# 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.

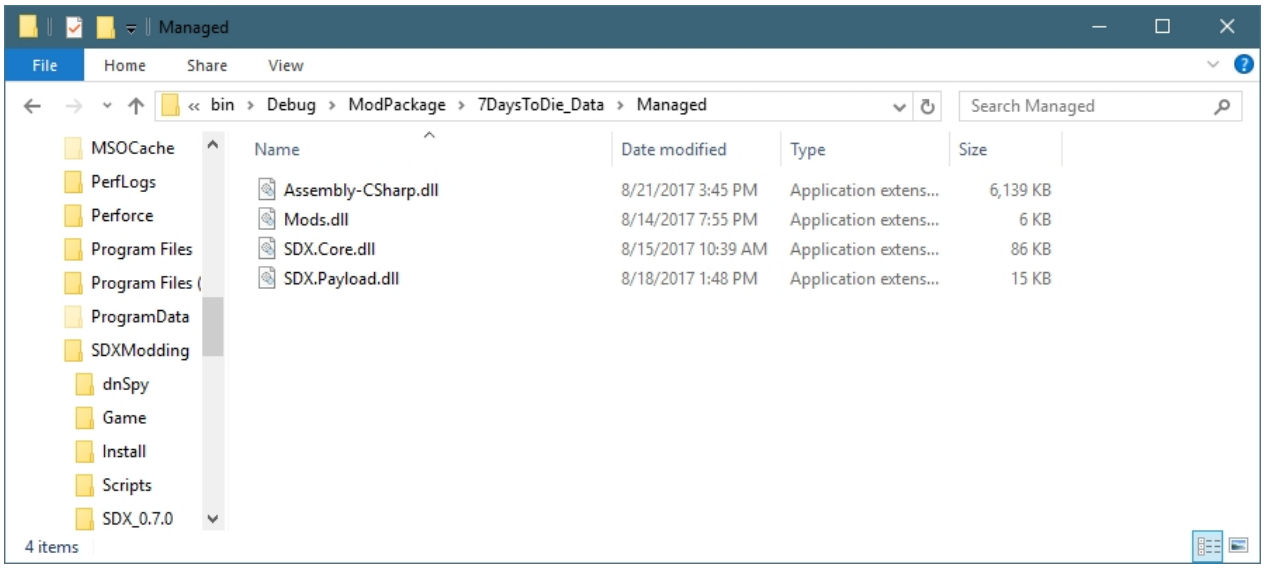
***Note: SDX is not EAC compatible. If you attempt to load up SDX with EAC enabled, you will get an error.***

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

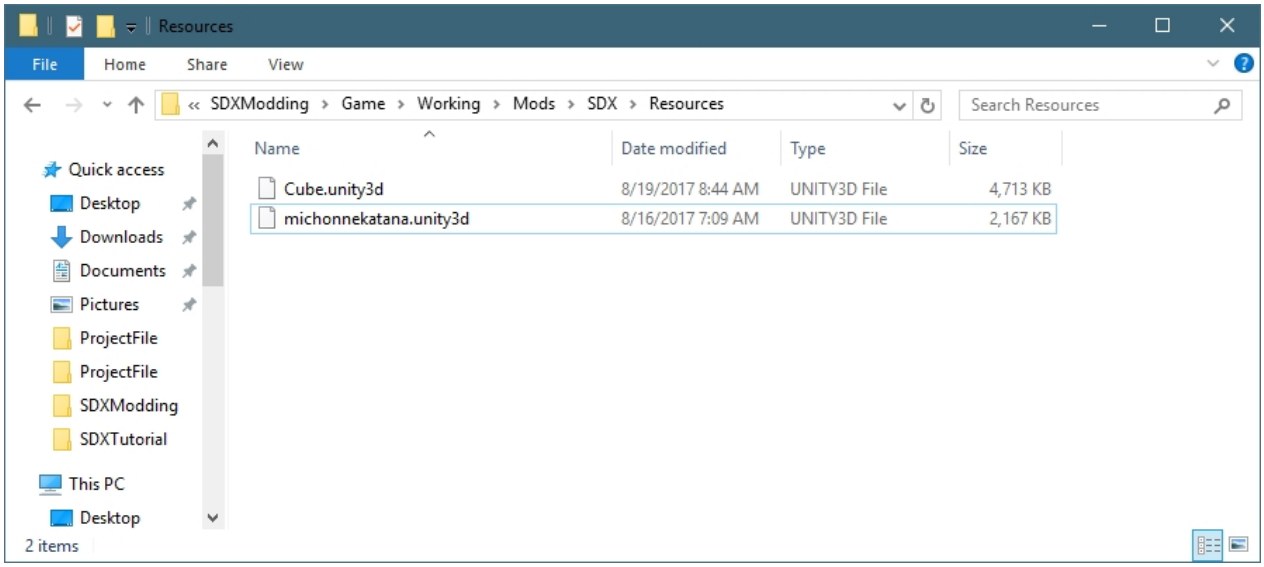
*So what is SDX modding?*

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






## 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

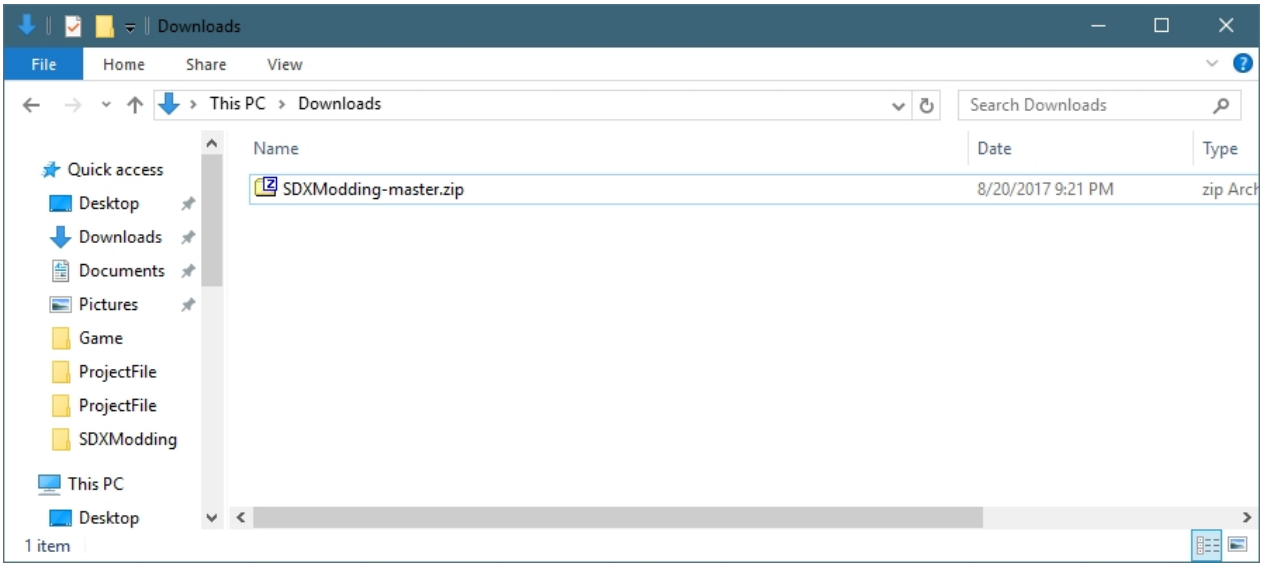
This tutorial will guide you through step-by-step on how to make your first SDX mod, and set it up for distribution.

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.

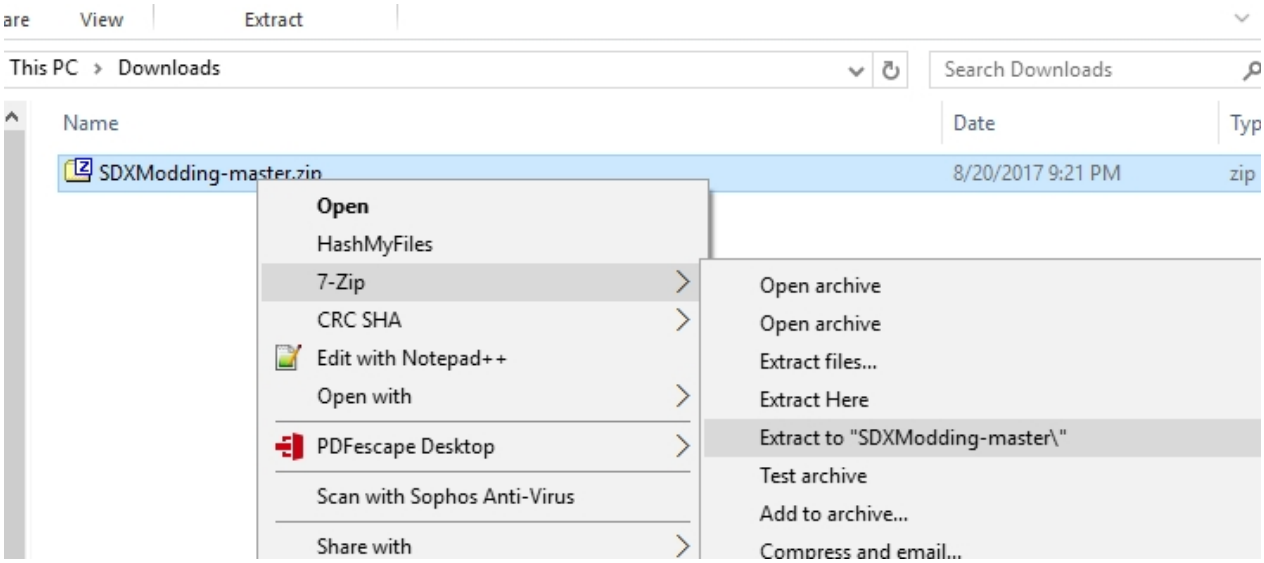
> *Most of the tool chain can be downloaded from the SDXModding Github depot, by downloading this link:*
>
> *DIrect Download Link: https://github.com/7D2DSDX/SDXModding/archive/master.zip*

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:

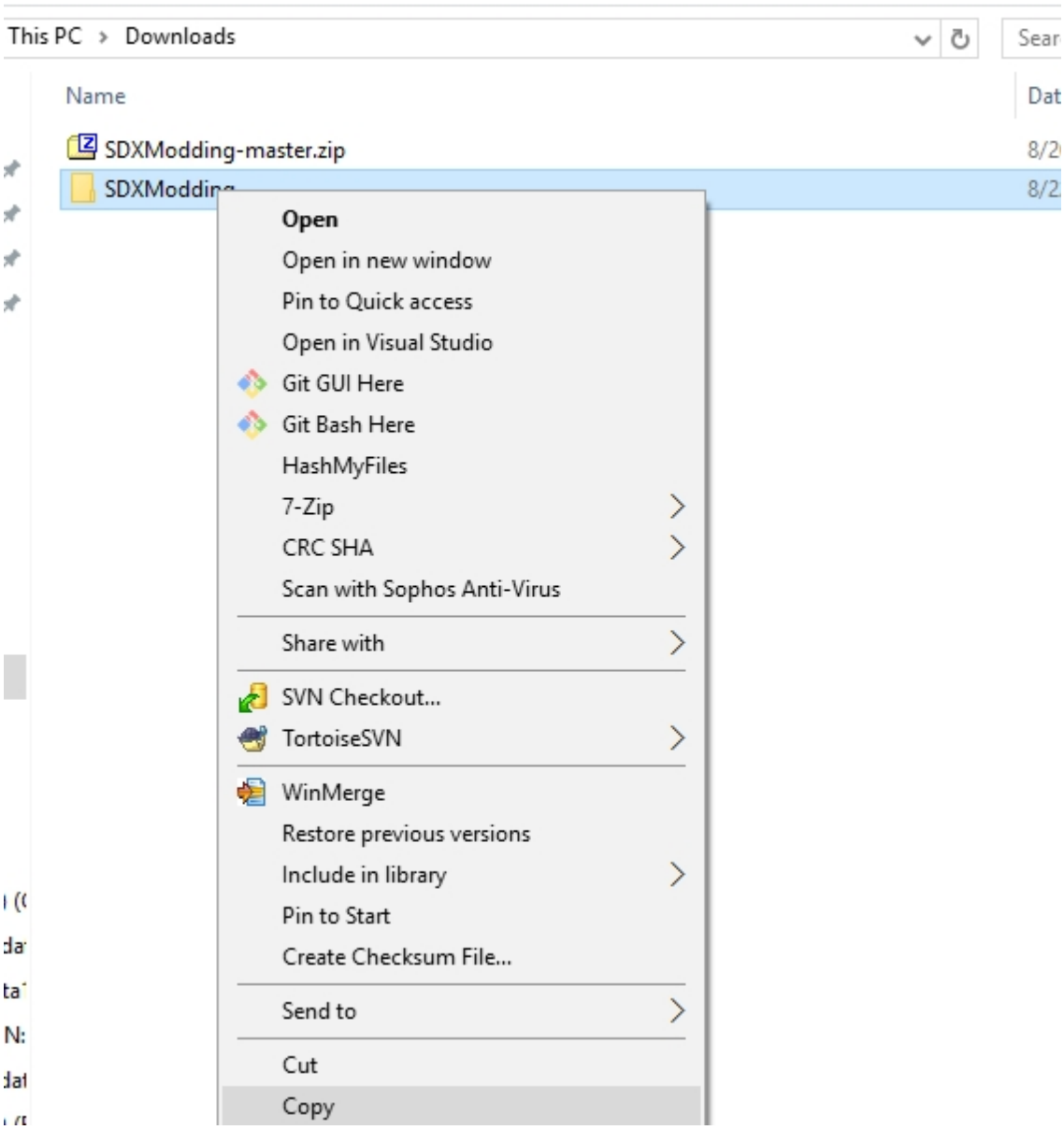



Once extract, right click on the new SDXModding-master folder, and click on rename:

Rename the folder to be SDXModding

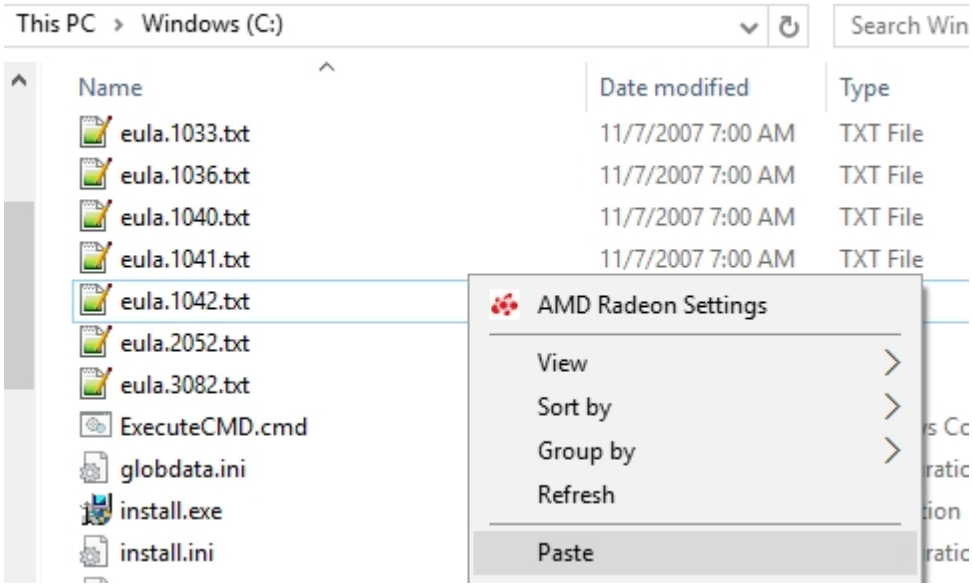
Now we want to copy it to your C:\ or D:\. For the purpose of this Tutorial, it's assumed to be under C:\ or D:\

Right click on the SDXModding folder, and select Copy

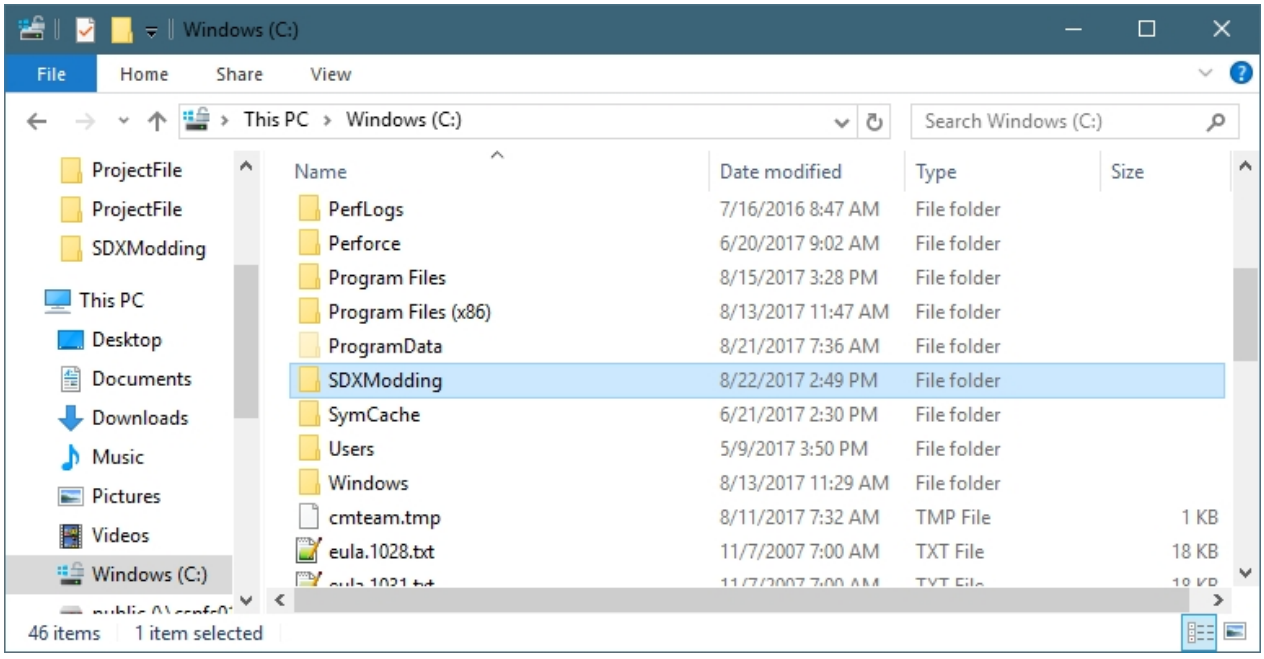



In Explorer, navigate to C:\ or D:\.

Right click on the C:\, and select Paste




You should see something like this:

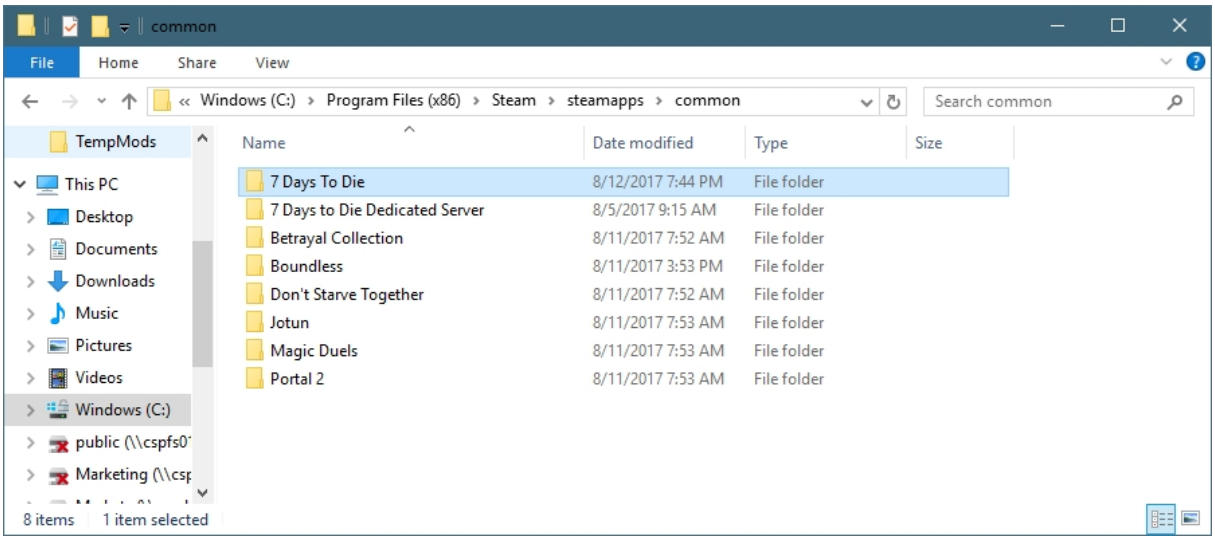




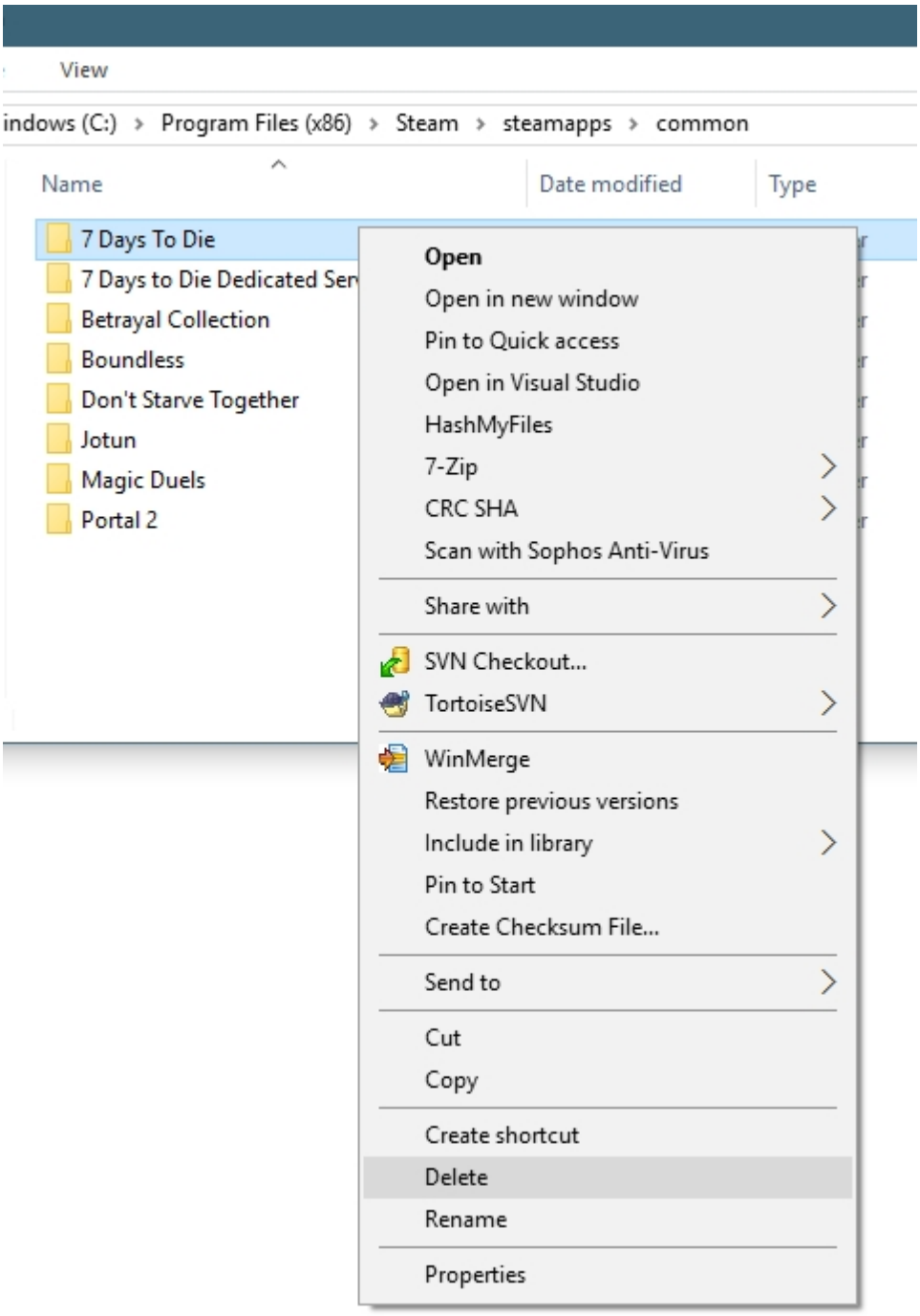

## Starting off Clean

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

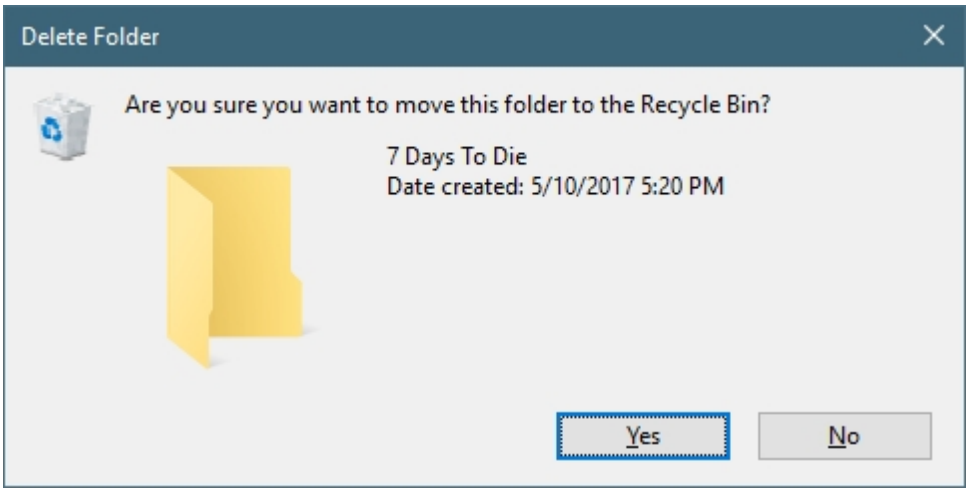
Using Windows explorer, navigate to your Steam Folder, which is by default "*C:\Program Files (x86)\Steam\steamapps\common*"




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

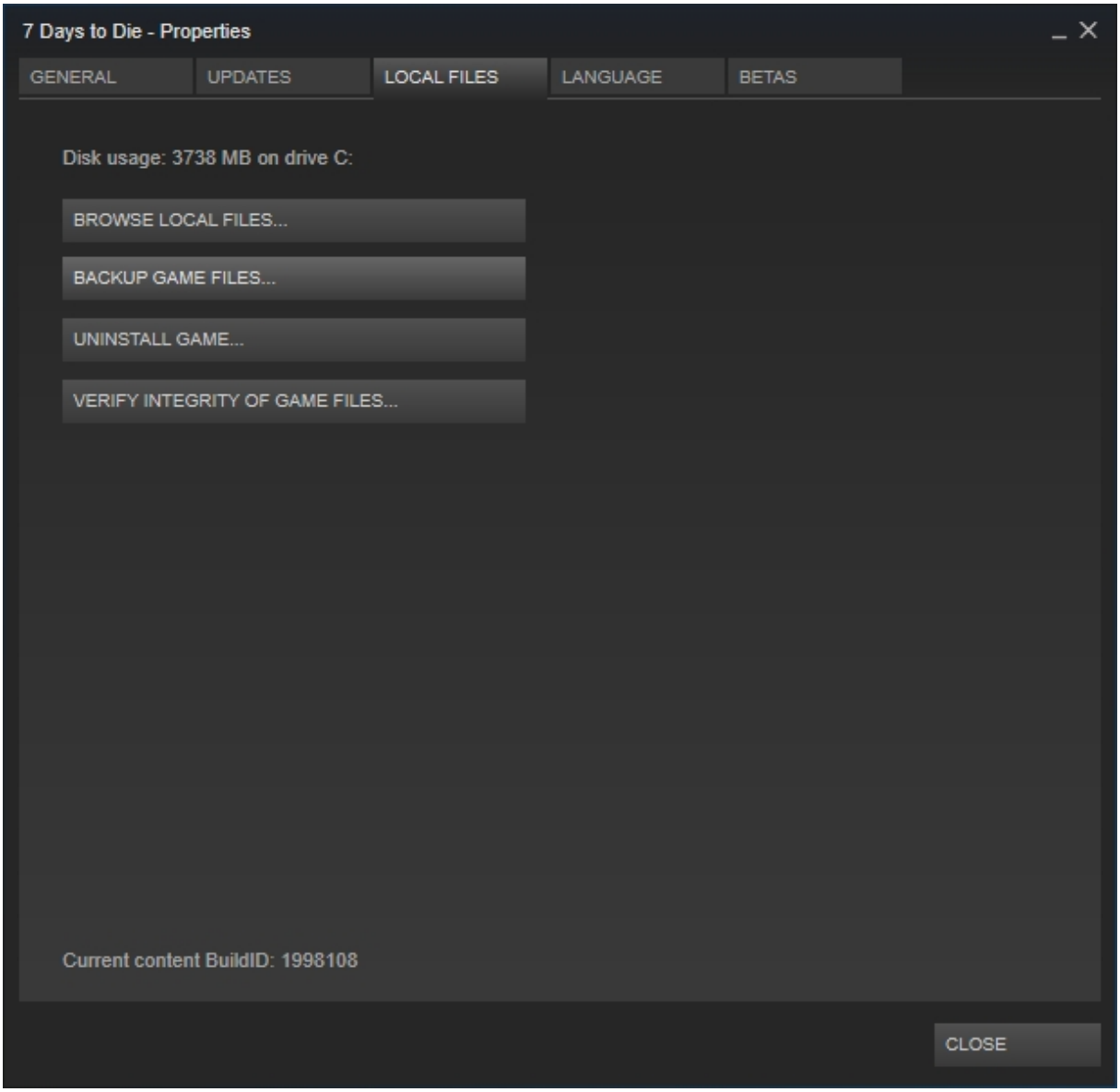



Confirm that you want to delete the "*7 Days To Die*" folder

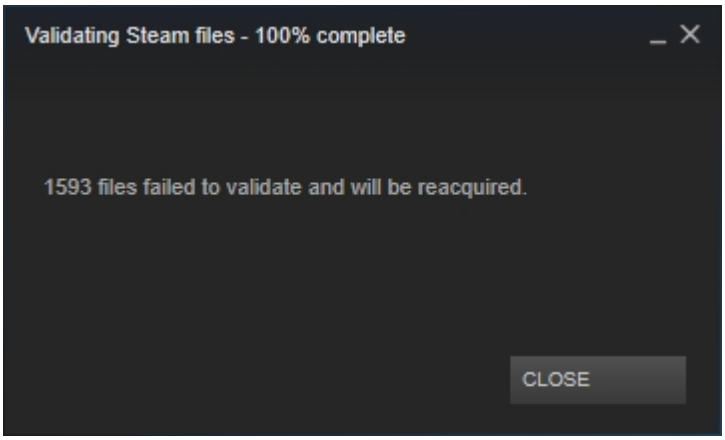



In Steam, right click on the "*7 Days to Die*", and click on Properties


Click on the "*Local Files*" tab




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



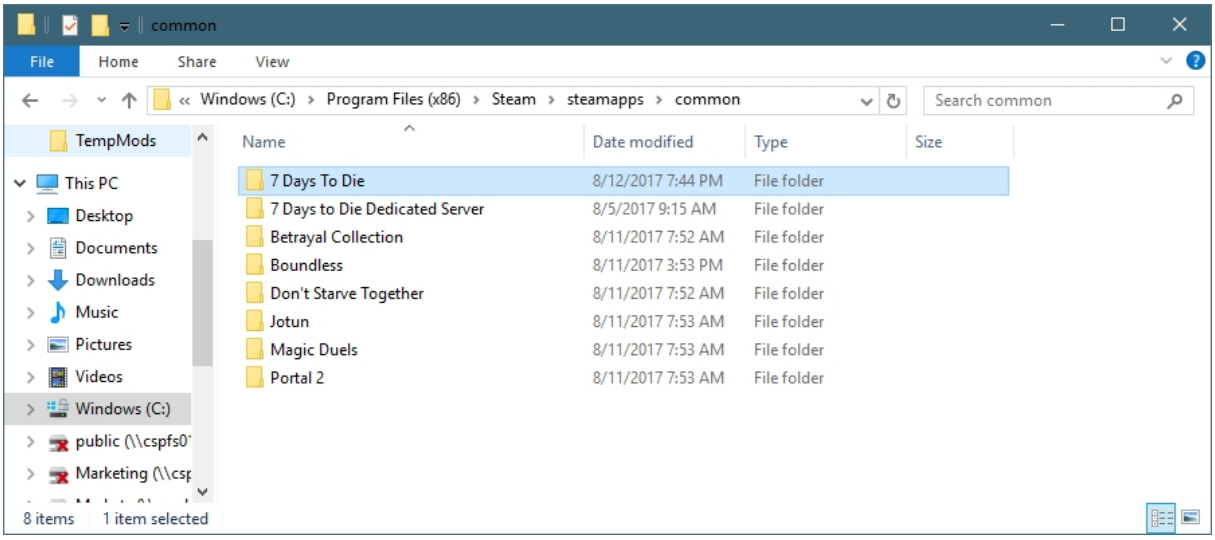

This will download a fresh install of 7 Days to Die of the latest stable release.

## Making a Clean Backup

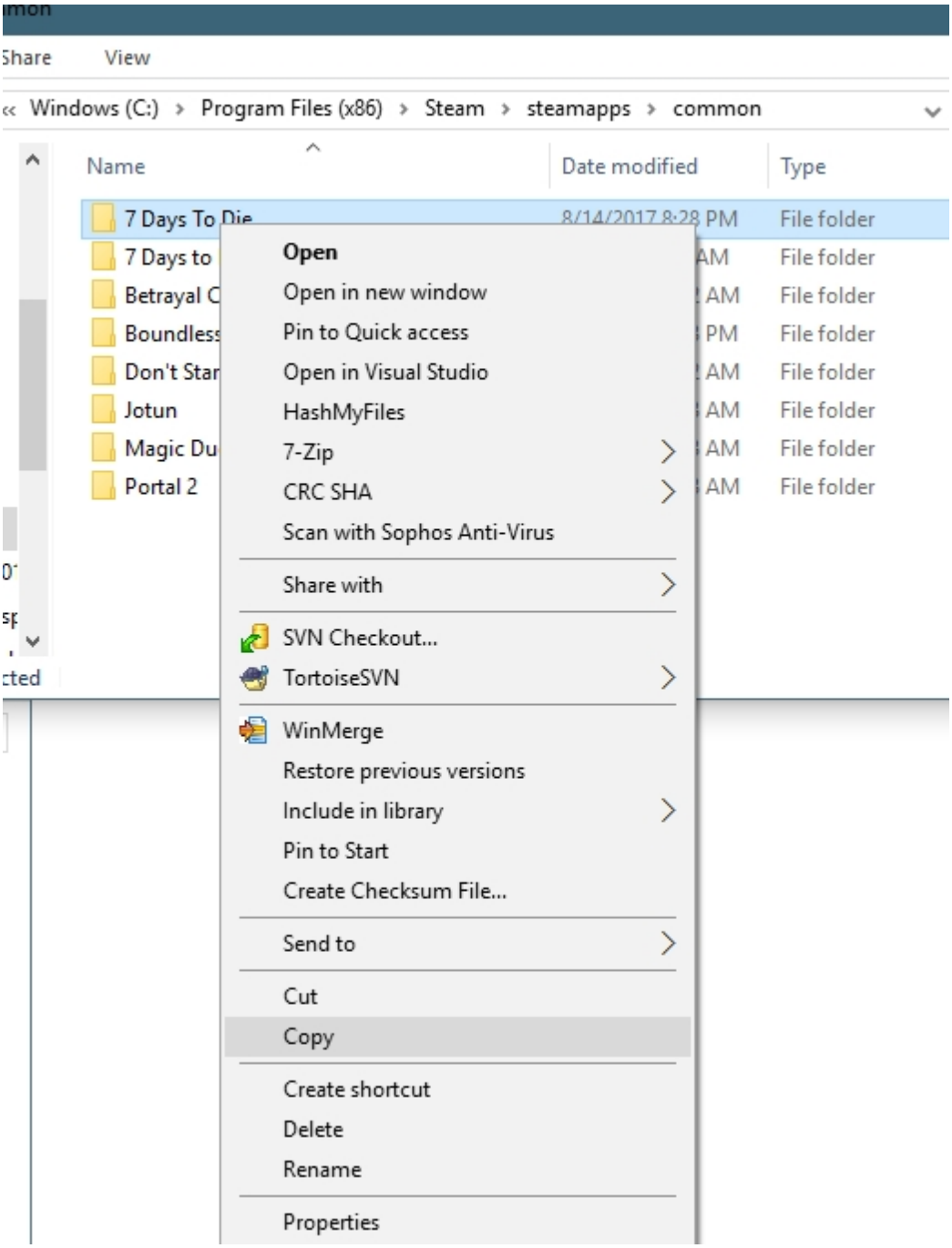
One you have completed the "Starting off Clean" section, it's time to make two copies.

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.

