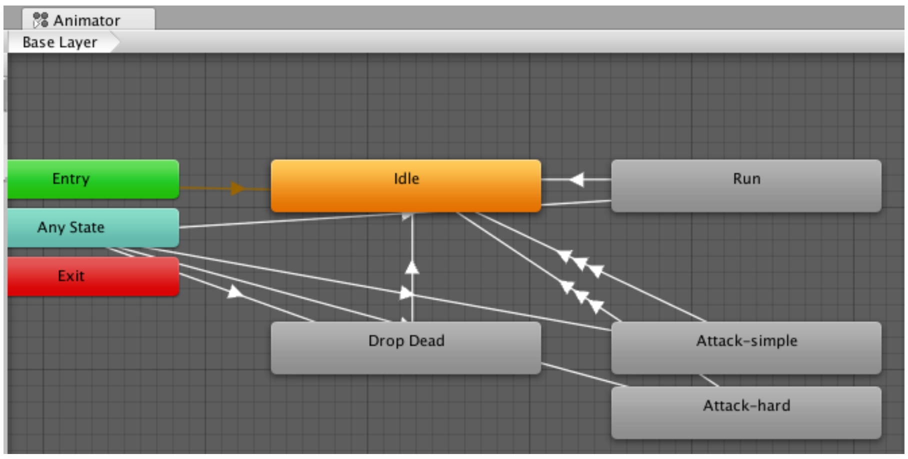
Now it should look properly rotated in the Scene:



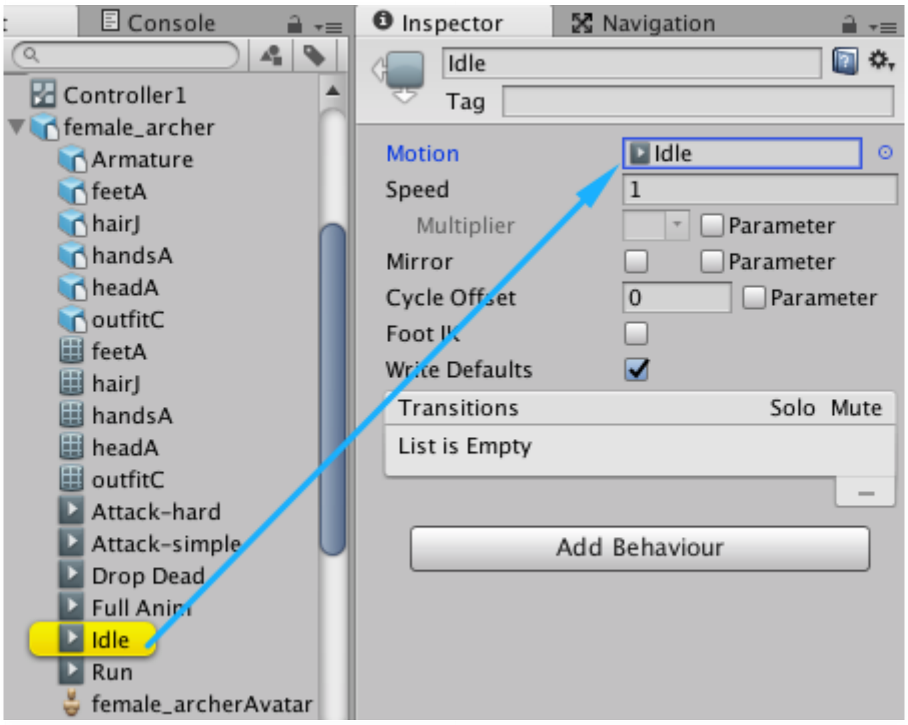
Every animated model needs an animation controller. So let’s duplicate the Warrior’s controller *(CTRL+D)* and then move it into the Archer model folder:



Double click to open the Controller in the Animator:



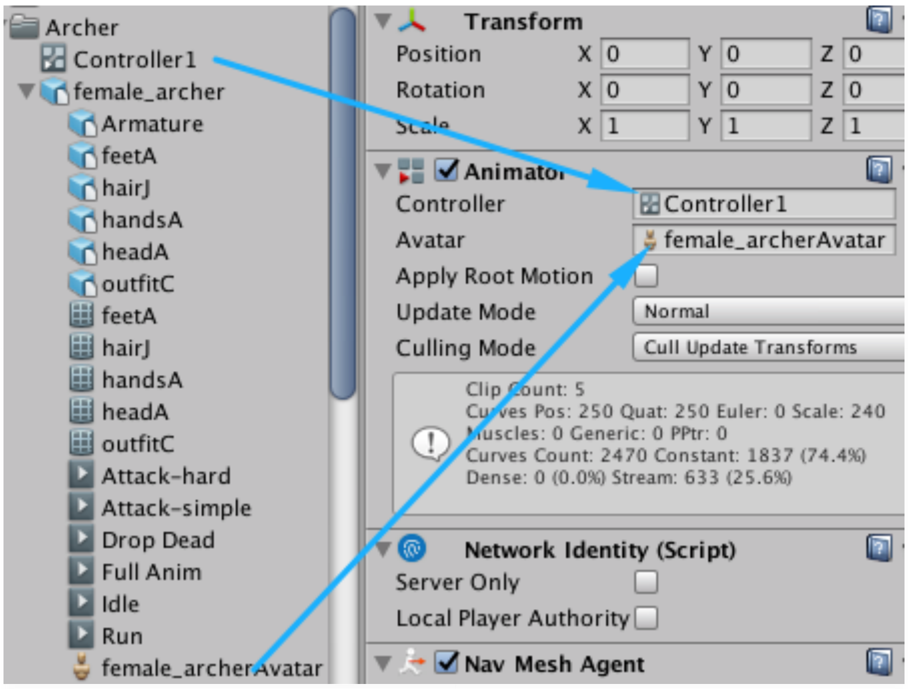
Select the Idle state, then drag the archer model's idle animation *(it's a child of the model file)* into the Motion slot in the Inspector:



Idle
Tag
Motion: Idle
Speed: 1
Multiplier: Parameter
Mirror: Parameter
Cycle Offset: 0 Parameter
Foot IK
Write Defaults ✓
Transitions Solo Mute
List is Empty

Replace all other state's animations as well. Note that you may have to modify the Transition *(the white arrows)* Duration times later if your animations stop too abruptly or too slowly.

