Unity AssetBundles for VaM 1.xx

MacGruber's Tutorial Series

This tutorial gives you a quick introduction on how to export assets from Unity into VaM.

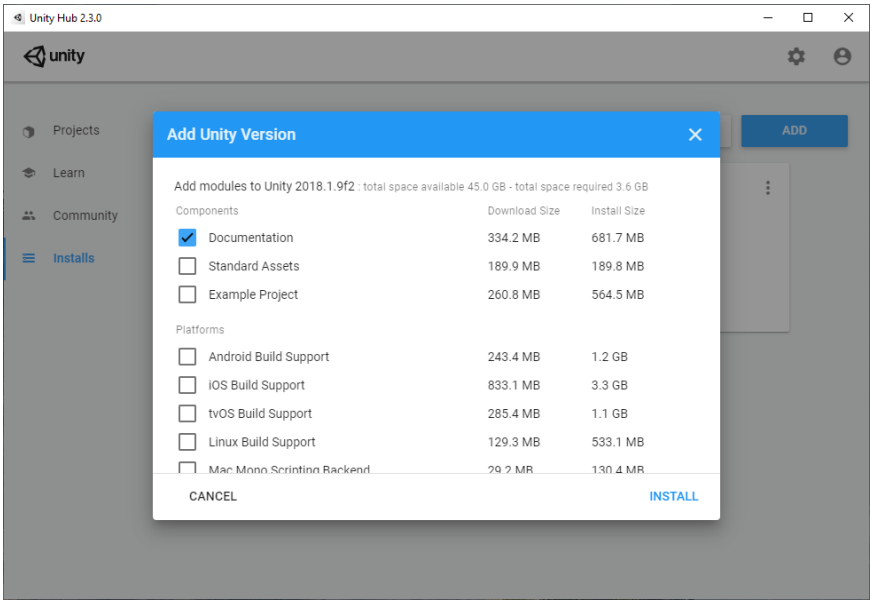
Overview

Install the correct Unity version

Note that **you can't just install the newest version of Unity**, you will run into trouble. The reason is that VaM 1.xx is still being build with an old Unity version back from the year 2018. It can't load anything produced with newer versions of Unity. Assuming you want to make AssetBundles **for VaM 1.xx versions, you will need Unity 2018.1.9** or older. At some point VaM will likely switch to a newer Unity version, though, likely for VaM 2.xx.

I recommend installing Unity Hub, which allows you to have multiple versions of Unity installed in parallel. You might need that once VaM 2.xx is released. It's also a nice tool to manage your Unity projects. Once you got Unity Hub, you can install Unity 2018.1.9f2 with the unityhub-Link below:

- Download Unity Hub: https://unity3d.com/get-unity/update
- Install Unity 2018.1.9f2: unityhub://2018.1.9f2/a6cc294b73ee
- Alternative without Unity Hub, not recommended: https://unity3d.com/get-unity/download/archive



When installing Unity you don't need any of the optional stuff. I recommend to install the documentation, though. There is online documentation, but the servers are insanely slow. Having the documentation local is just way faster.

Setup your Unity project

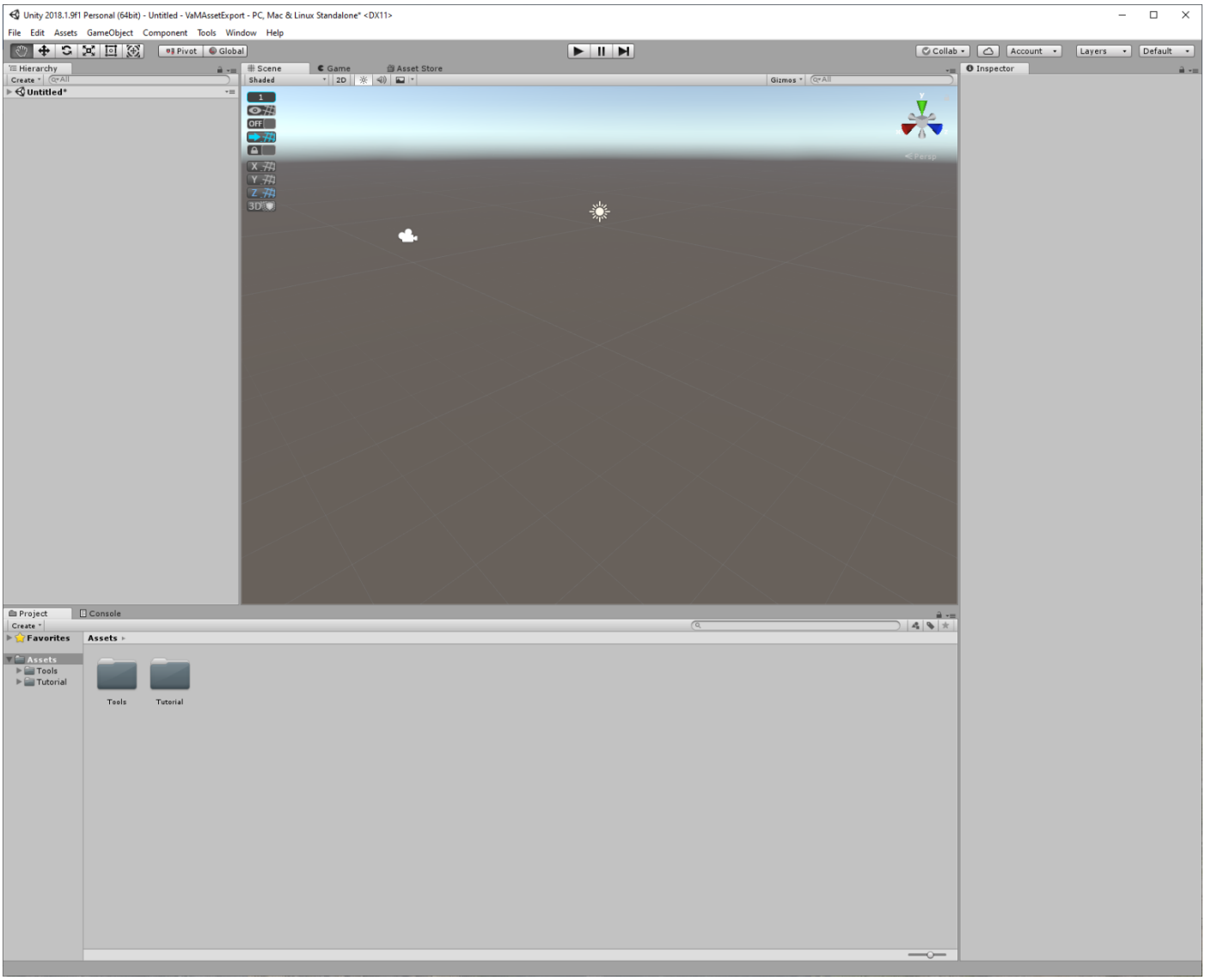
Luckily, I already got a ready-to-go Unity project for you. You just need to download the ZIP and extract it to some nice place on your hard drive.

