7 Days To Die SDX Tutorial and Help

Table of contents

Introduction

7D2D SDX Tutorials

SDX modding is an enhanced form of modding for the 7 Days to Die game. It allows us to add custom scripts, custom textures and blocks, as well as add entirely new biomes to the world, creating an immersive experience styled to your players and yourself.

While the SDX community has done some amazing things with the game since it's introduction, there lacks a central distribution and documentation aspects to it. This has caused considerable confusion for people who are anxious to get started, since they don't necessarily know how to get started, or even what all of this SDX stuff is about.

That's where this site comes in, to de-mystify SDX and help you get started creating new mods, reaching more people, and finally helping you create a game play that is styled perfectly for you.

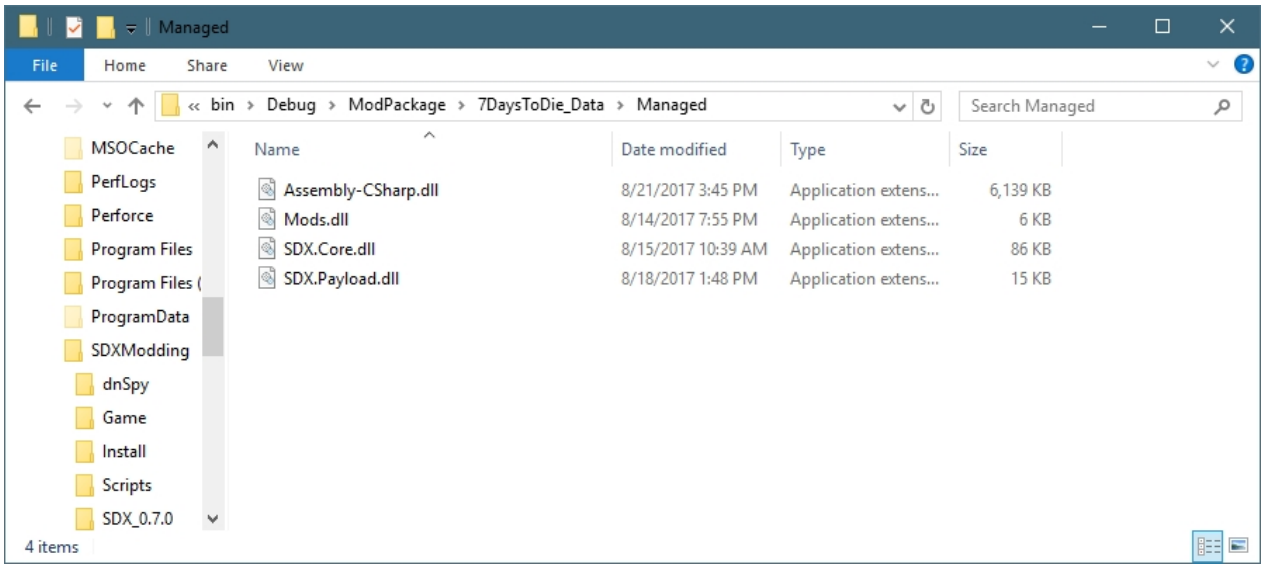
Note: SDX is not EAC compatible. If you attempt to load up SDX with EAC enabled, you will get an error.

Note: SDX Servers require a Client Install for players to connect to, and play.

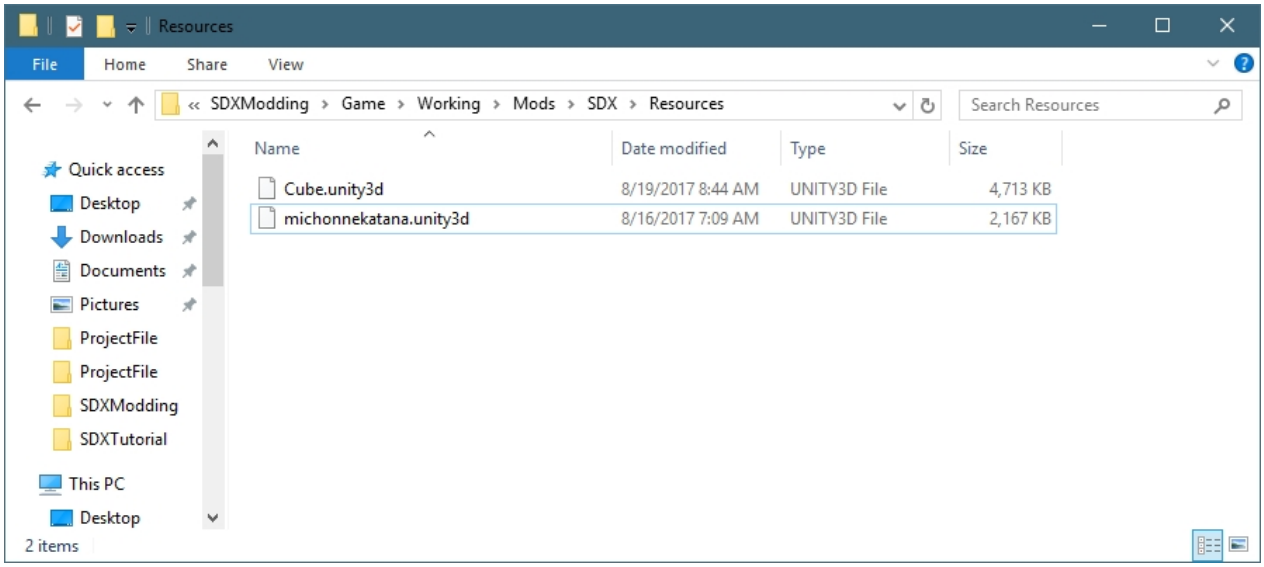
So what is SDX modding?

SDX modding lets you add your own textures, your own blocks, characters, animations, and complex code. It does so by instrumenting the Assembly-CSharp.dll with SDX hooks, allowing it to load up Unity3D texture bundles.

Custom scripts can be written that get compiled into the Mods.dll file, found in the 7DaysToDie_Data/Managed folder, and merged with the Assembly-CSharp.dll at run time.



The actual Unity3D bundles, which are the files where the textures and models are stored in, can be found under your Mods/SDX/Resources folder



What's New

- New Tutorial Sections
    - Beginner
    - Intermediate
    - Advanced

- Added Preliminary SDX and Mod Standards section
- Added Initial Advanced Tools
    - Unity Quick Review
    - Github Quick Review
    - dnSpy Quick Review

Getting Started

Getting set up to do SDX modding is not that difficult. However, you will need a few tools to help you get started.

This section will cover what your basic system requirements are before beginning. The more complex a mod

is, the more memory you will need, above and beyond what you need to play the base game.

In addition to the system requirements, we also list the tools that you'll be using, where to get them, how to install them, and what they'll do for you.

The list of tools you'll need may look big and confusing, but we'll take a slow approach in showing you what each tool will do. To make getting started easier, we created an SDX Modding folder structure, which you can download to get you started. Inside of that download, there's everything you need to get started on building your first sample mod, the Katana Mod.

A Video Tutorial for the following sections can be found here, created by Xyth.

System requirements

The following are the System Recommendation for 7 Days to Die base game. SDX is very memory heavy, so use this as a guide

RECOMMENDED:
OS: Windows 7 or higher
Processor: 3.0 Ghz Quad Core CPU or faster
Memory: 16 GB RAM
Graphics: 2 GB Dedicated Memory
DirectX: Version 10

Initial Setup

In order to be fully successful with SDX, it's strongly recommend following this Getting Set up guide close.

You'll want a 100% vanilla install of 7 Days To Die, without any mods or any additional changes.

Getting the SDX Modding Kit

This tutorial will guide you through step-by-step on how to make your first SDX mod, and set it up for distribution.

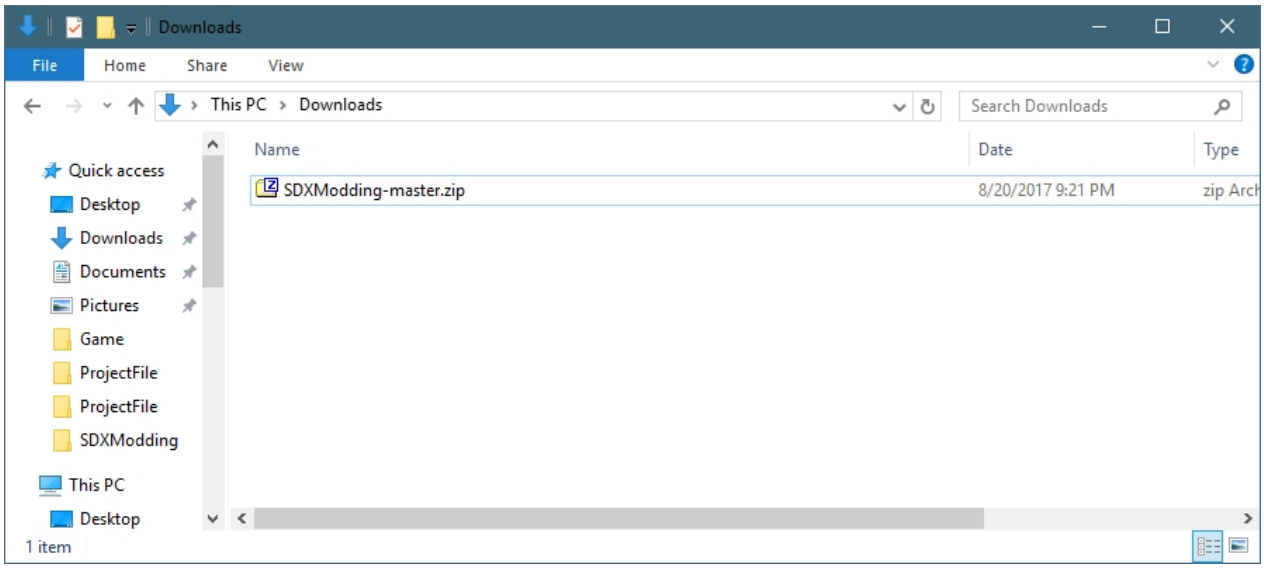
In order to do this, some software needs to be installed and set up. We've listed the tools you should install, including download links.

Most of the tool chain can be downloaded from the SDXModding Github depot, by downloading this link:

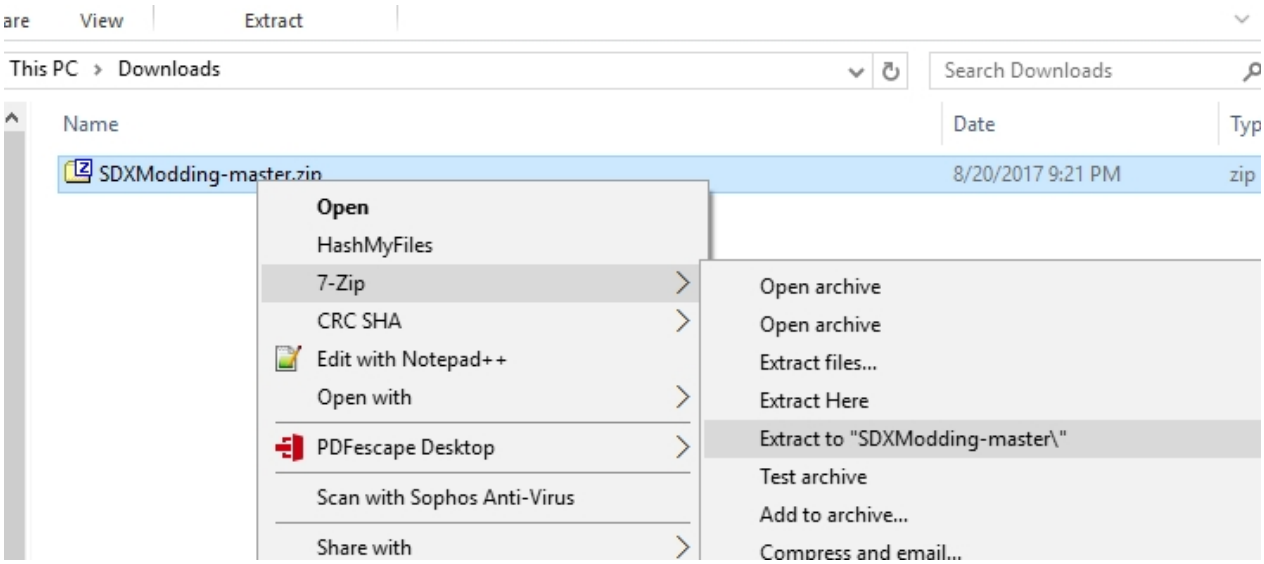
DIrect Download Link: https://github.com/7D2DSDX/SDXModding/archive/master.zip

The SDXModding Download Package contains everything you need to get started with SDX mods.

Once downloaded, go to your Downloads folder



Right click on SDXModding-master.zip, and extract:

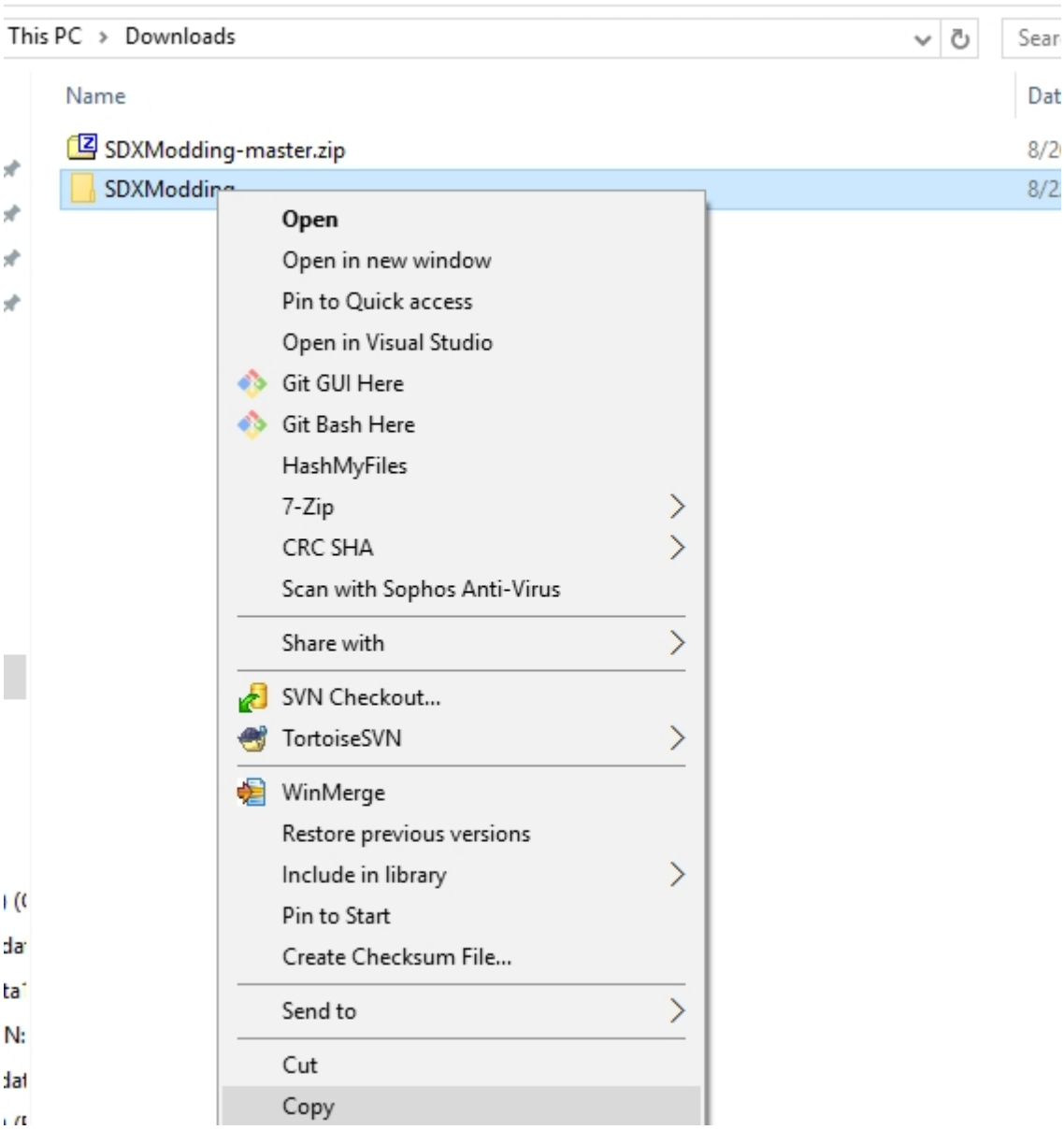


Once extract, right click on the new SDXModding-master folder, and click on rename:

Rename the folder to be SDXModding

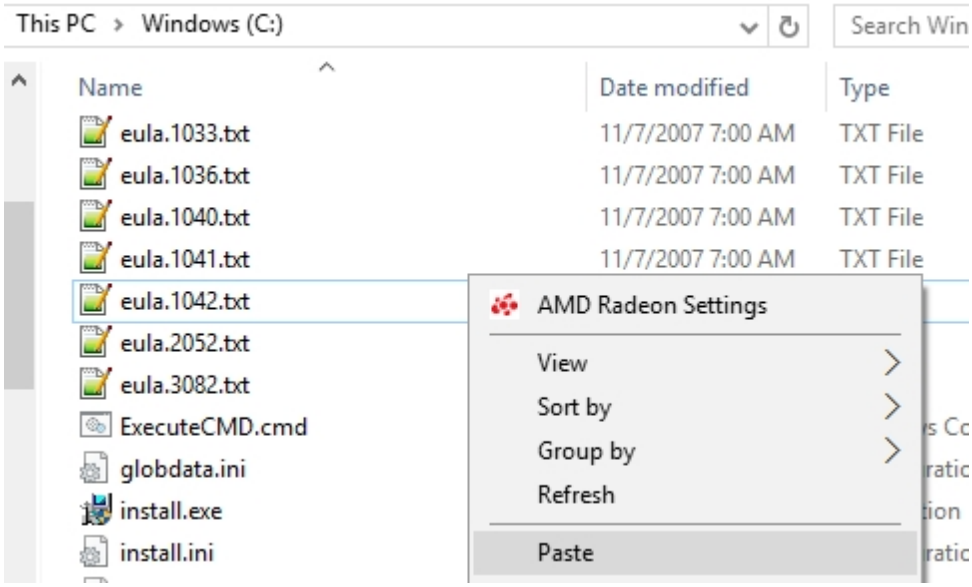
Now we want to copy it to your C:\ or D:\. For the purpose of this Tutorial, it's assumed to be under C:\ or D:\

Right click on the SDXModding folder, and select Copy

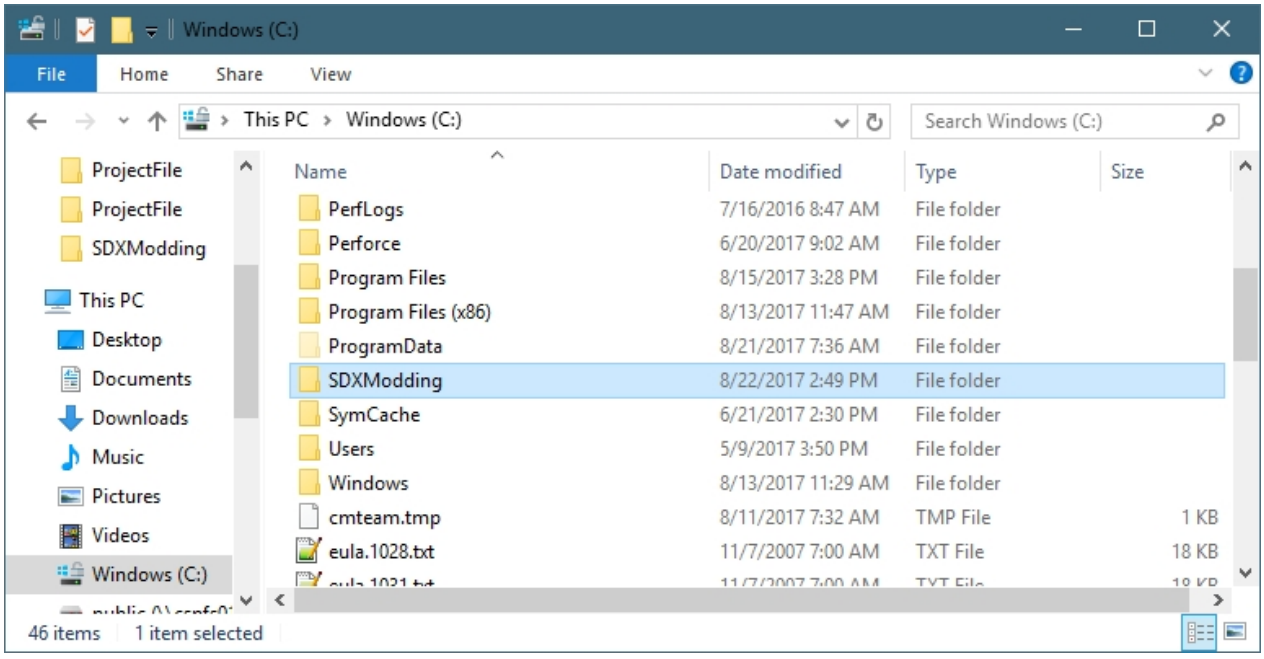


In Explorer, navigate to C:\ or D:\.

Right click on the C:\, and select Paste



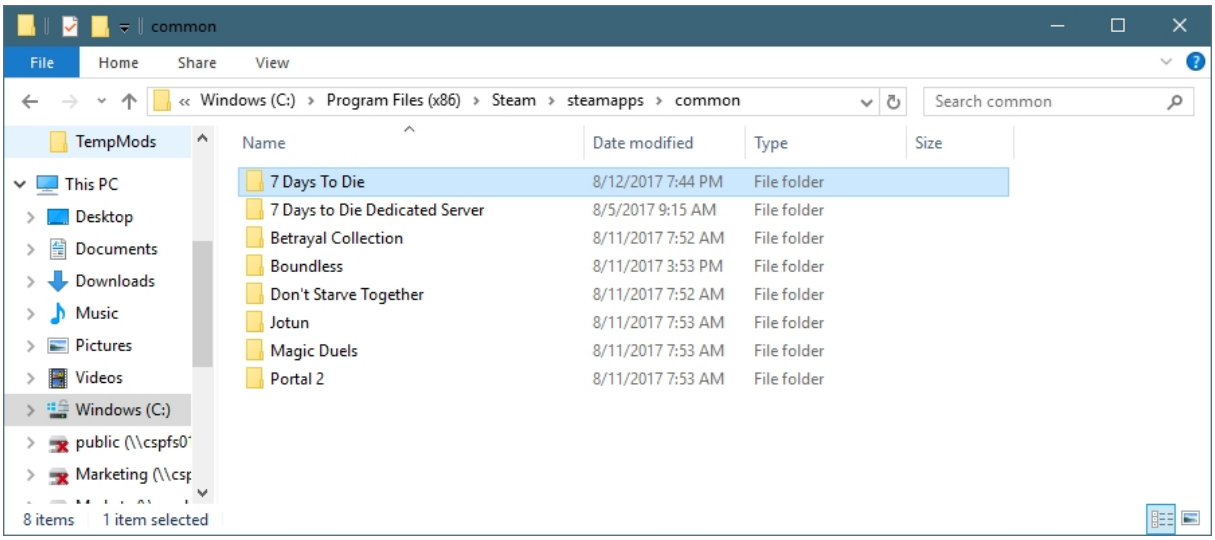
You should see something like this:



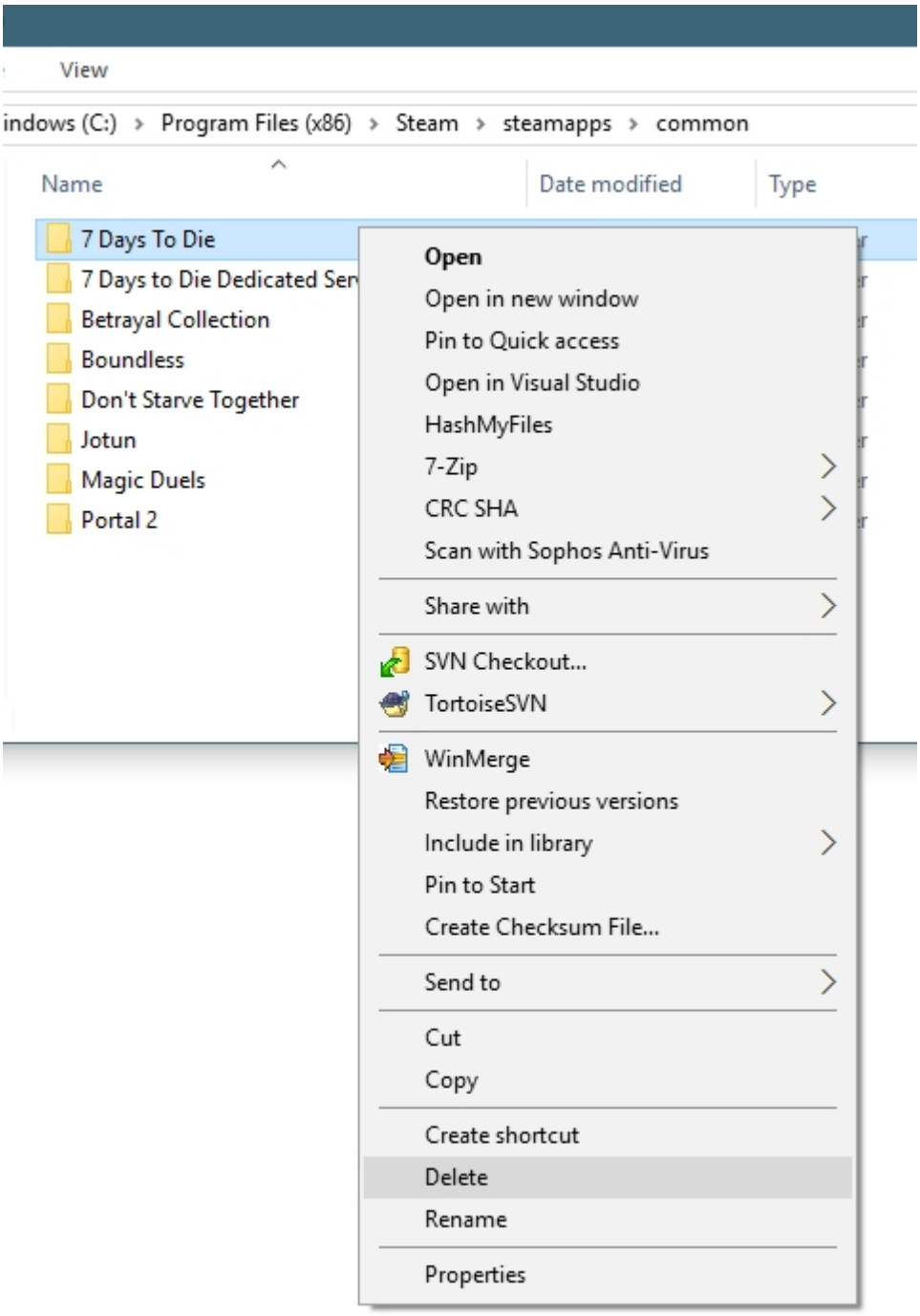
Starting off Clean

NOTE: This will remove any mods or any changes you've done to the game.

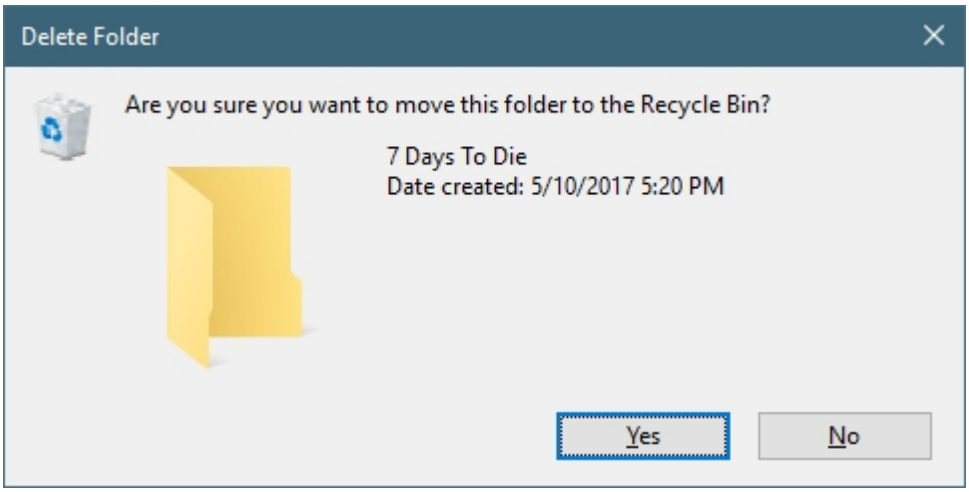
Using Windows explorer, navigate to your Steam Folder, which is by default "*C:\Program Files (x86)\Steam\steamapps\common*"



Right Click on the "*7 Days To Die*" folder, and select Delete

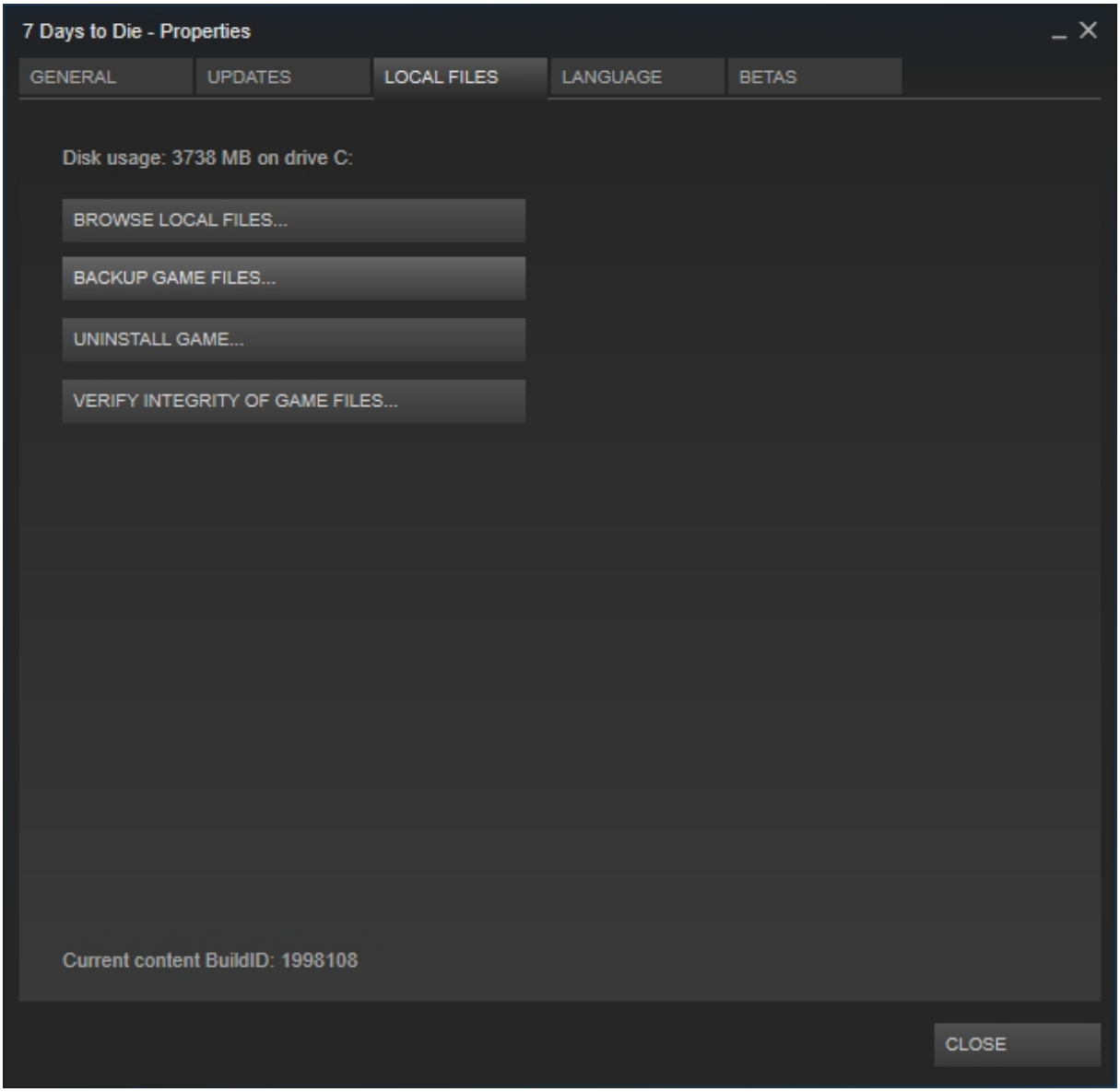


Confirm that you want to delete the "*7 Days To Die*" folder

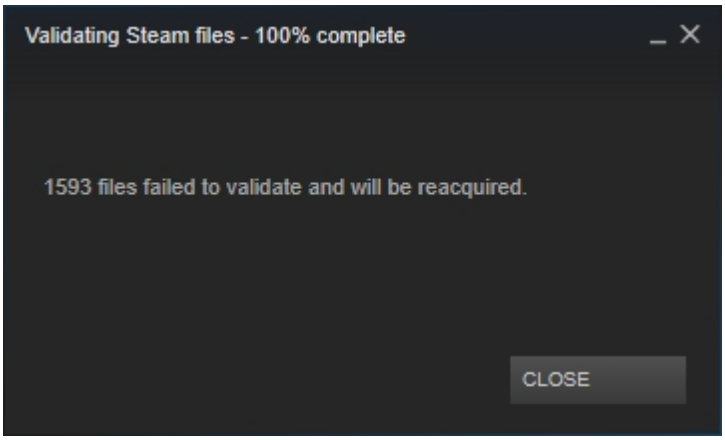


In Steam, right click on the "*7 Days to Die*", and click on Properties

Click on the "*Local Files*" tab



And click on "*VERIFY INTEGRITY OF GAME FILES*"



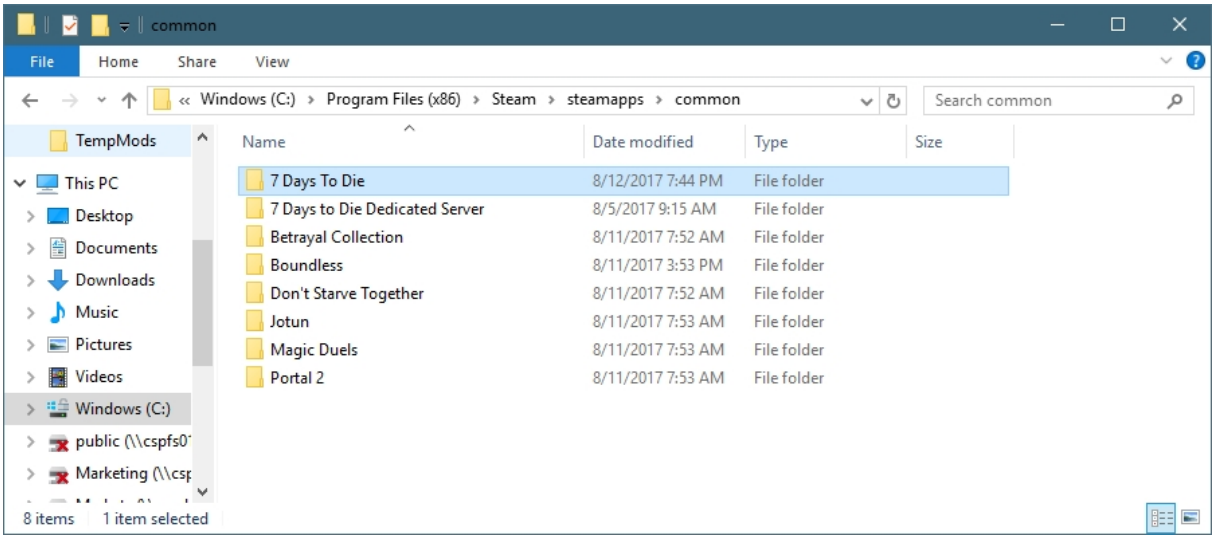
This will download a fresh install of 7 Days to Die of the latest stable release.

Making a Clean Backup

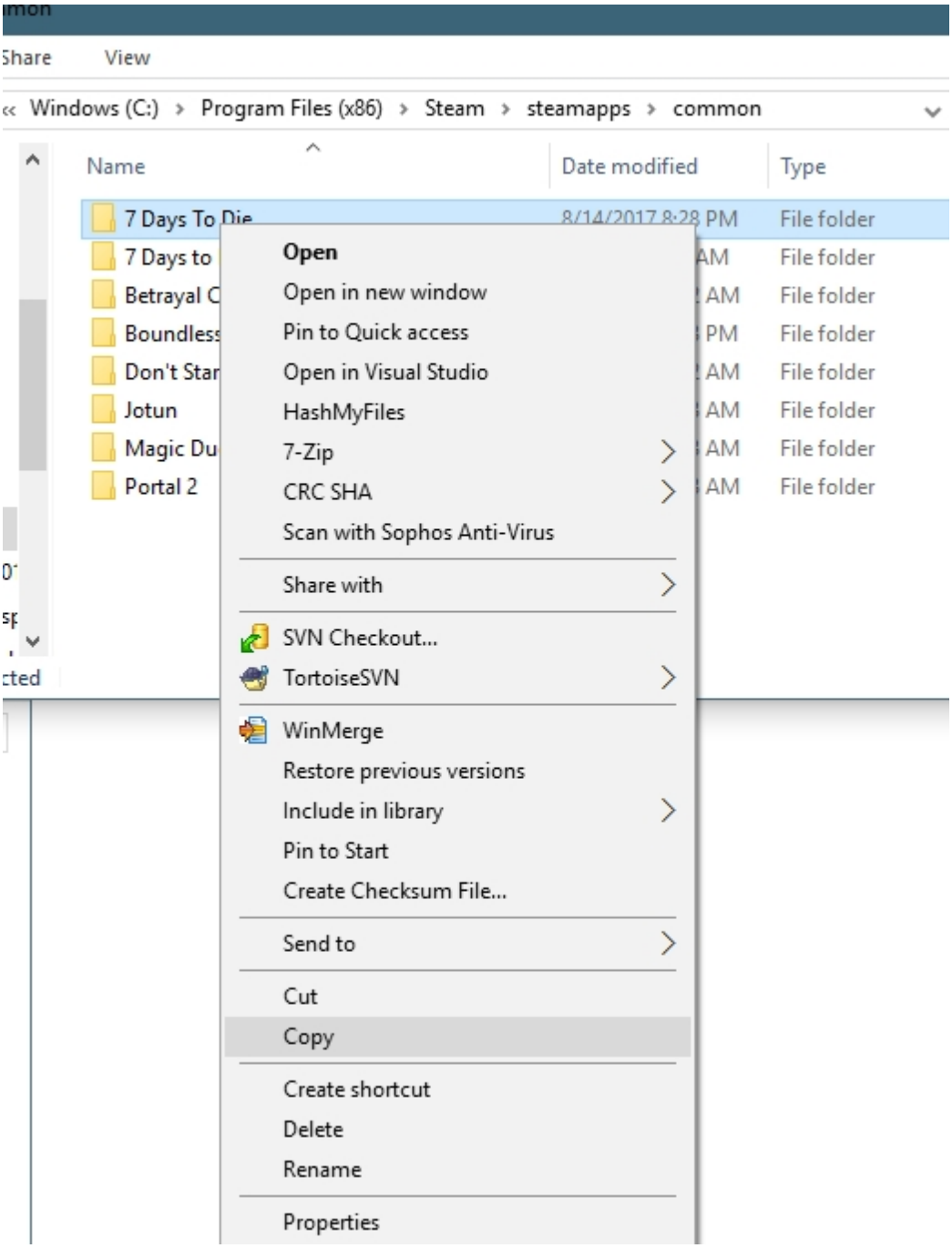
One you have completed the "Starting off Clean" section, it's time to make two copies.

Why two copies? One copy will be used as a plain vanilla version. Since Steam will auto-update if a new release is pushed out, you may not be ready to jump to that new version just yet, especially if you are learning how to mod. The other copy will be your working copy, where you'll be applying mods.

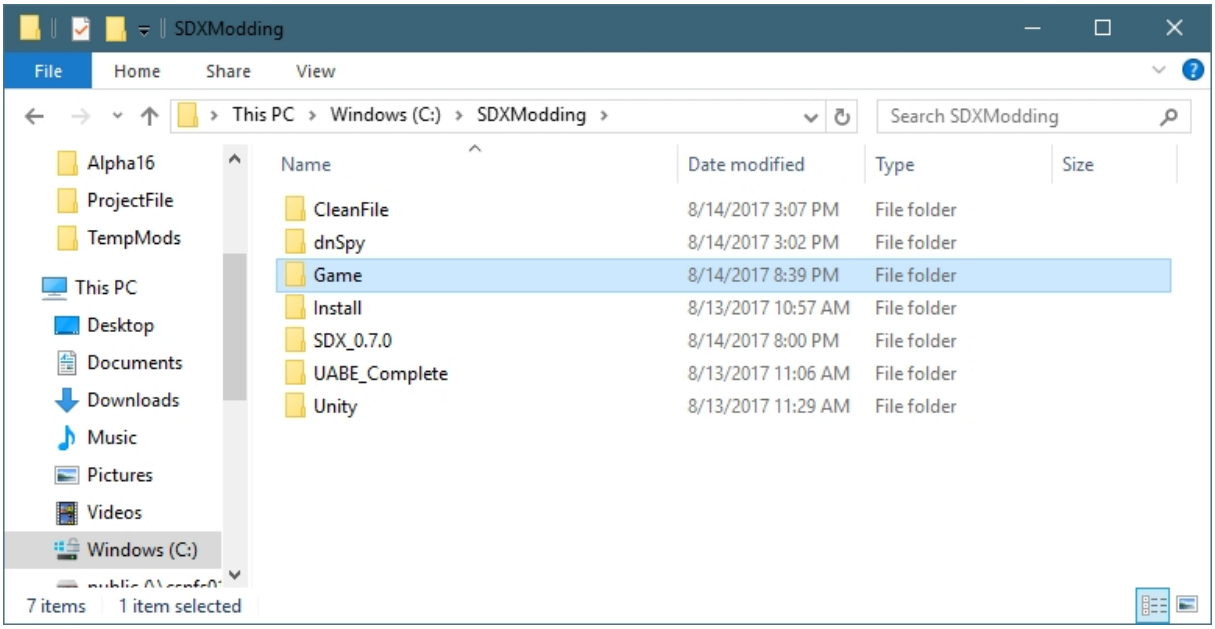
Using Windows explorer, navigate to your Steam Folder, which is by default "*C:\Program Files (x86)\Steam\steamapps\common*"



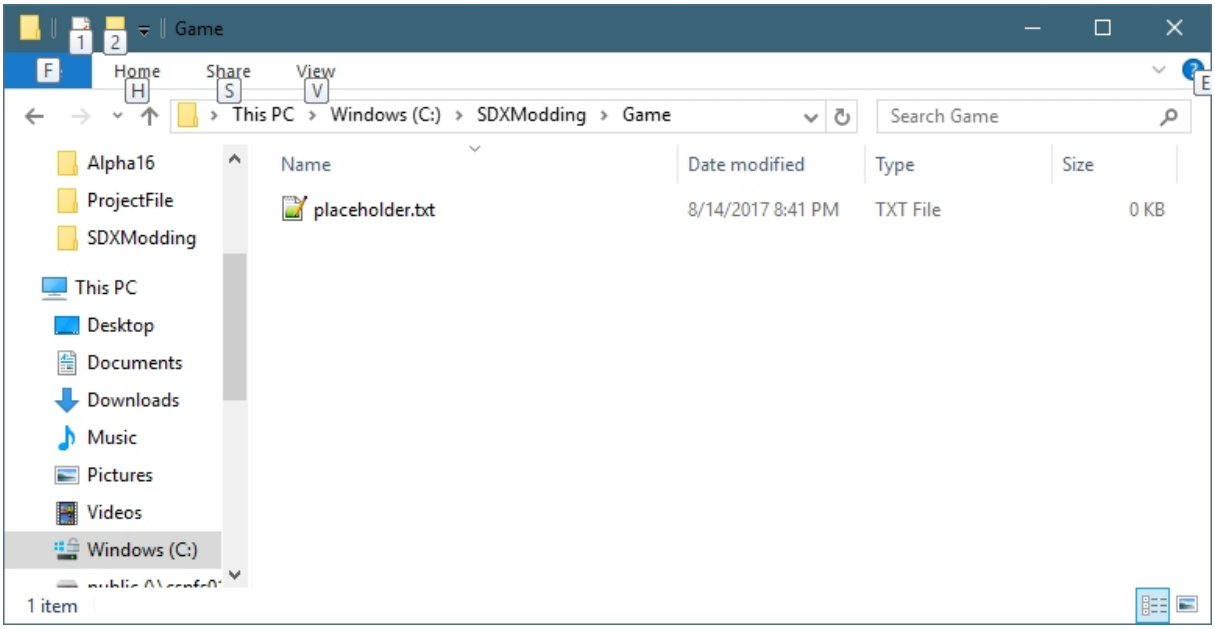
Right Click on the "*7 Days To Die*" folder, and select Copy



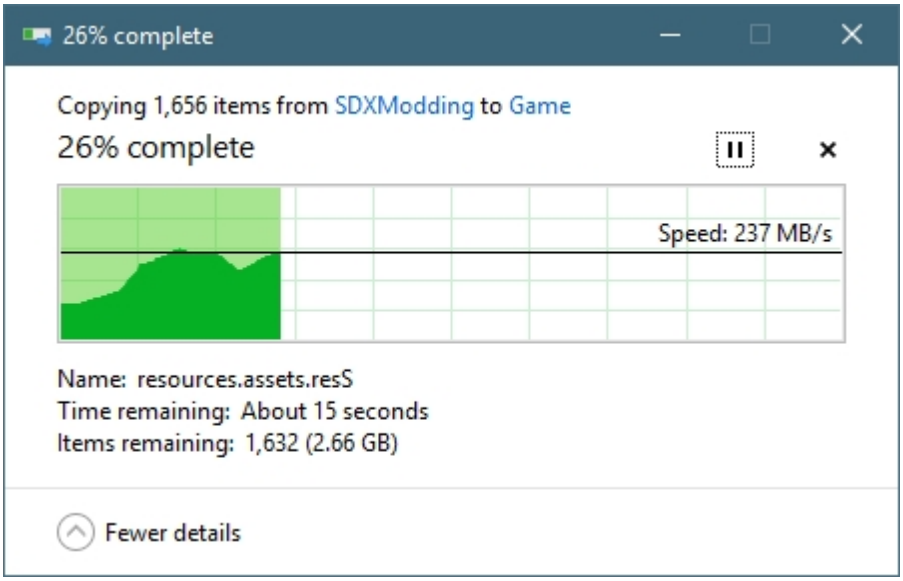
Open up an Explorer Window where you have installed the SDXModding, such as C:\SDXModding\Game\



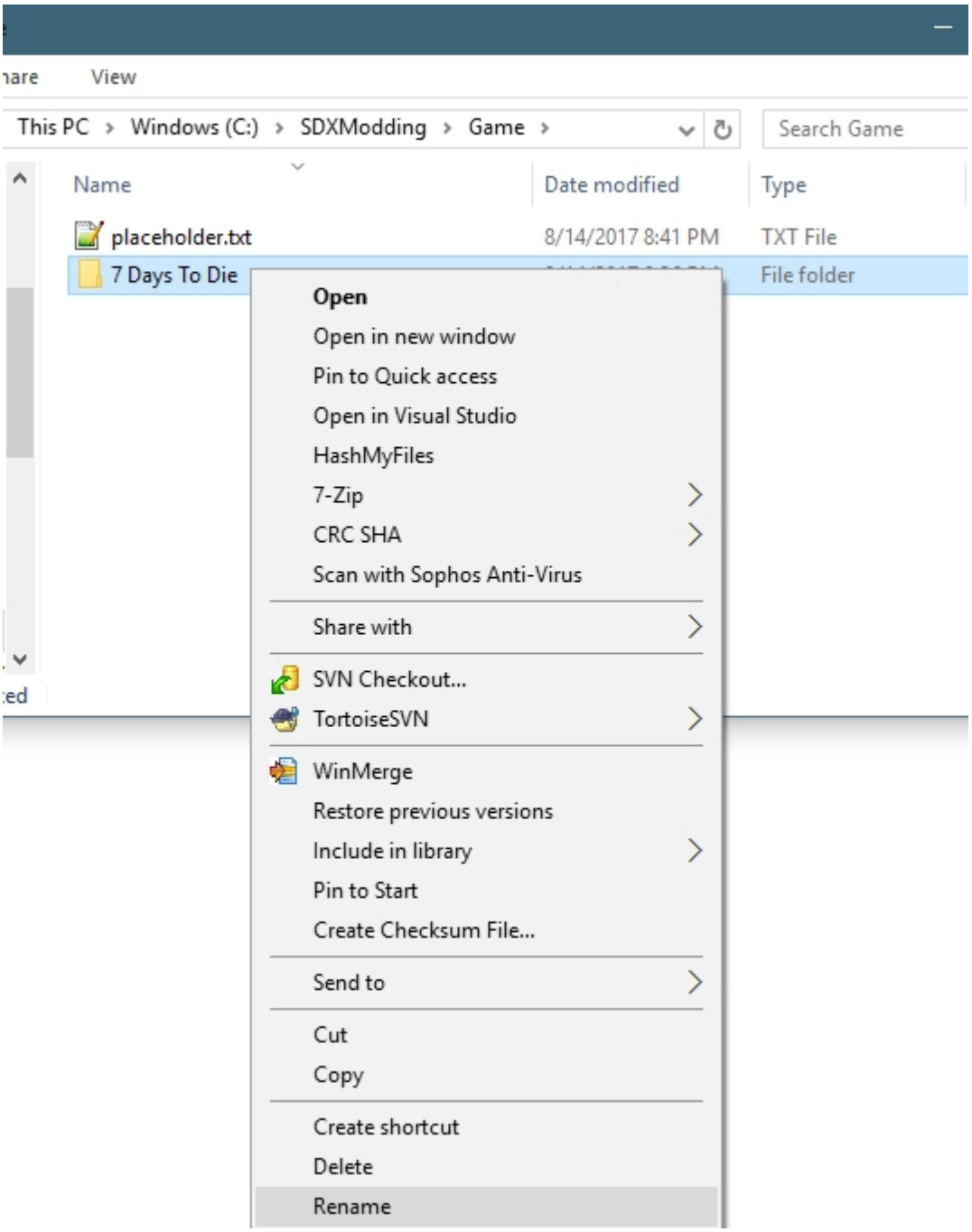
Double Click to open the "Game" Folder.



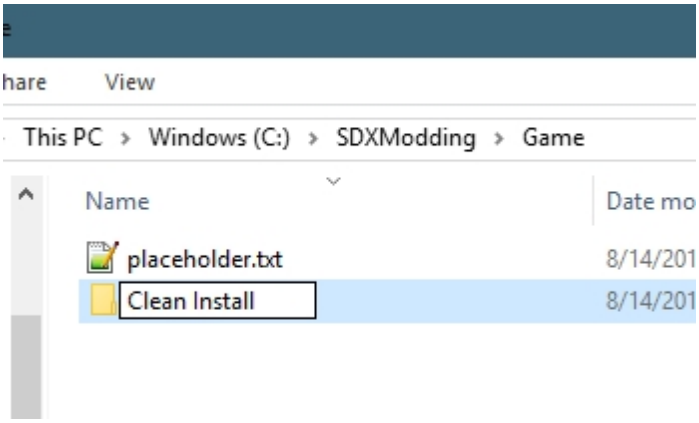
Right click on the folder, and select "*Paste*". It will then make a copy of the game:



Once it's finished, we'll rename the folder to "Clean Install". Right click on "7 Days To Die" in the "C:\SDXModding\Game\" folder, and select "Rename"



Rename the folder "Clean Install"

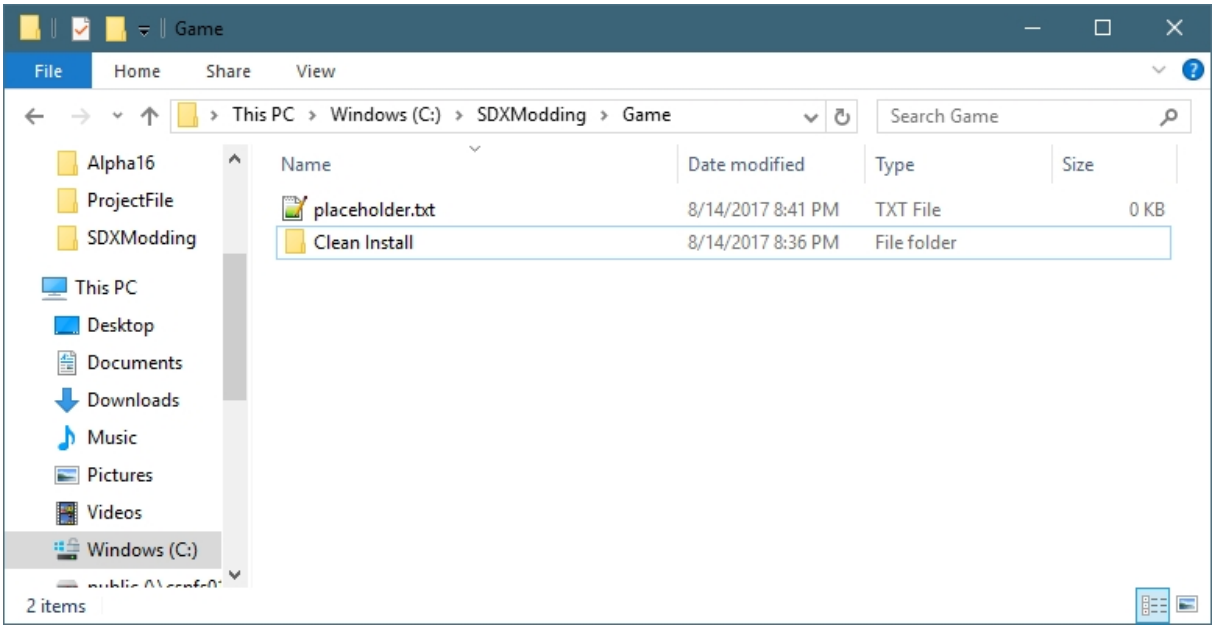


You now have a perfectly preserved copy of the game.

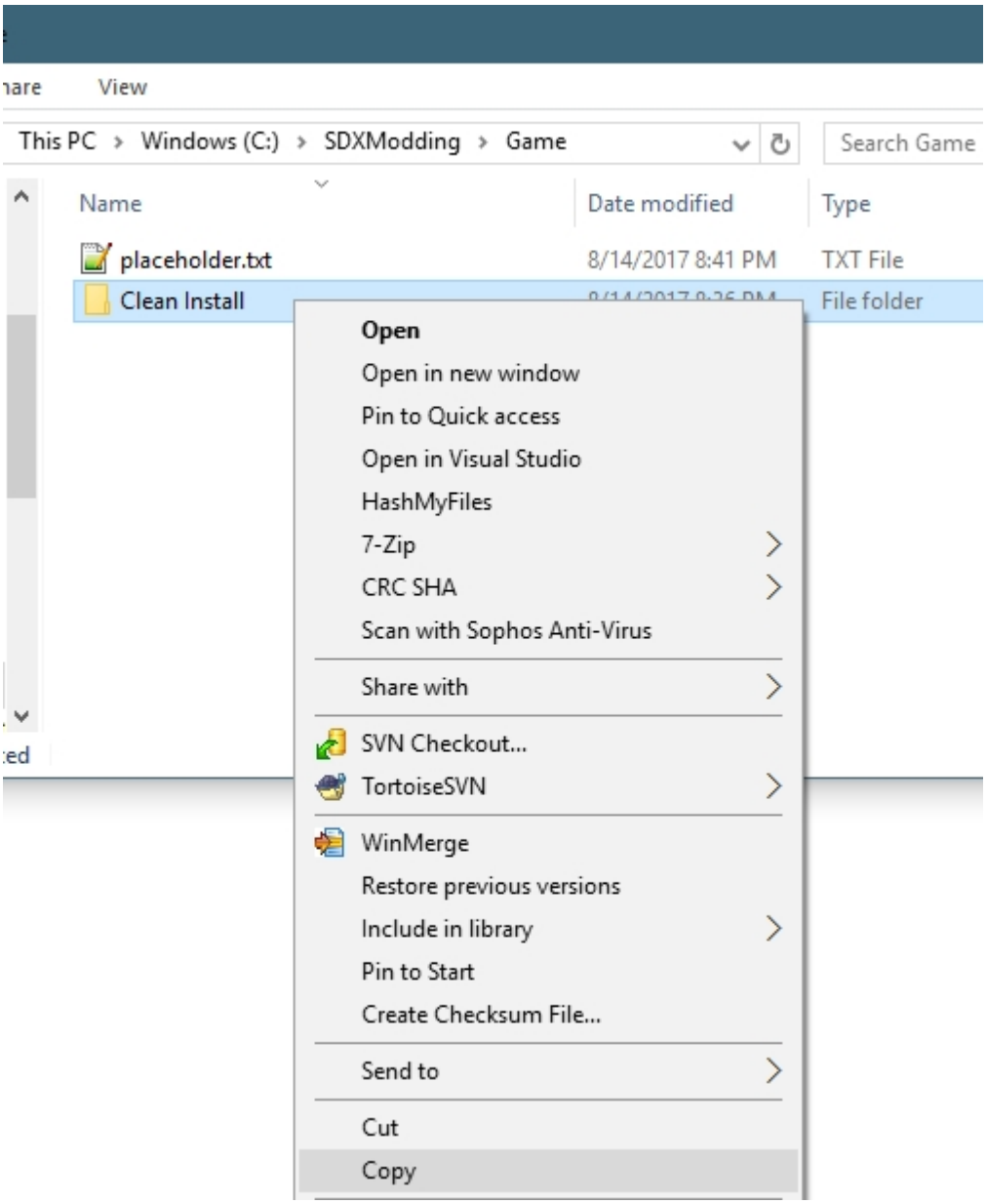
Making a Working Folder

The second folder you will create will be called a Working. This is where you will be adding your SDX mods, trying out new things.

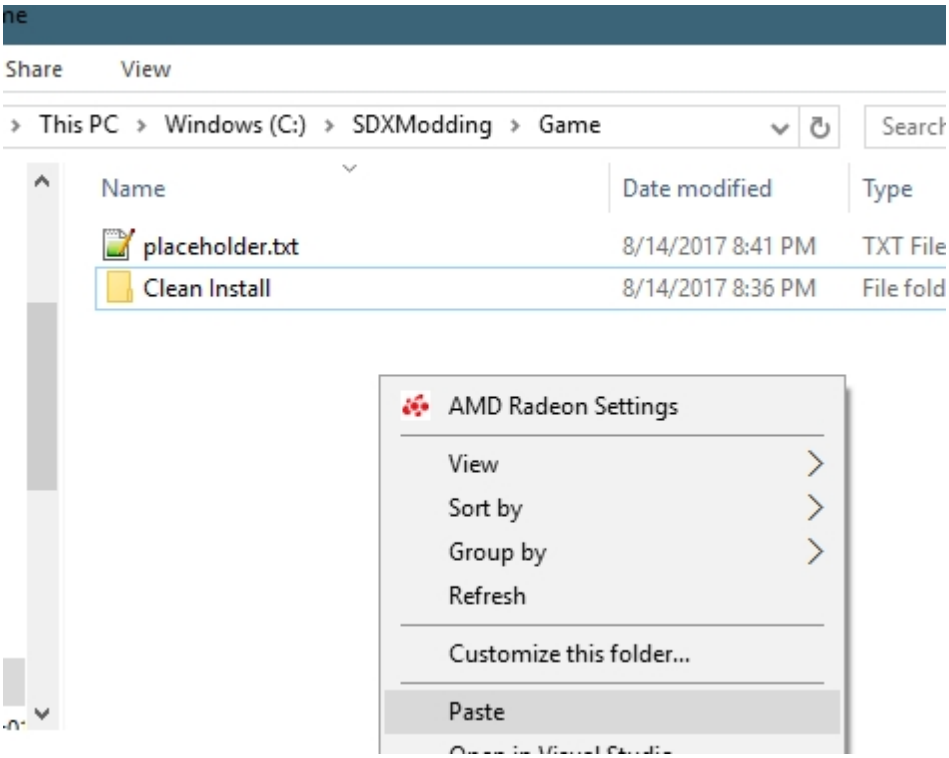
Using Explorer, navigate to your "*C:\SDXModding\Game*" Folder.



Right click on the "*Clean Install*" folder, and click on "*Copy*"

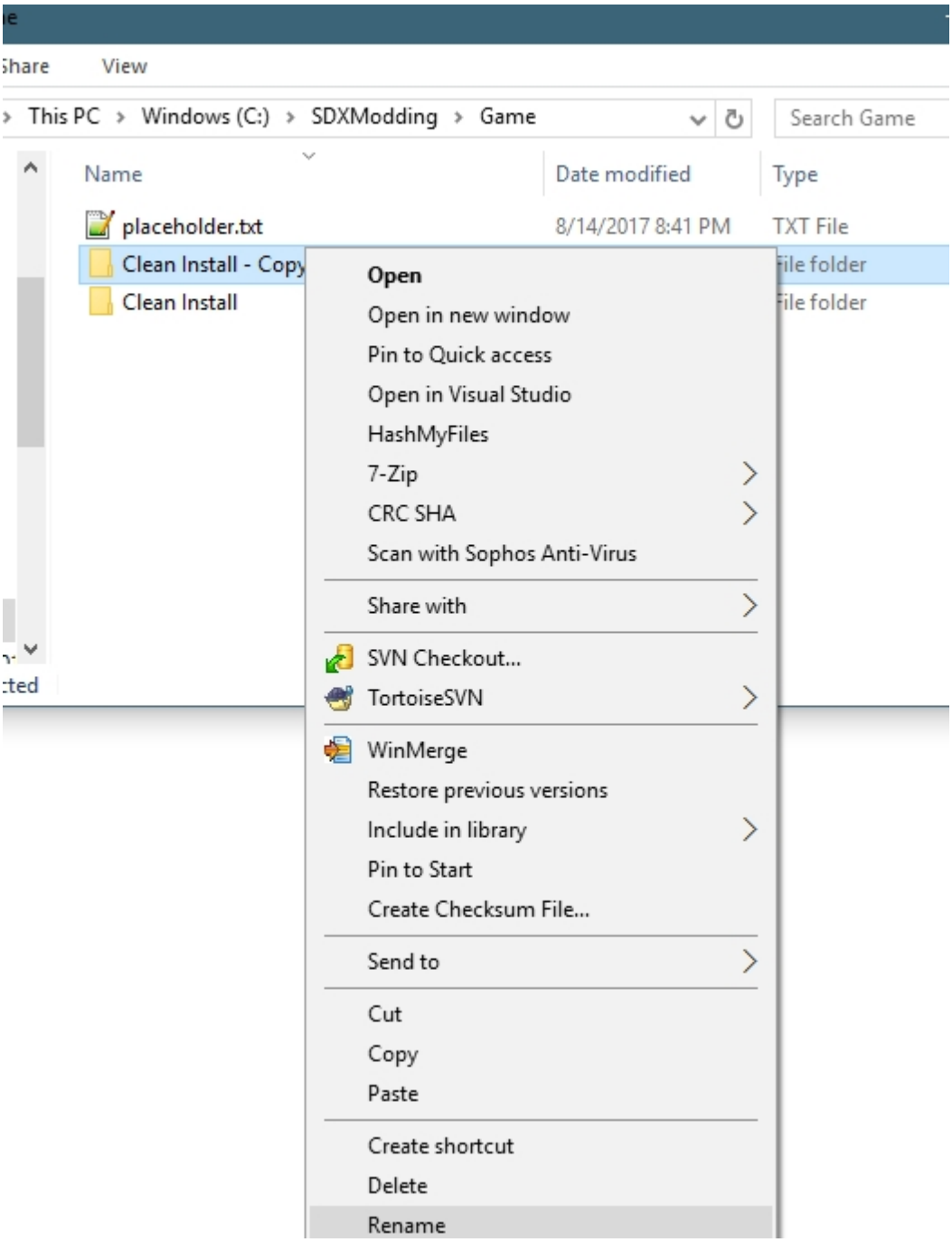


Right click on the Explorer Window again, and select "*Paste*"

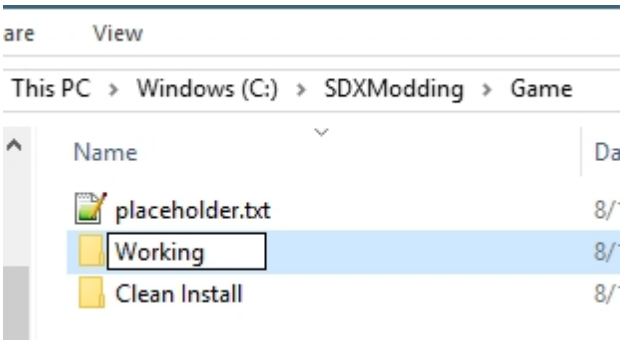


This will make a new copy called "*Clean Install - Copy*".

Right click on "*Clean Install - Copy*", and select "*Rename*"



Rename it to "*Working*"



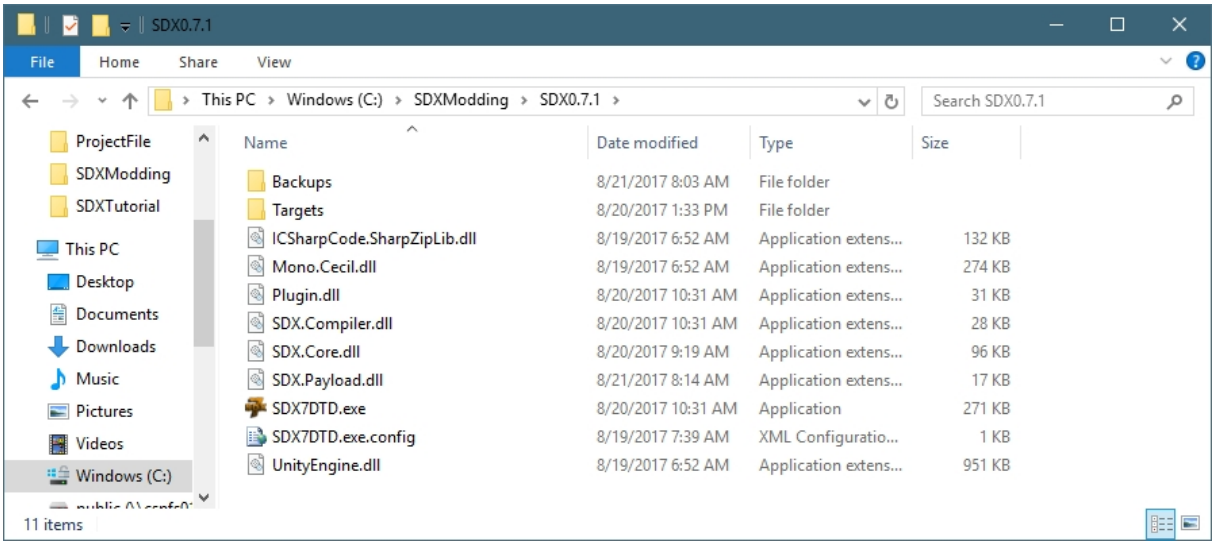
There we go! Now we have two copies of the game. One for a back up, and the other as our working folder.

SDX Launcher

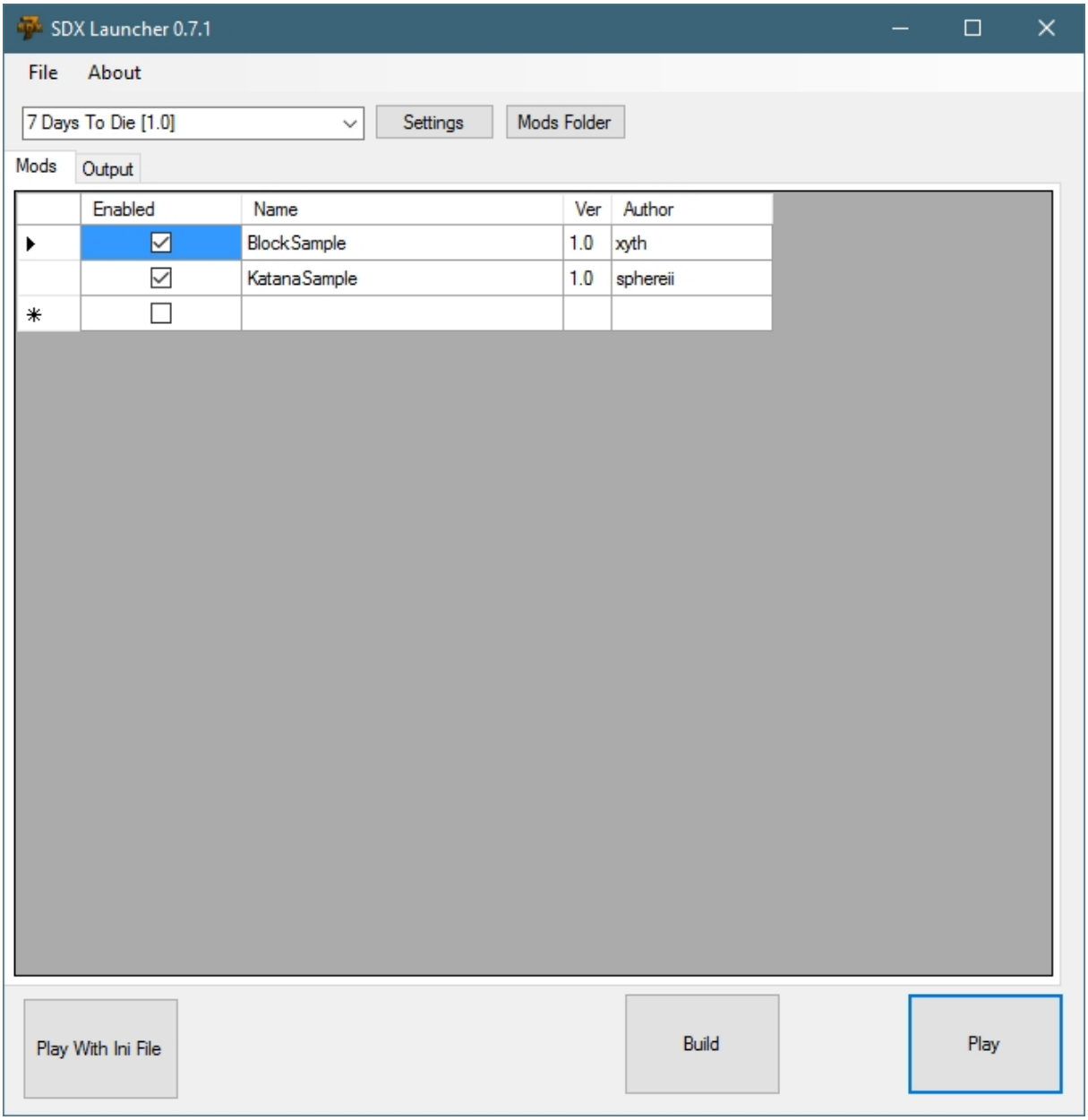
The SDX7DTD.exe is the GUI front end of the SDX Tool, and will be used to compile the mods.

A Video Tutorial for this section can be found here, by Xyth.

In the "C:\SDXModding\SDX0.7.1\" folder, double click on SDX7DTD.exe



Once loaded, you'll see this screen:

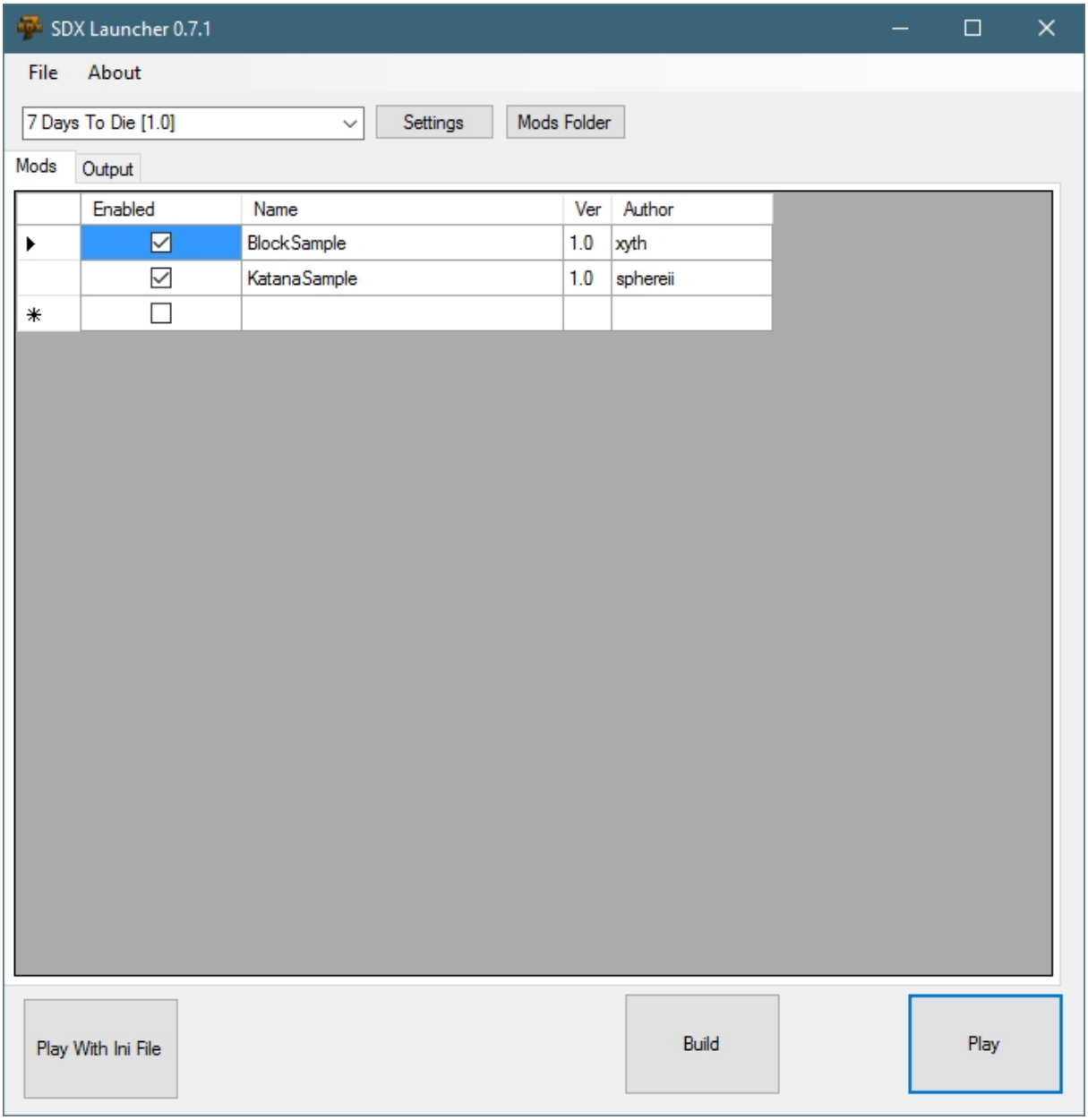


Settings Button

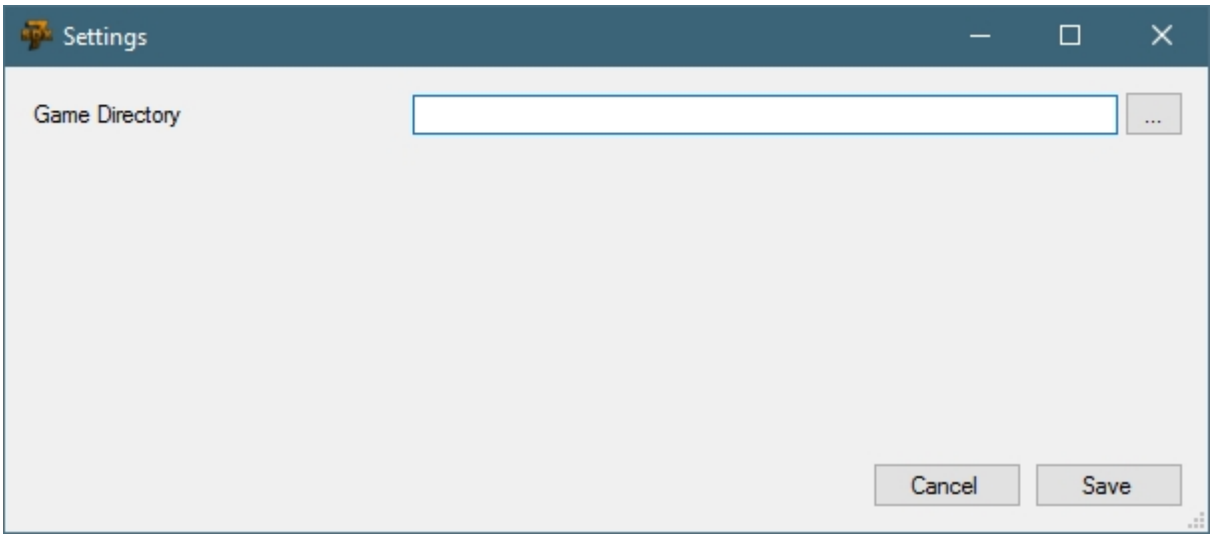
Once you have started the SDX Launcher 0.7.1, you will need to configure it for the first time.

In previous versions of the SDX Launcher, you had to select either 7 Days To Die, or 7 Days To Die Server. It is now auto-detected.

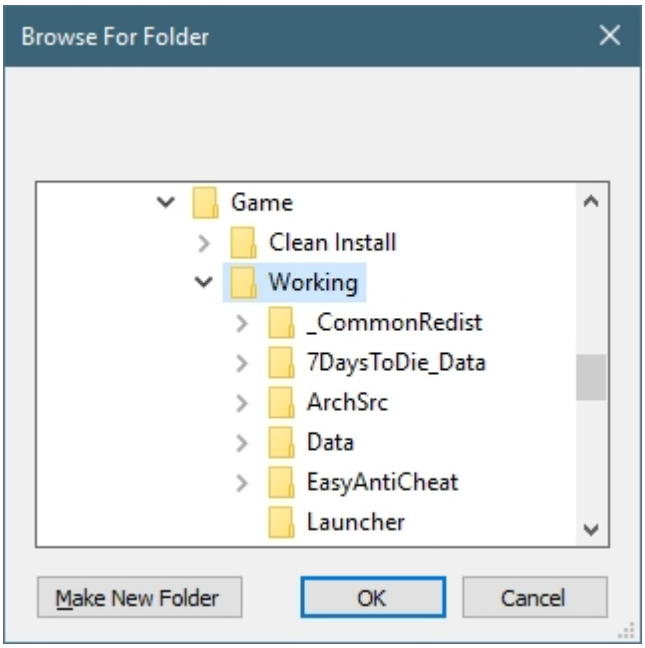
Click on the "Settings" button.



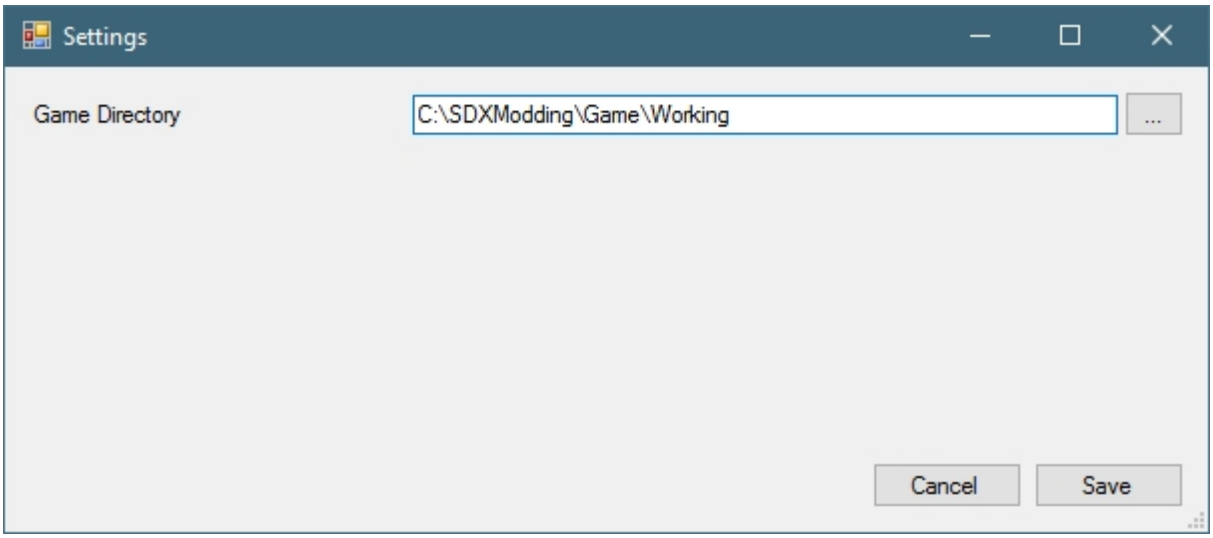
This is asking where your "Game Directory" exist. For us, it'll be that "Working" Folder we created here "C:\SDXModding\Game\Working"



Click on the "..." button and navigate to "C:\SDXModding\Game\Working", or Copy and Paste the URL.



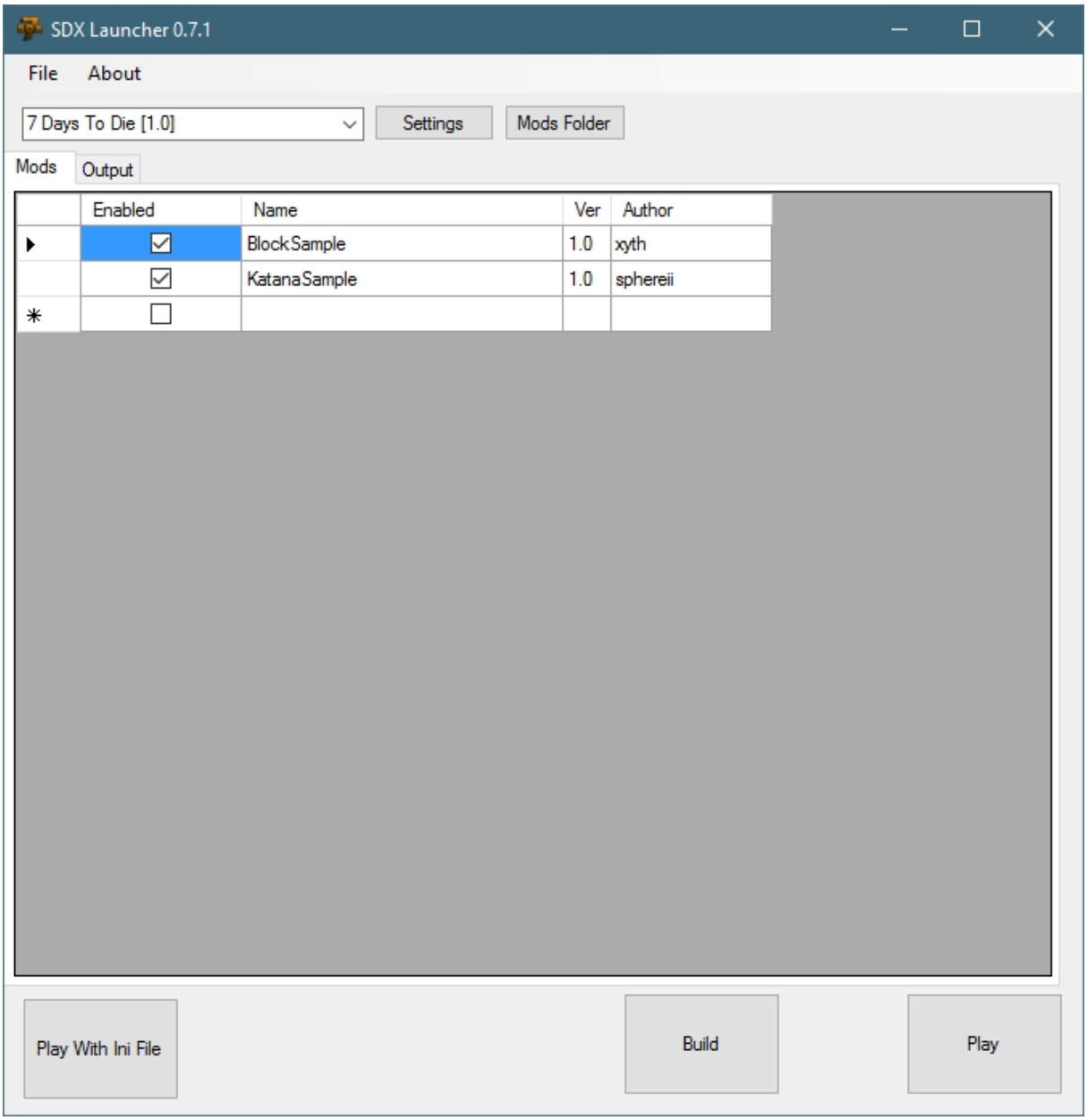
End result should look something like this:



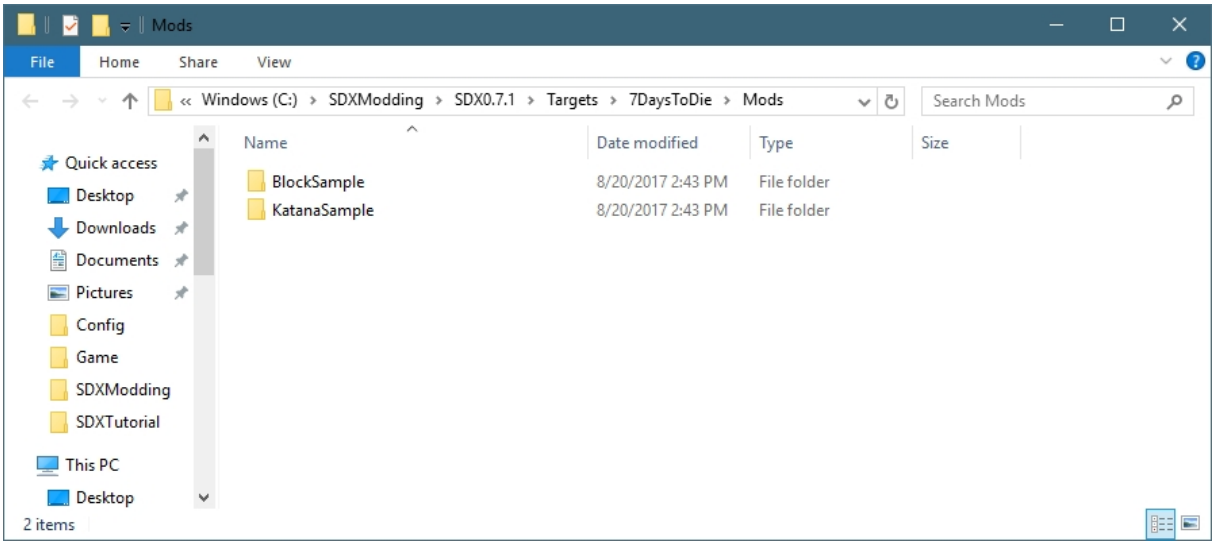
Mods Folder Button

The Mods Folder is where your SDX Mods will be located at. This is the Mod's code, resources, and other XML files that are needed.

Click on the "*Mods Folder*" button. This will open an Explorer window under "*C:\SDXModding\SDX0.7.1\Targets\7DaysToDie\Mods*"

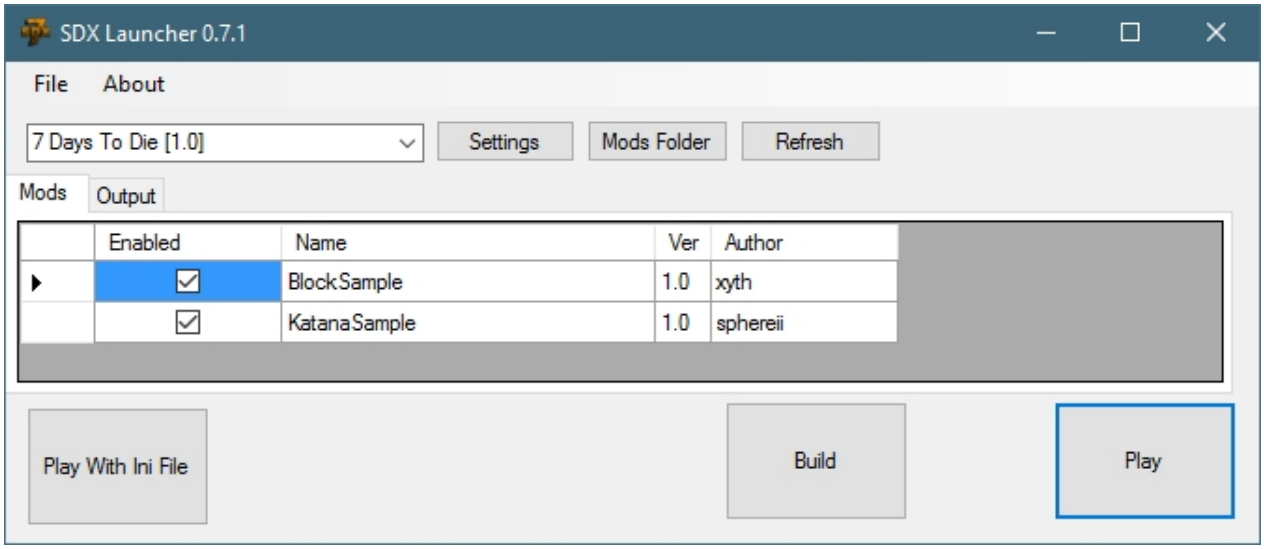


Mods Folder in Explorer



The Play and Build Buttons

There are three key buttons at the bottom of the SDX Launcher.



Play With Ini File

SDX 0.7.0 and previous versions came with an -sdxconfig= parameter, that pointed where the SDX mods were located. In SDX 0.7.1 and above, this -sdxconfig parameter has been made optional. However, in order to support existing SDX mods, it is left for compatibility.

When the Play With Ini File is pressed, the game will launch as you would expect. However, rather than referencing the Games' Mods/SDX/Resources folder, it will reference SDX 0.7.1's Target/7DaysDie/Mods folder.

Build

The Build button will do the following:

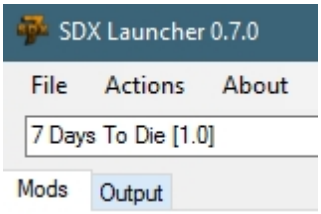
Restore the Assembly-CSharp.dll and all the XML files from its local back up, if it exists.
If no back up exists, it will make a backup of the Assembly-CSharp and the XML files
Builds and merges any SDX mods you have enabled
Copy the resulting build to your Working folder you've set in the Settings button.

Play

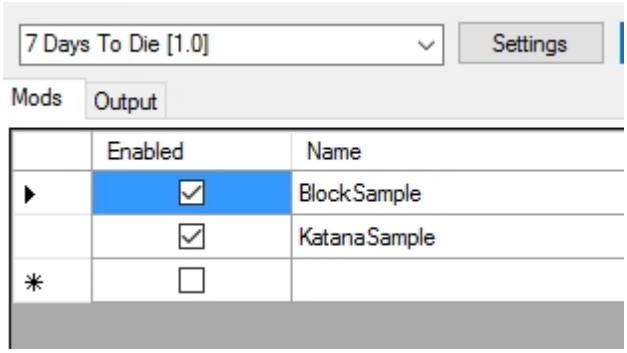
The Play button will run the game's 7daystodie.exe in your Working folder, using its copy of the Mods/SDX/Resources file.

The Mods / Output tab

The SDX Launcher contains two Tabs, called "Mods", and "Output".

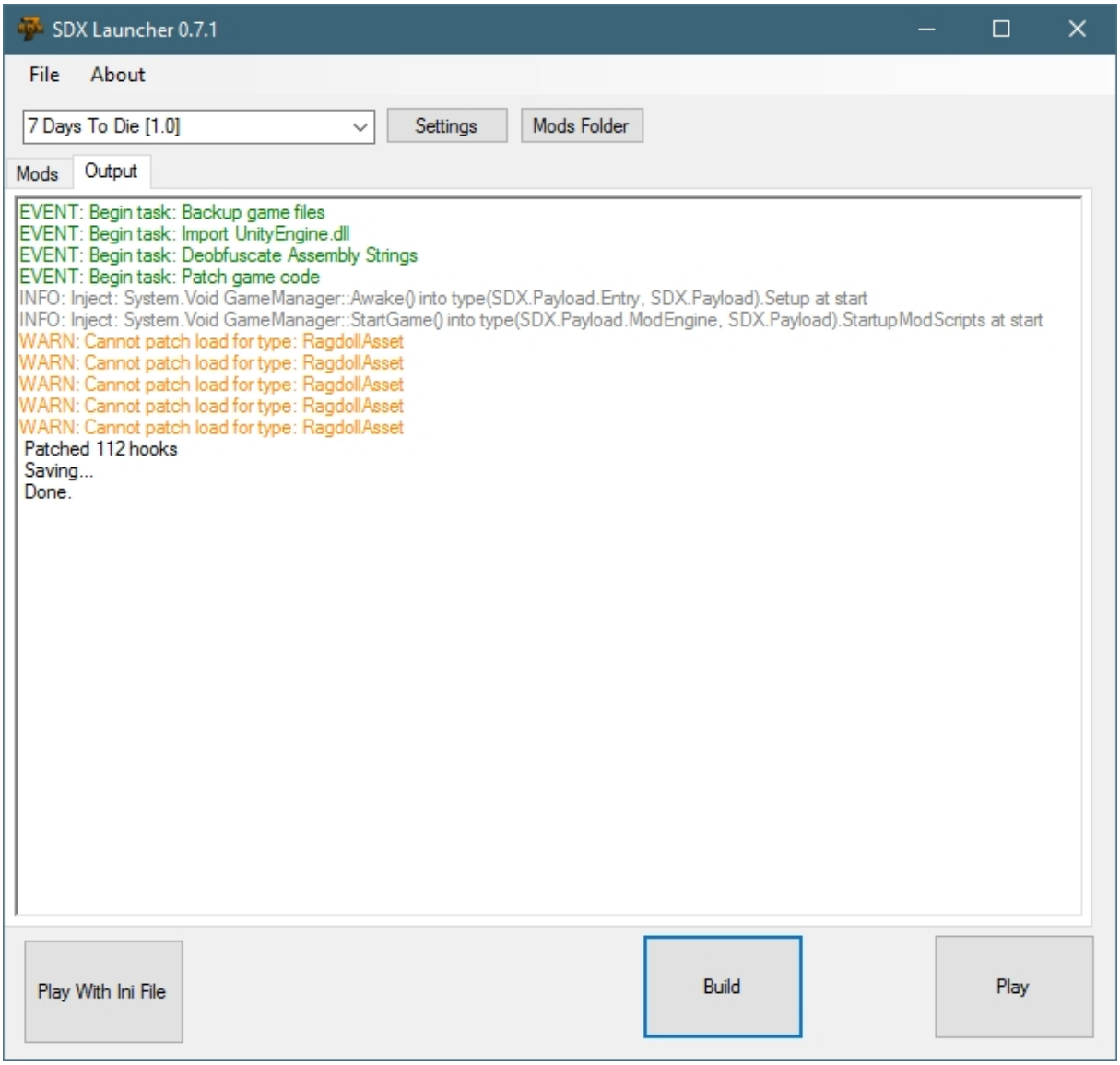


The "Mods" tab shows you the available Mods:



The Enabled check box indicates whether or not that SDX mod will be included in your compile or not.

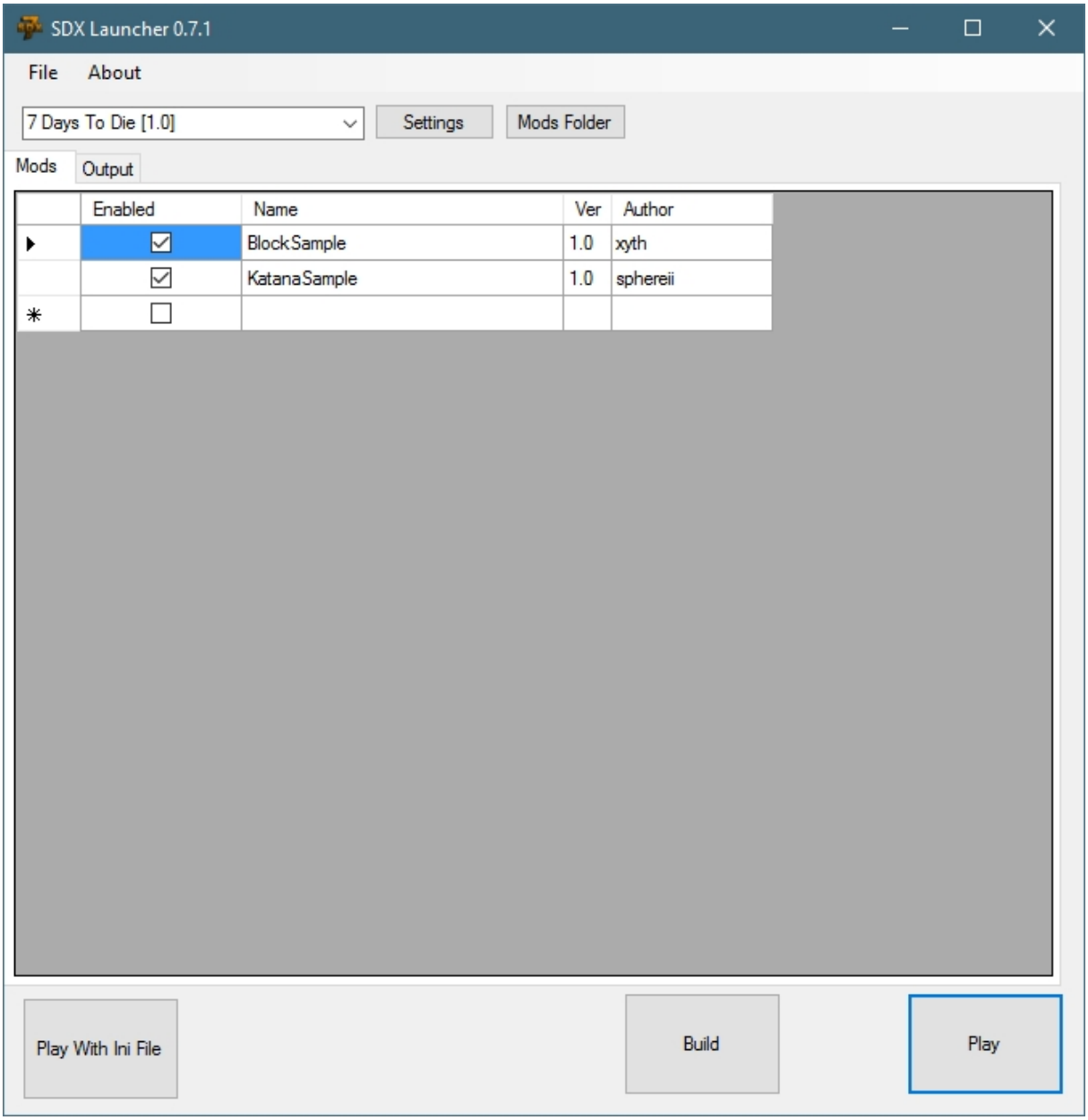
The "Output" tab shows you the SDX Launcher's actions.



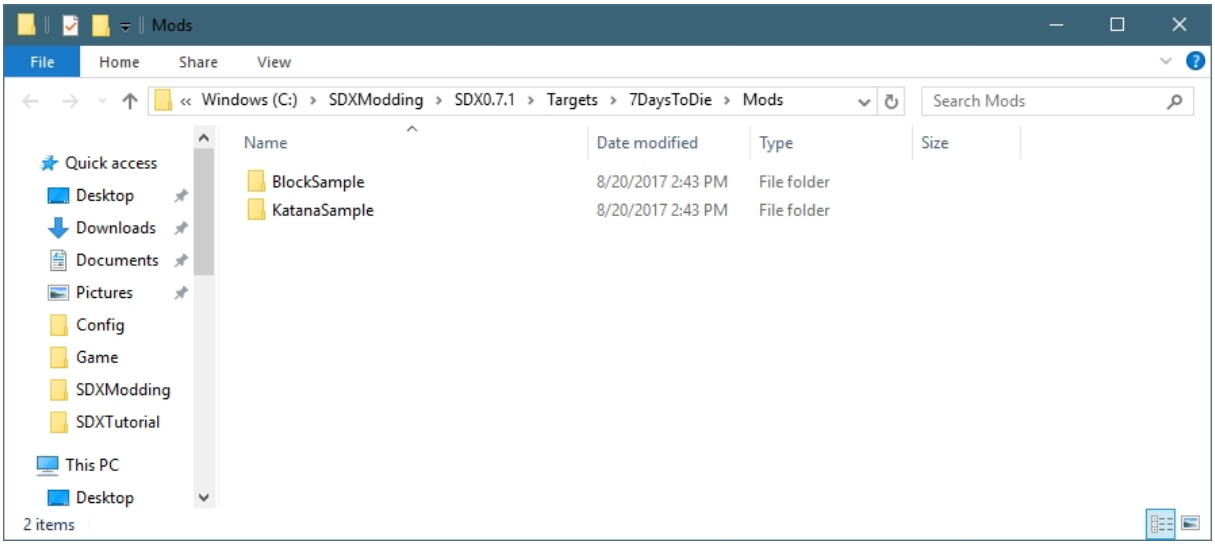
You can click back and forth between the Mods and Output folder.

Understanding an SDX mod

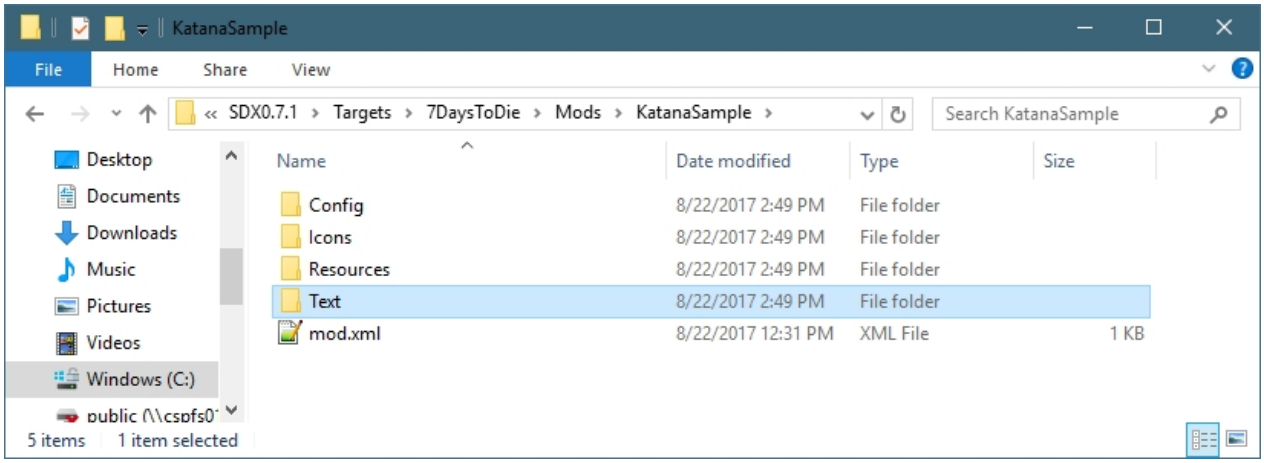
Start the SDX Launcher



Click on the "Mods Folder". This will open up the Explorer Window where the mods are installed.



Double click on the "KatanaSample" Folder, and look at the contents of the folder



The Config folder contains an XML file, that includes all the XML snippets and new items, blocks, and other items you want merged into SDX.

The Resources folder contains the Unity3D bundles, which are the new items and blocks.

mod.xml is a basic XML configuration file for SDX.

mod.xml:

The mod.xml gives information for the SDX Launcher, to show up in the tool.

```
<mod>
      <info>

            <author>sphereii</author>
            <name>Sample</name>
            <description>Sample Basic SDX Mod</description>
            <mod_version>1.0</mod_version>
            <game_version>16.2</game_version>
            <launcher_version>0.0.0</launcher_version>
      </info>

      <!-- This references any config files that SDX needs to merge into
your files -->
      <config_mods>
            <import file="Config\Sample.xml" />
    </config_mods>

</mod>
```

Author: The name of the author who created this mod, shows up in the SDX Launcher
Name: The name of the mod, shows up in the SDX Launcher
Description: Short description on what the mod does; Shows up in the launcher
Mod version: Which version of the mod it is, determined by the author
game_version: Which Game version the mod was designed for.

The <config_mods> points to where your mods' XML files are stored.

"Config" Folder:

Note: The use of the Config folder is optional. You can run your SDX Mod through the SDX7D2D without it merging any XML files. You can then edit your XML files manually as you normally would.

Double click on the "Config" folder, and open up the KatanaSample.xml in Notepad++. This XML adds a katana bundle using a unity3d bundle for its mesh.

```
<configs>
      <!--  This tells SDX to add to the Items.xml -->
      <config name="items">
            <!-- This tells SDX to add the following Items to the bottom of
the Items list -->
            <append xpath="/items">

                  <!-- New item will be Katana -->
                  <item id="" name="katanamichonne">
                        <!-- Extend it from the machete, but add the custom
mesh -->
                        <property name="Extends" value="machete"/>
                        <property name="Meshfile" value="#michonnekatana?
katana" />
                  </item>
            </append>
      </config>
</configs>
```

SDX will read this XML file, and add its contents to the right XML of the game. At the bottom of the file, it shows where it's adding the recipe on how to make a Katana using your new items.

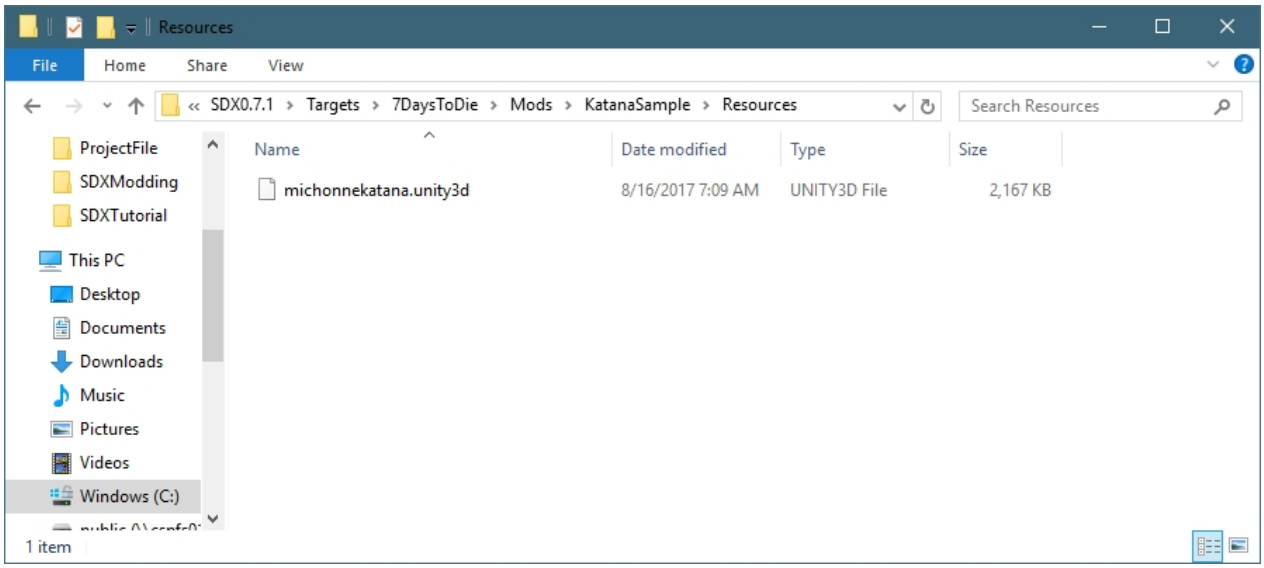
All of the <config> tags need to be in between the root node, which is the <configs> tags.

Sample Items Code	Description
<config name="items">	This tells SDX that everything in between the <config> tags will be included in the Items.xml file
<append xpath="/items">	This tells SDX that everything in between the <Append> path will be included inside of the <items> tag of items.xml
<item id="" name="katanamichonne">	This tells SDX the name of the new item to add. Notice there is no ID? SDX will auto-assign it an ID at build time.
<property name="Meshfile" value="#michonnekatana?katana" />	This will tell SDX, at run time, what the meshfile is called.
</append>	Closes the append tag
</config>	Closes the Items tag

Notice that there is no Item ID specified? When an Item ID, or block ID is not specified, SDX, at build time, will auto-assign an ID. Therefore, all your items and blocks will be auto-numbered.

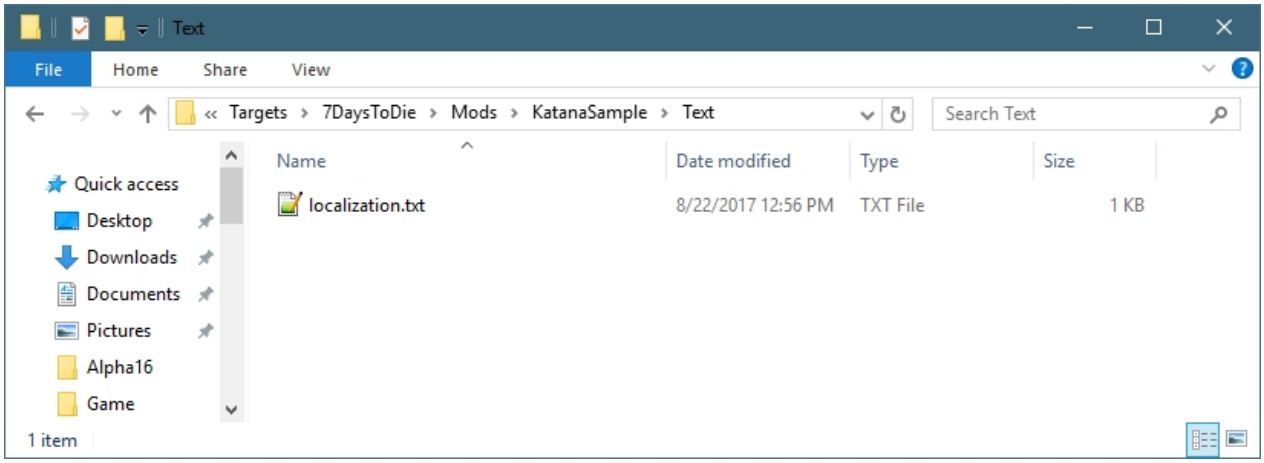
Resources:

The Resources folder contains all your Unity 3D models and textures, stored as Unity Bundle 3D files.



Text:

The Text folder is optional, but contains the localization files.

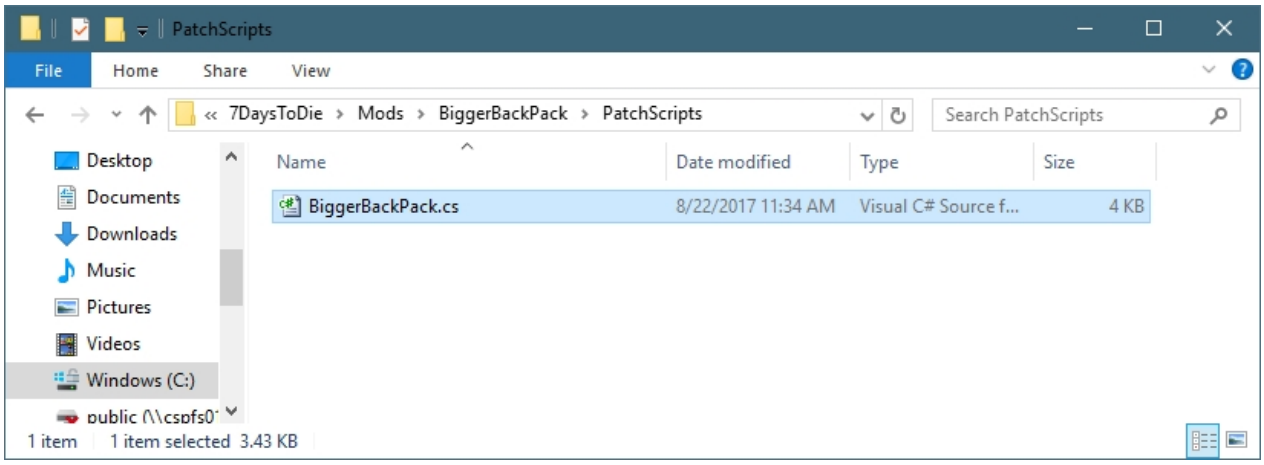


Example:

```
Key,Source,Context,Changes,English,French,German,Klingon,Spanish
katanamichonne,items,Melee,KgNone,Michionne's katana,,,,
katanamichonneDesc,items,Melee,New,Michionne's katana is ready to slice and dice up the zombies,,,,
```

PatchScript:

Some SDX Mods, like the Bigger Back Pack Mod, contains a PatchScript folder. A PatchScript is a C# script that is compiled at SDX Build time, and allows injection into the Assembly-CSharp. The SDX Launcher reads any files in this folder, and tries to compile it.



SDX Beginner Tutorial

If you have never used SDX, or just want a refresher, this beginner tutorial is what you'll want to start with.

> ***The Beginner Tutorial assumes you have already followed the "Getting Set up" guide.***

It's so easy to get confused and lost during your SDX journey, so we crafted this guide to take you step by step on how to get your SDX environment up and running.

We'll guide you through the steps of making a clean 7 Days to Die game folder, since we don't want any extra mods or tweaks to interfere with this tutorial.

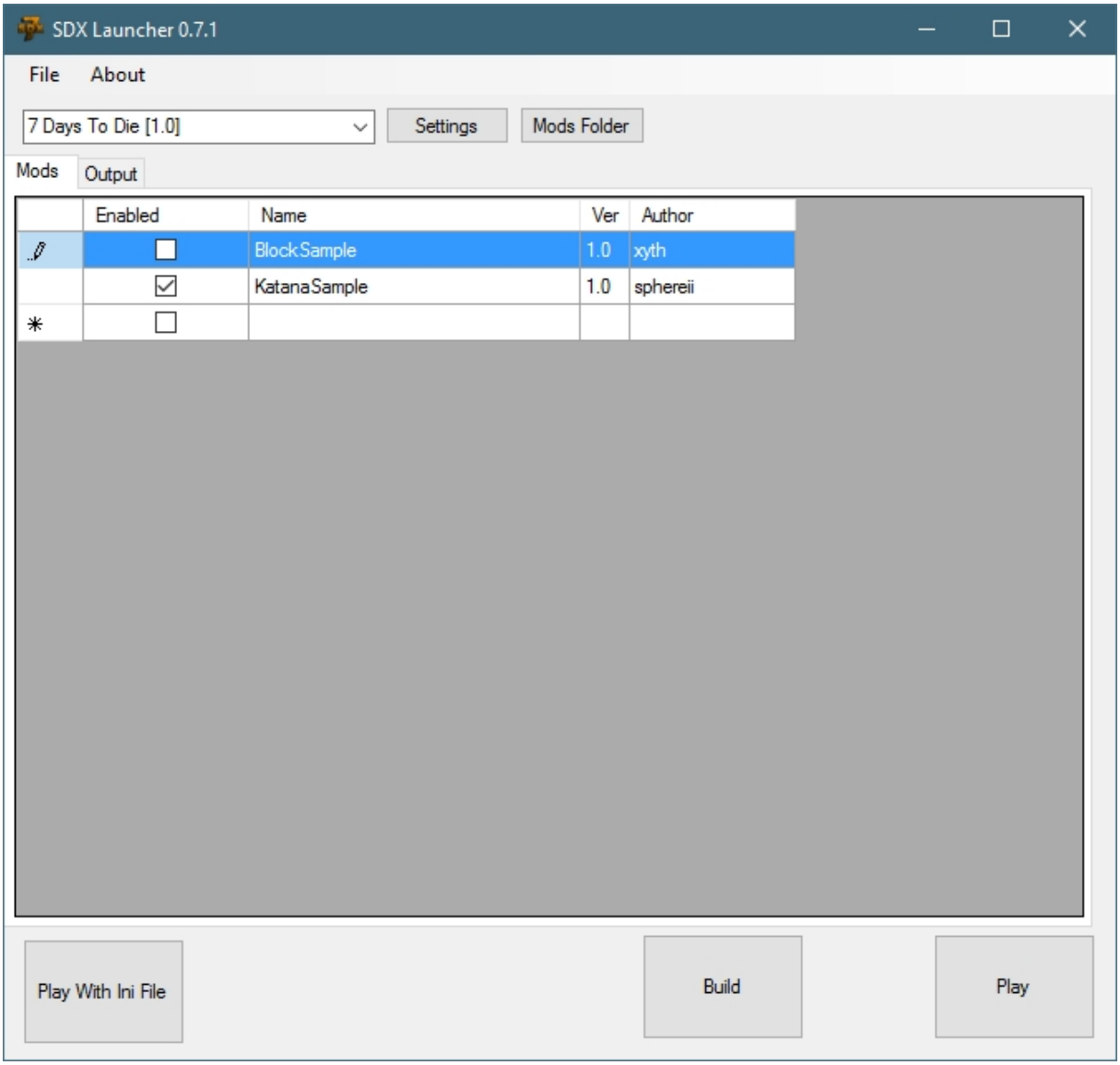
Once that's complete, we'll guide you through compiling an empty mod, and show you what each step does.

Finally, when you are ready, we'll add an actual item into the game, using the supplied Katana Mod.

Building for the first time

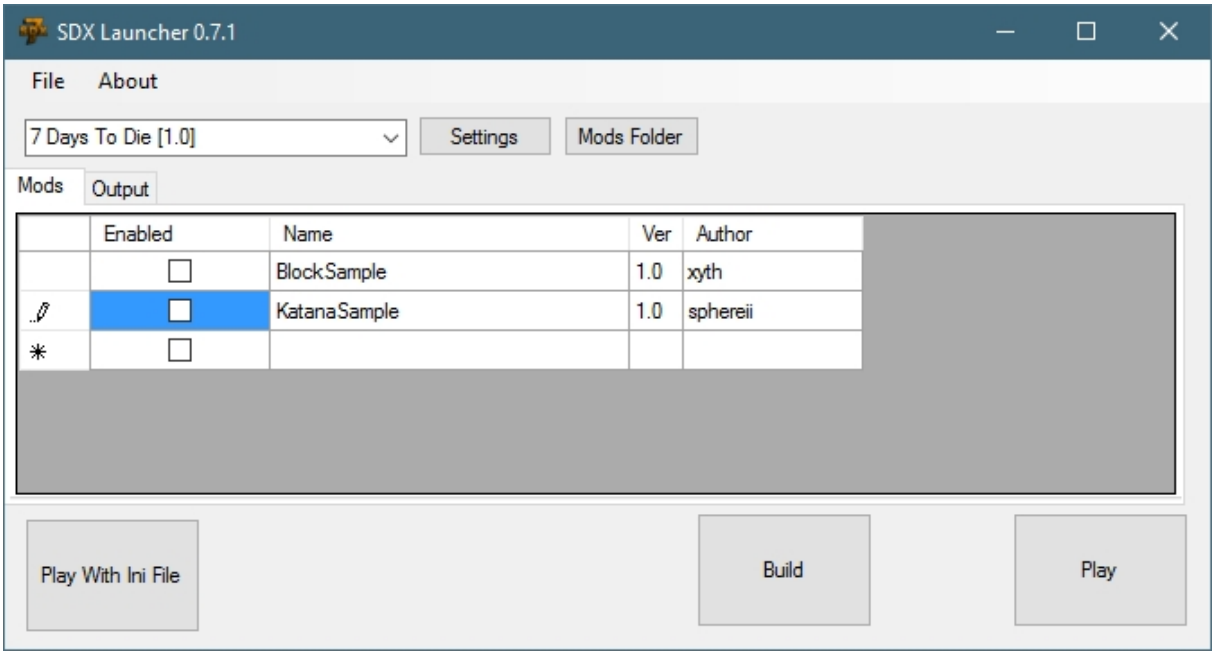
Once you have completed all the steps in the Getting Started section, it's time to trigger your first SDX compile!

Start the SDX Launcher



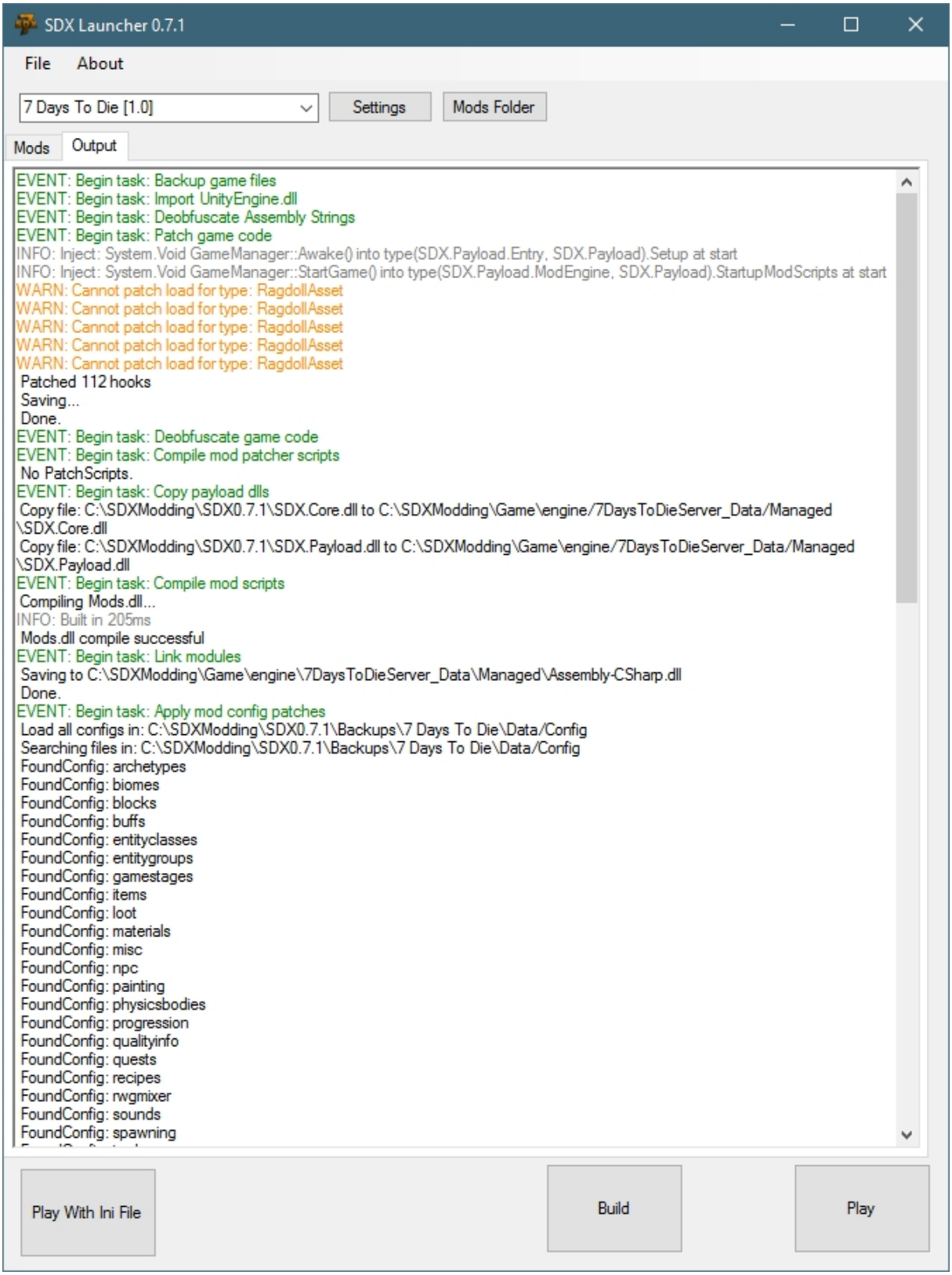
For the first compile, we want to disable all the available mods, to make sure everything is set up correctly.