Now select our GameObject in the Hierarchy again and then drag the archer model's Controller and Avatar into the Animator component:


Archer
Controller1
female_archer
Armature
feetA
hairJ
handsA
headA
outfitC
feetA
hairJ
handsA
headA
outfitC
Attack-hard
Attack-simple
Drop Dead
Full Anim
Idle
Run
female_archerAvatar

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

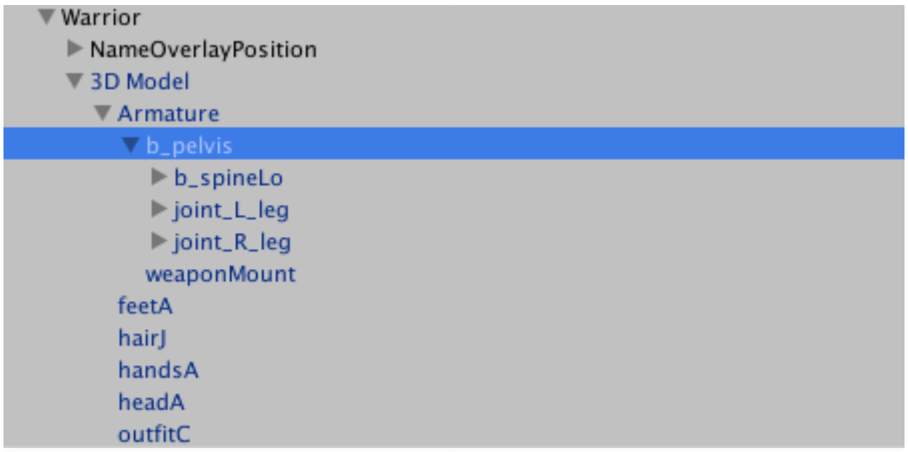
Animator
Controller: Controller1
Avatar: female_archerAvatar
Apply Root Motion: ☐
Update Mode: Normal
Culling Mode: Cull Update Transforms

```
Clip Count: 5
Curves Pos: 250 Quat: 250 Euler: 0 Scale: 240
Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 2470 Constant: 1837 (74.4%)
Dense: 0 (0.0%) Stream: 633 (25.6%)
```

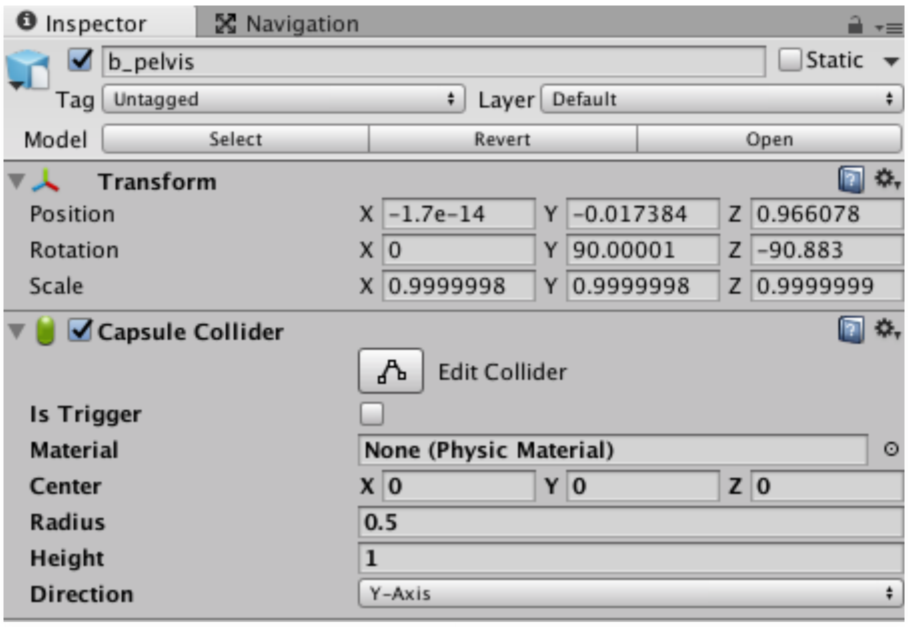
Network Identity (Script)
Server Only: ☐
Local Player Authority: ☐

Nav Mesh Agent

Each entity also needs a Collider. We could add it to the main GameObject, but it’s usually a better idea to add it to the Pelvis bone, so that it follows the animation. Imagine a standing monster and a dead monster that lays on the ground. If we would attach the Collider to the main GameObject, then it would be in the default standing position at all times. If we attach it to the pelvis then it moves around with the Animation and the Collider will stand while the monsters is standing and it will lay on the floor when the monster is laying on the floor. So let’s find the pelvis in the model’s armature *(bone structure)* in the Hierarchy:


Warrior
NameOverlayPosition
3D Model
Armature
b_pelvis
b_spineLo
joint_L_leg
joint_R_leg
weaponMount
feetA
hairJ
handsA
headA
outfitC

And then add a Capsule Collider to it in the Inspector:


Inspector | Navigation

b_pelvis | Static

Tag: Untagged | Layer: Default

Model: Select | Revert | Open

Transform

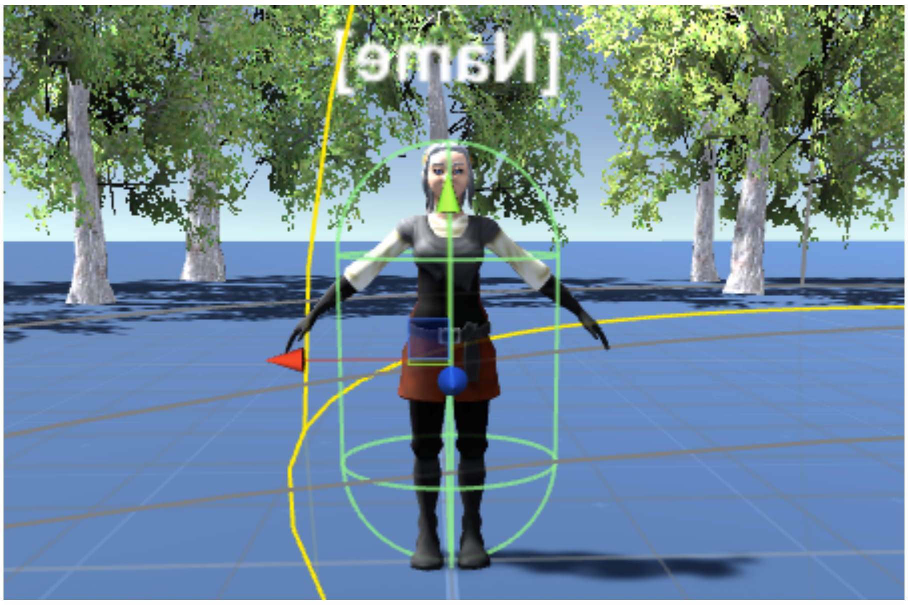
Position	X	-1.7e-14	Y	-0.017384	Z	0.966078
Rotation	X	0	Y	90.00001	Z	-90.883
Scale	X	0.9999998	Y	0.9999998	Z	0.9999999

