DD2 Modder Guide

Updated Jun 13, 2025

Intro

What is Darkside?

Darkside is a Unity project used to create and bundle Darkest Dungeon 2 files together into a mod that can be submitted directly to Steam Workshop. This tool is found on Steam via Tools under the **Darkest Dungeon 2: Mod Tools.**

If you are not on Steam: We have also uploaded the Darkside project to Google Drive for non-Steam users to download. The Darkside project obtained via Steam will always be up to date with the latest version. If you download the project from Google Drive, you'll just need to check back every once in a while to ensure you're using the most recent version of Darkside.

What mods can be created with Darkside?

For this initial testing, **Darkside** can be used to create most types of items; **trinket**, **combat**, **rest** (inn items), **memory** (under the hood memories are items), and **general stagecoach**, **pet**, **trophy**, and **flames**

Most of this documentation covers **adding new items**, however, **overriding existing data** is also supported but requires a bit more manual setup. Details for override data setup can be found under the Overriding Data section.

Where can I provide feedback and ask for help?

We will be collecting feedback and holding discussions in the DISCORD channel. With this initial deployment and guide we want to challenge you to take these basic tools and set ups as far as you can, and help construct more robust guides on how we can help onboard future modders. So while this is the beginning introduction of what can be done, we believe there is more currently possible, but will need to be mapped out by you! All the while reporting to us where the biggest points of friction are or what is currently still unaccessible.

Are there mod examples for reference?

YES! The default data created whenever you initially make a new item type is filled with examples for that specific type. The examples provided have been created primarily for teaching the mod tools so the numbers, effects, and buffs will be all over the place.

In addition, there is a supplementary XLSM file that can be opened with excel, used for managing/exporting the data from an easily readable source. It contains the same example data that's setup with select drop down options and notes explaining specific cells.

Another excellent reference is the base game data! Using **Visual Studio Code** (this is the tool I recommend), you can easily view and search through our data. Details for this process can be found under the **TIPS AND TROUBLESHOOTING!** section (this section is currently WIP).

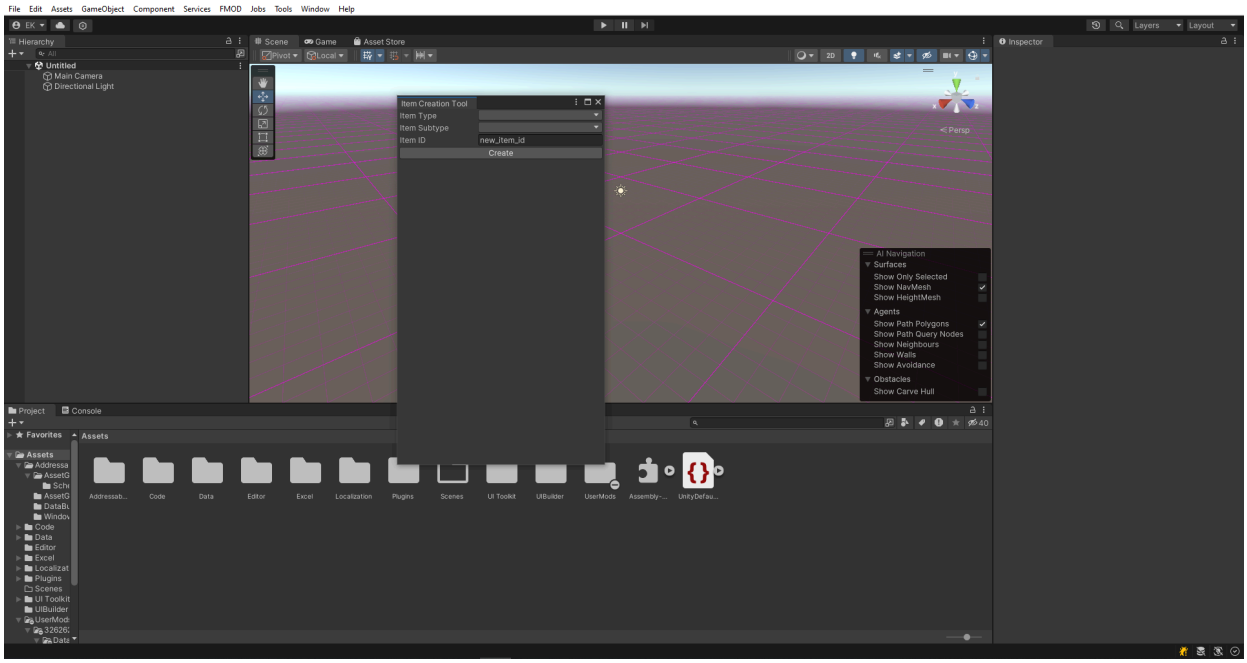
Requirements

1. Install **Unity 2022.3.16f**
2. Download **Darkside** project
3. Signed into **Steam/Steam Workshop**
4. Updated build of **Darkest Dungeon 2** with mod support
5. Recommendations
    a. Access to **Visual Studio Code** or another IDE (useful for viewing base game data files)
        i. Details: Visual Studio Code to view base game data
    b. Access to **Excel** (Google sheets won't work since we're providing an XLSM file with macros)
        i. Details: Using the Excel exporter tool
    c. Get acquainted with the Unity Editor: Explore the Unity Editor
    d. Throughout this document there are sections titled **VERY IMPORTANT INFO**, we recommend paying close attention to these!

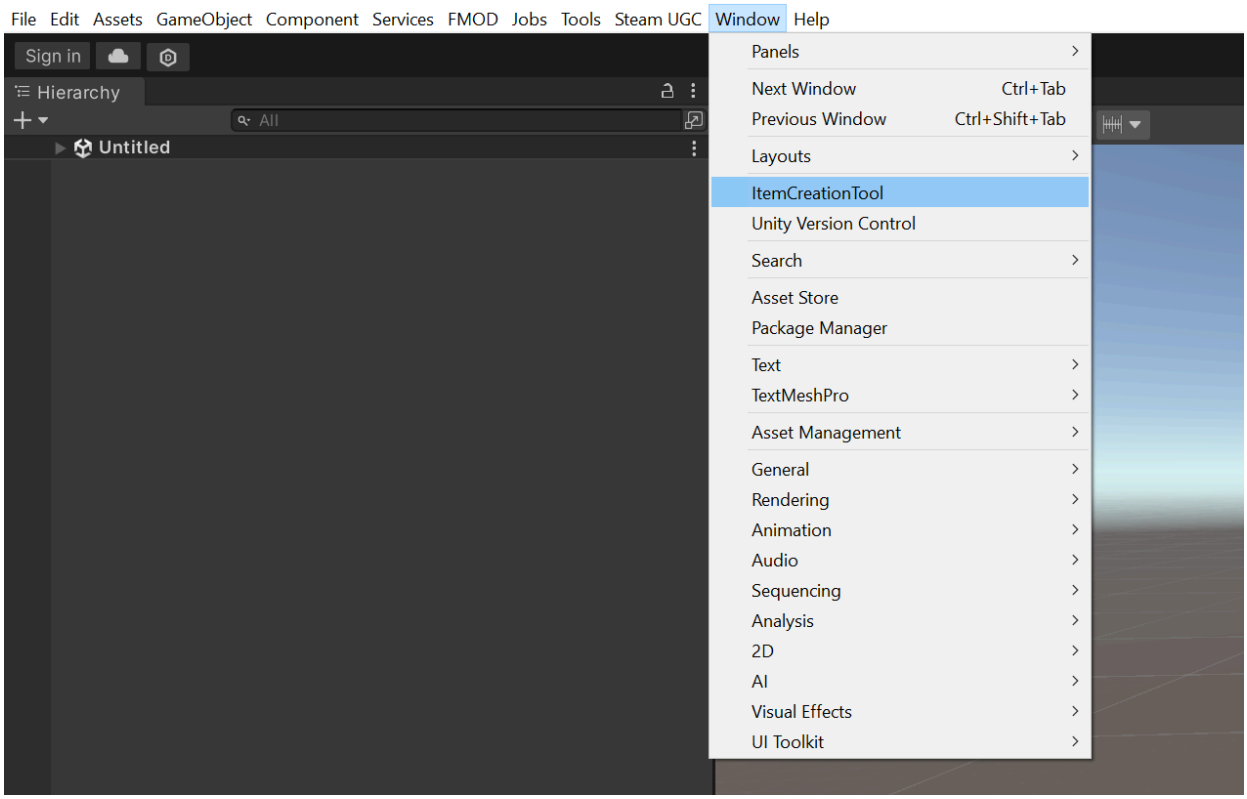
Tool Basics

Opening the project

After opening Darkside, the editor should look something like this with the **Item Creation Tool** window already open



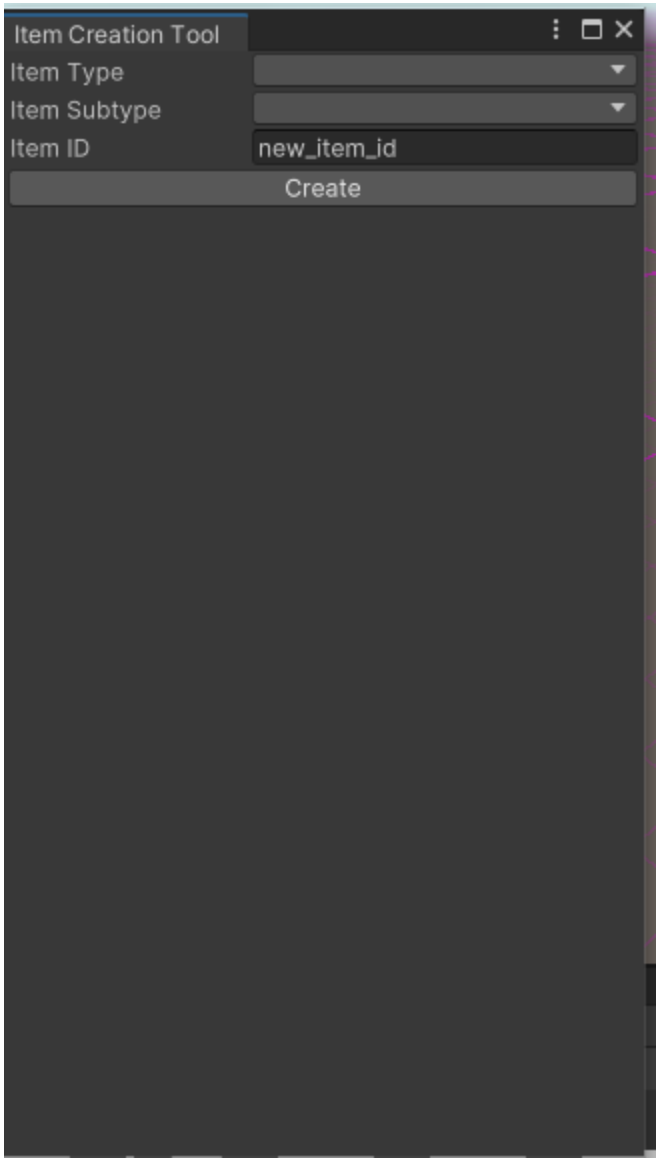
If the **Item Creation Tool** window isn't open or it gets closed, you can open it again by selecting **Window > ItemCreationTool**



Item Creation Tool

The **Item Creation Tool** is what you use to create the default files for the various item types.

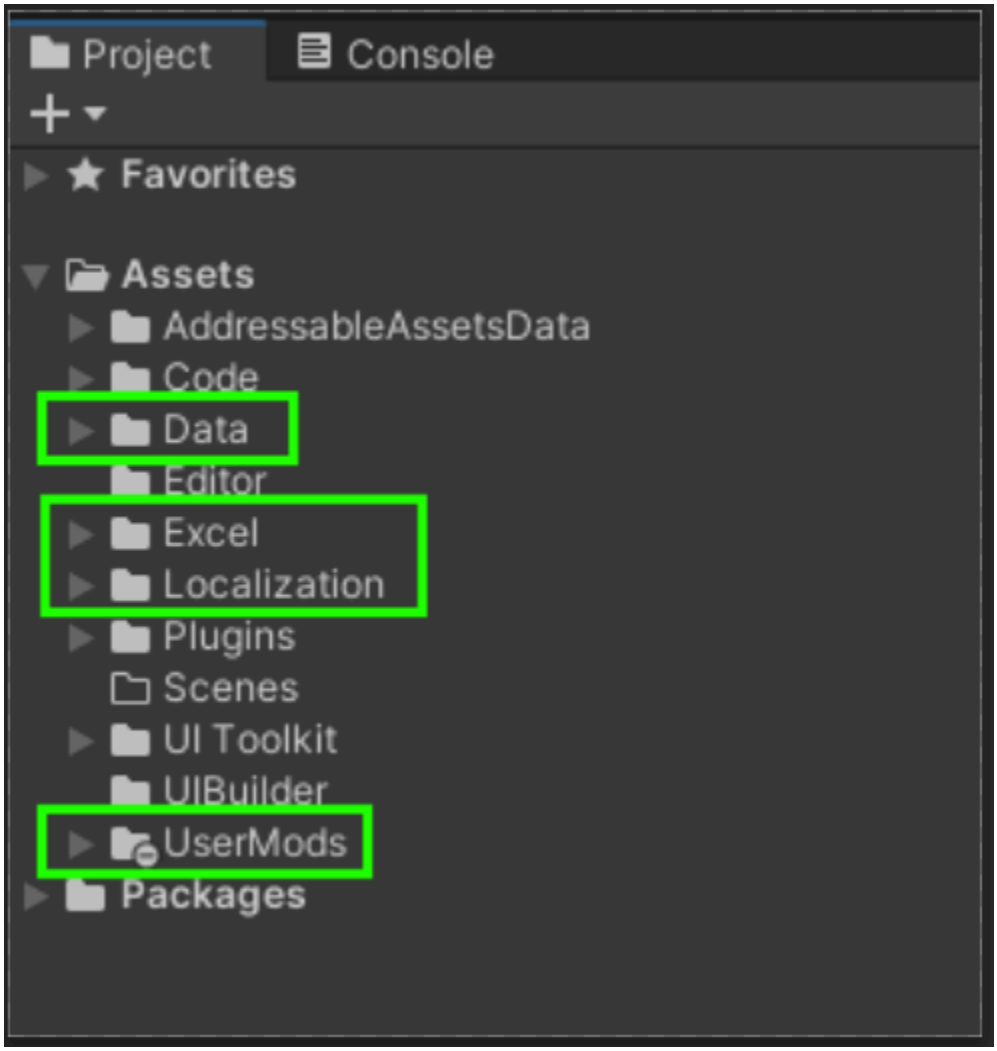
The available **Item Types** are: **rest** (inn item), **combat**, **trinket**, **stage_coach_upgrade**, and **memory**
Item Subtype is currently used for stage_coach_upgrade only and they are: **general, pet**, **trophy**, **infernal**, and **radiant**



Project Folders

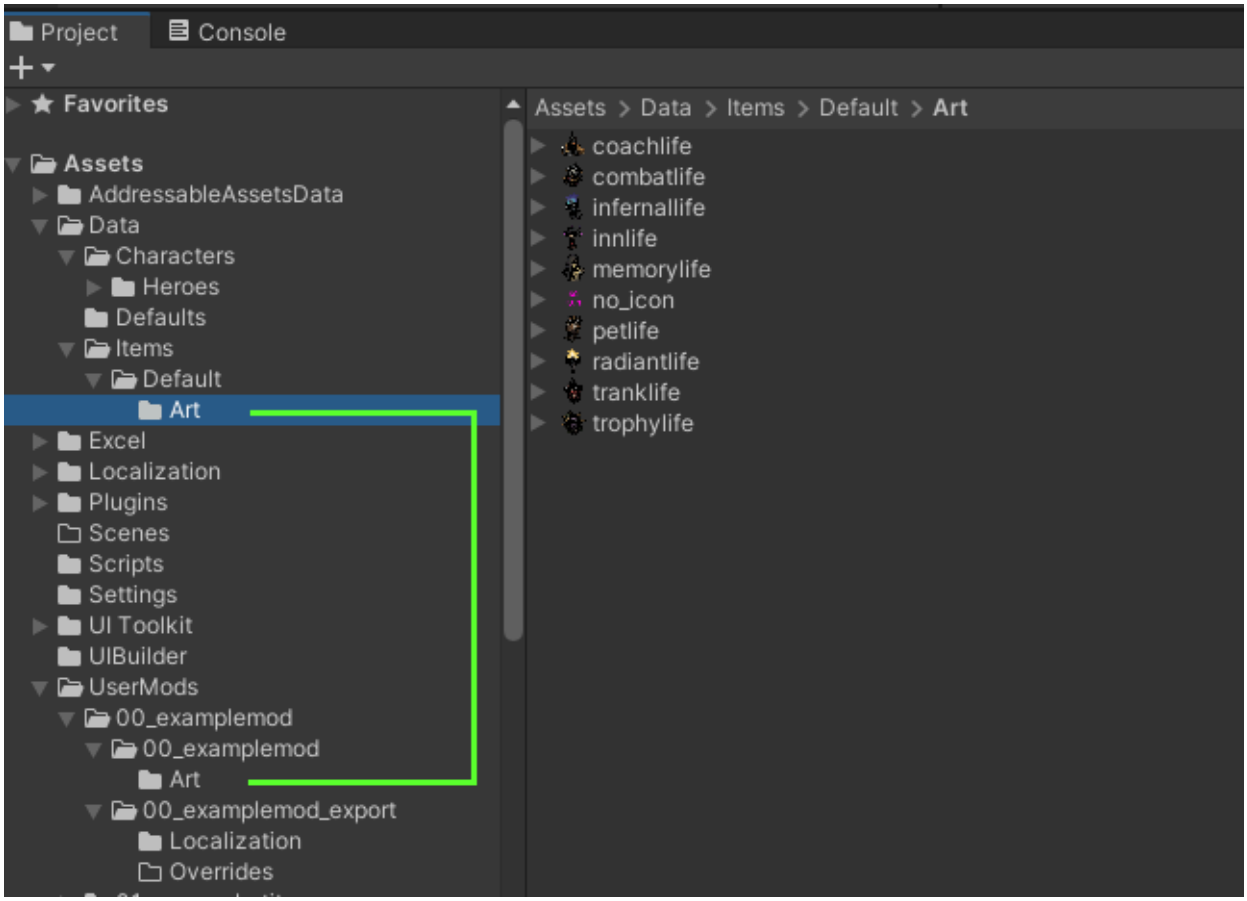
Project Window

The **Project** window displays all the files contained within the Unity project. For the purposes of creating a mod the important folder are **Data**, **Excel**, **Localization**, and **UserMods**



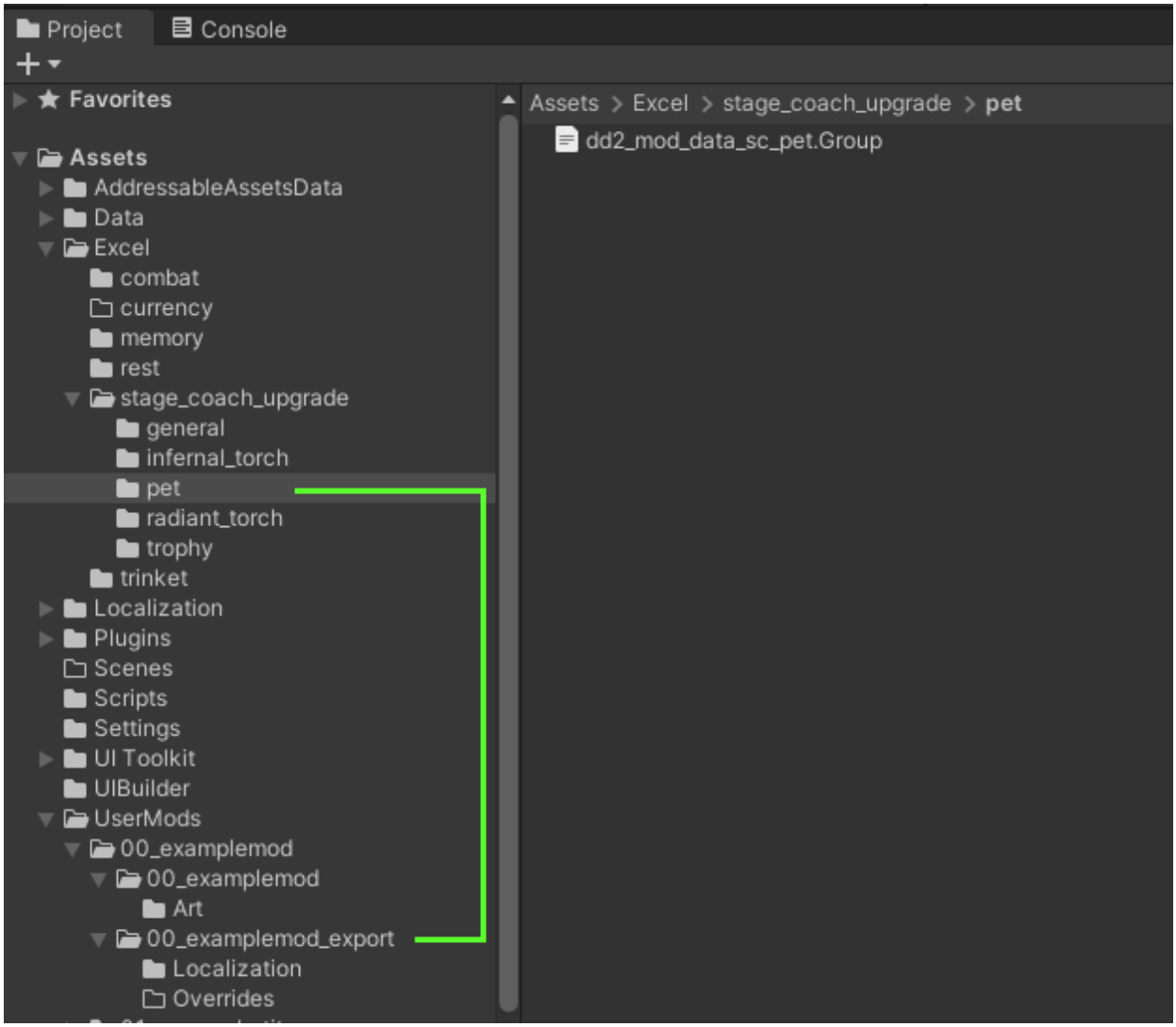
Data

This folder contains default images which are automatically created and added to the files wherever a new item is created. After a new item is created, these images are NOT required to use, they are for placeholder use. The default images can be removed or added to.