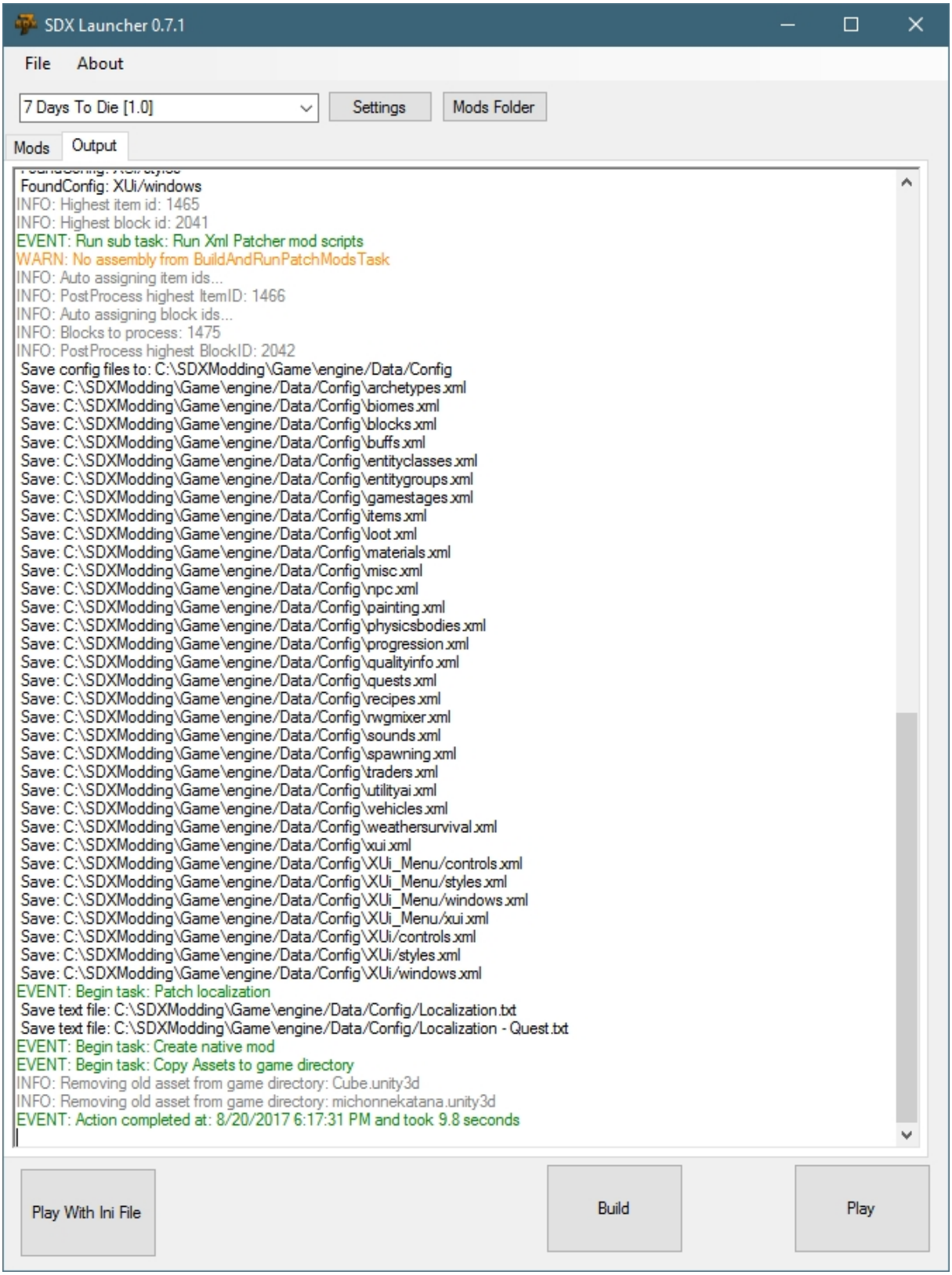
For each Mod listed in the "Mods" tab, click on the checkmark to disable it.



Click on the "*Build*" Button.

The SDX Launcher will print a lot of information in the "Output" window:


SDX Launcher 0.7.1
File
About
7 Days To Die [1.0]
Settings
Mods Folder
Mods
Output
EVENT: Begin task: Backup game files
EVENT: Begin task: Import UnityEngine.dll
EVENT: Begin task: Deobfuscate Assembly Strings
EVENT: Begin task: Patch game code
INFO: Inject: System.Void GameManager::Awake() into type(SDX.Payload.Entry, SDX.Payload).Setup at start
INFO: Inject: System.Void GameManager::StartGame() into type(SDX.Payload.ModEngine, SDX.Payload).StartupModScripts at start
WARN: Cannot patch load for type: RagdollAsset
WARN: Cannot patch load for type: RagdollAsset
WARN: Cannot patch load for type: RagdollAsset
WARN: Cannot patch load for type: RagdollAsset
WARN: Cannot patch load for type: RagdollAsset
Patched 112 hooks
Saving...
Done.
EVENT: Begin task: Deobfuscate game code
EVENT: Begin task: Compile mod patcher scripts
No PatchScripts.
EVENT: Begin task: Copy payload dlls
Copy file: C:\SDXModding\SDX0.7.1\SDX.Core.dll to C:\SDXModding\Game\engine/7DaysToDieServer_Data/Managed\SDX.Core.dll
Copy file: C:\SDXModding\SDX0.7.1\SDX.Payload.dll to C:\SDXModding\Game\engine/7DaysToDieServer_Data/Managed\SDX.Payload.dll
EVENT: Begin task: Compile mod scripts
Compiling Mods.dll...
INFO: Built in 205ms
Mods.dll compile successful
EVENT: Begin task: Link modules
Saving to C:\SDXModding\Game\engine\7DaysToDieServer_Data\Managed\Assembly-CSharp.dll
Done.
EVENT: Begin task: Apply mod config patches
Load all configs in: C:\SDXModding\SDX0.7.1\Backups\7 Days To Die\Data/Config
Searching files in: C:\SDXModding\SDX0.7.1\Backups\7 Days To Die\Data/Config
FoundConfig: archetypes
FoundConfig: biomes
FoundConfig: blocks
FoundConfig: buffs
FoundConfig: entityclasses
FoundConfig: entitygroups
FoundConfig: gamestages
FoundConfig: items
FoundConfig: loot
FoundConfig: materials
FoundConfig: misc
FoundConfig: npc
FoundConfig: painting
FoundConfig: physicsbodies
FoundConfig: progression
FoundConfig: qualityinfo
FoundConfig: quests
FoundConfig: recipes
FoundConfig: rwgmixer
FoundConfig: sounds
FoundConfig: spawning
Play With Ini File
Build
Play



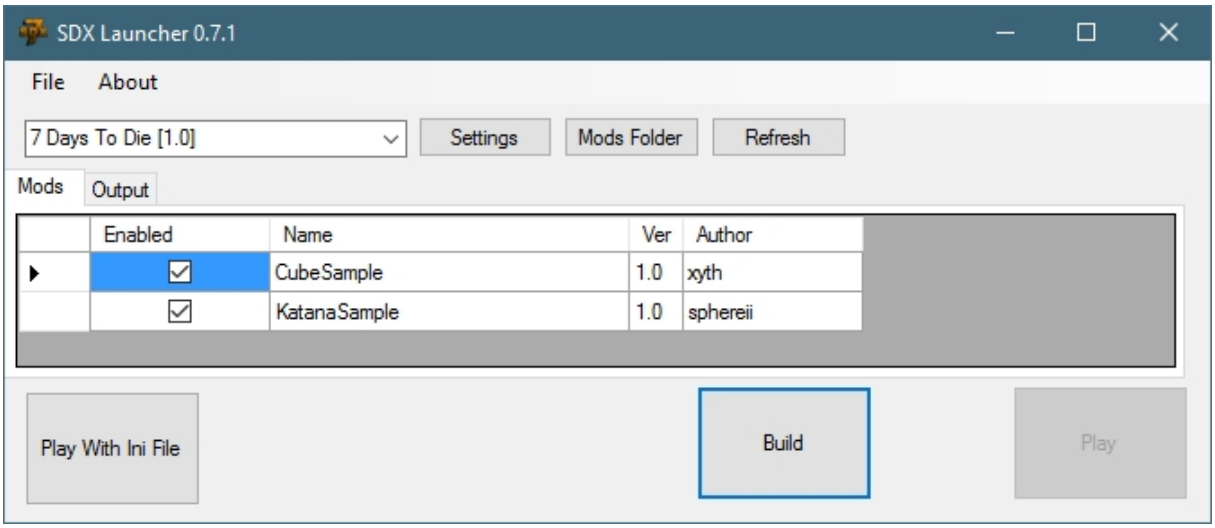
The Cube Mod

The first SDX Mod we are going to add is the Cube Mod. This will add a new cube to the game, letting you place it in game.

Building the Cube Mod

Once you have completed all the steps in the "Building for the first time" option, it's time to trigger your first SDX compile!

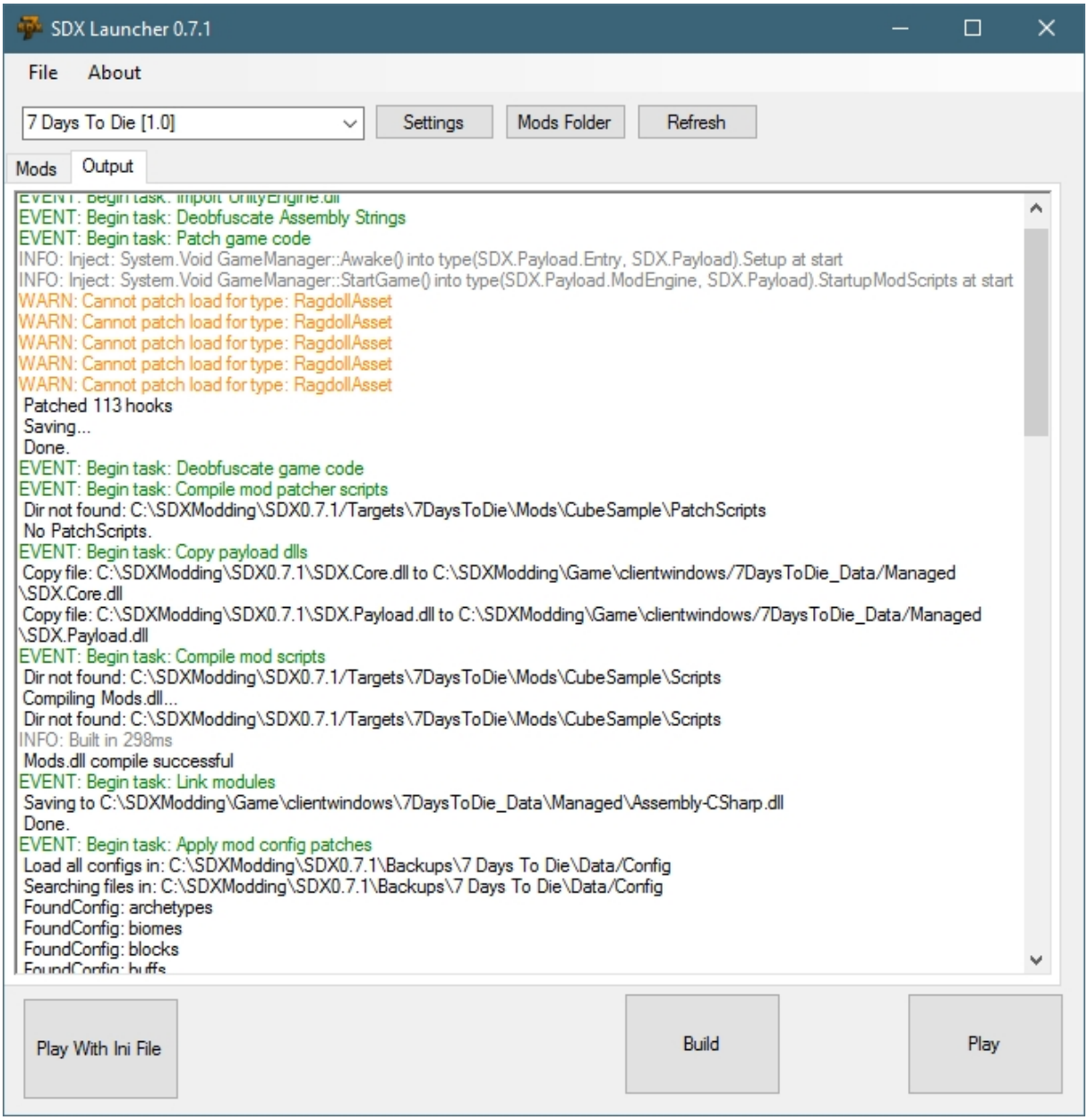
Click on the white square to enable the Cube Sample to enable it.



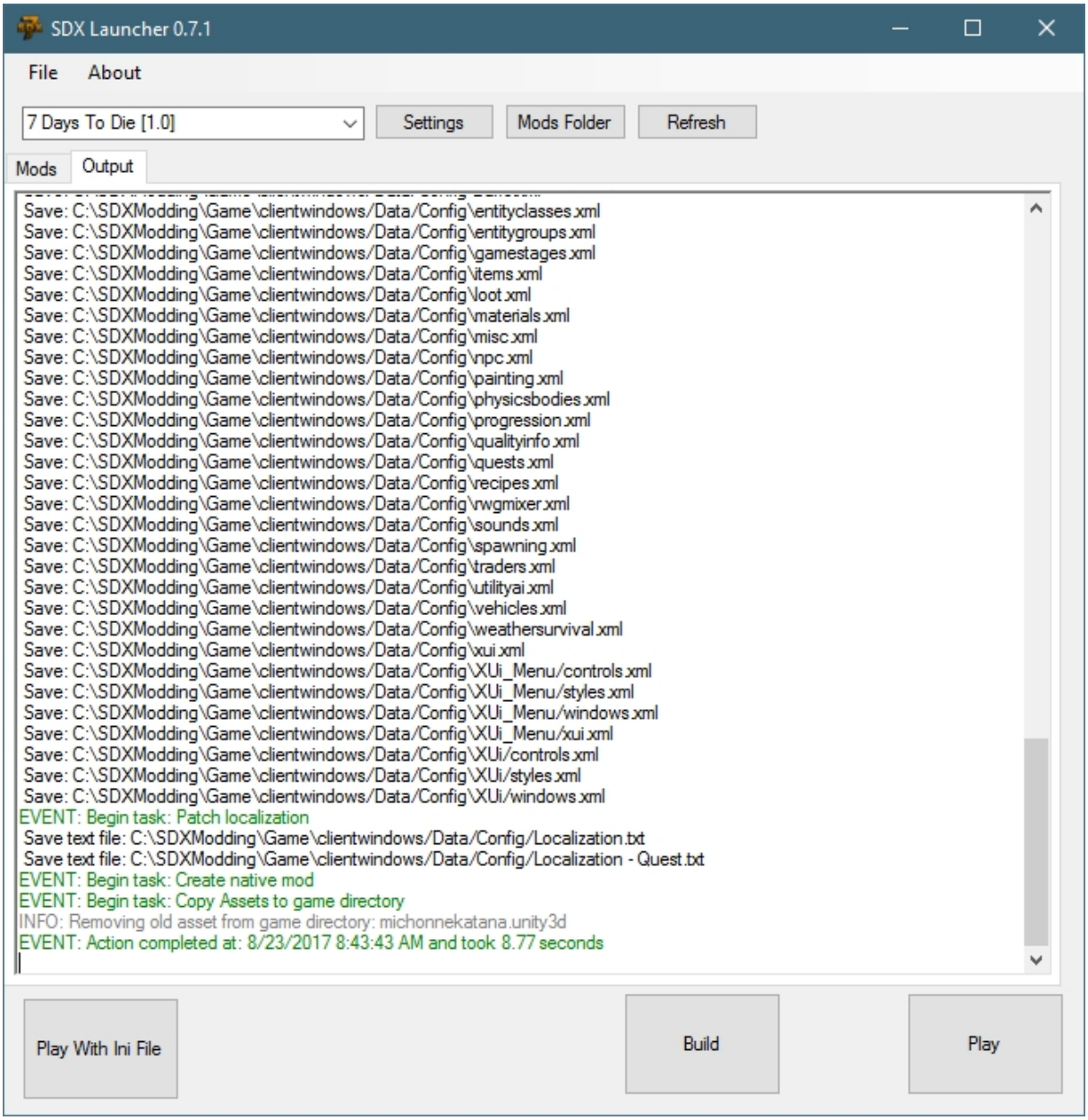
And click on Build.

The SDX Build starts by switching from the "Mods" tab to the "Output" tab.

You'll see the SDX compile start, and display the log file as it does its actions.



When SDX is finished, you'll see this at the end of the Output window



SDX is now compiled into your Working copy of the game.

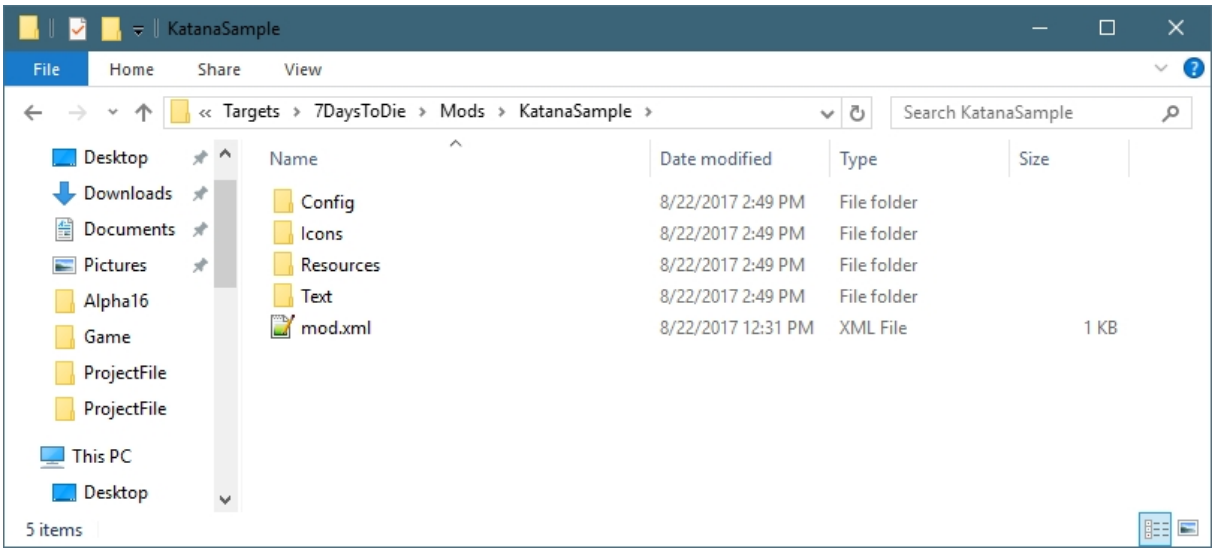
You can now hit the Play button to launcher the game. Spawn a new world, or log into an existing one, and go into the Creative Menu. You'll see that the cube is now available.

And showing the cube in hand:

The Katana Mod

The Katana Mod

The Katana Mod is a simple Mod to install and add to the game. It follows the same principles to the Cube Mod, but adds more details and features.



Config Folder

The Config Folder of the Katana Mod contains the XML snippet that will be included into the game's DLL files. For the Katana Mod, this includes an Item that simply extends the existing machete item, but uses a custom mesh file.

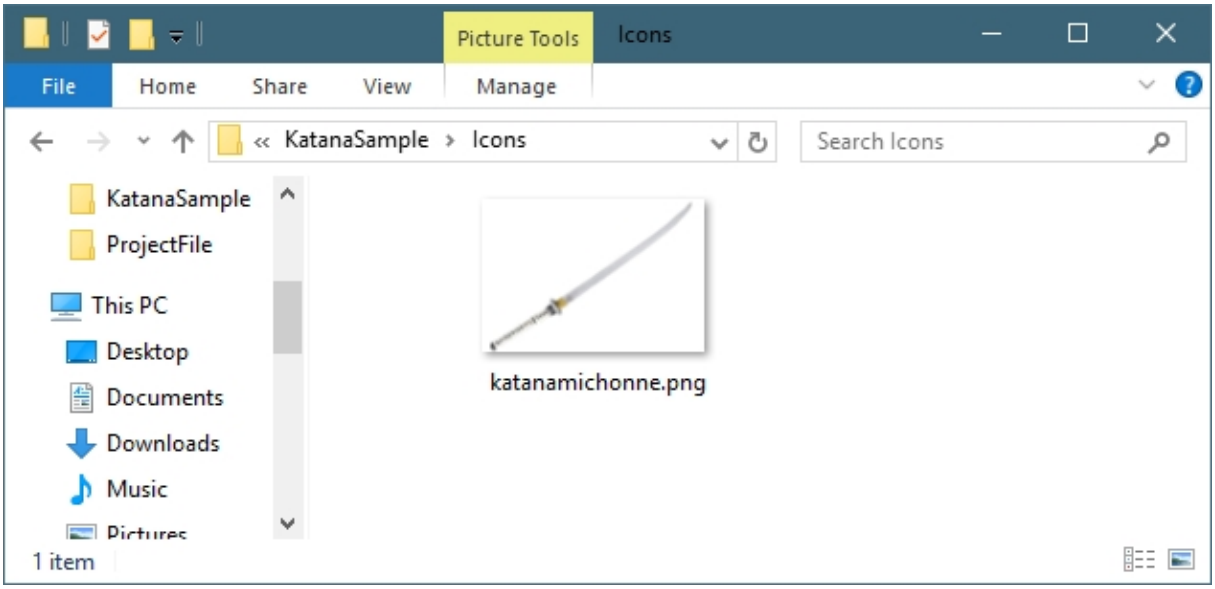
File: KatanaSample\Config\Katana.xml

```
<configs>
      <!--  This tells SDX to add to the Items.xml -->
      <config name="items">
            <!-- This tells SDX to add the following Items to the bottom
of the Items list -->
            <append xpath="/items">
```

```
                <!-- New item will be Katana -->
                <item id="" name="katanamichonne">
                      <!-- Extend it from the machete, but add the
custom mesh -->
                      <property name="Extends" value="machete"/>
                      <property name="Meshfile" value="#michonnekatana?
katana" />
                </item>
            </append>
      </config>
</configs>
```

Icons Folder

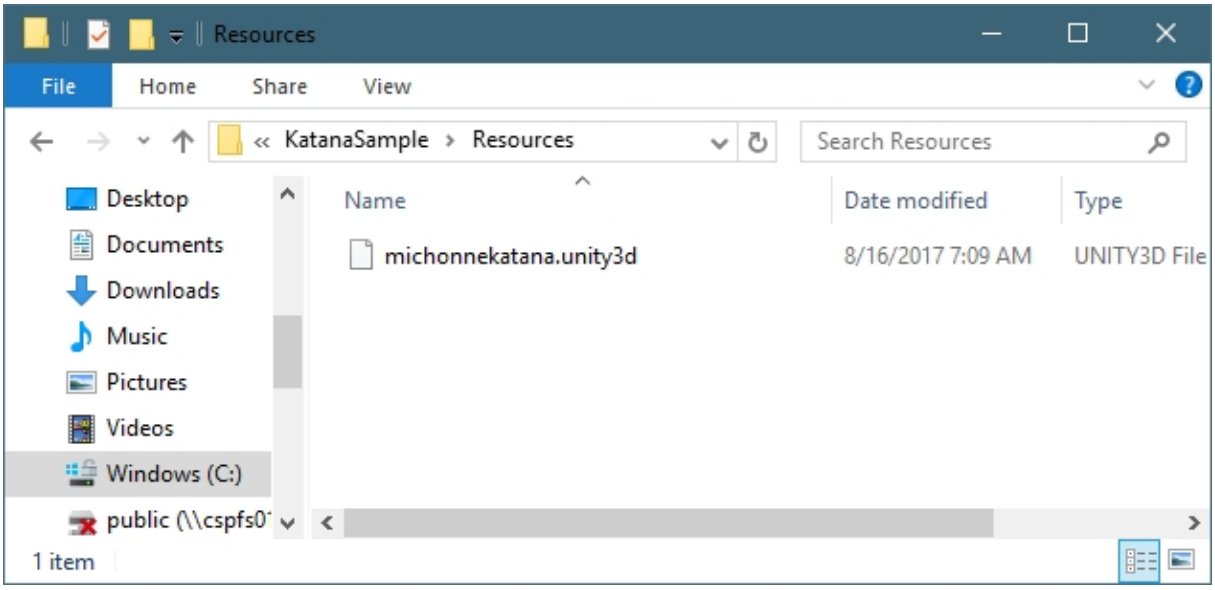
The Icons folder contains the item's inventory icons for the mod.



When the mod gets compiled, all files will be copied to your "Working\Mods\SDX\ItemIcons\" folder.

Resource Folder

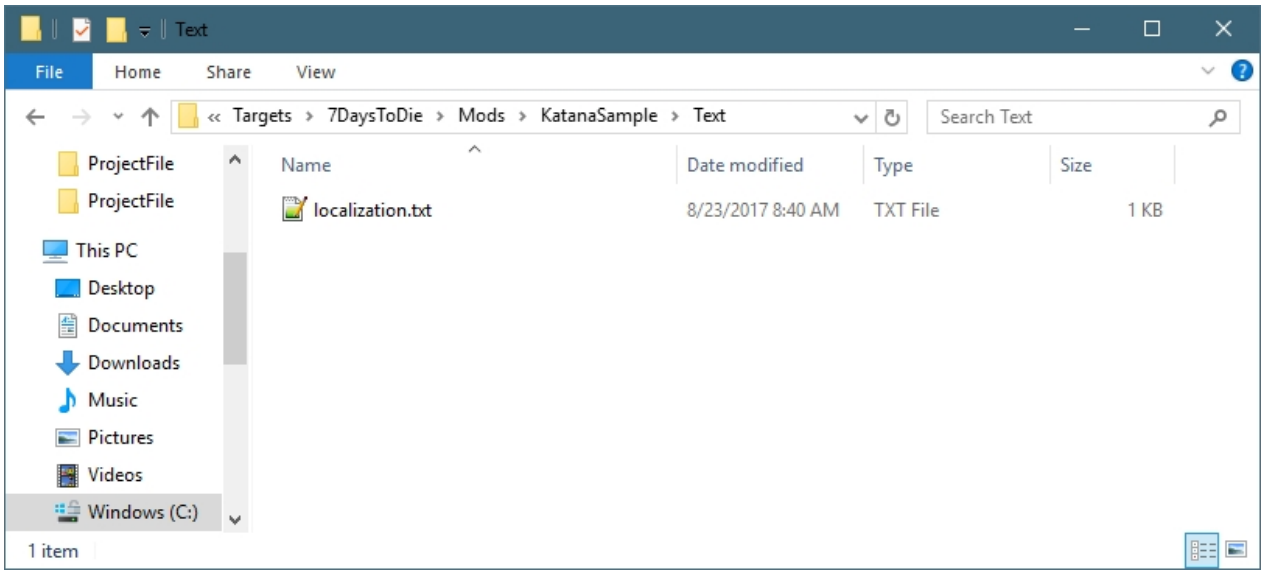
The Resource folder contains the unity3D bundle, which contains the 3D model of the Katana sword.



When the mod gets compiled, all files will be copied to your "Working\Mods\SDX\Resources\" folder.

Text Folder

The Text folder contains the localization files, either localization.txt or Localization - Quest.txt




```
Key,Source,Context,Changes,English,French,German,Klingon,Spanish
katanamichonne,items,Melee,KgNone,Michionne's katana,,,,
katanamichonneDesc,items,Melee,New,Michionne's katana is ready to slice and dice up the zombies,,,,
```

mod.xml

The mod.xml file is used by the SDX Launcher to compile and find its files. This information shows up in the SDX Launcher.

File: KatanaSample\mod.xml

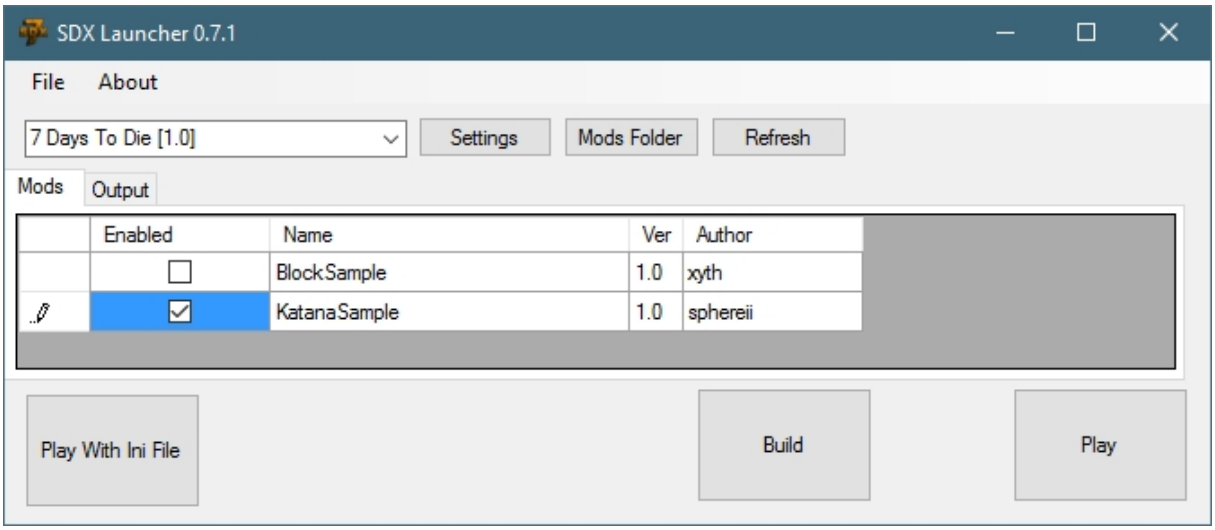
```
<mod>
```

```
      <info>
            <!-- Information about the Mod, the author and version
information -->
            <!-- These are displayed in the SDX Launcher -->
            <author>sphereii</author>
            <name>KatanaSample</name>
            <description>Sample Katana SDX Mod</description>
            <mod_version>1.0</mod_version>
            <game_version>16.2</game_version>
            <launcher_version>0.0.0</launcher_version>
      </info>

      <!-- This references any config files that SDX needs to merge into
your files -->
      <config_mods>
            <import file="Config\Katana.xml" />
    </config_mods>

</mod>
```


Building the Katana Mod

Once you have completed all the steps in the "Building for the first time" option, it's time to trigger your first SDX compile!

Click on the white square to enable the KatanaSample to enable it.

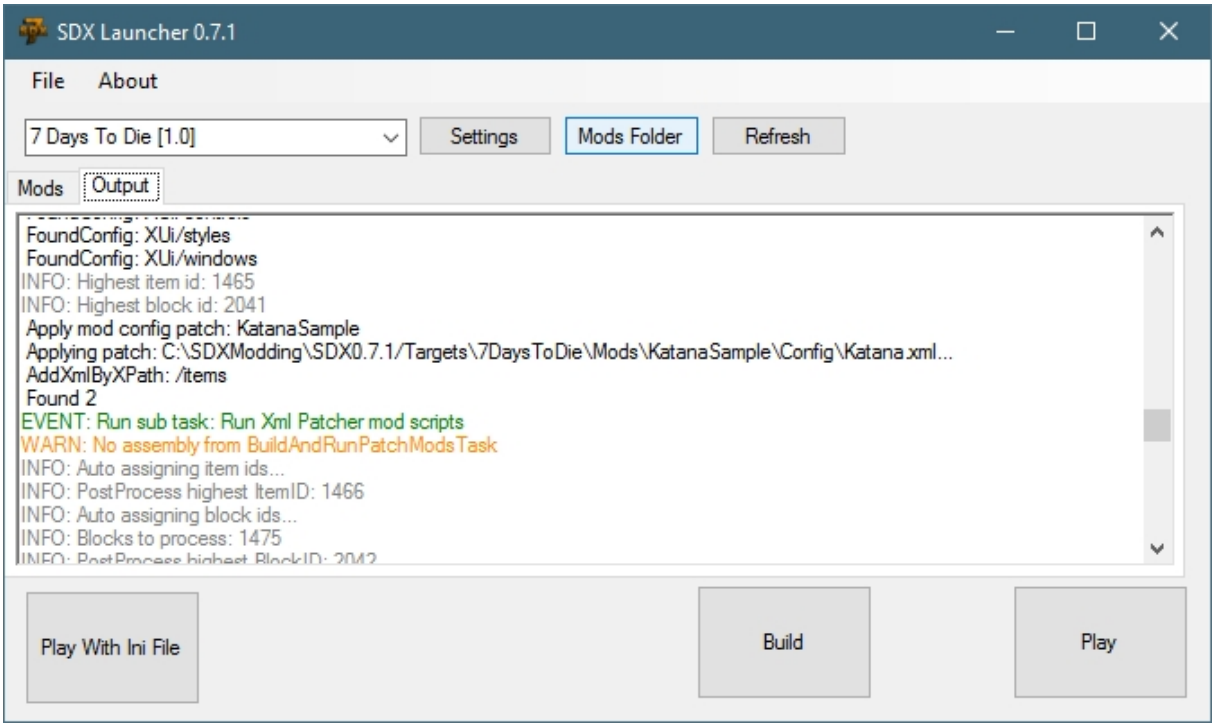


And click on Build.

The SDX Build starts by switching from the "Mods" tab to the "Output" tab.

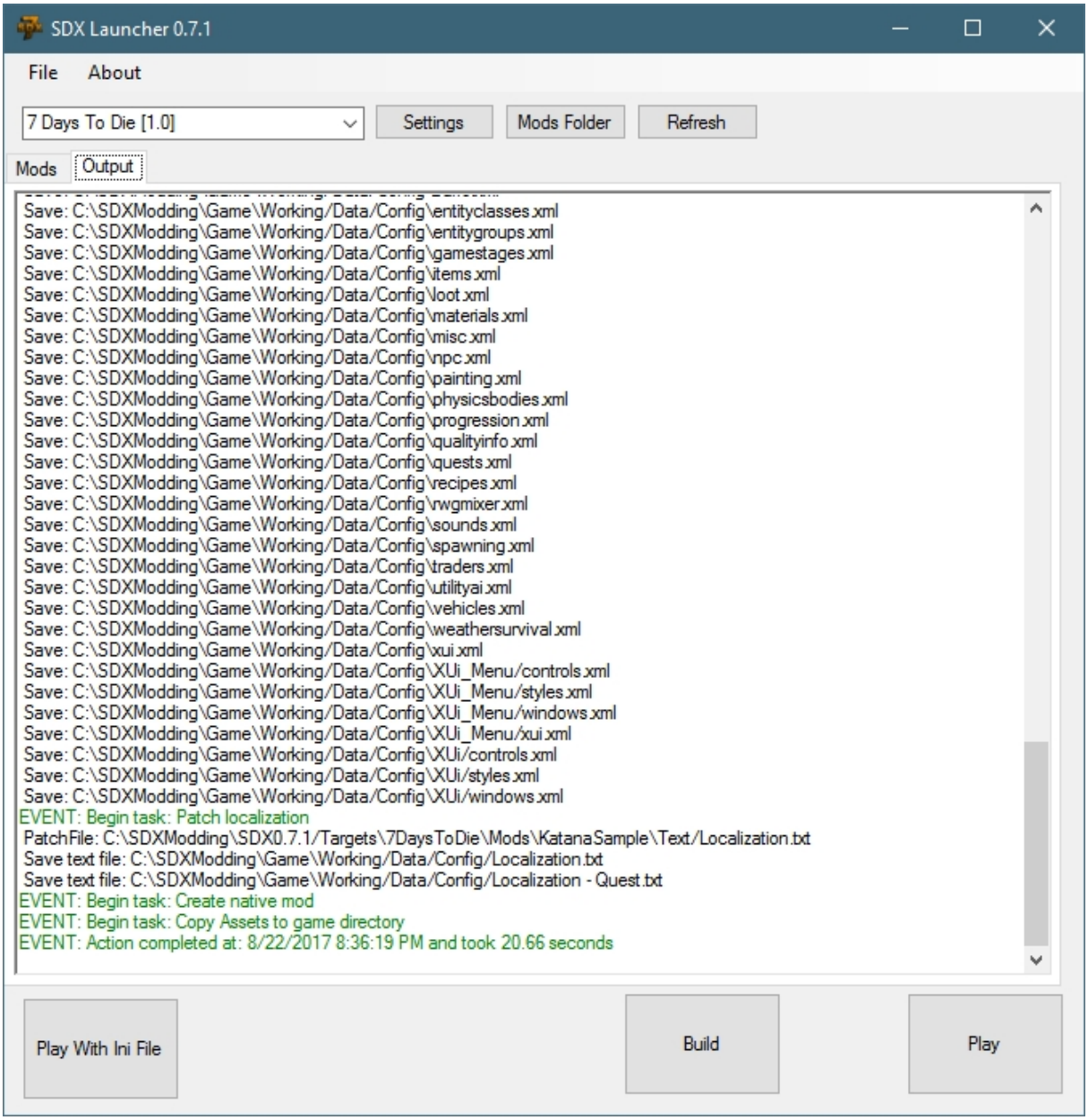
You'll see the SDX compile start, and display the log file as it does its actions.

As it builds, you may see the following error:



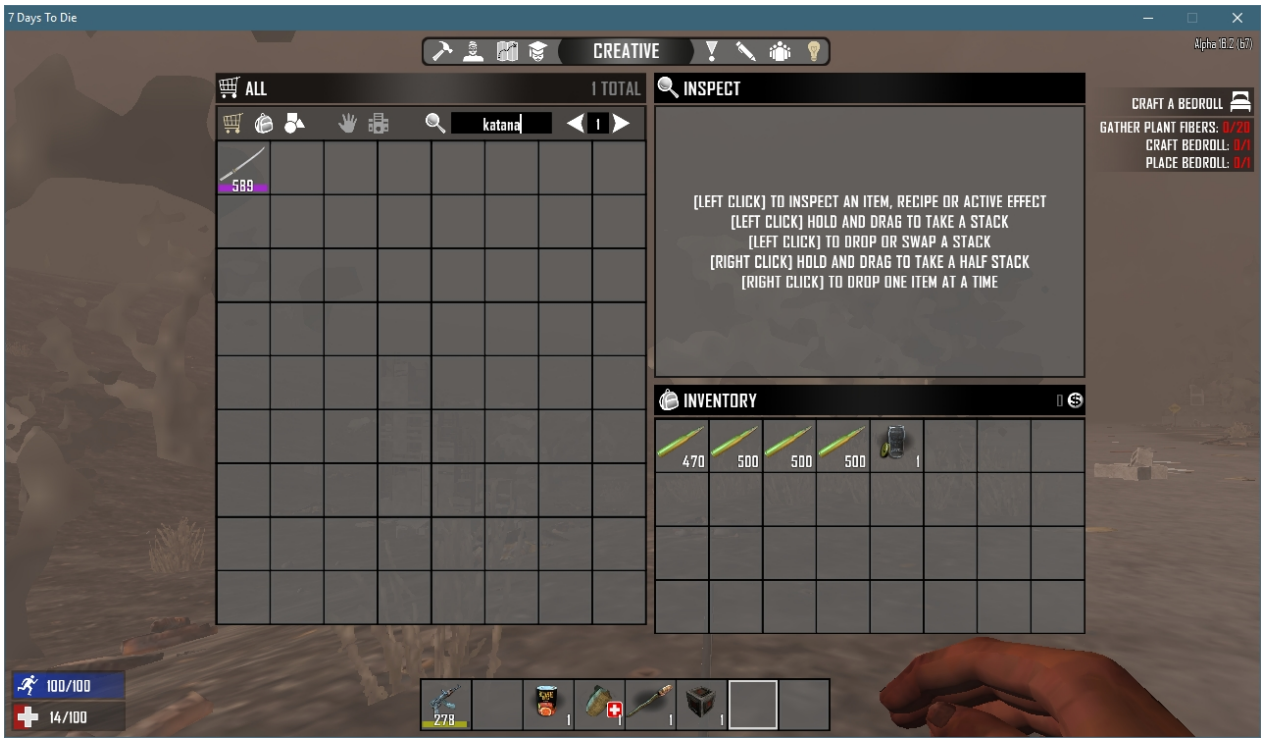
In this exercise, the "WARN: No assembly from BuildAndRunPatchModsTask" is a not a concern, since there is no Mods scripts yet enabled.

When SDX is finished, you'll see this at the end of the Output window



SDX is now compiled into your Working copy of the game.

You can now hit the Play button to launcher the game. Spawn a new world, or log into an existing one, and go into the Creative Menu. You'll see that the Katana sword is now available.

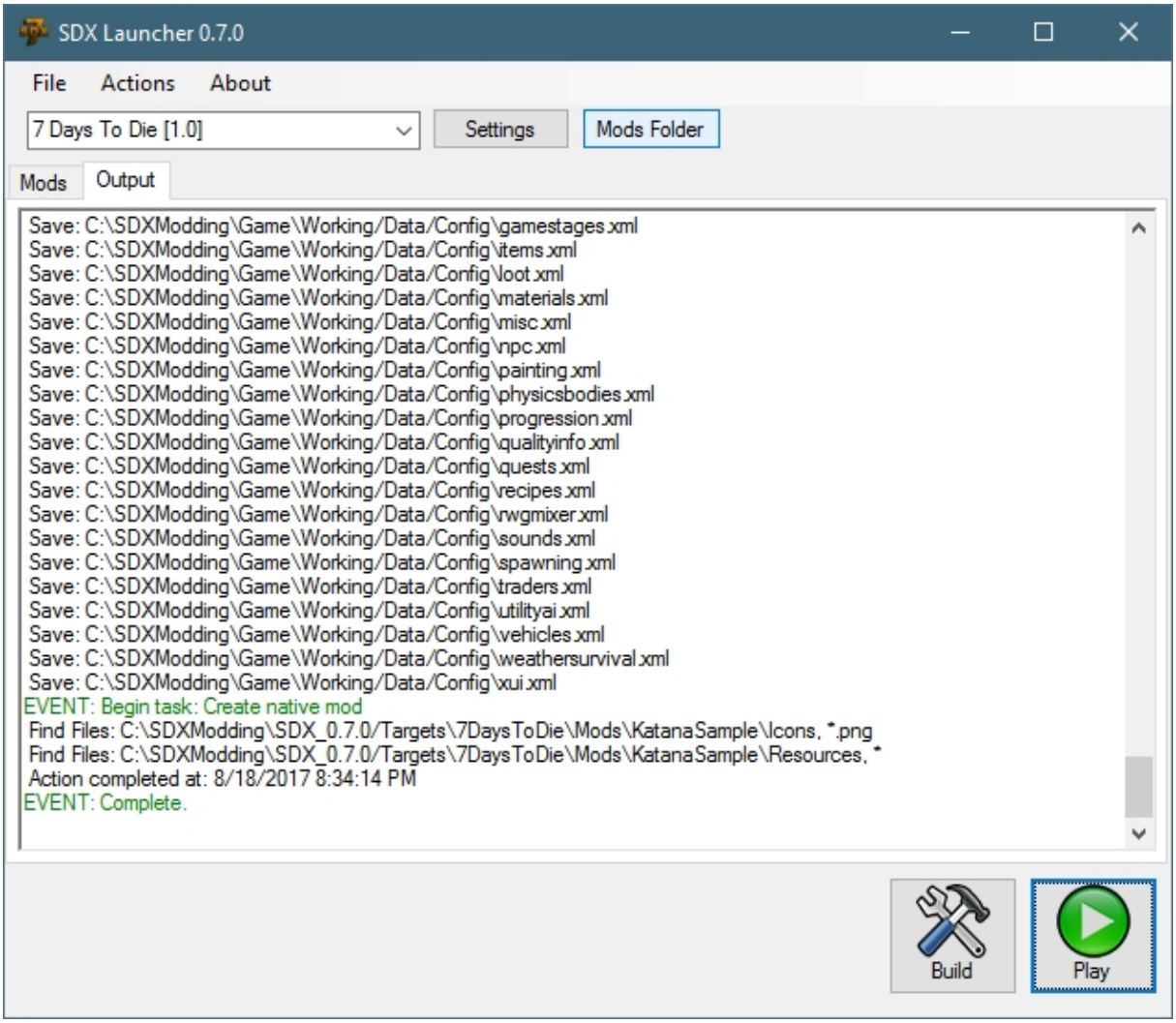


And showing the Katana in hand:

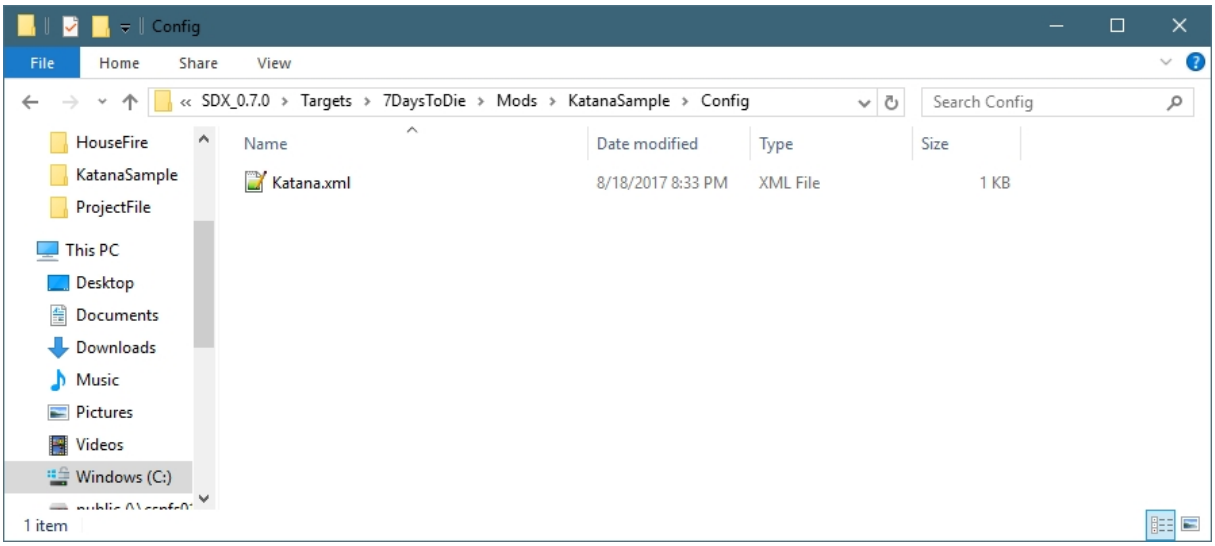
Adding a Recipe for the Katana Mod

What fun is the Katana if you can't make it in game? Let's go through the steps on adding a recipe list for it.

In the SDX Launcher, click on the Mods Folder.



Then navigate into the double click on the Katana Sample folder, then the Config folder:



Using Notepad++, edit the file

Add a new "<config> node under the katanamichonne item by copy and pasting the Recipes Snippet, highlighted in Blue for clarity

Recipes Snippet

```
      <!-- Adding a new recipe for the mod -->
      <config name="recipes">
            <append xpath="/recipes" >
                  <recipe name="katanamichonne" count="1"
craft_area="workbench">
                        <ingredient name="forgedSteel" count="20"/>
                        <ingredient name="wood" count="4"/>
                        <ingredient name="leather" count="4"/>
                  </recipe>
            </append>
      </config>
```

The new file will look like this:

File: KatanaSample\Config\Katana.xml

```
<configs>
      <!--  This tells SDX to add to the Items.xml -->
      <config name="items">
            <!-- This tells SDX to add the following Items to the bottom of
the Items list -->
            <append xpath="/items">

                  <!-- New item will be Katana -->
                  <item id="" name="katanamichonne">
                        <!-- Extend it from the machete, but add the custom
mesh -->
                        <property name="Extends" value="machete"/>
                        <property name="Meshfile" value="#michonnekatana?
katana" />
                  </item>
            </append>
      </config>

      <!-- Adding a new recipe for the mod -->
      <config name="recipes">
            <append xpath="/recipes" >
                  <recipe name="katanamichonne" count="1"
craft_area="workbench">
                        <ingredient name="forgedSteel" count="20"/>
                        <ingredient name="wood" count="4"/>
                        <ingredient name="leather" count="4"/>
                  </recipe>
            </append>
      </config>
</configs>
```

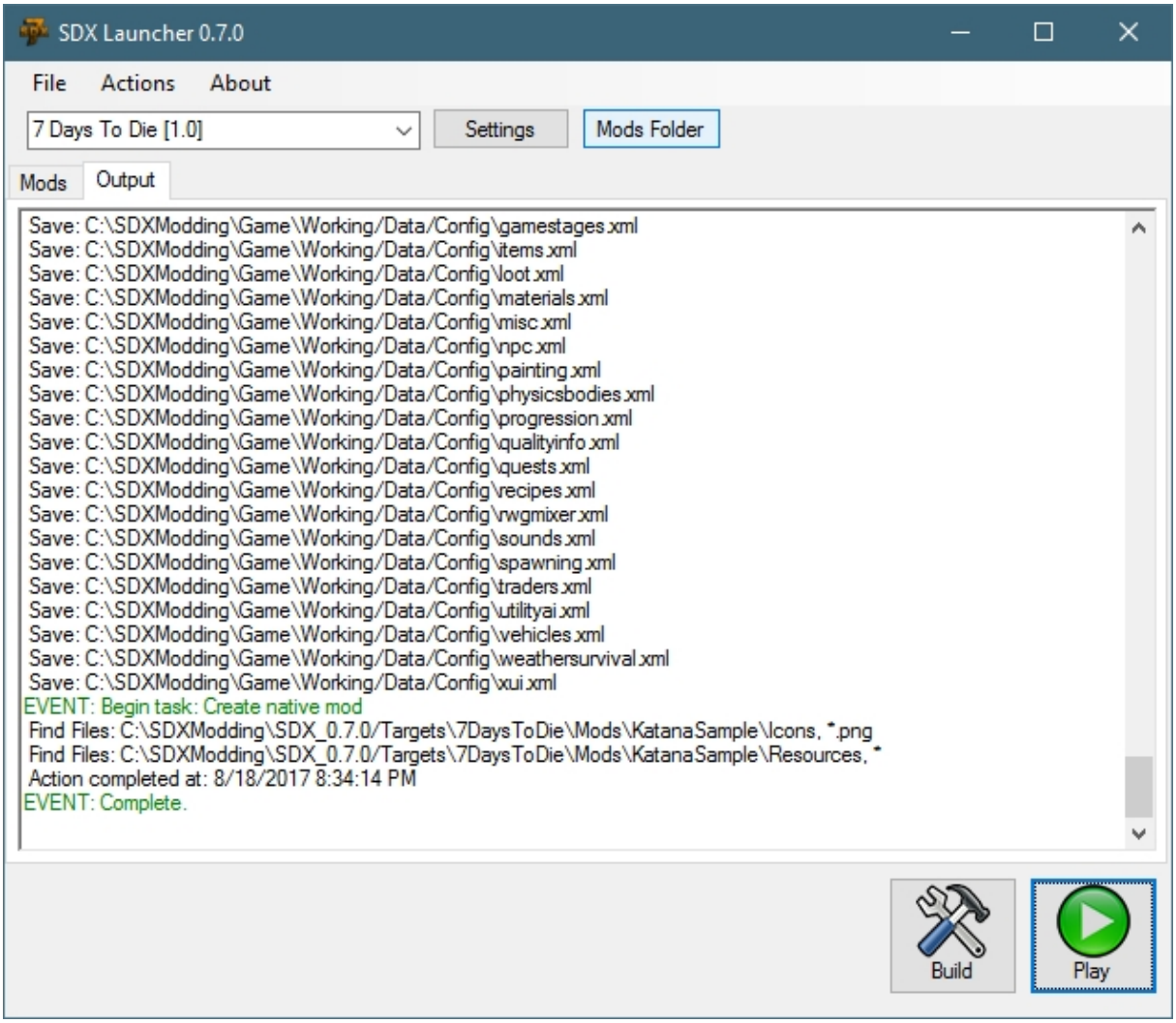
Save your changes, and go back to the SDX Launcher, and do another Build.

Load up your save game, and search for the Katana sword in your Crafting menu

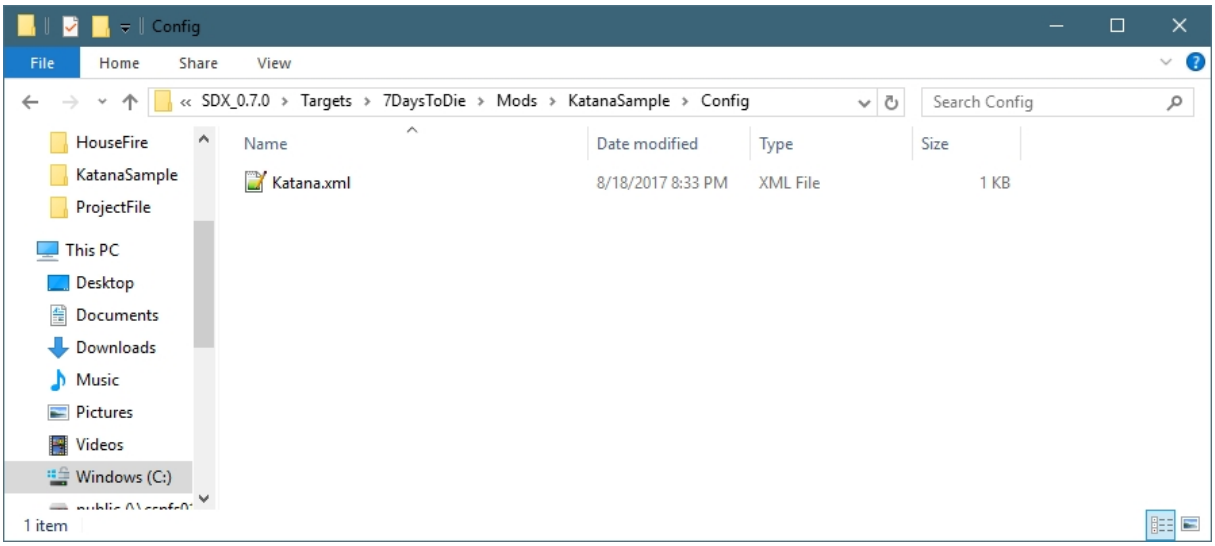
Adding Katana to a Loot Group

With Steel being an expensive resource for a katana, we want to give players the chance of finding one. Let's add the Katana to a loot group.

In the SDX Launcher, click on the Mods Folder.



Then navigate into the double click on the Katana Sample folder, then the Config folder:



Using Notepad++, edit the file

Add a new "<config> node under the katanamichonne recipe by copy and pasting the Loot Snippet, highlighted in Blue for clarity

Loot Snippet

```
	<!-- Let's add the Katana to the melee loot group -->
	<config name="loot">
		<append xpath="/lootcontainers/lootgroup[@name='weaponsMelee']">
			<item name="katanamichonne" prob="0.05" />
		</append>
	</config>
```

So this one is more complex the other ones. Notice the "<append xpath"?

That's an xpath script that tells the SDX Launcher to look in the <lootcontainers> tag, find the <lootgroup> that has the attribute name='weaponsMelee', and appends it to the node list.

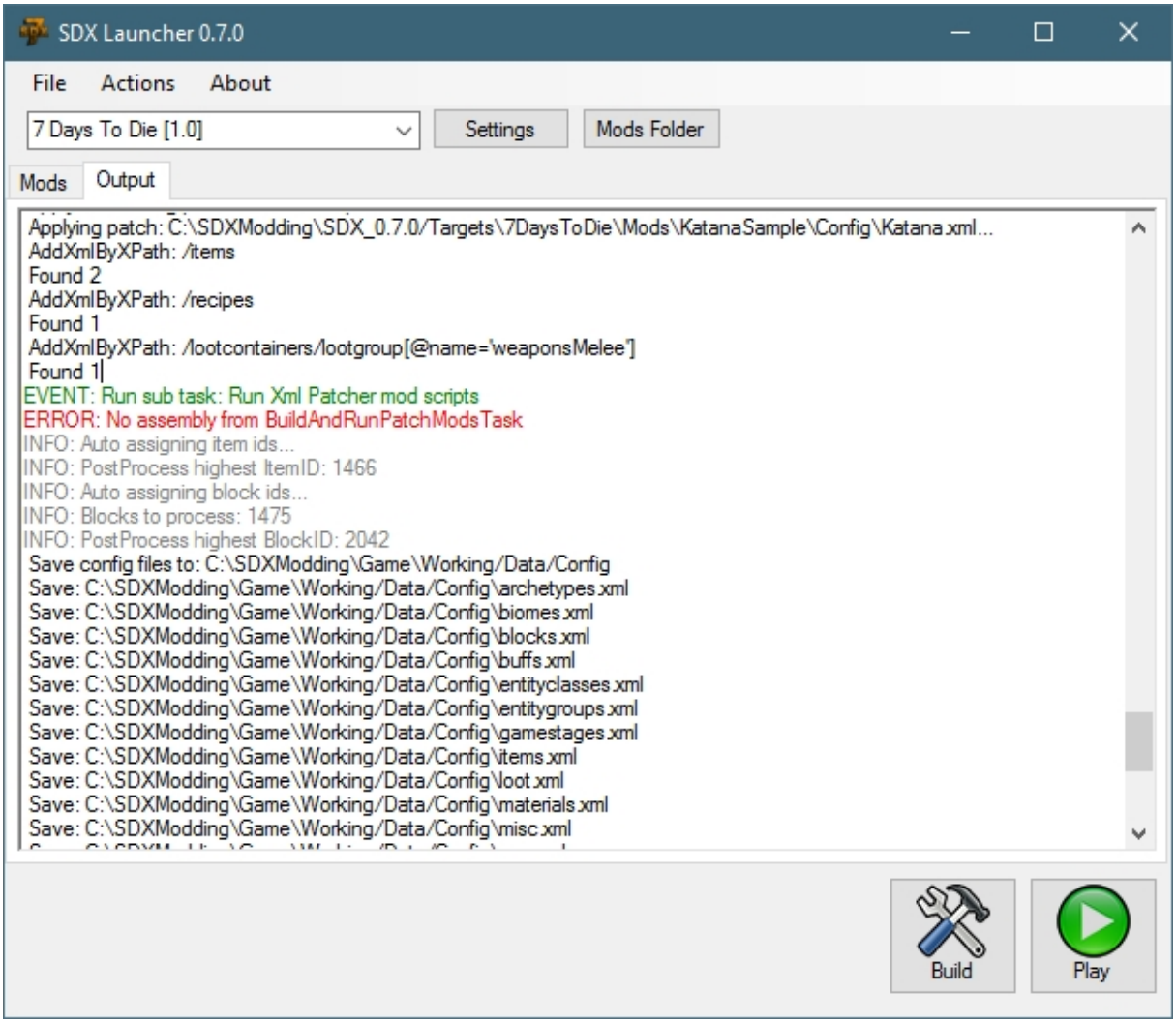
The Katana.xml should look like this now:

File: KatanaSample\Config\Katana.xml

```
<configs>
	<!--  This tells SDX to add to the Items.xml -->
	<config name="items">
		<!-- This tells SDX to add the following Items to the bottom of
the Items list -->
		<append xpath="/items">

			<!-- New item will be Katana -->
			<item id="" name="katanamichonne">
				<!-- Extend it from the machete, but add the custom
mesh -->
				<property name="Extends" value="machete"/>
				<property name="Meshfile" value="#michonnekatana?
katana" />
			</item>
		</append>
	</config>

	<!-- Adding a new recipe for the mod -->
	<config name="recipes">
		<append xpath="/recipes" >
			<recipe name="katanamichonne" count="1"
craft_area="workbench">
				<ingredient name="forgedSteel" count="20"/>
				<ingredient name="wood" count="4"/>
				<ingredient name="leather" count="4"/>
			</recipe>
		</append>
	</config>

	<!-- Let's add the Katana to the melee loot group -->
	<config name="loot">
		<append xpath="/lootcontainers/lootgroup[@name='weaponsMelee']">
			<item name="katanamichonne" prob="0.05" />
		</append>
	</config>
</configs>
```

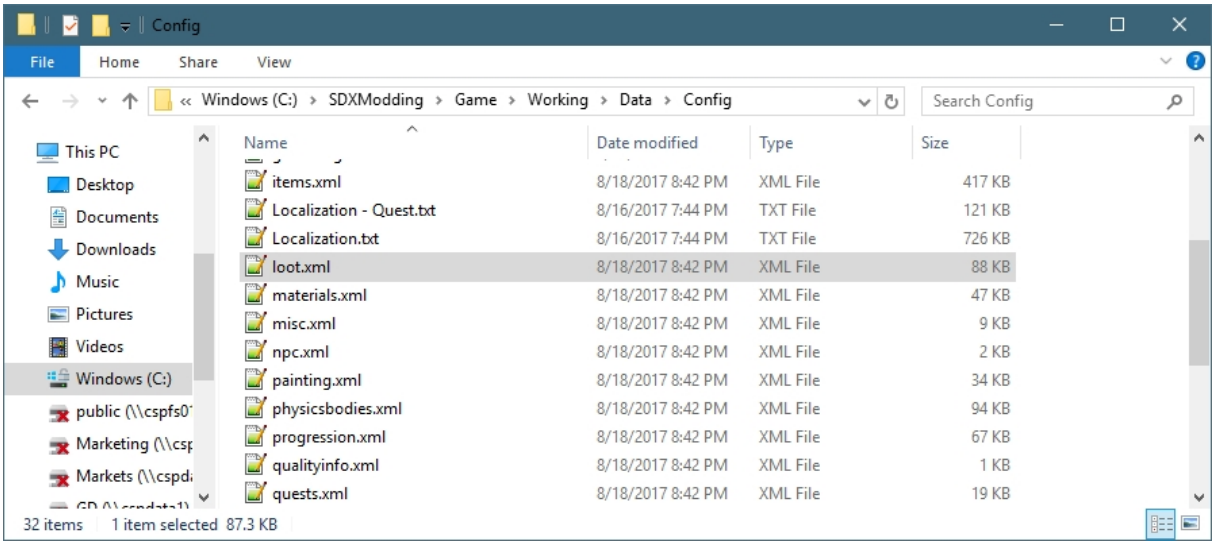
Once you've saved your changes, go back to the SDX Launcher, and click on the Build Button.

If you look through the log file, you'll see where it's adding the new XML:



Remember, the "ERROR: No assembly from BuildAndPatchModsTask" is not a fatal error right now, since we are not compiling any scripts for the Katana

Once compiled, look in the loot.xml of the Working game:



And search for "katana"

File: Data\Config\loot.xml

```
        <item group="weaponsMagnumParts" prob="0.03" />
    </lootgroup>
    <lootgroup name="weaponsMelee">
        <item name="clubWood" />
        <item name="clubIron" />
        <item name="clubBarbed" prob="0.2" />
        <item name="clubSpiked" prob="0.15" />
        <item name="huntingKnife" prob="0.2" />
        <item name="machete" prob="0.1" />
        <item name="katanamichonne" prob="0.05" />
    </lootgroup>
    <lootgroup name="weaponsCrossbow+ammo" count="all">
        <item name="crossbow" count="1" />
        <item name="ironCrossbowBolt" count="10,30" />
    </lootgroup>
```


Tricks and Tips

Now that you've completed the Beginner Tutorial successfully, it's time to let you in on some tricks and tips.

Building for the first time step walked you through how to trigger your first SDX build. At first, we didn't enable the Katana mod, so *nothing* happened, right?

Actually, the SDX Launcher *did* do something to the Assembly-CSharp.dll even without using any SDX mods, and that was instrumenting it with some handy SDX hooks. The initial hooks that SDX does every time it builds, regardless of mods enabled, is to allow loading the unity3d bundles for adding new blocks and items to the game.

Adding new Resources

With these hooks, it's now possible to just copy the KatanaSample's Resource file, michonnekatana.unity3d, into the "*Working\Mods\SDX\Resources*" folder, and reference it in your XML files, adding a new item as you would with traditional XML edits. The property Meshfile is then changed to point to #michonnekatana?katana.

File: Data\Config\items.xml

```
<item id="1466" name="katanamichonne">
        <!-- Extend it from the machete, but add the custom mesh -->
        <property name="Extends" value="machete" />
```

```
            <property name="Meshfile" value="#michonnekatana?katana" />
      </item>
```

When the game starts, the SDX hooks are activated, and as the game parses the XML files, it'll see the special model reference, and look under the Resources folder for a matching model.

#michonnekatana is translated to michonnekatana.unity3d
?katana is translated to look for the katana game object inside of michonnekatana.unity3d.

You can continue to do that, dropping the unity3d bundles into the Resources folder, and making references to the models, without using the SDX Launcher again.

Using Non-Vanilla Files as a Base

For our examples, we have been using the vanilla XML files as a base. This makes a good, consistent starting point to get you comfortable with SDX without getting too overwhelmed.

But that's not the only way you can enjoy SDX mods, nor does it mean you have to piece together a mod yourself. If you have an existing XML mod that you want to use, such as Clockwork Project, or Valmar Overhaul, you can use those mods as a base file.

All you need to do is install the mod, as per the modders instructions, into your Working folder. Verifying the Mod works as-is, and run it through the SDX Launcher. From there, you can copy and paste unity3d files into the Resources folder, and update the XML files to point to the new model.

The Mods/KatanaSample/Config/ Folder

In our examples, we included a Config/Katana.xml

```
<configs>
      <!--  This tells SDX to add to the Items.xml -->
      <config name="items">
            <!-- This tells SDX to add the following Items to the bottom of
the Items list -->
            <append xpath="/items">

                  <!-- New item will be Katana -->
                  <item id="" name="katanamichonne">
                        <!-- Extend it from the machete, but add the custom
mesh -->
                        <property name="Extends" value="machete"/>
                        <property name="Meshfile" value="#michonnekatana?
katana" />
                  </item>
            </append>
      </config>
</configs>
```

It's actually optional in SDX to need a Config/ folder. Once you add in your SDX hooks, you can edit the XML files as you normally would. Be sure to edit your mod.xml file and comment out the Config line:

```
      <config_mods>
            <!--import file="Config\Katana.xml" /-->
```

```
    </config_mods>
```

So why we do we start with a Config folder if we don't need it?

The reason to use a Config folder for a SDX mod is *ease of maintenance and distribution.*

If you code all your XML into the Config folder, using the recommended format, then your XML snippets will be merged into the vanilla files, or into whatever modded files you are using. When a new Alpha release comes out, you'll just need to re-run the SDX Launcher, and merge your changes. With some minor tweaks, which are necessary for each of the Alpha releases, your mod will be ready.

If you are creating a complete SDX overhaul, then you could probably skip the Config folder in your SDX Mod. However, if you are building individual mods to be used by others, then it's best to use the Config folder for your mod.

SDX Intermediate Tutorial

The SDX Intermediate Tutorial assumes you have already followed the "Getting Set up" guide and completed the "SDX Beginner Tutorial"

In our Beginner Tutorial examples, we added in the Katana and a Cube block to the game. These mods added a new unity3d model, and made some XML changes. What it didn't do, was change or add any functionality in the game.

There's two types of coding that can be added in by SDX: PatchScripts and Scripts.

PatchScripts are code that is added as part of the Build process, and make changes directly in the Assembly-CSharp.dll, or more precisely, they *patch* the Assembly-CSharp.dll. PatchScripts are meant to update or change base game features.

Scripts are code that are compiled into the Mods.dll, and loaded at when the game starts. Scripts are meant to add new classes and functionality, often extending the base game.

For the Intermediate Tutorial, we'll explore the Patch Scripts by adding the Bigger Back Pack Mod.

HAL9000 has created a series of extremely valuable SDX 7 Videos that take you step by step through some of the advanced methods

The 7D2D SDX 7 Patch Scripts video takes you step by step through in how to create your first PatchScript.

https://www.youtube.com/watch?v=Yo092Z_Mirk&feature=youtu.be

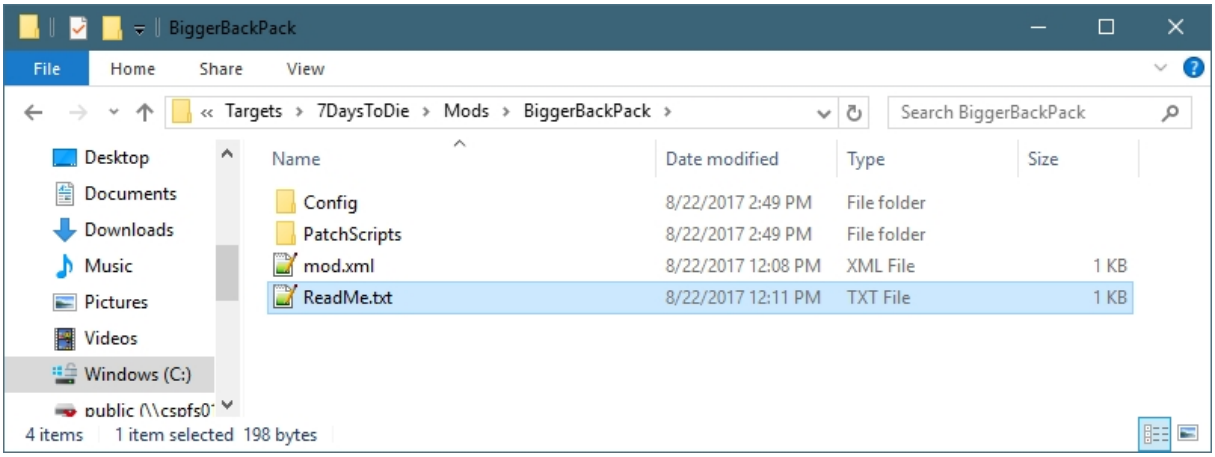
Adding the Bigger Back Pack Mod

The Bigger Back Pack mod. A lot of users love the extra space that it gives you, especially for larger mods that add a lot more diverse lootable items.

We decided to port the Bigger Back Pack mod to SDX, to show how the Patch Script system works, as well as demonstrate another, more advanced Config file.

Download the SDX Mods zip and extract the Bigger Back Pack mod under your Target/Mods/ folder.

The Config folder contains the XML snippet, while the PatchScript contains the build scripts. We'll look at both, and explain what's going on in each one.



Understanding the XML Config

Config\BiggerBackPack.xml:

This mod makes a few changes to files that the other SDX Mods haven't shown you yet. The Windows.xml under the XUi, and the xui.xml file under the Config folder.

```
<configs>
	<config name="XUi/windows">
		<!-- Back pack dimensions are set for 5 x 9, based on a 45 slot
back pack -->
		<set
xpath="/windows/window[@name='windowBackpack']/panel[@name='content']/grid[@
name='inventory']/@rows">5</set>
		<set
xpath="/windows/window[@name='windowBackpack']/panel[@name='content']/grid[@
name='inventory']/@cols">9</set>

		<set
xpath="/windows/window[@name='windowBackpack']/panel[@name='content']/grid[@
name='inventory']/@cell_width">67</set>
		<set
xpath="/windows/window[@name='windowBackpack']/panel[@name='content']/grid[@
name='inventory']/@cell_height">67</set>

		<remove
xpath="/windows/window[@name='windowBackpack']/panel[@name='content']/grid[@
name='inventory']/item_stack" />
		<append
xpath="/windows/window[@name='windowBackpack']/panel[@name='content']/grid[@
name='inventory']">
			<item_stack_inventory name="0"/>
		</append>

	</config>


	<config name="XUi/controls"
>https://raw.githubusercontent.com/7D2DSDX/Mods/master/BiggerBackPack/Config
/BiggerBackPack.xml
		<append xpath="/controls">
			<item_stack_inventory>
				<rect controller="ItemStack" style="itemStack,
hover">
<!-- Snipped for brevity. Check out the full XML here:
https://raw.githubusercontent.com/7D2DSDX/Mods/master/BiggerBackPack/Config/
BiggerBackPack.xml
				</rect>
			</item_stack_inventory>
		</append>
	</config>
	<!-- changing the scale of the panel to better fit -->
	<config name="xui">
		<set
xpath="/xui/ruleset[@name='default']/@stackpanel_scale">1.03</set>
	</config>
</configs>
```

Notice we are using <config name="XUi/windows"> ? This lets SDX know that the windows.xml file is to be searched under the XUI folder. By default, SDX will search for the Data/Config/*.xml files, so for any subfolders off of Data/Config, you'll need to specify the folder name.

The next part about this Config is the <set xpath> line. This line lets us change *individual attributes* in XML nodes, rather than adding a new recipe or block, as we did in the other examples. The Lines look scary, but it's not that bad! It allows us a very precise change in XML files.

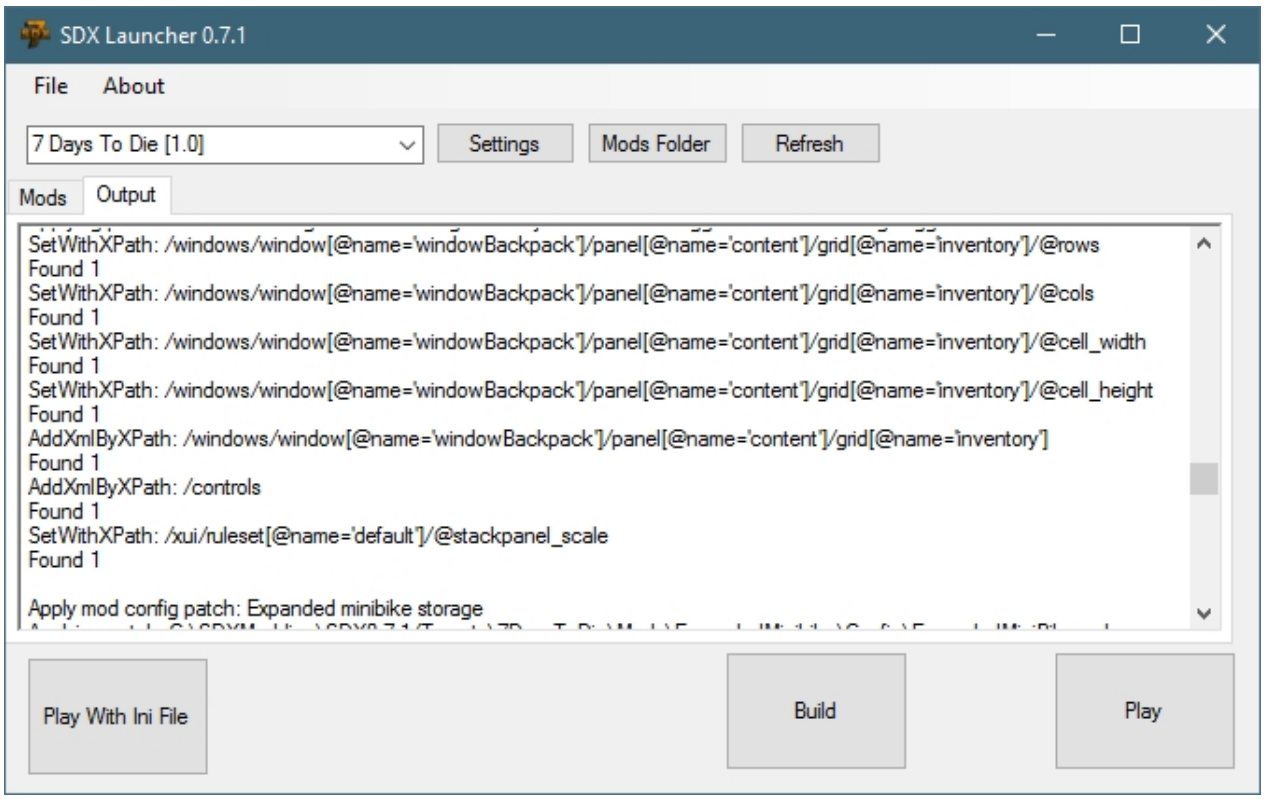
Let's break it down:

```
<set
xpath="/windows/window[@name='windowBackpack']/panel[@name='content']/grid[@
name='inventory']/@rows">5</set>
```

/windows	Top level node in the file </windows>
/window[@name='windowBackpack']	Look for the <window> that has a name of "windowBackpack"
/panel[@name='content']	Look for the <panel> tag that has the name of 'content', that's inside of the above window
/grid[@name='inventory']	Look for the <grid> tag with the name inventory, that's inside of the above panel
/@rows	Look for the rows attribute

There's a website that'll help you building your more complicated xpath:
https://xmltoolbox.appspot.com/xpath_generator.html

When you run it through SDX Launcher, you'll see this:

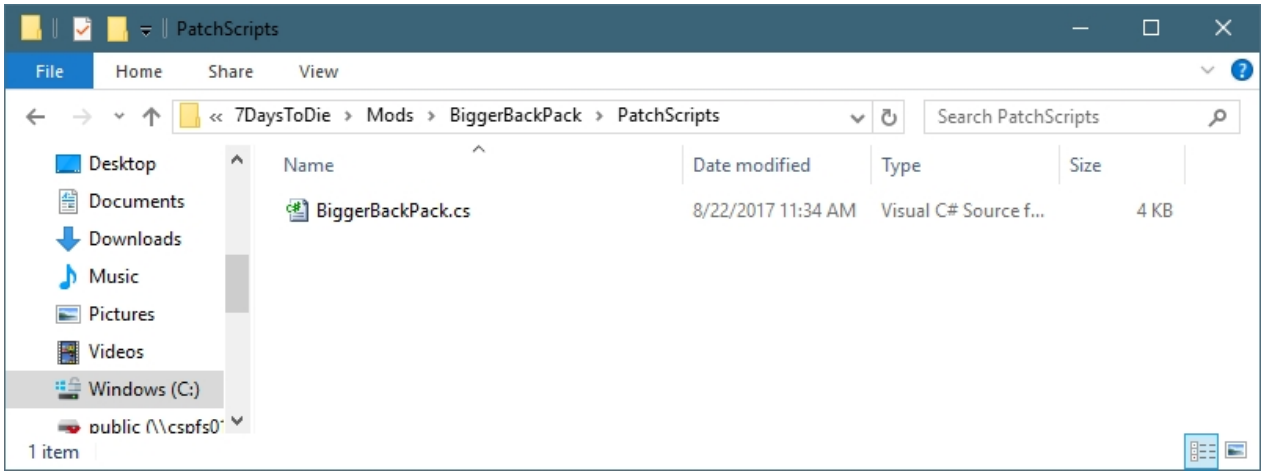


Set With XPath indicates what value it's looking for, and how many matching lines it found.

Take a look at the SDX XPath Configurations for more examples and help on this.

Understanding the PatchScript

The PatchScript for the Bigger Back Pack Mod let's us change the Assembly-CSharp.dll at build time, allowing us to increase the size of the Back Pack.



The PatchScripts are C# Scripts. You can either use Visual Studio, or even Notepad++ for them.

This will not be an in depth tutorial on how to write C#, but rather just to show the anatomy of a SDX PatchScript

```
using System;
using SDX.Compiler;
using Mono.Cecil;
using Mono.Cecil.Cil;
using System.Linq;

public class BiggerBackPack : IPatcherMod
{
    private sbyte OldInventorySize = 32;
    private sbyte NewIntenvotrySize = 45;

    public bool Patch(ModuleDefinition module)
    {

    private void SetAccessLevels(ModuleDefinition module)
    {

    private void SetBackpackSize( ModuleDefinition module)
    {

    // Helper function to update the backPack module
    public void UpdateBackPack(ModuleDefinition module, String strModuleName, String strMethodName, int maxCounter)
    {

    // Called after the patching process and after scripts are compiled.
    // Used to link references between both assemblies
    // Return true if successful
    public bool Link(ModuleDefinition gameModule, ModuleDefinition modModule)
    {


    // Helper functions to allow us to access and change variables that are otherwise unavailable.
    private void SetMethodToVirtual(MethodDefinition meth)
    {

    private void SetFieldToPublic(FieldDefinition field)
    {
    private void SetMethodToPublic(MethodDefinition field)
    {
}
```

The public class BiggerBackPack, inheirits from the IPatcherMod. This IPatcherMod is an SDX class which allows patching. All your PatchScripts need this.

Two functions are required:

bool Patch(ModuleDefinition module)
bool Link(ModuleDefinition gameModule, ModuleDefinition modModule)

The Patch() call does the initial assembly, and is where most of the work gets called at. The Link() happens after the compile. In the Bigger Back Pack Mod, we do not use the Link() method, but it still needs to exists, even if it's only does a simple return true.

The rest of the methods, SetAccessLevels(), SetBackpackSize(), UpdateBackPack(), SetMethodToVirtual, SetMethodToPublic, and SetFieldToPublic() are methods we declared to help us out. Some of the fields in the Assembly-CSharp.dll are private, and therefore cannot be access by SDX without changing.

The SetMethodToVirtual, SetFieldToPublic and SetMethodToPublic are all helper functions that can be called to change these private variables, to public ones.

The supplied BiggerBackPack.cs is a documented C# script. We encourage you to review it, and understand what it's doing.

SDX Advanced Tutorial

The SDX Advanced Tutorial assumes you have already followed the "Getting Set up" guide, and completed the SDX Beginner Tutorial and SDX Intermediate Tutorial.

In the Bigger Back Pack mod, we went over the PatchScripts, and how they were added to the Assembly-CSharp.dll during SDX compile time. Those changes, if you remember, were added inside of the game's DLL.

Scripts, on the other hand, are compiled into a Mods.dll, and loaded during run-time. When the game loads, the Mods.dll is injected into the Assembly-CSharp.dll.

Besides when and how they are loaded, how are they different? And why are they different?

PatchScripts, as we've mentioned before, changes how the base game does things. In our previous examples, we increased storage by adjusting a few pieces of code in the game, and the XML. If you are only doing small adjustments, like we did for the bigger back pack mod, then PatchScripts are fine, and preferred way. However, if we wanted to do more, then PatchScripts would be tedious, and limited to changing existing code.

But what if we wanted to do more with our mods? We aren't limited to just adjusting the way the base game behaves with SDX. Rather, we can use the Scripts functionality to add completely new objects into the game, including new animations, new blocks triggers, and new ways of interacting with the world.

For the Advanced Tutorial, we'll explore the Scripts by adding in Three08's Neon Sign Mod.

Note: The Advanced Tutorial covers C# .NET code, and it's assumed the reader has at least some understanding of the concepts in C#.

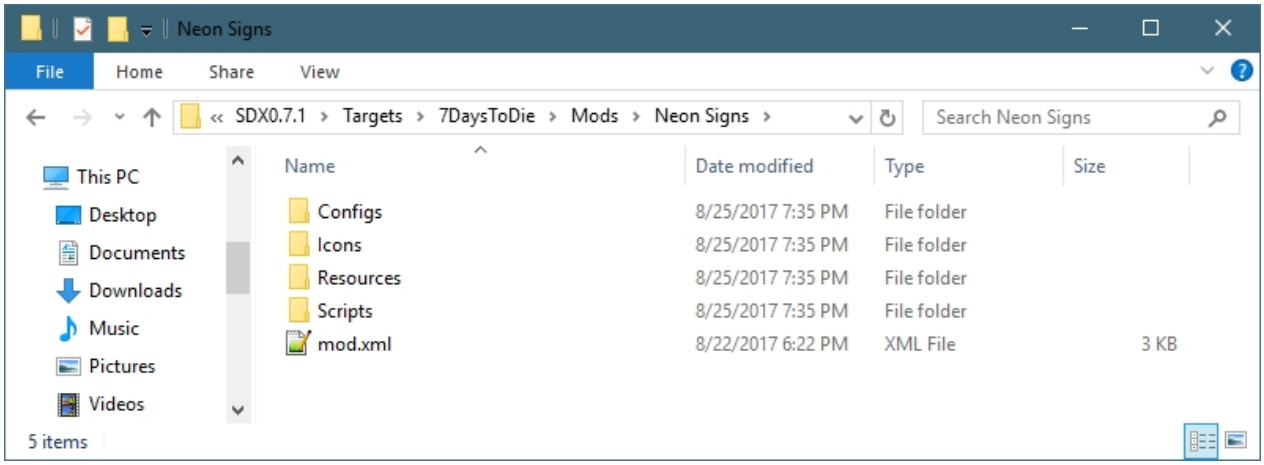
Neon Signs Mod

Three08's Neon Sign adds a stand out asset to the game. Flashing Neons lights of various signs are bound to introduce more variety into your crafting world.

The Advanced Tutorial is going to go through the Neon Sign's steps on how it was created, and once again shows what can be done with SDX.

Reviewing the Folder Structure

The Neon Signs mod adds in custom XML, custom Icon, custom Resource, and custom Scripts to the game.



We've already covered the Configs folder, but we'll touch base again with it to show how three08 is using it, and more precisely, what he's added.

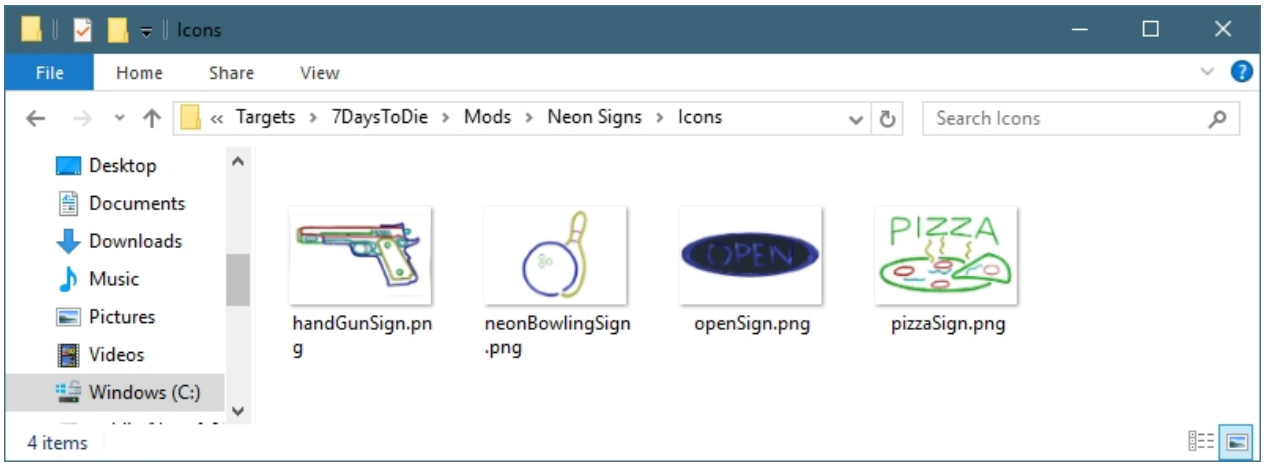
```
<configs>
      <config name="blocks">
            <append xpath="/blocks">
                  <block id="1630" name="Neon Open Sign">
                        <property name="Extends"
value="ceilingLight02_player" />
                        <property name="CustomIcon" value="openSign" />
                        <property name="Class" value="NeonSign, Mods" />
                        <property name="Model" value="#NeonSign?
NeonOpenSignPrefab"/>
                        <property name="Collide"
value="movement,melee,rocket" />
                        <property name="HandleFace" value="South" />
                        <property name="TakeDelay" value="5" />
                        <property name="AllowRemotePower" value="false" />
                        <property name="LightObject" value="NeonText" />
                  </block>
 <!-- Snipped contents -->
```

Three08's Neon Sign mod contains a few new blocks and recipes. In the one above, he's Extending the ceilingLight02_player block.

He's also referencing a CustomIcon, which we will find under the Icons folder. Even the Model value looks familiar, making a reference to the #Neon?NeonOpenSignPrefab (#Neon = Filename is Neon.Unity3d, and NeonOpenSignPrefab is the asset in it).

What is different, *is the "Class" line*. It's referencing "NeonSign, Mods". That tells the game that there's special code to be run for it, located in the Mods.dll, called NeonSign. We'll go over this new NeonSign Class in the next section.

The Icons and Resources folders, as we've shown previously, contains the custom icons and unity3d files for the mod.

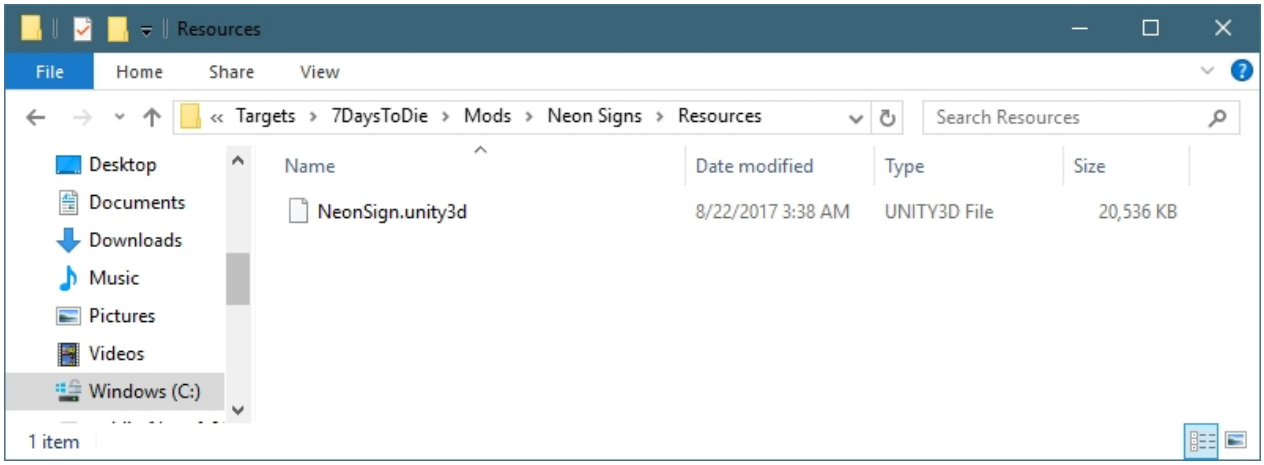


So three08 added 4 new custom icons for the game, with only one unity3d file (shown below). In our previous examples, there's was usually just one model in each unity3d file. However, that doesn't mean that there can't be more. A unity3d bundle can have many custom models embedded in it, being referenced individually using the ?<model name> reference, while keeping the #<BundleName> the same.

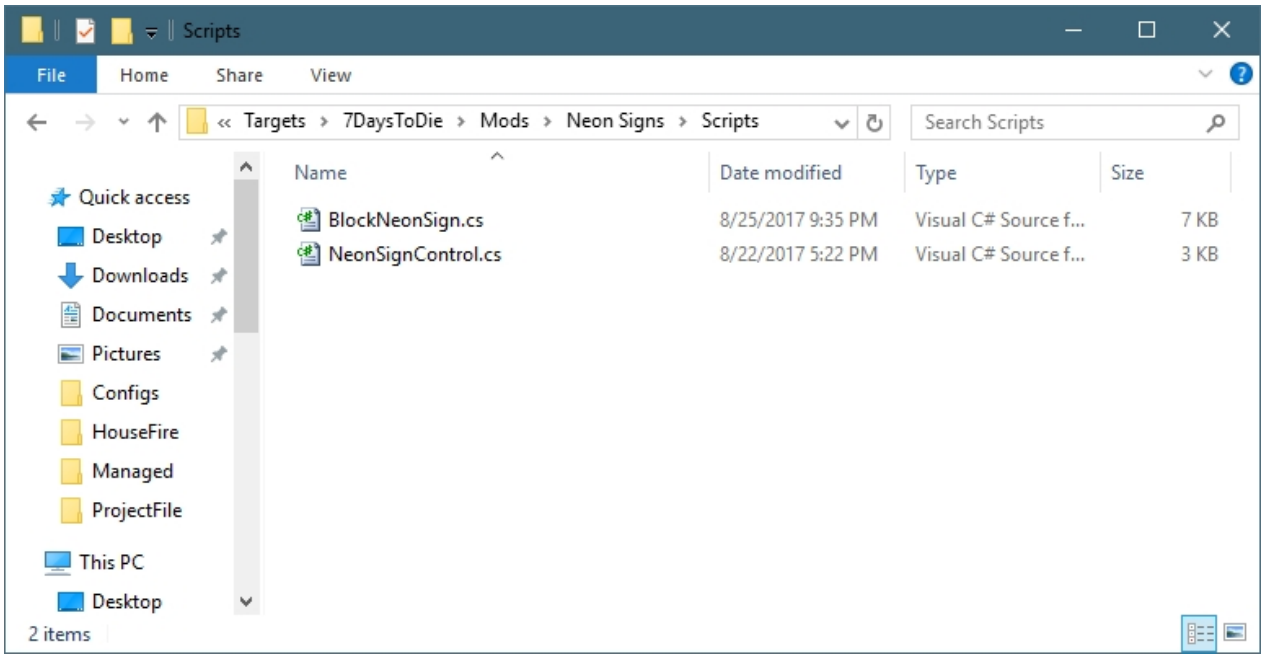
If you look at the Neon Sign XML file, you'll see the following references. We only copied the relevant lines, rather then entire blurb.

```
<property name="Model" value="#NeonSign?BowlingNeonSignPrefab"/>
<property name="Model" value="#NeonSign?NeonOpenSignPrefab"/>
<property name="Model" value="#NeonSign?NeonOpenSign2Prefab"/>
<property name="Model" value="#NeonSign?PizzaSignPrefab"/>
<property name="Model" value="#NeonSign?GunNeonSignPrefab"/>
```

So we see in his Mod that he actually has 5 custom models, and they are all stored in the same unity3d file.



The Scripts folder, however, is new for us.

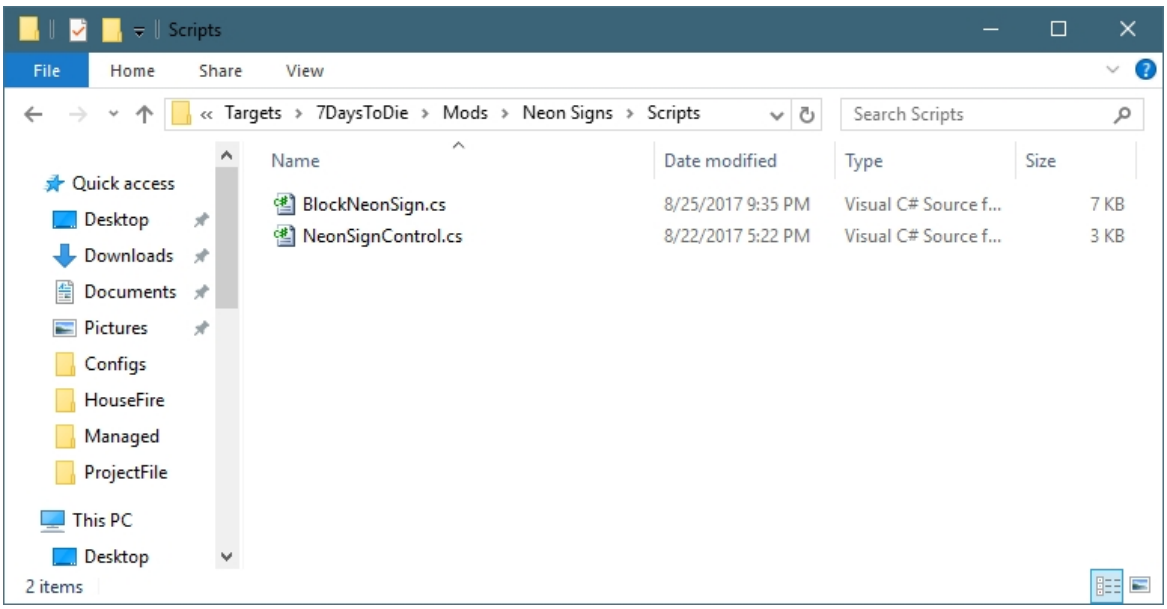


Let's explore on what these files are, and how they are different than the PatchScripts in the Bigger Back Pack Mod.

Understanding the Scripts

Under the Scripts folder, you'll see you files:

BlockNeonSign.cs and NeonSignControl.cs.



Let's open BlockNeonSign.cs using your favorite text editor. In the example below, we are using Visual Studio, and collapsing all the methods so we just need to see their names.

We are not going to cover all of three08's lines of code, since this isn't a C# Tutorial, but we'll cover the important parts.

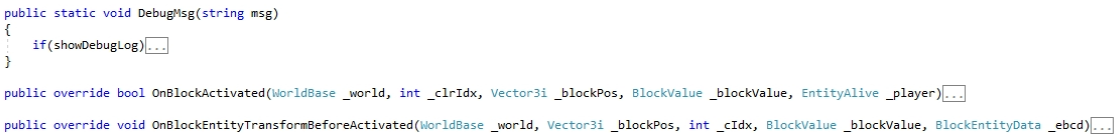
```
using System;
using UnityEngine;

public class BlockNeonSign : BlockPowered
```

The "public class BlockNeonSign : BlockPowered" is the key part we are looking at.

BlockNeonSign is three08's class that he wrote, and is called a *derived* class.

BlockPowered is the *base* class, which means it's actual code in in the base game.

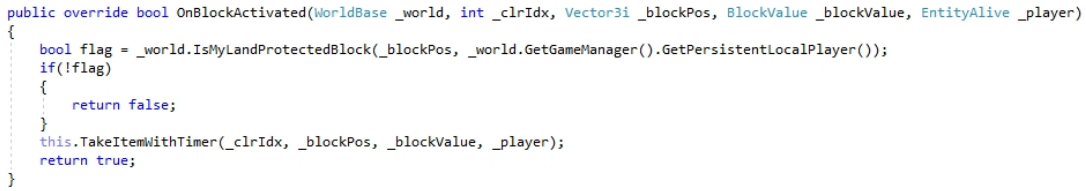
Because BlockNeonSign is derived from the BlockPowered class, it has all the abilities of the base class of BlockPowered. By making a *derived* class, three08 is saying "I want to do everything that the BlockPowered class does, but I want to make changes to some things'.



If you look above, you can see some calls with the keyword "*override*", such as "OnBlockActivated", and "OnBLockEntityTransformBeforeActivated". That keyword indicates that the base class, BlockPowered, has those functions, and that three08 wants to make changes to those. The methods that do not have override as a keyword, is new functionality that three08 added.

By extending a class, three08 gets all the functionality of an existing class, without copying or duplicating it, and changing the methods that he wants to behave differently.

Three08's OnBlockActivated:



Base Game OnBlockActivated:

```
// Token: 0x0600169A RID: 5786 RVA: 0x000D1D90 File Offset: 0x000CFF90
public override bool OnBlockActivated(int, WorldBase, int, Vector3i, BlockValue, EntityAlive)
{
}
```

So the base class for BlockPowered doesn't do anything when OnBlockActivated is called. However, Three08 wanted to do something. In his method, he makes a quick check to see if the block is within your Land Claim area. If it isn't, then do anything. However, if it is within your land claim block, he let's you pick up the Neon Sign.

That was a simple override example. So why did he choose to override OnBlockActivated? The default BlockPowered was not a block you could pick up once your placed it, however, three08 wanted all the benefits of the powered block, but also wanted the ability to pick it up too.

If he had decided to use a PatchScript to allow his blocks to be picked up, he would have had to make all the BlockPowered blocks have the same feature. With this change, only his powered Neon Lights are able to be picked up.

He over-rides another method too, called "OnBlockEntityTransformBeforeActivated". He wrote more code in this method than the first one, as its more complicated.

```
public override void OnBlockEntityTransformBeforeActivated(WorldBase _world, Vector3i _blockPos, int _cIdx, BlockValue _blockValue, BlockEntityData _ebcd)
{
    this.shape.OnBlockEntityTransformBeforeActivated(_world, _blockPos, _cIdx, _blockValue, _ebcd);
    DebugMsg("OnBlockEntityTransformBeforeActivated");

    try
    {
        if (_ebcd != null && _ebcd.bHasTransform)
        {
            GameObject gameObject = _ebcd.transform.gameObject;
            if (LitObject() == null)
                DebugMsg("LitObject is null!");


            GameObject litSignObject = _ebcd.transform.Find(LitObject()).gameObject;
            if (litSignObject == null)
            {
                DebugMsg("litSignObject is null");
            }
            else
                DebugMsg("Found litSignObject");
            NeonSignControl neonSignScript = gameObject.GetComponent<NeonSignControl>();
            if (neonSignScript == null)
            {
                neonSignScript = gameObject.AddComponent<NeonSignControl>();
            }
            neonSignScript.enabled = true;
            neonSignScript.cIdx = _cIdx;
            neonSignScript.blockPos = _blockPos;
            neonSignScript.litSignObject = litSignObject;
            neonSignScript.litSignObject.active = false;
        }
        else
            DebugMsg("ERROR: _ebcd null (OnBlockEntityTransformBeforeActivated)");
    }
    catch (Exception ex)
    {
        DebugMsg("Error Message: " + ex.ToString());
    }
}
```

In this method, he's doing a few interesting things:

```
            this.shape.OnBlockEntityTransformBeforeActivated(_world, _blockPos, _cIdx,
_blockValue, _ebcd);
```

His first line in his method call is calling this. He's telling the game that he wants still do everything that base class does. But he wants to do more with it after the base class is done.

Let's take a look at the other interesting thing being done in this method:

```
GameObject litSignObject = _ebcd.transform.Find(LitObject()).gameObject;
if (litSignObject == null)
{
    DebugMsg("litSignObject is null");
}
else
    DebugMsg("Found litSignObject");
NeonSignControl neonSignScript = gameObject.GetComponent<NeonSignControl>();
if (neonSignScript == null)
{
    neonSignScript = gameObject.AddComponent<NeonSignControl>();
}
```

He's calling another class, called NeonSignControl. If we look back in the Scripts folder, we'll see the other C# script called NeonSignControl.cs.

Let's take a look at the NeonSignControl.cs

```
using System;
using UnityEngine;

public class NeonSignControl : MonoBehaviour
{
    public int cIdx;
    public Vector3i blockPos;
    public GameObject litSignObject;
    private bool isSignActive;
    private int flashSpeed;
    private DateTime nextStateChangeTime;
    private bool flash;
    private bool flicker;

    void Awake()...

    void Update()...

    private void GetColorSlots(Block block)...

    private void SetColor(GameObject _slotObject, Vector3 _colorVector)...
}
```

In his class, he's only using a base class of MonoBehaviour. It has a few methods, such as Awake and Update. These method names are special for MonoBehaviour, in that when a game tick occurs, it will call the Update() method calls for each class that is connected to it. In this case, each tick, the Update() call of the NeonSignControl executes.

Let's take a quick look at the Update() method call to see what's it's doing.

```
void Update()
{
    if(BlockNeonSign.isBlockPoweredUp(blockPos, cIdx))
    {
        isSignActive = true;

    }
    else
    {
        isSignActive = false;
        nextStateChangeTime = default(DateTime);
    }
    if(isSignActive)
    {
        if(litSignObject != null && flicker)
        {
            litSignObject.active = !litSignObject.active;
            return;
        }
        if(flash)
        {
            //Flashing
            if(nextStateChangeTime == default(DateTime))
            {
                nextStateChangeTime = DateTime.Now;
            }
            if (DateTime.Now > nextStateChangeTime)
            {
                if(litSignObject != null && litSignObject.active)
                {
                    litSignObject.active = false;
                }
                else
                {
                    litSignObject.active = true;
                }
                nextStateChangeTime = DateTime.Now.AddSeconds(flashSpeed);
            }
        }
        else
        {
            //No Flashing
            if(litSignObject != null && !litSignObject.active)
            {
                litSignObject.active = true;
            }
        }
```

The first thing it does is check to see if the NeonLight is currently powered or not. If it's not, don't both doing anything else. However, if it is connected to power, than the light is active.

The next section of the code deals with flashing the neon lights on and off. This allows the Neon Lights to flicker on and off.

The Animation SDX mod

HAL9000 originally did a proof of concept for adding in custom animations through SDX. We've taken his solid base, and expanded on it, making it a bit more flexible in what it can do.

So, what can the Animation SDX mod do? It allows us to add in 3D models, and have them animated. This includes adding new zombies, NPCs, and animals into your game, using different animations than what's available in the base game.

The Animation SDX mod uses Unity calls to play various animations that you can set up in Unity. These calls do not rely on internals of the game to function, so they should be fairly resilient against changes between alphas. HAL9000's original code was built around Alpha 14.x, and still mostly works. We've updated it to work more reliably with A16.x and beyond, and made it a bit easier to use.

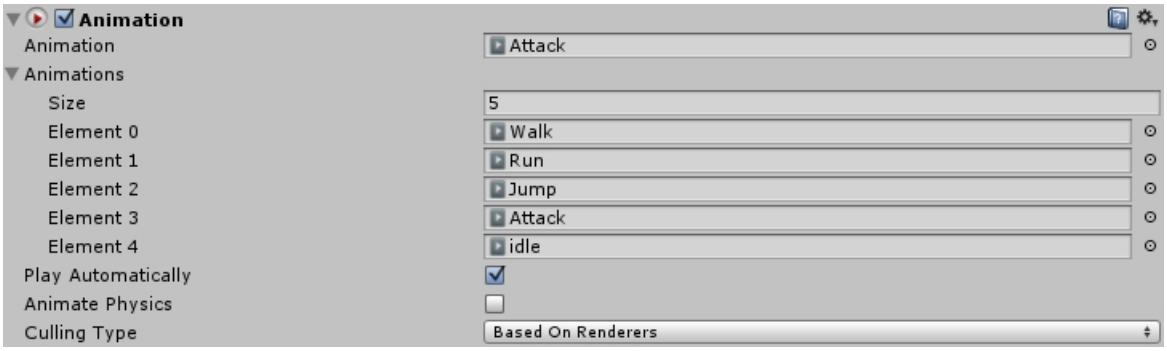
AnimationSDX Class

The Animation SDX mod relies on a new class called "`AnimationSDX`". It exposes Unity calls that allow us to add in assets with custom animations, which we can control through the XML files.

Originally, there was separate classes for animals vs BipedAnimations, however, there's little need for the BipedAnimations class anymore, so we are funneling everything through the AnimalAnimationSDX.

Using the Animation SDX class, with a new custom Unity asset, we can make calls directly to the embedded animation in the unity3d file.

For example, in the Red Samurai How To, we had a few existing animations:



By using the animation names in the Elements, we can populate the following new properties in the XML.

```
        <!-- These are the animation map names. The value comes from the animation
from Unity -->
        <property name="AnimationIdle" value="Idle" />
        <property name="AnimationSecondIdle" value="Idle2" />
        <property name="AnimationMainAttack" value="Attack(2)" />
        <property name="AnimationSecondAttack" value="Attack(3)" />
        <property name="AnimationPain" value="Get_Hit" />
        <property name="AnimationJump" value="Jump" />
        <property name="AnimationDeath" value="Dead" />
        <property name="AnimationRun" value="Run" />
        <property name="AnimationWalk" value="Walk" />
        <property name="AnimationSpecialAttack" value="Attack(4)" />
```

The above properties are what is currently supported by the Animation class. Some are optional, such as AnimationSpecialAttack, AnimationSecondAttack, etc. However, it's best to try to populate them as best as you can for a full experience.

When compiled, a new entity_class is added to the entityclasses.xml called "SDXTemplate". All new entities that want to use this class should inherit from this, or copy it's attributes

```
      <entity_class name="SDXTemplate" extends="zombieTemplateMale">

            <!-- For fun, let's use the zombie transmogrifier class -->
            <property name="Class" value="EntityZombieSDX, Mods" />

            <!-- We are using the Animal Animation, since the BipedAnimation
is a bit wonky with the hit animation -->
            <property name="AvatarController" value="AnimationSDX, Mods" />

            <!-- With RootMotion true, the zombie just kind of chills out in
one place. Set to false to watch it move. -->
            <property name="RootMotion" value="false" />

            <!-- Baseline Speed is way too slow, so we need to bump it up a
notch or 5 -->
            <property name="WanderSpeed" value="0.8" />
            <property name="ApproachSpeed" value="0.8" />
            <property name="NightWanderSpeed" value="0.8" />
            <property name="NightApproachSpeed" value="1.1" />
            <property name="HasRagdoll" value="false" />
      </entity_class>
```

This changes the Class to use the ZombieTransmogrifier mod. However, you can change this back to the regular zombie class.

The AvatarController is already pointing to the AnimationSDX class.

RootMotion is disabled, as it causes the new entities to not move around if its enabled.

We bump up the default speeds, as there are internal modifiers that affect the zombie speeds that are not available to the new AvatarController.

SDX XPath Configurations

SDX supports different ways to insert XML files from your mod's Config folder.

set - Replaces attributes
append - Adds a node or nodes
remove - Removes a node or nodes
insertBefore - Allows you to add a node before another node
insertAFter - Allows you add a node directly after another node

Set Example:

The Bigger Back Pack mod includes an example for the <set> tag

```
      <set
xpath="/windows/window[@name='windowBackpack']/panel[@name='content']/grid[@na
me='inventory']/@rows">5</set>
```

Append Example:

The Katana Mod includes an example for the <append>. Basically, it tells SDX to add the following snippets to the bottom of the node.

```
      <append xpath="/items">
            <!-- more XML Nodes -->
      </append>

In the above example, it will include all the subsequent nodes at the bottom
of the Items list.


remove Example:

The remove node can be used to remove a complete node.

      <remove xpath="/items/item[@name='club']" />

This tells SDX to remove the club item.


insertBefore / insertAfter

This tells SDX to insert the node before, or after the xpath location.

      <insertAfter xpath="/items/item[@name='club']" >
            <!-- mode XML Nodes -->
```

```
    </insertAfter>
```

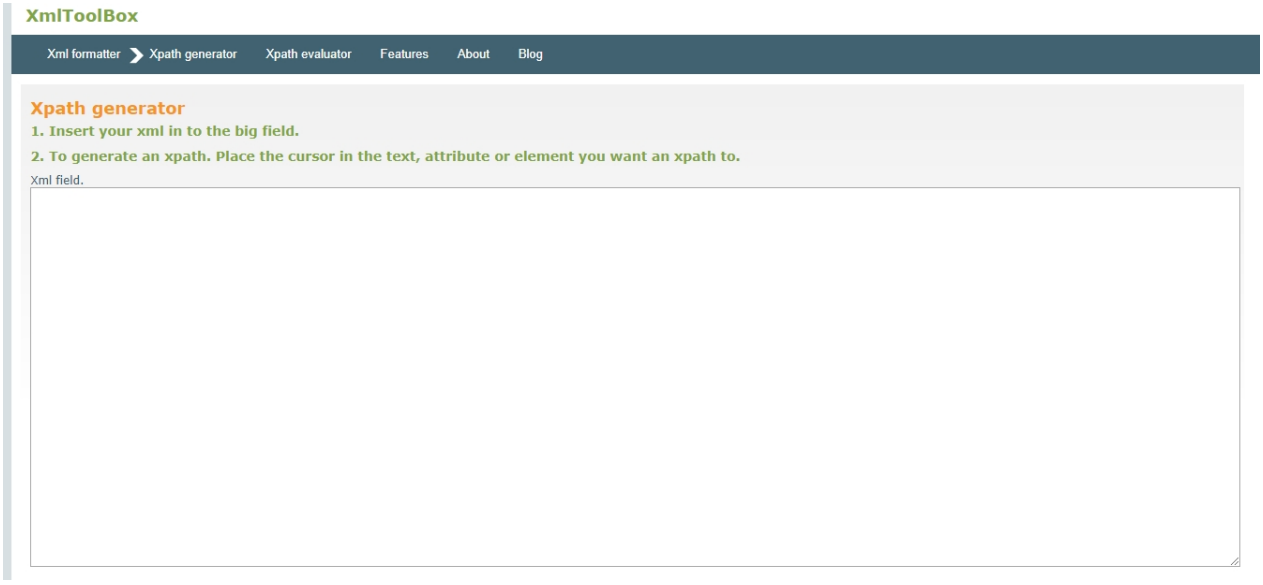
Any XML nodes will be inserted after the Club item.

Creating an XPath Line

Now that you've completed the Intermediate Tutorial successfully, it's time to show you a bit about XPath

The XML portion of the Bigger Back Pack mod can be a bit confusing, because of the xpath.

I recommend using a site like https://xmltoolbox.appspot.com/xpath_generator.html to help generate your xpath. You will still need to make some changes, but it'll go a long way.



In the XML Input, copy and paste the window.xml file, found under your Data/Config/XUi/windows.xml

Once the XML is copy / pasted, you can now click on different values inside of the XML box to get it's Xpath result:

Note: I highlighted the 'controller' name for visibility. In order to get the XPath value, you do not need to highlight, just click on it.

The XPath result it came up with is: /windows/window[1]/grid[@name="hud"]//rect[1]@controller

While it may work, we can actually fix it to be even more reliable. By default, it's trying to use /window[1]/, so the first window in the file. However, if we are using a modded windows.xml, or if the vanilla one changes in the future, then this script won't return what you want. Looking at the xml, we know that the window name is actually "HUDLeftStatBars"

Let's fix it:

```
/windows/window[@name='HUDLeftStatBars']/grid[@name="hud"]//rect[1]
@controller
```

The @name= allows us to specify the name attribute asa string. For this example, we want to specify exactly HUDleftStatBars. If this window ever changes spots in the future, the xpath with the name will still return the right window, while the generated one would likely fail.

The grid, mysteriously, has used the @name tag to make sure the right one is found. So we don't need to fix that one.

The rect[1] value, however, does need to be fixed. For this, we don't have a name attribute, so we need to find another unique attribute to use.

```
<rect width="168" height="43" controller="HUDStatBar" stat_type="Stamina"
```

```
visible="{statvisible}">
```

Width, Height, Controller, and visible are not unique, as other <Rect> nodes have it. But the stat_type is unique.

Let's flesh it out:

```
/windows/window[@name='HUDLeftStatBars']/grid[@name="hud"]//rect[@stat_type
="Stamina"]@controller
```

By default, that website uses double quotes to wrap around strings. We'll want to change those to single quotes, as our entire xpath will be wrapped around double quotes.

If we wanted to change that line, our Config xpath command would look like this:

```
    <set
xpath="/windows/window[@name='HUDLeftStatBars']/grid[@name='hud']//rect[@stat_t
ype='Stamina']@controller">HUDStatBar2</set>
```


Quick Start

Impatient? Already went through the Tutorial and just looking to get started without going through each step?

Here's the short and sweet story of building and running your first SDX mod.

1) Download the SDX Modding Kit
2) Extract the SDX Modding Kit. *For the purpose of this tutorial, it's assumed to be under C:\SDXModding or D:\SDXModding*
3) Copy a vanilla copy of 7 Days To Die to C:\SDXModding\Game\Clean Install\
    - This will be your back up, in case Steam pushes an update.
4) Copy another copy of 7 Days to Die to C:\SDXModding\Game\Working
    This will be the copy of the game that will have SDX added
5) Go into C:\SDXModding\SDX0.7.1\ in Explorer
6) Double click on SDX7DTD.exe
7) Click on the Settings button, and navigate to C:\SDXModding\Game\Working\
8) Click on Build
9) Click on Play

Video Tutorials

This tutorial was meant to guide you through the initial steps of getting SDX, installing it and its support tools, as well as compiling the sample mods.

A series of video tutorials have been created to help you with other tools, and to give you different perspectives of some of the items listed in this document.

Xyth's YouTube Tutorials

Watch Xyth's Tutorials Playlist, and let him take you on a journey through all things SDX and Unity, as he goes step by step through the entire processes for:

Video Tutorial for the 7D2D SDX Tutorials

Video Tutorial on creating new assets, and adding them to the game

Pacco's YouTube Tutorials

Ready to take your Unity journey even further? Jump right into the action with Pacco's thorough walkthrough on how to do such amazing things as:

SDX Ranged Weapon Tutorial

PowerMod Tutorials, covering WorkBenches, Generators, and Mining Machines

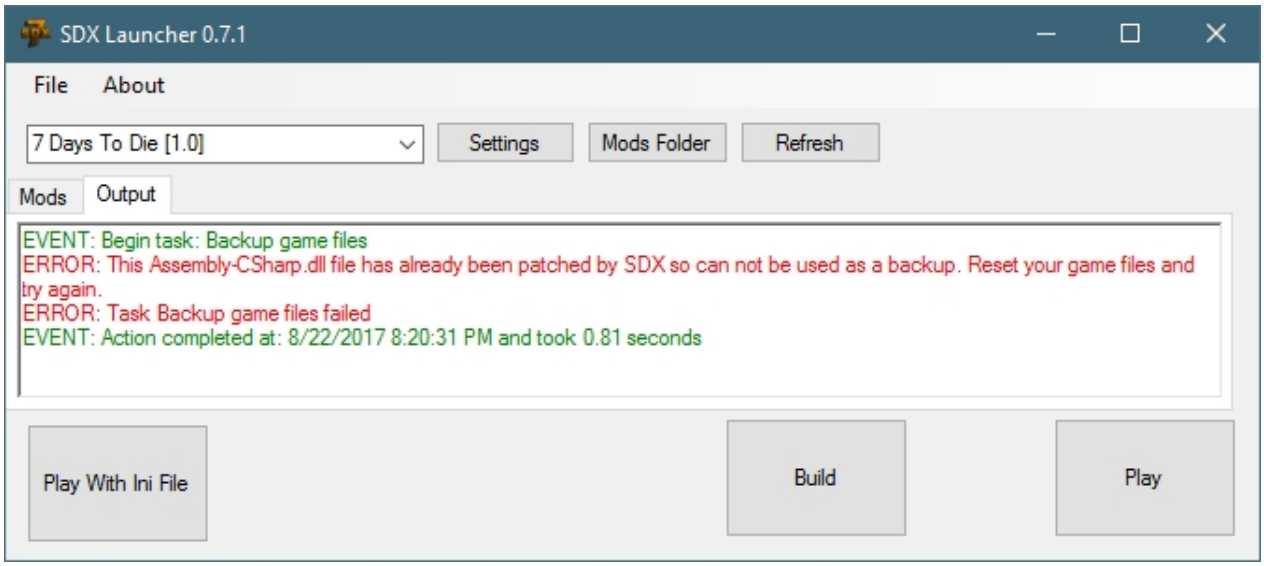
PatchScripts and Custom Entities

HAL9000 has created some very helpful YouTube videos showing different things you can do with SDX.

7D2D SDX 7 - Patch Scripts

Troubleshooting

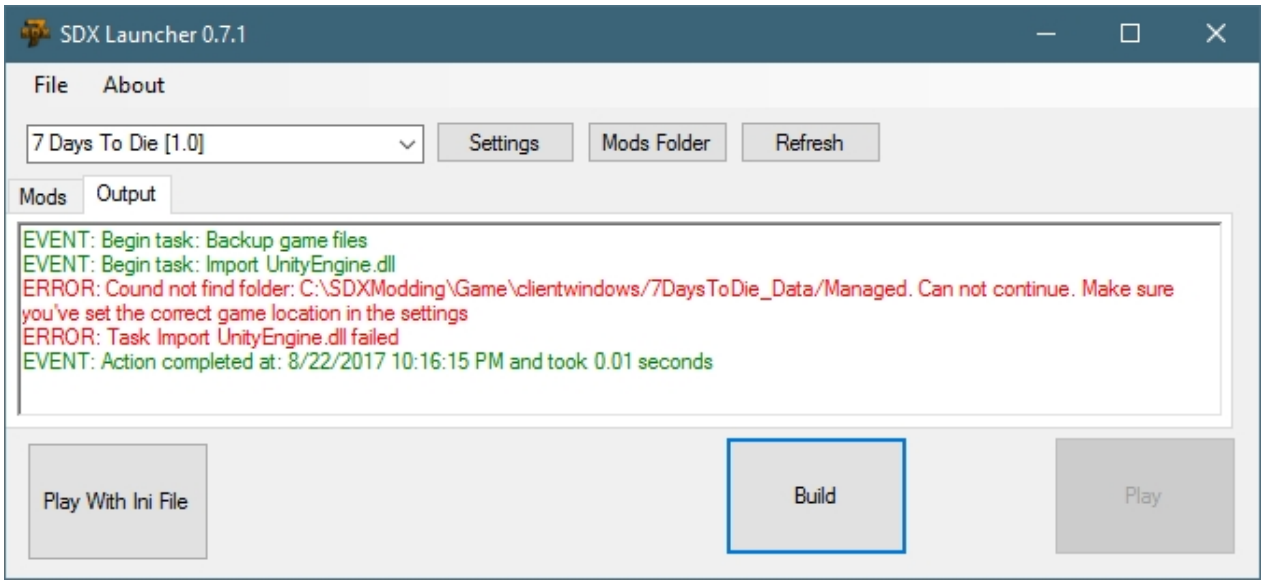
When you press the Build button, you get the following error:



This error gets displayed when you have no local back up of a vanilla file, and the Working folder has already been re-instrumented.

To Fix, follow the instructions in the Start off Clean section.

When you press the Build button, you get the following error:



This indicates that the Path specified in the Settings button isn't pointing to a 7 Days To Die folder.

To Fix, click on the Settings button, and navigate to a valid 7 Days To Die directory.

Advanced Tools

Overview of Tools

The Included Software

SDX 0.7.x

SDX is used to compile the individual SDX scripts into Mods.dll, add hooks into the Assembly-CSharp.dll, and copies the Resources and icons over.

GitHub Desktop

For the 7 Days To Die modding community, it is recommended to use GitHub to store and distribute mods. It's free to use, provides a history of changes, and persistent download links.

Unity Assets Bundle Extrator (UABE)

Unity Assets Bundle Extractor (UABE) is a stylish tool that allows editing asset bundles and .assets. It can export .assets files from bundles, import them back, modify most asset formats with plugins and dumps with type information and create a standalone installer from the modifications.

dnSpy

dnSpy is a tool to reverse engineer .NET assemblies. It includes a decompiler, a debugger and an assembly editor (and more) and can be easily extended by writing your own extension. It uses dnlib to read and write assemblies so it can handle obfuscated assemblies (eg. malware) without crashing.

Optional Software

The following software will be useful when you get more comfortable with SDX, and ready to do more advanced tasks, such as converting and creating your own textures.

Unity 5.3.8

Unity will allow to to manipulate assets, and convert them into a format that SDX needs.

Blender

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. This is useful for making new textures and meshes.

Unity 5.3.8p2 Quick Review

Referenced Article

For us, in the 7 Days To Die Community, Unity is the programming environment that is used by 7 Days to Die, as well as the SDX Mods. It allows us to add new models, new textures, and add in scripts that can manipulate the game in ways you've always dreamed about, and in some cases, things you haven't dreamed about.

This section will show you how to install and set up Unity 5.3.8p2.

What is Unity3D?

Unity3D is a powerful cross-platform 3D engine and a user friendly development environment. Easy enough for the beginner and powerful

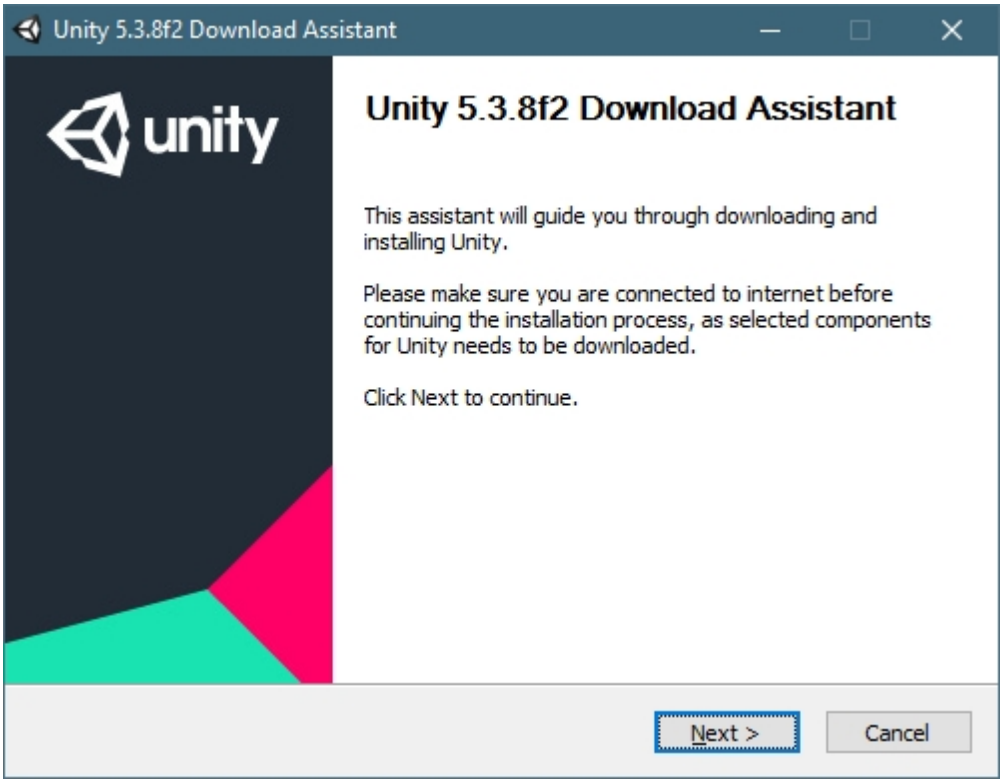
enough for the expert; Unity should interest anybody who wants to easily create 3D games and applications for mobile, desktop, the web, and consoles.