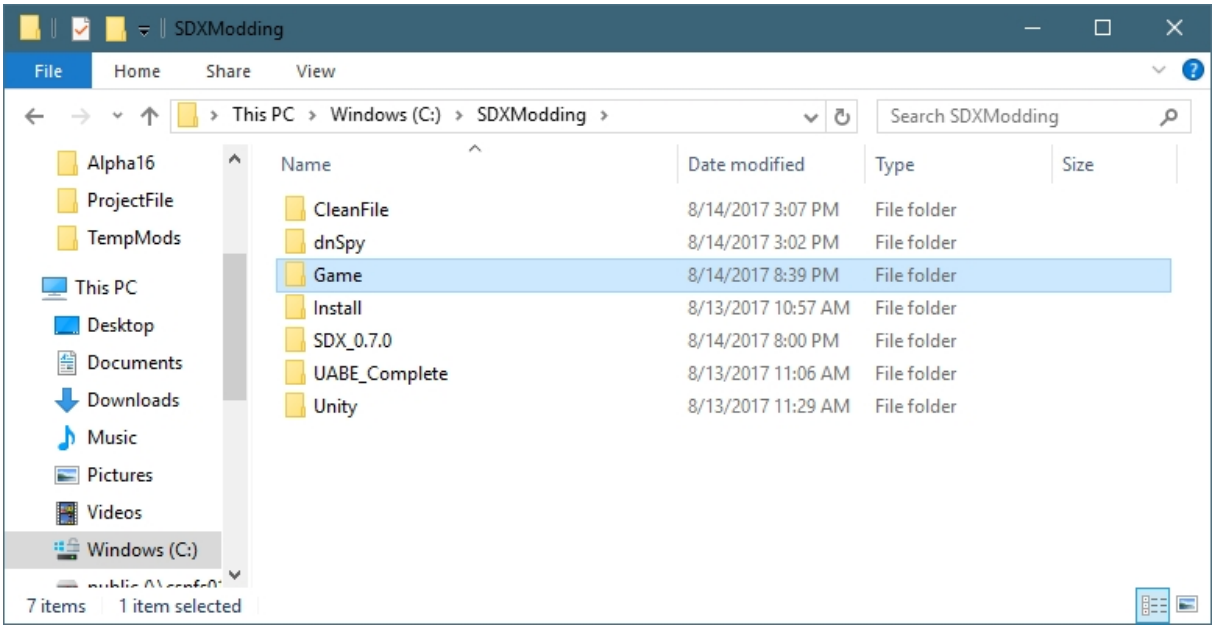
Using Windows explorer, navigate to your Steam Folder, which is by default "*C:\Program Files (x86)\Steam\steamapps\common*"



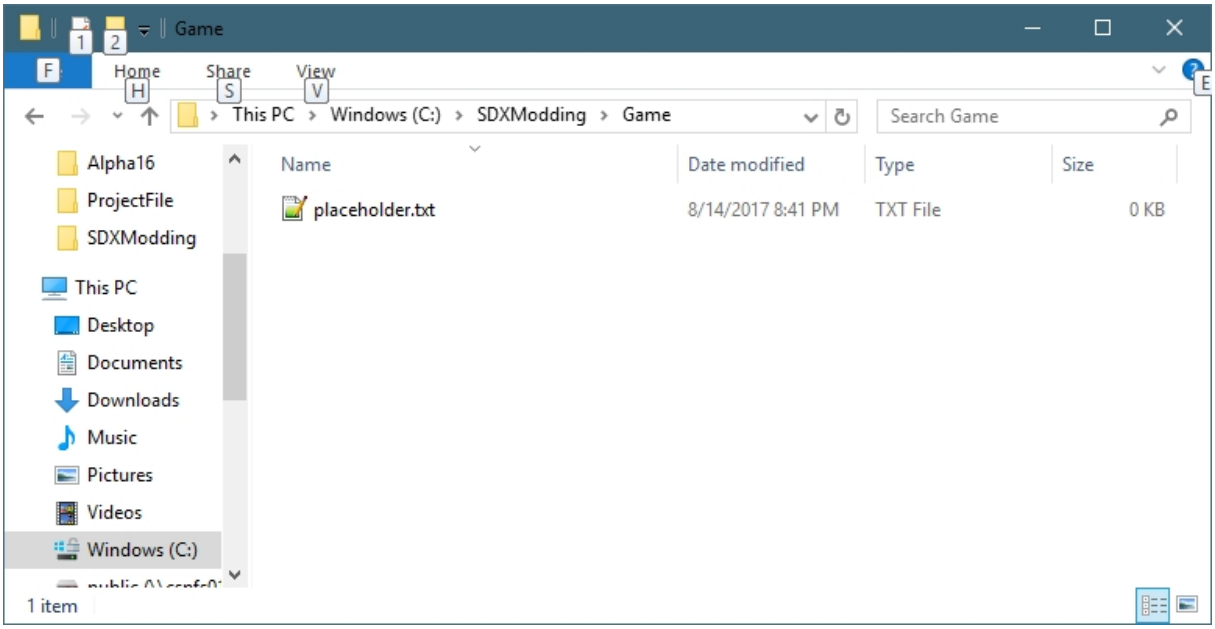

Right Click on the "*7 Days To Die*" folder, and select Copy



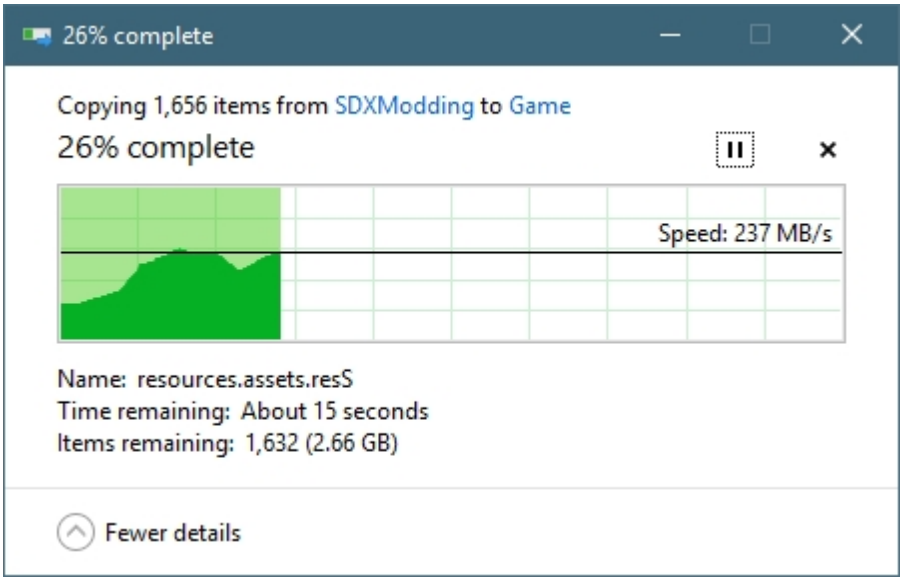

Open up an Explorer Window where you have installed the SDXModding, such as C:\SDXModding\Game\



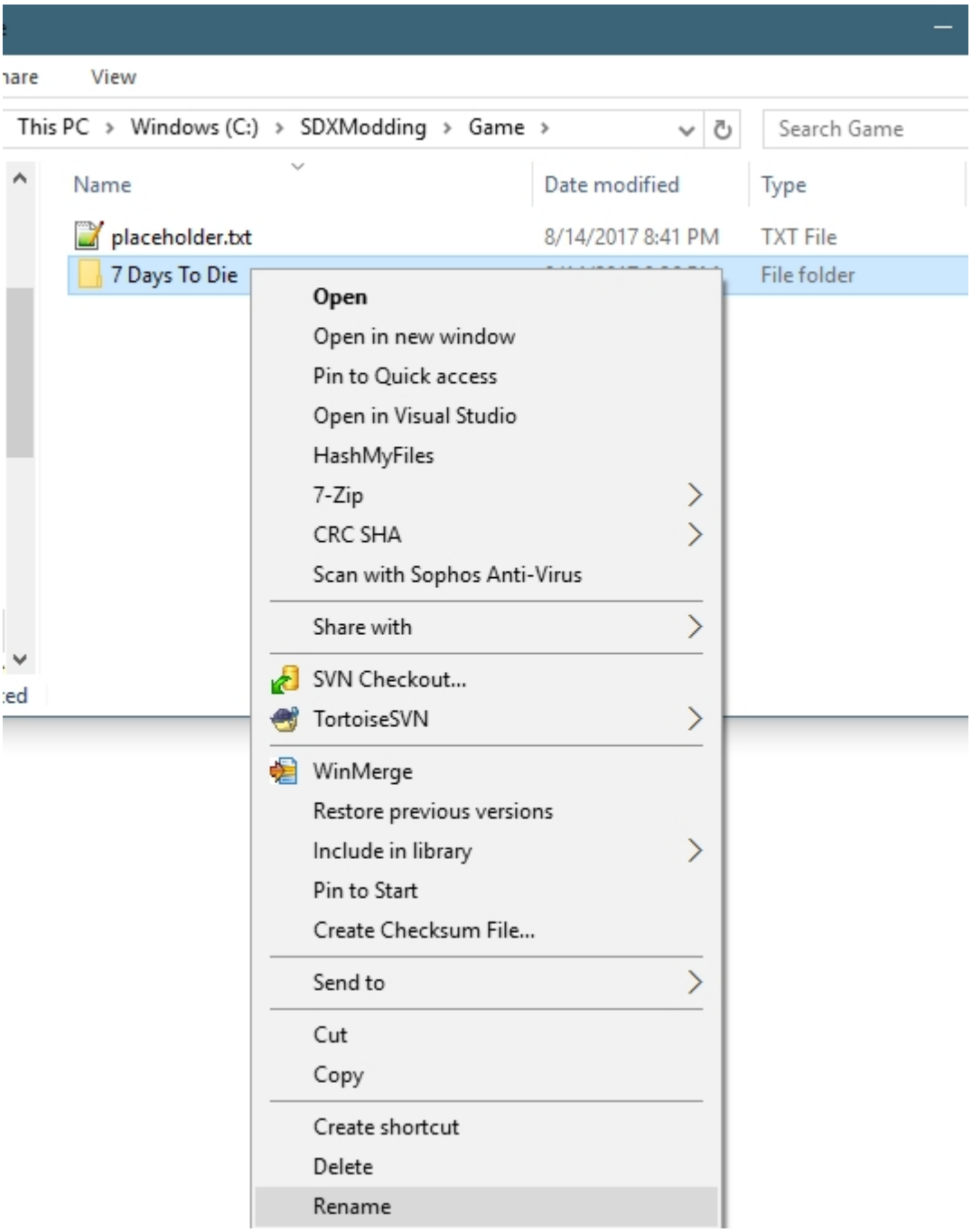

Double Click to open the "Game" Folder.



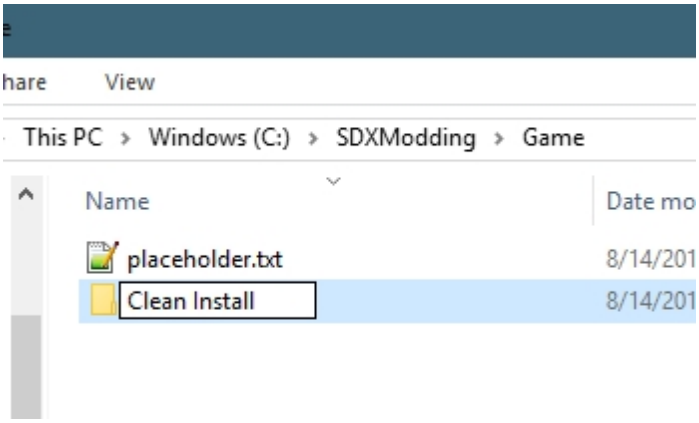

Right click on the folder, and select "*Paste*". It will then make a copy of the game:




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"



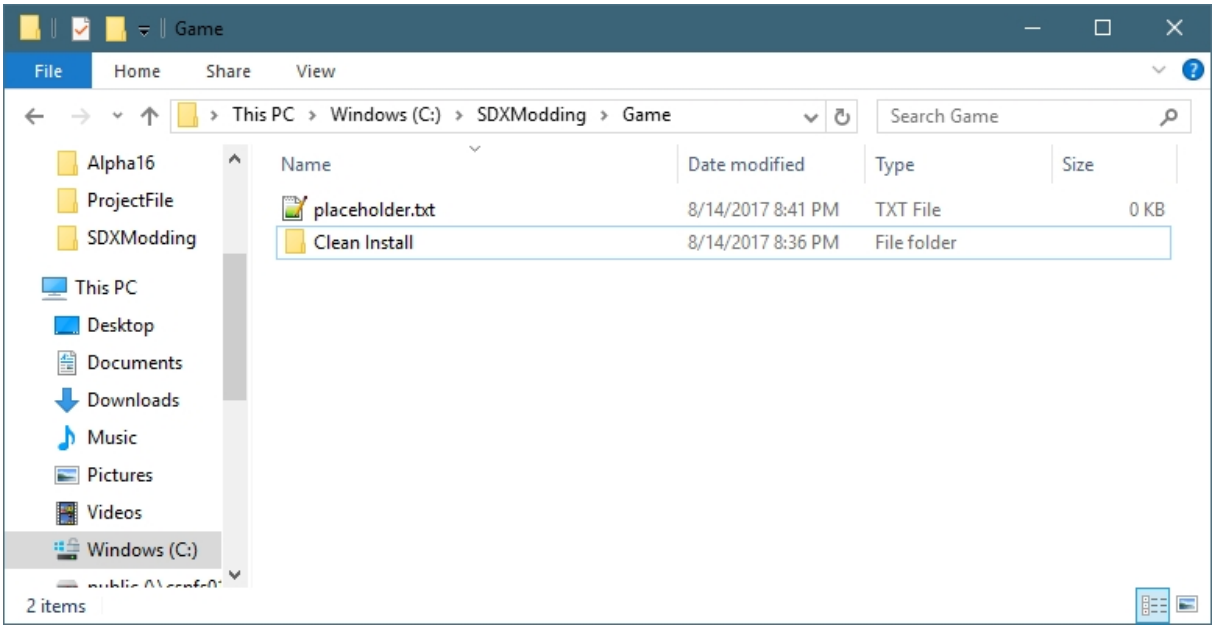
You now have a perfectly preserved copy of the game.



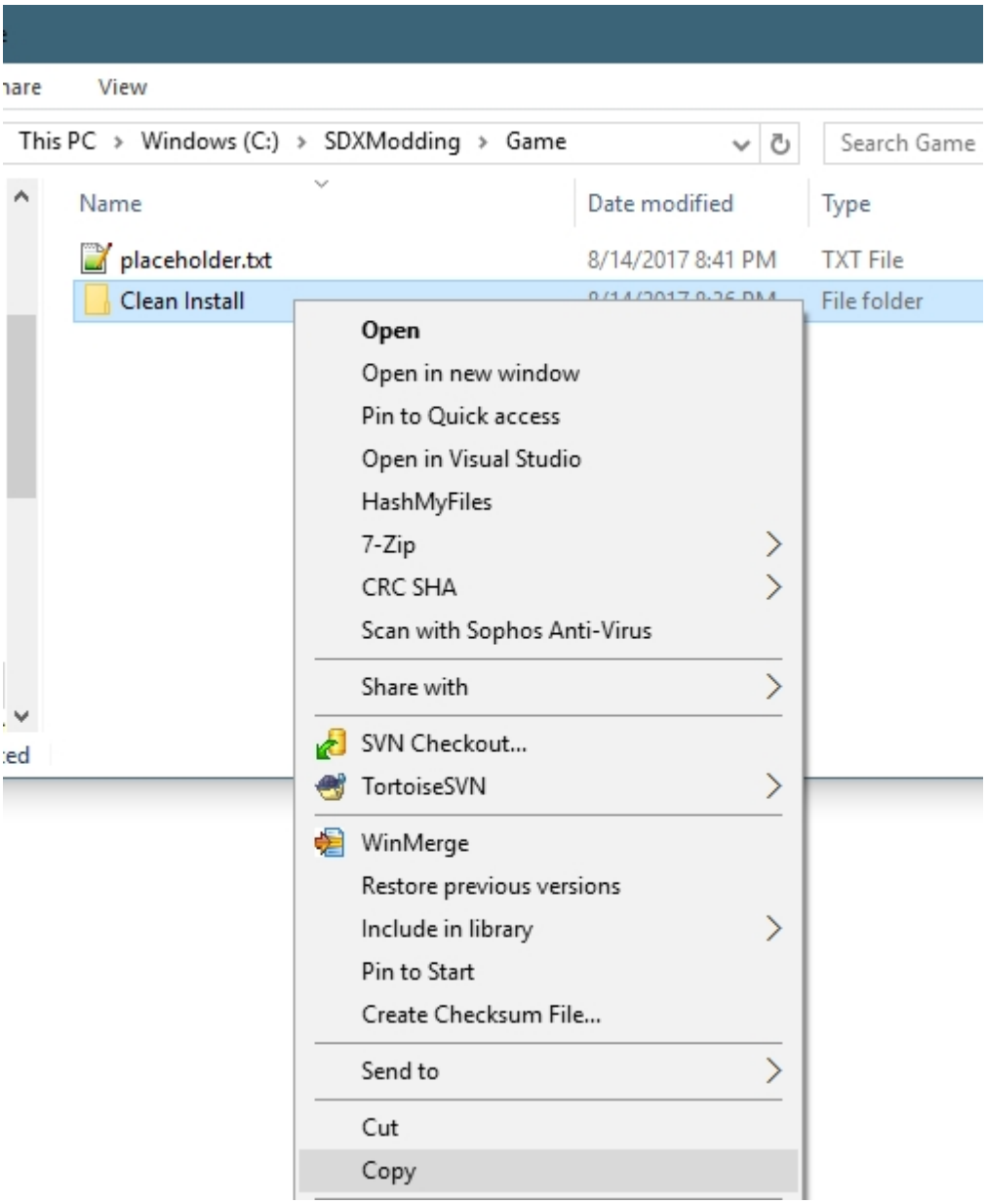
## Making a Working Folder

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.

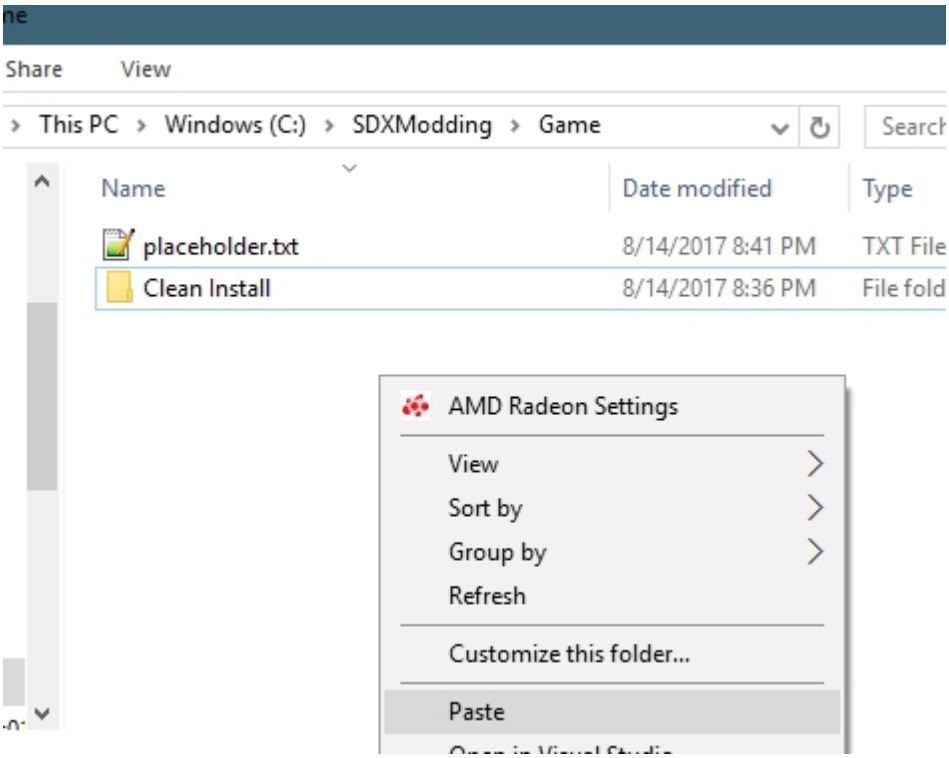
Using Explorer, navigate to your "*C:\SDXModding\Game*" Folder.




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

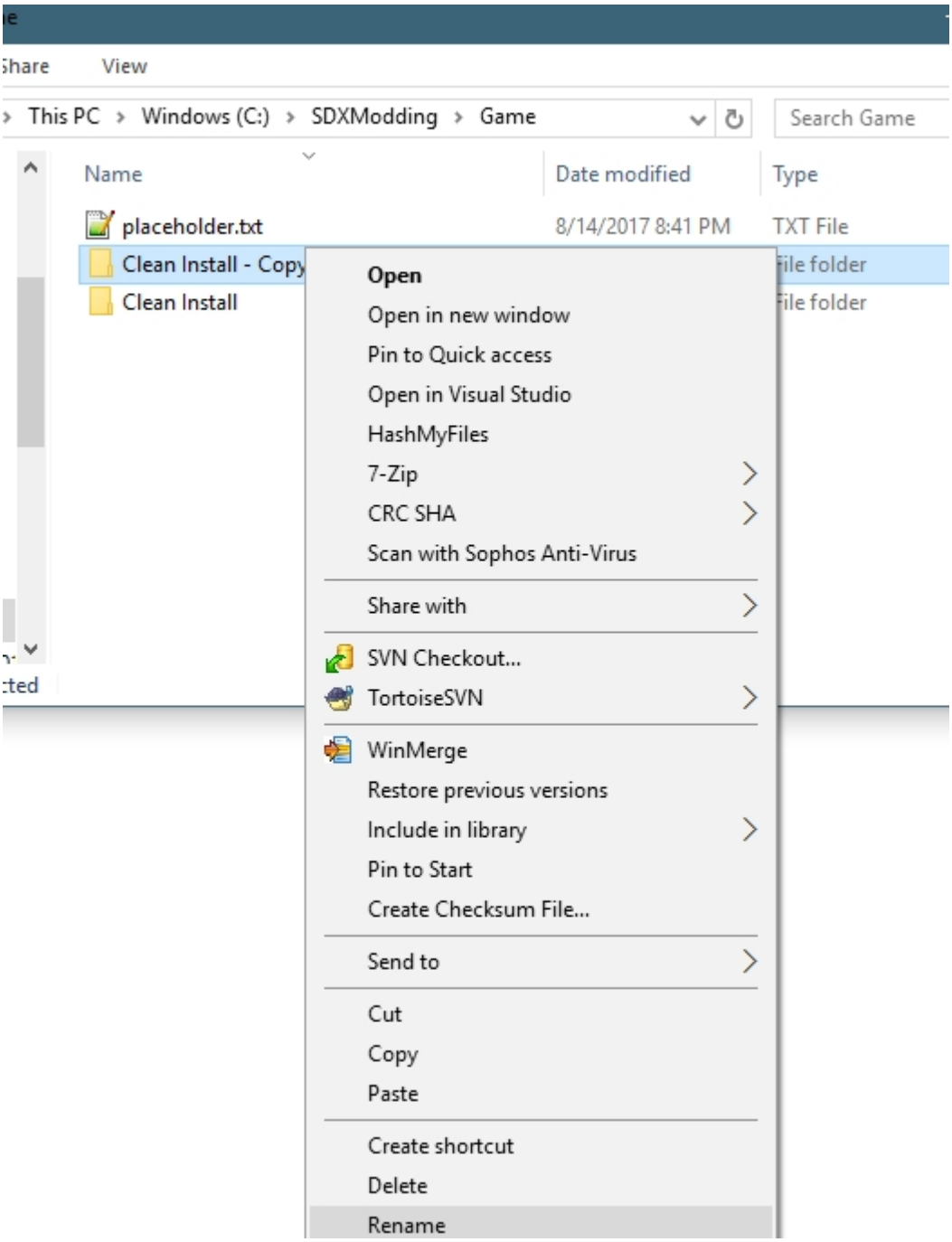



Right click on the Explorer Window again, and select "*Paste*"

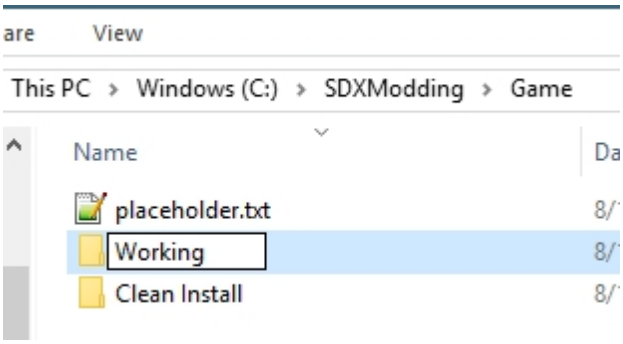



This will make a new copy called "*Clean Install - Copy*".

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




Rename it to "*Working*"



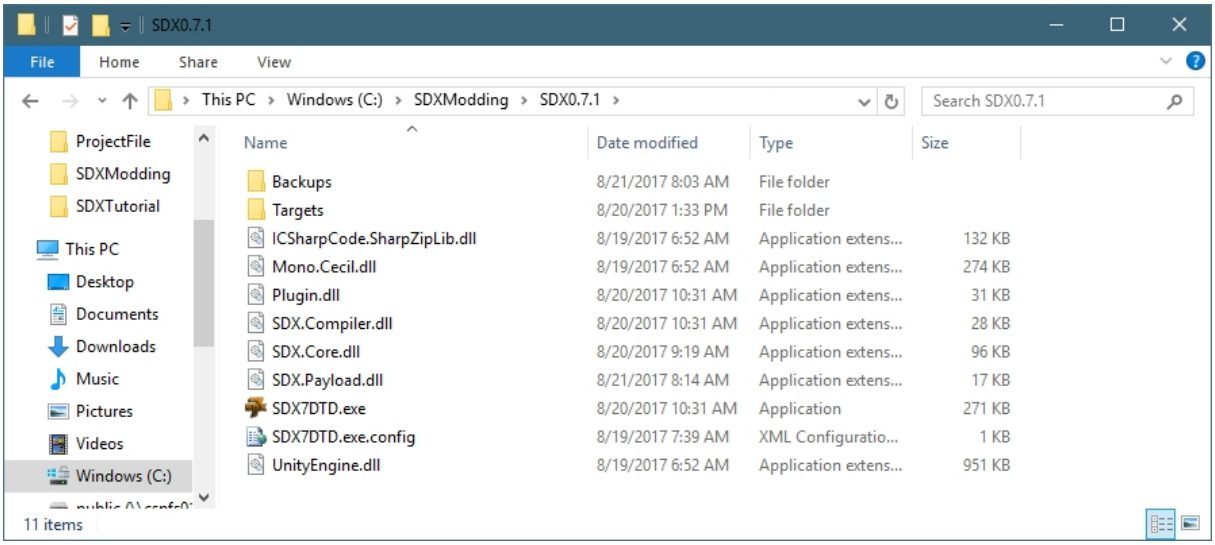

There we go! Now we have two copies of the game. One for a back up, and the other as our working folder.

## SDX Launcher

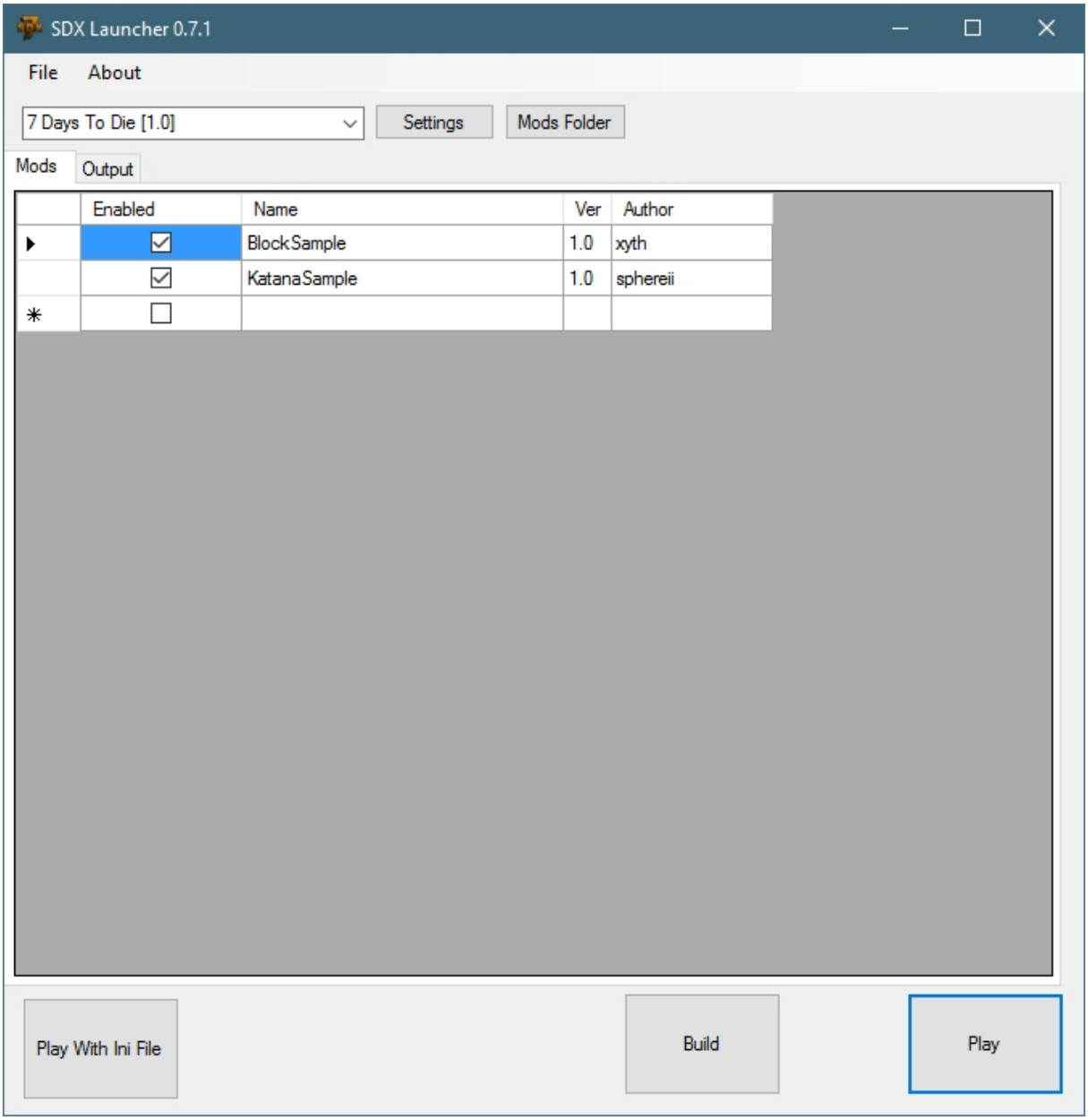
The SDX7DTD.exe is the GUI front end of the SDX Tool, and will be used to compile the mods.

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:



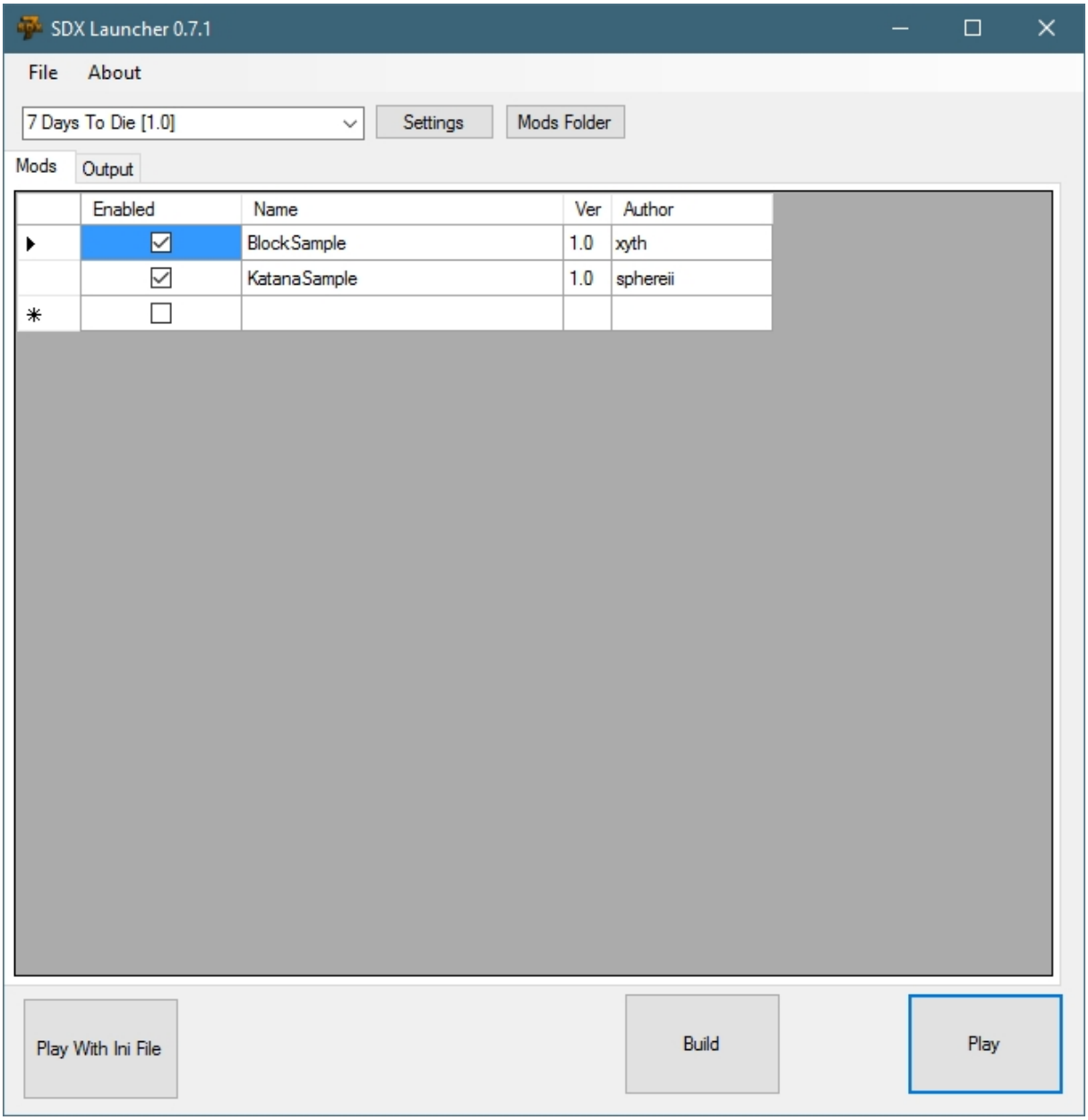



## Settings Button

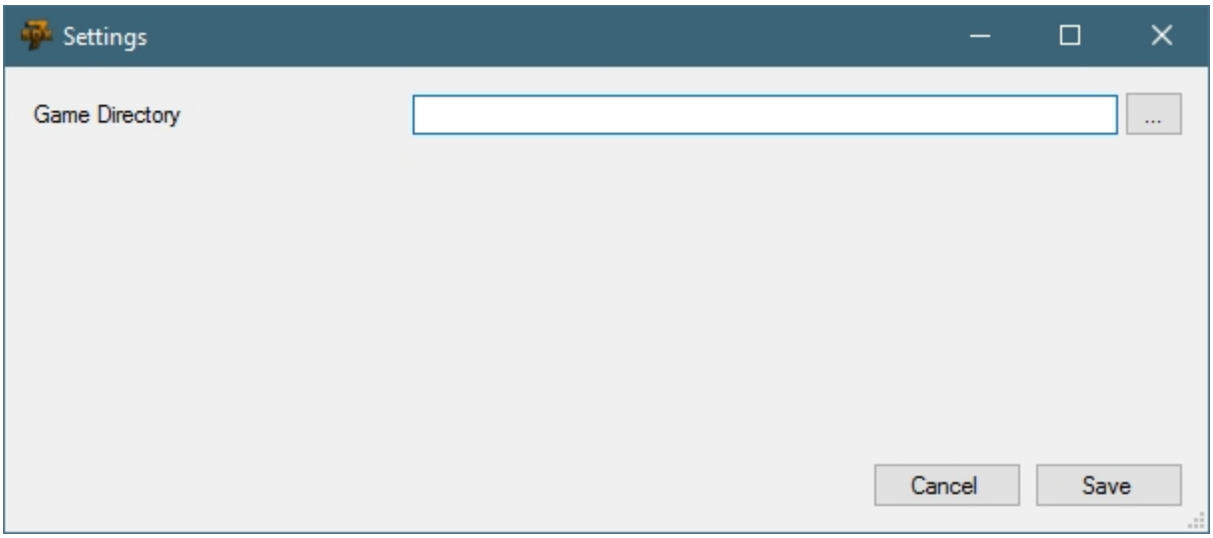
Once you have started the SDX Launcher 0.7.1, you will need to configure it for the first time.

*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.*

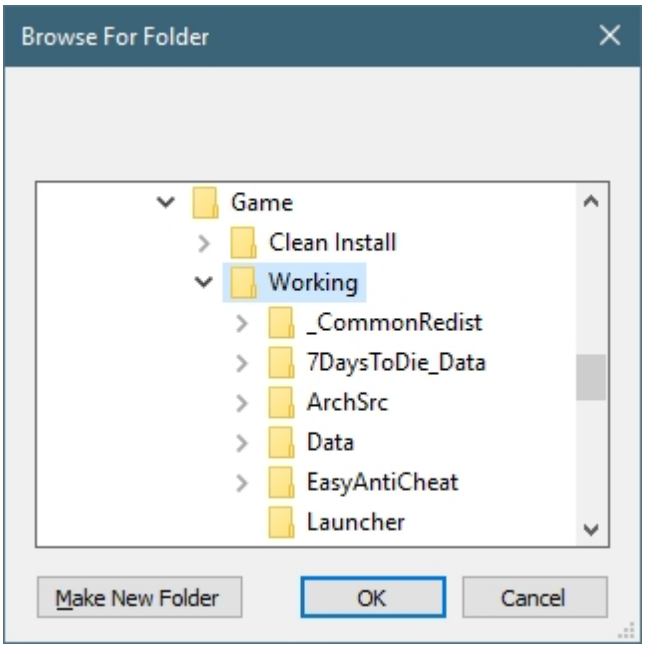
Click on the "Settings" button.