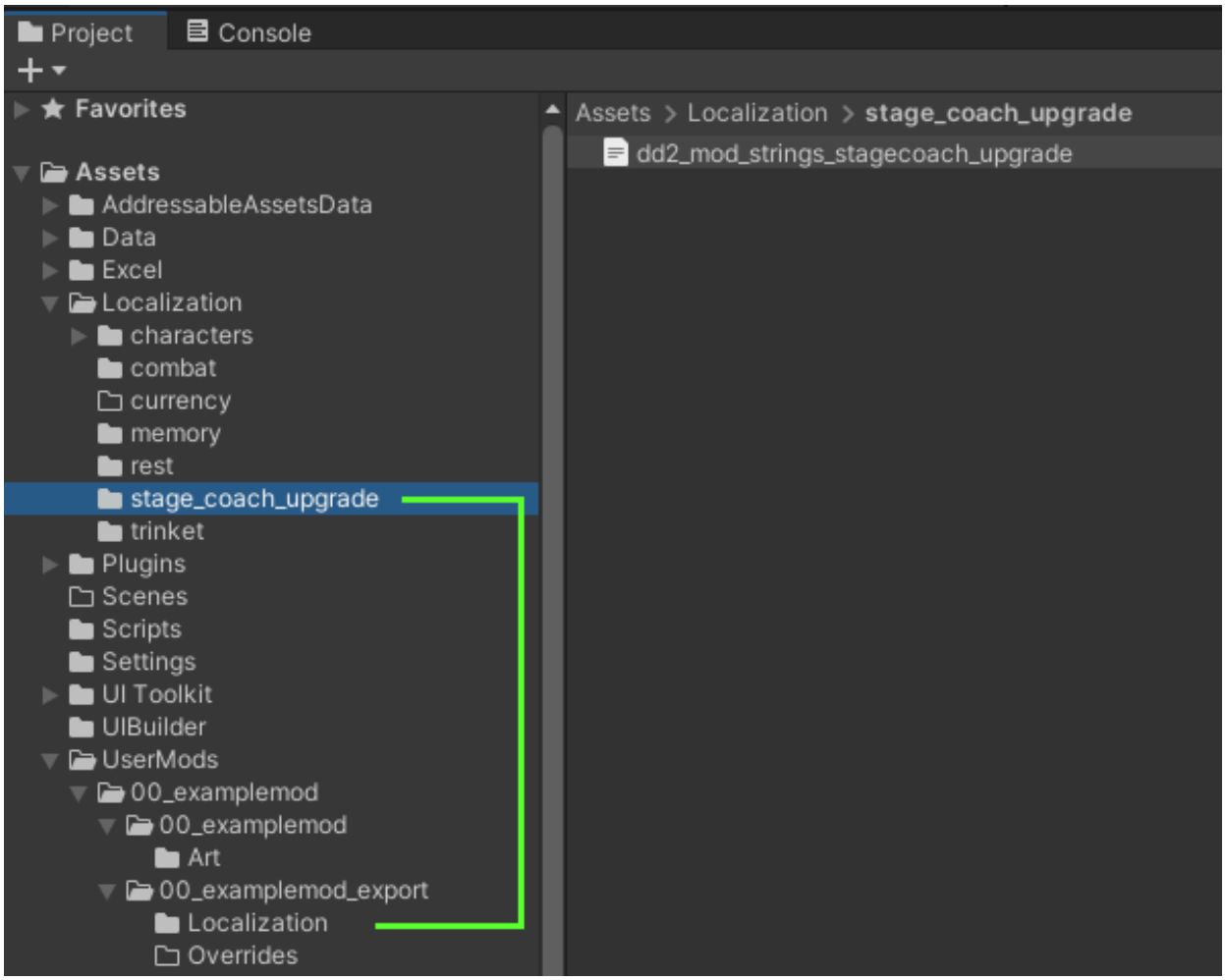
Excel

This folder contains the default CSV files for the various Item Types. The _export folder is pre-populated with example data. The default data can be manipulated for custom pre-populated data when items are created.



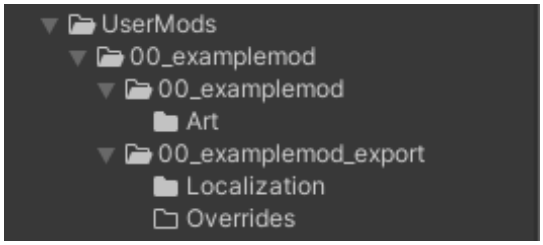
Localization

This folder contains the default TEXT files for the various Item Type strings for their names, descriptions, tooltips, etc. When an item is created the default text files in **UserMods > [MODNAME] > [MODNAME]_export > Localization** will be pre-populated with **Item ID** assigned.



UserMods

This folder contains the mod files that are created when you create a new item and are primarily the files that will be bundled and added to your Workshop Item.
The top level folder has the **MODNAME** that you chose when creating the mod.

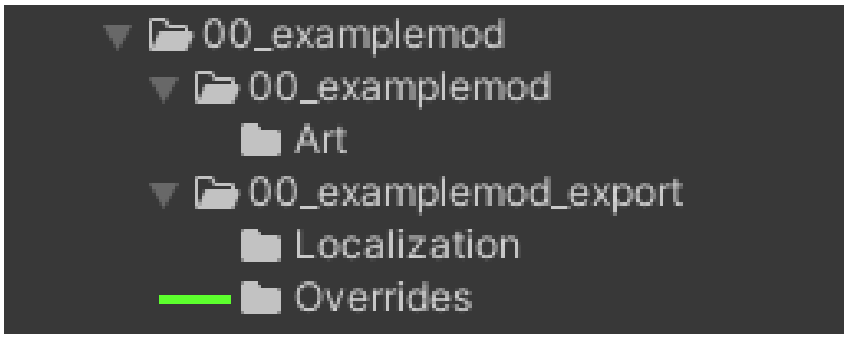


Overriding Data

By default, most of the game data cannot have multiple data pieces of the same types using the same ID. There are exceptions and ways to use the same IDs to append data together OR override base game data.

Certain data types, primarily loot tables and battle configs, can have multiple instances that share the same ID and they will be appended to each other like they're a single table. An example of how we utilize this functionality can be found when searching for **TRINKETS_HERO_ALL** in **loot_data_export_TRINKETS.Group.csv** and **loot_data_export_DLC1.Group.csv**. This is how we append Crusader and Duelist trinkets to the loot tables for hero trinket drops.

For modding purposes, there's an option to override data that share the same type and ID. When an item is created with the **Item Creation Tool**, an empty **Overrides** folder is automatically created (**UserMods > [MODNAME] > [MODNAME]_export > Overrides**). Data files contained in this folder will OVERRIDE any data that shares a type and ID with base game data.



Images can be overwritten as well, you just need to ensure that the **[ITEMID]** prefab file name created when you create a new item **EXACTLY** matches the item ID in the base game you're looking to override. To assign a custom image, follow the steps found in the Item Icons section.

VERY IMPORTANT INFO

By default, the item templates created have all the data contained in a single CSV. If you want to create mods that override existing items, you'll need to create a separate CSV with the OVERRIDE data and move that into the Overrides folder. All other NON-OVERRIDE data should be contained in the default location (**UserMods > [MODNAME] > [MODNAME]_export**)

Creating an Item

Creating the initial files

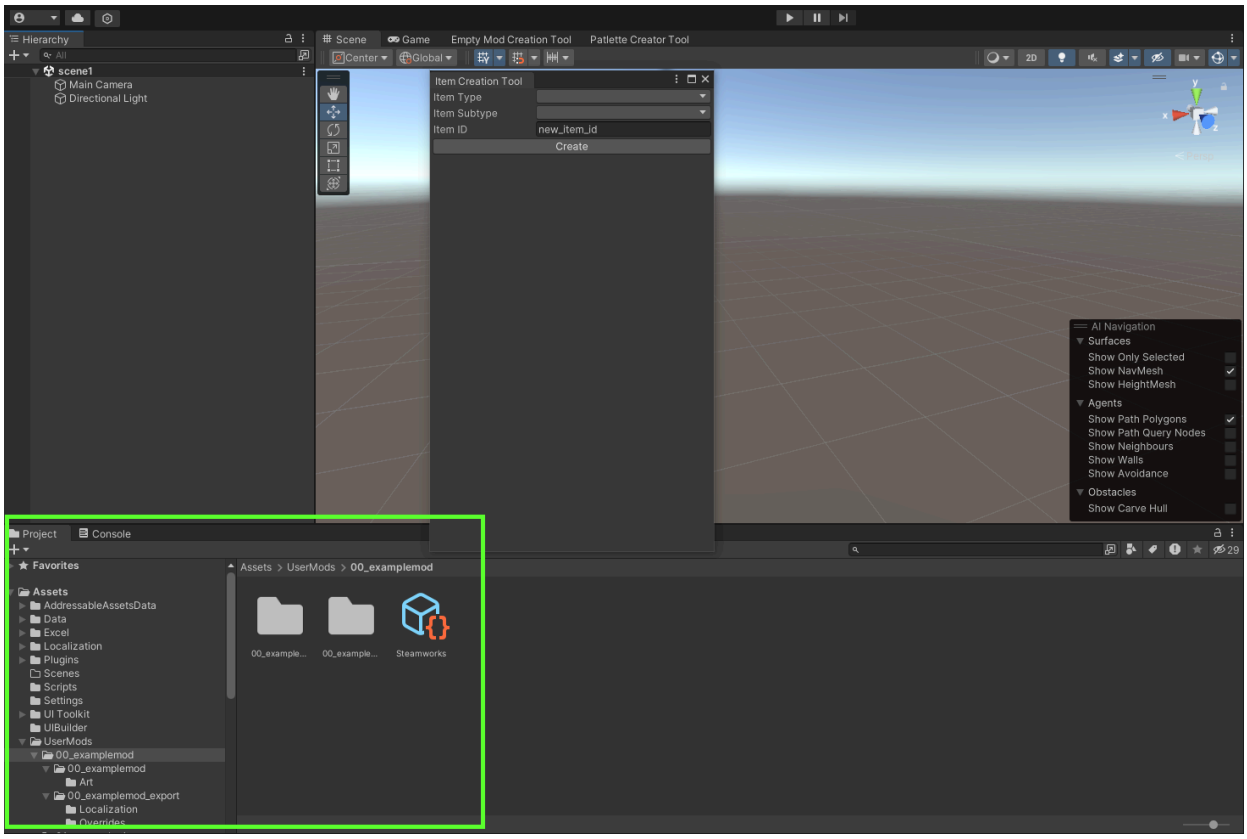
In the **Item Creator Tool** window, assign an **Item Type/Subtype**, enter an **Item ID**, then select **Create** to create your first item (congrats you've made an item!)

VERY IMPORTANT INFO
When assigning an **Item ID**, you want to choose an ID that's unique to the item you're creating OR if you're using the example data, the IDs match Example IDs below based on the Item Type you're creating

Example IDs for Item Types/Subtypes

- trinket: **rh_example_trinket**
- rest: **rh_example_rest_item**
- combat: **rh_example_combat_item**
- stage_coach_upgrade/general: **rh_example_sc_general**
- stage_coach_upgrade/pet: **rh_example_sc_pet**
- stage_coach_upgrade/trophy: **rh_example_sc_trophy**
- stage_coach_upgrade/infernal: **rh_example_sc_infernal**
- stage_coach_upgrade/radiant: **rh_example_sc_radiant**

When creating an item, most of the files will be named based on this **Item ID** and changing them after the fact can be a bit cumbersome, so doing it right initially will help you down the road!



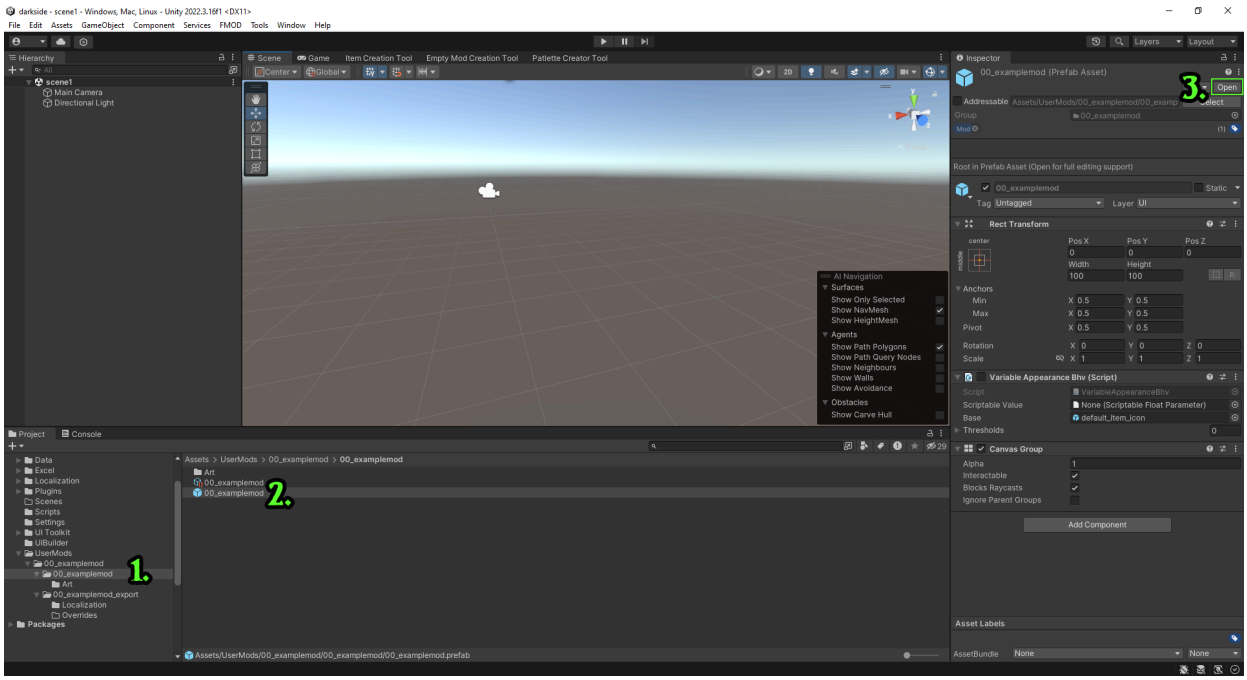
After creating an item, in the **Project** window you'll find the item files you've just created contained within the **UserMods** folder.

Assigning item visuals

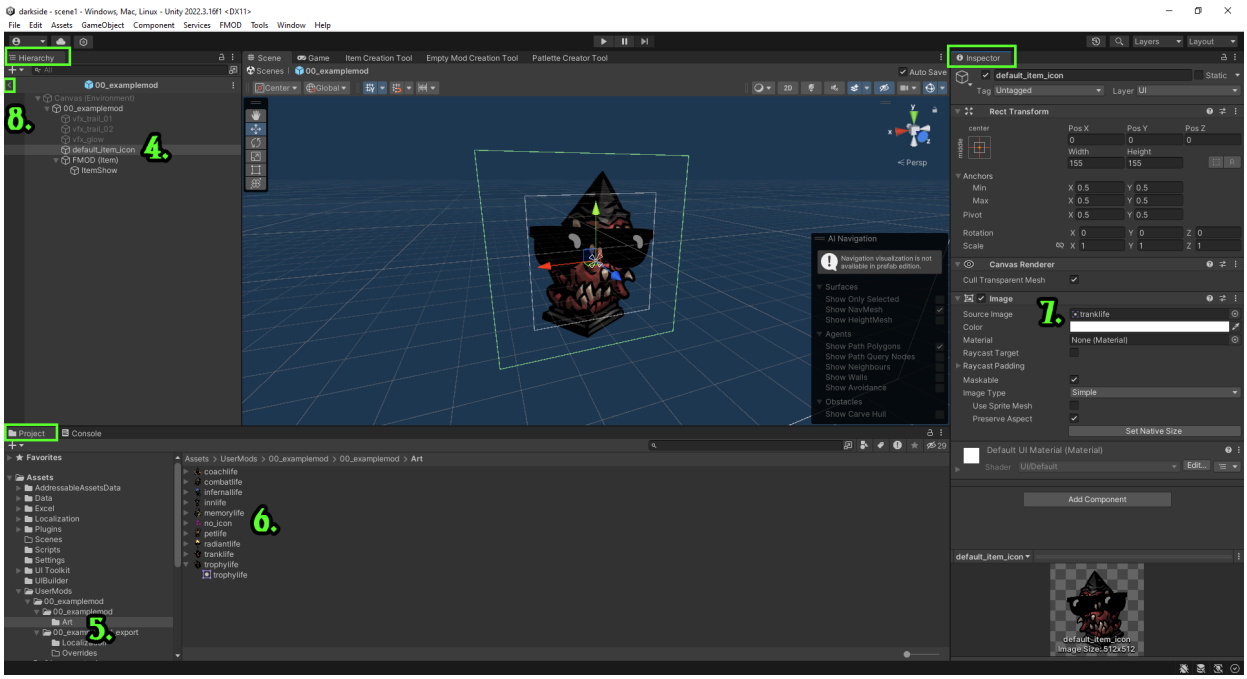
Item Icons

Details for using custom images can be found under Step 6.

1. Find the folder that's named after your Item ID (**UserMods > [MODNAME] > [ITEMID]**)
2. In this folder, select the prefab that's named after your Item ID
3. Open the **[ITEMID]** prefab (In the **Inspector** window, press the **Open** button in the top right corner)



4. In the **Hierarchy** window, select the **default_item_icon** gameobject
5. In the **Project** window, select the **Art** folder (**UserMods > [MODNAME] > [ITEMID] > Art**)
6. From the **Art** folder, select an image
    - **CUSTOM IMAGES** - This is the folder where you can add custom images. The standard **image size for item icons is 512x512**. For reference, you can duplicate an example image, then edit it with your custom image in Photoshop or another photo editing tool, as this will keep Unity settings and will match our image sizes. HOWEVER, if you feel comfortable with arranging the correct settings, feel free to import custom images directly into the appropriate folder locations (see the point below this one!) - in the **Inspector** Window, make sure to set the “Texture Type” of the image to “Sprite (2D and UI)” and “Pixels Per Unit” to 1.
    - **VERY IMPORTANT INFO** - The image used MUST be contained in **UserMods > [MODNAME] > [ITEMID] > Art** in order to be bundled correctly
7. With the selected image, drag/drop the image onto the empty **Source Image** field found in the **Inspector** window
8. **SAVE THE PREFAB** - To ensure this is saved, close the prefab by clicking the back arrow button < in the **Hierarchy** window

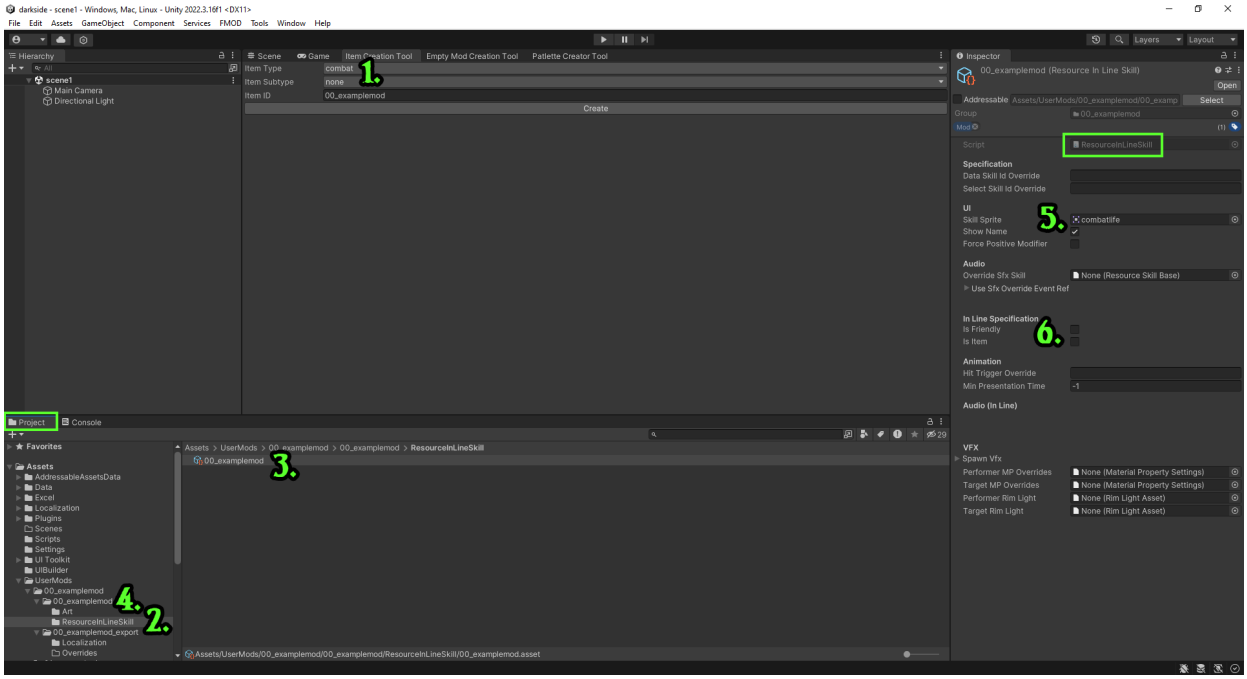


Combat Item Skill Icons

In addition to the item icon, Combat items also have a skill icon. Under the hood Combat item actions taken in combat are technically skills and have some overlap with skills for requirements (skill data, skill
ResourceAssets

1. This is **SPECIFICALLY** if you've created a combat item
2. Find and select the ResourceInLineSkill folder (**UserMods > [MODNAME] > [ITEMID] > ResourceInLineSkill**)
    a. This folder doesn't exist for most item types as this is the asset for connecting the item for item use in combat
3. Select the **ResourceInLineSkill** asset name **[ITEMID]** contained in the **ResourceInLineSkill** folder. You will see the details for this asset in the **Inspector** window
    a. You will likely see error messages in the **Console** window when selecting this file, this is normal!
4. In the **Project** window, from the **Art** folder (**UserMods > [MODNAME] > [ITEMID] > Art)**, select an image
    a. **VERY IMPORTANT INFO** - The image used MUST be contained in **UserMods > [MODNAME] > [ITEMID] > Art** in order to be bundled correctly
    b. This image **does NOT** have to match the item icon
5. With the selected image, drag/drop the image onto the empty **Skill Sprite** field found in the **Inspector** window
6. In the **Inspector** window with the **ResourceInLineSkill** selected, find the checkboxes under **In Line Specification**
    a. Ensure the **Is Item** check box is marked **TRUE** (checked)

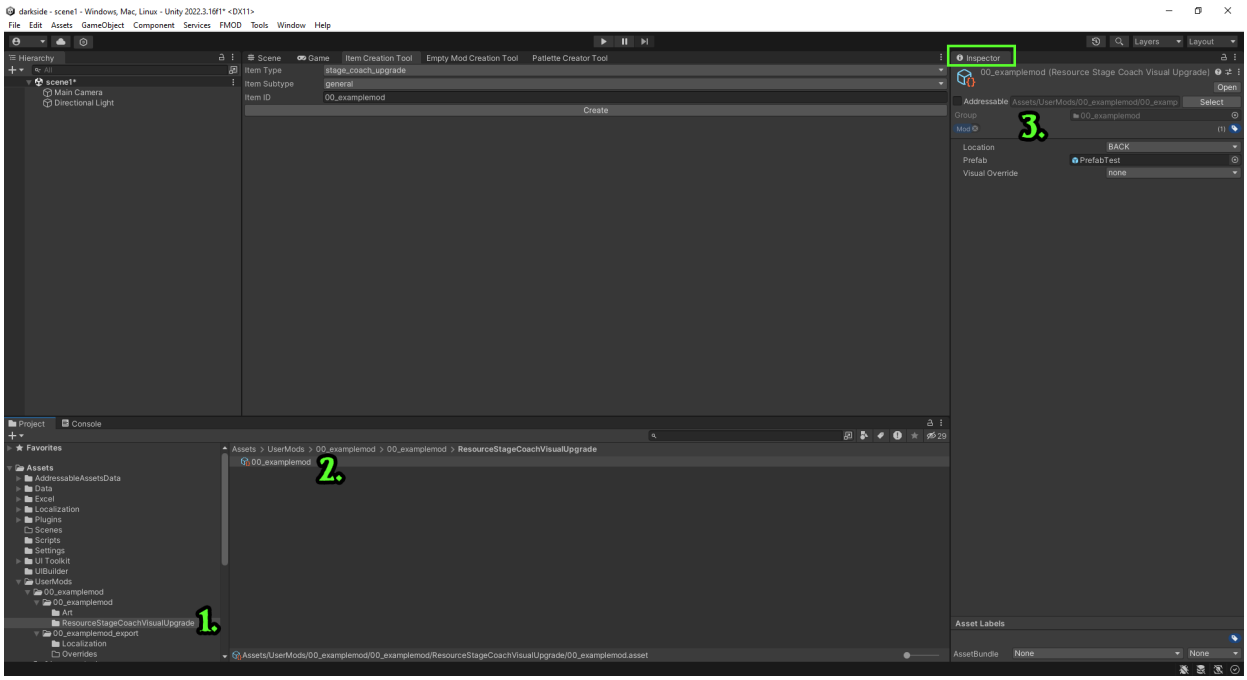
b. If the combat item targets a hero, **Is Friendly** should be marked **TRUE** (checked)
c. If the combat item targets an enemy, **If Friendly** should be marked **FALSE** (unchecked)



Stagecoach attachments (Preset and Custom)

When creating a **stagecoach** item (any sub_type), there is an asset that allows for stagecoach attachments, preset or custom.