Installing Unity 5.3.8p2

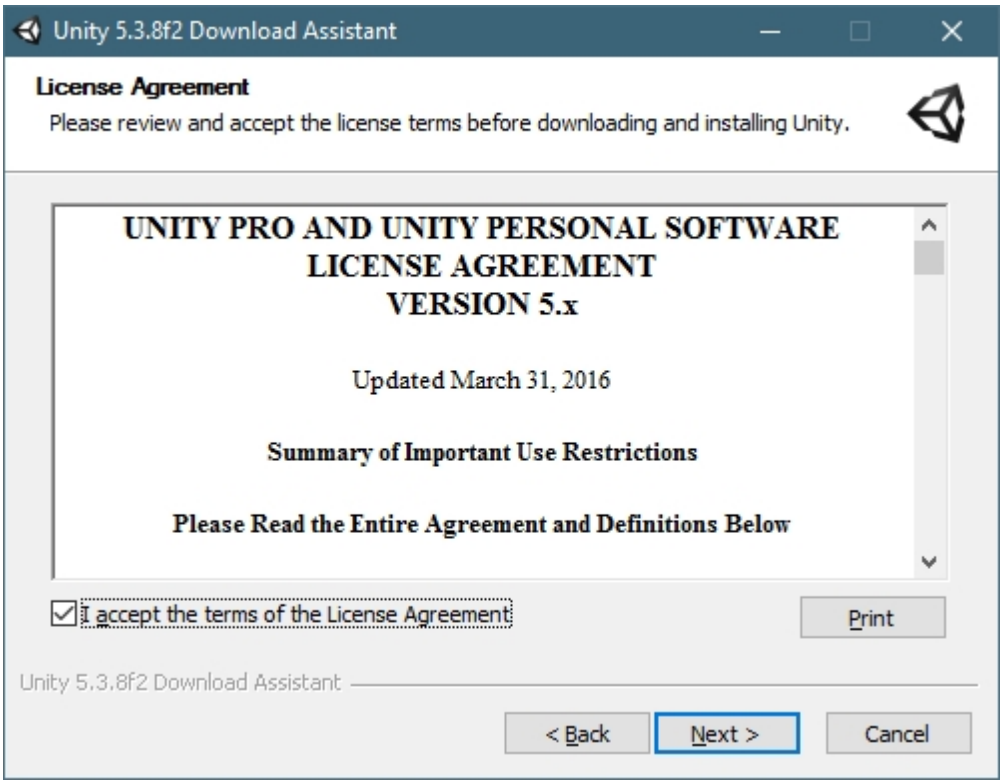
Unity 5.3.8p2

Download and install Unity 5.3.8p2 here. Unity 5.3.8 will allow you to create custom Unity3D bundles, which will be used by SDX to add prefabs, and new blocks.

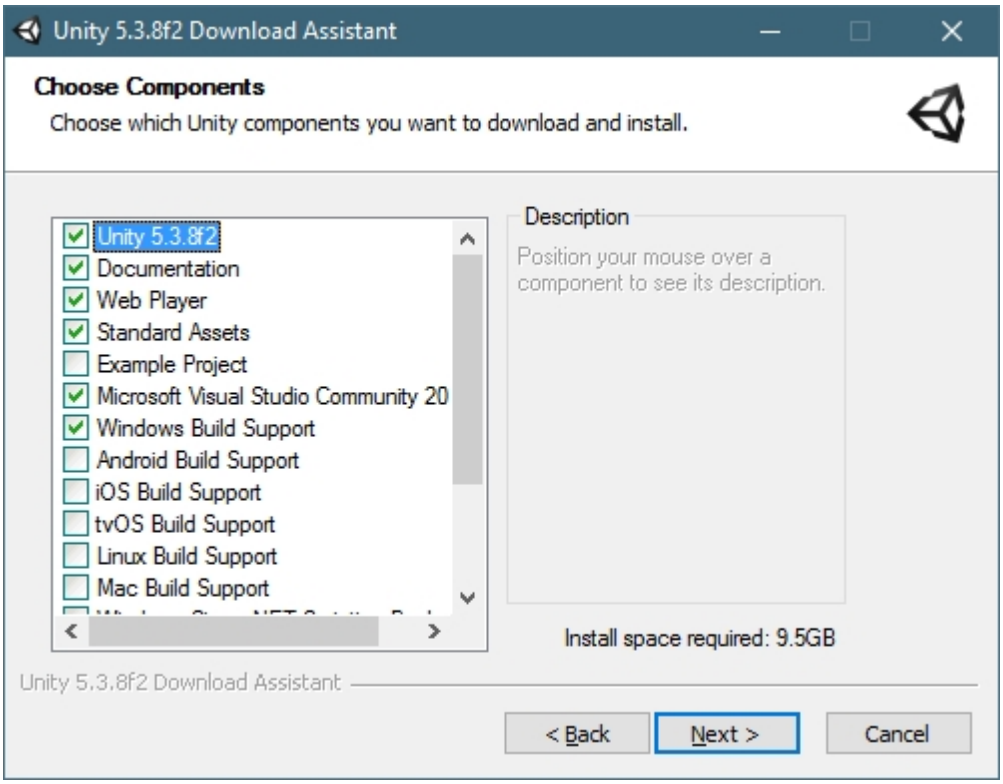
Once downloaded, double click on the UnityDownloadAssistant-5.3.8f2.exe



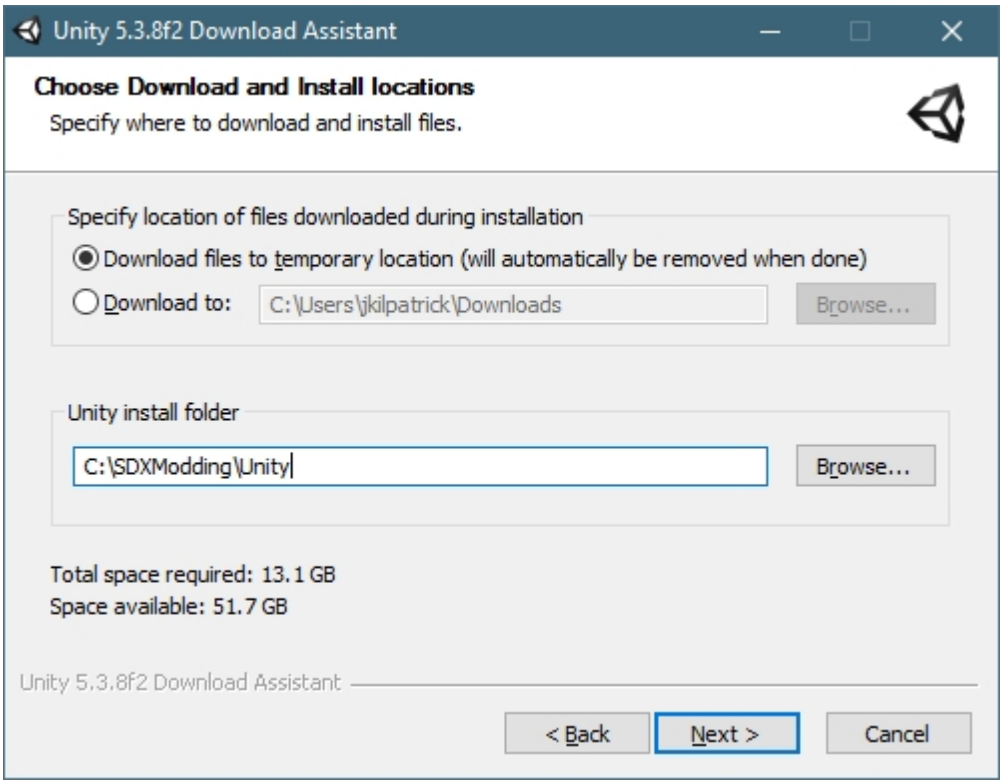
Accept the License Agreement



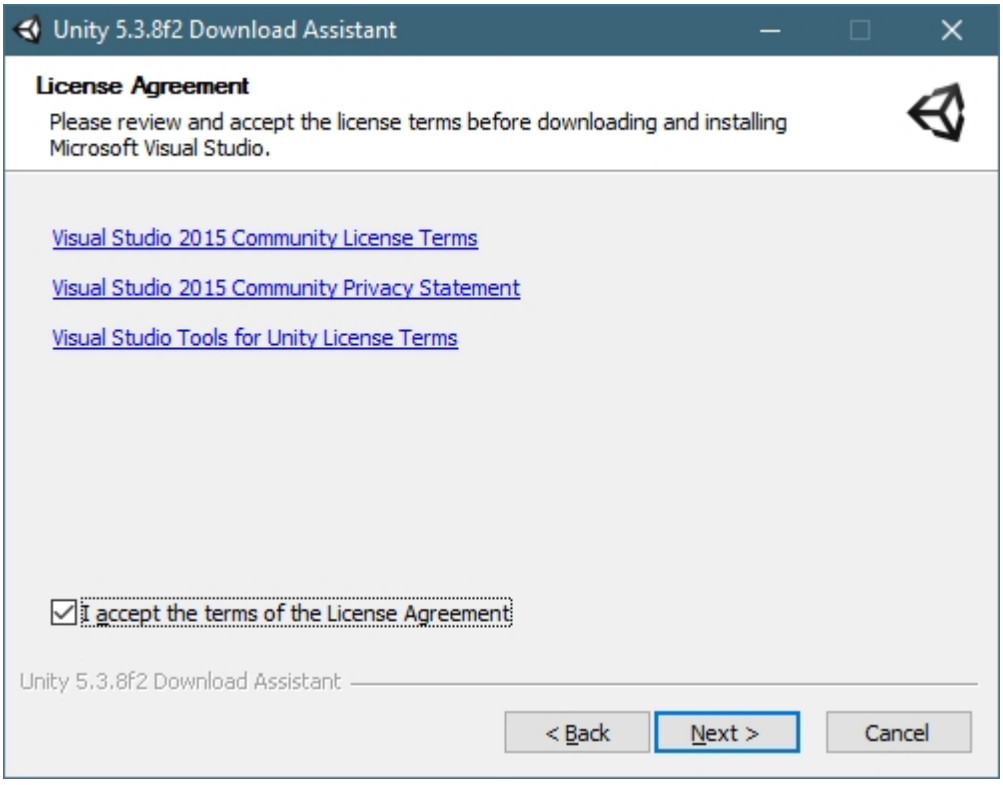
Accept Defaults:



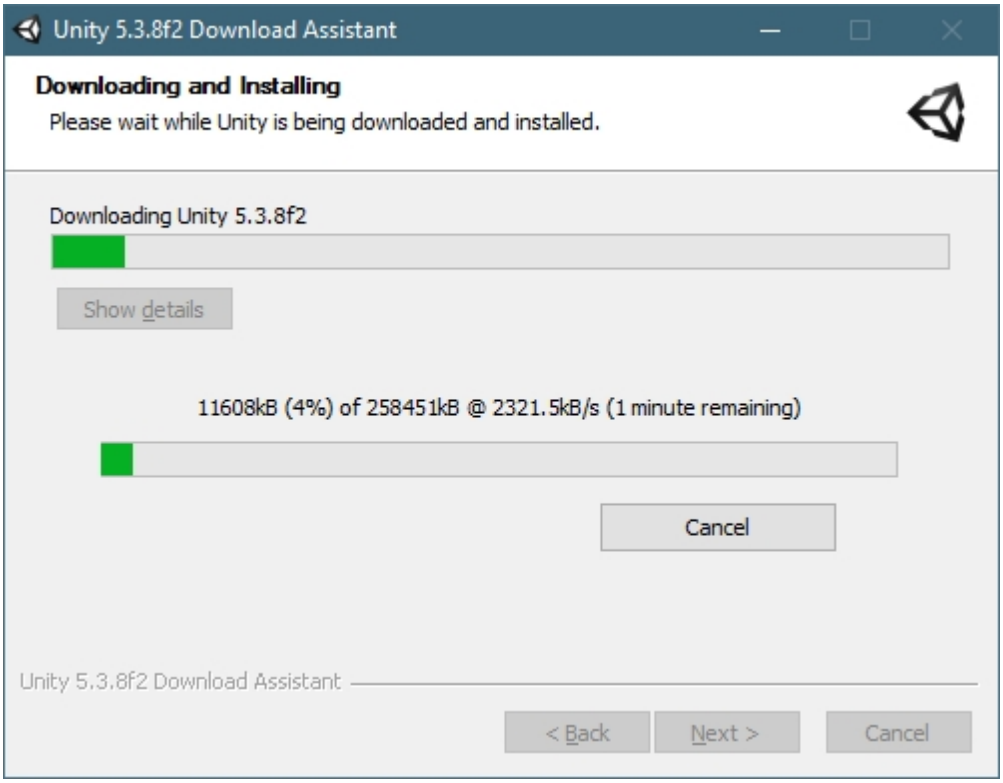
Select the Unity Install location, either as default, of under your C:\SDXModding\ folder.

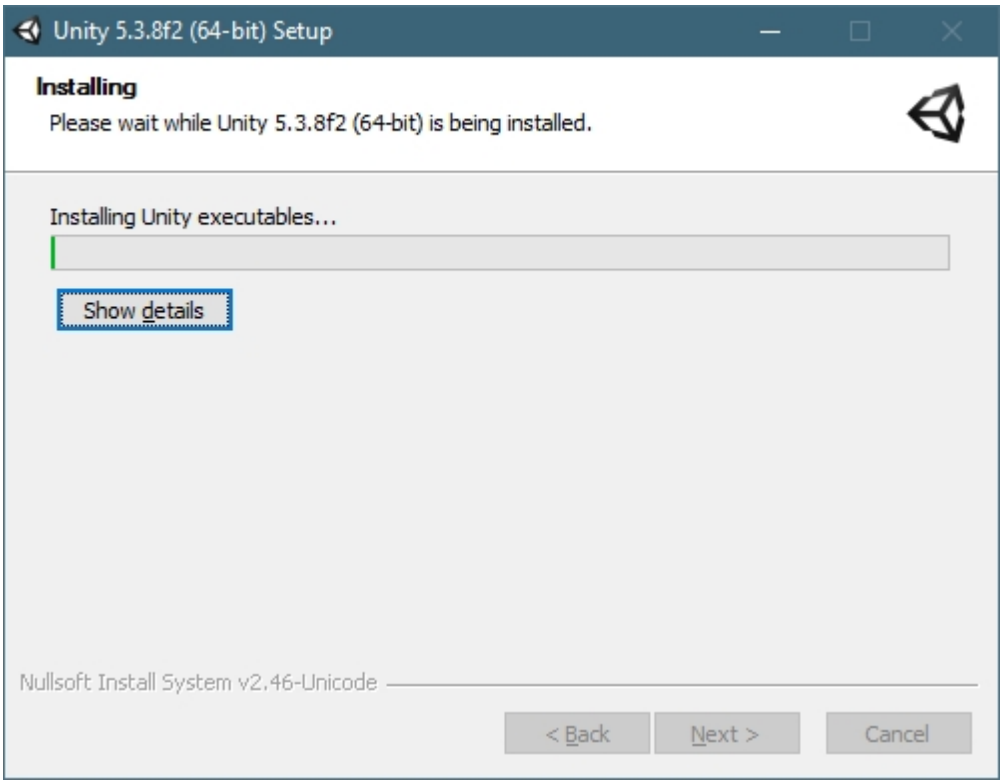


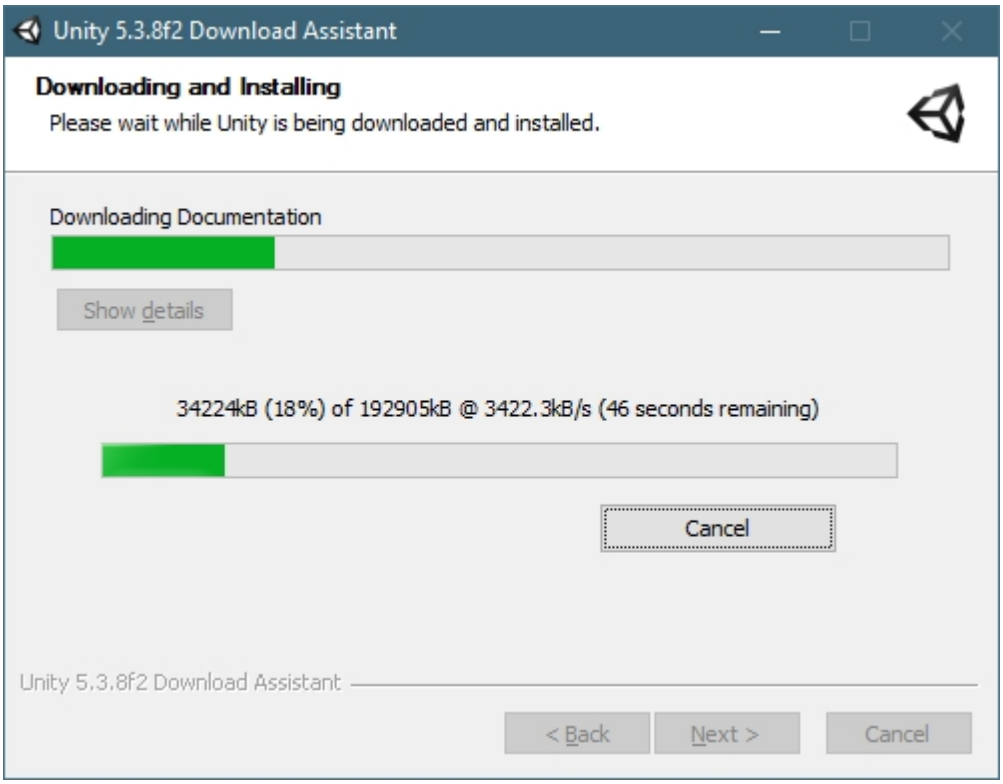
Accept the License Agreement by checking on the check box, once you have reviewed and accepted the licenses



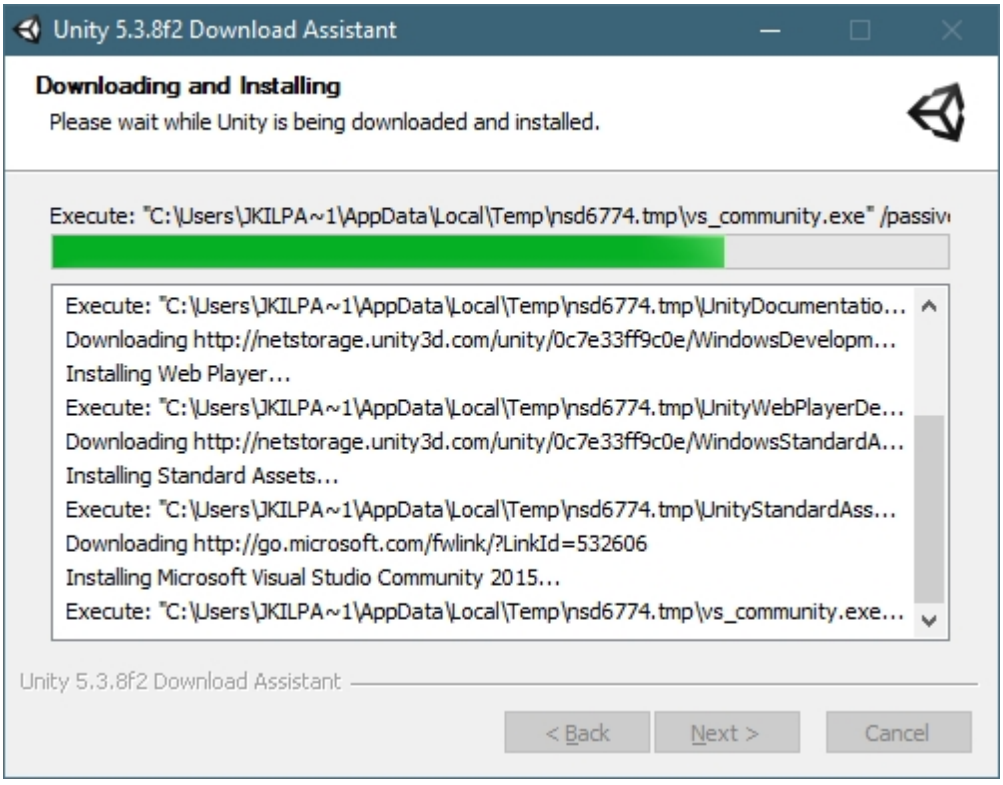
Wait for the automated download and install to be completed


Unity 5.3.8f2 Download Assistant
Downloading and Installing
Please wait while Unity is being downloaded and installed.
Downloading Unity 5.3.8f2
Show details
11608kB (4%) of 258451kB @ 2321.5kB/s (1 minute remaining)
Cancel
Unity 5.3.8f2 Download Assistant
< Back
Next >
Cancel

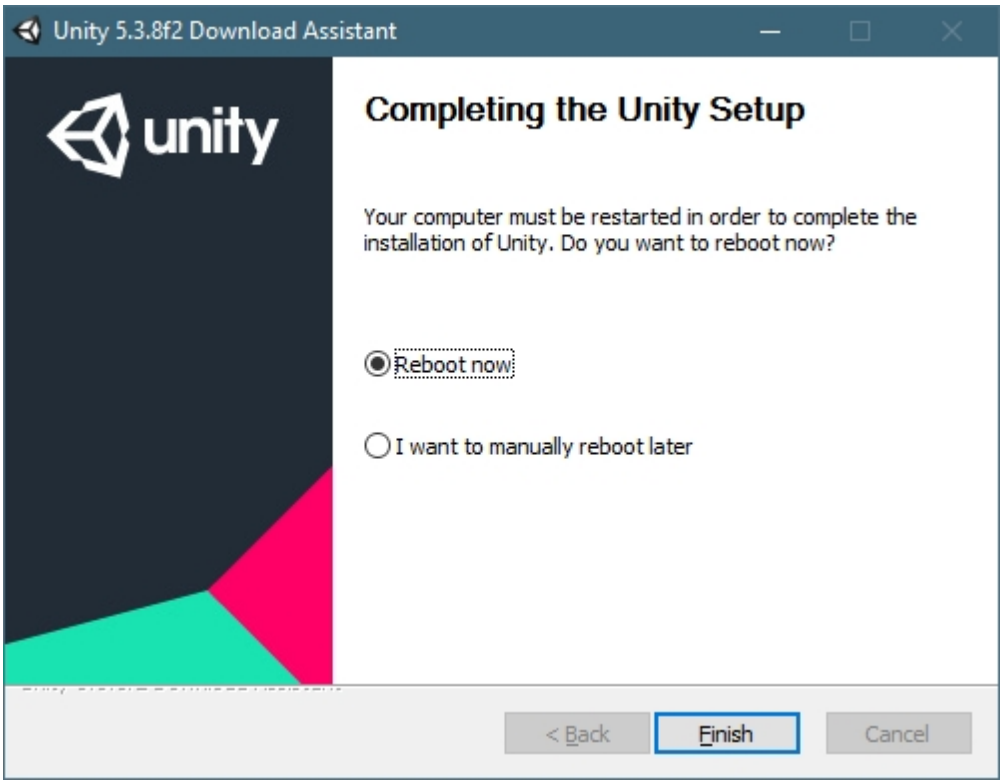

Unity 5.3.8f2 (64-bit) Setup
Installing
Please wait while Unity 5.3.8f2 (64-bit) is being installed.
Installing Unity executables...
Show details
Nullsoft Install System v2.46-Unicode
< Back
Next >
Cancel



During the installation process, other tools will be installed as needed, such as Visual Studio 2015 Community Edition.

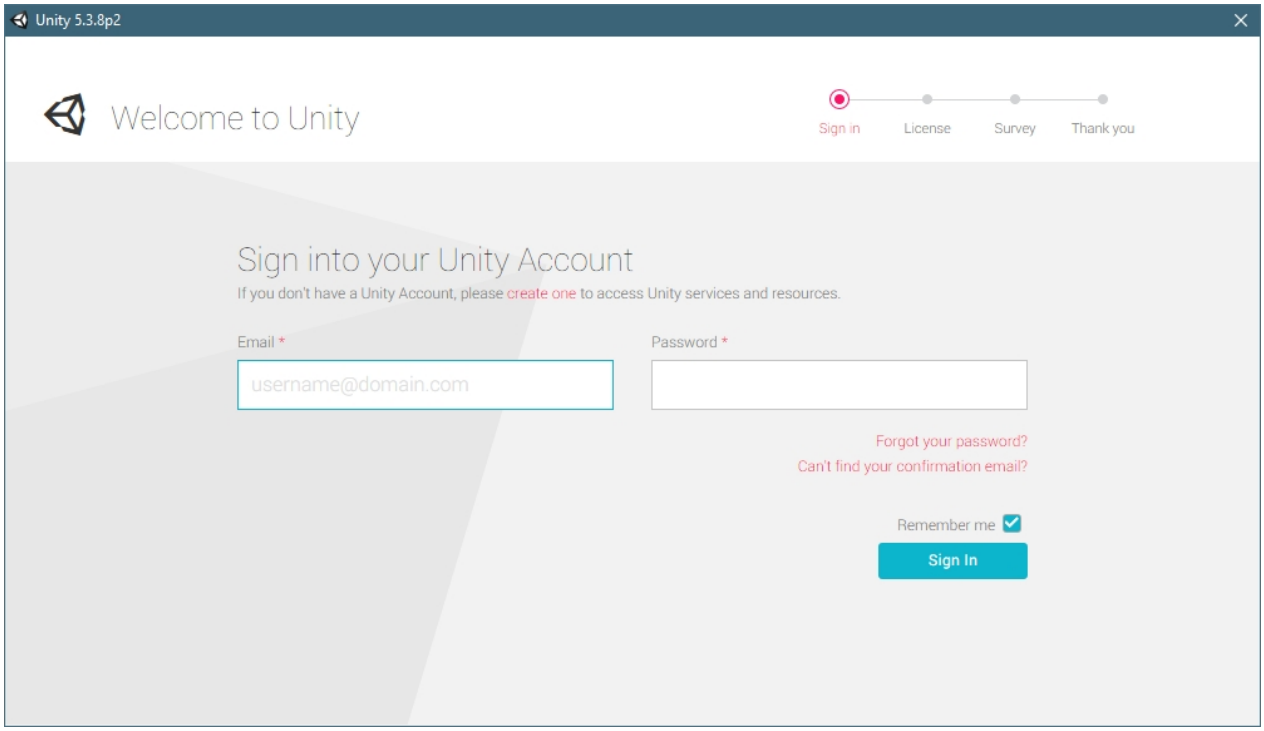


Reboot when done.

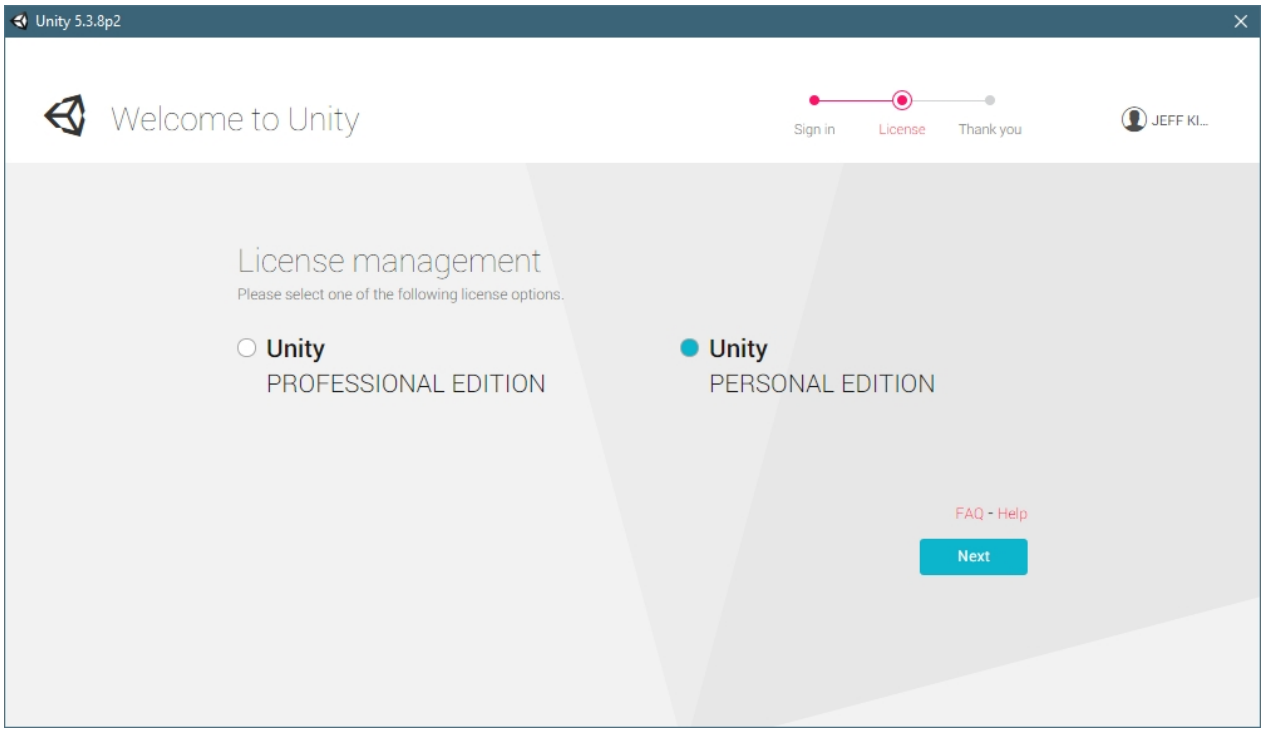


Starting Unity for the first time

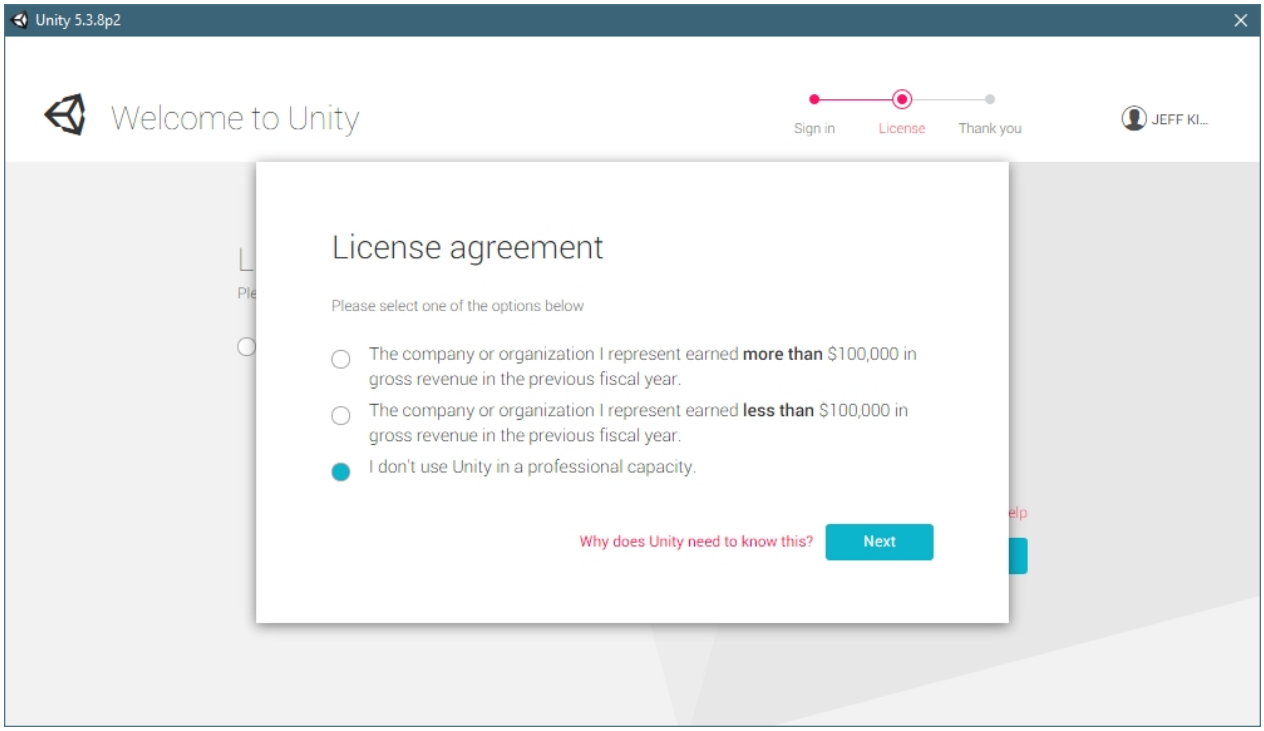
When Unity is started, it will ask you to either login, or create a new account. This will allow you to download assets from the Unity store.



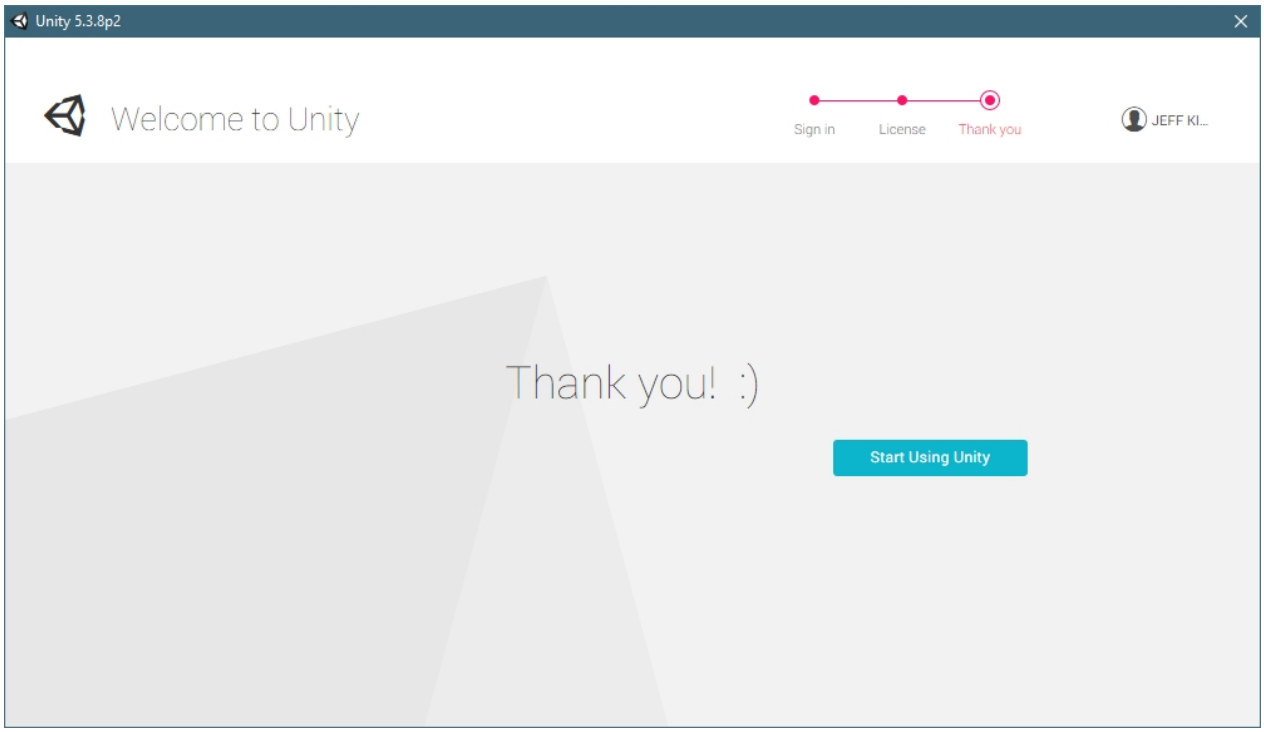
Unity offers two options, Professional Editior of Personal Edition. For this tutorial, we'll select Personal Edition.



Unity is a paid program once your organization makes over $100,000, and then you must buy a license. Unless you are doing exceptionally well in your modding, you'll want to select the "I don't use Unity in a professional capacity"



After a quick config, Unity is ready for you

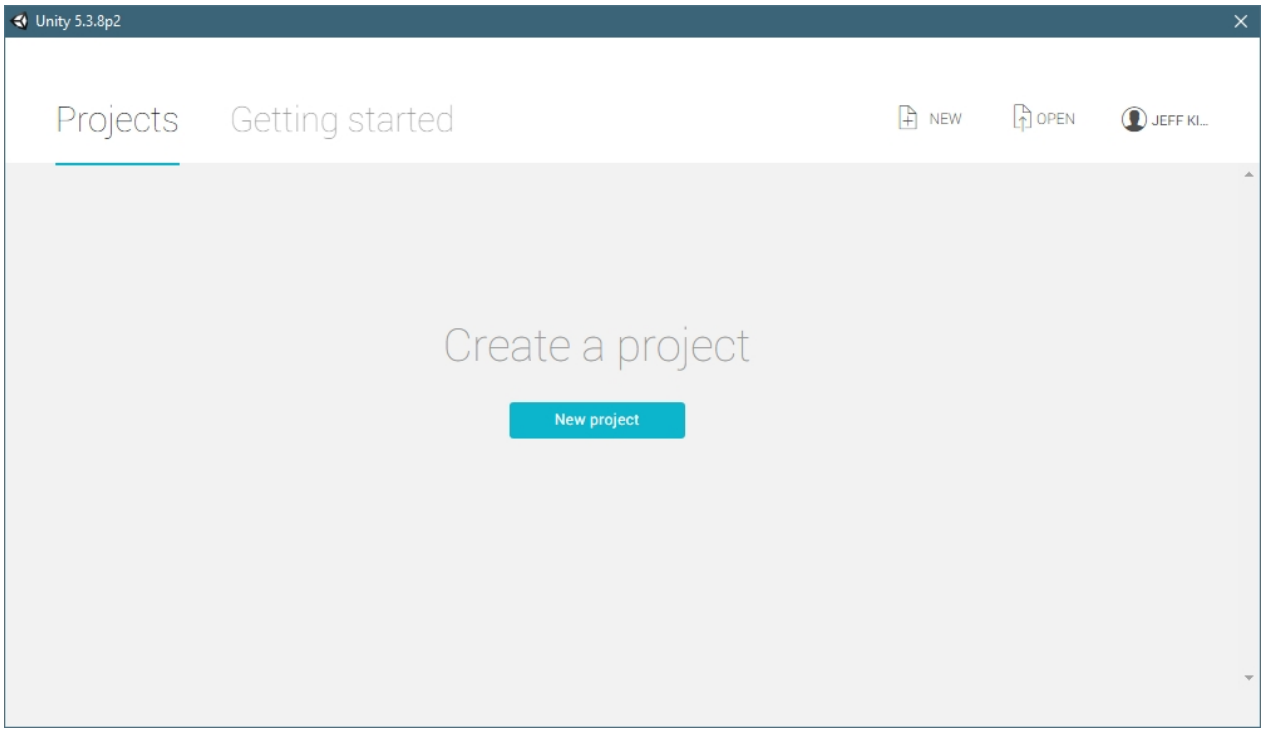


Creating a new Unity Project

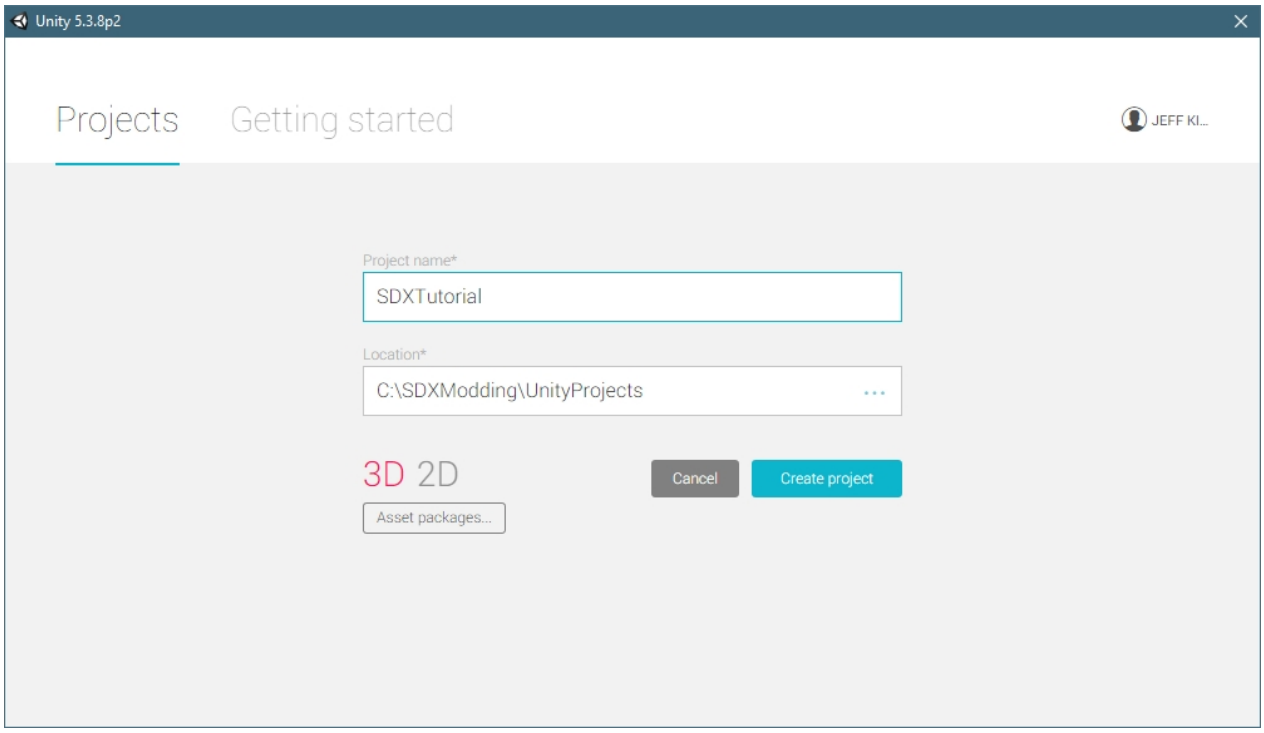
Xyth has created a series of YouTube videos guiding you through on how to make a Unity Project and exporting assets. Those videos supersede this part of the tutorial, however, they are left for review if so desired.

Creating and Exporting models From Unity for use in 7D2D

Let's create a New Unity Project

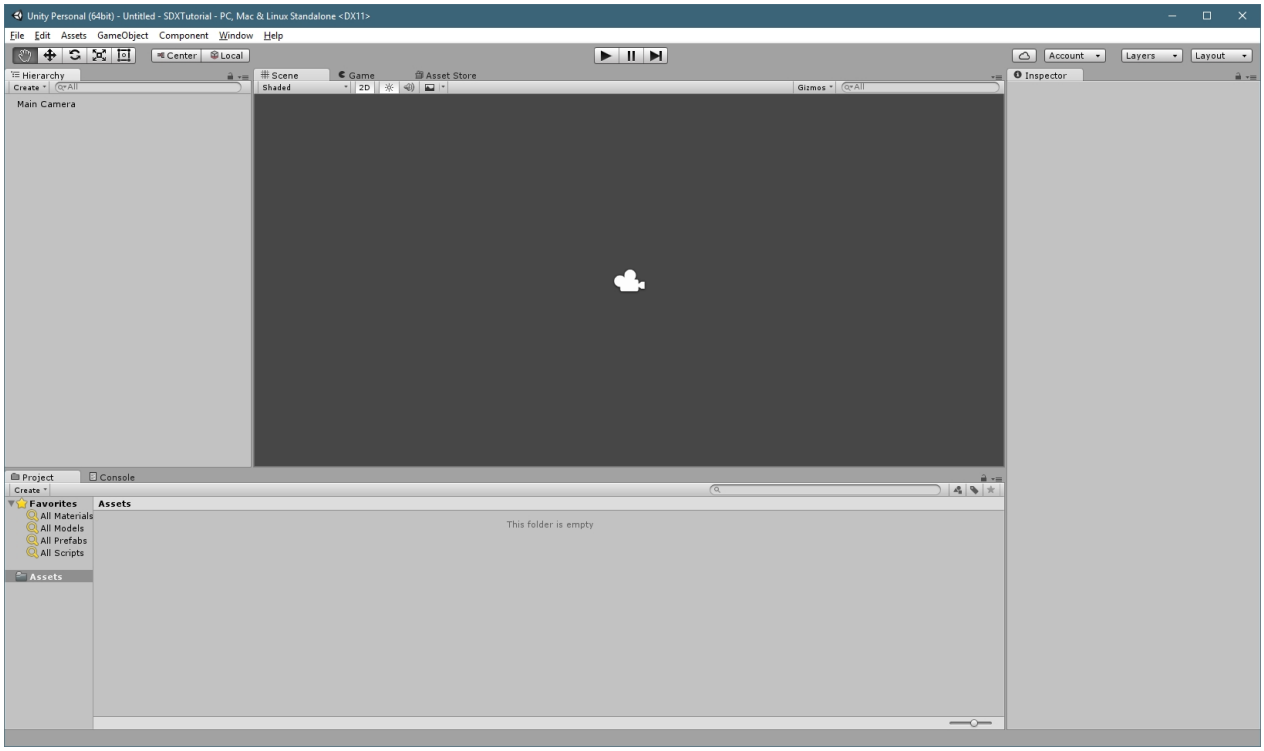


Give it a relevant name:

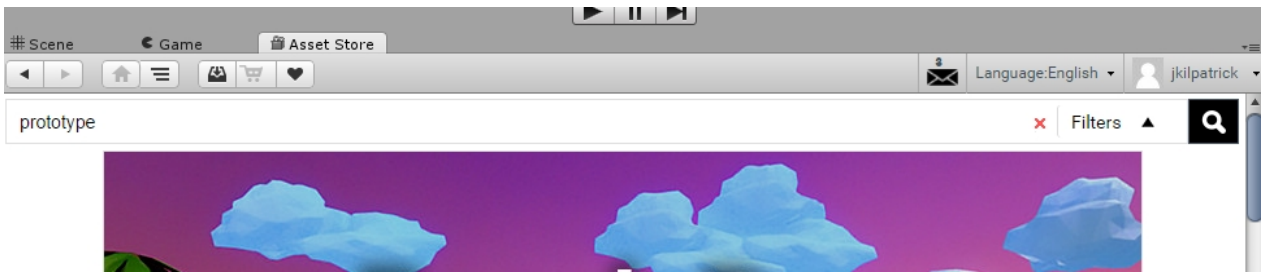


Adding A Demo Prototype Asset

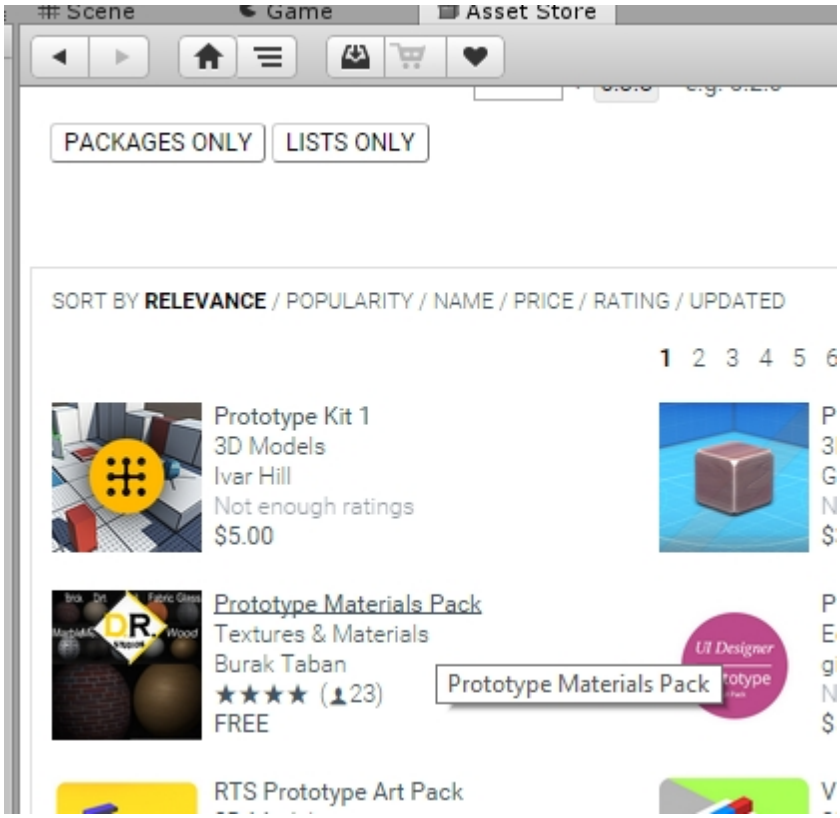
Click on the Asset Store link at the center of the screen, so that we can grab a sample Texture Package



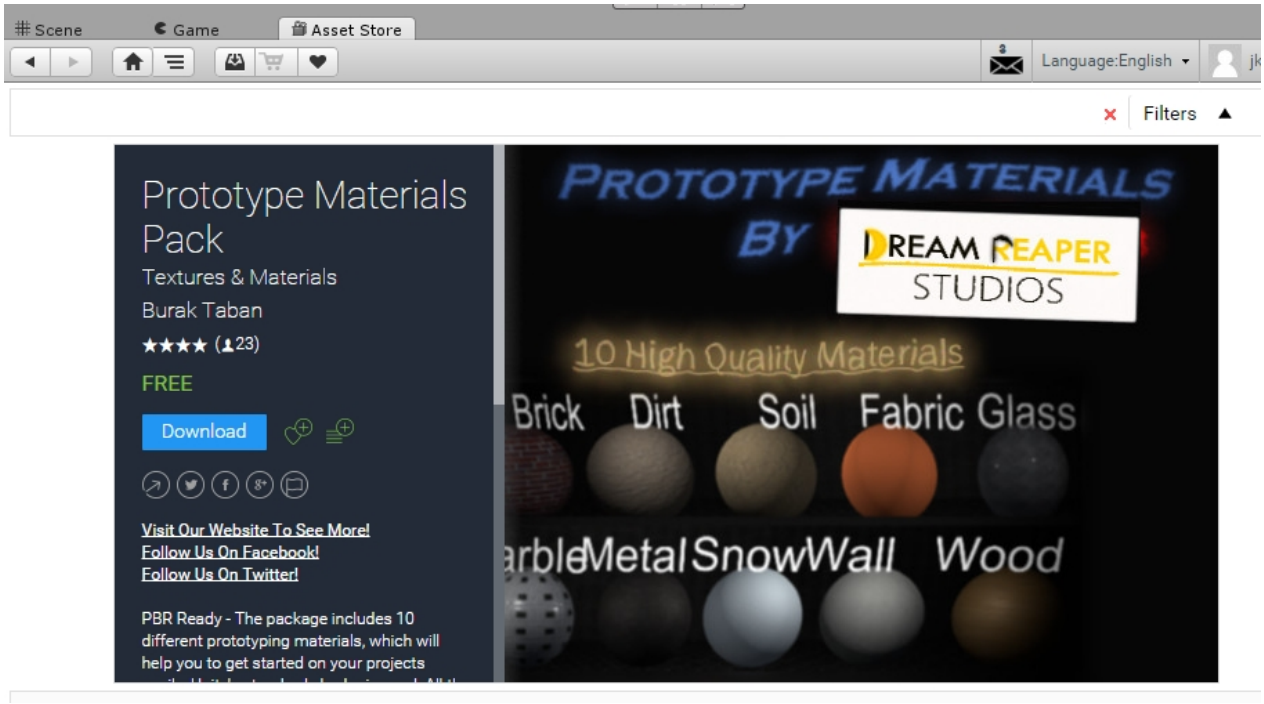
In the Search Window, type in prototype



Scroll down until you see "Prototype Materials pack"



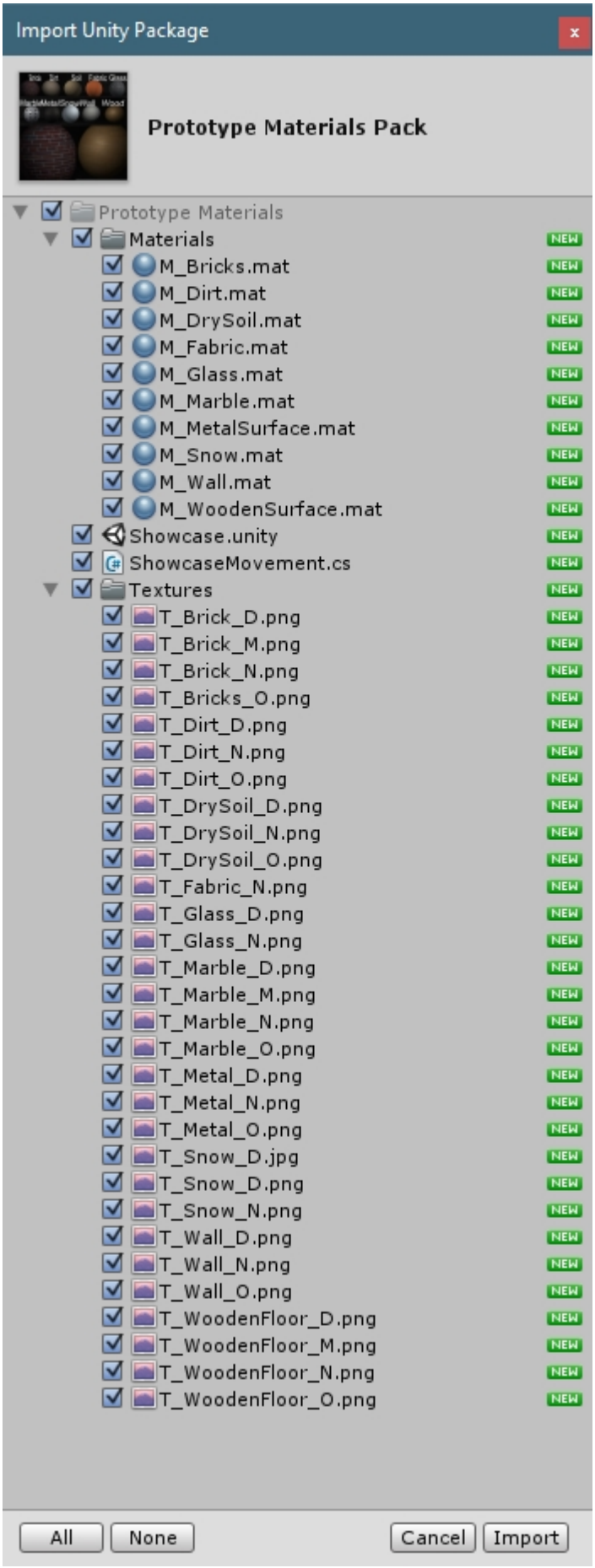
Click on it to load the Store Page for it



Click on Download for it to be added to your local Asset Area.

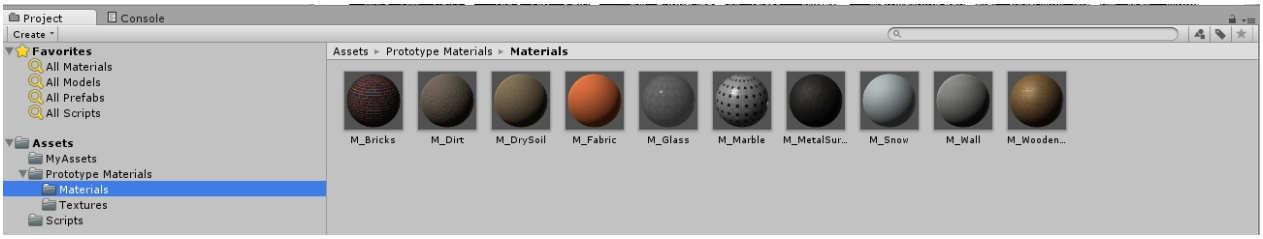
It will download the asset

Inside each Asset download, you can import some, or all of the available assets.



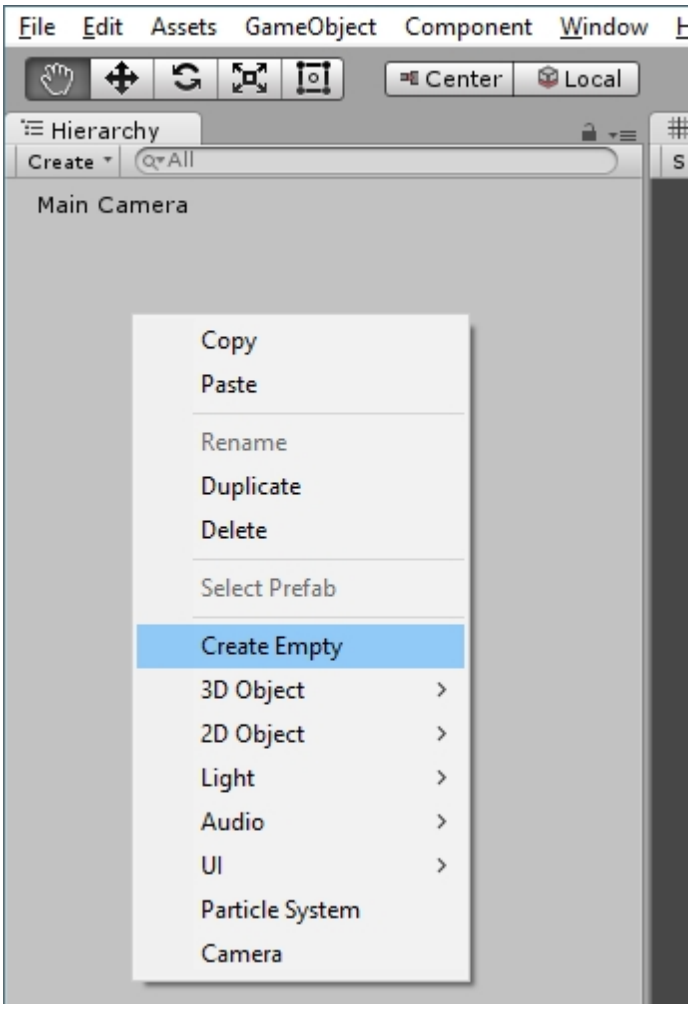
In this example, leave everything checked, and click on Import.

In the bottom window, under Assets, you'll see a new folder structure with the Prototype materials

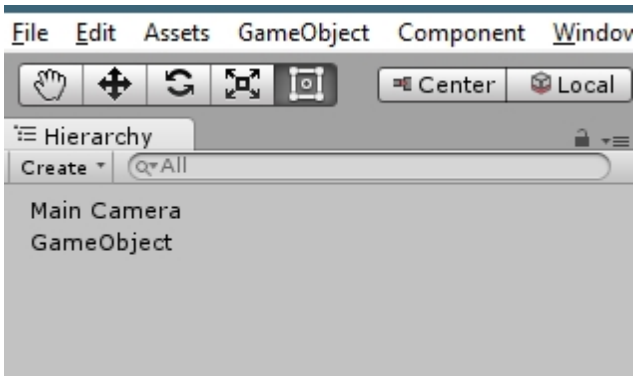


Creating a Sample Cube

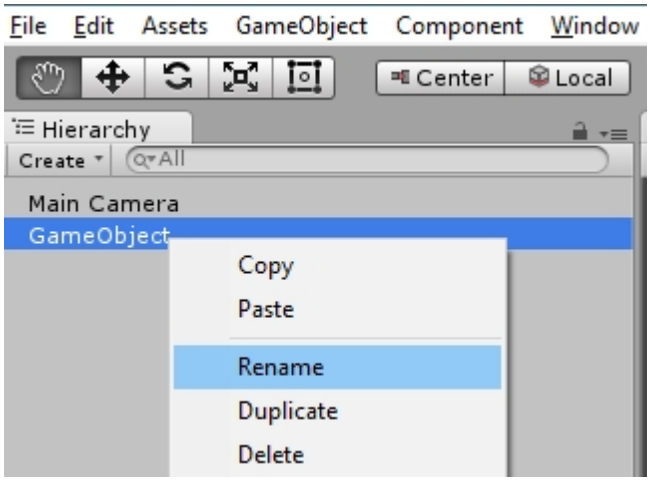
Right click under "Main Camera", and select "Create Empty"



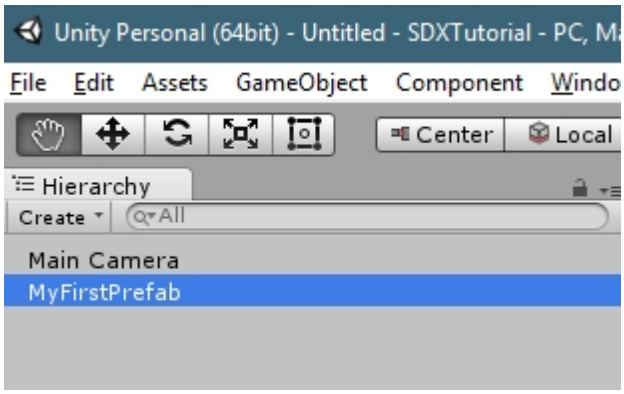
This creates a new "GameObject" item under the "Main Camera"



Right click on the newly created "GameObject", and select "Rename"

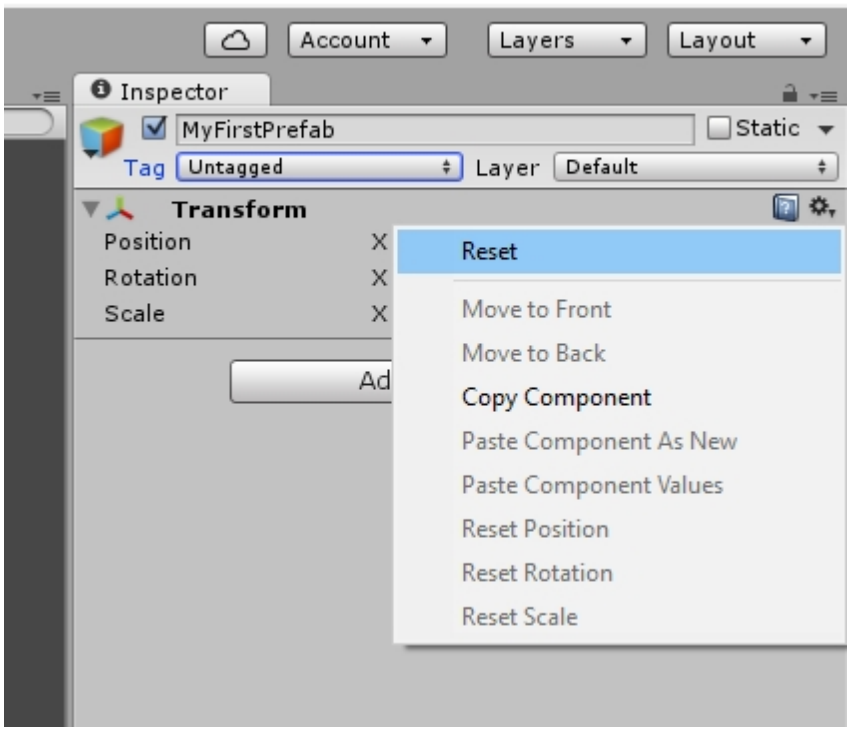


Rename it to "MyFirstPrefab"

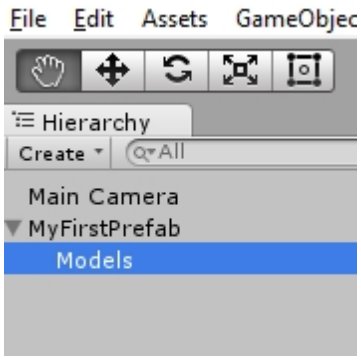


With "MyFirstPrefab" highlighted, we'll want to make sure that the Transform is reset to 0.

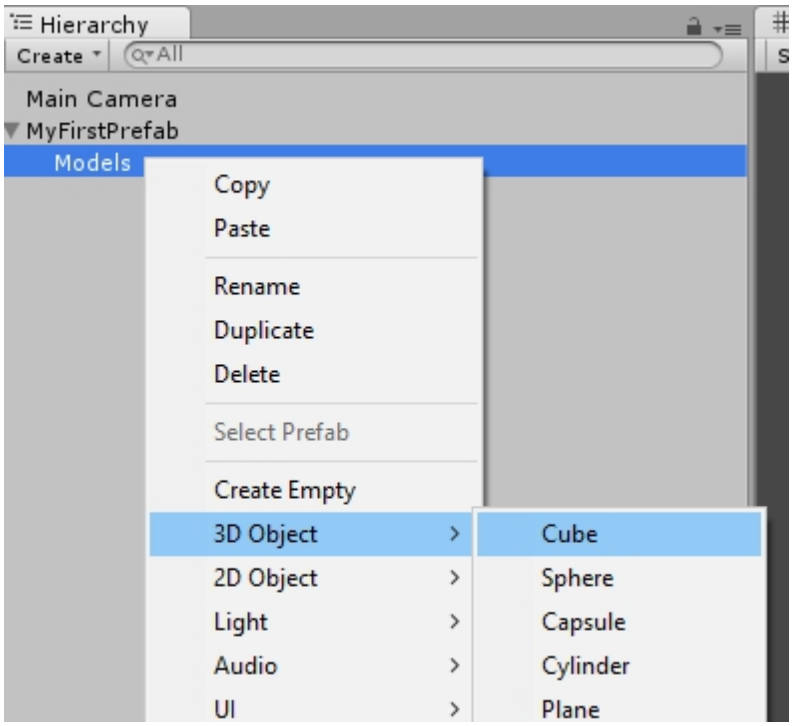
Click on the little Gear on the right hand side of the screen, and select "Reset"



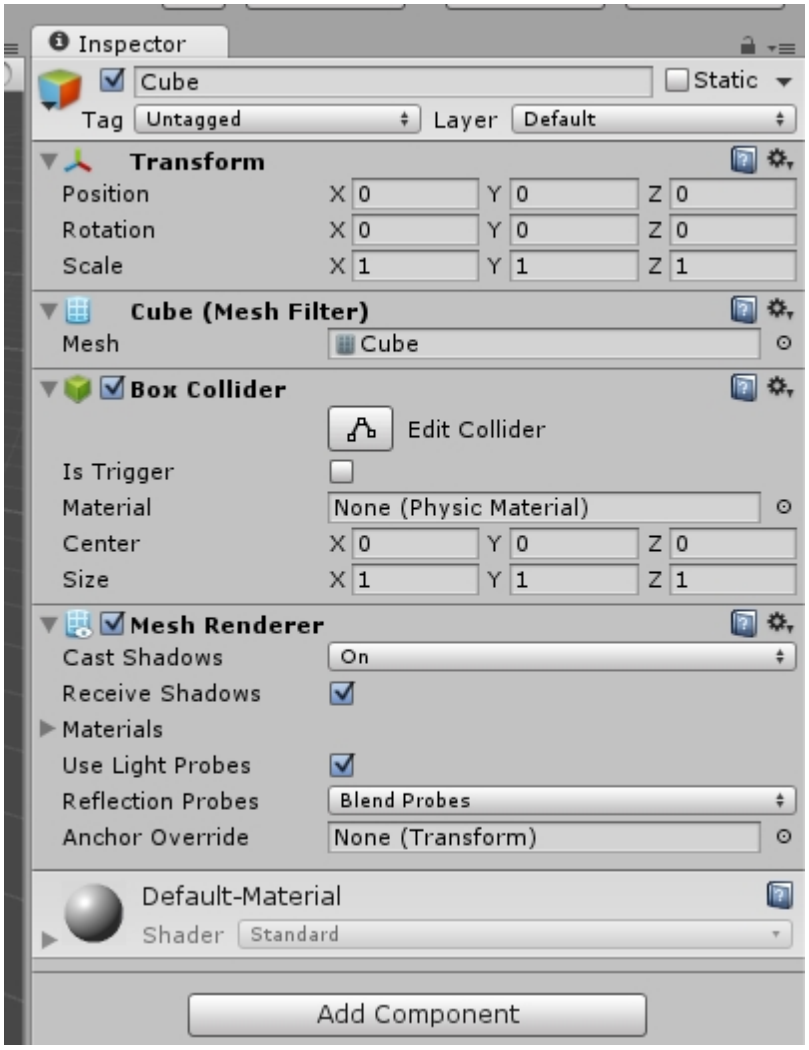
Right click on "MyFirstPrefab", and select "Create Empty". Rename the new "GameObject" to "Models"



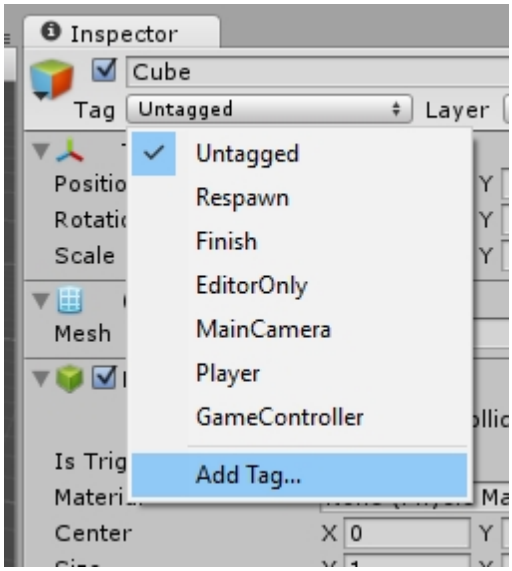
Right Click on "Models", and select "3D Object", then Cube



With the new Cube highlighted, look over in the Inspector window

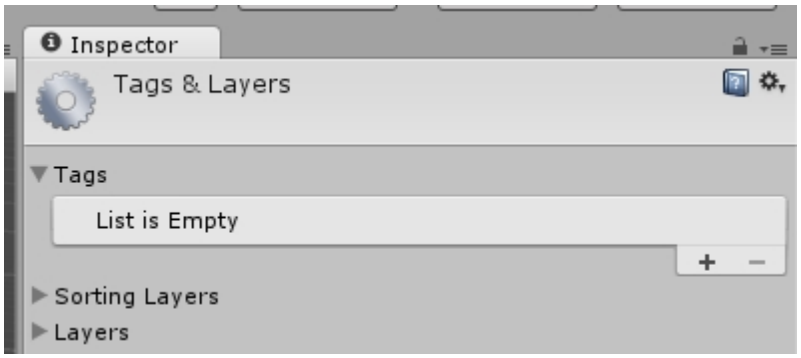


Click on the Tag drop down, that says "Untagged".



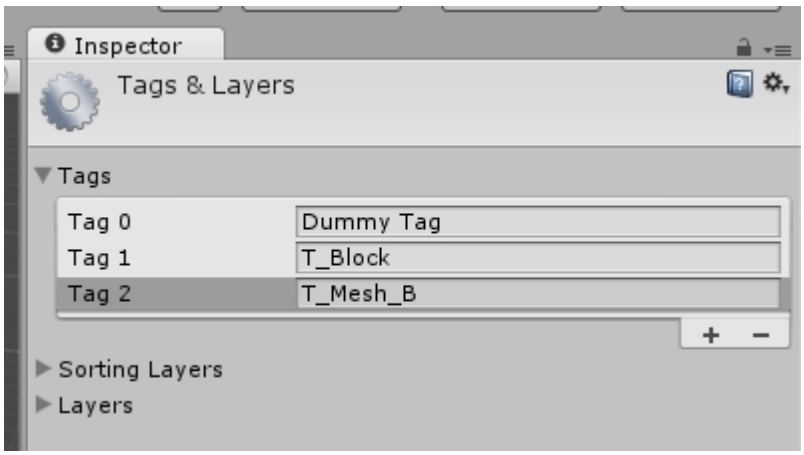
All Game Objects for 7 Days To Die that need meshes, are required to have a T_Mesh_B tag. If you see the screen above, without the T_Mesh_B, click on "Add Tag..."

You'll need to add these tags if they don't exist:

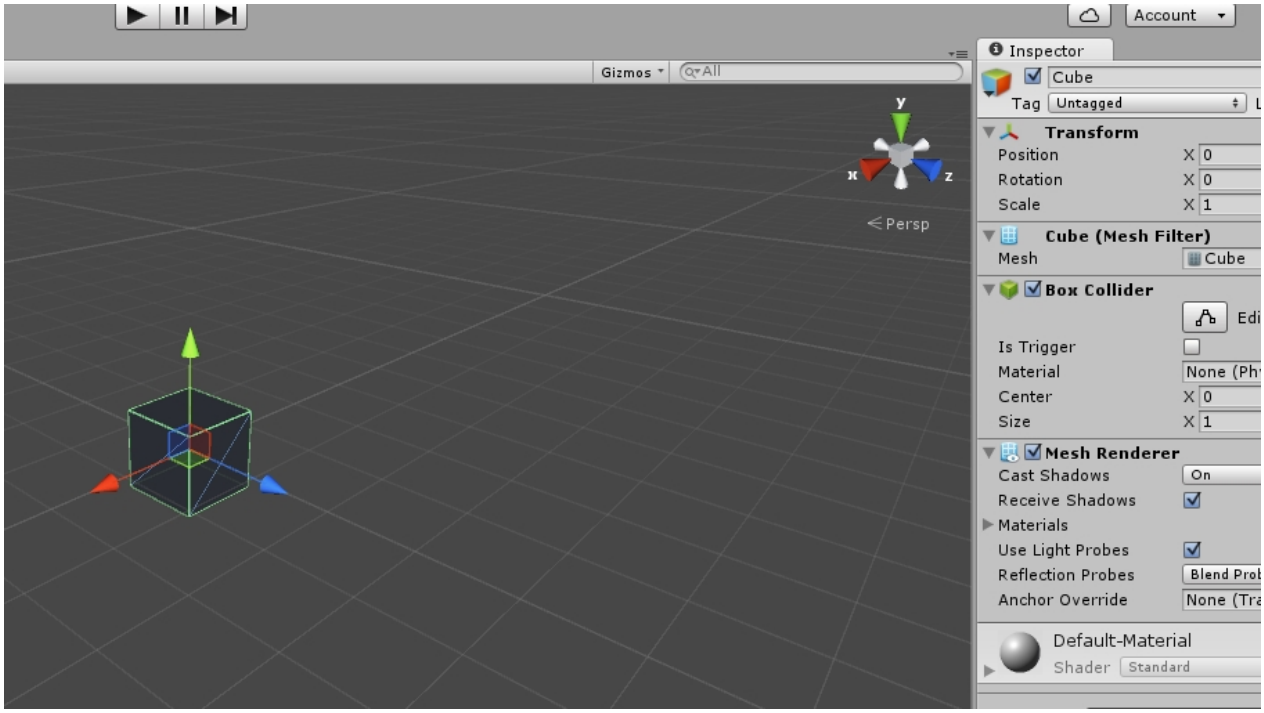


Click on the + button, and add two new tags, in this order:

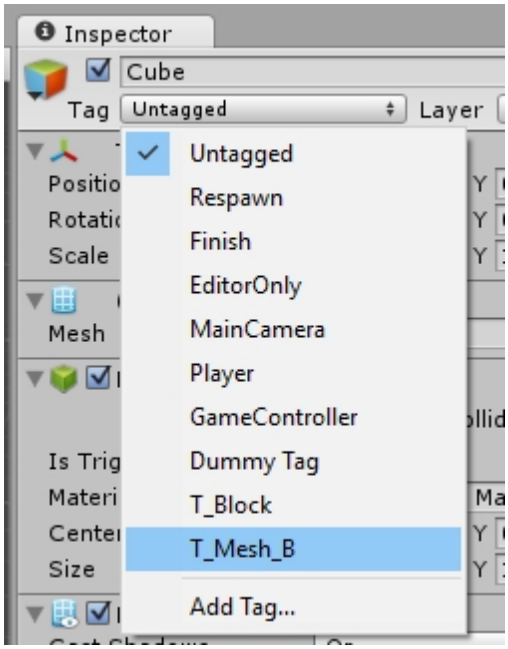
Dummy Tag
T_Block
T_Mesh_B



Once done, click on your Cube in the center of the screen to get the Inspector back

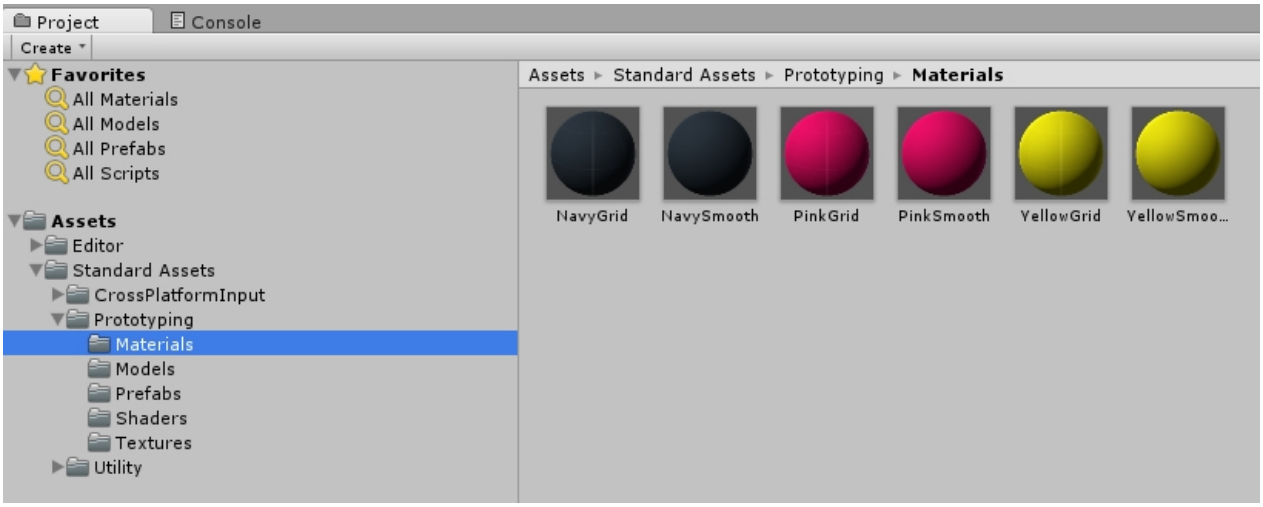


Then, click on the Tag drop down again, and select "T_Mesh_B".



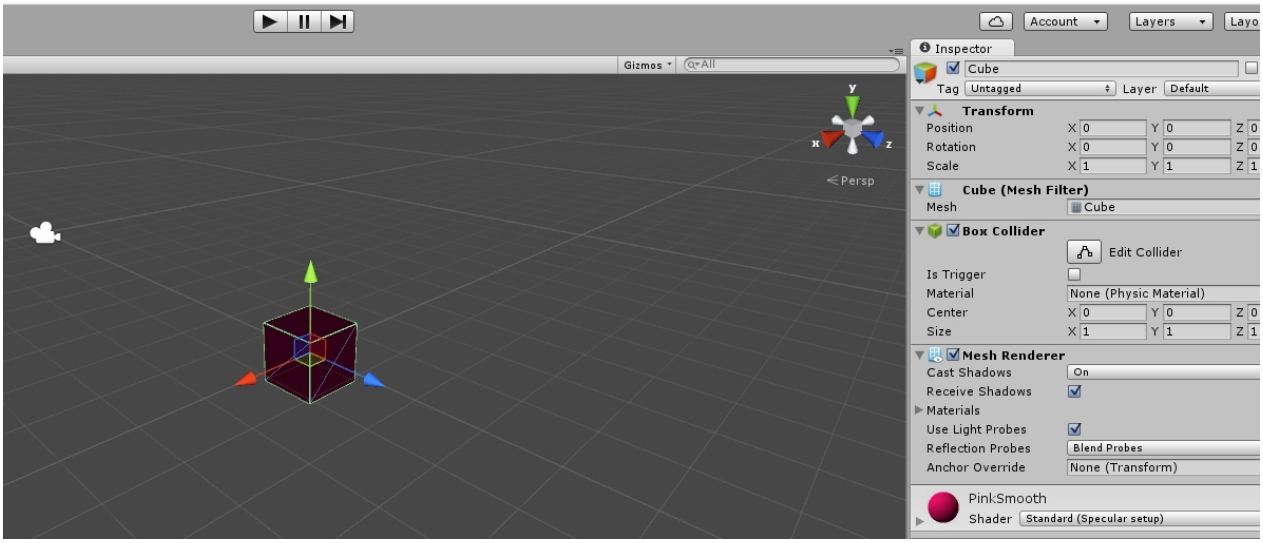
Click on the Cube on the Heirarchy.

Then, at the bottom of the screen in the "Project" Tab, expand the "Asset" list, then "Standard Asset", then "Prototype", and finally click on Materials



Drag the colour of which material you want to use. In this example, we are going to use PinkSmooth, as it'll stand out.

Click and hold on the PinkSmooth, and drag unto your Cube game object.



Notice that the colour is now PinkSmooth.

Now, under the Heirachy section, click and hold on "MyFirstPrefab", and drag down to the My assets folder.

GitHub Quick Review

Referenced Article

For us, in the 7 Days To Die Community, Github is the preferred way to host mods and code, that allows free storage, persistent download links, and a versioning system, so you can see how your mod, or your favorite mod, has evolved over time.

This section will show you how to install and set up GitHub Desktop. We've listed a few words you'll see in this section

Terminology:

Repository: A repository is the most basic element of GitHub. They're easiest to imagine as a project's folder. A repository contains all of the project files (including documentation), and stores each file's revision history. Repositories can have multiple collaborators and can be either public or private.

This allows multipler modders to work together to update the mod, and share it with others.

Clone: A clone is a copy of a repository that lives on your computer instead of on a website's server somewhere, or the act of making that copy. With your clone you can edit the files in your preferred editor and use Git to keep track of your changes without having to be online. It is, however, connected to the remote version so that changes can be synced between the two. You can push your local changes to the remote to keep them synced when you're online.

A clone makes a local copy so you can download and make changes to it.

Commit: A commit, or "revision", is an individual change to a file (or set of files). It's like when you *save* a file, except with Git, every time you save it creates a unique ID (a.k.a. the "SHA" or "hash") that allows you to keep record of what changes were made when and by who. Commits usually contain a commit message which is a brief description of what changes were made.

A commit is just update the files online, for a release or when you've finished making changes you want to share.

Fork: A fork is a personal copy of another user's repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original. Forks remain attached to the original, allowing you to submit a pull request to the original's author to update with your changes. You can also keep your fork up to date by pulling in updates from the original.

A fork lets you make changes, or fixes, and then you can send them back to the original author for them to add them.

Organizations: Organizations are shared accounts where businesses and open-source projects can collaborate across many projects at once. Owners and administrators can manage member access to the organization's data and projects with sophisticated security and administrative features.

You can use organizations that separate your user from your project, while still giving you contorl

The “Git” in GitHub

To understand GitHub, you must first have an understanding of Git. Git is an open-source version control system that was started by Linus Trovalds – the same person who created Linux. Git is similar to other version control systems – Subversion, CVS, and Mercurial to name a few.

Version control systems

So, Git is a “version control system,” what’s that mean? When developers are creating something (an application, for example), they are making constant changes to the code and releasing new versions, up to and after the first official (non-beta) release.

Version control systems keep these revisions straight, and store the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.

Similarly, people who have nothing to do with the development of a project can still download the files and use them. Most Linux users should be familiar with this process, as using Git, Subversion, or some other similar method is pretty common for downloading needed files, especially in preparation for compiling a program from source code (a rather common practice for Linux geeks).

In case you are wondering why Git is the preferred version control system of most developers, it has multiple advantages over the other systems available, including a more efficient way to store file changes and ensuring file integrity. If you’re interested in knowing the details, check out this page to read a thorough explanation on how Git works.

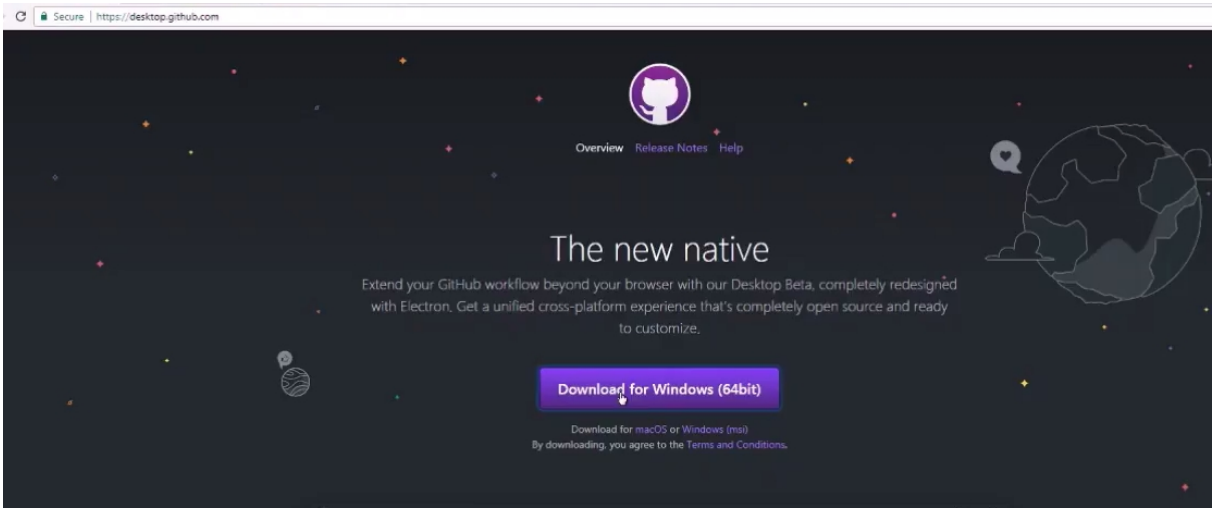
Installing and Configuring Github

GitHub Desktop

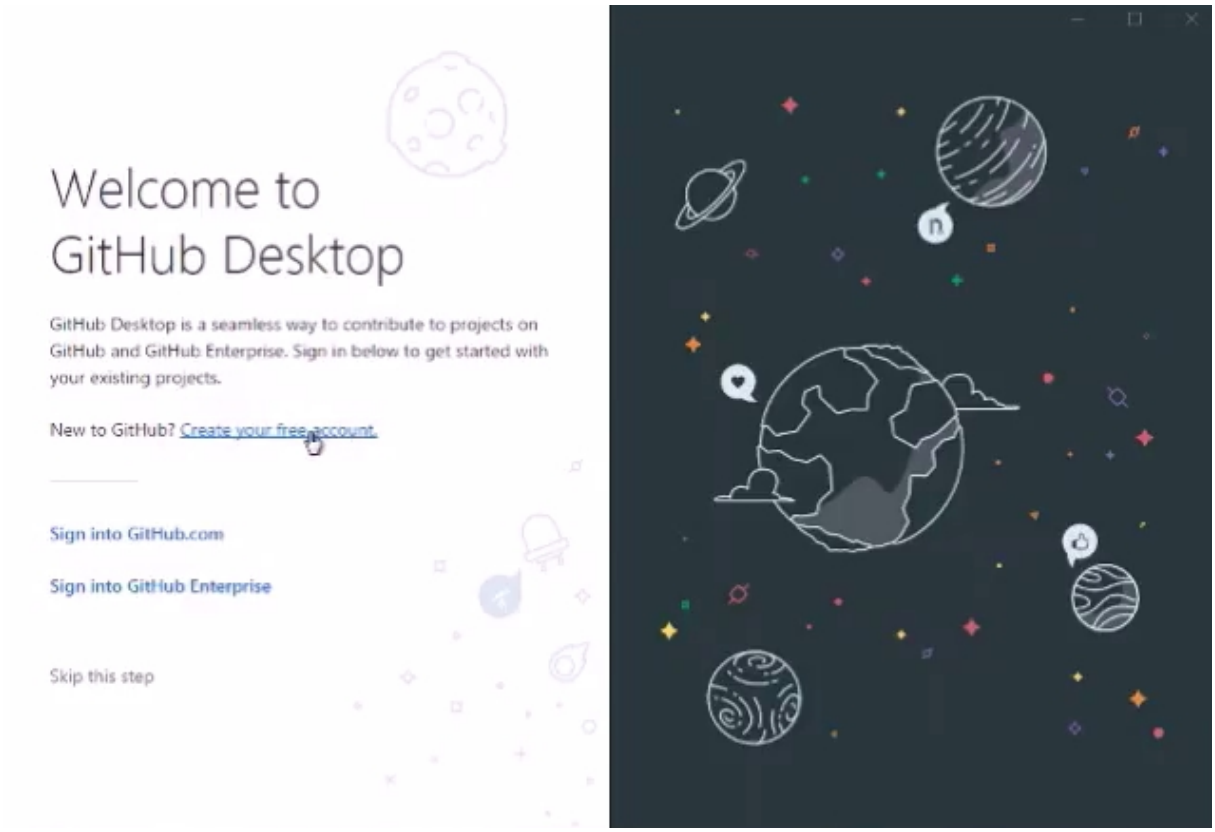
GitHub Desktop is an easy to use tool to manage code. You can download it here.

GitHub is a free hosting site, useful for making small web sites, storing code, and distributing files to other people.

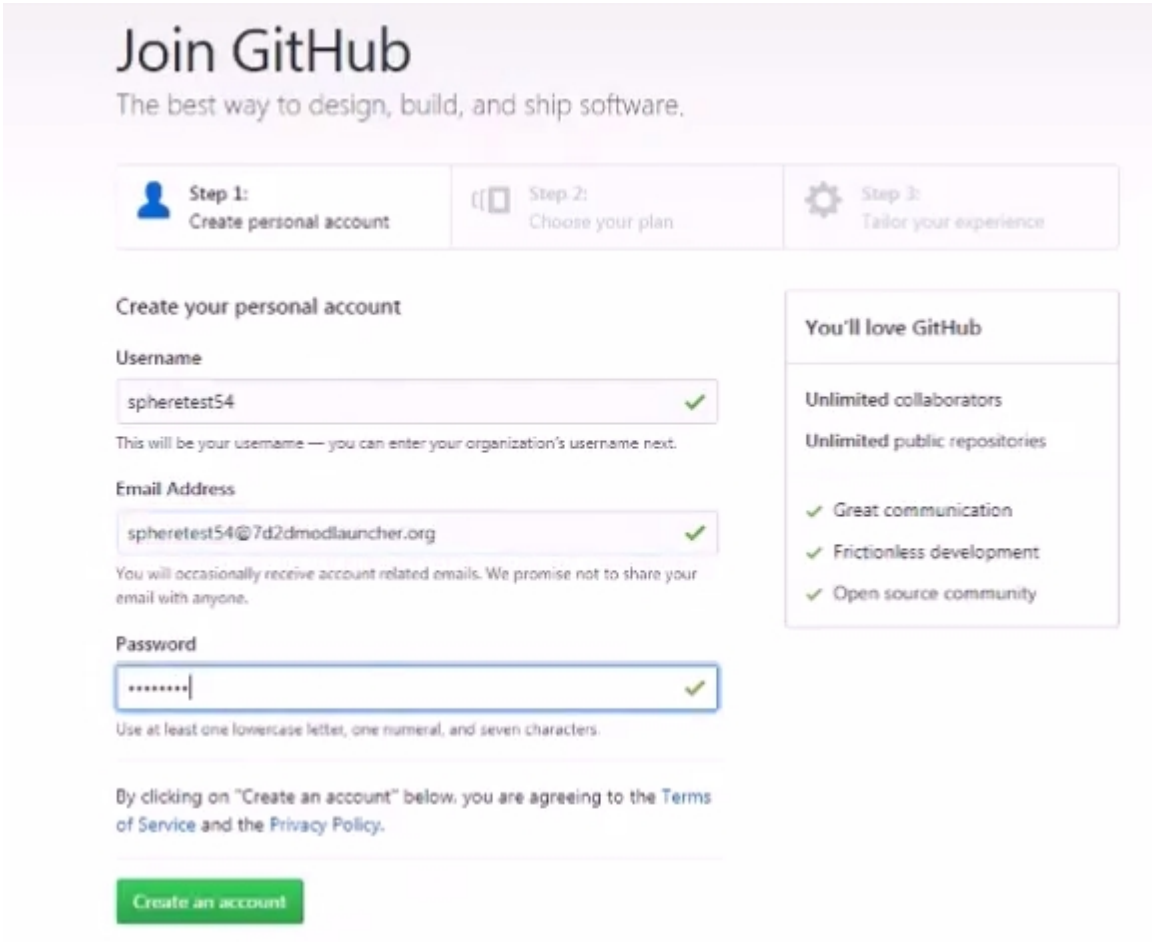
To get started, install GitHub Desktop



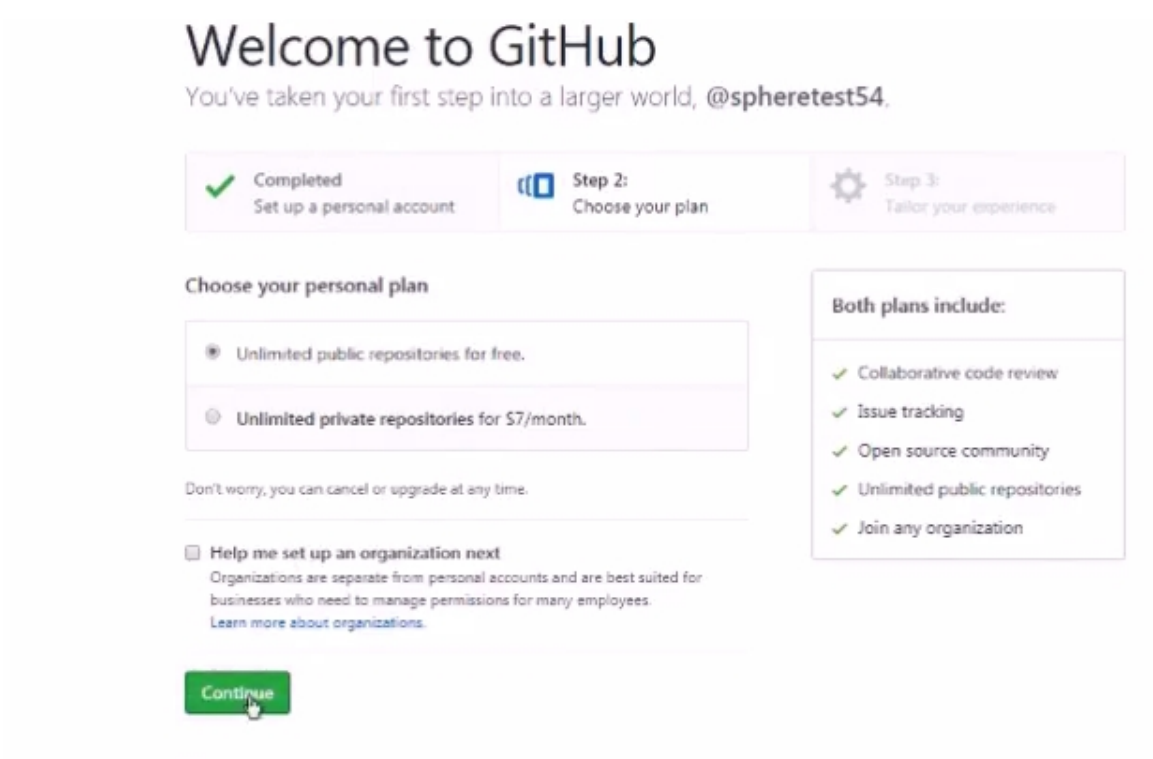
Click on "Create your free account". This will open a new browser window or tab:



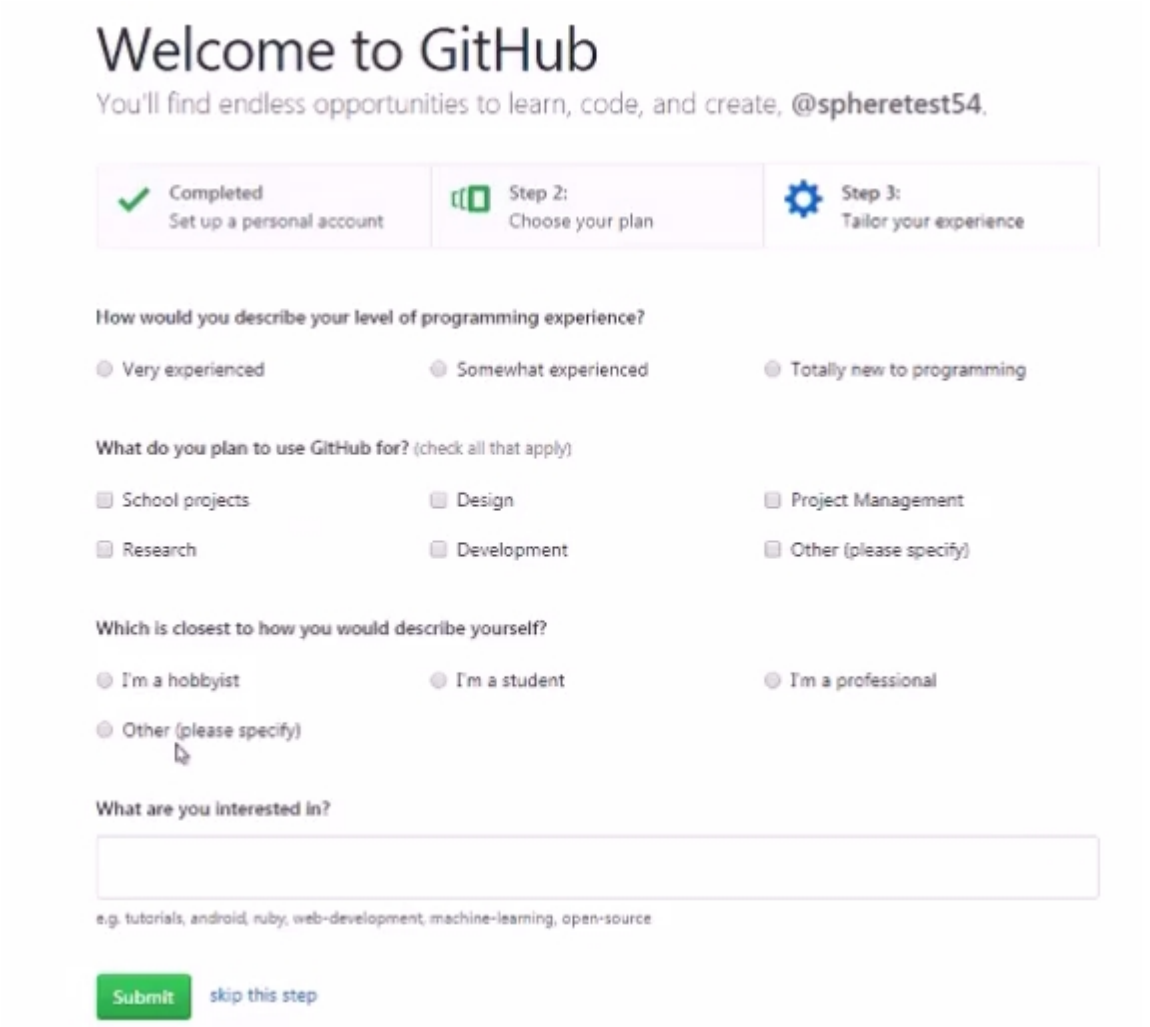
Enter in a user name, your email address (you'll get a verify email link), and your password.



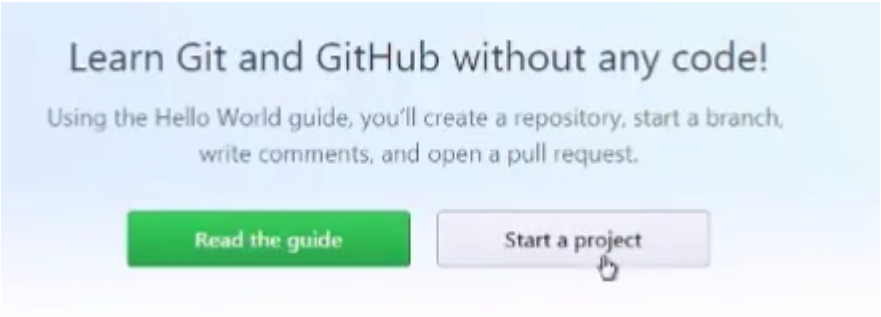
Accept the default of "Unlimited public repositories for free"



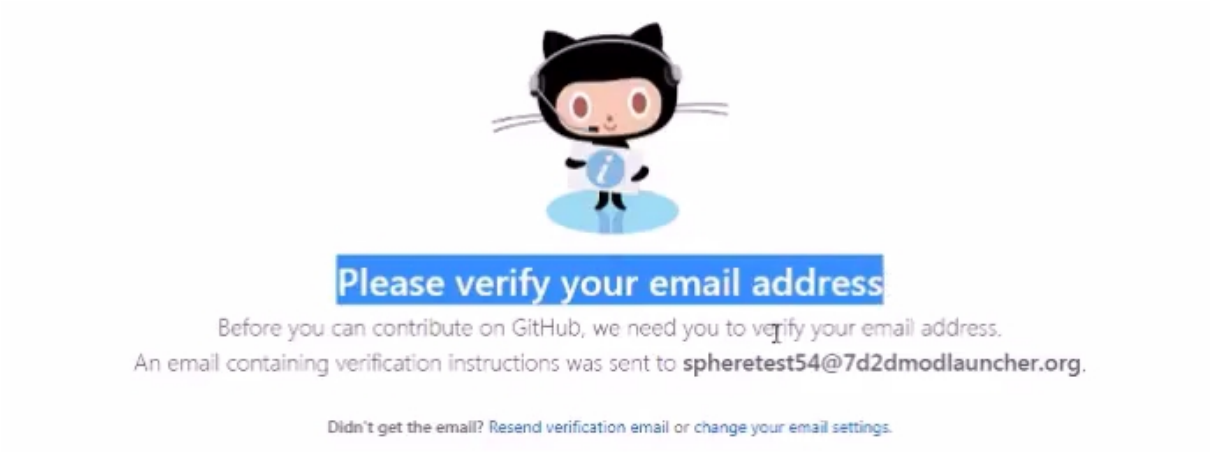
Optionally, you may fill out their small questionnaire



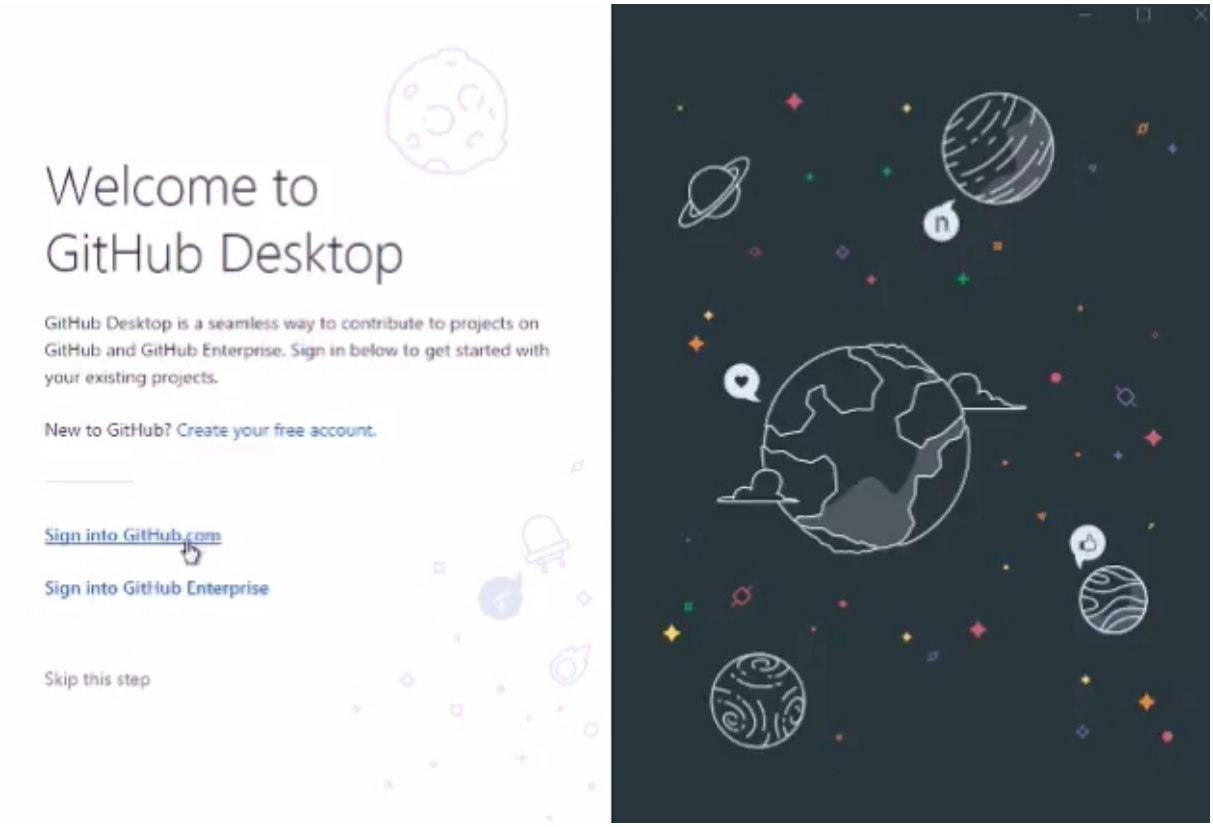
Create a New Project



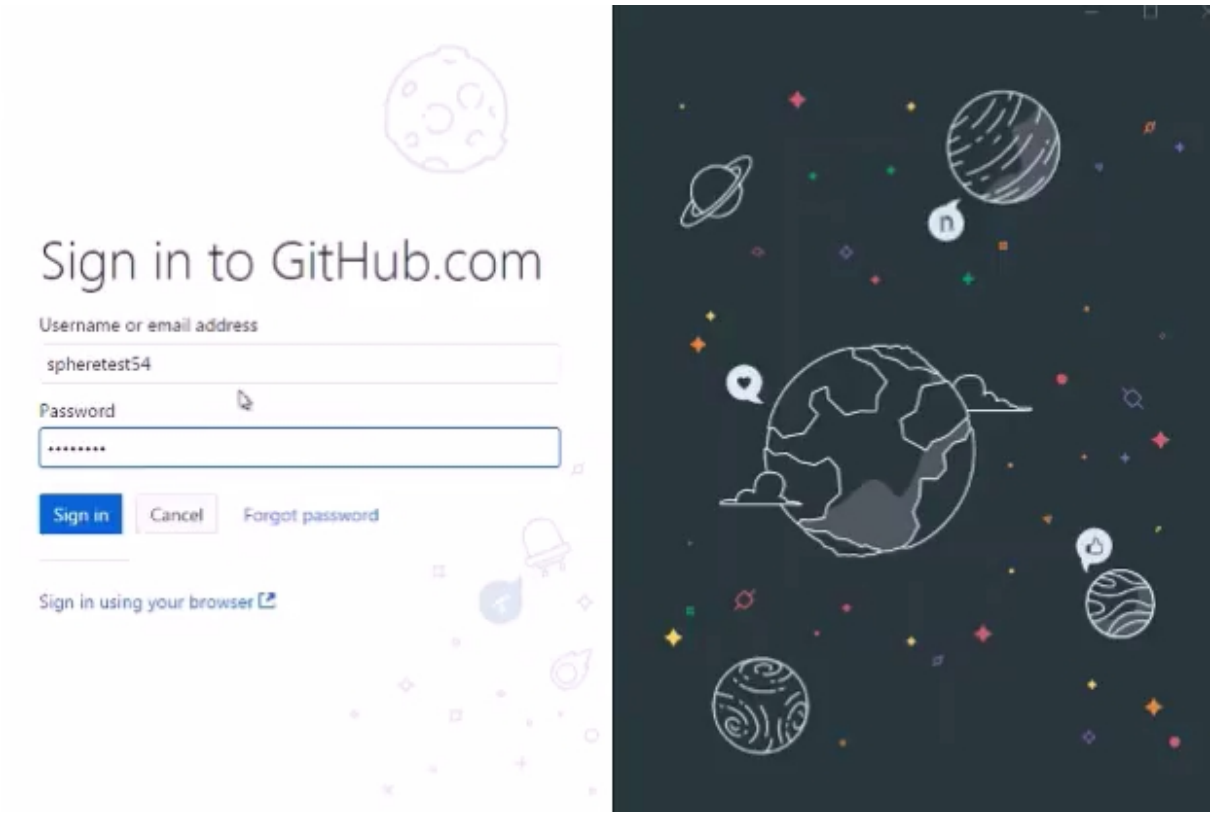
Check your email for the Verification link



After verifying your email address, go back to the Github Desktop, and click on "Sign into Github.com"

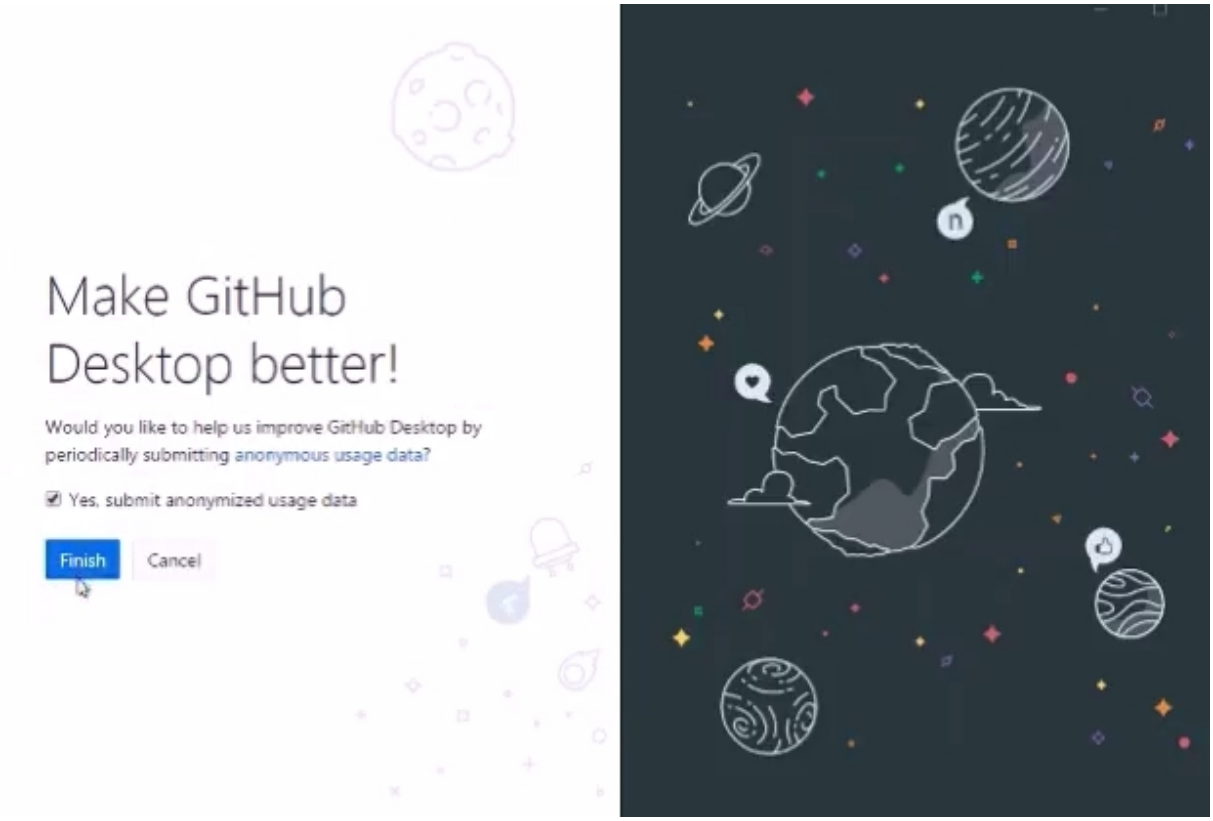


Enter in your username or email address, and your password.



Configure Git. You can just accept the defaults.

Click on Finish.

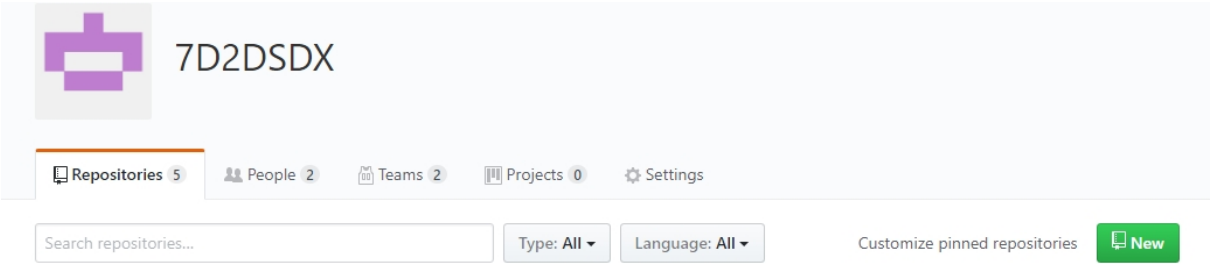


Adding a new Repos

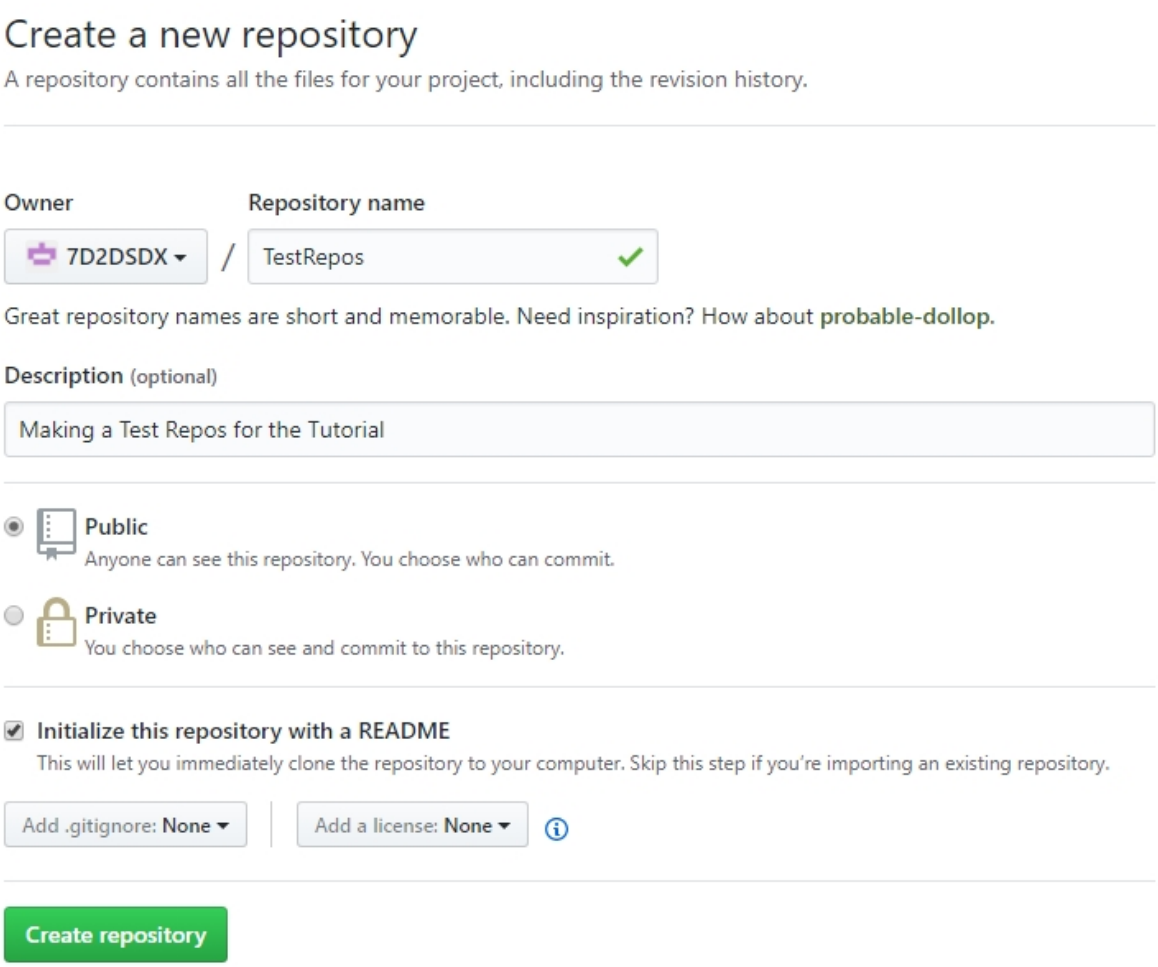
Once you have GitHub Desktop installed, and an account created, it's time to make a "repo", which is short for Repository. For each of your projects, you'll usually want to have a separate repos for.

In your web browser, go to your main GitHub Page. The website is http://github.com/<YourUsername>

Click on the New button on your main screen.

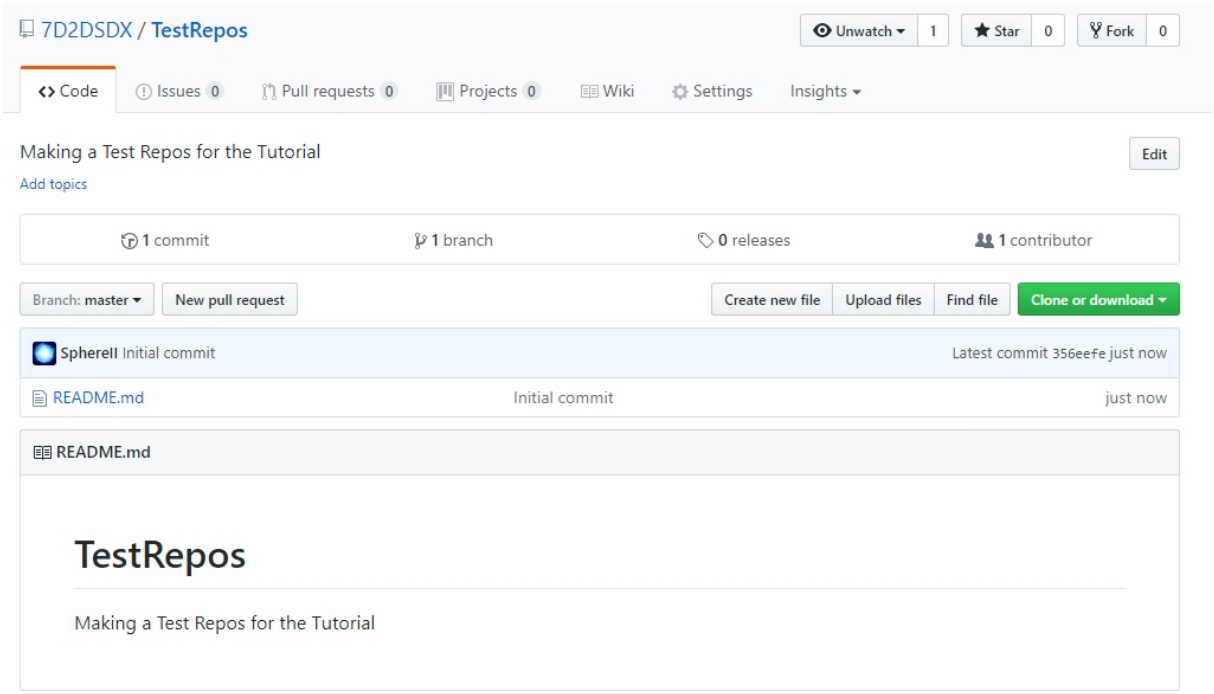


Fill in the form with relevant values:

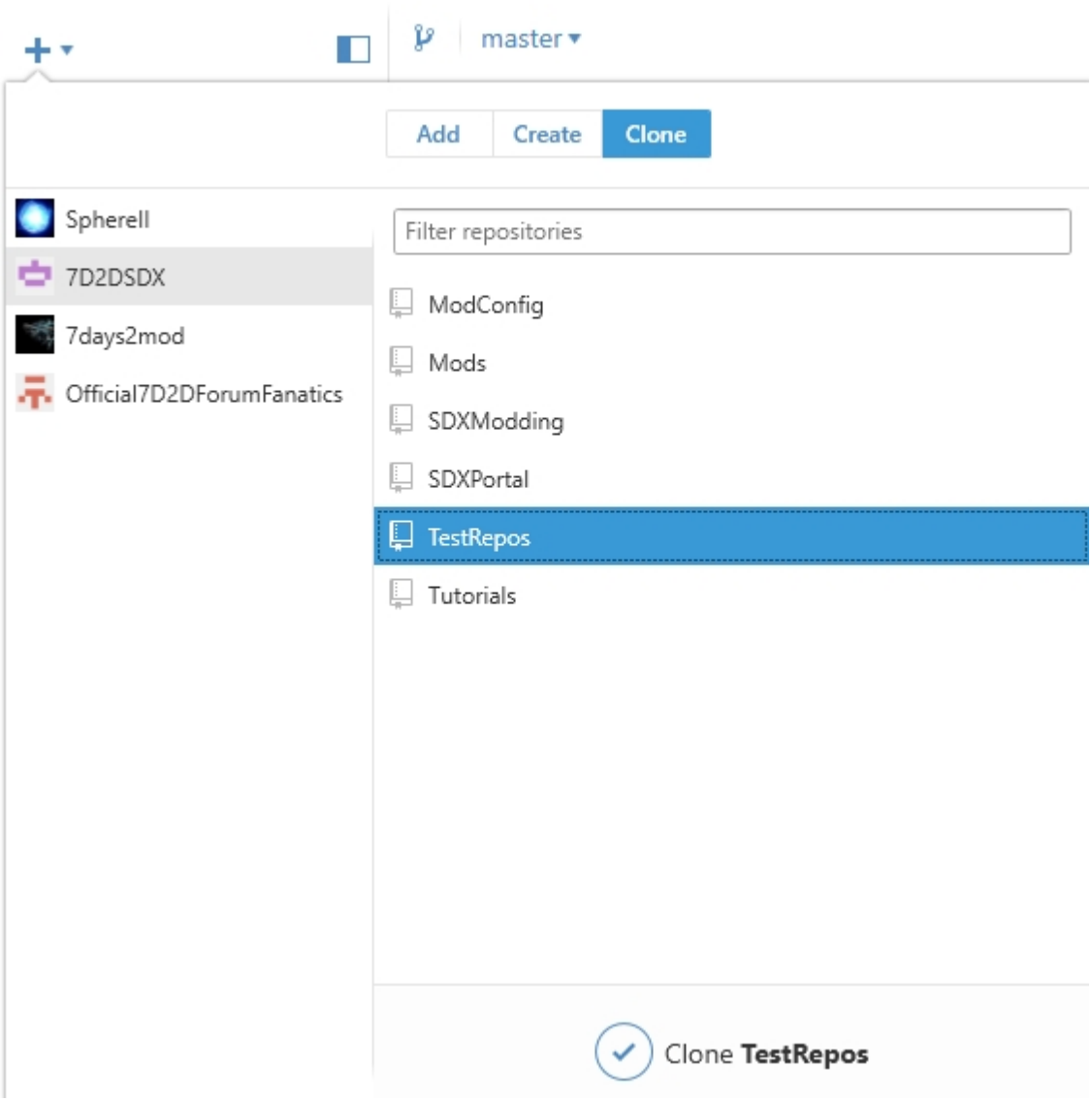


We are making a new repo called "TestRepos", and we added a short description. We decided to "Initialize this repository with a README", so it'll do a bit of leg work for us. We can then edit that readme to add more stuff

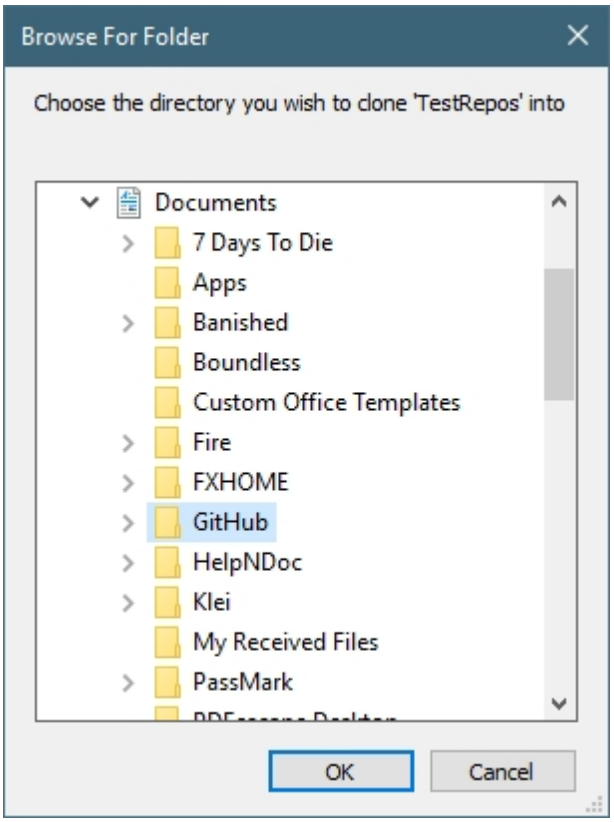
Once you've clicked on "Create repository", you'll be taken to your new GitHub Repos:



In GitHub Desktop, click on the Plus Sign at the top corner, select your account ("7D2DSDX" in this example), and select which Repos ("TestRespos" in this example):



Once you click on "Clone TestRepos", you'll be asked where you want to clone, or copy, the repos. By default, it's under Documents\GitHub\

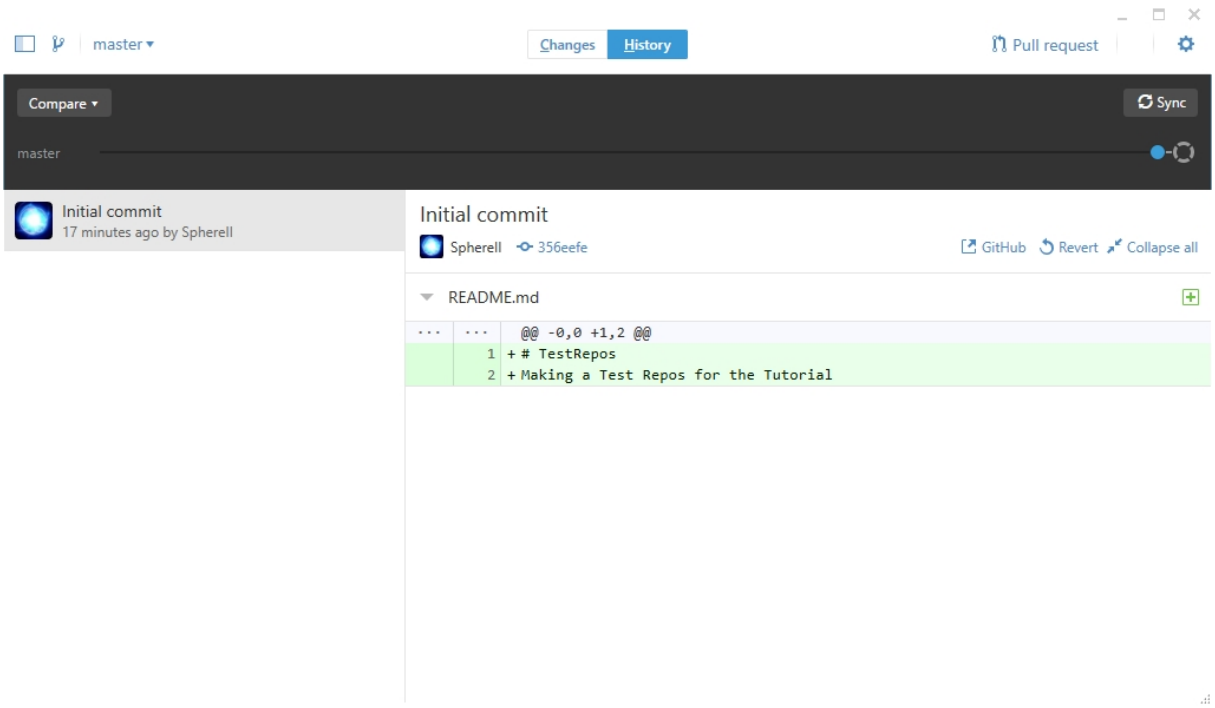


Clicking okay will create a new folder under Documents\Github\TestRepos\.