- UnityAssetBundles-Project-20200307.zip (via MEGA)

This already got a bit of setup done to make the produced assets compatible with VaM VR. Additionally, it includes a number of tools you will find helpful:

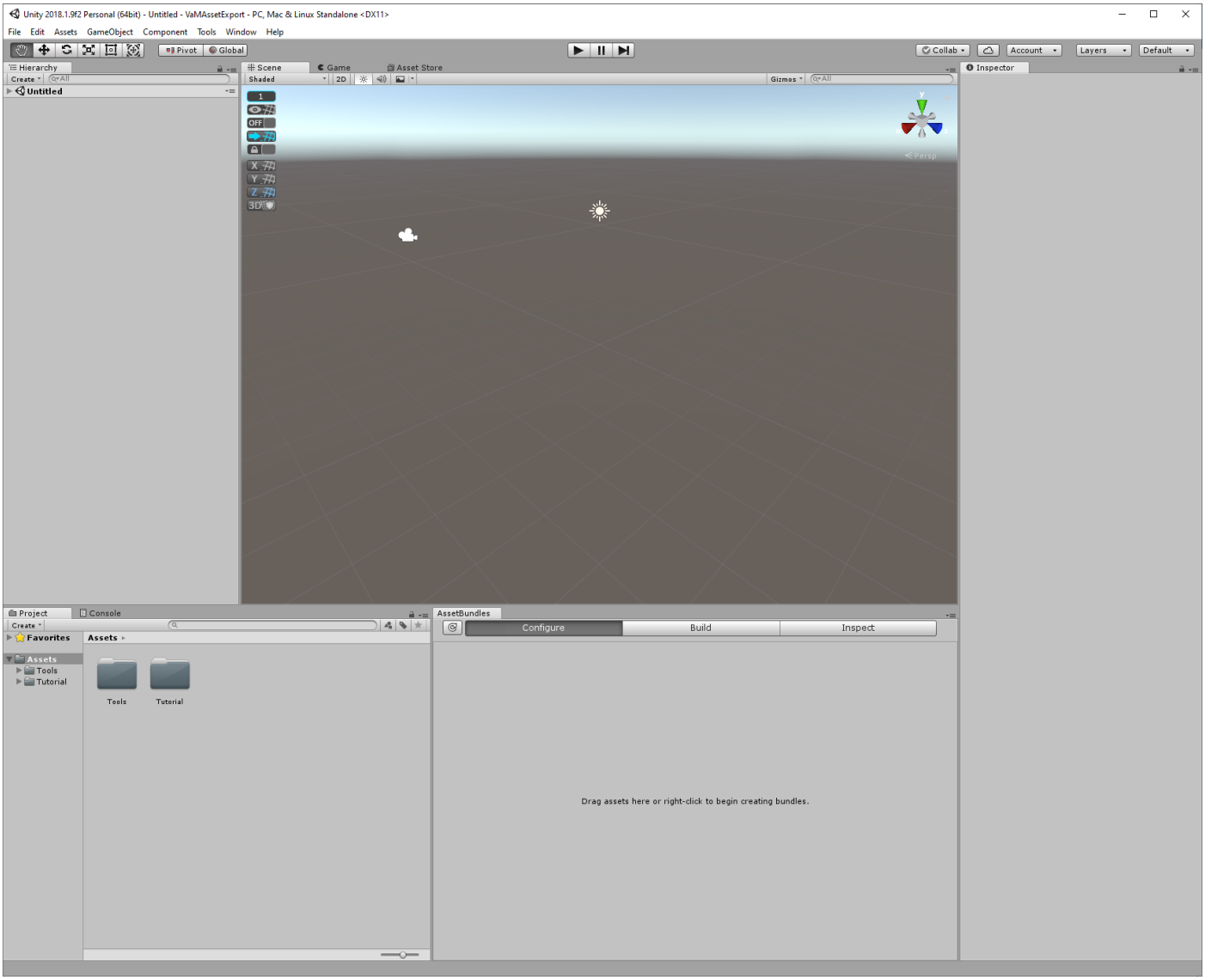
- AssetBundleBrowser (by Unity)
- ProBuilder + ProGrids (by ProCore)
- VaMExporter (by MacGruber)
- EditorOnly (by MacGruber)

Open the project from Unity Hub by adding the project folder to the Projects tab. Assuming you have not used Unity before, the project will look like this when you open it for the first time:

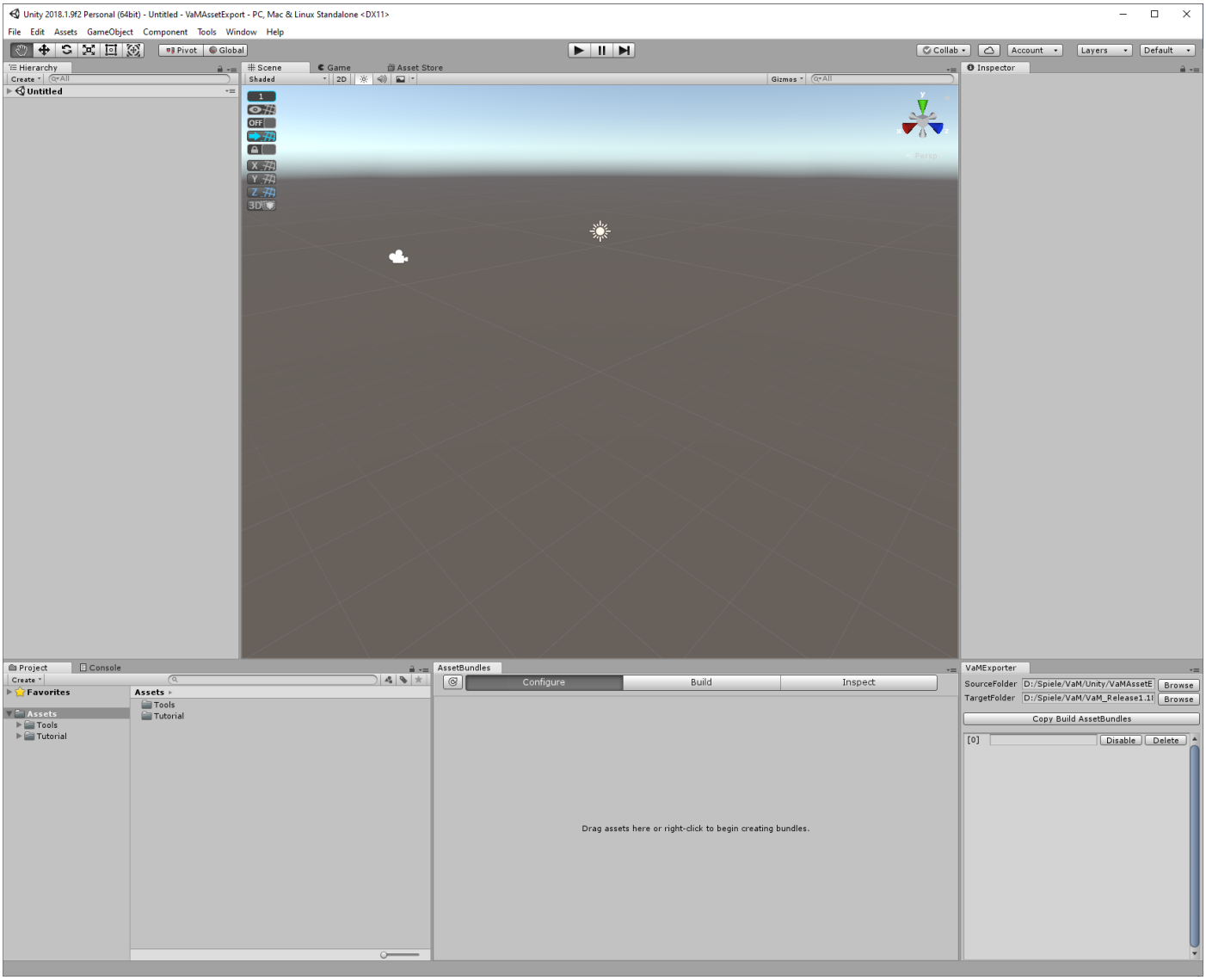


First, we are going to add some tools to your UI layout, so you can access them easily.

Go to *Window* → *AssetBundle Browser*, which opens a new window. Take the tab labeled “AssetBundles” that is in the upper left corner of that window and drag&drop it to dock the window anywhere in the main window. I just put it right of the *Project* / *Console* window. You might want to arrange things differently at a later point once you know what you need to do, but let’s just stick with this for the moment:



Next do the same with *Tools* → *VaMExporter*, this one I docked in the lower part of the *Inspector* window. Also, I resized the windows slightly and pulled the little slider at the bottom of the Project window all the way to the left. It controls the icon size and I hate large icons 😉 You screen should look about like this now:



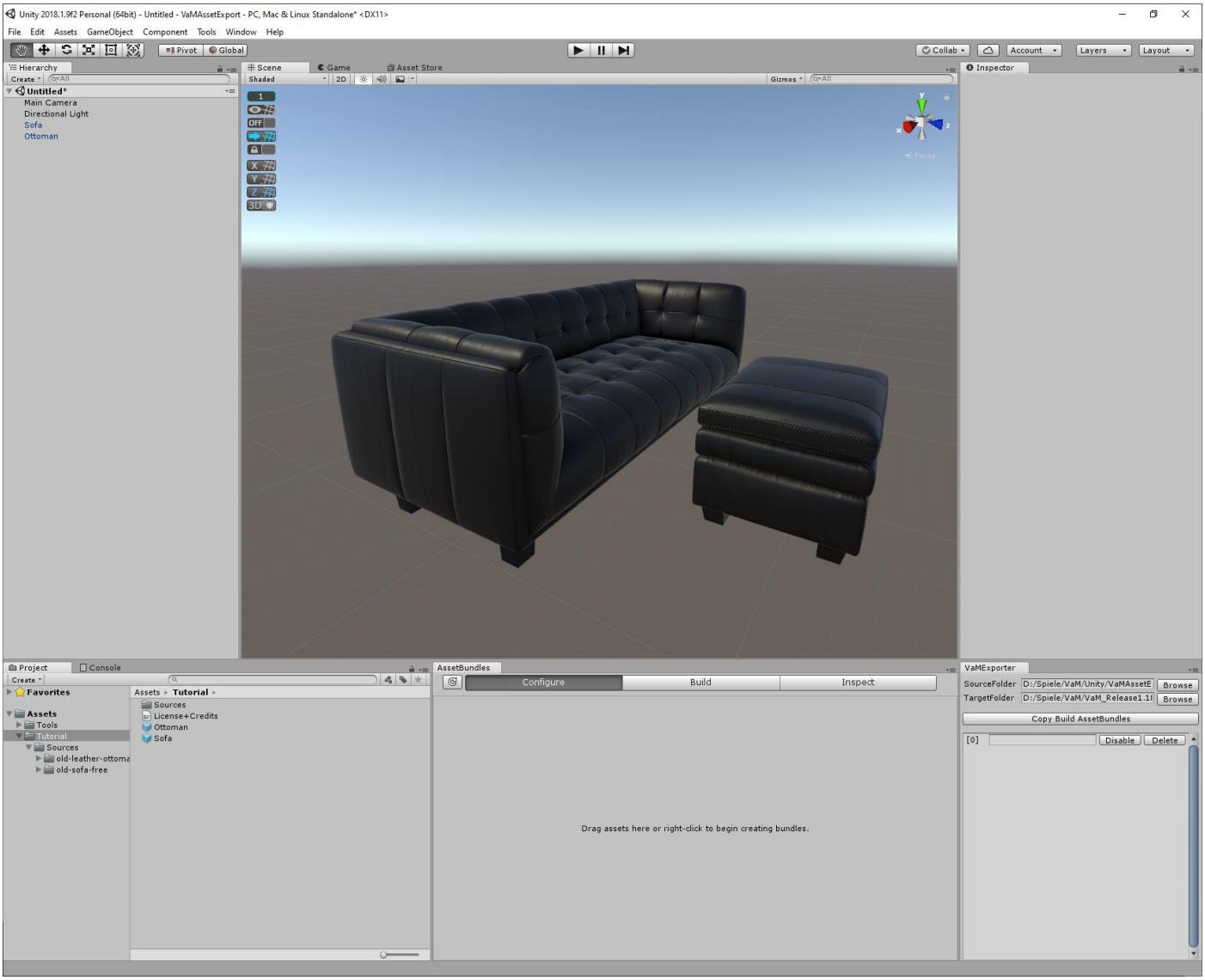
Getting some Assets

For the purpose of this tutorial I have included two nice assets with the project, a sofa and a matching ottoman. These are a based on these two free Sketchfab assets by Visja Filip Rumin: Old Sofa and Old Leather Ottoman. Both are under CC-BY-4.0 license, which made it possible to include them in this tutorial.

I made a number of modifications. I won't go into too much detail here as this is not VaM specific and there should be like a million tutorials around.