1. In the **ResourceStageCoachVisualUpgrade** folder (**UserMods > [MODNAME] > [ITEMID] > ResourceStageCoachVisualUpgrade**)
2. Select the asset **[MODNAME]**
3. In the **Inspector** window you will find options for the following
    a. Location: various attachment points on the stagecoach (currently this list does NOT include attachment points for pet cages, torches, or trophies)
    b. Prefab: This is used for **custom** stagecoach attachments (see below)
    c. Visual Override: This is used to select **preset** stagecoach attachments (see below)

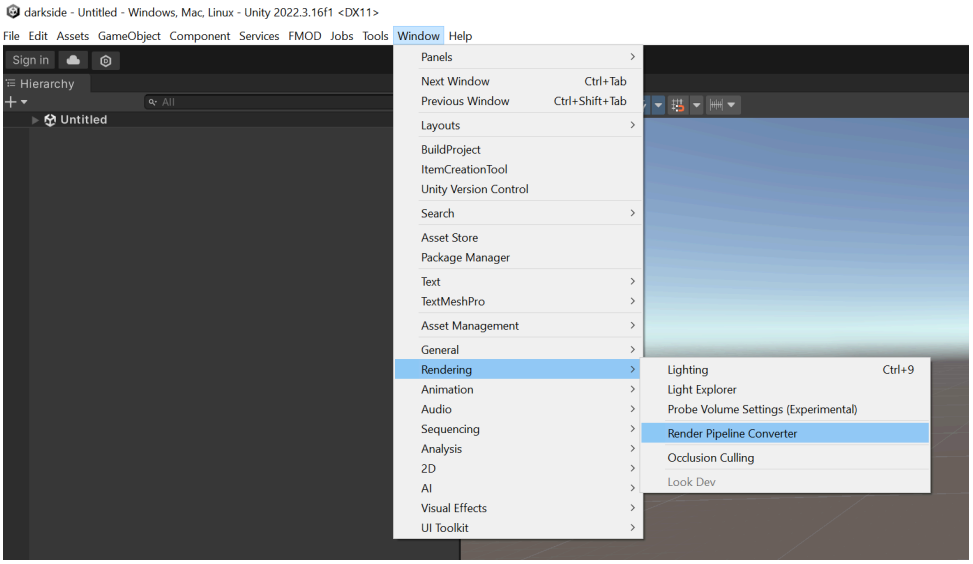
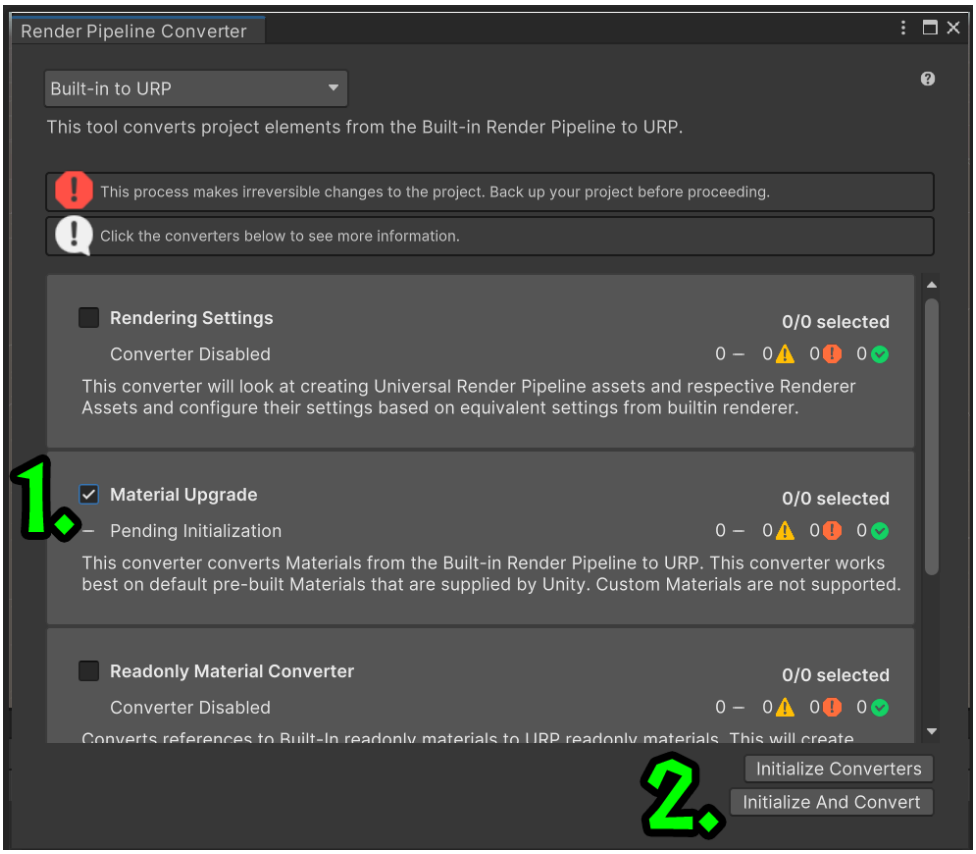


Custom stagecoach attachments

- Create a prefab in **UserMods > [MODNAME] > [ITEMID] > Art**
    - The prefab MUST be contained in **UserMods > [MODNAME] > [ITEMID] > Art** in order to be bundled correctly
- Drag/drop the prefab onto the **Prefab** field on the **ResourceStageCoachVisualUpgrade** asset
- Ensure the **Visual Override** is set to **none**
- Select an attachment **Location**
    - This location can be used as a starting point THEN in the prefab, you can offset any visuals to ANY spot you desire
- Be careful to not have any Camera objects attached to your prefab. The game will try to set it as the main camera, resulting in a black screen.

Things to keep in mind when creating the prefab

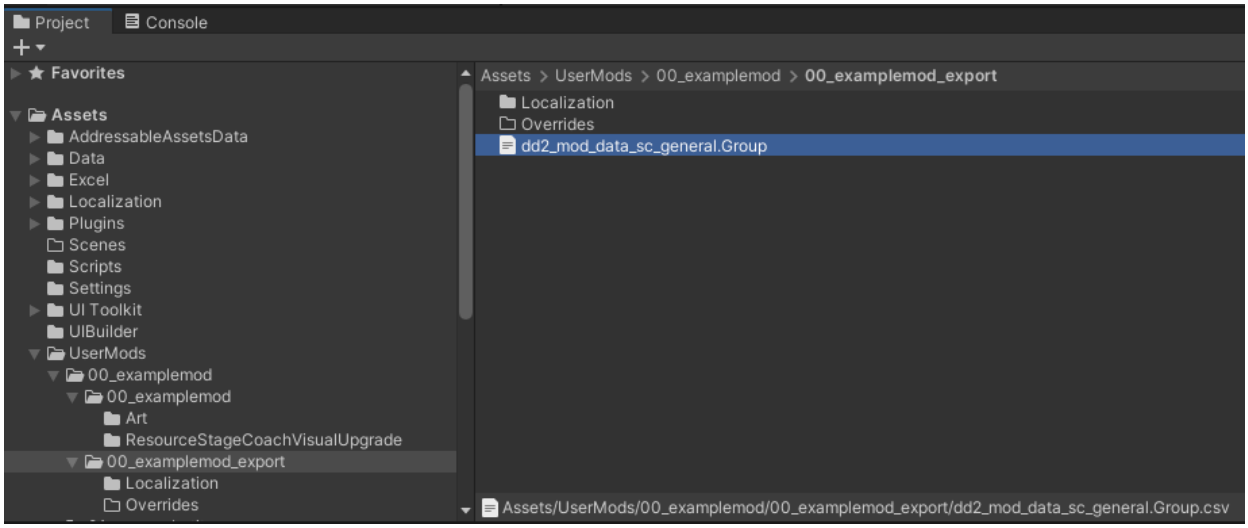
- Models can be added to the prefab, but you need to ensure that any materials are also added and contained in **UserMods > [MODNAME] > [ITEMID] > Art**. To keep things organized, we recommend keeping all prefab assets (models, materials, etc.) in a subfolder within the **Art** folder.
- If your materials are NOT set up for **Universal Render Pipeline**, you may need to run the **Render Pipeline Converter**. Check **Material Upgrade** then **Initialize And Convert** to convert all non-URP materials to URP supported materials.
    - Note: Custom stagecoach items are automatically converted to URP when the mod is uploaded. No need to manually convert materials if you're making a custom coach item.

- 
darkside - Untitled - Windows, Mac, Linux - Unity 2022.3.16f1 <DX11>
File Edit Assets GameObject Component Services FMOD Jobs Tools Window Help
Sign in
Hierarchy
Untitled
Panels
Next Window Ctrl+Tab
Previous Window Ctrl+Shift+Tab
Layouts
BuildProject
ItemCreationTool
Unity Version Control
Search
Asset Store
Package Manager
Text
TextMeshPro
Asset Management
General
Rendering
Animation
Audio
Sequencing
Analysis
2D
AI
Visual Effects
UI Toolkit
Lighting Ctrl+9
Light Explorer
Probe Volume Settings (Experimental)
Render Pipeline Converter
Occlusion Culling
Look Dev
- 
Render Pipeline Converter
Built-in to URP
This tool converts project elements from the Built-in Render Pipeline to URP.
This process makes irreversible changes to the project. Back up your project before proceeding.
Click the converters below to see more information.
Rendering Settings
0/0 selected
Converter Disabled
This converter will look at creating Universal Render Pipeline assets and respective Renderer Assets and configure their settings based on equivalent settings from builtin renderer.
1.
Material Upgrade
0/0 selected
Pending Initialization
This converter converts Materials from the Built-in Render Pipeline to URP. This converter works best on default pre-built Materials that are supplied by Unity. Custom Materials are not supported.
Readonly Material Converter
0/0 selected
Converter Disabled
2.
Initialize Converters
Initialize And Convert

Prepping/editing item data and functionality

Editing CSVs

This is the most direct way to edit item data. In the **[MODNAME]_export** folder (**UserMods > [MODNAME] > [MODNAME]_export**), you will find a CSV file pre-populated with data with basic functionality for each item type.



This file can be opened and edited with any software used for text editing (Notepad, Notepad++, Excel, VS Code, etc.)

There is A LOT to understand when it comes to this data. If you're taking this approach, it would be best to start with the default CSV file, edit the numbers/IDs there to build a familiarity with the data structure.

Referencing the base game data is an excellent way to create specific functionality for any item mods.

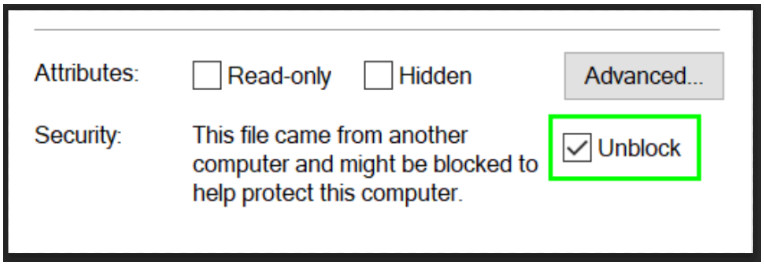
Using the Excel exporter tool

We've provided access to a supplementary XLSM **dd2_mod_data_exporter** that requires Excel to open and export CSVs. This sheet PRIMARILY supports NEW and APPENDED data. Overriding existing data requires a bit more work.

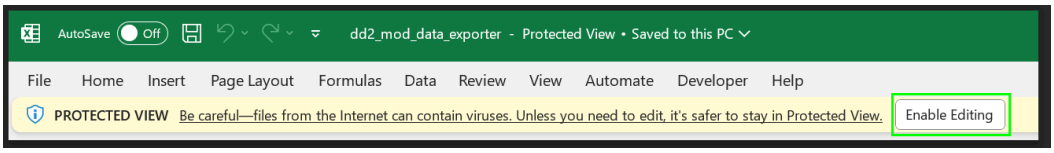
VERY IMPORTANT INFO
To utilize the export macros for exporting you need to do the following

1. After downloading the file
    a. RIGHT-CLICK the file and open **Properties**
    b. At the bottom of **Properties**, under **Security** mark **Unblock** as **TRUE** (checked)

c.

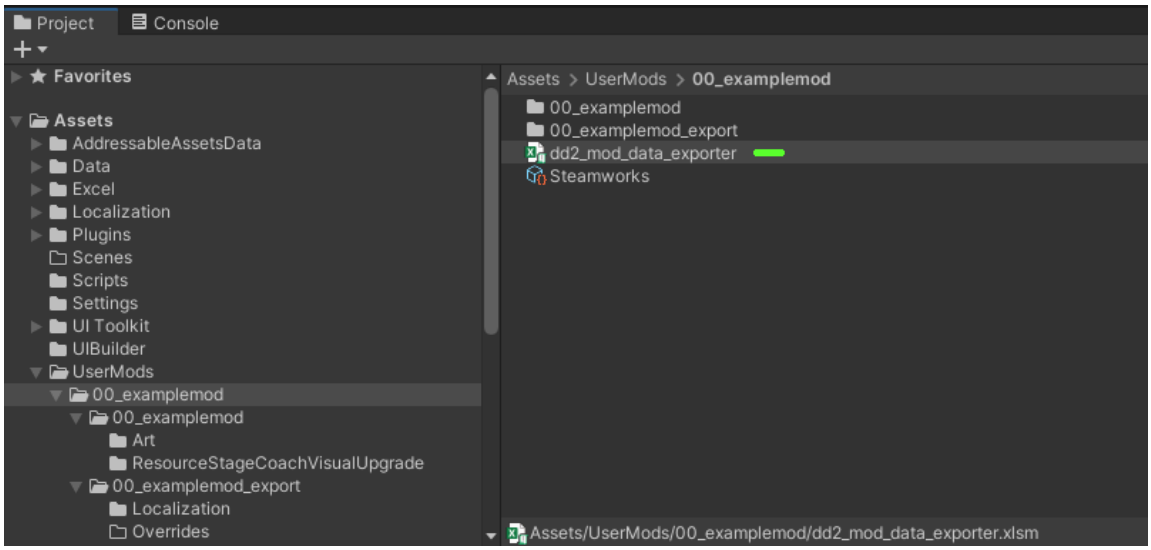


2. After opening the file
    a. Click **Enable Editing** on the top bar (this allow the macro button for Export Group and Export Multiple to function
    b.



Using dd2_mod_data_exporter in Darkside

VERY IMPORTANT INFO Drag/drop the XLSM file into your **[MODNAME]** folder (**UserMods > [MODNAME]**)



What's contained in dd2_mod_data_exporter

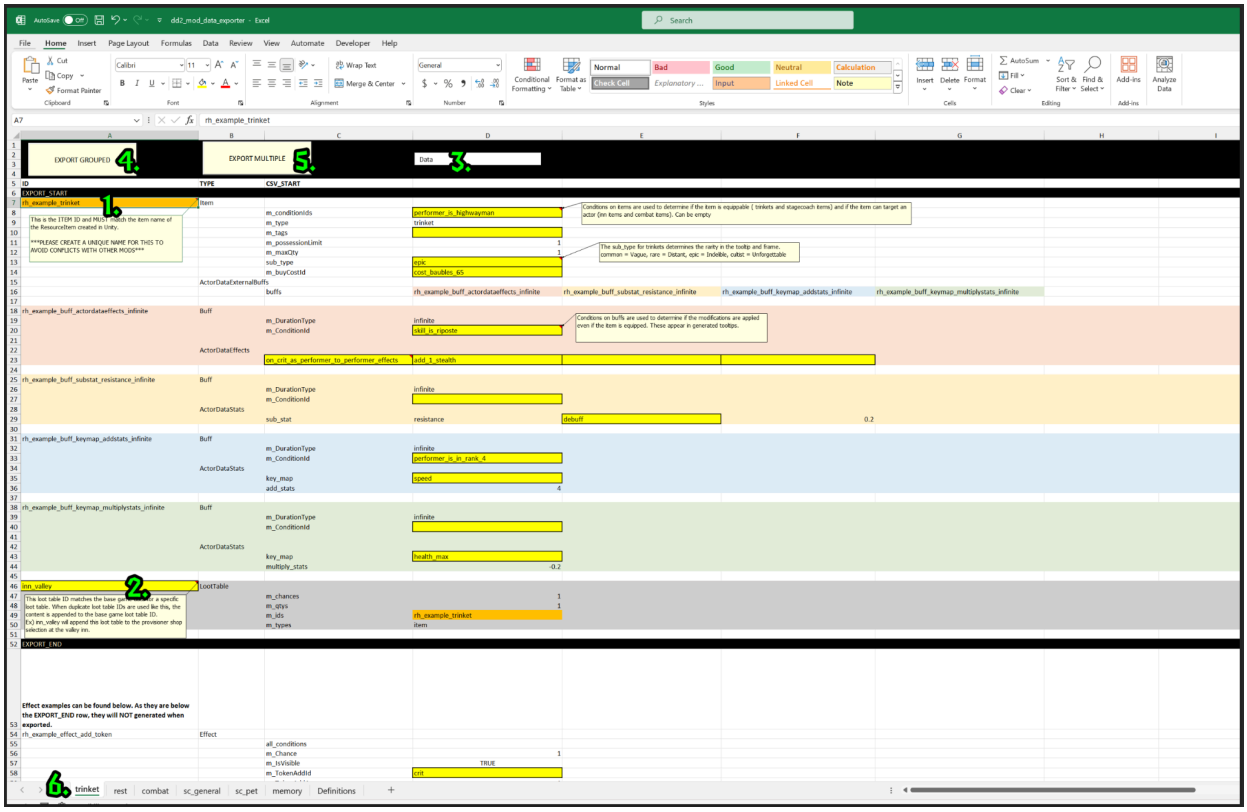
There are several example items that match the default CSV files created when you make a new item using the **Item Creation Tool**; trinket, rest, combat, sc_general, and sc_pet

VERY IMPORTANT INFO

All highlighted cells in **dd2_mod_data_exporter** have drop down options. The drop down options point to lists found on the **Definitions** tab of the sheet. Entries can be added here as well to appear in the drop down selections if you'd like additional options.