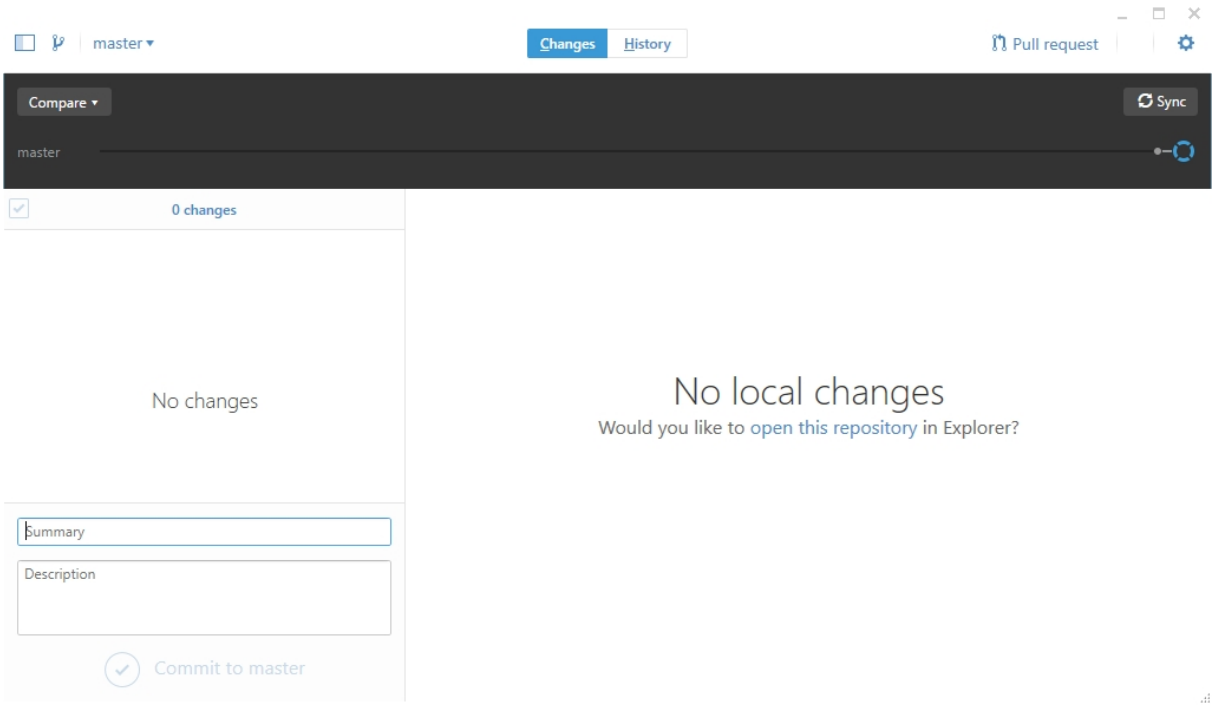
Adding Files

In GitHub Desktop, it's easy to add or update files.

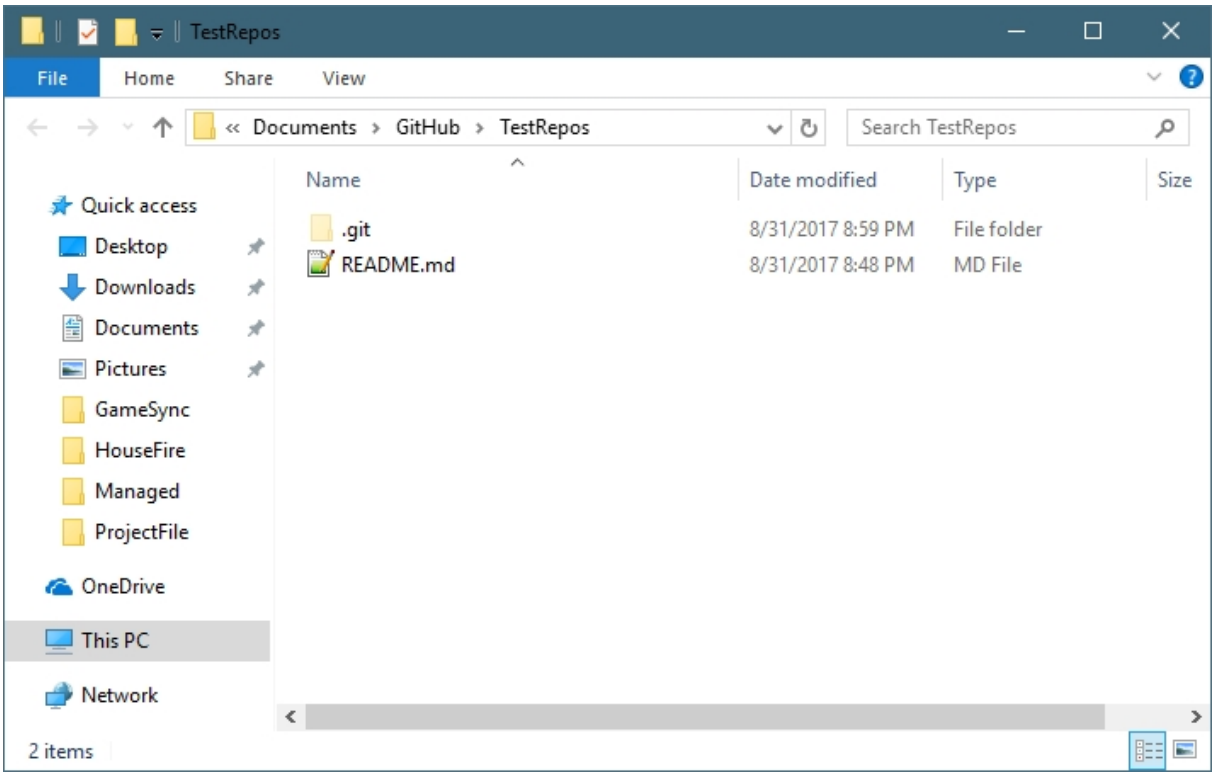
This is your default view:



To see changes, click on the "Changes" button, in the center of the screen.



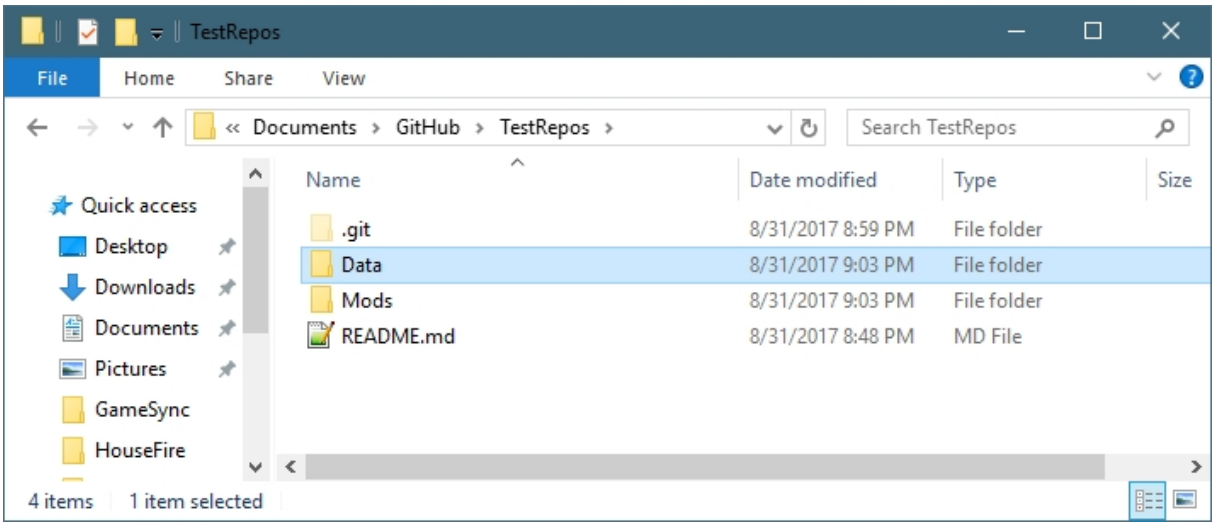
You'll see No Local changes, if you haven't added any new files, or made any changes. Let's click on "open this repository" in Explorer

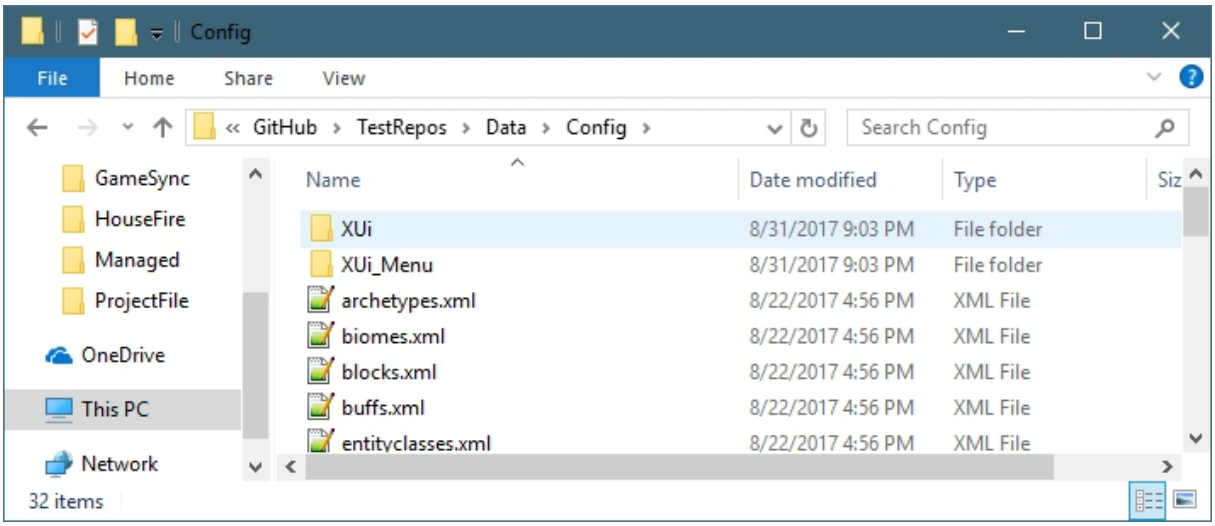


In a fresh depot, if you selected "Initialize this Repository with a ReadMe.md", you'll see the ReadMe file. You can edit this in Notepad++, or your favorite editor.

So what now? Since this is Explorer, you can create folders, and add files of your mod as you normally would.

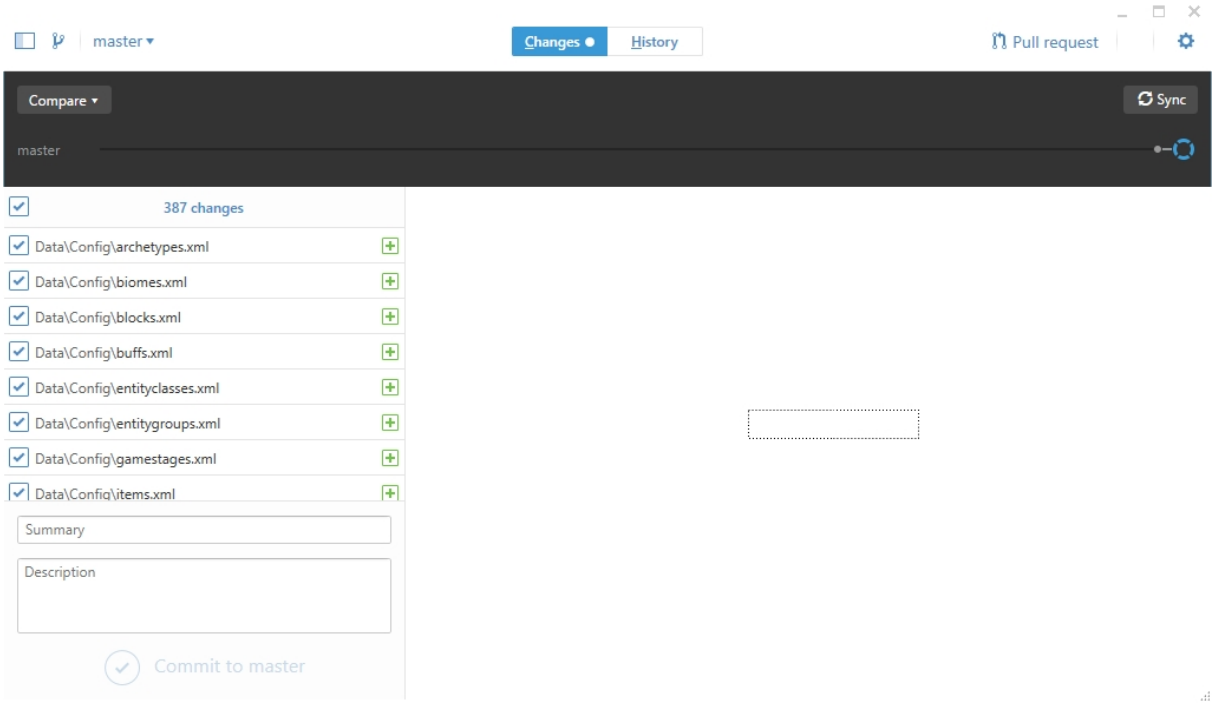
For this example, we used Valmar's Overhaul as a template.





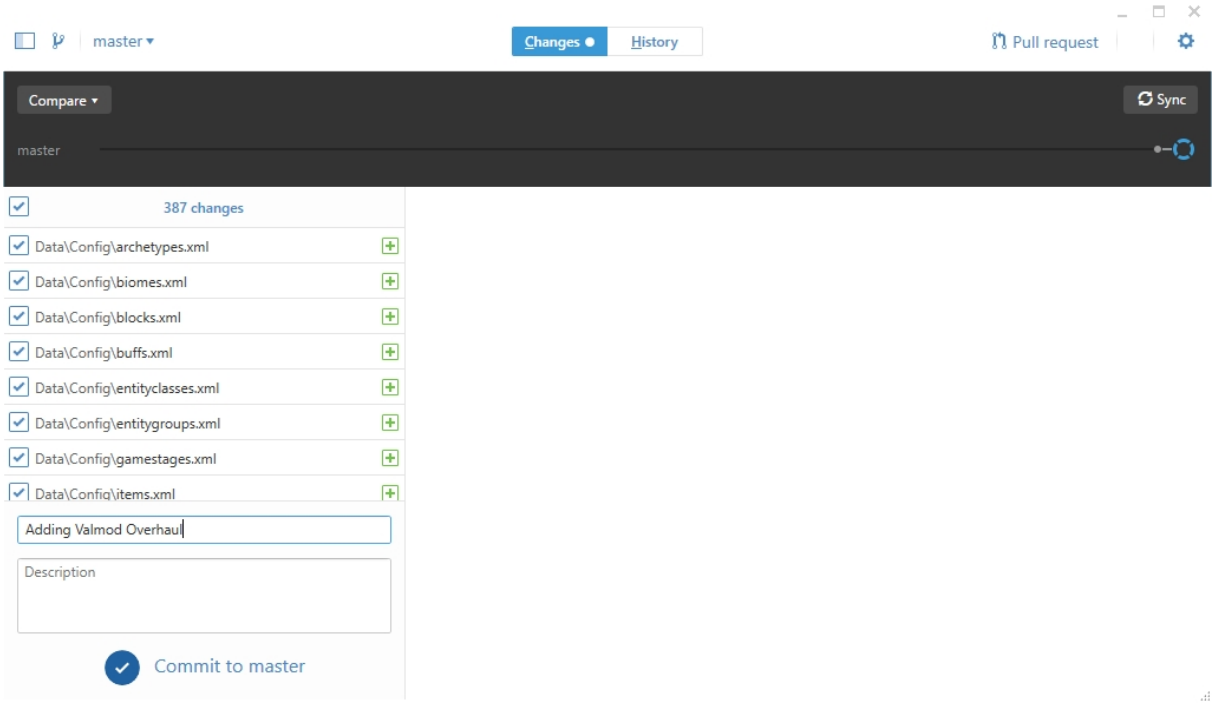
It's important to maintain the correct folder structure of your mod, placing all the files in the right spot. This makes it easier for the users to download and install the mods.

After you are done with your Explorer changes, go back to GitHub Desktop, and look at the Changes now:



In the above, we can see we made 387 changes. Let's do a test commit, which means, we are going to upload the files.

In the Summary box, enter in a short description of your changes.

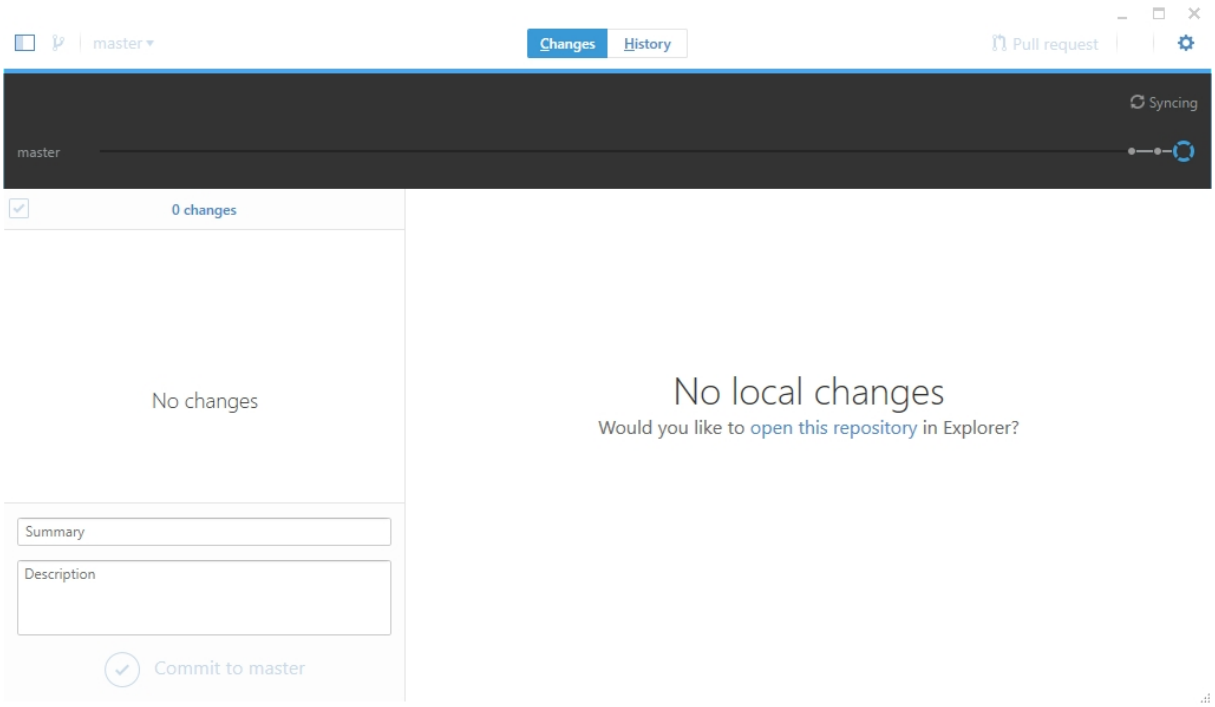


Then click on Commit To Master. *This button won't become enabled until you type something in the Summary.*

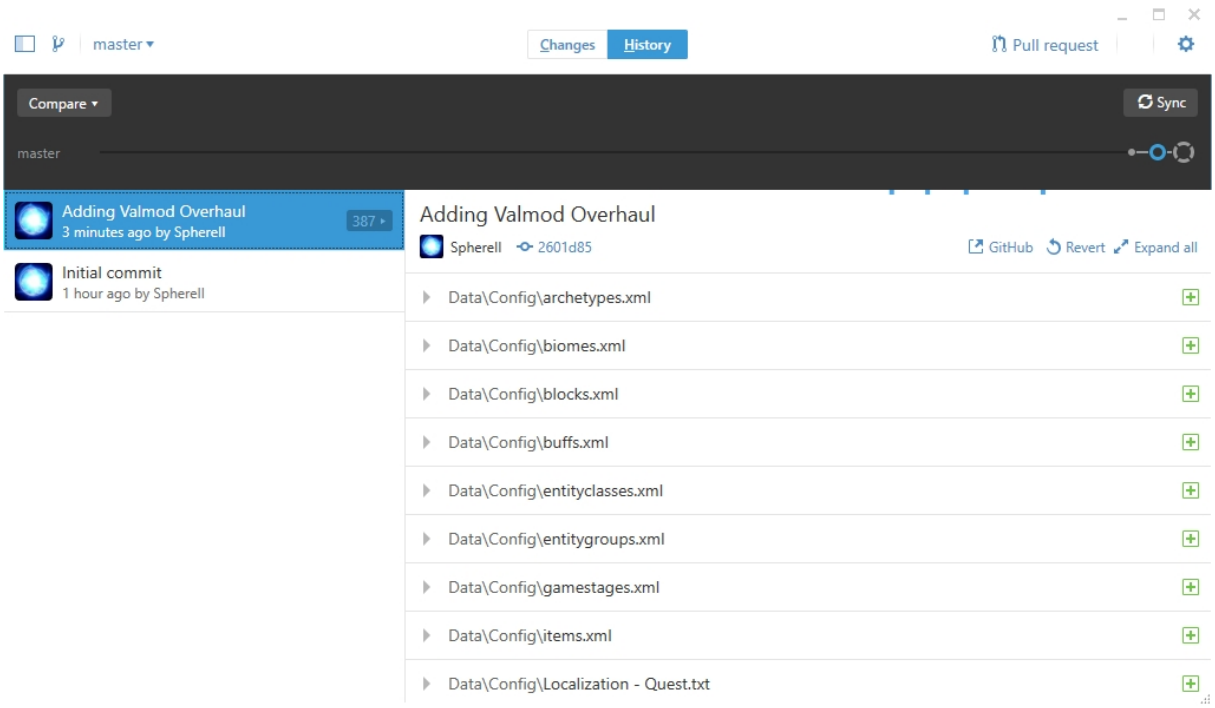
The Commit to master button will then gray out as it uploads.

Once it's complete, you'll be back to the "no local changes". That's because the files that are online are exactly the same as your local copy.

Next, click on the Sync button at the top of the screen.

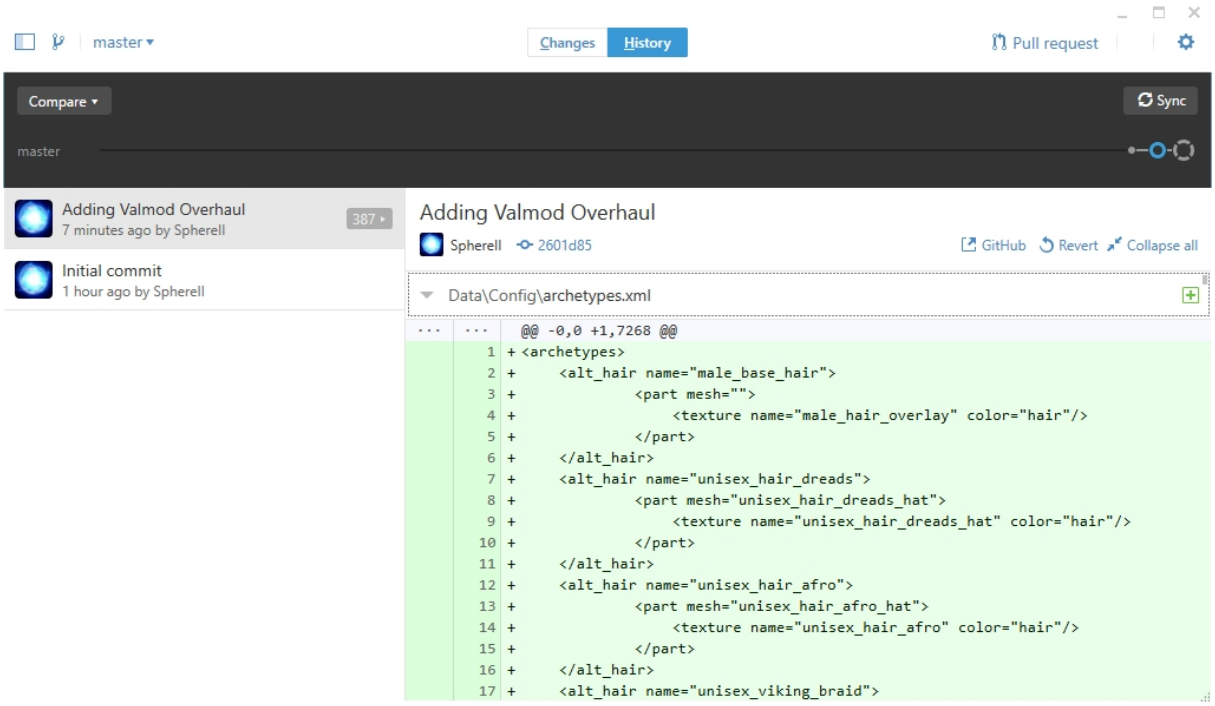


Click on the History button at the top of the screen

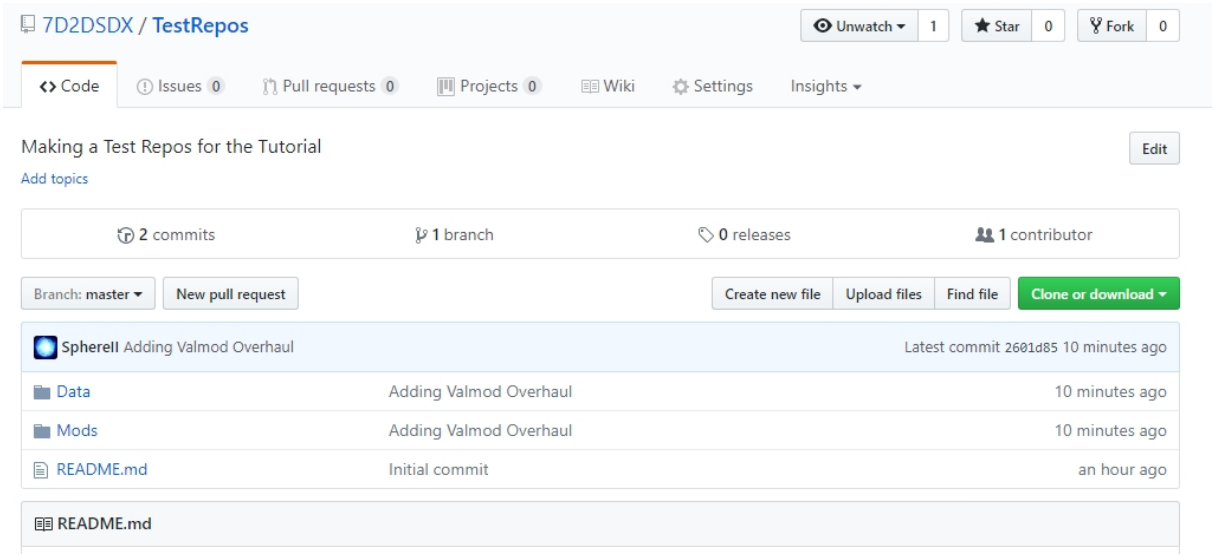


This window shows all the changes we've done.

If you click on the Green plus, you'll see the changes that were done to that file



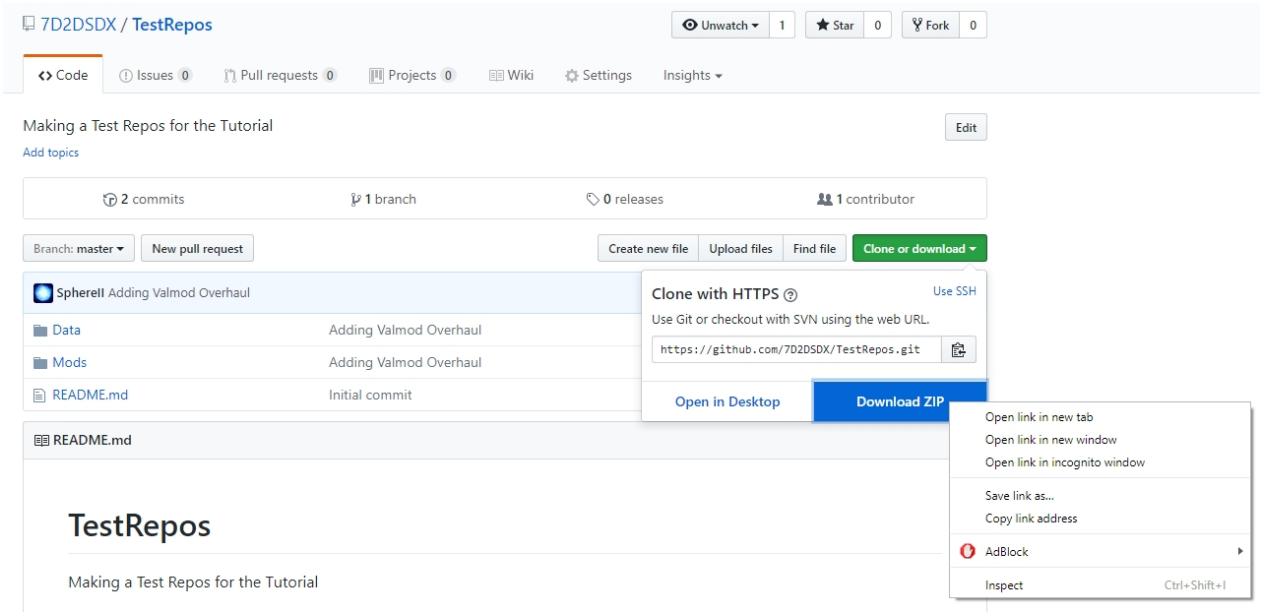
We can then check the website again to see it available:



Sharing the Link

Having your mod hosted on GitHub is great, but how do we let other people use it?

On your GitHub web page, click on the Clone or Download green button, then right click on Download Zip:



You can "Copy Link address", and it will give you a link that looks like:
https://github.com/7D2DSDX/TestRepos/archive/master.zip

You can now give that link to your players, and they'll be able to install your mod.

Creating a SDX Mod structure

dnSpy Quick Review

dnSpy is a tool to reverse engineer .NET assemblies. It includes a decompiler, a debugger and an assembly editor (and more) and can be easily extended by writing your own extension.

This tool, for us, allows us to look and make changes to the game, on a level far deeper than XML changes. We've included dnSpy as part of the SDX Modding Kit, but there are others out there.

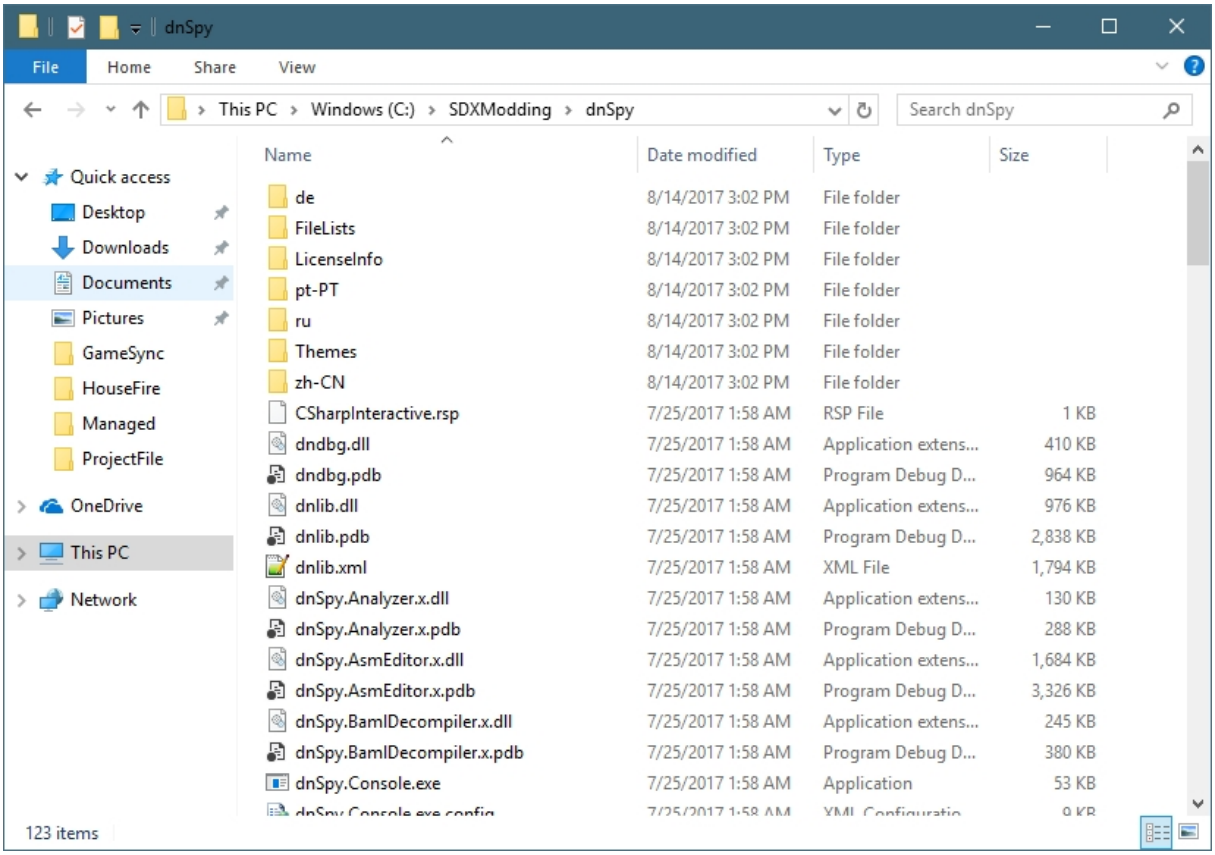
dotPeek: https://www.jetbrains.com/decompiler/

ILSpy: http://ilspy.net/

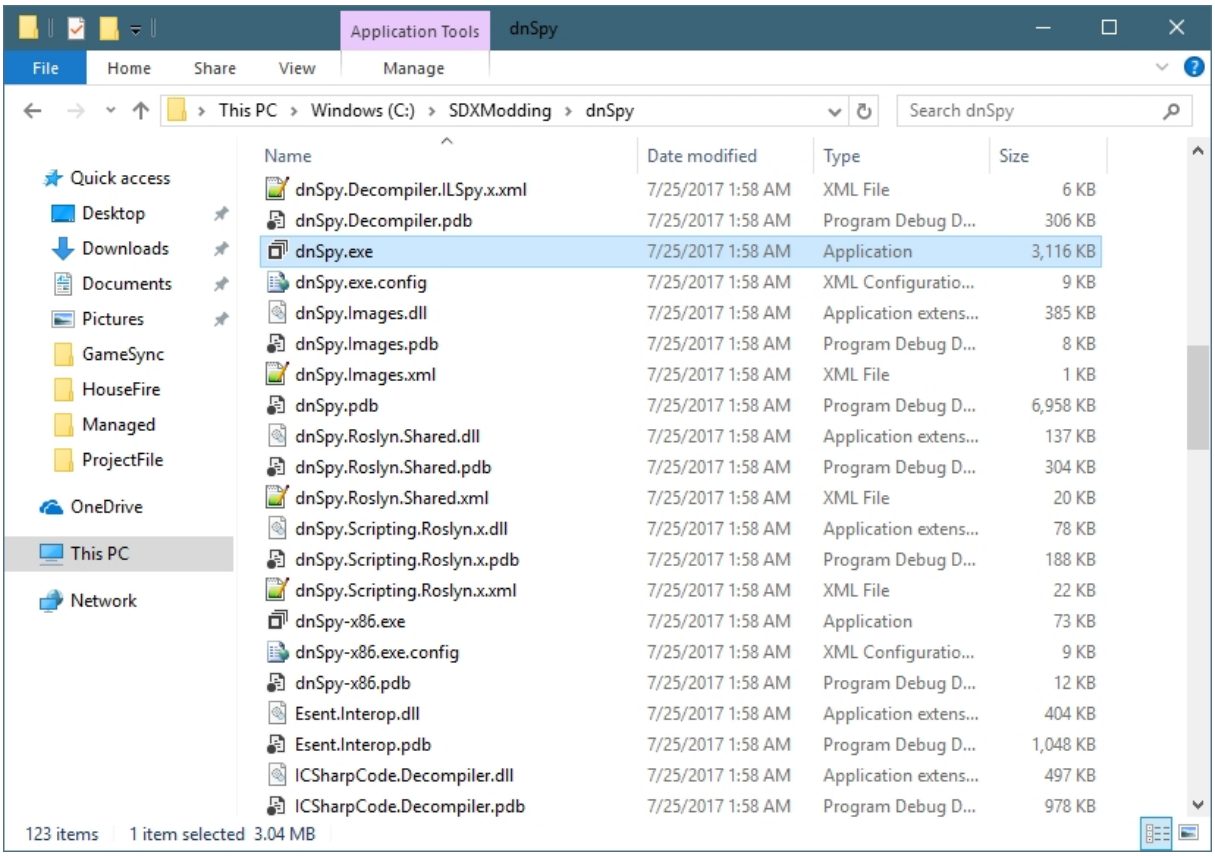
In order to keep the tutorial concise, we will be using dnSpy.

Starting dnSpy

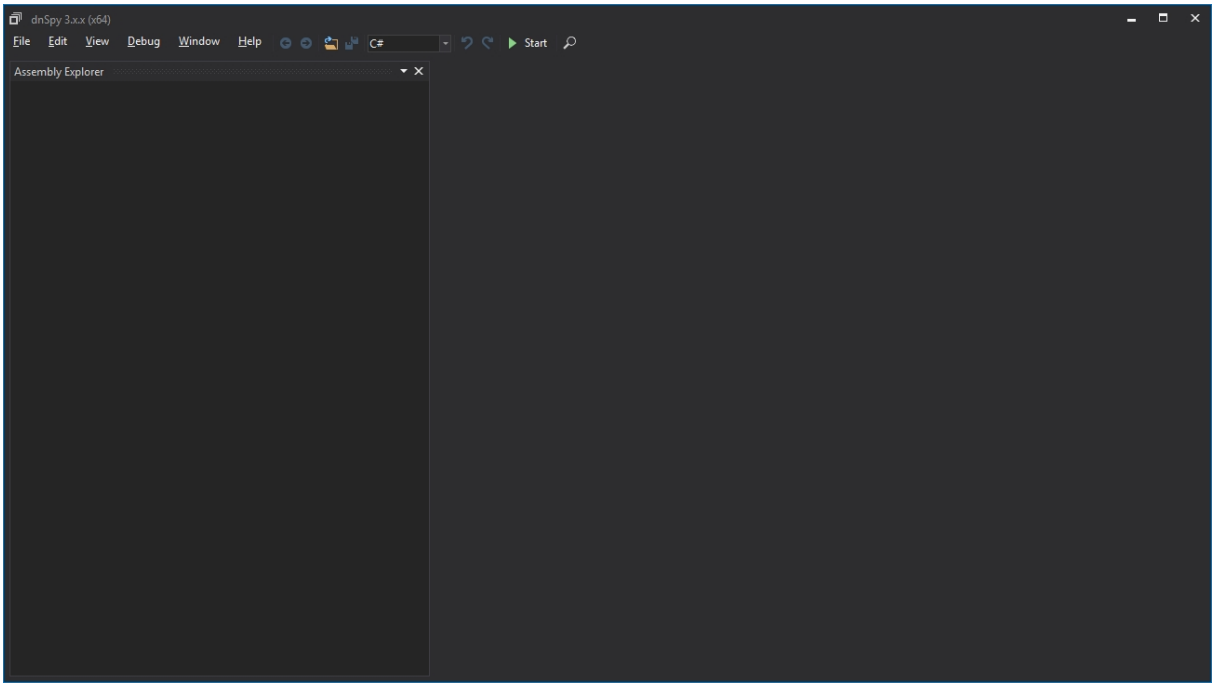
You'll find dnSpy under the C:\SDXModding\dnSpy\ folder, if you've started with the SDX Modding Kit



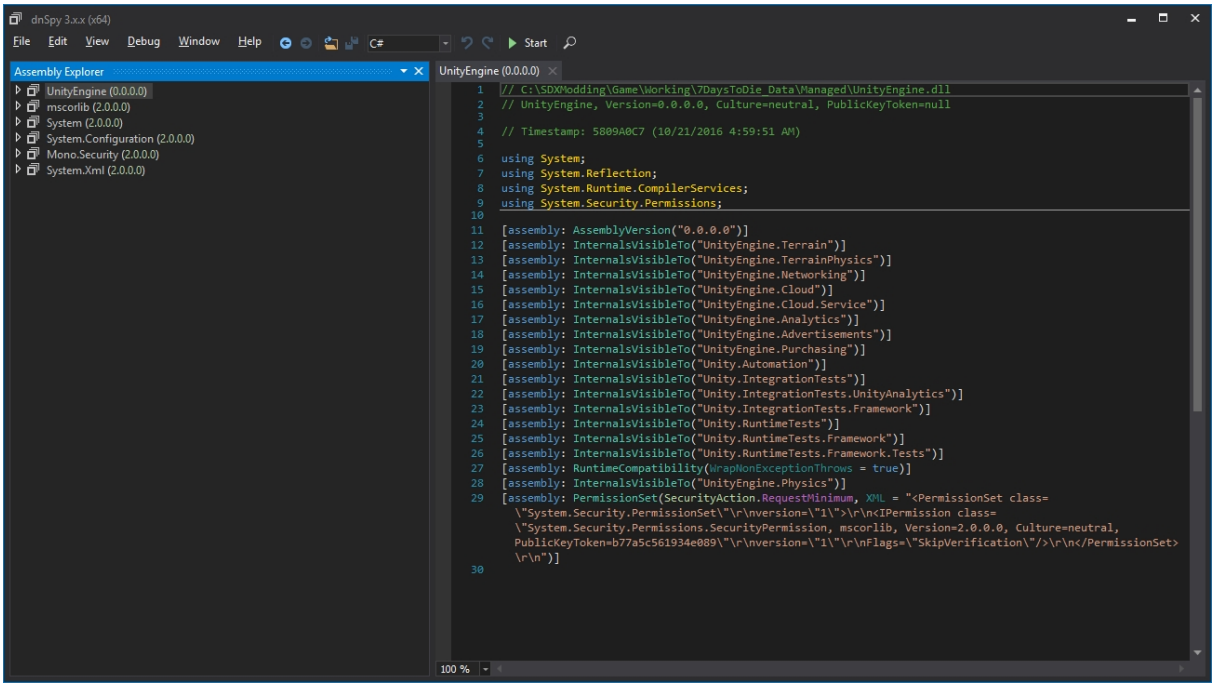
Scroll down until you see dnSpy.exe, and double click on it:



Once loaded, you'll see a screen like this:



Depending on your computer setup, you may find some other files listed under the Assembly Explorer.



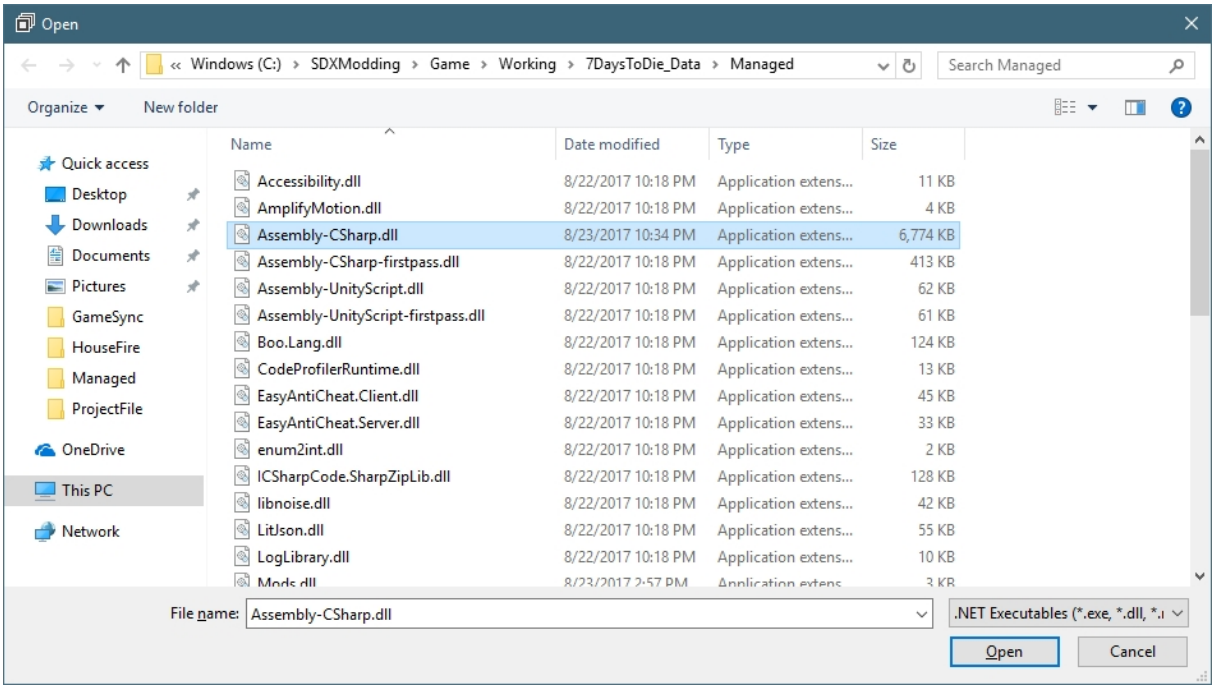
To make things clean for the purpose of this tutorial, go to the File menu, and click on "Close All". This will clear the list. You only need to do this for the tutorial purposes, as it makes a cleaner interface, and easier to understand.

Note: As you explore various DLL files using dnSpy, files will be added to the Assembly Explorer automatically. That indicates there's a link between the file you are looking at, and the new file.

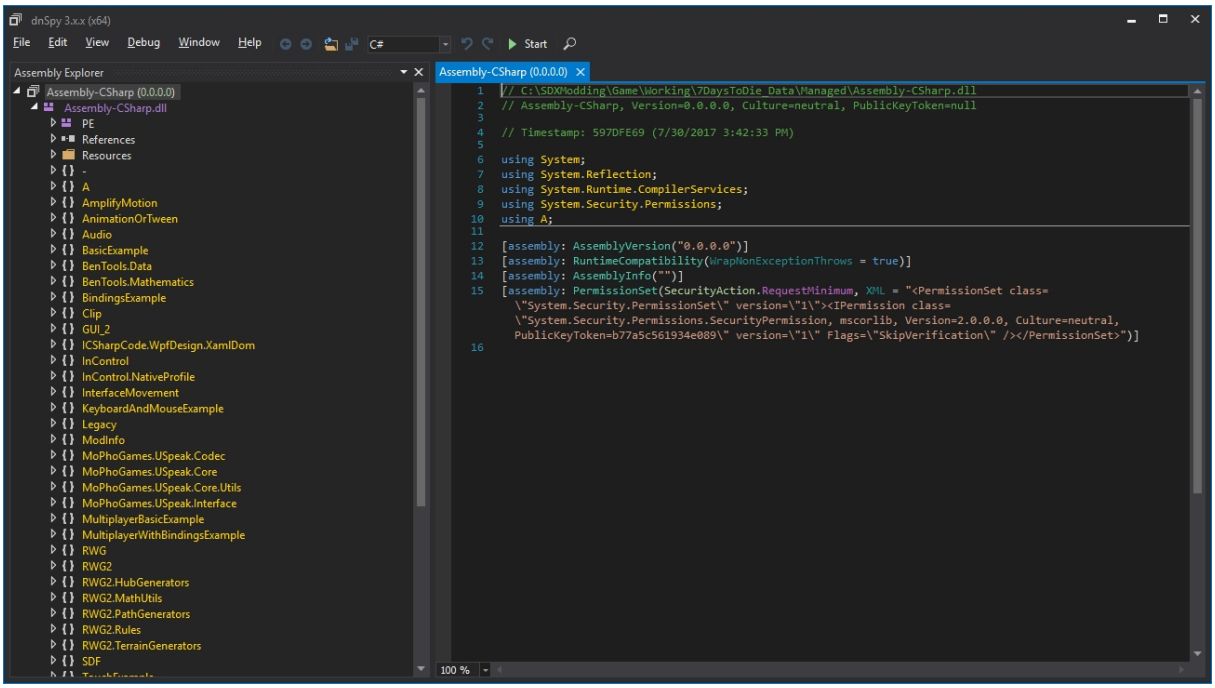
Opening up the Working copy

As with our previous examples, we are going to be working off our Working folder.

In dnSpy, click on the File menu, and select "Open...". Navigate to your C:\SDXModding\Game\Working\7DaysToDie_Data\Managed\Assembly-CSharp.dll

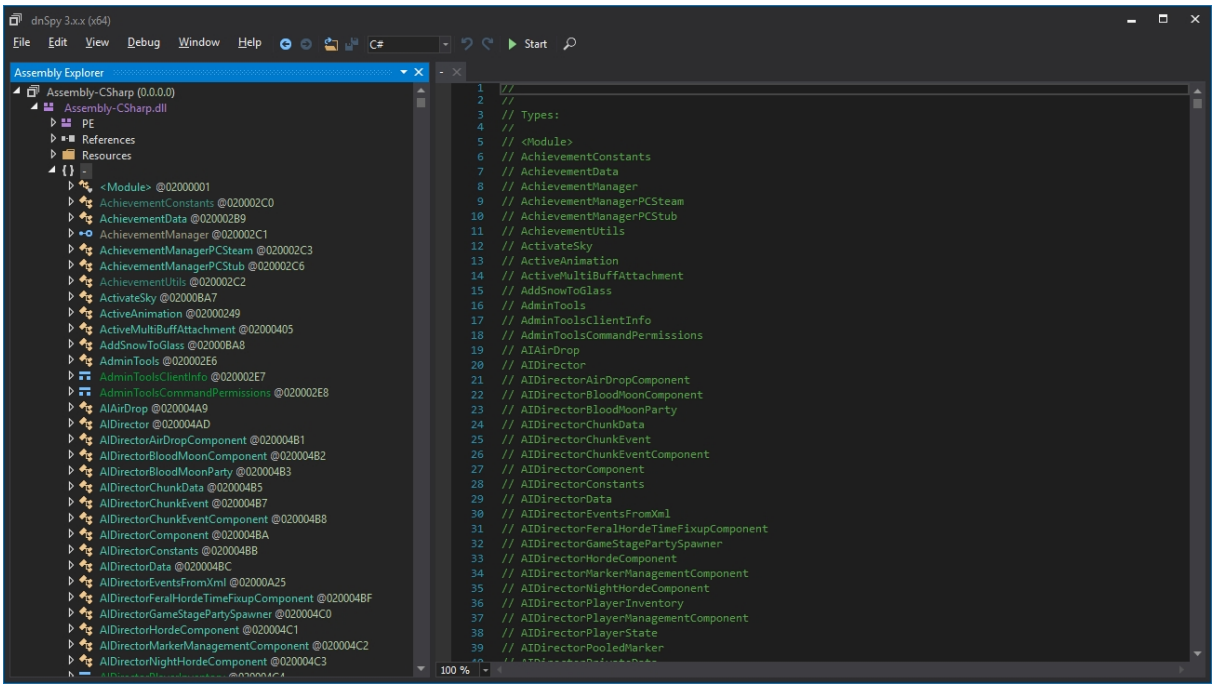


Click on Open.



Expand the Assembly-CSharp (0.0.0.0) by click on the Arrow, until your screen matches the above.

The Assembly Explorer window shows you a list of namespaces. These are usually isolated parts of code. Thankfully, we aren't too interested in these. What we are looking for, is the "{ } -" line.



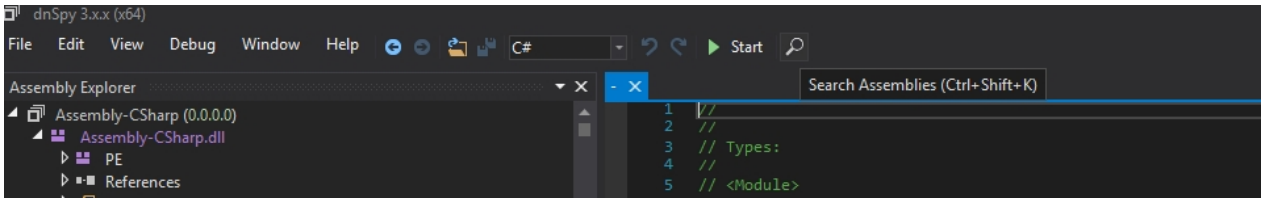
Searching for the Class

In the earlier parts of the Tutorial, we covered the Bigger Back Pack mod for SDX. We are going to revisit some of the mechanics of that mod, by exploring the DLL and finding out exactly what SDX did automatically for us.

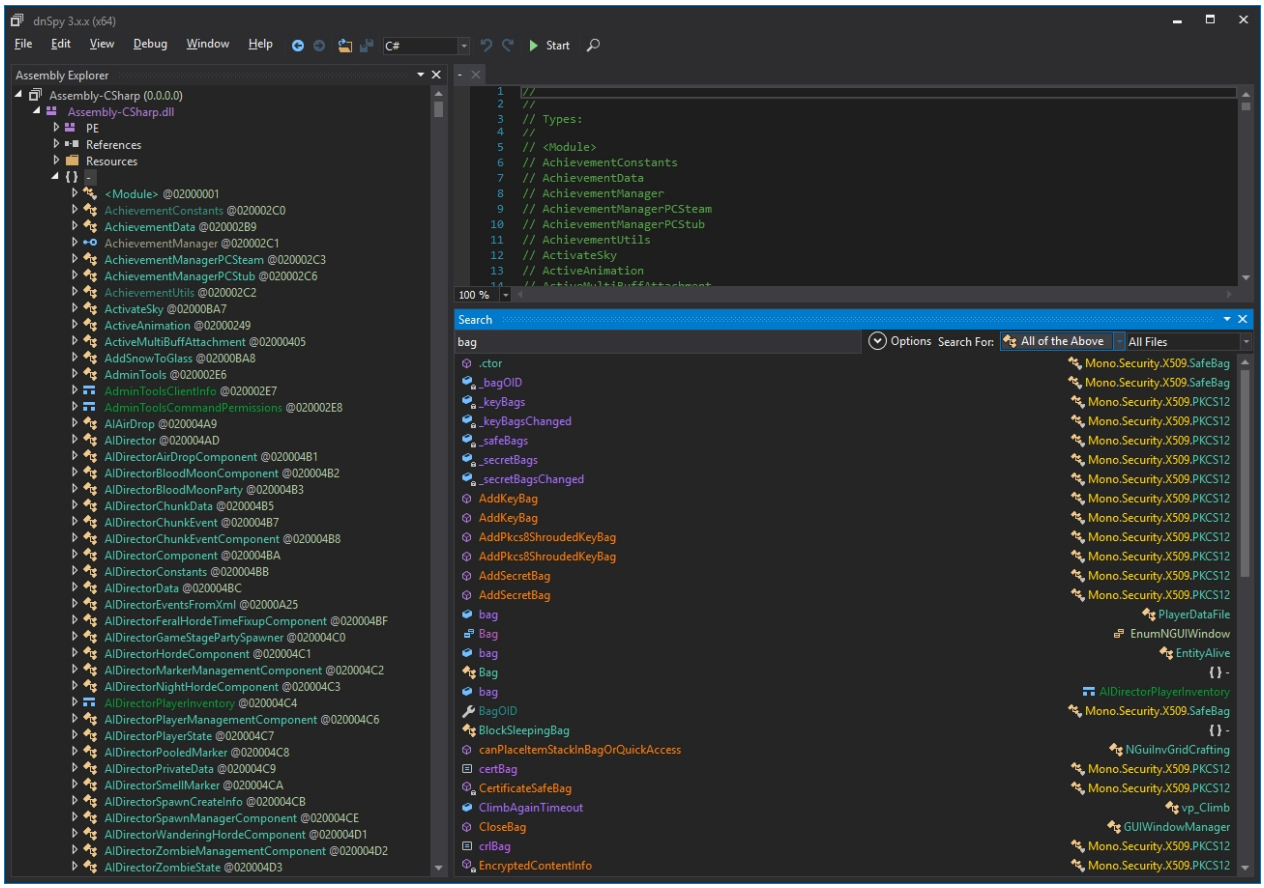
For this section, we are going to follow this [7 Days To Die Forum Post.]

For the Bigger Back Pack mod, we want to edit the Bag class.

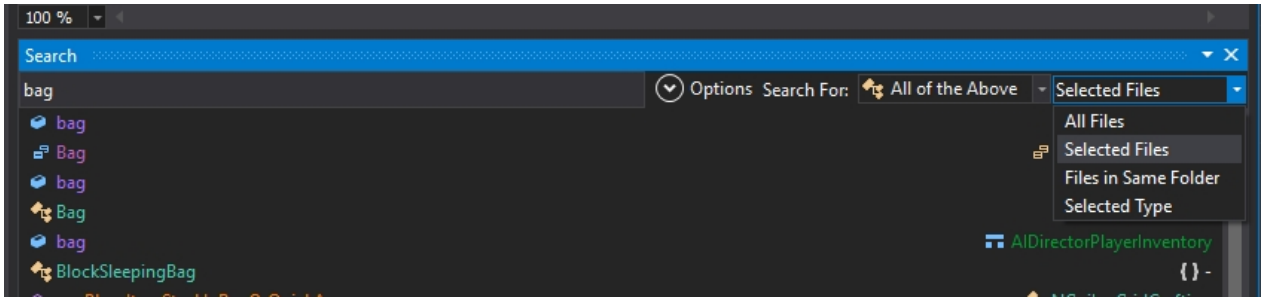
Look for the Magnifying Glass on the main window, and click on it.



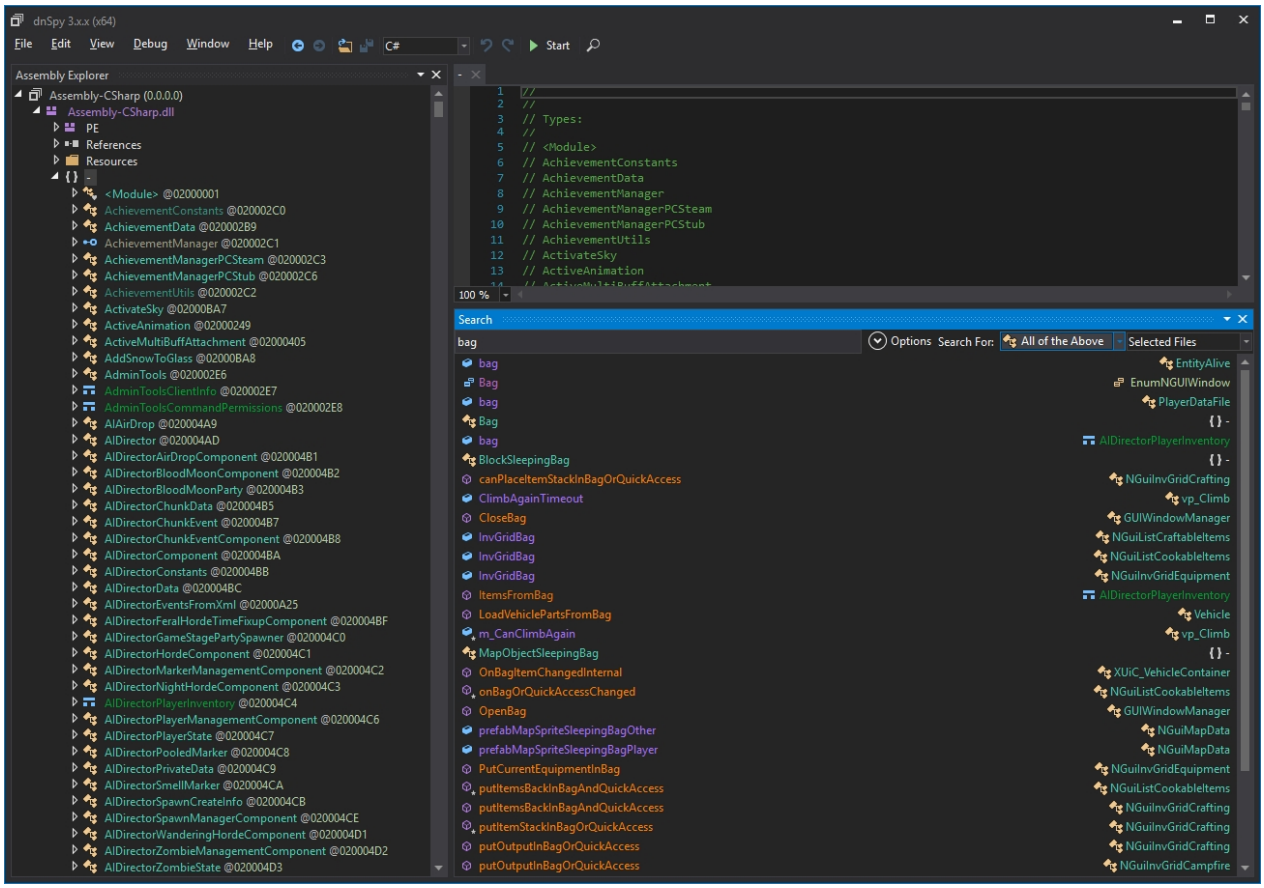
The "bag" search comes up with a lot of different references. You can manually look through, trying to identify what you are want to change. You could also try changing the filter.



"All Files" Filter

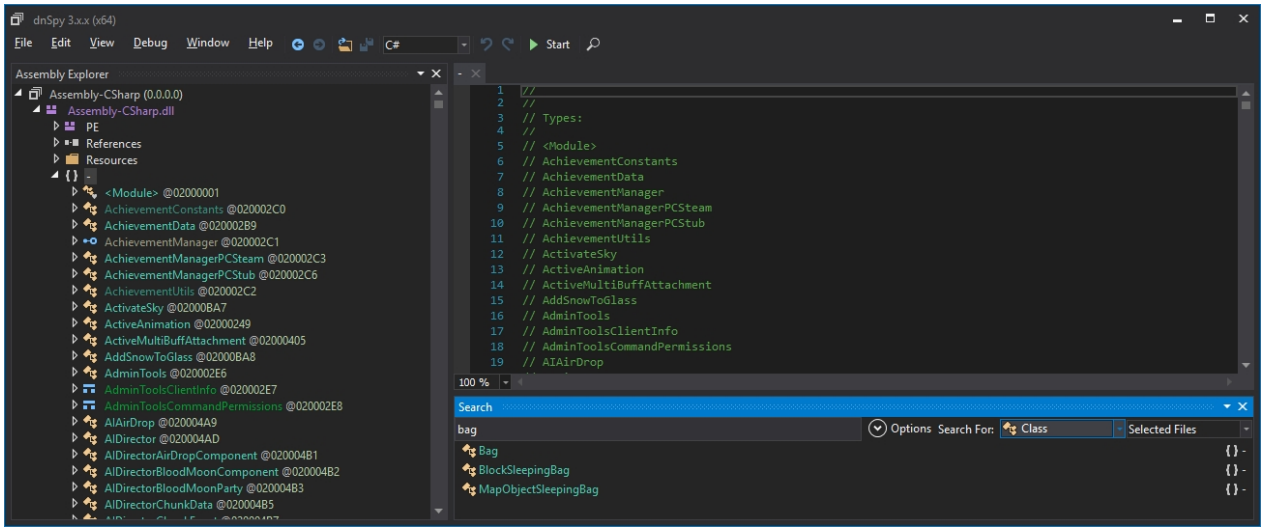


By default, it searches for "All files" that are opened. You can narrow it down by "Selected Files".



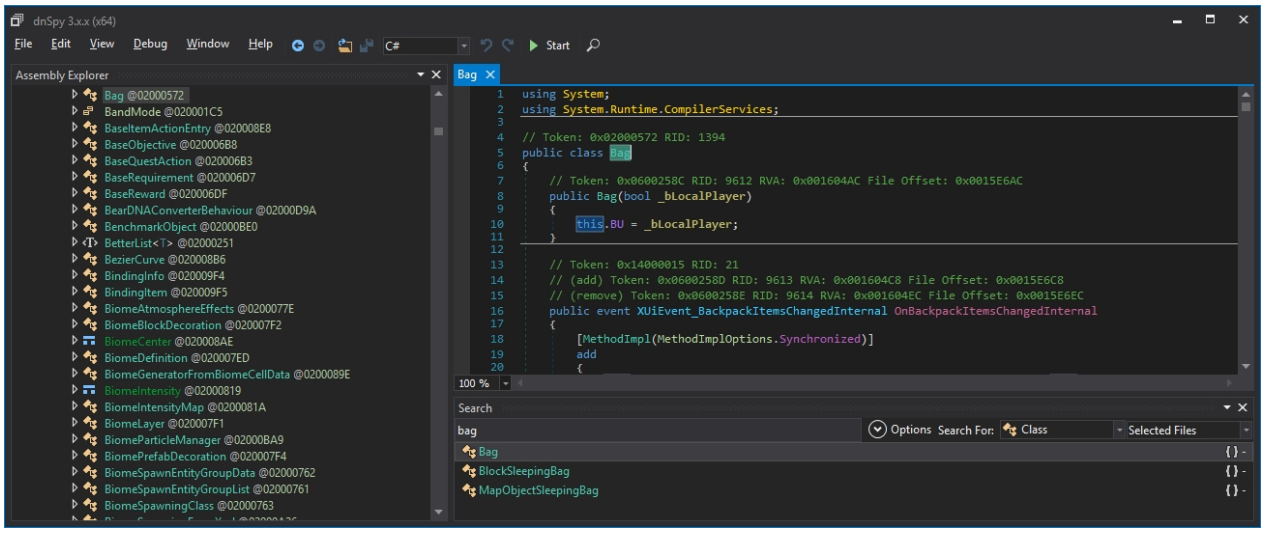
That narrowed the list down a bit. The Bag reference we are searching for is a C# *class*.

In the "All of the Above", we can further choose out filter by Class

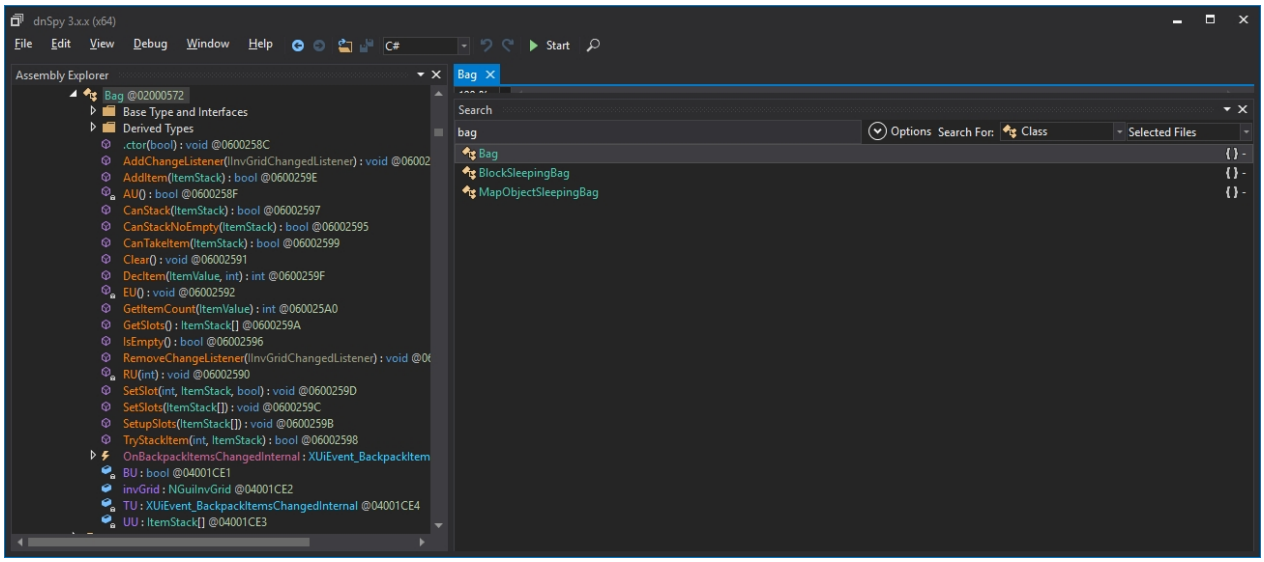


Now we only see three: Bag, BlockSleepingBag, and MapObjectSleepingBag.

For this Tutorial, we are just interested in the "Bag" class. Double click on it to be taken to the class.



You'll notice that the Bag is highlighted over the Assembly Explorer, and the code is open on the other window. Click on the arrow by the "Bag", in the Assembly Explroer:

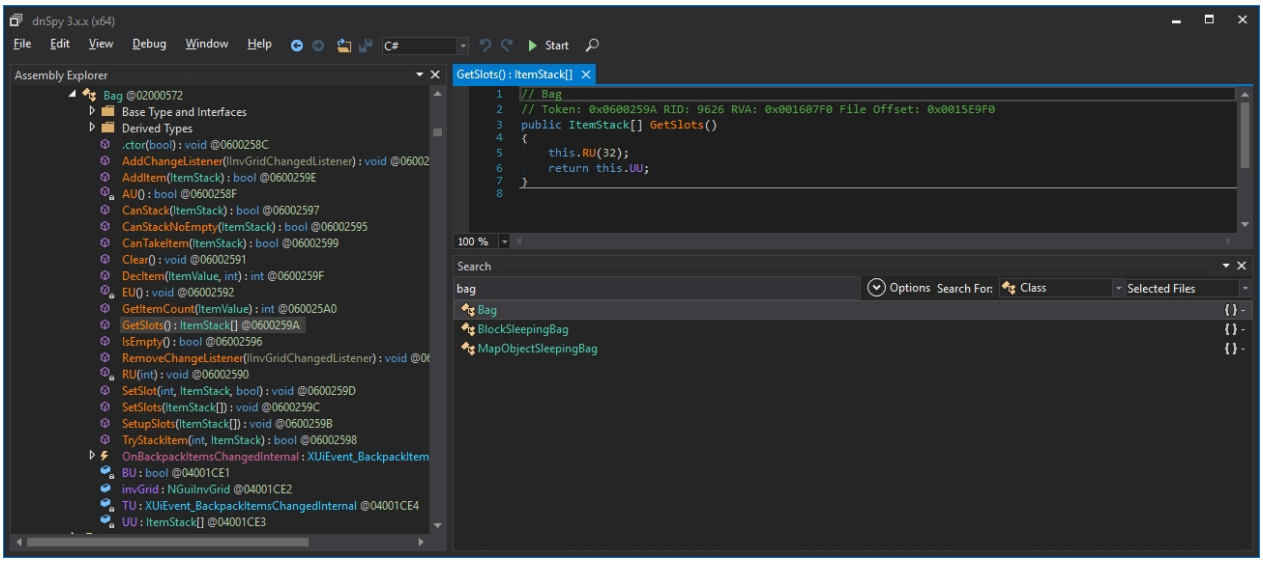


This shows all the methods that are the type class. You may click on each one to view the contents of it.The one we are interested in, is just the "Bag".

Editing a Method

In the previous section, we found the Bag Class. The next step is to look at the GetSlots method.

Note: Because De-obfuscation can change the way that each file and method looks, only follow the theory of this work flow, as opposed to hard truths. Such as "this.RU(32)" may not exist for you. It could be something different, like "this.SO(32)".



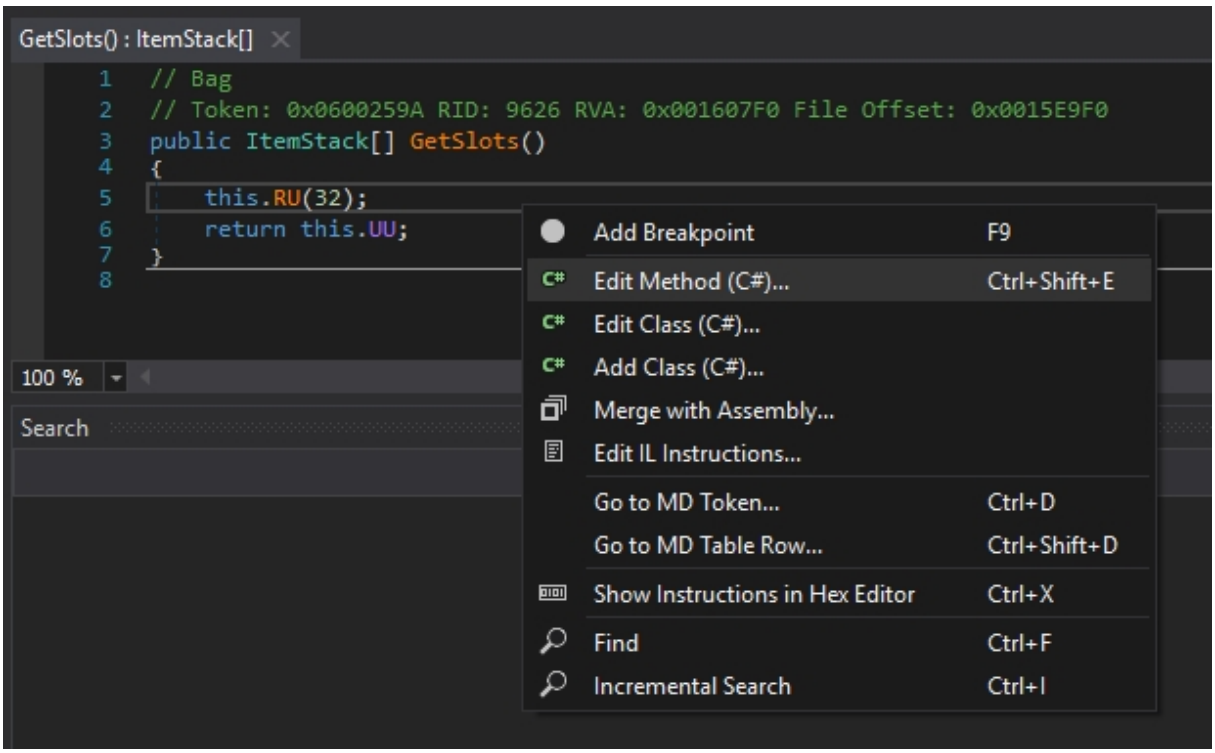
Double clicking on the GetSlots() in the Assembly Explorer will open up the Method.

The contents: this.RU(32) is the key. This tells that game how many slots are in the back pack.

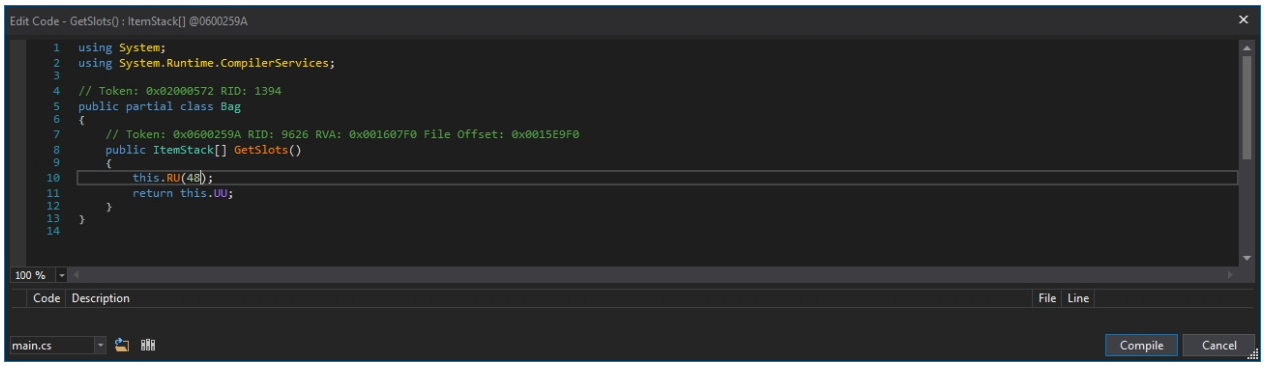
The contents look strange. this.RU(32) is such an odd name. That's because we are disassembling the code; and it's not always a clean process. When you see a strange method that has a call to a two letter method, like this.RU(32), it indicates that it will change every build. It will also look different from one computer to the next.

The next step in the instructions is to change the 32 value to whatever you want for a larger back pack. We'll use 48 as our new number.

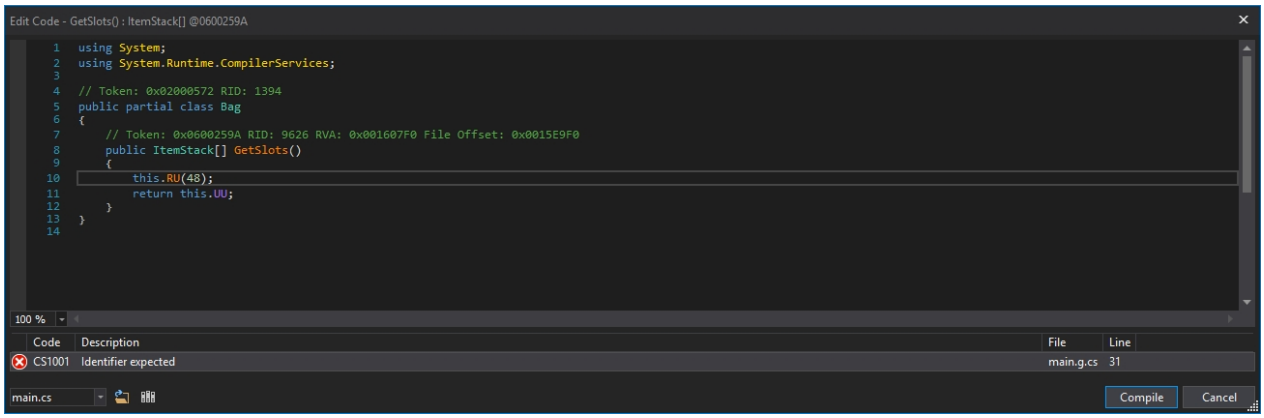
In the GetSlots() window, right click, and select "Edit Method (C#)".



Highlight 32, and change it to 48.

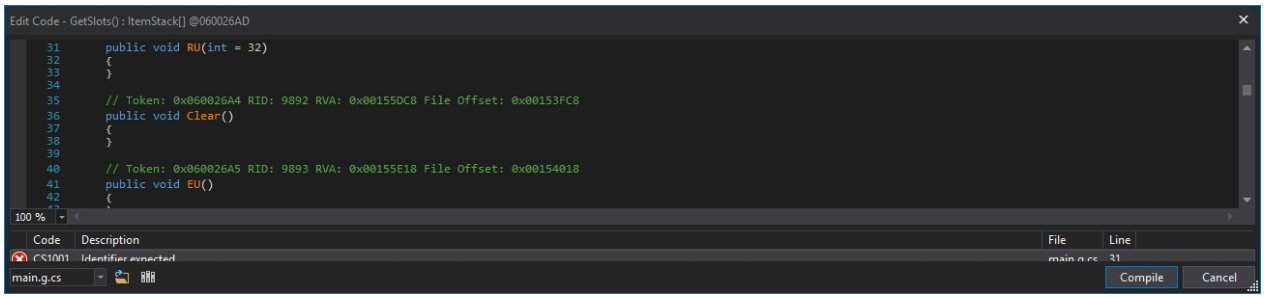


Then click on the Compile button.



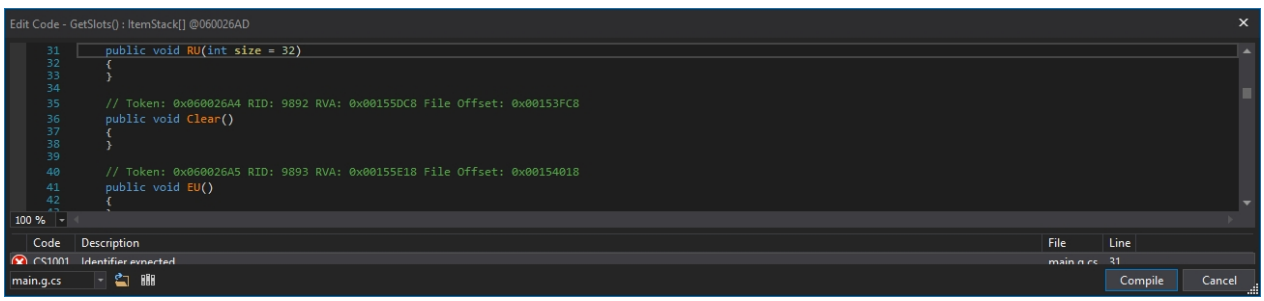
Oops! Identifier expected error pops up. That means something is wrong with our change. In the above example, it's really a really simple change: changing 32 to 48.

Double click on the "CS1001 Identifier expected". You'll be taken to the RU method.



The RU method is actually empty. but the parameter, is not correct. "int = 32" is invalid C# code. What it's trying to do is set a default value, if none are passed.

We can fix this by adding something to the parameter:

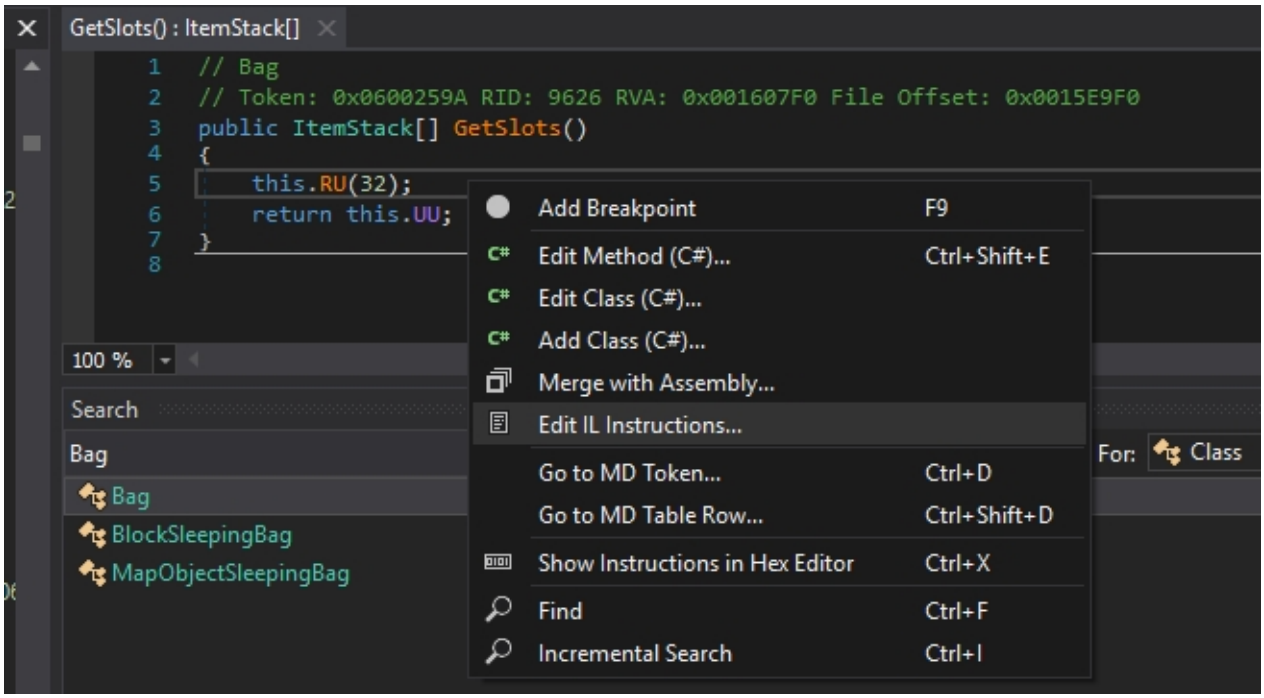


Now it says "int size = 32". Press compile, and you'll see that both the RU method, and the GetSlots() call

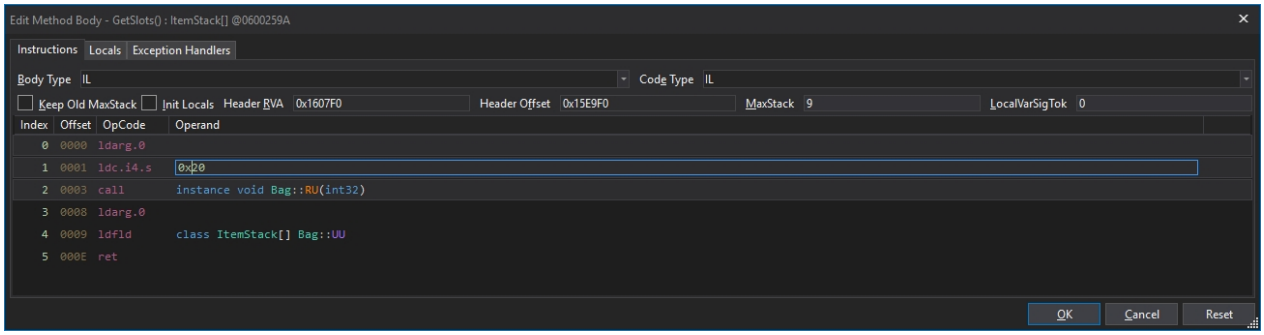
were updated.

Let's look at the method another way to make the change.

In the GetSlots() window, right click, and select "Edit IL Instructions"

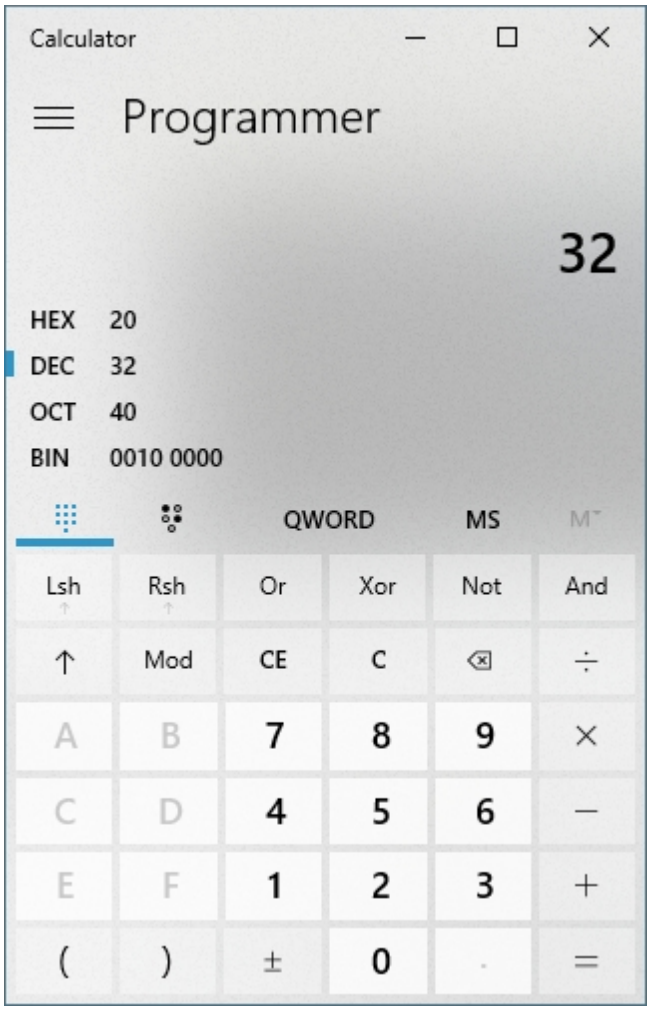


A new window will pop up, allowing us to change the code:

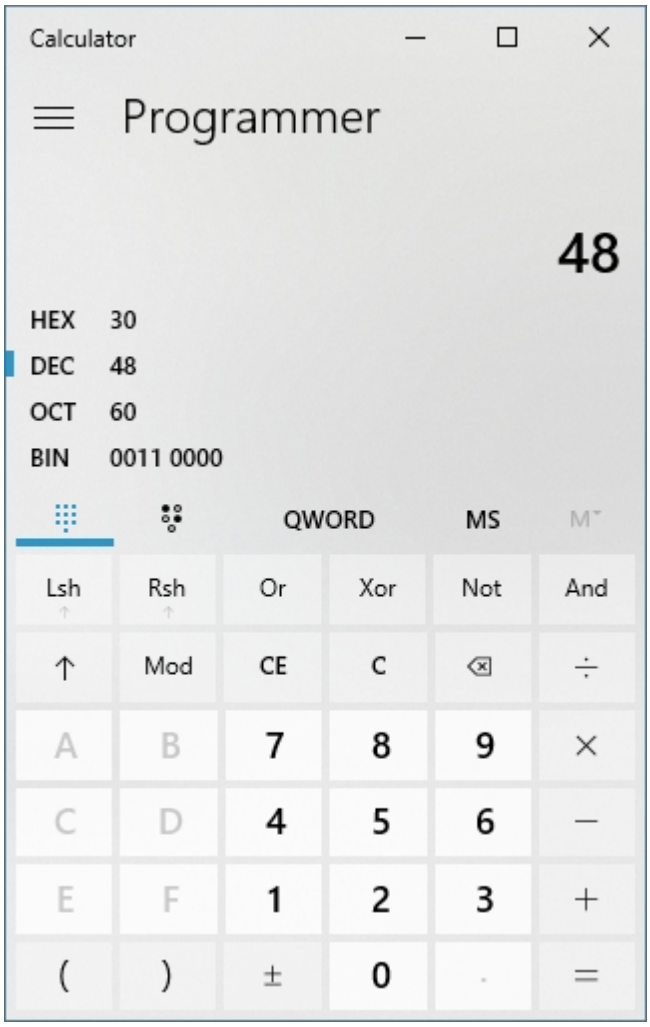


Where did the 32 number go? The 0x20 is Hex, for DEC 32. So dnSpy converted the number for us in the "Get Method", but won't allow us to change it.

Using Windows Calculator, we changed from "Standard" to "Programmer". Then we typed in 32. It shows Hex is 20.

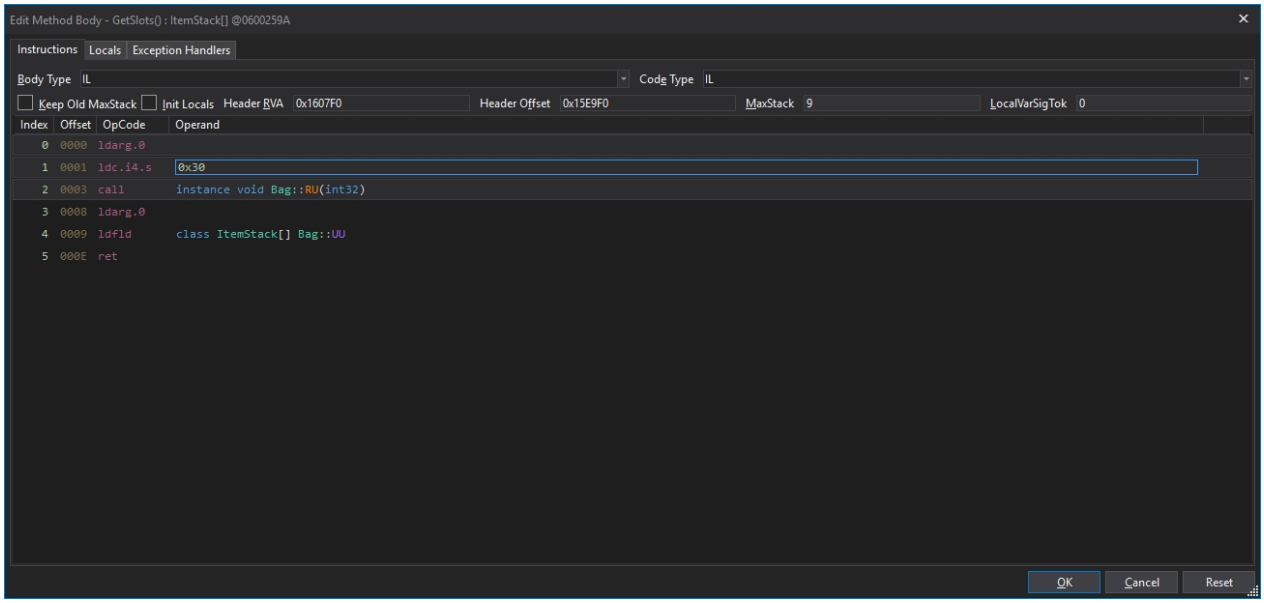


So we if want to go with 48, we'll want to put in:

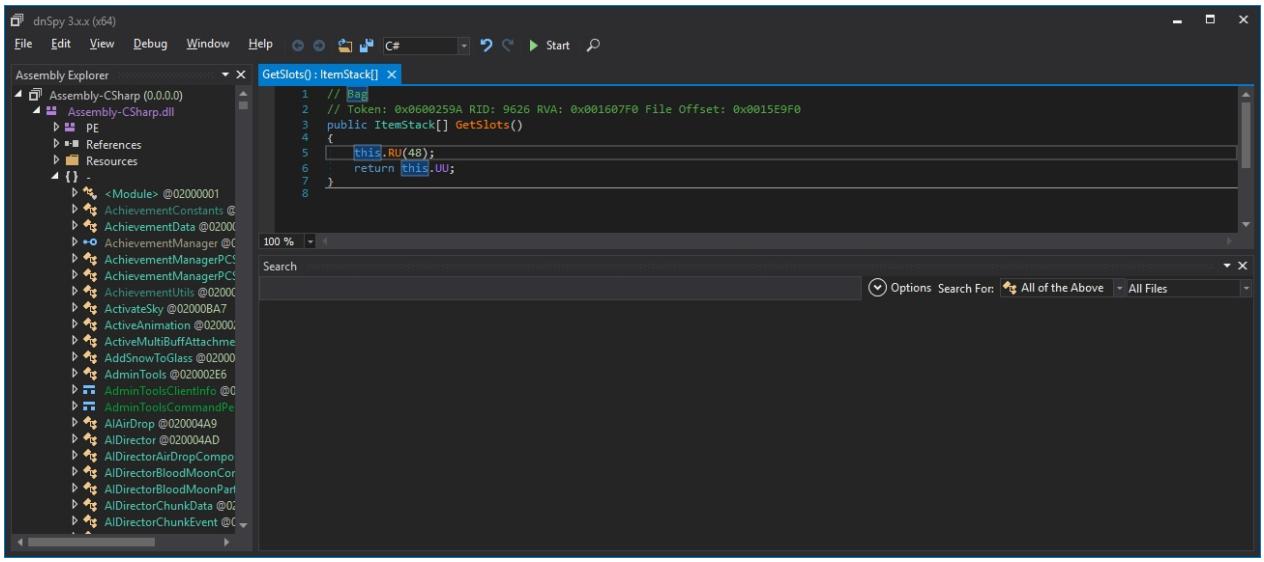


So 48 in hex is 30.

Click on the 0x20 value, and change it so it says 0x30.



Then click on OK.

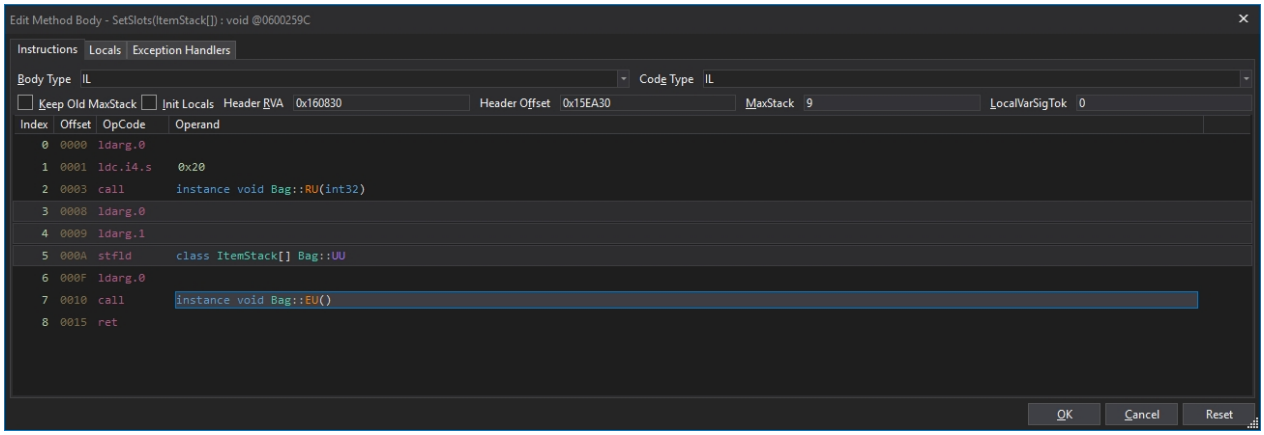


So that worked. If you are trying to manipulate numbers, it may be easier to "Edit IL instructions", rather than "Edit Method".

The Get IL Window

Let's take a closer look at the "Edit IL" window.

The IL stands for Intermediate Language. Since we are using .NET and C#, it's actually fully called CIL, or *Common Intermediate Language.* It's common because all .NET languages compile into it, regardless if it's C# or VB .NET.



The Instructions tab is what shows default, and really the only tab that we are interested in.

And precisely what we are interested in is the OpCode and the Operand. The index just shows the order of the call.

The OpCode is the "machine language" of the computer, and specifies what kind of operation it does.

Operand is the value of that instructions. In some cases, there's OpCodes that do not have an Operand.

Reading IL is a bit counter intuitive.

For example, let's look at Index 1 and 2:

IL View:

```
1  0001  ldc.i4.s    0x20
2  0003  call        instance void Bag::RU(int32)
```

Code View:

```
this.RU(32);
```

This is showing 0x20 (Decimal 32), and a method call to Bag::RU.

From the call line, we see that the Bag::RU takes an int32 value. The line above, we see ldc.i4.s, which is an OpCode for int32. When we look at the IL, we set, or "push" the value we want on the stack first, then we tell it which method to call.

In order words, when we are speaking the IL language, we are saying this:

> "Push 0x20 into position so that we can pass it to the Bag::RU method".

When we are speaking the C# language, we are saying this:

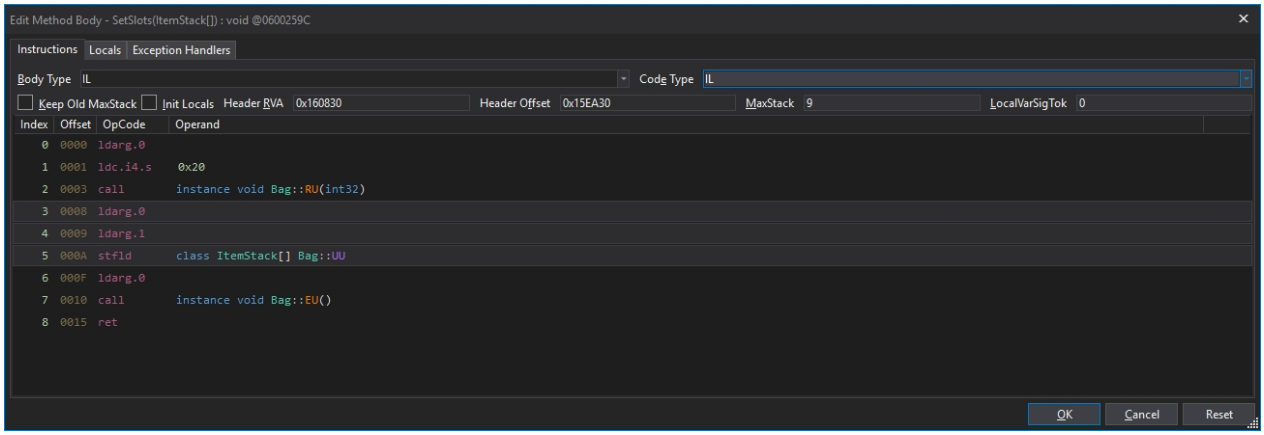
> "Call Bag::RU() and pass in the number 32.

OpCodes Example

For a full list of OpCodes, click here.

Here's a sample table that lists the OpCodes and what value is expected in them.

For reference, here's a method that's displaying our OpCodes



And here's the C# Code to go alone with it:

```
// Token: 0x0600259C RID: 9628 RVA: 0x00160830 File Offset: 0x0015EA30
public void SetSlots(ItemStack[] _slots)
{
    this.RU(32);
    this.UU = _slots;
    this.EU();
}
```

OpCode	Operand Type	Description
ldarg.0	Blank	It's the first argument. In nearly all cases, it translates to "this"
ldc.i4.s	0x20	This "pushes" a 32-bit int into position. In our example, it's showing 32 rather than 0x20. However, it's displaying the decimal value in the code view (32), whereas the IL is hex: 0x20
call	Method call	This 'calls' another method', passing in the operand that was pushed above it. In this case, it's calling Bag::RU(0x20), or Bag::RU(ldc.i4.s)
ldarg.0	Blank	Same as above, it calls "This"
ldarg.1	blank	This the "second" parameter passed to the method. However, it's second to "this", which is passed by default. ldarg.1, in this example, refers to "_slots"
stfld	ItemStack [] Bag::UU	Updates the value in the Operand with the previous instruction. In this case, this.UU is being updated by the _slots value.
ldarg.0	Blank	Again "this"
call	Method Call	Calling another method, in this case: instance void Bag::EU()
ret	Blank	returns. This is a "void" method, so there's nothing to return.

Dealing with De-obfuscation

One of the biggest challenges of using dnSpy, and any of the above tools, is the de-obfuscation. That's the term used for decompiling a DLL, or in our case, the Assembly-CSharp.dll. Sometimes we see the code as it was originally written, but other times it looks and behaves completely differently.

Here's an example of the Bag class, in the Clear Method.

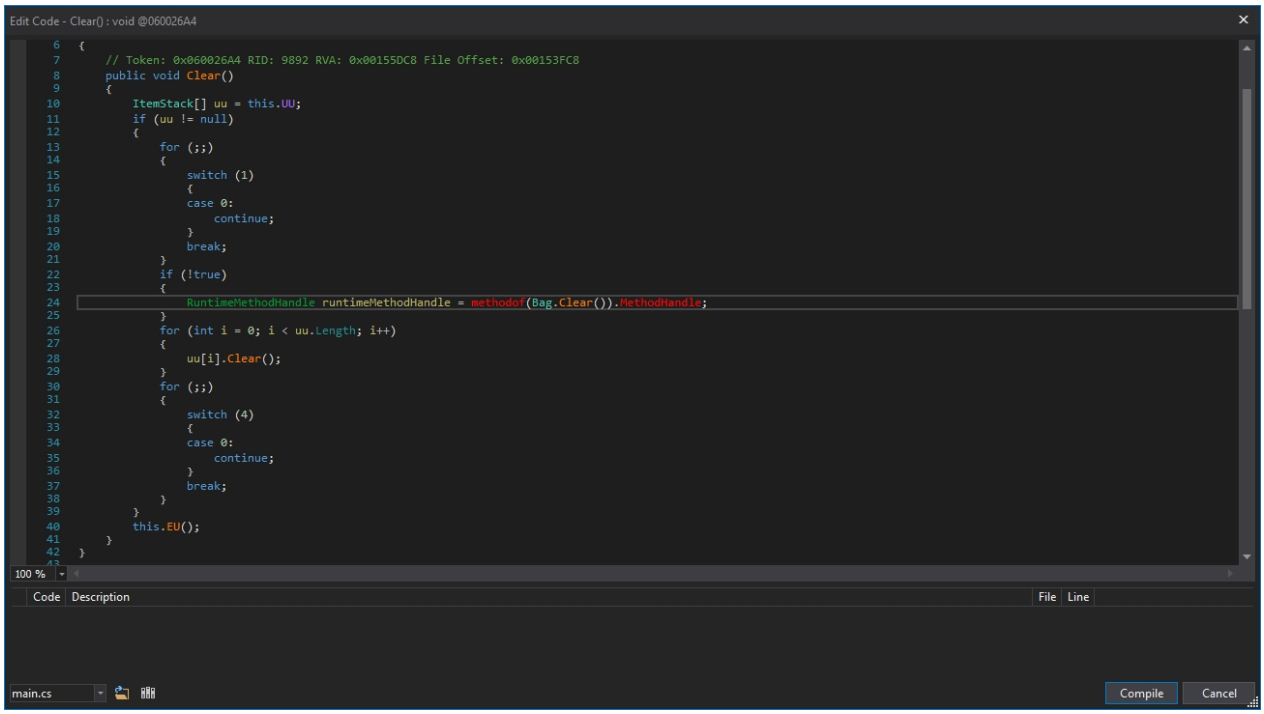
```
// Token: 0x060026A4 RID: 9892 RVA: 0x00155DC8 File Offset: 0x00153FC8
public void Clear()
{
    ItemStack[] uu = this.UU;
    if (uu != null)
    {
        for (;;)
        {
            switch (1)
            {
            case 0:
                continue;
            }
            break;
        }
        if (!true)
        {
            RuntimeMethodHandle runtimeMethodHandle = methodof(Bag.Clear()).MethodHandle;
        }
        for (int i = 0; i < uu.Length; i++)
        {
            uu[i].Clear();
        }
        for (;;)
        {
            switch (4)
            {
            case 0:
                continue;
            }
            break;
        }
    }
    this.EU();
}
```

It's a Clear method in the Bag Class.Looks a bit messy, and non-sensical.

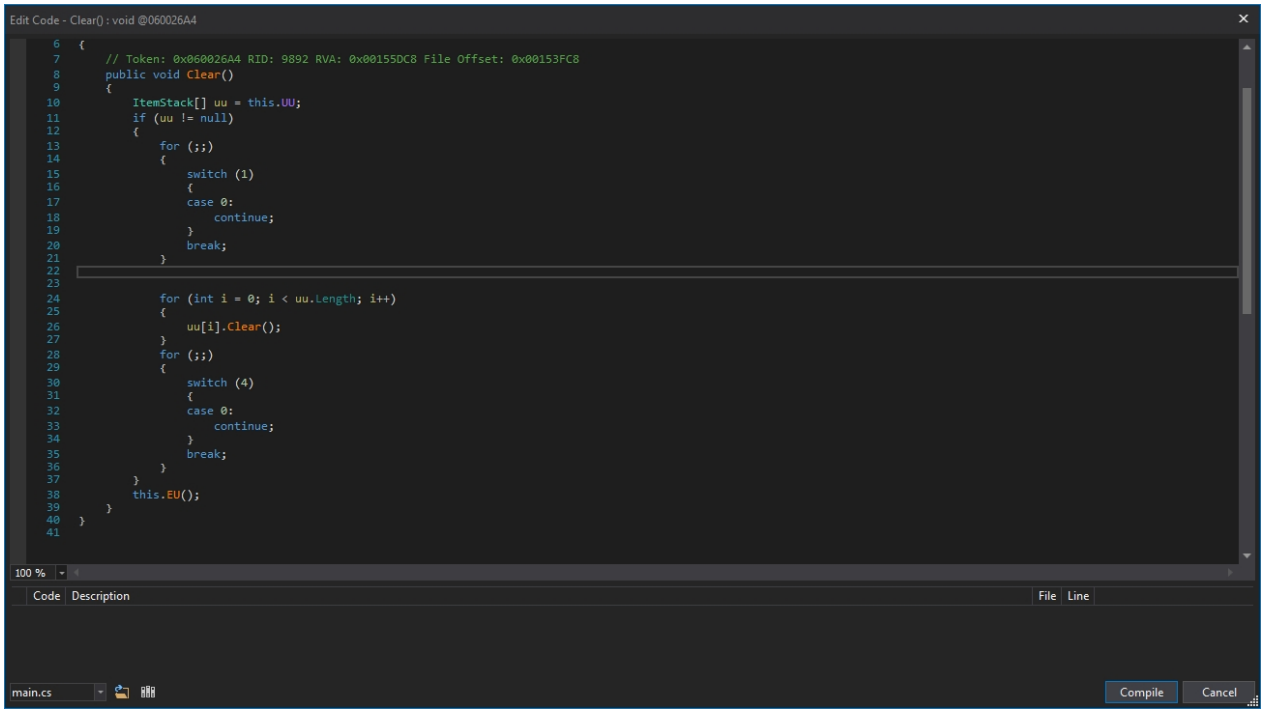
```
    if (!true)
    {
        RuntimeMethodHandle runtimeMethodHandle = methodof(Bag.Clear()).MethodHandle;
    }
```

if (! true)? So if True is not true? It's confusing, and you can actually remove it. *It's an artifact of the de-compiling phase.*

Let's right click on the body of the method, and go to "Edit Method (C#)"



Notice the red against methodof? It's because it's not actually valid C #. And it's inside of that !true statement. Let's remove it.



And press Compile. If there's any errors, fix those up, until you can compile.

After your compile, the code will look similar to this:

```
// Token: 0x060026A4 RID: 9892
public void Clear()
{
    ItemStack[] uu = this.UU;
    if (uu != null)
    {
        for (int i = 0; i < uu.Length; i++)
        {
            uu[i].Clear();
        }
    }
    this.EU();
}
```

Looks better? And probably a lot closer to the original code.

Visual Studio

SDX And Mod Standards

There are a few recommended guide lines that have been set forth by the community in order to raise and maintain a high bar of quality mods.

mod.xml

The mod.xml contains information about the mod. Not only does it show up in the SDX Launcher, but will also show up in some of the support tools we are rolling out.

```
<mod>
      <info>
            <!-- Information about the Mod, the author and version
information -->
            <!-- These are displayed in the SDX Launcher -->
            <author>sphereii</author>
            <name>Bigger Back Pack</name>
            <description>Bigger Back Pack SDX Mod 5x9 45</description>
            <mod_version>1.0</mod_version>
            <game_version>16.2</game_version>
            <launcher_version>0.0.0</launcher_version>
            <type>Mod</type>
      </info>
```

```
      <!-- This references any config files that SDX needs to merge into
your files -->
      <config_mods>
            <import file="Config\BiggerBackPack.xml" />
    </config_mods>

</mod>>
```

Author: The one who wrote or put together the mod
Name: The unique name for the Mod. This shows up in the SDX Launcher.
Description: Short description of the Mod
Mod Version: The version number here is in a Major.Minor example.

If your mod is relatively small, it is recommended to use a two digit version system.
Major: Complete overhaul, such as going from Alpha 15 to Alpha 16.
Minor: Minor tweaks, such as going from Alpha 16.2 or Alpha 16.3, and for bug fixes as well.

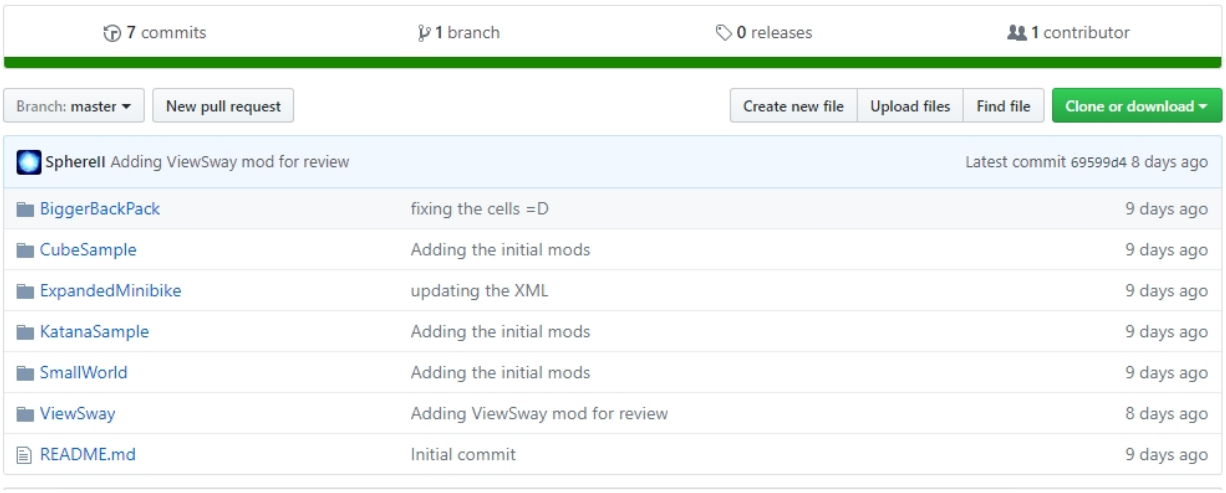
For larger, more comphrensive Mods, or if you would like more fine tuning of a version, you may opt to do a Major.Minor.Defect.
Major: Complete overhaul, such as going from Alpha 15 to Alpha 16.
Minor: Minor Tweaks, such as going from Alpha 16.2 to Alpha 16.3
Defect: Bug fixes

game_version: The latest version of the Game which the mod was tested and found to be working.
launcher_version: Which version of SDX Launcher it is compatible. (Optional)
type: The type of mod package you have:

Type Value	Description
Mod	A full integrated Mod, maybe include XML, Assets and code.
Sample	A small mod used for demonstration, as opposed to a real mod.
AssetBundle	A mod with just a Unity Asset Bundle, to introduce new textures and sounds.
XML	A mod with only XML changes
QoL	Quality of Life Improvement. Does not add new items or recipes, but rather just makes the game run better.
Icons	A mod that contains just Icons
Overhaul	A Complete Overhaul Mod, includes many assets and changes.
Expansion	A Complete Expansion; adding new recipes items, but keeping with the spirit of Vanilla

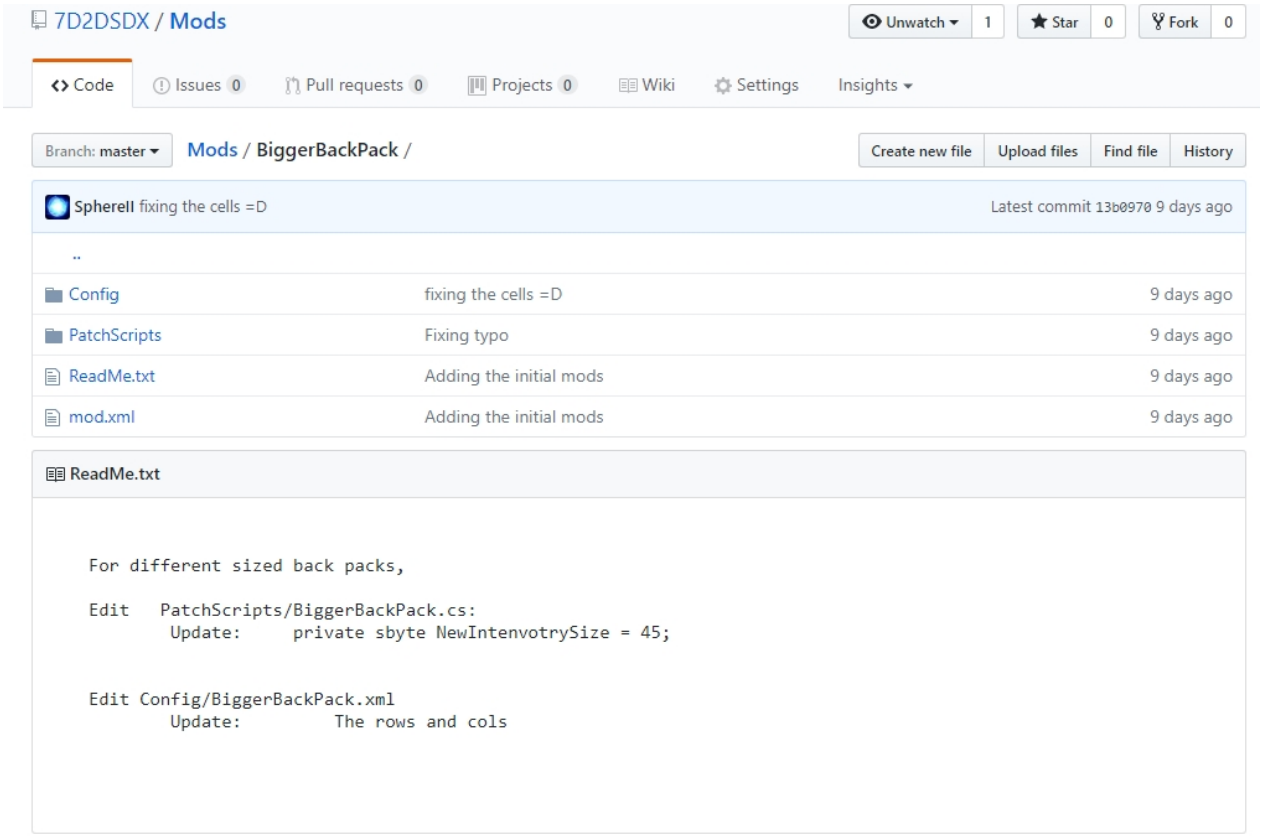
Mod Structure

Here's an example of a Mod Folder structure.



This folder structure is meant to be downloaded, and extracted to your SDX0.7.1\Target\7DaysToDie\Mods folder

In the BiggerBackPack folder, we see this:



Note the ReadMe.txt is automatically displayed on the website, so be sure to add information to it.

About 7D2D SDX Project

The 7D2D SDX Project was created to help those who are struggling with working with SDX, and those who were too unsure of how to even get started.

We came together and joined to create a comprehensive tutorial system to help guide you through your SDX

journey, running from simple SDX instrumentation up to advanced topics, like C# Scripting and creating completely new items in Unity.

We leveraged the following Tools to make this tutorial a success:

- GitHub.com: Free hosting, and free web service are a huge boon to this project. However, the key to this site is the ability to fork the project, and allow other people to continue the work.

- HelpNDoc: The software was used to create this tutorial system.

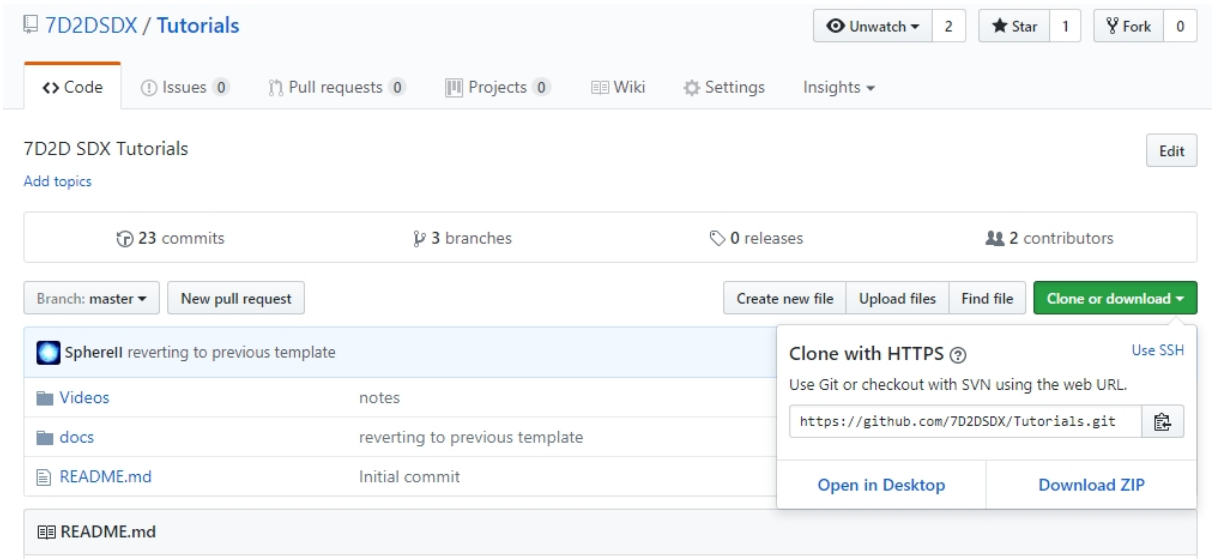
- Visual Studio Community 2017: A development environment to help us build our mods

Updating this Documentation

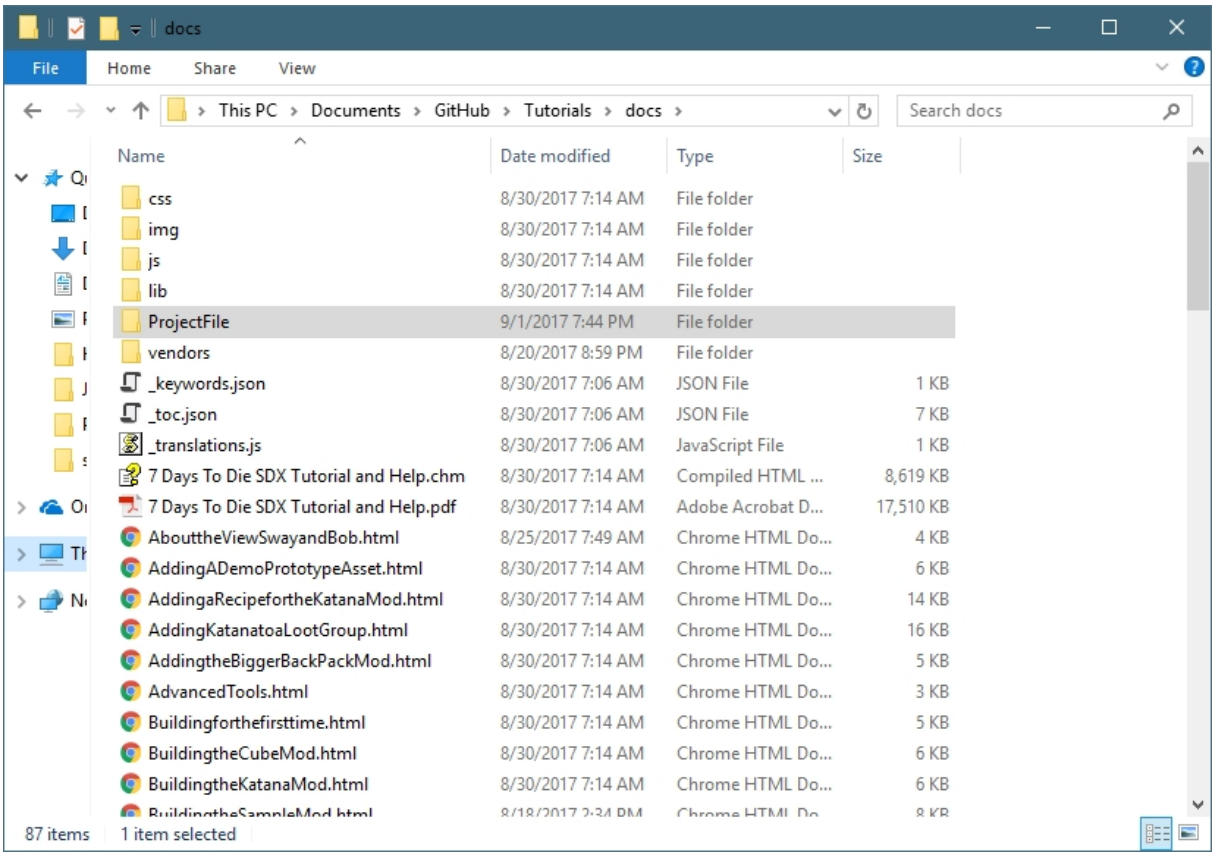
This document was generated by HelpNDoc Personal Edition 5.2.