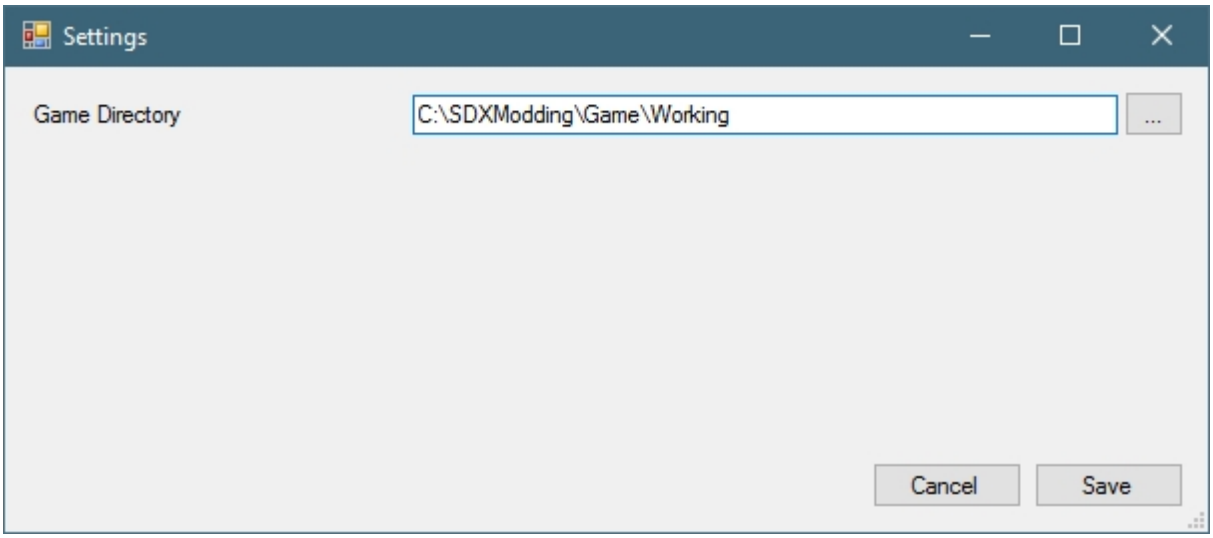

This is asking where your "Game Directory" exist. For us, it'll be that "Working" Folder we created here "C:\SDXModding\Game\Working"




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




End result should look something like this:

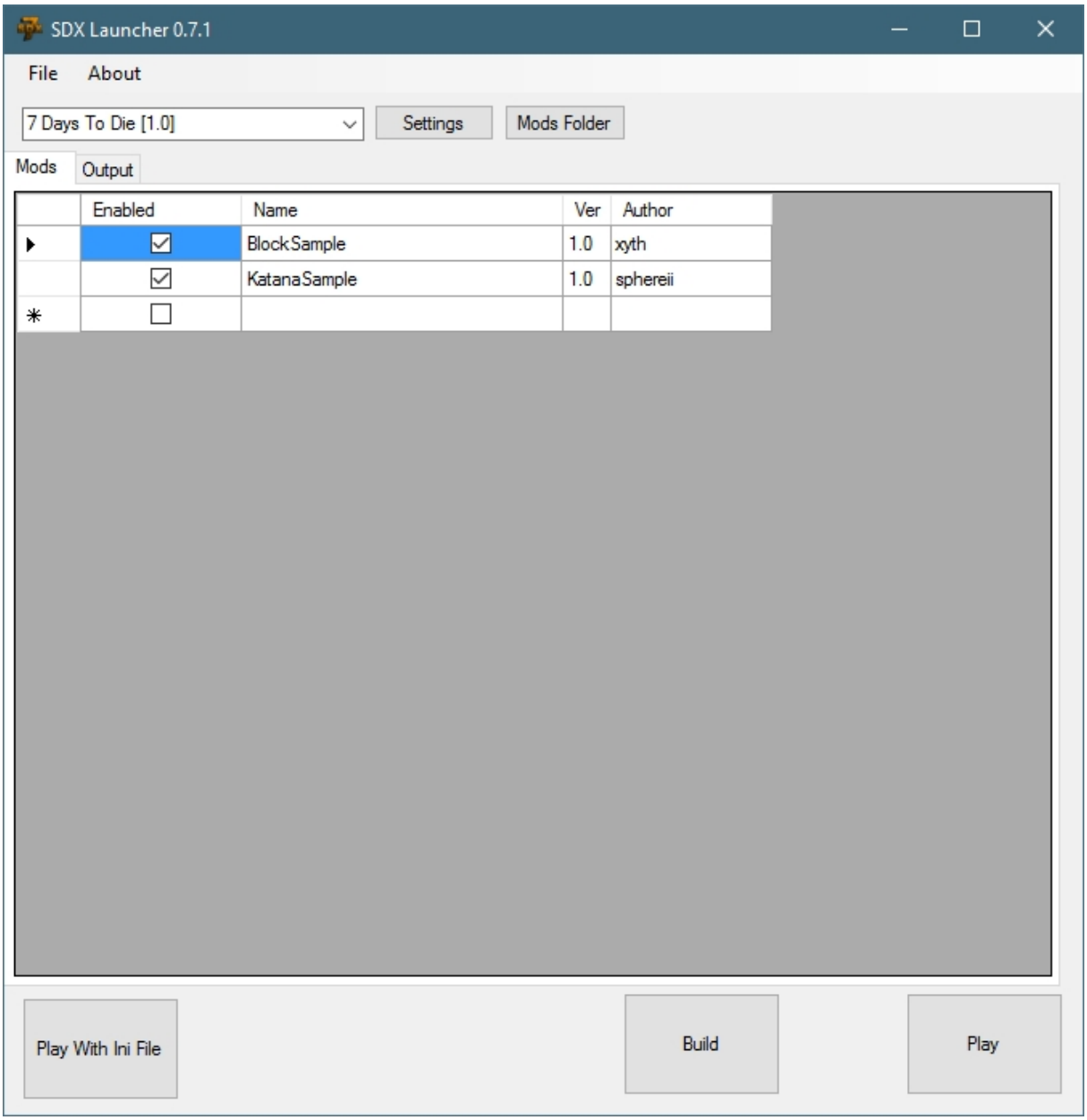




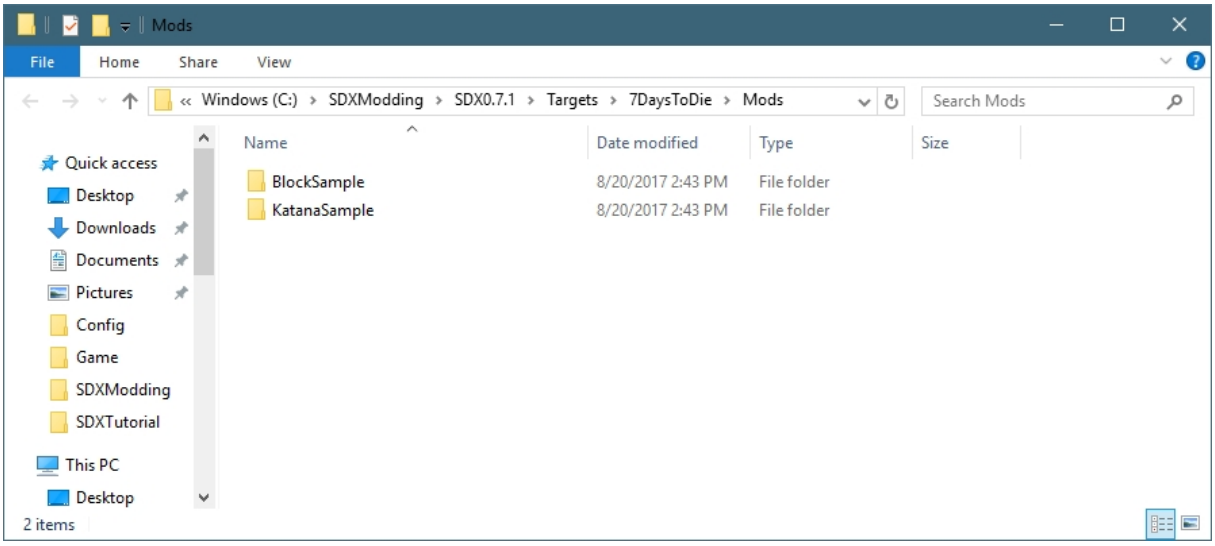

## Mods Folder Button

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*"

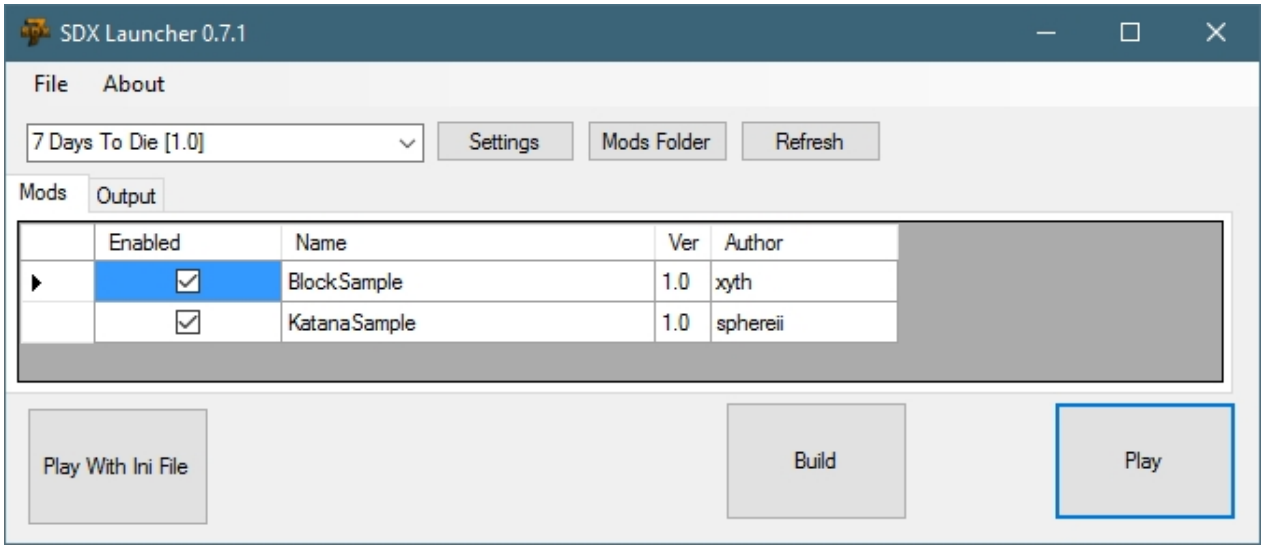



Mods Folder in Explorer



## 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.

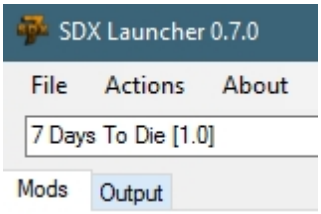
## *Play*

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

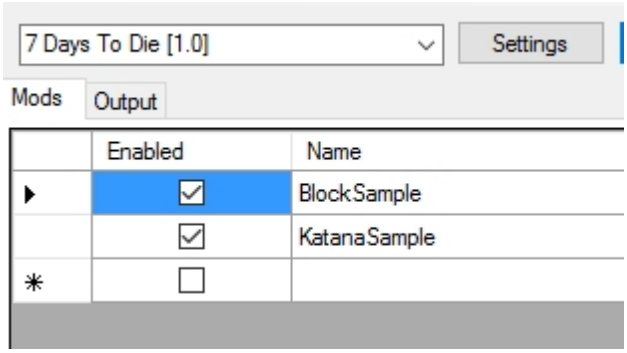


## The Mods / Output tab

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

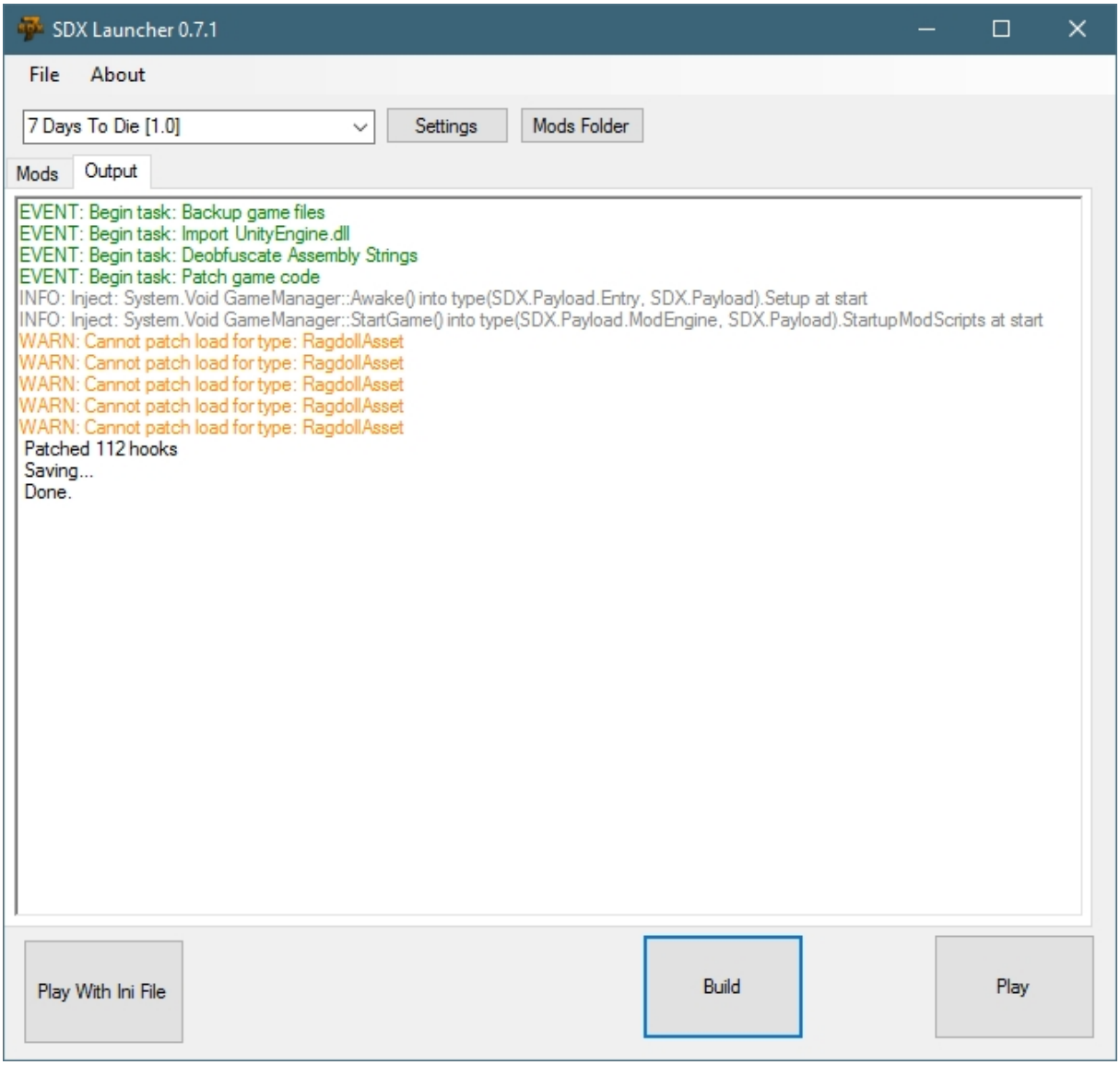



The "Mods" tab shows you the available Mods:




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.

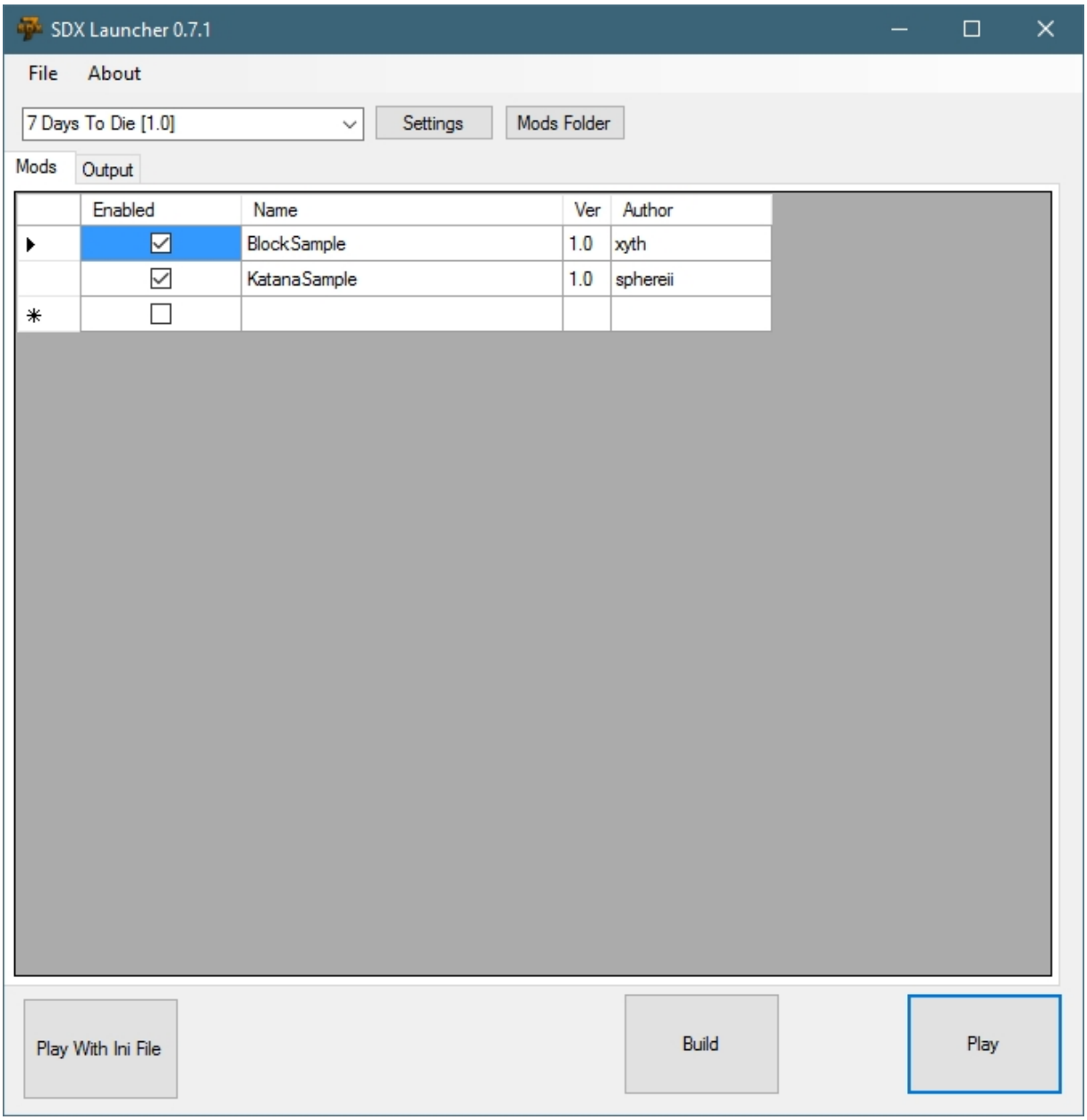



You can click back and forth between the Mods and Output folder.

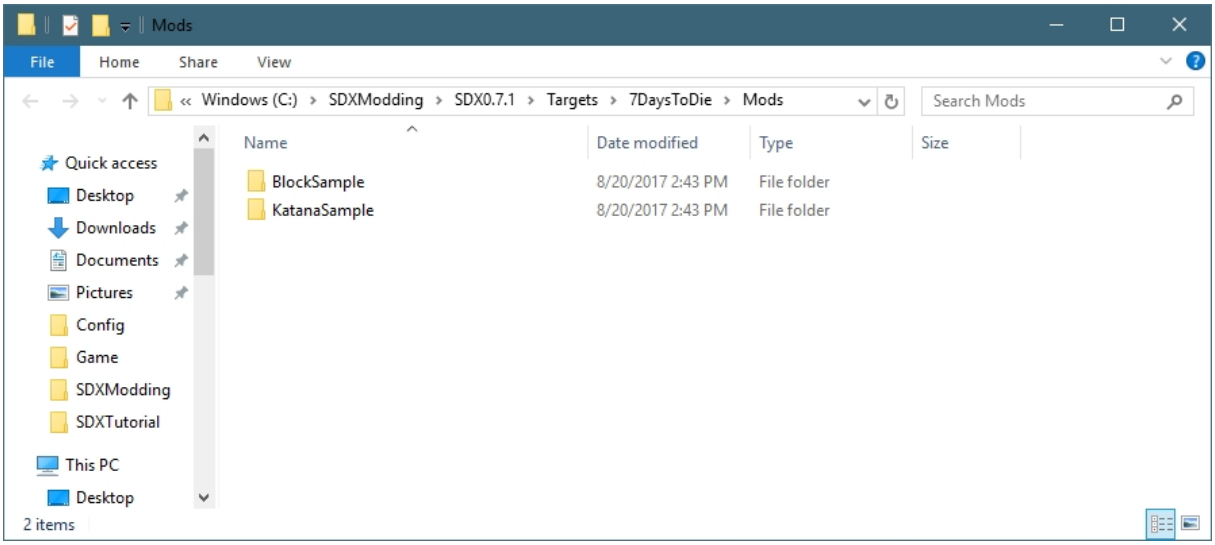


## Undertanding an SDX mod

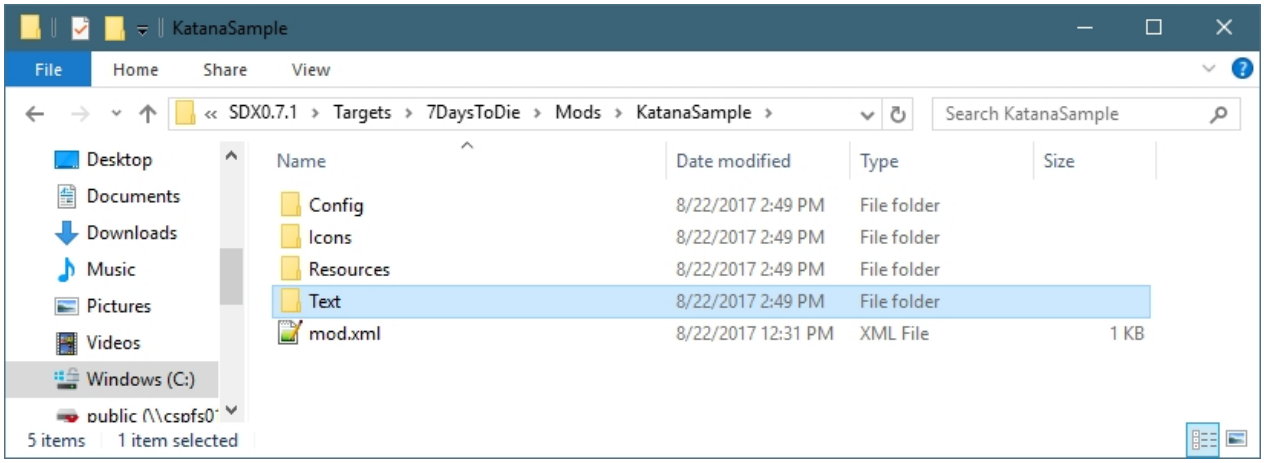
Start the SDX Launcher




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.

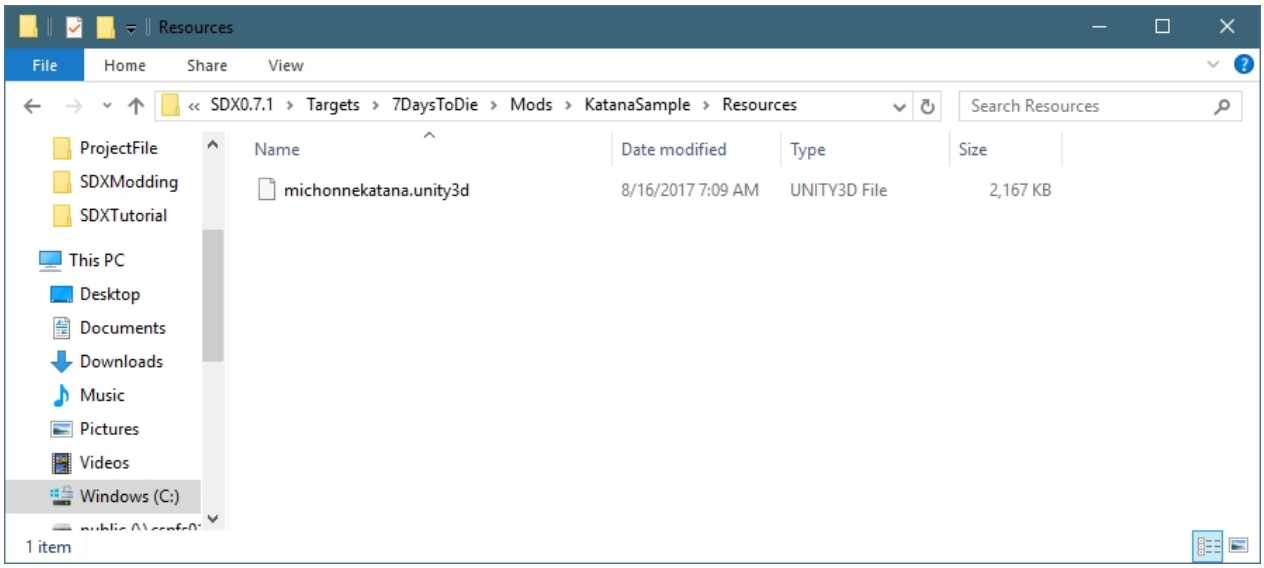
All of the <config> tags need to be in between the root node, which is the <configs> tags.

| 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.

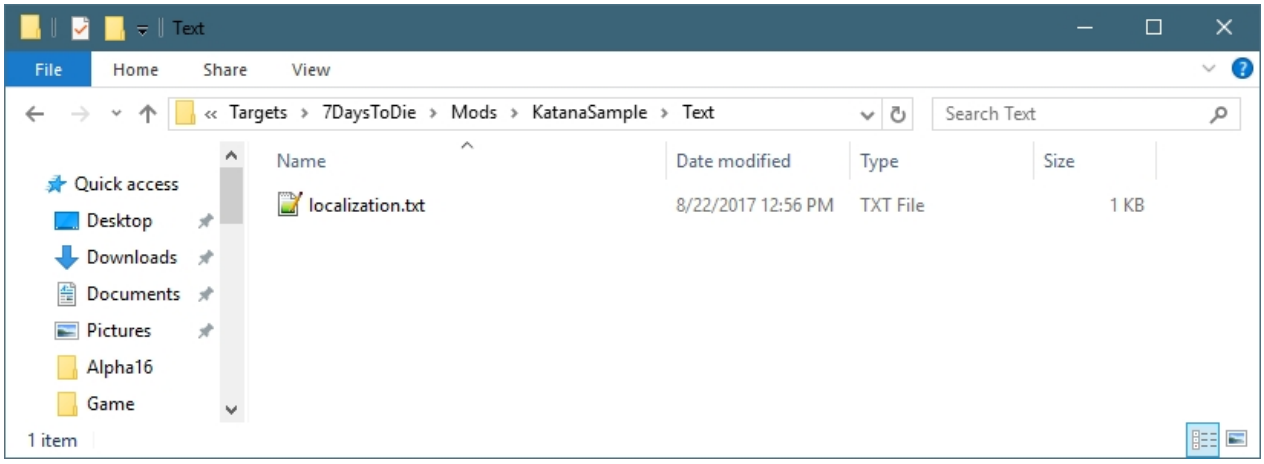
# Resources:

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.