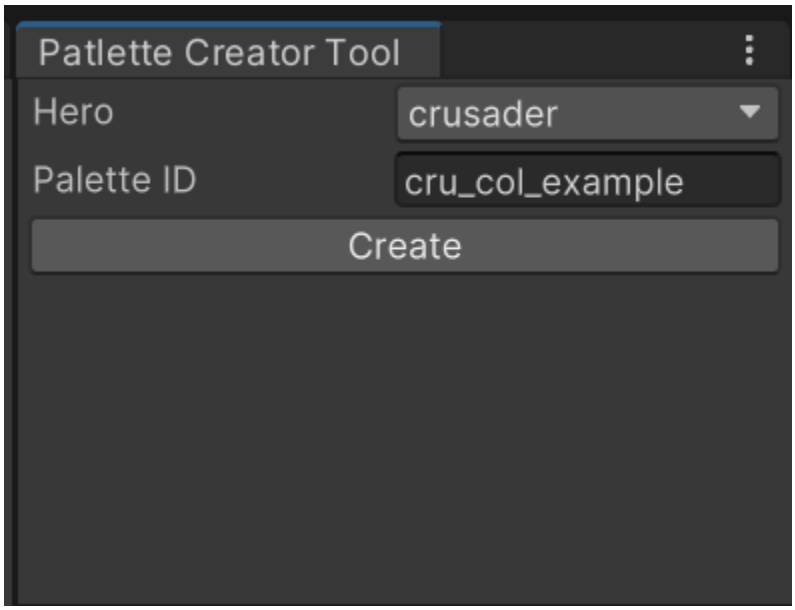
1. As long as you match the **Item ID** assigned with the **Item Creation Tool** to the **Item ID** in **dd2_mod_data_export**, the data should all be connected respective to the various **Item Types/Subtypes**

2. The default data assigns ALL of these items to the valley inn shop. The drop down allows you to change which loot table this item will be appended to.
3. This text field defines the folder location where the CSV will be exported to. As long as **dd2_mod_data_export** is contained in the proper [MODNAME] folder, this should have **[MODNAME]_export** in the field
4. Once all your data has been entered as desired, clicking EXPORT GROUPED will export and override the CSV file contained in the **UserMods > [MODNAME] > [MODNAME]_export** folder
5. **EXPORT MULTIPLE** will create a CSV file for EACH data type within the tab
    a. In the example below, there will be 6 files created (1 for the item, 4 for the buffs, and 1 for the loot table)
    b. We recommend using **EXPORT GROUPED** as this keeps all the data for the table contained in a single file
    c. **NOTE FOR OVERRIDES** - This export option may be useful to separate pieces of data to more easily move into Overrides and default data folders
6. Each tab is used for each Item Type example and will create a self contained CSV file when using **EXPORT GROUPED**.
    a. We recommend starting with a single item type, thus only utilizing a single tab, for a single mod. Once you start to become more familiar with this process and modding, you could create a mod that contains several items.



Creating custom hero palettes

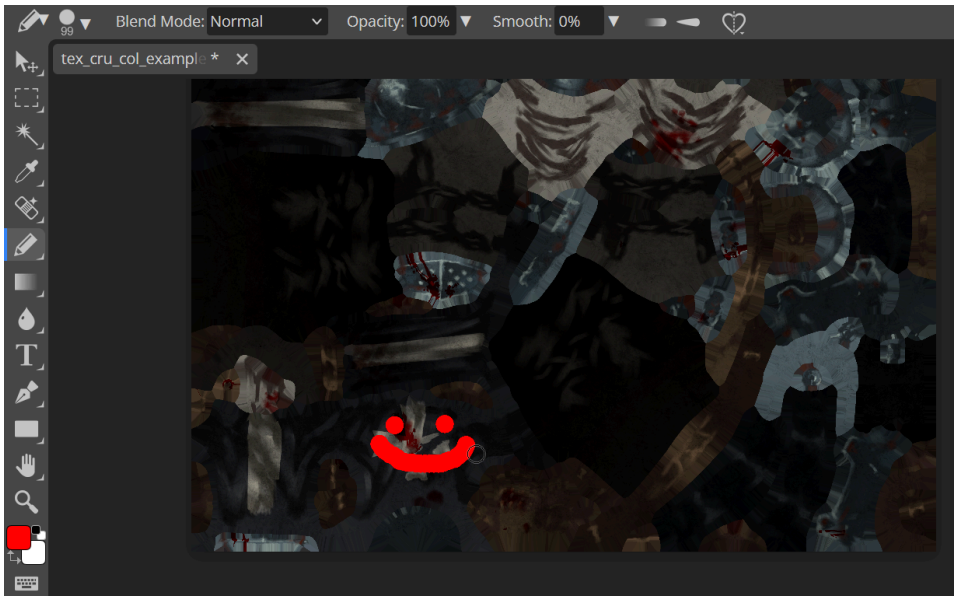
In the Palette Creator tab, select the hero you wish to make a custom palette for, give it an ID, and press Create.



This will create a new folder named after the ID you entered. In this folder, you should find an image titled tex_[your_id]. This is the “col” texture for the hero you selected.

Textures in DD2 are broken into two parts, the col and the ink. The ink texture is the black outlines that are layered on top of the col texture. The col texture contains all of the color and details. Custom palettes use a custom col texture.

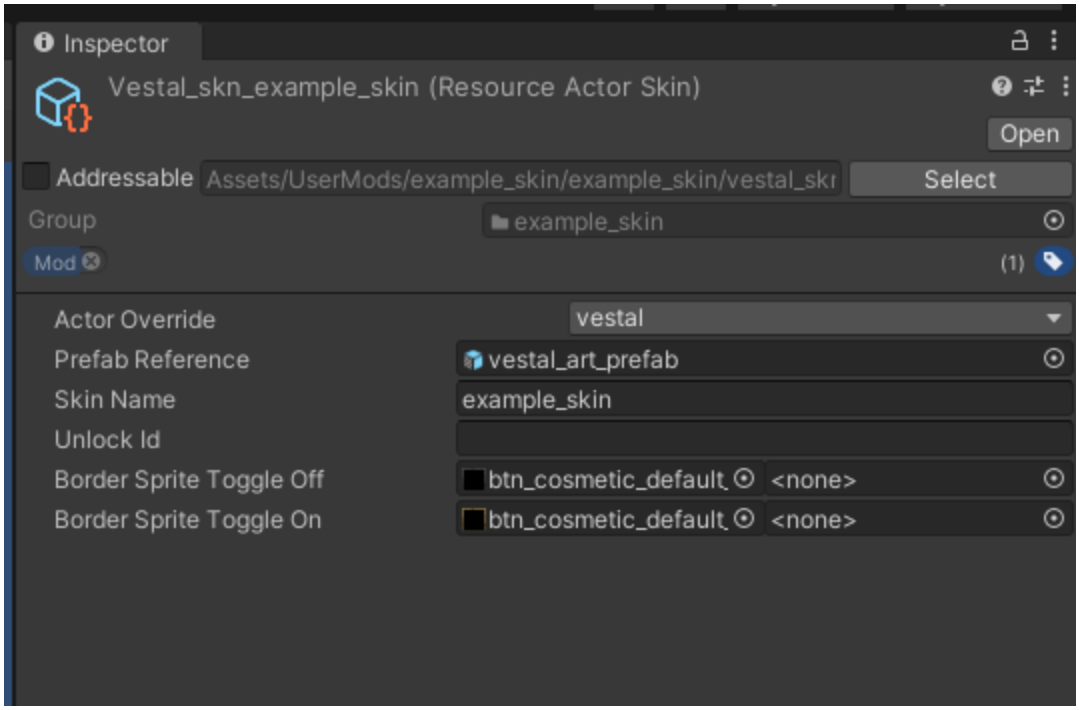
Open the ‘tex_[your_id]’ texture in a photo editing program of your choice and edit the texture to your liking (You can right-click on the texture in Unity and select “Show in Explorer” to find it quickly.) Make sure to save over the original image.



That’s pretty much it! You can now use the Steamworks tool to upload the palette to the workshop. You can test if it worked by clicking through all of the hero’s Palettes at the crossroads.

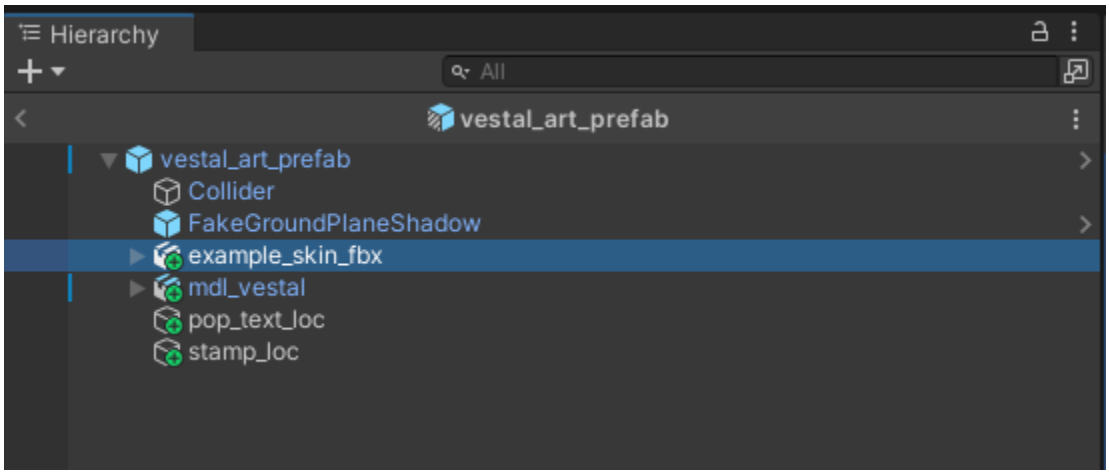
Hero Skins / Custom Hero Models

Use the Hero Skin Creator Tool to make a new mod. In the mod folder, you will have a couple files. The main one is the skin Resource, which has the default name: (hero)_skn_(modID). In the skin Resource, you can assign the art prefab (the prefab has the skin's 3D model, the skeleton, vfx, Unity components, etc.), a name id (note: you still need to do Localization), an Unlock ID (leave it blank if you want the skin unlocked by default), and custom button sprites for the cosmetics tab.



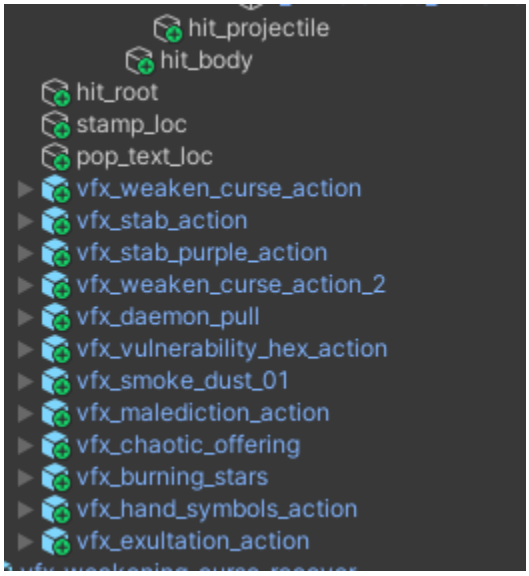
If you want to just add things to the skin, you can just drag them onto the proper bone inside the art prefab. Most likely you will want to do some custom modeling. In this case, you should take the mdl_heroname.fbx file, and import it into your 3d program of choice (e.g. Blender). Create your new model, rig it to the skeleton, and export it back into darkside (see import and export settings below).

Drag your .fbx into the project view in darkside to import it. Then open the art prefab, and drag the imported fbx into it, on the same level as mdl_heroname, which your model will eventually be replacing. Click on mdl_heroname, look at the components in the inspector, and add the same components to your model, most likely these will be: an Animator (with an AnimationController assigned), TimelinePropertyMapBhv, and AnimatorStateSender. Also check that your scale matches, several of the heroes are scaled slightly differently (e.g. vestal is scaled to 1.03).



Click on the meshes in the mdl_heroname, and in the inspector on the right, click on the materials they use, and copy them into your mod folder if you want to make changes to them without altering the base versions in darkside. Click on your copied materials, and drag in your custom textures for Base and Ink, then assign the materials to your own meshes in the art prefab. While going through the meshes, you should also be checking for components and adding them to your own. It varies by hero which components are used. Weapons have a MaterialPropertyBhv with the text 'weapon' in the filter field, and occasionally some will have different components, e.g. the Occultist's knife has a ToggleActiveStateByAnimationCurve.

Right click the skeleton (SHJntGrp) and click 'Select Children' to expand the entire hierarchy. Look through the skeleton for all the vfx (which have blue boxes next to them) and all the objects with white names, and move them over to the same bone in your own fbx. Click on the art prefab, and in the BoneRemapping component, check that all the remaps are properly assigned to the white named objects (hit_body, etc.) in your own model. These manage where visual effects that don't come from the character appear, such as enemy attacks or damage numbers.



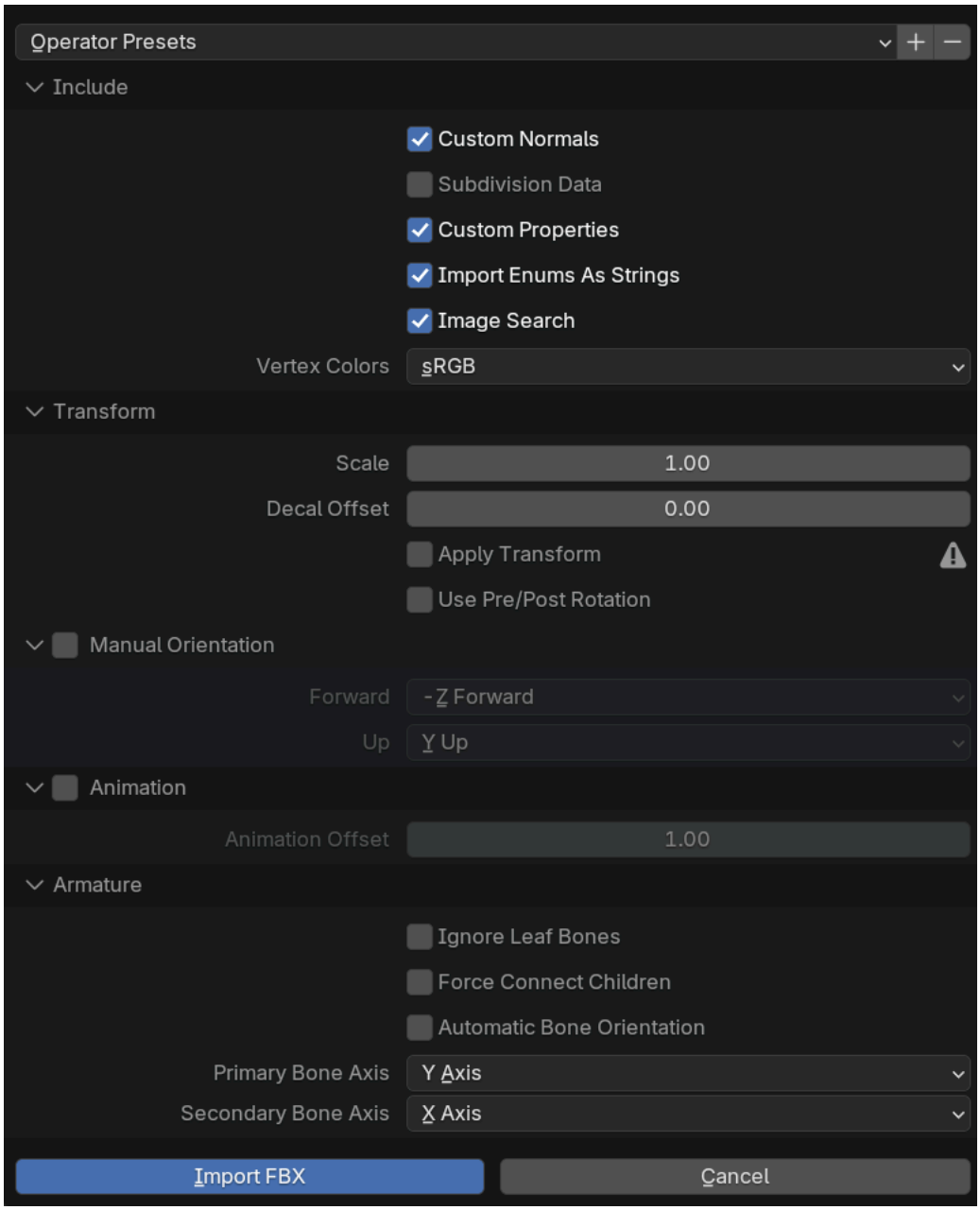
You can now delete the original mdl_heroname, at which point the art prefab should consist of a Collider, a FakeGroundPlaneShadow, your model, and various hero-specific objects. The game will simply load the art prefab as the model for the hero, so you don't have to follow these steps exactly for it to work, and you're welcome to do other things like change vfx or animations in the animator (make sure you copy it first, or use a Right Click > Create > Animator Override Controller).

At this point your skin mod should be good to go.

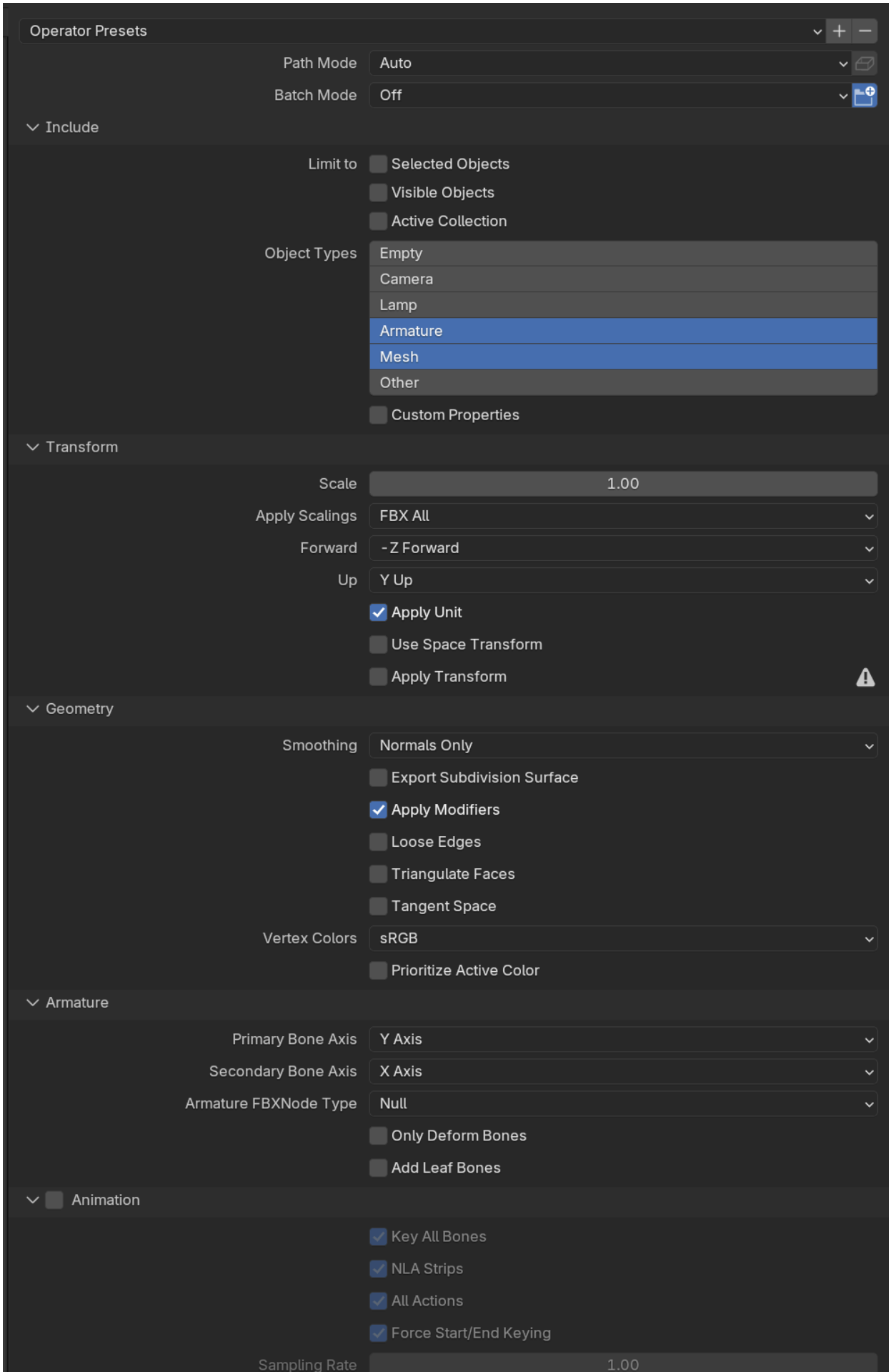
Blender Settings

After importing, and again before exporting, reset the armature poses by selecting the armature in object mode, switching to pose mode, and pressing: A, Alt+R, Alt+S, Alt+G.
Before exporting, set Armature X rotation to 0.

Import Settings



Export Settings:


Operator Presets
Path Mode
Auto
Batch Mode
Off
Include
Limit to
Selected Objects
Visible Objects
Active Collection
Object Types
Empty
Camera
Lamp
Armature
Mesh
Other
Custom Properties
Transform
Scale
1.00
Apply Scalings
FBX All
Forward
-Z Forward
Up
Y Up
Apply Unit
Use Space Transform
Apply Transform
Geometry
Smoothing
Normals Only
Export Subdivision Surface
Apply Modifiers
Loose Edges
Triangulate Faces
Tangent Space
Vertex Colors
sRGB
Prioritize Active Color
Armature
Primary Bone Axis
Y Axis
Secondary Bone Axis
X Axis
Armature FBXNode Type
Null
Only Deform Bones
Add Leaf Bones
Animation
Key All Bones
NLA Strips
All Actions
Force Start/End Keying
Sampling Rate
1.00

Updating names, descriptions, and other strings

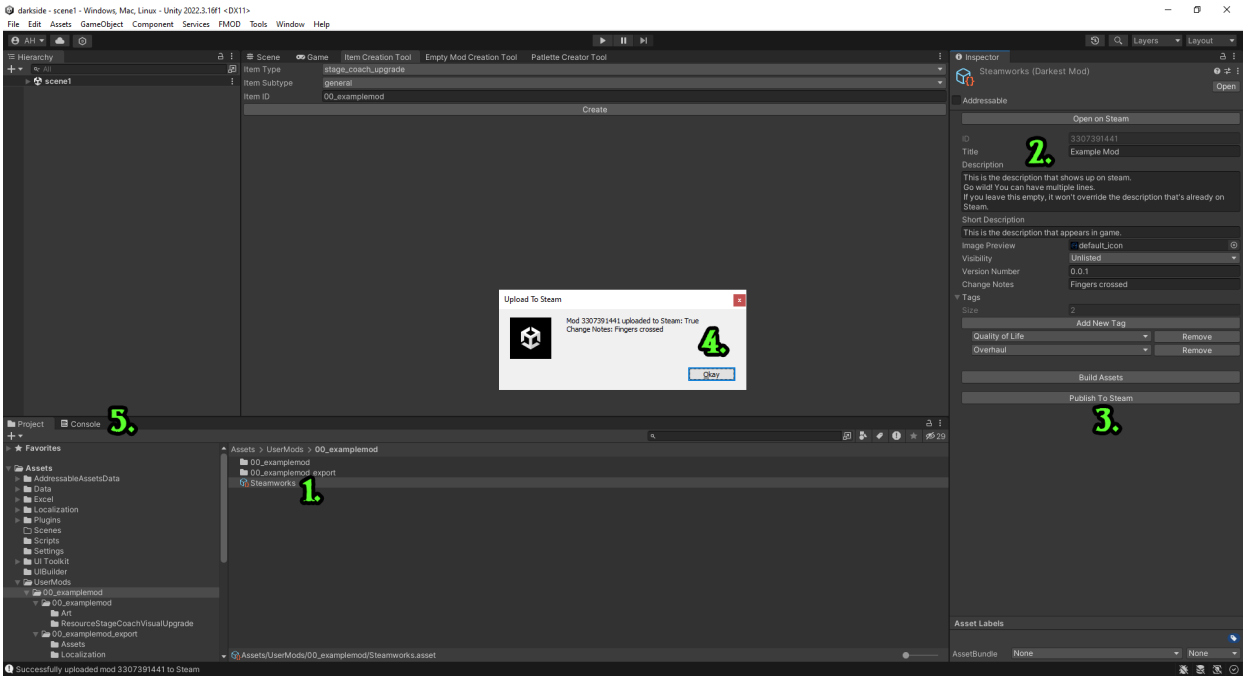
In the **Localization** folder (**UserMods > [MODNAME] > [MODNAME]_export > Localization**), there's a pre-populated text file with string IDs that have automatically been created to match the Item ID when the item was created. ex)
item_name_rh_example_combat_item=rh_example_combat_item

Each Item Type has slightly different string requirements. Reference and edit the pre-populated data however you see fit.