To Update this document, and re-host, follow these steps:

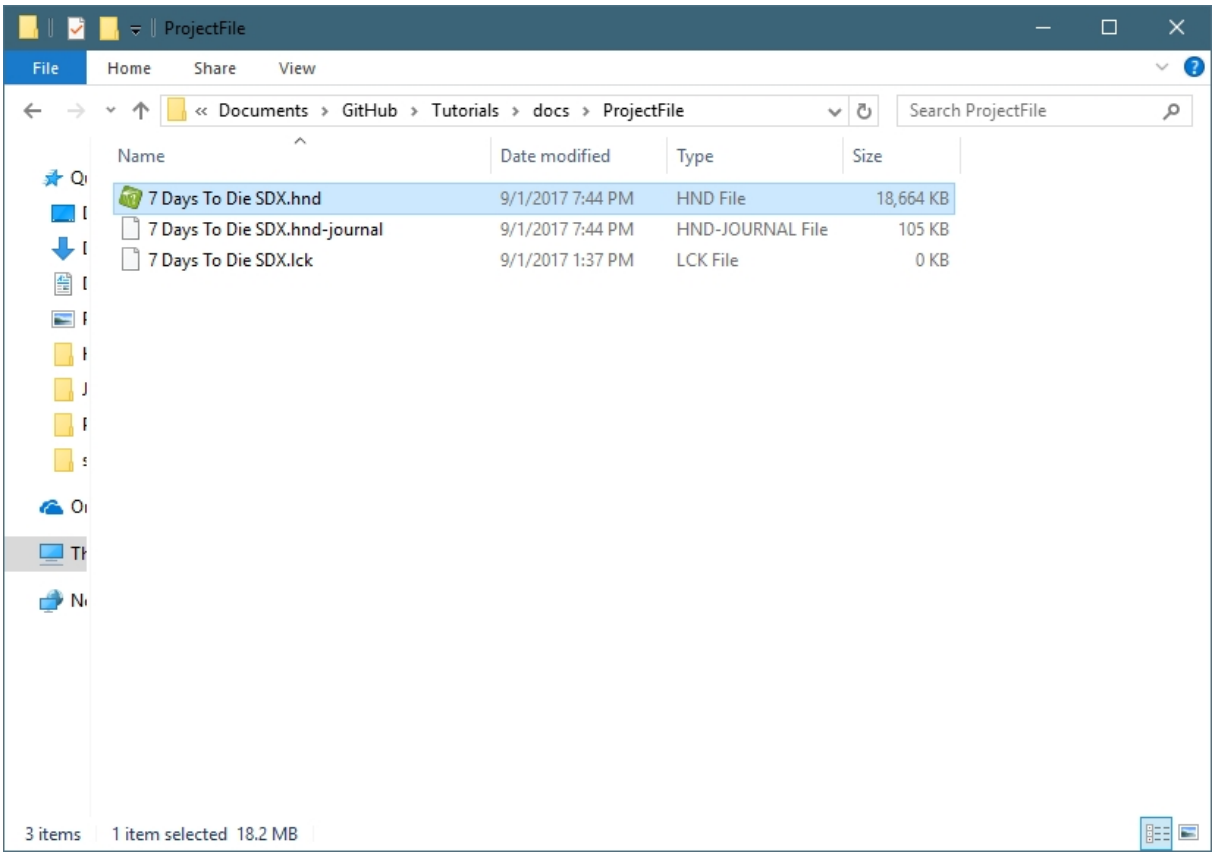
- Clone, Fork, or download the main Repos: https://github.com/7D2DSDX/Tutorials



- Under the docs folder, look for the ProjectFile

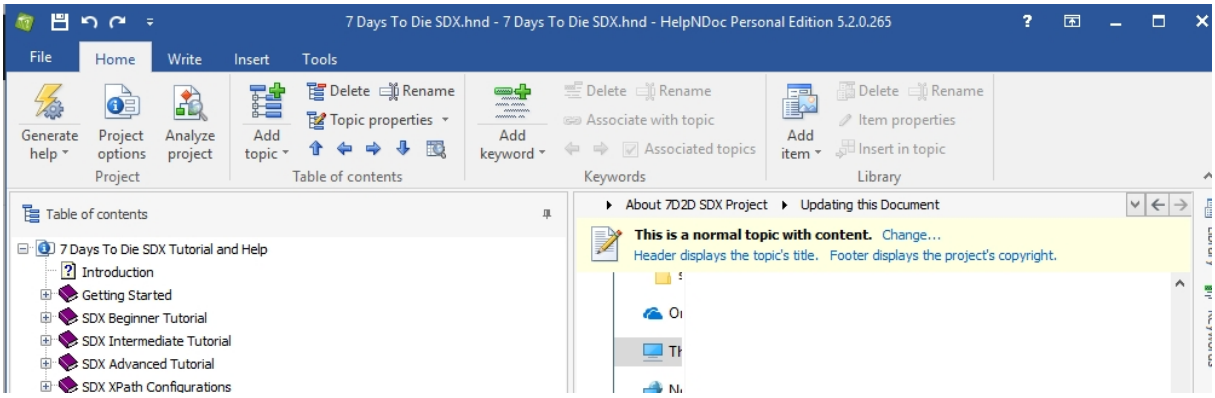


Double click on the Project File called "7 Days To Die SDX.hnd"

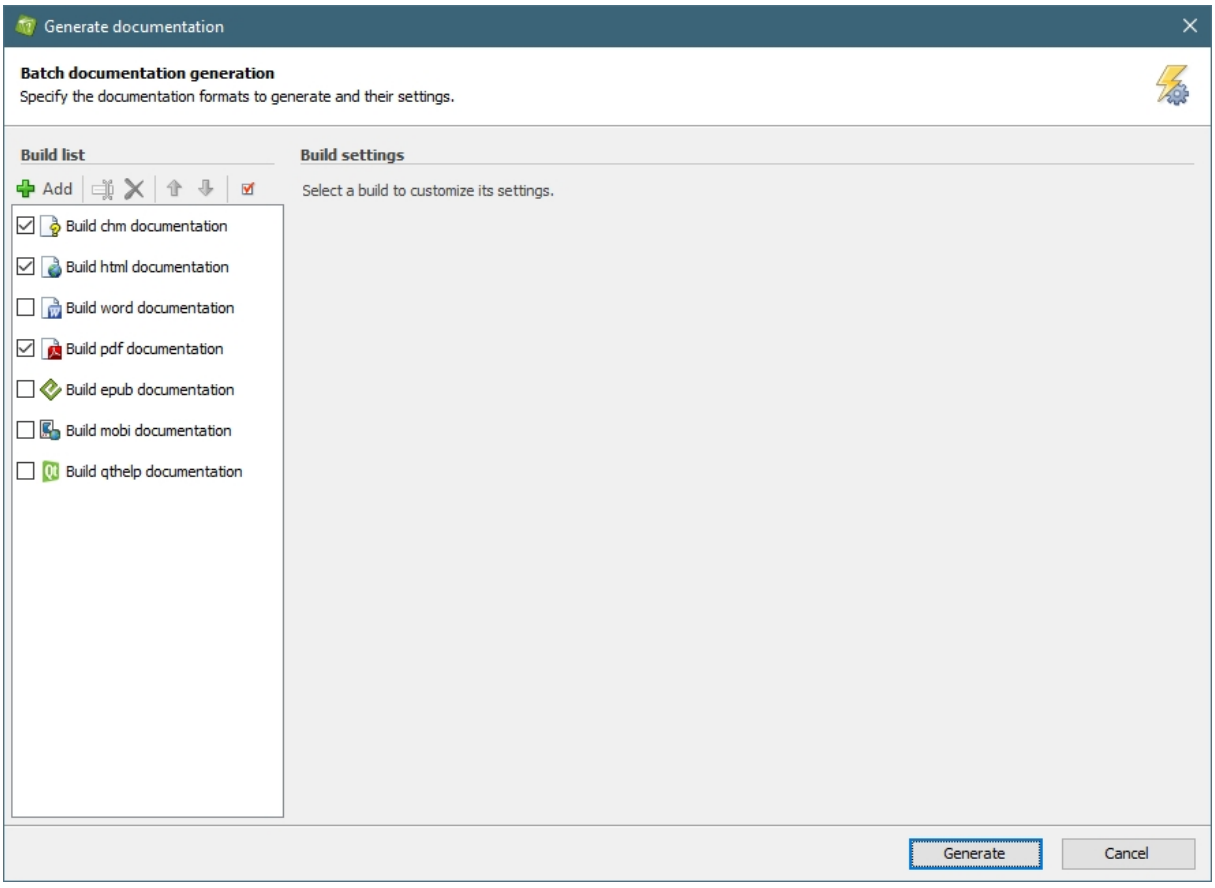


Edit the tutorial as needed.

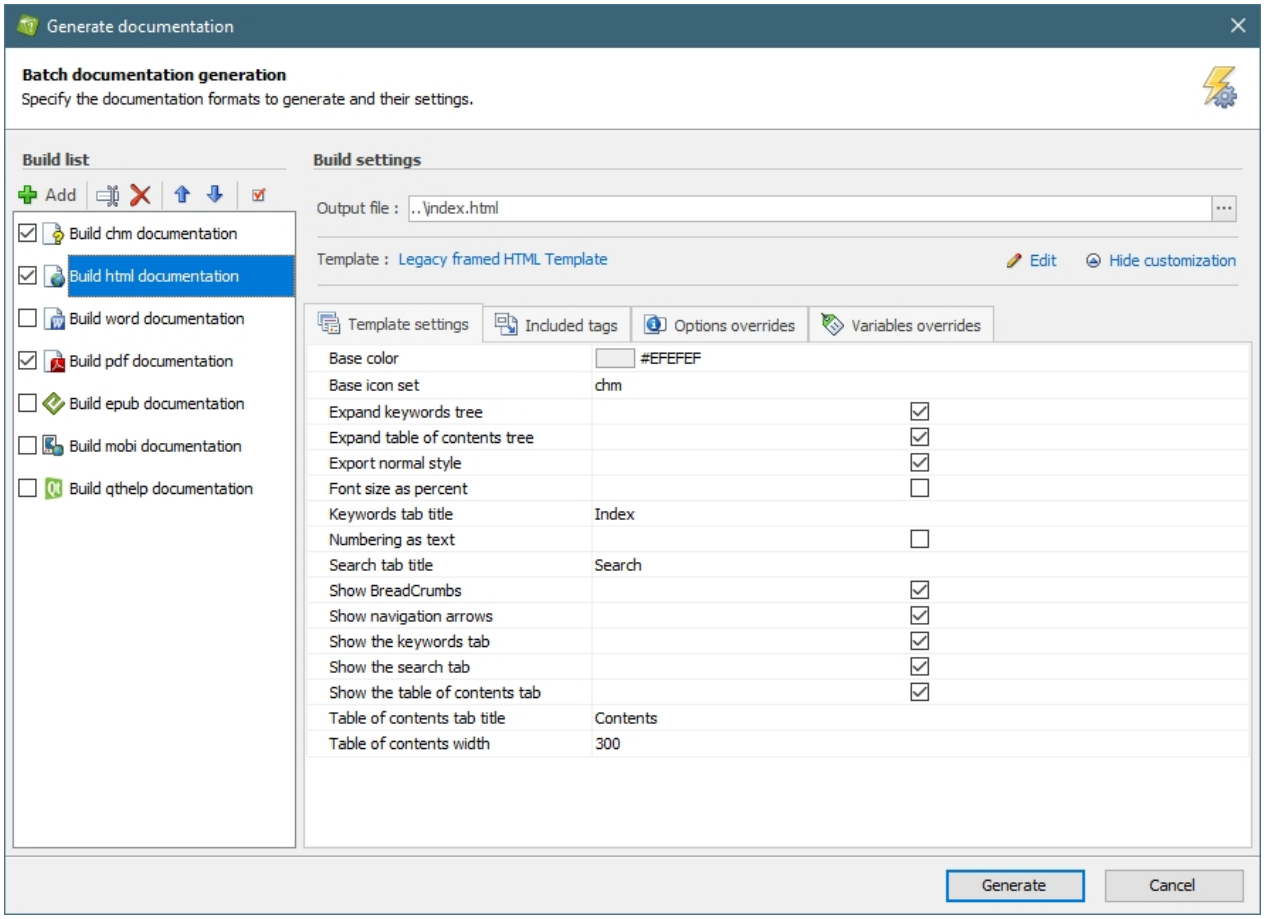
When you are complete, click on the Home tab, then "Generate Help"



By default, the 7 Days To Die SDX Tutorial generates a .chm, a website, and a PDF of its documentation.

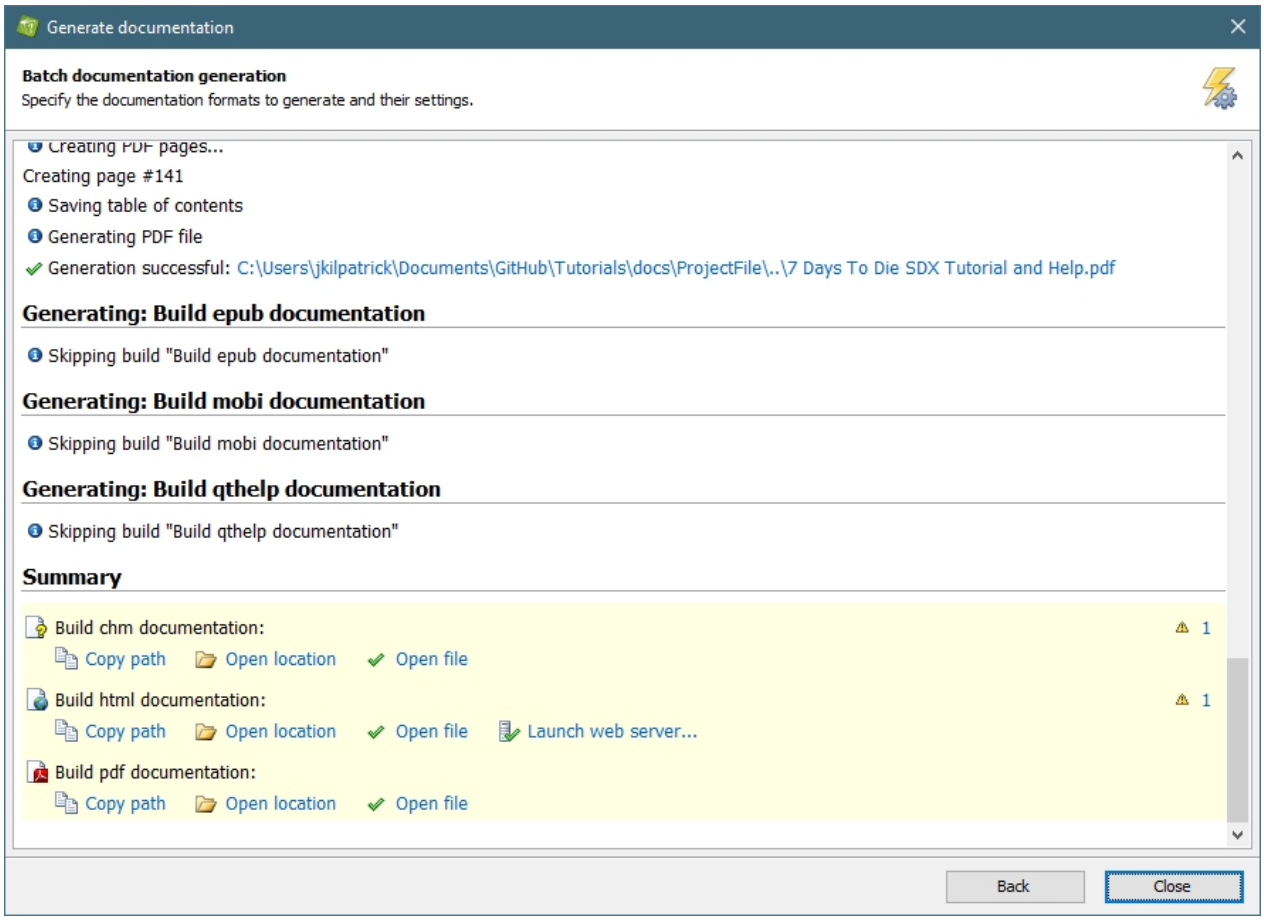


If you click on "Build html documentation", you'll see the only changes from default that we used:



You'll notice the website will get built ..\, which is one folder up from the ProjectFile

Click on Generate. It will take a few minutes to generate.



How Tos

How to download SDX Mods

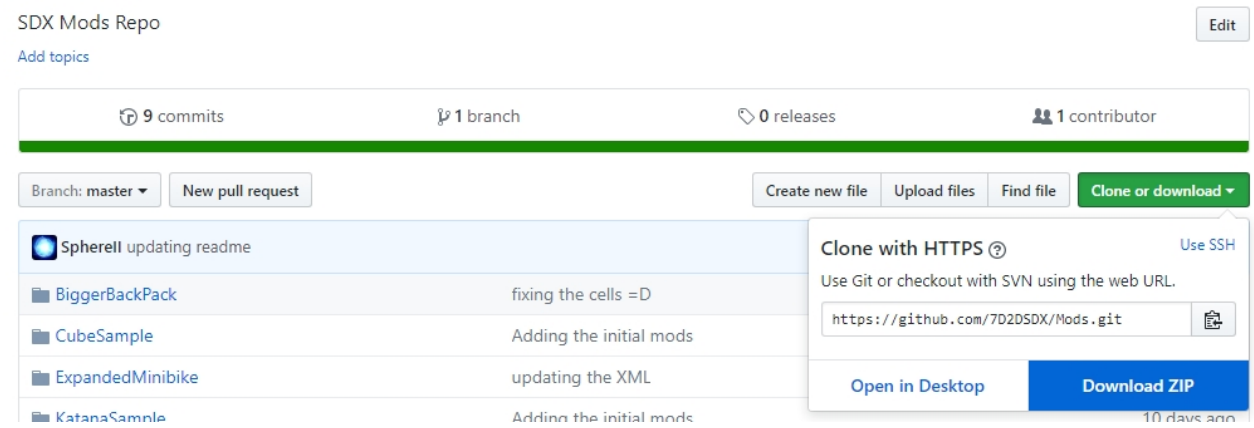
A few Repos have been created where you can download and get started with SDX:

Xyth's Repos

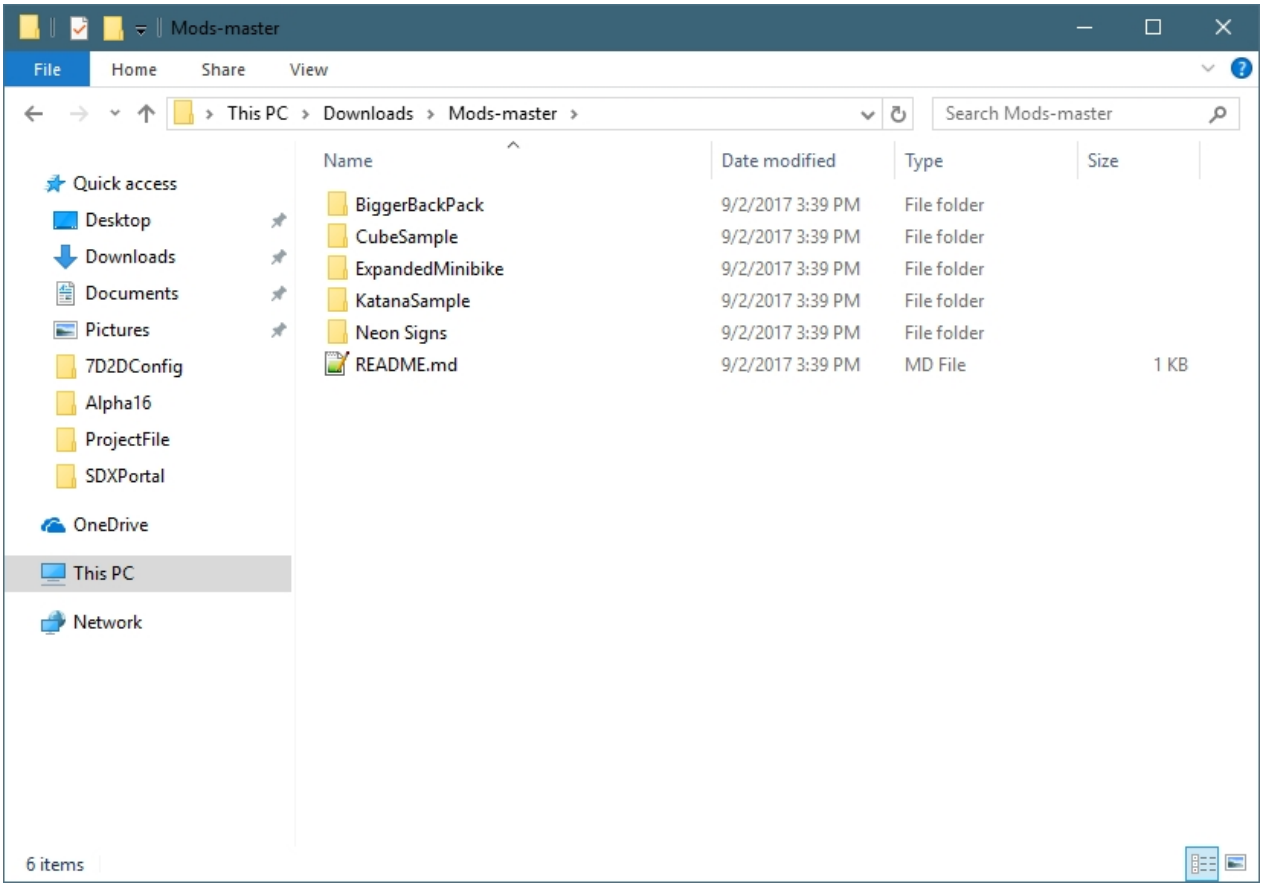
7D2D SDX Mod Repos

To Install:

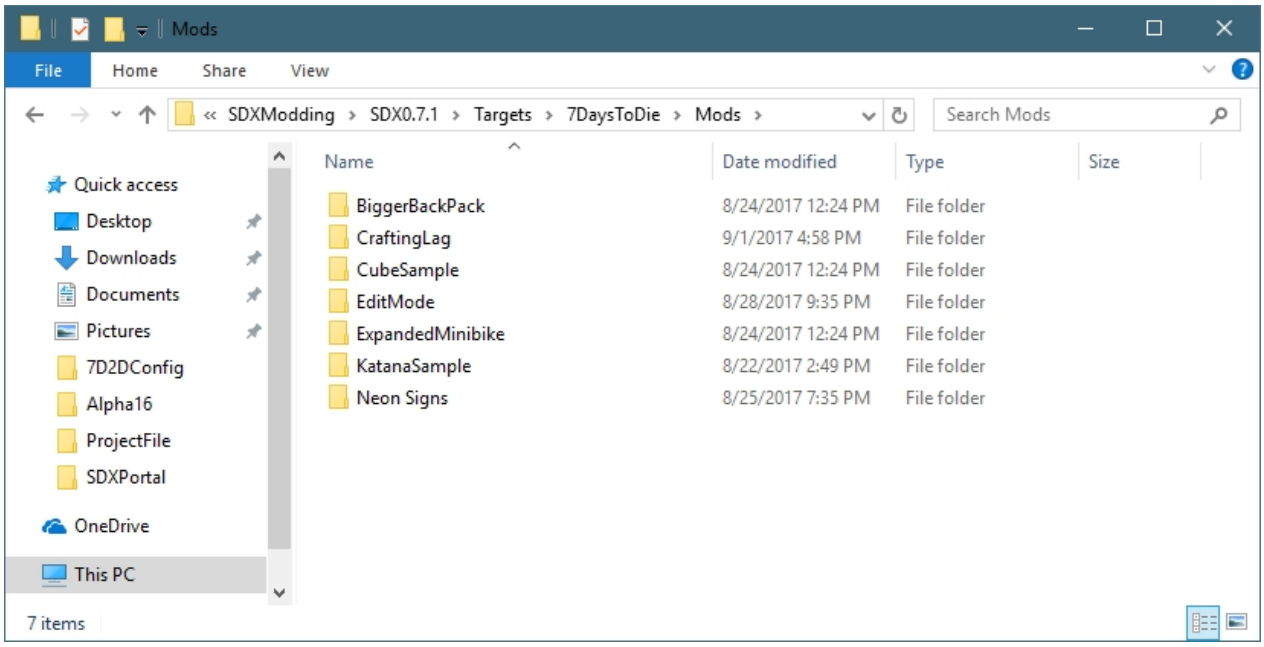
Click on "Clone Or download" button, and click on Download Zip



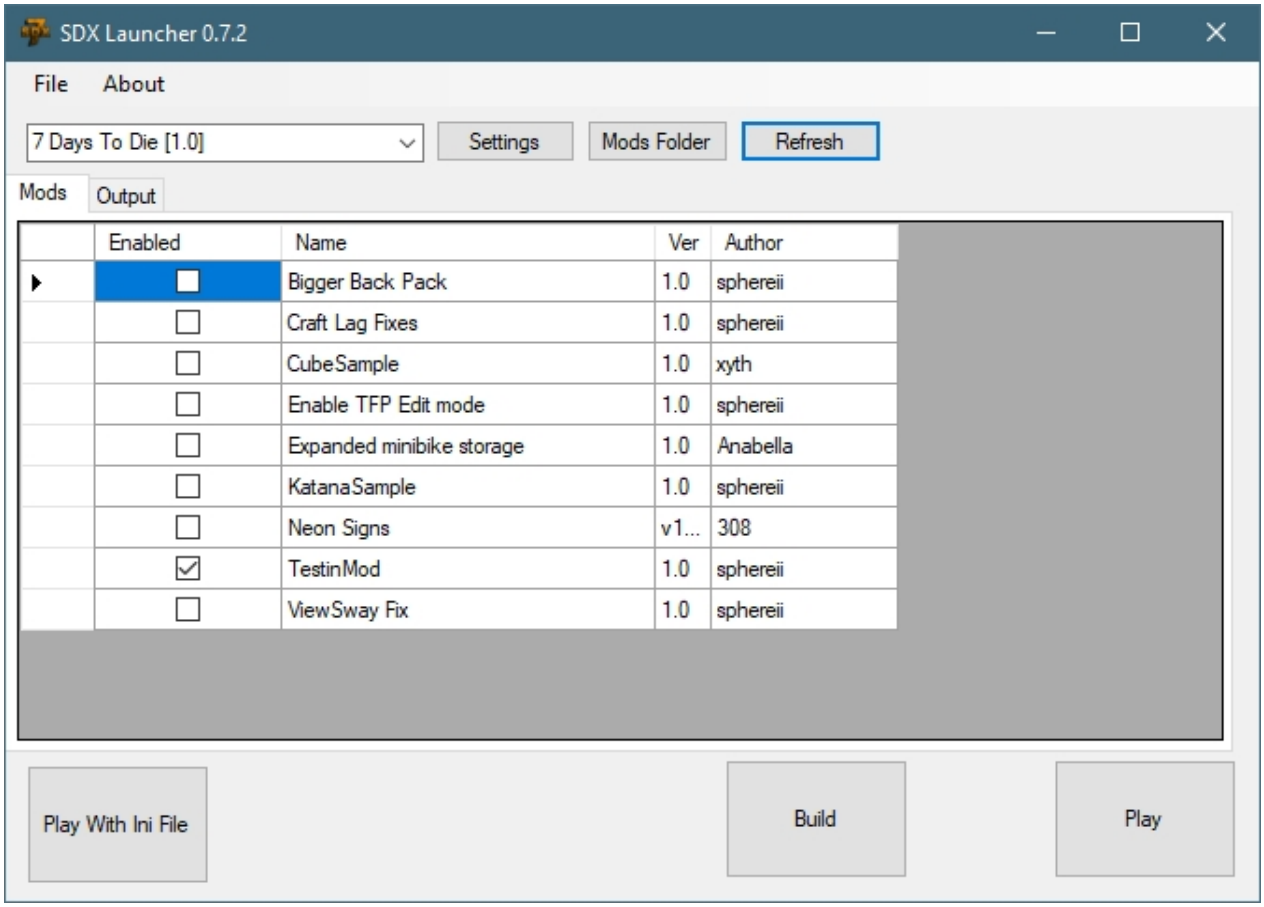
On the downloaded Mods-master.zip file, extract the file



Copy some, or all the mods to your C:\SDXModding\SDX0.7.1\Targets\7DaysToDie\Mods folder



In the SDX Launcher, click on refresh to see the new Mods:



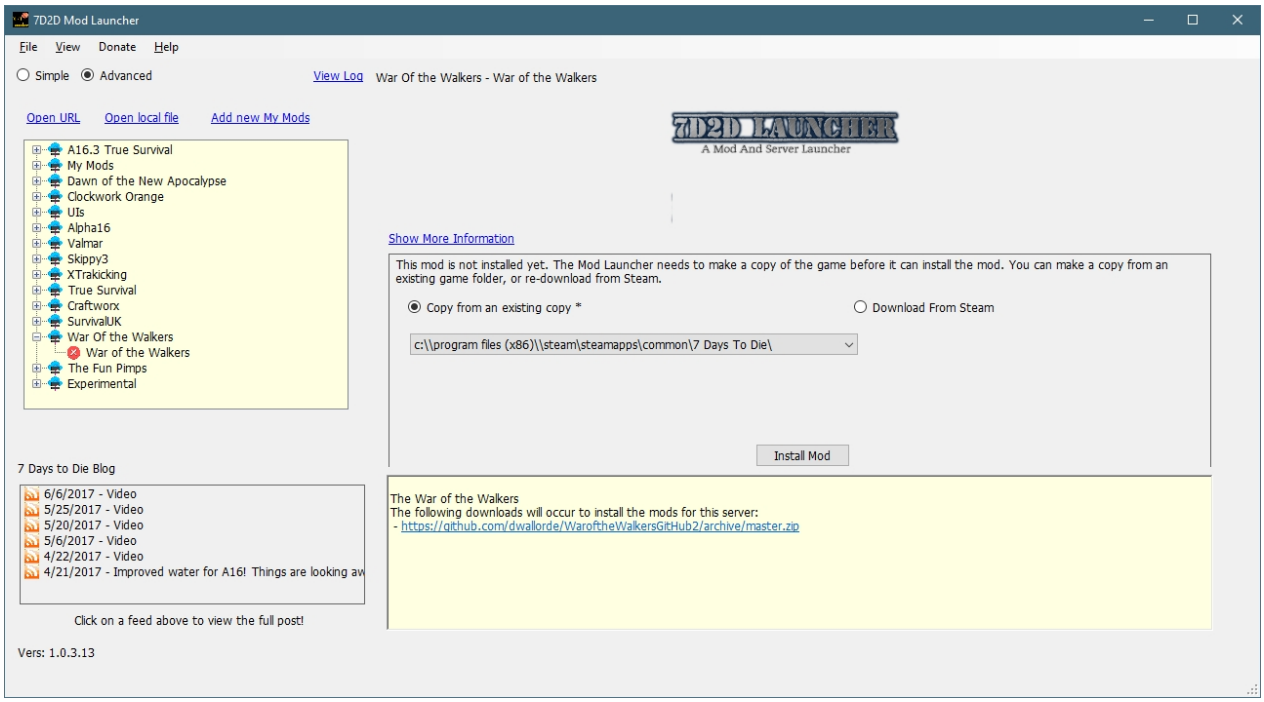
How to manipulate War of the Walkers

Let's see what we can do with the tool chain, and War of the Walkers.

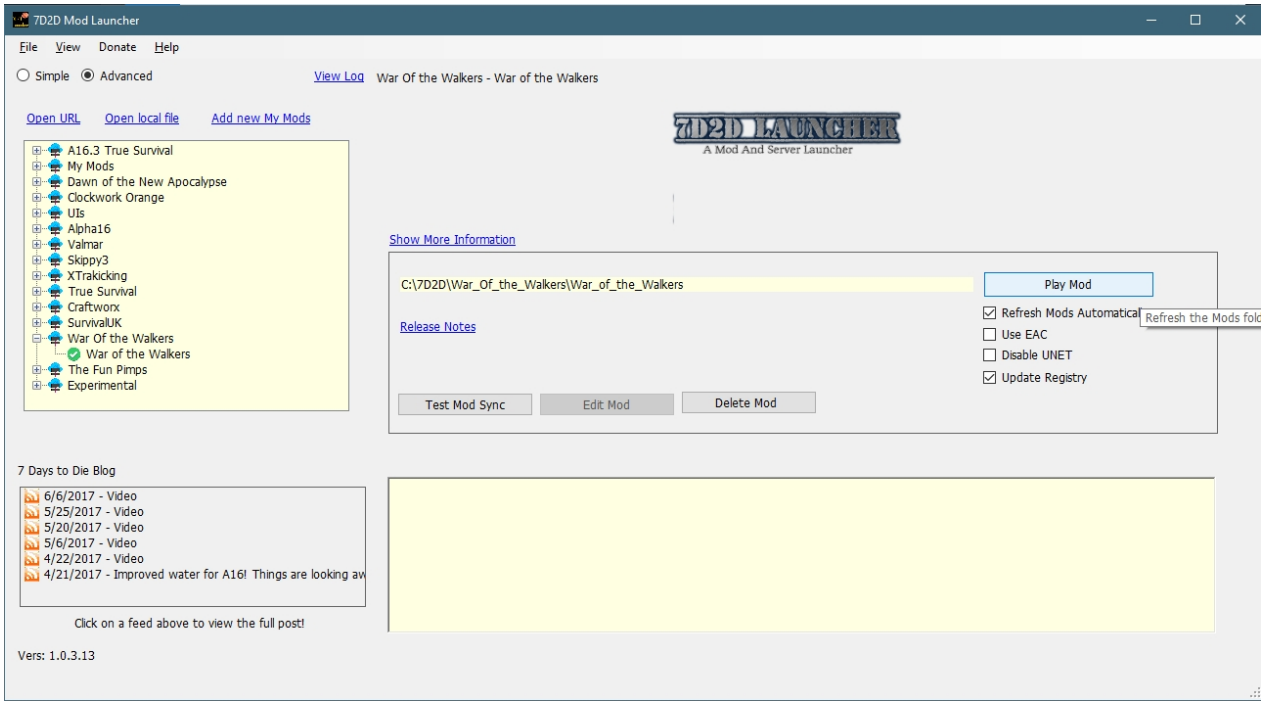
What you'll need:

1) The 7D2D Mod Launcher
2) The SDXModding Kit
3) The SDX Mod Helper Program

Install War of the Walkers, using the 7D2D Mod Launcher



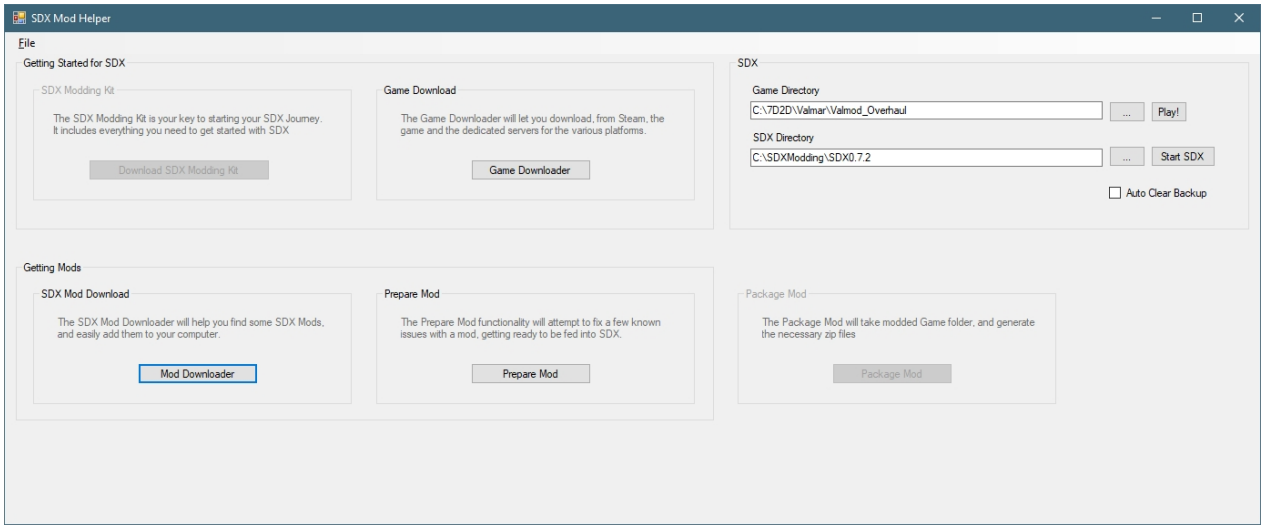
Enter the game once, in order for the Mod Launcher to install the files correctly.



After that, exit the game, and exit the Mod Launcher.

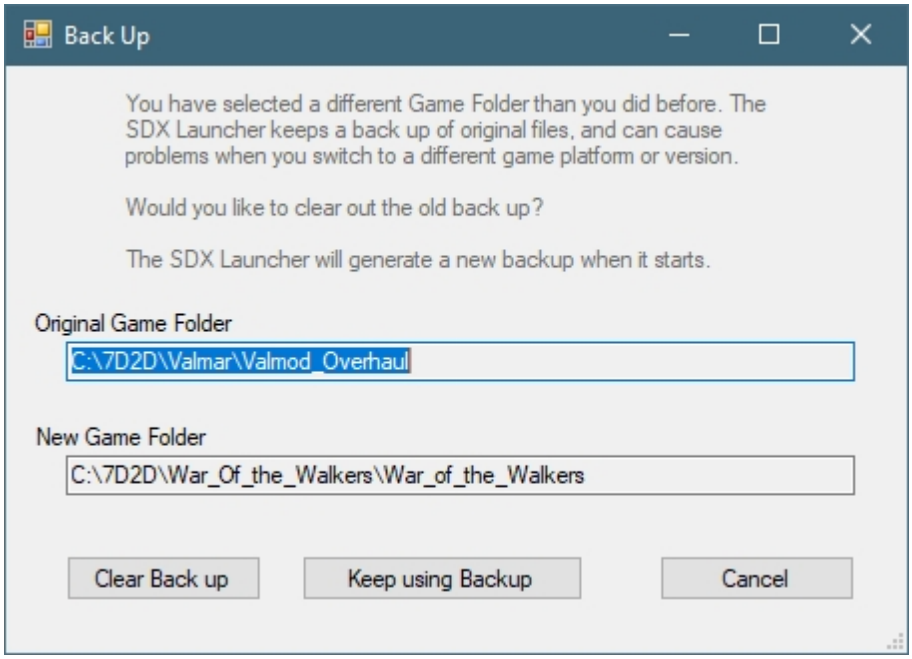
Before you begin the next step, close the SDX Launcher, if you have it open.

Start the SDX Mod Helper.



Click on the  by the Game Directory, and pick the War of the Walker's Mod

If you've previously ran SDX Launcher, then it would have created a back up file. Since we are using a new mod, we'll want to remove the old back up.



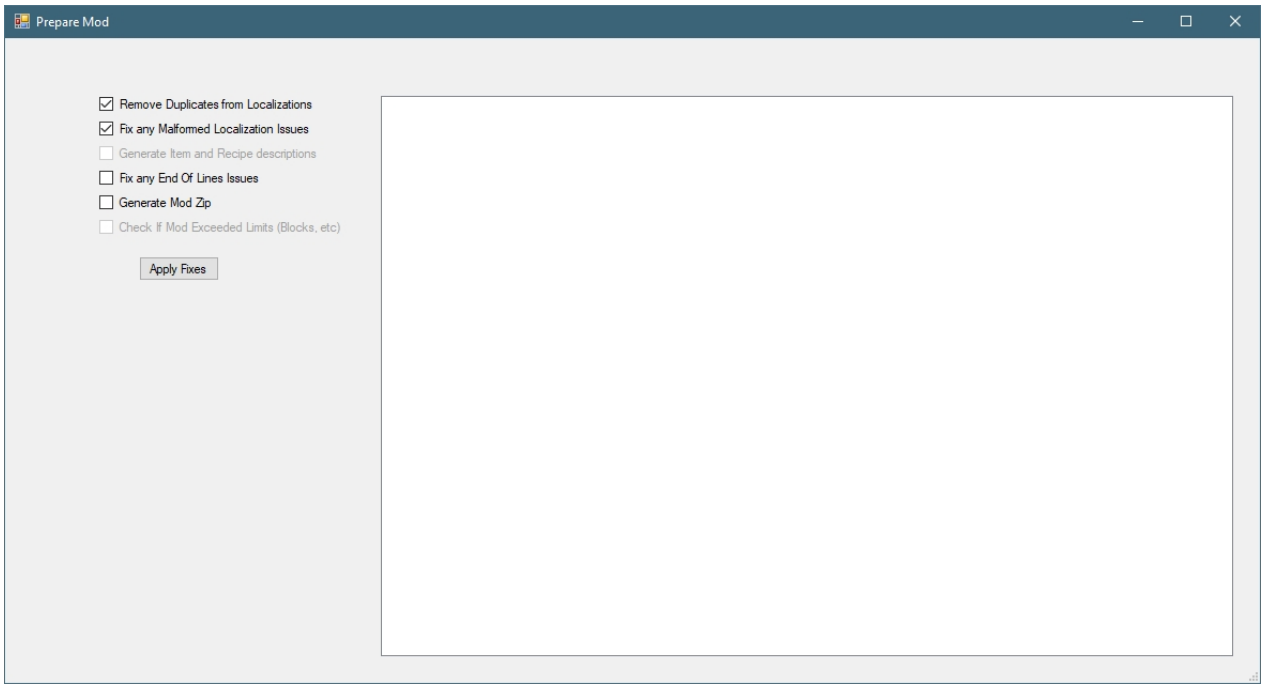
When you change the Game Dir, it'll prompt you to remove your old Back up. The SDX Launcher keeps a lock on the Back up folder, which is why we closed it at the start of this tutorial.

What's in the Backup? When SDX Launcher runs initially, it makes a back up of the original DLL and the original XML. However, when you change your Game Dir, you'll want to create a new back up, rather than use the old one.

Click on Clear Back up to remove it.

The next step we want to take, is to do a health check on the Mod. Click on the Prepare Mod button.

Here, you'll find a few options that attempt to scan and check for problems that SDX may have with the mod.

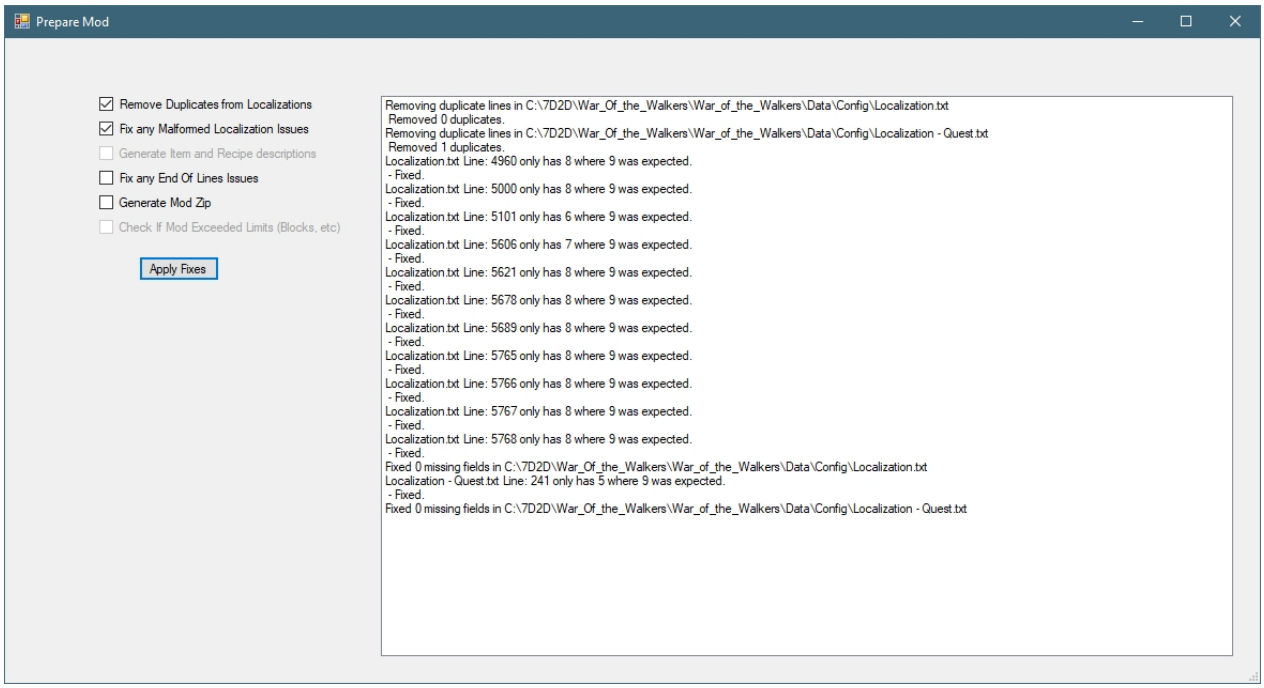


In the above example, we are just going to check Remvoe Duplicates From Localizations, and Fix any Malformed Localiations.

Remove Duplicates From Localizations: This will scan for duplicate lines in the localization, and removes them.

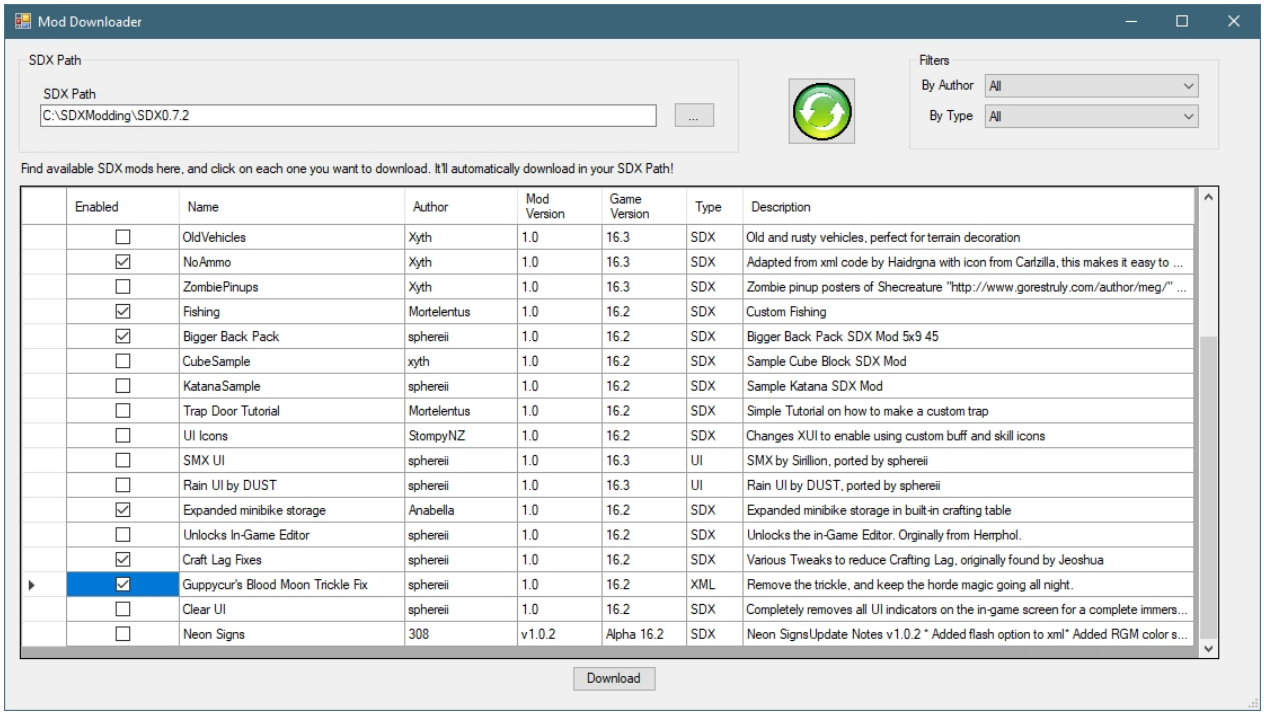
Fix any Malformed Localization Issue This checks to see if each line of the localization files has the right amount of fields. It will fix some of those issues, such as padding in missing commas. For other issues, you'll need to manually fix them.

Click on Apply Fixes.



Great! We fixed a few issues detected. You can close the Prepare Mod window now.

Click on the Mod Downloader in the SDX Mod Helper.



Enabled	Name	Author	Mod Version	Game Version	Type	Description
☐	OldVehicles	Xyth	1.0	16.3	SDX	Old and rusty vehicles, perfect for terrain decoration
☑	NoAmmo	Xyth	1.0	16.3	SDX	Adapted from xml code by Haidrgna with icon from Carlzilla, this makes it easy to ...
☐	ZombiePinups	Xyth	1.0	16.3	SDX	Zombie pinup posters of Shecreature "http://www.gorestruly.com/author/meg/" ...
☑	Fishing	Mortelentus	1.0	16.2	SDX	Custom Fishing
☑	Bigger Back Pack	sphereii	1.0	16.2	SDX	Bigger Back Pack SDX Mod 5x9 45
☐	CubeSample	xyth	1.0	16.2	SDX	Sample Cube Block SDX Mod
☐	KatanaSample	sphereii	1.0	16.2	SDX	Sample Katana SDX Mod
☐	Trap Door Tutorial	Mortelentus	1.0	16.2	SDX	Simple Tutorial on how to make a custom trap
☐	UI Icons	StompyNZ	1.0	16.2	SDX	Changes XUI to enable using custom buff and skill icons
☐	SMX UI	sphereii	1.0	16.3	UI	SMX by Sirillion, ported by sphereii
☐	Rain UI by DUST	sphereii	1.0	16.3	UI	Rain UI by DUST, ported by sphereii
☑	Expanded minibike storage	Anabella	1.0	16.2	SDX	Expanded minibike storage in built-in crafting table
☐	Unlocks In-Game Editor	sphereii	1.0	16.2	SDX	Unlocks the in-Game Editor. Orginally from Herphol.
☑	Craft Lag Fixes	sphereii	1.0	16.2	SDX	Various Tweaks to reduce Crafting Lag, originally found by Jeoshua
☑	Guppycur's Blood Moon Trickle Fix	sphereii	1.0	16.2	XML	Remove the trickle, and keep the horde magic going all night.
☐	Clear UI	sphereii	1.0	16.2	SDX	Completely removes all UI indicators on the in-game screen for a complete immers...
☐	Neon Signs	308	v1.0.2	Alpha 16.2	SDX	Neon SignsUpdate Notes v1.0.2 * Added flash option to xml* Added RGM color s...

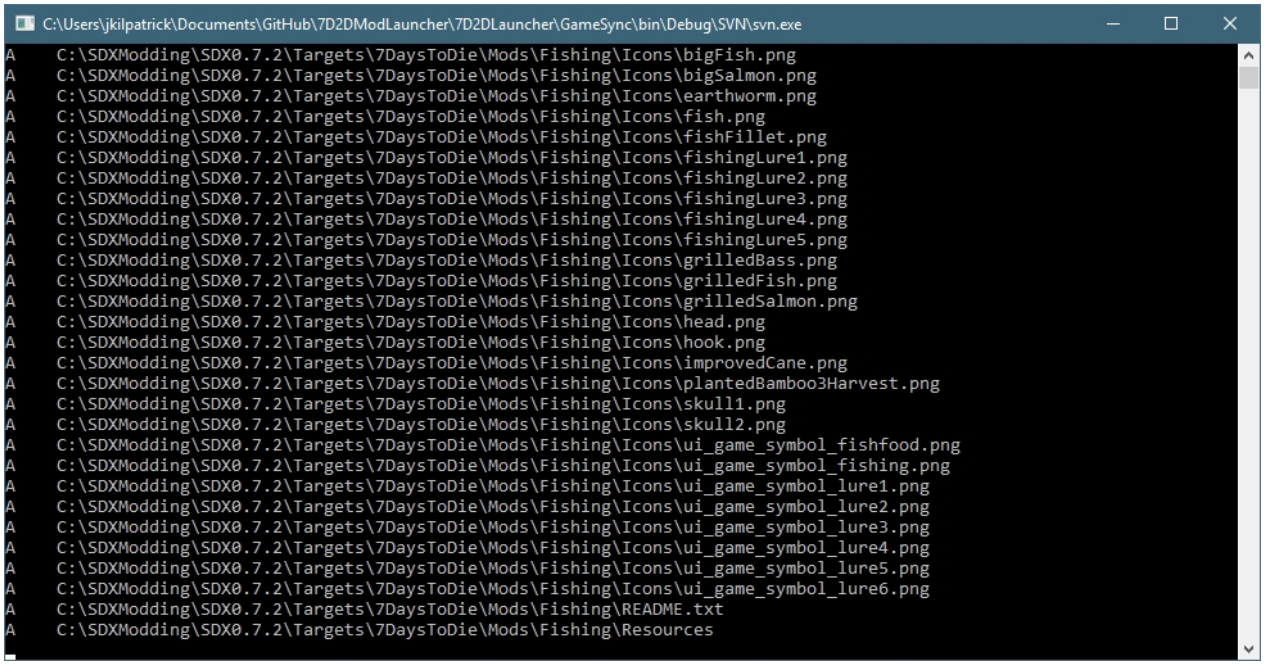
War of the Walkers is a great mod. Let's see if we can extend it just a bit using SDX.

In this example, we are going to add in:

- NoAmmo: Xyth's SDX port of Haidrgna's No Ammo mod, which allows you to unload your ammo from your guns, without disassembly
- Fishing: Mortelentus' Custom fishing mod
- Bigger Back Pack: sphereii's port of the back pack, which gives you up to 45 slots of extra space
- Expanded minibike storage: sphereii's port of Anabella's minibike expansion, which includes the minibike combine table; Combine objects on the road
- Craft Lag Fixes: sphereii's port of Jeoshua's crafting lag fixes. Reduces the amount of times the recipes are read and displayed
- Guppycur's Blood Moon Trickle Fix: sphereii's port of Guppycur's fix, ensuring your horde night won't get boring.

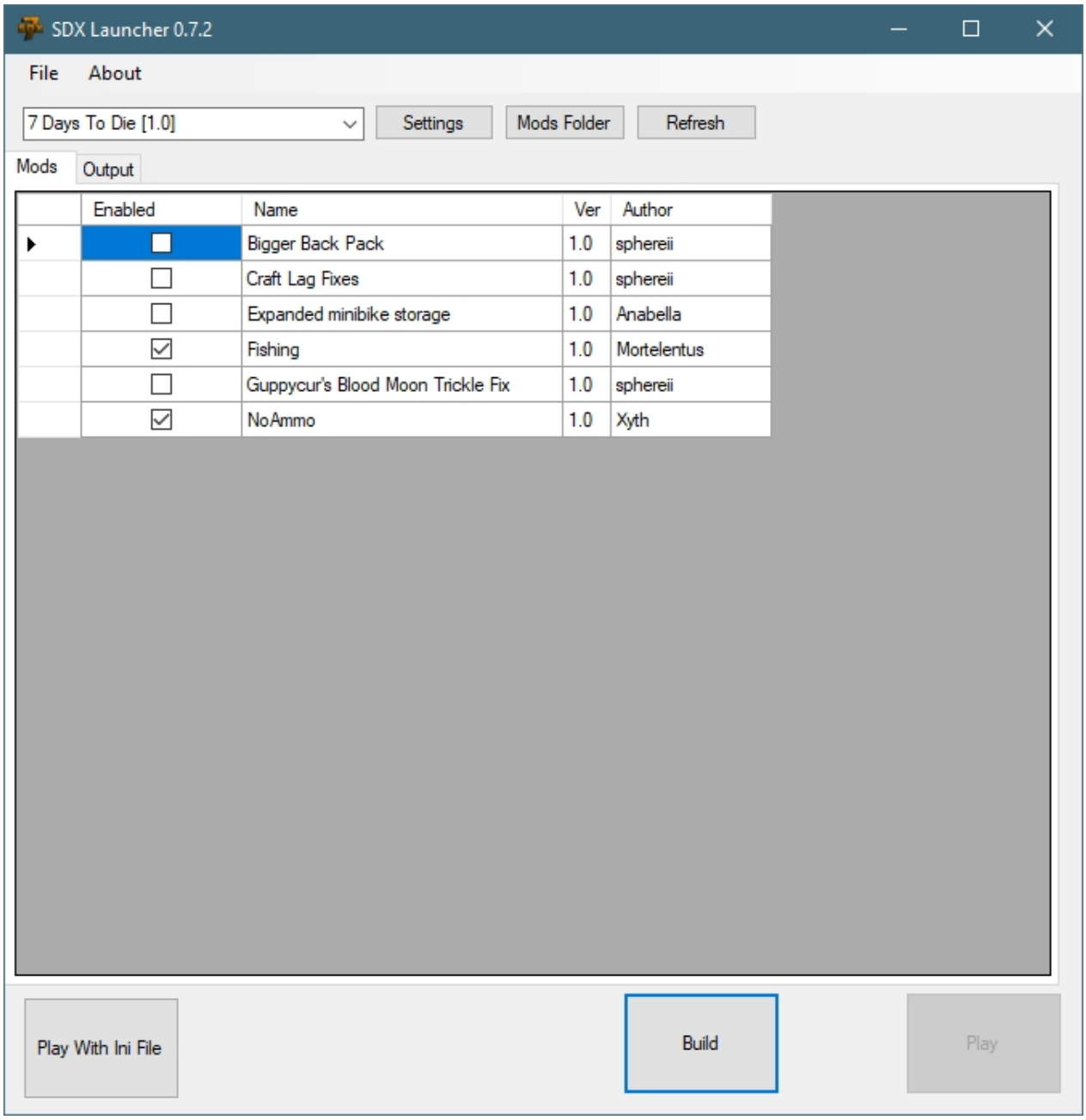
Place a checkmark in each box, and click on the Download button. It'll download the mods into your SDX Folder.

For each Mod being downloaded, you'll see a black pop box:



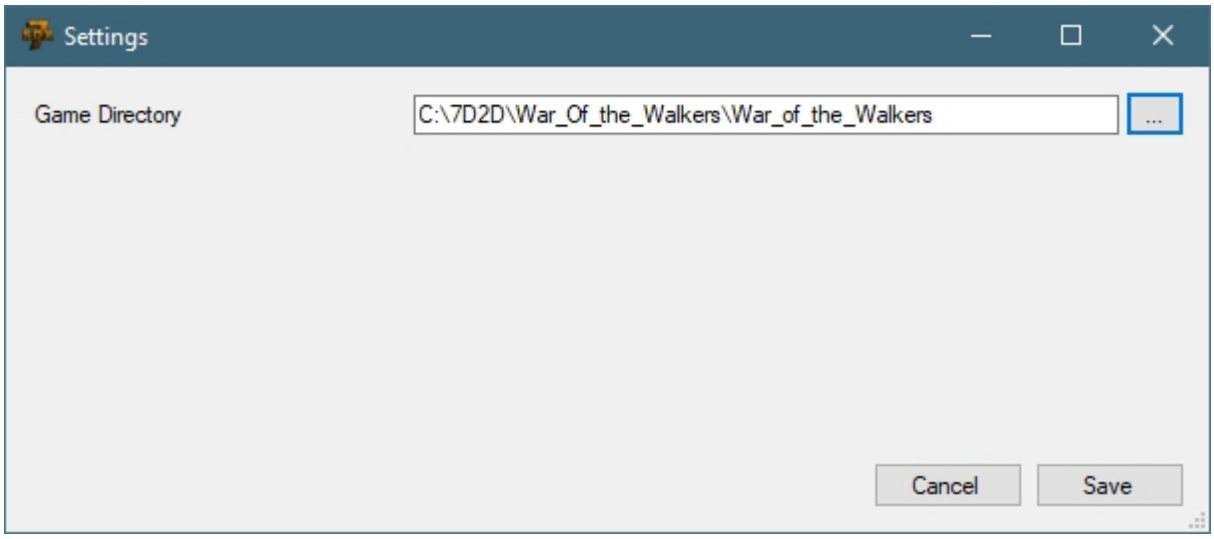
This is normal, and you'll see a pop up for each mod, as it downloads.

After that's complete, you can start the SDX Launcher.



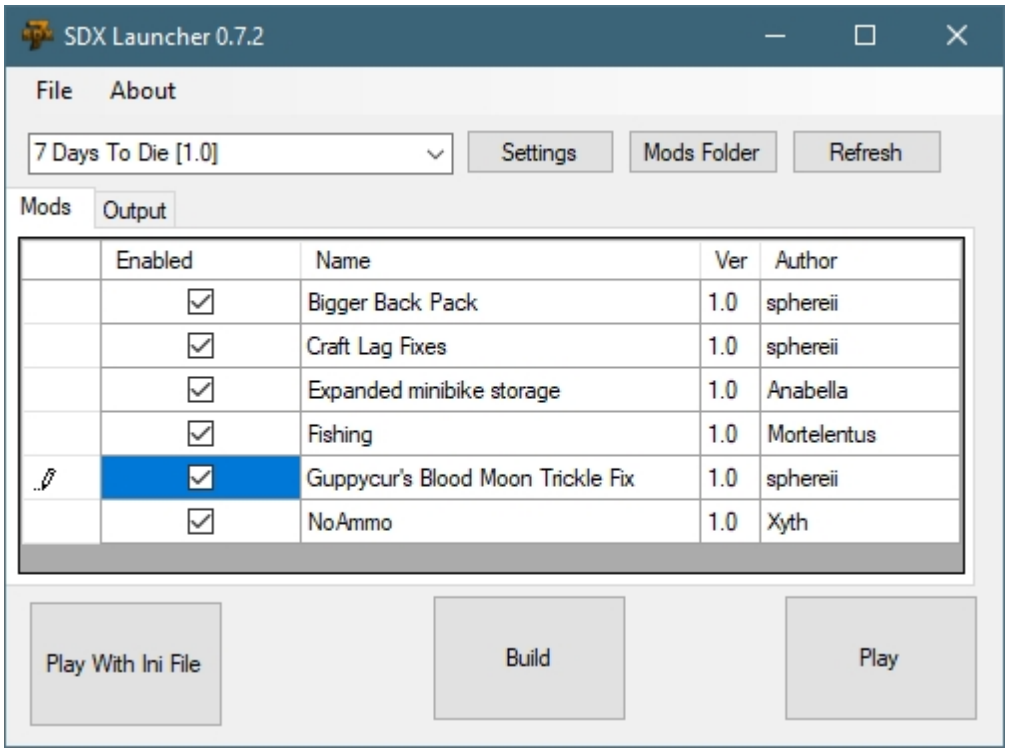
In the SDX Launcher, you'll see the new Mods that you downloaded. You may have more than what's listed above, and that's okay. We'll only enable the ones we want.

Click on the Settings folder, and change the Game Dir to point to the War of the Walkers Mod:



And click on Save

Enabled the Mods we downloaded. If you have extra mods, be sure to uncheck them.



Then, click on Build.

How to set up Visual Studio for SDX Mod Development

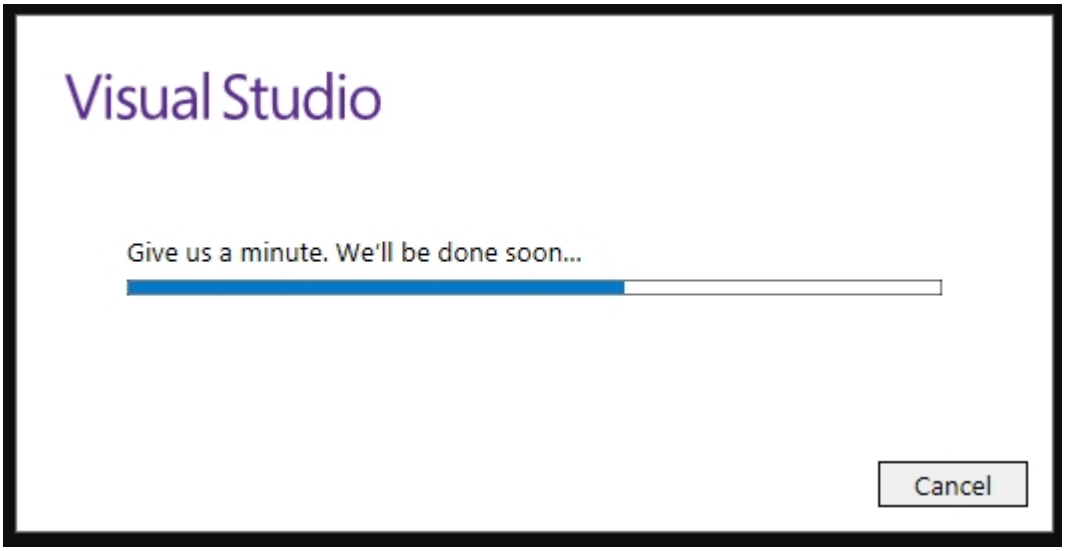
This How To will show you where to get Visual Studio Community 2017, how to install it, and how to get your SDX mods linked in it for development.

Visual Studio allows you to add references to the Assembly-CSharp.dll, and give you Intellisense, which

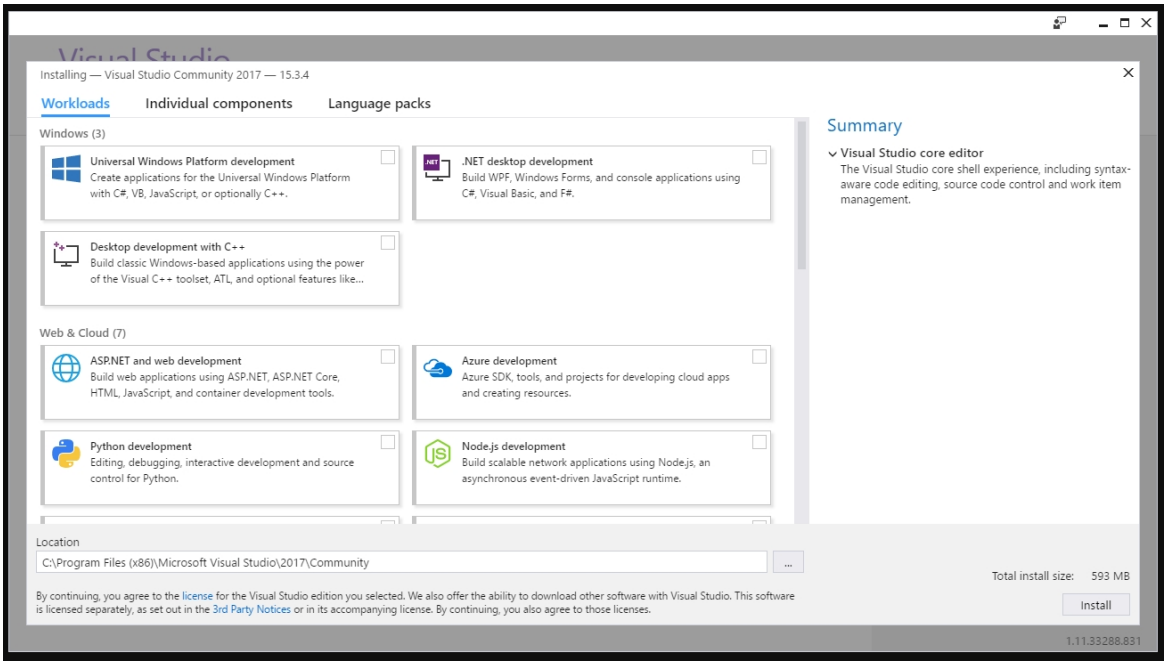
makes it easier to find and program things.

[You can download Visual Studio Community 2017 here.]

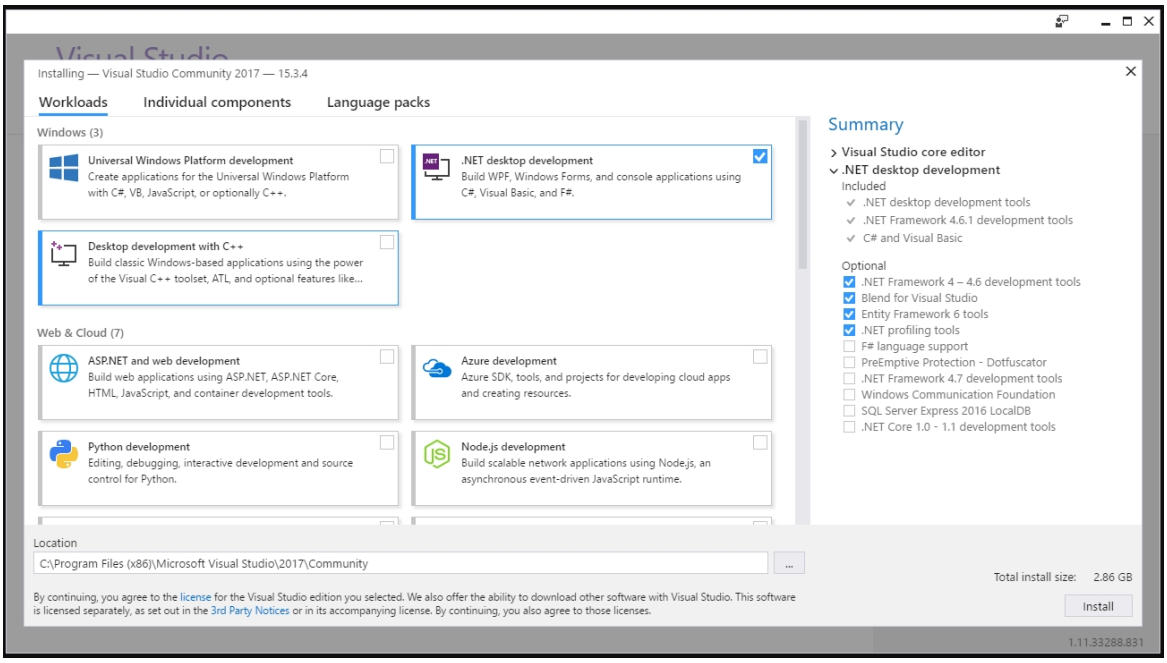
Begin the installation process by doubling clicking on the downloaded file.



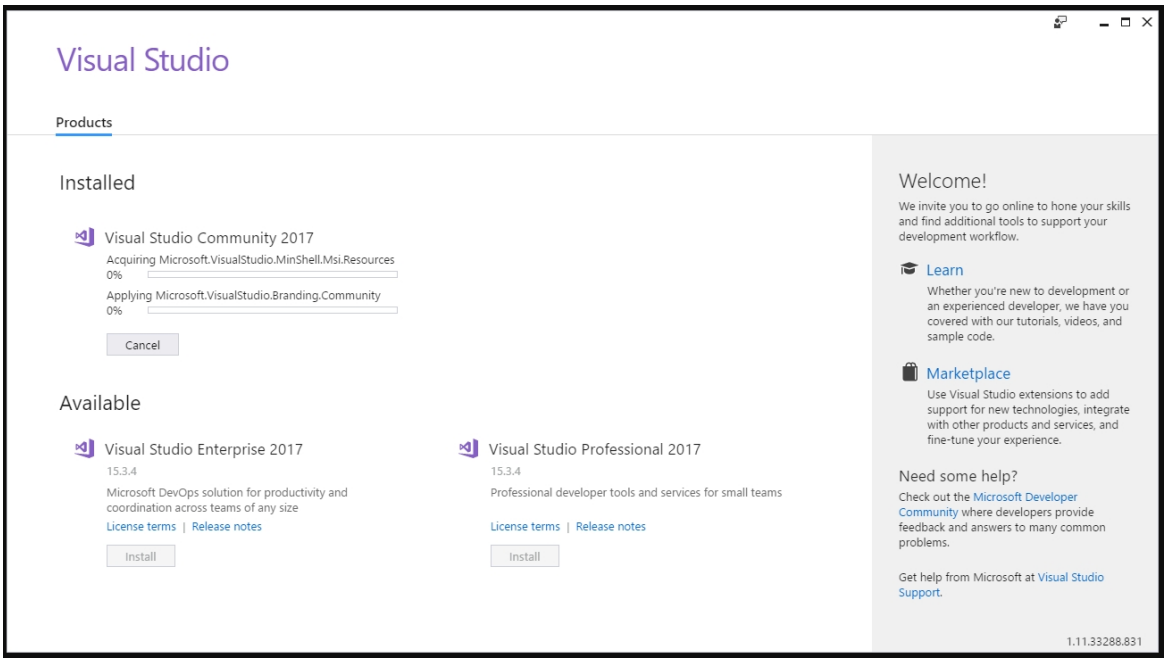
The initial installation screen can be intimidating, you you'll want to look for ".NET desktop development"



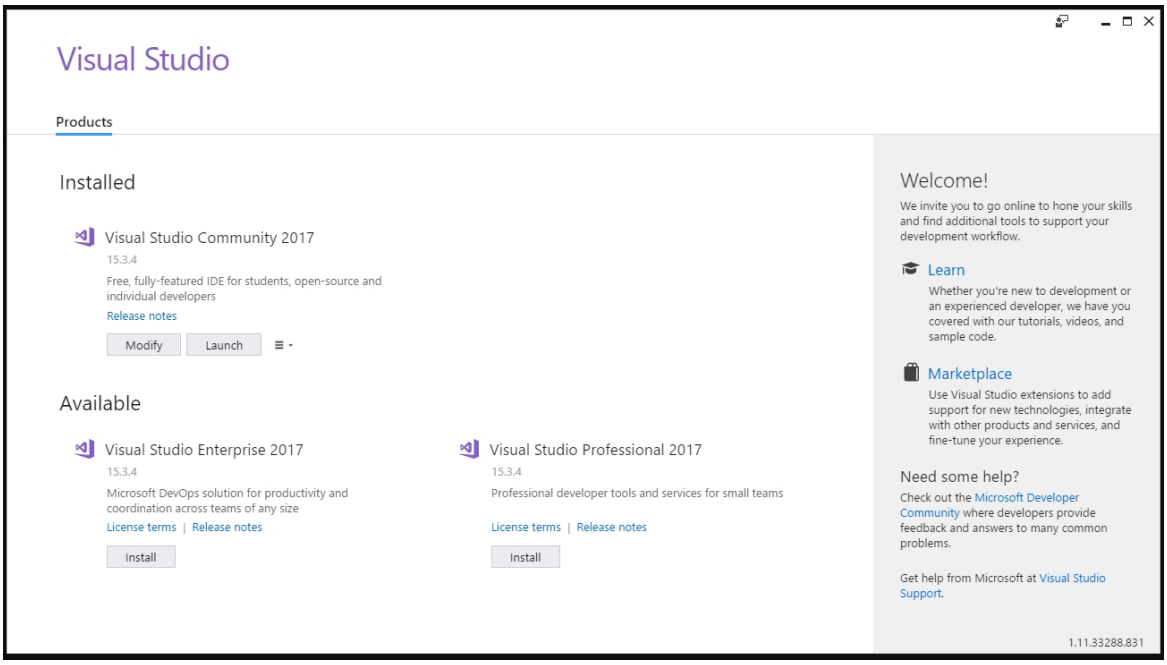
Select it by clicking the white box, and clicking on the Install.



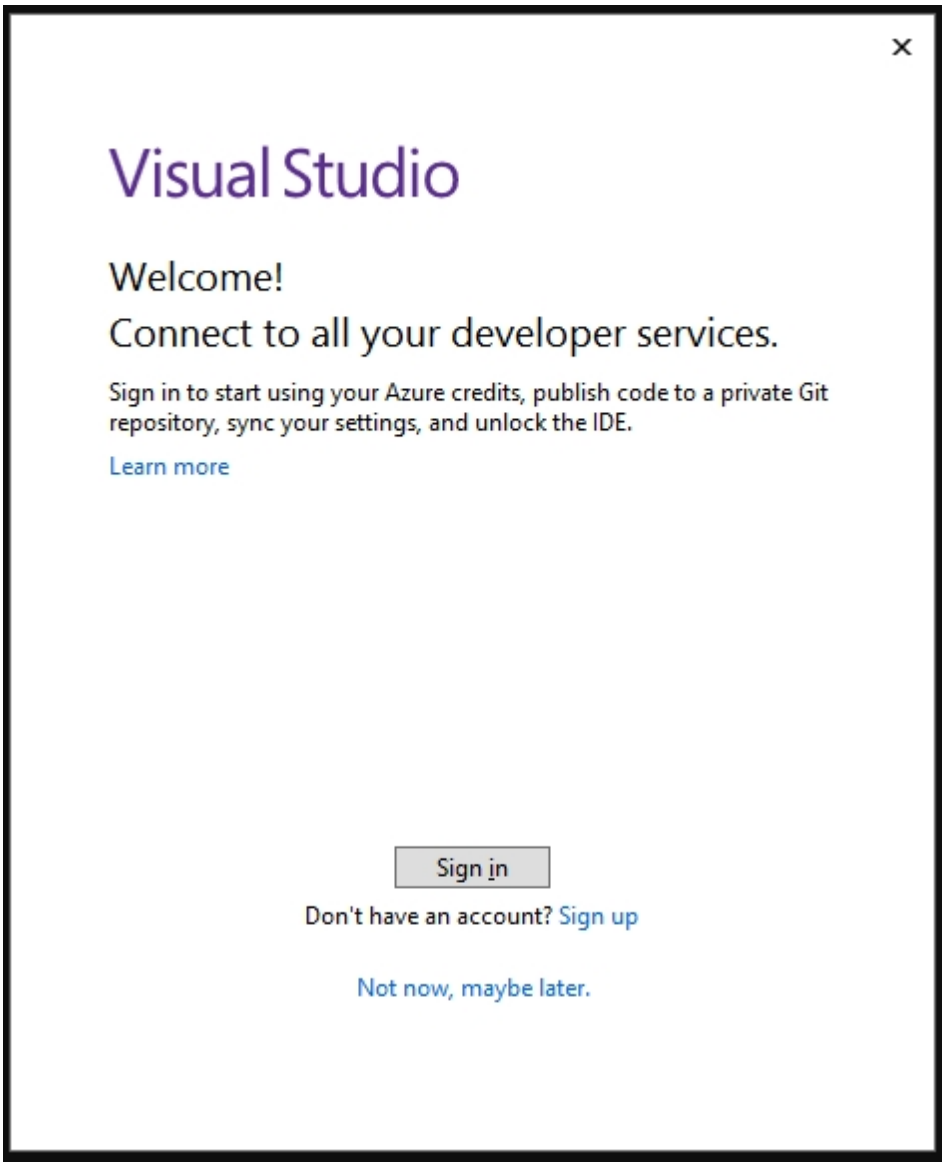
The installation process can take anywhere from 30 minutes to a few hours, depending on your internet speed and computer..



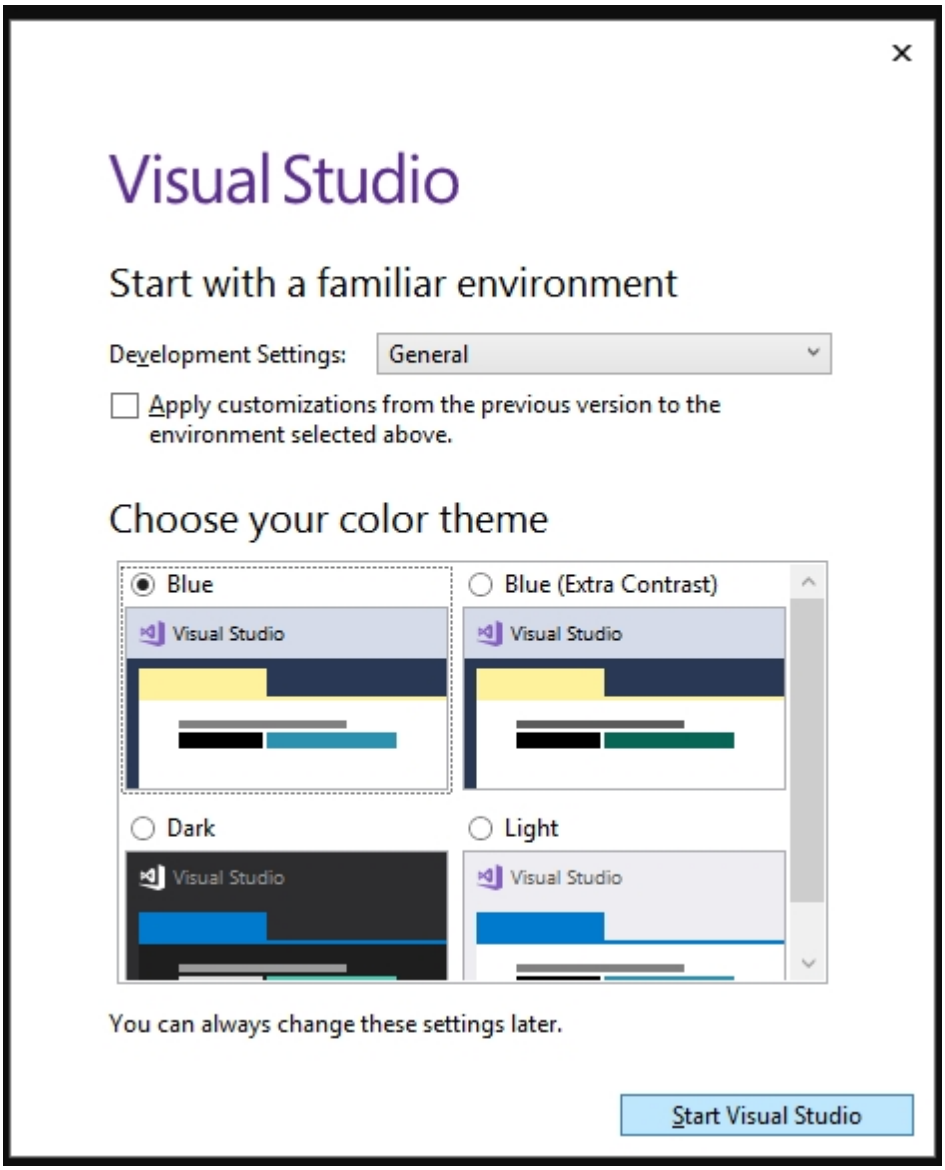
Once finished, you'll be presented with the Installed screen.



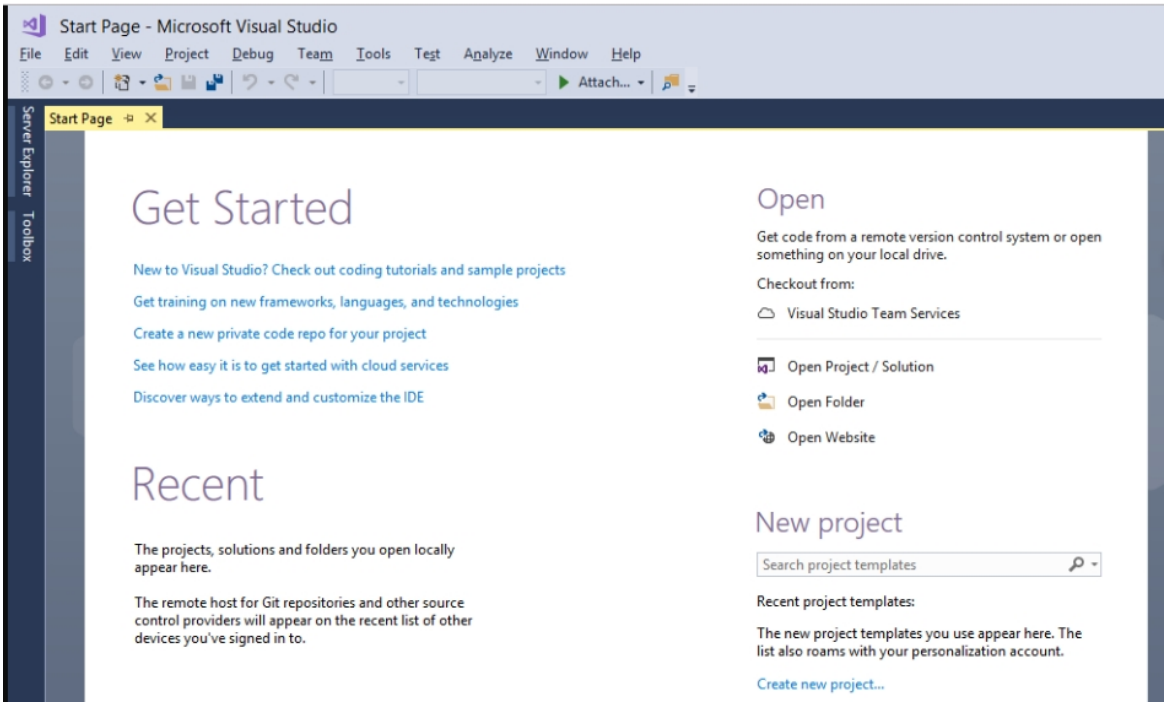
When ready, click on the Launch button to go through an initial configuration. You can skip the Sign In process by clicking on "Not now, maybe later."



The next screen determines the colour theme of your Visual Studio.



Once you have selected your option, click on Start Visual Studio

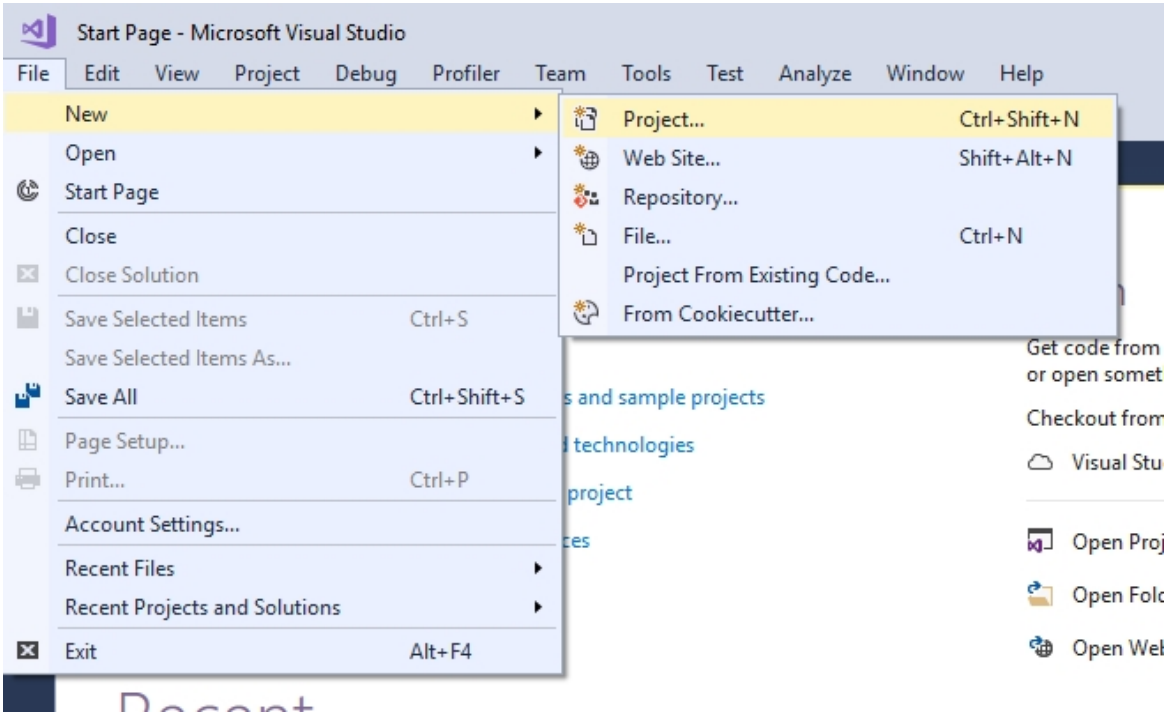


Creating New Visual Studio Project

If you want to be able to use Visual Studio to make new mods, or make changes to the existing ones, you may follow this guide, after you finish installing Visual Studio 2017.

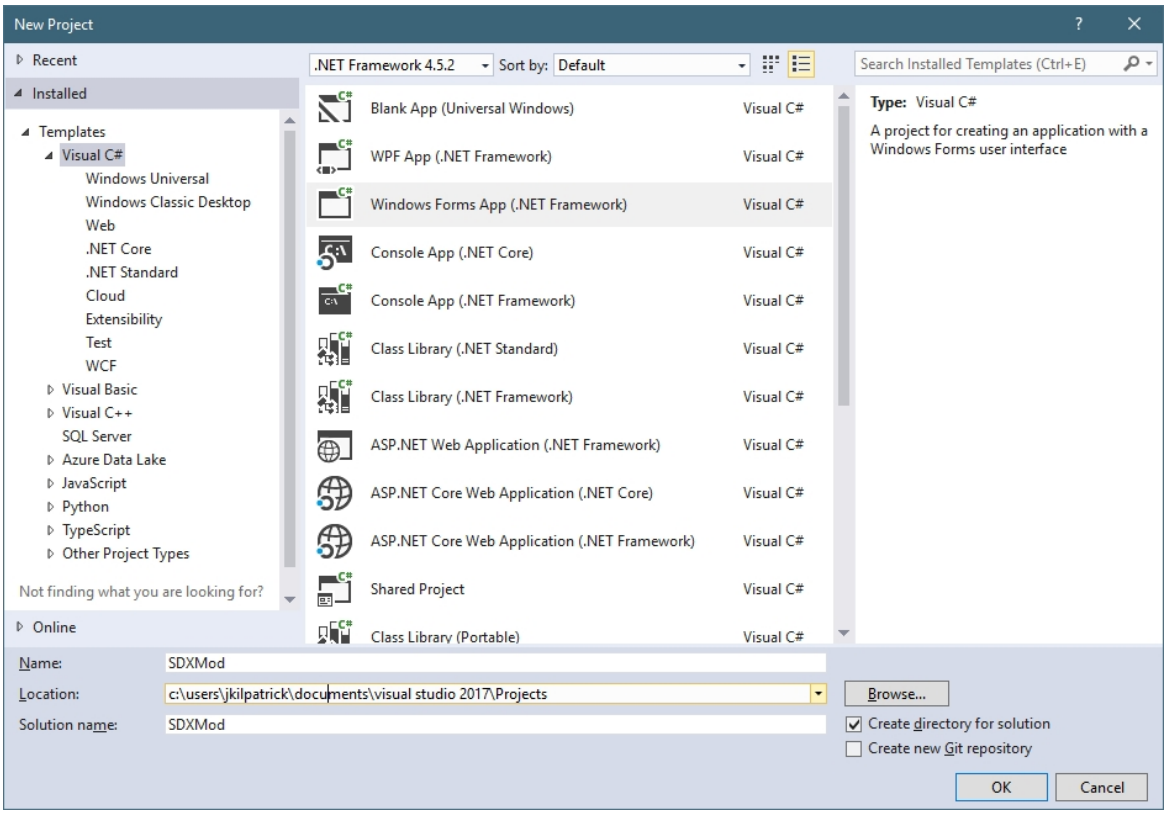
Creating a New Project

In Visual Studio 2017, create a new Project by clicking on the File menu, then New -> Project...

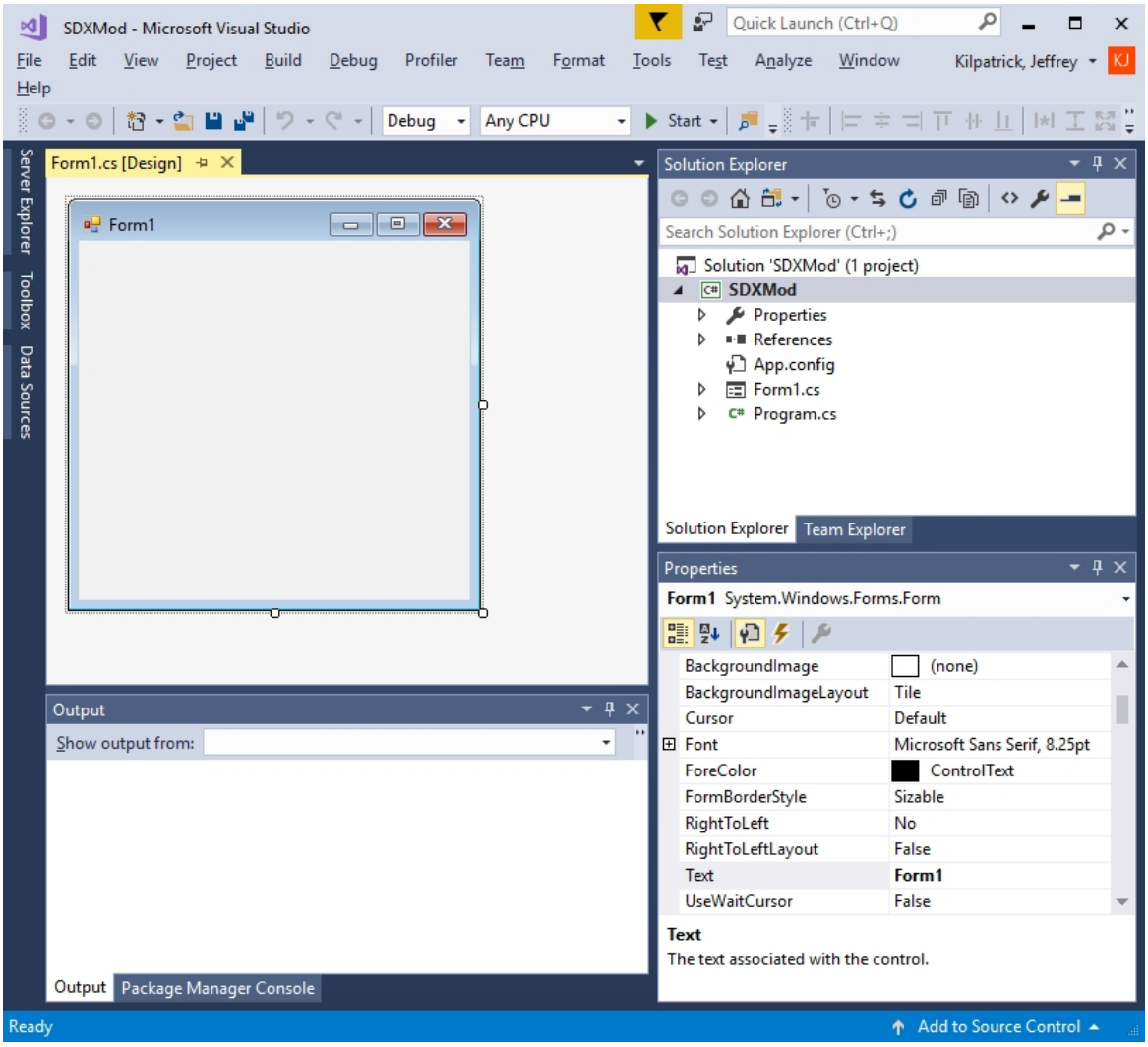


By default, Windows Forms App (.NET Framework) will be the selected Project Type. If it's not, you may select it.

Note: The Project type is not critical here. We could just as easily as choose Console App (.NET Framework).



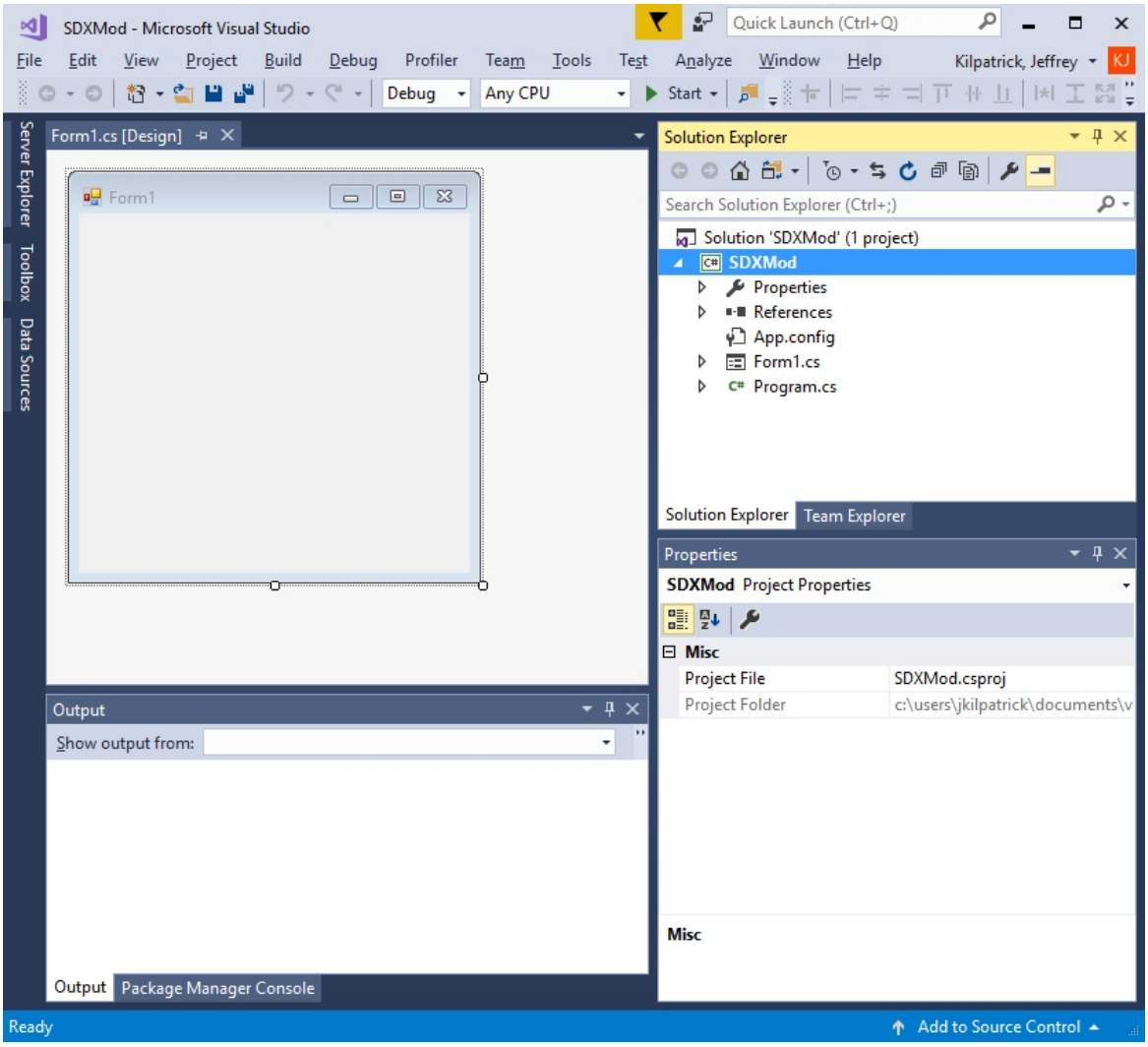
For the Name, choose something like SDXMod, and click on OK.



Linking your SDX Mods into Visual Studio

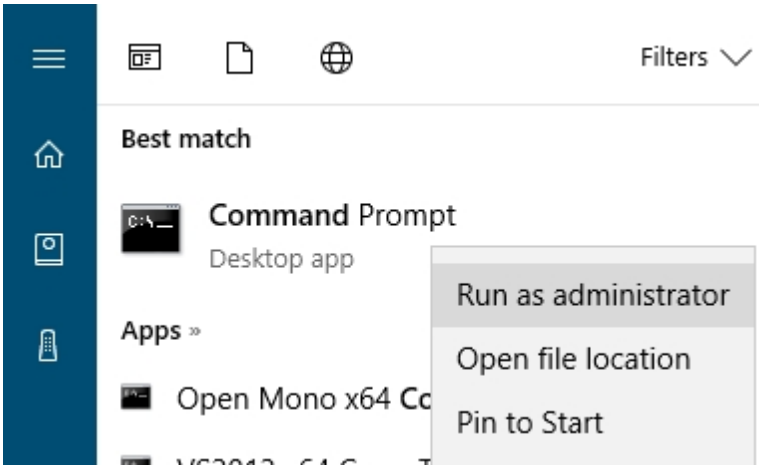
Linking SDX Mods into Visual Studio

In the Solution Explorer, click on SDXMod

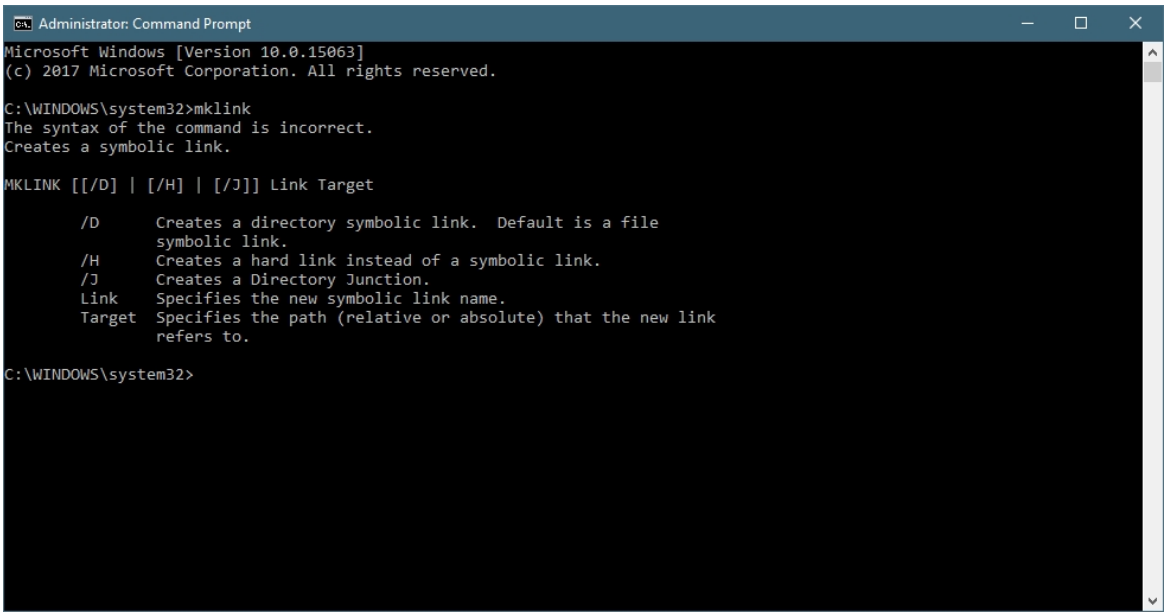


In the Properties screen, you'll see the Project Folder. Copy and Paste that into your Clip board.

Open up a Windows Command Prompt (as Administrator):



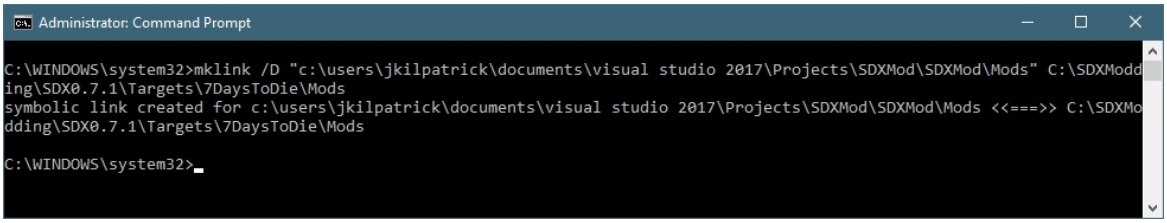
The command we are going to run is the mklink command. This will create a Windows Symlink from your SDX Folder to your Visual Studio Project.



We want to make a link between your Visual Studio project, and the SDX Folder.

The syntax is:

mklink /D "Your Project Folder" "YourSDX Mods Folder"



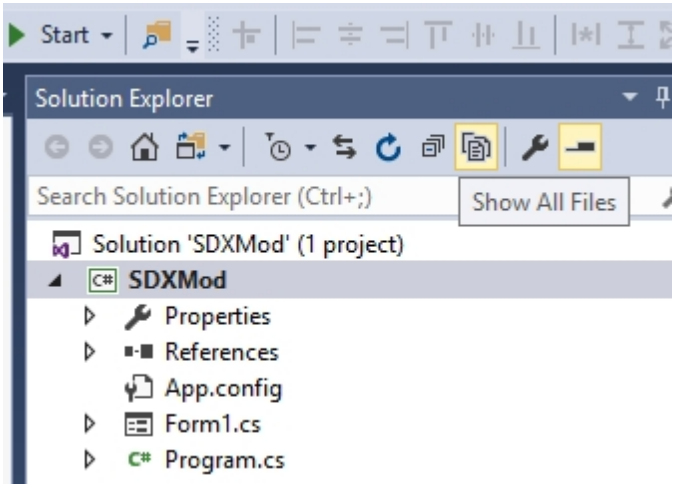
Example:

```
mklink /D "c:\users\jkilpatrick\documents\visual studio
2017\Projects\SDXMod\SDXMod\Mods" C:
\SDXModding\SDX0.7.1\Targets\7DaysToDie\Mods
```

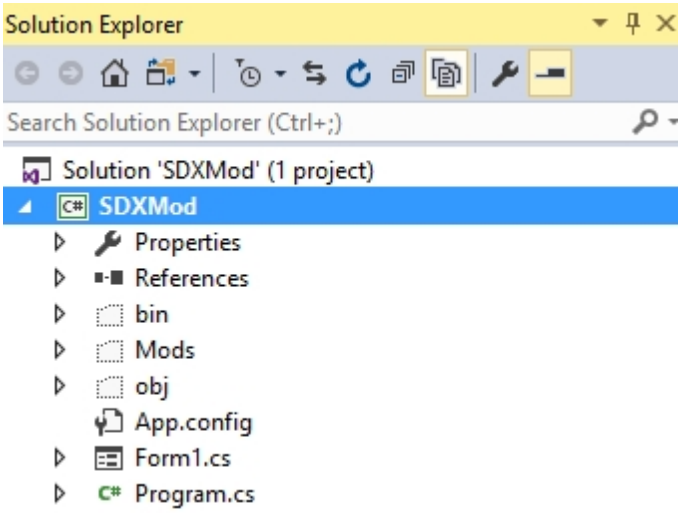
The first part of the command points to your Visual Studio Project, and to a non-existent "Mods" folder.

The second part of the command points to where your SDX Target Files are.

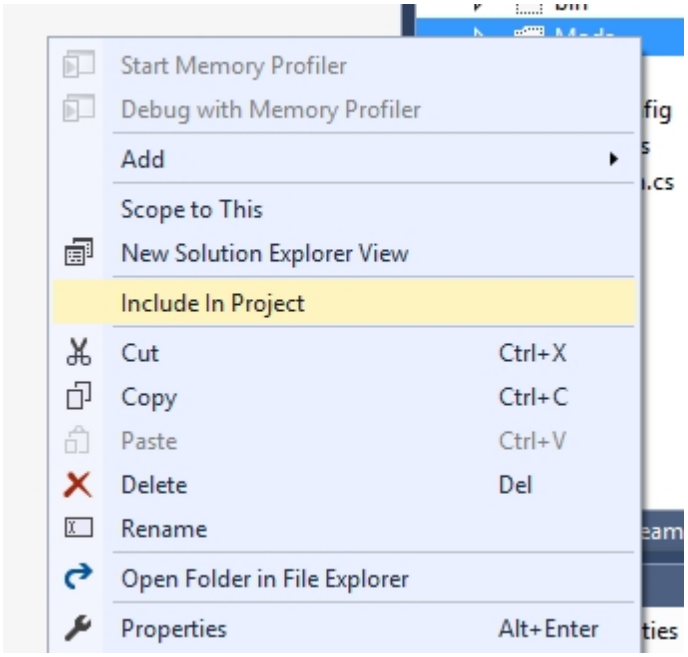
Back in your Visual Studio Project, go to your Solution Explorer, and look for "Show All Files"



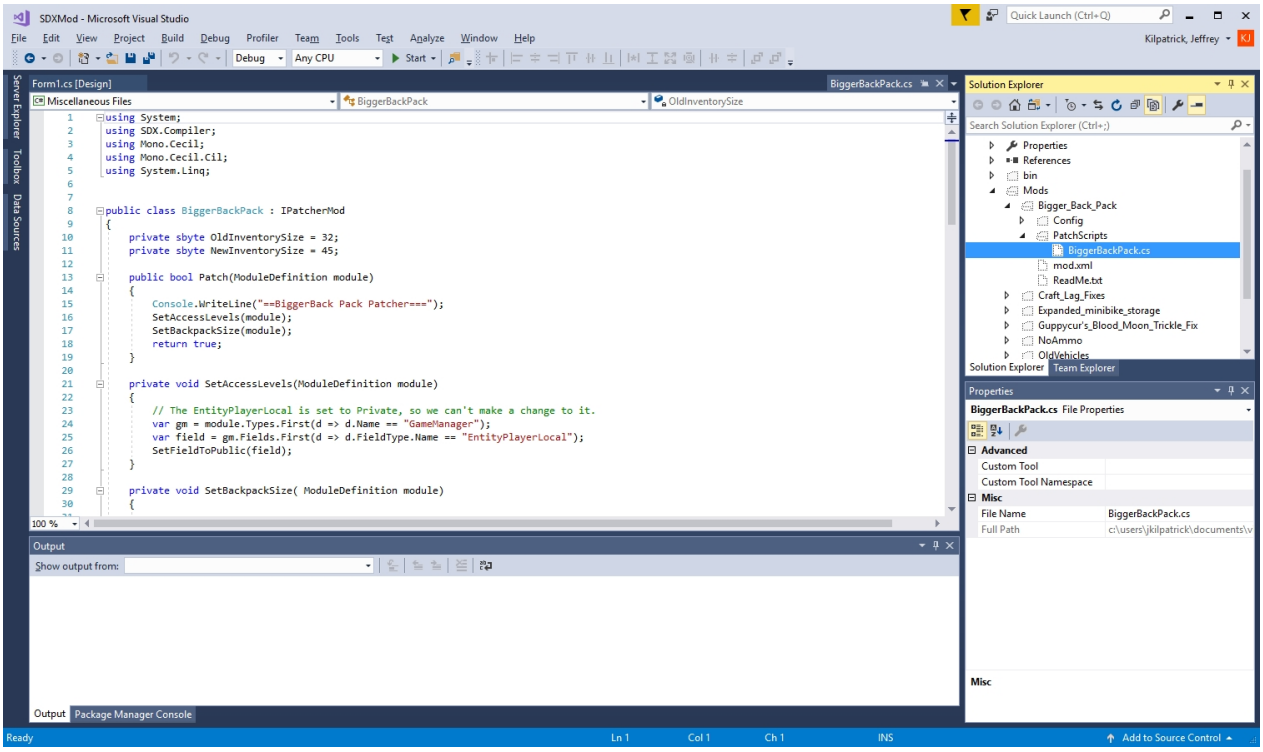
After you click on Show All Files, you'll see some faded folders:



Right click on the Mods folder, and choose Include In Project:



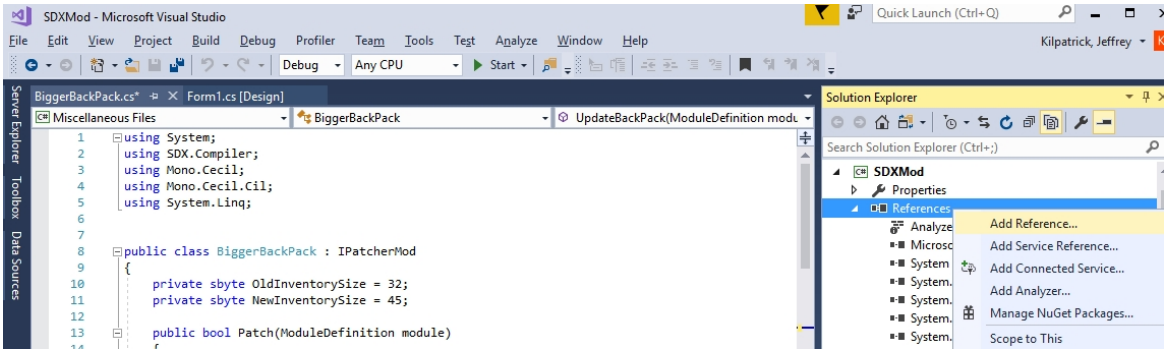
Click on the Mods folder, and drill down through your Mod files.



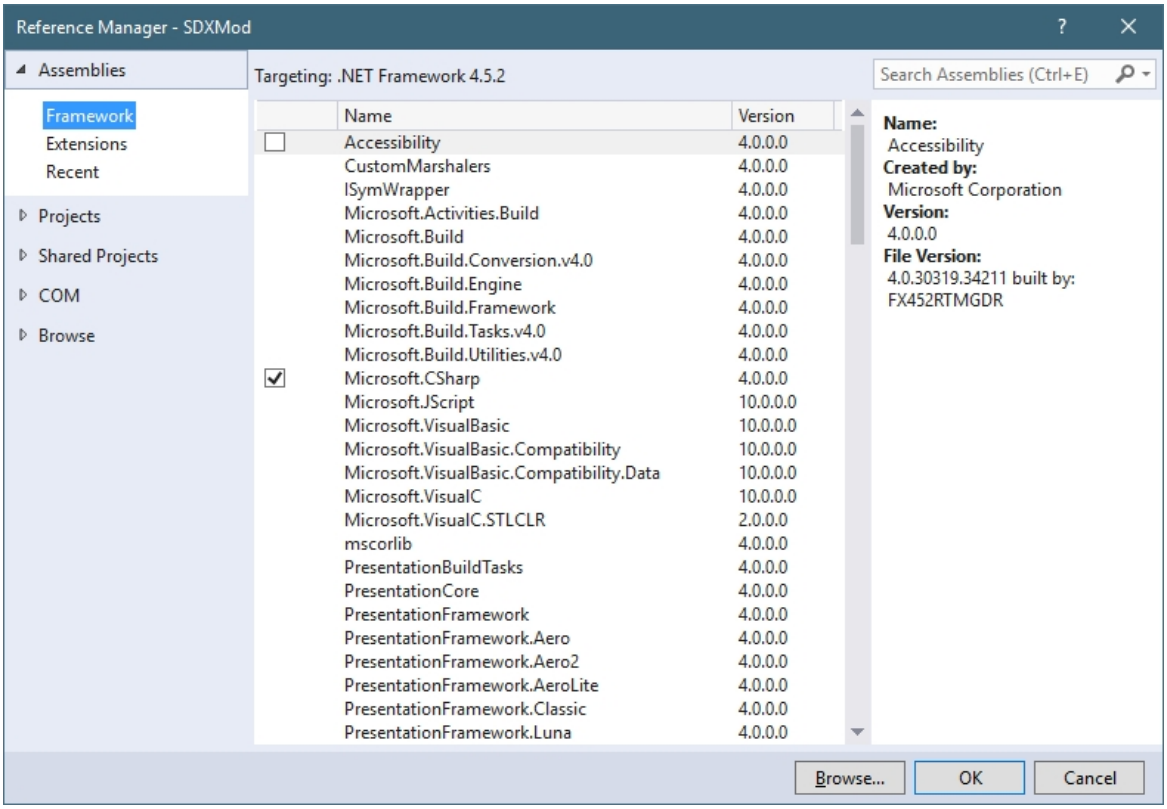
Adding in Assembly References

Add Assembly References for Intellisense

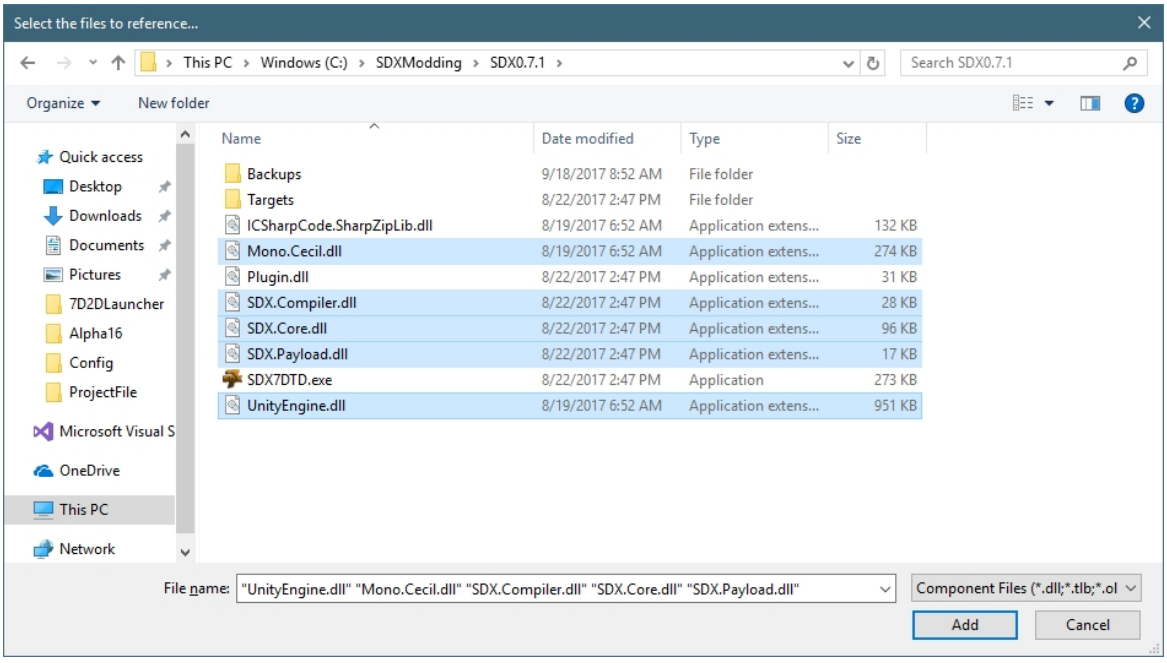
Under the SDMod, in the Solution Explorer, right click on the References and select "Add Reference..."



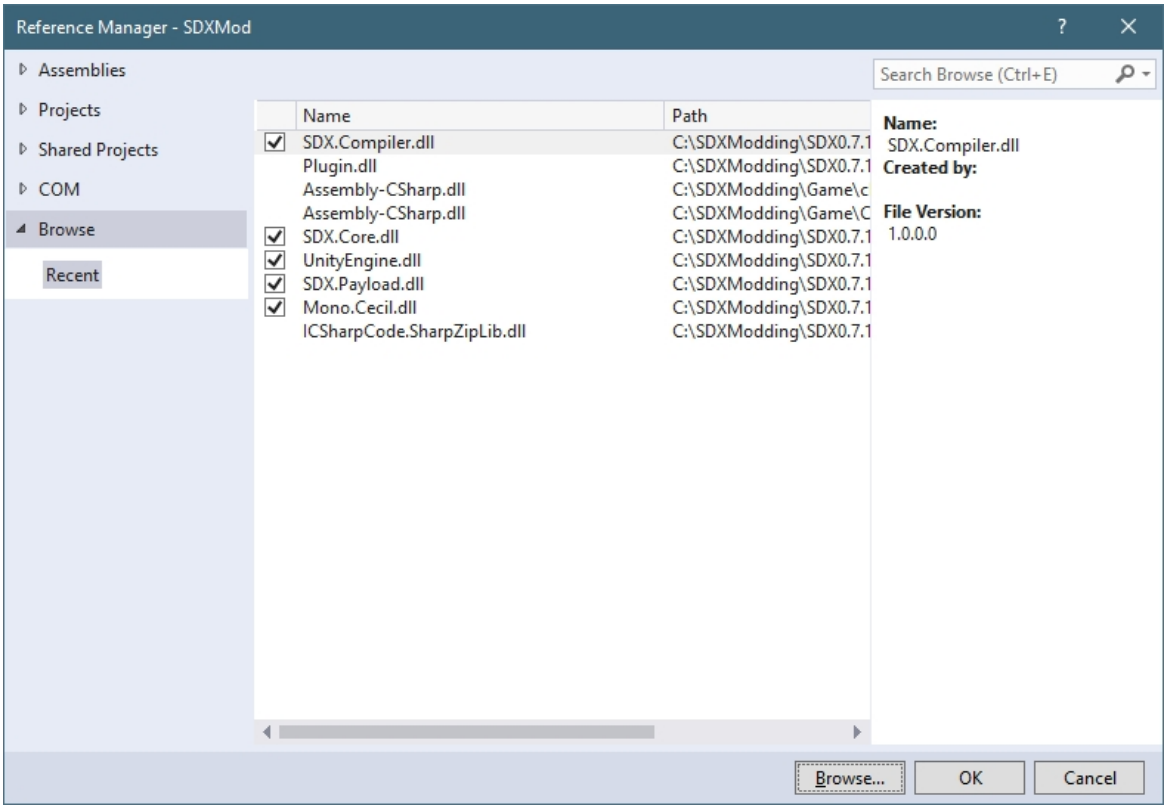
This will open the Reference Manager



Click on the Browse button, and navigate to your SDX Folder.



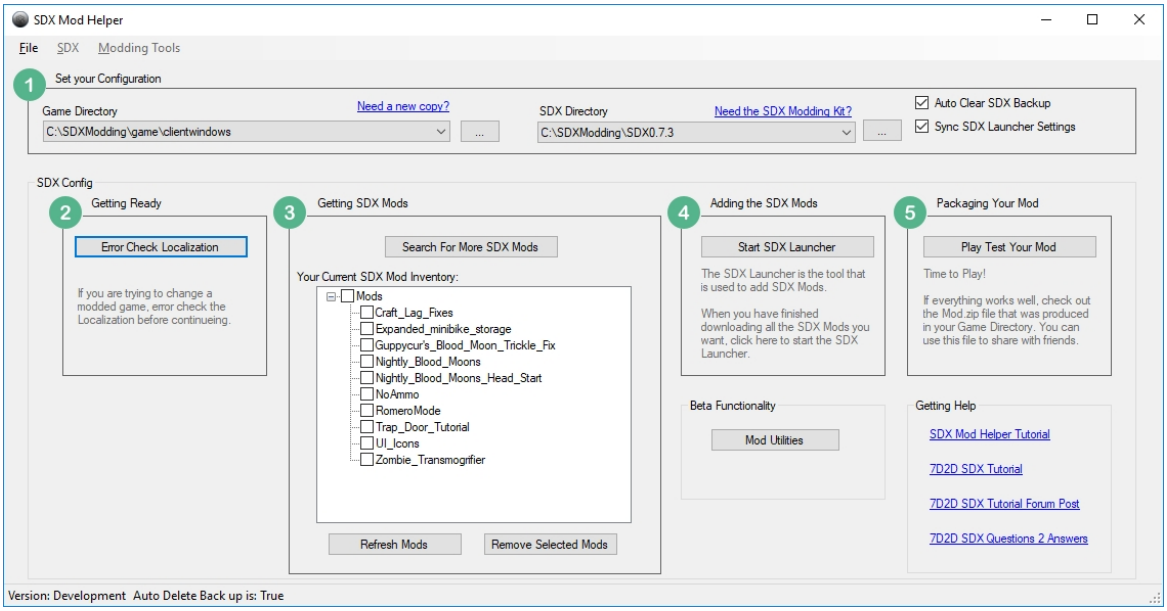
Multi-select the above DLLs, and click on "Add"



You may also add a reference to the Assembly-CSharp.dll by following the same steps, but navigating to your game folder instead.