Capsule Collider

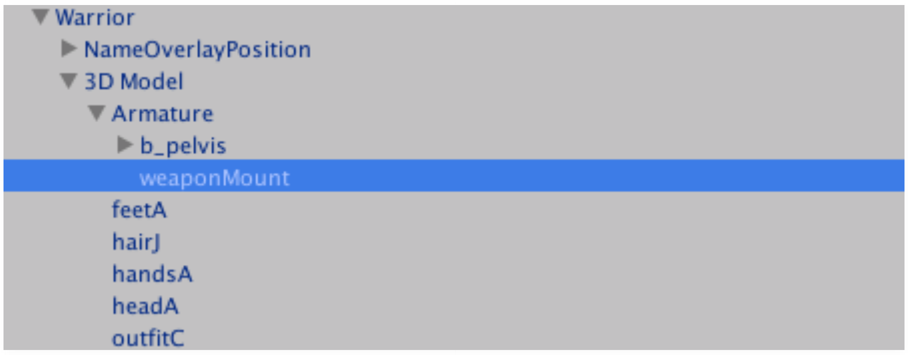
Edit Collider

Is Trigger	☐
Material	None (Physic Material)
Center	X 0 Y 0 Z 0
Radius	0.5
Height	1
Direction	Y-Axis

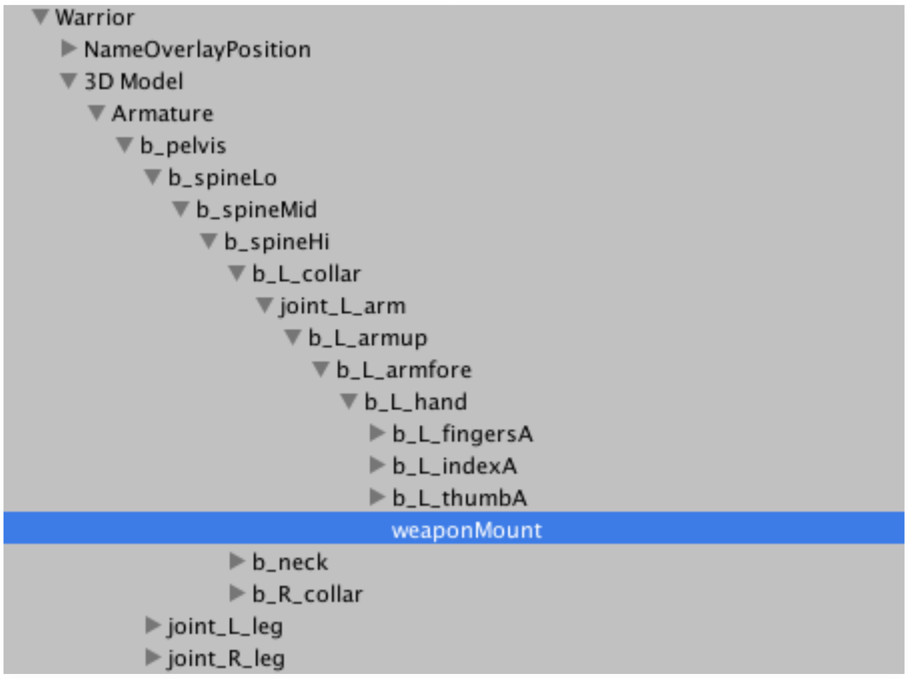
Adjust the Collider properties in the Inspector until the Collider fits the model:



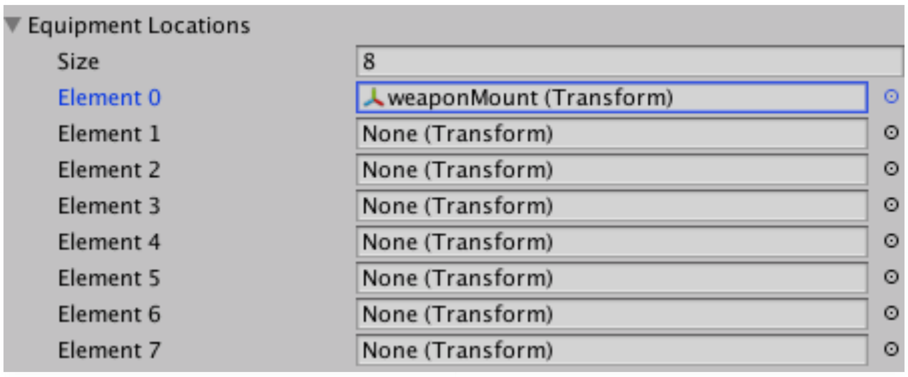
Players have the ability to equip and unequip items. If a player equips a weapon, then the item's 3D model should be shown in the player 3D model's hand. The code for this mechanic is already written, all we have to do is find the bone that should hold the weapon. 3D Artists usually include a "weaponMount" bone somewhere in the armature:


```
▼ Warrior
  ▶ NameOverlayPosition
  ▼ 3D Model
    ▼ Armature
      ▶ b_pelvis
        weaponMount
    feetA
    hairJ
    handsA
    headA
    outfitC
```

If there is no weapon mount yet, then find the left hand bone and add an empty “weaponMount” GameObject to it:


```
▼ Warrior
  ▶ NameOverlayPosition
  ▼ 3D Model
    ▼ Armature
      ▼ b_pelvis
        ▼ b_spineLo
          ▼ b_spineMid
            ▼ b_spineHi
              ▼ b_L_collar
                ▼ joint_L_arm
                  ▼ b_L_armup
                    ▼ b_L_armfore
                      ▼ b_L_hand
                        ▶ b_L_fingersA
                        ▶ b_L_indexA
                        ▶ b_L_thumbA
                          weaponMount
              ▶ b_neck
              ▶ b_R_collar
        ▶ joint_L_leg
        ▶ joint_R_leg
```