Publishing to Steam Workshop

Once you're satisfied with the item data, you can begin the process of uploading your mod to the Steam Workshop!

1. In the **[MODNAME]** folder (**UserMods > [MODNAME]**), select the file called **Steamworks**. The details will appear in the **Inspector** window
2. Fill out the fields for your mod. These details will appear on the Steam Workshop page
    a. This info will NOT appear in the game and does not need to match any of the item data
    b. The image used for the Image Preview field can be anywhere in the project files (it does NOT need to be in the Art folder like images used for actual item data)
3. Once you're satisfied with the **Steamworks** details, click **Publish to Steam**
4. Assuming all went well, you should see a popup with the title **Upload To Steam** and in the description you should see the change notes you included in the Steamworks details
    a. **VERY IMPORTANT INFO** - The first time you hit **Publish to Steam**, you'll see a prompt for **Addressable Reports**. You can hit Yes or No, it doesn't affect the outcome when publishing, selecting Yes just provides additional details found in the **Addressable Report** window.
5. The text in the popup should say "Mod [**ID**] uploaded to Steam: **True**". If it says False, your mod failed to upload. You should get an error message in the Unity console in that case.

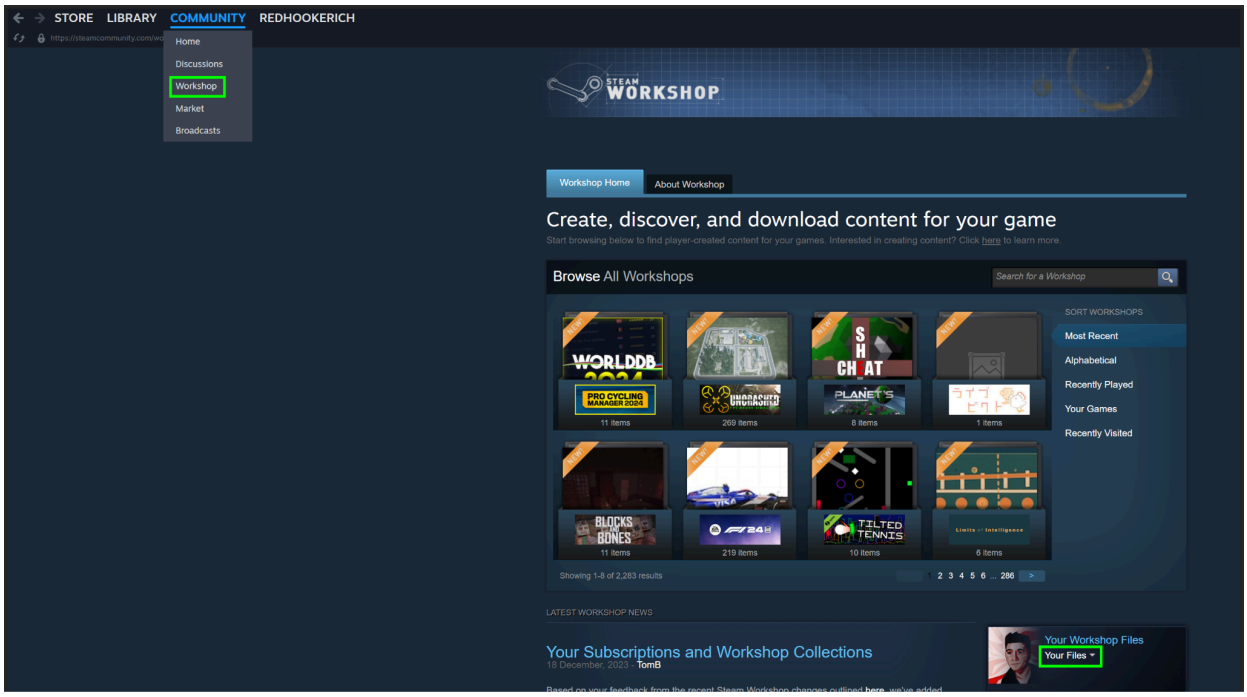


Testing your mod

Congratulations on creating your mod!

Finding your mod on Steam

In **Steam** you can find your **Workshop Files** under **Community > Workshop** then on this page, select the link for **Your Files**



Subscribing (downloading) your mod

On the **Your Files** page, you'll find all of your files (go figure). This includes ANY item created with the **Item Creation Tool**, even before clicking **Publish to Steam** for the first time.

Select the mod you'd like to test and hit **Subscribe** to download the files to the appropriate file locations

redhookerich » Workshop Items

Filter by game: Select a game Show: By redhookerich redhookerich's Favorites

Screenshots | Artwork | Videos | Workshop Items | Merchandise | Collections | Guides

Showing 1-7 of 7 entries



Hale Draught Override Test
Darkest Dungeon® II



Duck Tails
Darkest Dungeon® II



rh_example_memory
Darkest Dungeon® II



rh_example_sc_general
Darkest Dungeon® II



rh_example_combat_item
Darkest Dungeon® II



rh_example_rest_item
Darkest Dungeon® II



rh_example_trinket
Darkest Dungeon® II

Per page: 9 **18** **30**



Your Workshop
0 Followers

View Legal Agreement

Items
Favorited

Subscribed Items
Played

Browse the Workshop:

STEAM WORKSHOP

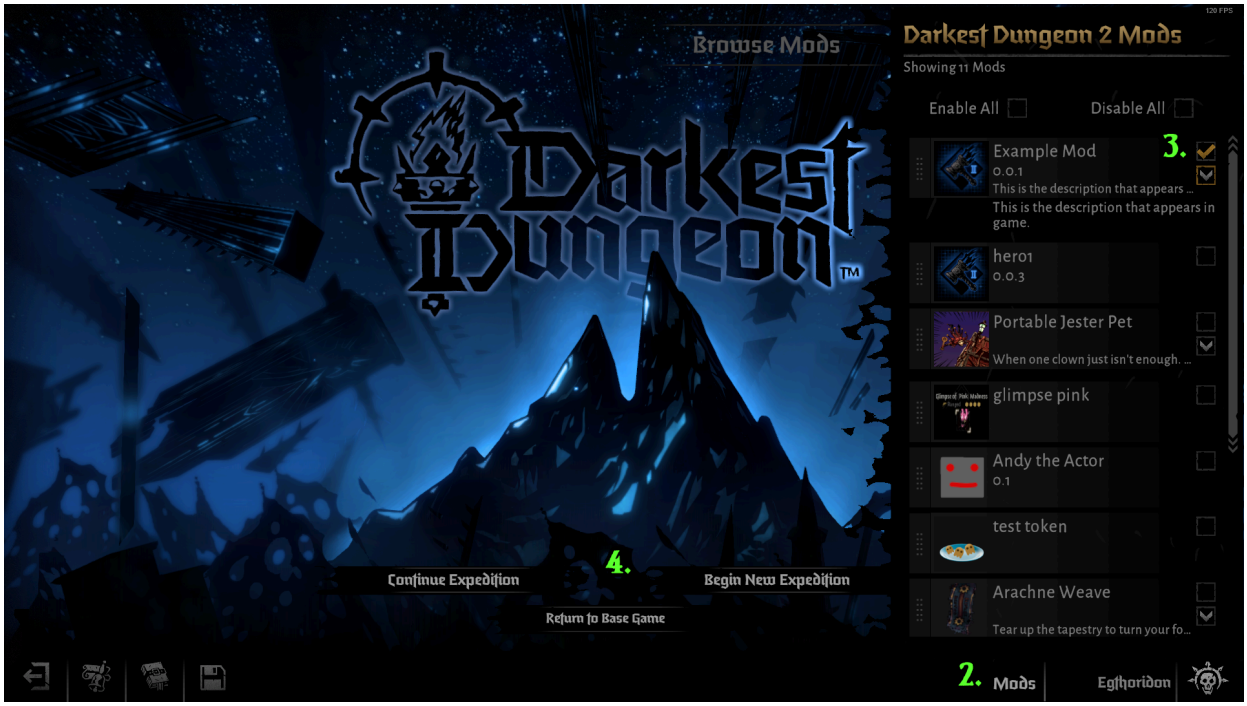


Viewing your mod in game

Run **Darkest Dungeon 2** on a mod supported build

1. On the main menu screen, you'll find an option for Mods
2. Once in the mods menu (the dark side of the mountain), click Mods to open your mod list.
3. Check the mods you want to use
4. Select **Continue Expedition** or **Begin New Expedition**





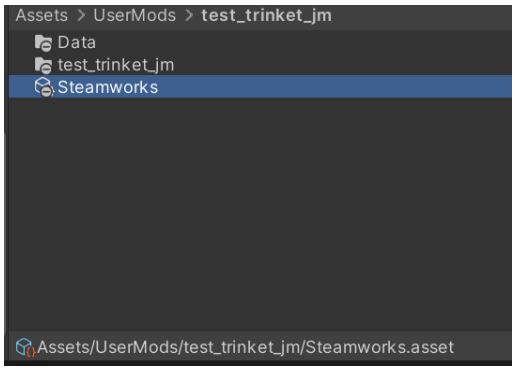
For mods to run, you have to load your run via the Mods menu. Loading a run via the regular menu will not enable any mods.

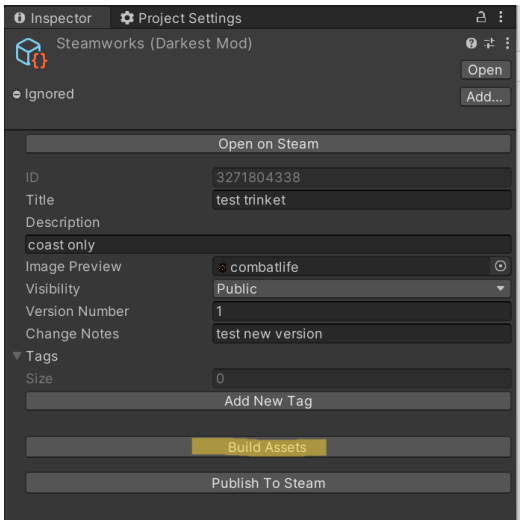
After playing and upon arriving at the valley inn, assuming you're using example item data, your item mod should appear in the valley inn shop!

Local Mod Testing

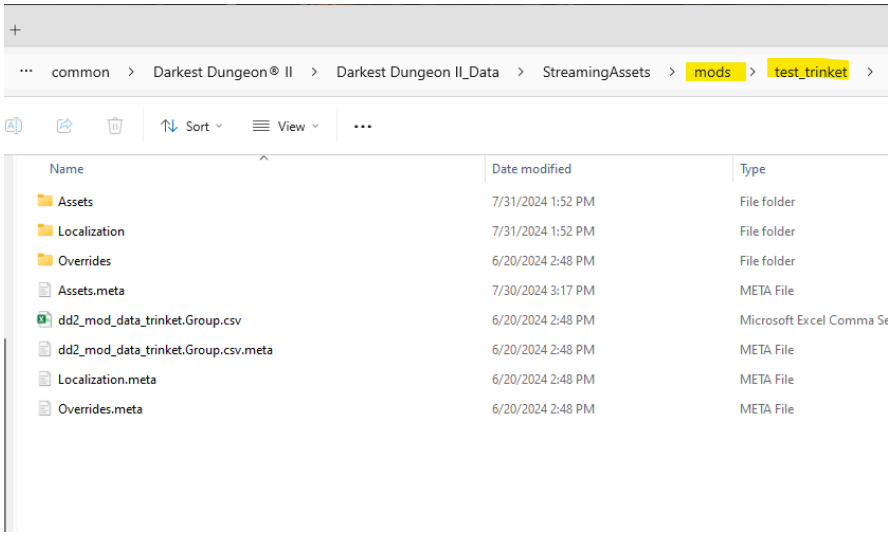
Here are instructions to help you test your mods without having to publish to Steam. You can do this by building the assets.

Inside the mod you want to test, find the Steamworks asset for the mod you want to build and select it [Assets/UserMods/[mod]/Steamworks.asset. Select the Build Assets button in the Inspector.

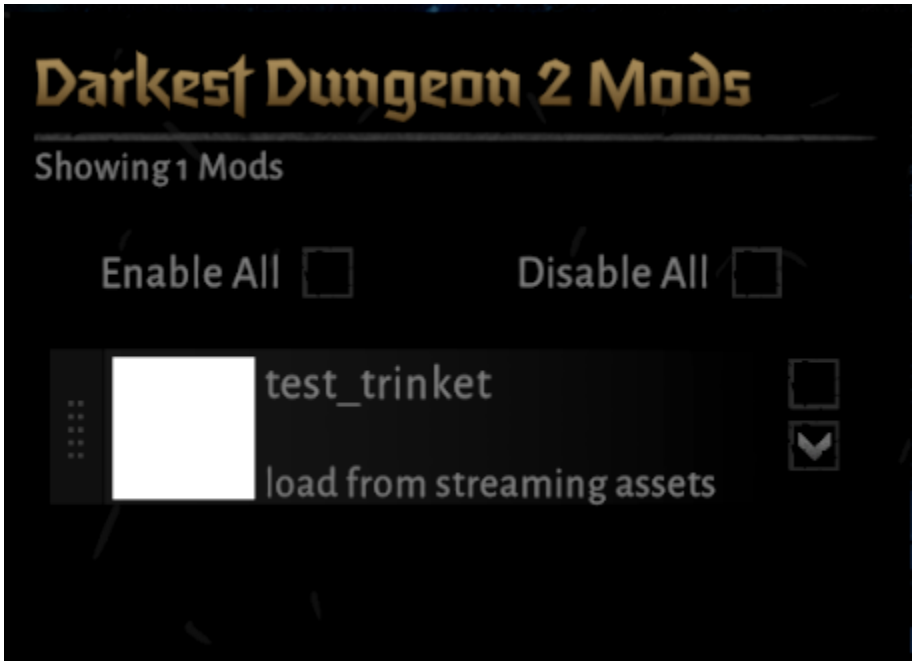




From here, open up Darkest Dungeon II's streamingassets folder (Steam>Steamapps>common>Darkest Dungeon II > Darkest Dungeon II Data > Streaming Assets) and copy the data folder of the mod you want to test into the mods/[data] folder. Rename the data folder to your desired mod name. In the example below, I renamed it to test_trinket.



Once that is done, load up the game to see it with the description “load from streaming assets”



TIPS AND TROUBLESHOOTING!

This section is WIP and will be added upon once testing is underway

Visual Studio Code to view base game data

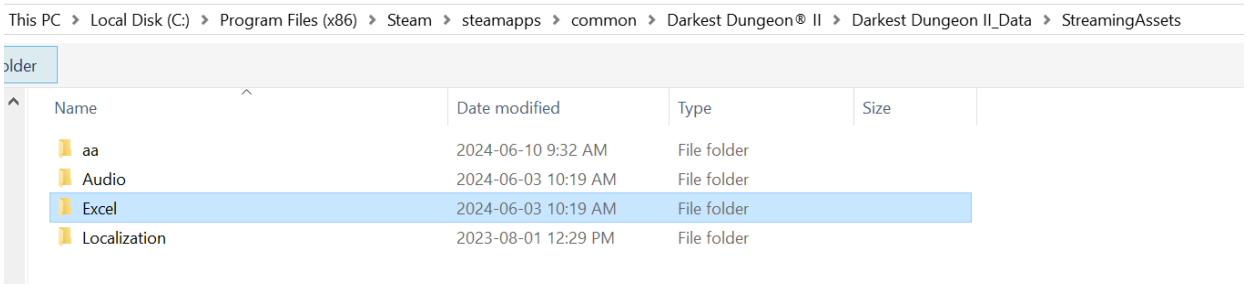
After downloading Visual Studio Code, you can view the base game data by selecting File > Open Folder…

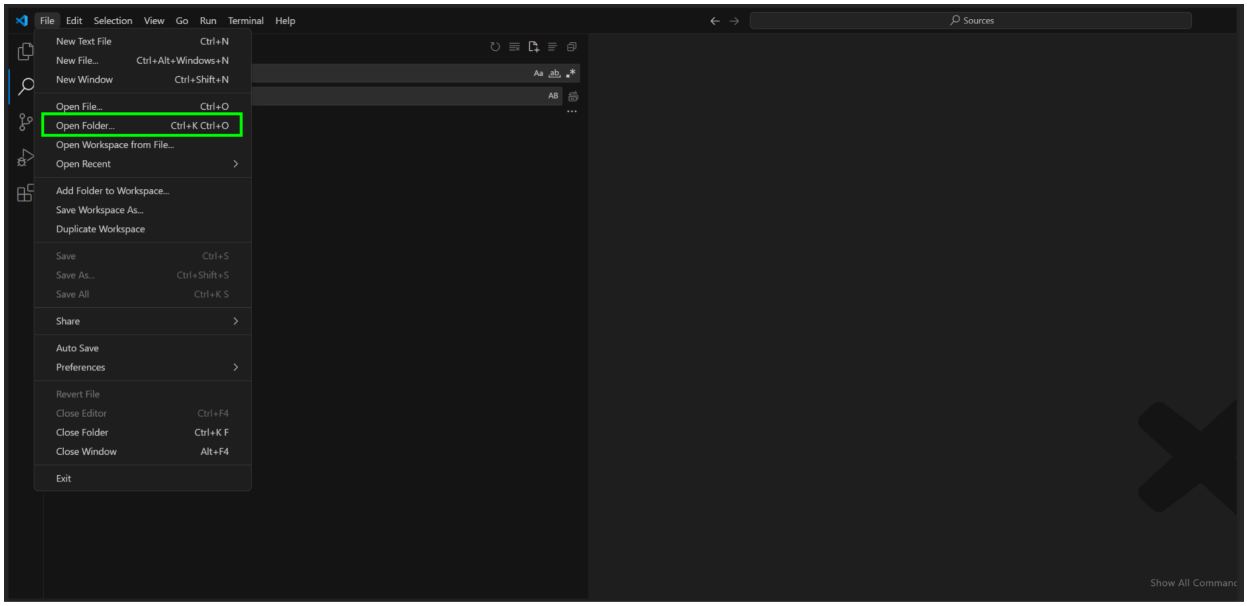
For game data files

Open the Excel folder which can likely be found here
C:\Program Files (x86)\Steam\steamapps\common\Darkest Dungeon® II\Darkest Dungeon II_Data\StreamingAssets

For string/localization files

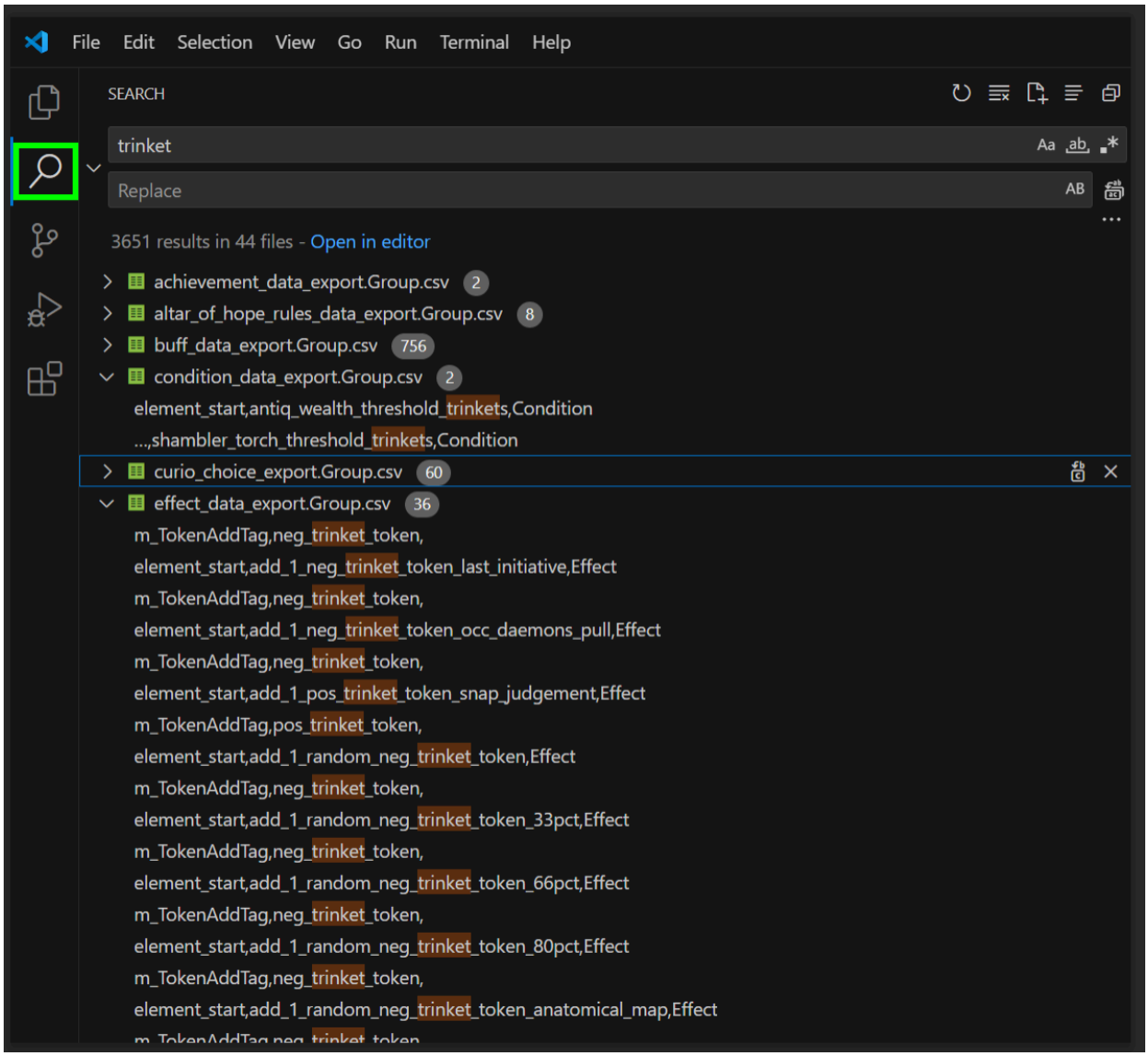
Open the Excel folder which can likely be found here
C:\Program Files (x86)\Steam\steamapps\common\Darkest Dungeon® II\Darkest Dungeon II_Data\StreamingAssets\Localization\Sources