How to use the 7D2D SDX Mod Helper

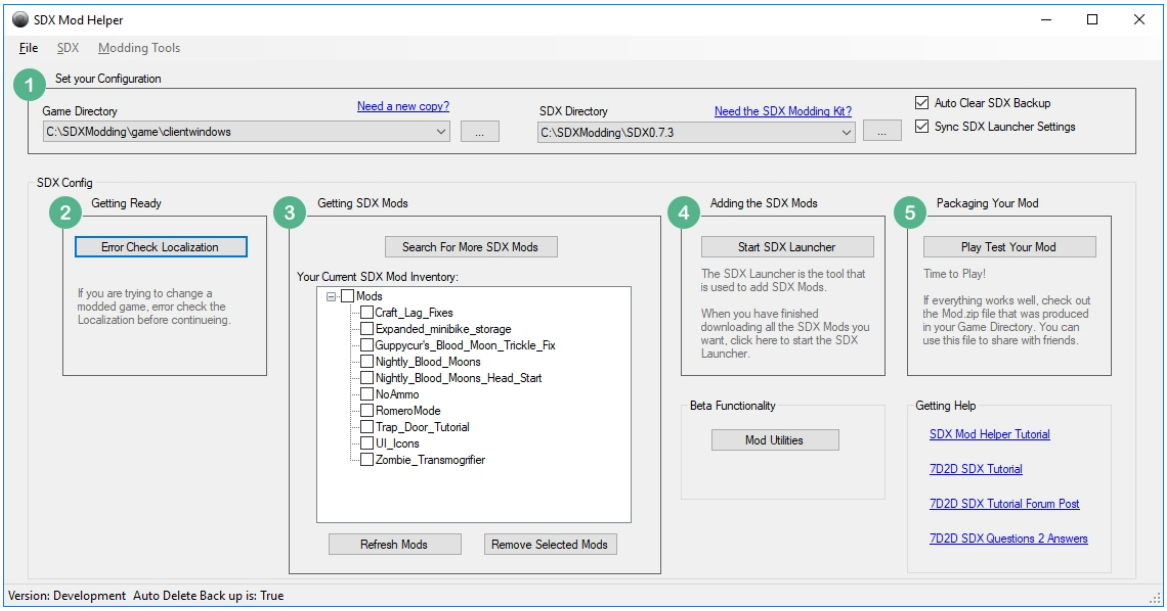
The 7D2D SDX Mod Helper is a tool designed to help you manage your SDX Mods, by giving you a guided work flow that'll lead you step by step through the process.



You may download the SDX Mod Helper by installing the SDX Modding kit, or downloading directly here.

Introducing the 7D2D SDX Mod Helper Screen

The 7D2D SDX Mod Helper is a tool designed to help you manage your SDX Mods, by giving you a guided work flow that'll lead you step by step through the process.



Step 1 - Set your Configuration

Your ***Game Directory*** points to which folder you want to add SDX mods from. This can be a vanilla install, a dedicated server, or it could be a modded game, such as War of the Walkers. You may click on the button with the three dots to bring up a directory browser, and navigate to another folder.

If you don't have a good copy of vanilla, you can click on "Need a new copy?" link. This will open up the Game Downloader screen, where you can install different versions of the game.

The ***SDX Directory*** needs to point to where you have installed SDX. If you are using the SDX Modding kit, this is by default going to be under C:\SDX\Modding\SDX0.7.1. You may click on the button with the three dots to bring up a directory browser, and navigate to your SDX Directrory.

Don't have SDX yet? Click on "Need the SDX modding Kit?" to be directed to the 7D2D SDX Tutorial site to get it.

The **Auto Clear SDX Backup** will clear the SDX back up folder if you have changed your Game Directory. For example, if you added SDX to your game, and you want to add those same SDX mods to the dedicated Linux server, you'd want to clear your back up first, then proceed. Selecting this will automatically do it.

The **Sync SDX Launcher Settings** will keep your SDX Mod Helper's game directory in sync with the SDX Launcher's settings. If you have this checked, it will keep the tools in sync. If you change your Game Directory in one tool, it'll update the other tool automatically.

Step 2 - Getting Ready

If you have selected a modded version of the game to add SDX mods to, then you will want to run the ***Error Check Localization***. This will check your Game Directory's localization files, looking for any errors, such

as:

Mis-aligned Columns: This happens when you have too many, or too few, columns in your localization files. For example, sometimes an extra comma is missed, or added.
Duplication: Sometimes multiple entries of the same localization exists. This will remove the duplicate entries.

Having a bad set of Localization files will cause SDX to fail. When in doubt, run the ***Error Check Localization*** before you begin.

All Localization Errors *MUST* be fixed before continuing.

Step 3 - Getting SDX Mods

One of the key features of the SDX mod Helper is the ability to find and download mods directly into your SDX Directory.

The ***Search For More SDX Mods*** will open up the SDX Mod Browser window, where you can read about the available mods, and download them easily.

Your Current SDX Mod Inventory shows you the SDX Mods you have currently downloaded.

Refresh Mods will refresh your Mod Inventory, if you have manually copied some SDX mods into the folder.

The ***Remove Selected Mods*** will delete the SDX mods you have selected in the ***Your Current SDX Mod Inventory***.

Step 4 - Adding the SDX Mods

This section will start the SDX Launcher, allowing you to build the SDX mods into your Game Directory.

Step 5 - Packaging Your Mod

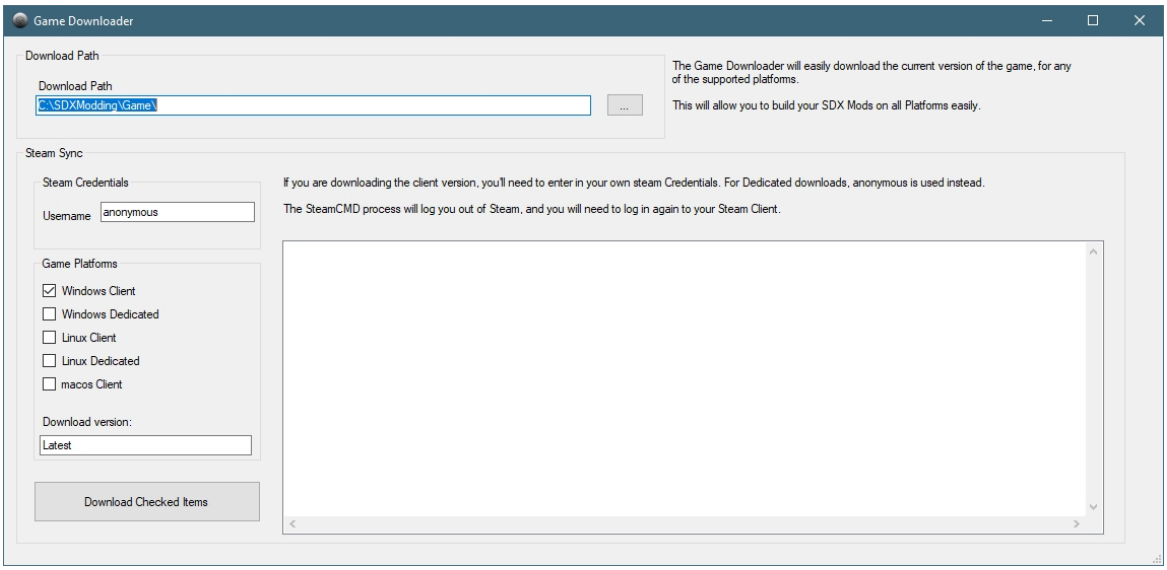
This section will start the game for you to run a Play test through your new SDX mod. It will also automatically:

Remove duplicate ItemIcons - It will look for any items that exists more than once under the Mods folder in your Game Directory, and remove duplicates.
Generate a Mod zip file - It will also generate a Mod.zip file, located in your Game Directory, that contains all the files that have been changed recently. This file can be shared with others to play your mod.

Game Downloader

The Game Downloader is available when you click on the "Need a new copy?" on the SDX Mod Helper main

screen. This will connect to Steam, either using your Steam Credentials, or using anonymous for dedicated server downloads.



Download Path: Select where you want to download the game from. You may click on the button with three dots to change the default localtion.

Each Game Platform will be downloaded to a separate folder, ie, C:\SDXModding\Game\windowsclient, C:\SDXModding\Game\linuxclient

Steam Credentials: This is where you'd put your Steam username. This is only necessary if you are downloading the client version, either Linux, Windows, or MacOS. You will be prompted for your Steam Password, and Steam Guard code, if it's your first time running from this PC.

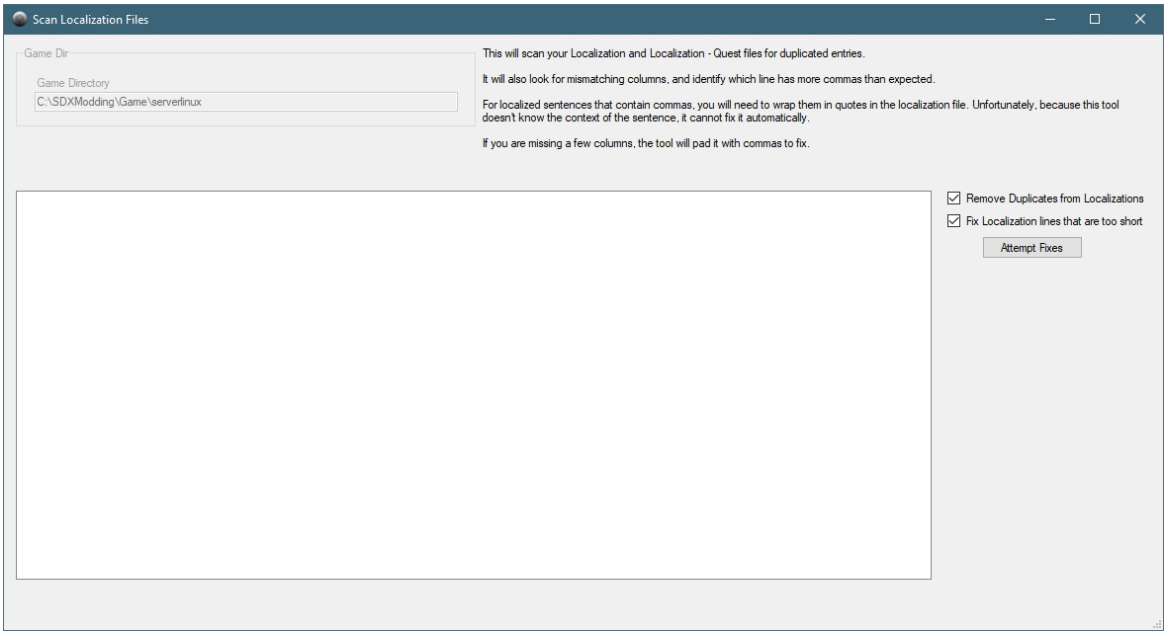
Game Platforms: You may select one or more different types of downloads. For example, if you want a copy of the Windows client and a dedicated version, you may select Windows Client and Windows Dedicated.

Download version: By default, it will download the latest version of the game. However, if you want to download an older version, or an experimental version, simple change "Latest" to the beta name in Steam.

For example, if you want the latest experimental, change "Latest" to "latest_experimental". For a previous version, change it to "alpha16.2"

Scan Localization Files

The Error Check Localization button on the SDX MOd Helper main screen will open the Scan Localization Files screen.



The Game Directory is set from the SDX Mod Helper main screen.

The following options are available:

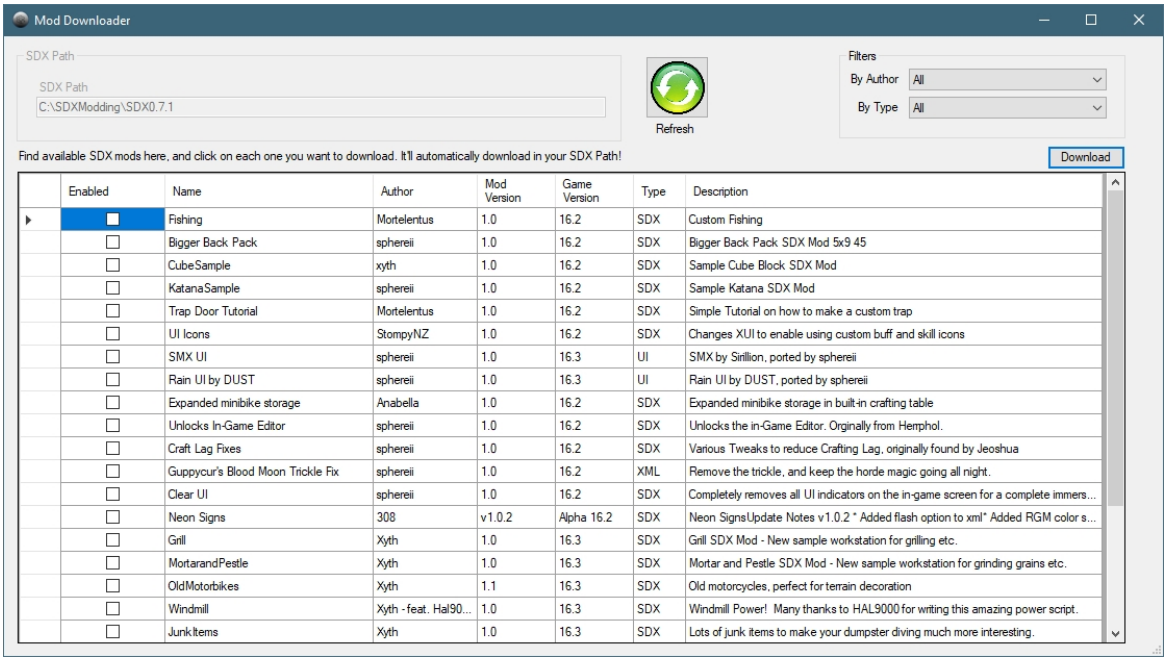
Remove Duplicates From Localizations - This will remove duplicate lines from the localization files in the Game Directory.
Fix Localization lines that are too short - This will scan for lines which have too many, or too few, commas.

Attempt Fixes button will attempt to fix some of the localization errors it detects.

If you have too many commas in a Localizations entry, the Scan Localization Files cannot automatically fix this.You must fix this manually. That is because the Scan Localization Files does not understand the context of your line. More often than not, extra commas are related due to missing quotes over a line.

Mod Downloader

The SDX Mod Downloader will allow you to view the available SDX Mods, and download them easily.



Enabled	Name	Author	Mod Version	Game Version	Type	Description
☐	Fishing	Mortelentus	1.0	16.2	SDX	Custom Fishing
☐	Bigger Back Pack	sphereii	1.0	16.2	SDX	Bigger Back Pack SDX Mod 5x9 45
☐	CubeSample	xyth	1.0	16.2	SDX	Sample Cube Block SDX Mod
☐	KatanaSample	sphereii	1.0	16.2	SDX	Sample Katana SDX Mod
☐	Trap Door Tutorial	Mortelentus	1.0	16.2	SDX	Simple Tutorial on how to make a custom trap
☐	UI Icons	StompyNZ	1.0	16.2	SDX	Changes XUI to enable using custom buff and skill icons
☐	SMX UI	sphereii	1.0	16.3	UI	SMX by Sirillion, ported by sphereii
☐	Rain UI by DUST	sphereii	1.0	16.3	UI	Rain UI by DUST, ported by sphereii
☐	Expanded minibike storage	Anabella	1.0	16.2	SDX	Expanded minibike storage in built-in crafting table
☐	Unlocks In-Game Editor	sphereii	1.0	16.2	SDX	Unlocks the in-Game Editor. Orginally from Hemphol.
☐	Craft Lag Fixes	sphereii	1.0	16.2	SDX	Various Tweaks to reduce Crafting Lag, originally found by Jeoshua
☐	Guppycur's Blood Moon Trickle Fix	sphereii	1.0	16.2	XML	Remove the trickle, and keep the horde magic going all night.
☐	Clear UI	sphereii	1.0	16.2	SDX	Completely removes all UI indicators on the in-game screen for a complete immers...
☐	Neon Signs	308	v1.0.2	Alpha 16.2	SDX	Neon SignsUpdate Notes v1.0.2 * Added flash option to xml* Added RGM color s...
☐	Grill	Xyth	1.0	16.3	SDX	Grill SDX Mod - New sample workstation for grilling etc.
☐	MortarandPestle	Xyth	1.0	16.3	SDX	Mortar and Pestle SDX Mod - New sample workstation for grinding grains etc.
☐	OldMotorbikes	Xyth	1.1	16.3	SDX	Old motorcycles, perfect for terrain decoration
☐	Windmill	Xyth - feat. Hal90...	1.0	16.3	SDX	Windmill Power! Many thanks to HAL9000 for writing this amazing power script.
☐	JunkItems	Xyth	1.0	16.3	SDX	Lots of junk items to make your dumpster diving much more interesting.

SDX Path: This is set on the main SDX Mod Helper. This is the directory that the Mod Downloader will download to.

Filters:

By Author: If you want to see Mods by a specific author, select the Author from this list.
By Type: If you are looking for a particular type of SDX mod, select the type from this list.

The Enabled column in the SDX Mod browser selects which mod you want to download when you click on the Download button

Name: The SDX Mod's name.

Author: The author of the SDX Mod

Mod Version: The version of the SDX mod, determined by the Author.

Game Version: This determines which version of the game it was last tested on it. It may or may not be compatible with future, or past versions of the game.

Type: The type of SDX mod it is. For example, SDX implies there's changes to the DLL and code. A UI or XML type, would only be updating the XML files.

Description: A short description of what the SDX mod does.

When you have selected which SDX Mods you want to download, the Download button will download them into your SDX Path.

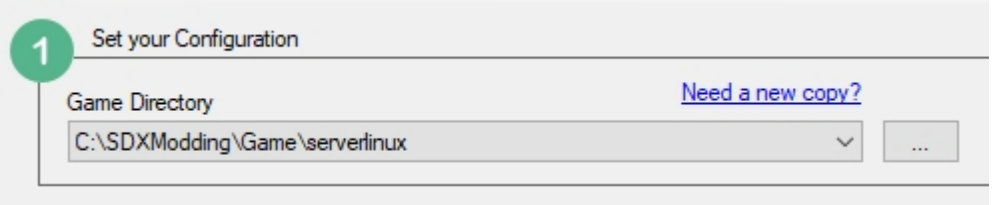
Adding an SDX mod to a Vanilla Install

This guide will take you through the steps on how to download a fresh vanilla version of the game, and install the Bigger Back Pack mod, using the SDX Mod Helper.

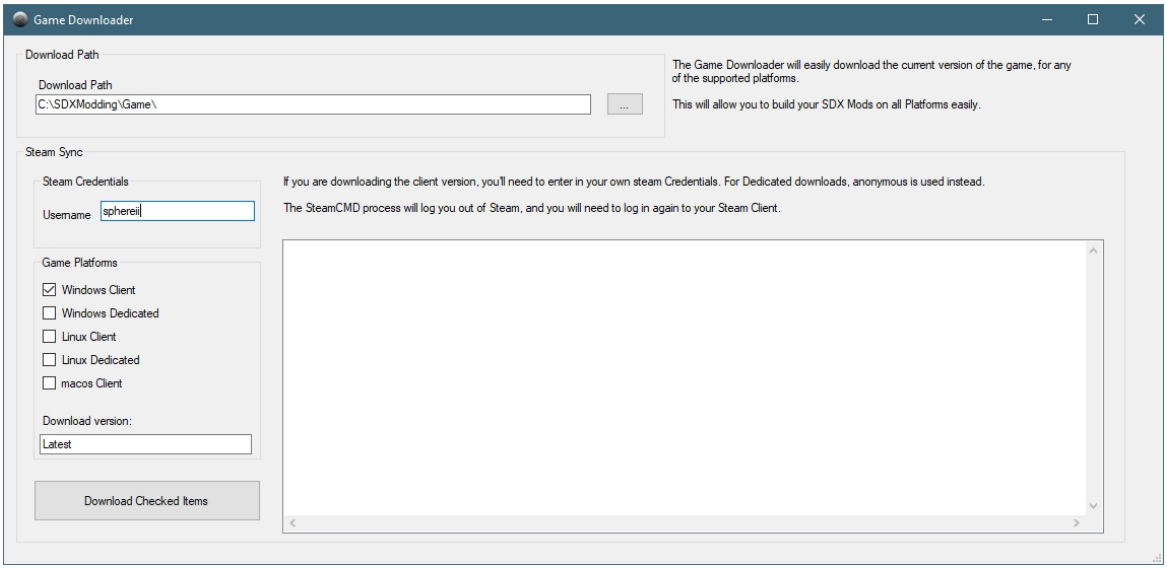
Installing a new Game Directory

In this section, we will walk you through the process of adding a few SDX mods to a vanilla install of the game.

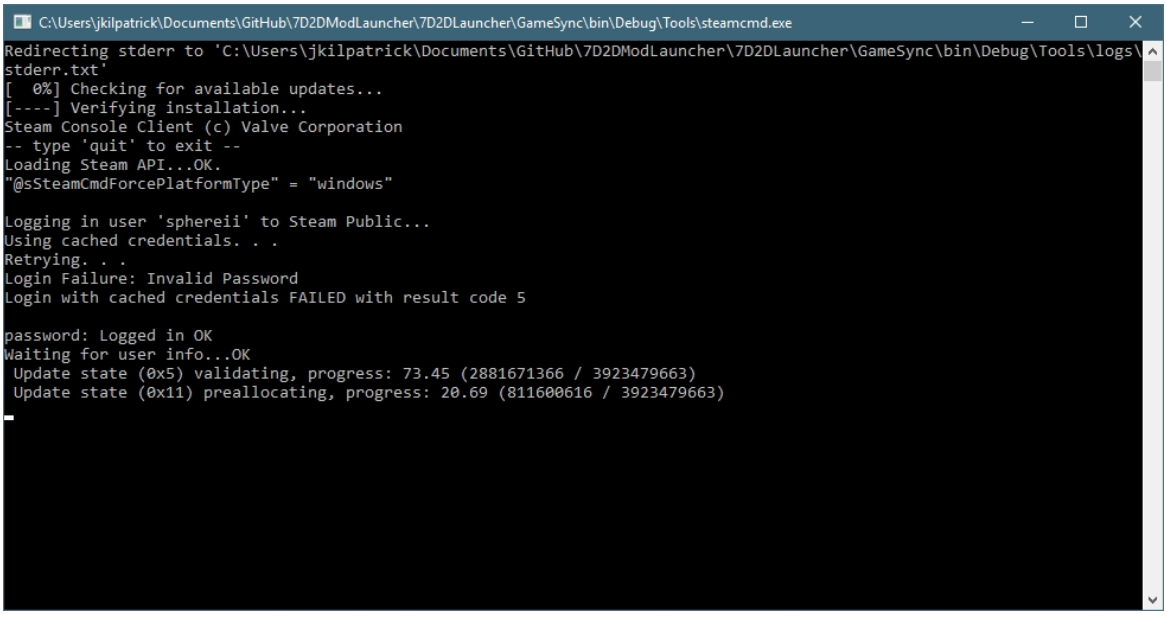
In the Set your Configuration:



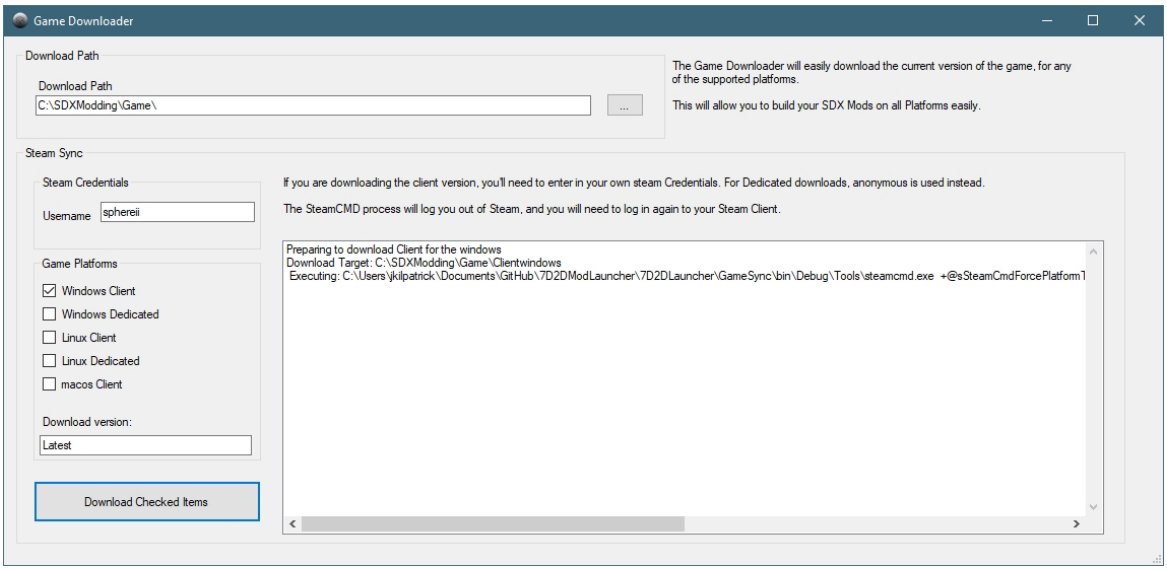
Click on the "Need a new copy?" to open the Game Downloader screen.



You will be prompted for your Steam Password, and your Steam Guard code.

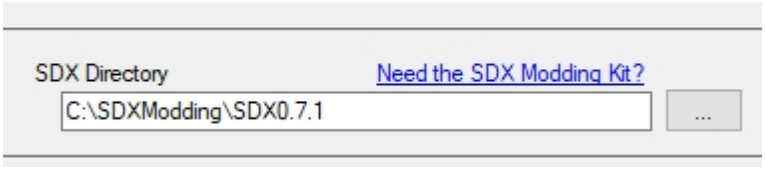


When the process is complete, you'll be returned to the Game Downloader screen

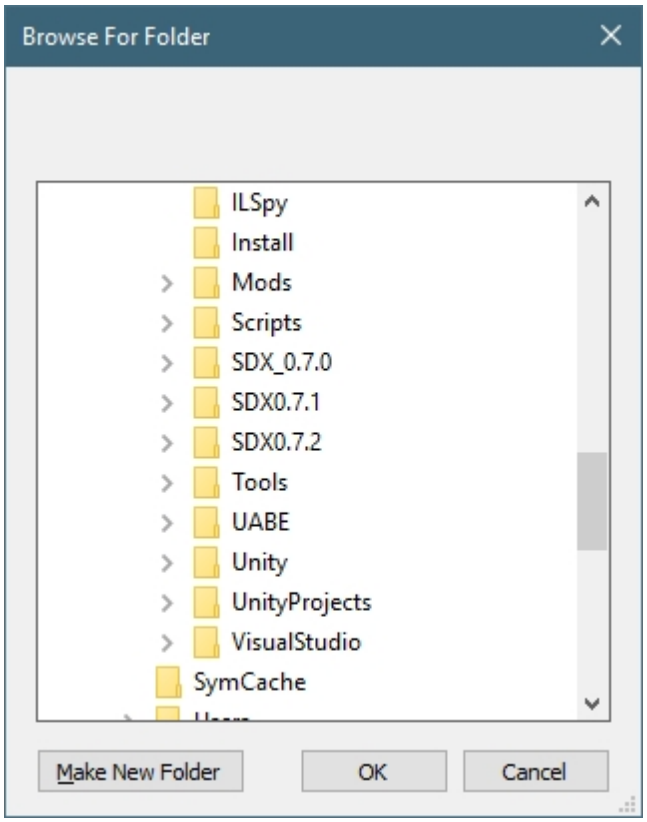


Updating the SDX Path

Set your SDX Directory



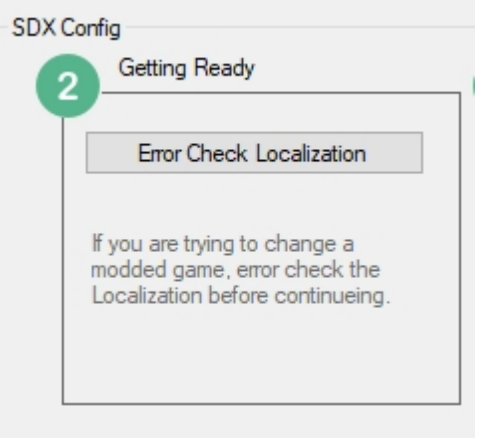
If the SDX Directory isn't correct, click on the button with the 3 dots to open up a folder browser



Then select which folder you want, and click on OK

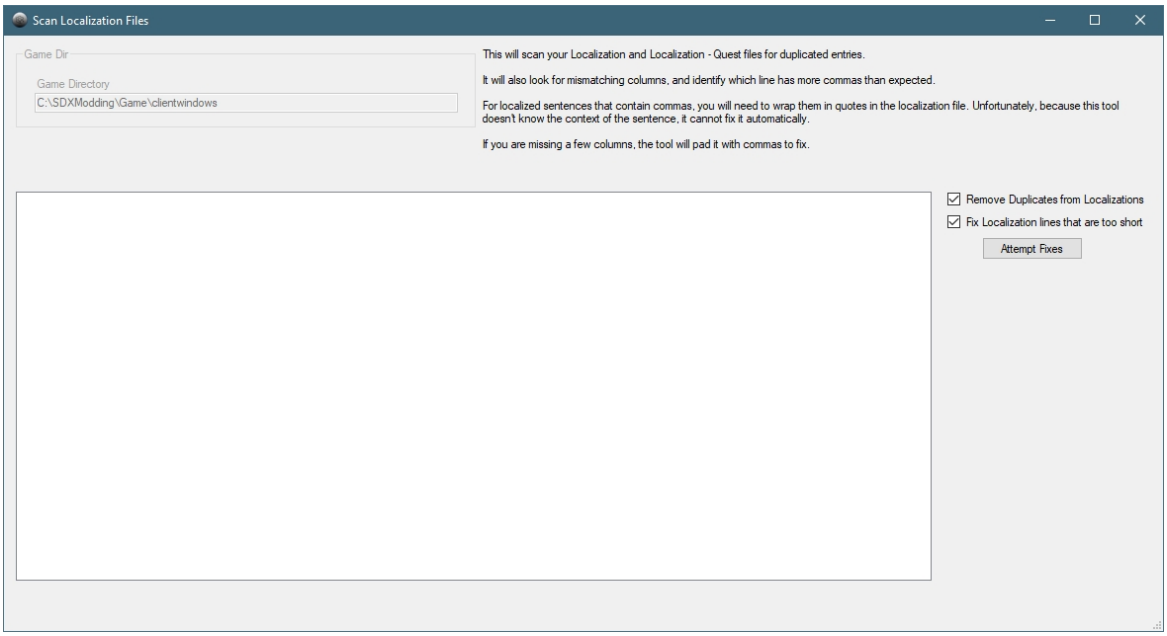
Error Checking Localization

If you are using a vanilla version, as we are in this example, you typically won't have to worry about Error Checking Localization. However, as many mods have errors in their localization files, it's best to make it a habit of checking every time.



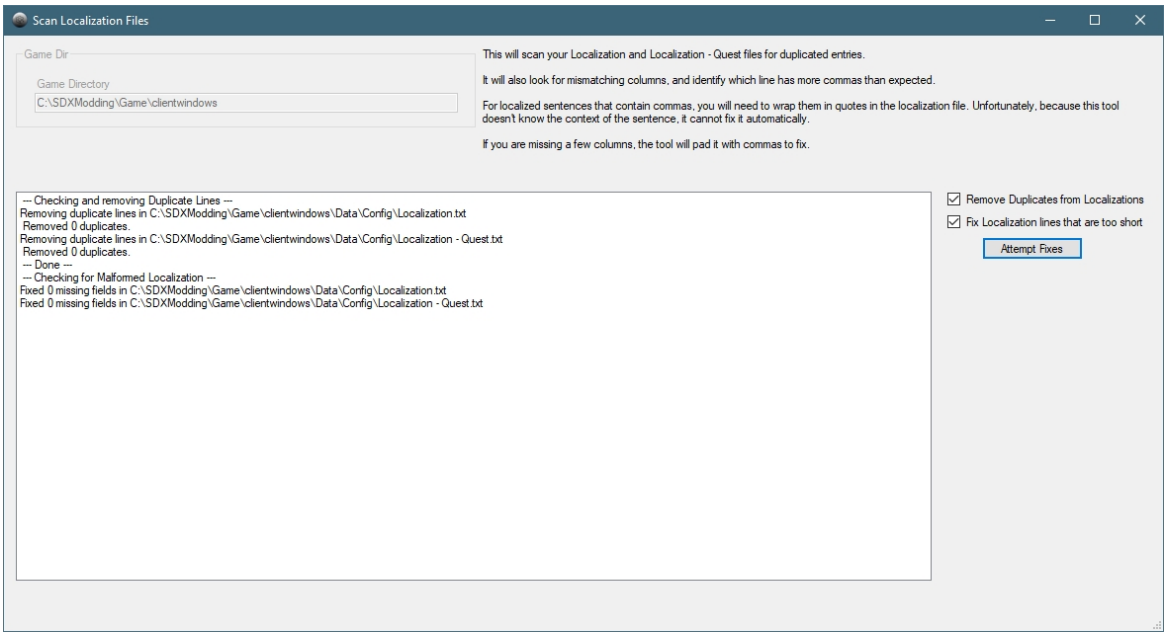
If you have any errors in the localization files, SDX will fail.

Click on the Error Check Localization file to open the Scan Localization Files screen.



Verify that the Game Directory is set correctly. If not, go back to the SDX Mod Helper main screen and change it there.

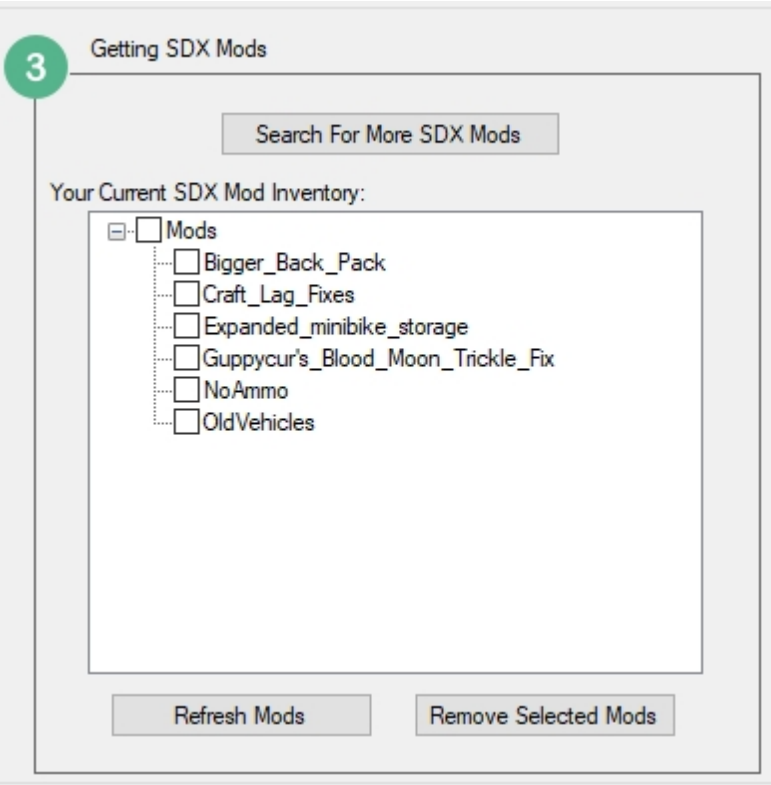
Click on the Attempt Fixes button.



It removed 0 duplicates, meaning, it did not find any. It also did not find any Malformed Localization entries, by indicating it did not have to fix any.

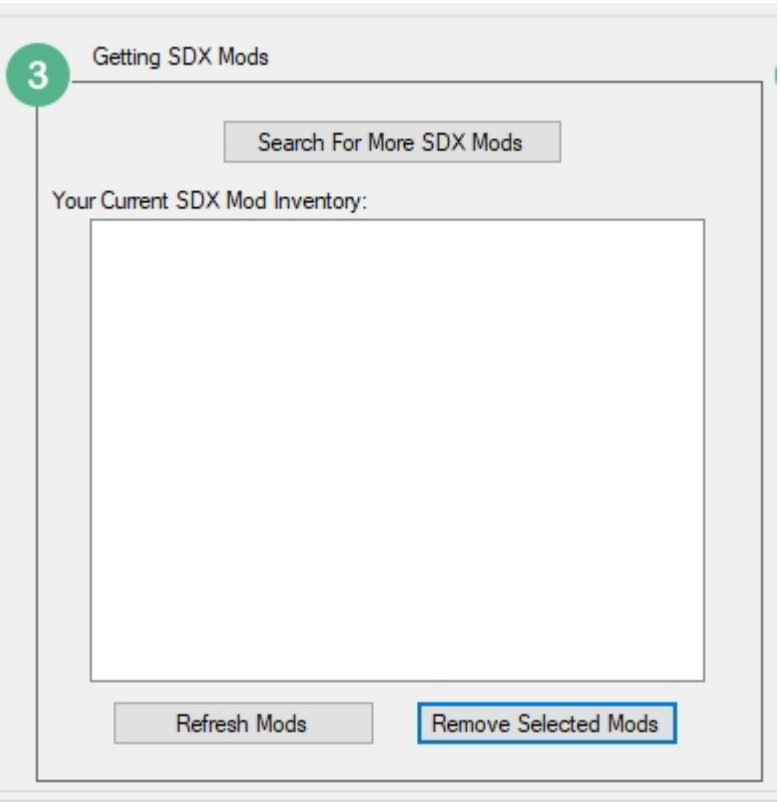
Downloading SDX Mods

We now want to find which SDX Mods we want to add to this vanilla entry.



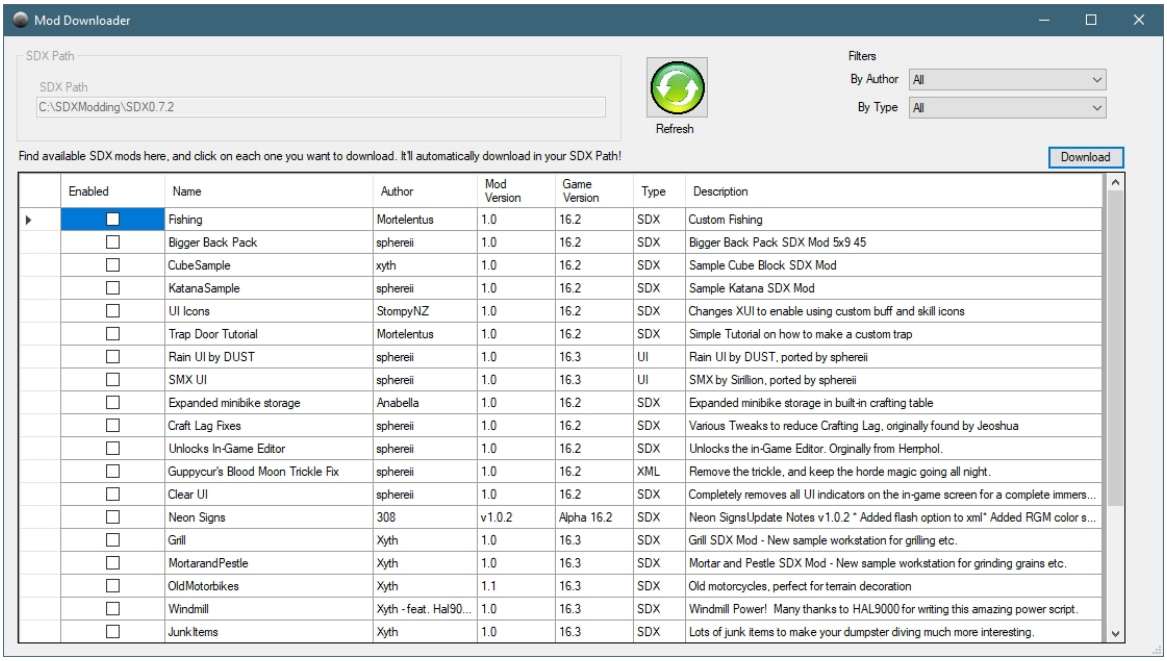
In the above, I have some Mods already downloaded. For this step, we are going to remove the already downloaded mods.

We'll click on the Mods check box to enable it, and click on Remove Selected Mods



If we just wanted to delete a few, we could have just selected the ones we wanted to remove, and then click on Remove Selected Mods

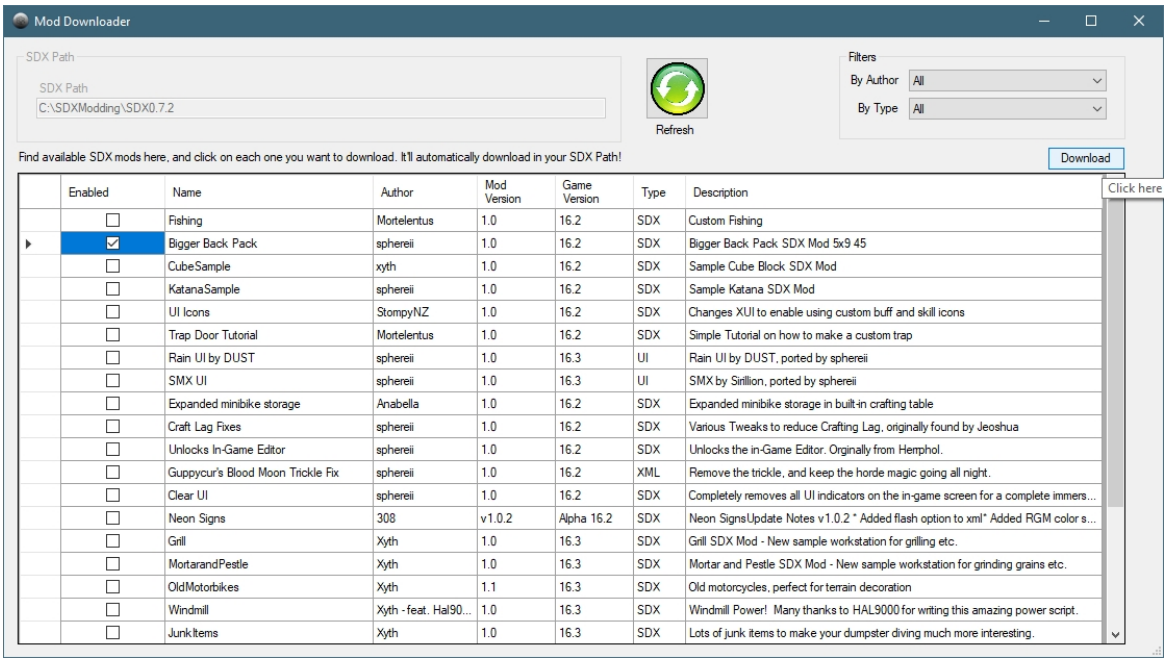
Click on the "Search For More SDX Mods"



Enabled	Name	Author	Mod Version	Game Version	Type	Description
☐	Fishing	Mortelentus	1.0	16.2	SDX	Custom Fishing
☐	Bigger Back Pack	sphereii	1.0	16.2	SDX	Bigger Back Pack SDX Mod 5x9 45
☐	CubeSample	xyth	1.0	16.2	SDX	Sample Cube Block SDX Mod
☐	KatanaSample	sphereii	1.0	16.2	SDX	Sample Katana SDX Mod
☐	UI Icons	StompyNZ	1.0	16.2	SDX	Changes XUI to enable using custom buff and skill icons
☐	Trap Door Tutorial	Mortelentus	1.0	16.2	SDX	Simple Tutorial on how to make a custom trap
☐	Rain UI by DUST	sphereii	1.0	16.3	UI	Rain UI by DUST, ported by sphereii
☐	SMX UI	sphereii	1.0	16.3	UI	SMX by Sirillion, ported by sphereii
☐	Expanded minibike storage	Anabella	1.0	16.2	SDX	Expanded minibike storage in built-in crafting table
☐	Craft Lag Fixes	sphereii	1.0	16.2	SDX	Various Tweaks to reduce Crafting Lag, originally found by Jeoshua
☐	Unlocks In-Game Editor	sphereii	1.0	16.2	SDX	Unlocks the in-Game Editor. Orginally from Herphol.
☐	Guppycur's Blood Moon Trickle Fix	sphereii	1.0	16.2	XML	Remove the trickle, and keep the horde magic going all night.
☐	Clear UI	sphereii	1.0	16.2	SDX	Completely removes all UI indicators on the in-game screen for a complete immers...
☐	Neon Signs	308	v1.0.2	Alpha 16.2	SDX	Neon SignsUpdate Notes v1.0.2 * Added flash option to xml* Added RGM color s...
☐	Grill	Xyth	1.0	16.3	SDX	Grill SDX Mod - New sample workstation for grilling etc.
☐	MortarandPestle	Xyth	1.0	16.3	SDX	Mortar and Pestle SDX Mod - New sample workstation for grinding grains etc.
☐	OldMotorbikes	Xyth	1.1	16.3	SDX	Old motorcycles, perfect for terrain decoration
☐	Windmill	Xyth - feat. Hal90...	1.0	16.3	SDX	Windmill Power! Many thanks to HAL9000 for writing this amazing power script.
☐	JunkItems	Xyth	1.0	16.3	SDX	Lots of junk items to make your dumpster diving much more interesting.

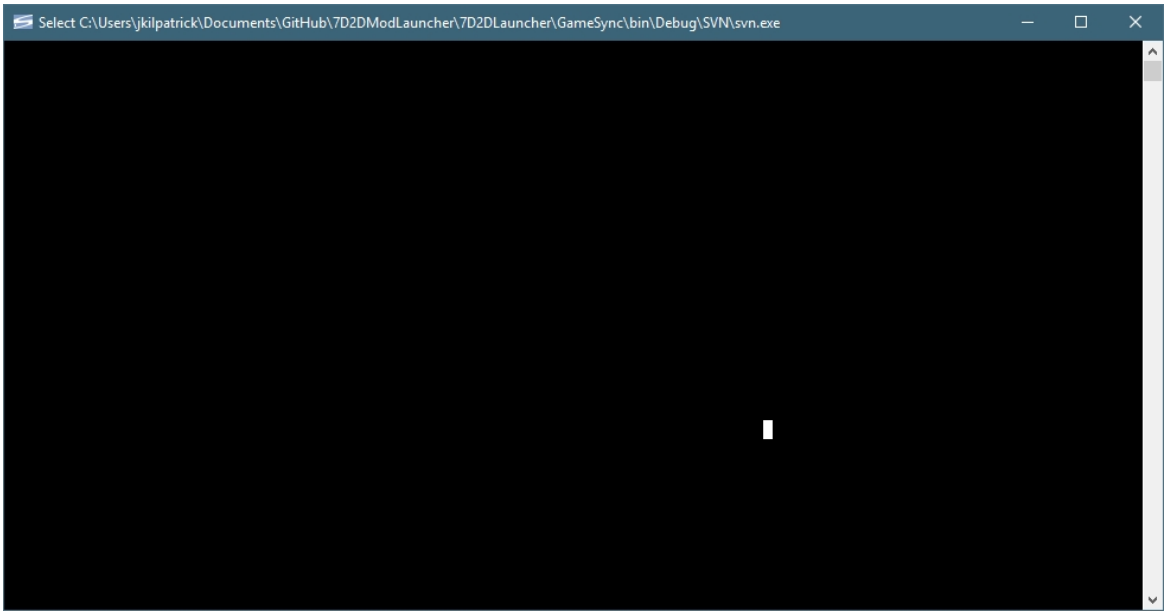
After a few seconds, the Mod listing will appear.

For this example, we are just going to select the Bigger Back Pack mod, and click on Download.

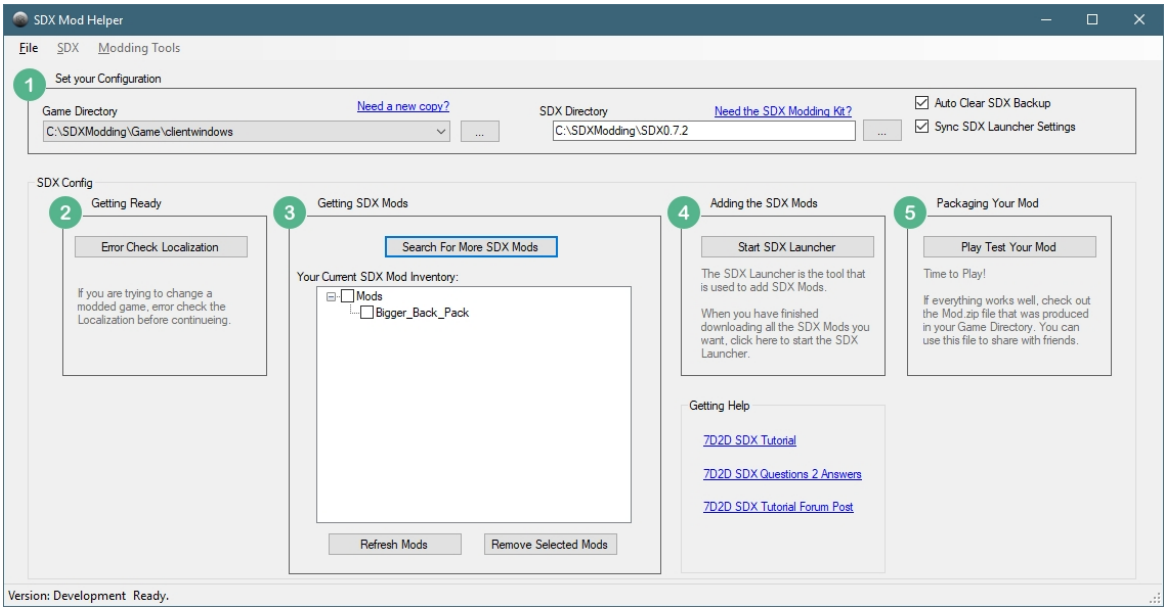


Enabled	Name	Author	Mod Version	Game Version	Type	Description
☐	Fishing	Mortelentus	1.0	16.2	SDX	Custom Fishing
☑	Bigger Back Pack	sphereii	1.0	16.2	SDX	Bigger Back Pack SDX Mod 5x9 45
☐	CubeSample	xyth	1.0	16.2	SDX	Sample Cube Block SDX Mod
☐	KatanaSample	sphereii	1.0	16.2	SDX	Sample Katana SDX Mod
☐	UI Icons	StompyNZ	1.0	16.2	SDX	Changes XUI to enable using custom buff and skill icons
☐	Trap Door Tutorial	Mortelentus	1.0	16.2	SDX	Simple Tutorial on how to make a custom trap
☐	Rain UI by DUST	sphereii	1.0	16.3	UI	Rain UI by DUST, ported by sphereii
☐	SMX UI	sphereii	1.0	16.3	UI	SMX by Sirillion, ported by sphereii
☐	Expanded minibike storage	Anabella	1.0	16.2	SDX	Expanded minibike storage in built-in crafting table
☐	Craft Lag Fixes	sphereii	1.0	16.2	SDX	Various Tweaks to reduce Crafting Lag, originally found by Jeoshua
☐	Unlocks In-Game Editor	sphereii	1.0	16.2	SDX	Unlocks the in-Game Editor. Orginally from Herphol.
☐	Guppycur's Blood Moon Trickle Fix	sphereii	1.0	16.2	XML	Remove the trickle, and keep the horde magic going all night.
☐	Clear UI	sphereii	1.0	16.2	SDX	Completely removes all UI indicators on the in-game screen for a complete immers...
☐	Neon Signs	308	v1.0.2	Alpha 16.2	SDX	Neon SignsUpdate Notes v1.0.2 * Added flash option to xml* Added RGM color s...
☐	Grill	Xyth	1.0	16.3	SDX	Grill SDX Mod - New sample workstation for grilling etc.
☐	MortarandPestle	Xyth	1.0	16.3	SDX	Mortar and Pestle SDX Mod - New sample workstation for grinding grains etc.
☐	OldMotorbikes	Xyth	1.1	16.3	SDX	Old motorcycles, perfect for terrain decoration
☐	Windmill	Xyth - feat. Hal90...	1.0	16.3	SDX	Windmill Power! Many thanks to HAL9000 for writing this amazing power script.
☐	JunkItems	Xyth	1.0	16.3	SDX	Lots of junk items to make your dumpster diving much more interesting.

For each mod that you have selected, you'll see a black box appear. If the mod is small, you may only see a quick flash.



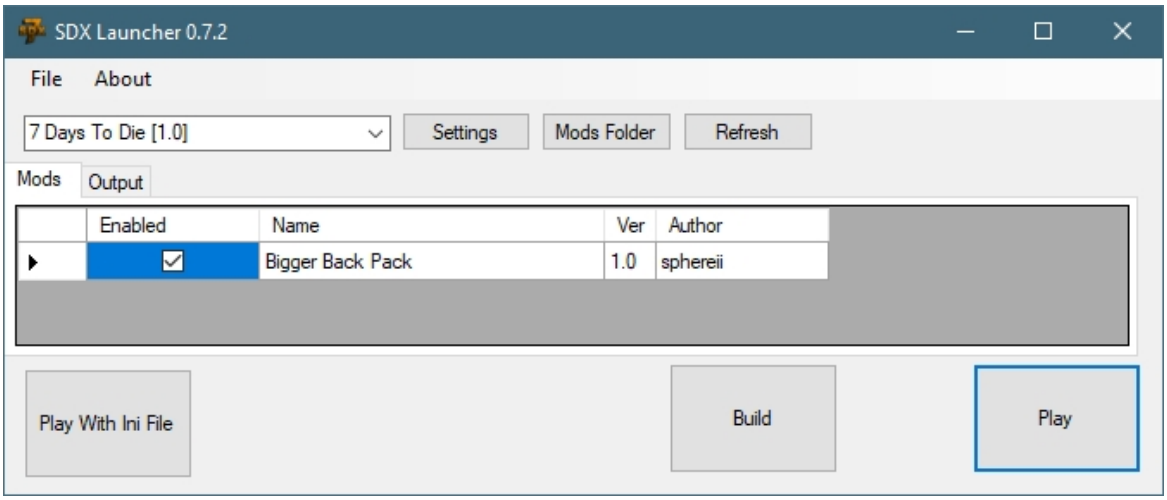
In the SDX Mod Helper main screen, you'll see your Current SDX Mod Inventory updated:



Running the SDX Launcher

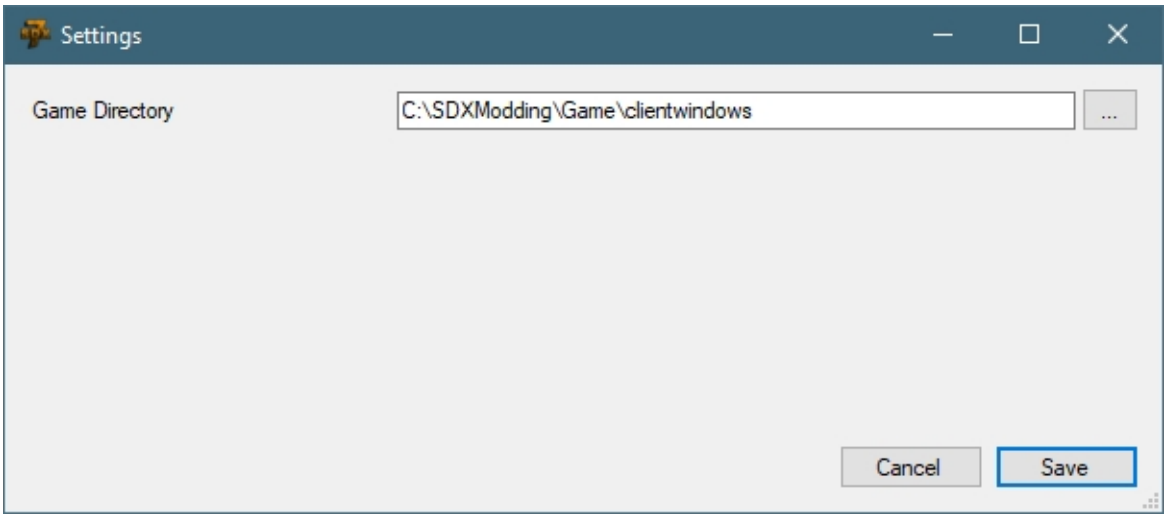
With your SDX Mod downloaded, it's time to add it to your Game Directory.

Click on the Start SDX Launcher



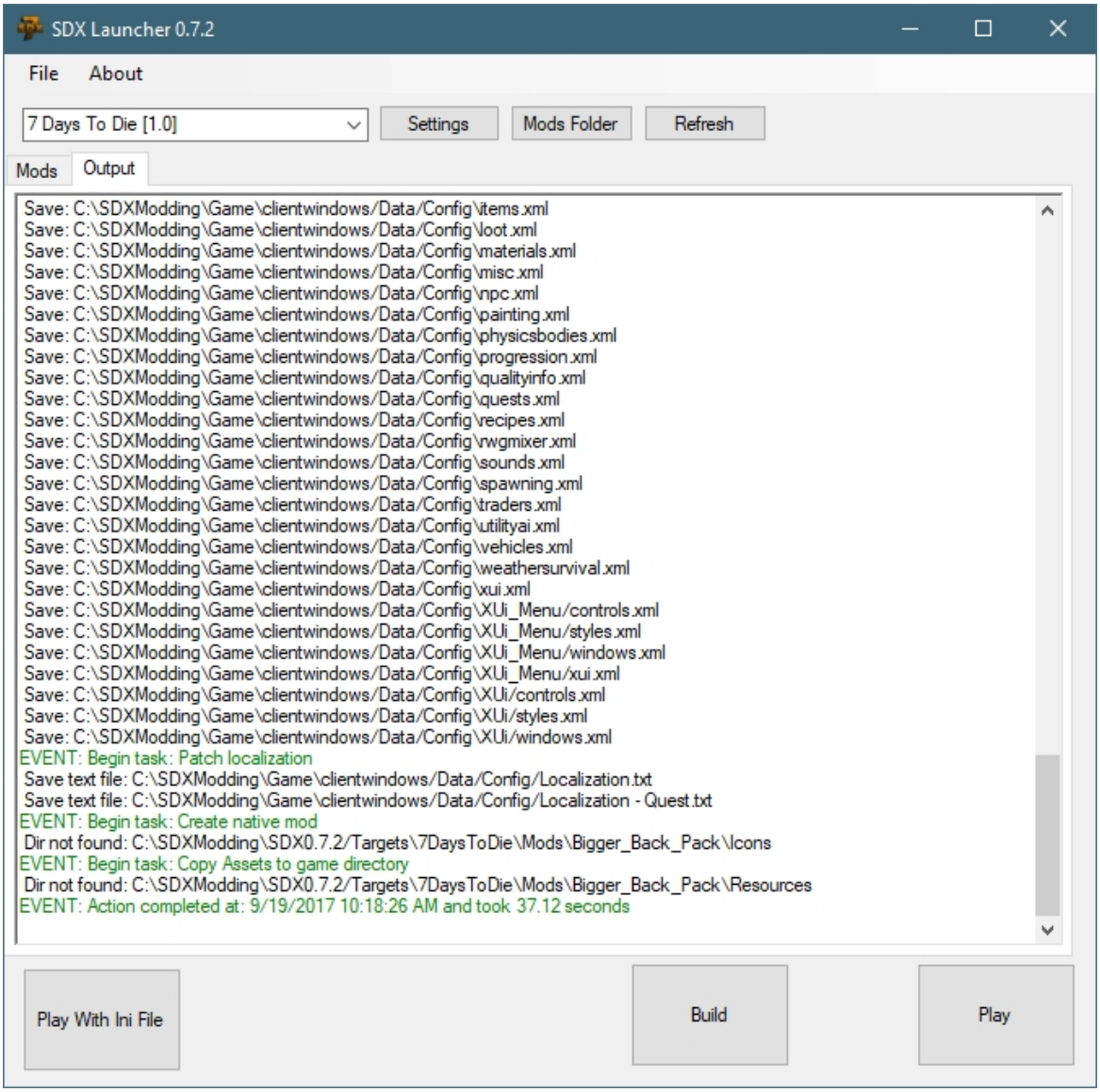
Click on the Settings button to update your Game Directory to point to your new vanilla install.

If you have "Sync SDX Launcher Settings" checked, this step is done for you.



Click on Save to save your changes, if any, and return to the SDX Launcher.

Click on the Build button on the SDX Launcher screen



You may scroll up to see if there was any errors or warnings.

You have two options now: Use the Play button in the SDX Launcher, or close the SDX Launcher and return to the SDX Mod Helper to Play.

What's the different?

The Play button in the SDX Launcher will launch the game, and you can begin your play test.

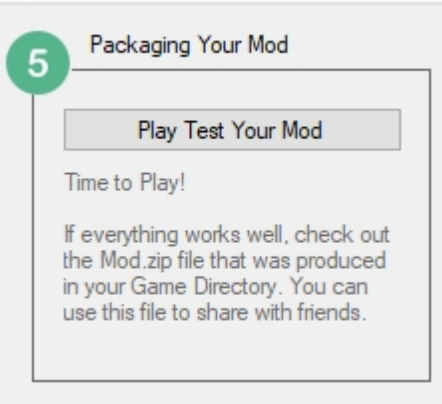
The Play button in the SDX Mod Helper does the same thing, however, it does a few more things as well. If you are merging a few mods together, then there could be duplicate ItemIcons under your Mods folder now. The game will put warnings and errors in your build log, complaining about the duplicate icons.

The SDX Mod Helper will automatically move the icons around, and get rid of duplicates for you.

The SDX Mod Helper will also generate a Mod.zip file. This zip file contains all the files that have been modified, and will allow you to easily share your mods with other people.

Play Testing the Game

The final step is to Package Your Game, and begin your Play test.



Click on the Play Test Your mod button to start the game.

What's the difference between this Play button and the Play button in the SDX Launcher?

The Play button in the SDX Launcher will launch the game, and you can begin your play test.

The Play button in the SDX Mod Helper does the same thing, however, it does a few more things as well. If you are merging a few mods together, then there could be duplicate ItemIcons under your Mods folder now. The game will put warnings and errors in your build log, complaining about the duplicate icons.

The SDX Mod Helper will automatically move the icons around, and get rid of duplicates for you.

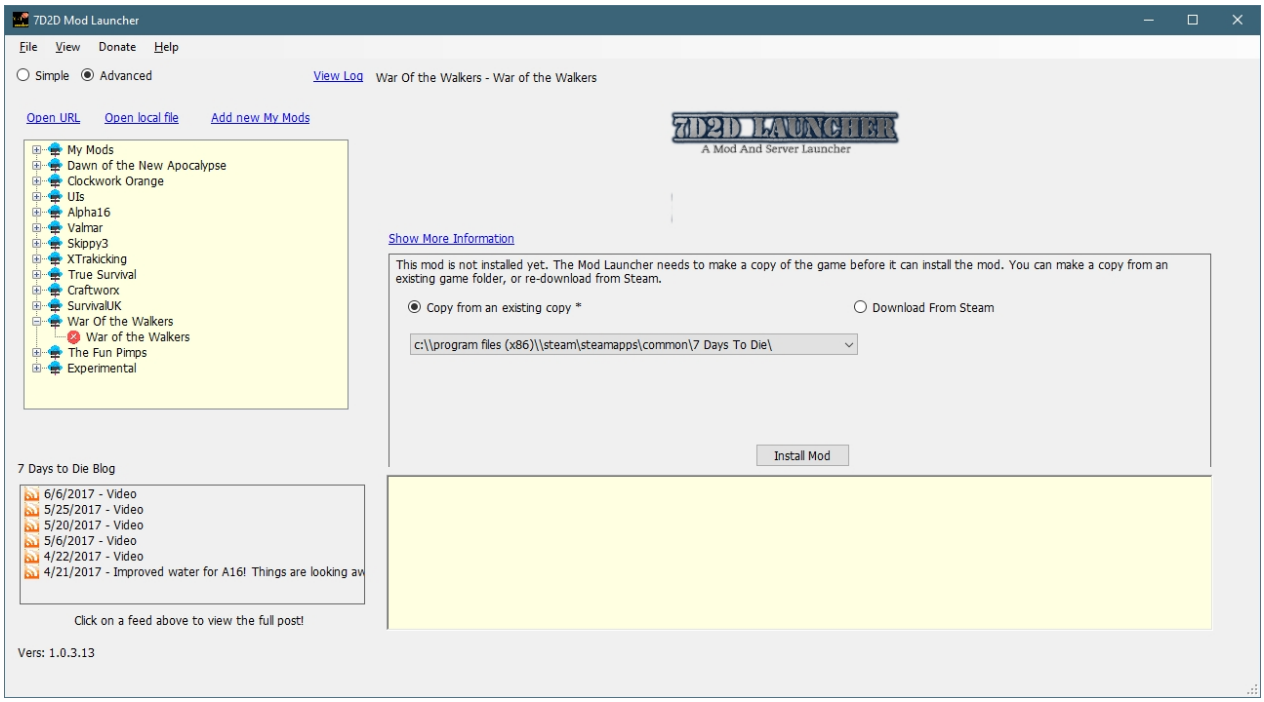
The SDX Mod Helper will also generate a Mod.zip file. This zip file contains all the files that have been modified, and will allow you to easily share your mods with other people.

Adding an SDX mod to a War of the Walkers Install

This guide will take you through the steps on how to download the War of the Walkers mod, and install the Bigger Back Pack mod, using the SDX Mod Helper.

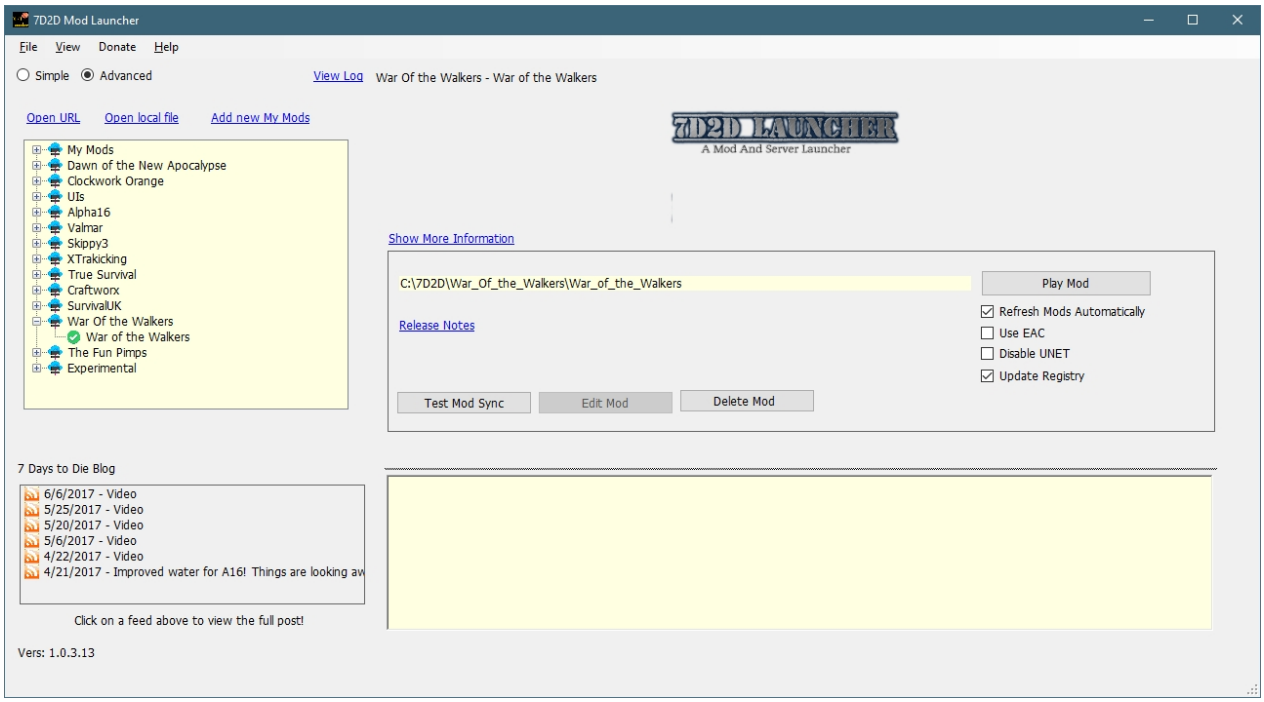
Installing a new Game Directory

We will be using the 7D2D Mod Launcher to install a clean version of the War of the Walkers Mod.



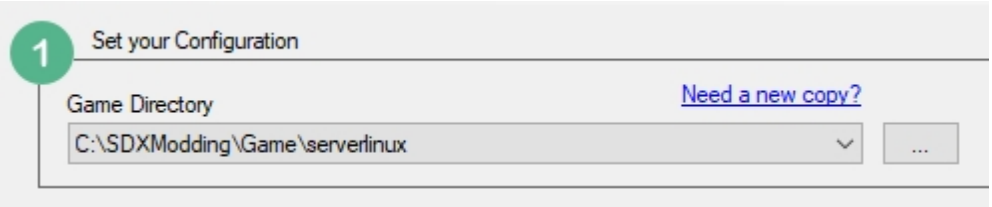
Click on Install Mod to make a copy from your clean Steam folder.

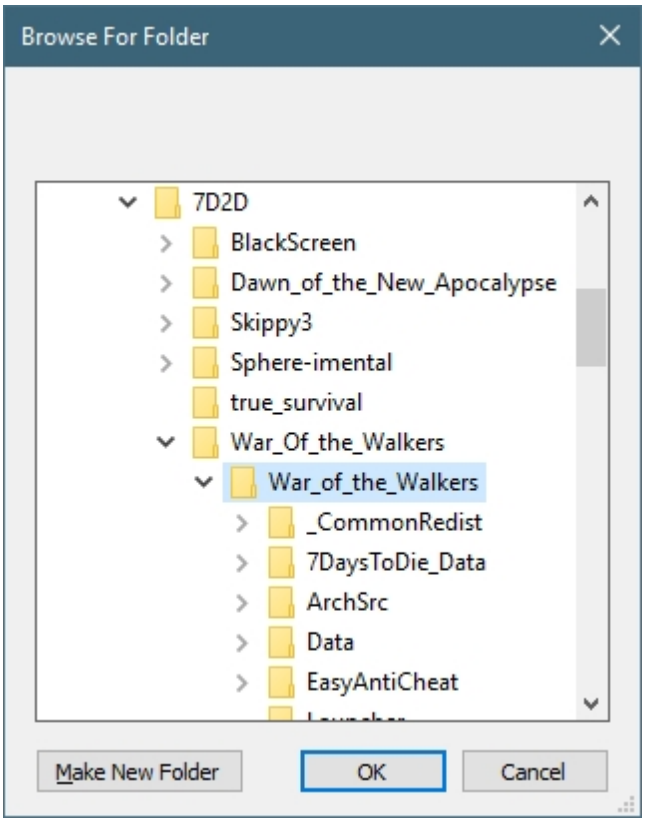
Once installed, click on the Play button in the 7D2D Mod Launcher to start the game for the first time.



This will install the mod correctly. When the game is loaded, you may exit the game and resume the tutorial.

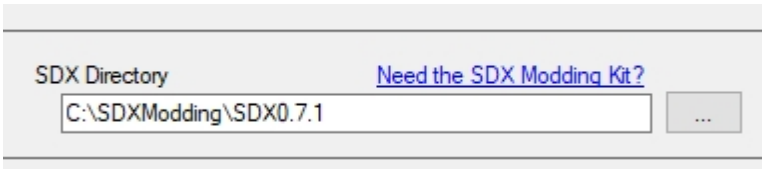
In the Set your Configuration, in the 7D2D Mod Helper, click on the button with the three dots and navigate to the War of the Walkers folder.



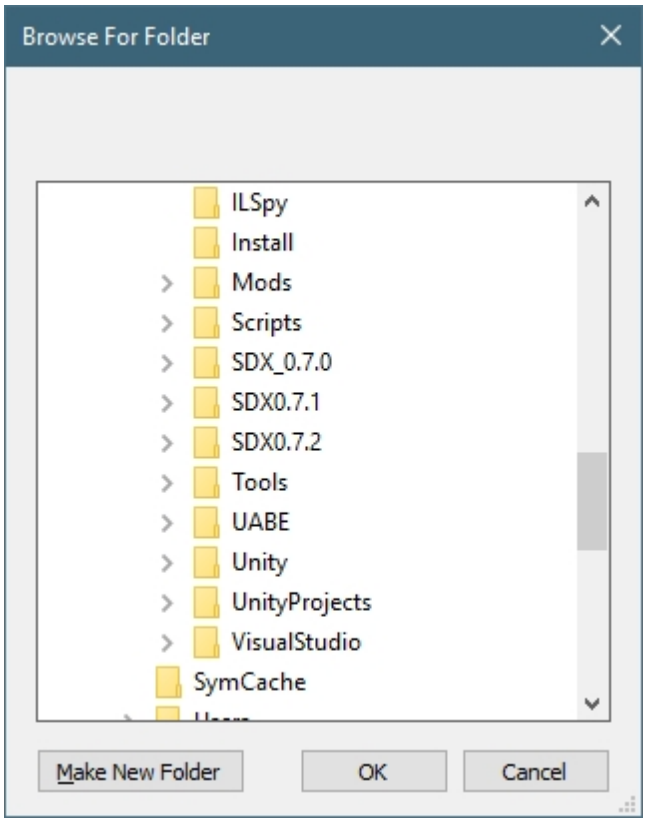


Updating the SDX Path

Set your SDX Directory



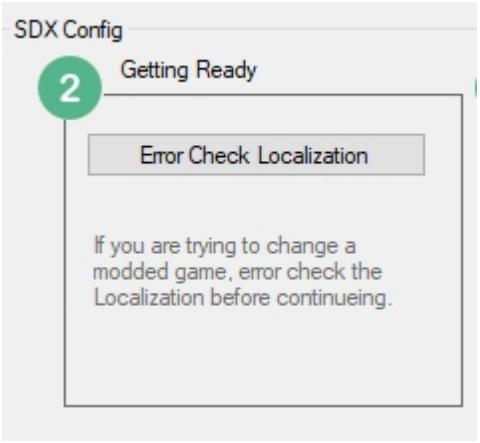
If the SDX Directory isn't correct, click on the button with the 3 dots to open up a folder browser



Then select which folder you want, and click on OK

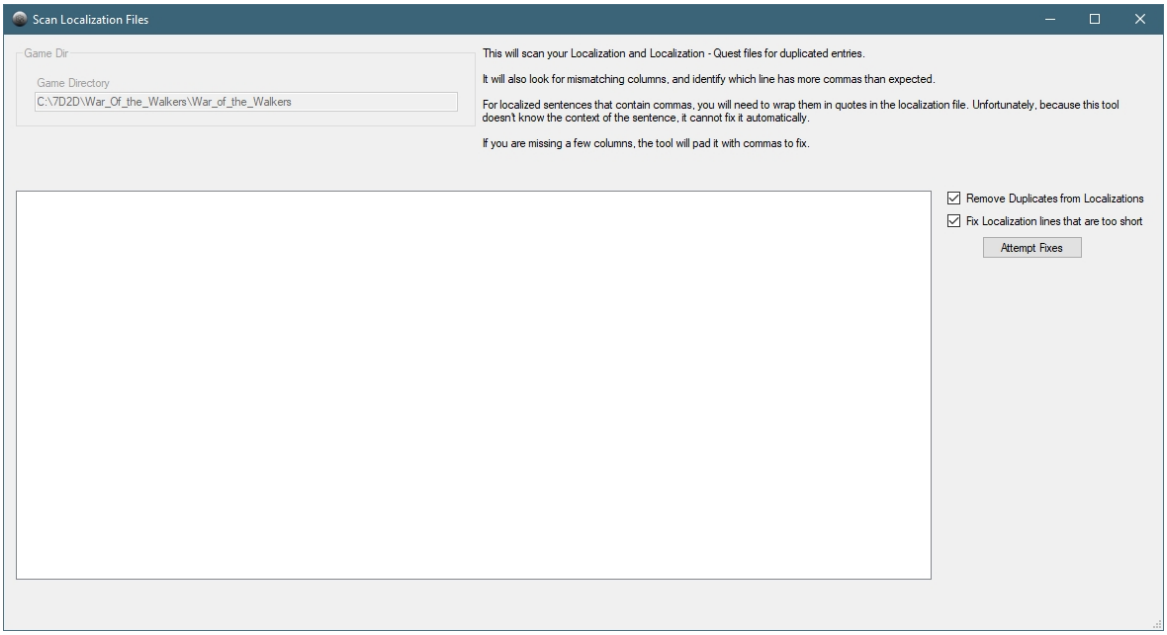
Error Checking Localization

Because we are using a mod (War of the Walkers) for this part of the tutorial, you will want to run the Error Check Localization.



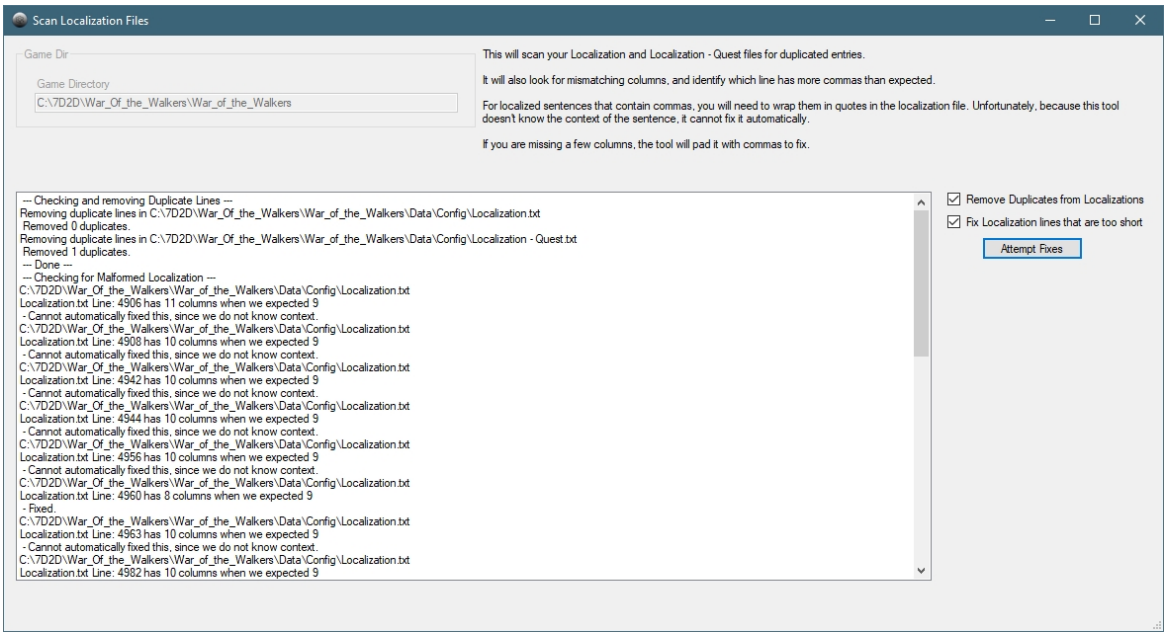
If you have any errors in the localization files, SDX will fail.

Click on the Error Check Localization file to open the Scan Localization Files screen.



Verify that the Game Directory is set correctly. If not, go back to the SDX Mod Helper main screen and change it there.

Click on the Attempt Fixes button.



It found a duplicate in the Localization - Quest.txt and removed it.

It did find some malformed entries in the Localization.txt file.

"Localization.txt Line 4905 has 11 columns when we expected 9"

This is one of the offending lines:

```
TungstenFragmentDesc,items,Item,KgNone,A advanced metal that can be used in the forge to make Tungsten Items,Un métal avancé qui peut être utilisé dans la forge pour faire des objets de tungstène,Ein vorgerücktes Metall, das in der Schmiede verwendet werden kann, um Wolfram-Einzelteile herzustellen,,Un metal avanzado que se puede utilizar en la fragua para hacer artículos de tungsteno
```

When we look closely at the file, we see this:

```
TungstenFragmentDesc,items,Item,KgNone,A advanced metal that can be used in
the forge to make Tungsten Items,Un métal avancé qui peut être utilisé dans la
forge pour faire des objets de tungstène,Ein vorgerücktes Metall, das in der
Schmiede verwendet werden kann, um Wolfram-Einzelteile herzustellen,,Un metal
avanzado que se puede utilizar en la fragua para hacer artículos de tungsteno
```

In this case, it's the German translation for TungstenFragment. Notice the commas with the space after it? This indicates that it should be wrapped around in quotes, as the commas are actually part of the sentence structure.

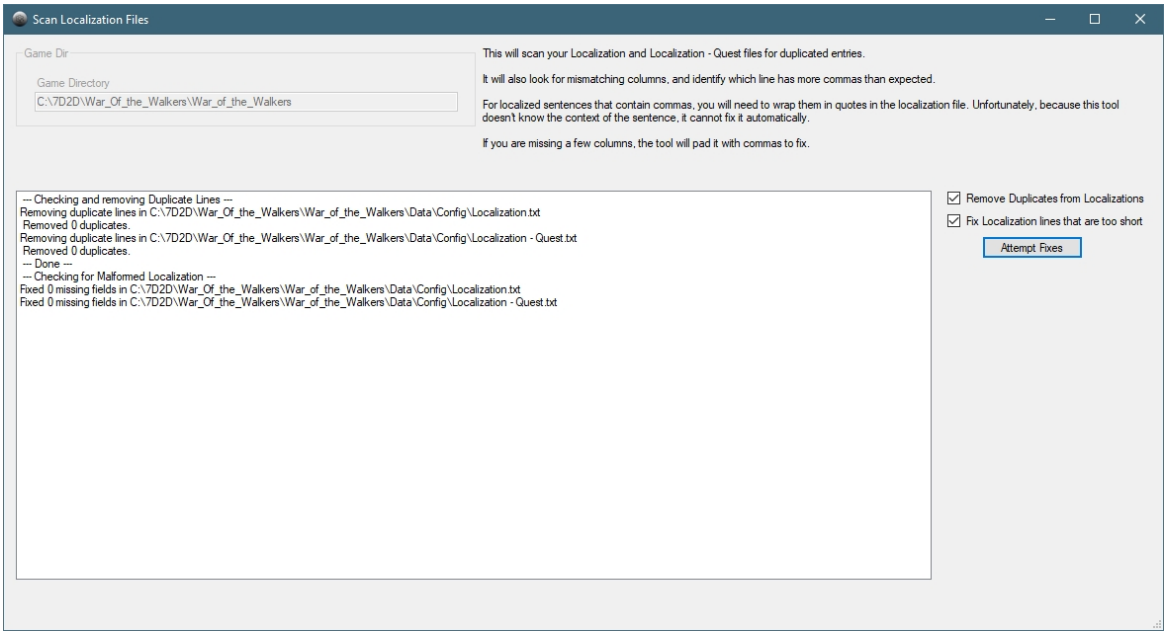
You will need to fix each of those individually.

The other localization error it picked up says this:

"Localilzation.txt Line: 4960 has 8 columns when we expected 9"

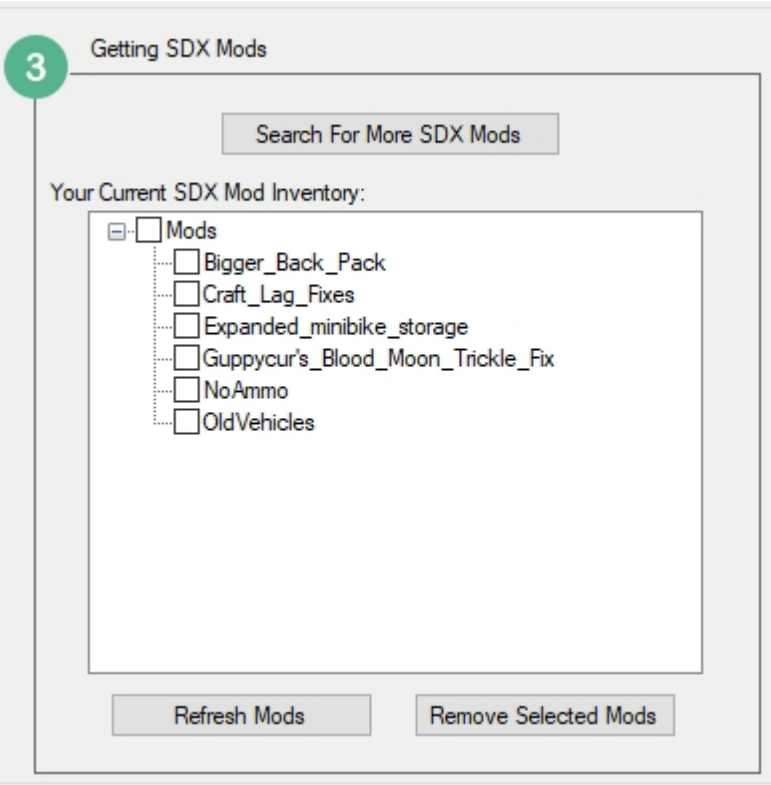
That means there is probably a missing comma. The SDX Mod Helper has padded this line with an extra comma (or more, depending on the number of missing commas).

After you have fixed the Localization file, re-run the Scan Localization Files.



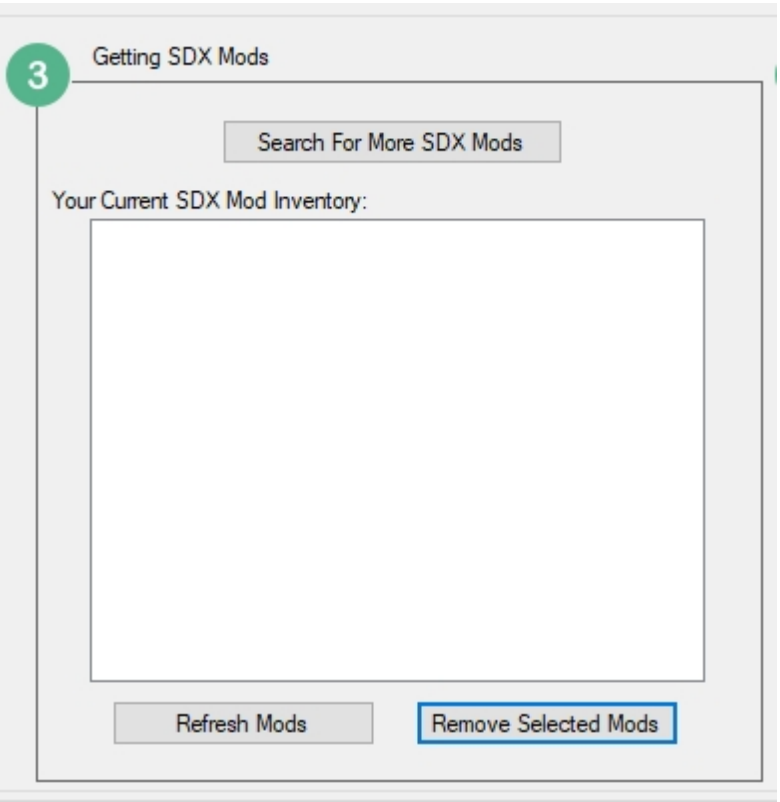
Downloading SDX Mods

We now want to find which SDX Mods we want to add.



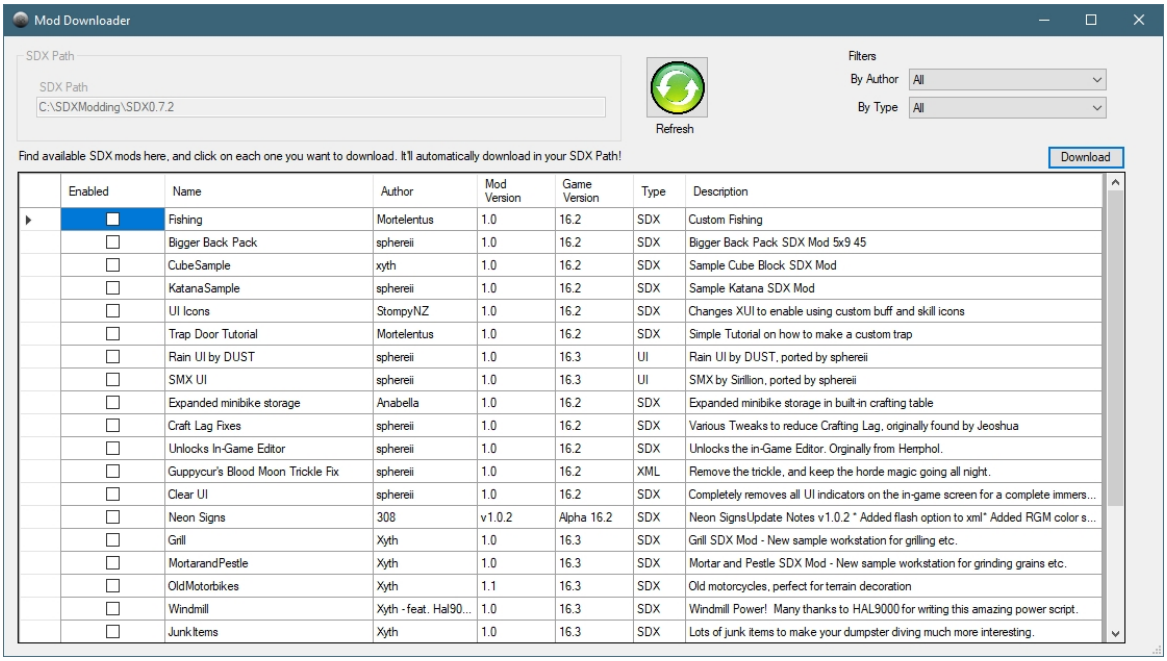
In the above, I have some Mods already downloaded. For this step, we are going to remove the already downloaded mods.

We'll click on the Mods check box to enable it, and click on Remove Selected Mods



If we just wanted to delete a few, we could have just selected the ones we wanted to remove, and then click on Remove Selected Mods

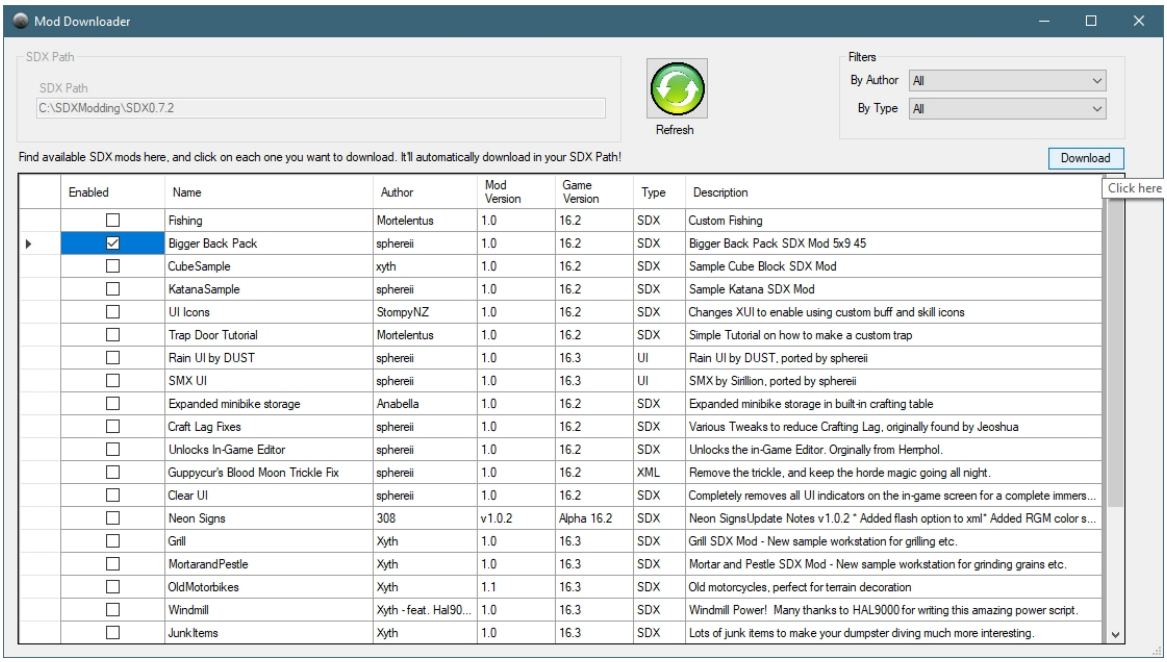
Click on the "Search For More SDX Mods"



Enabled	Name	Author	Mod Version	Game Version	Type	Description
☐	Fishing	Mortelentus	1.0	16.2	SDX	Custom Fishing
☐	Bigger Back Pack	sphereii	1.0	16.2	SDX	Bigger Back Pack SDX Mod 5x9 45
☐	CubeSample	xyth	1.0	16.2	SDX	Sample Cube Block SDX Mod
☐	KatanaSample	sphereii	1.0	16.2	SDX	Sample Katana SDX Mod
☐	UI Icons	StompyNZ	1.0	16.2	SDX	Changes XUI to enable using custom buff and skill icons
☐	Trap Door Tutorial	Mortelentus	1.0	16.2	SDX	Simple Tutorial on how to make a custom trap
☐	Rain UI by DUST	sphereii	1.0	16.3	UI	Rain UI by DUST, ported by sphereii
☐	SMX UI	sphereii	1.0	16.3	UI	SMX by Sirillion, ported by sphereii
☐	Expanded minibike storage	Anabella	1.0	16.2	SDX	Expanded minibike storage in built-in crafting table
☐	Craft Lag Fixes	sphereii	1.0	16.2	SDX	Various Tweaks to reduce Crafting Lag, originally found by Jeoshua
☐	Unlocks In-Game Editor	sphereii	1.0	16.2	SDX	Unlocks the in-Game Editor. Orginally from Herphol.
☐	Guppycur's Blood Moon Trickle Fix	sphereii	1.0	16.2	XML	Remove the trickle, and keep the horde magic going all night.
☐	Clear UI	sphereii	1.0	16.2	SDX	Completely removes all UI indicators on the in-game screen for a complete immers...
☐	Neon Signs	308	v1.0.2	Alpha 16.2	SDX	Neon SignsUpdate Notes v1.0.2 * Added flash option to xml* Added RGM color s...
☐	Grill	Xyth	1.0	16.3	SDX	Grill SDX Mod - New sample workstation for grilling etc.
☐	MortarandPestle	Xyth	1.0	16.3	SDX	Mortar and Pestle SDX Mod - New sample workstation for grinding grains etc.
☐	OldMotorbikes	Xyth	1.1	16.3	SDX	Old motorcycles, perfect for terrain decoration
☐	Windmill	Xyth - feat. Hal90...	1.0	16.3	SDX	Windmill Power! Many thanks to HAL9000 for writing this amazing power script.
☐	JunkItems	Xyth	1.0	16.3	SDX	Lots of junk items to make your dumpster diving much more interesting.

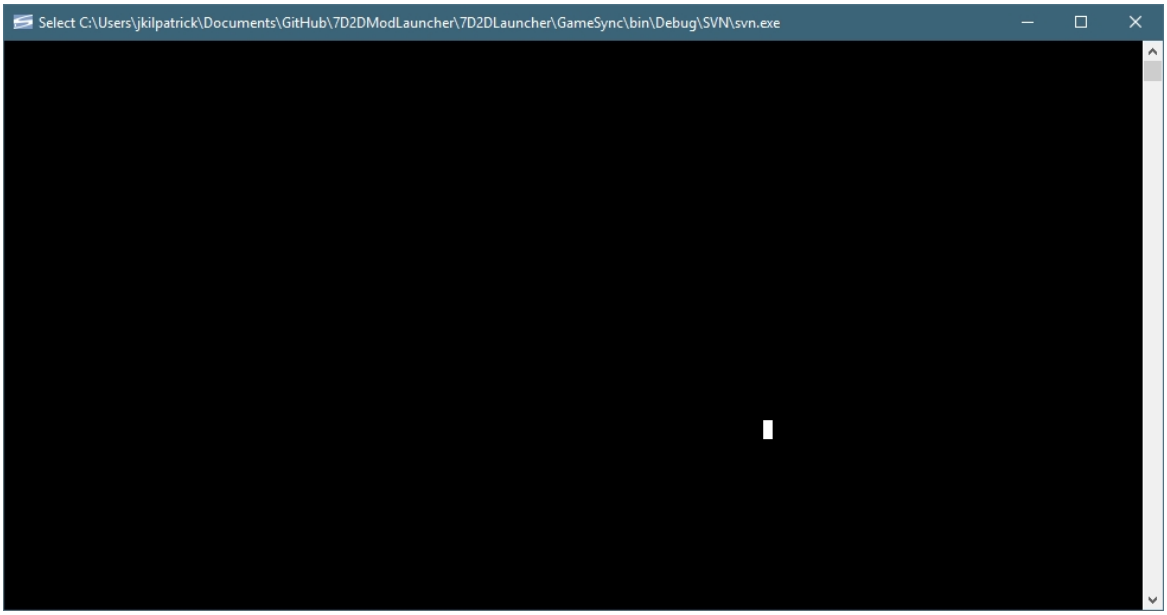
After a few seconds, the Mod listing will appear.

For this example, we are just going to select the Bigger Back Pack mod, and click on Download.

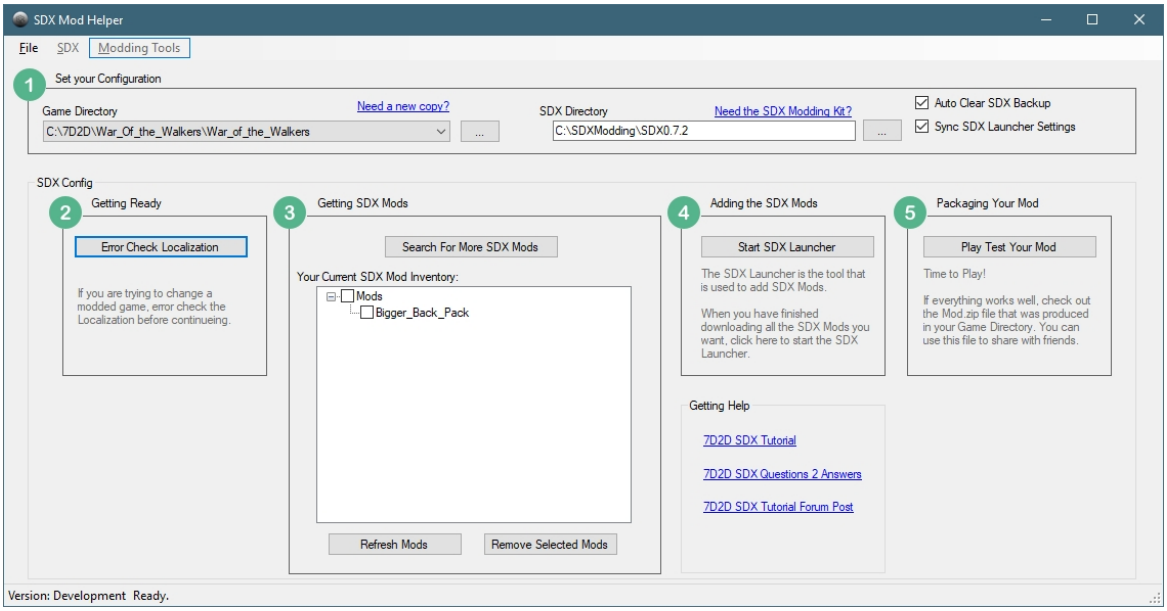


Enabled	Name	Author	Mod Version	Game Version	Type	Description
☐	Fishing	Mortelentus	1.0	16.2	SDX	Custom Fishing
☑	Bigger Back Pack	sphereii	1.0	16.2	SDX	Bigger Back Pack SDX Mod 5x9 45
☐	CubeSample	xyth	1.0	16.2	SDX	Sample Cube Block SDX Mod
☐	KatanaSample	sphereii	1.0	16.2	SDX	Sample Katana SDX Mod
☐	UI Icons	StompyNZ	1.0	16.2	SDX	Changes XUI to enable using custom buff and skill icons
☐	Trap Door Tutorial	Mortelentus	1.0	16.2	SDX	Simple Tutorial on how to make a custom trap
☐	Rain UI by DUST	sphereii	1.0	16.3	UI	Rain UI by DUST, ported by sphereii
☐	SMX UI	sphereii	1.0	16.3	UI	SMX by Sirillion, ported by sphereii
☐	Expanded minibike storage	Anabella	1.0	16.2	SDX	Expanded minibike storage in built-in crafting table
☐	Craft Lag Fixes	sphereii	1.0	16.2	SDX	Various Tweaks to reduce Crafting Lag, originally found by Jeoshua
☐	Unlocks In-Game Editor	sphereii	1.0	16.2	SDX	Unlocks the in-Game Editor. Orginally from Herphol.
☐	Guppycur's Blood Moon Trickle Fix	sphereii	1.0	16.2	XML	Remove the trickle, and keep the horde magic going all night.
☐	Clear UI	sphereii	1.0	16.2	SDX	Completely removes all UI indicators on the in-game screen for a complete immers...
☐	Neon Signs	308	v1.0.2	Alpha 16.2	SDX	Neon SignsUpdate Notes v1.0.2 * Added flash option to xml* Added RGM color s...
☐	Grill	Xyth	1.0	16.3	SDX	Grill SDX Mod - New sample workstation for grilling etc.
☐	MortarandPestle	Xyth	1.0	16.3	SDX	Mortar and Pestle SDX Mod - New sample workstation for grinding grains etc.
☐	OldMotorbikes	Xyth	1.1	16.3	SDX	Old motorcycles, perfect for terrain decoration
☐	Windmill	Xyth - feat. Hal90...	1.0	16.3	SDX	Windmill Power! Many thanks to HAL9000 for writing this amazing power script.
☐	JunkItems	Xyth	1.0	16.3	SDX	Lots of junk items to make your dumpster diving much more interesting.

For each mod that you have selected, you'll see a black box appear. If the mod is small, you may only see a quick flash.



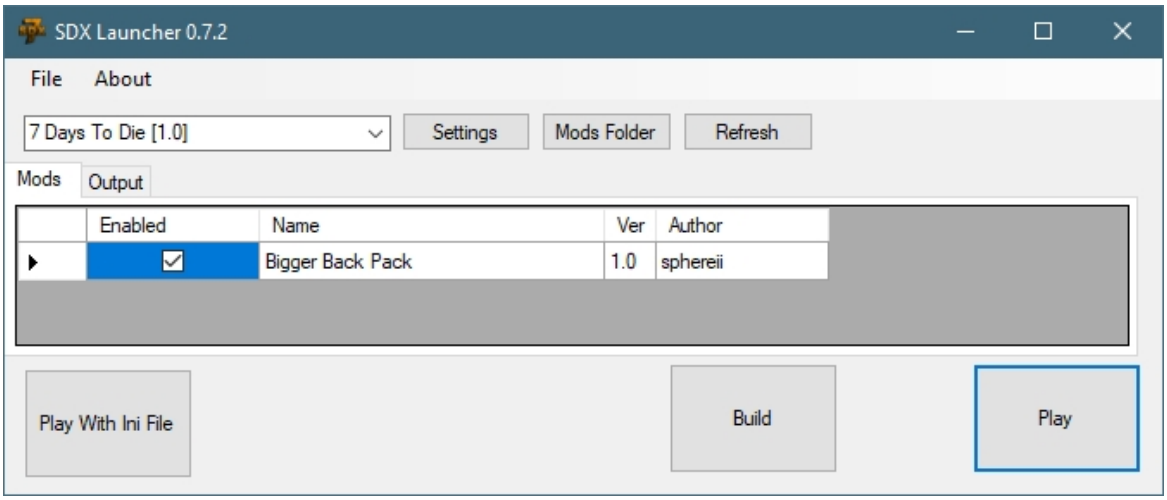
In the SDX Mod Helper main screen, you'll see your Current SDX Mod Inventory updated:



Running the SDX Launcher

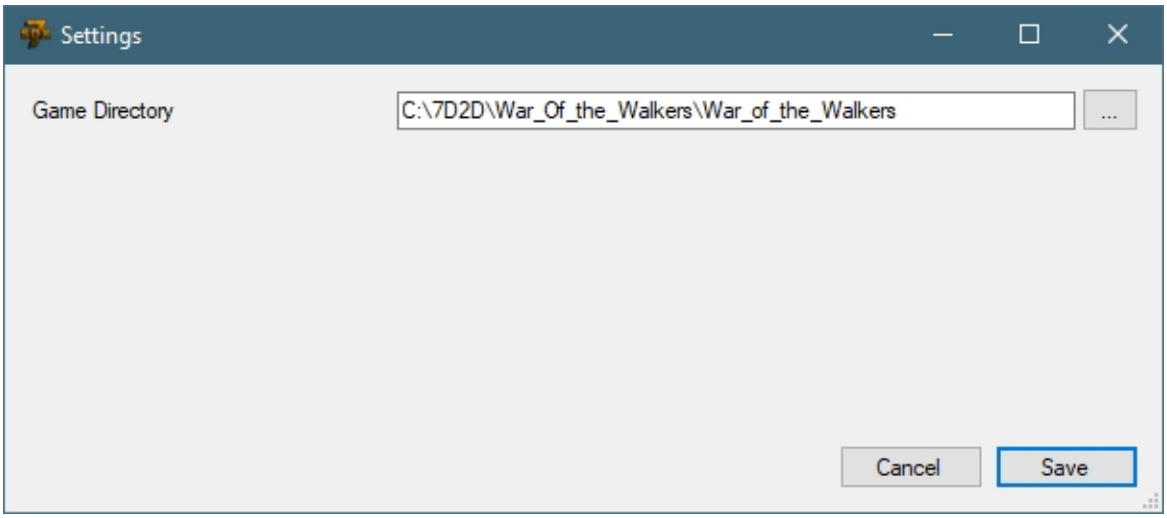
With your SDX Mod downloaded, it's time to add it to your Game Directory.

Click on the Start SDX Launcher



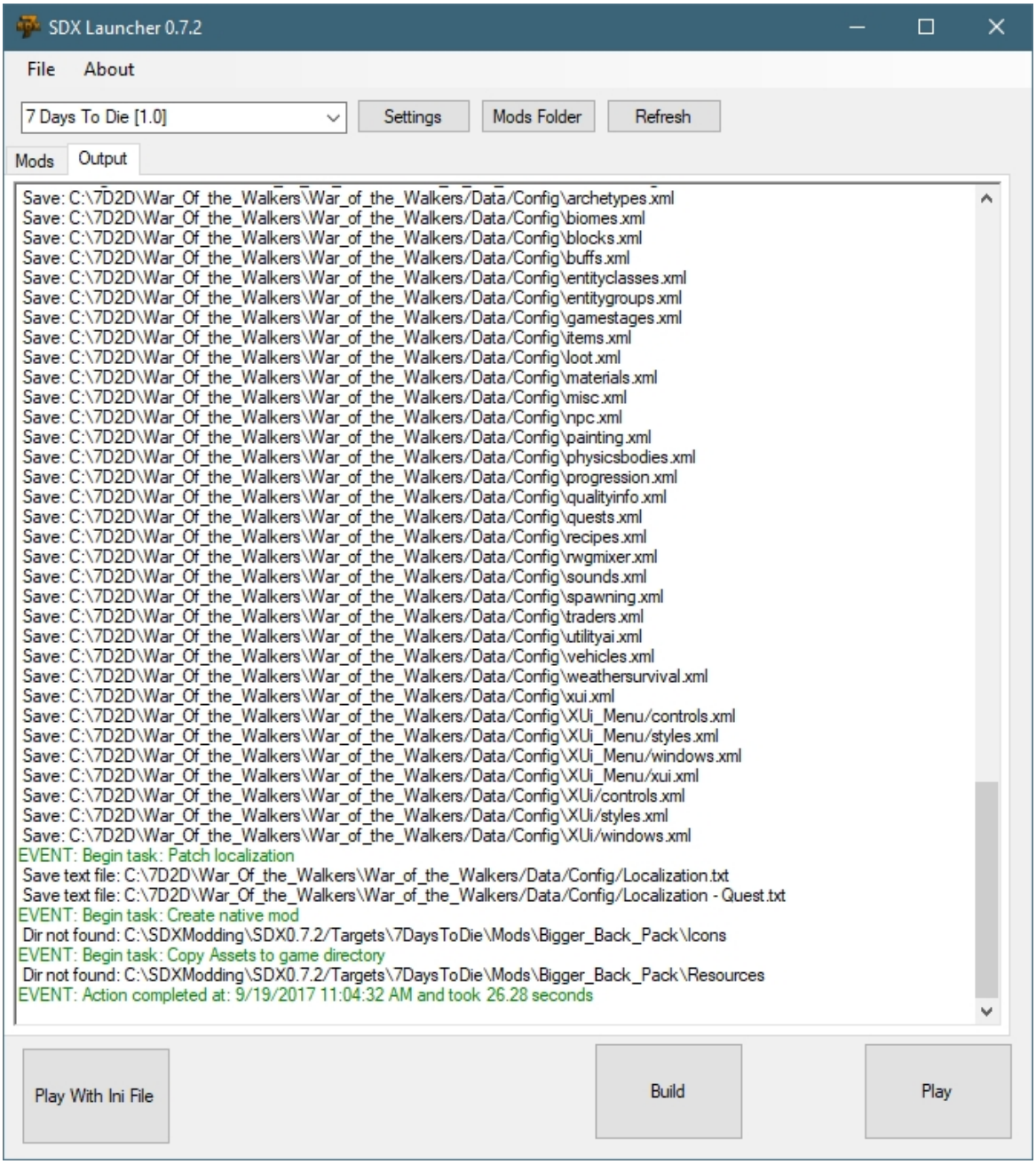
Click on the Settings button to update your Game Directory to point to your War of the Walker.

If you have "Sync SDX Launcher Settings" checked, this step is done for you.



Click on Save to save your changes, if any, and return to the SDX Launcher.

Click on the Build button on the SDX Launcher screen



You may scroll up to see if there was any errors or warnings.

You have two options now: Use the Play button in the SDX Launcher, or close the SDX Launcher and return to the SDX Mod Helper to Play.

What's the different?

The Play button in the SDX Launcher will launch the game, and you can begin your play test.

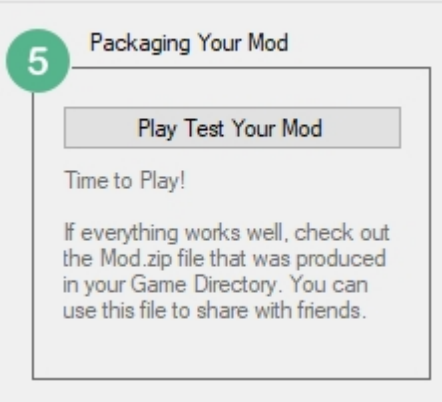
The Play button in the SDX Mod Helper does the same thing, however, it does a few more things as well. If you are merging a few mods together, then there could be duplicate ItemIcons under your Mods folder now. The game will put warnings and errors in your build log, complaining about the duplicate icons.

The SDX Mod Helper will automatically move the icons around, and get rid of duplicates for you.

The SDX Mod Helper will also generate a Mod.zip file. This zip file contains all the files that have been modified, and will allow you to easily share your mods with other people.

Play Testing the Game

The final step is to Package Your Game, and begin your Play test.



Click on the Play Test Your mod button to start the game.

What's the difference between this Play button and the Play button in the SDX Launcher?

The Play button in the SDX Launcher will launch the game, and you can begin your play test.

The Play button in the SDX Mod Helper does the same thing, however, it does a few more things as well. If you are merging a few mods together, then there could be duplicate ItemIcons under your Mods folder now. The game will put warnings and errors in your build log, complaining about the duplicate icons.

The SDX Mod Helper will automatically move the icons around, and get rid of duplicates for you.

The SDX Mod Helper will also generate a Mod.zip file. This zip file contains all the files that have been modified, and will allow you to easily share your mods with other people.

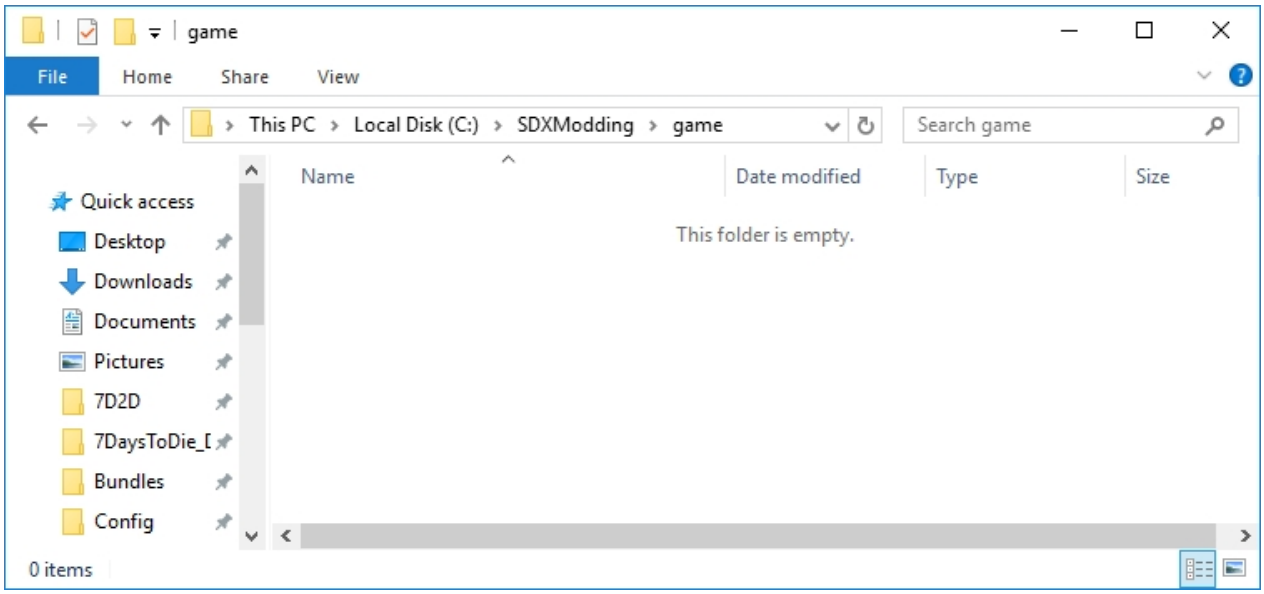
How to set up a build environment for Client and Servers

So you want to setup an environment where you can add SDX to a mod for Client and Server?

This will let you set up and configure your SDX mods for both Client and Server uploads.

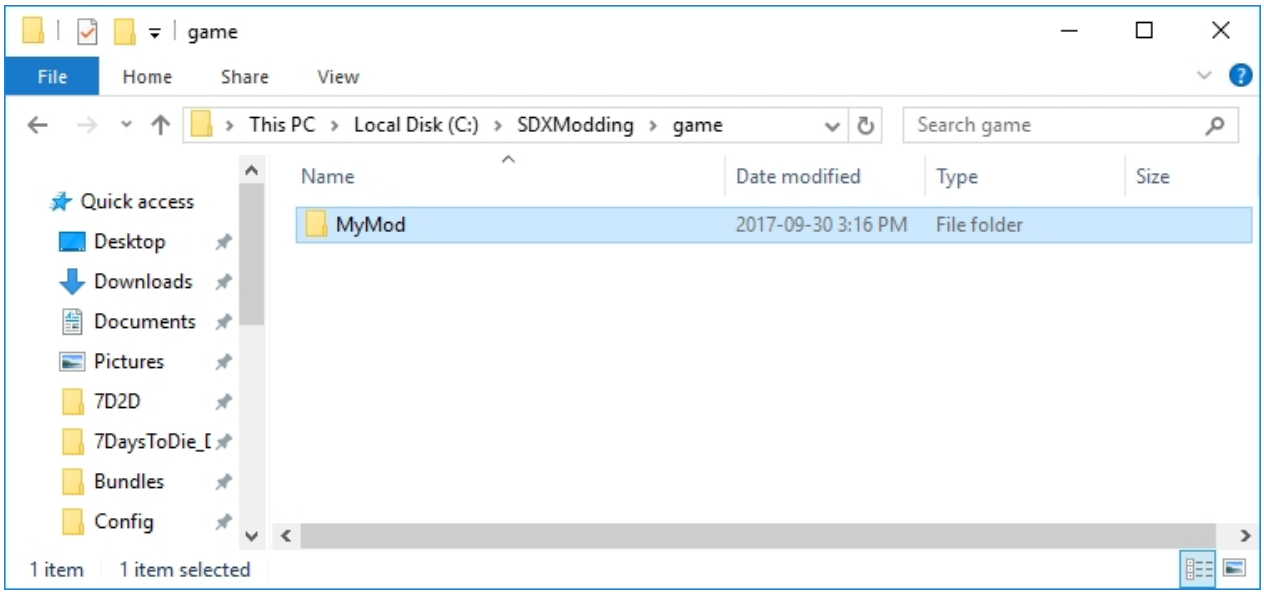
Making a new top level folder

For this example, we are going to set up a folder structure under the SDXModding folder, in the game folder:



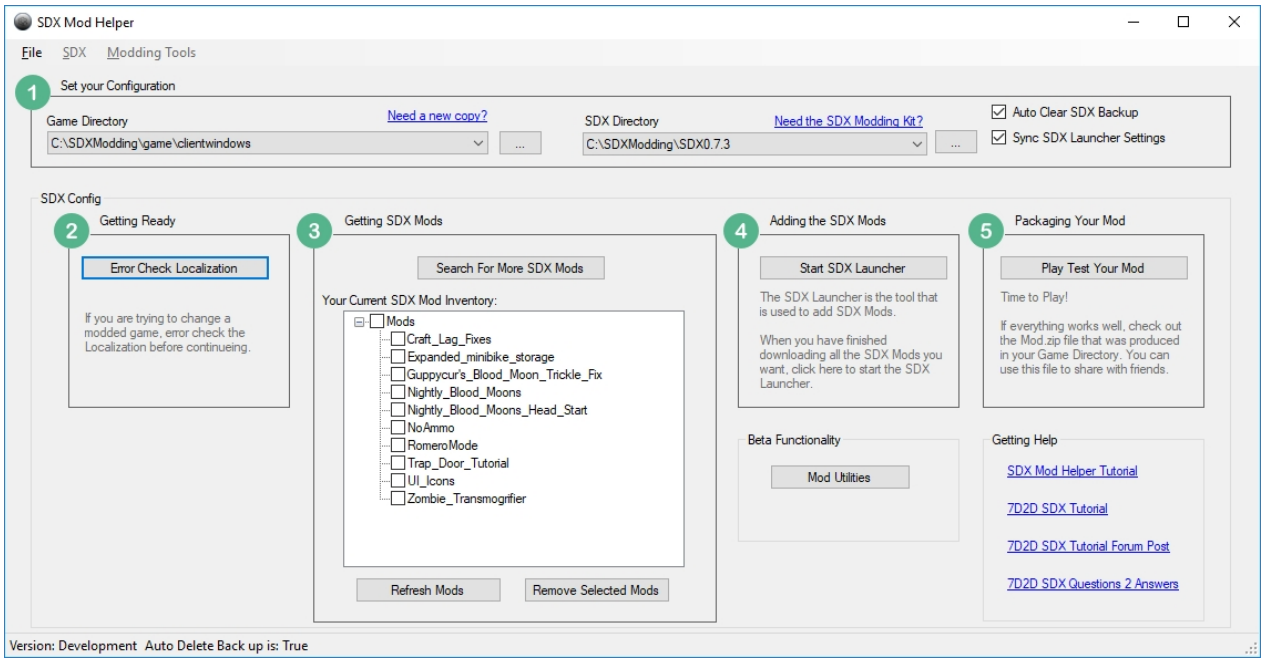
Let's create a new folder, by right clicking on the background, and selecting "New", then "Folder"

We'll name it "MyMod"

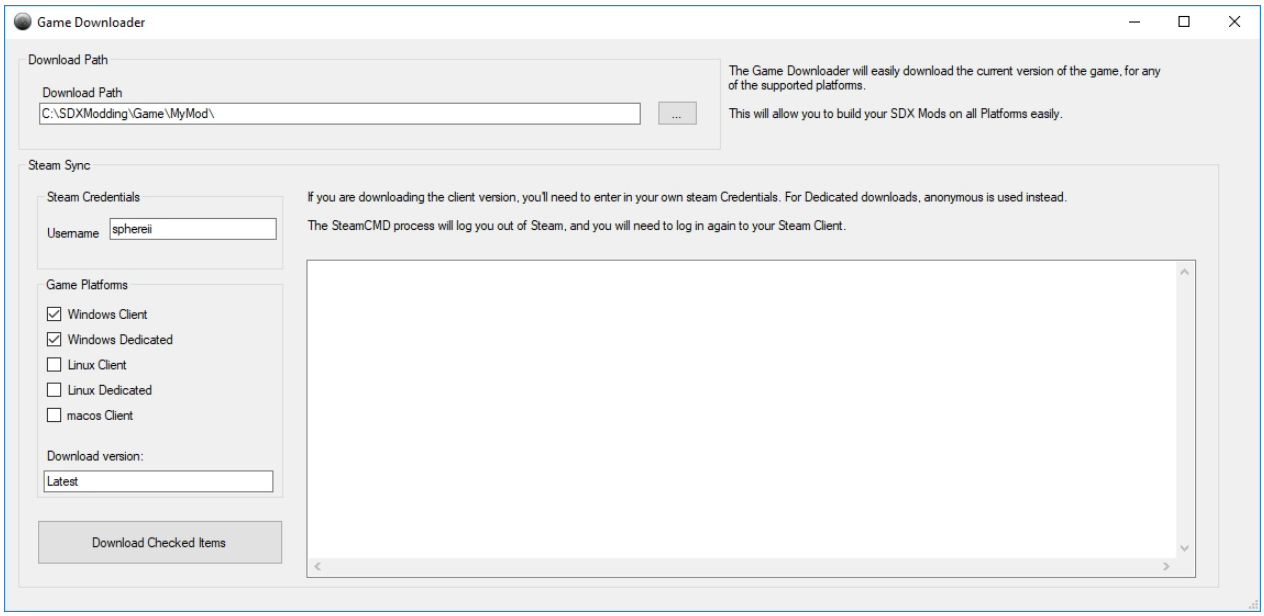


Making a copy of the game

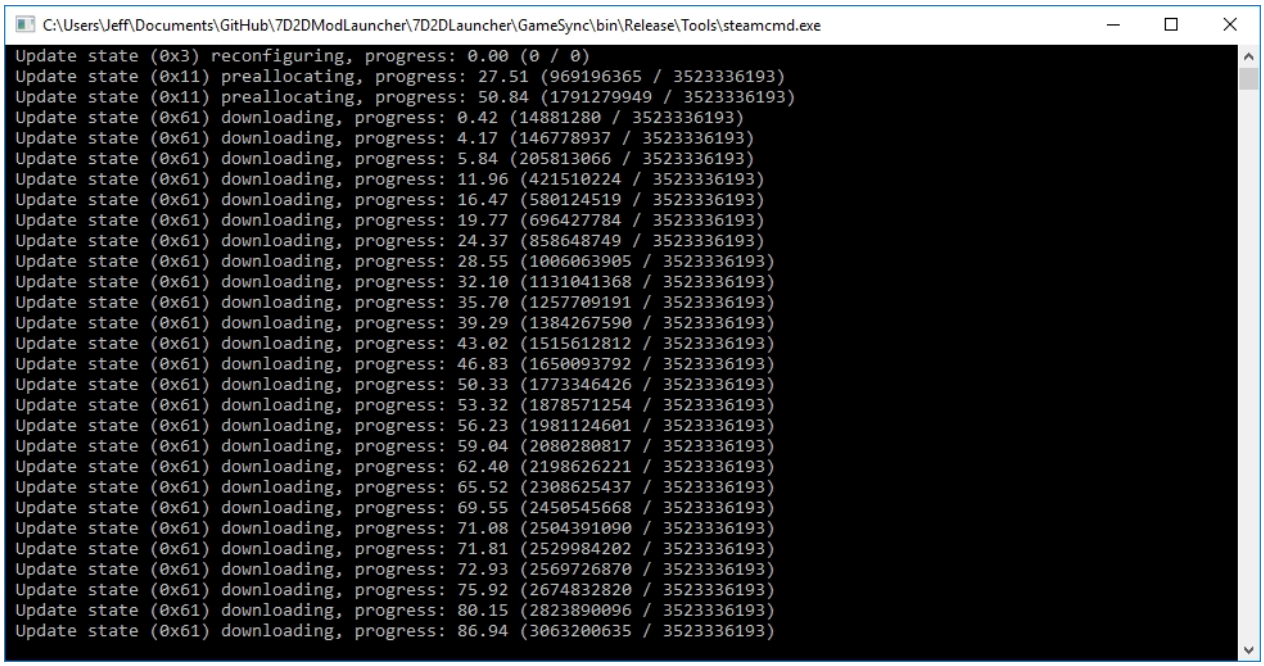
We will be using the 7D2D SDX Mod Helper to download the game for us. We'll want the client, and the dedicated version, so we can add our mods to both.