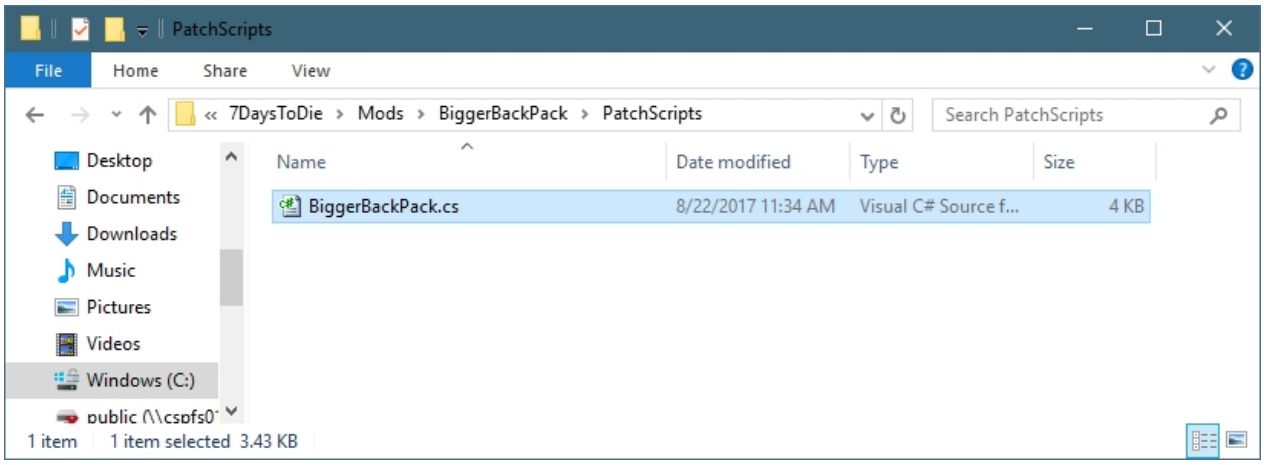



Example:

```
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.

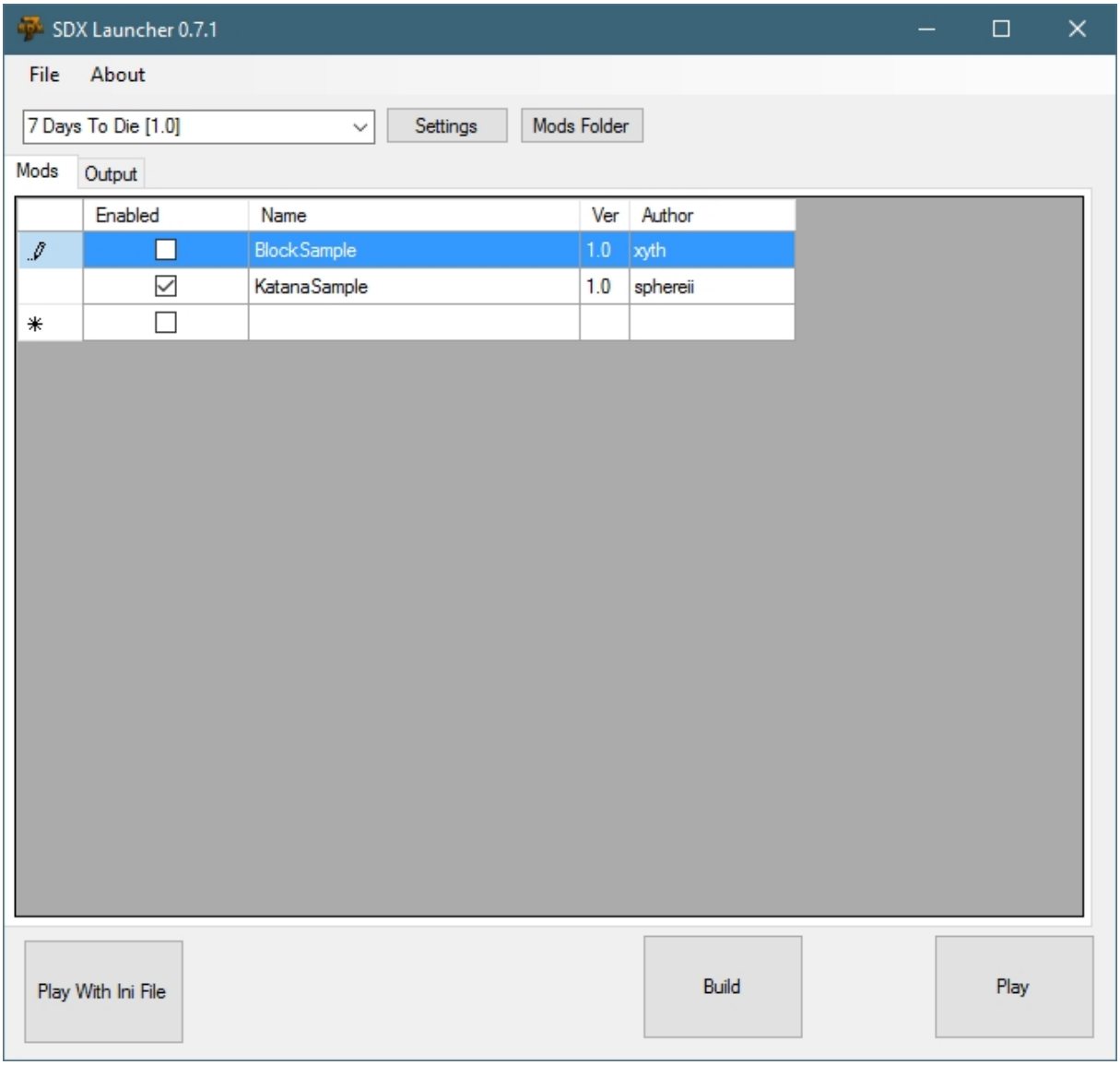
Finally, when you are ready, we'll add an actual item into the game, using the supplied Katana Mod.



## Building for the first time

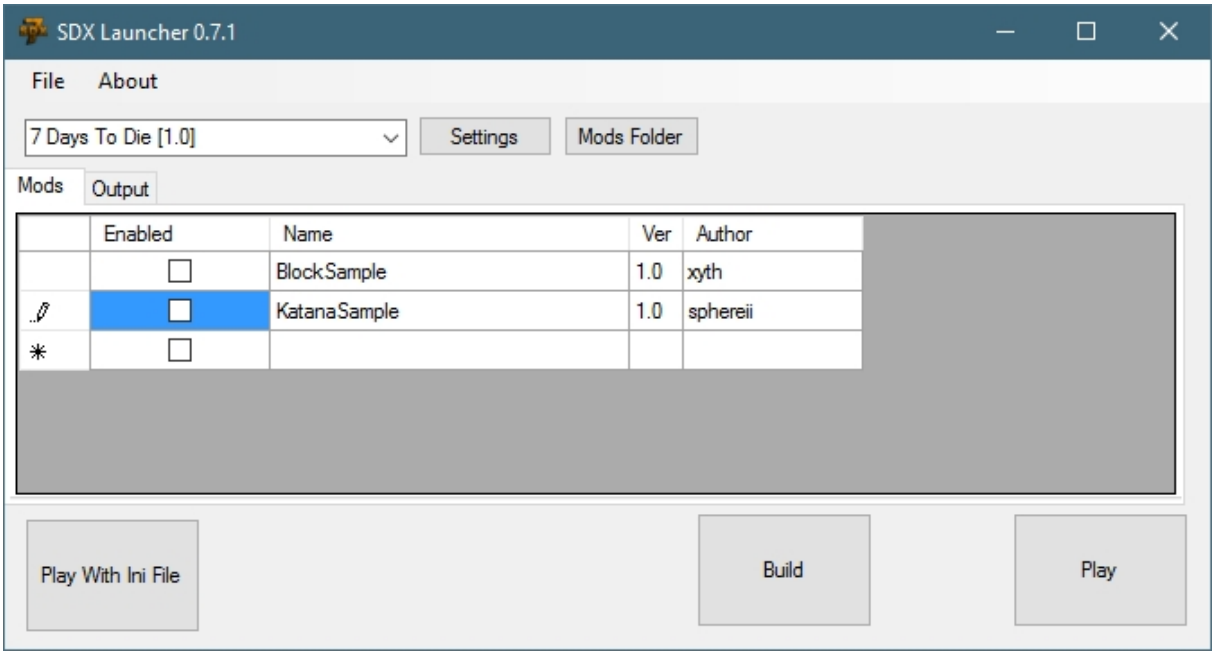
Once you have completed all the steps in the Getting Started section, it's time to trigger your first SDX compile!

Start the SDX Launcher



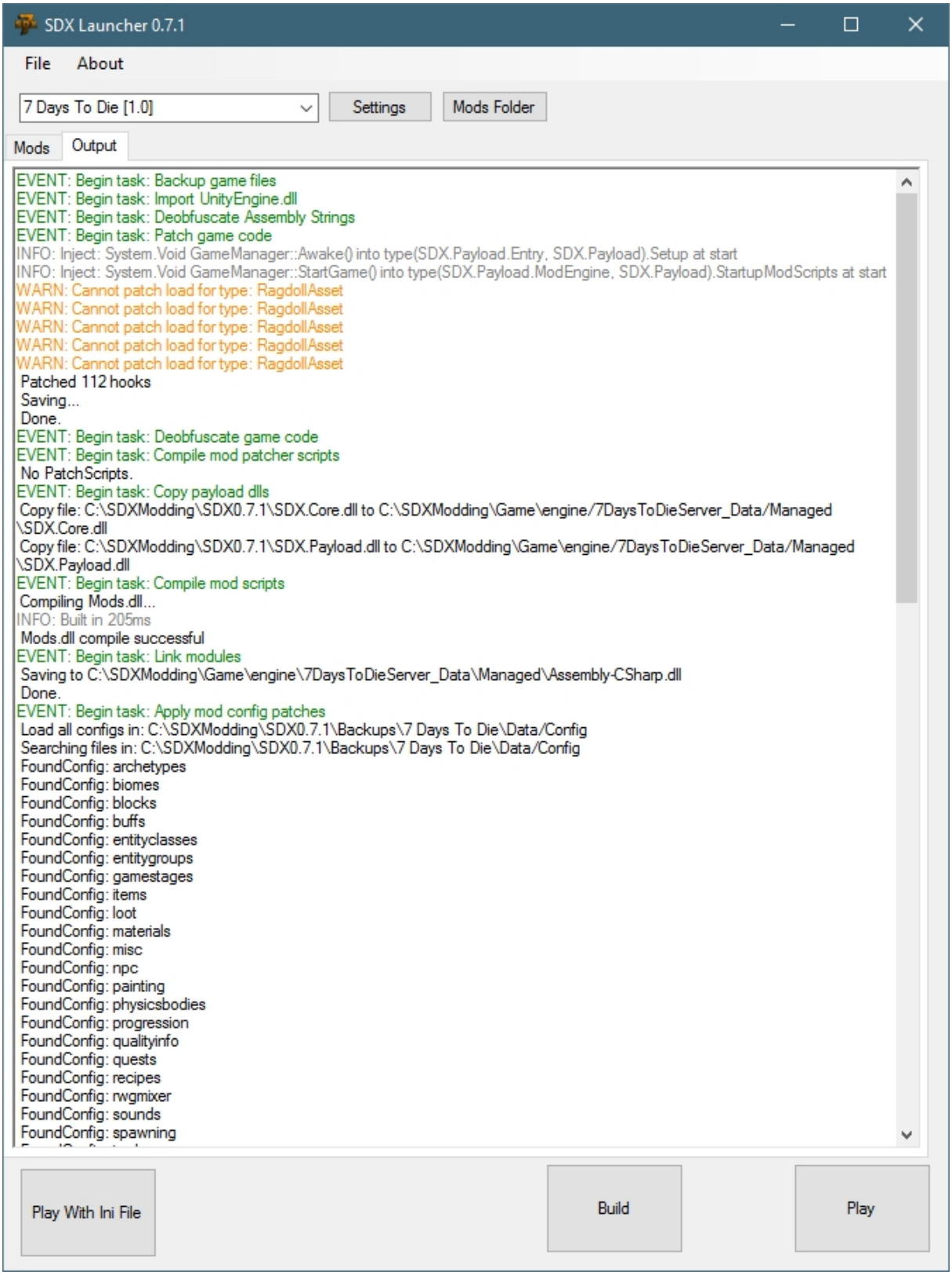

For the first compile, we want to disable all the available mods, to make sure everything is set up correctly.

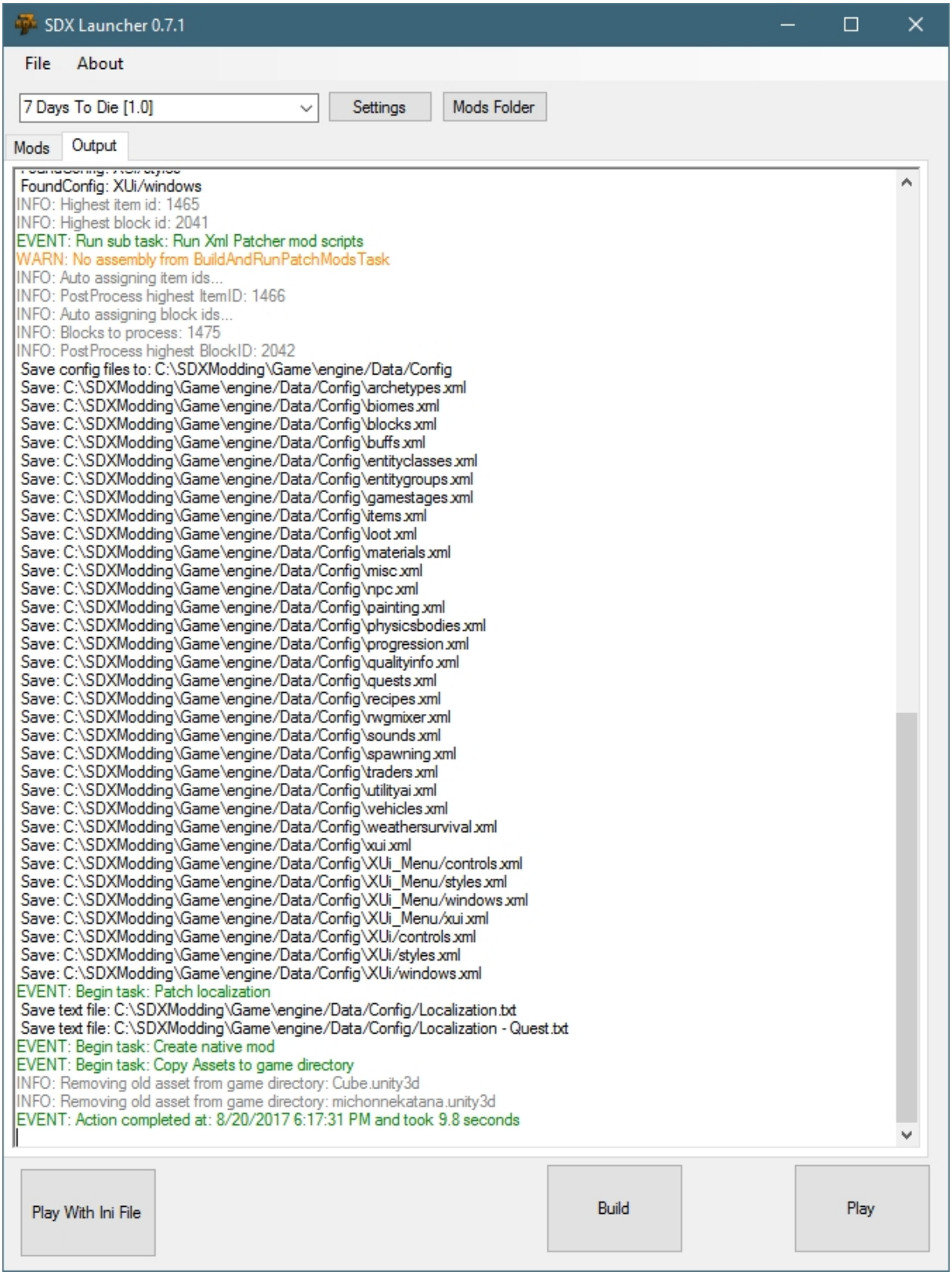
For each Mod listed in the "Mods" tab, click on the checkmark to disable it.




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

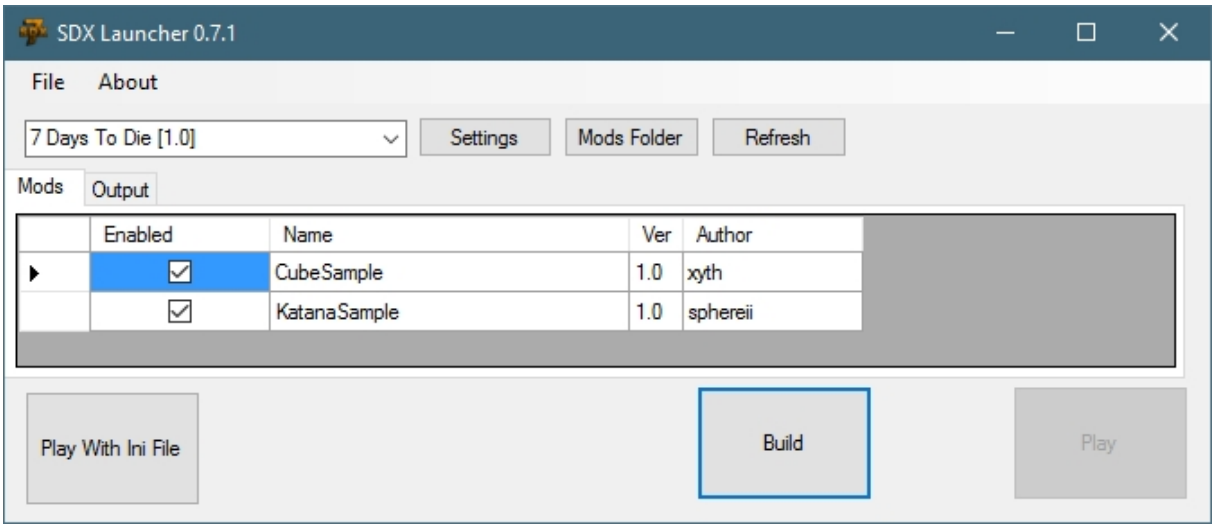

The first SDX Mod we are going to add is the Cube Mod. This will add a new cube to the game, letting you place it in game.



## 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!

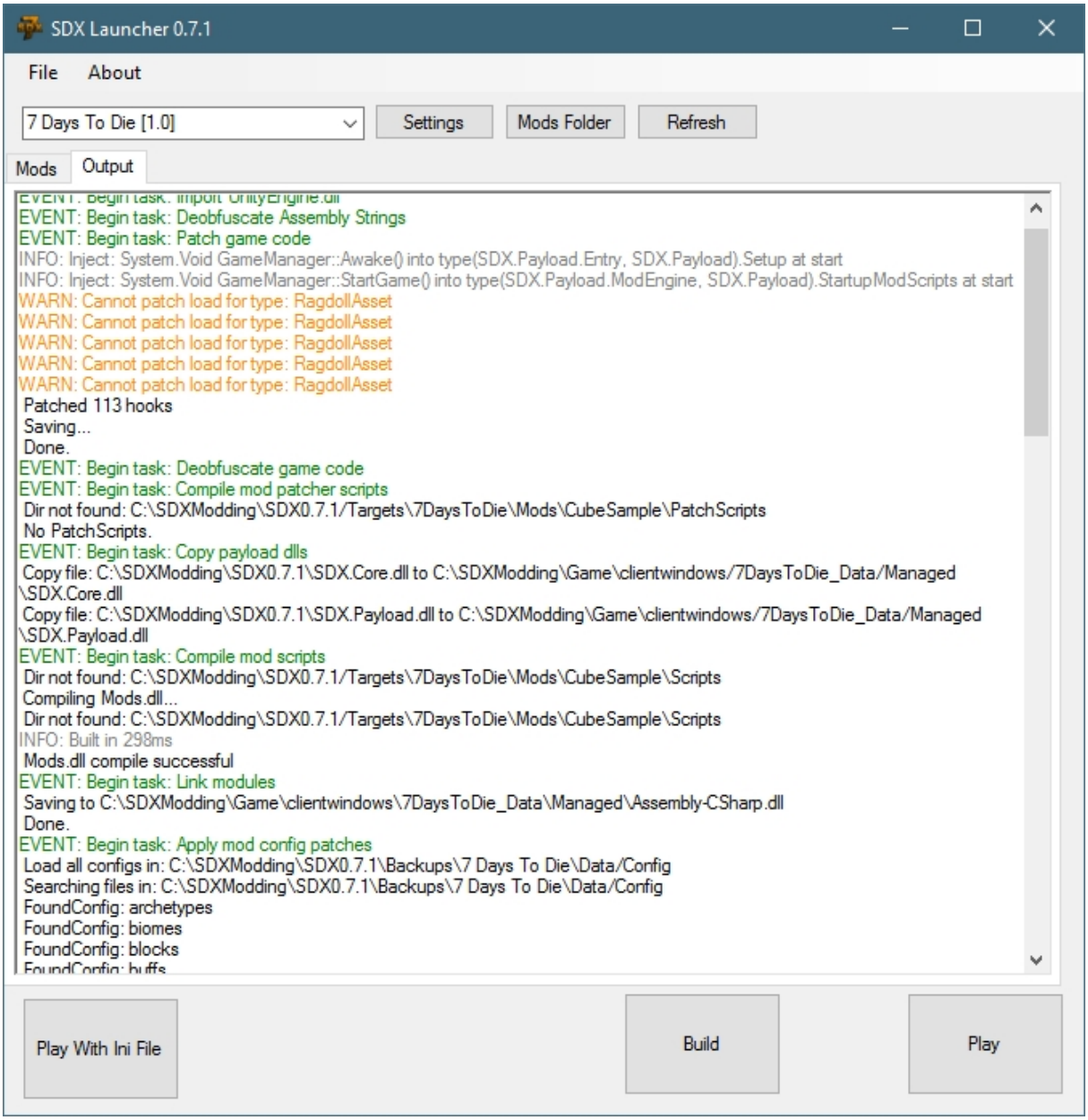
Click on the white square to enable the Cube Sample to enable it.



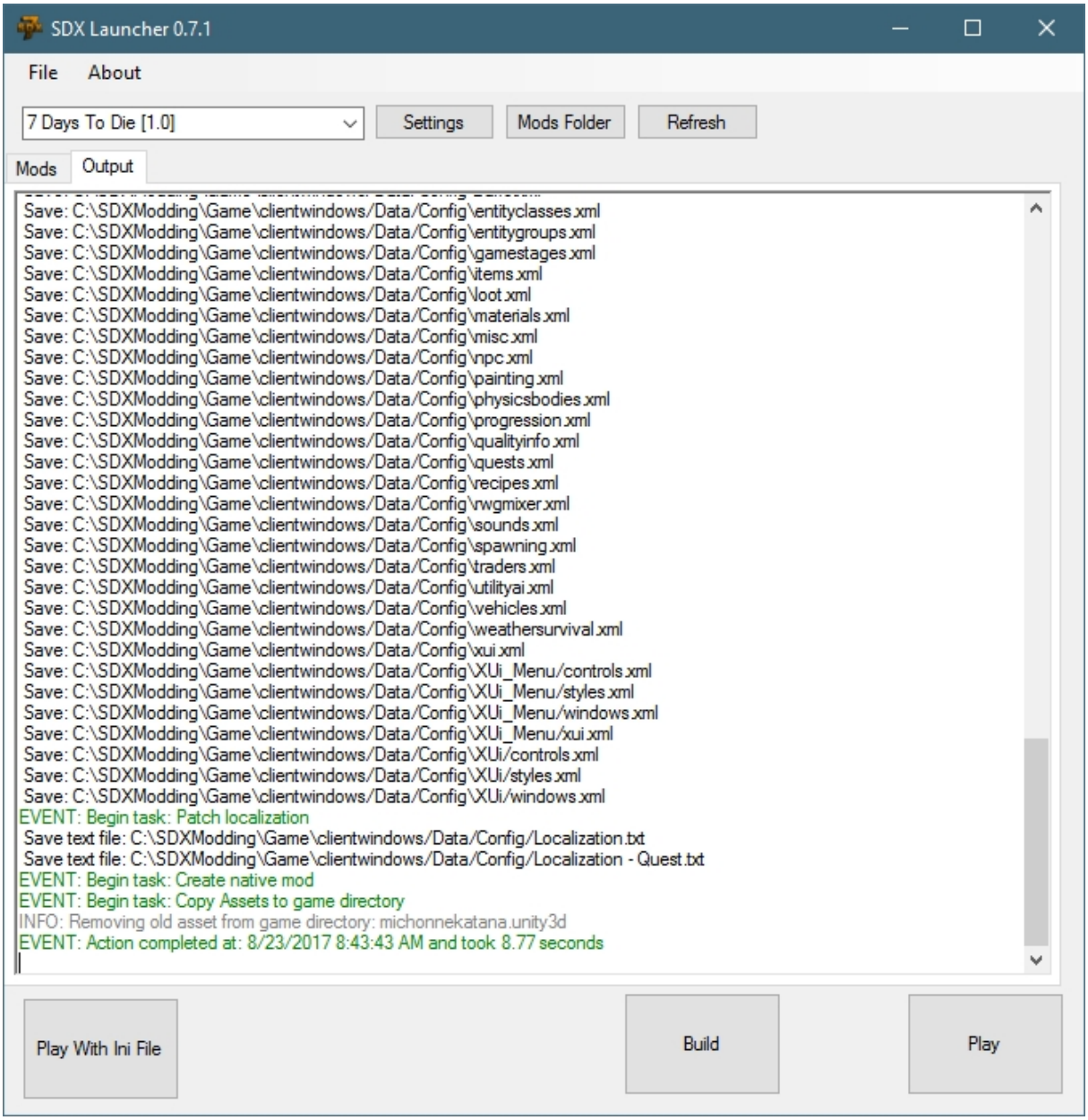

And click on Build.

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.

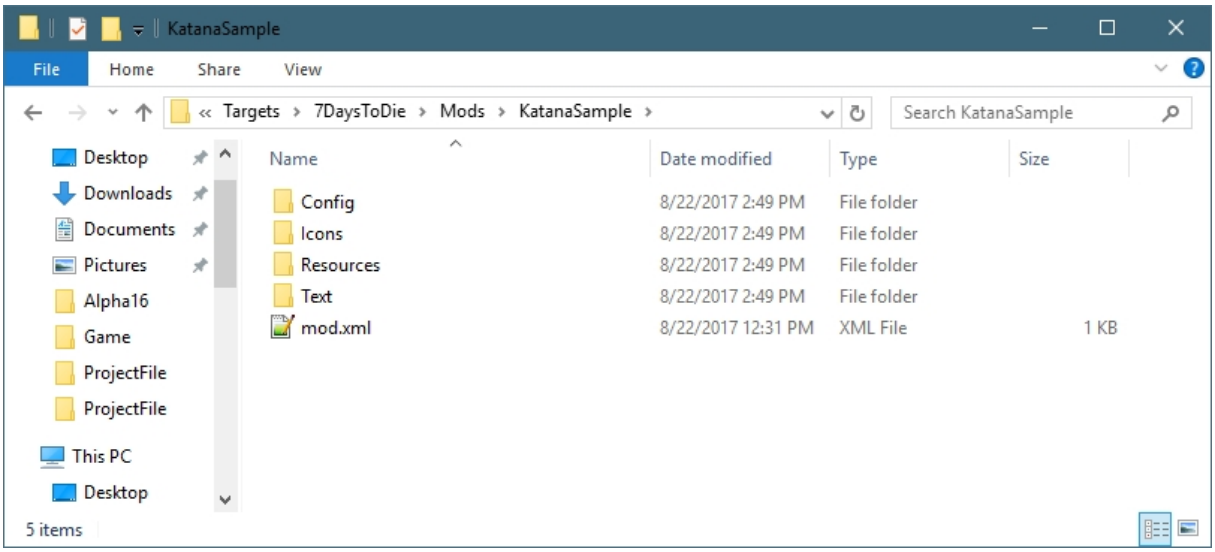

And showing the cube in hand:




## 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.

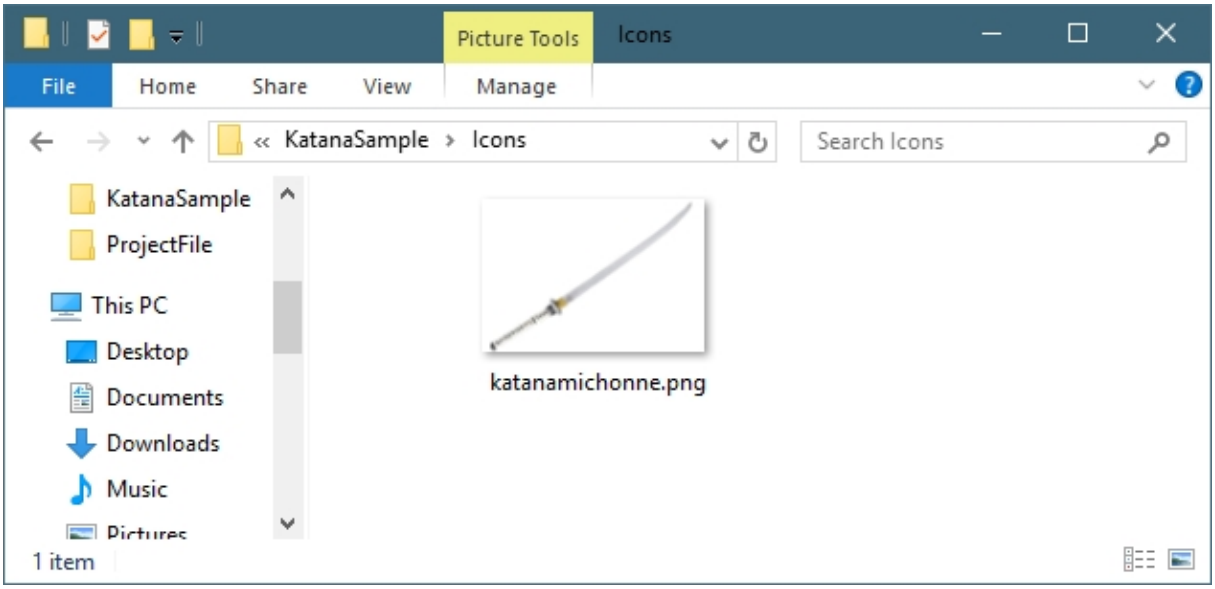
File: KatanaSample\Config\Katana.xml

```
<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*

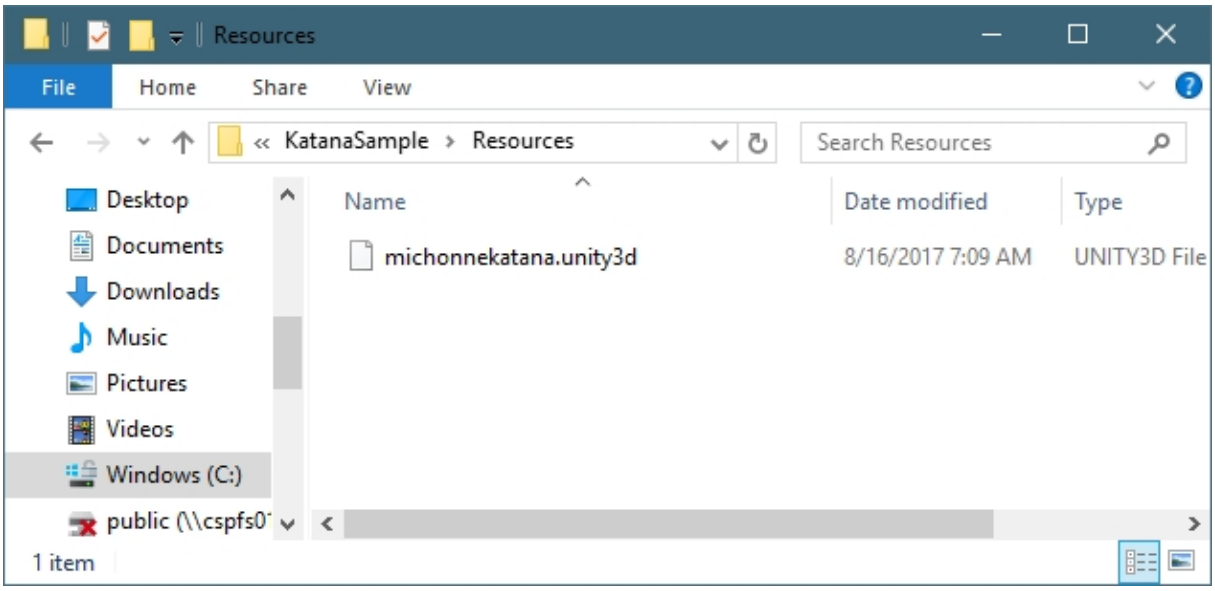
The Icons folder contains the item's inventory icons for the mod.




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

### *Resource Folder*

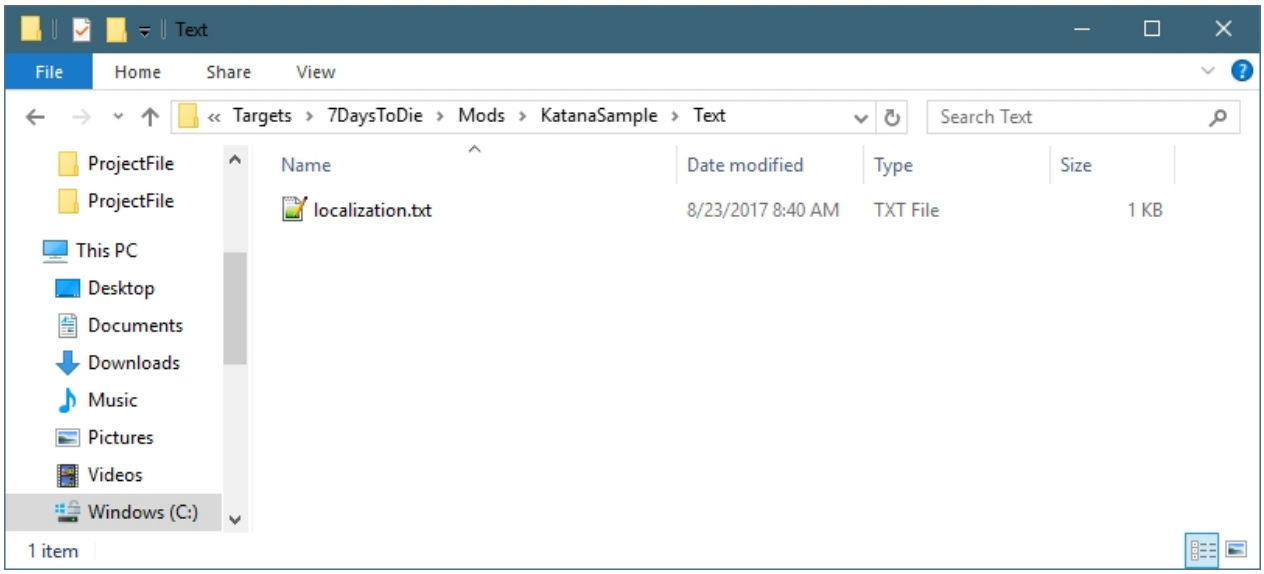
The Resource folder contains the unity3D bundle, which contains the 3D model of the Katana sword.




*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>
```

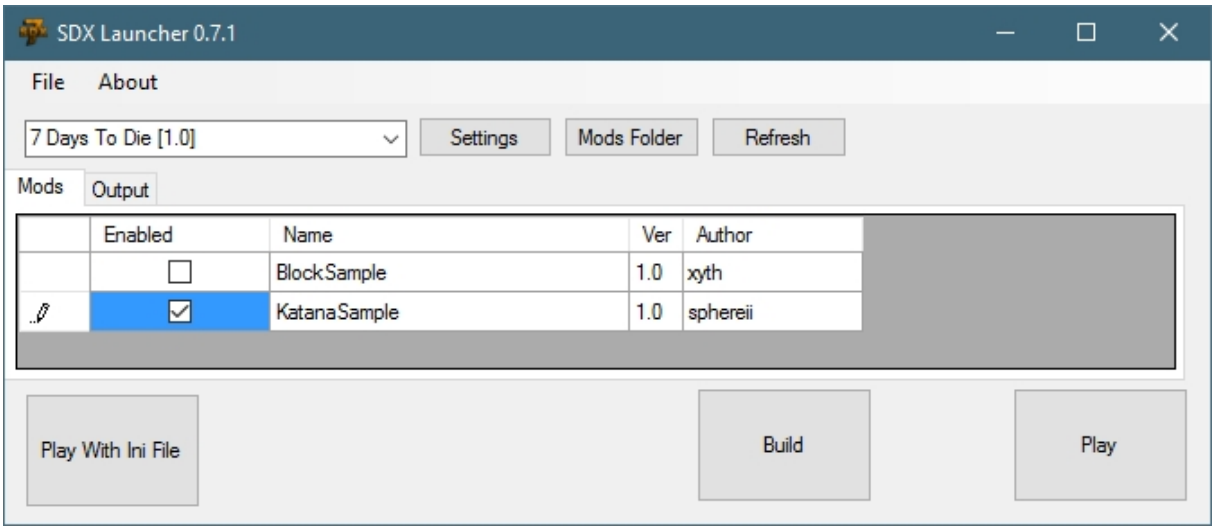
```
      <info>
            <!-- Information about the Mod, the author and version
information -->
            <!-- These are displayed in the SDX Launcher -->
            <author>sphereii</author>
            <name>KatanaSample</name>
            <description>Sample Katana SDX Mod</description>
  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

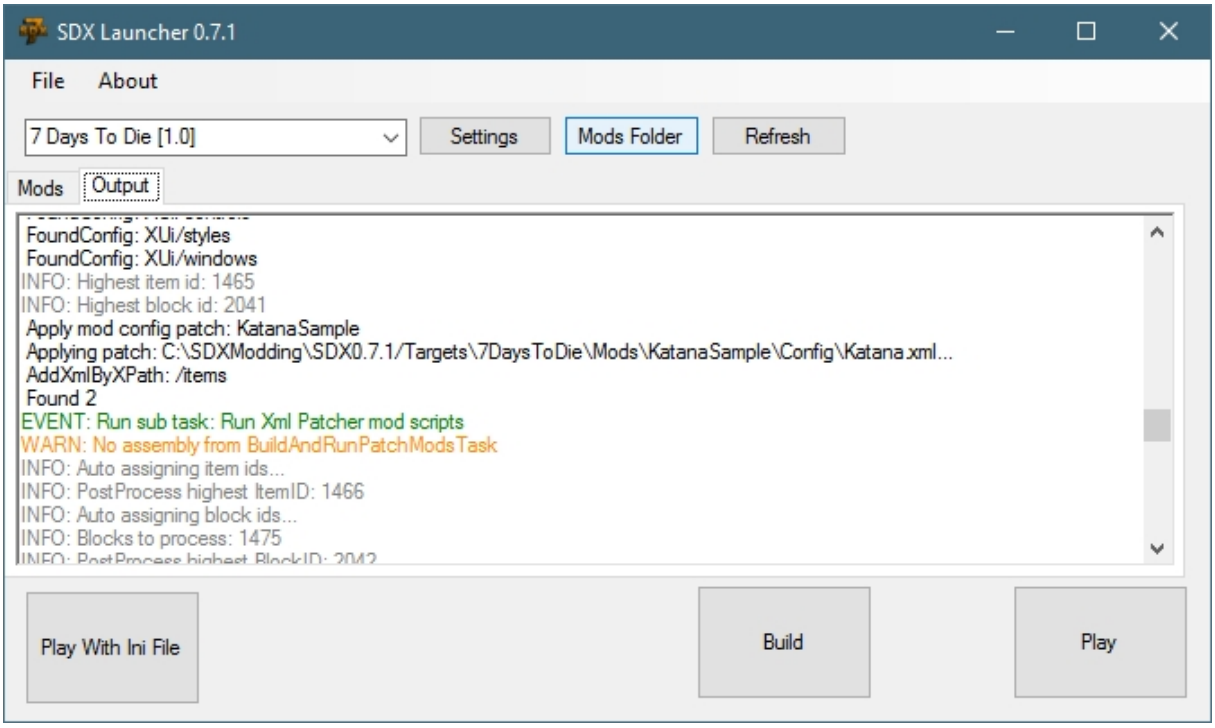



And click on Build.

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

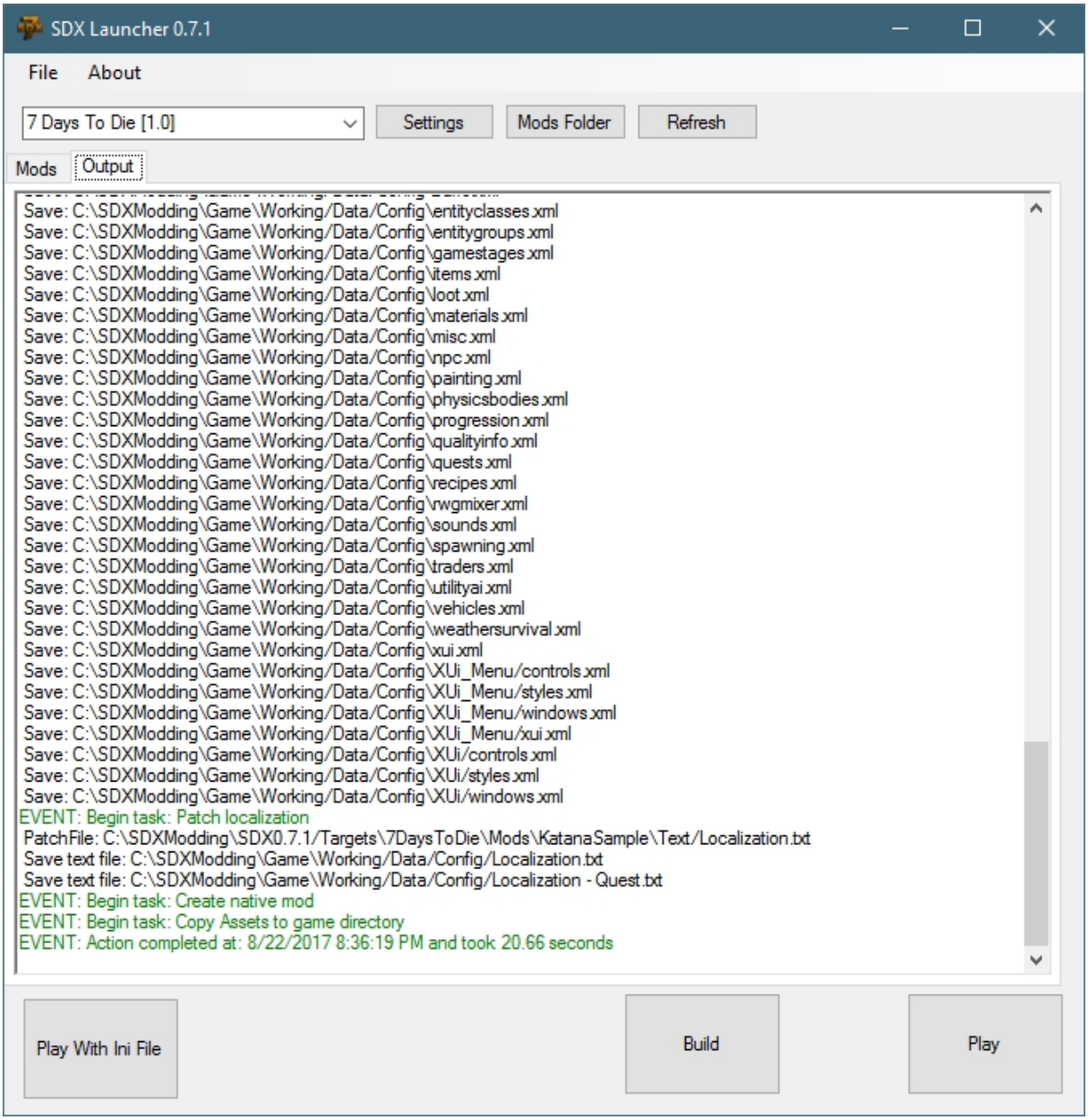
You'll see the SDX compile start, and display the log file as it does its actions.

As it builds, you may see the following error:



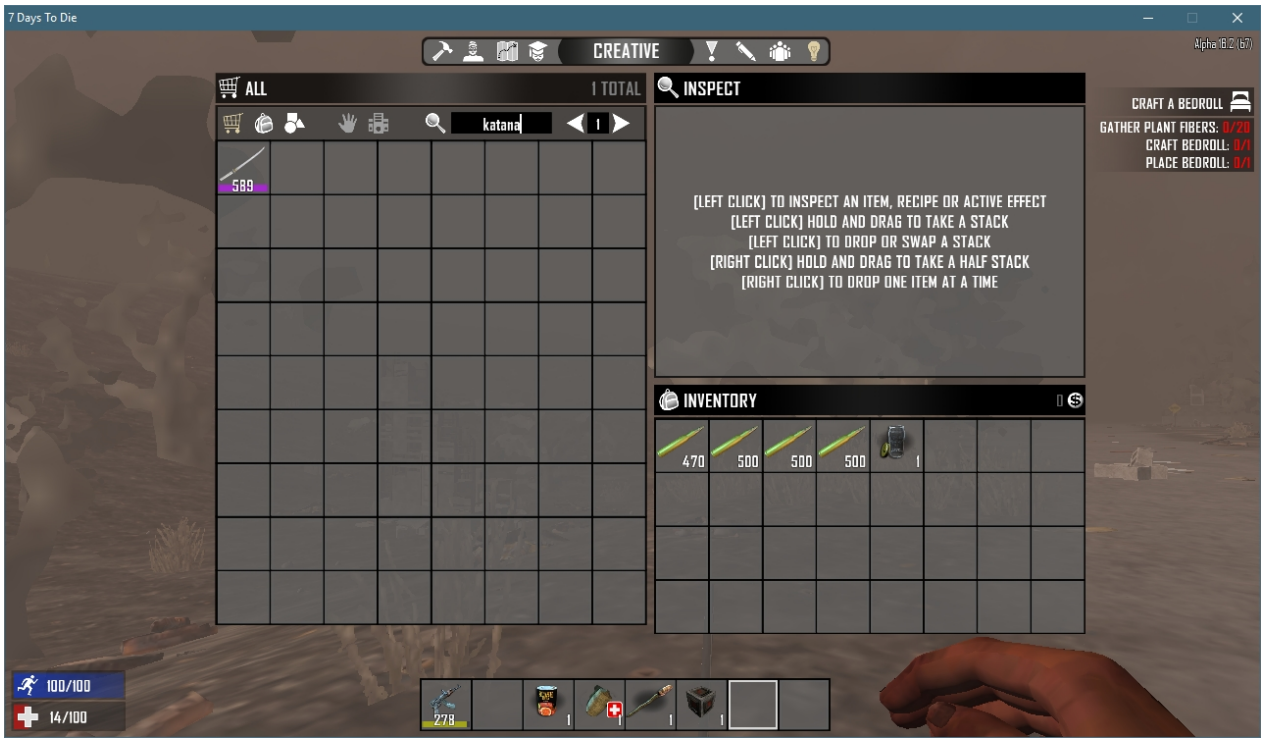

In this exercise, the "WARN: No assembly from BuildAndRunPatchModsTask" is a not a concern, since there is no Mods scripts yet enabled.

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.