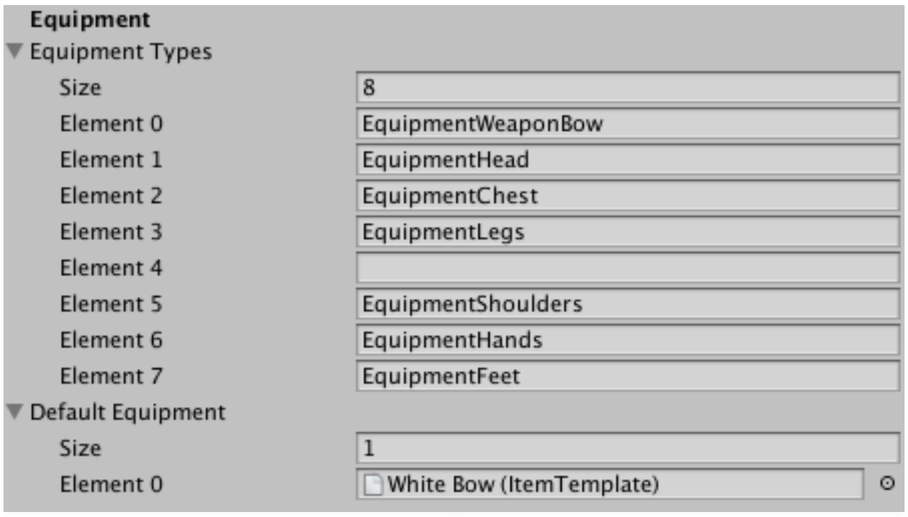
And then drag the bone into the Archer’s Player component’s Equipment Location slot in the Inspector:



▼ Equipment Locations

Size	8
Element 0	weaponMount (Transform)
Element 1	None (Transform)
Element 2	None (Transform)
Element 3	None (Transform)
Element 4	None (Transform)
Element 5	None (Transform)
Element 6	None (Transform)
Element 7	None (Transform)

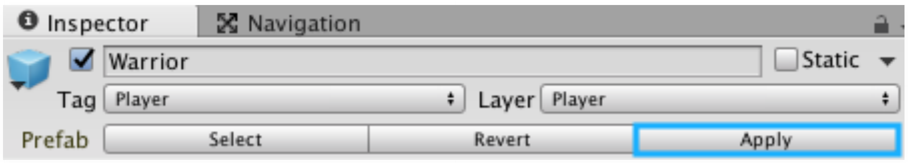
Now we can adjust the rest of the Player component to our needs. We will keep it simple and only adjust the Equipment Types so that the archer is not allowed to use a shield, and is only allowed to use bow weapons. We will also drag the bow into the Default Equipment list:



Equipment	
▼ Equipment Types	
Size	8
Element 0	EquipmentWeaponBow
Element 1	EquipmentHead
Element 2	EquipmentChest
Element 3	EquipmentLegs
Element 4	
Element 5	EquipmentShoulders
Element 6	EquipmentHands
Element 7	EquipmentFeet
▼ Default Equipment	
Size	1
Element 0	White Bow (ItemTemplate)

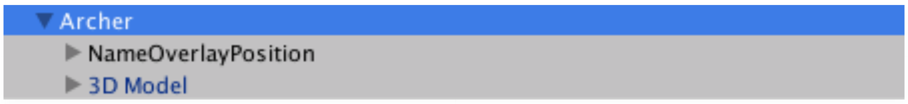
So far, all we did was modify the Warrior prefab in the Hierarchy. We can now either save our changes to the Warrior Prefab, or create a completely different Archer Prefab from it.

If we want to save our modifications to the Warrior prefab, then we can simply press the Apply button:

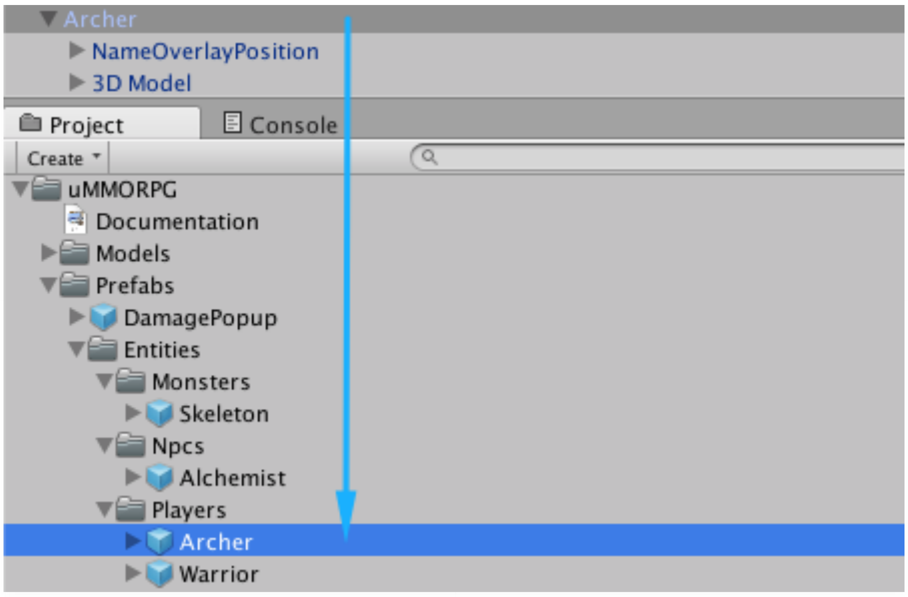


Afterwards we can remove it from the Hierarchy again.

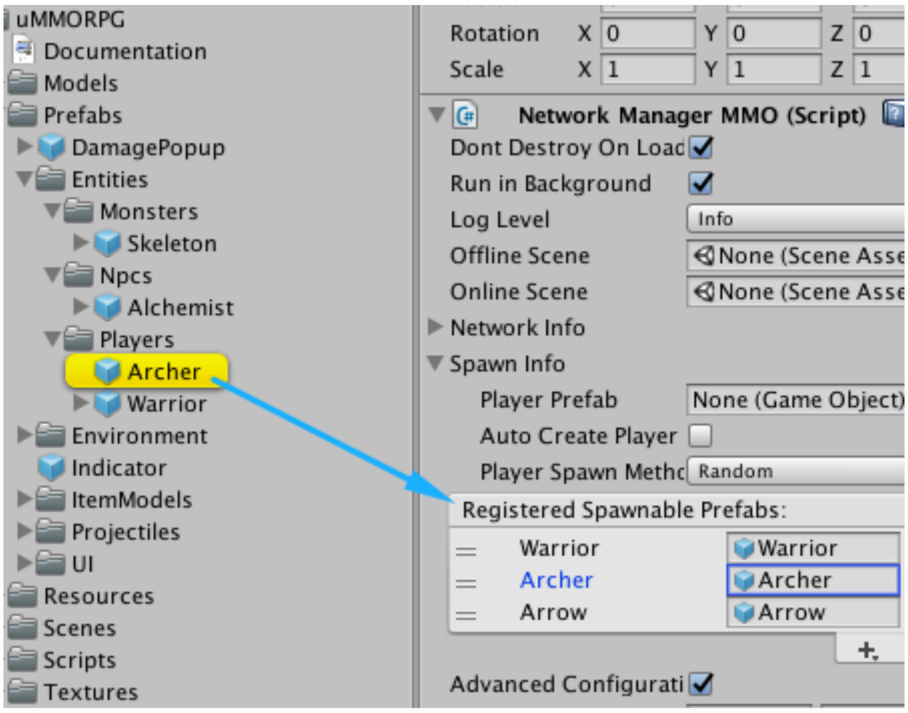
If we want to save our modifications to a new Archer prefab, then we first rename our Warrior in the Hierarchy to “Archer”:



Then drag it into the Prefabs folder:



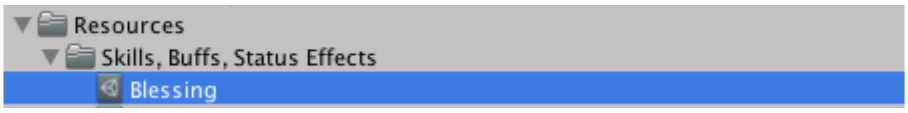
And then we drag that Prefab into the NetworkManager's Spawnable Prefabs list:



Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Network Manager MMO (Script)
Dont Destroy On Load ✓
Run in Background ✓
Log Level Info
Offline Scene None (Scene Asse
Online Scene None (Scene Asse
Network Info
Spawn Info
Player Prefab None (Game Object)
Auto Create Player
Player Spawn Methc Random
Registered Spawnable Prefabs:
Warrior Warrior
Archer Archer
Arrow Arrow
Advanced Configurati ✓

Afterwards we can remove it from the Hierarchy again.

How to add a Skill

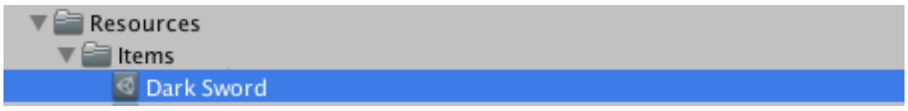


Adding a new Skill is very easy. At first we right click in the Project Area and select **Create**->**Scriptable Object** and press the **SkillTemplate** button. This creates a new ScriptableObject that we can now rename to something like "Strong Attack". Afterwards we can select it in the Project Area and the modify the skill's properties in the

Inspector. It's usually a good idea to find a similar existing skill in the ScriptableObjects folder and then copy the category, cooldown, cast range etc.

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.

How to add an Item



Adding a new Item is very easy. At first we right click in the Project Area and select **Create**->**Scriptable Object** and press the **ItemTemplate** button. This creates a new ScriptableObject that we can now rename to something like "Strong Potion". Afterwards we can select it in the Project Area and the modify the item's properties in the Inspector. It's usually a good idea to find a similar existing item in the ScriptableObjects folder and then copy the category, max stack, prices and other properties.

Afterwards we have to make sure that the item can be found in the game world somehow. There are several options, for example:

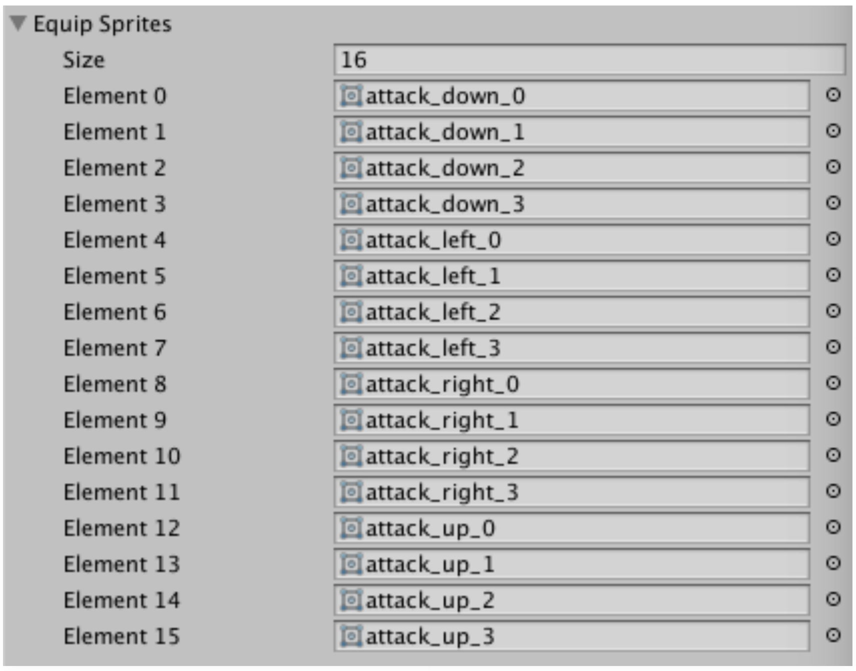
- We could add it to a Monster's **Drop Chances** list
- We could add it to an Npc's **Sale Items** list
- We could add it to a Player's **Default Items** list
- We could add it to a Player's **Default Equipment** list

How to use the Equipment System in 2D

2D equipment is simply a sprite that is put on top of the character sprite. Take a look at the existing sword sprite sheet to see how it consists of additional sword parts on top of the attack animation.

Make sure that the separate sword slices for each animation frame have the exact same name as the character slices. So for example, walk_right_0, walk_right_1 etc.

Afterwards drag all the slices into an item's EquipSprites property:



Equip Sprites

Size	16
Element 0	attack_down_0
Element 1	attack_down_1
Element 2	attack_down_2
Element 3	attack_down_3
Element 4	attack_left_0
Element 5	attack_left_1
Element 6	attack_left_2
Element 7	attack_left_3
Element 8	attack_right_0
Element 9	attack_right_1
Element 10	attack_right_2
Element 11	attack_right_3
Element 12	attack_up_0
Element 13	attack_up_1
Element 14	attack_up_2
Element 15	attack_up_3

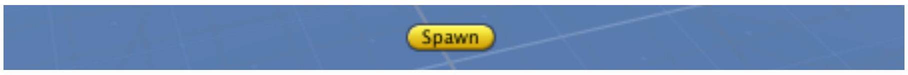
How to use the Equipment System in 3D

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

For 3D equipment like pants that should follow the character's animation, the equipment model needs to be rigged.

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.

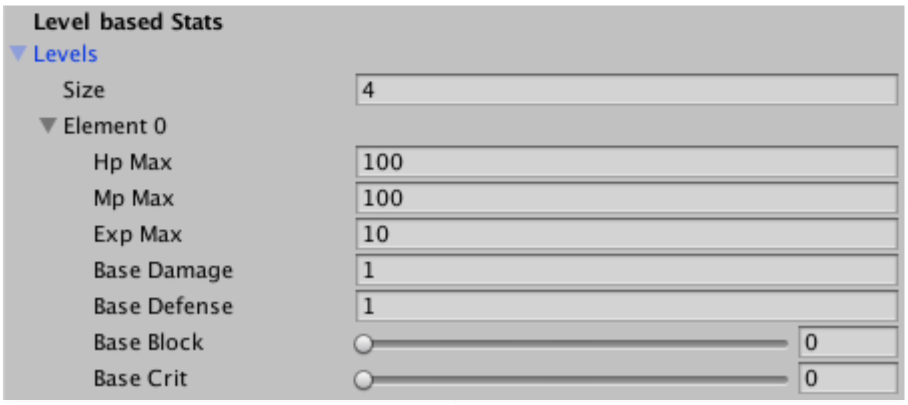
How to change the Start Position



The NetworkManager will always search for a GameObject with a **NetworkStartPosition** component attached to it and then use it as the start position.

uMMORPG has a **Spawn** GameObject in the Hierarchy, which can be moved around in order to modify the character's start position.