- I my opinion the legs of the sofa were not really matching the ones from the ottoman. So, I used Blender to remove the sofa legs and replace them with the much nicer ottoman legs.
- I merged the two separate meshes of the ottoman into a single mesh with two materials. That way we only need a single Unity GameObject.
- When importing the FBX or OBJ files into Unity, it often can't import the materials correctly, textures are not assigned. That happens often, but it's a quick fix. In the *Project* window find the FBX/OBJ and unfold it. There select all the materials and click "Extract From Prefab" from the right-mouse-button menu. That allows you to modify the materials. Using the Unity *Standard* shader is the way to go in most cases. Just assign the textures. Afterwards I renamed the materials to make clearer what is what and tuned the color a bit darker from the original brown.
- To make rendering more efficient, I made the sofa legs use the ottoman leg material, as they are identical now. Since the leg textures were essentially uniform black only anyway (LEGS_albedo and LEGS_metallic), there is no point assigning them. Saves a bit of memory and renders faster.

To see the two assets in Unity just drag&drop the Sofa.prefab and Ottoman.prefab from the *Tutorial* folder into the *Hierarchy* window. Since both assets spawn at 0/0/0, I moved the ottoman a bit so they are not inside each other:

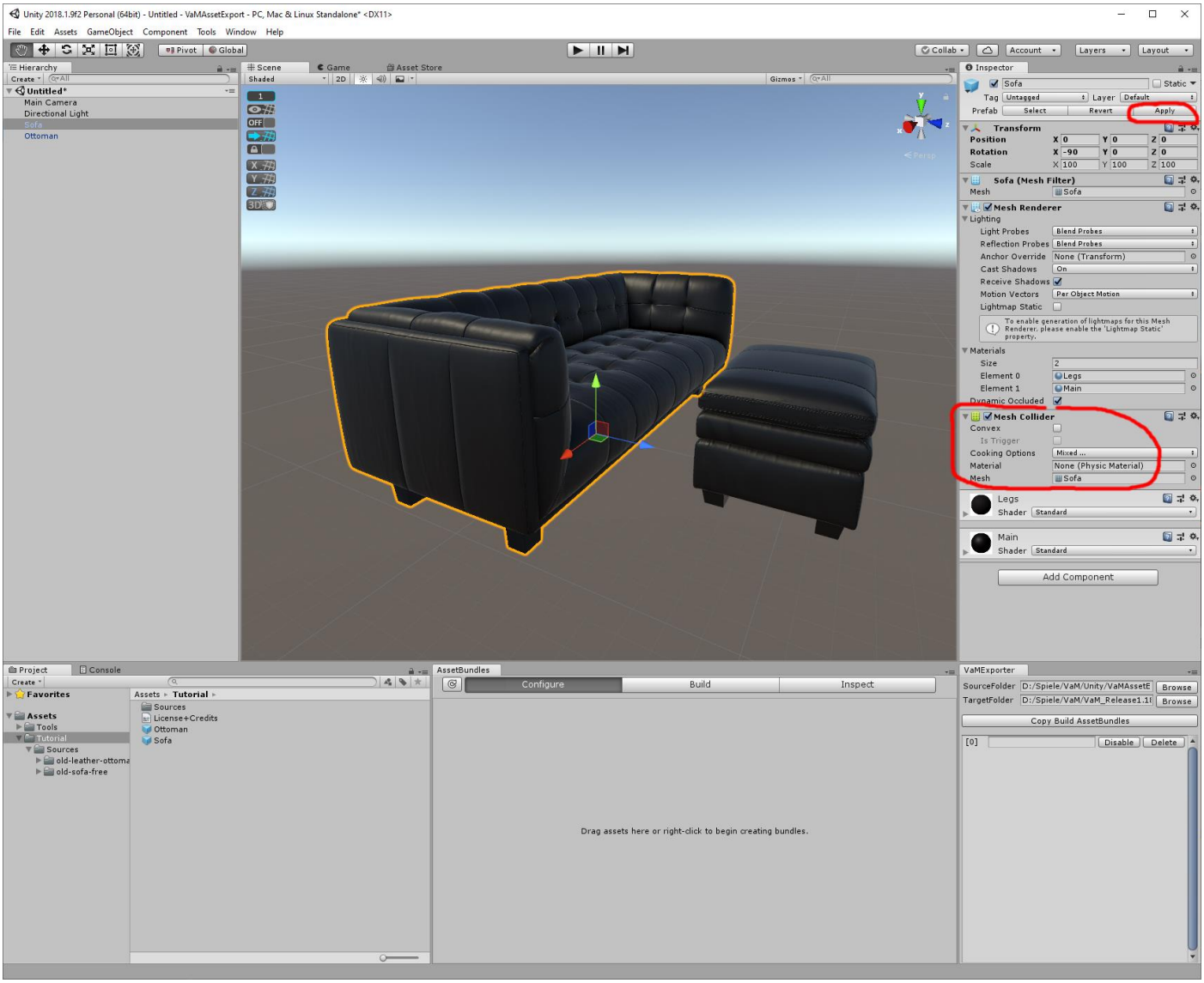


Adding collision

Later in VaM we will want to place a character on our assets, obviously. To make sure our character does not just sink into nothing, we need collision.

Select the sofa and scroll to the bottom of the Inspector window, click the *Add Component* button. Search for *Mesh Collider* and add one. Then click *Apply* at the top, to save the changes into the prefab.

Do the same for the ottoman.