Now that the folder for the base game data is open, you can search and reference various data we use for different items, skills, effects, buffs, loot tables, etc.

ex) Searching **trinket** will display all data that contain the string **‘trinket’**



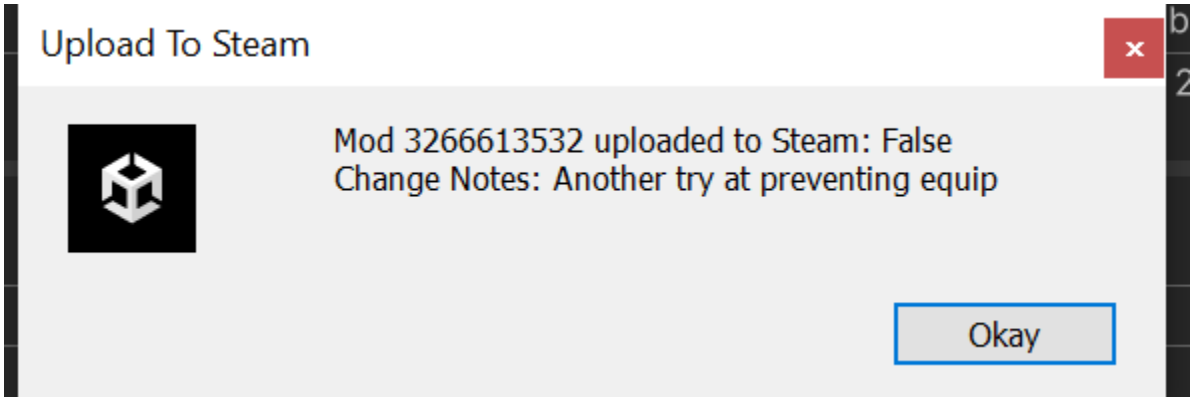
Unity

Drag the slider handle in the bottom right corner to the left so the files are list view



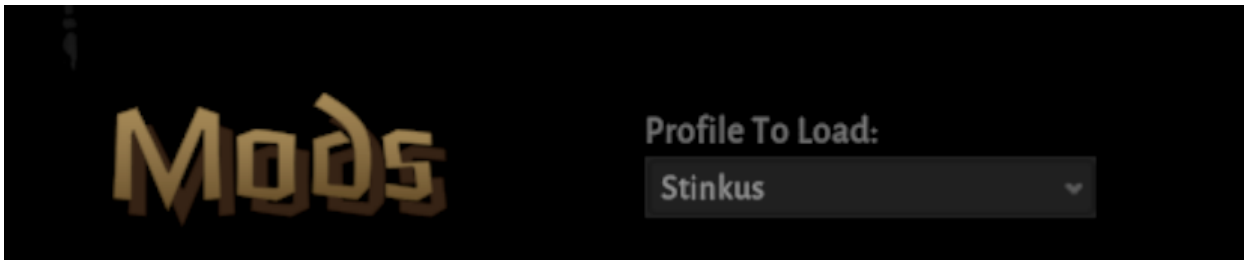


Publishing to Steam



Make sure you close the CSV file before attempting to upload the mod to the Workshop. If the file is still open, the export will fail.

Transferring save files



Mods can only be used on save files that reside in the following folder:

\Users\[XXXXX]\AppData\LocalLow\RedHook\Darkest Dungeon II\mods\SaveFiles\[XXXXXX]\profiles

XXXXX will be replaced by values created by you or steam for your account. If you wish to use a pre-existing save file for mods, you can manually copy the save over to the mods/SaveFiles folder. Regular save files reside in the following folder:

\Users\[XXXXX]\AppData\LocalLow\RedHook\Darkest Dungeon II\SaveFiles[XXXXXX]\profiles

Adding items to base game loot tables

VERY IMPORTANT INFO You'll see many instances of **[LOOTABLENAME]_all**, the **_all** is used to denote the TOP level loot table that has the type **all_sub_table** which means it will be provide ALL of the contents of the loot table contained within rather than a random selection. **It's best to avoid appending to any loot table ending in _all**

Adding items to base game loot tables is done through appending newly created loot tables to base game loot tables **AS LONG AS THE LOOT TABLE IDs MATCH**. When duplicate loot table IDs are used for new loot tables, the content is appended to the base game loot table ID as if they were combined all along.

Ex) **inn_valley** will append this loot table to the provisioner shop selection at the valley inn. You'll see this in the default data file when a new item mod is created).

ADDITIONAL LOOT TABLE

element_start	inn_valley	LootTable
m_chances	1	
m_qtys	1	
m_ids	[ITEMID]	
m_types	item	
element_end		

BASE GAME LOOT TABLE

element_start	inn_valley	LootTable							
m_chances	1	1	1	1	1	1	1	1	
m_qtys	10	1	1	1	1	1	1	2	
m_ids	inn_valley_pets	BLEED_COMBAT_ITEMS	BLIGHT_COMBAT_ITEMS	BURN_COMBAT_ITEMS	BUFF_COMBAT_ITEMS	CLEANSE_COMBAT_ITEMS	DEBUFF_COMBAT_ITEMS	laudanum	torc
m_types	sub_table	sub_table	sub_table	sub_table	sub_table	sub_table	sub_table	item	item
m_tags									
m_conditions									
element_end									

VERY IMPORTANT INFO Any base game loot table that is **ALL CAPS** denote that the loot table will contain **NO sub tables**. These loot tables are like the base “ingredients” to be shared and utilized by other loot tables.

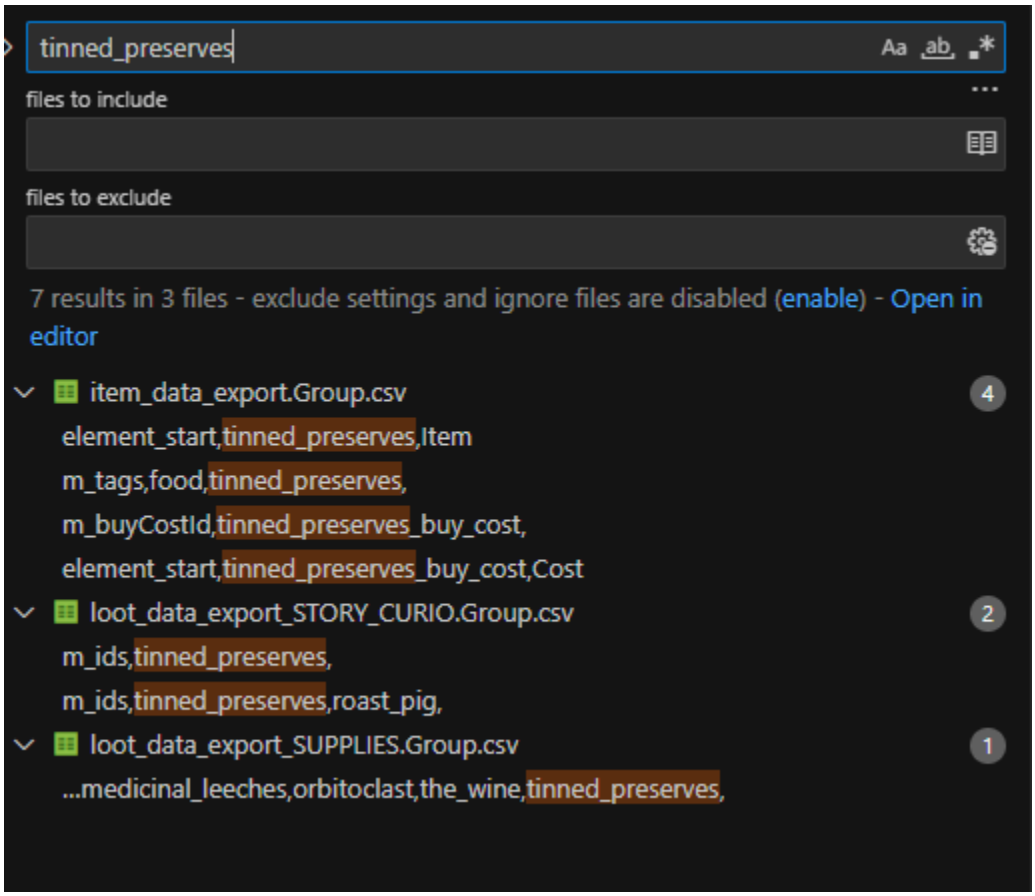
Common item base game loot tables

- **Trinkets:** TRINKETS_COMMODITY_COMMON, TRINKETS_COMMODITY_RARE, TRINKETS_COMMODITY_EPIC, TRINKETS_GENERAL_ALL, TRINKETS_HERO_ALL, TRINKETS_SPECIAL_ANTIQ, TRINKETS_HOARDER, TRINKETS_SHAMBLER_BOSS, TRINKETS_DEATH_BOSS, TRINKETS_COLLECTOR_BOSS, TRINKETS_CHIRURGEON_BOSS, TRINKETS_CULTIST, TRINKETS_CULTIST_KEYS
    - Found in **loot_data_export_TRINKETS**
- **Rest (Inn items):** ALL_REST_ITEMS, POULTICE_INN_ITEMS, TYPICAL_INN_ITEMS, ODD_INN_ITEMS, LITERATURE_INN_ITEMS, SPECIAL_INN_ITEMS, REST_ITEMS_BOOKS, SIGNATURE_INN_ITEMS
    - **ALL_REST_ITEMS** contains only basic inn items NOT signature inn items so this is safe to use for basic inn item pulls
    - Found in **loot_data_export_SUPPLIES**
- **Combat:** ALL_COMBAT_ITEMS, BLIGHT_COMBAT_ITEMS, BLEED_COMBAT_ITEMS, BURN_COMBAT_ITEMS, BUFF_COMBAT_ITEMS, CLEANSE_COMBAT_ITEMS, DEBUFF_COMBAT_ITEMS, SPECIAL_COMBAT_ITEMS
    - **ALL_COMBAT_ITEMS** contains only basic combat items NOT special combat items like Spring Water or Otherworldly Fragment so this is safe to use for basic combat item pulls
    - Found in **loot_data_export_SUPPLIES**
- **Stagecoach**
    - **General:** SC_UPGRADES_ALL, SC_UPGRADES_STORAGE, SC_UPGRADES_GUIDEBOOKS, SC_UPGRADES_FOOD_GEAR, SC_UPGRADES_LUXURY_GEAR, SC_UPGRADES_MEDICINE_GEAR, SC_UPGRADES_ROAD_GEAR, SC_UPGRADES_SCOUTING_GEAR, SC_UPGRADES_TINKERS_GEAR
    - **Flames:** SC_INFERNAL_FLAME, SC_RADIANT_FLAME
    - **Trophies:** SC_TROPHY

- **Pets:** SC_PET

Besides common loot tables, there are many unique tables in DD2 to create specific moments. We recommend searching the base game files with a specific item ID to view all the sources this item can be obtained from.

Ex) Tinned Delicacies (tinned_preserves)



Various Battles: Look in **loot_data_export_COMBATS**, here you'll find IDs for shared loot tables for various battles
Commonly used base game battle loot tables

- **Road battles:** road_gaunt_rewards, road_pillager_rewards, road_faction_rewards, road_caves_rewards
- **Resistance battles:** resistance_faction_rewards, resistance_caves_rewards
- **Guardian battles:** guardian_cultist_1_rewards, guardian_cultist_2_rewards, guardian_cultist_3_rewards
- **Special battles:** antiquarian_rewards, chirurgeon_rewards, collector_rewards, death_rewards, shambler_rewards
- **Lair battles:** lair_prewave_1_rewards, lair_prewave_2_rewards, lair_boss_rewards

Enemy Actors: This is a bit more complicated as individual actors require a specific parameter to drop loot upon defeat. ex) In **cave_swine_skiver_data_export** you can find **m_DeathLootIds,skiver_rewards_all**,.

Corrupted/Missing data

If you notice that DD2 seems to be running incorrectly, such as hero skills missing information, tokens going missing, hero goals breaking, ect., this likely means that **a base game CSV file has been modified or access to it has been blocked**. This will also cause the main menu to display “Mod Detected” in the upper right corner. This can happen if you have a modified base game CSV file open while the game is launching.

To prevent this from happening, be sure to:

- Close all base game CSV files before launching DD2.
- Avoid deleting any data from the base game CSV files.
- If you resized the cells in the CSV excel file, be sure to close it before launching DD2 and discard any changes made. Excel treats resizing cells as modifying the file and thus prevents the game from accessing it while it’s still open.

This can be fixed either by verifying the integrity of the game files or by reinstalling the game entirely. **Be sure to close all CSV files before reinstalling, as this can prevent the installation from completing.**

TIFTID SEZ: If you wish to modify your game data while the program is running, consider copying your CSV files into an external folder and modifying them therein - then, copy these files back to the original folder before running the game. This will also protect your modded data files from being permanently overridden by game updates.

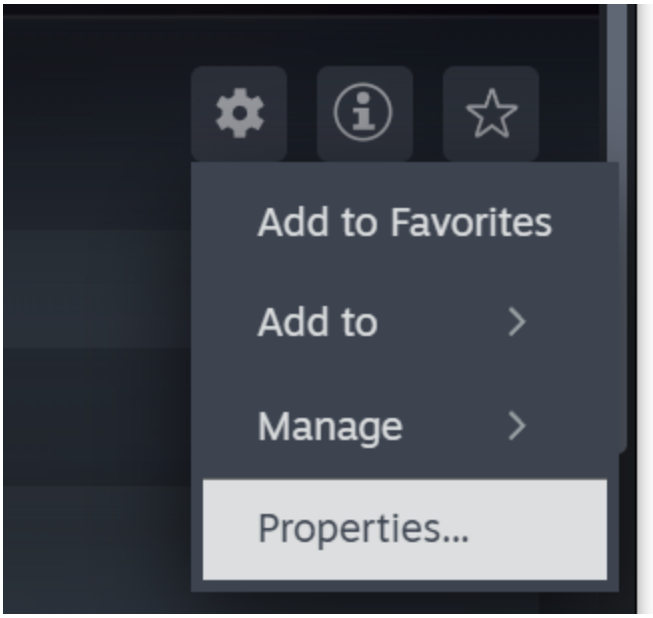
Mod not appearing in-game

If you uploaded a mod to the workshop and subscribed to it, but it’s not appearing in the mod menu in-game, this likely means you’re using an outdated version of the Darkside project.
Make sure you’re using the Darkside project that is provided through Steam (Darkest Dungeon 2 Mod Tools) and NOT the Mod project provided in the google drive download.
Also make sure that the Darkest Dungeon 2 Mod Tools project is updated to the latest version.

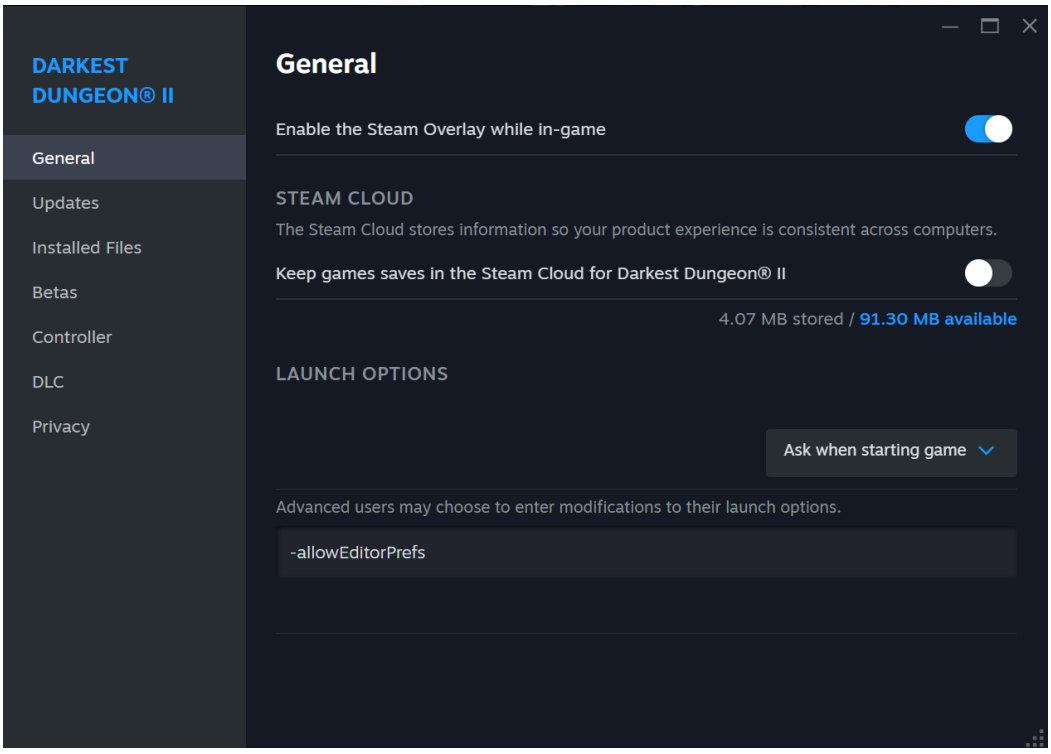
Testing mods with Editor Prefs and Cheat options

Using Editor Prefs to enable the Cheat options can help with mod testing.
To enable Editor Prefs

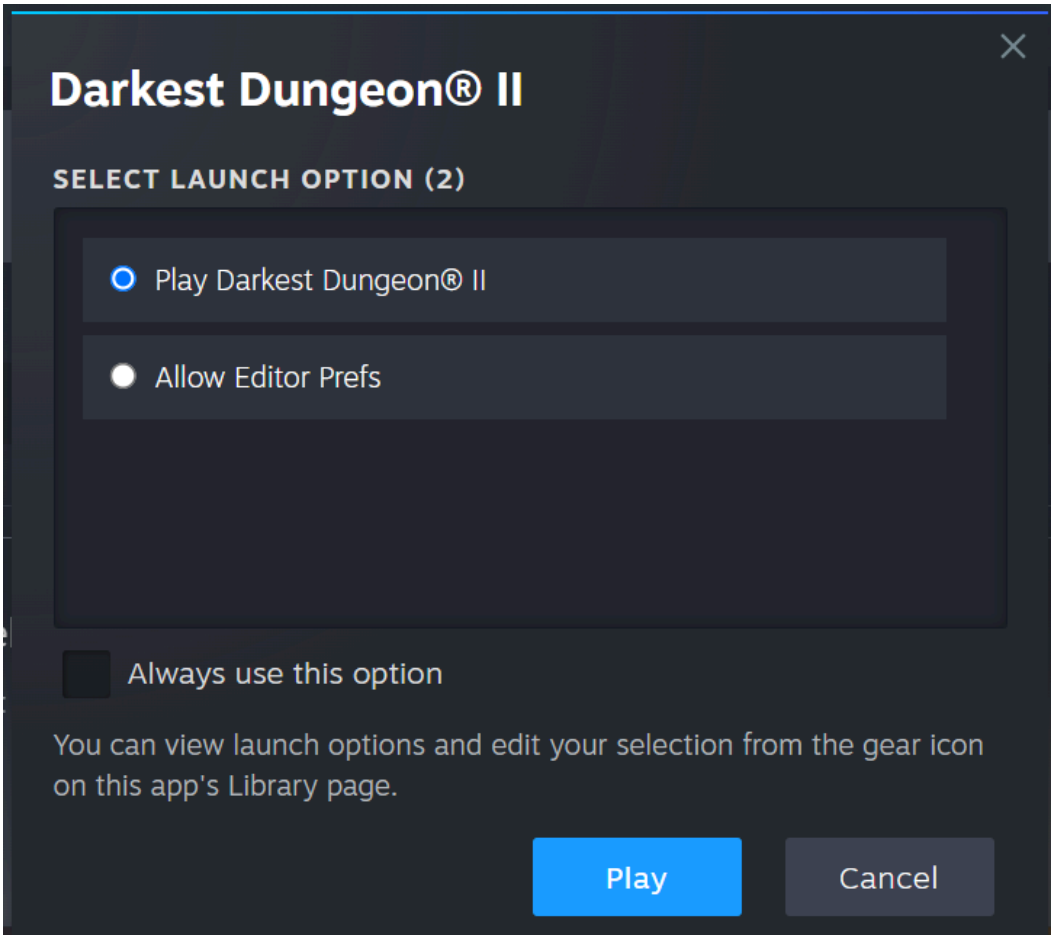
1. On the DD2 Steam page, open Properties from the settings (Gear icon)

a. 