How to add more Player Levels


Level based Stats
Levels
Size 4
Element 0
Hp Max 100
Mp Max 100
Exp Max 10
Base Damage 1
Base Defense 1
Base Block 0
Base Crit 0

Each player Prefab has a **Levels** property in the Player component. If you want 4 levels, then set the Size to 4. Afterwards you can configure each level's stats, so that a player can have 100 health in level 1, 110 health in level 2, and so on.

How to change the ground or use a different Terrain

We can use just about any mesh or terrain as the ground element. In order to allow entities to move on it, we simply have to make it **Static**, enable the collider and then select **Window**->**Navigation** and press the **Bake** button in order to refresh the navigation mesh.

How to add Environment Models

Environment models like rocks, trees and buildings can simply be dragged into the Scene. They don't need a Collider, but they should be made **Static** so that the Navigation system recognizes them.

Afterwards we can select **Window**->**Navigation** and press the **Bake** button in order to refresh the navigation mesh. This makes sure that the player can't walk through those new environment objects.

How to add a Skill Effect

Skills can either deal damage directly or launch a skill effect into the world. Skill effects can be anything from arrows to lightning bolts.

Skills have a **Effect Prefab** property that we can use to assign skill effect prefabs. uMMORPG already comes with an example prefab, which is the Arrow. It's a 3D model of an arrow together with a script that inherits from SkillEffect.

We designed the system in a way that allows for just about any effect to work. uMMORPG simply spawns the effect into the game world and sets its **target**, **caster** and **sourceSkill** properties for you to use in your inheriting script.

For example, the Arrow simply uses FixedUpdate to fly to the **target**.

Once it reached the target, it deals damage by calling **caster.DealDamageAt(target)**.

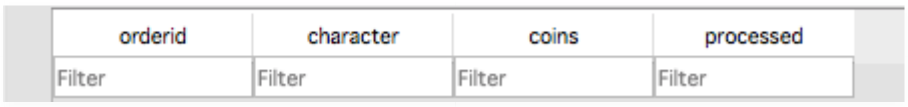
A more complex effect like a lightning bolt that jumps around between targets would be very similar. It could fly to **target**, deal damage and then instead of destroying itself like the Arrow, it could just continue flying to the next entity around it, up until for example 5 entities.

How to change the Login Screen

uMMORPG uses Unity's new UI system, so everything can be modified easily. The Unity UI Manual has a lot of information on that topic.

If you want to modify the background, you can simply move the camera to another position, for example behind a tree or in front of a camp fire.

How to use the Item Mall



orderid	character	coins	processed
Filter	Filter	Filter	Filter

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:

- Set an item's **ItemMallPrice** to a value > 0 to make it appear in the item mall automatically
- Select the Canvas/ItemMall window and change the **Buy Url** to the website where your users should buy item mall coins
- 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.
- 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.

How to use another Database System

The Database.cs class is the the only place that has to be modified in order to use another database system like MYSQL for our Unity MMORPG.

We decided to use SQLITE because it's really simple to use and doesn't require a complicated database server configuration. We also used XML a while ago, but a SQL based system is definitely need in the future when we start to support server instances after UNET's Phase 3 plan.

Note: Item, Skill and Quest templates don't really need to be stored in a Database. Right now they are just ScriptableObjects that can be referenced in the game world, which makes development very easy.

How to modify Database Characters

uMMORPG uses SQLite for its 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.

How to use another Scene

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: having multiple Scenes at the same time and allowing players to teleport to them makes sense too. It's very likely that UNET will get a feature like that sooner or later, which would be the best solution. Right now we can't use UNET to connect from one UNET server to another UNET server, which makes transferring players impossible without some really weird workarounds, hence why we don't want to implement that feature just yet.

How to add another attribute like Dexterity

uMMORPG already has strength and intelligence attributes which can be upgraded after each level up. To add more attributes, simply take a look at the Player.cs script, find the Attributes section, duplicate one of the current attributes rename it.

You can then modify the Player's Health, Mana, Damage, Defense properties to include your attribute in the formula.

You can show your new attribute in the UI by first adding elements to the CharacterInfo panel in the Canvas and then updating them with the UIRefresh script where attributes can be found in the UpdateCharacterInfo function.

You will notice that the UI script uses Commands when the player asks the server to increase an attribute. The final step is to add one of those commands to the Player script. This is very easy again, since you can simply copy one of the existing attribute commands and modify it to your attribute.

How to connect to the Server over the Local Network

If you want to connect to the game server from another computer in your local network, then all you have to do is select the NetworkManager in the Hierarchy and modify the Network Info -> Network Address property to the IP address of the server.

Note: you also have to configure both computers so that they can talk to each other without being blocked by firewalls.

How to host a Server on the Internet

You should run uMMORPG in headless mode on a Linux server *(recommended)* or in batchmode on a Windows server.

For a detailed step by step guide, check out our UNET Server Hosting Tutorial. It recommends a hoster and explains the whole process step-by-step.

Note: it's sometimes useful to also show the log messages in the console. Here is how to do it on Linux:

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

And on Windows:

```
uMMORPG.exe -batchmode -nographics -logfile log.txt
powershell -noexit Get-Content log.txt -Wait
```

How to hide 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 makes indie mmorpgs possible. 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 MMO 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.

Furthermore it's usually a good idea for MMO 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.

How to work around UNET's ReadString/ReadBytes out of Range error

UNET has a critical bug that causes ReadString/ReadBytes out of range errors. I reported it to seanr on the Unity forum in January 2016 and reported it again in April, when it was added to the Issue Tracker. It's still the highest voted UNET bug and was never fixed.

Eventually after two years I decided to fix it myself, so I started HLAPI Pro. It's highly recommended to use it for uMMORPG and it fixes a whole lot of bugs, including the ReadString/ReadBytes out of range bug.

How to update your own modified MMORPG to the latest uMMORPG

uMMORPG's code is likely to change a bit every now and then, so upgrading your modified MMORPG to the latest uMMORPG version could always break something in your project. uMMORPG V1.69 came with a new Addon system, we recommend to use that for your modifications as much as possible, so that the original scripts can be updated easily without losing your changes.