Click on "Need a new copy?" button to open the Game Downloader



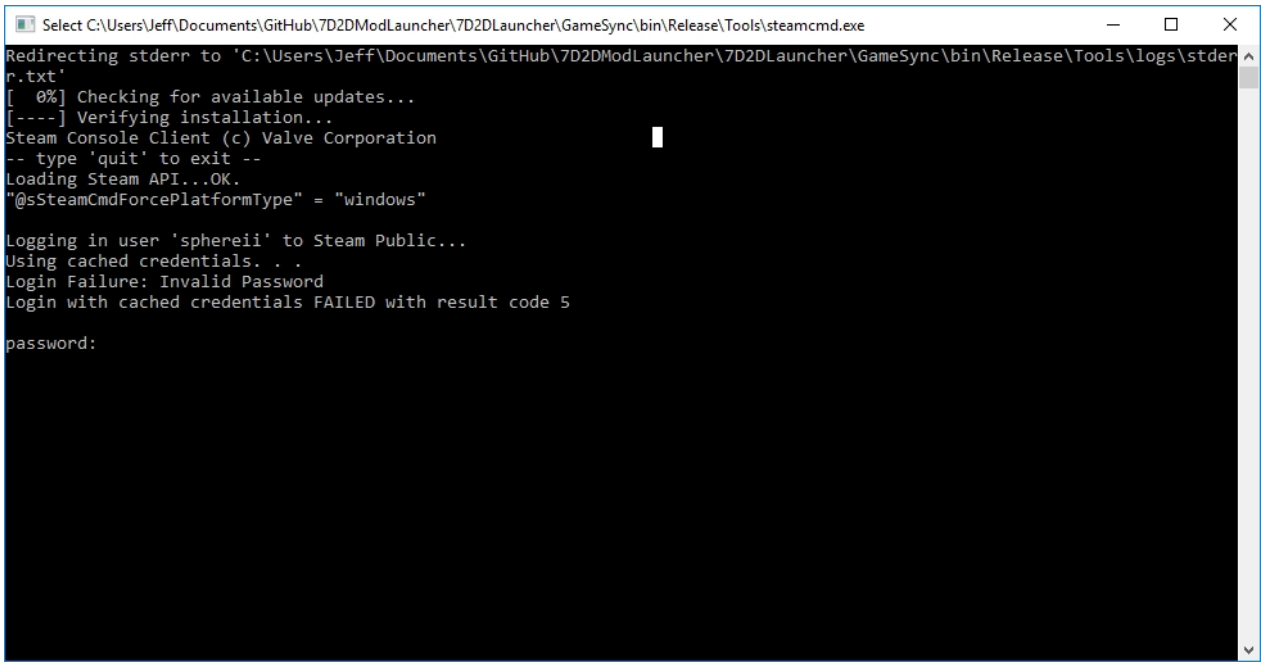
Change the Download Path to point to the new MyMod folder, and enter in your Steam username. After that, select "Windows Client" and "Windows Dedicated", and click on the "Download Checked Items"



Depending on your Internet speed, it may take a little while to download.

When you download the client, it'll ask for your Steam Password, and possibly, your Steam Guard code.

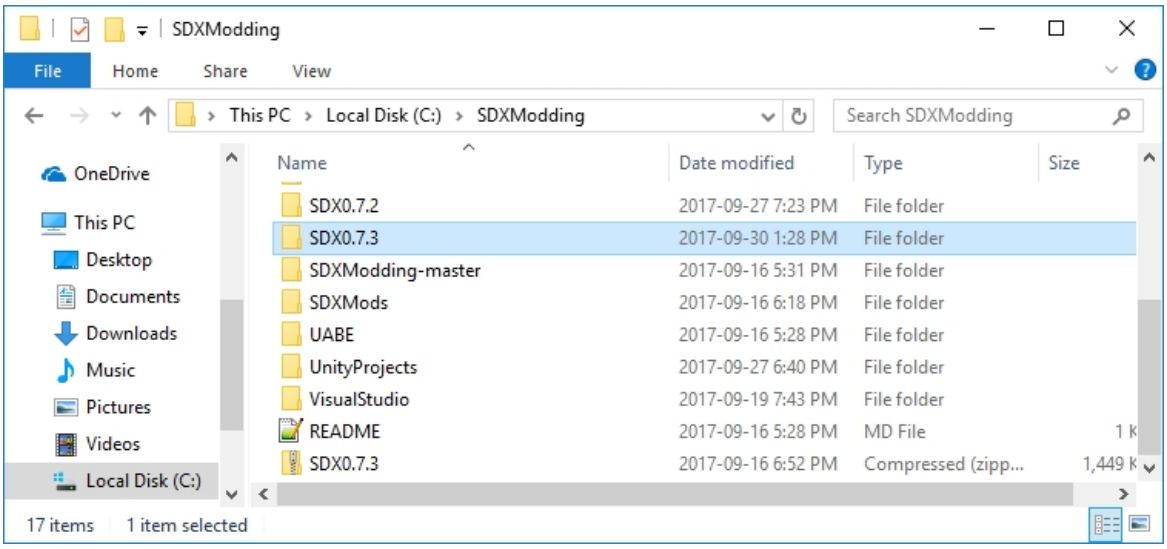
Note: For Windows Dedicated and Linux Dedicated build, the Game Downloader will use the "anonymous" user, and thus no password needed.

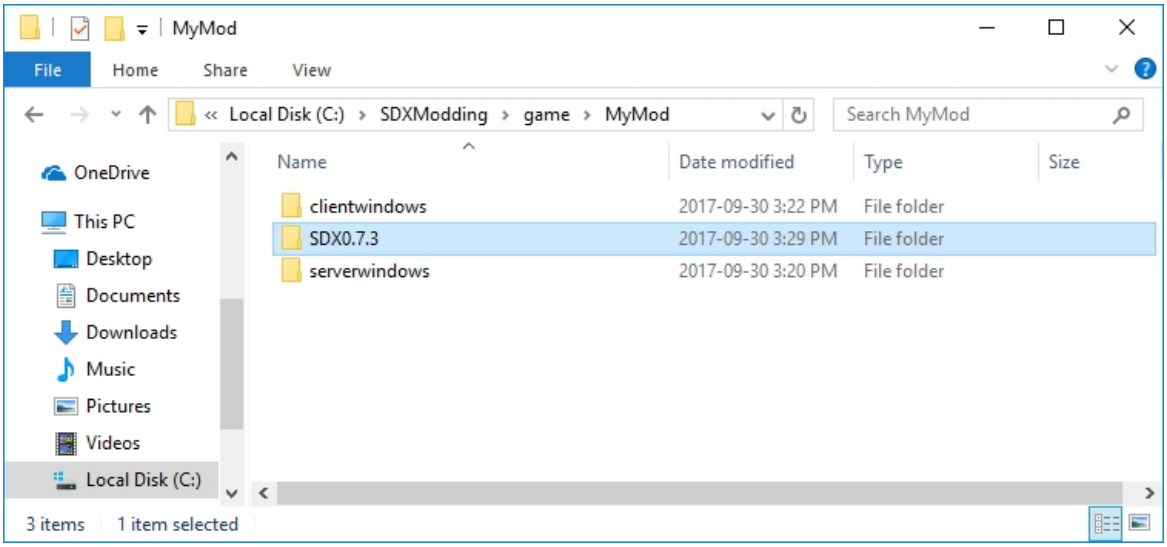


Making a copy of the SDX Launcher

To isolate your SDX work even further, you may make a copy of your SDX folder.

In the SDXModding folder, copy your current SDX0.7.x folder, and paste it into your SDXModding\Game\MyMod\ folder



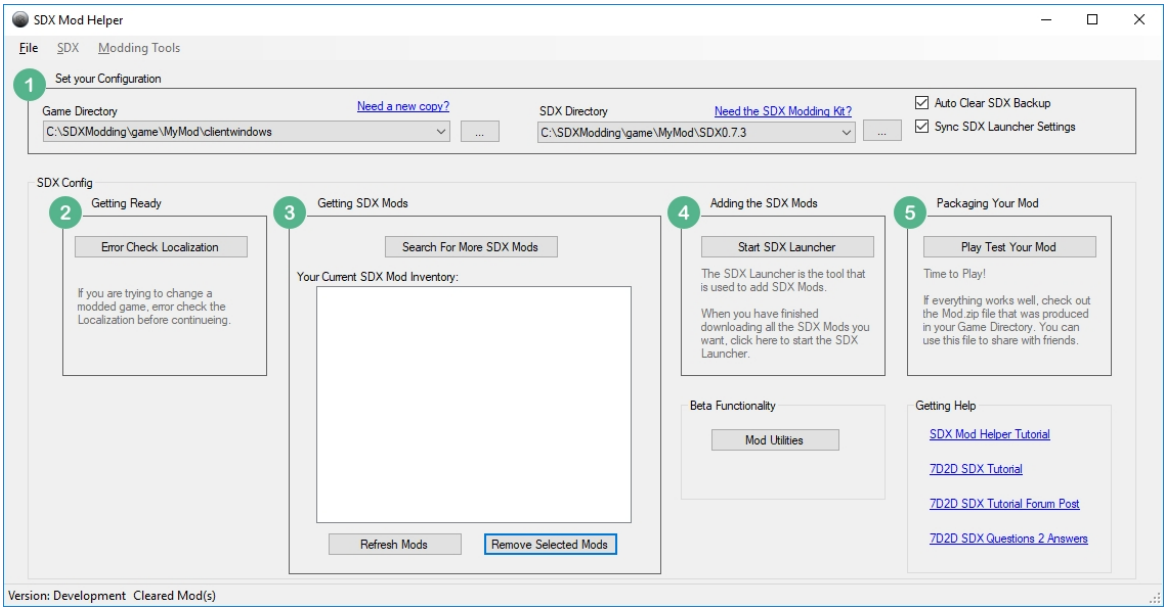


Finding SDX Mods

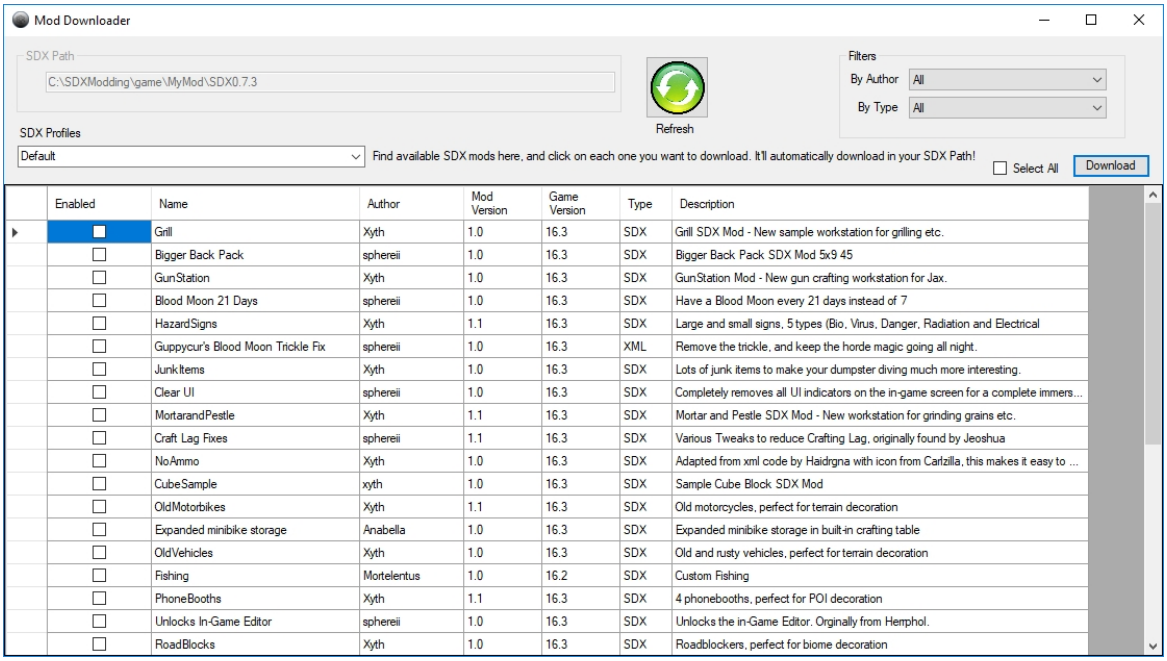
Using the 7D2D SDX Mod Helper, you will want to set up your Game Directory and SDX directories

In the Game Directory, click on the button with three dots, and choose the clientwindows you selected.

Do the same for the SDX Directory until it looks like this:



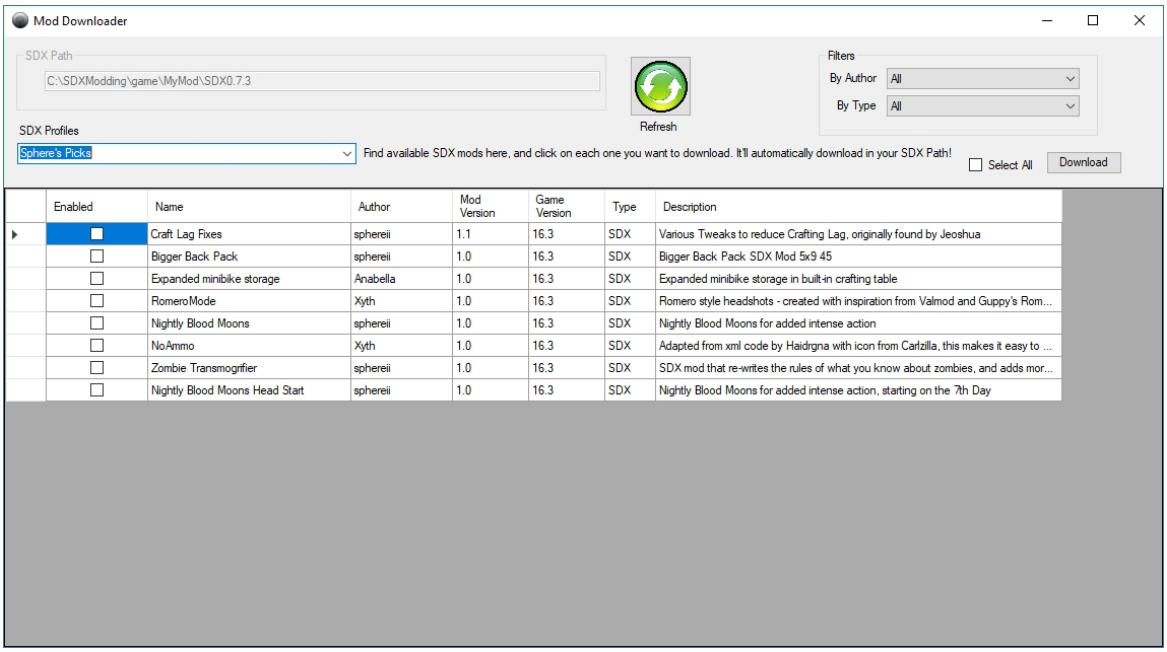
To find SDX Mods, click on the Search For mode SDX Mods



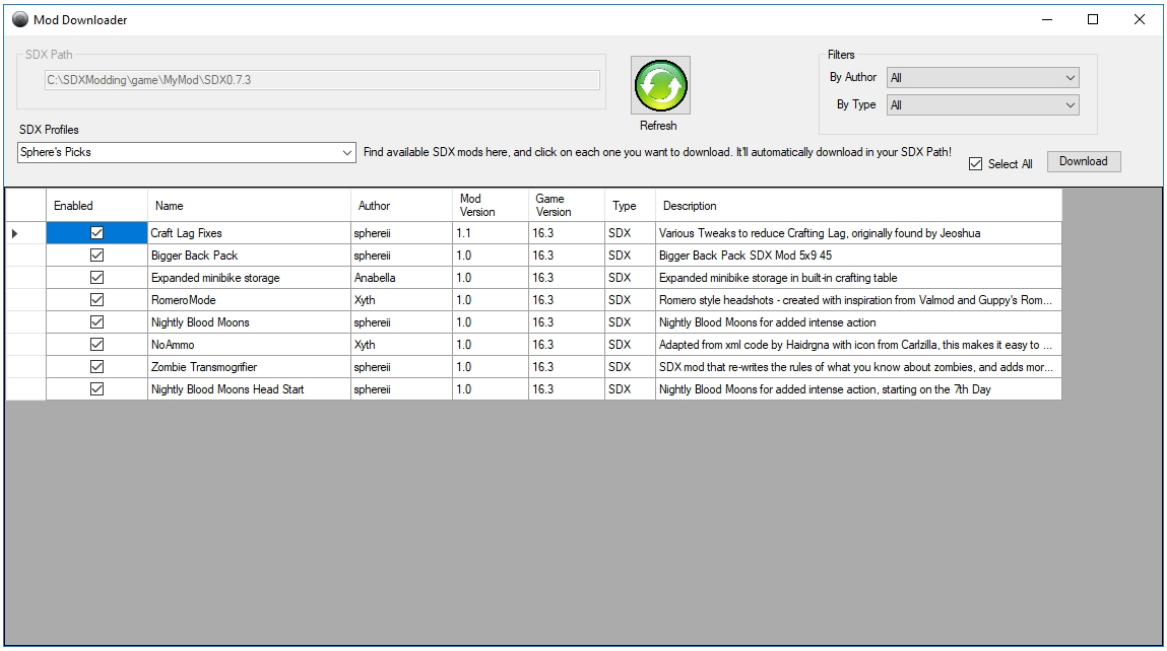
Enabled	Name	Author	Mod Version	Game Version	Type	Description
☐	Grill	Xyth	1.0	16.3	SDX	Grill SDX Mod - New sample workstation for grilling etc.
☐	Bigger Back Pack	sphereii	1.0	16.3	SDX	Bigger Back Pack SDX Mod 5x9 45
☐	GunStation	Xyth	1.0	16.3	SDX	GunStation Mod - New gun crafting workstation for Jax.
☐	Blood Moon 21 Days	sphereii	1.0	16.3	SDX	Have a Blood Moon every 21 days instead of 7
☐	HazardSigns	Xyth	1.1	16.3	SDX	Large and small signs, 5 types (Bio, Virus, Danger, Radiation and Electrical
☐	Guppycur's Blood Moon Trickle Fix	sphereii	1.0	16.3	XML	Remove the trickle, and keep the horde magic going all night.
☐	JunkItems	Xyth	1.0	16.3	SDX	Lots of junk items to make your dumpster diving much more interesting.
☐	Clear UI	sphereii	1.0	16.3	SDX	Completely removes all UI indicators on the in-game screen for a complete immers...
☐	MortarandPestle	Xyth	1.1	16.3	SDX	Mortar and Pestle SDX Mod - New workstation for grinding grains etc.
☐	Craft Lag Fixes	sphereii	1.1	16.3	SDX	Various Tweaks to reduce Crafting Lag, originally found by Jeoshua
☐	NoAmmo	Xyth	1.0	16.3	SDX	Adapted from xml code by Haidrgna with icon from Carlzilla, this makes it easy to ...
☐	CubeSample	xyth	1.0	16.3	SDX	Sample Cube Block SDX Mod
☐	OldMotorbikes	Xyth	1.1	16.3	SDX	Old motorcycles, perfect for terrain decoration
☐	Expanded minibike storage	Anabella	1.0	16.3	SDX	Expanded minibike storage in built-in crafting table
☐	OldVehicles	Xyth	1.0	16.3	SDX	Old and rusty vehicles, perfect for terrain decoration
☐	Fishing	Mortelentus	1.0	16.2	SDX	Custom Fishing
☐	PhoneBooths	Xyth	1.1	16.3	SDX	4 phonebooths, perfect for POI decoration
☐	Unlocks In-Game Editor	sphereii	1.0	16.3	SDX	Unlocks the in-Game Editor. Orginally from Herphol.
☐	RoadBlocks	Xyth	1.0	16.3	SDX	Roadblockers, perfect for biome decoration

The SDX Profiles lets you narrow your view on which available mods, depending on the experience you want to see. You may leave it to Default to view all of them.

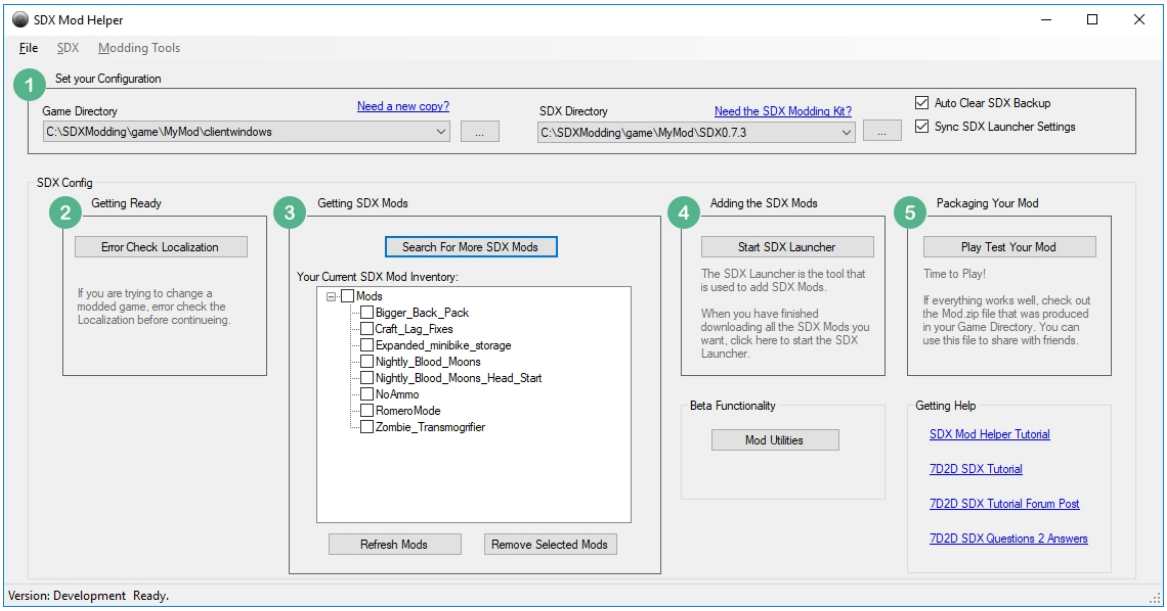
For this example, we'll select Sphere's Picks as the SDX Profile:



You may select individual ones, or click on Select All to check them all.

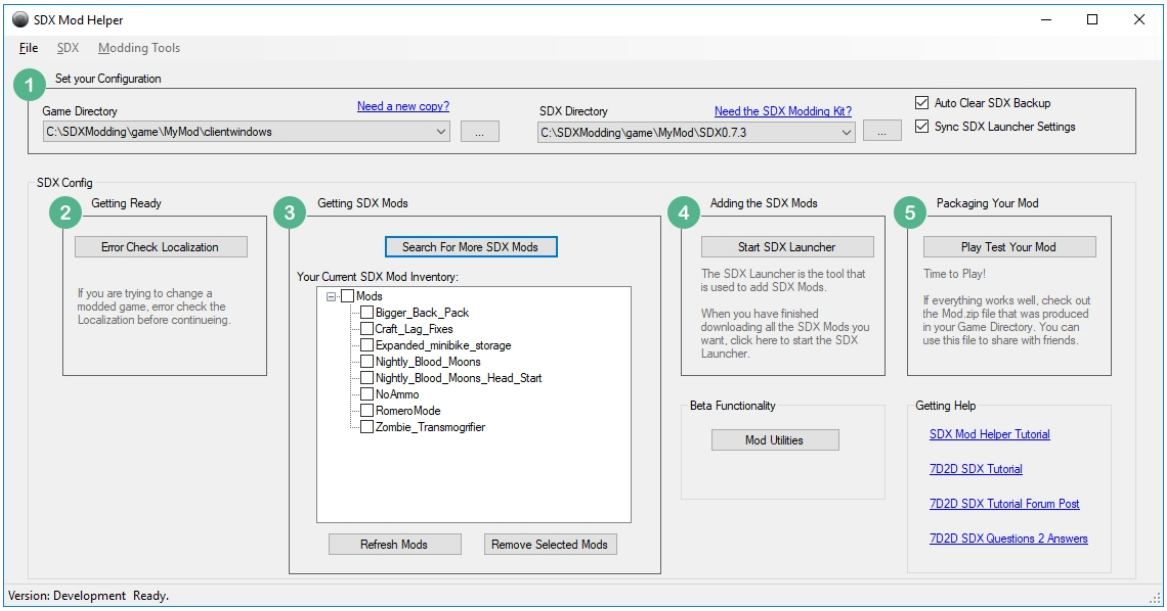


Then click on the Download button. A series of black windows will pop up, and download the mod.

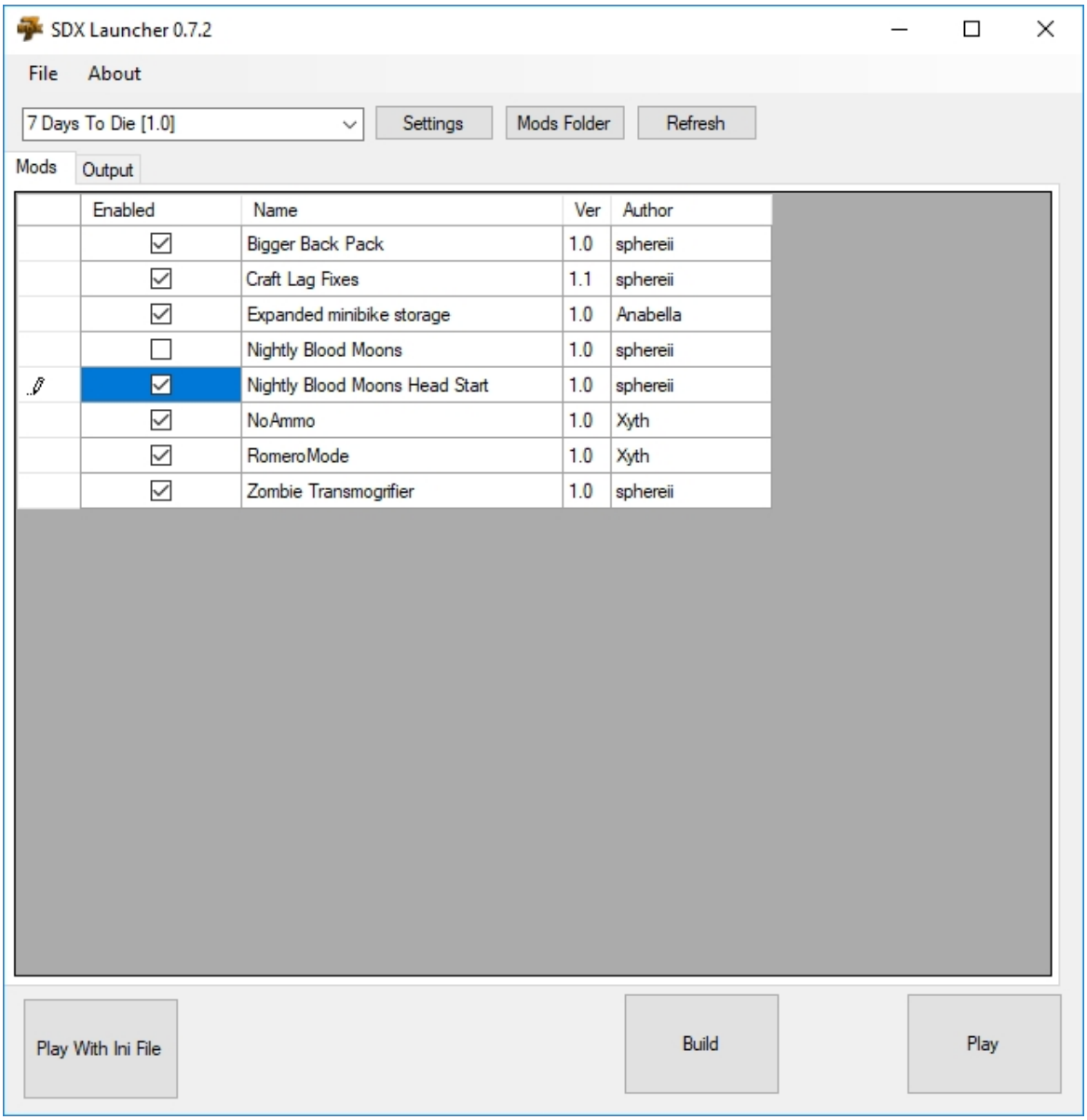


Building the SDX mods

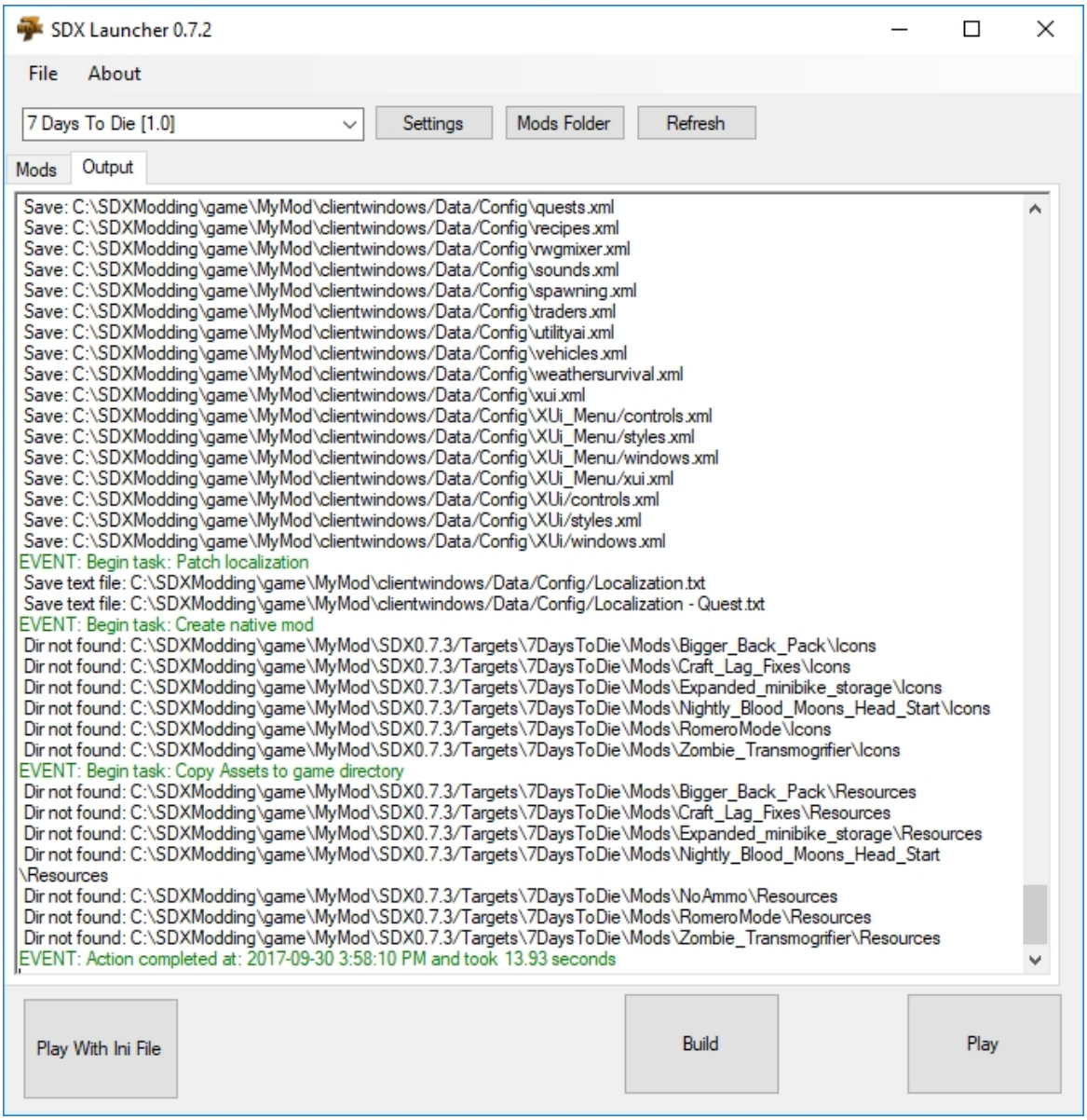
With the game environments setup, and the SDX mods downloaded, it's time to build the mods into your game.



Click on the Start SDX Launcher button.



Click on the Enabled button on all the mods you want to enable. Then click the Build button.



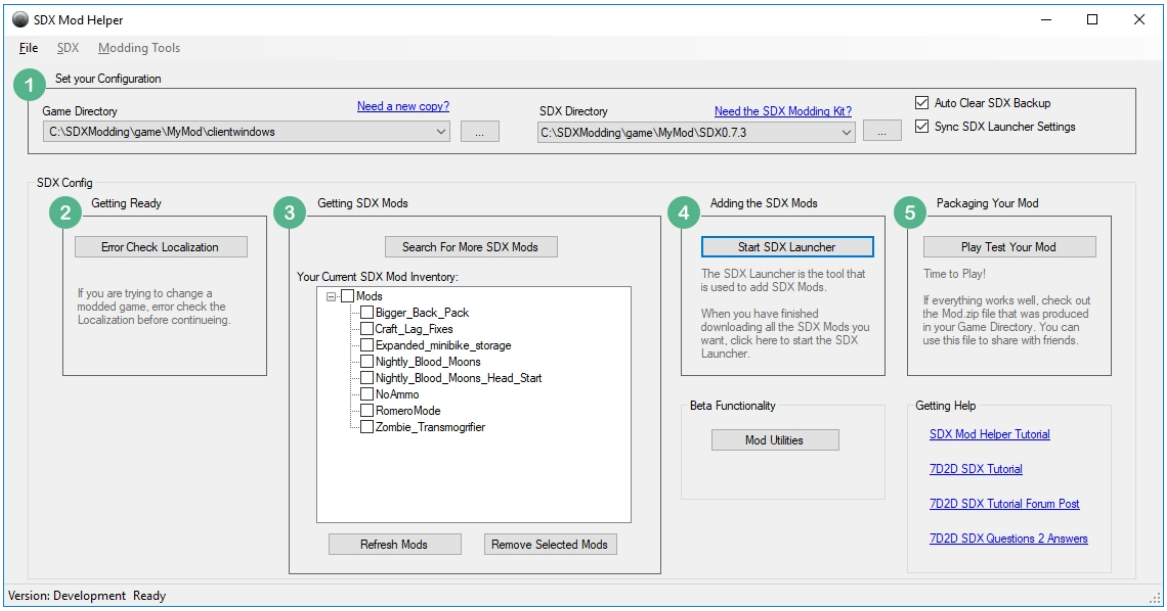
Scroll up, reviewing the logs for any Red errors that may have stopped it from working.

Once you are satisfied that it completed successfully, close the SDX Launcher and return to the SDX Mod Helper.

Note: You may run the game from the Play button in the SDX Launcher. However, the 7D2D SDX Mod Helper does a few extra steps to remove potential warnings from your build log, including removing duplicate icons that are found.

Play Testing the Mod

You are now ready to play test your SDX mod.

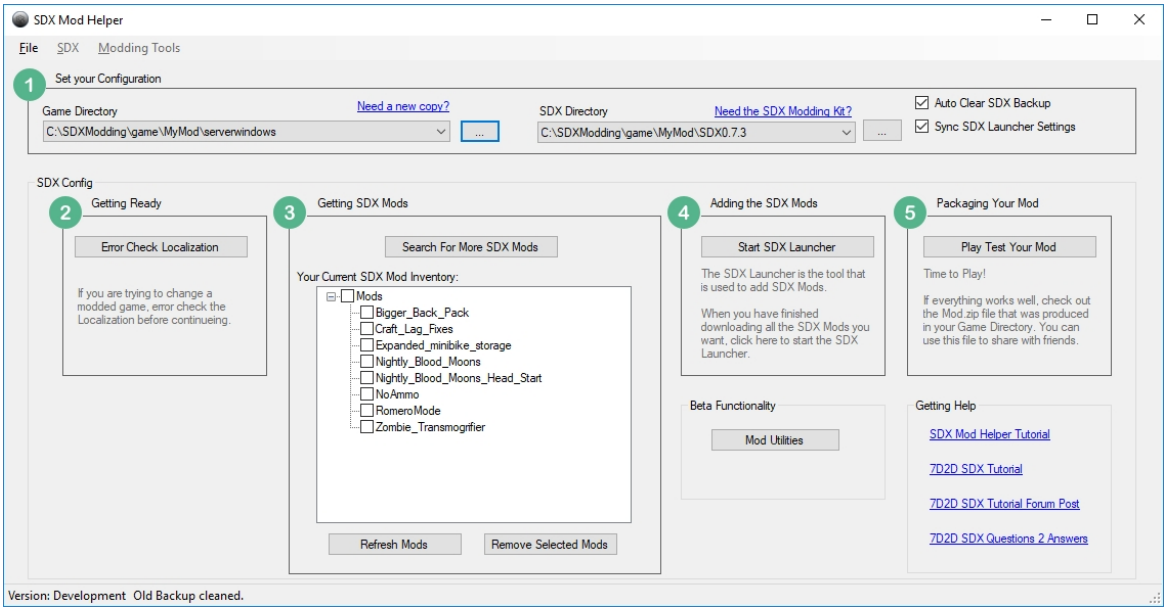


Click on Play Test Your mod to launch the game and play.

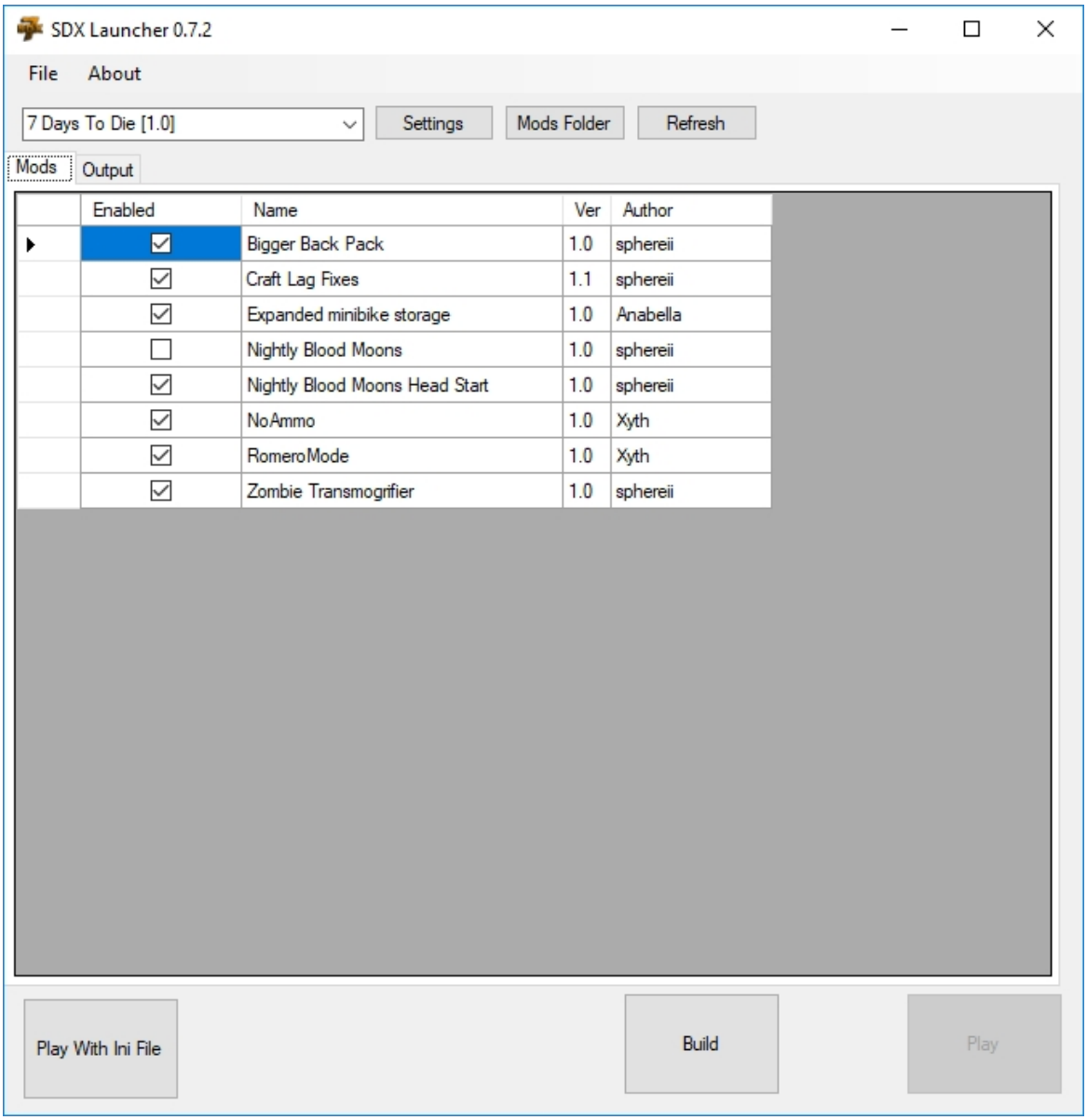
Building the SDX mods for the dedicated server

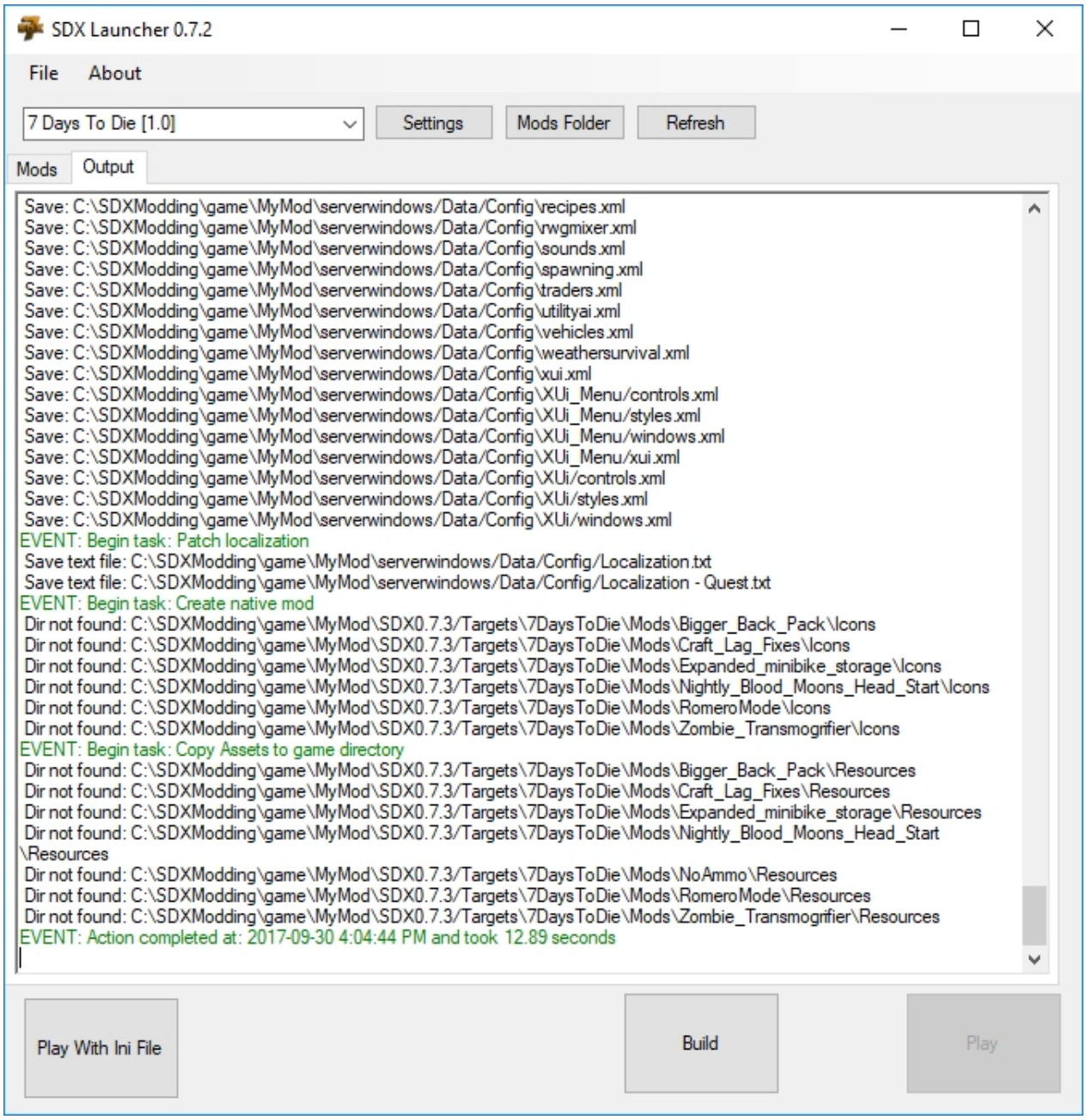
Once you are satisfied with your mix of SDX mods, you may choose to add SDX to the dedicated version.

In the 7D2D SDX Mod Helper, change the Game Directory to point to your Windows Dedicated build.



Click on Start SDX Launcher, and perform a build.


SDX Launcher 0.7.2
File
About
7 Days To Die [1.0]
Settings
Mods Folder
Refresh
Mods
Output
Enabled
Name
Ver
Author
Bigger Back Pack
1.0
sphereii
Craft Lag Fixes
1.1
sphereii
Expanded minibike storage
1.0
Anabella
Nightly Blood Moons
1.0
sphereii
Nightly Blood Moons Head Start
1.0
sphereii
NoAmmo
1.0
Xyth
RomeroMode
1.0
Xyth
Zombie Transmogrifier
1.0
sphereii
Play With Ini File
Build
Play



Note: The play button is disabled because its a dedicated version.

Close the SDX Launcher and return to the SDX Mod Helper.

How to set up the animated custom entity

The AnimationSDX Module allows you to add in your own unique creatures and characters into the world, without relying on in-game animation calls.

This tutorial will guide you on downloading an asset from the Unity store, and converting it into an in-game creature. Some of the steps we will be doing will be similar to adding in a block, but there's some big differences as well.

The following Tutorials are highly recommend to be done before doing this:

Unity 5.3.8p2 Quick Review

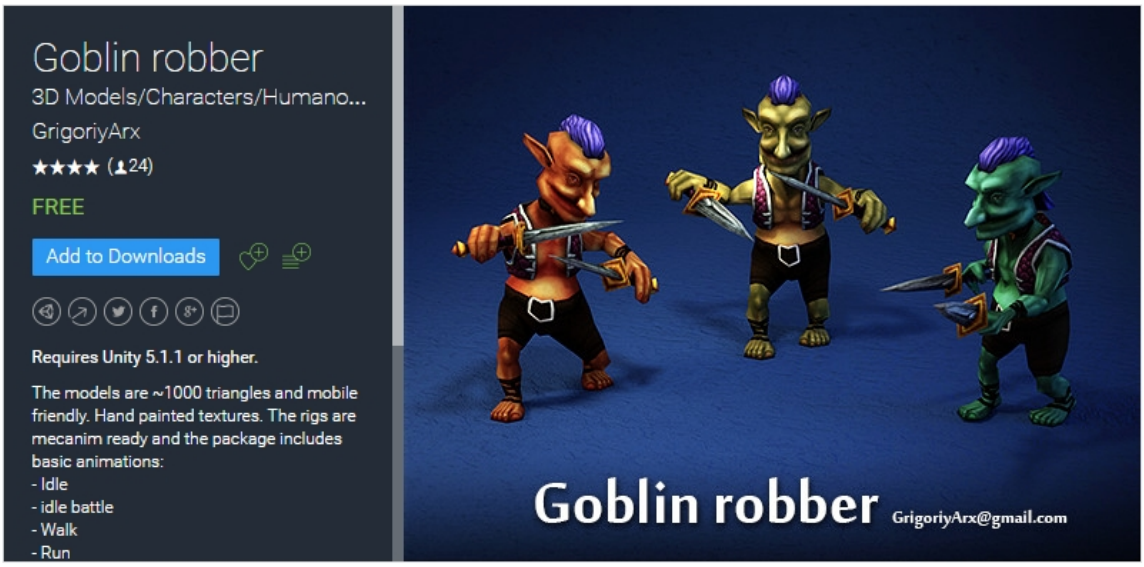
Video Tutorials - Xyth's second tutorial on adding assets to unity

Note: Animations are completely separate from the base game. The animations are part of the unity asset. This means that we cannot hook it into the game's rag doll physics, nor will you be able to add dismemberment to your creations.

But we can still have fun with it :)

There are a lot of assets on the Unity store that will look great in game, while others may be fine for your unique vision of the game. Here's a few things to look for.

Typically, any 3D Model that has Legacy animations will work with this AnimationSDX Module. For a full easy experience, you'll want to look for assets with added animations, such as the Goblin Robber by GrigoriArx, found at https://www.assetstore.unity3d.com/en/#!/content/66959



In the description, it requires Unity 5.1.1 (We use 5.3.x), and contains some basic animations that we need.

Depending on your end goals, you'll want to have animations that can be used for the following tasks:

Attack
Die
Jump
Run
Walk
Idle
Pain

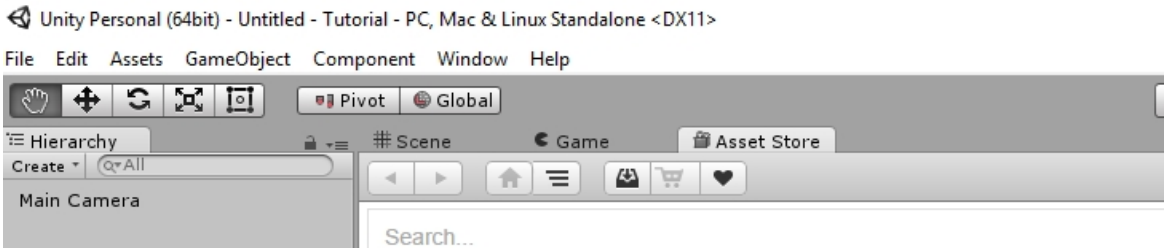
With the SDX Animation module, you can have 2 different attacks, in addition to a special attack that gets randomly called. Idle can also have two separate animations that get called. These extras are optional, as is most of the animations types listed above. However, to make them feel part of the world, you'll want those basics covered.

If you are comfortable with Unity and animations, you can add in your own custom animations. For this tutorial, however, we are going to be looking for assets that already come with animations.

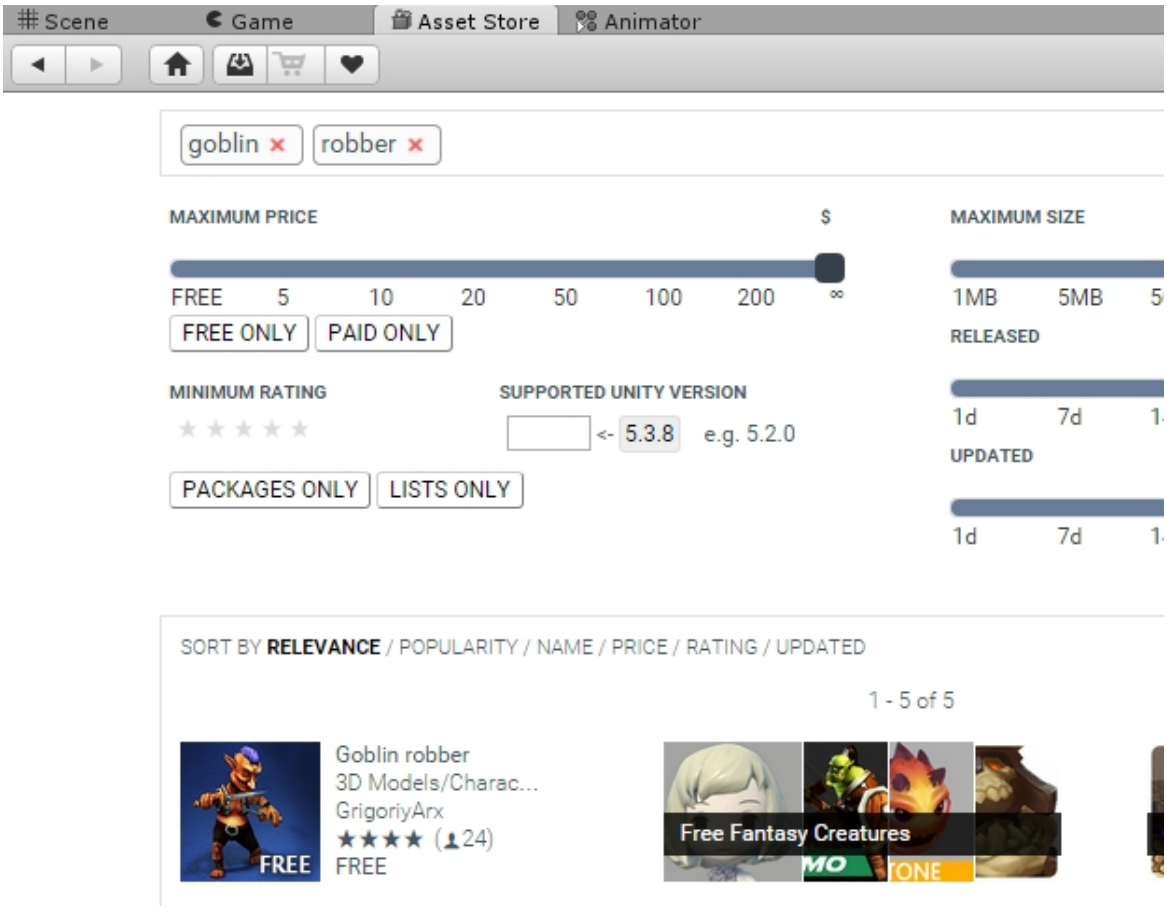
Downloading and Importing the Goblin robber

It's a simple free object that has included animations.

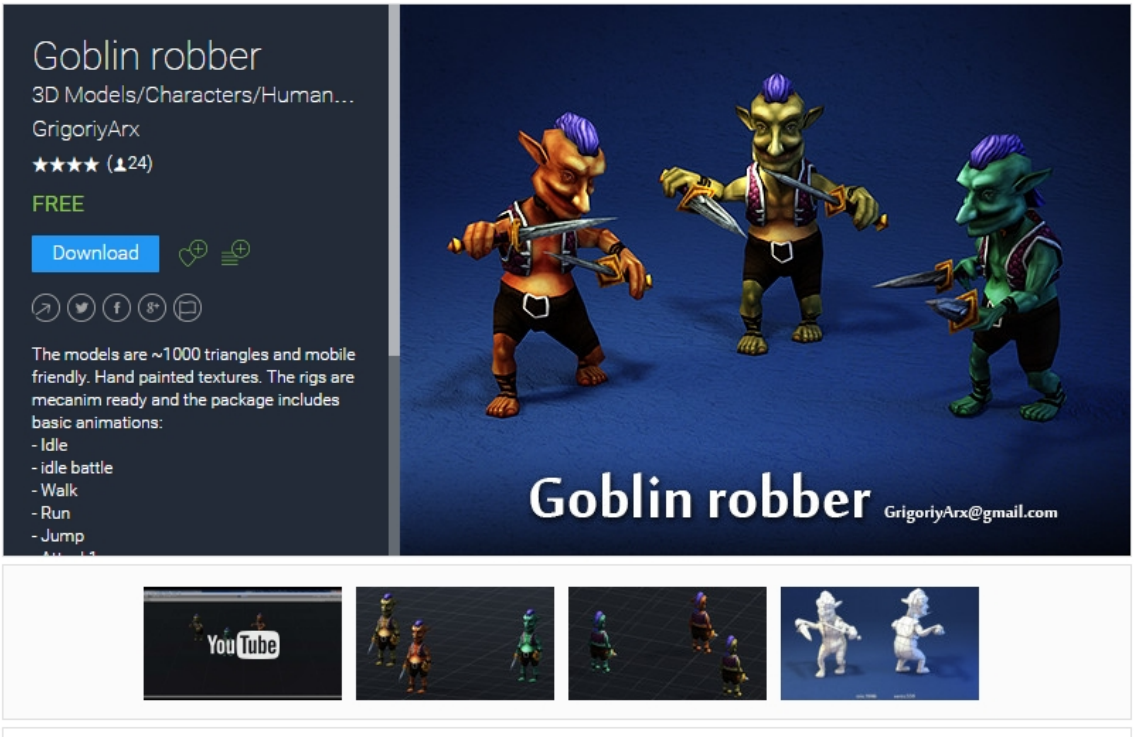
In Unity, click on the Asset Store Tab



And searching for "Red Samuria". It should be the first hit in the search results

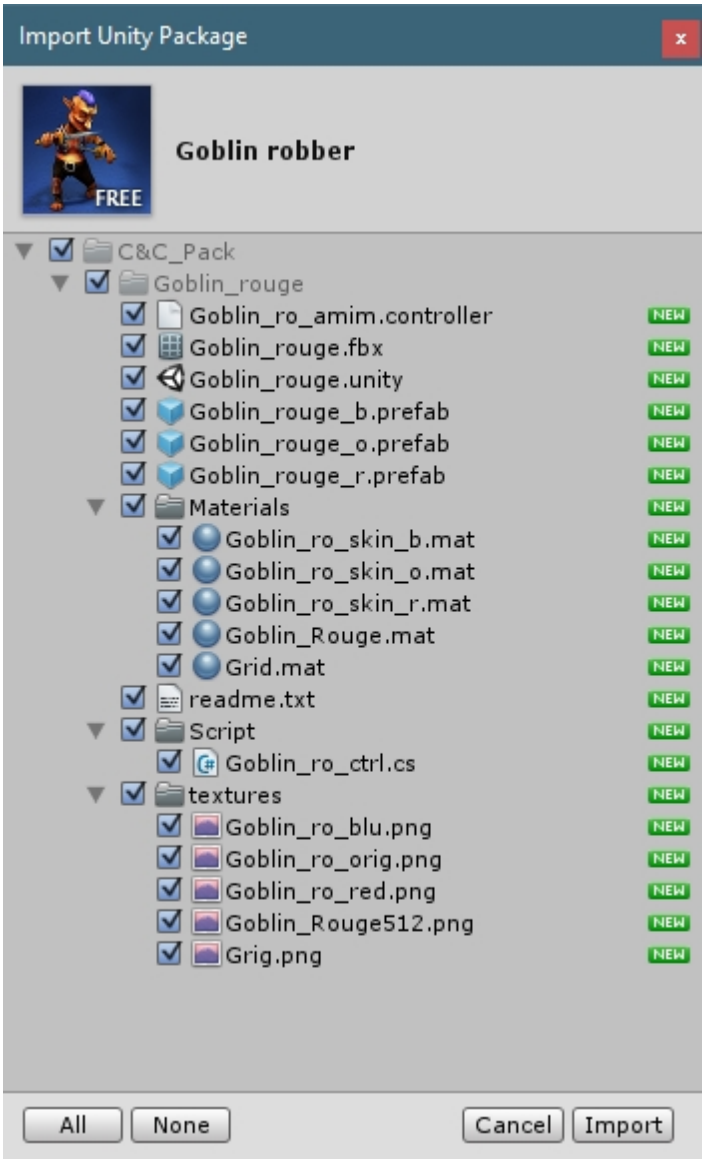


Click on the Goblin robber (free),and click on the Download button:

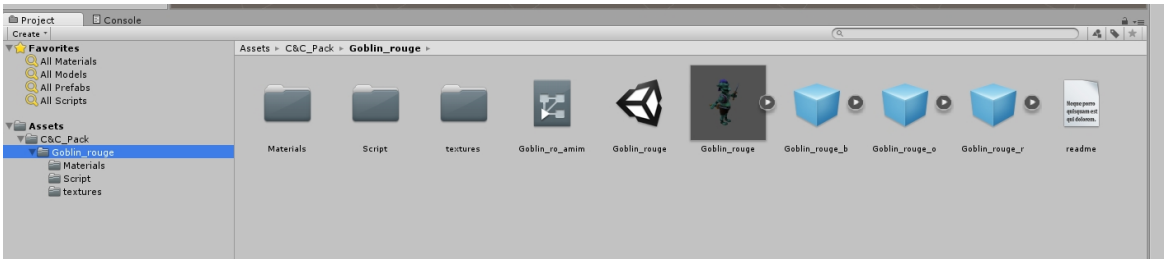


Once downloaded, the Download link will change to "Import". Click on Import to add theRed Samurai to your new Unity Project.

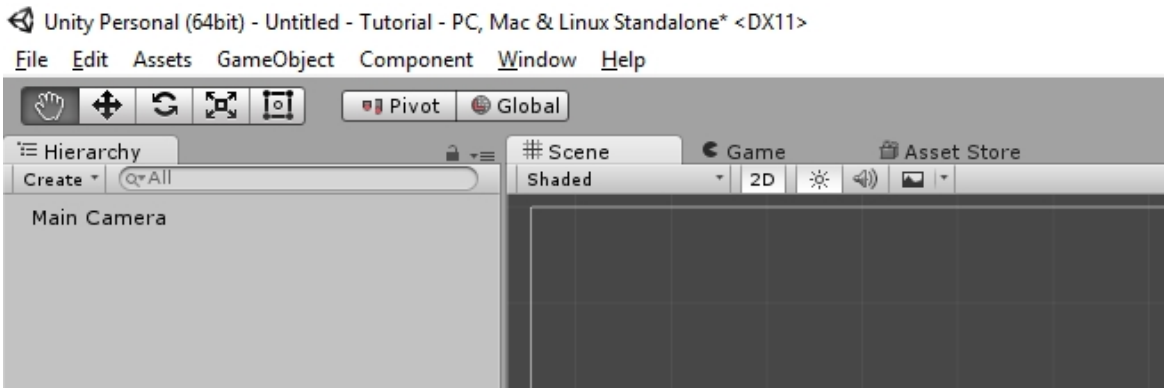
The Import dialog box will pop up, showing everything included. Click on Import to bring them all into your project.



Once the Import is done, you'll see the new "C&C_Pack" folder, in your Project tab, under Assets.



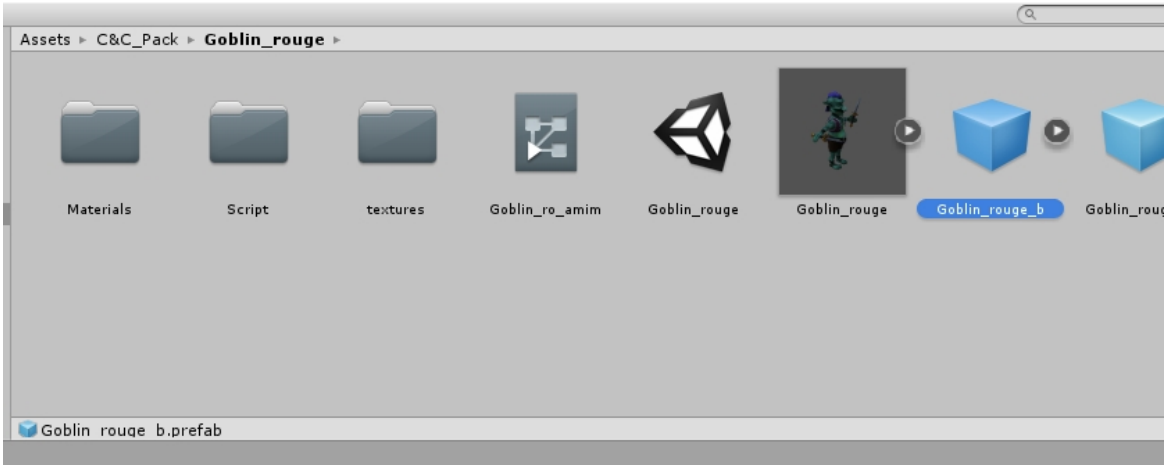
Click on the Scene tab to get back to your starting area.



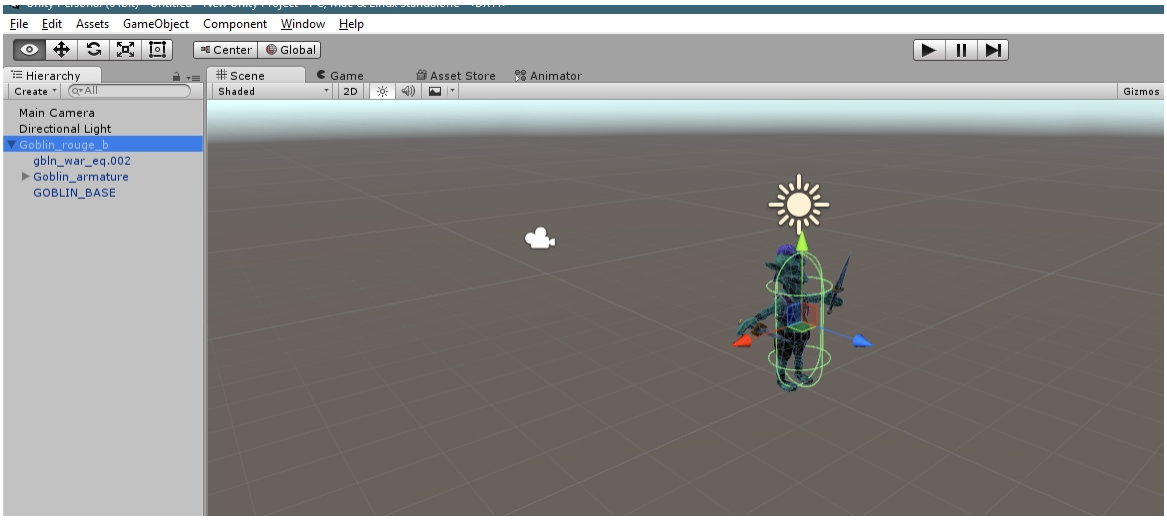
Adding the Entity Prefab

An avatar or creature that you want to add is similar to what you'd want to do for a block.

Under the C&C_Pack, in the Goblin_rouge folder, in your Project window, look for the a "prefab" file in the folder structure. In the Goblin Robber, we'll use Goblin_rouge_b:



Click and drag the goblin_rouge_b up into your Hierarchy window. Unlike blocks, the Transform Position doesn't matter. Even after you reset it to 0,0,0, your model may not be centered. Double click on the Goblin_rouge_b in the Hierarchy window to center your object to your view. Note: You do not need an Empty Game Object to add it. It can go right in to the root.

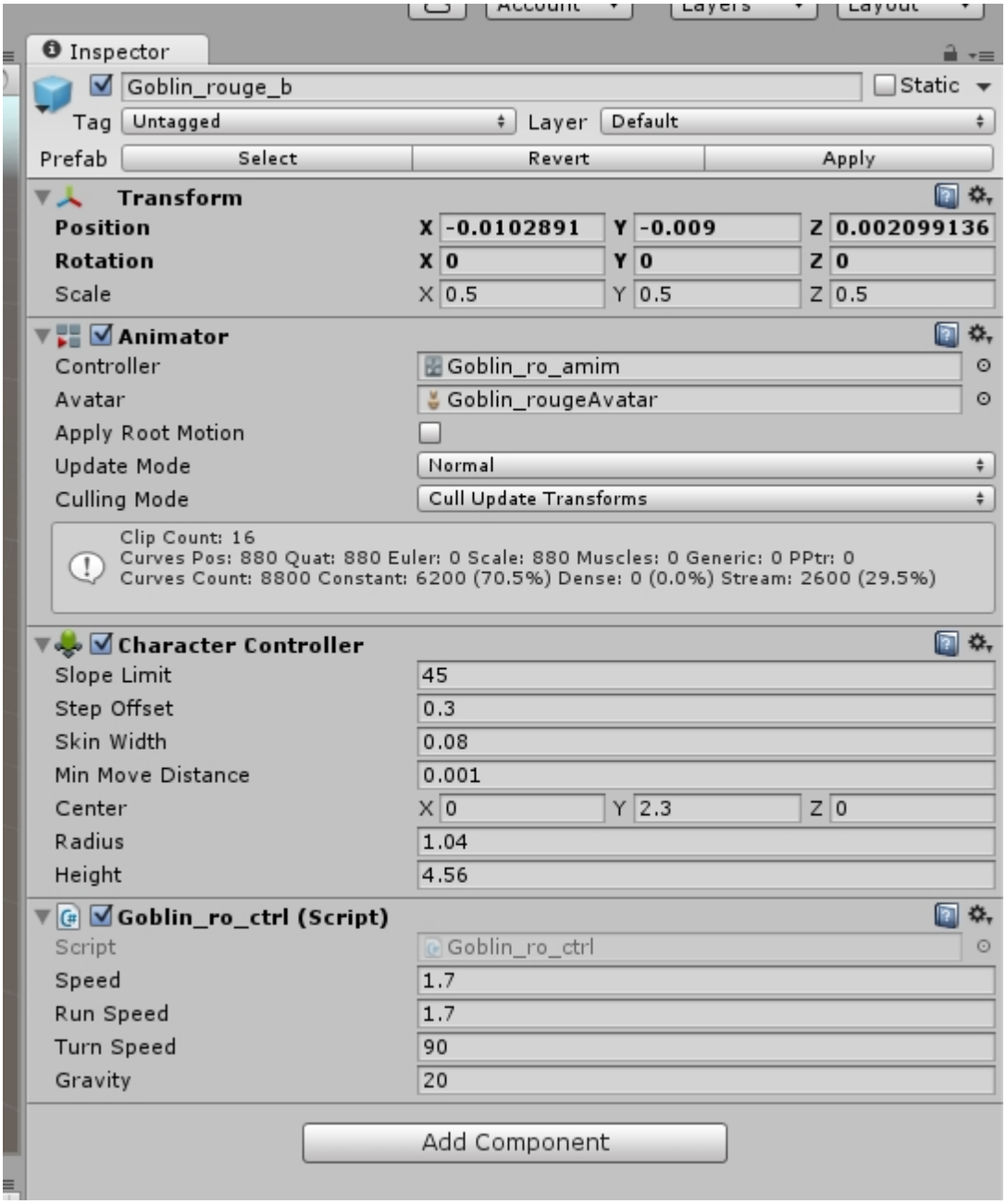


Checking out the new Components

In previous tutorials, we had to deal with the Transform Position (so the block sits right in the world), and the box collider.

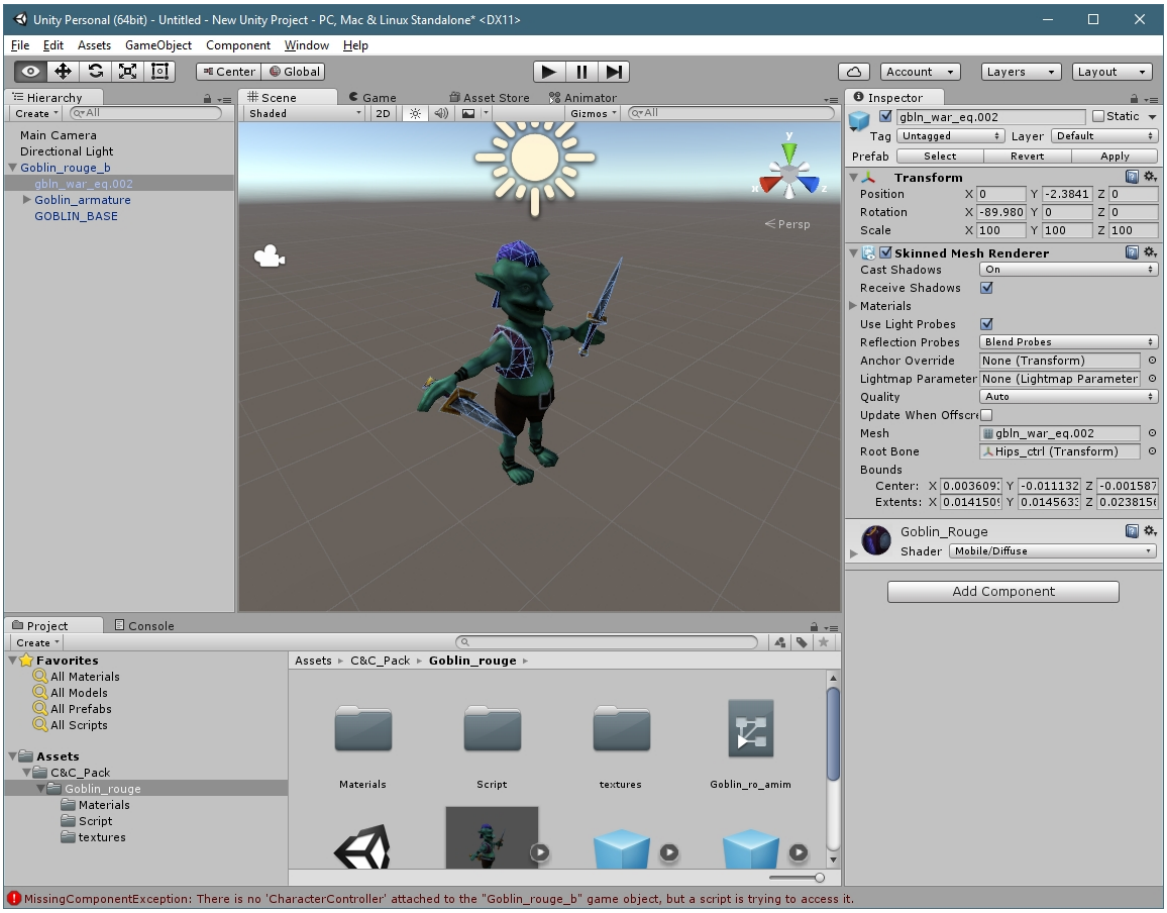
An animated character has more Components in order for everything to work well. We'll review them here. Each model you work with will have different names, and ordering to them as they appear in the hierarchy.

With the goblin_rouge_b prefab selected, we'll look at the Inspector



We have our Transform, as always, but we also have some new components called Animator, Character Controller, and a Script. We don't need a Character Controller, or a Script component to add the game, and they may even give us errors. Click on the Gear Cog by each component, and Remove Component.

Under the Hierarchy, click on the second part of the prefab, the "gbln_war_eq.002", and look at the Inspector.



On this object, you can see in the Scene window that its swords, jacket, and hair are all highlighted. This tells us that that this layer just involves those items. We have our transform, and our Skinned Mesh Renderer. The Mesh Rendered is what applies the mesh for the highlighted objects.

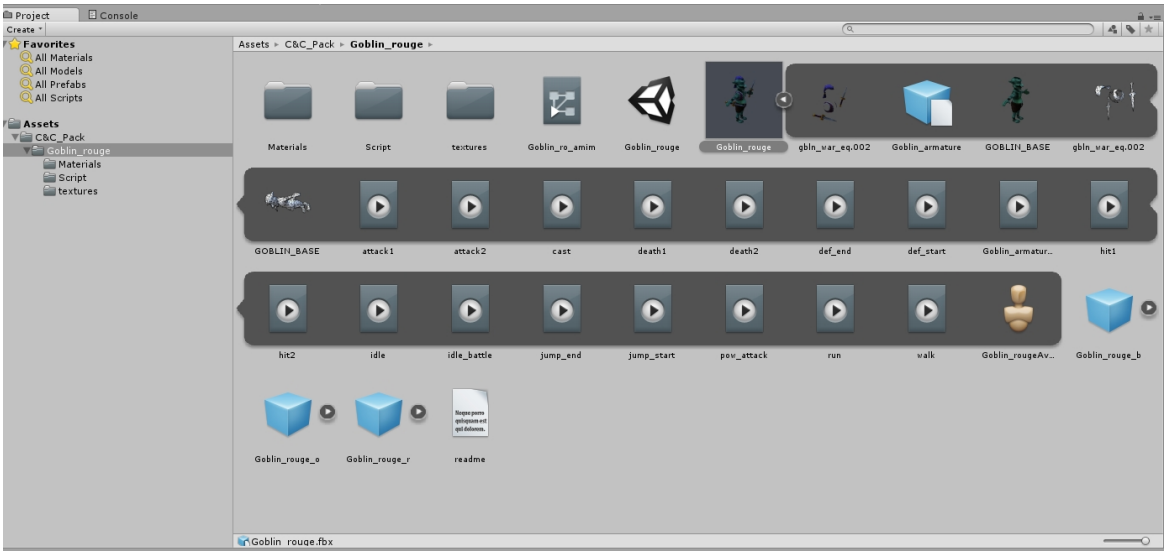
In the Hierarchy, click on the Goblin_armature. If you expand the selection, you'll see there's just 'empty' game objects. These are the Goblin's joints and body parts. We typically don't need to make any changes to them.

Back in the Hierarchy, click on the GOBLIN_BASE. You'll notice it has a Skinned Mesh Rendered, and it covers up nearly every aspect of the Goblin. It doesn't includle his hair, or his weapons, but that's fine. Since this game object is the one that covers the object the most, it's where we'll want to place our Mesh Collider later.

Changing Animation Type to Legacy

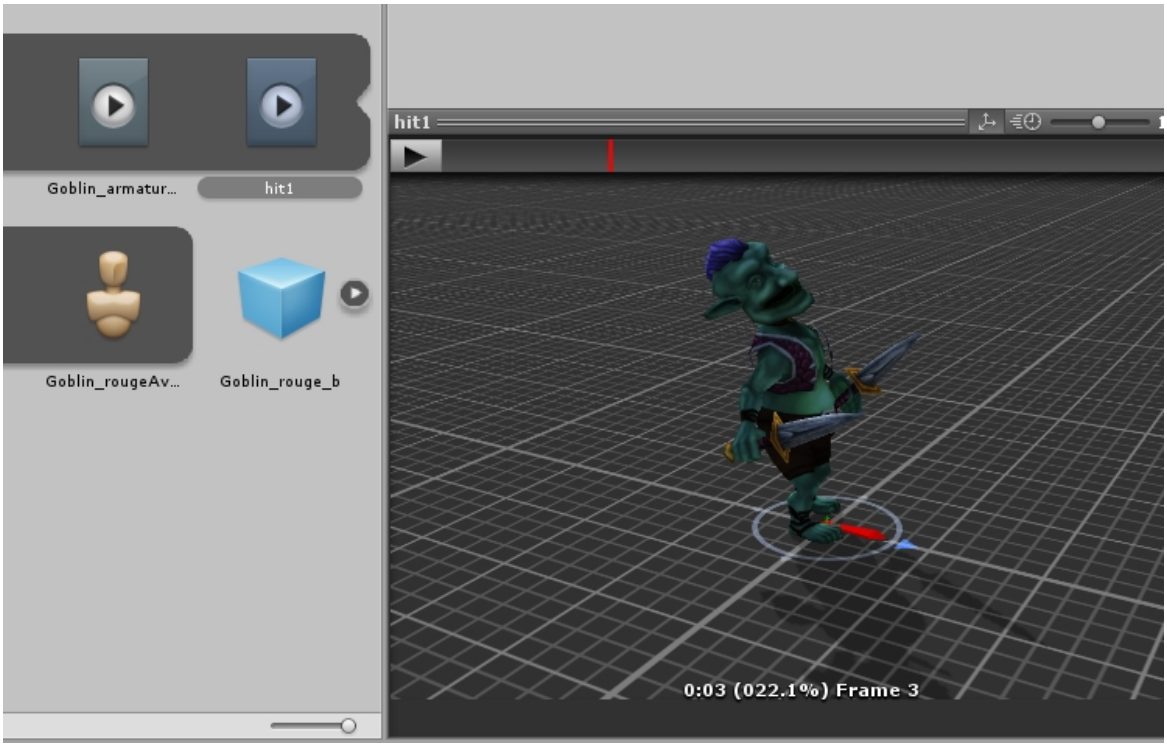
Some of the animations available are for different versions of unity, or won't work well in the game.

For these, we want to turn them to Legacy. In the Project window, click on the C&C_Pack, and then Goblin_rouge asset. Animation assets are typically stored in a .fbx file. when you find it, you can click on the little Play sign to expand it and see the animations

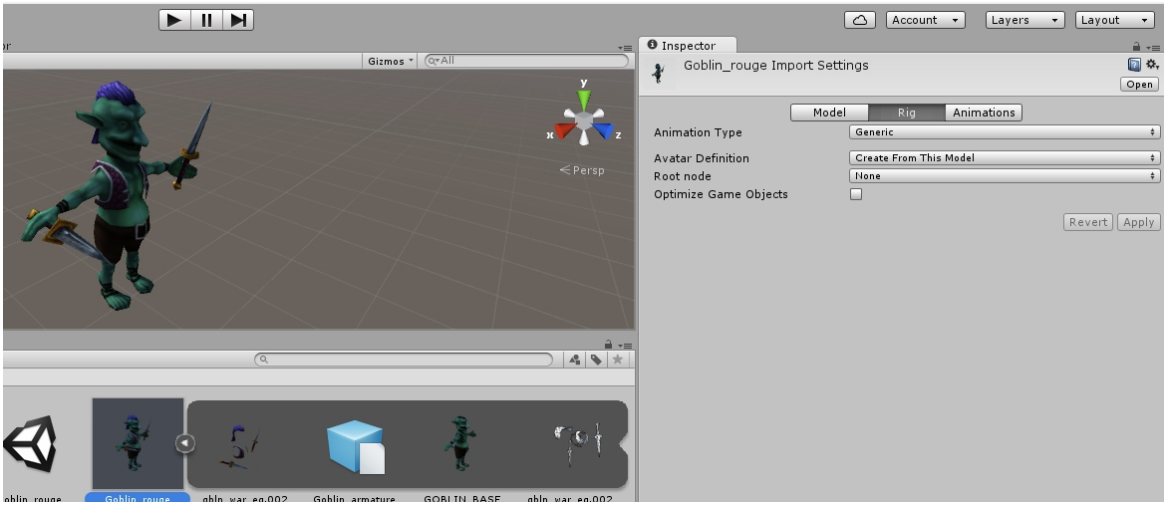


The Animations are the objects that have the Play button on them. When you click and select one, you'll have a window that pops up that let's your preview what the animation looks like.

.



Click on the main FBX object, which is Goblin_rouge.fbx, and look in the Inspector.

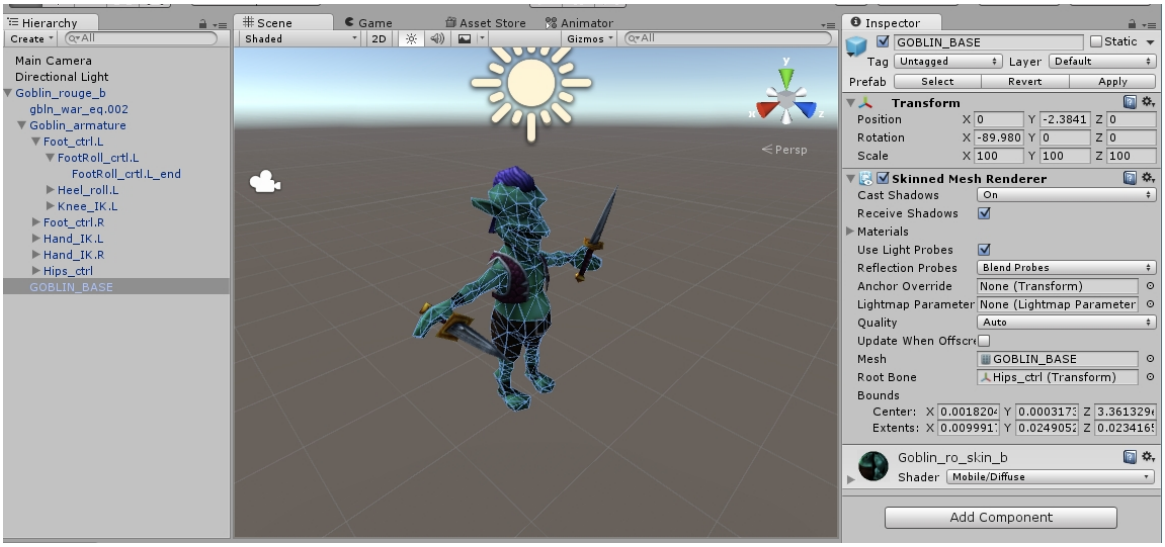


For the Animation Type, we want to select the drop down to "Legacy". Click on Apply, which may take a few minutes to process, as its converting all the animations.

Adding a Mesh Collider

We'll need to add in a collider in game, so that we can interact, or kill it.

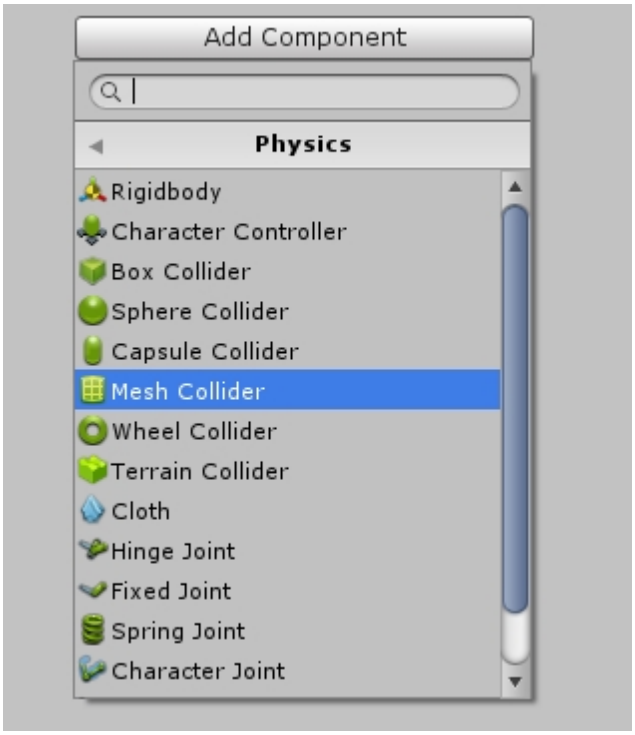
In the Hierarchy window, click on the GOBLIN_BASE. This is the same object that has the Skinned Mesh Renderer, and the game object that covers most of the asset.



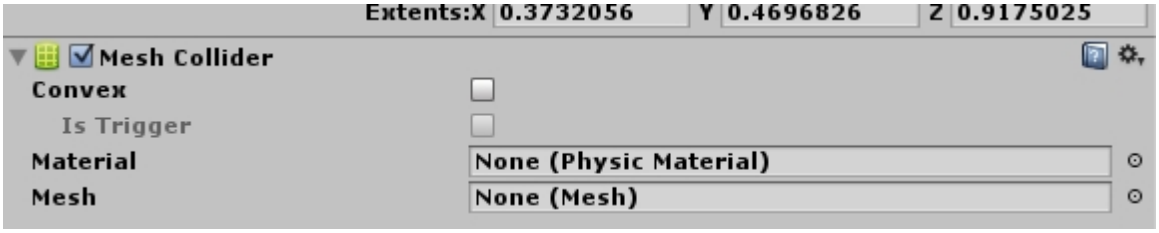
Whereas for the blocks, we've always chosen a Box Collider, we'll try to add a Mesh Collider to this animation. The Mesh colliders are more resource intensive, as they form around the shape of the object, but make more sense when adding an entity you want to interact with.

Note: Some animated avatars don't have a clear mesh to use, so you may have to resort to using a block collider.

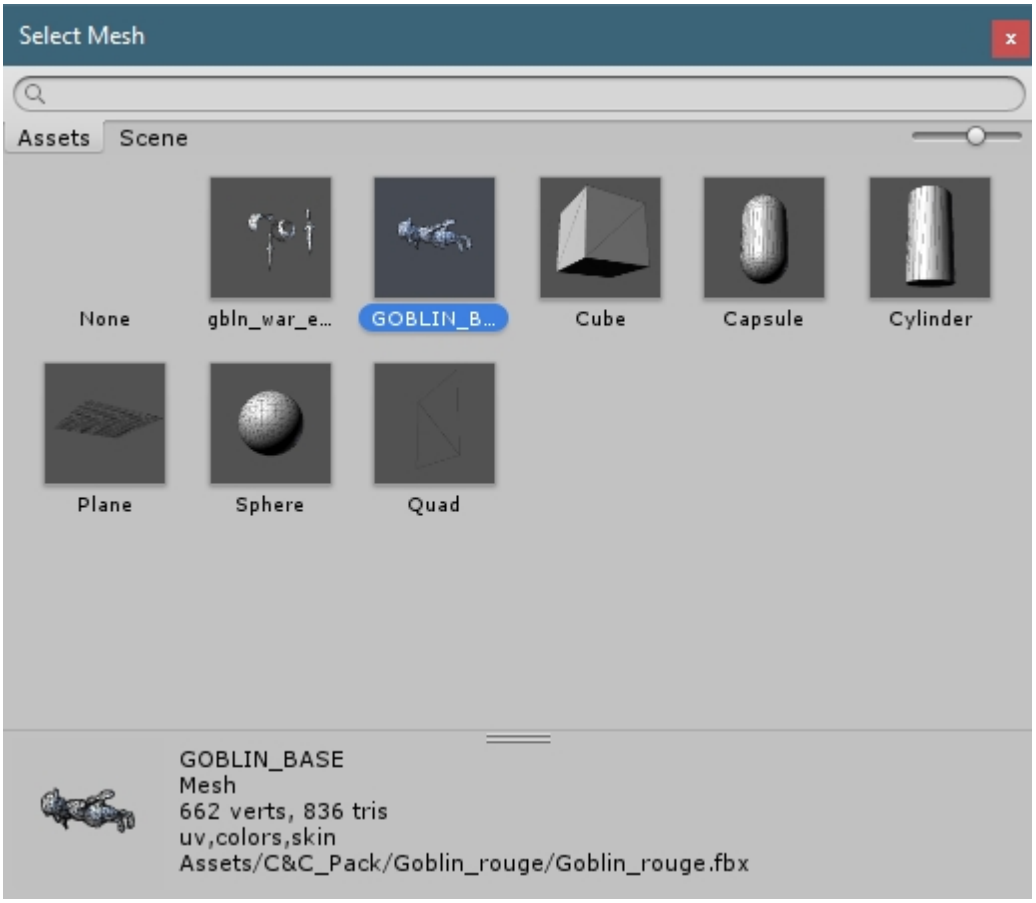
In the Inspector window, click on "Add Component" -> "Physics" -> "Mesh Collider"



The Mesh Collider Component is then added.



At the end of the Mesh text box, click on the circle to select which Mesh you want to use. The mesh is basically the skin of the avatar.



Select the "GOBLIN_BASE" Mesh. Similar to when we added the Animations, you want to be sure to select the right mesh for the character.

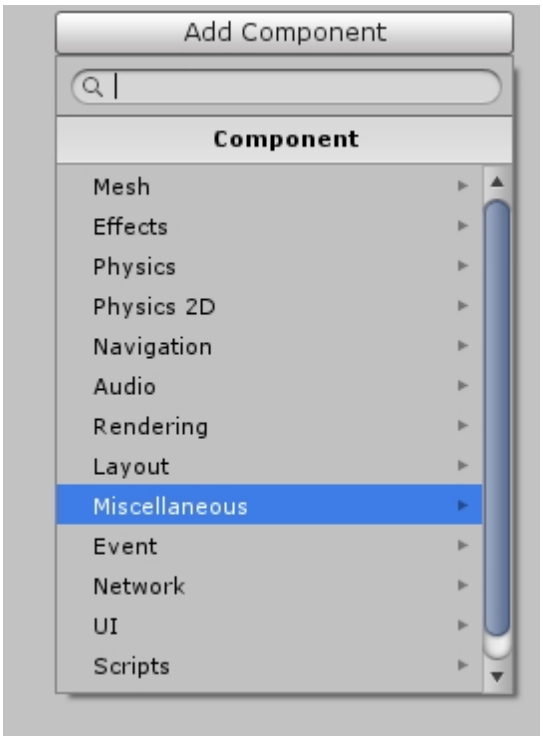
If your Mesh is laying down, while your avatar is standing up, look at the Transform. Adjust the rotation of the object until the Mesh lines up to the character.

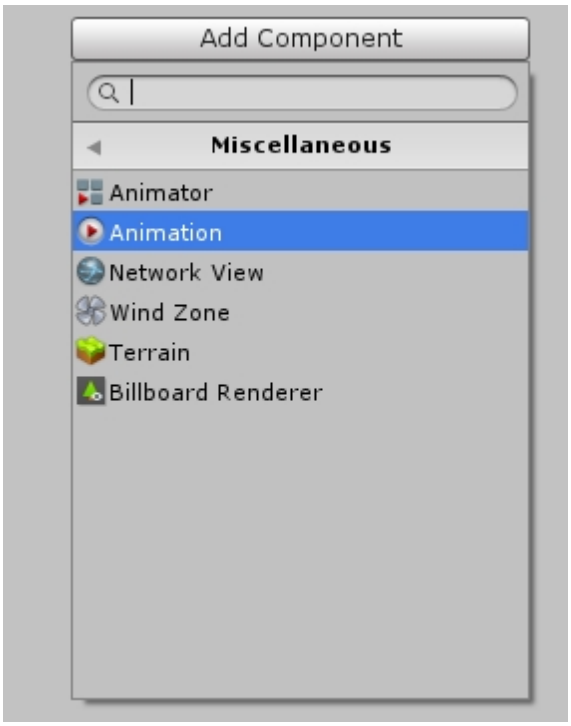
Some Meshes don't appear to fit the object precisely, but try to find the best one.

Adding the Animations

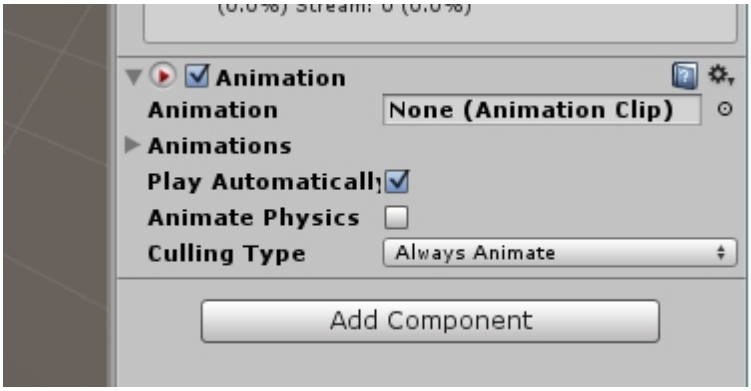
Some models have Animations already applied to the prefab, while others do not. In our case, our Goblin does not animations applied to him yet. Animations need to be applied at the top level of your Game Object. In our case, that's Goblin_rouge_b.

Click on the Add Component button, and select Miscellaneous, then Animation.



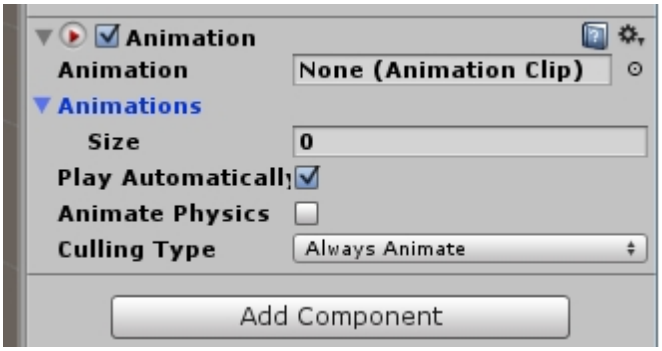


This adds a new component, called Animation

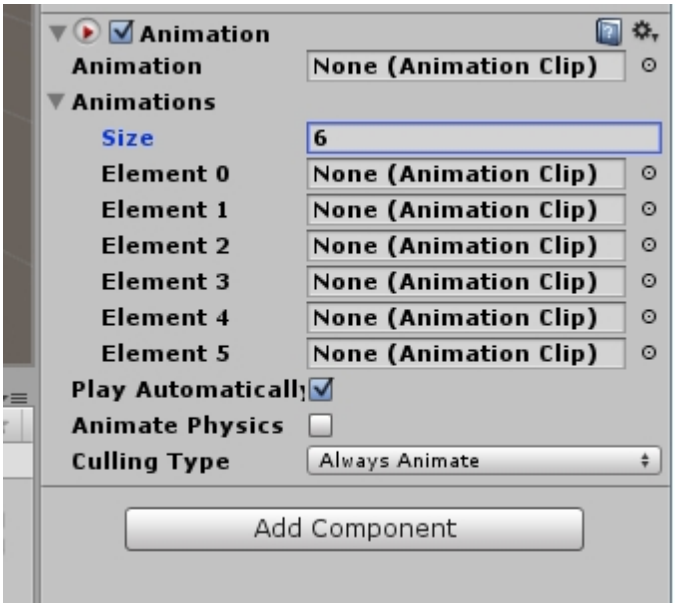


Animation: This lists the default Animation you are going to apply.
Animations: This subsection is where you can apply the various animations

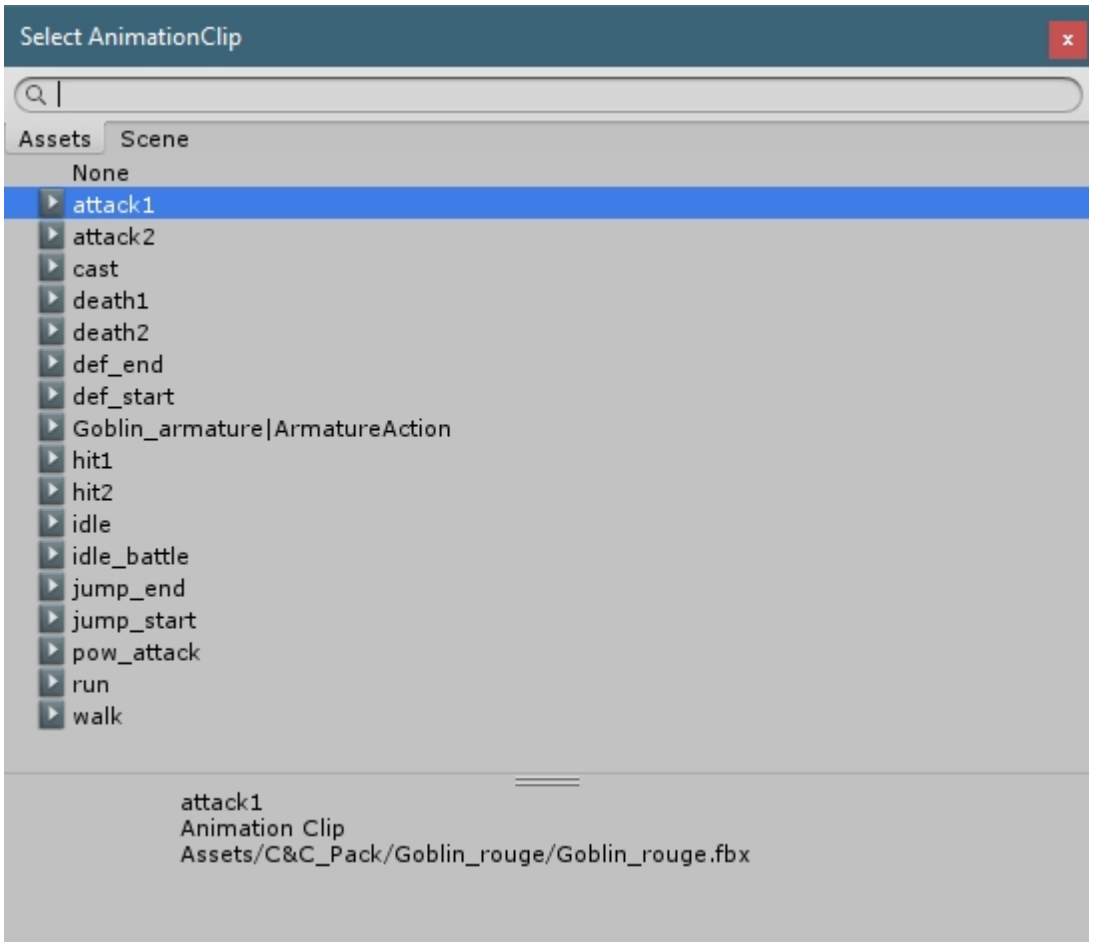
To start off, click the down arrow to open up the Animations subsection.



The Size determines how many animations you want to add. This can be changed afterwards, you initially, you can set it to 6, and hit Enter.



Each Element can hold a separate animation. You can click on the circle at the end of the text box to open up, and select your animations.



You'll typically want to add at least the following type of animations: Idle, attacking, walking, running, jumping, dying and getting hit. This allows the entity to behave as you'd expect in game. If you do not have a particular animation, then the game just won't register it. So if you have no jump animation, the entity may just "appear" at the top of the block, without the jump animation applied.

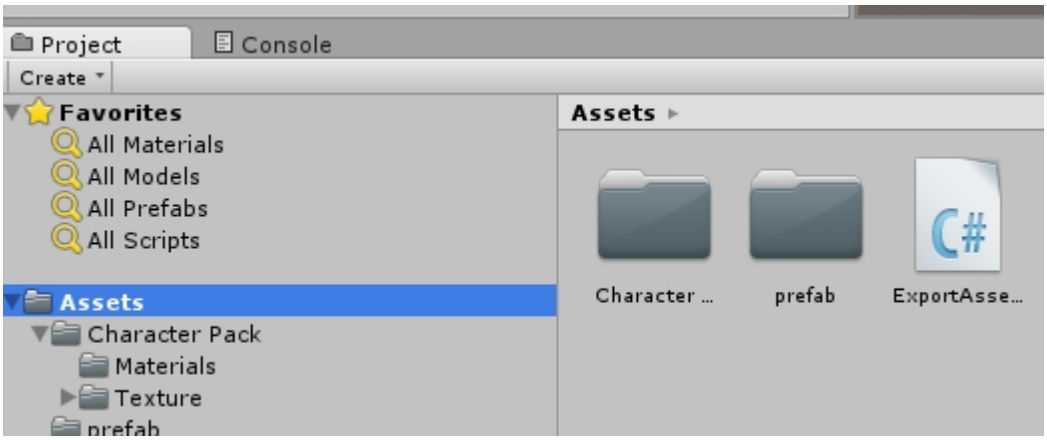
If you need to expand, and add more elements, change the 6 to an 8 or 9, however many you need. Press enter for it to take effect.

Once you have selected all your animations, your screen should look like this:

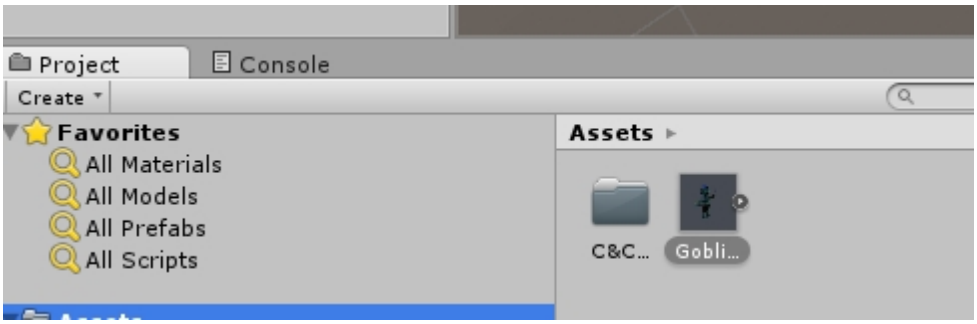
Exporting the Entity

Once you have the Goblin has been all tweaked, it's time to export it. These are the same steps as you'd do to export a block.

In your Project window, click on the Assets Folder.



In the Hierarchy window, click and drag the Goblin_route_b down to the Assets window.

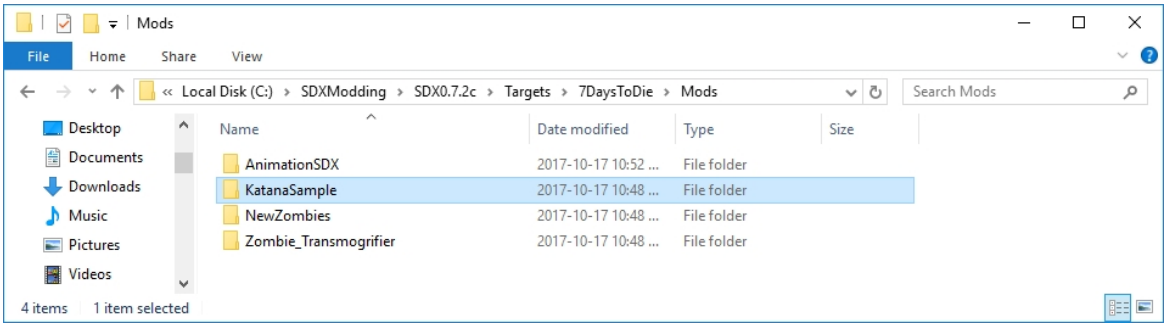


Right click on the Goblin_route_b in the Asset Window, and select "Build AssetBundle ...". Save the Resource as Goblin.unity3d.

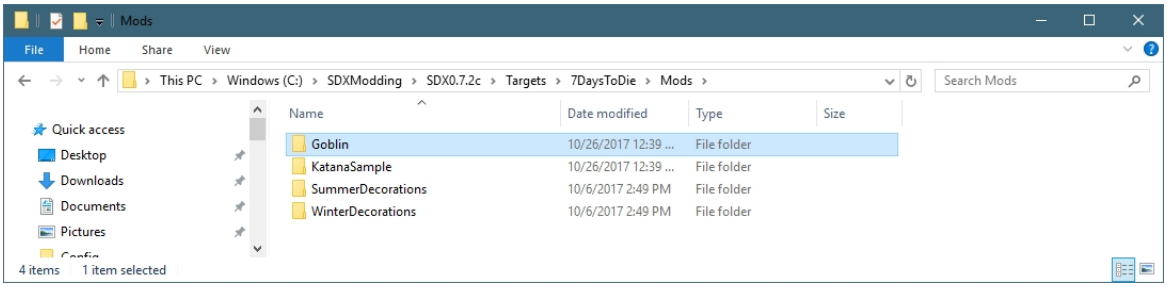
Creating the SDX Mod Folder

We'll now want to create an SDX mod folder structure for this Mod.

To make things easier, we'll make a copy of the Katana Mod Folder



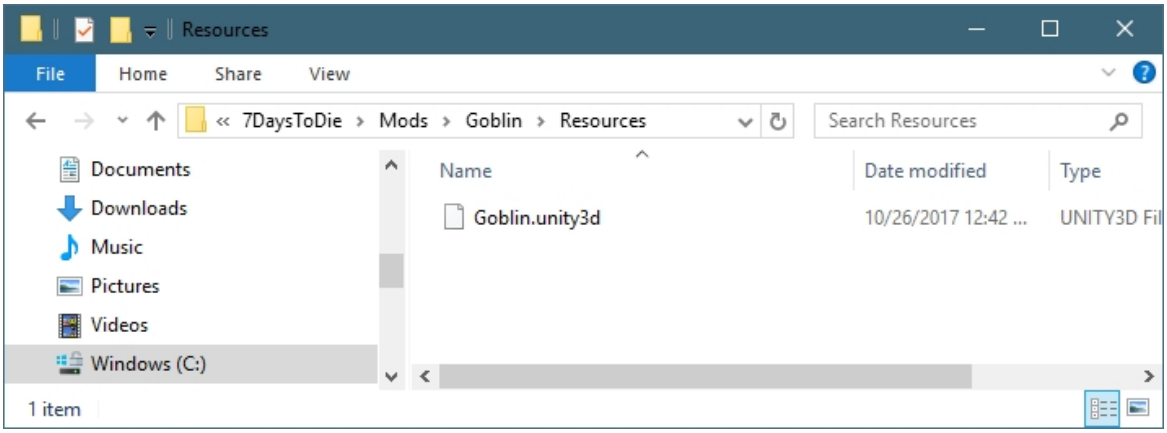
Right click on the KatanaSample, and rename to Goblin.



Edit the Mod.xml, and update as needed. You only need to change the <name> to show up uniquely in the SDX Launcher.

```
<mod>
      <info>
            <!-- Information about the Mod, the author and version
information -->
            <!-- These are displayed in the SDX Launcher -->
            <author>sphereii</author>
            <name>Goblin</name>
            <description>Goblin Animation Demo</description>
            <mod_version>1.0</mod_version>
            <game_version>16.3</game_version>
            <launcher_version>0.0.0</launcher_version>
      </info>

      <!-- This references any config files that SDX needs to merge into
your files -->
      <config_mods>
            <import file="Config\Katana.xml" />
    </config_mods>

</mod>
```

Under Resources, delete the michonnekatana.unity3d, and copy in your Goblin.unity3d that you exported.



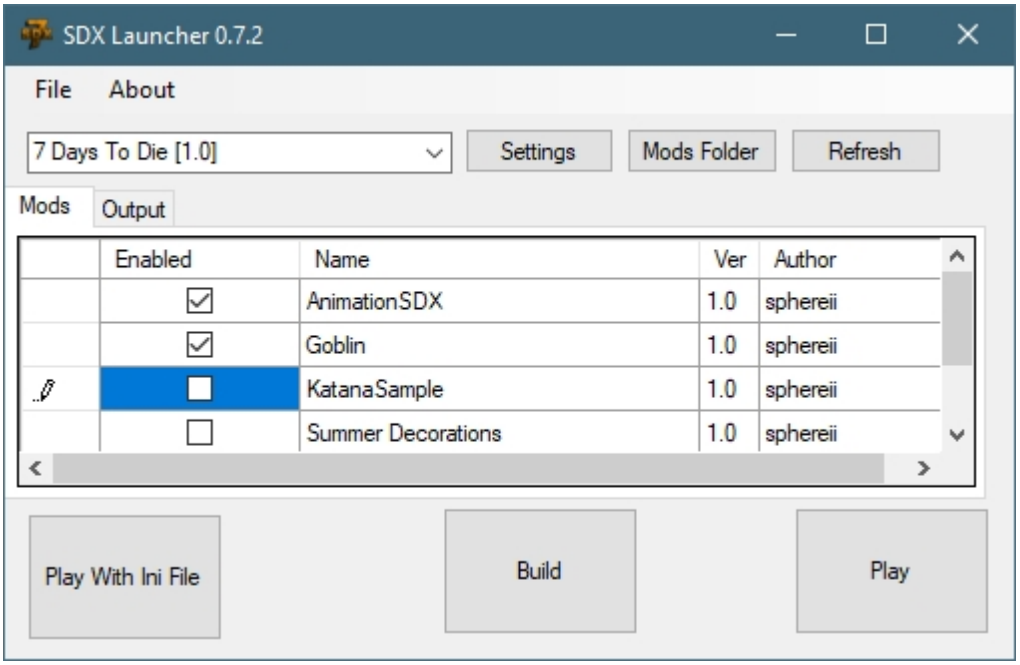
In the Config folder, you can leave the name of the file as is, and just update the contents. in the Value fields of the animation, we picked the animations that we saw in the Animations section in unity. We are extending from the SDXTemplate, which is an entity_class added as part of the AnimationSDX package. You do not need to extend from it, however, the settings in the SDXTemplate seem to apply well to all the different entities.

```
<configs>
      <config name="entityclasses">
            <append xpath="/entity_classes">

                  <entity_class name="Goblin" extends="SDXTemplate" >
                        <property name="Mesh" value="#Goblin?Goblin_rouge_b"
/>

                        <!-- These are the animation map names. The value
comes from the animation from Unity -->
                        <property name="AnimationIdle" value="idle" />
```

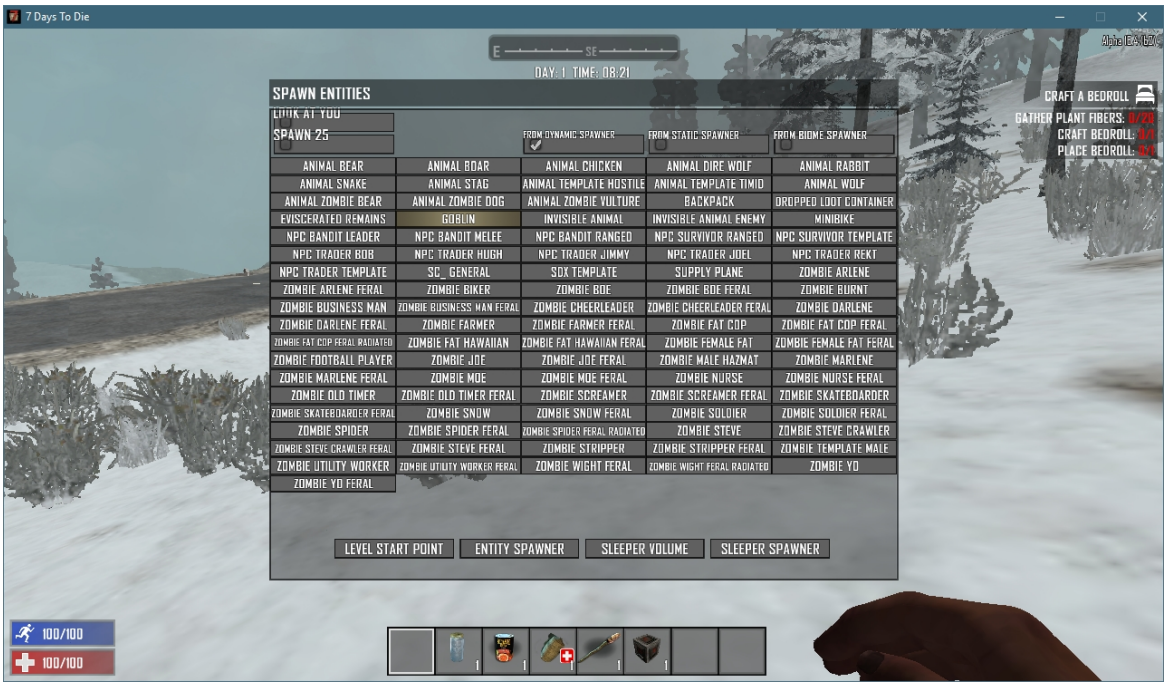
```
                        <property name="AnimationMainAttack" value="attack1"
/>
                        <property name="AnimationPain" value="hit1" />
                        <property name="AnimationDeath" value="death1" />
                        <property name="AnimationRun" value="run" />
                        <property name="AnimationWalk" value="walk" />
                    </entity_class>
            </append>
      </config>
</configs>
```


Building the SDX Mod

In the SDX Launcher, be sure to include the AnimationSDX mod!



Launch the game after it's built, and look in the spawn menu for the Goblin



How to add Custom Sounds

This tutorial will help cover how to add custom sounds, like .wav files, into the game, and how to access them.

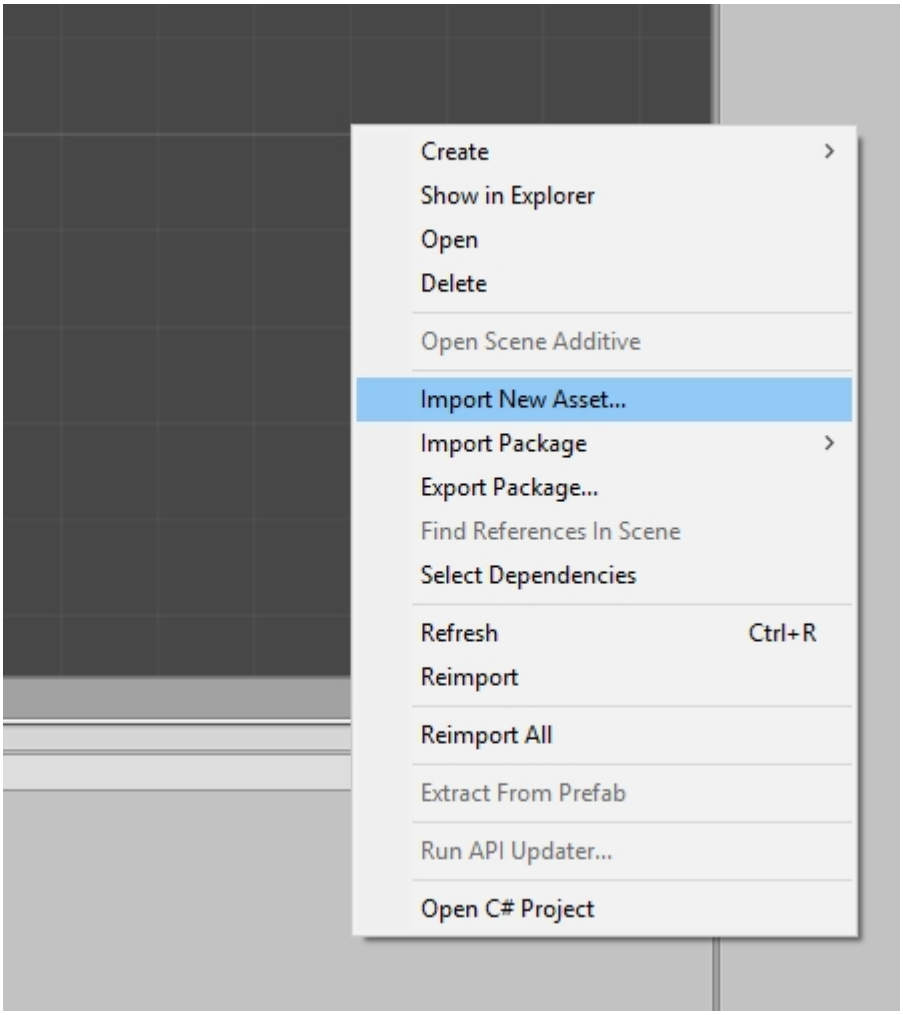
There are many free sound effects you can find online. For example, soundbible.com is a free sound site that has thousands of sound effects. For this tutorial, we are going to download a wav file from soundbible, create an asset bundle, then write the XML to add it to the game.

http://soundbible.com/free-sound-effects-1.html

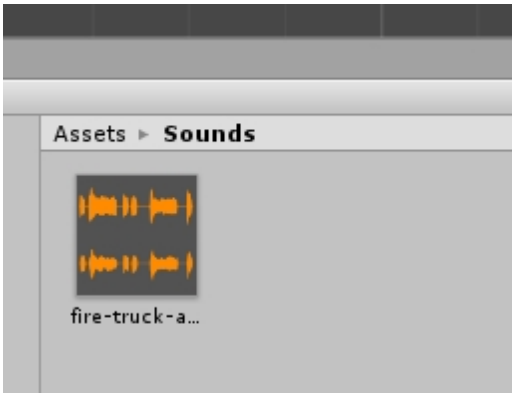
You can choose any sound you want. For this tutorial, we are going to use the Fire Truck Horn sound by Daniel Simion: http://soundbible.com/2192-Fire-Truck-Horn.html

You can click on the Play button and hear the sound effects. From there, click on the WAV icon to download the file.

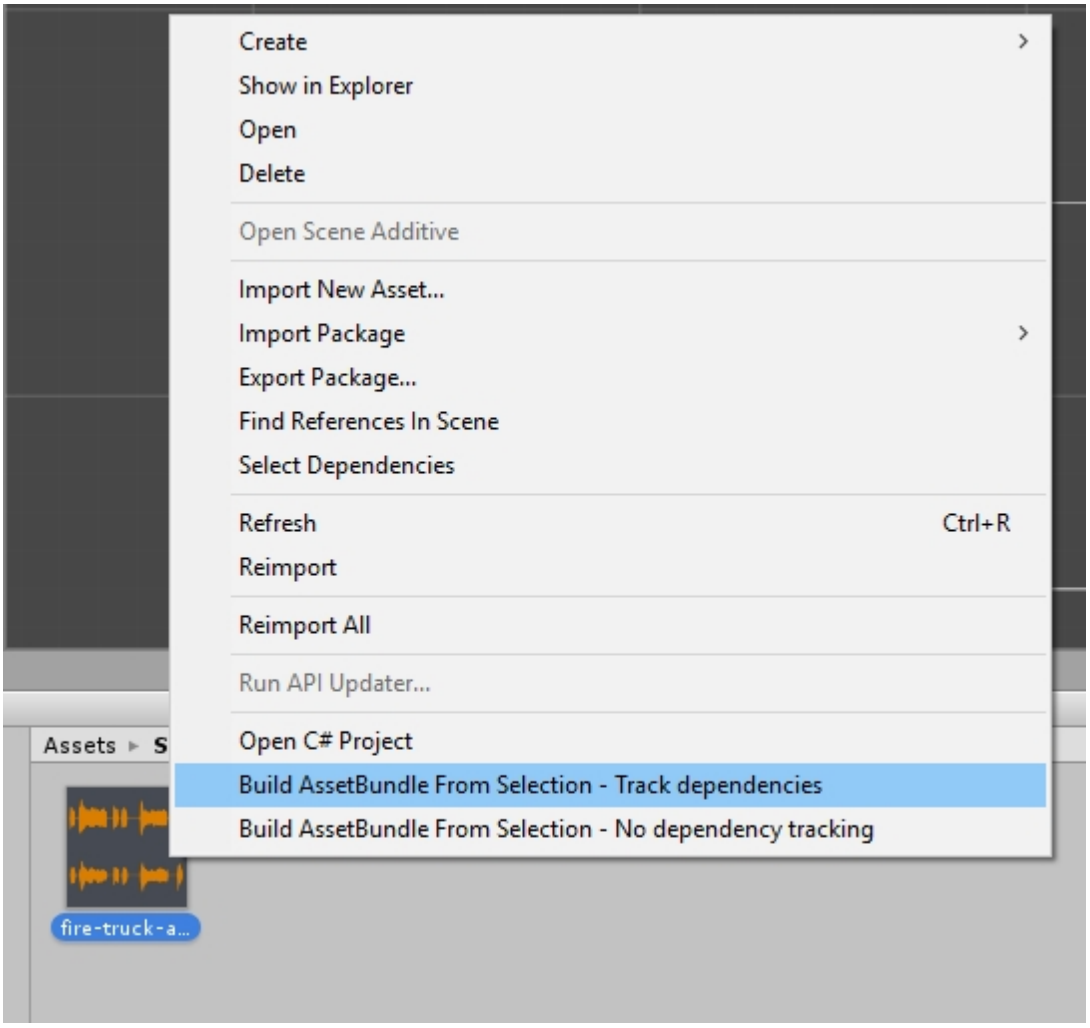
Once downloaded, go into Unity and right click in the Asset pane, and select Import New Asset



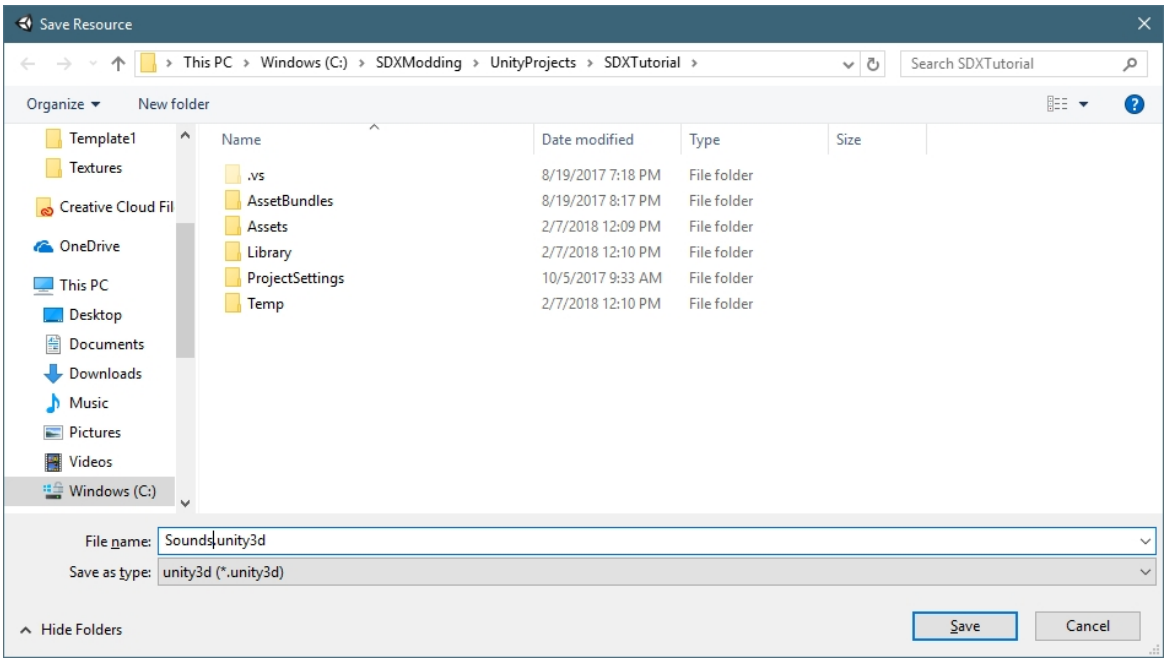
And select your new Wav file. After a few seconds, it'll appear in your Asset window.