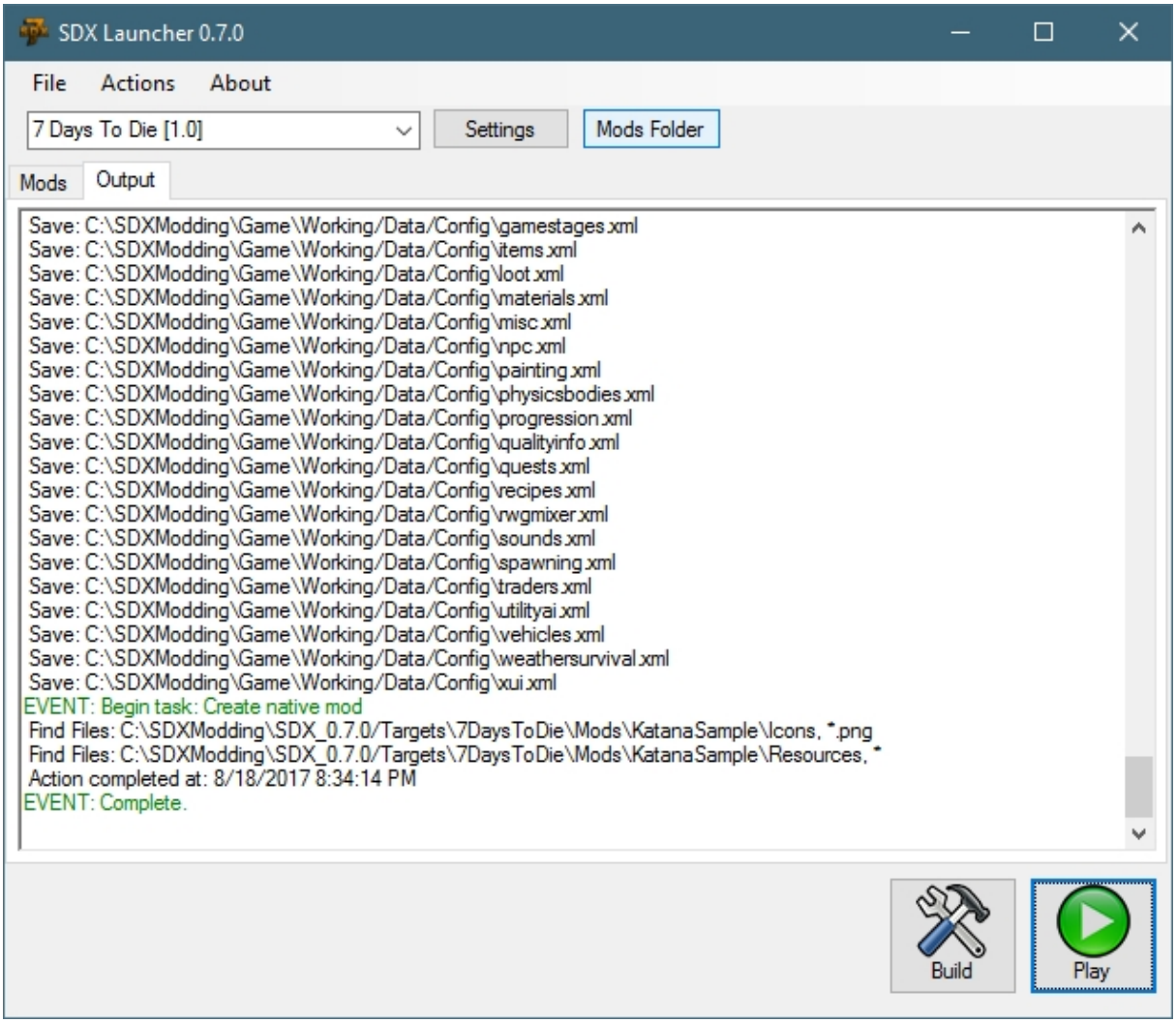

And showing the Katana in hand:



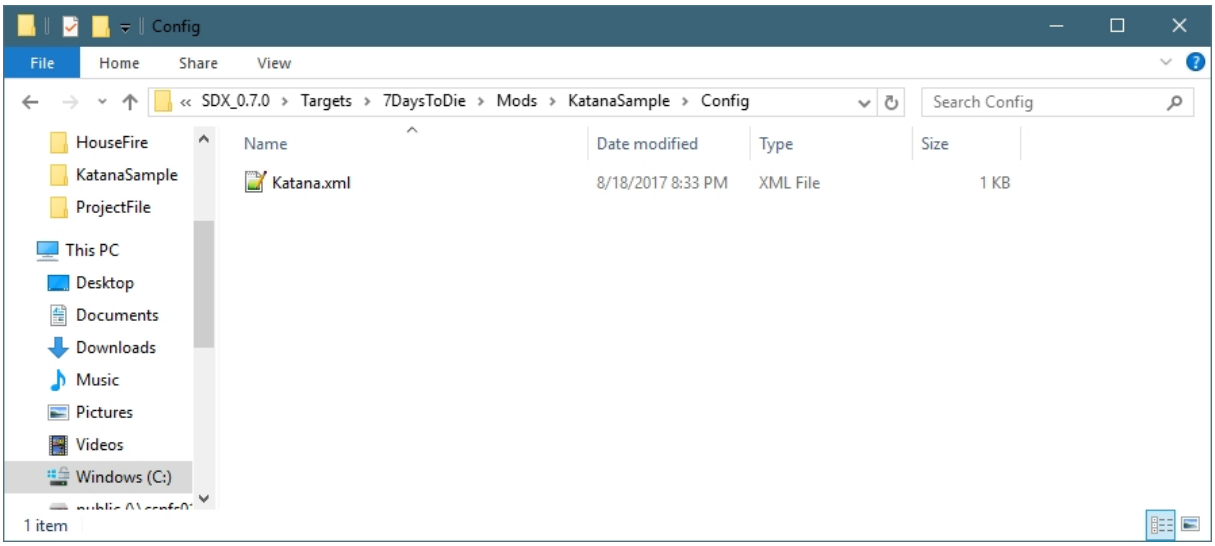

## Adding a Recipe for the Katana Mod

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.

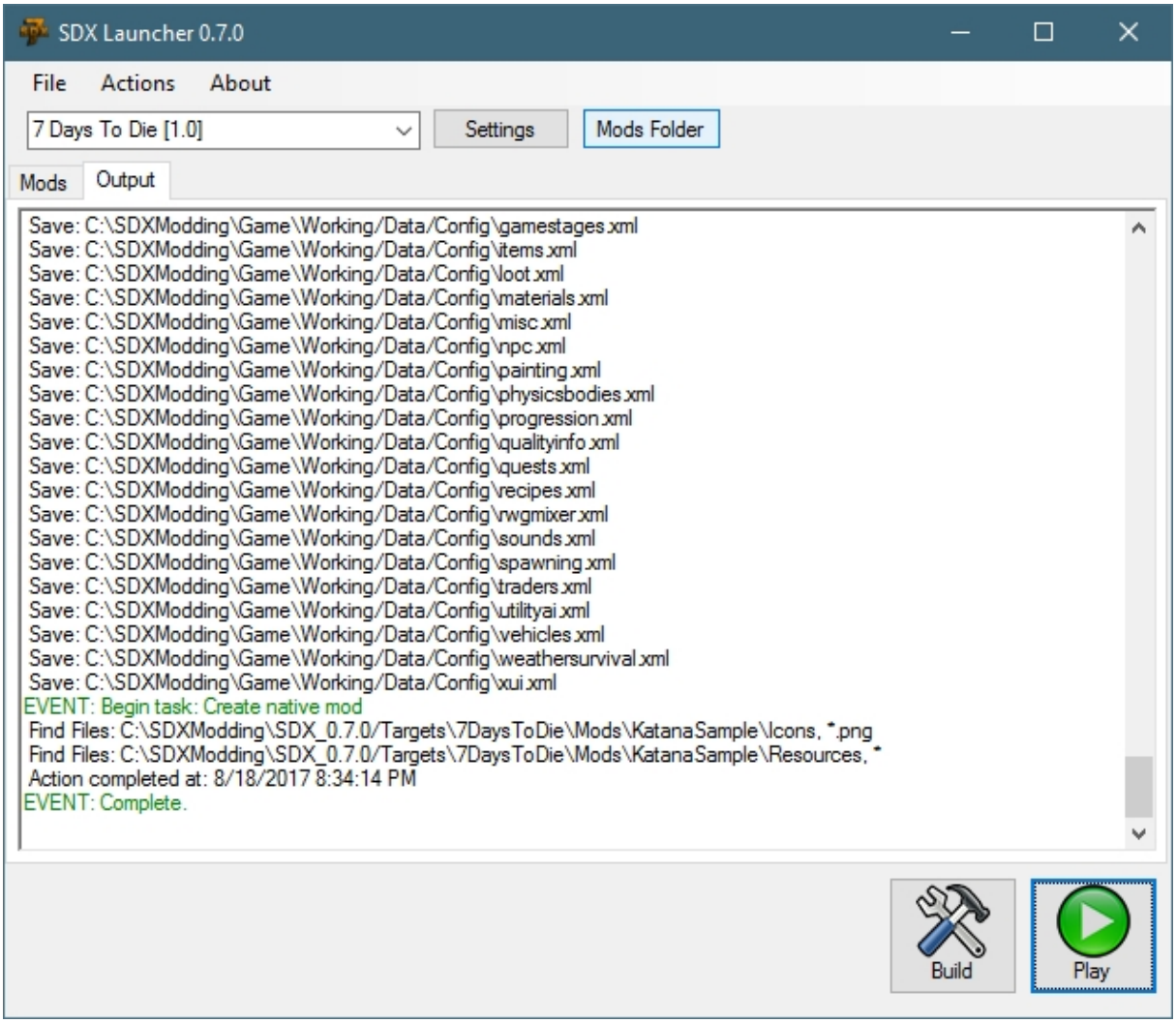
Load up your save game, and search for the Katana sword in your Crafting menu



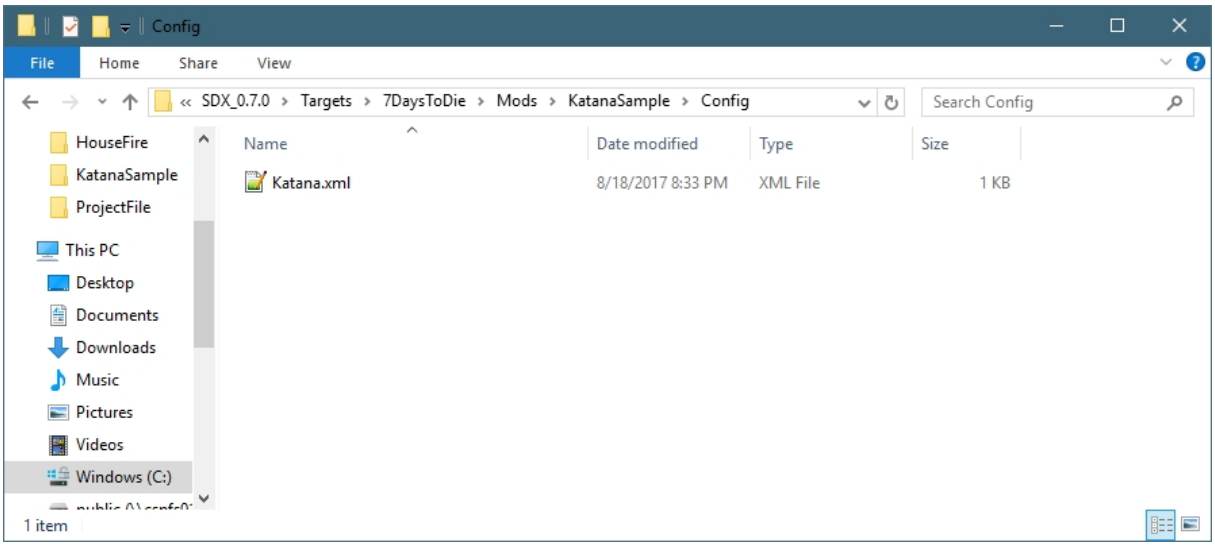

## Adding Katana to a Loot Group

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.

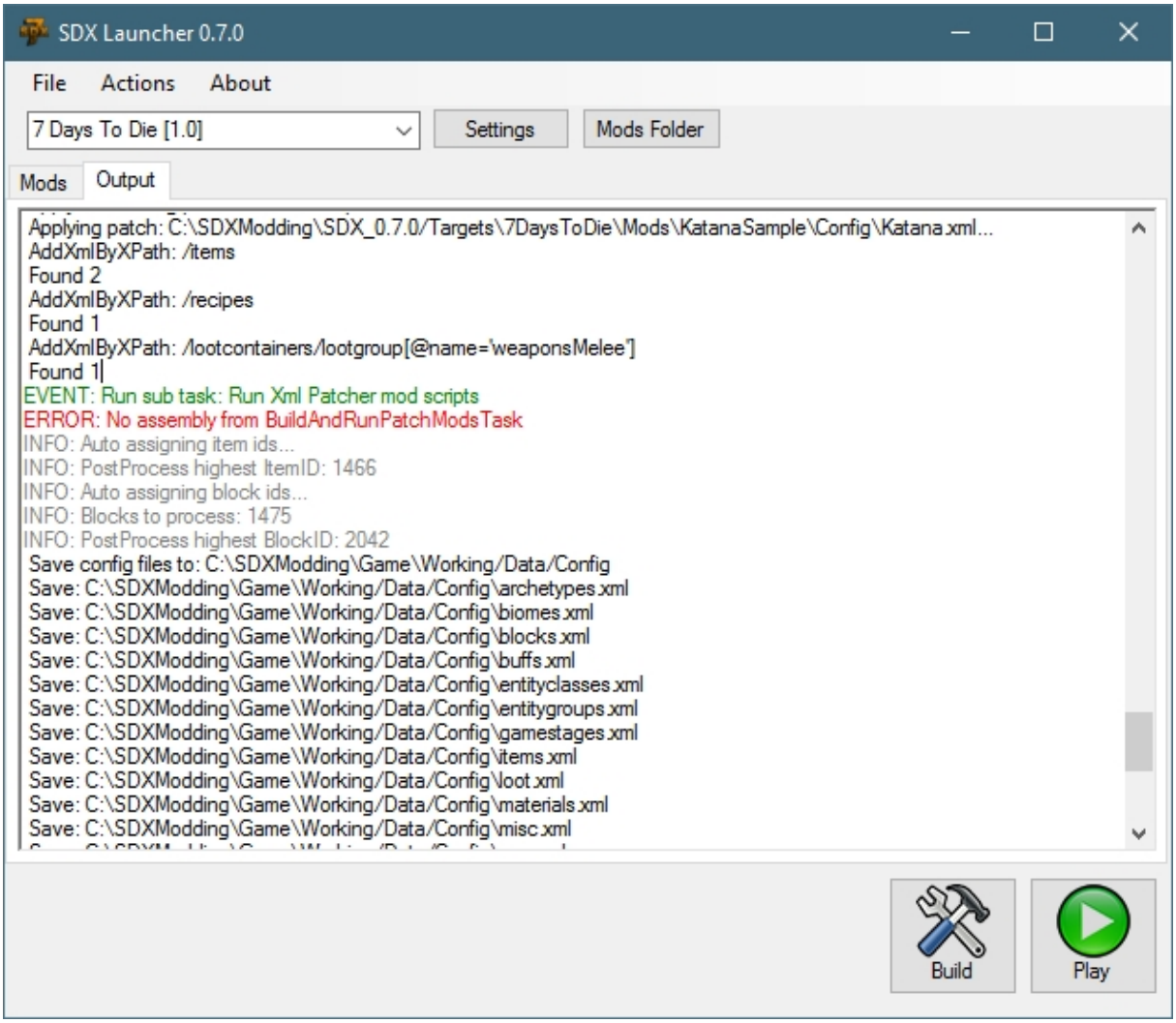
The Katana.xml should look like this now:

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>
```

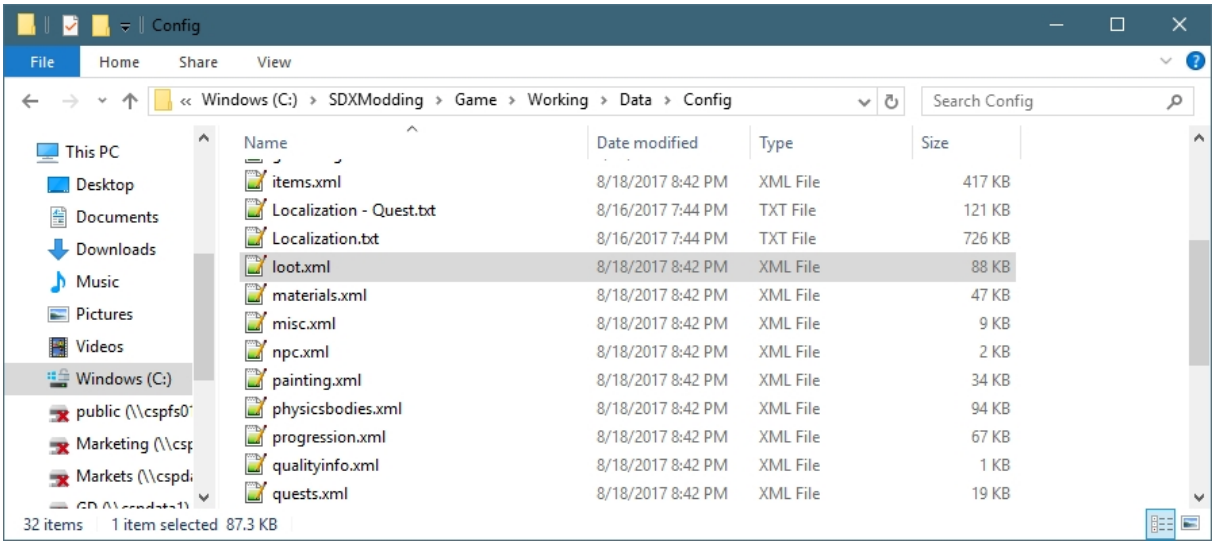
Once you've saved your changes, go back to the SDX Launcher, and click on the Build Button.

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



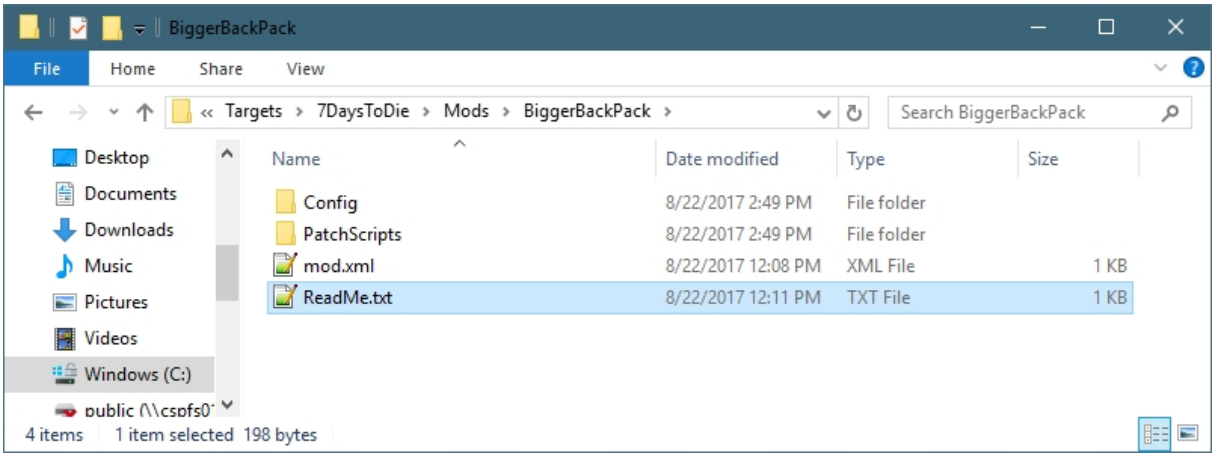
## Adding the Bigger Back Pack Mod


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 and install 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.

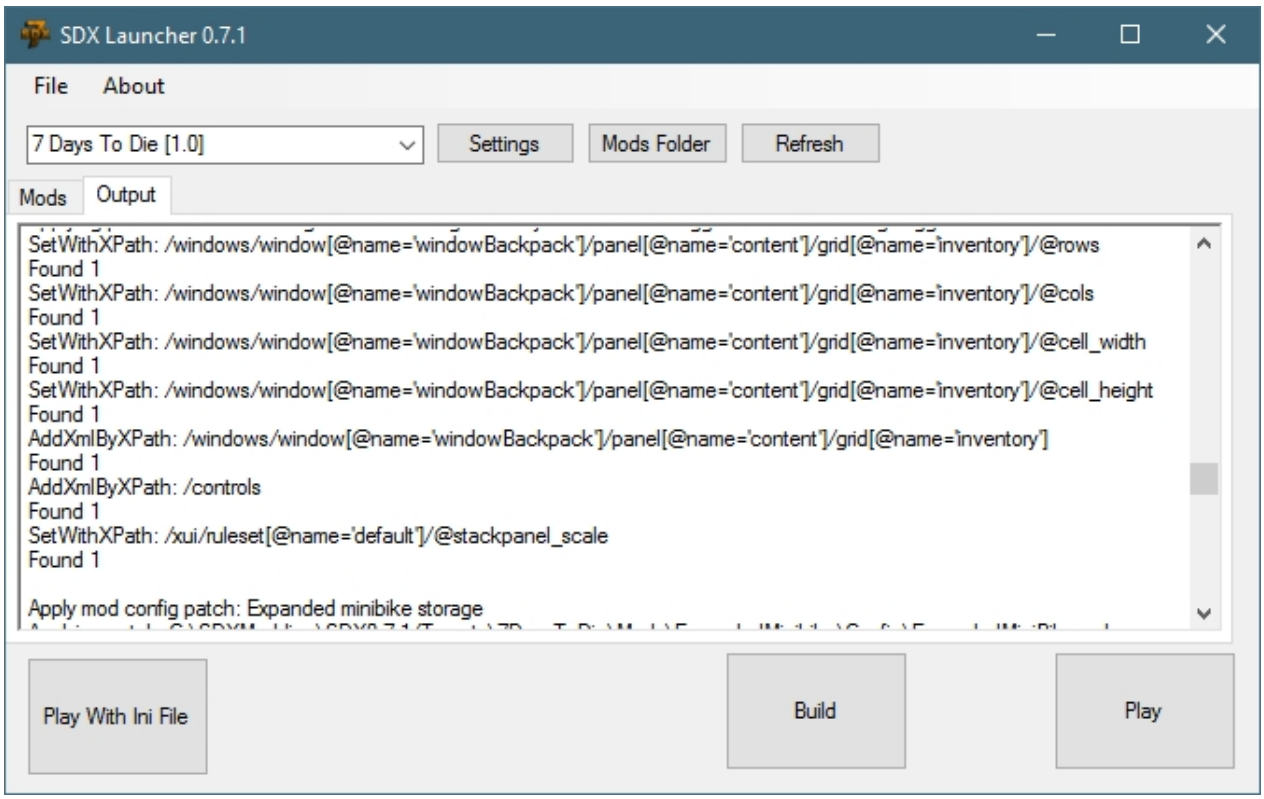
Let's break it down:

```
<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:

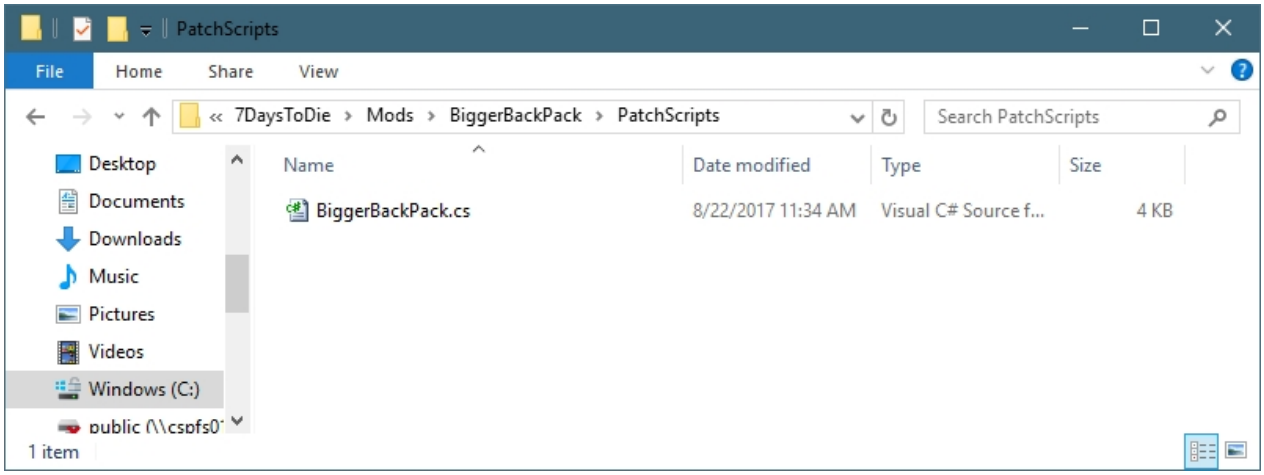



Set With XPath indicates what value it's looking for, and how many matching lines it found.

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 ruing 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#.***



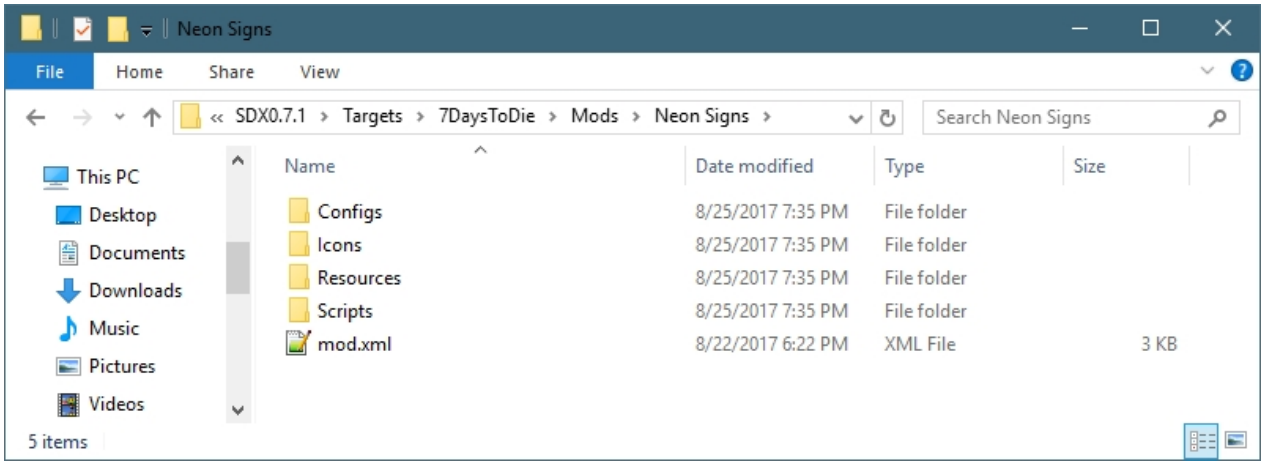
## Neon Light Mod

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 Light 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.

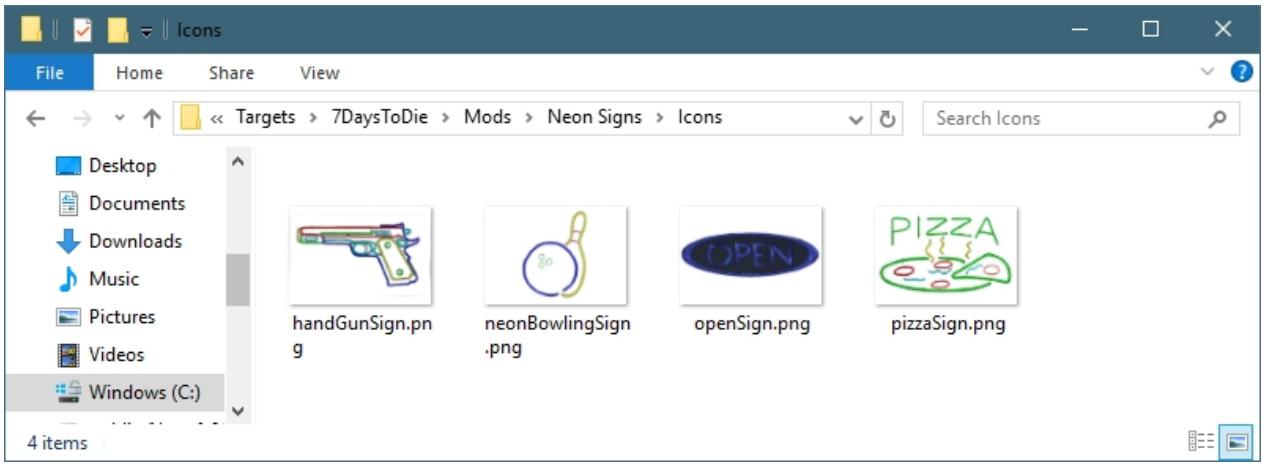
```
<configs>
      <config name="blocks">
            <append xpath="/blocks">
                  <block id="1630" name="Neon Open Sign">
                        <property name="Extends"
value="ceilingLight02_player" />
                        <property name="CustomIcon" value="openSign" />
                        <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.

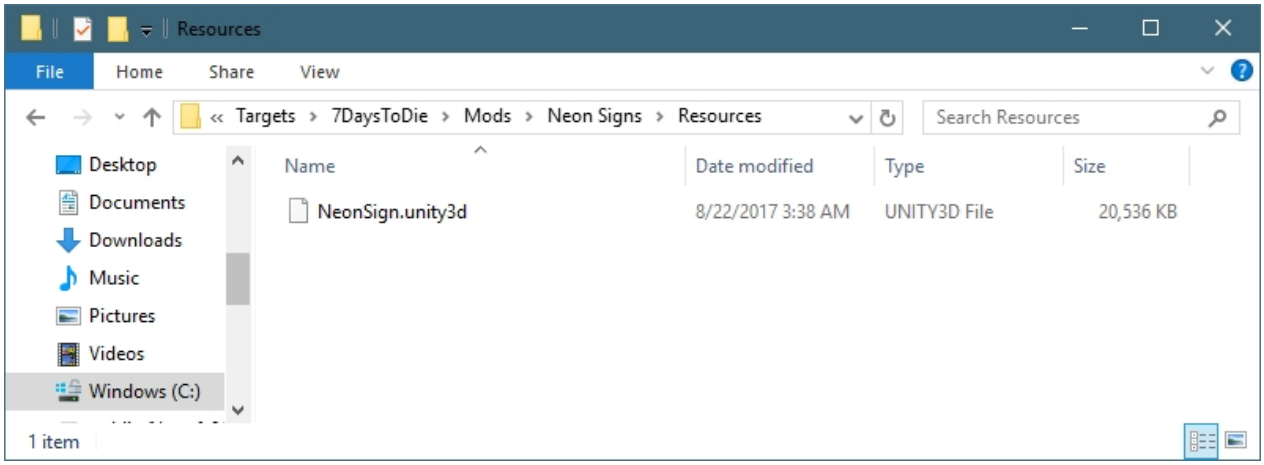



So three08 added 4 new custom icons for the game, with only one unity3d file (shown below). In our previous examples, there's was usually just one model in each unity3d file. However, that doesn't mean that there can't be more. A unity3d bundle can have many custom models embedded in it, being referenced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

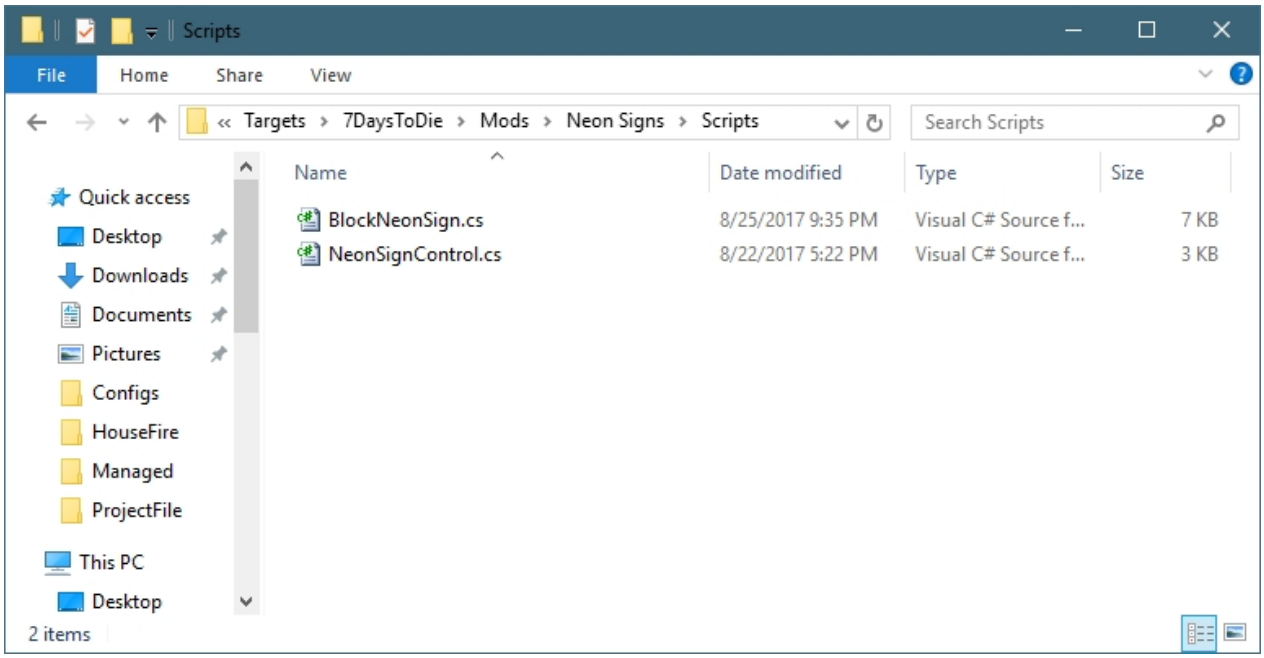
If you look at the Neon Sign XML file, you'll see the following references. We only copied the relevant lines, rather then entire blurb.

```
<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.



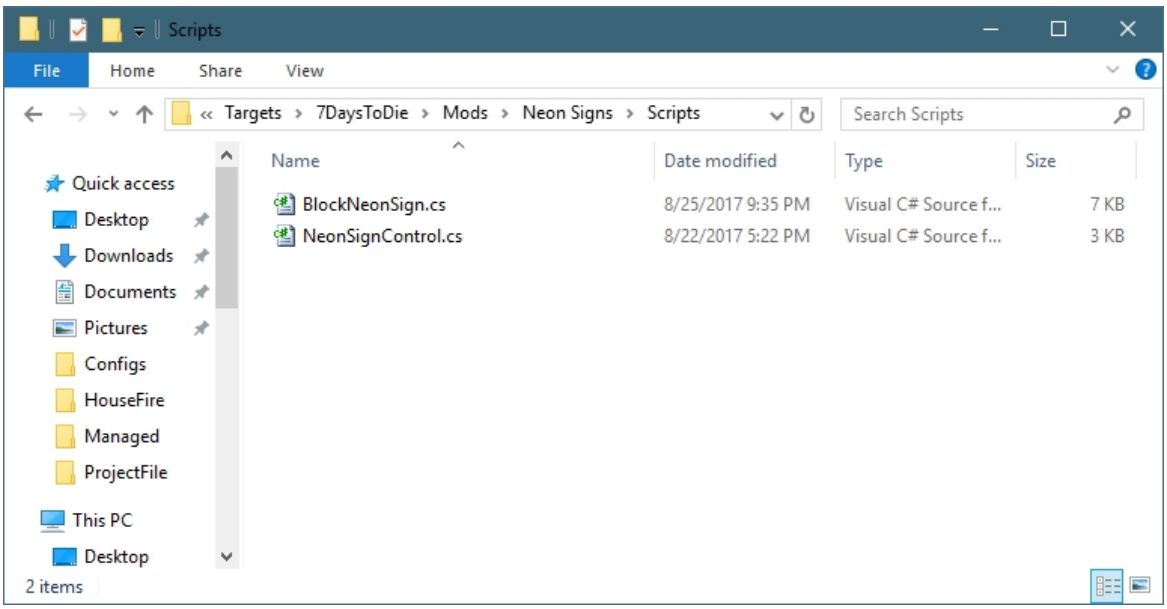

Let's explore on what these files are, and how they are different than the PatchScripts in the Bigger Back Pack Mod.