In any case, it's important to always create backups before updating, in case things go wrong.

If you still need to do a lot of modifications in the original scripts, it might be smart to pick one uMMORPG version and just stick with it. You can still manually copy over bugfixes / minor improvements from future updates if necessary.

Of course, you can always try to just update your current project and hope for the best / fix occurring errors. Just be warned that this might be stressful, because uMMORPG is a source code project. Conventional software can easily be made downwards compatible because the user never modifies the source code directly. But if you modify code that we change later on, then there can always be complications.

Note: we always try to fix all known bugs before releasing a new uMMORPG version, so you really won't have to update to the latest version all the time and can work with one version for a long time instead.

How to keep track of uMMORPG changes in detail

If you want to know every single line of code that was changed between versions, there are several ways to do that:

- Look at the file modified dates to see which ones were changed lately
- Use a any text comparison tool
- Use Git or similar version control tools

If you don't know how to use git, here is how you can see version changes with just a few mouse clicks:

- Create a new Unity project, download uMMORPG from the Asset Store
- Put a .gitignore file into that your project folder to ignore some files that we don't need to track:[sourcecode

```
language=”batch”]Temp/
Library/
ExportedObj/
obj/
*.svd
*.userprefs
/*.csproj
*.pidb
*.suo
/*.sln
*.user
*.unityproj
*.booproj
.DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
ehthumbs.db
Thumbs.db[/sourcecode]
```

- Download and open SmartGit
    - Select Repository -> Add or Create
    - Select your project folder, press OK
    - Press ‘Initialize’ in the next window
    - Now you see the list of files that are new. Select all of them and press Commit and then click Commit in the next window, use the current uMMORPG version like ‘V1.1’ as the commit message
- Now if a new uMMORPG update is released:
    - Update to the new uMMORPG version for that project
    - Open SmartGit, select your repository on the left
    - Now you see all the changed files in the middle
    - You can click each file to see a comparison between the old and the new content
    - Once you are done, select all files and Commit with ‘V1.2’, so that this version is now ‘default’. Next time you will see all the changed files again.

Note: please don’t put uMMORPG into a public repository on the internet. Git works perfectly fine on your local machine.

Development Info

A list of planned features, bugs and requests can be found in our Official uMMORPG Thread on the Unity forums.

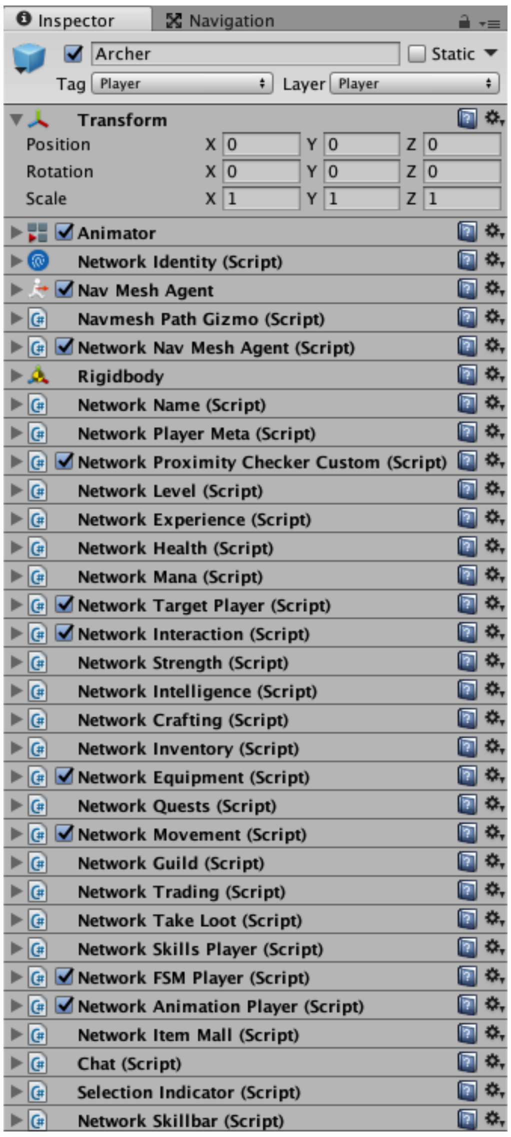
Technology Choices

Designing an MMORPG is difficult, there are a lot of technology choices to make. We will explain the most important ones for uMMORPG:

- **Networking:** When it comes to networking, we have several different choices like UNET, TCP/IP or external solutions. UNET is a really good choice for networking. Dealing with TCP Clients & Servers means lots of manual serialization, lots of packets and opcodes and dealing with networking issues. UNET takes care of all these problems. UNET is built directly into Unity, so it's really the most simple choice too, in comparison to external solutions. When it comes to UNET, we can either use SyncVars/SyncLists or use manual Serialization via OnSerialize and OnDeserialize. Using SyncVars/SyncLists saves us a lot of work when synchronizing things like item lists, skills or gold. Doing that manually with OnSerialize and OnDeserialize would require dirty bits and lots of serialization that we don't really want to worry about. We should also keep in mind that if we were to add something to our item type, we would always have to remember to serialize it as well. Luckily for us, Unity takes care of that automatically when using SyncLists. The only downside of SyncLists is the fact that we can only use simple types like int and float, or structs. We can choose between different Qos Channels in the NetworkManager. We use *Reliable Fragmented* for uMMORPG because packets should be reliable (we really do need the client to receive them). Synchronizing bigger structures like a SyncList of Items will often exceed the maximum packet size, hence why we need a *fragmented* channel type that can split big packets into smaller fragments.
- **NavMeshAgents vs. real Physics:** we don't really use any *real* physics in our MMO. Instead we completely rely on the NavMeshAgent for movement. A lot of MMORPGs have problems with gravity, falling through the level or walking through thin walls. Using only a NavMeshAgent without any physics is the solution to all of those problems because the agent can only walk on the NavMesh. There is absolutely no way to fall through the map or to walk through walls. On a side note, this also

means that we only have to synchronize the agent's destination once, instead of synchronizing a player's position every few milliseconds, hence we save a LOT of computations and bandwidth.