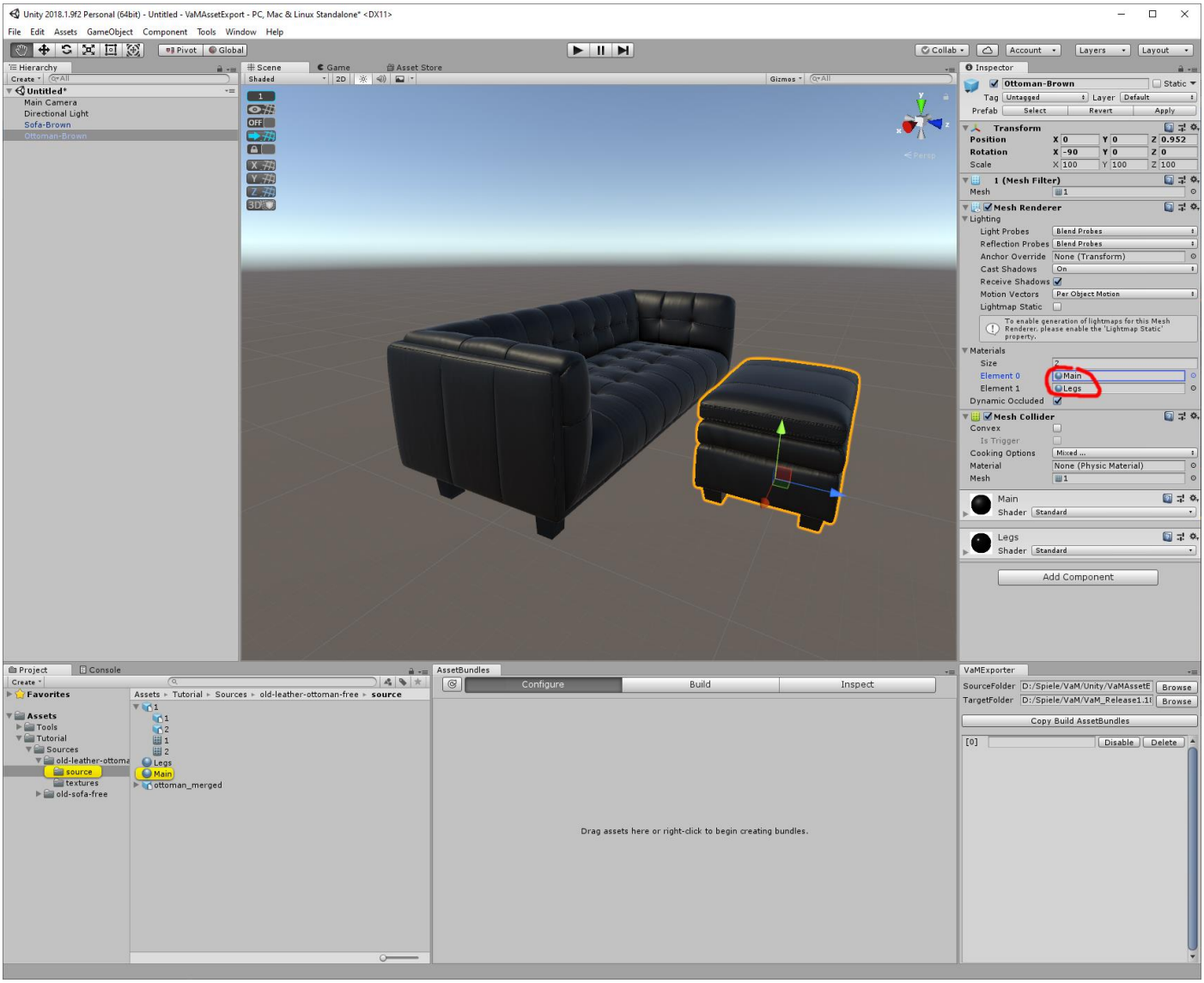
Theoretically we could also use a different mesh for the collision than the one we are using for rendering. A mesh with fewer polygons would certainly be faster, and physics is a bottleneck of VaM. Also, at least when dealing with VaM, it can be helpful to have the collision mesh slightly smaller (like 0.5cm) than the visual mesh. This allows the character to sink in slightly instead of hovering over the asset. However, we are not going fancy today.

Creating a material variation

Let's create a brown version of our assets. Since we will use the same textures and meshes, this won't make the resulting AssetBundle much larger. It's just a nice way of offering a few options to your users without blowing up the package size.

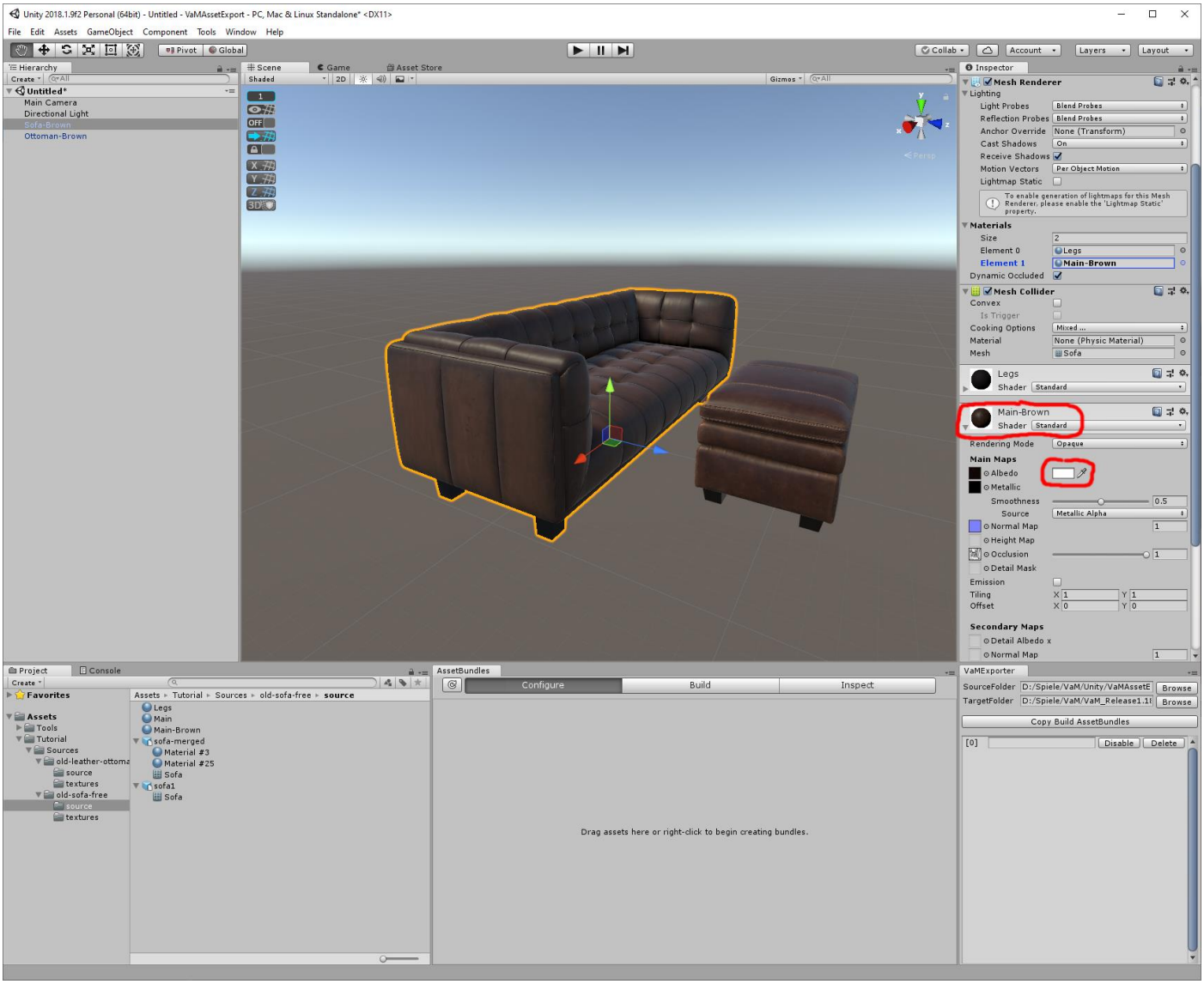
First, I'm going to rename the GameObjects in the scene. This will be the name of the prefab later. While not strictly necessary, it's just less confusing if name of the prefab and name of the contained GameObject match. Let's name these *Sofa-Brown* and *Ottoman-Brown*.