## 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.

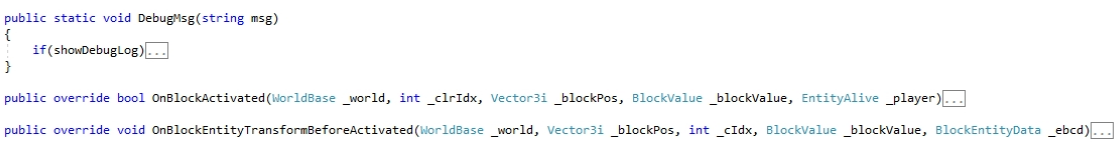
```
using System;
using UnityEngine;

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.

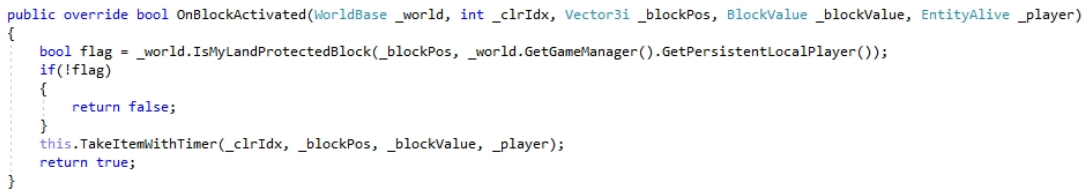
Because BlockNeonSign is derived from the BlockPowered class, it has all the abilities of the base class of BlockPowered. By making a *derived* class, three08 is saying "I want to do everything that the BlockPowered class does, but I want to make changes to some things'.




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.



## 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.

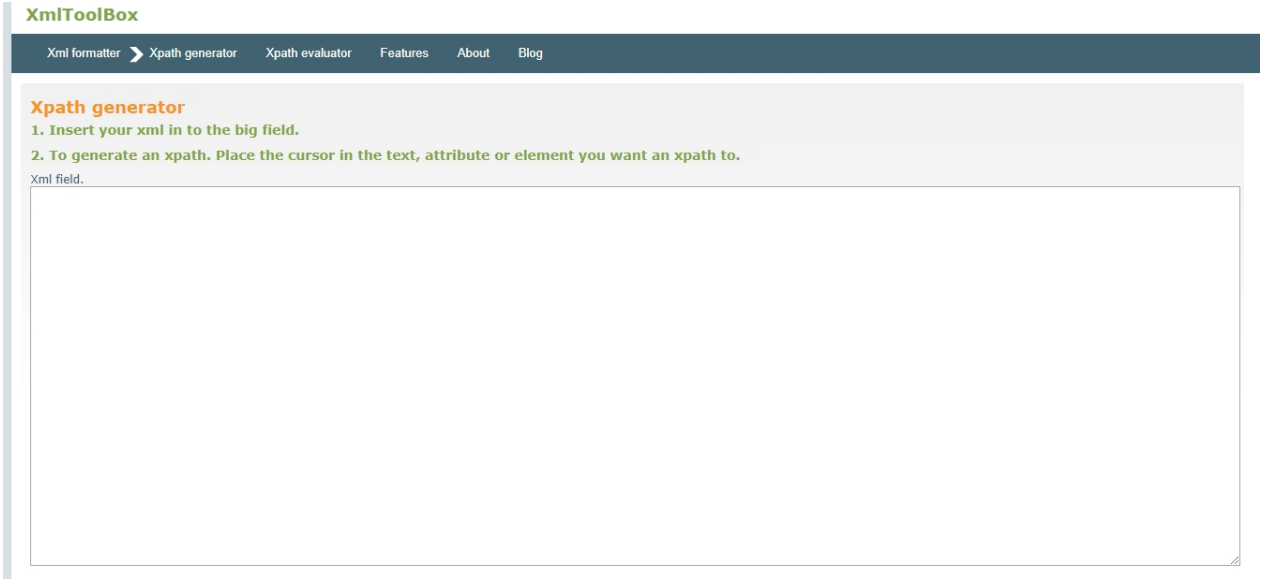


## Creating an XPath Line


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.

## Unity

xyth has created some great tutorials, and they can be found here

## 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

# 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

### *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.

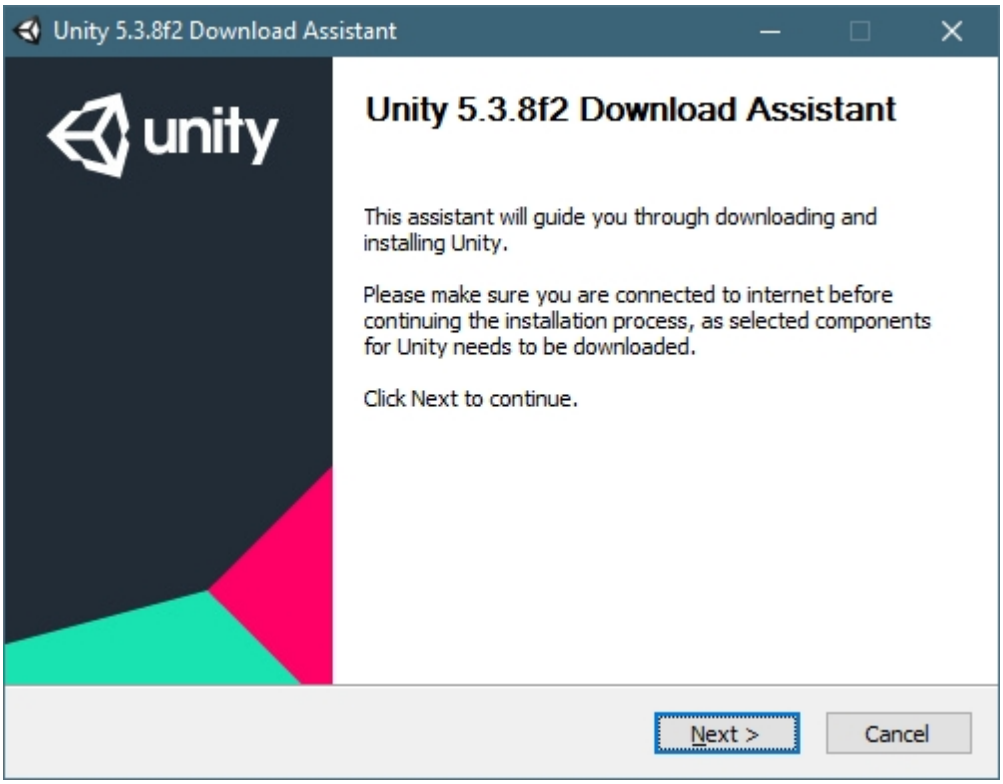


### Installing Unity 5.3.8p2

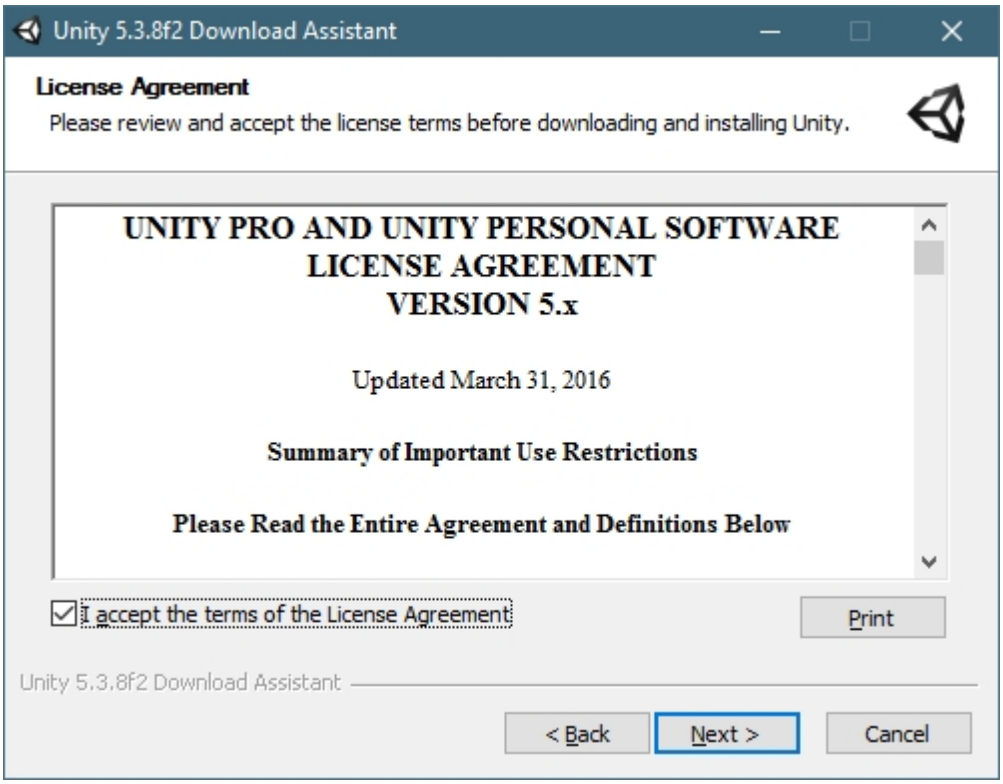
## Unity 5.3.8p2

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.

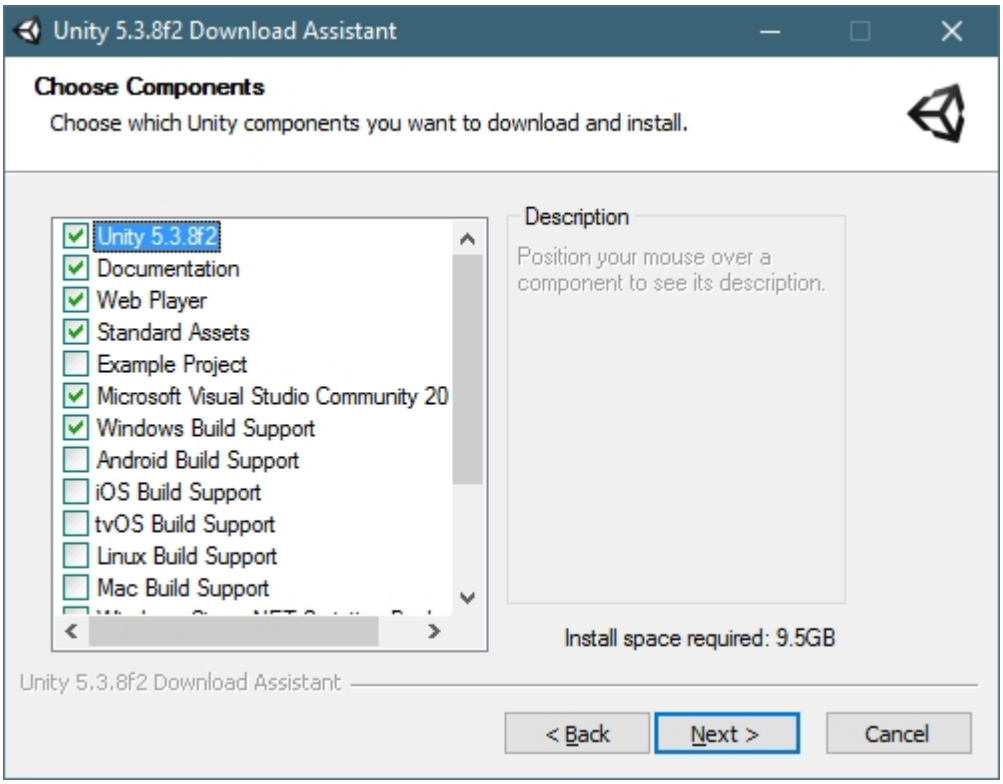
Once downloaded, double click on the UnityDownloadAssistant-5.3.8f2.exe



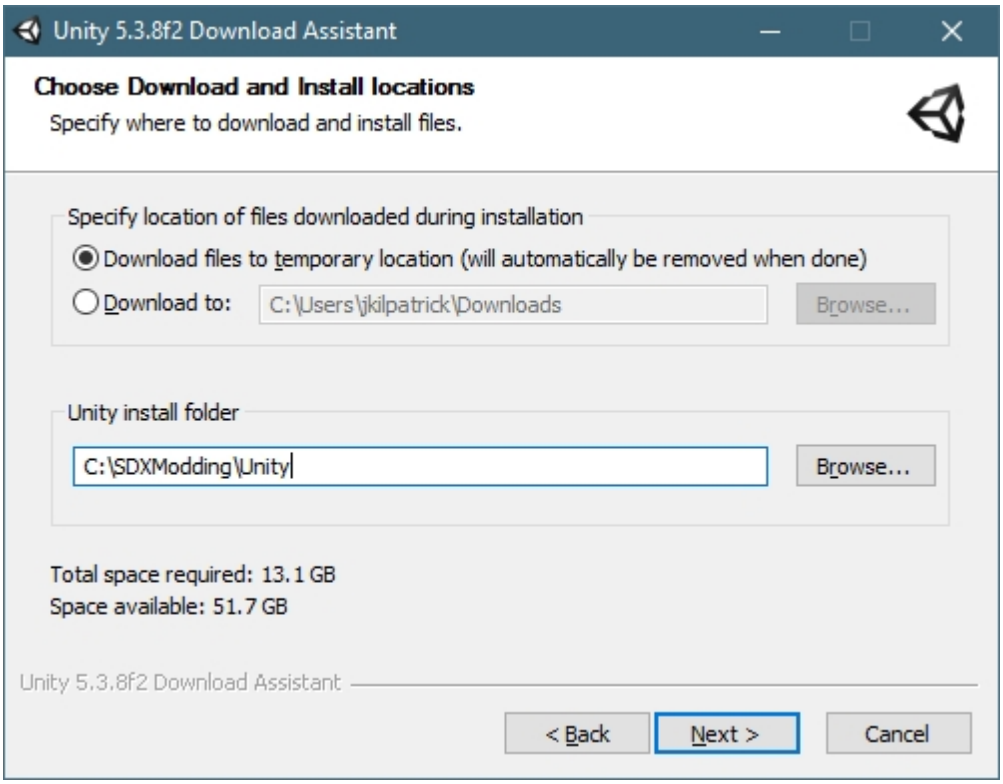

Accept the License Agreement



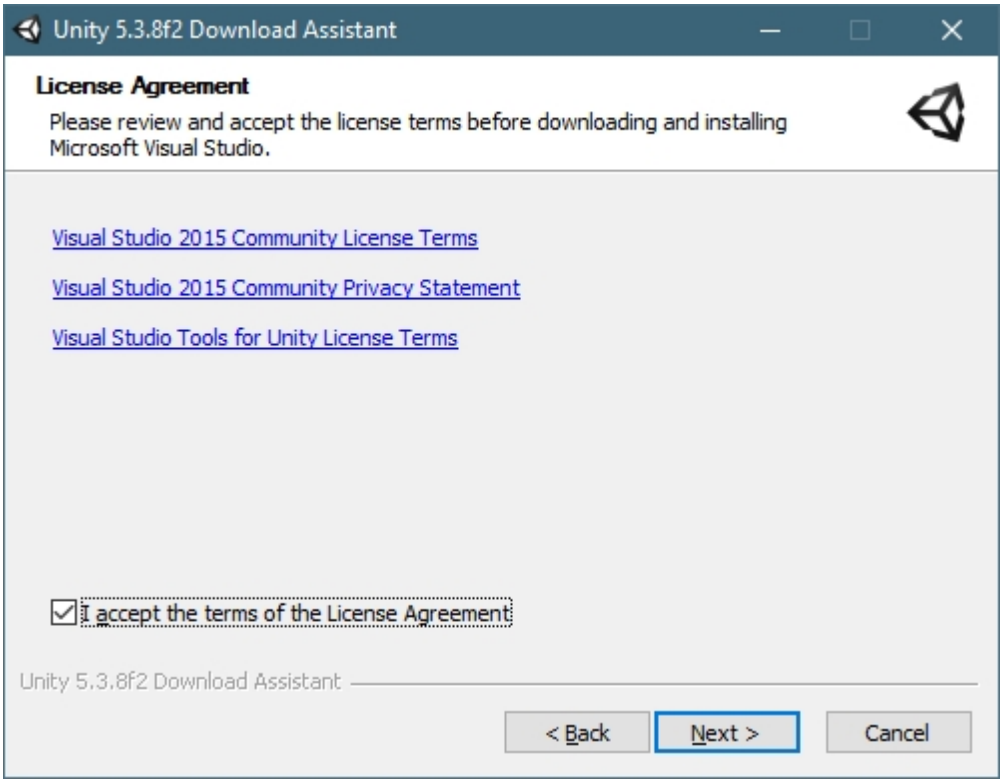

Accept Defaults:



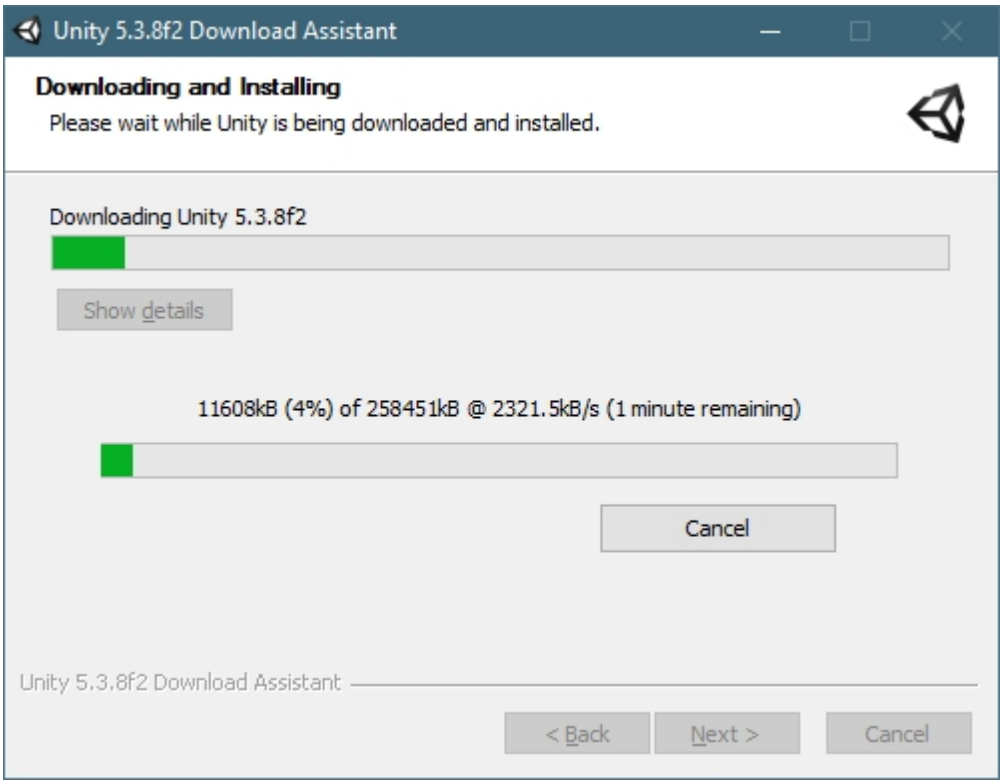

Select the Unity Install location, either as default, of under your C:\SDXModding\ folder.

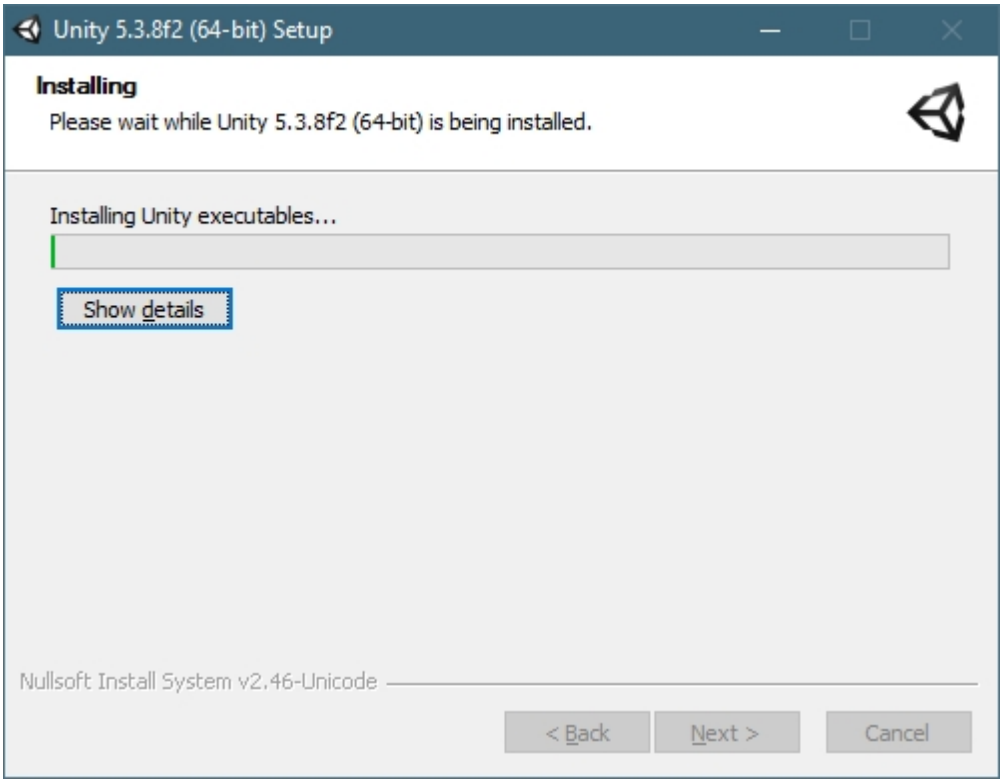


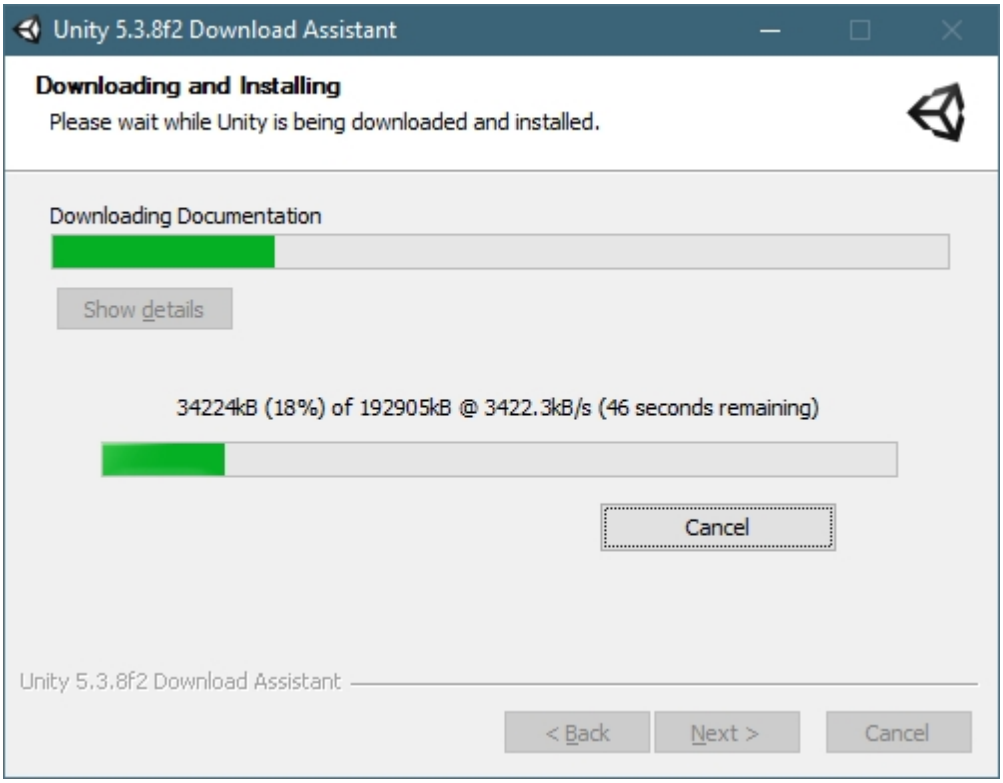

Accept the License Agreement by checking on the check box, once you have reviewed and accepted the licenses



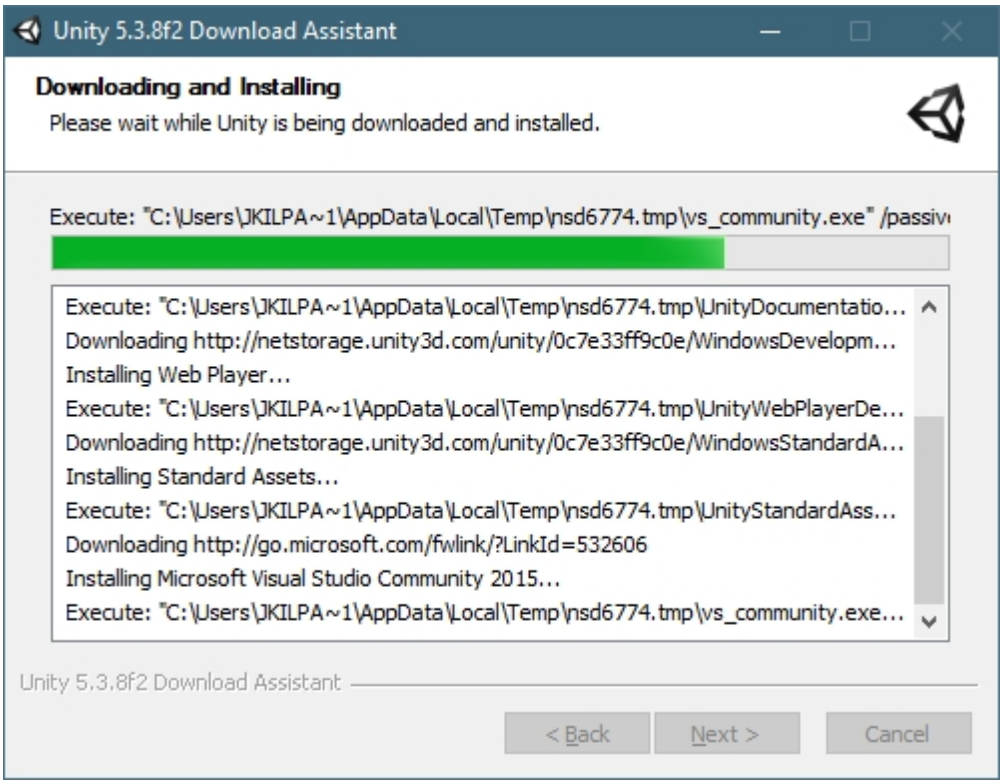

Wait for the automated download and install to be completed



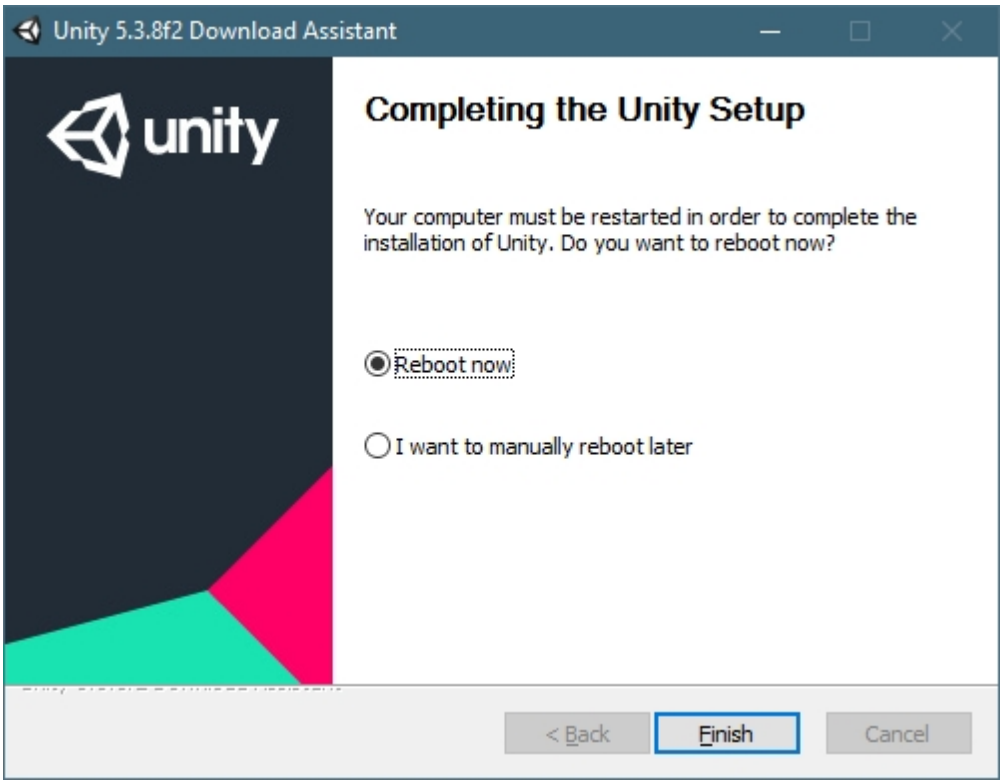



During the installation process, other tools will be installed as needed, such as Visual Studio 2015 Community Edition.



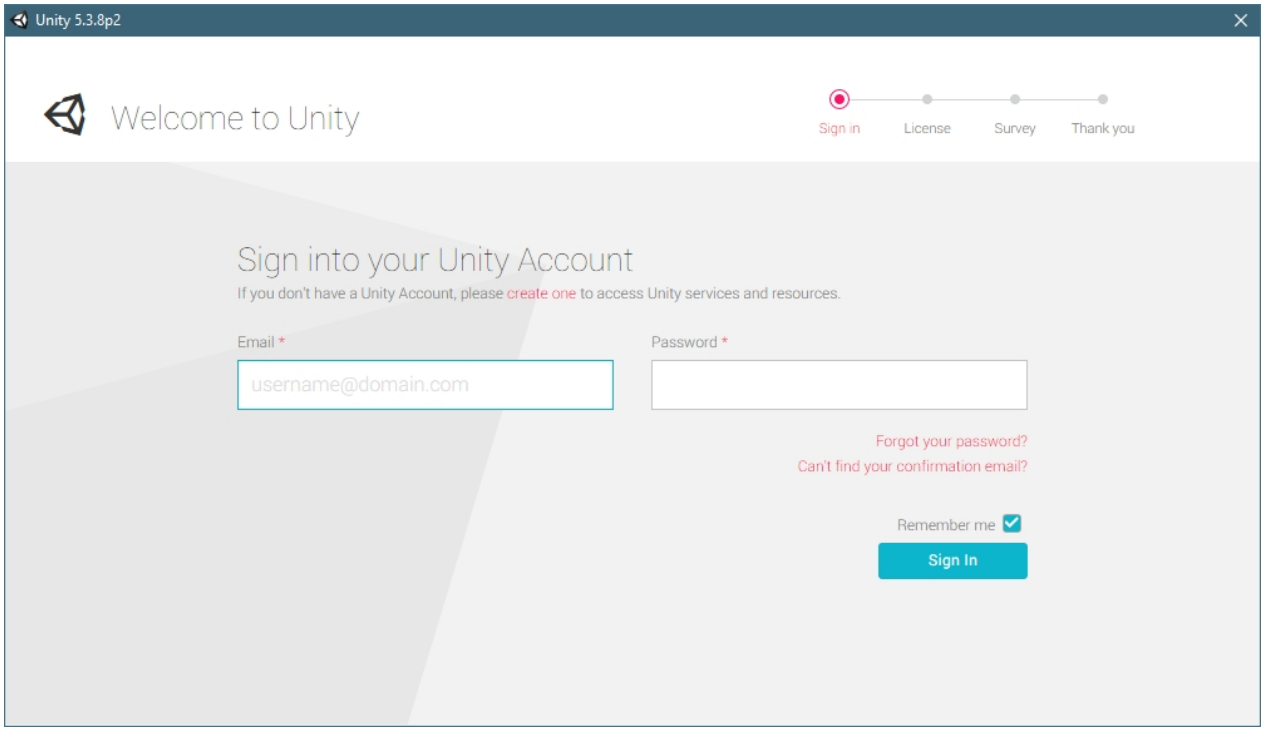

Reboot when done.



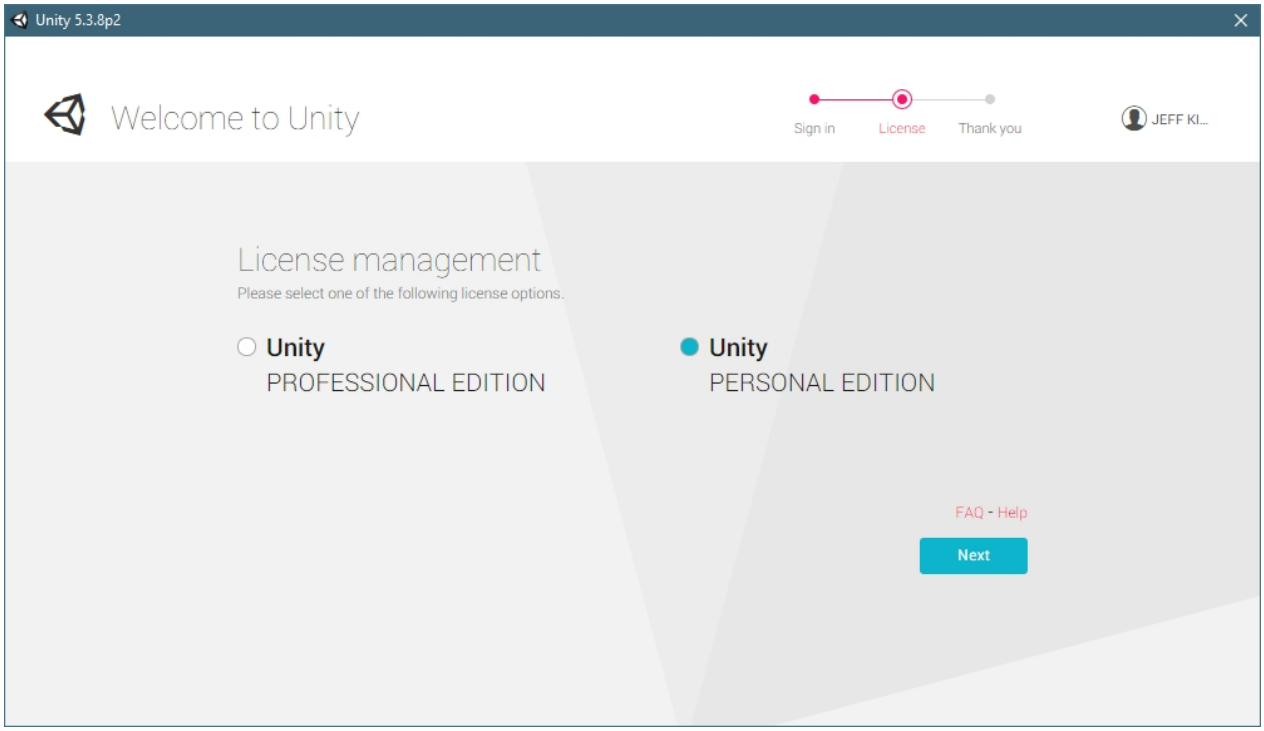



## Starting Unity for the first time

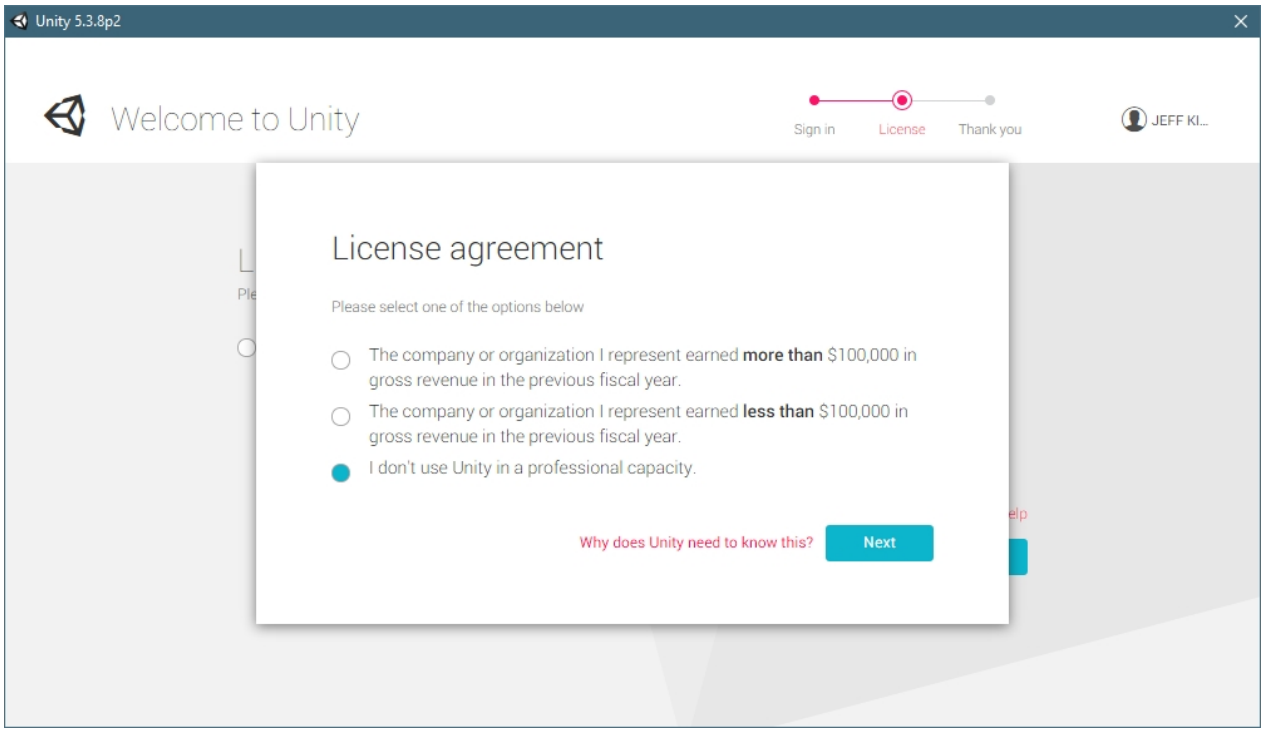
When Unity is started, it will ask you to either login, or create a new account. This will allow you to download assets from the Unity store.



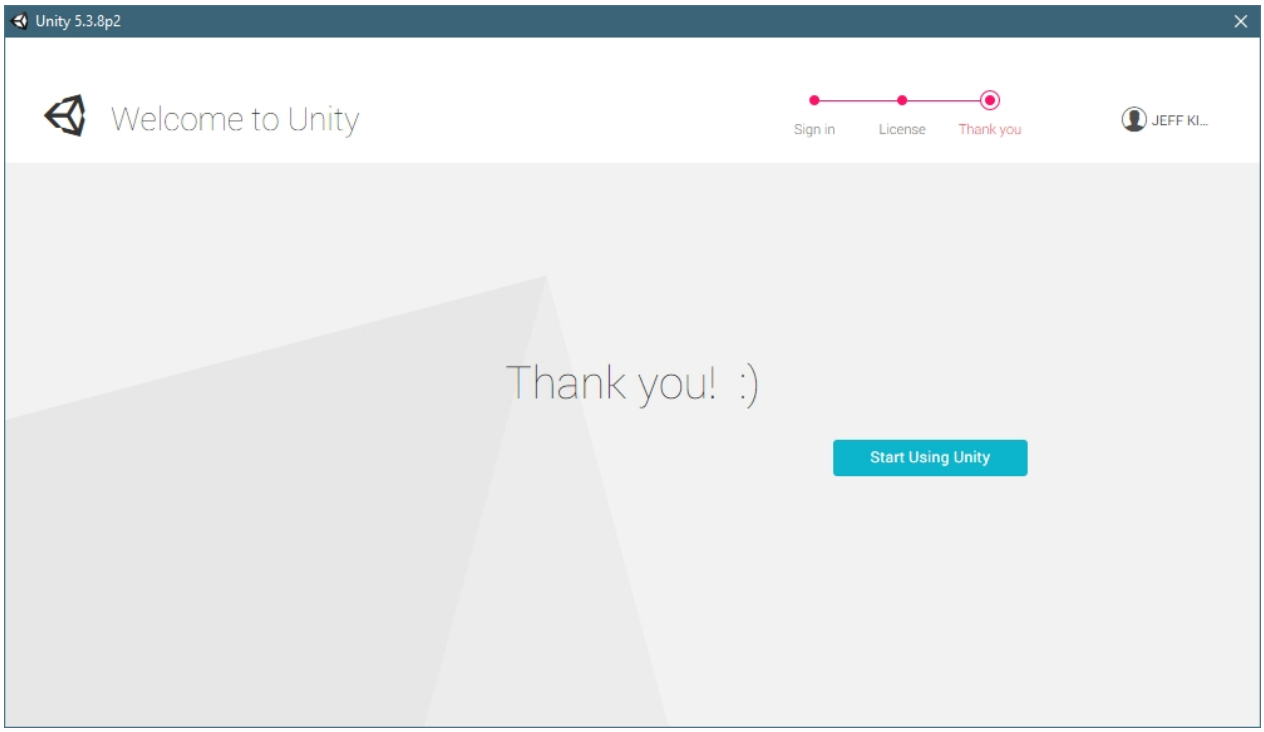