2. Under General > Launch Options, add the line '-allowEditorPrefs' to the Advanced users launch option field

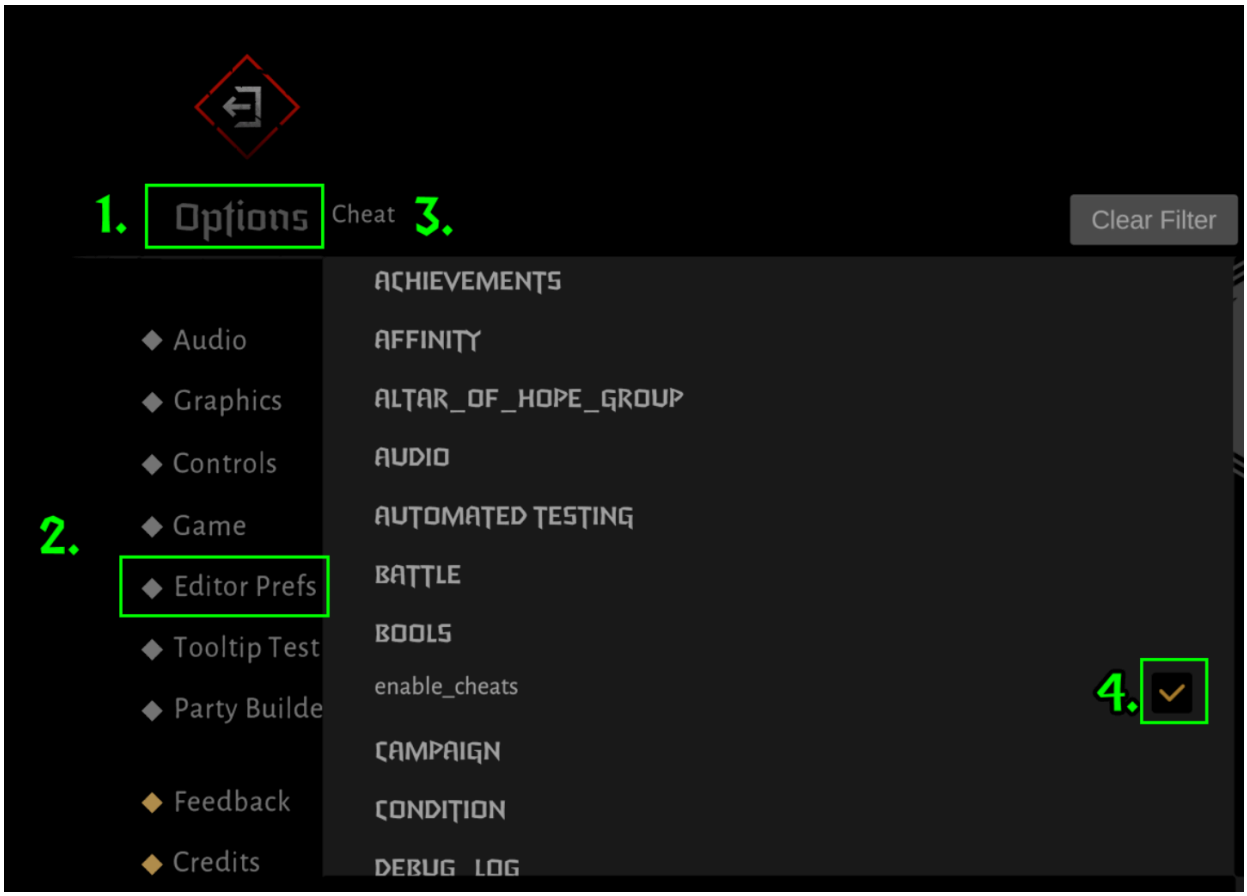
a. 

3. Next, when launching the game from Steam and presented with Launch Options, select Allow Editor Prefs

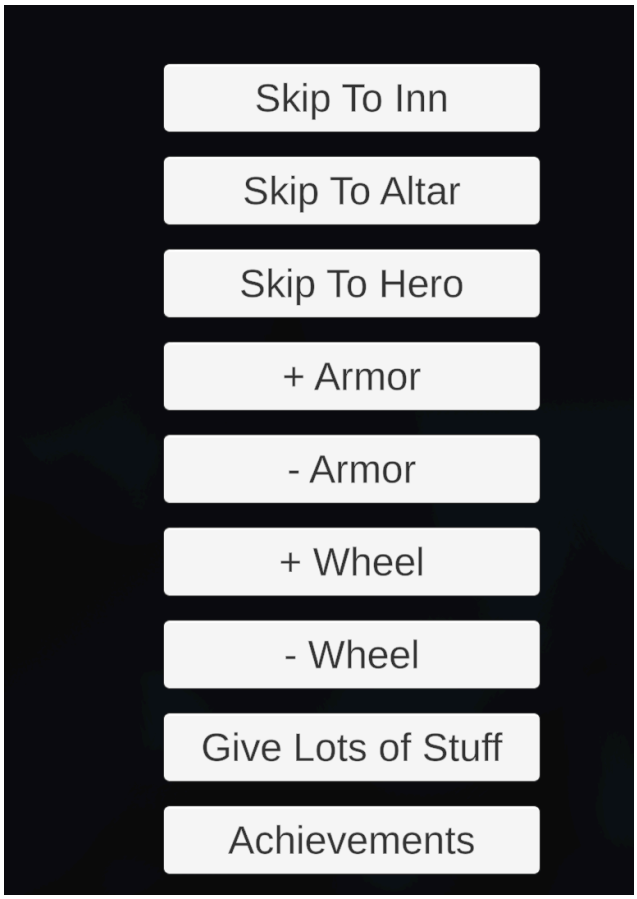
a. 

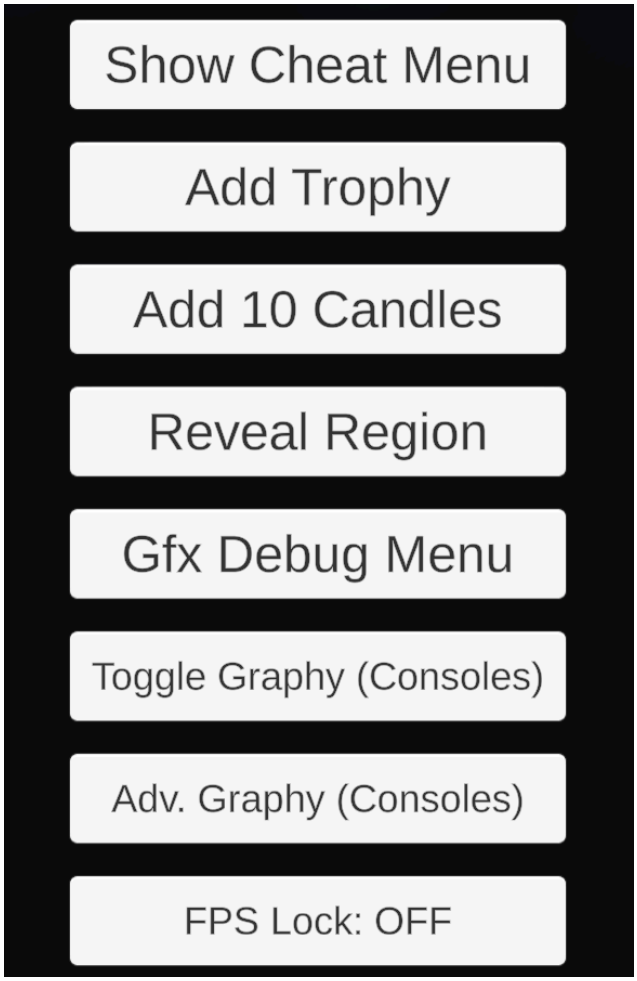
To enable Cheats

1. Open the Options Menu
2. Select Editor Prefs
3. Type ‘Cheat’ into the ‘Enter text…’ field
4. Enable Cheats

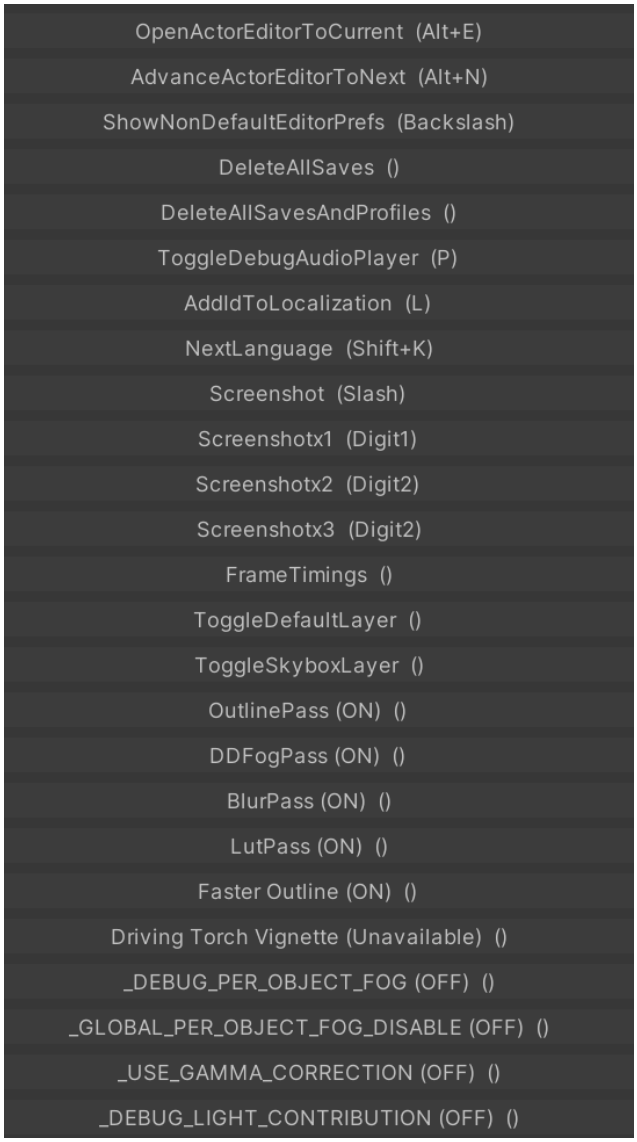


Once cheats are enabled, there will be options available on the Options Menu and a selection of Hotkeys while Cheats are enabled.


Skip To Inn
Skip To Altar
Skip To Hero
+ Armor
- Armor
+ Wheel
- Wheel
Give Lots of Stuff
Achievements


Show Cheat Menu
Add Trophy
Add 10 Candles
Reveal Region
Gfx Debug Menu
Toggle Graphy (Consoles)
Adv. Graphy (Consoles)
FPS Lock: OFF





Adding Editor Prefs to Mods

If you want to add editor prefs to a mod you can do that by adding a text file to the data folder in your mod. It will be loaded additively to the editor prefs that are already loaded or that are loaded by other mods.

Ex:
A Cheats.txt in the data folder with
enable_cheats-True
Will enable cheats anytime that mod is loaded.

End of guide, below are feature suggestions and other info for Red Hook

---

MOD TOOL WISHLIST

Put any mod tools you wish to have access to here

1. Barks triggered via effects displaying as positive/negative
2. Custom tokens
3. Sound effects for items (equipping, moving in inventory, using combat items)

4. Custom abilities
5. Assigning existing animations to custom skills
    a. TIFTID SEZ: This idea is based
    b. Related idea: Custom SkillReplacements for hero paths
6. Custom paths
7. Custom weapon kits
8. Recolored weapon kits (if easy to implement before above item)
9. Custom models
10. Custom animations
11. A “Deliverable” preset for custom items
12. Route generation settings and related models (custom biomes / modifying base-game biomes)
13. Custom map nodes
14. Ability to modify base game objects, gameplay settings, for example starting with <4 heroes, overwriting/modifying base game objects like actor models, scenery, etc
15. Access to base game files, ability to write custom code, and ultimately be able to work at the developer level.
16. Custom Stagecoach skins
17. Ability to modify confession run length, both in leagues to travel and in inns to reach.
    a. TIFTID SEZ: You can already modify the number of regions in a run by modifying boss_data_export.Group.csv
    b. With this in mind, allowing modification of leagues to travel is the only thing that is desired
~~18. Offline/local mod usage~~
19. Conditional loading of items and overrides
    a. Example use case: a mod that wishes to add an override only when a certain dlc or other mod is present
20. Code injection to override existing core code functionality or the ability to load in a library that is capable of doing that (eg. harmony, bepinex, etc)
21. Prevent the mod creation tools from auto-opening when you open darkside
~~22. The ability to modify EditorPrefs settings via mods so they don’t need to be enabled every time you launch the game~~
    ~~a. Ex: Setting Fast Driving to always be Enabled and have the multiplier always set to a specified number~~
23. Ability to modify art/visual fx/animations of skills changed by paths
24. When you unsubscribe from a mod, it leaves behind a (small) “catalog.jsondd” file which means the mod folder isn’t deleted fully
25. Mod options.
26. A condition type that can determine if a certain mod is installed (as well as its opposite)
27. When you open the Mods menu in the main menu, the scroll bar isn’t scrolled to the top
28. A version of (or update for) death armor, so it behaves more logically with heroes.
    a. As it stands, death armor is immediately lost when a hero enters death’s door, effectively wasting a token.
29. Expose lockable quirk amounts to a .csv file

30. Skills having a separate animation on crits
31. ~~Steamworks description field opt-out (ability to not overwrite existing description on mod page when publishing)~~ Should be done
32. Allow torches with custom torch levels to be ambushed (correctly). Currently, torches that have custom torch levels and are not marked with the “no ambush” tag will be ambushed, but will not load the ambush battle, leading to a softlock.
33. A way to manipulate the amount of trinket slots as well as the amount of skill slots via effects/buffs/rundatastats. In other words, enable the ability to increase or decrease them mid-run.
34. In Cheat Menu > CombatDebug > Update Actors, the drop down lists become very cumbersome with large lists. The wished item here is a search filter.
35. Ability to override only specific parameters instead of having to override the entire element
36. Add a new Mods section to the tutorials page and allow us to create custom tutorials for our mods
37. The ability to use custom sprites in localization files

Tangentially related requests

- “skill_damage_dealt” m_ConditionType (no m_ConditionActor needed, tested only when performer uses a skill)
    - “skill_damage_received” variant which is tested when performer RECEIVES damage from a skill?
    - “damage_received” variant which also applies to non-skill damage?
- on_crit_heal_as_performer_to_performer_effects, on_crit_heal_as_performer_to_target_effects, etc…
- On *x* token lost hook
- If has x token conditiontype
- Conditions for AffinityLeaningLevel (so we can change how they behave with items)
- “DD1 character” unity class - essentially the same thing as the Actor class, but instead of specifying models/animations, we specify a sprite for the actor during its combat idle stance, when it dodges, when it takes damage, when it’s on Death’s Door and when it’s doing each of its skills - the goal is to make porting DD1 actors across to DD2 as easy as possible, and to make it so that people don’t have to make a “dummy model” for sprite-based characters
- m_BuffStealTags effect parameter

Wishlist items fulfilled

1. Empty mod creation tool (7/30/24)
2. Improved Steamworks description field (7/30/24)
3. Local mod building and testing (7/31/24)

Reference for how difficult any given task is (as far as we know) [OUT OF DATE AS OF STEADFAST STEWARD]

Bespoke tools for modders (not necessarily feature complete, see wishlist)

- Custom items
- Custom hero palettes
- Local mod building and testing

Other functional tools present in darkside

- Custom tokens
- Custom paths (pretty hacky, has some visual bugs)

Doable with workarounds using darkside + third party tools

- Custom skills
- Custom animations (except blend shapes)
- Custom heroes
- Custom hero models
- Custom/edited models/animations for base game heroes (probably?)

Not currently possible

- **Not exposed in data:**
    - Leagues to travel per region
    - Locked quirks cap
- Custom non-skill animations for existing heroes (idle, antic, inn, etc.)
- Custom monsters (maybe?)
- Custom models for monsters, nodes, stagecoach, etc
- Custom textures for monsters, nodes, stagecoach, etc
- Custom VFX
- Custom SFX
- Custom weapon kits
- Custom biomes
- Custom nodes
- Conditional mod loading
- Scripting, library loading, code injection
- Custom cutscenes (like act bosses being defeated)
- Custom UI elements and UI changes

Implementation Ideas

Options menus for mods

I noticed that the guy making the enemy barks mod wanted to add options that enable/disable barks for each enemy faction.
This makes me think that in reality, a lot of mod options could be boiled down to “do/don’t load this csv”.
So, for each csv and txt file in a mod, a checkbox could be created under that mod in the menu, and a special localization file provides names to each of those checkboxes.
(for instance - “effects_bark_faction_lost_battalion.csv” becomes “Enable Lost Battalion Barks”)
Perhaps there could also be a special csv or json file that controls whether or not each file is loaded by default when you download the mod.

A possible better way to do it is just to have it all be managed by one json file in which each option name is the header of a list of csv or txt filenames - that way a single option can control the loading of multiple files, so you can have one checkbox seamlessly turn off both effects and their associated localisation.

scScripting

A lot of people have requested scripting, but I personally think scripting won’t ever really happen, since it would require either providing the game’s source code (which will never happen for obvious reasons) or developing a new scripting language a la QuakeC (which is clearly too much work).

So I’m here to suggest an alternate solution.

VTMB

Vampire: The Masquerade - Bloodlines (2004) has a scripting system which is generally well-praised.
The long and short of it is that it uses Python 2.1.2, released 16/Jan/2002.
But it doesn’t just provide the raw interpreter binary with the game - instead, the developers downloaded Python’s C source code, changed it a little and pre-compiled it into a DLL that is shipped along with the main game’s binaries.

This is a critically important facet of using Python as a scripting language, because it allows developers to remove standard libraries that would potentially allow malicious code execution, such as the os module (which allows file read/writes) and the ctypes module (which allows Python to utilise entrypoints into C DLLs, including various win32 libraries).

It also (presumably) allows for very fast interoperability between the interpreter and functions in the game's C++ DLLs.

The fact that Python is running alongside the main game and is an interpreted language also means that Python code can be run directly from the game's developer console, either line-by-line or as executed from a .cfg file.

Another important aspect of VTMB's Python scripting is that it exposes very little of the game's source code.
Most of the ways to actually interface with the game involve using the "__main__" object - a static object which is always in scope.
The player can be accessed by calling "__main__.FindPlayer()", and most of the game's global variables (the most important one being an int called StoryState) can be accessed through "__main__.G".
Here's some example code:

```
import __main__
from __main__ import G

Find = __main__.FindEntityByName

#F.U. SYNDICATE: Spawns bertram's key after mandarin dies.
def spawnBarabusKey():
    mandarin = Find("Mandarin")
    center = mandarin.GetCenter()
    point = (center[0], center[1], center[2] + 20)
    key = __main__.CreateEntityNoSpawn("item_k_fu_cell_key", point,
(0,0,0) )
    key.SetName("fu_key")
    sparklies = __main__.CreateEntityNoSpawn("inspection_node", point,
(0,0,0) )
    sparklies.SetParent("fu_key")
    __main__.CallEntitySpawn(key)
    __main__.CallEntitySpawn(sparklies)
```

Advantages of Python custom DLL as a scripting language

- High-level and garbage-collected, so reasonably hard for newbies to create memory leaks and for experienced programmers to do deliberate evil memory mismanagement
- Easy-to-understand for novice programmers since it doesn't use C syntax
- Dynamic typing makes it difficult to encounter common errors such as passing a float to a function that expects an int
- A Python exception in the interpreter doesn't halt execution of the rest of the game
- Being an interpreted language makes it natively open-source, so users know exactly what they're downloading unlike recompilations where you could be downloading anything

Disadvantages of Python custom DLL

- Dynamic typing means that strict typechecking (assert isinstance(variable, type)) would need to be enforced for any parameters that are being passed to the C# runtime of the game
- Interpreted languages are slow and allowing .pyc or .pyx files to be executed is potentially dangerous unless we make the interpreter itself compile them at runtime
    - JIT compiling (available through numba library and natively supported in experimental branches of Python) makes performance better, but holds up code execution at runtime while it's compiling the script and adds potential for malicious code execution

How should DD2 do it?

Effect definitions in CSV files should have a new "m_RunScriptFile" parameter, with the name of the file afterwards. (e.g. "m_RunScriptFile,make_invincible.py,")
It will search inside "steamapps\common\Darkest Dungeon II\DarkestDungeon II Data\StreamingAssets\Python" - or, in the case of mods, "steamapps\workshop\content\1940340\MOD_STEAMID\Python".
The game's Python interpreter will then run this script file (__name__ == "__main__") when the effect is triggered.

By default, effects of this nature will be formatted like <color=#{notable}>Run Script File:</color> <color=#{mastery}>{0}</color> where {0} is replaced by the filename of the script.
Since this is obviously not ideal, it's recommended that mod developers override the formatting of any effects that trigger script files.

For a __main__ object, there should be an object called "darkest" (I'm not good enough with Python to know if calling it __main__ in VTMB was some kind of technical requirement).
The performer and target of the current effect are members of __main__ (converted from DD2's C# Actor class to a Python **IronCrownActor** class), and if the effect is actorless then both are **None**.