Second, we want to change the color. Since prefabs just contain references to other things like materials, textures and meshes, we need to duplicate the materials. To find the correct material file, you can just click it in the *Inspector* window and the *Project* window will highlight the correct file reference:



To duplicate a material, we select it in the *Project* window and press Ctrl+D. Since I only want to change the leather material, it is enough to copy only the *Main* material. After duplicating I rename the file to *Main-Brown*. This did just copy the material, but our ottoman is still referencing the original material. So, select the ottoman again and drag&drop the *Main-Brown* material onto the material slot in the *Inspector* window. Do the same for the sofa, duplicate the *Main* material, rename it and assign to the material slot.

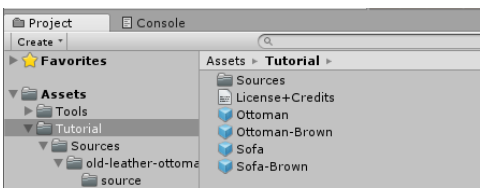
Last but not least we we change the new material to our liking. I just set the albedo color to white, so the get the original color from the brown albedo texture:



Creating Prefabs

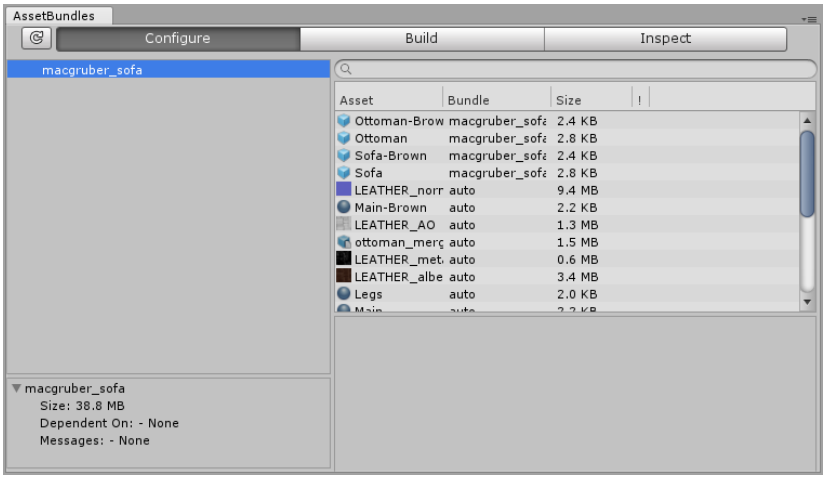
When loading this later in VaM, we want our assets to appear right where the CustomUnityAsset atom is located. Here in Unity that position is 0/0/0. Therefore, we need to position our asset in the scene in the same way we want it located later in VaM in relation to 0/0/0. It can help sometimes to add a little helper object in the scene and position it at 0/0/0 to figure out where something needs to go. In this particular case the mesh is already nicely aligned, we can just move the ottoman back to 0/0/0. No worries about both being inside of each other. Inside VaM these will be separate objects.

Just as you can drag&drop things from the Project window into the scene, you can also drag&drop them back. So, select the *Sofa-Brown* GameObject in the *Hierarchy* window and drag&drop it back into a folder in the *Project* window. This will create a so-called prefab object, which would also contain all the children of the object, if there were any. Do the same for the *Ottoman-Brown*.

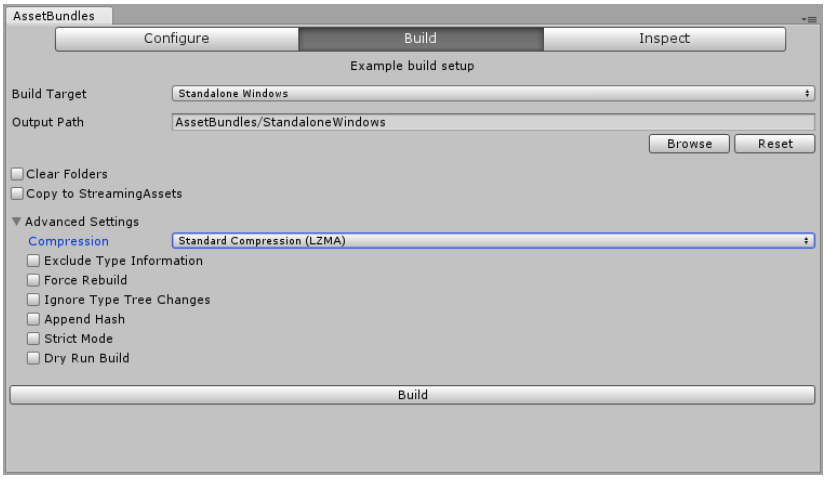


Creating the AssetBundle

Now that we got prefabs for our assets, let's create our AssetBundle. Make sure the *Configure* tab is selected in the *AssetBundles* window. Then select the 4 prefabs in the *Project* window and drag&drop them into the *AssetBundles* window. Unity will ask if you want to create a single combined bundle or separate bundles for each prefab. We want a single bundle. As per convention I recommend to name the AssetBundle with your creator name, an underscore and then the actual bundle name. So, I'm going with *MacGruber_Sofa*. Note that Unity will automatically lower case your filename. We will fix that in a minute. This should look like this now, our 4 prefabs and a list of all the dependencies in form of materials, textures and meshes:



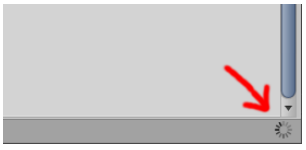
Now switch to the *Build* tab of the *AssetBundles* window:



The default settings are fine here. However, a word on the *Compression* setting:

- **Standard Compression (LZMA)** provides the smallest file size, but VaM will have to load the entire bundle into memory, which is slow.
- **Chunk Based Compression (LZ4)** has a larger file size, but can be loaded WAY faster. If you want only a few selected assets out of a huge bundle, only the actually needed parts need to be loaded and decompressed.
- **No Compression** gives you obviously a large file size, but the AssetBundle creation process is much faster. Loading is also faster, at least when using an SSD. Use this if you want to iterate quickly between Unity and VaM and don't care about file size as you not actually want to share your stuff, yet.

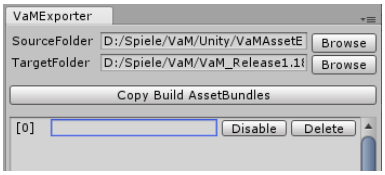
Since this a selection of prefabs and you will only place one or two of them in your scene, *LZ4* compression is a good choice. Now click *Build*, which will take a moment. Unity is done after the progress bar dialogs are gone and the little circling icon in the lower right corner is gone as well:



Using VaMExporter

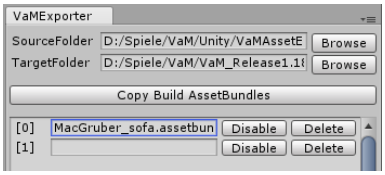
This is just a little productivity tool I made. Usually you would have to go to the AssetBundle output folder now, find your AssetBundle, fix the filename (since it was made lowercase), give it the proper file extension and then copy it to the correct VaM directory. However, since I'm, like most programmers, as absolutely lazy person I made this little tool.