Unity offers two options, Professional Editior of Personal Edition. For this tutorial, we'll select Personal Edition.




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



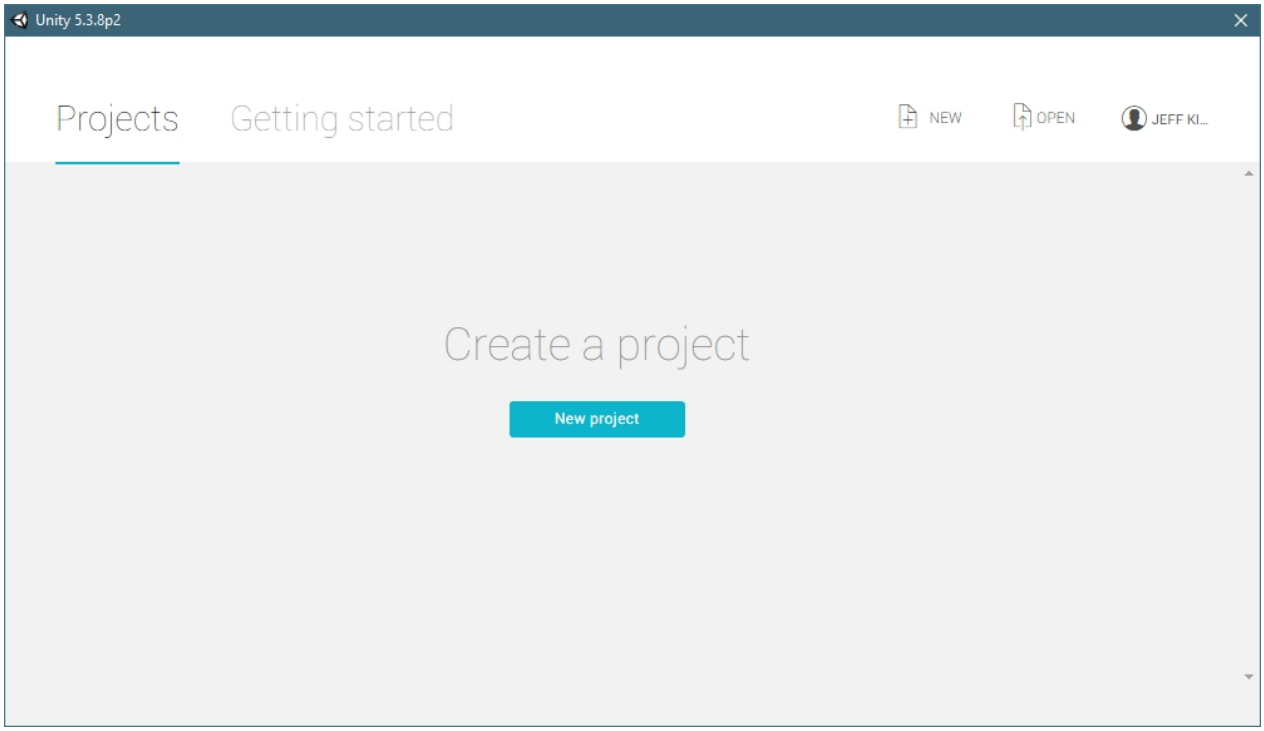



## Creating a new Unity Project

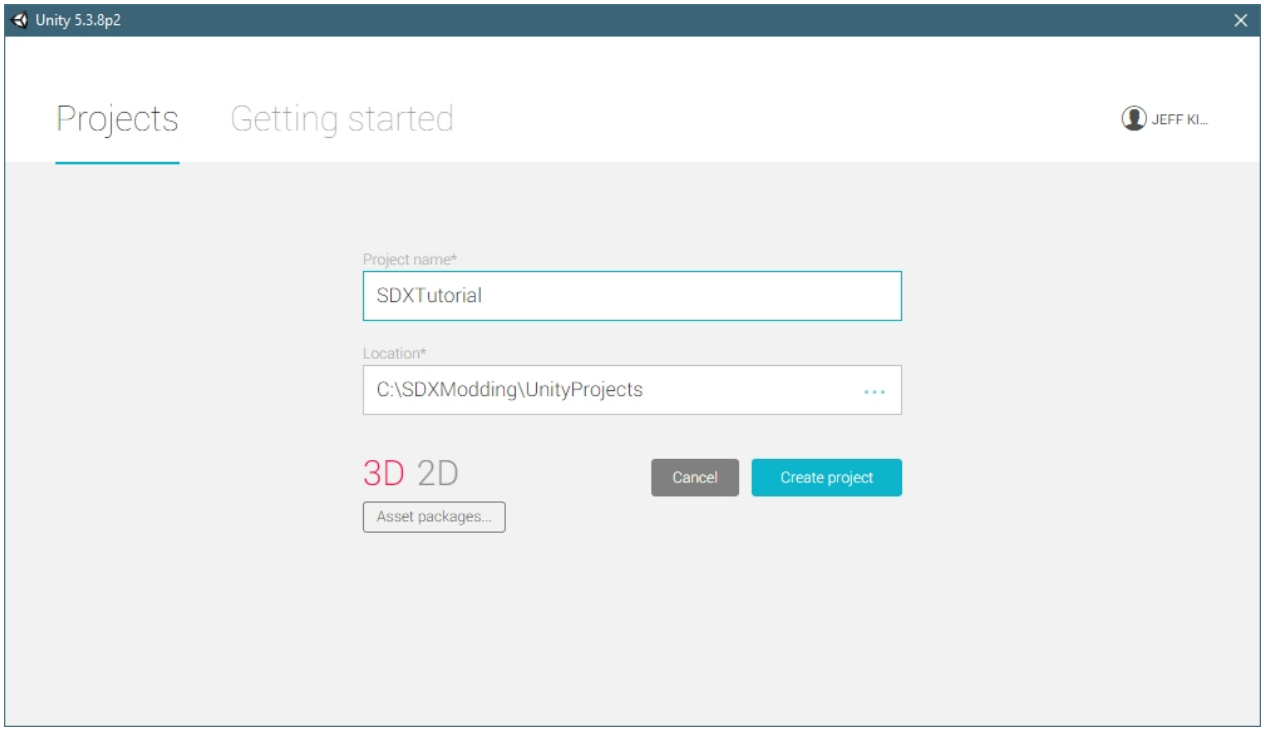
Xyth has created a series of YouTube videos guiding you through on how to make a Unity Project and exporting assets. Those videos supersede this part of the tutorial, however, they are left for review if so desired.

# ***Creating and Exporting models From Unity for use in 7D2D***

Let's create a New Unity Project

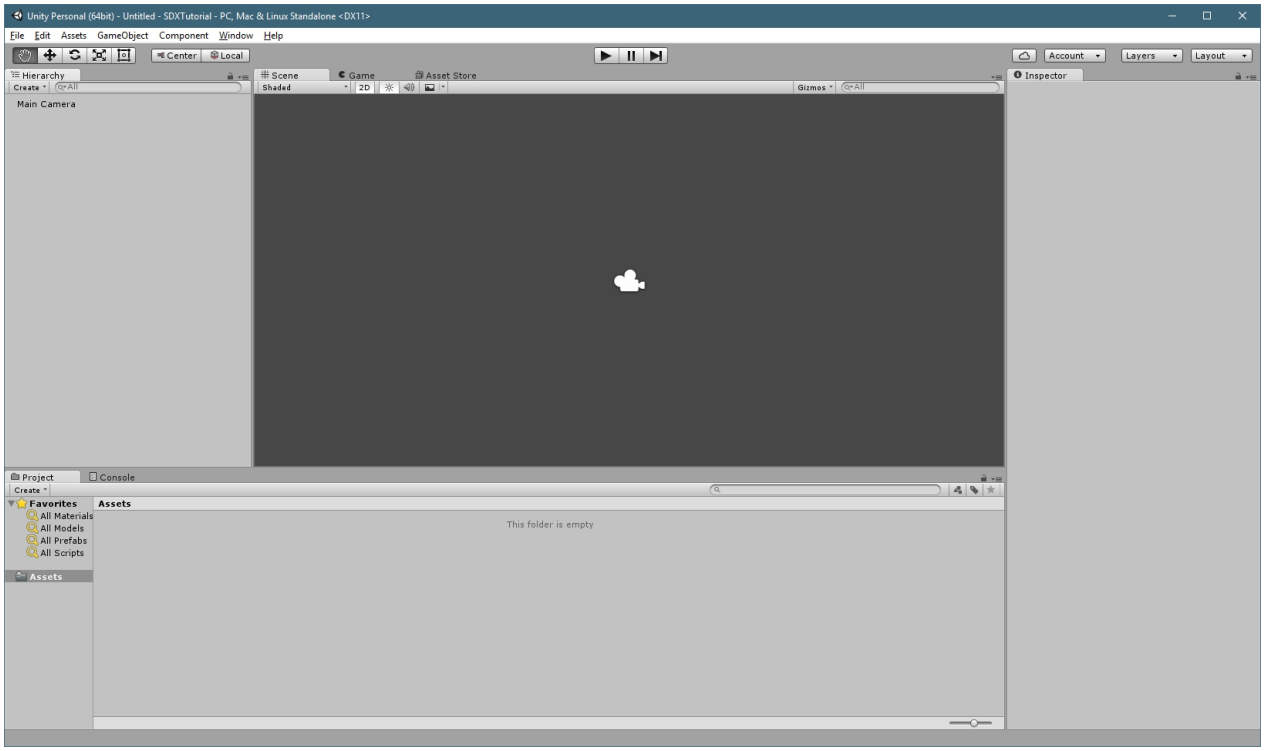



Give it a relevant name:

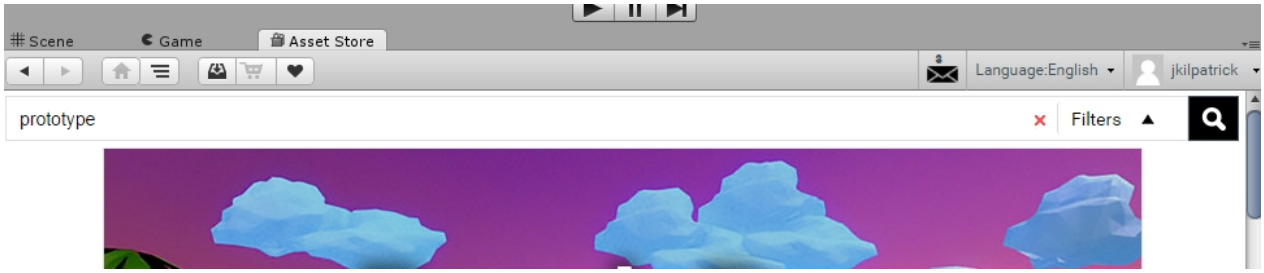


## Adding A Demo Prototype Asset

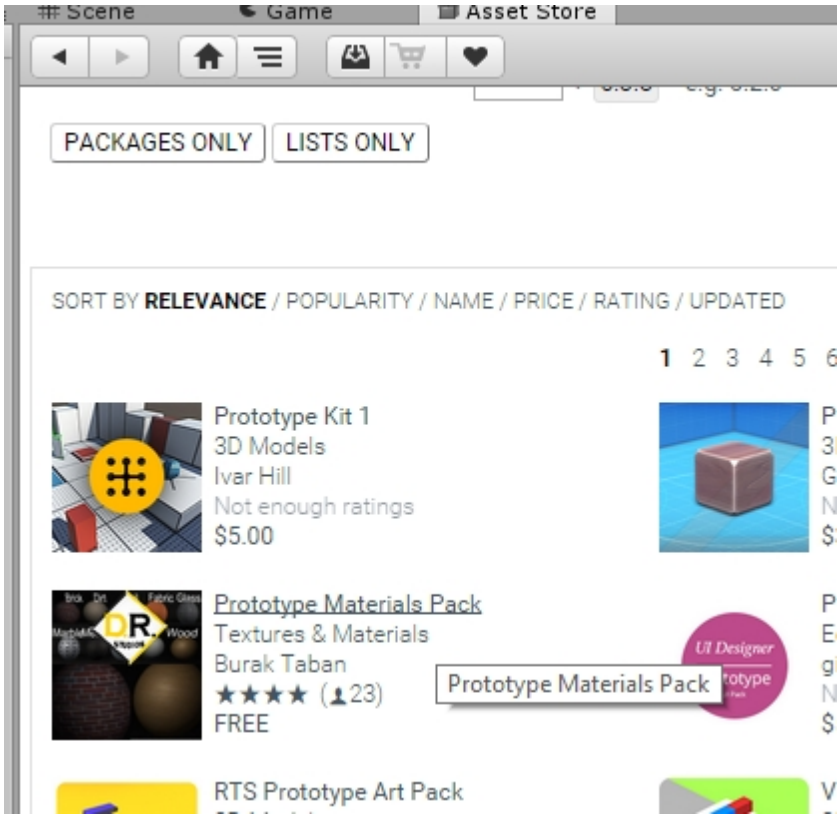
Click on the Asset Store link at the center of the screen, so that we can grab a sample Texture Package



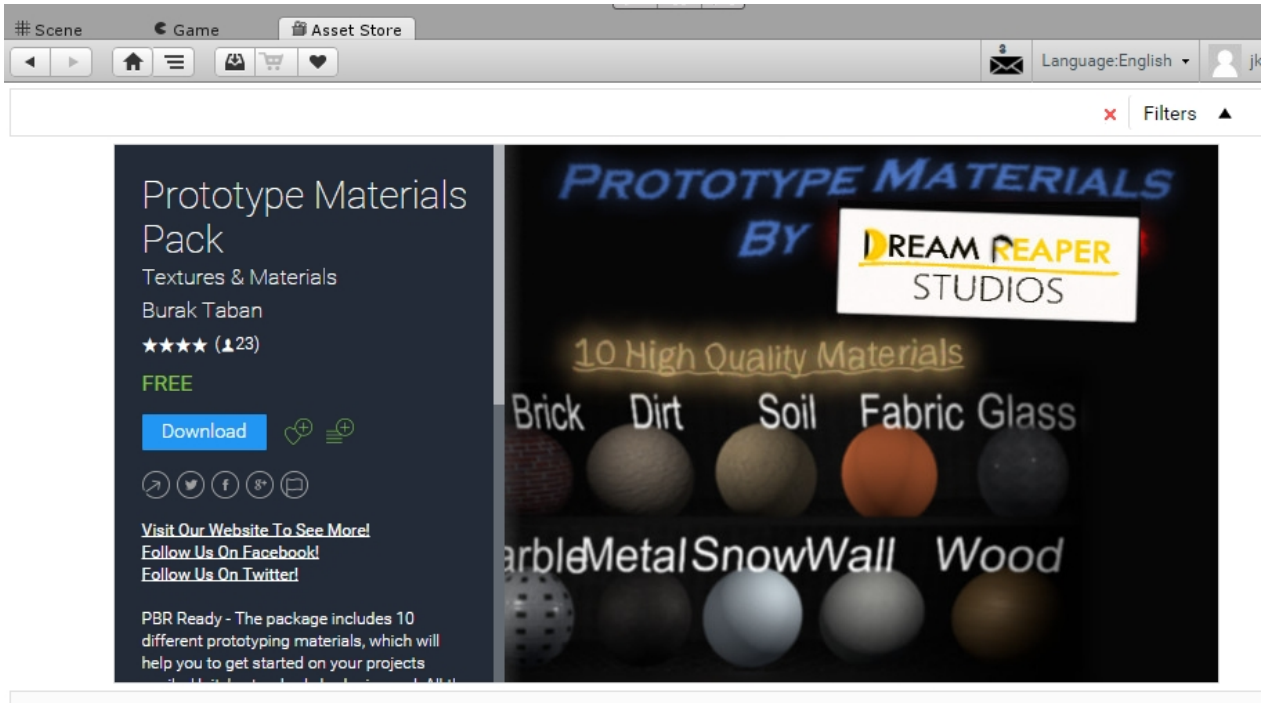

In the Search Window, type in prototype




Scroll down until you see "Prototype Materials pack"



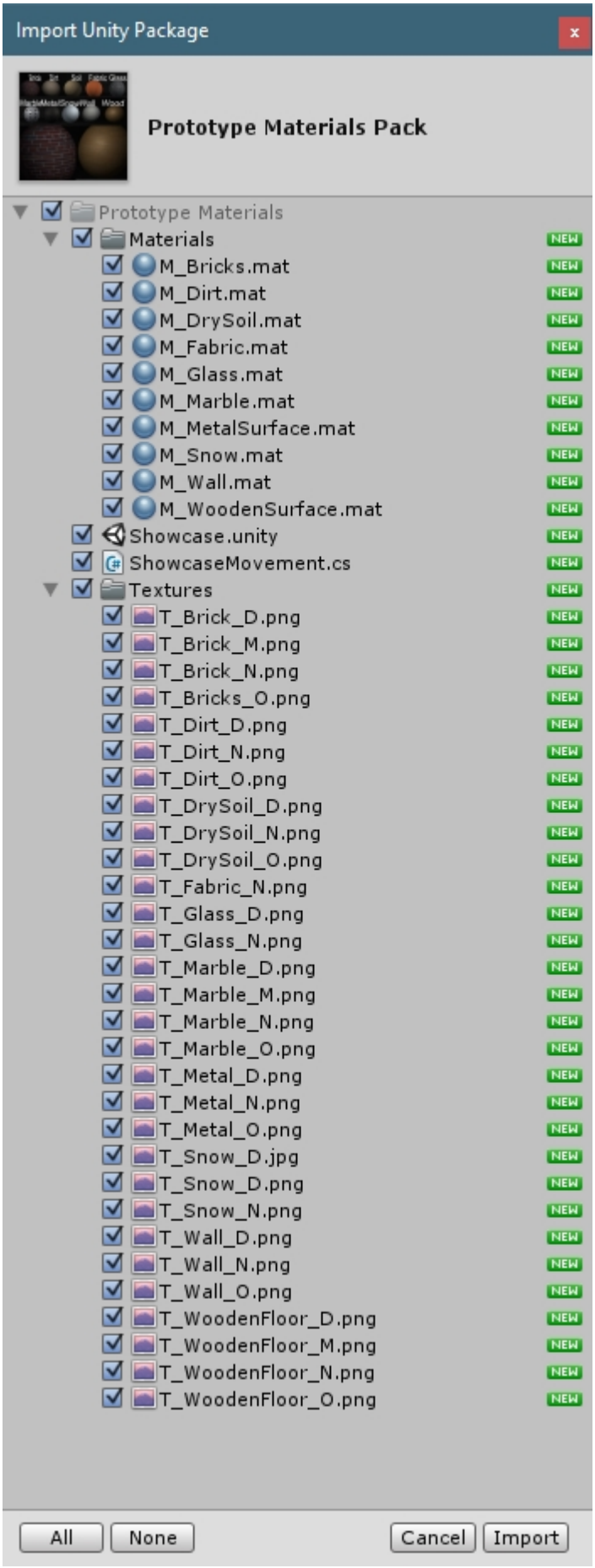

Click on it to load the Store Page for it




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

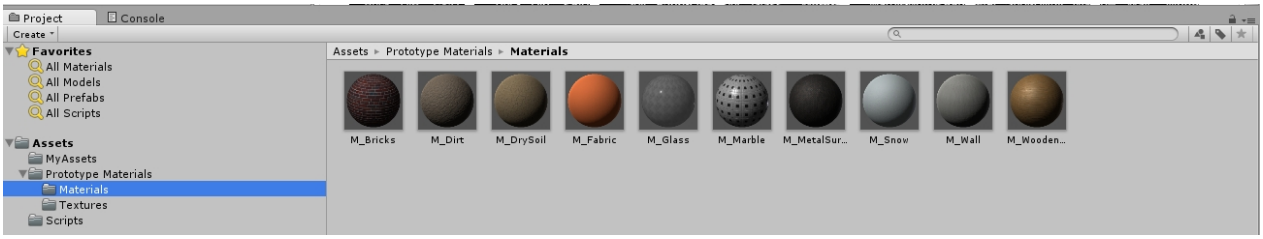
It will download the asset


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




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

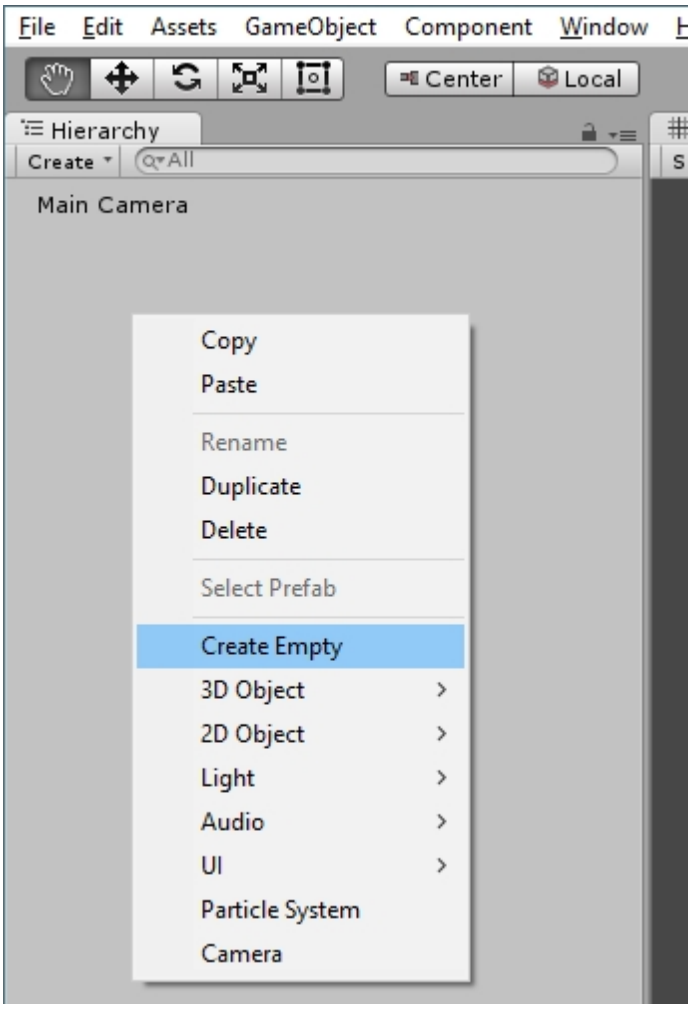
In the bottom window, under Assets, you'll see a new folder structure with the Prototype materials



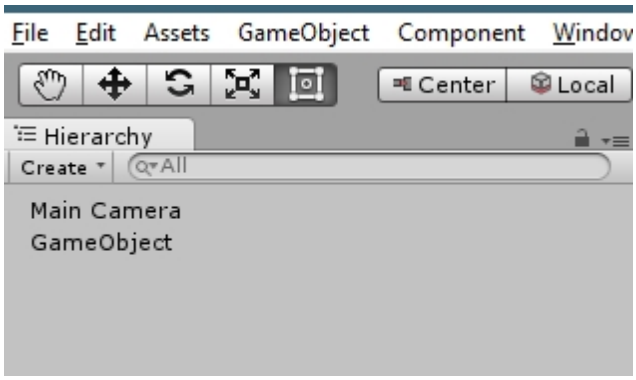



## Creating a Sample Cube

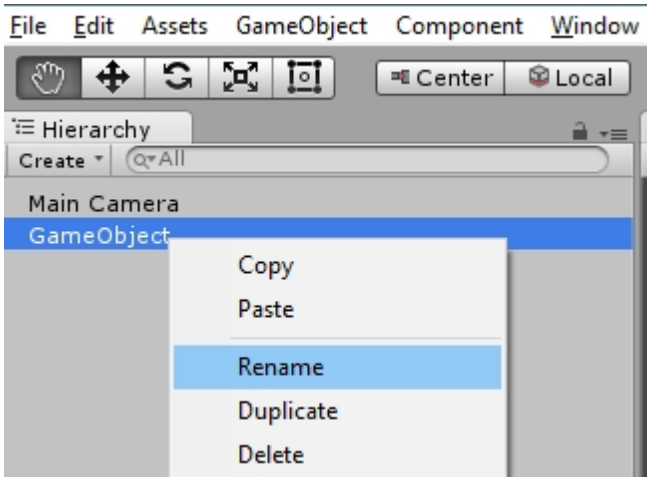
Right click under "Main Camera", and select "Create Empty"



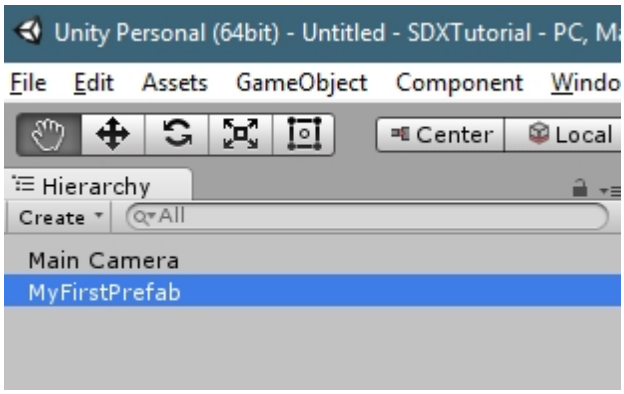

This creates a new "GameObject" item under the "Main Camera"




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

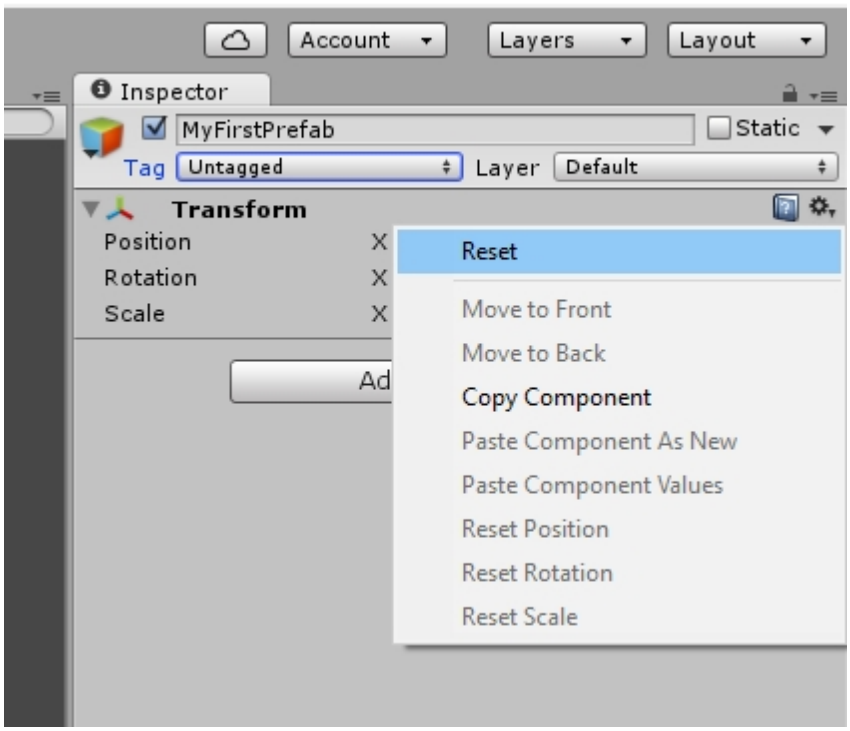



Rename it to "MyFirstPrefab"

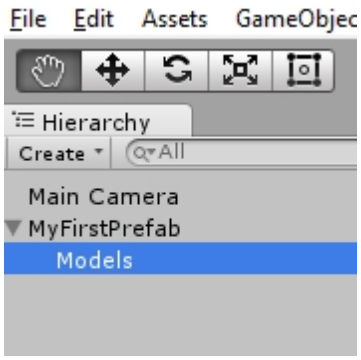



With "MyFirstPrefab" highlighted, we'll want to make sure that the Transform is reset to 0.

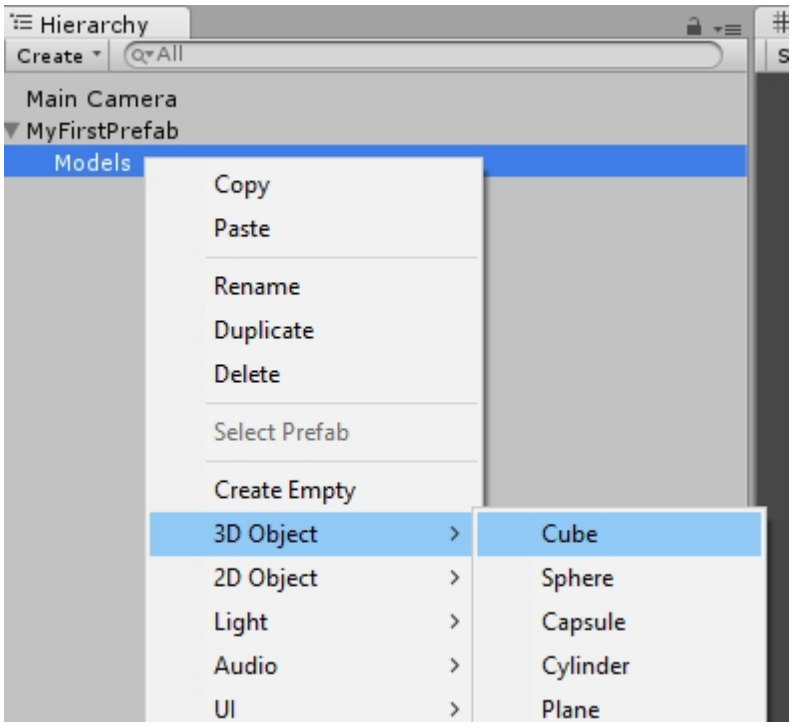
Click on the little Gear on the right hand side of the screen, and select "Reset"



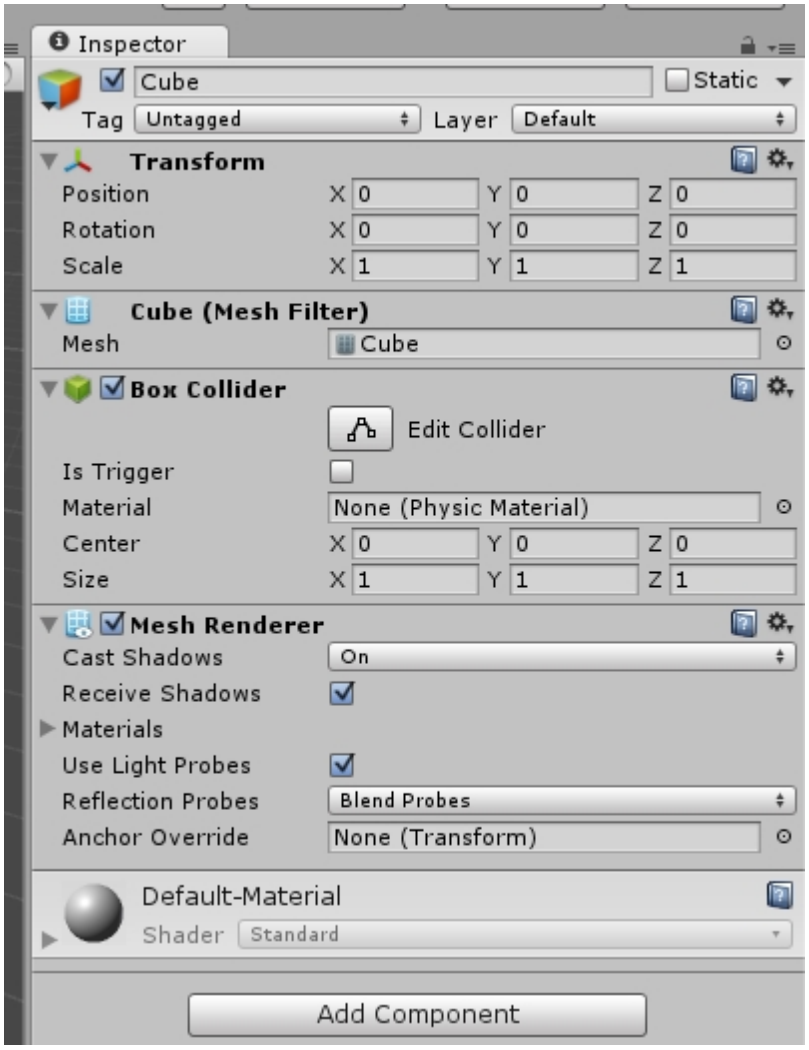

Right click on "MyFirstPrefab", and select "Create Empty". Rename the new "GameObject" to "Models"



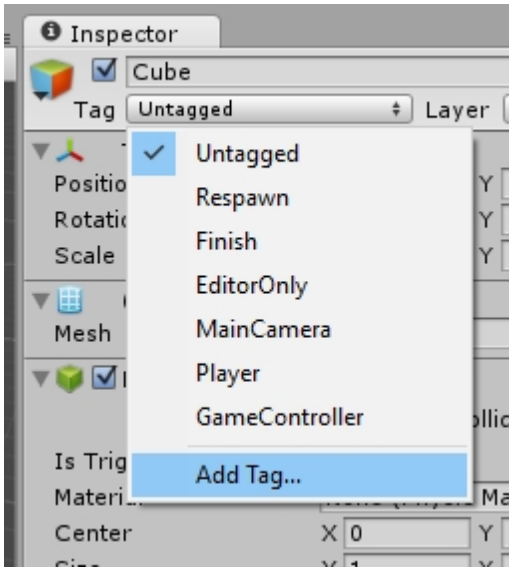

Right Click on "Models", and select "3D Object", then Cube