Some other members of darkest:

- **heroes** - a list of type **IronCrownActor** which contains the four heroes (fewer if there are fewer)
    - **IronCrownActor** class members:
        - name - string
        - localized_name - string
        - rank - int
        - health - 0-1 float
        - health_max - int
        - spd - int
        - turns_per_round - int

- stress - int
- stress_max - int
- size - int - controls the actor's size - use with caution!
- hero_path_id - int - index of hero path this actor has equipped - 0 is Wanderer, -1 is Reserve, this value is **None** if the actor is not a hero
    - This can be set to change a hero's path mid-run, and even update the SkillReplacements
- skills - list[**IronCrownSkill**] - list of combat skills
- skills_equipped - list[**IronCrownSkill**] - list of equipped combat skills
- skill_modifiers - list[**IronCrownSkillModifier**] - list of skill modifiers due to relationships
- buffs - list[**IronCrownBuff**] - list of buffs this actor has
- tokens - list[**IronCrownToken**] - list of tokens this actor has
- relationships - list[int] - list of indices into **darkest**.relationships for the relationships this hero is part of
    - Modifying this list can change which relationships this actor has in the C# runtime - so, changing it to an empty list will clear all relationships
    - If this actor is not a hero, this list is empty
- resistances - dict[string, float]
- immunities - list[string] - corresponds with m_ResistAlwaysIds
- tags - list[string]
- is_healthless - bool - controls whether the actor is invulnerable or not
- is_act_out_valid - bool - controls whether the actor can perform relationship act outs or not
- is_token_view_valid - bool - controls whether the Academic View can be used on the actor
- **use_skill** - method with arguments (skill: **IronCrownSkill**, target: **IronCrownActor**) that forces the actor to use a skill on a target (even if the skill isn't in actor.skills or the skill isn't valid according to the skill's launch ranks, target ranks and conditions)
- **move** - method with arguments (amount: int, can_be_resisted: bool) that moves the actor in its party by the specified amount
- **move_to** - convenience method with arguments (rank: int) that calls **move**(rank - self.rank, False) to move the actor to the specified rank
- **add_buff** - method with arguments (buff: **IronCrownBuff**) that queues a specified buff to be added to the target by the performer at the end of script execution
- **remove_buff** - method with arguments (buffid: str) that removes all instances of the given buff on the actor
- **remove_buff_by_tag** - method with arguments (tag: str) that removes all buffs on the actor which have the tag
- **change_health** - method with arguments (amount: int) that queues an event that deals damage to or heals the hero, with the performer as the

performer and the target and the target. Since it's queued, it can be given the appropriate SFX and VFX in the C# runtime.
        - **change_stress** - similar to change_health, but with stress - also has a can_be_resisted argument since Stress RES exists
- **get_hero_in_rank** - a method that returns the hero in the given rank, or None if there is no hero in the specified rank
    - All functions that expect a rank as input expect the rank to be in the range 1-4, not 0-3
- **hero_definitions** - a dict of type [string, **IronCrownActor**] which contains all the heroes as defined in the game's CSV files - indexed by ActorDataClass id
- **monsters** - a list of type **IronCrownActor** which contains the monsters of the current combat - this list is None if the heroes are not in combat
- **get_monster_in_rank** - the same thing as **get_hero_in_rank**, but for monsters - particularly useful for monsters because the list of heroes is passed into __main__ from C# already ordered by rank, but since monsters can have an m_Size other than 1, it's not always obvious how position in a rank-ordered list can correspond to index in that list
- **monster_definitions** - a dict of type [string, **IronCrownActor**] which contains all the monsters as defined in the game's CSV files - indexed by ActorDataClass id
- **variables** - a standard Python dynamically typed dictionary - since this remains in scope as long as darkest does (so, the entire lifetime of the interpreter), it can be used to create variables that persist between multiple executions of the same script. However, it also won't be garbage-collected until the interpreter is killed, so repeatedly defining variables might take up a large amount of memory. To prevent this, counting the memory used by the defined variables and throwing a **ValueError** when it exceeds a certain value might be a good solution.
    - The game also stores some variables from the C# runtime of the game in this dict when the script is run (or maybe when the interpreter is started) - these can be changed by the script and have their new values set in the C# runtime at the end of script execution, but they're strictly typechecked before this happens
    - Examples of variables for the game to store - "combat_round_number", "run_value_doom", "run_value_sc_armor", "run_value_sc_wheels"
- **relationships** - a list of type **IronCrownAffinityRelationship** that contains the party's six relationships - strictly checked so that this list still contains six valid non-duplicate relationships before being passed back to the C# runtime
    - **IronCrownAffinityRelationship** class members - **tuple**[**IronCrownActor**] actors, int affinity
- **buffs** - a dict containing type **IronCrownBuff** that holds all the buffs the game has loaded from its CSV files - this is created once on interpreter creation and is thereafter never passed back to the C# runtime, so it can be modified freely
    - IronCrownBuff class members:
        - id - string
        - duration_type - string
        - duration_amount - int or **None** if duration_type is "infinite"
        - condition - **IronCrownCondition**, defaults to **None**

        - desc_override - string, defaults to **None**
        - tooltip_override - string, defaults to **None**
        - is_visible - bool - defaults to **True**
        - actor_data_stats - **IronCrownActorDataStats**, defaults to **None**
        - actor_data_effects - **IronCrownActorDataEffects**, defaults to **None**
        - run_data_stats - **IronCrownRunDataStats**, defaults to **None**
        - tags - list of strings - defaults to empty list
- **conditions** - similar to **darkest**.buffs, but with conditions
- **effects** - similar to **darkest**.buffs, but with effects
- **test_conditions** - method with arguments (conditions: list[**IronCrownCondition**], condition_type: string = “all”) that tests the specified conditions (using the performer, target, heroes, monsters and variables) and returns **True** or **False** based on the result
- **trigger_effect** - method with arguments (effect: **IronCrownEffect**) that first calls **darkest**.**test_conditions** with the effect’s conditions and condition type as input, and then if that returns True, queues the effect to be triggered in the C# runtime at the end of script execution
- **test_mod_installed** - method with arguments (steamid: int) that returns **True** if a mod with the specified SteamID is installed - useful for script files that want to test for the presence of popular community mods (for compatibility)

What would you use this for?

As a way to gauge interest and understand what parts of the game would need to be exposed to the “darkest” object, this section can act as a place to collect psuedocode scripts that act as potential use-cases for the system.
I’ll start:

Tiftid - Fragile Flame TMTRAINER script

```
# Script by Tiftid 04/Aug/2024
# This script is triggered on each hero via a modded version of The
Fragile Flame, which has in its ActorDataEffects block
"enter_biome_effects,run_inf1a_script,"
# run_inf1a_script also has a condition to ensure that it only runs if
biome_typical is equal to 1, so we only need to run this script once
# Its purpose is to generate a bunch of random buffs from the effects and
conditions in the base-game, and then create a buff which randomly applies
a few of these on turn start and give each hero that buff
# The goal being to create a similar kind of item to TMTRAINER from The
Binding of Isaac: Repentance

import __main__ as darkest
import random
import math
```

```
valid_actor_data_effects_types = [
    "performer_effects",
    "target_effects",
    "on_hit_as_performer_to_performer_effects",
    "on_hit_as_performer_to_target_effects",
    "on_hit_as_target_to_performer_effects",
    "on_hit_as_target_to_target_effects",
    "on_miss_as_performer_to_performer_effects",
    "on_miss_as_performer_to_target_effects",
    "on_miss_as_target_to_performer_effects",
    "on_miss_as_target_to_target_effects",
    "on_attack_as_performer_to_performer_effects",
    "on_attack_as_performer_to_target_effects",
    "on_attack_as_target_to_performer_effects",
    "on_attack_as_target_to_target_effects",
    "on_crit_as_performer_to_performer_effects",
    "on_crit_as_performer_to_target_effects",
    "on_crit_as_target_to_performer_effects",
    "on_crit_as_target_to_target_effects",
    "performer_on_crit_single_effects",
    "on_kill_as_performer_to_performer_effects",
    "performer_on_kill_fail_effects",
    "friendly_team_effects",
    "enemy_team_effects",
    "performer_team_others_effects",
    "target_team_effects",
    "target_team_others_effects",
    "combat_health_damage_effects",
    "combat_health_heal_effects",
    "combat_stress_damage_effects",
    "combat_stress_heal_effects",
    # "friendly_death_effects", - this is known to not apply buffs
    # "enemy_death_effects",
    # "deaths_door_enter_effects",
    # "deaths_door_survive_effects",
    # "deaths_door_exit_effects",
    "turn_start_effects",
    "turn_end_effects",
    "round_start_effects",
```

```
    "round_end_effects",
    "turn_start_friendly_team_effects",
    "turn_end_friendly_team_effects",
    "turn_start_enemy_team_random_effects",
    "turn_start_enemy_team_effects",
    "turn_end_enemy_team_effects",
    "combat_health_damage_friendly_team_effects",
    "combat_health_damage_enemy_team_effects",
    "combat_health_heal_friendly_team_effects",
    "combat_health_heal_enemy_team_effects",
    "combat_stress_damage_friendly_team_effects",
    "combat_stress_damage_enemy_team_effects",
    "combat_stress_heal_friendly_team_effects",
    "combat_stress_heal_enemy_team_effects",
]
valid_actor_data_stats_types = {
    "key_map,health_max": "any_30_0.9", # int value, then float value
    "key_map,crit_chance": "float_0.5",
    "key_map,speed": "int_9",
    "key_map,health_damage_dealt_percent": "float_2.5",
    "key_map,health_damage_received_percent": "float_2.5",
    # "key_map,speed_number_of_turns": "int_1",
    # "key_map,stress_max": "int_2",
    "sub_stat,resistance,stun": "float_1", # desired data type of value,
and desired maximum value for random range
    "sub_stat,resistance,bleed": "float_1",
    "sub_stat,resistance,blight": "float_1",
    "sub_stat,resistance,burn": "float_1",
    "sub_stat,resistance,disease": "float_1",
    "sub_stat,resistance,move": "float_1",
    "sub_stat,resistance,debuff": "float_1",
    "sub_stat,resistance,death": "float_0.5",
    "sub_stat,resistance,stress": "float_1",
    "sub_stat,resistance_ignore,stun": "float_1",
    "sub_stat,resistance_ignore,bleed": "float_1",
    "sub_stat,resistance_ignore,blight": "float_1",
    "sub_stat,resistance_ignore,burn": "float_1",
    # "sub_stat,resistance_ignore,disease": "float_1", # lol
    "sub_stat,resistance_ignore,move": "float_1",
    "sub_stat,resistance_ignore,debuff": "float_1",
```

```
    "sub_stat,dot_effect_value_dealt_change,bleed": "int_10",
    "sub_stat,dot_effect_value_dealt_change,blight": "int_10",
    "sub_stat,dot_effect_value_dealt_change,burn": "int_10",
    "sub_stat,dot_effect_value_dealt_change,hot": "int_6",
    "sub_stat,dot_effect_value_dealt_multiplier,bleed": "intfloat_4_0.25",
# number of possible increments, and float value added with each increment
    "sub_stat,dot_effect_value_dealt_multiplier,blight":
"intfloat_4_0.25", # so, this range is (0.25, 0.5, 0.75, 1)
    "sub_stat,dot_effect_value_dealt_multiplier,burn": "intfloat_4_0.25",
    "sub_stat,dot_effect_value_received_change,bleed": "int_10",
    "sub_stat,dot_effect_value_received_change,blight": "int_10",
    "sub_stat,dot_effect_value_received_change,burn": "int_10",
    "sub_stat,dot_effect_value_received_change,hot": "int_6",
    "sub_stat,dot_extra_duration_dealt,bleed": "int_2",
    "sub_stat,dot_extra_duration_dealt,blight": "int_2",
    "sub_stat,dot_extra_duration_dealt,burn": "int_2",
    "sub_stat,dot_extra_duration_received,bleed": "int_2",
    "sub_stat,dot_extra_duration_received,blight": "int_2",
    "sub_stat,dot_extra_duration_received,burn": "int_2",
    "sub_stat,health_heal_dealt_percent,skill": "float_1",
    "sub_stat,health_heal_received_percent,skill": "float_1",
    "sub_stat,overstress_chance_modifier,resolute": "float_0.25",
    "sub_stat,overstress_chance_modifier,meltdown": "float_0.25",
}
valid_conds: list
valid_effects: list
manager_buff_id = f"inf1a_buff_manager{darkest.target.rank - 1}"
manager_buff_effect_names_ads = []
manager_buff_effect_names_ade = []

def trim_data():
    global valid_conds
    global valid_effects
    valid_conds = list(filter(test_cond(), darkest.conditions))
    valid_effects = list(filter(test_cond(), darkest.conditions))


def test_cond(cond) -> bool:
    id = cond.id.split("_")
    if not cond.is_visible:
```

```
        return False # Don't pull from invisible conds to make
auto-formatting easier
    if id[0] == "performer":
        if id[1] == "skill" and id[2] == "hist":
            return False # Don't pull from hero goal conds cause they have
very specific overrides
        elif id[1] == "item" and id[2] == "use":
            return False # Also a hero goal cond
        elif id[1] == "kill":
            return False # Also a hero goal cond
        elif id[1] == "visit":
            return False # Also a hero goal cond
        elif id[1] == "is":
            return False # Ignore conditions that require the performer to
be a specific class, because most of these ask the performer to be a
monster and these buffs will only be on heroes
    elif id[0] == "herostory":
        return False # Ignore hero story conds
    elif id[0] == "item" and id[1] == "tooltip":
        return False # Ignore signature item conds, cause they have very
specific overrides (the hero's name; they won't display the body of the
buff cause they have no {0})
    return True


def test_effect(effect) -> bool:
    for cond in effect.conditions:
        if not test_cond(cond):
            return False  # Don't use effects with invisible conditions
    return True


def assemble_random_buff(index: int, type: str = "ads"):
    buffid = f"inf1a{index:03}"
    conditional = False if random.random() < 0.5 else True # 50% chance
for the buff to have a random condition
    cond = random.choice(valid_conds) if conditional else None
    if type == "ads":
        ads = darkest.IronCrownActorDataStats()
```

```
        stat_key =
random.choice(list(valid_actor_data_stats_types.keys()))
        stat_class = valid_actor_data_stats_types[stat_key].split("_")
        random_roll = (random.random() - 0.5) * 2 # -1 to 1
        if stat_class[0] == "float":
            stat_value = random_roll * float(stat_class[1])
        elif stat_class[0] == "int":
            stat_value = int(random_roll * int(stat_class[1]))
        elif stat_class[0] == "intfloat":
            stat_value = math.ceil(random_roll * int(stat_class[1])) *
float(stat_class[2]) # Use ceil cause otherwise there would be a chance of
adding +0% which is pretty pointless
        elif stat_class[0] == "any":
            coinflip = random.randint(0, 1)
            if coinflip == 0:
                stat_value = int(random_roll * int(stat_class[1]))
            else:
                stat_value = random_roll * float(stat_class[1])
        is_key_map = True if stat_key.split(",")[0] == "key_map" else
False
        if is_key_map:

ads.key_map.append(darkest.IronCrownKeyMap(stat_key.split(",")[1],
stat_value))
        else:

ads.sub_stats.append(darkest.IronCrownSubStat(stat_key.split(",")[1],
stat_key.split(",")[2], stat_value))
        buff = darkest.IronCrownBuff(id=buffid, duration_type="round_end",
duration_amount=2, condition=cond, tags=["buff"], actor_data_stats = ads)
    elif type == "ade":
        ade = darkest.IronCrownActorDataEffects()
        ade_type_key =
random.choice(list(valid_actor_data_effects_types.keys())) # TODO:
Implement chance for buff to have apply_limit_effects
        ade[ade_type_key].append(random.choice(valid_effects)) # Gives it
a random effect under a random ADE type
        buff = darkest.IronCrownBuff(id=buffid, duration_type="round_end",
duration_amount=2, condition=cond, tags=["buff"], actor_data_effects =
ade)
```

```
    return buff


def create_effect_from_buff(buff):
    effect = darkest.IronCrownEffect(id=f"add_{buff.id}", chance=1)
    effect.buffs.append(buff.id)
    return effect


def create_manager_buff(type = "ads"):
    ade = darkest.IronCrownActorDataEffects()
    if type == "ads":
        ade["turn_start_apply_limit_effects"] =
manager_buff_effect_names_ads
        ade["turn_start_apply_limit"] = 2 # 2 ADS buffs per turn
    elif type == "ade":
        ade["turn_start_apply_limit_effects"] =
manager_buff_effect_names_ade
        ade["turn_start_apply_limit"] = 3 # 3 ADE buffs per turn
    return
darkest.IronCrownBuff(id=f"inf1a_manager_{type}{manager_buff_id}",
duration_type="infinite", actor_data_effects = ade)


if __name__ == "__main__":
    trim_data()
    for i in range(256):
        buff_ads = assemble_random_buff(i)
        darkest.buffs[buff_ads.id] = buff_ads
        buff_ads_effect = create_effect_from_buff(buff_ads)
        darkest.effects[buff_ads_effect.id] = buff_ads_effect
        manager_buff_effect_names_ads.append(buff_ads_effect.id)
    for j in range(512):
        buff_ade = assemble_random_buff(255 + j, "ade")
        darkest.buffs[buff_ade.id] = buff_ade
        buff_ade_effect = create_effect_from_buff(buff_ade)
        darkest.effects[buff_ade_effect.id] = buff_ade_effect
        manager_buff_effect_names_ade.append(buff_ade_effect.id)
    darkest.target.add_buff(create_manager_buff("ads"))
    darkest.target.add_buff(create_manager_buff("ade"))
```

C# Libraries and Harmony

Another alternative and proven method is allowing us to load in our own assemblies at startup. This will allow us to use library dlls to run custom code. Many Unity games (eg. Rimworld, Risk of Rain 2, Cities Skylines, Valheim, etc) use this as a cornerstone of their modding scenes. They also use a patch/code injection library known as Harmony (https://harmony.pardeike.net/articles/intro.html) to allow modders the ability to extend and change core game functionality. They don't need access to the game's source code, as long as their CIL is unobfuscated, we can decompile that and bootstrap our own code with relative ease. This is how Binarizer's Lib and Speedwagon both worked.

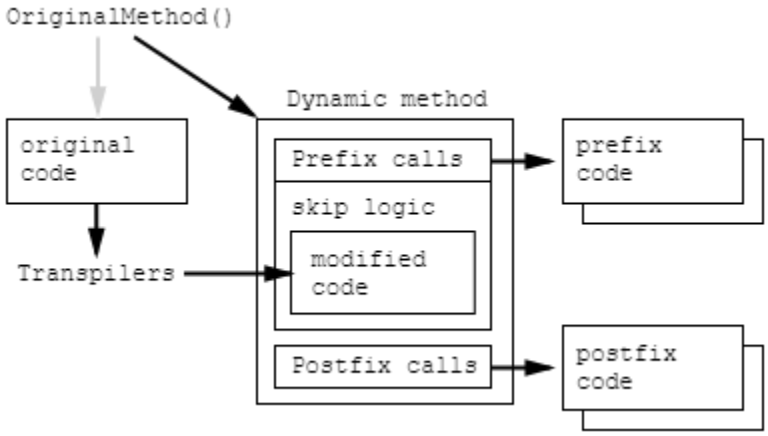


Diagram showing how Harmony allows us to modify existing game code without touching source files

Officially, we can already do this, we don't need an extra scripting language. Assembly sideloaders like Bepinex and Unity Doorstop exist to facilitate this.
But for first party support, we'd need the devs to write a mechanism to load our own assemblies at startup. This would be a lower effort requirement in comparison to accommodating a new scripting language/DSL or exposing a whole new set of scripting hooks (and documenting them) and/or introducing a new language competence to the project and to the modding scene.

Realistic example use case

ConditionTypes exist within the game's codebase (ironcrown.dll) as a hardcoded enum:

```
public class ConditionType : CustomEnum<ConditionType>
{
    private ConditionType(string name, ConditionCalculationType conditionCalculationType, bool isValidForInverse, bool isValidForBuff)
        : base(name)
    {
        this.m_ConditionCalculationType = conditionCalculationType;
        this.m_IsValidForInverse = isValidForInverse;
        this.m_IsValidForBuff = isValidForBuff;
    }

    [StaticInitializer]
    public static readonly ConditionType ALWAYS = new ConditionType("always", ConditionCalculationType.ALWAYS, false, true);

    public static readonly ConditionType TOKEN_AMOUNT = new ConditionType("token_amount", ConditionCalculationType.ACTOR, false, true);

    public static readonly ConditionType TOKEN_TAG_AMOUNT = new ConditionType("token_tag_amount", ConditionCalculationType.ACTOR, false,
      true);

    public static readonly ConditionType HEALTH_PERCENT = new ConditionType("health_percent", ConditionCalculationType.ACTOR, false, true);
```

This CustomEnum is used when condition validation happens (within GetRunValueForConditionType):

```
private static float GetRunValueForConditionType(ConditionType conditionType, bool conditionIsInverse, string conditionString)
{
    float num = 0f;
    if (conditionType == ConditionType.RUN_VALUE)
    {
        num = SingletonMonoBehaviour<RunBhv>.Instance.RunValues.GetValue(CustomEnum<RunValueType>.Cast(conditionString));
    }
    else if (conditionType == ConditionType.RUN_VALUE_PERCENT)
    {
        num = SingletonMonoBehaviour<RunBhv>.Instance.RunValues.GetPercentage(CustomEnum<RunValueType>.Cast(conditionString));
    }
    else if (conditionType == ConditionType.BOSS)
    {
        if (SingletonMonoBehaviour<RunBhv>.Instance.RunManager.Boss != null && SingletonMonoBehaviour<RunBhv>.Instance.RunManager.Boss.m_Id == conditionString)
        {
            num = 1f;
        }
    }
    else if (conditionType == ConditionType.BIOME)
    {
        string text = null;
        BiomeType activeBiome = RunBhv.ActiveBiome;
        text = ((activeBiome == null) ? TextBasedEditorPrefs.GetString(TextBasedEditorPrefsBaseType.CONDITION_TEST_BIOME) : activeBiome.ToString());
        if (text == conditionString)
        {
            num = 1f;
        }
    }
    else if (conditionType == ConditionType.BIOME_SUB_TYPE)
```

By injecting our own code we can alter how GetRunValueForConditionType happens and even add our own ConditionTypes that we define within our own dlls.
This would require no new infrastructure or alterations to this code from Red Hook, all being done by the modders themselves.

The Downsides

While incredibly powerful, this approach does come with downsides:

- By injecting code directly at runtime without developer supervision, it allows bad code to be written by any mod developer to go unchecked for users. This usually comes in the form of NullReferenceExceptions breaking something within the game, or hard crashes.
- Being able to run any code on a player's machine theoretically allows bad actors to do bad things on people's PCs.
    - Fortunately Valve takes their user generated content very seriously and has very strict checks in place to protect the upload of malware to the steam workshop. This of course gives no such protections to other file distribution platforms though.

- This would use the C# syntax and requires the modder in question to have decompiled the game's code. This is a functional barrier to entry for making new behaviors, though does not affect CSV only mods and their barrier to entry.
- Mods that add C# libraries will need to be recompiled against every major version and would not be evergreen, as the cost of maintenance is passed from the developers maintaining an API to modders having to make changes to their injected code to accommodate for changes within ironcrown.

Questions, Requests and Bugs: Steadfast Steward Update

Questions:

-
- What is the default outline width that the game uses? The default value in the outline shader provided seems to be too thin.
- How can we have our custom materials fade to black when the heroes do? (Example: fade to black after exiting a victory screen)

Requests:

- An example combat scene for testing. Currently we can only test how our custom models will look in game by uploading the mod and playing it. If we had an example scene in Unity, it would save a lot of time.
- ~~Can we have access to hero skill timelines? This would allow us to edit the "Action" pose of attack animations. We cannot currently edit the action pose without making a new timeline for that skill. We also don't seem to have any action or recovery anims available. This will also make it pretty easy to create new path skills.~~
- We'll need access to the DD2 FMOD library in order to use sounds in our mods. We were given the FMOD Plugin, but not the DD2 library.
- ~~Workshop tags for custom skins, weapon kits, and animations~~
- The ability to override a hero prefab without the player needing to change skins (Example: mods that change a hero's animation shouldn't require changing skins)
- Animator State behaviors or Animation events to set Animator variables.
- Allow using Weapon kits with Hero Skins.

Bugs:

- Changing a hero's palette while a custom weapon kit is applied will apply the palette's texture to the weapon kit.

- Outlines seem inconsistent. Custom weapon kits appear to have outlines in some scenes and lack outlines in others (whether or not the outline shader was applied to the object in Unity)
- When importing the models into Blender, they turn into a tiny blob. The models look correct when set to their rest pose, but any other pose breaks the model.
- Skins and Palettes for Abom don’t function in-game
- Constraints targeting a character rig's arms are out of place when executing in the inn. (Potentially an execution order issue?)