Go to the VaMExporter window. For first time use you will have to tell it where to find its files:

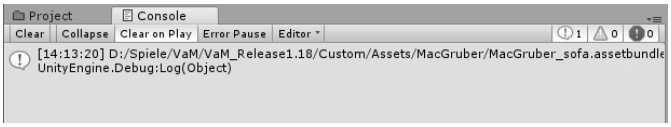


- *SourceFolder* should default to the correct path, assuming you did not change the *Output Path* setting in the *AssetBundles* window. That would be *AssetBundles/StandaloneWindows* inside your Unity project folder.
- *TargetFolder* you have to set. It's the folder where you want the AssetBundles inside your VaM install directory. I recommend *Custom\Assets\YOURNAME*, so that would be *Custom\Assets\MacGruber* for me.

Next to the [0] field enter the name of the AssetBundle. It has to be identical to the name we used earlier, but you can use uppercase letters here. Also you can add your desired file extension here. For assets like this one I recommend *.assetbundle*, while VaM would also understand *.scene*. Certain scripts might use their own extensions, e.g. my Life plugin uses *.audiobundle*. Anyway, we named the bundle *macgruber_sofa* earlier, so I'm entering *MacGruber_sofa.assetbundle*.



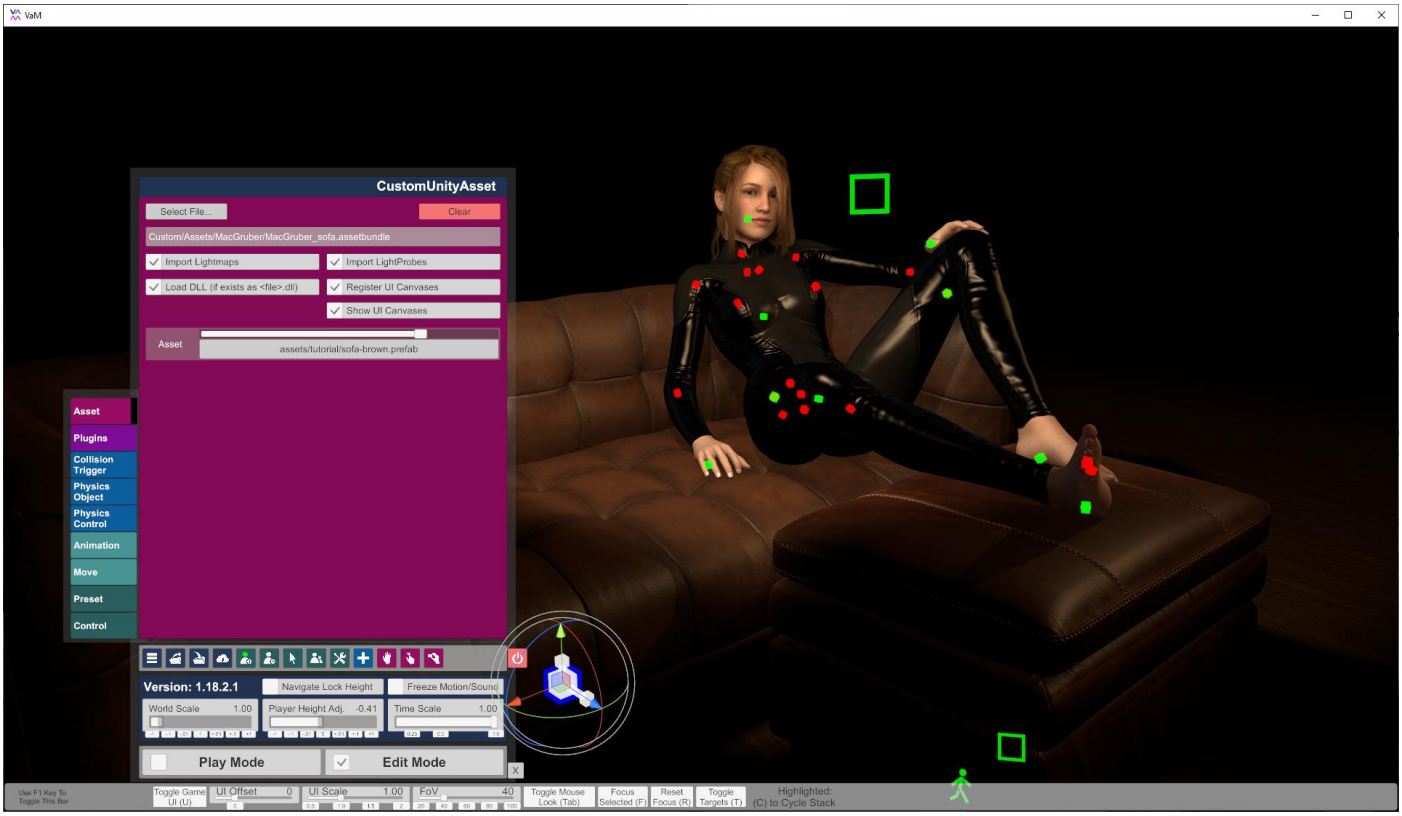
You can enter names for multiple bundles here, temporarily disable some entries, etc. However, we only got one bundle to worry about now, so you can just press *Copy Build AssetBundles*. This will create a log entry in the console.



If you got an error there that is likely because the file was write protected. That might be the case when VaM still got a hold on the file because it is used in an opened scene. If that is the case, temporarily clear the CustomUnityAsset in question or load a different scene.

Loading in VaM

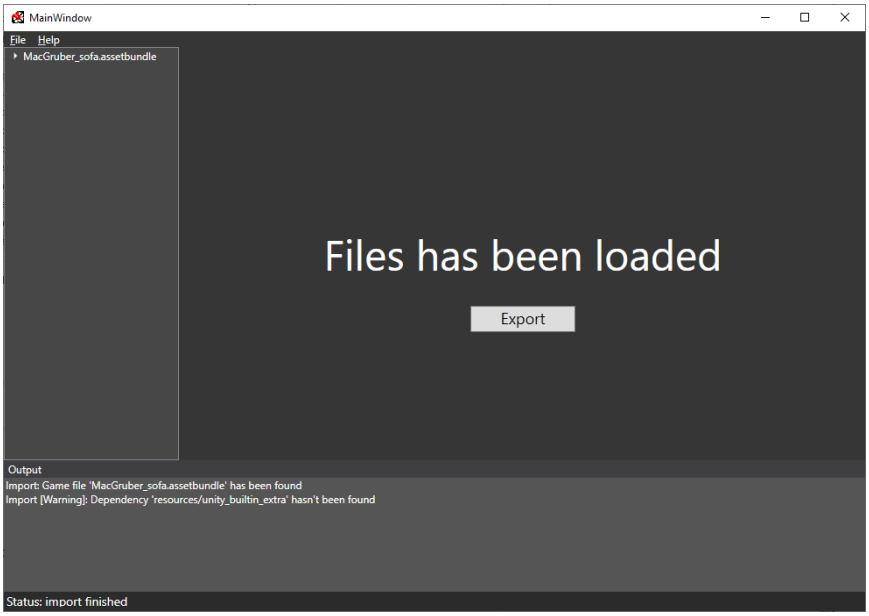
Add a *CustomUnityAsset* atom to your scene, select the AssetBundle just created and select the desired prefab from it. In this scene we got two *CustomUnityAsset*'s each referencing to the same AssetBundle. No worries, VaM will only load it once.