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

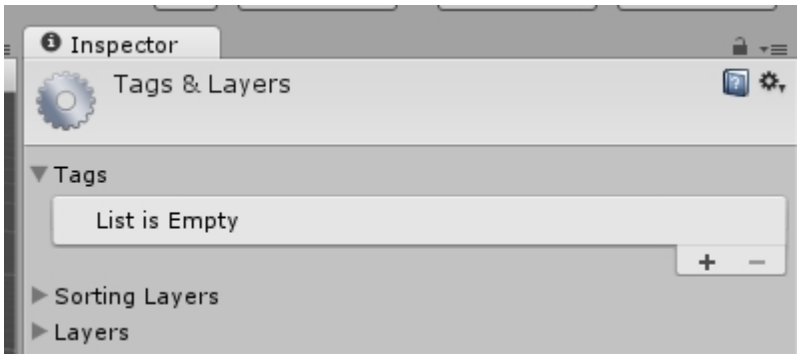


Click on the Tag drop down, that says "Untagged".



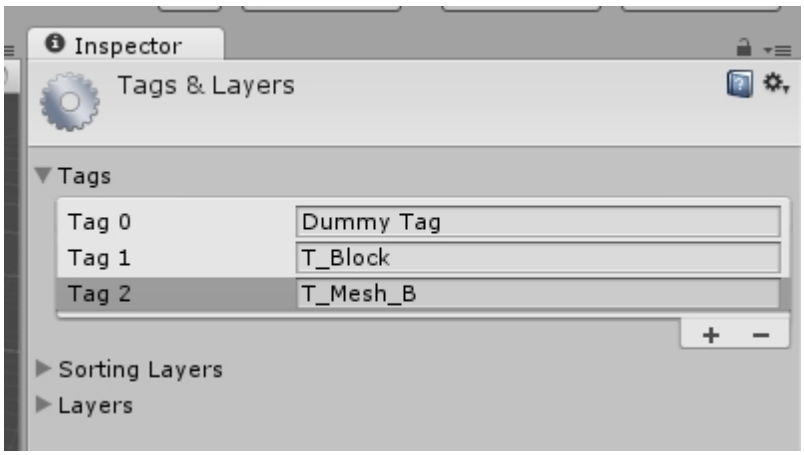

All Game Objects for 7 Days To Die that need meshes, are required to have a T_Mesh_B tag. If you see the screen above, without the T_Mesh_B, click on "Add Tag..."

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

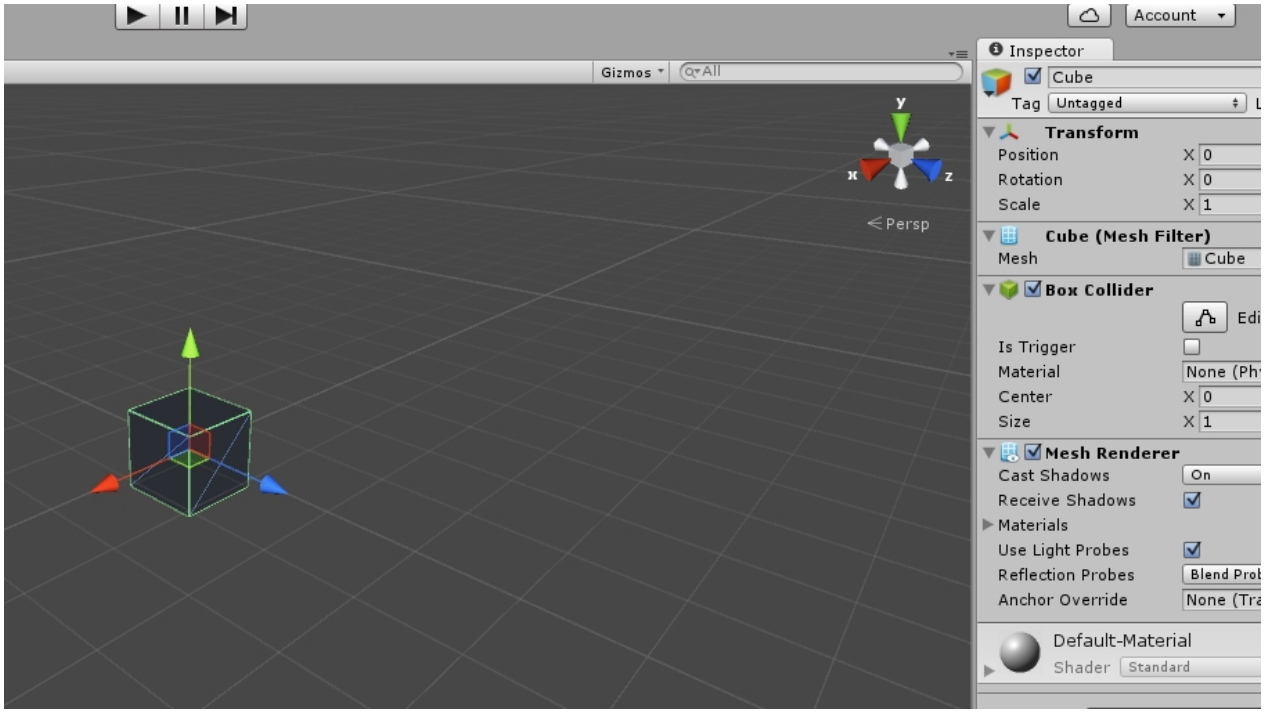



Click on the + button, and add two new tags, in this order:

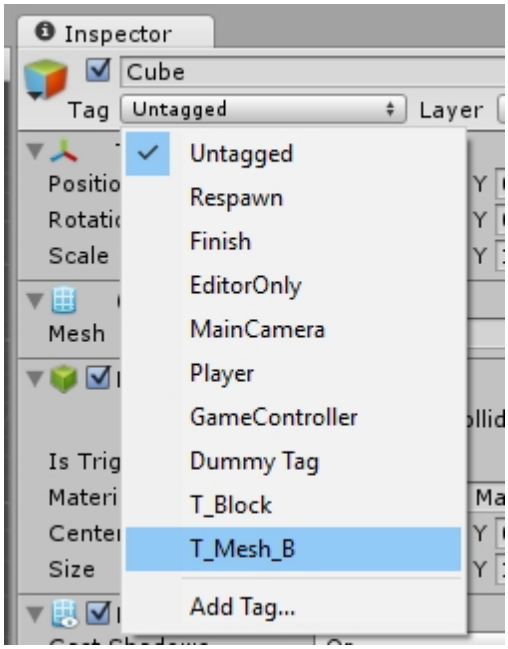
Dummy Tag
T_Block
T_Mesh_B




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

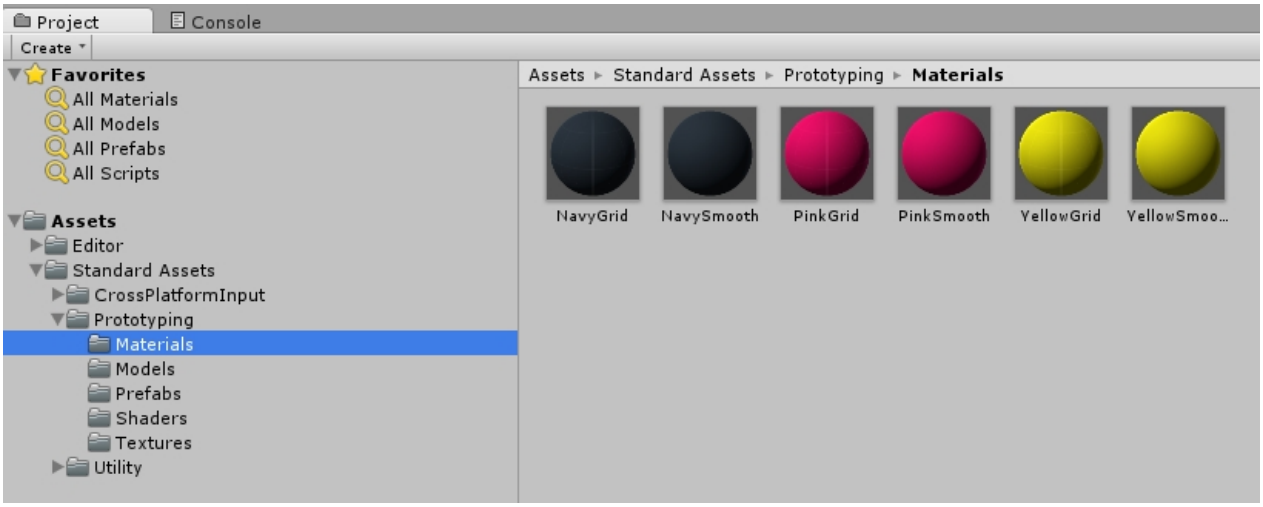



Then, click on the Tag drop down again, and select "T_Mesh_B".



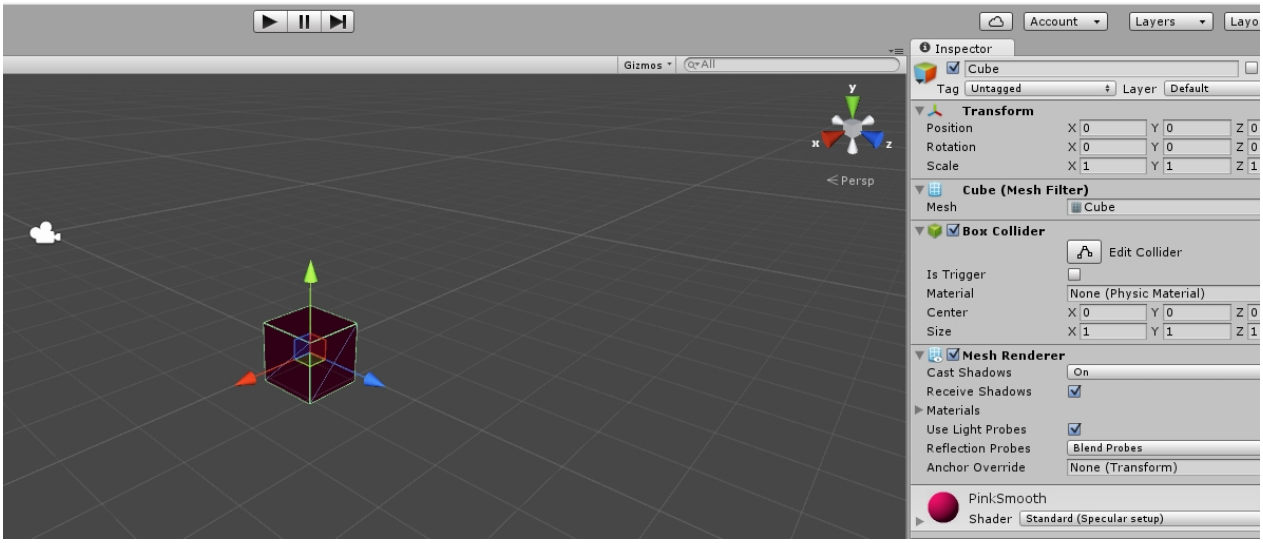

Click on the Cube on the Heirarchy.

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

### *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.

# 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.



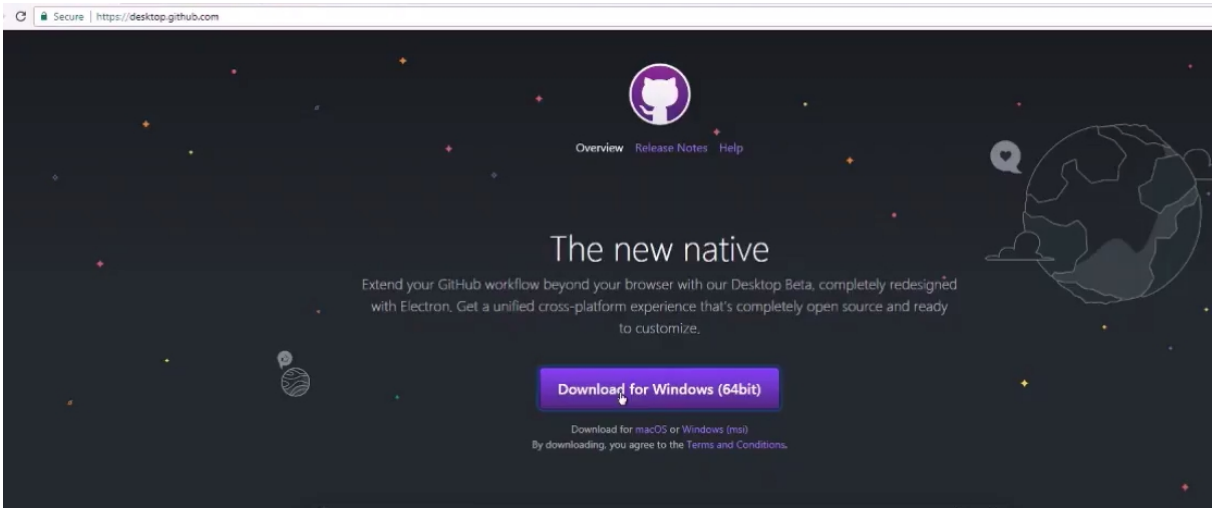
### Installing and Configuring Github

# GitHub Desktop

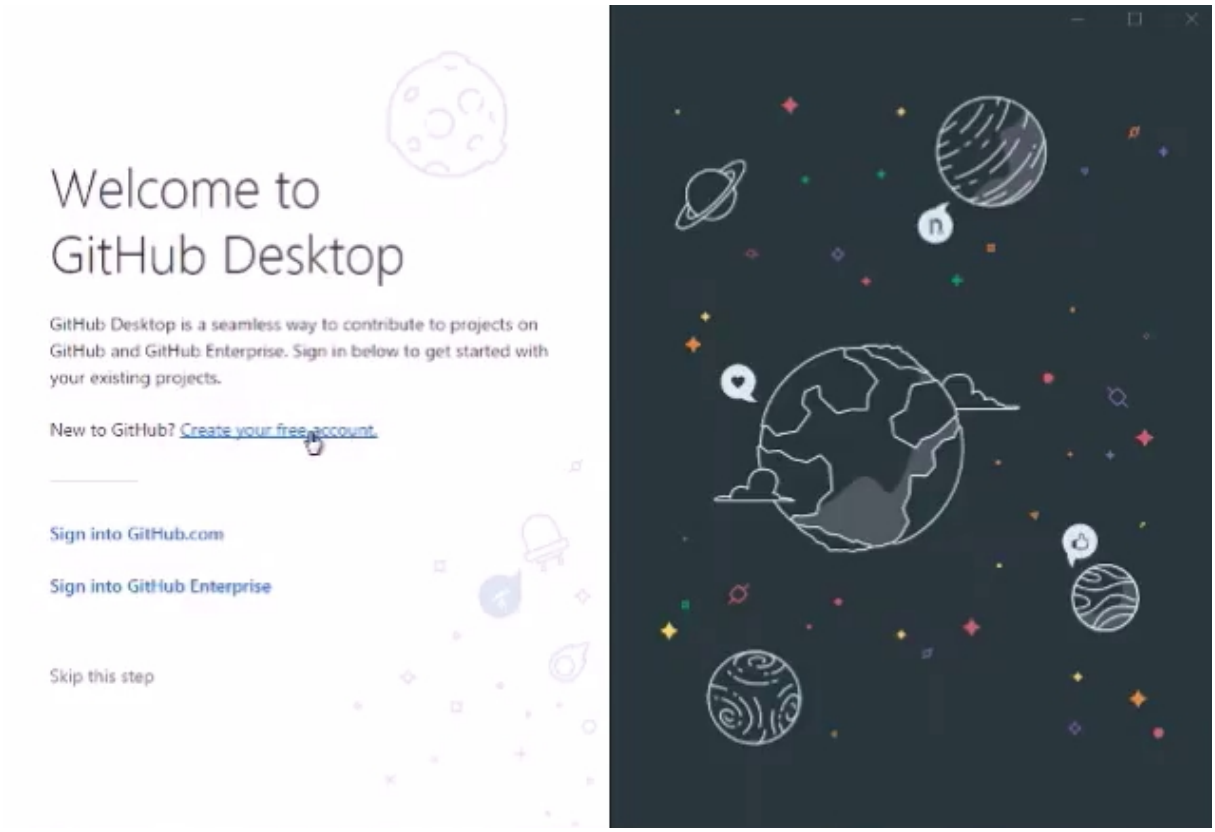
GitHub Desktop is an easy to use tool to manage code. You can download it here.

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

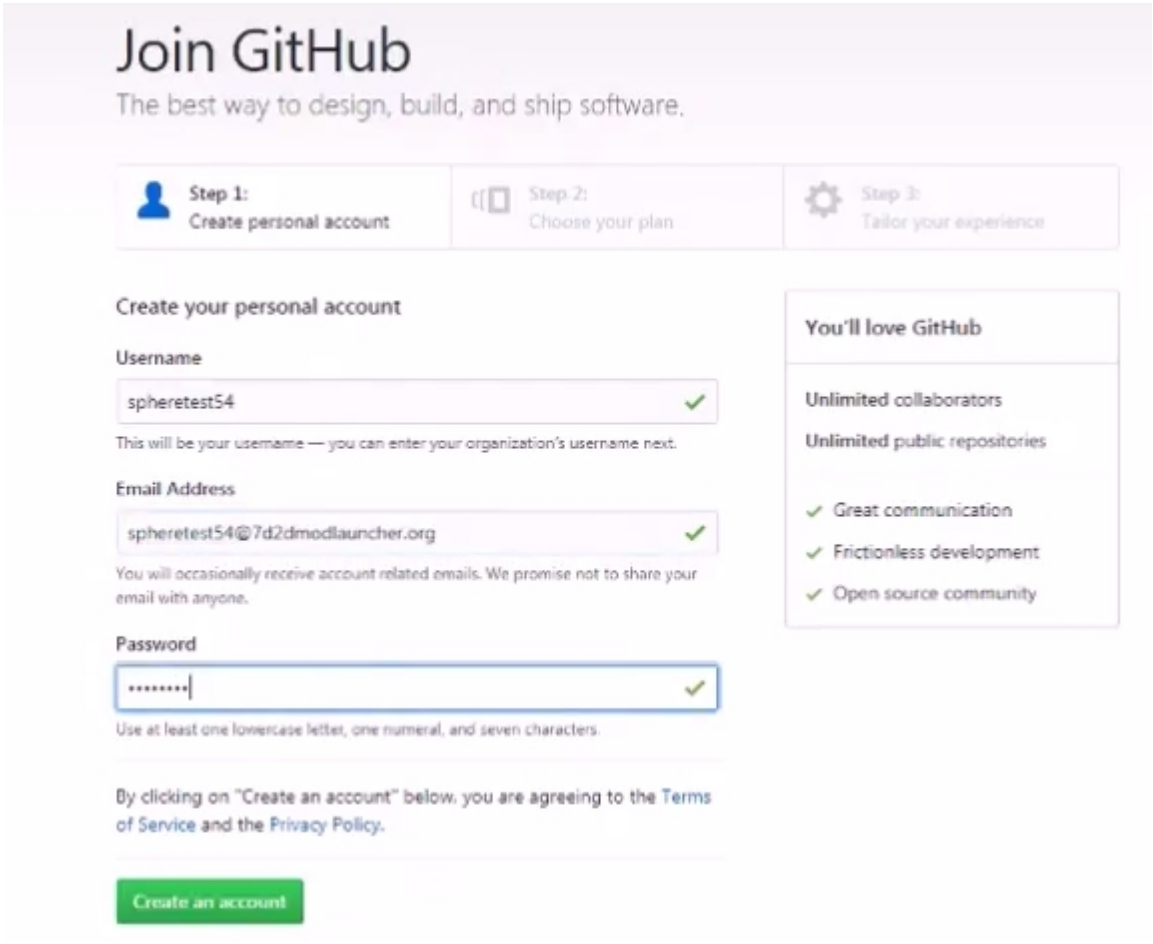
To get started, install GitHub Desktop



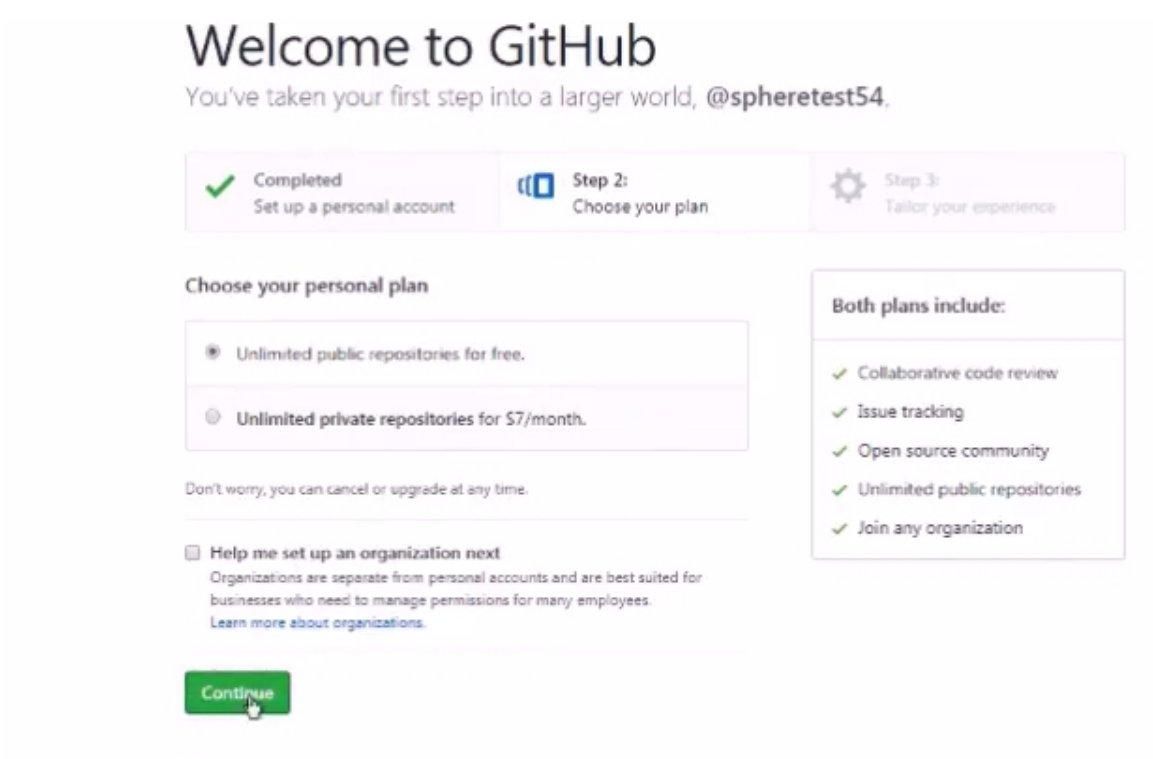

Click on "Create your free account". This will open a new browser window or tab:



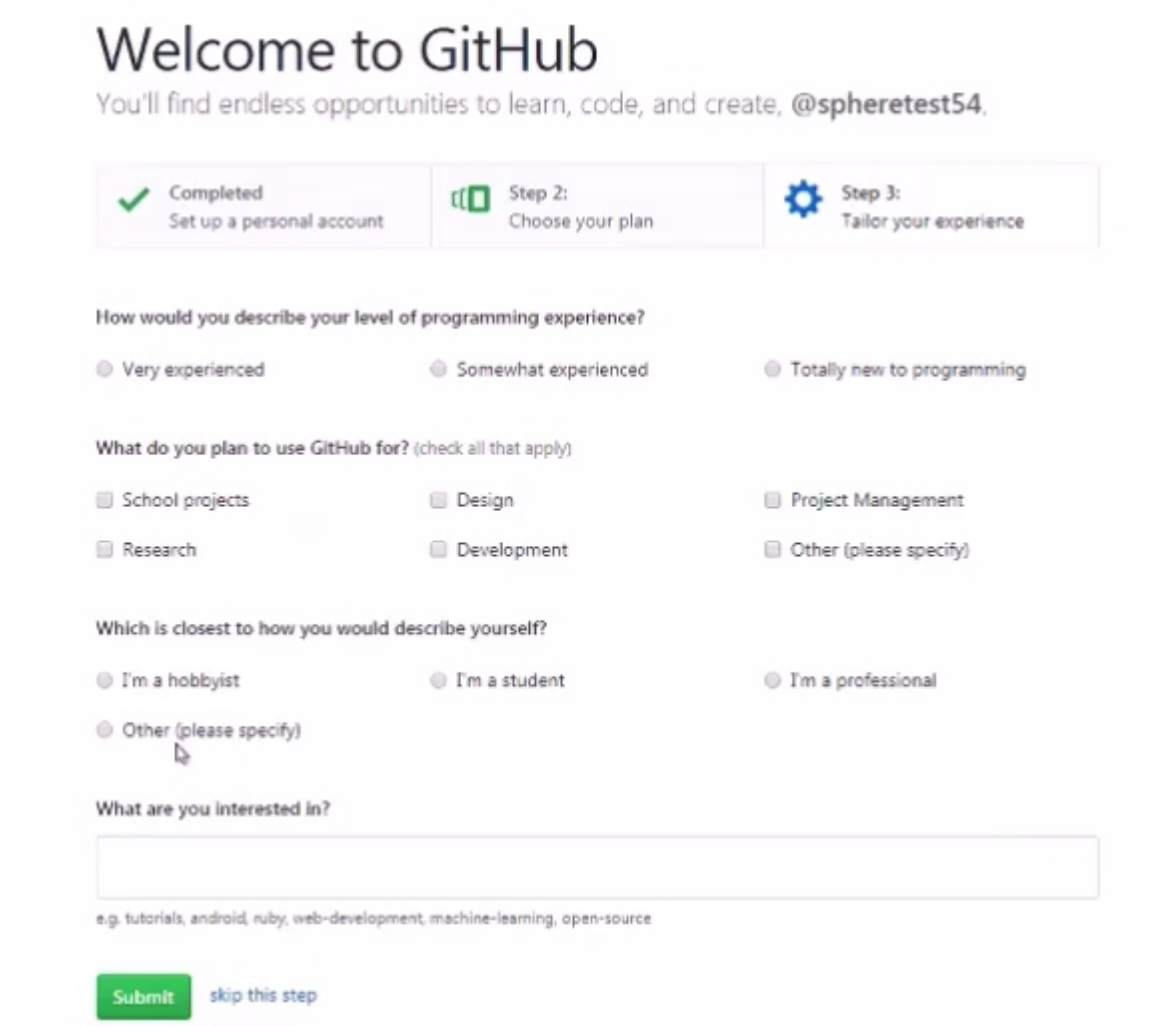

Enter in a user name, your email address ( you'll get a verify email link), and your password.



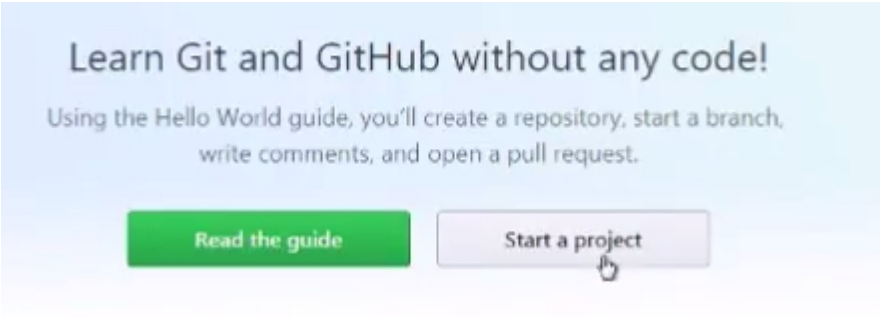

Accept the default of "Unlimited public repositories for free"



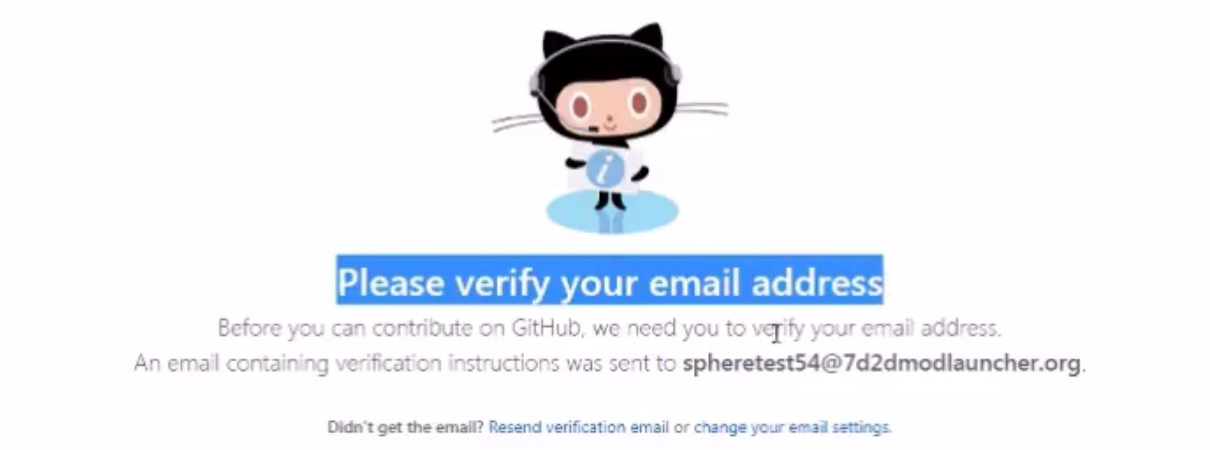

Optionally, you may fill out their small questionnaire



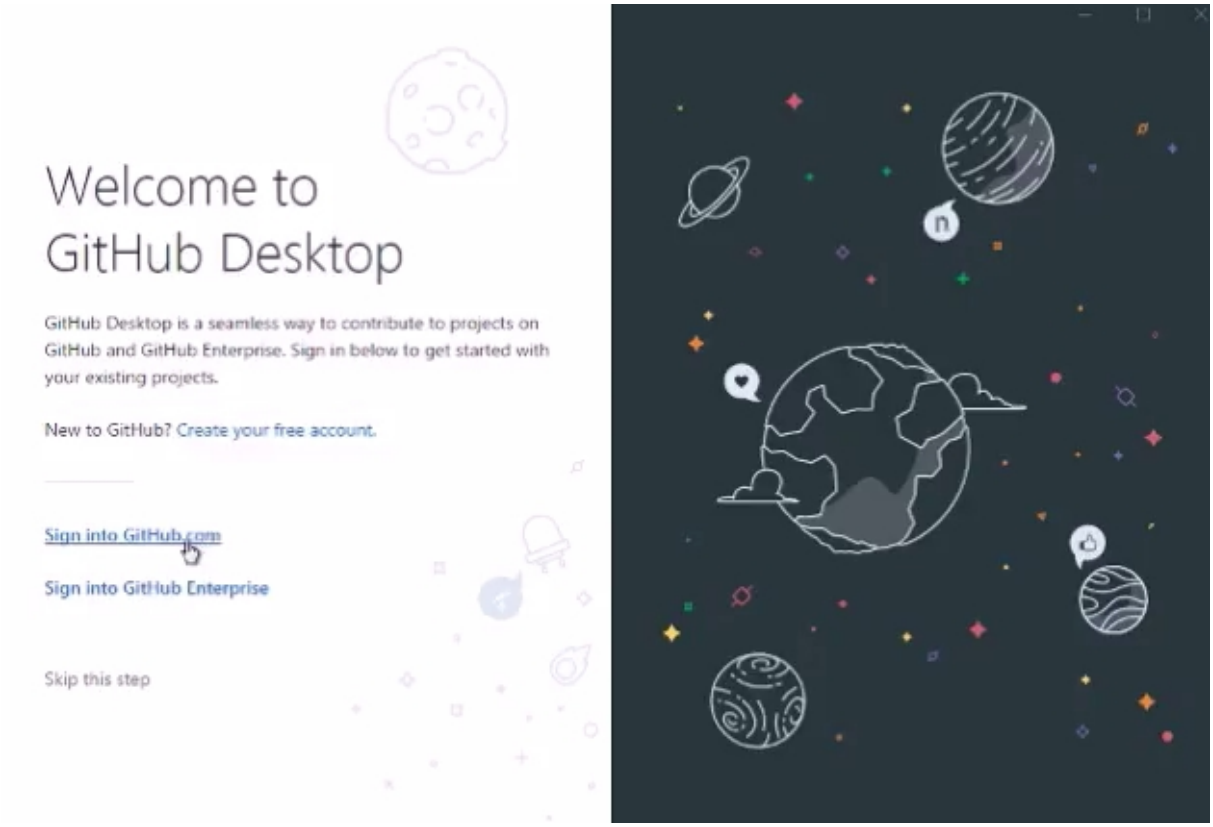

Create a New Project



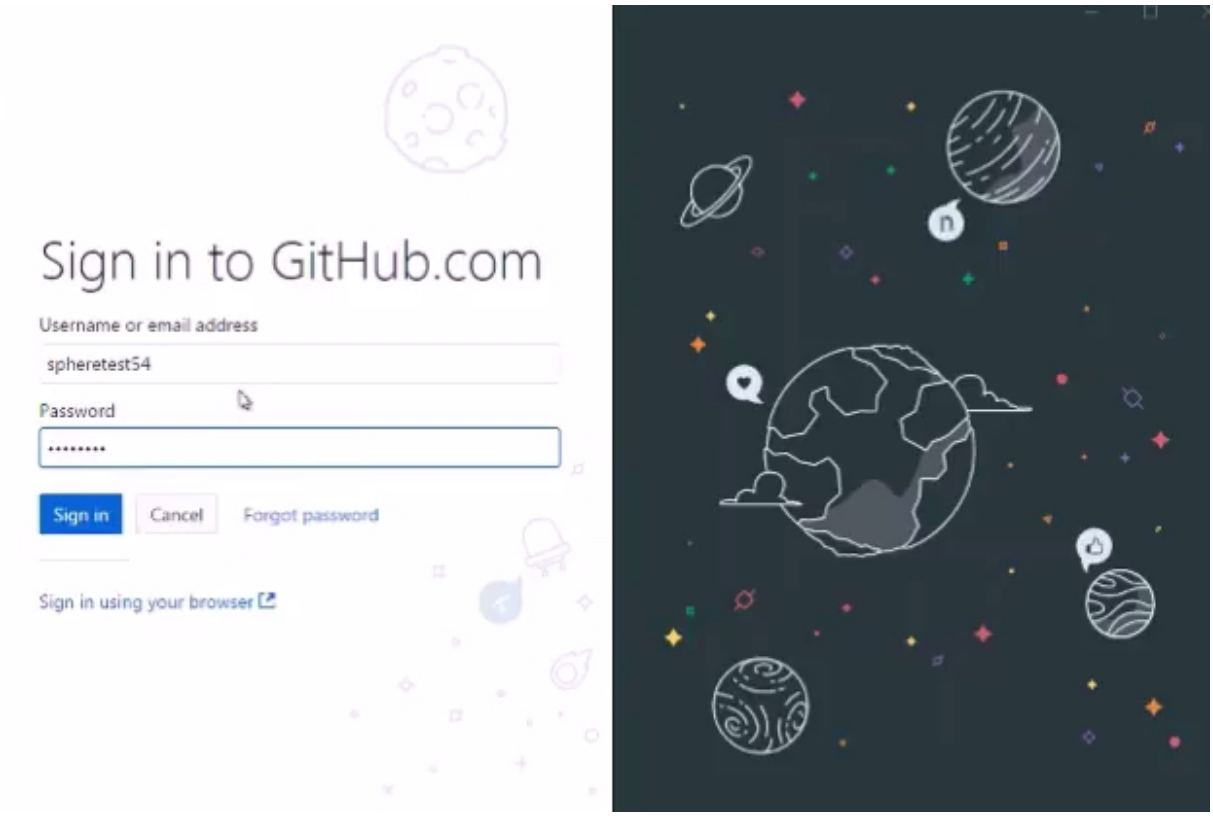

Check your email for the Verification link




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

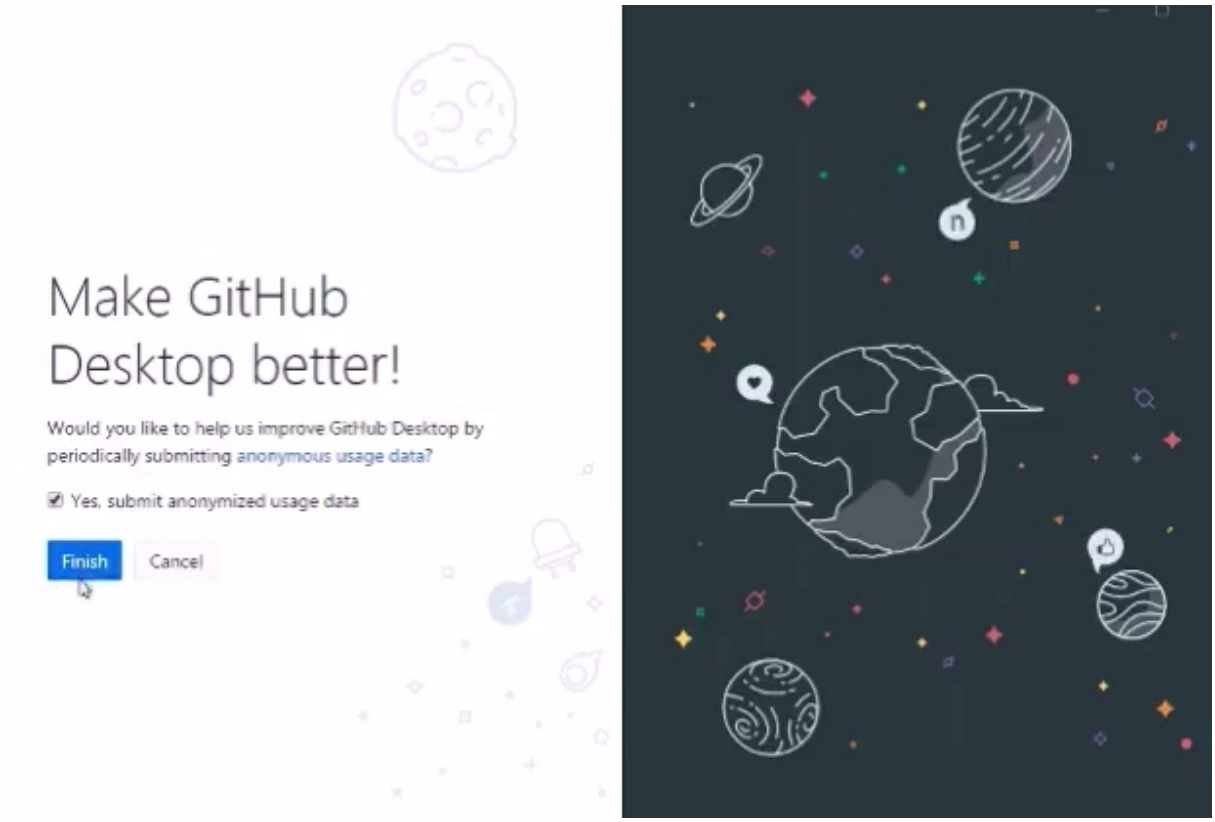



Enter in your username or email address, and your password.




Configure Git. You can just accept the defaults.

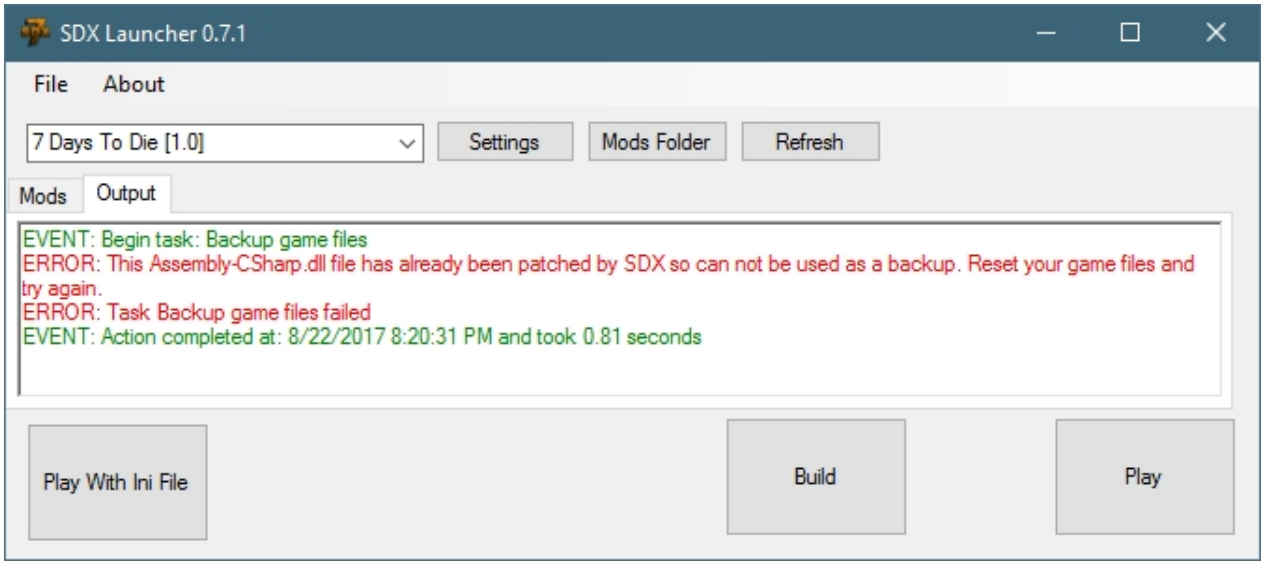

Click on Finish.






## Troubleshooting

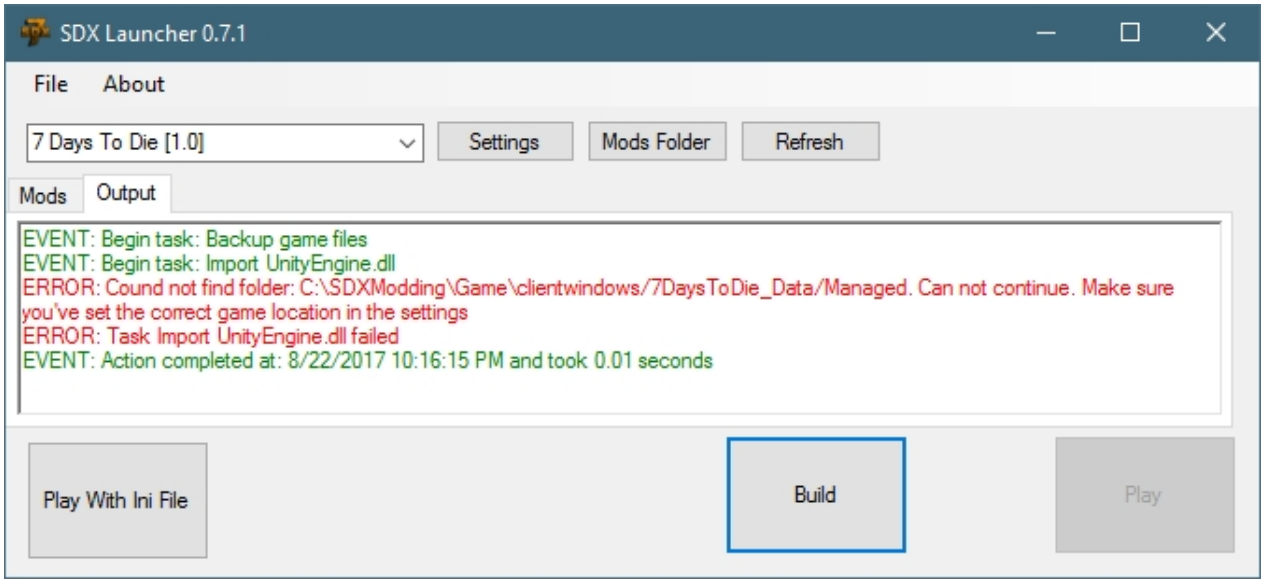
When you press the Build button, you get the following error:




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.