To export, you do the same steps as you would for a block or entity, by right clicking, and generating an Asset Bundle. You can multi-select the sounds as well.



For this example, we are saving it as Sounds.unity3d, and place it in your Resources folder.



In your Config XML, you'll reference the sounds like this:

```
<configs>
      <config name="sounds">
            <append xpath="/Sounds">
                  <SoundDataNode name="FireTruck">
                        <AudioSource name="Sounds/AudioSource_VO" />
                        <Noise ID="3" range="100" volume="25" time="10"
muffled_when_crouched="0.65" heat_map_strength="0.05" heat_map_time="60" />
                        <AudioClip ClipName="#Sounds?fire-truck-air-
horn_daniel-simion" />
                        <LocalCrouchVolumeScale name="1.0" />
                        <CrouchNoiseScale name="0.5" />
                        <NoiseScale name="1" />
                        <MaxVoices name="10" />
                        <MaxRepeatRate name="1.111" />
                  </SoundDataNode>
            </append>
      </config>
</configs>
```

Notice the ClipName is the Unity3d file, along with the name of the asset itself.

To attach the new clip to an entity, you can adjust the value to point to the new SoundDataNode:

```
<property name="SoundSense" value="FireTruck" />
```


How to create an Animator State Machine

** WORK IN PROGRESS **

A previous "How To" guided you through the steps on how to add animations to a custom entity, using the AnimationSDX class. This was limited to using Legacy animations on the entity, and limited us on what kind of asset we wanted to add to the game. We typically had to rely on baked in animation for that particular asset.

A new mecanim class has been created, which allows us to use standard Motion Captured (mocap) animations, and add them to entities that are rigged, but not animated.

Using mecanim opens up many new features, including using non-Legacy animations, and setting state transitions, allowing a smoother transition between animations. It also gives you control, in Unity, in how to set up transitions, swap out animations, all without using or changing any code.

Here's what Unity has to say about it.

This tutorial will guide you through the process of creating an Animator Controller in Unity, adding in animations, control flow, and states, and finally add a new entity in game.

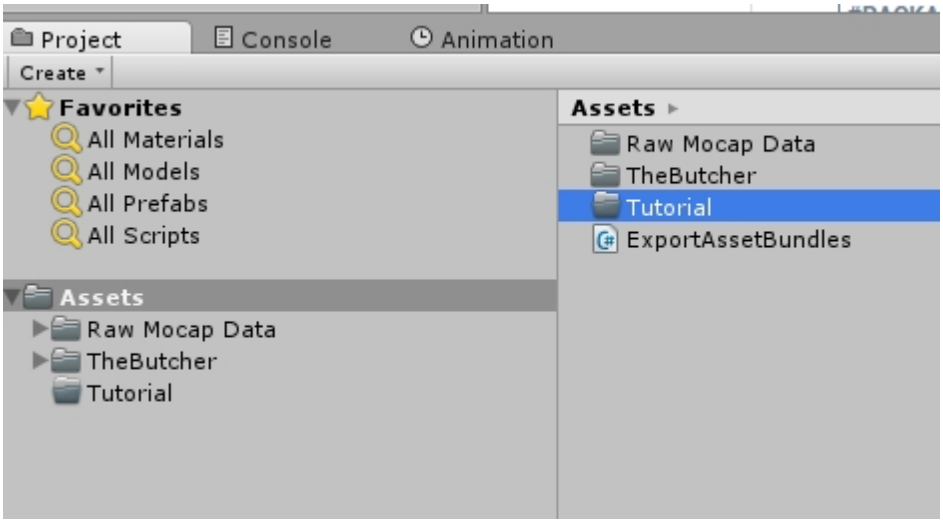
Initial Tutorial Setup

Create a new Unity 5.3 project, and import these Unity assets from the Asset Store:

The Butcher, by Shaun L.: https://www.assetstore.unity3d.com/en/#!/content/61178
Fighting motions Vol 1., by Magicpot Inc:https://www.assetstore.unity3d.com/en/#!/content/76699

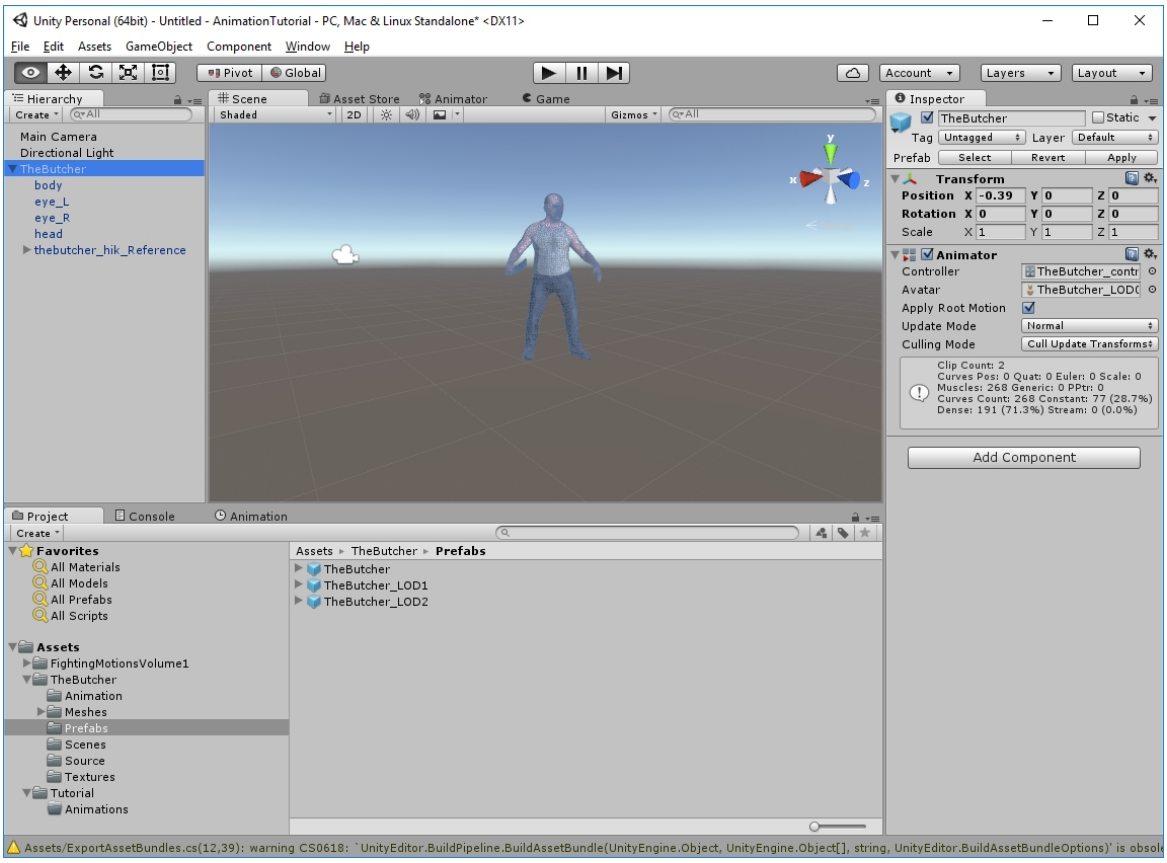
We'll be using the Butcher, by Shaun L. It's a free, rigged humanoid asset. We'll also be using the Fighting Motions Vol 1. Once you are familiar with the concepts, you can explore other mocap projects.

In the Project view, under the Assets folder, right click and go to Create New Folder, and call it "Tutorial".

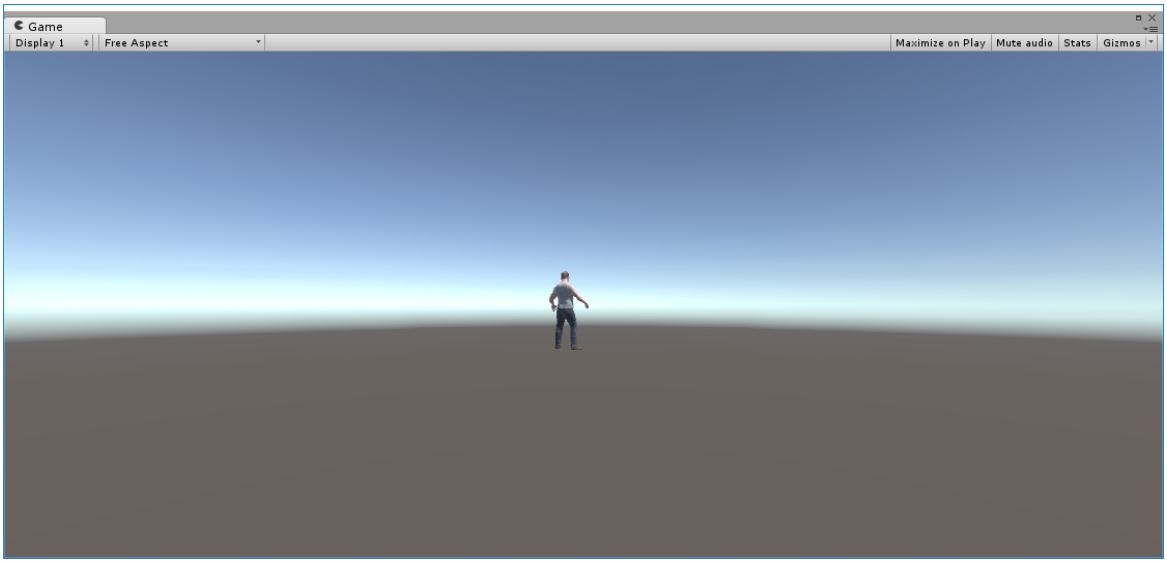


Looking at the Butcher's Animator Controller

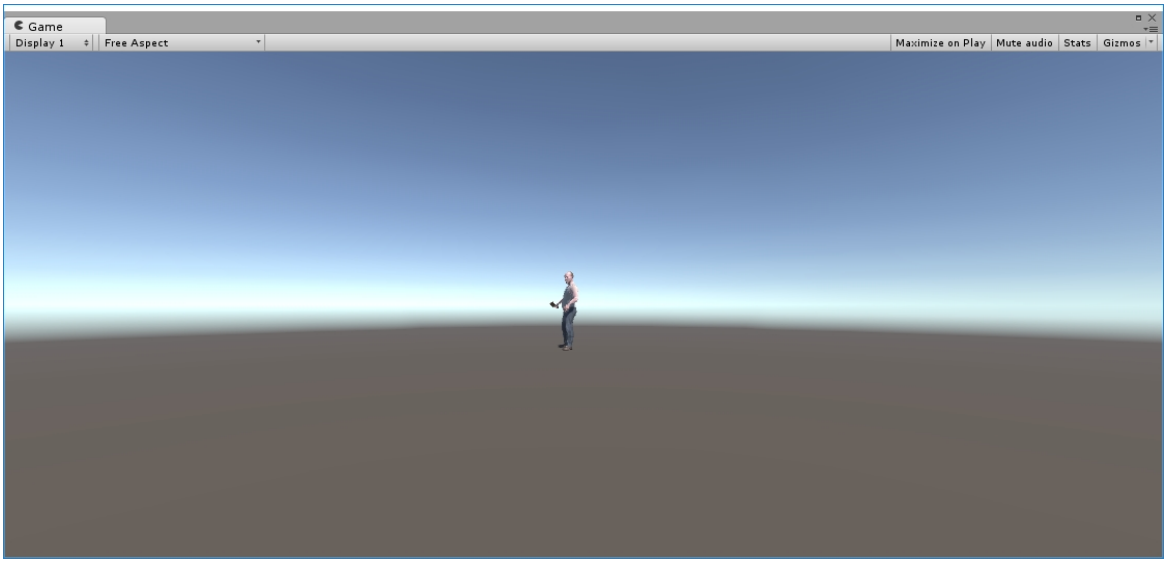
In the Project View, under TheButcher, look for TheButcher.prefab under TheButcher/Prefabs folder, and drag it into your Hierarchy screen. This will be our template entity.



Notice that The Butcher already has an Animator Component, in the Inspector window. Let's take a look at his Animator Controller to give a hint on what we'll be creating. If you have a Game tab, click and drag it off of the main Unity screen, so that it's separate window.



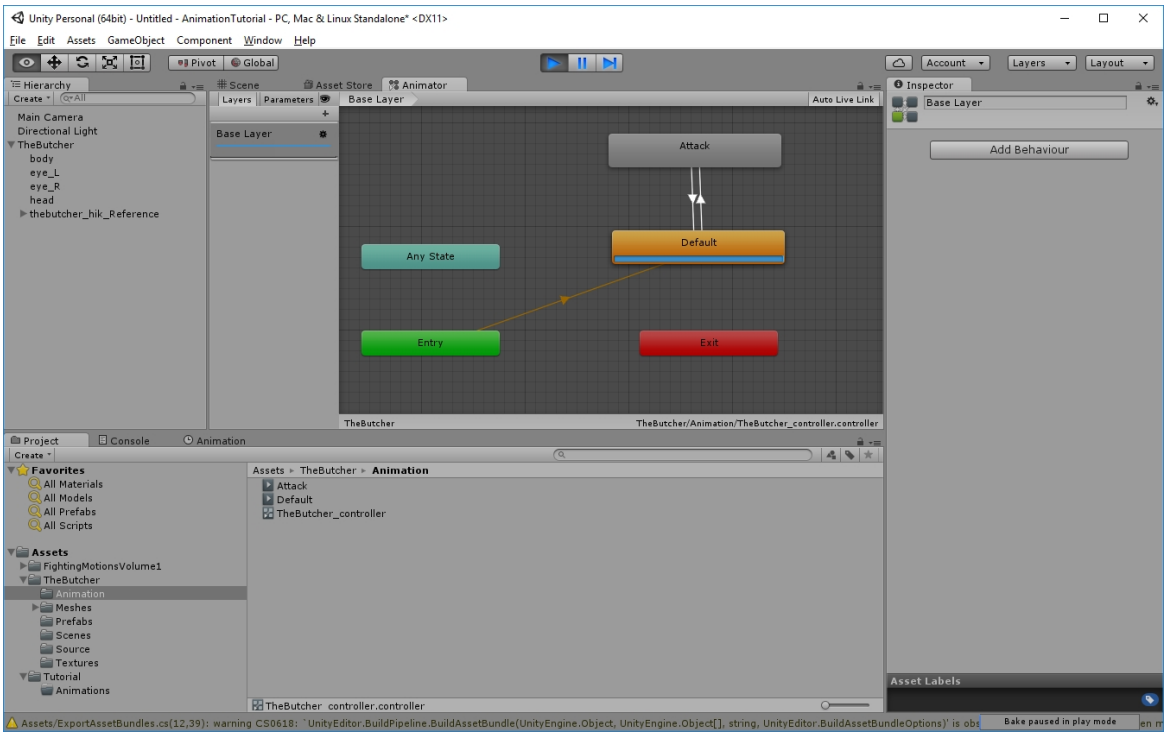
If you see that the Butcher has his back to you, we'll want to turn him around so you can see his animations better. To turn him around, go to the main Unity Window, click on the Butcher in the Hierarchy. This will open up his Inspector window. Change the Y rotation to 220.



In the main Unity screen, you'll see a Play button. When depressed, it'll play the Animation in the Game tab. Press the Play button, and he'll swing his butcher knife once, then stop.

Let's see what's happening.

In the Unity main screen, click on the Animator tab. If you do not have an Animator tab, go to Window, and select Animator.



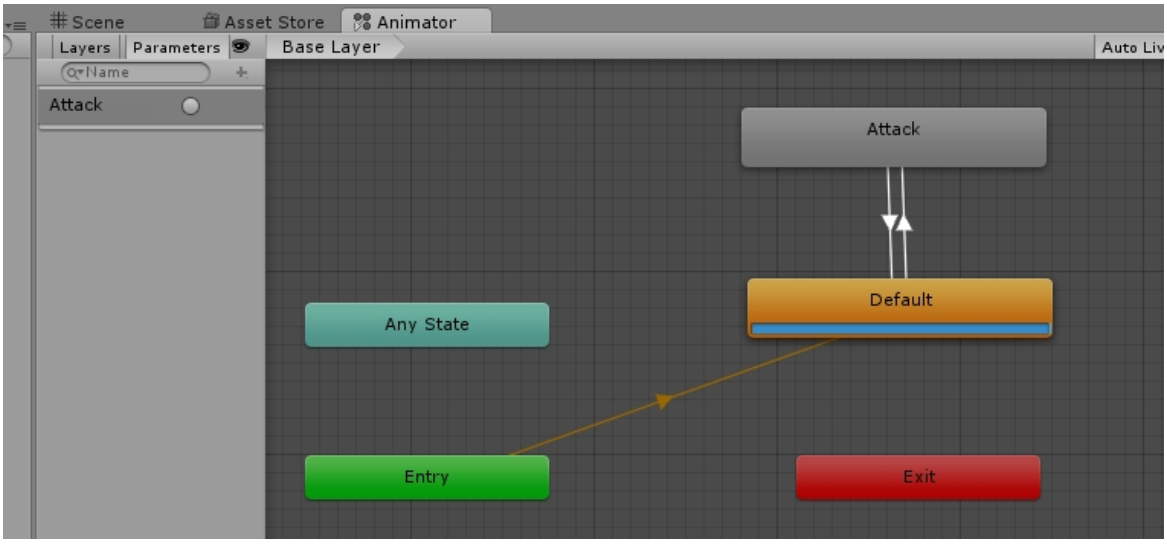
In the Animator, you can see some colourful boxes. These are called States, and the lines between them are Transitions. A transitions allow us to flow from one state to another, based on the transition Conditions on the Transitions. The states themselves can be animation clips, Entry / Any State / Exit states, Empty states or Sub-States. We'll cover these more in depth later.

If the Play button is still blue, press it again to turn it off, then turn it back on. You'll see Default briefly flash, then the Attack state lights up with a progress bar, as it goes through the animation, as viewable through

the Game screen.You'll also notice, briefly, that the Transition Line from Default lights up and can see it going up into Attack, then, once the Attack animation is complete, it goes back down the other Transition line.

Above the Base Layer, you'll see a Layers and Parameters tab. Layers allow you to have complex animations in which two or more parts of the entity can be animated differently. Parameters is what triggers our events.

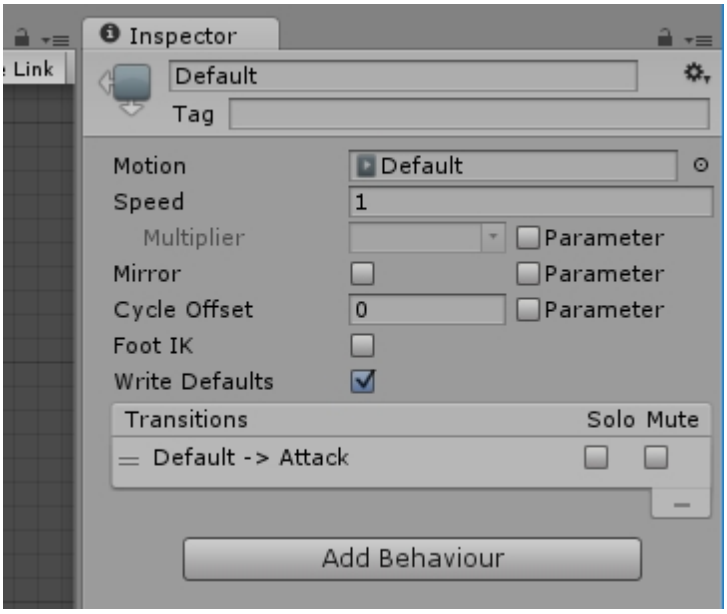
Let's click on the Parameters tab, and see what the Butcher has by default.



Under Parameters, we see Attack. With the Play button still blue, click on the circle beside the Attack. You'll see the blue Transition dot go up the Transition line, display the Attack Animation, then flow back down to the Default again.

Let's see what's really happening here.

Click on the "Default State", and it will bring up the Inspector window.



The Inspect window shows the name, and other information about that state.

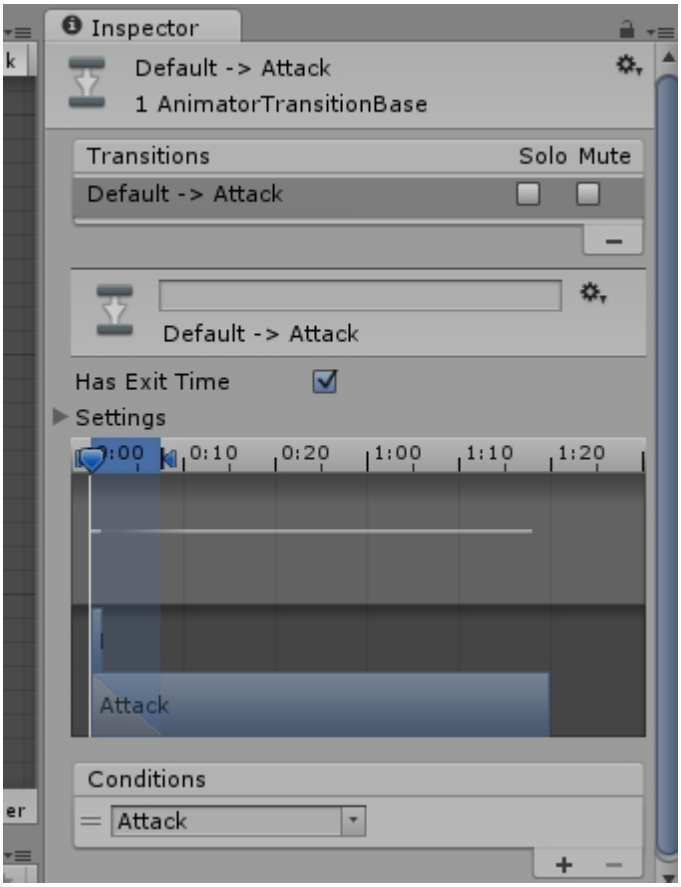
"Default" is the name of the current state. You can rename any States that you want by clicking on the name in the Inspector window.

The Motion is the actual Animation Clip that is attached to that State. Not all States have an Animation clip, however.

We'll rarely change anything on the State itself, so this information is provided purely for informational value:

Property:	Function:
Speed	The default speed of the animation
Motion	The animation clip assigned to this state
Foot IK	Should Foot IK be respected for this state. Applicable to humanoid animations.
Write Defaults	Whether or not the AnimatorStates writes back the default values for properties that are not animated by its Motion.
Mirror	Should the state be mirrored. This is only applicable to humanoid animations.
Transitions	The list of transitions originating from this state

Let's click on the Transition line going from Default, to Attack, to bring up it's Inspector window.

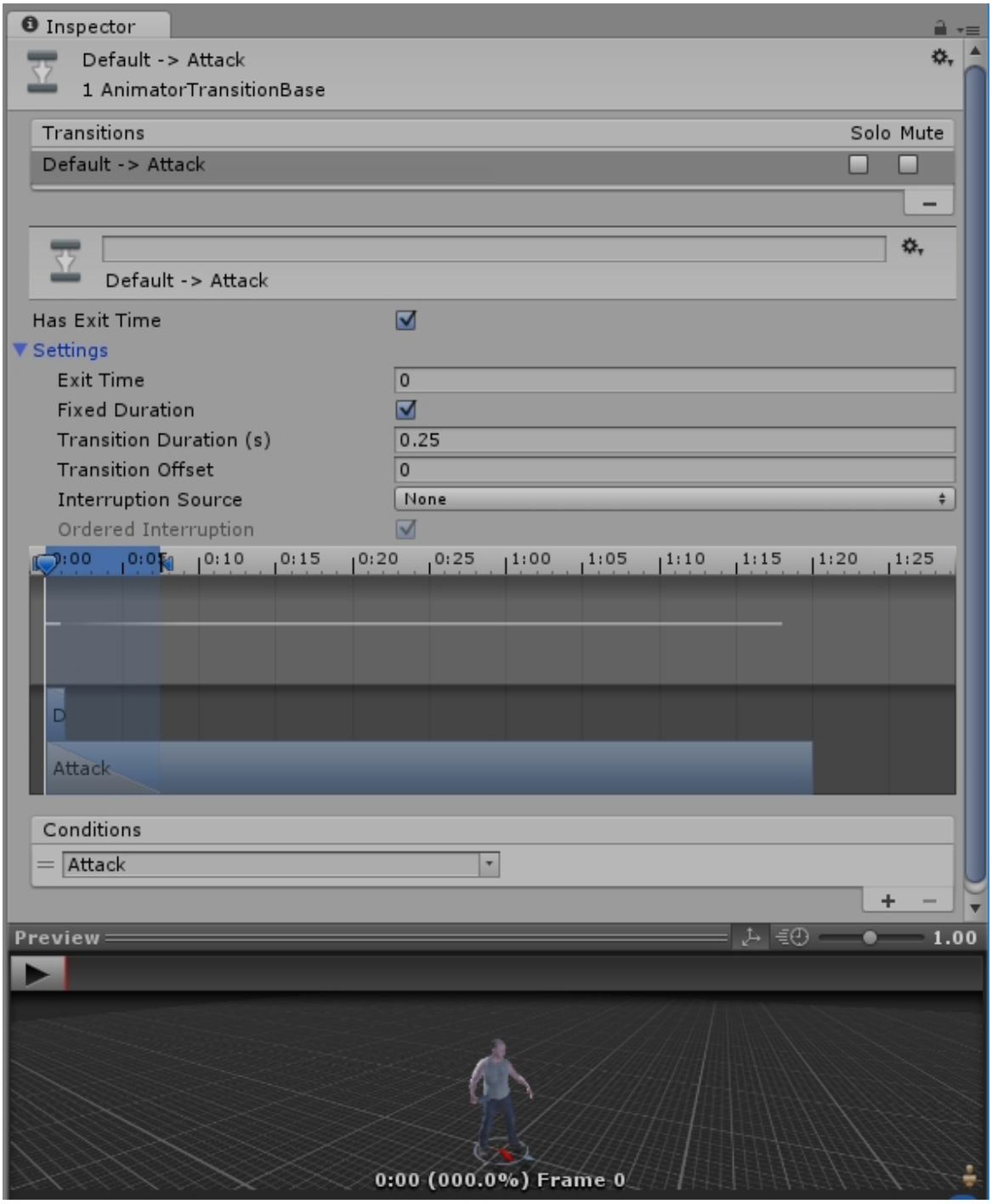


Under Transitions, it shows you all the transitions you have. It's going from "Default" state -> "Attack" state.

The Solo and Mute are only used for testing and isolated small parts of a complicated state machine, and are not used for production.

The Has Exit Time tells the the Transition that the State needs to finish it's Animation before triggering a transition.

Click on the arrow beside the Settings label, and expand all the settings.



Under the Conditions settings, you see "Attack". This is where you can add and remove your parameters you've set. The Attack matches the Parameter, so when we click on the Attack circle, it fires off the state transition automatically for us. You can have multiple Conditions on each Transition, in which all must evaluate to True for a successful transition.

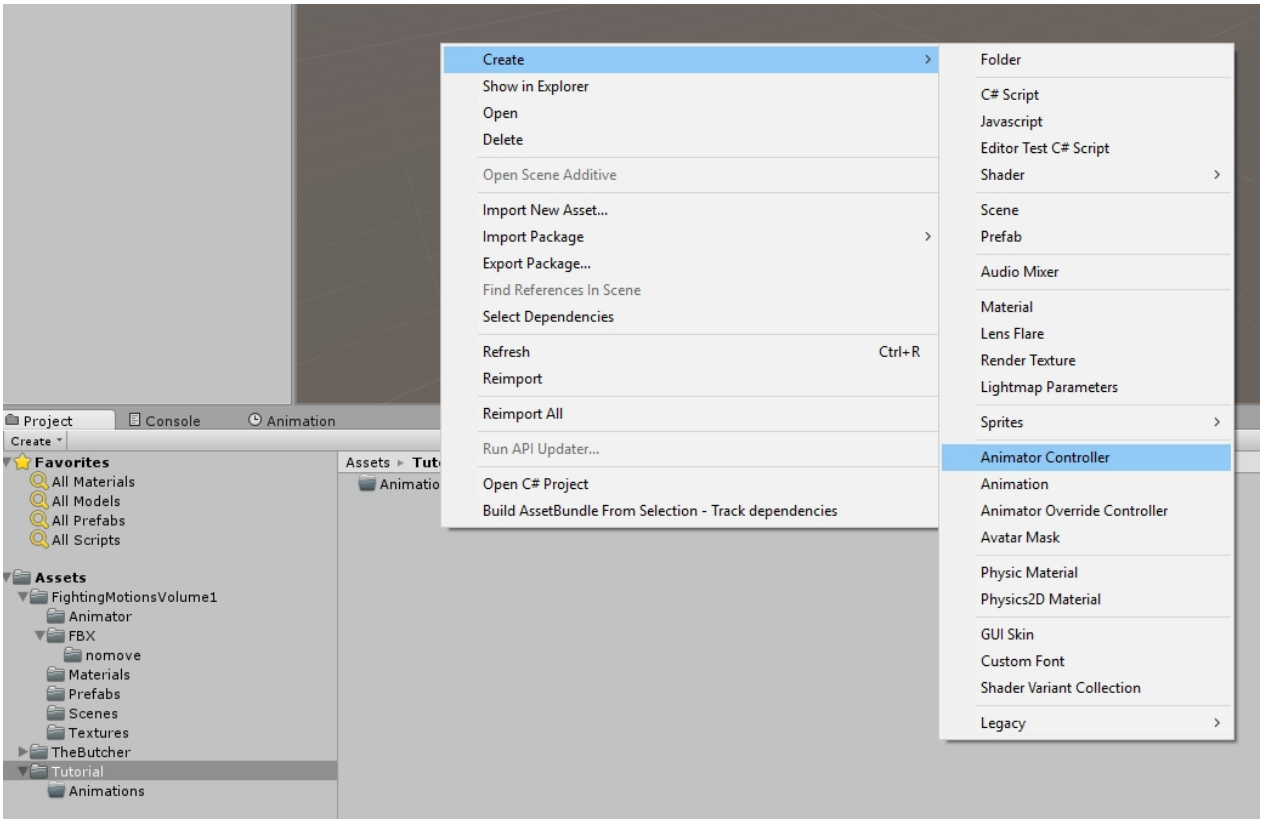
The other options in this screen are as follows:

Property	Function
Has Exit Time	**Exit Time** is a special transition that doesn't rely on a parameter. Instead, it relies on the normalized time of the state. Check to make the transition happen at the specific time specified in **Exit Time**.
Settings	Fold-out menu containing detailed transition settings as below.
Exit Time	If **Has Exit Time** is checked, this value represents the exact time at which the transition can take effect. This is represented in normalized time (for example, an exit time of 0.75 means that on the first frame where 75% of the animation has played, the **Exit Time** condition is true). On the next frame, the condition is false. For looped animations, transitions with exit times smaller than 1 are evaluated every loop, so you can use this to time your transition with the proper timing in the animation every loop. Transitions with an **Exit Time** greater than 1 are evaluated only once, so they can be used to exit at a specific time after a fixed number of loops. For example, a transition with an exit time of 3.5 are evaluated once, after three and a half loops.
Fixed Duration	If the **Fixed Duration** box is checked, the transition time is interpreted in seconds. If the **Fixed Duration** box is not checked, the transition time is interpreted as a fraction of the normalized time of the source state.
Transition Duration	The duration of the transition, in normalized time or seconds depending on the **Fixed Duration** mode, relative to the current state's duration. This is visualized in the transition graph as the portion between the two blue markers.
Transition Offset	The offset of the time to begin playing in the destination state which is transitioned to. For example, a value of 0.5 means the target state begins playing at 50% of the way through its own timeline.
Interruption Source	Use this to control the circumstances under which this transition may be interrupted (see Transition interruption below).
Ordered Interruption	Determines whether the current transition can be interrupted by other transitions independently of their order (see Transition interruption below).
Conditions	A transition can have a single condition, multiple conditions, or no conditions at all. If your transition has no conditions, the Unity Editor only considers the **Exit Time**, and the transition occurs when the exit time is reached. If your transition has one or more conditions, the conditions must all be met before the transition is triggered. A condition consists of: - An event parameter (the value considered in the condition). - A conditional predicate (if needed,for example, 'less than' or 'greater than' for floats). - A parameter value (if needed). If you have **Has Exit Time** selected for the transition and have one or more conditions, note that the Unity Editor considers whether the conditions are true after the **Exit Time**. This allows you to ensure that your transition occurs during a certain portion of the animation.

Creating A new Animator Controller

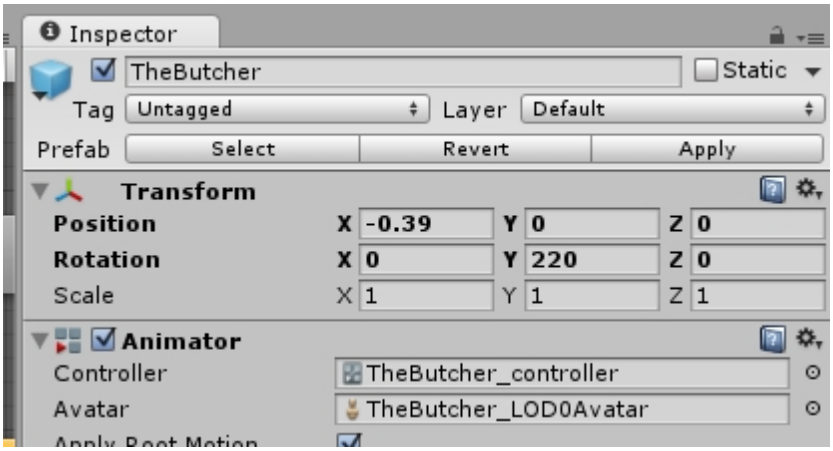
An Animator Controller is a Unity Object that acts like a state machine for animations. That is, it allows us to assign animation clips to it, then set up rules on when those animation clips are allowed to trigger.

In the Tutorial project, right click, go to Create and select Animator Controller.



By default, this will create a "New Animator Controller" object. You can double click, and rename it to whatever you'd like. For this tutorial, we are calling it the TutorialController.

Once created, click on TheButcher under the hierarchy to bring up the Inspector window.



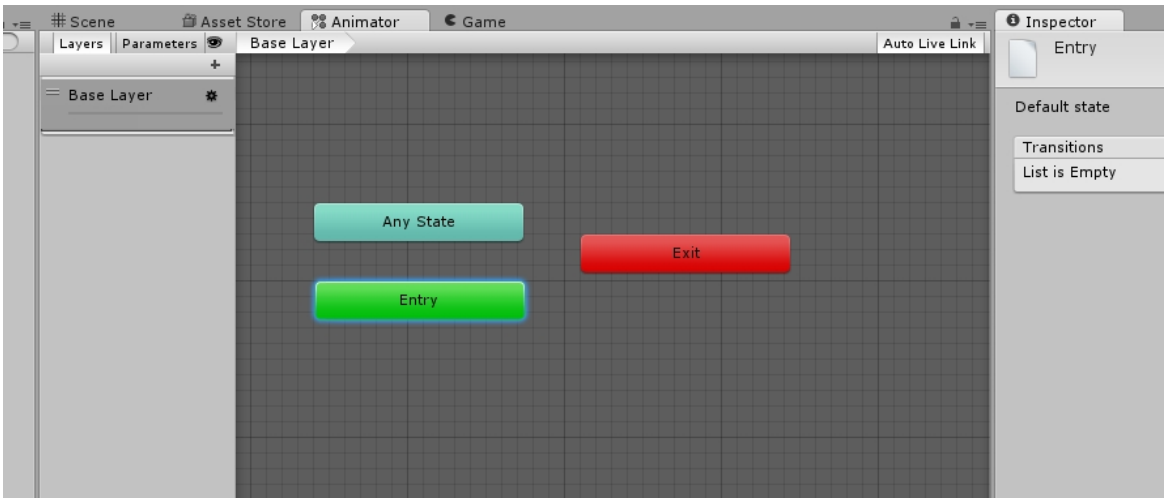
Under the Animator section, click on the circle at the end of the TheButcher_controller, and select the new Animator Controller you created. Once updated, you'll notice that the Clip Count is now 0. That's because we haven't added any animation clips yet.

Now The Butcher has been assigned our new Animator Controller, and is ready to learn new animations.

The Animator Window

The Animator window allows you to add animation clips, triggers, and control the flow of the animation sequences.

You can open the Animator window by double clicking on your new Animator Controller, created in the previous step.



Inside the window, you have the following default events:

Entry: is the start point, and will auto-link with your first animation clip you enter. This sets up the default animation that will be played when the Animator is activated

Any State : allows us to add animations that can be fired from any other event or trigger. For example, to allow jumping and attacking from any state, you'd link those animation clips from Any State.

Exit: is the end point, and causes the Animator controller to exit. In the Unity Editor window, Exit will immediately loop back to Entry.

Under the Scene tab, you'll see two other options: Layers and Parameters.

Layers - This allows you to sort complex animations, such as if the bottom torso of your avatar must behave independently from your top of the avatar. We'll be using the default base layer for this tutorial.

Parameter - This allows you to trigger events for your animator to execute. A Parameter can be a float (0.0), Integer (1,2,3,etc) a bool (true / false) or a Trigger ("Attack"). We'll be using Parameters for state machine.

Tips

You can click on any State and move it around the panel by dragging it.

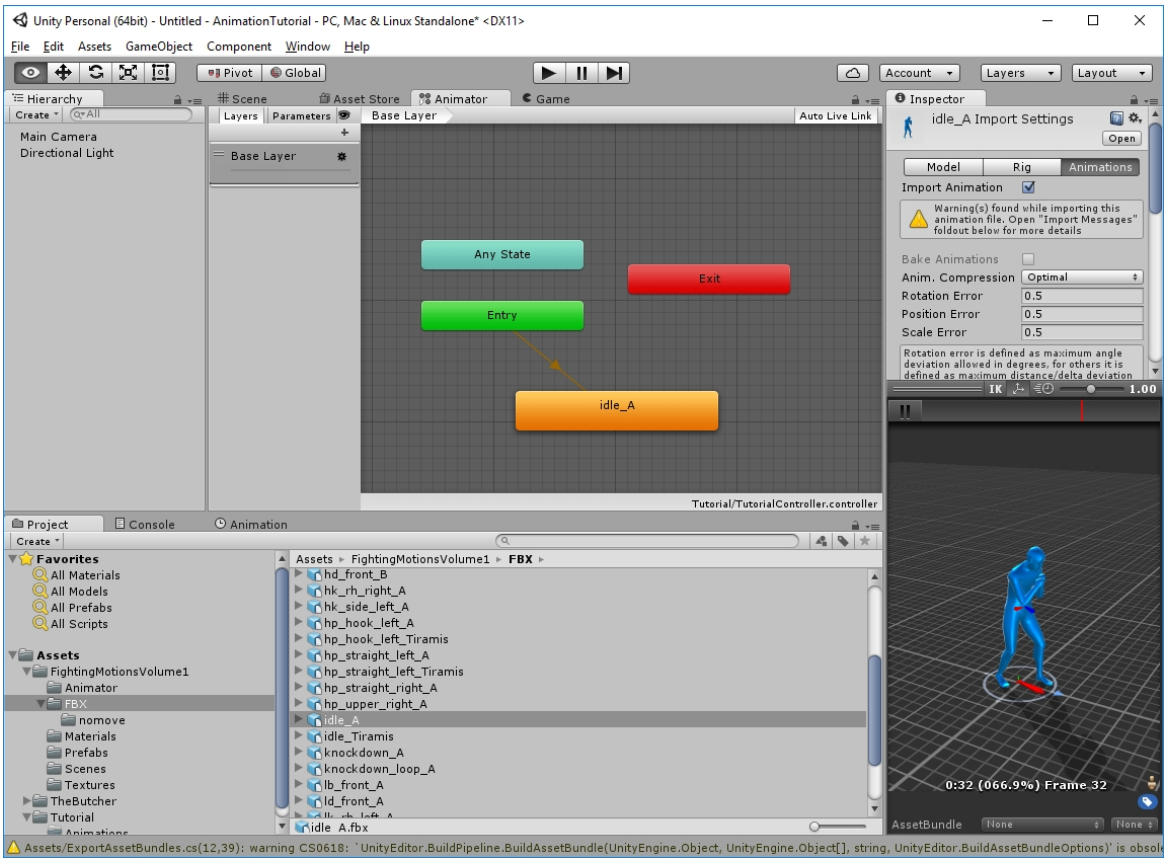
You may hold down Alt, and click to drag and move the window around as well.

Adding Animation Clips to the Animator Controller

Now we are going to add an Animation clip to the Animator Controller, and set up a few parameters to allow us to control it.

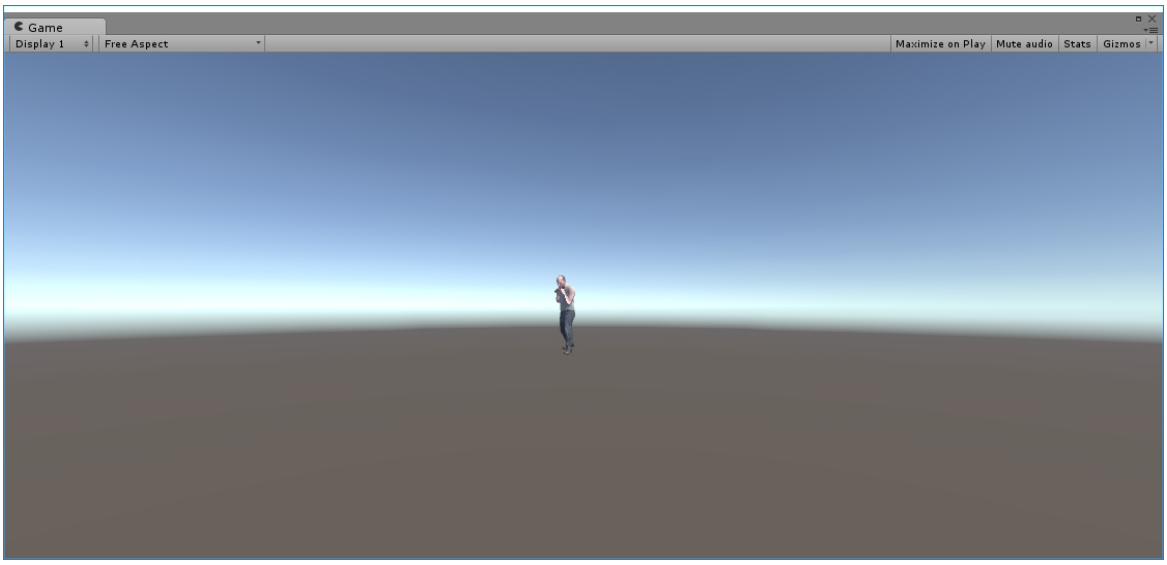
For starting off, we will be adding a state for the character to idle in.

Under the Fighting Motions Volume 1, look for the FBX folder, and the idle_A animation clip. Once you've found it, drag the Idle_A clip up into the Animator Window.



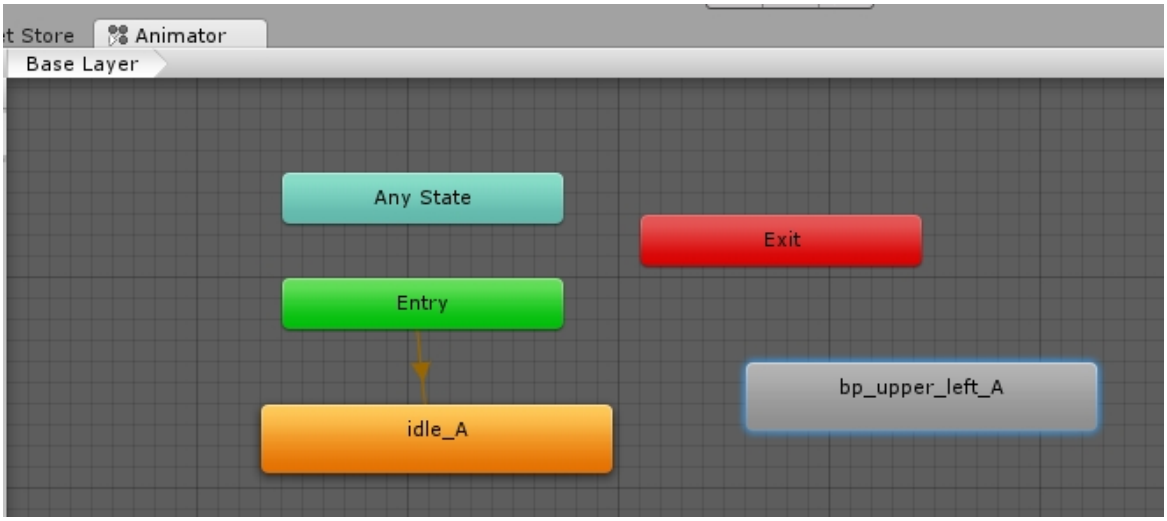
Because there are no other user-defined States, a link is auto-generating from the Entry State. This is now our starting state.

Now press the Unity Play button, and watch the Game screen for The Butcher's new animation, which should be a boxing pose.



Adding an Attack

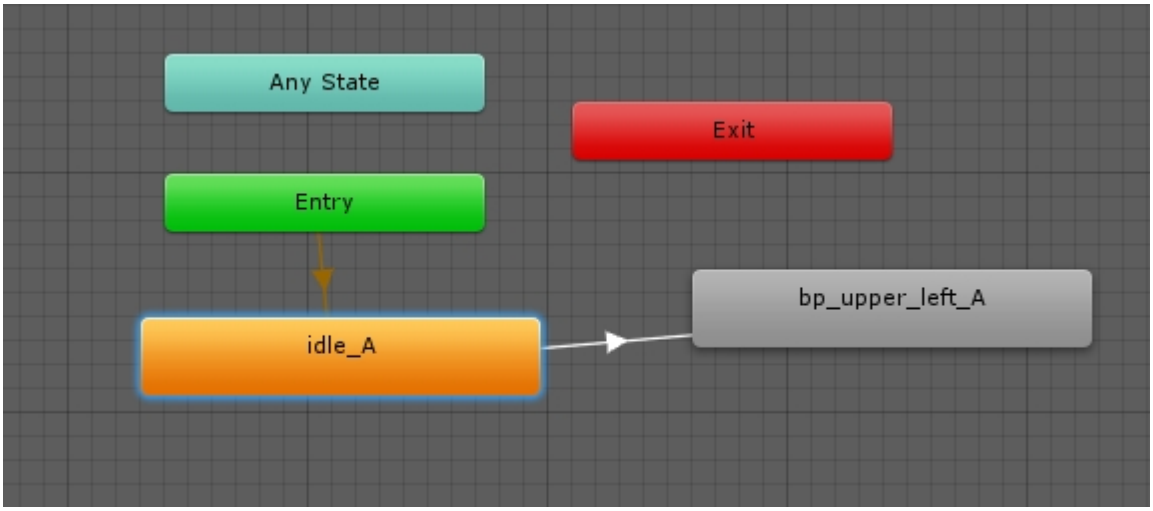
Let's give him an attack animation. Look for the animation called "bp_upper_left"A, and drag it into the Animator window.



A new state is created, with the name of the Animation Clip as its name.

Since our default state is idle_A, we want the Butcher to punch as an attack. Let's create a transition from Idle_A to bp_upper_left_A.

Right click on idle_A, and click on Make Transition. Then, move your mouse over to bp_upper_left_A. As you move the mouse, you'll see a line from idle_A extend from it. Once you are over bp_upper_left_A, click on it to make the state connection.



Now, click on the Play button again. You'll notice that the Butcher will go into pose, then perform the punching animation, and stop.

Because we didn't set any conditions on the transition from Idle_A to bp_upper_left_A, the transition occurred immediately after the idle_A finished it's first loop. When bp_upper_left_A completed it's animation sequence, it stopped, because it didn't have any transitions to anywhere else.

Let's make a transition back to Idle. Right click on bp_upper_left_A, and click on Make Transition, then click on Idle_A.

Now when the animation plays, it goes from Idle_A, to bp_upper_left_A, then back to Idle, and continues. Again, since there are no conditions on either transition, the animation loops through both states automatically.

Adding Parameters to the Animator Controller

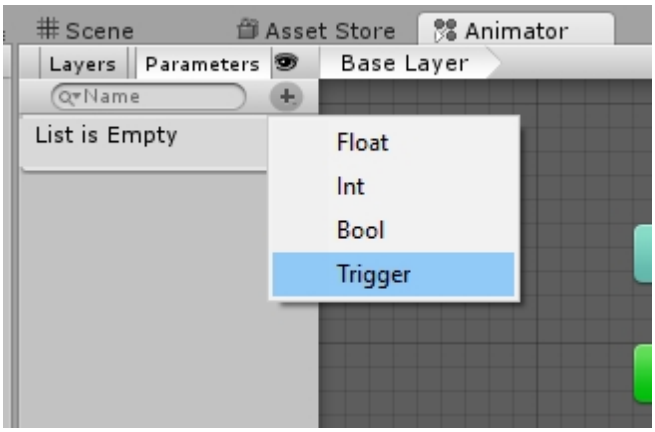
Adding A Parameter (or Event)

The Butcher will get exhausted if we continue to let him punch forever. Let's add a Parameter, so that we have more control over when he punches.

A Parameter is a setting that allows us to control and fire off conditions, either through the Unity screen, or through the code in the mecanim class. These are our hooks that connect the game's code to the Animator State machine, and allows our entities to be animated and active in-game.

If the Play button is pressed and the animation is running, please click it again to stop it.

In the Parameters tab, you'll see an entry that says "List is Empty". Click on the plus button.



We have four options: Float, Int (Integer), Bool, or Trigger.

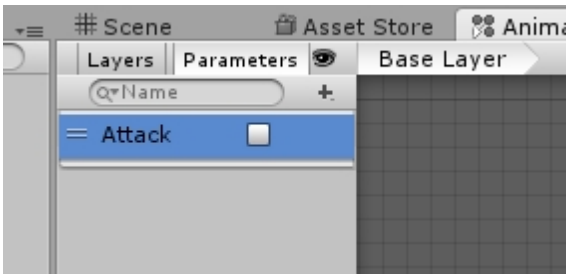
- *Int* - an integer (whole number)
- *Float* - a number with a fractional part
- *Bool* - true or false value (represented by a checkbox)
- *Trigger* - a boolean parameter that is reset by the controller when consumed by a transition (represented by a circle button)

Parameters can be assigned values from a script using functions in the Animator class: SetFloat, SetInt, SetBool, SetTrigger and ResetTrigger.

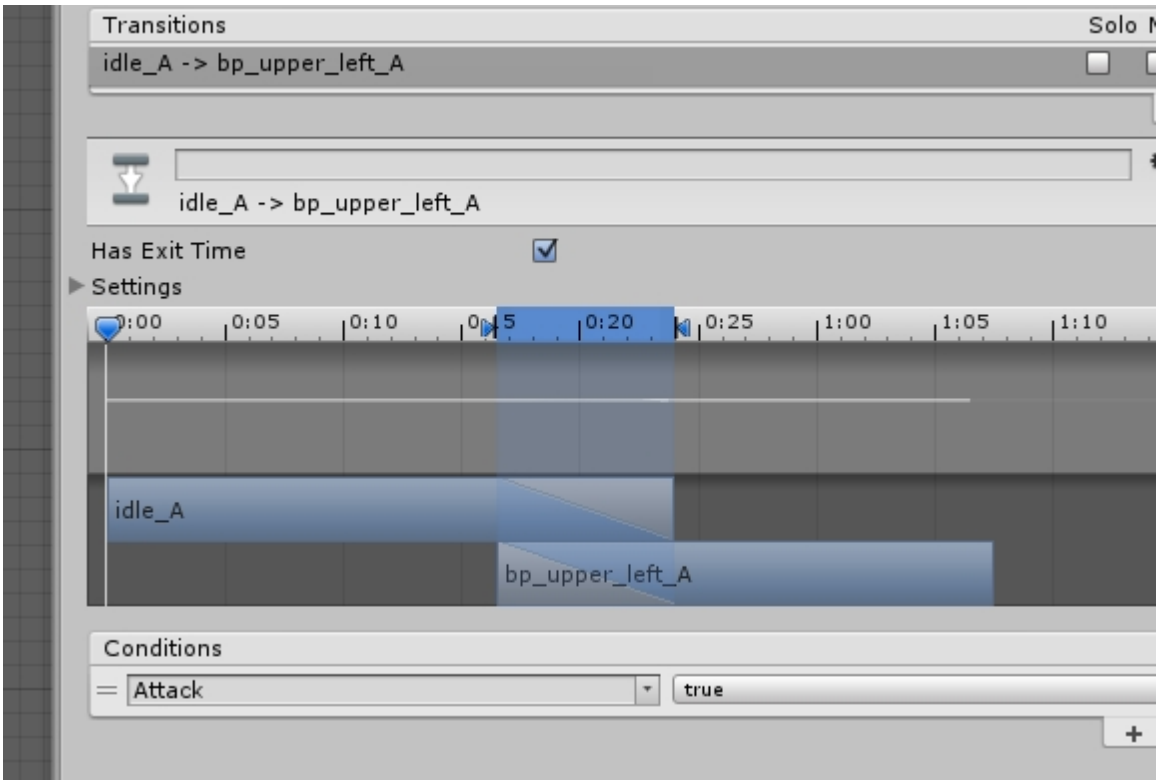
Float and Int allows us to enter a numeric value. This type of parameter would be good for a speed, index, multiplier, or any other where a numeric value would make sense.

A Bool is either true or false, until something changes it, while a Trigger allows a named value to enable / disabled, in a single fire, and turns itself off

For starting off, select the Bool Parameter, and give it the name "Attack".



On the Transition line from Idle_A to bp_upper_left_A, click on it to bring up its Inspector window. Under Conditions, click on the Plus. A new Condition is added, and since we only have one Parameter, it sets it to that.



Press the Play button. You'll notice that the Butcher is idling in the Game window. In the Animator window, click on the Attack. Because the parameter is type "Bool", it stays either true or false until we change it.

Since our transition between Idle_A to bp_upper_left_A relied on the condition of Attack to be True, the Animator State finishes the Idle_A animation, then moves up the Transition line to bp_upper_left_A. We don't have a condition on the return Transition, so once the bp_upper_left_A animation finishes, it goes back to Idle_A.

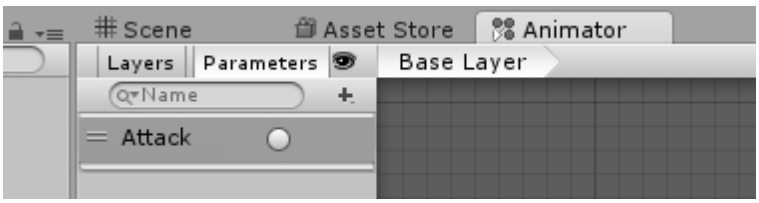
The Bool Parameter Attack is either true or false, and won't change unless we make it change to false. Click on the Attack trigger again unchecks it, which changes the condition of the Transition. Once the animations are all complete, it will settle back to the idle_A animation and no longer transition to the bp_upper_left_A.

For an Attack animation, the bool doesn't make a lot of sense. If we were to use a bool, we'd be responsible for turning it on and off to stop the attack animation.

Rather than using a Bool, let's try a Trigger Parameter.

Stop the animation by click on the Play button again, and click on the Attack Parameter. Right click on it, and click on Delete.

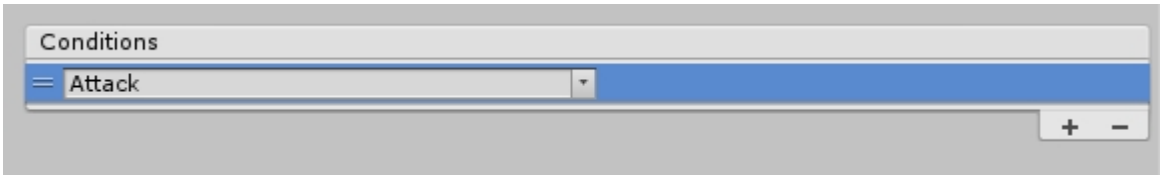
Press the Plus button again, and set a Trigger type Parameter, naming it "Attack".



The Attack parameter now has a round circle, rather than a checkbox.

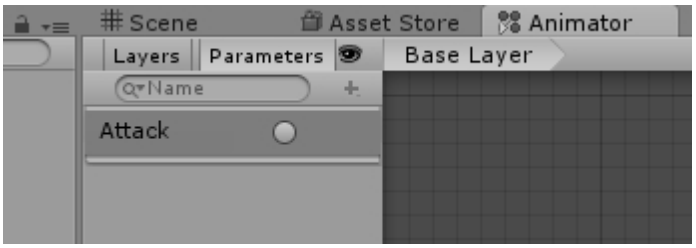
Click on the Transition line from Idle_A to bp_upper_left_A, and adjust the Condition. Since we named it the same as our our Condition, it automatically took effect.

If it did not, click on the two lines in front of the Parameter name to highlight the line, and press the Minus button.



Then re-add the Parameter using the Plus sign.

Press the play button again. The Butcher will start in his Idle animation and stay idling. Click on the circle by the Attack Parameter to fire off the event.



The Butcher finishes his Idle animation, then transitions to bp_upper_left_A, then back to Idle. However, since we using a Trigger, it's a one time cycle. The Attack trigger only fires off once, then turns itself off. This is unlike the bool, which stayed either True or False until we changed it.

This is a more natural flow for an attack animation. Fire off, and wait for the next trigger.

However, it's not ideal yet. The Idle_A animation wants to finish before firing off the transition to the next state. When we are calling a Trigger for Attack, we want the transition to begin immediately.

Click on the Transition line from Idle_A to bp_upper_left_A. Remember that Has Exit Time setting? That tells the State that in order for it to Transition, the Exit Time must pass. The Exit Time is the length of the animation itself. Uncheck the Has Exit Time, and re-play the Animation and fire off the Attack Trigger.

You'll notice that now when you click on the Attack trigger, the Transition happens immediately, firing off the bp_upper_left_A trigger, interrupting the Idle_A state.

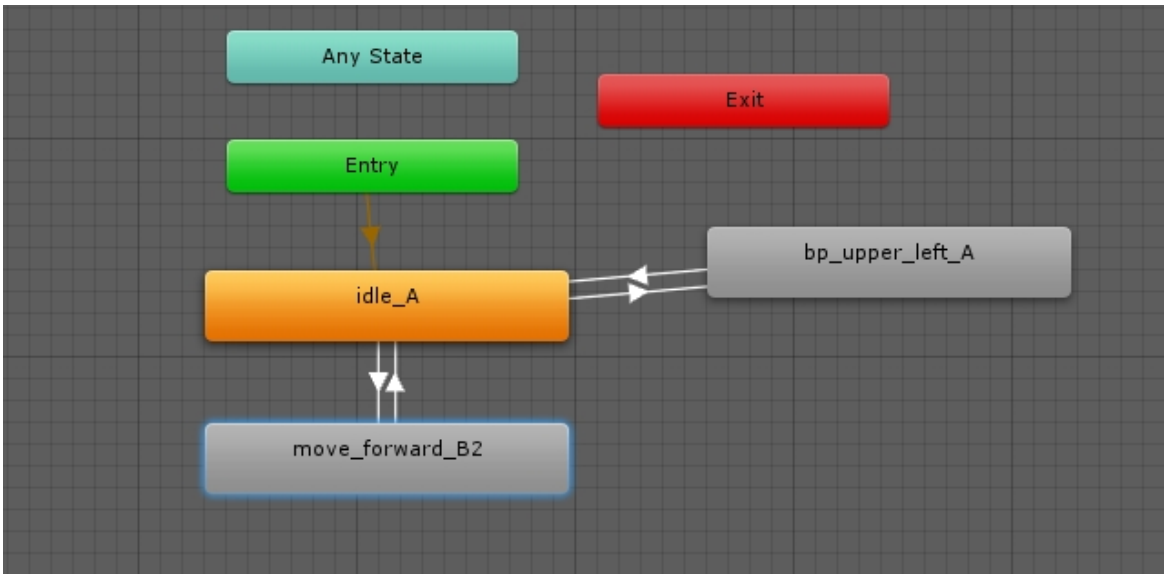
Adding a Movement State

In the previous section, we added an idle and punching animation, and discovered that a Trigger Parameter was the best option to make it fire.

In this section, we will add a walking and running animation, and using a different parameter type.

In the Project view, look under the FightingMotionsVolume1\FBX\nomove folder. We want to use the NoMove animations for this, since we haven't enabled Root Motion yet.

Select move_forward_B2, and drag it into the Animator window. Make a Transition from Idle_A to move_forward_B2, and make another one to return.



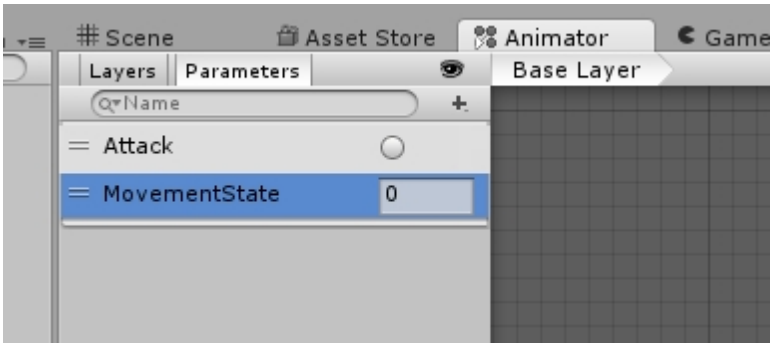
If we press the Play button now, we'll go from Entry to idle_A, to move_forward_B2, and back. We haven't set an Transition triggers yet, or a parameter, so it continually flows.

Let's decide what type of Parameter we think we need.

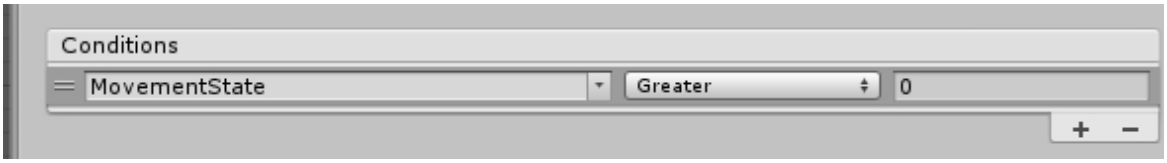
We covered a Bool and Trigger in the previous section: A Bool is either True or False until we decide it's not, and that a Trigger is a one time event.

Firing off a Trigger every time we want to take a step seems excessive. We could set a Bool like "IsMoving". If it's True, the character will continue to move until we tell it not too. This makes sense if the character can just walk, but what if want it to run as well? If an entity is running, it would be IsMoving true as well.

Let's try something else.Select a Parameter Integer, and call it "MovementState".



MovementState will default to 0. By default, our animation should just idle, so from now on, MovementState 0 will be considered an Idle state. Since 0 is not moving, let's say that 1 is moving. Set a Condition on the Transition line from Idle_A to move_forward_B2. But default, it'll show the first Parameter we made, which is Attack. Click on the drop down and select the new Parameter, MovementState.



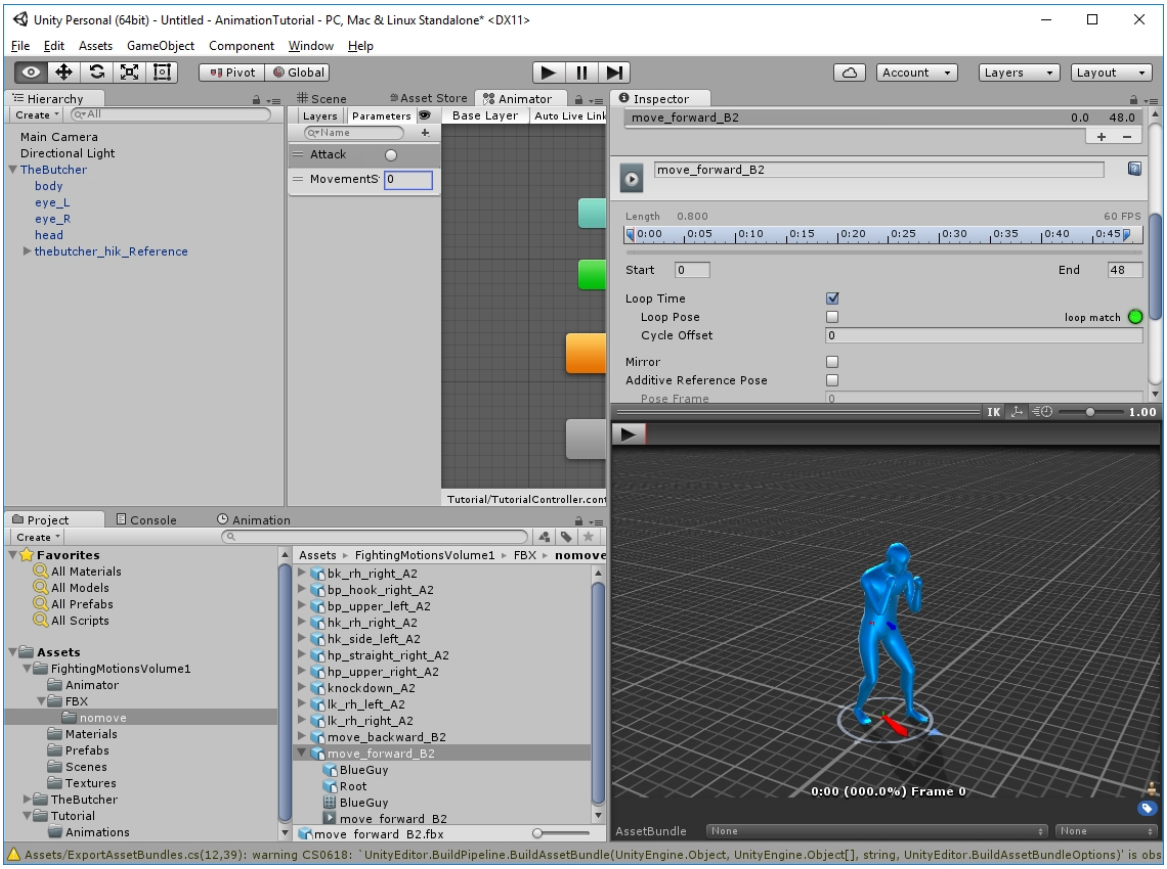
By default, the MovementState is set to Greater than 0. If the MovementState is greater than 0, the character will start walking.

Press Play button and see what happens. The Butcher Idles peacefully. In the Parameters section, change the MovementState to 1. The Butcher will start to walk, then Idle. The MovementState will stay to the value you entered, 1, until you change it again. Without a condition on the return transition, the Butcher will Idle, walk one step, then idle, etc

Stop the Animation and set a Condition on the return Transition between move_forward_B2 and idle_A. This time, set the Condition to MovementState is Less than 1. This will stop the State from Transitioning back to Idle_A if the MovementState is 1 or higher.

Press Play again, and see how the Idle_A state runs, then see the transition moves to move_forward_B2 fires. However, the walking animation doesn't loop. This is an Animation setting, rather than an Animator.

Through the Project view, go back to the Animation under FightingMotionsVolume1/FBX/nomove, and select the move_forward_B2 animation again. In the Inspector window, click on the Animations tab, and scroll down to "Loop Time". By default, the Loop Time is unchecked for this particular Animation. Click on the check mark to enable it, and click on Apply to save your changes. The Apply button is available in the Inspector window, although you may have to scroll down to see it.

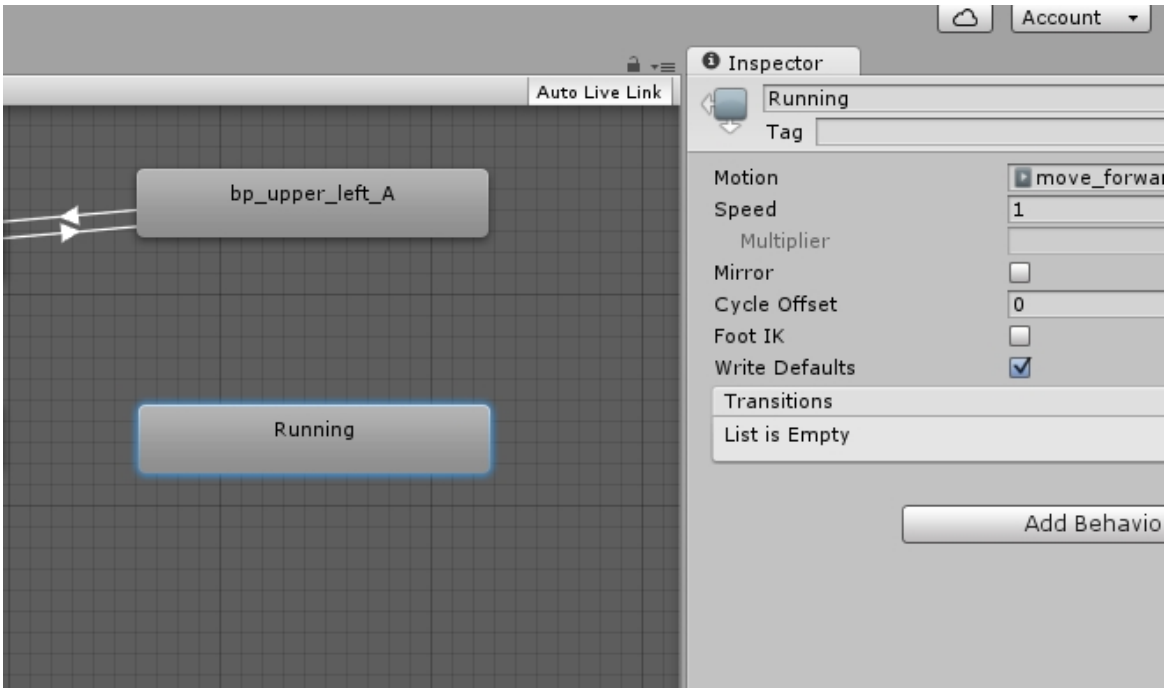


Now re-Play the sequence, triggering the MovementState again by changing 0 to 1. The Butcher will do an awkward shuffle.

Let's continue with the movement state, and add a Running animation. The FightingMotionsVolume1 does not contain an actual running animation, but we can improvise. We'll duplicate the move_forward_B2 State, and increase it's speed.

Right click on the move_forward_B2 state, and select Copy, or click once on the State, and press Ctrl-D to

duplicate it. You can also find the original animation, and re-drag it up into the Animator window. This will create a new State, with the same name, but with a number at the end. Click on the new State to bring up the Inspector window, and rename the State to Running.



This Animation is a walking animation, but we can adjust the speed value to make it look like it's animating faster. Adjust the Speed from 1 to 2. The Speed setting on the State does not change the Animation Clip. This allows us to re-use the same animation clip in multiple states, but we can have them running at different speeds.

Create your Transition lines from move_forward_B2 to Running state, and back again.The Transition line from move_forward_B2 should have a new Condition set, where MovementState is Greater than 1. The return Transition should have a Condition that MovementState is Less Than 2.

Re-run the Animation by pressing the Play button, and adjusting the MovementState from 0, to 1, to 2, then 1, and 0 again, and watch the States transition.

Each State needs to finish an Animation sequence before acknowledging the new Transition Condition. However, like we did for the bp_upper_left_A transition, we can select each Transition line, and de-select Has Exit Time. This allows the current State to check its Transition Condition before the end of its Animation, allowing the next state to fire at any time.

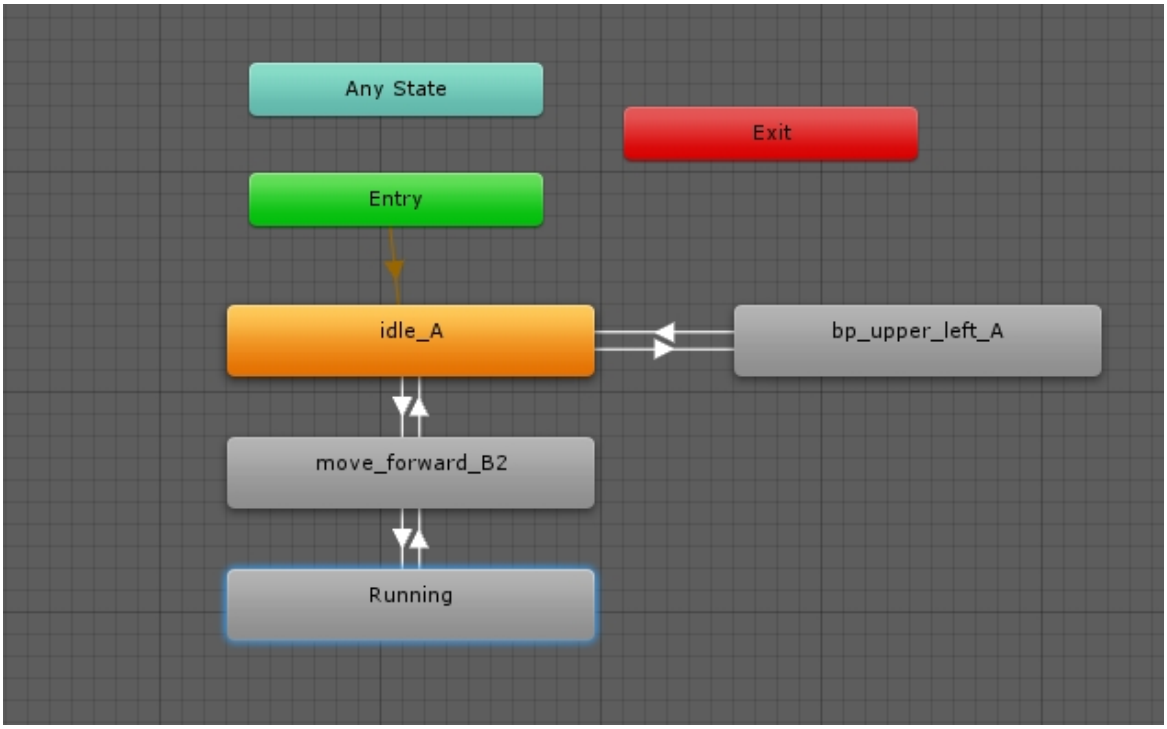
Remove the "Has Exit Time" on each of the Transition lines going from Idle to move_forward_B2, to Running, and the ones returning to those states.

This allows each State to have it's Transition interruption by the next State. For example, if we are running (MovementState 2), and suddenly we want to stop (MovementState 0), we don't want the Running animation to continue until it's done. We are stopped, so we want the animation to be interrupted.

If you start with MovementState 0 and change it immediately to 2, you'll see the Transition from Idle to move_forward_B2, then moves unto the Running State. Why? Our Transition from Idle to move_forward_B2 had a condition that MovementState is Greater than 0. As move_forward_B2 starts, it finds that the Transition Condition to Running is satisfied, as the movementState is Greater than 1.

Setting up that type of State Transition allows a more natural flow of the animation. Rather than going from a dead stand still to a full out run, the entity will be idle, take a step or two, then let's it break into a run.

At this point, your Animator should look similar to this:

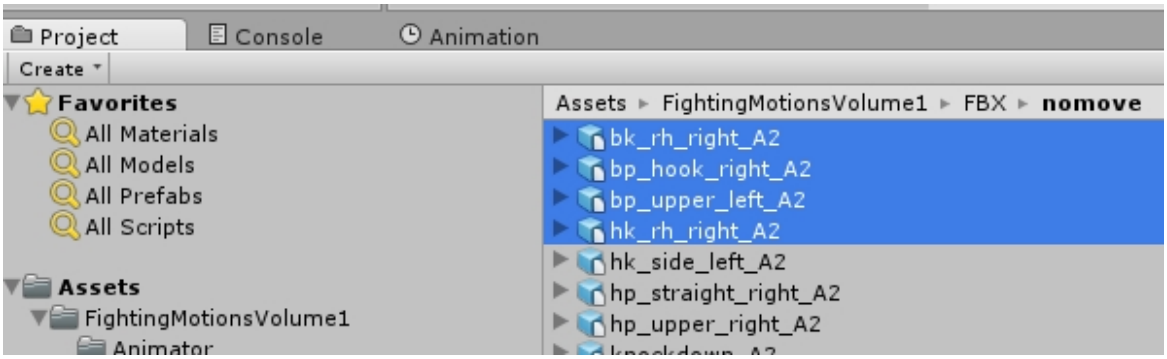


Adding a Variety of Animations for a single Trigger

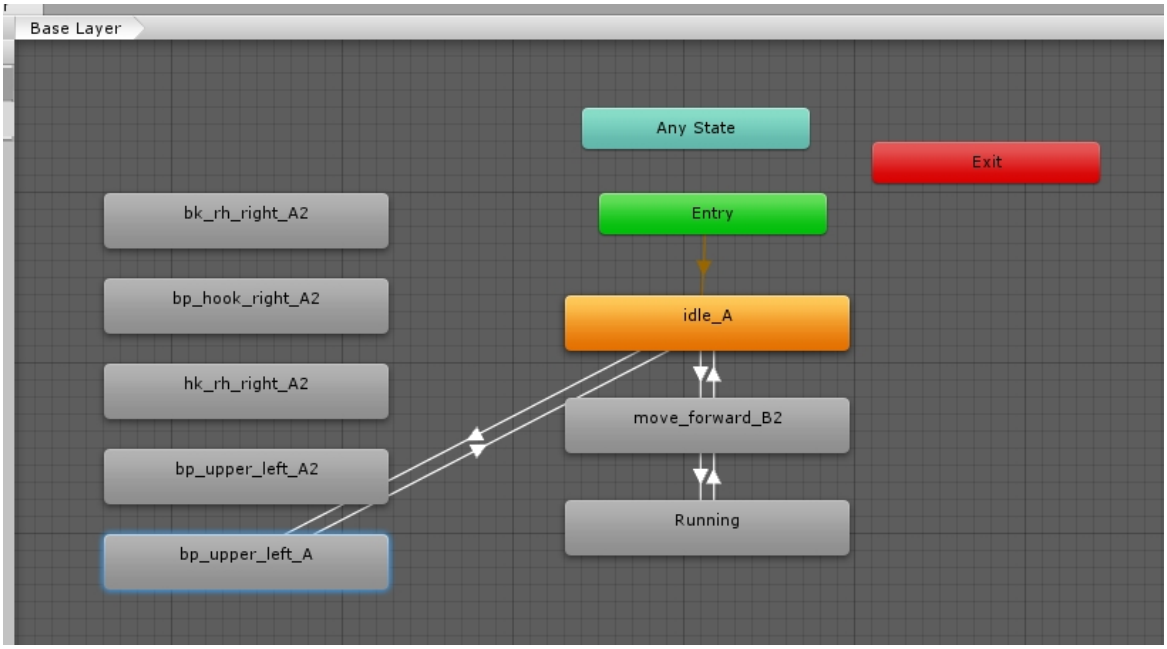
In our previous step, we created a walking and running animation sequence, and we added a punching animation, which transitions off of Idle.

For this next step, we are going to add multiple Attack animations, and allow them to be triggered at any time.

Let's add more Animation Clips to the Animator window. Under the FightingMotionsVolume1/FBX/nomove folder, drag the following animations to the Animator window:



Separate them out in the Animator window so they can be seen clearly, and move our old bp_upper_left_A to be with the new attack animations.

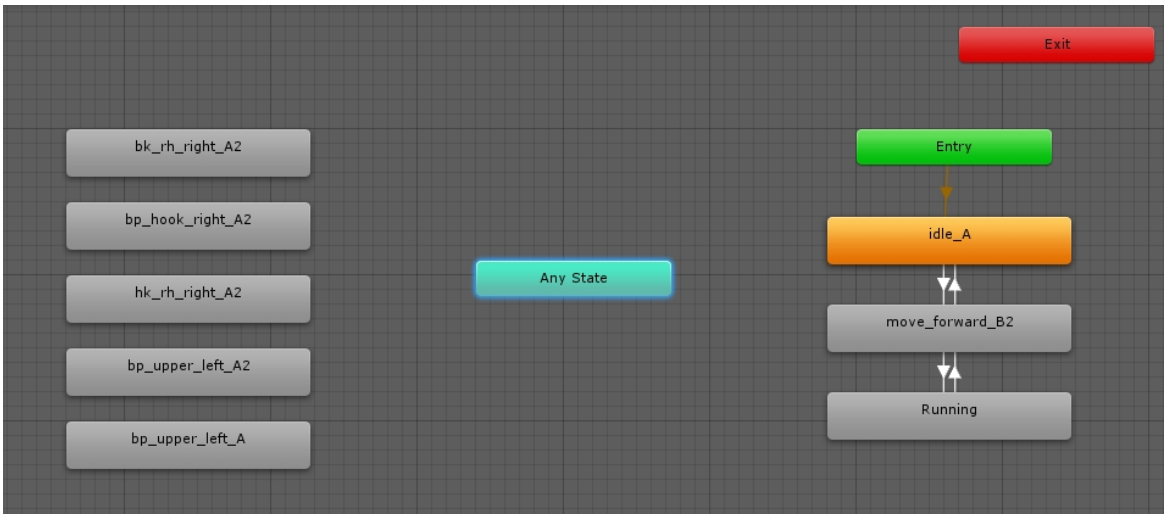


Before, we transitioned from Idle_A into bp_upper_left_A. However, that limited us to only triggering a punch during the idle state. Meaning, if our entity was running, it would have to go from Running, to move_forward_B2, and to Idle_A, then trigger the punching animation. Let's fix that.

Click on the Transition lines from Idle_A to bp_upper_left_A, and delete the Transition lines by pressing the Delete key.

Going forward, we want to be able to Transition our attack animations from any state. So rather than making transitions from every state, we are going to use the special "Any State" node. The Any State is an Animator shortcut that behaves like any other individual States, but acts like a wild card. As such, you cannot do a return transition to the Any State. Instead, we'll want to do a return Transition to our Idle_A. This implies that after we attack, we want to go back to our Idle state.

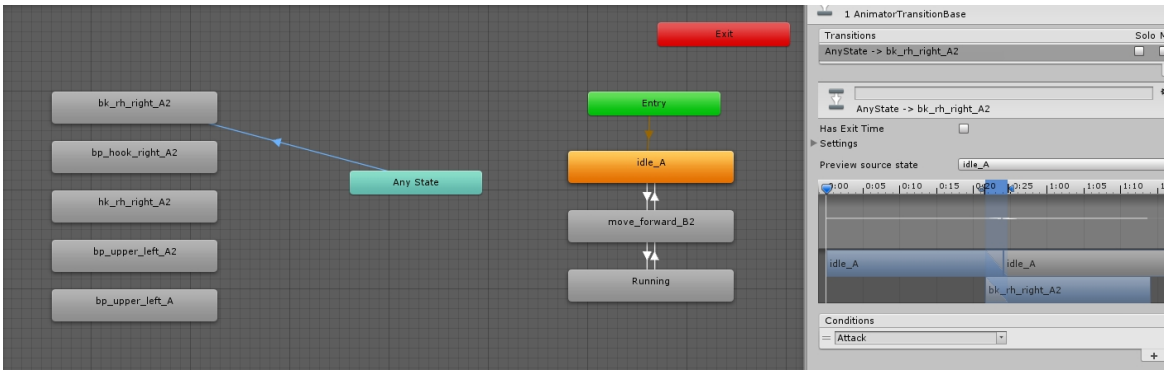
Re-Arrange your States so you have more room to allow connections between the Any State and attack animation States.



Make Transition lines from Any State to bk_rh_right_A2.

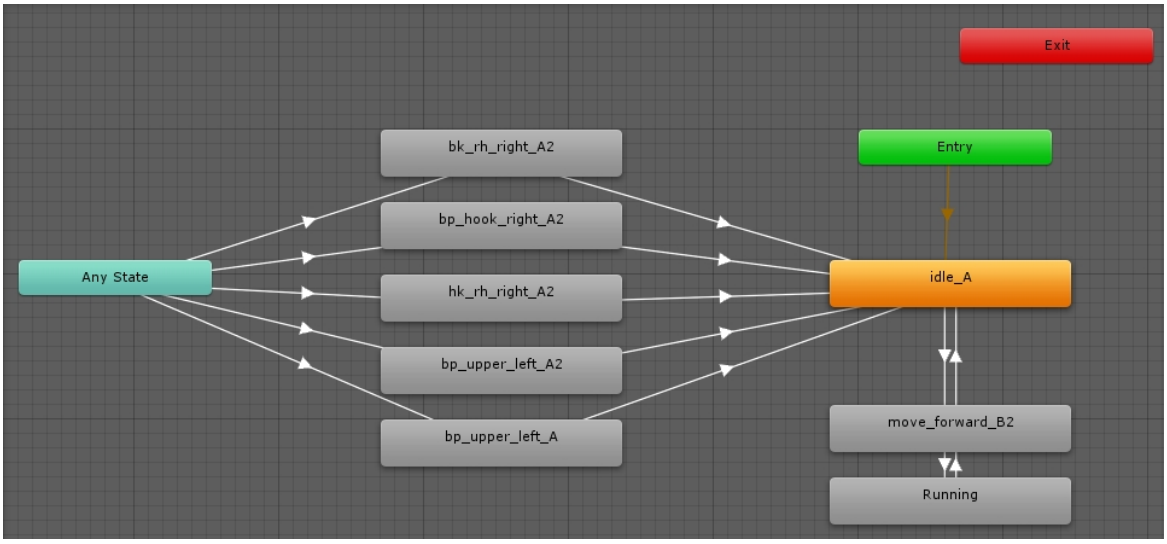
For the Transition Condition, add the Attack Trigger Parameter as a condition.

Since this is an Attack animation, and we want to be able to interrupt the current animation to fire off the attack, we also want to uncheck the Has Exit Time, since we want the attack animation to trigger immediately.



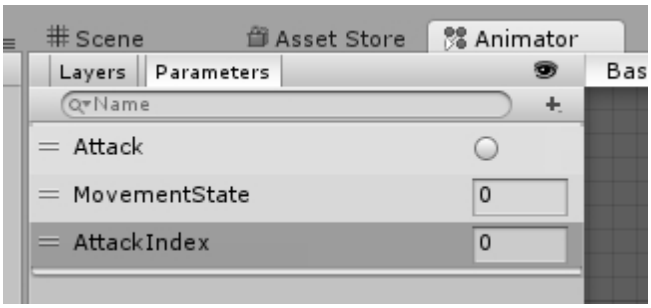
Proceed to make a Transition line from Any State, to each of the other attack States, setting the same Condition of Attack Trigger, and unchecking the Has Exit Time.

For our return Transition, we want to go directly to the Idle_A state. Re-arrange the states so that it looks clearer, as we did below.



Play the Animation again. When you click on the Attack Parameter to fire off the event, you'll notice that it only fires the first condition that it meets. In the above example, that bk_rh_right_A2, since it was added first. The other attacks are ignored. So how can we get the other Attack animations to fire?

Stop the Animation, and add a new Int Parameter, and call it AttackIndex.

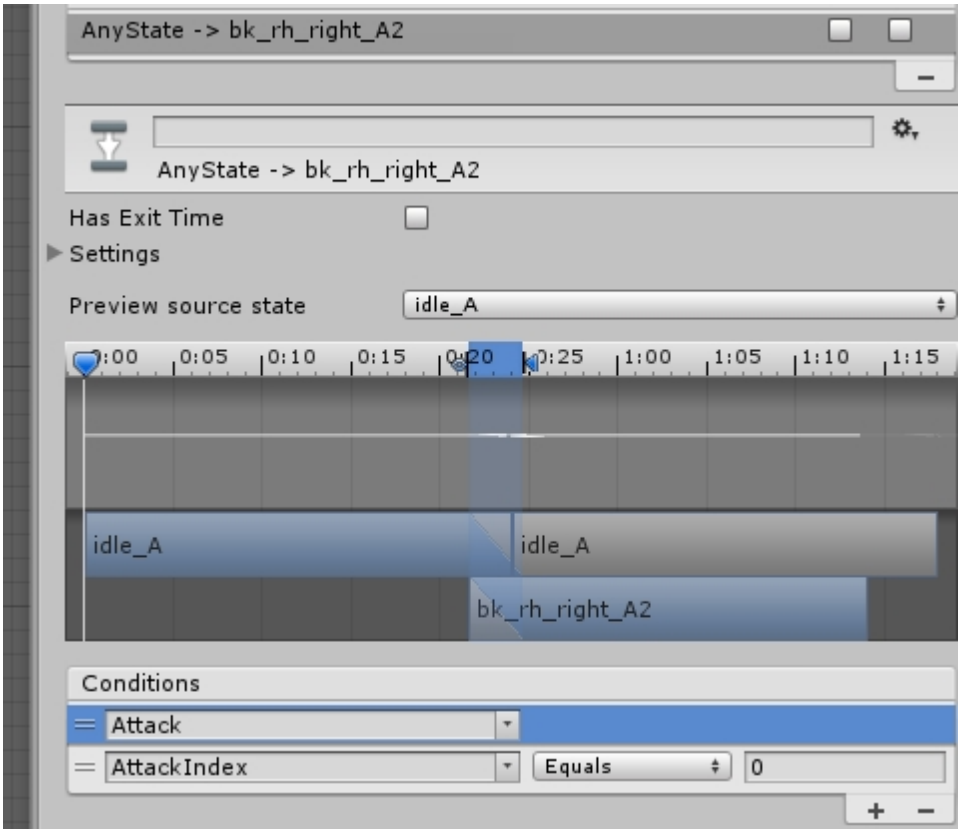


Much like we did for the MovementState, and decided that 0 was idle, 1 was for walking, and 2 was for running, we'll give each of our Attacks an index value. Our first attack animation will be AttackIndex 0, the

second attack Animation will be 1, etc.

For each Transition line from Any State to the Attack States, add a new Condition on AttackIndex Equals with our numbering system. For example:

bk_rh_right_A2 will have AttackIndex Equals 0
bp_hook_right_A2 will have AttackIndex Equals 1
hk_rh_right_A2 will have AttackIndex Equals 2
etc



Once all the Transitions from Any State to Attack States have a unique AttackIndex, press the Play button again. The Butcher will idle.

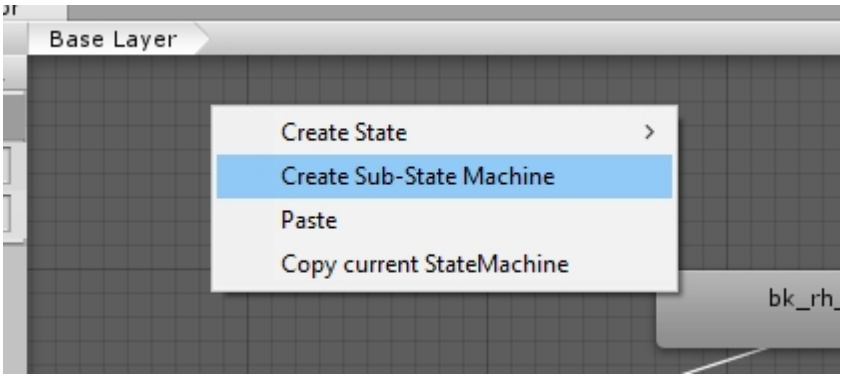
In the Parameters section, change the AttackIndex to 2, and click on the Attack Trigger. The State will interrupt the Idle_A animation, transition from Any State to the appropriate Attack Animation, then back to Idle after completing. Since the Attack Trigger only fires once, the Butcher will continue to idle after his attack. Change the AttackIndex to different values, pressing the Attack Trigger after each one.

Adding a Sub-State Machine

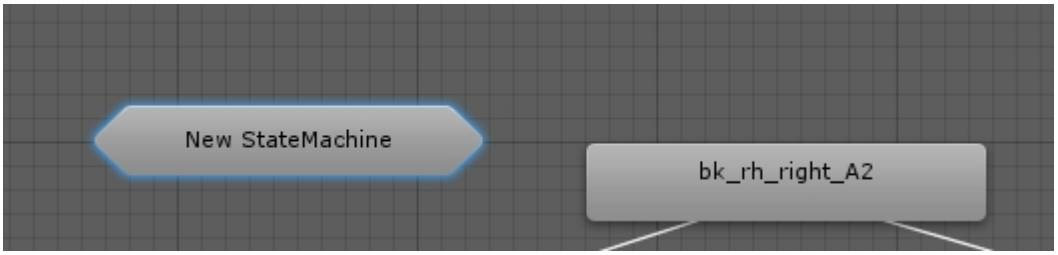
In our previous exercise, we added 5 new Attack Animators, each with two Conditions: Attack Trigger Parameter, and the AttackIndex Integer Parameter.

As you can see, the Animator Window is getting a bit cluttered. To Clean it up a bit, we can create a Sub-State Machine to group them together. A Sub-State machine does not change any of the Transitions or States,but rather just cleans up the UI.

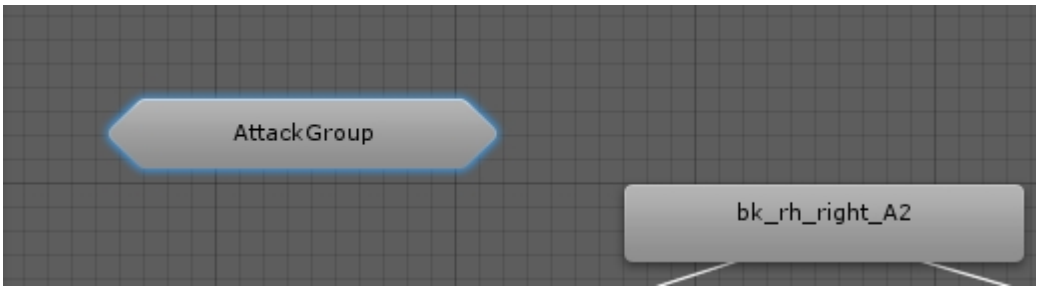
In the Animator window, right click on the empty background, and choose Create Sub-State Machine.



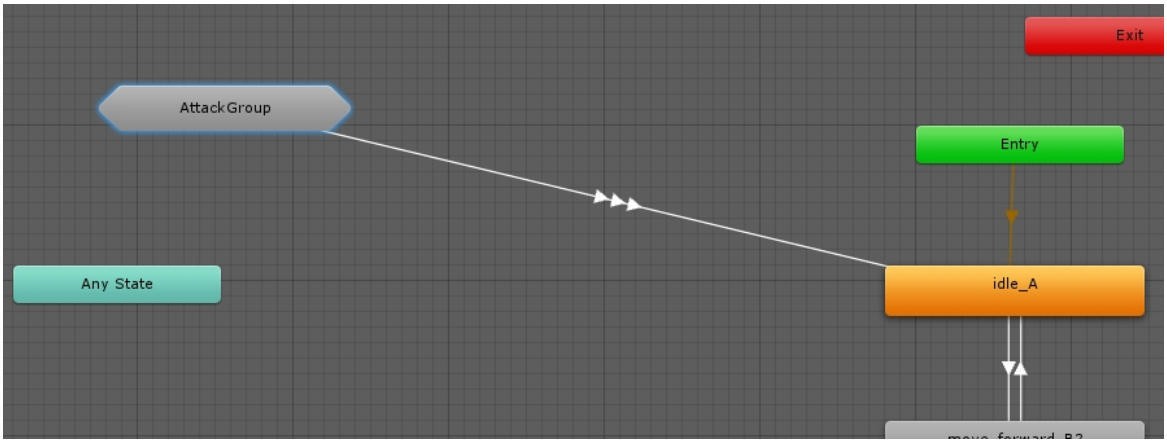
A "New StateMachine" state appears:



However, it's a different shape; it's a hexagon, rather than box. That's an indicator that it's a Sub-State machine, and may contain more states. Click on it once to open up it's Inspector window, and change the name. We'll call it the AttackGroup.

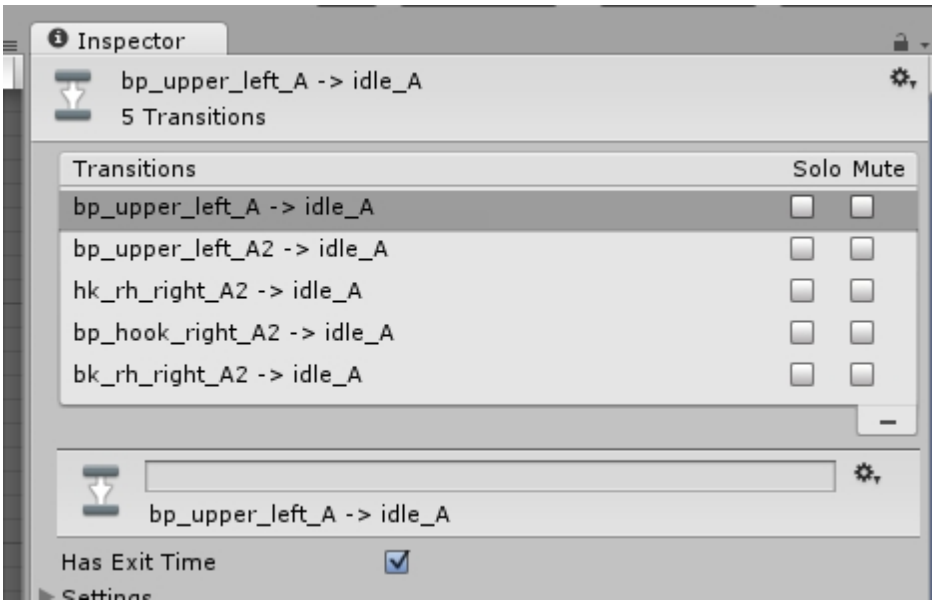


Multi-select your Attack States, and drag them over the AttackGroup. You'll notice your mouse cursor will have a little + added to it when it's over the Sub-State machine.



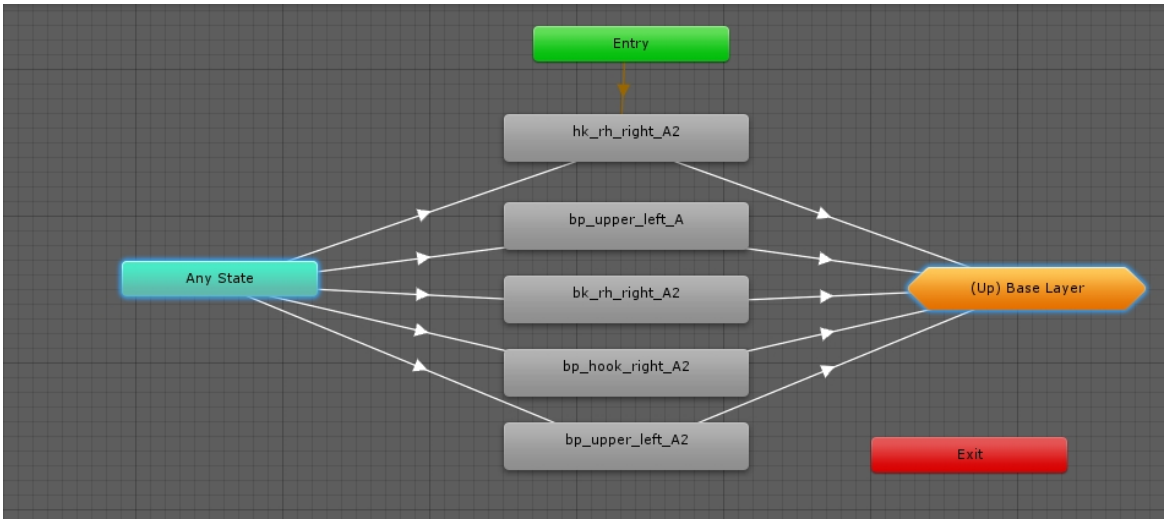
The Attack Sates will disappear, the links from Any State have disappeared, and a new Transition line, with three arrows, point from the AttackGroup Sub-State Machine, to Idle_A.

The Transition line with 3 arrows indicates that there's multiple Transition states coming out of the AttackGroup. Click on the Transition line to bring up the Inspector Window.



Under the Transitions, we see there's multiple transitions now. It shows which State goes to which State. All our links are still intact to the Idle_A, along with any of the conditions we have set on them. Our link from the Any State has disappeared as well. However, a Sub-State Machine itself has an Any State, so there's no need to visualize the link here.

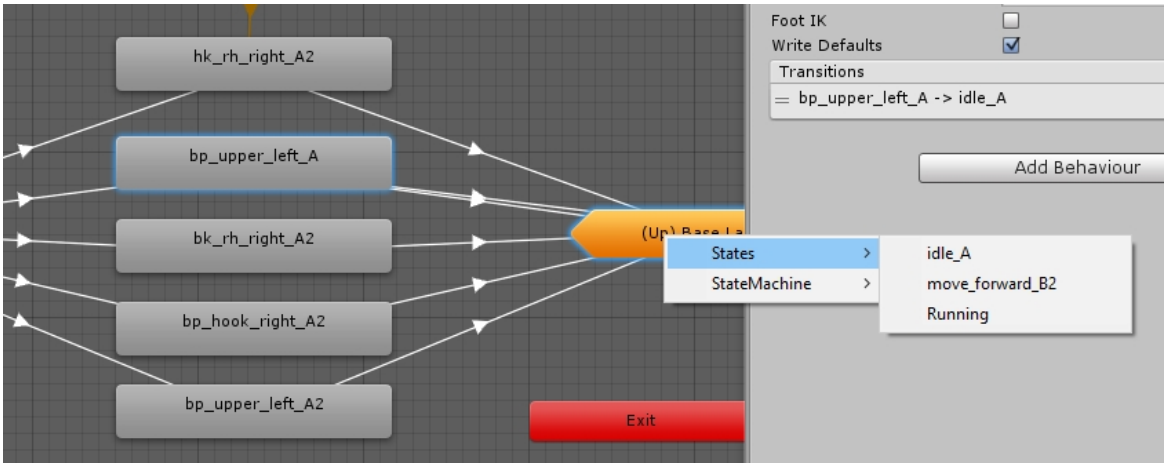
Double click on the AttackGroup to access the States within it. Initially, it'll be a bit of a mess. However, you can re-arrange the states to make them easier to see and read.



The Sub-State Machine has an Entry, Any State, and an Exit, like the top level State Machine. However, it has a new State: (Up) Base Layer. This is the link outside of the Sub-State Machine, and it's what connects our AttackGroup to our Idle_A.

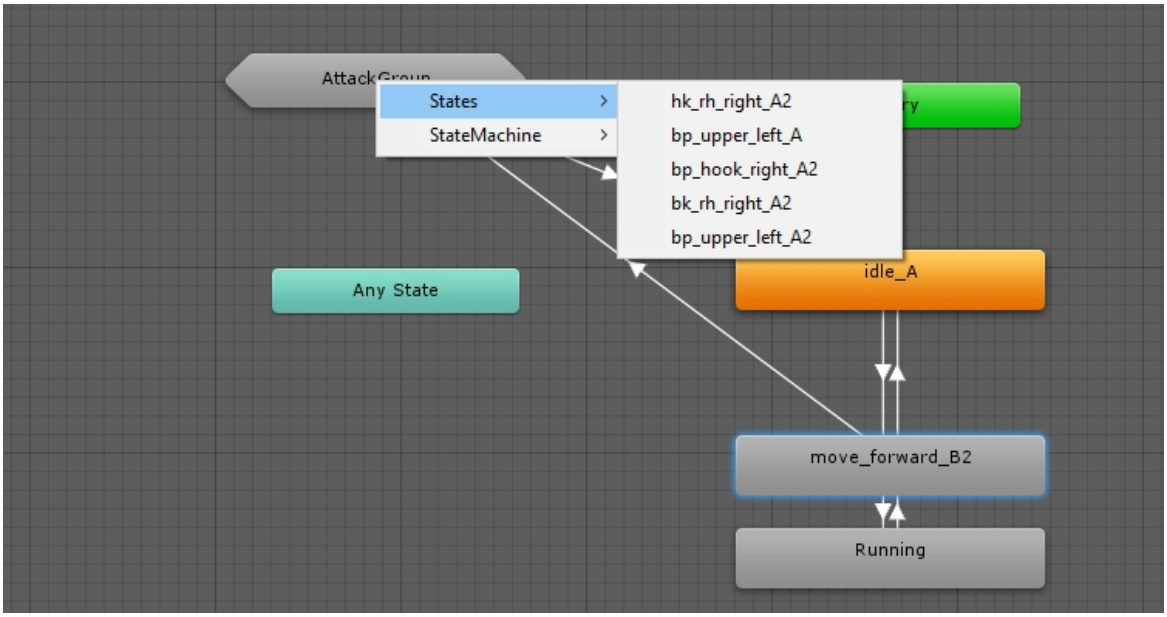
Making Links Outside of the Sub-State Machine

If you need to add a new Attack Animation to the sub-state, and need to Transition back to the main Idle_A, drag your Transition to the (Up) Base Layer. A small pop up will show you the available States to connect too.



Making Links Inside of the Sub-State Machine

Likewise, if you need to make a transition from the main State Machine, into a Sub-State machine, make a Transition to the AttackGroup state machine, and a similar pop up appears.

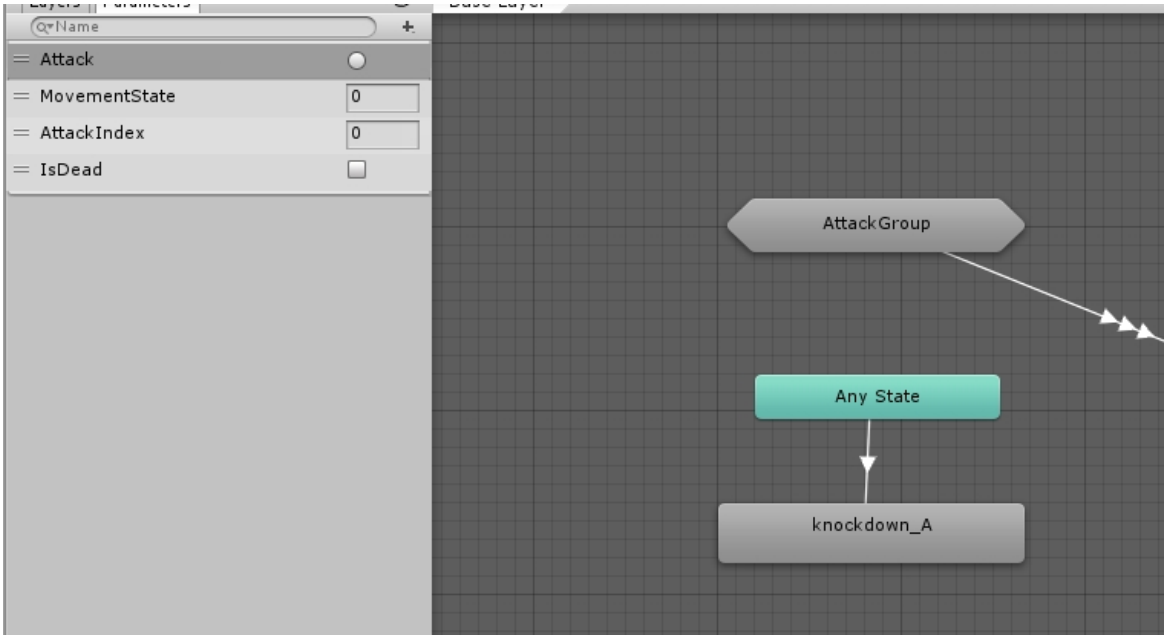


Adding a Death Animation

A Death animation is a different type of animation. It needs to be transitioned to once, and not return to any other transition.

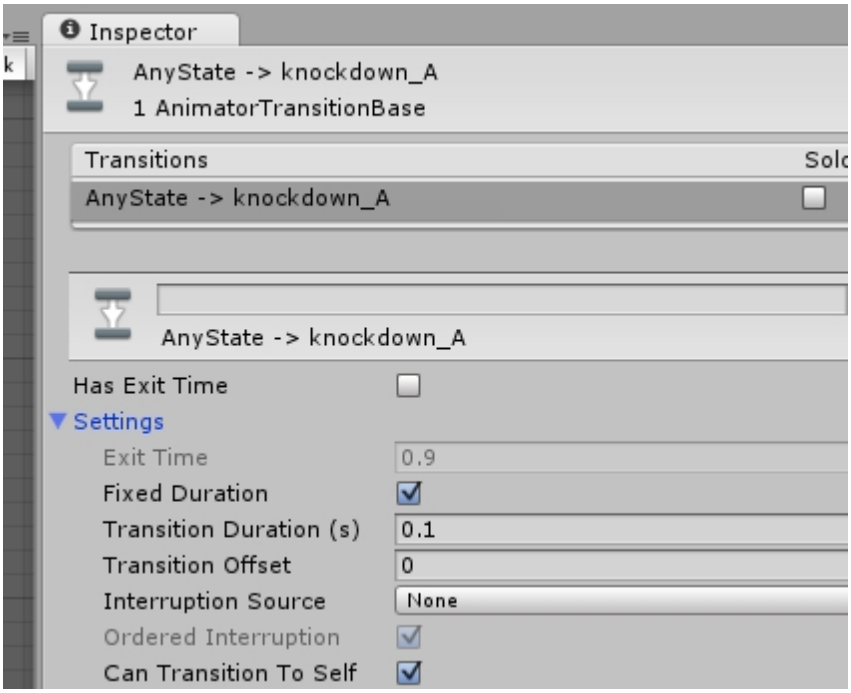
Under the FightingMotionsVolume1/FBX folder, find the "knockdown_A" animation, and drag it into the Animator window. Create a Transition from Any State to the new knockdown_A state.

Under the Parameters, add a new Bool Parameter called "IsDead". Create a Transition from Any State to the new knockdown_A state, setting the Transition Condition to IsDead true.



Now Play the Animation, and and click on the IsDead Parameter. The Butcher looks like he's going into a seizure. That's because the IsDead condition is true, and stays triggered. The Any State is told to Transition to knockdown_A if that's true. This creates a loop in the Animator State Machine, where knockdown_A is calling itself over and over again.

Stop the Animation, and click on the Transition line. Under the Settings option, uncheck Can Transition to Self. This disables the Transition from triggering the same State.



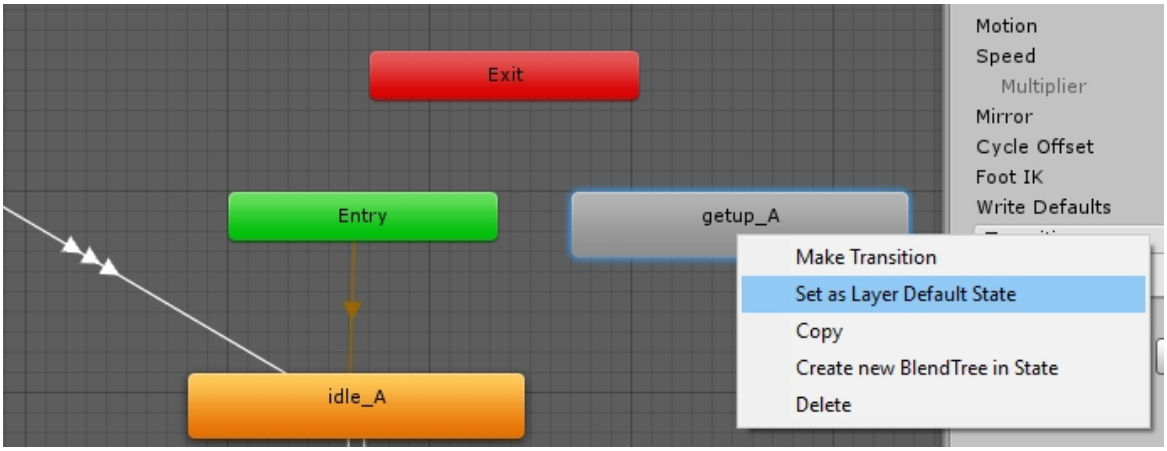
Re-Play the Animation, and click on the IsDead. The Butcher falls down, and stays down.

Adding an Alive Animation

For more immersion, we can start the Butcher's initial animation from laying down, to standing, as if he was just waking up.

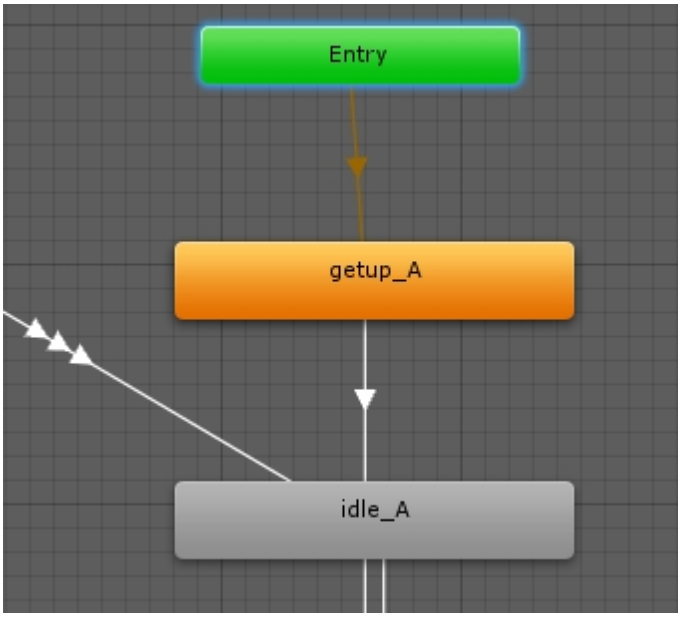
Under the FightingMotionsVolume1/FBX, look for the getup_A animation, and drag up into the Animator Window.

Right click on the getup_A State, and select Set Layer as Default State



Add a new Bool Parameter called IsAlive.

Then, add a Transition from getup_A to idle_A, with a Transition Condition on IsAlive is true.

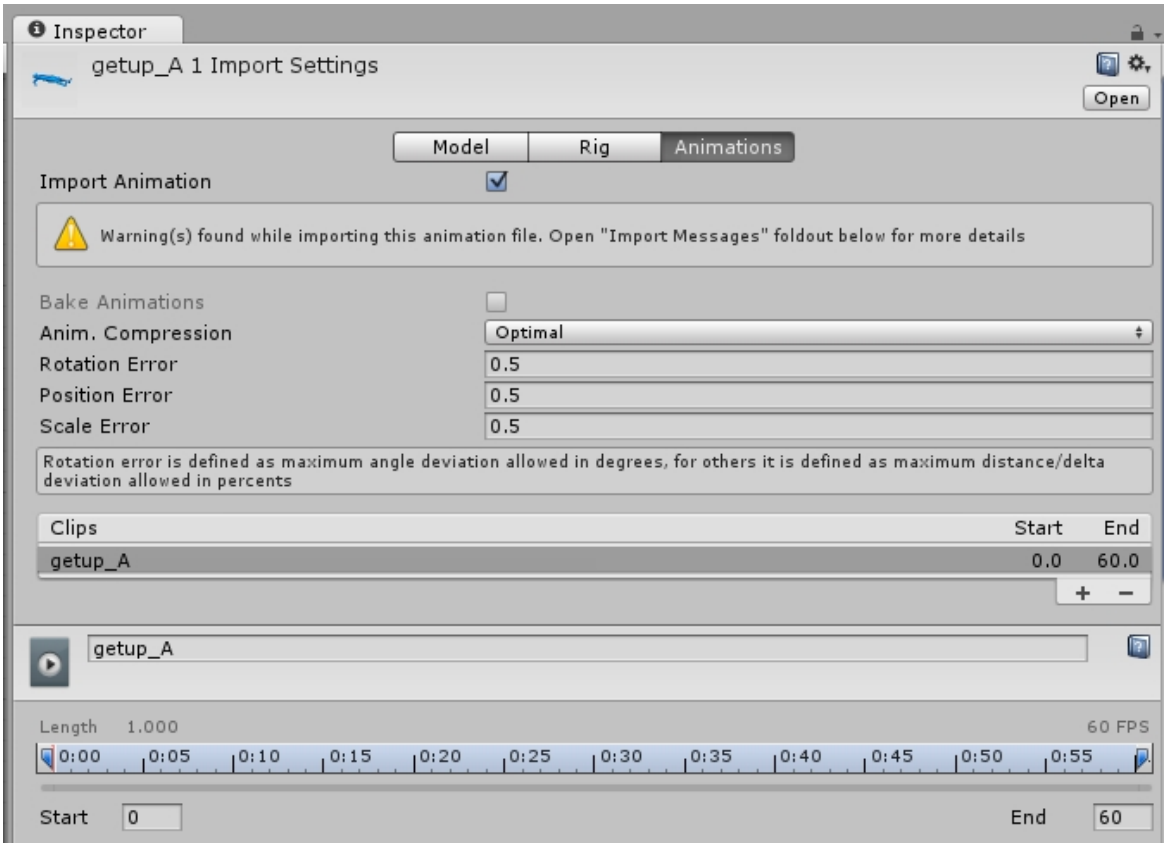


Play the Animation. The Butcher stands up immediately.

That's because the Entry state is now pointing to the getup_A state. We cannot set a transition on the Entry State to the next State, so it automatically fires the getup_A State. How can we make the Butcher stay

laying down?

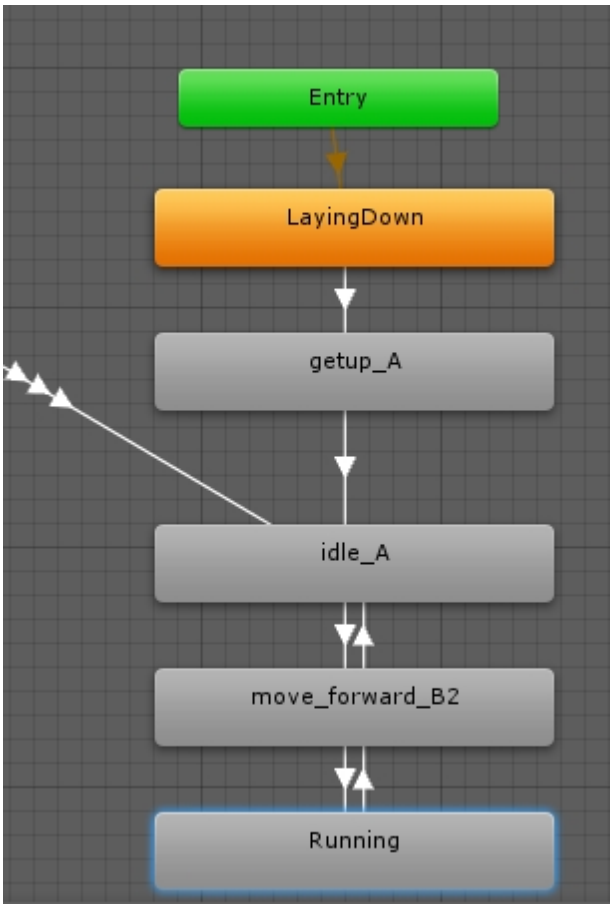
Find the getup_A animation in the Project Window, and select it. Press Ctrl-D to duplicate it, which gives you "getup_A 1". Select the new Animation "getup_A 1", and open its Inspector window



By looking at the Animation clip, we see it Start 0, and Ends at 60. Adjust the End value to 0.1, so the entire animation loops around the first frame, which is of the Butcher laying down. Click on the Loop Time, and Loop Pose as well. Click on Apply to save the changes.

Drag the new "getup_A 1" animation into the Animator window, and rename LayingDown.

Right click on the new state and select "Set Layer as Default State", and set a Transition from it to getUp_A, with the Transition Condition of IsAlive.



Now rePlay the Animation. the Butcher moves into the laying down state, and continues to lay down until the IsAlive Parameter is clicked, then he stands up, ready to fight.

The Mecanim SDX Class Parameter List

In order for your Animator State Machine to work in-game, the following Parameters are triggered from the Mecanim Class.

Required Parameters

These parameters are required for a basic entity.

Parameter Name	Parameter Type	Description
MovementState	INTEGER	0 for Idle, 1 for Walk, and 2 for Running
Speed	FLOAT	Controls the speed of the entity for Root Motion
Attack	TRIGGER	Triggers the Attack animation
Jump	TRIGGER	Triggers the Jump Animation
IsAlive	BOOL	Triggers the Wakeup Animation
IsMoving	BOOL	True if entity is moving
IsDead	BOOL	Triggers the Death Animation

The following Parameters is what the full Mecanim Class allows. If a trigger or value is not available, the mecanim class will fail the call quietly, and continue.

BOOL

The Animator State Machine uses bool for values that are explicitly set and unset. That is, if you set IsDead to false, it will stay false until you change it.

The following Bool Parameters are supported in the MecAnimSDX Class

Parameter Name	Parameter Type	Description
IsDead	BOOL	True if Entity is Dead, otherwise False
IsAlive	BOOL	True if the Entity is awake and alive.
IsMoving	BOOL	True if the Entity is moving
IsEating	BOOL	True if the Entity is Eating
IsCrouching	BOOL	True if the Entity is Crouching
IsStunned	BOOL	True if the Entity is Stunned
IsSleeping	BOOL	True if the Entity is Sleeping

INTEGER

The Animator State Machine uses integer for numeric values.

Indexes

For the MecAnim SDX class, we use Integer types to offer Index values. That is, they are used if you want to add multiple animations for the same type of state. For example, if we have 5 different animation attacks, we can assign a unique attack index for each Transition. Attack number 1, would have an AttackIndex of 0. Attack number 2 would have an AttackIndex of 1. This allows us to have more variety in the various types of animations than we have had before.

Parameter Name	Parameter Type	Description
AttackIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for Attack animations
SpecialAttack	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for Special Attack animations
SpecialSecondAttack	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for Special Secondary Attack animations
RagingIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for any Raging animations
ElectrocutionIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for electrocution animations
CrouchIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for crouching animations
StunIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for

		any Stunned animations
SleeperIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for Sleeper Animations
HarvestIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for any harvesting animations
PainIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for any Pain animations
DeathIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for any Death Animations
RunIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for any Running Animations
WalkIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for any Walking Animations
IdleIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for any Idle Animations
JumpIndex	INTEGER	A 0 to ... range of numbers to uniquely identify transitions for any Jump Animations

Other Flags

We use Integers for other flags as well

Parameter Name	Parameter Type	Description
MovementState	Integer	0 for Idle, 1 for Walking, and 2 for Running

FLOAT

We use Float values when we will need decimal precisions, like for forward speed, when using Root Motion

Parameter Name	Parameter Type	Description
Forward	FLOAT	The current speed of the entity
Strafe	FLOAT	The speed in which its strafing
IdleTime	FLOAT	The Idle time in the mecanim class

TRIGGER

Triggers are one-time events per call. We use triggers to let the Animator know we want to do an action once.

Parameter Name	Parameter Type	Description
Attack	TRIGGER	This triggers one of the attack

		animations
Pain	TRIGGER	This triggers the Getting Hit Animation
Alive	TRIGGER	This tells the entity to wake up
Jump	TRIGGER	This tells the entity to Jump
SpecialAttack	TRIGGER	This tells the entity to trigger a Special Attack
SpecialSecondAttack	TRIGGER	This tells the entity to trigger a Second Special Attack
Raging	TRIGGER	This tells the entity to begin Raging
Electrocution	TRIGGER	This tells the entity to begin being electrocuted
Harvest	TRIGGER	This tells the entity to trigger harvest animation
Crouch	TRIGGER	This tells the entity to begin crouching