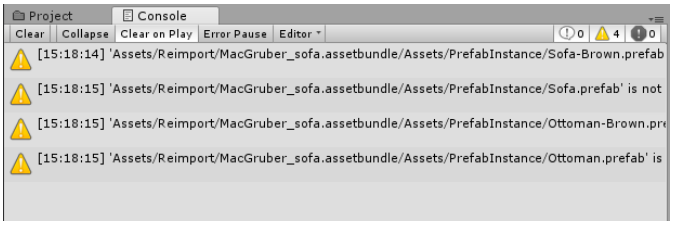
Extracting assets from AssetBundles

So far, this tutorial covered the process of creating AssetBundles with Unity. However, you can also extract assets again from an AssetBundle to reimport them back into Unity. Within limitations at least.

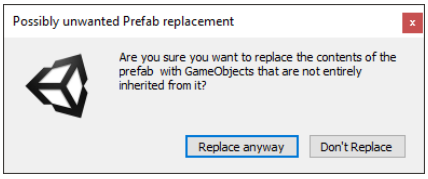
There are a number of tools for this, the only one that kind of worked for me is uTinyRipper. Its easy to use, essentially drag&drop your AssetBundle from WindowsExplorer into the uTinyRipper window.



Press *Export* and it will let you choose a folder where to put the extracted files. You likely want them somewhere inside the *Assets* folder of your *VaMAssetExport* Unity project. Once the export is done, Unity will automatically detect and import the assets. Note that Unity will complain about broken prefab references with some warnings:



However, you can fix that by dragging each of the reimported prefabs into a scene and back out again onto the same prefab. Unity will ask if you want to replace the prefab, click *Replace anyway* and you are done.



License considerations

Now that you know how to export Unity assets and even import them again, you will start your hunt for assets. The most obvious location is the Unity Asset Store. However, if you want to share your assets with the VaM community, you will have to consider the whether you are allowed to do so. For example, the Unity Asset Store license of course allows you to build AssetBundles, but it requires you to build an entire game around those assets. **As sad as it is, sharing just some ready asset from the store will likely not be allowed.** Wherever you find your assets, you will have to check if you are actually allowed to do what you want to do. **Most commercial assets do not allow sharing in the way required for a sandbox game like VaM.** Of course, you could always contact the author and try to make a deal.

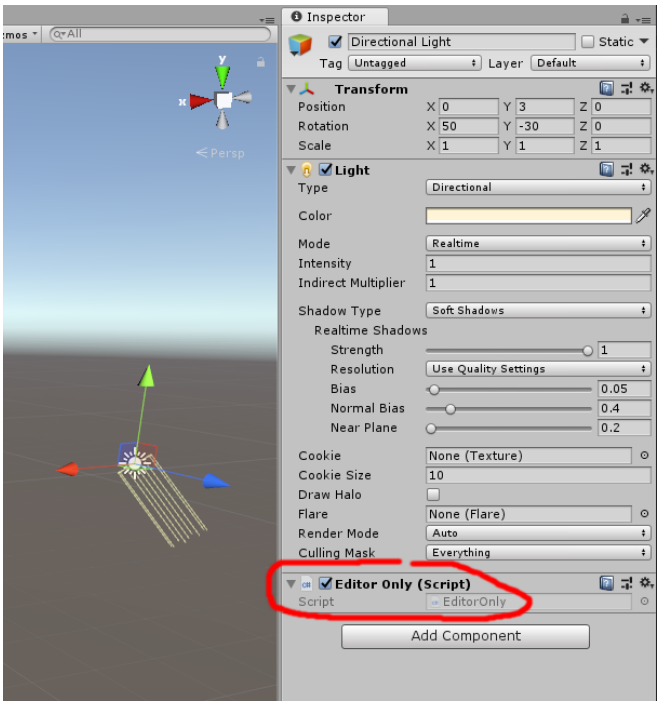
Included Tools: EditorOnly, ProBuilder and ProGrid

As mentioned in the beginning, the ready-to-go project made for this tutorial got some helpful tools included:

EditorOnly

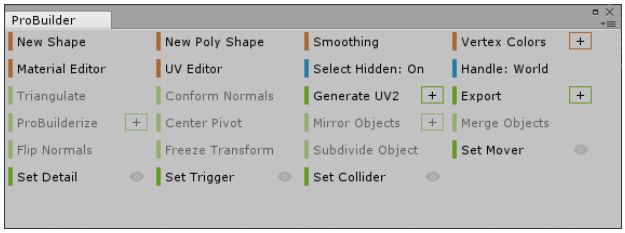
This is a simple script you can put on GameObjects in your scene or prefab inside the Unity editor. Adding this script will cause that particular GameObject and all of its children to not be included when building an AssetBundle.

For example, you might want to have some light sources, cameras or other helpful things in your scene that you don't want to export into VaM. Instead of removing or deactivating them by hand everytime your want to export, you just put this script on them.



ProBuilder

ProBuilder is a simple modeling tool. Mainly it is intended for quick prototyping. However, I build the entire *SecretRoom* environment with this. Especially the auto-stitch feature for texturing is awesome. There are some bugs you might encounter, though, since this is a **rather old version** to be still compatible with Unity 2018.1.9.



Open it via *Tools* → *ProBuilder* → *ProBuilder Window* and dock the window somewhere.

There is a nice to the point video tutorial here, which covers all the essentials in just 5 minutes. No worries about the UI looking completely different, they are just using *ProBuilder* in *icon mode* instead of the default *text mode*. Also, they are using Unity in dark mode, which is not available in the free Unity version.

Once you modeled your mesh and textured it, you can use the *Export* function (with *Export Format* set to *Asset*) to produce a prefab out of it. If needed, this can also be converted back again to *ProBuilder* by using the *ProBuilderize* button.

ProGrid

This is an extension to *ProBuilder*, allowing you to snap things to a grid when moving for precise alignment. When building *SecretRoom* I used a 1cm grid. Usage should be self-explanatory; the buttons have tooltips.