- **Simplicity vs. Performance:** Our #1 goal when designing uMMORPG was simplicity. We prefer simple and clean code over highly optimized and overly complex code. A 100.000 lines of code MMORPG that can handle 5000 players at once is nice if you have enough people to work on that huge amount of code. But for indie developers a 5000 lines of code Project that only supports 1000 players at once is much more valuable.
- **OnGUI vs. UI:** Unity's old OnGUI system was great, we used it for uMMORPG in the beginning. People started requesting the new UI, so we changed over to that one later. The new UI system is more complex, but it's more mobile friendly and has easier drag and drop support. When using the new UI, we always create a new script like UISkillbar and attach it to the UI component in the Hierarchy. This is a better architecture than one big UI script, or handling UI in the player class, etc.
- **Scripting Language:** Unity currently supports CSharp, JavaScript and Boo. Boo is pretty much C# with a shorter syntax, but the language has no support for SyncVar hooks, hence it's not an option. CSharp is well documented and used by most of the Unity developers, hence the choice over JavaScript.
- **Architecture: God Classes vs. Tiny Components vs. Object Oriented design:** There are lots of different architecture options for uMMORPG and we tried most of them. We ended up using a God class for Players, Monsters, Npcs and Entities. Here is why this is superior to the other architectures:
    - **Tiny Components:** Unity is a component based Game Engine, so it makes a lot of sense to split everything into tiny little components. For example, a Health component that can be reused by Players, Monsters and Npcs. We tried to split all the uMMORPG code into tiny components, here is an example of the Player Prefab and a list of Pros/Cons about this architecture:

- 
Inspector | Navigation

☑ Archer ☐ Static

Tag Player | Layer Player

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

☑ Animator
Network Identity (Script)
☑ Nav Mesh Agent
Navmesh Path Gizmo (Script)
☑ Network Nav Mesh Agent (Script)
Rigidbody
Network Name (Script)
Network Player Meta (Script)
☑ Network Proximity Checker Custom (Script)
Network Level (Script)
Network Experience (Script)
Network Health (Script)
Network Mana (Script)
☑ Network Target Player (Script)
☑ Network Interaction (Script)
Network Strength (Script)
Network Intelligence (Script)
Network Crafting (Script)
Network Inventory (Script)
☑ Network Equipment (Script)
Network Quests (Script)
☑ Network Movement (Script)
Network Guild (Script)
Network Trading (Script)
Network Take Loot (Script)
Network Skills Player (Script)
☑ Network FSM Player (Script)
☑ Network Animation Player (Script)
Network Item Mall (Script)
Chat (Script)
Selection Indicator (Script)
Network Skillbar (Script)
- **Good:** What's great about this approach is that it's very easy to find code for a certain feature. All the Inventory code is in NetworkInventory.cs, which is just a tiny file that is nice to work with. UNET's 32 SyncVar Limit also isn't a problem when using lots of smaller components.
- **Bad:** All these components have to access each other all the time. We either need lots of GetComponent calls or lots of component caching, which results in a whole lot of source code. The biggest downside is that the Syntax becomes way too verbose, for example:

```
target.health
```

becomes

```
targetComponent.target.GetCompor
```

which is just awful to work with and the main reason why we decided against this approach. Another downside is that Update doesn't really have to be called for entities that aren't around a player. With the component approach, we would have to manually enable and disable every single Update function, which is cumbersome too.

- **Object Oriented:** another option is to use lots of objects. So for example, the Player component would have objects like:

```
Guild guild = new Guild();
Inventory inventory = new Inventory();
Equipment equipment = new Equipment();
```

  - **Good:** the Object Oriented approach allows us to put all the Inventory related code into an Inventory.cs class, all the Guild code into a Guild.cs class and so on. It also keeps the Player component lean by encapsulating all the big stuff into objects.
  - **Bad:** UNET's SyncVars and SyncLists only work with simple values and structs, never with objects. There is absolutely no way to use them inside of something like a Guild object. We would have to use OnSerialize and OnDeserialize to serialize important values manually, which is a whole lot of work. UNET just wasn't developed for that kind of approach.
- **God Classes:** so called god classes are simply unusually large classes, like uMMORPG's Player.cs Script that has 1500 lines of code.
  - **Good:** there is no overhead. Unity is ultra fast because instead of 20 little components, we only have one. There are no GetComponent calls required because everything is in this one class.

The syntax is ultra short because we can write

```
if (health == 0) ...
```

instead of:

```
if (GetComponent<NetworkHealth>(
```

We also won't get a mini heart attack when trying to understand the Player prefab. It's just a Transform, NavMeshAgent, Rigidbody and Player component. There are no 20 little components that somehow interact with each other. The same applies to the Scripts folder. Instead of having 20 component scripts that do something somewhere, we have all the Player logic in one Player script. Simple and stupid, no bullshit.

- **Bad:** the only downside of god classes is that finding parts of the code is a bit more difficult when the file is bigger.
- **Note:** our Player/Monster/Npc/Entity classes are partial, which means that we can still split parts of it into different files without changing the functionality whatsoever. This is really cool.

- **Template / Addon / Plugin / Mod Systems:** we want everyone to be able to build their dream MMO with uMMORPG. There are a lot of different approaches to game modifications and we put a lot of research into finding the perfect solution. Here is an overview about the different options:
    - **Mods:** we could use Unity's Resources folder and Asset Bundles to allow users to modify the game after it was released. This is very cumbersome and strongly limits the modifications that can be made. Existing scripts and algorithms could hardly be modified. Examples are games like Skyrim where the game files can be modified with a special tool.
    - **Addons:** we could also design uMMORPG as kind of an MMORPG engine that can be extended by addon scripts. So there would be a

uMMORPG core that should never be modified by anyone, and there would be an addon class that people can use for their custom behaviours like PlayerInventory, PlayerQuests, NpcTrading, etc. The benefit of this system is that all addons are small, manageable components and that the core can easily be updated to newer versions. The downside of this system is that there will always be a limit to what can be modified. Even if we were to provide all kinds of hooks like OnDeath, OnRespawn, OnCastSkill, OnMove, OnHeal, etc., there would still be parts that simply can't be extended, like the Item, Skill and Quest properties (because they are structs), or the damage algorithm, or the finite state machines. There would also be UNET related issues because commands couldn't be called by anything that is not a player, so a lot of addons would need helper addons that sit on the player and call commands for them. There would also be additional cognitive overhead because the core would have to deal with addons in a lot of places. UNET is a good example for this system, where we can inherit from NetworkBehaviour and build our custom games with it.

- **Templates:** in this approach, the project is just a template that is meant to be modified in it's entirety. There is no special addon type, people can just hack around in the source files and change what they like to change. The downside of this system is that people can't just download addons from someone else, put them into their folder and run the game, since the modifications are often all over the place. The benefit of this system is that people can modify 100% of it. The simplicity is also worth mentioning, since it's a less complex system and there are no special addon types, hooks or callbacks. Quake 3 mods are a good example for this system, where people would hack custom behaviour directly into the source code.

As usual, we chose the simplest solution and made uMMORPG a template for your own dream MMO.